

Renesas RA Family

DSP Design Workflow with RA8 MCUs

Introduction

This application note describes the use of an ARM® CMSIS-DSP based Audio Equalizer application project that is created with a MATLAB Filter Design workflow. First, the audio equalizer is designed with MATLAB Filter Design Toolboxes and validated in MATLAB by running the included scripts to process audio with the resulting filter implementation. Next, the CMSIS-DSP source code is generated with the MATLAB Embedded Coder tool and is then optimized to run on the Renesas Arm Cortex-M85-based MCUs with Digital Signal Processing (DSP) extension, and Floating Point Unit (FPU) support. This application note explains the steps to design, import, build, and execute the Audio Equalizer project that is included to accompany the application note.

Additionally, the application note concludes by describing how to improve filter performance with Renesas RA8 MCUs using Arm® Cortex-M85 core with Helium™.

The Audio Equalizer is implemented for floating-point filter processing with the CMSIS-DSP filtering function **Biquad Cascade IIR Direct Form II**. All information regarding Arm CMSIS-DSP can be found at the following link:

[Arm Software GitHub CMSIS 5 DSP](#)

The details on the Biquad Cascade IIR function can be found at the link:

[Biquad Cascade IIR Filters Using a Direct Form II Transposed Structure](#)

Required Resources

Hardware

- EK-RA8D1, Evaluation Kit for RA8D1 MCU Group ([renesas.com/ra/ek-ra8d1](https://www.renesas.com/ra/ek-ra8d1))
- SparkFun Electronics Audio Codec breakout board with WM8960
- Assorted jumper cables

Development Tools and Software

- The e² studio IDE v2024-10
- Renesas Flexible Software Package (FSP) v5.6.0
- LLVM Embedded Toolchain version 18.1.3
- Segger J-Link® USB driver v7.98g and Serial Terminal App Tera Term v4.99
- Optional - MATLAB Filter Design Tools
- Optional - Segger SystemView v3.58

Target Devices

This application note focuses on the Audio EQ design workflow for the RA8D1 MCU. However, it also applies to any Arm Cortex-M85-based Renesas MCU that supports Serial Sound Interface Enhanced (SSIE), such as the RA8M1, RA8E1, and RA8E2 MCUs.

Contents

1.	Application Project Requirements	4
1.1	Required Development Tools	4
1.2	Required Hardware	4
1.3	Required Software.....	4
2.	How to Build and Run the Audio Equalizer Application	4
2.1	Build the Project Source Code	5
2.2	Connect Audio Codec and Signal Connections	7
2.3	Setup Command Line Interface (CLI)	11
2.4	Download and Operate Project.....	11
3.	Audio Equalizer Design Workflow	13
3.1	MATLAB Software Requirements	14
3.2	Create Filter Coefficient and Processing Functions.....	16
3.3	Export Filter Functions with Embedded Coder Tool	18
3.4	Understanding the Generated Source Code.....	20
3.5	Modify Source Code for Best Performance on RA8 MCUs	24
3.6	DSP Process Latency Measurement	25
3.6.1	How to Enable Latency Measurement	25
3.6.2	Adding Support Code to Measure Latency	25
3.7	Changing Segger SystemView Support.....	27
3.7.1	Integration of Segger SystemView into Example Project.....	28
4.	Audio Equalizer Software Architecture.....	29
4.1	Audio Circular Buffers	29
4.2	Audio Codec Thread	31
4.2.1	I2S Stack Settings	31
4.2.2	I2S Audio Clock.....	32
4.2.3	I2S Callback	33
4.2.4	Codec Initialization	35
4.2.5	I2S Initialization	37
4.2.6	Codec Thread Processing.....	37
4.3	DSP Processing Thread.....	38
4.3.1	DSP Process Initialization	39
4.3.2	DSP Processing	39
4.3.3	Audio EQ Filter Implementation	41
4.3.4	CMSIS DSP Stack.....	43
4.4	Command Line Interface Thread	43
4.5	LED Blinky Thread	45
4.6	Using Segger SystemView.....	45
5.	Optimizing Audio EQ Performance with Arm® Helium™ on RA8 MCU	48
5.1	Introduction to Arm® Helium™	48
5.2	Arm® Helium™ Support in Renesas FSP and LLVM Toolchain	48
5.3	Improve DSP Performance	49
5.3.1	Improve Performance with Data Cache	49
5.3.2	Improve Performance Using DTCM and ITCM	50
5.3.3	Improve Performance Using ARM CMSIS DSP Optimizations.....	52

6. Next Steps 54

7. References 54

8. Website and Support 54

Revision History 55

1. Application Project Requirements

The following items are required to build and run the Audio Equalizer.

1.1 Required Development Tools

The following tools are required for building the application project. The tools are available for downloading from Renesas.com:

- e² studio version 24.10.0 with FSP version 5.6.0 and LLVM for ARM version 18.1.3
<https://www.renesas.com/en/software-tool/flexible-software-package-fsp>
or by internet web search keyword: RA FSP Download

1.2 Required Hardware

The following items are required to operate the application project:

- EK-RA8D1
<https://www.renesas.com/en/products/microcontrollers-microprocessors/ra-cortex-m-mcus/ek-ra8d1-evaluation-kit-ra8d1-mcu-group>
or by internet web search keyword: EK-RA8D1
- 1x micro USB cable for programming, debugging, and status display on the serial terminal (for example, Tera Term)

The following items are required for the operation of the application project:

- Wolfson WM8960 Codec mounted on SparkFun Electronics Audio Codec Breakout WM8960 (SparkFun part number BOB-21250)
- Stereo Audio source devices, for example, Compact Disc Players and Stereo Audio output devices such as Powered Audio Speakers, or Headphones
- 10 x jumper wires to connect SparkFun Audio Codec Breakout to the EK-RA8D1 digital connections
- 3 x jumper wires to connect audio source device signals, for example, a Compact Disc Player, to Audio Codec audio inputs
- 3 x jumper wires to connect the Audio Codec audio outputs to Powered Speakers or Headphones

1.3 Required Software

The following software item is required for the application project operation

- Tera Term v4.99 for Command Line Interface [Tera Term Download](#)

The following software items are optional for the audio filter design and for profiling the operation

- MATLAB with Audio Filter Design Toolboxes – See [Section 3. Audio Equalizer Design Workflow](#), [Table 2. MATLAB Toolboxes and Support Add-Ons](#) for more details
- Segger SystemView v3.58 Application

2. How to Build and Run the Audio Equalizer Application

After importing the example project, you can build, download, and run the project. See [Figure 1 Build Audio Equalizer Project](#), which shows the Audio Equalizer project.

Details on how to import example projects can be found in the [Renesas e² studio 2023 – 10 or Higher Quick Start Guide](#).

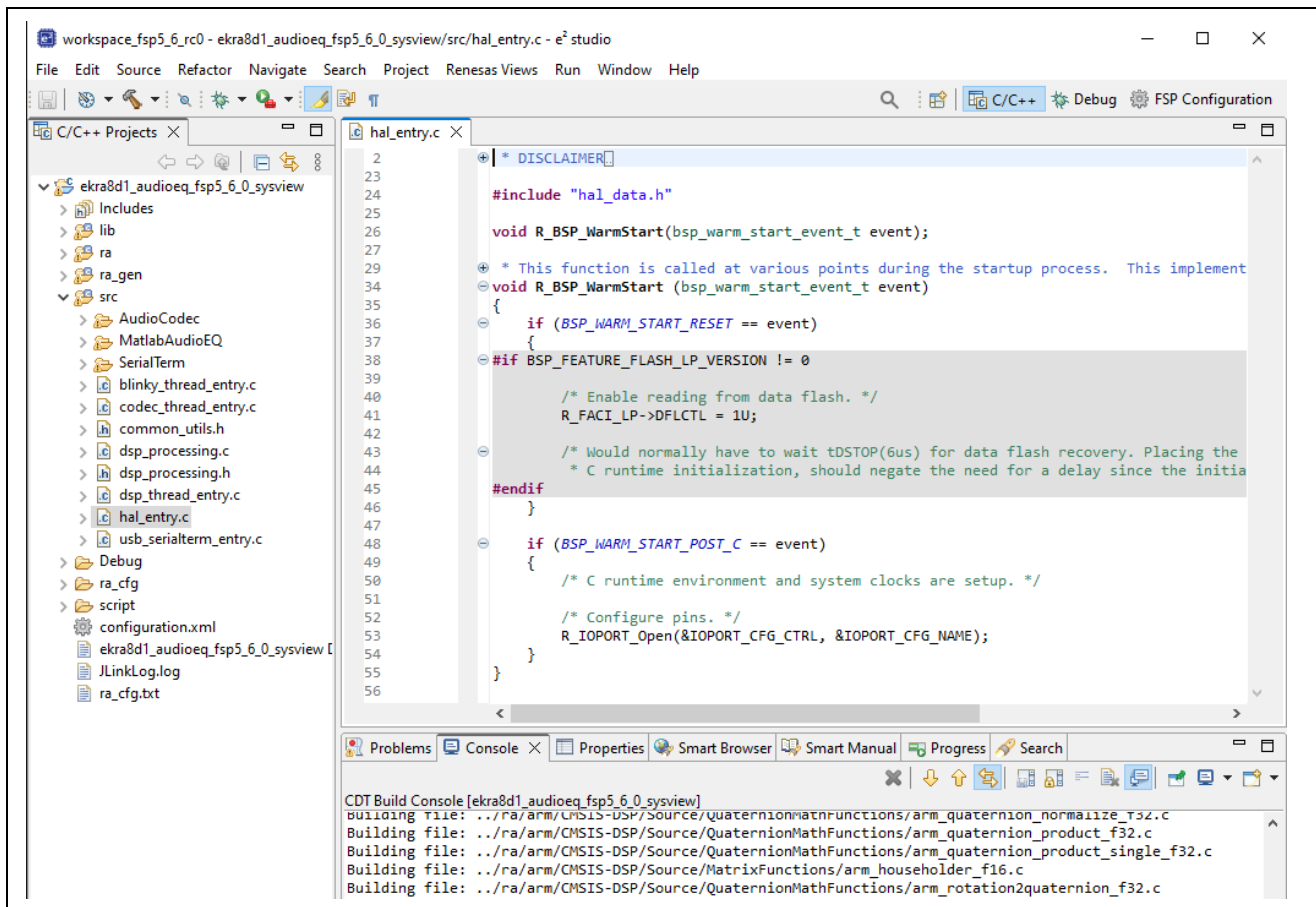



Figure 1. Build Audio Equalizer Project

2.1 Build the Project Source Code

Follow the steps to import and build the project:

- Place the project zip folder into a clean workspace directory and unzip the project there.
- Open e2 studio in your workspace directory.
- Navigate to **File -> Import**, which brings an import dialog. In this dialog, select the wizard titled “Existing Projects into Workspace” to finish importing the project (see [Figure 1. Build Audio Equalizer Project](#)).
- Build the firmware using the hammer icon on the top toolbar 

The compiler uses the -O3 (optimize most) setting and the LLVM Embedded Toolchain for Arm v18.1.3 in addition to the Arm DSP Library Source, which has optimizations for the Helium extension (see [Figure 2. Project Toolchain](#) and [Figure 3. Project Summary](#)).

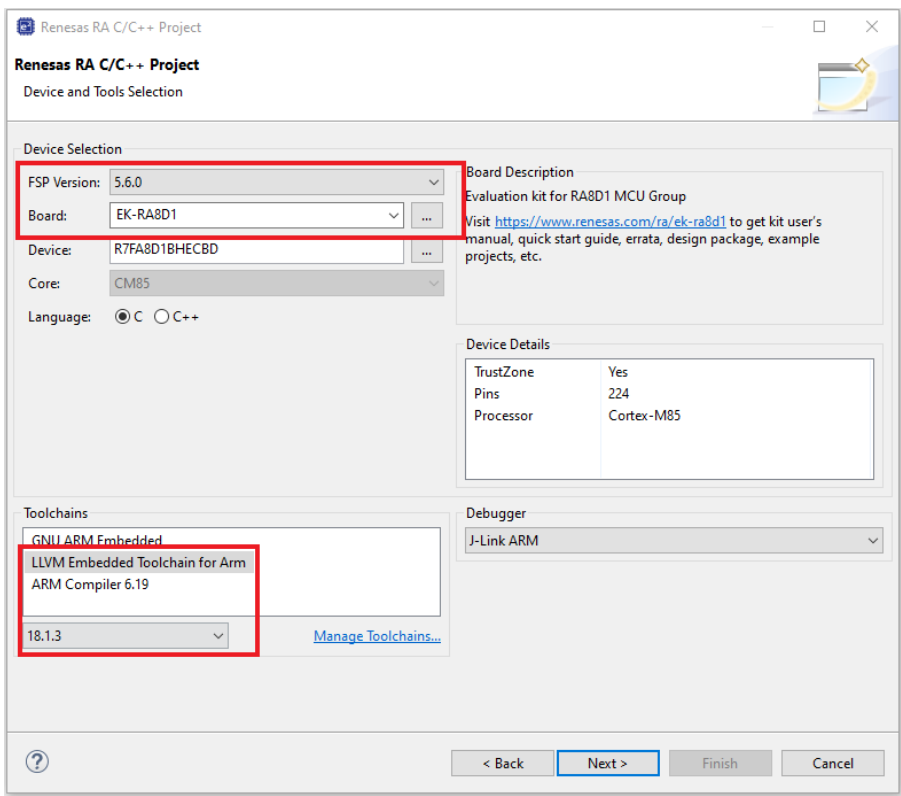


Figure 2. Project Toolchain

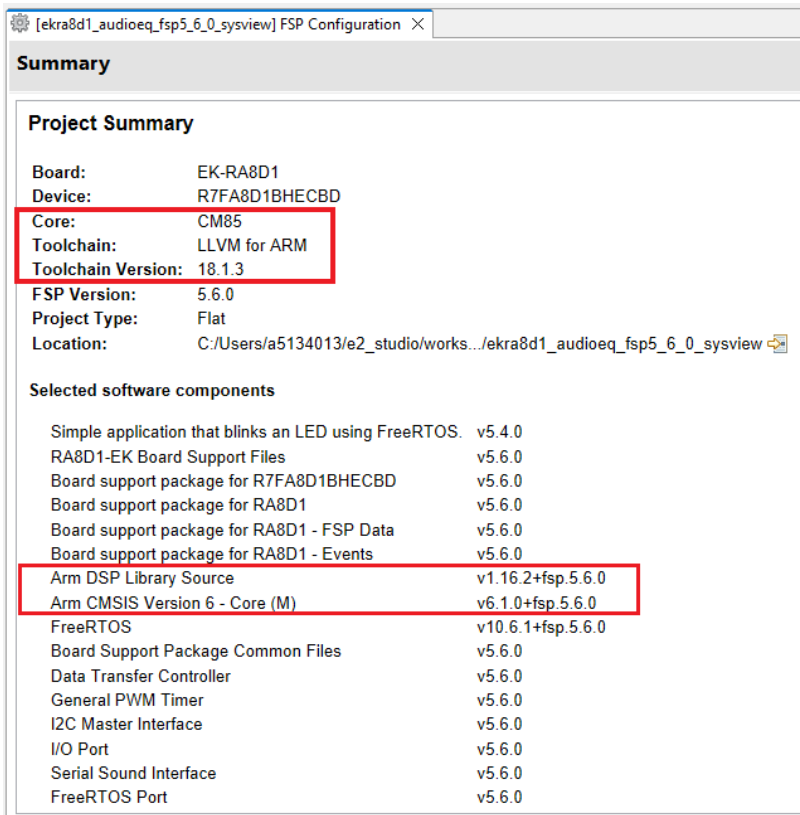


Figure 3. Project Summary

2.2 Connect Audio Codec and Signal Connections

Use the jumper wires to connect the SparkFun Audio Codec board to the EK-RA8D1 according to the connections in [Table 1. Audio Codec to EK-RA8D1 Connections](#).

Table 1. Audio Codec to EK-RA8D1 Connections

Audio Codec	EK-RA8D1	Description
SCL	P512 SCL1	I2C Clock
SDA	P511 SDA1	I2C Data
ADCDAT	P406 SSIRXD0	I2S ADC data to MCU
DACDAT	P405 SSITXD0	I2S DAC data from MCU
DACLRC	P404 SSIWS0	I2S Word Select from MCU
ADCLRC → connect to DACLRC	-	I2S Word Select from MCU
BCLK	P403 SSISCK0	I2S Bit Clock from MCU
VIN	Power 5V	DC Voltage for Codec Analog
3V3	3V3	DC Voltage for Codec Digital
GND	GND	0V reference for Codec
LINPUT1	-	Audio input Left
RINPUT1	-	Audio input Right
Audio input GND to Codec GND	-	GND for audio inputs
HPL	-	Audio output Left
HPR	-	Audio output Right
Codec GND to Audio output GND	-	GND for audio outputs

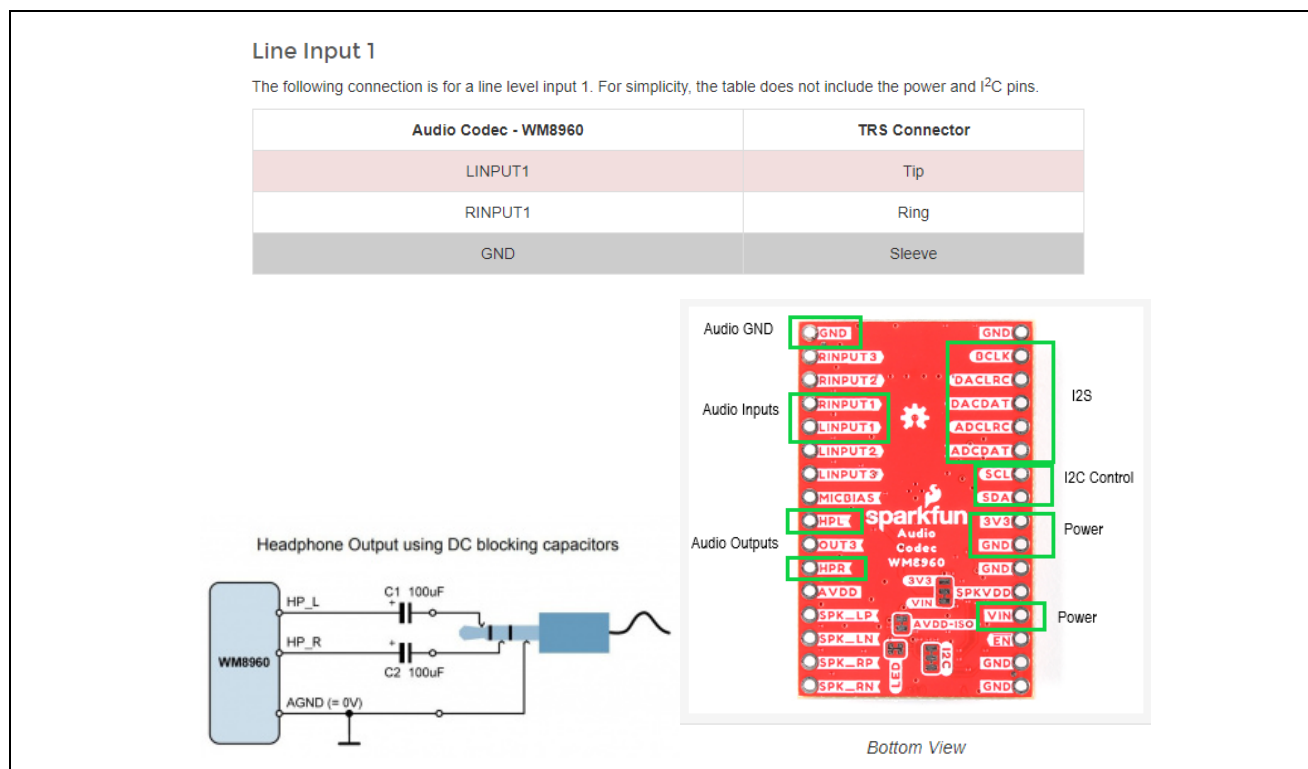


Figure 4. Audio Codec Connections

Use Table 1 to connect the Codec Digital and Power Connections to the EK-RA8D1 first.

In [Figure 4. Audio Codec Connections](#), the SparkFun Codec Board shows the I2S, I2C Control, and Power connections. Use the jumper wires to connect from the Audio Codec board to the EK-RA8D1 Connection Headers, as shown in [Figure 5. EK-RA8D1 Connection Headers](#). For the best signal integrity, solder the wires to the SparkFun board first and then make the connections to the EK-RA8D1.

Next, connect an audio signal source, such as a Compact Disc player, to the SparkFun Codec Audio Inputs LINPUT1, RINPUT1, and GND. Connect the Audio Outputs HPL, HPR, and GND from the SparkFun Audio Codec to the amplified speakers or headphones. For the best signal integrity, solder the connection wires to the SparkFun board.

For additional information, refer to the connections information in [Figure 4. Audio Codec Connections](#), [Figure 5. EK-RA8D1 Connection Headers](#) and the EK-RA8D1 User Manual [EK-RA8D1 v1 – User Manual](#).

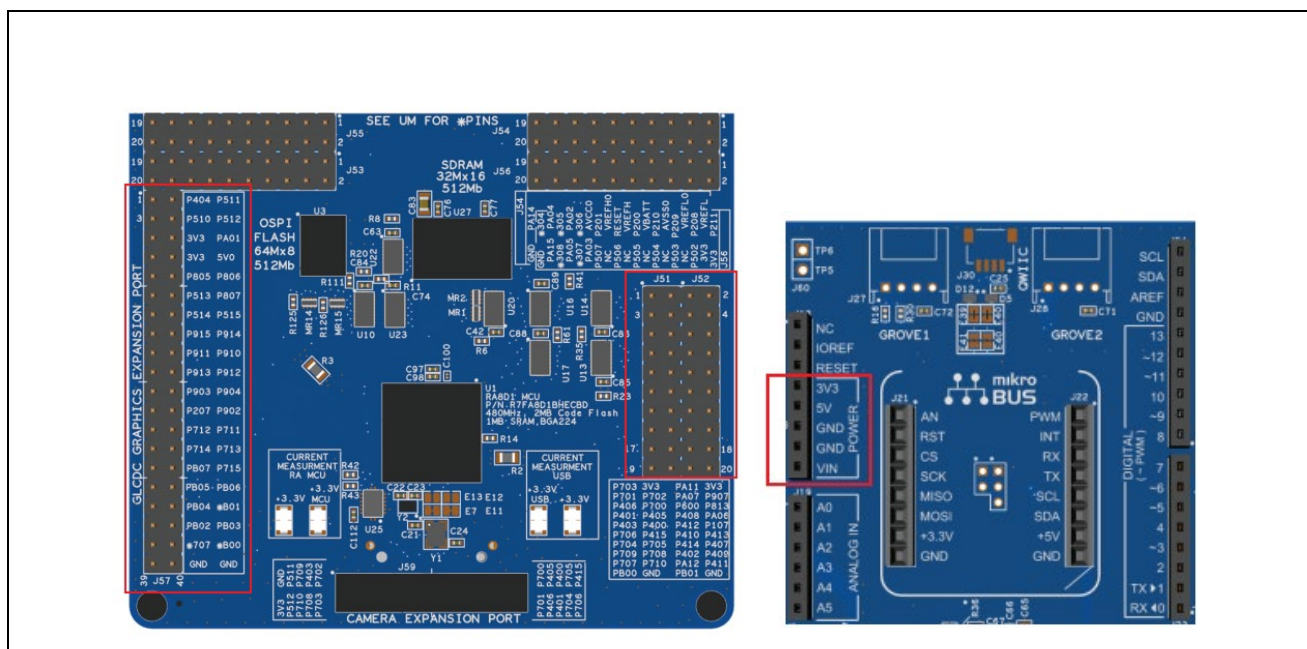


Figure 5. EK-RA8D1 Connection Headers

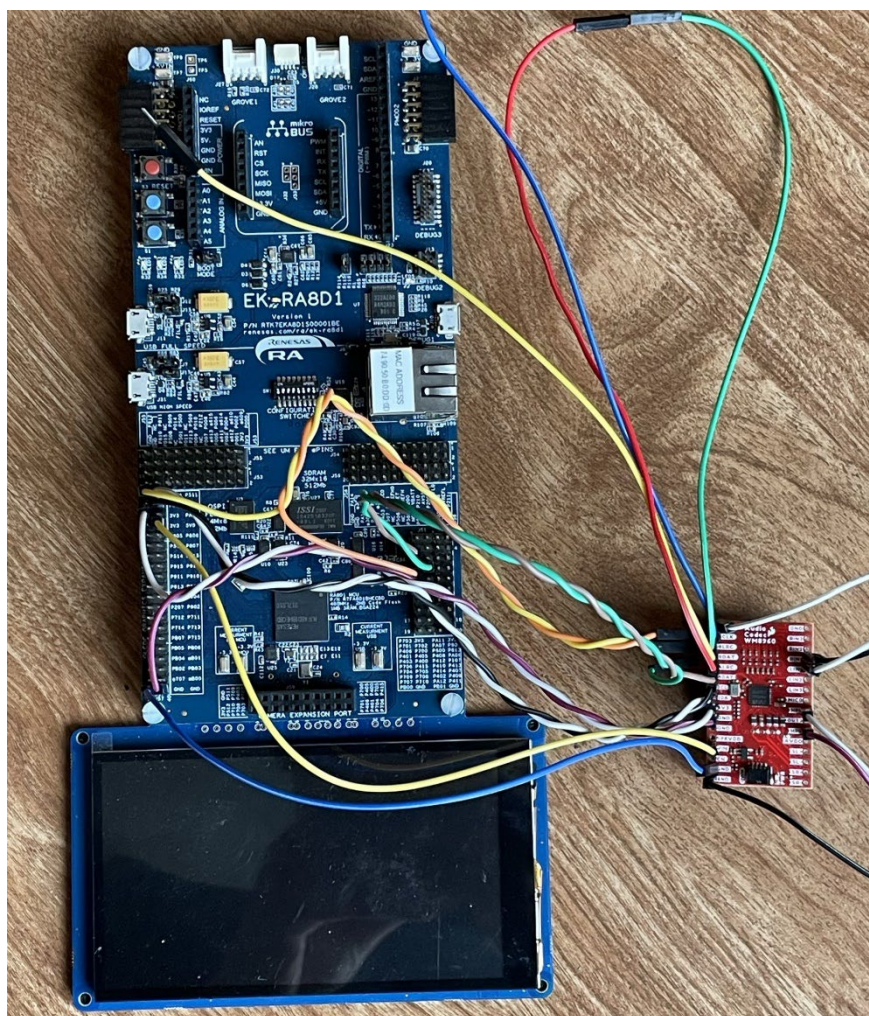


Figure 6. EK-RA8D1 and Audio Codec Wiring Example

See [Figure 6. EK-RA8D1 and Audio Codec Wiring Example](#) for an example of how to wire the Audio Codec to the EK-RA8D1.

Twist the analog audio input wires and audio output wires into pairs to minimize conducted noise due to proximity to the digital connections between the Audio Codec and EK-RA8D1 board.

Where possible, twist the wire pairs for signal wires carrying I2S ADC and DAC digital data/clock connections. It was determined that digital noise conduction was minimized for the I2S ADCDAT and DACLRC by tapping the twisted wire pair close to the board near the camera expansion port.

