Renesas RA Family

Audio AI Workflow on RA8

Introduction

Audio AI Workflow on RA8 provides an end-to-end process for optimizing, converting, and deploying AI models developed with widely used frameworks such as TensorFlow as efficient executable programs for Edge AI on the RA8 MCU. It demonstrates how RA8 microcontrollers can be used effectively for AI-based audio processing applications.

The accompanying application project presents a frame-based audio noise suppression solution implemented on the Renesas RA8 MCU. The system is built on the Arm® Cortex®-M85 core, which features Digital Signal Processing (DSP) extensions and AI acceleration capabilities through the integrated Neural Processing Unit. It demonstrates how digital signal processing and neural network inference can be seamlessly combined to process continuous audio streams with low latency and high efficiency.

This application note provides guidance for the complete implementation of an AI based audio application on RA8P1, including:

- System architecture overview.
- Application demonstration.
- Software and hardware configuration.
- AI model deployment using the RUHMI (Robust Unified Heterogeneous Model Integration) workflow.

Required Resources

The following resources are referenced throughout this application note.

Development Tools and Software

- e² studio version: 2026-04.2 (26.4.2)
  Please download the package from "RA Flexible Software Package (FSP)" or search online for the keyword "RA FSP Download".
- Renesas Flexible Software Package (FSP) v6.5.0
- Renesas RUHMI (Robust Unified Heterogeneous Model Integration) Framework (installed as an add-on for e2 studio).
- LLVM Embedded Toolchain for ARM v21.1.1
- SEGGER J-Link RTT-Viewer version 9.42 and System View version 4.10a.

Hardware

- Renesas RA8P1 Evaluation Kit (RA8P1 MCU Group) (RA8P1 Group User's Manual: Hardware)
- PMOD AMP2 (Pmod AMP2 - Reference Link).
- 1 x Speaker with 3.5mm audio jack.
- Host PC for programming, debugging, terminal control and system observation.

Prerequisites and Intended Audience

This application project assumes that the audience understands AI/ML design concepts and has good experience using the Renesas e2 studio IDE and the Renesas RA Smart Configurator. Basic instructions on how to import and compile an application are not provided. Refer to the "Importing an Existing Project into e² studio" section in the Flexible Software Package (FSP) User's Manual (UM) for guidance.

The intended audience is all users who are or will be developing AI applications using Renesas RA MCUs, specifically those with Ethos-U NPU support and pre-quantized int8 models.

Contents

1. Application Overview

This section provides an overview of the application.

The application runs on the RA8P1 evaluation board under FreeRTOS and demonstrates a real-time audio processing pipeline. Incoming audio data is captured from the on-board microphones, processed using signal processing and AI-based noise suppression, and then output through a connected speaker. The following sections describe the system architecture, audio data flow, and playback mechanism in more detail, providing a complete view of how each functional stage operates and interacts within the application.

The application uses a pre-trained RNNoise model from the ML-Zoo repository (Arm-Examples/ML-zoo). This model is converted into an embedded-friendly format and integrated into the project as C source files for execution on the target device using RUHMI workflow. In addition to describing the deployed model, this document also outlines the training workflow, enabling users to understand the model generation process, reproduce the steps, and adapt the solution to their own application requirements if needed.

To optimize performance, the application adopts the RUHMI workflow to efficiently distribute processing across the CPU and NPU.

The RA8P1 integrates the Arm® Ethos™-U55 NPU to offload AI processing from the CPU. Machine learning applications typically rely on neural network inference, which can run in software on a general-purpose processor but achieve higher performance when accelerated in hardware. Ethos-U NPUs are compact, power-efficient processors that reduce inference time and memory usage for neural network workloads. Arm provides the Ethos-U NPU hardware Register Transfer Level (RTL) design, enabling MCU vendors to integrate NPU acceleration into their devices. RA8P1 further optimizes system performance through tightly coupled memory (TCM), cache utilization, efficient data paths, and AI software libraries to maximize Ethos-U efficiency while maintaining low power consumption. In addition, neural network and signal processing operations are accelerated using Arm Helium instructions.

Enabling the Floating Point Unit (FPU) can improve AI application performance when using floating-point models. However, floating-point models require more storage space and typically run slower than quantized models.

2. Run Audio AI Workflow on RA8 application

This section describes how to run and evaluate the Audio AI workflow on RA8 application.

2.1 Project import and compilation

After importing the application project, build it, download it to the target board, and run it.

Step 1: Navigate to the top-left corner of e² studio.

Select **"File"** → **"Import"** → **"General"** → then double click **"Existing Projects into Workspace"**.

Then select the root directory and browse to your workspace, application directory, or directly to your repository.

Select the **"Search for nested projects"** option to ensure the entire application, including nested files, is imported. Then click **"Finish"** and wait until the import completes.

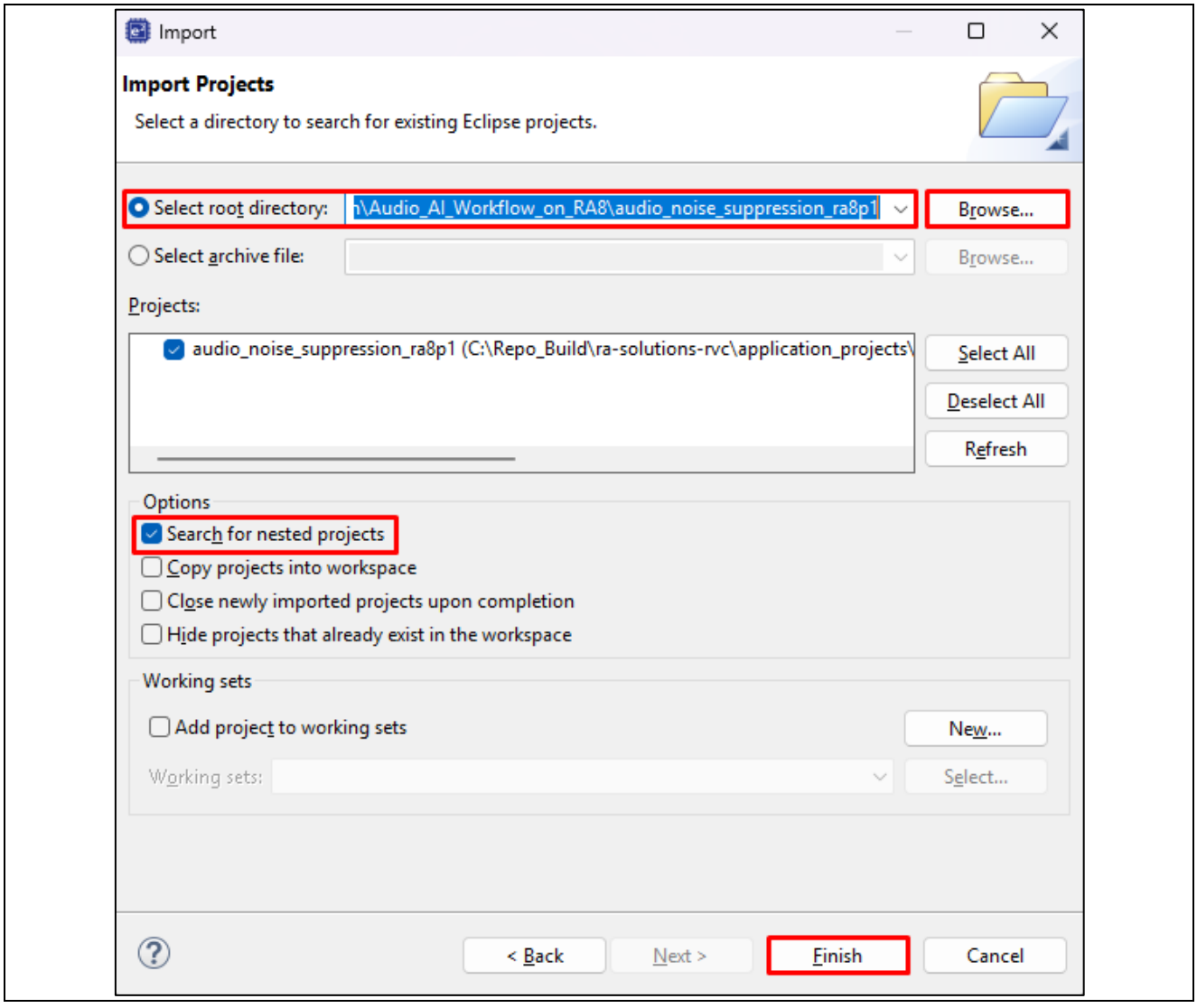


Figure 1. Import project to e² studio

In the **Project Explorer** window, expand project tree and double-click the "`configuration.xml`" file to open the FSP Configuration perspective. Application information can be viewed in the **Summary** tab.

Step 2: To build the application project, select the application folder name, then at the top-left corner of the e² studio, click the **"Build"** icon "🔨" to start building the project. Wait until the build process completes without errors.

Note: Place the project folder near the root of the drive (e.g., `C:\Workspace`) to avoid exceeding the maximum path length limit on Windows.

2.2 Setting Up the Hardware

In addition to the Hardware listed earlier, the following items are required for the hardware connection:

- 1 x USB debug cable for programming and debugging.
- 1 x 20k Ω resistor to limit current and stabilize the signal to protect the circuit.
- 5 x jumper wires for connecting signals between the board and the module.

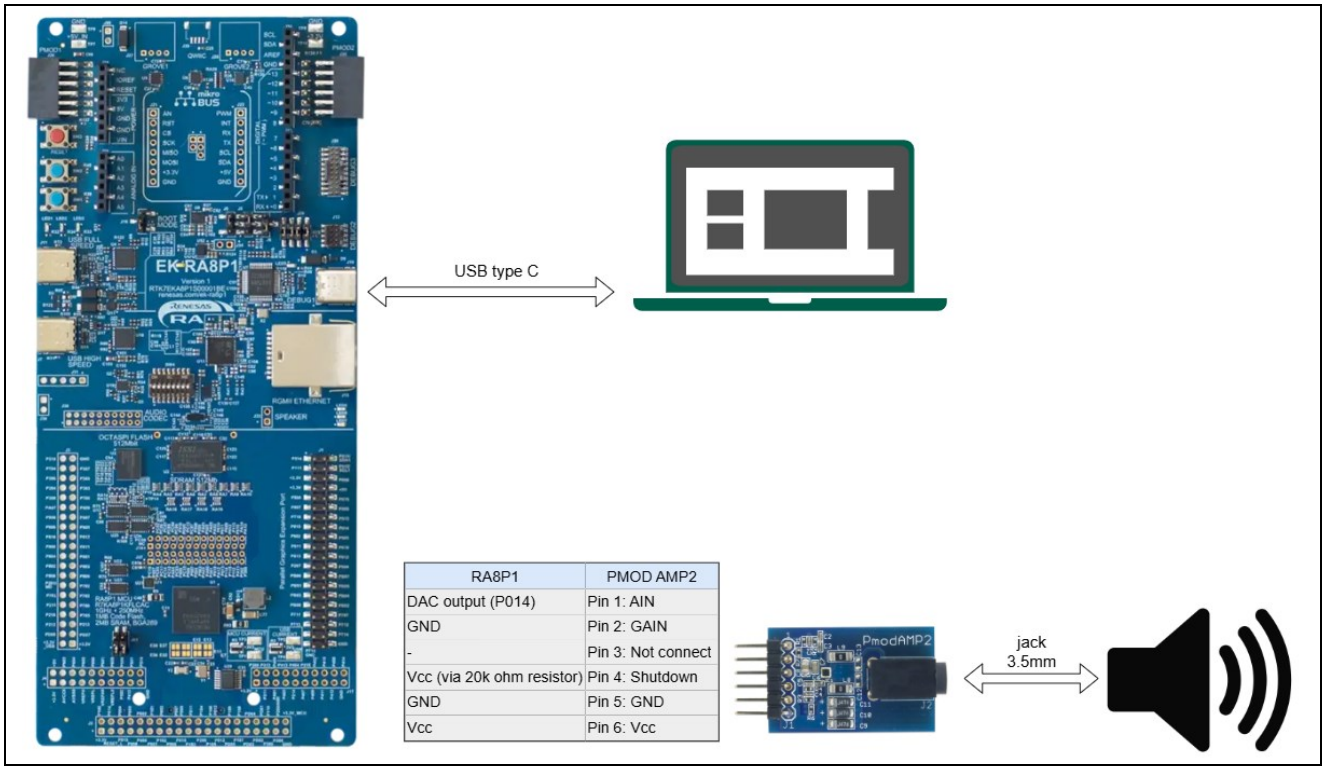
Follow the steps below to set up the hardware for the application.

Step 1: Prepare the EK-RA8P1 Board

Place the RA8P1 evaluation board on a stable, non-conductive surface. Ensure that all required peripherals are accessible and that the board is not powered during initial connections.

Step 2: Connect the PMOD AMP2 Module

Identify the DAC output pin (P014) on the EK-RA8P1. This pin outputs the analog audio signal generated by the application. Interface the PMOD AMP2 audio amplifier module with the evaluation board using the connection shown in Figure 2. Ensure that all connections are secure and correctly aligned with the PMOD header.



RA8P1	PMOD AMP2
DAC output (P014)	Pin 1: AIN
GND	Pin 2: GAIN
-	Pin 3: Not connect
Vcc (via 20k ohm resistor)	Pin 4: Shutdown
GND	Pin 5: GND
Vcc	Pin 6: Vcc

Figure 2. Hardware Connection

Step 3: Connect the Audio Output Device

Attach an external speaker or headphones to the PMOD AMP2 module using a 3.5 mm audio jack. This output device is used to evaluate the playback audio after signal processing.

Step 4: Connect the Host PC and power ON the system

Use a USB Type-C cable to connect the EK-RA8P1 board to a host PC. This is used to connect with the Debug USB port on the EK-RA8P1 and the PC's USB port. We recommend using a USB Type-C to Type-C cable for a reliable power supply to the system.

After verifying all connections, confirm that the board is recognized by the host PC.

Based on the hardware connections described above, the overall application workflow can be summarized as follows.

- The EK-RA8P1 board acts as the central controller of the system and is operated through a command-line interface from a host PC via the USB Type-C connection.
- Each audio frame is processed by the system and immediately forwarded by the MCU to the speaker, enabling real-time evaluation of the output.
- The PMOD AMP2 amplifies low-power audio signals to drive a mono output. It provides selectable digital gain settings of 6 dB or 12 dB and includes pop-and-click suppression.

Note: The PMOD AMP2 uses a standard 1/8-inch (3.2 mm) mono speaker jack. Although it is designed for a mono connection, 3-pole or 4-pole plugs can still be used. For a 4-pole plug, insert it only partially so that the first three contacts (Left, Right, and Ground) are connected, while the fourth contact remains unconnected. This ensures proper audio output.

2.3 Run the application and system observation

2.3.1 Download and operate application

Step 1: Start the Debug

After completing the build instructions in section 2.1 and configuring all necessary hardware in section 2.2, download and run the project.

Select the project folder name, then click the **"Debug"** icon "  " at the top-left corner of the e² studio. The debugger will start and download the application image to the board.

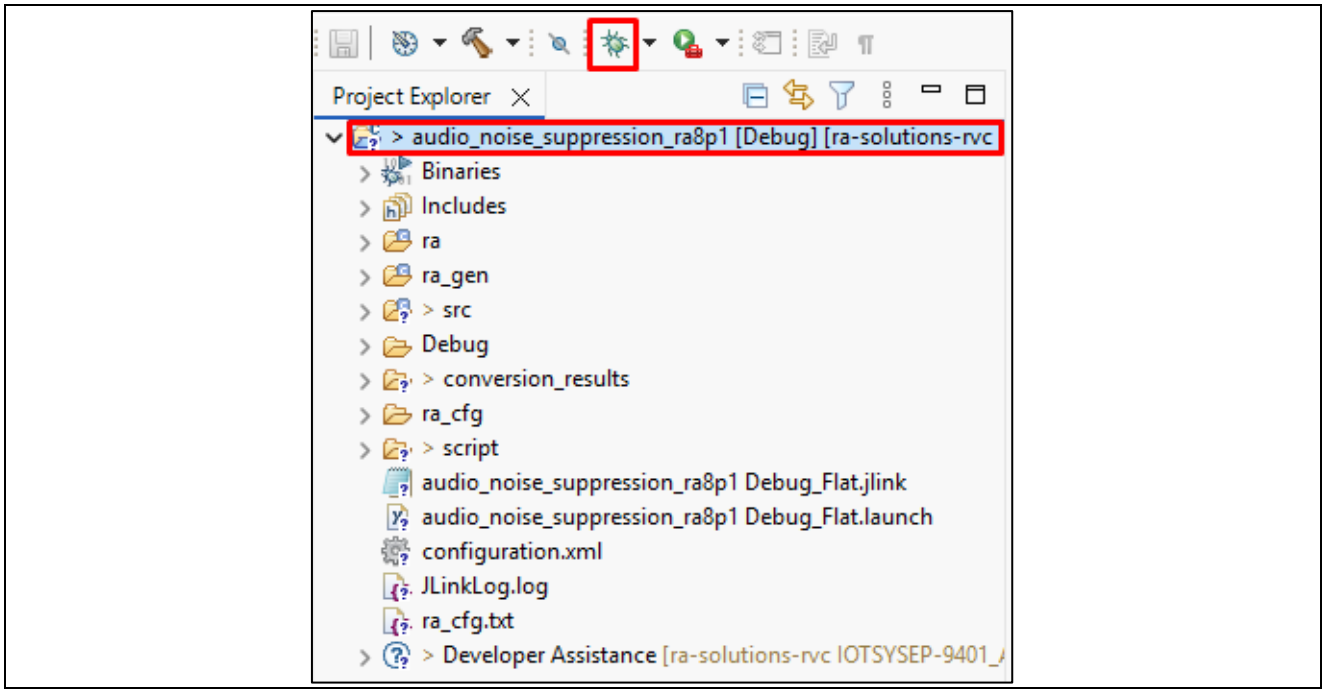


Figure 3. Start debug to run application

Click **"Switch"** if the Confirm Perspective Switch dialog box appears.

Then click **"Resume"** icon "" twice to run the application.

Step 2: Open J-Link RTT-Viewer

To obtain RTT Control block address, open directory `./Debug/audio_noise_suppression_ra8p1.map`

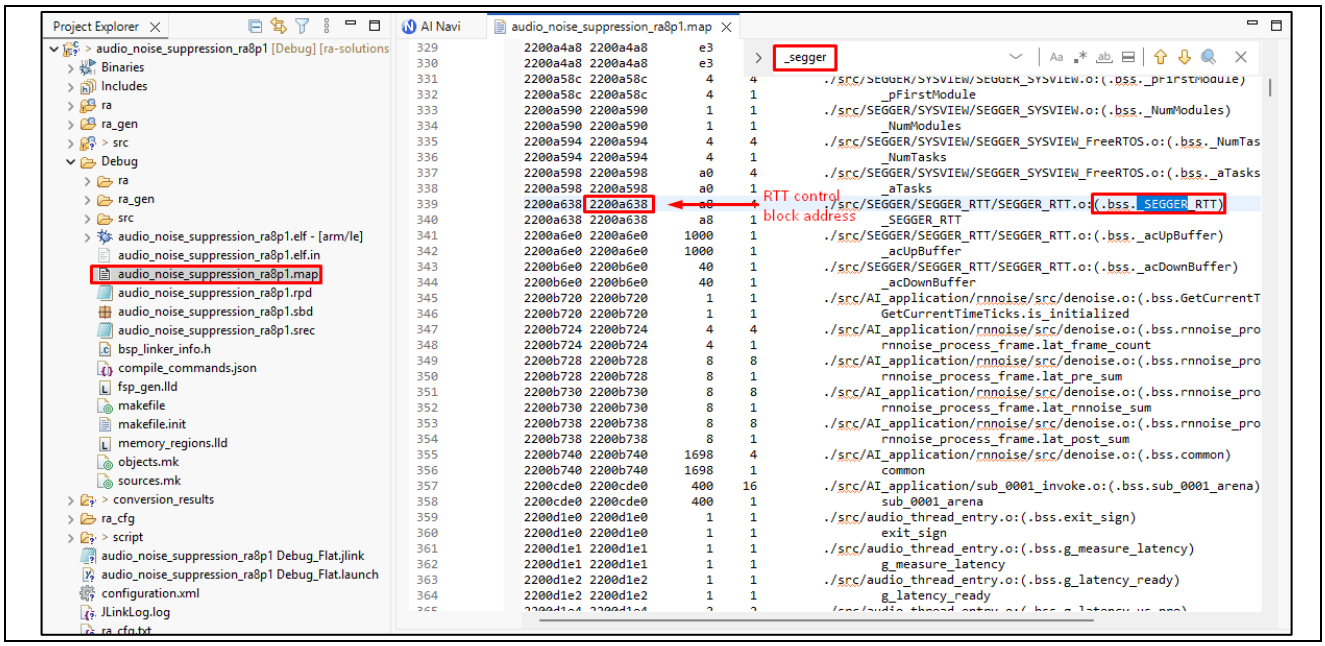


Figure 4. Obtain RTT Control block address

After obtaining RTT Control block address, open J-Link RTT-Viewer.

