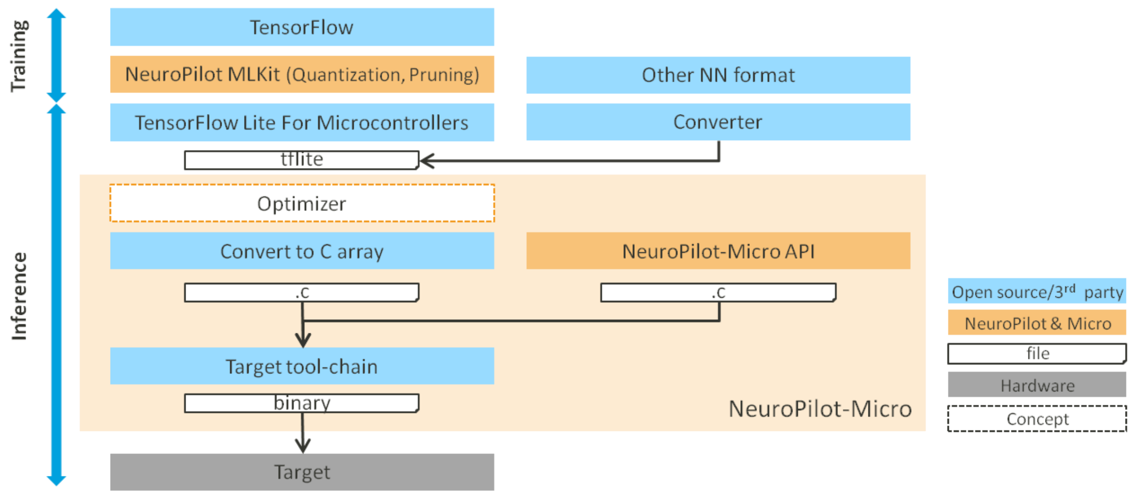
NeuroPilot-Micro MT3620 Getting Started

Version:	0.2 (Beta Release)
Release date:	2020-07-10

Getting Started with NeuroPilot-Micro & TinyML on MT3620

Getting Started: NeuroPilot-Micro SDK

NeuroPilot-Micro is a part of NeuroPilot and specifically designed for TinyML applications, leveraging NeuroPilot optimization tools and provides popular Neural Network frameworks for inference. NeuroPilot-Micro SDK supports TensorFlow Lite for microcontrollers (TFLm) and optimization APIs for model inference and optimization.



NeuroPilot-Micro can be split into two parts: (1) **model training** and (2) **model inference**.

The upper part highlights model training, TensorFlow and NeuroPilot MLKit can be used to train and optimize (e.g., quantization and pruning) the models. In addition to the tools provided by TensorFlow, other neural network tools and converters can be used as long as the model can be transferred into the TensorFlow Lite format.

The lower part is for model inference. The trained model is converted into the TensorFlow Lite format and the result is a FlatBuffers format that is a model file with .tflite filename extension. The .tflite is converted to C array for a non-file system OS. The NeuroPilot-Micro SDK framework is responsible for run-time, resource arrangement, and optimization. Finally, the target toolchain combines all these elements together in a binary file that can be run on the target hardware platform, such as one using the MediaTek MT3620.

With NeuroPilot-Micro frameworks, the model will run efficiently on MediaTek platforms. Using the MT3620 as an example, the model running using Flash as a larger-but-slower pool of memory, yet still can be as fast as at TCM (smaller and faster memory). This helps developers easily run the inference model and we recommend they use Flash for models.

1.1.1 How to run Inference with NeuroPilot-Micro

The following sections will introduce how to run inference with a Neural Network model on a target device with NeuroPilot-Micro. It includes the following:

- Brief Introduction of TensorFlow Lite for Microcontrollers
- Step-by-Step Information on how to run inference at a target
- Optimization Guidelines

TensorFlow Lite For Microcontrollers

TensorFlow Lite for microcontrollers (TFLm) is a part of TensorFlow Lite project, with specific support for the microcontroller ecosystem. Due to the computing performance and memory limitations of a microcontroller, the TFLm implementation is lightweight and doesn't require operating system support, standard C or C++ libraries, or dynamic memory allocation.

The following limitations of TFLm should be considered:

- Support for a limited subset of TensorFlow operations (OPs)
- Necessity of Basic C and C++ runtimes (e.g., printf, memory management, etc.)
- Requirement of a math library for the operation implementation

Please note training is not supported.

1.1.2 Step-by-Step guide how to run Inference at a Target

There are 3 steps for running inference at target:

- Creating the model
- Perform Inference
- Deploy on the target

- **Creating the Model**
  1. **Create or Obtain a TensorFlow Model**
     The model must be small enough to fit on your target device after conversion and it can only use supported operations. You may, however, add your own operations via a custom OP interface.

     NeuroPilot optimization/quantization/reduction tool: ML Kit

  2. **Convert the Model to TensorFlow Lite FlatBuffers**
     Next, convert the model to the standard TensorFlow Lite format using the TensorFlow Lite converter. You may also wish to output a quantized model since these are smaller in size and more efficient to execute. Pre-built models can also be found at Neural Network Models for reference.

     Please refer to Post-training quantization **for the TensorFlow quantization tool.**

  3. **Convert the FlatBuffers to a C Byte Array**
     The following command shows how to convert the FlatBuffers into a C array of a tflite model by unix builtin tool xxd.

     ```
     $ xxd -i mnist.tflite mnist_model.cc
     ```

- **Perform Inference**
  This section will describe how to convert a model to a char array and execute the inference on Linux x86.
  1. **Download NeuroPilot-Micro SDK**
     Extract it.

