AN12284

LPC55xx CoreMark on Cortex-M33 Porting Guide

Rev. 1 — December 2019 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.

LPC55xx is an Arm® Cortex® -M33 based microcontroller for embedded applications. These devices include:

- An Arm Cortex-M33 coprocessor
- CASPER Crypto/FFT engine
- PowerQuad hardware accelerator for DSP functions
- Up to 320 KB of on-chip SRAM, up to 640 KB on-chip flash
- PRINCE module for on-the-fly flash encryption/decryption
- High-speed and full-speed USB host and device interface with crystal-less operation for full-speed, SDIO/MMC
- 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 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 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 LPC55xx, 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 herewith for reference.

2 Integration of CoreMark library to SDK2.0 framework

The software package associated with this application note contains SDK2.0 based project framework. It allows developers to drop in the CoreMark library sources and quickly get up and running with benchmarking the LPC55xx. To get started, go to: https://www.eembc.org/coremark, Click the **Download** link as shown in Fig. 1 and follow the instructions on the page.

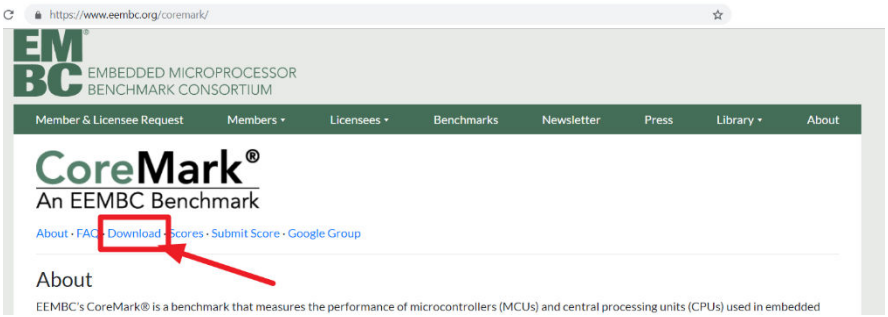


Figure 1. EEMBC CoreMark download link webpage

After reviewing the license terms, go through the readme and documentation file. The readme provides step-by-step instructions on unpacking and building the distribution. It also helps in getting familiar with the CoreMark terminology used throughout the application note.

2.1 Porting CoreMark library into CoreMark framework

There are two variants of CoreMark projects for each IDE. One executes the CoreMark application from internal flash and other executes the CoreMark application from internal SRAMX

The CoreMark projects are:

1. run_in_flash_xxmhz – Cortex-M33 executes CoreMark application from internal flash.

2. run_in_ramx_xxmhz – Cortex-M33 executes CoreMark application from internal RAM.

The locations of CoreMark projects are:

Keil MDK IDE :

- lpc5500_coremark_mdk\coremark.uvprojx.eww

IAR Workbench IDE:

- lpc5500_coremark_iar\coremark.eww

Each of executes settings have four frequency settings : 12 MHz, 48 MHz, 96 MHz and 150 MHz.

Depending on the toolchain, the workspace should look as shown in below 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 run_in_xxxx_xxmhz project must be set as active before the CoreMark files can be added.

