AN13035

LPC55(S)0x CoreMark Porting Guide

Rev. 0 — 30 October 2020 Application Note

Contents

1 Introduction

CoreMark, developed by EEMBC, is a simple, yet sophisticated benchmark that 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.

LPC55S0x/LPC550x 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
- Running at a frequency of up to 96 MHz
- PRINCE module for on-the-fly flash encryption/decryption
- CASPER Crypto/FFT engine
- One CAN-FD with dedicated DMA controller
- 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, I2C, or I2S interface)
- Programmable Logic Unit (PLU)
- One 16-bit 2.0 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, and Cortex-M4 official CoreMark is 3.40 CoreMark/MHz.

This application note describes how to port CoreMark code to LPC55S0x/LPC550x, 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 SDK 2.8 framework

The software package associated with this application note contains SDK 2.8 based project framework that allows developers to drop in the CoreMark library sources and quickly get up and running with benchmarking the LPC55S0x/LPC550x. To get started, go to: https://www.eembc.org/coremark. Click the "Download" link as shown in the following figure and follow the instructions on that page.

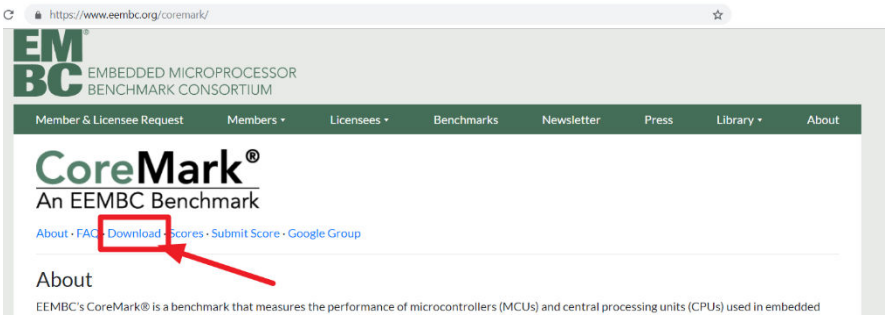

Figure 1. EEMBC CoreMark download link

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 also helps with getting familiar with the CoreMark terminology used throughout the application note.

2.1 Porting CoreMark library into CoreMark framework

There are four variants of CoreMark projects in this application note for each IDE. The four variants execute the CoreMark application from internal flash and other variants 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 – measures current when Coremark is executed on Flash.
4. coremark_uAMHz_on_sramx – measures current when Coremark is executed on RAM.

The CoreMark projects are found in the following locations:

- Keil MDK IDE:

  lpc55s0x_coremark_mdk\ lpc55s0x_coremark_mdk.uvprojx

- IAR Workbench IDE:

  lpc55s0x_coremark_iar\ lpc55s0x_coremark_iar.eww

Each of execute settings has three frequency settings : 12 MHz (FRO), 48 MHz (FRO), and 96 MHz (FRO).

Depending on the toolchain, the workspace should look like as shown in the figures in the following sections. 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 lpc55s0x_coremark_xxx project must be set as active before the CoreMark source code files can be added.

