Renesas RA Family

Getting Started with CoreMark Benchmarking

Introduction

As processors in embedded systems become more complex, more sophisticated benchmarks are needed to better understand performance and analysis.

CoreMark is a modern and sophisticated benchmark that is recommended by ARM® and allows you to accurately measure the performance of a processor. Rather than using arbitrary and synthetic code, CoreMark uses basic data structures and algorithms that are common in any embedded application.

Using CoreMark is encouraged due to its ANSI C compliance, and the fact that it is designed to ensure that compilers cannot pre-compute numbers to influence the results and that it does not make any library calls during the benchmarked portion of the code.

Running CoreMark produces a single-number score, allowing users to make quick comparisons between processors. Results can be uploaded to the CoreMark website for certification, as CoreMark has a standard format for reporting results.

This document aims to present and explain the results and the process of benchmarking Renesas RA MCUs using CoreMark.

This application note walks you through all the steps necessary to benchmark using CoreMark.

Required Resources

Development of tools and software

- e² studio v2024-07
- Renesas Flexible Software Package (FSP) v5.5.0
- Arm Compiler 6.21
- IAR Embedded Workbench v9.50.2

Hardware

- Renesas RA kit: EK-RA6M5

Reference Manuals

- RA Flexible Software Package Documentation Release v5.5.0
- User’s Manual: Renesas RA6M5 Group User’s Manual Rev.1.10
- Schematics: EK-RA6M5-v1.0

Contents

1. CoreMark Project

The official CoreMark source is available at EEMBC GitHub. As we plan to use CoreMark on a bare-metal target, the source files we are going to use consist of the following C source and header files:

- coremark.h
- core_main.c
- core_list_join.c
- core_matrix.c
- core_state.c
- core_util.c
- core_portme.c
- core_portme.h
- cvt.c
- ee_printf.c

The three key algorithms used are related to linked lists, matrix multiplication, and state machines.

At EEMBC GitHub, you will also find more information on the rules for building and running the CoreMark code.

The procedure to create a CoreMark project for Renesas RA MCUs is as follows.

- Create a Bare-Metal Minimal Project using e^2 studio and Flexible Software Package (FSP)
- Copy CoreMark source code to the “src” folder
- Add a 32-bit general-purpose timer (GPT) to the project
- Change the main stack size setting to 0x4000 to accommodate CoreMark benchmarking
- Exclude the main.c generated by FSP from the build
- Update project optimization with the maximum speed option
- Port core_portme.h, core_portme.c to add necessary code for the GPT
- Port ee_printf.c to print benchmarking results.

This document explains the procedure for EK-RA6M5, but the same can be applied to other RA MCUs.

2. Run CoreMark on Renesas RA MCUs

Apart from the official CoreMark source, in order to be able to replicate exactly the process used in benchmarking the RA MCUs, you will need the e^2 studio IDE together with FSP. You can download and install setup_fsp_v5_5_0_e2s_v2024-07.exe from https://github.com/renesas/fsp/releases

Moreover, you will need the IAR Arm compiler available at https://www.iar.com/products/architectures/arm/ and the Arm compiler available at https://developer.arm.com/documentation/ka005198/latest. You can use the Arm compiler in Keil MDK installation for CoreMark benchmarking.

2.1 Integrating Toolchains with e^2 studio

2.1.1 IAR Embedded Workbench Plugin

Download and install the IAR Embedded workbench before you integrate the IAR compiler with e^2 studio.

Start e^2 studio, then select “Help -> IAR Embedded Workbench plugin manager”.

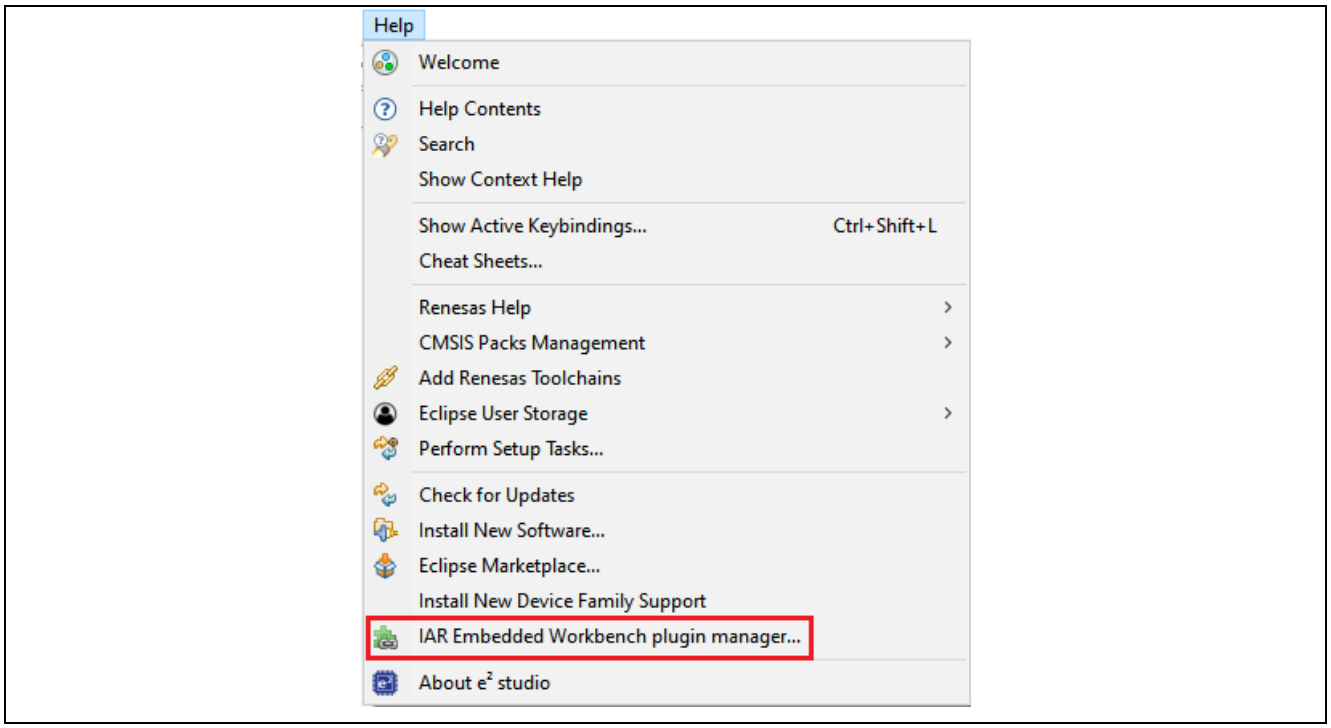


Figure 1. Select IAR Embedded Workbench Plugin Manager

You choose the desired toolchain and press install.

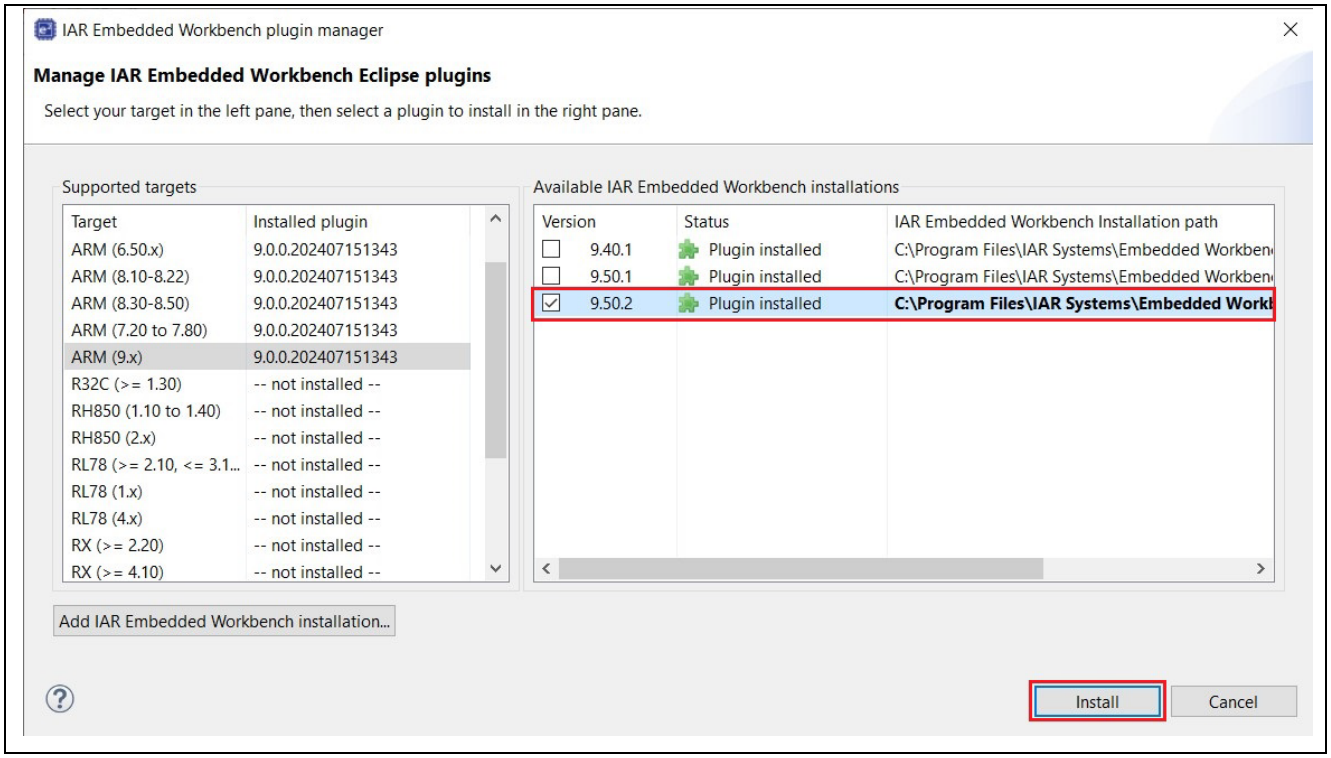


Figure 2. Select IAR Plugin

The bottom right corner of the e² studio IDE will show configuration progress.

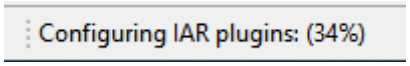


Figure 3. IAR Configuration Progress

Press “Next”, then “Next”. Accept the terms of the license agreements then click “Finish”.

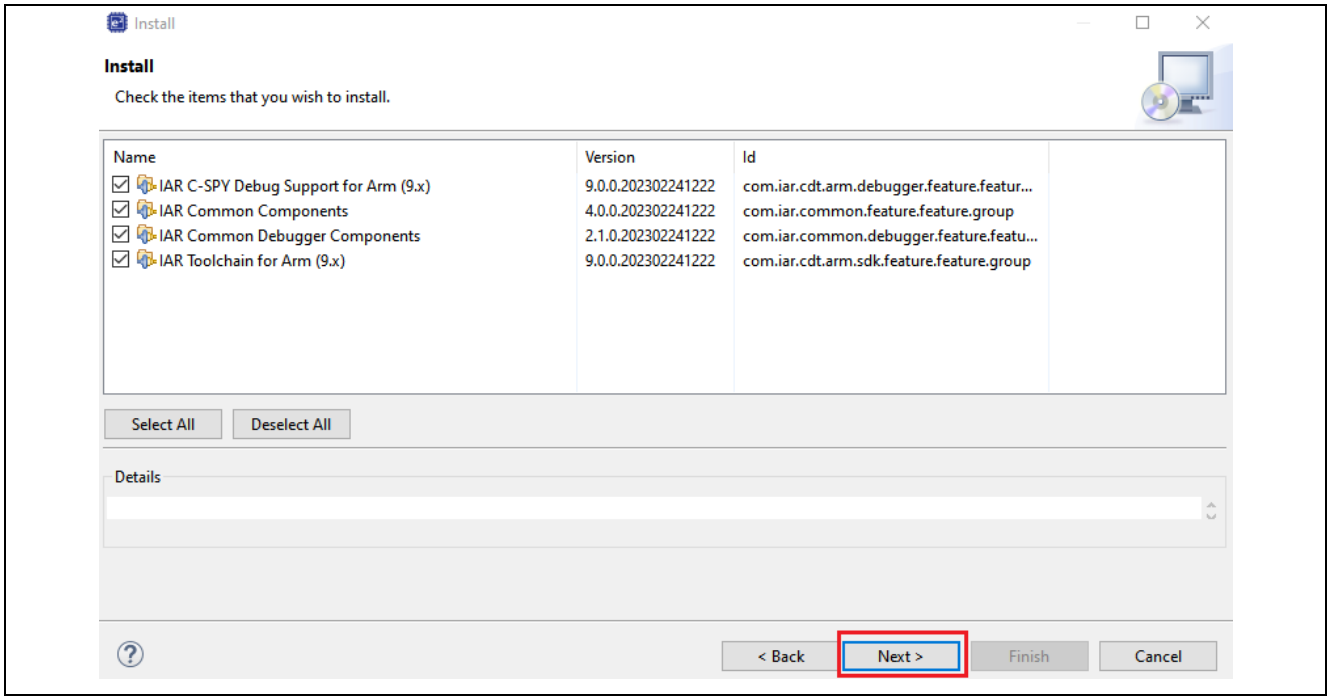


Figure 4. Install the IAR Embedded Workbench Plugin

The bottom right corner of the e² studio IDE will show the installation progress.

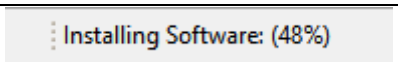


Figure 5. IAR Plugin Installation Process

Wait for the plugin to be installed and click “Restart Now” to complete the installation process.

