

Renesas RZ Family

Arm® DSP Examples

Introduction

Arm® DSP Examples describes how to use Arm® CMSIS-DSP example projects ported to Renesas Arm® Cortex-R52-based MPUs with DSP extension and a Floating Point Unit (FPU). It also explains how to import, configure, build, and run the DSP examples and demonstrates how to measure the processing time of a selected DSP function.

In addition, this application note describes memory optimization for supported Transform Functions using a customized library based on Arm CMSIS-DSP Version 1.16.2. The customization provides:

- Compile-time selection of lookup tables (twiddle factors and bit-reversal tables) for each supported API function and FFT length.
- Application-level configuration through macros to include only the lookup tables required for the selected FFT lengths.

Refer to section “Memory Optimization for Transform Functions” for more details.

All information regarding Arm CMSIS-DSP can be found at the following link:

[CMSIS-DSP: Overview](#)

Required Resources

Hardware

- [Renesas Starter Kit+ for RZ/N2L](#)
- [Renesas Starter Kit+ for RZ/T2L](#)
- [Renesas Starter Kit+ for RZ/T2M](#)
- Evaluation Board Kit for RZ/T2N

Development Tools and Software

IDE:

- e² studio IDE version 2026-07
- IAR Embedded Workbench (EWARM) IDE version 9.60.3

Tool:

- FSP Smart Configurator (FSP SC)
- RZ Flexible Software Package (FSP) v4.2.0
- GNU Arm Embedded Toolchain version 13.3.Rel1
- SEGGER J-Link® USB driver and RTT Viewer version 9.44

The above software components are bundled in a downloadable platform installer available on the FSP webpage at renesas.com/rz/fsp or [Release v4.2.0 · renesas/rz-fsp](#) and the SEGGER webpage at segger.com/j-link/rtt-viewer.

Target Devices

This application note focuses on the RZ/N2L, RZ/T2L, RZ/T2M, and RZ/T2N groups.

Contents

1. Arm® CMSIS-DSP	3
2. Arm® CMSIS-DSP Examples	4
2.1 Building and Running a CMSIS-DSP Example	5
2.2 Project Settings	8
3. Performance Measurement	9
3.1 Enable Performance Measurement	9
3.1.1 Using LED Function Code to Measure DSP Processing Interval	9
3.2 Displaying Example Status	10
4. Memory Optimization for Transform Functions	12
4.1 Customized CMSIS-DSP Library	12
4.2 Configuration	13
4.2.1 How to Configure	13
4.2.2 Memory Usage	17
4.3 Transform Function Reference Information	18
4.3.1 Initialization Function	18
4.3.2 Number of Points	18
4.3.3 Macro	19
4.3.4 Memory Usage	20
5. Memory Optimization Results for the Arm® DSP Examples	24
5.1 Configure the Application Project	24
5.2 Results for the Updated Example	26
5.2.1 Example Status	26
5.2.2 Memory Usage Comparison	27
6. Website and Support	28
Revision History	29

1. Arm® CMSIS-DSP

The CMSIS-DSP software library is a suite of common signal-processing functions used on Cortex-R devices. The library is organized into the following functional categories:

- Basic math functions
- Fast math functions
- Complex math functions
- Filtering functions
- Matrix functions
- Transform functions
- Motor control functions
- Statistical functions
- Support functions
- Interpolation functions
- Support Vector Machine (SVM) functions
- Bayes classifier functions
- Distance functions
- Quaternion functions
- Window functions

The library generally has separate functions for operating on 8-bit integers, 16-bit integers, 32-bit integers, and 32-bit floating-point values. It is imported into the project from the Arm® CMSIS-DSP package. The CMSIS-DSP library used in this project is a customized version based on Arm CMSIS-DSP Version 1.16.2. The memory-optimization modifications described in Chapter 4 have already been applied and are included in the project source code. Users can use the library directly without making any additional modification.

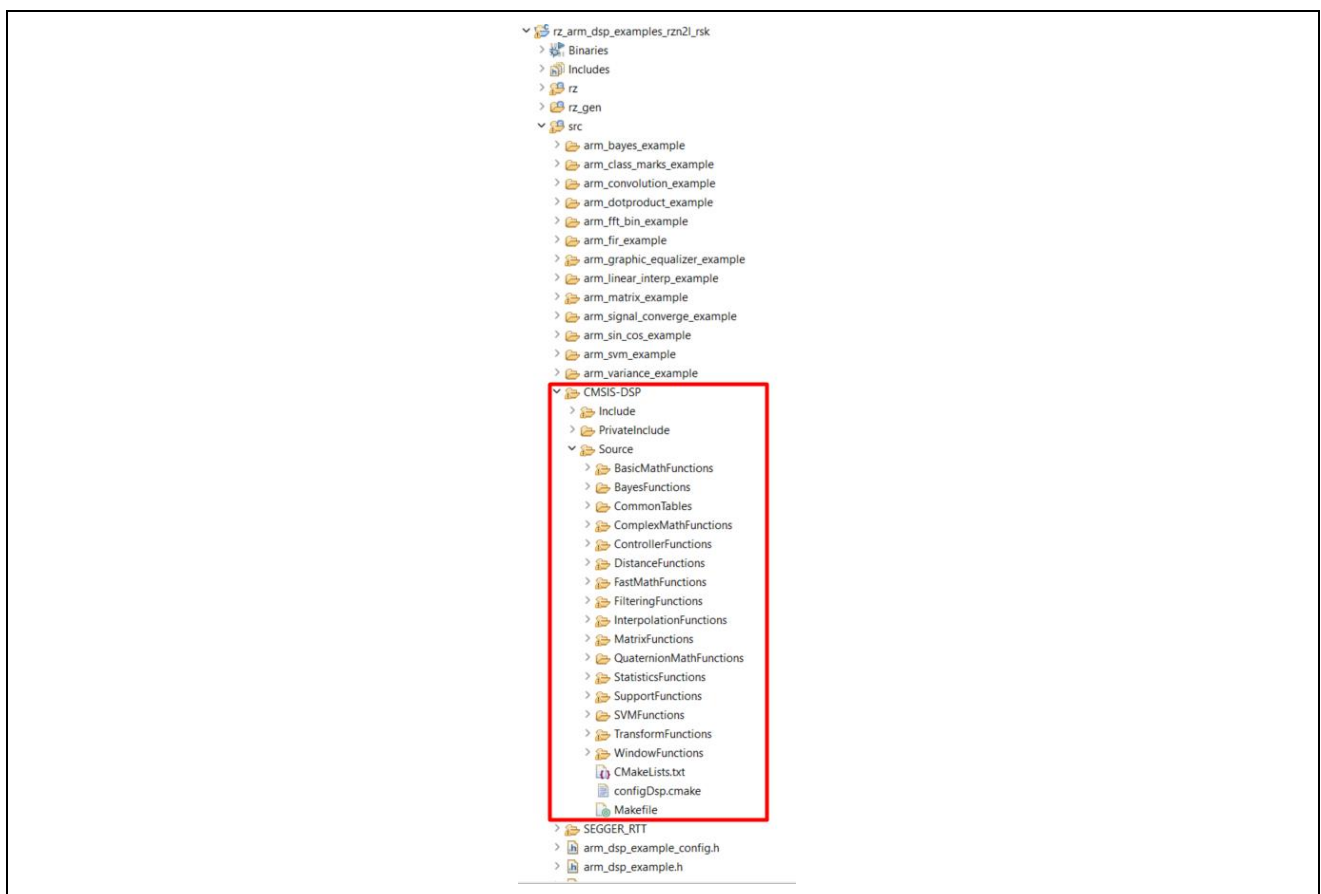


Figure 1. CMSIS-DSP Library Source in Project

2. Arm® CMSIS-DSP Examples

This example project includes multiple CMSIS-DSP examples. The following examples are supported by this project:

- Sin Cos example
- Signal Convergence example
- Linear Interpolate example
- Graphic Audio Equalizer example
- Lowpass Filter example
- Frequency Bin example
- Dot Product example
- Convolution example
- Variance example
- Matrix example
- Class Marks example
- Bayes example
- SVM example

Details on importing the example project are provided below:

- IDE e² studio: [here](#)
- IDE EWARM: Users need to follow these instructions first (to generate files):
 1. Open the project in EWARM ([here](#))
 2. Re-configuring project with FSP SC ([here](#))
 3. Open project and generate the source files by FSP SC ([here](#)).

Details of the examples listed above can be found [here](#).

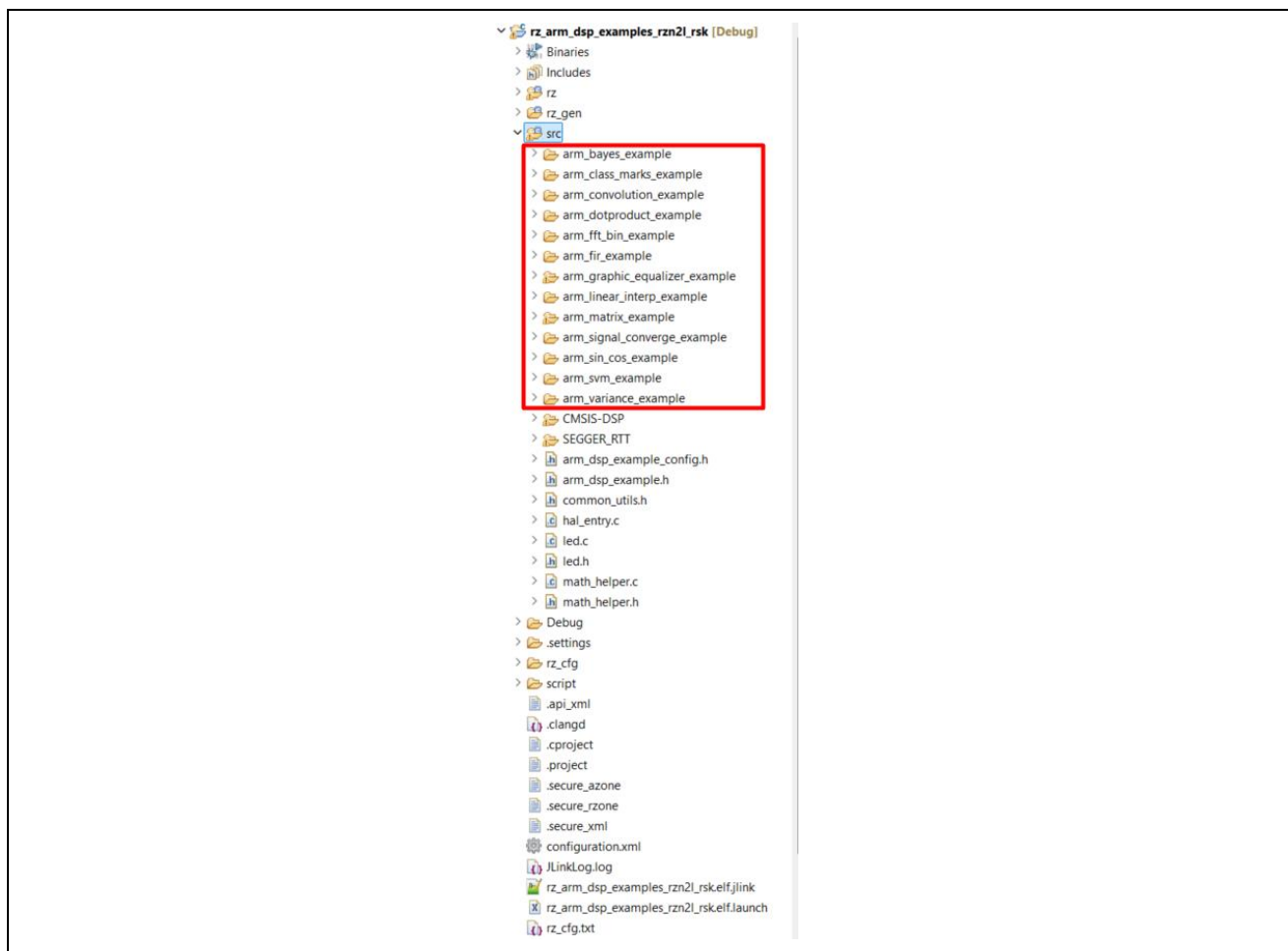


Figure 2. CMSIS-DSP Example Source Folders in the imported e² studio Project

Each CMSIS-DSP example is enabled using its macro definition in arm_dsp_example_config.h.

