NeuroPilot-Micro for MT3620 User Guide

Version:	0.1 (Alpha Release)
Release date:	2020-05-30

Document Revision History

Revision	Date	Author	Description
0.1	2020-05-30	Sarine Weng	Initial draft

Table of Contents

1 Introduction

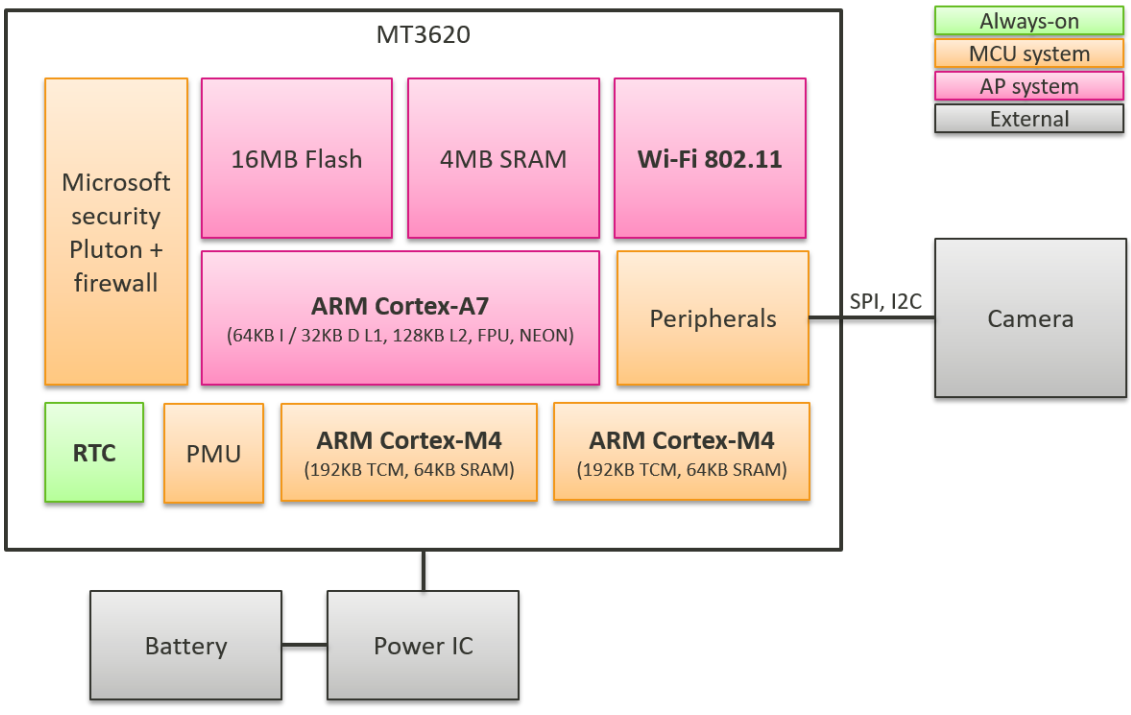
This document is to describe NeuroPilot-Micro functionality and programming methodology based on MT3620 M4 processor. The first chapter gives a brief introduction about MT3620 and NeuroPilot-Micro. The second chapter describes NeuroPilot-Micro architecture, including training and inference phase. Finally, the last chapter demonstrates how to integrate NeuroPilot-Micro with MT3620 by a Person Detection example.

1.1 MT3620 Overview

MT3620 is a highly integrated single chip tri-core MCU designed to meet the requirements of modern, robust internet-connected devices. It leverages the Microsoft Azure Sphere security architecture to provide an unprecedented level of security to connected device manufacturers. For more information, please refer Microsoft "Azure Sphere Platform Overview" document.

Following is the MT3620 block diagram. ARM Cortex-A7 is for Azure Sphere operating system and high-level user applications. In NeuroPilot-Micro, we use it to communicate with peripherals such as camera and display. There are two ARM Cortex-M4 cores, each with dedicated 192KB TCM, 64KB SRAM, and integrated FPU. Besides, there are 1M Flash shared by CA7 and two CM4 cores. For inter-core communication, MT3620 provides 1KB shared memory between CA7 and CM4.

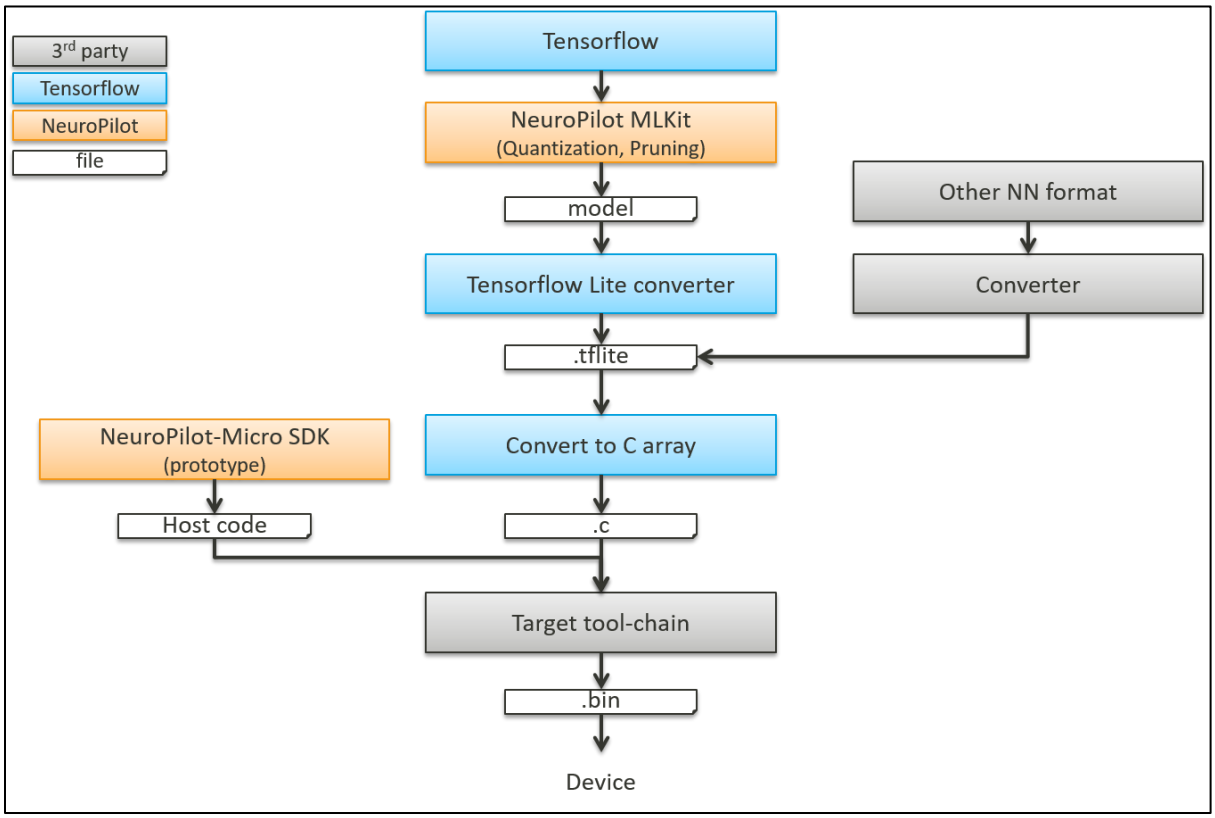
For setup and developing MT3620 applications, please refer to https://docs.microsoft.com/en-ca/azure-sphere/install/overview.



