AN13072

LPC55S1x/LPC551x CoreMark Porting Guide

Rev. 0 — November, 2020 Application Note

by: NXP Semiconductors

Contents

1 Introduction

CoreMark, developed by EEMBC, is a simple and yet sophisticated benchmark. It is designed specifically to test the functionality of an embedded processor core. Running CoreMark produces a single-number score allowing users to make quick comparisons between processors.

LPC55S1x/LPC551x is an Arm® Cortex®-M33 based microcontroller for embedded applications. These devices include:

- Up to 96 KB of on-chip SRAM, up to 256 KB on-chip flash
- PRINCE module for on-the-fly flash encryption/decryption
- CASPER Crypto/FFT engine
- High-speed and full-speed USB host and device interface with crystal-less operation for full-speed
- One SDIO/MMC
- One CAN-FD
- Five general-purpose timers, one SCTimer/PWM, one RTC/alarm timer
- One 24-bit Multi-Rate Timer (MRT)
- A Windowed Watchdog Timer (WWDT)
- Nine flexible serial communication peripherals (which can be configured as a USART, SPI, high speed SPI, I^2C, or I^2S interface)
- Programmable Logic Unit (PLU)
- One 16-bit 1.0 Msamples/sec ADC, comparator, and temperature sensor

The Cortex-M33 offers 18.2% performance increase in the same process technology compared to the high embedded performance bars already established by Cortex-M4 processors, while improving power efficiency. Cortex-M33 official CoreMark is 4.02 CoreMark/MHz, Cortex-M4 official CoreMark is 3.40 CoreMark/MHz.

This application note describes how to port CoreMark code to LPC55S1x/LPC551x, which involves setting up software and hardware including memory partitioning, compiler setting, and board setup. It also describes how to measure CoreMark scores on the Cortex-M33 and the result including CoreMark scores and power consumption in µA/MHz. Separate CoreMark projects for different software development tools, Keil MDK, IAR EWARM and MCUXpresso IDE, are also included here for reference.

2 Integration of CoreMark library to SDK2.6 framework

The software package associated with this application note contains SDK2.6 based project framework that allows developers to drop in the CoreMark library sources and quickly get up and running with benchmarking the LPC55S1x/LPC551x. To get started, go to CoreMark. Click the download link as shown in Figure 1, and follow the instructions on that page.

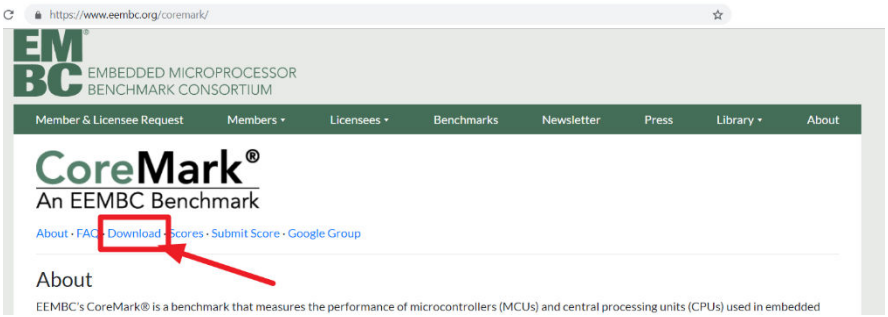

Figure 1. EEMBC CoreMark download link

After reviewing the license terms, look through the readme and documentation file. The readme gives step-by-step instructions on unpacking and building the distribution. This will also help with getting familiar with the CoreMark terminology used throughout the application note.

2.1 Port CoreMark library into CoreMark framework

There are four variants of CoreMark projects in this application note for each IDE. Two execute the CoreMark application from internal flash and the others execute the CoreMark application from internal SRAMX.

The various CoreMark projects are:

1. `coremark_score_on_flash`: Executes CoreMark application from internal Flash.
2. `coremark_score_on_sramx`: Executes CoreMark application from internal RAM.
3. `coremark_uAMHz_on_flash`: Measurement current when Coremark execute on Flash.
4. `coremark_uAMHz_on_sramx`: Measurement current when Coremark execute on RAM.

The CoreMark projects are found in the following locations:

- Keil MDK IDE:

  /lpc55s1x_coremark_mdk\lpc55s1x_coremark_mdk.uvprojx

- IAR Workbench IDE:

  /lpc55s1x_coremark_iar\lpc55s1x_coremark_iar.eww

Each of executes settings have three frequency settings: 12 MHz (FRO), 96 MHz (FRO), 100MHz (PLL) and 150MHz (PLL).

Depending on the toolchain, the workspace are as shown in following figures. The CoreMark framework requires the addition of the CoreMark files from EEMBC.

2.1.1 Coremark framework for Keil MDK/IAR EWARM/MCUXpresso IDE

The `lpc55s1x_coremark_xxx` project must be set as active before the CoreMark source code files can be added.

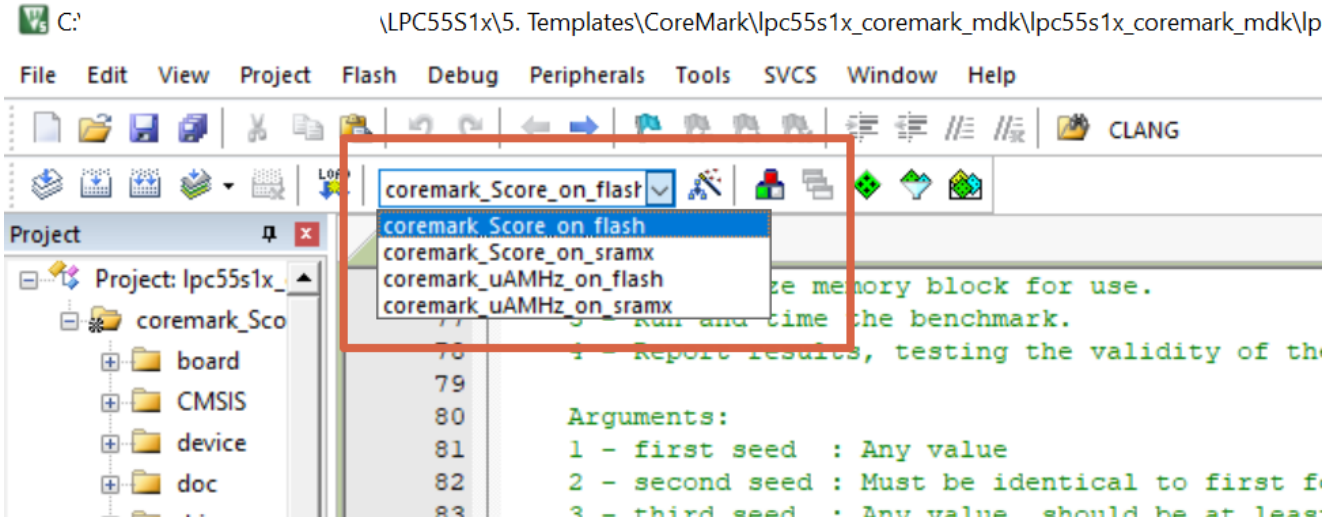

Figure 2. Keil MDK CoreMark project configuration select

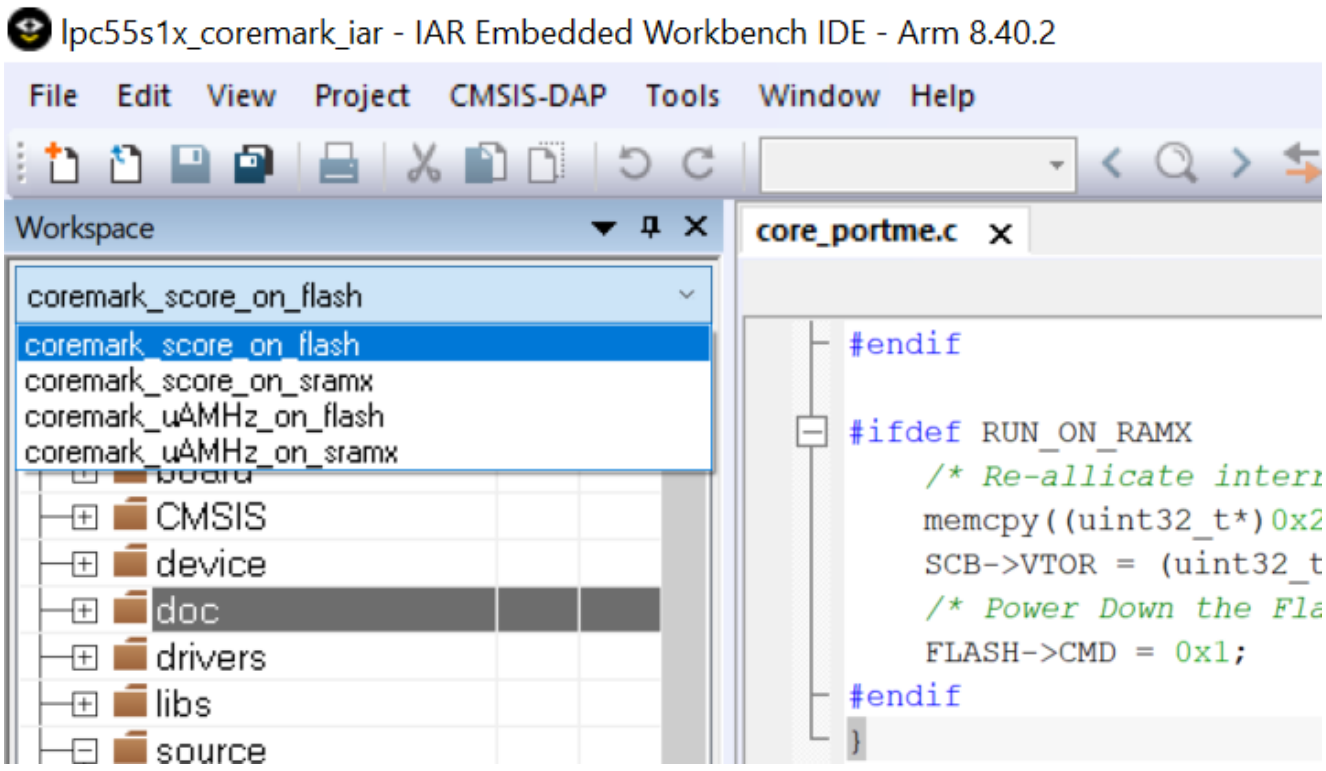

Figure 3. IAR EWARM workspace

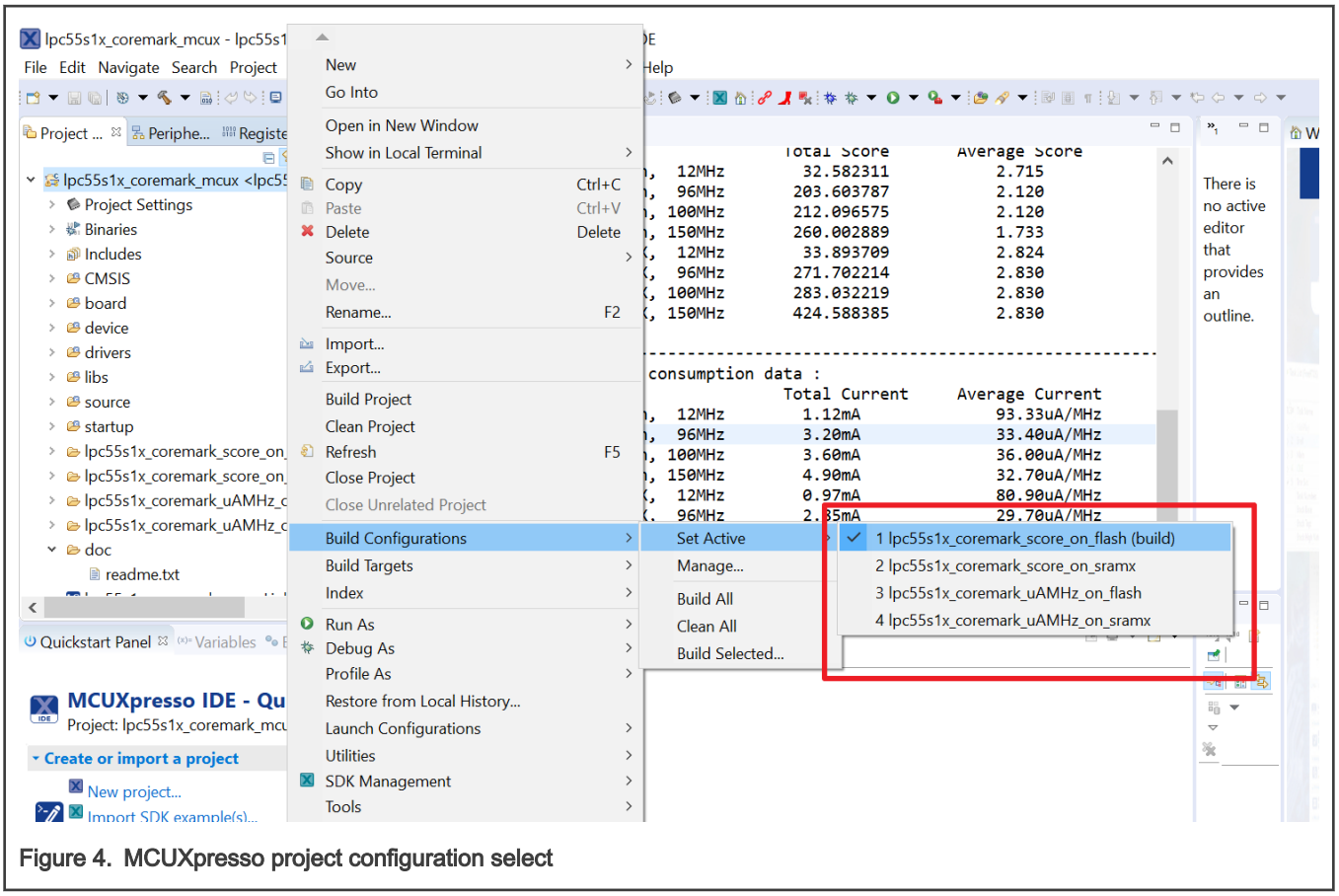

Figure 4. MCUXpresso project configuration select

Copy the following files from the CoreMark package downloaded from EEMBC.