     ```
     $ unzip Azure-Sphere-MT3620-M4-Samples.zip
     $ cd Azure-Sphere-MT3620-M4-Samples/MT3620_M4_NeuroPilot-Micro
     ```

2. **Convert the Model**
   Download the MNIST model from our model list.
   Convert the mnist.tflite to a hexdump file named mnist_model.cc.

```
$ wget
http://neuropilot.mediatek.inc/resources/public/latest/en/images
/npu_models/mnist.tflite
$ xxd -i mnist.tflite mnist_model.cc
$ cp mnist_model.cc app/lib_src/simple_example/
```

You can replace app/lib_src/simple/mnist_model.cc with the new model file you generated.

3. **A Glance at the Code**
   In this example, there are only three files at app/lib_src/simple.
   - mnist_model.cc : hexdump of the mnist.tflite.
   - mnist_model.h : header file of mnist_model.cc.
   - main.cc : main file.

   This MNIST model needs 22 KB of tensor arena to store all runtime resources.

```
const int tensor_arena_size = 22 * 1024;
```

The input image shape of this model is [1, 28, 28, 1] and output shape is [10].

```
const int inputTensorSize = 28 * 28;
const int outputTensorSize = 10;
```

Set up all the operators you need.
You can use AllOpsResolver when testing on Linux x86.
However, AllOpsResolver requires too much memory to use on an MT3620, so only MicroMutableOpResolver can be used. More details about resolver

```
tflite::AllOpsResolver resolver;
```

Create an instance of Tensorflow Lite Micro interpreter.

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

Allocate runtime resources.

```
TfLiteStatus allocate_status = interpreter->AllocateTensors();
```

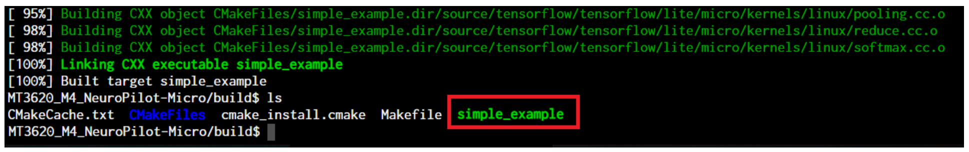
Run the model.

```
interpreter->Invoke();
```

4. **Build all the Files**
   Create a build folder, generate cmake cache, and build the files.

```
$ mkdir build
$ cd build
$ cmake ..
$ make
```

   If built successfully, simple_example will be generated in the build folder.



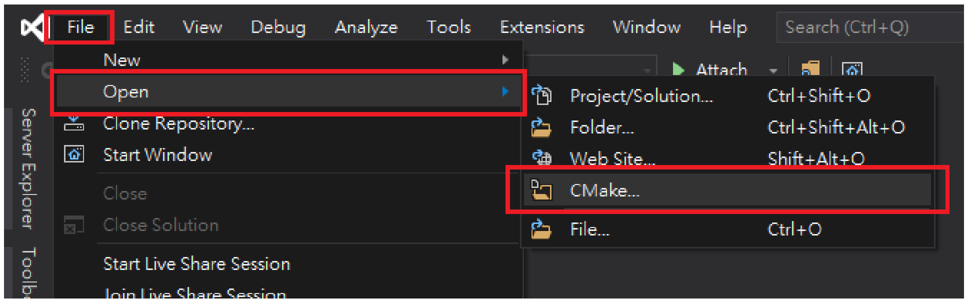
```
[ 95%] Building CXX object CMakeFiles/simple_example.dir/source/tensorflow/tensorflow/lite/micro/kernels/linux/pooling.cc.o
[ 98%] Building CXX object CMakeFiles/simple_example.dir/source/tensorflow/tensorflow/lite/micro/kernels/linux/reduce.cc.o
[ 98%] Building CXX object CMakeFiles/simple_example.dir/source/tensorflow/tensorflow/lite/micro/kernels/linux/softmax.cc.o
[100%] Linking CXX executable simple_example
[100%] Built target simple_example
MT3620_M4_NeuroPilot-Micro/build$ ls
CMakeCache.txt  CMakeFiles  cmake_install.cmake  Makefile  simple_example
MT3620_M4_NeuroPilot-Micro/build$
```

5. **Execution**
   Run simple_example.

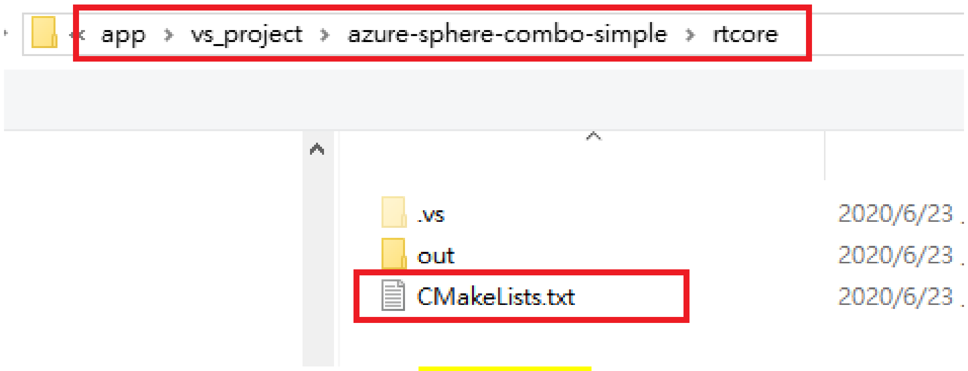
```
$ ./simple_example
GetModel done, size 41112 bytes.

Get prediction 7.

Get prediction 7.

Get prediction 7.
```

