AN13854

NPU Migration Guide from i.MX 8M Plus to i.MX 93

Rev. 1 — 18 September 2023 — Application note

Document information

Information	Content
Keywords	i.MX 93, i.MX 8M Plus, neural processing unit (NPU), TensorFlow Lite (TFLite), AN13854
Abstract	This application note describes how to migrate a machine learning application from i.MX 8M Plus to i.MX 93 with NPU acceleration.

1 Introduction

This application note describes how to migrate a machine learning application from i.MX 8M Plus to i.MX 93 with neural processing unit (NPU) acceleration. The NPU of the i.MX 8M Plus and i.MX 93 devices are different IPs, and their features and usage methods are also different. This document introduces the differences between the i.MX 8M Plus NPU and the i.MX 93 NPU, and covers the operation guidance and optimization suggestions. However, if the CPU inference is used, the i.MX 8M Plus and i.MX 93 devices function in a similar manner.

2 NPU overview

The NPU provides hardware acceleration for AI/ML workloads and vision functions. NPU with different IP is used by i.MX 8M Plus and i.MX 93.

2.1 Block diagram

The following figure shows the i.MX 8M Plus NPU high-level block diagram.

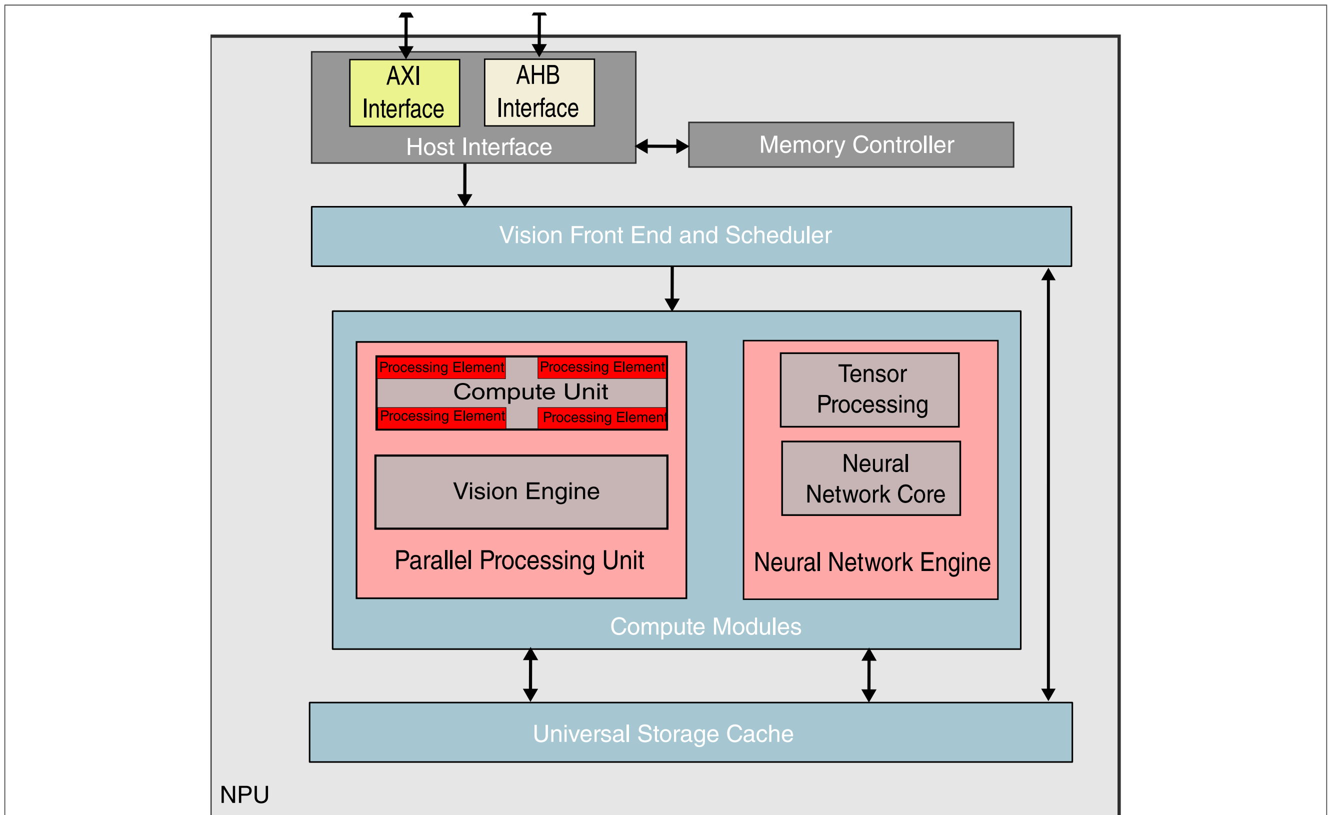

Figure 1. i.MX 8M Plus NPU high-level block diagram

Table 1. i.MX 8M Plus NPU functional blocks

i.MX 8M Plus NPU block	Description
Host interface	Allows the NPU to communicate with external memory and the CPU through the AXI / AHB bus. In this block, data crosses clock domain boundaries
Memory controller	Internal memory management unit that controls the block-to-host memory request interface

All information provided in this document is subject to legal disclaimers.

© 2023 NXP B.V. All rights reserved.

Table 1. i.MX 8M Plus NPU functional blocks...*continued*

i.MX 8M Plus NPU block	Description
Vision front end	Inserts high-level primitives and commands into the vision pipeline
Neural network core	Provides parallel convolution MAC for recognition functions using 8 bits or 16 bits integer
Tensor processing fabric	Provides data preprocessing and supports compression and pruning for multidimensional array processing for Neural Nets
Compute unit	SIMD processor programmable execution unit that performs as a compute unit. The NPU block has one vector4 parallel processor unit, which also acts as four processing elements
Vision engine	Provides advanced image processing functions
Universal storage cache	Cache shared between the vision front end and the parallel processing unit

Note:

For i.MX 8M Plus NPU supported operator list, refer to https://www.nxp.com.cn/docs/en/user-guide/IMX-MACHINE-LEARNING-UG.pdf— OVXLIB Operation Support with NPU.

