Renesas RA Family

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

Introduction

This application note describes the creation of applications with improved performance with Renesas RA8 MCUs using CM85 core with Helium™. It is intended to highlight the performance advantages of the CM85 core, including low latency operation. Helium™, Arm's M-Profile vector extension with integer and floating-point support enables advanced Digital Signal Processing (DSP), Machine Learning (ML) capabilities and helps accelerate compute-intensive applications such as endpoint Artificial Intelligence (AI), ML.

This application note walks you through all the steps necessary to achieve higher performance, including:

- Application overview
- Application highlights
- Tool configuration
- Application confirmation

Required Resources

Development tools and software

- e² studio version: 2024-01.1 (24.1.1)
- LLVM Embedded Toolchain for Arm v17.0.1
- Renesas Flexible Software Package (FSP) v5.2.0 or later.

Hardware

- Renesas EK-RA8M1 kit (RA8M1 MCU Group)

Reference Manuals

- RA Flexible Software Package Documentation Release v5.2.0
- Renesas RA8M1 Group User's Manual Rev.1.10
- EK-RA8M1-v1.0 Schematics

Contents

1. Application Overview…………………………………………………………………..…………3

2. Arm® Cortex®-M85 Core and Helium™ Technology 3
2.1 Arm® Cortex®-M85 core 3
2.2 Renesas RA8 MCU 4
2.3 Single Instruction Multiple Data 5
2.4 Helium™ Applications 6

3. Helium™ Support in Renesas FSP and LLVM Arm Toolchain 8

4. Application Project 9
4.1 Vector Multiply Accumulate Instruction VMLA Example 11
4.2 Vector Instruction VMLADAVA Example 14
4.3 ARM DSP Dot Product Example 17
4.4 Performance Improvement 19
4.4.1 Tightly Coupled Memory (TCM) 19
4.4.2 Improve Performance Using DTCM 22
4.4.3 Improve Performance Using ITCM 23
4.5 Improve Performance by Utilizing Data Cache 24
4.6 Using General Purpose (GPT) Timer for Benchmarking 26

5. Verify the Project 27
5.1 Import The Projects 27
5.2 Build Project 27
5.3 Download and Run Project 29
5.4 Benchmarking Performance 31
5.4.1 VMLA Project HELIUM_VMLA_EK_RA8M1 31
5.4.2 VMLAVADA Project HELIUM_VMLADAVA_EK_RA8M1 32
5.4.3 DSP Dot Product Project HELIUM_DOT_PRODUCT_EK_RA8M1 33

6. Conclusion 34

Revision History 36

1. Application Overview

The application projects accompanying this document showcase the performance advantages of the Renesas RA8 MCU with CM85 core. Helium™ intrinsics and Arm® CMSIS DSP Library functions are benchmarked to highlight the improvements versus the scalar version of these intrinsics.

It also utilizes Tightly Coupled Memory (TCM) and caches together with Helium™ for further performance improvement.

2. Arm® Cortex®-M85 Core and Helium™ Technology

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. There are two variants of MVE, the integer and floating-point variant:

- MVE-I operates on 32-bit, 16-bit, and 8-bit data types, including Q7, Q15, and Q31.
- MVE-F operates on half-precision and single-precision floating-point values.

MVE operations are divided orthogonally in two ways, lanes, and beats.

- Lanes

  Lane is a portion of a vector register or operation. The data that is put into a lane is referred to as an element. Multiple lanes can be executed per beat. There are four beats per vector instruction. The permitted lane widths, and lane operations per beat, are:

  - For a 64-bit lane size, a beat performs half of the lane operation.
  - For a 32-bit lane size, a beat performs a one lane operation.
  - For a 16-bit lane size, a beat performs a two-lane operation.
  - For an 8-bit lane size, a beat performs four lane operations.

- Beats

  Beat is a quarter of an MVE vector operation. Because the vector length is 128 bits, one beat of a vector add instruction equates to computing 32 bits of result data. This is independent of lane width. For example, if a lane width is 8 bits, then a single beat of a vector add instruction would perform four 8-bit additions. The number of beats for each tick describes how much of the architectural state is updated for each architecture tick in the common case. Systems are classified by:

  - In a single-beat system, one beat might occur for each tick.
  - In a dual-beat system, two beats might occur for each tick.
  - In a quad-beat system, four beats might occur for each tick.

Cortex®-M85 implements a dual-beat system, and it supports overlapping up to two beat-wise MVE instructions at any time so that an MVE instruction can be issued after another MVE instruction without additional stall. Refer to Arm® Cortex®-M85 Processor Devices for more information.

2.1 Arm® Cortex®-M85 core

Main features of Arm® Cortex®-M85 core in Renesas RA8 MCU are as follows.

- Maximum operating frequency: up to 480 MHz
- Arm® Cortex®-M85 core

  – Revision: (r0p2-00rel0)

  – ARMv8.1-M architecture profile

  – Armv8-M Security Extension

  – Floating Point Unit (FPU) compliant with the ANSI/IEEE Std 754-2008

Scalar half, single, and double-precision floating-point operation

– M-profile Vector Extension (MVE)

Integer, half-precision, and single-precision floating-point MVE (MVE-F)

– Helium™ technology is M-profile Vector Extension (MVE)

- Arm® Memory Protection Unit (Arm MPU)

– Protected Memory System Architecture (PMSAv8)

– Secure MPU (MPU_S): 8 regions

– Non-secure MPU (MPU_NS): 8 regions

- SysTick timer

– Embeds two Systick timers: Secure instance (SysTick_S) and Non-secure instance (SysTick_NS)

– Driven by CPUCLK or SYSTICKCLK (MOCO/8).

- CoreSight™ ETM-M85

Figure 1 shows the block diagram of Arm® Cortex®-M85 core.

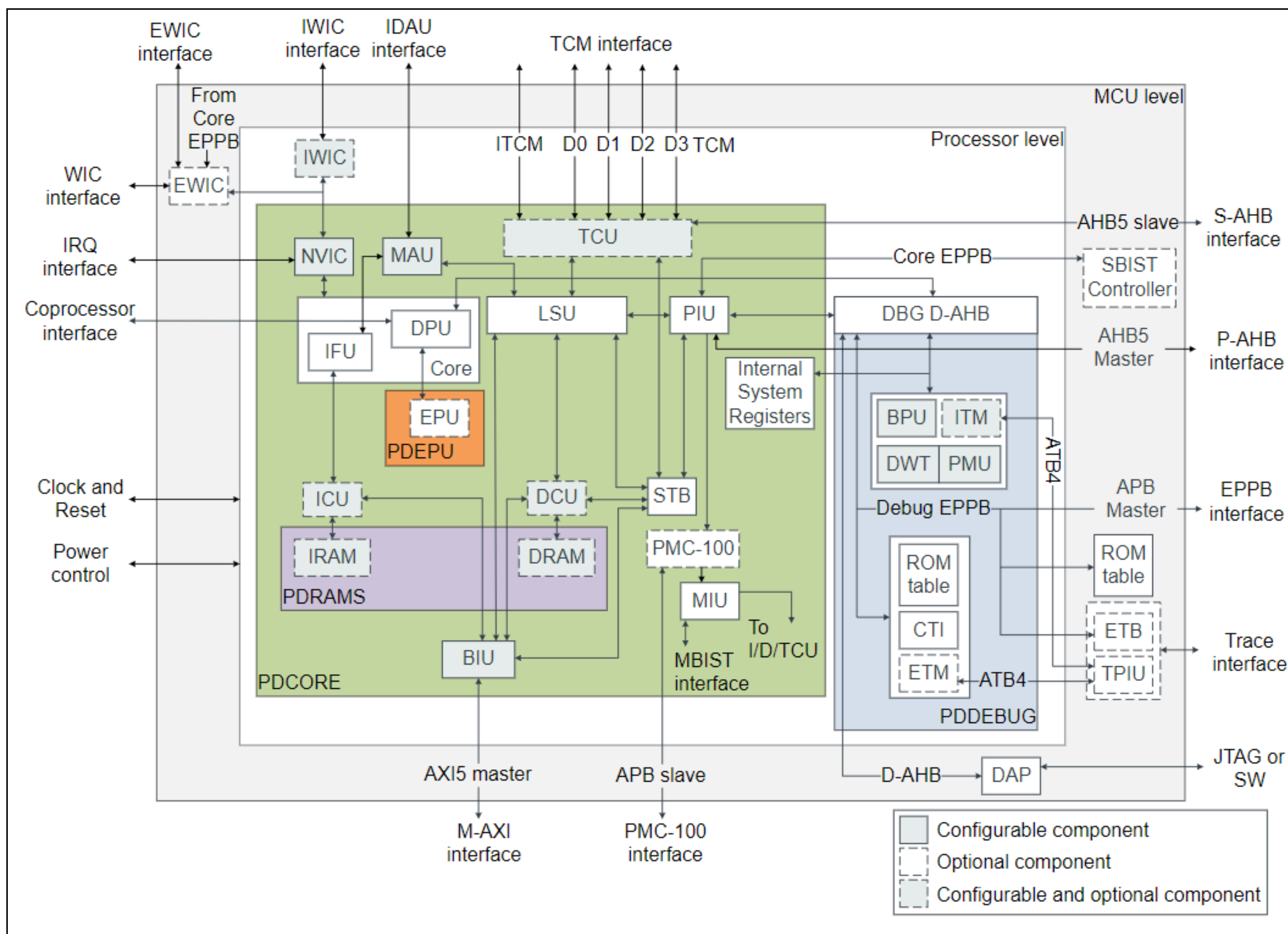

Figure 1. Cortex®-M85 Core Block Diagram

2.2 Renesas RA8 MCU

The RA8M1 MCU group incorporates a high-performance Arm® Cortex®-M85 core as shown in the previous section with Helium™ running up to 480 MHz with the following features.

- Up to 2 MB code flash memory
- 1 MB SRAM (128 KB of TCM RAM, 896 KB of user SRAM)
- Octal Serial Peripheral Interface (OSPI)
- Ethernet MAC Controller (ETHERC), USBFS, USBHS, SD/MMC Host Interface
- Analog peripherals
- Security and safety features.

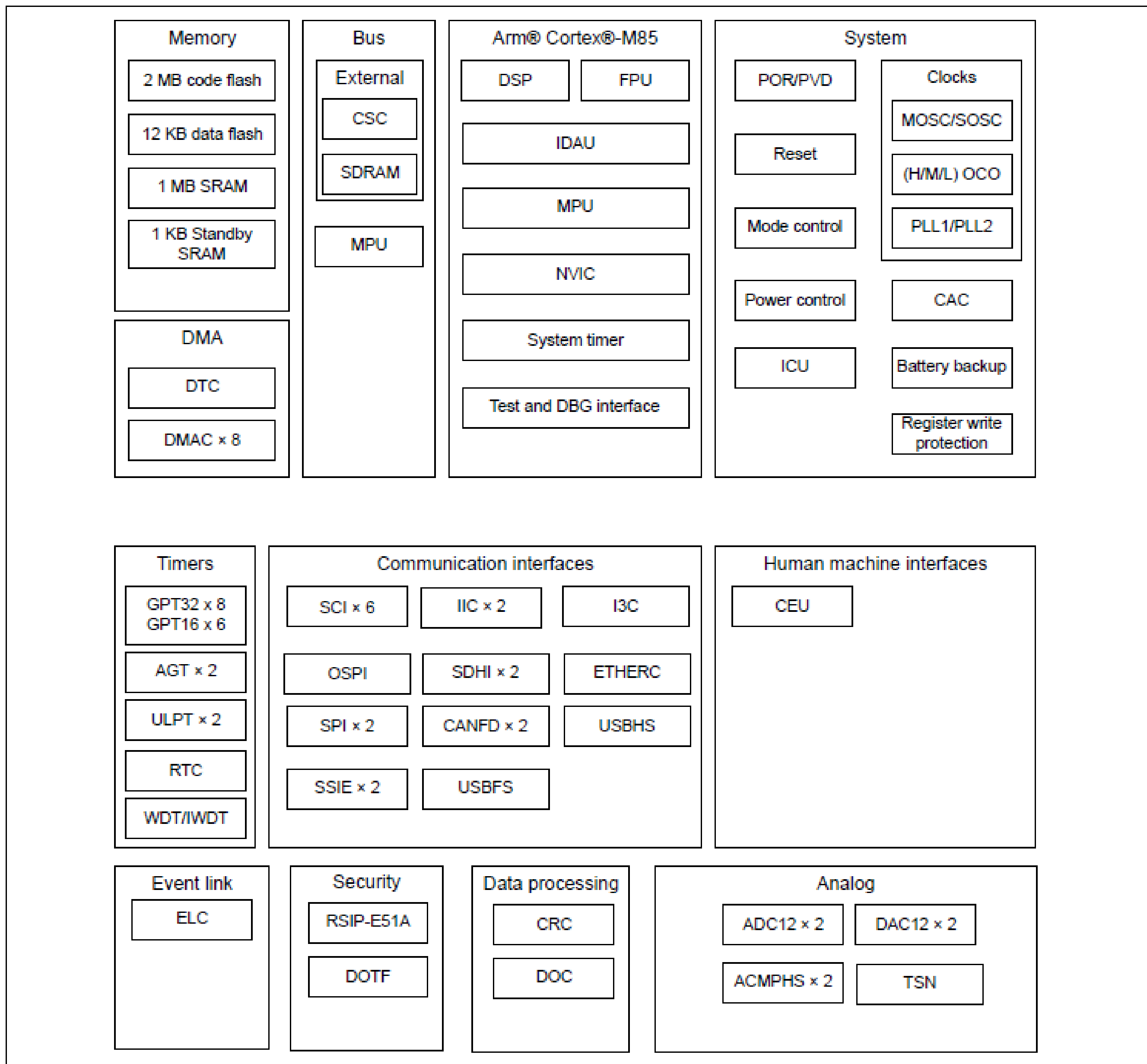

Figure 2. Block Diagram of Renesas RA8M1 MCU

2.3 Single Instruction Multiple Data

Most Arm® instructions are Single Instruction Single Data (SISD) instructions. The SISD instruction only operates on a single data item. It requires multiple instructions to process data items.

The Single Instruction Multiple Data (SIMD), on the other hand, performs the same operation on multiple items of same data type, concurrently. It means invoking/executing a single, multiple operations are being performed simultaneously.

Figure 3 shows the operation of VADD.I32 Qd, Qn, Qm instruction that adds the four pairs of 32-bit data together. Firstly, the four pairs of 32-bit input data are packed into separate lanes in two 128-bit Qn, Qm registers. Then, each lane in the 1st source register is then added to the corresponding lane in the 2nd source register. The results are stored in the same lane in the destination register Qd.