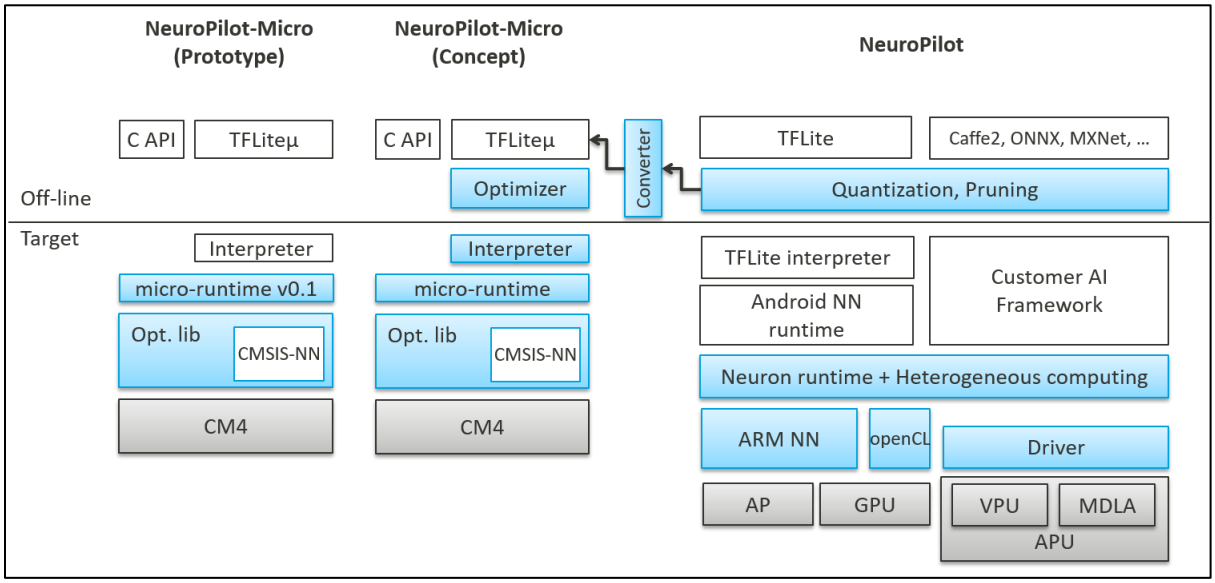
1.2 NeuroPilot-Micro Overview

NeuroPilot-Micro is based on TFLite-Micro, with MTK optimize framework to speedup inference time. Below is the development flow of NeuroPilot-Micro, it can be split into training and inference part. The upper part is for training, we can use TensorFlow and NeuroPilot MLKit to train, quantize, and pruning. The result is a FlatBuffers format model with .tflite as suffix. Instead of using TensorFlow provided tools, other NN tools and converts can be used too. As long as it can be transfer into FlatBuffers format.

The lower part is for inference. The .tflite is converted to C array before the inference works. NeuroPilot-Micro SDK framework is the most important part, it is responsive for run-time and resource arrangement. Finally, the target tool-chain combines all these together to a result binary file.



Following is the road map of NeuroPilot-Micro. The left part is the current status, and is the future concept. In current status, NeuroPilot-Micro is based on TFLite-Micro and CMSIS-NN with our optimizer to speed up inference time. In the future, we design to implement an offline optimizer to optimize trained model before inference. Then we will also improve our interpreter and micro-runtime. Besides, NeuroPilot is a mature framework for training and inference on Android. We plan to develop a converter which can convert NeuroPilot result for NeuroPilot-Micro to use.