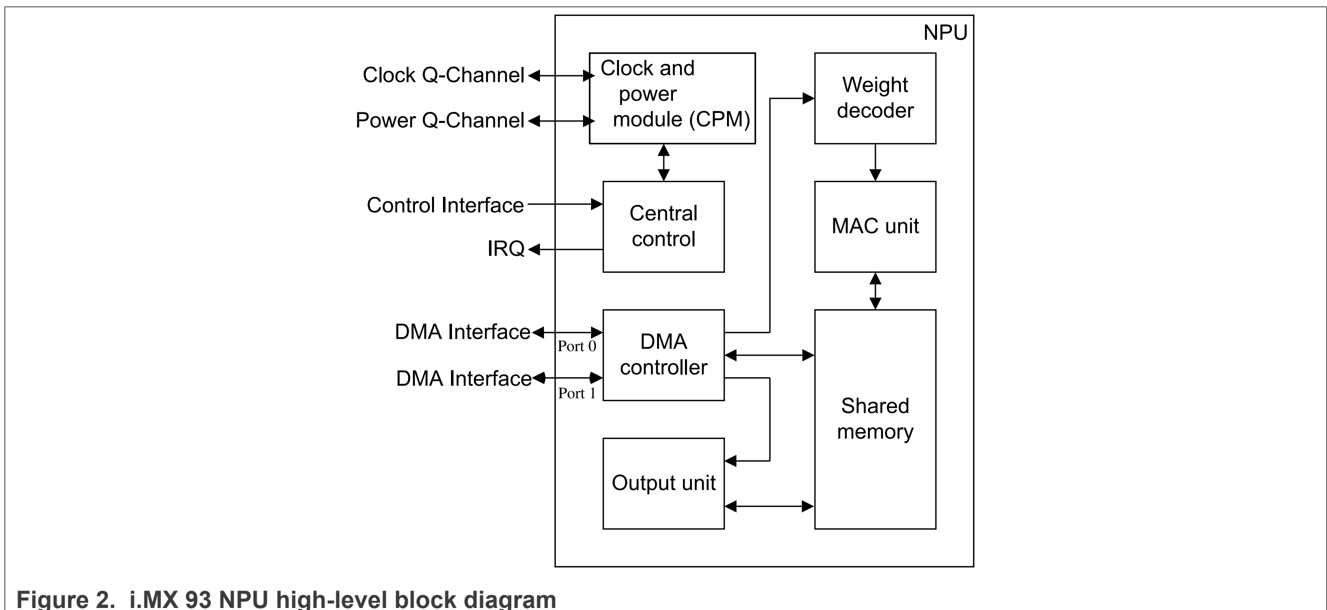

Figure 2. i.MX 93 NPU high-level block diagram

Table 2. i.MX 93 NPU functional blocks

i.MX 93 NPU block	Description
Clock and power module (CPM)	Handles hard and soft resets, contains registers for the current security settings, the main clock gate, and the QLPI interface
Central control	Controls how the NPU processes neural networks, maintains synchronization, and handles data dependencies
DMA controller	Manages all transactions that use the Arm AMBA 5 AXI interfaces
Weight decoder	Reads the weight stream from the DMA controller. The decoder decompresses and stores this stream in a double-buffered register, ready for the MAC unit to consume it
MAC unit	The MAC unit performs multiply-accumulate operations that are required for convolution, depth-wise pooling, vector products, and the max operation required for max pooling

All information provided in this document is subject to legal disclaimers.

© 2023 NXP B.V. All rights reserved.

Table 2. i.MX 93 NPU functional blocks*...continued*

i.MX 93 NPU block	Description
Output unit	Reads finished accumulators from the shared RAM and converts them into output activations. This process includes performing scaling for each OFM, adding the bias to values, and applying the activation function to each point
Shared memory	Memory is shared between the DMA controller, the MAC unit, and the Output unit

Note:

For the i.MX 93 NPU supported operator list, refer to https://www.nxp.com.cn/docs/en/user-guide/IMX-MACHINE-LEARNING-UG.pdf— Supported ML operators and constraints.

2.2 Differences in NPU key features

The following table describes the NPU features of i.MX 8M Plus and i.MX 93.

Table 3. NPU features of i.MX 8M Plus and i.MX 93

Feature	i.MX 8M Plus	i.MX 93
Host	Cortex-A53	Cortex-M33
NPU IP	VIP8000Nano	Ethos-U65
Device node name	/dev/galcore	/dev/ethous0
Primary APIs	OpenVX with NN Extensions	Ethos-U operator
MAC per cycle	1152	256
Clock	1000 MHz	1000 MHz

2.3 Ethos-U subsystem overview

The i.MX 8M Plus NPU is attached to the AXI-BUS and the Cortex-A core controls it, whereas the Cortex-M core controls the i.MX 93 NPU Ethos-U65. This i.MX 93 machine learning system involves several hardware components working collaboratively to support the acceleration of the tensor computation of an ML model: Cortex-A, Cortex-M, messaging unit (MU), and Ethos-U NPU.

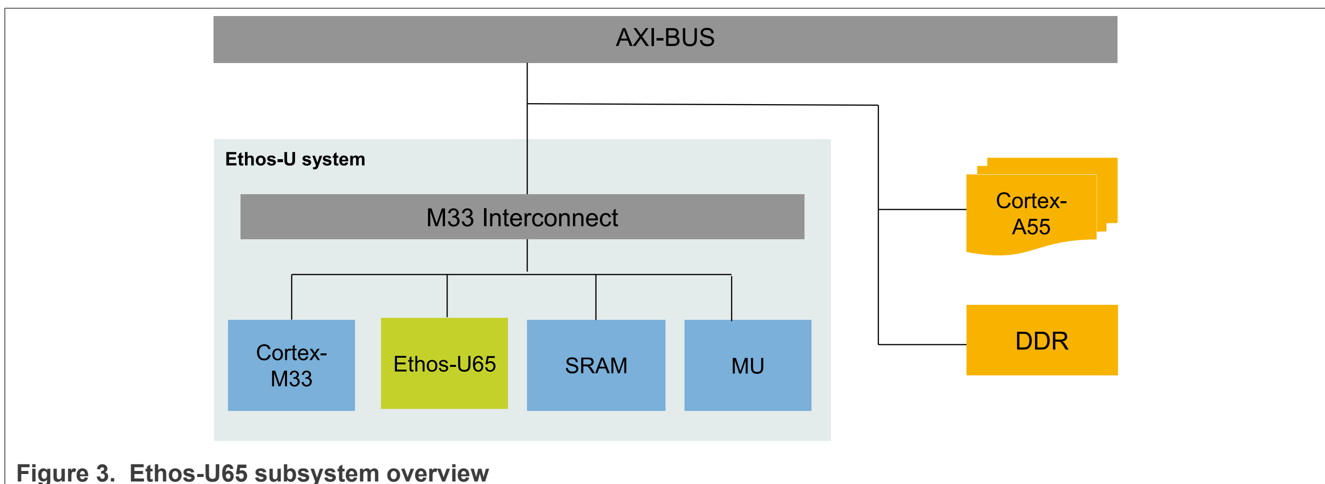

Figure 3. Ethos-U65 subsystem overview

The Cortex-A55 is responsible for loading the ML model, capturing, and pre-processing the dynamic inputs with Linux OS and rich libraries. The Cortex-M is the controller of the attached Ethos-U NPU. It prepares the offloading descriptor for the NPU and triggers the NPU execution. It also provides the unsupported kernel

All information provided in this document is subject to legal disclaimers.

© 2023 NXP B.V. All rights reserved.

execution for NPU. The MU is the message unit IP to facilitate the core communication between Cortex-A and Cortex-M.

- Supports TensorFlow Lite (TFLite) inference with fallback to Cortex-A
- Supports TensorFlow Lite Micro (TFLite-Micro) inference with fallback to Cortex-M
- Supports the inference API to offload the entire model to TFLite-Micro and NPU on Cortex-M
- Supports TFLite API to offload the customized “ethos-u” operator to NPU on Cortex-M
- Provides Vela model tool to optimize the model performance and memory usage for the Ethos-U65 target

2.4 Ethos-U software architecture

Figure 4 shows the three main components of the software required for Ethos-U support.