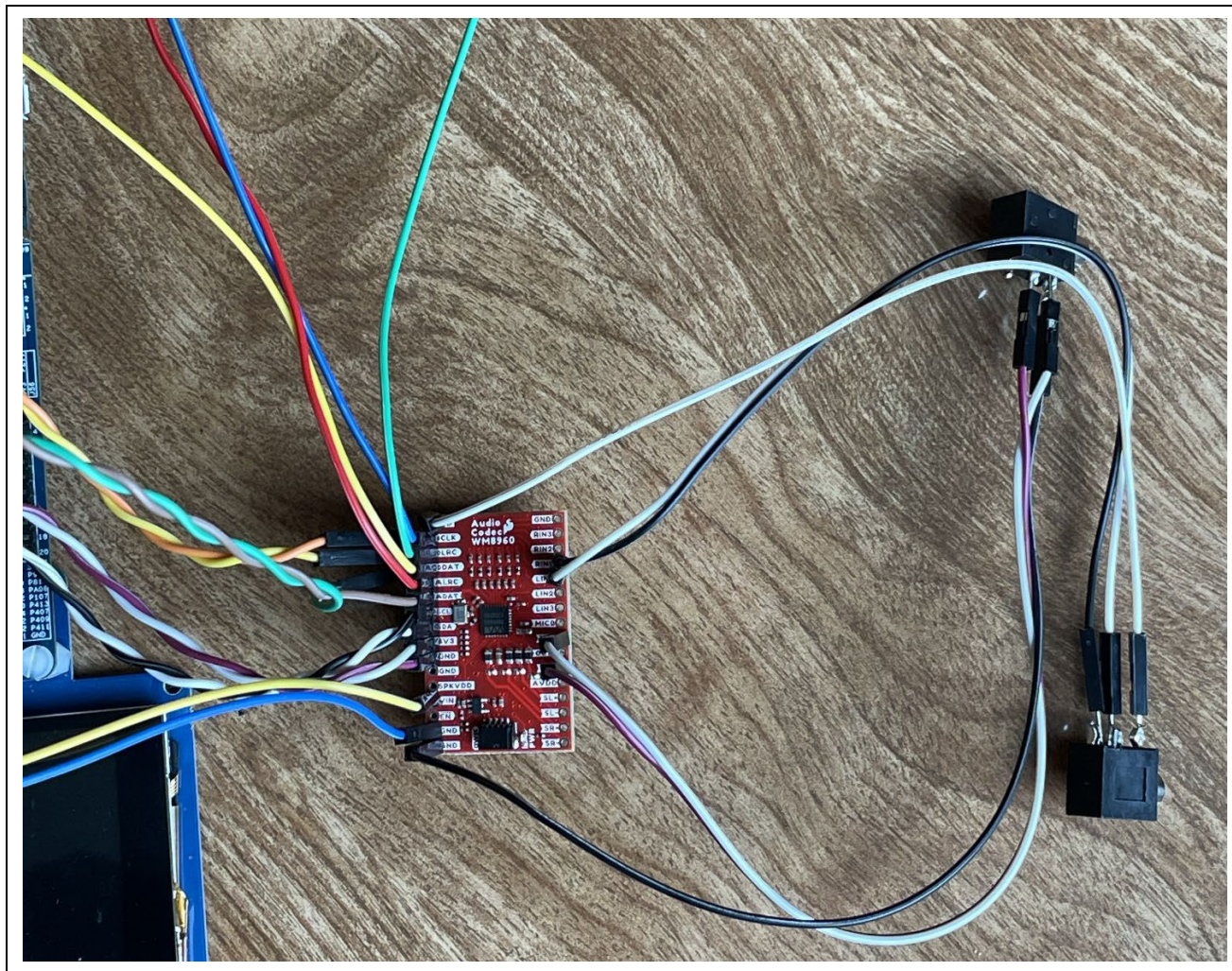


Figure 7. Audio Codec Wiring

See [Figure 7. Audio Codec Wiring](#) is an example of how the SparkFun Audio Codec can be connected. Notice the two female 3.5mm stereo audio jacks that are used to connect the audio source (LINPUT1, RINPUT1, and GND) and audio outputs (HPL, HPR, and GND) to the Audio Codec board.

After the Audio Equalizer (EQ) project is downloaded and running, be careful to set the audio signal source with a low-level (amplitude), feeding the Audio Codec. Overdriving the Audio Codec results in audible distortion at the Codec audio output. Slowly raise the audio signal source level until the output signal can be heard clearly but without distortion.

2.3 Setup Command Line Interface (CLI)

The Audio Equalizer command line interface with menu, operation mode, and processing latency are sent to the serial terminal emulator over UART. Connect to Tera Term using the following steps.

Launch Tera Term from the Windows Start Menu. As seen in [Figure 8. Launch Serial Terminal Application](#), open a new connection by selecting **Serial** option and click on **Port** to choose “COMxx: JLink CDC UART Port (COMxx)” and then click **OK**. The Com port number is different from what is shown. This depends on the number of serial devices connected to your PC.

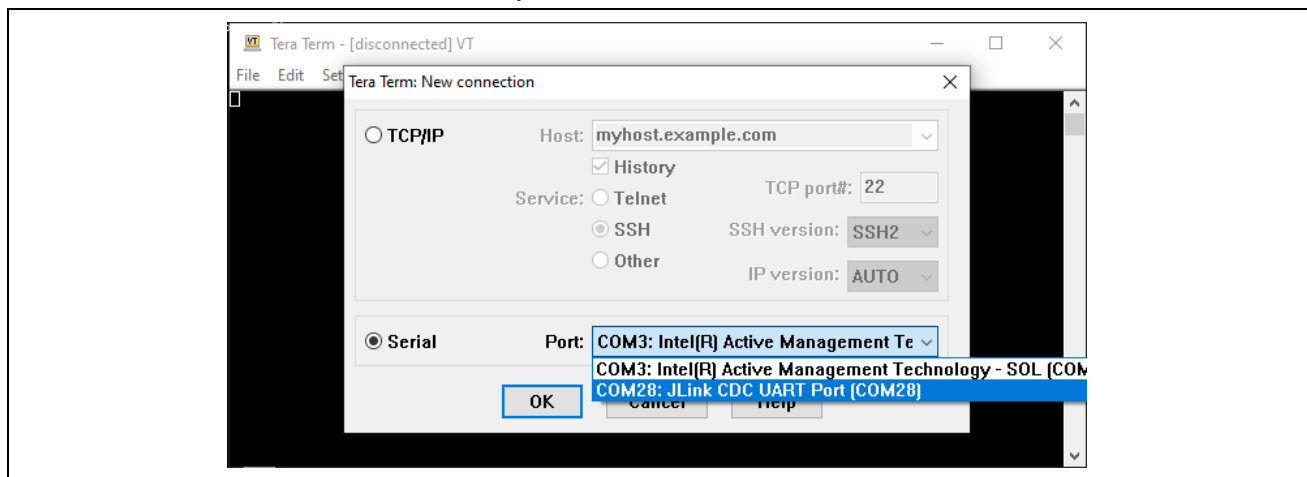


Figure 8. Launch Serial Terminal Application

Configure Tera Term by selecting **Setup -> Serial port**. Set **Speed** to **115200**. All other settings should match those as shown in Figure 9. Then click **New setting**.

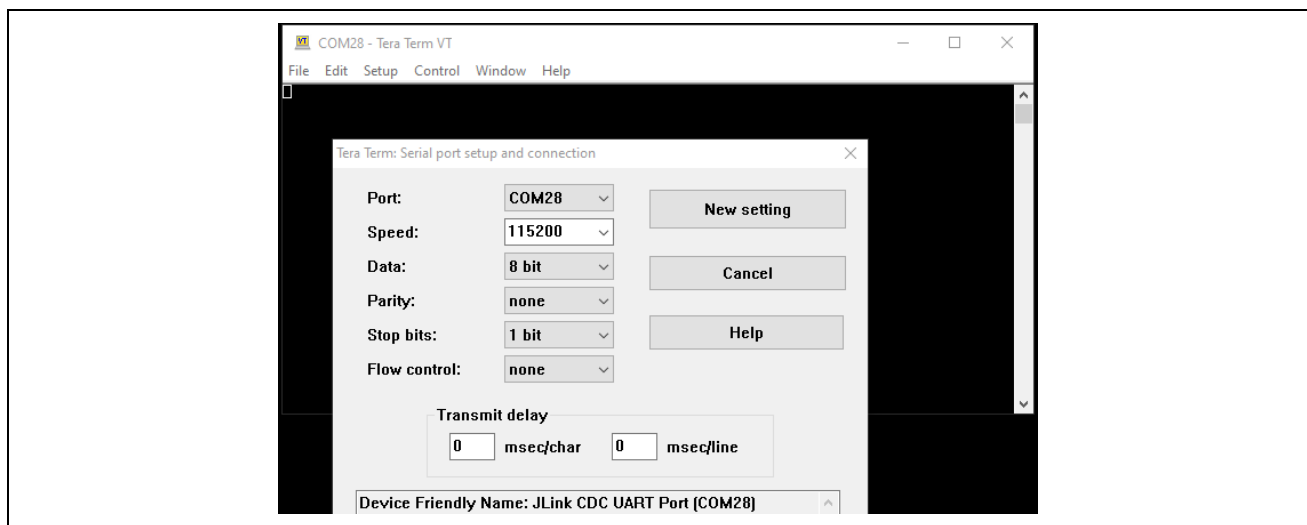




Figure 9. Tera Term Settings

Proceed with the steps to download and start the Audio EQ project in [Section 2.4 Download and Operate Project](#). When the Audio EQ project is started, the Command Line Interface menu is displayed on Tera Term. There are four menu options to select with the Audio EQ project. See [Figure 10. Audio EQ Command Line Interface](#) for more information on working with the Audio EQ application.

2.4 Download and Operate Project

Once the build process is finished, then run the project by clicking the Debug icon  to download to the target board and click Run  .

The Audio EQ starts with DSP processing active. [Figure 10 Audio EQ Command Line Interface](#) shows the CLI menu.

The Audio EQ starts with DSP Processing active, and you will hear several seconds of silence before the audio output plays once the audio input source is started. This is due to the circular buffers in the system.

To hear the unprocessed audio input, press 1 to bypass DSP Processing.

After listening to the audio for a few seconds, press 2 to Run DSP Processing.

You will hear a difference in the frequency response with the processed audio signal. Compared to the unprocessed input signal there will be more Bass (Low) and Treble (High) frequency content present.

```

COM28 - Tera Term VT
File Edit Setup Control Window Help

*****
* Renesas App Project for the EK-RA8D1 CMSIS DSP Module *
* App Project Version 1.0 *
* Flex Software Pack Version 5.6.0 *
*****

Refer to the Application Note for more details on the App Project
and FSP User's Manual for more information about the EK-RA8D1 CMSIS DSP

This Project demonstrates the functionality of an Audio Equalizer
developed in Matlab. The Matlab codegen exported source code is
implemented on the EK-RA8D1 with CMSIS DSP

Audio EQ Menu Options:
1. Bypass DSP Processing
2. Run DSP Processing
3. Show CPU Workload for Codec RX / TX 32 Frames L/R Audio
4. Show Latency for DSP Process
  
```

Figure 10. Audio EQ Command Line Interface

The following list details each of the CLI menu options and how it affects the application:

1. Bypass DSP Processing

Used to stop DSP processing. Audio input received is passed (with no processing) to Audio output. Use this menu option to switch between DSP processing mode and Bypass mode to hear the result of the Audio EQ processing. Also, Bypass mode can be used to set the audio input signal. Set the Bypass DSP mode and raise the level of the audio input signal so that no distortion is present while monitoring the audio output. If you hear audio distortion in the output signal in Bypass or Run DSP mode, then lower the input signal level.

2. Run DSP Processing

Activates the DSP Processing, which is the Audio Equalizer (5 bands with frequency centers at [140 Hz, 1000 Hz, 2400 Hz, 4800 Hz, 10000 Hz]. Gains values in Decibels (dB) are set for each of the frequency centers [6.0, 0.0, 1.0, 1.0, 6.0].

3. Show CPU Workload for Codec RX / TX 32 Frames L/R Audio

Shows the amount of time required to receive 32 frames of interleaved channel data from the Audio Codec ADC and to transmit 32 frames of interleaved left/right audio data to the Audio Codec DAC over I2S interface. I2S Sample rate is 44.1 kHz with 16-bit fixed point (integer) data. One frame is one sample of data for the left and right audio channels.

4. Show Latency for DSP Process

Shows the amount of time required to process 32 samples of left/right audio data. The fixed data is copied from the input circular buffer and de-interleaved, converted to floating point and processed, and then converted to fixed point interleaved data before storing it in the output circular buffer.

An example frequency response is shown in [Figure 11. Audio EQ Frequency Response](#) for the Audio EQ in Bypass DSP Processing mode (yellow graph) and Run DSP Processing mode (blue graph). The graphs show the frequency vs. amplitude for a Rock music input signal that contains many instruments and vocals.

For the yellow graph, notice that the bass frequencies (20 Hz to 1000 Hz) are lower in amplitude (dBm) than the DSP-processed signal (blue graph), and high frequencies (10 kHz to 20 kHz) are lower in amplitude than the DSP-processed signal. In Run DSP Processing mode, the filter response is set to boost frequencies for the band centered at 140 Hz by 6 dB and the band 10 kHz by 6 dB. With the DSP Processing mode active, the higher amplitude response of the frequency bands can be observed, as indicated by the blue waveform.

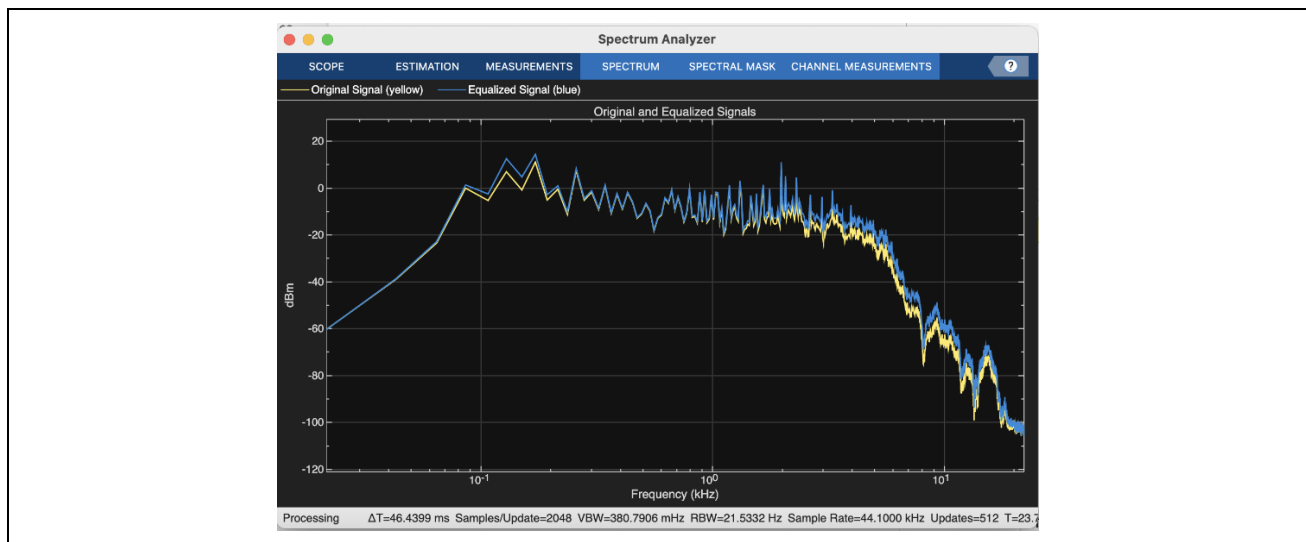


Figure 11. Audio EQ Frequency Response

3. Audio Equalizer Design Workflow

The Arm CMSIS-DSP examples [Arm Software GitHub CMSIS_5 DSP](#) provide an introduction and reference on how to implement fundamental processing with CMSIS-DSP functions. However, in the field of Digital Signal Processing, de facto standard design tools such as MathWorks MATLAB are often used to iteratively design and validate filters before implementing and optimizing the source code for an embedded system target.

This audio equalizer (EQ) application note goes into detail on using MATLAB as an audio filter design workflow, generating the CMSIS-DSP supported source code for Cortex-M processors, and optimizing the implementation on Renesas RA8 MCUs for the lowest processing latency.

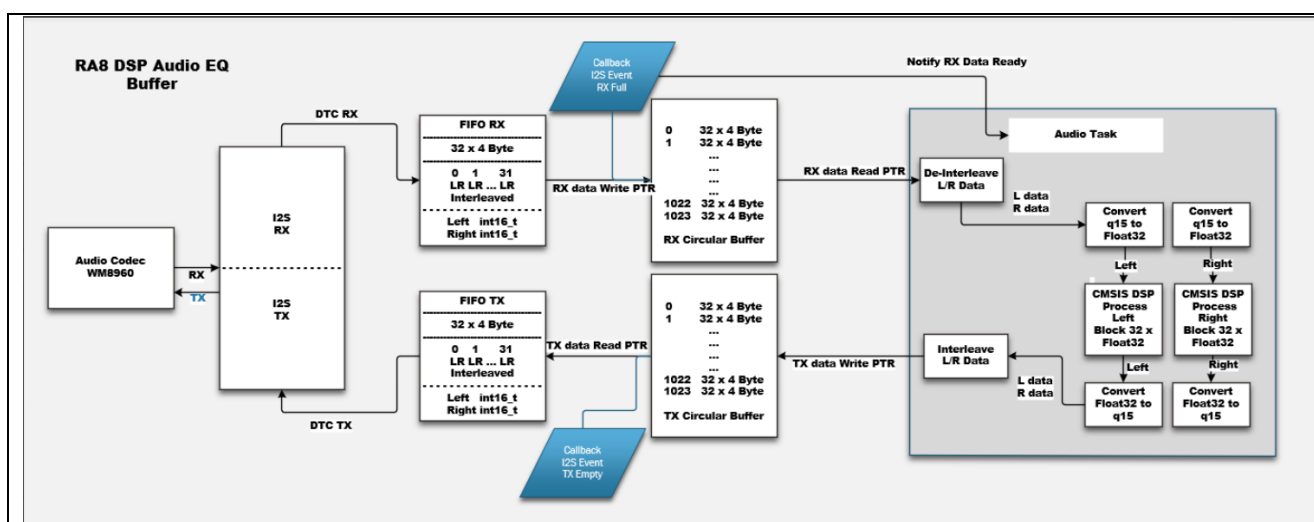


Figure 12. Audio EQ Block Diagram

Figure 12. Audio EQ Block Diagram shows the system block diagram of the audio EQ. The basic operation steps are:

- Input audio data is received from the Audio Codec by FIFO RX and stored in the RX Circular Buffer

- RX Circular Buffer fixed-point data is de-interleaved into left and right channel data and converted to floating point data.
- Process L & R channel data with the Audio EQ CMSIS DSP floating point functions on a block of 32 samples.
- Processed Left and Right channel data is converted back to fixed-point data and interleaved.
- Interleaved data is stored in the TX Circular Buffer, loaded into FIFO TX, and sent to the Audio Codec as output audio for the listener.

3.1 MATLAB Software Requirements

For this project, MATLAB version R2023b [MATLAB \(mathworks.com\)](https://www.mathworks.com) was used along with the Audio Toolbox [Audio Toolbox](#) and DSP System Toolbox [DSP System Toolbox](#). In addition, the Signal Processing Toolbox [Signal Processing Toolbox](#) was installed since it is a dependency for the DSP System Toolbox and Audio Toolbox.

In order to export the audio filter designed in MATLAB and generate source code targeted for Arm Cortex-M processors, it is necessary to install MATLAB Coder [MATLAB Coder](#), Embedded Coder [Embedded Coder](#), and the support package [Embedded Coder Support Package for ARM Cortex-M Processors](#).

For instructions on how to install the toolboxes and support add-ons, consult the installation instructions at the links provided above. Trial versions for evaluation of the toolboxes and add-ons are available by contacting MATLAB customer support. If you have any installation issues or questions about the software components, contact MATLAB customer support at [Support - MATLAB](#).

See [Table 2. MATLAB Toolboxes and Support Add-Ons](#) for an explanation of the MATLAB toolboxes and support add-ons used in the project.

Table 2. MATLAB Toolboxes and Support Add-Ons

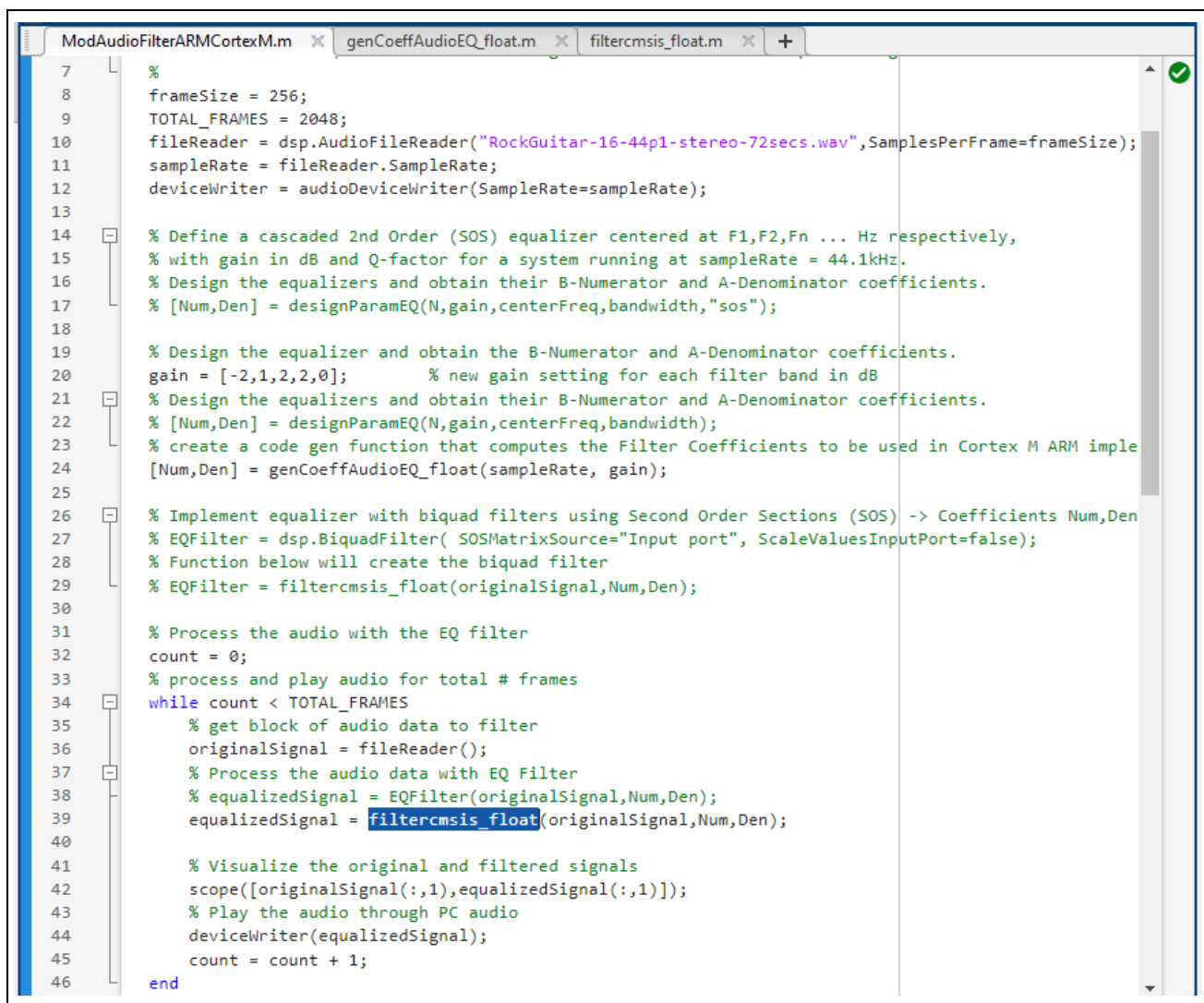
Software Component	Component Type	Description
MATLAB R2023b	Application	Main application
Audio Toolbox v23.2	Toolbox (Add-On)	Used to add audio file streaming support to test filter design on PC in real-time
DSP System Toolbox v23.2	Toolbox (Add-On)	Required by Audio Toolbox and used for Audio Filter Design
Signal Processing Toolbox v23.2	Toolbox (Add-On)	Required by DSP System Toolbox and Audio Toolbox
MATLAB Coder v23.2	Add-On	Generates C/C++ code from MATLAB code
Embedded Coder v23.2	Add-On	Extends MATLAB coder to generate C/C++ code optimized for embedded systems
Embedded Coder Support Package for ARM Cortex-M Processors v23.2.1	Add-On	Extends Embedded Coder to generate code optimized for Cortex-M processors

For this project, the DSP System Toolbox was chosen to design the Audio EQ filter because it provides useful System objects that include embedded coder support for ARM Cortex-M processors. Specifically, the Audio EQ was implemented as a Band Pass Filter (BPF) using cascaded Second Order Sections (SOS) with the `dsp.BiquadFilter` System object [Biquad Filter](#).

A Biquad is a type of linear filter that implements a transfer function that is the ratio of two quadratic functions (the numerator and denominator). What is unique about a Biquad filter is the ability of the user parameter settings (the coefficient design) to control the response characteristics of the filter. The parameters are Gain, Band Center Frequency, and Bandwidth (Selectivity/Quality-Q). The user sets the desired parameters, and the numerator and denominator coefficients of the transfer function are calculated to design the filter.

It is beyond the scope of this application note to cover the filter design specifics and details of the MATLAB filter toolboxes. Refer to MATLAB documentation for more information. For other filter System objects that support Cortex-M code generation, see the documentation for [DSP System Toolbox](#).

