Renesas RA Family

Arm® DSP Examples

Introduction

This application note describes the use of ARM® CMSIS-DSP example projects that are ported to Renesas Arm® Cortex-M33, Cortex-M85, and Cortex-M23 core-based MCUs with digital signal processing (DSP) extension and Floating Point Unit (FPU). This application note will discuss the steps to import, configure, build, and execute these DSP examples and measure their performances.

Additionally, this application note describes improved performance with Renesas RA8 MCUs using Arm® Cortex-M85 core with Helium™.

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

https://arm-software.github.io/CMSIS_5/DSP/html/group__groupExamples.html

Required Resources

Hardware

- EK-RA8M1, Evaluation Kit for RA8M1 MCU Group (renesas.com/ra/ek-ra8m1)
- EK-RA6M4, Evaluation Kit for RA6M4 MCU Group (renesas.com/ra/ek-ra6m4)
- EK-RA2E1, Evaluation Kit for RA2E1 MCU Group (renesas.com/ra/ek-ra2e1)

Development Tools and Software

- The e^2 studio IDE v2024-07
- Renesas Flexible Software Package (FSP) v5.5.0
- GCC ARM Embedded Toolchain version 13.2.1.arm-13-7
- LLVM Embedded Toolchain version 18.1.3
- Segger J-Link® USB driver and RTT Viewer version 7.98b

The above software components are bundled in a downloadable platform installer available on the FSP webpage at renesas.com/ra/fsp and the Segger webpage at segger.com/j-link/rtt-viewer.

Target Devices

This application note focuses on RA6M4 MCU. However, it also applies to any Arm Cortex-M33, Arm Cortex-M85, and Cortex-M23 core-based Renesas MCUs. It includes M23 RA2 devices, M33 RA4 and RA6 devices and M85 RA8 devices.

Contents

1. Arm® CMSIS-DSP Library in Renesas Flexible Software Package (FSPv5.5.0)

The CMSIS DSP software library is a suite of common signal-processing functions used on Cortex-M and Cortex-A processor-based devices.

The library is divided into several functions, each covering a specific category.

- 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

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 ported to FSP and can be easily included in your project by using **New Stack** in the **Stacks** tab in the FSP configurator, as shown in Figure 1.

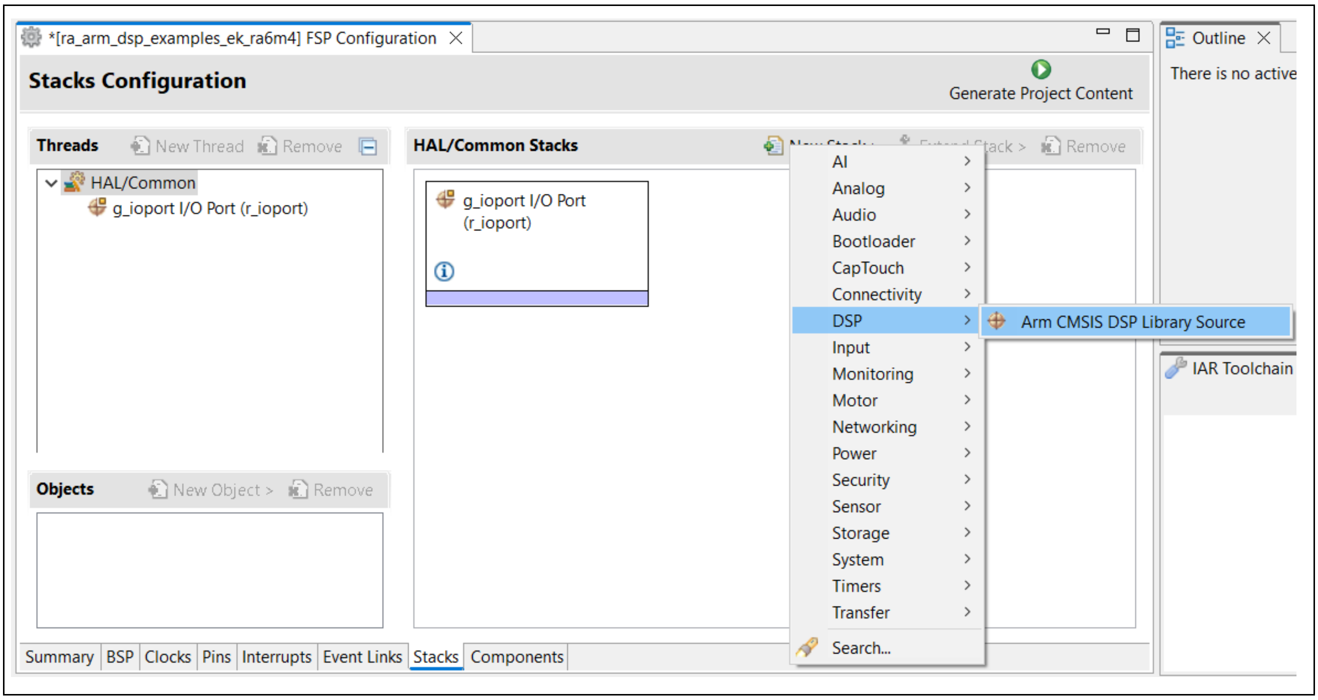


Figure 1. Adding CMSIS-DSP Library Source in FSP Configurator Using New Stack Feature

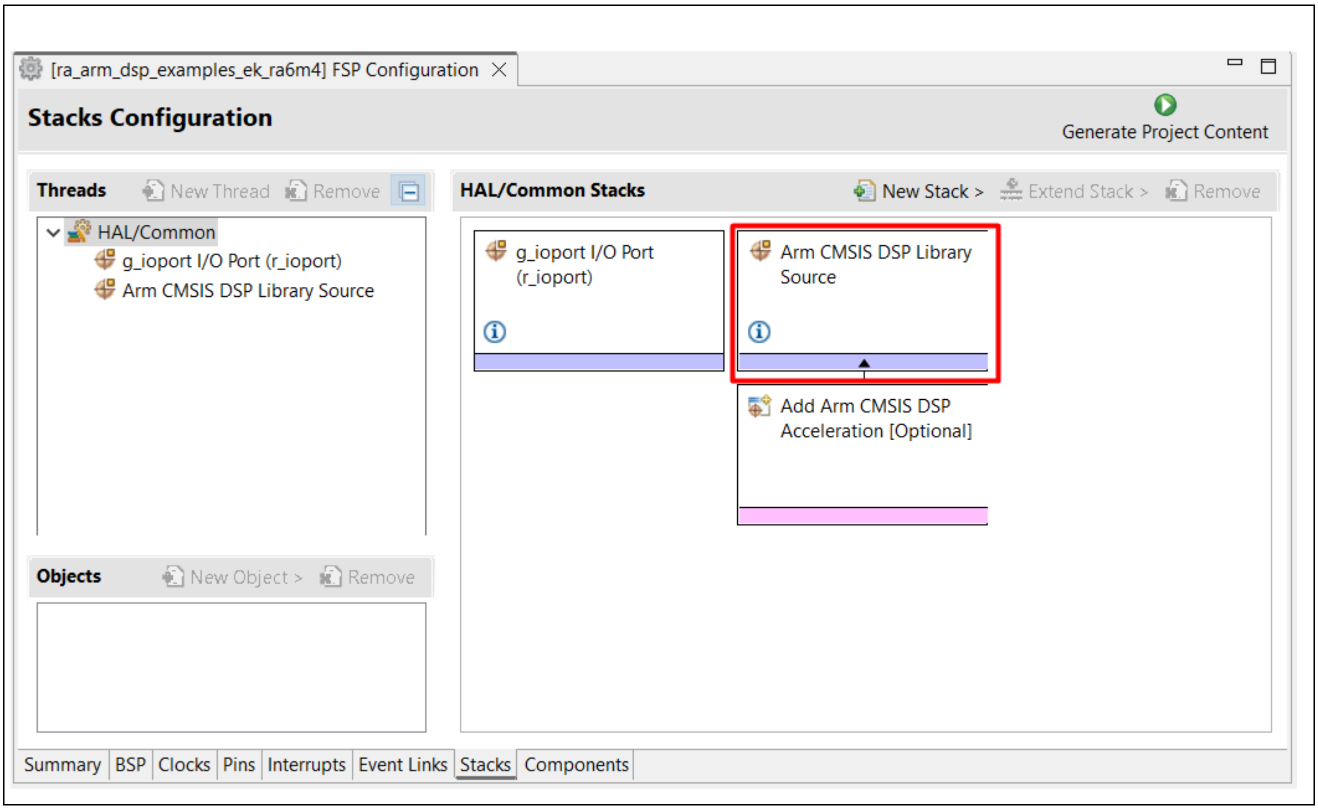


Figure 2. Adding CMSIS-DSP Source Library in FSP Configurator

Pressing **Generate Project Content** will generate the CMSIS-DSP library in your project.

Header and source files are created as shown in Figure 3.

