# Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Description</th>
</tr>
</thead>
</table>
| December 2018 | 0.95     | R5 release  
  • Updated bitstreams  
  • Updated BSP  
  • Added information about using USB-Blaster to program the R5 BSP  
  • Removed troubleshooting and workarounds that are no longer needed |
| November 2018 | 1.0      | R4 release  
  • Updated the kernel version from 4.13 to 4.14/4.15  
  • Updated the R4 BSP and OpenVINO links  
  • Added MXNet* and TensorFlow*  
  • Removed troubleshooting and workarounds that are no longer needed |
| October 2018  | 1.0      | R3 Early Access release |
## Contents

**Revision History**..................................................................................................................3

### 1.0 Introduction................................................................................................................. 7
  1.1 Intended Audience..................................................................................................7
  1.2 Operating System Requirements........................................................................7
  1.3 Software Requirements.......................................................................................7
  1.4 Document Conventions.......................................................................................8
  1.5 Terminology.......................................................................................................9

### 2.0 About this FPGA Vision Accelerator.............................................................................10
  2.1 Hardware Specifications....................................................................................10
  2.2 OpenVINO™ Support........................................................................................13
    2.2.1 Model Optimizer....................................................................................16
    2.2.2 Inference Engine....................................................................................16

### 3.0 Install and Configure OpenVINO™...........................................................................18
  3.1 Install Intel® Quartus® Prime Lite Edition software ...........................................18
  3.2 Install the OpenVINO™ Core Components.....................................................18
  3.3 Set the Environment Variables.........................................................................19

### 4.0 Install and Configure the Intel® Vision Accelerator Design with an Intel® Arria®
10 FPGA Software........................................................................................................21
  4.1 Install the Intel® Vision Accelerator Design with an Intel® Arria® 10 FPGA Board
    Support Package (BSP) for the OpenVINO™ Toolkit, R5..................................21
  4.2 Initialize the Intel® Vision Accelerator Design with an Intel® Arria® 10 FPGA.....24
  4.3 Verify Your Configuration...............................................................................26
  4.4 Intel® DLIA Bitstreams....................................................................................27

### 5.0 Configure and Use the Model Optimizer....................................................................28
  5.1 Configure the Model Optimizer........................................................................28
  5.2 Use the Model Optimizer..................................................................................29

### 6.0 Build the Sample Applications....................................................................................31

### 7.0 Use the Sample Applications....................................................................................32
  7.1 `classification_async_Sample` with Maximum Optimization..........................32
  7.2 `object_detection_ssd`....................................................................................34
  7.3 Other Samples.................................................................................................34

### 8.0 Using Multiple FPGA Devices....................................................................................36
## Figures

1. PCIe Interface Card
2. PCIe Card Layout
3. PCIe Card Connectors
4. PCIe Card Block Diagram
5. PCIe Card Dimensions
6. Intel® DLIA Software Stack
7. How the Model Optimizer Works
8. OpenVINO™ Deployment Workflow
Tables

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Intel® Vision Accelerator Design with an Intel® Arria® 10 FPGA Specifications</td>
<td>13</td>
</tr>
</tbody>
</table>
1.0 Introduction

AI impacts every aspect of our daily lives and is expected to be the next computing wave to transform the way businesses operate. AI, by nature, is processor-intensive and complex. In response to the processor-intensive nature of AI, Intel provides an FPGA hardware-acceleration solution that can handle challenging deep learning models at unprecedented levels of performance and flexibility.

The Linux version of the OpenVINO™ toolkit that includes FPGA supports the Intel® Vision Accelerator Design with an Intel® Arria® 10 FPGA. Therefore, this document includes information about key OpenVINO™ components, including the Model Optimizer and the Inference Engine.

Note: In this document, the Intel® Vision Accelerator Design with an Intel® Arria® 10 FPGA is sometimes referred to as the "vision accelerator."

Note: You must follow all instructions in this guide before you can use the sample applications.

1.1 Intended Audience

- System engineers
- Platform architects
- Software developers

1.2 Operating System Requirements

The host operating system and Linux kernel below are validated and recommended.

- Ubuntu* 16.04.3 LTS, 64-bit
- Linux Kernel 4.15

To check your kernel version:

```
cat /proc/version
```

1.3 Software Requirements

You must provide:
1.4 Document Conventions

<table>
<thead>
<tr>
<th>Convention</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>This font</td>
<td>File names, commands, keywords Long command lines sometimes wrap to multiple lines in documents. Type your command on one line unless otherwise specified.</td>
</tr>
<tr>
<td>#</td>
<td>Type the command as root</td>
</tr>
<tr>
<td>$</td>
<td>Type the command as a user</td>
</tr>
<tr>
<td>&lt;variable&gt;</td>
<td>Replace the text between the brackets with a value. Do not type the brackets.</td>
</tr>
<tr>
<td><strong>Bold text</strong></td>
<td>Click an option on a screen</td>
</tr>
</tbody>
</table>
# Terminology