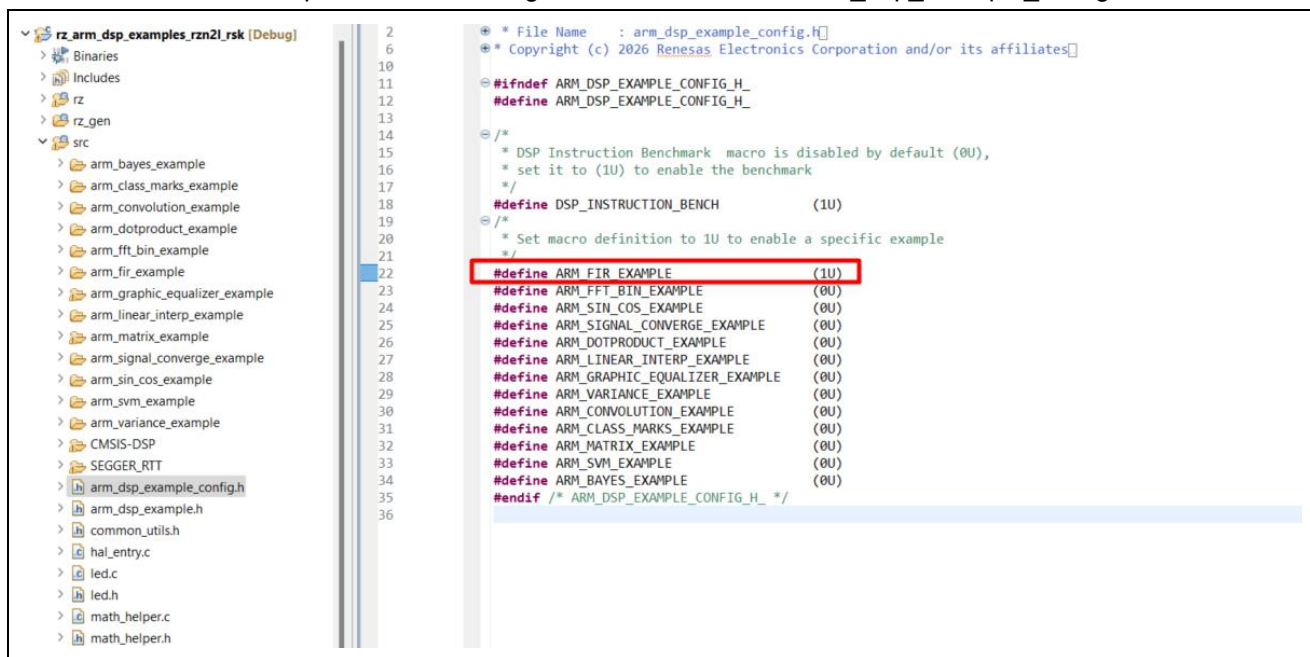


Figure 3. Set MACRO for CMSIS-DSP Example in arm_dsp_example_config.h

Note: Only one CMSIS-DSP example can be enabled at a time; set its corresponding macro to 1U and all other example macros to 0U, as enabling multiple examples will cause build errors.

2.1 Building and Running a CMSIS-DSP Example

After importing the example project, enable the example you want to run in arm_dsp_example_config.h.

You can then build, download, and run the project. The screenshots that show in Figure 4 and Figure 5 is an example of running the Lowpass Filter example.

After downloading and running the project, wait a few seconds, then click the **Suspend** button (e² studio) or **Break** button (EWARM) to pause execution.

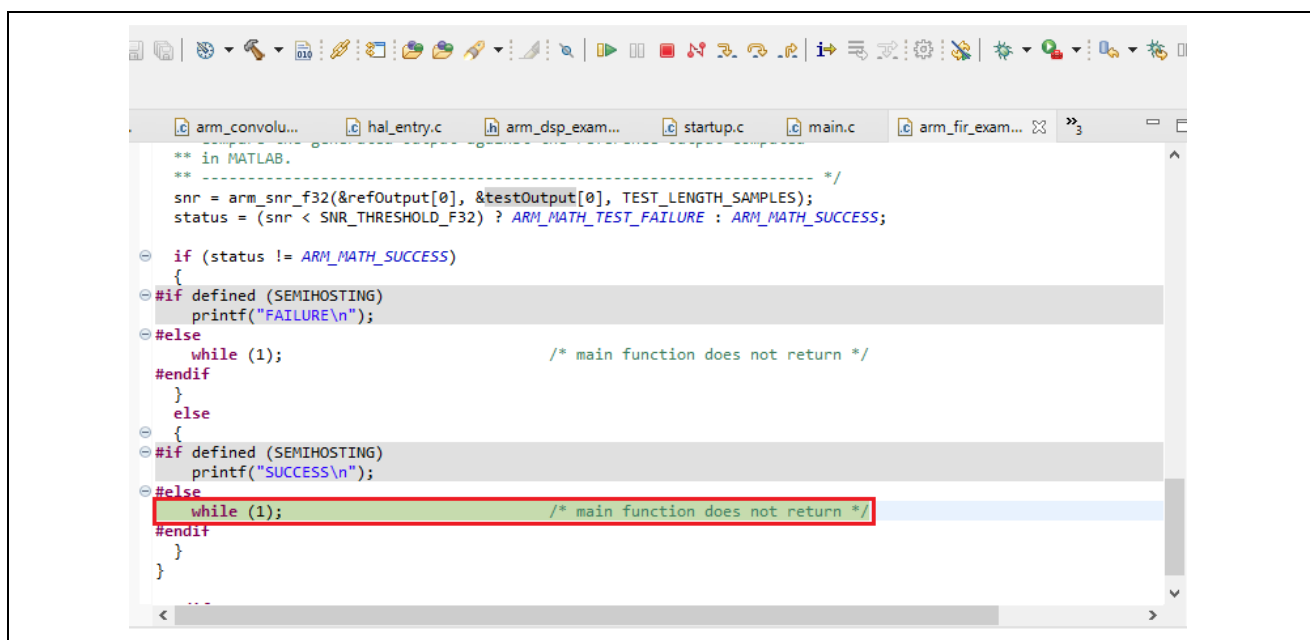


Figure 4. Successful Completion of the Lowpass Filter Example in e² studio

```

main_arm_fir_example_f32.c
main_arm_fir_example_f32()
{
    arm_fir_init_f32(85, NUM_TAPS, (float32_t *)&firCoeffs32[0], &firStateF32[0], blockSize);
    #if DSP_INSTRUCTION_BENCH //Performance Benchmark
    R_LED_LED10N();
    #endif

    /* -----
    ** Call the FIR process function for every blockSize samples
    ** ----- */
    for(i=0; i < numBlocks; i++)
    {
        arm_fir_f32(85, inputF32 + (i * blockSize), outputF32 + (i * blockSize), blockSize);
    }

    #if DSP_INSTRUCTION_BENCH //Performance Benchmark
    R_LED_LED10FF();
    #endif

    /* -----
    ** Compare the generated output against the reference output computed
    ** in MATLAB.
    ** ----- */
    snr = arm_snr_f32(&refOutput[0], &testOutput[0], TEST_LENGTH_SAMPLES);
    status = (snr < SNR_THRESHOLD_F32) ? ARM_MATH_TEST_FAILURE : ARM_MATH_SUCCESS;

    if (status != ARM_MATH_SUCCESS)
    {
        #if defined (SEMIHOSTING)
        printf("FAILURE\n");
        #else
        /* Print result */
        APP_PRINT("FAILURE\n");
        while (1); /* main function does not return */
        #endif
    }
    else
    {
        #if defined (SEMIHOSTING)
        printf("SUCCESS\n");
        #else
        /* Print result */
        APP_PRINT("ARM_FIR_EXAMPLE: SUCCESS\n");
        while (1); /* main function does not return */
        #endif
    }
}

```

Figure 5. Successful Completion of the Lowpass Filter Example in EWARM

The input and output buffer waveforms can be used to verify the result visually. Add the testOutput buffer to the **Memory** view as shown in Figure 6, then add a new **Fixed Floating Point** rendering for the same buffer.

Address	0 - 3	4 - 7	8 - B	C - F
00000000000044E0	0.867901	0.7952493	0.0	0.59246594
00000000000044F0	-0.09473435	0.19134171	1.0	0.4174197
0000000000004500	0.35355338	1.2552931	0.8660254	0.46193975
0000000000004510	1.3194792	1.1827866	0.5	1.1827866
0000000000004520	1.3194792	0.46193975	0.8660254	1.2552931
0000000000004530	0.35355338	0.4174197	1.0	0.19134171
0000000000004540	-0.09473435	0.59246594	-0.0	-0.59246594
0000000000004550	0.09473435	-0.19134171	-1.0	-0.4174197
0000000000004560	-0.35355338	-1.2552931	-0.8660254	-0.46193975
0000000000004570	-1.3194792	-1.1827866	-0.5	-1.1827866
0000000000004580	-1.3194792	-0.46193975	-0.8660254	-1.2552931
0000000000004590	-0.35355338	-0.4174197	-1.0	-0.19134171
00000000000045A0	0.09473435	-0.59246594	0.0	0.59246594
00000000000045B0	-0.09473435	0.19134171	1.0	0.4174197
00000000000045C0	0.35355338	1.2552931	0.8660254	0.46193975
00000000000045D0	1.3194792	1.1827866	0.5	1.1827866
00000000000045E0	1.3194792	0.46193975	0.8660254	1.2552931
00000000000045F0	0.35355338	0.4174197	1.0	0.19134171
0000000000004600	-0.09473435	0.59246594	0.0	-0.59246594
0000000000004610	0.09473435	-0.19134171	-1.0	-0.4174197
0000000000004620	-0.35355338	-1.2552931	-0.8660254	-0.46193975
0000000000004630	-1.3194792	-1.1827866	-0.5	-1.1827866
0000000000004640	-1.3194792	-0.46193975	-0.8660254	-1.2552931
0000000000004650	-0.35355338	-0.4174197	-1.0	-0.19134171
0000000000004660	0.09473435	-0.59246594	0.0	0.59246594
0000000000004670	-0.09473435	0.19134171	1.0	0.4174197
0000000000004680	0.35355338	1.2552931	0.8660254	0.46193975
0000000000004690	1.3194792	1.1827866	0.5	1.1827866
00000000000046A0	1.3194792	0.46193975	0.8660254	1.2552931
00000000000046B0	0.35355338	0.4174197	1.0	0.19134171

Figure 6. testInput Buffer in Floating Point format of Lowpass Filter Example

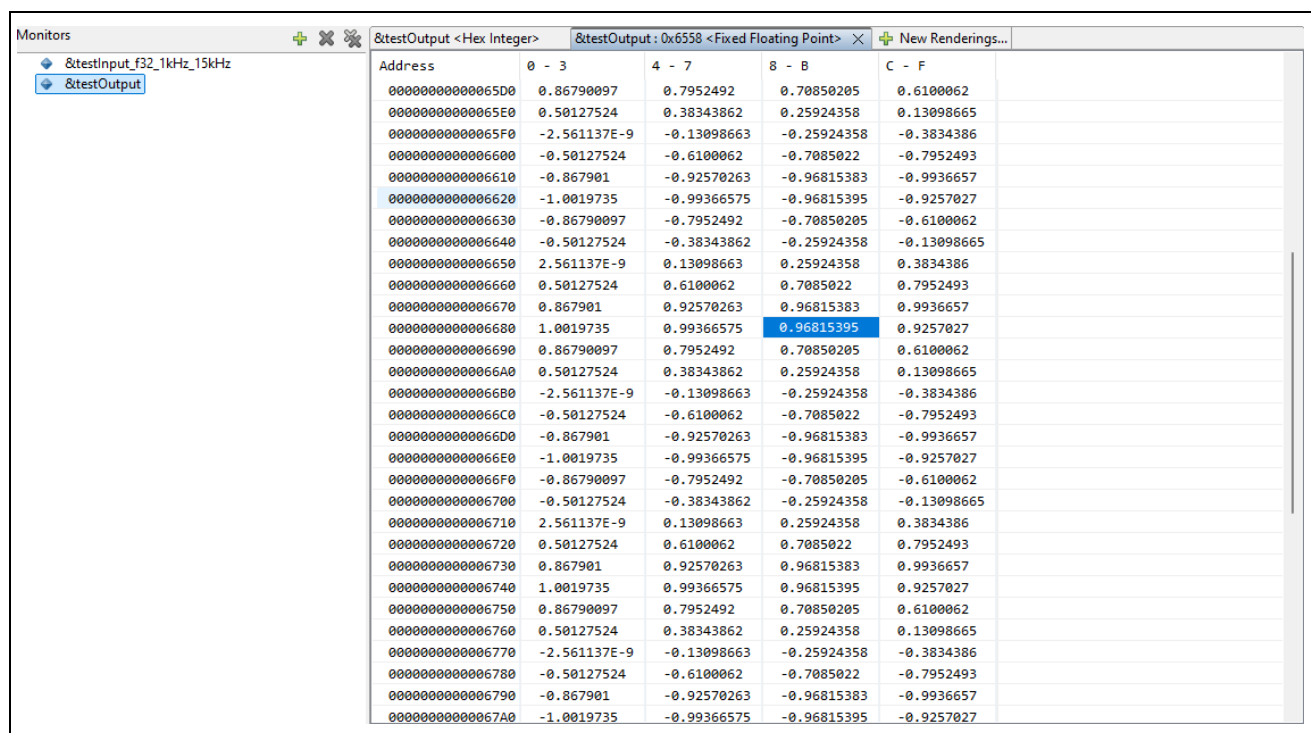


Figure 7. testOutput Buffer in Floating Point format of Lowpass Filter Example

The testInput and testOutput buffers can also be plotted using software such as MATLAB.