- `core_list_join.c`
- `core_main.c`
- `core_matrix.c`
- `core_state.c`
- `core_util.c`
- `coremark.h`

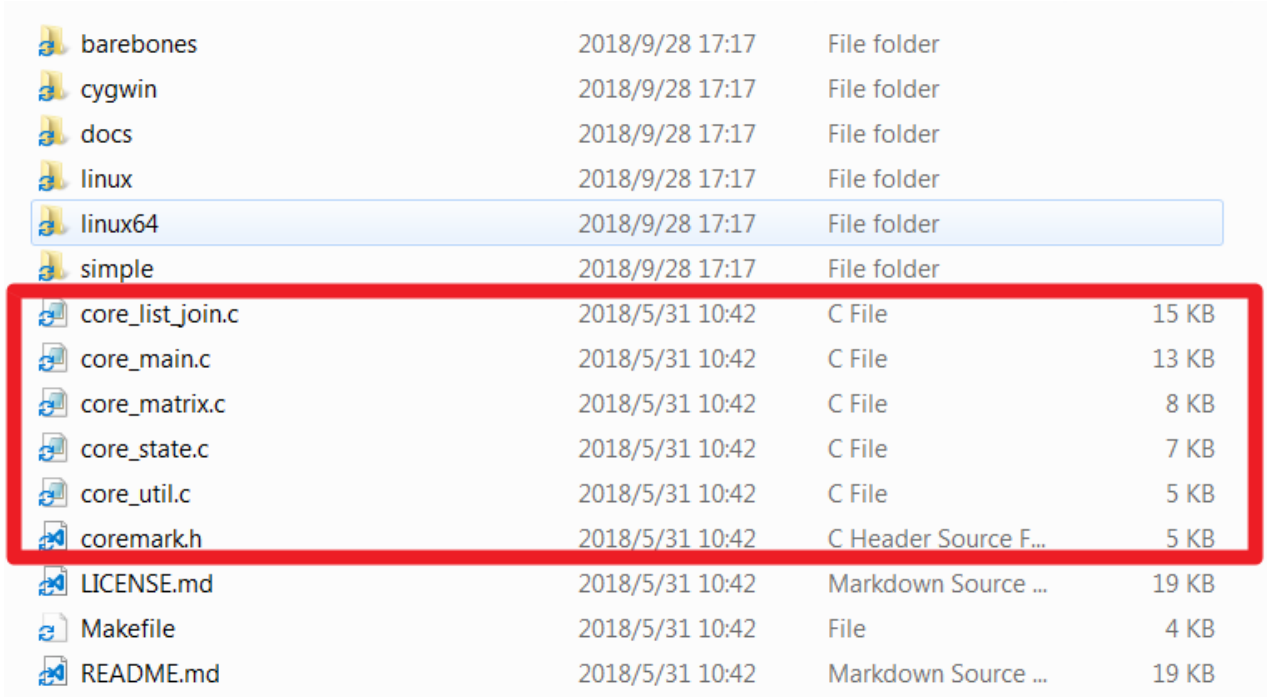

Figure 5. CoreMark files

- For Keil MDK, place these files in the project directory of *lpc55s1x_coremark_mdk\source*.
- For IAR Embedded Workbench, place these files in the project directory of *lpc55s1x_coremark_iar\source*.
- For MCUXpresso, place these files in the project directory of *lpc55s1x_coremark_mcux\source*.

The files, *ee_printf.c*, *core_portme.c*, and *core_portme.h* (under the *port_lpc5500* folder, need to be copied to the following folder locations.

- For Keil IDE:
  - Place the files in the *lpc55s1x_coremark_mdk\source\port_lpc5500*.
  - Add the files into the Keil MDK project framework to the respective groups source by double clicking on the groups.
- For IAR Embedded workbench:
  - Place the files in the *lpc55s1x_coremark_iar\source\port_lpc5500*.
  - Add the files into the IAR project framework to the respective groups source by double clicking on the groups.
- For MCUXpresso:
  - Place the files in the *lpc55s1x_coremark_mcux\source\port_lpc5500*
  - Add the files into the MCUXpresso project framework to the respective groups source by click the **Refresh** selection.

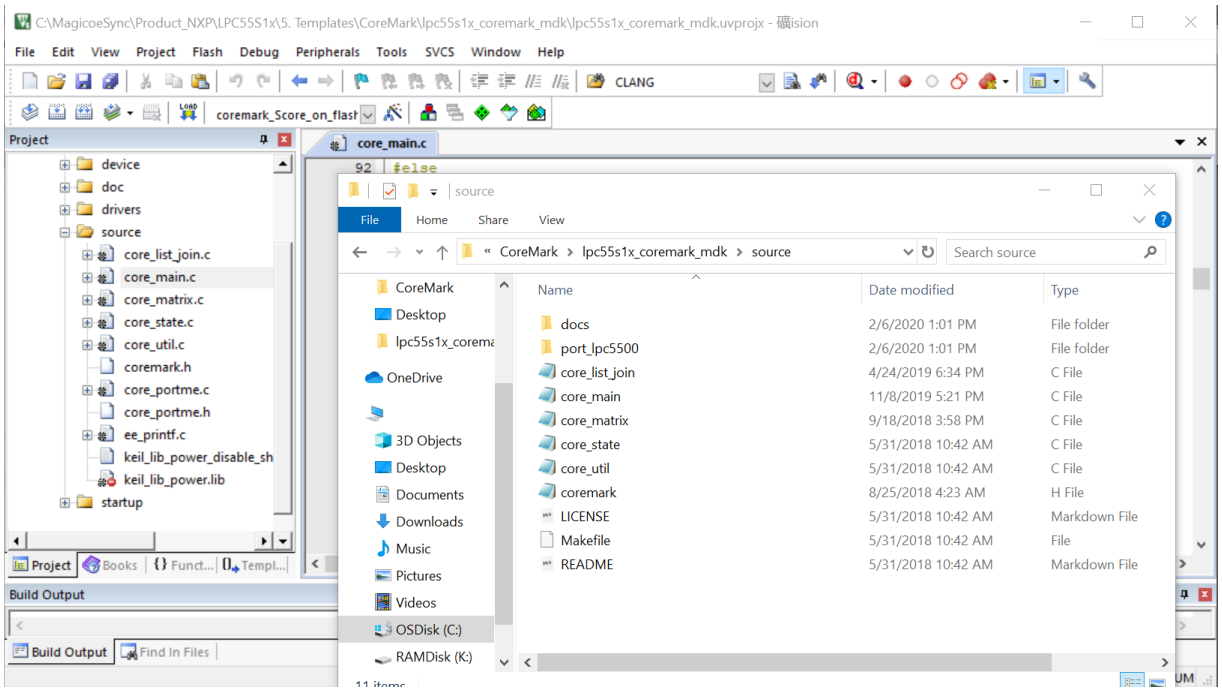

Figure 6. Adding files in Keil MDK

- For KEIL MDK project:
  - Right-click the source folder and select **Add** and then **Add Files...**.

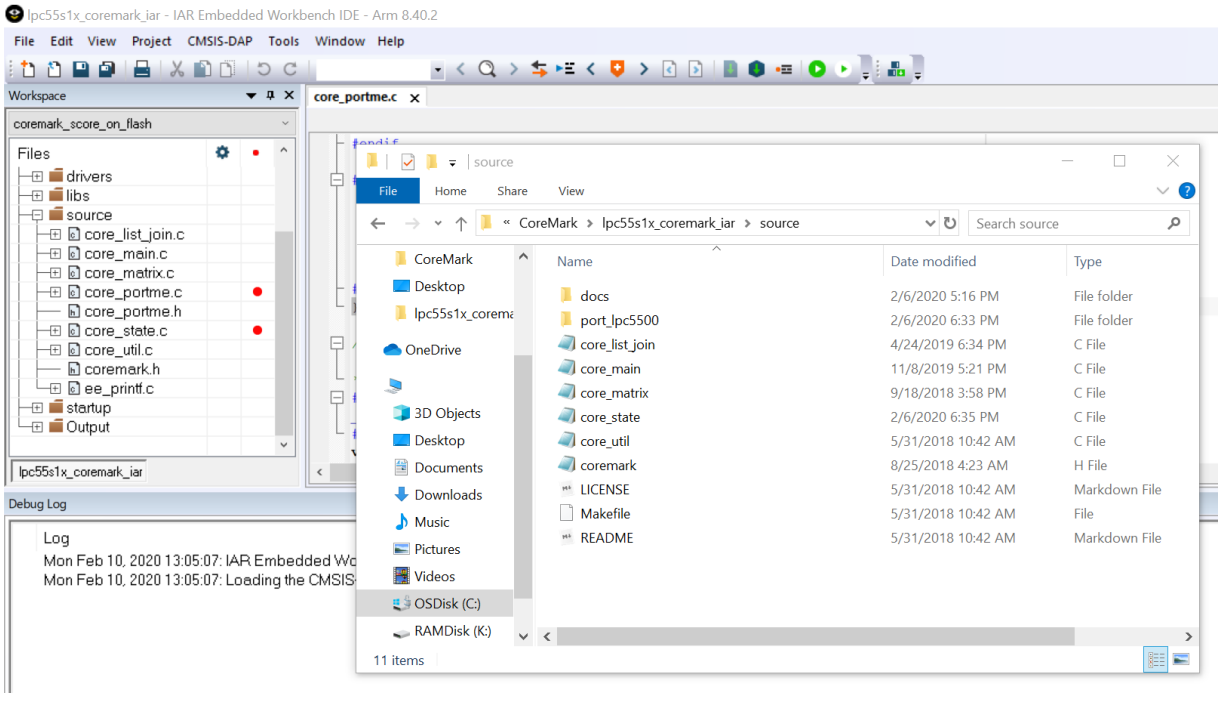

Figure 7. Adding Files in IAR EWARM workspace

- For IAR Embedded workbench:
  - Right-click the source folder and select **Add** and then **Add Files...**.