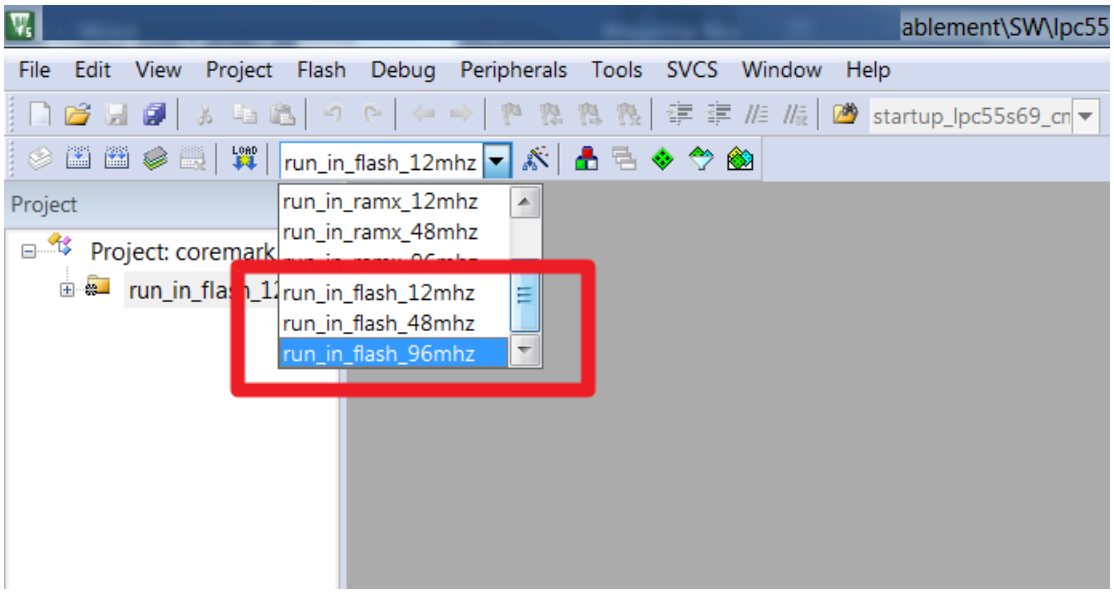


Figure 2. Keil MDK CoreMark framework

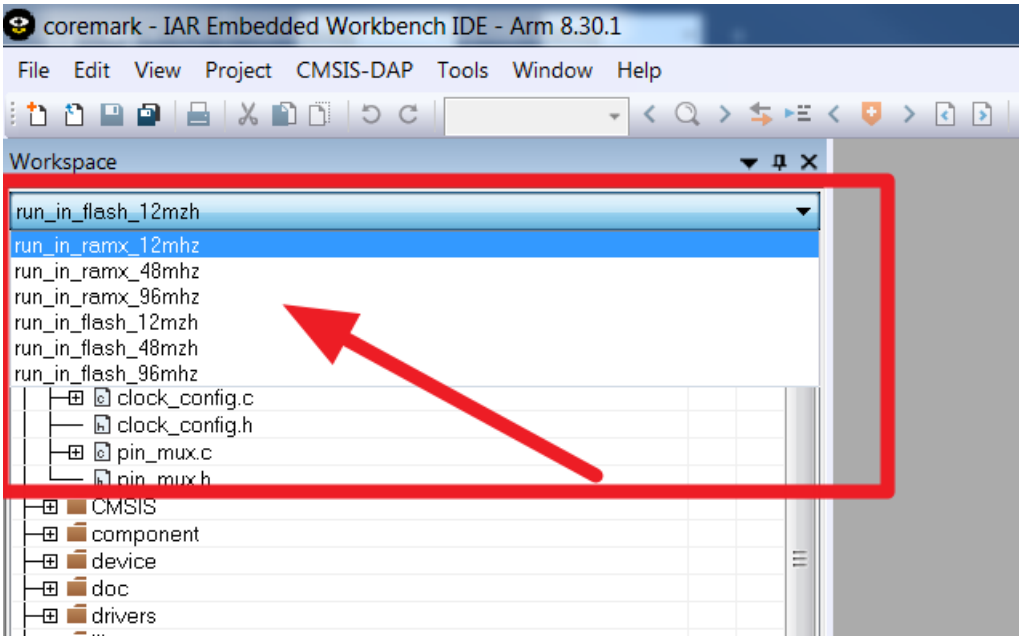


Figure 3. IAR EWARM workspace

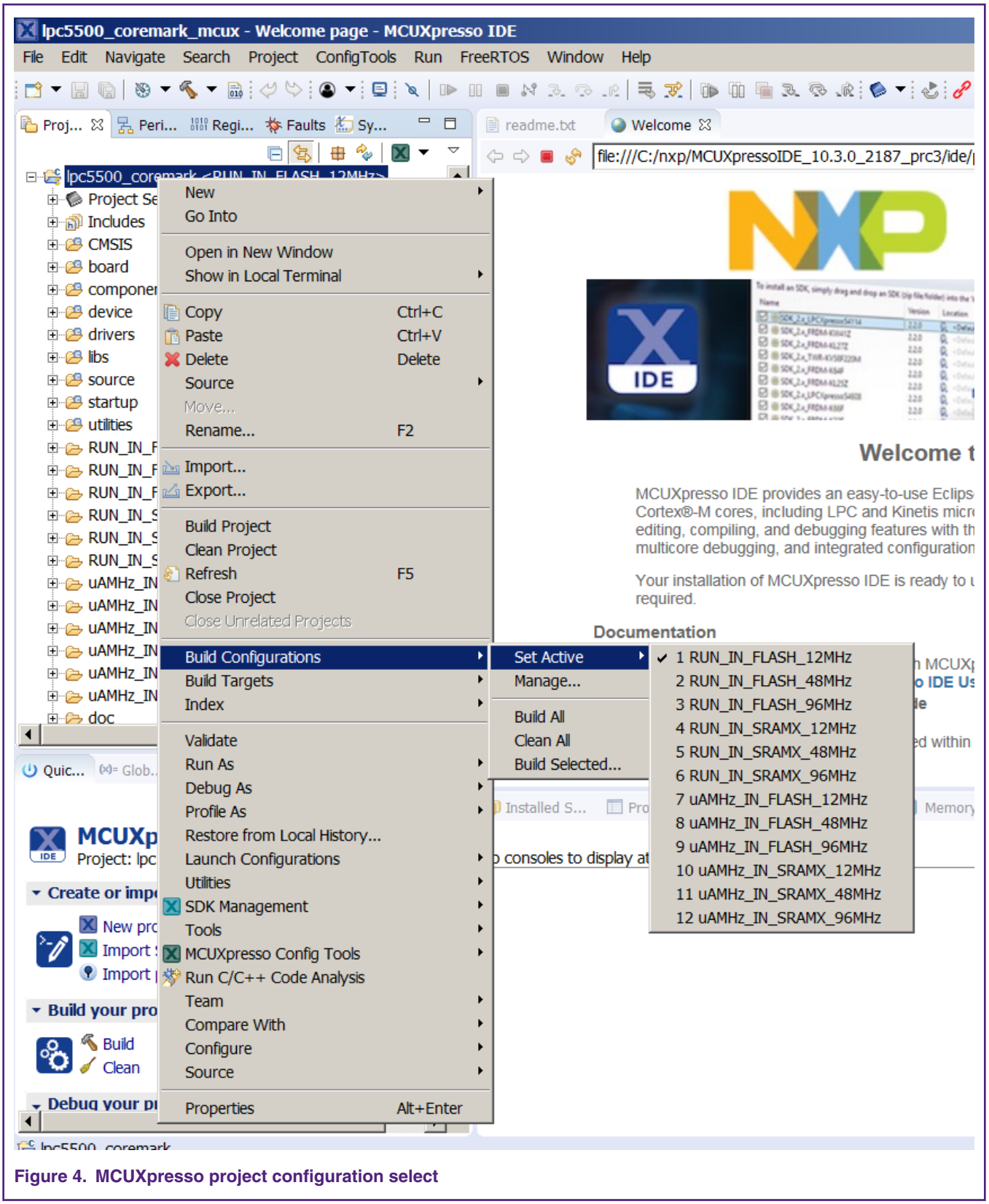


Figure 4. MCUXpresso project configuration select

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