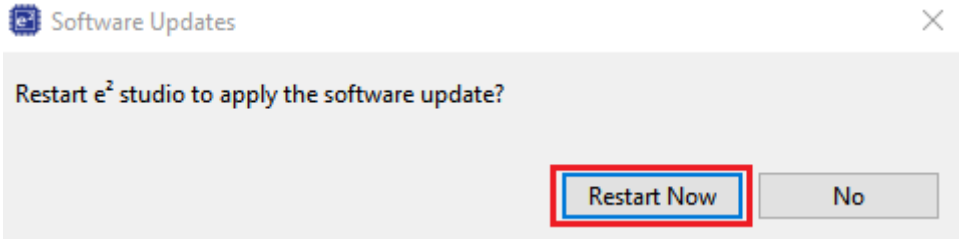


Figure 6. Restart e² studio

2.1.2 Integrate with Arm Compiler

Download and install the Arm compiler or Keil MDK before you integrate the Arm compiler with e² studio.

Start e² studio, then select “Window -> Preferences” to add toolchains to e² studio.

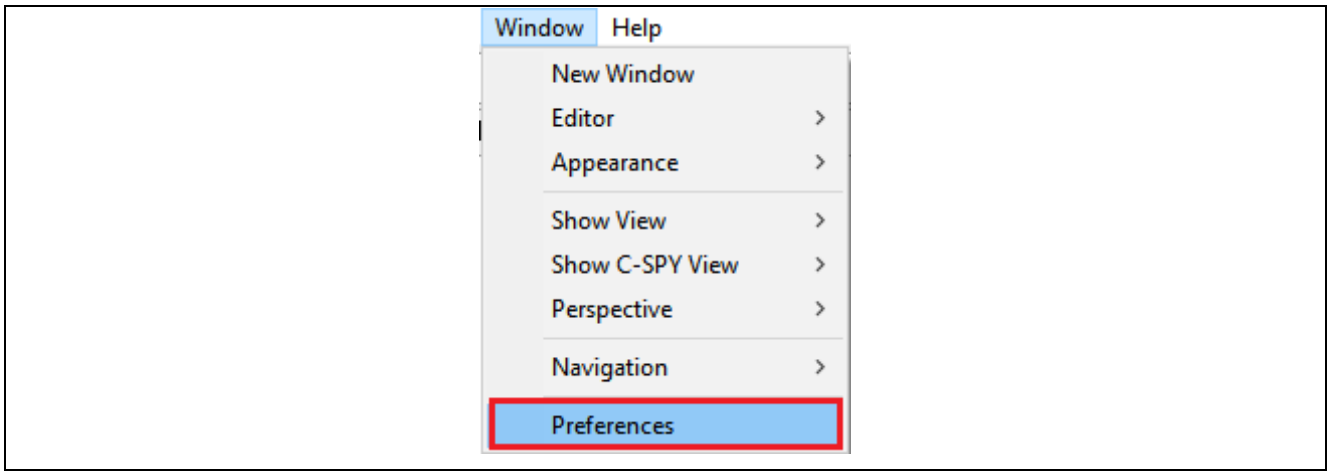


Figure 7. e² studio Preferences

Select the desired Arm Compiler toolchain, then click “Apply and Close” to add the Arm compiler to e² studio.

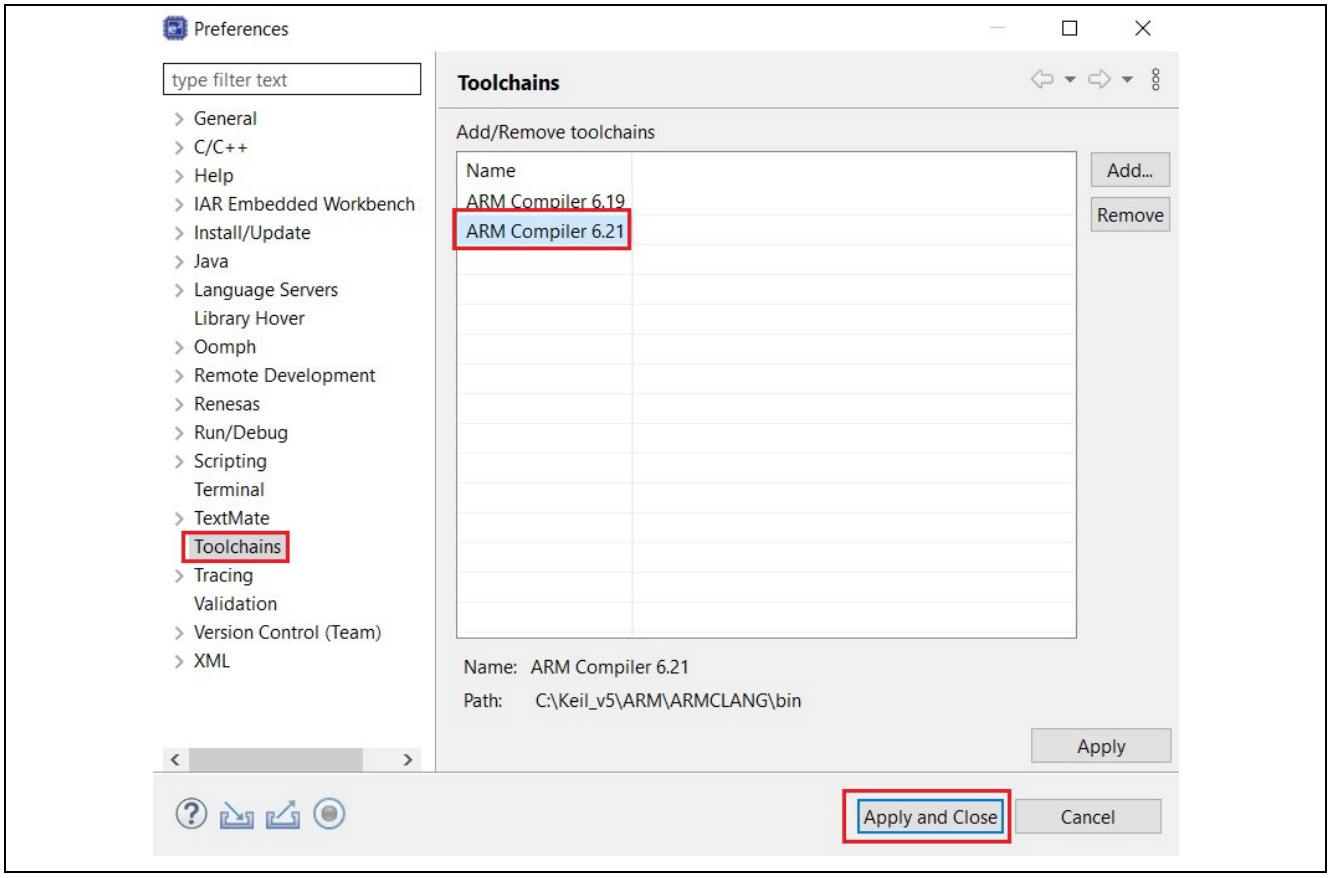


Figure 8. Add Arm Compiler to e² studio

If the Arm compiler is not present in the Toolchains windows, click “Add”, then browse to the toolchain folder, e.g., C:\Keil_v5\ARM\\ARMCLANG\bin

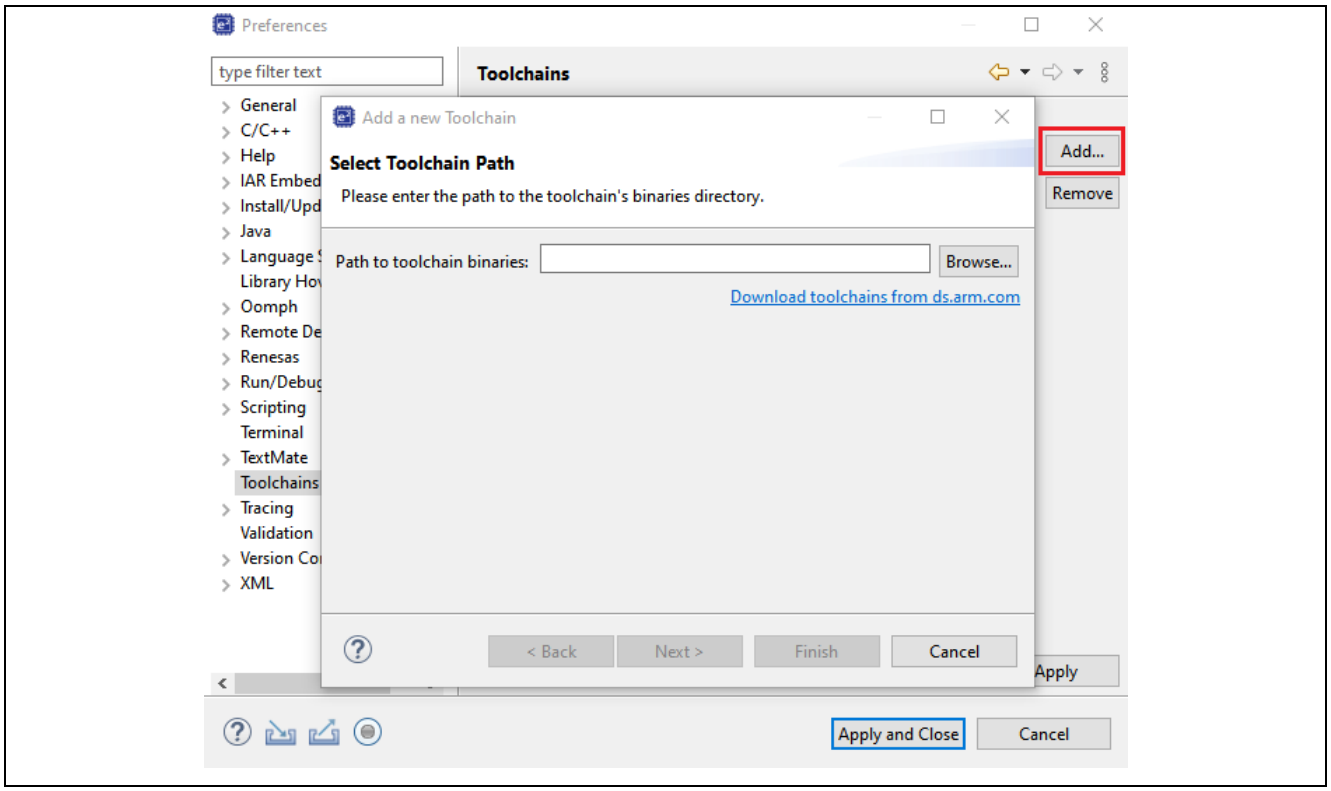


Figure 9. Add Toolchain’s Path

2.2 Create CoreMark e² studio Project Used for Benchmarking using the IAR Compiler

Ensure you integrated the IAR compiler with e² studio before creating a CoreMark project.

On e² studio, select “File -> New-> C/C++ Project, then click “Next”.

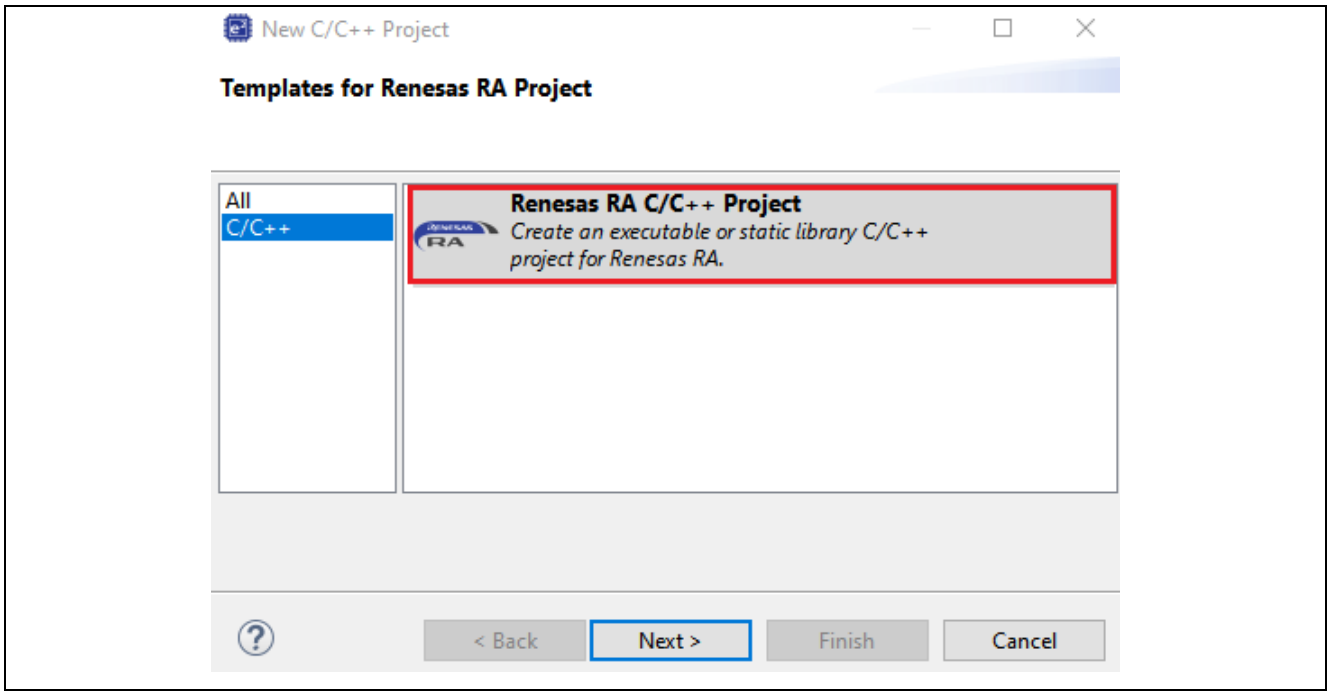


Figure 10. Select C/C++ Template

Name your project an appropriate name, e.g., RA6M5_CoreMark_IAR for EK-RA6M5 kit using the IAR compiler.

