

Vitis AI Library User Guide

UG1354 (v1.1) March 23, 2020



Revision History

The following table shows the revision history for this document.

Section	Revision Summary
03/23/2020 Version 1.1	
Chapter 1: Introduction	Added "Vitis AI Library 1.1 Release notes"
Chapter 2: Installation	Added cloud operation content. Updated the method of setting up the host.
Chapter 7: Performance	Added the performance data of U50
12/02/2019 Version 1.0	
Entire document	Updated the content for Vitis™ AI v1.0. Removed support for the Ultra96 board.
08/13/2019 Version 2.0	
Entire document	Updated framework figure. Added "About this document" and "Release Notes" in Chapter 1. Updated Installation in Chapter 2. Added Programming Examples chapter. Added Application demos chapter. Added Resnet18, face landmark and ReID model. Updated Performance data for ZCU102, ZCU104, Ultra96. Removed the original Chapter 3: Installation. Removed the original Chapter 4: Cross-Compiling Removed the original Chapter 6: Libraries Advanced Application. Removed Roadline_deepphi Model.
05/31/2019 Version 1.2	
Chapter 3: Libraries and Samples	Added Inception_V4, YOLOV2, Roadline_deepphi model. Removed RefineDet_640x480 model.
Chapter 7: Performance	Updated Performance data for ZCU102, ZCU104, Ultra96.
05/24/2019 Version 1.1	
Entire document	Editorial updates.
04/29/2019 Version 1.0	
Initial release.	N/A

Table of Contents

Revision History.....	2
Chapter 1: Introduction.....	5
About this Document.....	5
Overview.....	6
Block Diagram.....	7
Features.....	8
Vitis AI Library 1.1 Release Notes.....	8
Vitis AI Library 1.0 Release Notes.....	11
Chapter 2: Installation.....	15
Downloading the Vitis AI Library.....	15
Setting Up the Host.....	15
Setting Up the Target.....	19
Running Vitis AI Library Examples.....	21
Support.....	24
Chapter 3: Libraries and Samples.....	25
Model Library.....	26
Model Samples.....	37
Chapter 4: Programming Examples.....	39
Developing With Vitis AI API_0.....	40
Developing with User Model and AI Library API_2.....	42
Customizing Pre-Processing.....	44
Using the Configuration File.....	45
Implementing User Post-Processing Code.....	49
How to Using the AI Library's Post-Processing Library.....	50
Chapter 5: Application Demos.....	52
Demo Overview.....	52
Demo Platform and Setup.....	52
Demo 1: Multi-Task Segmentation + Car Detection and Road Line Detection.....	54

Demo 2: Multi-Task Segmentation+Car Detection and Pose Detection.....	55
Chapter 6: Programming APIs.....	57
Chapter 7: Performance.....	58
ZCU102 Performance.....	58
ZCU104 Performance.....	60
U50 Performance.....	62
Appendix A: Additional Resources and Legal Notices.....	64
Xilinx Resources.....	64
Documentation Navigator and Design Hubs.....	64
Please Read: Important Legal Notices.....	65

Introduction

About this Document

Related Version

The following AI Library version is related to this document.

Table 1: Vitis AI Library Packet List

No	Packet Name	Version
1	<code>vitis_ai_library-r1.1-video.tar.gz</code>	r1.1
2	<code>vitis_ai_library-r1.1-image.tar.gz</code>	r1.1
3	<code>libvitis_ai_library-1.1.0-Linux-build46.deb (aarch64)</code>	r1.1
4	<code>vitis_ai_model_ZCU102-2019.2-r1.1.0.deb</code>	r1.1
5	<code>vitis_ai_model_ZCU104-2019.2-r1.1.0.deb</code>	r1.1
6	<code>libvitis_ai_library-1.1.0-Linux-build46.deb (X86_64)</code>	r1.1
7	<code>xilinx_model_zoo-1.1.0-Linux.deb</code>	r1.1

Intended Audience

The users of Vitis AI libraries are as follows:

- Users who want to use Xilinx's models to quickly build applications.
- Users who use their own models that are retrained by their own data under the Vitis AI library support network list.
- Users who have custom models, similar to the model supported by the Vitis AI libraries, and use the Vitis AI's post processing library.

Note: If you have custom models that are completely different from the model supported by the AI Library or has a special post-processing part, they can also use our samples and libraries implementation for reference.

Document Navigation

This document describes how to install, use, and develop with the AI Library.

- Chapter 1 is an introduction to the AI Library. This chapter provides a clear understanding of the AI Library in general, its framework, supported networks, supported hardware platforms and so on.
- Chapter 2 describes how to install the AI Library and run the example. The information in this chapter will help quickly set up the host and target environments, compile and execute the AI Library related examples.
- Chapter 3 describes, in detail, each model library supported by the AI Library. This chapter provides an understanding of the model libraries supported by the AI Library, the purpose of each library, how to test the library with images or videos, and how to test the performance of the library.
- Chapter 4 describes, in detail, how to develop applications with AI Library. This chapter provides an understanding of the following:
 - Development using Vitis API
 - Development using your models
 - Customizing pre-processing
 - Using the configuration file as pre-processing and post-processing parameters
 - Using the post-processing library in AI Library
 - Implementing your post-processing code
- Chapter 5 describes how to set up a test environment and run the application demos. There are two application demos provided with the Vitis AI Library.
- Chapter 6 describes how to find the programming APIs.
- Chapter 7 describes, in detail, the performance of the Vitis AI library on different boards.

Overview

The Vitis AI Library is a set of high-level libraries and APIs built for efficient AI inference with Deep-Learning Processor Unit (DPU). It is built based on the Vitis AI Runtime with unified APIs, and it fully supports XRT 2019.2.

The Vitis AI Library provides an easy-to-use and unified interface by encapsulating many efficient and high-quality neural networks. This simplifies the use of deep-learning neural networks, even for users without knowledge of deep-learning or FPGAs. The Vitis AI Library allows users to focus more on the development of their applications, rather than the underlying hardware.

For the intended audience for the AI Library, please refer to the [About this Document](#) section.

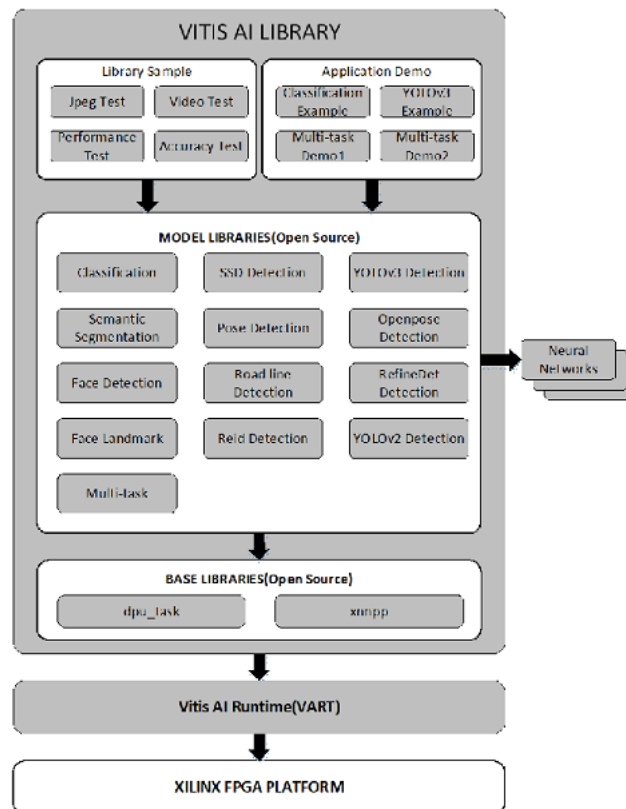
Block Diagram

The Vitis™ AI Library contains four parts: the base libraries, the model libraries, the library test samples, and the application demos.

The base libraries provide the operation interface with the DPU and the post-processing module of each model. *dpu_task* is the interface library for DPU operations. *xnnpp* is the post-processing library of each model, with build-in modules such as optimization and acceleration.

The model libraries implement most of the neural network deployment which are open source. They include common types of networks, such as classification, detection, segmentation, and so on. These libraries provide an easy-to-use and fast development method with a unified interface, which are applicable to the Xilinx models or custom models. The library test samples are used to quickly test and evaluate the model libraries. The application demos show users how to use AI Library to develop applications. The Vitis AI Library block diagram is shown in the following figure.

Figure 1: Vitis AI Library Block Diagram



Features

The Vitis AI Library features include:

- A full-stack application solution from top to bottom
- Optimized pre- and post-processing functions/libraries
- Open-source model libraries
- Unified operation interface with the DPU and the pre-processing and post-processing interface of the model
- Practical, application-based model libraries, pre-processing and post-processing libraries, and application examples

Vitis AI Library 1.1 Release Notes

This section contains information regarding the features and updates of the Vitis™ AI Library 1.1 release.

Key Features And Enhancements

This AI Library release includes the following key features and enhancements.

- **Support for the cloud:** Alveo U50 card is supported by this release.
- **New DPU support:** DPUV3e is supported which can be used for the cloud.
- **New Open Source Library:** The `xnnpp` library is open source in this release, which shows how to do the pre-processing and post-processing for the neural networks.
- **Model Library Update:** The new model library unifies the interface between the cloud and edge.

Changes

The installation mode of the host for the edge is changed and the original Docker installation mode is no longer used.

Compatibility

- Vitis AI Library 1.1 has been tested with the following images.
 - `xilinx-zcu102-dpu-v2019.2-v2.img`
 - `xilinx-zcu104-dpu-v2019.2-v2.img`

Model Support

The following models are supported by this version of the Vitis AI Library.

Table 2: Model Supported by the AI Library

No.	Neural Network	ZCU102/ZCU104	U50	Application
1	inception_resnet_v2_tf	Y	Y	Image Classification
2	inception_v1_tf	Y	Y	
3	inception_v3_tf	Y	Y	
4	inception_v4_2016_09_09_tf	Y	Y	
5	mobilenet_v1_0_25_128_tf	Y	N/A	
6	mobilenet_v1_0_5_160_tf	Y	N/A	
7	mobilenet_v1_1_0_224_tf	Y	N/A	
8	mobilenet_v2_1_0_224_tf	Y	N/A	
9	mobilenet_v2_1_4_224_tf	Y	N/A	
10	resnet_v1_101_tf	Y	Y	
11	resnet_v1_152_tf	Y	Y	
12	resnet_v1_50_tf	Y	Y	
13	vgg_16_tf	Y	Y	
14	vgg_19_tf	Y	Y	
15	ssd_mobilenet_v1_coco_tf	Y	N/A	Object Detection
16	ssd_mobilenet_v2_coco_tf	Y	N/A	
17	ssd_resnet_50_fpn_coco_tf	Y	Y	
18	yolov3_voc_tf	Y	Y	
19	mlperf_ssd_resnet34_tf	Y	N/A	
20	resnet50	Y	Y	Image Classification
21	resnet18	Y	Y	
22	inception_v1	Y	Y	
23	inception_v2	Y	Y	
24	inception_v3	Y	Y	
25	inception_v4	Y	Y	
26	mobilenet_v2	Y	N/A	
27	squeezenet	Y	Y	
28	ssd_pedestrain_pruned_0_97	Y	Y	ADAS Pedestrian Detection
29	ssd_traffic_pruned_0_9	Y	Y	Traffic Detection

Table 2: Model Supported by the AI Library (cont'd)

No.	Neural Network	ZCU102/ZCU104	U50	Application
30	ssd_adas_pruned_0_95	Y	Y	ADAS Vehicle Detection
31	ssd_mobilenet_v2	Y	N/A	Object Detection
32	refinedet_pruned_0_8	Y	Y	
33	refinedet_pruned_0_92	Y	Y	
34	refinedet_pruned_0_96	Y	Y	
35	vpgnet_pruned_0_99	Y	Y	ADAS Lane Detection
36	fpn	Y	Y	ADAS Segmentation
37	sp_net	Y	Y	Pose Estimation
38	openpose_pruned_0_3	Y	Y	
39	densebox_320_320	Y	Y	Face Detection
40	densebox_640_360	Y	Y	
41	face_landmark	Y	Y	Face Detection and Recognition
42	reid	Y	Y	Object tracking
43	multi_task	Y	Y	ADAS
44	yolov3_adas_pruned_0_9	Y	Y	Object Detection
45	yolov3_voc	Y	Y	
46	yolov3_bdd	Y	Y	
47	yolov2_voc	Y	Y	
48	yolov2_voc_pruned_0_66	Y	Y	
49	yolov2_voc_pruned_0_71	Y	Y	
50	yolov2_voc_pruned_0_77	Y	Y	

Notes:

- No1-No19 neural network models are trained based on the Tensorflow framework.
- No20-No50 neural network models are trained based on the Caffe framework.

