Renesas RA Family

Using the Ethos-U NPU with RA8 MCUs

Introduction

The Arm Ethos-U Neural Processing Units (NPUs) are ultra-low-power NPUs designed to accelerate machine learning (ML) inference on microcontrollers. Some Renesas RA8 series MCUs such as the RA8P1, integrate the Ethos-U NPUs. The Ethos-U NPU works alongside the host CPU, providing efficient AI/ML acceleration for applications like computer vision, speech recognition, and anomaly detection.

By offloading ML computations from the CPU to the Ethos NPU, the system achieves higher performance and energy efficiency, enabling AI at the edge without a significant power overhead.

Target Devices

- RA8P1

Required Resources

Software and development tools

- Renesas e^2 studio with RUHMI Platform installed
- Renesas Flexible Software Package (FSP) v6.0.0 (https://github.com/renesas/fsp)

Prerequisites and Intended Audience

Users of this Application Note should have some experience with the Renesas e^2 studio. Additionally, users are required to read the following reference document as prerequisites. Please refer to the Reference section to acquire these documents.

- The Ethos-U (rm_ethosu) section of the FSP User's Manual
- The Arm Ethos-U55 Technical Reference Manual
- Renesas RUHMI (Robust Unified Heterogeneous Model Integration) Framework Quick Start Guide

The intended audience includes all users who are currently developing or planning to develop AI applications using Renesas RA MCUs with the Ethos-U NPU.

Contents

1. Introduction to the Ethos-U NPU

Most Machine Learning (ML) applications rely on Neural Network (NN) inference operations. While these operations can be executed in software on a general-purpose processor, leveraging a hardware accelerator can significantly enhance performance. Ethos-U NPUs are compact, power-efficient processors designed to reduce both inference time and memory requirements for running machine learning neural networks.

Arm delivers the hardware Register Transfer Level (RTL) design of the Ethos-U NPU. MCU vendors can integrate the Ethos-U NPU into their microcontrollers to enhance AI/ML processing capabilities while maintaining low power consumption.

Renesas optimizes both the hardware and software stack by providing tightly coupled memory, caches, efficient data paths, and AI software libraries to maximize the performance of the Ethos-U NPU-accelerated system.

For the Ethos-U NPU parameters that are vendor-configurable, Renesas has selected the optimal configuration for the RA8P1, aiming to make it well-suited for edge AI applications.

Table 1 Ethos-U55 NPU Specification on RA8P1

Parameter	Specifications
Number of parallel 8x8 MAC operations the NPU can perform per NPU clock cycle	256 MACS/cycle
Shared RAM	48KB

The NPU communicates with the host CPU and the MCU internal memory through a level triggered interrupt, this signal is asserted (goes high) when the NPU completes processing or encounters an error. The NPU driver uses the APB bus to configure the NPU. The AXI bus transfers the command stream, weight and bias data, scratch tensor, inputs to the NPU, and outputs from the NPU.

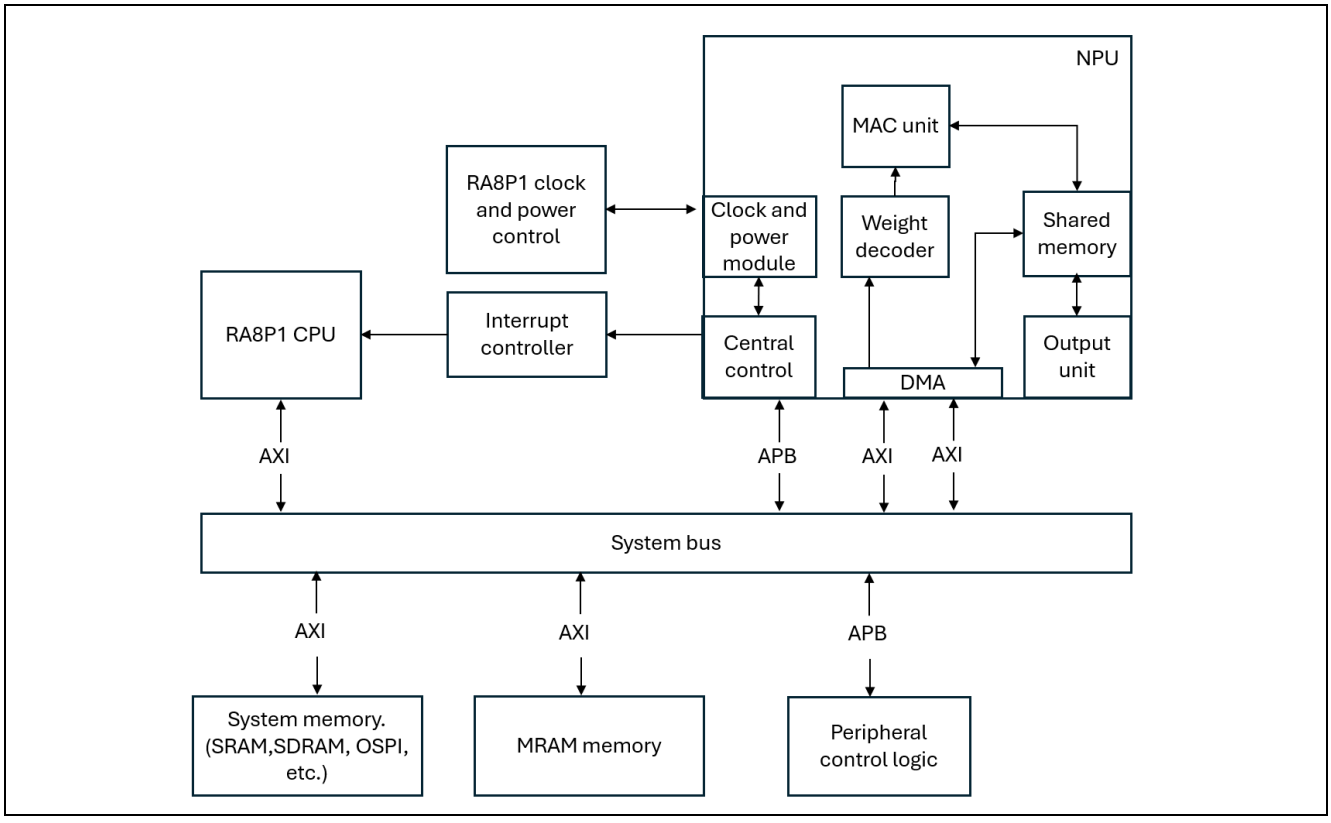


Figure 1. Integration of NPU inside RA8P1

To learn about the architecture and configuration of the NPU, refer to the Arm NPU Technical Reference Manual.

Arm Ethos-U55 Technical Reference Manual

2. Overview of Deploying an AI Model to Ethos-U NPU

Figure 2 depicts the major software components for deploying an AI model. If there is a desire to port a float model to be used by the NPU, the model needs to be quantized first.

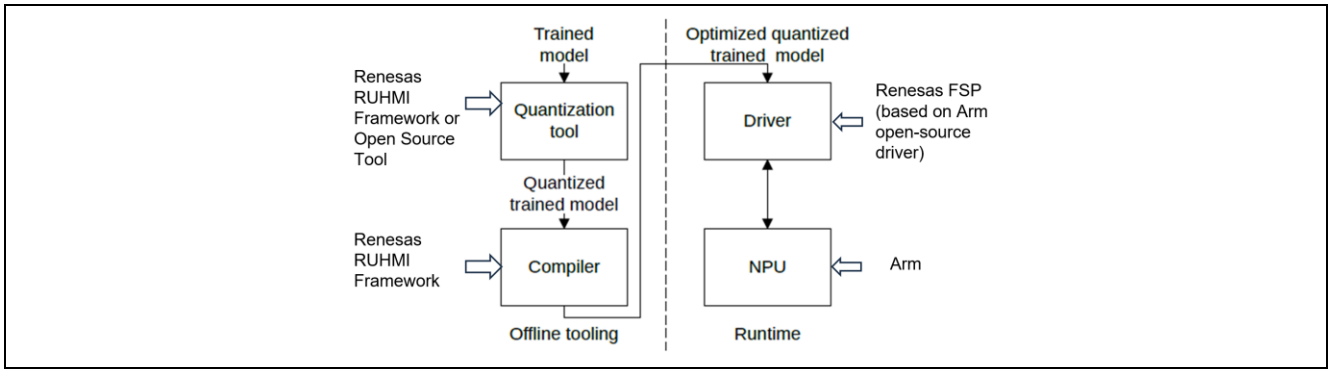


Figure 2. Software Stack for NPU Usage

During an inference, the application invokes the driver, which communicates with the NPU to tell it where the command stream is and to initiate the network traversal. The command stream describes the steps necessary for the NPU to execute the operators compiled into the command stream autonomously. When the inference completes, the NPU raises an IRQ to the driver.

