# DNNDK User Guide for the SDSoC Development Environment

**UG1331 (v 1.0) January 22, 2019**

# Revision History

The following table shows the revision history for this document.

| Section | Revision Summary |
|---|---|
| **01/22/2019 Version 1.0** | |
| General updates | Initial Xilinx release. |

XILINX

# *Table of Contents*


Revision History ..... 2
Table of Contents ..... 3
Chapter 1: Overview ..... 4
The SDSoC Development Environment ..... 4
DNNDK ..... 4
Chapter 2: Prepare the Environment ..... 7
Prepare the Evaluation Board ..... 7
Set Up the Software Environment ..... 9
Chapter 3: Quick Start ..... 11
Hardware ..... 11
System ..... 11
Libraries ..... 12
Application ..... 13
Chapter 4: DNNDK Workspace ..... 14
Workspace Overview ..... 14
Hardware Project ..... 15
Gstreamer Plugin Project ..... 17
Standalone Application Project ..... 18
Chapter 5: Running Samples on Evaluation Board ..... 19
Steps for Running an Application ..... 19
Application Descriptions ..... 21
Legal Notices 25
Please Read: Important Legal Notices ..... 25

# The SDSoC Development Environment

The Xilinx SDSoC™ development environment provides a familiar embedded C/C++/OpenCL application development experience including an easy-to-use Eclipse IDE, and a comprehensive design environment for heterogeneous Zynq®-7000 SoC and Zynq UltraScale+™ MPSoC deployment. Complete with the industry's first C/C++/OpenCL full-system optimizing compiler, SDSoC delivers system level profiling, automated software acceleration in programmable logic, automated system connectivity generation, and libraries to speed programming. It also enables end-user and third-party platform developers to rapidly define, integrate, and verify system level solutions, and enable their end customers with a customized programming environment. For more information, refer to the Xilinx website.

# DNNDK

The Deep Neural Network Development Kit (DNNDK) is designed as an integrated framework, which aims to simplify and accelerate deep learning application development and deployment on Deep learning Processor Unit (DPU). DNNDK is an optimizing inference engine, and it makes the computing power of DPU become easily accessible. It offers the best of simplicity and productivity to develop deep learning applications, covers the phases of neural network model compression, programming, compilation, and runtime enablement. Refer to the Xilinx website for a quick overview of DNNDK, or download the *DNNDK User Guide* (UG1327) from the Xilinx Support website for more detailed information.

DNNDK provides a set of easy-to-use C/C++ programming APIs for deep learning application developers. It is seamlessly integrated into the SDSoC development environment since the 2018.3 release. This offers the capability for SDSoC users to easily deploy many diverse deep learning algorithms on Xilinx Zynq-7000 SoC and Zynq UltraScale+ MPSoC devices with DNNDK toolchain and programming APIs.

The DNNDK package for SDSoC 2018.3 is available for free download from the Xilinx Downloads website. Notice, however, that some tools such as DExplorer and DSight are not supported in the DNNDK for the SDSoC environment.

There are four sub-directories included in this package. This document refers to the DeePhi package.

```
deephi_dnndk_sdsoc_2018.3
    |-- dnndk_prebuilt
    |   |-- bin
    |   |-- bootfiles_zcu102
    |   |-- bootfiles_zcu104
    |   |-- lib
    |   |-- resnet50
    |   |-- scripts
    |   `-- video
    |-- dnndk_ws
    |   |-- dpucore_zu7
    |   |-- dpucore_zu9
    |   |-- gstsdxfacedetect
    |   |-- gstsdxgesturedetect
    |   |-- gstsdximgclassifier
    |   |-- gstsdxpedestriandetect
    |   |-- gstsdxsegmentation
    |   |-- gstsdxtrafficdetect
    |   |-- include
    |   |-- lib
    |   `-- resnet50
    `-- host_x86
        |-- install.sh
        |-- models
        `-- pkgs
```

## *deephi_prebuilt*

This directory contains the pre-built binaries and scripts for running the demos.

- **bin**: video_cmd tool
- **bootfiles_zcu102**: Pre-built system files for ZCU102, with two DPU cores inside that run at 300MHz.
- **bootfiles_zcu104**: Pre-built system files for ZCU104, with one DPU core inside that runs at 300MHz.
- **lib**: Libraries related to DPU v1.3.0. See Table 1 for details.
- **resnet50**: A standalone resnet50 application.
- **scripts**: Scripts for running the detection demos.
- **video**: Input video files. Notice that the gstreamer detection scripts take raw `.bgr` files as input. You must convert these videos to raw `.bgr` format first. See Chapter 5: Running Samples on Evaluation Board for more details.

**Table 1. Description of Prebuilt Libraries**

| libraries | description | demo |
|---|---|---|
| libn2cube.so, libdputils.so | DNNDK libraries, encapsulating high level APIs of DPU | All demos |
| libdpumodelssd.so | DPU model for SSD network | traffic detection |
| libgstsdxtrafficdetect.so | gstreamer plugin for traffic detection | |
| libdpumodelyolo_google.so | DPU model for YOLOv2 network | pedestrian detection |
| libgstsdxpedestriandetect.so | gstreamer plugin for pedestrian detection | |
| libdpumodeldensebox.so | DPU model for densebox network | face detection |
| libgstsdxfacedetect.so | gstreamer plugin for face detection | |
| libdpumodel14pt.so | DPU model for human joints network | gesture detection |
| libdpumodelssd_person.so | DPU model for SSD network targeting person detection | |
| libgstsdxgesturedetect.so | gstreamer plugin for gesture detection | |
| libdpumodelsegmentation.so | DPU model for segmentation network | segmentation |
| libgstsdxsegmentation.so | gstreamer plugin for segmentation | |
| libdpumodelresnet50.so | DPU model for ResNet50 network | image classification |
| libgstsdximgclassifier.so | gstreamer plugin for image classification | |

## *deephi_ws*

This is the workspace of the DPU demos. The projects list as follows

- **dpucore_zu7**: A hardware project that integrates DPU v1.3.0 c-callable IP for zu7 FPGA(ZCU104).
- **dpucore_zu9**: A hardware project that integrates DPU v1.3.0 c-callable IP for zu9 FPGA(ZCU102).
- **gstsdxfacedetect**: A face detection gstreamer plugin project.
- **gstsdxgesturedetect**: A gesture detection gstreamer plugin project.
- **gstsdximgclassifier**: A image classify gstreamer plugin project.
- **gstsdxpedestriandetect**: A pedestrian detection gstreamer plugin project.
- **gstsdxsegmentation**: A scene segmentation gstreamer plugin project.
- **gstsdxtrafficdetect**: A traffic detection gstreamer plugin project.
- **resnet50**: A standalone project which implements classification using resnet50 model.