Device Support

The following platforms and EVBs are supported by the Vitis AI Library1.1.

Table 3: Edge Device Support

Platform	EVb	Version
Zynq UltraScale+ MPSoC ZU9EG	Xilinx ZCU102	V1.1
Zynq® UltraScale+™ MPSoC ZU7EV	Xilinx ZCU104	V1.0

Table 4: Cloud Device Support

Accelerator Cards
Xilinx Alveo U50

Limitations

Some neural networks with mobilenet as the backbone are not supported on U50.

Vitis AI Library 1.0 Release Notes

This section contains information regarding the features and updates of the Vitis™ AI Library 1.0 release. This release is the successor of last Xilinx® AI SDK v2.0 release.

The Vitis AI Library is a set of high-level libraries and APIs built for efficient AI inference with Deep-Learning Processor Unit (DPU). It provides an easy-to-use and unified interface by encapsulating many efficient and high-quality neural networks.

Key Features And Enhancements

This Vitis AI Library release includes the following key features and enhancements.

- **Support for new Vitis AI Runtime:** The Vitis AI Library is updated to be based on the new Vitis AI Runtime with unified APIs. It also fully supports XRT 2019.2.
- **New DPU support:** Besides DPUv2 for edge devices, new AI Library will support new cloud based DPU IPs using same codes (runtime and models for cloud DPU will not be included in this release).
- **New Tensorflow model support:** There are up to 21 tensorflow models supported, which are from official Tensorflow repository. The pre-compiled models for edge devices are included, while original models are released by updated Model Zoo.
- **New Libraries and Demos:** There are two new libraries `libdpmultitask` and `libdptfssd` which supports multi-task models and SSD models from official tensor repository.

There is an updated classification demo that shows how to uses unified APIs in Vitis AI runtime.

- **New Open Source Library:** The `libdpbase` library is open source in this release, which shows how to use unified APIs in Vitis AI runtime to construct high-level libraries.
- **New Installation Method:** The host side environment adopts docker image installation, which simplifies and unifies the installation process.

Compatibility

- Vitis AI Library 1.0 has been tested with the following images.
 - xilinx-zcu102-dpu-v2019.2.img
 - xilinx-zcu104-dpu-v2019.2.img
- For existing Xilinx AI SDK v2.0 users, the library interface remains consistent and the application can be directly ported to the new Vitis AI Library.

Model Support

The following models are supported by this version of the Vitis AI Library.

Table 5: Model Supported by the AI Library

No.	Neural Network	Application
1	inception_resnet_v2_tf	Image Classification
2	inception_v1_tf	
3	inception_v3_tf	
4	inception_v4_2016_09_09_tf	
5	mobilenet_v1_0_25_128_tf	
6	mobilenet_v1_0_5_160_tf	
7	mobilenet_v1_1_0_224_tf	
8	mobilenet_v2_1_0_224_tf	
9	mobilenet_v2_1_4_224_tf	
10	resnet_v1_101_tf	
11	resnet_v1_152_tf	
12	resnet_v1_50_tf	
13	vgg_16_tf	
14	vgg_19_tf	
15	ssd_mobilenet_v1_coco_tf	Object Detection
16	ssd_mobilenet_v2_coco_tf	
17	ssd_resnet_50_fpn_coco_tf	
18	yolov3_voc_tf	
19	mlperf_ssd_resnet34_tf	
20	resnet50	Image Classification
21	resnet18	
22	inception_v1	
23	inception_v2	
24	inception_v3	
25	inception_v4	
26	mobilenet_v2	
27	squeezenet	

Table 5: Model Supported by the AI Library (cont'd)

No.	Neural Network	Application
28	ssd_pedestrian_pruned_0_97	ADAS Pedestrian Detection
29	ssd_traffic_pruned_0_9	Traffic Detection
30	ssd_adas_pruned_0_95	ADAS Vehicle Detection
31	ssd_mobilenet_v2	Object Detection
32	refinedet_pruned_0_8	
33	refinedet_pruned_0_92	
34	refinedet_pruned_0_96	
35	vpgnet_pruned_0_99	ADAS Lane Detection
36	fpn	ADAS Segmentation
37	sp_net	Pose Estimation
38	openpose_pruned_0_3	
39	densebox_320_320	Face Detection
40	densebox_640_360	
41	face_landmark	Face Detection and Recognition
42	reid	Object tracking
43	multi_task	ADAS
44	yolov3_adas_pruned_0_9	Object Detection
45	yolov3_voc	
46	yolov3_bdd	
47	yolov2_voc	
48	yolov2_voc_pruned_0_66	
49	yolov2_voc_pruned_0_71	
50	yolov2_voc_pruned_0_77	

Notes:

- No1-No19 neural network models are trained based on the Tensorflow framework.
- No20-No50 neural network models are trained based on the Caffe framework.

Device Support

The following platforms and EVBs are supported by the Vitis AI Library1.0.

Table 6: Device Support

Platform	EVB	Version
Zynq UltraScale+ MPSoC ZU9EG	Xilinx ZCU102	V1.1
Zynq® UltraScale+™ MPSoC ZU7EV	Xilinx ZCU104	V1.0

Limitations

Because of the complicated configuration for SSD models from the official Tensorflow repository, there is a new `libdptfssd` library that is different from the original `libdpssd` library for caffe models. These two libraries may be merged in future releases.

Deprecated Features

The following features are deprecated in Vitis AI Library 1.0.

- Removed demos.

The squeezenet and SSD demos have been removed. Because we highly encourage customers to use high-level APIs from AI Library for applications and solutions, we only provide one classification demo in this release to show how to use low-level unified APIs in Vitis AI runtime.

- Removed pre-compiled models.

We removed six Tensorflow models in this release but provided in the previous Xilinx AI v2.0 release, to keep sync with updated Model Zoo. Models that are removed can be replaced by similar models in updated Model Zoo which come from Tensorflow slim models. The models are:

- . resnet_50_tf
- . inception_v1_tf
- . resnet_18_tf
- . mobilenet_v1_tf
- . mobilenet_v2_tf
- . ssd_voc_tf

Installation

Downloading the Vitis AI Library

The Vitis™ AI Library package can be freely downloaded from the public github <https://github.com/Xilinx/Vitis-AI>.

Xilinx recommends that you use Vitis AI Library-supported evaluation board or Alveo card to allow you to become familiar with the product. See the <https://www.xilinx.com/products/design-tools/ai-inference/ai-developer-hub.html#edge> for more details about the Vitis AI Library-supported evaluation boards. And refer to <https://www.xilinx.com/products/boards-and-kits/alveo.html> for more details about Alveo cards.

The evaluation boards supported for this release are:

- Xilinx ZCU102
- Xilinx ZCU104

The Alveo cards supported for this release are:

- Xilinx Alveo U50

Setting Up the Host

For Edge

Use the following steps to set up the host for edge device development.

1. Download the `sdk.sh` from <https://github.com/Xilinx/Vitis-AI>.
2. Install the cross-compilation system environment.

```
$. /sdk.sh
```

3. Follow the prompts to install.

Note:

The `~/petalinux_sdk` path is recommended for the installation. Regardless of the path you choose for the installation, make sure the path has read-write permissions. Here we install it under `~/petalinux_sdk`.

4. When the installation is complete, follow the prompts and execute the following command.

```
$. ~/petalinux_sdk/environment-setup-aarch64-xilinx-linux
```

If you close the current terminal, you need to re-execute the above instructions in the new terminal interface.

5. Download the `vitis_ai_2019.2-r1.1.0.tar.gz` from <https://github.com/Xilinx/Vitis-AI> and install it to the petalinux system.

```
$tar -xzvf vitis_ai_2019.2-r1.1.0.tar.gz -C ~/petalinux_sdk/sysroots/aarch64-xilinx-linux
```

6. Update the glog to v0.4.0.

1. Download the glog package and untar it.

```
$cd ~
$curl -Lo glog-v0.4.0.tar.gz https://github.com/google/glog/archive/v0.4.0.tar.gz
$tar -zxvf glog-v0.4.0.tar.gz
$cd glog-0.4.0
```

2. Build it and install it to the PetaLinux system.

```
$mkdir build_for_petalinux
$cd build_for_petalinux
$unset LD_LIBRARY_PATH; source ~/petalinux_sdk/environment-setup-aarch64-xilinx-linux
$make -DCPACK_GENERATOR=TGZ -DBUILD_SHARED_LIBS=on -DCMAKE_INSTALL_PREFIX=$OECORE_TARGET_SYSROOT/usr ..
$make && make install
$make package
```

7. Clone the Vitis AI repository:

```
$cd ~
$git clone https://github.com/xilinx/vitis-ai
```

8. Cross compile the demo in the AI Library, using yolov3 as example.

```
$cd ~/Vitis-AI/Vitis-AI-Library/overview/demo/yolov3
$bash -x build.sh
```

If you do not want to print information during compilation, execute the following command.

```
$bash build.sh
```

If the compilation process does not report any error and the executable file `demo_yolov3` is generated, the host environment is installed correctly.

9. To compile the library sample in the AI Library, take `classification` for example, execute the following command.

```
$cd ~/Vitis-AI/Vitis-AI-Library/overview/samples/classification
$bash -x build.sh
```

The executable program is now produced.

10. To modify the library source code, view and modify them under `~/Vitis-AI/Vitis-AI-Library`.

Before compiling the AI libraries, please confirm the compiled output path. The default output path is : `$HOME/build`.

If you want to change the default output path, please modify the `build_dir_default` in `cmake.sh`. Such as, change from

```
build_dir_default=$HOME/build/build.${target_info}/${project_name}
to build_dir_default=/workspace/build/build.${target_info}/${project_name}
```

Note:

If you want to modify the `build_dir_default`, it's suggested to modify `$HOME` only.

Execute the following command to build the libraries all at once.

```
$cd ~/Vitis-AI/Vitis-AI-Library
$./cmake.sh --clean --cmake-options='-DCMAKE_NO_SYSTEM_FROM_IMPORTED=on'
```

After compiling, you can find the generated AI libraries under `build_dir_default`. If you want to change the compilation rules, check and change the `cmake.sh` in the library's directory.

Note: All the source code, samples, demos, and head files can be found in `~/Vitis-AI/Vitis-AI-Library`.

For Cloud

The host side development environment for the cloud is setting up by docker image.

1. Download the `vitis-ai-cpu` image from <https://github.com/Xilinx/Vitis-AI>.
2. Set up the docker runtime system according to the docker installation document.

```
$/docker_run.sh -X xilinx/vitis-ai-cpu:<x.y.z>
```

Note: A `workspace` folder will be created by the docker runtime system. And it will be mounted in `/workspace` of the docker runtime system.

3. Place the program, data and other files to be developed in the `workspace` folder. After the docker system starts, you will find them in the `/workspace` of the docker system.

Do not put the files in any other path of the docker system. They will be lost after you exit the docker system.

4. Download the `libvitis_ai_library-1.1.0-Linux-build46.deb` from <https://github.com/Xilinx/Vitis-AI> and install it to the docker system.

```
$sudo dpkg -i libvitis_ai_library-1.1.0-Linux-build46.deb
```

5. Download the U50 model package from <https://github.com/Xilinx/Vitis-AI> and Install it.

```
$sudo dpkg -i xilinx_model_zoo-1.1.0-Linux.deb
```

6. Download the xclbin files from <https://github.com/Xilinx/Vitis-AI> and Install them.

```
$sudo cp dpu.xclbin hbm_address_assignment.txt /usr/lib
```

7. Using Git, clone the corresponding AI Library from <https://github.com/Xilinx/Vitis-AI>.

```
$cd /workspace
$git clone https://github.com/Xilinx/Vitis-AI.git
```

8. Export the library path.

```
$export INTERNAL_BUILD=1
$export LD_LIBRARY_PATH=/opt/xilinx/xrt/lib:/usr/lib:/usr/lib/x86_64-
linux-gnu:/opt/vitis_ai/conda/envs/vitis_ai_tensorflow/lib/
```

9. Compile the demo in the AI Library, using yolov3 as example.

```
$cd /workspace/Vitis-AI/Vitis-AI-Library/overview/demo/yolov3
$bash -x build.sh
```

If you do not want to print information during compilation, execute the following command.

```
$bash build.sh
```

If the compilation process does not report any error and the executable file `demo_yolov3` is generated, the host environment is installed correctly.