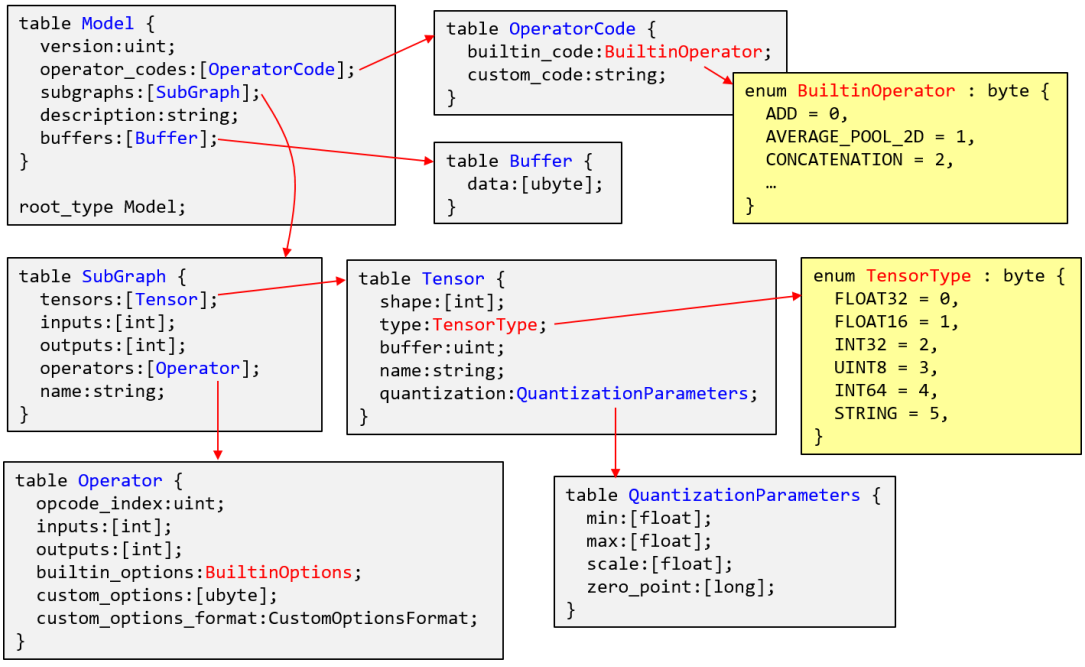
2 NeuroPilot-Micro

As describe before, NeuroPilot-Micro is based on TFLite-Micro, with MTK optimize framework to speed up inference time. So in this chapter, we will first show the TFLite model structure. Then we will describe how to do the training and inference work. Finally, we will mention the optimize mechanism and the SDK structure.

2.1 TFLite Model

TFLite model is FlatBuffers format, it uses schema.fbs to describe the model structure. Schema.fbs is a nested structure with root table Model. Following is the partial content of schema.fbs and its visualization.

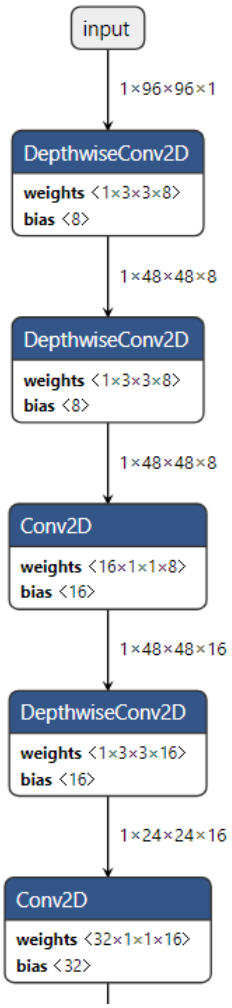
```
table Model {
  // Version of the schema.
  version:uint;

  // A list of all operator codes used in this model. This is
  // kept in order because operators carry an index into this
  // vector.
  operator_codes:[OperatorCode];

  // All the subgraphs of the model. The 0th is assumed to be the main
  // model.
  subgraphs:[SubGraph];

  // A description of the model.
  description:string;

  // Buffers of the model.
  // Note the 0th entry of this array must be an empty buffer (sentinel).
  // This is a convention so that tensors without a buffer can provide 0 as
  // their buffer.
  buffers:[Buffer];

  // Metadata about the model. Indirects into the existings buffers list.
  // Deprecated, prefer to use metadata field.
  metadata_buffer:[int];

  // Metadata about the model.
  metadata:[Metadata];
}
```



TFLite model is a binary file in FlatBuffers format, so there are tools for visualizing it.

- https://lutzroeder.github.io/netron/ – this is an online tool to display model content. It shows the model as graph and lists properties, attributes, input, and output of each node.
- flatc tool – this tool is provided by FlatBuffers, and it can be used to parse and convert model in different format. The most case we use is this command "flatc -t --strict-json --defaults-json schema.fbs -- my_model.tflite". This command transfer tflite model into json format, so it can be viewed by json viewer. Moreover, we can use "--cpp" option to generate C++ header for all definitions in the schema.fbs. Schema_generated.h is the header file name, and the generated functions can be used to get the required model data.
- xxd – this tool is for convert binary into C array. In MT3620 CM4 environment, there is no file system can store model file. So we transfer binary model into C array, and build into execution file.

Following is a visualized example of person detection model using https://lutzroeder.github.io/netron/. It shows the model as a graph. After clicking on the node, the properties will be displayed on the right-hand side.



2.2 Training

There are many online resources to train a TFLite model, such as TensorFlow or Keras. Here we use Keras MNIST training as an example. Training script is located at NeuroPilot-Micro SDK/tools/mnist_keras/keras_cnn.py. In the training script, we first define the model structure and call *model.compile()* to construct the model. Then we call *model.fit()* or *model.fit_generator()* to do the training work.

```
model.add(Convolution2D(14, (3, 3) , activation='relu', input_shape=(28,28,1)))
model.add(Dropout(0.1))
model.add(Convolution2D(10, (1,1) , activation='relu'))
model.add(Dropout(0.1))
model.add(AveragePooling2D(2))
model.add(Convolution2D(14, (3,3) , activation='relu'))
model.add(Dropout(0.1))
model.add(Convolution2D(10, (1,1) , activation='relu'))
model.add(Dropout(0.1))
model.add(AveragePooling2D(2))

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))

opt = keras.optimizers.Adadelta(learning_rate=1.0, rho=0.95)
#opt = optimizer=keras.optimizers.Adam()

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=opt,
              metrics=['accuracy'])
```

```
if not data_augmentation:
    print('Not using data augmentation.')
    model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test),
          callbacks=[history])
else:
    print('Using real-time data augmentation.')
    datagen = ImageDataGenerator(
        rotation_range=15,
        width_shift_range=10,
        height_shift_range=3,
        horizontal_flip=False,
        vertical_flip=False,
        zoom_range=[0.5, 1.1],
        data_format="channels_last")

    datagen.fit(x_train)
    model.fit_generator(datagen.flow(x_train, y_train, batch_size=batch_size),
                        epochs=epochs,
                        verbose=1,
                        validation_data=(x_test, y_test),
                        callbacks=[history])
```

To convert model into TFLite FlatBuffers format, TensorFlow provides following three functions to do the work.