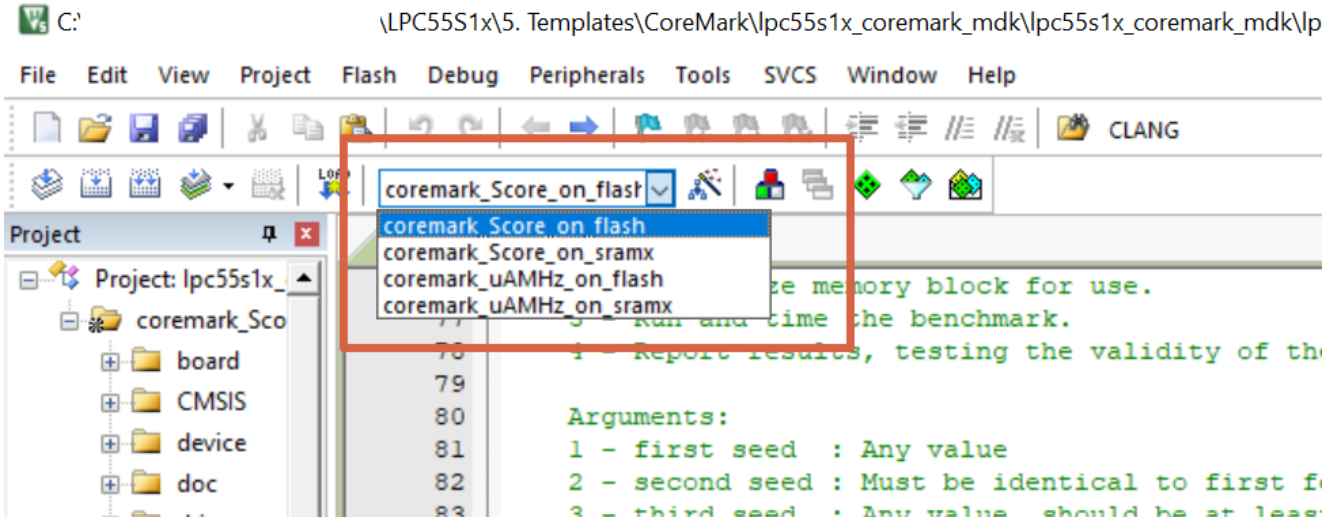

Figure 2. Keil MDK CoreMark project configuration select

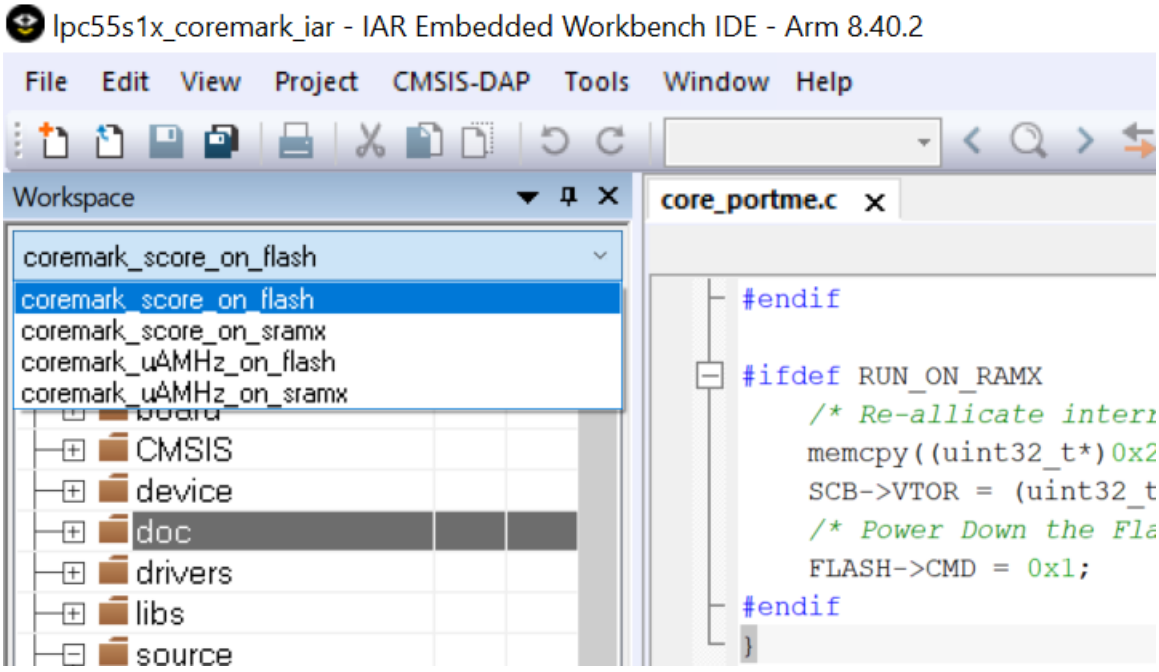

Figure 3. IAR EWARM workspace

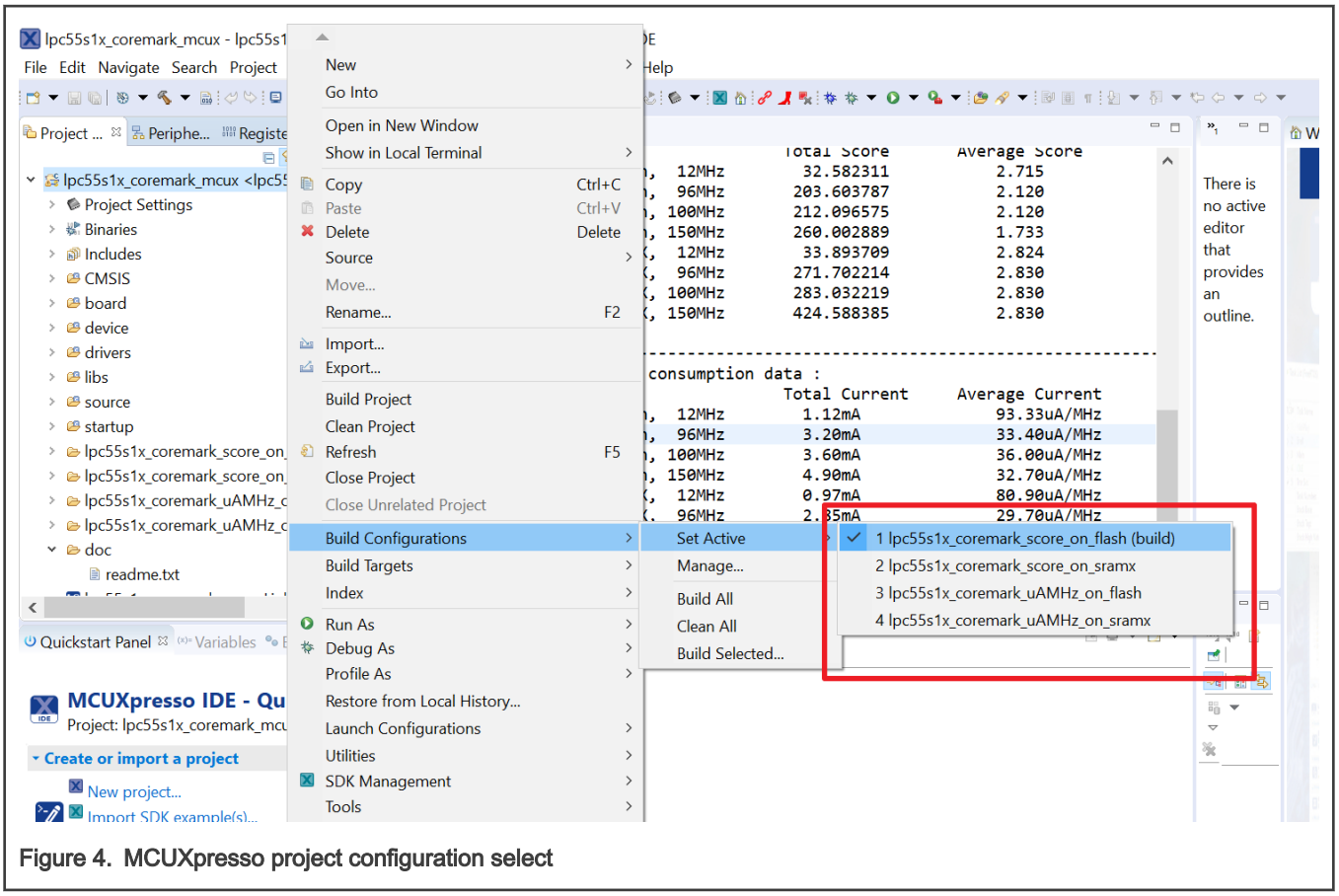

Figure 4. MCUXpresso project configuration select

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

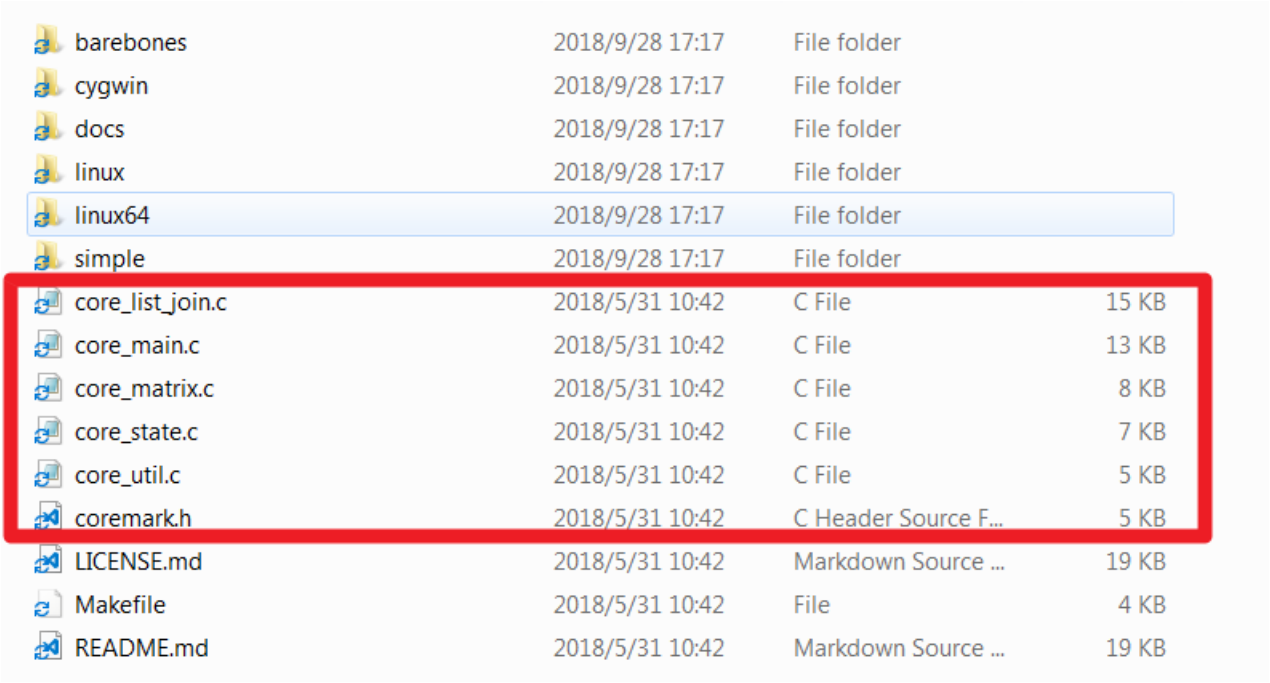

Figure 5. CoreMark files

- For Keil MDK, place these files in the project directory:

```
lpc55s0x_coremark_mdk\source
```

- For IAR Embedded Workbench, place these files in the project directory:

```
lpc55s0x_coremark_iar\source
```

