Programmer's Guide

VectorBlox Accelerator SDK v1.0

November 2020

Revision History

Revision	Revision Date	Details of Change
1	November 2020	Revision 1.0 is the first publication of the document. The document was released in November 2020.

Contents

1 Preface

1.1 About this Document

This programmers' guide provides details about Microsemi® CoreVectorBlox Tool Flow environment and its use for converting neural networks and generating graphs to accelerate inference on an FPGA using CoreVectorBlox IP.

1.2 Intended Audience

Software developer’s using Microsemi CoreVectorBlox IP to accelerate neural network applications.

2 Introduction

2.1 Overview

VectorBlox Programmer's Guide covers both the installation and use of the VectorBlox Accelerator SDK. It includes the steps needed to convert a high-level model description to graph that runs on CoreVectorBlox IP or the bit-accurate simulator. The guide also covers the use of the underlying C based firmware and API needed to run the CoreVectorBlox IP on hardware.

2.2 Supported Operating Systems

Only Ubuntu 16.04 using Python 3.5 and Ubuntu 18.04 using Python 3.6 are supported.

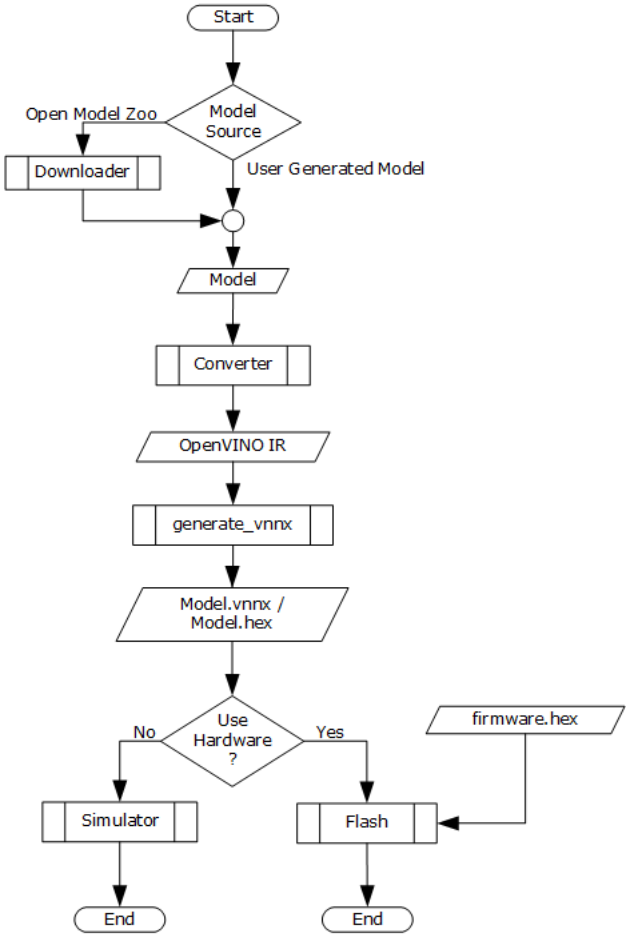
Users who are required to use Microsoft Windows are recommended to try out Windows Subsystem for Linux: *https://docs.microsoft.com/en-us/windows/wsl/install-win10* . However, there are a few things to be aware of when using Windows Subsystem for Linux:

1. DNS is broken when connected to a VPN. The VPN should be down when installing or running the VectorBlox Accelerator SDK.
2. The sdk must be installed in the Linux filesystem, and not in the Windows filesystem available in /mnt/c.

2.3 Flow Diagram – SDK

The following figure shows a complete view of the VectorBlox SDK. This diagram begins with the model downloader and ends with a graph running on hardware or the simulator. The user may enter this flow at any point if they have the necessary files. All steps are not required to be run from the SDK.