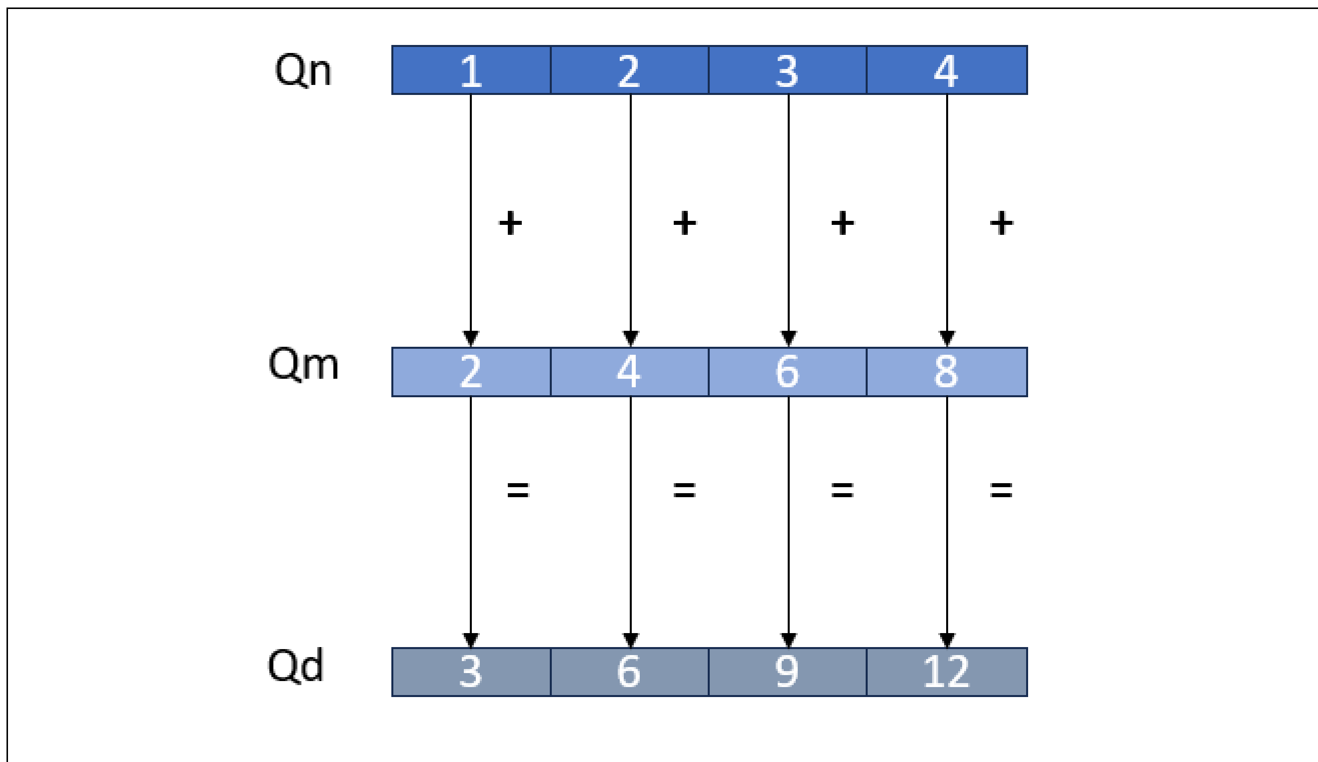

Figure 3. Operation of VADD.I32 Qd, Qn, Qm Instruction

2.4 Helium™ Applications

Digital Signal Processing (DSP) and Machine Learning (ML) are the main target applications for Helium™. Helium™ offers significant performance increases in these applications. Typically, Helium applications are created using Helium™ intrinsics.

Helium™ instructions are made available as intrinsic routines through the arm_mve.h in LLVM Embedded Toolchain for Arm toolchain installation, located in “<e2studio installation folder>\toolchains\llvm_arm\LLVMEmbeddedToolchainForArm-17.0.1-Windows-x86_64\lib\clang\17\include”. They give users access to the Helium™ instructions from C and C++ without the need to write assembly code.

Many functions in CMSIS-DSP and CMSIS-NN libraries have been optimized by Arm to use the Helium™ instructions instead. Renesas FSP supports both libraries, making it easier for users to develop applications based on these libraries. In the FSP configuration, select Arm DSP Library Source (CMSIS5-DSP version 5.9.0 or later) and Arm NN Library Source (CMSIS-NN version 4.1.0 or later) when generating projects to add CMSIS-DSP and CMSIS-NN supports to your project.

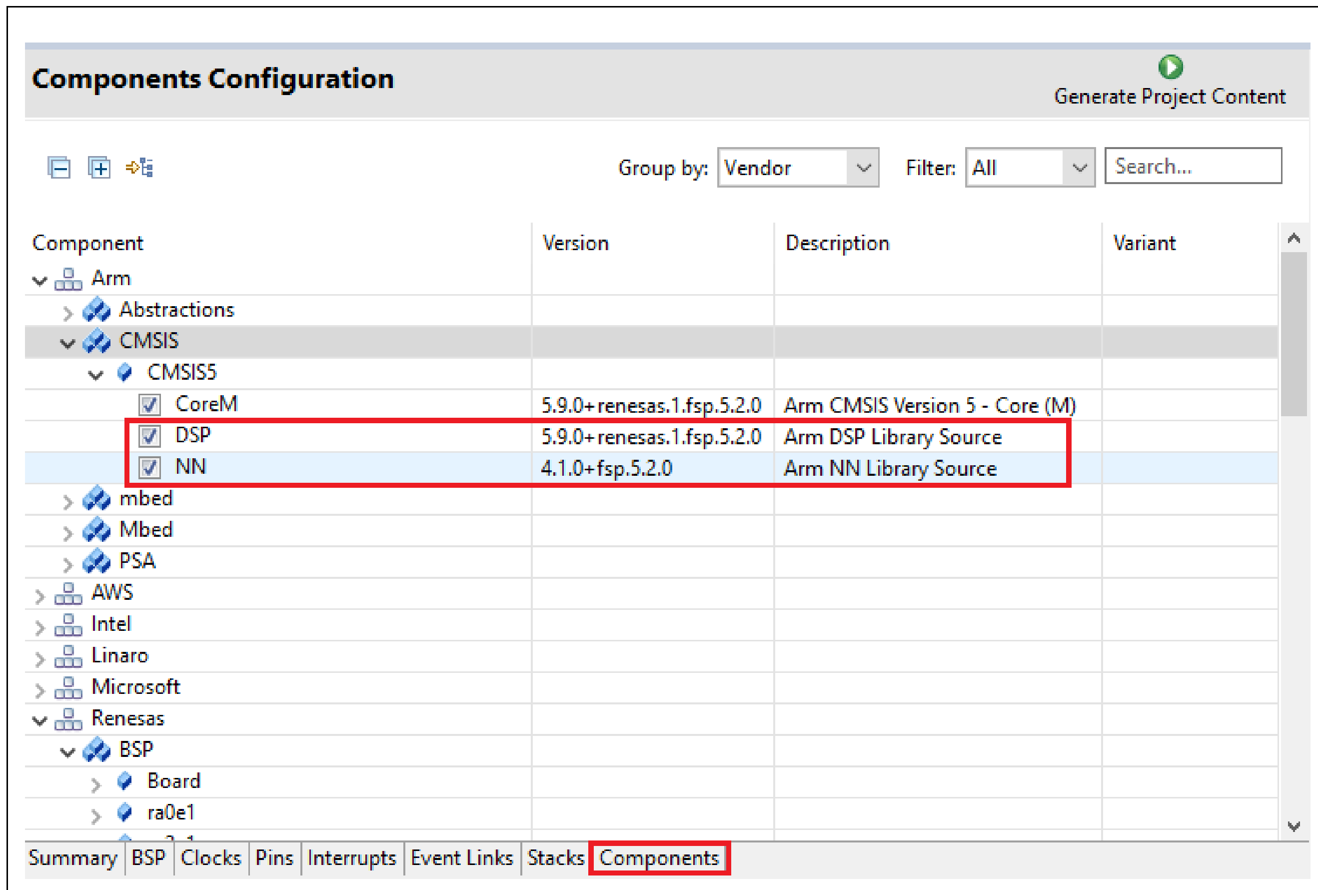

Figure 4. CMSIS-DSP and CMSIS-NN supports in Renesas FSP

CMSIS-DSP and CMSIS-NN can also be added using Stacks tab in FSP configurator, as shown below.

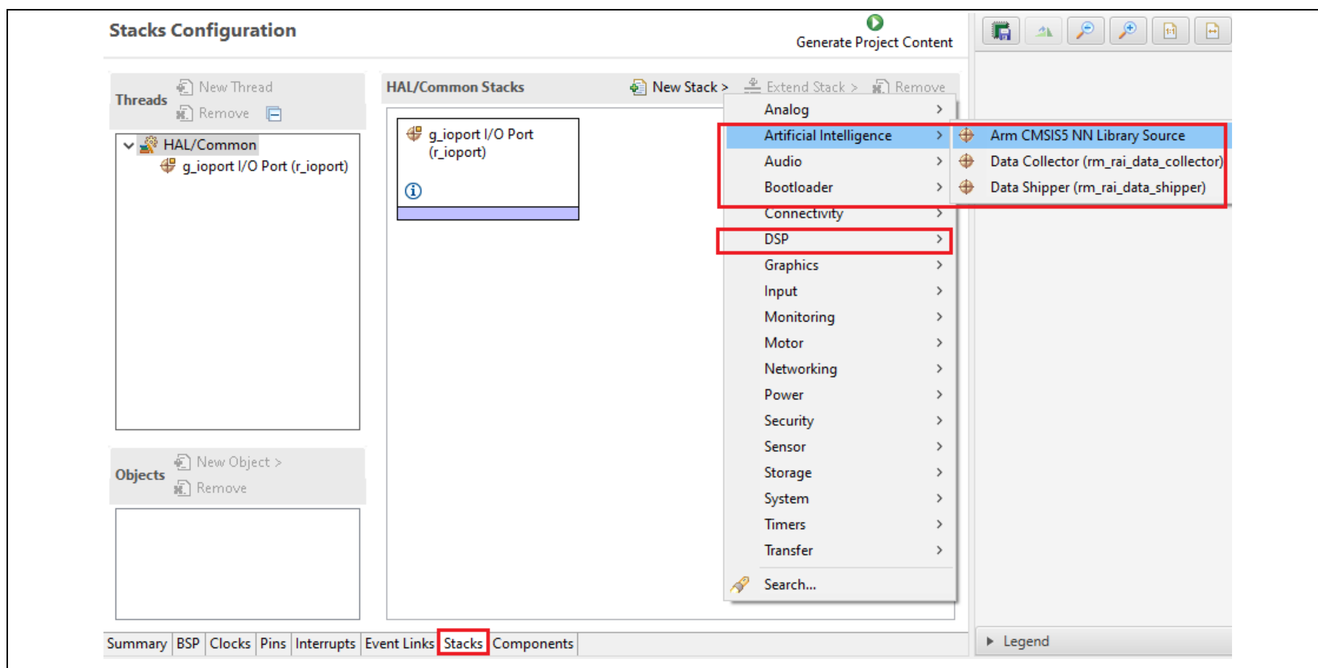

Figure 5. Adding CMSIS-DSP and CMSIS-NN Using Stacks Tab in FSP Configurator

3. Helium™ Support in Renesas FSP and LLVM Arm Toolchain

LLVM Embedded Toolchain for Arm supports Helium™ instructions with the compiler settings. 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. Refer to the Flexible Software Package Documentation for the steps to create a project for RA MCU.

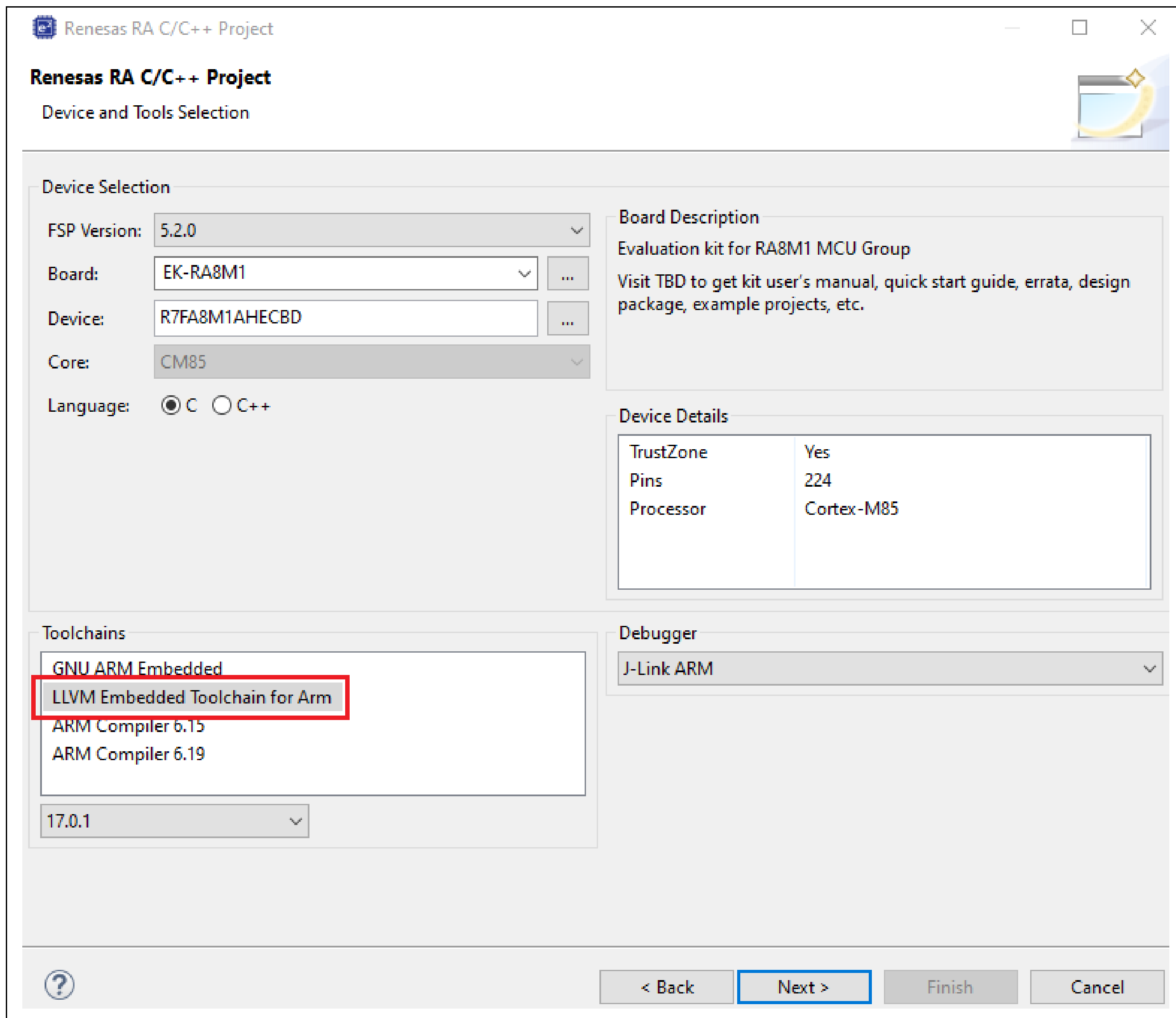

Figure 6. Create an EK-RA8M1 Project using e² studio

The Cortex®-M85 core will be selected in the tool settings, as shown below.

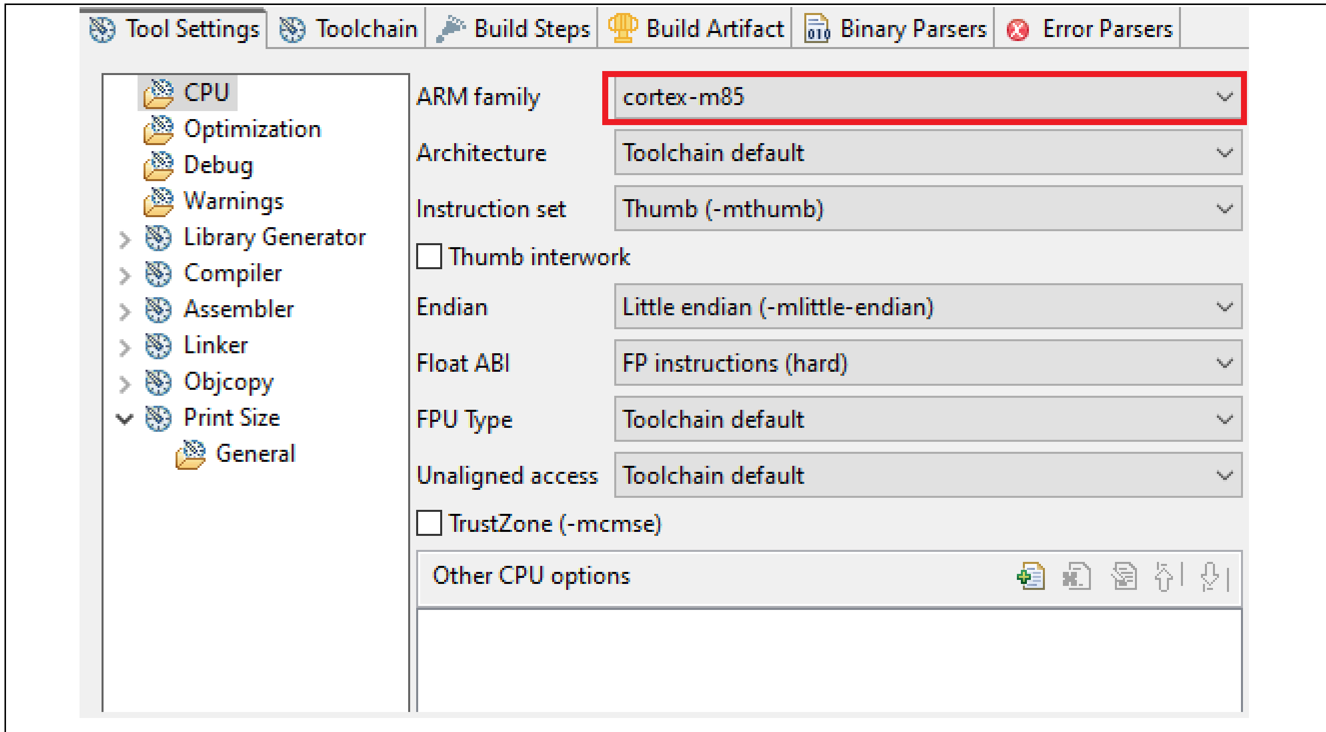

Figure 7. Example of Tool Settings

Even though, the project settings are pre-optimized for Cortex®-M85, they can be customized if needed. Macro definitions can be added to select project configurations to enable and disable some portions of the code in a project. Go to Project->C/C++ Project Settings to change setups for the project if needed.