You can copy floating-point data from the **Floating Point Rendering** tab by selecting the memory area and using **Copy to Clipboard**. You may need to manually edit the copied data before using it with plotting tools.

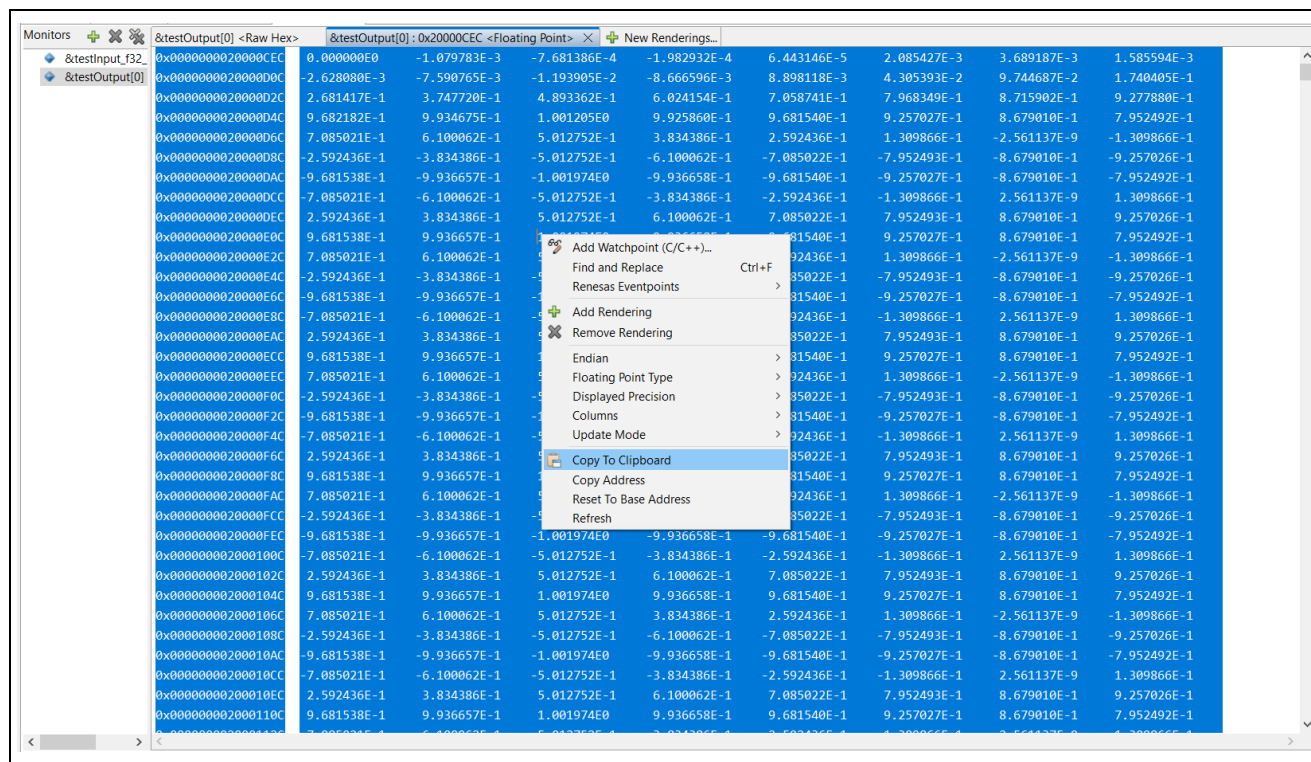


Figure 8. Copy testOutput buffer to clipboard

Note: Alternatively, the buffers can be exported using the debugger's memory export function. Available export formats depend on the debugger configuration. When exporting as raw binary, select the buffer's address range and import the file into the plotting tool using the correct data type and target byte order. For these floating-point buffers, use 32-bit floating-point values.

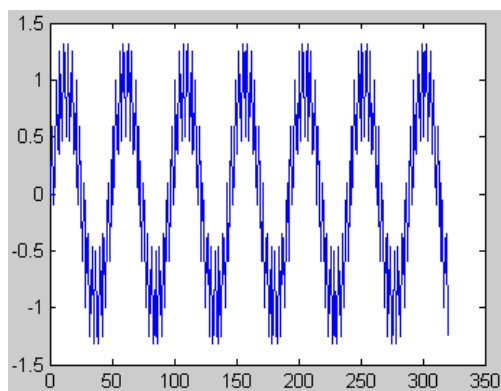


Figure 9. Waveform of Input Buffer of Lowpass Filter Example

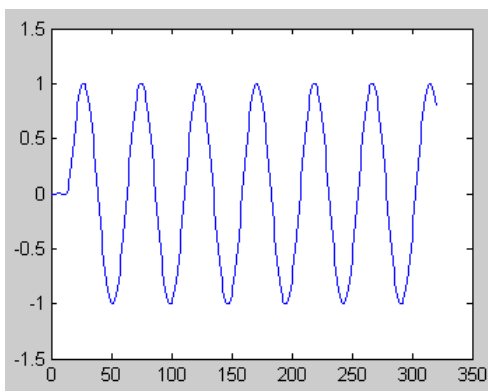


Figure 10. Waveform of Output Buffer of Lowpass Filter Example

2.2 Project Settings

The following settings are recommended to achieve the best performance on the Cortex-R52 core with DSP extension.

Parameter	Default	Setting	Notes
semihosting-enable	e ² studio: false EWARM: Semihosted	e ² studio: false EWARM: None	Semihosting is not used
Optimization	e ² studio: -Og EWARM: Low	e ² studio: -O2 EWARM: Medium	Optimize more
BSP/Heap Size	RZ/T2L, RZ/T2M : 0x8000 RZ/N2L, RZ/T2N : 0x2000	Default	Configured heap size
Miscellaneous	e ² studio: false	e ² studio: Use float with nano printf	Cross ARM C Linker

3. Performance Measurement

3.1 Enable Performance Measurement

The DSP_INSTRUCTION_BENCH macro enables performance measurement code for the arm_fir_f32 function in the arm_fir_example_f32 example. Set DSP_INSTRUCTION_BENCH to 1U to enable this performance measurement. Set it to 0U to disable performance measurement.

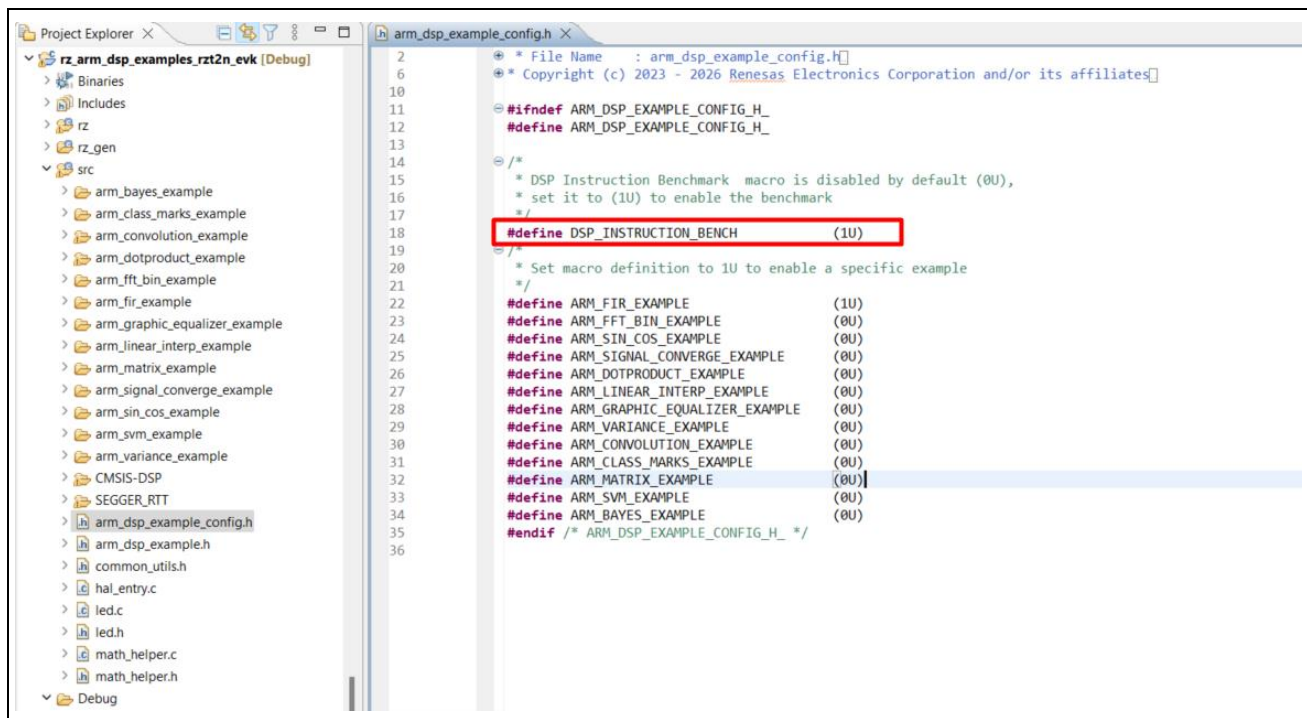


Figure 11. Enable Performance Measurement in arm_dsp_example_config.h

3.1.1 Using LED Function Code to Measure DSP Processing Interval

The LED1 on the RZ kits can be probed to measure the time required to complete the signal-processing operation. The LED1 is turned on before the DSP operation starts and turned off after it completes. This technique (GPIO toggling-based measurement) provides a simple method for measuring DSP processing time. Refer to arm_fir_example_f32.c for an example of this implementation, as shown Figure 12.

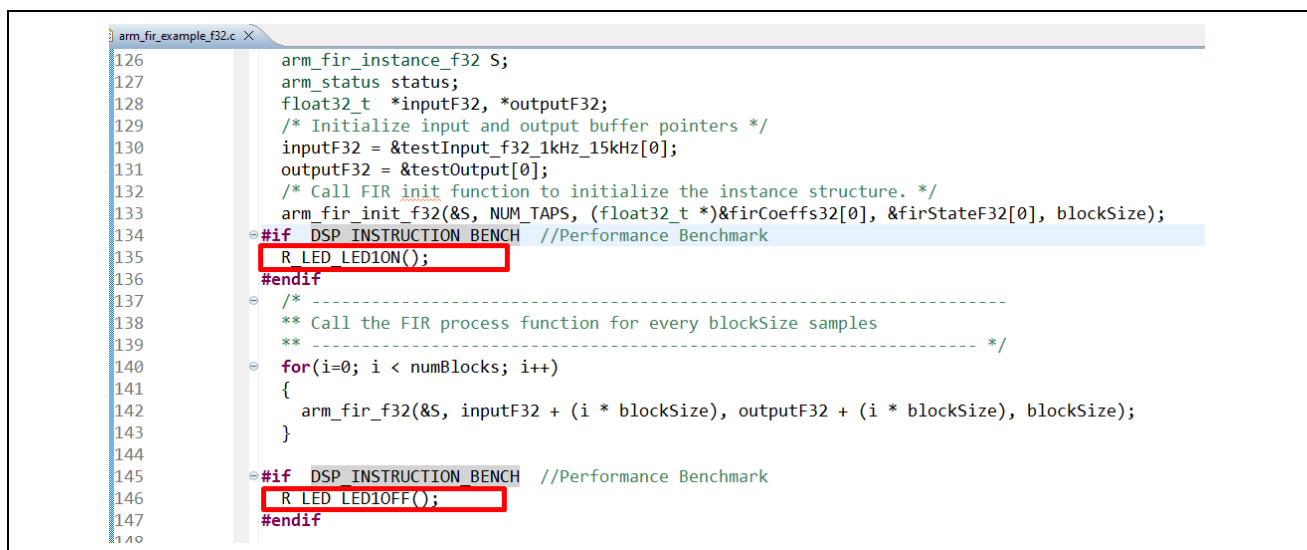
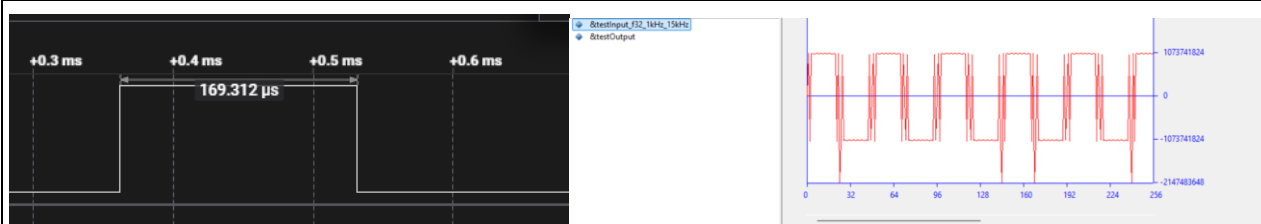


Figure 12. Setting Points of LED1 Light Start and End for Timing Measurement

- Using a digital oscilloscope (as Figure 13), probe the anode of the LED on the RZ/N2L, RZ/T2L, RZ/T2N or RZ/T2M board and trigger on a high-to-low transition.