Learn more about TensorFlow Lite for microcontrollers: Inference a model by TensorFlow Lite

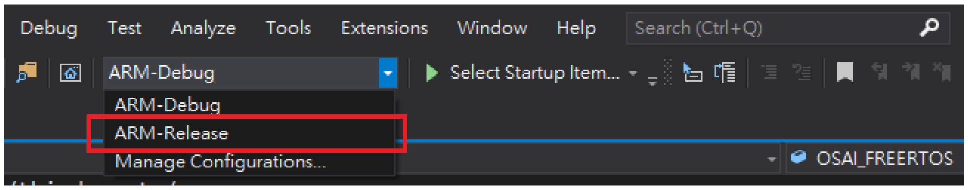
- ## Deploy on the Target
  Now that you have been taught how to run the model on the Linux x86, we can now apply this to the MT3620 running a simple example on the RT-core (Cortex-M4F). The following steps will showcase how to build RT-core and execute this on the MT3620.
  - Start Visual Studio
  - Open RT-core project
    Select File -> Open -> CMake



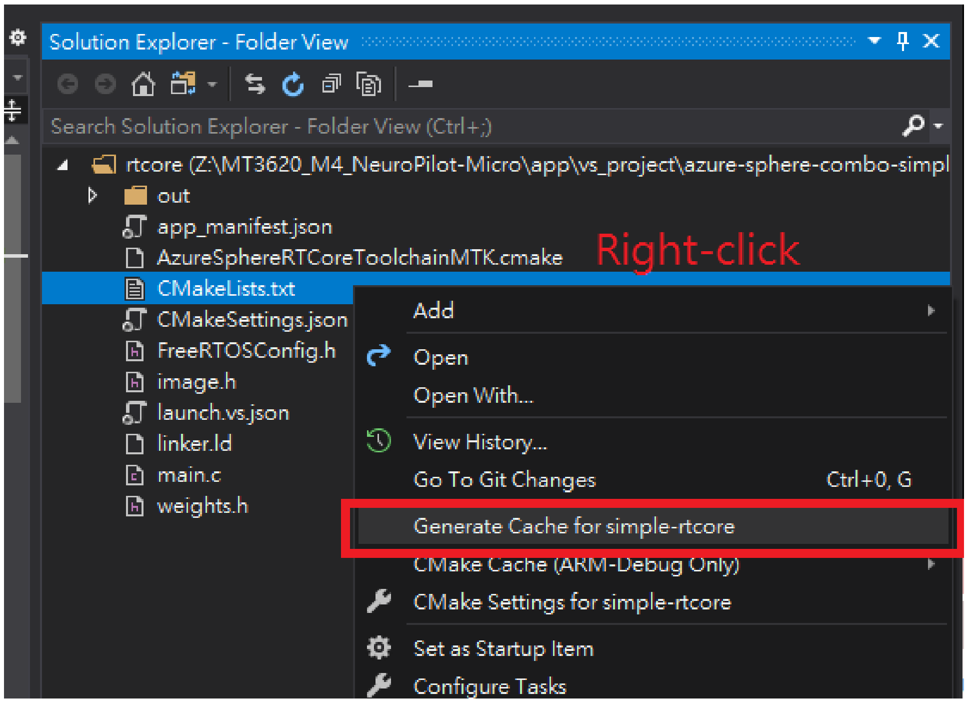
Navigate to app/vs_project/azure-sphere-combo-simple/rtcore and select CMakeLists.txt to open



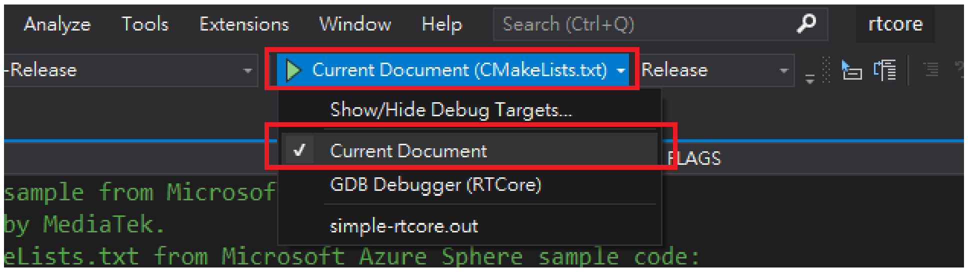
- Change project configuration to ARM-Release



4. In Solution Explorer, right-click CMakeLists.txt and select Generate Cache for simple-rtcore

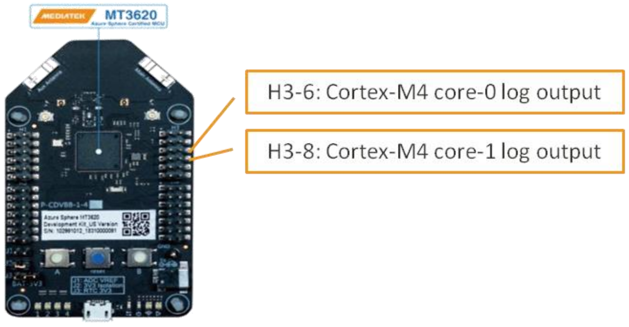

Solution Explorer - Folder View
Search Solution Explorer - Folder View (Ctrl+;)
rtcore (Z:\MT3620_M4_NeuroPilot-Micro\app\vs_project\azure-sphere-combo-simpl
out
app_manifest.json
AzureSphereRTCoreToolchainMTK.cmake
CMakeLists.txt
CMakeSettings.json
FreeRTOSConfig.h
image.h
launch.vs.json
linker.ld
main.c
weights.h
Right-click
Add
Open
Open With...
View History...
Go To Git Changes Ctrl+0, G
Generate Cache for simple-rtcore
CMake Cache (ARM-Debug Only)
CMake Settings for simple-rtcore
Set as Startup Item
Configure Tasks