If you are developing a filter with a different System object or Toolbox, be sure to consult the MATLAB documentation to ensure that the filter provides the required embedded coder support for ARM Cortex-M cores. Not all MATLAB signal processing and filter design Toolboxes provide embedded coder support for ARM Cortex-M cores. Before creating the design, review the documentation first for the filter or filter design tool under consideration. If the documentation for the Toolbox or System object is not clear, then Contact MATLAB customer support at [Support - MATLAB](#).



```

7 %
8 frameSize = 256;
9 TOTAL_FRAMES = 2048;
10 fileReader = dsp.AudioFileReader("RockGuitar-16-44p1-stereo-72secs.wav", SamplesPerFrame=frameSize);
11 sampleRate = fileReader.SampleRate;
12 deviceWriter = audioDeviceWriter(SampleRate=sampleRate);
13
14 % Define a cascaded 2nd Order (SOS) equalizer centered at F1,F2,Fn ... Hz respectively,
15 % with gain in dB and Q-factor for a system running at sampleRate = 44.1kHz.
16 % Design the equalizers and obtain their B-Numerator and A-Denominator coefficients.
17 % [Num,Den] = designParamEQ(N,gain,centerFreq,bandwidth,"sos");
18
19 % Design the equalizer and obtain the B-Numerator and A-Denominator coefficients.
20 gain = [-2,1,2,2,0]; % new gain setting for each filter band in dB
21 % Design the equalizers and obtain their B-Numerator and A-Denominator coefficients.
22 % [Num,Den] = designParamEQ(N,gain,centerFreq,bandwidth);
23 % create a code gen function that computes the Filter Coefficients to be used in Cortex M ARM imple
24 [Num,Den] = genCoeffAudioEQ_float(sampleRate, gain);
25
26 % Implement equalizer with biquad filters using Second Order Sections (SOS) -> Coefficients Num,Den
27 % EQFilter = dsp.BiquadFilter( SOSMatrixSource="Input port", ScaleValuesInputPort=false);
28 % Function below will create the biquad filter
29 % EQFilter = filtercmsis_float(originalSignal,Num,Den);
30
31 % Process the audio with the EQ filter
32 count = 0;
33 % process and play audio for total # frames
34 while count < TOTAL_FRAMES
35     % get block of audio data to filter
36     originalSignal = fileReader();
37     % Process the audio data with EQ Filter
38     % equalizedSignal = EQFilter(originalSignal,Num,Den);
39     equalizedSignal = filtercmsis_float(originalSignal,Num,Den);
40
41     % Visualize the original and filtered signals
42     scope([originalSignal(:,1),equalizedSignal(:,1)]);
43     % Play the audio through PC audio
44     deviceWriter(equalizedSignal);
45     count = count + 1;
46 end
  
```

Figure 13. MATLAB Audio Equalizer Design

See [Figure 13. MATLAB Audio Equalizer Design](#), [Figure 14. Audio EQ Filter Coefficient Design](#), and [Figure 15. Audio EQ Biquad Filter Function](#), which details the MATLAB script (ModAudioFilterARMCortex.m) for the workflow used to develop and validate the Audio EQ filter. The steps are summarized as follows:

1. Set up Audio Toolbox resource PC Audio File Reader to acquire input signal to process with DSP Filter and set up resource File Writer to output the processed audio signal.
2. Design the Filter Coefficients used to implement the five Frequency Bands at the specified Gain Setting for each band. This determines the numerator and denominator (coefficients) of the Filter Transfer function.
3. Implement and run the Filter processing on the audio input signal.
4. Validate the Frequency Response of the input and output (processed) audio signals with the MATLAB Scope Tool.
5. Listen to processed audio to evaluate the Filter Design with the Audio File Writer on PC.
6. Repeat Steps 2,3,4 and 5 for Filter Design and Validation until the desired result is obtained.
7. Configure Embedded Coder for Cortex-M Source Code Generation.
8. Generate and Export Floating-Point or Fixed-Point Source Code for Cortex-M Processor.
9. Implement and optimize the Cortex-M processor.

The workflow is covered in detail in the next sections.

3.2 Create Filter Coefficient and Processing Functions

For the Embedded Coder tool to export CMSIS DSP C/C++ source code, the top-level MATLAB script (see [Figure 13. MATLAB Audio Equalizer Design](#)) must call modular functions that implement filter coefficient generation and filter processing. The functions must be put into their own module script file with a callable and exportable function format.

For more information on Embedded Coder [Code Generation from MATLAB Code](#) and specifics on fine-grain control of the generated functions, see [Embedded Coder Capabilities for Code Generation from MATLAB Code](#). It is recommended to take the time to review the MATLAB documentation on Embedded Coder. It can be difficult to get the code generation to work successfully when the codegen step is performed in [Section 3.3 Export Filter Functions with Embedded Coder Tool](#). Some example tutorials can be found at [codegen - Generate C/C++ code](#).

For this project, the Filter Coefficient design function prototype has a specific form, as shown in [Figure 14. Audio EQ Filter Coefficient Design](#).

```
function [Num,Den] = genCoeffAudioEQ_float(sampleRate, gain)
```

Input arguments are sampleRate, and gain

With **return argument list** at the end of the function:

```
[Num,Den] = designParamEQ(N, gain, centerFreq, bandwidth,"sos");
```

The function, designParamEQ(...) is the DSP System Toolbox filter design tool that creates the transfer function coefficients for the Second Order Sections ("sos") used with the dsp.BiquadFilter system object.

Audio EQ is implemented as a series of five cascaded Biquads with Second Order Sections. It is similar to a Band pass filter but with more direct control over the filter response characteristics set by user parameters that calculate the transfer function coefficients. The filter parameters are Gain, Band Center Frequency, and Bandwidth (the inverse is Quality – Q). A high bandwidth is a low Q value, such as 0.707. A low bandwidth is a high Q value, such as 2.0.

The band center frequencies and bandwidths have been chosen so the user can easily notice the changes to the input signal when DSP processing is active. The low frequency in band 1 (140 Hz and lower) has a low bandwidth (high Q), so gain changes are focused on the bass guitar and bass drum instruments of an input signal. The high-frequency response (10 kHz and higher) in band 5 has a high bandwidth (low Q) so gain changes are noticeable in the upper vocal harmonics, cymbals, drums, and percussion instruments.

MATLAB embedded coder produces the best results (for example, exports source code with no errors) when the top-level MATLAB script is made up of callable modular functions that perform minimal functionality.


```

1 function [Num,Den] = genCoeffAudioEQ_float(sampleRate, gain)
2 % Calculate the 5 Band Audio EQ filter coefficients with DSP Toolbox using float32 data type
3 %
4 %codegen
5 % Design a cascaded 2nd Order (SOS) equalizer centered at F1,F2,Fn ... Hz respectively,
6 % with gain in dB and Q-factor for a system running at sampleRate = 44.1kHz.
7 N = [2,2,2,2,2]; % order for each filter band -> 2nd Order filter sections cascaded
8 F1 = 140; % center freq in Hz - Band 1
9 W1 = F1/(sampleRate/2); % center freq (as ratio of freq / nyquist)
10 Q1 = 2.0; % Selectivity Q (Quality) - Band 1
11 BW1 = W1/Q1; % Bandwidth (inverse of Q) - Band 1
12
13 F2 = 1000; % center freq in Hz - Band 2
14 W2 = F2/(sampleRate/2); % center freq (as ratio of freq / nyquist)
15 Q2 = 1.0; % Selectivity Q (Quality) - Band 2
16 BW2 = W2/Q2; % Bandwidth (inverse of Q) - Band 2
17
18 F3 = 2400; % center freq in Hz - Band 3
19 W3 = F3/(sampleRate/2); % center freq (as ratio of freq / nyquist)
20 Q3 = 1.0; % Selectivity Q (Quality) - Band 3
21 BW3 = W3/Q3; % Bandwidth (inverse of Q) - Band 3
22
23 F4 = 4800; % center freq in Hz - Band 4
24 W4 = F4/(sampleRate/2); % center freq (as ratio of freq / nyquist)
25 Q4 = 0.707; % Selectivity Q (Quality) - Band 4
26 BW4 = W4/Q4; % Bandwidth (inverse of Q) - Band 4
27
28 F5 = 10000; % center freq in Hz - Band 5
29 W5 = F5/(sampleRate/2); % center freq (as ratio of freq / nyquist)
30 Q5 = 0.707; % Selectivity Q (Quality) - Band 5
31 BW5 = W5/Q5; % Bandwidth (inverse of Q) - Band 5
32 %
33 centerFreq = [W1,W2,W3,W4,W5]; % center freq for each filter band (radians)
34 bandwidth = [BW1,BW2,BW3,BW4,BW5]; % bandwidth of each band in (radians)
35 % Design the equalizers and obtain their B-Numerator and A-Denominator coefficients.
36 [Num,Den] = designParamEQ(N,gain,centerFreq,bandwidth,"sos");
37 end

```

Figure 14. Audio EQ Filter Coefficient Design

The Filter Processing function prototype has the form shown in [Figure 15. Audio EQ Biquad Filter Function](#)

function output = filtercmsis_float(input, num, den)

Input arguments are input (samples), and filter transfer function numerator and denominator

With **return argument output (samples)** at the end of the function:

output = filter(single(input), single(num), single(den));

The type-cast single(arg) forces the CMSIS DSP implementation to use single precision float (float32).

The Audio EQ filter processing function is implemented with the transfer function specified by input arguments num and den, which are the filter coefficients designed by the Filter Coefficient design function in [Figure 14. Audio EQ Filter Coefficient Design](#).

filter = dsp.BiquadFilter(...) is the DSP System Toolbox filter that creates the Audio EQ as five bands using cascaded Second Order Sections ("sos").

```

1 function output = filtercmsis_float(input,num,den)
2 % implement the function with DSP Toolbox filter with float32 data type
3 % and use Cortex M CMSIS DSP functions
4 %codegen
5 persistent filter;
6 if isempty(filter)
7     filter = dsp.BiquadFilter( SOSMatrixSource="Input port", ScaleValuesInputPort=false);
8 end
9 % typecast filter input args as single precision float to use CMSIS DSP
10 output = filter(single(input),single(num),single(den));
11 end
12

```

Figure 15. Audio EQ Biquad Filter Function

3.3 Export Filter Functions with Embedded Coder Tool

The MATLAB script is in [Figure 16. MATLAB Code Gen - Floating Point for Audio Equalizer](#) shows how to create an Embedded Coder configuration to generate source code for the Cortex-M processor using single precision float implementation. The **cfg.dialog** step opens a Configuration Dialog at run-time to review the settings for code generation. Review the information and click OK to proceed with code generation.

The **codegen** commands create the Cortex-M CMSIS DSP-based C language source code implementations in single precision float for the following functions:

For the Filter Coefficient design, the implementation is created with function **genCoeffAudioEQ_float(sampleRate, gain)** with single float args **sampleRate**, and **gain** for each frequency band and returns the **num** and **den** of the filter transfer function coefficients.

For the Filter processing, the implementation is created with the function

filtercmsis_float(input, num, den) with single float args **input** samples, **num** and **den** coefficients, and returns **output** samples.

```

47
48 %% Configure for ARM Cortex M CMSIS DSP Source Code Generation
49 % Create a code generation configuration object for use with codegen when generating a C/C++ static library.
50 %
51 % I used EmbeddedCodeConfig to generate Script which is pasted here after setting Hardware->Devices to use
52 % HW ARM Cortex M and GNU GCC compiler tool chain.
53 cfg = coder.config( "lib", "ecoder", true );
54 cfg.CodeReplacementLibrary = "ARM Cortex-M";
55 cfg.Toolchain = "GNU Tools for ARM Embedded Processors";
56 cfg.HardwareImplementation.ProdHWDeviceType = "ARM Compatible->ARM Cortex-M";
57 cfg.HardwareImplementation.TargetHWDeviceType = "Generic->Custom";
58 cfg.GenCodeOnly = true;
59
60 % Open the Custom code panel of the configuration dialog and verify the settings.
61 cfg.dialog
62
63 %% Generate Code for Cortex ARM M Floating Point versions of Filter and EQ Coefficient Generation
64 % Generate C code for the MATLAB function
65 % filtercmsis_float.m
66 %
67 % Typecast the input args to single (precision float) to get CMSIS DSP
68 % source code generated.
69 %
70 % Case the input datatype as single precision float
71 codegen filtercmsis_float -args {single(u1),single(Num),single(Den)} -config cfg -report
72 % When code generation finishes successfully, click View report to display the code generation report.
73 %
74 % See genCoeffAudioEQ_float.c file
75 codegen genCoeffAudioEQ_float -args {single(sampleRate),single(gain)} -config cfg -report
76

```

Figure 16. MATLAB Code Gen - Floating Point for Audio Equalizer

```

77 %% Generate Code for Cortex ARM M Fixed-Point versions of Filter and EQ Coefficient Generation
78 % Generate fixed-point C code for the MATLAB function filtercmsis_fixedpt_q15.m
79 % cast the Input data and Numerator and Denominator coeffs as
80 % Fixed-Point Numeric objects -> fi(v,s,w,f)
81 % v = data to convert
82 % s = Signedness : true
83 % w = Word length in bits : 16      => 16-bit
84 % f = Fraction length in bits : 15 => q15
85 codegen filtercmsis_fixedpt_q15 -args {fi(u1, true, 16, 15),fi(Num),fi(Den)} -config cfg -report
86
87 % See genCoeffAudioEQ_fixedpt.c file
88 % Design and Generate Audio EQ filter coefficients in Fixed-Point datatype
89 % [Num,Den] = designParamEQ(N,gain,centerFreq,bandwidth,"sos") function can not be converted to
90 % Fixed-Point
91 % A work around is to call with single precision args and convert
92 % [Num,Den] coefficients to fixed-point q14. Then pass to filter
93 % filtercmsis_fixedpt_q15() source code on ARM Cortex M
94 codegen genCoeffAudioEQ_fixedpt -args {single(sampleRate),single(gain)} -config cfg -report
95
96 %% Release Audio Toolbox Resources
97 release(deviceWriter)
98 release(fileReader)
99 release(scope)
100

```

Figure 17. MATLAB Code Gen - Fixed-Point for Audio Equalizer

Although we will focus on the floating-point implementation in the Audio EQ project, the fixed-point codegen command is shown here for those who wish to investigate fixed-point filters.

The MATLAB script is in [Figure 17. MATLAB Code Gen - Fixed-Point for Audio Equalizer](#) shows the codegen commands to create the Cortex-M CMSIS DSP-based C language source code implementations in fixed-point (q15, 16-bit integer) for the following functions:

For the Filter Coefficient design, **genCoeffAudioEQ_fixedpt(sampleRate, gain)** calls the filter design function **designParamEQ(..)**, but it does not have an equivalent in the MATLAB toolbox that can generate a set of fixed-point coefficients.

So, as a workaround, the floating-point version must be called to generate the **num** and **den** coefficients, and then it is required to convert to fixed-point values to call filter processing function **filtercmsis_fixedpt_q15(input, fi(Num), fi(Den))** where the input arguments are fixed-point values.

This brings up an important point: not all MATLAB signal processing and filter design Toolboxes provide Cortex-M embedded coder support for both floating-point and fixed-point source code implementations. Consult the MATLAB documentation first to determine the coder support for the Toolbox and Filter System objects you wish to use.

3.4 Understanding the Generated Source Code

The MATLAB code generation script is in [Figure 16. MATLAB Code Gen - Floating Point for Audio Equalizer](#) and [Figure 17. MATLAB Code Gen - Fixed-Point for Audio Equalizer](#) produces the source code output with folder structure shown in [Figure 18. MATLAB Code Generation Output](#).

The folder contains the MATLAB *.m files, which include the main MATLAB script `ModAudioFilterOnARM CortexM` that calls the exportable functions defined in `genCoeffAudioEQ_float.m` and `filtercmsis_float.m`. We discussed these functions previously, as shown in [Figure 14. Audio EQ Filter Coefficient Design](#) and [Figure 15. Audio EQ Biquad Filter Function](#).

ModAudioFilterOnARM CortexM			
Name	Date modified	Type	Size
codegen	2/6/2024 1:15 PM	File folder	
filtercmsis_fixedpt_q15.m	2/12/2024 3:43 PM	MATLAB Code	2 KB
filtercmsis_float.m	2/12/2024 3:50 PM	MATLAB Code	1 KB
genCoeffAudioEQ_fixedpt.m	2/14/2024 3:29 PM	MATLAB Code	3 KB
genCoeffAudioEQ_float.m	2/13/2024 11:19 AM	MATLAB Code	2 KB
ModAudioFilterARM CortexM.asv	2/14/2024 3:37 PM	ASV File	6 KB
ModAudioFilterARM CortexM.m	2/14/2024 3:47 PM	MATLAB Code	7 KB

Figure 18. MATLAB Code Generation Output

In [Figure 19. MATLAB Code Generation – Codegen Library](#), the codegen library folder contains the CMSIS DSP C language source code for the exportable functions. Notice that each `codegen` command called in the MATLAB script from [Figure 16. MATLAB Code Gen - Floating Point for Audio Equalizer](#) and [Figure 17. MATLAB Code Gen - Fixed-Point for Audio Equalizer](#) produces a separate folder with the source code for the exported function.

ModAudioFilterOnARM CortexM > codegen > lib		
Name	Date modified	Type
filtercmsis_fixedpt_q15	2/14/2024 3:48 PM	File folder
filtercmsis_float	2/14/2024 3:48 PM	File folder
genCoeffAudioEQ_fixedpt	2/14/2024 3:48 PM	File folder
genCoeffAudioEQ_float	11/12/2024 3:29 PM	File folder

Figure 19. MATLAB Code Generation – Codegen Library

ModAudioFilterOnARM CortexM > codegen > lib > genCoeffAudioEQ_float			
Name	Date modified	Type	Size
examples	2/14/2024 3:48 PM	File folder	
designHPEQFilter.c	2/14/2024 3:48 PM	C File	3 KB
designHPEQFilter.h	2/14/2024 3:48 PM	H File	1 KB
genCoeffAudioEQ_float.c	2/14/2024 3:48 PM	C File	7 KB
genCoeffAudioEQ_float.h	2/14/2024 3:48 PM	H File	1 KB
genCoeffAudioEQ_float_data.h	2/14/2024 3:48 PM	H File	1 KB
genCoeffAudioEQ_float_initialize.c	2/14/2024 3:48 PM	C File	1 KB
genCoeffAudioEQ_float_initialize.h	2/14/2024 3:48 PM	H File	1 KB
genCoeffAudioEQ_float_rtutil.c	2/14/2024 3:48 PM	C File	2 KB
genCoeffAudioEQ_float_rtutil.h	2/14/2024 3:48 PM	H File	1 KB
genCoeffAudioEQ_float_terminate.c	2/14/2024 3:48 PM	C File	1 KB
genCoeffAudioEQ_float_terminate.h	2/14/2024 3:48 PM	H File	1 KB
genCoeffAudioEQ_float_types.h	2/14/2024 3:48 PM	H File	1 KB
mw_cmsis.h	2/13/2024 11:24 AM	H File	6 KB
rt_nonfinite.c	2/14/2024 3:48 PM	C File	2 KB
rt_nonfinite.h	2/14/2024 3:48 PM	H File	1 KB
rtGetInf.c	2/14/2024 3:48 PM	C File	2 KB
rtGetInf.h	2/14/2024 3:48 PM	H File	1 KB
rtGetNaN.c	2/14/2024 3:48 PM	C File	1 KB
rtGetNaN.h	2/14/2024 3:48 PM	H File	1 KB
rtwtypes.h	2/14/2024 3:48 PM	H File	4 KB

Figure 20. MATLAB Code Generation – Generate Filter Coefficients

We will focus on the floating-point implementations in the Audio EQ project. [Figure 20. MATLAB Code Generation – Generate Filter Coefficients](#) shows, the source code files to generate the Filter Coefficients and [Figure 21. MATLAB Code Gen – Filter](#) shows the files to implement the Filter Processing.

Each source code directory contains an **examples** folder with main.c and main.h files that show a recommendation of how the source code should be implemented by your target application. The main.c is a starting point for porting this project to the RA8D1 version of the Audio EQ.

ModAudioFilterOnARM CortexM > codegen > lib > filtercmsis_float			
Name	Date modified	Type	Size
examples	2/14/2024 3:48 PM	File folder	
filtercmsis_float.c	2/14/2024 3:48 PM	C File	5 KB
filtercmsis_float.h	2/14/2024 3:48 PM	H File	1 KB
filtercmsis_float_data.c	2/14/2024 3:48 PM	C File	1 KB
filtercmsis_float_data.h	2/14/2024 3:48 PM	H File	1 KB
filtercmsis_float_initialize.c	2/14/2024 3:48 PM	C File	1 KB
filtercmsis_float_initialize.h	2/14/2024 3:48 PM	H File	1 KB
filtercmsis_float_terminate.c	2/14/2024 3:48 PM	C File	1 KB
filtercmsis_float_terminate.h	2/14/2024 3:48 PM	H File	1 KB
filtercmsis_float_types.h	2/14/2024 3:48 PM	H File	1 KB
rtwtypes.h	2/14/2024 3:48 PM	H File	4 KB

Figure 21. MATLAB Code Gen – Filter

See [Figure 22. Examples - Filter Coefficient Call](#) and [Figure 23. Examples - Filter Processing Call](#) for the Coefficient Generation and Filter Processing examples.

```
int main(int argc, char **argv)
{
    (void)argc;
    (void)argv;
    /* The initialize function is being called automatically from your entry-point
     * function. So, a call to initialize is not included here. */
    /* Invoke the entry-point functions.
    You can call entry-point functions multiple times. */
    main_genCoeffAudioEQ_float();
    /* Terminate the application.
    You do not need to do this more than one time. */
    genCoeffAudioEQ_float_terminate();
    return 0;
}

/*
 * Arguments      : void
 * Return Type    : void
 */
void main_genCoeffAudioEQ_float(void)
{
    float Num[15];
    float Den[10];
    float fv[5];
    /* Initialize function 'genCoeffAudioEQ_float' input arguments. */
    /* Initialize function input argument 'gain'. */
    /* Call the entry-point 'genCoeffAudioEQ_float'. */
    argInit_1x5_real32_T(fv);
    genCoeffAudioEQ_float(argInit_real32_T(), fv, Num, Den);
}
```

```
genCoeffAudioEQ_float.c X
2      * File: genCoeffAudioEQ_float.c
7
8      /* Include Files */
9      #include <genCoeffAudioEQ_float.h>
10     #include <math.h>
11     #include <designHPEQFilter.h>
12     #include <genCoeffAudioEQ_float_rtutil.h>
13     #include <rt_nonfinite.h>
14
15
16     /* Function Definitions */
17     /*
18      * Calculate the 5 Band Audio EQ filter coefficients with DSP Toolbox using
19      * float32 data type
20      *
21      *
22      * Arguments      : float sampleRate
23                      const float gain[5]
24                      float Num[15]
25                      float Den[10]
26      * Return Type    : void
27      */
28     void genCoeffAudioEQ_float(float sampleRate, const float gain[5], float Num[15],
29                               float Den[10])
30     {
```

Figure 22. Examples - Filter Coefficient Call

In [Figure 22. Examples – Filter Coefficient Call](#), the array of gain values for each frequency band is named **float fv[5]**. The variable was renamed to float gain[5] in the EK-RA8D1 ported version to improve readability in the source code. One point to remember is that the exported source code is a starting point for the target application and may require additional optimizations in the final application.

Details on optimizing and improving readability are covered in [Section 3.5 Modify Source Code for Best Performance on RA8 MCUs](#)

In [Figure 23. Examples - Filter Processing Call](#), numerator array **float fv1[15]**, and denominator array **float fv2[10]** were renamed to **num[15]** and **den[10]**. Later, it was determined to improve processing latency that all filter coefficient initialization should be performed once in a custom function **filtercmsis_biquad_init(const float num[15], const float den[10])**.

So, the call to **filtercmsis_float(...)** was changed to remove **num[15]** and **den[10]** in the function argument list. Since this is a stereo EQ, there are two instances required to maintain the filter state:

filtercmsis_float_L(input[32], output[32]) and **filtercmsis_float_R(input[32], output[32])** are called in the EK-RA8D1 ported version.

Note: The input audio samples array **float fv[75]** has size = 75 samples. This value was changed to 32 samples in the EK-RA8D1 ported version to better align with the maximum number of samples in a I2S Receive (DTC) transfer payload.

```

int main(int argc, char **argv)
{
    (void)argc;
    (void)argv;
    /* The initialize function is being called automatically from your entry-point
    * function. So, a call to initialize is not included here. */
    /* Invoke the entry-point functions.
    You can call entry-point functions multiple times. */
    main_filtercmsis_float();
    /* Terminate the application.
    You do not need to do this more than one time. */
    filtercmsis_float_terminate();
    return 0;
}

/*
 * Arguments    : void
 * Return Type  : void
 */
void main_filtercmsis_float(void)
{
    float fv[75];
    float output[75];
    float fv1[15];
    float fv2[10];
    /* Initialize function 'filtercmsis_float' input arguments. */
    /* Initialize function input argument 'input'. */
    /* Initialize function input argument 'num'. */
    /* Initialize function input argument 'den'. */
    /* Call the entry-point 'filtercmsis_float'. */
    argInit_75x1_real32_T(fv);
    argInit_3x5_real32_T(fv1);
    argInit_2x5_real32_T(fv2);
    filtercmsis_float(fv, fv1, fv2, output);
}

```



```

56
57 // I removed num,den since filtercmsis_biquad_init(num,den) called once to init biquad filter
58 void filtercmsis_float_L(const float input[32], float output[32])
59 {
60     /*
61     * I Modified the original Matlab codegen source code by moving biquad filter init to
62     * separate function filtercmsis_biquad_init(const float num[15],const float den[10])
63     */
64

```

Figure 23. Examples - Filter Processing Call

3.5 Modify Source Code for Best Performance on RA8 MCUs

After analyzing the MATLAB source code generated with the provided example functions in [Figure 22. Examples - Filter Coefficient Call](#) and [Figure 23. Examples - Filter Processing Call](#), it was determined that modifications would be required for the best performance on the RA8 MCU. A multi-threaded FreeRTOS project has been created in e² studio to use the generated MATLAB source code. A thread called DSP Thread is used to implement the Audio EQ processing with the ARM CMSIS DSP Library. The entry function **dsp_thread_entry**(void *pvParameters) is implemented to handle DSP filter coefficient and memory initialization as well as processing with **DSP_Process_Data**(...) which runs continuously to process audio data arriving from the I2S connected Audio Codec. See [Figure 24. DSP Thread](#).

```

56  /* DSP Thread entry function */
57  /* pvParameters contains TaskHandle_t */
58  void dsp_thread_entry(void *pvParameters)
59  {
60      FSP_PARAMETER_NOT_USED (pvParameters);
61      const TickType_t hw_wait_ms = 5000 / portTICK_PERIOD_MS;
62
63      DSP_Process_Init();
64      xEventGroupWaitBits(g_dsp_app_event, HARDWARE_INIT_DONE, pdFALSE, pdFALSE, hw_wait_ms);
65
66      #ifdef ENABLE_DSP_PROFILING
67      /* Measure with GPT0 start timer for benchmarking */
68      R_BSP_MODULE_START(FSP_IP_GPT, 0);
69      system_clock_hz = R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_CPUCLK);
70      #endif
71
72      while (1)
73      {
74          if (g_rxSamplesToProcess > 0)
75          {
76              #ifdef ENABLE_DSP_PROFILING
77              /* GPT0: Clear counter */
78              R_GPT0->GTCNT = 0;
79              /* GPT0 Timer Measure: Get the timer counter */
80              ts_cycle_dsp_0 = R_GPT0->GTCNT;
81              /* GPT0: Start timer */
82              R_GPT0->GTCR = 1;
83              /* Measure with Data Watch point/Trace Unit: get start cycle count */
84              ts_0 = DWT->CYCCNT;
85              #endif
86
87              if (1 == g_run_dsp_process)
88              {
89                  DSP_Process_Data(g_rx_circ_buf[g_rxdata_read_index], g_tx_circ_buf[g_txdata_write_index]);
90                  /* Advance audio circular buffer indices and keep circular */
91                  g_rxdata_read_index++;
92                  g_rxdata_read_index %= BUF_ROW_SIZE;
93                  g_txdata_write_index++;
94                  g_txdata_write_index %= BUF_ROW_SIZE;
95              }
96          }
71

```

Figure 24. DSP Thread

In the MATLAB script, filter coefficient design and generation are called every time the filter processing is run on a group of audio samples. However, it was determined that the EK-RA8D1 embedded system processing latency could be improved by running the filter coefficient generation and Biquad filter initialization once with **DSP_Process_Init**() before calling the filter processing **DSP_Process_Data**(...), which runs continuously on the incoming audio. **DSP_Process_Init**() should be called again when any of the filter parameters change such as Gain. For example, in the case of a user parameter control with LCD is added to the project.