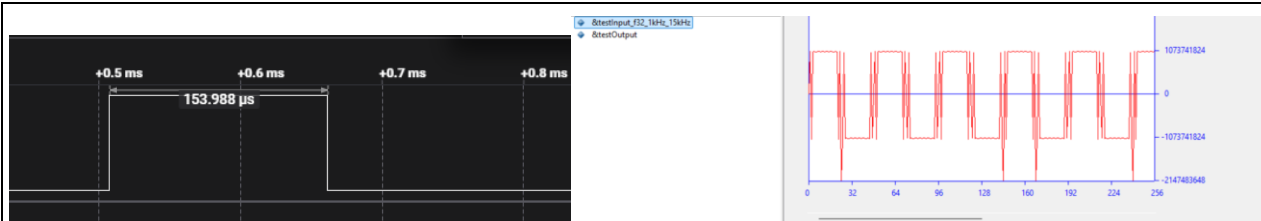
$RZ/N2L = 169.312 \text{ us} = 0.169312 \text{ ms}$ (CPU clock set to 400 MHz).



$RZ/T2L = 152.214 \text{ us} = 0.152214 \text{ ms}$ (CPU clock set to 800 MHz).



$RZ/T2M = 153.988 \text{ us} = 0.153988 \text{ ms}$ (CPU clock set to 800 MHz).



$RZ/T2N = 159.813 \text{ us} = 0.159813 \text{ ms}$ (CPU clock set to 400 MHz).

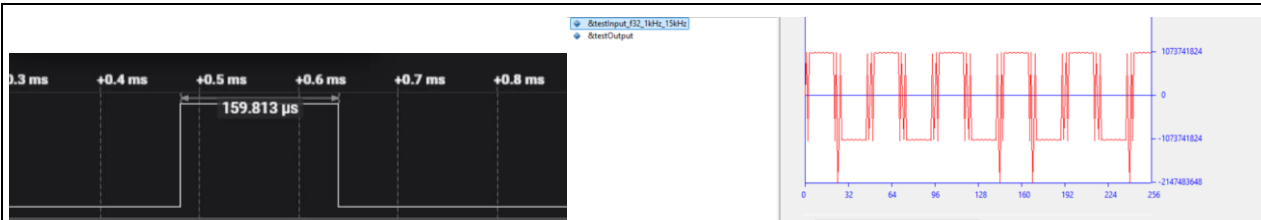


Figure 13. Processing-Time Measurements for the RZ/N2L, RZ/T2L, RZ/T2M and RZ/T2N

3.2 Displaying Example Status

Example status information is output to the SEGGER J-Link RTT Viewer. You can connect to the SEGGER RTT Viewer by following the steps below.

Launch SEGGER RTT Viewer V9.44 or later from the Windows Start menu.

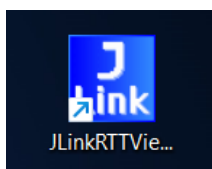


Figure 14. Launch J-Link RTT Viewer

Note: For communication between J-Link RTT Viewer and Cortex-R52-based devices, select the **Address** option for the RTT Control Block and enter the memory address shown in Figure 15. This address can be obtained from the map file located at Debug > rz_arm_dsp_example_rzxxx_xxx.map (with e² studio) or Debug > List > rz_arm_dsp_examples_rzxxx_xxx.map (with EWARM) in the

rz_arm_dsp_examples_rzxxx_xxx project. Search for “_SEGGER_RTT”. Each arm_dsp_example build may generate a different RTT control block address.

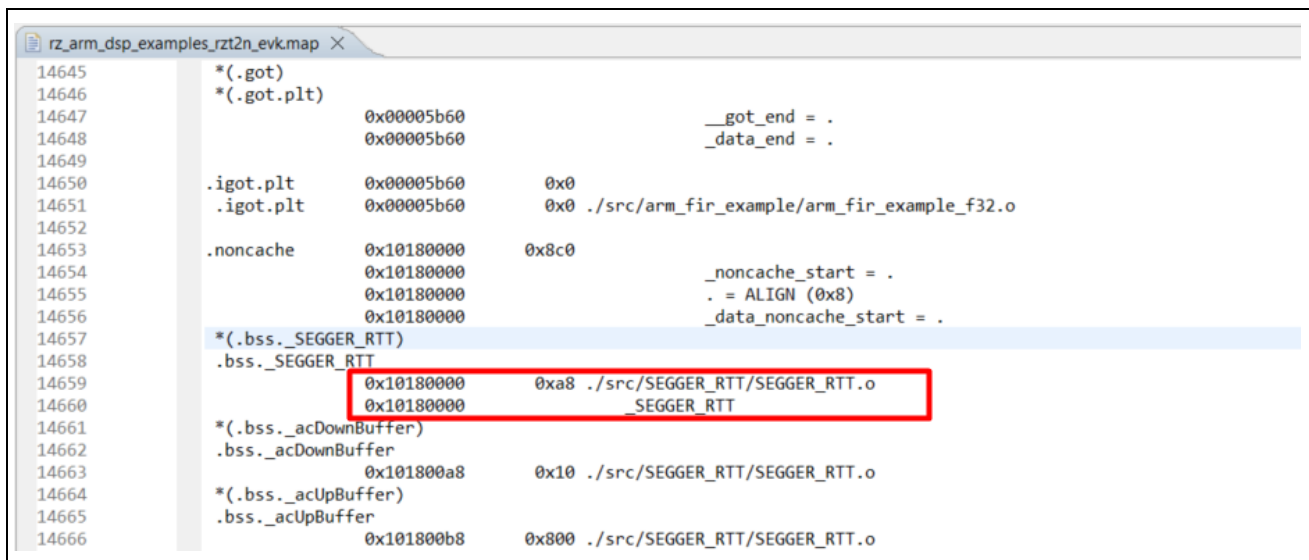


Figure 15. J-Link RTT Viewer Control Block Address

Configure the SEGGER J-Link RTT Viewer.

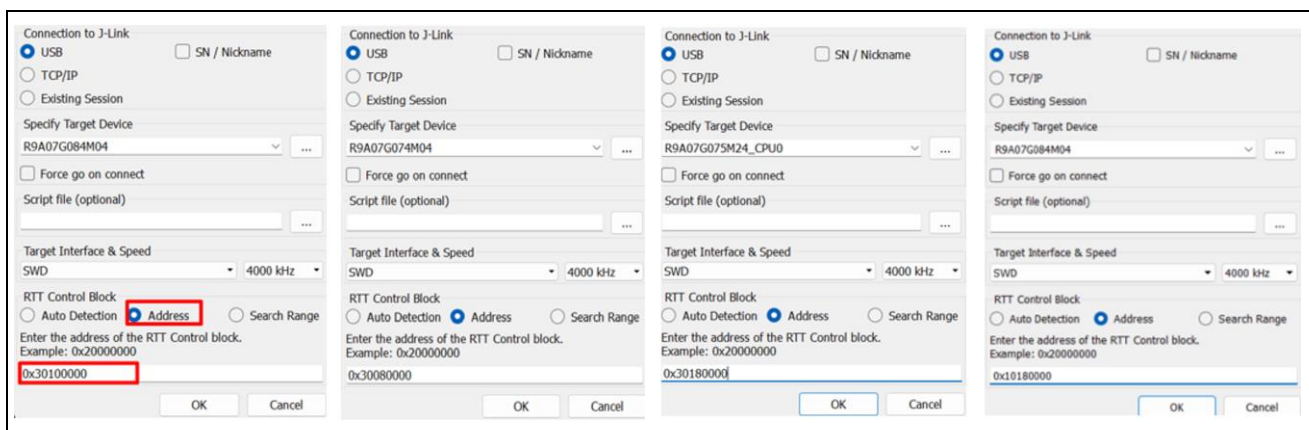


Figure 16. RTT Viewer Settings for RZ/N2L, RZ/T2L, RZ/T2M and RZ/T2N

A typical example status output is shown in Figure 17.

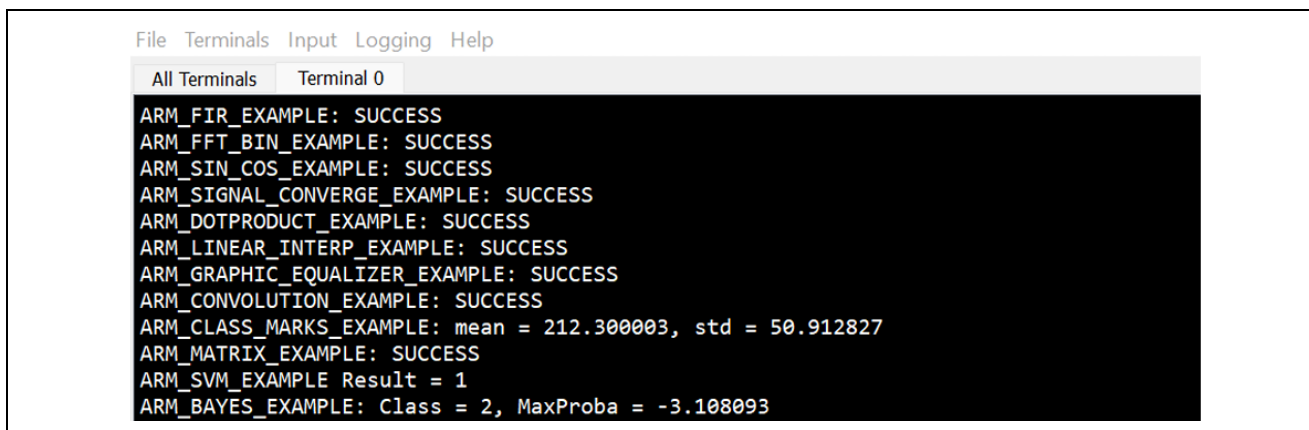


Figure 17. Example Status Output in SEGGER J-Link RTT Viewer

4. Memory Optimization for Transform Functions

This chapter describes the memory optimization applied to CMSIS-DSP Transform Functions for the RZ/N2L, RZ/T2L, RZ/T2M, and RZ/T2N groups. The project uses a customized CMSIS-DSP library based on the original Arm CMSIS-DSP source. The library has been modified to reduce memory usage for the supported transform functions.

It explains how to use the customized library in the application project, configure the required macros for each FFT length, and evaluate the resulting memory footprint.

- Section 4.1 describes the customized CMSIS-DSP library included in the project.
- Section 4.2 explains the required configuration flow and macro settings.
- Section 4.3 summarizes the supported transform functions, FFT lengths, configuration macros, and memory usage information.

4.1 Customized CMSIS-DSP Library

The CMSIS-DSP library included in this project is a customized version based on Arm CMSIS-DSP Version 1.16.2. The modifications focus on reducing the memory footprint of lookup tables used by the supported Transform Functions.

The following modifications have already been applied:

- Lookup table partitioning: The twiddle-factor and bit-reversal tables have been divided according to the API function and FFT length, allowing the required tables to be selected individually.
- Macro-controlled table selection: Compile-time macro definitions select the tables required for each supported API function and FFT length. The macros follow the naming convention [XXX]_SAMP[YYYY], where [XXX] is the API function name in uppercase and [YYYY] is the number of points. For example, ARM_CFFT_RADIX4_F32_SAMP64 selects the tables required for a 64-point transform using arm_cfft_radix4_f32.
- Selective inclusion of lookup tables: The customized implementation allows the application to include only the lookup tables required for the configured FFT lengths, rather than all default library tables. Multiple FFT lengths can be enabled by defining the corresponding macros.