5. Select Current Document or GDB Debugger(RTCore) in Select startup item.



```
sample from Microsof
by MediaTek.
eLists.txt from Microsoft Azure Sphere sample code:
```

6. Press F5 to build and start the RT-core.
7. The log of RTCore will be output through H3-6.


MT3620
H3-6: Cortex-M4 core-0 log output
H3-8: Cortex-M4 core-1 log output

You may use a USB to UART TTL serial cable and a terminal software to read the output. The USB port attaches to your PC. The RX, which is normally the white wire, attaches to the MT3620 H3-6. Set the baud rate to 115200. And then, the output log will be printed on the terminal software.

```
predic_label = 1
tick_diff = 55

predic_label = 1
tick_diff = 55
```

The execution time of single model_invoke is also printed on terminal software.

If you can only work on Windows x86, please refer to Debug on Windows for additional information.

How to Optimize

With NeuroPilot-Micro, and using the MT3620 as an example, the model can utilize Flash to embrace up to 1MB of memory as fast as TCM which is 192KB fast memory. The speed optimization is larger than 3x.

The optimization is **turned-on by default** but developers will have the opportunity to fine-tune for better performance through the optimization APIs. More information can be obtained by referring to Optimize your Model.

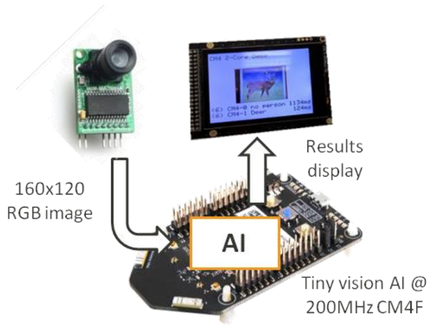
Congratulation!

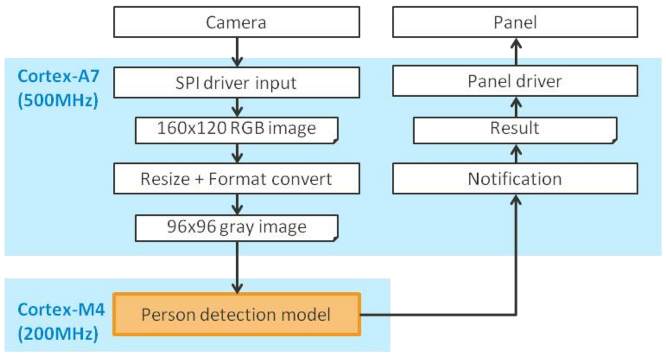
You are now ready to dive into NeuroPilot-Micro. Here the following topics will be covered:

- Reproducing more complex Demos (Tiny Vision Demo / Dual-Core Processing Demo / Handwriting Recognition Demo) and trying to set up a display panel and camera.
- The NeuroPilot-Micro SDK User Guide provides a better understanding of the MT3620 and NeuroPilot-Micro.

Getting Started: Tiny Vision Demo

With this demo, you will be shown examples for Tiny Vision AI and learn about how easy it can be to execute and optimize a model inference by NeuroPilot-Micro.





This simple demo platform uses a camera as input for vision-related applications, and an LCD panel to display the captured images and results. The demo also uses Sphere OS running on the Cortex-A7 for camera input and UI drivers.

The camera captures continuous images with larger than 9 FPS (Frame-Per-Second). After receiving the camera input from the SPI driver, it is resized from a QVGA (160x120) RGB image to a 96x96 gray image for person detection input or a 32x32 RGB image for object detection input. The resized input image is then sent to a Cortex-M4 for person detection and then the result is sent back to the Cortex-A7 for display on the panel.

Step-by-Step Guide: Person Detection Demo

The person detection model size is about 250KB and needs approximately 70KB of tensor area for execution. Given that this is too large to run with TCM (192KB max), with NeuroPilot-Micro optimization the model can use serial Flash (up to 1MB), generating a performance that's similar to TCM. For more information please refer to the NeuroPilot-Micro User Guide.

The demo platform uses the Seeed MT3620 EVK, a camera, and an LCD panel. To set up the tiny vision demo platform, please refer to Learn More: Tiny Vision Platform.

Please note that when doing the demo, you will need two Visual Studio projects--one for the RT-core app and another for the HL-core app. NeuroPilot-Micro SDK has already put the Visual Studio projects in app/vs_project/azure-sphere-combo-vision. Following are the steps below will build and deploy both RT-core and HL-core apps. **Important**: the RT-core app needs to be loaded **before** the HL-core app.

- **RT-Core App Creation and Execution**
    1. Copy app/vs_project/azure-sphere-combo-vision/rtcore/AzureSphereRTCoreToolchainVFP.cmake to Azure SDK installation directory (only once).
    2. Copy build/libperson_detection_demo_static.a to app/vs_project/azure-sphere-combo-vision/rtcore.
    3. Copy prebuilts/lib/libtensorflow-microlite.a and prebuilts/lib/libcmsis.a to app/vs_project/azure-sphere-combo-vision/rtcore.
    4. Start Visual Studio.
    5. Open RT-core project by selecting File -> Open -> CMake, then navigate to app/vs_project/azure-sphere-combo-vision/rtcore and select CMakeLists.txt to open.
    6. Change project configuration to ARM-Release.
    7. In Solution Explorer, right-click CMakeLists.txt and select Generate Cache for RT-core.
    8. In Solution Explorer, right-click CMakeLists.txt and select Build to build the project and generate image package.
    9. Open CMakeLists.txt and press F5 to start the RT-core app with debugging.
    10. The log will be output through H3-6 and can use terminal software for reading.
- **HL-Core App Creation and Execution**
    1. Start Visual Studio.
    2. Open HL-core project by selecting File -> Open -> CMake, then navigate to app/vs_project/azure-sphere-combo-vision/hlcore and select CMakeLists.txt to open.