- *TFLiteConverter.from_saved_model()* – converts SavedModel directories
- *TFLiteConverter.from_keras_model()* – converts tf.keras models.
- *TFLiteConverter.from_concrete_functions()* – converts concrete functions.

After getting model, we can reduce model size by post quantization. There are three types of post quantization. Dynamic range quantization is the simplest form, it statically quantizes only the weights from floating point to 8-bits of precision. Another is full integer quantization of weights and activations, it can get further

improvements but need representative dataset. The other is float16 quantization, it quantizes weights to 16 bits float. Below is an example of full integer quantization.

```
mnist_train, _ = load_local_data(mnist_data_path)
images = tf.cast(mnist_train[0], tf.float32) / 255.0
mnist_ds = tf.data.Dataset.from_tensor_slices((images)).batch(1)

def representative_data_gen():
        for input_value in mnist_ds.take(100):
                yield [tf.reshape(input_value, [1, 28, 28, 1])]

converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.representative_dataset = representative_data_gen

converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
converter.inference_input_type = tf.uint8
converter.inference_output_type = tf.uint8
converter.optimizations = [tf.lite.Optimize.OPTIMIZE_FOR_SIZE]
tflite_model = converter.convert()
```

2.3 Inference

Model inference can be separated into following five parts. For more information, please refer to person detection example in NeuroPilot-Micro SDK/app/mt3620/person_detection_demo.

- load model – apply xxd to convert tflite model into C array, then call *::tflite::GetModel()* to load model into inference.

```
model = ::tflite::GetModel(person_detect_model_data);
if (model->version() != TFLITE_SCHEMA_VERSION) {
    error_reporter->Report(
        "Model provided is schema version %d not equal "
        "to supported version %d.\r\n",
        model->version(), TFLITE_SCHEMA_VERSION);
    return;
}
```

- OP resolver – TFLite-Micro provide two OP resolvers, one is AllOpsResolver and another is MicroMutableOpResolver. AllOpsResolver adds all TFLite-Micro OPs in advance, so we do not need to add each OP individually. But it requires larger memory. On the other hand, MicroMutableOpResolver adds each OPs by hands. So we can only add the minimal required OPs to save memory space. Following is the example of using MicroMutableOpResolver, it use *AddBuiltin()* function to add OPs.

```
static tflite::MicroMutableOpResolver resolver;
resolver.AddBuiltin(
    tflite::BuiltinOperator_DEPTHWISE_CONV_2D,
    tflite::ops::micro::Register_DEPTHWISE_CONV_2D());
resolver.AddBuiltin(tflite::BuiltinOperator_CONV_2D,
    tflite::ops::micro::Register_CONV_2D());
resolver.AddBuiltin(tflite::BuiltinOperator_AVERAGE_POOL_2D,
    tflite::ops::micro::Register_AVERAGE_POOL_2D());
```

- initialize interpreter – interpreter is the central part of TFLite-Micro. It is feed with a static working buffer called tensor_arena and responsive for allocate tensor and invoke.

```
static tflite::MicroInterpreter static_interpreter(model, resolver, tensor_arena,
    tensor_arena_size, error_reporter);
```

- allocate tensor – tensor is the input and output of each model node. After interpreter allocated, we call *AllocateTensors()* function to allocate tensors from tensor_arena.

```
TfLiteStatus allocate_status = interpreter->AllocateTensors();
if (allocate_status != kTfLiteOk) {
    error_reporter->Report("AllocateTensors() failed.\r\n");
    return;
}
```

- invoke – finally, we call *Invoke()* function to the inference work when receiving each input data. The result can be get by *output()* function and be parsed according to the algorithm.

```
if (kTfLiteOk != interpreter->Invoke()) {
    error_reporter->Report("Invoke failed.");
    return -1;
}
```

2.4 NeuroPilot-Micro Optimization

When using TFLite-Micro, the most significant problem we encounter is the TCM space. MT3620 CM4 has 192KB TCM, however, a tflite model is about 200~300KB in average. It is too large to fill in TCM. Although we can put the model in flash memory, the execution time become 3x longer than put it in TCM.

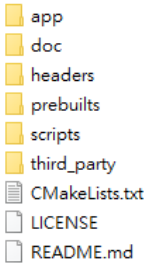
NeuroPilot-Micro provide a mechanism call dynamic loading. It can speed up by 2x~3x with only little TCM storage. To use dynamic loading, we need to prepare a storage in TCM for loading weights and bias into the buffer. The default buffer size is 5KB because it can be reused to load the model separately. This size is enough for most cases according to our experiments. In our experiments, person detection invoke time decrease from 3300ms to 1130ms. Cifar10 invoke time decrease from 3000ms to 870ms.

Dynamic loading is default turned on in our release. There are two functions in interpreter, *EnableDynamicLoad()* and *DisableDynamicLoad()*. We can use these two functions to enable or disable dynamic loading. The header file is located at NeuroPilot-Micro SDK/headers/tensorflow/lite/micro/micro_interpreter.h.