The customized library source code is included in the following project directory as show in Figure 18:

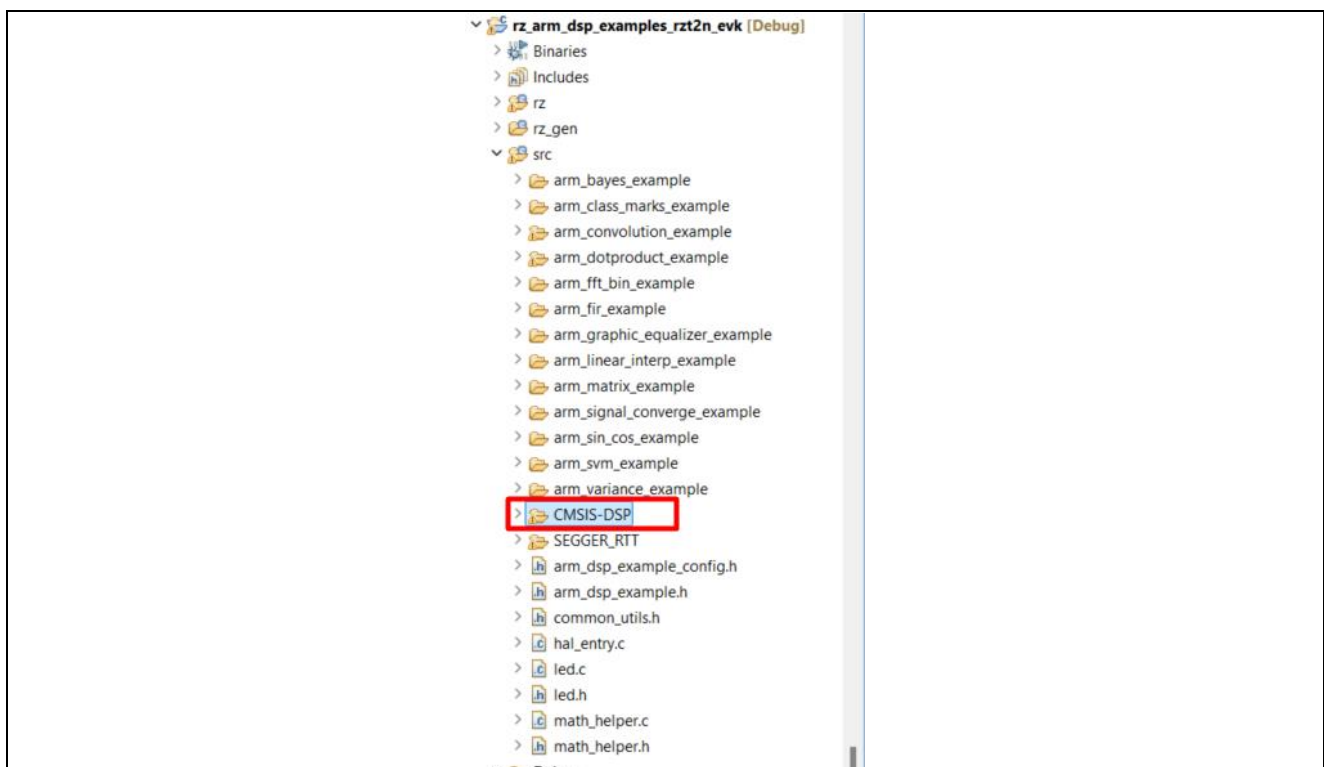


Figure 18. CMSIS-DSP Folder in the Project

No additional library modification or replacement is required. Configure the macros for the API functions and FFT lengths used by the application, as described in 4.2, and then build the project. Refer to 4.3 for the supported functions, FFT lengths, configuration macros, and memory usage information.

4.2 Configuration

The API functions in the Transform Functions category of the CMSIS-DSP library for the RZ/N2L, RZ/T2L, RZ/T2M and RZ/T2N groups have been extended to optimize the tables used for the number of points required by the user application. With this extension, for example, the `arm_cfft_f32` function can place library data in high-speed memory for configurations up to 4096 points.

Users can enable this extension by adding macro definitions in the IDE that correspond to the number of points used by the selected API functions. For more information, refer to the procedure in Section “How to Configure”, and the macro definitions.

4.2.1 How to Configure

This section describes how to enable the required FFT lengths (number of points) for each supported Transform API in the customized CMSIS-DSP library by adding the corresponding macros to the compiler symbol definitions in the IDE. These macros select the lookup tables to be included in the build.

Procedure (Overview)

- Select the API function and the number of points you want to use.
- Enable the number of points selected in step 1 by adding the corresponding macro from Section “Macro”, to the compiler symbol definitions in the IDE.
- If you want to enable multiple point sizes, repeat steps 1 and 2.

Procedure (Detail)

- Select the API function and the number of points you want to use.
For the supported combinations, refer to “Table 5. List of Supported Number of Points”.

Table 1. Number of Points Supported by API Function “arm_cfft_f32” (*2)

No.	API Function Name	Supported Number of Points									
		16	32	64	128	256	512	1024	2048	4096	8192
1	arm_cfft_f32	○	○	○	○	○	○	○	○	○	—

*2 Excerpted from “Table 5. List of Supported Number of Points”.

- Enable the number of points selected in step 1 by adding the corresponding macro from Section “Macro”, to the compiler symbol definitions in the IDE.

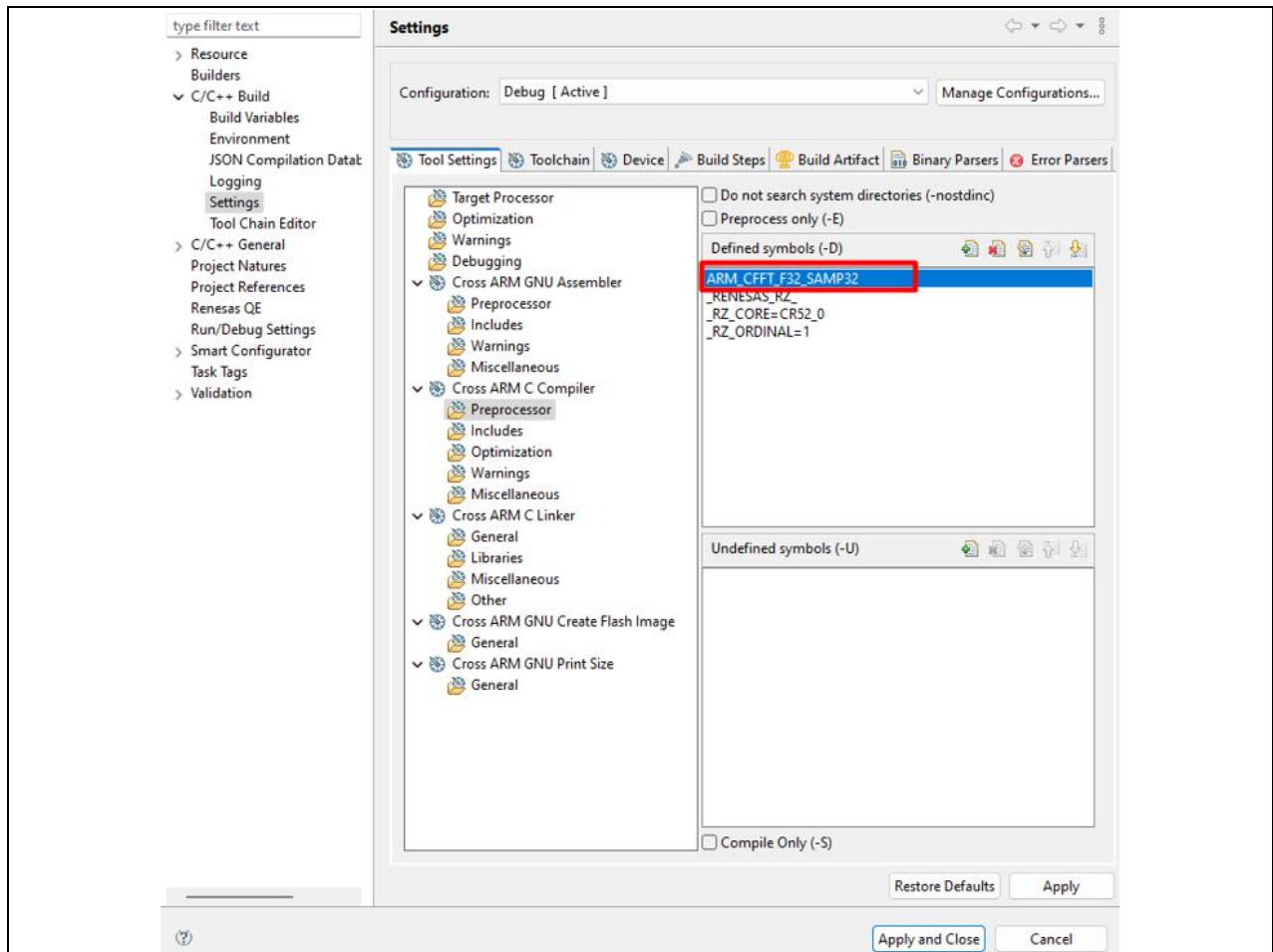


Figure 19. Example of Enabling “Number of Points 32” for “arm_cfft_f32” in e² studio

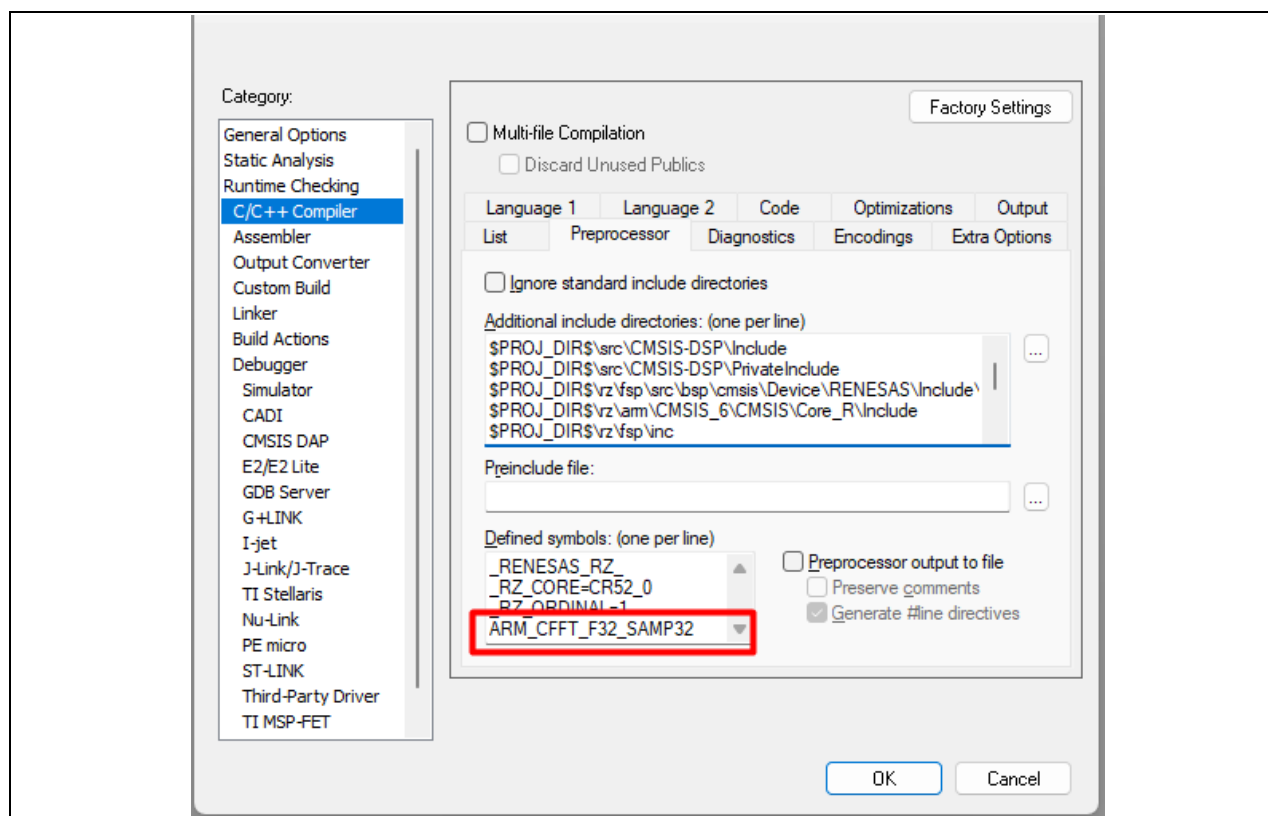


Figure 20. Example of Enabling “Number of Points 32” for “arm_cfft_f32” in EWARM

- If you want to enable multiple point sizes, repeat steps 1 and 2.
If you enable multiple point sizes, the compiler symbol definitions in the IDE must include all macros you want to enable.
Add all required macro definitions to the compiler symbol settings in the IDE.
The same applies if you use multiple API functions.

