



Android&Linux

Compilation and Integration Guide


Revision: 0.2

Release Date: 2019-12-18

Copyright

© 2019 Amlogic. All rights reserved. No part of this document may be reproduced, transmitted, transcribed, or translated into any language in any form or by any means without the written permission of Amlogic.

Trademarks

 , and other Amlogic icons are trademarks of Amlogic companies. All other trademarks and registered trademarks are property of their respective holders.

Disclaimer

Amlogic may make improvements and/or changes in this document or in the product described in this document at any time.

This product is not intended for use in medical, life saving, or life sustaining applications.

Circuit diagrams and other information relating to products of Amlogic are included as a means of illustrating typical applications. Consequently, complete information sufficient for production design is not necessarily given. Amlogic makes no representations or warranties with respect to the accuracy or completeness of the contents presented in this document.

Contact Information

- Website: www.amlogic.com
- Pre-sales consultation: contact@amlogic.com
- Technical support: support@amlogic.com

Revision History

Issue 0.2 (2019-12-18)

This is the 0.2 release.

1. buildroot_sdk add 32bit compilation environment under linux environment, compiles 32bit/64bit executable files through parameter control.
2. Added a description of the openvx_case directory in the case code.

Issue 0.1 (2019-07-22)

This is the Initial release.

Contents

Revision History	ii
1. Introduction	1
2. Case Code	2
2.1 Introduction	2
2.2 Process	2
2.2.1 Case Code Generation Process	2
2.2.2 Major interfaces and Parameter Acquisition	2
3. Android-based Compilation and Integration	5
3.1 Introduction	5
3.2 1.TF-lite-based Development	5
3.2.1 Instructions	5
3.2.2 Running Environment Confirmation	6
3.2.3 FAQ	6
3.3 JNI-based Development	7
3.3.1 Introduction	7
3.3.2 Compilation Environment	7
3.3.3 Ndk Compilation	7
4. Linux-based Compilation and Integration	9
4.1 Introduction	9
4.2 Compiling buildroot Executable Files	9
4.2.1 Introduction to the SDK Package	9
4.2.2 Compiling Executable Files	9
4.2.3 Running Executable Files	9
5. FAQ	10

1. Introduction

This document provides the guidance to the case code compilation and integration process in Linux and Android OSs after model transcoding for internal developers and external users.

2. Case Code

2.1 Introduction

The following figure shows the model-transcoded case code:

```

577 12月 18 16:55 BUILD
5835 12月 18 16:55 main.c
2000 12月 18 16:55 makefile.linux
12691 12月 18 16:55 mobilenet_tf.vcxproj
1063 12月 18 16:55 nbg_meta.json
3447390 12月 18 16:55 network_binary.nb
4096 12月 18 16:55 openvx_case
358 12月 18 16:55 vnn_global.h
7505 12月 18 16:55 vnn_mobilenet_tf.c
977 12月 18 16:55 vnn_mobilenet_tf.h
3629 12月 18 16:55 vnn_post_process.c
462 12月 18 16:55 vnn_post_process.h
24058 12月 18 16:55 vnn_pre_process.c
1545 12月 18 16:55 vnn_pre_process.h

```

1. **mobilenet_tf.nb** generates an nbg file with its default name of **network_binary.nb**, but you can rename the file as needed.
2. **vnn_mobilenet_tf.c/vnn_mobilenet_tf.h** is the header file and results corresponding to model creation and release. (The **vnn_modelname.c** file is generated upon case code generation.)
3. **vnn_pre_process.c/vnn_post_process.h** is the model pre-processing file, where case code quantization is performed upon the result of the number of read images minus the average value and then divided by scale.
4. **vnn_post_process.c/vnn_pre_process.h** is the model post-processing file, where the quantized output results are obtained, converted to float32 data, and processed with top operations. (Top5 processing is performed for all case code due to a single template.)
5. **openvx_case**: directly call the openvx interface to run the case code of the nbg model. currently It is not used.