2.5 Release SDK

The released SDK structure is showed in following figure. App folder contains two folders, lib_src and vs_project. Lib_src puts inference examples. New inference example can be put here, but remember to change file in CMakeLists.txt before build it. There are three projects in vs_projects folder. The first one is the MNIST project, which detects the hand writing number through the touch panel. The second is the vison project, it contains the person detection, and CIFAR10 recognition. But it only executes one type of recognition in the same time. Using compile definition to choose PERSON_DETECTION_DEMO or CIFAR10_DEMO in CMakeLists.txt. Besides, we can choose to use USER_MODE or ENG_MODE. The difference between user_mode and eng_mode is the display image size. User_mode shows the origin 160x120 image, and eng_mode displays the image size according to the recognition size. It directly shows the recognition input. The last example is the 2-core project, which utilizes 2 Cortex-M4 and one for person detection another for CIFAT10. So it can recognize person and CIFAR10 objects simultaneously. For more information, please refer to next chapter about MT3620 integration.

Doc folder puts the release documents. Headers folder includes TFLite-Micro header files and CMSIS-NN header files. Prebuilts folder contains the prebuilt libs, libcmsis.a and libtensorflow-microlite.a, which are needed for integrating into MT3620 Visual Studio project. Tools folder contains cmake files and other related materials. Third_party folder puts TFLite-Micro third party libraries header files.



3 Integration

In this chapter, we will first demonstrate how to integrate person detection example into MT3620 CM4 application. Then we will discuss the whole example, including how to connect camera as input, TFT display as output, and inter-core communication. Finally, we will list the steps to build and deploy the whole application.

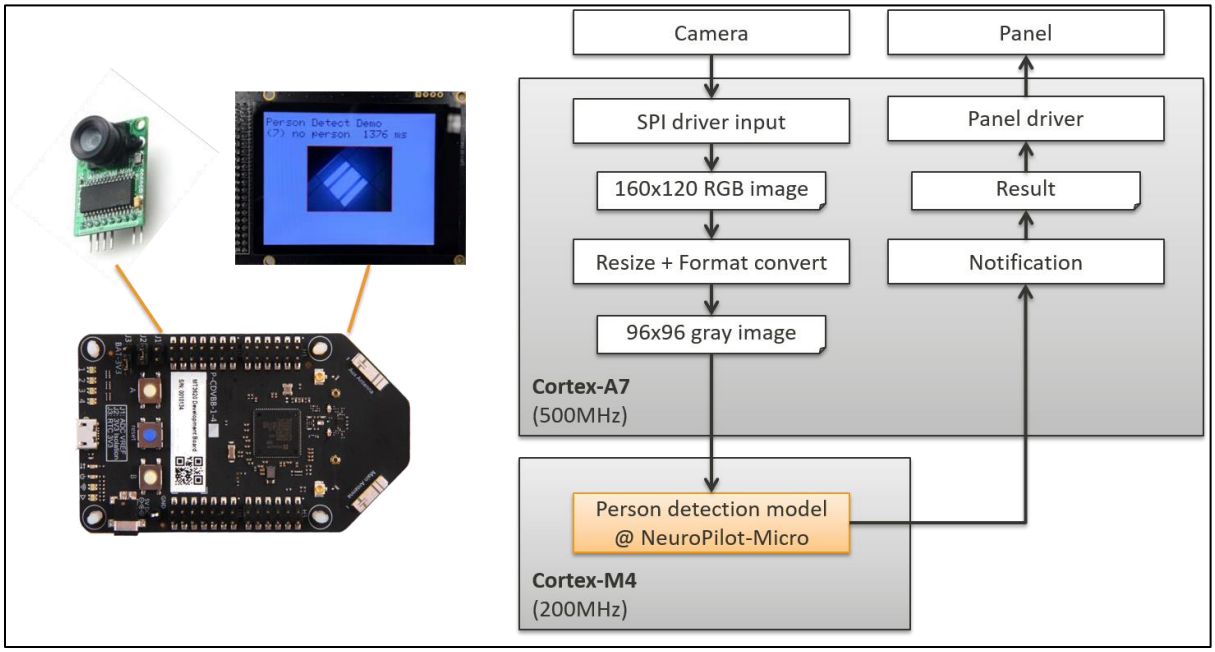
The following are the hardware used in the demo:

- Azure Sphere MT3620 development board
  - ***Seeed Azure Sphere Development Kit***, https://www.seeedstudio.com/Azure-Sphere-MT3620-Development-Kit-US-Version-p-3052.html
- Camera
  - ***Arducam 2MP OV2640 Mini Module SPI Camera***, https://www.arducam.com/product/arducam-2mp-spi-camera-b0067-arduino/
- TFT Display

2.8 inch TFT Capacitive Touch Screen, https://www.buydisplay.com/2-8-inch-tft-touch-shield-for-arduino-w-capacitive-touch-screen-module, with the following configuration: "Pin Header Connection-4-Wire SPI", "VDD=3.3V" and "2.8 inch Capacitive Touch Panel"

3.1 Person Detection Example

The overview of person detection example is shown in following figure. It uses camera as input and TFT panel as output. After receiving camera input from SPI driver, we resize it from 160x120 RGB image to 96x96 gray image for person detection input. Then we send the 96x96 gray image to Cortex-M4 for person detection inference. Finally, the result notifies back to Cortex-A7 and display on panel.



Person detection example is located at NeuroPilot-Micro SDK/app/mt3620/person_detection_demo directory. There are two functions in main.cc, one is *person_detection_setup()* and another is *person_detection_loop()*. The function *person_detection_setup()* do the initial works, including get model, load OPs, declare interpreter, and allocate tensor. The other function *person_detection_loop()* do the inference work, it calls invoke function and decide final result based on the output score. To make sure these two function be used in pure C environment, we need to add the extern "C" key words before them.

To build the static library, type following commands in the NeuroPilot-Micro SDK root folder. (Optional, there is already one copy of the static library in the RTApp folder.)