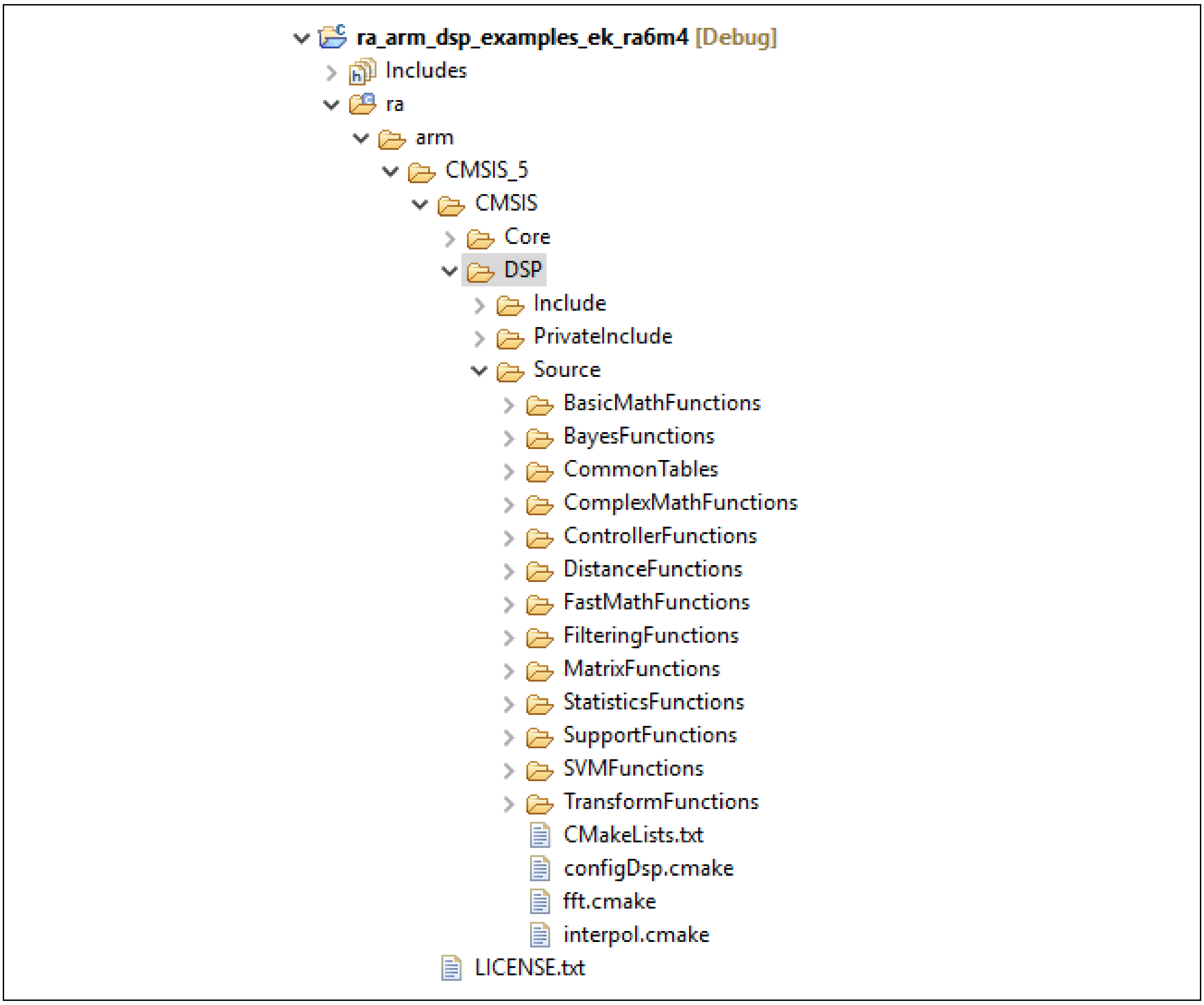


Figure 3. CMSIS-DSP Library Source in Example Project

2. Arm CMSIS-DSP Examples

Original Arm CMSIS-DSP examples are ported to the e² studio/FSP environment in a single project form. This example project includes multiple CMSIS-DSP examples. Below is the list of examples supported by this example 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 of the example project import can be found here.

Details of the above examples can be found here.

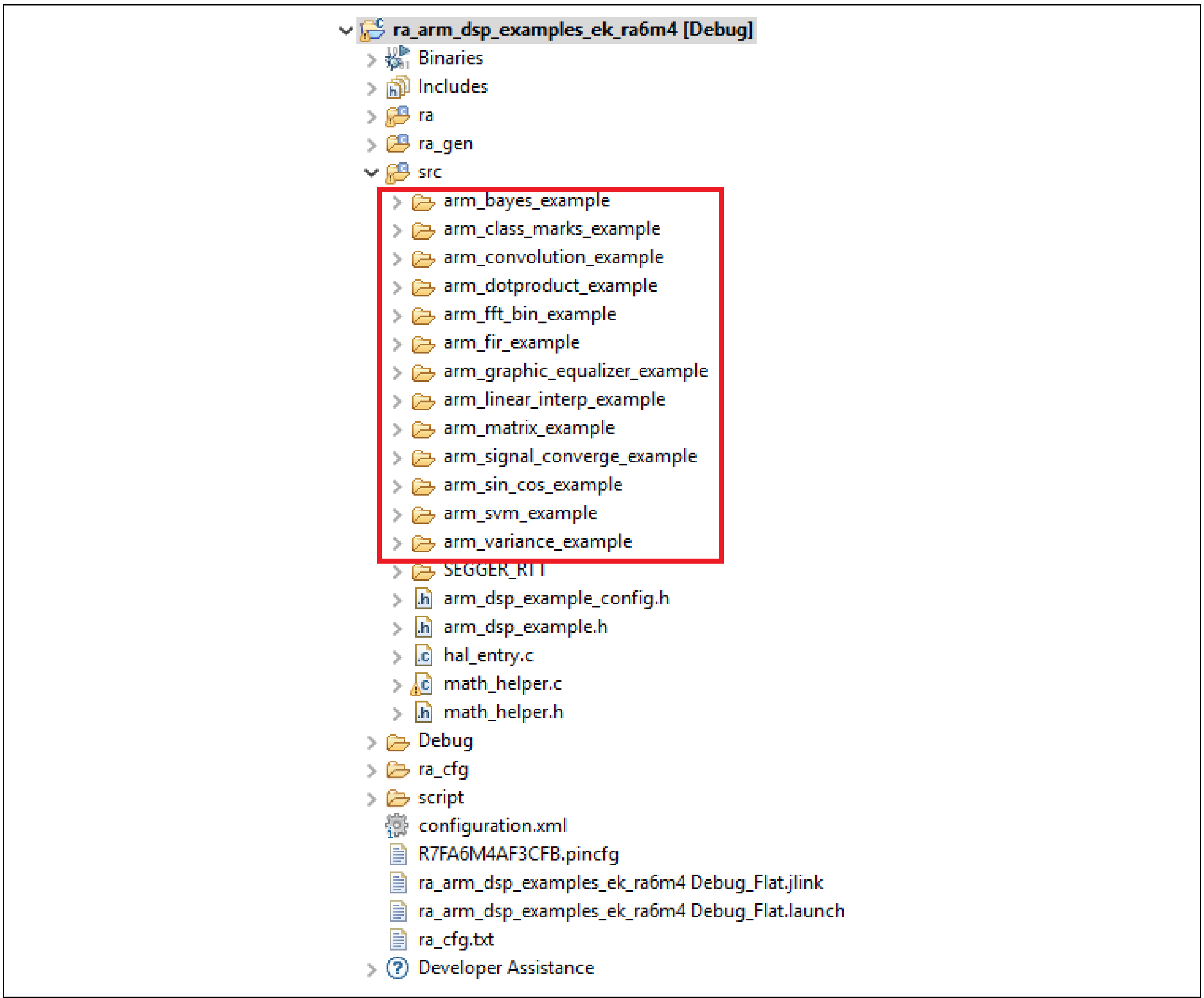


Figure 4. Example Projects are Imported to e² studio Including Multiple CMSIS-DSP Examples

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

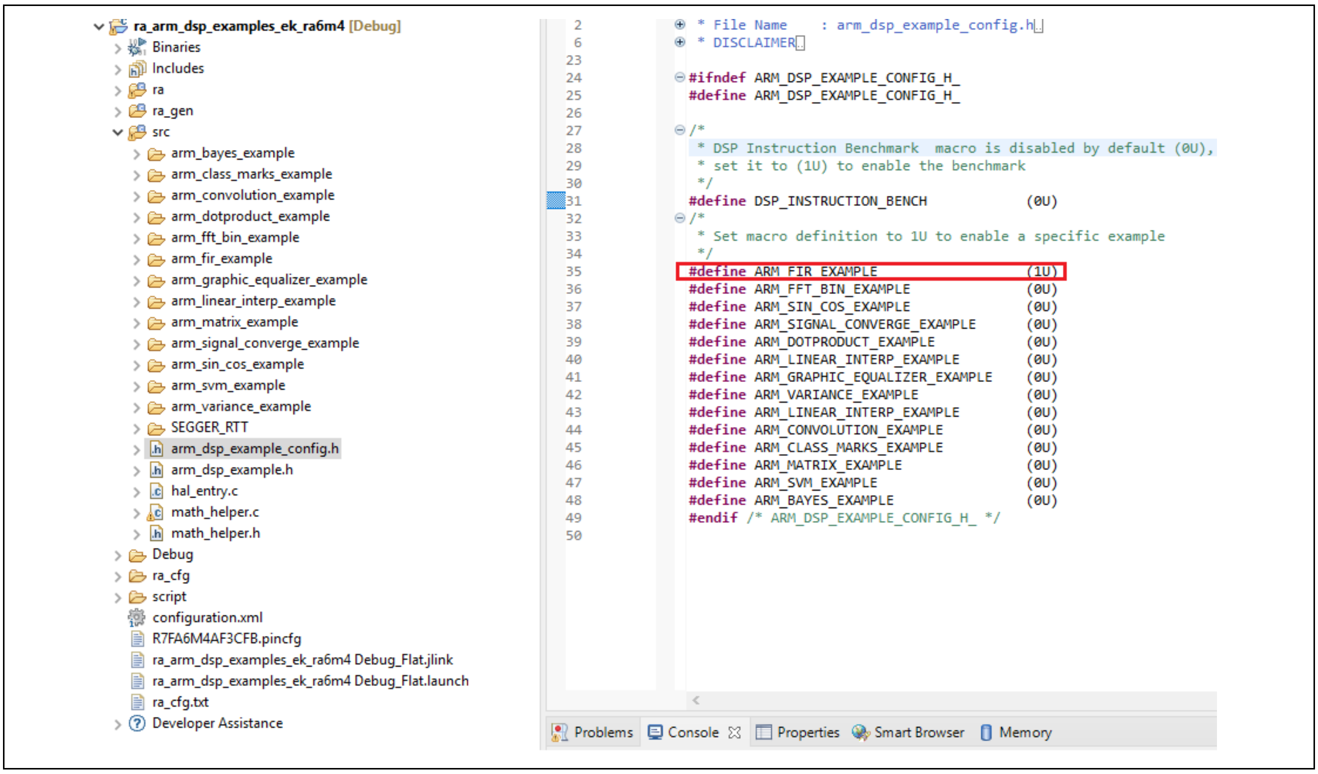
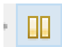


Figure 5. arm_dsp_example_config.h

2.1 Build and Run CMSIS-DSP Example

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

You can now build, download, and run the project. The following screenshots show an example of running the Lowpass Filter example.