Figure 1 • VectorBlox SDK Flow Diagram



3 SDK

3.1 Overview

The SDK is provided as a single zip file, containing everything needed to produce and test neural networks. The neural networks produced are ready to deploy on the PolarFire Video Kit (or any PolarFire FPGA with a CoreVectorBlox IP instantiated). The layout of the archive is as follows:

```
docs
drivers
    vectorblox
example
    c
    python
    README.md
    sim-run-model
fw
install_dependencies.sh
install_venv.sh
lib
    libvbx_cnn_sim.so
python
README.md
requirements.txt
setup_vars.sh
tutorials
    caffee
    darknet
    onnx
    pytorch
    sample_images
    tensorflow
```

3.2 Installation

Before a user is ready to use the SDK, they must first install dependencies, setup the Python3 virtual environment (venv), and set several environment variables.

To install dependencies, unzip to Linux location, navigate to the root folder of the SDK, and run the following commands:

```
bash install_dependencies.sh
```

Note: sudo access is required to install packages

The script `install_dependencies.sh` will install Linux packages necessary for running some of the C based libraries used by the Python based tools of the SDK.

Next, the Python3 virtual environment must be created and environment variables set with the following command:

```
source setup_vars.sh
```

Once installed, the user can activate the VBX python environment (and then run the various tools), from any location, by running:

```
source $VBX_SDK/vbx_env/bin/activate
```

You will know you are in the VBX python environment when your shell prompt is prefixed with “(vbx_env)”.

To exit, or deactivate, the environment, run:

```
deactivate
```

Note:

The examples of activating and deactivating the virtual environment is shown in the tutorial scripts.

3.3 OpenVINO™ Toolkit Components

The OpenVINO™ Toolkit is an open-source tool flow project that allows users to deploy pre-trained networks to hardware. The VectorBlox Accelerator SDK leverages several of the OpenVINO™ tools in the design flow. The following components of the 2021.1 release are used:

- OpenVINO model zoo (and model downloader Python script)
- OpenVINO model converter (for model conversion from various input frameworks)

Each of these tools are described below, along with links to additional OpenVINO documentation. The tool commands are available from the command line within the VBX python environment.

3.3.1 OpenVINO Model Zoo

The OpenVINO model zoo provides access to many popular neural networks, along with key information about preprocessing and performance. A tool to download models is also provided. This tool is a modified version of openvino's `downloader.py`. A short description of the command usage is shown as follows. The full documentation is available in *OpenVINO Toolkit*.

Print all models available in the model zoo:

```
~/sdk (vbx_env active)
  downloader --print_all
```

Download a model:

```
~/sdk (vbx_env active)
  downloader --name MODEL
```

3.3.2 OpenVINO Model Converter and Optimizer

The OpenVINO `converter` command is a tool to convert and optimize models for inference. Models from various source frameworks are converted to the OpenVINO Intermediate Representation (IR). The networks are further optimized for inference, removing training-time layers like Dropout and applying layer fusion. This tool is a modified version of openvino's `converter.py`. A short description of the commands usage is shown as follows. The full documentation is available in *OpenVINO Toolkit*.

Convert and optimize a Caffe model without scale and mean values:

```
~/sdk (vbx_env active)
  converter –framework caffe --input_model MODEL.caffe
```

Convert and optimize a Caffe model with scale and mean values:

```
~/sdk (vbx_env active)
  converter –framework caffe --input_model MODEL.caffe \
--scale_values [127., 127., 127] --mean_values [64., 64., 64.]
```

Note:

The model's documentation must be reviewed to determine if there was pre-processing or if scale and mean values need to be included with this convert command.

3.4 VBX Graph Generation

The OpenVINO IR model is created in the previous step. This model runs through the VectorBlox graph generation tool which produces a quantized VNNX file that runs both on hardware and with the simulator. The user must provide the input xml file, the output file name, the hardware configuration (V1000,V500,V250), and a directory of sample images to use for calibrating the quantized model. Optionally, a test image can be included in the graph, allowing inference and post-processing to be easily tested. A graph can also be cut at a given node if a user's post-processing code replaces later operations. Then the user can add the bias correction flag, which adjusts biases for activations to closely match the original floating point model, using the provided sample images.