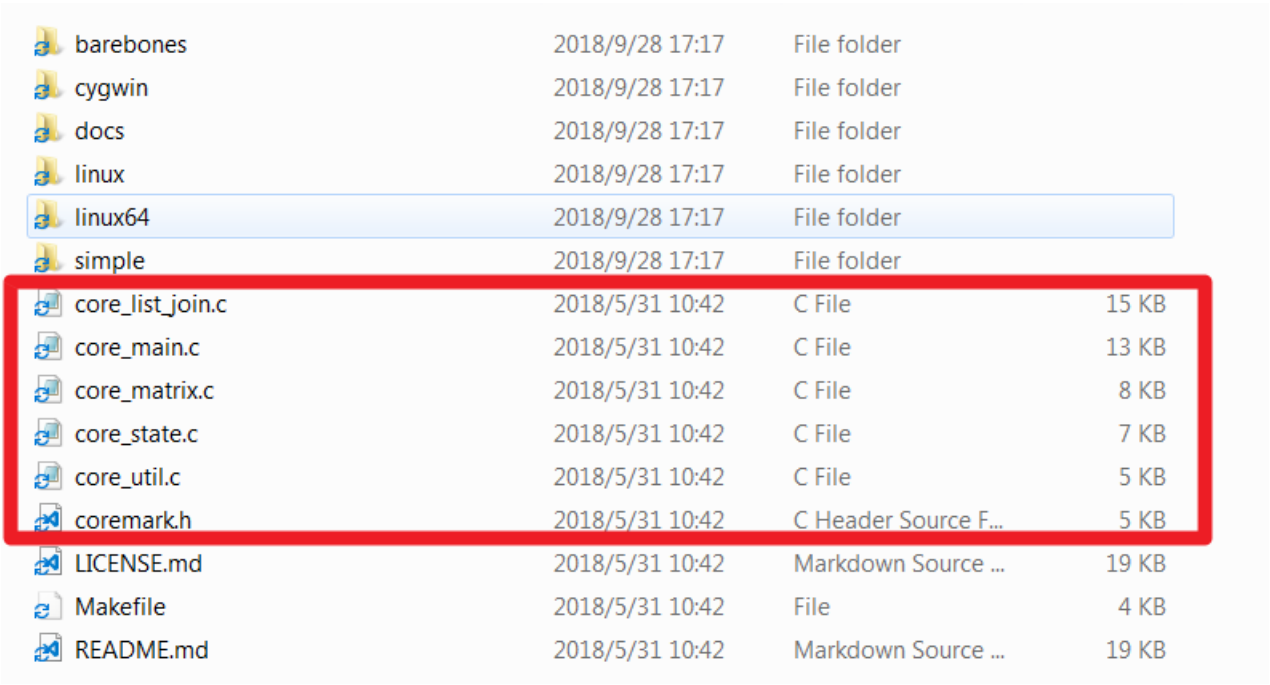


Figure 5. CoreMark files to copy

-For Keil MDK place these files in the project directory

lpc5500_coremark_mdk\source

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

lpc5500_coremark_iar\source

-For MCUXpresso place these files in the project directory.