4. Application Project

There are three projects accompanying this application note. All have the scalar code equivalent to Helium™ functions.

- The Vector Multiply Accumulate (VMLA) and the scalar code equivalent.
- The Vector Multiply Accumulate Add Accumulate Across Vector (VMLADAVA) and the scalar code equivalent.
- The ARM® DSP Dot Product function and the scalar code equivalent.

The projects are configured in various settings to utilize DTCM, ITCM, and cache to showcase the performance improvements of Helium technology compared to scalar code.

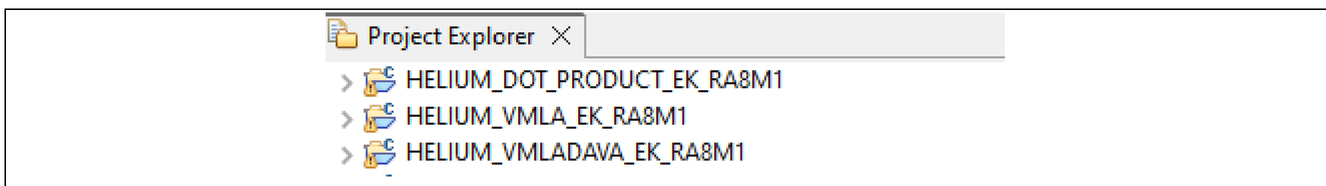

Figure 8. Application Projects in the Workspace

The available configuration for each project is as follows.

Project Configuration	HELIUM_VMLA_EK_RA8M1	HELIUM_VMLADAVA_EK_RA8M1	HELIUM_DOT_PRODUCT_EK_RA8M1
I32_SCALAR(w/o Auto Vectorization)	✓	✓	
I32_SCALAR(w/ Auto Vectorization)	✓	✓	✓
I32_HELIUM	✓	✓	✓
I32_HELIUM_DTCM	✓	✓	✓
I32_HELIUM_ITCM	✓	✓	

Figure 9. Configuration Available in Application Projects

Where I32_SCALAR is for the scalar code, I32_HELIUM is for the Helium code, I32_HELIUM_DTCM is for the Helium code that utilizes DTCM, and I32_HELIUM_ITCM is for the Helium code placed ITCM.

The optimization level of the projects in this application note are set to "-O2".

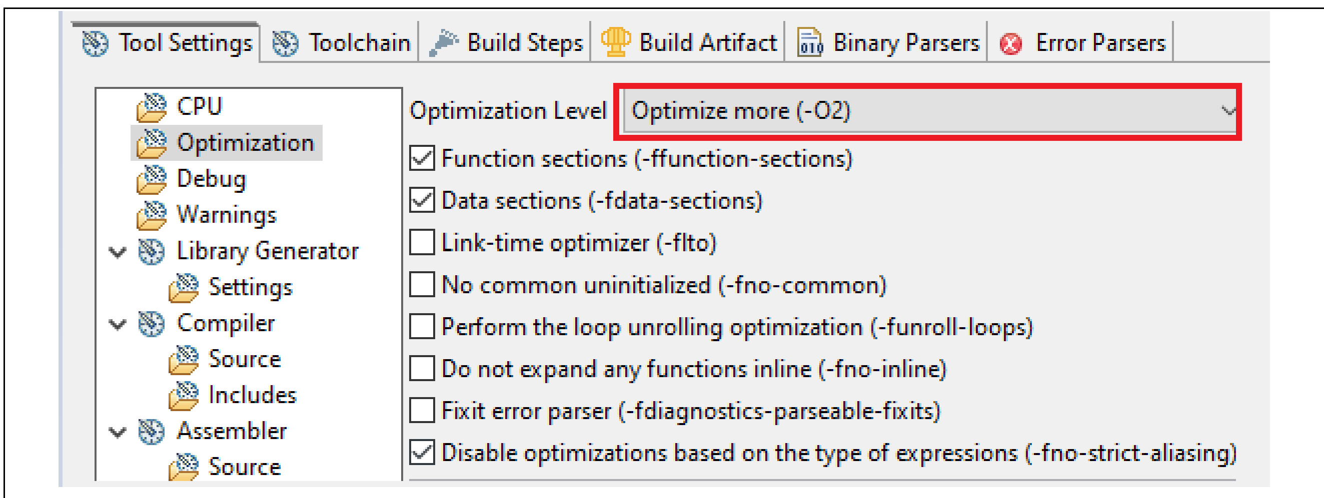

Figure 10. Compiler Optimization Setting

The _CONFIG_HELIUM_ symbol is preset to select scalar operation, Helium operation, or enable the code to utilize DTCM and ITCM together with Helium operation.

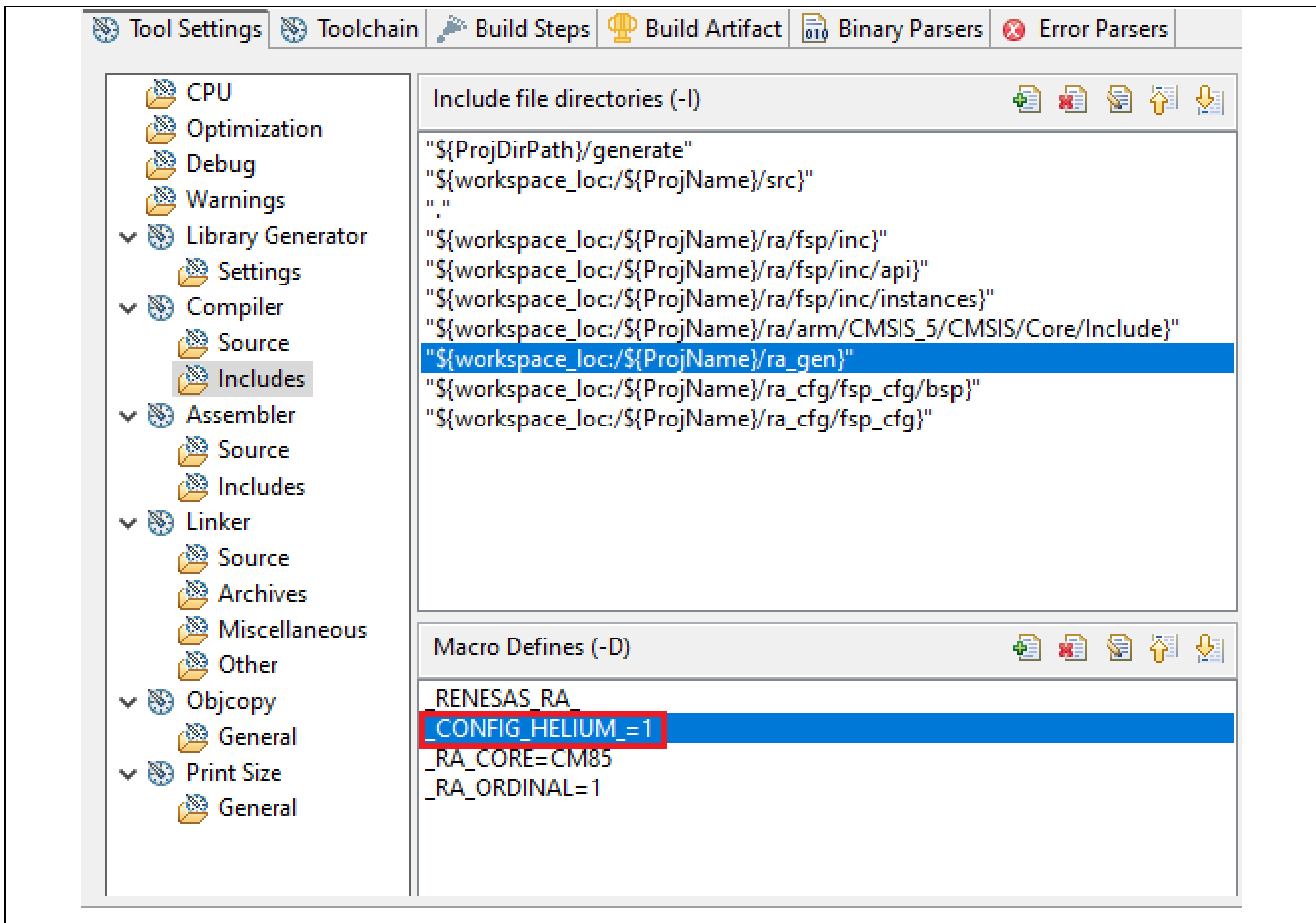

Figure 11. _CONFIG_HELIUM_ symbol is Used to Select Helium Code and Scalar Code Options

4.1 Vector Multiply Accumulate Instruction VMLA Example

In VMLA instruction, each element in the input vector2 is multiplied by the scalar value. The result is added to the respective element of input vector1. The results are stored in the destination register.

The steps of VMLA.S32 Qda, Qn, Rm instruction are shown in the following diagram:

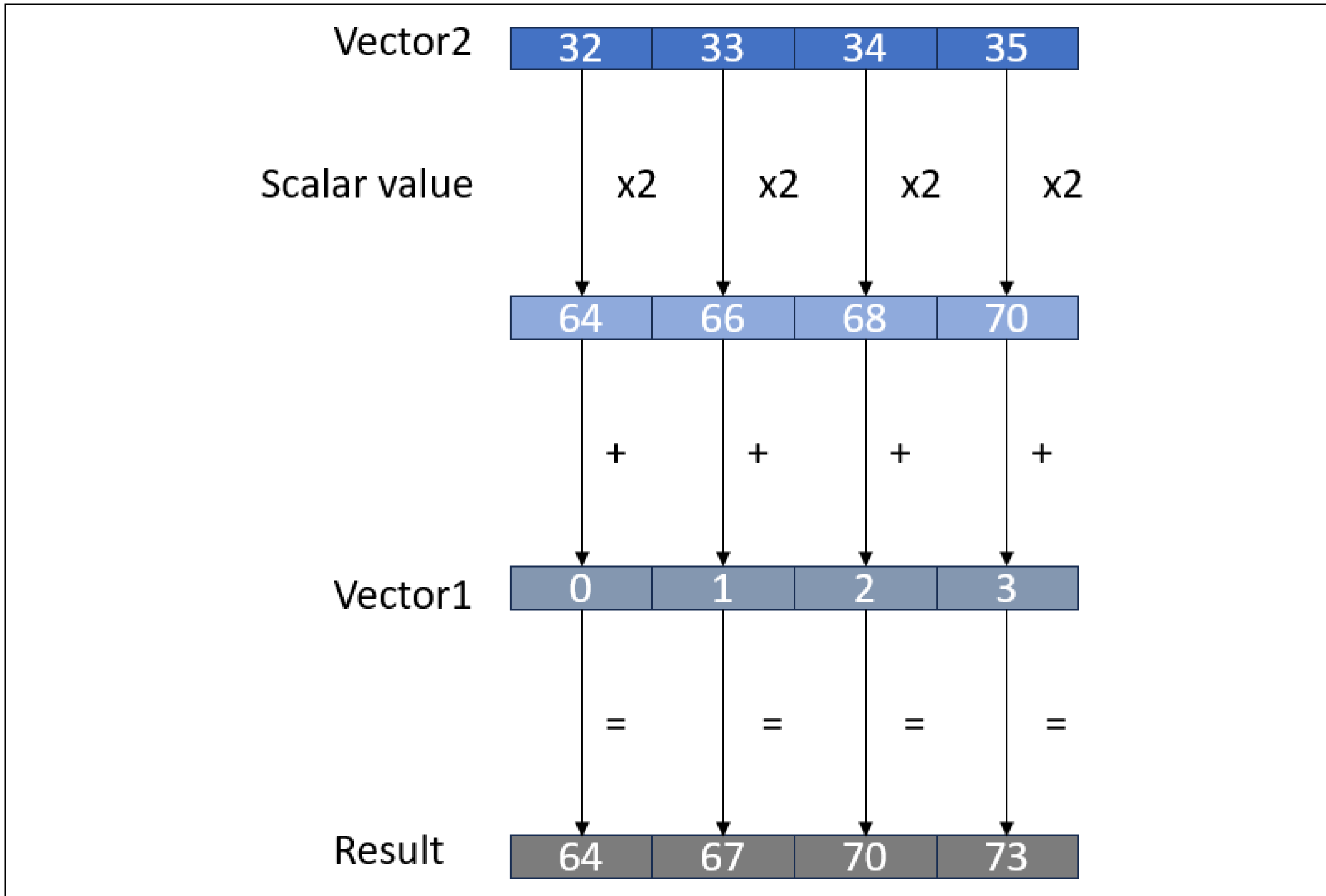

Figure 12. VMLA Operation

The intrinsic function vmlag_n_s32 in Figure 13 is used to showcase the performance of VMLA.S32 Qda, Qn, Rm instruction versus the scalar equivalent.

```
    //Get the timer counter
    ts_cycle_0 = R_GPT0->GTCNT;
    R_GPT0->GTCR  = 1; // Start timer

#if (_CONFIG_HELIUM_ == I32_SCALAR)
    //Perform scalar calculation (Equivalent with the multiply accumulate ins

    for (i = 0; i<DAT_BUF_SIZE; i++)
     {
        data1[i] += (data2[i] * scalarval);
     }
    //Get the timer counter
    ts_cycle_1 = R_GPT0->GTCNT;
#elif (_CONFIG_HELIUM_ == I32_HELIUM) || (_CONFIG_HELIUM_ == I32_HELIUM_DTCM)

#if (_CONFIG_HELIUM_ == I32_HELIUM) || (_CONFIG_HELIUM_ == I32_HELIUM_DTCM)
    //Sine calculating 4 outputs at a time, the loop will be 32/4 = 8 (LOOP_N
    for (i = 0; i<LOOP_NO; i++)
    {
      //Load 4 data from the array
      vector1   = vld1q_s32(p_data1);
      vector2   = vld1q_s32(p_data2);
      //Multiply (vector2*scalarval), add vector1 and store the results
      result[i] = vmlaq_n_s32(vector1, vector2, scalarval);
      //Increase pointers
      p_data1 += 4;
      p_data2 += 4;
     }
    //Get the timer counter
    ts_cycle_1 = R_GPT0->GTCNT;
```

```
136           R_GPT0->GTCR  = 1; // Start timer
02000876:   str.w    r4, [r5, #-28]
0200087a:   movt     r6, #8704        ; 0x2200
0200087e:   movw     r1, #4940        ; 0x134c
02000882:   movw     r2, #5068        ; 0x13cc
02000886:   movt     r1, #512         ; 0x200
0200088a:   movt     r2, #512         ; 0x200
0200088e:   mov      r3, r6
154             vector2   = vld1q_s32(p_data2);
02000890:   vldrw.u32        q0, [r2], #16
153             vector1   = vld1q_s32(p_data1);
02000894:   vldrw.u32        q1, [r1], #16
156             result[i] = vmlaq_n_s32(vector1, vector2, scalarval);
02000898:   vshl.i32         q0, q0, #1
0200089c:   vadd.i32         q0, q0, q1
020008a0:   vstrb.8 q0, [r3], #16
150           for (i = 0; i<LOOP_NO; i++)
020008a4:   le       lr, 0x2000890 <hal_entry+124>
162           ts_cycle_1 = R_GPT0->GTCNT;
020008a8:   movw     r1, #2432        ; 0x980
020008ac:   ldr      r2, [r5, #0]
020008ae:   movt     r1, #8704        ; 0x2200
020008b2:   movs     r4, #0
020008b4:   str      r2, [r1, #0]
172           R_GPT0->GTCR = 0; //Stop timer
020008b6:   str.w    r4, [r5, #-28]
177           APP_PRINT("Timer counter cycle: %d \n", ts_cycle_1 - ts_cycle_0);
020008ba:   ldr      r1, [r1, #0]
```