10. To compile the library sample in the AI Library, take `classification` for example, execute the following command.

```
$cd /workspace/Vitis-AI/Vitis-AI-Library/overview/samples/classification
$bash -x build.sh
```

The executable program is now produced.

11. To modify the library source code, view and modify them under `/workspace/Vitis-AI/Vitis-AI-Library`.

Before compiling the AI libraries, please confirm the compiled output path. The default output path is : `$HOME/build`.

If you want to change the default output path, please modify the `build_dir_default` in `cmake.sh`. Such as, change from

```
build_dir_default=$HOME/build/build.${target_info}/${project_name}
to build_dir_default=/workspace/build/build.${target_info}/${
{project_name}}
```

Note: If you want to modify the `build_dir_default`, it is suggested to modify `$HOME` only.

Execute the following command to build the libraries all at once.

```
$cd /workspace/Vitis-AI/Vitis-AI-Library
$./cmake.sh --clean --cmake-options='-DCMAKE_NO_SYSTEM_FROM_IMPORTED=on'
```

After compiling, you can find the generated AI libraries under `build_dir_default`. If you want to change the compilation rules, check and change the `cmake.sh` in the library's directory.

AI Library File Locations

The following table shows the AI Library file location after the installation is complete.

Table 7: AI Library File Location List

Files	Location
Sour code of the libraries	/workspace/Vitis-AI/Vitis-AI-Library
Samples	/workspace/Vitis-AI/Vitis-AI-Library/overview/samples
Demos	/workspace/Vitis-AI/Vitis-AI-Library/overview/demo
Test	/workspace/Vitis-AI/Vitis-AI-Library/[model]/test

Notes:

The following symbols/abbreviations are used.

- `/workspace/` is the path to extract the AI Library compressed package in the docker system.
- "Samples" is used for rapid application construction and evaluation, and it is for users.
- "Demos" provides more practical examples for user development, and it is for users.
- "Test" is a test example for each model library which is for library developers.

Setting Up the Target

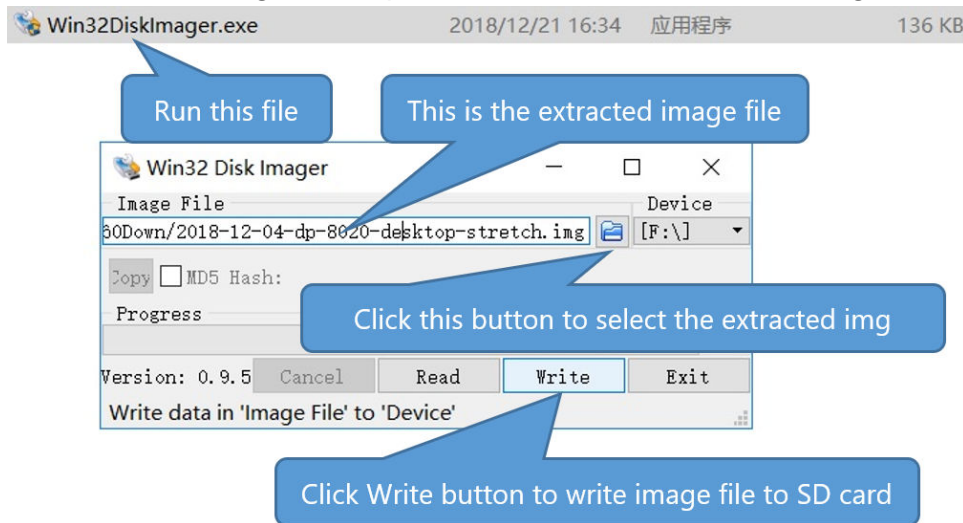
To set up the target, you should follow three steps. The first step is to install the board image, the second step is to install the AI model packet, and the third step is to install the AI Library packet.

Note: The version of the board image should be 2019.2 or above.

Step 1: Installing a Board Image

1. Download the SD card system image files from <https://github.com/Xilinx/Vitis-AI> (such as ZCU102 or ZCU104).

2. Use Win32DiskImager (free opensource software) to burn the image file onto the SD card.



3. Insert the SD card with the image into the destination board.
4. Plug in the power and boot the board using the serial port to operate on the system.
5. Set up the IP information of the board using the serial port.

You can now operate on the board using SSH.

Step 2: Installing AI Model Package

1. Download the `vitis_ai_model_ZCU102_2019.2-r1.1.0.deb` packet.
2. Copy the downloaded file to the board using `scp` with the following command.

```
$scp vitis_ai_model_ZCU102_2019.2-r1.1.0.deb root@IP_OF_BOARD:~/
```

Note: The deb package can be taken as a normal archive, and you can extract the contents on the host side if you only need part of the models. The operation command is as follows.

```
$mkdir extract
$dpkg -X vitis_ai_model_ZCU102_2019.2-r1.1.0.deb extract
```

3. Log in to the board (usong ssh or serial port) and install the model package.
4. Run the following command.

```
#dpkg -i vitis_ai_model_ZCU102_2019.2-r1.1.0.deb
```

After the installation is complete, the model files are stored in `/usr/share/vitis_ai_library/models` on the target side.

Step 3: Installing AI Library Package

1. Download the `libvitis_ai_library-1.1.0-Linux-build46.deb` (aarch64) packet.

- Copy the downloaded file to the board using `scp` with the following command.

```
$scp libvitis_ai_library-1.1.0-Linux-build46.deb root@IP_OF_BOARD:~/
```

Note: You can take the deb package as a normal archive, and extract the contents on the host side, if you only need some of the libraries. Only model libraries can be separated dependently, while the others are common libraries. The operation command is as follows.

```
$mkdir extract
$dpkg -X libvitis_ai_library-1.1.0-Linux-build46.deb extract
```

- Copy the `glog-0.4.0-Linux.tar.gz` from host to board with the following command.

```
$cd ~/glog-0.4.0/build_for_petalinux
$scp glog-0.4.0-Linux.tar.gz root@IP_OF_BOARD:~/
```

- Log in to the board using `ssh`.

You can also use the serial port to login.

- Update the `glog` to v0.4.0.

```
#tar -xzvf glog-0.4.0-Linux.tar.gz --strip-components=1 -C /usr
```

- For ZCU104, enable the power patch to make sure all models run normally for long-periods of time under heavy load.

```
#irps5401
```

- For ZCU102, reduce the DPU frequency to 93% to make sure all models run normally for long-periods of time under heavy load.

```
#dpu_clk 93
```

- Install the Vitis AI Library.

Execute the following command.

```
#dpkg -i libvitis_ai_library-1.1.0-Linux-build46.deb
```

After the installation is complete, the directories are as follows.

- Library files are stored in `/usr/lib`
- The header files are stored in `/usr/include/vitis/ai`

Running Vitis AI Library Examples

Before running the Vitis™ AI Library examples on Edge or on Cloud, download the `vitis_ai_library_r1.1_image.tar.gz` and `vitis_ai_library_r1.1_video.tar.gz` from <https://github.com/Xilinx/Vitis-AI>. The images or videos used in the following example can be found in both packages.

For Edge

There are two ways to compile a program. One is to cross-compile the program through the host and the other is to compile the program directly on the target board. Both methods have advantages and disadvantages. In this section, we compile and run the examples directly on the target machine.

1. Copy the sample and demo from host to the target using `scp` with the following command.

```
$scp -r ~/Vitis-AI/Vitis-AI-Library/overview root@IP-OF-BOARD:~/
```

2. Enter the extracted directory of example in target board and then compile each of the examples.

```
#cd ~/overview/samples/facedetect
```

3. Run the example.

```
#./test_jpeg_facedetect densebox_640_360 sample_facedetect.jpg
```

If the above executable program does not exist, run the following command to compile and generate the corresponding executable program.

```
#bash -x build.sh
```

4. View the running results.

There are two ways to view the results. One is to view the results by printing information, while the other is to view images by downloading the `sample_facedetect_result.jpg` image as shown in the following image.

Figure 2: Face Detection Example



- To run the video example, run the following command:

```
#./test_video_facedetect densebox_640_360 video_input.mp4 -t 8
Video_input.mp4: The video file's name for input.The user needs to
prepare the videofile.
-t: <num_of_threads>
```

- To test the program with a USB camera as input, run the following command:

```
#./test_video_facedetect densebox_640_360 0 -t 8
0: The first USB camera device node. If you have multiple
USB camera, the value might be 1,2,3 etc.
-t: <num_of_threads>
```



IMPORTANT! Enable X11 forwarding with the following command (suppose in this example that the host machine IP address is 192.168.0.10) when logging in to the board using an SSH terminal because all the video examples require a Linux windows system to work properly.

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

- To test the performance of model, run the following command:

```
#./test_performance_facedetect densebox_640_360
test_performance_facedetect.list -t 8 -s 60

-t: <num_of_threads>

-s: <num_of_seconds>
```

For more parameter information, enter `-h` for viewing. The following image shows the result of performance testing in 8 threads.

Figure 3: Face Detection Performance Test Result

```
root@xilinx-zcu102-2019_2:/usr/share/vitis_ai_library/samples/facedetect# ./test_performance_facedetect densebox_640_360
test_performance_facedetect.list -t 8 -s 60
WARNING: Logging before InitGoogleLogging() is written to STDERR
I1121 00:14:21.785212 4323 benchmark.hpp:195] writing report to <STDOUT>
I1121 00:14:21.803946 4323 benchmark.hpp:215] waiting for 0/60 seconds, 8 threads running
I1121 00:14:31.804129 4323 benchmark.hpp:215] waiting for 10/60 seconds, 8 threads running
I1121 00:14:41.804368 4323 benchmark.hpp:215] waiting for 20/60 seconds, 8 threads running
I1121 00:14:51.804600 4323 benchmark.hpp:215] waiting for 30/60 seconds, 8 threads running
I1121 00:15:01.804787 4323 benchmark.hpp:215] waiting for 40/60 seconds, 8 threads running
I1121 00:15:11.804975 4323 benchmark.hpp:215] waiting for 50/60 seconds, 8 threads running
I1121 00:15:21.805235 4323 benchmark.hpp:224] waiting for threads terminated
FPS=439.033
E2E_MEAN=18261.7
DPU_MEAN=8038.61
```

- To check the version of Vitis AI Library, run the following command:

```
#vitis_ai
```

- To run the demo, refer to [Chapter 5: Application Demos](#).

For Cloud

- Load and Run Docker Container.

```
$. /docker_run.sh -X xilinx/vitis-ai-cpu:<x.y.z>
```


2. Enter the directory of sample and then compile it. Take `facedetect` as an example.

```
$cd /workspace/vitis-ai/vitis_ai_library/overview/samples/facedetect
$bash -x build.sh
```

3. Run the sample.

```
./test_jpeg_facedetect densebox_640_360 sample_facedetect.jpg
```

4. If you want to run the program in batch mode, which means that the DPU processes multiple images at once to prompt for processing performance, you have to compile the entire Vitis AI Library according to "Setting Up the Host" section. Then the batch program will be generated under `build_dir_default`. Enter `build_dir_default`, take `facedetect` as an example, execute the following command.

```
./test_facedetect_batch densebox_640_360 <img1_url> [<img2_url> ...]
```

5. To run the video example, run the following command:

```
./test_video_facedetect densebox_640_360 video_input.mp4 -t 8
Video_input.mp4: The video file's name for input. The user needs to
prepare the videofile.
-t: <num_of_threads>
```

6. To test the performance of model, run the following command:

```
#./test_performance_facedetect densebox_640_360
test_performance_facedetect.list -t 8 -s 60
-t: <num_of_threads>
-s: <num_of_seconds>
```

For more parameter information, enter `-h` for viewing.

Note that the performance test program is automatically run in batch mode.

Support

You can visit the Vitis AI Library community forum on the Xilinx website <https://forums.xilinx.com/t5/Machine-Learning/bd-p/DeepHi> for topic discussions, knowledge sharing, FAQs, and requests for technical support.

Libraries and Samples

The Vitis™ AI Library contains the following types of neural network libraries based on Caffe framework:

- Classification
- Face detection
- SSD detection
- Pose detection
- Semantic segmentation
- Road line detection
- YOLOV3 detection
- YOLOV2 detection
- Openpose detection
- RefineDet detection
- ReID detection
- Multitask

Also, the Vitis AI Library contains the following types of neural network libraries based on Tensorflow framework:

- Classification
- SSD detection
- YOLOv3 detection

The related libraries are open source and can be modified as needed. The open source codes are available on the github. You can find them in <https://github.com/Xilinx/Vitis-AI>.

