Renesas RA Family

Getting Started with CoreMark Benchmarking

Introduction

As processors in embedded systems become more complex, more sophisticated benchmarks are needed for a better understanding of 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 also so 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 tools and software

- e^2 studio v2023-01
- Renesas Flexible Software Package (FSP) v4.3.0
- Arm Compiler 6.15
- IAR Embedded Workbench v9.32.1

Hardware

- Renesas RA kit: EK-RA6M5

Reference Manuals

- RA Flexible Software Package Documentation Release v4.3.0
- 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 e2 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 result.

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 e2 studio IDE together with FSP. You can download and install setup_fsp_v4_3_0_e2s_v2023-01.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 Arm compiler in Keil MDK installation for CoreMark benchmarking.

2.1 Integrating Toolchains with e2 studio

2.1.1 IAR Embedded Workbench Plugin

Download and install IAR Embedded workbench before you integrate IAR compiler with e2 studio.

Start e2 studio, then select “Help -> IAR Embedded Workbench plugin manager”

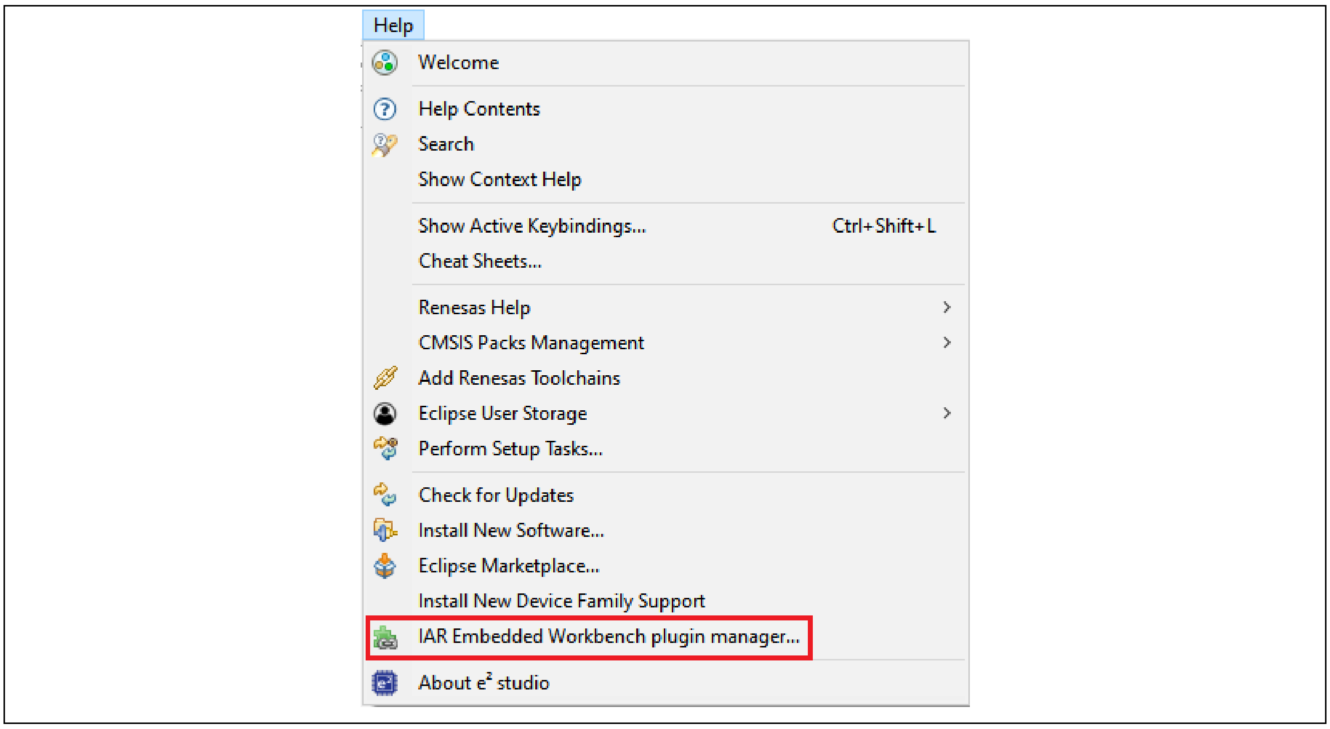
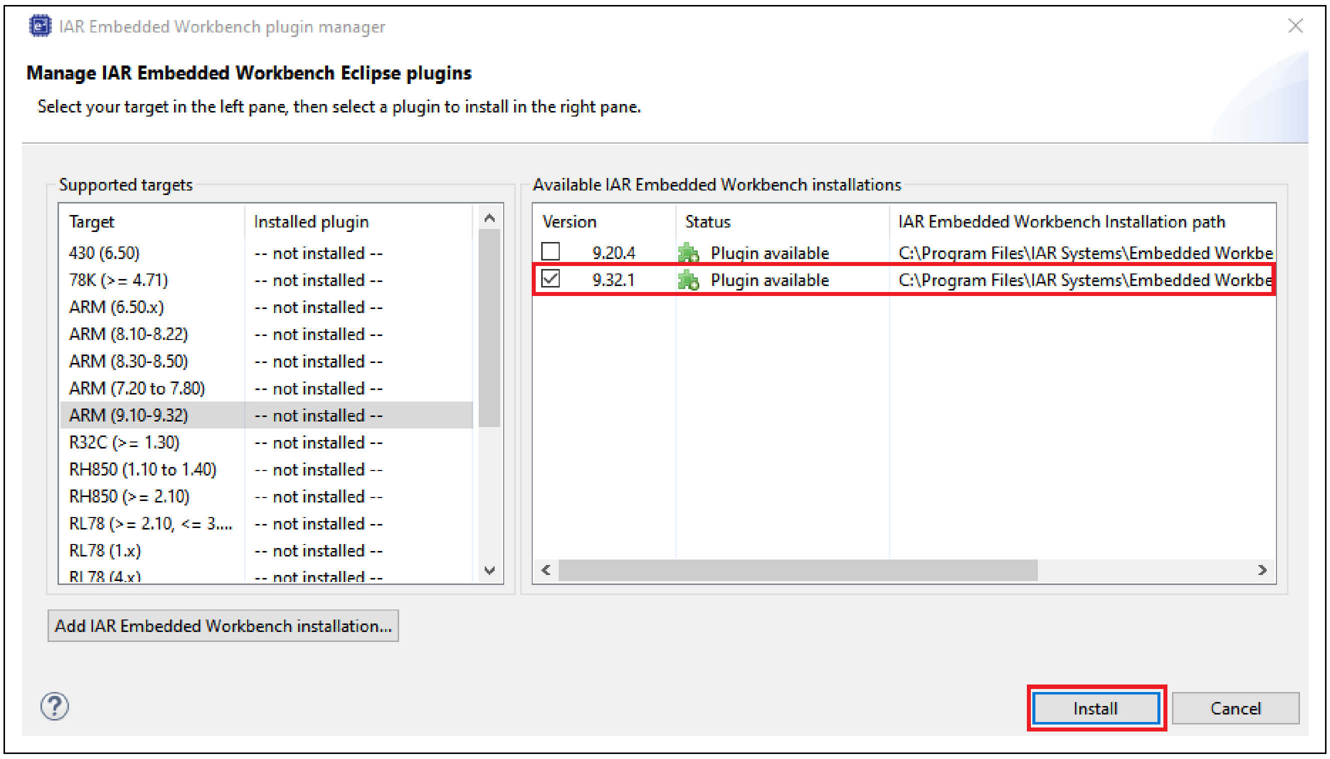


Figure 1. Select IAR Embedded Workbench Plugin Manager

You choose the desired toolchain and press install.


IAR Embedded Workbench plugin manager

Manage IAR Embedded Workbench Eclipse plugins

Select your target in the left pane, then select a plugin to install in the right pane.

Supported targets

Target	Installed plugin
430 (6.50)	-- not installed --
78K (>= 4.71)	-- not installed --
ARM (6.50.x)	-- not installed --
ARM (8.10-8.22)	-- not installed --
ARM (8.30-8.50)	-- not installed --
ARM (7.20 to 7.80)	-- not installed --
ARM (9.10-9.32)	-- not installed --
R32C (>= 1.30)	-- not installed --
RH850 (1.10 to 1.40)	-- not installed --
RH850 (>= 2.10)	-- not installed --
RL78 (>= 2.10, <= 3....	-- not installed --
RL78 (1.x)	-- not installed --
RL78 (4.x)	-- not installed --

Available IAR Embedded Workbench installations

Version	Status	IAR Embedded Workbench Installation path
☐ 9.20.4	Plugin available	C:\Program Files\IAR Systems\Embedded Workbe
☑ 9.32.1	Plugin available	C:\Program Files\IAR Systems\Embedded Workbe

Figure 2. Select IAR Plugin

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

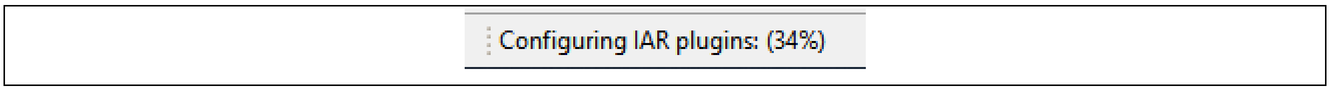
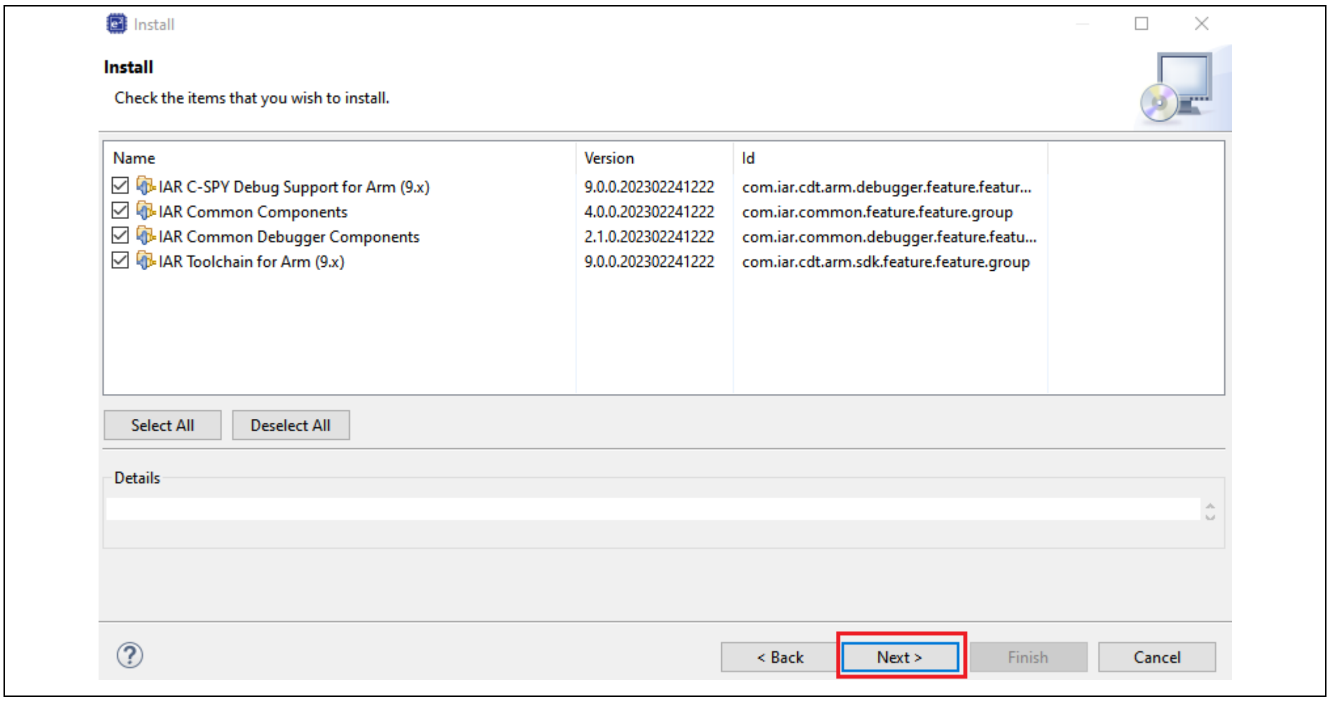

Configuring IAR plugins: (34%)

Figure 3. IAR Configuration Progress

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



Name	Version	Id
☑ IAR C-SPY Debug Support for Arm (9.x)	9.0.0.202302241222	com.iar.cdt.arm.debugger.feature.featur...
☑ IAR Common Components	4.0.0.202302241222	com.iar.common.feature.feature.group
☑ IAR Common Debugger Components	2.1.0.202302241222	com.iar.common.debugger.feature.featu...
☑ IAR Toolchain for Arm (9.x)	9.0.0.202302241222	com.iar.cdt.arm.sdk.feature.feature.group

Figure 4. Install IAR Embedded Workbench Plugin

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

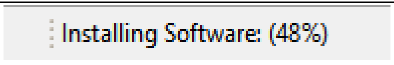
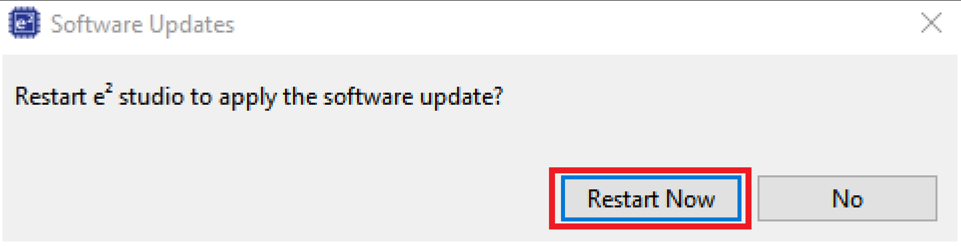


Figure 5. IAR Plugin Installation Process

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



Restart e² studio to apply the software update?

Figure 6. Restart e2 studio

2.1.2 Integrate with Arm Compiler

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

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

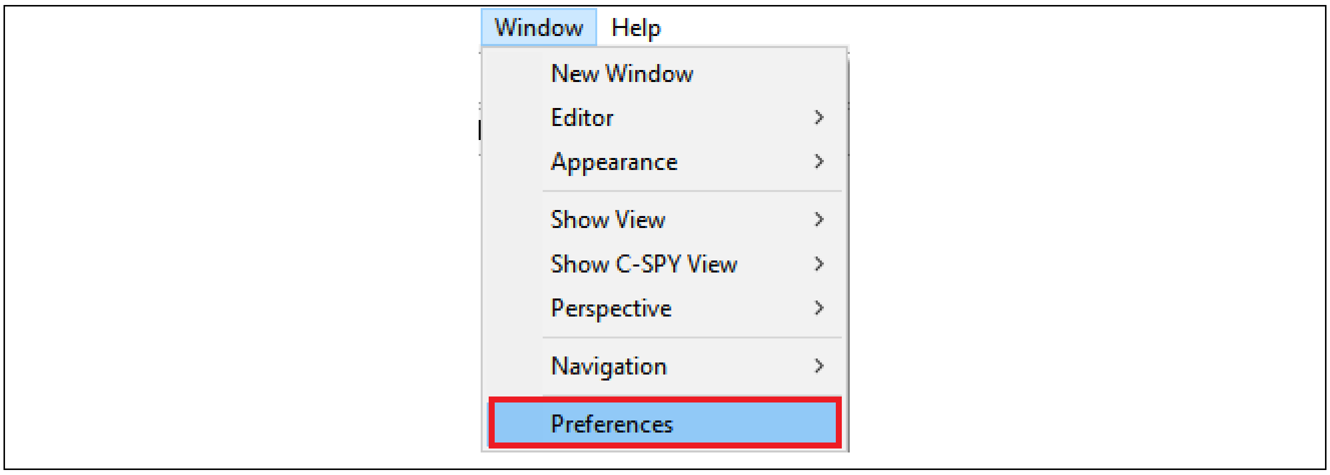
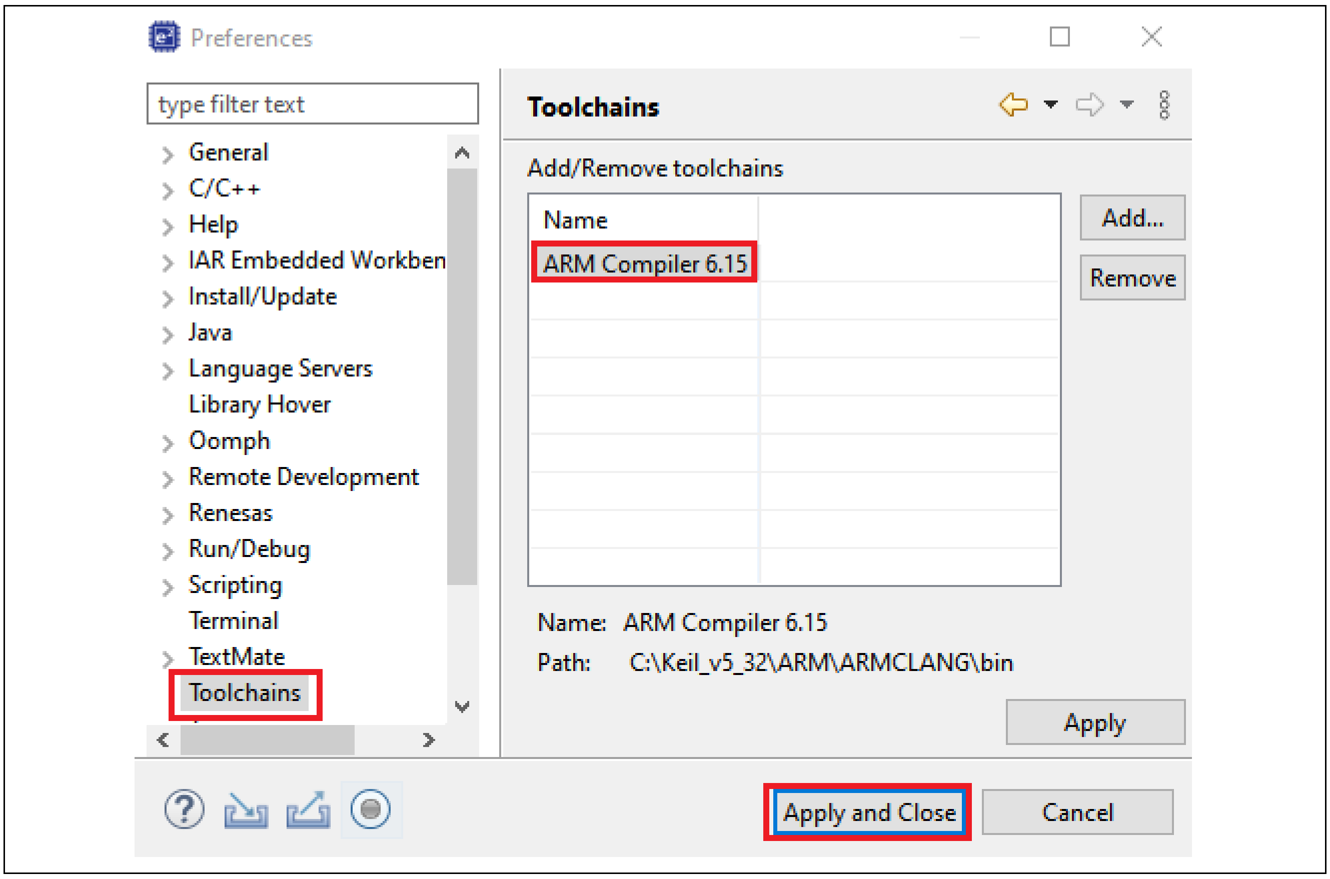