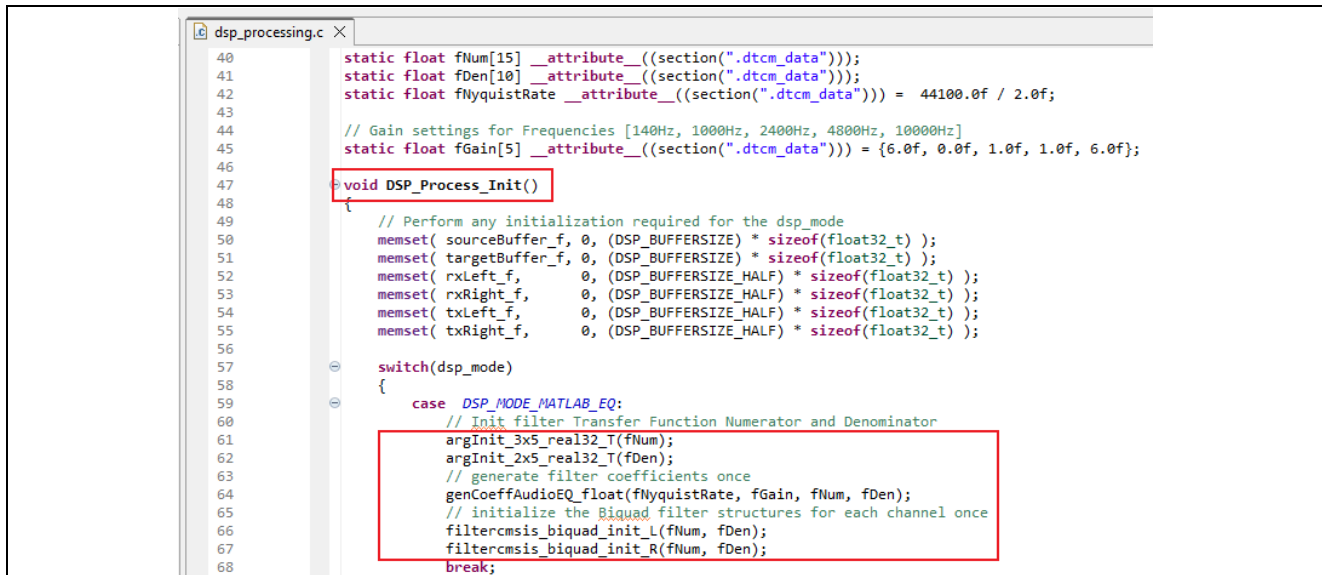


Figure 25. DSP Process Init

As seen in [Figure 25. DSP Process Init](#), `DSP_Process_Init()` is implemented by calling the source code functions generated by MATLAB.

The functions `filtercmsis_biquad_init_L(fNum, fDen)` and `filtercmsis_biquad_init_R(fNum, fDen)` are custom functions added to initialize the CMSIS Biquad filter with `arm_biquad_cascade_df2T_init_f32(...)`.

The next section covers how to enable the DSP process latency measurement.

3.6 DSP Process Latency Measurement

3.6.1 How to Enable Latency Measurement

Support has been added to the Audio EQ project to enable execution time (latency) measurement on the Audio Codec data handler and DSP process. In this section, the term latency refers to the time it takes to process a group of samples with a filter or send and receive a group of samples with the Audio Codec data handler. This allows profiling before and after changes are made to the Compiler optimization settings and DSP processing implementation.

As observed in [Figure 26. Enable Latency Measurement](#), to enable latency measurement use `#define ENABLE_CODEC_PROFILING` and

`#define ENABLE_DSP_PROFILING` in files `codec_thread_entry.c` and `dsp_thread_entry.c`.

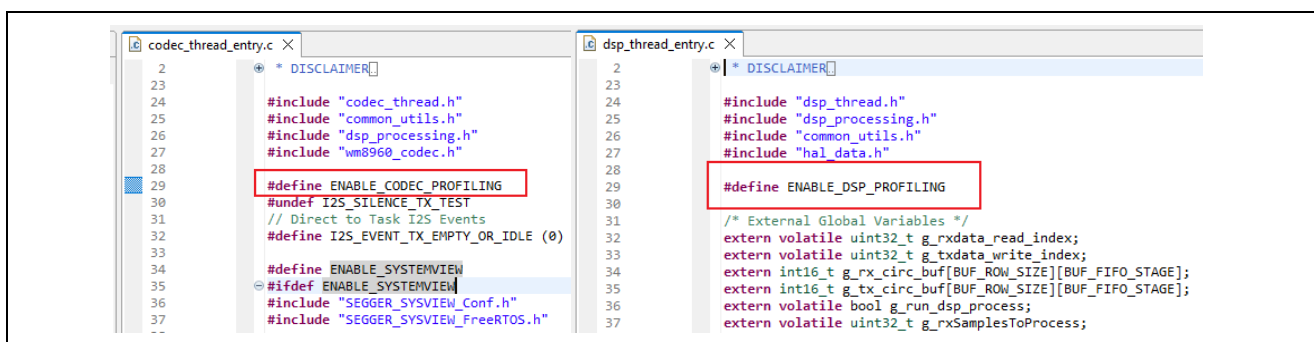


Figure 26. Enable Latency Measurement

3.6.2 Adding Support Code to Measure Latency

As shown in [Figure 27. Store DWT and GPT Before and After Calling the DSP Process Function](#), the DWT counter (DWT->CYCCNT) and GPT timer count are saved before and after calling the Audio Codec handler `R_SSI_WriteRead(...)` or DSP processing function.

When running a debug session with e2 studio IDE, the DWT measurement is valid but is not for operating the board in standalone mode. When the Command Line Interface is used, the GPT timer measurement is

always valid for debug or standalone operation. The latency values are displayed in the Command Line Interface (CLI) with menu options 3 and 4. See [Section 2.4 Download and Operate Project](#) for more details.

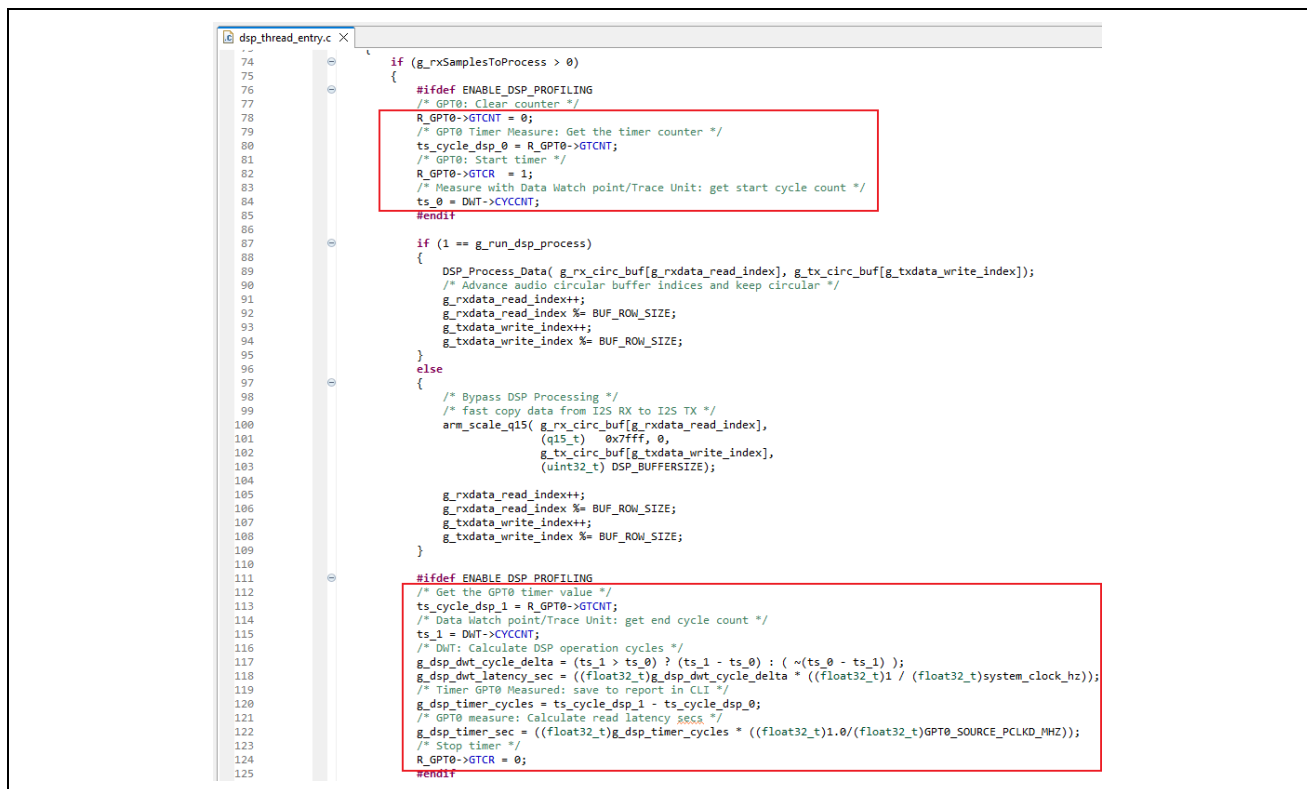


Figure 27. Store DWT and GPT Before and After Calling the DSP Process Function

As observed in the [Figure 27. Store DWT and GPT Before and After Calling the DSP Process Function](#), the execution time of the DSP processing function call is calculated based on DWT or GPT timer cycles and the MCU system or timer clock frequency using the following formula:

Time = DWT cycles x (1 / System Clock)

Time = GPT0 Timer cycles x (1 / Timer Clock)

Note:

- DWT cycles are the count of MCU execution cycles.
- R_GPT0 -> GTCNT contains the count of GPT0 timer cycles
- For the EK-RA8D1, the system clock is CPUCLK set to 480 MHz in this application project.
- For the EK-RA8D1, the GPT0 timer clock source is PCLKD set to 120 MHz

3.7 Changing Segger SystemView Support

A system level perspective of the Audio EQ project can be obtained by using support for Segger SystemView. The tool is integrated into the project and enabled by default to show the process thread priorities, timing, and resource utilization.

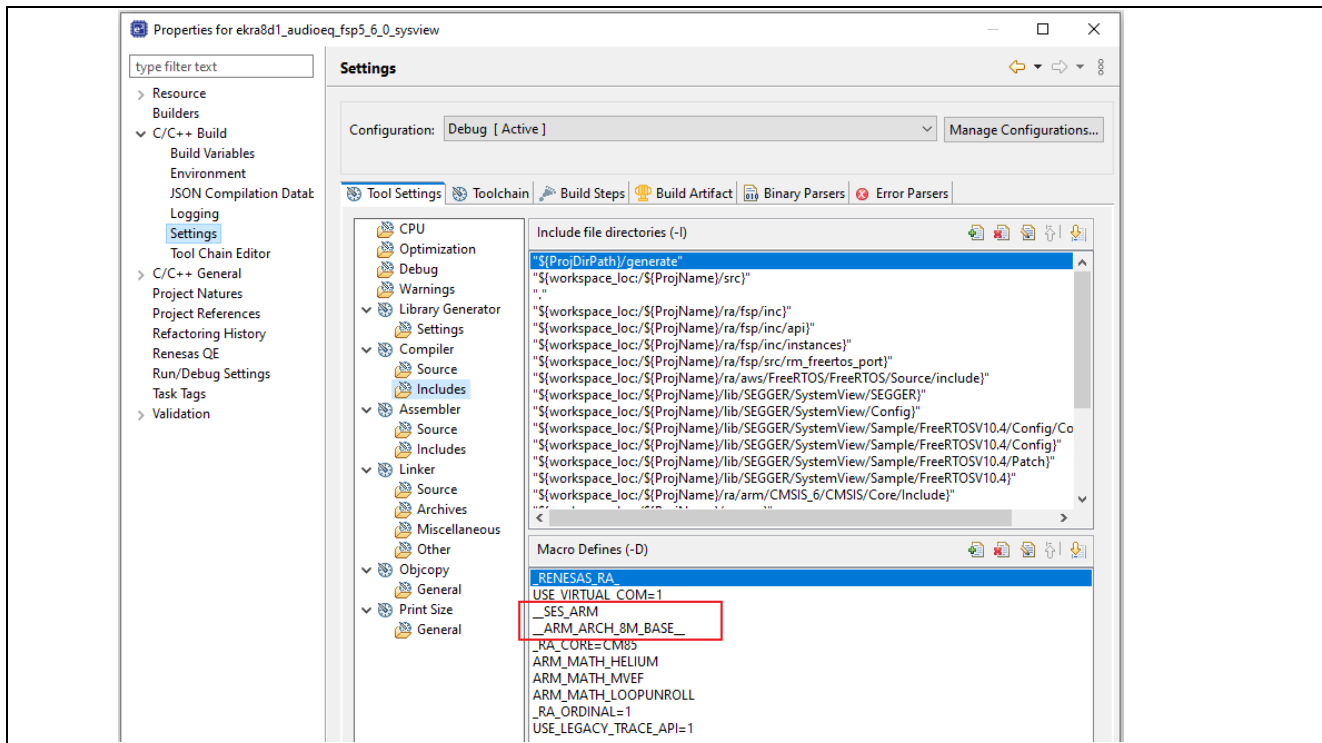


Figure 28. Compiler Macro Defines for SystemView

As observed in [Figure 28. Compiler Macro Defines for SystemView](#), [Figure 29. Codec Thread – Defines for SystemView](#) and [Figure 30. Codec Thread - Enable Trace for SystemView](#), Segger SystemView support is enabled to profile the multi-threaded processes in the Audio EQ project. The Macro Defines (-D) `__SES_ARM` and `__ARM_ARCH_8M_BASE__` are used to set the SystemView configuration options required for the RA8 MCU (CM85 core).

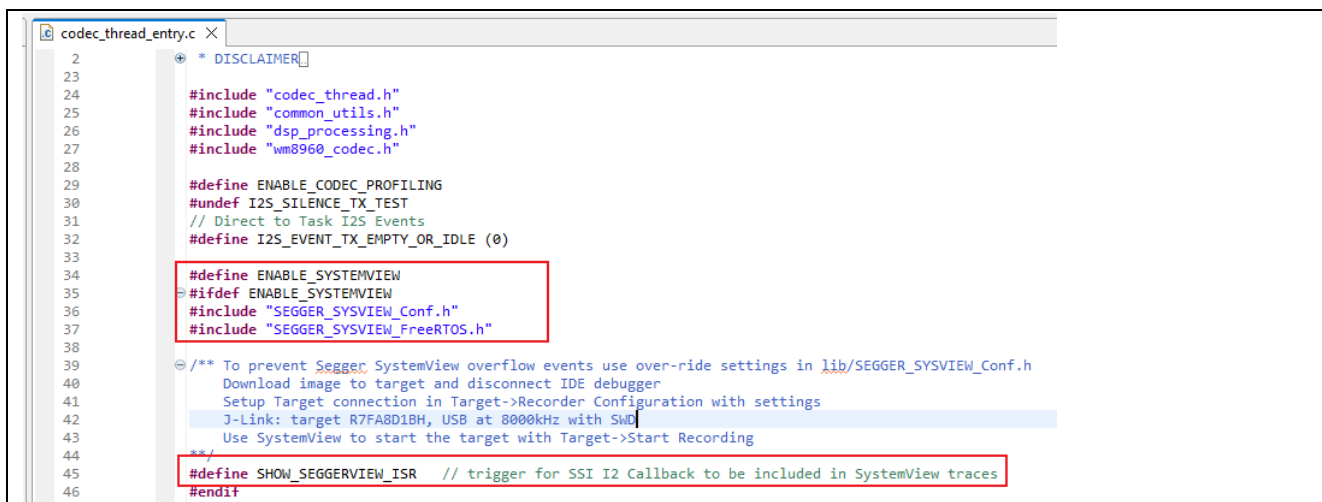


Figure 29. Codec Thread – Defines for SystemView

See [Figure 29. Codec Thread – Defines for SystemView](#) and [Figure 30. Codec Thread - Enable Trace for SystemView](#), where SystemView support is enabled in the source code. All SystemView support can be disabled with `#undef ENABLE_SYSTEMVIEW`.

Since it can generate a large amount of trace buffer data, the Audio Codec handler ISR trace can be disabled with `#undef SHOW_SEGGERVIEW_ISR` as required.

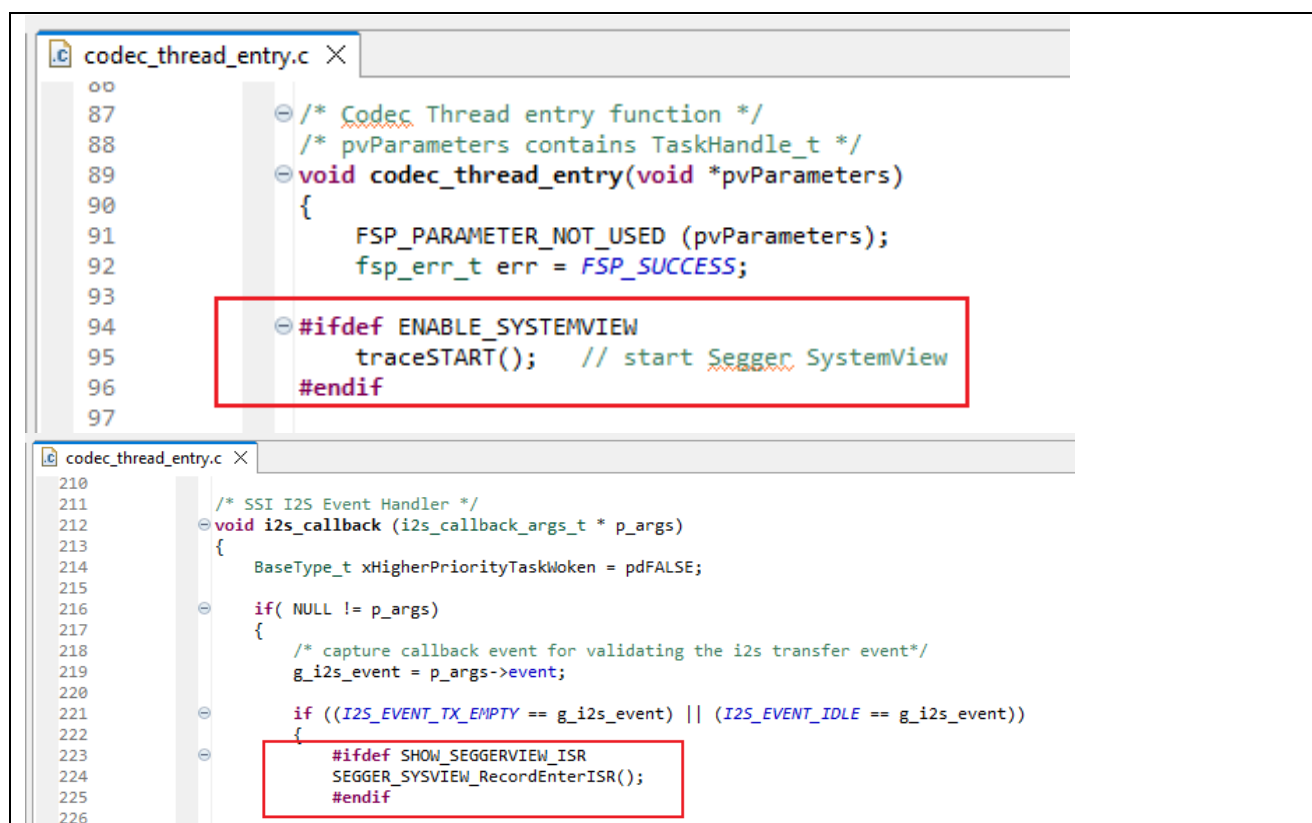


Figure 30. Codec Thread - Enable Trace for SystemView

3.7.1 Integration of Segger SystemView into Example Project

Segger SystemView tool was integrated into the Audio EQ example project running on FSP v5.6 with FreeRTOS v10.6.1

Use the following steps to integrate the SystemView source code:

- Follow the instructions to integrate the SystemView source code with the project from [FreeRTOS with SystemView - SEGGER Wiki](#).
- Use the source code from [SEGGER - The Embedded Experts - Downloads - SystemView](#)
- Choose SystemView Target Sources v3.58 and use the Sample -> FreeRTOSV10.4 even though the FreeRTOS version in FSP v5.6.0 is v10.6.1.
- Segger Tech Support recommends V10.4 example for use with FSP v5.6 and FreeRTOS V10.6.1 with EK-RA8D1 Core CM85.
- Add #include "SEGGER_SYSVIEW_FreeRTOS.h" to all FreeRTOS/Source files that have occurrences of #include "FreeRTOS.h".
- Be sure to place #include "SEGGER_SYSVIEW_FreeRTOS.h" before #include "FreeRTOS.h" in all the affected files. See [Figure 31. SystemView Integration](#)

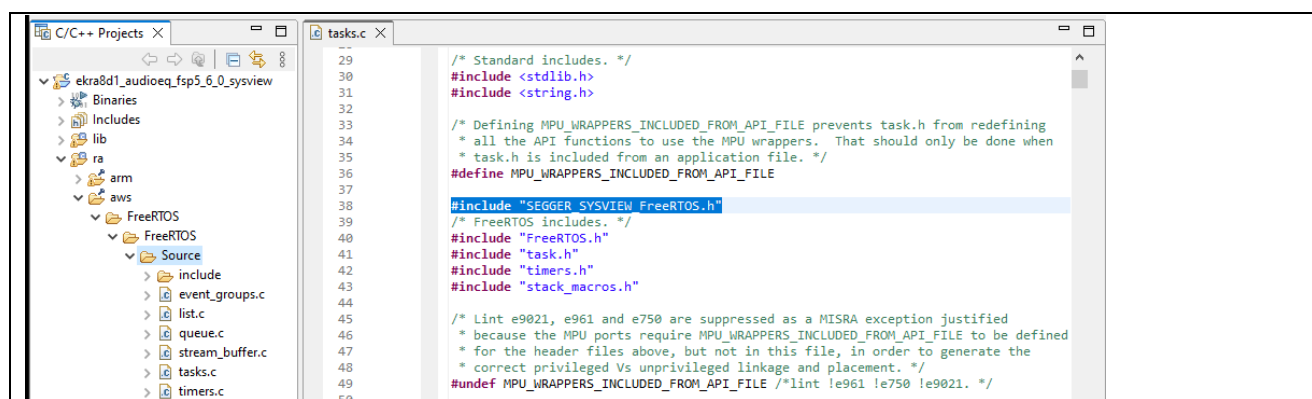


Figure 31. SystemView Integration

To build Segger SystemView with LLVM 18.1.3, the directories under `/lib/SEGGER/SystemView/...` must be excluded from the build since the files are not required in the build:

- `project/lib/SEGGER/SystemView/Sample/RTT`
- `project/lib/SEGGER/SystemView/SEGGER/Syscalls`

If the Audio EQ example project is ever updated to use a newer version of FSP with FreeRTOS, the SystemView source code must be reintegrated into the example project by following the steps covered in [Section 3.7 Codec Thread – Defines for SystemView](#).

Additional coverage of SystemView is provided in [Section 4.6 Using Segger SystemView](#) with instructions on how to measure CPU Utilization for the Audio EQ project.

4. Audio Equalizer Software Architecture

The next sections detail the software architecture for the Audio EQ project. There are several sub-systems that handle moving the audio data from the Audio Codec, storing the data, processing the data with the DSP filters, and providing the Command Line Interface to control the Audio EQ and measure latency.

Central to the architecture of the system is the Audio Codec. Since the Audio Codec depends on the buffering sub-system, the Audio Circular Buffer is introduced first.

4.1 Audio Circular Buffers

As seen in [Figure 32. Audio Circular Buffers](#), the Audio EQ, requires a data buffering sub-system that receives and transmits data with the Audio Codec connected to the SSI I2S Master Interface.

The Circular Buffer scheme buffers about 743 milliseconds of audio data. This permits the CMSIS DSP Filter to obtain data for processing and prevents I2S RX and TX FIFO underflow and overflow conditions by decoupling the audio codec interface from the audio DSP processing task.

The I2S sample rate is 44.1 kHz with stereo data @ 16-bit sample data per channel. So, the RX and TX Circular Buffer provides 32×1024 of 16-bit data = 32,768 samples of 16-bit data per channel. At 44100 samples/sec, this provides $(32768 \text{ samples}) / (44100 \text{ samples/sec}) = 0.743 \text{ secs}$ of buffered audio data per channel.

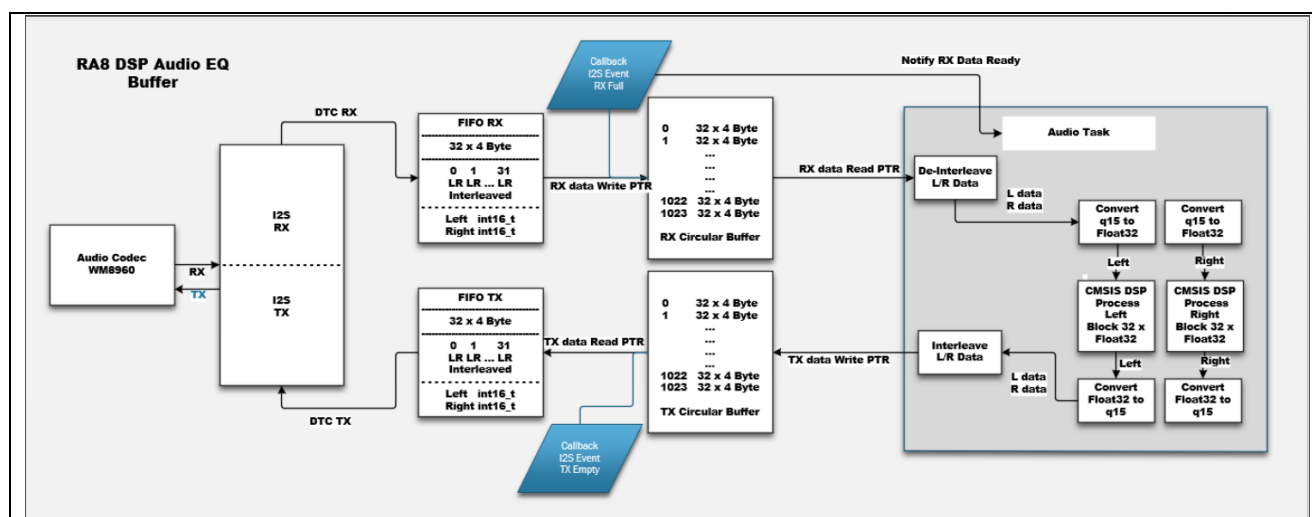


Figure 32. Audio Circular Buffers

The maximum length of the RX and TX Circular Buffers can be scaled as required. However, consult the total amount of internal or external memory available for the RA8 MCU.

```

dsp_processing.h
2  * DISCLAIMER
23
24  #ifndef DSP_PROCESSING_H_
25  #define DSP_PROCESSING_H_
26
27  #include "hal_data.h"
28
29  /* Defines */
30  #define BUF_ROW_SIZE 1024
31  #define BUF_FIFO_STAGE 64
32  #define DSP_BUFFERSIZE 64
33  #define DSP_BUFFERSIZE_HALF 32
34

codec_thread_entry.c
71
72  int16_t g_rx_circ_buf[BUF_ROW_SIZE][BUF_FIFO_STAGE] BSP_ALIGN_VARIABLE(32) BSP_PLACE_IN_SECTION(".nocache");
73  int16_t g_tx_circ_buf[BUF_ROW_SIZE][BUF_FIFO_STAGE] BSP_ALIGN_VARIABLE(32) BSP_PLACE_IN_SECTION(".nocache");
74
75
76  /* Local Variables */
77  static uint32_t rxdata_write_index = 0;
78  static uint32_t txdata_read_index = 0;
79

```

Figure 33. Audio Circular Buffers – Source Code

If the total audio input-to-output latency of the system must be reduced, then the buffer size can be decreased. Refer to [Figure 33. Audio Circular Buffers – Source Code](#) in the source code for #defines **BUF_ROW_SIZE** and **BUF_FIFO_STAGE** that set the Circular Buffer size.