After downloading and running the project, wait for a few seconds and hit the **Suspend** button  to stop project execution.

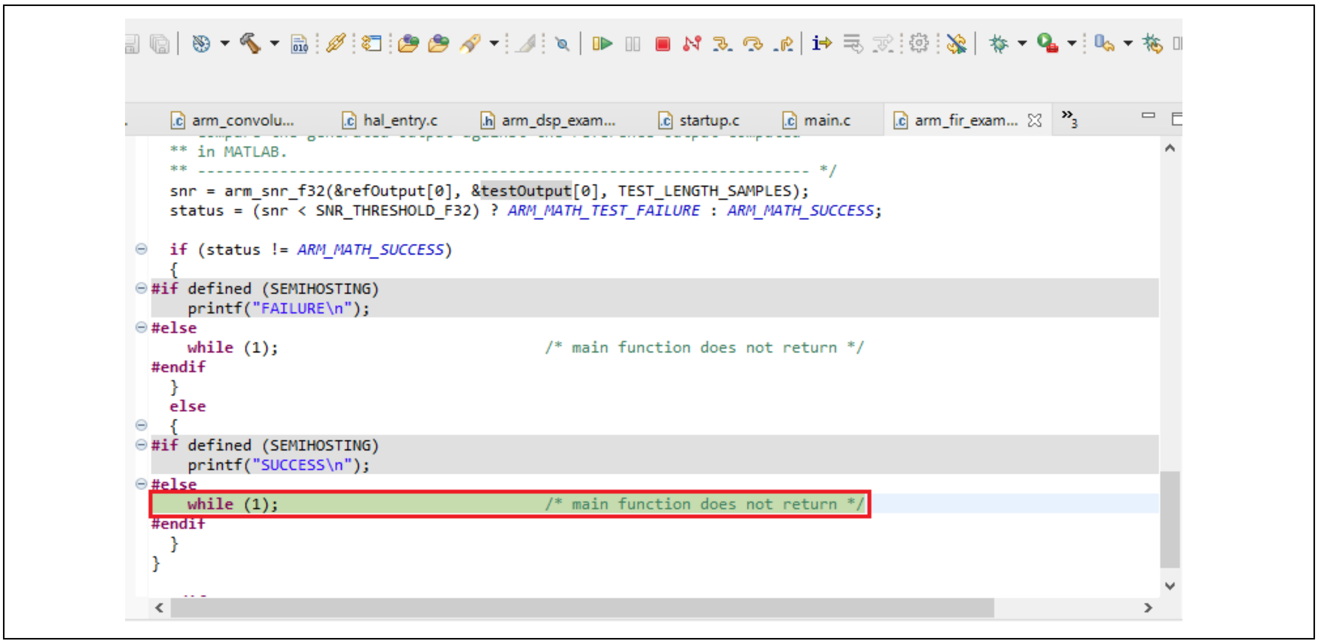


Figure 6. Lowpass Filter example Ran Successfully and Stopped at “SUCCESS” while (1)

The waveforms of input and output buffers can be used to verify the result visually. Add testOutput buffer to the **Memory** tab as shown in Figure 7, then add a new **Fixed Floating Point** rendering for the same buffer.