While Arm provides tool support for the Ethos-U NPU, Renesas provides the more advanced tool based the Arm offers:

- Renesas RUHMI Framework: a GUI based tool which offers further optimization during model compilation to improve inference using the Ethos-U NPU. The RUHMI framework also supports improved compilation for inference using the CPU. The RUHMI framework is powered by EdgeCortix® MERA.
- Renesas Flexible Software Package (FSP): FSP v6.0.0 or later integrates TensorFlow Lite Micro with Ethos-U acceleration to simplify the AI application development.

Renesas's highly optimized AI model Compiler and Ethos-U driver are key differentiating factors that enhance the model inference performance. Refer to the Appendix section for model compilation using the open-source solution.

3. Preparing a Model for Embedded Application Deployment

3.1 Operators Supported by Ethos-U NPU

The SUPPORTED_OPS.md file in the following Arm GitHub repository contains a summary table of TensorFlow Lite operators that can be executed on the Ethos-U NPU, along with specific constraints for each operator. If the constraints are not met, then that operator will be scheduled on the CPU instead. Any other TFLite operator not listed will be left untouched and scheduled on the CPU. This resource can be used to evaluate whether a model can be ported to run on the Ethos-U NPU.

Ethos_NPU_Supported_Operators_Through_Vela

To allow NN model processing to be accelerated by the Ethos-U NPU, Tensor Flow Lite Micro (TFLM) supports a feature called custom operators. The definition of scheduling an operator on Ethos-U NPN means during the compilation process (described in section 3), the operator can be converted to the custom NPU operator.

During the compilation of a neural network model, the RUHMI compiler analyzes the model to identify sequences of operators that the Ethos-U NPU can accelerate. It then groups each supported sequence into a custom Ethos-U operator for delegation to the NPU.

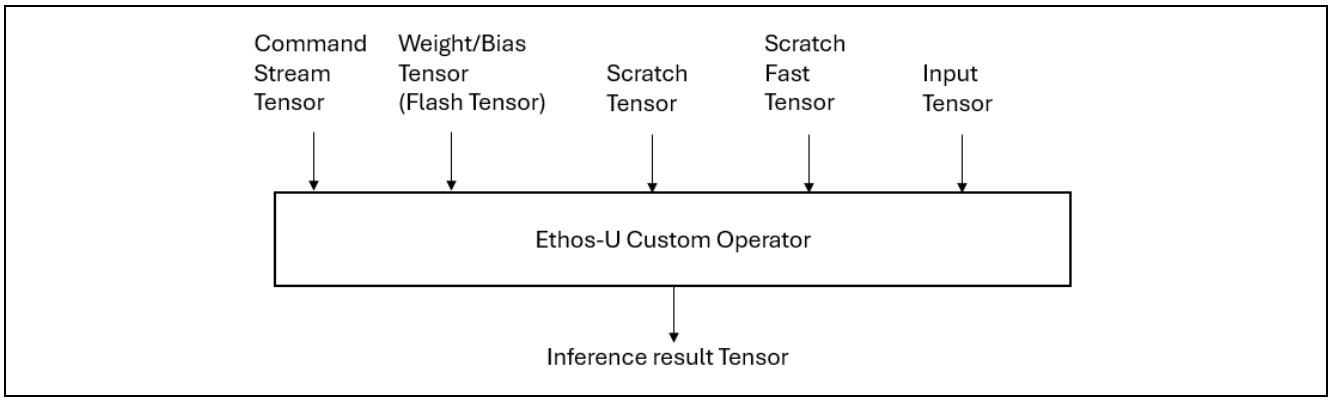


Figure 3. Ethos-U Custom Operator

Additionally, this Arm reference page describes the Data Types and additional operators that can be generated by combining features of the NPU.

Types and Additional Operators supported by the Ethos NPU

When trying to port a model to perform inference using Ethos NPU, if desired, the open-source tool Netron can be used to visualize the operators in the original model. This tool supports various frameworks, including TensorFlow Lite (.tflite), ONNX (.onnx), PyTorch (.pt, .pth), and Keras (.h5).

Using the Netron tool, the float or quantized model can be visualized to observe the original operators that exist in the AI model.

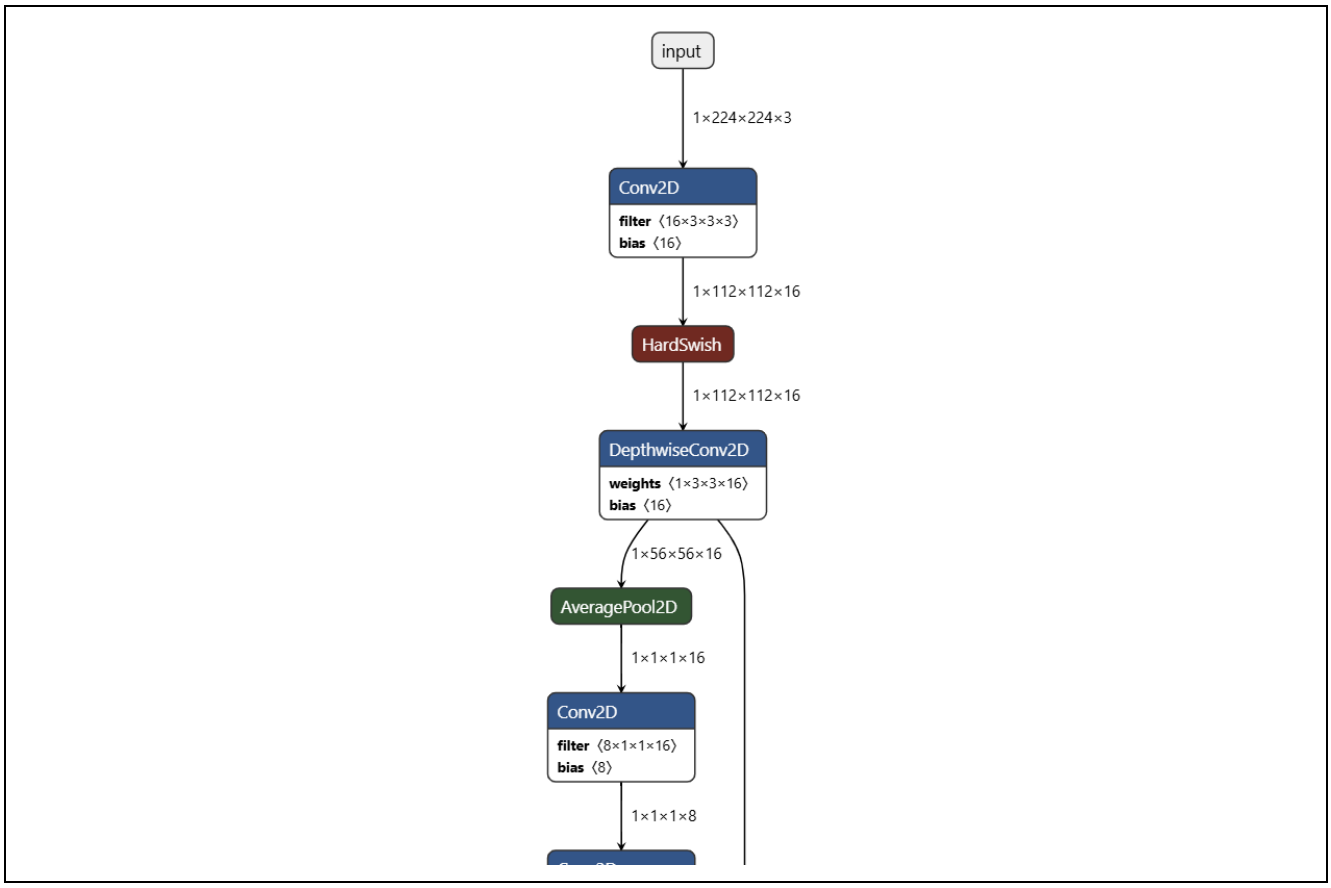


Figure 4. Example Operator Visualization of the Quantized TFLM Model

3.2 Models Supported by Ethos-U NPU

The Ethos-U NPU targets 8-bit and 16-bit integer-quantized Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). The NPU supports 8-bit weights.

The Arm Ethos-U NPU supports different model formats. TensorFlow Lite is natively supported due to Arm's toolchain integration. An ONNX floating-point model can be converted to TFLite, then quantized and compiled for optimized execution on the Ethos NPU. Similarly, a floating-point PyTorch model can be converted to a TFLite model or ONNX model and then compiled for optimized execution on the Ethos NPU.

There are some 8-bit integer-quantized TFLite models for Ethos-U55 are provided at the URL below by Arm Limited. These models can be used for performing inference using the NPU.

The Arm ML Zoo https://github.com/ARM-software/ML-zoo

The models are .tflite files, which is a TensorFlow Lite model file designed for efficient inference on embedded and mobile devices. It is stored in a FlatBuffer format, making it lightweight and fast to load. They are the inputs to the RUHMI framework and will be compiled into .c/.h file to be integrated with the embedded system.