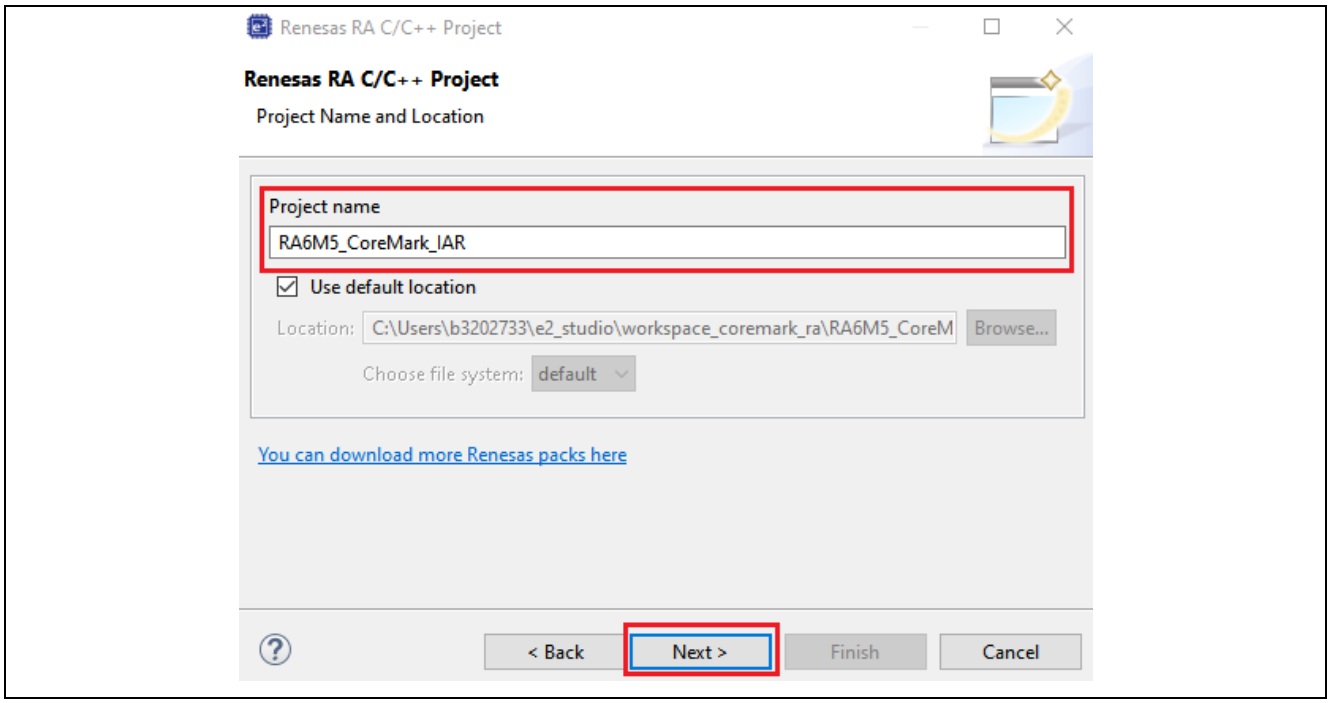


Figure 11. Name Your Project

Select the Board, Device, and Toolchain you want to use for benchmarking.

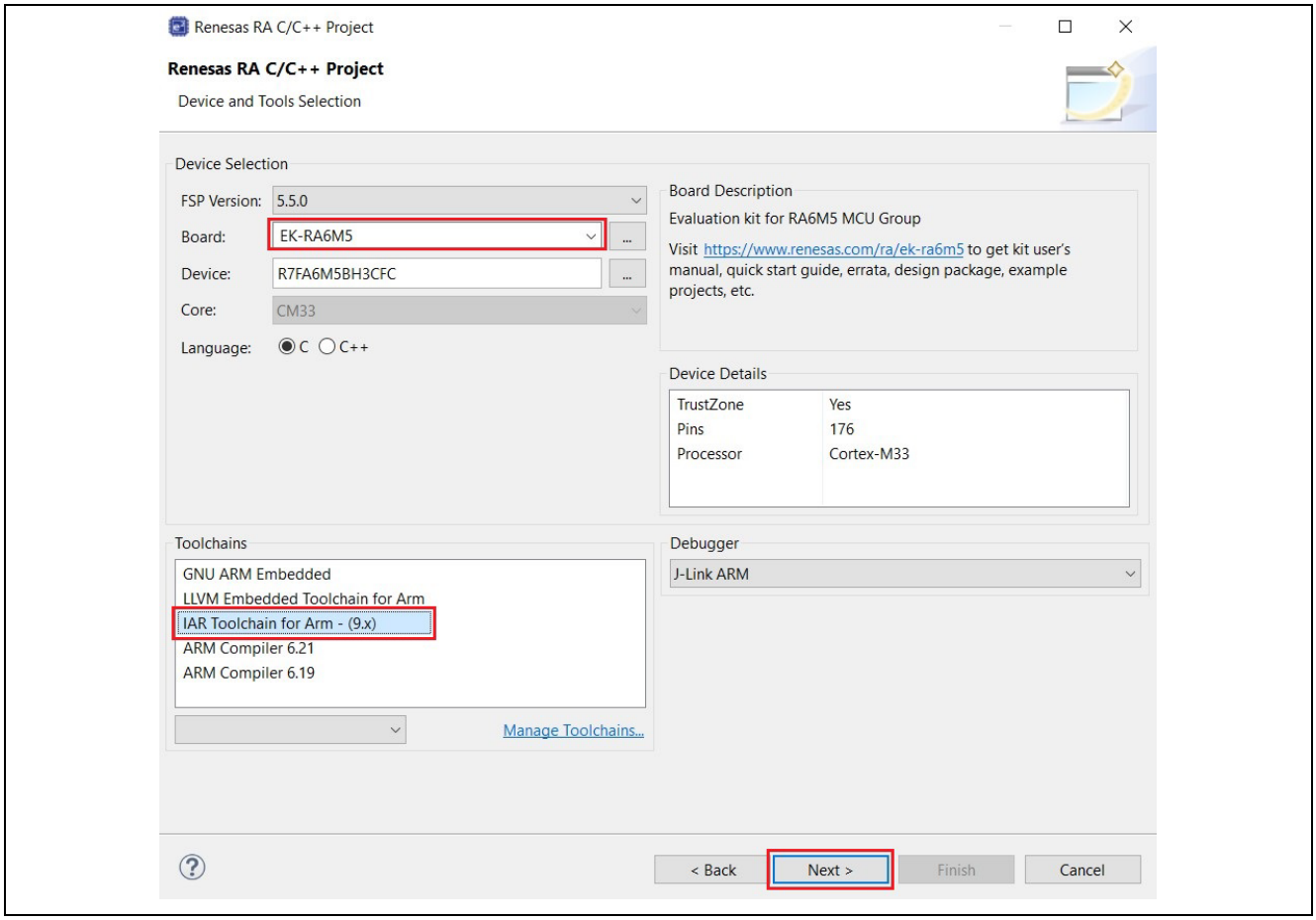


Figure 12. RA Project Options

Select Flat (Non-TrustZone) Project.

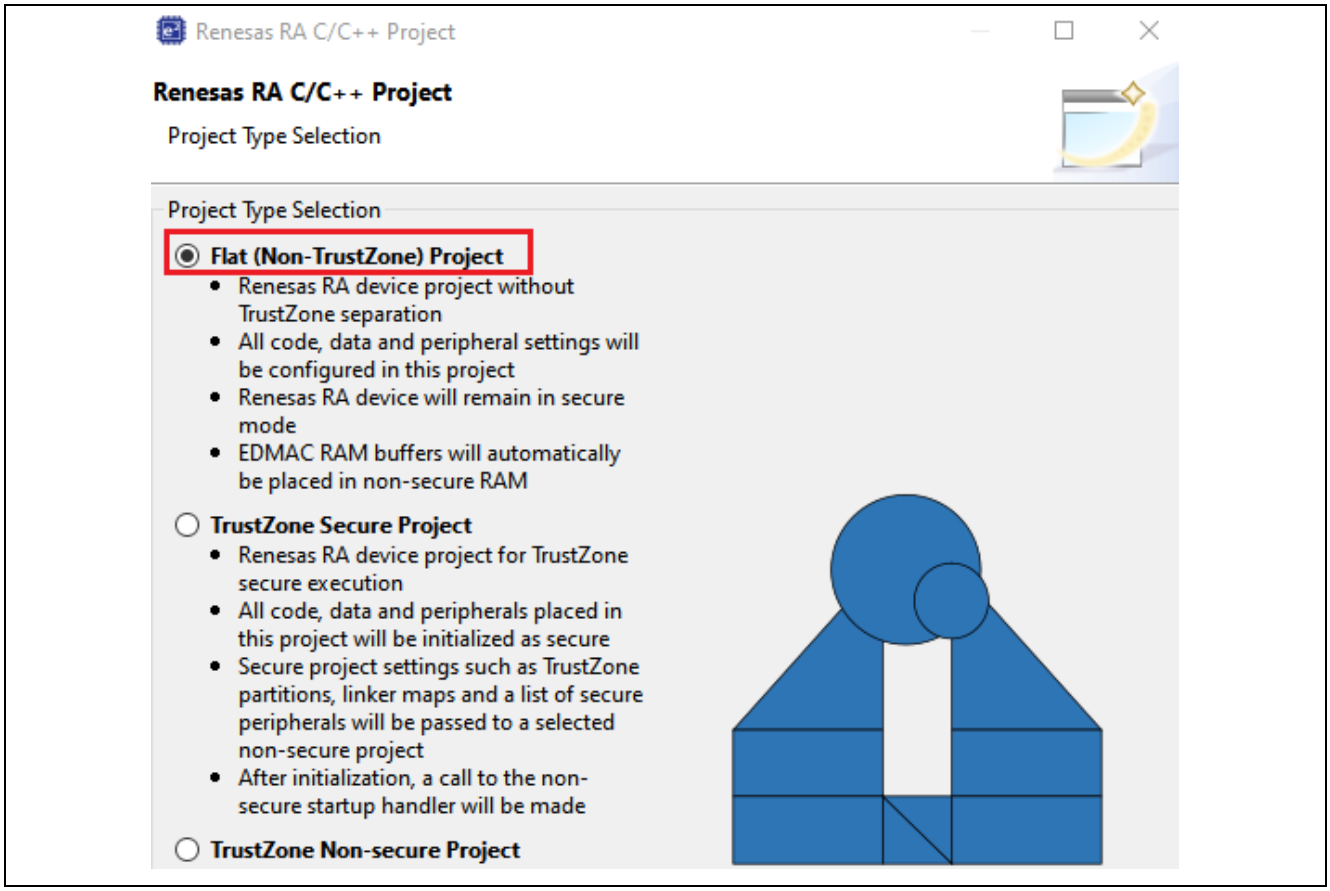


Figure 13. Flat Project Selection

After this step, select Executable project type with No RTOS.

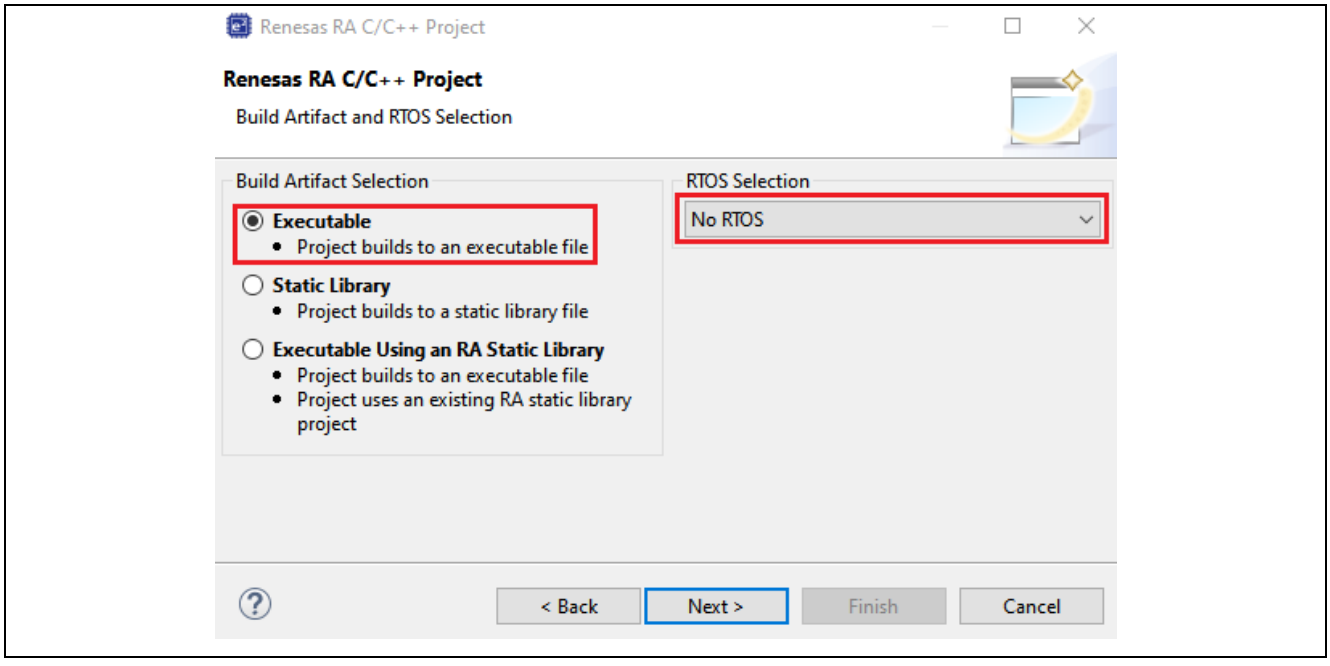


Figure 14. Select No RTOS Project

Select Bare Metal – Minimal Project Template. Click “Finish” to generate the project.