The Vitis AI Library provides image test samples and video test samples for all the above networks. In addition, the kit provides the corresponding performance test program. For video based testing, we recommend to use raw video for evaluation. Because decoding by software libraries on Arm® CPU may have inconsistent decoding time, which may affect the accuracy of evaluation.

Note: For edge, all the sample programs can only run on the target side, but all the sample programs can be cross compiled on the host side or compiled on the target side.

Model Library

After the model packet is installed on the target, all the models are stored under `/usr/share/vitis_ai_library/models/`. Each model is stored in a separate folder, which is composed of the following files by default.

- `[model_name].elf`
- `[model_name].prototxt`
- `meta.json`

Take the "inception_v1" model as an example. "inception_v1.elf" is the model data. "inception_v1.prototxt" is the parameter of the model. `meta.json` is the configuration file of the model. The application will get the model info through this configuration file. The following table is detail description of `meta.json`.

Table 8: The content of the "meta.json"

Element	Instruction
target	The type of DPU, such as DPUv2 and DPUv3e
lib	The driver of DPU, such as "libvart-dpu-runner.so"
filename	The name of model file
kernel	The kernel name of the model
config_file	The parameter file name of the model

Note:

The `meta.json` file should be under the same directory with the model file and the name of the model directory should be the same with the model name.

Model Type

Classification

This library is used to classify images. Such neural networks are trained on ImageNet for ILSVRC and they can identify the objects from its 1000 classification. The AI Library r1.1 integrated Resnet18, Resnet50, Inception_v1, Inception_v2, Inception_v3, Inception_v4, Vgg, mobilenet_v1, mobilenet_v2 and Squeezenet into our library. The input is a picture with an object and the output is the top-K most probable category.

Figure 4: Classification Example



The following table shows the classification model supported by the AI Library.

Table 9: The Classification Model List

No	Model Name	Framework
1	inception_resnet_v2_tf	Tensorflow
2	inception_v1_tf	
3	inception_v3_tf	
4	inception_v4_2016_09_09_tf	
5	mobilenet_v1_0_25_128_tf	
6	mobilenet_v1_0_5_160_tf	
7	mobilenet_v1_1_0_224_tf	
8	mobilenet_v2_1_0_224_tf	
9	mobilenet_v2_1_4_224_tf	
10	resnet_v1_101_tf	
11	resnet_v1_152_tf	
12	resnet_v1_50_tf	
13	vgg_16_tf	
14	vgg_19_tf	

Table 9: The Classification Model List (cont'd)

No	Model Name	Framework
15	resnet50	Caffe
16	resnet18	
17	inception_v1	
18	inception_v2	
19	inception_v3	
20	inception_v4	
21	mobilenet_v2	
22	squeezenet	

Face Detection

This library uses DenseBox neuron network to detect human face. Input is a picture with some faces you would like to detect. Output is a vector of the result structure containing each box's information.

The following image show the result of face detection.

Figure 5: Face Detection Example



The following table shows the face detection model supported by the AI Library.

Table 10: The face detection model list

No	Model Name	Framework
1	densebox_320_320	Caffe
2	densebox_640_360	

Face Landmark Detection

The Face Landmark Network is used to detect five key points of a face. The five points include the left eye, the right eye, nose, left lip of mouth, right lip of mouth. This network is used to correct face direction before face feature extraction. The input image should be a face which is detected by the face detection network. The outputs of the network are 5 key points. The 5 key points are normalized. The following image show the result of face detection.

Figure 6: Face Landmark Detection Example



The following table shows the face landmark model supported by the AI Library.

Table 11: The face landmark model list

No	Model Name	Framework
1	face_landmark	Caffe

SSD Detection

This library is in common use to SSD neuron network. SSD is a neural network which is used to detect objects. Input is a picture with some objects you'd like to detect. Output is a vector of the result structure containing each box's information. The following image shows the result of SSD detection.

Figure 7: SSD Detection Example



The following table shows the SSD detection model supported by the AI Library.

Table 12: The SSD Model List

No	Model Name	Framework
1	ssd_mobilenet_v1_coco_tf	Tensorflow
2	ssd_mobilenet_v2_coco_tf	
3	ssd_resnet_50_fpn_coco_tf	
4	mlperf_ssd_resnet34_tf	
5	ssd_pedestrain_pruned_0_97	Caffe
6	ssd_traffic_pruned_0_9	
7	ssd_adas_pruned_0_95	
8	ssd_mobilenet_v2	

Pose Detection

This library is used to detect posture of the human body. This library includes a neural network which can mark 14 key points of the human body (you can use our SSD detection library). The input is a picture that is detected by the pedestrian detection neural network. The output is a structure containing coordinates of each point .

The following image shows the result of pose detection.

Figure 8: Pose Detection Example



The following table shows the pose detection model supported by the AI Library.

Table 13: The pose detection model list

No	Model Name	Framework
1	sp_net	Caffe

Note: If the input image is arbitrary and the user does not know the exact location of the person, we must perform the SSD detection first. See the `test_jpeg_posedetect_with_ssd.cpp` file. If the input picture is the picture of the person who has been cut out, you can only perform pose detection. See the `test_jpeg_posedetect.cpp` file.

Semantic Segmentation

The semantic segmentation of image is to assign a semantic category to each pixel in the input image, so as to obtain the pixelated intensive classification. Libsegmentation is a segmentation lib which can be used in ADAS field. It offers simple interfaces for developer to deploy segmentation task on Xilinx FPGA.

The following is an example of semantic segmentation, where the goal is to predict class labels for each pixel in the image.

Figure 9: Semantic Segmentation Example



The following table shows the semantic segmentation model supported by the AI Library.

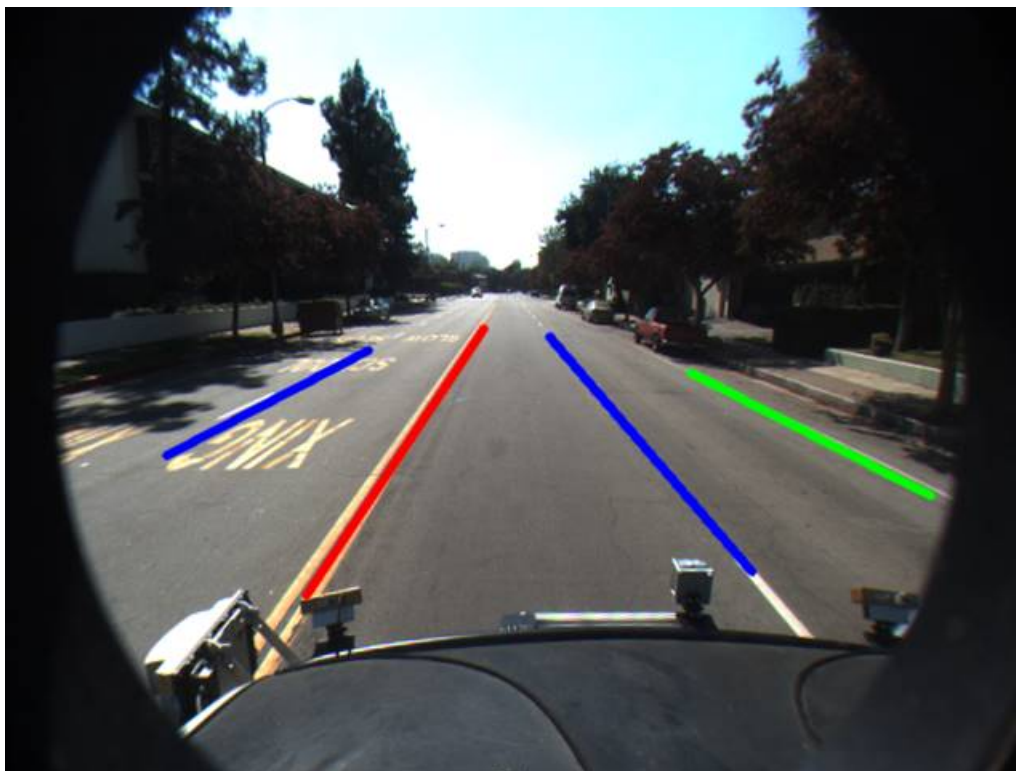
Table 14: The semantic segmentation model list

No	Model Name	Framework
1	fpn	Caffe

Road Line Detection

The library is used to draw lane lines in the adas library and each lane line is represented by a number type representing the category and a vector<Point> used to draw the lane line. In the test code, color map is used. Different types of lane lines are represented by different colors. The point is stored in the container vector, and the polygon interface cv::polyline() of OpenCv is used to draw the lane line. The following image show the result of road line detection.

Figure 10: Road Line Detection Example



The following table shows the road line detection model supported by the AI Library.

Table 15: The road line detection model list

No	Model Name	Framework
1	vpgnet_pruned_0_99	Caffe

Note: The input of the image is fixed at 480x640 and images of other sizes need to be resized.

YOLOV3 Detection

This lib is in common use to YOLO neuron network. YOLO is a neural network which is used to detect objects. Now its version is v3. Input is a picture with one or more objects. Output is a vector of the result struct which is composed of the detected information. The following image shows the result of YOLOv3 detection.

Figure 11: YOLOv3 Detection Example



The following table shows the YOLOv3 detection model supported by the AI Library.

Table 16: The YOLOv3 detection model list

No	Model Name	Framework
1	yolov3_voc_tf	Tensorflow
2	yolov3_adas_pruned_0_9	Caffe
3	yolov3_voc	
4	yolov3_bdd	

YOLOV2 Detection

YOLOV2 does the same thing as YOLOV3, which is an upgraded version of YOLOV2. The following table shows the YOLOv2 detection model supported by the AI Library.

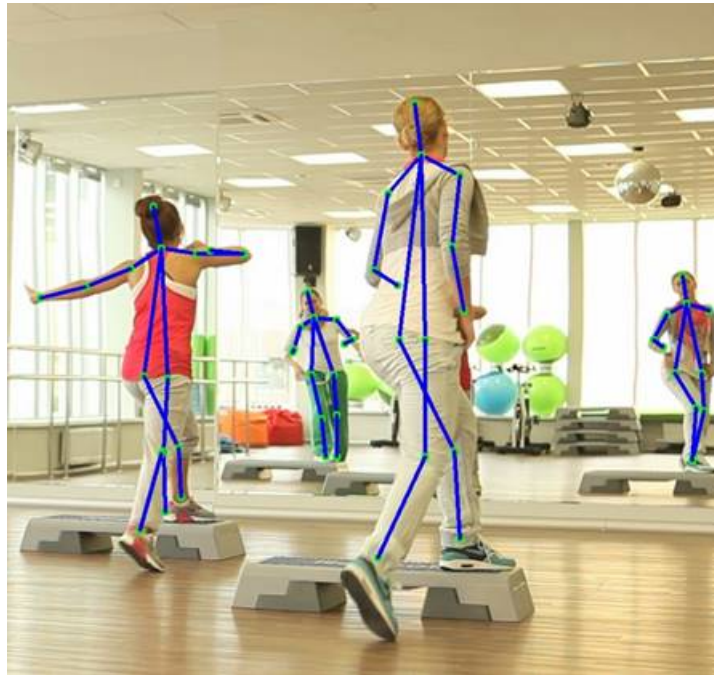
Table 17: The YOLOv2 detection model list

No	Model Name	Framework
1	yolov2_voc	Caffe
2	yolov2_voc_pruned_0_66	
3	yolov2_voc_pruned_0_71	
4	yolov2_voc_pruned_0_77	

Openpose Detection

The library is used to draw the person's posture. It is represented by the line between the point and the point, with points stored as pairs. Every pair represents a connection and the result is the set of pairs, stored as vectors. The following image show the result of openpose detection.

Figure 12: Openpose Detection Example



The following table shows the Openpose detection model supported by the AI Library.