- For MCUXpresso place these files in the project directory

```
lpc55s0x_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 `lpc55s0x_coremark_mdk\source\port_lpc5500`.

  Add the files into the Keil MDK project framework to the respective group source by double-clicking on the groups.

- For IAR Embedded workbench, place the files in `lpc55s0x_coremark_iar\source\port_lpc5500`.

  Add the files into the IAR project framework to the respective group source by double-clicking on the groups.

- For MCUXpresso, place the files in `lpc55s0x_coremark_mcux\source\port_lpc5500`.

  Add the files into the MCUXpresso project framework to the respective group source by clicking "refresh".

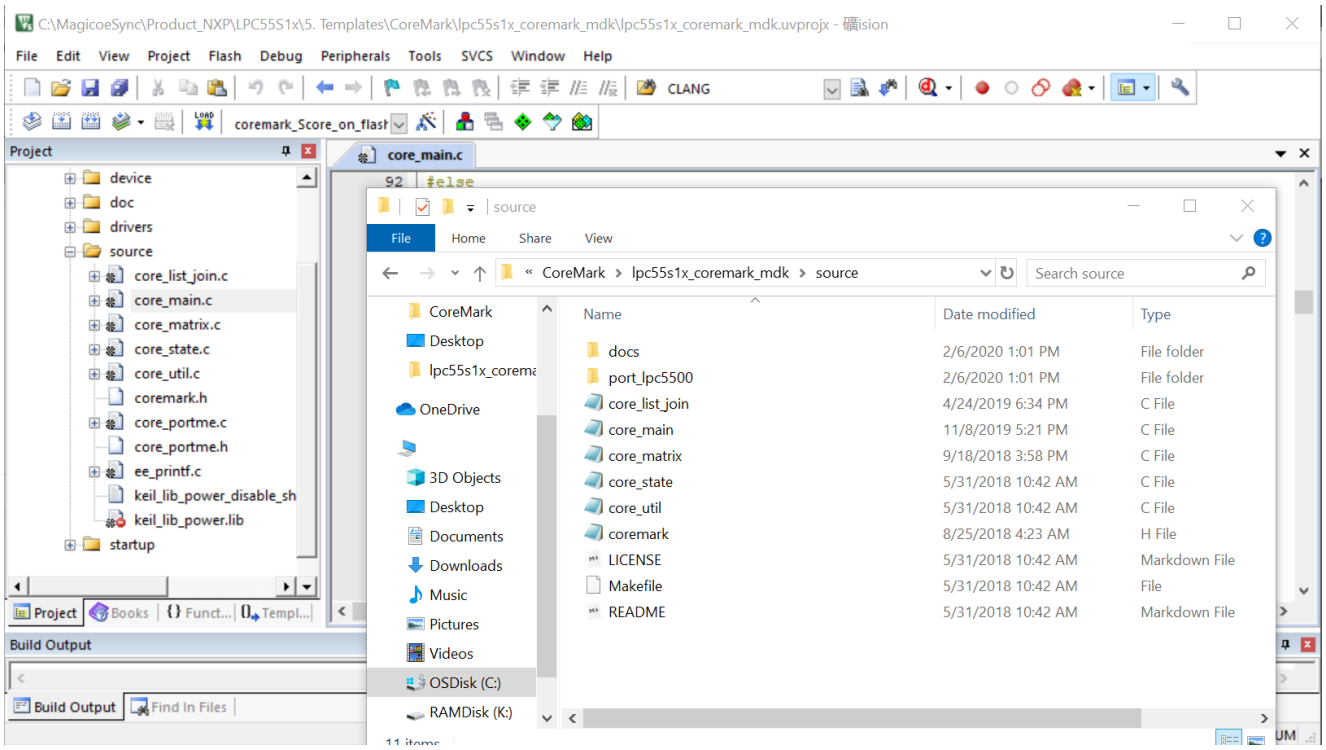

Figure 6. Adding files in Keil MDK

For KEIL MDK project, right-click the source folder and select "Add", and then select "Add Files...".

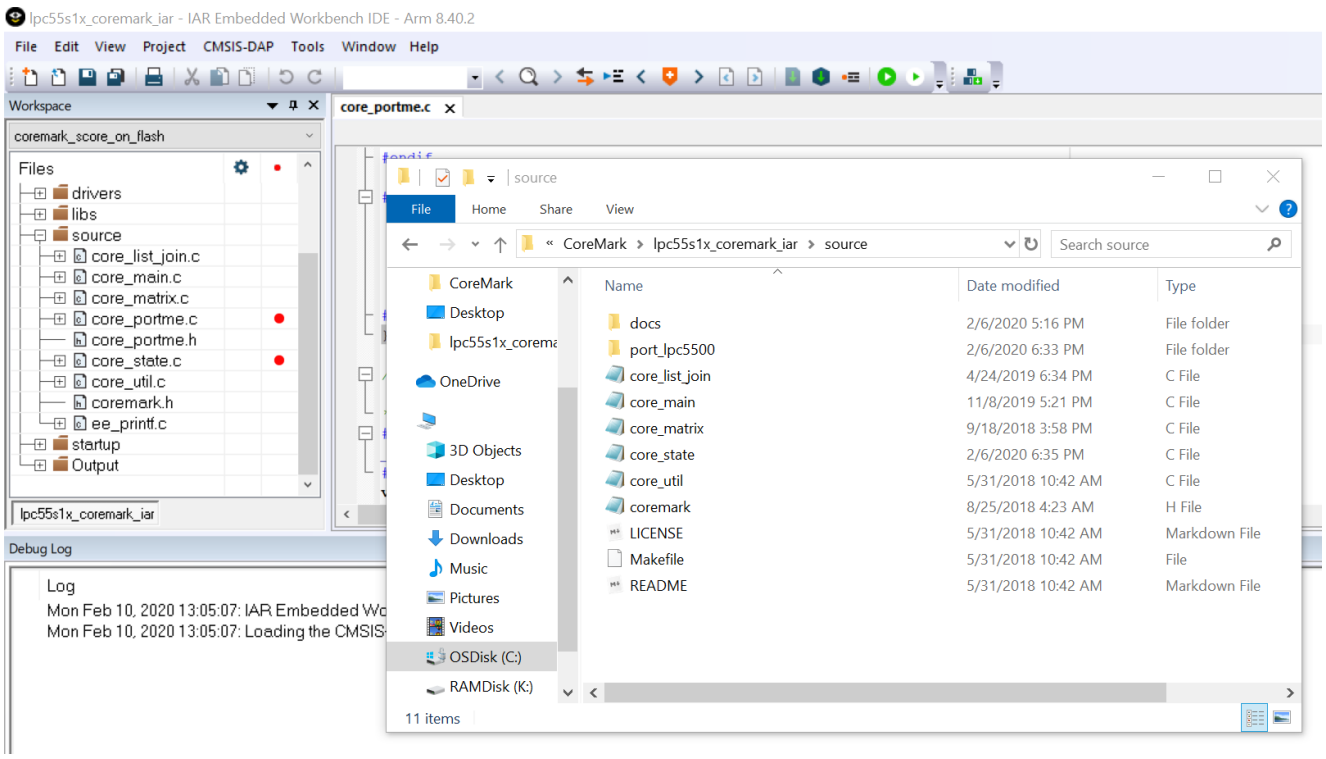

Figure 7. Adding files in IAR EWARM workspace

For IAR Embedded workbench right click the source folder and select "Add", and then select "Add Files...".

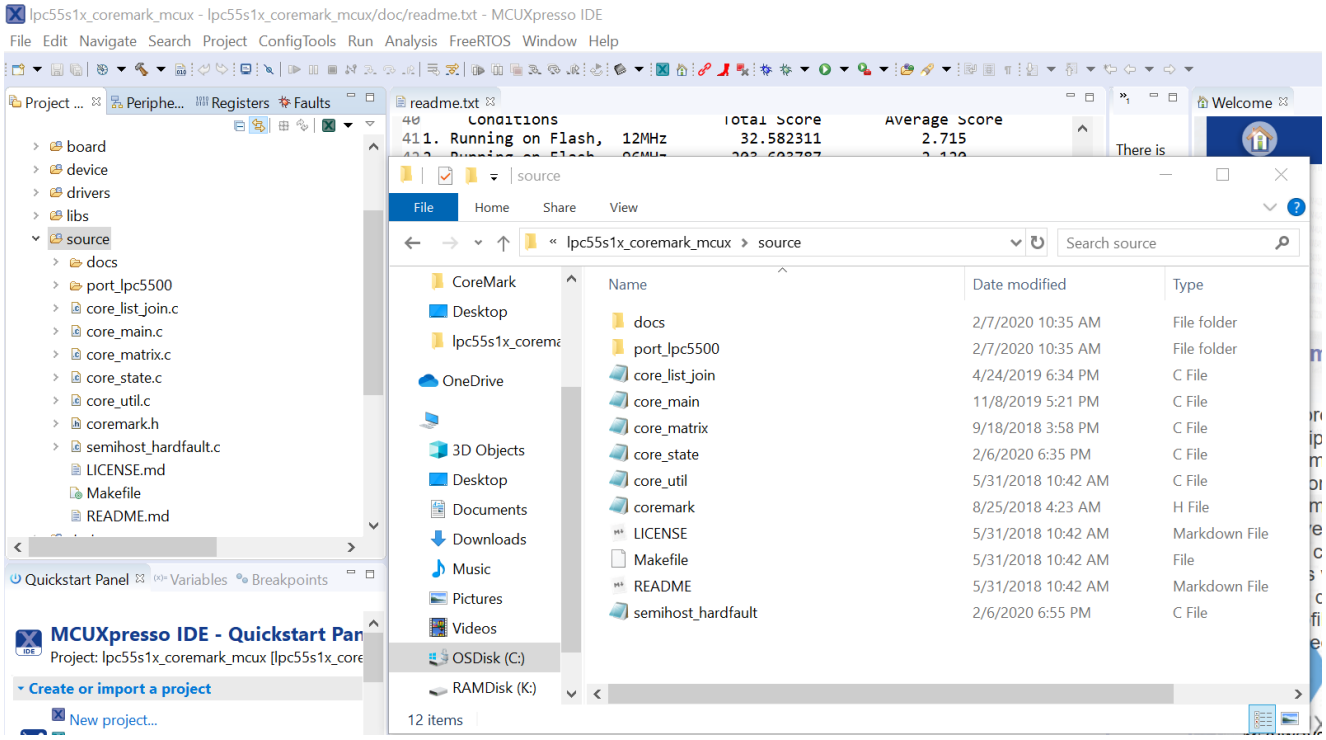

Figure 8. Adding files in MCUXpresso workspace

For MCUXpresso project, copy the files into the "source" folder, and then click "refresh". The files will be added in the 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 to 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.

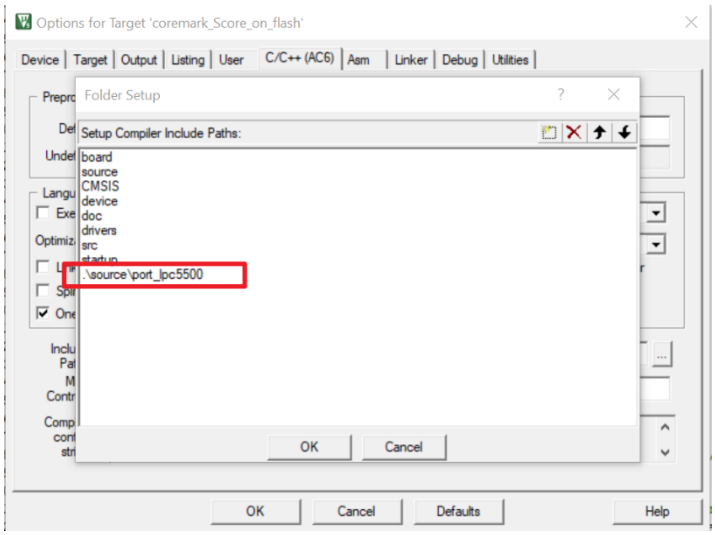

Figure 9. Keil MDK compiler include path

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

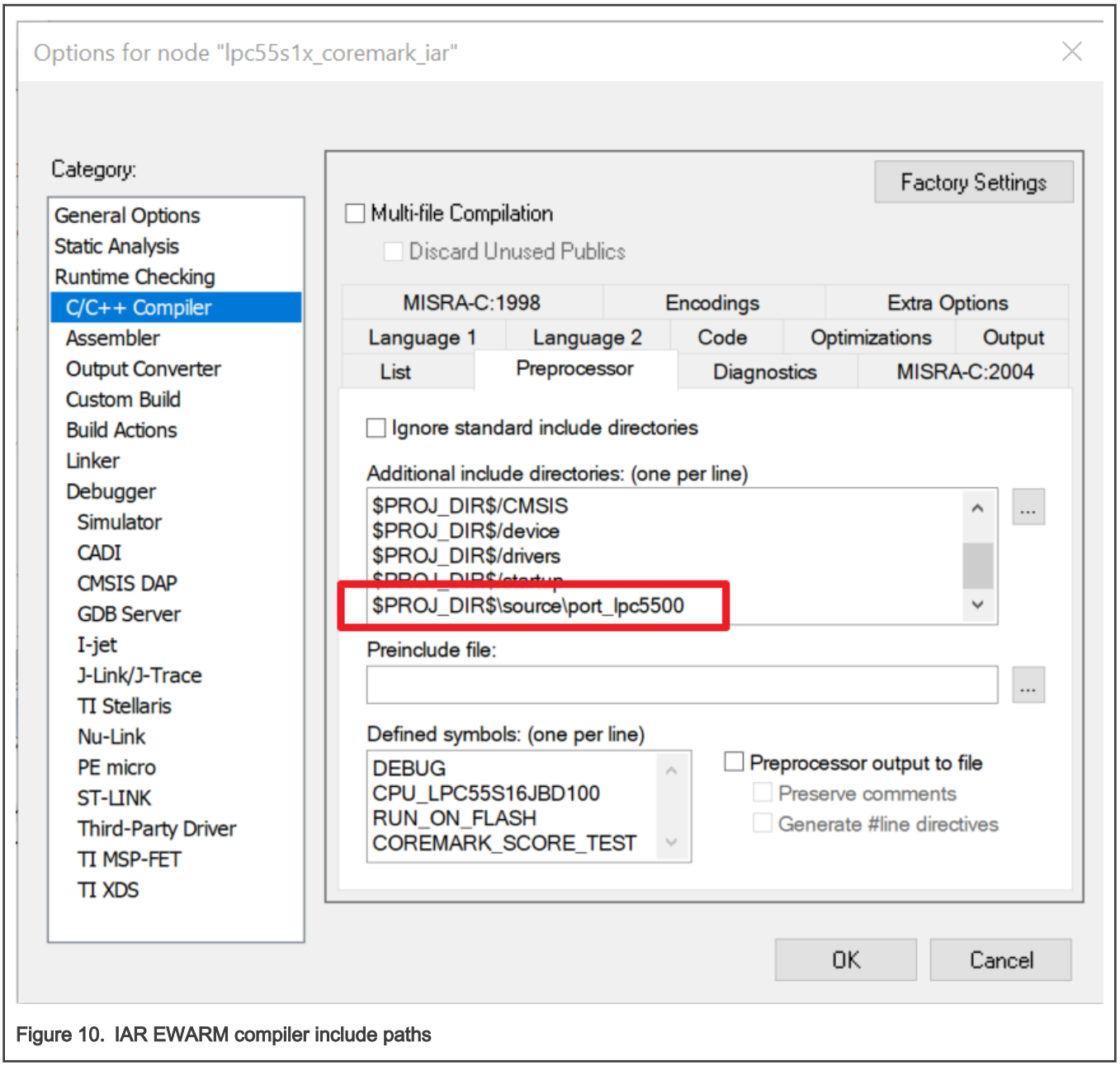

Figure 10. IAR EWARM compiler include paths

The CoreMark files are successfully ported into the CoreMark project framework.

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

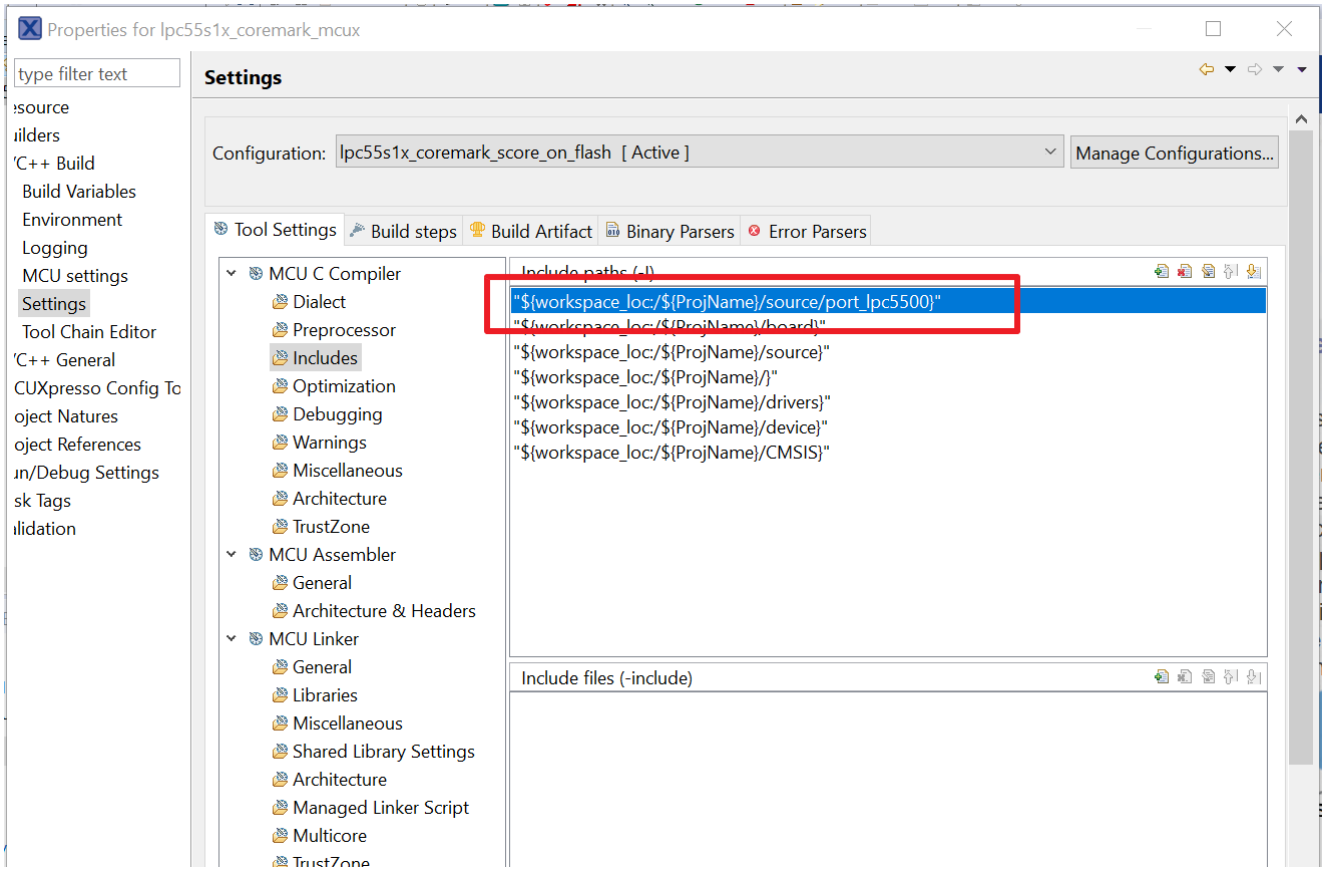

Figure 11. MCUXpresso compiler include paths

The CoreMark files are successfully ported into the CoreMark project framework.

2.1.2 CoreMark framework to execute from internal SRAM

The project lpc55s0x_coremark_xxx_on_sramx executes the CoreMark application from the 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:

```
.\lpc55s0x_coremark_mdk\LPC55S06_coremark_score_sramx.scf
```

The following figure shows the linker script setting for the lpc55s0x_coremark_xxx_on_sramx project.

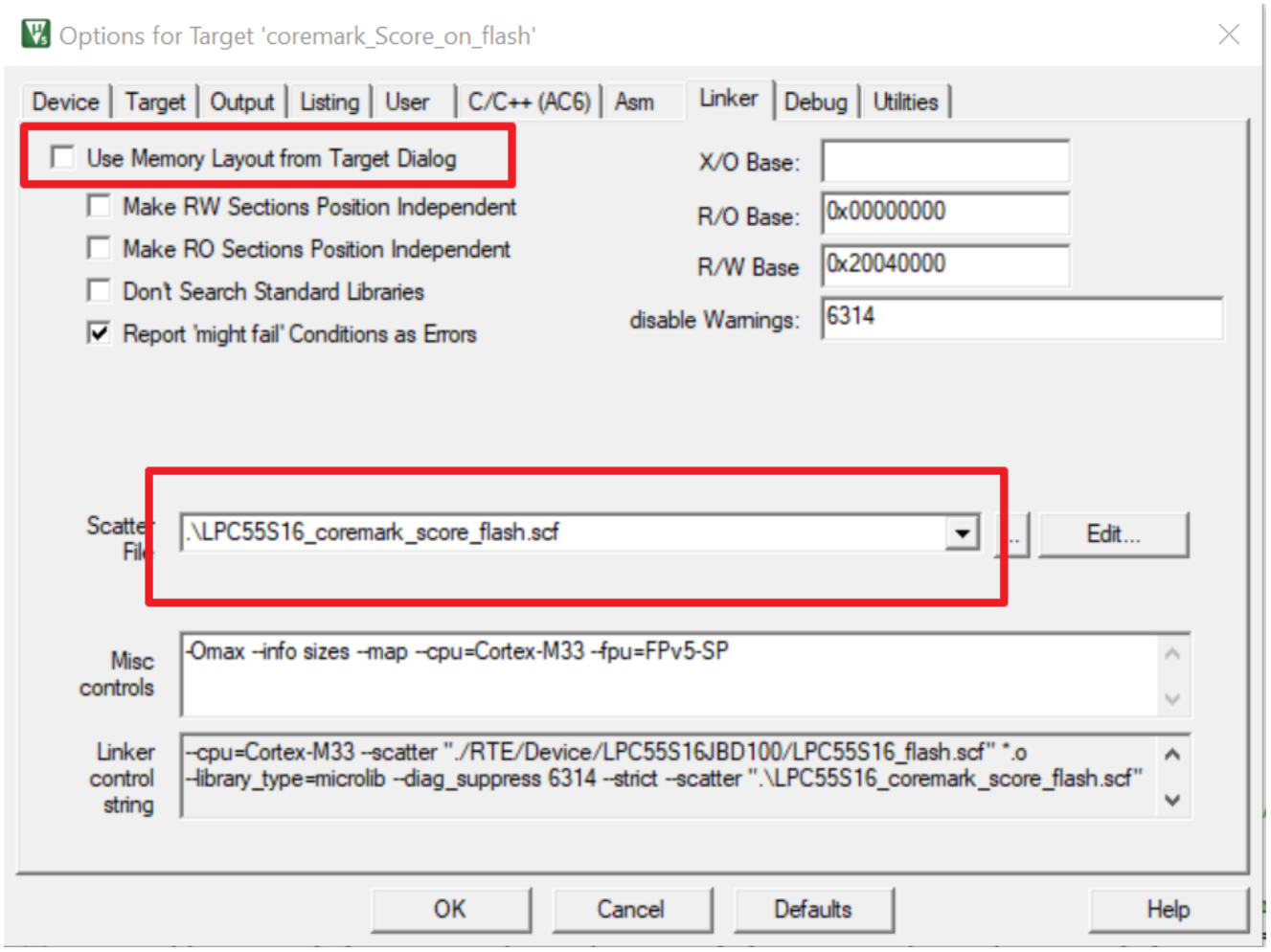

Figure 12. Linker script in Keil IDE

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 the following figure.

```
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 allocating Code to SRAM area

For MCUXpresso to execute CoreMark in Internal SRAM, selecte the linker file as "LPC55S06_coremark_score_sramx.ld" in "Managed Linker script", as shown in the following figure.

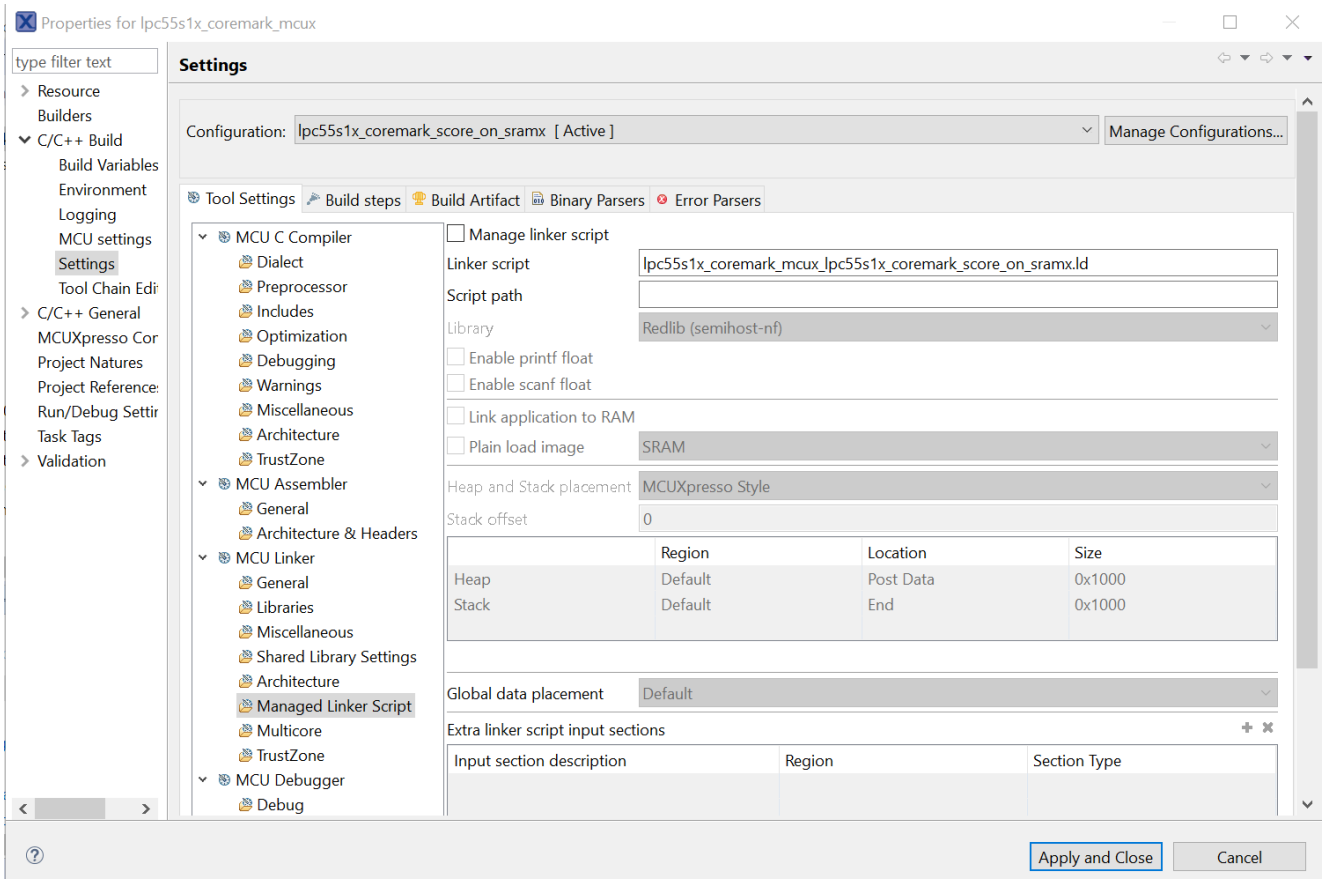

Figure 14. MCUXpresso allocating Code to SRAM area

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 be able to produce the best scores from all these 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 following are the IDE versions that are applicable to this application note:

- Keil MDK v5.28
- IAR EWARM 8.50.6
- MCUXpresso 11.2.1

2.2.1 Memory considerations

Due to the inherent architecture of SRAM and flash, CoreMark executes faster when running out of SRAM. The LPC55S0x/LPC550x internal memory uses a multilayer AHB matrix system that provides a separate instruction and data bus for Cortex-M33 and SRAMX bank. See the following figure. SRAM0 to SRAM2 are on the 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 states 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. The LPC55S0x/LPC550x user manual contains more information on correctly configuring the flash memory, such as the minimum amount of wait states allowed at a given core frequency.