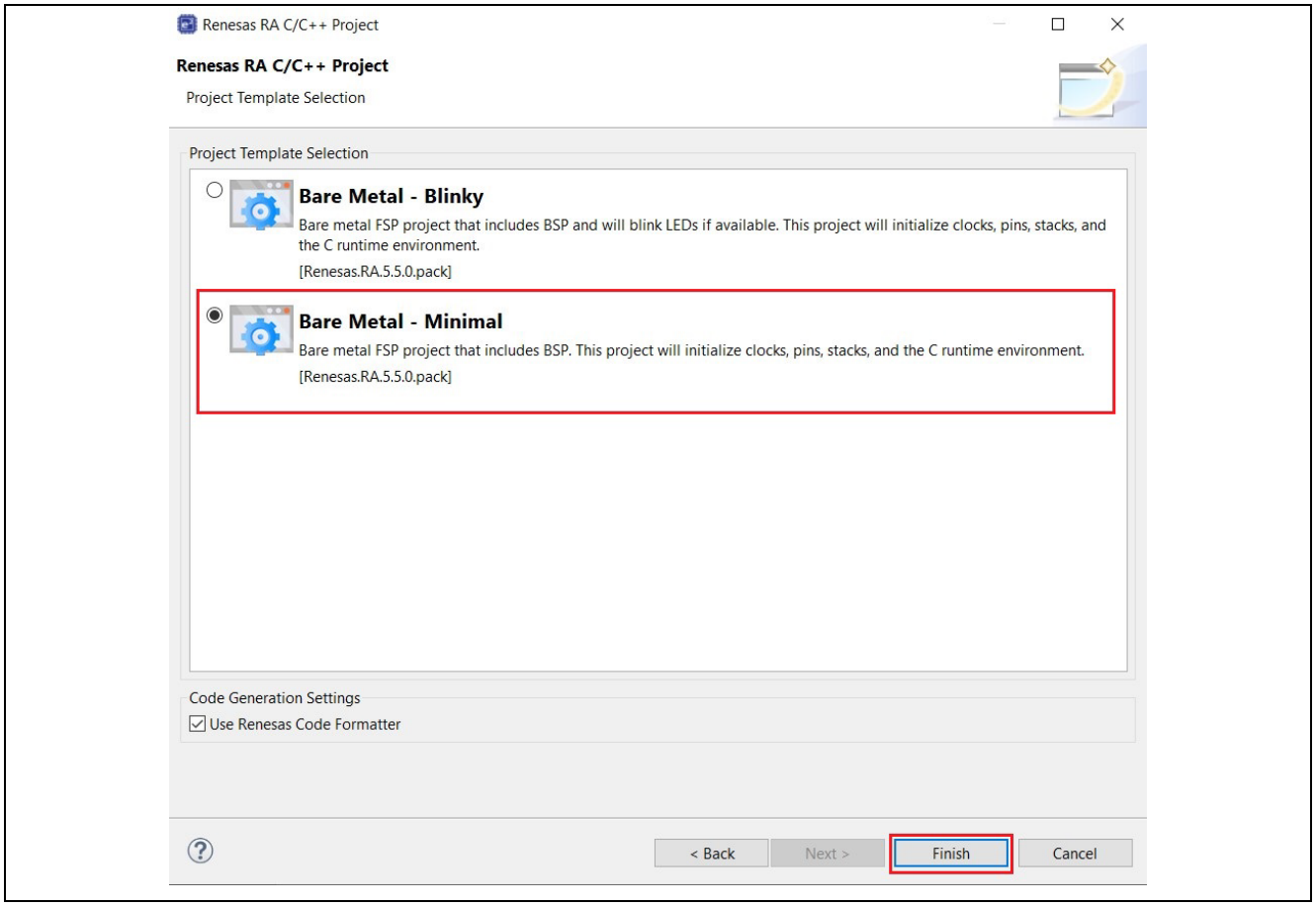


Figure 15. Bare Metal Minimal Option

2.3 Add CoreMark to e² studio Project

Copy the CoreMark source code to the “src” folder in your newly created project. The project structure should look as follows.

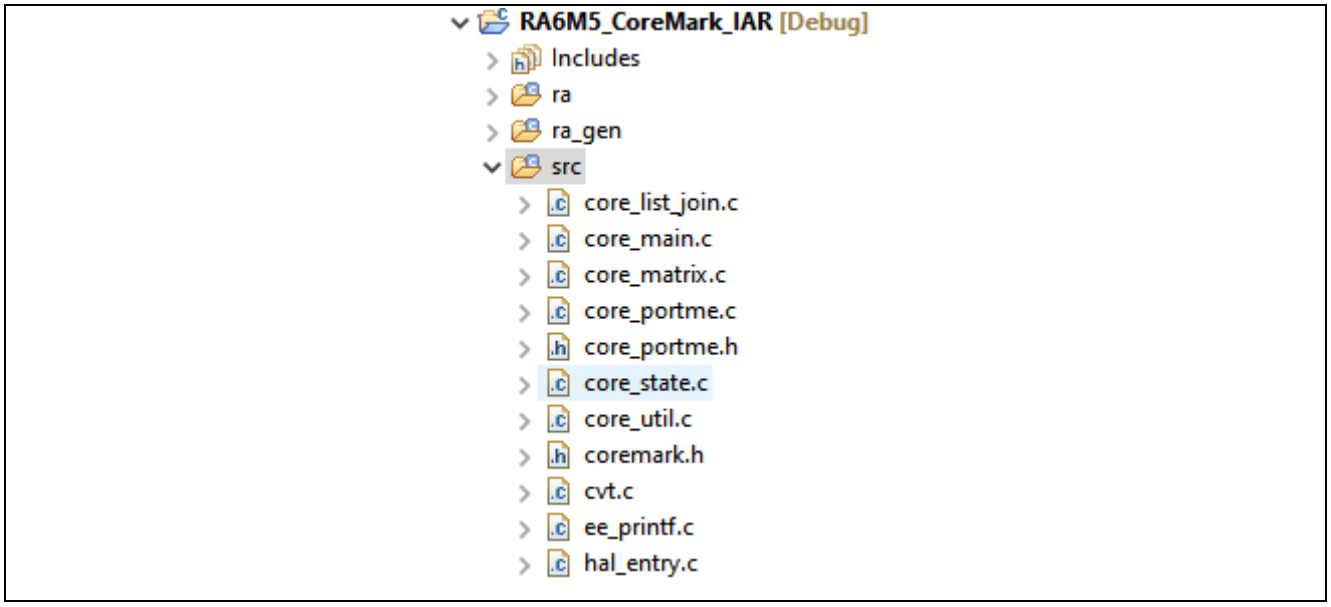


Figure 16. CoreMark Project

You now need to add a periodic timer and modify the core_portme.c source file in order to use the modified barebones_clock(), portable_init(core_portable *p, int *argc, char *argv[]) and portable_fini(core_portable *p) functions.

To add a new periodic timer, open the configuration.xml file and go to Stacks. You should see something similar to the picture below.

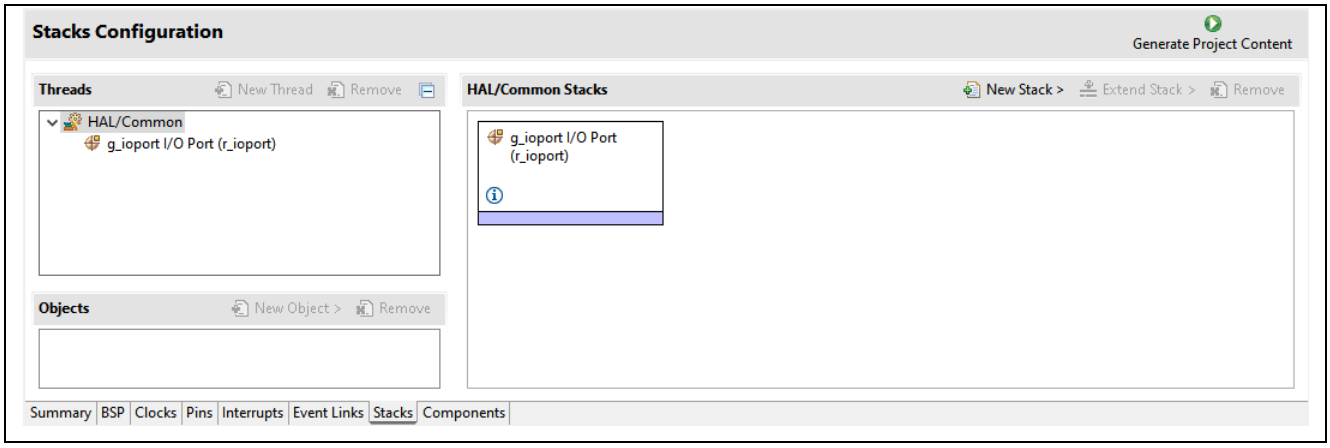


Figure 17. Stack Configuration

2.4 Add Timer for Benchmarking

The next step is to add a New Stack, then select Timers and, finally, Timer, General PWM(r_gpt).

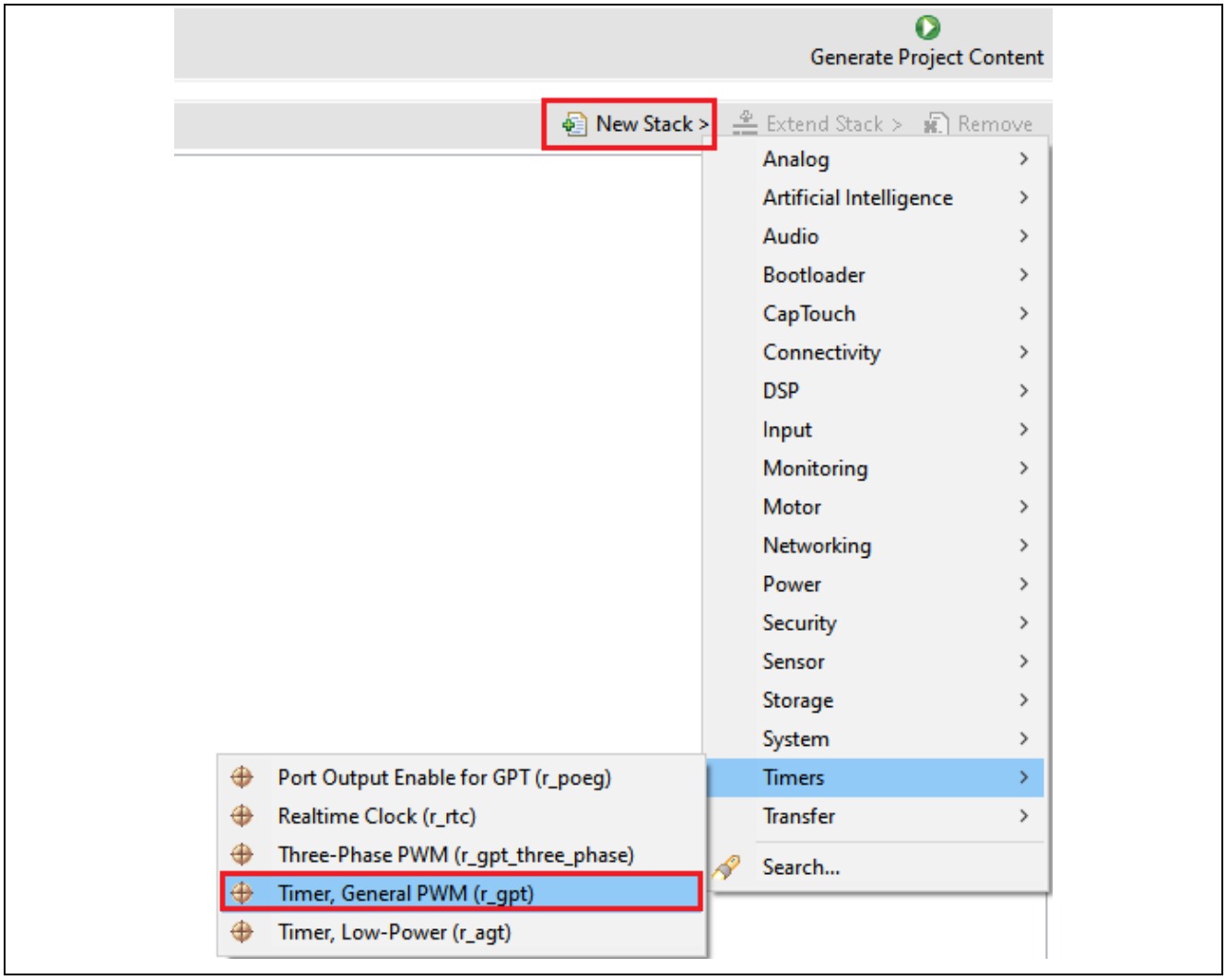


Figure 18. Add GPT Timer

After adding the GPT timer, you need to edit the settings. Clicking on the block representing the newly added GPT timer, then go to the Properties Window. You use this Properties window to change the timer's name to g_timer_periodic, the period to 50, and the period unit to Seconds. You also need to expand the Interrupts block, add the Callback as timer_callback and set the Priority to 2. The following image captures the changes needed.