Table 18: The Openpose detection model list

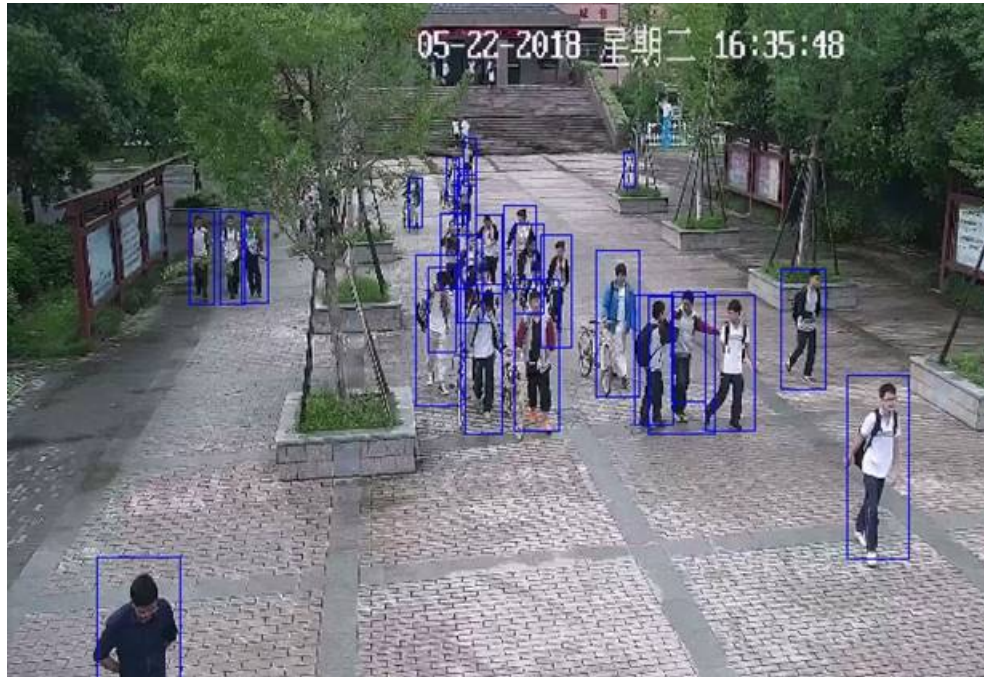
No	Model Name	Framework
1	openpose_pruned_0_3	Caffe

RefineDet Detection

This library is commonly used to RefineDet neuron network. RefineDet is a neural network that is used to detect human bodies. The input is a picture with some individuals that you would like to detect. The output is a vector of the result structure that contain each box's information.

The following image shows the result of RefineDet detection:

Figure 13: RefineDet Detection Example



The following table shows the RefineDet detection model supported by the AI Library.

Table 19: The RefineDet detection model list

No	Model Name	Framework
1	refinedet_pruned_0_8	Caffe
2	refinedet_pruned_0_92	
3	refinedet_pruned_0_96	

ReID Detection

The task of person re-identification is to identify a person of interest at another time or place. This is done by extracting the image feature and comparing the features. Images of same identity ought to have similar features and get small feature distance, while images of different identities have large feature distance. With a queried image and a pile of candidate images given, the image that has the smallest distance is identified as the same person as the queried image. The following table shows the ReID detection model supported by the AI Library.

Table 20: The ReID detection model list

No	Model Name	Framework
1	reid	Caffe

Multi-task

The multi-task library is appropriate for a model that has multiple sub-tasks. The Multi-task model in AI Library has two sub-tasks. One is Semantic Segmentation and the other is SSD Detection. The following table shows the Multi-task model supported by the AI Library.

Table 21: The Multi-task model list

No	Model Name	Framework
1	multi_task	Caffe

Model Samples

There are up to 14 model samples that are located in `~/Vitis-AI/Vitis-AI-Library/overview/samples`. Each sample has the following four kinds of test samples.

- test_jpeg_[model type]
- test_video_[model type]
- test_performance_[model type]
- test_accuracy_[model type]

Take yolov3 as an example.

1. Before you run the yolov3 detection example, you can choose one of the following yolov3 model to run.
 - a. yolov3_bdd
 - b. yolov3_voc
 - c. yolov3_voc_tf
2. Ensure that the following test program exists.
 - a. test_jpeg_yolov3
 - b. test_video_yolov3
 - c. test_performance_yolov3
 - d. test_accuracy_yolov3

If the executable program does not exist, it can be compiled and generated as follows:

```
#sh -x build.sh
```

3. To test the image data, execute the following command.

```
#./test_jpeg_yolov3 yolov3_bdd sample_yolov3.jpg
```

You will see the printing result on the terminal. Also, you can view the output image:
`sample_yolov3_result.jpg`.

4. To test the video data, execute the following command.

```
#./test_video_yolov3 yolov3_bdd video_input.mp4 -t 8
```

5. To test the model performance, execute the following command.

```
#./test_performance_yolov3 yolov3_bdd test_performance_yolov3.list -t 8
```

You will see the printing result on the terminal.

6. To test the model accuracy, prepare your own image dataset, image list file and the ground truth of the images. Then execute the following command.

```
#./test_accuracy_yolov3 yolov3_bdd [image_list_file] [output_file]
```

After the `output_file` is generate, a script file is needed to automatically compare the results. Finally, the accuracy result can be obtained.

Programming Examples

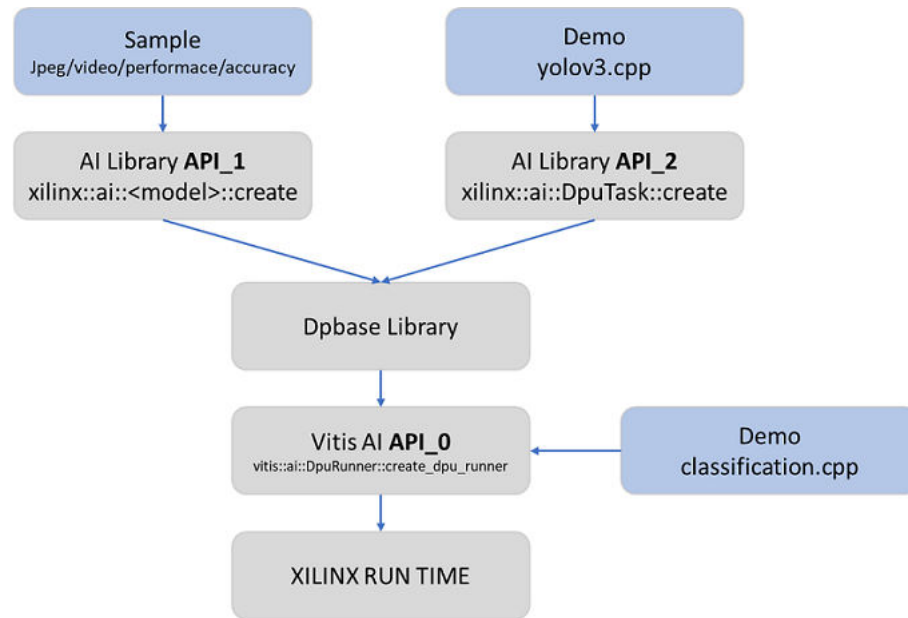
In practice, users have many different application requirements, but it basically falls into three categories. The first is to use the ready-made models provided by Vitis AI Library to quickly build their own applications, and the second is to use users' own custom models which are similar to the models in the AI Library and the last is to use new models that are totally different from the models in the AI Library. This chapter describes the detailed development steps for the first two cases. For the third case, users can also use the AI Library's samples and libraries implementation for reference. Therefore, this chapter describes the following contents:

- How to customize pre-processing
- How to use the configuration file as pre-processing and post-processing parameter
- How to use the AI Library's post-processing library
- How to implement user post-processing code

The following figure shows the relationships of the various AI Library APIs and their corresponding example. And there are three kinds of APIs in this release.

- Vitis AI *API_0*
- AI Library *API_1*
- AI Library *API_2*

Figure 14: The Diagram of AI Library API



Developing With Vitis AI API_0

1. Install the cross-compilation system on the host side, refer to [Chapter 2: Installation](#).
2. Download the `vitis_ai_model_ZCU102_2019.2-r1.1.0.deb` packet, and copy it to the board via scp.
3. Install the Xilinx Model Package on the target side.

```
#dpkg -i vitis_ai_model_ZCU102_2019.2-r1.1.0.deb
```

After the installation, the models can be found in the `/usr/share/vitis_ai_library/models` directory on the target side.

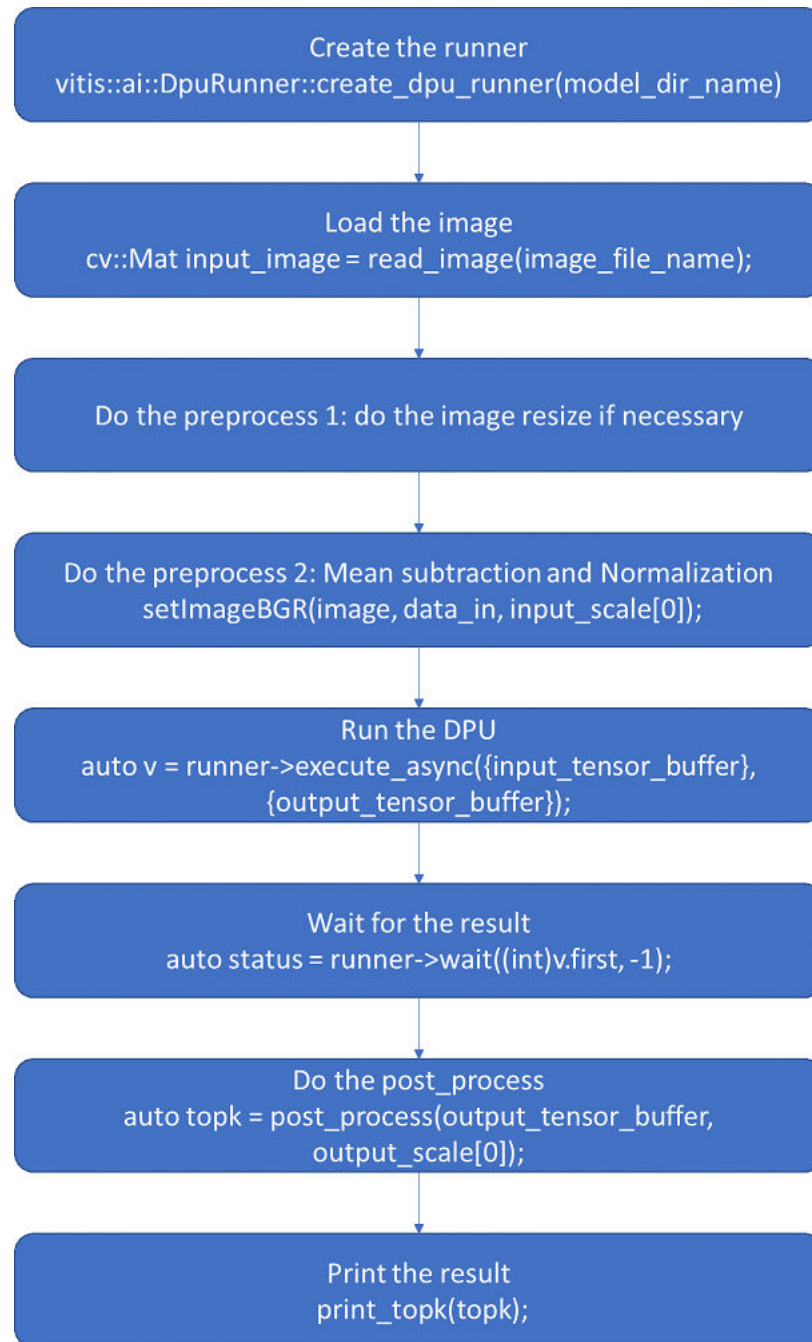
Note: You do not need to install the Xilinx model packet if they want to use their own model.

4. Git clone the corresponding AI Library from <https://github.com/Xilinx/Vitis-AI>.
5. Create a folder under your workspace, using classification as an example.

```
$mkdir classification
```

6. Create the `demo_classification.cpp` source file. The main flow is shown below. See `~/Vitis-AI/Vitis-AI-Library/demo/classification/demo_classification.cpp` for a complete code example.