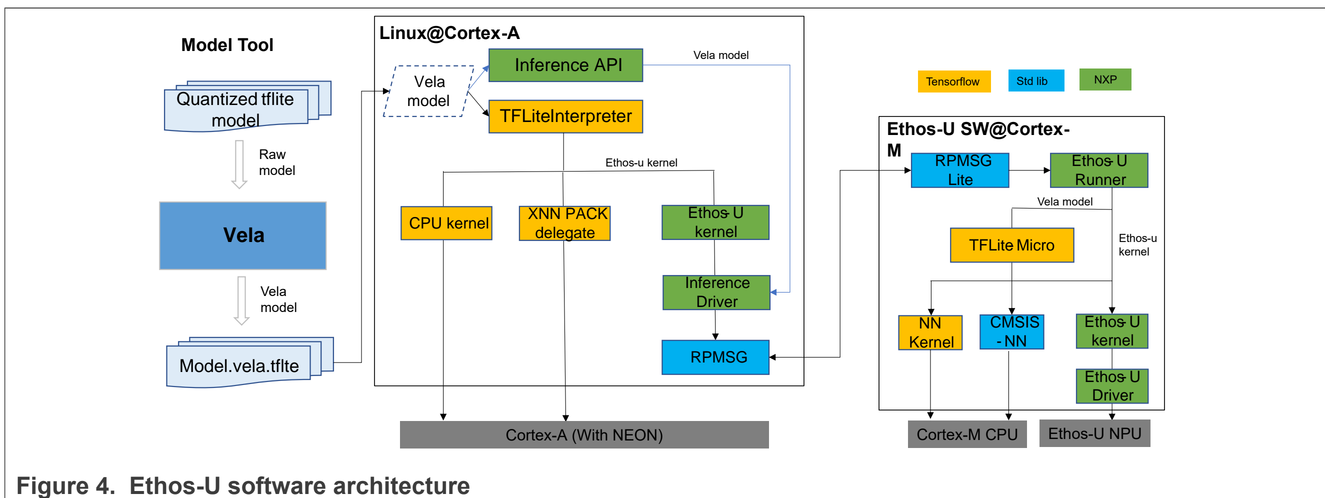

Figure 4. Ethos-U software architecture

- Vela model compiler: offline tool to compile the TFLite model graph for Ethos-U. The compiler replaces supported operators in the model with a custom “ethos-u” operator containing the command stream for Ethos-U NPU. The output of the compiler is a modified TFLite model graph for TFLite/TFLite-Micro inference engines.
- Cortex-A software stack for Linux: contains MPU inference engine (TensorFlow Lite), driver library, and kernel-side device driver for the Linux kernel
- Cortex-M software stack: contains MCU inference engine software (TFLite-Micro, CMSIS-NN) and NPU driver

The typical inference workflow is as follows:

1. Converts the TFLite model into a Vela model using the Vela model compiler and generates the optimized version for Ethos-U NPU.
2. The optimized model is fed to either of the following:
   a. TFLite inference engine, which recognizes the custom “ethos-u” operator, allocates the buffer for input/output feature map (IFM/OFM) and executes the operator via the Ethos-U Linux driver.
   b. Inference API, which allocates the buffer for the input/output feature map and sends the entire model via the Ethos-U driver.
3. The Ethos-U driver composes the inference task message and sends it over RPMsg to Cortex-M.
4. The Ethos-U runner on Cortex-M dispatches the task to the TFLite-Micro or Ethos-U driver directly according to the task type.
   a. If the task type is accelerating the “ethos-u” operator (using the TFLite), the Runner calls the Ethos-U driver directly.
   b. If the task type is accelerating the entire model (using the Inference API), the Runner dispatches the model to TFLite-Micro and further calls the Ethos-U driver for processing.

All information provided in this document is subject to legal disclaimers.

© 2023 NXP B.V. All rights reserved.

5. After the Ethos-U driver completes the inference task, it writes the result into the output features map buffer and sends the response back to Cortex-A via RPMsg.

Note:

The model is loaded from Cortex-A and shared with Cortex-M over RPMsg. The Cortex-M software is prebuilt with both the model and Ethos-U operator acceleration capabilities in a single-binary firmware. This firmware is integrated into Yocto rootfs and is loaded automatically when the user starts an inference task using the TFLite or Inference API by opening the Ethos-U device.

3 Migrating TFLite applications from i.MX 8M Plus to i.MX 93

This section describes the migration workflow for the TFLite applications from i.MX 8M Plus to i.MX 93 using a few examples.

3.1 TensorFlow Lite software stack

Figure 5 shows the TensorFlow Lite software stack. The TensorFlow Lite supports computation on the following hardware units:

- CPU Arm Cortex-A cores
- GPU/NPU hardware accelerator using the VX delegate
- NPU hardware acceleration on i.MX 93 NPU

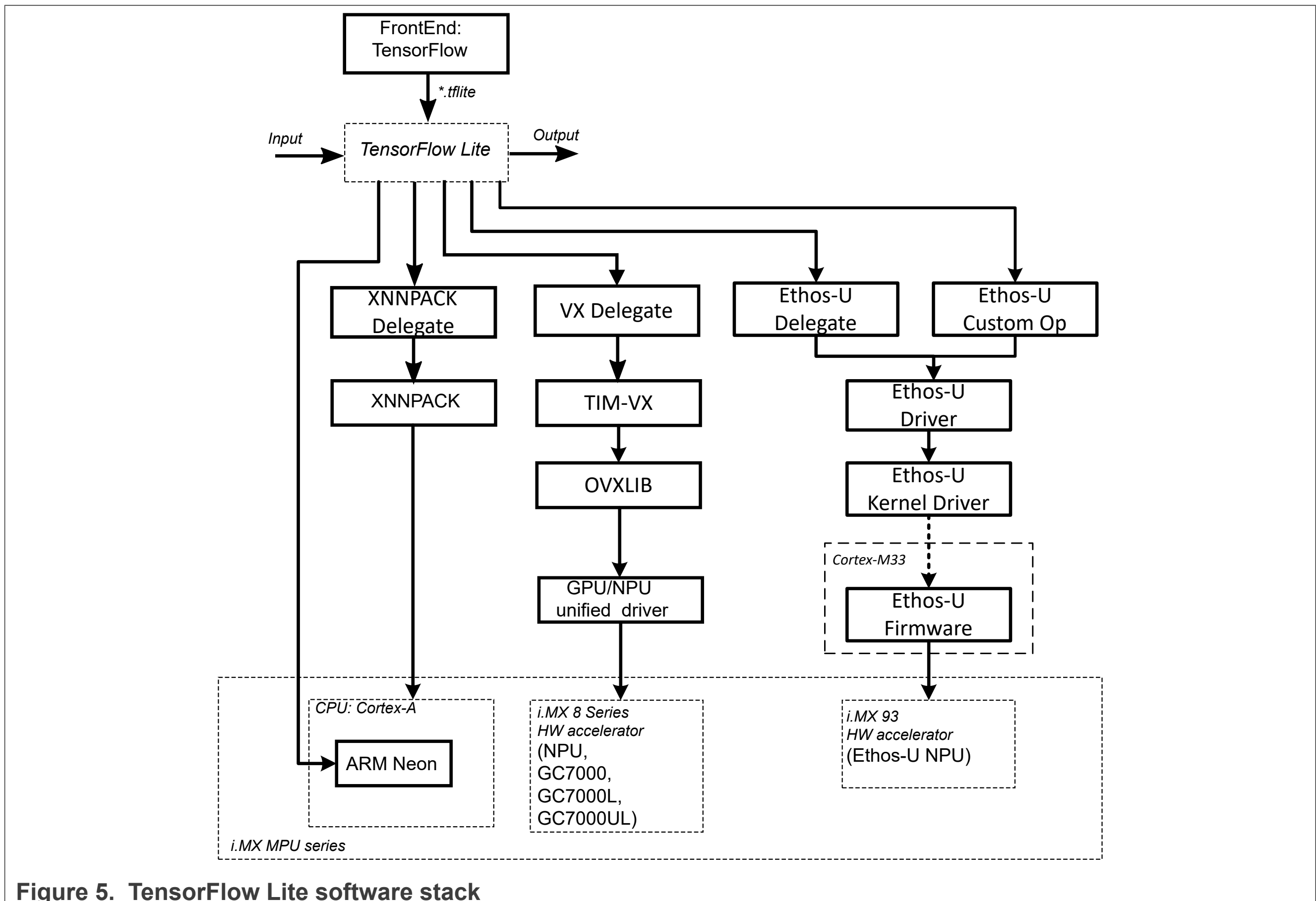

Figure 5. TensorFlow Lite software stack

All information provided in this document is subject to legal disclaimers.

© 2023 NXP B.V. All rights reserved.