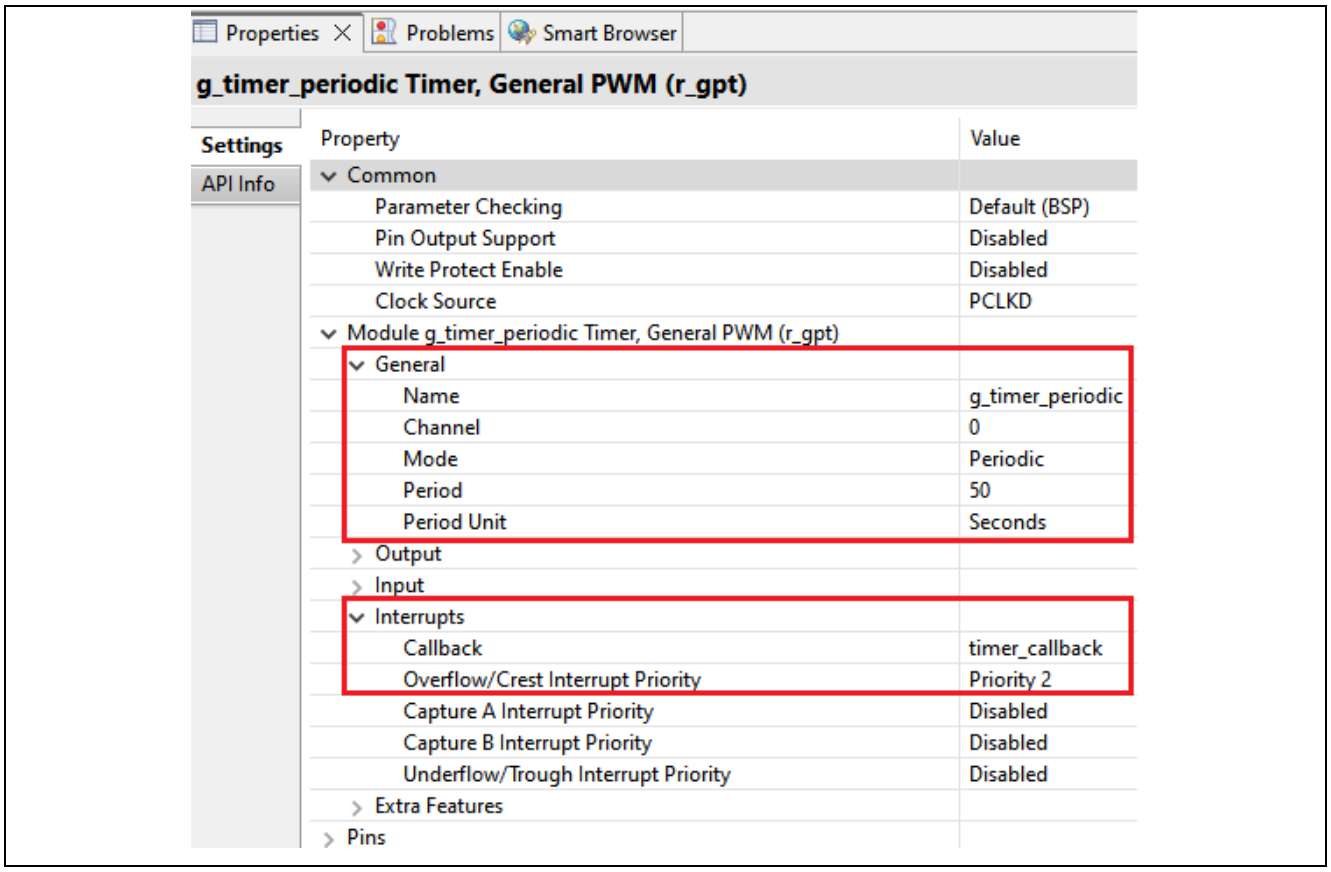


Figure 19. GPT Configuration

2.5 Update Main Stack

Change the Main Stacks Size in BSP properties to 0x4000.

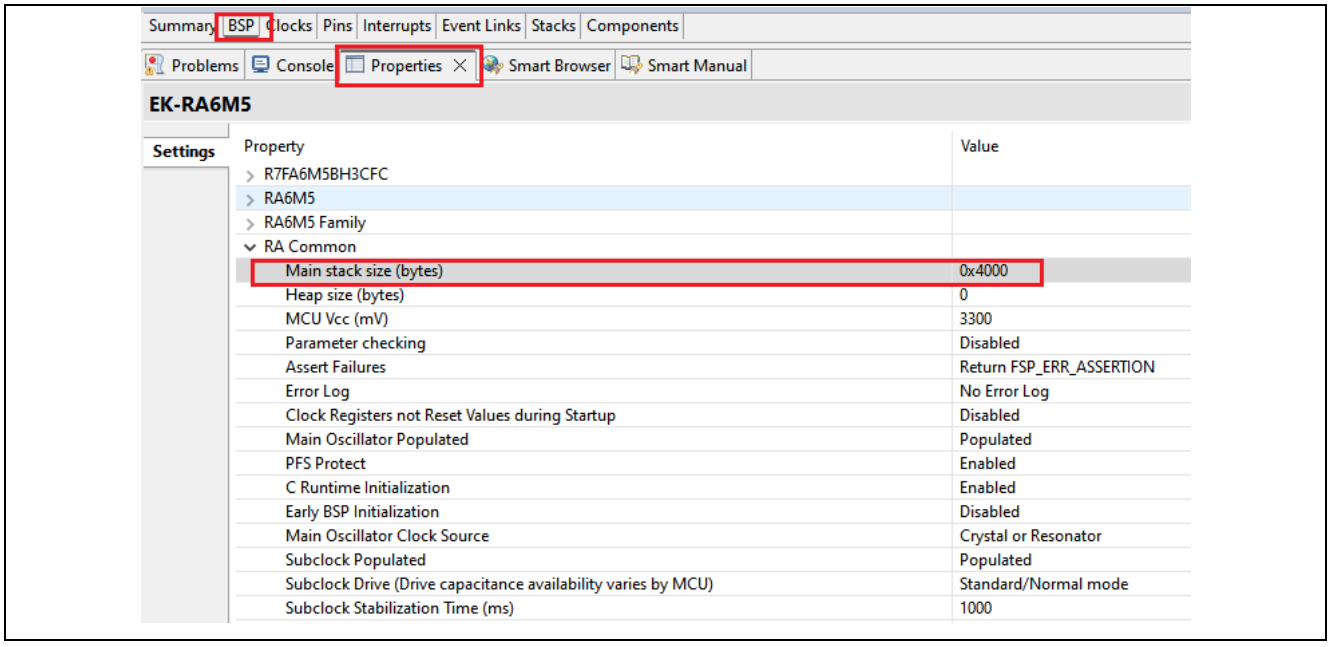


Figure 20. Change Main Stack Size

Click “Generate Project Content”, then the next step is to modify the source files.

Right click on the ra_gen\main.c and exclude it from the Build, so there is no conflict with the main from core_main.c.

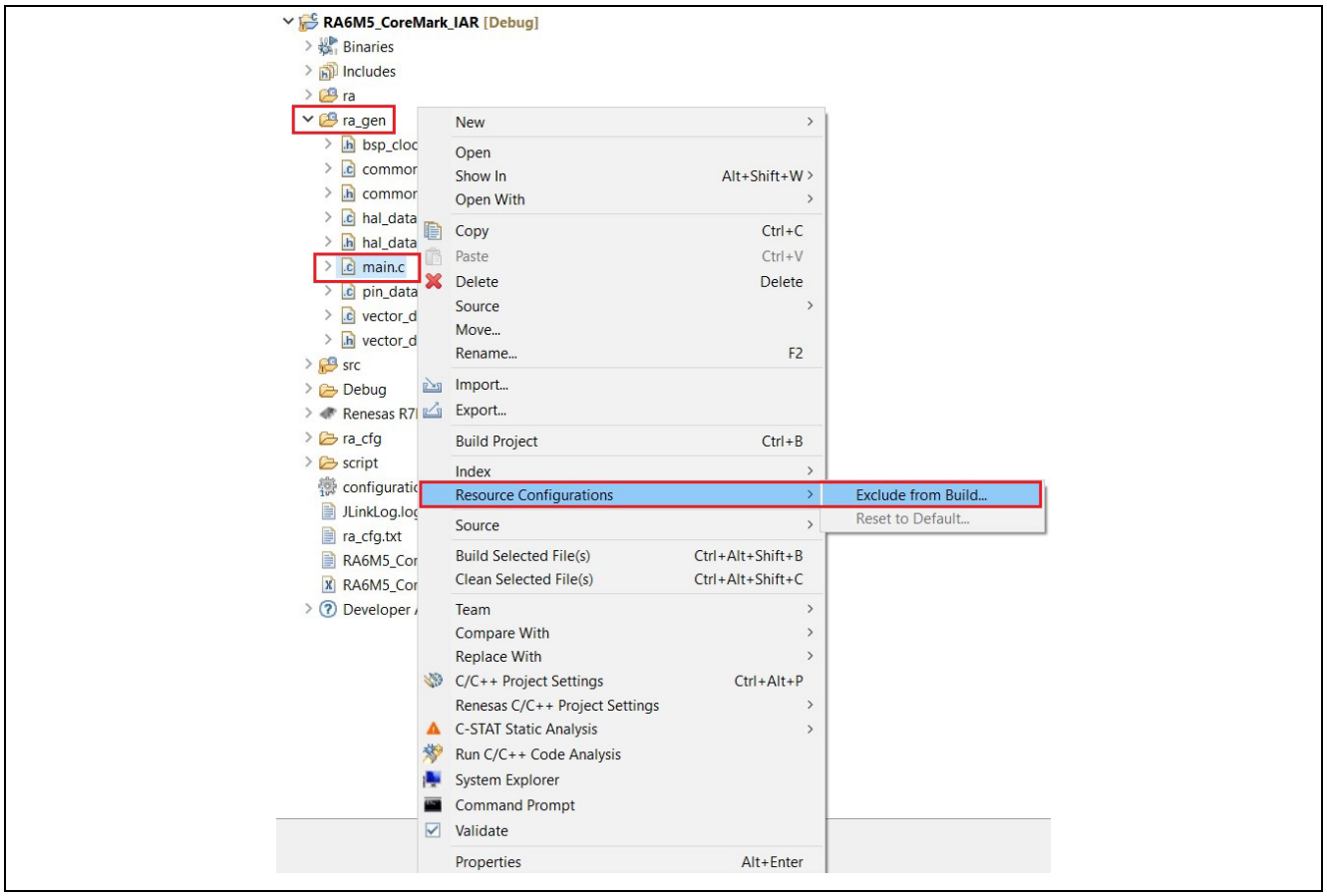


Figure 21. Exclude main.c from Build

2.6 Port CoreMark Code

You modify the core_portme.h and the core_portme.c in the “src” folder.

In core_portme.h, add “#include <stddef.h>” before the code “typedef size_t ee_size_t;”, as shown below.

```
/* Data Types :
        To avoid compiler issues, define the data types that need ot be used for
    8b, 16b and 32b in <core_portme.h>.

        *Imprtant* :
        ee_ptr_int needs to be the data type used to hold pointers, otherwise
    coremark may fail!!!
*/
typedef signed short   ee_s16;
typedef unsigned short ee_u16;
typedef signed int     ee_s32;
typedef double         ee_f32;
typedef unsigned char  ee_u8;
typedef unsigned int   ee_u32;
typedef ee_u32         ee_ptr_int;
#include <stddef.h>
typedef size_t         ee_size_t;
#define NULL ((void *)0)
```

Figure 22. Add “#include <stddef.h>”

Also, in core_portme.h, modify the “#define COMPILER_FLAGS” depending on the toolchain used. If you use IAR Compiler version 9.50.2, change it to #define COMPILER_FLAGS "High Speed; No size constraints".

The code should look as follows.

```
/* Definitions : COMPILER_VERSION, COMPILER_FLAGS, MEM_LOCATION
#ifndef COMPILER_VERSION
#ifdef __GNUC__
#define COMPILER_VERSION "GCC"__VERSION__
#else
#define COMPILER_VERSION "IAR Compiler version 9.50.2"
#endif
#endif
#ifndef COMPILER_FLAGS
#define COMPILER_FLAGS \
    "High Speed; No size constraints" /* "Please put compiler flags here (e.g. -o3)" */
#endif
#ifndef MEM_LOCATION
#define MEM_LOCATION "STACK"
#endif
```

Figure 23. Modify core_portme.h

In core_portme.c, before the barebones_clock() function, add the below code.

```
volatile ee_s32 seed4_volatile = ITERATIONS;
volatile ee_s32 seed5_volatile = 0;

/* Since we set the timer period to 50s, the actual value is 50000000 with the counter clock at 100MHz/2 */
#define CLOCKS_PER_SEC 50000000
timer_info_t g_timer_info;
uint32_t g_capture_overflows = 0U;
```

Figure 24. Modify core_portme.c

You can check and correct the CLOCKS_PER_SEC setting by getting the correct value from the clock_frequency, shown in the figure below, when running the project for the first time.

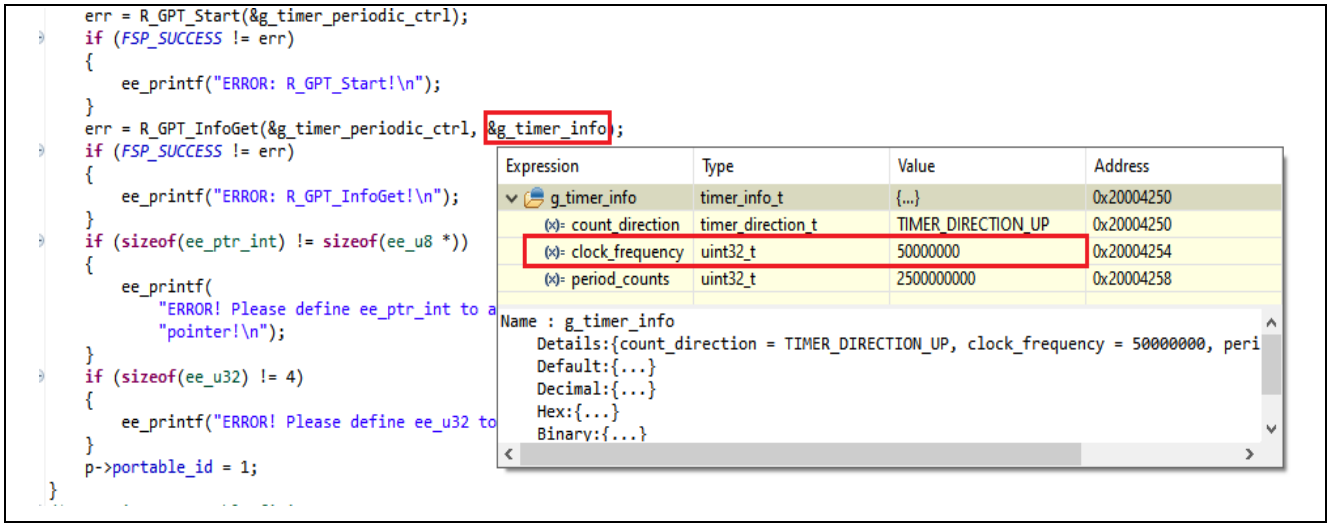


Figure 25. Check CLOCK_PER_SEC Setting

Change the barebones_clock() functions as follows.

```
/* Porting : Timing functions
    How to capture time and convert to seconds must be ported to whatever is
   supported by the platform. e.g. Read value from on board RTC, read value from
   cpu clock cycles performance counter etc. Sample implementation for standard
   time.h and windows.h definitions included.
*/
CORETIMETYPE
barebones_clock()
{
    fsp_err_t err = FSP_SUCCESS;
    timer_status_t status;
    err = R_GPT_StatusGet (&g_timer_periodic_ctrl, &status);
    if (FSP_SUCCESS != err)
    {
        ee_printf("ERROR: R_GPT_StatusGet!\n");
    }
    /* The period is set to 50s we shouldn't overflow but just in case
       report an error if we do. If we set the a shorter period we need to do:
       info.period_counts * g_capture_overflows   */
    if(g_capture_overflows > 0)
    {
        ee_printf("ERROR: Timer overflow!\n");
    }
    return status.counter;
}
```