Figure 15: Main program Flow Chart



7. Create a `build.sh` file as shown below, or copy one from the AI Library's demo and modify it.

```

#!/bin/sh
CXX=${CXX:-g++}
$CXX -std=c++11 -O3 -I. -o demo_classification demo_classification.cpp -
lopencv_core -lopencv_video -lopencv_videoio -lopencv_imgproc -
lopencv_imgcodecs -lopencv_highgui -lglog -lvitis_ai_library-dpu_task -
lvitis_ai_library-model-config -lvart-runner
  
```

8. Cross compile the program.

```
$ssh -x build.sh
```

9. Copy the executable program to the target board via scp.

```
$scp demo_classification root@IP_OF_BOARD:~/
```

10. Execute the program on the target board. Before running the program, make sure the target board has the AI Library installed, and prepare the images you want to test.

```
#./demo_classification resnet50 input_image.jpg
```

Note:

- `demo_classification.cpp` uses user-defined pre-processing parameter as input.
- `demo_classification.cpp` uses user post-processing code. And if you want to use the AI Library's post-processing library, please check [How to Using the AI Library's Post-Processing Library](#)

Developing with User Model and AI Library API_2

When you use their own models, it is important to note that your model framework should be within the scope supported by the Vitis™ AI Library. The following is an introduction of how to deploy a retrained YOLOv3 Caffe model to ZCU102 platform based on Vitis AI Library step by step.

1. Download the corresponding docker image from <https://github.com/Xilinx/Vitis-AI>.
2. Load and run the docker.
3. Create a folder and place the float model under it on the host side, then use `AI Quantizer` tool to do the quantization. For more details, see [Vitis AI User Guide](#) in the *Vitis AI User Documentation* (UG1431).
4. Use `AI Compiler` tool to do the model compiling to get the elf file, such as `yolov3_custom.elf`. For more information, see [Vitis AI User Guide](#) in the *Vitis AI User Documentation* (UG1431).
5. Create the meta.json file, as shown in the following

```
{
  "target": "DPUv2",
  "lib": "libvart-dpu-runner.so",
  "filename": "yolov3_custom.elf",
  "kernel": [ "yolov3_custom" ],
  "config_file": "yolov3_custom.prototxt"
}
```

6. Create the yolov3_custom.prototxt, as shown in the following.

```
model {
  name: "yolov3_custom"
  kernel {
    name: "yolov3_custom"
    mean: 0.0
    mean: 0.0
    mean: 0.0
    scale: 0.00390625
    scale: 0.00390625
    scale: 0.00390625
  }
  model_type : YOLOv3
  yolo_v3_param {
    num_classes: 20
    anchorCnt: 3
    conf_threshold: 0.3
    nms_threshold: 0.45
    biases: 10
    biases: 13
    biases: 16
    biases: 30
    biases: 33
    biases: 23
    biases: 30
    biases: 61
    biases: 62
    biases: 45
    biases: 59
    biases: 119
    biases: 116
    biases: 90
    biases: 156
    biases: 198
    biases: 373
    biases: 326
    test_mAP: false
  }
}
```

Note: The <model_name>.prototxt only take effect when you use AI Library API_1.

When you use AI Library API_2, the parameter of the model needs to be loaded and read manually by the program. See the ~/Vitis-AI/Vitis-AI-Library/demo/yolov3/demo_yolov3.cpp for details.

7. Create the demo_yolov3.cpp file. See ~/Vitis-AI/Vitis-AI-Library/demo/yolov3/demo_yolov3.cpp for reference.
8. Create a build.sh file as shown below, or copy one from the AI Library demo and modify it.

```
#!/bin/sh
CXX=${CXX:-g++}
$CXX -std=c++11 -O3 -I. -o demo_yolov3 demo_yolov3.cpp -lopencv_core -
lopencv_video -lopencv_videoio -lopencv_imgproc -lopencv_imgcodecs -
lopencv_highgui -lglog -lxnnpp-xnnpp -lvitis_ai_library-model_config -
lprotobuf -lvitis_ai_library-dpu_task
```

9. Exit the docker tool system and start the docker runtime system.

10. Cross compile the program, generate executable file `demo_yolov3`.

```
$sh -x build.sh
```

11. Create model folder under `/usr/share/vitis_ai_library/models` on the target side.

```
#mkdir yolov3_custom /usr/share/vitis_ai_library/models
```

Note that `/usr/share/vitis_ai_library/models` is the default location for the program to read the model. You can also place the model folder in the same directory as the executable program.

12. Copy the `yolov3_custom.elf`, `yolov3_custom.prototxt` and `meta.json` to the target and put them under `/usr/share/vitis_ai_library/models/yolov3_custom`.

```
$scp yolov3_custom.elf yolov3_custom.prototxt meta.json
root@IP_OF_BOARD:/usr/share/vitis_ai_library/models/yolov3_custom
```

13. Copy the executable program to the target board using `scp`.

```
$scp demo_yolov3 root@IP_OF_BOARD:~/
```

14. Execute the program on the target board and get the following results. Before running the program, make sure the target board has the AI Library installed, and prepare the images you want to test.

```
#./demo_yolov3 yolov3_custom sample.jpg
```

Customizing Pre-Processing

Before convolution neural network processing, image data generally needs to be pre-processed. The basics of some pre-processing techniques that can be applied to any kind of data are as follows:

- Mean subtraction
- Normalization
- PCA and Whitening

Call the `setMeanScaleBGR` function to implement the Mean subtraction and normalization, as shown in the following figure. See `~/Vitis-AI/Vitis-AI-Library/dpu_task/include/vitis/ai/dpu_task.hpp` for details.

Figure 16: `setMeanScaleBGR` Example

```
// Please check /etc/dpu_model_param.conf.d/ssd_vehicle_v3_480x360.prototxt
// or your caffe model, e.g. deploy.prototxt.
task->setMeanScaleBGR({104.0f, 117.0f, 123.0f}, {1.0f, 1.0f, 1.0f});
```

Call the `cv::resize` function to scale the image, as shown in the following figure.

Figure 17: `cv::resize` Example

```
// Resize it if its size is not match.
cv::Mat image;
auto input_tensor = task->getInputTensor();
CHECK_EQ(input_tensor.size() , 1) << " the dpu model must have only one input";
auto width = input_tensor[0].width;
auto height = input_tensor[0].height;
auto size = cv::Size(width, height);
if (size != input_image.size()) {
    cv::resize(input_image, image, size);
} else {
    image = input_image;
}
```

Using the Configuration File

The Vitis™ AI Library provides a way to read model parameters by reading the configuration file. It facilitates uniform configuration management of model parameters. The configuration file is located in `/usr/share/vitis-ai-library/models/[model_name]/[model_name].prototxt`.

```
model
{
  name: "yolov3_voc"
  kernel {
    name: "yolov3_voc"
    mean: 0.0
    mean: 0.0
    mean: 0.0
    scale: 0.00390625
    scale: 0.00390625
    scale: 0.00390625
  }
  model_type : YOLOv3
  yolo_v3_param {
    ...
  }
  is_tf: false
}
```

Table 22: Compiling Model and Kernel Parameters

Model/Kernel	Parameter Type	Description
model	name	This name should be same as the \${MODEL_NAME}.
	model_type	This type should depend on which type of model you used.

Table 22: Compiling Model and Kernel Parameters (cont'd)

Model/Kernel	Parameter Type	Description
kernel	name	This name should be filled as the result of your DNNC compile. Sometimes, its name may have an extra postfix “_0”, here need fill the name with such postfix. (For example: inception_v1_0)
	mean	Normally there are three lines, each of them corresponding to the mean-value of “BRG” which are pre-defined in the model.
	scale	Normally there are three lines. Each of them is corresponds to the RGB-normalized scale. If the model had no scale in training stage, here should fill with one.
	is_tf	Bool type, if your model is trained by tensorflow, please add this and set with “true”. It could be blank in the prototxt or set as “false” when the model is caffe.

yolo_v3_param

```

model_type : YOLOv3
yolo_v3_param {
  num_classes: 20
  anchorCnt: 3
  conf_threshold: 0.3
  nms_threshold: 0.45
  biases: 10
  biases: 13
  biases: 16
  biases: 30
  biases: 33
  biases: 23
  biases: 30
  biases: 61
  biases: 62
  biases: 45
  biases: 59
  biases: 119
  biases: 116
  biases: 90
  biases: 156
  biases: 198
  biases: 373
  biases: 326
  test_mAP: false
}

```

Below are the YOLOv3 model's parameters. You can modify them as your model requires.

Table 23: YOLOv3 Model Parameters

Parameter Type	Description
num_classes	The actual number of the model's detection categories
anchorCnt	The number of this model's anchor
conf_threshold	The threshold of the boxes' confidence, which could be modified to fit your practical application

Table 23: YOLOv3 Model Parameters (cont'd)

Parameter Type	Description
nms_threshold	The threshold of NMS
biases	These parameters are same as the model's. Each bias need writes in a sperate line. (Biases amount) = anchorCnt * (output-node amount) * 2. set correct lines in the prototxt.
test_mAP	If your model was trained with letterbox and you want to test its mAP, set this as "true". Normally it is "false" for executing much faster.

SSD_param

```

model_type : SSD
ssd_param :
{
  num_classes : 4
  nms_threshold : 0.4
  conf_threshold : 0.0
  conf_threshold : 0.6
  conf_threshold : 0.4
  conf_threshold : 0.3
  keep_top_k : 200
  top_k : 400
  prior_box_param {
    layer_width : 60,
    layer_height: 45,
    variances: 0.1
    variances: 0.1
    variances: 0.2
    variances: 0.2
    min_sizes: 21.0
    max_sizes: 45.0
    aspect_ratios: 2.0
    offset: 0.5
    step_width: 8.0
    step_height: 8.0
    flip: true
    clip: false
  }
}

```

Below are the SSD parameters. The parameters of SSD-model include all kinds of threshold and PriorBox requirements. You can reference your SSD deploy.prototxt to fill them.

Table 24: SSD Model Parameters

Parameter Type	Description
num_classes	The actual number of the model's detection categories
anchorCnt	The number of this model's anchor
conf_threshold	The threshold of the boxes' confidence. Each category could have a different threshold, but its amount must be equal to num_classes.
nms_threshold	The threshold of NMS

Table 24: SSD Model Parameters (cont'd)

Parameter Type	Description
biases	These parameters are same as the model's. Each bias need writes in a separate line. (Biases amount) = anchorCnt * (output-node amount) * 2. Set correct lines in the prototxt.
test_mAP	If your model was trained with letterbox and you want to test its mAP, set this as "true". Normally it is "false" for executing much faster
keep_top_k	Each category of detection objects' top K boxes
top_k	All the detection object's top K boxes, except the background (the first category)
prior_box_param	There is more than one PriorBox, which could be found in the original model (deploy.prototxt) for corresponding each different scale. These PriorBoxes should oppose each other. (see the following table for Prior Box Parameters)

Table 25: PriorBox Parameters

Parameter Type	Description
layer_width/layer_height	The input width/height of this layer. Such numbers could be computed from the net structure.
ariances	These numbers are used for boxes regression, just only to fill them as original model. There should be four variances.
min_sizes/max_size	Filled as the "deploy.prototxt", but each number should be written in a separate line.
aspect_ratios	The ratio's number (each one should be written in a separate line). Default has 1.0 as its first ratio. If you set a new number here, there will be two ratios created when the opposite is true. One is a filled number; another is its reciprocal. For example, this parameter has only one set element, "ratios: 2.0". The ratio vector has three numbers: 1.0, 2.0, 0.5
offset	Normally, the PriorBox is created by each central point of the feature map, so that offset is 0.5.
step_width/step_height	Copy from the original file. If there are no such numbers there, you can use the following formula to compute them: step_width = img_width ÷ layer_width step_height = img_height ÷ layer_height
offset	Normally, PriorBox is created by each central point of the feature map, so that the offset is 0.5.
flip	Control whether rotate the PriorBox and change the ratio of length/width.
clip	Set as false. If true, it will let the detection boxes' coordinates keep at [0, 1].

Example Code

The following is the example code.

```
Mat img = cv::imread(argv[1]);
auto yolo = vitis::ai::YOLOv3::create("yolov3_voc", true);
auto results = yolo->run(img);
for(auto &box : results.bboxes){
    int label = box.label;
    float xmin = box.x * img.cols + 1;
    float ymin = box.y * img.rows + 1;
    float xmax = xmin + box.width * img.cols;
```



```
float ymax = ymin + box.height * img.rows;
if(xmin < 0.) xmin = 1.;
if(ymin < 0.) ymin = 1.;
if(xmax > img.cols) xmax = img.cols;
if(ymax > img.rows) ymax = img.rows;
float confidence = box.score;
cout << "RESULT: " << label << "\t" << xmin << "\t" << ymin <<
"\t"
<< xmax << "\t" << ymax << "\t" << confidence << "\n";
rectangle(img, Point(xmin, ymin), Point(xmax, ymax), Scalar(0, 255, 0),
1, 1, 0);
}
imshow("", img);
waitKey(0);
```

You should use the “create” to create the YOLOv3 object.

```
static std::unique_ptr<YOLOv3> create(const std::string& model_name, bool
need_mean_scale_process = true);
```

Note: The model_name is same as the prototxt's. For more details about the example, refer to ~/Vitis-AI/Vitis-AI-Library/yolov3/test/test_yolov3.cpp.