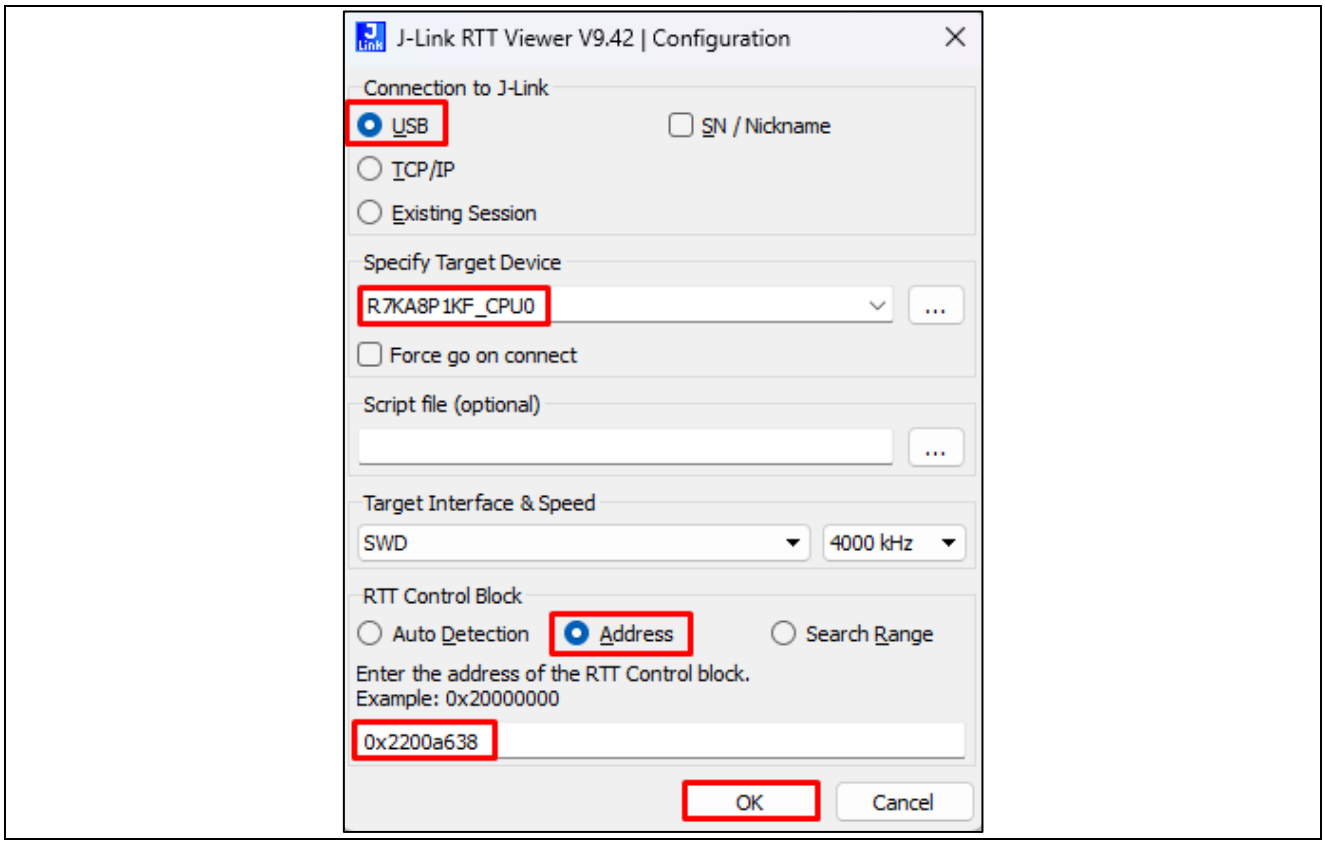


Figure 5. J-Link RTT Viewer configuration

Configure the J-Link RTT Viewer as shown in the Figure 5. After clicking **"OK"**, the application interface will appear.

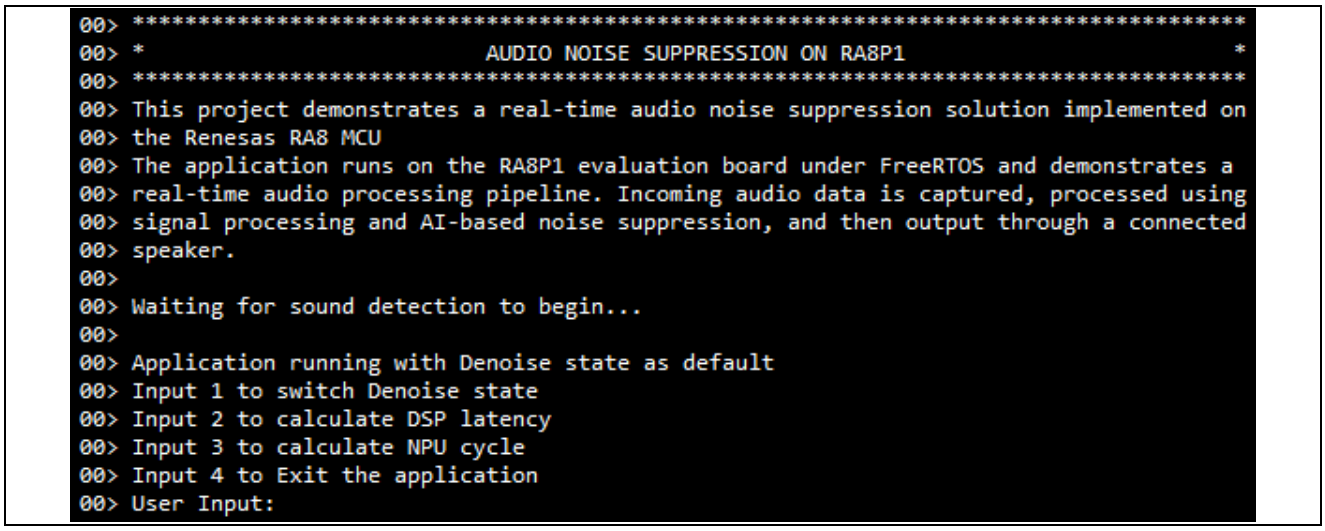


Figure 6. RTT Viewer GUI

Next, enter **"1"** to toggle the denoise function between two states: **DISABLED (raw audio)** and **ENABLED**, then compare the audio quality improvement with and without noise suppression.

Alternatively, the same operation can be performed by pressing the **SW1 button** on the EK-RA8P1 board. When Denoise is **ENABLED**, blue LED (LED 1) will turn ON.

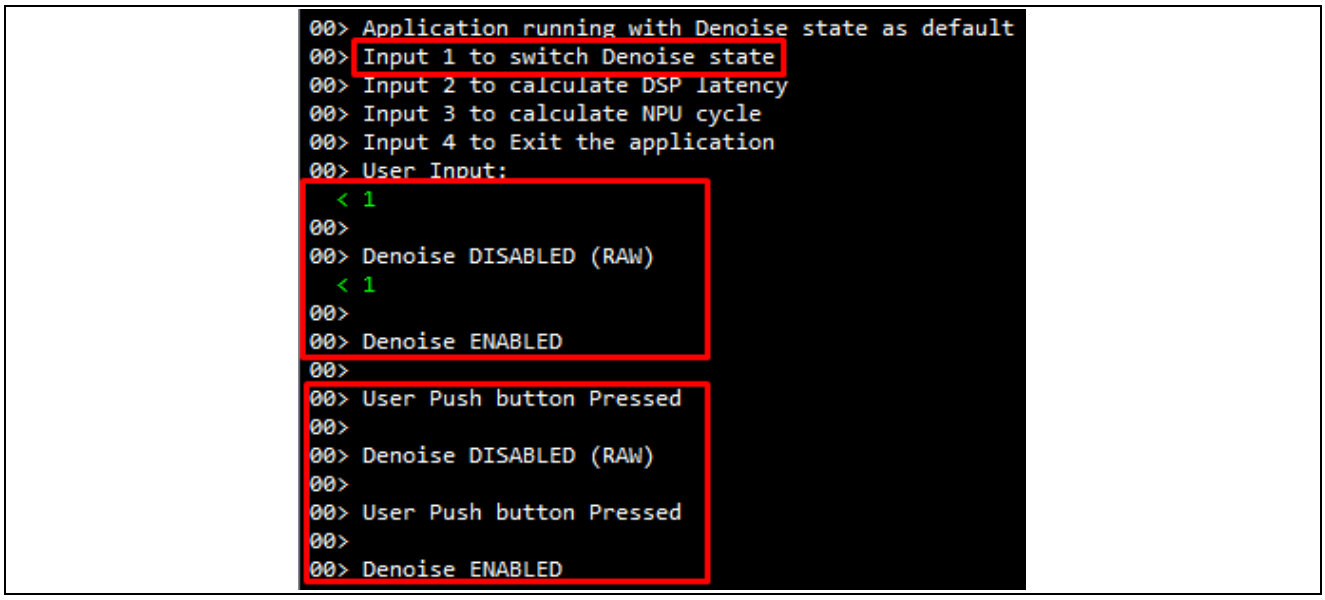


Figure 7. Denoise state switch control

The interface also provides two additional functionalities: Latency Measurement and NPU Timeline Capturing

User can enter:

- **"2"** to start DSP latency measurement
- **"3"** to start NPU execution time (cycle) measurement
- **"4"** to exit the application

```
  < 2
00>
00> [Latency Measure] Measuring over 100 frames...
00>
00> pre-DSP:   4521   us/frame
00> rnnoise:   87     us/frame
00> post-DSP:  1427   us/frame
00> Total DSP latency: 5948 us/frame
  < 3
00>
00> [NPU time line Capture] Capturing for 1 frame...
00>
00> Inference took 70549 cycles
00>
00> NPU: npu_idle: 4992
00> NPU: npu_active: 65690
00> NPU: npu_axi0_en: 35404
00> NPU: npu_axi1_en: 70870
00>
00> CPU part: 4859
00>
00> Completed capture
  < 4
00>
00> Application exited!!!
```

Figure 8. Application operations

2.3.2 Audio data visualization

Click the **"Restart"** icon "  " to restart the application without repeating the full setup procedure, then click **"Resume"** to run the application.

To start with visualization, provide audio input to the application while AI noise suppression is enabled. The observation buffer stores 3 seconds of audio, so the input duration should be longer than 3 seconds.

In the top-left corner of the **Debug** view, click the **"Suspend"** icon "  " to temporarily pause the application.

Open the **Memory** view with applied configuration to observe the differences between raw and denoised audio data. Click the **"Real-time Refresh"** icon "  " if continuous updates are required.

Note: For better observation, users should speak directly into the microphone or place the sound source close to the microphone.

Figure 9 shown the position of the integrated MIC on EK-RA8P1.

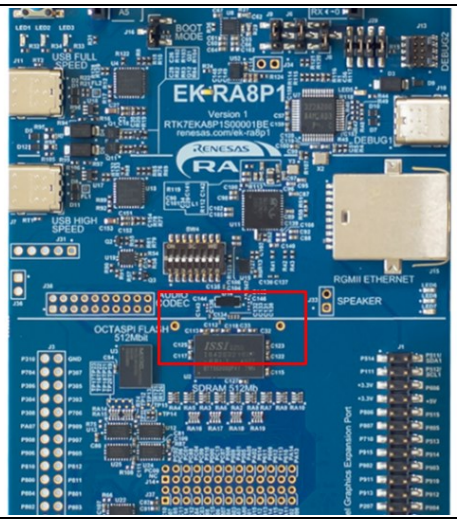


Figure 9. Integrated MIC on EK-RA8P1

Switch between `&g_raw_stream_buffer_memory` and `&g_denoised_stream_buffer_memory` to compare the raw and denoised audio streams.

The raw and denoised audio can also be played back for comparison on e2studio IDE. Right-click on the waveform view, select “Play Sample Rate”, and choose “48000” Hz to match the correct sample rate. Click the start of the waveform, then click the Play button icon “ ” to listen to the raw or denoised audio frame.

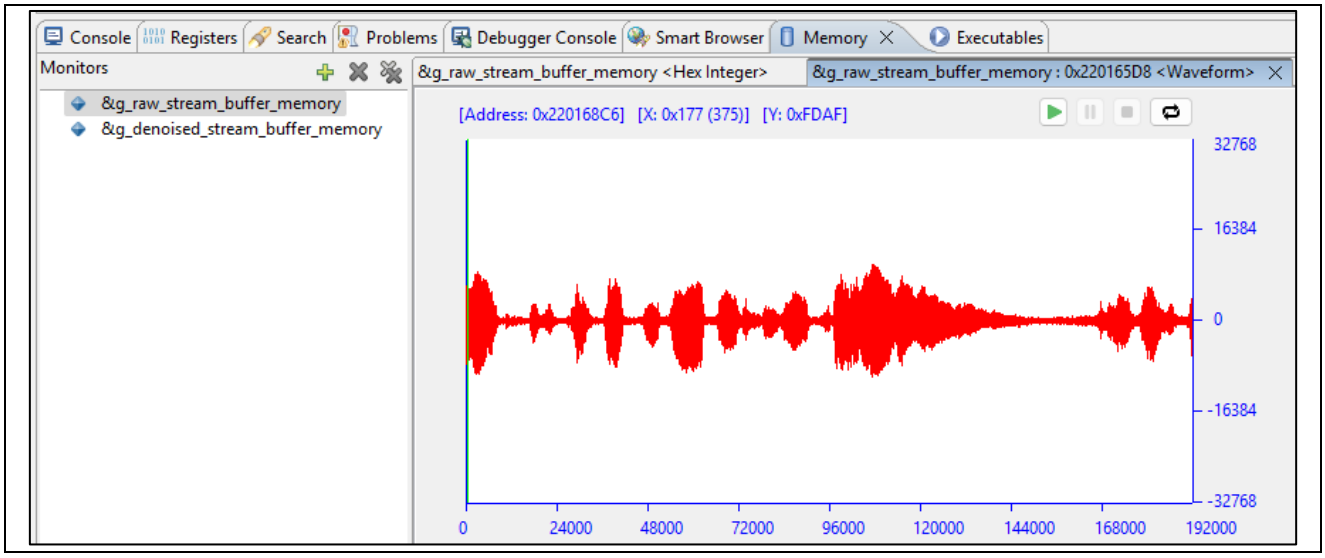


Figure 10. g_raw_stream Stream Buffer memory view

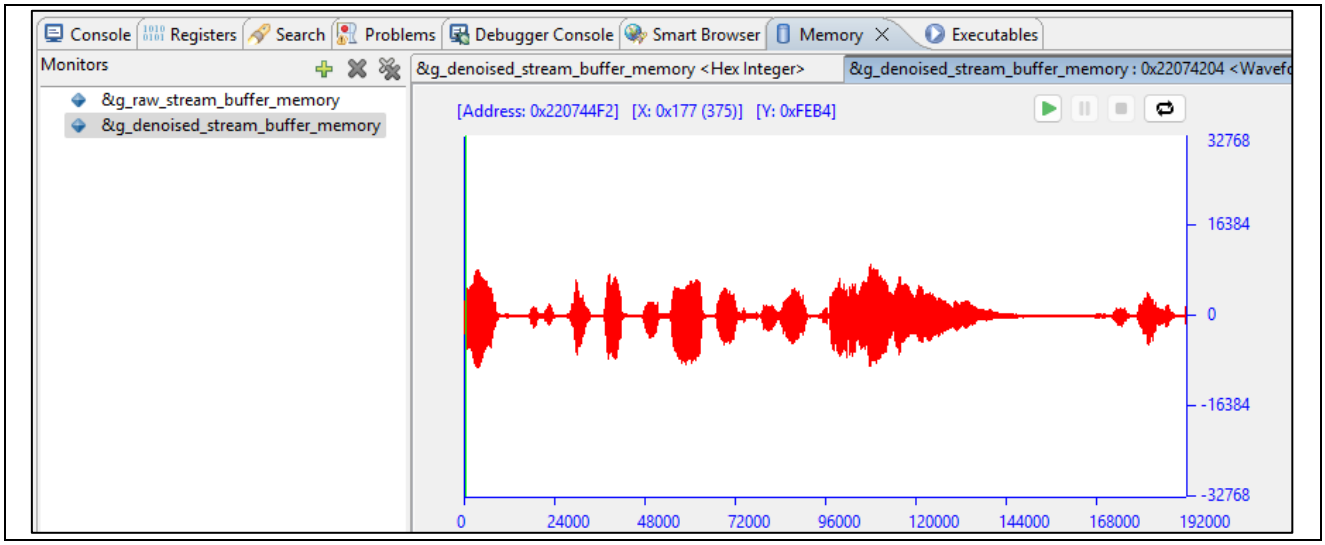


Figure 11. g_denoised_stream Stream Buffer memory view

2.3.3 SEGGER SystemView Observation

SEGGER SystemView can be used to observe the system-level behavior of the Audio AI Workflow on RA8 application, including task execution, interrupt activity, and timing behavior. The sample application provided with this project is already configured to use SEGGER SystemView with default settings.
For integrating SEGGER SystemView into a custom project, refer to the SEGGER SystemView User Manual.

The SEGGER source files, configuration files, and required recorder initialization code are included in the project so that runtime events can be captured through the J-Link interface, see Figure 12.

When porting SystemView to another project, refer to the SEGGER SystemView user manual as the main reference for the required source files, configuration files, initialization sequence, and porting procedure.

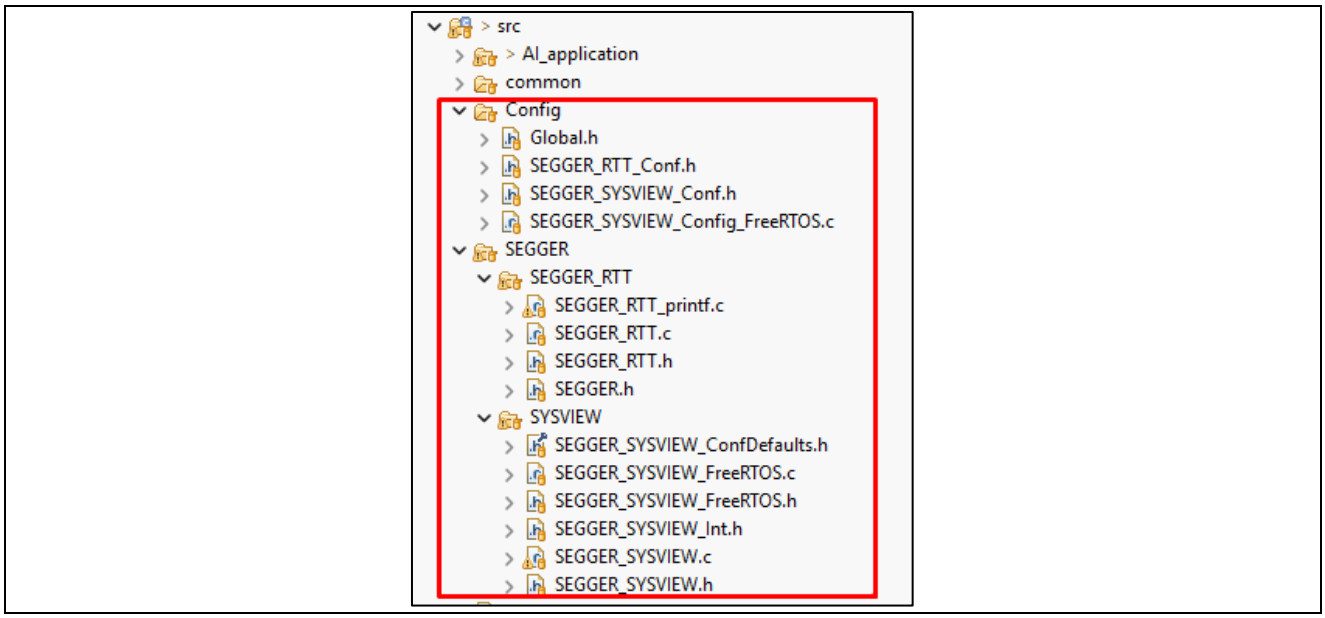


Figure 12. SEGGER SystemView files

The preprocessor symbols **"__SES_ARM"** and **"__ARM_ARCH_8M_BASE__"** are also defined for SystemView configuration on the Cortex-M85 based RA8 MCU. These symbols can be added from the project properties by right-clicking the application name and selecting **Properties**, then navigating to **"C/C++ Build"** → **"Settings"** → **"Tool Settings"** → **"Compiler"** → **"Includes"** → **"Macro Defined (-D)"**.