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application programming interface</td>
</tr>
<tr>
<td>Caffe*</td>
<td>The computer vision framework supported by the vision accelerator discussed in this document</td>
</tr>
<tr>
<td>CNN</td>
<td>Convolutional neural network</td>
</tr>
<tr>
<td>DSS</td>
<td>Digital Surveillance Solution</td>
</tr>
<tr>
<td>Intermediate Representation</td>
<td>A set of two files that result from using the Model Optimizer. The Intermediate Representation files are required as input to the Inference Engine.</td>
</tr>
<tr>
<td>IR</td>
<td>See Intermediate Representation</td>
</tr>
<tr>
<td>Model Optimizer</td>
<td>This command-line tool optimizes a model that was trained with a supported framework. For information, see <a href="https://software.intel.com/en-us/articles/OpenVINO-ModelOptimizer">https://software.intel.com/en-us/articles/OpenVINO-ModelOptimizer</a>.</td>
</tr>
<tr>
<td>OpenVINO™</td>
<td>A free toolkit used to optimize and perform inference on pretrained models. OpenVINO™ includes the Intel® DL Deployment Toolkit. OpenVINO™ supports this FPGA vision accelerator. Learn more and download OpenVINO from the <a href="https://software.intel.com/en-us/articles/OpenVINO-ModelOptimizer">OpenVINO Web site</a>.</td>
</tr>
<tr>
<td>Pretrained model</td>
<td>A model that was created by and trained with a framework.</td>
</tr>
<tr>
<td>prototxt file</td>
<td>Protocol buffer definition file. Frameworks, such as Caffe*, MXNet* and TensorFlow*, use this file to define the model architecture.</td>
</tr>
</tbody>
</table>
2.0 About this FPGA Vision Accelerator

The Intel® Vision Accelerator Design with an Intel® Arria® 10 FPGA has a small form factor. It is a low power consumption product, and is a low latency FPGA-based AI edge computing solution.

The Intel® Vision Accelerator Design with Intel® Arria® 10 FPGA:

- Provides outstanding performance/power/price per inference:
  - Energy-efficient inference
  - Scalable throughput gains that are better than the processor alone
  - Lower total cost of ownership for high-throughput systems
- Fits in the Intel® Xeon® processor infrastructure:
  - Multiple 1U and 2U server system options
  - PCIe® Gen3 x8 enables fast communication between the host computer and Intel® DLIA adapter
- Is a flexible and portable software architecture:
  - Accelerates time to market by simplifying deployment with a turnkey solution and software ecosystem
  - Supports CPU fallback of CNN primitives that the FPGA does not implement
  - Unified user experience and code that can be ported across Intel product families

The FPGA vision accelerator also consists of an Intel® Deep Learning Inference Accelerator (Intel® DLIA), providing a way for you to develop applications and solutions through the Linux FPGA version of the OpenVINO™ toolkit.

2.1 Hardware Specifications

The Intel® Vision Accelerator Design with an Intel® Arria® 10 FPGA is a standard half-height, half-length and single-width PCIe Gen 3 x8 add-in card. It contains the Intel® Deep Learning Inference Accelerator (Intel® DLIA) that is preprogrammed to accelerate the convolution neural network primitives with optimized performance. The add-in card communicates with the host system through the PCIe interface.
**Figure 1.** PCIe Interface Card

**Figure 2.** PCIe Card Layout

**Figure 3.** PCIe Card Connectors

<table>
<thead>
<tr>
<th>Callout Letter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Power select: Power via PCIe or on-board external 6-pin VGA port</td>
</tr>
<tr>
<td>B</td>
<td>Programming interface: FPGA JTAG</td>
</tr>
</tbody>
</table>

*continued...*
<table>
<thead>
<tr>
<th>Callout Letter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Card-ID rotary-switch: Rotate the switch to select from 0-9 and A-F</td>
</tr>
<tr>
<td>D</td>
<td>Seven-segment LED: Shows the card-ID</td>
</tr>
<tr>
<td>E</td>
<td>Micro-USB port: Use to update the firmware with a standard USB cable</td>
</tr>
</tbody>
</table>

**Figure 4. PCIe Card Block Diagram**

![PCIe Card Block Diagram](image)

**Figure 5. PCIe Card Dimensions**

![PCIe Card Dimensions](image)
Table 1. **Intel® Vision Accelerator Design with an Intel® Arria® 10 FPGA Specifications**