Implementing User Post-Processing Code

You can also call their own post-processing functions on their own request. Take demo_yolov3.cpp and demo_classification.cpp as an example. Use vitis::ai::DpuTask::create or vitis::ai::DpuRunner::create_dpu_runner to create the task, and after DPU processing is complete, the post-processing function can be invoked. The post_process function in the following figure is a user post-processing code.

Figure 18: User Post-Processing Code Example

```
// start the dpu
task->run();
// get output.
auto output_tensor = task->getOutputTensor();
// post process
auto topk = post_process(output_tensor[0]);
// print the result
print_topk(topk);
return 0;

static std::vector<std::pair<int, float>> post_process(
    const xilinx::ai::OutputTensor &tensor) {
    // run softmax
    auto softmax_input = convert_fixpoint_to_float(tensor);
    auto softmax_output = softmax(softmax_input);
    constexpr int TOPK = 5;
    return topk(softmax_output, TOPK);
}

static std::vector<float> convert_fixpoint_to_float(
    const xilinx::ai::OutputTensor &tensor) {
    auto scale = xilinx::ai::tensor_scale(tensor);
    auto data = (signed char *)tensor.data;
    auto size = tensor.width * tensor.height * tensor.channel;
    auto ret = std::vector<float>(size);
    transform(data, data + size, ret.begin(),
        [scale](signed char v) { return ((float)v) * scale; });
    return ret;
}
```

For more details, see `~/vitis-ai/vitis_ai_library/overview/demo/classification/demo_classification.cpp`.

How to Using the AI Library's Post-Processing Library

Post-processing is an important step in the whole process. Each neural network has different post-processing methods. The `xnnpp` post-processing library is provided in AI Library to facilitate user calls. It is a closed source library. It supports the following neural network post-processing.

- Classification
- Face detection
- Face landmark detection
- SSD detection
- Pose detection
- Semantic segmentation

- Road line detection
- YOLOV3 detection
- YOLOV2 detection
- Openpose detection
- RefineDet detection
- Reld detection
- Multi-task

There are two ways to call xnnpp.

- One is automatic call, through `vitis::ai::<model>::create` create the task, such as `vitis::ai::YOLOv3::create("yolov3_bdd", true)`. After `<model>->run` finished, xnnpp will be automatically processed, users can modify the parameters through the model configuration file.
- One is manual call, through `vitis::ai::DpuTask::create` to create the task. Then, create the object of the post-process and run the post-process. Take SSD post-processing as an example, the specific steps are as follows.

1. Create a config and set the correlating data to control post-process.

```
using DPU_conf = vitis::ai::proto::DpuModelParam;
DPU_conf config;
```

2. If it is a caffemodel, set the "is_tf" false.

```
config.set_is_tf(false);
```

3. Fill other parameters.

```
fillconfig(config);
```

4. Create an object of SSDPostProcess.

```
auto input_tensor = task->getInputTensor();
auto output_tensor = task->getOutputTensor();
auto ssd = vitis::ai::SSDPostProcess::create(input_tensor,
output_tensor, config);
```

5. Run the post-process.

```
auto results = ssd->ssd_post_process();
```

Note: For more details about the post processing examples, see the `~/vitis-ai/vitis_ai_library/overview/demo/yolov3/demo_yolov3.cpp` and `~/vitis-ai/vitis_ai_library/yolov3/test/test_yolov3.cpp` in the host syetem.

Application Demos

This chapter describes how to set up a test environment and to run the application demos. There are two application demos provided within the VitisAI Library. Here, we take ZCU102 board as the test platform.

Demo Overview

There are two application demos provided within the Vitis AI Library. They use the AI Library to build their applications. The codes are stored in `~/Vitis-AI/Vitis-AI-Library/overview/demo/seg_and_roadline_detect` and `~/Vitis-AI/Vitis-AI-Library/overview/demo/seg_and_pose_detect`.

`seg_and_roadline_detect` is a demo that includes multi-task segmentation network processing, vehicle detection and road line detection. It simultaneously performs 4-channel segmentation and vehicle detection and 1-channel road lane detection.

`seg_and_pose_detect` is a demo that includes multi-task segmentation network processing and pose detection. It simultaneously performs 1-channel segmentation process and 1-channel pose detection.

Note: To achieve the best performance, the demos use the DRM (Direct Render Manager) for video display. Log in the board using `ssh` or serial port and run the demo remotely. If you do not want to use DRM for video display, set "USE_DRM=0" in the compile option.

Demo Platform and Setup

Demo Platform

- **Hardware:**
 - 1 x ZCU102 Prod Silicon <https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html>
 - 1 x Win7/10 laptop
 - 1 x 16GB SD card

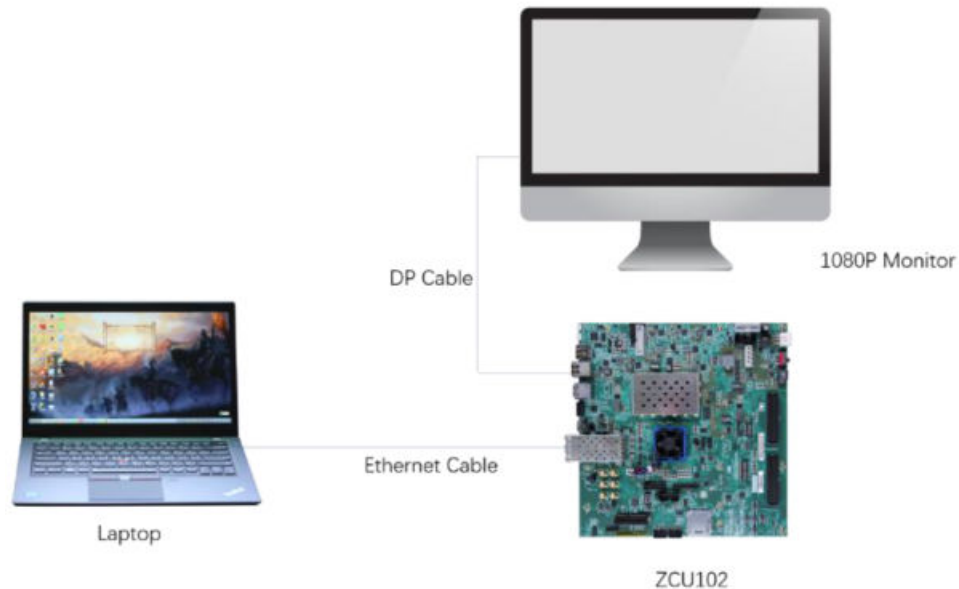
- 1 x Ethernet cables
- 1 x DP 1080P compatible monitor
- 1 x DP cable
- **Software:**
 - ZCU102 board image <https://www.xilinx.com/products/design-tools/ai-inference/ai-developer-hub.html#edge>
 - Vitis-AI-Library
 - Images and video files
 - Terminal software like MobaXterm, Putty

DPU Configuration & Dev Tool Used

- 3xB4096 @281MHz
- Vivado 2019.2, AI Library r1.1

Demo Setup Illustration

Figure 19: Demo Setup



Demo 1: Multi-Task Segmentation + Car Detection and Road Line Detection

Target Application

- ADAS/AD

AI Model, Performance, and Power

- FPN
 - 512x288, 4ch, 20fps
- VPGNET
 - 640x480, 1ch, 56fps
- 20W @ ZU9EG

Build and Run the Demo

Build the demo in the host and copy the program to the target board.

```
$cd ~/Vitis-AI/Vitis-AI-Library/overview/demo/secs_and_roadline_detect
$bash -x build.sh
$scp secs_and_roadline_detect_x secs_and_roadline_detect_drm
root@IP_OF_BOARD:~/
```

To use OpenCV display, run the following command.

```
#!/secs_and_roadline_detect_x seg_512_288.avi seg_512_288.avi
seg_512_288.aviseq_512_288.avi lane_640_480.avi -t 2 -t 2 -t 2 -t 2 -t 3
>/dev/null 2>&1
```

If you want to use DRM display, please connect to the board using SSH and run the following command.

```
#!/etc/init.d/weston stop
#!/secs_and_roadline_detect_drm seg_512_288.avi seg_512_288.avi
seg_512_288.avi
seg_512_288.avi lane_640_480.avi -t 2 -t 2 -t 2 -t 2 -t 3 >/dev/null 2>&1
```

Note that the video files are in the `vitis_ai_library_r1.1_video.tar.gz`. Please download the package from <https://github.com/Xilinx/Vitis-AI>.

Demo Picture

Figure 20: Segmentation and Roadline Detection Demo Picture



Demo 2: Multi-Task Segmentation+Car Detection and Pose Detection

Target Application

- ADAS/AD
- Smartcity

AI Model, Performance, and Power

- FPN
 - 960x540, 1ch, 30fps
- Openpose
 - 960x540, 1ch, 30fps
- 20W @ ZU9EG

Build and Run the Demo

Build the demo in the host and copy the program to the target board.

```
$cd ~/Vitis-AI/Vitis-AI-Library/overview/demo/seg_and_pose_detect
$bash -x build.sh
$scp seg_and_pose_detect_x seg_and_pose_detect_drm root@IP_OF_BOARD:~/
```

To use OpenCV display, run the following command.

```
#./seg_and_pose_detect_x seg_960_540.avi pose_960_540.avi -t 4 -t 4 >/dev/
null 2>&1
```

If you want to use DRM display, please connect to the board via SSH. And run the following command.

```
#!/etc/init.d/weston stop
#./seg_and_pose_detect_drm seg_960_540.avi pose_960_540.avi -t 4 -t 4 >/dev/
null 2>&1
```

The video files are in the `vitis_ai_library_r1.1_video.tar.gz`. Please download the package from <https://github.com/Xilinx/Vitis-AI>.

Demo Picture

Figure 21: Segmentation and Pose Detection Demo Picture



Programming APIs

For details about the Programming APIs, see the *Vitis AI Library Programming Guide* (UG1355). You can download it from the Xilinx website <https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html>.

Also, for the Vitis AI APIs, see the *Vitis AI User Guide* in the *Vitis AI User Documentation* (UG1431). You can download it from the Xilinx website <https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html>.

Performance

This chapter describes in detail the performance of the Vitis™ AI Library on the following different boards.

- ZCU102 (0432055-05)
- ZCU104
- Alveo U50

ZCU102 Performance

The ZCU102 evaluation board uses the mid-range ZU9 UltraScale+ device. There are two different hardware versions of ZCU102 board, one with the serial number 0432055-04 as the header and the other with the serial number 0432055-05 as the header. The performance of the Vitis AI Library varies between the two hardware versions (because of different DDR performance). Since 0432055-04 version of ZCU102 has been discontinued, the following table only shows the performance of ZCU102 (0432055-05). In ZCU102 board, triple B4096F DPU cores are implemented in program logic.

Refer to the following table for throughput performance (in frames/sec or fps) for various neural network samples on ZCU102 (0432055-05) with DPU running at 281 MHz.