Display usage:

```
~/sdk (vbx_env active)
usage: generate_vnnx [-h] -x XML -f
       SAMPLES_FOLDER [-sc SAMPLES_COUNT][-i IMAGE] -c
       {V250,V500,V1000} [--cut CUT] -o OUTPUT -bc

optional arguments:
  -h, --help                  show this help message and exit
  -x XML, --xml XML           OpenVINO I8 model description (.xml)
  -i IMAGE, --image IMAGE     provide test input image for model, must be correct
dimensions
  -d {mnist,imagenet,yolov2,yolov3,deep,generic}, --description
{mnist,imagenet,yolov2,yolov3,deep,generic}
  -c {V250,V500,V1000}, --size-conf {V250,V500,V1000}
                              size configuration to build model for
  --cut CUT                   cuts graph after OpenVINO node (use name specified by
Netron)
  -o OUTPUT, --output OUTPUT  name of output file

  -f SAMPLES_FOLDER,    --samples-folder SAMPLES_FOLDER
                             provide folder of sample images to gather layer statistics


  -sc SAMPLES_COUNT,    --samples-count SAMPLES_COUNT
                              provide max number of sample images to run
  -bc, --bias-correction
                              apply bias correction to layers to improve accuracy
```

Generate graph from OpenVINO ImageNet model, for V1000 CoreVectorBlox size configuration:

```
~/sdk (vbx_env active)
  generate_vnnx -x MODEL.xml -i IMAGE.jpg -c V1000 -o MODEL.vnnx
```

The command `generate_vnnx` is available from the command line within the VBX python environment.

3.5 Supported OpenVINO Layers

The OpenVINO operator set that Core Vectorblox takes its layers from are documented in this section. The following table lists the supported layers and any caveats to the layer's support.

Table 1 • Supported OpenVINO Layers

Layer Name	Notes
Add	—
AvgPool	—

Layer Name	Notes
Clamp	—
Concat	—
Constant	—
Convolution	—
Gather	Only supported if output is constant
GroupConvolution	—
Interpolate	—
LRN	Very slow implementation
MatMul	—
MaxPool	—
Multiply	Only supported if it is multiplying each channel by a scalar
PReLU	—
Pad	Only constant padding is supported
Parameter	—
RegionYolo	Removed from graph. Must be handled with post processing.
ReLU	—
ReorgYolo	—
Reshape	—
Result	—
SoftMax	Only supported if it is output layer
ShapeOf	—
Squeeze	Only Supported on axis==1, mode ==max
TopK	—
Transpose	Supported if node can be replaced by a reshape or order parameter is equal to (0,2,3,1)
Unsqueeze	—

3.6 VBX Inference Simulation

The VBX graph (VNNX) is generated and ready to run on the functionally accurate VBX simulator. Example scripts which call the simulator and run post-processing are provided for classification and YOLO-based

object detection tasks. Further details on simulator use and post-processing that these scripts leverage, can be found in subsequent sections of this guide.

3.6.1 C Simulation

To run any graph on the simulator within the C API, use the following command:

```
~/sdk (vbx_env active)
$VBX_SDK/examples/sim-run-model MODEL_FILE.vnnx IMAGE.jpg
(CLASSIFY|YOLOV2|TINYYOLOV2|TINYYOLOV3)
```

3.6.2 Python Simulation

The following scripts run the indicated models with the Python version of the simulator and print the inference results. They also generate an annotated output image.

Run MNIST graph on the simulator:

```
~/sdk (vbx_env active)
  $VBX_SDK/example/python/mnist.py MODEL.vnnx IMAGE.jpg
```

Run ImageNet graph on the simulator:

```
~/sdk (vbx_env active)
  $VBX_SDK/example/python/imagenet.py MODEL.vnnx IMAGE.jpg
```

Run YOLOv2 VOC graph on the simulator:

```
~/sdk (vbx_env active)
  $VBX_SDK/example/python/yolov2.py MODEL.vnnx IMAGE.jpg
```

Run YOLOv3 COCO graph on the simulator:

```
~/sdk (vbx_env active)
  $VBX_SDK/example/python/yolov3.py MODEL.vnnx IMAGE.jpg
```

