Renesas RA

CoreMark 基准测试入门指南

要点

随着嵌入式系统中的处理器趋于复杂化，我们需要更复杂的基准来更好地去理解性能和分析。

CoreMark 是 ARM®公司推荐的一个复杂的现代基准测试，可为您准确测评处理器的性能。CoreMark 并不是使用任意或合成的代码，而是使用嵌入式应用程序中常见的基本数据结构和算法。

之所以推荐 CoreMark，是因为它符合 ANSI C 标准，而且可以确保编译器不预先计算数字以影响结果，同时在代码的基准测试部分不会进行任何库调用。

运行 CoreMark 后会产生评分，方便用户在处理器之间进行快速比较。用户还可以将结果上传到 CoreMark 网站进行认证，CoreMark 会出一个具有标准格式的结果报告。

本文档旨在介绍和解释使用 CoreMark 对瑞萨 RA MCU 进行基准测试的结果和过程。

本应用笔记将带您完成使用 CoreMark 进行基准测试所需的所有步骤。

所需资源

开发工具及软件

- e^2 studio v2024-07
- Renesas Flexible Software Package (FSP) v5.5.0
- Arm Compiler 6.21
- IAR Embedded Workbench v9.50.2

硬件

- Renesas RA 评估套件：EK-RA6M5

参考手册

- 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

目录

1 CoreMark 工程

官方 CoreMark 源代码可从 EEMBC GitHub 获得。我们计划在裸机上使用 CoreMark，使用的源文件由以下 C 源文件和头文件组成：

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

所用的三个关键算法与链表、矩阵乘法和状态机有关。

在 EEMBC GitHub 上，您还可以查看更多有关构建和运行 CoreMark 代码的规则信息。

为瑞萨 RA MCU 创建 CoreMark 工程的过程如下。

- 使用 e² studio 和灵活软件包(FSP)创建 Bare-Metal Minimal Project
- 将 CoreMark 源代码复制到“src”文件夹
- 将 32 位通用定时器(GPT)添加到工程中
- 将主栈大小设置为 0x4000 以适应 CoreMark 基准测试
- Build 时排除 FSP 生成的 main.c（不参与编译）
- 工程优化更新为最大速度选项
- 移植 core_portme.h、core_portme.c 为 GPT 添加必要的代码
- 移植 ee_printf.c 以打印基准测试结果。

本文档是以 EK-RA6M5 的程序为例，其步骤同样适用于其他 RA MCU。

2 在 Renesas RA MCU 上运行 CoreMark

除了官方 CoreMark 源代码之外，为了能够准确复制 RA MCU 基准测试中使用的流程，您需要使用 e² studio IDE 和 FSP。您可以从 https://github.com/renesas/fsp/releases 下载并安装 setup_fsp_v5_5_0_e2s_v2024-07.exe

此外，您还需要从 https://www.iar.com/products/architectures/arm/下载安装 IAR Arm 编译器，从 https://developer.arm.com/documentation/ka005198/latest 下载安装 Arm 编译器。您可以在 Keil MDK 安装中使用 Arm 编译器进行 CoreMark 基准测试。

2.1 将工具链与 e² studio 集成

2.1.1 IAR Embedded Workbench 插件

在将 IAR 编译器与 e² studio 集成之前，请下载并安装 IAR Embedded Workbench。启动 e² studio，然后选择“Help -> IAR Embedded Workbench plugin manager”。