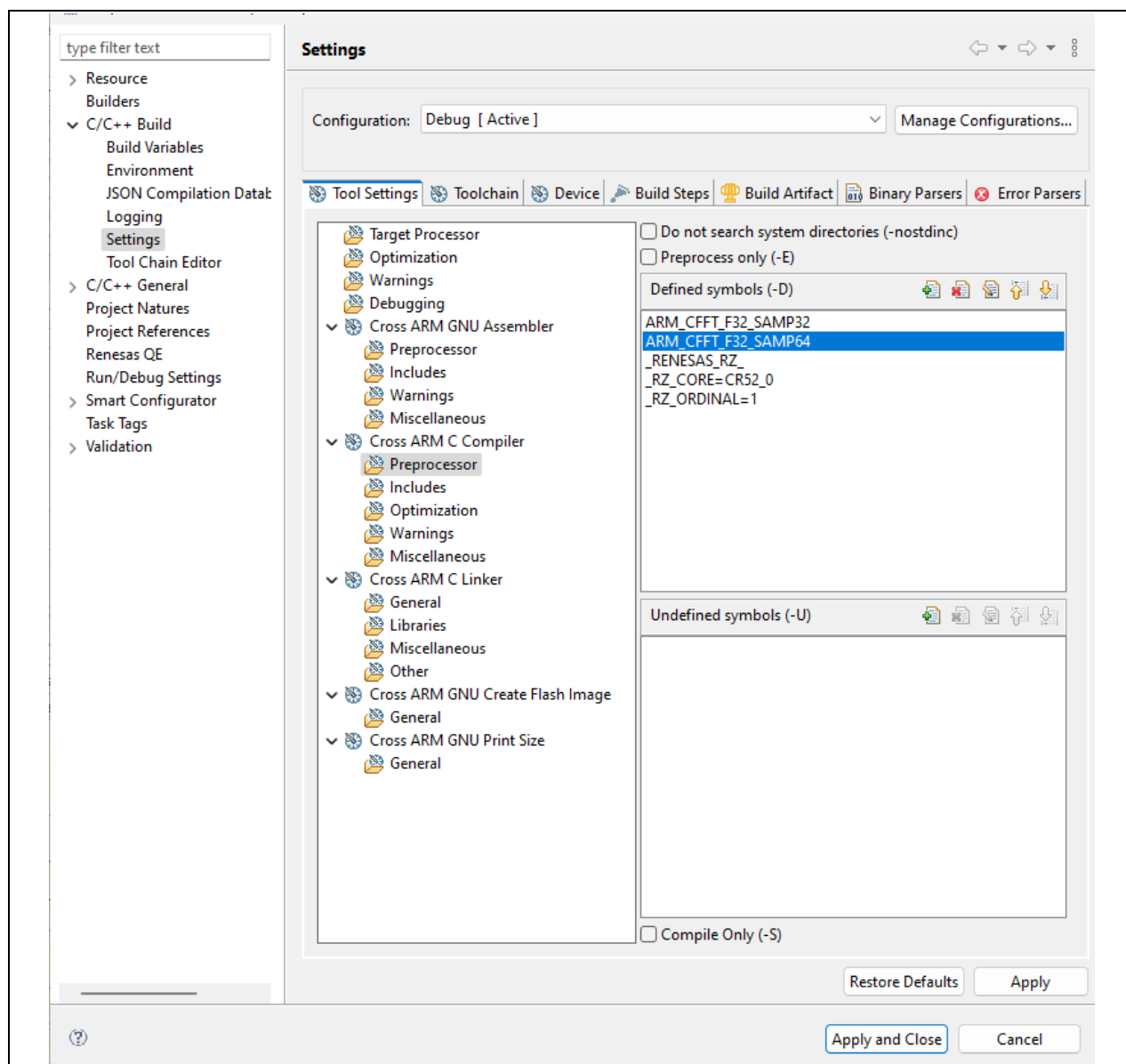


Figure 21. Example of Enabling “Number of Points 32 and 64” for “arm_cfft_f32” in e² studio

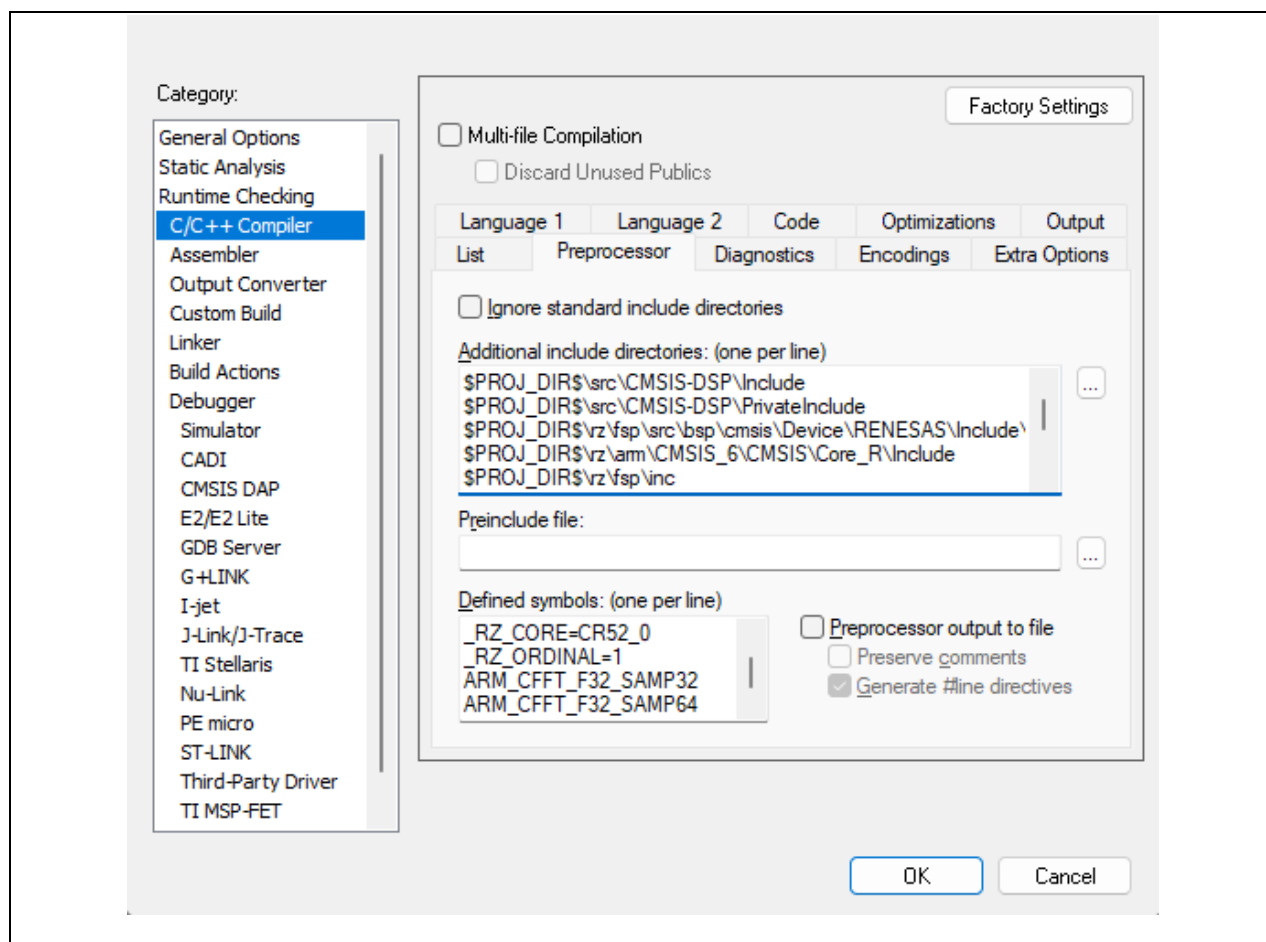


Figure 22. Example of Enabling “Number of Points 32 and 64” for “arm_cfft_f32” in EWARM

4.2.2 Memory Usage

The memory usage for the selected API function and point size can be checked in “Table 7. List of Memory Usage of Transform Functions”.

For example, if you select “number of points 32” in API function “arm_cfft_f32”, the memory usage is “0.6KB” from Table 2. Memory Usage when you selected “Number of Points 32” in API Function “arm_cfft_f32” (*2).

Table 2. Memory Usage when you selected “Number of Points 32” in API Function “arm_cfft_f32” (*2)

No.	API Function name	Number of Points	Table (KB)	Buffer (KB)	Total (KB)
2	arm_cfft_f32	32	0.3	0.3	0.6

When you select multiple point sizes, the memory usage is the sum of the table sizes and the maximum buffer size.

For example, if you select “number of points 32 and 64” in API function “arm_cfft_f32”, the memory usage is as follows, based on

Table 3. Memory Usage when you selected “Number of Points 32 and 64” in API Function “arm_cfft_f32” (*2).

Table size: 0.3(KB) + 0.6(KB) = 0.9(KB)

Buffer Size: Maximum Buffer size 0.5(KB)

Total : 1.4(KB)

Table 3. Memory Usage when you selected “Number of Points 32 and 64” in API Function “arm_cfft_f32” (*2)

No.	API Function Name	Number of Points	Table (KB)	Buffer (KB)	Total (KB)
2	arm_cfft_f32	32	0.3	0.3	0.6
3		64	0.6	0.5	1.1

*2 Excerpts from “Table 7. List of Memory Usage of Transform Functions”.

4.3 Transform Function Reference Information

This section describes the information about how to use the API function in Transform Functions.

4.3.1 Initialization Function

The table below shows the initialization function required for each supported Transform Function API. APIs marked “—” do not require a dedicated initialization.

- Explanation of symbol
— : Not Applicable

Table 4. List of Initialization Functions

No.	API Function Name	Initialization Function Name
1	arm_cfft_f32	—
2	arm_cfft_q15	—
3	arm_cfft_q31	—
4	arm_cfft_radix2_f32	arm_cfft_radix2_init_f32
5	arm_cfft_radix2_q15	arm_cfft_radix2_init_q15
6	arm_cfft_radix2_q31	arm_cfft_radix2_init_q31
7	arm_cfft_radix4_f32	arm_cfft_radix4_init_f32
8	arm_cfft_radix4_q15	arm_cfft_radix4_init_q15
9	arm_cfft_radix4_q31	arm_cfft_radix4_init_q31
10	arm_cfft_radix8_f32	—
11	arm_dct4_f32	arm_dct4_init_f32
12	arm_dct4_q15	arm_dct4_init_q15
13	arm_dct4_q31	arm_dct4_init_q31
14	arm_rfft_f32	arm_rfft_init_f32
15	arm_rfft_fast_f32	arm_rfft_fast_init_f32
16	arm_rfft_q15	arm_rfft_init_q15
17	arm_rfft_q31	arm_rfft_init_q31

4.3.2 Number of Points

The table below lists the supported number of points for each API function in the Transform Functions category.

- Explanation of symbol
O : Supported
— : Unsupported

Table 5. List of Supported Number of Points

No.	API Function Name	Supported Number of Points									
		16	32	64	128	256	512	1024	2048	4096	8192
1	arm_cfft_f32	O	O	O(*1)	O	O	O(*1)	O	O	O(*1)	—
2	arm_cfft_q15	O(*2)	O	O(*2)	O	O(*2)	O	O(*2)	O	O(*2)	—
3	arm_cfft_q31	O(*2)	O	O(*2)	O	O(*2)	O	O(*2)	O	O(*2)	—
4	arm_cfft_radix2_f32	O	O	O	O	O	O	O	O	O	—
5	arm_cfft_radix2_q15	O	O	O	O	O	O	O	O	O	—
6	arm_cfft_radix2_q31	O	O	O	O	O	O	O	O	O	—
7	arm_cfft_radix4_f32	O	—	O	—	O	—	O	—	O	—
8	arm_cfft_radix4_q15	O	—	O	—	O	—	O	—	O	—
9	arm_cfft_radix4_q31	O	—	O	—	O	—	O	—	O	—
10	arm_cfft_radix8_f32	—	—	O	—	—	O	—	—	O	—
11	arm_dct4_f32	—	—	—	O	—	O	—	O	—	O
12	arm_dct4_q15	—	—	—	O	—	O	—	O	—	O
13	arm_dct4_q31	—	—	—	O	—	O	—	O	—	O
14	arm_rfft_f32	—	—	—	O	—	O	—	O	—	O
15	arm_rfft_fast_f32	—	O	O	O	O	O	O	O	O	—
16	arm_rfft_q15	—	O	O	O	O	O	O	O	O	O
17	arm_rfft_q31	—	O	O	O	O	O	O	O	O	O

*1 Treat as “arm_cfft_radix8_f32”.

*2 Treat “arm_cfft_radix4_q15” and “arm_cfft_radix4_q31” as equivalent.

4.3.3 Macro

The table below shows the naming convention for macros used to enable point sizes for API functions in the Transform Functions category.

Table 6. Macro

Macro Name	Description
[XXX]_SAMP[YYYY]	Specifies number of points of API function to enable. [XXX] = API function name in the Transform Functions category, written in uppercase. [YYYY] = Specifies number of points. The value is one of the following: 16, 32, 64, 128, 256, 512, 1024, 2048, 4096 or 8192.

Example of a macro name specifying 16 points for the arm_cfft_f32 function

ARM_CFFT_F32_SAMP16

4.3.4 Memory Usage

The tables below show the memory usage for each supported Transform Function API and FFT length. The values include the table size, buffer size, and total memory usage associated with each configuration. The Buffer Size value is provided as a reference for the memory required by the application when allocating input and output buffers for the corresponding API function.