3.3 Compile an AI Model

3.3.1 General Flow

During the compilation process, the compiler performs the following major operations:

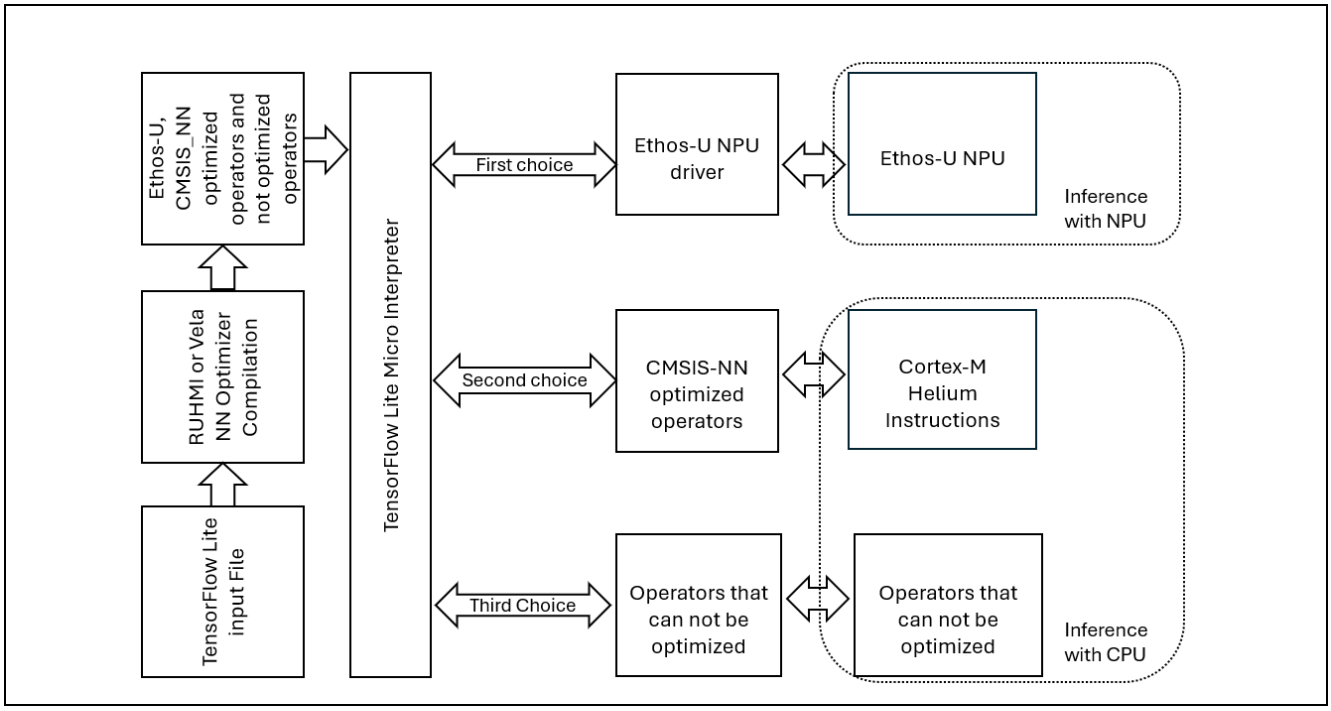


Figure 5. AI Model Operator Dispatch

Key Operations during the compilation:

1. Check each operator (layer) in the model
2. If the operator is supported by the Ethos-U NPU, the compiler assigns the operator to the Ethos-U NPU.
3. If the operator is not supported by the Ethos-U NPU but it is supported by CMSIS-NN, the compiler assigns the operator to CMSIS-NN. When CMSIS-NN is compiled for a processor that supports Helium (such as Cortex-M55 or Cortex-M85), it can leverage these vector instructions to achieve significantly higher performance and energy efficiency for AI inference.
4. If the operator is not supported by either the Ethos-U NPU or CMSIS-NN, the compiler assigns the operator to the CPU using the TFLM reference kernel.

3.3.2 Using the Renesas RUHMI Framework

Renesas RUHMI Framework offers optimized performance compared with the Vela tool result. Following the Renesas RUHMI Framewrok Quick Start Guide to install the RUHMI Framework to the e2studio environment. Example Projects are provided in the RUHMI interface via the “Select Sample AI Application” button. To compile a new model for deployment, select “Use Your Project and AI Model”. Follow the RUHMI Framework Quick Start Guide to understand how to use these two options.

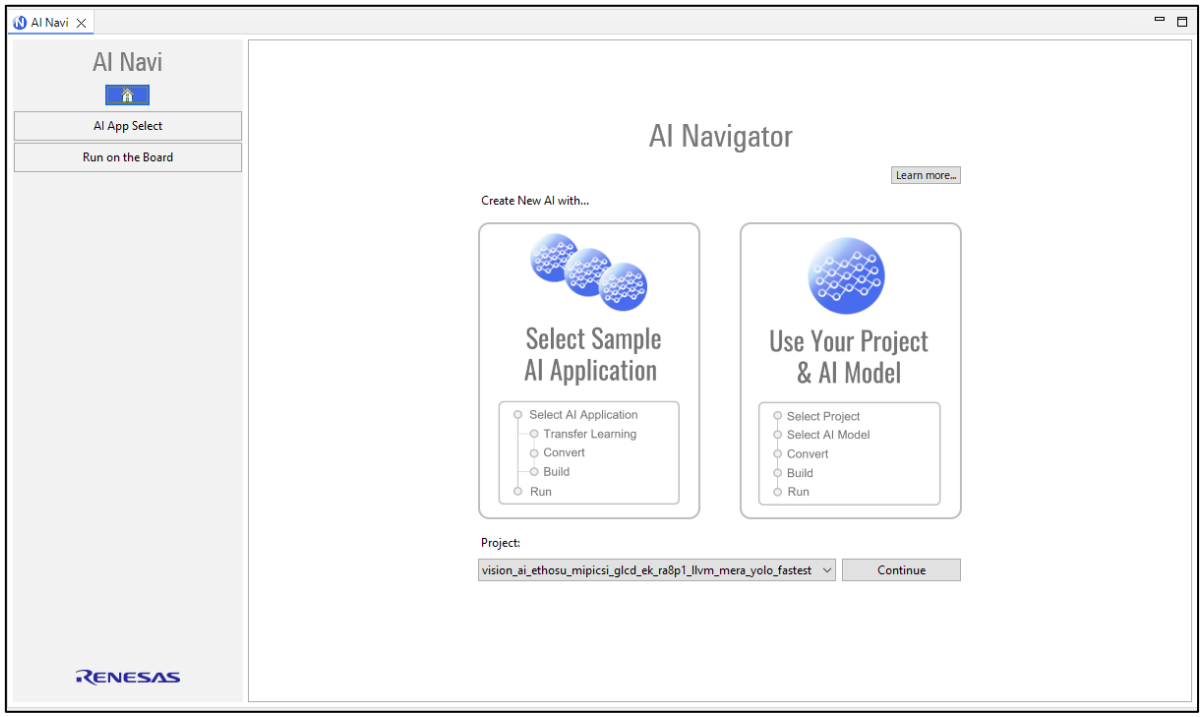


Figure 6. RUHMI Tool User Interface

The following are examples of RUHMI compilation results. During the compilation, the memory usage for SRAM, SDRAM and Flash (MRAM) is calculated and provided in the output file. When using the RUHMI Framework, the ONNX and PyTorch Frontends are currently only meant to be used with Quantizer flow. Please refer to the RUHMI Framework Quick Start Guide documentation to understand more on how to use the output files.

The following is an example of the RUHMI Framework output for Ethos-U NPU inference.

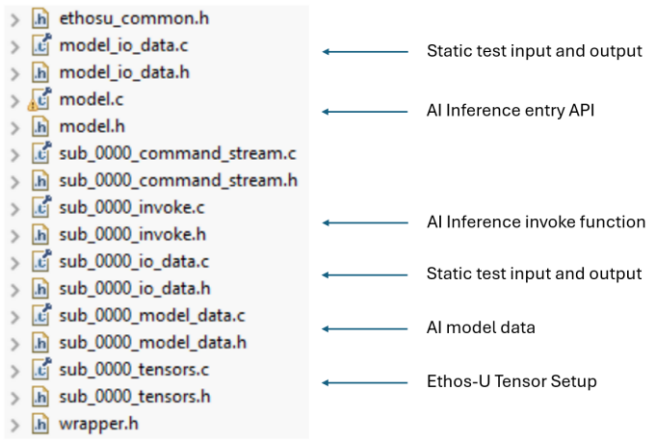


Figure 7. Compilation Result of MobileNet V1 0.25 using Ethos-U for Inference

The following is an example of the RUHMI Framework output for CPU inference.