Note: *i.MX 8M Plus inference back end can choose CPU/GPU/NPU. However, i.MX 93 does not have a GPU, and if it uses the CPU to do inference, APP does not make any changes. Therefore, the NPU acceleration usage is only discussed in this document.*

3.2 TensorFlow Lite workflow for i.MX 8M Plus / i.MX 93

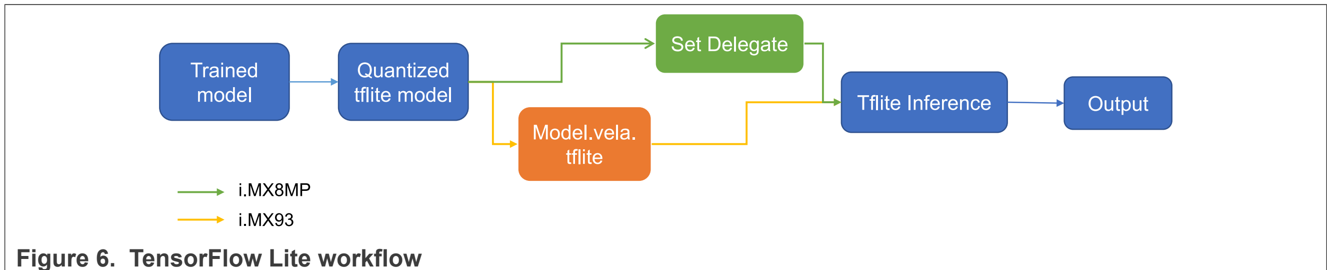

Figure 6. TensorFlow Lite workflow

Both i.MX 8M Plus and i.MX 93 support TensorFlow Lite with NPU acceleration. The major differences are as follows:

- The i.MX 93 NPU software stack depends on the offline tool to compile the TensorFlow Lite model to Ethos-U command stream for Ethos-U NPU execution, while i.MX 8M Plus uses online compilation to generate the NPU commands stream for NPU execution. This means that i.MX 93 NPU users must use the Vela tool to convert the TensorFlow Lite model to the Vela model first. For detail, see Section 4.
- The i.MX 8M Plus uses the TensorFlow Lite external delegate (VX delegate) mechanism to support NPU acceleration, however, i.MX 93 uses the TensorFlow Lite Custom OP mechanism to support NPU acceleration.

In addition, when the i.MX 8M Plus model is deployed on i.MX 93, it is recommended to use PCQ quantization in the model quantization stage to obtain better performance. However, the final model performance depends on the actual application.

3.3 Migration example

When TFLite has to offload the ethos-u operator and fallback to Cortex-A (recommended), the change is minimal. Use Section 4 to compile the quantized TFLite mode, comment out the VX delegate. Afterward, run the ML application of i.MX 8M Plus on i.MX 93 and get NPU acceleration.

3.3.1 NPU accelerate on i.MX 8M Plus

Run an image classification example on i.MX 8M Plus with NPU accelerate.

```
$ cd  /usr/bin/tensorflow-lite-2.9.1/examples
$ USE_GPU_INFERENCE=0 ./label_image -m mobilenet_v1_1.0_224_quant.tflite
 -i grace_hopper.bmp -l labels.txt --external_delegate_path=/usr/lib/
libvx_delegate.so
```

The output of the NPU acceleration on the i.MX 8M Plus processor is as follows:

```
INFO: Loaded model ./mobilenet_v1_1.0_224_quant.tflite
INFO: resolved reporter
Vx delegate: allowed_builtin_code set to 0.
Vx delegate: error_during_init set to 0.
Vx delegate: error_during_prepare set to 0.
Vx delegate: error_during_invoke set to 0.
```


All information provided in this document is subject to legal disclaimers.

© 2023 NXP B.V. All rights reserved.

```
EXTERNAL delegate created.
INFO: Applied EXTERNAL delegate.
W [HandleLayoutInfer:257]Op 18: default layout inference pass.
INFO: invoked INFO: average time: 2.567 ms
INFO: 0.768627: 653 military uniform
INFO: 0.105882: 907 Windsor tie
INFO: 0.0196078: 458 bow tie
INFO: 0.0117647: 466 bulletproof vest
INFO: 0.00784314: 835 suit
```

3.3.2 NPU accelerate on i.MX 93 with TFLite inference engine

Compile the model for Ethos-U using Vela tool, reusing the model *mobilenet_v1_1.0_224_quant.tflite* from */usr/bin/tensorflow-lite-2.9.1/examples/*. If it runs successfully, an optimized Vela model ***mobilenet_v1_1.0_224_quant_vela.tflite*** is generated in the output folder.

```
$ vela ../../tensorflow-lite-2.9.1/examples/ mobilenet_v1_1.0_224_quant.tflite
```

Run the model with the TFLite inference engine (offload the “ethos-u” operator to Cortex-M).

```
$ cd /usr/bin/tensorflow-lite-2.9.1/examples
$ ./label_image -m ../../ethosu/examples/output/
mobilenet_v1_1.0_224_quant_vela.tflite
```

The following is printed if no error occurs:

```
INFO: Loaded mode[ 2712.710545] imx-rproc imx93-cm33: can't change firmware
 while running ../../ethosu/examples/output/
mobilenet_v1_1.0_224_quant_vela.tflite
INFO: resolved reporter INFO: invoked
INFO: average time: 4.433 ms
INFO: 0.780392: 653 military uniform
INFO: 0.105882: 907 Windsor tie
INFO: 0.0156863: 458 bow tie
INFO: 0.0117647: 466 bulletproof vest
INFO: 0.00784314: 835 suit
```

3.3.3 NPU accelerate on i.MX 93 with inference API

Run the model with the inference API (offloads the entire model to TFLite-Micro).

```
$ ./inference_runner -n ./output/ mobilenet_v1_1.0_224_quant_vela.tflite -i
 grace_hopper.bmp -l labels.txt -o output.txt
```

The following is printed if no error occurs:

```
Send capabilities request
Capabilities:
version_status:1
version:{ major=0, minor=0, patch=0 }
product:{ major=6, minor=0, patch=0 }
```


All information provided in this document is subject to legal disclaimers.

© 2023 NXP B.V. All rights reserved.

```
architecture:{ major=1, minor=0, patch=6 }
driver:{ major=0, minor=16, patch=0 }
macs_per_cc:8 cmd_stream_version:0
custom_dma:false
Create network
Create inference
Wait for inferences
Inference status: success
Detected: military uniform, confidence:70
```

4 Vela tool

The Vela tool is used to compile a TensorFlow Lite for microcontrollers neural network (NN) model into an optimized version that can run on an embedded system containing an Arm Ethos-U NPU. The optimized model contains TFLite custom operators for those parts of the model that can be accelerated by the Ethos-U NPU. Parts of the model that cannot be accelerated are left unchanged and run on a CPU (Cortex-A or Cortex-M) using an appropriate kernel (such as the Arm optimized CMSIS-NN kernels). After compilation, the optimized model can only be run on an Ethos-U NPU embedded system. The tool also generates performance estimates for the compiled model.

To deploy the NN model on Ethos-U, the first step is to use the Vela tool to compile the prepared model. To be accelerated by the Ethos-U NPU, the network operators must be quantized to either 8-bit (unsigned or signed) or 16-bit (signed).

4.1 Installing the Vela tool

You can run the Vela tool on the i.MX 93 board or Linux PC. It is already available in NXP Yocto rootfs. This section describes how to install it on the X86 Linux PC. The steps are as follows.

1. Get the Vela source code.

   ```
   $ git clone https://github.com/nxp-imx/ethos-u-vela.git
   ```

2. Install with Python pip.

   ```
   $ cd ethos-u-vela
   $ git checkout lf-5.15.71_2.2.0
   $ pip3 install
   ```

3. After all the commands are successful, you can use `vela --help` to check if the Vela tool is installed successfully.

   ```
   $ vela --version 3.x.x
   ```

4.2 Compiling the TFLite model

After the Vela tool is installed, the following commands can be used to compile a TFLite model to the optimized version for Ethos-U NPU. The optimized model is stored into the OUTPUT_DIR ("./output" by default). The output file has the suffix _vela.tflite. It is also a TFLite model. After the compilation, Vela outputs the detailed log in the console.