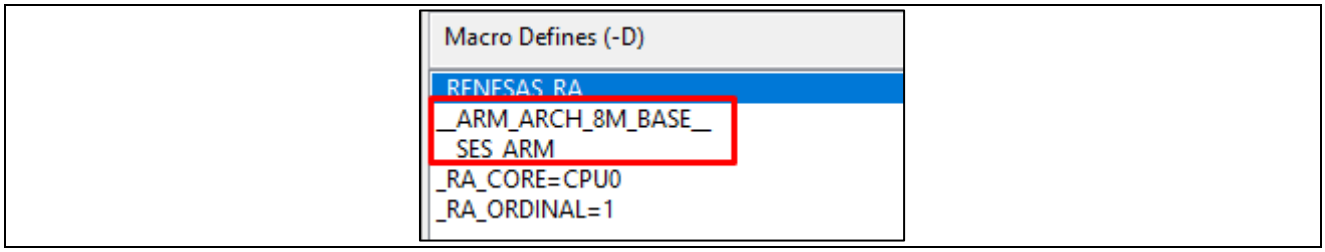


Figure 13. SEGGER SystemView Preprocessor symbols

Open SEGGER SystemView software, follow these steps below to create a recording project for observation.

Step 1: Create new Project

Choose **"File"** → click **"New project"** then fill Project Name and browse to your workspace to save the recording then click **"Next"**.

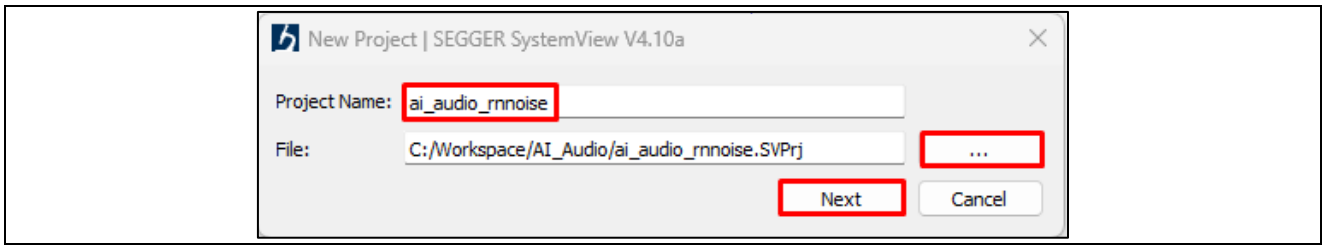


Figure 14. Create new project with SystemView

Step 2: Configure the Recorder

Select J-Link as SystemView Recorder then click **"Next"**.

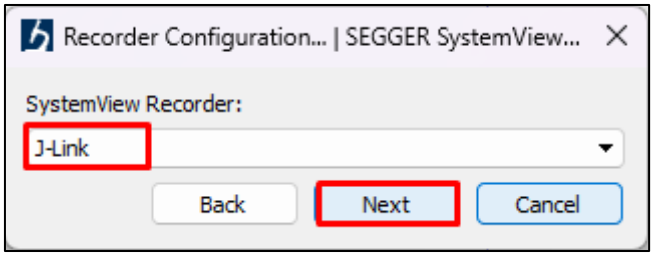


Figure 15. Configure the Recorder

Next, configure the Recorder Configuration as Figure 16, click **"Finish"** to establish the connection.

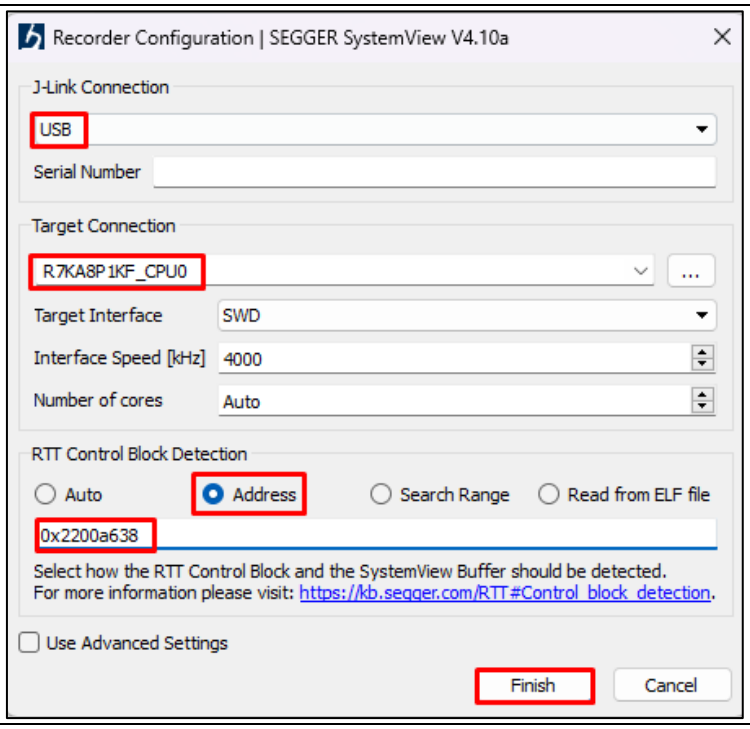


Figure 16. Recorder Configuration

Step 3: Start the Recorder

Click **"Target"** → **"Start Recording"** → the Recorder Configuration box may appear again, just click **"Next"** then **"Finish"** to continue.

During recording, switch between **denoise** and **raw** modes to observe differences in the system runtime timeline. Then click the **"Stop"** icon "■" or choose **"Target"** → **"Stop recording"** to stop the recording.

For detailed analysis, see section 6.

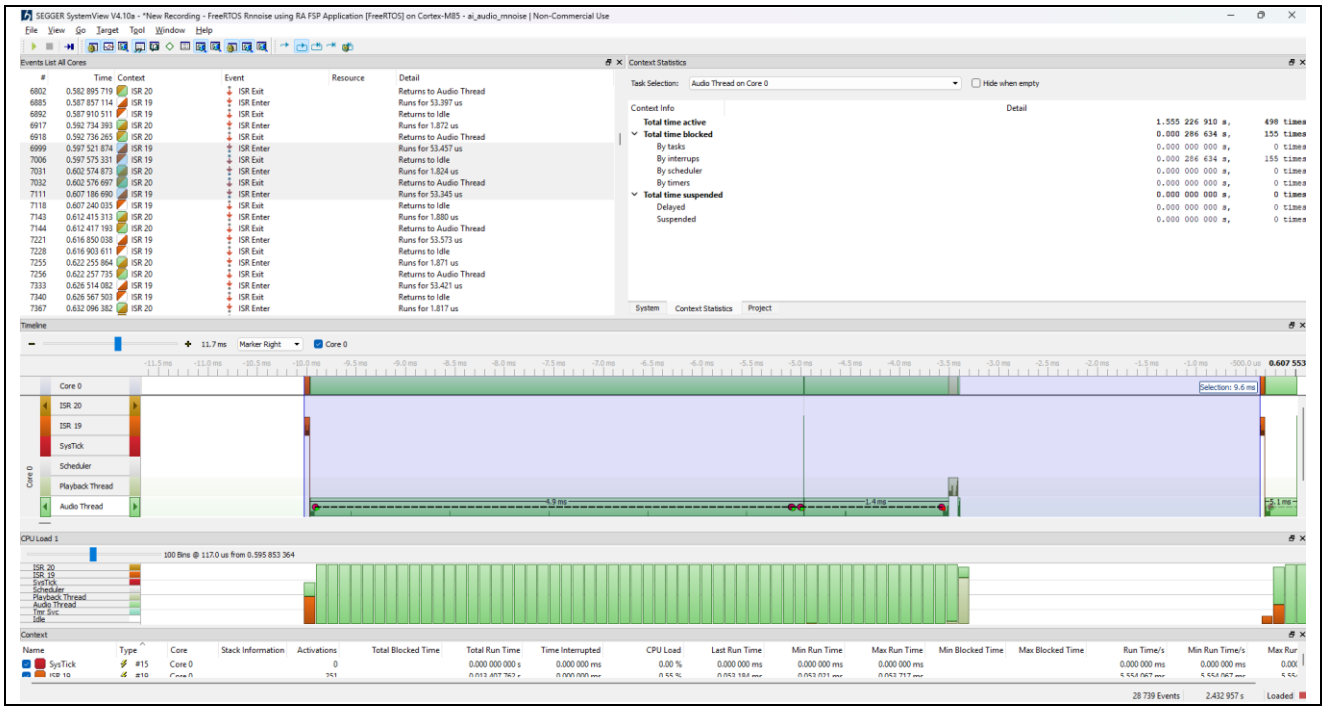


Figure 17. SEGGER SystemView GUI

3. Application Design Workflow

This section describes the application high level design, key model-related concepts, the model training process, and how the generated model is integrated into an FSP project.

3.1 System Architecture

3.1.1 High Level Design

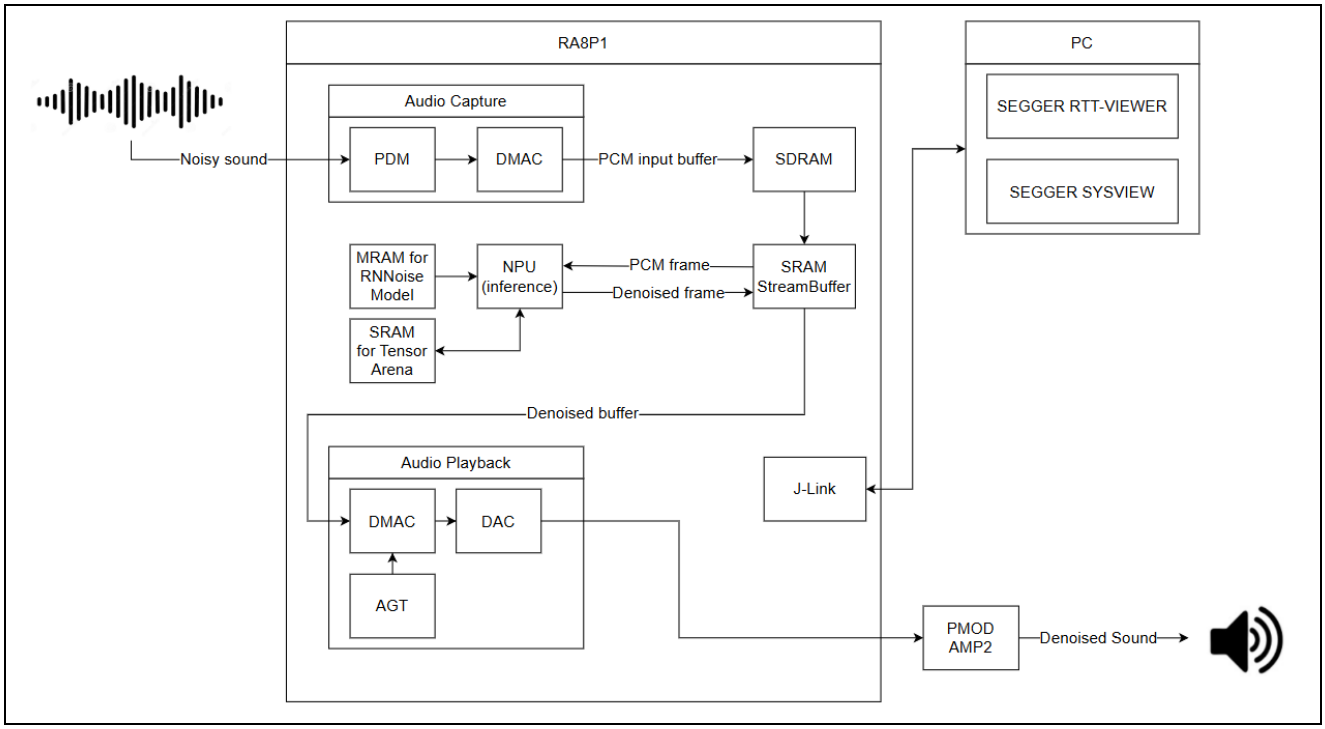


Figure 18. Application High Level Design

Figure 18 shows the High Level Design of the Audio AI Workflow on RA8 application.

The application is designed as a frame-based audio processing pipeline running on the RA8P1 platform, where audio flows sequentially from capture, through AI-based noise suppression, to playback.

The pipeline begins with the audio capture stage. Noisy sound is captured through the PDM microphone interface. The PDM interface converts the incoming 1-bit microphone stream into PCM audio samples. To minimize CPU involvement, this interface works together with the DMAC to transfer PCM data directly into SDRAM via a PCM input buffer. This buffering mechanism decouples the capture stage from the processing stage, enabling continuous data acquisition without interruption.

From SDRAM, audio data is organized into fixed-length PCM frames for processing. Each frame corresponds to 10 ms of audio (480 samples at 48 kHz). These PCM frames are then fed into the RNNoise processing block, which is accelerated by the integrated NPU. The RNNoise model is stored in MRAM, while intermediate tensors are allocated in SRAM as the tensor arena. During inference, the NPU performs the neural network computation to estimate noise suppression gains. The processing stage includes pre-processing, neural network inference, and post-processing, producing a denoised audio frame as output.

The resulting denoised frames are written into a stream buffer located in SRAM. This buffer acts as an intermediate stage between processing and playback, ensuring that audio output can proceed smoothly even if processing latency varies slightly. The denoised data is then forwarded to the audio playback stage through a denoised buffer.

In the playback stage, DMAC is used to transfer audio data efficiently to DAC. The DAC converts the digital PCM samples into an analog signal, while the AGT timer provides the timing reference to ensure a correct sampling rate. The analog output is routed through the PMOD AMP2 module, which amplifies the signal and drives the connected speaker, producing the denoised sound in real time.

The application also includes runtime control and system observation features. SEGGER RTT Viewer is used as the user interface while the application is running. Through this interface, users can switch between raw audio and denoised audio to compare the effect of noise suppression on the output sound. SEGGER SystemView is also connected through the J-Link interface to monitor the system in real time. It helps users observe task execution, interrupt activity, timing behavior, and overall system performance while the audio pipeline is operating.

3.1.2 Data Flow

The system processes audio in a frame-based pipeline, consisting of three main stages: Capture, Denoise, and Playback.

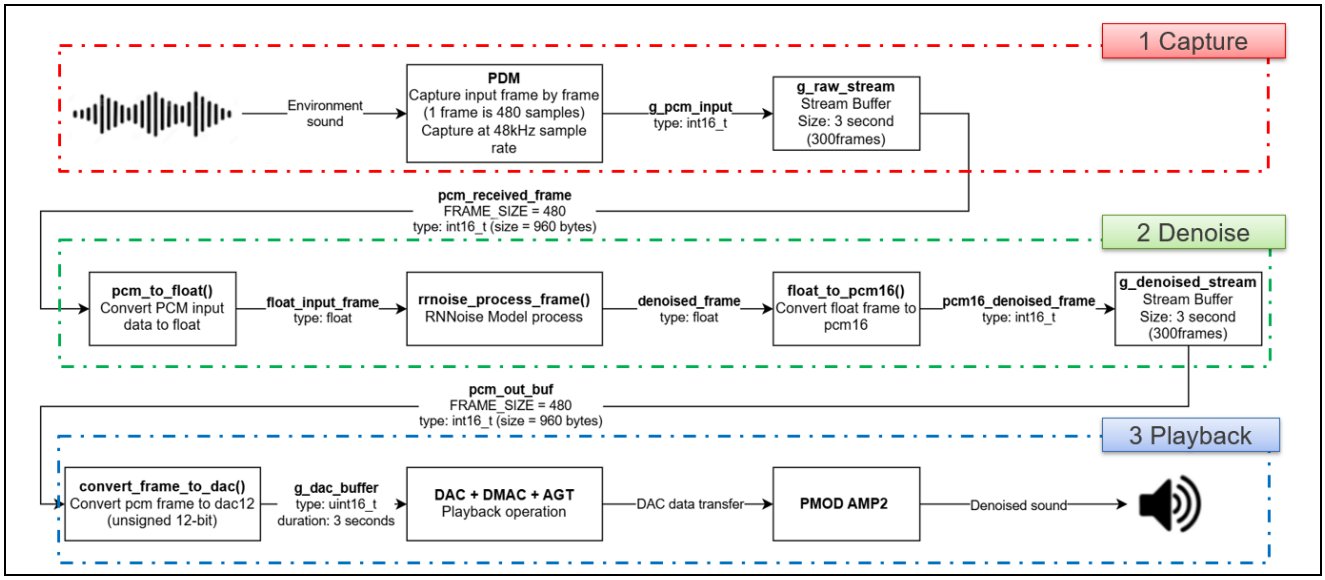


Figure 19. Application data flow

In the capture stage, environmental sound is continuously sampled by the PDM microphone and converted by the PDM interface into a 48 kHz PCM signal to meet the RNNoise model requirements. The PDM peripheral continuously captures incoming audio data and streams PCM samples into memory via DMA. The data is organized into fixed-size buffers, where each buffer contains 480 samples. A callback is triggered whenever a buffer is filled, enabling frame-based processing. The PDM peripheral converts the incoming 1-

bit data stream into 16-bit PCM samples. These PCM samples are then transferred by the DMAC and stored in SDRAM as the PCM input buffer.

From this buffer, PCM frames are transferred to the processing pipeline using a FreeRTOS stream buffer . This mechanism enables asynchronous data exchange and decouples the capture and denoise threads.

In the denoise stage, PCM frames are read from the raw stream buffer and processed sequentially. Each input frame is first converted from PCM (int16) to floating-point format to match the input requirements of the pre-processing. The floating-point frame is then passed to the `rnnoise_process_frame()` function, where noise suppression is applied. The resulting denoised frame is still in floating-point format and is subsequently converted back to PCM16 format. The processed PCM frames are stored in a dedicated denoised stream buffer to be consumed by the playback stage.

In the playback stage, denoised PCM frames are retrieved from the output buffer and prepared for audio output. Each frame is converted from PCM16 format to the DAC12 input format (unsigned 12-bit resolution). The converted data is then fed into DAC using DMAC, with timing controlled by the AGT timer to ensure continuous and synchronized playback. Finally, the analog signal generated by the DAC is amplified by the PMOD AMP2 module and delivered to a connected speaker.

This buffered, frame-based architecture allows each stage (capture, processing, and playback) to operate independently, improving system robustness and enabling audio processing without blocking between tasks.

3.2 Audio processing with RNNoise

RNNoise is a neural network for noise reduction that suppresses background noise while preserving speech quality. This project uses a quantized TensorFlow Lite (TFLite) version of the model. The model takes signal-processing features as input and outputs gain values used to attenuate noise. It also performs voice activity detection (VAD) to estimate whether speech is present in the input signal. For more information, refer to the Arm-Examples/ML-zoo repository and RNNoise: Learning Noise Suppression article.