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main FPGA</td>
<td>Intel® Arria® 10 1150 GX FPGAs, delivering up to 1.4 TFLOPs</td>
</tr>
<tr>
<td>Memory</td>
<td>DDR4 2400 Hz 8 GB on board</td>
</tr>
<tr>
<td>Dataplane interface</td>
<td>PCI Express® x8 Compliant with PCI Express specification V3.0</td>
</tr>
<tr>
<td>Power consumption</td>
<td>38W - 42 W (maximum at 48 W)</td>
</tr>
<tr>
<td>Operating temperature</td>
<td>0° C - 65° C ambient temperature</td>
</tr>
<tr>
<td>Operating humidity</td>
<td>5% ~ 90%</td>
</tr>
<tr>
<td>Cooler</td>
<td>Active fan</td>
</tr>
<tr>
<td>Power connector</td>
<td>12 V external power (optional)</td>
</tr>
<tr>
<td>Dimension</td>
<td>Standard half-height, half-length, double-width PCIe</td>
</tr>
<tr>
<td>DIP switch / LED indicator</td>
<td>Identify card number</td>
</tr>
</tbody>
</table>

2.2 **OpenVINO™ Support**

*Note:* You can use this vision accelerator with the OpenVINO™ toolkit, but not with the Intel® DL Deployment Toolkit.

OpenVINO™ is a free software development package that quickly deploys CNN-based solutions. OpenVINO™ extends computer vision workloads across Intel hardware, maximizing its performance.

The Linux version of the OpenVINO™ toolkit that includes FPGA support includes the Model Optimizer utility that accepts pretrained models and prototxt files from Caffe framework. The Convolution Neural Network nodes are then accelerated in the Intel DLIA while the rest of the vision pipelines are executed in the host system.
OpenVINO™ has two key components:

- Model Optimizer
- Inference Engine

The OpenVINO™ toolkit uses a utility called the Model Optimizer that accepts pretrained models and prototxt files from several frameworks, including Caffe®. The Convolutional Neural Network (CNN) nodes are accelerated in the Intel DLIA while the rest of the vision pipelines are executed in the host system.

The Model Optimizer utility generates Intermediate Representation (IR) files that a second OpenVINO™ utility, the Inference Engine, processes. You use the Inference Engine to apply the model to your applications, including classification, object detection, feature segmentation, and security surveillance.

The IR files generated by the Model Optimizer consist of a topology file and a pretrained weight file that are loaded into the Inference Engine runtime.
You can use the Intel® Vision Accelerator Design with an Intel® Arria® 10 FPGA with the Linux version of OpenVINO™ that supports FPGA. This version of OpenVINO™ provides:

- Convolution Neural Network based deep learning inference in the edge.
- Heterogenous execution across Intel CPUs and Intel FPGAs.
- Pre-compiled FPGA bitstream samples for this vision accelerator.
- Fast time-to-market with its easy-to-use computer vision libraries and pre-optimized kernels.
- Optimized calls for computer vision standards inclusive of OpenCV® and OpenCL™.
OpenVINO™ Deployment Workflow

Figure 8.

For more information about using OpenVINO™, see the OpenVINO Web site.

2.2.1 Model Optimizer

The Model Optimizer is the one of two key OpenVINO components. It is a cross-platform command-line tool that facilitates the transition of your trained model from the training environment to CNN deployment. The tool supports pre-trained Caffe models and provides a flexible extension mechanism for processing custom layers.

The Model Optimizer optimizes and converts trained models into Intermediate Representation (IR) files for use by the Inference Engine, the other key OpenVINO component.

Later in this document, you will use the Model Optimizer to prepare your pre-trained Caffe model for the Inference Engine.

For more information about the Deep Learning Mode Optimizer, see the Model Optimizer Developer Guide.

2.2.2 Inference Engine

The Inference Engine is the second key component of the Intel® DL Deployment Toolkit and the OpenVINO™ toolkit. The Inference Engine is also a cross-platform command-line tool. The Inference Engine offers a unified API for supported Intel® platforms that might have different inference low-level APIs.

The Inference Engine executes different layers on different target platforms. It uses a unified API to work on IR files and optimize inference with application logic to deploy deep learning solutions.
The Inference Engine FPGA plugin can load different networks on multiple FPGA devices.

As input, the Inference Engine takes a deep learning model in the IR format, generated by the Model Optimizer. The core `libinference_engine.so` library implements loading and parsing the IR model and triggers inference using a specified plugin. The core library has the following APIs:

<table>
<thead>
<tr>
<th>API</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InferenceEngine::IInferencePlugin</td>
<td>Main plugin interface. Every Inference Engine plugin implements this method. This can be used through the InferenceEngine::InferenceEnginePluginPtr instance.</td>
</tr>
<tr>
<td>InferenceEngine::pluginDispatcher</td>
<td>This class finds a suitable plug-in for specified devices.</td>
</tr>
<tr>
<td>InferenceEngine::CNNNetReader</td>
<td></td>
</tr>
<tr>
<td>InferenceEngine::CNNNetwork</td>
<td></td>
</tr>
<tr>
<td>InferenceEngine::Blob,InferenceEngine::TBlob</td>
<td></td>
</tr>
<tr>
<td>InferenceEngine::BlobMap</td>
<td></td>
</tr>
<tr>
<td>InferenceEngine::InputInfo</td>
<td></td>
</tr>
<tr>
<td>InferenceEngine::InputsDataMap</td>
<td></td>
</tr>
</tbody>
</table>

For more information about the Inference Engine, see the Inference Engine Developers Guide.
3.0 Install and Configure OpenVINO™

If you have not done so already, download the OpenVINO™ toolkit, R5. Be sure to download the Linux version that includes FPGA support.