3.7 API for interacting with Core Vectorblox

3.7.1 C API (Hardware and Simulator)

Enumerations

vbx_cnn_calc_type_e	VBX_CNN_CALC_TYPE_UINT8, VBX_CNN_CALC_TYPE_INT8, VBX_CNN_CALC_TYPE_INT16, VBX_CNN_CALC_TYPE_INT32, VBX_CNN_CALC_TYPE_UNKNOWN
vbx_cnn_size_conf_e	VBX_CNN_SIZE_CONF_V250 = 0, VBX_CNN_SIZE_CONF_V500 = 1, VBX_CNN_SIZE_CONF_V1000 = 2
vbx_cnn_err_e	INVALID_FIRMWARE_ADDRESS = 1, FIRMWARE_ADDRESS_NOT_READY = 2, START_NOT_CLEAR = 3,

	OUTPUT_VALID_NOT_SET = 4, FIRMWARE_BLOB_VERSION_MISMATCH = 5, INVALID_NETWORK_ADDRESS = 6, MODEL_BLOB_INVALID = 7, MODEL_BLOB_VERSION_MISMATCH = 8, MODEL_BLOB_SIZE_CONFIGURATION_MISMATCH = 9, FIRMWARE_BLOB_STALE = 10
vbx_cnn_state_e	READY = 1, RUNNING = 2, RUNNING_READY = 3, FULL = 6, ERROR = 8

3.7.1.1 Function Documentation

3.7.1.1.1 int model_check_sanity

`int model_check_sanity(const model_t * model)`

Model Parsing Function Check to see if the model parameter looks like a valid model.

Parameters

model
The model to check

Returns

Non-zero if the model does not look valid.

3.7.1.1.2 size_t model_get_allocate_bytes

`size_t model_get_allocate_bytes(const model_t * model)`

Get size required to store the entire model, include temporary buffers. The temporary buffers must be contiguous with the model data.

Parameters

model
The model to query

Returns

The size required to store entire model in memory.

3.7.1.1.3 size_t model_get_data_bytes

`size_t model_get_data_bytes(const model_t * model)`

Get the size required to store only the data part of the model.

Parameters

model

The model to query

3.7.1.1.4 vbx_cnn_calc_type_e model_get_input_datatype

`vbx_cnn_calc_type_e model_get_input_datatype(const model_t *` *model*`, int` *input_index*`)`

Get the datatype of an input buffer.

Parameters

model
The model to query
input_index
The index of the input to get the datatype of

Returns

The data type of model input.

3.7.1.1.5 size_t model_get_input_length

`size_t model_get_input_length(const model_t *` *model*`, int` *input_index*`)`

Get length in elements of an input buffer.

Parameters

model
The model to query
input_index
The index of the input to get the length of

Returns

The length in elements of input buffer.

3.7.1.1.6 size_t model_get_num_inputs

`size_t model_get_num_inputs(const model_t *` *model*`)`

Get the number of input buffers for the model.

Parameters

model
The model to query

Returns

The number of inputs for the model.

3.7.1.1.7 size_t model_get_num_outputs

`size_t model_get_num_outputs(const model_t *` *model*`)`

Get the number of output buffers for the model.

Parameters

model

The model to query

Returns

The number of outputs for the model.

3.7.1.1.8 vbx_cnn_calc_type_e model_get_output_datatype

`vbx_cnn_calc_type_e model_get_output_datatype(const model_t * model, int output_index)`

Get the datatype of an output buffer.

Parameters

model
The model to query
output_index
The index of the output to get the datatype of

Returns

The data type of model output.

3.7.1.1.9 size_t model_get_output_length

`size_t model_get_output_length(const model_t * model, int output_index)`

Get length in elements of an output buffer.

Parameters

model
The model to query
output_index
The index of the output to get the length of

Returns

The length in elements of output buffer.

3.7.1.1.10 float model_get_output_scale_value

`float model_get_output_scale_value(const model_t * model, int output_index)`

Get the amount the output values should be scaled to get the true values.