Table 26: ZCU102 (0432055-05) Performance

No	Neural Network	Input Size	GOPS	Performance (fps) (Single thread)	Performance (fps) (Multiple thread)
1	inception_resnet_v2_tf	299x299	26.4	23.3	51
2	inception_v1_tf	224x224	3.0	172.1	455.6
3	inception_v3_tf	299x299	11.5	55.9	136.8
4	inception_v4_2016_09_09_tf	299x299	24.6	28.1	71.4
5	mobilenet_v1_0_25_128_tf	128x128	0.027	848.6	2260.9
6	mobilenet_v1_0_5_160_tf	160x160	0.15	577.2	1913.7
7	mobilenet_v1_1_0_224_tf	224x224	1.1	261.3	788.6
8	mobilenet_v2_1_0_224_tf	224x224	0.60	218.6	598.7
9	mobilenet_v2_1_4_224_tf	224x224	1.2	162.4	412

Table 26: ZCU102 (0432055-05) Performance (cont'd)

No	Neural Network	Input Size	GOPS	Performance (fps) (Single thread)	Performance (fps) (Multiple thread)
10	resnet_v1_101_tf	224x224	14.4	43.5	94.5
11	resnet_v1_152_tf	224x224	21.8	29.9	66.1
12	resnet_v1_50_tf	224x224	7.0	78.3	164.7
13	vgg_16_tf	224x224	31.0	19.8	44.7
14	vgg_19_tf	224x224	39.3	17.1	40.3
15	ssd_mobilenet_v1_coco_tf	300x300	2.5	87.2	323.5
16	ssd_mobilenet_v2_coco_tf	300x300	3.8	63	198.7
17	ssd_resnet_50_fpn_coco_tf	640x640	178.4	1.3	5
18	yolov3_voc_tf	416x416	65.6	13.5	37.8
19	mlperf_ssd_resnet34_tf	1200x1200	433	1.8	7.7
20	resnet50	224x224	7.7	72.8	155.5
21	resnet18	224x224	3.7	174.7	461.6
22	inception_v1	224x224	3.2	166.6	444
23	inception_v2	224x224	4.0	136.7	335.6
24	inception_v3	299x299	11.4	56	138.5
25	inception_v4	299x299	24.5	28.1	71.3
26	mobilenet_v2	224x224	0.6	214.7	580.6
27	squeezenet	227x227	0.76	269.4	1045.9
28	ssd_pedestrain_pruned_0_97	360x360	5.9	75.7	294.4
29	ssd_traffic_pruned_0_9	360x480	11.6	54.4	206.7
30	ssd_adas_pruned_0_95	360x480	6.3	82.6	296.3
31	ssd_mobilenet_v2	360x480	6.6	38.4	112.5
32	refinedet_pruned_0_8	360x480	25	31.7	103.8
33	refinedet_pruned_0_92	360x480	10.1	59.8	204
34	refinedet_pruned_0_96	360x480	5.1	82.6	290.1
35	vpgnet_pruned_0_99	480x640	2.5	102.3	397.3
36	fpn	256x512	8.9	59.7	185.2
37	sp_net	128x224	0.55	491.6	1422.8
38	openpose_pruned_0_3	368x368	49.9	3.5	15.3
39	densebox_320_320	320x320	0.49	388.3	1279
40	densebox_640_360	360x640	1.1	195	627.8
41	face_landmark	96x72	0.14	846.7	1379.9
42	reid	80x160	0.95	361.9	672.8
43	multi_task	288x512	14.8	35.4	133
44	yolov3_adas_pruned_0_9	256x512	5.5	83.8	235.3
45	yolov3_voc	416x416	65.4	13.5	38.2
46	yolov3_bdd	288x512	53.7	13	37.1

Table 26: ZCU102 (0432055-05) Performance (cont'd)

No	Neural Network	Input Size	GOPS	Performance (fps) (Single thread)	Performance (fps) (Multiple thread)
47	yolov2_voc	448x448	34	24.8	77.1
48	yolov2_voc_pruned_0_66	448x448	11.6	53.2	194.2
49	yolov2_voc_pruned_0_71	448x448	9.9	59.8	224
50	yolov2_voc_pruned_0_77	448x448	7.8	68	266.2

ZCU104 Performance

The ZCU104 evaluation board uses the mid-range ZU7ev UltraScale+ device. Dual B4096F DPU cores are implemented in program logic and delivers 2.4 TOPS INT8 peak performance for deep learning inference acceleration.

Refer to the following table for the throughput performance (in frames/sec or fps) for various neural network samples on ZCU104 with DPU running at 300 MHz.

Table 27: ZCU104 Performance

No	Neural Network	Input Size	GOPS	Performance (fps) (Single thread)	Performance (fps) (Multiple thread)
1	inception_resnet_v2_tf	299x299	26.4	25.4	46.1
2	inception_v1_tf	224x224	3.0	196.1	401.7
3	inception_v3_tf	299x299	11.5	60.3	116.4
4	inception_v4_2016_09_09_tf	299x299	24.6	30.2	58.6
5	mobilenet_v1_0_25_128_tf	128x128	0.027	1263.6	3957.7
6	mobilenet_v1_0_5_160_tf	160x160	0.15	763.1	2038.1
7	mobilenet_v1_1_0_224_tf	224x224	1.1	311.8	731.1
8	mobilenet_v2_1_0_224_tf	224x224	0.60	250.9	546.6
9	mobilenet_v2_1_4_224_tf	224x224	1.2	185.5	381.5
10	resnet_v1_101_tf	224x224	14.4	47.3	87.2
11	resnet_v1_152_tf	224x224	21.8	32.5	60.1
12	resnet_v1_50_tf	224x224	7.0	86.3	157.5
13	vgg_16_tf	224x224	31.0	21.3	38.3
14	vgg_19_tf	224x224	39.3	18.4	33.8
15	ssd_mobilenet_v1_coco_tf	300x300	2.5	92.4	330
16	ssd_mobilenet_v2_coco_tf	300x300	3.8	66.7	185
17	ssd_resnet_50_fpn_coco_tf	640x640	178.4	1.3	5.1
18	yolov3_voc_tf	416x416	65.6	14.1	29.3

Table 27: ZCU104 Performance (cont'd)

No	Neural Network	Input Size	GOPS	Performance (fps) (Single thread)	Performance (fps) (Multiple thread)
19	mlperf_ssd_resnet34_tf	1200x1200	433	1.6	5.3
20	resnet50	224x224	7.7	80.2	146.8
21	resnet18	224x224	3.7	196.7	403.7
22	inception_v1	224x224	3.2	189.8	387
23	inception_v2	224x224	4.0	152.7	298.2
24	inception_v3	299x299	11.4	60.5	117.3
25	inception_v4	299x299	24.5	30.2	58.6
26	mobilenet_v2	224x224	0.6	249.3	536.6
27	squeezenet	227x227	0.76	274.4	941.8
28	ssd_pedestrain_pruned_0_97	360x360	5.9	78.1	221.5
29	ssd_traffic_pruned_0_9	360x480	11.6	56.1	153.2
30	ssd_adas_pruned_0_95	360x480	6.3	84.7	231.9
31	ssd_mobilenet_v2	360x480	6.6	25.4	101.3
32	refinedet_pruned_0_8	360x480	25	32.6	76.1
33	refinedet_pruned_0_92	360x480	10.1	61.3	154.2
34	refinedet_pruned_0_96	360x480	5.1	83.6	228.7
35	vpgnet_pruned_0_99	480x640	2.5	105.5	364.9
36	fpn	256x512	8.9	62	169.9
37	sp_net	128x224	0.55	552.4	1245.6
38	openpose_pruned_0_3	368x368	49.9	3.6	11
39	densebox_320_320	320x320	0.49	397.4	1250.3
40	densebox_640_360	360x640	1.1	198.7	606.6
41	face_landmark	96x72	0.14	890.1	1363.2
42	reid	80x160	0.95	385.6	668.8
43	multi_task	288x512	14.8	36	108.4
44	yolov3_adas_pruned_0_9	256x512	5.5	84.5	218.5
45	yolov3_voc	416x416	65.4	14.2	29.5
46	yolov3_bdd	288x512	53.7	13.6	28.7
47	yolov2_voc	448x448	34	26.2	59.1
48	yolov2_voc_pruned_0_66	448x448	11.6	55.6	153.2
49	yolov2_voc_pruned_0_71	448x448	9.9	62.4	180.2
50	yolov2_voc_pruned_0_77	448x448	7.8	70.5	217.4

U50 Performance

The Xilinx® Alveo U50 Data Center accelerator cards are peripheral component interconnect express (PCIe®) Gen3x16 compliant and Gen4x8 compatible cards featuring the Xilinx 16 nm UltraScale+ technology. In this release, DPUv3e is implemented in program logic for deep learning inference acceleration.

See the following table for the throughput performance (in frames/sec or fps) for various neural network samples on U50 Gen3x16 with DPUv3e running at 250 MHz.

Table 28: U50 Performance

No	Neural Network	Input Size	GOPS	Performance (fps) (Single thread)	Performance (fps) (Multiple thread)
1	inception_resnet_v2_tf	299x299	26.4	21.9	46.5
2	inception_v1_tf	224x224	3.0	227.7	682
3	inception_v3_tf	299x299	11.5	57	133.3
4	inception_v4_2016_09_09_tf	299x299	24.6	28.5	61.3
5	mobilenet_v1_0_25_128_tf	128x128	0.027	N/A	N/A
6	mobilenet_v1_0_5_160_tf	160x160	0.15	N/A	N/A
7	mobilenet_v1_1_0_224_tf	224x224	1.1	N/A	N/A
8	mobilenet_v2_1_0_224_tf	224x224	0.60	N/A	N/A
9	mobilenet_v2_1_4_224_tf	224x224	1.2	N/A	N/A
10	resnet_v1_101_tf	224x224	14.4	104.9	247
11	resnet_v1_152_tf	224x224	21.8	74	165.8
12	resnet_v1_50_tf	224x224	7.0	169.9	460.7
13	vgg_16_tf	224x224	31.0	62.3	137.9
14	vgg_19_tf	224x224	39.3	52.6	114.4
15	ssd_mobilenet_v1_coco_tf	300x300	2.5	N/A	N/A
16	ssd_mobilenet_v2_coco_tf	300x300	3.8	N/A	N/A
17	ssd_resnet_50_fpn_coco_tf	640x640	178.4	5.8	15.2
18	yolov3_voc_tf	416x416	65.6	24.6	54.7
19	mlperf_ssd_resnet34_tf	1200x1200	433	N/A	N/A
20	resnet50	224x224	7.7	166.4	394
21	resnet18	224x224	3.7	334.6	995
22	inception_v1	224x224	3.2	212.1	551
23	inception_v2	224x224	4.0	175.5	426.4
24	inception_v3	299x299	11.4	60.5	133.3
25	inception_v4	299x299	24.5	29.4	61.5
26	mobilenet_v2	224x224	0.6	N/A	N/A
27	squeezenet	227x227	0.76	166.3	418

Table 28: U50 Performance (cont'd)

No	Neural Network	Input Size	GOPS	Performance (fps) (Single thread)	Performance (fps) (Multiple thread)
28	ssd_pedestrian_pruned_0_97	360x360	5.9	33.9	83.1
29	ssd_traffic_pruned_0_9	360x480	11.6	36.3	91.1
30	ssd_adas_pruned_0_95	360x480	6.3	46.5	118.1
31	ssd_mobilenet_v2	360x480	6.6	N/A	N/A
32	refinedet_pruned_0_8	360x480	25	28.7	64.4
33	refinedet_pruned_0_92	360x480	10.1	41.3	97.6
34	refinedet_pruned_0_96	360x480	5.1	41	96.8
35	vpgnet_pruned_0_99	480x640	2.5	28.3	65.1
36	fpn	256x512	8.9	37.3	116.9
37	sp_net	128x224	0.55	405.5	1074.5
38	openpose_pruned_0_3	368x368	49.9	8.7	22.3
39	densebox_320_320	320x320	0.49	238.4	796.2
40	densebox_640_360	360x640	1.1	103.3	360.6
41	face_landmark	96x72	0.14	2107.7	6631.9
42	reid	80x160	0.95	751.4	2301
43	multi_task	288x512	14.8	24.9	70.8
44	yolov3_adas_pruned_0_9	256x512	5.5	33.1	75.4
45	yolov3_voc	416x416	65.4	24.5	54.9
46	yolov3_bdd	288x512	53.7	19.2	42
47	yolov2_voc	448x448	34	50.9	141.6
48	yolov2_voc_pruned_0_66	448x448	11.6	63.4	187.5
49	yolov2_voc_pruned_0_71	448x448	9.9	66.3	203
50	yolov2_voc_pruned_0_77	448x448	7.8	70.9	227.9

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see [Xilinx Support](#).

Documentation Navigator and Design Hubs

Xilinx[®] Documentation Navigator (DocNav) provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

- From the Vivado[®] IDE, select **Help** → **Documentation and Tutorials**.
- On Windows, select **Start** → **All Programs** → **Xilinx Design Tools** → **DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In DocNav, click the **Design Hubs View** tab.
- On the Xilinx website, see the [Design Hubs](#) page.

Note: For more information on DocNav, see the [Documentation Navigator](#) page on the Xilinx website.

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

© Copyright 2019-2020 Xilinx, Inc. Xilinx, the Xilinx logo, Alveo, Artix, Kintex, Spartan, Versal, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. AMBA, AMBA Designer, Arm, ARM1176JZ-S, CoreSight, Cortex, PrimeCell, Mali, and MPCore are trademarks of Arm Limited in the EU and other countries. PCI, PCIe, and PCI Express are trademarks of PCI-SIG and used under license. All other trademarks are the property of their respective owners.