Figure 7. e2 studio Preferences

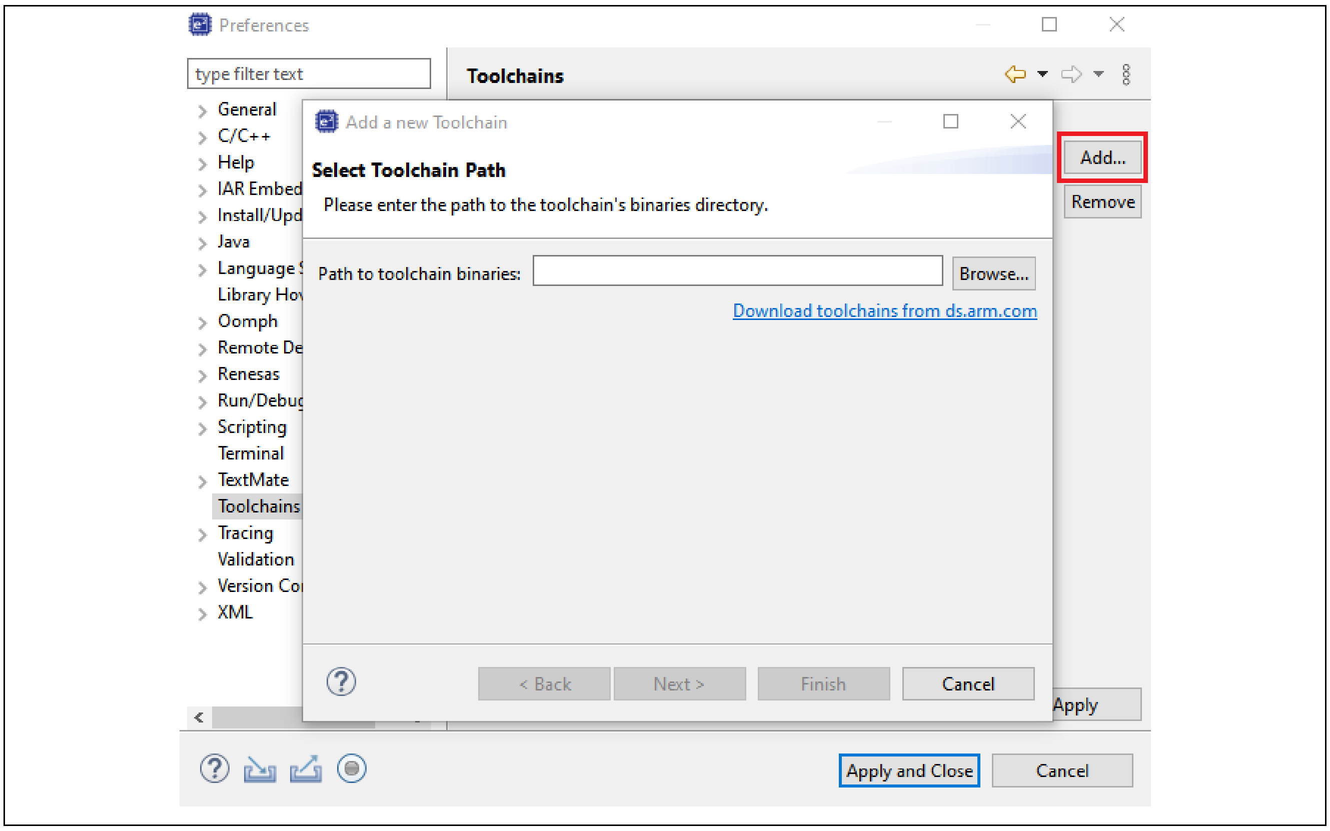
Select the desired Arm Compiler toolchain, then click “Apply and Close” to add the Arm compiler to e2 studio.



Name: ARM Compiler 6.15
Path: C:\Keil_v5_32\ARM\ARMCLANG\bin

Figure 8. Add Arm Compiler to e2 studio

If 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



Select Toolchain Path

Please enter the path to the toolchain's binaries directory.

Path to toolchain binaries: [ ] Browse...

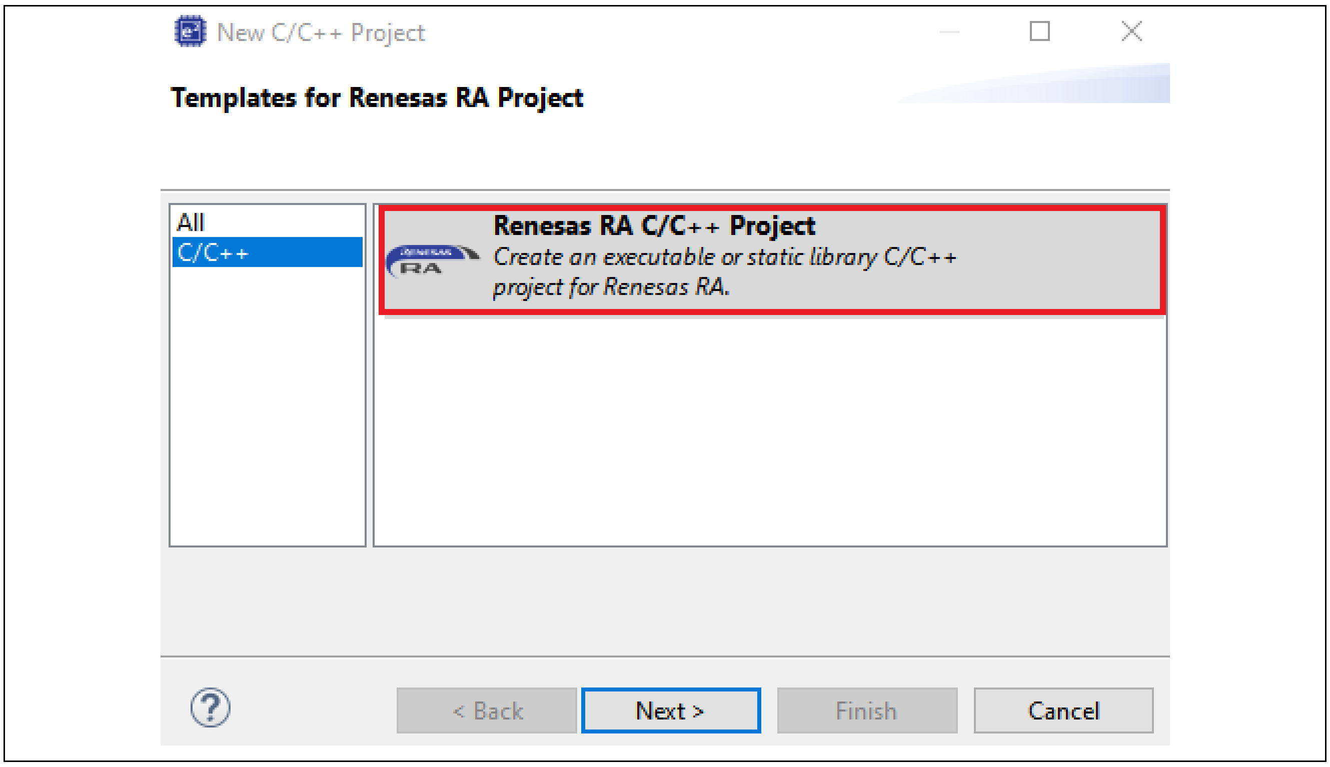
Download toolchains from ds.arm.com

Figure 9. Add Toolchain's Path

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

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

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

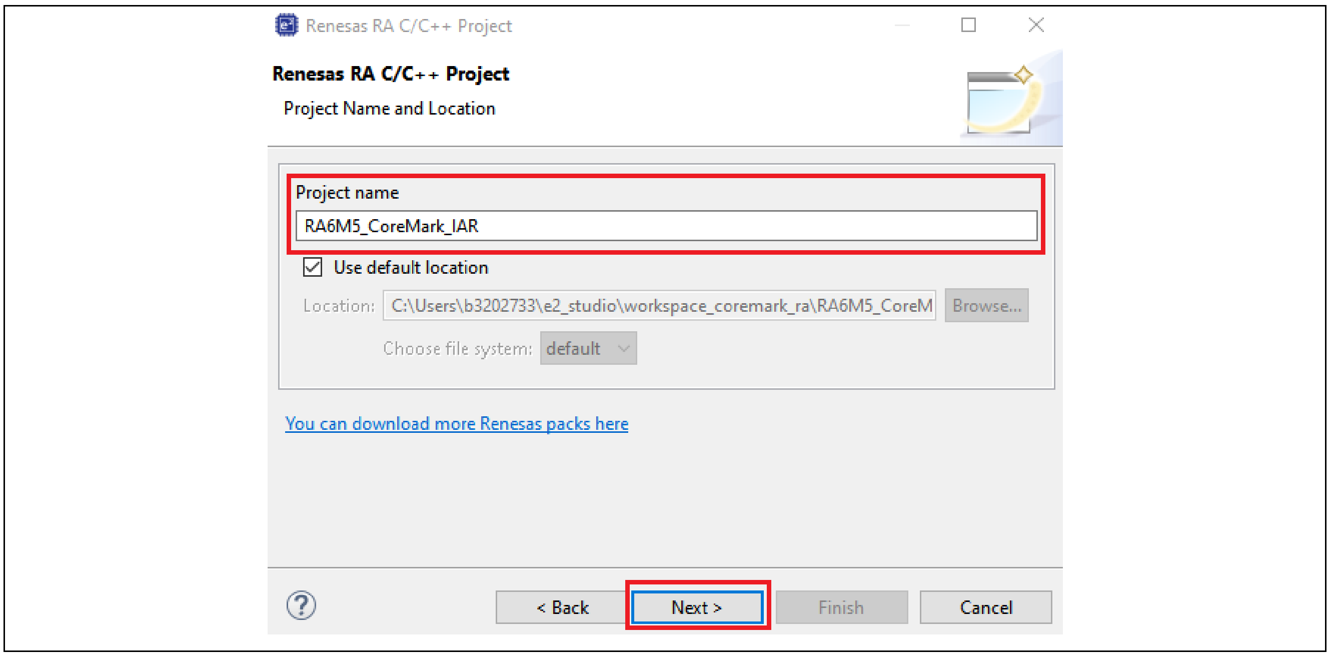


Templates for Renesas RA Project

Renesas RA C/C++ Project
Create an executable or static library C/C++ project for Renesas RA.

Figure 10. Select C/C++ Template

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



Project name: RA6M5_CoreMark_IAR

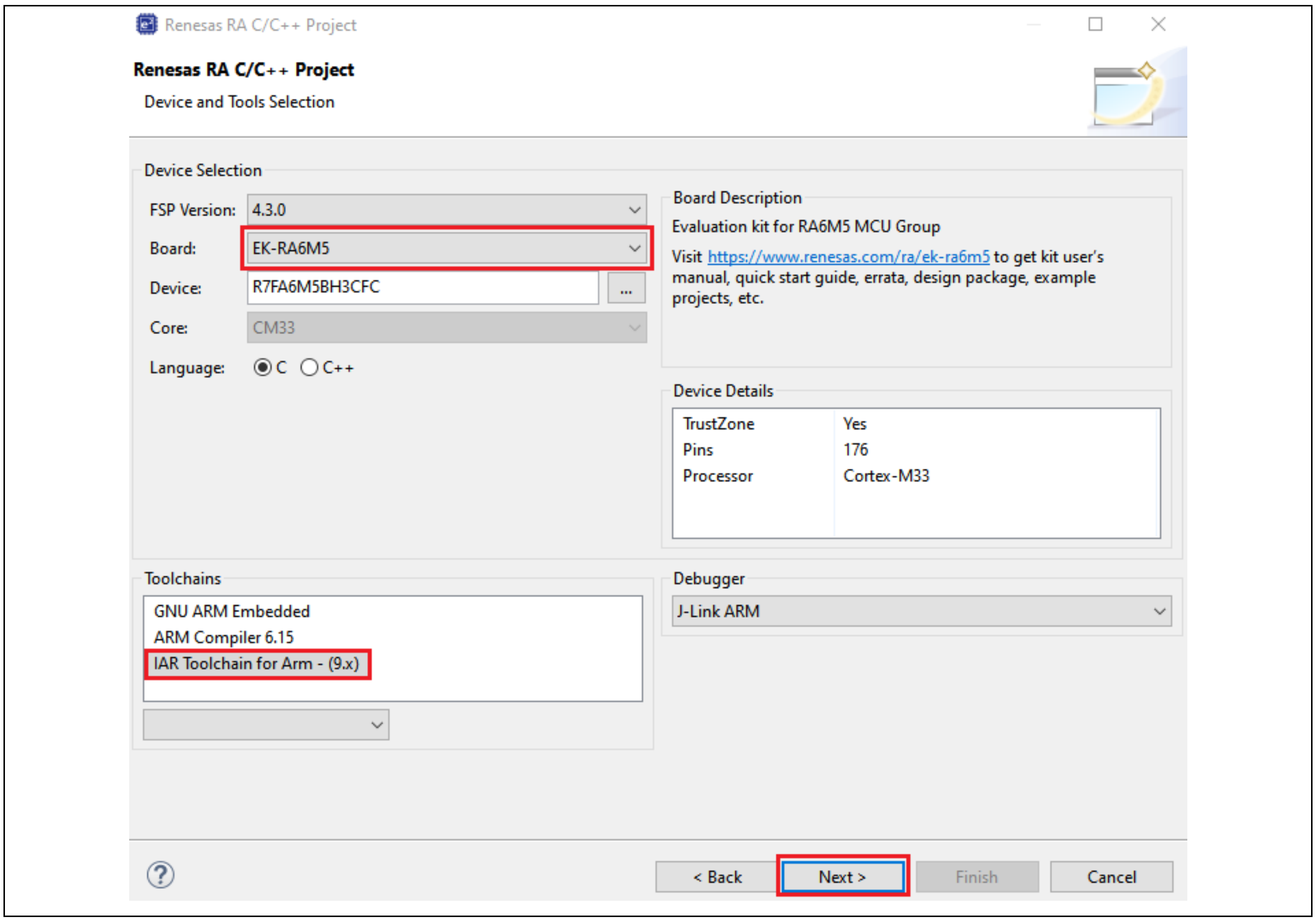
☑ Use default location

Location: C:\Users\b3202733\e2_studio\workspace_coremark_ra\RA6M5_CoreM

Choose file system: default

Figure 11. Name Your Project

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



FSP Version: 4.3.0

Board: EK-RA6M5

Device: R7FA6M5BH3CFC

Core: CM33

Language: [✓ C] C++

Board Description

Evaluation kit for RA6M5 MCU Group

Visit https://www.renesas.com/ra/ek-ra6m5 to get kit user's manual, quick start guide, errata, design package, example projects, etc.