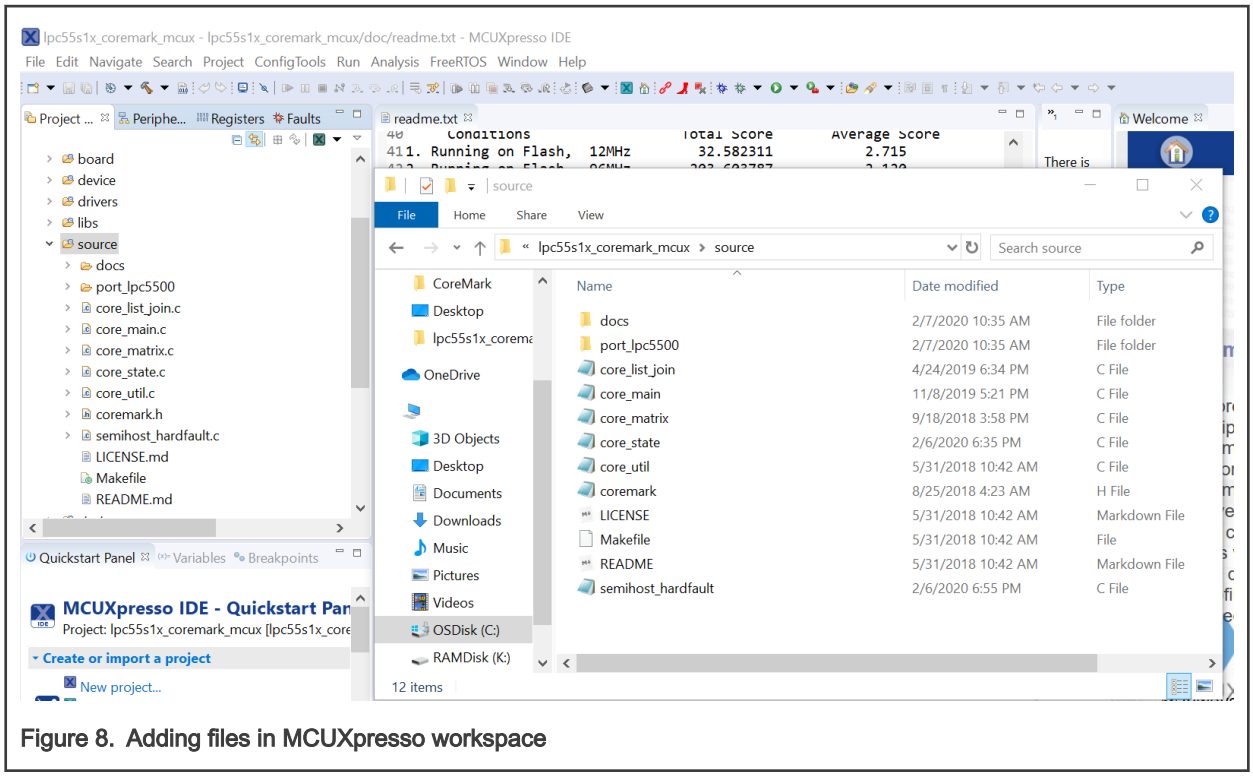

Figure 8. Adding files in MCUXpresso workspace

- For MCUXpresso project:
  - Copy the files into the *source* folder and then click the **Refresh**. The files will be added in project automatically.

Use the *core_portme.c* and *core_portme.h* files provided with the application note and not the one from the EEMBC CoreMark package. For convenience these files have the required porting changes ready for use.

Copy these files to the source folder for all three tool chains and add the *core_portme.c* file in the project framework under the source group.

A few files need to be modified to support CoreMark and are described below.

In the project scatter file change the stack size as `0x1000`.

```
define symbol __size_cstack__        = 0x1000;
define symbol __size_heap__          = 0x1000;
```

To add the path to the header files used in the project:

- In Keil MDK, under **Project** -> **Options** -> **C/C++(AC6)**, click **Include path** and add the following paths that contain the header files.

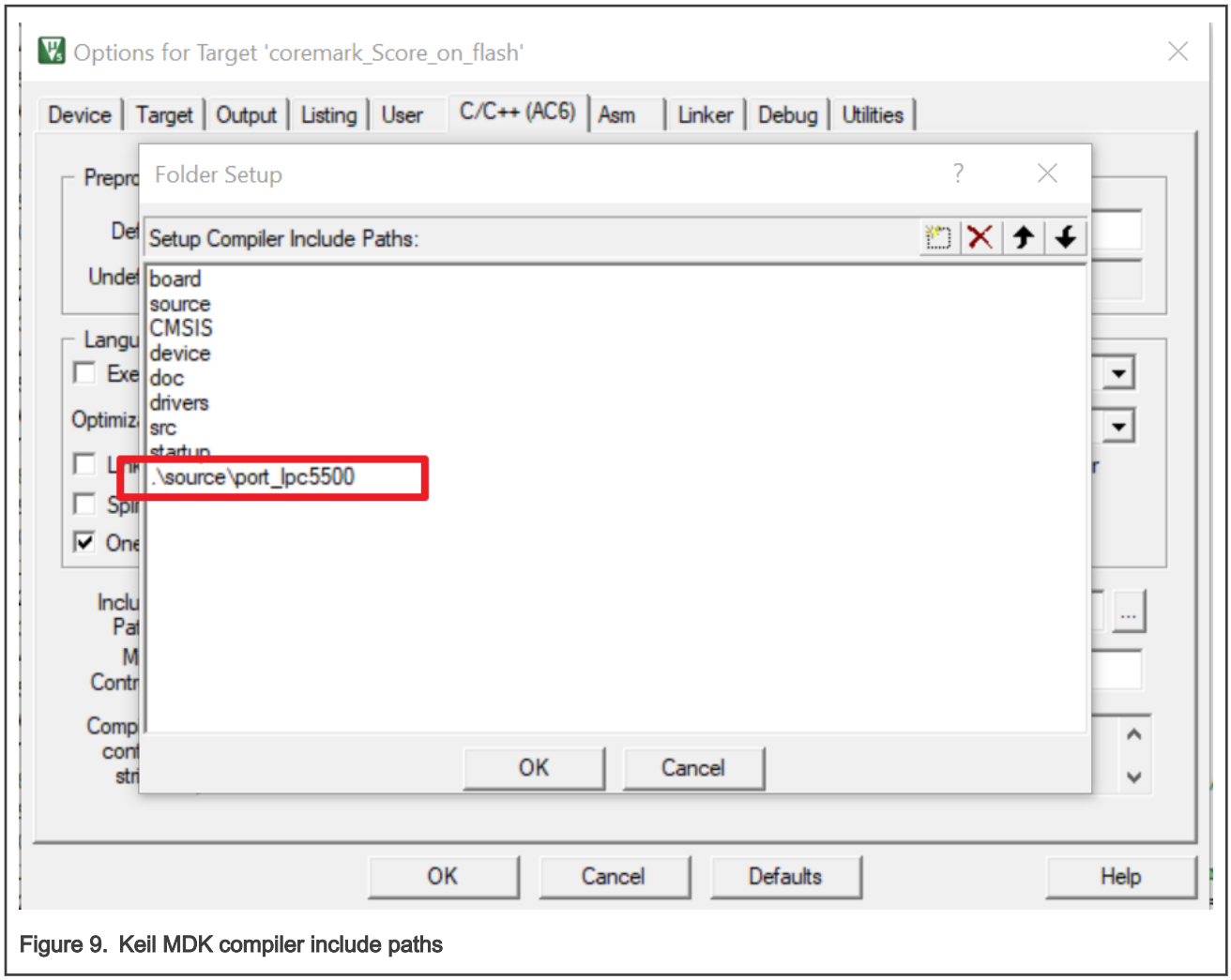

Figure 9. Keil MDK compiler include paths

- In IAR, under **Project** -> **Options** -> **C/C++ Compiler**, click **Preprocessor** and add the following paths that contains the header files.

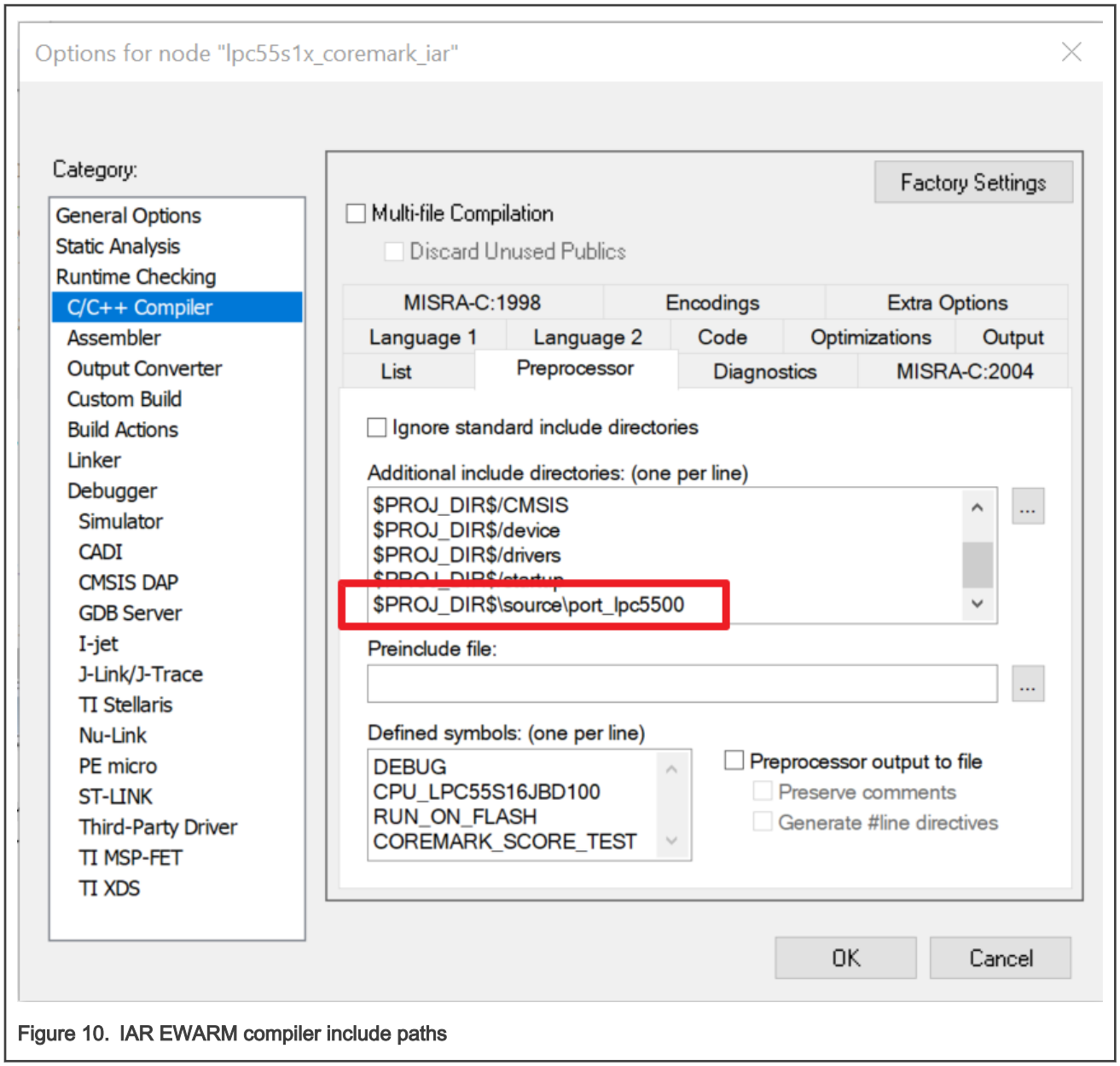

Figure 10. IAR EWARM compiler include paths

- In MCUXpresso, under **Properties for xxxx** -> **C/C++ Build** -> **Settings**, click **Includes** and add the following paths that contains the header files.

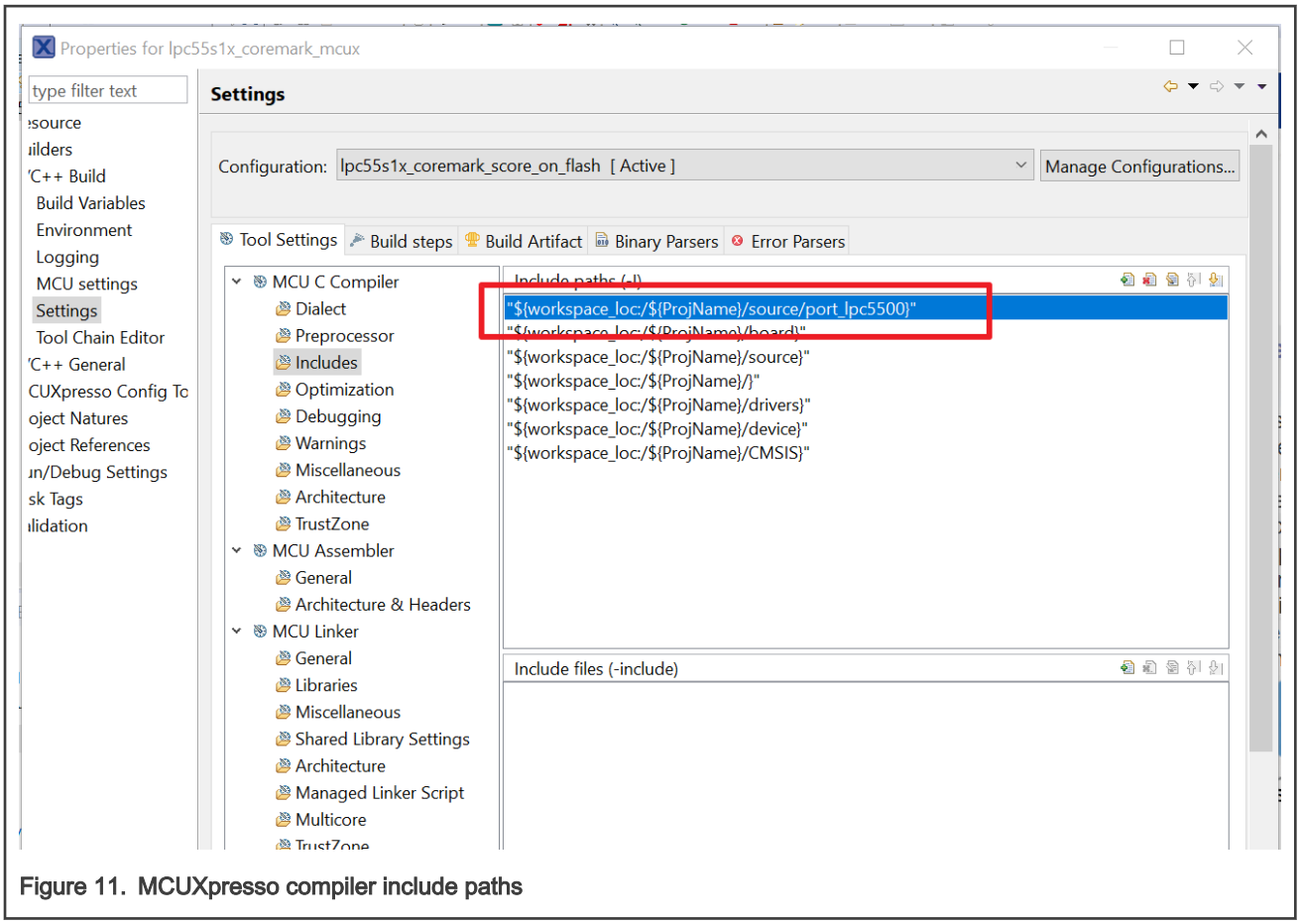

Figure 11. MCUXpresso compiler include paths

Now, the CoreMark files have been successfully ported into the CoreMark project framework.