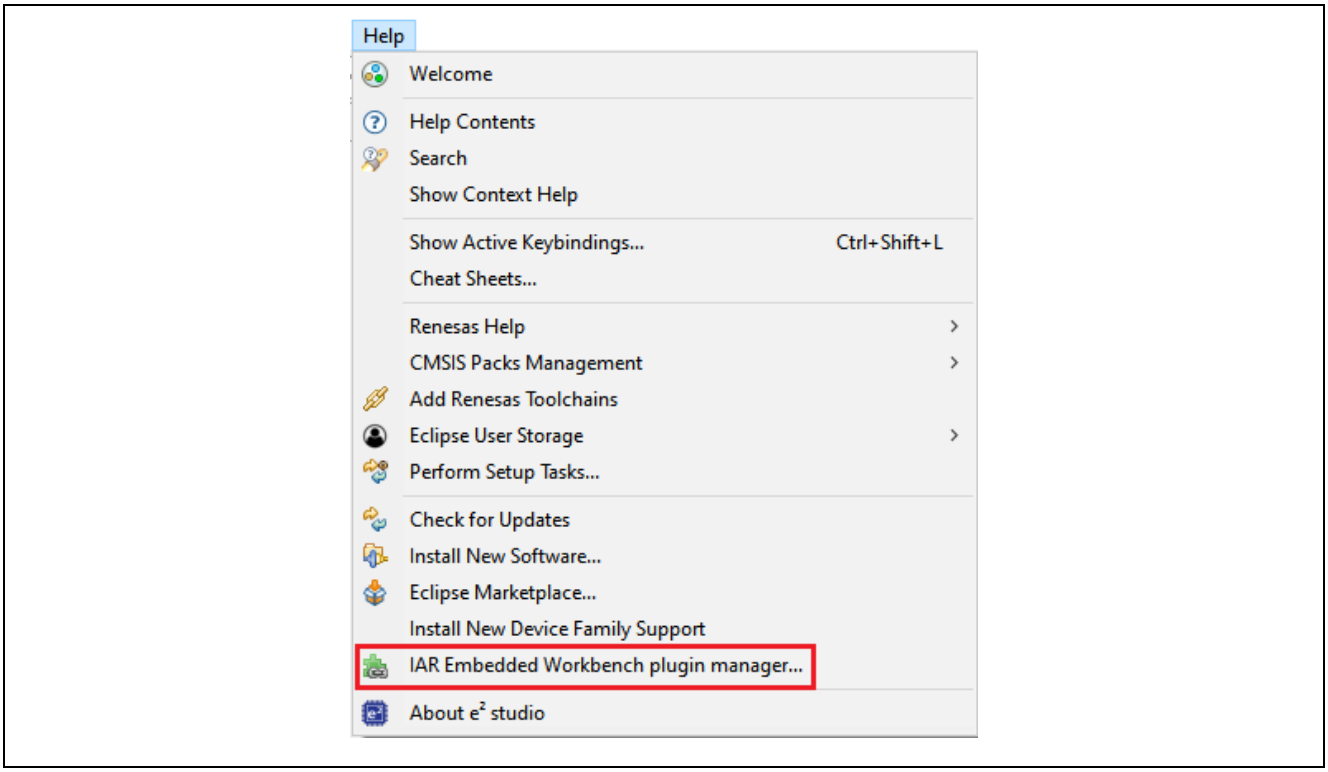

图 1. 选择 IAR Embedded Workbench plugin manager

选择所需的工具链，然后点击“Install”。

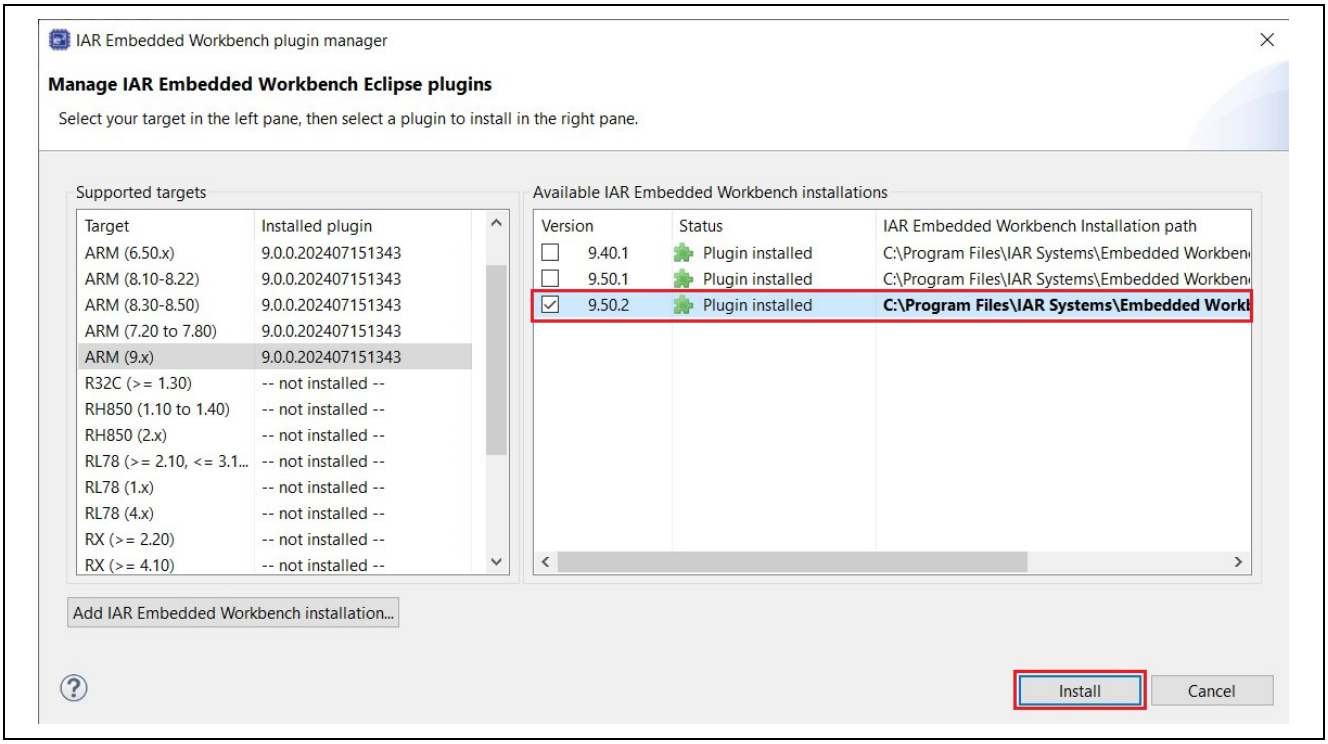

图 2. 选择 IAR 插件

e² studio IDE 的右下角将显示配置进度。

图 3. IAR 配置进度

单击“Next”，接受许可协议条款，然后单击“Finish”。

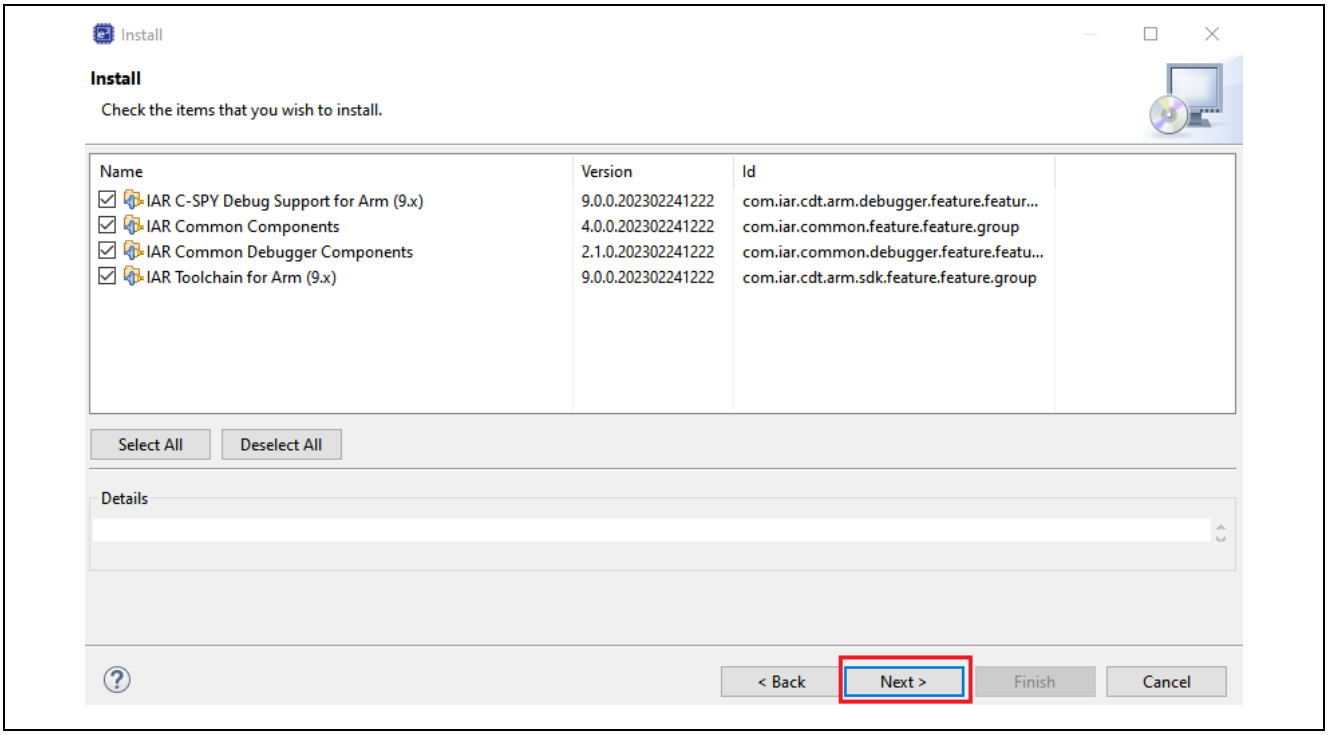

图 4. 安装 IAR Embedded Workbench 插件

e² studio IDE 的右下角将显示安装进度。

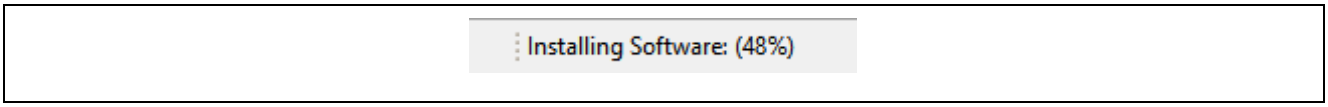

图 5. IAR 插件安装过程

插件安装完毕后，单击“Restart Now”完成安装过程。

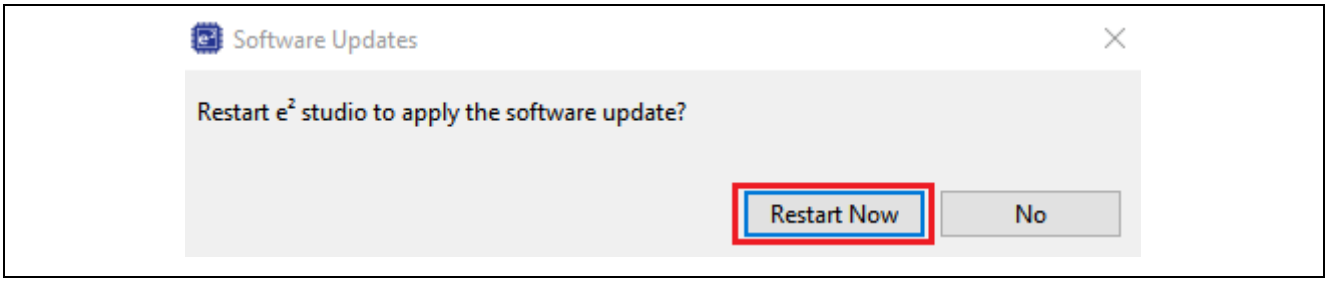

图 6. 重新启动 e² studio

2.1.2 与 Arm 编译器集成

在将 Arm 编译器与 e² studio 集成之前，请下载并安装 Arm 编译器或 Keil MDK。启动 e² studio，然后选择“Window -> Preferences”将 Toolchains 添加到 e² studio 中。

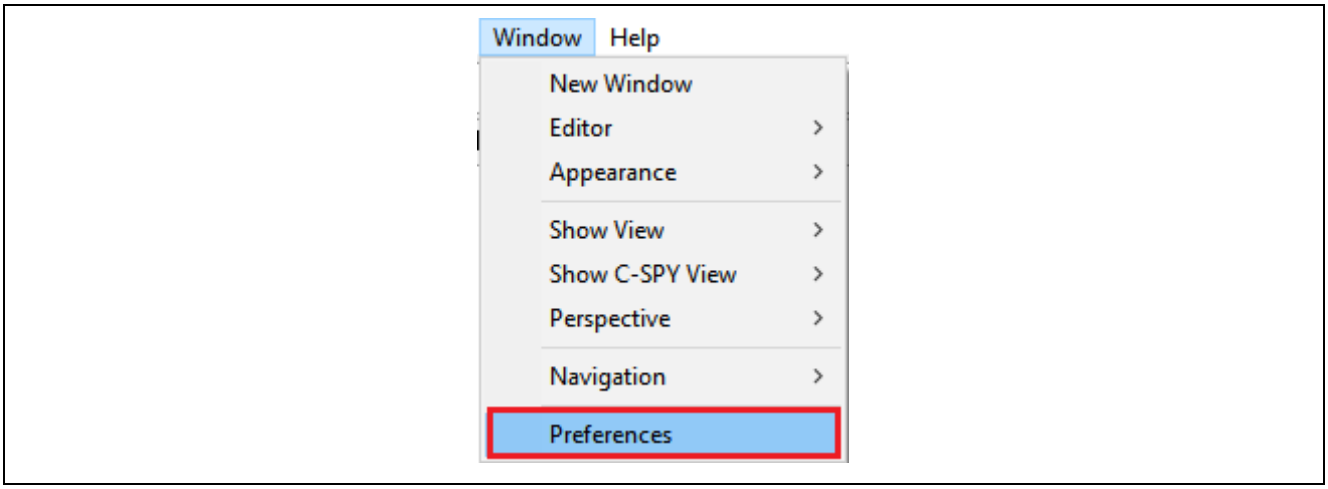

图 7. e² studio 首选项（Preferences）

选择所需的 Arm 编译器 Toolchains，然后单击“Apply and Close”将 Arm 编译器添加到 e² studio。

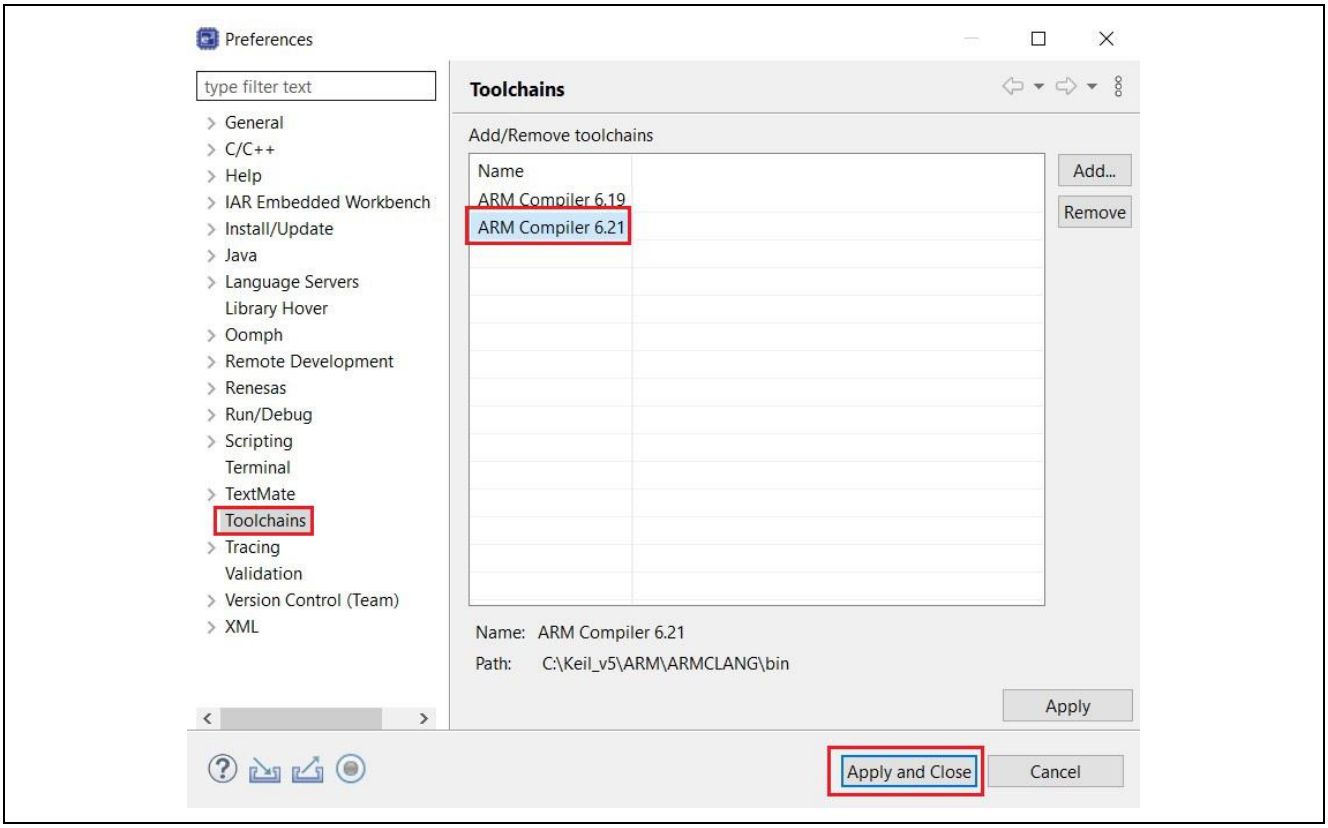

图 8. 将 Arm 编译器添加到 e² studio

如果 Toolchains 窗口中不存在 ARM 编译器，请单击“Add”，选择工具链文件夹，例如：C：\Keil_v5\Arm\\ARMCLANG\bin

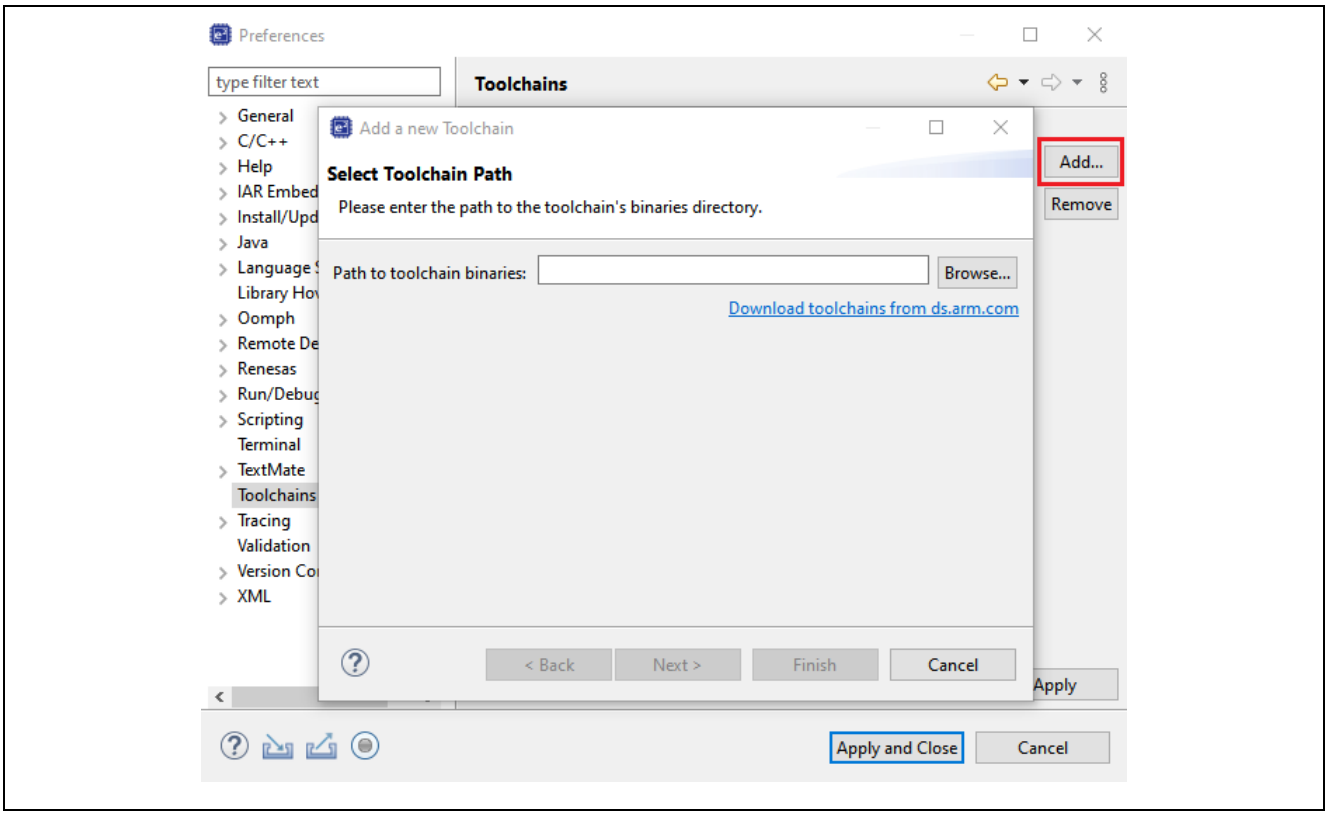

图 9. 添加 Toolchains 的路径

2.2 使用 IAR 编译器创建用于基准测试的 CoreMark e² studio 工程

在创建 CoreMark 工程之前，请确保将 IAR 编译器与 e² studio 集成。在 e² studio 中选择“File -> New-> C/C++ Project”，然后单击“Next”。