- mkdir build
- cd build
- cmake ..
- make

After build finish, the result static lib will be shown at the build folder. Copy *libperson_detection_demo_static.a* in the build folder and *libtensorflow-microlite.a*, *libcmsis.a* in the prebuilts/lib folder to Visual Studio RT-core project folder. Then modify CMakeLists.txt to add these three libraries.

```
add_library( libperson_detection_static STATIC IMPORTED)
SET_TARGET_PROPERTIES( libperson_detection_static PROPERTIES IMPORTED_LOCATION ${CMAKE_SOURCE_DIR}/libperson_detection_static.a)

add_library( libtensorflow-microlite STATIC IMPORTED)
SET_TARGET_PROPERTIES( libtensorflow-microlite PROPERTIES IMPORTED_LOCATION ${CMAKE_SOURCE_DIR}/libtensorflow-microlite.a)

add_library( libcmsis STATIC IMPORTED)
SET_TARGET_PROPERTIES( libcmsis PROPERTIES IMPORTED_LOCATION ${CMAKE_SOURCE_DIR}/libcmsis.a)

TARGET_LINK_LIBRARIES(${PROJECT_NAME}
                libperson_detection_static
                libtensorflow-microlite
                libcmsis
                stdc++ supc++ m c gcc nosys
                )
```

To use *person_detection_setup()* and *person_detection_loop()* functions, add the extern key word before declaration in main.c. In addition, since person detection model is too large to fill TCM, we need to put it to Flash. The methodology is to change RODATA_REGION to Flash in linker.ld. Followings are the screen shot of the settings.

```
REGION_ALIAS("CODE_REGION", TCM);
REGION_ALIAS("RODATA_REGION", FLASH);
REGION_ALIAS("DATA_REGION", TCM);
REGION_ALIAS("BSS_REGION", TCM);
```

3.2 Camera Input and TFT Display Output

"Arducam 2MP OV2640 Mini Module SPI camera" and "2.8 inch TFT Capacitive Touch Screen" are used for this demo. Following is the hardware connection to setup camera input and TFT display output.

ArduCAM mini 2MP Plus	RDB	MT3620
SCL	H2-7	GPIO27_MOSI0_RTS0_CLK0
SDA	H2-1	GPIO28_MISO0_RXD0_DATA0
VCC	H3-3	3V3
GND	H3-2	GND
SCK	H4-7	GPIO31_SCLK1_TXD1
MISO	H4-5	GPIO33_MISO1_RXD1_DATA1
MOSI	H4-11	GPIO32_MOSI1_RTS1_CLK1
CS	H1-10	GPIO3

ER-TFTM028-4 Pin	Name	RDB	MT3620
1	GND	H4-2	GND
2	VCC	J3-3	3V3 (Jumper on 1-2 with CR2032 battery)
21	RST	H1-4	GPIO0
23	CS	H3-11	GPIO69_CSA3_CTS3
24	SCK	H3-5	GPIO66_SCLK3_TXD3
25	DC	H1-6	GPIO1
27	SDI	H3-7	GPIO67_MOSI3_RTS3_CLK3
28	SDO	H3-9	GPIO68_MISO3_RXD3_DATA3
29	BL	H1-8	GPIO2

The camera and TFT display setting is describe in sample_hardware.h and sample_hardware.json. Camera driver is in arducam_driver folder, main.c call driver function to init camera and get input to s_CameraBuffer. The camera can be configured to use JPEG or BMP format, the default is BMP format. TFT display driver is in ili9341_driver folder. It provides functions to set text size, cursor position, display string, and draw bitmap. One thing to note is that although camera and display color format are all RGB565, the endian order is different. So we need to change the endian order before display the output image.

3.3 Inter-Core Communication

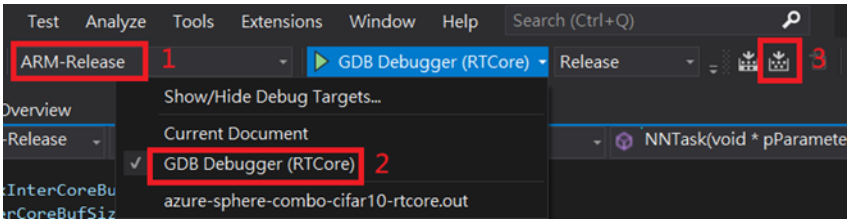
The HL-core is in charge of receiving camera data and handling the screen display, while the RT-core is in charge of the person detection logic. Inter-core communication between HL-core and RT-core is realized by MT3620 hardware mailbox and shared memory. Azure Sphere OS provides 1KB buffer for HL-core and RT-core inter-core communication. Once HL-core receives a complete image from Arducam, HL-core uses the hardware mailbox to notify RT-core and transmit the image data to RT-core with shared memory, and then show the image on the TFT display. The person detection takes about 1.1s to do the inference work. Once the inference is completed on RT-core, the result is sent back to HL-core by using mailbox and shared memory. Note that, the inter-core buffer size is 1KB, if data exceed 1KB then it needs to be send and receive multiple times.

3.4 Build and Deploy

Following are the steps to build and deploy RT-core and HL-core apps. The steps are quite similar, but RT-core app needs to be loaded before HL-core app. Moreover, before build the RT-core app, the AzureSphereRTCoreToolchainVFP.cmake file needs to be copied to Azure SDK installation directory.