Figure 13. Example of VMLA Instruction using Intrinsics and Disassembly Code

Figure 14 shows the scalar code equivalent to the Helium code in Figure 13.

```
    R_GPT0->GTCR  = 1; // Start timer

#if (_CONFIG_HELIUM_ == I32_SCALAR)
    //Perform scalar calculation (Equivalen
    for (i = 0; i<DAT_BUF_SIZE; i++)
    {
        data1[i] += (data2[i] * scalarval);
    }
    //Get the timer counter
    ts_cycle_1 = R_GPT0->GTCNT;
#elif (_CONFIG_HELIUM_ == I32_HELIUM) || (_
```

```
136          R_GPT0->GTCR  = 1; // Start timer
02000876:  str.w   r4, [r5, #-28]
0200087a:  movw    r2, #4896        ; 0x1320
0200087e:  movt    r1, #8704        ; 0x2200
02000882:  movt    r2, #512         ; 0x200
142            data1[i] += (data2[i] * scalarval);
02000886:  ldr.w   r3, [r2], #4
0200088a:  ldr     r6, [r1, #0]
0200088c:  add.w   r3, r6, r3, lsl #1
02000890:  str.w   r3, [r1], #4
140          for (i = 0; i<DAT_BUF_SIZE; i++)
02000894:  le      lr, 0x2000886 <hal_entry+114>
```

Figure 14. Example of Scalar Code Equivalent of VMLA and Typical Disassembly Code Without Auto Vectorization

Notes that the LLVM for Arm v17.0.1 supports auto vectorization by default and generates MVE instructions equivalent to scalar code automatically where applicable.

```
    R_GPT0->GTCR  = 1; // Start timer

#if (_CONFIG_HELIUM_ == I32_SCALAR)
    //Perform scalar calculation (Equivalent with t

    for (i = 0; i<DAT_BUF_SIZE; i++)
    {
        data1[i] += (data2[i] * scalarval);
    }
    //Get the timer counter
    ts_cycle_1 = R_GPT0->GTCNT;
#elif (_CONFIG_HELIUM_ == I32_HELIUM) || (_CONFIG_H

#if (_CONFIG_HELIUM_ == I32_HELIUM) || (_CONFIG_HEL
    //Sine calculating 4 outputs at a time, the loo
```

```
136          R_GPT0->GTCR  = 1; // Start timer
02000876:  str.w   r4, [r5, #-28]
0200087a:  movw    r2, #0
0200087e:  movt    r1, #512         ; 0x200
02000882:  movt    r2, #8704        ; 0x2200
142            data1[i] += (data2[i] * scalarval);
02000886:  vldrw.u32      q0, [r1], #16
0200088a:  vldrw.u32      q1, [r2, #0]
0200088e:  vshl.i32       q0, q0, #1
02000892:  vadd.i32       q0, q1, q0
02000896:  vstrb.8 q0, [r2], #16
140          for (i = 0; i<DAT_BUF_SIZE; i++)
0200089a:  le      lr, 0x2000886 <hal_entry+114>
145          ts_cycle_1 = R_GPT0->GTCNT;
```

Figure 15. Example of Scalar Code Equivalent of VMLA and Its Vectorization Code Generated by LLVM for Arm v17.0.1

To disable auto vectorization, add “-fno-vectorize” option to compiler setting as shown in Figure 16.

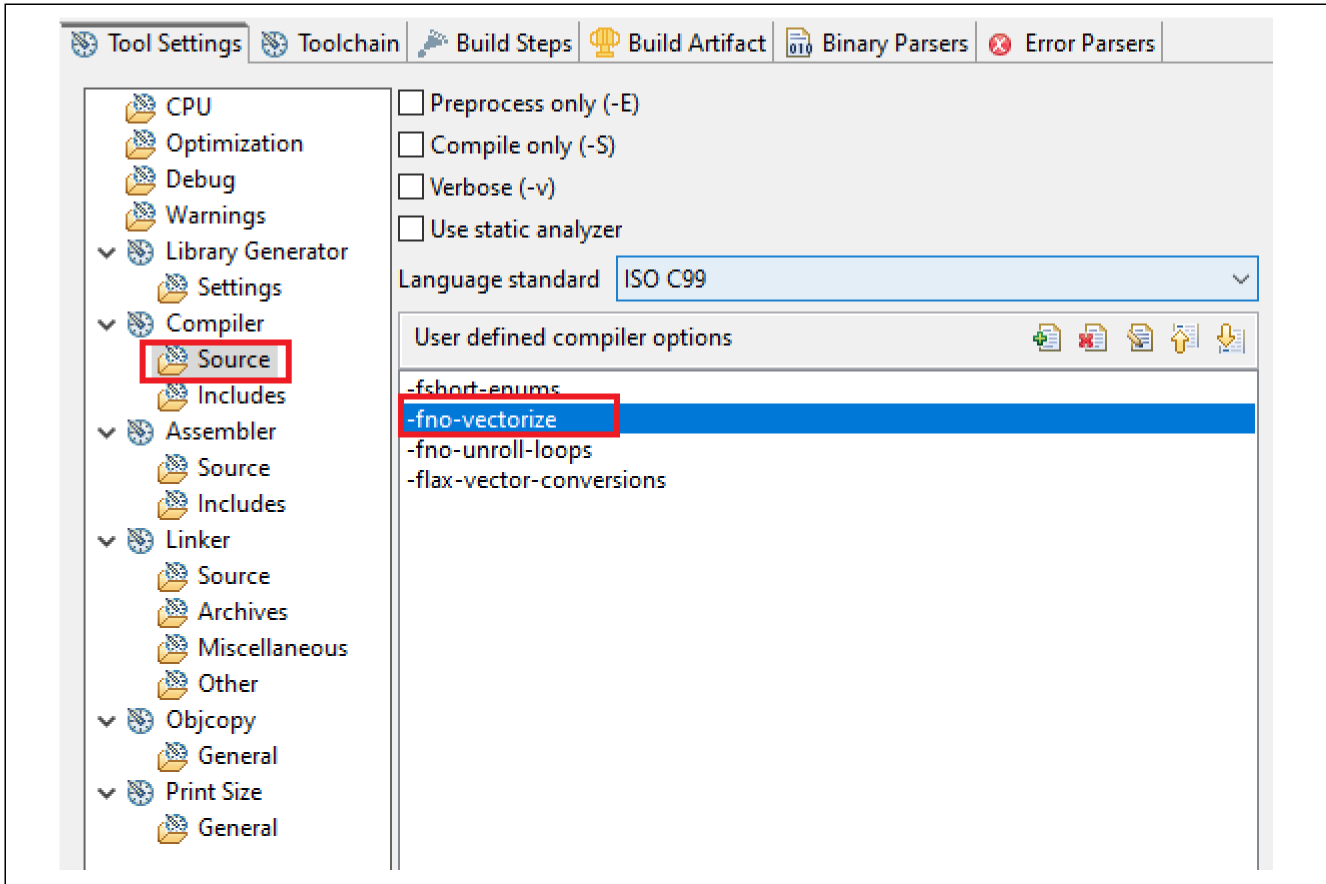

Figure 16. Example of Disabling Auto Vectorization Option in LLVM Compiler

4.2 Vector Instruction VMLADAVA Example

The VMLADAVA instruction multiplies the corresponding lanes of two input vectors, then sums these individual results to a produce a single value.

The steps of VMLADAVA.S32 Rda, Qn, Qm instruction are shown in the following diagram:

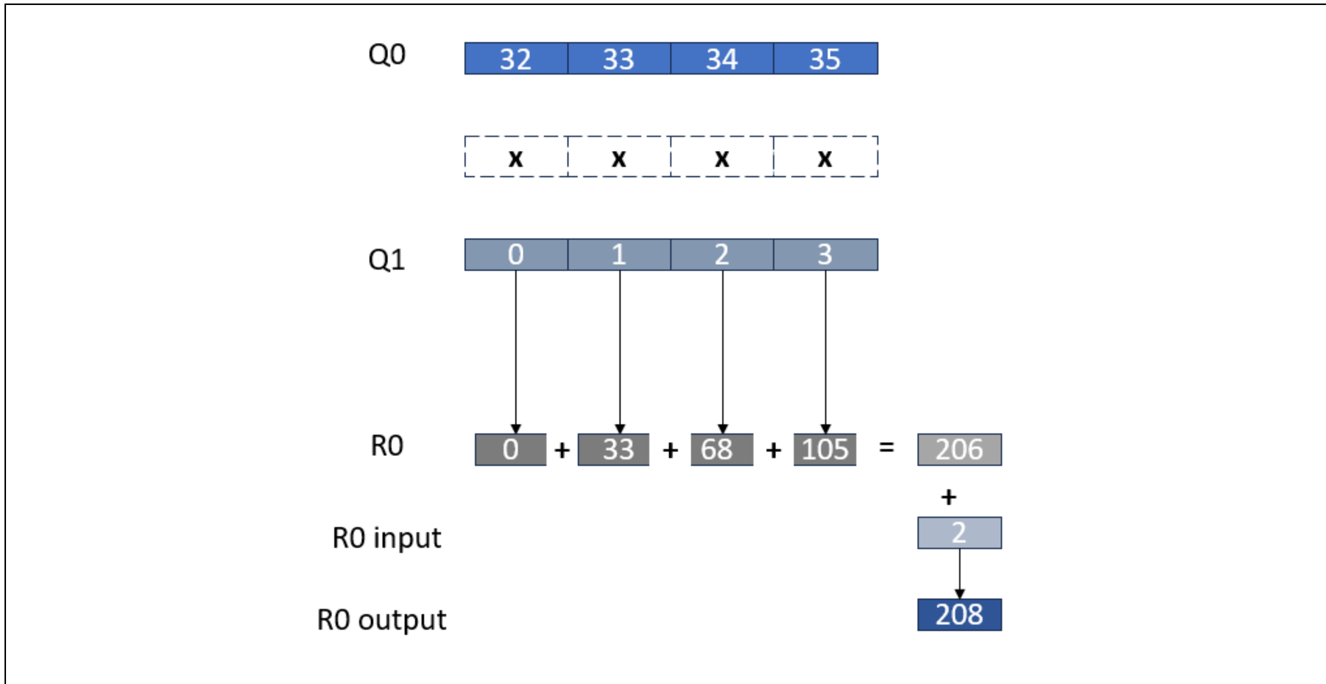

Figure 17. VMLADAVA Operation

The intrinsic function vmladavaq_s32 in Figure 18 is used to showcase the performance of VMLADAVA. S32 Rda, Qn, Qm instruction versus the scalar equivalent.

```
#if (_CONFIG_HELIUM_ == I32_HELIUM) || (_CONFIG_HELIUM_ == I32_HELIUM_DTCM)
    //Sine calculating 4 outputs at a time, the loop will be 32/4 = 8 (LOOP_
    for (i = 0; i<LOOP_NO; i++)
    {
        //Load 4 data from the array
        vector1    = vld1q_s32(p_data1);
        vector2    = vld1q_s32(p_data2);
        //Perform (vector1*vector2), sum 4 multiplication results, then add s
        result[i] = vmladavaq_s32(scalarval, vector1, vector2);
        //Increase pointers
        p_data1 += 4;
        p_data2 += 4;
    }
    //Get the timer counter
    ts_cycle_1 = R_GPT0->GTCNT;
#endif

#if (_CONFIG_HELIUM_ == I32_HELIUM_ITCM)
```

```
142              R_GPT0->GTCR  = 1; // Start timer
02000876:   str.w   r4, [r5, #-28]
0200087a:   movw    r2, #5040        ; 0x13b0
0200087e:   movw    r3, #2300        ; 0x8fc
02000882:   movt    r1, #512         ; 0x200
02000886:   movt    r2, #512         ; 0x200
0200088a:   movt    r3, #8704        ; 0x2200
167                 vector1    = vld1q_s32(p_data1);
0200088e:   vldrw.u32       q0, [r1], #16
168                 vector2    = vld1q_s32(p_data2);
02000892:   vldrw.u32       q1, [r2], #16
170                 result[i] = vmladavaq_s32(scalarval, vector1, vector2);
02000896:   movs    r6, #2
02000898:   vmlava.s32      r6, q0, q1
0200089c:   str.w   r6, [r3], #4
164              for (i = 0; i<LOOP_NO; i++)
020008a0:   le      lr, 0x200088e <hal_entry+122>
```

Figure 18. Example of VMLADAVA Instruction using Intrinsics

Figure 19 shows the scalar code equivalent to the Helium™ code in Figure 18.

```
    R_GPT0->GTCR  = 1; // Start timer

#if (_CONFIG_HELIUM_ == I32_SCALAR)
    //Perform scalar calculation (Equivalen
    for (i = 0; i<DAT_BUF_SIZE; i += 4)
    {
        for(j = 0; j<4; j++)
        {
            data1[i+j] *= data2[i+j];
        }
        for(j = 0; j<4; j++)
        {
            result[i/4] += data1[i+j];
        }
        result[i/4] += scalarval;
    }
    //Get the timer counter
    ts_cycle_1 = R_GPT0->GTCNT;
#elif (_CONFIG_HELIUM_ == I32_HELIUM) || (_

#if (_CONFIG_HELIUM_ == I32_HELIUM) || (_CO
    //Sine calculating 4 outputs at a time,
    for (i = 0; i<LOOP_NO; i++)
    {
        //Load 4 data from the array
        vector1  = vld1q_s32(p_data1);
        vector2  = vld1q_s32(p_data2);
        //Perform (vector1*vector2), sum 4 mu
        result[i] = vmladavaq_s32(scalarval,
        //Increase pointers
        p_data1 += 4;
        p_data2 += 4;
    }
    //Get the timer counter
    ts_cycle_1 = R_GPT0->GTCNT;
#endif
```

```
142           R_GPT0->GTCR  = 1; // Start timer
02000878:   str.w   r6, [r8, #-28]
0200087c:   movw    r6, #0
02000880:   movt    r6, #8704        ; 0x2200
02000884:   movt    r2, #512         ; 0x200
02000888:   mov.w   r9, #4
0200088c:   movt    r10, #8704       ; 0x2200
148               for(j = 0; j<4; j++)
02000890:   dls     lr, r9
02000894:   mov     r0, r2
02000896:   mov     r5, r6
150                   data1[i+j] *= data2[i+j];
02000898:   ldr.w   r1, [r0], #4
0200089c:   ldr     r3, [r5, #0]
0200089e:   muls    r1, r3
020008a0:   str.w   r1, [r5], #4
020008a4:   le      lr, 0x2000898 <hal_entry+132>
020008a8:   ldr.w   r0, [r10, r4]
152               for(j = 0; j<4; j++)
020008ac:   dls     lr, r9
020008b0:   lsrs    r5, r4, #2
020008b2:   mov     r1, r6
154                   result[i/4] += data1[i+j];
020008b4:   ldr.w   r3, [r1], #4
020008b8:   add     r0, r3
152               for(j = 0; j<4; j++)
020008ba:   le      lr, 0x20008b4 <hal_entry+160>
146           for (i = 0; i<DAT_BUF_SIZE; i += 4)
020008be:   cmp     r4, #28
020008c0:   add.w   r4, r4, #4
020008c4:   add.w   r6, r6, #16
020008c8:   add.w   r2, r2, #16
156               result[i/4] += scalarval;
020008cc:   add.w   r0, r0, #2
020008d0:   str.w   r0, [r10, r5, lsl #2]
146           for (i = 0; i<DAT_BUF_SIZE; i += 4)
020008d4:   bcc.n   0x2000890 <hal_entry+124>
```

Figure 19. Example of Scalar Code Equivalent of VMLADAVA Instruction and Typical Disassembly Code without Auto Vectorization