## *host_x86*

This directory contains the Deep Compression Tool (DECENT), Deep Neural Network Compiler (DNNC) tools, and a trained float resnet50 Caffe model.

- DECENT is a quantized tool that can convert the float model to a quantized model.
- DNNC is used to compile the quantized model for specified DPU.

There are two evaluation boards enabled and verified for the DNNDK v2.08 beta release for the Xilinx® SDSoC™ development environment: ZCU102 and ZCU104. If you plan to evaluate it with your own evaluation board, create a corresponding SDSoC platform and integrate DPU C-Callable IP and DNNDK.

# Preparing the Evaluation Board

## *Preparing the ZCU102 Evaluation Board*

The Xilinx ZCU102 evaluation board enables you to jumpstart designs for machine learning, automotive, industrial, video, and communications applications. For more information about ZCU102, refer to the Xilinx Zynq® UltraScale+™ MPSoC ZCU102 Evaluation Kit website. The main peripheral and connection interfaces for ZCU102 are shown in the following figure. For ZCU102, two DPU B4096 cores can be used.

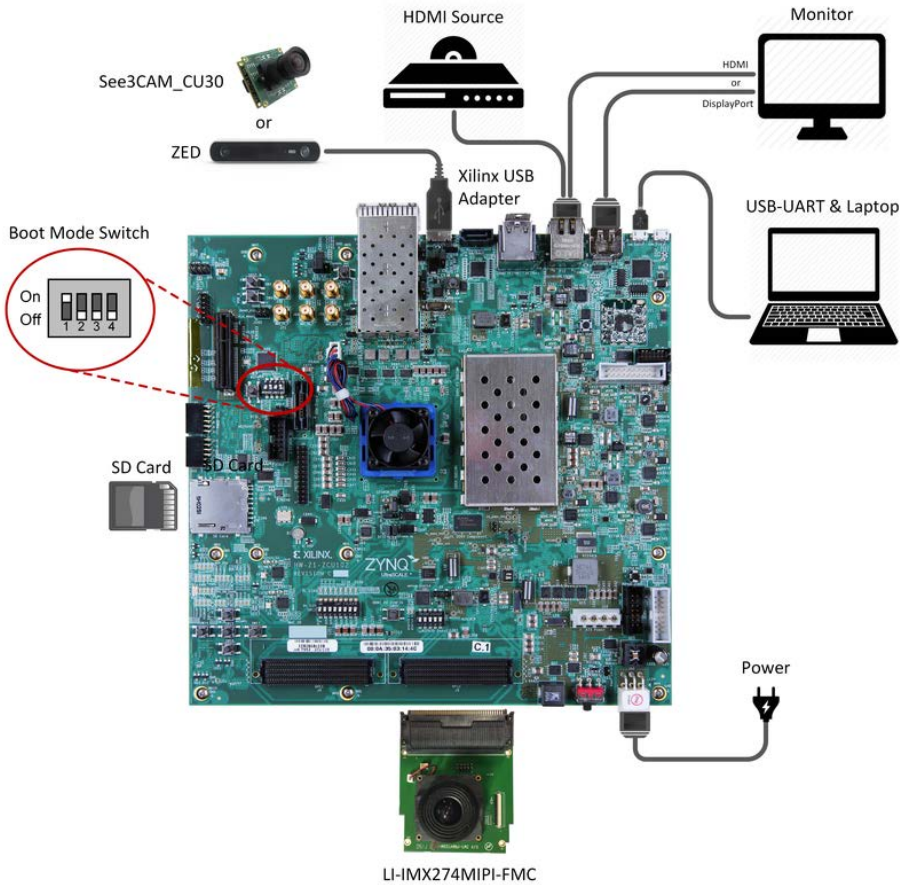


**Figure 1: Xilinx ZCU102 Evaluation Board**

## *Preparing the ZCU104 Evaluation Board*

The Xilinx ZCU104 Evaluation Kit enables you to jumpstart designs for embedded vision applications such as surveillance, Advanced Driver Assisted Systems (ADAS), machine vision, Augmented Reality (AR), drones, and medical imaging. For more information about the Xilinx Zynq UltraScale+ MPSoC ZCU104 evaluation board, refer to the ZCU104 Evaluation Kit website. The main peripheral and connection interfaces for ZCU104 are shown in the following figure.