lpc5500_coremark_mcux\source

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

-For Keil IDE place the files in “lpc5500_coremark_mdk\source”

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

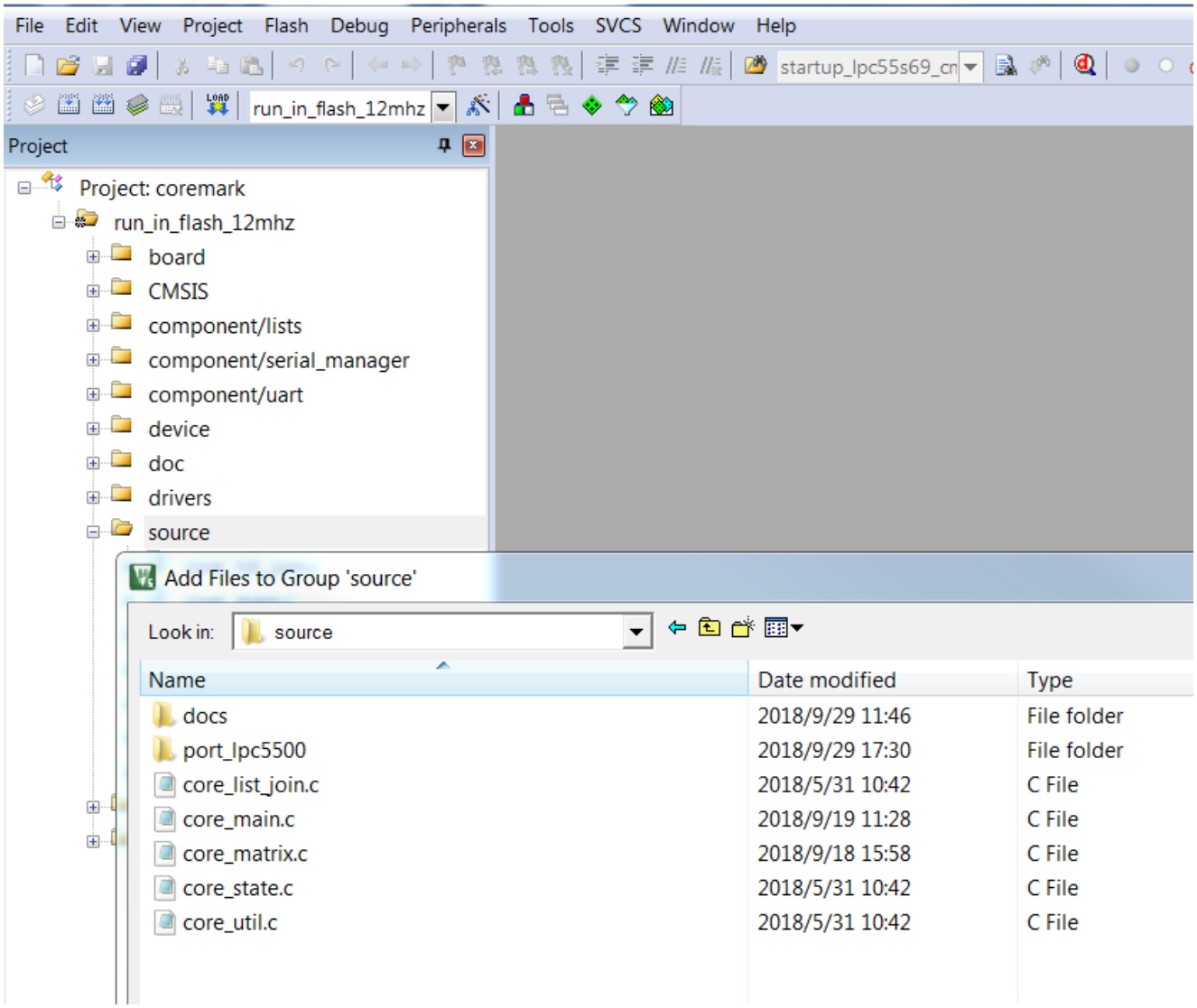
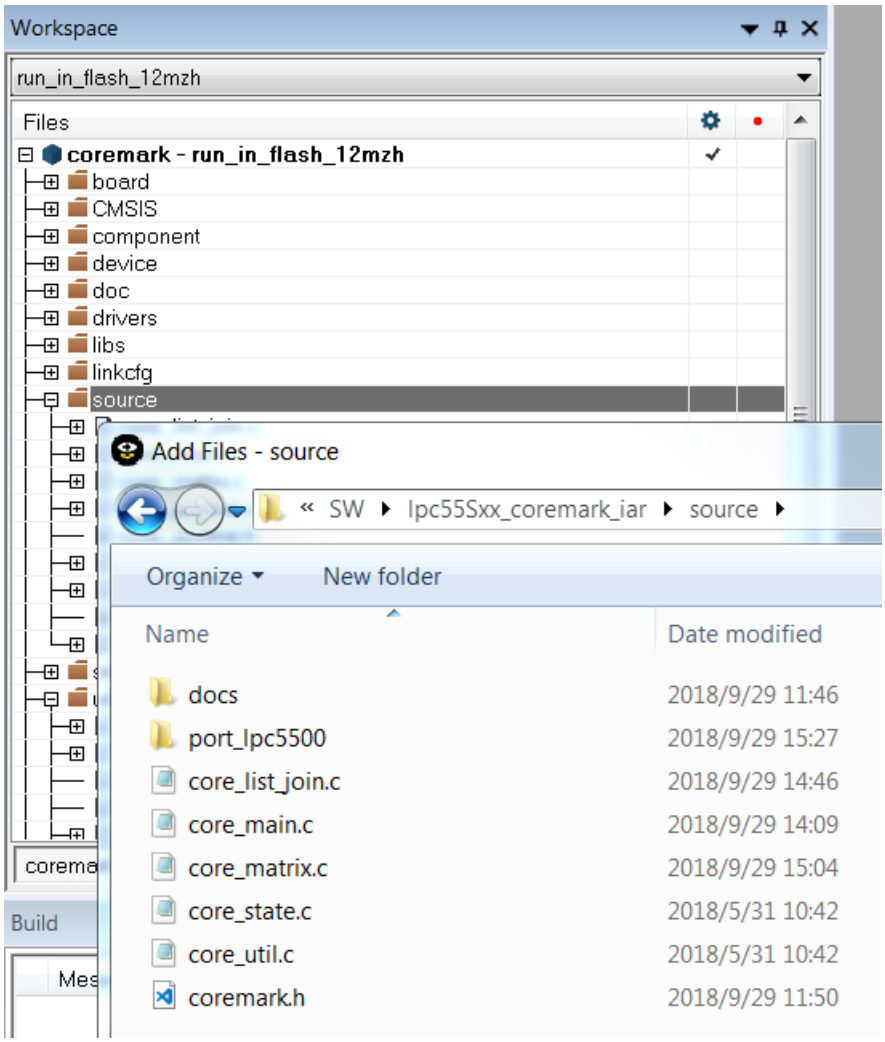