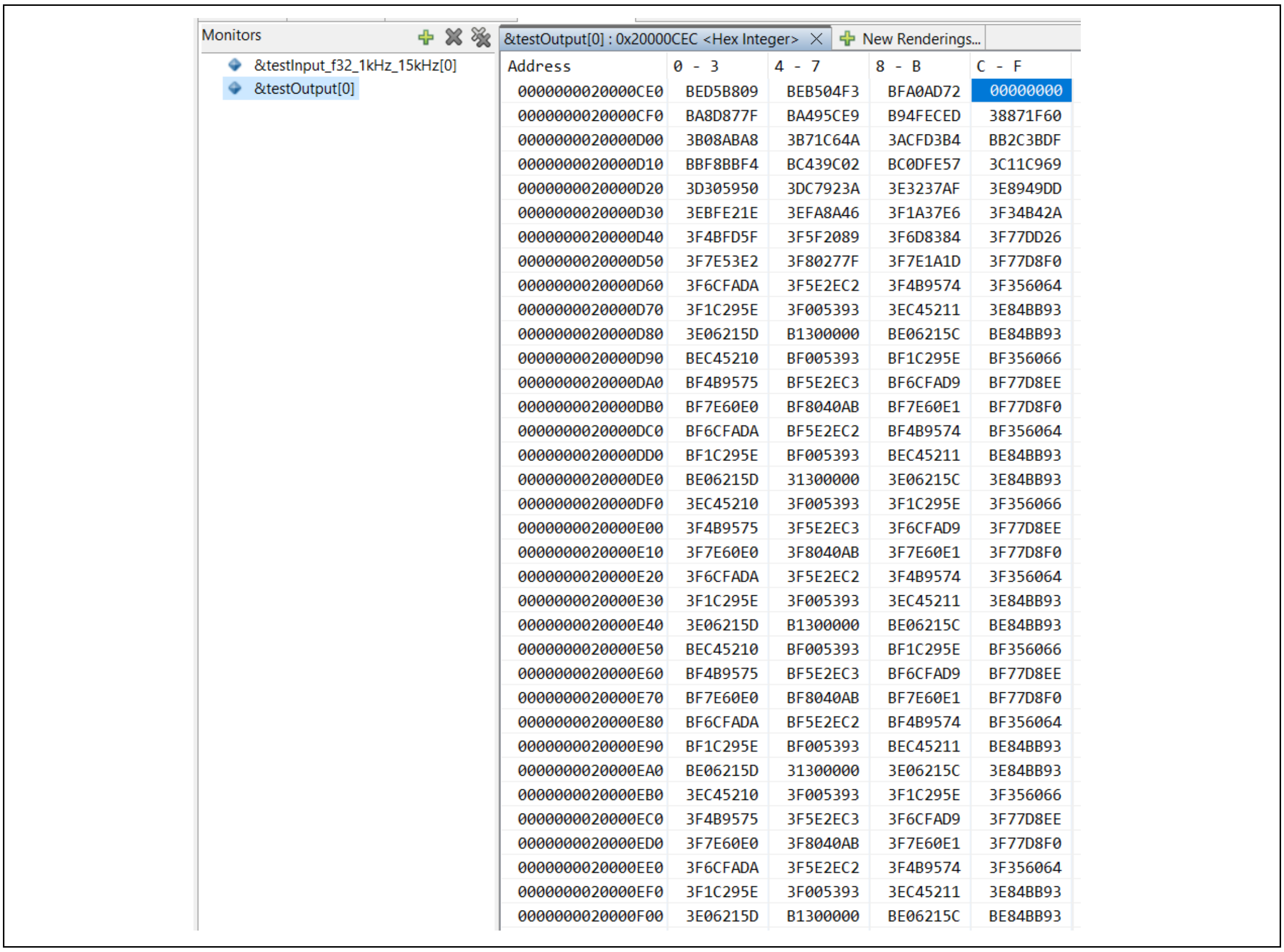


Figure 7. testOutput Buffer of Lowpass Filter Example

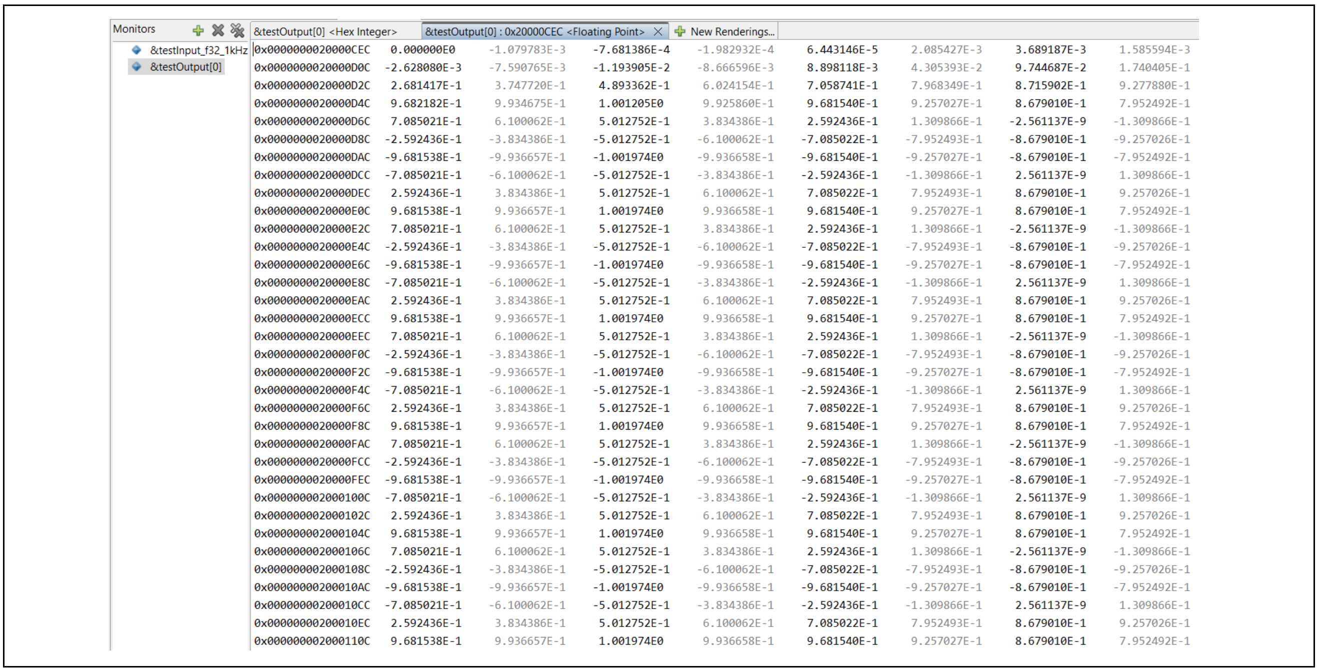


Figure 8. testOutput Buffer in Floating Point format of Lowpass Filter Example

The testInput and testOutput buffers can be visually shown using plotting 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** to copy. You need to manually edit the data to be able to use it with plotting tools.

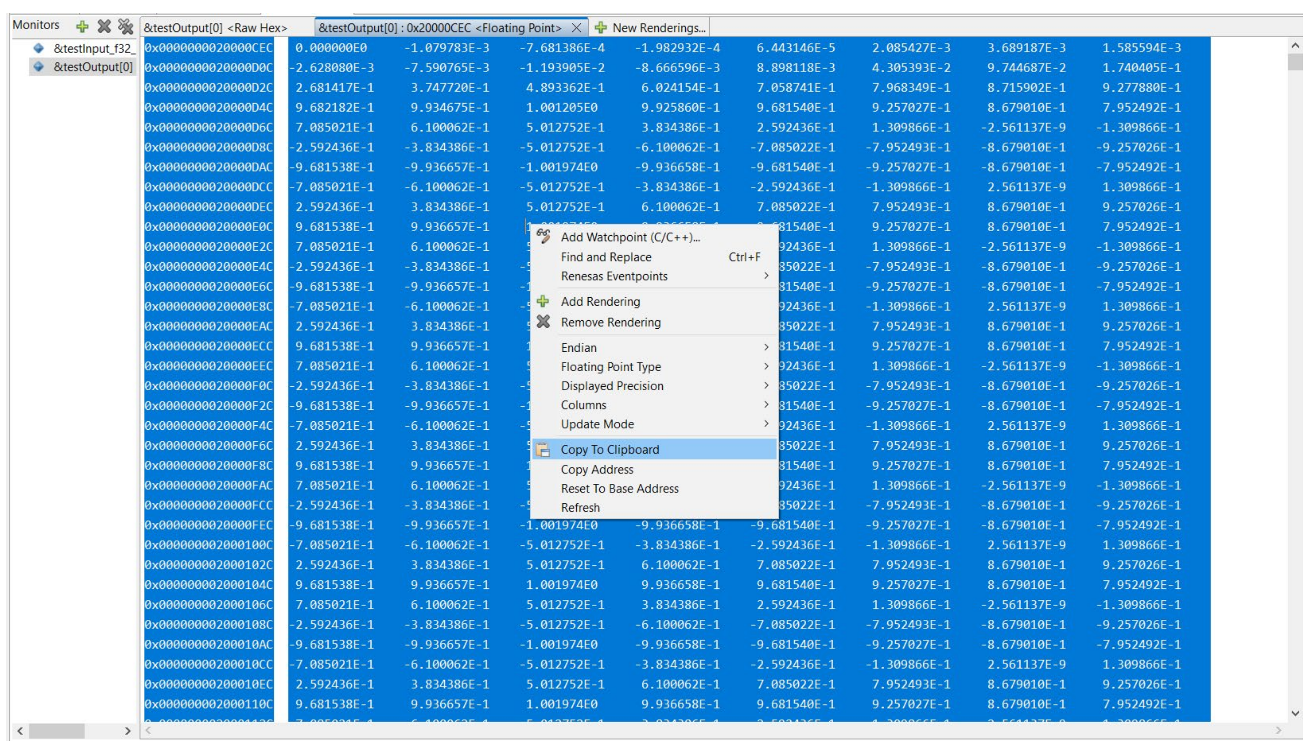


Figure 9. Copy testOutput Buffer to Clipboard

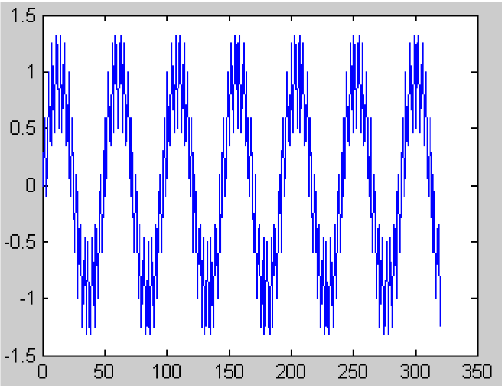


Figure 10. Waveform of Input Buffer of Lowpass Filter Example

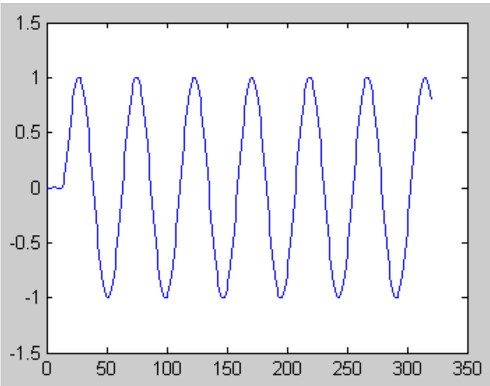


Figure 11. Waveform of Output Buffer of Lowpass Filter Example

2.2 Project Settings

The following settings are needed to get the best performance from the Cortex-M33 core with DSP extension.

Parameter	Default	Setting	Notes
-mfloat-abi	hard	hard	MCU has hardware support
-mfpu	fpv5-sp-d16	fpv5-sp-d16	MCU supports this version
semihosting-enable	false	false	Not use semihosting
-march	Toolchain default	armv8-m.main+dsp	MCU has DSP support
Optimization	-O2	-O3	Optimize most
BSP/Heap Size	0x0	0x200	Heap size
Miscellaneous	false	Use float with nano printf	GNU ARM Cross Linker

The following settings are needed to get the best performance from the Cortex-M23 core with DSP extension.

Parameter	Default	Setting	Notes
-mfloat-abi	hard	hard	MCU has hardware support
-mfpu	Toolchain default	Toolchain default	Toolchain default
semihosting-enable	false	false	Not use semihosting
-march	Toolchain default	Toolchain default	MCU has DSP support
Optimization	-O2	-O2	Optimize more
BSP/Heap Size	0x0	0x100	Heap size
Miscellaneous	false	Use float with nano printf	GNU ARM Cross Linker

The following settings are needed to get the best performance from the Cortex-M85 core with DSP extension.

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	-O2	-O2	Optimize more
BSP/Heap Size	0x0	0x100	Heap size

3. Using Arm® Helium™ in DSP Example with RA8 MCU

3.1 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. You can refer to the R01AN7127EU application note on the Renesas website for more details.

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

LLVM Embedded Toolchain for ARM supports Helium™ instructions with the compiler settings by default. When generating a RA8M1 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.