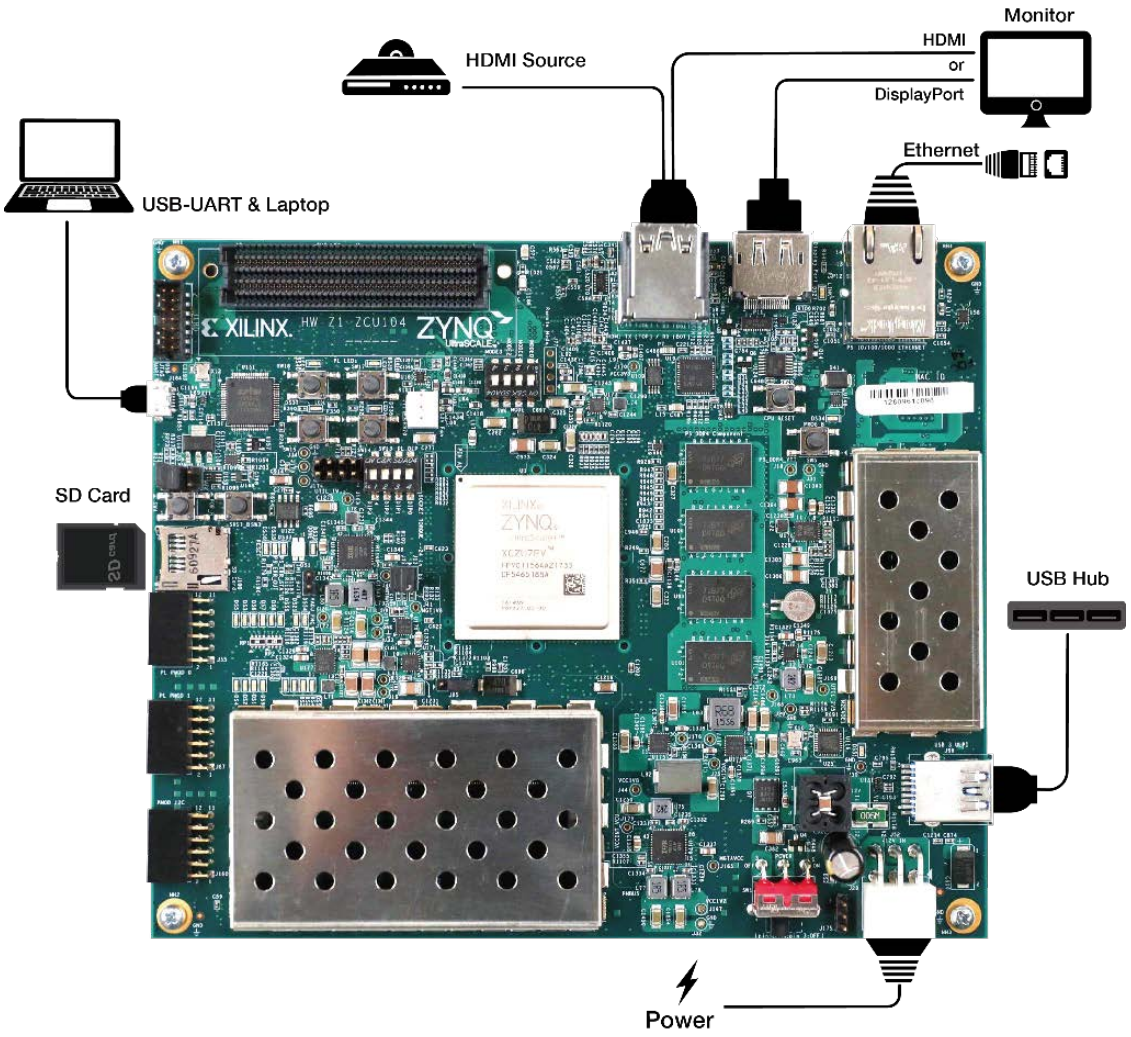


**Figure 2: Xilinx ZCU104 Evaluation Board**

# Setting Up the Software Environment

## *Installing SDSoC*

See the Xilinx SDSoC Development Environment website to download and install the SDSoC version 2018.3 environment.

## *Installing the reVISION Platform for DNNDK*

To enable DNNDK applications for the SDSoC development environment, download and install the reVISION platforms created for DNNDK. The platforms for Xilinx ZCU102 and ZCU104 are available for download.

**NOTE:** The standard reVISION platform for ZCU102 and ZCU104 do not support the running of DNNDK applications because RGB mixer is not available.

The reVISION platform for DNNDK is modified based on the official reVISION platform. The following two figures show The Video Mixer setting in official reVISION platform and the modified configuration of Video Mixer, respectively. The main difference is that the Video Format of some layers changes from "YUYV8" to "BGR8." The device tree file is modified accordingly.