Device Details

TrustZone	Yes
Pins	176
Processor	Cortex-M33

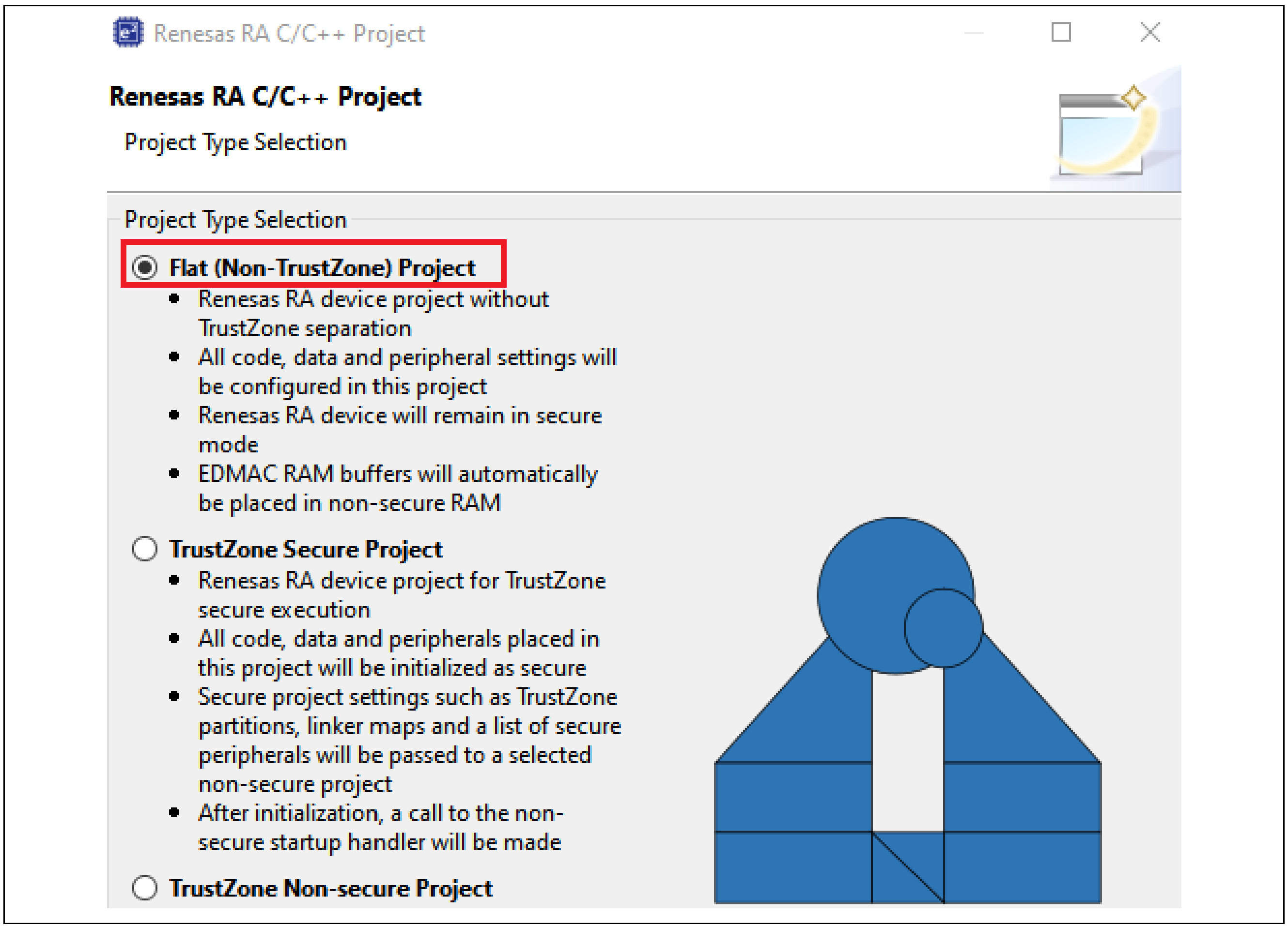
Toolchains

- GNU ARM Embedded
- ARM Compiler 6.15
- IAR Toolchain for Arm - (9.x)

Debugger: J-Link ARM

Figure 12. RA Project Options

Select Flat (Non-TrustZone) Project.



Renesas RA C/C++ Project

Renesas RA C/C++ Project

Project Type Selection

Project Type Selection

[✓ Flat (Non-TrustZone) Project]

- Renesas RA device project without TrustZone separation
- All code, data and peripheral settings will be configured in this project
- Renesas RA device will remain in secure mode
- EDMAC RAM buffers will automatically be placed in non-secure RAM

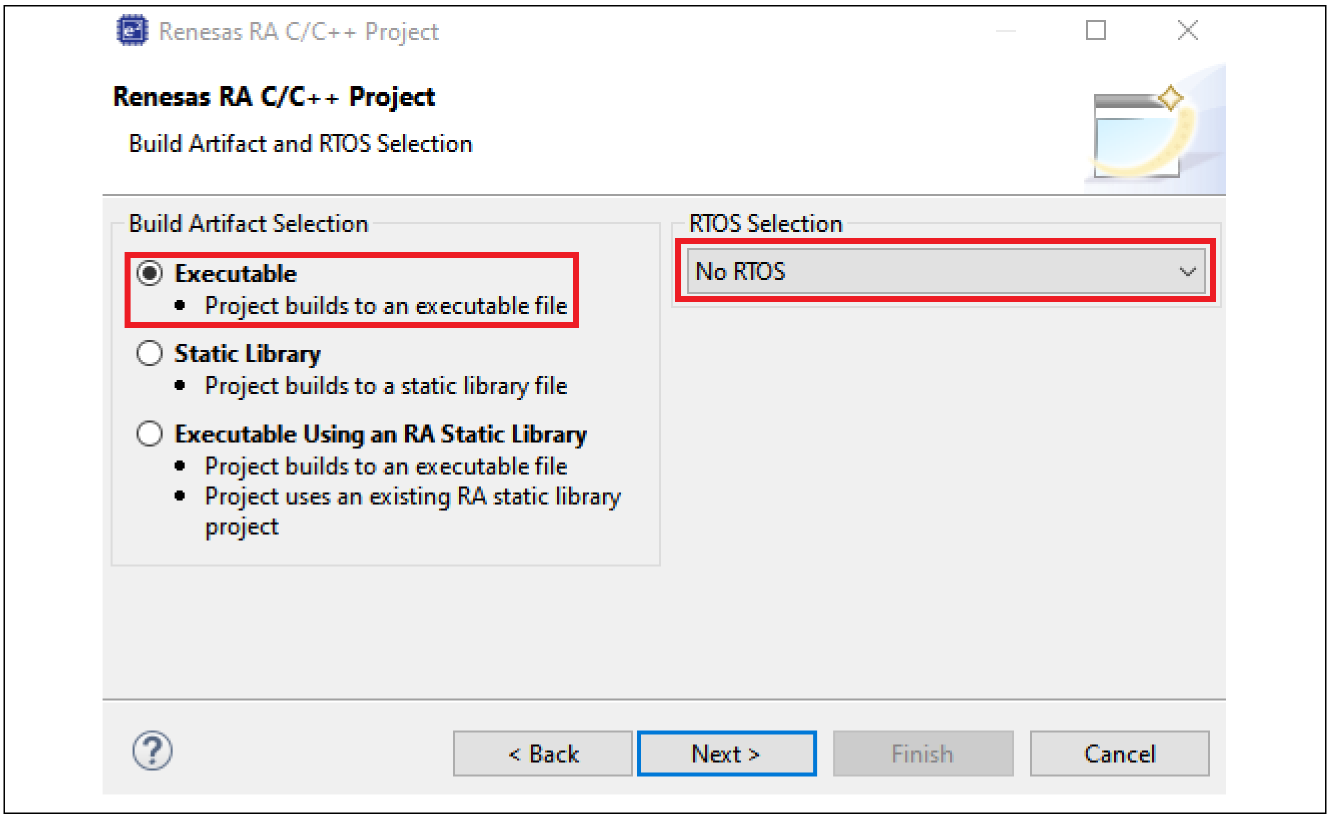
TrustZone Secure Project

- Renesas RA device project for TrustZone secure execution
- All code, data and peripherals placed in this project will be initialized as secure
- Secure project settings such as TrustZone partitions, linker maps and a list of secure peripherals will be passed to a selected non-secure project
- After initialization, a call to the non-secure startup handler will be made

TrustZone Non-secure Project

Figure 13. Flat Project Selection

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



Renesas RA C/C++ Project

Renesas RA C/C++ Project

Build Artifact and RTOS Selection

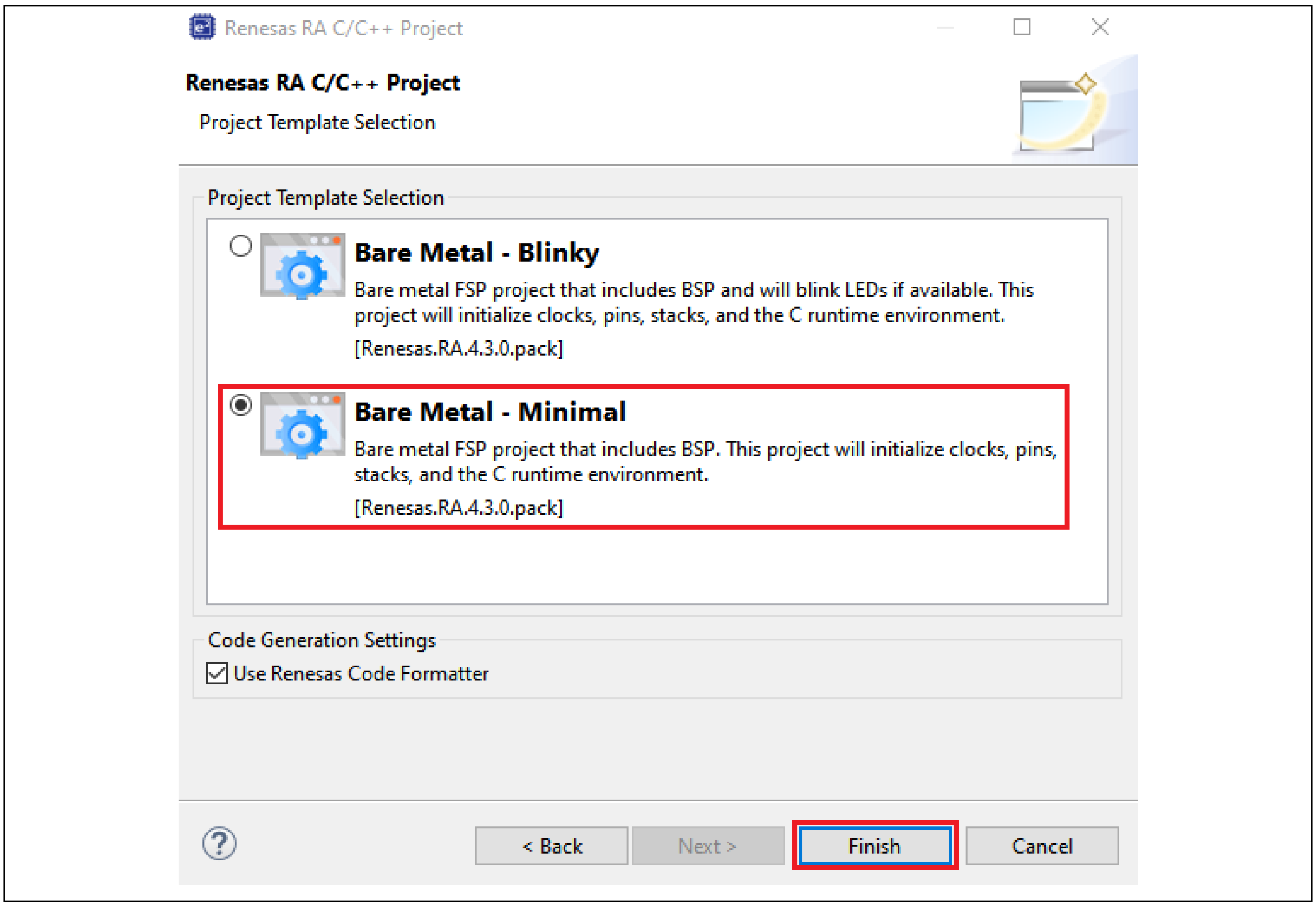
Build Artifact Selection

- [✓ Executable]
  - Project builds to an executable file
- Static Library
  - Project builds to a static library file
- Executable Using an RA Static Library
  - Project builds to an executable file
  - Project uses an existing RA static library project

RTOS Selection: No RTOS

Figure 14. Select No RTOS Project

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



Renesas RA C/C++ Project

Renesas RA C/C++ Project

Project Template Selection

Project Template Selection

- Bare Metal - Blinky: Bare metal FSP project that includes BSP and will blink LEDs if available. This project will initialize clocks, pins, stacks, and the C runtime environment. [Renesas.RA.4.3.0.pack]
- [✓ Bare Metal - Minimal]: Bare metal FSP project that includes BSP. This project will initialize clocks, pins, stacks, and the C runtime environment. [Renesas.RA.4.3.0.pack]

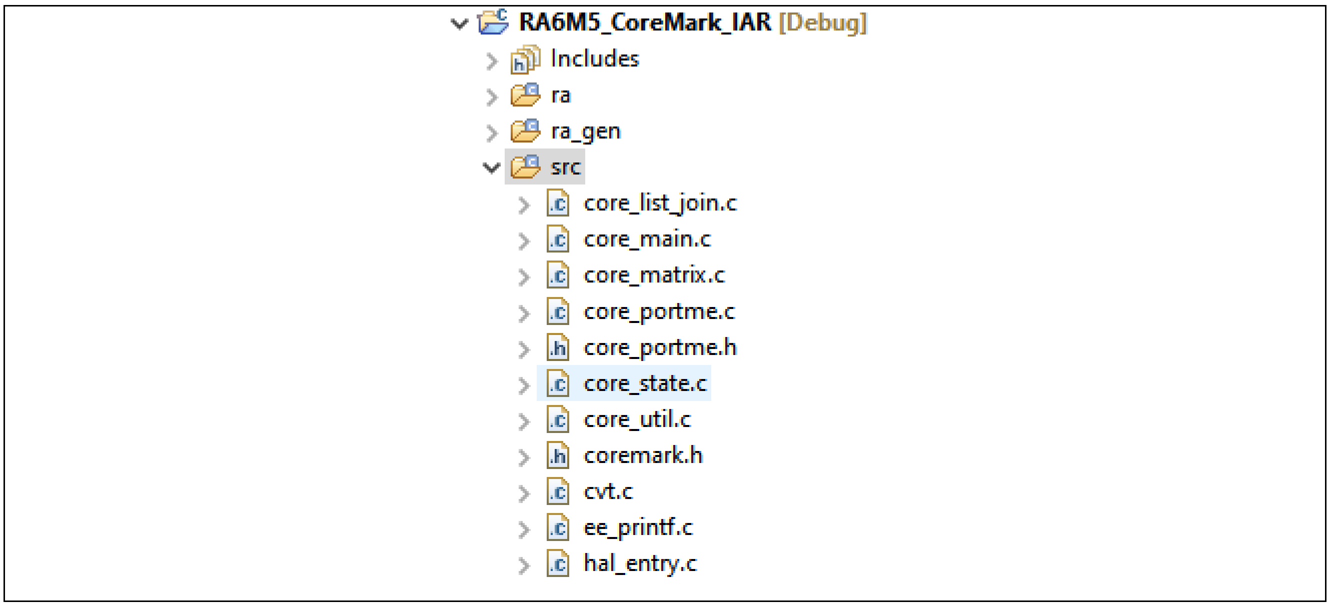
Code Generation Settings

☑ Use Renesas Code Formatter

Figure 15. Bare Metal Minimal Option

2.3 Add CoreMark to e2 studio Project

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



RA6M5_CoreMark_IAR [Debug]
- Includes
- ra
- ra_gen
- src
  - core_list_join.c
  - core_main.c
  - core_matrix.c
  - core_portme.c
  - core_portme.h
  - core_state.c
  - core_util.c
  - coremark.h
  - cvt.c
  - ee_printf.c
  - hal_entry.c

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.

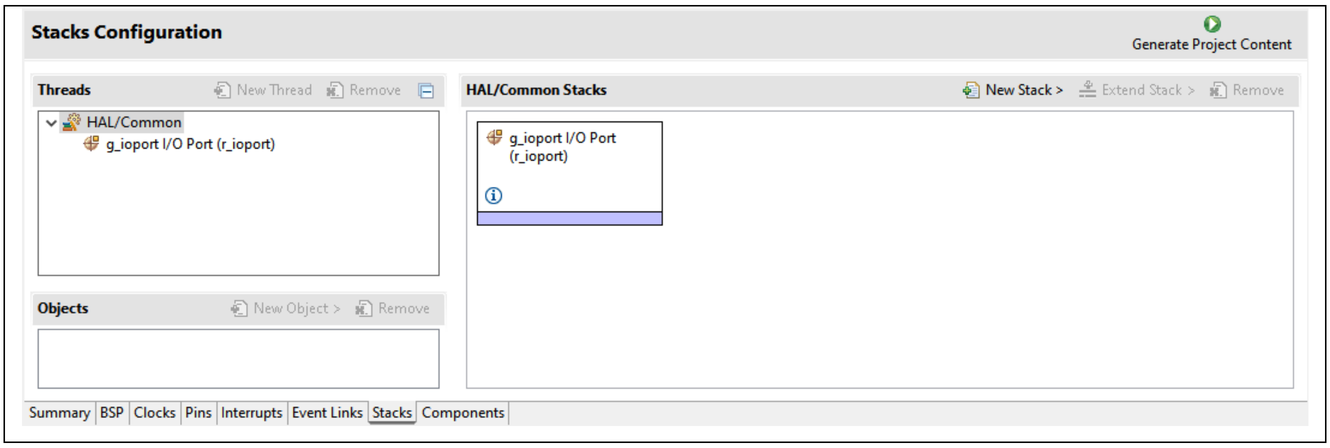


Figure 17. Stack Configuration

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).