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

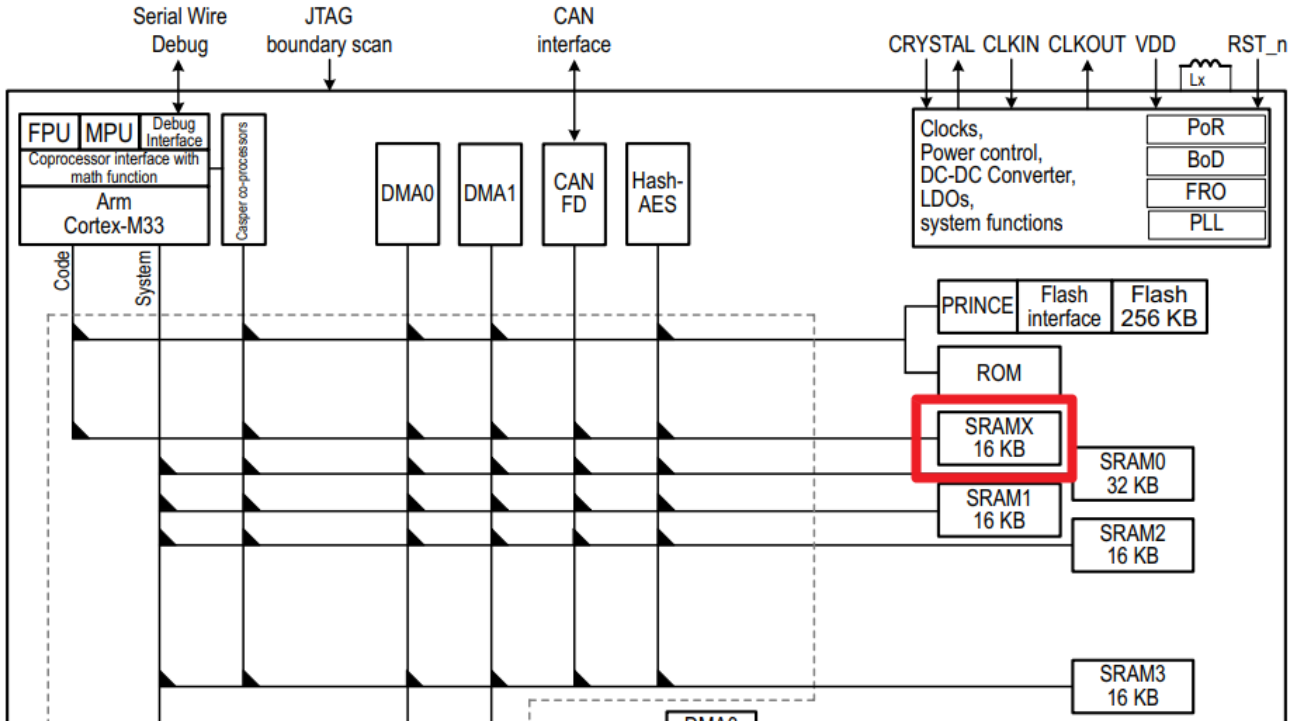

Figure 15. LPC55S0x/LPC550x AHB matrix

In both the SRAM and flash projects, there is a `COREMARK_SCORE_TEST` macro defined in `core_portme.h` that 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, the µA/MHz test will run. Use this macro to switch between the two benchmarks cases.

2.2.2 IDE optimization settings

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 each Coremark source code files' Options and under the C/C++(AC6) tab, the optimization level needs to be set to "-mcpu=Cortex-m33 --target=arm-arm-none-eabi -Omax -g -mthumb -mfpu=fpv5-sp-d16 -mfloat-abi=hard -fno-common -ffp-mode=fast" in Misc Ctonrols.

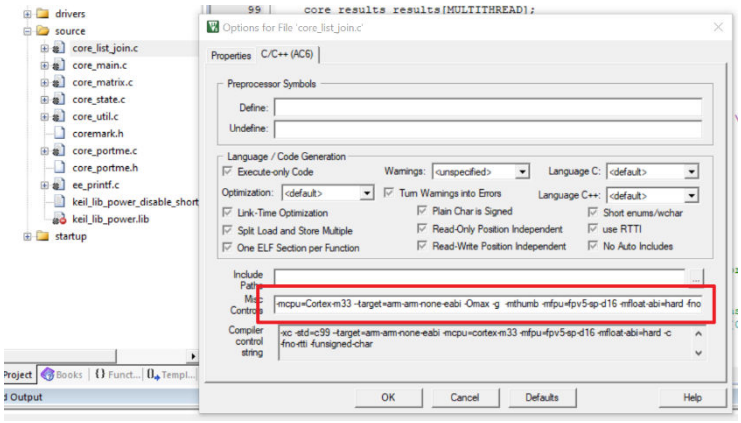

Figure 16. Keil MDK CoreMark score optimization

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.

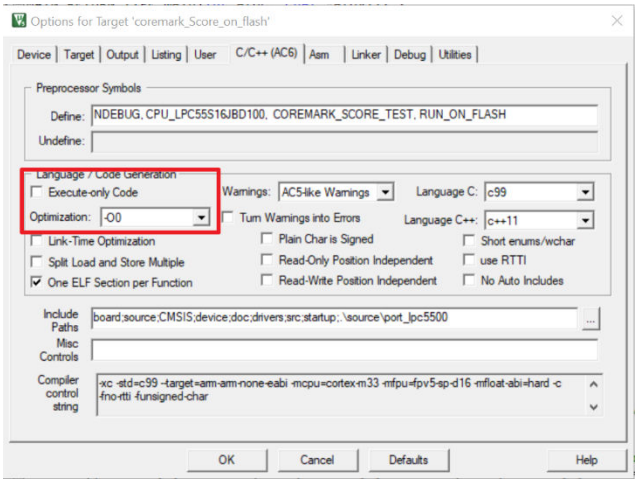

Figure 17. Keil MDK µA/MHz optimization

2.2.2.2 IAR optimization

There are two compiler optimizations that can be done to improve the CoreMark score. Set the optimization level to “High”, select “Speed” from the drop down menu and check the “No size constraints” checkbox.

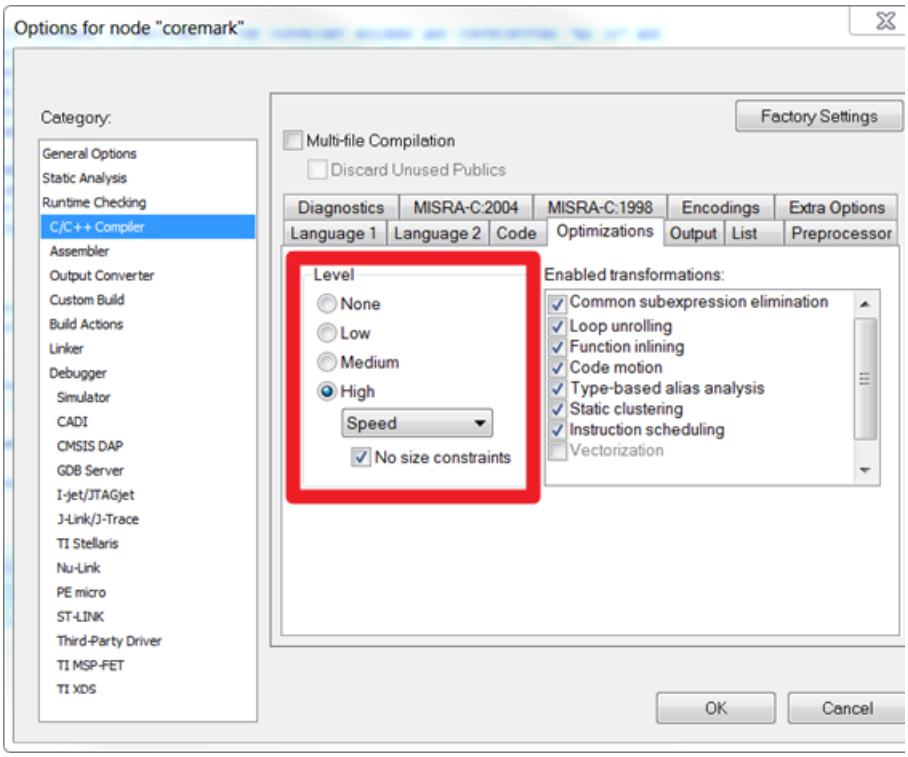

Figure 18. IAR EWARM CoreMark score optimization

When benchmarking the power consumption of the MCU, the optimization level should be set to “None”.