Parameters

model
The model to query
output_index
The output for which to get the scale value

Returns

Scale value for model output.

3.7.1.1.11 vbx_cnn_size_conf_e model_get_size_conf

`vbx_cnn_size_conf_e model_get_size_conf(const model_t *` *model*)

Get size configuration that the model was generated for; will be one of:

- 0 = V250
- 1 = V500
- 2 = V1000

Parameters

model
The model to query

Returns

The size configuration the model was generated for.

3.7.1.1.12 void* model_get_test_input

`void* model_get_test_input(const model_t *` *model*, `int` *input_index*)

Get a pointer to test input to run through the graph for an input buffer.

Parameters

model
The model to query
input_index
The input from which to get the test input

Returns

Pointer to test input of model.

3.7.1.1.13 vbx_cnn_err_e vbx_cnn_get_error_val

`vbx_cnn_err_e vbx_cnn_get_error_val(vbx_cnn_t *` *vbx_cnn*)

Read error register and return the error.

Parameters

vbx_cnn
The vbx_cnn object to use

Returns

Current value of Error Register.

3.7.1.1.14 vbx_cnn_state_e vbx_cnn_get_state

`vbx_cnn_state_e vbx_cnn_get_state(vbx_cnn_t *` *vbx_cnn*)

Query vbx_cnn to see if there is a model running.

Parameters

vbx_cnn

The vbx_cnn object to use

Returns

Current state of the core. One of:

- READY = 1 (Can accept model immediately)
- RUNNING = 2 (Model Running,can accept model eventually)
- RUNNING_READY = 3 (Model Running,can accept model immediately)
- FULL = 6 (Cannot accept model)
- ERROR = 8 (IP Core stopped.)

3.7.1.1.15 vbx_cnn_t* vbx_cnn_init

`vbx_cnn_state_e vbx_cnn_get_state(volatile void * ctrl_reg_addr, void * firmware_blob)`

Initialize `vbx_cnn` IP Core. After this, the core will accept instructions. If any other function in this file is run without a valid initialized `vbx_cnn_t` the result is undefined..

Parameters

ctrl_reg_addr

The address of the VBX CNN S_control port

firmware_blob

A block of memory containing valid instructions for the IP Core. Users must ensure that the instruction blob is reachable by the VBX CNN IP Core's M_AXI port. The firmware blob is packaged with the SDK.

Returns

A `vbx_cnn_t` structure. On success, `.initialized` is set, otherwise it is zero.

3.7.1.1.16 int vbx_cnn_model_poll

`int vbx_cnn_model_poll(vbx_cnn_t * vbx_cnn)`

Wait for model to complete.

Parameters

vbx_cnn

The `vbx_cnn` object to use

Returns

- 1 if network running
- 0 if network done
- −1 if error processing network
- −2 No Network Running

3.7.1.1.17 int vbx_cnn_model_start

`int vbx_cnn_model_start(vbx_cnn_t * vbx_cnn, model_t * model, vbx_cnn_io_ptr_t * io_buffers)`

Run the model specified with the IO buffers specified. One model can be queued while another model is running to achieve peak throughput. In that case, the calling code would look something like:

```
vbx_cnn_model_start(vbx_cnn,model,io_buffers);
 while (input = get_input()){
   io_buffers[0] = input;
   vbx_cnn_model_start(vbx_cnn,model,io_buffers);
   while(vbx_cnn_model_poll(vbx_cnn)>0);
}
while(vbx_cnn_model_poll(vbx_cnn)>0);
```

Parameters

vbx_cnn
The `vbx_cnn` object to use
model
The model to query
io_buffers
Array of pointers to the input and output buffers. The pointers to the input buffers are first, followed by the pointers to the output buffers.

Returns

nonzero if model not run. Occurs if `vbx_cnn_get_state()` returns FULL or ERROR.

3.7.2 Python API (Simulator Only)

3.7.2.1 vbx.sim.Model

Methods

__init__(self,model_bytes)
Create the model object from the bytes object passed into method.
run(self,inputs)
Run the model with inputs passed as a list of numpy arrays. Returns a list of numpy arrays as output.