Note: *The Vela tool expects that the TFLite model is quantized already. Vela supports asymmetric quantization to 8 bit (signed and unsigned) and 16 bit (signed), as defined by TFLite. To accelerate model operators with Ethos-U NPU, the input model to Vela has to be quantized. Nonquantized operators fall back to the CPU.*

© 2023 NXP B.V. All rights reserved.

The following provides an example of how to compile a model and shows the corresponding output log.

```
$ vela mobilenet_v1_1.0_224_pb_int8.tflite
```

Output log:

```

Network summary for mobilenet_v1_1.0_224_pb_int8
Accelerator configuration               Ethos_U65_256
System configuration                 internal-default
Memory mode                          internal-default
Accelerator clock                                1000 MHz
Design peak SRAM bandwidth                      16.00 GB/s
Design peak DRAM bandwidth                       3.75 GB/s
Total SRAM used                                377.02 KiB
Total DRAM used                               4293.56 KiB
CPU operators = 0 (0.0%)
NPU operators = 60 (100.0%)
Average SRAM bandwidth                           5.29 GB/s
Input   SRAM bandwidth                          11.71 MB/batch
Weight  SRAM bandwidth                          12.61 MB/batch
Output  SRAM bandwidth                           0.00 MB/batch
Total   SRAM bandwidth                          24.43 MB/batch
Total   SRAM bandwidth            per input     24.43 MB/inference (batch size
 1)
Average DRAM bandwidth                           2.33 GB/s
Input   DRAM bandwidth                           1.77 MB/batch
Weight  DRAM bandwidth                           3.92 MB/batch
Output  DRAM bandwidth                           5.06 MB/batch
Total   DRAM bandwidth                          10.76 MB/batch
Total   DRAM bandwidth            per input     10.76 MB/inference (batch size
 1)
Neural network macs                         572406226 MACs/batch
Network Tops/s                                   0.25 Tops/s
NPU cycles                                    3885202 cycles/batch
SRAM Access cycles                             988663 cycles/batch
DRAM Access cycles                            1835595 cycles/batch
On-chip Flash Access cycles                         0 cycles/batch
Off-chip Flash Access cycles                        0 cycles/batch
Total cycles                                  4619795 cycles/batch
Batch Inference time                 4.62 ms,  216.46 inferences/s (batch size
 1)
```

The following is the computational graph after the model (mobilenet_v1_1.0_224_pb_int8.tflite) is compiled. Here, Vela encapsulates all supported OPs into one Ethos-U OP.

© 2023 NXP B.V. All rights reserved.

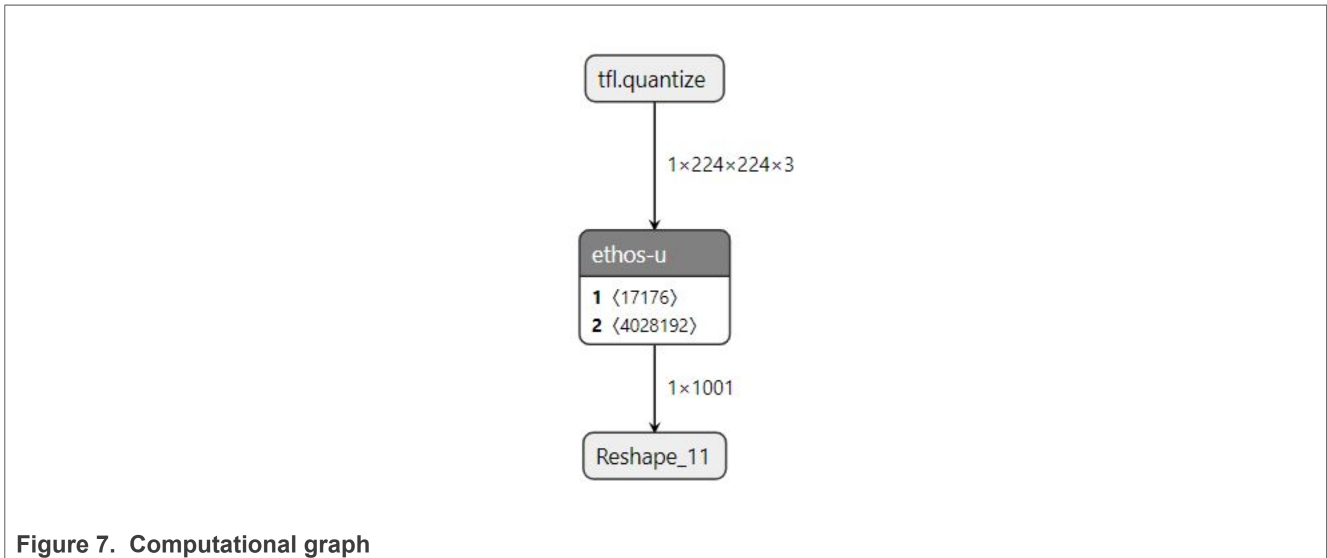

Figure 7. Computational graph

4.3 Memory hierarchy for Cortex-M

For Cortex-M, several types of memory media with different capacity, speed, and cost can be accessed by the CPU. Figure 8 shows the memory hierarchy on i.MX 93 with speed decreasing order.

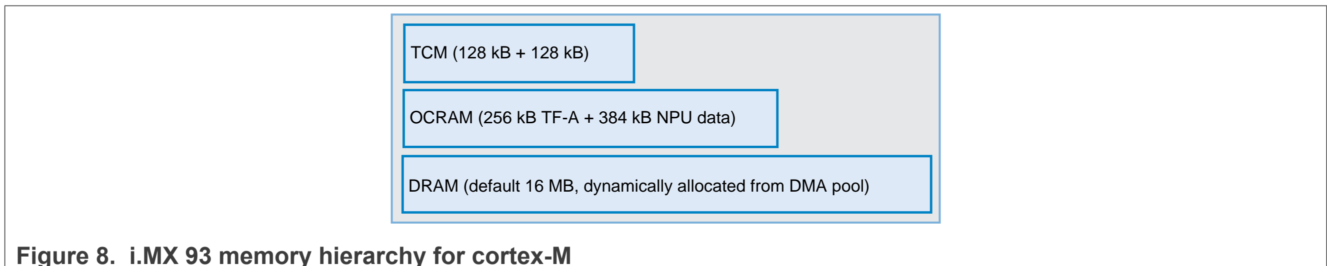

Figure 8. i.MX 93 memory hierarchy for cortex-M

The TCM size is 256 kB, used for Cortex-M runtime data. By design, this memory space is not allocated for the system purpose after booting. How to use it effectively is left for the user decision.

OCRAM size is 640 kB. By design, the first 256 kB is allocated for the Arm trusted firmware (ATF), which used to bootstrap the Cortex-A before the DRAM is available. The rear 384 kB is reserved for NPU data: the weight/ bias of an ML model.

DRAM size is 2 GB on the i.MX 93 EVK board. However, only the shared DMA region between Cortex-A and Cortex-M can be used. The Ethos-U Linux driver requests DMA buffers for Tensor Arena dynamically from the DMA pool and passes the buffer address to the Ethos-U firmware on Cortex-M. If not explicitly specified, a 16 MB DMA buffer is requested by default.

Ethos-U can only access the DRAM and OCRAM memory by design. Figure 9 shows the current memory mapping for Ethos-U firmware.