2.1.2 CoreMark framework to execute from Internal SRAM

The project, *lpc55s1x_coremark_xxx_on_sramx*, executes the CoreMark application from 16 KB SRAMX memory region.

The files, *core_list_join.c*, *core_main.c*, *core_matrix.c*, *core_state.c*, and *core_util.c*, are relocated to execute from SRAMX using the linker scripts.

For Keil MDK, the linker script is located at:

```
.\lpc55s1x_coremark_mdk\LPC55S16_coremark_score_sramx.scf
```

Figure 12 shows the linker script settings for `lpc55s1x_coremark_xxx_on_sramx` project.

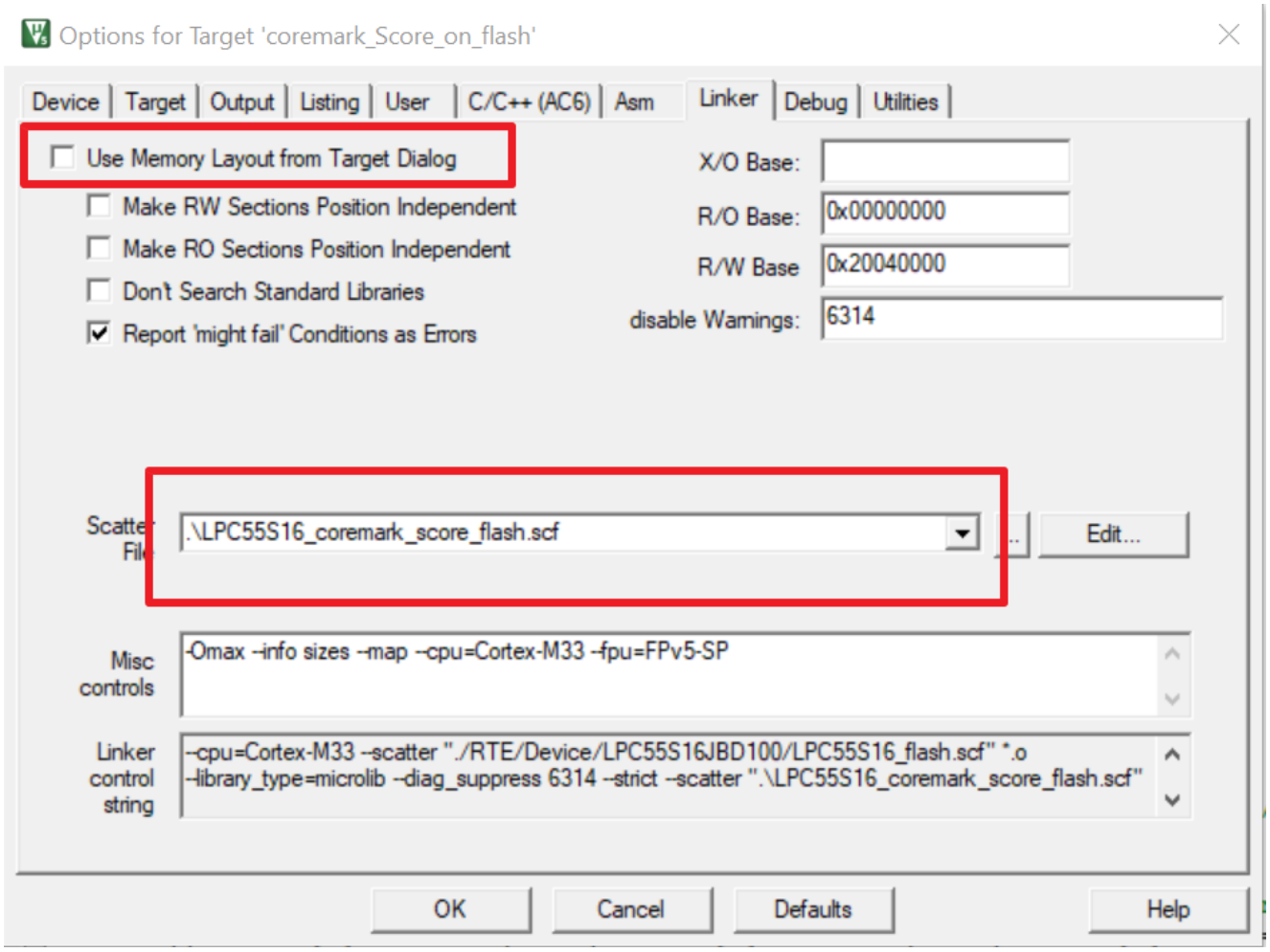

Figure 12. Linker script in Keil IDE

For IAR EWARM IDE to execute CoreMark in Internal SRAM, to place Coremark operation codes into RAM section, add the following line of code in the *icf* file, as shown in Figure 13.

```
initialize by copy { readwrite, section .textrw };
do not initialize  { section .noinit };

if (isdefinedsymbol(__USE_DLIB_PERTHREAD))
{
  /* Required in a multi-threaded application */
  initialize by copy with packing = none { section __DLIB_PERTHREAD };
}

place at address mem: m_interrupts_start    { readonly section .intvec };
place in TEXT_region                        { readonly };
place in DATA_region                        { block RW };
place in DATA_region                        { block ZI };
place in DATA_region                        { last block HEAP };
place in CSTACK_region                      { block CSTACK };

place in XCODE_region                       { section .critical_code };
initialize by copy                          { section .critical_code };
place in XCODE_region                       { rw object core_portme.o,
                                              rw object core_main.o,
                                              rw object core_list_join.o,
                                              rw object core_matrix.o,
                                              rw object core_state.o,
                                              rw object core_util.o,
                                            };
initialize by copy                          { object core_portme.o,
                                              object core_main.o,
                                              object core_list_join.o,
                                              object core_matrix.o,
                                              object core_state.o,
                                              object core_util.o,
                                            };
```

Figure 13. IAR EWARM allocate Code to SRAM area

For MCUXpresso to execute CoreMark in Internal SRAM, select the linker file as *LPC55S16_coremark_score_sramx.ld* in **Managed Linker script**, as shown in Figure 14.

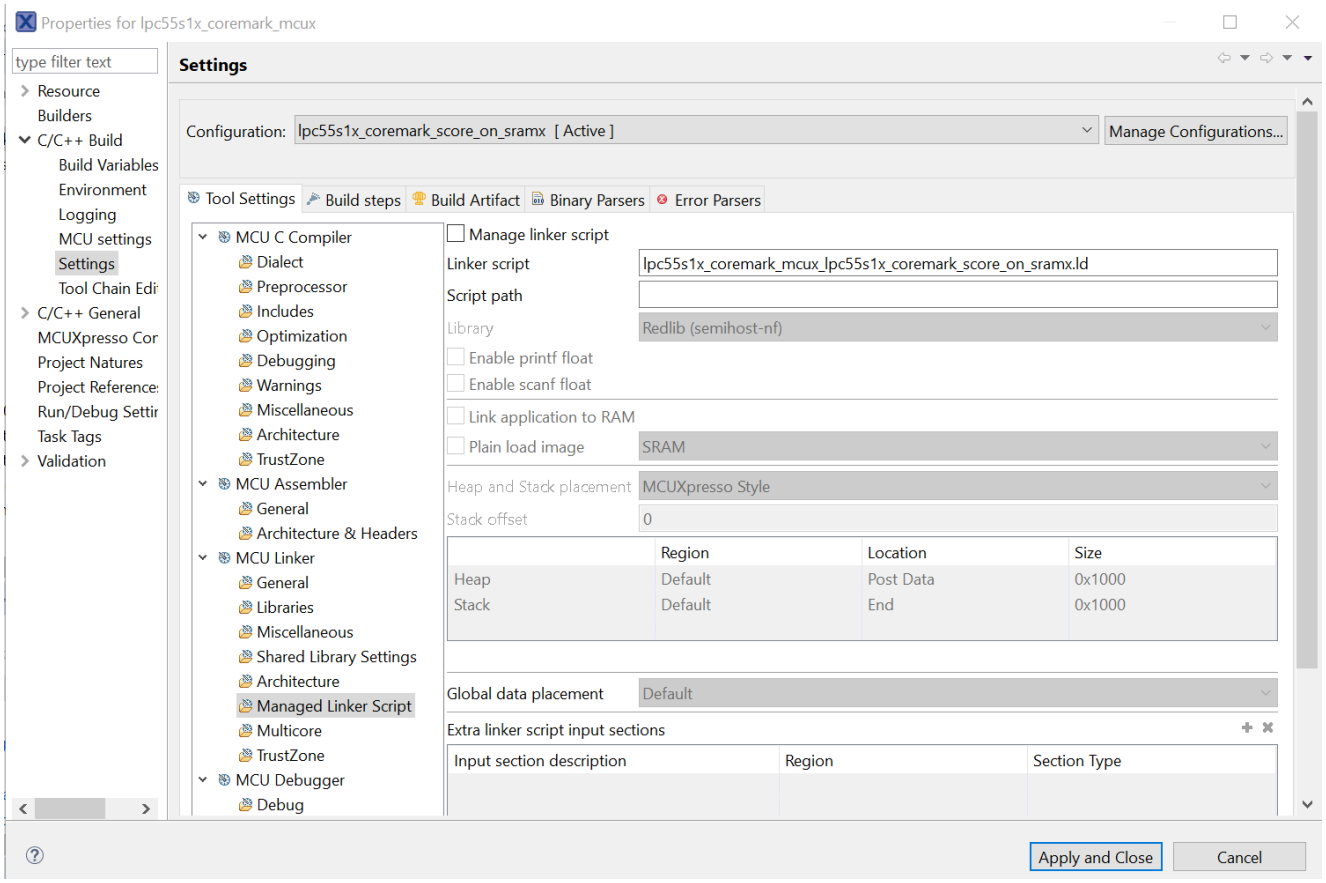

Figure 14. MCUXpresso allocate code to SRAM area

2.2 Optimizing the CoreMark framework

There are many factors that affect the CoreMark and µA/MHz score that can be optimized. Some of these factors are IDE-dependent optimizations, while others leverage the MCU architecture for better performance. The goal is to produce the best scores from all three IDEs. It is important to understand that these IDEs are constantly changing and a different version of a given IDE may add or remove features that may make these optimizations obsolete or ineffective.

The followings are the IDE versions that are applicable to this application note:

- Keil MDK v5.28
- IAR EWARM 8.40.2
- MCUXpresso 11.1.0 Build[3209]

2.2.1 Memory considerations

Due to the inherent architecture of SRAM and flash, CoreMark executes faster when running out of SRAM. The LPC55S1x/LPC551x internal memory uses a multilayer AHB matrix system that provides a separate instruction and data bus for Cortex-M33 and SRAMX bank. As shown in Figure 15, SRAM0 to SRAM2 are on System bus. Placing the CoreMark code and data in different SRAM banks minimizes bus contention and improves instruction and data parallelism.