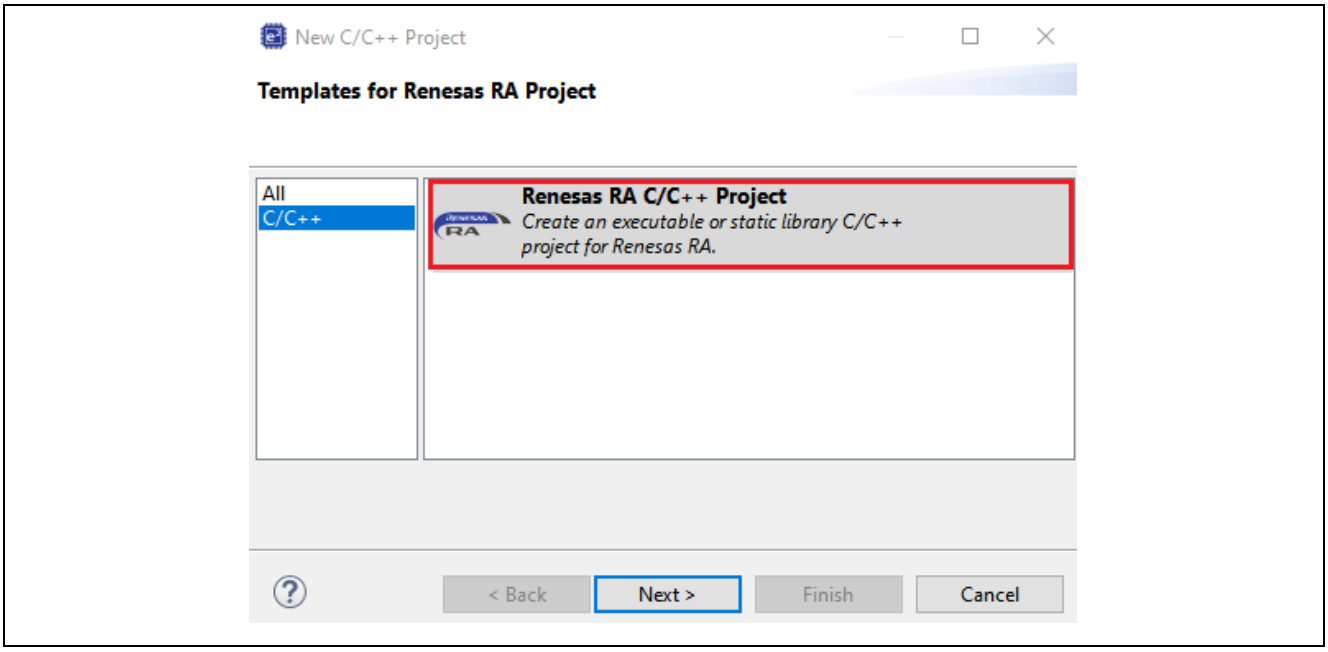

图 10. 选择 C/C++模板

使用 IAR 编译器为您的工程命名，如：RA6M5_CoreMark_IAR。

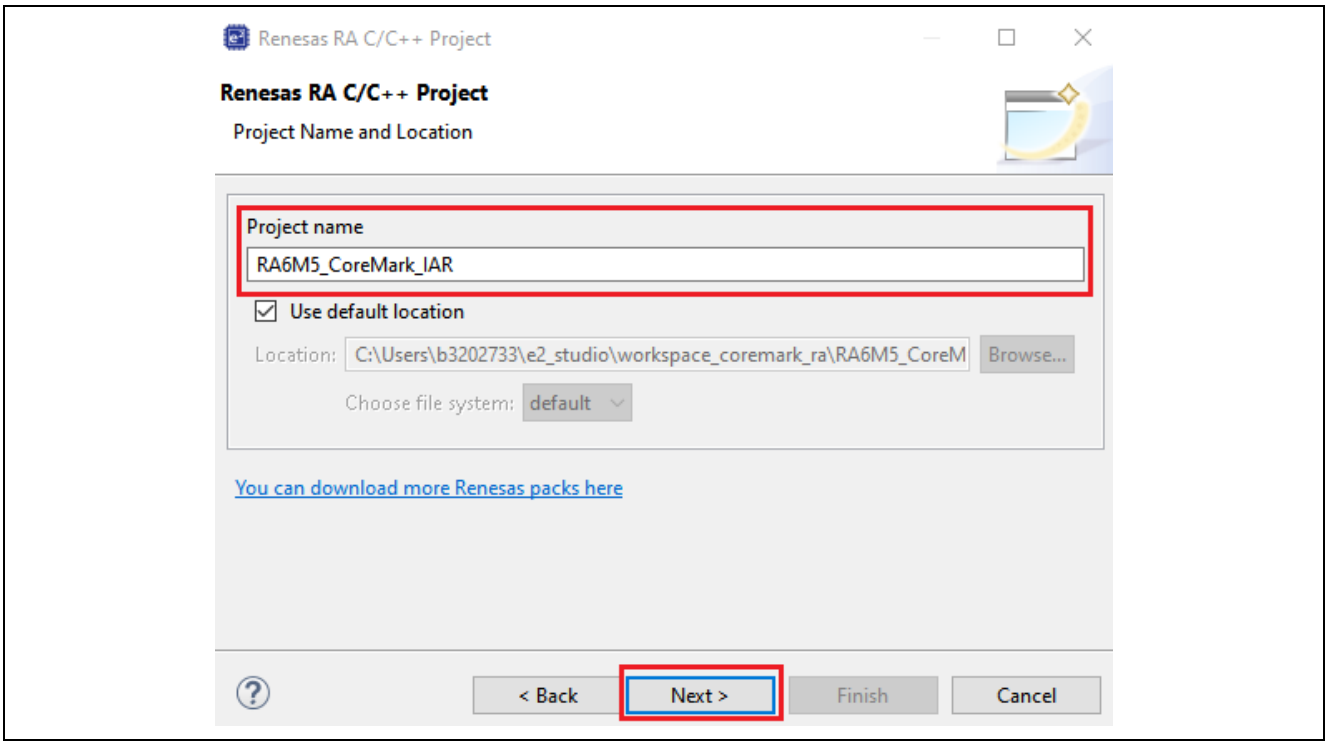

图 11. 为工程命名

选择要用于基准测试的电路板、设备和 Toolchains。

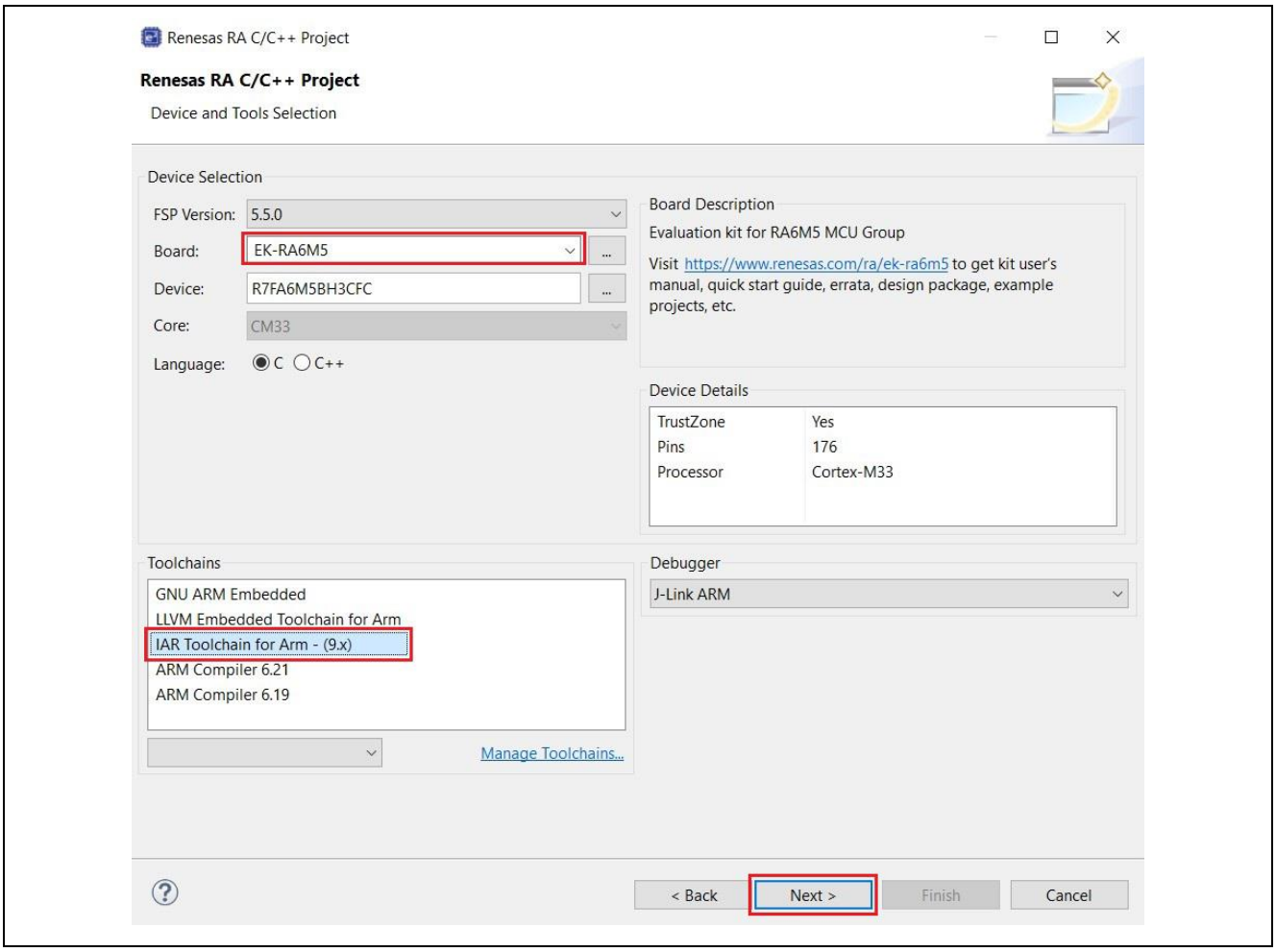

图 12. RA 工程选项

选择“Flat (Non-TrustZone) Project”。

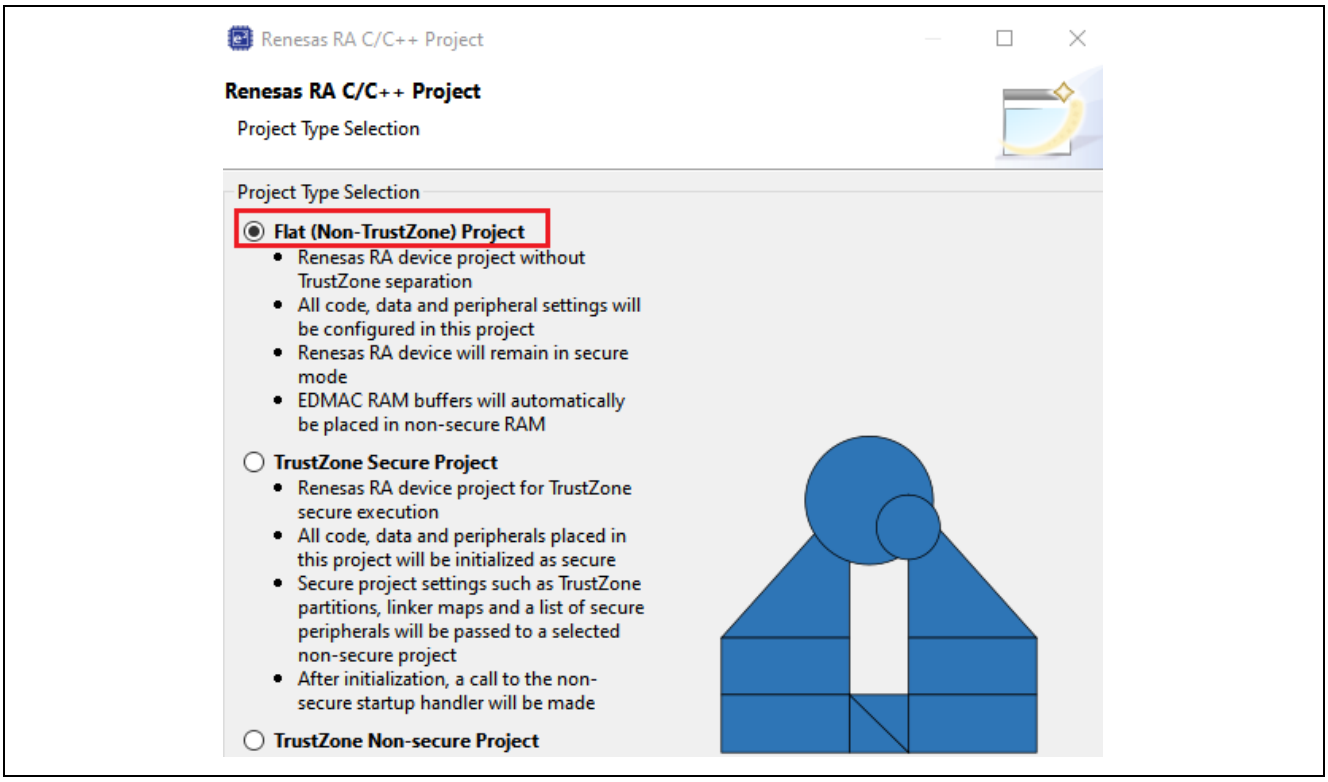

图 13. 选择 Flat (Non-TrustZone) Project

在此步骤之后，选择无 RTOS 的可执行工程类型。

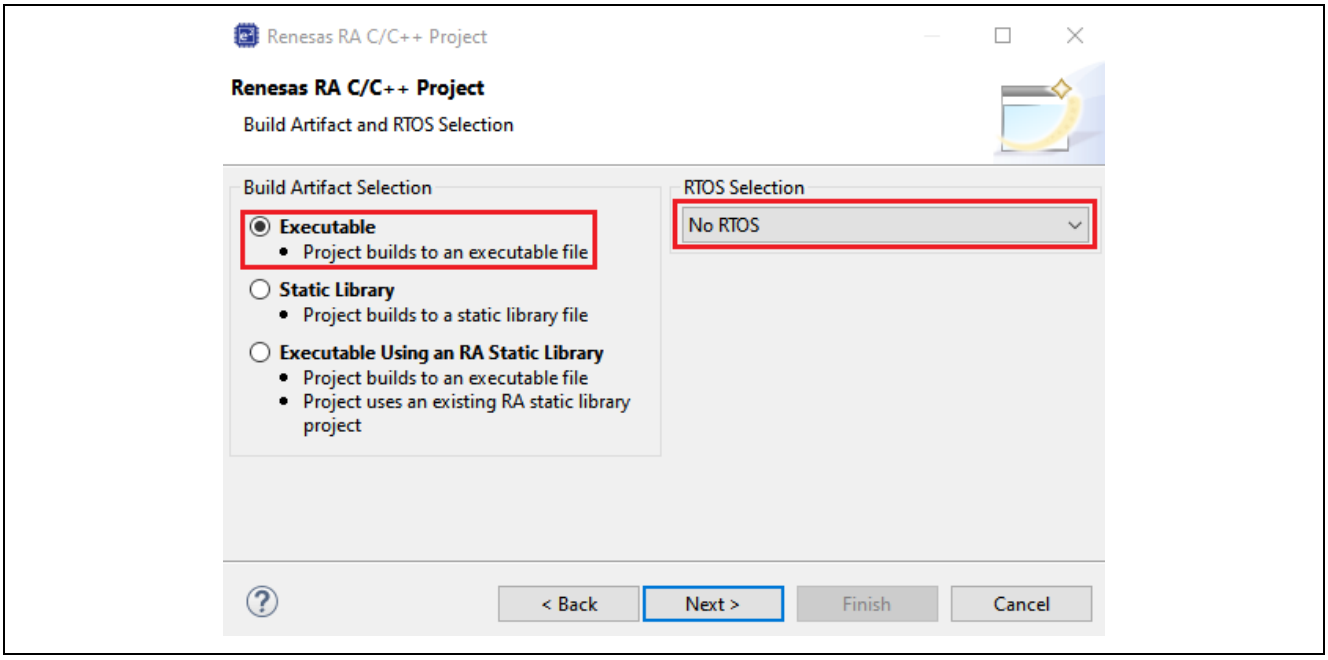