If the block size of data (a set of samples) processed by the CMSIS Filter in one function call must be increased, then the max length of the buffer may be increased. The system input-to-output latency and CMSIS Filter processing time can be balanced by adjusting the Circular Buffer maximum length, and the CMSIS Filter processing block size. In Audio EQ, the DSP processing is performed on a block of 32 samples each for left-channel and right-channel audio every time the DSP filter function is called.

The Audio Codec I2S Callback Event handles filling the RX Circular Buffer with data received from the DTC FIFO RX. Both read/write pointers wrap back to offset 0 when the end of the buffer is reached. The audio task is notified when RX data is present in the buffer and ready for processing. The I2S Callback Event TX FIFO Empty handles the dequeuing of the TX Circular Buffer and filling the DTC FIFO TX.

In [Figure 33. Audio Circular Buffers – Source Code](#), the #define **BUF_FIFO_STAGE 64** was set to match the maximum number of bytes that can be transferred by the DTC FIFO in one transaction. The DTC FIFO left- and right-channel data is interleaved, giving a total of 64 x 2 bytes = (left: 32 x 2 bytes) + (right: 32 x 2 bytes).

In [Figure 32. Audio Circular Buffers](#), the q15 to Float32 data conversion blocks use the fast CMSIS DSP Library functions.

The De-Interleave and Interleave blocks handle converting the serialized I2S format (left channel then right channel) data to two parallel streams of channel processing with the CMSIS DSP functions.

In this project, CMSIS DSP functions process one channel at a time, so two function calls are required to handle stereo audio.

4.2 Audio Codec Thread

In conjunction with the audio circular buffers, the Audio Codec thread is responsible for receiving and transmitting the audio data from the I2S-connected Audio Codec. The DSP processing thread picks up the data stored in the receive circular buffer, processes one block of data (left channel: 32 samples of 16-bit data, right channel: 32 samples of 16-bit data), and stores the results back to the transmit circular buffer.

4.2.1 I2S Stack Settings

As observed in [Figure 34. FSP Configuration - I2S Stack Settings](#), the FSP configuration for the I2S Master stack settings is shown. The callback `i2s_callback` implemented in `codec_thread_entry.c` is used to receive and transmit the audio data that is stored in the Audio Circular Buffers. See [Section 4.1 Audio Circular Buffers](#) and [Section 4.2.3 I2S Callback](#) for more information.

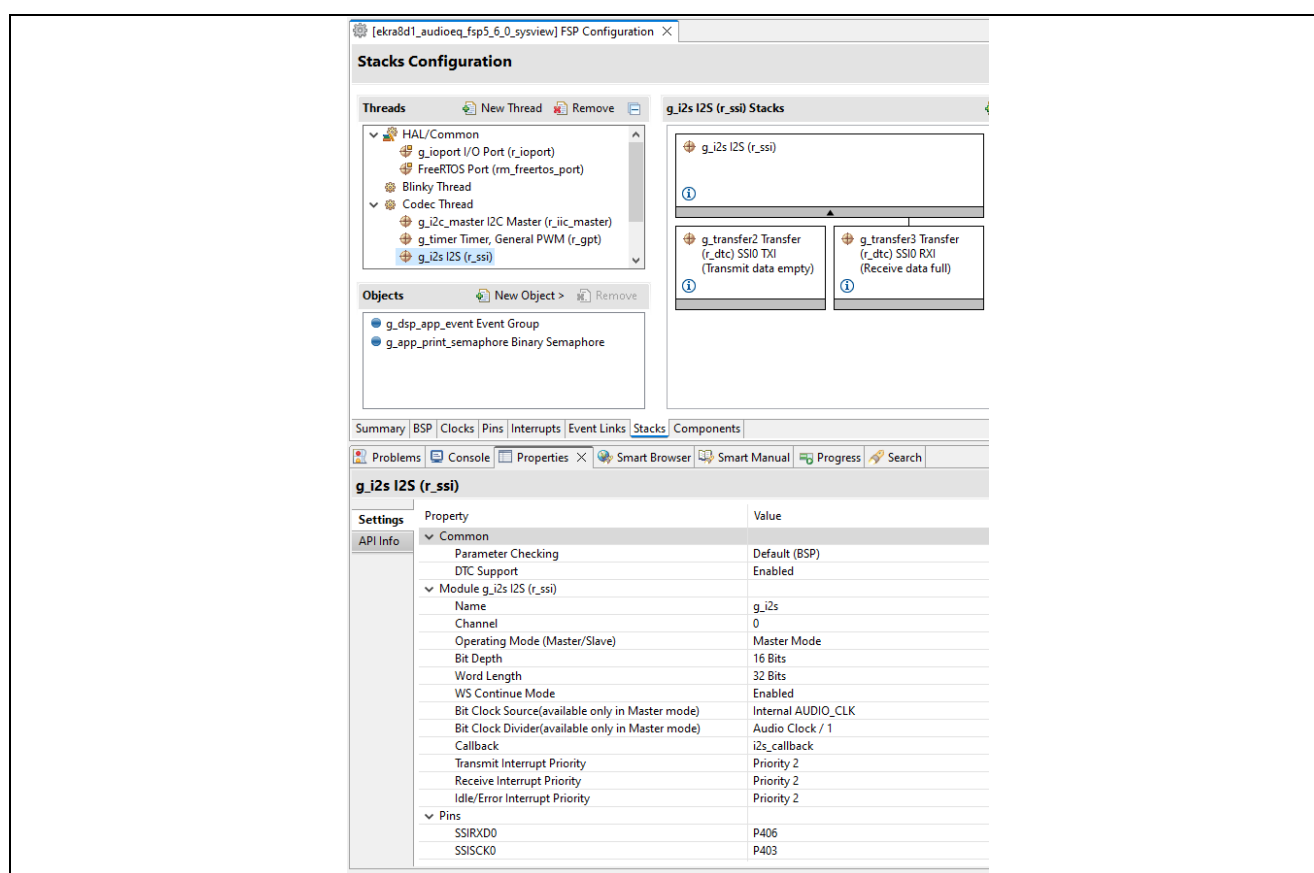


Figure 34. FSP Configuration - I2S Stack Settings

The I2S Master clock source for the Audio Codec is the **Internal AUDIO_CLK** which is provided by the General PWM Timer running on channel 2.

Note in the RA8 MCU series, only the GTIOC2A output or AUDIO_CLK input pin can be used as AUDIO_CLK. Refer to the Hardware User's Manual, as it may vary depending on the MCU series. The details for the I2S Audio Clock are covered in the next section.

The I2S Master bit depth is 16-bit per sample, and the Word Length is 32-bit since one frame of data consists of interleaved left and right samples. 1 x Left sample (16) + 1 x Right sample (16) = 32-bits.

In the example project, I2S DTC-supported transfers are enabled, as observed in [Figure 34. FSP Configuration - I2S Stack Settings](#). DTC support is optional and can be disabled. With DTC support enabled, the DSP processing latency is improved. For example, with DTC disabled, the Audio EQ takes 9.8 μ s to

process 32 samples of stereo data. With DTC enabled, the latency is 9.64 μ s to process 32 samples of stereo data. A percentage change of -1.63% is not a significant decrease, but for large amounts of data processing, and percent decrease helps with improving processing latency.

Use Segger SystemView to see the effect on the system throughput and MCU with DTC transfers enabled and disabled. To disable DTC support, see [Figure 35. FSP Configuration - I2S DTC Support Disabled](#).

For more information, see [Section 4.6 Using Segger SystemView](#).

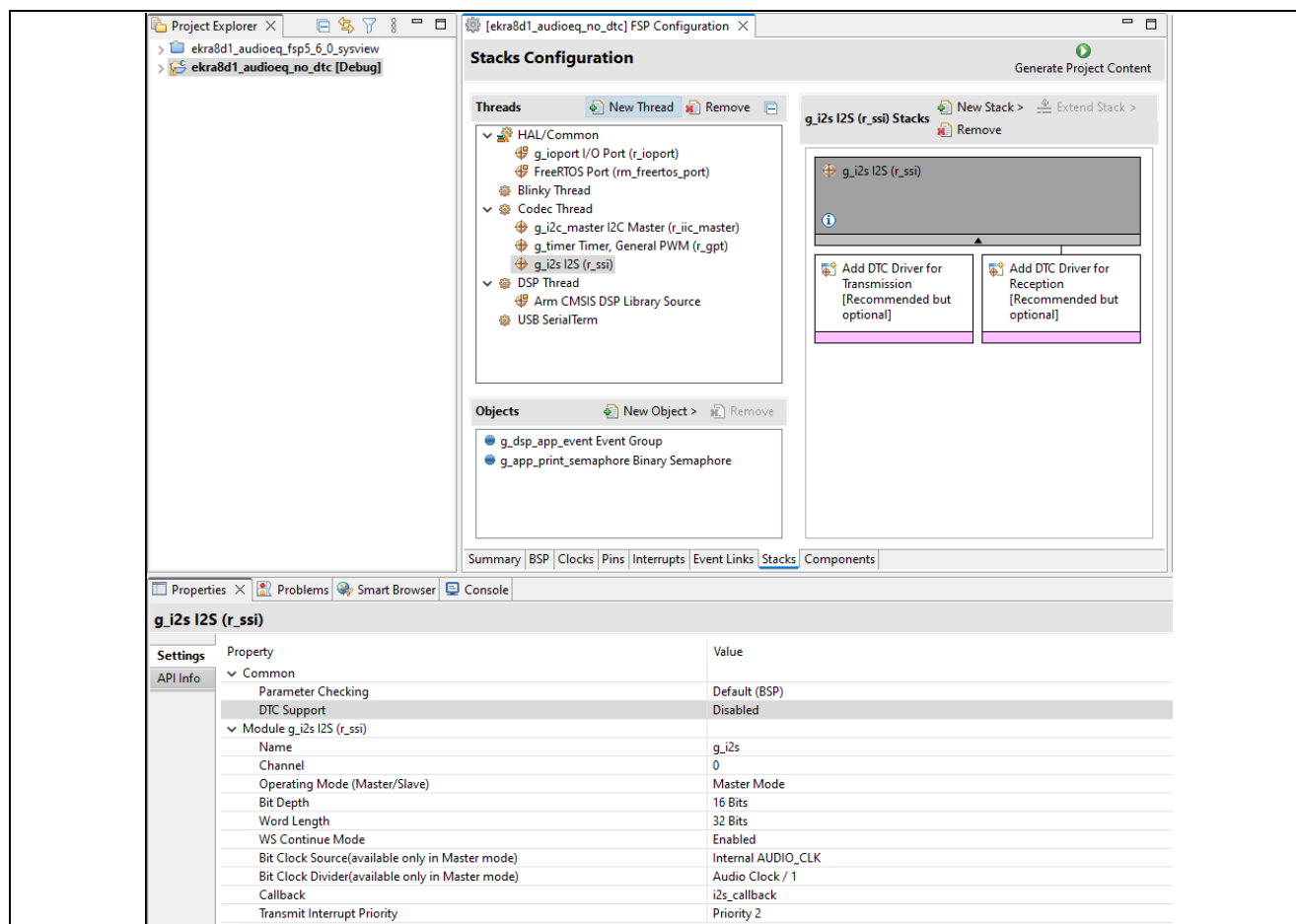


Figure 35 FSP Configuration - I2S DTC Support Disabled

4.2.2 I2S Audio Clock

In [Figure 36. FSP Configuration GPT Timer Settings - I2S Audio Clock](#), the FSP configuration is shown for the General PWM Timer, which is being used as the internal I2S Audio Clock source.

The bit rate of the Timer is used to clock the interleaved stereo channel data of 16-bit per channel sample at 44100 samples/sec.

Timer Bit Rate: 1411200 bits/sec = 44100 samples/sec x 16-bits/channel-sample x 2 channels

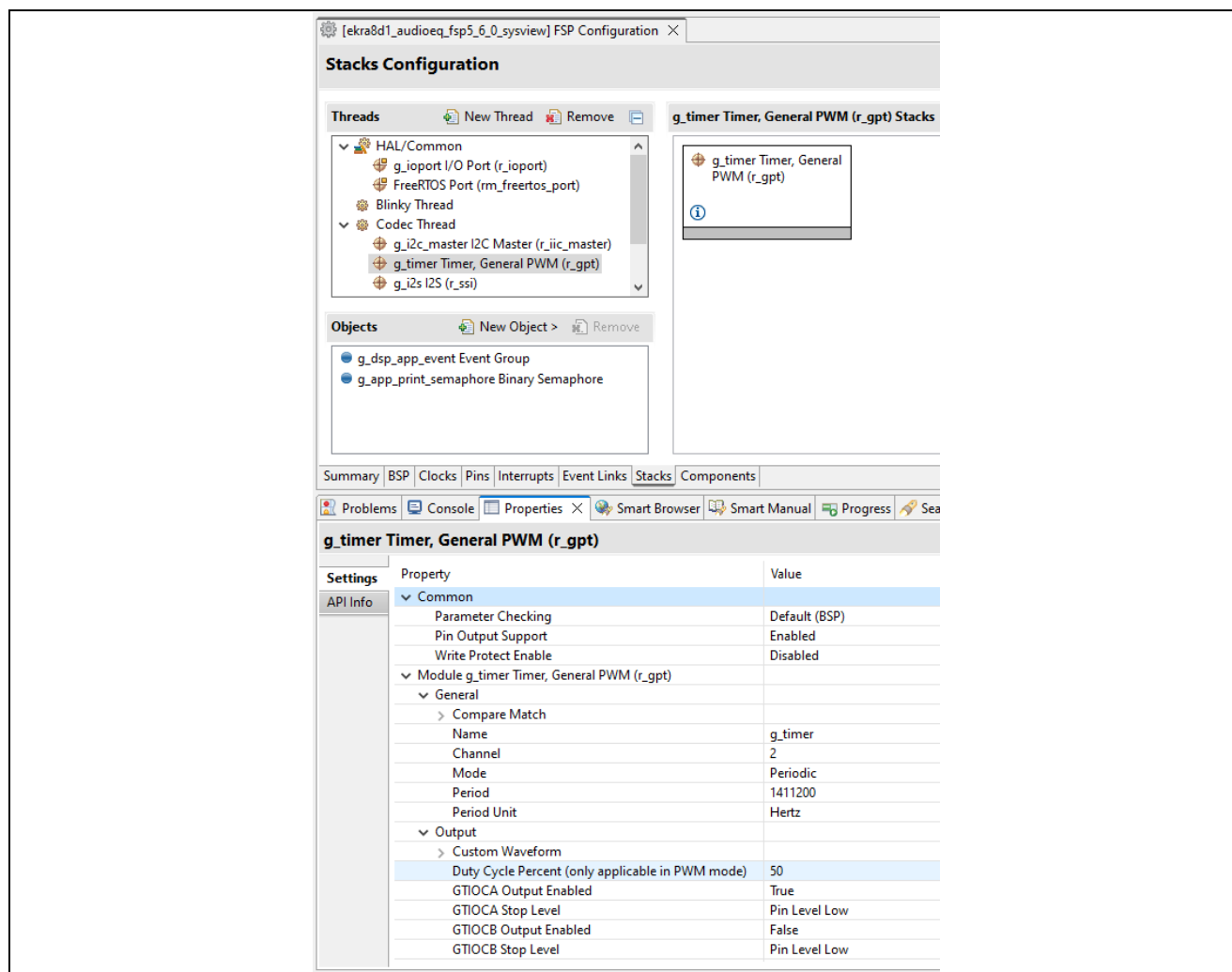


Figure 36. FSP Configuration GPT Timer Settings - I2S Audio Clock

4.2.3 I2S Callback

In [Figure 37. Codec Thread - I2S Callback](#), the I2S Master callback function `i2s_callback(...)` is implemented in `codec_thread_entry.c`. It is used to receive and transmit the audio data stored in the Audio Circular Buffers. For I2S Events **I2S_EVENT_TX_EMPTY** and **I2S_EVENT_IDLE**, the DTC RX FIFO is read and written to the Receive Audio Circular Buffer, and the DTC TX FIFO is filled with the Transmit Audio Circular Buffer data.

```

212 /* SSI I2S Event Handler */
213 void i2s_callback (i2s_callback_args_t * p_args)
214 {
215     BaseType_t xHigherPriorityTaskWoken = pdFALSE;
216
217     if( NULL != p_args)
218     {
219         /* capture callback event for validating the i2s transfer event*/
220         g_i2s_event = p_args->event;
221
222         if ((I2S_EVENT_TX_EMPTY == g_i2s_event) || (I2S_EVENT_IDLE == g_i2s_event))
223         {
224             #ifdef SHOW_SEGGERVIEW_ISR
225
226             /* Use Direct to Task Notifications */
227             uint32_t ulStatusRegister = 0;
228             /* Unblock the handling task so the task can perform any processing */
229             xTaskNotifyIndexedFromISR( codec_thread,
230                                     I2S_EVENT_TX_EMPTY_OR_IDLE,
231                                     ulStatusRegister,
232                                     eSetBits,
233                                     &xHigherPriorityTaskWoken );
234
235             #ifdef ENABLE_CODEC_PROFILING
236
237             R_SSI_WriteRead(&g_i2s_ctrl,
238                             g_tx_circ_buf[txdata_read_index],
239                             g_rx_circ_buf[rxdata_write_index],
240                             (BUF_FIFO_STAGE) * sizeof(int16_t));
241
242             #ifdef ENABLE_CODEC_PROFILING
243
244             #ifdef SHOW_SEGGERVIEW_ISR
245
246             /* Force a context switch if xHigherPriorityTaskWoken is now set to pdTRUE.
247              The macro used to do this is dependent on the port and may be called portEND_SWITCHING_ISR. */
248             portYIELD_FROM_ISR( xHigherPriorityTaskWoken );
249
250         }
251     }

```

Figure 37. Codec Thread - I2S Callback

The **R_SSI_WriteRead(...)** performs this operation, and both RX and TX operations are handled simultaneously to keep the I2S interface data flow from stalling. If the data flow stalls, then a click or pop in the audio input or output can be heard.

Each call to **R_SSI_WriteRead(...)** reads and writes 32 samples of left-and right-channel data (16-bit integer) interleaved to the Audio Codec over the I2S Master interface.

The initialization of the Audio Codec is covered next in [Section 4.2.4 Codec Initialization](#).

4.2.4 Codec Initialization

The Wolfson WM8960 is the Audio Codec used for the Audio EQ project. The I2C interface is used to initialize and control the codec, and the digital data of the audio is transmitted and received by interfacing with the MCU I2S Master interface. In [Figure 38. Codec Initialization](#), the Codec thread initializes the device with the function `init_codec()`.

In file `codec_thread_entry.c`, the `#defines` **CMSIS_PROCESS_I2S_MODE**, **CODEC_ANALOG_PASSTHRU**, and **CODEC_ADC_DAC_PASSTHRU** set the operating mode for the Audio Codec. For Audio EQ normal operation, the mode **CMSIS_PROCESS_I2S_MODE** is enabled.

```

100 void codec_thread_entry(void *pvParameters)
101 {
102     FSP_PARAMETER_NOT_USED (pvParameters);
103     fsp_err_t err = FSP_SUCCESS;
104
105     #ifdef ENABLE_SYSTEMVIEW
106     traceSTART(); // start Segger SystemView
107     #endif
108
109     /*
110     * Audio Codec Op Modes
111     * Pick one mode to define
112     * CMSIS_PROCESS_I2S_MODE : used for DSP Filter Operation
113     * CODEC_ANALOG_PASSTHRU : test mode that patches analog input
114     * to analog output - Digital ADC/DAC path not used
115     * CODEC_ADC_DAC_PASSTHRU : test mode that patches Digital
116     * input to Digital output, relies on MCU for I2S clock gen
117     */
118     #define CMSIS_PROCESS_I2S_MODE
119     #undef CODEC_ANALOG_PASSTHRU
120     #undef CODEC_ADC_DAC_PASSTHRU
121
122     #ifdef CMSIS_PROCESS_I2S_MODE
123     err = init_codec();
124     if (FSP_SUCCESS != err)
125     {
126         /* Audio Codec init failed, so cleanup*/
127         deinit_codec();
128         APP_ERR_TRAP(err);
129     }
130
131     #elif defined CODEC_ANALOG_PASSTHRU
132     init_codec_analog_passthru();
133     #elif defined CODEC_ADC_DAC_PASSTHRU
134     init_codec_loopback();
135     #endif
136 }
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2
```

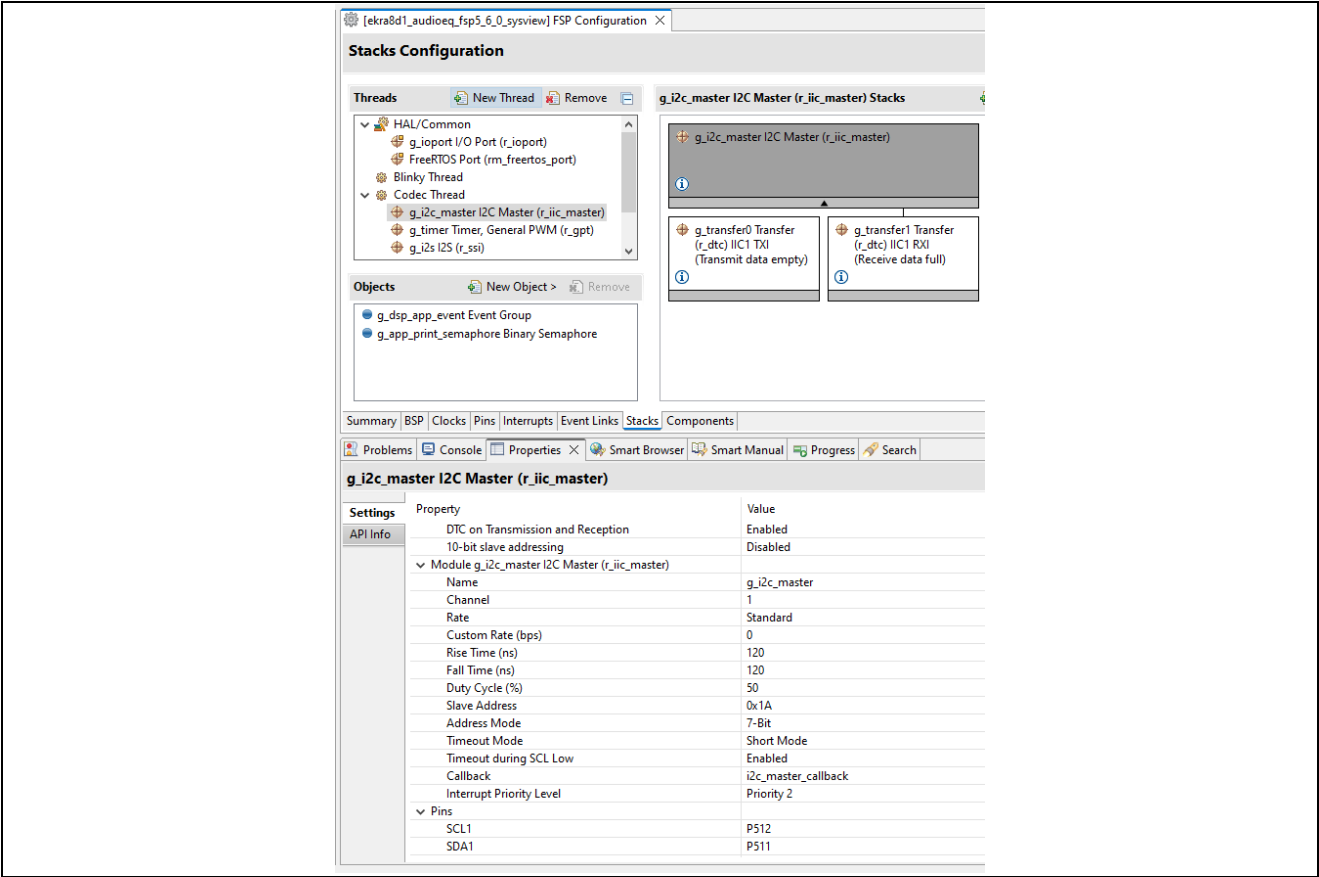


Figure 39. FSP Configuration - I2C Master Stack Settings

4.2.5 I2S Initialization

In the Codec thread, the SSI module is initialized in Master mode for use with the Audio Codec and uses the General Purpose Timer in Periodic mode as the Audio SSI bit clock source. As observed in [Figure 40. I2S Initialization](#), the Codec thread initializes the SSI and Timer modules, starts the SSI I2S callback event processing, and notifies the DSP processing thread that the audio hardware initialization has been completed.

```

127      /* Open SSI module */
128      err = R_SSI_Open(&g_i2s_ctrl, &g_i2s_cfg);
129      /* Handle error */
130      if (FSP_SUCCESS != err)
131      {
132          APP_ERR_TRAP(err);
133      }
134
135      /* Open GPT in periodic mode as internal clock for SSI bit clock */
136      err = R_GPT_Open(&g_timer_ctrl, &g_timer_cfg);
137      /* Handle error */
138      if (FSP_SUCCESS != err)
139      {
140          deinit_ssi();
141          APP_ERR_TRAP(err);
142      }
143      /* Start GPT in periodic mode */
144      err = R_GPT_Start(&g_timer_ctrl);
145      /* Handle error */
146      if (FSP_SUCCESS != err)
147      {
148          // APP_ERR_PRINT("\r\n GPT start failed, Closing SSI and GPT \r\n");
149          deinit_ssi();
150          deinit_gpt();
151          APP_ERR_TRAP(err);
152      }
153
154      for (uint32_t row = 0; row < BUF_ROW_SIZE; row++)
155      {
156          memset( g_rx_circ_buf[row], 0, (BUF_FIFO_STAGE)* sizeof(int16_t) );
157          memset( g_tx_circ_buf[row], 0, (BUF_FIFO_STAGE)* sizeof(int16_t) );
158          #ifdef I2S_SILENCE_TX_TEST
159          memset( g_tx_silence_buf[row], 0, (BUF_FIFO_STAGE)* sizeof(int16_t) );
160          #endif
161      }
162
163      /*Start SSI I2S callback event processing */
164      err = R_SSI_WriteRead(&g_i2s_ctrl, g_tx_circ_buf[0], g_rx_circ_buf[0], (BUF_FIFO_STAGE)* sizeof(int16_t));
165      if (FSP_SUCCESS != err)
166      {
167          deinit_ssi();
168          deinit_gpt();
169          APP_ERR_TRAP(err);
170      }
171      #ifdef ENABLE_CODEC_PROFILING
172      /* Measure with GPT0 start timer for benchmarking */
173      R_BSP_MODULE_START(FSP_IP_GPT, 0);
174      system_clock_hz = R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_CPUCLK);
175      #endif
176
177      xEventGroupSetBits(g_dsp_app_event, HARDWARE_INIT_DONE);

```

Figure 40. I2S Initialization

4.2.6 Codec Thread Processing

In [Figure 41. Codec Thread Processing](#), the Codec thread processing waits for I2S events, maintains the Audio Circular Buffer indices, and determines the number of received samples available to process.

```

177      xEventGroupSetBits(g_dsp_app_event, HARDWARE_INIT_DONE);
178      while (1)
179      {
180          /* Direct to Task Notifications: Wait for any event to start processing */
181          xTaskNotifyWaitIndexed( I2S_EVENT_TX_EMPTY_OR_IDLE, /* Wait for Notification */
182                                0x00, /* Don't clear any bits on entry. */
183                                I2S_EVENT_TX_EMPTY_OR_IDLE, /* Clear bit on exit. */
184                                &ulNotifiedValue, /* Receives the notification value. */
185                                portMAX_DELAY ); /* Block indefinitely. */
186
187          if ((I2S_EVENT_TX_EMPTY == g_i2s_event) || (I2S_EVENT_IDLE == g_i2s_event))
188          {
189              /* TX Empty and Idle Events: RX and TX FIFO unload/load was started in the SSI I2S Callback.
190               Increment buffer indices and calculate the audio left/right channel data frames available for the DSP process thread. */
191
192              /* Buffer enough RX Circular buffer data to start DSP processing */
193              g_rxSamplesToProcess = (rxdata_write_index >= g_rxdata_read_index) ?
194                                   (rxdata_write_index - g_rxdata_read_index) : ( g_rxdata_read_index - rxdata_write_index);
195              if ( g_rxSamplesToProcess < (BUF_ROW_SIZE >> 1) )
196              {
197                  g_rxSamplesToProcess = 0;
198              }
199
200              /* Increment indices and keep Circular */
201              txdata_read_index++;
202              txdata_read_index %= BUF_ROW_SIZE;
203              rxdata_write_index++;
204              rxdata_write_index %= BUF_ROW_SIZE;
205          }
206          vTaskDelay (1);
207      }
208  }

```

Figure 41. Codec Thread Processing

The DSP Processing sub-system is covered next in [Section 4.3 DSP Processing Thread](#).