**NOTE**: The official samples cannot run on the modified reVISION platform.

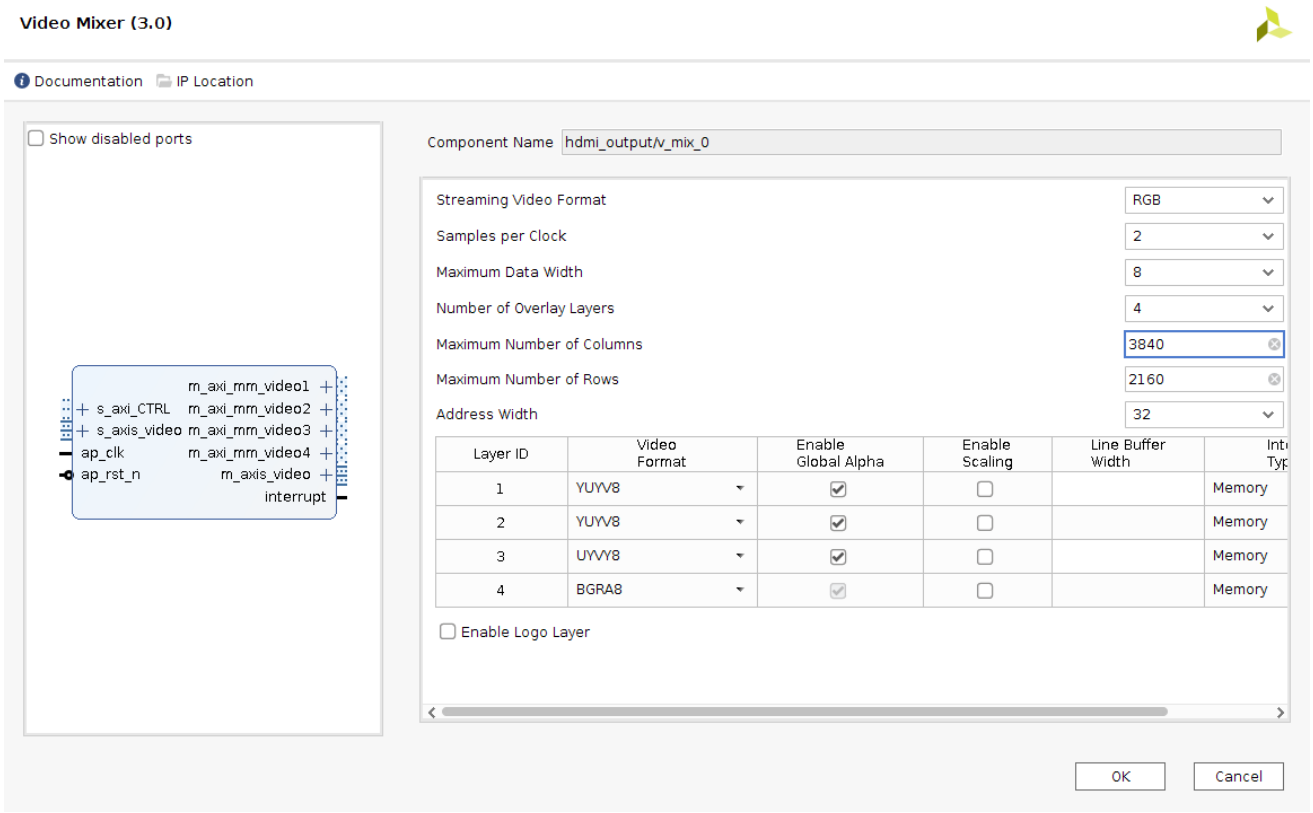


**Figure 3: Original Configuration of the reVISION Platform**

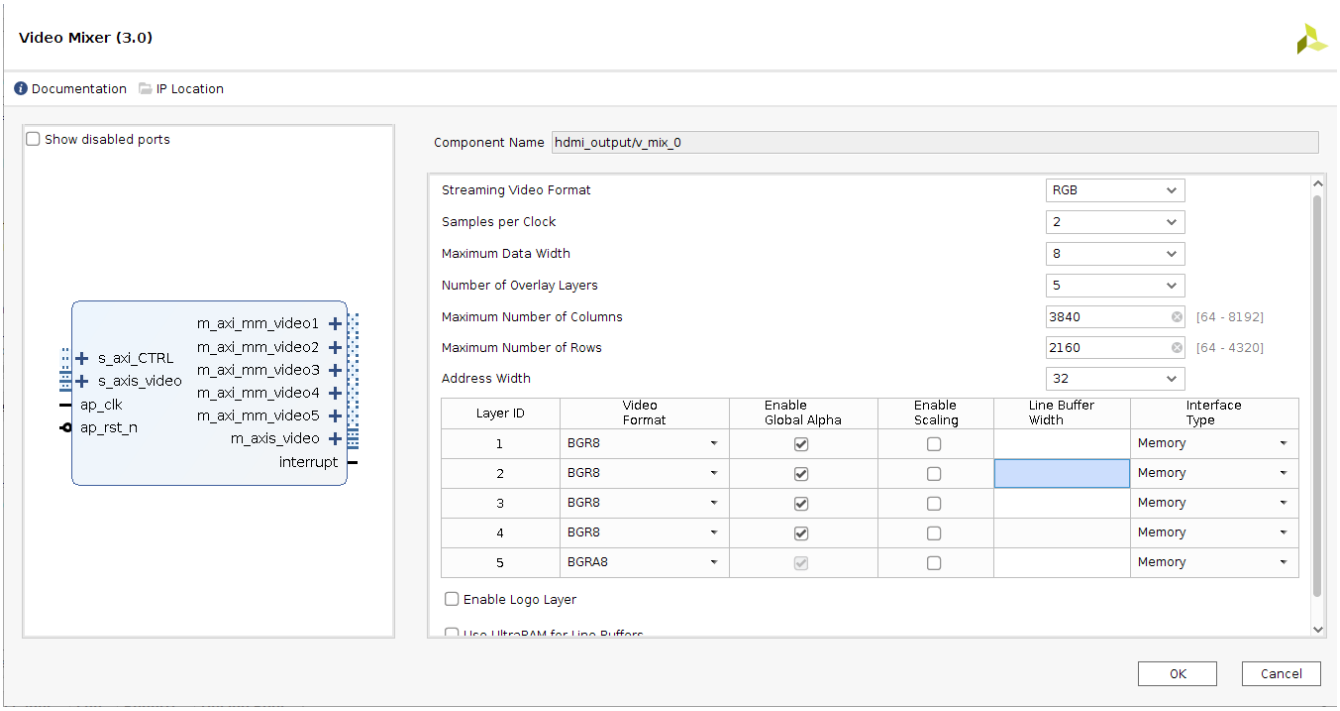


**Figure 4: Modified Configuration**

## *Installing the DNNDK Package*

The Deep Compression Tool (DECENT) and the Deep Neural Network Compiler (DNNC) tools are included in the *host_x86* folder of the DeePhi package. There are four prebuilt editions for different system environments. The `install.sh` script automatically inspects the system environment and installs the appropriate edition in the case that you have root authority. Otherwise, you can manually select the correct edition.

For the prerequisite of these two tools, refer to the *DNNDK User Guide* (UG1327) for more detailed information.

In this chapter, the example of ResNet-50 image classification illustrates the typical workflow for developing DNNDK applications using the SDSoC™ development environment.

The necessary components needed to run a DNNDK application are as follows:

- Hardware
- System
- Libraries
- Application

These components are described in the subsequent sections of this chapter.

## Hardware Requirements

The hardware necessary for running a DNNDK application is the target board you are working on, such as the Xilinx® ZCU102 evaluation board.

## System Requirements

The SDSoC development environment, version 2018.3, is necessary for running a DNNDK application. You must also download the desired platform, such as the reVISION platform for ZCU102, or the modified version based on it.

Build the `dpucore` project to create the following three files:

- `BOOT.BIN`
- `image.ub`
- `libdpucore.so`

Building the `dpucore` project is described in Chapter 4: DNNDK Workspace. For quick start, you can use the pre-built binaries in the appropriate Package.

# Libraries

There are two kinds of libraries that are related to DPU in this application:

- DNNDK library: including `libn2cube.so` and `libdputils.so`
- ResNet50 neural network model: `libdpumodelresnet50.so`

Use the following steps to generate the model library by yourself.

## *Prerequisites*

- Floating-point ResNet50 model file for Caffe (`resnet50.prototxt`)
- Pre-trained weights file for ResNet50 with Caffe (`resnet50.caffemodel`)
- calibration dataset

  Extract 100 to 1000 images from ImageNet training dataset, and change the path in `resnet50.prototxt` accordingly.

## *Steps*

1. Convert ResNet-50 floating-point model to quantized INT8 model with the Deep Compression Tool (DECENT).

   ```
   decent     quantize                              \
              -model float.prototxt                 \
              -weights float.caffemodel             \
              -output_dir decent_output             \
               -method 1
   ```

   Files `deploy.prototxt` and `deploy.caffemodel` are generated.