图 14. 选择“No RTOS”工程

选择“Bare Metal – Minimal”工程模板。单击“Finish”生成工程。

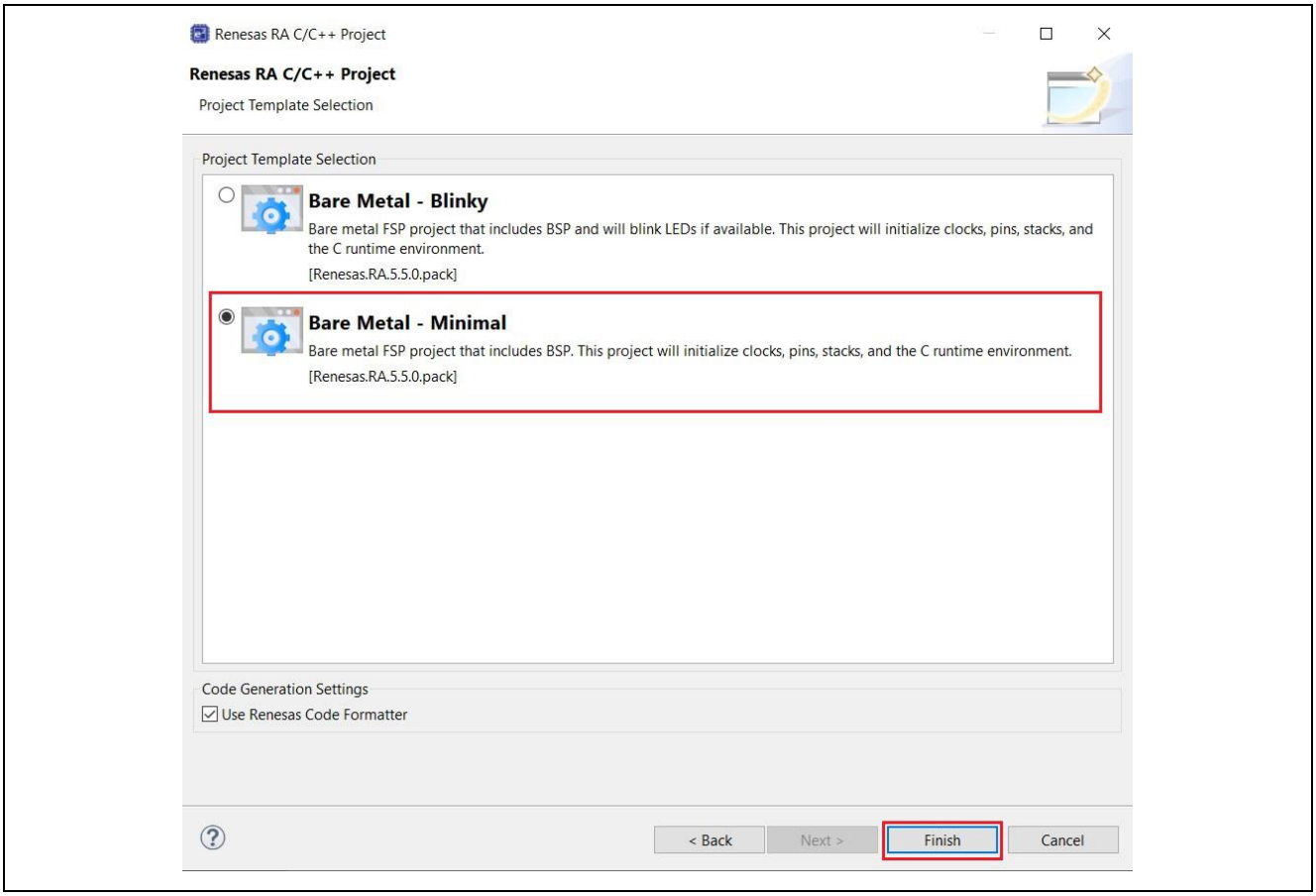

图 15. 选择“Bare Metal – Minimal”

2.3 将 CoreMark 添加到 e² studio 工程

将 CoreMark 源代码复制到新创建的工程中的“src”文件夹内。工程结构如下所示。

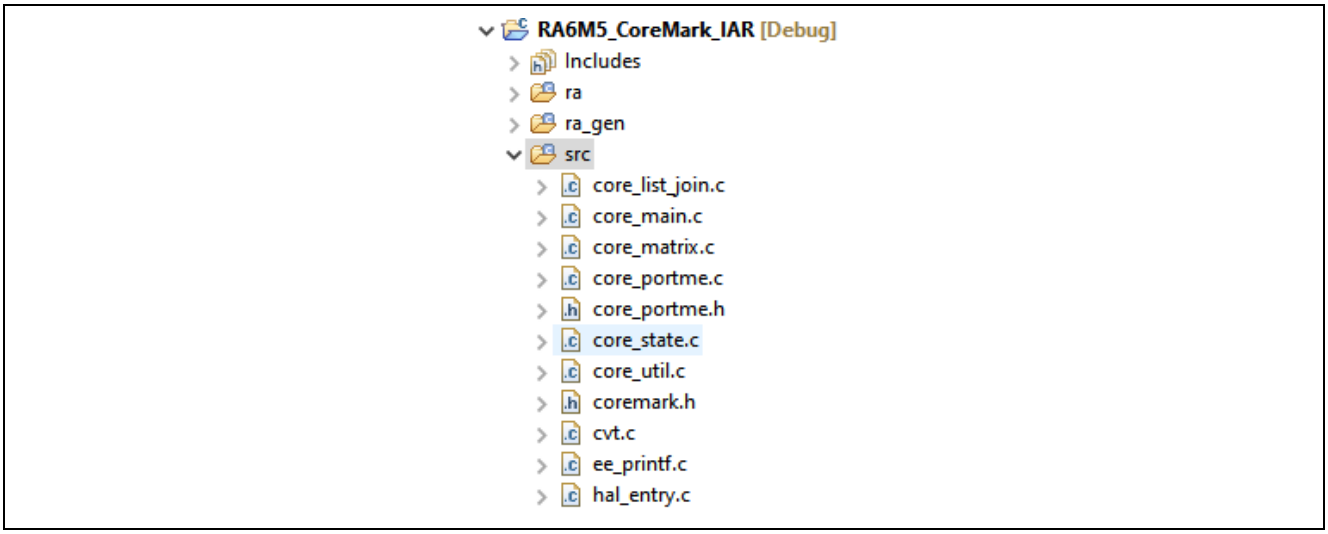

图 16. CoreMark 工程

现在需要添加一个周期性定时器并修改 core_portme.c 源文件，以便使用修改后的 barebones_clock()、portable_init(core_portable*p、int*argc、char*argv[])和 portable_fini(core_portable*p)函数。