2.2 Process

2.2.1 Case Code Generation Process

TODO

2.2.2 Major interfaces and Parameter Acquisition

The unique global handle is **vsi_nn_graph_t * graph**.

Step 1 Create a handle **graph = vnn_CreateNeuralNetwork(path_to_XXX.nb)**.

Alternatively, use **graph = vnn_CreateXXXX(path_to_XXX.nb, NULL)** in **vnn_XXXmodel.c**.

Step 2 **vnn_VerifyGraph(vsi_nn_graph_t *graph)**

Step 3 Prepare the following parameters:

Number of input sources: graph->input.num

Obtain the corresponding vsi_nn_tensor_t * tensor:

```
tensor = vsi_nn_GetTensor( graph, graph->input.tensors[i]);
```

obtain the size of the related tensor:

```
sz = vsi_nn_GetElementNum(tensor)
```

obtain the dimension information:

```
dim_num = tensor->attr.dim_num
```

```
width = tensor->attr.size[0];
```

```
height = tensor->attr.size[1];
```

```
channel = tensor->attr.size[2];
```

```
batch = tensor->attr.size[3];
```

obtain quantization parameters:

Int8/int16: indicates the number of decimal places, fl:

```
fl = tensor->attr.dtype.fl
```

U8: indicates scale and zero_point:

```
scale = tensor->attr.dtype.scale
```

```
zero_point= tensor->attr.dtype.zero_point
```

Copy the data to the related tensor:

```
vsi_nn_CopyDataToTensor(graph, tensor, data)
```

Perform quantization:

Quantize input values after the operations of average value deduction and scale division:

Int8/16:

```
float fl = pow(2.,tensor->attr.dtype.fl)
```

```
value_int8 = (int)(value_float * fl)
```

U8:

```
value_uint8 = value_float/scale + zero_point
```

Step 4 Deduce and calculate:

```
vnn_ProcessGraph(graph)
```

Step 5 Obtain output results:

Number of output results: graph->output.num

Obtain the corresponding tensor:

```
tensor = vsi_nn_GetTensor(graph, graph->output.tensors[i])
```

Obtain the results and convert them to float32 data:

```
stride = vsi_nn_TypeGetBytes(tensor->attr.dtype.vx_type); (It is 2 for int16 and
1 for int8/u8.)

uint8_t * tensor_data = (uint8_t *)vsi_nn_ConvertTensorToData(graph, tensor);

sz = vsi_nn_GetElementNum(tensor);

buffer = (float *)malloc(sizeof(float) * sz);

Obtain quantization parameters:

Int8/int16: indicates the number of decimal places, fl:

fl = tensor->attr.dtype.fl

U8: indicates scale and zero_point:

float scale = tensor->attr.dtype.scale

int zero_point= tensor->attr.dtype.zero_point

Convert the quantized results to float32 data:

float fl_value = pow(2., -tensor->attr.dtype.fl);

for (i = 0; i < sz; i++) {
    buffer[i] = tensor_data[stride * i] * fl_value;
    or buffer[i] = (tensor_data[stride * i] - zero_point) * scale
}
```

Step 6 Perform debugging:

Save tensor to the following files:

```
vsi_nn_SaveTensorToBinary(graph, tensor, "data.txt");
vsi_nn_SaveTensorToText( graph, tensor, "input.txt", NULL);
vsi_nn_SaveTensorToTextByFp32( graph, tensor, filename, NULL);
```


3. Android-based Compilation and Integration

3.1 Introduction

Currently, the NN function is integrated into the Android platform. The following describes two Android-based development methods:

1. TFLite-based development:

Advantage: If an apk has been developed based on TFLite, you only need to add a code line and quantize the model to int8 data.

Disadvantages:

- Only quantized TFLite models are supported.
- There is a limit on operators and might be compatible problems once an operator is not supported by the TFLite structure.

2. JNI-based development:

Advantage: A variety of model types are supported, including caffe, tensorflow, onnx, darknet, and tflite. Many algorithms are also supported.