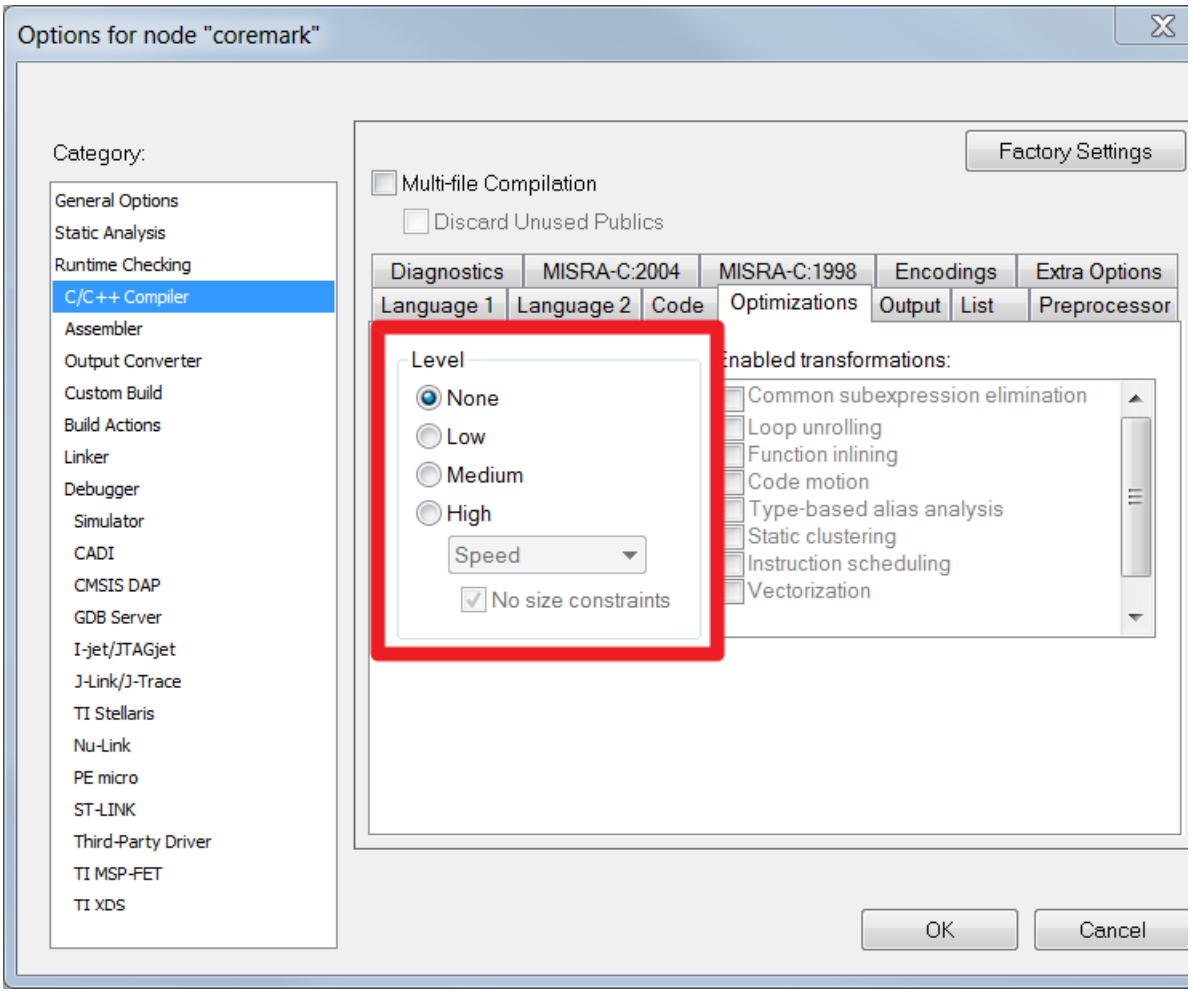

Figure 19. IAR EWARM µA/MHz optimization

2.2.2.3 MCUXpresso optimization

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

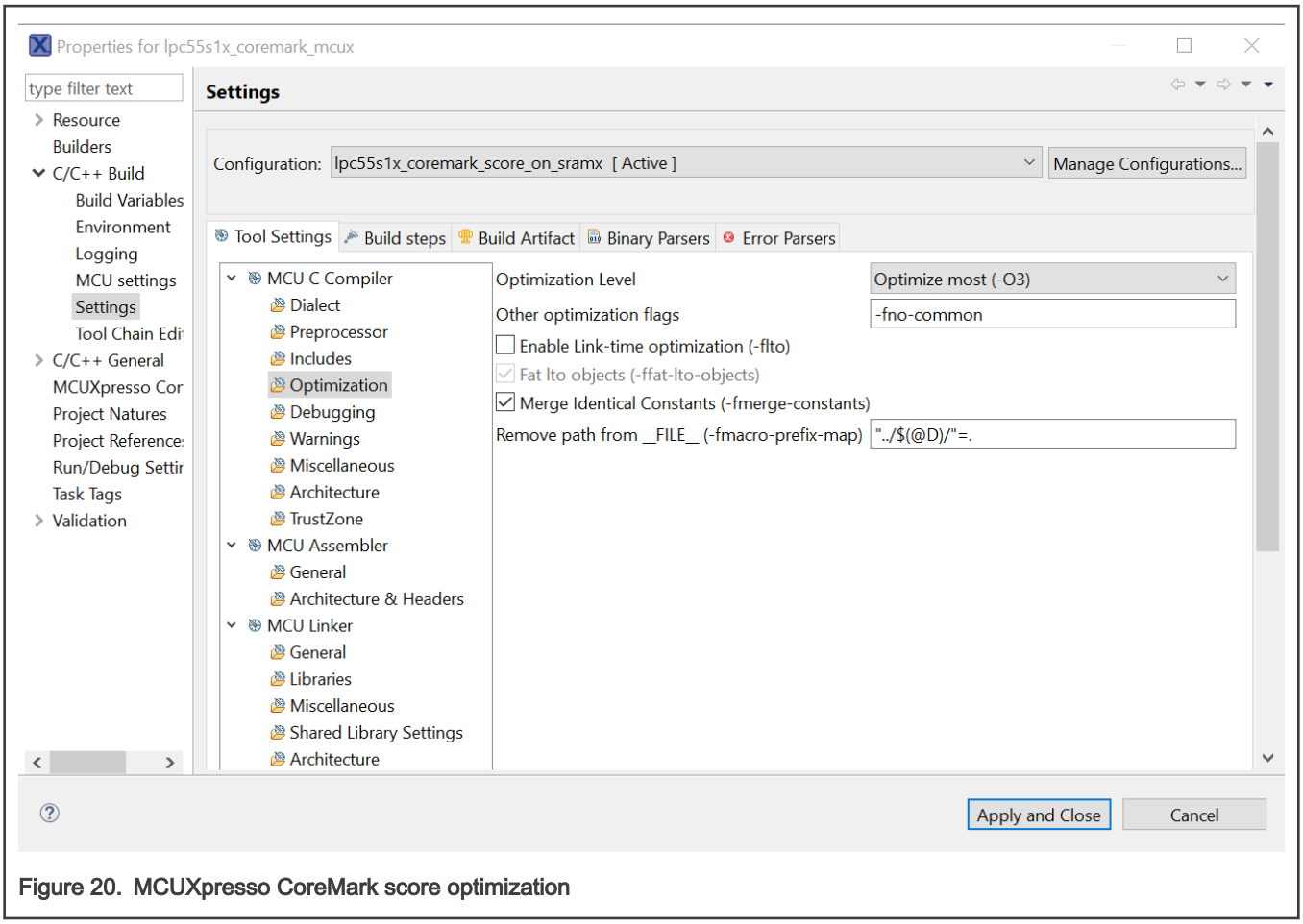

Figure 20. MCUXpresso CoreMark score optimization

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

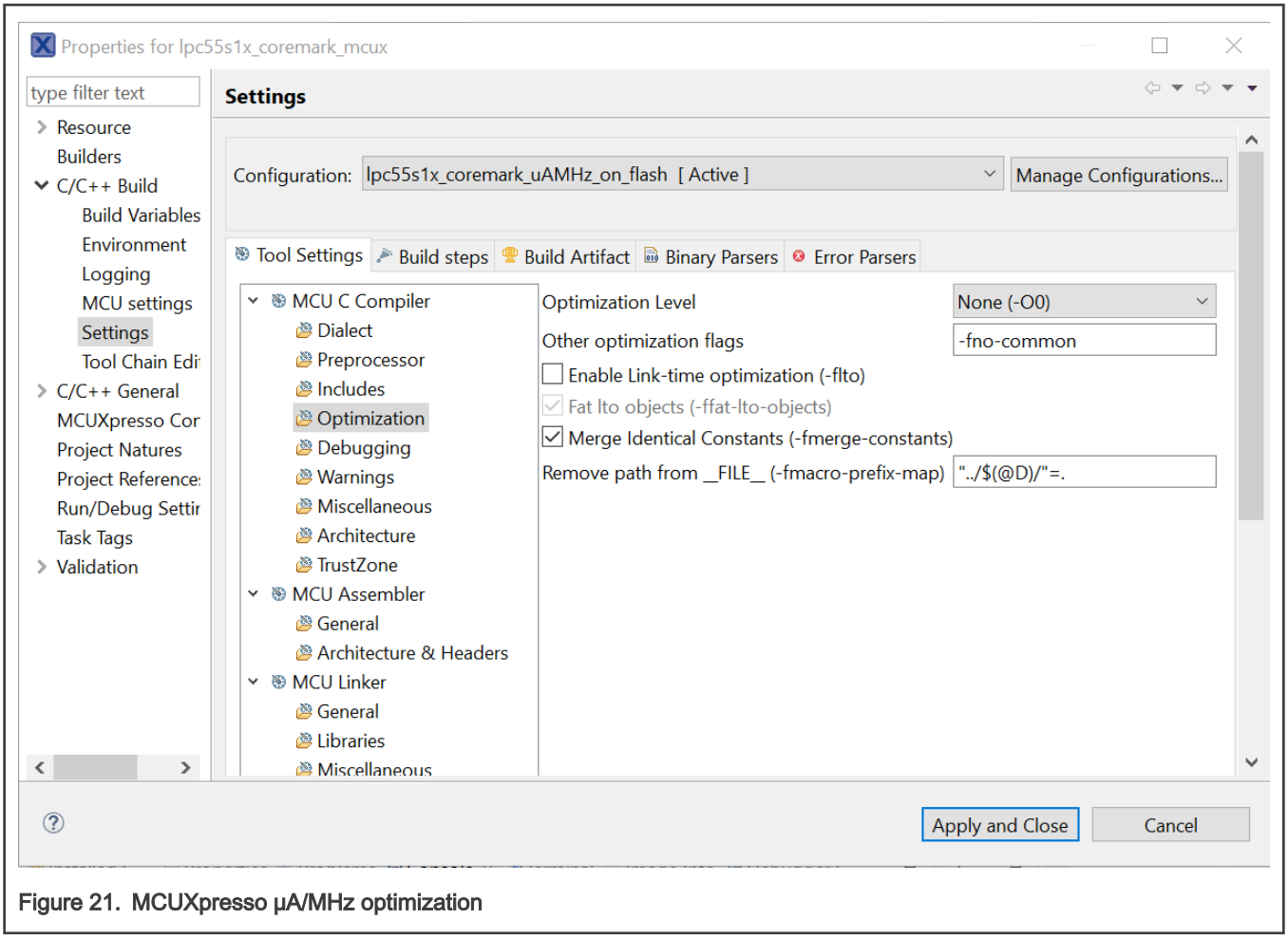

Figure 21. MCUXpresso µA/MHz optimization

3 Measuring CoreMark on board

3.1 LPC55S06Xpresso board

The LPC55S06Xpresso board supports a VCOM serial port connection through **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 LPC55S06-EVK Rev. A1 development board is used for benchmarking.

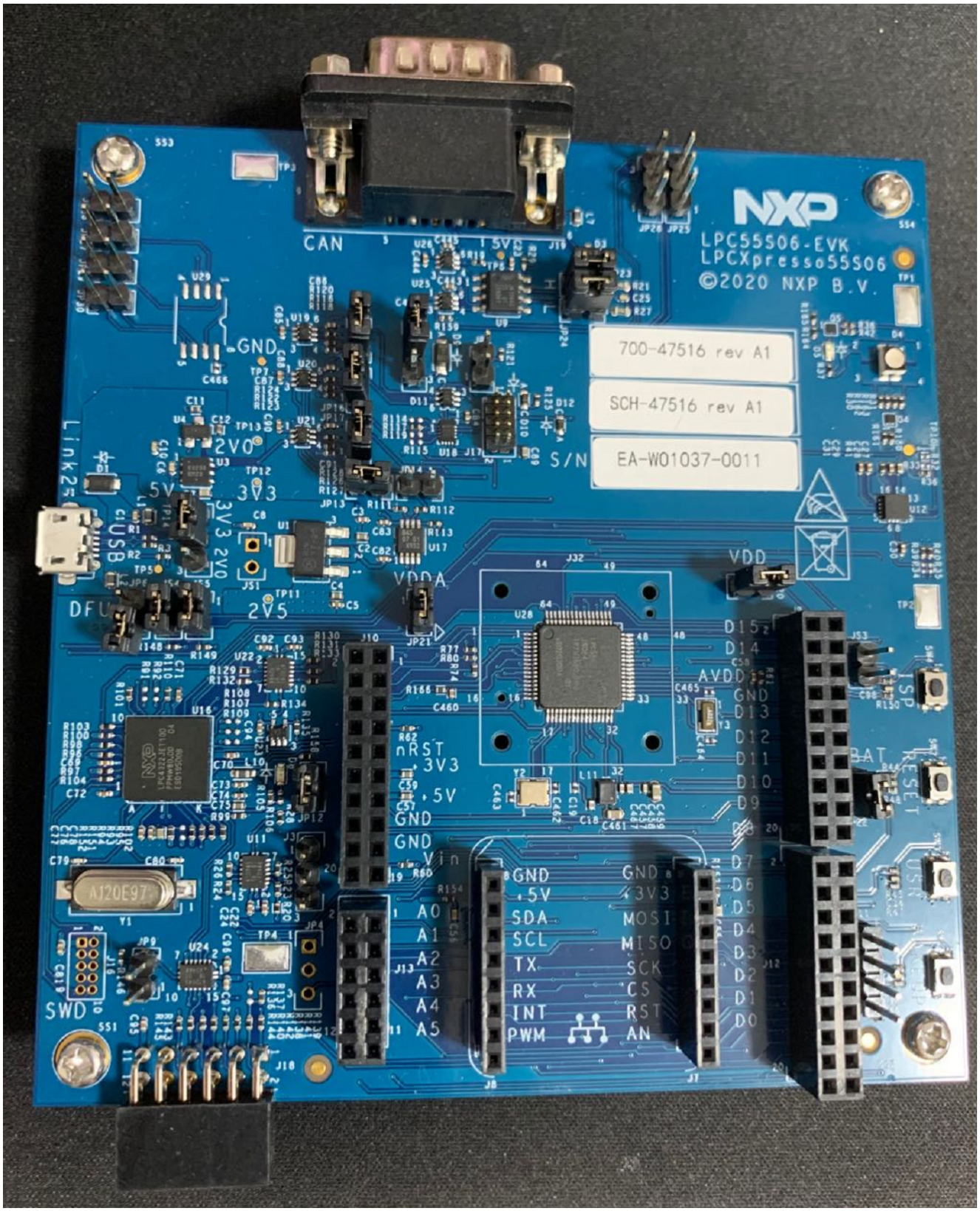

Figure 22. LPC55S06-EVK Development Board

The board ships with CMSIS-DAP debug firmware programmed. For more information on the CMSIS_DAP debug firmware, see the following FAQs:

https://www.nxp.com/downloads/en/software/lpc_driver_setup.exe

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

3.2.1 µA/MHz measurement setup

To measure the LPC55S0x/LPC550x power consumption, connect the ammeter across JP22, as shown in the following figure.