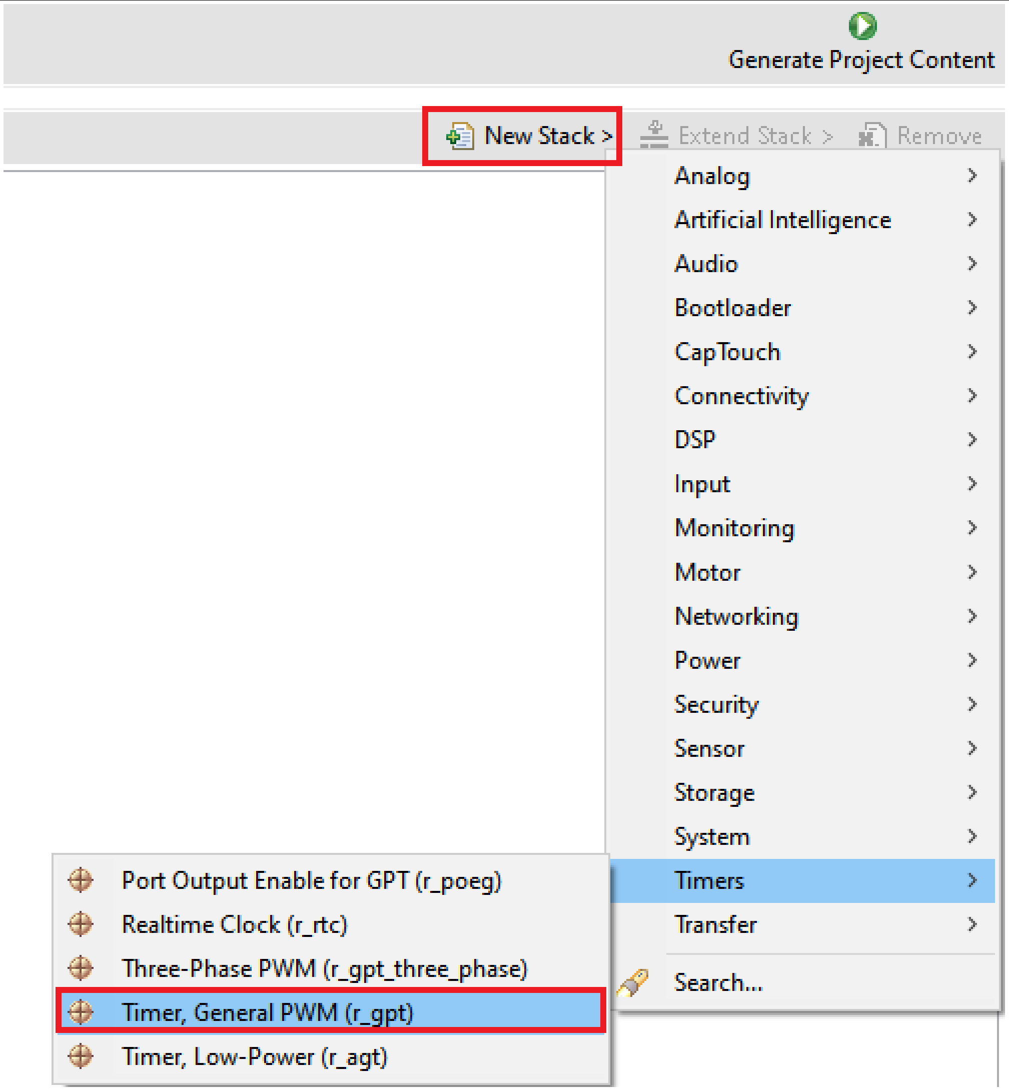
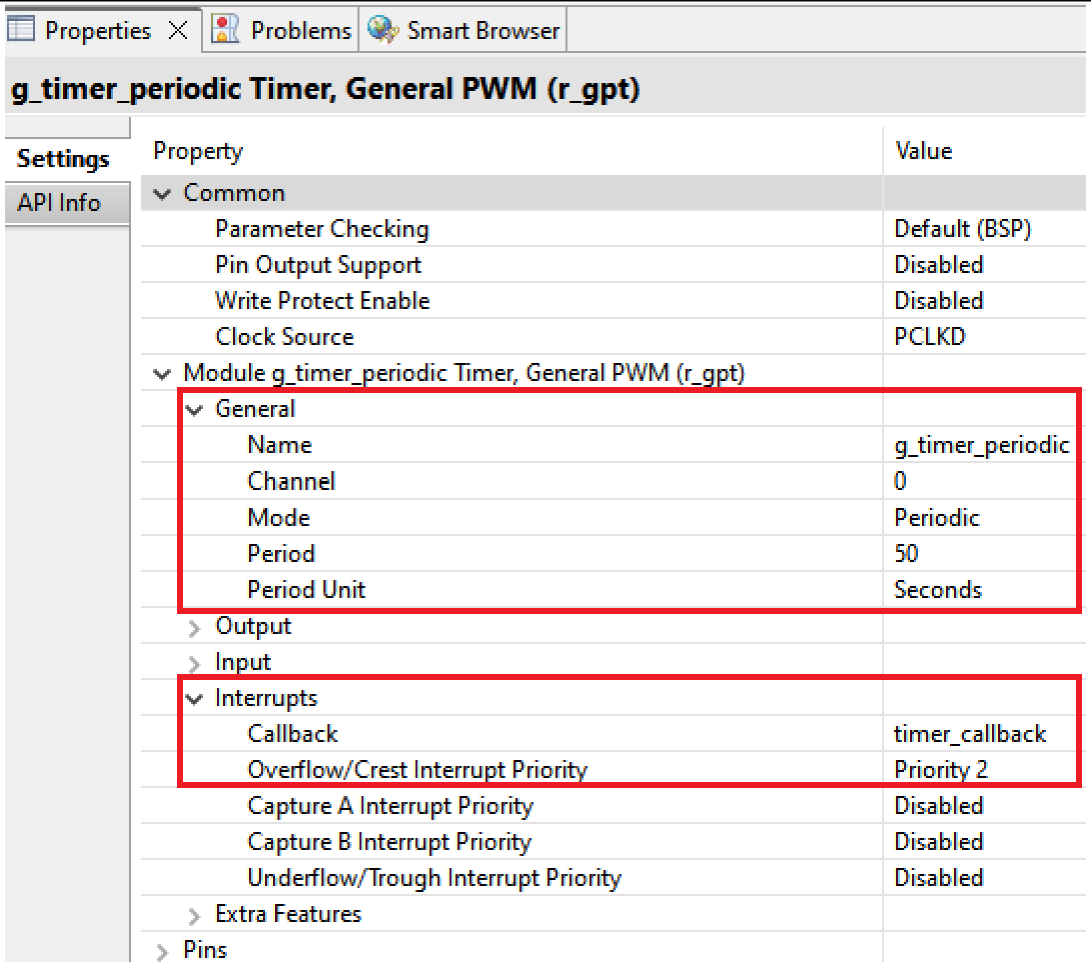


Figure 18. Add GPT Timer

After adding the GPT timer, you need to edit the settings. Clicking on the block representing the newly added GPT timer, then go to 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.


Properties | Problems | Smart Browser

g_timer_periodic Timer, General PWM (r_gpt)

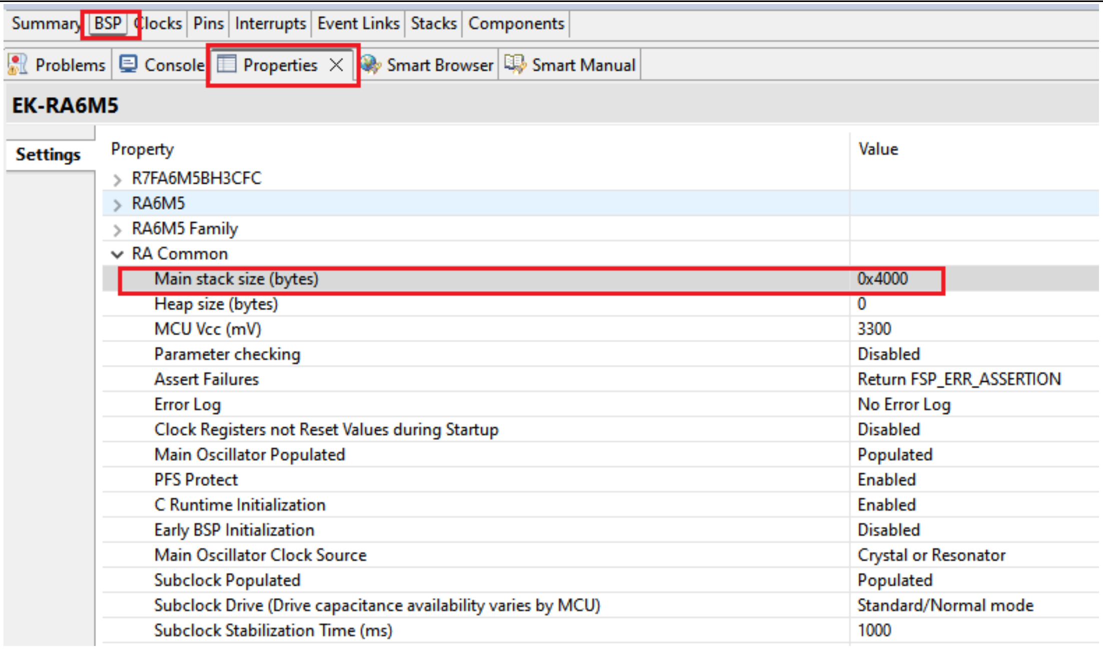
Settings | API Info

Property	Value
Common	
Parameter Checking	Default (BSP)
Pin Output Support	Disabled
Write Protect Enable	Disabled
Clock Source	PCLKD
Module g_timer_periodic Timer, General PWM (r_gpt)	
General	
Name	g_timer_periodic
Channel	0
Mode	Periodic
Period	50
Period Unit	Seconds
Output	
Input	
Interrupts	
Callback	timer_callback
Overflow/Crest Interrupt Priority	Priority 2
Capture A Interrupt Priority	Disabled
Capture B Interrupt Priority	Disabled
Underflow/Trough Interrupt Priority	Disabled
Extra Features	
Pins	

Figure 19. GPT Configuration

2.5 Update Main Stack

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


Summary | BSP | Clocks | Pins | Interrupts | Event Links | Stacks | Components

Problems | Console | Properties | Smart Browser | Smart Manual

EK-RA6M5

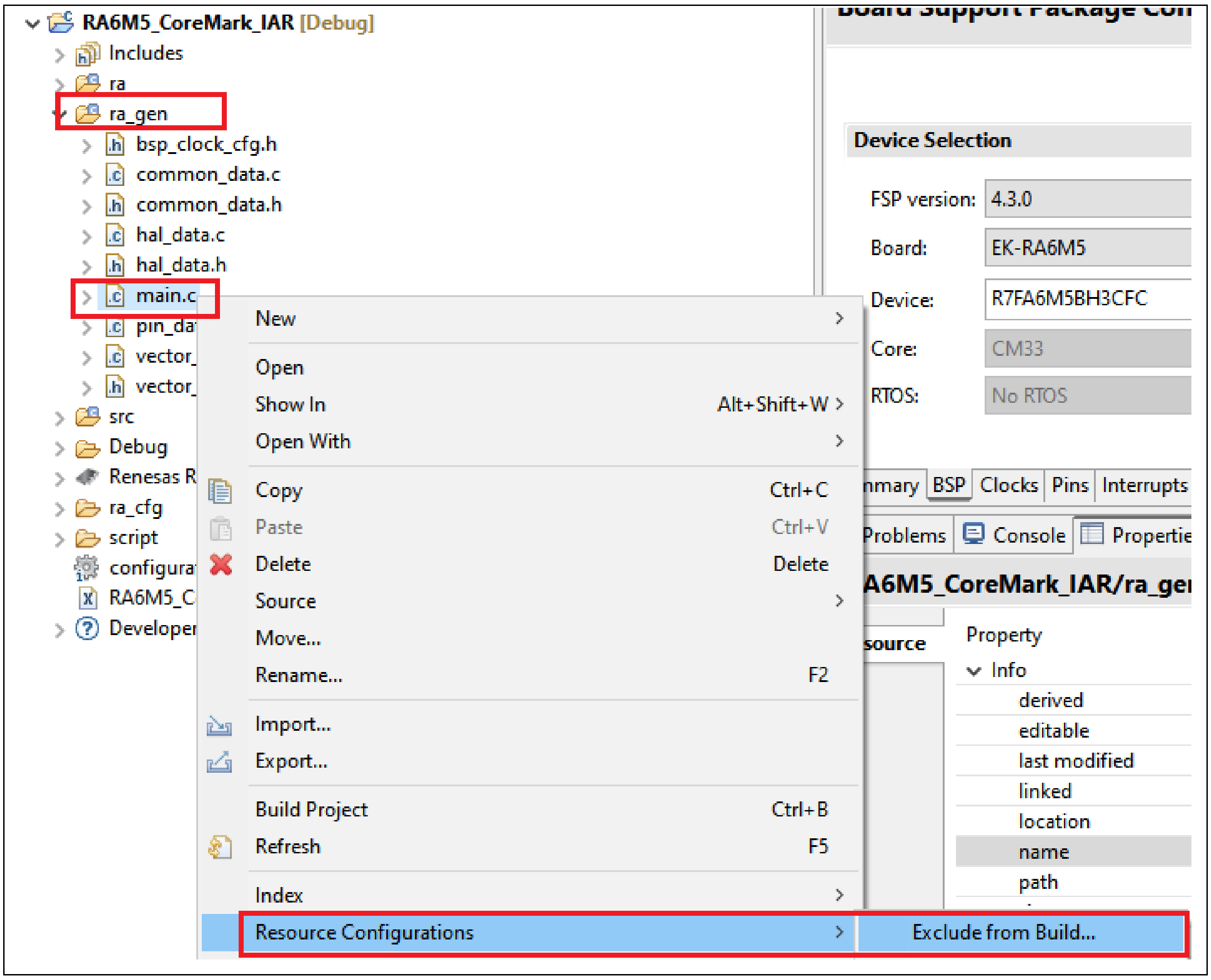
Settings

Property	Value
R7FA6M5BH3CFC	
RA6M5	
RA6M5 Family	
RA Common	
Main stack size (bytes)	0x4000
Heap size (bytes)	0
MCU Vcc (mV)	3300
Parameter checking	Disabled
Assert Failures	Return FSP_ERR_ASSERTION
Error Log	No Error Log
Clock Registers not Reset Values during Startup	Disabled
Main Oscillator Populated	Populated
PFS Protect	Enabled
C Runtime Initialization	Enabled
Early BSP Initialization	Disabled
Main Oscillator Clock Source	Crystal or Resonator
Subclock Populated	Populated
Subclock Drive (Drive capacitance availability varies by MCU)	Standard/Normal mode
Subclock Stabilization Time (ms)	1000

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.



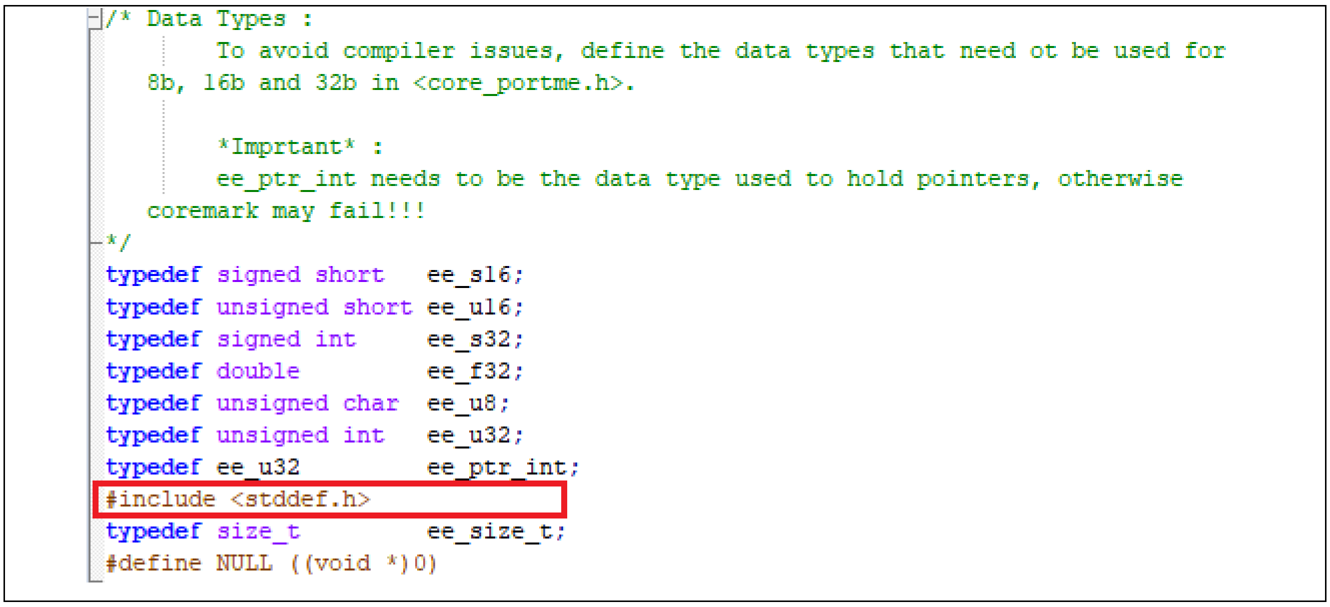
FSP version: 4.3.0
Board: EK-RA6M5
Device: R7FA6M5BH3CFC
Core: CM33
RTOS: No RTOS

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.