要添加新的周期性定时器，请打开 configuration.xml 文件并转到 Stacks。然后显示类似下图的界面。

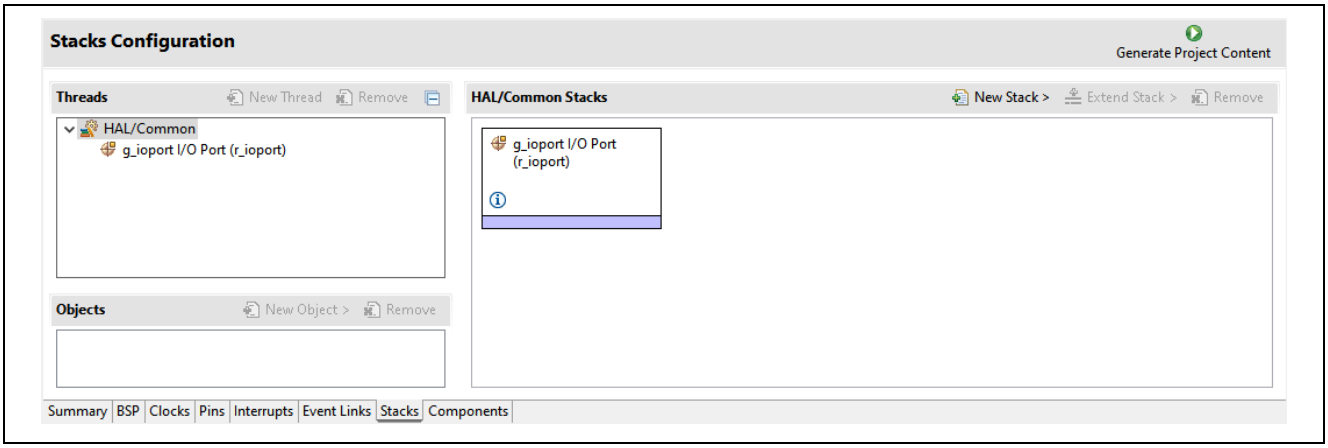

图 17. Stack Configuration（栈配置）

2.4 添加定时器进行基准测试

下一步要添加一个“New Stack”，然后选择“Timers”，最后选择“Timer, General PWM(r_gpt) ”。

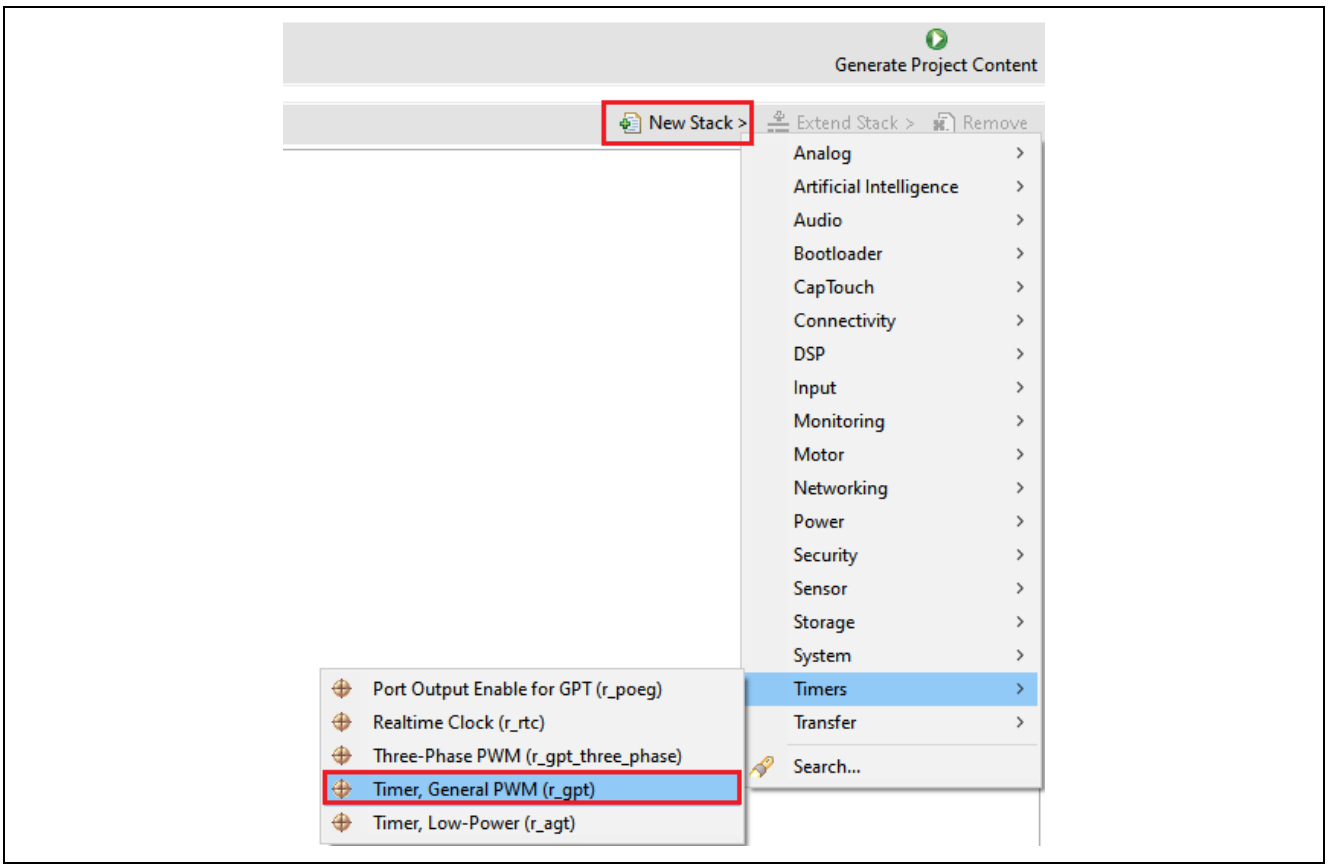

图 18. 添加 GPT 定时器

添加 GPT 定时器后需要进行设置。单击新添加的 GPT 定时器，然后转到属性窗口。可以使用此属性窗口将定时器的名称更改为 g_timer_periodic，周期改为 50，周期单位改为秒。展开 Interrupts 部分，将回调函数添加为 timer_callback，并将优先级设置为 2。更改设置后的界面如下图所示。

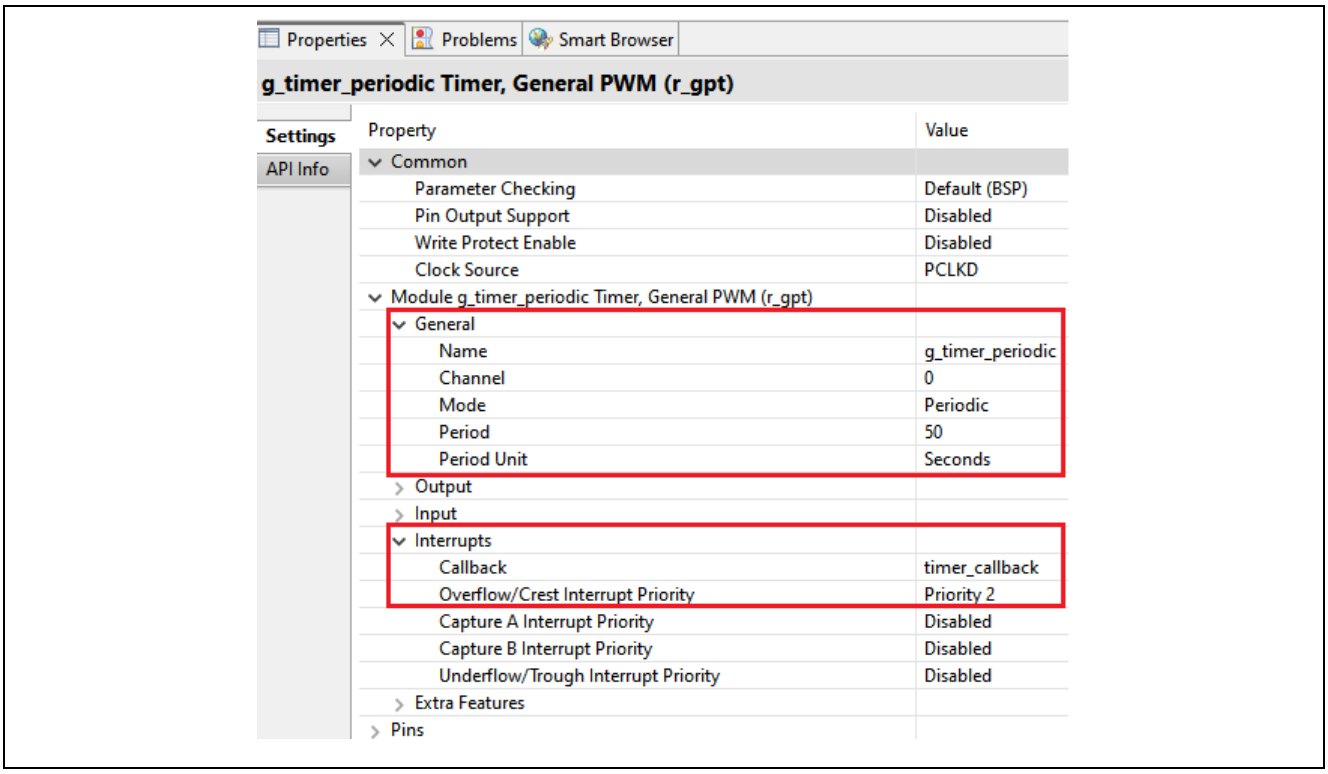

图 19.　GPT 配置

2.5　更新主栈

将 BSP 属性中的“Main stacks size (bytes)”主栈大小更改为 0x4000。

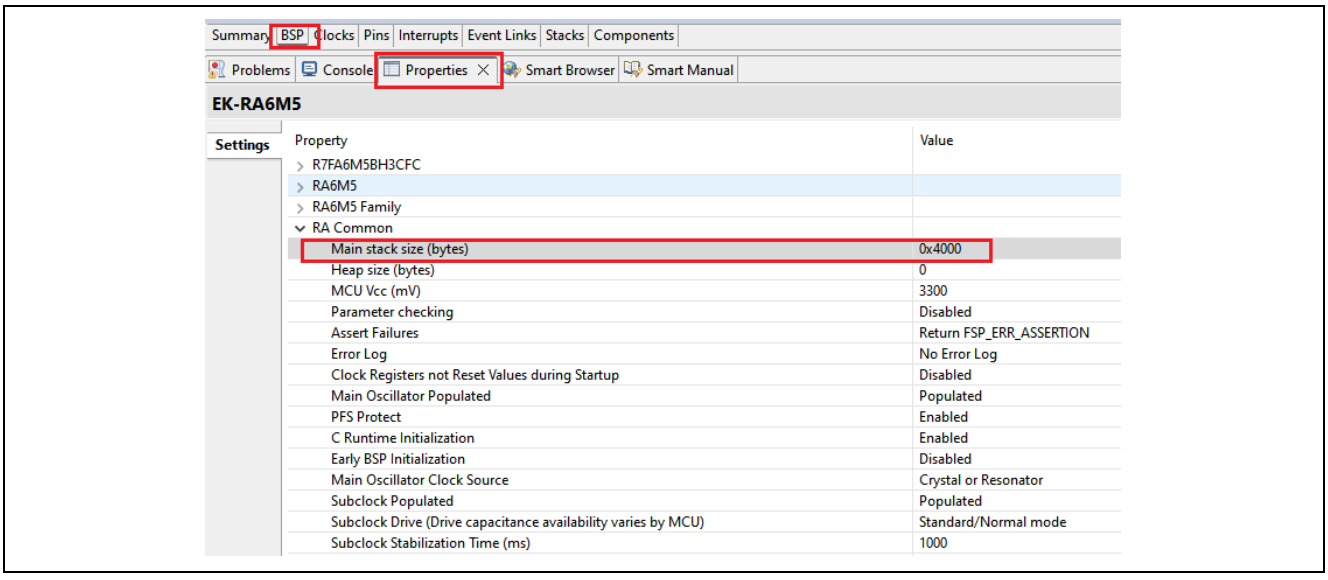

图 20.　更改 Main Stacks Size

点击“Generate Project Content”，下一步是修改源文件。

右键单击 ra_gen\main.c 并选择“Exclude from Build”，可避免与 core_main.c 中的 main 发生冲突。

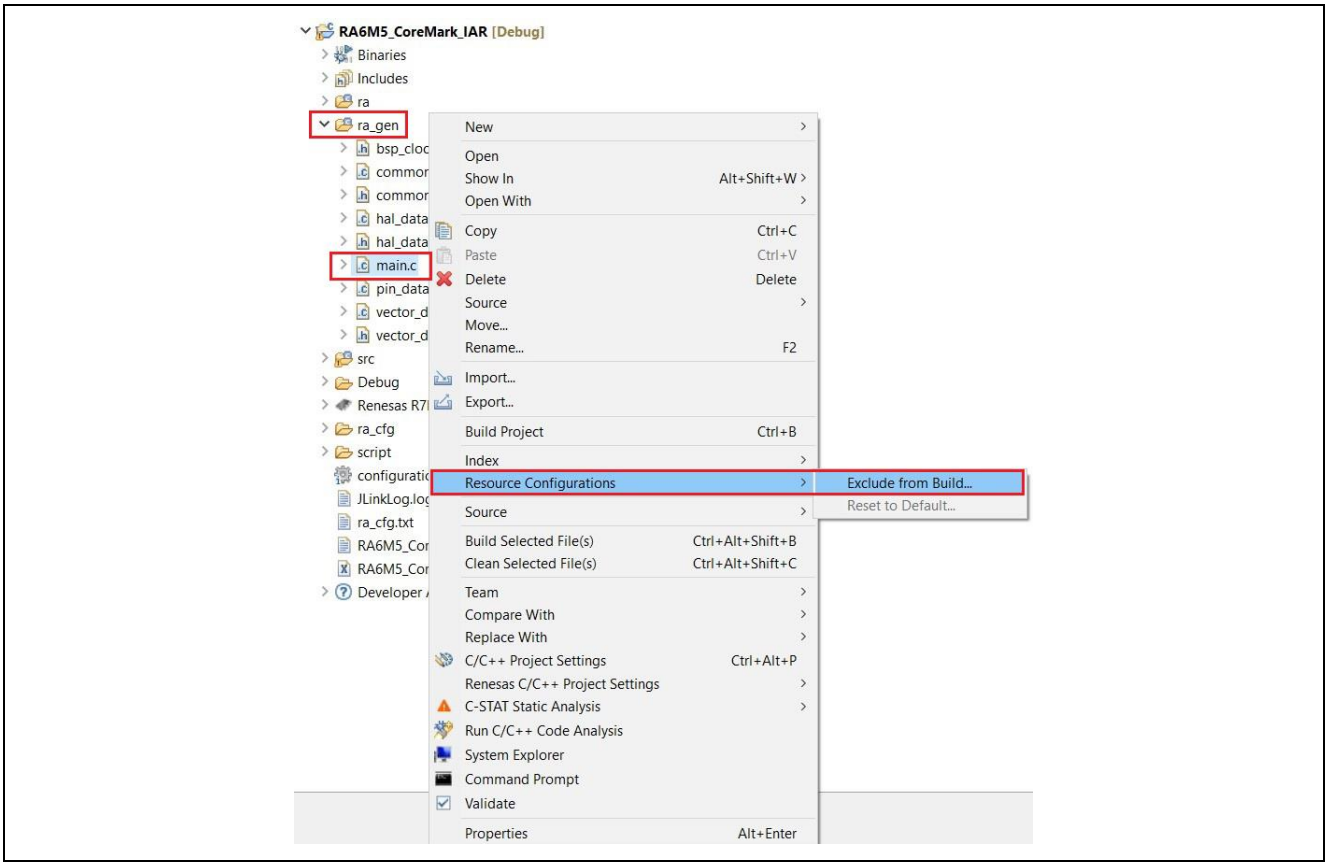

图 21. 从 Build 中排除主函数

2.6 移植 CoreMark 代码

您可以修改“src”文件夹中的 core_portme.h 和 core_portme.c。

在 core_portme.h 中的“typedef size_t ee_size_t;”代码前添加“#include<stddef.h>”，如下所示。

```
/* 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)
```

图 22. 添加“#include <stddef.h>”

此外，在 core_portme.h 中，根据所用 Toolchain 修改“#define COMPILER_FLAGS”。如果使用的是 IAR Compiler version 9.50.2，请将其更改为 #define COMPILER_FLAGS“High Speed; No size constraints”。

代码如下所示。

```
/* 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
```

图 23. 修改“core_portme.h”

在 core_portme.c 中的 barebones_clock()函数之前，添加以下代码。

```
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;
```

图 24. 修改“core_portme.c”

首次运行工程时，可以通过从 clock_frequency 获取正确的值来检查和修改 CLOCKS_PER_SEC 的设置，如下所示。

```
    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:{...}
```

图 25. 检查“CLOCK_PER_SEC”的设置

按照下图修改 bareborns_clock() 函数。

```
/* 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;
}
```