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

2. Generate the model file for DPU from quantized model with the Deep Neural Network Compiler (DNNC) compiler.

```
dnnc    --prototxt=deploy.prototxt                  \
        --caffemodel=deploy.caffemodel              \
        --output_dir=dnnc_output                    \
        --net_name=resnet50                         \
        --dpu=4096FA                                \
        --mode=debug                                \
        --cpu_arch=arm64                            \
        --abi=0
```

   Files `resnet50_0.elf` and `resnet50_2.elf` are generated after compilation.

   **Note**: The specify the option "`--abi=0`" to DNNC compiler. Refer to the *DNNDK User Guide* (UG1327) for more detailed information.

3. Convert the model file to a shared library.

```
aarch64-linux-gnu-gcc -fPIC -shared \
        dpu_resnet50_*.elf -o libdpumodelresnet50.so
```

# Application

After building the resnet50 project successfully, the classification executable program *resnet50.elf* is created. Also, you can use the ready-to-use binaries in the package for a quick start.

Now, you have all the dependencies to run the classification application on DPU. Run this program following the steps below:

1. Prepare a SD card that is formatted to FAT32 filesystem.
2. Copy the `BOOT.bin`, `image.ub`, and `libdpucore.so` files to the root folder of your SD card.
3. Create a `lib` folder in your SD card and copy the `libn2cube.so`, `libdputils.so`, and `libdpumodelresnet50.so` files to the `lib` folder.
4. Copy the application binary `resnet50.elf` as well as the test `image` folder in the project to the SD card.
5. Boot up the ZCU102 board with the SD card.
6. Connect to the board through a serial port with a terminal software and change the current directory of your console to the folder of your application `resnet50.elf` file (the SD card is mounted on `/media/card`), then run the program with `./resnet50.elf`. The top five classification results for each picture appears in the image folder.

This chapter describes how to create a firmware with DPU C-callable IP for a target board, and how to build some DPU applications based on the package you already have.

# Workspace Overview

Do the following to open a workspace.

1. Start the SDx™ GUI.
2. Select **File > Switch Workspace** and then select your workspace folder.
3. View the available projects in the Project Explorer tab.

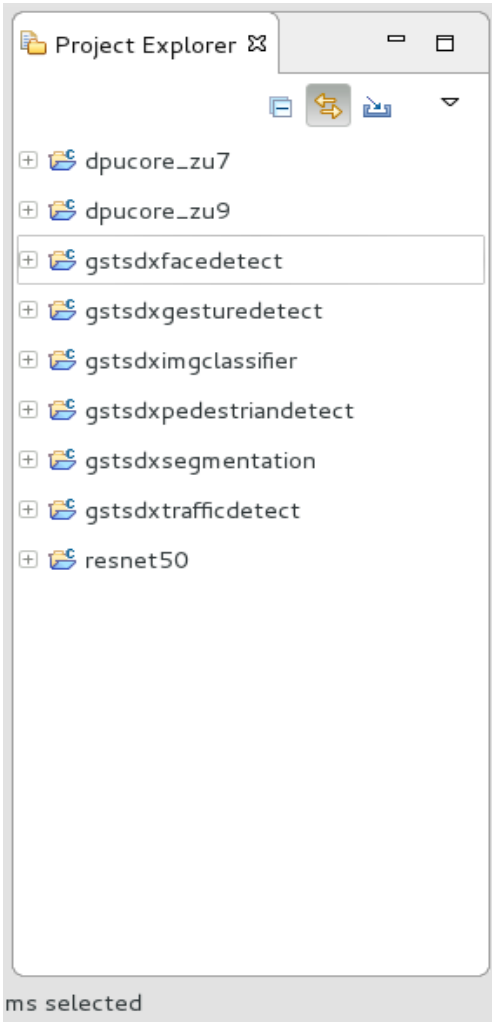


**Figure 5: Workspace Overview**

The `dpucore_zu7` and `dpucore_zu9` projects are hardware projects aimed to integrate the DPU C-callable IP. The `gstsdx*` projects generate some gstreamer detection plugins. The `resnet50` is a standalone application that uses the resnet50 model to classification.

### *Importing Projects to Your Own Workspace*

If you prefer to work in your own workspace or start from a clean workspace, you can import these projects into your own by doing the following:

1. Select **File > Import**.
2. Click **Existing Projects into Workspace**.
3. Click **Next**.
4. Browse to the target directory and select your projects.
5. Click **Finish**.
6. Manually copy the **include** and **lib** folders in the provided workspace to your own workspace.

## Hardware Project

This project `dpucore_zu9` is used to create firmware with DPU for a target board. To build the project:

1. Prepare a DPU C-callable library (`libdpu130_b4096_zu9.a`), and target platform.

   **Note**: The first time you switch into this workspace or import this project to a clean workspace, you should update the platform setting, which is described later in this document.
2. Right-click the project, and then click **Build Project**.

**Note**: It could take hours to build the project, depending on the platform you select.

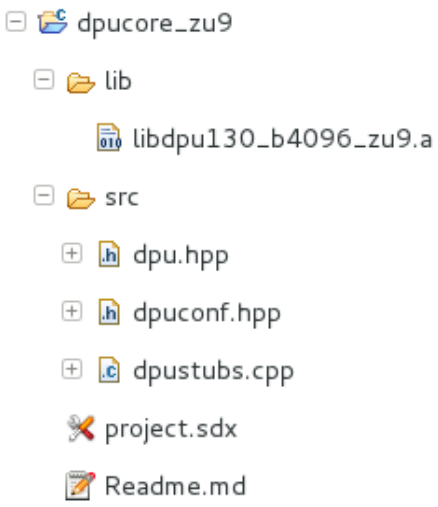


**Figure 6: dpucore_zu9 Project Structure**

You get three files after successfully building the project:

- `BOOT.bin`
- `image.ub`
- `libdpucore_zu9.so`

Use these files to boot up your target board. The dynamic library `liddpucore_zu9.so` encapsulates some lower read and write APIs of DPU IP for DNNDK.

Notes:

- The DNNDK libraries are linked to `libdpucore.so`. You must rename the generated shared library `libdpucore_zu9.so` to `libdpucore.so`.
- You can modify the macro, `DPU_CORE_NUM` in `dpuconf.hpp` to enable multiple DPU instances before building the project, but ensure that your target FPGA device has enough resources.
- If you must change the platform, double-click the `project.sdx` file in the Project Explorer and select your application in the configuration tab.