3.2.1 Frame Processing Pipeline

The frame processing pipeline is the core algorithm of RNNoise. It operates on fixed-size audio frames (480 samples per frame, corresponding to 10 ms at a 48 kHz sampling rate) and applies a sequence of signal processing and neural network operations to suppress noise while preserving speech quality.

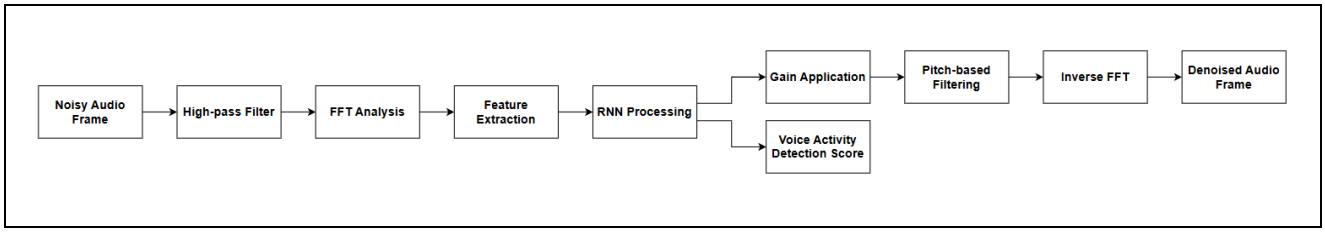


Figure 20. Single frame denoise process with RNNoise

Frame Input and High-pass Filtering

Each input frame is first preprocessed using a high-pass filter to remove DC offset and very low-frequency components, which are typically not relevant for speech but may introduce unwanted noise.

The biquad filter uses coefficients a = [-1.99599, 0.99600] and b = [-2, 1] to implement a high-pass filter with a cutoff frequency of approximately 50 Hz.

Frame Analysis

The filtered frame is transformed into the frequency domain using FFT (Fast Fourier Transform) for spectral analysis.

The frame analysis stage applies a half-window function before the FFT to reduce spectral leakage. It uses an FFT size of 960 samples (WINDOW_SIZE), which produces 480 complex frequency bins (FREQ_SIZE). It then computes energy across 22 critical bands (NB_BANDS) based on the Equivalent Rectangular Bandwidth (ERB) scale.

Feature Extraction

The spectral representation is used to derive a set of features for the neural network.

The feature extraction stage estimates the fundamental frequency through pitch analysis, calculates band energy correlation between the current frame and the pitch-delayed frame, and applies a Discrete Cosine Transform (DCT) to the log band energies to decorrelate spectral features. The final feature vector contains 42 elements (NB_FEATURES).

RNN Processing

The extracted features are processed by a recurrent neural network (RNN) to estimate optimal gain values for noise suppression.

The RNN accepts the 42-dimensional feature vector as input, processes the data using convolutional and GRU (Gated Recurrent Unit) layers, and outputs gain values for 22 frequency bands along with a voice activity detection (VAD) probability. It also maintains temporal context across frames through the RNNState structure.

Gain Application and Pitch Filtering

The estimated gain values are applied to the spectral components, with additional pitch-based filtering to further enhance speech harmonics.

The gain application stage interpolates band-level gains to obtain per-frequency-bin gains, applies temporal smoothing to reduce audible artifacts, and uses pitch filtering to reinforce harmonic structures in speech signals.

Frame Synthesis

The processed spectral frame is then converted back to the time domain.

The synthesis process performs an inverse FFT (IFFT) to reconstruct the time-domain signal, applies the same window function used during analysis, and uses an overlap-add method with the previous frame to ensure smooth signal continuity. It also stores the second half of the current frame so it can be overlapped with the next frame.

System Integration in a Frame-Based Pipeline

Figure 21 illustrates how RNNoise can be integrated into a frame-based audio processing pipeline.

In the audio capture stage, input audio from the microphone is handled by the audio driver and segmented into fixed-size buffers of 10 ms, ensuring alignment with the RNNoise frame size. These buffered frames are then passed to the processing stage.

The RNNoise processing block applies noise suppression on a per-frame basis using the `rnnoise_process_frame()` function. The denoising operation is managed through the `DenoiseState` structure, which maintains internal state information across frames. In parallel, voice activity detection (VAD) is performed to estimate speech presence in the input signal.

Finally, the processed audio frames are written to the output buffer in the audio output stage. The enhanced signal can then be used for playback or transmission, depending on the application requirements.

For streaming audio applications, process incoming audio in 10 ms frames, 480 samples per frame, and use the returned VAD probability to determine whether speech is present.

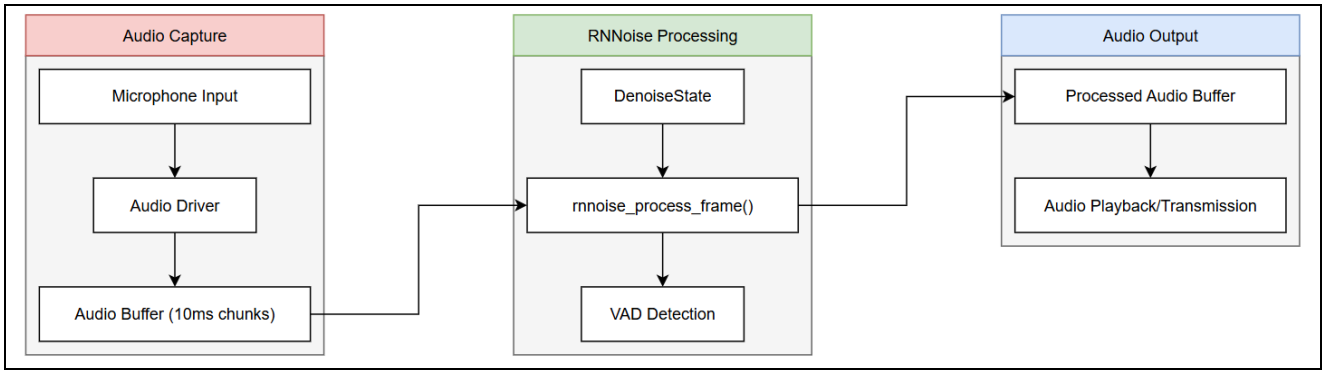


Figure 21. RNNoise integration into an application overview

3.3 AI Audio Model Generation and Integration

This application uses a pre-trained model obtained from ML-Zoo for the RNNoise processing described in Introduction.

To recreate the RNNoise model, refer to the workflow in section 3.3.1.

3.3.1 How to run the tool to generate the AI Audio Model

To train an RNNoise model, both clean speech data and noise data are required. All audio data must be sampled at 48 kHz and stored in 16-bit PCM format (machine endian).

Final model performance depends on several factors, including the diversity and composition of the training dataset and the total training duration.

The following steps outline a high-level workflow for training an RNNoise AI model.

This section describes the procedure for recreating, training, and evaluating the RNNoise model using the provided scripts and dataset.

Before proceeding with the steps below, clone the ML-Zoo repository, the original RNNoise repository, or both, depending on the method you choose. ML-Zoo provides scripts for recreating the model and evaluating its performance after training, but the overall process typically takes longer. In contrast, the original RNNoise repository is more complex to use, but each individual step generally runs faster.

The following steps start with the ML-Zoo repository. Folder `recreate_model` in path `ML-zoo\models\noise_suppression\RNNoise\tflite_int8\` contain necessary file, script to recreate model.

Step 1: Dataset preparation

To reproduce the RNNoise model accurately, it is required to use the same dataset utilized during the original training process. (see .\`get_data.sh`)

This script downloads the Noisy Speech Database and extracts it into separate clean speech and noisy speech directories for training and testing.

Step 2: Generation of Training and Testing Data (`.h5`)

The RNNoise training pipeline requires preprocessed `.h5` (HDF5) files that contain input feature vectors and corresponding training labels.

There are two methods available to generate the `.h5` file

Method 1 - RNNoise Reference Pipeline (Recommend)

This method follows the original RNNoise preprocessing pipeline and is recommended for accuracy and performance by combining clean speech and noise files into separate `.wav` files, then running the RNNoise feature extraction pipeline to generate training features.

An example helper function is provided: `create_combined_wavs(...)`
(see `data.py` of ML-zoo repository)

This method is preferred because it replicates the original training workflow, improves preprocessing speed, and maintains consistency with the RNNoise feature extraction process.

For users who want to follow this method, refer to the official RNNoise repository (rnnoise) for the original detailed documentation.

This application project provides a Google Colab notebook that can be run directly to recreate and train an RNNoise model (see `rnnoise_recreate_and_training_model.ipynb` in application project package)

Method 2 - Python-Based Preprocessing (Alternative)

As an alternative, a Python-based preprocessing script is provided:

```
> python data.py \ --clean_train_wav_folder=./clean_trainset_56spk_wav \
                   --noisy_train_wav_folder=./noisy_trainset_56spk_wav \
                   --clean_test_wav_folder=./clean_testset_wav \
                   --noisy_test_wav_folder=./noisy_testset_wav
```

This method directly processes the audio folders and generates `train.h5` and `test.h5` files.

However, it should be noted that this approach is significantly slower than the recommended method 1 above.

Step 3: Model Training and Quantization

Once the `.h5` datasets are generated, the model training and quantization process can be initiated using `train.py` and `convert.py` (see `train_and_quantise_model.sh`)

This step will generate `rnnoise_INT8.tflite` as quantized model file and `rnnoise_FP32.tflite` as non-quantized model.

To evaluate the model performance, run the following Python script

This evaluation uses the PESQ (Perceptual Evaluation of Speech Quality) score to measure speech quality automatically. PESQ is an international standard that estimates how clear and natural the processed speech sounds to human listeners in communication systems. The ML-Zoo repository provides a script for evaluating model performance after training. The PESQ score is generated by the `test.py` script and may vary depending on the training dataset and training duration.

```
> python test.py  --clean_wav_folder=./clean_testset_wav
                  --noisy_wav_folder=./noisy_testset_wav
                  --tflite_path=<path_to_tflite>
```

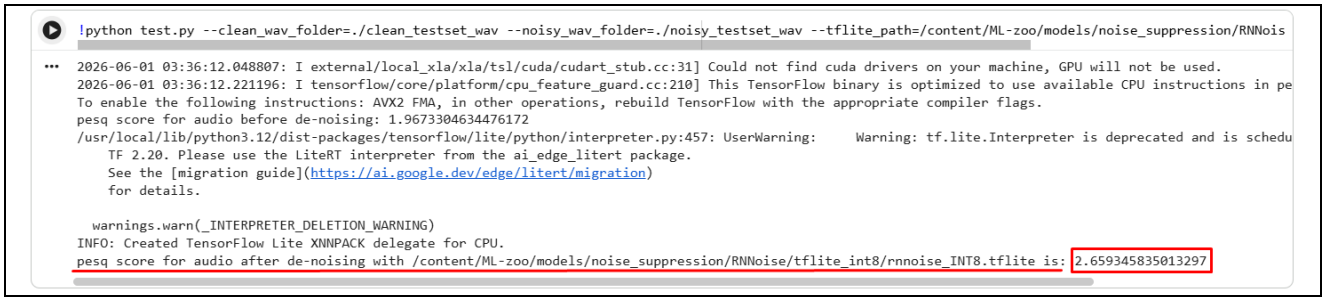


Figure 22. Evaluate the model performance

Note that some provided scripts by ML-zoo may be outdated and may require updates or modifications before use.

3.3.2 How to convert AI processing to C source code and include it into an FSP project

After obtaining the `rnnoise_INT8.tflite` file after completing step 3 in section 2.3.2 or use pre-trained model provided by ML-zoo, follow this section for the process of converting a quantized AI model into C source code, enabling seamless integration into an FSP project.

Users should read the RUHMI workflow in advance or have basic knowledge of it before performing the following steps. Follow the RUHMI workflow QSG (r11qs0065ej0101-ruhmi-framework-qsg) to install the AI tools and integrate them into the e2studio IDE.

There are two integrated tools that users need to be aware of:

- **AI Navigator**: Allows you to select either a sample application (AI application example) or your own project with a converted model.
- **Conversion Tool**: Provides model conversion functionality using your own input module.

AI Navigator not only allows users to run sample applications but also provides support for developing their own AI applications.

The following steps describe how to convert AI model into C source code and integrate it into an FSP project. These steps can be directly applied to an existing project, except for the project creation step.

Step 1: Create a project

Firstly, create an FSP project as usual. Select **“File”** → **“New”** → **“Renesas C/C++ Projects”** → **“Renesas RA”**, then choose **“Renesas RA C/C++ Project”** and click **“Next”** to provide project name.

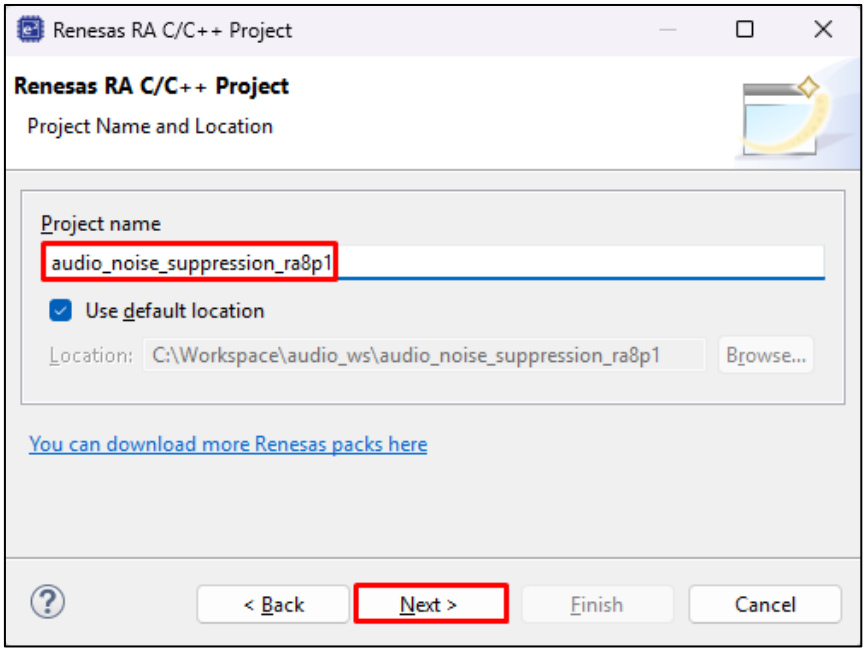


Figure 23. Create a new project

Select project Device and Tools as Figure 24 then click **"Next"** and create application project as usual.

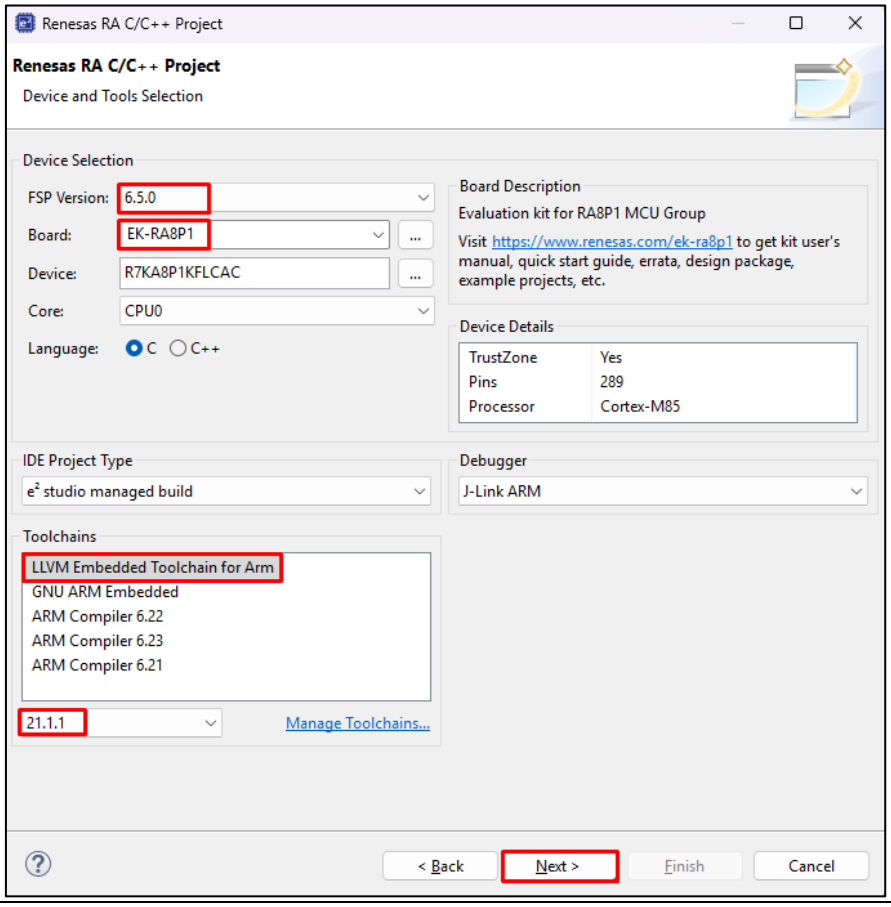


Figure 24. Select project Device and Tool Selection

After that, add the following stacks if users want to use the Ethos accelerator:

- Google TFLM Core Library
- Google TFLM CMSIS-NN Kernel

On the Stacks tab, add **“New Stack” “AI” → “Google TFLM Core Lib”**.

Then Click on **“Add CMSIS-NN Kernel” → “New” → “Google TFLM CMSIS-NN Kernel”**.

Once these are added, the required dependent stacks (such as Arm Ethos-U Core Driver, Arm CMSIS-NN Library Source, and Arm CMSIS-DSP Library Source) will be automatically included.

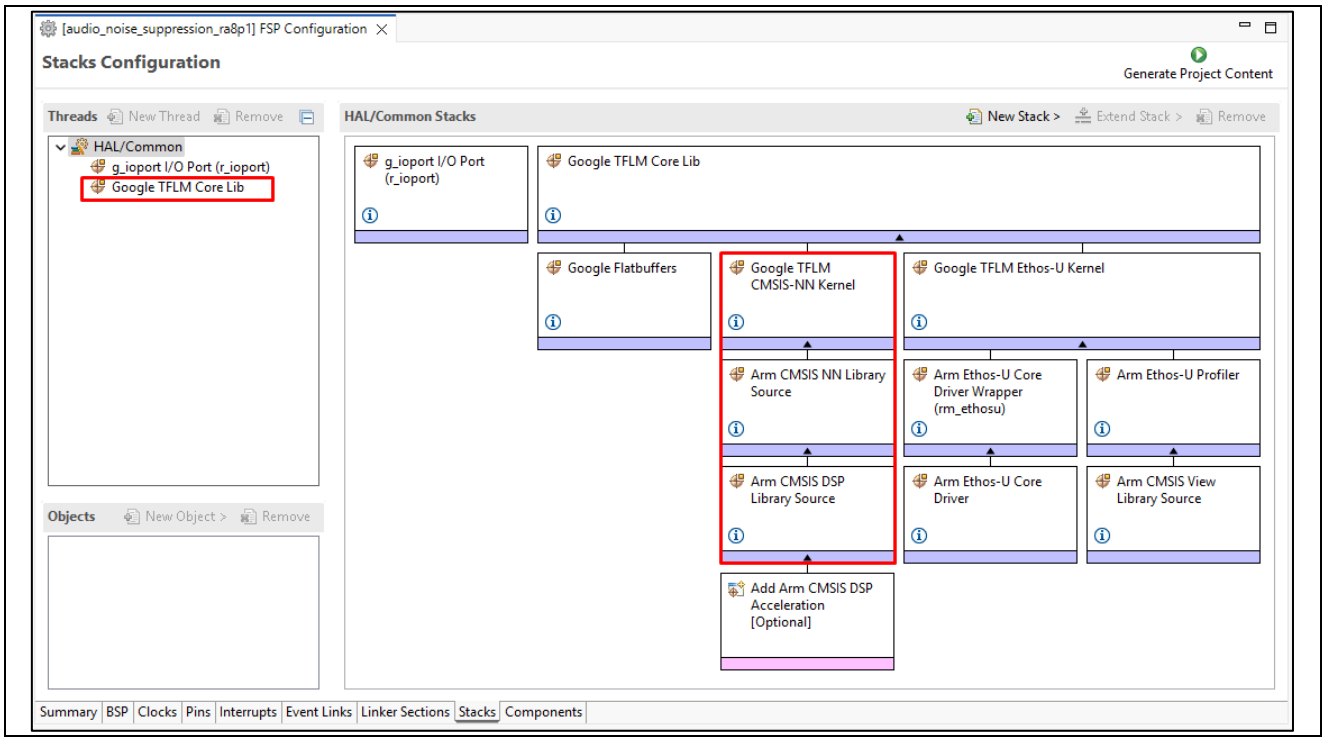


Figure 25. Add Arm CMSIS NN and DSP Library source

Step 2: Work with AI Navigator

On e2studio workspace, select **“Renesas AI”** → Click **“AI Navi”** to open AI Navigator perspective.