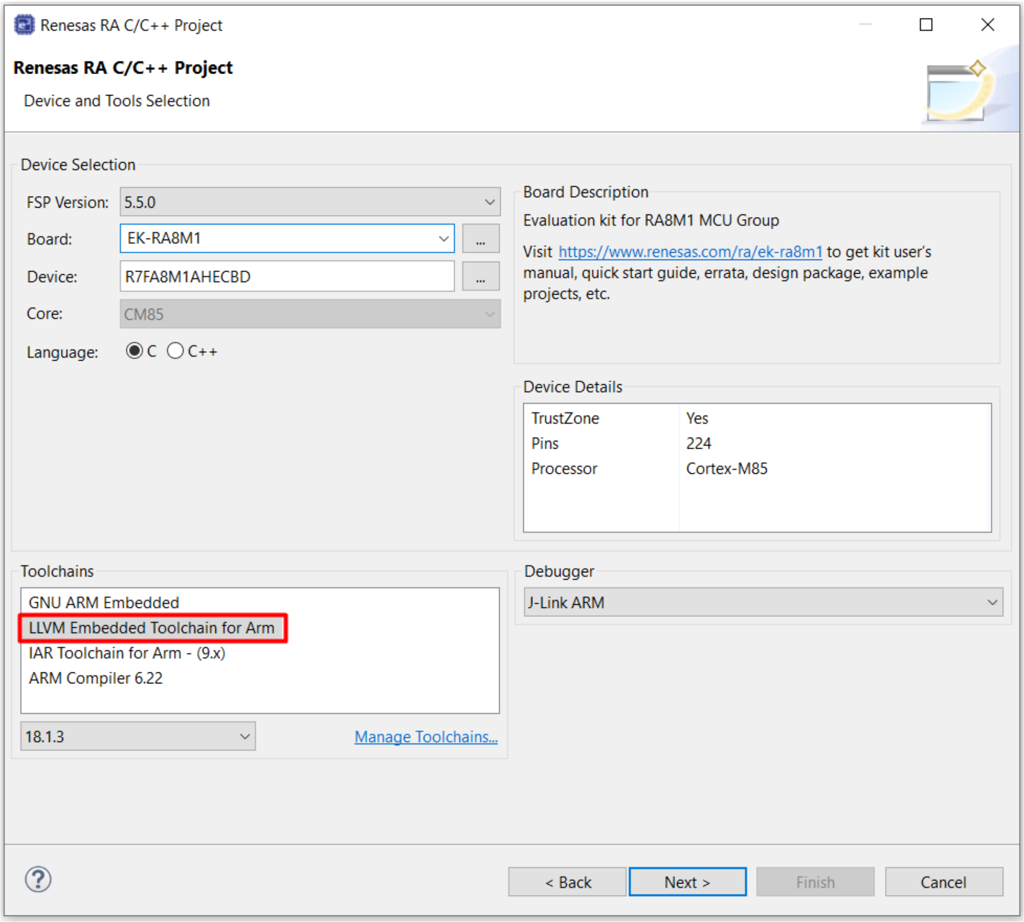


Figure 12. Create an EK-RA8M1 Project with LLVM Toolchain Using e² studio

3.3 Comparison between Renesas MCUs with and without Helium™ Intrinsics

This application note uses a matrix example to compare Renesas MCUs with and without Helium™ Intrinsics. All information regarding Arm MVE Intrinsics can be found at the following link:

https://arm-software.github.io/acle/mve_intrinsics/mve.html

In the matrix example, the below CMSIS DSP functions are used:

- `arm_mat_init_f32()`
- `arm_mat_trans_f32()`
- `arm_mat_mult_f32()`
- `arm_mat_inverse_f32()`

The matrix example demonstrates the use of Matrix Transposition, Matrix Multiplication, and Matrix Inverse functions to apply least squares fitting to input data.

Figure 13 shows MVE Instructions are generated by the LLVM Embedded Toolchain for Arm.

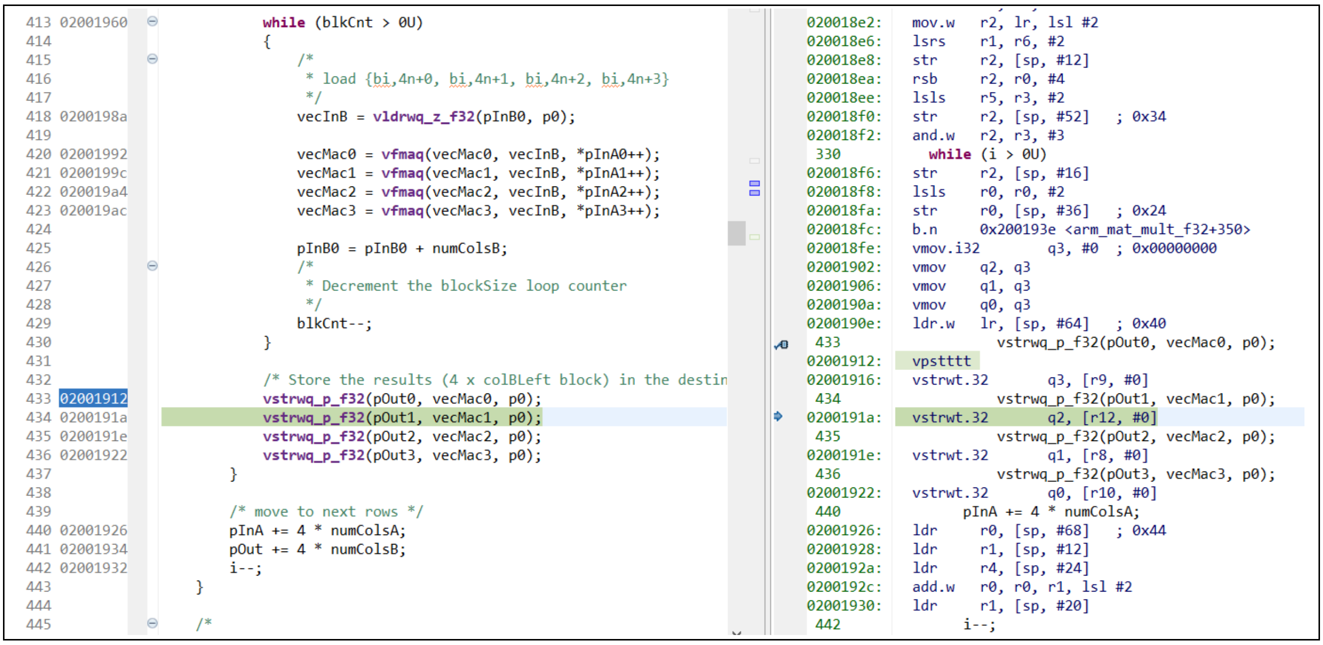


Figure 13. arm_mat_mult_f32 Function with Helium™ Code on EK-RA8M1 Board

Figure 14 shows a typical disassembly code without Helium™ Code.

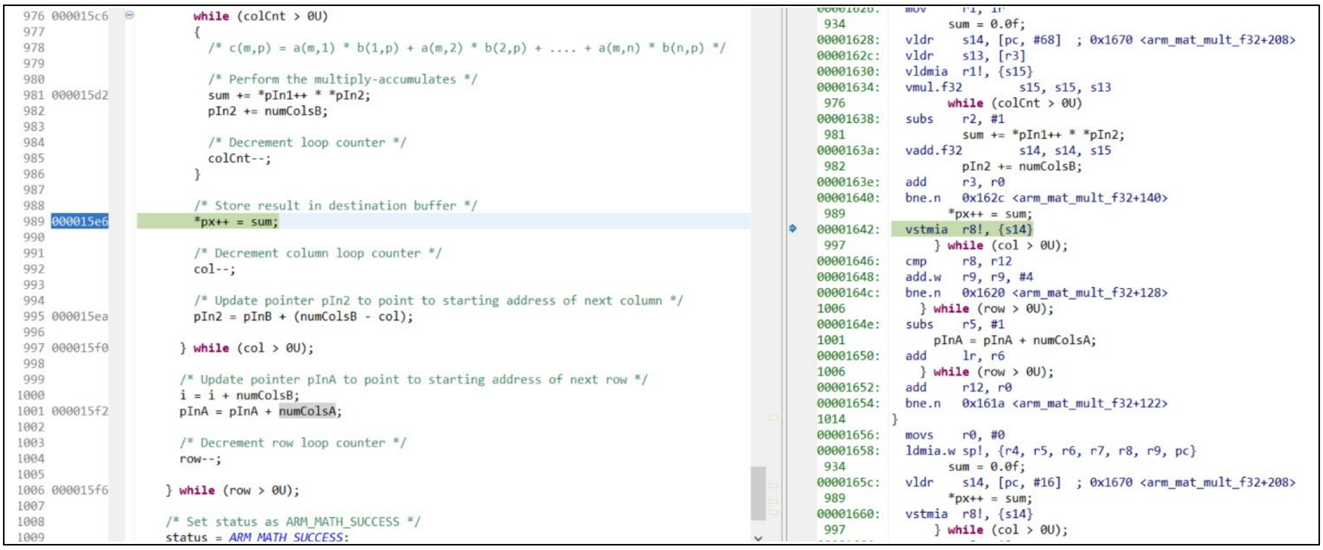


Figure 14. arm_mat_mult_f32 Function without Helium™ Code on EK-RA6M4 Board

The performances between using Helium™ and not using Helium™ are shown in **Figure** 15.

DSP Example	Board	DWT Cycle	Performance Increase (vs w/o Helium) %
Graphic Equalizer Example	EK-RA6M4	390600	
	EK-RA8M1	216124	80.73
Variance Example	EK-RA6M4	3224	
	EK-RA8M1	2726	18.27

Figure 15. Performance Data between Helium™ and without Helium™

Helium™ Technology excels at parallel processing, performing the same operation simultaneously on multiple data, which is in line with the Graphic Equalizer Example, which is why it offers superior performance compared to the Variance Example. Therefore, to improve the performance of the Variance Example and other DSP Examples, you can refer to section 3.4.

3.4 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.

3.4.1 Tightly Coupled Memory (TCM)

The RA8 family 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 the MCU User Manual for TCM memory address areas.

Figure 16 shows ITCM and DTCM in the Local CPU Subsystem.