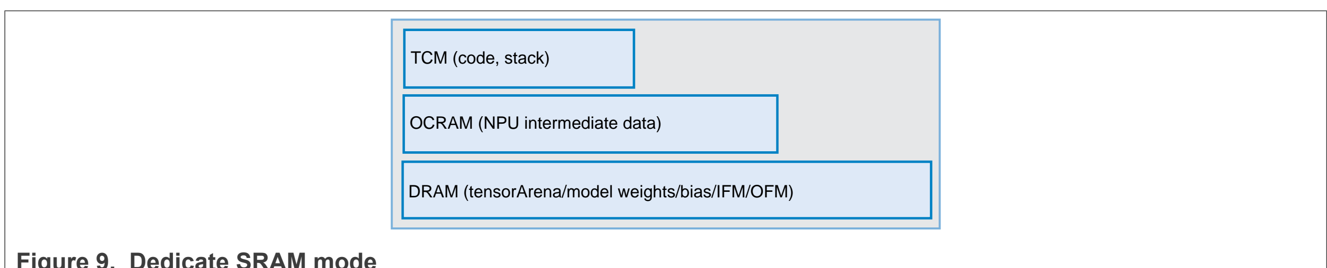

Figure 9. Dedicate SRAM mode

All information provided in this document is subject to legal disclaimers.

© 2023 NXP B.V. All rights reserved.

With this configuration, the model data and tensor arena are allocated in DRAM and the OCRAM is used as an NPU cache. Use "**Dedicated_Sram**" memory mode for model compilation with Vela (vlea.ini can be found in *ethos-u-vela/ethosu/config_files*):

```
$ vela --accelerator-config ethos-u65-256 --system-config Ethos_U65_High_End --
memory-mode Dedicated_Sram --config vela.ini {tflite-model}
```

For a standalone Cortex-M application, the memory mapping is as follows:

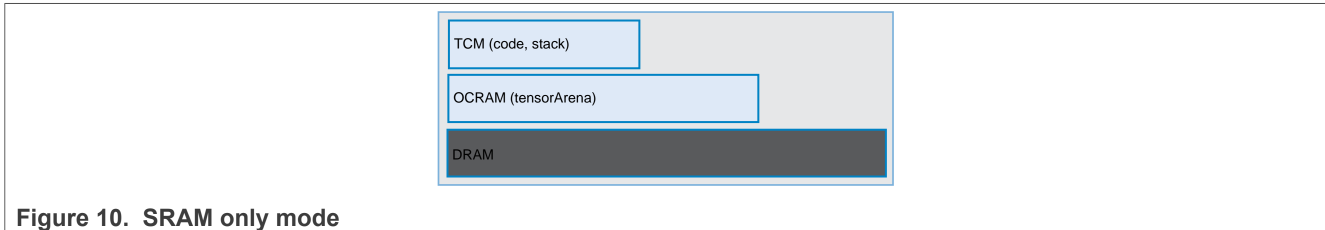

Figure 10. SRAM only mode

With this configuration, no DRAM is used. All the model data and tensor arena memory for NPU is allocated in OCRAM. Use "**Sram_Only**"memory mode for model compilation with Vela:

```
$ vela --accelerator-config ethos-u65-256 --system-config Ethos_U65_High_End --
memory-mode Sram_Only --config vela.ini {tflite-model}
```

5 Hardware acceleration with Ethos-U on i.MX 93 platform

The Ethos-U65 is an NPU on i.MX 93, which supports user space Inference APIs.

- **TFLite API** to offload ethos-u operator and fallback to Cortex-A, nonintrusive
- **Arm inference API** to offload Vela model and fallback to Cortex-M

5.1 Inference with TFLite

The Ethos-U custom operator enables accelerating the inference on the Ethos-U accelerator. The OP directly uses the hardware accelerator driver to use the accelerator capabilities fully.

See Section 3.3.2 for an example.

5.2 Inference with Ethos-U inference API

The Ethos-U inference API provides the methods to use the Ethos-U NPU on the Linux OS without the TensorFlow Lite inference engine. It takes the compiled model and IFM/OFM as inputs, composes an inference task, and dispatches the inferences to the Cortex-M with Ethos-U.

The Ethos-U driver provides the C++ APIs for dispatching the inference to the Ethos-U kernel driver. The library and the corresponding header file are available on Yocto rootfs and SDK.

- /usr/include/ethosu.hpp
- /usr/lib/libethosu.so

5.2.1 How to use the inference API (C++)

The following steps describe how to run a Vela model from Cortex-A.

© 2023 NXP B.V. All rights reserved.

1. Create the inference device.

```
device = Device("/dev/ethosu0")
```

2. Load the model into a buffer from the Vela model file.

```
shared_ptr model_buf = allocAndFill(device, vela_model);
```

3. Create the Network instance with the model buffer.

```
shared_ptr network = make_shared(device, model_buf);
```

4. Load the IFM from the input file (such as a picture for an image classification application) into a buffer. If there are multiple inputs, create the buffers one by one and push back to a vector.

```
vector<shared_ptr> ifm;
ifm_size = network->getIfmDims()[0];
ifm_buf = make_shared(device, ifm_size);
memcpy(ifm_buf ->data(), input_data, input_size);
ifm.push_back(ifm_buf)
```

5. Create the OFM buffers according to the output dimensions in the model. If there are multiple outputs, create the buffer one by one and push back to a vector.

```
vector<shared_ptr> ofm;
ofm_size = network->getOfmDims()[0];
ofm_buf = make_shared(device, ofm_size);
ofm.push_back(ofm_buf);
```

6. Create an inference instance with the Network buffer, IFM buffer, and OFM buffer.

```
inf = make_shared(net, ifm.begin(), ifm.end(), ofm.begin(), ofm.end());
```

7. Call Inference->invoke() to trigger and wait for the completion of the inference.

```
task. Inf->invoke()
```

8. Access the OFM buffers to get the inference result.

```
Outputs = inf->getOfmBuffers()
```

5.2.2 How to use the inference API (Python)

In addition to the C++ API, the Ethos-U driver also provides the Python API.

It is installed into Yocto rootfs: */usr/lib/python3.10/site-packages/ethosu*.

© 2023 NXP B.V. All rights reserved.

Example usage:

```
import ethosu.interpreter as ethosu
# loading the vela model file into interpreter
interpreter = ethosu.Interpreter(args.vela_model_file)
# get the input and output dimensions
inputs = interpreter.get_input_details()
outputs = interpreter.get_output_details()
# resize the input according to the model input dimensions
w, h = inputs[0]['shape'][1], inputs[0]['shape'][2]
img = Image.open(args.image).resize((w, h))
data = np.expand_dims(img, axis=0)
# associcate the input data with interpreter
interpreter.set_input(0, data)
# invoke the inference, this is a blocking API, timeout is 60s
 interpreter.invoke()
# get back the inference results, different models have different      #
 results.
# Check the model output dimensions and get all the outputs with index.
 output_data = interpreter.get_output(0)
```

5.3 Building and deploying the Ethos-U firmware

This section describes how to to build and deploy the Ethos-U firmware.

5.3.1 Getting the source

The ethos-u-core-software is part of the i.MX 93 Ethos-U NPU machine learning software package, which is an optional middleware component of the MCUXpresso SDK. The ethos-u-core-software is integrated into the MCUXpresso SDK Builder delivery system available on mcuxpresso.nxp.com. To include Ethos-U NPU machine learning into the MCUXpresso SDK package, the ethos-u-core-software middleware component is selected in the software component selector on the SDK Builder page when building a new package.

Figure 11 shows the SDK Builder page.