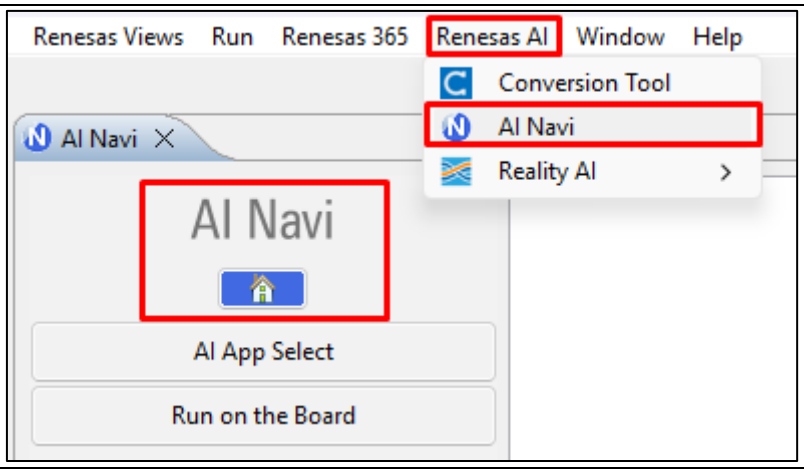


Figure 26. Open AI Navigator perspective

Next, choose **“Use Your Project & AI Model”** → a Select project box shows up → select your desire project → **“Finish”**.

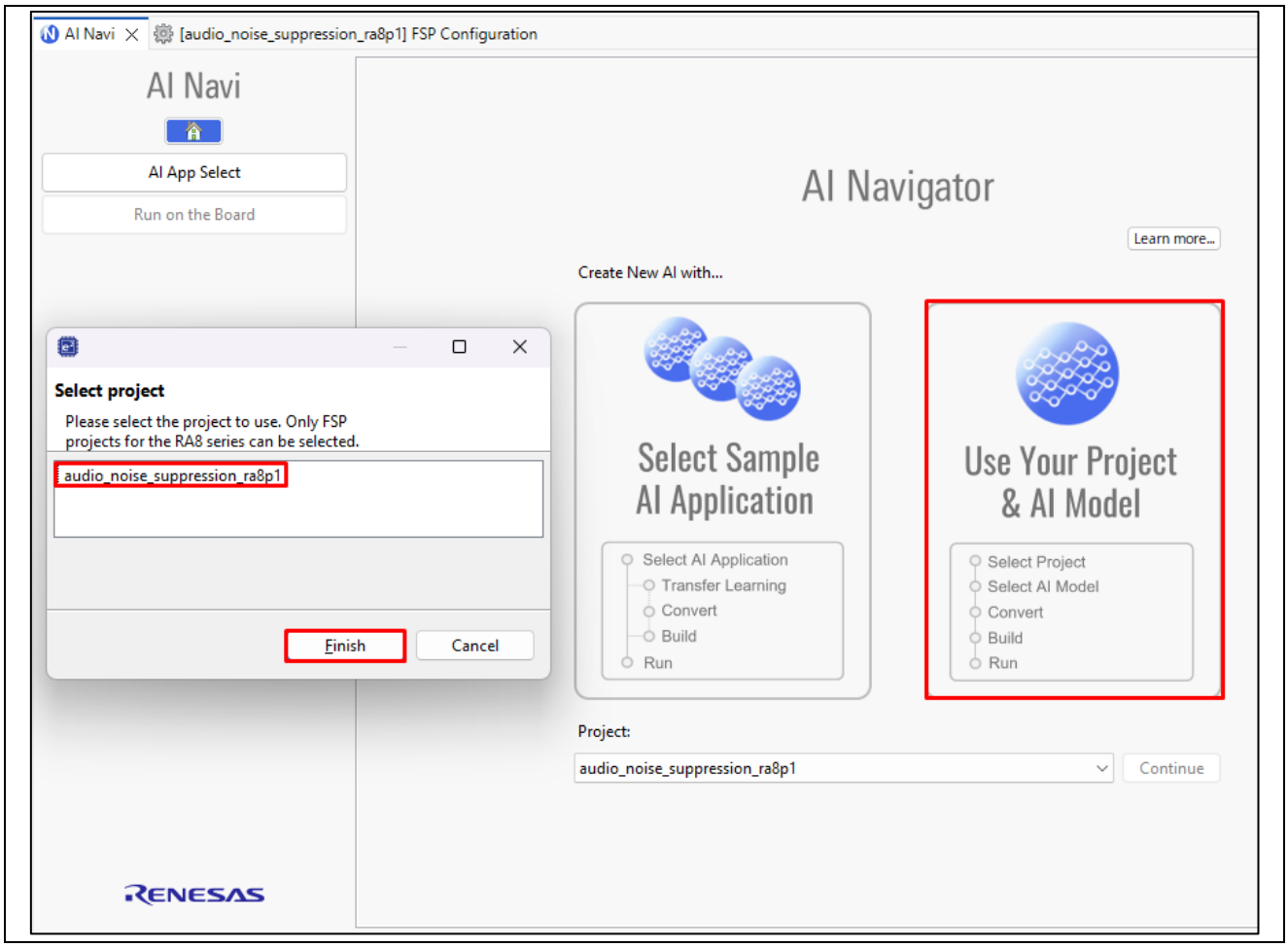


Figure 27. Select desired project

Click **"Use AI Model on Your PC"** → Select your AI model (`*.tflite / *.onnx / *.pte`).

Note: Please avoid using spaces in your AI model name. The AI model conversion may not succeed.

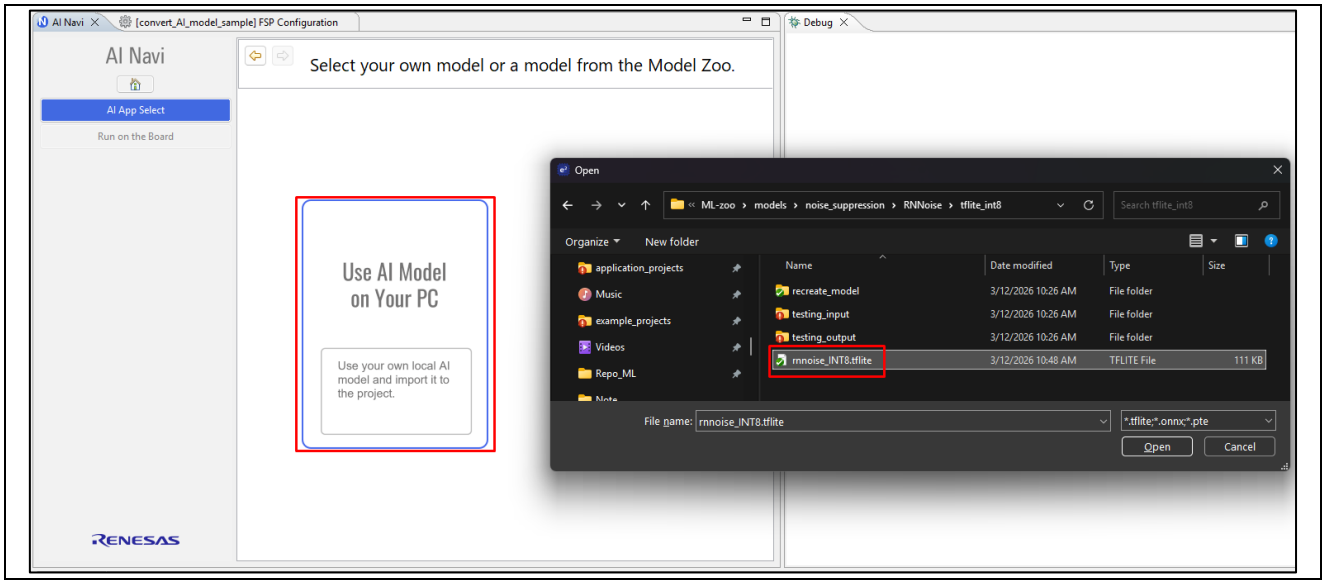


Figure 28. Select target model

The Project and AI information view will appear when the project and AI model are imported successfully.

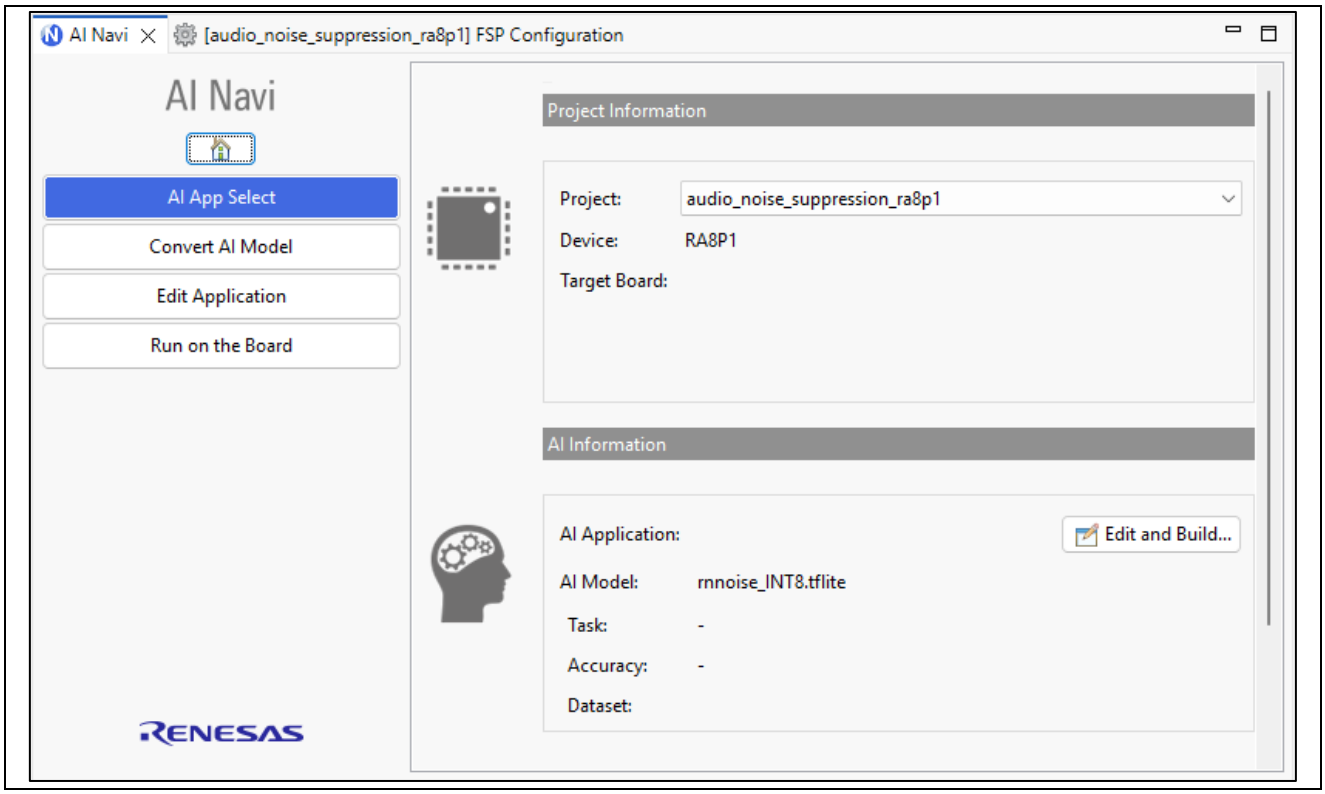


Figure 29. Project and AI information view

Step 3: Convert AI model

Click **"Convert AI Model"** → then click **"Convert…"** to launch Conversion Tool.

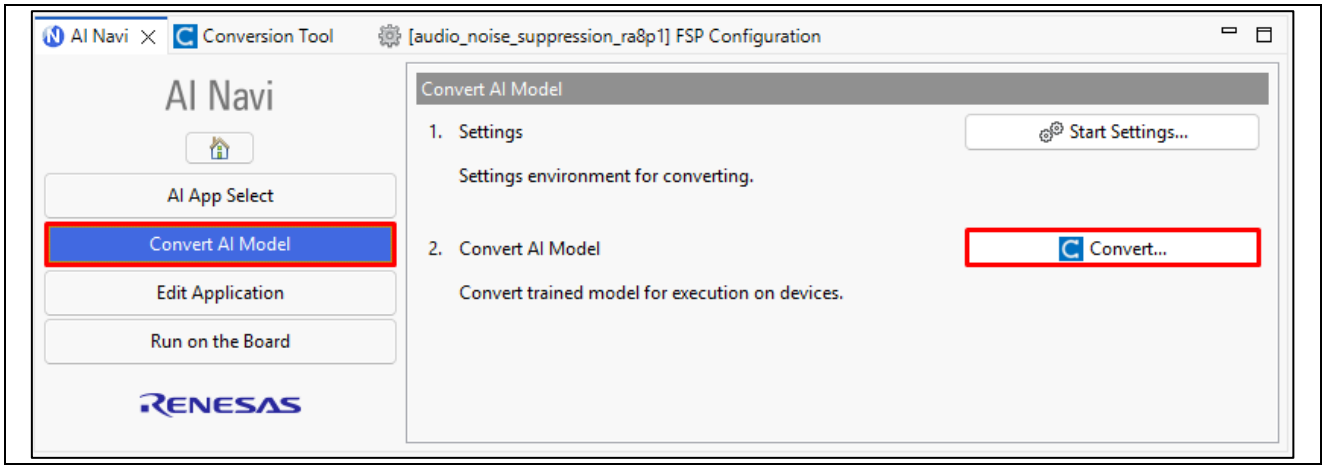


Figure 30. Open Conversion Tool

Input File Settings

In the Conversion Tool GUI, configure the following settings:

- Project name: Select your current working project
- Device: Select your target device
  - RA8P1 (CPU + Ethos-U55): Select this option when targeting an RA8P1 device with Ethos-U55 NPU acceleration. The conversion tool partitions supported neural network operators for NPU execution, while unsupported operators remain on the CPU.
  - RA8 Series (CPU only): Select this option when targeting an RA8 device without Ethos-U acceleration, or when the model should run entirely on the CPU. In this mode, inference is executed using software kernels such as CMSIS-NN or reference kernels.

- Tool: Select the RUHMI Framework AI Compiler
- Framework selection: Choose the framework that matches your AI model
    - ONNX (`.onnx`)
    - TensorFlow Lite (`.tflite`)
    - PyTorch (`.pth / .py`)
    - ExecuTorch (`.pte`)
- Input model file: Select the model generated in the previous step
- Output directory: Specify the directory for the conversion results

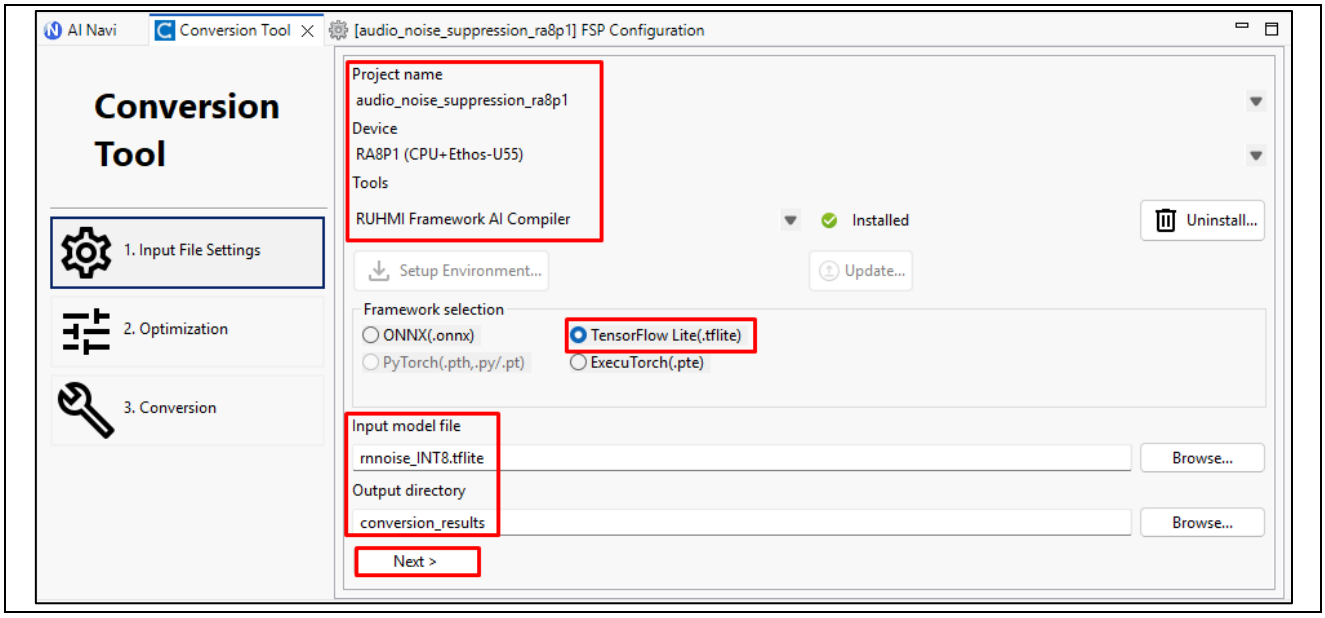


Figure 31. Conversion Tool Input File Settings

Click **"Next"** and wait for the process to be completed.