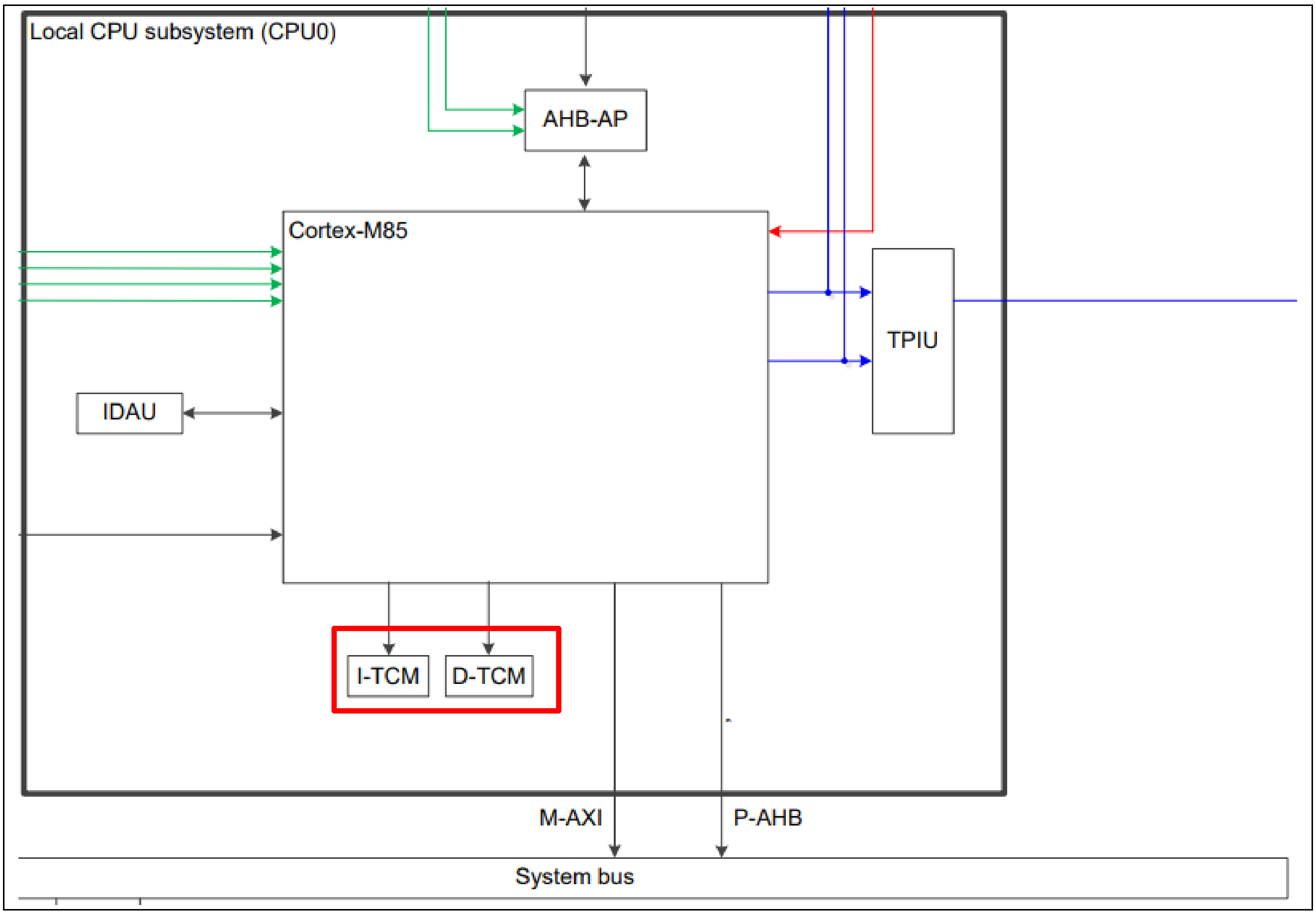


Figure 16. ITCM and DTCM in Local CPU Subsystem

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. Refer to project `arm_fir_example` for more details.

3.4.2 Improve Performance Using DTCM and ITCM

You can place data in the DTCM section (.dtcm_data) in an FSP-based project using the __attribute__ directive, as shown in **Figure** 17.

```
#if defined(ARM_MATH_MVEF) && !defined(ARM_MATH_AUTOVECTORIZE)
const float32_t firCoeffs32[NUM_TAPS_ARRAY_SIZE] __attribute__((section(".dtcm_data"))) = {
  -0.0018225230f, -0.0015879294f, +0.0000000000f, +0.0036977508f, +0.0080754303f, +0.0085302217f, -0.0000000000f, -0.0173976984f,
  -0.0341458607f, -0.0333591565f, +0.0000000000f, +0.0676308395f, +0.1522061835f, +0.2229246956f, +0.2504960933f, +0.2229246956f,
  +0.1522061835f, +0.0676308395f, +0.0000000000f, -0.0333591565f, -0.0341458607f, -0.0173976984f, -0.0000000000f, +0.0085302217f,
  +0.0080754303f, +0.0036977508f, +0.0000000000f, -0.0015879294f, -0.0018225230f, 0.0f,0.0f,0.0f
};
#else
const float32_t firCoeffs32[NUM_TAPS_ARRAY_SIZE] = {
  -0.0018225230f, -0.0015879294f, +0.0000000000f, +0.0036977508f, +0.0080754303f, +0.0085302217f, -0.0000000000f, -0.0173976984f,
  -0.0341458607f, -0.0333591565f, +0.0000000000f, +0.0676308395f, +0.1522061835f, +0.2229246956f, +0.2504960933f, +0.2229246956f,
  +0.1522061835f, +0.0676308395f, +0.0000000000f, -0.0333591565f, -0.0341458607f, -0.0173976984f, -0.0000000000f, +0.0085302217f,
  +0.0080754303f, +0.0036977508f, +0.0000000000f, -0.0015879294f, -0.0018225230f
};
#endif
```

Figure 17. Placing Variables and Data in the DTCM Section

The above data placement can be confirmed using the memory map (*.map) file generated by the compiler, as shown in **Figure** 18

```
20000000  2002bd0       80   16 .dtcm_data
20000000  2002bd0        0    1         __tz_DTCM_S = ABSOLUTE ( __DTCM_START )
20000000  2002bd0        0    1         __dtcm_data_start = .
20000000  2002bd0       80    4         ./src/arm_fir_example/arm_fir_example_f32.o:(.dtcm_data)
20000000  2002bd0       80    1                 firCoeffs32
20000080  2002c50        0    1         . = ALIGN ( 8 )
20000080  2002c50        0    1         __dtcm_data_end = .
20000080  2002c50        0    1 . = __dtcm_data_end
```

Figure 18. Example of Variables and Data Placed in DTCM Area in Memory Map

You can also place program instructions in the ITCM section (.itcm_data) using the __attribute__ directive. Figure 19 shows an example of a modification to place the function

\ra\arm\CMSIS-DSP\Include\dsp\filtering_functions.h\arm_fir_init_f32 in the ITCM area.

```
/**
 * @brief  Initialization function for the floating-point FIR filter.
 * @param[in,out] S          points to an instance of the floating-point FIR filter structure.
 * @param[in]     numTaps    Number of filter coefficients in the filter.
 * @param[in]     pCoeffs    points to the filter coefficients.
 * @param[in]     pState     points to the state buffer.
 * @param[in]     blockSize  number of samples that are processed at a time.
 */
void __attribute__((section(".itcm_data"))) arm_fir_init_f32 (
      arm_fir_instance_f32 * S,
      uint16_t numTaps,
const float32_t * pCoeffs,
      float32_t * pState,
      uint32_t blockSize);
```

Figure 19. Placing a Function in the ITCM Section

You can confirm code placement using the memory map (*.map) file generated by the compiler, as shown in **Figure** 20

```
 0  20021a0       30   16 .itcm_data
 0  20021a0        0    1         __tz_ITCM_S = ABSOLUTE ( __ITCM_START )
 0  20021a0        0    1         __itcm_data_start = .
 0  20021a0        1    1         BYTE ( 0xFF )
 2  20021a2       22    2         ./ra/arm/CMSIS-DSP/Source/FilteringFunctions/arm_fir_init_f32.o:(.itcm_data)
 2  20021a2        0    1                 $t.0
 3  20021a3       22    1                 arm_fir_init_f32
24  20021c4        a    4         <internal>:(.text.thunk)
2e  20021ce        2    1         . = ALIGN ( 8 )
30  20021d0        0    1         __itcm_data_end = .
```

Figure 20. Example of DSP Function in ITCM Area in Memory Map

4. Performance Measurement

4.1 Enable Performance Measurement

The DSP_INSTRUCTION_BENCH macro in arm_dsp_example_config.h is used to enable example code to measure the performance of 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 the measurement. Refer to section 4.2 for more details.