Figure 26. bareborns_clock() Function

Then change the portable_fini(core_portable *p), portable_init(core_portable *p, int *argc, char *argv[]) to add the GPT timer that is needed for benchmarking.

```
/* Function : portable_init
      Target specific initialization code
      Test for some common mistakes.
*/
void
portable_init(core_portable *p, int *argc, char *argv[])
{
    fsp_err_t err = FSP_SUCCESS;

    /* Flush C cache */
    uint32_t * c_cache = (uint32_t *)0x40007004;
    *c_cache = 1;
    /* Enable C cache */
    c_cache = (uint32_t *)0x40007000;
    *c_cache = 1;

    /* Flush S cache */
    uint32_t * s_cache = (uint32_t *)0x40007044;
    *s_cache = 1;
    /* Flush S cache */
    s_cache = (uint32_t *)0x40007040;
    *s_cache = 1;


    /* Initialize GPT Timer */
    err = R_GPT_Open(&g_timer_periodic_ctrl, &g_timer_periodic_cfg);
    if (FSP_SUCCESS != err)
    {
        ee_printf("ERROR: R_GPT_Open!\n");
    }
    err = R_GPT_Start(&g_timer_periodic_ctrl);
    if (FSP_SUCCESS != err)
    {
        ee_printf("ERROR: R_GPT_Start!\n");
    }
    err = R_GPT_InfoGet(&g_timer_periodic_ctrl, &g_timer_info);
    if (FSP_SUCCESS != err)
    {
        ee_printf("ERROR: R_GPT_InfoGet!\n");
    }
    if (sizeof(ee_ptr_int) != sizeof(ee_u8 *))
    {
        ee_printf(
            "ERROR! Please define ee_ptr_int to a type that holds a "
            "pointer!\n");
    }
    if (sizeof(ee_u32) != 4)
    {
        ee_printf("ERROR! Please define ee_u32 to a 32b unsigned type!\n");
    }
    p->portable_id = 1;
}
```

Figure 27. portable_init Function

```
/* Function : portable_fini
        Target specific final code
*/
void
portable_fini(core_portable *p)
{
    fsp_err_t err = FSP_SUCCESS;

    err = R_GPT_Stop(&g_timer_periodic_ctrl);
    if (FSP_SUCCESS != err)
    {
        ee_printf("ERROR: R_GPT_Stop!\n");
    }
    p->portable_id = 0;
    BSP_CFG_HANDLE_UNRECOVERABLE_ERROR(0);
}
```

Figure 28. portable_fini Function

At the end of the file, add the callback method function of the GPT timer.

```
/* Example callback called when timer expires. */
void timer_callback (timer_callback_args_t * p_args)
{
    if (TIMER_EVENT_CYCLE_END == p_args->event)
    {
        g_capture_overflows++;
    }
}
```

Figure 29. Add timer_callback to core_portme.c

In ee_printf.c, change the uart_send_char(char c) and add the code below for printing benchmarking results.

```
#define MAXBUFFER 1000
volatile char uart_buffer[MAXBUFFER + 1];
volatile unsigned int uart_buffer_cnt = 0;

void
uart_send_char(char c)
{
    if(uart_buffer_cnt < MAXBUFFER)
    {
        uart_buffer[uart_buffer_cnt++] = c;
        uart_buffer[uart_buffer_cnt] = '\0';
    }
    else
    {
        uart_buffer[uart_buffer_cnt] = '\0';
    }
}
```

Figure 30. Add uart_send_char Function

In the project properties setting, add “ITERATIONS=9000” to the IAR C/C++ Compiler for ARM->Preprocessor.

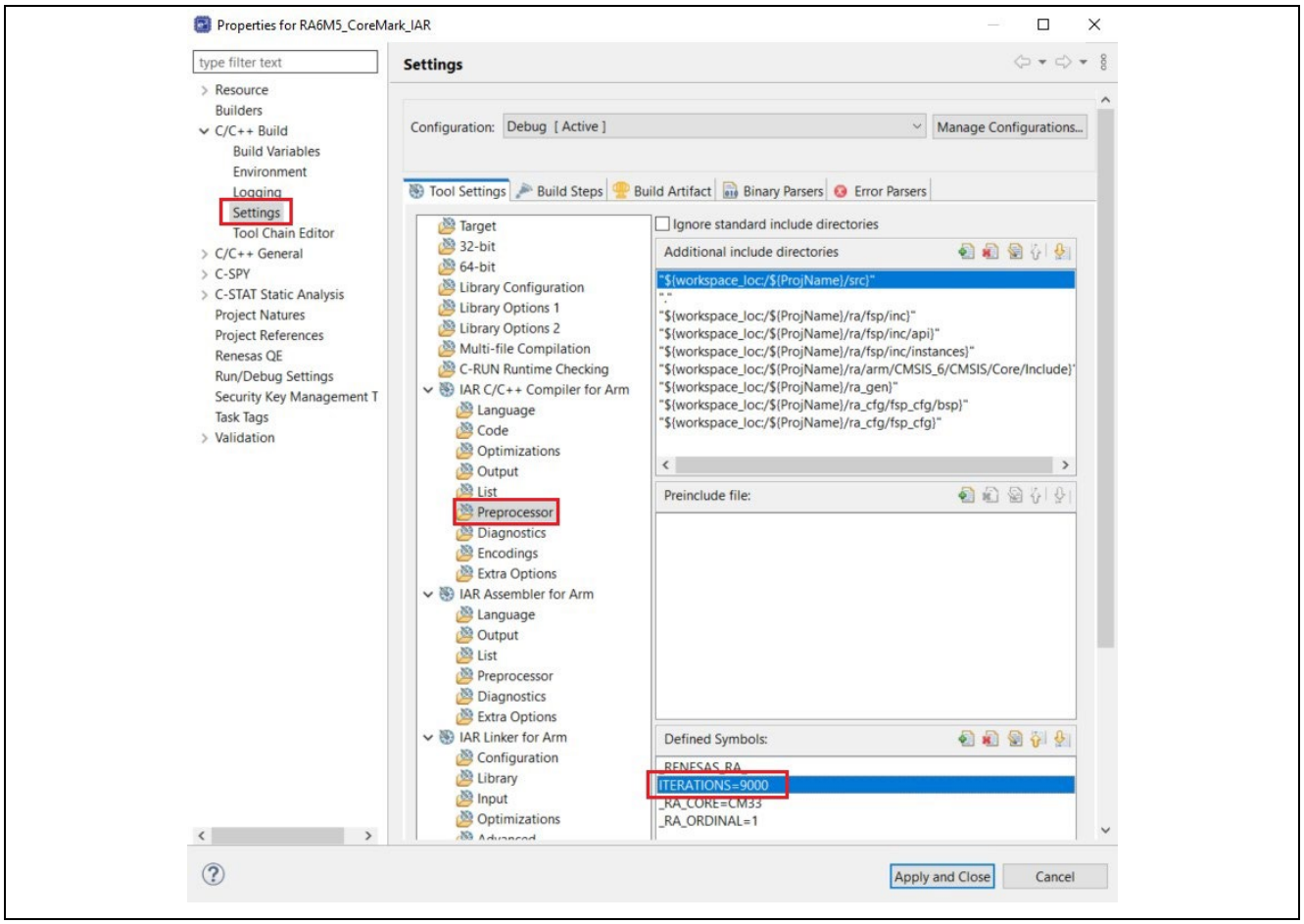


Figure 31. Preprocessor Setting

In the project properties setting, change IAR C/C++ Compiler for ARM->Optimization to “High, Speed” with “No size constraints”.

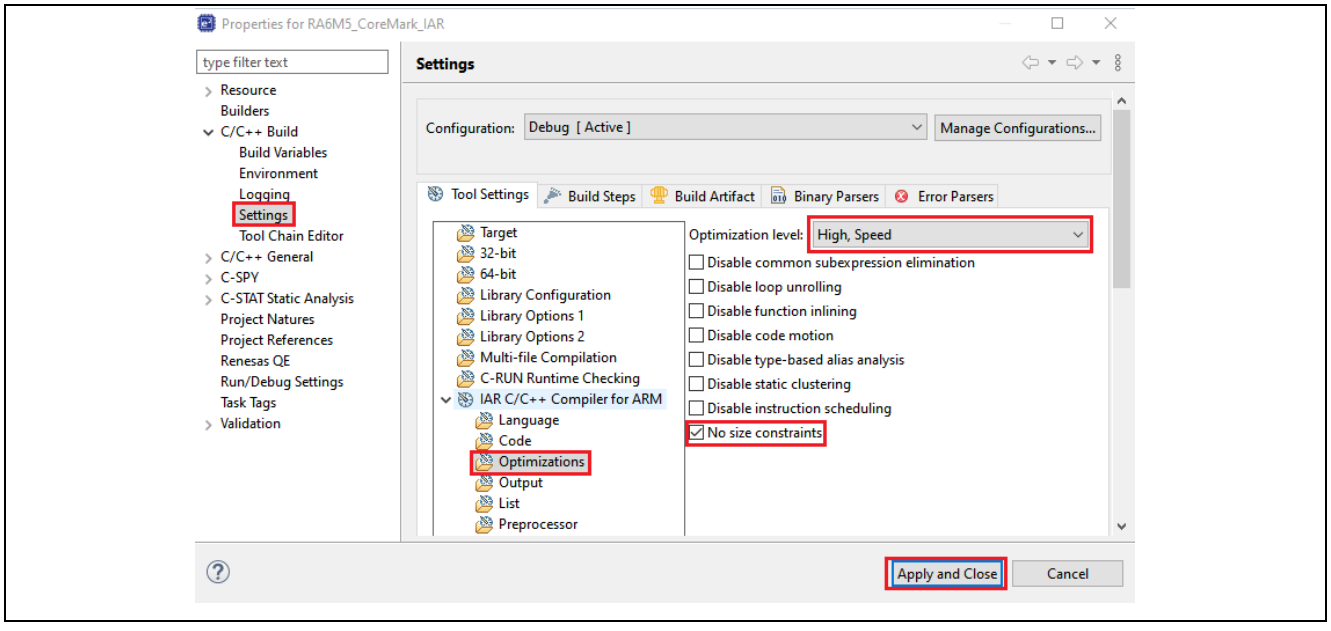


Figure 32. Optimization Setting

Now, you can build the project without errors.

2.7 Create CoreMark e² studio Project Used for Benchmarking using Arm Compiler

Ensure you integrated the Arm compiler with e² studio before creating a CoreMark project. Select the Board, Device, and Toolchain you want to use for benchmarking and process to create an Arm compiler-based project similar to the IAR compiler. Follows sections 2.3, 2.4, 2.5, and 2.6 to add the GPT module, configure your project, and port the CoreMark. Note that you need a commercial license to use "--lto" option in Arm Compiler.

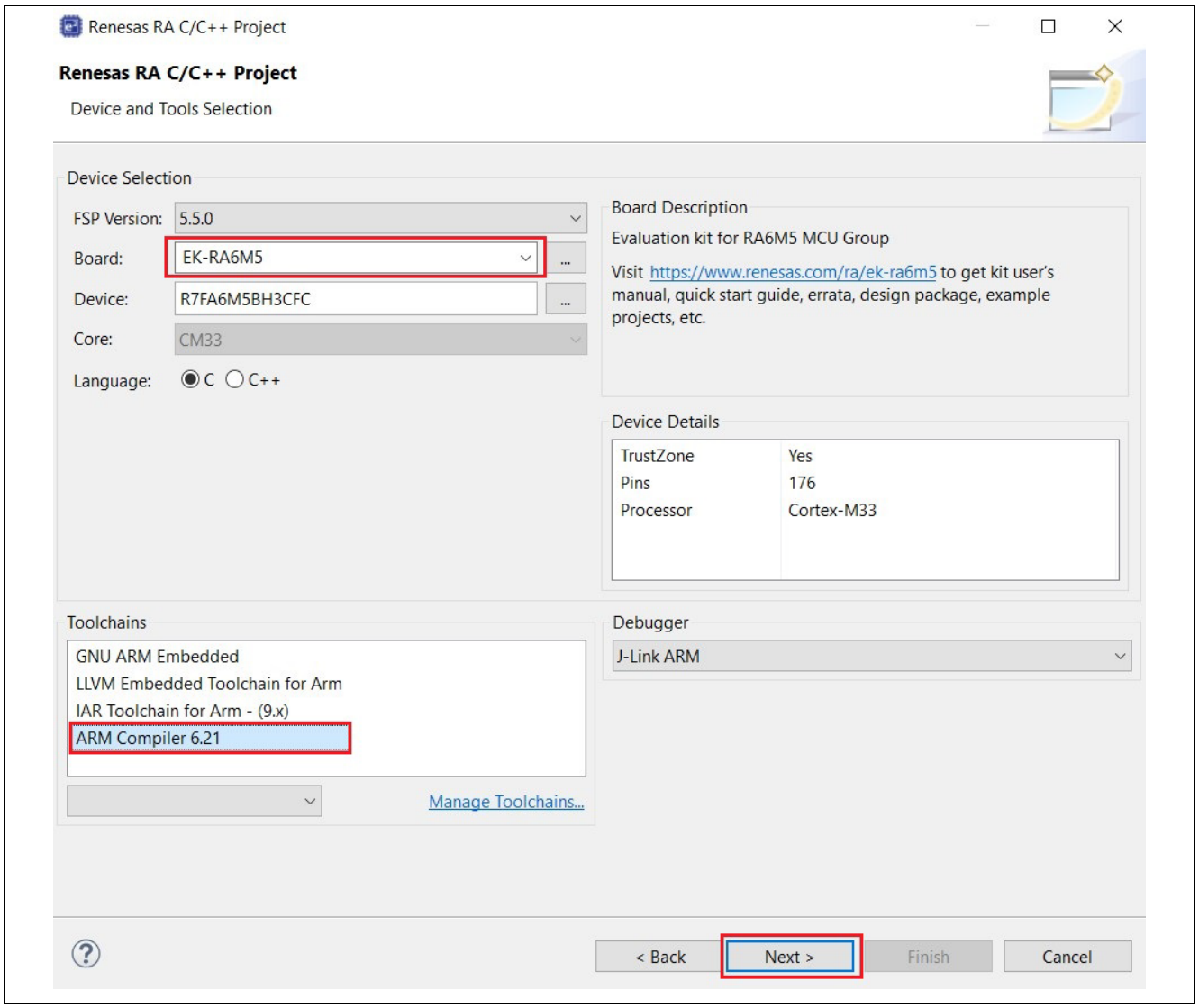


Figure 33. Create An Arm Compiler - Based Project Options

Also, in core_portme.h, modify the “#define COMPILER_FLAGS” depending on the toolchain used. In the case of Arm Compiler, change it to "-Omax".