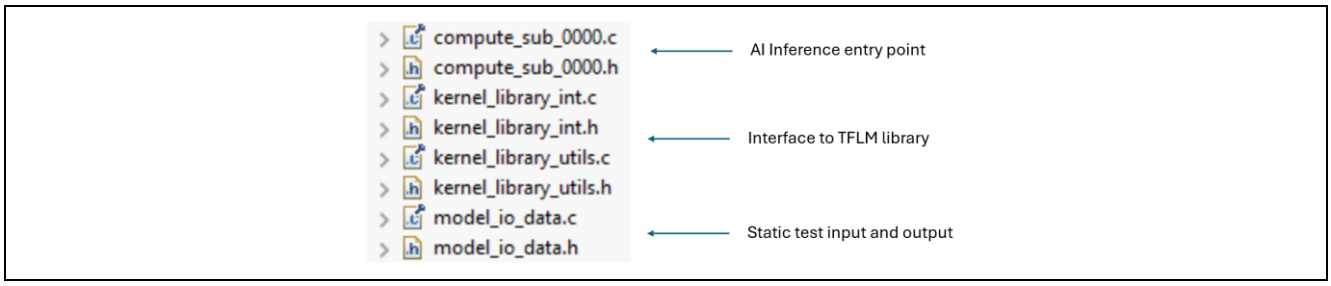


Figure 8. Compilation Result of MobileNet V1 0.25 using CPU for Inference

Refer to R11AN0995 for an example of interfacing these output files with a real time AI inference example.

The static test input and output included in the RUHMI compilation result can be used to establish a test project for quick evaluation of the compilation results. Comparing the output generated from the test input with the output from the compilation result can serve as a method to evaluate inference performance.

During the compilation stage, the MACs/Inference (MACs/Batch) for the model used will be provided at the end. This can be a relative indication of the power consumption when using this model.

4. FSP Stacks Support for Ethos-U NPU

Arm has provided an open-source Ethos-U NPU driver and compiler. Renesas FSP has integrated the driver to interface with the Arm Open-Source TFL Micro Library.

4.1 Overview

Use the following stack when performing AI inference using the Ethos-U NPU or CPU. This stack can be accessed using New Stack -> AI -> Google TFLM Core Lib.

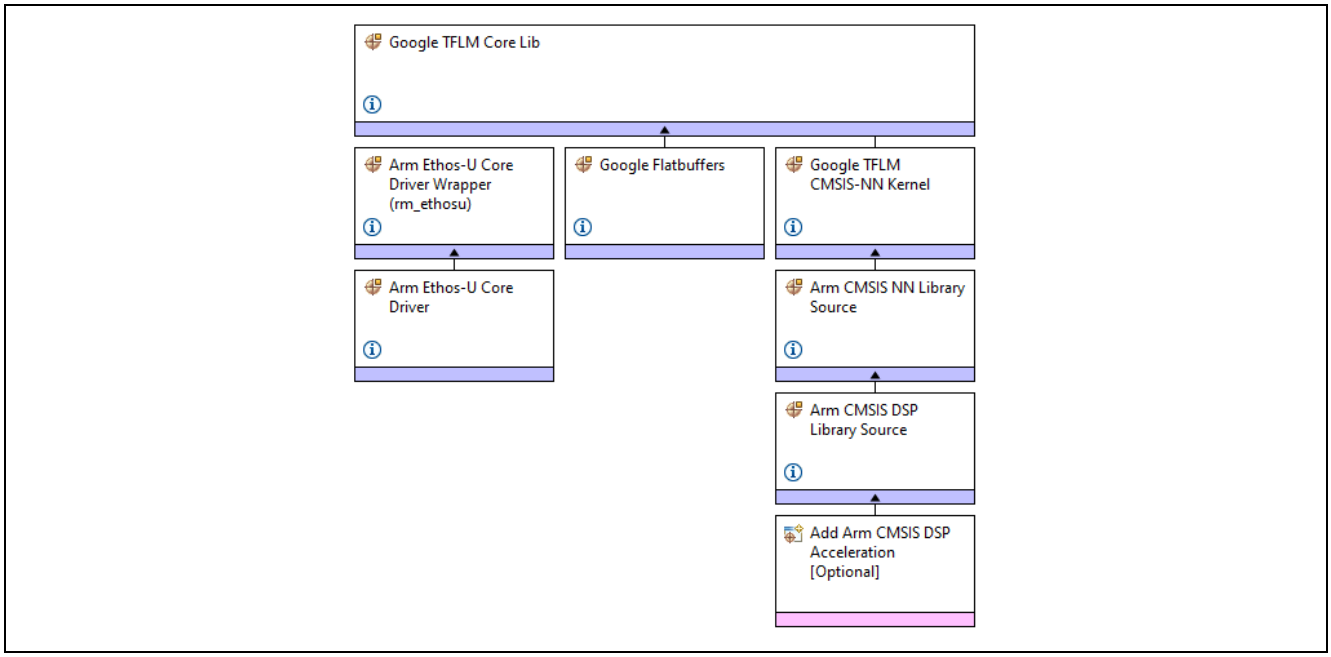


Figure 9. TFLM Core Library for Ethos-U

When the FSP stack shown in **Error! Reference source not found.** is extracted, the following software components will be extracted to the user's application. Refer to section for the instruction of the key software components used in an AI application.

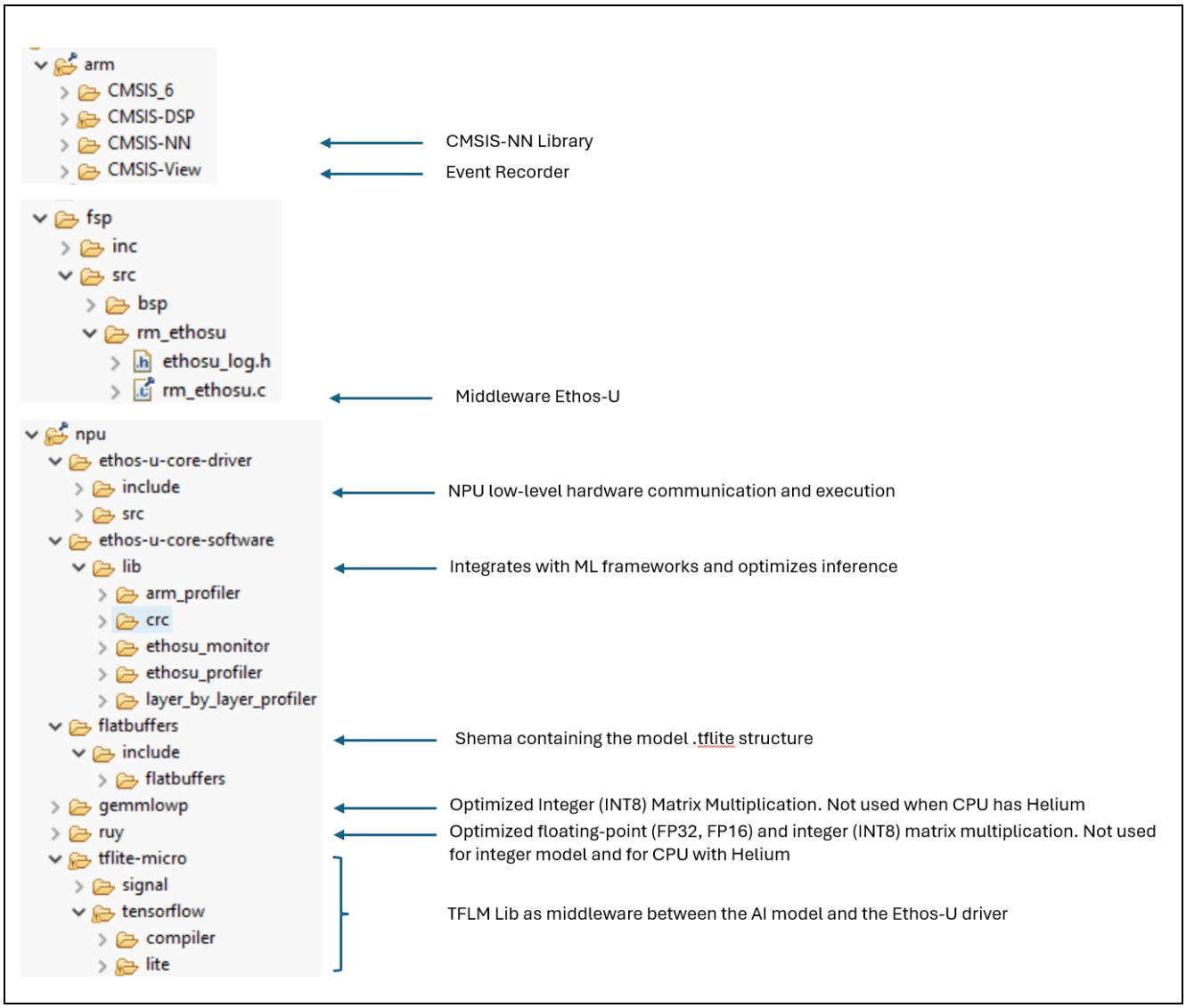


Figure 10. FSP Generated Code for NPU Usage

4.2 FSP Driver Stacks

4.2.1 Google TFLM Core Library

From the TFLM application code, an interpreter is defined to specify the model, the operator resolver, the tensor arena, and its size and to these parameters to TFLM.