4.3 DSP Processing Thread

DSP processing is handled in the DSP thread, which implements Audio EQ with the ARM CMSIS DSP Library. The entry function **dsp_thread_entry**(void *pvParameters) handles DSP filter initialization as well as the processing of data. The DSP processing runs continuously to filter audio data arriving from the I2S connected Audio Codec. See [Figure 42. DSP Processing Thread](#).

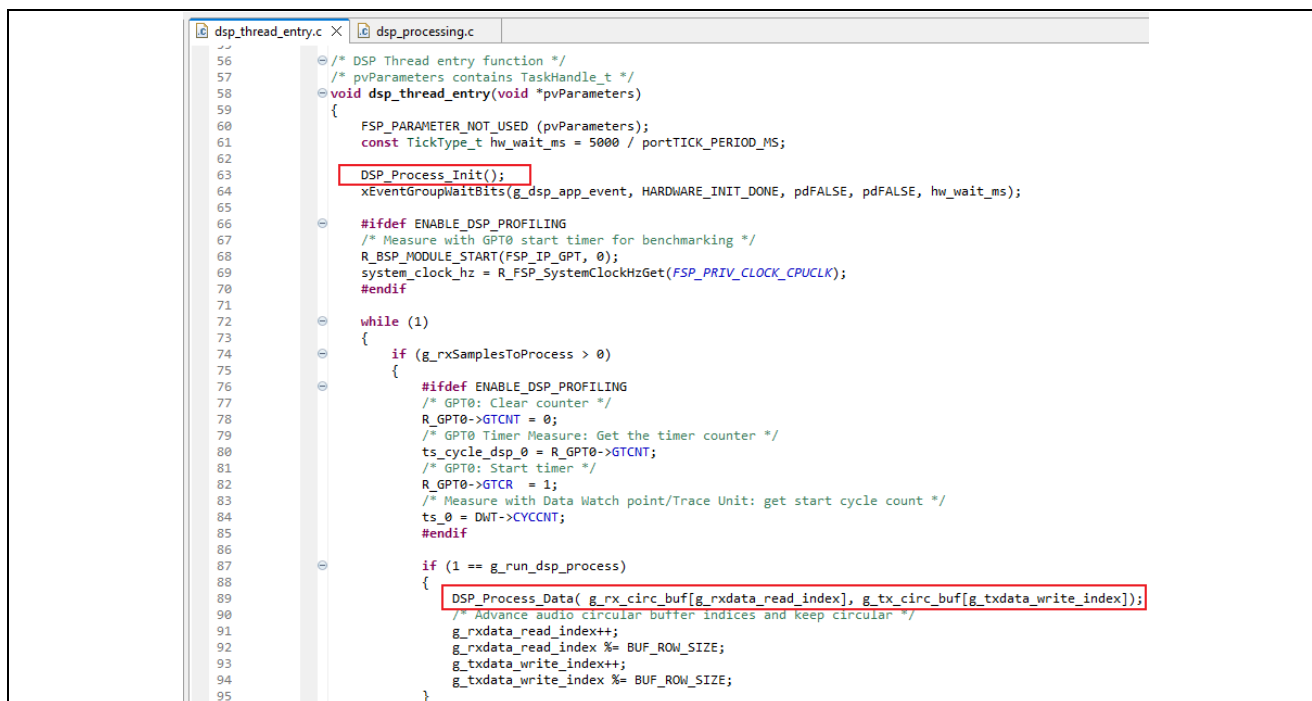


Figure 42. DSP Processing Thread

The function **DSP_Process_Init()** handles creating the Audio EQ filter coefficients, initializing the Biquad filter structures, and clearing the processing memory.

For more details, see [Section 4.3.1 DSP Process Initialization](#).

Signal processing is performed by **DSP_Process_Data()** which is covered in [Section 4.3.2 DSP Processing](#).

4.3.1 DSP Process Initialization

As observed in [Figure 43. DSP Process Init](#), the DSP process initialization handles creating the Audio EQ filter coefficients, setting up the Biquad filter structures, and clearing the processing memory. The initialization is run once at MCU start-up.

```

24 #include "dsp_processing.h"
25 #include "filtercmsis_entry.h"
26 #include "filtercmsis_float.h"
27 #include <genCoeffAudioEQ_float.h>
28
29 #define SCALE_FOR_HEADROOM
30
31 e_dsp_mode_t dsp_mode = DSP_MODE_MATLAB_EQ;
32
33 // Use Data Tightly Coupled Memory
34 static float32_t sourceBuffer_f[DSP_BUFFER_SIZE] __attribute__((section(".dcm_data"))) __attribute__((aligned(8)));
35 static float32_t targetBuffer_f[DSP_BUFFER_SIZE] __attribute__((section(".dcm_data"))) __attribute__((aligned(8)));
36 static float32_t rxLeft_f[DSP_BUFFER_SIZE_HALF] __attribute__((section(".dcm_data"))) __attribute__((aligned(8)));
37 static float32_t rxRight_f[DSP_BUFFER_SIZE_HALF] __attribute__((section(".dcm_data"))) __attribute__((aligned(8)));
38 static float32_t txLeft_f[DSP_BUFFER_SIZE_HALF] __attribute__((section(".dcm_data"))) __attribute__((aligned(8)));
39 static float32_t txRight_f[DSP_BUFFER_SIZE_HALF] __attribute__((section(".dcm_data"))) __attribute__((aligned(8)));
40 static float fNum[15] __attribute__((section(".dcm_data")));
41 static float fDen[10] __attribute__((section(".dcm_data")));
42 static float fNyquistRate __attribute__((section(".dcm_data"))) = 44100.0f / 2.0f;
43
44 // Gain settings for Frequencies [140Hz, 1000Hz, 2400Hz, 4800Hz, 10000Hz]
45 static float fGain[5] __attribute__((section(".dcm_data"))) = {6.0f, 0.0f, 1.0f, 1.0f, 6.0f};
46
47 void DSP_Process_Init()
48 {
49     // Perform any initialization required for the dsp_mode
50     memset( sourceBuffer_f, 0, (DSP_BUFFER_SIZE) * sizeof(float32_t) );
51     memset( targetBuffer_f, 0, (DSP_BUFFER_SIZE) * sizeof(float32_t) );
52     memset( rxLeft_f, 0, (DSP_BUFFER_SIZE_HALF) * sizeof(float32_t) );
53     memset( rxRight_f, 0, (DSP_BUFFER_SIZE_HALF) * sizeof(float32_t) );
54     memset( txLeft_f, 0, (DSP_BUFFER_SIZE_HALF) * sizeof(float32_t) );
55     memset( txRight_f, 0, (DSP_BUFFER_SIZE_HALF) * sizeof(float32_t) );
56
57     switch(dsp_mode)
58     {
59         case DSP_MODE_MATLAB_EQ:
60             // Init filter Transfer Function Numerator and Denominator
61             argInit_3x5_real32_T(fNum);
62             argInit_2x5_real32_T(fDen);
63             // generate filter coefficients once
64             genCoeffAudioEQ_float(fNyquistRate, fGain, fNum, fDen);
65             // initialize the Biquad filter structures for each channel once
66             filtercmsis_biquad_init_L(fNum, fDen);
67             filtercmsis_biquad_init_R(fNum, fDen);
68             break;
69         case DSP_MODE_GAIN:
70             break;
71         default:
72             break;
73     }
74 }

```

Figure 43. DSP Process Init

The coefficients are generated for the Audio EQ filter based on the audio band gains set with array **fGain[5]**. The function **genCoeffAudioEQ_float(fNyquistRate, fGain, fNum, fDen)** is generated by MATLAB as discussed previously in [Section 3.4 Understanding the Generated Source Code](#). It creates the filter coefficients for the numerator and denominator of the filter transfer function.

The functions **filtercmsis_biquad_init_L(fNum, fDen)** and **filtercmsis_biquad_init_R(fNum, fDen)** are custom implementations that were factored out of the MATLAB generated code. The reason for the custom functions is to initialize the Biquad filter structures once in **DSP_Process_Init()** instead of calling the function each time a group of samples are filtered in the DSP Processing loop. This greatly improves the DSP processing latency.

See [Section 3.5 Modify Source Code for Best Performance on RA8 MCUs](#) for more details.

4.3.2 DSP Processing

In [Figure 44. Audio EQ and Buffers](#), DSP processing of the audio data is implemented on a block of 32 samples after the RX Circular Buffer fixed-point data is deinterleaved into separate left- and right-channel data and converted to floating point data. After processing with the Audio EQ CMSIS DSP functions, the left- and right-channel data is converted back to fixed-point data and interleaved. Finally, the processed data is stored in the TX Circular Buffer to be transmitted to the Audio Codec.

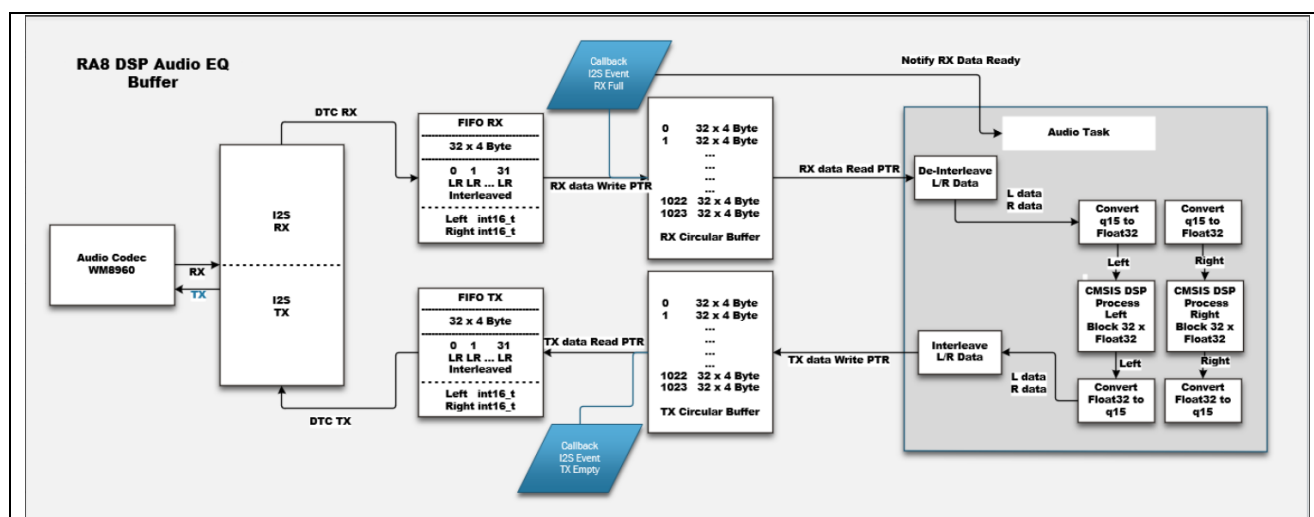


Figure 44. Audio EQ and Buffers

```

dsp_processing.c
79 int DSP_Process_Data(int16_t *sourceBuffer, int16_t *targetBuffer)
80 {
81     uint16_t index1, index2;
82     int status;
83
84     // convert I2S Codec sourceBuffer (64 samples int16_t = 32 Left + 32 Right audio) to
85     arm_q15_to_float(
86         (const q15_t *) sourceBuffer,
87         (float32_t *) sourceBuffer_f,
88         (uint32_t) DSP_BUFFERSIZE);
89
90     #ifndef SCALE_FOR_HEADROOM
91     /*
92     ** Scale down by 20%. This provides additional processing headroom so
93     ** that the graphic EQ can apply gain at bands without clipping.
94     */
95     arm_scale_f32( sourceBuffer_f, 0.80f, sourceBuffer_f, (uint32_t) DSP_BUFFERSIZE);
96     #endif
97
98     // De-Interleave source L/R data to separate left and right channel buffers
99     for (index1 = 0, index2 = 0; index1 < DSP_BUFFERSIZE_HALF; index1++)
100     {
101         rxLeft_f[index1] = sourceBuffer_f[index2++];
102         rxRight_f[index1] = sourceBuffer_f[index2++];
103     }
104
105     switch(dsp_mode)
106     {
107     case DSP_MODE_MATLAB_EQ:
108         filtercmsis_entry_L( rxLeft_f, txLeft_f); // process Left Channel
109         filtercmsis_entry_R( rxRight_f, txRight_f); // process Right Channel
110         break;
111     case DSP_MODE_GAIN:
112     default:
113         arm_scale_f32( rxLeft_f, 2.0f, txLeft_f, (uint32_t) DSP_BUFFERSIZE_HALF);
114         arm_scale_f32( rxRight_f, 2.0f, txRight_f, (uint32_t) DSP_BUFFERSIZE_HALF);
115         break;
116     }
117
118     // Interleave Left/Right channel Data into targetBuffer_f
119     for (index1 = 0, index2 = 0; index1 < DSP_BUFFERSIZE_HALF; index1++)
120     {
121         targetBuffer_f[index2++] = txLeft_f[index1];
122         targetBuffer_f[index2++] = txRight_f[index1];
123     }
124
125     // convert targetBuffer (float32_t) to int16_t for I2S TX Codec
126     arm_float_to_q15(
127         (const float32_t *) targetBuffer_f,
128         (q15_t *) targetBuffer,
129         (uint32_t) DSP_BUFFERSIZE );

```

Figure 45. DSP Process Data

In Figure 45. DSP Process Data, the source code implementation of the Audio EQ DSP processing is shown.

The functions **filtercmsis_entry_L(rxLeft_f, txLeft_f)** and **filtercmsis_entry_R(rxRight_f, txRight_f)** handle the Audio EQ filtering. The function to process one channel of data is generated by MATLAB as discussed in Section 3.4 Understanding the Generated Source Code. In this case, two calls to the function are required to process each of the two audio channels.

4.3.3 Audio EQ Filter Implementation

The functions in `filtercmsis_entry.c` are generated by MATLAB and implement the Audio EQ filter.

To decrease the DSP processing latency, an optimization modification has been made to move the filter coefficient generation function call out of `filtercmsis_entry_L(...)` and `filtercmsis_entry_R(...)`.

The filter coefficient generation call to `genCoeffAudioEQ_float(fNyquistRate, fGain, fNum, fDen)` is moved to `DSP_Process_Init(...)`. For more information, see [Section 4.3.1 DSP Process Initialization](#).

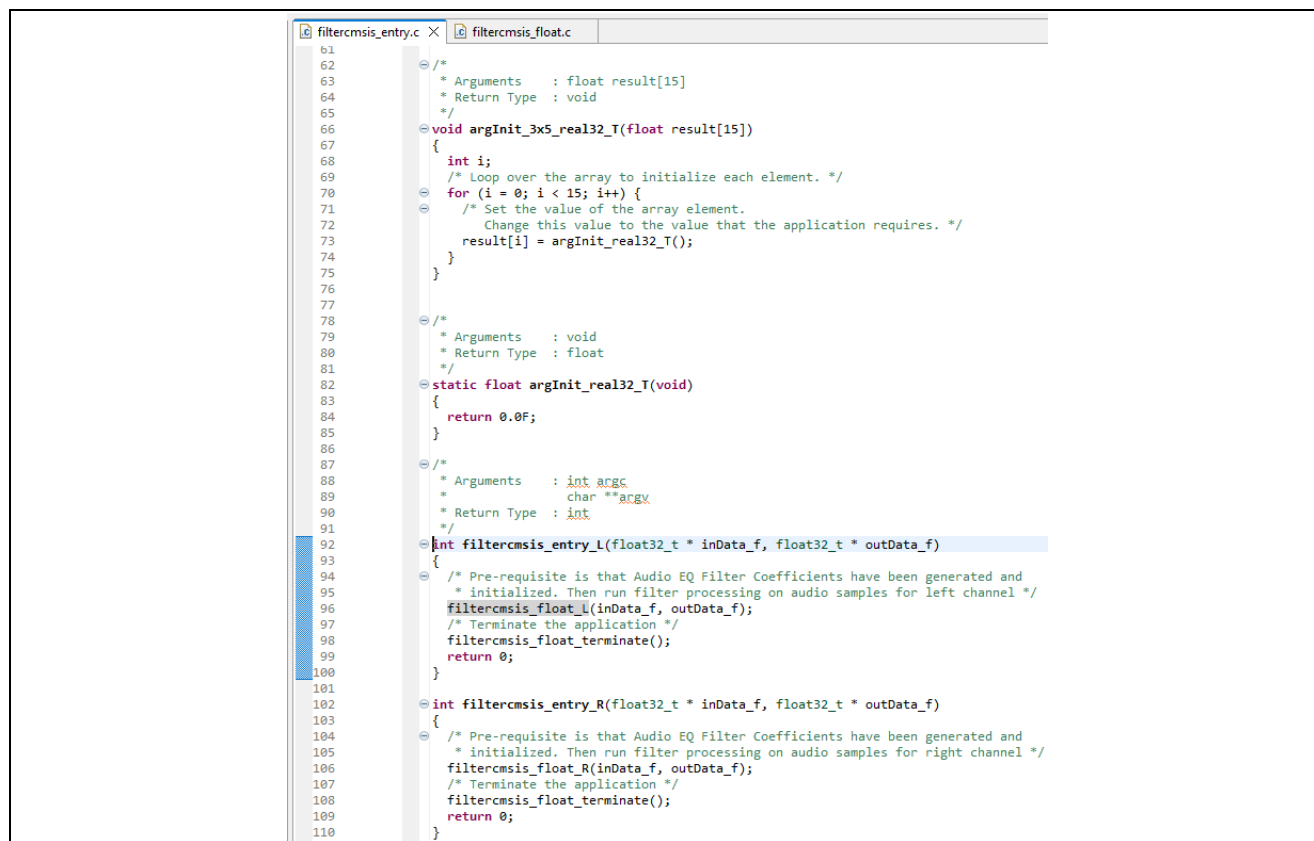


Figure 46. Audio EQ - Filter CMSIS Entry

In [Figure 46. Audio EQ - Filter CMSIS Entry](#), the functions `filtercmsis_entry_L(...)` and `filtercmsis_entry_R(...)` call `filtercmsis_float_L(input[32], output[32])` and `filtercmsis_float_R(input[32], output[32])` that use the CMSIS DSP function `arm_biquad_cascade_df2T_f32(...)` to implement the Biquad cascade used in the five band Audio EQ.

In order to improve DSP processing latency, it is required to make optimization modifications to `filtercmsis_float_L(...)` and `filtercmsis_float_R(...)`.

As observed in [Figure 47. Audio EQ – Filter CMSIS Float Function](#), the first modification is to remove the call to `memcpy(&U0[0], &input[0], 32U * sizeof(float))`, which is unnecessary since all memory buffers are initialized in `DSP_Process_Init()`.

The second modification is to the Biquad initialization, which is moved to `filtercmsis_biquad_init_L(num[15], den[10])` and `filtercmsis_biquad_init_R(num[15], den[10])`.

The modified functions are called in `DSP_Process_Init()`. See [Section 3.5 Modify Source Code for Best Performance on RA8 MCUs](#) for more information.

As observed in [Figure 47. Audio EQ – Filter CMSIS Float Function](#) and [Figure 48. CMSIS DSP Biquad Cascade Function](#), the last modifications are to move functions `filtercmsis_float_L(...)`, `filtercmsis_float_R(...)`, and `arm_biquad_cascade_df2T_f32(...)` to instruction tightly coupled memory (ITCM). For more information on how to use ITCM, see [Section 5.3.2 Improve Performance Using DTCM and ITCM](#).

```

38
39 /* Variable Definitions */
40 static bool filter_not_empty;
41 static dspcodegen_BiquadFilter filter_L;
42 static dspcodegen_BiquadFilter filter_R;
43
44 /* Function Definitions */
45
46 /* implement the function with DSP Toolbox filter with float32 data type
47 * and use Cortex M CMSIS DSP functions
48 *
49 * Arguments : const float input[1024]
50 *             const float num[15]
51 *             const float den[10]
52 *             float output[1024]
53 * Return Type : void
54 */
55 void __attribute__((section(".itcm_data"))) filtercmsis_float_L(const float input[32], float output[32]);
56
57 // I removed num,den since filtercmsis_biquad_init(num,den) called once to init biquad filter
58 void filtercmsis_float_L(const float input[32], float output[32])
59 {
60     /*
61     * I Modified the original Matlab codegen source code by moving biquad filter init to
62     * separate function filtercmsis_biquad_init(const float num[15],const float den[10])
63     */
64
65     /*
66     * Optimization: I modified to not use memcpy, instead pass input[0] to arm_biquad_cascade_df2T_f32(...)
67     */
68     // memcpy(&U0[0], &input[0], 32U * sizeof(float)); // not used, reduces latency
69     arm_biquad_cascade_df2T_f32(&filter_L.cSFunObject.S, &input[0], &output[0], 32U);
70 }
71
72 void __attribute__((section(".itcm_data"))) filtercmsis_float_R(const float input[32], float output[32]);
73
74 // I removed num,den since filtercmsis_biquad_init(num,den) called once to init biquad filter
75 void filtercmsis_float_R(const float input[32], float output[32])
76 {
77     /*
78     * I Modified the original Matlab codegen source code by moving biquad filter init to
79     * separate function filtercmsis_biquad_init(const float num[15],const float den[10])
80     */
81
82     /*
83     * Optimization: I modified to not use memcpy, instead pass input[0] to arm_biquad_cascade_df2T_f32(...)
84     */
85     // memcpy(&U0[0], &input[0], 32U * sizeof(float)); // not used, reduces latency
86     arm_biquad_cascade_df2T_f32(&filter_R.cSFunObject.S, &input[0], &output[0], 32U);
87 }

```

Figure 47. Audio EQ – Filter CMSIS Float Function

```

345
346 void __attribute__((section(".itcm_data"))) arm_biquad_cascade_df2T_f32(const arm_biquad_cascade_df2T_instance_f32 *
347                               const float32_t * pSrc, const float32_t * pDst,
348                               float32_t * pDst, uint32_t blockSize);
349
350 ARM_DSP_ATTRIBUTE void arm_biquad_cascade_df2T_f32(
351     const arm_biquad_cascade_df2T_instance_f32 * S,
352     const float32_t * pSrc,
353     const float32_t * pDst,
354     uint32_t blockSize)
355 {
356     const float32_t *pIn = pSrc; /* Source pointer */
357     float32_t *pOut = pDst; /* Destination pointer */
358     float32_t *pState = S->pState; /* State pointer */
359     const float32_t *pCoeffs = S->pCoeffs; /* Coefficient pointer */
360     float32_t acc1; /* Accumulator */
361     float32_t b0, b1, b2, a1, a2; /* Filter coefficients */
362     float32_t Xn1; /* Temporary input */
363     float32_t d1, d2; /* State variables */
364     uint32_t sample, stage = S->numStages; /* Loop counters */
365
366     do
367     {
368         /* Reading the coefficients */
369         b0 = pCoeffs[0];
370         b1 = pCoeffs[1];
371         b2 = pCoeffs[2];
372         a1 = pCoeffs[3];
373         a2 = pCoeffs[4];
374
375         /* Reading the state values */
376         d1 = pState[0];
377         d2 = pState[1];
378
379         pCoeffs += 5U;
380
381     #if defined (ARM_MATH_LOOPUNROLL)
382         /* Loop unrolling: Compute 16 outputs at a time */
383         sample = blockSize >> 4U;
384
385         while (sample > 0U) {

```

Figure 48. CMSIS DSP Biquad Cascade Function

4.3.4 CMSIS DSP Stack

The CMSIS DSP stack provides the CMSIS DSP functions required by the MATLAB-generated source code. There are no FSP configuration settings to make. As shown in [Figure 49. FSP Configuration - CMSIS DSP Stack](#), just add the **Arm CMSIS DSP Library Source** stack to the project.

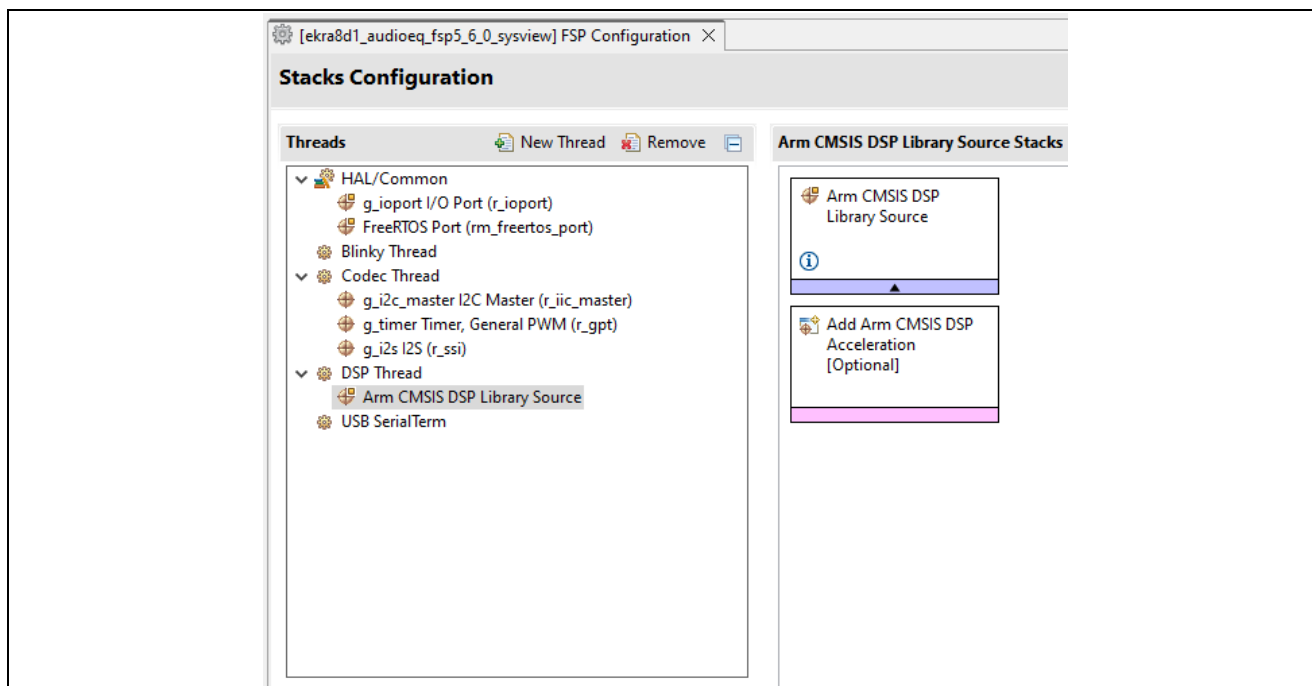


Figure 49. FSP Configuration - CMSIS DSP Stack

4.4 Command Line Interface Thread

The command line interface (CLI) is used to display the Audio EQ operation mode and processing latency. It is implemented with a Serial Terminal emulator over UART with Segger J-Link VCOM on-board through the Debug1 USB connection port.