The code should look as follows.

```
/* Definitions : COMPILER_VERSION, COMPILER_FLAGS, MEM_LOCATION
        Initialize these strings per platform
*/
#ifndef COMPILER_VERSION
#ifdef __GNUC__
#define COMPILER_VERSION __VERSION__
#else
#define COMPILER_VERSION "Please put compiler version here (e.g. gcc 4.1)"
#endif
#endif
#ifndef COMPILER_FLAGS
#define COMPILER_FLAGS "-Omax"
#endif
#ifndef MEM_LOCATION
#define MEM_LOCATION "STACK"
#endif
```

Figure 34. Modify “#define COMPILER_FLAGS”

In the project’s Properties-> ARM C Compiler 6.15->Miscellaneous->Other flags, add “-Omax”.

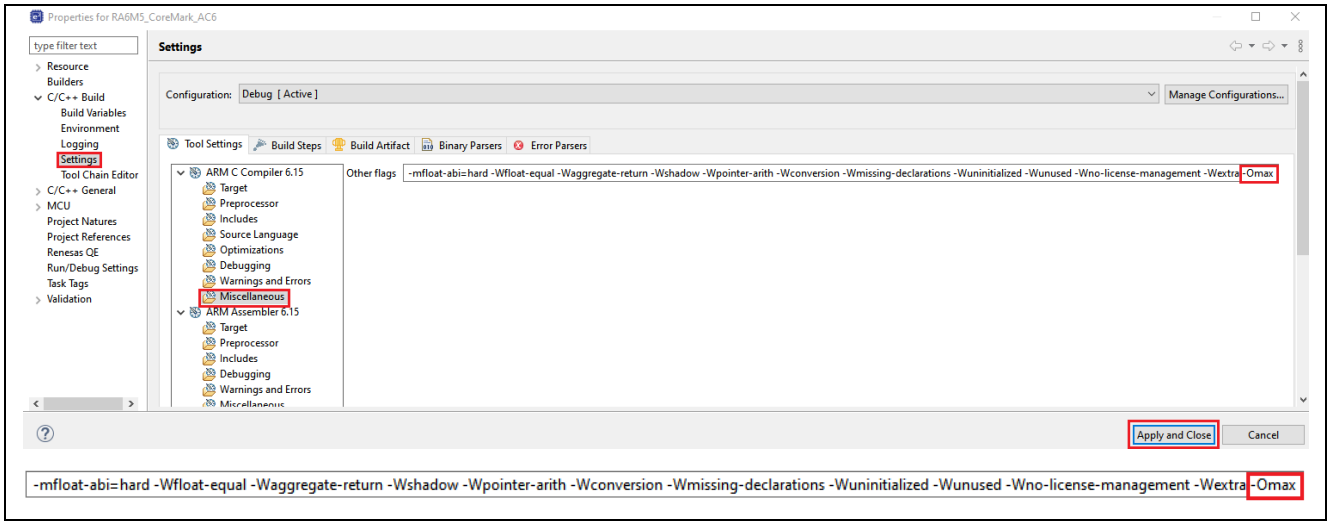
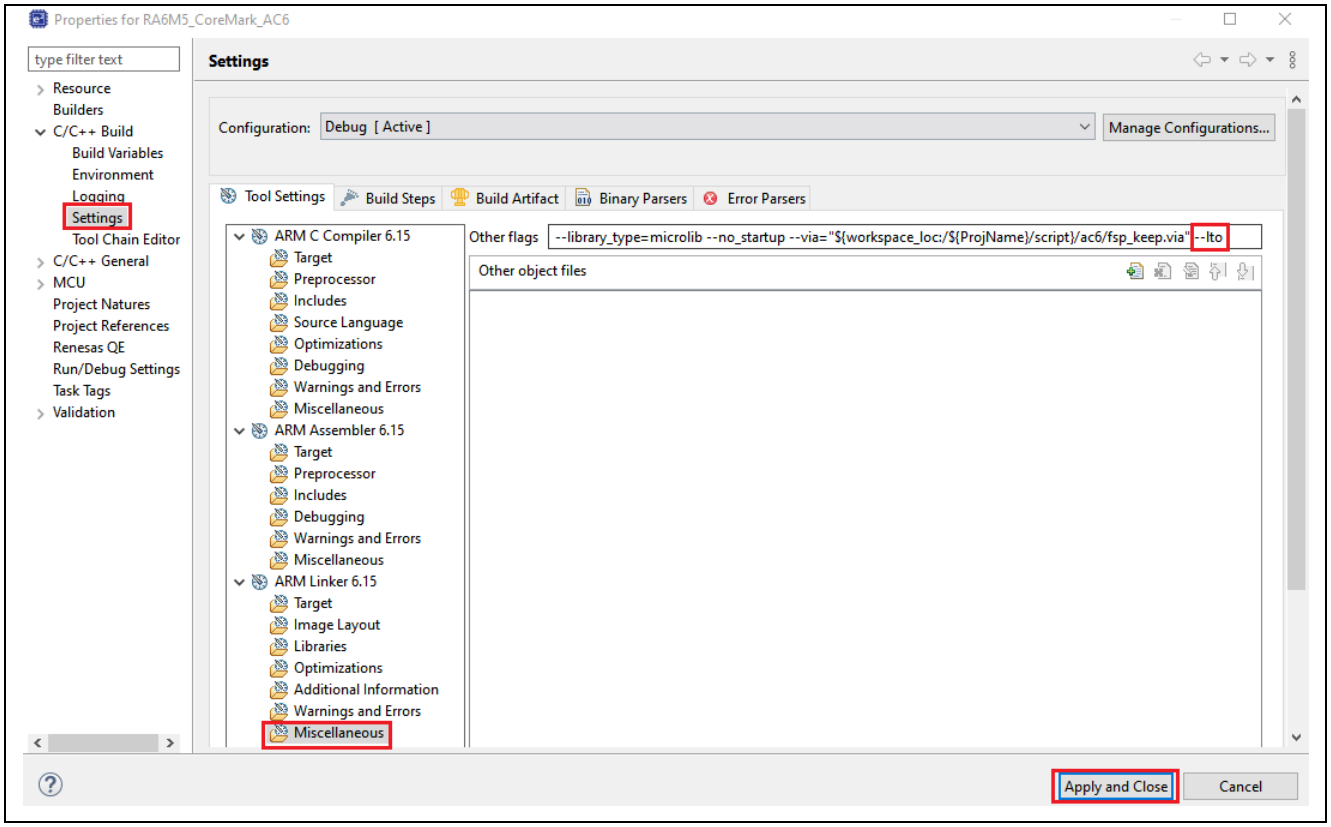


Figure 35. Add “-Omax” Option to Project Settings

In the project's Properties-> ARM Linker 6.15->Miscellaneous->Other flags, add "--lto".



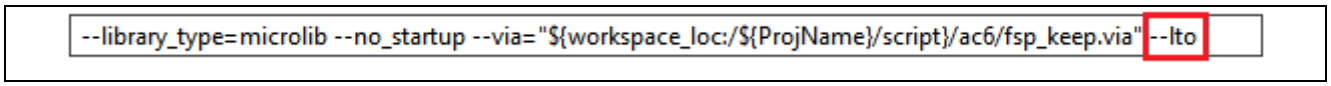


Figure 36. Add "--lto" Option to Project Settings.

2.8 Run CoreMark Project

2.8.1 Board Setup

The EK-RA6M5 kit has a few switch settings that must be configured before running the projects associated with this application note. In addition to these switch settings, the boards also contain a USB debug port and connectors to access the J-Link® programming interface.

Table 1. Switch settings for EK-RA6M5

Switch	Setting
J8	Jumper on pins 1-2
J9	Open

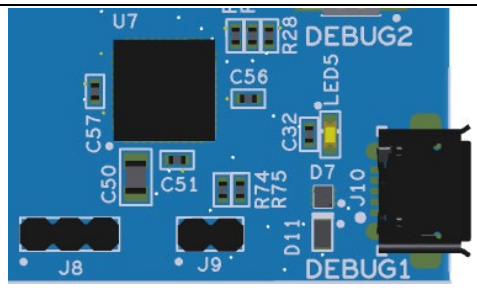


Figure 37. J8 and J9 on EK-RA6M5

The figure below shows the picture of the EK-RA6M5 kit.

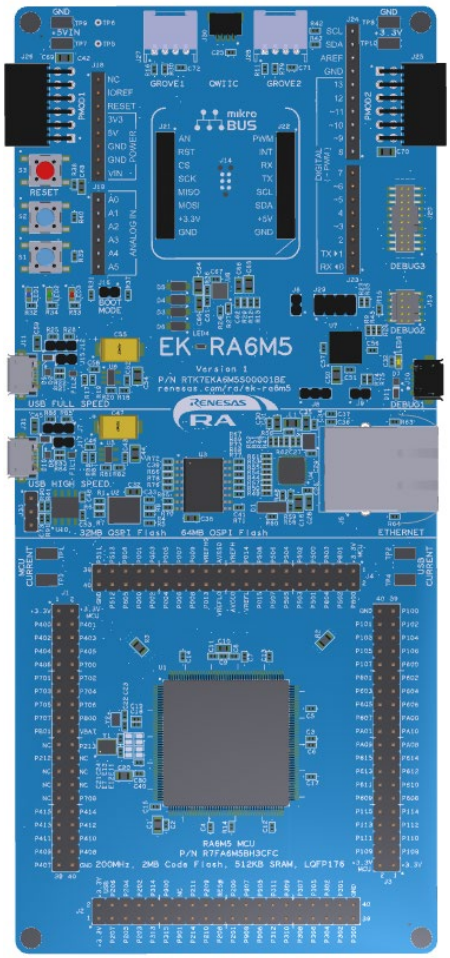


Figure 38. EK-RA6M5

Connect the board to your PC using the USB cable into the port labeled “Debug1”.

2.8.2 Add Run Commands to Print Out Benchmarking Result.

In Debug Configuration, add the below command.

dprintf portable_fini,"%s",uart_buffer

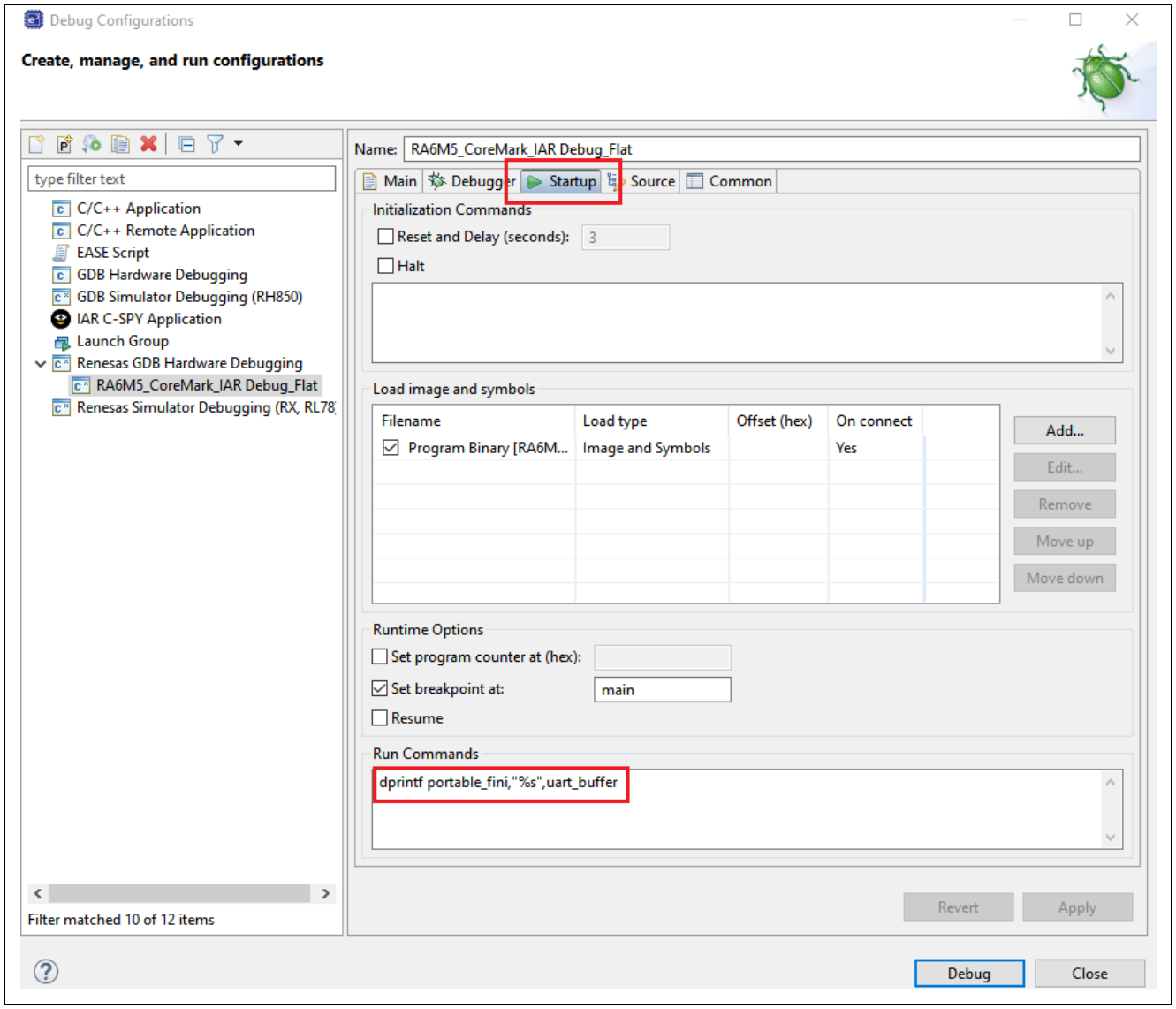


Figure 39. Add dprint Command

2.8.3 Run the e² studio Project

After successfully building the project, it can be debugged using Renesas GDB Hardware Debugging. Right-click on the project -> Debug AS -> Renesas GDB Hardware debugging or “Debug Configurations…” and choose the desired one.