Figure 6. Adding files in Keil MDK

-For IAR Embedded workbench place the files in “lpc5500_coremark_iar\source”

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 “lpc5500_coremark_mcux\source”

Add the files into the MCUXpresso project framework to the respective groups source by click the “refresh” selection.

Figure 7. Adding files in IAR EWARM workspace

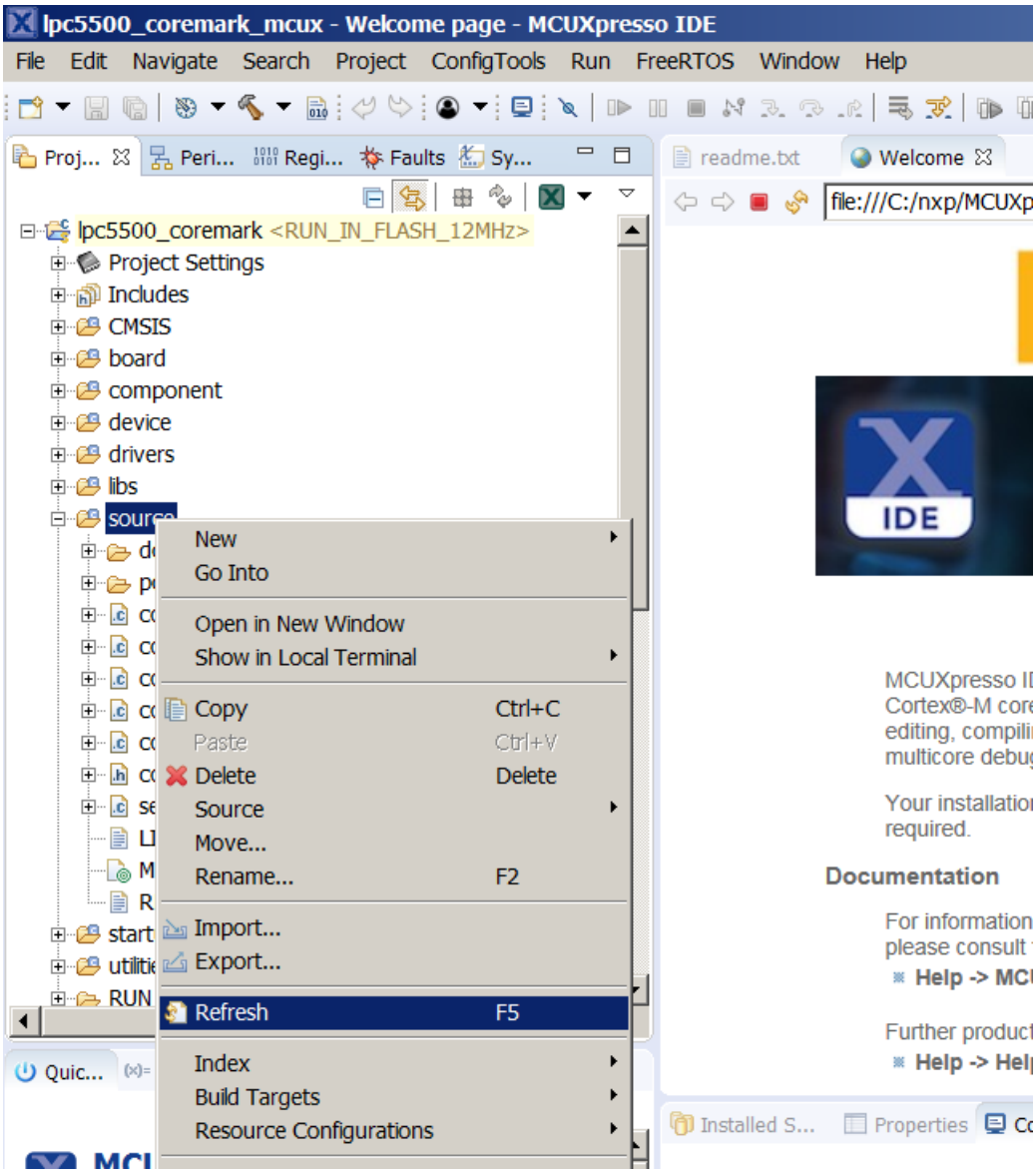


Figure 8. Adding files in MCUXpresso project

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.