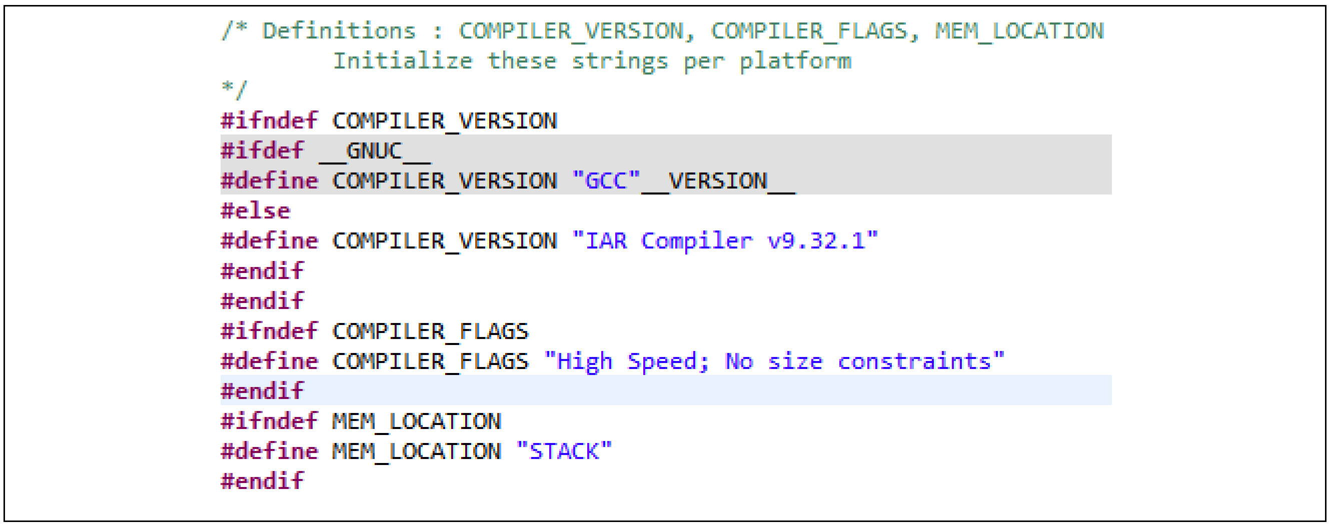


```
/* Data Types :
        To avoid compiler issues, define the data types that need ot be used for
    8b, 16b and 32b in <core_portme.h>.

        *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 9.32.1, change it to #define COMPILER_FLAGS "High Speed; No size constraints".

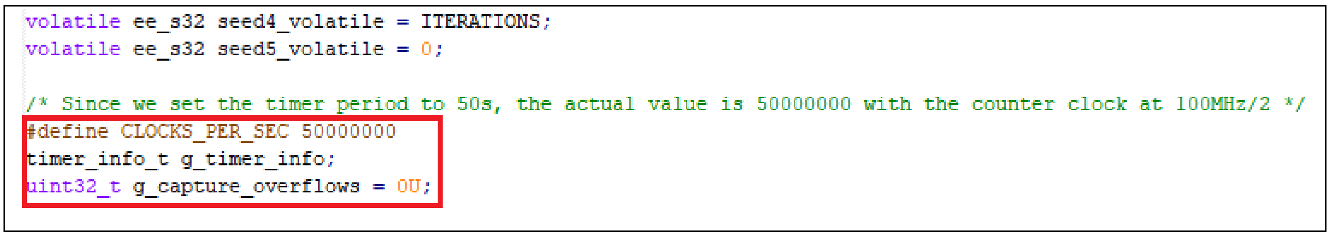
The code should look as follows.



```
/* Definitions : COMPILER_VERSION, COMPILER_FLAGS, MEM_LOCATION
        Initialize these strings per platform
*/
#ifndef COMPILER_VERSION
#ifdef __GNUC__
#define COMPILER_VERSION "GCC"__VERSION__
#else
#define COMPILER_VERSION "IAR Compiler v9.32.1"
#endif
#endif
#ifndef COMPILER_FLAGS
#define COMPILER_FLAGS "High Speed; No size constraints"
#endif
#ifndef MEM_LOCATION
#define MEM_LOCATION "STACK"
#endif
```

Figure 23. Modify core_portme.h

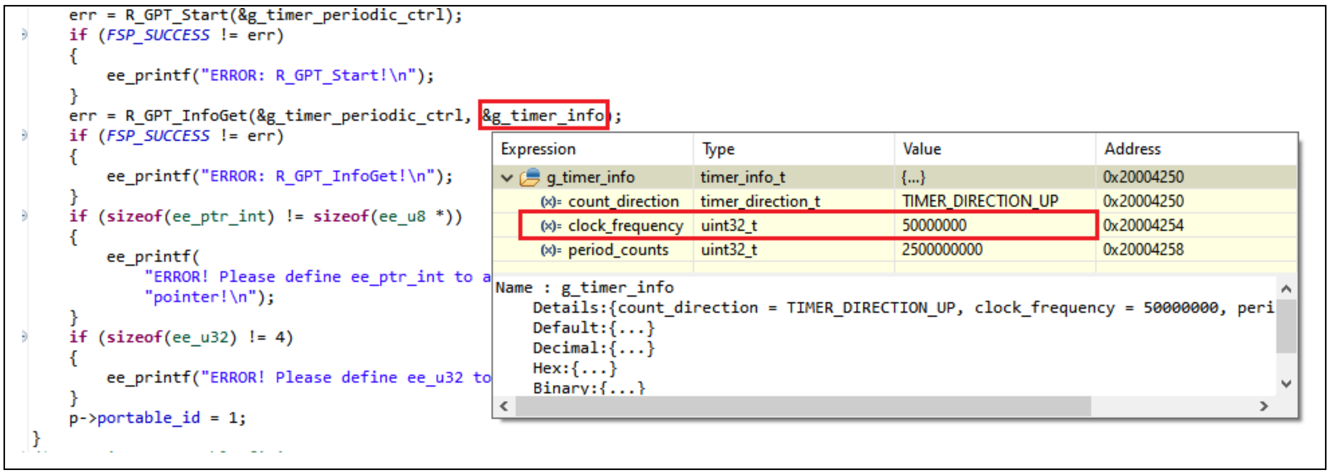
In core_portme.c, before the barebones_clock() function, add below code.



```
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.



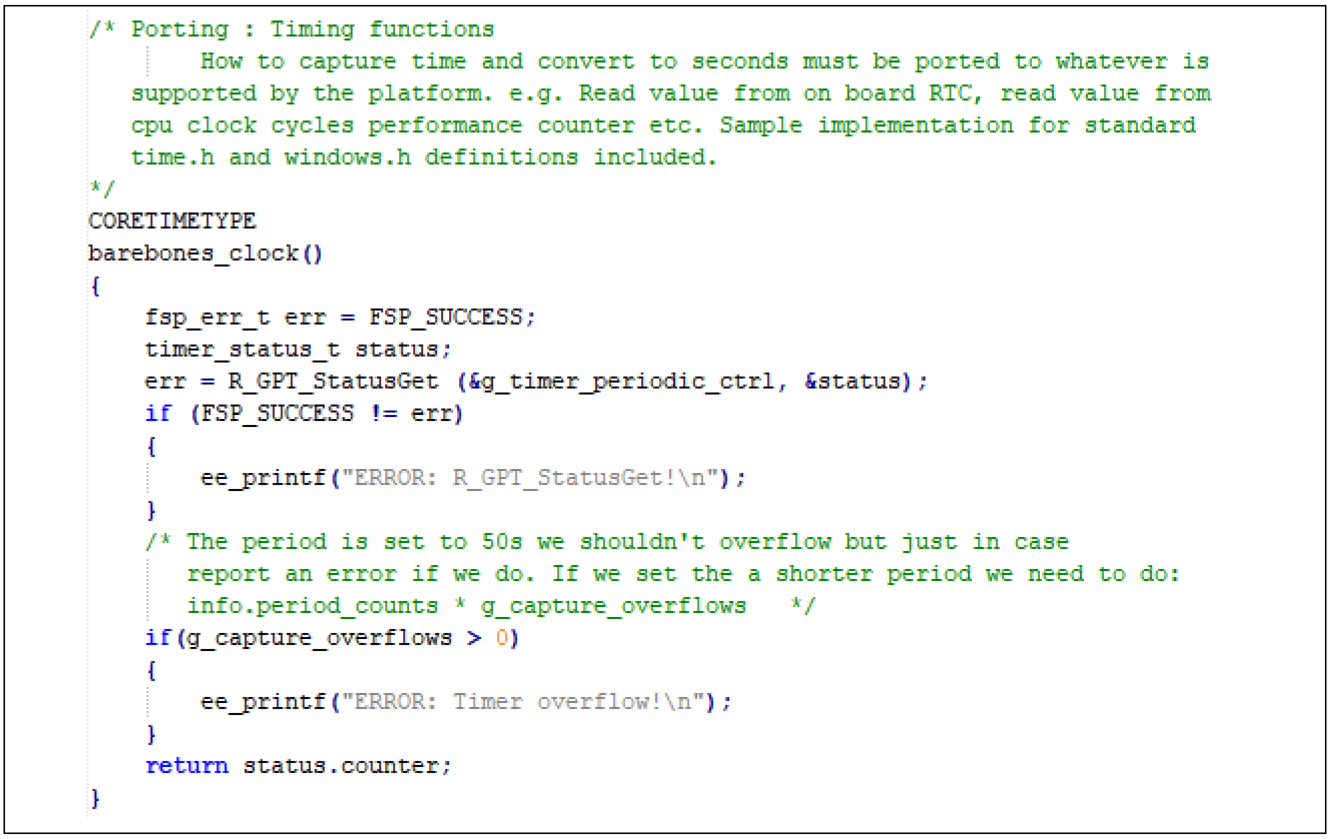
```
    err = R_GPT_Start(&g_timer_periodic_ctrl);
    if (FSP_SUCCESS != err)
    {
        ee_printf("ERROR: R_GPT_Start!\n");
    }
    err = R_GPT_InfoGet(&g_timer_periodic_ctrl, &g_timer_info);
    if (FSP_SUCCESS != err)
    {
        ee_printf("ERROR: R_GPT_InfoGet!\n");
    }
    if (sizeof(ee_ptr_int) != sizeof(ee_u8 *))
    {
        ee_printf(
            "ERROR! Please define ee_ptr_int to a
            "pointer!\n");
    }
    if (sizeof(ee_u32) != 4)
    {
        ee_printf("ERROR! Please define ee_u32 to
    }
    p->portable_id = 1;
}
```

Expression	Type	Value	Address
g_timer_info	timer_info_t	{...}	0x20004250
count_direction	timer_direction_t	TIMER_DIRECTION_UP	0x20004250
clock_frequency	uint32_t	50000000	0x20004254
period_counts	uint32_t	2500000000	0x20004258

```
Name : g_timer_info
    Details:{count_direction = TIMER_DIRECTION_UP, clock_frequency = 50000000, peri
    Default:{...}
    Decimal:{...}
    Hex:{...}
    Binary:{...}
```

Figure 25. Check CLOCK_PER_SEC Setting

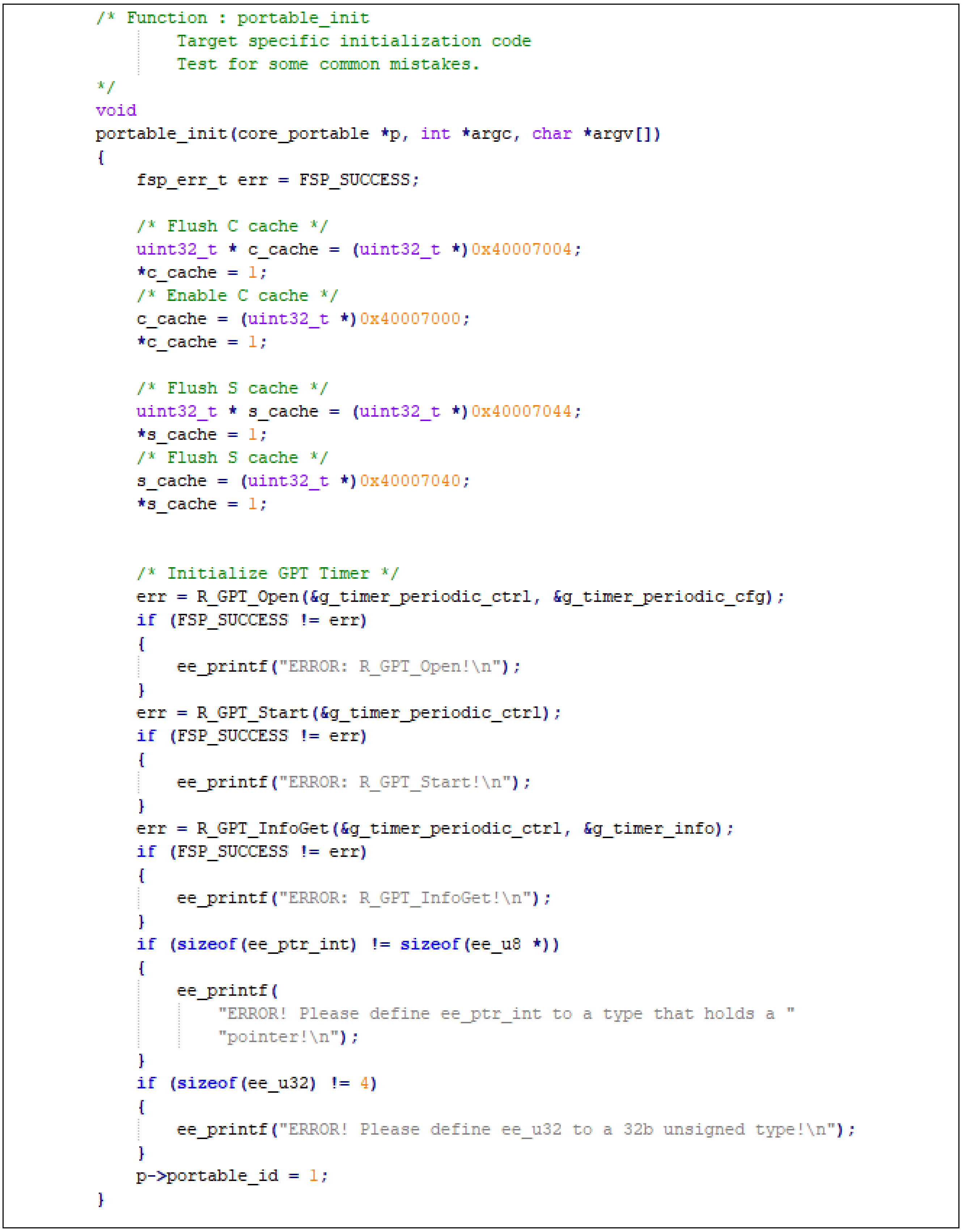
Change the barebones_clock() functions as follows.



```
/* 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;
}
```