3. In Solution Explorer, right-click CMakeLists.txt and select Generate Cache for HL-core.
4. In Solution Explorer, right-click CMakeLists.txt and select Build to build the project and generate image package.
5. Open CMakeLists.txt and press F5 to start the HL-core app with debugging.
6. The camera image and the result will be shown on the TFT display, the log will be shown in the Visual Studio output window.

Step-by-Step Guide: Object Detection Demo

Object detection is another popular vision application, and both are also supported within the same demo platform. For providing the correct input, the camera image is re-sized to 32x32 RGB. To learn more about the vision demo SDK, please refer to the **Tiny Vision Demo User Guide**.

Please refer to the person detection demo first, as the **step-by-step process** is the same, however take note of the following modifications are required:

- **RT-Core App Creation and Execution**
  1. Change ${infer_model} in app/vs_project/azure-sphere-combo-vision/rtcore/CMakeLists.txt from PERSON_DETECTION_DEMO to CIFAR10_DEMO.

- **HL-Core App Creation and Execution**
  1. Change ${infer_model} in app/vs_project/azure-sphere-combo-vision/hlcore/CMakeLists.txt from PERSON_DETECTION_DEMO to CIFAR10_DEMO

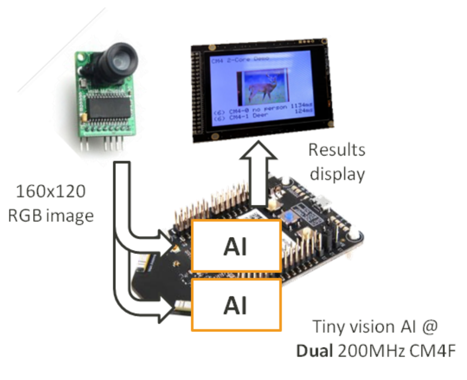
Next Steps

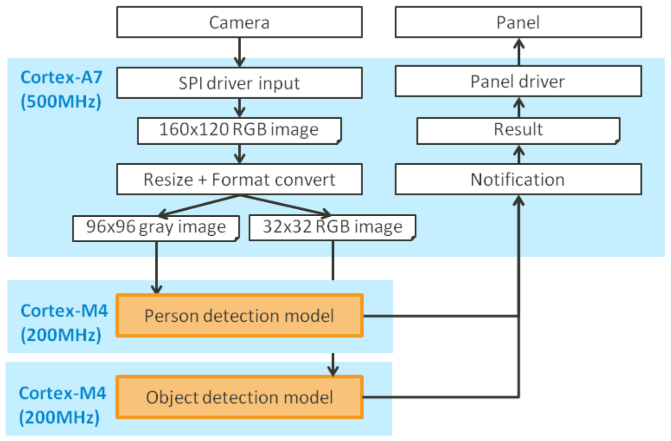
You may want to refer to the other **Getting Started and User Guide** when learning how to use the SDK.

Learn more about Tiny Vision SDK: Tiny Vision SDK User Guide
How to assemble the platform? Learn More: Vision Platform
Learn more about NeuroPilot-Micro: Getting Started with NeuroPilot-Micro

Getting Started: Dual-Core Processing Demo

MT3620 has two powerful Cortex-M4 cores (RT-cores) enabling it to execute two models concurrently for enhanced real-time performance. This demo uses one core for person detection and the other core for object detection. These two models run at the same time and not sequentially: thus, each model continues to operate independently at its own speed and will not be impacted by the load of the other.





The demo platform is the same as the tiny vision demo with dual-core processing enabled. Please refer to Learn More: Tiny Vision Platform for more information. For providing both models with separate input, the input image is resized to two images. One is 96x96 grayscale for person detection and the other is 32x32 RGB for object detection. The two images are then sent to the corresponding Cortex-M4s (RT-core) for model execution. Additionally, the two results are collected by Sphere OS using the Cortex-A7 (HL-core) for display. Please refer to the Dual-Core Demo User Guide to learn more.

Step-by-Step Guide

The demo needs three Visual Studio Projects consisting of the following: one for each of the RT-core apps and another for the HL-core app. It also requires an Azure Sphere SDK command line window. The NeuroPilot-Micro SDK has already put the Visual Studio projects in app/vs_project/azure-sphere-combo-2core. The following steps build and deploy the two RT-core apps and HL-core apps. **Important**: the RT-core apps need to be loaded **before** the HL-core app.

- **RT-Core App for Core-1**
    1. Start Visual Studio.

2. Open RT-core project by selecting File -> Open -> CMake, then navigate to app/vs_project/azure-sphere-combo-vision/rtcore and select CMakeLists.txt to open.
3. Change project configuration to ARM-Release.
4. In Solution Explorer, right-click CMakeLists.txt and select Generate Cache for RT-core.
5. In Solution Explorer, right-click CMakeLists.txt and select Build to build the project and generate image package.
6. The output image package is located at app/vs_project/azure-sphere-combo-2core/rtcore/out/ARM-Release/azure-sphere-combo-cifar10-rtcore.imagepackage.