Once all the files have been added, the workspace should look as shown below:

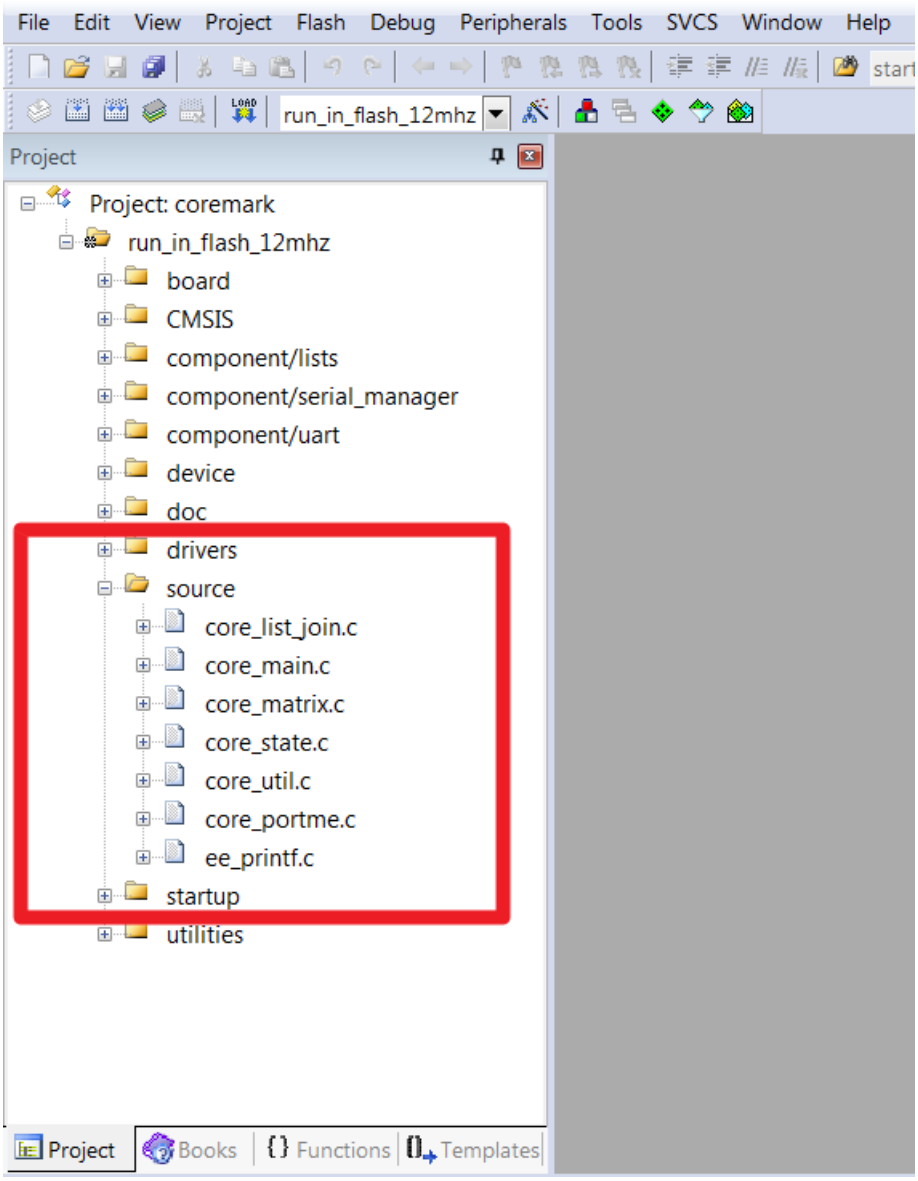


Figure 9. Keil MDK project workspace after adding CoreMark files

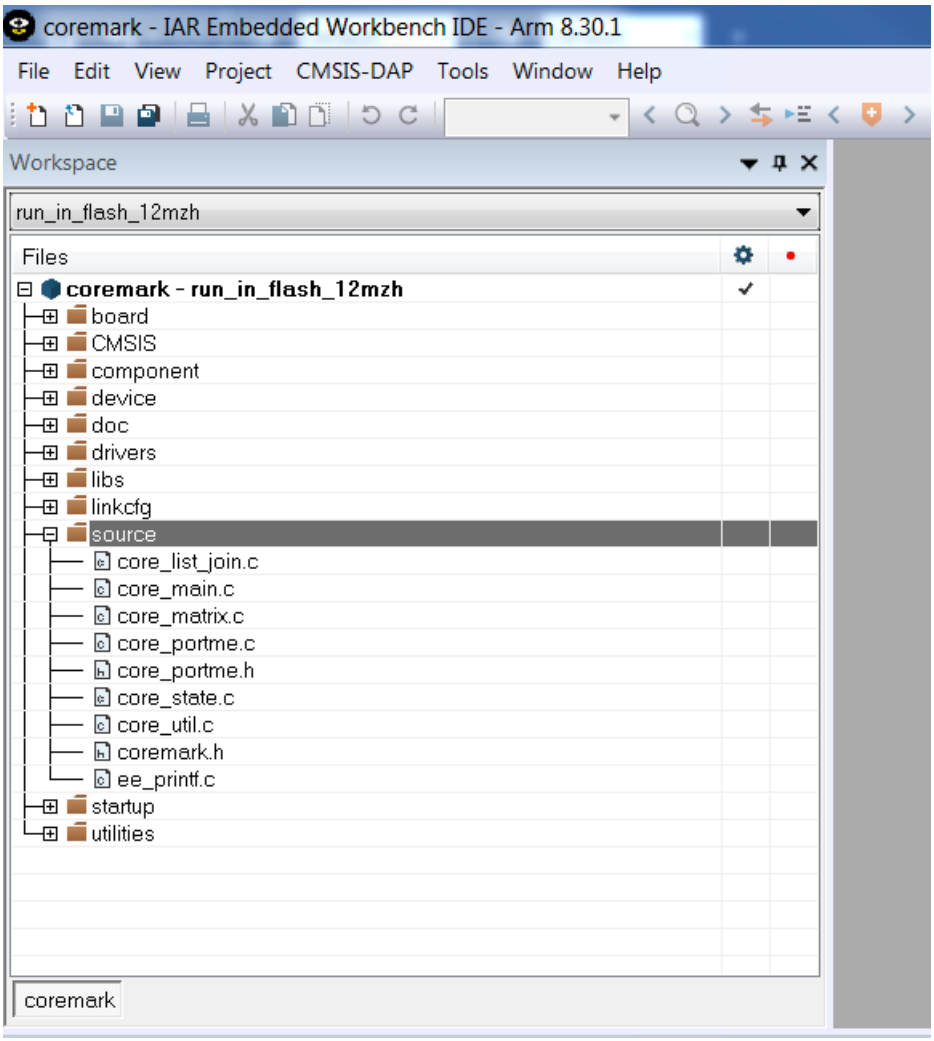


Figure 10. IAR EWARM workspace after adding CoreMark files

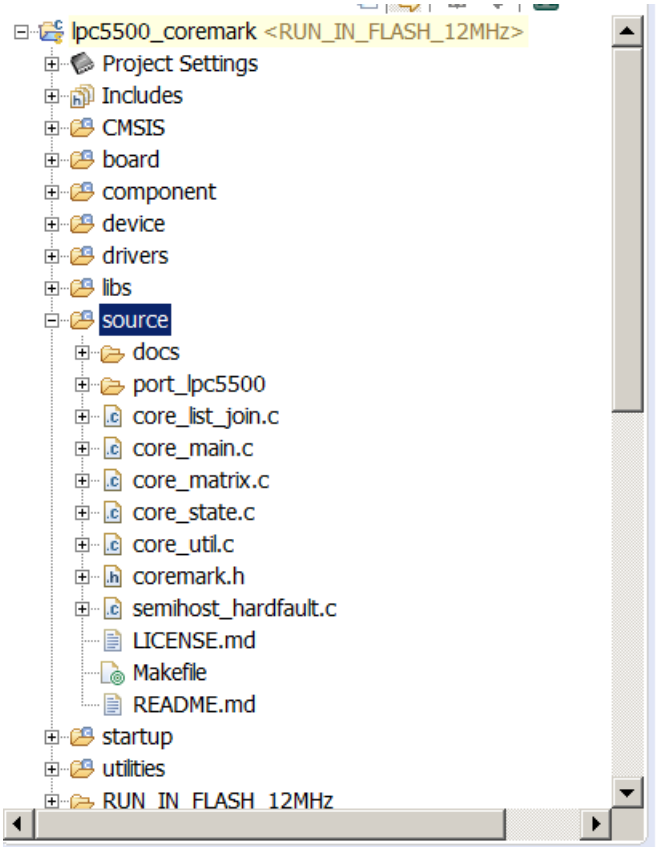


Figure 11. MCUXpresso project after adding CoreMark files

A few files need to be modified to support CoreMark and are described below. In the project scatter file change the stack size as 0x2000.

```
define symbol __size_cstack__        = 0x2000;
```

To support 'printf' statements to a PC terminal, the 'core_portme.h' file needs to be modified. Add the following line of code for ee_printf function.

```
#if HAS_PRINTF
#else
#ifdef COREMARK_SCORE_TEST
#define ee_printf printf
#else
extern int ee_printf_template(const char *fmt, ...);
#define ee_printf ee_printf_template
#endif
#endif
#endif
```

In 'eeprintf.c' file, add #ifdef COREMARK_SCORE_TEST and the function ee_printf(const char *fmt,…).

```
#ifndef COREMARK_SCORE_TEST
int ee_printf_template(const char *fmt, ...)
{
  return 0;
```

```
}
#endif
```

This is added so that the printf code is optimized when running the µA/MHz test. In ‘core_portme.h’ there is a #define COREMARK_SCORE_TEST that dictates whether or not the application is executing the CoreMark score test.

In order to add the path to the header files used in the project, in Keil MDK under Project->Options-> C/C++(AC6) tab, click ‘Include path’ and add the following paths that contain the header files.