- The TFLM interpreter is initialized with a pointer to a pre-compiled model array, avoiding file I/O and reducing memory overhead.
- TFLite has an Ethos-U delegate. During inference TFLM recognizes the Ethos-U custom operator and offloads supported operations (e.g., convolutions, activations, pooling, fully connected layers) to the Ethos-U NPU. If certain operations are not supported by Ethos-U, they are executed on the CPU.
- Notes on CPU Inference:
    - Although the Ethos-U Core Driver Wrapper and Core Driver are not needed, it is recommended to include this stack because the TFLM Core Library uses some of the default debug log utilities provided by the Ethos-U stack.
    - When using the AI model compilation result compiled with the Renesas RUHMI or MERA tool in an embedded system, the FlatBuffer is not used for either Ethos-U Inference or CPU inference. However, since the upstream TFLM library integrated the FlatBuffer utilities, the FlatBuffer stack needs to be included for successful compilation.

- The TFLM Core Library also supports inference using the open-source Vela tool. In this case, it reads an NN model, in the form of .tflite data in memory, and executes out the functions of the NN operators.

4.2.2 Arm Ethos-U Core Driver Wrapper

The following needs to be configured for the Arm Ethos-U Core Driver Wrapper (rm_ethosu) stack. Most of the Properties in this stack are self-explainable, the Callback can be set up to notification user an inference is finished.

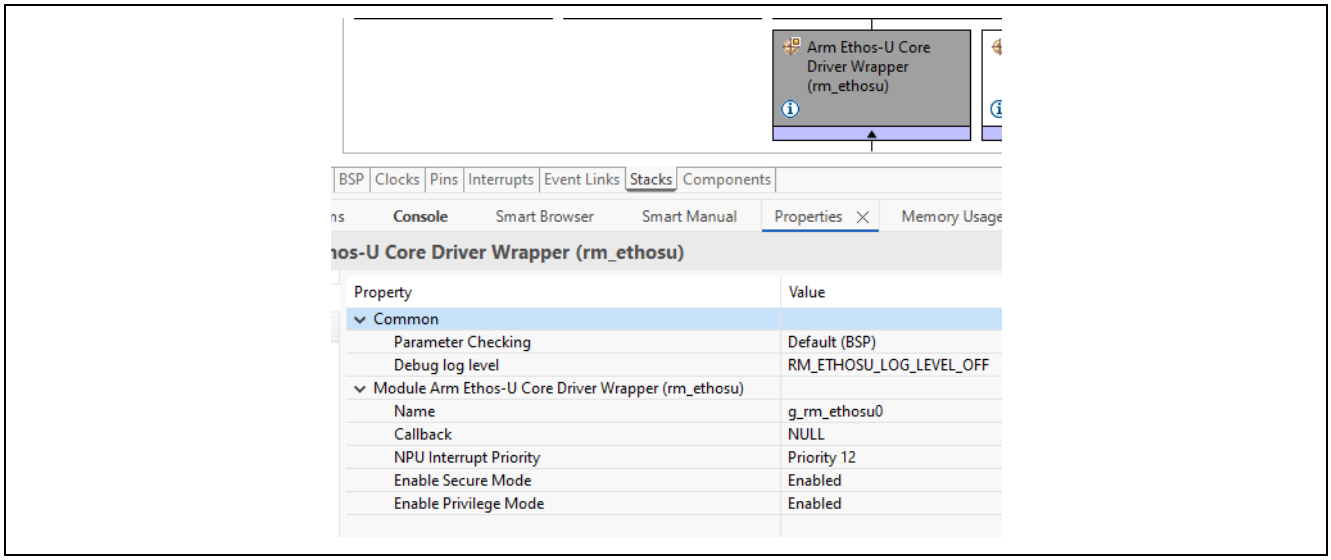


Figure 11. FSP Ethos-U Core Driver Middleware

The Ethos-U Middleware was created by Renesas. It provides high-level APIs user can access from the applications. The following are the public APIs that can be called from the application.

RM_ETHOSU_Open: this function starts the NPU, calls the ethosu_init API which is part of the Ethos NPU Core Driver to initialize the NPU, enables the NPU interrupt, configure the callback and then mark the NPU as initialized. The ethosu_init API does a soft reset on the NPU and then reinitializes the AXI for communication with the CPU.

RM_ETHOSU_CallbackSet: this callback is triggered when the inference finishes

RM_ETHOSU_Close: this function disables the NPU interrupt, calls the ethosu_deinit which is part of the Ethos NPU Core Driver to clean up the resources and stops the NPU.

Refer to section 5.5.2 for the explanation of the **Debug log level** property.

4.2.3 Arm Ethos-U Core Driver

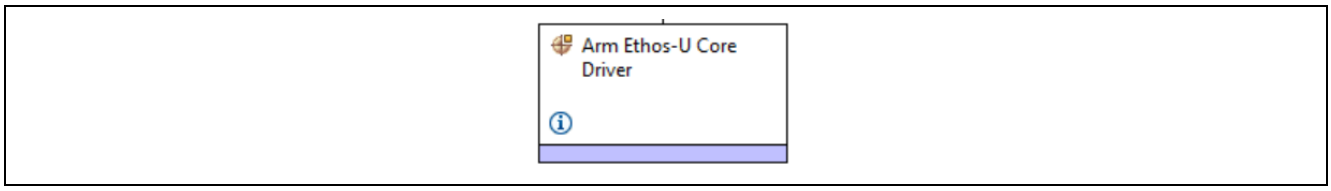


Figure 12. Arm Ethos-U Core Driver

The Ethos-U Core Driver stack is a low-level software layer that provides an interface between the NPU hardware and the TFLM runtime. It integrates the open-source Ethos-U driver from Arm. As a reference, Arm hosts its open-source Ethos-U driver on this GitHub repo: GitHub Arm Ethos-U. There are no user-configurable properties for the Ethos-U Core Driver stack.

The major functionalities of this stack are:

- Submitting workloads to the Ethos-U NPU.
- Managing memory buffers for model execution on resource-constrained devices.
- Handling custom Ethos-U operators that are injected by the AI model compiler.
- Ensuring secure and efficient communication between the MCU (Cortex-M) and the Ethos-U NPU.
- Providing a core software driver for the PMU to perform cycle counting, event monitoring, performance profiling and resource utilization analysis.

4.2.4 Google TFLM CMSIS-NN Kernel

If the model is allocated to run partially on the Ethos-U NPU and partially on the CPU, or entirely on the CPU, the CMSIS-NN library is utilized to optimize performance. This library is specifically designed to leverage the **Arm Helium** technology available in the Cortex-M MCUs, enabling efficient execution of neural network layers on the CPU. By using CMSIS-NN, the application can achieve significantly improved inference speed and reduced CPU load when executing supported operations on the Cortex-M85.

5. Develop AI Application for Real-World Use Cases

5.1 Integrating an AI Inference Overview

With the above introduction as background, the AI model inference operational steps are summarized as the following:

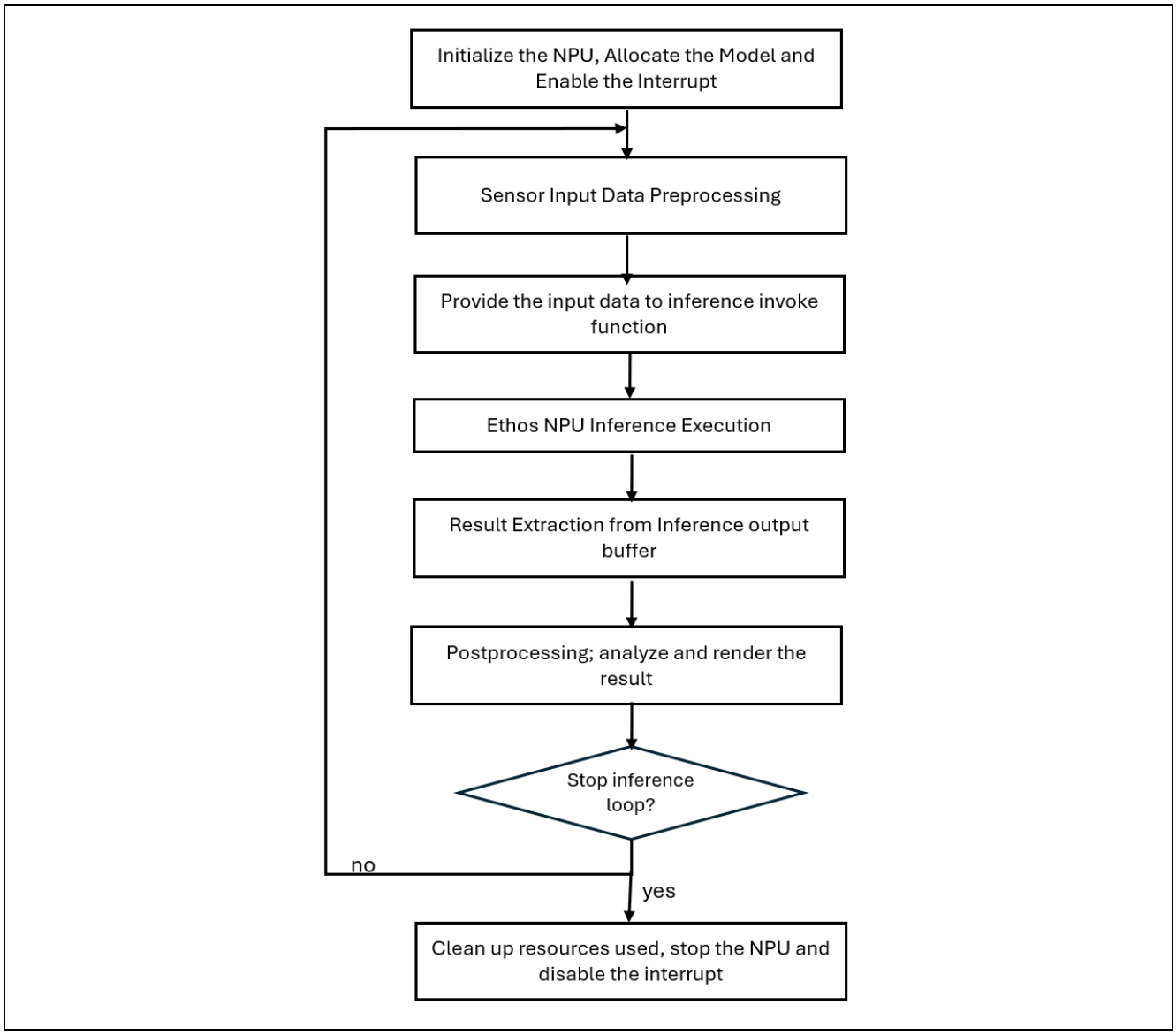


Figure 13. Typical AI Inference Flow

5.1.1 Achieving Good Accuracy

The key in achieving high inference accuracy for a given model in an embedded application is closely related to being able to provide accurate and stable input.

- The Ethos-U NPU requires dedicated memory regions for input tensors, output tensors, and intermediate activations (the Tensor Arena area). They should be separated from CPU stack/heap to avoid contention.
- When performing the Ethos-U Inference, the input tensor data should be stable and not changed during inference. It is recommended to copy the runtime sensor data to a dedicated buffer that is not updated during the inference.
- Align buffers to 32-byte boundaries to optimize NPU memory access.
- Use TCM (Tightly Coupled Memory) if available for frequently accessed small data and interrupts to minimize latency.

5.1.2 Achieving Fast Inference

The key factor that influences the inference speed for a given model is the speed of the memory where the input tensor, the Tensor Arena and the output Tensor. The AI post-processing can be an asynchronized event, so the location of the output Tensor is typically not a factor which influences the inference speed.

5.2 Memory Architecture Considerations

The RA8 MCUs, which has NPU support, integrates MRAM for code storage, which is faster and more power efficient than the flash version for the same process. Additionally, the evaluation kits are supplemented with large SRAM memory, external fast SDRAM and OSPI devices for large AI models, runtime large sensor data and static data storage support.

The following guidelines on the usage of various memory regions are based on the EK-RA8P1.

Type of Memories			Size on EK-RA8P1	Potential Usage in an AI application
MRAM			1MB	model storage, application code
SRAM	User SRAM		1664 KB	model storage, sensor data storage, tensor Arena , inference result, application code Stack and Heap
	Cortex-M85 TCM	ITCM	128 KB	interrupt routine, other time critical routines, sensor data
		DTCM	128 KB	sensor data storage, frequently accessed data which needs time deterministic
	Cortex-M33 TCM	ITCM	64 KB	interrupt routine, other time critical routines, sensor data
		DTCM	64 KB	sensor data storage, frequently accessed data which needs time deterministic
Data Cache			16 KB with ECC	should be enabled for faster Tensor computation, reduced power consumption and accelerated ML processing
Instruction Cache			16 KB with ECC	should be enabled for faster instruction fetching, reduced latency and efficient branch prediction.
Off chip SDRAM			64 MB organized as 16M x 32 bits	model storage, run time sensor data storage, graphics framebuffer storage, Tensor Arena, application code Stack and Heap
SIP Flash Memory			8MB	model storage, Graphics data storage, code storage for those that are not executed frequently.
Off chip OSPI			64 MB	model storage, static image storage, encrypted sensitive information, application code for non-time-sensitive usage
NPU Internal Memory Area			48 KB	DMA buffers: Command stream buffers, Tensor data (input, intermediate and output), neural network weight, scratch buffer. This memory is not available to application code.

5.2.1 Memory Partitioning & Region Assignment

The memory allocation of AI application using the Ethos-U NPU should take the following into consideration.

- System performance, faster memories allow faster inference given the same model
- Usage of the system memory for other applications that also reside in the same application as the AI application, for example, a graphic application.
- The specific AI application targeted, the variation on the size of the model and sensor data can be an important factor in deciding how to manage the system memory.

By default, the IDE provides a default memory configuration of the application based on the application code provided. The memory configuration for the project is defined by \Debug\memory_regions.lld and \Debug\fsp_gen.lld. These two linker files are used in the generic linker script file fsp.lld.

For AI applications, it might be desirable to customize the memory configuration, for example creating a shared memory region between the two cores. Another example might be creating a specific memory section for one AI inference instance and another memory region for another AI inference instance.

The Smart Configurator can be used to customize the memory region assignment for dual core and single core RA8 MCUs.

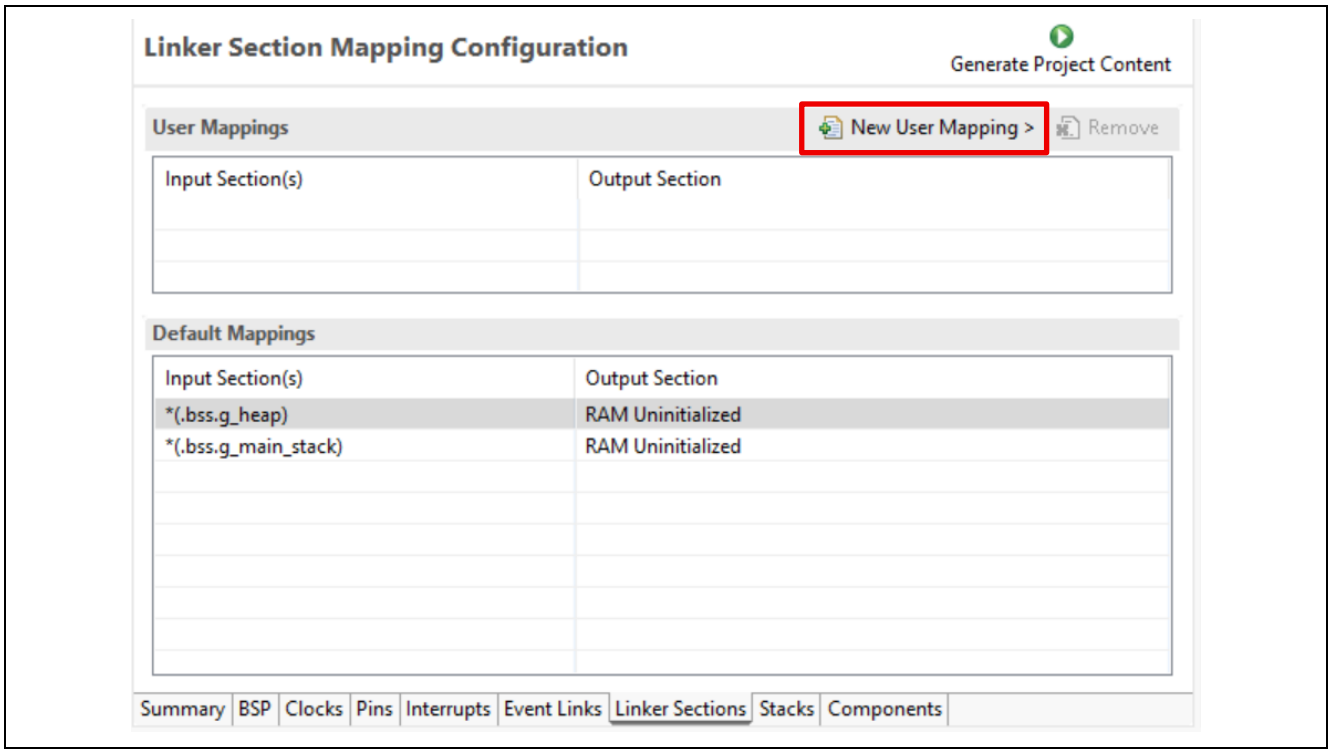


Figure 14. Customize the Memory Region Assignment using the RA Smart Configurator

5.2.2 Code Flash, SRAM vs. External SDRAM

The following are some general considerations on allocating contents to Code Flash (MRAM), SRAM and External SDRAM:

- SRAM is faster than SDRAM. If possible, always allocate the model input sensor data and Tensor Arena to SRAM. If the Tensor Arena area does not fit the SRAM area, the SDRAM area can be used as the Tensor Arena area. This can be achieved by allocating the tensor Arena to the .sdram_noinit region.
- By default, the model is stored to Non-volatile memory region. If the model or activations exceed the internal code flash region, external DRAM (e.g., SDRAM) may be required but will introduce higher access latency. In this case, the model can be stored in external flash area (for example, the OSPI) and copied to the SDRAM area at MCU reset. This can be achieved by allocating the model to the ".sdram_from_ospi0_cs1" region.
- If there is a desire to load the model to SRAM, the model can be stored in the external flash area, like the OSPI and copied to SRAM at the MCU reset. This can be achieved by allocating the model to the ".ram_from_ospi0_cs1" region.
- Use a memory hierarchy where:
    - Weights & bias parameters are stored in Flash, or MRAM, or OSPI.
    - Input/Output/Tensor Arena are stored in SRAM for faster access.