3.4.1 RT-Core App

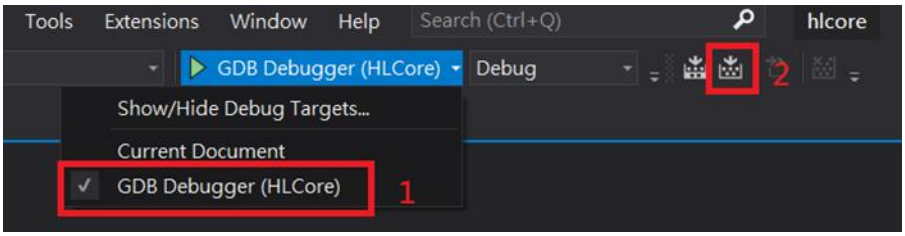
- Start Visual Studio
- Open RT-core project by selecting File -> Open -> CMake, then navigate to the RT-core folder and select CMakeLists.txt to open
- Change project configuration to “ARM-Release”
- Change the debug target to “GDB Debuffer (RTCore)”
- From **Build** menu, click **Build ALL**. (or click the button in the right hand side of the following figure)



- Press F5 to start the RT-core app with debugging
- The log will be output through H3-6 and user could use terminal software for reading and logging

3.4.2 HL-Core App

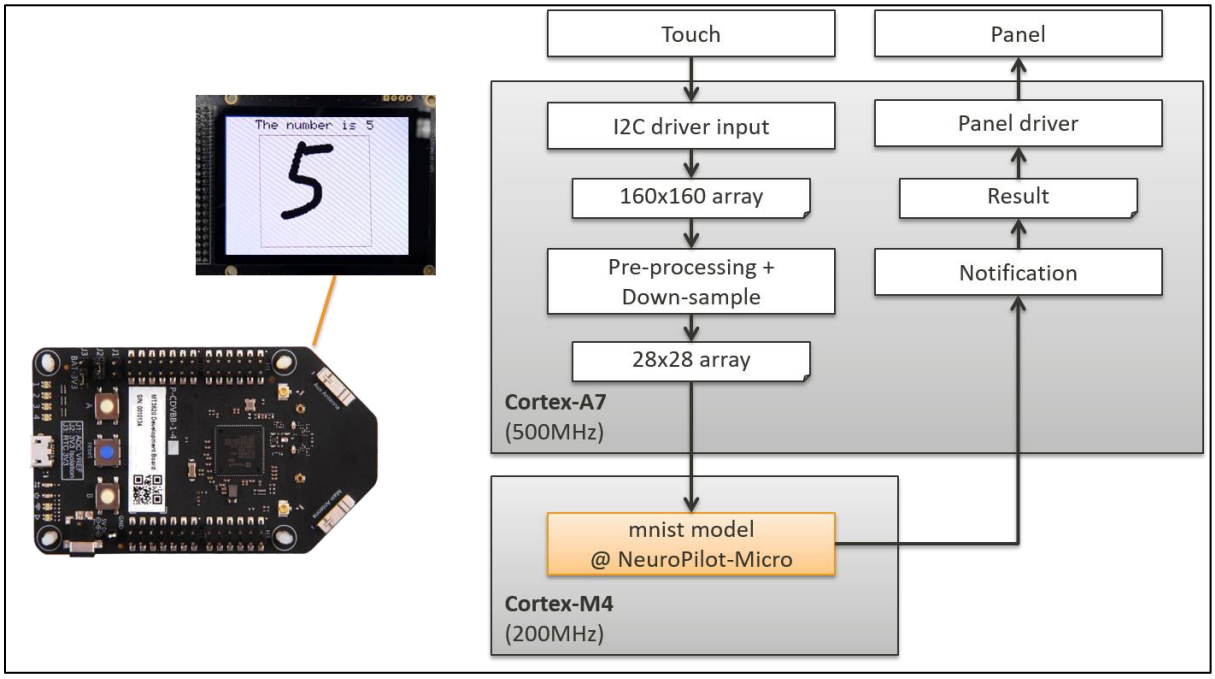
- Start another Visual Studio instance
- Open HL-core project by selecting File -> Open -> CMake, then navigate to the HL-core folder and select CMakeLists.txt to open
- Change the debug target to "GDB Debuffer (HLCore)"
- From **Build** menu, click **Build ALL**. (or click the button in the right hand side of the following figure)



- Press F5 to start the HL-core app with debugging
- The camera image and the result will be shown on the TFT display, the log will be shown in the Visual Studio output window

3.5 MNIST Example

The MNIST example is to recognize hand writing numbers. It receives input from touch panel, pre-process and down-sample from 160x160 array to 28x28 array. Then it sends to Cortex-M4 for MNIST model recognition. After it gets the result, Cortex-M4 notifies Cortex-A7 to display the result.