- **RT-Core App for Core-2**
  1. Start Visual Studio.
  2. Open RT-core project by selecting File -> Open -> CMake, then navigate to app/vs_project/azure-sphere-combo-vision/rtcore and select CMakeLists.txt to open.
  3. Change project configuration to ARM-Release.
  4. In Solution Explorer, right-click CMakeLists.txt and select Generate Cache for RT-core.
  5. In Solution Explorer, right-click CMakeLists.txt and select Build to build the project and generate image package.
  6. The output image package is located at app/vs_project/azure-sphere-combo-2core/rtcore_2/out/ARM-Release/azure-sphere-combo-cifar10-rtcore-2.imagepackage.
- **HL-Core App**
  1. Start Visual Studio.
  2. Open HL-core project by selecting File -> Open -> CMake, then navigate to app/vs_project/azure-sphere-combo-vision/hlcore and select CMakeLists.txt to open.
  3. In Solution Explorer, right-click CMakeLists.txt and select Generate Cache for HL-core.
  4. In Solution Explorer, right-click CMakeLists.txt and select Build to build the project and generate image package.
  5. The output image package is located at app/vs_project/azure-sphere-combo-2core/hlcore/out/ARM-Debug-5+Beta2004/azure-sphere-combo-cifar10-hlcore.imagepackage.
- **Side load to MT3620**

1. Open Azure Sphere SDK command line window.
2. Deploy rtcore-0 image by enter "azsphere device sideload deploy --imagepackage azure-sphere-combo-2core\rtcore\out\ARM-Release\azure-sphere-combo-cifar10-rtcore.imagepackage".
3. Deploy rtcore-1 image by enter "azsphere device sideload deploy --imagepackage azure-sphere-combo-2core\rtcore_2\out\ARM-Release\azure-sphere-combo-cifar10-rtcore-2.imagepackage".
4. Deploy hlcore by Visual Studio open CMakeLists.txt and press F5 to start the HL-core app with debugging.

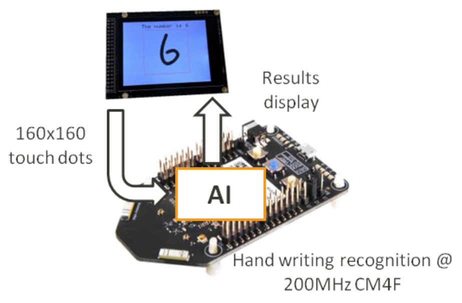
Next Steps

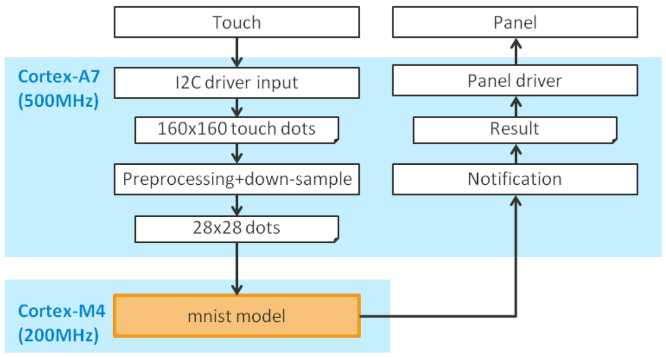
The **Getting Started** and **User Guide** are also useful when learning how to use the SDK.

Learn more about dual-core processing SDK: Dual-Core Processing SDK User Guide
How to assemble the platform? Learn More: Tiny Vision Platform
Learn more about NeuroPilot-Micro: Getting Started with NeuroPilot-Micro

Getting Started: Handwriting Recognition Demo

The demo using the MNIST model can recognize hand-written numbers through the following process: user writes a number input via the touch panel. Then pre-processing and down-sampling from a 160x160 array to 28x28 array. The array is sent to the Cortex-M4 for MNIST model recognition. After it calculates the result, the Cortex-M4 notifies the Cortex-A7 to display the result.





To setup the handwriting demo platform, please refer to Learn More: Handwriting Platform.

Step-by-Step Guide

Two Visual Studio projects are needed: one for the RT-core app and another for the HL-core app. The NeuroPilot-Micro SDK has already put the Visual Studio projects in app/vs_project/aazure-sphere-combo-mnist. The Following are the steps to build and deploy the RT-core and HL-core apps. **Important**: the RT-core app needs to be loaded **before** the HL-core app.

- ### RT-Core App Creation and Execution
    1. Start Visual Studio.
    2. Open RT-core project by selecting File -> Open -> CMake, then navigate to app/vs_project/azure-sphere-combo-mnist/rtcore and select CMakeLists.txt to open.
    3. Change project configuration to ARM-Release.
    4. In Solution Explorer, right-click CMakeLists.txt and select Generate Cache for RT-core.
    5. In Solution Explorer, right-click CMakeLists.txt and select Build to build the project and generate image package.
    6. Open CMakeLists.txt and press F5 to start the RT-core app with debugging.
    7. The log will be output through H3-6 and can use terminal software for reading.
- ### HL-Core App Creation and Execution
    1. Start Visual Studio.
    2. Open HL-core project by selecting File -> Open -> CMake, then navigate to app/vs_project/azure-sphere-combo-mnist/hlcore and select CMakeLists.txt to open.
    3. In Solution Explorer, right-click CMakeLists.txt and select Generate Cache for HL-core.
    4. In Solution Explorer, right-click CMakeLists.txt and select Build to build the project and generate image package.
    5. Open CMakeLists.txt and press F5 to start the HL-core app with debugging.
    6. The camera image and the result will be shown on the TFT display, the log will be shown in the Visual Studio output window.