It is important to minimize the flash wait state according to the MCU frequency to optimize the CoreMark score. In contrast, when performing the µA/MHz test, it is possible to save power by disabling the flash's prefetch ability. For more information about how to correctly configure the flash memory, such as the minimum amount of wait states allowed at a given core frequency, see *LPC55S1x/LPC551x User Manual* (document UM11295).

The provided CoreMark framework projects include separate SRAM and flash based projects that implement various memory optimizations.

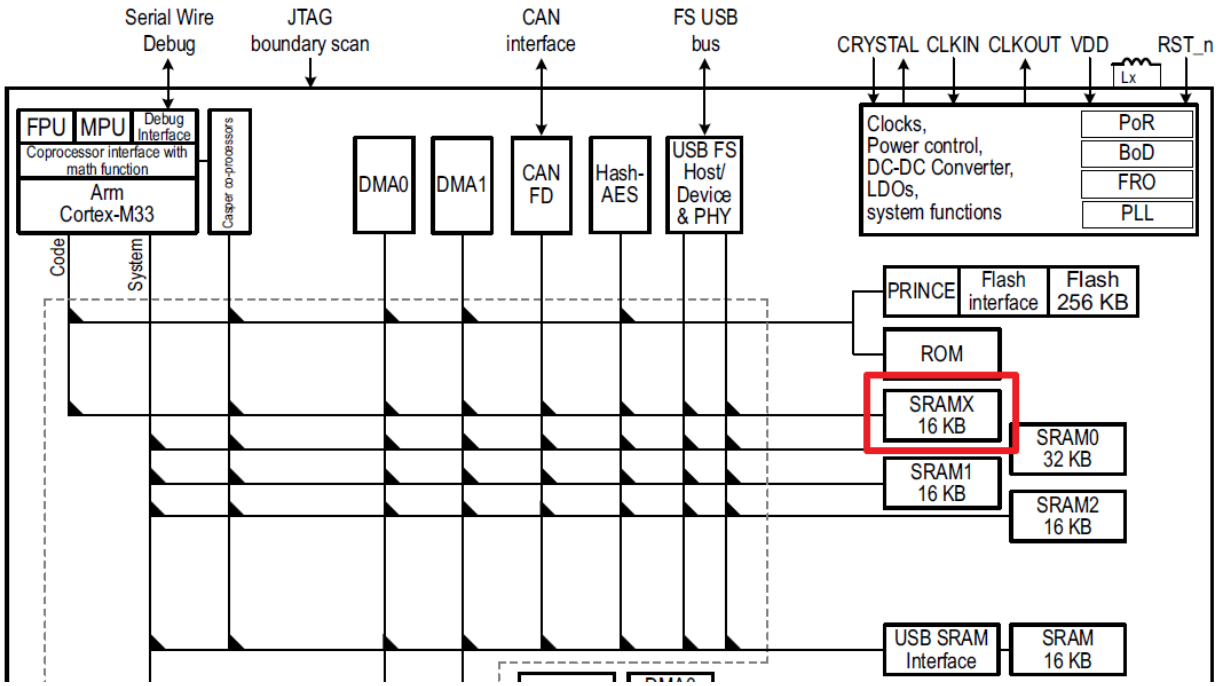

Figure 15. LPC55S1x/LPC551x AHB matrix

In both the SRAM and flash projects, there is a **COREMARK_SCORE_TEST** macro defined in *core_portme.h*. It indicates whether the project is configured to execute the CoreMark benchmark or the µA/MHz test.

- If this macro is defined, the CoreMark score test will run.
- If this macro is commented out, µA/MHz test will run.

Use this macro to switch between the two benchmarks cases.

2.2.2 IDE optimization setting

The following optimizations are compiler-based and therefore IDE dependent. These optimizations apply to both the SRAM and flash based projects.

2.2.2.1 Keil optimizations

There are two compiler optimizations that can be done to improve the CoreMark score. In the **Options** of each Coremark source code files, on the **C/C++(AC6)** tab, the optimization level needs to be set as **-mcpu=Cortex-m33 --target=arm-arm-none-eabi -Omax -g -mthumb -mfpu=fpv5-sp-d16 -mfloat-abi=hard -fno-common -ffp-mode=fast** in **Include Paths Misc Controls**.

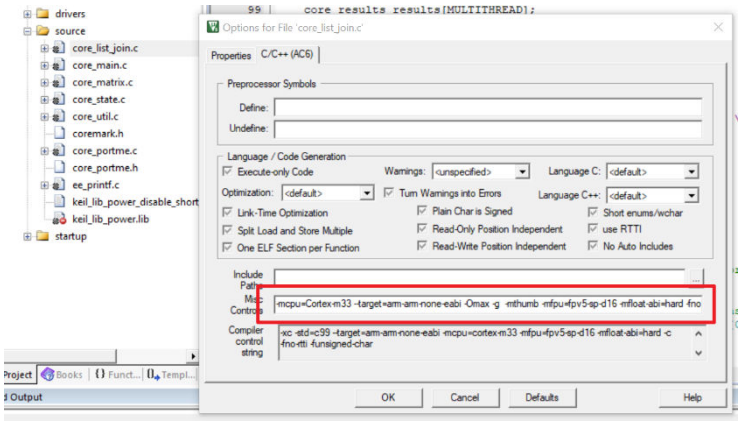

Figure 16. Keil MDK CoreMark score optimization

When benchmarking the power consumption of the MCU, the optimization setting must be set to Level 0 (-O0) and **Optimized for time** must be unchecked.

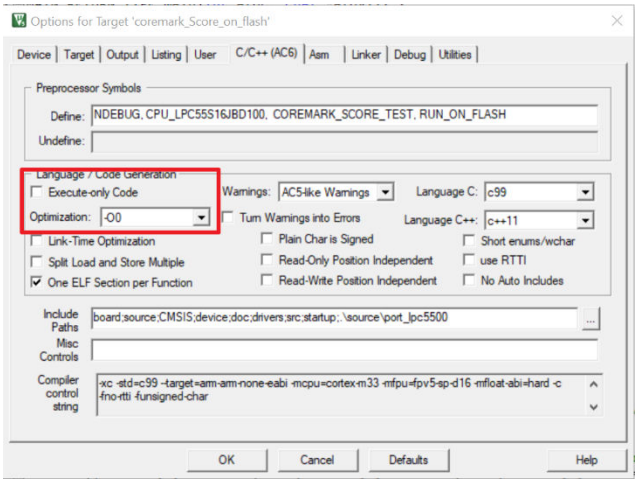

Figure 17. Keil MDK µA/MHz optimization

2.2.2.2 IAR optimization

Two compiler optimizations can be done to improve CoreMark score. Set the optimization level to **High**, select **Speed** from the drop-down menu, and check the **No size constraints**.

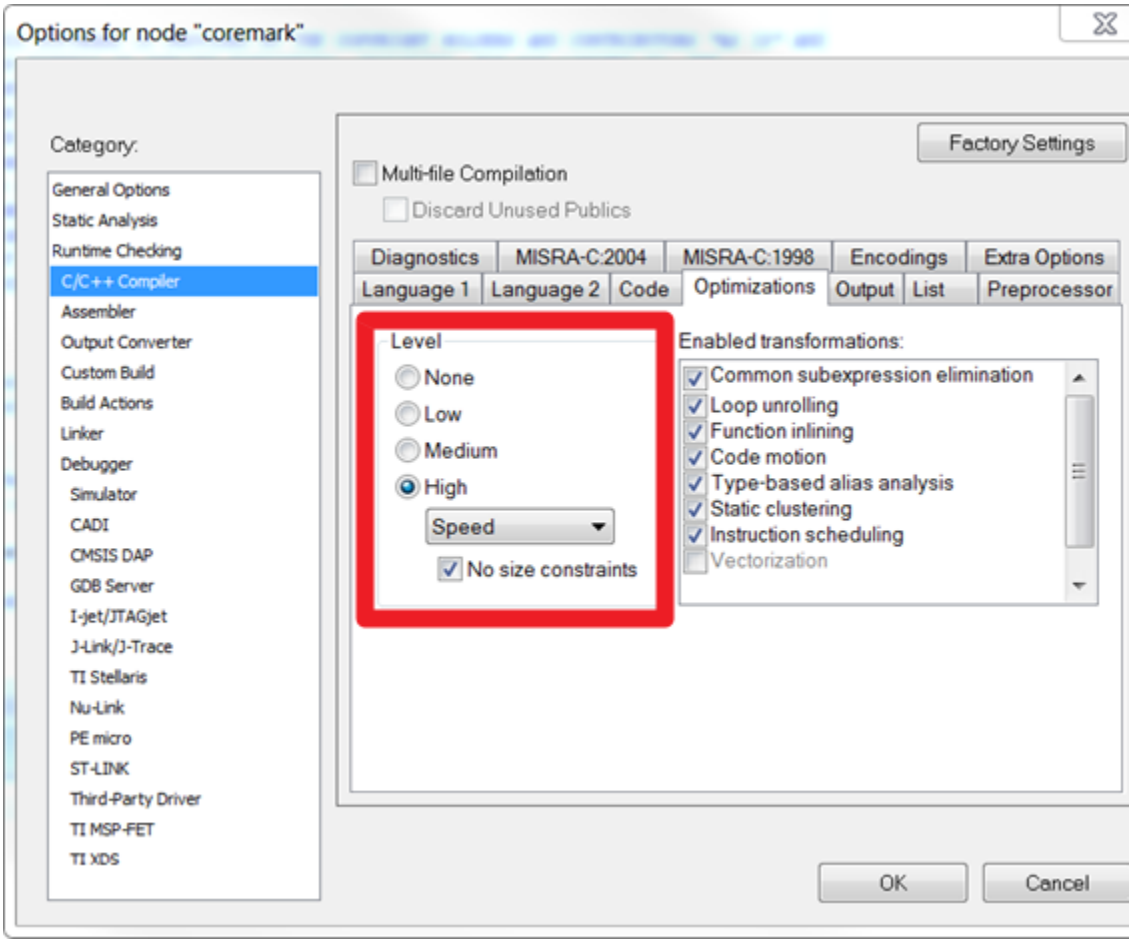

Figure 18. IAR EWARM CoreMark score optimization

When benchmarking the power consumption of the MCU, set the optimization level to **None**.

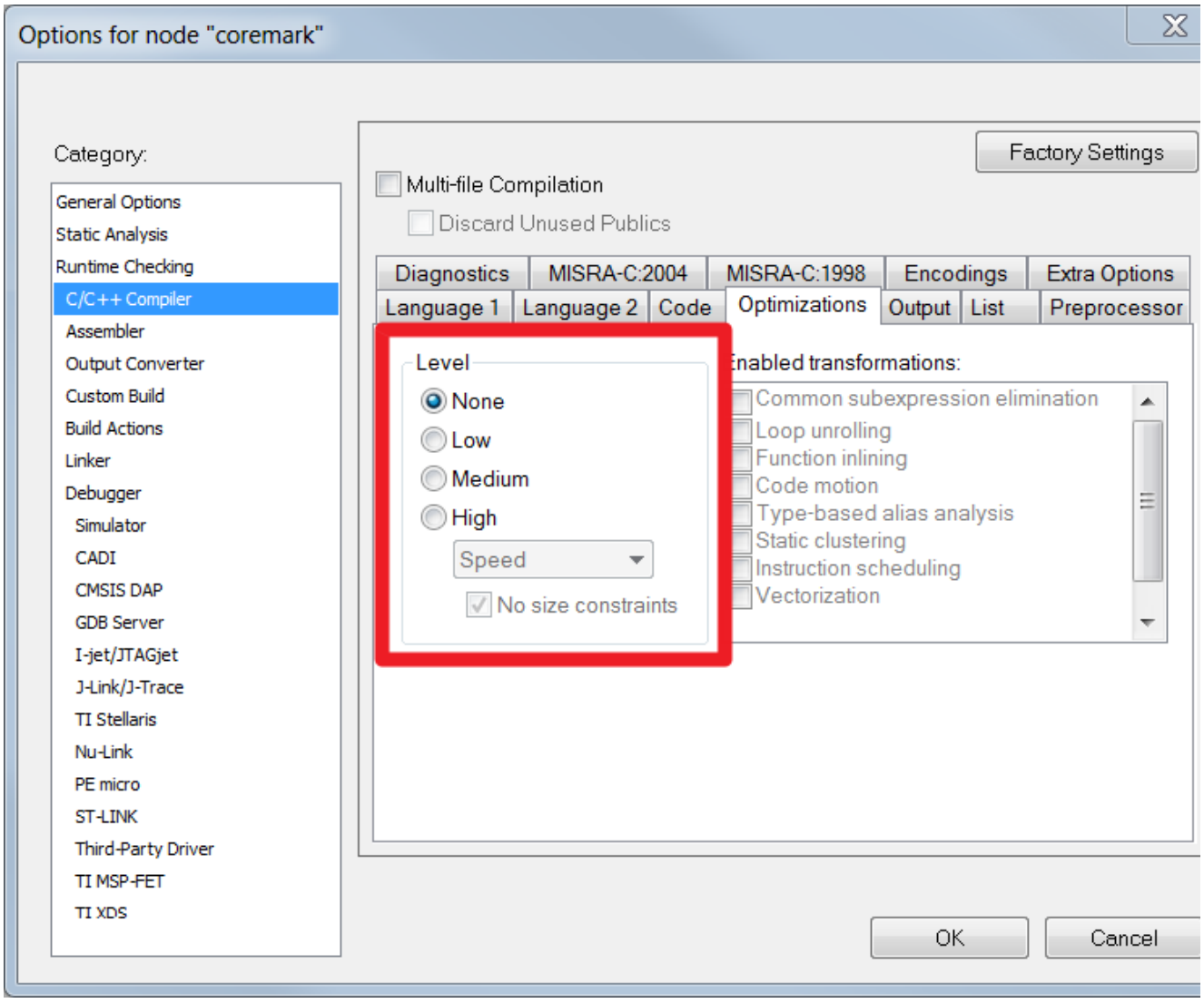

Figure 19. IAR EWARM µA/MHz optimization

2.2.2.3 MCUXpresso Optimization

There are two compiler optimizations that can be done to improve CoreMark score. To set the optimization level to **-O3**, select **Optimize most(-O3)** from the drop-down menu.