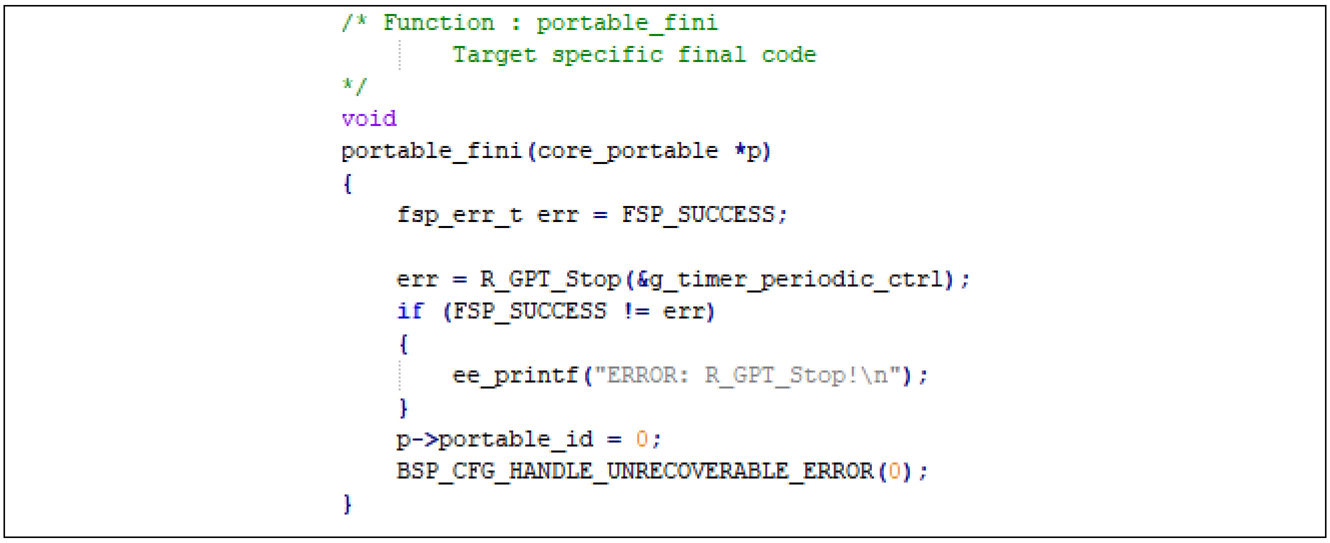
Figure 26. bareborns_clock() Function

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;
}
```

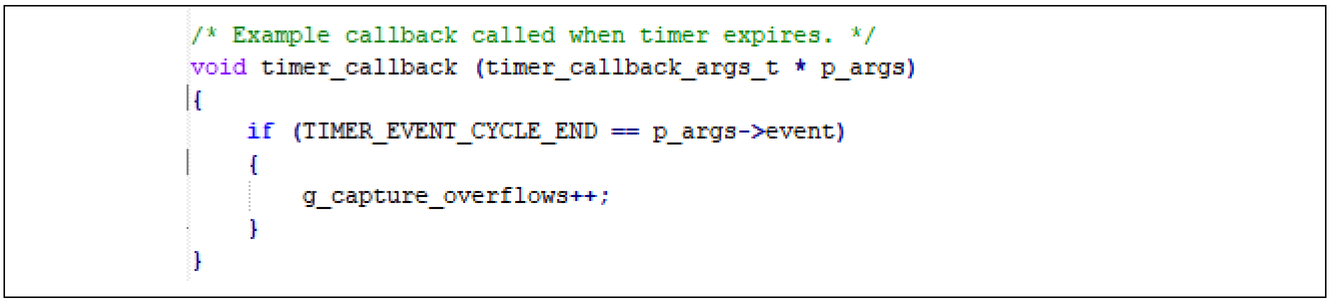
Figure 27. portable_init Function



```
/* 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

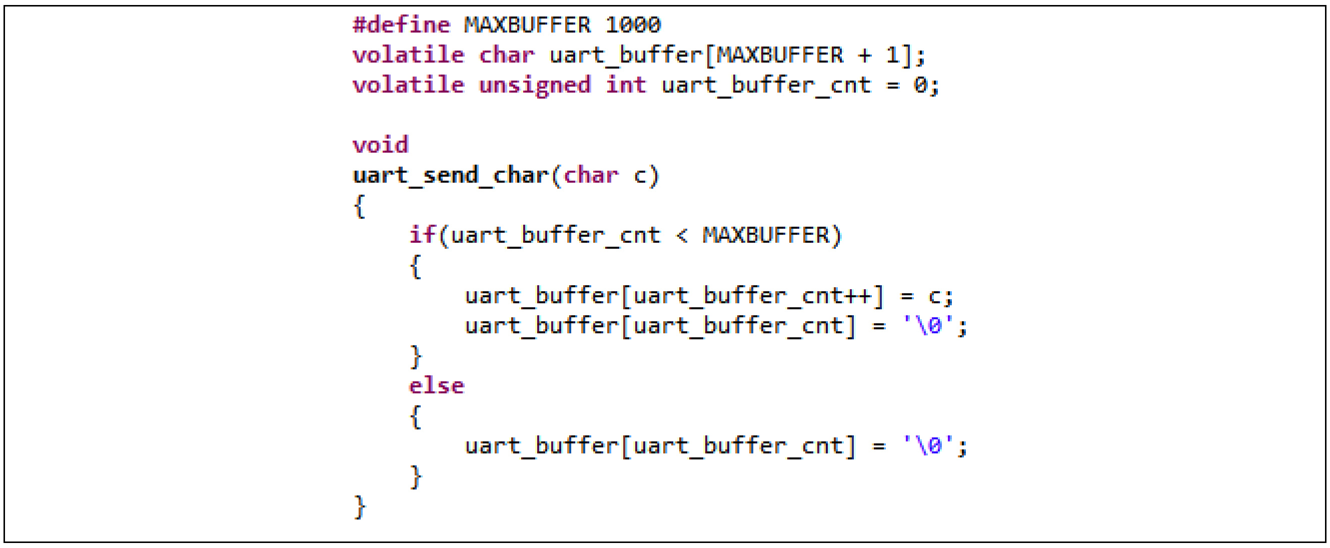
At the end of the file, add the callback method function of the GPT timer.



```
/* 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

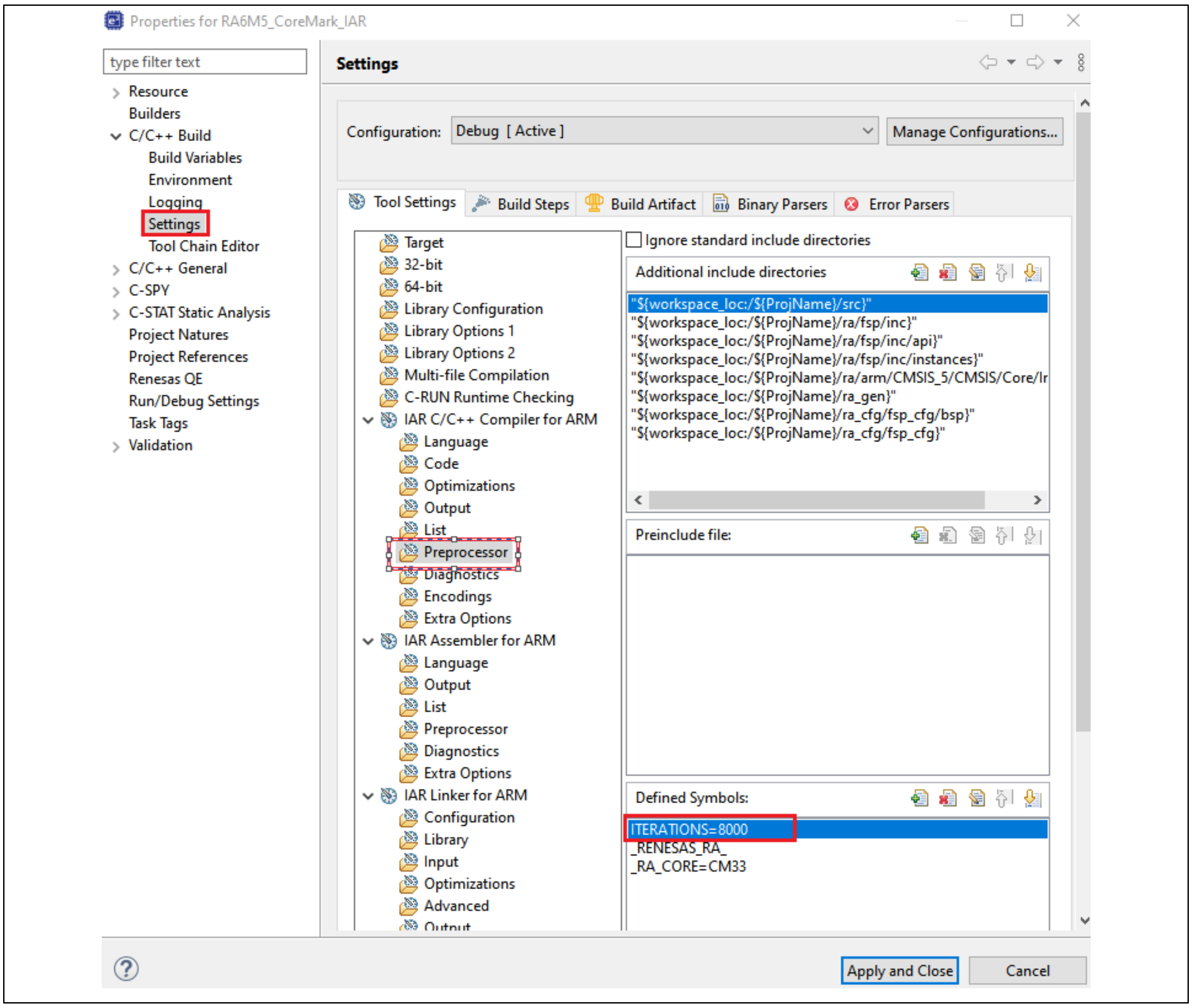
In ee_printf.c, change the uart_send_char(char c) and add the below code for printing benchmarking results.



```
#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=8000” to IAR C/C++ Compiler for ARM->Preprocessor.

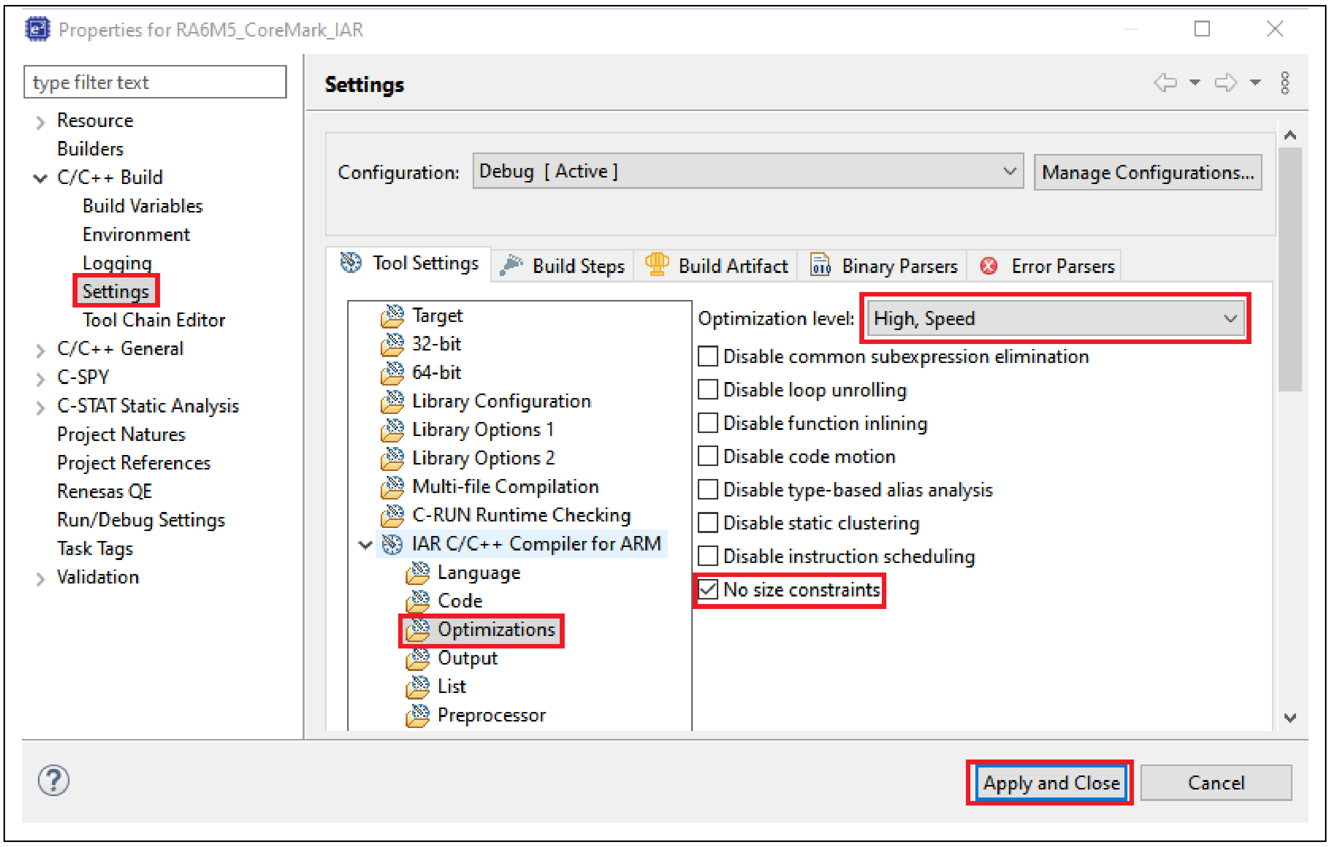


```
"${workspace_loc:/${ProjName}/src}"
"${workspace_loc:/${ProjName}/ra/fsp/inc}"
"${workspace_loc:/${ProjName}/ra/fsp/inc/api}"
"${workspace_loc:/${ProjName}/ra/fsp/inc/instances}"
"${workspace_loc:/${ProjName}/ra/arm/CMSIS_5/CMSIS/Core/Ir
"${workspace_loc:/${ProjName}/ra_gen}"
"${workspace_loc:/${ProjName}/ra_cfg/fsp_cfg/bsp}"
"${workspace_loc:/${ProjName}/ra_cfg/fsp_cfg}"
```

```
ITERATIONS=8000
_RENESAS_RA_
_RA_CORE=CM33
```

Figure 31. Preprocessor Setting

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


Properties for RA6M5_CoreMark_IAR

Settings

Configuration: Debug [Active]

Optimization level: High, Speed

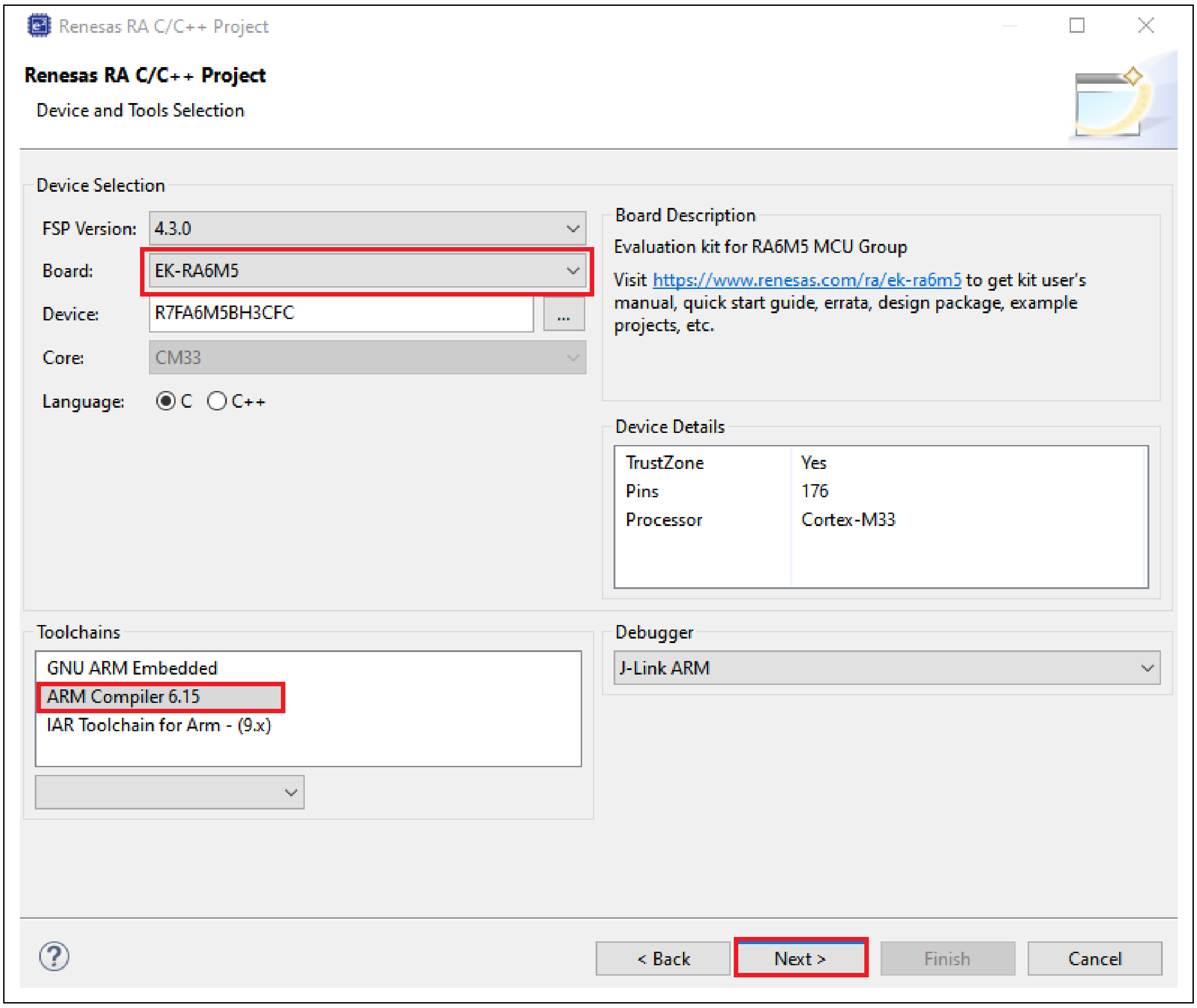
- [ ] Disable common subexpression elimination
- [ ] Disable loop unrolling
- [ ] Disable function inlining
- [ ] Disable code motion
- [ ] Disable type-based alias analysis
- [ ] Disable static clustering
- [ ] Disable instruction scheduling
- [x] No size constraints

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.