图 26. bareborns_clock() 函数

更改 portable_fini(core_portable*p)、portable_init(core_portable*p、int*argc、char*argv[])以添加基准测试所需的 GPT 定时器。

```
/* 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;
}
```

图 27. portable_init 函数

```
/* 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);
}
```

图 28. portable_fini 函数

在文件末尾，添加 GPT 定时器的回调函数。

```
/* 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++;
    }
}
```

图 29. 将 timer_callback 添加至 core_portme.c 中

在 ee_printf.c 中修改 uart_send_char(char c)并添加以下代码用以打印基准测试的结果。

```
#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';
    }
}
```

图 30. 添加 uart_send_char 函数

在工程属性设置中将“ITERATIONS=9000”添加至 IAR C/C++ Compiler for ARM- >Preprocessor 中。

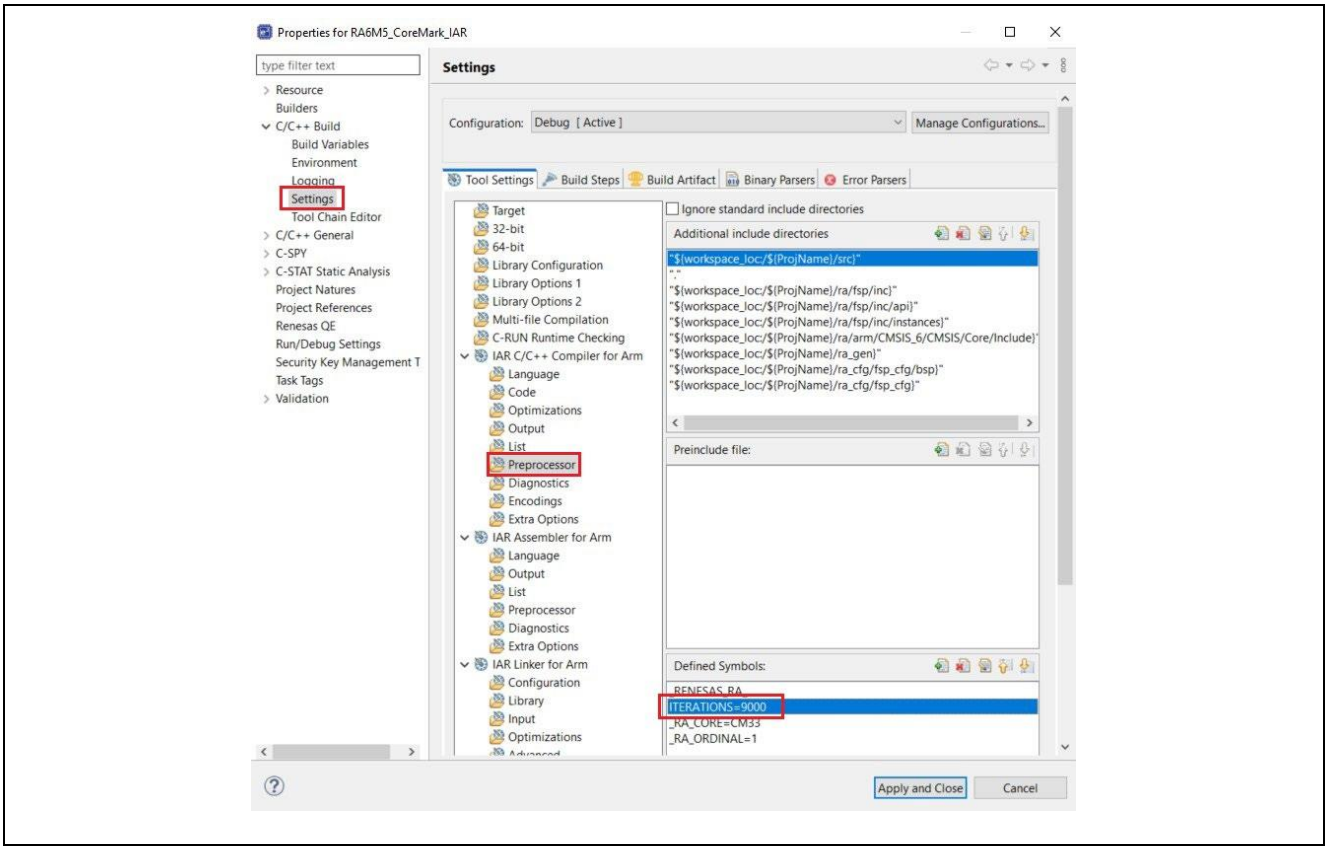

图 31. 设置 Preprocessor

在工程属性设置中，将 IAR C/C++ Compiler for ARM->Optimization 修改为“High, Speed”及“No size constraints”。

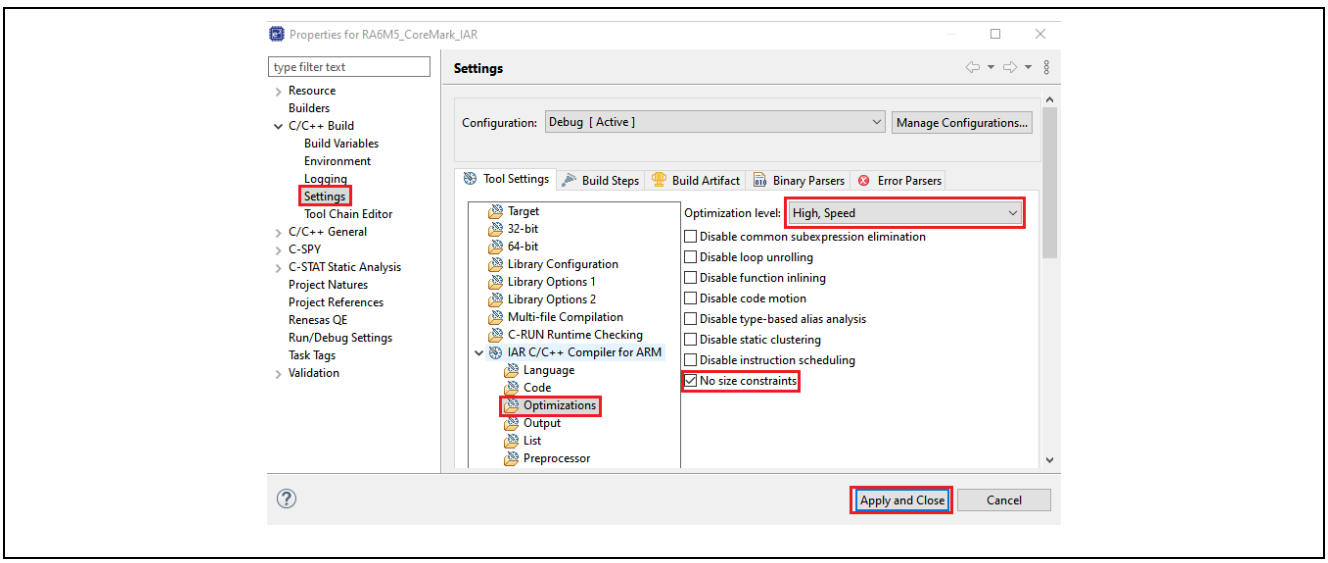

图 32. 优化

接下来便可以无报错地编译该工程了。

2.7 使用 Arm 编译器创建用于基准测试的 CoreMark e² studio 工程

在创建 CoreMark 工程之前，请确保 Arm 编译器已与 e² studio 集成。选择用于基准测试的电路板、设备和 Toolchain，并创建类似于 IAR 编译器的基于 Arm 编译器的工程。按照第 2.3、2.4、2.5 和 2.6 节的内容添加 GPT 模块、配置工程并移植 CoreMark。需要注意的是，要在 Arm 编译器中使用“-lto”选项，需要相应的商业许可证。

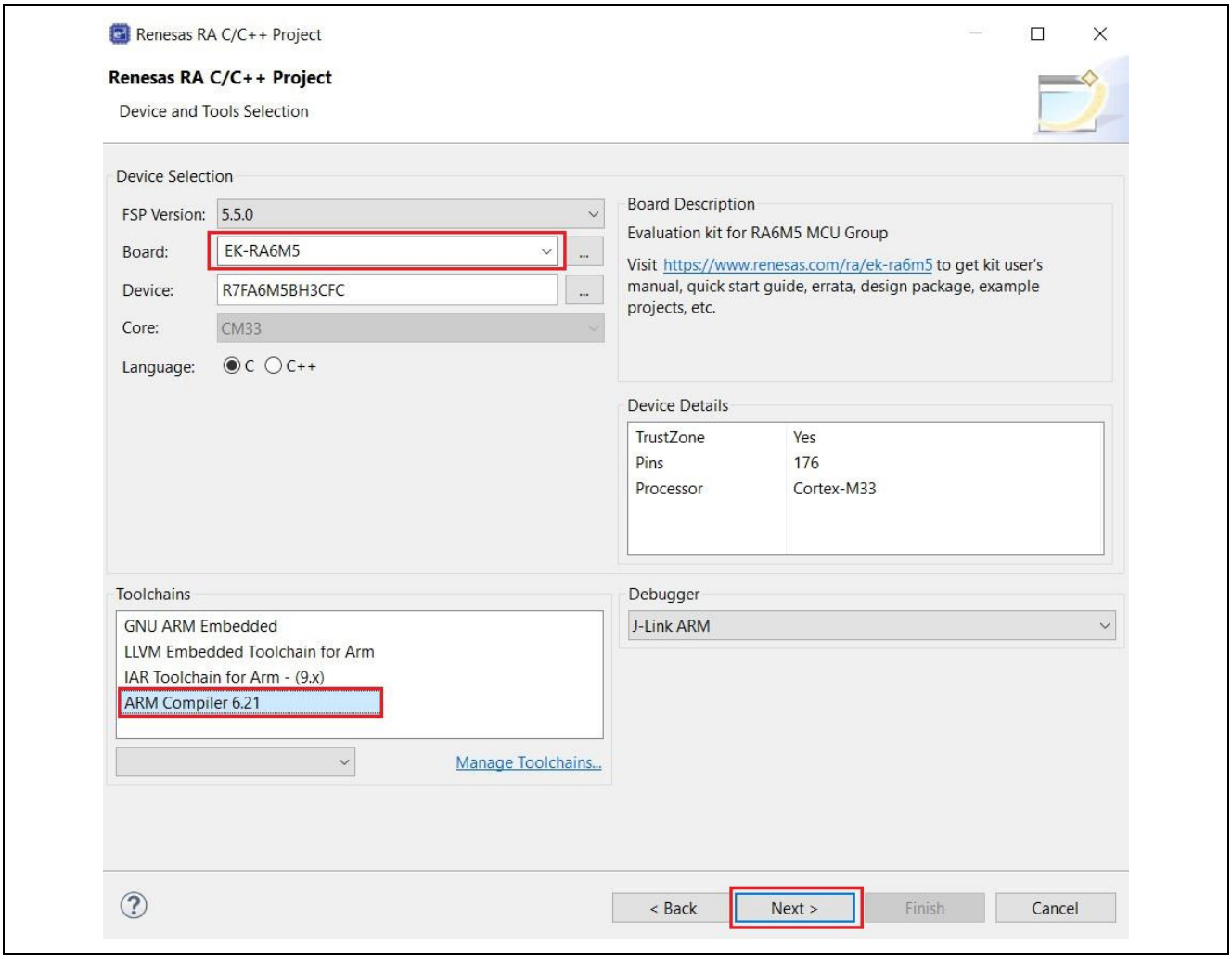

图 33. 创建基于 Arm Compiler 的工程选项

按照以下代码修改 bareborns_clock() 函数。此外，在 core_portme.h 中，根据所用 Toolchain 修改“#define COMPILER_FLAGS”。如果是 Arm 编译器，将其改为“-Omax”。