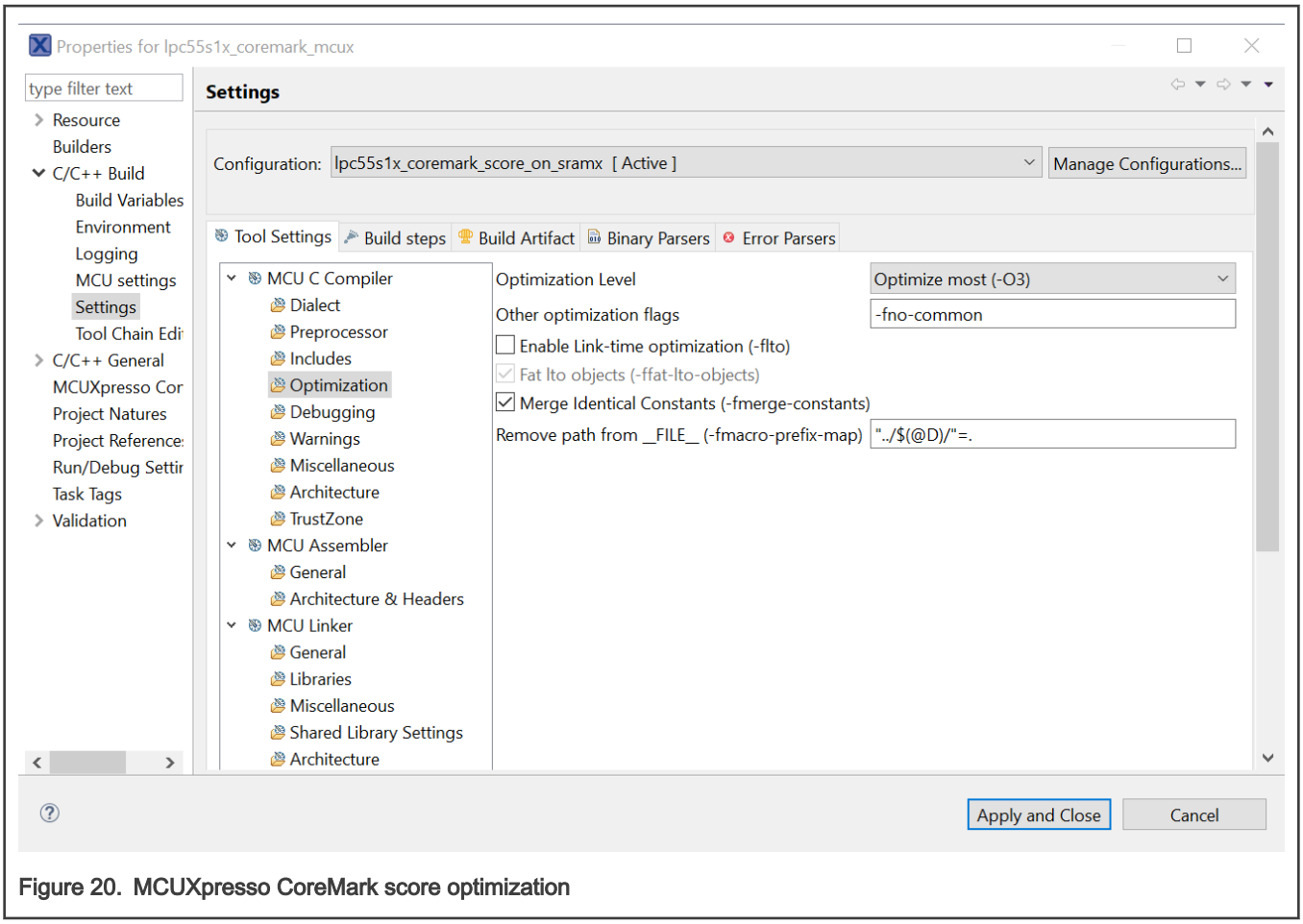

Figure 20. MCUXpresso CoreMark score optimization

When benchmarking the power consumption of the MCU, set the optimization level to **None(-O0)**.

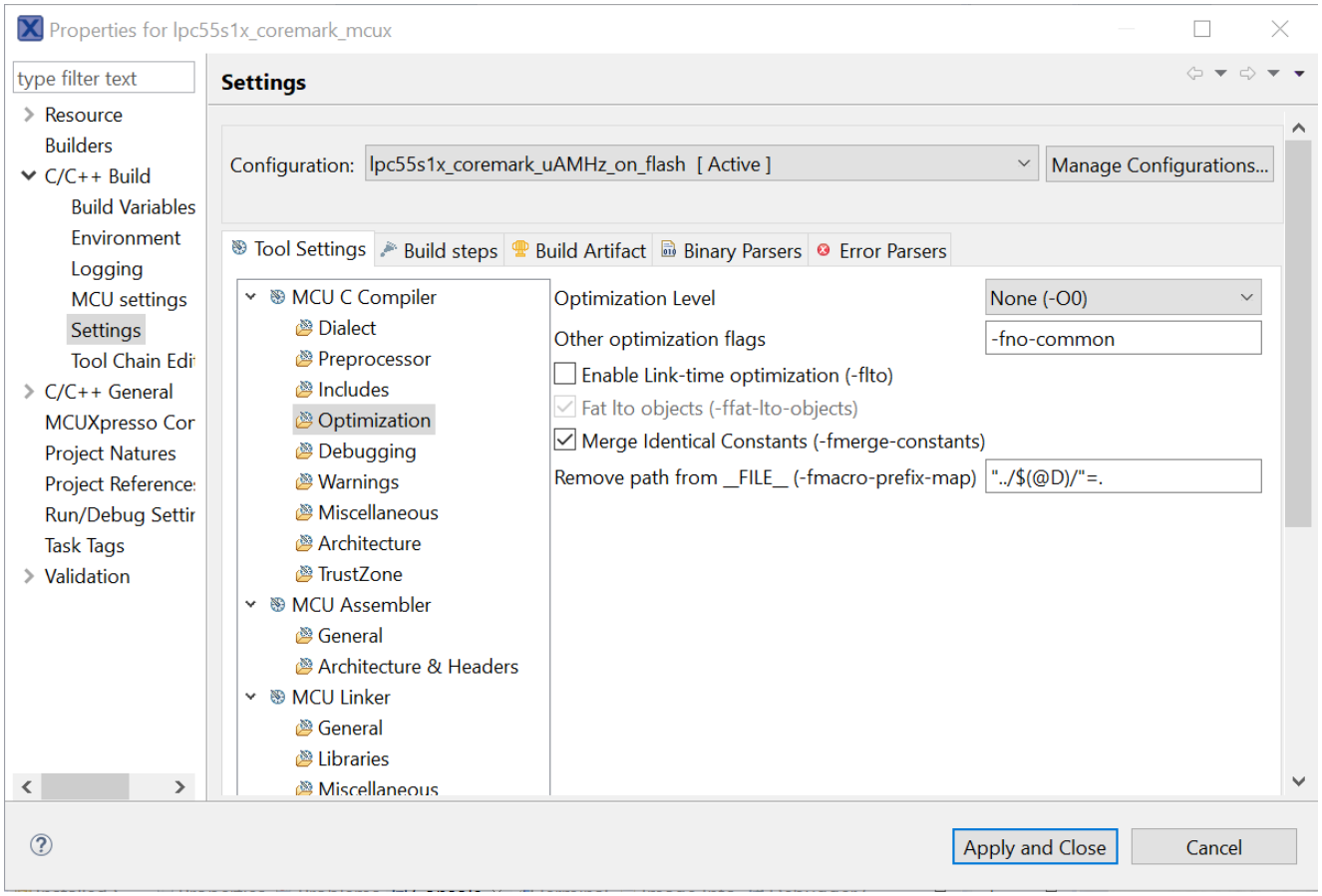

Figure 21. MCUXpresso µA/MHz optimization

3 Measuring CoreMark on board

3.1 LPC55S69Xpresso board

The LPC55S69Xpresso board supports a VCOM serial port connection via **J1**. To observe debug messages from the board set the terminal program to the appropriate COM port and use the setting **115200-8-N-1-none**. To make the debug messages easier to read, set the new line receive setting to **Auto**.

3.2 Board setup

The LPC55S16-EVK Rev A1 development board is used for benchmarking.

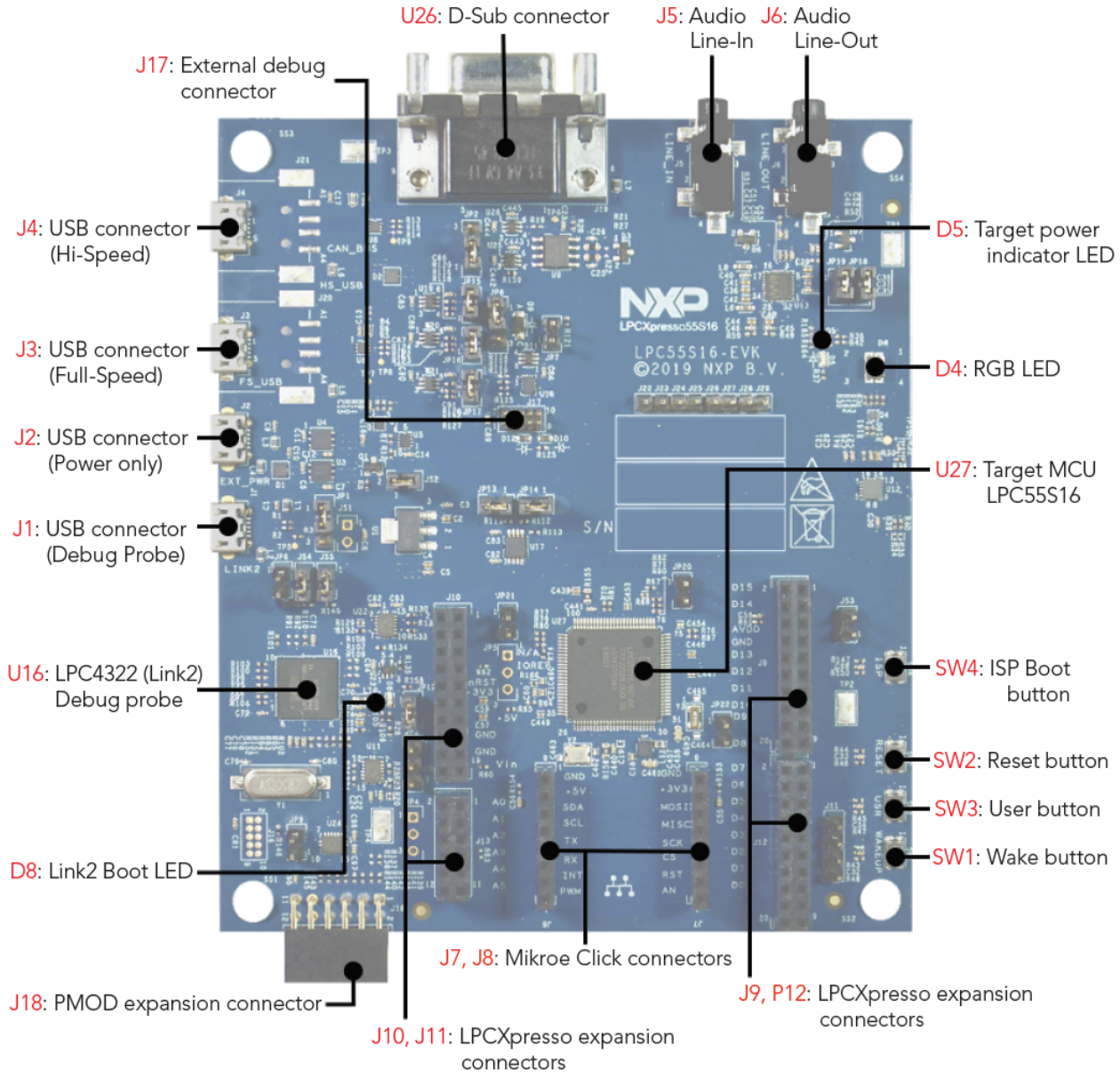

Figure 22. LPC55S16-EVK development board

The board ships with CMSIS-DAP debug firmware programmed. For more information on CMSIS_DAP debug firmware, visit FAQ.