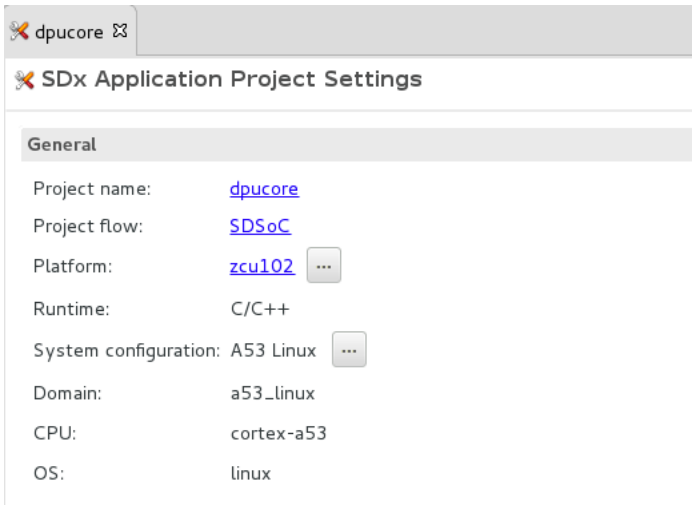


**Figure 7: dpucore Configuration Tab**

If the target platform is not in the list, click the **Add** button + to add it manually.

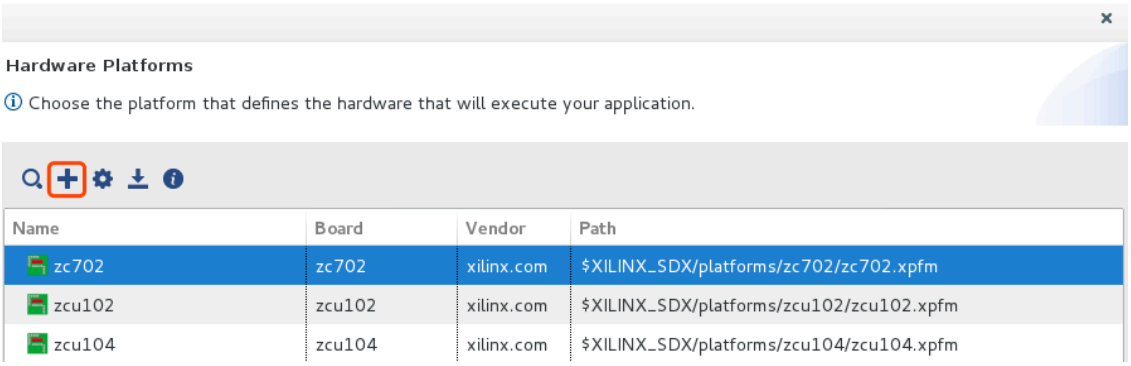


**Figure 8: Platform Management Dialog Box**

# Gstreamer Plugin Project

There are several gstreamer plugin projects in this workspace. Each project implements a practical function. For example, the `gstsdximgclassifier` project generates a plugin for image classification.

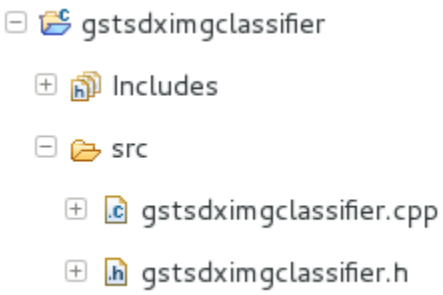


**Figure 9: Project Structure of gstsdximgclassifier**

To build a gstreamer plugin project:

1. Right-click the project and select **Properties**.
2. Navigate to the **Environment** tree node and modify the **SYSROOT** variable according to your system. Usually, this variable can be found in your sub folder of your platform.

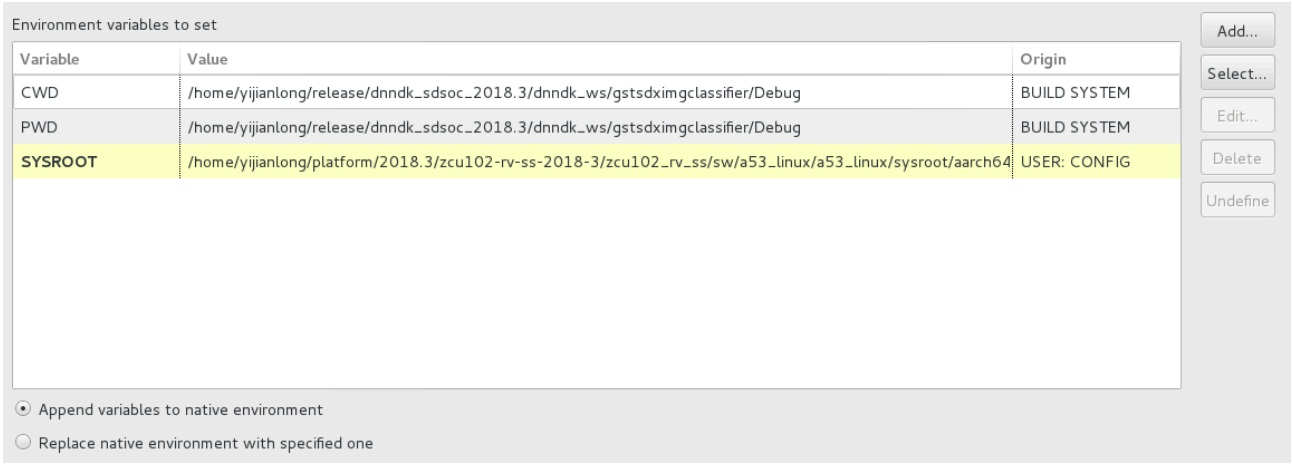


**Figure 10: gstsdximgclassifier Property Page**

3. After configuration, right-click the project in the Project Explorer and then select in the **Build Projects** menu.

   A dynamic library called `libgstsdximgclassifier.so` is generated. You can use it in a gstreamer pipeline to perform a classification task for each frame of an input stream.