Attributes

num_outputs
The number of outputs this model has
num_inputs
The number of inputs this model has
output_lengths
A list of lengths in number of elements for each output buffer of the model
input_lengths
A list of lengths in number of elements for each input buffer of the model
output_dtypes
A list of `numpy.dtype` for output describing the element type of each output buffer of the model
input_dtypes
A list of `numpy.dtype` for output describing the element type of each input buffer of the model
output_scale_factor
A list of floats describing how to scale each output buffer of the model
description
A string that the model is generated with describing the model.
test_input
A list of inputs that can be passed into `Model.run()` as test data.

3.8 Post Processing

Example post-processing is provided for various computer-vision neural network tasks. We provide both C-code and Python-code for MNIST, ImageNet and YOLO-based Object detection tasks. These can be used and modified for a customer's needs.

3.8.1 C API (Hardware and Simulator)

3.8.1.1 Data Structures

fix16_box	Int xmin; Int xmax; Int ymin; Int ymax; fix16_t confidence; Int class_id; const char* class_name;

3.8.1.2 Function Documentation

3.8.1.2.1 void post_process_classifier

`void post_process_classifier(fix16_t * outputs, const int output_size, int16_t* output_index, int topk)`

Post processing for classifiers such as MNIST or ImageNet networks. Returns the *topk* indices of the outputs, sorted from highest to lowest.

Parameters

outputs
The unsorted output produced from the model
output_size
The number of outputs
output_index
The returned indexes, sorted by scores highest to lowest
topk
The number of indices to return sorted

Returns

Pointer to the *topk* sorted indices.

3.8.1.2.2 void post_process_tiny_yolov2_voc

`void post_process_tiny_yolov2_voc(fix16_t * outputs, int* detections, fix16_t boxes)`

Post processing for Tiny YOLOv2 VOC (20-category). Returns the number of detected objects, along with the boxes containing coordinates, confidence, and class information.

Parameters

outputs
The output produced from the model

detections
The number of detected objects
boxes
The array of object-detection boxes, one per detection

Returns

Pointer to the number of objects detected, along with pointer to the array of boxes.

3.8.1.2.3 void post_process_tiny_yolov3_coco

`void post_process_tiny_yolov3_coco(fix16_t * outputs[n], int* detections, fix16_t boxes)`

Post processing for Tiny YOLOv3 COCO (80-category). Returns the number of detected objects, along with the boxes containing coordinates, confidence and class information.

Parameters

outputs0
The first output produced from the model
outputs1
The second output produced from the model
detections
The number of detected objects
boxes
The array of object-detection boxes, one per detection

Returns

Pointer to the number of objects detected, along with pointer to the array of boxes.

3.8.2 Python API

3.8.2.1 vbx.postprocess.classifier

Methods

topk(outputs)
Return the sorted indices from highest to lowest.

3.8.2.2 vbx.postprocess.yolo

Methods

yolov2_tiny_voc(outputs)
Returns the boxes of all objects found
yolov3_tiny_coco(outputs)
Returns the boxes of all objects found.

3.9 Tutorials

The SDK contains a set of end-to-end tutorials in the `tutorials` directory.

Microsemi
2355 W. Chandler Blvd.
Chandler, AZ 85224 USA

Within the USA: +1 (480) 792-7200
Fax: +1 (480) 792-7277

Microsemi Corporation, a subsidiary of Microchip Technology Inc. (Nasdaq: MCHP), and its corporate affiliates are leading providers of smart, connected and secure embedded control solutions. Their easy-to-use development tools and comprehensive product portfolio enable customers to create optimal designs which reduce risk while lowering total system cost and time to market. These solutions serve more than 120,000 customers across the industrial, automotive, consumer, aerospace and defense, communications and computing markets. Headquartered in Chandler, Arizona, the company offers outstanding technical support along with dependable delivery and quality. Learn more at **www.microsemi.com**.

50200927