For debugging and terminal debug messages, connect a USB cable to P6 USB connector. Board schematics are available on NXP.

3.2.1 µA/MHz measurement setup

To measure the LPC55S1x/LPC551x power consumption, connect ammeter across JP22, as shown in Figure 23.

NOTE

The current data on EVK maybe little higher than datasheet, due to the EVK have more other components may cost more power.

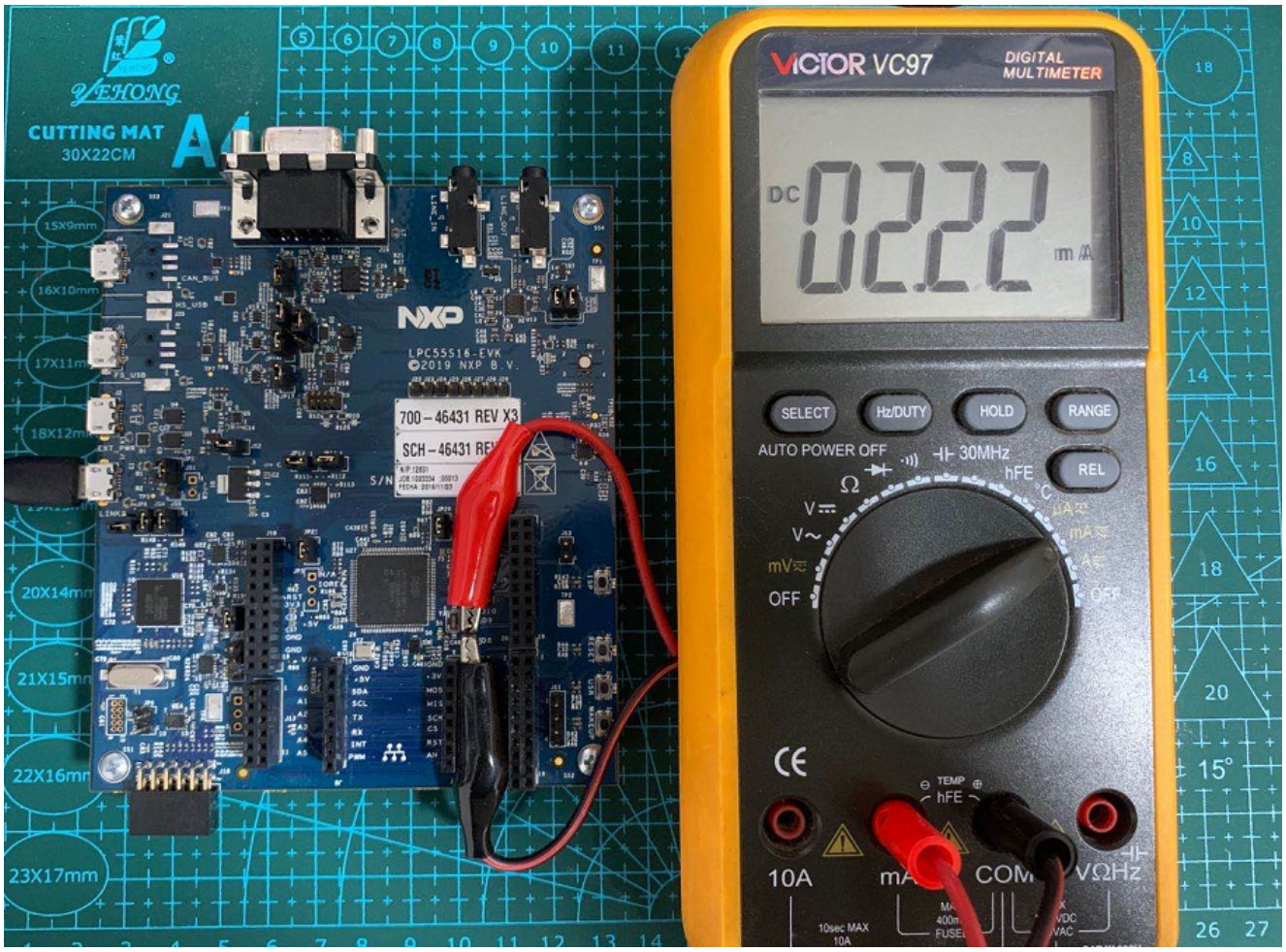

Figure 23. µA/MHz measurement setup

Users can measurement the current through JP22 with a multimeter.

When performing the µA/MHz benchmark, use J2 USB connector to provide power to the board. Additionally, after the µA/MHz benchmark project has been downloaded, power cycling the board by removing the USB cable and reinserting. It is recommended to make sure the debug probe be not connected.

The core clock frequency can be changed by selecting different configurations through the shell terminal by MCU uart0.

3.3 Run CoreMark code

To get CoreMark result:

1. Connect the board's connector J1 with PC,
2. PC recognizes the LPC-Link2 debugger with a Simulate Serial Port, as shown in Figure 24.

   If PC cannot find the serial port driver, please download the LPCScrypt and install on your PC.

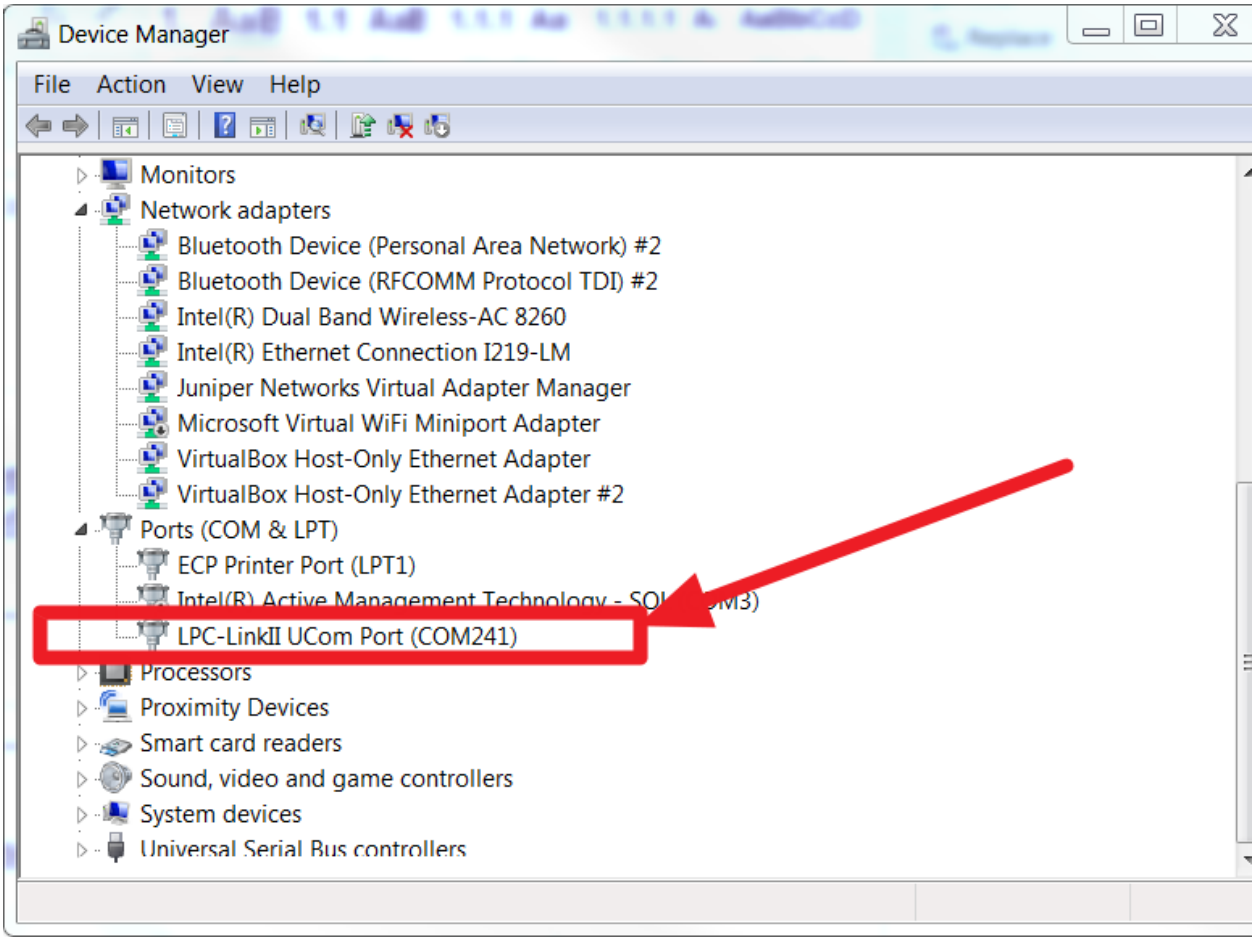

Figure 24. LPC-LinkII UCom port

3. Open a UART debug terminal, such as, Tera Term, putty, etc, and configure as 115200, 8 data bits, no parity, 1 stop bit, as shown in Figure 25.

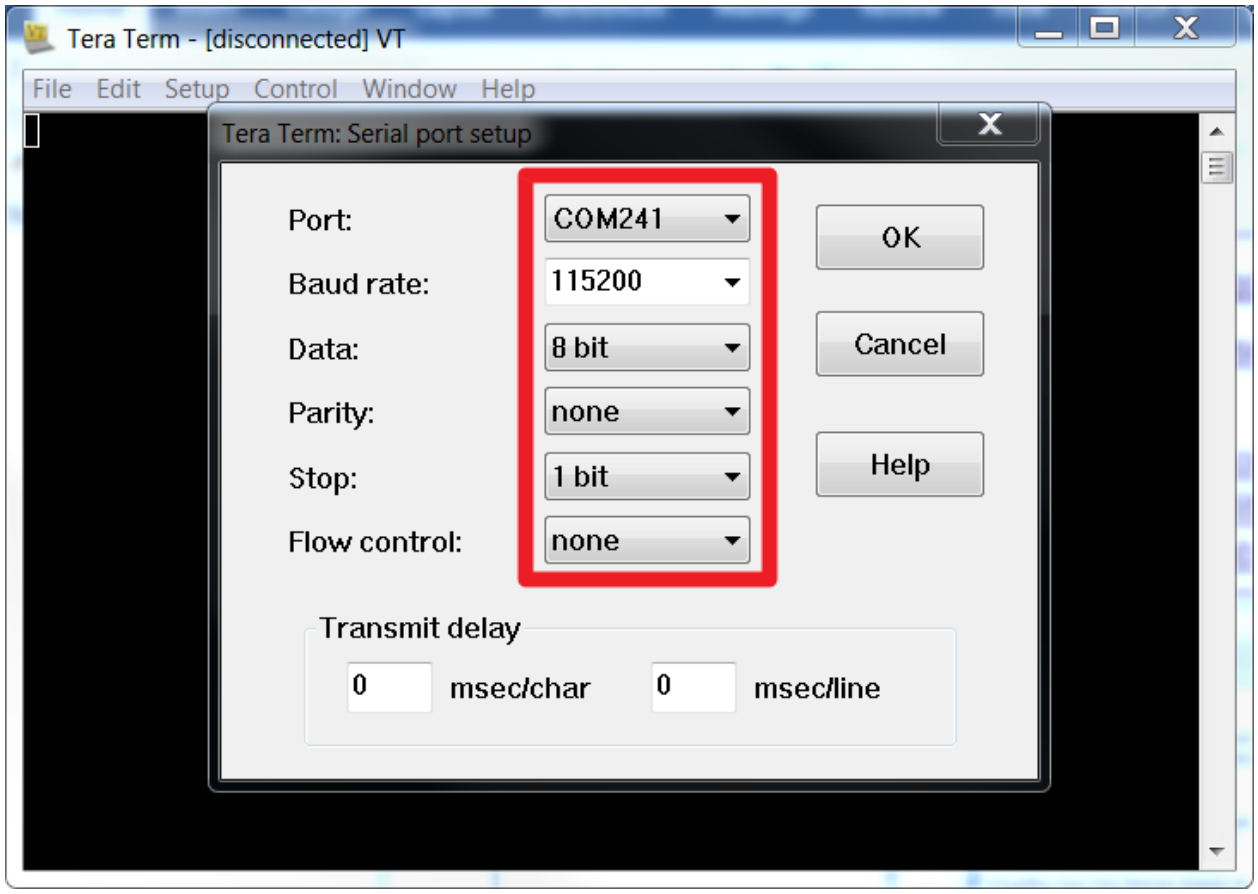

Figure 25. UART debug terminal configuration

4. Once the CoreMark necessary file is added into the project, by following instructions in Port CoreMark library into CoreMark framework, compile the project and download to the LPC55S16-EVK board.
5. Click the **Reset** button, as shown in Figure 26, and the prompt information is displayed on the terminal. Users can input **1**, **2**, **3**, or **4** from PC keyboard to select the Core frequency like 12 MHz, 96 MHz, 100 MHz (PLL), and 150 MHz (PLL). Once a character is input, the Coremark test program starts immediately and then waits for ten senconds or more. The CeMark benchmark prints on the terminal after a few seconds, as shown in Figure 27.