# Standalone Application Project

The project `resnet50` is a standalone application that implements a classifier using the resnet50 model. The project structure is as follows:

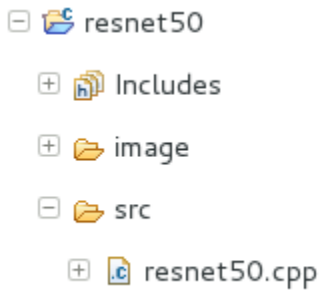


**Figure 11: Project Structure of Resnet50**

Use these steps to build this project.

1. Modify the `SYSROOT` variable according to your system just as you did for the gstreamer plugin project in the previous section.
2. Right-click the project in the Project Explorer and select the **Build Projects**.

   An executable file "`resnet50`" is generated. This application reads each of the pictures in the "`image`" folder and outputs its classifier label and probability.

This chapter introduces the typical step to run the applications and describes the demos included in the workspace.

## Steps for Running an Application

To run the standalone classification program `resnet50.elf` on a target board, do the following.

1. Copy your system files and the application binary to an SD card, putting the necessary libraries in the `lib` folder of the SD card.

   **Note**: The SD card will be mounted to `/media/card automatically.` The `/media/card/lib` is included in the system's link path.

2. Execute the binary `resnet50.elf`. This processes each picture in the `image` folder and outputs the top five classification results.

Other DPU samples are compiled in the form of a gstreamer plugin and you must launch a gstreamer pipeline to invoke it. It is similar to that of running the standalone program, but something you must pay attention to. The typical steps are as follows:

1. Prepare a SD card that is formatted to a FAT32 filesystem.
2. Copy the system files (`BOOT.bin`, `image.ub,` and `libdpucore.so`) to the `root` folder of the SD card.
3. Copy the libraries to the `lib` folder of the SD card.
   - DNNDK libraries: `libn2cube.so` and `libdputils.so.`
   - DPU model libraries: `libdpumodel*.so.`
   - Gstreamer detection plugin libraries: `libgstsdx*.so.`

   **Notes:**

   - Refer to Table 1. Description of Prebuilt Libraries in Chapter 2: Prepare the Environment for a description of each library that explains the dependency of each sample.
   - Alternatively, you can copy the entire lib folder to the SD card.

4. Copy the input files and necessary scripts to your SD card.

   **Notes:**

   - These detection plugins take raw `.bgr` format video as input, but the video files in the DeePhi package are encoded `.mp4 files`. The gstreamer pipeline in the scripts use `multifilesrc` to process input videos. If the input file is too large, you must split it into smaller pieces (several megabytes for each file is appropriate), and use a wildcard to match all the files. You can copy the video folder to your server, and then use the `convert.sh` script under the `video` folder to do this conversion.

     Then you can simply copy the entire video folder from your server to the SD card.
     You should have already installed ffmpeg and gstreamer tools on your server before running *`convert.sh`*. If not, you can install it with the following commands.

     ```
     sudo add-apt-repository ppa:kirillshkrogalev/ffmpeg-next
     sudo apt-get update
     sudo apt-get install ffmpeg
     ```

   - These are several ready-to-use gstreamer scripts in the "`deephi_prebuilt`" folder of the DeePhi package. You can copy the entire scripts folder to the SD card.

5. Boot up the target board with your SD card.

6. Launch a gstreamer pipeline manually, or execute the ready-to-use scripts.

   The following code snippet is typical pipeline:

   ```
   gst-launch-1.0 \
       multifilesrc location=/media/card/video/test_1%04d.bgr loop=true ! \
       videoparse width=640 height=480 format=bgr framerate=30 ! \
       sdxfacedetect need-cma-input=false ! \
   fpsdisplaysink video-sink=" kmssink sync=false plane-id=29 bus-
   id="b00c0000.v_mix" render-rectangle=\"<0,0,640,480 >\" " text-overlay=true
   sync=false
   ```

   You must modify some of the properties to keep the pipeline agreed with those in your system.

   **Notes:**

   - You must set up your USB camera and modify the `device` property of v4l2src plugin according to your system before running the `run_facedetect.sh` script.
   - The reVISION platform for DNNDK supports four BGR planes of HDMI output, the plane-ids are 29, 30, 31, and 32. You can inspect the plane-ids with the command "`modetest -D b00c0000.v_mix`". The ready-to-use scripts are only for one channel detection. You can modify the pipeline if you want to launch multiple channels detection.

# Application Descriptions

There are five other gstreamer plugin projects in the DeePhi package. You can import these projects by following the instructions described in Chapter 4: DNNDK Workspace.

## *Face Detection*

This project detects faces in each frame. It uses densebox neural network model for detection.

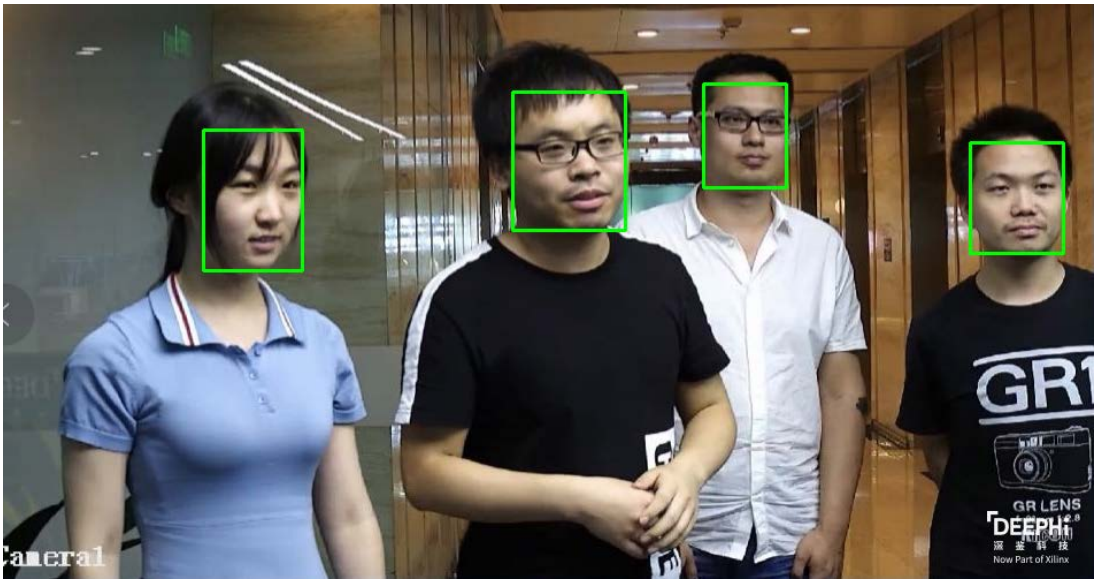


**Figure 12: Face Detection**

## *Traffic Detection*

This project detects motor vehicle, non-motor vehicle, and pedestrian traffic in each frame. It uses SSD as its object detecting algorithm.