The LLVM for Arm v17.0.1 supports vectorization and generates MVE instructions equivalent to scalar code automatically as shown in Figure 20.

```
    //Get the timer counter
    ts_cycle_0 = R_GPT0->GTCNT;
    R_GPT0->GTCR  = 1; // Start timer

#if (_CONFIG_HELIUM_ == I32_SCALAR)
    //Perform scalar calculation (Equivalent with the VMLADAVA i
    for (i = 0; i<DAT_BUF_SIZE; i += 4)
    {
        for(j = 0; j<4; j++)
        {
            data1[i+j] *= data2[i+j];
        }
        for(j = 0; j<4; j++)
        {
            result[i/4] += data1[i+j];
        }
        result[i/4] += scalarval;
    }
    //Get the timer counter
    ts_cycle_1 = R_GPT0->GTCNT;
#elif (_CONFIG_HELIUM_ == I32_HELIUM) || (_CONFIG_HELIUM_ == I32

#if (_CONFIG_HELIUM_ == I32_HELIUM) || (_CONFIG_HELIUM_ == I32_H
    //Sine calculating 4 outputs at a time, the loop will be 32/
```

```
142           R_GPT0->GTCR  = 1; // Start timer
02000876:   str.w   r4, [r5, #-28]
0200087a:   movw    r2, #0
0200087e:   movw    r3, #2428        ; 0x97c
02000882:   movt    r1, #512         ; 0x200
02000886:   movt    r2, #8704        ; 0x2200
0200088a:   movt    r3, #8704        ; 0x2200
150                   data1[i+j] *= data2[i+j];
0200088e:   vldrw.u32       q0, [r1], #16
02000892:   vldrw.u32       q1, [r2, #0]
02000896:   ldr     r6, [r3, #0]
02000898:   vmul.i32        q2, q1, q0
154                   result[i/4] += data1[i+j];
0200089c:   vmlava.u32      r6, q1, q0
150                   data1[i+j] *= data2[i+j];
020008a0:   vstrb.8 q2, [r2], #16
156               result[i/4] += scalarval;
020008a4:   adds    r6, #2
020008a6:   str.w   r6, [r3], #4
146           for (i = 0; i<DAT_BUF_SIZE; i += 4)
020008aa:   le      lr, 0x200088e <hal_entry+122>
159           ts_cycle_1 = R_GPT0->GTCNT;
```

Figure 20. Example of Scalar Code Equivalent of VMLADAVA Instruction and Its Vectorization Code Generated by LLVM for Arm v17.0.1

4.3 ARM DSP Dot Product Example

The dot product example uses the arm_dot_product_f32 function in the Arm DSP library to calculate the dot product of two input vectors by multiplying element by element and sum them up. The performance of the Helium version of arm_dot_product_f32 will be compared with its scalar version.

```
void arm_dot_prod_f32(
    const float32_t * pSrcA,
    const float32_t * pSrcB,
    uint32_t    blockSize,
    float32_t * result)
{
    f32x4_t vecA, vecB;
    f32x4_t vecSum;
    uint32_t blkCnt;
    float32_t sum = 0.0f;
    vecSum = vdupq_n_f32(0.0f);

    /* Compute 4 outputs at a time */
    blkCnt = blockSize >> 2U;
    while (blkCnt > 0U)
    {
        /*
         * C = A[0]* B[0] + A[1]* B[1] + A[2]* B[2] + .....
         * Calculate dot product and then store the result i
         * and advance vector source and destination pointe
         */
        vecA = vld1q(pSrcA);
        pSrcA += 4;

        vecB = vld1q(pSrcB);
        pSrcB += 4;

        vecSum = vfmaq(vecSum, vecA, vecB);
        /*
         * Decrement the blockSize loop counter
         */
        blkCnt --;
    }


    blkCnt = blockSize & 3;
    if (blkCnt > 0U)
    {
        /* C = A[0]* B[0] + A[1]* B[1] + A[2]* B[2] + .....
```

```
020011e8:   vdup.32 q0, r2
020011ec:   vctp.32 r1
 88           *pDst++ = value;
020011f0:   vpst
020011f4:   vstrwt.32       q0, [r0, #0]
 94         }
020011f8:   pop     {r7, pc}
            arm_dot_prod_f32:
020011fa:   push    {r4, r5, r7, lr}
020011fc:   add     r7, sp, #8
 80           blkCnt = blockSize >> 2U;
020011fe:   lsrs    r5, r2, #2
02001200:   vmov.i32        q0, #0  ; 0x00000000
 81           while (blkCnt > 0U)
02001204:   wls     lr, r5, 0x2001228 <arm_dot_prod_f32+46>
02001208:   lsls    r4, r5, #4
0200120a:   vmov.i32        q0, #0  ; 0x00000000
0200120e:   add.w   r12, r0, r5, lsl #4
02001212:   mov     r5, r1
 88               vecA = vld1q(pSrcA);
02001214:   vldrw.u32       q1, [r0], #16
 91               vecB = vld1q(pSrcB);
02001218:   vldrw.u32       q2, [r5], #16
 94               vecSum = vfmaq(vecSum, vecA, vecB);
0200121c:   vfma.f32        q0, q1, q2
02001220:   le      lr, 0x2001214 <arm_dot_prod_f32+26>
02001224:   add     r1, r4
02001226:   mov     r0, r12
102           blkCnt = blockSize & 3;
02001228:   ands.w  r2, r2, #3
103           if (blkCnt > 0U)
0200122c:   beq.n   0x2001242 <arm_dot_prod_f32+72>
108               vecA = vld1q(pSrcA);
0200122e:   vldrw.u32       q1, [r0, #0]
109               vecB = vld1q(pSrcB);
02001232:   vldrw.u32       q2, [r1, #0]
107               mve_pred16_t p0 = vctp32q(blkCnt);
02001236:   vctp.32 r2
```

Figure 21. arm_dot_product_f32 Function with Helium™ Code

Renesas Flexible Software Package FSP supports Arm DSP Library Source for Cortex®-M85 that uses Helium intrinsics. It will improve performance significantly compared to scalar code. Select Arm DSP Library Source in Project Configurator to add the DSP source to your project, as shown in Figure 22.

Figure 22. Adding Arm Library DSP Source in FSP Configurator

Click Generate Project Content, the Arm DSP library source will be added to the project.

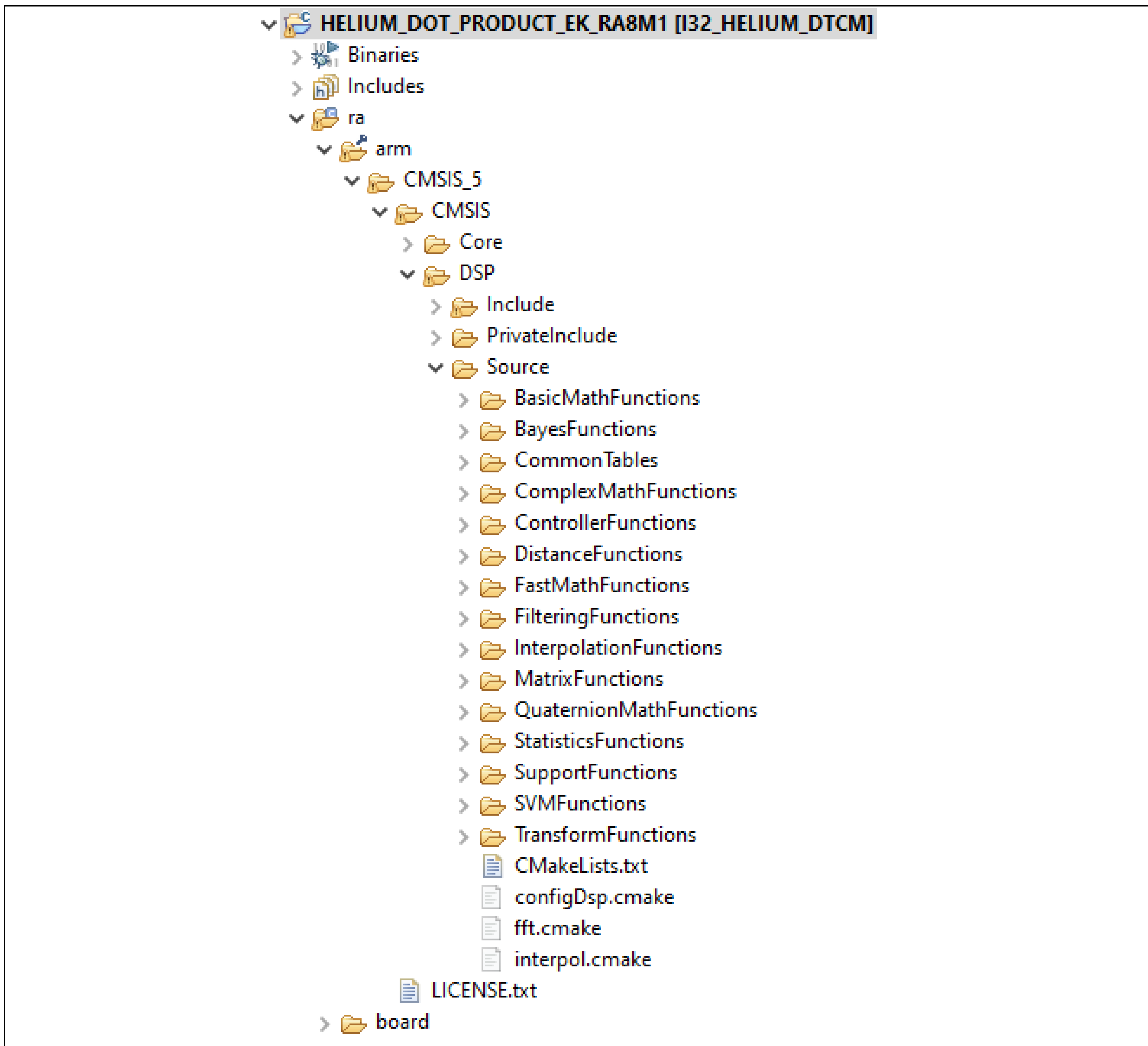

Figure 23. Arm Library DSP Source Added in FSP Project

4.4 Performance Improvement

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

4.4.1 Tightly Coupled Memory (TCM)

The 128 KB TCM memory in RA8 MCU 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.

Figure 24 shows ITCM and DTCM in the Local CPU Subsystem.

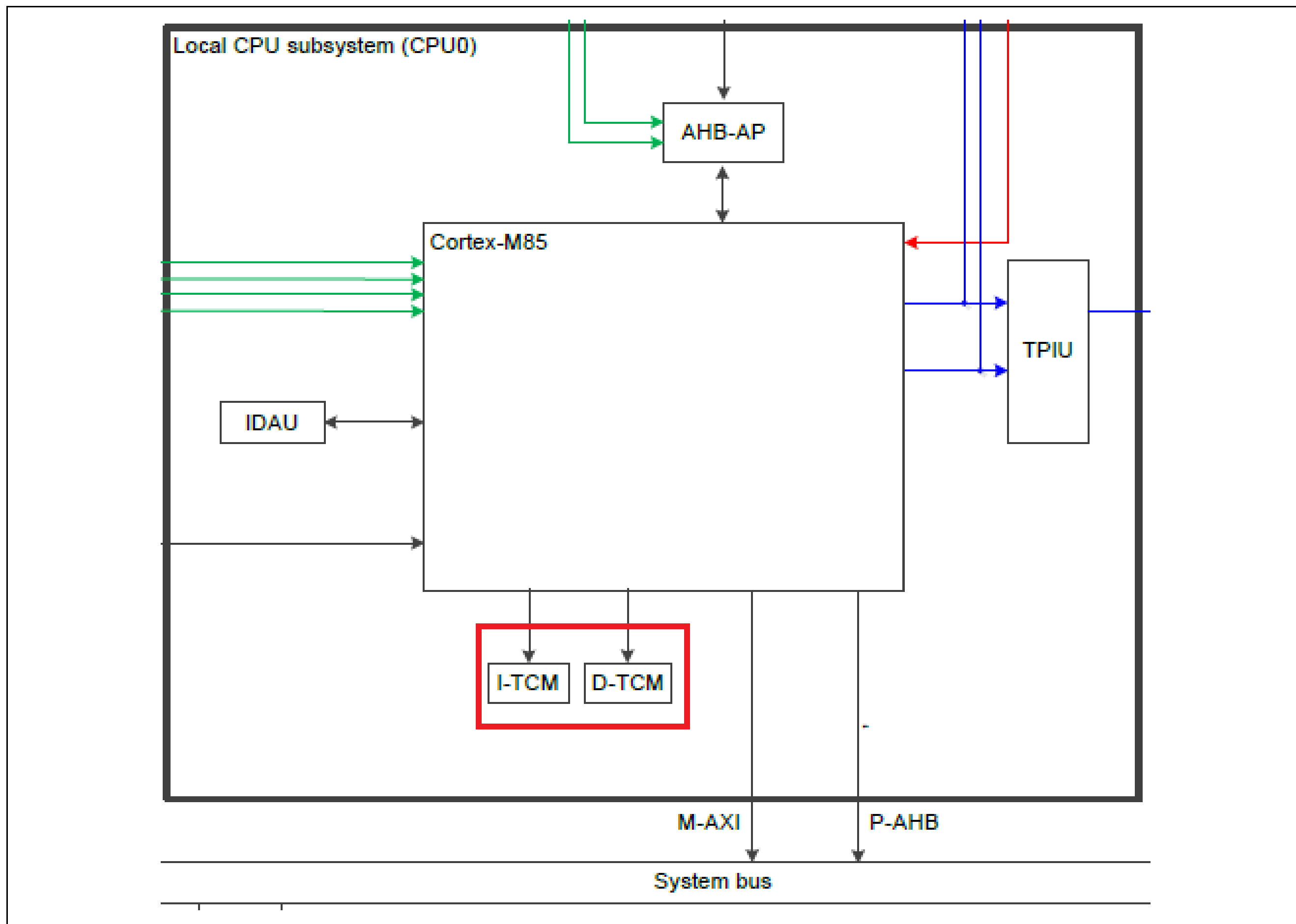

Figure 24. ITCM and DTCM in Local CPU Subsystem

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.

Figure 25 and Figure 26 are snapshots of ITCM and DCTM locations in RA8 MCU.