Before beginning the installation, check your Linux kernel version:

```
cat /proc/version
```

Make sure you are using Linux kernel version 4.14 or above.

3.1 Install Intel® Quartus® Prime Lite Edition software

1. Download the Intel® Quartus® Prime Lite Edition software, version 17.1.1
2. Go to the directory to which you downloaded the software. The default directory is ~/Downloads, and the default filename is . If you used a different directory or renamed the file, change the following instructions according to your naming conventions.
3. Run the setup file:

```
./QuartusProProgrammerSetup-17.1.1.273-linux.run
```

Intel® Quartus® Prime Lite Edition software is downloaded and installed. Continue with the next section to install the OpenVINO™ core components.

3.2 Install the OpenVINO™ Core Components

1. If you haven't already done so, download the OpenVINO™ toolkit.
2. Go to the directory in which you downloaded the file. These steps assume the file is in ~/Downloads. If you used a different directory, change the rest of these instructions to reflect the directory you used.
3. Unpack the file:

```
tar -xf l_openvino_toolkit_fpga_p_<version>.tgz
```

A directory named l_openvino_toolkit_fpga_p_<version>.

4. Run a script to download and install the external software dependencies:

```
sudo ./install_cv_sdk_dependencies.sh
```

5. Choose between installing with or without a GUI. Only the visual aspects are different between these options. Choose ONE option:
• If you want to use a GUI installation wizard to prompt you for input:
  
  sudo ./install_GUI.sh

• If you want to use command-line instructions to prompt you for input:
  
  sudo ./install.sh

6. Follow the instructions on your screen.

The base installation is complete. Continue to the next section to set the environment variables.

### 3.3 Set the Environment Variables

1. View the PCIe device on your computer:

   lspci | grep -i Altera

   Success is indicated by a response similar to:

   01:00.0 Processing accelerators: Altera Corporation Device 2494 (rev 01)

2. Download fpga_support_files.tgz from the Intel Resource Center. The contents of this .tgz file makes sure the FPGA card and OpenVino™ work correctly.

3. Go to the download directory. These instructions assume ~Downloads. If you use a different location, change the remainder of these instructions to reflect the directory you use.

4. Unpack the file:

   tar -xvzf fpga_support_files.tgz

   A directory named fpga_support_files is created.

5. Switch to superuser:

   sudo su

6. Go to the fpga_support_files directory:

   cd /home/<user>/Downloads/fpga_support_files/

7. Source setup_env.sh from fpga_support_files to set up the environment variables:

   source setup_env.sh

8. Run a script to allow OpenCL to support Ubuntu and recent kernel versions:

   ./install_openvino_fpga_dependencies.sh
Note: The OpenVINO™ environment variables are removed when you close the shell. As an option, use your preferred method to permanently set the variables.

Continue to the next section to initialize the vision accelerator.
4.0 Install and Configure the Intel® Vision Accelerator Design with an Intel® Arria® 10 FPGA Software

In the previous chapter, you installed the OpenVINO® toolkit. In this chapter, you will install and configure the Intel® Vision Accelerator Design with an Intel® Arria® 10 FPGA software.

The processes you will follow are:

- **Install the Intel® Vision Accelerator Design with an Intel® Arria® 10 FPGA Board Support Package (BSP) for the OpenVINO™ Toolkit, R5** on page 21
- **Initialize the Intel® Vision Accelerator Design with an Intel® Arria® 10 FPGA** on page 24
- **Verify Your Configuration** on page 26
- **Intel® DLIA Bitstreams** on page 27

4.1 Install the Intel® Vision Accelerator Design with an Intel® Arria® 10 FPGA Board Support Package (BSP) for the OpenVINO™ Toolkit, R5

You need an Intel® FPGA Download Cable to complete the steps below.

These steps assume you download files to ~/Downloads. If you use a different directory, change the following steps to reflect the directory you use to download files.