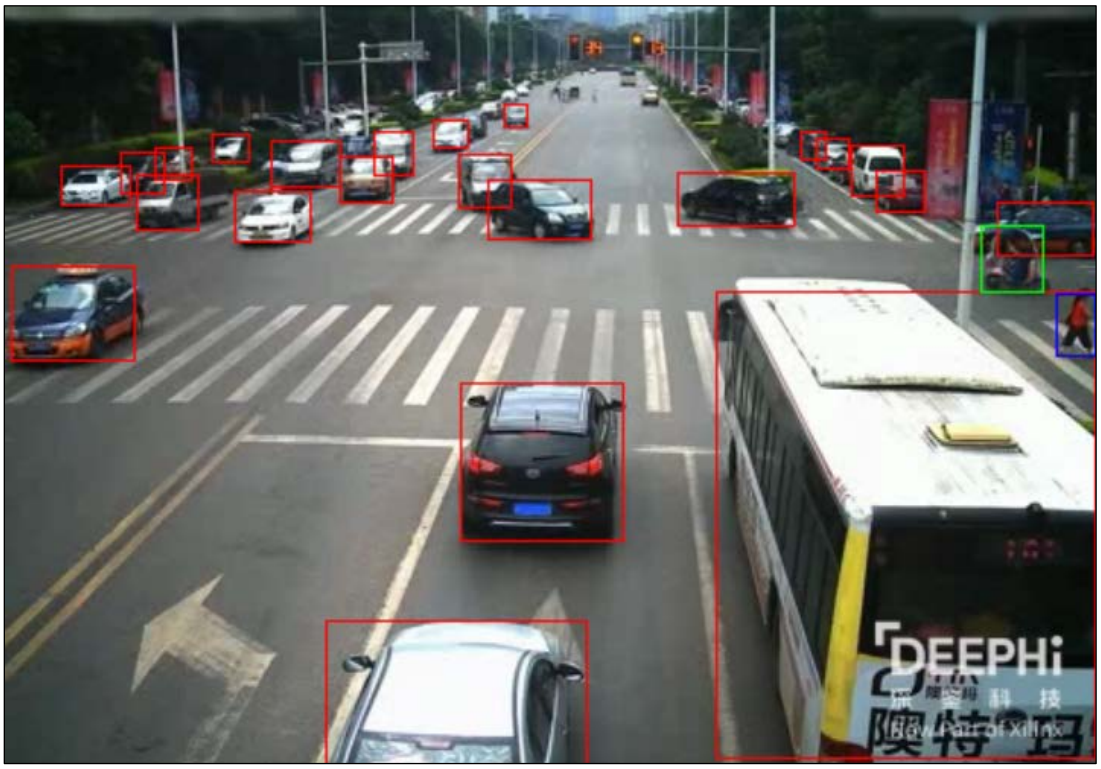


**Figure 13: Traffic Detection**

## *Gesture Detection Project*

This project recognizes the major joints of each person in a frame. It uses SSD for human detection, and performs a joint recognition for each detected person.


**Figure 14: Gesture Detection Project**

## *Pedestrian Detection Project*

This project detects pedestrians in each frame. It uses YOLO as its object detecting algorithm.

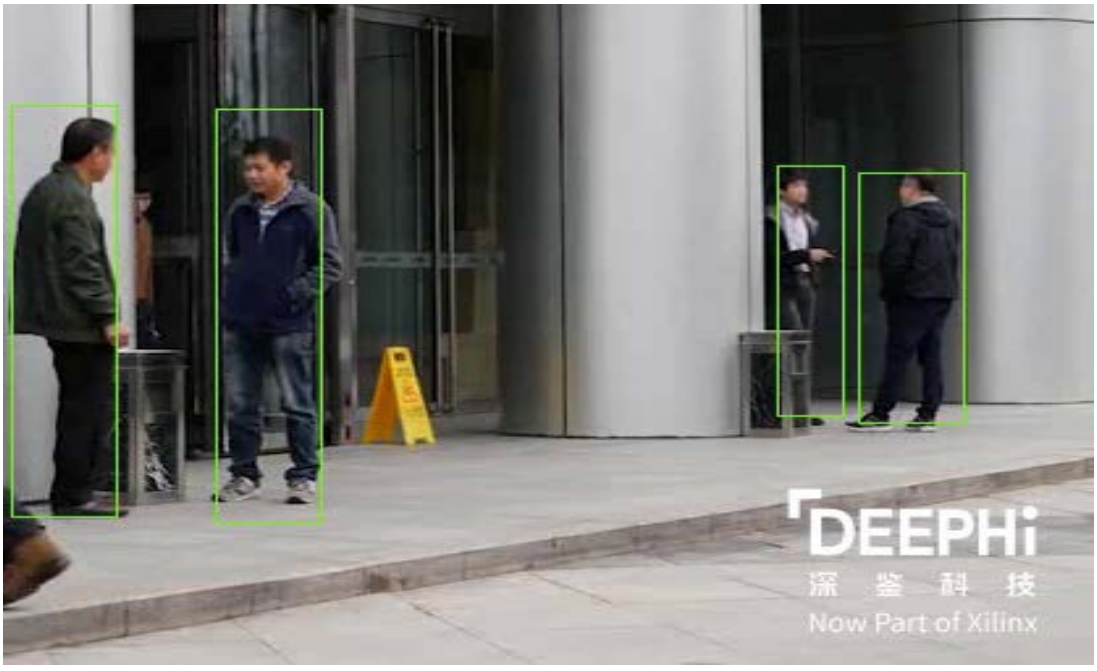


**Figure 15: Pedestrian Detection**

## *Segmentation Project*

This project performs a scene segmentation for each frame.


**Figure 16: Scene Segmentation Project**

# 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.