NOTE

- Users must remove the jumper on JP20, JP21, and JP22, and connect JP20/21/P22 pin2 together. Then use multi-meter to measure the current between JP20/21/22's pin1 to All JP20/21/22's pin2.
- The current data on EVK maybe little higher than datasheet, because the EVK have more other components that may cost more power.

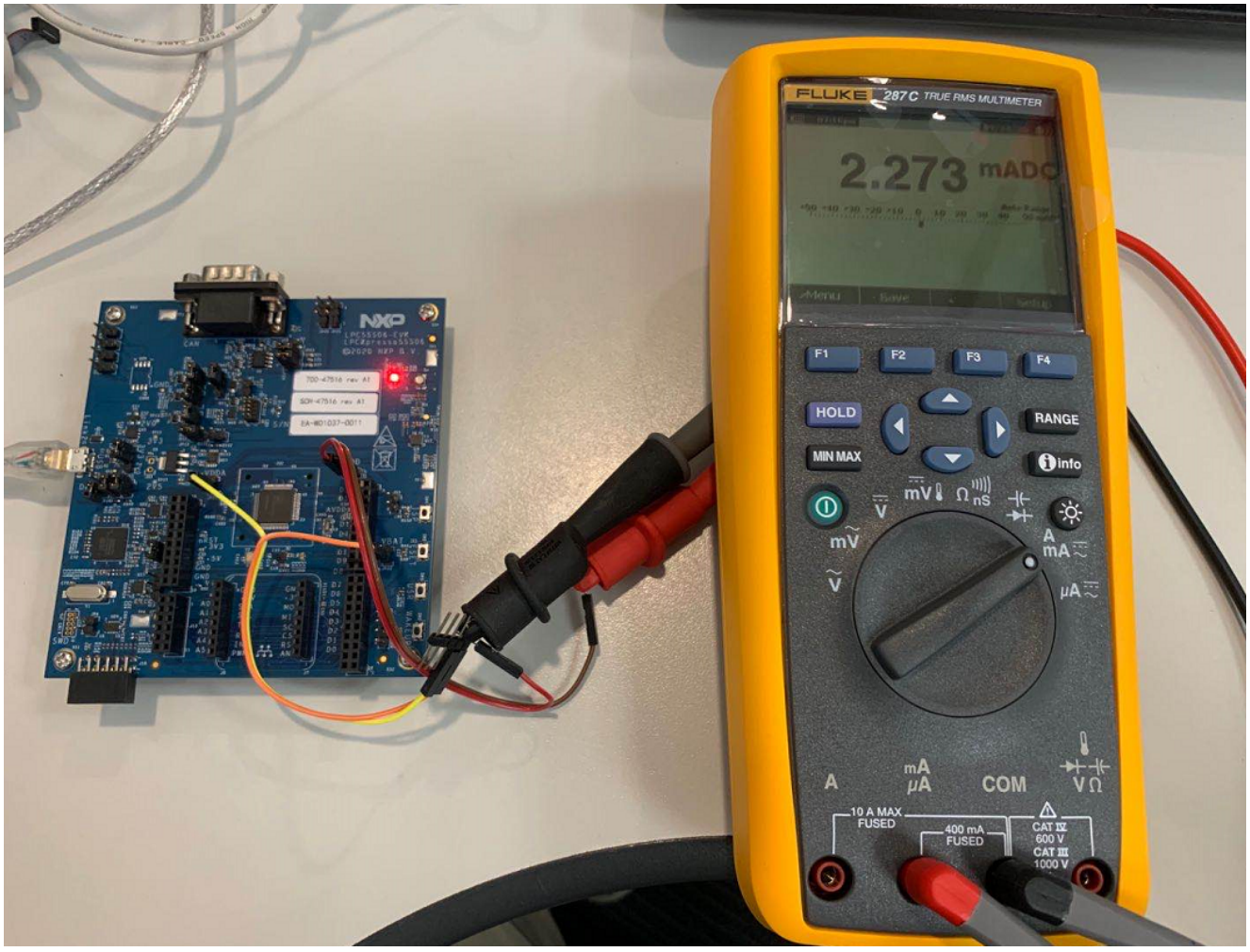

Figure 23. µA/MHz measurement setup

Users can measure the current by multimeter.

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

The core clock frequency can be changed by selecting different configurations through the shell terminal by MCU UART0.

3.3 Running CoreMark code

To obtain the CoreMark result, perform the following steps:

1. Connect the board's connector J1 with PC. Then, the PC will recognize the LPC-Link2 debugger with a Simulate Serial Port, as shown in the following figure.

   If the PC cannot find the serial port driver, download the LPCScrypt from the following link and install it on your PC. https://www.nxp.com/support/developer-resources/software-development-tools/lpc-developer-resources-/lpc-microcontroller-utilities/lpcscrypt-v2.0.0:LPCSCRYPT?tab=Design_Tools_Tab

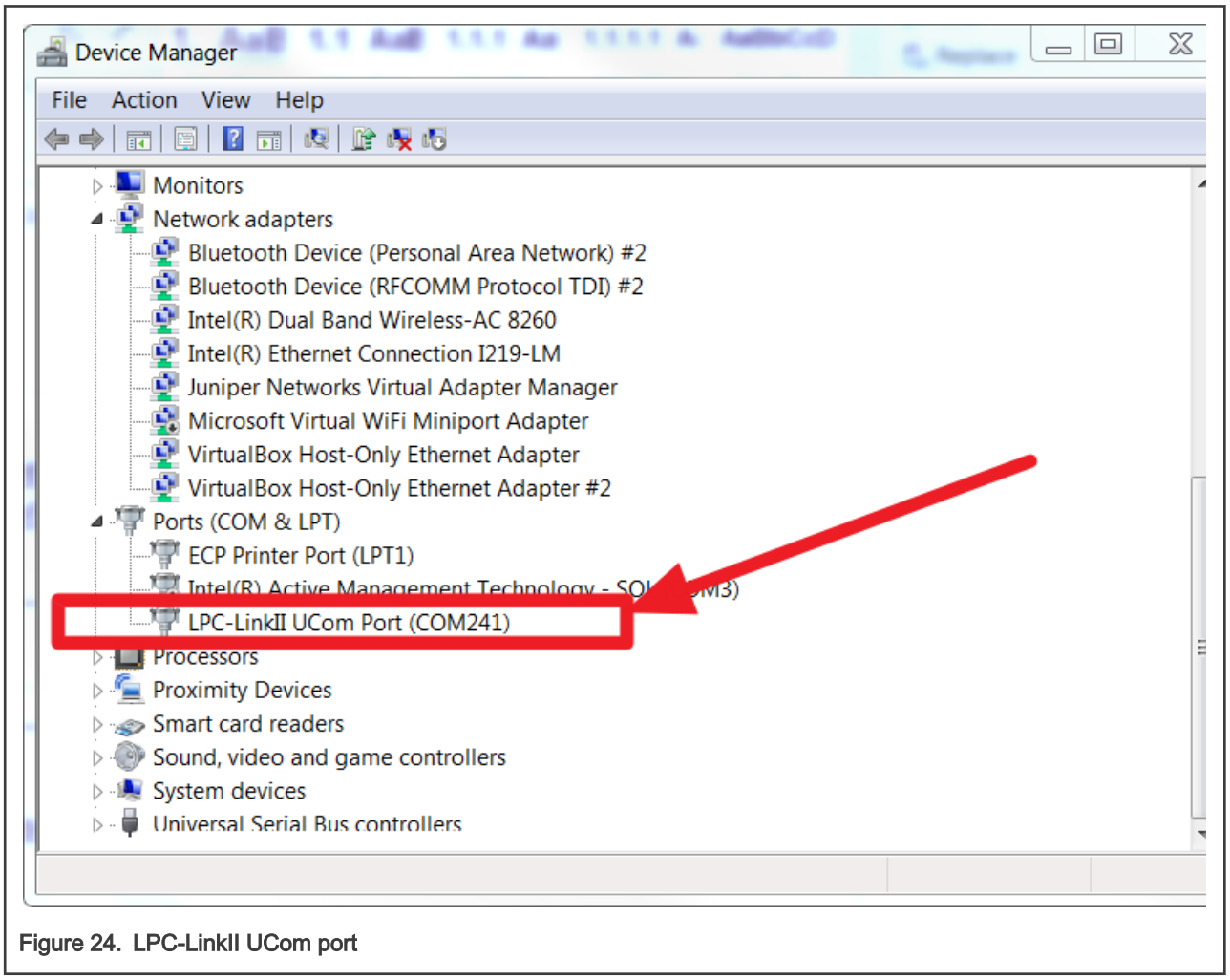

Figure 24. LPC-LinkII UCom port

2. Open a UART debug terminal (such as Tera Term, putty, etc.) and configure the settings as 115200, 8 data bits, no parity, 1 stop bit, as shown in the following figure.