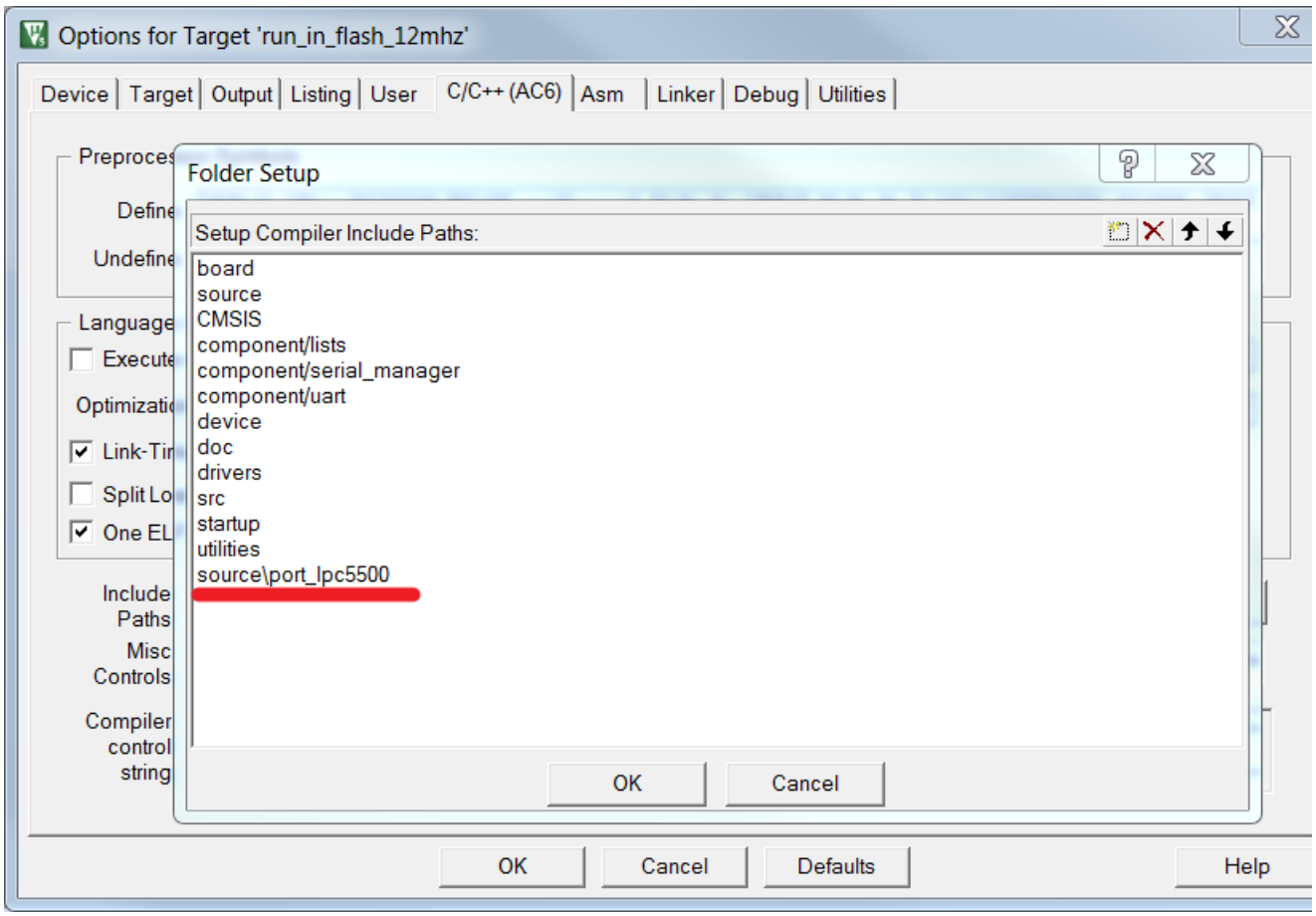


Figure 12. Keil MDK compiler include paths

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

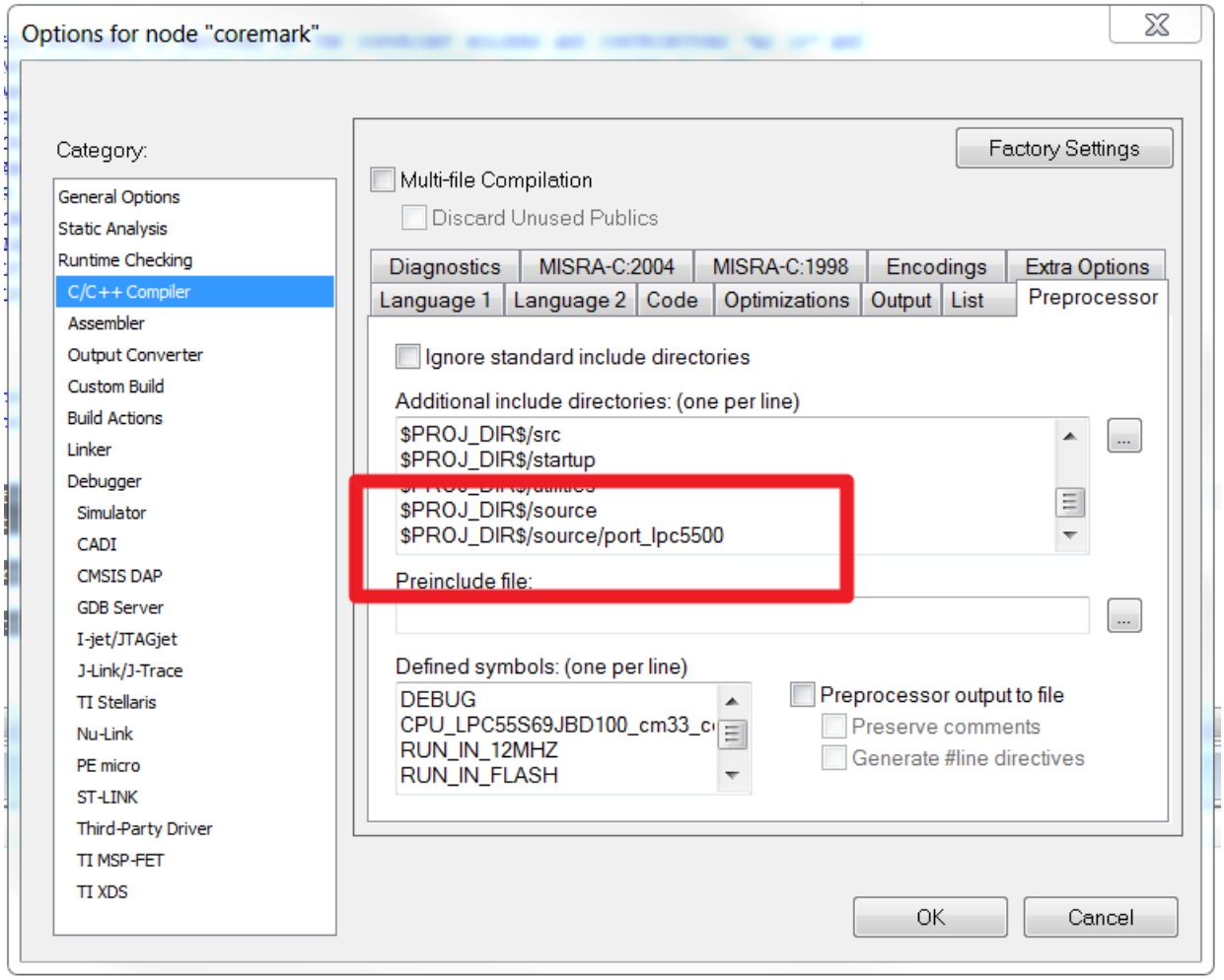


Figure 13. IAR EWARM compiler include paths

The CoreMark files have now been 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 contains the header files.