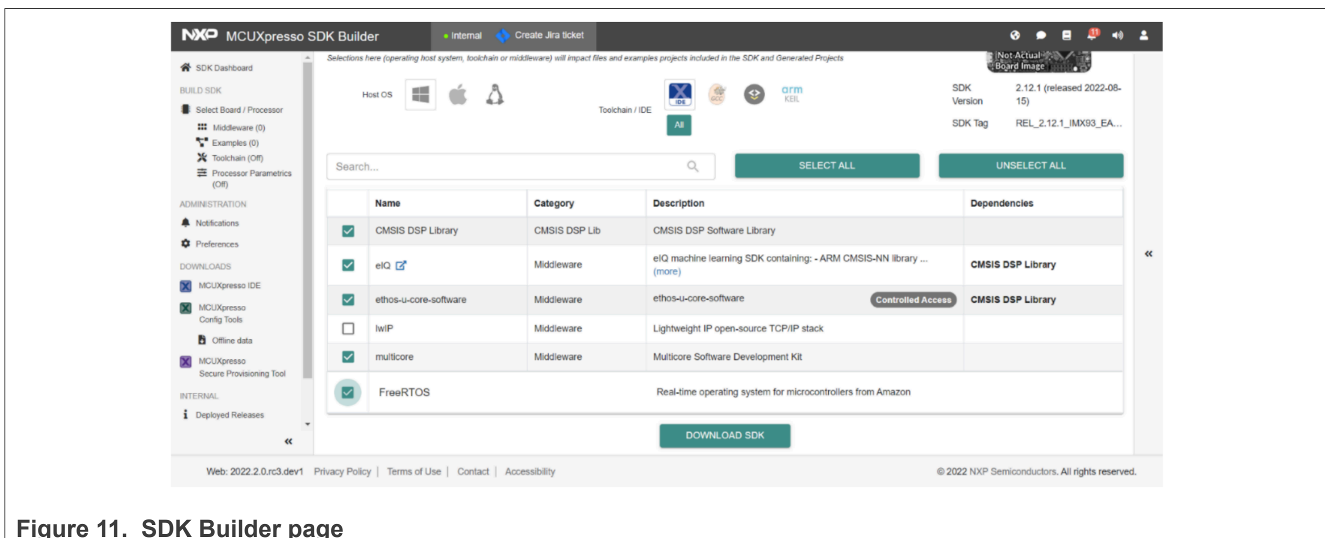

Figure 11. SDK Builder page

Once the MCUXpresso SDK package is downloaded, it can be extracted on a local machine or imported into the MCUXpresso IDE. For more information on the MCUXpresso SDK folder structure, refer to the *Getting*

All information provided in this document is subject to legal disclaimers.

© 2023 NXP B.V. All rights reserved.

Started with MCUXpresso SDK User's Guide (Document ID: MCUSDKGSUG). The package directory structure is similar as follows.

```
<MCUXpresso-SDK-root>
|-- boards
|   -- <board>
|      -- demo_apps      - Example build projects
|         -- ethosu_apps_rpmsg - Ethos-U default firmware
with rpmsg
|         -- ethosu_apps - Ethos-U standalone app example
|
|-- middleware/ethos-u-core-software
-- applications - The inference process APIs
-- boards - The board related initialization and configuration files
-- core_driver - Ethos-U core driver which includes reading/writing registers
-- examples - Ethos-U example applications
-- ethosu_apps_rpmsg - Ethos-U default firmware with rpmsg
-- ethosu_apps - Ethos-U standalone app example
```

5.3.2 Ethos-U example applications

This section describes the Ethos-U example applications and supported toolchains.

5.3.2.1 Introduction

The two Ethos-U applications are available as follows:

- ethosu_apps_rpmsg: firmware for Yocto Linux BSP
- ethosu_apps: standalone example for Cortex-M

The ethosu_apps_rpmsg is the firmware for the Ethos-U subsystem for Linux OS. It contains core message handling, inference request processing from the Cortex-A core, NPU's registers configuration, inference execution, and inference result providing to the Cortex-A core. The supported inference engine is TFLite or TFlite-Micro (if the inference API is used).

The example ethosu_apps is a Cortex-M standalone app that demonstrates the inference execution entirely on the Cortex-M core that can be used in the low-power scenario with the Cortex-A sleeping. The example uses a conv2d op model. There is no core message handling and only supports TFLite-Micro. The apps are available in the /boards//demo_apps/ethosu_apps* folders.

5.3.2.2 Toolchains supported

- IAR Embedded Workbench for Arm when the project is opened in IAR, press the "Make" button to build the project in IAR as follows.

© 2023 NXP B.V. All rights reserved.

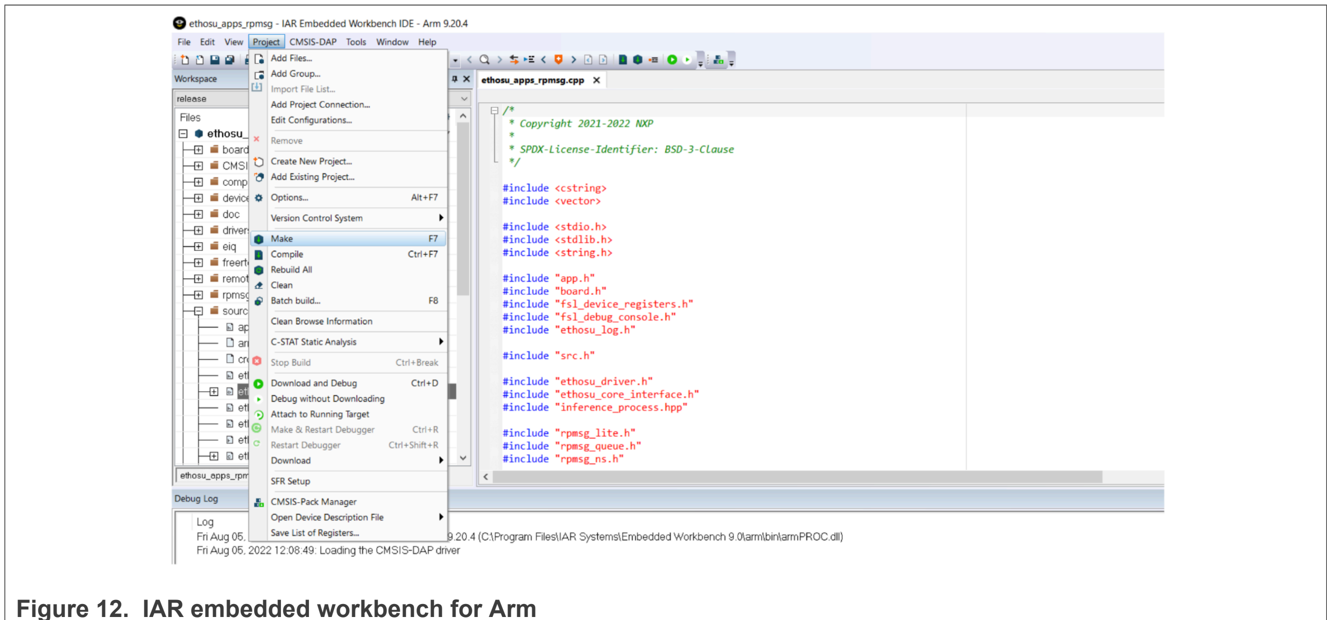

Figure 12. IAR embedded workbench for Arm

- ArmGCC - GNU tools Arm embedded
  Run the following command to build the project.

```
$ cd mcu-sdk-2.0/boards/mcimx93evk/demo_apps/ ethosu_apps_rpmsg/armgcc $ export
 ARMGCC_DIR=${YOUR_TOOLCHAIN_LOC}/gcc-arm-noneeabi-10-2020-q4-major
$ export PATH=$PATH:${YOUR_TOOLCHAIN_LOC}/gcc-arm-noneeabi-10-2020-q4-major/bin
$ ./build_release.sh
```

5.3.2.3 Deploy procedure

1. Deploy the ethosu_apps_rpmsg firmware. Example ethosu_apps_rpmsg is built as .out or .elf and installed in rootfs as the name of “ethosu_firmware”. The prebuilt binary is integrated in the rootfs and loaded by the Linux Ethos-U driver upon an inference request. To rebuild the firmware, the rebuilt ethosu_apps_rpmsg.out or ethosu_apps_rpmsg.elf should be copied to /lib/firmware/ in rootfs and renamed as the name of “ethosu_firmware” as follows:

```
$ cp ethosu_apps_rpmsg.elf ./lib/firmware/ethosu_firmware
```

2. Deploy the ethosu_apps with U-Boot.
   The ethosu_apps is built as .bin. In the U-Boot terminal, you can run the following command to do inference for the conv2d op model.

```
=> tftp 0x80000000 ethosu_apps.bin;cp.b 0x80000000 0x201e0000 0x20000;
bootaux 0x201e0000 0
```

When the example runs, the log and inference result is displayed on the Cortex-M terminal as follows:

```
Initialize Arm Ethos-U
Inference status: success
```

Note:

All information provided in this document is subject to legal disclaimers.