代码如下所示。

```
/* 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
```

图 34. 修改 #define COMPILER_FLAGS

在工程的 Properties-> ARM C Compiler 6.15->Miscellaneous->Other flags 中添加 “-Omax”。

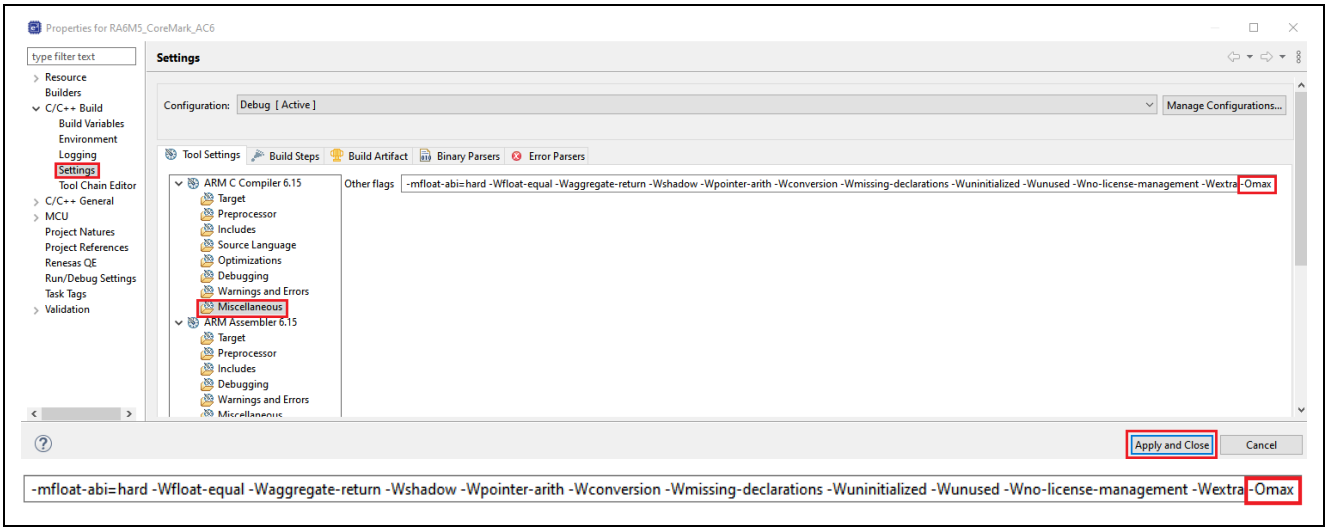

图 35. 在工程设置中添加“-Omax”选项

在工程的 Properties-> ARM Linker 6.15->Miscellaneous->Other flags 中添加“--lto”。

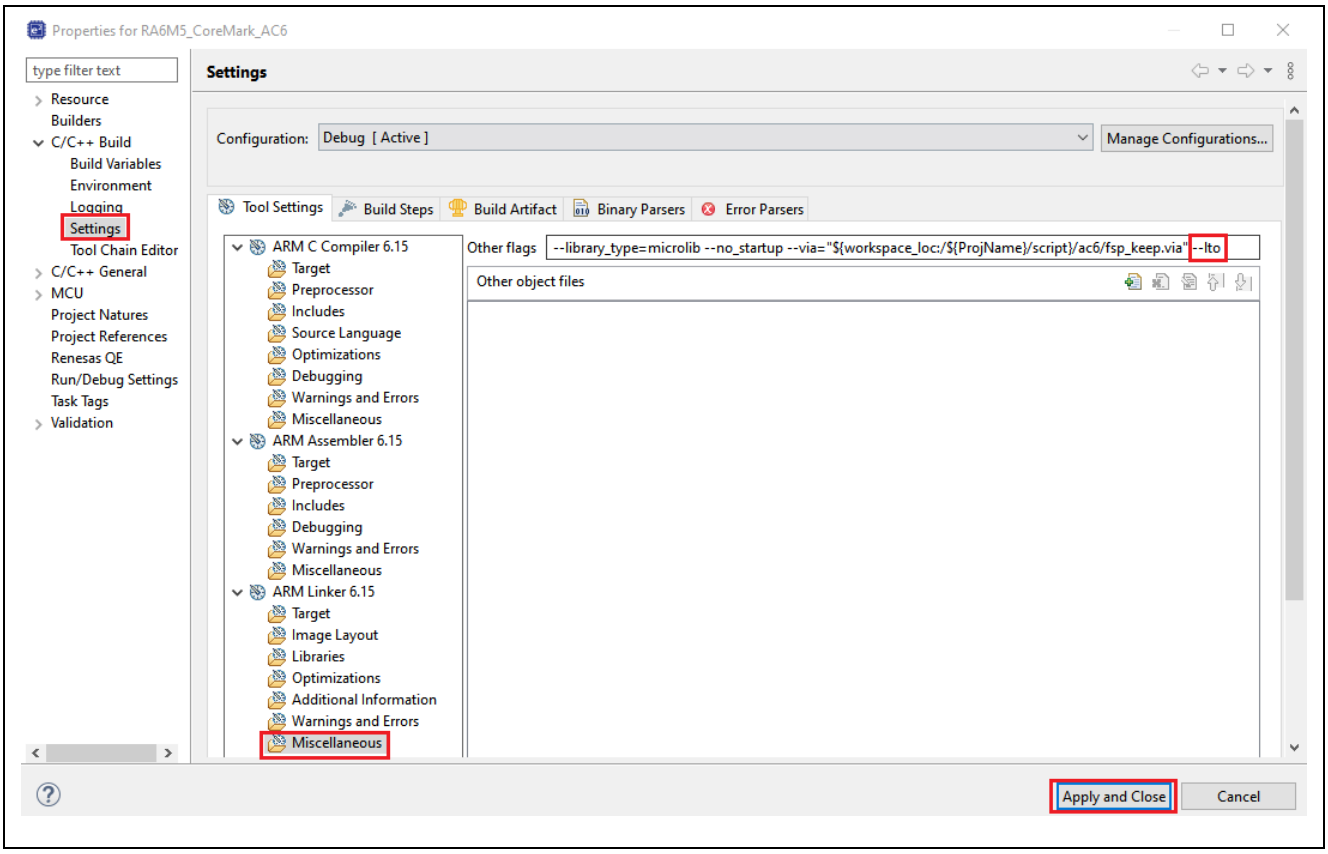

图 36. 在工程设置中添加“--lto”选项

2.8 运行 CoreMark 工程

2.8.1 电路板设置

EK-RA6M5 评估套件中的某些跳线开关设置必须在运行与本应用说明相关的工程之前进行配置。除了这些跳线开关设置，主板还包含一个 USB 调试端口和连接器，用于连接 J-Link®编程接口。

表 1 EK-RA6M5 的跳线开关设置

跳线开关	设置
J8	引脚 1-2 上的跳线
J9	打开

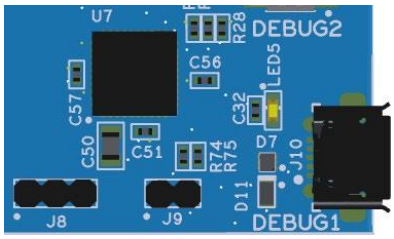

图 37. EK-RA6M5 上的 J8 和 J9

下图为 EK-RA6M5 评估套件图。

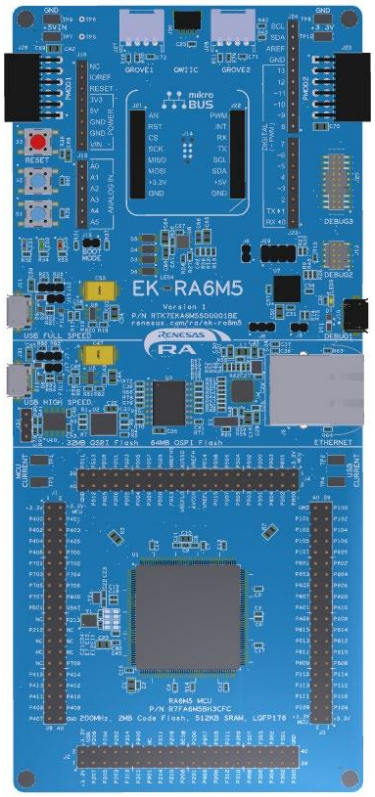

图 38. EK-RA6M5

使用 USB 线将 PC 连接到评估板上标有“Debug1”的调试端口。

2.8.2 添加运行命令以打印基准测试结果

在调试配置中，添加以下指令：dprintf portable_fini,“%s”, uart_buffer。

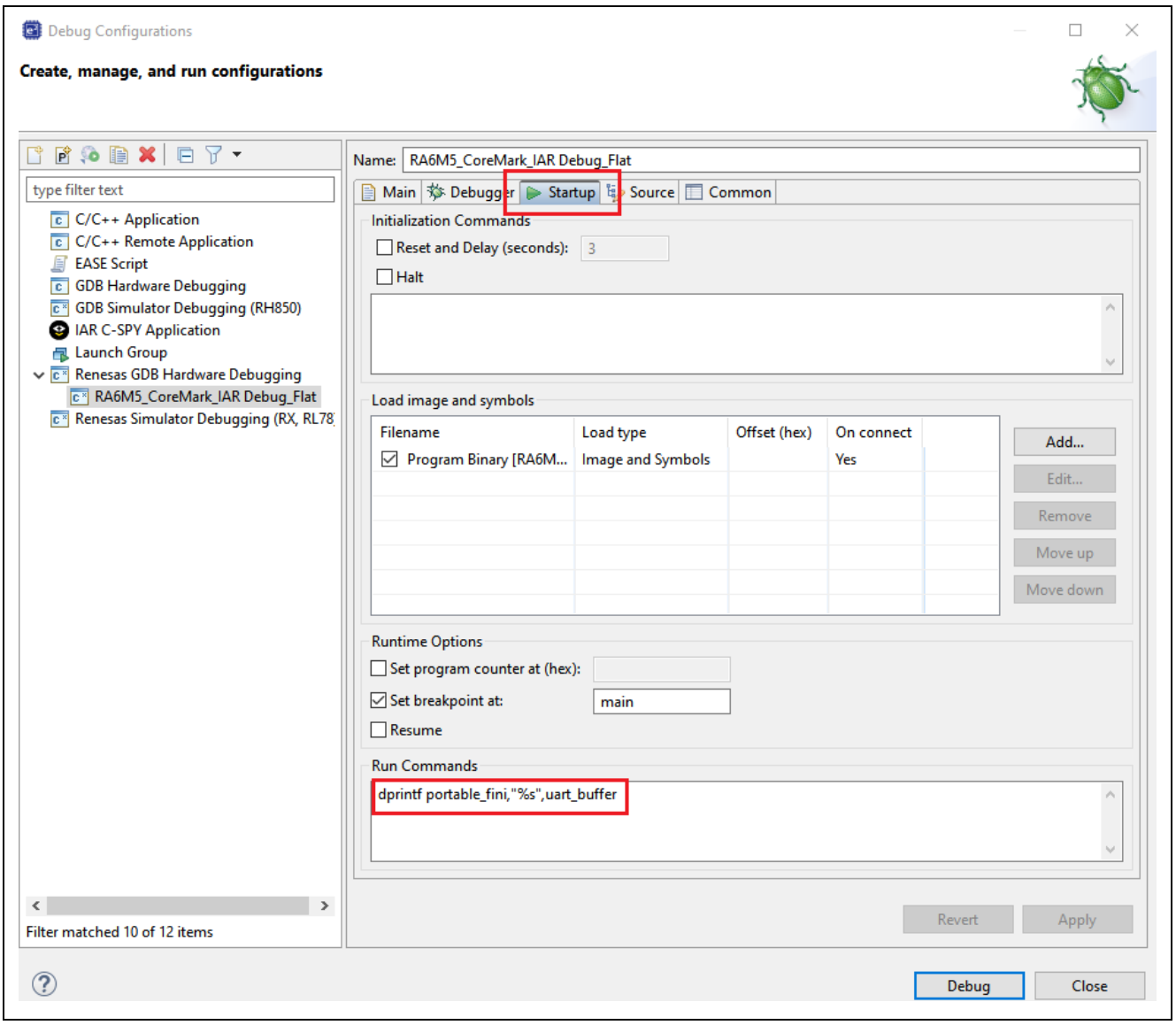

图 39. 添加 dprintf 指令

2.8.3 运行 e² studio 工程

工程创建成功后，可以使用 Renesas GDB Hardware Debugging 进行调试。在工程上右键单击 Debug As -> Renesas GDB Hardware debugging 或 “Debug Configurations…”，然后选择所需的条目。