1. Go to ~/Downloads/fpga_support_files/config/
2. Copy the a10_1150_sgl directory to /opt/altera/aocl-prorte/aclrte-linux64/board/

```
sudo cp -rf a10_1150_sgl /opt/altera/aocl-prorte/aclrte-linux64/board/
```

3. Convert the BSP files from DOS to UNIX:

```
sudo chmod +x a10_1150_sgl

find a10_1150_sgl -type f -print0 | xargs -0 dos2unix
```

4. Connect the Intel® FPGA Download Cable between the board and the host system. See the diagram below for the connection points.
   - Connect the B end of the cable to point B on the board.
   - Connect the F end of the cable to point F on the Intel® FPGA Download Cable.
When connected, the cable assembly looks like this:

5. Source the `setup_env.sh` script from the `fpga_support_files` to set up the environment variables:

```bash
source home/<user>/Downloads/fpga_support_files/setup_env.sh
```

6. Update the Intel® FPGA Download Cable TJAG connection.

```bash
sudo cp config/51-usbblaster.rules /etc/udev/rules.d
```
7. Disconnect and reconnect the Intel® FPGA Download Cable to enable the JTAG connection.

8. Make sure the Intel® FPGA Download Cable is ready to use:

```bash
jtagconfig
```

The output is similar to:

```
1) USB-Blaster [1-6]
  02E660DD  10AX115H1(.|E2|ES)/10AX115H2/..
```

9. Download Intel® Quartus® Prime Lite Edition software

10. Install the software to /home/<user>/intelFPGA/17.1

   **Note:** Install the full Intel® Quartus® Prime Lite Edition software if you want to program `boardtest_1ddr_top.aocx` into the flash for permanent availability.

   **Note:** `l` in `lspci`, is the number 1.

11. Export the Intel® Quartus® Prime Lite Edition software environment variable:

```bash
export QUARTUS_ROOTDIR=/home/<user>/intelFPGA/17.1/quartus
```

12. Go to `/opt/altera/aocl-pro-rte/board/a10_1150_sg1/bringup`. This is the location of `boardtest_1ddr_top.aocx`

13. Use the Intel® FPGA Download Cable to program `boardtest_1ddr_top.aocx` to the flash. This makes the file permanently available, even after a power cycle:

```bash
aocl flash ac10 boardtest_1ddr_top.aocx
```

14. Reboot the host computer.

15. Make sure the host computer detects the PCIe card:

```bash
lspci | grep -i Altera
```

   **Note:** `l` in `lspci` is the letter l.

   Your output is similar to:

```
01:00.0 Processing accelerators: Altera Corporation Device 2494 (rev 01)
```

16. Export the environment script:

```bash
export AOCL_BOARD_PACKAGE_ROOT=/opt/altera/aoco-pro-rte/aclrte-linux64/board/a10-1150_sg1
```

   **Note:** The `l` in `aclrte` is the letter l

17. Source the environment script:

```bash
source /opt/altera/aocl-pro-rte/aclrte-linux64/init_opencl.sh
```
18. Install aocl:

```
aocl install
```

19. Confirm the installation:

```
aocl diagnose
```

The message DIAGNOSTIC_PASSED indicates success. After the installation is confirmed as successful, continue to Intel® DLIA Bitstreams on page 27. Do not continue to the next steps until you see this message.

4.2 **Initialize the Intel® Vision Accelerator Design with an Intel® Arria® 10 FPGA**

You must initialize the vision accelerator for the Intel® FPGA RTE for OpenCL™. This is required before you can use the Intel® FPGA plugin for the Inference Engine. Improper board initialization might damage the accelerator board.

1. Download and install the Intel® Quartus Prime Pro Edition Programmer, version 17.1.1. Make sure you get the version that includes the word "Programmer" in the name. Use the next two figures to help you identify the correct file.
2. Add Intel® Quartus® Prime Pro Programmer to your environment variables:

```
export PATH=/opt/intelFPGA_pro/17.1/qprogrammer/bin:$PATH
```

Continue to the next section to install the vision accelerator.

### 4.3 Verify Your Configuration

1. View the PCIe device:

```
lspci | grep -i Altera
```

Success is indicated by a response similar to:

```
01:00.0 Processing accelerators: Altera Corporation Device 2494 (rev 01)
```

2. Run the AOCL diagnose command from a command line prompt.

```
aocl diagnose
```
If the configuration is successful, the command returns Diagnostic PASSED.

You are ready to set up the Intel® DLIA Bitstreams.

4.4 Intel® DLIA Bitstreams

You must set up the Intel® Vision Accelerator Design with an Intel® Arria® 10 FPGA before you program the bitstreams. Make sure that the board and environment are properly configured and set up before you program the bitstream.

Pre-compiled bitstream samples for the vision accelerator are available with the OpenVINO™ toolkit.