Next Steps

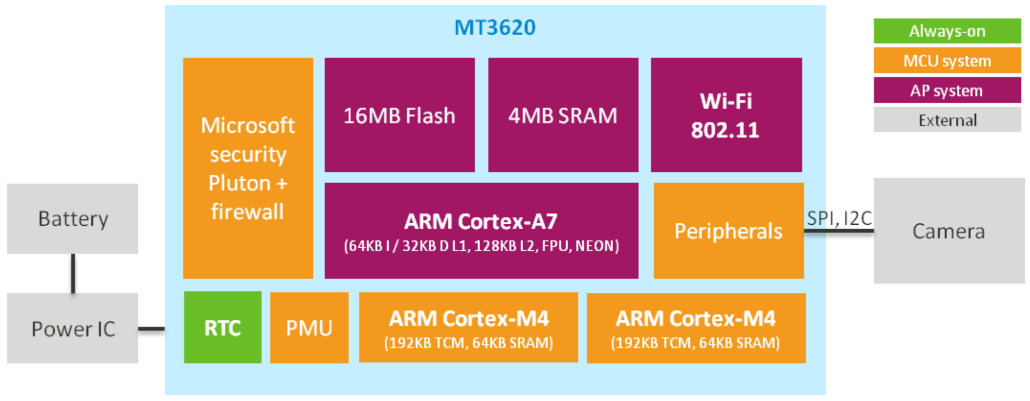
The **Getting Started** and **User Guide** are also useful when learning how to use the SDK.

Learn more about handwriting recognition SDK: Handwriting Recognition SDK User Guide
How to assemble the platform? Learn More: Handwriting Platform
Learn more about NeuroPilot-Micro: Getting Started with NeuroPilot-Micro

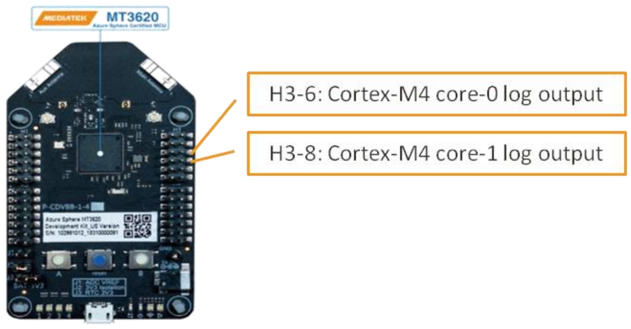
Learn More: MT3620 Development Board

The MT3620 is a highly integrated, heterogeneous tri-core MCU specifically designed to meet the requirements of modern and robust internet-connected devices. It leverages the Microsoft Azure Sphere security architecture to provide an unprecedented level of security to connected device manufacturers. For setup and development of MT3620 compatible applications, please refer to the Microsoft Azure Sphere Document.

The following is a block diagram of the MT3620. The ARM Cortex-A7 (HL-core) hosts the Azure Sphere operating system and high-level user applications. NeuroPilot-Micro demo uses the Cortex-A7 to communicate with peripherals, such as camera and display. There are also two ARM Cortex-M4 cores with a dedicated 192KB TCM per core; 64KB SRAM; and integrated FPU. There is also 1MB Flash, which is shared by the Cortex-A7 and two Cortex-M4 cores. For inter-core communication, the MT3620 provides 1KB shared memory between the Cortex-A7 and Cortex-M4 cores.


MT3620
Microsoft security Pluton + firewall
16MB Flash
4MB SRAM
Wi-Fi 802.11
ARM Cortex-A7 (64KB I / 32KB D L1, 128KB L2, FPU, NEON)
Peripherals
SPI, I2C
Camera
Battery
Power IC
RTC
PMU
ARM Cortex-M4 (192KB TCM, 64KB SRAM)
ARM Cortex-M4 (192KB TCM, 64KB SRAM)
Always-on
MCU system
AP system
External

There are also different types of MT3620 development boards. The NeuroPilot-Micro SDK uses the Seeed Azure Sphere Development Kit for easy peripheral connection. The following image shows the development board pin-out, highlighting the MT3620 TTL to the USB transmission pins for output of the Cortex-M4 log. The Cortex-M4 core-0 output is H3-6 and core-1 output is H3-8 respectively.


MEDIATEK MT3620
H3-6: Cortex-M4 core-0 log output
H3-8: Cortex-M4 core-1 log output

Next Steps

The **Getting Started** and **User Guide** are also useful when learning how to use the SDK.

Learn more about MT3620: MT3620 Datasheet
Learn more about Microsoft Azure Sphere: Microsoft Azure Sphere Document