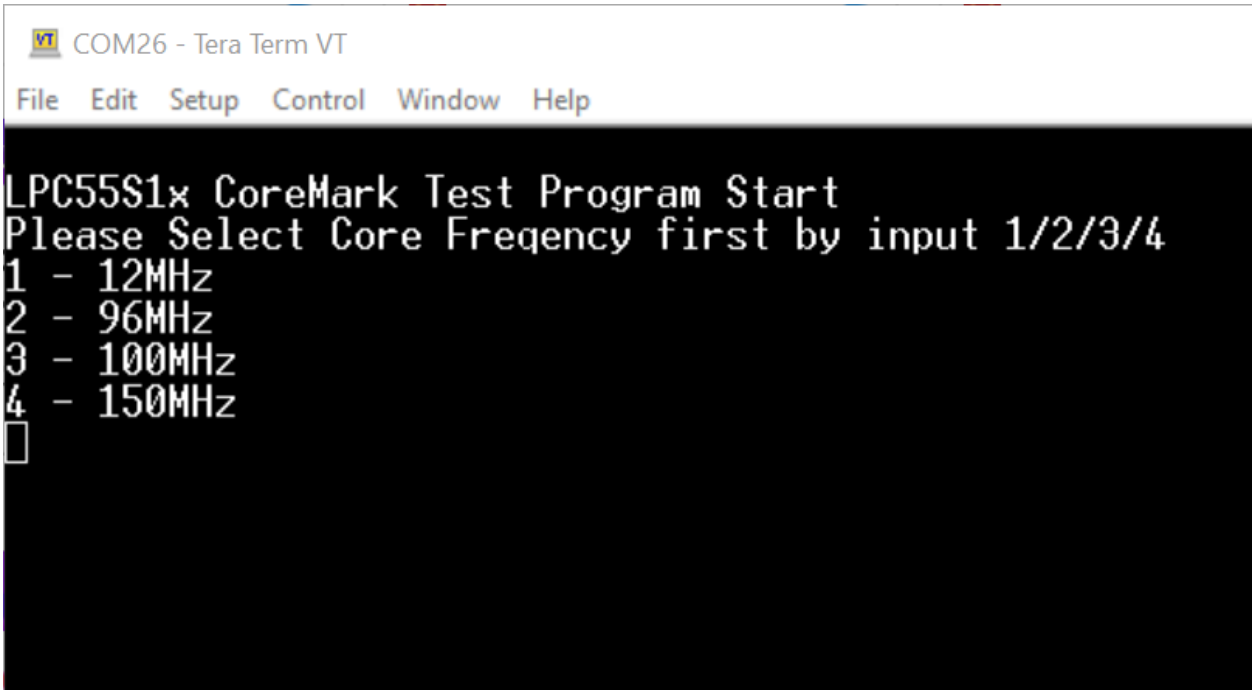

Figure 26. Coremark test core frequency menu

4 Result

Figure 27 shows the CoreMark benchmark result when running LPC55S1x/LPC551x at 150 MHz core frequency in IAR. The CoreMark benchmark score is the number of iterations per second. The CoreMark/MHz score executing from internal flash for this run is 603.298029/150 MHz = 4.021 CoreMark/MHz.

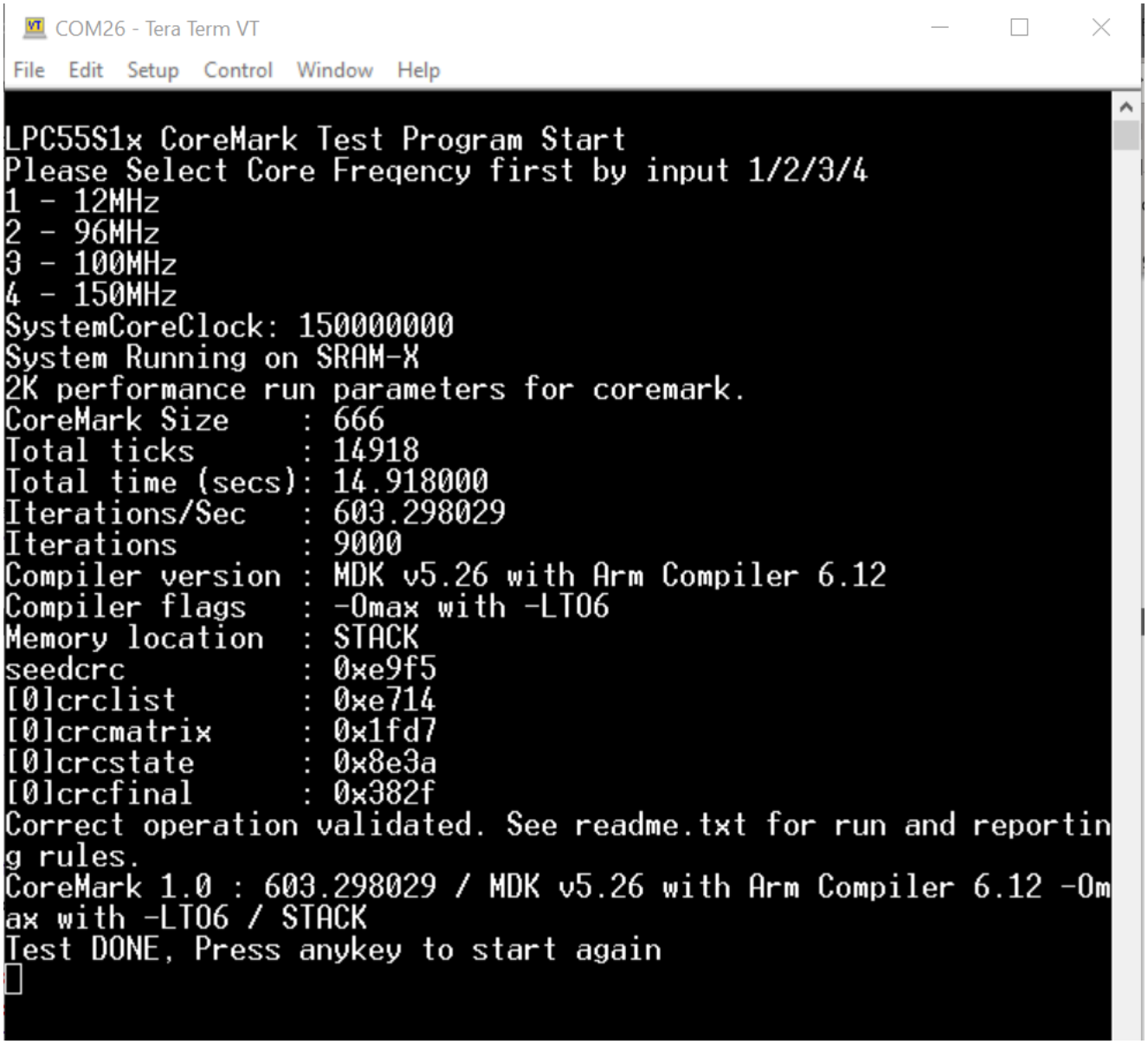

Figure 27. CoreMark result

Table 1 describes typical CoreMark score when benchmarked on Keil MDK, IAR EWARM and MCUXpresso IDE when running from internal flash and SRAM at 12 MHz core frequency.

Table 1. LPC55S16-EVK board CoreMark/MHz score when 12 MHz

IDE	CoreMark/MHz score (SRAMX)	CoreMark/MHz score (Flash)
KEIL MDK	4.012	3.760
IAR EWARM	3.872	3.668
MCUXpresso	2.824	2.715

Table 2 describes typical CoreMark score when benchmarked on Keil MDK, IAR EWARM and MCUXpresso IDE when running from internal flash and SRAM at 96 MHz core frequency.

Table 2. LPC55S16-EVK board CoreMark/MHz score when 96 MHz

IDE	CoreMark/MHz score (SRAMX)	CoreMark/MHz score (Flash)
KEIL MDK	4.021	2.514
IAR EWARM	3.879	2.668
MCUXpresso	2.830	2.120

Table 3 describes typical CoreMark score when benchmarked on Keil MDK, IAR EWARM and MCUXpresso IDE when running from internal flash and SRAM at 100 MHz core frequency.

Table 3. LPC55S16-EVK board CoreMark/MHz score when 100 MHz

IDE	CoreMark/MHz score (SRAMX)	CoreMark/MHz score (Flash)
KEIL MDK	4.021	2.514
IAR EWARM	3.879	2.668
MCUXpresso	2.830	2.120

Table 4 describes typical CoreMark score when benchmarked on Keil MDK, IAR EWARM and MCUXpresso IDE when running from internal flash and SRAM at 150 MHz core frequency.

Table 4. LPC55S16-EVK board CoreMark/MHz score when 150 MHz

IDE	CoreMark/MHz score (SRAMX)	CoreMark/MHz score (Flash)
KEIL MDK	4.021	1.910
IAR EWARM	3.880	2.129
MCUXpresso	2.830	1.733

NOTE

The current data on EVK may be a little higher or lower than what are described in datasheet, as the EVK contains more components and cost more power.

The average current in 100&150 MHz will be higher than other modes, because the two modes enable PLL and the PLL costs more power.

Table 5. Keil MDK µA/MHz score

Frequency	Avg. power consumption (mA, SRAM X)	µA/MHz score (SRAM X)	Avg. power consumption (mA, Flash)	µA/MHz score (Flash)
12 MHz	1.01	84.20	1.12	100.00
96 MHz	2.95	30.72	3.22	33.55

Table continues on the next page...

Table 5. Keil MDK µA/MHz score (continued)

Frequency	Avg. power consumption (mA, SRAM X)	µA/MHz score (SRAM X)	Avg. power consumption (mA, Flash)	µA/MHz score (Flash)
100 MHz	3.31	33.10	3.58	35.80
150 MHz	5.07	33.80	4.90	32.67

Table 6. IAR EWARM µA/MHz score

Frequency	Avg. power consumption (mA, SRAM X)	µA/MHz score (SRAM X)	Avg. power consumption (mA, Flash)	µA/MHz score (Flash)
12 MHz	0.94	78.50	1.09	91.00
96 MHz	2.71	28.30	2.93	31.00
100 MHz	3.06	30.60	3.32	33.20
150 MHz	4.70	31.50	4.60	31.00

Table 7. MCUXpresso µA/MHz score

Frequency	Avg. power consumption (mA, SRAM X)	µA/MHz score (SRAM X)	Avg. power consumption (mA, Flash)	µA/MHz score (Flash)
12 MHz	0.97	80.90	1.12	93.33
48 MHz	2.85	29.70	3.20	33.40
96 MHz	3.20	32.00	3.60	36.00
150 MHz	4.90	32.67	4.90	32.67

5 Conclusion

This application note describes three types of CoreMark benchmarking on the LPC55S1x/LPC551x with different IDEs (Keil, IAR, MCUXpresso): the CoreMark score, power consumption, and the µA/MHz . It also describes how to optimize the benchmark results when running the benchmark out of internal SRAM and flash.

The CoreMark results are measured on LPC55S16-EVK. The best CoreMark number is **4.021**, achieved by using KEIL MDK (Arm Compiler 6.12) and running CoreMark from SRAM X. The best CoreMark power consumption in **µA/MHz** is 28.30, achieved by running CoreMark from SRAM when core frequency is 96 MHz.

6 Reference

1. CoreMark Benchmarking for ARM Cortex Processors
2. *LPC5411x CoreMark Cortex-M4 Porting Guide* (document AN11181)
3. *LPC55S1x User Manual* (document UM11295)

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

Security — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer’s applications and products. Customer’s responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer’s applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: November, 2020

Document identifier: AN13072

arm