© 2023 NXP B.V. All rights reserved.

The default firmware ethosu_apps_rpmsg contains the following operators support with TFLite-micro on Cortex-M33: Ethos-U, TFLite_Detection_PostProcess, and Dequantize. If an operator is supposed to fall back on Cortex-M33 but not included, rebuild the source code and deploy the firmware. The ethosu_apps is a standalone Cortex-M application running without Cortex-A interactions, therefore, it is deployed at the U-Boot stage.

5.3.3 Using the Ethos-U on Cortex-M

The Ethos-U NPU on i.MX 93 is accessible by the TFLite-Micro library. The TFLite-Micro interprets the optimized Vela model and delegates the kernels to different execution providers.

Currently, three types of execution providers are supported:

- **NN kernel:** default kernel implementation provided by TFLite-Micro for Cortex-M CPU.
- **CMSIS-NN kernel:** optimized kernel implementation by Arm using the CMSIS-NN library. The CMSIS-NN library executes the kernel on the Cortex-M CPU or Ethos-U.
- **Ethos-U kernel:** kernel implementation for the custom Ethos-U operator. This operator is registered in the TFLite-Micro framework and executes the computation on Ethos-U using the NPU driver.

5.3.3.1 Running Vela model with TFLite-Micro

The following provides the steps to run the Vela model on Cortex-M directly.

1. Get the flatbuffer Vela model.

```
const tflite::Model* model = tflite::GetModel(vela_model);
```

2. Configure / allocate the inputs, outputs tensors statically.

```
constexpr int kTensorArenaSize = 1024 * 1024;
static uint8_t tensorArena[kTensorArenaSize];
```

3. Build the TFLite-Micro interpreter for the inference.

```
static tflite::MicroInterpreter interpreter(
model, //the flatbuffer model
microOpResolver, //resolve to kernel implementers
tensorArena, // tensor memory address
kTensorArenaSize, //tensor memory length
microErrorReporter); //error reporter
```

4. Set the input tensors.

```
// Get access to the input tensor data
TfLiteTensor* inputTensor = interpreter->input(0);
// Copy the input tensor data from an application buffer
for (int i = 0; i < inputTensor->bytes; i++)
inputTensor->data.int8[i] = input_data[i];
```

5. Run the inference and get the output.

```
// Invoke the inference
interpreter->Invoke();
// Get access to the output tensor data TfLiteTensor* outputTensor =
 interpreter->output(0);
// Copy the output tensor data to an application buffer
```


All information provided in this document is subject to legal disclaimers.

© 2023 NXP B.V. All rights reserved.

```
for (int i = 0; i < outputTensor->bytes / sizeof(float32); i ++)
 output_data[i] = outputTensor->data.f[i];
```

TFLite-Micro does not depend on dynamic memory allocation, therefore, it requires users (application developers) to supply a memory arena when an interpreter is created. In practice, the user allocates this memory arena as a static buffer when the program starts. For example:

```
#define TENSOR_ARENA_SIZE (1024 * 1024 * 16)
uint8_t tensorArena[TENSOR_ARENA_SIZE];
```

TFLite-Micro framework uses this memory arena as inputs/outputs/intermediate tensors store. This memory size “TENSOR_ARENA_SIZE” must be adjusted according to the practical usage to consider the following points:

- Model used for the application
- Size of the input/output data
- Memory needed for intermediate result
- Memory arena mapping to SRAM or TCM, considering the effective usage of memory hierarchy

6 Acronym

Table 4 lists and defines the acronyms used in this document.

Table 4. Acronyms

Term	Definition
AHB	Advanced high-performance bus
API	Application programming interface
ATF	Arm trusted firmware
AXI	Advanced eXtensible Interface
BSP	Board support package
CPM	Communications processor module
DMA	Direct memory access
DRAM	Dynamic random-access memory
IFM	Input feature map
MAC	Media access control
NPU	Neural processing unit
OFM	Output feature map
SDK	Software development kit
SIMD	Single instruction / multiple data
SRAM	Static random-access memory
TCM	Trellis-coded-modulation
TFLite	TensorFlow Lite

7 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

© 2023 NXP B.V. All rights reserved.

Copyright 2023 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

8 Revision history

Table 5 summarizes revisions to this document.

Table 5. Revision history

Revision number	Release date	Description
1	18 September 2023	Initial public release

© 2023 NXP B.V. All rights reserved.

9 Legal information

9.1 Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

9.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at http://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

9.3 Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

© 2023 NXP B.V. All rights reserved.

AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, μVision, Versatile — are trademarks and/or registered trademarks of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved.

IAR — is a trademark of IAR Systems AB.

i.MX — is a trademark of NXP B.V.

Microsoft, Azure, and ThreadX — are trademarks of the Microsoft group of companies.

TensorFlow, the TensorFlow logo and any related marks — are trademarks of Google Inc.

© 2023 NXP B.V. All rights reserved.

Contents

1 Introduction 2
2 NPU overview 2
2.1 Block diagram 2
2.2 Differences in NPU key features 4
2.3 Ethos-U subsystem overview 4
2.4 Ethos-U software architecture 5
3 Migrating TFLite applications from i.MX 8M Plus to i.MX 93 6
3.1 TensorFlow Lite software stack 6
3.2 TensorFlow Lite workflow for i.MX 8M Plus / i.MX 93 7
3.3 Migration example 7
3.3.1 NPU accelerate on i.MX 8M Plus 7
3.3.2 NPU accelerate on i.MX 93 with TFLite inference engine 8
3.3.3 NPU accelerate on i.MX 93 with inference API 8
4 Vela tool 9
4.1 Installing the Vela tool 9
4.2 Compiling the TFLite model 9
4.3 Memory hierarchy for Cortex-M 11
5 Hardware acceleration with Ethos-U on i.MX 93 platform 12
5.1 Inference with TFLite 12
5.2 Inference with Ethos-U inference API 12
5.2.1 How to use the inference API (C++) 12
5.2.2 How to use the inference API (Python) 13
5.3 Building and deploying the Ethos-U firmware 14
5.3.1 Getting the source 14
5.3.2 Ethos-U example applications 15
5.3.2.1 Introduction 15
5.3.2.2 Toolchains supported 15
5.3.2.3 Deploy procedure 16
5.3.3 Using the Ethos-U on Cortex-M 17
5.3.3.1 Running Vela model with TFLite-Micro 17
6 Acronym 18
7 Note about the source code in the document 18
8 Revision history 19
9 Legal information 20

Please be aware that important notices concerning this document and the product(s) described herein, have been included in section 'Legal information'.

© 2023 NXP B.V. **All rights reserved.**

For more information, please visit: http://www.nxp.com

Date of release: 18 September 2023
Document identifier: AN13854