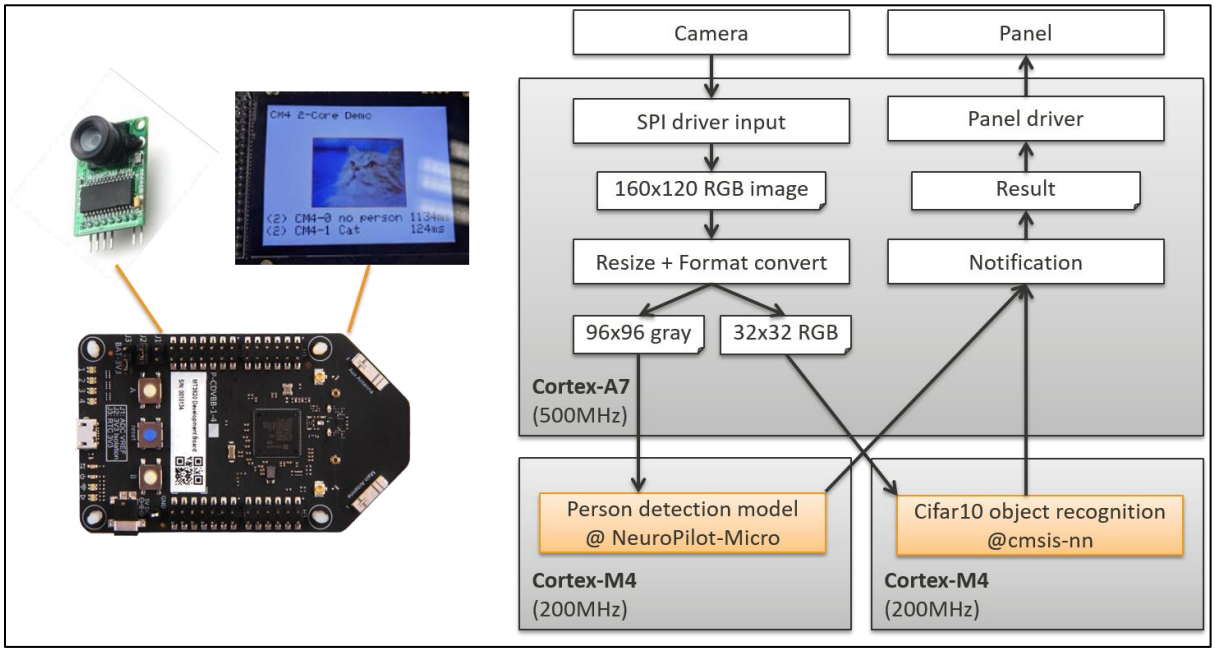


The hardware connection is same as person detection except following additional connects. The TFT touch panel driver is in ft6x06_driver folder. It provides *ft6x06_init()*, *ft6x06_detect_touch()*, and *ft6x06_get_xy()* three functions to detect touch and get touch points. In this example, we create a timer for checking touch per 20ms, and use SM_IDLE, SM_DRAWING, and SM_DONE three states to identify input. When there is no input over 400ms, we regard as touch done. And clean screen after touch done 1s.

ER-TFTM024-4 Pin	Name	RDB	MT3620
30	SCL	H4-12	GPIO37_MOSI2_RTS2_SCL2
31	SDA	H4-6	GPIO38_MISO2_RXD2_SDA2

3.6 2-Core Example

Finally is the 2-core example. MT3620 has 2 Cortex-M4, so we use one core for person detection and another core for CIFAR10 recognition. Following is the application flow. When Cortex-A7 receives camera input, it resizes 160x120 RGB image to 96x96 gray image and 32x32 RGB image. Then it send these two images to each CM4 core respectively. After each CM4 finish their inference work, they notify back and CA7 updates to panel for display.



Cortex-M4 is using UART to display output. For 2-core cases, we need two TTL to USB cables. Connect RXT to H3-6 for core-0 and H3-8 for core-1.

For RT-core, you could use the following commands to sideload the image package:

- azsphere device sideload deploy --imagepackage azure-sphere-combo-2core\rtcore\out\ARM-Release\azure-sphere-combo-cifar10-rtcore.imagepackage
- azsphere device sideload deploy --imagepackage azure-sphere-combo-2core\rtcore_2\out\ARM-Release\azure-sphere-combo-cifar10-rtcore-2.imagepackage

For HL-core application, you could choose to use command or Visual Studio to deploy. The HL-core log will be displayed on the Visual Studio output window.

Terms and Conditions

Your access to and use of this document and the information contained herein (collectively this "Document") is subject to your (including the corporation or other legal entity you represent, collectively "You") acceptance of the terms and conditions set forth below ("T&C"). By using, accessing or downloading this Document, You are accepting the T&C and agree to be bound by the T&C. If You don't agree to the T&C, You may not use this Document and shall immediately destroy any copy thereof.

This Document contains information that is confidential and proprietary to MediaTek Inc. and/or its affiliates (collectively "MediaTek") or its licensors and is provided solely for Your internal use with MediaTek's chipset(s) described in this Document and shall not be used for any other purposes (including but not limited to identifying or providing evidence to support any potential patent infringement claim against MediaTek or any of MediaTek's suppliers and/or direct or indirect customers). Unauthorized use or disclosure of the information contained herein is prohibited. You agree to indemnify MediaTek for any loss or damages suffered by MediaTek for Your unauthorized use or disclosure of this Document, in whole or in part.

MediaTek and its licensors retain titles and all ownership rights in and to this Document and no license (express or implied, by estoppels or otherwise) to any intellectual property rights is granted hereunder. This Document is subject to change without further notification. MEDIATEK DOES NOT ASSUME ANY RESPONSIBILITY ARISING OUT OF OR IN CONNECTION WITH ANY USE OF, OR RELIANCE ON, THIS DOCUMENT, AND SPECIFICALLY DISCLAIMS ANY AND ALL LIABILITY, INCLUDING, WITHOUT LIMITATION, CONSEQUENTIAL OR INCIDENTAL DAMAGES.

THIS DOCUMENT AND ANY OTHER MATERIALS OR TECHNICAL SUPPORT PROVIDED BY MEDIATEK IN CONNECTION WITH THIS DOCUMENT, IF ANY, ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE. MEDIATEK SPECIFICALLY DISCLAIMS ALL WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, COMPLETENESS OR ACCURACY AND ALL WARRANTIES ARISING OUT OF TRADE USAGE OR OUT OF A COURSE OF DEALING OR COURSE OF PERFORMANCE. MEDIATEK SHALL NOT BE RESPONSIBLE FOR ANY MEDIATEK DELIVERABLES MADE TO MEET YOUR SPECIFICATIONS OR TO CONFORM TO A PARTICULAR STANDARD OR OPEN FORUM.

Without limiting the generality of the foregoing, MEDIATEK MAKES NO WARRANTY, REPRESENTATION OR GUARANTEE REGARDING THE SUITABILITY OF ITS PRODUCTS FOR ANY PARTICULAR PURPOSE, NOR DOES MEDIATEK ASSUME ANY LIABILITY ARISING OUT OF THE APPLICATION OR USE OF ANY PRODUCT, CIRCUIT OR SOFTWARE. You agree that You are solely responsible for the designing, validating and testing Your product incorporating MediaTek's product and ensure such product meets applicable standards and any safety, security or other requirements.

The above T&C and all acts in connection with the T&C or this Document shall be governed, construed and interpreted in accordance with the laws of Taiwan, without giving effect to the principles of conflicts of law.