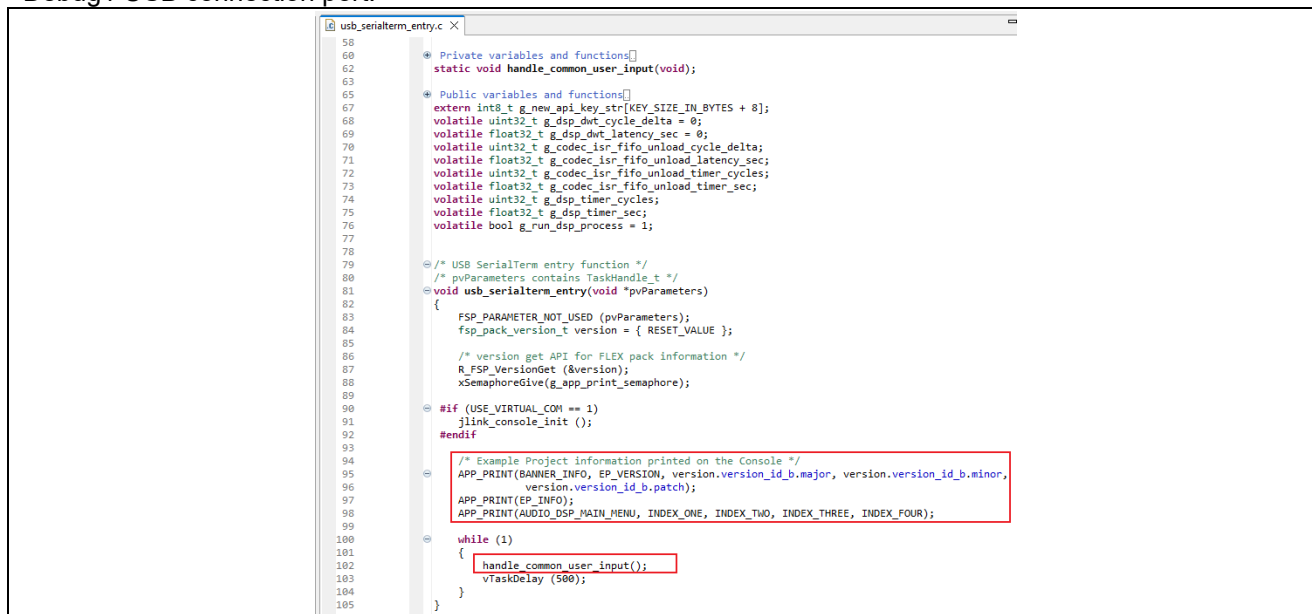


Figure 50. CLI - USB SerialTerm Thread

As shown in [Figure 50. CLI - USB SerialTerm Thread](#), the CLI is implemented with the USB SerialTerm thread. The J-Link console starts with VCOM, and then the info banner and menu are displayed. The user interacts with the CLI by selecting a menu option.

As shown in [Figure 51. CLI – Handle User Input Function](#), the user input handler is implemented with the function `handle_common_user_input()`.

The menu option handlers are provided to bypass processing, activate processing, display the Audio Codec handler latency, and display the DSP processing latency.

```

109  * @brief This function analyzes user input and activates the corresponding operation.
113  static void handle_common_user_input(void)
114  {
115      char perf_info[64] = {'\0'};
116
117      /* Get user input and process requested operation */
118      if (INPUT_STATUS_HAVE_INPUT == APP_GET_USER_INPUT(true))
119      {
120          int8_t key_index = (int8_t) g_new_api_key_str[INDEX_ZERO] - '0';
121          if ((key_index >= MENU_INDEX_MIN) && (key_index <= MENU_INDEX_MAX))
122          {
123              switch ( key_index )
124              {
125                  case INDEX_ONE:
126                      APP_PRINT("\r\n Bypass DSP \r\n");
127                      g_run_dsp_process = 0;
128                      break;
129                  case INDEX_TWO:
130                      APP_PRINT(CTRL_TEXT_BRIGHT_GREEN "\r\n Run DSP \r\n" CTRL_RESET);
131                      g_run_dsp_process = 1;
132                      break;
133                  case INDEX_THREE:
134                      if (UINT32_MAX > g_codec_isr_fifo_unload_cycle_delta)
135                      {
136                          // Debug session is running, so DWT value is available
137                          snprintf( perf_info, 64, "\r\nDWT Measured: Codec %u cycles, %3.4f uSec", g_codec_isr_fifo_unload_cycle_delta, (float)g_codec_isr_fifo_unload_cycle_delta / 1000);
138                          APP_PRINT(perf_info);
139                      }
140                      snprintf( perf_info, 64, "\r\nTimer Measured: Codec %u cycles, %3.4f uSec", g_codec_isr_fifo_unload_cycle_delta, (float)g_codec_isr_fifo_unload_cycle_delta / 1000);
141                      APP_PRINT(perf_info);
142                      break;
143                  case INDEX_FOUR:
144                      if (UINT32_MAX > g_dsp_dwt_cycle_delta)
145                      {
146                          // Debug session is running, so DWT value is available
147                          snprintf( perf_info, 64, CTRL_TEXT_BRIGHT_GREEN "\r\nDWT Measured: DSP %u cycles, %3.4f uSec", g_dsp_dwt_cycle_delta, (float)g_dsp_dwt_cycle_delta / 1000);
148                          APP_PRINT(perf_info);
149                      }
150                      snprintf( perf_info, 64, CTRL_TEXT_BRIGHT_GREEN "\r\nTimer Measured: DSP %u cycles, %3.4f uSec", g_dsp_dwt_cycle_delta, (float)g_dsp_dwt_cycle_delta / 1000);
151                      APP_PRINT(perf_info);
152                      break;
153                  default :
154                      APP_PRINT("\r\nInvalid Key Pressed\r\n");
155              }
156          }
157          else
158          {
159              APP_PRINT("\r\nInvalid Key Pressed\r\n");
160          }
161      }
162      /* End of function handle_common_user_input() */

```

Figure 51. CLI – Handle User Input Function

By default, the Audio EQ processing is active at program startup, and processing can be bypassed by selecting Menu 1 option. See [Section 2.3 Setup Command Line Interface](#) and [Section 2.4 Download and Operate Project](#) for more information.

4.5 LED Blinky Thread

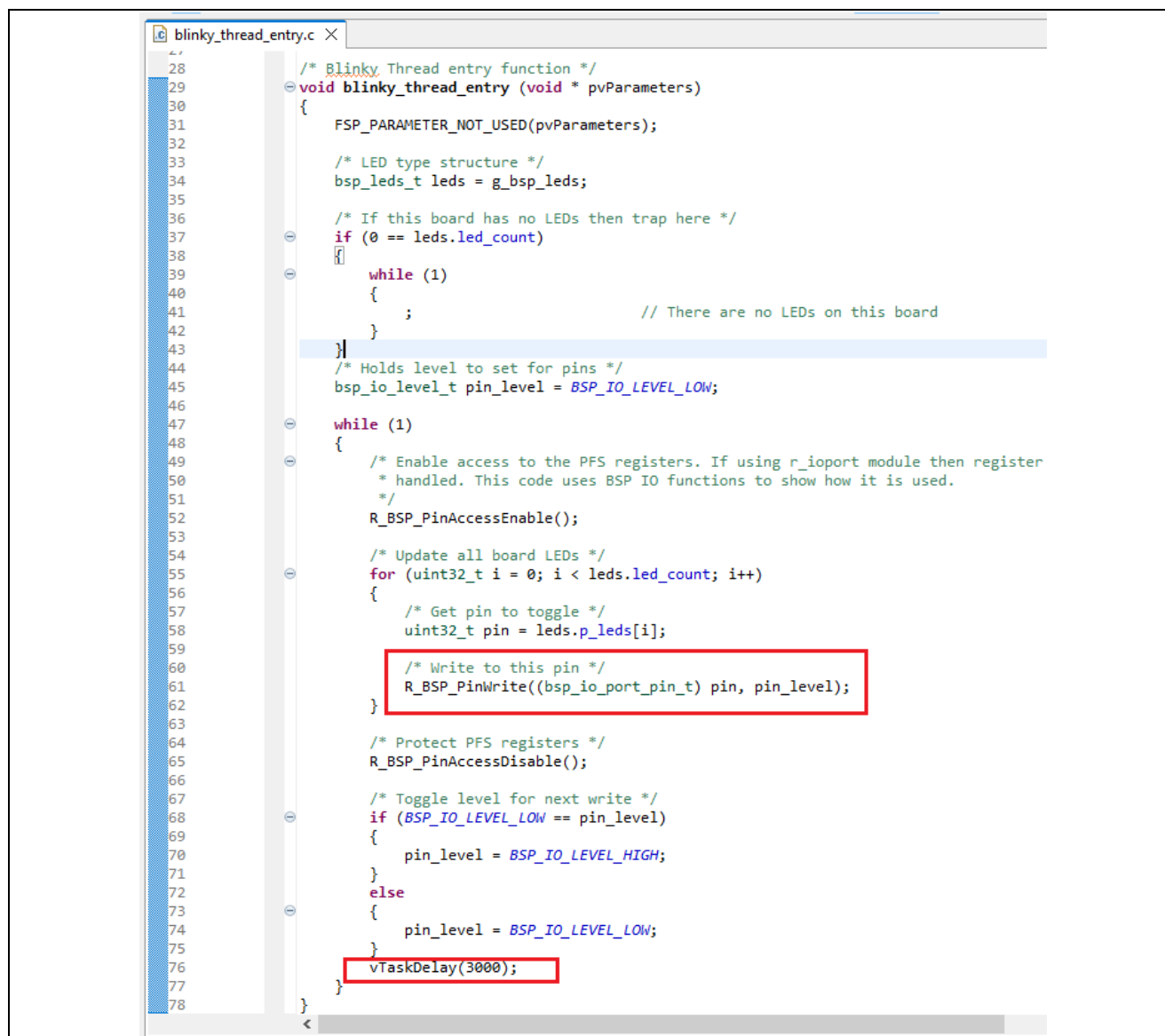


Figure 52. LED Blinky Thread

As observed in [Figure 52. LED Blinky Thread](#), the LED Blinky thread operates Blue (LED1), Green (LED2), and Red (LED3) LEDs on the EK-RA8D1 board. The purpose of the thread is to indicate the board and software are operating properly, with the lighted LEDs cycling on and off about every 3 seconds. The blinking LED indicates that the MCU is operating properly. If the blinking LED stops, it is an indication that the MCU has halted due to an unhandled exception. If this occurs, press the red Reset button Switch S3.

4.6 Using Segger SystemView

A system level view of the Audio EQ project can be obtained with Segger SystemView. The tool is integrated into the project and enabled by default to show the process threads, thread priorities, thread timing, and MCU utilization. To disable Segger SystemView in the project, see [Section 3.7 Changing Segger SystemView Support](#).

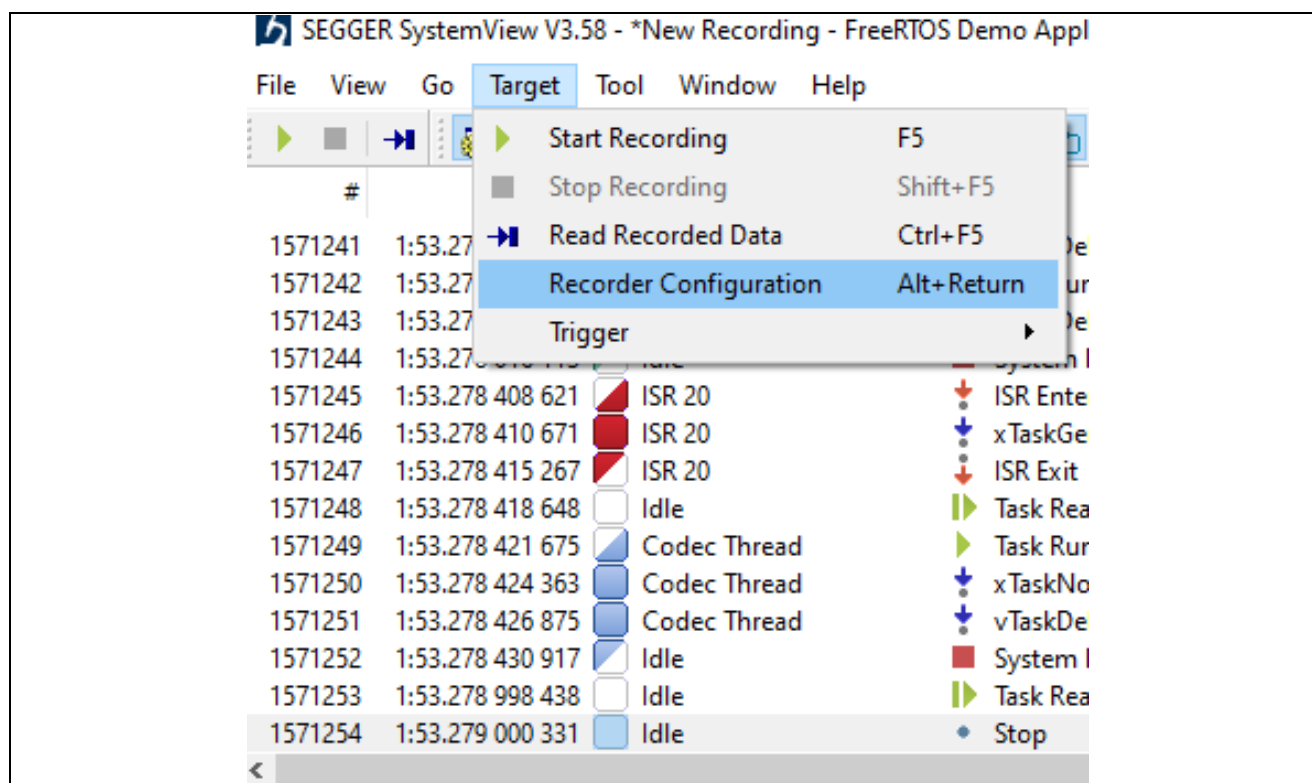


Figure 53. SystemView – Recorder Configuration

A recording of the system is obtained by configuring and starting a recording. In [Figure 53. SystemView – Recorder Configuration](#) and [Figure 54. SystemView – Recorder J-Link Settings](#), use **Target -> Recorder Configuration** to configure the J-Link settings. If the project is modified, then find the updated RTT Control Block Address in /Debug/ekra8d1_audioeq_fsp5_6_0_system.map file. Look for the address assigned to **_SEGGER_RTT**.

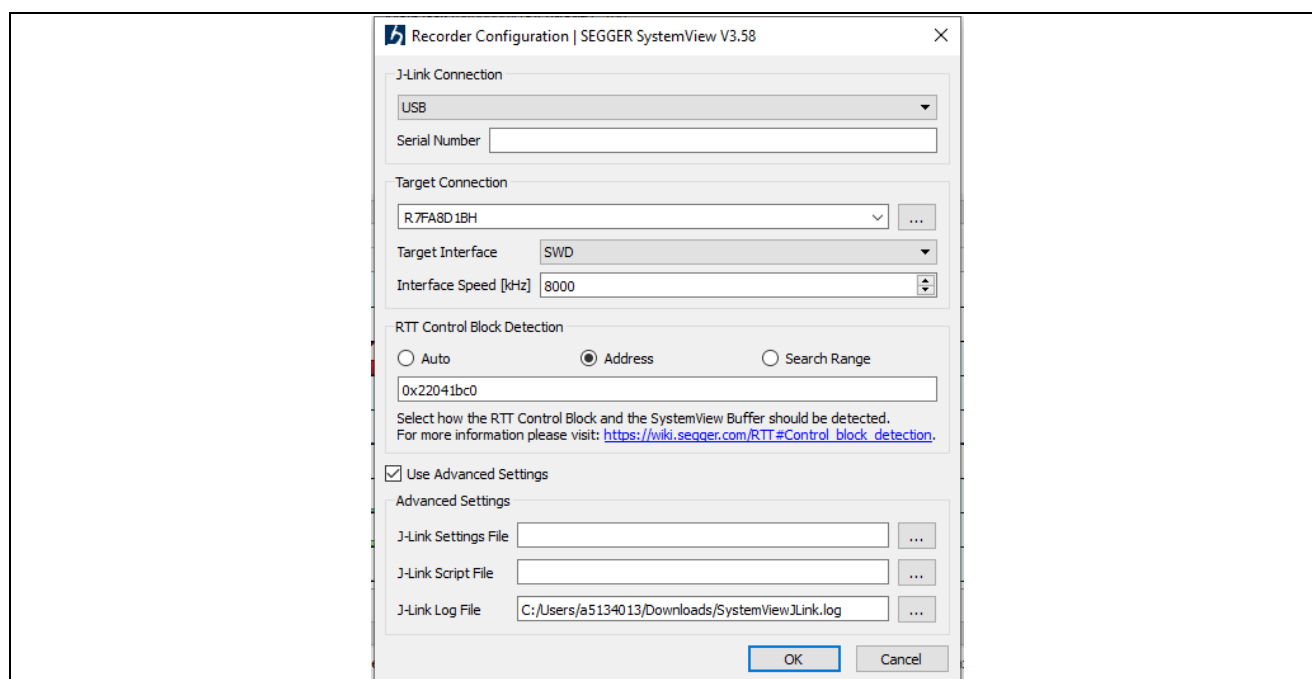


Figure 54. SystemView – Recorder J-Link Settings

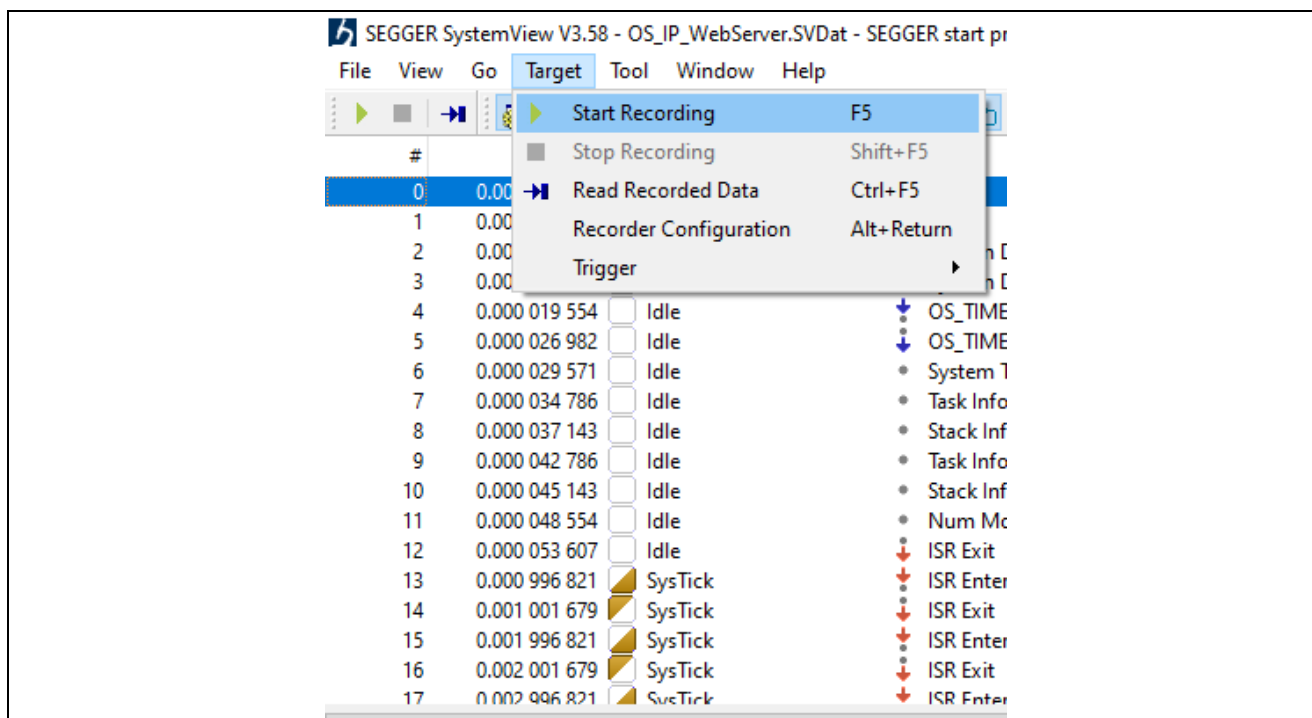


Figure 55. SystemView – Start Recording

Confirm the settings with SystemView and ensure that a debug session is running with e² studio before re-coding. As seen in [Figure 55. SystemView – Start Recording](#), next, start a recording with **Target -> Start Recording**. Capture two or more minutes of system activity. The recording results are shown in [Figure 56. SystemView – CPU Utilization](#).

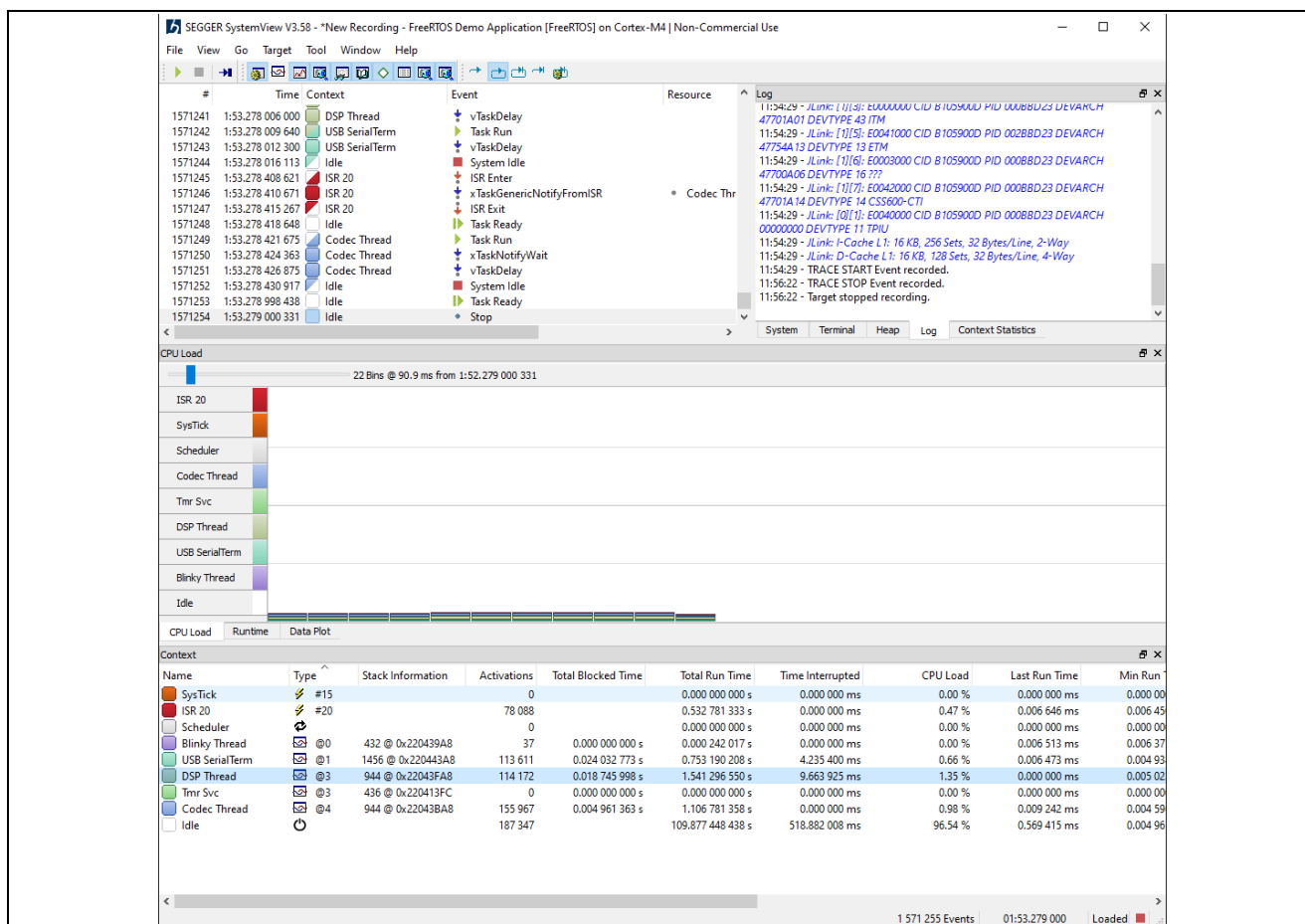


Figure 56. SystemView – CPU Utilization

The top window view shows the Audio EQ threads, ISRs, and Events with time stamps. The bottom window view shows the thread statistics that include the CPU load. The DSP, Codec, and Idle thread of the CPU load percent provides an indication of the MCU utilization.

In the Audio EQ project, the DSP thread is using 1.35% and Codec thread is using 0.98% of the CPU Load. From observing the idle task, the CPU load is 96.54%. More processing can be added to the DSP processing. The RA8D1 MCU is clearly not over-utilized in this project.

A good practice is measuring the system utilization before and after making changes to the project. It may be required to rebalance the priorities of the threads in the project if other threads and functionality are added to extend the Audio EQ project. An example of extended functionality would be adding a display for parameter control.

For other next steps to consider with this project, see [Section 6 Next Steps](#).