5.2.3 Tensor Arena Allocation

The tensor arena is typically statically allocated to SRAM or SDRAM buffer for inference. If using RUHMI or MERA for compilation, the size needed is defined in the output file during the compilation process.

The following is an example of the Tensor Arena memory definition in the compilation result, which is taken from the compilation result for the MobileNet V1 0.25 integer model. In this case, the Tensor Arena is allocated to SRAM. Similar definition like Figure 15 can be found in the sub_0000_invoke.c file from the RUHMI or MERA compilation output.

```
// Define arenas with allocation and 16-byte alignment
__attribute__((aligned(16))) uint8_t sub_0000_arena[401408];
```

Figure 15. Tensor Arena Area needed by an AI Model for Ethos-U Inference

The following is an example of the Tensor Arena area size definition when inferencing using CPU. Application code is responsible for allocating this area in SRAM or SDRAM and provide the buffer to the compute_sub API which invoke the inference process. Similar definition like Figure 16 can be found in the compute_sub_0000.h file from the RUHMI or MERA compilation result.

```
/*
  kBufferSize_sub_0000 is a compile-time constant to be used by the user of compute_sub_0000 function
  to allocate a buffer with at least the specified size.

  Example of how to call the compute function:

    // it is possible to use either heap, stack or a custom data section to allocate this buffer
    uint8_t my_buffer[kBufferSize_sub_0000];

    int main() {
      ...
      compute_sub_0000(my_buffer, input, output);
    }
*/
enum BufferSize_sub_0000 {
  kBufferSize_sub_0000 = 605712
};
```

Figure 16. Tensor Arena Area needed by an AI Model for CPU Inference

5.2.4 DMA and Cache Management

The following are some key considerations on the DMA and Cache Management

- Ethos-U uses DMA for memory transfers; ensure buffers are in cache-coherent memory regions.
- If using non-cacheable memory, align memory buffers to cache line boundaries.
- Flush and invalidate caches before and after tensor processing to avoid stale data.

Reference Application Project R11AN0995 for how to Data Cache is handled before and after AI inference.

5.3 Power and Performance Trade-offs

Although power consumption and performance management can be two conflicting goals in many applications, the application design should take the following two considerations to maximize the overall performance and power usage of the system:

- Optimize memory access patterns to reduce power consumption.
- Minimize the use of off-chip memory to reduce energy overhead.

5.4 Performance Analysis

The following software components provided by the Renesas FSP package can be used for the performance analysis of the AI application.

5.4.1 Using the CMSIS-NN Event Recorder

The CMSIS-NN Event Recorder is a feature within Arm's Cortex Microcontroller Software Interface Standard (CMSIS) that enables developers to monitor and analyze the performance of neural network (NN) applications on Cortex-M processors. By integrating the Event Recorder into your CMSIS-NN-based projects, you can gain insights into the execution flow, timing, and potential bottlenecks, facilitating effective debugging and optimization.

Key Features:

- Event Annotation: Allows insertion of event markers in the code to monitor specific functions or operations.
- Timing Analysis: Provides timestamps for events to measure execution durations. Additional information can be acquired through Arm Developer

- Real-Time Monitoring: Enables observation of events as they occur during program execution.
- Minimal Intrusion: Designed to have a low impact on system performance, ensuring accurate profiling

Renesas has integrated the Event Recorder as part of the Arm CMSIS-View library in the FSP package when the NPU stack is added to the project.

5.4.2 Using the Ethos-U Performance Counter

The Arm Ethos-U Performance Monitoring Unit (PMU) is an integral feature of Arm's Ethos-U series Neural Processing Units (NPUs), such as the Ethos-U55 and Ethos-U65. This unit provides developers with tools to monitor and analyze the performance of neural network workloads on these NPUs, facilitating optimization and efficient resource utilization.

Key Features of the Ethos-U PMU:

- Cycle Counter: A 48-bit counter that tracks the number of cycles taken by the NPU to execute tasks, offering insights into processing time and efficiency.Arm Developer
- Event Counters: Four 32-bit counters that can be programmed to monitor specific events within the NPU, such as memory accesses or instruction executions, enabling detailed performance analysis.

By leveraging the PMU, developers can gain a comprehensive understanding of how their neural network models perform on Ethos-U NPUs, identify potential bottlenecks, and make informed decisions to enhance application performance. User applications can use the Ethos-U PMU to check the time used for the inference operation.

The APIs that can be used by the user application is in the NPU Core Driver (\ra\npu\ethos_u_core_driver\inc\ethos_drvier.h). Reference the FSP UM section on “Getting Started: Defining functions and PMU events for MCU performance profiling” to understand the preparations needed to use this software tool.

5.4.3 Using a GPT Timer

A dedicated timer can also be used in an application project to measure the system performance. The RA8P1 MCU group offers 32 bit General Purpose PWM timers, which can be used to measure the system performance on the application layer. Refer to R11AN0995 for an example of this implementation.

5.5 Debugging Support

The TFLM library as well as the NPU core software stack provides the following two debug logging schemes which can be routed to an interface of the application’s choice.

5.5.1 Using the TFLM Debug Log Callback

The TFLM library provides a debug function hook RegisterDebugLogCallback. This function is defined in \ra\npu\tflite-micro\tensorflow\lite\micro\cortex_m_generic_debug_log.cc.

Users can create a callback function to print or log the debug information and provide the function pointer to this function. For example, users can implement a print function to channel the debug log output to the SEGGER RTT output or the JLink Console output. Refer to R11AN0995 for an example implementation.

5.5.2 Using the Ethos-U NPU Debug Log

This is configurable from the FSP stack. There are several debug log levels. This debug level will be used in the macro functions defined in \ra\fsp\src\rm_ethosu\ethos_log.h. The ethos_log.h defined printf function which can be routed to the SEGGER RTT output or the JLink Console output in the future.

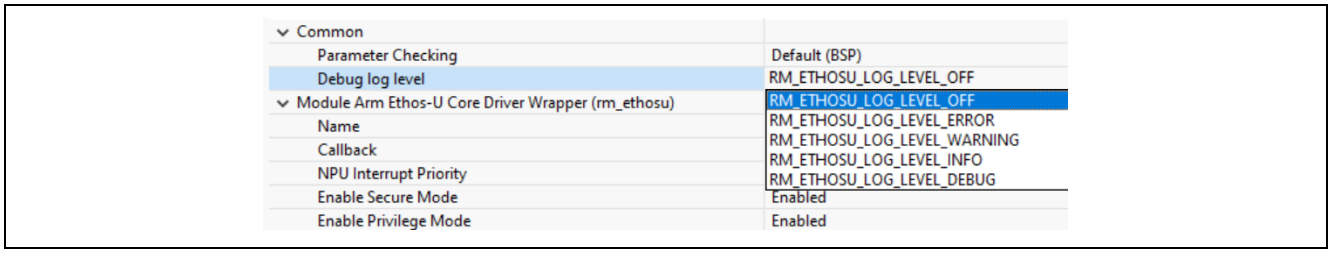


Figure 17. Debug Utility Offered in the Ethos-U Stack

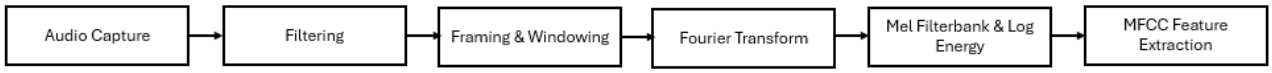
5.6 Use Case Analysis

Although the model compilation and deployment to the embedded system can be evaluated relatively easily using static input and output features, the end-to-end development typically includes major development effort in the sensor data collection, pro-processing and post-processing steps.

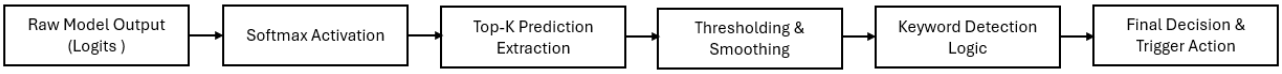
5.6.1 Key Word Spotting

Typical pre-processing and post-processing for key word spotting AI applications is provided here for your reference. The exact operations depend on the model used and the sensors used.

Pre-processing for Key Word Spotting



Post-processing for Key Word Spotting

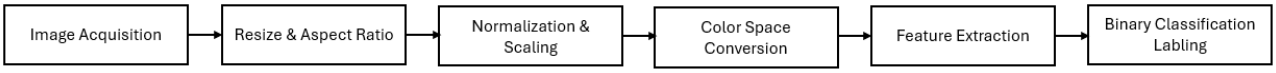


Key Word Spotting model and the Tensor Arena area are typically well below 1MB. Users can optimize the system performance by utilizing the large SRAM and TCM on the RA8P1 for improved performance.