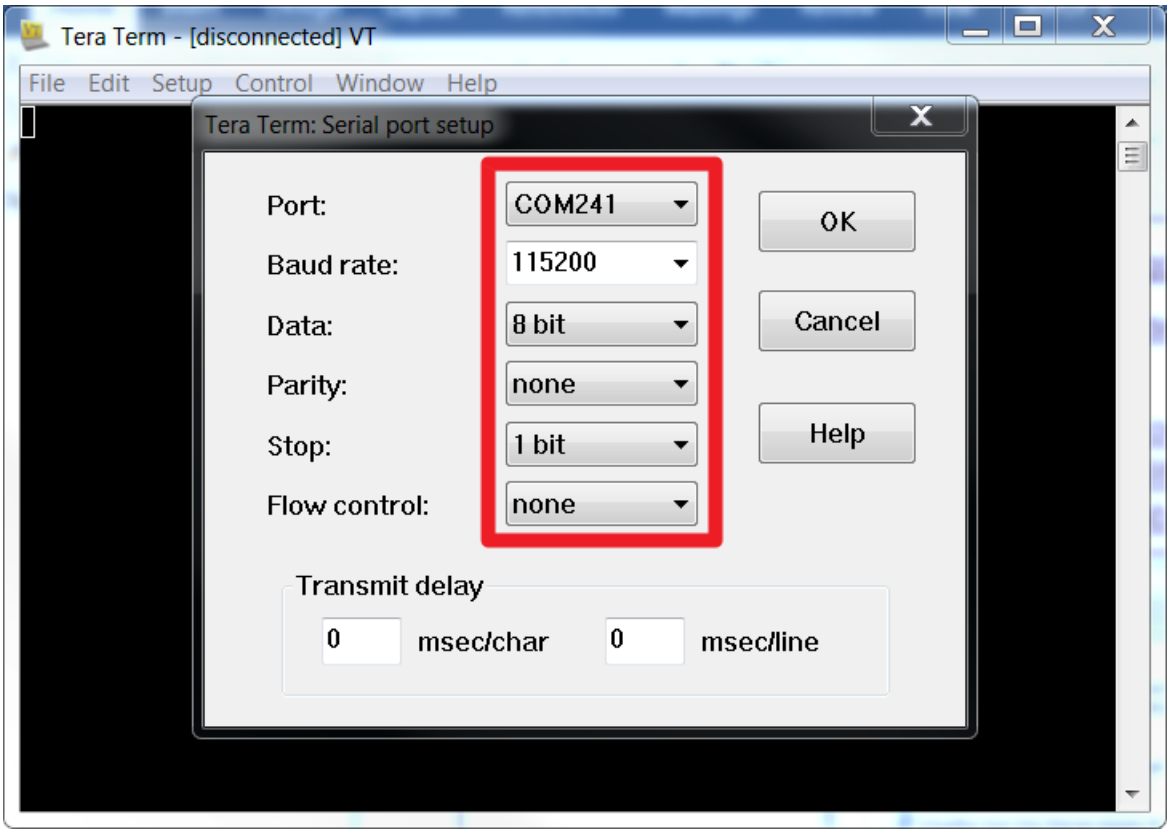

Figure 25. UART debug terminal configuration

3. Once the CoreMark necessary files are added into the project (by following the instructions in Chapter 2.1), compile the project and download it to the LPC55S06-EVK board.
4. Click the "Reset" button. The terminal displays the prompt information, as shown in the following figure. Users can input "1", "2", or "3" from the PC keyboard to select the Core frequency like FRO 12 MHz, FRO 96 MHz, PLL 96 MHz once input a character. The Coremark test program starts immediately. Then, wait for 10 senconds or more. The CoreMark benchmark is displayed on the terminal after a few seconds, as shown in the following figure in Chapter 4.

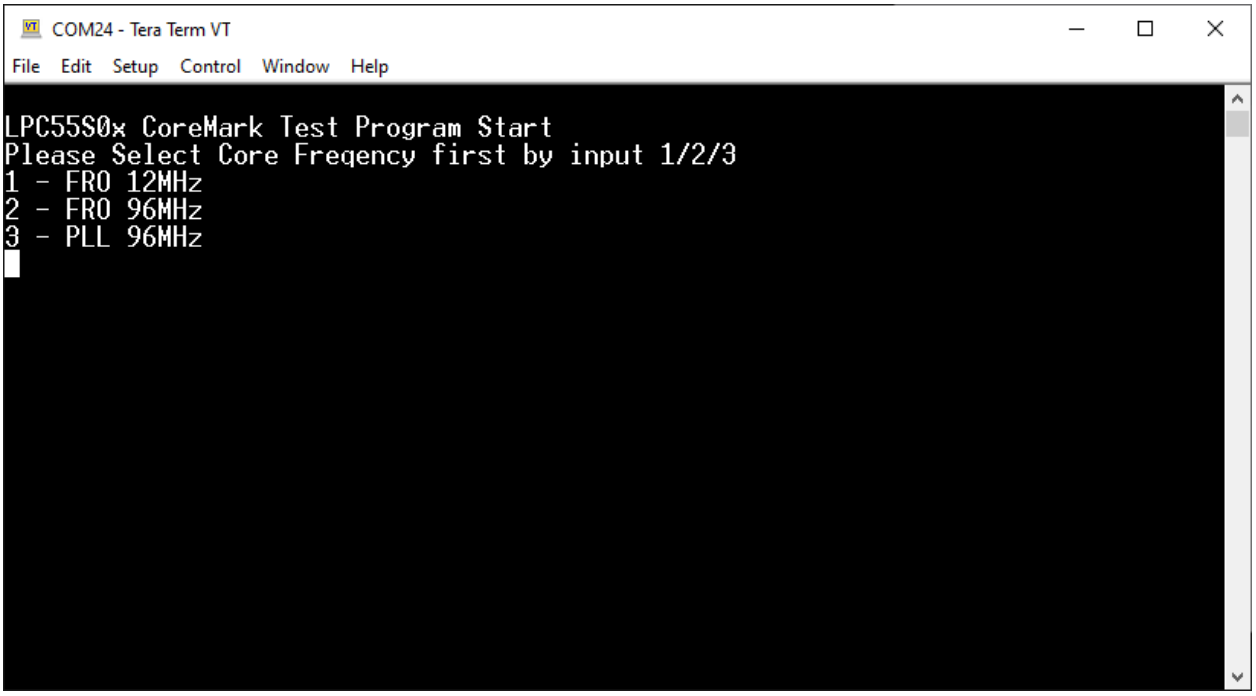

Figure 26. Coremark Test Core Frequency Choose menu

4 Result

The following figure shows the CoreMark benchmark result when running LPC55S0x/LPC550x at 96 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 390.52/96 MHz = 4.06CoreMark/MHz.

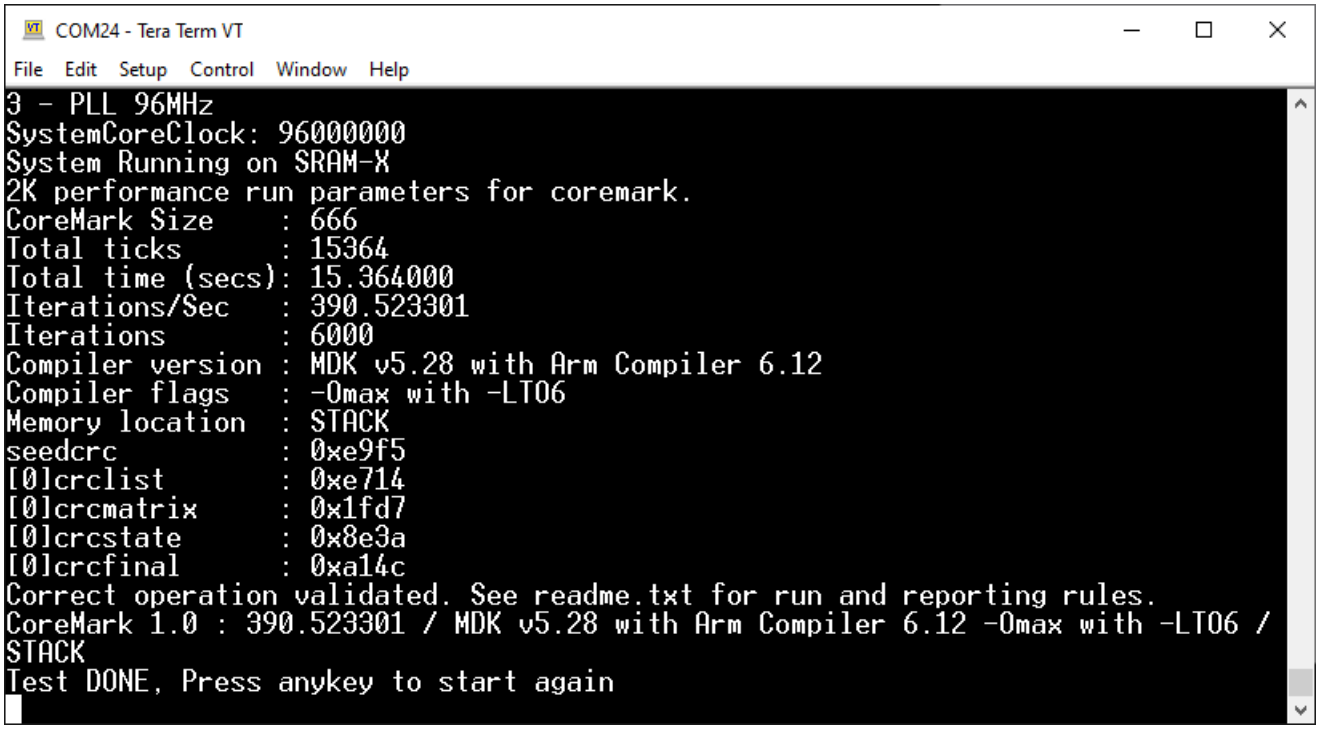

Figure 27. CoreMark result

The following table shows the typical CoreMark score when benchmarked on Keil MDK, IAR EWARM, and MCUXpresso IDE when running from the internal flash and SRAM when using FRO 12 MHz as the core clock resource.

Table 1. LPC55S06-EVK board CoreMark/MHz score when using FRO 12 MHz

IDE	CoreMark/MHz Score (SRAMX)	CoreMark/MHz Score (Flash)
KEIL MDK	4.05	3.79
IAR EWARM	3.88	3.68
MCUXpresso	2.96	2.84

The following table shows the typical CoreMark score when benchmarked on Keil MDK, IAR EWARM, and MCUXpresso IDE when running from the internal flash and SRAM when using FRO 96 MHz as the core clock resource.

Table 2. LPC55S06-EVK board CoreMark/MHz score when clock resource is FRO 96 MHz

IDE	CoreMark/MHz Score (SRAMX)	CoreMark/MHz Score (Flash)
KEIL MDK	4.02	2.53
IAR EWARM	3.89	2.64
MCUXpresso	2.96	2.25

The following table shows the typical CoreMark score when benchmarked on Keil MDK, IAR EWARM, and MCUXpresso IDE when running from the internal flash and SRAM when using PLL 96 MHz as the core clock resource.

Table 3. LPC55S06-EVK board CoreMark/MHz score when clock resource is PLL 96 MHz

IDE	CoreMark/MHz Score (SRAMX)	CoreMark/MHz Score (Flash)
KEIL MDK	4.02	2.53
IAR EWARM	3.89	2.64
MCUXpresso	2.96	2.25

For µA/MHz, the following tables show typical results when running on the LPC55S06-EVK board with VDD = 3.3 V at room temperature.

NOTE

The current data on the EVK may be little higher or lower than the datasheet, because the EVK has more components that may cost more power.

Table 4. 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)
FRO 12 MHz	1.01	84.20	1.18	98.34
FRO 96 MHz	3.07	32.00	3.35	34.90
PLL 96 MHz	3.34	34.80	3.58	37.30

Table 5. 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)
FRO 12 MHz	1.06	88.34	1.14	95.00
FRO 96 MHz	2,84	29.59	3.07	31.98
PLL 96 MHz	3.14	32.71	3.37	35.11

Table 6. 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)
FRO 12 MHz	1.09	90.84	1.16	96.67
FRO 96 MHz	2.98	31.05	3.22	33.55
PLL 96 MHz	3.24	33.75	3.52	36.67

5 Conclusion

In this application note, three types of CoreMark benchmarking on the LPC55S0x/LPC550x are presented 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 LPC55S06-EVK. The best CoreMark number is **4.06**, achieved by using KEIL MDK (Arm Compiler 6.12) and running CoreMark from SRAM X. The best CoreMark power consumption in **µA/MHz is 29.59**, achieved by running CoreMark from SRAM when the core frequency is FRO 96 MHz.

6 Reference

1. *CoreMark Benchmarking for ARM Cortex Processors*
2. *LPC5411x CoreMark Cortex-M4 Porting Guide* (document AN11811)
3. *LPC55S0x/LPC550x Data Sheet, Rev. 1.0* (document LPC55S0x/LPC550x)
4. *LPC55S0x/LPC550x User Manual, Rev. 0.4* (document UM11424)

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

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

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

Date of release: 30 October 2020

Document identifier: AN13035

arm