Available bitstreams with their associated supported topologies:

- **FP11**
  - 5-0_PL1_FP11_Alexnet_GoogleNet.aocx
  - 5-0_PL1_FP11_ELU.aocx
  - 5-0_PL1_FP11_Generic.aocx
  - 5-0_PL1_FP11_MobileNet_Clamp.aocx
  - 5-0_PL1_FP11_ResNet.aocx
  - 5-0_PL1_FP11_RMNet.aocx
  - 5-0_PL1_FP11_SqueezeNet.aocx
  - 5-0_PL1_FP11_TinyYolo_SSD300.aocx
  - 5-0_PL1_FP11_VGG.aocx

- **FP16**
  - 5-0_PL1_FP16_Alexnet_GoogleNet_SqueezeNet.aocx
  - 5-0_PL1_FP16_MobileNet_Clamp.aocx
  - 5-0_PL1_FP11_ResNet_TinyYolo_ELU.aocx
  - 5-0_PL1_FP11_RMNet.aocx
  - 5-0_PL1_FP16_SSD300.aocx
  - 5-0_PL1_FP16_VGG_Generic.aocx
5.0 Configure and Use the Model Optimizer

This section provides instructions to configure the Model Optimizer either for all of the supported frameworks at the same time or to configure one or more individual frameworks. The full list of supported frameworks is Caffe, TensorFlow, MXNet, ONNX, and Kaldi. After configuring the Model Optimizer, this section goes on to help you use the tool.

Notes:

- The samples in this guide use the Caffe framework.
- Other popular public models are created by the open developer community. These are available in Model Downloader. To use them, make sure you have `sudo pip install yaml` before running `downloader.py`, which is available in the OpenVINO™ toolkit folder at `/opt/intel/computer_vision_sdk_fpga_<version>/deployment_tools/model_optimizer/`


5.1 Configure the Model Optimizer

Choose the configuration option below that best suits your needs.

**Option 1: Configure the Model Optimizer for all Supported Frameworks**

1. Go to the Model Optimizer prerequisites directory: `/opt/intel/computer_vision_sdk_fpga_<version>/deployment_tools/model_optimizer/install_prerequisites`
2. Type `install_prerequisites` to configure Model Optimizer for Caffe, TensorFlow, MXNet, Kaldi*, and ONNX.

The Model Optimizer is configured. Continue to Use the Model Optimizer on page 29.

**Option 2: Configure the Model Optimizer for Individual Frameworks**

1. Go to the Model Optimizer prerequisites directory: `/opt/intel/computer_vision_sdk_fpga_<version>/deployment_tools/model_optimizer/install_prerequisites`
2. Type the command to configure one or more frameworks:
   - Caffe: `install_prerequisites_caffe`
   - TensorFlow: `install_prerequisites_tf`
   - MXNet: `install_prerequisites_mxnet`
   - ONNX: `install_prerequisites_onnx`
   - Kaldi: `install_prerequisites_kaldi`

The Model Optimizer is configured. Continue to Use the Model Optimizer on page 29.
5.2 Use the Model Optimizer

Before you use the Inference Engine APIs, you must use the Model Optimizer to create the Intermediate Representation (IR) files from your pre-trained Caffe model. For this conversion, the Model Optimizer Python script converts the prototxt and caffemodel files to generate two files that describe the network:

- .xml: Describes the network topology
- .bin: Contains the weights and biases binary data

For information about the Model Optimizer command line arguments and options:

```
python3 mo_caffe.py --help
```

1. Temporarily set the environment variables:

```
source /opt/intel/computer_vision_sdk_fpga_<version>/bin/setupvars.sh
```

*Note:* The OpenVINO™ environment variables are removed when you close the shell. As an option, use your preferred method to permanently set the variables.

2. Get the mean file for the AlexNet or ResNet topology. This file provides optimized performance.
   - ResNet mean file location: [https://github.com/ry/tensorflow-resnet/tree/master/data](https://github.com/ry/tensorflow-resnet/tree/master/data)

3. Go to the Model Optimizer directory:

```
cd /opt/intel/computer_vision_sdk_fpga_<version>/deployment_tools/model_optimizer/
```

4. Run `mo_caffe.py` on the caffemodel and prototxt files that have the data type that you need. FP11 bitstreams use data type FP16 when generating the IR files:
   - For AlexNet or ResNet:

     ```
     python3 mo_caffe.py --input_model $<caffemodel> --input_proto $<prototxt_file> -n $<name_out> --data_type $<data_type> --scale 1 --mean_file $<mean_file> --output_dir $<xml_path>
     ```
   
   - For GoogleNet, SqueezeNet, VGG16, or SSD300 topology, provide the mean value for optimized performance:

     ```
     python3 mo_caffe.py --input_model $<caffemodel> --input_proto $<prototxt_file> -n $<name_out> --data_type $<data_type> --scale 1 --mean_value [104,117,123] --output_dir $<xml_path>
     ```
For MobileNet v1 and MobileNet v2 topology, provide the scale factor and mean value for optimized performance:

```
python3 mo_caffe.py --input_model $<caffemodel> --input_proto $<prototxt_file> -n $<name_out> --data_type $<data_type> --scale 58.824 --mean_value [104,117,123] --output_dir $<xml_path>
```

**Notes:**  For more information on using the Model Optimizer to convert:

6.0 **Build the Sample Applications**

This section uses CMake to build the sample applications.