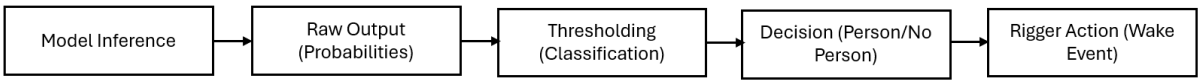
5.6.2 Visual Wake Word

The typical pre-processing and post-processing for Visual Wake Word AI applications is provided here for your reference. The exact operations depend on the model used and the sensors used.

Pre-processing for Visual Wake Word



Post-processing for Visual Wake Word

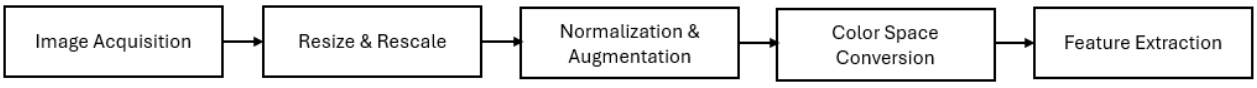


The typical Visual Wake Word model and the Tensor Arena area each are well below 1MB. Users can optimize the system performance by utilizing the large SRAM and TCM on the RA8P1 for improved performance.

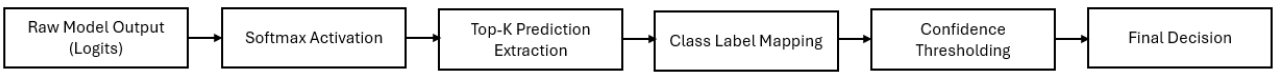
5.6.3 Image Classification

Image Classification detects what objects exist in a view without pointing out the location of the object. The typical pre-processing and post-processing for Image Classification AI applications is provided here for your reference. The exact operations depend on the model used and the sensors used.

Pre-processing for Image Classification



Post-processing for Image Classification



The size of the image classification model size and Tensor Arean area size varies from several hundred KB to several MB. For large models, the model can be stored in an external flash area and copied to the SDRAM for runtime usage for improved inference performance. Please refer to R11AN0995 for an example application project on image classification.

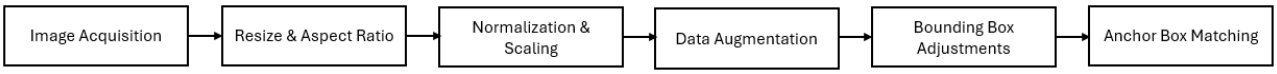
5.6.4 Object Detection

Object Detection answers the question of what objects are present and where are they located. The typical pre-processing and post-processing for Object Detection AI applications is provided here for your reference. The exact operations depend on the model used and the sensors used.

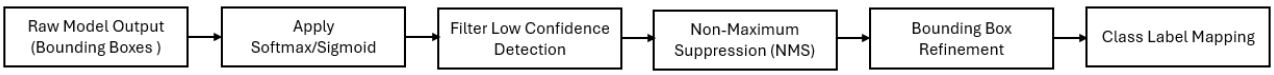
The model size for object detection is closely related to the number of object categories it can detect, but it also depends on the model architecture.

A special type of object detection is face detection. Refer to the APN ‘Referential AI Vision Application with Face Detection on EK-RA8D1” for an example of using the RA8D1 for an example of Object Detection. The face detection model used in this application project, although relatively smaller than the object detection model which detects many object categories, does demonstrate the following pre-processing and post processing steps.

Pre-processing for Object Detection



Post-processing for Object Detection:



The size of the object detection model size and Tensor Arean area size varies from several hundred KB to several MB. For large models, the model can be stored in an external flash area and copied to the SDRAM for runtime usage for improved inference performance.

6. Appendix

6.1 Model Quantization using open-source solution:

Model quantization is supported by the open-source solution as well as Renesas RUHMI tool. Please refer to the model optimization page of the TensorFlow Lite website to understand the model quantization operations for open-source solution support. Please refer to the RUHMI Platform Quick Start Guide for using RUHMI for model quantization.

6.2 Compile a Model using Arm Vela

A quantized neural network can be compiled offline using the Ethos-U NPU open-source compiler to produce a command stream. Arm provides the open-source compiler Vela to perform this compilation. For more information on the usage of Vela compiler, refer to this Arm website: Ethos-U Vela compiler.

6.2.1 Using the Open-Source Tool

Vela is an open-source tool developed by Arm to optimize neural network models for execution on Arm Ethos-U NPUs. It takes a pre-trained and TFL quantized model (.rflite), optimizes the orders of operations and compresses weights to improve efficiency on Ethos-U NPUs. The compilation result is an optimized .tflite model ready for deployment to the embedded systems. Here is the Arm's introduction for the Vela compiler.

The following is the overview of using this tool to convert an integer quantized TFLite model to c array to be used by the Ethos-U NPU for AI inference.

The .ini file is used to configure the MCU and NPU hardware. The .ini file defines the NPU core clock and the ratio of the memory bandwidth of the memory while the Tensor Arena is location to the NPU internal bus bandwidth.

The generated .h file contains the model and some metadata, like the size of the model. Note that although the model file format before and after Vela compilation is the same, the layers that are converted to use the custom Ethos-U layer are changed.

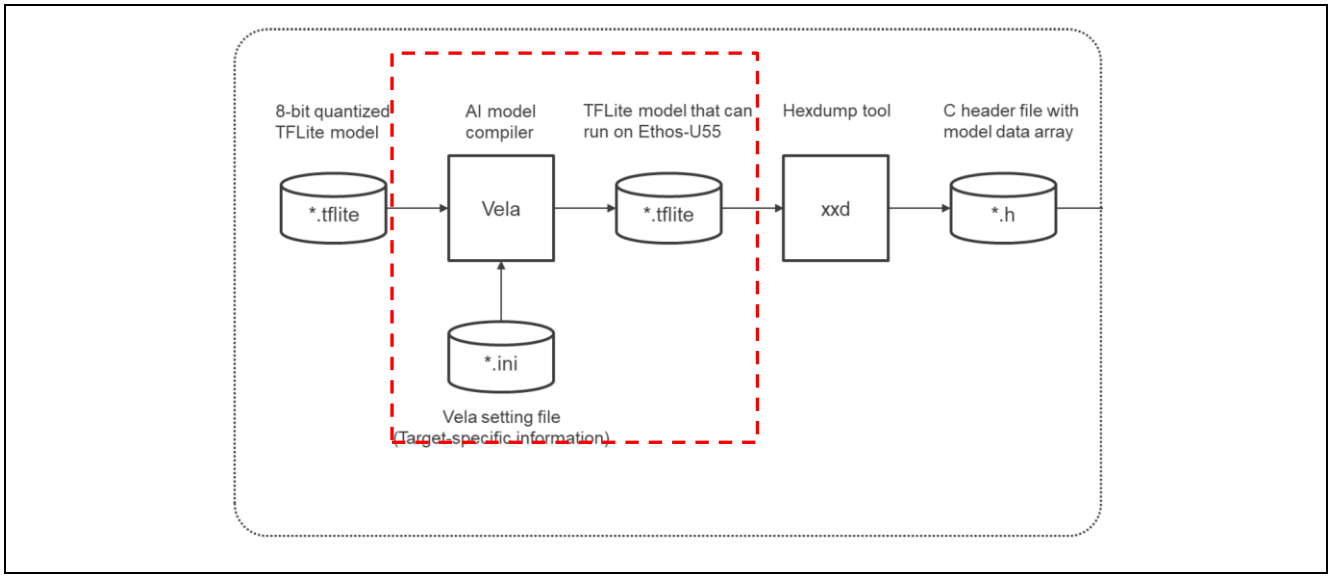


Figure 18. Compile the Model for Ethos-U NPU Usage using Open-Source Solution

To evaluate the model inference using the CPU only, there is no need to install the Software tool and the Vela Compiler. The steps included in the red dotted box should be skipped.

7. References

1. Flexible Software Package (FSP) User's Manual
2. Renesas RA8P1 Group User's Manual: Hardware
3. Renesas RA Family Referential Vision AI for Image Classification using Ethos-U NPU (R11AN0995)
4. Renesas RUHMI Framework Quick Start Guide (R11QS0065)
5. Renesas RUHMI Framework GitHub
6. Arm Ethos-U55 Technical Reference Manual

8. Website and Support

Visit the following URLs to learn about the RA family of microcontrollers, download tools and documentation, and get support.

EK-RA8M1 Resources	renesas.com/ra/ek-ra8p1
RA Product Information	renesas.com/ra
Flexible Software Package (FSP)	renesas.com/ra/fsp
RA Product Support Forum	renesas.com/ra/forum
Renesas Support	renesas.com/support

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	July 18.25	—	Initial release

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins

   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan

www.renesas.com

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.