Optimization

In the optimization step, prepare and optimize the model according to the selected input model type:

- If a **quantized model** is already selected as input:
    - Verify the message indicating that a quantized model is loaded
    - Click **"Next"**
- If a **non-quantized model** is selected:
    - Configure quantization parameters
    - Click **"Start quantization"**
    - After successful quantization, click **"Next"**

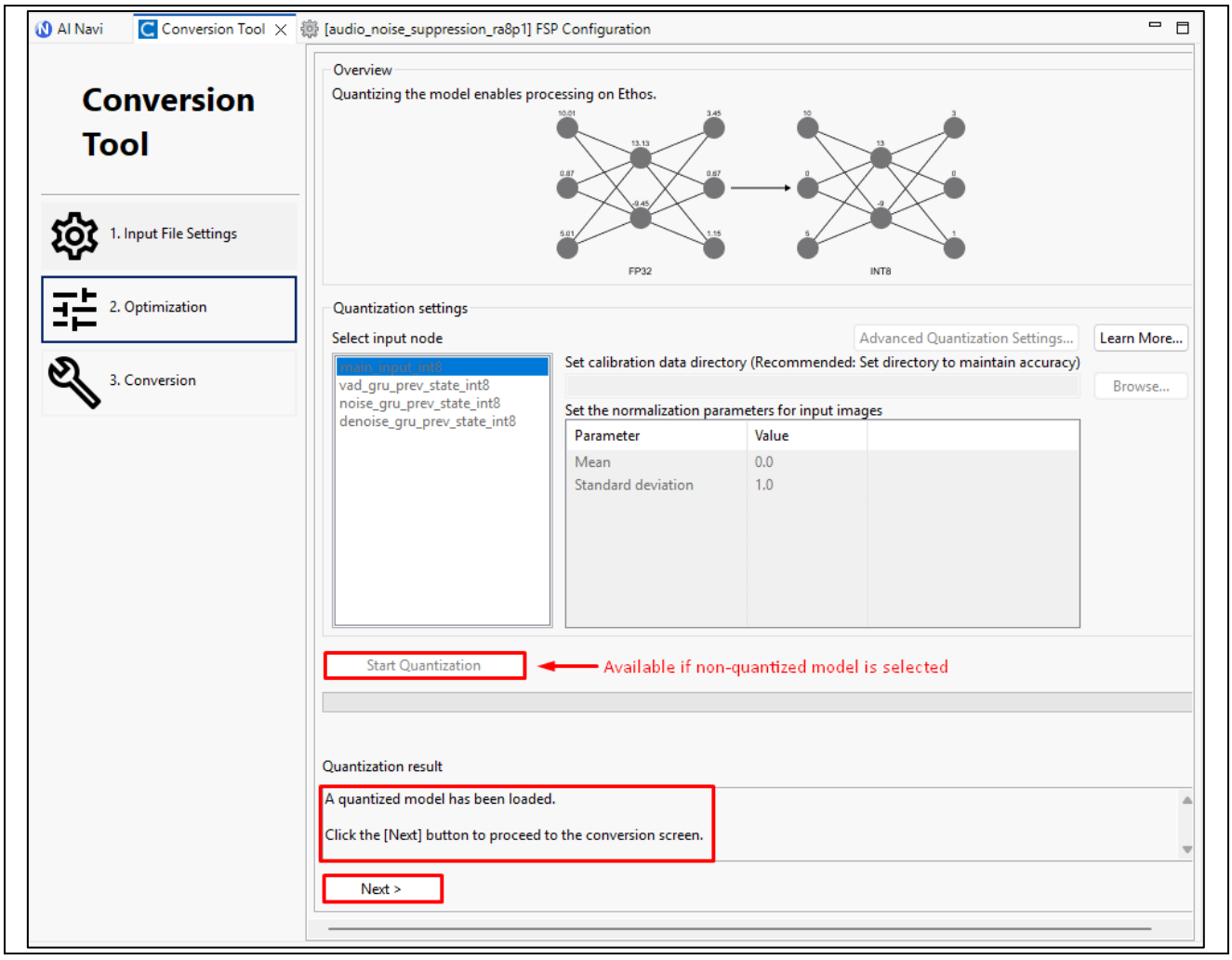


Figure 32. Conversion Tool Optimization

Conversion Option Settings

Target configuration: CPU + Ethos-U55

- Optimization mode
  - Performance (default): Optimize for maximum performance
  - Size: Optimize for minimal RAM usage
- Memory mode
  - Sram_Only (default): Place weights in internal memory
  - Shared_Sram: Place weights in external memory

After configuring the options, click **“Start conversion”**.

If the conversion is successful, a confirmation message will appear in the **Conversion Result** section.

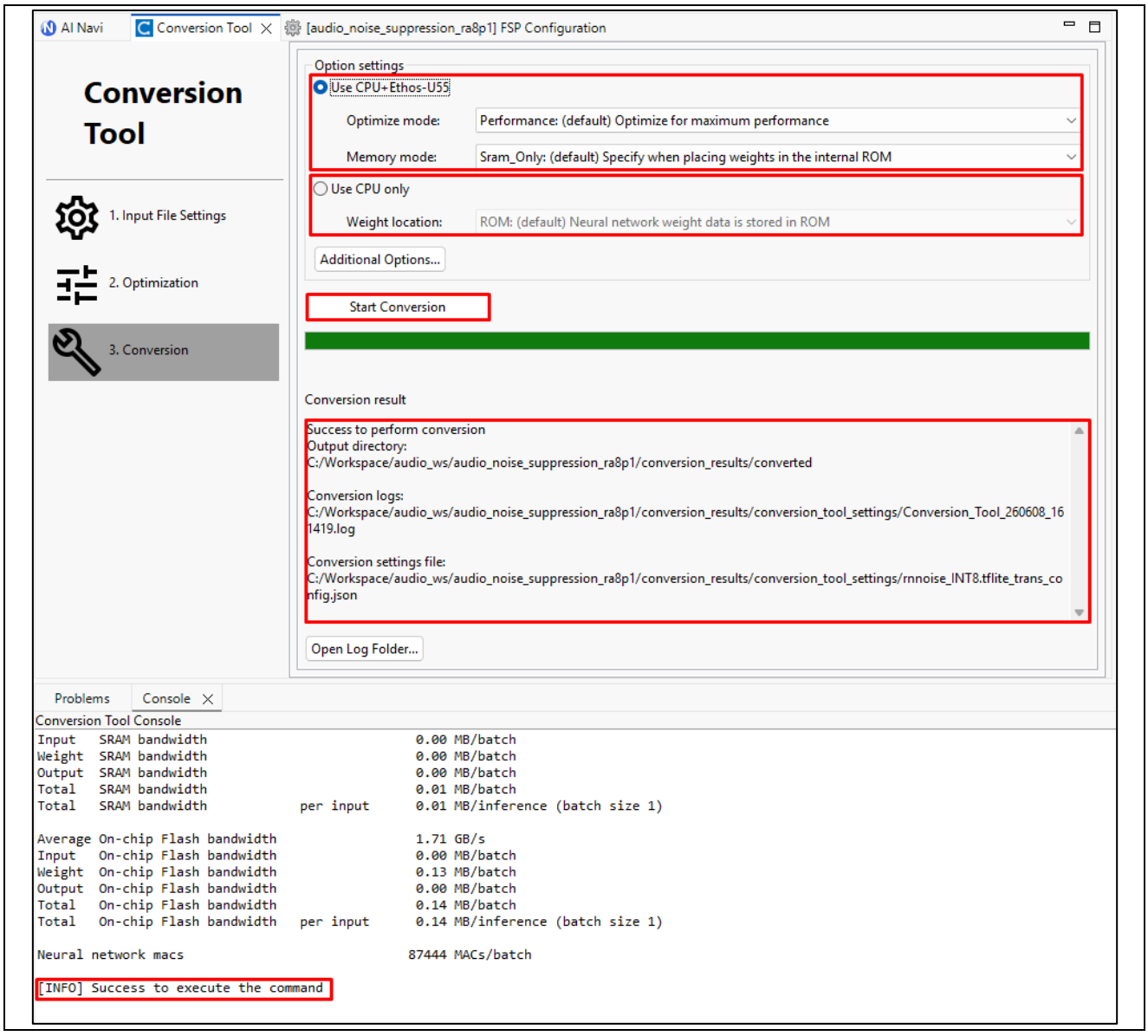


Figure 33. Conversion Tool Option settings and Start Conversion

Integration of Generated Model

You can review the conversion log by opening the generated log file, which is useful for troubleshooting or seeking support from Renesas.

See `./conversion_result` folder for converted Model C source code.

Check the converted output directory:

`./conversion_result/converted/build/MCU/compilation/src/`

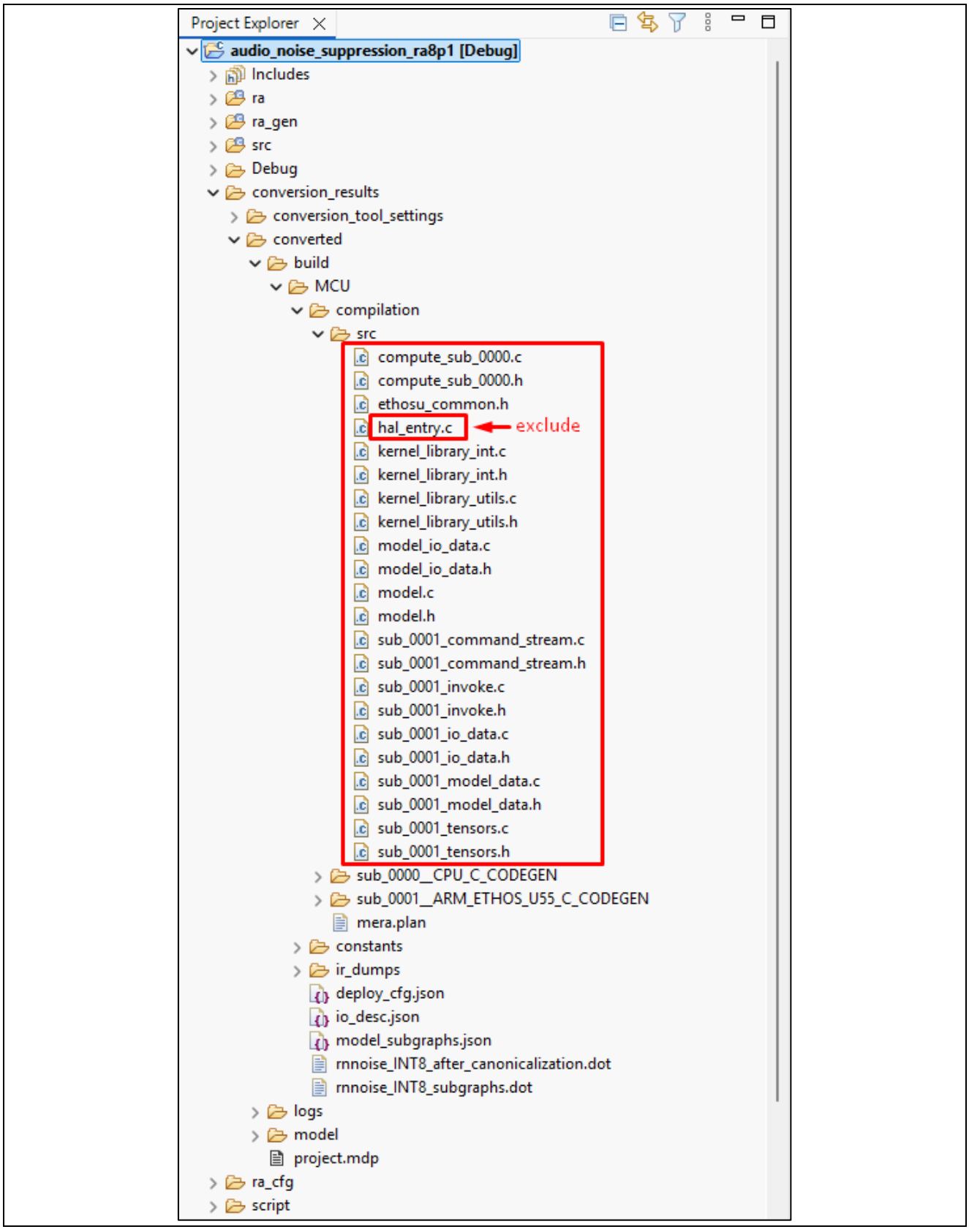


Figure 34. Conversion result

To integrate the model into your application, copy all source files from this directory into your project, excluding `hal_entry.c`, as it is provided only as a sample demonstrating how to run the AI model. You can refer to this file for guidance on how to simply use and integrate the model into your application.

4. Application Optimization

Application performance can be improved through several optimization techniques. This section introduces the methods commonly used in a project to enhance overall performance.

Arm® Helium™ technology, also known as the M-profile Vector Extension (MVE), is part of the Armv8.1-M architecture for Arm Cortex-M processors. It improves DSP and machine learning performance by using Single Instruction Multiple Data (SIMD) operations, which apply the same operation to multiple data elements in parallel. This application also applies MVE to optimize supported DSP and machine learning operations on the RA8P1 MCU. For detailed information on applying or verifying MVE in an application, refer to the High Performance with RA8 MCU using Arm® Cortex®-M85 core with Helium™ application note.

The LLVM Embedded Toolchain for Arm supports Helium™ instructions through the default compiler settings. When an RA8P1 project is generated using e² studio and the Flexible Software Package (FSP), the CPU and software settings are preconfigured for the Cortex®-M85 core and CMSIS Helium™ support, as shown in Figure 24.

The RA8 MCU family includes 128 KB of TCM memory, consisting of 64 KB ITCM (Instruction TCM) and 64 KB DTCM (Data TCM). TCM access is not available in CPU Deep Sleep, Software Standby, or Deep Software Standby modes. FSP initializes both ITCM and DTCM areas by default. The linker script defines sections for the ITCM and DTCM areas, making them easy to use in applications. Refer to the applicable MCU User's Manual for the TCM memory address ranges. TCM is not used in this application because it provides only limited performance improvement.

For more technical guidance on optimizing an application, refer to section **5. Optimizing Audio EQ Performance with Arm® Helium™ on RA8 MCU** in the DSP Design Workflow with RA8 MCUs application note. The optimization techniques described may be applied selectively, depending on their effectiveness in the application.

5. Application architecture

5.1 Thread Architecture

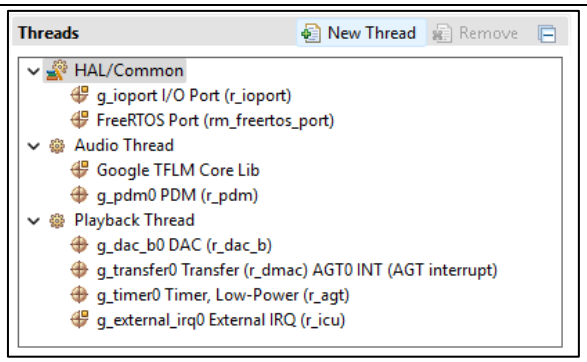


Figure 35. Application Thread Architecture

The application is organized around two main threads: the Audio Thread and the Playback Thread.

Table 1 Audio AI Workflow on RA8 application thread properties

Task	Stack size	Priority	Description
Audio Thread	0x10000	8	Handles audio capture and executes the RNNoise model for frame-based noise suppression.
Playback Thread	0x5000	11	Processes denoised audio for playback, including format conversion and DAC output, while also handling latency measurement and NPU timeline capture.

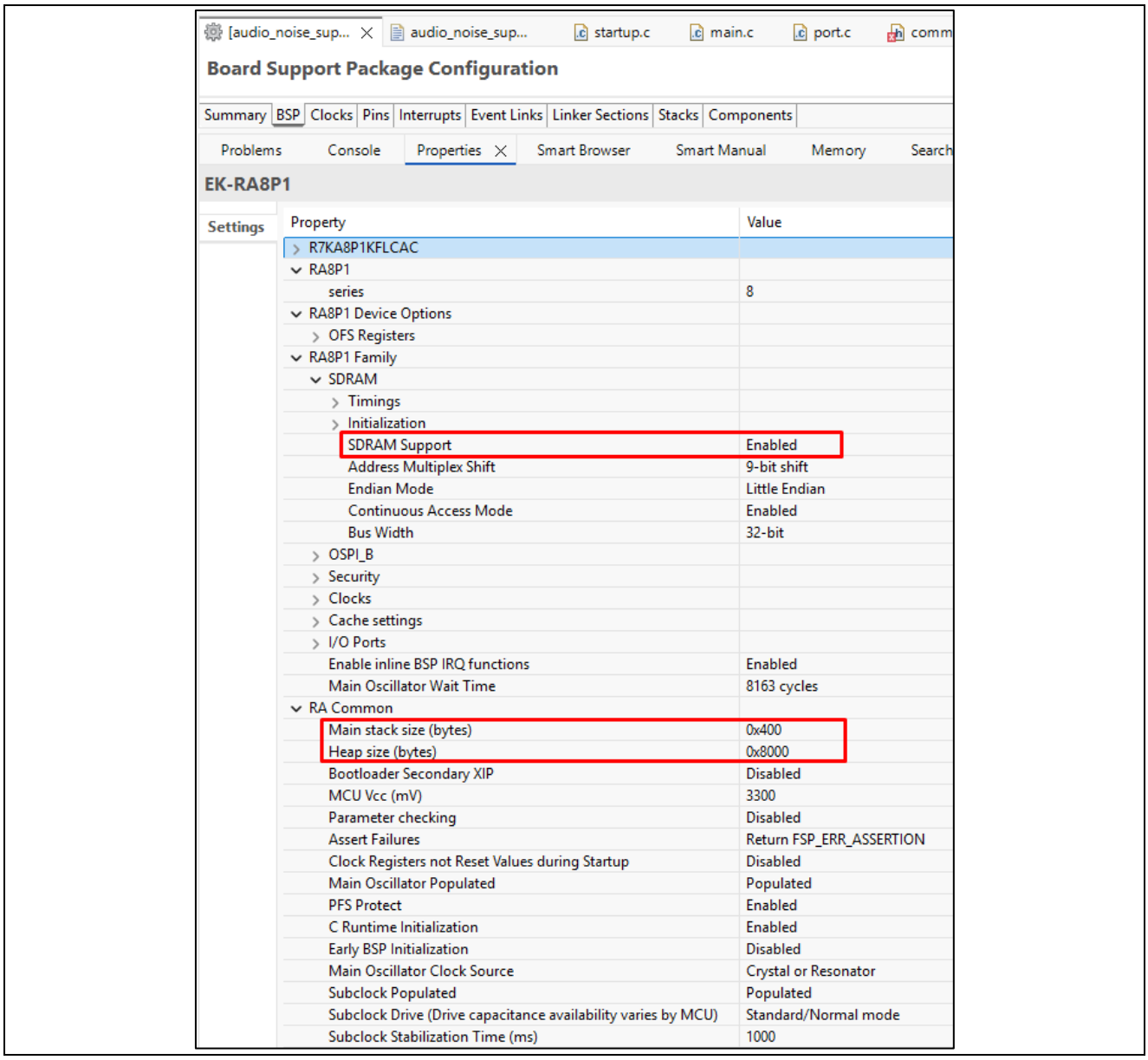


Figure 36. Board support package configuration

The data flow between these threads is implemented using two stream buffers:

- `g_raw_stream`: Transfers raw PCM audio data captured from the PDM interface to the denoising thread (AI model processing).
- `g_denoised_stream`: Transfers denoised audio data from the model output to the playback thread.

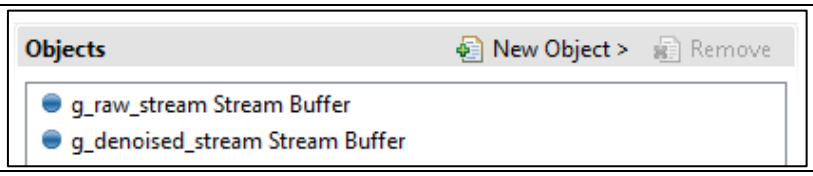


Figure 37. Application has two Stream Buffers

The Figure 38 shows the configuration of two Stream Buffers used in the application: `g_raw_stream` and `g_denoised_stream`. Both buffers are configured with the same parameters.

- Stream Buffer Size (Bytes): 384000
  This defines the total capacity of each stream buffer. It determines how much audio data can be buffered between producer and consumer threads.
- Trigger Level (Bytes): 960
  This specifies the minimum number of bytes required in the buffer before the receiving task is unblocked.
  In this case, the playback or processing thread will wait until at least 960 bytes are available before reading.
- Memory Allocation: Static
  The buffer memory is allocated statically at compile time, meaning no dynamic memory allocation (heap) is used.