Table 7. List of Memory Usage of Transform Functions

No.	API Function Name	Number of Points	Table (KB)	Buffer (KB)	Total (KB)
1	arm_cfft_f32	16	0.2	0.1	0.3
2		32	0.3	0.3	0.6
3		64	0.6	0.5	1.1
4		128	1.4	1.0	2.4
5		256	2.9	2.0	4.9
6		512	4.9	4.0	8.9
7		1024	11.5	8.0	19.5
8		2048	23.4	16.0	39.4
9		4096	39.9	32.0	71.9
10	arm_cfft_q15	16	0.1	0.1	0.2
11		32	0.1	0.3	0.4
12		64	0.3	0.5	0.8
13		128	0.6	1.0	1.6
14		256	1.2	2.0	3.2
15		512	2.4	4.0	6.4
16		1024	4.9	8.0	12.9
17		2048	9.9	16.0	25.9
18		4096	19.9	32.0	51.9
19	arm_cfft_q31	16	0.1	0.1	0.2
20		32	0.2	0.3	0.5
21		64	0.5	0.5	1.0
22		128	1.0	1.0	2.0
23		256	2.0	2.0	4.0
24		512	3.9	4.0	7.9
25		1024	7.9	8.0	15.9
26		2048	15.9	16.0	31.9
27		4096	31.9	32.0	63.9
28	arm_cfft_radix2_f32	16	0.1	0.1	0.2

No.	API Function Name	Number of Points	Table (KB)	Buffer (KB)	Total (KB)
29		32	0.3	0.3	0.6
30		64	0.5	0.5	1.0
31		128	1.1	1.0	2.1
32		256	2.1	2.0	4.1
33		512	4.3	4.0	8.3
34		1024	8.5	8.0	16.5
35		2048	17.0	16.0	33.0
36		4096	34.0	32.0	66.0
37	arm_cfft_radix2_q15	16	0.1	0.1	0.2
38		32	0.1	0.1	0.3
39		64	0.2	0.3	0.5
40		128	0.4	0.5	0.9
41		256	0.9	1.0	1.9
42		512	1.8	2.0	3.8
43		1024	3.5	4.0	7.5
44		2048	7.0	8.0	15.0
45		4096	14.0	16.0	30.0
46	arm_cfft_radix2_q31	16	0.1	0.1	0.2
47		32	0.2	0.3	0.5
48		64	0.4	0.5	0.9
49		128	0.8	1.0	1.8
50		256	1.6	2.0	3.6
51		512	3.3	4.0	7.3
52		1024	6.5	8.0	14.5
53		2048	13.0	16.0	29.0
54		4096	26.0	32.0	58.0
55	arm_cfft_radix4_f32	16	0.1	0.1	0.2
56		64	0.5	0.5	1.0
57		256	2.1	2.0	4.1
58		1024	8.5	8.0	16.5
59		4096	34.0	32.0	66.0
60	arm_cfft_radix4_q15	16	0.1	0.1	0.2

No.	API Function Name	Number of Points	Table (KB)	Buffer (KB)	Total (KB)
61		64	0.2	0.3	0.5
62		256	0.9	1.0	1.9
63		1024	3.5	4.0	7.5
64		4096	14.0	16.0	30.0
65	arm_cfft_radix4_q31	16	0.1	0.1	0.2
66		64	0.4	0.5	0.9
67		256	1.6	2.0	3.6
68		1024	6.5	8.0	14.5
69		4096	26.0	32.0	58.0
70	arm_cfft_radix8_f32	64	0.6	0.5	1.1
71		512	4.9	4.0	8.9
72		4096	39.9	32.0	71.9
73	arm_rfft_f32	128	1.5	2.0	3.5
74		512	6.1	8.0	14.1
75		2048	24.5	32.0	56.5
76		8192	98.0	128.0	226.0
77	arm_rfft_fast_f32	32	0.3	0.5	0.8
78		64	0.6	1.0	1.6
79		128	1.1	2.0	3.1
80		256	2.4	4.0	6.4
81		512	4.9	8.0	12.9
82		1024	8.9	16.0	24.9
83		2048	19.5	32.0	51.5
84		4096	39.4	64.0	103.4
85	arm_rfft_q15	32	0.2	0.3	0.5
86		64	0.4	0.5	0.9
87		128	0.8	1.0	1.8
88		256	1.6	2.0	3.6
89		512	3.2	4.0	7.2
90		1024	6.4	8.0	14.4
91		2048	12.9	16.0	29.9
92		4096	25.9	32.0	57.9

No.	API Function Name	Number of Points	Table (KB)	Buffer (KB)	Total (KB)
93		8192	51.9	64.0	115.9
94	arm_rfft_q31	32	0.4	0.5	0.9
95		64	0.7	1.0	1.7
96		128	1.5	2.0	3.5
97		256	3.0	4.0	7.0
98		512	6.0	8.0	14.0
99		1024	11.9	16.0	27.9
100		2048	23.9	32.0	55.9
101		4096	47.9	64.0	111.9
102		8192	95.9	128.0	223.9
103	arm_dct4_f32	128	3.0	2.0	5.0
104		512	12.1	8.0	20.1
105		2048	48.5	32.0	80.5
106		8192	194.0	128.0	322.0
107	arm_dct4_q15	128	1.5	1.0	2.5
108		512	6.2	4.0	10.2
109		2048	24.9	16.0	40.9
110		8192	99.9	64.0	163.9
111	arm_dct4_q31	128	3.0	2.0	5.0
112		512	12.0	8.0	20.0
113		2048	47.9	32.0	79.9
114		8192	191.9	128.0	319.9

5. Memory Optimization Results for the Arm® DSP Examples

This section describes how to demonstrate the results of the optimized library. Proceed with the following steps:

5.1 Configure the Application Project

- Update the `ARM_CONVOLUTION_EXAMPLE` Macro for the Convolution Example
Open the file `rz_arm_dsp_examples_rzxxx_xxx/src/arm_dsp_example_config.h` and set the `ARM_CONVOLUTION_EXAMPLE` macro to 1.

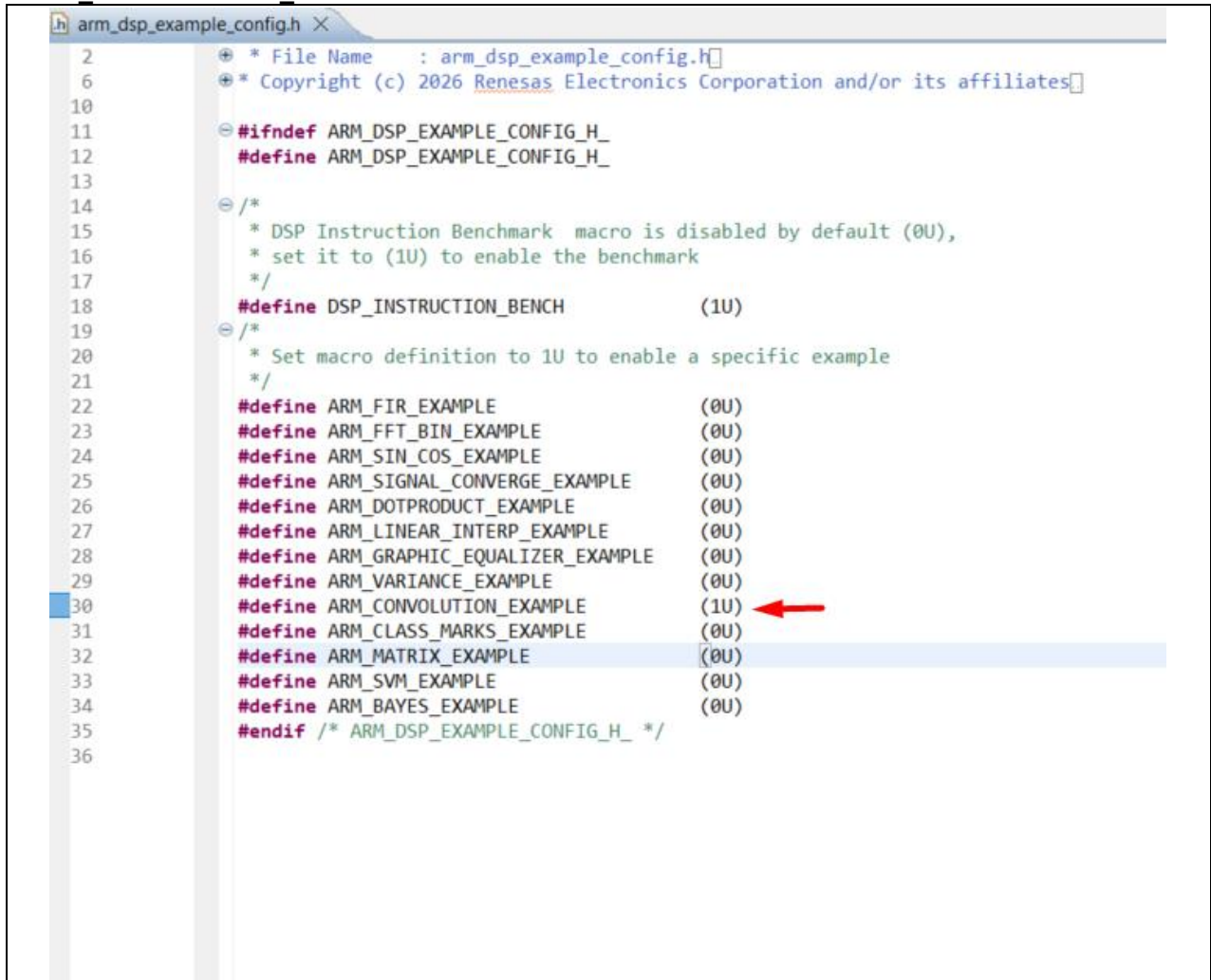


Figure 23. Updating the `ARM_CONVOLUTION_EXAMPLE` Macro in `arm_dsp_example_config.h`

- Adding a Macro for the Number of Points
The Convolution Example uses the `arm_cfft_radix4_f32` function with **64 points**. Therefore, add the macro `ARM_CFFT_RADIX4_F32_SAMP64` to the compiler symbol definitions (refer to Section 4.2.1 How to Configure”).