Address	Area	Security
	Reserved area*2	Non-secure
0x1300_A300	On-chip flash (option-setting memory)	
0x1300_A100	Reserved area*2	
0x1300_81B4	On-chip flash (Factory Flash)	
0x1300_80F0	Reserved area*2	
0x122F_8000	On-chip flash (code flash) (read only)*1	
0x1200_0000	Reserved area*2	
0x1001_0000	ITCM	
0x1000_0000	Reserved area*2	Non-secure callable for CPU Secure for other bus masters
0x0300_A300	On-chip flash (option-setting memory)	
0x0300_A100	Reserved area*2	
0x0300_81B4	On-chip flash (Factory Flash)	
0x0300_80F0	Reserved area*2	
0x022F_8000	On-chip flash (code flash) (read only)*1	
0x0200_0000	Reserved area*2	
0x0001_0000	ITCM	
0x0000_0000		

Figure 25. Example of ITCM Areas in RA8 MCU

Address	Area	Security
0x3001_0000	Reserved area*2	Non-secure callable for CPU Secure for other bus masters
0x3000_0000	DTCM	
0x2703_0400	Reserved area*2	
0x2703_0050	On-chip flash (option-setting memory)	
0x2700_3000	Reserved area*2	
0x2700_0000	On-chip flash (data flash)	
0x2600_0400	Reserved area*2	
0x2600_0000	Standby SRAM	
0x220E_0000	Reserved area*2	
0x2200_0000	On-chip SRAM	
0x2001_0000	Reserved area*2	
0x2000_0000	DTCM	
	Reserved area*2	Non-secure
0x1300_A300	On-chip flash (option-setting memory)	
0x1300_A100	Reserved area*2	
0x1300_81B4	On-chip flash (Factory Flash)	
0x1300_80F0	Reserved area*2	
0x122F_8000	On-chip flash (code flash) (read only)*1	
0x1200_0000	Reserved area*2	
0x1001_0000	ITCM	
0x1000_0000	Reserved area*2	Non-secure callable for CPU Secure for other bus masters
0x0300_A300	On-chip flash (option-setting memory)	
0x0300_A100	Reserved area*2	
0x0300_81B4	On-chip flash (Factory Flash)	
0x0300_80F0	Reserved area*2	
0x022F_8000	On-chip flash (code flash) (read only)*1	
0x0200_0000	Reserved area*2	
0x0001_0000	ITCM	
0x0000_0000		

Figure 26. Example of DTCM Areas in RA8 MCU

4.4.2 Improve Performance Using DTCM

You can place data in the DTCM section (.dtcm_data) in an FSP-based project using the _attribute_ directive, as shown in Figure 27.

```
#elif (_CONFIG_HELIUM_ == I32_HELIUM_DTCM)
static int32x4_t vector1;
static int32x4_t vector2;
static int32x4_t __attribute__((section(".dtcm_data"))) result[8];
static int32_t __attribute__((section(".dtcm_data"))) *p_data1;
static int32_t __attribute__((section(".dtcm_data"))) *p_data2;
//Input Data
static int32_t data1[] __attribute__((section(".dtcm_data"))) __attribute__((aligned(8))) = {0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7,
                                                                                            0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF,
                                                                                            0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
                                                                                            0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F};

static int32_t data2[] __attribute__((section(".dtcm_data"))) __attribute__((aligned(8)))  = {0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27,
                                                                                             0x28, 0x29, 0x2A, 0x2B, 0x2C, 0x2D, 0x2E, 0x2F,
                                                                                             0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37,
                                                                                             0x38, 0x39, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F};
#endif
```

Figure 27. Placing Variables and Data in DTCM Section

The above data placement can be confirmed using the memory map generated by the compiler.

```
20000000  2001690      190    16 .dtcm_data
20000000  2001690        0     1         __tz_DTCM_S = ABSOLUTE ( __DTCM_START )
20000000  2001690        0     1         __dtcm_data_start = .
20000000  2001690      190     8         ./src/hal_entry.o:(.dtcm_data)
20000000  2001690       80     1                 data1
20000080  2001710        4     1                 p_data1
20000088  2001718       80     1                 data2
20000108  2001798        4     1                 p_data2
20000110  20017a0       80     1                 result
20000190  2001820        0     1         . = ALIGN ( 8 )
20000190  2001820        0     1         __dtcm_data_end = .
20000190  2001820        0     1 . = __dtcm_data_end
```

Figure 28. Example of Variables and Data Placed in DTCM Area in Memory Map

4.4.3 Improve Performance Using ITCM

One of the methods to place some portions of the code in the ITCM section (.itcm_data) is using the _attribute_ directive, as shown in Figure 29.

```
#if (_CONFIG_HELIUM_ == I32_HELIUM_ITCM)
//Placing functions in section .itcm_data
void __attribute__((section(".itcm_data"))) itcm_func(void);
void itcm_func(void)
{
    unsigned int i;
    //Pointer values for both arrays
    int32_t *p_data1  = &data1[0];
    int32_t *p_data2  = &data2[0];
    R_GPT0->GTCR  = 0; // Stop timer
    R_GPT0->GTCNT = 0; // Clear counter
    //Get the timer counter
    ts_cycle_0 = R_GPT0->GTCNT;
    R_GPT0->GTCR  = 1; // Start timer
    //Sine calculating 4 outputs at a time, the loop will be 32/4 = 8 (LOOP_NO )
    for (i = 0; i<LOOP_NO; i++)
    {
      //Load 4 data from the array
      vector1   = vld1q_s32(p_data1);
      vector2   = vld1q_s32(p_data2);
      //Multiply (vector2*scalarval), add vector1 and store the results
      result[i] = vmlaq_n_s32(vector1, vector2, scalarval);
      //Increase pointers
      p_data1 += 4;
      p_data2 += 4;
     }
     //Get the timer counter
     ts_cycle_1 = R_GPT0->GTCNT;
}
//End placing functions in section .itcm_data
#endif
```

Figure 29. Example of Placing a Function in ITCM Section

You can confirm code placement using the .map file generated by the compiler or when debugging the project.

```
66            #if (_CONFIG_HELIUM_ == I32_HELIUM_ITCM)
67            //Placing functions in section .itcm_data
68            void __attribute__((section(".itcm_data"))) itcm_func(void);
69            void itcm_func(void)
70 00000002   {
71                unsigned int i;
72                //Pointer values for both arrays
73                int32_t *p_data1  = &data1[0];
74                int32_t *p_data2  = &data2[0];
75 0000001c       R_GPT0->GTCR  = 0; // Stop timer
76 00000024       R_GPT0->GTCNT = 0; // Clear counter
77                //Get the timer counter
78 0000000a       ts_cycle_0 = R_GPT0->GTCNT;
79 00000032       R_GPT0->GTCR  = 1; // Start timer
80                //Sine calculating 4 outputs at a time, the loop will be 32/4 = 8 (LOOP_NO )
81 0000005a       for (i = 0; i<LOOP_NO; i++)
82                {
83                  //Load 4 data from the array
84 0000004a         vector1   = vld1q_s32(p_data1);
85 00000046         vector2   = vld1q_s32(p_data2);
86                  //Multiply (vector2*scalarval), add vector1 and store the results
87 0000004e         result[i] = vmlaq_n_s32(vector1, vector2, scalarval);
88                  //Increase pointers
89                  p_data1 += 4;
90                  p_data2 += 4;
91                 }
92                 //Get the timer counter
93 0000005e        ts_cycle_1 = R_GPT0->GTCNT;
94 00000064   }
95            //End placing functions in section .itcm_data
96            #endif
```

Figure 30. Function Placed in ITCM Section Shown on Debugger

4.5 Improve Performance by Utilizing Data Cache

When a function utilizes long loops, it executes the same code repeatedly. Furthermore, in many applications, data access may be repeated and sequential. Performance in these scenarios can improve significantly with cache enabled. Notes that data cache is not enabled in the projects by default.

In FSP, the instruction cache enable is done in a function named SystemInit in system.c, as shown in Figure 31 and Figure 32.

```
/***********************************************************************************************************
 * Macro definitions
 **********************************************************************************************************/

/* Mask to select CP bits( 0xF00000 ) */
#define CP_MASK                                     (0xFU << 20)

/* Startup value for CCR to enable instruction cache, branch prediction and LOB extension */
#define CCR_CACHE_ENABLE                            (0x000E0201)

/* Value to write to OAD register of MPU stack monitor to enable NMI when a stack overflow is detected. */
#define BSP_STACK_POINTER_MONITOR_NMI_ON_DETECTION  (0xA500U)
```

Figure 31. Macro Definition to Enable Cache in system.c in FSP

```
/*******************************************************************************************************************//**
 * Initialize the MCU and the runtime environment.
 **********************************************************************************************************************/
void SystemInit (void)
{
#if defined(RENESAS_CORTEX_M85)

    /* Enable the ARM core instruction cache, branch prediction and low-overhead-branch extension.
     * See Section 5.5 of the Cortex-M55 TRM and Section D1.2.9 in the ARMv8-M Architecture Reference Manual */
    SCB->CCR = (uint32_t) CCR_CACHE_ENABLE;
    __DSB();
    __ISB();
#endif

    /* Enable the ARM core instruction cache, branch prediction and low-overhead-branch extension.
     * See Section 5.5 of the Cortex-M55 TRM and Section D1.2.9 in the ARMv8-M Architecture Reference Manual */
    SCB->CCR = (uint32_t) CCR_CACHE_ENABLE;
```

Figure 32. Code to Enable Instruction Cache in FSP

The application projects have a setting to enable data cache. Add the macro definition "_DCACHE_ENABLE_=1" in the project option to enable data cache. Even though data cache improves performance, it can cause concurrency and coherency issues. It is good practice to enable the cache for application code that has repeated access to the same set of data.

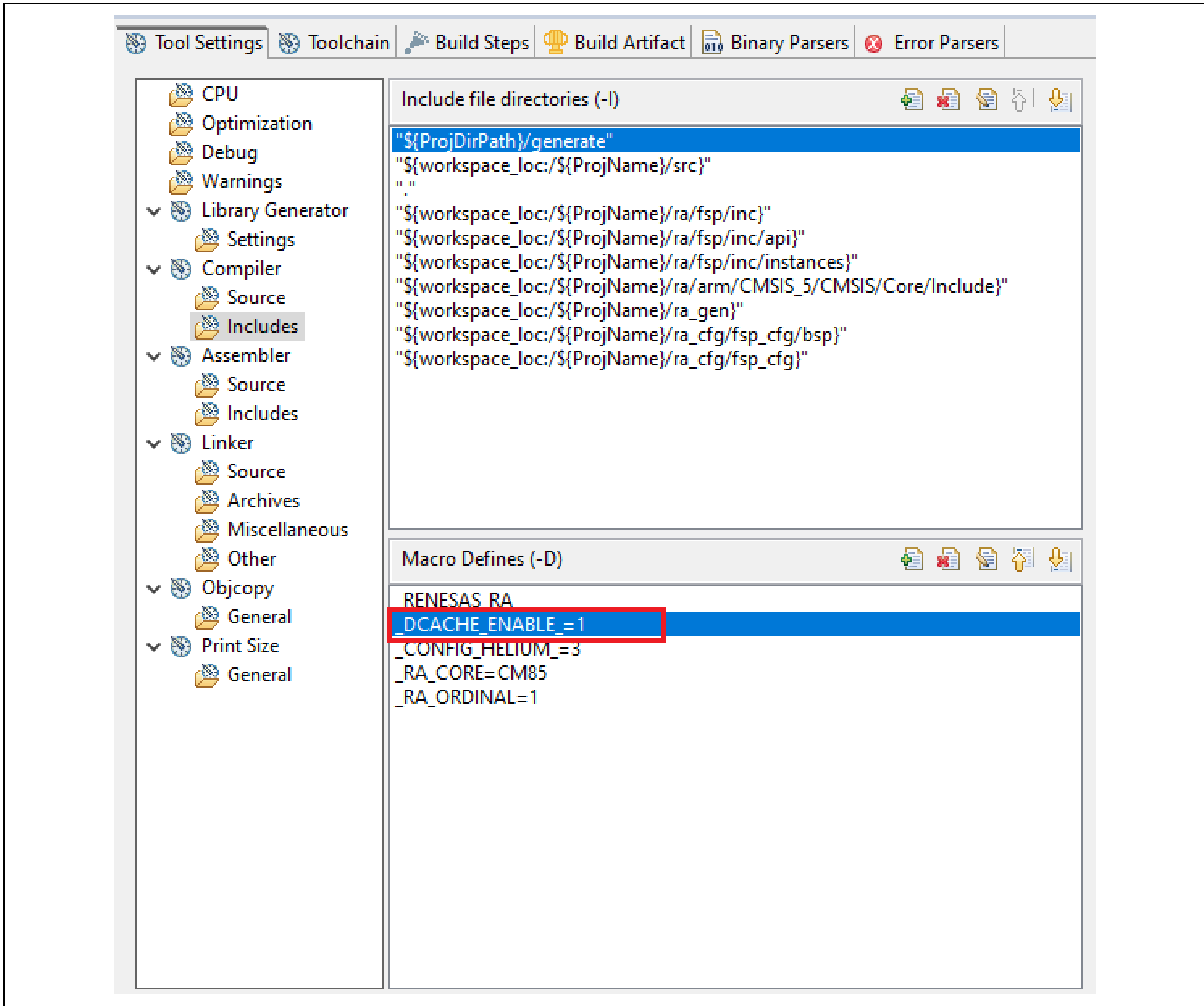

Figure 33. Example of Data Cache Enable in Application Project

Example code to enable and disable data cache are shown in Figure 34 and Figure 35.

```
#if (_DCACHE_ENABLE_ == DCACHE_ENABLE_YES)
    SCB_EnableDCache(); //Enable DCache
#endif
```

Figure 34. Example Code to Enable DCACHE

```
#if (_DCACHE_ENABLE_ == DCACHE_ENABLE_YES)
    SCB_DisableDCache(); // Disable Dcache
#endif
```

Figure 35. Example Code to Disable DCACHE

Another method to enable data cache is using FSP Configurator: BSP->Properties->Settings->MCU (RA8M1) Family->Cache settings->Data cache, as shown in Figure 36.

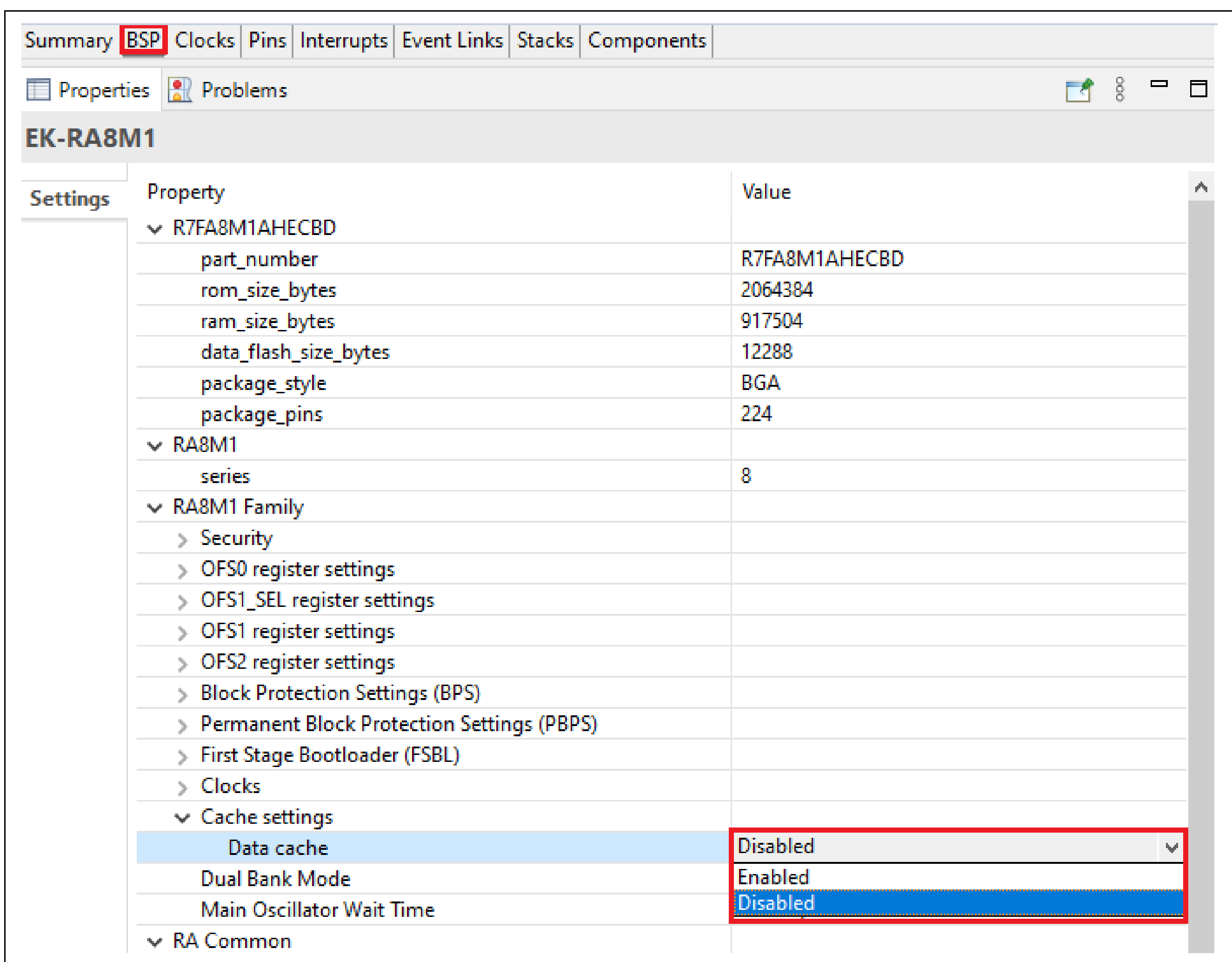

Figure 36. Example of Data Cache Enable using FSP Configurator

4.6 Using General Purpose (GPT) Timer for Benchmarking

In the projects, GPT0 timer is used to measure time for performance benchmarking.

```
    //Clear and start timer for benchmarking
    R_BSP_MODULE_START(FSP_IP_GPT, 0);
    R_GPT0->GTCNT = 0; // Clear counter
    //Get the timer counter
    ts_cycle_0 = R_GPT0->GTCNT;
    R_GPT0->GTCR  = 1; // Start timer

#if (_CONFIG_HELIUM_ == I32_SCALAR)
    //Perform scalar calculation (Equivalent with the multiply accumulate instruction)
    for (i = 0; i<DAT_BUF_SIZE; i++)
     {
        data1[i] += (data2[i] * scalarval);
     }
    //Get the timer counter
    ts_cycle_1 = R_GPT0->GTCNT;
#elif (_CONFIG_HELIUM_ == I32_HELIUM) || (_CONFIG_HELIUM_ == I32_HELIUM_DTCM) || (_CONFIG_HELIUM_ == I32_HELIUM_ITCM)

#if (_CONFIG_HELIUM_ == I32_HELIUM) || (_CONFIG_HELIUM_ == I32_HELIUM_DTCM)
    //Sine calculating 4 outputs at a time, the loop will be 32/4 = 8 (LOOP_NO )
    for (i = 0; i<LOOP_NO; i++)
    {
      //Load 4 data from the array
      vector1   = vld1q_s32(p_data1);
      vector2   = vld1q_s32(p_data2);
      //Multiply (vector2*scalarval), add vector1 and store the results
      result[i] = vmlaq_n_s32(vector1, vector2, scalarval);
      //Increase pointers
      p_data1 += 4;
      p_data2 += 4;
     }
    //Get the timer counter
    ts_cycle_1 = R_GPT0->GTCNT;
```

Figure 37. Example of the Timer Code for Benchmarking

5. Verify the Project

5.1 Import The Projects

The software tools required to run the application projects are as follows.