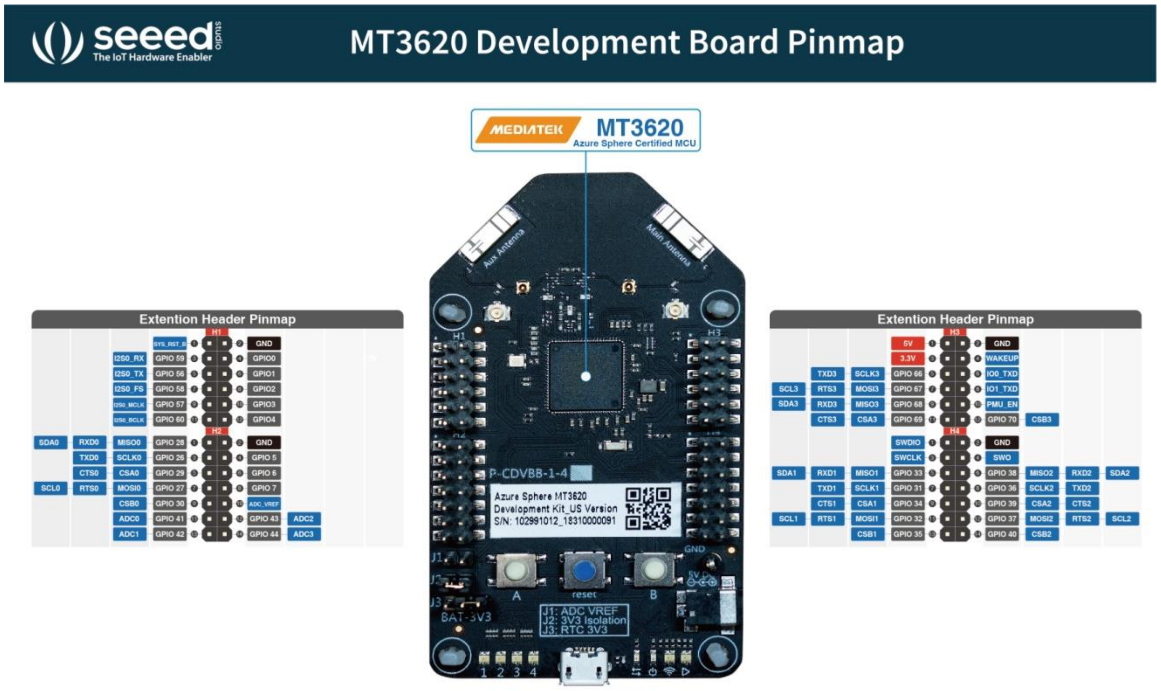
Learn More: Tiny Vision Platform

The Tiny Vision platform requires an MT3620 development board, a compatible camera and display. The camera should support, at a minimum, a QVGA (160x120) RGB image capture for supporting both object detection (32x32, RGB) and person detection (96x96, gray). A recommended display resolution of VGA or greater is desirable for the best user interface experience.

Demo platform components:

- Seeed MT3620 Development Board
- Arducam 2MP OV2640 Mini Module SPI Camera
- TFT Capacitive Touch Screen

The image below is
from https://wiki.seeedstudio.com/Azure_Sphere_MT3620_Development_Kit/.


seeed studio
The IoT Hardware Enabler
MT3620 Development Board Pinmap
MEDIATEK MT3620
Azure Sphere Certified MCU
Extention Header Pinmap
H1
SYS_RST_B 1 2 GND
I2S0_RX GPIO 59 3 4 GPIO0
I2S0_TX GPIO 56 5 6 GPIO1
I2S0_FS GPIO 58 7 8 GPIO2
I2S0_MCLK GPIO 57 9 10 GPIO3
I2S0_BCLK GPIO 60 11 12 GPIO4
H2
SDA0 RXD0 MISO0 GPIO 28 1 2 GND
TXD0 SCLK0 GPIO 26 3 4 GPIO 5
CTS0 CSA0 GPIO 29 5 6 GPIO 6
SCL0 RTS0 MOSI0 GPIO 27 7 8 GPIO 7
CSB0 GPIO 30 9 10 ADC_VREF
ADC0 GPIO 41 11 12 GPIO 43 ADC2
ADC1 GPIO 42 13 14 GPIO 44 ADC3
Extention Header Pinmap
H3
5V 1 2 GND
3.3V 3 4 WAKEUP
TXD3 SCLK3 GPIO 66 5 6 IO0_TXD
SCL3 RTS3 MOSI3 GPIO 67 7 8 IO1_TXD
SDA3 RXD3 MISO3 GPIO 68 9 10 PMU_EN
CTS3 CSA3 GPIO 69 11 12 GPIO 70 CSB3
H4
SWDIO 1 2 GND
SWCLK 3 4 SWO
SDA1 RXD1 MISO1 GPIO 33 5 6 GPIO 38 MISO2 RXD2 SDA2
TXD1 SCLK1 GPIO 31 7 8 GPIO 36 SCLK2 TXD2
CTS1 CSA1 GPIO 34 9 10 GPIO 39 CSA2 CTS2
SCL1 RTS1 MOSI1 GPIO 32 11 12 GPIO 37 MOSI2 RTS2 SCL2
CSB1 GPIO 35 13 14 GPIO 40 CSB2
Aux Antenna
Main Antenna
Azure Sphere MT3620
Development Kit_US Version
S/N: 102991012_18310000091
A
reset
B
J1: ADC VREF
J2: 3V3 Isolation
J3: RTC 3V3
BAT-3V3
1 2 3 4

Arducam 2MP OV2640 Mini Module SPI Camera and a 2.8 inch TFT Capacitive Touch Screen with the configuration, ’Pin Header Connection-4-Wire SPI’, ’VDD=3.3V’, and a ’2.8 inch Capacitive Touch Panel’. The following are the connections to setup the camera and display hardware.

ArduCam Connections

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

TFT Connections

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

Next Steps

The **Getting Started** and **User Guide** are also useful when learning how to use the SDK.

How to reproduce the demo? Tiny Vision SDK User Guide

Learn More: Handwriting Recognition Platform

The Handwriting Recognition Platform requires an MT3620 development board and a touch display. The touch resolution should be higher than 160x160 for input quality and a recommended display resolution of VGA or greater for the best user interface experience.

Demo platform components:

- Seeed MT3620 Development Board
- TFT Capacitive Touch Screen

The Handwriting Recognition Platform uses the same TFT display as Tiny Vision Platform. The following connections are used for the touch panel:

Touch Connection

ER-TFTM028-4 Pin	Name	RDB	MT3620
30	SCL	H4-12	GPIO37_MOSI2_RTS_SCL2
31	SDA	H4-6	GPIO38_MISO2_RXD2_SDA2

Next Steps

The **Getting Started** and **User Guide** are also useful when learning how to use the SDK.

How to reproduce the demo? Handwriting Recognition SDK User Guide