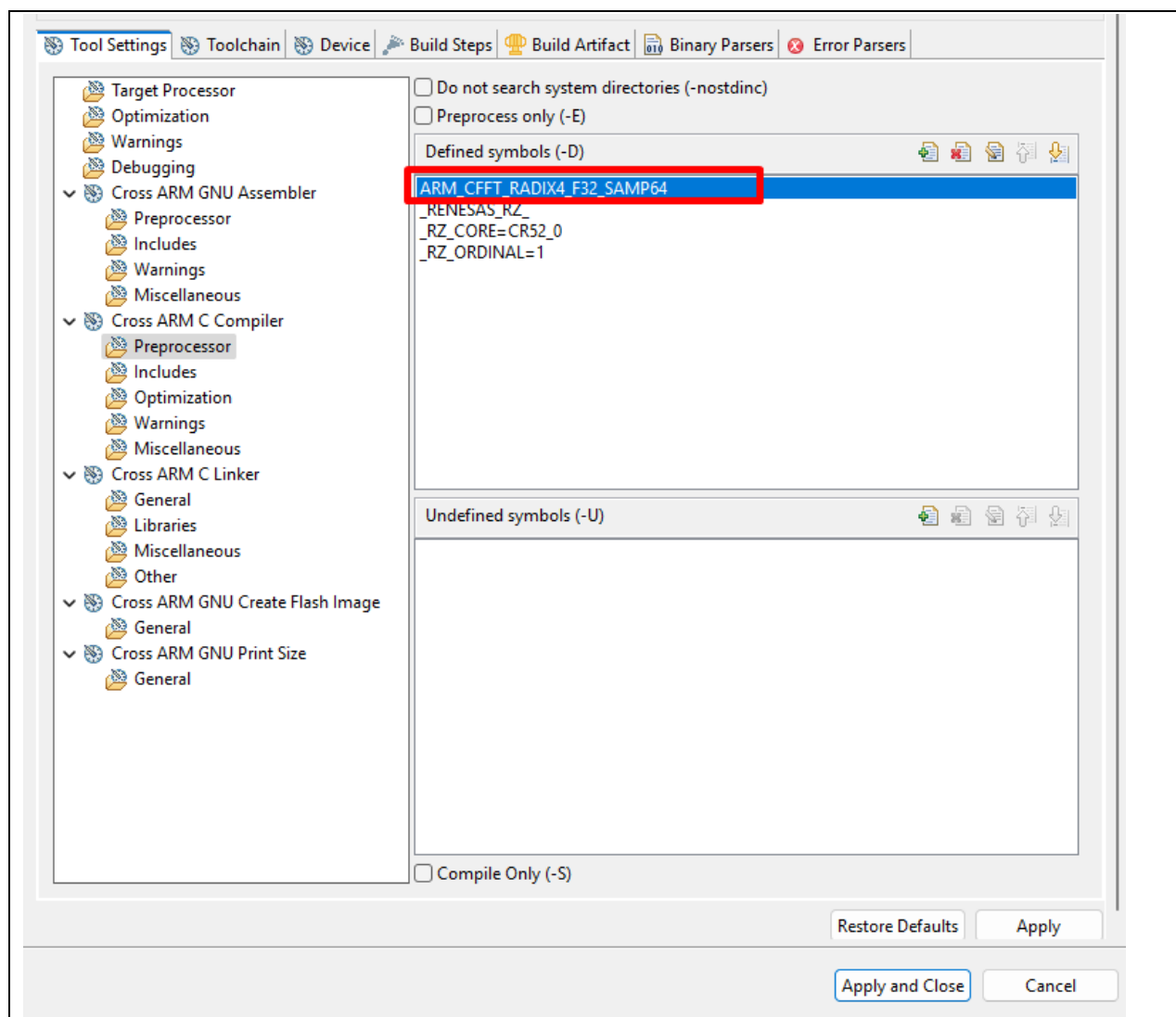


Figure 24. Enabling “Number of Points 64” for “arm_cfft_radix4_f32” in e² studio

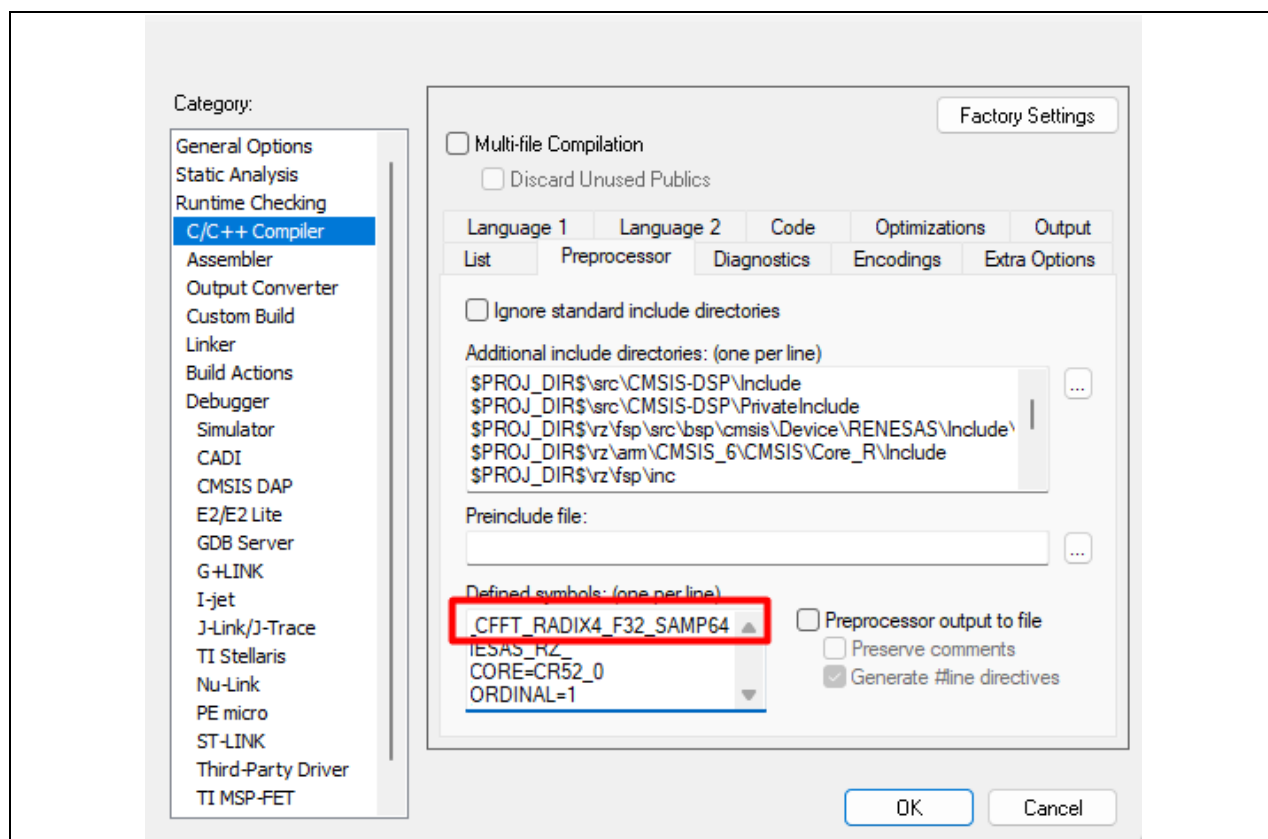


Figure 25. Enabling “Number of Points 64” for “arm_cfft_radix4_f32” in EWARM

- After completing the configuration, build the application project.

5.2 Results for the Updated Example

5.2.1 Example Status

By following the procedure in Section “Displaying Example Status”, the output log when running the Convolution Example is as follows:

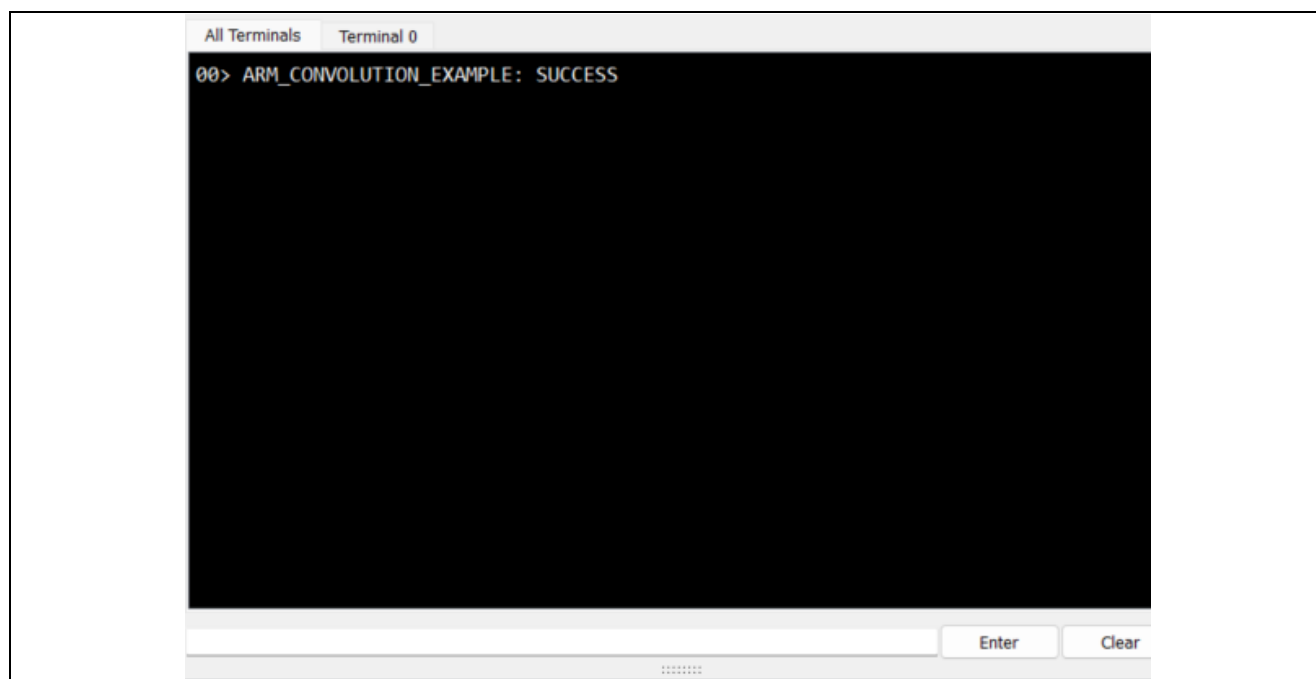


Figure 26. Log Output of the Convolution Example in SEGGER J-Link RTT Viewer

5.2.2 Memory Usage Comparison

“Memory Usage (C/C++)” in e² studio is a built-in feature that helps developers analyze and monitor memory consumption of embedded applications after a build.

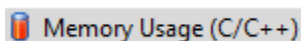


Figure 27. Memory Usage (C/C++) View in e² studio

Using the “Memory Usage (C/C++)” tool in e² studio, users can check the detailed memory footprint of the application.

The following images compare the memory usage of the original library and the customized library:

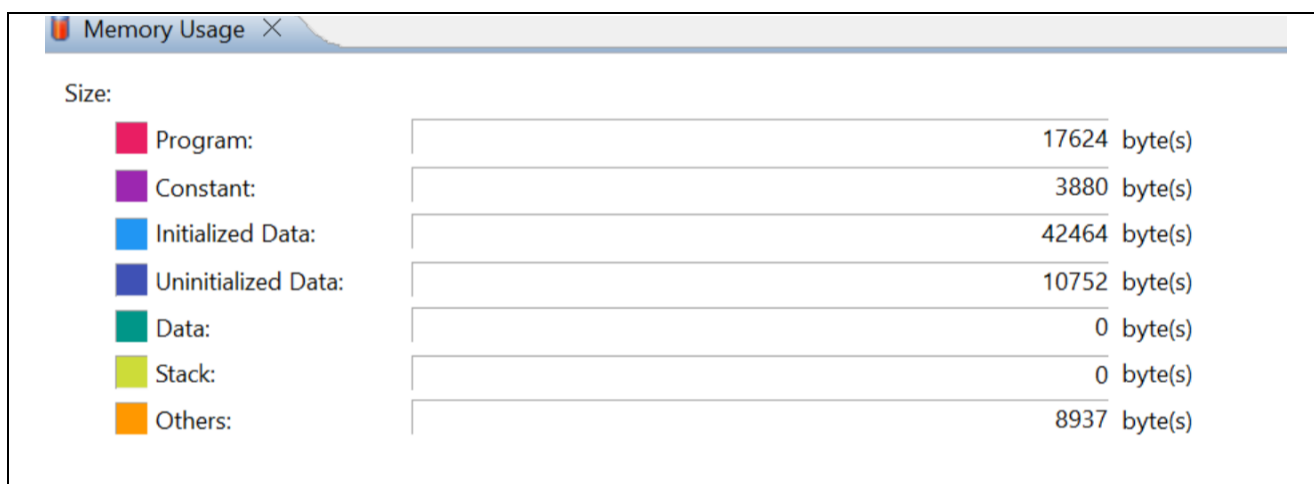


Figure 28. Memory Usage with the original Library

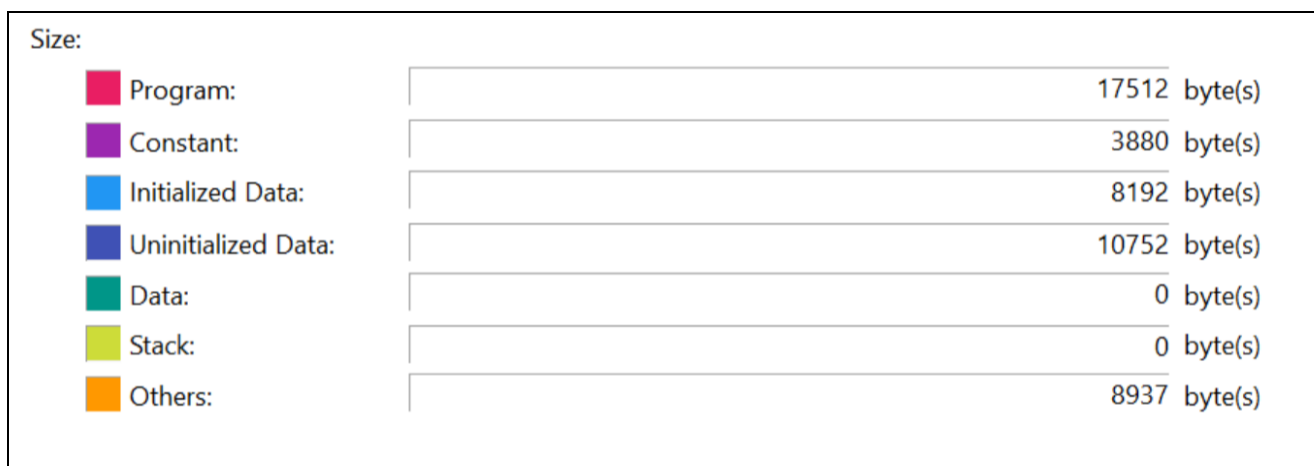


Figure 29. Memory Usage with the Customized Library

The detailed comparison is shown in the table below:

Table 8. Memory Usage Comparison Between the Original and Customized library

	Original library	Customized library	Reduction rate (%)
Initialized Data (bytes)	42464	8192	80.71
twiddleCoef (bytes)	32768	512	98.44
armBitRevTable (bytes)	2048	32	98.44

Note: *twiddleCoef* and *armBitRevTable* are included in Initialized Data (Figure 29). Their combined reduction accounts for the 34,272-byte decrease shown in Table 8.

6. Website and Support

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

RZ Product Information	renesas.com/rz
RZ Product Support Forum	renesas.com/rz/forum
RZ Flexible Software Package	renesas.com/fsp
Renesas Support	renesas.com/support

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Nov.04.26	—	First release document

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

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

1. Precaution against Electrostatic Discharge (ESD)

A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity.

Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.

2. Processing at power-on

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

3. Input of signal during power-off state

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

4. Handling of unused pins

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

5. Clock signals

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

6. Voltage application waveform at input pin

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

7. Prohibition of access to reserved addresses

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

8. Differences between products

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

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.

"Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.

"High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user's manual or other Renesas Electronics document.

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

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

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

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 May 2026)