- e² studio version: 2024-01.1 (24.1.1) or later
- LLVM Embedded Toolchain for Arm v17.0.1 or later
- Renesas Flexible Software Package (FSP) v5.2.0 or later
- SEGGER RTT Viewer v7.94g or later

Import the projects HELIUM_VMLA_EK_RA8M1.zip, HELIUM_VMLADAVA_EK_RA8M1.zip, and HELIUM_DOT_PRODUCT_EK_RA8M1.zip into your workspace.

5.2 Build Project

There are several configurations in each project. Select a project, then a project configuration you wish to run before going to the next step.

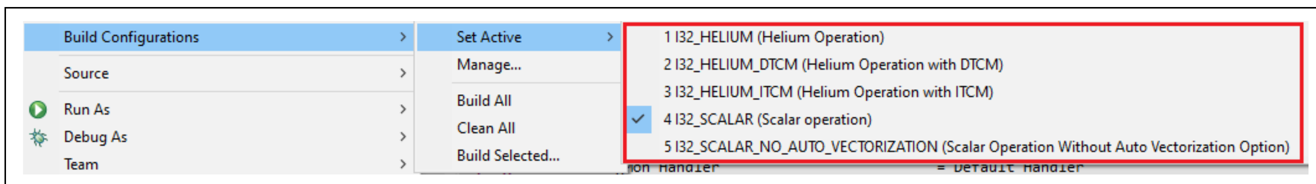

Figure 38. Set Active Build Configuration

Launch configuration.xml, and click “Generate Project Content” to generate project content.

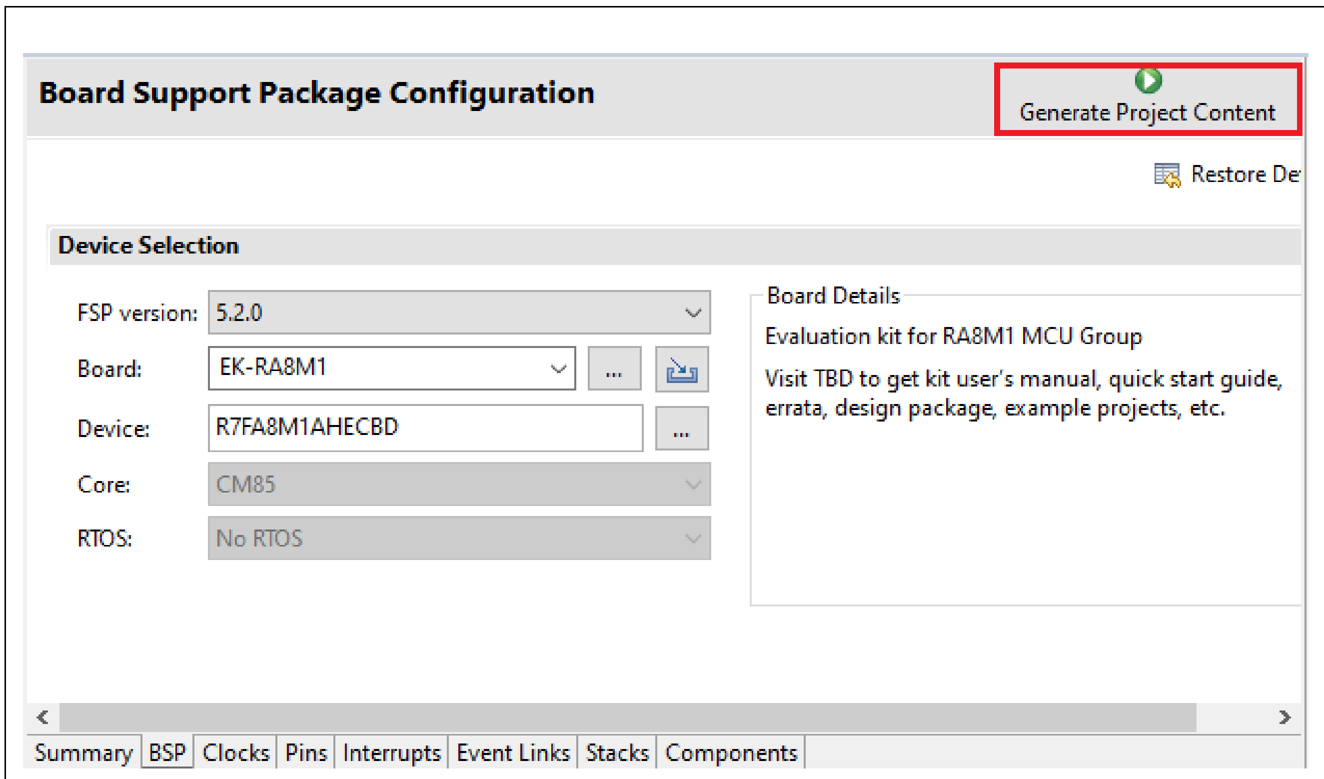

Figure 39. Example of Generating Project Content

Build the active project by selecting Project->Build Project.

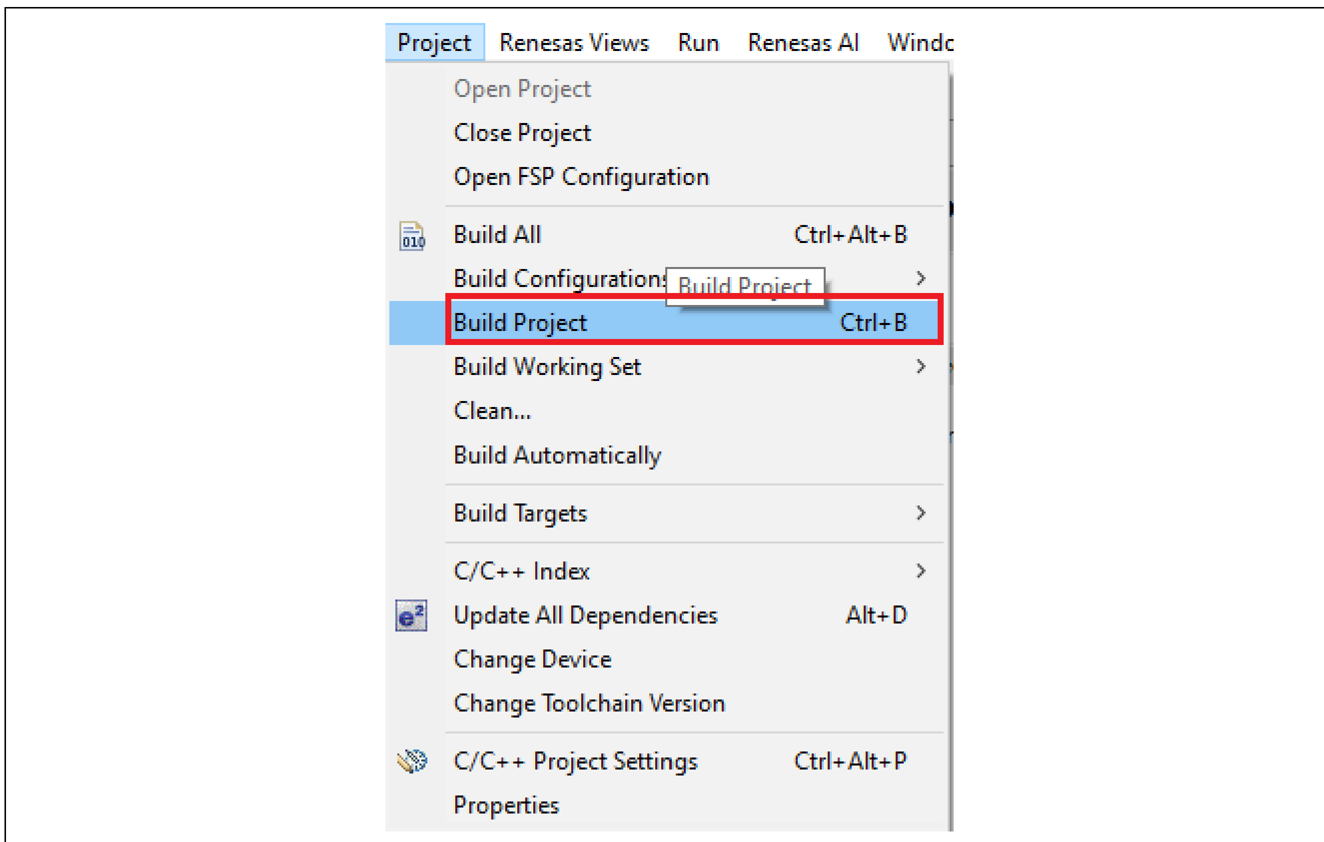

Figure 40. Build the Active Project

5.3 Download and Run Project

The EK-RA8M1 kit has a few switch settings that must be configured before running the projects associated with this application note. These switches must be returned to the default settings per the EK-RA8M1 user manual. In addition to these switch settings, the board also contains a USB debug port and connectors to access the J-Link® programming interface.

Table 1. Switch settings for EK-RA8M1

Switch	Setting
J8	Jumper on pins 1-2
J9	Open

Connect J10 on EK-RA8M1 kit to USB port on your PC, open and start SEGGER RTT Viewer with the below settings.

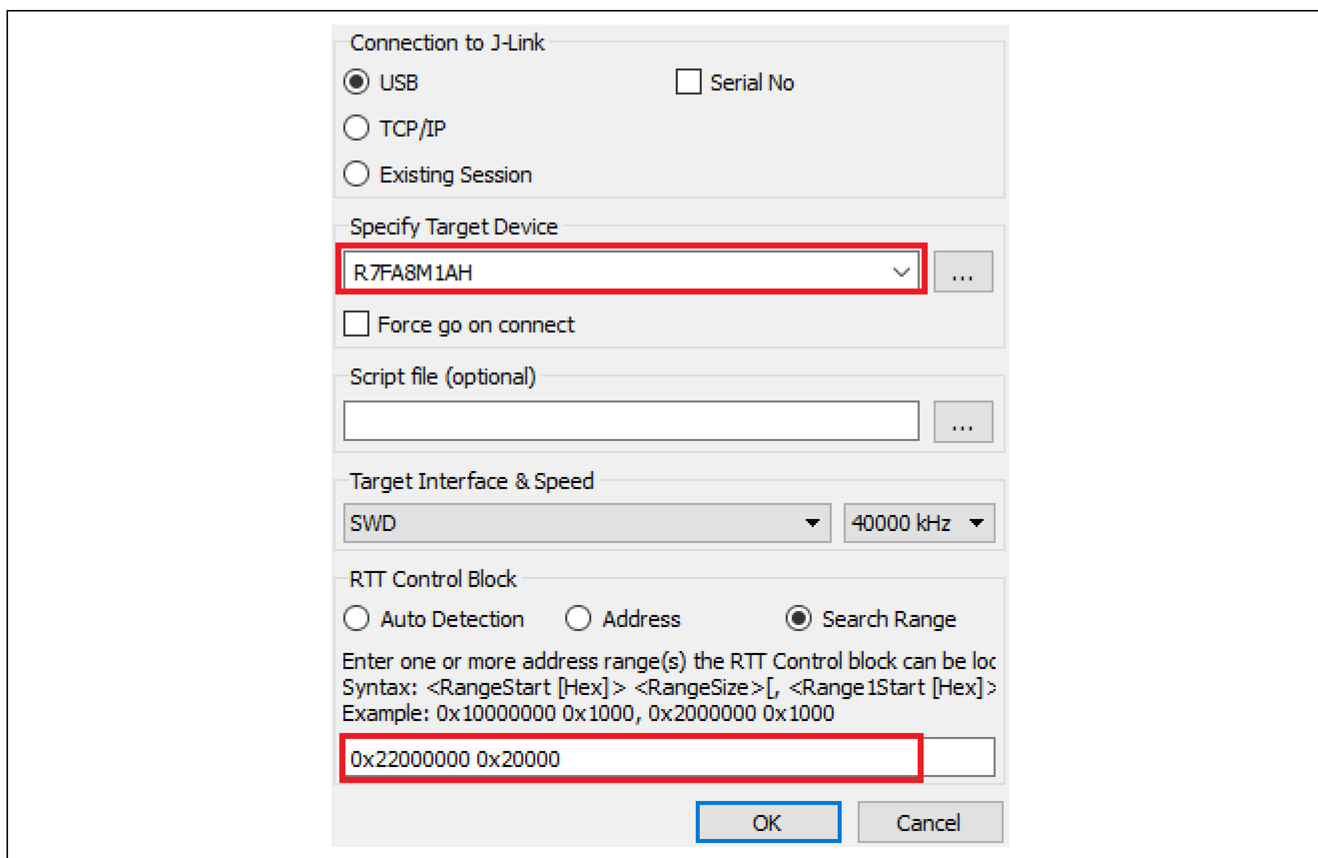

Figure 41. SEGGER RTT Viewer

Choose the Debug Configuration you wish to run. Make sure it points to the correct .elf file.

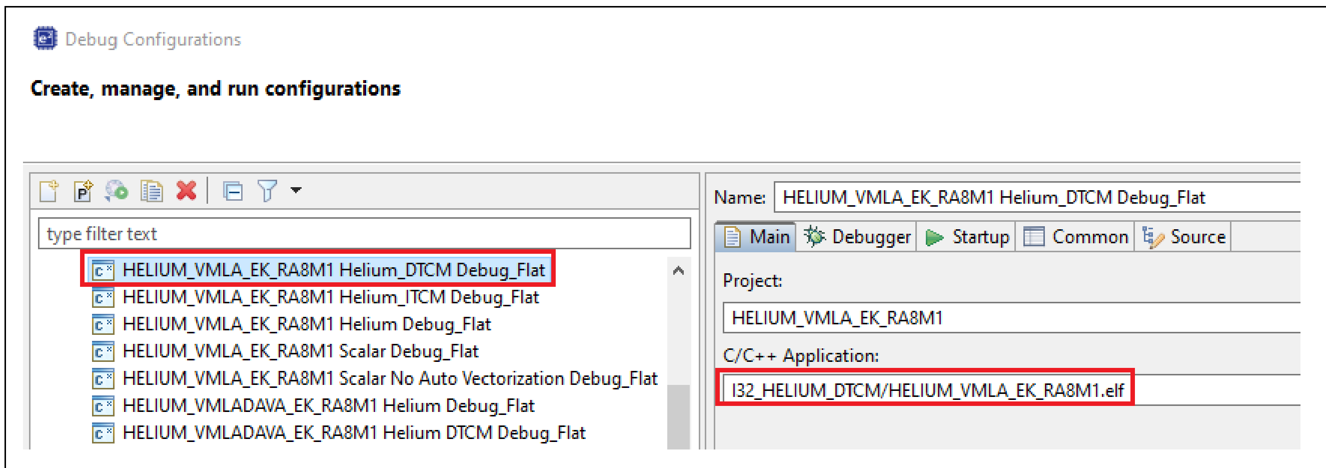

Figure 42. Start Running the Project

The operation results will be printed on SEGGER RTT Viewer, as shown in Figure 43.