1. Temporarily set the environment variables:

   ```
   source /opt/intel/computer_vision_sdk_<version>/bin/setupvars.sh
   ```

   *Note:* The OpenVINO™ environment variables are removed when you close the shell. As an option, use your preferred method to permanently set the variables.

2. Go to the Inference Engine samples directory:

   ```
   cd /opt/intel/computer_vision_sdk_fpga_<version>/deployment_tools/
       inference_engine/samples/
   ```

3. Create a build directory:

   ```
   mkdir build
   ```

4. Go to the Inference Engine samples build directory:

   ```
   cd /opt/intel/computer_vision_sdk_fpga_<version>/deployment_tools/
       inference_engine/samples/build
   ```

5. Run CMake to generate the Makefiles without debugging information:

   ```
   sudo cmake -DCMAKE_BUILD_TYPE=Release /opt/intel/
       computer_vision_sdk_fpga_<version>/deployment_tools/inference_engine/
       samples/
   ```

6. Build the sample applications:

   ```
   make
   make install
   ```

7. Confirm the build exists. If this directory exists, your build was successful:

   ```
   /opt/intel/computer_vision_sdk_fpga_<version>/deployment_tools/
       inference_engine/samples/build/intel64/Release/
   ```

   The existence of this directory confirms you successfully completed the steps in this section.

Continue to *Use the Sample Applications* on page 32.
7.0 Use the Sample Applications

*Important:* You must have completed the previous sections in this document before you will be successful using the sample applications.

For command-line arguments and options used with the sample applications:

```bash
python3 mo_caffe.py --help
```

7.1 classification_async_Sample with Maximum Optimization

- AlexNet* topology example:

```bash
cd /opt/intel/computer_vision_sdk_fpga_<version>/deployment_tools/inference_engine/samples/build/intel64/Release/
export CL_CONTEXT_COMPILER_MODE_INTELFPGA=3
sudo /opt/intel/computer_vision_sdk_fpga_<version>/deployment_tools/demo/squeezenet1.1.labels $<xml_path>
mv squeezenet1.1.labels alexnet_fp16.labels
./classification_sample_async -m $<xml_path>/alexnet_fp16.xml -i $<image_path> -d HETERO:FPGA,CPU -ni $<iteration_number> -nireq 2
```
• AlexNet topology example with a batch size of 96

```
cd /opt/intel/computer_vision_sdk_fpga_<version>/deployment_tools/inference_engine/samples/build/intel64/Release/

export CL_CONTEXT_COMPILER_MODE_INTELFPGA=3

sudo cp /opt/intel/computer_vision_sdk_fpga_<version>/deployment_tools/demo/squeezenet1.1.labels $<xml_path>

mv squeezenet1.1.labels alexnet_fp16.labels

./classification_sample_async -m $<xml_path>/alexnet_fp16.xml `for i in {1..96}; do echo -n "<image_path>"; done` -d HETERO:FPGA,CPU -ni $<iteration_number> -nireq 2
```

The output example shows the `classification_async` with data type FP16, 1000 iterations, and nireq set to 2 for the AlexNet topology:

```
[ INFO ]  InferenceEngine:
  API version ............ 1.4
  Build .................. 16050
[ INFO ]  Parsing input parameters
[ INFO ]  Files were added: 1
[ INFO ]   /opt/intel/computer_vision_sdk_2018.4.420/deployment_tools/demo/car.png
[ INFO ]  Loading plugin
  API version ............ 1.4
  Build .................. heteroPlugin
  Description .......... heteroPlugin
[ INFO ]  Loading network files
[ INFO ]  Preparing input blobs
[ WARNING ]  Image is resized from (787, 259) to (227, 227)
[ INFO ]  Batch size is 1
[ INFO ]  Preparing output blobs
[ INFO ]  Loading model to the plugin
[ INFO ]  Start inference (100 iterations)
[ INFO ]  Processing output blobs

Top 10 results:

```
Image /opt/intel/computer_vision_sdk_2018.4.420/deployment_tools/demo/car.png

479 0.7527428 label car wheel
511 0.0797053 label convertible
436 0.0745316 label beach wagon, station wagon, wagon, estate car, beach waggon, station waggon, waggon
817 0.0466407 label sports car, sport car
656 0.0310694 label minivan
661 0.0056141 label Model T
581 0.0031988 label grille, radiator grille
468 0.0030763 label cab, hack, taxi, taxicab
717 0.0023221 label pickup, pickup truck
627 0.0016857 label limousine, limo
```

Top 10 results:
7.2 object_detection_ssd

SSD300 topology:

```bash
cd /opt/intel/computer_vision_sdk_fpga_<version>/deployment_tools/
inference_engine/samples/build/intel64/Release/

export CL_CONTEXT_COMPILER_MODE_INTELFPGA=3

./object_detection_sample_ssd -m $<xml_path> -i $<image_path> -d HETERO:FPGA,CPU -i
$/opt/intel/computer_vision_sdk_fpga_<version>/deployment_tools/
inference_engine/samples/build/intel64/Release/lib/libcpu_extension.so
```

7.3 Other Samples

Other sample application are available to run on the Intel® Vision Accelerator Design with an Intel® Arria® 10 FPGA. For information on how to run the demos in OpenVINO™ toolkit, see the Inference Engine sample documentation.

For more information on pre-trained models available, see the pre-trained model information.

Other sample applications you can run with the Intel® Vision Accelerator Design with an Intel® Arria® 10 FPGA:

<table>
<thead>
<tr>
<th>Sample Application</th>
<th>Model Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>classification_sample</td>
<td>Model downloader - AlexNet</td>
</tr>
<tr>
<td>classification_sample_async</td>
<td>Model downloader - AlexNet</td>
</tr>
<tr>
<td>hello_autoresize_classification</td>
<td>Model downloader - AlexNet</td>
</tr>
<tr>
<td>hello_request_classification</td>
<td>Model downloader - AlexNet</td>
</tr>
</tbody>
</table>
## Use the Sample Applications—Intel Vision Accelerator Design with an Intel Arria 10 FPGA

<table>
<thead>
<tr>
<th>Sample Application</th>
<th>Model Used</th>
</tr>
</thead>
</table>
| interactive_face_detection_sample           | face-detection-retail-0004  
|                                              | age-gender-recognition-retail-0013                                     |
|                                              | head-pose-estimation-adas-0001                                            |
| security_barrier_camera_sample              | vehicle-license-plate-detection-barrier-0007                               |
|                                              | vehicle-attributes-recognition-barrier-0010                                |
|                                              | license-plate-recognition-barrier-0001                                    |
| object_detection_demo                       | faster_rcnn_vgg16                                                           |
| object_detection_sample_ssd                 | person-detection-retail-0013                                              |
| object_detection_demo_ssd_async             | person-detection-retail-0014                                              |
| validation_app                              | Model downloader - AlexNet                                                 |
| segmentation_demo                           | fcn8_FP16                                                                  |
| multi-channel-demo                          | face-detection-retail-0004                                                 |
| benchmark_app                               | person-vehicle-bike-detection-crossroad-0078                              |
8.0 Using Multiple FPGA Devices

The Inference Engine FPGA plugin can load different networks on multiple FPGA devices. Each Intel® Vision Accelerator Design with an Intel® Arria® 10 FPGA card is enumerated with a unique ID, starting from 0.

By default, all networks are loaded to the device with ID 0. To load a network to a non-default device, specify the KEY_DEVICE_ID to an incremental number. To load two Alexnet* networks on two different Intel® Vision Accelerator Design with an Intel® Arria® 10 FPGA cards, use these steps:

1. Program the first Intel® Vision Accelerator Design with an Intel® Arria® 10 FPGA card with a corresponding bitstream:
   ```
aocl program acl0 5-0_PL1_FP16_AlexNet_GoogleNet_SqueezeNet.aocx
   ```

2. Load the second Intel® Vision Accelerator Design with an Intel® Arria® 10 FPGA:
   ```
aocl program acl1 5-0_PL1_FP16_AlexNet_GoogleNet_SqueezeNet.aocx
   ```

3. Go to the sample application Release build directory:
   ```
cd /opt/intel/computer_vision_sdk_<version>/deployment_tools/inference_engine/samples/build/intel64/Release
   ```

4. Export the environment variable:
   ```
export CL_CONTEXT_COMPILER_MODE_INTELFPGA=3
   ```

5. Check the performance of the FPGA cards:
   ```
./perfcheck -m $<xml_path> -i $<image_path> -d HETERO:FPGA,CPU -inputs_dir $<image_dir> -num_networks 2 -num_fpga devices 2
   ```

   **Note:** This command can only be used with multiple FPGA cards on a single network.

   The result is the performance from two Alexnet networks on two Intel® Vision Accelerator Design with an Intel® Arria® 10 FPGA cards.

Use a Sample Application to Check the Performance

1. Go to the sample application Release build directory:
   ```
cd /opt/intel/computer_vision_sdk_<version>/deployment_tools/inference_engine/samples/build/intel64/Release
   ```

2. Export the environment variable:
   ```
export CL_CONTEXT_COMPILER_MODE_INTELFPGA=3
   ```
3. Check the performance of the FPGA cards:

```
./perfcheck -m $<xml_path> -i $<image_path> -d HETERO:FPGA,CPU -inputs_dir $<image_dir> -num_networks 2 -num_fpga devices 2
```

*Note:* This command can only be used with multiple FPGA cards on a single network.

The result is the performance from two Alexnet networks on two Intel® Vision Accelerator Design with an Intel® Arria® 10 FPGA cards.