Disadvantage: Currently, interfaces are not packaged. Therefore, it is mandatory for JNI-based development to extract the corresponding interfaces based on the transcoded case code, package the interfaces, and then compile them to so.

3.2 1.TFLite-based Development

3.2.1 Instructions

Amlogic implements compatibility with the bottom layer of the TFLite structure in the Android platform. The supported layers are listed in the **OperationCode** enumeration variable. (It is defined in the **NeuralNetworks.h** file in the **frameworks/ml/nn/runtime/include** directory for Android code.)

Only operators that are used onsite and listed in the **AddOpsAndParams** interface in the **nnapi_delegate.cc** file are supported. You can ask onsite configurations for reference.

Compatibility details about APK running on NN:

Step 1 Ensure that the tflite is a 8bit quantization model. If not, use the official TFLiteConverter tool provided by Tensorflow to quantize the mode.

- a) Compile the executable tf file: `bazel build tensorflow/contrib/lite/toco:toco`
- b) Perform quantization:

```
bazel-bin/tensorflow/contrib/lite/toco/toco \
--input_file=$(pwd)/mobilenet_v1_1.0_224/frozen_graph.pb \
--input_format=TENSORFLOW_GRAPHDEF \
--output_format=TFLITE \
--output_file=/tmp/mobilenet_v1_1.0_224.tflite \
```

```
--inference_type=FLOAT \  
--input_type=FLOAT \  
--input_arrays=input \  
--output_arrays=MobilenetV1/Predictions/Reshape_1 \  
--input_shapes=1,224,224,3
```

Step 2 Enable the bottom layer to run on NN by adding `tfliteOptions.setUseNNAPI(true)` to code.

3.2.2 Running Environment Confirmation

After you perform the operations in the previous section, compatibility between TFLite-based apk and NN is achieved. Before apk running, you need to check whether the Android version used in the board support the NN environment. Do as follows upon Adb shell connection:

1. Run the following command to check if the nn server is started:
: ps -A |grep neuralnetwork
2. Run the following command to check if the npu module is loaded:
: lsmod |grep galcore
3. Run the following command to check if the npu node is normal:(Permissions are 664)
: ls /dev/galcore -l

If the results for all the preceding checks are yes, the current Android version support the NN environment, and the apk can be revoked to the NPU through the nn server.

3.2.3 FAQ

How do I deal with the following error when I enable NNAPI after the TFLite model is loaded?

```
E/tflite: Op code 98 is currently not delegated to NNAPI  
E/tflite: Returning error since TFLite returned failure nnapi_delegate.cc:738.  
E/tflite: Failed to build graph for NNAPI  
-----
```

Root cause: As mentioned above, the TFLite structure facilitates compatibility between the NN bottom layer and some layers defined in **NeuralNetworks.h**. This error is reported because a custom layer that is not defined in the header file exists in the TFLite model. If you set NNAPI to true, the TFLite framework will check every layer of the TFLite model. Once there is any layer that is not defined in the **NeuralNetworks.h**, this error will be reported.

Solution: You can split the TFLite model, remove the undefined layer from **NeuralNetworks.h** through CPU operations, and generate a new TFLite model. Alternatively, you can use the JNI-based development method.

3.3 JNI-based Development

Compared with the TFLite-based development method, JNI-based development supports more layers and is more flexible. Currently, this method applies to all the models running on the Buildroot platform.

3.3.1 Introduction

Amlogic provides a sdk package upon ndk compilation, including so and header files that are required for case code compilation. Developers can use this package to compile executable files of the case code and even the so file that corresponds to the interface required by apk development.

3.3.2 Compilation Environment

Ndk installation

Step 1 Download the ndk package.

https://dl.google.com/android/repository/android-ndk-r17-linux-x86_64.zip

Step 2 Decompress the zip file and configure environment variables.