File Terminals Input Logging Help

All Terminals | Terminal 0 | Terminal 1 | Terminal 2 | Terminal 6

```
VMLA-Helium Operation on RA8M1 with DTCM Utilized
Timer counter cycle: 16
result[0]: 64
result[1]: 67
result[2]: 70
result[3]: 73
result[4]: 76
result[5]: 79
result[6]: 82
result[7]: 85
result[8]: 88
result[9]: 91
result[10]: 94
result[11]: 97
result[12]: 100
result[13]: 103
result[14]: 106
result[15]: 109
result[16]: 112
result[17]: 115
result[18]: 118
result[19]: 121
result[20]: 124
result[21]: 127
result[22]: 130
result[23]: 133
result[24]: 136
result[25]: 139
result[26]: 142
result[27]: 145
result[28]: 148
result[29]: 151
result[30]: 154
result[31]: 157
```

Figure 43. A Helium Operation with DTCM Utilized

5.4 Benchmarking Performance

Use the "Timer counter cycle" printed on SEGGER RTT Viewer for performance benchmarking. It shows how many GPT0 counter cycles have elapsed since the function was executed.

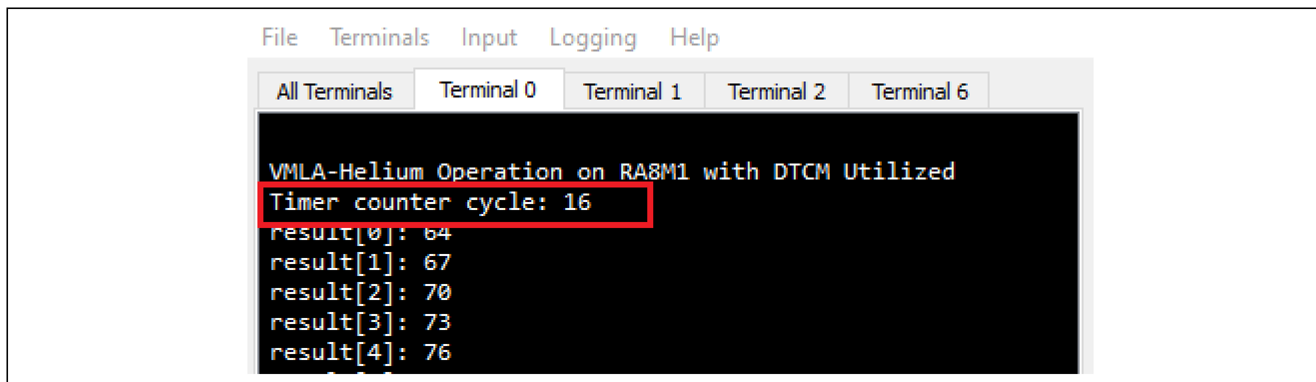

Figure 44. Example of Timer Counter Cycle on RTT Viewer

5.4.1 VMLA Project HELIUM_VMLA_EK_RA8M1

The performances of the function vmlaq_n_s32 in various configurations are as follows.

Project Configuration	Timer cycle	Performance Increase (vs I32_SCALAR_NO_AUTO_VECTORIZATION) %
I32_SCALAR_NO_AUTO_VECTORIZATION	241	
I32_SCALAR	142	69.72
I32_HELIUM	138	74.64
I32_HELIUM_DTCM	16	1406.25
I32_HELIUM_ITCM	136	77.21

Figure 45. Performance Data w/o Data Cache Enable

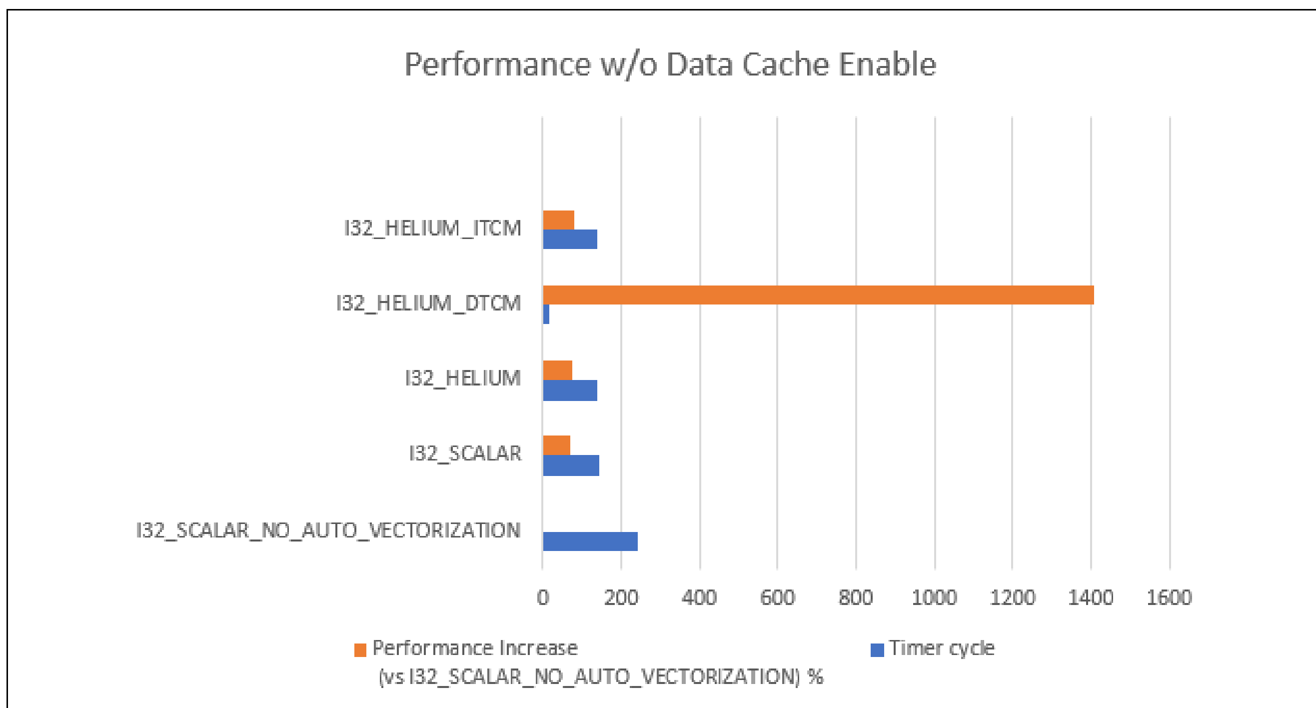

Figure 46. Performance Chart w/o Data Cache Enable

Below are the performances of the vmlaq_n_s32 function with data cache enabled in various configurations. To enable data cache in the project, follows steps in section 4.5, build and download it.

Project Configuration	Timer cycle	Performance Increase (vs I32_SCALAR_NO_AUTO_VECTORIZATION) %
I32_SCALAR_NO_AUTO_VECTORIZATION (w/o data cache enable)	241	
I32_SCALAR_NO_AUTO_VECTORIZATION (w/ data cache enable)	58	315.52
I32_SCALAR	49	391.84
I32_HELIUM	51	372.55
I32_HELIUM_DTCM	14	1621.43
I32_HELIUM_ITCM	49	391.84

Figure 47. Performance Data w/ Data Cache Enable

5.4.2 VMLAVADA Project HELIUM_VMLADAVA_EK_RA8M1

The performances of the function vmladavaq_s32 in various configurations are as follows.

Project Configuration	Timer cycle	Performance Increase (vs I32_SCALAR_NO_AUTO_VECTORIZATION) %
I32_SCALAR_NO_AUTO_VECTORIZATION	426	
I32_SCALAR	174	144.83
I32_HELIUM	139	206.47
I32_HELIUM_DTCM	14	2942.86
I32_HELIUM_ITCM	135	215.56

Figure 48. Performance Data w/o Data Cache Enable

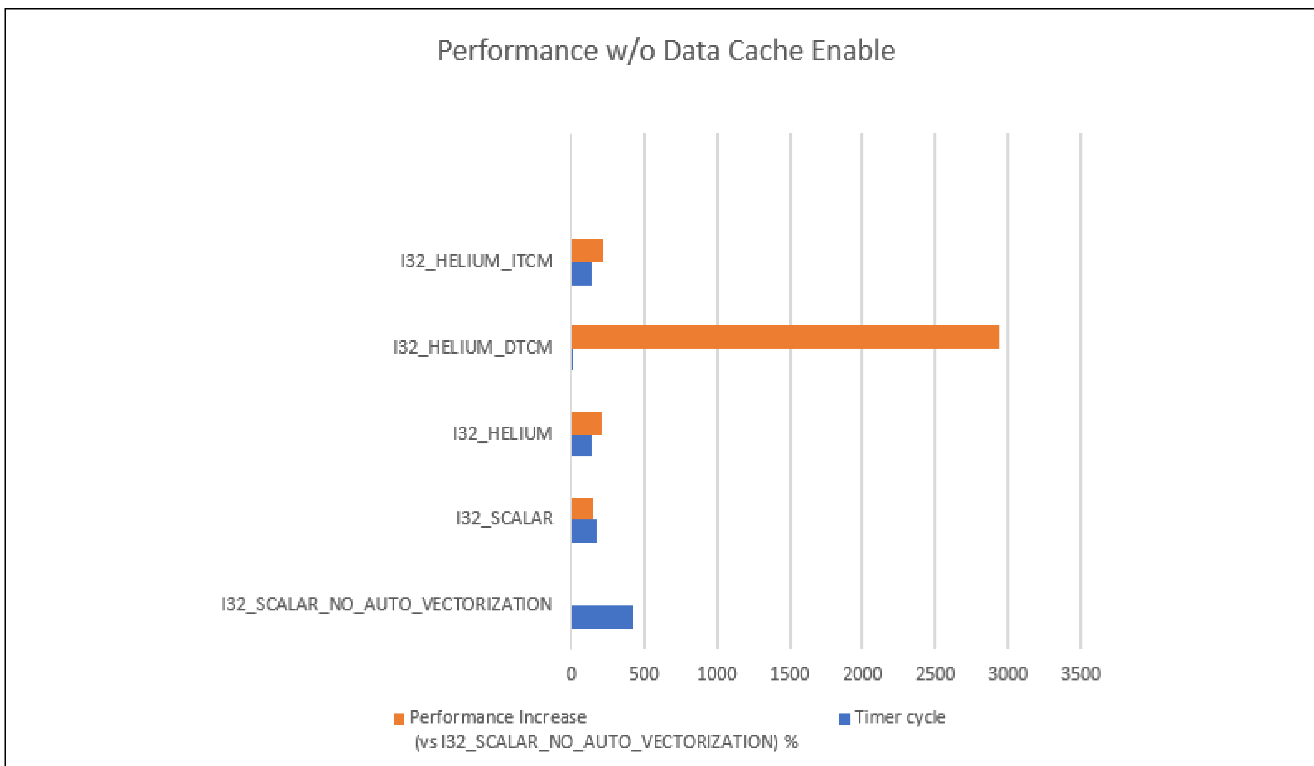

Figure 49. Performance Chart w/o Data Cache Enable

Below are the performances of the vmladavaq_s32 function with data cache enabled in various configurations. To enable data cache in the project, follows steps in section 4.5, build and download it.

Project Configuration	Timer cycle	Performance Increase (vs I32_SCALAR_NO_AUTO_VECTORIZATION) %
I32_SCALAR_NO_AUTO_VECTORIZATION (w/o data cache enable)	426	
I32_SCALAR_NO_AUTO_VECTORIZATION (w/ data cache enable)	103	313.59
I32_SCALAR	61	598.36
I32_HELIUM	43	890.70
I32_HELIUM_DTCM	14	2942.86
I32_HELIUM_ITCM	46	826.09

Figure 50. Performance Data w/ Data Cache Enable

5.4.3 DSP Dot Product Project HELIUM_DOT_PRODUCT_EK_RA8M1

The performances of the ARM DSP Dot Product arm_dot_prod_f32 function in various configurations are as follows.

Project Configuration	Timer cycle	Performance Increase (vs I32_SCALAR) %
I32_SCALAR	265	
I32_HELIUM	153	73.20
I32_HELIUM_DTCM	95	178.95

Figure 51. Performance Data w/o Data Cache Enable

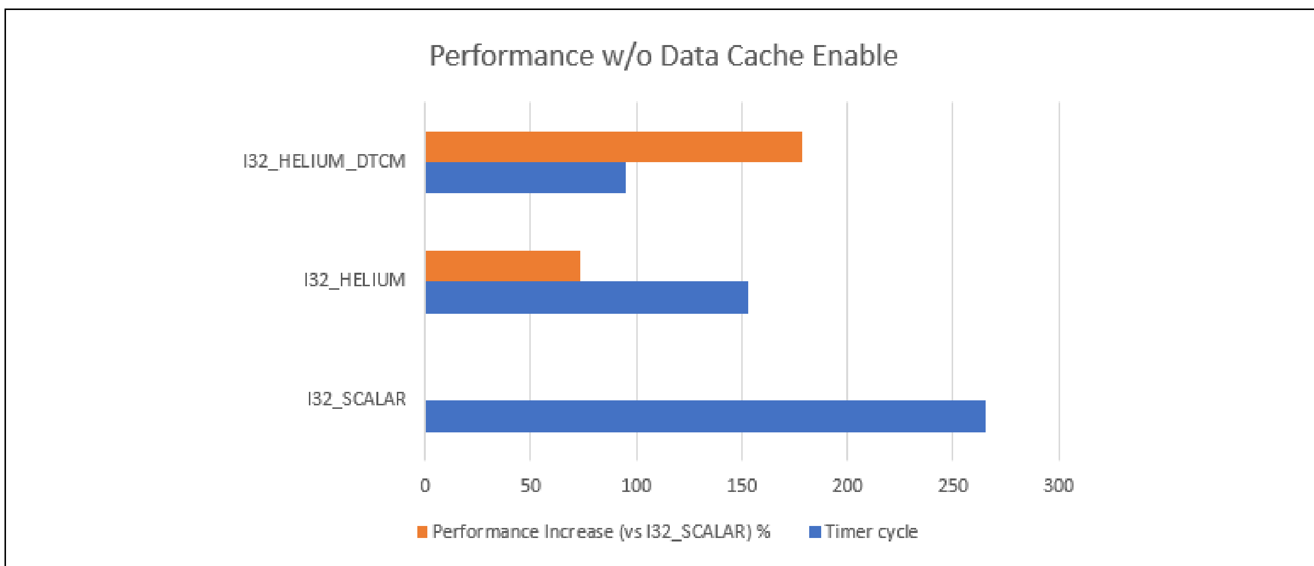

Figure 52. Performance Chart w/o Data Cache Enable

Below are the performances of the ARM Dot Product arm_dot_prod_f32 function with data cache enabled in various configurations. To enable data cache in the project, follows steps in section 4.5, build and download it.

Project Configuration	Timer cycle	Performance Increase (vs I32_SCALAR) %
I32_SCALAR (w/o data cache enable)	265	
I32_SCALAR	49	440.82
I32_HELIUM	26	919.23
I32_HELIUM_DTCM	17	1458.82

Figure 53. Performance Data w/ Data Cache Enable

6. Conclusion

The Renesas RA8 MCU with Arm® Cortex®-M85 supports significant scalar performance uplift. Furthermore, the Tightly Coupled Memory (TCM) support in Renesas FSP makes it easier to utilize Helium intrinsics and TCM for further improvement.

Website and Support

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

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

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Aug.11.23	-	Initial version
2.00	Apr.05.24	-	Support LLVM for Arm

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

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

1. Precaution against Electrostatic Discharge (ESD)

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

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

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

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

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

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

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

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

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.
   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

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

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

(Rev.5.0-1 October 2020)

Corporate Headquarters

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

Trademarks

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

Contact information

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

© 2024 Renesas Electronics Corporation. All rights reserved.