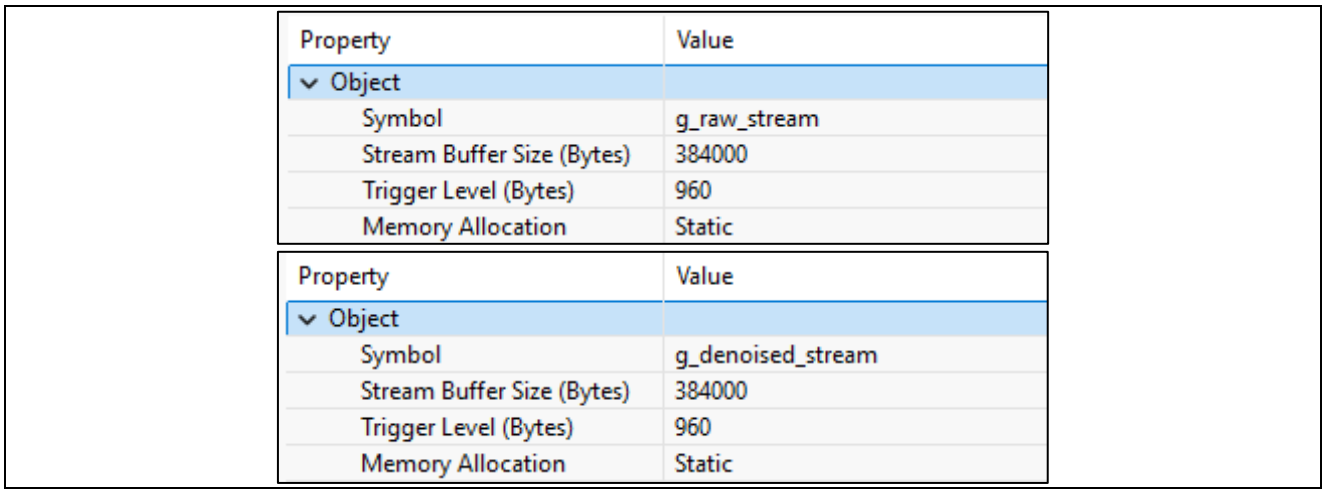


Figure 38. Stream Buffer properties

5.2 Thread Detail Configuration

5.2.1 Audio Thread

5.2.1.1 Google Core Lib configuration

The Google TFLM Core Library has added to support integration with the Arm CMSIS-NN and Arm CMSIS-DSP libraries.

TensorFlow Lite for Microcontrollers (TFLM) is a lightweight version of TensorFlow Lite designed to run machine learning models on microcontrollers, DSPs, and other memory-constrained devices. The TFLM CMSIS-NN kernels provide optimized neural network operations that are specifically tuned for Arm Cortex-M processors.

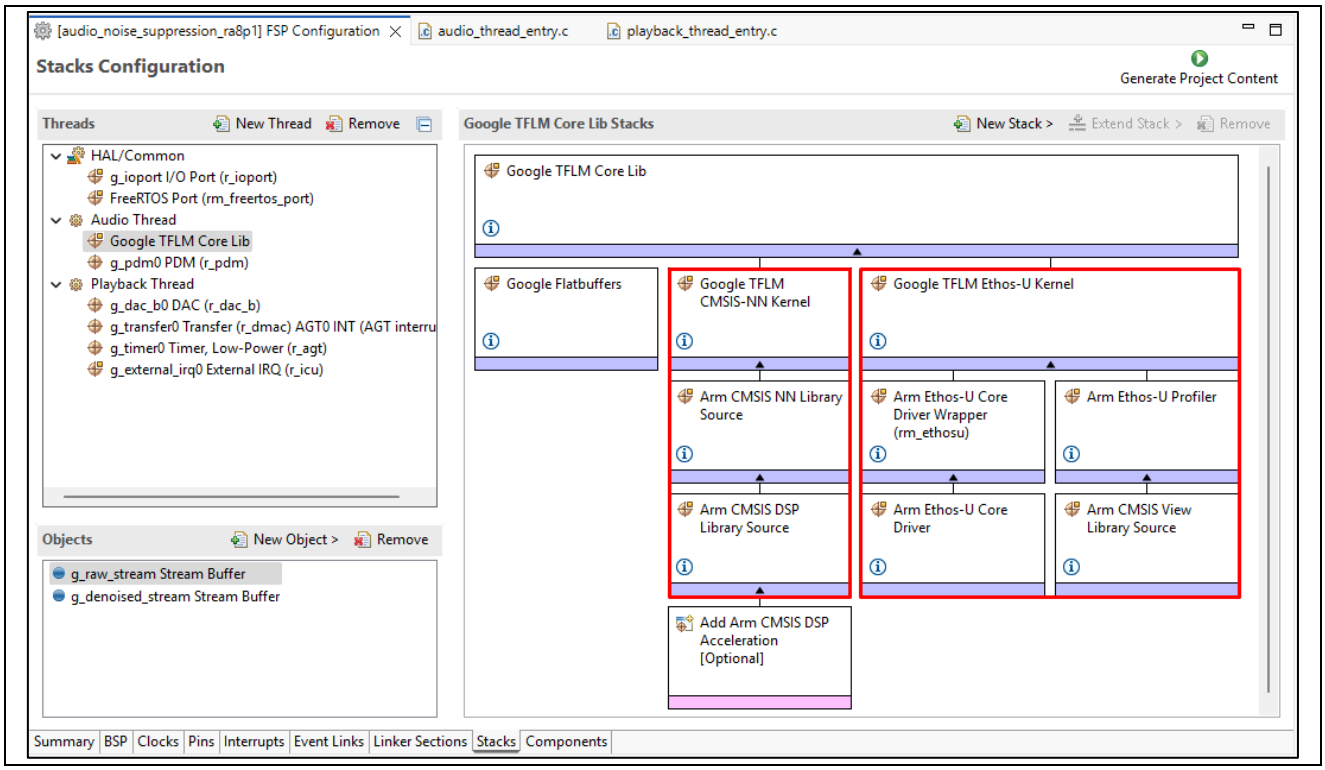


Figure 39. Google TFLM Core Lib configuration

Note that **CMSIS-DSP hardware acceleration is not available on RA8P1**. DSP functions from the Arm CMSIS library used for pre-processing and post-processing run on the CPU.

Before running a model on Ethos-U (NPU), inspect the model structure to identify which operators are used. Tools such as Netron can be used to visualize the model. Ethos-U55 accelerates only a subset of TFLite operators. If the model contains unsupported operators, those parts will fall back to the CPU. A model can also be executed entirely on the CPU if it is not optimized for NPU acceleration.

When Ethos-U (NPU) is enabled during model conversion to C source code, the model is partitioned into multiple subgraphs instead of being represented as a single block:

- Subgraphs executed on the MCU (CPU)
- Subgraphs executed on Ethos-U (NPU)

Each subgraph is implemented as a separate function. The output of one subgraph becomes the input of the next, and shared memory buffers are used to pass data between them.

For MCU execution, two kernel types are available: optimized CMSIS-NN kernels and reference kernels. The optimized CMSIS-NN kernels are recommended for Arm MCUs because they provide better performance than the reference implementation.

5.2.1.2 PDM and DMAC configuration

The Pulse Density Modulation Interface (PDM-IF) supports external microphones that output pulse-density modulated signals. It filters and converts high-sampling-rate 1-bit PDM data into lower-sampling-rate 20-bit or 16-bit digital audio data.

The PDM interface filters 1-bit input data and converts it to 20-bit or 16-bit little-endian digital audio data. Sound detection can be enabled independently for each channel, with an optional moving average filter, and can also wake the MCU from software standby mode. Each channel supports programmable filters, including a sinc filter, a high-pass filter for DC bias suppression, a correction filter for sinc passband distortion, and a half-band decimation filter for aliasing reduction. The interface also supports flexible decimation ratios, selectable first- through fourth-order sinc filtering, APB-based DMA operation, and a 32-stage FIFO for received PDM data. A configurable callback can notify the application when the receive threshold is reached or when sound detection or error occurs.

To meet the model requirements, configure the PDM interface to capture audio at a 48 kHz sampling rate in 16-bit PCM format.

Configure the PDM callback to trigger whenever a complete audio frame has been captured.

For more details about the PDM module and its configuration, refer to the PDM documentation or the corresponding PDM example project.

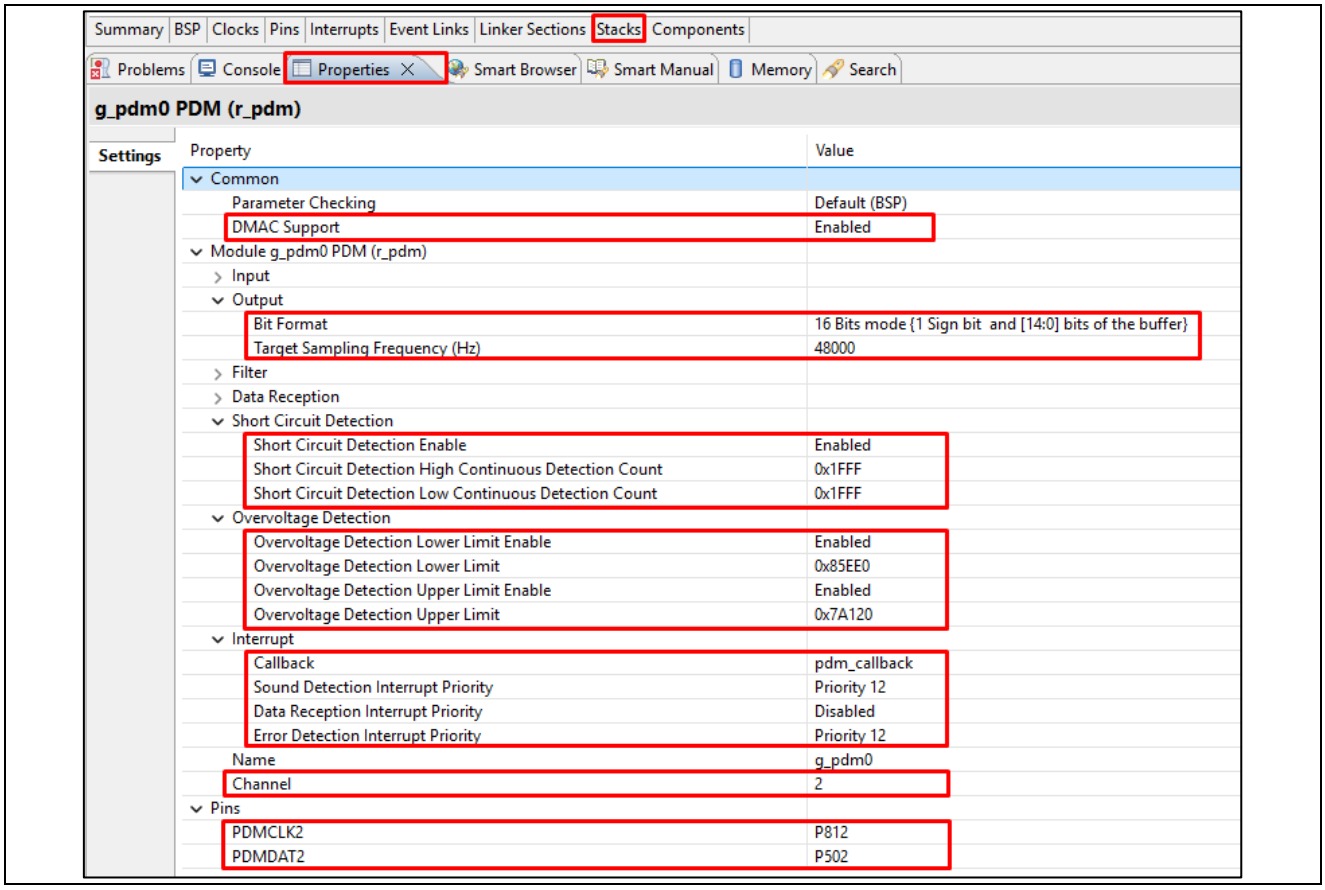


Figure 40. g_pdm0 PDM configuration

To reduce CPU load, the Direct Memory Access Controller (DMAC) is used to manage data transfers. DMAC allows data to be transferred between memory locations without CPU intervention.

To enable DMAC support in the PDM module:

- After adding the PDM stack, select **“Add DMAC Driver for Reception (Optional)”**
- Click **“New”** → **“Transfer (DMAC)”**
- Enable DMAC to offload data movement

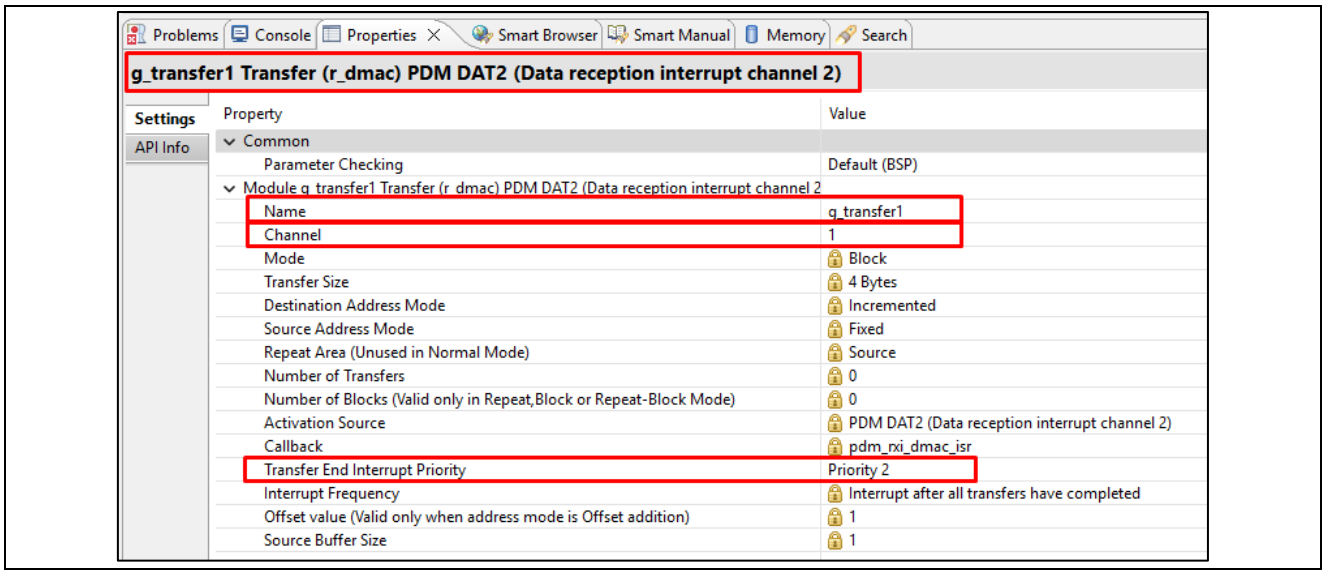


Figure 41. g_transfer1 Transfer configuration (for PDM data transfer)

When DMAC is enabled, the required configuration is applied automatically. Using DMAC offloads data transfers from the CPU, allowing the processor to focus on DSP tasks. This reduces CPU workload and improves the overall performance and efficiency of the audio processing system.

If DMAC support is not required, it can be disabled in the stack configuration. Open the **"Stack"** configuration, navigate to the PDM module, right-click the DMAC module below the PDM module, select **"Delete"**, and then click **"OK"**.

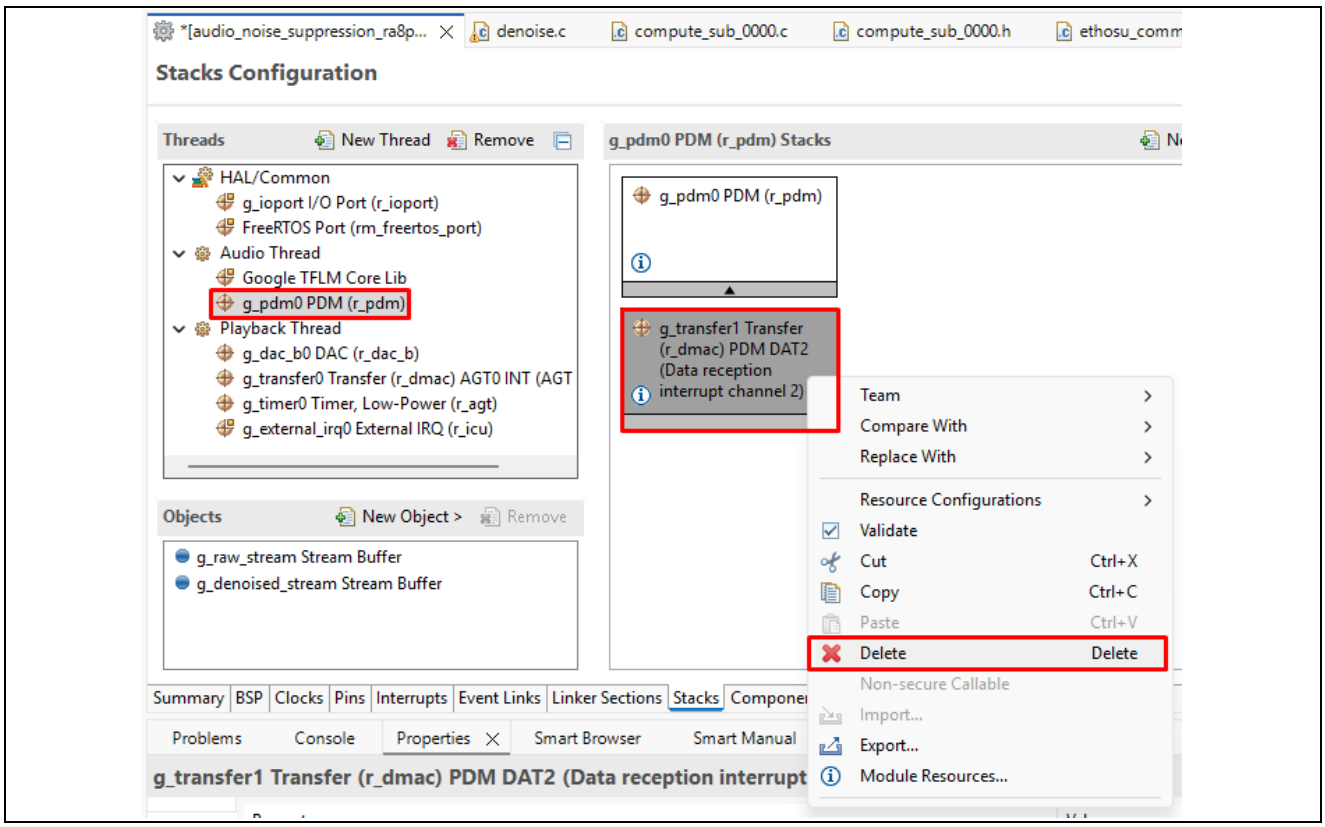


Figure 42. Remove DMAC module

Next, disable DMAC support in the PDM module because it is no longer required. Then enable the Data reception interrupt by setting its priority to 12, as shown in the Figure 43.

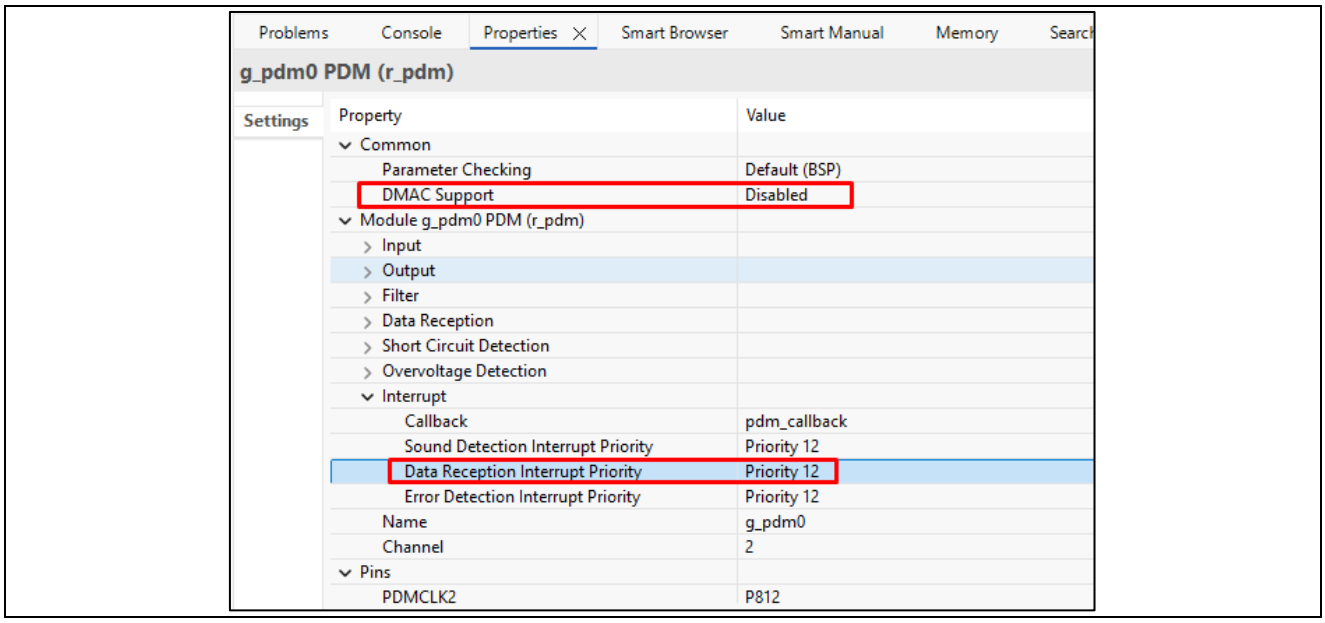


Figure 43. Disable DMAC support

5.2.2 Playback Thread

The playback pipeline consists of DAC, DMAC, AGT, and the PMOD AMP2 module. The denoised PCM data is transferred from the playback buffer to the DAC using DMAC, which offloads data movement from the CPU. The AGT timer generates a periodic trigger based on the target sampling frequency, ensuring that each audio sample is output at the correct rate. The DAC converts the digital PCM data into an analog signal, which is then amplified by the PMOD AMP2 module and sent to the speaker for audio playback.