Open `~/.bashrc` and add the following information to the end of the file (or `/etc/profile`).

```
export NDKROOT=/usr/ndk/android-ndk-r17b
```




```
export PATH=$NDKROOT:$PATH
```

Save and exit. Then, update the **source** `~/.bashrc` environment variable.

Step 3 Verify the ndk installation.

Open shell and enter **ndk-build** to check whether the ndk environment is successfully installed. If any information except for **ndk-build not found** is displayed, the environment is ready.

Sdk package introduction

 android_sdk	2019/12/18 16:53	文件夹
 conversion_scripts_nbg_unify	2019/12/18 17:03	文件夹
 ReadMe.txt	2019/11/2 15:54	文本文档

ReadMe.txt introduces how to install the ndk environment and compile so and executable files.

Android_sdk: Header files and so needed to compile case code.

conversion_scripts_nbg_unify: demo code, run cmd: `ndk-build` in this directory.

3.3.3 Ndk Compilation

Step 1 Compile Android executable files.

Refer to the current demo **conversion_scripts_nbg_unify**. Run **ndk-build** to compile case code.

Do as follows to compile your own case code:

- a) Create a directory named **code_dir** in the same directory as **conversion_scripts_nbg_unify**.
- b) In the **code_dir** directory, create a **jni** directory and store case code to this directory.
- c) Copy **Android.mk** and **Applicatoin.mk** in the **ovx_src_facenet_88\jni** directory to the **jni** directory.
- d) Modify **common_src_files** (listing all .c files) in **Android.mk**
- e) In the **code_dir** directory, run **ndk-build**.

**Note**

*If any information indicating no **libjpeg-t.so** is displayed, you can rename **libjpeg.so** in the **sdk/lib** directory to **libjpeg-t.so**, and then copy it to the board.*

Step 2 Compile the Android so file.

- a) Repeat the procedure for compiling executable files.
- b) Modify the end of **Android.mk** as follows:

```
44
45 #LOCAL_CFLAGS += -pie -fPIE
46 #LOCAL_LDFLAGS += -pie -fPIE
47 #include $(BUILD_EXECUTABLE)
48
49 include $(BUILD_SHARED_LIBRARY)
50
```

Step 3 Run the compilation command **ndk-build**.

**Note**

Currently, only the 32-bit so file can be compiled.



4. Linux-based Compilation and Integration

4.1 Introduction

For buildroot-based development, there is an independent major version, supporting compilation of the case code sdk package. This chapter describes how you can compile the transcoded case code to buildroot-based executable files. (Only guidance to compilation to executable files is provided because there is uncertainty about buildroot in other development environments.)

4.2 Compiling buildroot Executable Files

4.2.1 Introduction to the SDK Package

 buildroot_sdk	2019/12/13 10:08	文件夹
 nbg_unify_mobilenet_tf	2019/12/13 10:22	文件夹

buildroot_sdk is the sdk packaged upon buildroot compilation, including header and so files and other resources required for compilation.

nbg_unify_mobilenet_tf is the demo program.

4.2.2 Compiling Executable Files

- Step 1 Modify line15: **export AQROOT=/path_to/buildroot_sdk** of **build_vx.sh** in **nbg_unify_mobilenet_tf**.
- Step 2 In this directory, run **./build_vx.sh arm/arm64**.
- Step 3 Copy case code to a directory that is in the same level as **nbg_unify_mobilenet_tf** in case you want to compile your own case code.
- Step 4 Copy **build_vx.sh** in **nbg_unify_mobilenet_tf** to the case code directory.
- Step 5 Compile and run **./build_vx.sh arm/arm64** to generate executable files in the **bin_r** directory.

4.2.3 Running Executable Files

- Step 1 Copy the compiled executable files, **xxx.nb**, and images to the board.
- Step 2 Run **./xxxx xxxx.nb xxxx.jpg**.



Note

- Case code supports only jpeg images.
- By default, only images in the same size as the model can be processed.
- After running **export VSI_USE_IMAGE_PROCESS=1**, you can upload images in different sizes.

5. FAQ

None