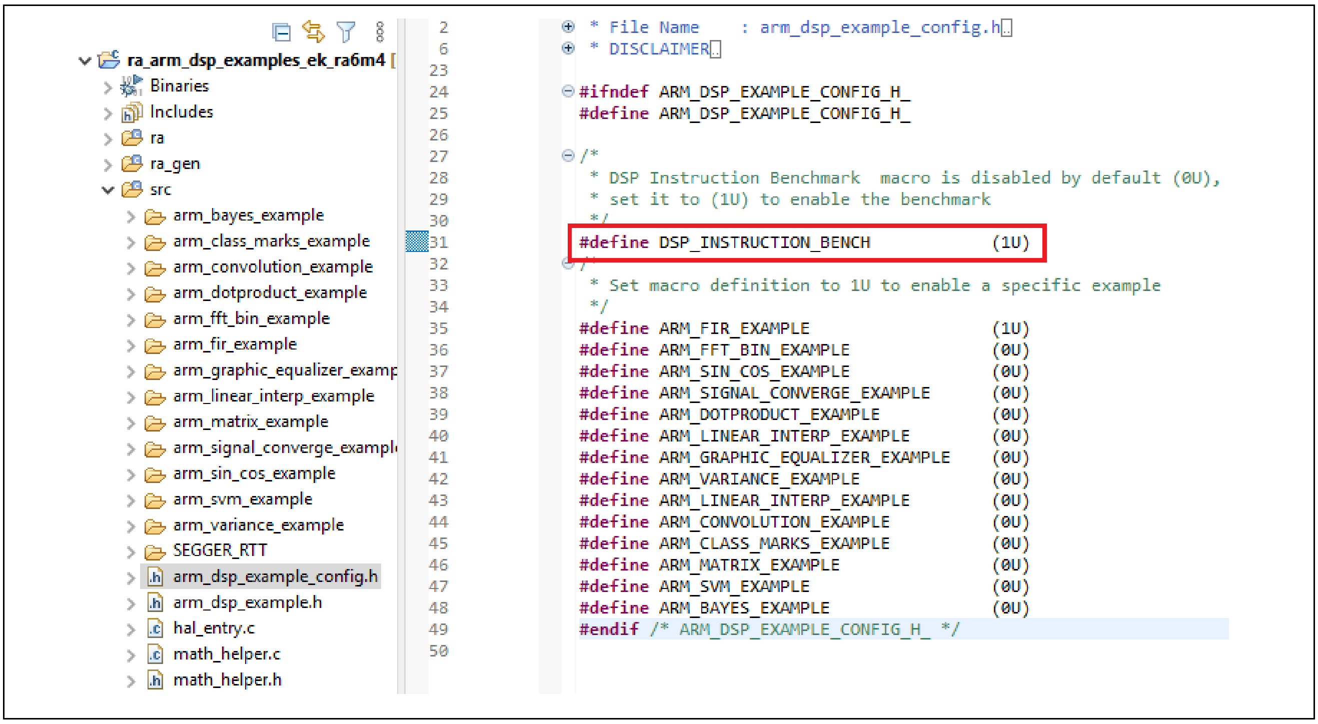


Figure 21. Enable Performance Measurement in arm_dsp_example_config.h

4.2 Adding Supporting Code to Measure DWT Cycles

Store the DWT counter (DWT->CYCCNT) before and after calling the DSP function.

```c
#if DSP_INSTRUCTION_BENCH //Performance Benchmark
  uint32_t ts_0      = 0;
  uint32_t ts_1      = 0;
  uint32_t dwt_cycle = 0;
  ts_0 = DWT->CYCCNT;
  R_LED_LED415ON();
#endif
  /* ------------------------------------------------------------------
  ** Call the FIR process function for every blockSize samples
  ** ------------------------------------------------------------------ */
  for(i=0; i < numBlocks; i++)
  {
    arm_fir_f32(&S, inputF32 + (i * blockSize), outputF32 + (i * blockSize), blockSize);
  }

#if DSP_INSTRUCTION_BENCH //Performance Benchmark
  ts_1 = DWT->CYCCNT;
  R_LED_LED415OFF();
#endif
```

Figure 22. Store DWT Before and After Calling DSP Functions

The execution time of a specific DSP function call can be calculated based on DWT cycles and MCU system clock frequency using the following formula.

Time = DWT cycle x (1/System Clock)

```
#if  DSP_INSTRUCTION_BENCH  //Performance Benchmark
    /* Calculate DW cycles */
    dwt_cycle = ts_1 > ts_0? ts_1 - ts_0 : ~(ts_0 - ts_1);
    APP_PRINT("DWT Clock Cycle: %d\n", dwt_cycle);

    /* Calculate Execution Time */
    char output_exec_time[16];
    memset(output_exec_time, 0, sizeof(output_exec_time));

    uint32_t system_clock_kHz = R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_ICLK) / 1000;
    float32_t exec_time = ((float32_t)dwt_cycle * ((float32_t)1 / (float32_t)system_clock_kHz));

    int res = snprintf(output_exec_time, 16, "%f", exec_time);
    if (res > 0)
    {
        APP_PRINT("Execution Time: %s (ms)\n", output_exec_time);
    }
#endif
    while (1);                      /* main function does not return */
#endif
  }
}
```

Figure 23. Calculate Execution Time on EK-RA6M4

```
#if  DSP_INSTRUCTION_BENCH  //Performance Benchmark
    /* Calculate DW cycles */
    dwt_cycle = ts_1 > ts_0? ts_1 - ts_0 : ~(ts_0 - ts_1);
    APP_PRINT("DWT Clock Cycle: %d\n", dwt_cycle);

    /* Calculate Execution Time */
    char output_exec_time[16];
    memset(output_exec_time, 0, sizeof(output_exec_time));

    uint32_t system_clock_kHz = R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_CPUCLK) / 1000;
    float32_t exec_time = ((float32_t)dwt_cycle * ((float32_t)1 / (float32_t)system_clock_kHz));

    int res = snprintf(output_exec_time, 16, "%f", exec_time);
    if (res > 0)
    {
        APP_PRINT("Execution Time: %s (ms)\n", output_exec_time);
    }
#endif
    while (1);                      /* main function does not return */
#endif
  }
}
```

Figure 24. Calculate Execution Time on EK-RA8M1

The Execution Time results are shown in Figure 23 and Figure 24 are similar to the result in Figure 27.

Note:

- DWT cycles are the count of MCU execution cycles.
- For the EK-RA6M4, the system clock is ICLK, set to 200MHz in this application project.
- For the EK-RA8M1, the system clock is CPUCLK, set to 480MHz in this application project.

```
#if  DSP_INSTRUCTION_BENCH  //Performance Benchmark
    /* Calculate DW cycles */
    dwt_cycle = ts_1 > ts_0? ts_1 - ts_0 : ~(ts_0 - ts_1);
    APP_PRINT("DWT Clock Cycle: %d\n", dwt_cycle);
#endif
```

Figure 25. Calculate DWT Cycle

Note: For the EK-RA2E1(Arm Cortex-M23) does not support DWT capabilities.

4.2.1 Adding LED Function Code to Measure DSP Processing Interval

The LED1 on the RA EK kits can be probed to measure the time taken to complete the signal processing operations. The LED1 is turned on before starting DSP operations and turned off after. This simple analysis technique is useful when a DWT timer is not available with the CPU or preferred for measurement. Refer to arm_fir_example_f32.c for an example of this implementation, as shown in Figure 26.

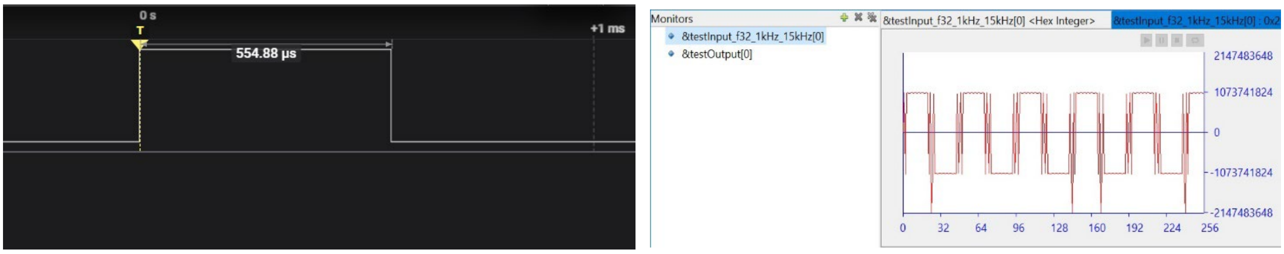
```
#if  DSP_INSTRUCTION_BENCH  //Performance Benchmark
  uint32_t ts_0      = 0;
  uint32_t ts_1      = 0;
  uint32_t dwt_cycle = 0;
  ts_0 = DWT->CYCCNT;
  R_LED_LED415ON();
#endif
  /* ----------------------------------------------------------------------
  ** Call the FIR process function for every blockSize samples
  ** ------------------------------------------------------------------- */
  for(i=0; i < numBlocks; i++)
  {
    arm_fir_f32(&S, inputF32 + (i * blockSize), outputF32 + (i * blockSize), blockSize);
  }

#if  DSP_INSTRUCTION_BENCH  //Performance Benchmark
  ts_1 = DWT->CYCCNT;
  R_LED_LED415OFF();
#endif
```

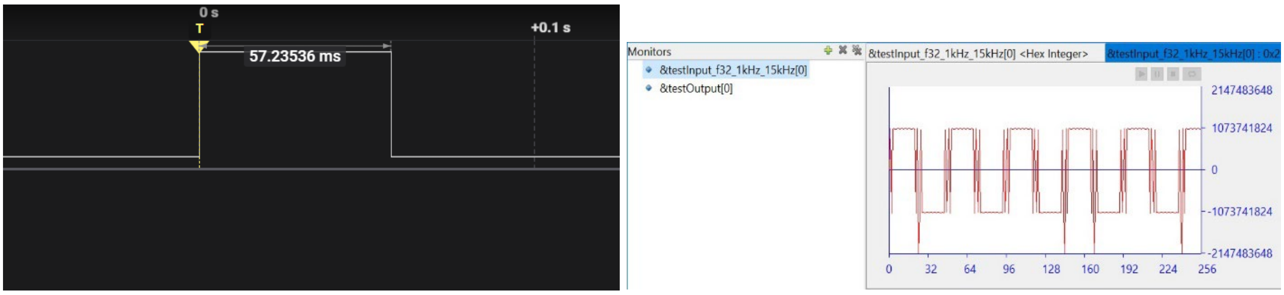
Figure 26. Setting Points of LED Light Start and End for Timing Measurement

- Using a digital oscilloscope and probing the anode of the LED1 of EK-RA2E1 or EK-RA6M4 or EK-RA8M1 trigger the signal “High to Low”.