5.2.2.1 DAC configuration

The DAC (Digital-to-Analog Converter) is responsible for converting digital PCM audio data into an analog signal. After the denoising process, the audio data is in PCM digital format and cannot be directly played by a speaker. The DAC converts this digital data into a continuous analog waveform, which is required for audio output.

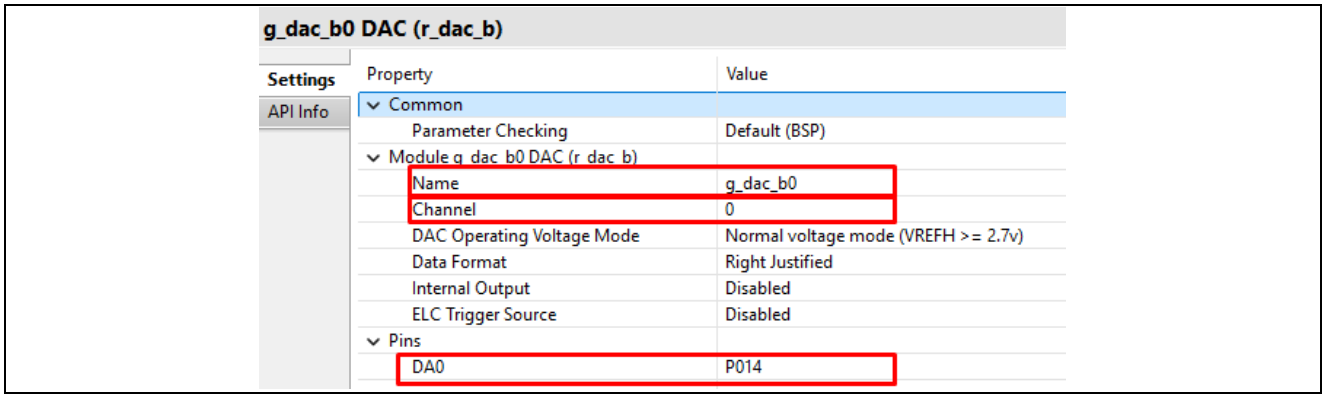


Figure 44. g_dac_b0 DAC configuration

Channel 0 is used so the DA0 is P014 pin.

5.2.2.2 DMAC and AGT configuration

DMAC is used to transfer audio data from memory to DAC without CPU intervention. Instead of the CPU manually writing each audio sample to the DAC, the DMAC automatically moves data in the background. This significantly reduces CPU load and ensures continuous data flow, which is critical for frame-based audio playback.

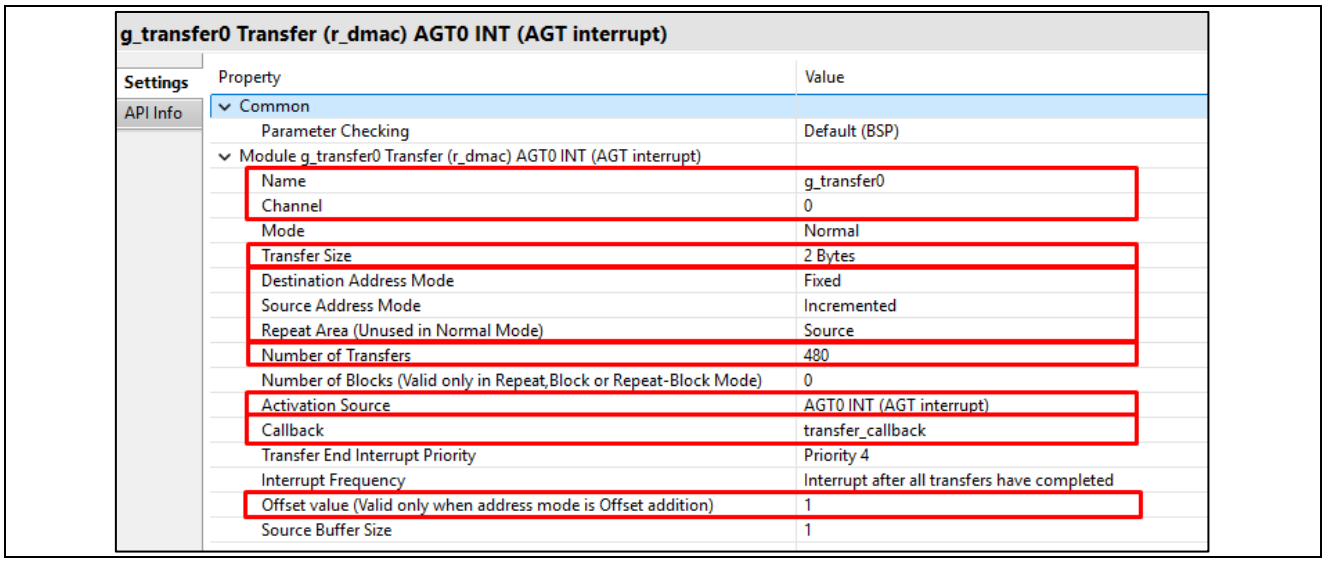


Figure 45. g_transfer0 with AGT interrupt configuration

The AGT (timer) is used to generate a periodic trigger signal that controls the DAC update rate during audio playback.

In this configuration, the AGT operates in periodic mode with a frequency of approximately **48.78 kHz**, meaning a trigger is generated every **~20.5 µs**. Each trigger event causes the DAC to output one audio sample, thereby defining the playback sampling rate.

Note that the actual playback frequency depends on the configured timer period. Therefore, it must be carefully adjusted to match the required audio sampling rate (e.g., 48 kHz) to avoid distortion or pitch variation.

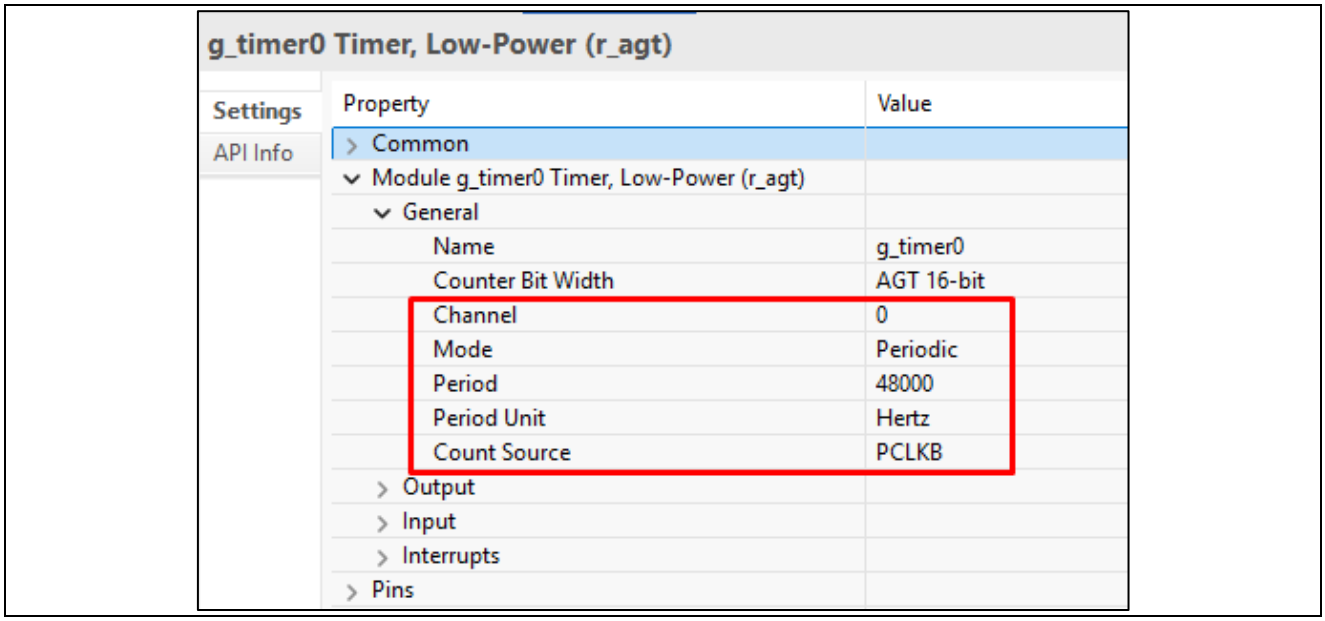


Figure 46. g_timer0 configuration

6. Application analysis

6.1 Frame timeline analysis

The application uses two main interrupt service routines (ISRs) to manage audio data flow.

- ISR 19 is the PDM capture callback. It runs frequently to support continuous audio acquisition from the PDM interface.
- ISR 20 is the DMAC transfer-complete callback. It is triggered after one full audio frame, 480 samples, has been transferred to memory, indicating that the frame is ready for processing.

To analyze ISR activity in SEGGER SystemView, select **“View” → “Event Filter” → “Show only ISRs”**, then select a time range that covers one ISR 19 period to observe the related system behavior.

After ISR 19 completes, which takes approximately **53**.1 µs, the Audio Thread is scheduled to process one complete audio frame. In the denoised path, the Audio Thread takes approximately 6.2 ms, including:

- Pre-processing: 4.63 ms
- Model inference: 87.7 µs
- Post-processing: 1.42 ms

These measurements show that DSP operations, mainly pre-processing and post-processing, account for most of the execution time, while AI inference contributes only a small portion.

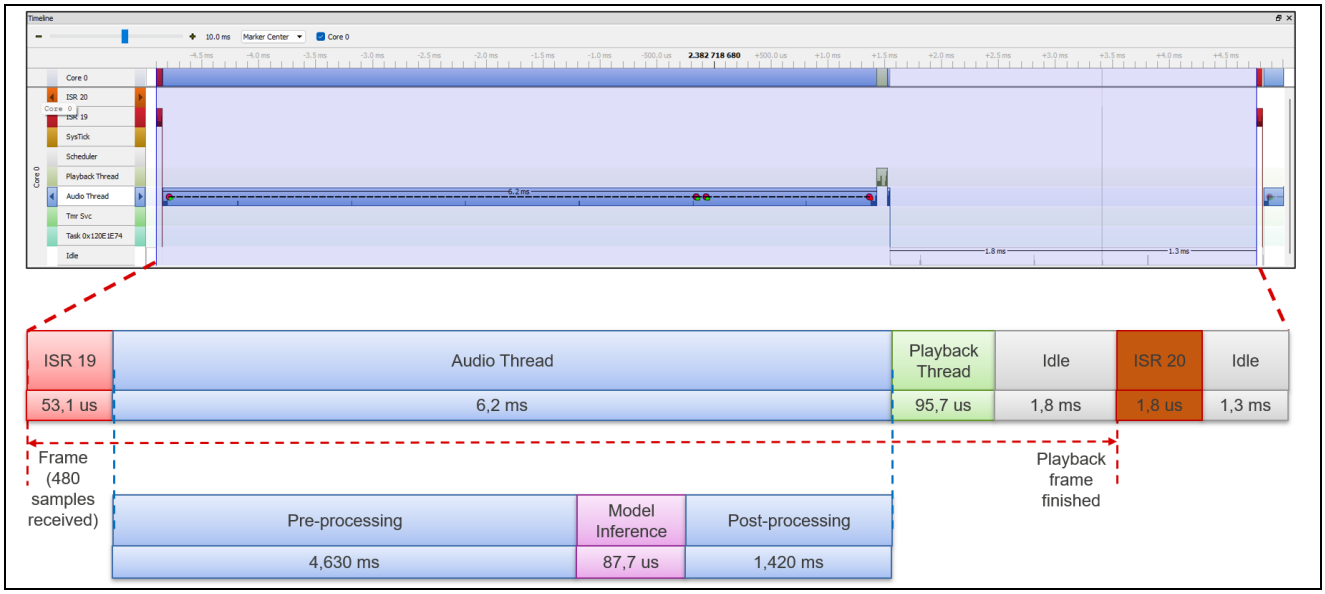


Figure 47. Denoised frame Configuration

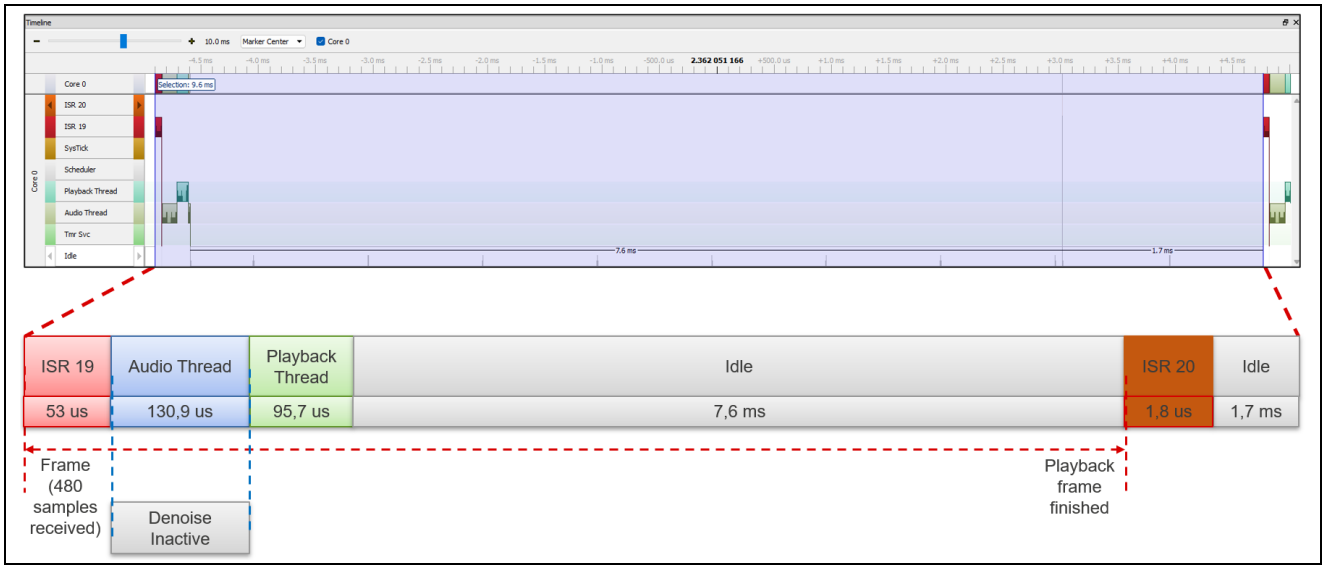


Figure 48. Raw frame Configuration

Once the Audio Thread completes, the Playback Thread executes for approximately 95.7 µs to output the processed frame, followed by a “Playback frame finished” event.

The system then enters an idle state (~1.8 ms), indicating available processing margin before the next frame cycle.

During this period, ISR 20 (PDM callback) is briefly triggered (~1.8 µs) to maintain continuous audio acquisition, followed by an additional idle duration of approximately 1.3 ms.

In contrast, for the noise suppression is disabled, the Audio Thread bypasses all DSP and AI processing and directly outputs the input frame. As a result, the execution time is significantly reduced, and idle time becomes the dominant portion of the timeline.

Overall, the total processing time of the denoised path remains well within the frame period, ensuring that real-time processing requirements are met while still maintaining sufficient timing margin.

6.2 CPU bandwidth analysis

To analyze CPU bandwidth, the Context view in the SystemView application can be utilized. The Context view provides a detailed runtime breakdown of all system components.

In the current system, the execution contexts can be grouped as follows:

- **Kernel/ISR group**: SysTick, Scheduler, Timer Service (Tmr Svc), ISR20, ISR19
- **Application group**: Playback Thread, Audio Thread
- **Idle**: Idle task

Each row in the Context view represents an execution context and includes comprehensive metrics such as:

- Total execution time
- Blocked time
- CPU usage
- Minimum, maximum, and average runtime

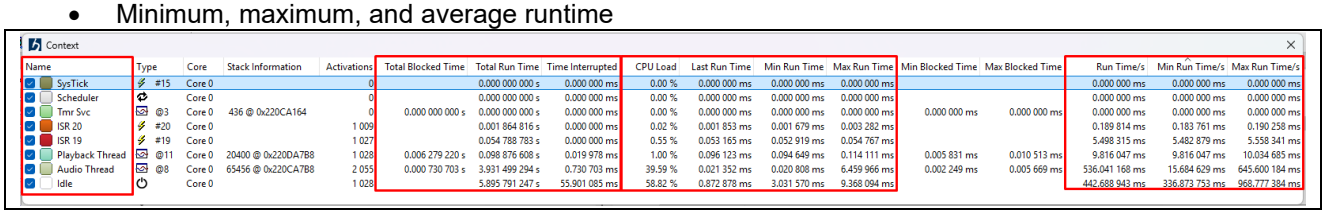


Figure 49. Context Analysis

The most important metric is the **CPU Load**. It can be observed that the Audio Thread consumes **39.59% CPU**, while the Playback Thread only consumes **1% CPU**. This indicates that Audio Thread requires significantly more processing resources. The reason is that the Audio Thread is responsible for pre-processing and post-processing operations, audio capture, and AI inference, which are computationally intensive. In contrast, the Playback Thread mainly manages DAC conversion and transfers data to the peripheral for playback using DMAC, resulting in a much lighter processing requirement.

The Idle task occupies **58.82%** CPU, indicating that there is still a considerable amount of available processing capacity. This leaves room for further application development or additional features, such as adding post-processing effects (e.g., equalization or gain control), supporting multi-channel audio, or integrating additional AI-based audio analytics.

The following section provides a summary of key components:

- **Audio Thread**: The most notable metric is the CPU load of **39.59%**, indicating that the thread runs very frequently and consumes a significant portion of CPU resources. The maximum runtime reaches **6.46 ms**, which is relatively high but still within the acceptable limit of a **10 ms processing window per frame**.
- **Playback Thread**: The CPU load is only **1%**, indicating a very lightweight task with no performance concerns.
- **Idle Task**: The Idle task accounts for 58.82% of CPU time, indicating that the system still has substantial processing headroom and is far from being overloaded. However, the CPU usage distribution is not balanced across tasks.

Other metrics such as blocked time, ISR CPU load, and ISR runtime are relatively small and do not significantly impact the system, indicating stable operation.

In summary, the system demonstrates these strengths:

- The CPU is not fully utilized
- Interrupt Service Routines (ISRs) are lightweight
- Playback operation is stable

However, there is one notable limitation: the Audio Thread is relatively heavy, consuming nearly **40% of CPU resources** for a single task.

7. References

RA8P1 Group User's Manual: Hardware

RA Flexible Software Package (FSP) | Renesas

High Performance with RA8 MCU using Arm® Cortex®-M85 core with Helium™

DSP Design Workflow with RA8 MCUs

r11qs0065ej0101-ruhmi-framework-qsg

Arm-Examples/ML-zoo

xiph/rnnoise

Website and Support

Visit the following URLs to learn about key elements of the RA family, download components and related documentation, and get support.

RA Product Information	www.renesas.com/ra
RA Product Support Forum	www.renesas.com/ra/forum
RA Flexible Software Package	www.renesas.com/FSP
Renesas Support	www.renesas.com/support

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Jul.09.26	-	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/.