Renesas RA C/C++ Project

Renesas RA C/C++ Project

Device and Tools Selection

Device Selection

FSP Version: 4.3.0

Board: EK-RA6M5

Device: R7FA6M5BH3CFC

Core: CM33

Language: [✓ C] C++

Board Description

Evaluation kit for RA6M5 MCU Group

Visit https://www.renesas.com/ra/ek-ra6m5 to get kit user's manual, quick start guide, errata, design package, example projects, etc.

Device Details

TrustZone	Yes
Pins	176
Processor	Cortex-M33

Toolchains

GNU ARM Embedded

[✓ ARM Compiler 6.15]

IAR Toolchain for Arm - (9.x)

Debugger

J-Link ARM

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.

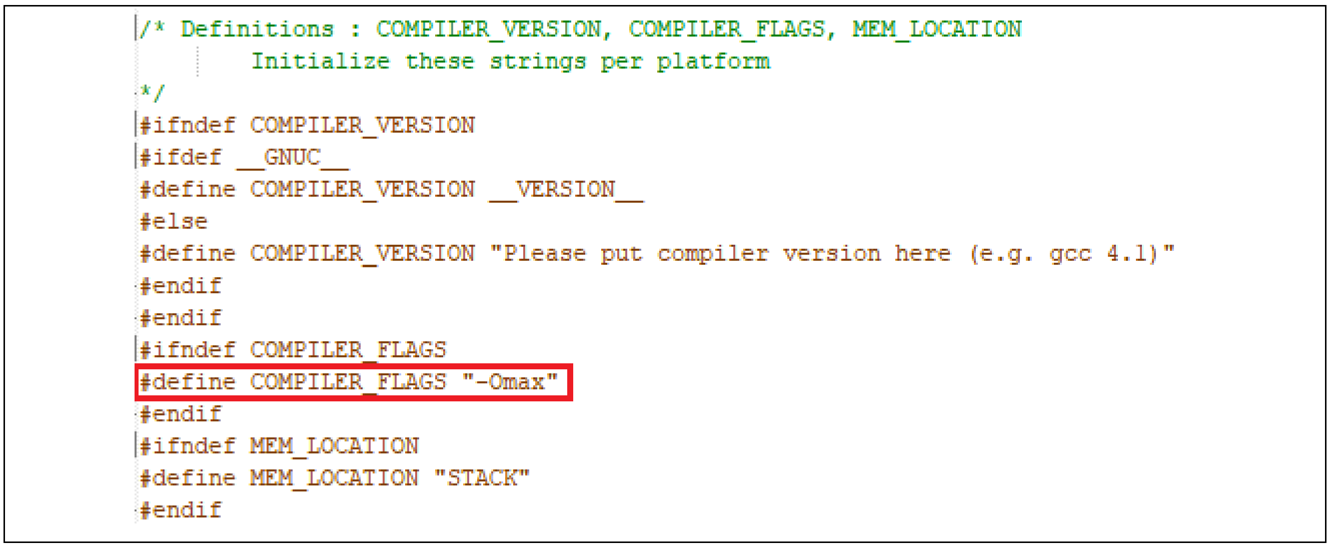
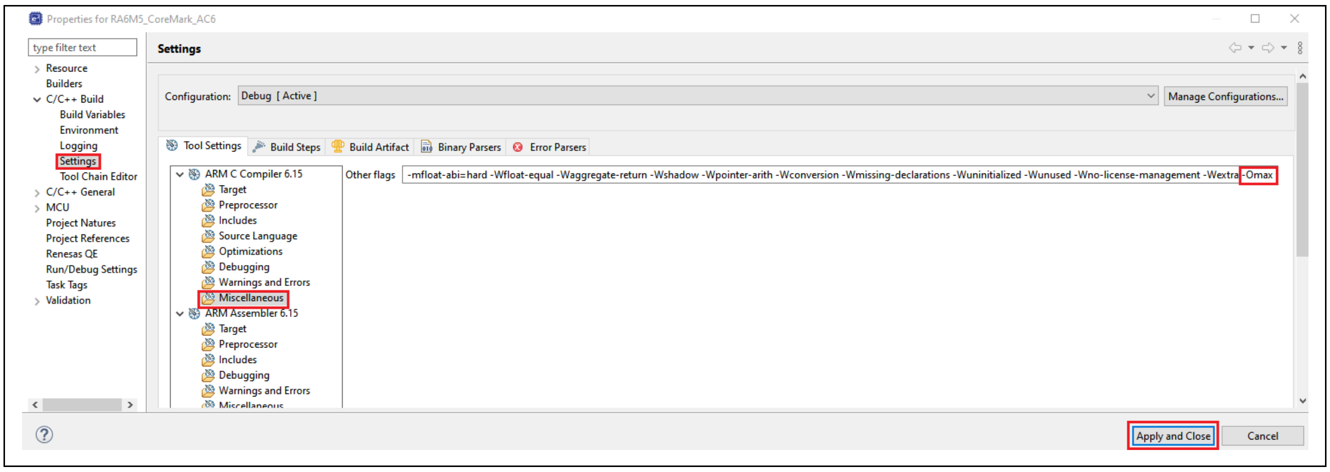

```
/* Definitions : COMPILER_VERSION, COMPILER_FLAGS, MEM_LOCATION
        Initialize these strings per platform
*/
#ifndef COMPILER_VERSION
#ifdef __GNUC__
#define COMPILER_VERSION __VERSION__
#else
#define COMPILER_VERSION "Please put compiler version here (e.g. gcc 4.1)"
#endif
#endif
#ifndef COMPILER_FLAGS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

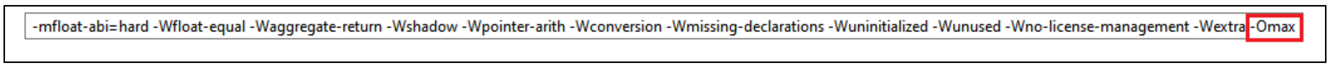
Figure 34. Modify "#define COMPILER_FLAGS"

In the project's Properties-> ARM C Compiler 6.15->Miscellaneous->Other flags, add "-Omax".



Configuration: Debug [Active]

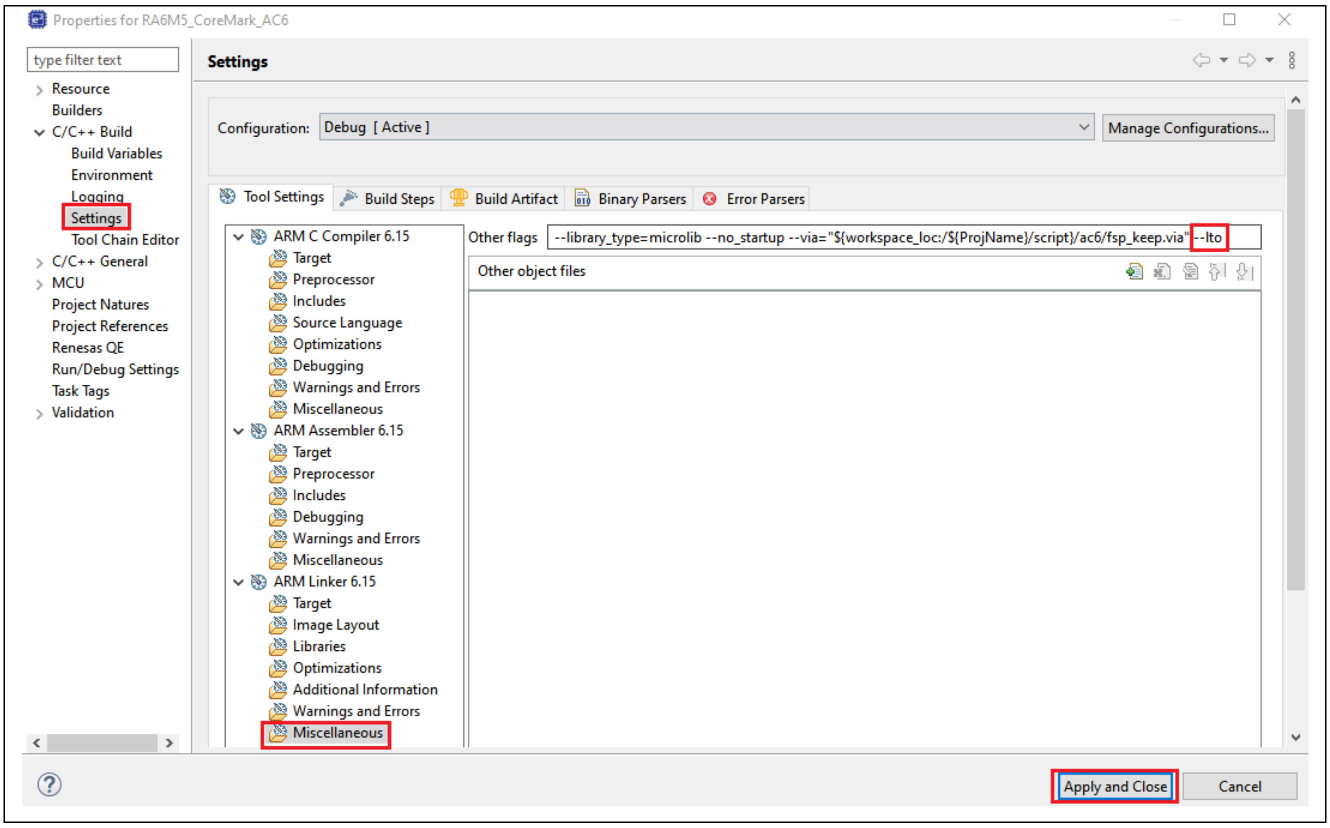
Other flags -mfloat-abi=hard -Wfloat-equal -Waggregate-return -Wshadow -Wpointer-arith -Wconversion -Wmissing-declarations -Wuninitialized -Wunused -Wno-license-management -Wextra -Omax



-mfloat-abi=hard -Wfloat-equal -Waggregate-return -Wshadow -Wpointer-arith -Wconversion -Wmissing-declarations -Wuninitialized -Wunused -Wno-license-management -Wextra -Omax

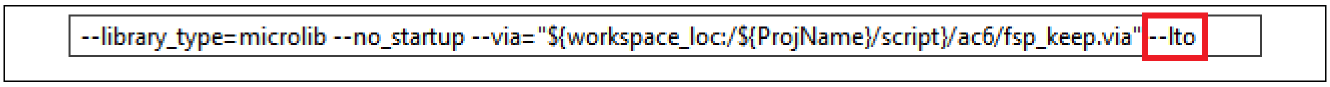
Figure 35. Add "-Omax" Option to Project Settings

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



Configuration: Debug [Active]

Other flags --library_type=microlib --no_startup --via="${workspace_loc:/${ProjName}/script}/ac6/fsp_keep.via" --lto



--library_type=microlib --no_startup --via="${workspace_loc:/${ProjName}/script}/ac6/fsp_keep.via" --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 which 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

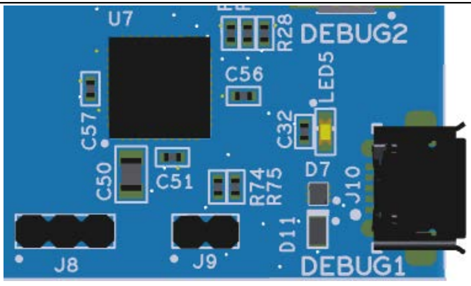


Figure 37. J8 and J9 on EK-RA6M5

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

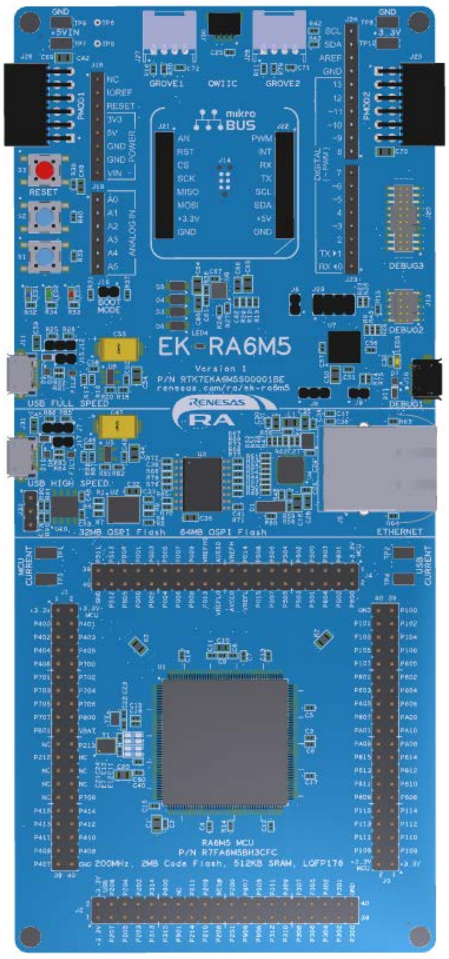


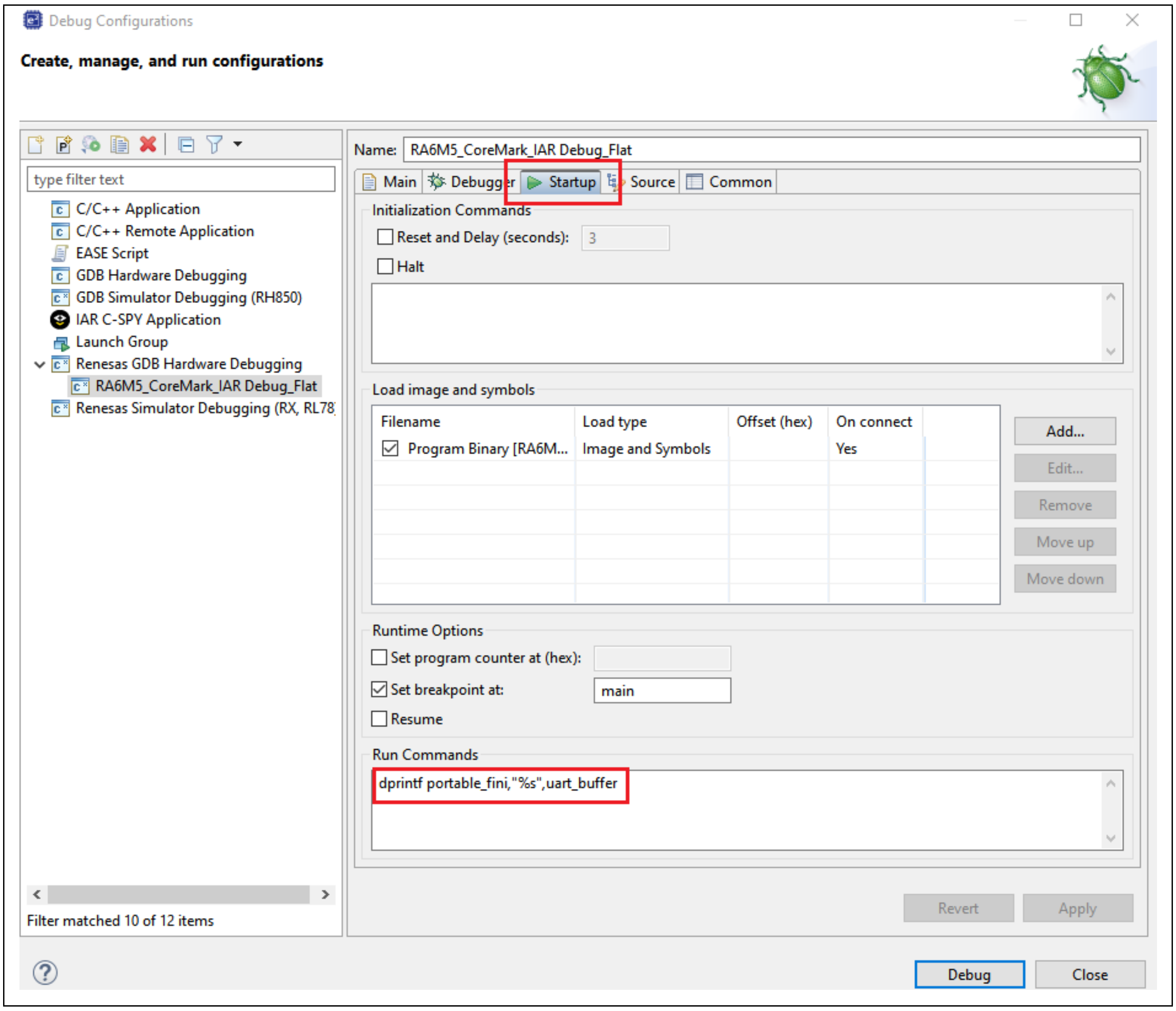
Figure 38. EK-RA6M5

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



Name: RA6M5_CoreMark_IAR Debug_Flat

Reset and Delay (seconds): 3

Filename	Load type	Offset (hex)	On connect
✓ Program Binary [RA6M...	Image and Symbols		Yes

Set program counter at (hex):

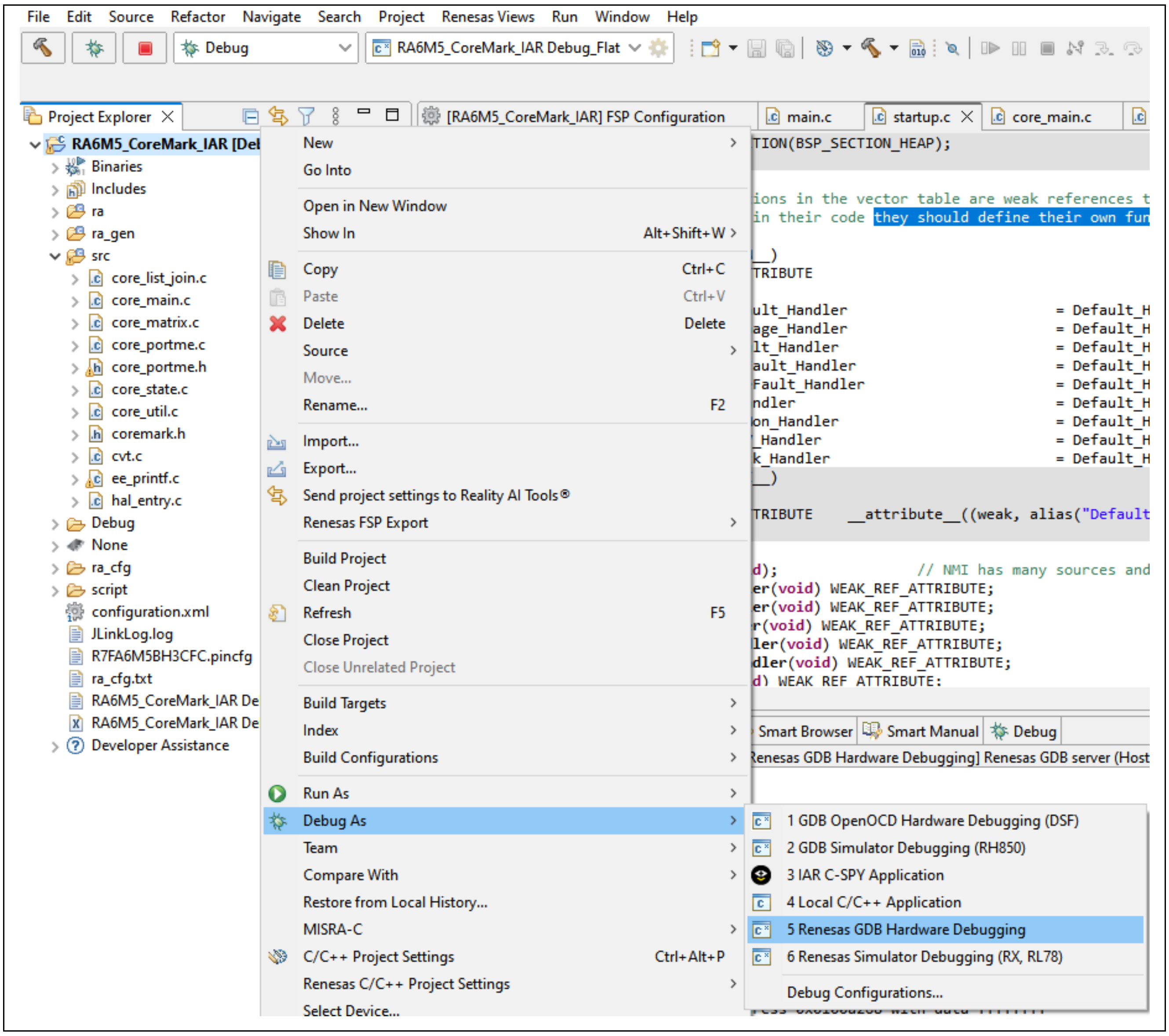
✓ Set breakpoint at: main