RA6M4 = 554.88us = 0.55488ms



RA2E1 = 57.24ms



RA8M1 = 241.76us = 0.24176ms

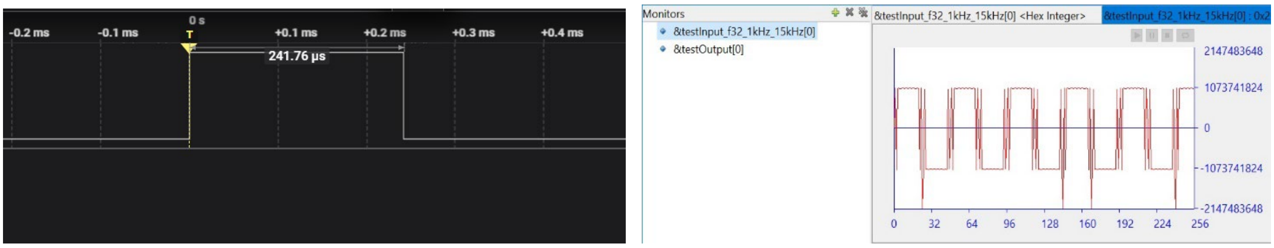


Figure 27. Timing measurement of EK-RA6M4, EK-RA2E1 and EK-RA8M1

4.3 Printing Example Status

Example status and DWT cycle will be sent to SEGGER J-Link RTT Viewer. You can connect to SEGGER RTT Viewer using the following steps.
Launch SEGGER RTT Viewer V7.98b or greater from Windows Start Menu.

Figure 28. Launch J-Link RTT Viewer

Note: For communication between J-Link RTT Viewer and CM33 MCUs such as EK-RA6M4, users need to select the **Address** option of RTT Control Block and enter the memories address shown in Figure 29. This address can be obtained from “ra_arm_dsp_example_ra6m4” under Debug > ra_arm_dsp_example_ra6m4.map. You search for “_SEGGER_RTT”. Every single **arm_dsp_example** will generate a new control block memory address.

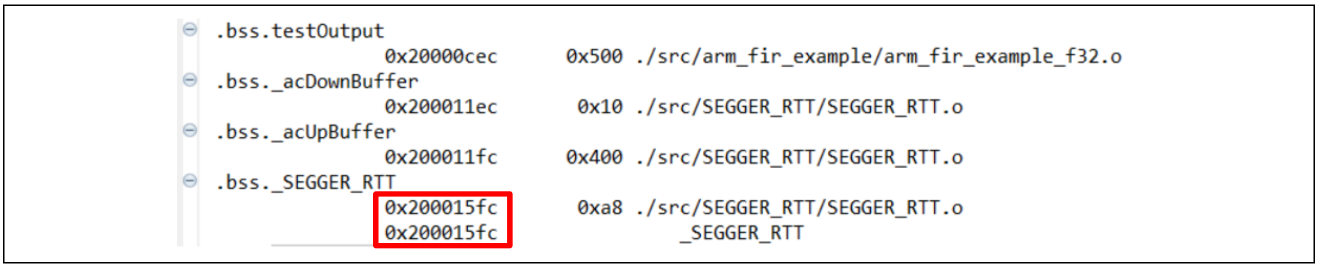


Figure 29. J-Link RTT Viewer Control Block Address

Configure the SEGGER RTT Viewer.

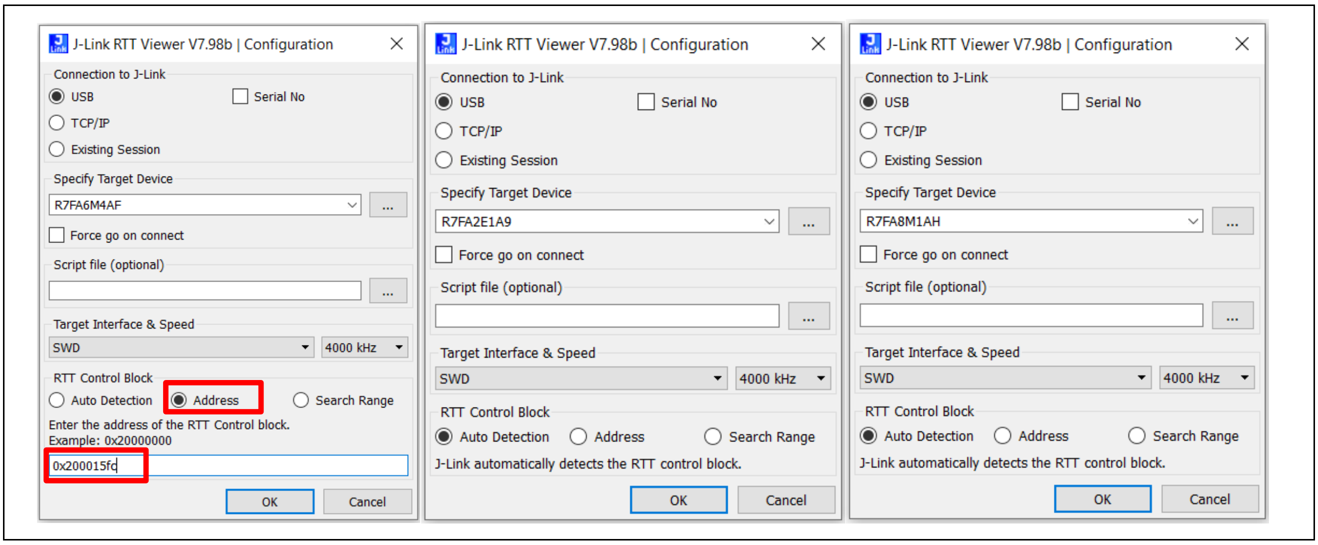


Figure 30. RTT Viewer Settings for EK-RA6M4, EK-RA2E1 and EK-RA8M1

A typical example status output is shown below.

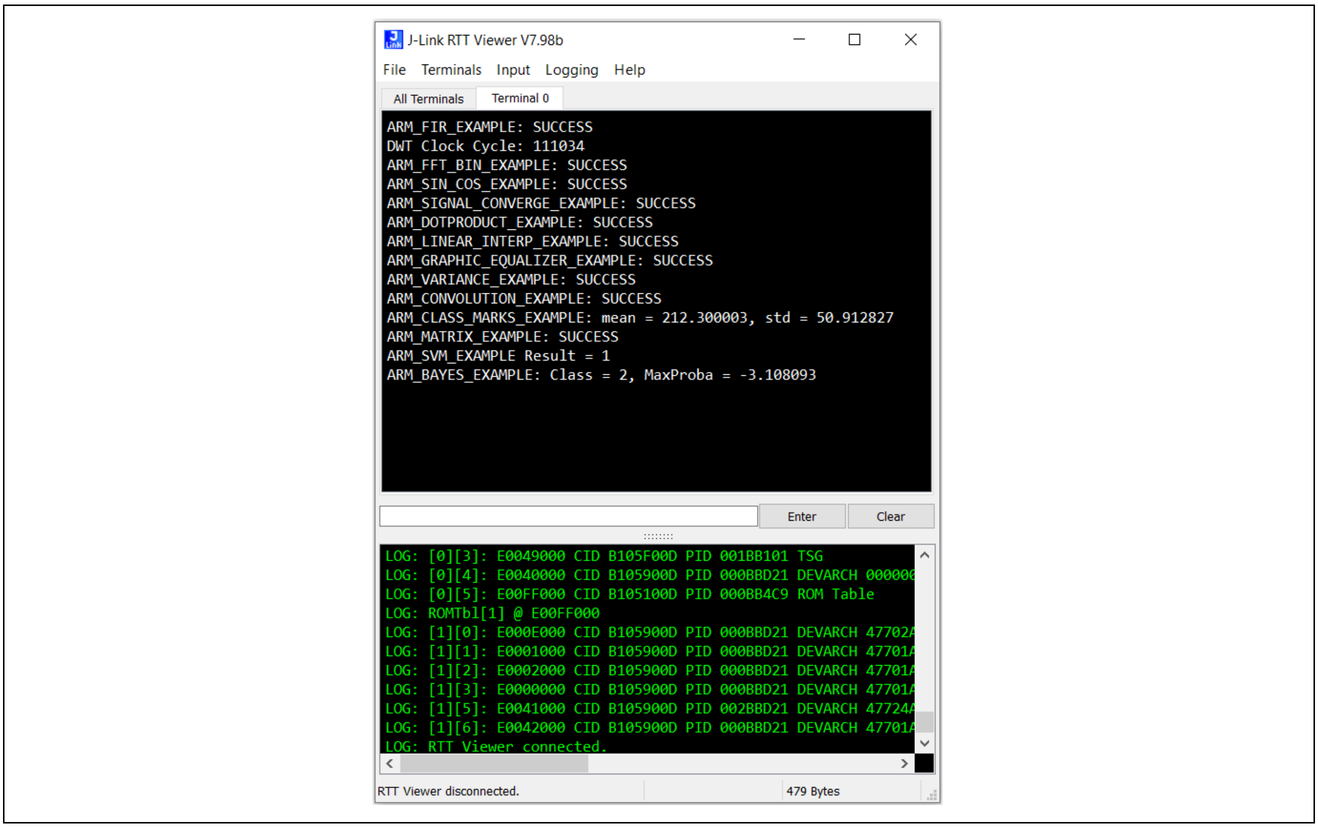


Figure 31. J-Link RTT Viewer Output

5. 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	May.17.21	—	First release document
1.10	March.21.23	— —	Added support and configuration for RA2 series MCU(s). Updated RA6M4 source code project and properties configuration
1.20	Feb.23.24	—	Update to FSP v5.0.0
1.30	Sep.05.24	—	Update to FSP v5.5.0 Added support and configuration for RA8 series MCU(s) Updated source code project and properties configuration

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.

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

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)