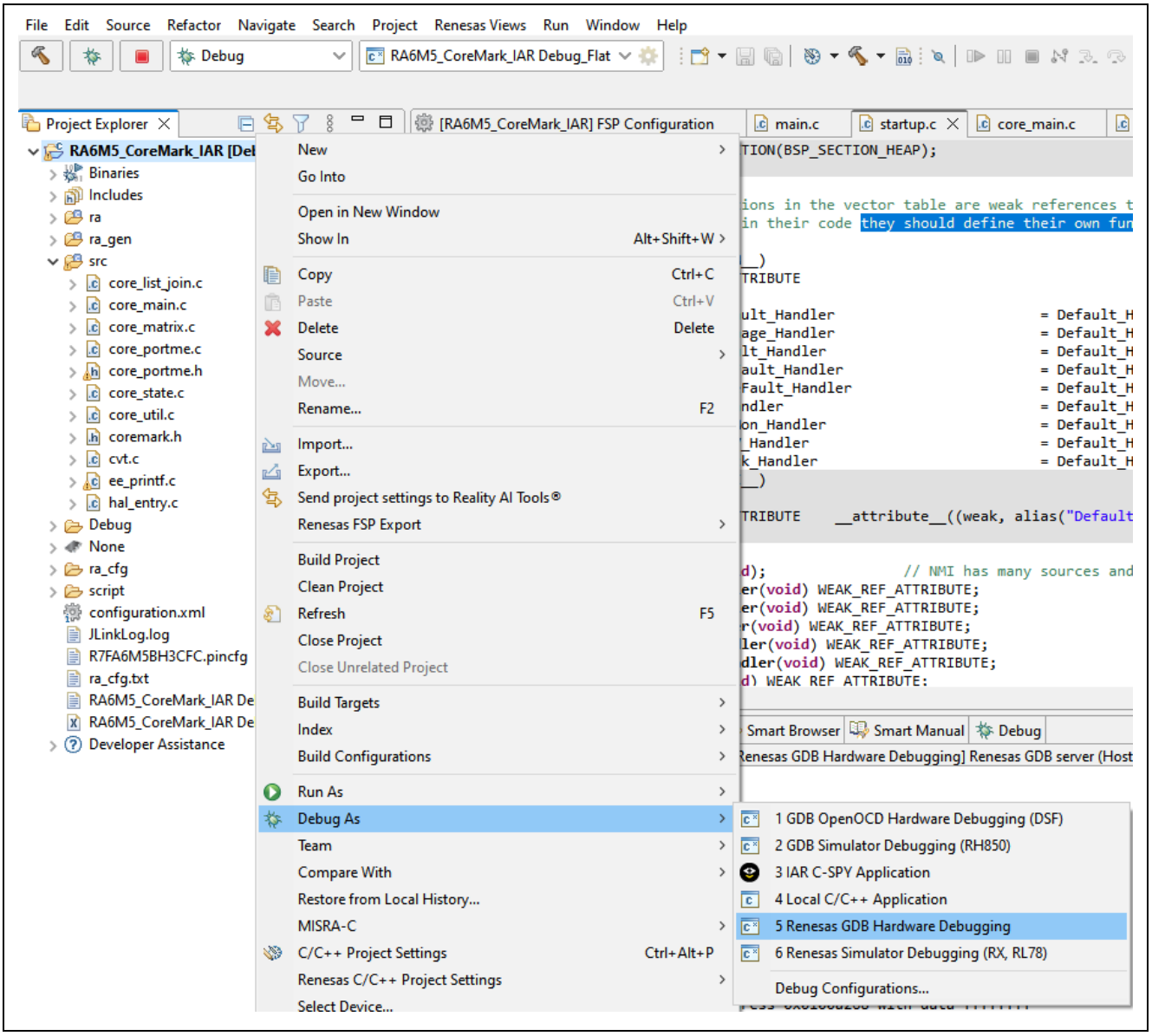

图 40. 调试工程

程序会在 Reset_Handler 处停止。

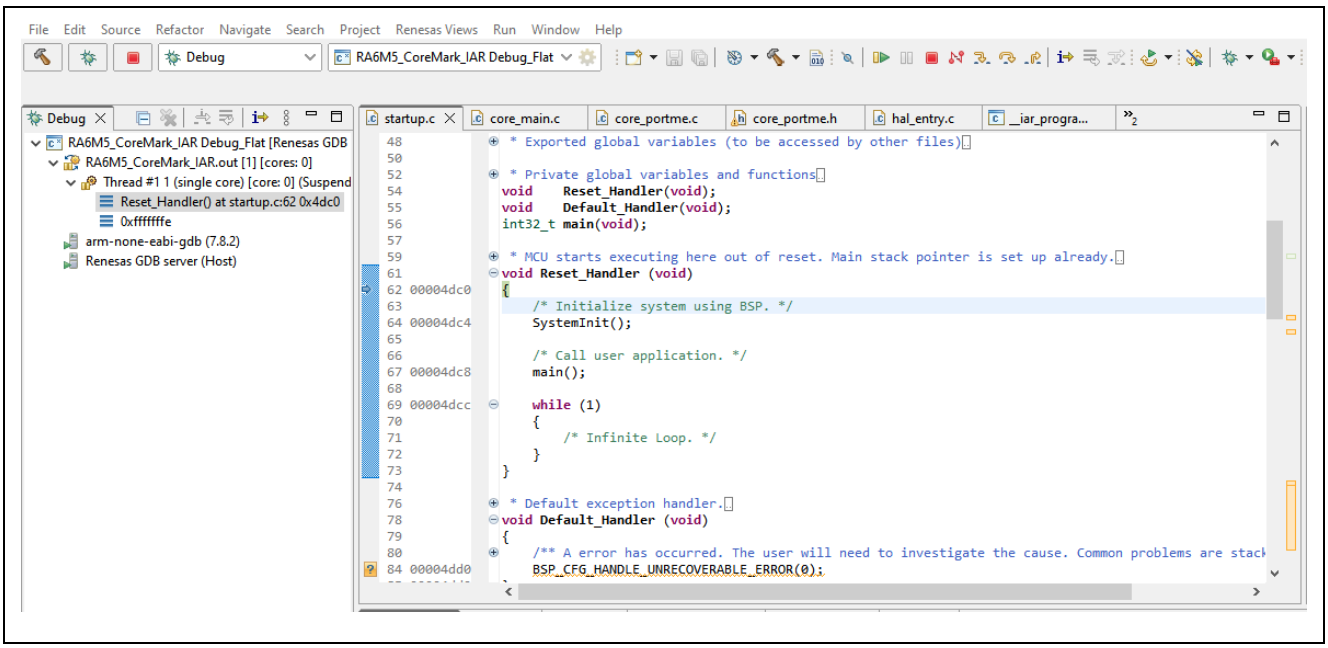

图 41. 调试工程（续）

单击 Resume，程序从 core_main.c 运行至 main，此时再次单击 Resume 运行工程。

一段时间后，程序将停在 portable_fini 中，CoreMark 评分会显示在 Debugger Console 窗口中，如下所示。

```
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
```

图 42. 通过 IAR Compiler 进行 CoreMark 评分

```
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
```

图 43. 通过 Arm Compiler 进行 CoreMark 评分

3 验证 RA 基准测试结果

您可以通过参考 EEMBC 网站上发布的 RA CoreMark 结果来验证您的结果，如下所示。

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

图 44. 发布于 EEMBC 网站的 RA CoreMark 评分

4 CoreMark 基准测试通用准则

由于搭载 Arm 处理器的目标设备可能存在多样化的存储器类型及存储架构层级，CoreMark 工程需合理配置存储器以实现高效编译。根据编译器的差异，可通过正确编辑链接脚本（linker script）或分散加载文件（scatter files）达成此目标。

鉴于 CoreMark 属于轻量级基准测试程序，需多次运行以获取可复现的评分结果。Arm 建议执行两次验证性运行（validation runs），随后进行至少十次性能分析运行（profile runs）。最终结果可通过计算性能分析运行的平均值得出。上述步骤旨在最大程度降低因处理器状态不一致导致的评分波动。

5 参考文献

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

网站和支持

可从以下链接了解更多关于 RA 系列的要点，下载组件和相关文档，以获得支持。

RA 产品	www.renesas.com/ra
RA 产品支持论坛	www.renesas.com/ra/forum
RA FSP	www.renesas.com/FSP
瑞萨技术支持	www.renesas.com/support

更新履历

版本号	日期	描述	
		页数	概述
1.00	2023 年 3 月 20 日	–	初版发布
1.10	2024 年 9 月 11 日	–	升级为 FSP v5.5.0

注意

1. 本文档中电路、软件和其他相关信息的描述仅用于说明半导体产品的操作和应用示例。用户应对产品或系统设计中电路、软件和信息的纳入或任何其他用途承担全部责任。对于您或第三方因使用这些电路、软件或信息而引起的任何损失和损害，Renesas Electronics 不承担任何责任。
2. Renesas Electronics 特此声明，对于因使用本文档中所述的 Renesas Electronics 产品或技术信息（包括但不限于产品数据、图纸、图表、程序、算法和应用示例）而引起的侵权或与第三方有关的专利、版权或其他知识产权的任何其他索赔，概不承担任何责任和赔偿。
3. 对 Renesas Electronics 或其他公司的任何专利、版权或其他知识产权均不授予任何明示、暗示或其他形式的许可。
4. 您应负责确定需要从任何第三方获得哪些许可，并在需要时为合法进口、出口、制造、销售、使用、分销或以其他方式处置包含 Renesas Electronics 产品的任何产品获得此类许可。
5. 不得对 Renesas Electronics 产品的全部或部分进行更改、修改、复制或逆向工程。对于因更改、修改、复制或逆向工程而导致您或第三方蒙受的任何损失或损害，Renesas Electronics 不承担任何责任。
6. Renesas Electronics 产品根据以下两个质量等级进行分类：“标准”和“优质”。Renesas Electronics 每种产品的预期应用取决于产品的质量等级，具体如下所示。
   “标准”：计算机、办公设备、通信设备、测试和测量设备、视听设备、家用电器、机械工具、个人电子设备、工业机器人等
   “优质”：运输设备（汽车、火车、轮船等）；交通管制（交通信号灯）；大型通信设备；关键金融终端系统；安全控制设备等
   除非在 Renesas Electronics 数据手册或 Renesas Electronics 其他文档中明确指定为高可靠性产品或用于恶劣环境的产品，否则 Renesas Electronics 产品不适合或不授权用于可能对人类生命构成直接威胁或造成人身伤害（人造生命支持设备或系统；手术植入物等），或者可能造成严重的财产损失（空间系统、海底中继器、核动力控制系统、飞机控制系统、关键设备系统、军事装备等）的产品或系统。对于因使用任何与 Renesas Electronics 数据手册、用户手册或其他 Renesas Electronics 文档不一致的 Renesas Electronics 产品而引起的您或任何第三方所造成的任何损坏或损失，Renesas Electronics 不承担任何责任。
7. 没有任何半导体产品是绝对安全的。尽管 Renesas Electronics 的硬件或软件产品中可能实施了任何安全措施或功能，Renesas Electronics 对因任何漏洞或侵骇（包括但不限于以任何未经授权的方式访问或使用 Renesas Electronics 产品或使用 Renesas Electronics 产品的系统）而产生的任何后果概不负责。RENESAS ELECTRONICS 不担保或保证 RENESAS ELECTRONICS 产品或使用 RENESAS ELECTRONICS 产品创建的任何系统不会被破坏，或者可免于数据损坏、攻击、病毒、干扰、黑客攻击、数据丢失或失窃或其他安全入侵（“漏洞问题”）。RENESAS ELECTRONICS 不承担由任何漏洞问题引起的或与之相关的任何和所有责任或义务。此外，在适用法律允许的范围内，RENESAS ELECTRONICS 不对本文件和任何相关或附带的软件或硬件提供任何和所有明示或暗示的保证，包括但不限于对适销性或对特定用途的适用性的暗示保证。
8. 使用 Renesas Electronics 产品时，请参见最新的产品信息（数据手册、用户手册、应用笔记、可靠性手册中的“处理和使用半导体器件的一般说明”等），并确保使用条件符合 Renesas Electronics 在最大额定值、工作电源电压范围、散热特性和安装等方面的规定。对于因在超出上述规定范围的条件下使用 Renesas Electronics 产品而引起的任何失常、故障或事故，Renesas Electronics 不承担任何责任。
9. 尽管 Renesas Electronics 致力于提高 Renesas Electronics 产品的质量和可靠性，但半导体产品具有特定的特性，例如在特定速率下发生故障以及在某些使用条件下出现故障。除非在 Renesas Electronics 数据手册或 Renesas Electronics 其他文档中指定为高可靠性产品或用于恶劣环境的产品，否则 Renesas Electronics 的产品将不受抗辐射设计的约束。用户应负责采取安全措施，以防止人身伤害、火灾造成的伤害，和/或因 Renesas Electronics 产品发生故障或失常而对公众造成的危险，例如硬件和设备的安全设计，包括但不限于冗余、火控和故障预防、针对老化退化的适当处理或任何其他适当的措施。由于对微型计算机软件进行评估非常困难且无实操性，因此用户有责任评估自己生产的最终产品或系统的安全性。
10. 请联系 Renesas Electronics 销售办事处，以获取有关环境事宜的详细信息，例如每个 Renesas Electronics 产品的环境相容性。用户有责任认真、充分地研究有关纳入或使用受控物质的适用法律和法规（包括但不限于欧盟 RoHS 指令），并按照所有适用法律和法规使用 Renesas Electronics 产品。对于因您未遵守适用的法律和法规而造成的损坏或损失，Renesas Electronics 不承担任何责任。
11. Renesas Electronics 产品和技术不得被用于或纳入为任何适用的本国或外国法律、法规所禁止制造、使用或销售的产品或系统范围内。用户应遵守由对当事方或交易拥有管辖权的任何国家/地区的政府颁布和管理的任何可适用的出口控制法律和法规。
12. 应由Renesas Electronics 产品的购买方或分销商，或者对产品进行分发、处置或以其他方式出售或转让给第三方的任何其他当事方，负责将本文档中阐明的内容和条件提前通知前述第三方。
13. 未经 Renesas Electronics 事先书面同意，不得以任何形式全部或部分重印、再现或复制本文档。
14. 如果对本文档中包含的信息或 Renesas Electronics 产品有任何疑问，请联系 Renesas Electronics 销售办事处。

（注 1）本文档中的“Renesas Electronics”是指 Renesas Electronics Corporation，也包括其直接或间接控制的子公司。
（注 2）“Renesas Electronics 产品”是指 Renesas Electronics 开发或制造的任意产品。

（版本 5.0-1 2020 年 10 月）

公司总部

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

联系信息

有关产品、技术、文档最新版本或离您最近的销售办事处的更多信息，请访问：www.renesas.com/contact/。

商标

Renesas 和 Renesas 徽标是 Renesas Electronics Corporation 的商标。所有商标和注册商标都是各自所有者的财产。

© 2025 Renesas Electronics Corporation。保留所有权利。