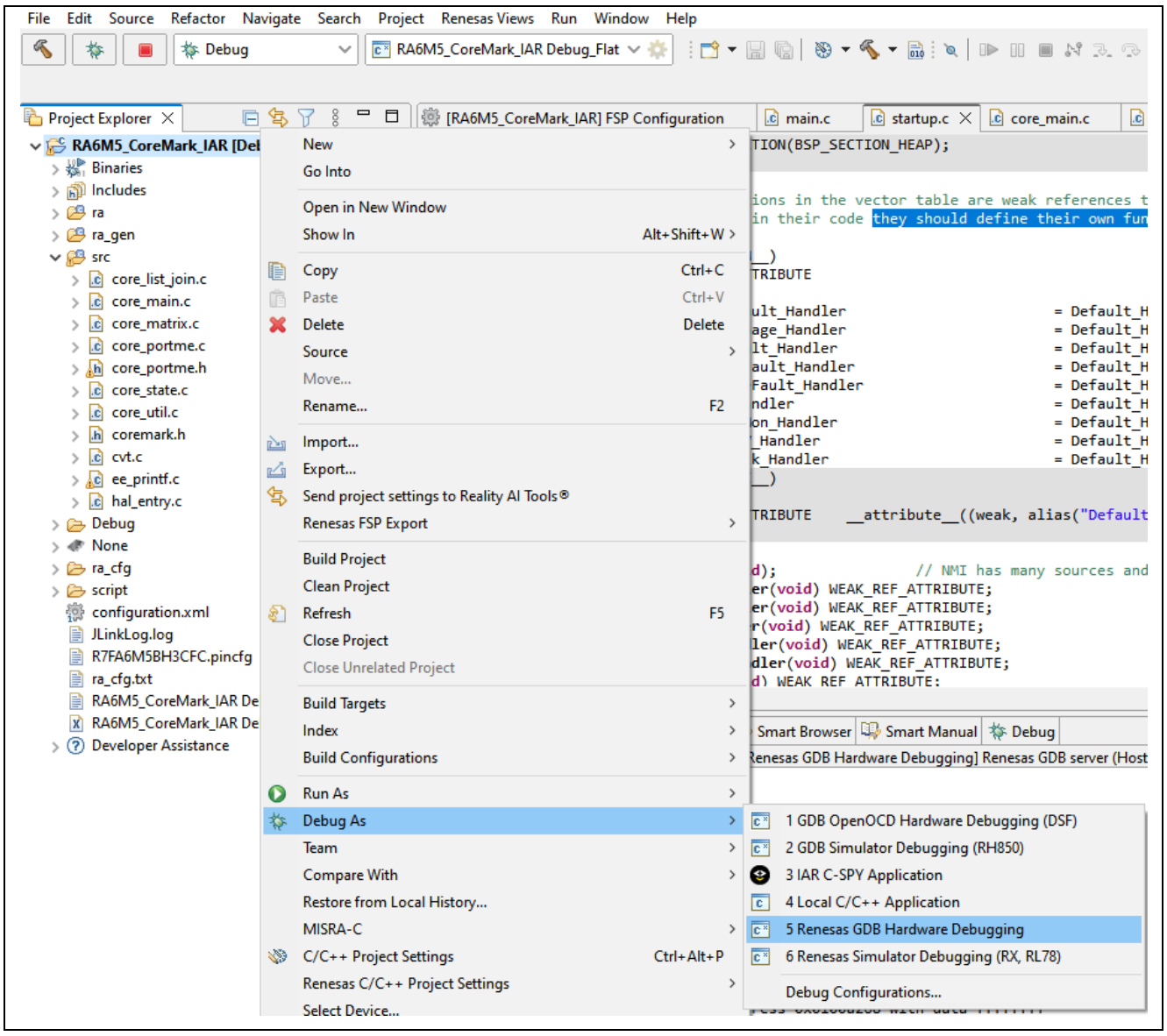


Figure 40. Debug the Project

The program should stop in the Reset_handler.

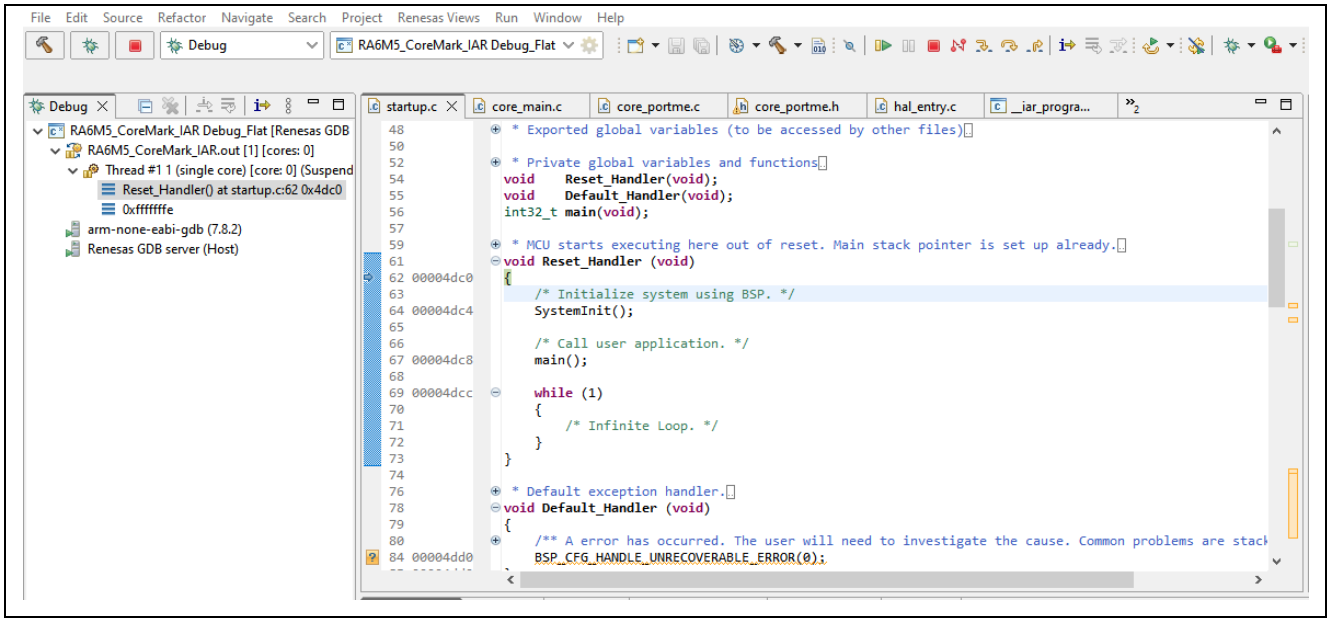


Figure 41. Debug the Project (cont'd)

Click the Resume button. The program will stop in main from core_main.c, and click the Resume button again to run the project.

After a while, the program will stop in portable_fini, and the CoreMark scores will be available in the Debugger Console window, as shown below.

```
CoreMark 1.0 : 805.740195 / IAR Compiler version 9.50.2 High Speed; No size constraints / STACK
2K performance run parameters for coremark.
CoreMark Size    : 666
Total ticks      : 558492679
Total time (secs): 11.169854
Iterations/Sec   : 805.740195
Iterations       : 9000
Compiler version : IAR Compiler version 9.50.2
Compiler flags   : High Speed; No size constraints
Memory location  : STACK
seedcrc          : 0xe9f5
[0]crclist       : 0xe714
[0]crcmatrix     : 0x1fd7
[0]crcstate      : 0x8e3a
[0]crcfinal      : 0x382f
Correct operation validated. See README.md for run and reporting rules.
CoreMark 1.0 : 805.740195 / IAR Compiler version 9.50.2 High Speed; No size constraints / STACK
```

Figure 42. CoreMark Score with IAR Compiler

```
2K performance run parameters for coremark.
CoreMark Size    : 666
Total ticks      : 580600338
Total time (secs): 11.612007
Iterations/Sec   : 775.059831
Iterations       : 9000
Compiler version : GCCClang 18.0.0
Compiler flags   : -Omax
Memory location  : STACK
seedcrc          : 0xe9f5
[0]crclist       : 0xe714
[0]crcmatrix     : 0x1fd7
[0]crcstate      : 0x8e3a
[0]crcfinal      : 0x382f
Correct operation validated. See README.md for run and reporting rules.
CoreMark 1.0 : 775.059831 / GCCClang 18.0.0 -Omax / STACK
```

Figure 43. CoreMark Score with Arm Compiler

3. Verify RA Benchmarking Results

You can verify your results by referring to RA CoreMark results published on the EEMBC website, as shown below.

Clear Sel.	Vendor	Processor	Cert.	Compiler	Execution Memory	MHz	Cores	CoreMark	CoreMark / MHz	Threads	Date↓
☐	Renesas Electronics	RA6T2	✓	ARM Clang Compile...	internal flash, intern...	240	1	962.45	4.01	1	2022-03-17
☐	Renesas Electronics	RA6T2	✓	IAR C/C++ Compiler...	internal flash, intern...	240	1	950.68	3.96	1	2022-03-17
☐	Renesas Electronics	RA2E2	✓	IAR C/C++ Compiler...	internal flash, intern...	48	1	110.24	2.29	1	2021-12-14
☐	Renesas Electronics	RA4E1	✓	IAR C/C++ Compiler...	internal flash, intern...	100	1	386.67	3.86	1	2021-09-23
☐	Renesas Electronics	RA4E1	✓	ARM Clang Compile...	internal flash, intern...	100	1	398.30	3.98	1	2021-09-23
☐	Renesas Electronics	RA6E1	✓	IAR C/C++ Compiler...	internal flash, intern...	200	1	770.75	3.85	1	2021-09-23
☐	Renesas Electronics	RA6E1	✓	ARM Clang Compile...	internal flash, intern...	200	1	790.27	3.95	1	2021-09-23
☐	Renesas Electronics	RA6M5	✓	IAR C/C++ Compiler...	internal flash, intern...	200	1	770.82	3.85	1	2021-04-26
☐	Renesas Electronics	RA6M5	✓	ARM Clang Compile...	internal flash, intern...	200	1	790.76	3.95	1	2021-04-26
☐	Renesas Electronics	RA4M2	✓	ARM Clang Compile...	internal flash, intern...	100	1	398.30	3.98	1	2021-04-26
☐	Renesas Electronics	RA4M2	✓	IAR C/C++ Compiler...	internal flash, intern...	100	1	386.00	3.86	1	2021-04-26
☐	Renesas Electronics	RA2E1	✓	IAR C/C++ Compiler...	internal flash, intern...	48	1	111.73	2.32	1	2021-04-26
☐	Renesas Electronics	RA6T1	✓	IAR C/C++ Compiler...	internal flash, intern...	120	1	405.90	3.38	1	2021-03-10
☐	Renesas Electronics	RA6M4	✓	ARM Clang Compile...	internal flash, intern...	200	1	790.75	3.95	1	2021-03-10
☐	Renesas Electronics	RA6M4	✓	IAR C/C++ Compiler...	internal flash, intern...	200	1	770.52	3.85	1	2021-03-10
☐	Renesas Electronics	RA4M3	✓	ARM Clang Compile...	internal flash, intern...	100	1	397.30	3.97	1	2021-03-10
☐	Renesas Electronics	RA4M3	✓	IAR C/C++ Compiler...	internal flash, intern...	100	1	386.11	3.86	1	2021-03-10
☐	Renesas Electronics	RA2L1	✓	IAR C/C++ Compiler...	internal flash, intern...	48	1	111.73	2.32	1	2021-03-10
☐	Broadcom Corporation	Broadcom BCM283...		GCC 7.2.1	LPDDR2 900MHz	1200	4	15363.93	12.80	4	2018-01-06

Figure 44. RA Coremark scores published on the EEMBC website

4. General Guidelines for CoreMark Benchmarking

Since target devices that contain Arm processors may have a wide variety of memories and memory hierarchies, your CoreMark project should be compiled using memory correctly and efficiently. Depending on the compiler, you can achieve this by correctly editing your linker script or scatter files.

Since CoreMark is a small benchmark, it should be run multiple times to obtain reproducible numbers.

Arm recommends performing two validation runs followed by at least ten profile runs. The results can be calculated by the average for the profile runs. These steps are necessary to minimize the variation caused by inconsistent processor states.

5. References

EEMBC's CoreMark® https://www.eembc.org/coremark/

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	March.20.23	-	Initial version
1.10	Sep.11.24	-	Update to FSP v5.5.0

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/.