```
dprintf portable_fini,"%s",uart_buffer
```

Figure 39. Add dprint Command

2.8.3 Run The e2 studio Project

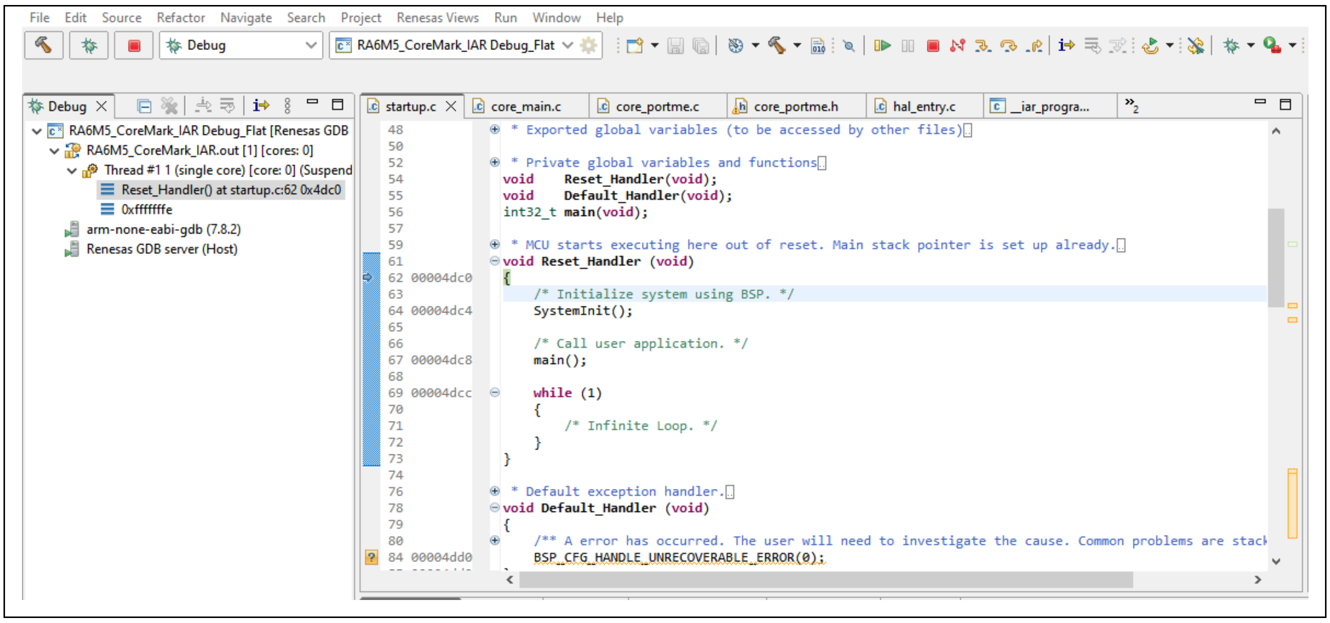
After successfully building the project, it can be debugged using Renesas GDB Hardware Debugging. Right click on the project -> Debug AS -> Renesas GDB Hardware debugging or “Debug Configurations…” and choose the desired one.



```
TION(BSP_SECTION_HEAP);

ions in the vector table are weak references t
in their code they should define their own fun

_)
TRIBUTE

ult_Handler                         = Default_H
age_Handler                         = Default_H
lt_Handler                          = Default_H
ault_Handler                        = Default_H
Fault_Handler                       = Default_H
ndler                               = Default_H
on_Handler                          = Default_H
_Handler                            = Default_H
k_Handler                           = Default_H
_)
TRIBUTE     __attribute__((weak, alias("Default

d);             // NMI has many sources and
er(void) WEAK_REF_ATTRIBUTE;
er(void) WEAK_REF_ATTRIBUTE;
r(void) WEAK_REF_ATTRIBUTE;
ler(void) WEAK_REF_ATTRIBUTE;
dler(void) WEAK_REF_ATTRIBUTE;
d) WEAK REF ATTRIBUTE:
```

Figure 40. Debug the Project

The program should stop in the Reset_handler.



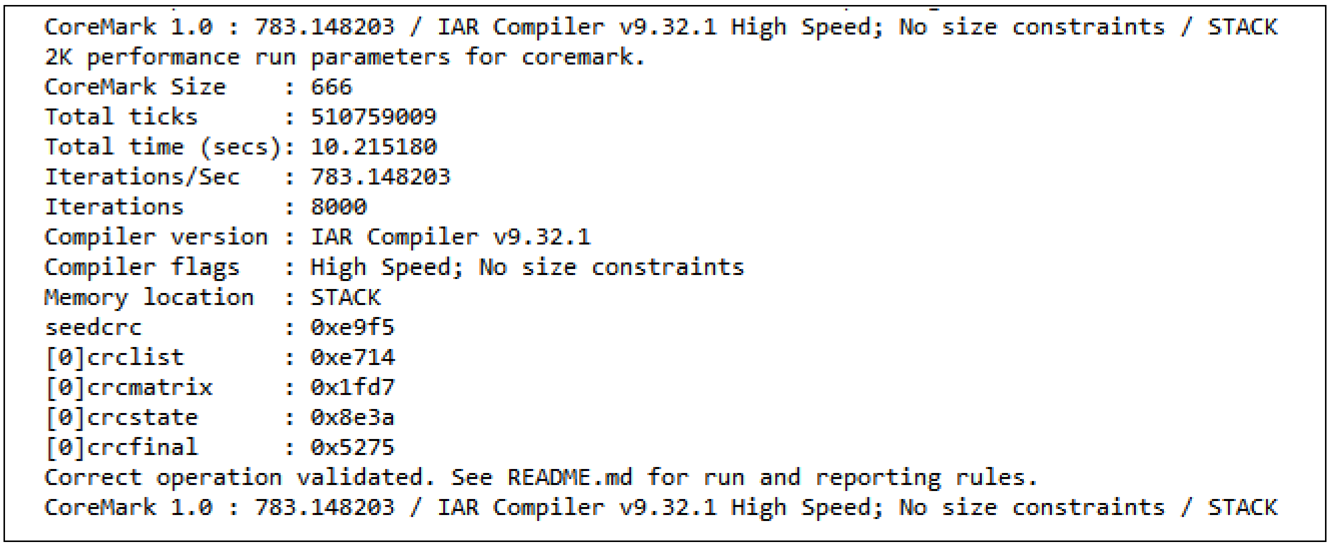
```
RA6M5_CoreMark_IAR Debug_Flat [Renesas GDB
  RA6M5_CoreMark_IAR.out [1] [cores: 0]
    Thread #1 1 (single core) [core: 0] (Suspend
      Reset_Handler() at startup.c:62 0x4dc0
      0xfffffffe
  arm-none-eabi-gdb (7.8.2)
  Renesas GDB server (Host)
```

```
          * Exported global variables (to be accessed by other files)

          * Private global variables and functions
         void    Reset_Handler(void);
         void    Default_Handler(void);
         int32_t main(void);

          * MCU starts executing here out of reset. Main stack pointer is set up already.
         void Reset_Handler (void)
00004dc0 {
             /* Initialize system using BSP. */
00004dc4     SystemInit();

             /* Call user application. */
00004dc8     main();

00004dcc     while (1)
             {
                 /* Infinite Loop. */
             }
         }

          * Default exception handler.
         void Default_Handler (void)
         {
             /** A error has occurred. The user will need to investigate the cause. Common problems are stack
00004dd0     BSP_CFG_HANDLE_UNRECOVERABLE_ERROR(0);
```

Figure 41. Debug the Project (cont'd)

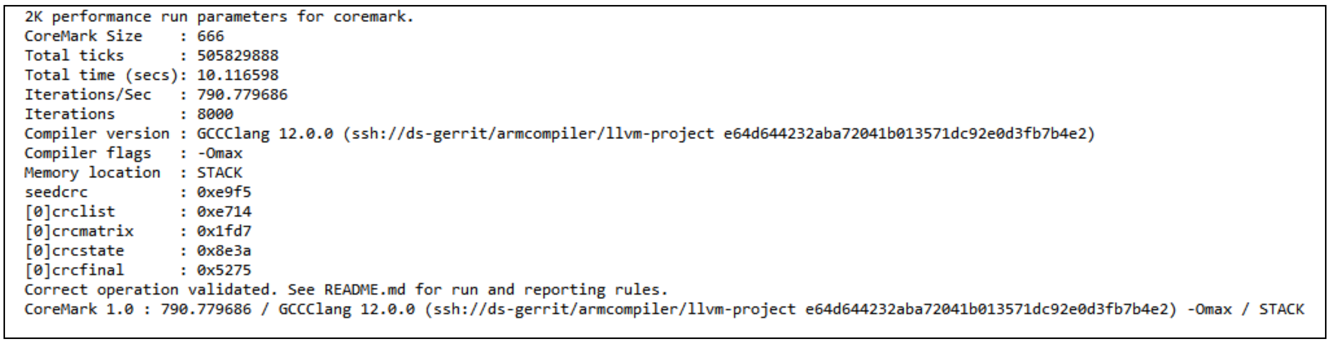
Click the Resume button. The program will stop in main from core_main.c, click the Resume button again to run the project.

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 : 783.148203 / IAR Compiler v9.32.1 High Speed; No size constraints / STACK
2K performance run parameters for coremark.
CoreMark Size    : 666
Total ticks      : 510759009
Total time (secs): 10.215180
Iterations/Sec   : 783.148203
Iterations       : 8000
Compiler version : IAR Compiler v9.32.1
Compiler flags   : High Speed; No size constraints
Memory location  : STACK
seedcrc          : 0xe9f5
[0]crclist       : 0xe714
[0]crcmatrix     : 0x1fd7
[0]crcstate      : 0x8e3a
[0]crcfinal      : 0x5275
Correct operation validated. See README.md for run and reporting rules.
CoreMark 1.0 : 783.148203 / IAR Compiler v9.32.1 High Speed; No size constraints / STACK
```

Figure 42. CoreMark Score with IAR Compiler

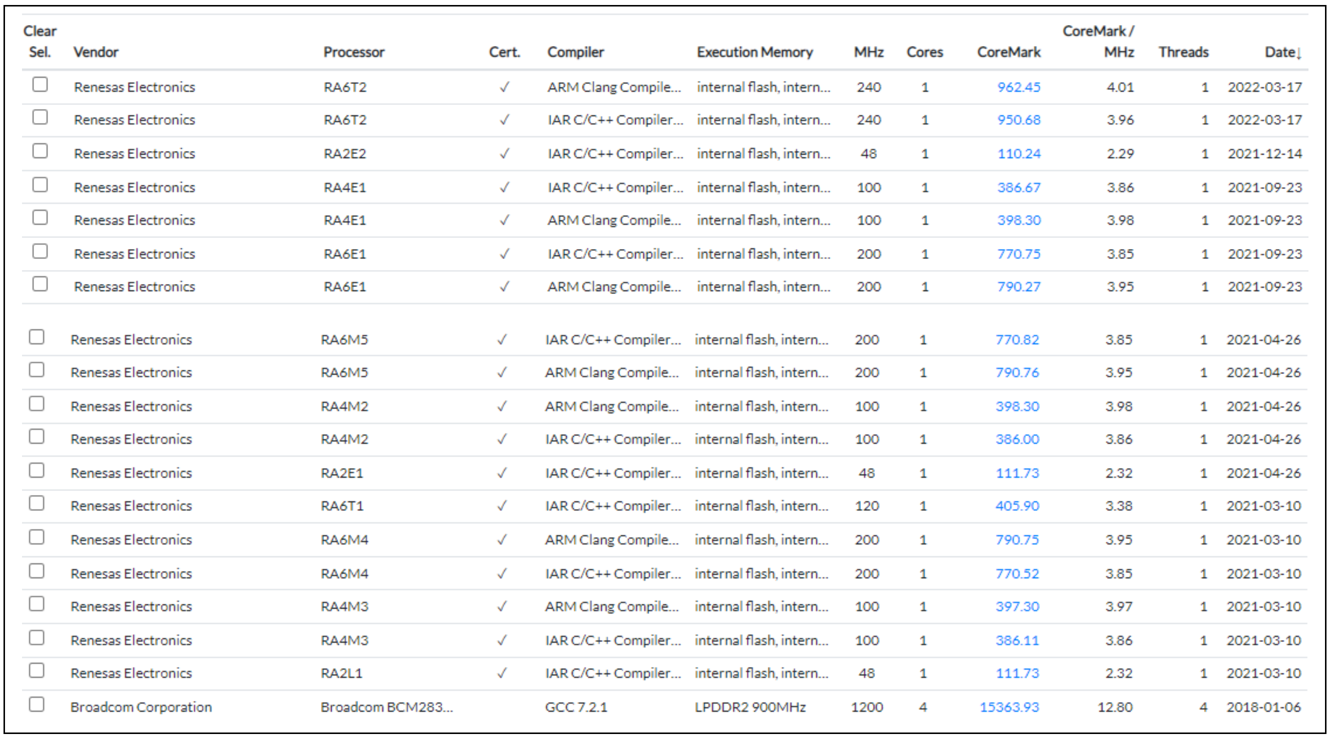


```
2K performance run parameters for coremark.
CoreMark Size    : 666
Total ticks      : 505829888
Total time (secs): 10.116598
Iterations/Sec   : 790.779686
Iterations       : 8000
Compiler version : GCCClang 12.0.0 (ssh://ds-gerrit/armcompiler/llvm-project e64d644232aba72041b013571dc92e0d3fb7b4e2)
Compiler flags   : -Omax
Memory location  : STACK
seedcrc          : 0xe9f5
[0]crclist       : 0xe714
[0]crcmatrix     : 0x1fd7
[0]crcstate      : 0x8e3a
[0]crcfinal      : 0x5275
Correct operation validated. See README.md for run and reporting rules.
CoreMark 1.0 : 790.779686 / GCCClang 12.0.0 (ssh://ds-gerrit/armcompiler/llvm-project e64d644232aba72041b013571dc92e0d3fb7b4e2) -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 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
1.0	March.20.23	-	Initial version

Corporate Headquarters

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

Contact information

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

Trademarks

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