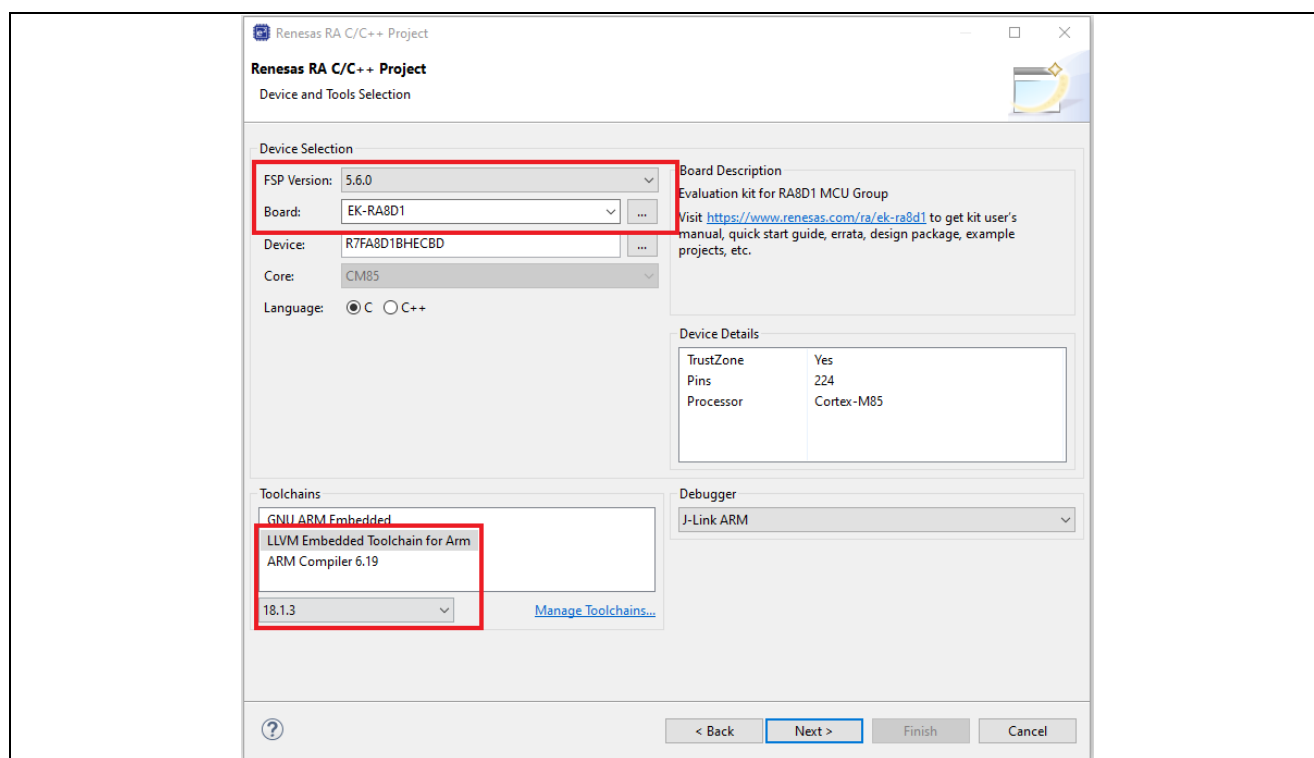
5. Optimizing Audio EQ Performance with Arm® Helium™ on RA8 MCU

5.1 Introduction to Arm® Helium™

Arm® Helium™ technology is the M-profile Vector Extension (MVE) for the Arm Cortex-M processor series. It is part of the ARMv8.1-M architecture and enables developers to realize a performance uplift for DSP and ML applications. Helium™ technology provides optimized performance using Single Instruction Multiple Data (SIMD) to perform the same operation simultaneously on multiple data. Refer to the R01AN7127EU Application Note [High Performance with RA8 MCU using Arm® Cortex®-M85 core with Helium™](#) for more details.

5.2 Arm® Helium™ Support in Renesas FSP and LLVM Toolchain

The LLVM Embedded Toolchain for ARM supports Helium™ instructions with the compiler settings by default. When generating a RA8D1 project using e² studio and Flexible Software Package (FSP), CPU settings and software settings are pre-optimized for Cortex®-M85 core and the CMSIS Helium™ support. See [Figure 57. Create a new EK-RA8D1 project with LLVM Toolchain using e² studio](#).

**Figure 57. Create a new EK-RA8D1 project with LLVM Toolchain using e² studio**

See [Table 3. LLVM Toolchain Settings](#) for the settings required with the LLVM Toolchain to get the best performance from the Cortex-M85 core with the CMSIS DSP extension.

Table 3. LLVM Toolchain Settings

Parameter	Default	Setting	Notes
-mfloat-abi	Hard	Hard	MCU has hardware support
-mfpu	Toolchain default	Toolchain default	Toolchain default
-march	Toolchain default	Toolchain default	MCU has DSP support
Optimization	-O3	-O3 or -Ofast	Optimize most or Optimize size
BSP/Main Stack Size	0x0	0x400	Size (bytes)
BSP/Heap Size	0x0	0x400	Size (bytes)

Depending on the memory resources required in your DSP project, the Main Stack and Heap size may need to be resized. In addition, consider testing your project with Optimization -O3 or -Ofast and measure processing latency before and after making the change. The optimization improvements are highly dependent on the CMSIS DSP functions that are used to implement the signal processing.

5.3 Improve DSP Performance

You can utilize Tightly Coupled Memory (TCM) and Cache together with Helium™ to achieve higher performance. Critical routines and data can be placed in TCM areas to ensure faster access. TCM does not use caches.

5.3.1 Improve Performance with Data Cache

Depending on the functionality of the DSP algorithm, the processing performance can be improved by enabling the cache in the FSP Configurator. Two example scenarios where Data Cache can improve performance include: a function that uses long loops to execute the same code repeatedly, or a function where data access is repeated and is sequential. To enable data cache with FSP Configurator, go to BSP -> Properties -> Settings -> RA8xx Family -> Cache settings -> Data cache. See [Figure 58. FSP Configurator Data Cache Settings](#).

Even though enabling the data cache can sometimes improve performance, it can cause concurrency and coherency issues. In general, it is good practice to enable the cache for applications that have repeated access to the same set of data for processing. We recommend measuring the latency of the DSP process before and after enabling the data cache to understand the impact of the cache on the processing latency and to check for any potential caching issues mentioned above.

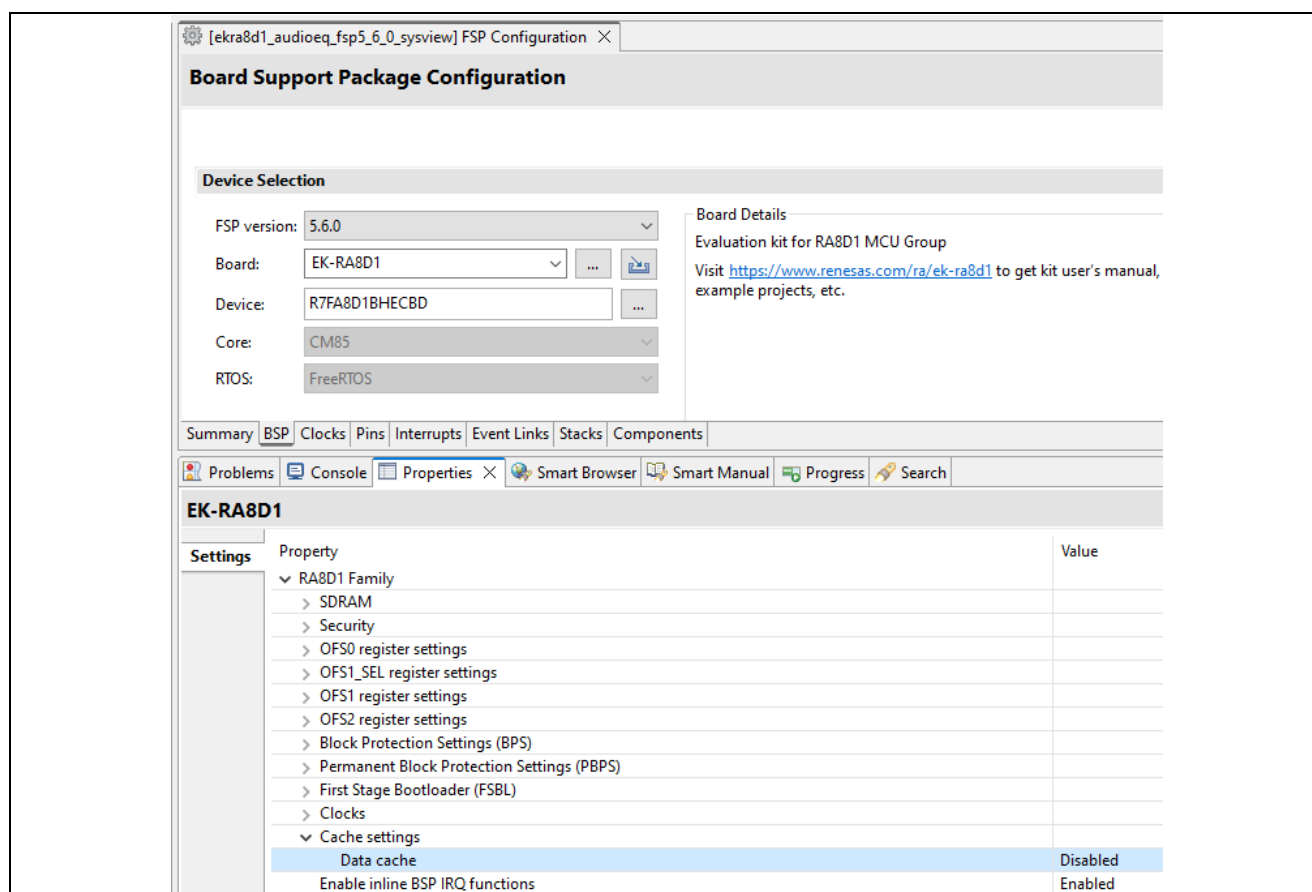


Figure 58. FSP Configurator Data Cache Settings

For the Audio EQ project, it was determined that no significant performance improvement could be made with data cache enabled. This is likely due to the processing of audio data streams that are continuously changing in the input and output circular buffers. However, using DTCM and ITCM did make a significant difference in performance. Continue to the next section to learn more about Tightly Coupled Memory (TCM).

5.3.2 Improve Performance Using DTCM and ITCM

The RA8 family of MCUs has 128 KB TCM memory that consists of 64 KB ITCM (Instruction TCM) and 64 KB DTCM (Data TCM). Note that accessing TCM is not available in CPU Deep Sleep mode, Software Standby mode, and Deep Software Standby mode. Refer to your specific MCU User's Manual for the TCM memory address areas.

Figure 59 shows ITCM and DTCM in the Local CPU Subsystem.

The FSP initializes both ITCM and DTCM areas by default. The linker script has defined sections for ITCM and DTCM areas, making it easy to utilize in user applications.

For the Audio EQ project with DTCM used for the DSP Processing subsystem local memory buffers, a change of 60.04% decrease in processing latency was measured.

For the Audio EQ, with the DSP functions placed in ITCM, no performance change was observed. Keep in mind any performance change is highly dependent on the application and the CMSIS-DSP functions that are used for processing. So, give it a try with your application, it may make an improvement in performance.

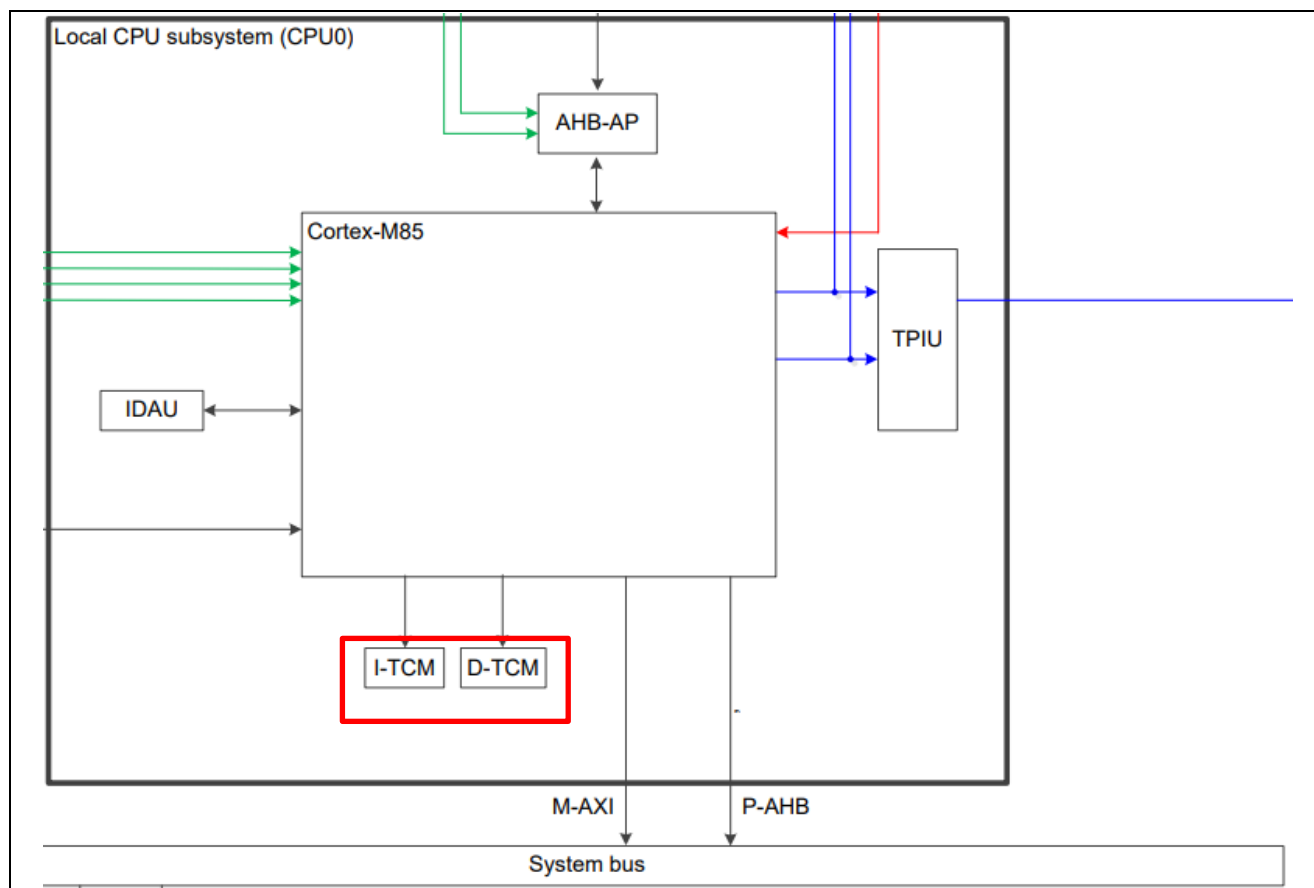


Figure 59. ITCM and DTCM in Local CPU Subsystem

You can place data in the DTCM section (.dtcm_data) in an FSP-based project using the `__attribute__` directive, as shown in [Figure 60. Placing Variables and Data in the DTCM Section](#).

```

dsp_processing.c
2
23
24
25 #include "dsp_processing.h"
26 #include "filtercmsis_entry.h"
27 #include "filtercmsis_float.h"
28 #include <genCoeffAudioEQ_float.h>
29
30 #define SCALE_FOR_HEADROOM
31
32 e_dsp_mode_t dsp_mode = DSP_MODE_MATLAB_EQ;
33
34 // Use Data Tightly Coupled Memory
35 static float32_t sourceBuffer_f[DSP_BUFFERSIZE] __attribute__((section(".dtcm_data"))) __attribute__((aligned(8)));
36 static float32_t targetBuffer_f[DSP_BUFFERSIZE] __attribute__((section(".dtcm_data"))) __attribute__((aligned(8)));
37 static float32_t rxLeft_f [DSP_BUFFERSIZE_HALF] __attribute__((section(".dtcm_data"))) __attribute__((aligned(8)));
38 static float32_t rxRight_f [DSP_BUFFERSIZE_HALF] __attribute__((section(".dtcm_data"))) __attribute__((aligned(8)));
39 static float32_t txLeft_f [DSP_BUFFERSIZE_HALF] __attribute__((section(".dtcm_data"))) __attribute__((aligned(8)));
40 static float32_t txRight_f [DSP_BUFFERSIZE_HALF] __attribute__((section(".dtcm_data"))) __attribute__((aligned(8)));

```

Figure 60. Placing Variables and Data in the DTCM Section

The above data placement in the DTCM section can be confirmed using the memory map (*.map) file generated by the compiler, as shown in [Figure 61. Memory Map - Data Placed in DTCM Area](#)

ekra8d1_audioeq_fsp5_6_0_sysview.map					
1185	22000289	22000289	0	1	. = .
1186	22000289	22000289	0	1	__dtcm_data_pre_location = LOADADDR (.tdata) + SIZEOF (.tdata)
1187	22000289	22000289	0	1	DTCM BYTE ALIGN_VALUE = (__DTCM_LENGTH > 0) ? 16 : 4
1188	20000000	200fb60	478	16	.dtcm_data
1189	20000000	200fb60	0	1	__tz_DTCM_S = ABSOLUTE (__DTCM_START)
1190	20000000	200fb60	0	1	__dtcm_data_start = .
1191	20000000	200fb60	478	8	./src/dsp_processing.o:(.dtcm_data)
1192	20000000	200fb60	100	1	sourceBuffer_f
1193	20000100	200fc60	100	1	targetBuffer_f
1194	20000200	200fd60	80	1	rxLeft_f
1195	20000280	200fde0	80	1	rxRight_f
1196	20000300	200fee0	80	1	txLeft_f
1197	20000380	200fee0	80	1	txRight_f

Figure 61. Memory Map - Data Placed in DTCM Area

You can also place program instructions in the ITCM section (.itcm_data) using the `__attribute__` directive.

Figure 62. Place a DSP Function in the ITCM Section shows an example of a modification to place the function

`\ra\arm\CMSIS-DSP\Include\dsp\filtering_functions.h\arm_biquad_cascade_df2T_f32` in the ITCM area.

```

1216  /**
1217  * @brief Processing function for the floating-point transposed direct form II Biquad cascade filter.
1218  * @param[in] S points to an instance of the filter data structure.
1219  * @param[in] pSrc points to the block of input data.
1220  * @param[out] pDst points to the block of output data
1221  * @param[in] blockSize number of samples to process.
1222  */
1223  void __attribute__((section(".itcm_data"))) arm_biquad_cascade_df2T_f32(
1224  const arm_biquad_cascade_df2T_instance_f32 * S,
1225  const float32_t * pSrc,
1226  float32_t * pDst,
1227  uint32_t blockSize);

```

Figure 62. Place a DSP Function in the ITCM Section

You can confirm code placement using the memory map (*.map) file generated by the compiler, as shown in Figure 63. Memory Map - DSP Function in ITCM Area.

ekra8d1_audioeq_fsp5_6_0_sysview.map

1093	0	200f580	4c8	16	.itcm_data
1094	0	200f580	0	1	__tz_ITCM_S = ABSOLUTE (__ITCM_START)
1095	0	200f580	0	1	__itcm_data_start = .
1096	0	200f580	1	1	BYTE (0xFF)
1097	2	200f582	30	2	./src/MatlabAudioEQ/filtercmsis_float.o:(.itcm_data)
1098	2	200f582	0	1	\$t.0
1099	3	200f583	18	1	filtercmsis_float_L
1100	1b	200f59b	18	1	filtercmsis_float_R
1101	32	200f5b2	36	2	./src/ArmIIRHpExample/arm_fir_hpf_example_f32.o:(.itcm_data)
1102	32	200f5b2	0	1	\$t.1
1103	33	200f5b3	1a	1	main_arm_hpf_iir_example_f32_L
1104	4d	200f5cd	1c	1	main_arm_hpf_iir_example_f32_R
1105	68	200f5e8	13c	4	./src/dsp_processing.o:(.itcm_data)
1106	68	200f5e8	0	1	\$t.2
1107	69	200f5e9	13c	1	DSP_Process_Data
1108	1a0	200f720	0	1	\$d.3
1109	1a4	200f724	2cc	2	./ra/arm/CMSIS-DSP/Source/FilteringFunctions/arm_biquad_cascade_df2T_f32.o:(.itcm_data)
1110	1a4	200f724	0	1	\$t.0
1111	1a5	200f725	2cc	1	arm_biquad_cascade_df2T_f32
1112	470	200f9f0	18	2	./ra/arm/CMSIS-DSP/Source/FilteringFunctions/arm_biquad_cascade_df2T_init_f32.o:(.itcm_data)
1113	470	200f9f0	0	1	\$t.0
1114	471	200f9f1	18	1	arm_biquad_cascade_df2T_init_f32

Figure 63. Memory Map - DSP Function in ITCM Area

If the CMSIS DSP Library source code is updated with a newer version of FSP, then you must add the `__attribute__` directive to any affected CMSIS DSP Functions again, as shown in Figure 62. Place a DSP Function in the ITCM Section.

5.3.3 Improve Performance Using ARM CMSIS DSP Optimizations

You can improve performance further with the CMSIS DSP Library by using compiler preprocessor macros (Defines -D) to select additional function implementation optimizations when the library is built, as shown in Figure 64. Compiler Optimization - Macro Defines for CMSIS DSP and Figure 65. Assembler Optimization - Macro Defines for CMSIS DSP.

Examples of preprocessor macros are:

- ARM_MATH_LOOPUNROLL
- ARM_MATH_HELIUM
- ARM_MATH_MVEF

In the Audio EQ project, the ARM_MATH_LOOPUNROLL optimization made the most significant improvement in DSP processing with a 24.32% decrease in latency.

The ARM_MATH_HELIUM optimization made an improvement with a 6% decrease in latency.

The ARM_MATH_MVEF optimization made a very small improvement with a 0.09% decrease in latency.

Under certain use cases, the optimizations provide further latency improvements but are MCU (ARM core) and CMSIS DSP function-dependent. It is recommended to use the latency measurement in this project to take a measurement before and after enabling an optimization option. Not all MCUs support the possible CMSIS function optimizations. Consult the CMSIS DSP documentation for specific details on which support is available for your MCU and the CMSIS DSP functions in use.

[Arm Software GitHub CMSIS_5 DSP](#)

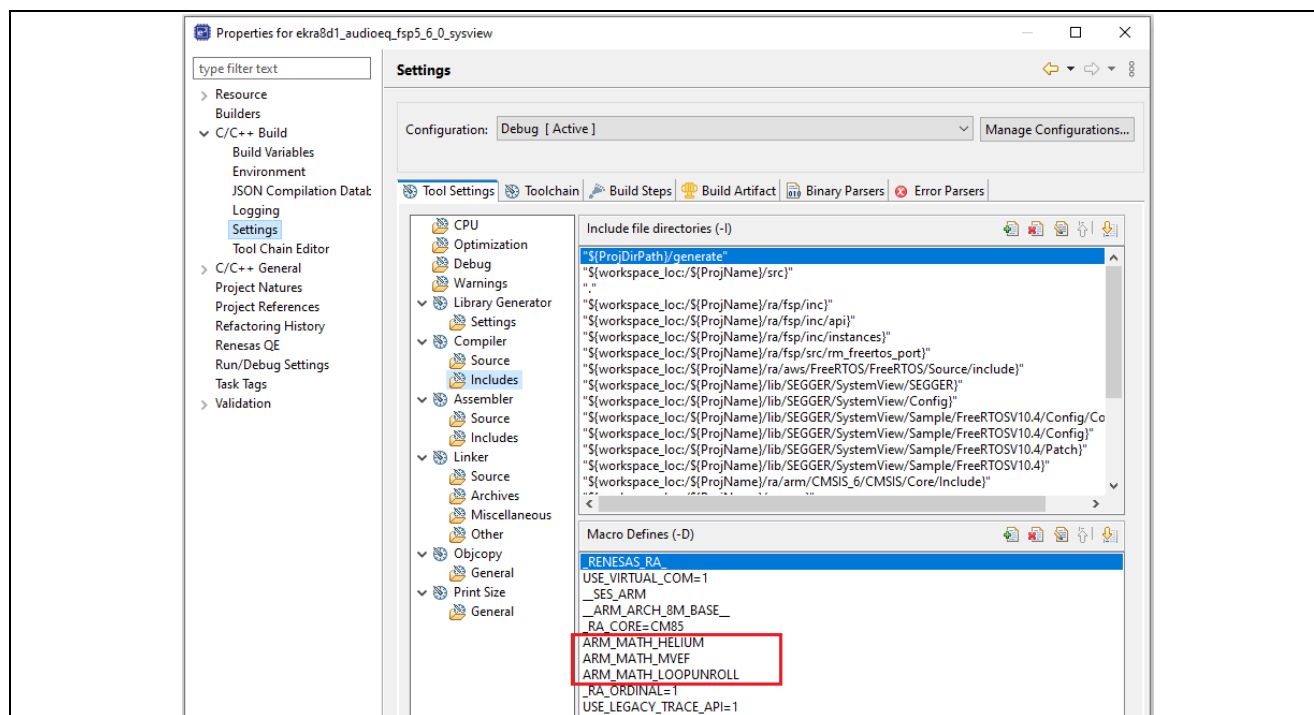


Figure 64. Compiler Optimization - Macro Defines for CMSIS DSP

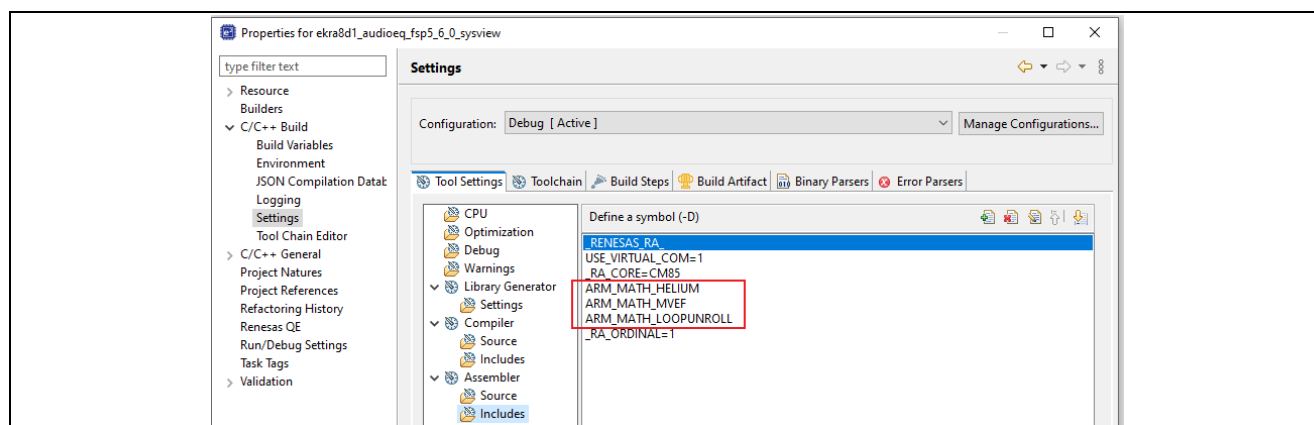


Figure 65. Assembler Optimization - Macro Defines for CMSIS DSP

6. Next Steps

After following the design workflow presented in this application note, the next steps that should be considered for your project are:

1. Replace the Audio EQ with your DSP filters or algorithms and observe changes in system bandwidth and processing latency.
2. Analyze the impact of the DMAC/DTC on system bandwidth and latency by removing its usage from the reference project.
3. Consider processing multiple audio streams using multiple threads and then synchronize the results at the end. Consider off-loading the data movement performed with the CPU by using DMAC/DTC transfers for audio stream data to and from memory.
4. Consider using DSP processing for multi-channel audio systems (for example, Dolby 5.1 Surround or Dolby Atmos).
5. Drive a display in parallel to control the DSP filter parameters.
6. Use a different audio codec or switch to an ADC/DAC for cost savings.
7. Move the audio circular buffers from internal SRAM to external SDRAM or OSPI RAM and evaluate the performance.
8. Consider Firmware OTA techniques for updating audio algorithms with a cloud server.

7. References

[RA8D1 MCU Hardware User's Manual](#)

[FSP Software User's Manual](#)

[ARM Cortex M85 CPU Technical Reference Manual](#)

[MATLAB Documentation](#)

Arm Helium Technology M-Profile Vector Extension (MVE) for Arm Cortex-M Processors Reference Book - Jon Marsh - ISBN: 978-1-911531-23-4.

8. 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	renesas.com/ra
RA Product Support Forum	renesas.com/ra/forum
RA Flexible Software Package	renesas.com/fsp
Renesas Support	renesas.com/support

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Feb.24.25	—	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 includes its directly or indirectly controlled subsidiaries.

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

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/

(Rev.5.0-1 October 2020)