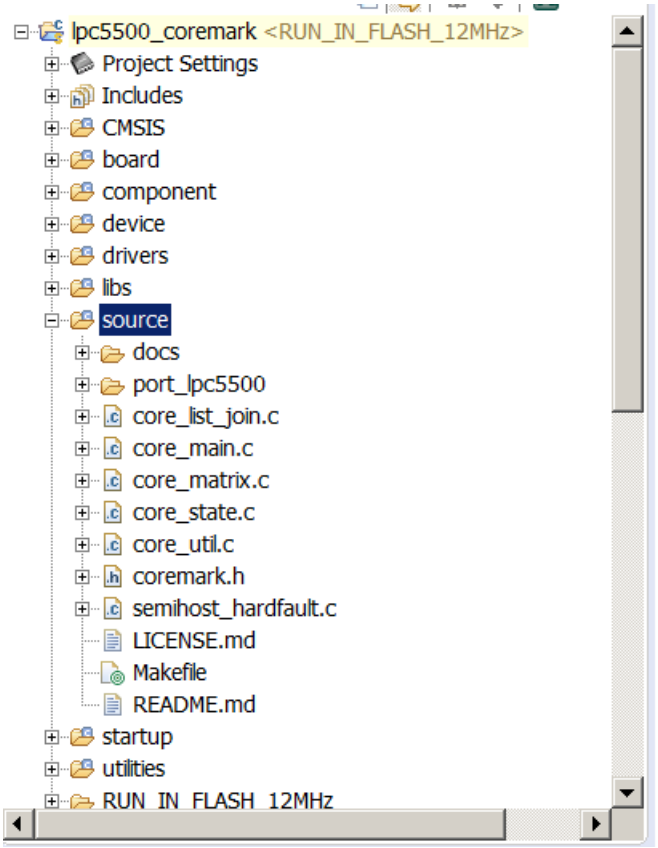


Figure 14. MCUXpresso compiler include paths

The CoreMark files have now been successfully ported into the CoreMark project framework.

2.1.2 CoreMark framework to execute from Internal SRAM

The project run_in_ram_xxmhz executes the CoreMark application from 32 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:

.\lpc5500_coremark_mdk\LPC55S69_cm33_core0_ramx.scf

The linker script setting for run_in_ramx_xxmhz project is shown in Fig 15

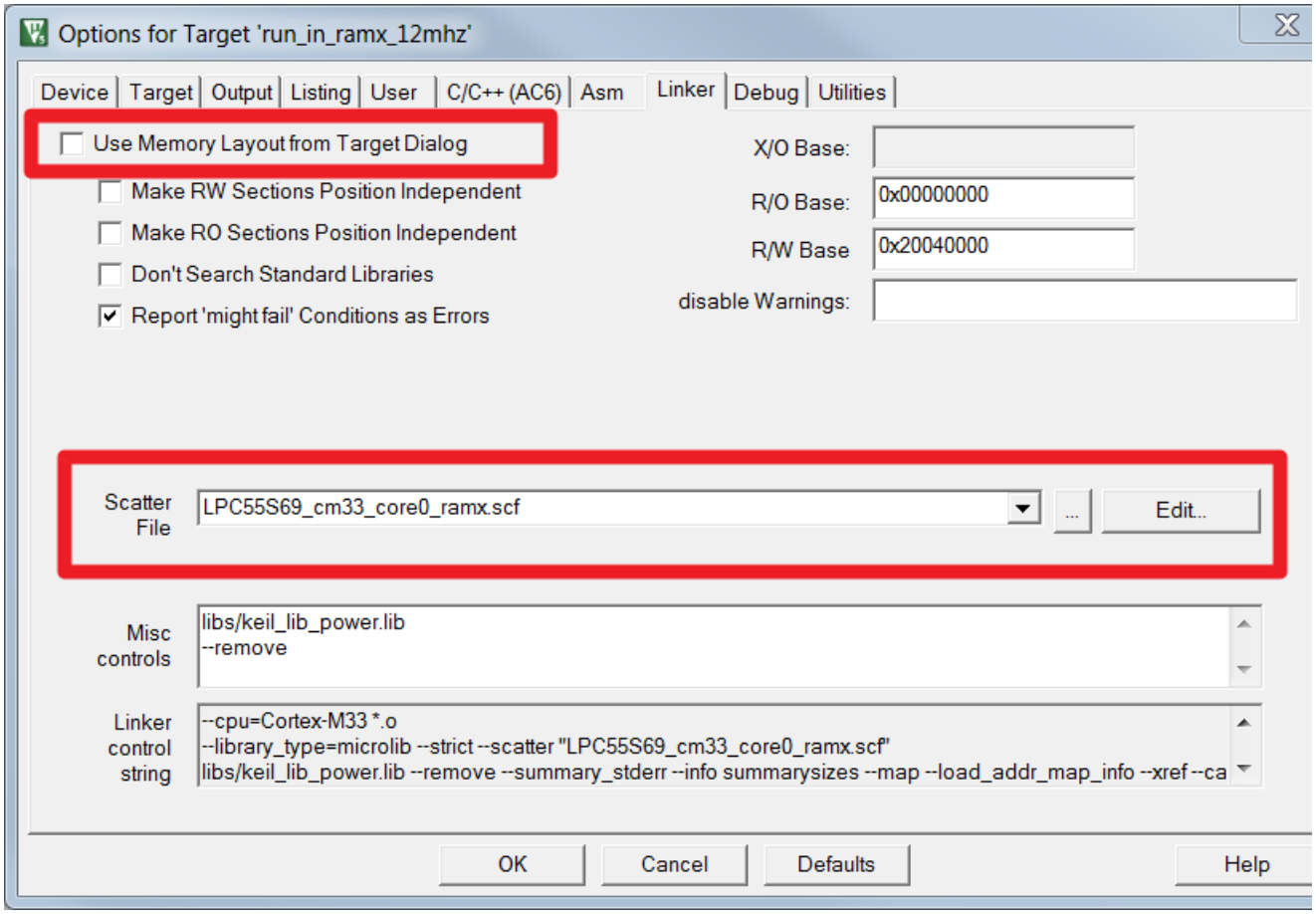


Figure 15. Linker script in Keil IDE

For IAR EWARM IDE to execute CoreMark in Internal SRAM, a line of code needs to be added to the files core_main.c, core_util,c, core_state.c, core_matrix.c and core_list_join.c, as Fig 18 shows, above #include in all five files.

These CoreMark files are labeled as their own IAR EWARM linker “section”. The provided .icf linker file in .\lpc5500_coremark_iar \LPC55S69_cm33_core0_ramx.icf

then places this section, which is called “critical_text” into SRAMX. To do this, add the following line of code in icf file, as shown in Fig 16.

```
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 16. IAR EWARM allocate Code to SRAM area

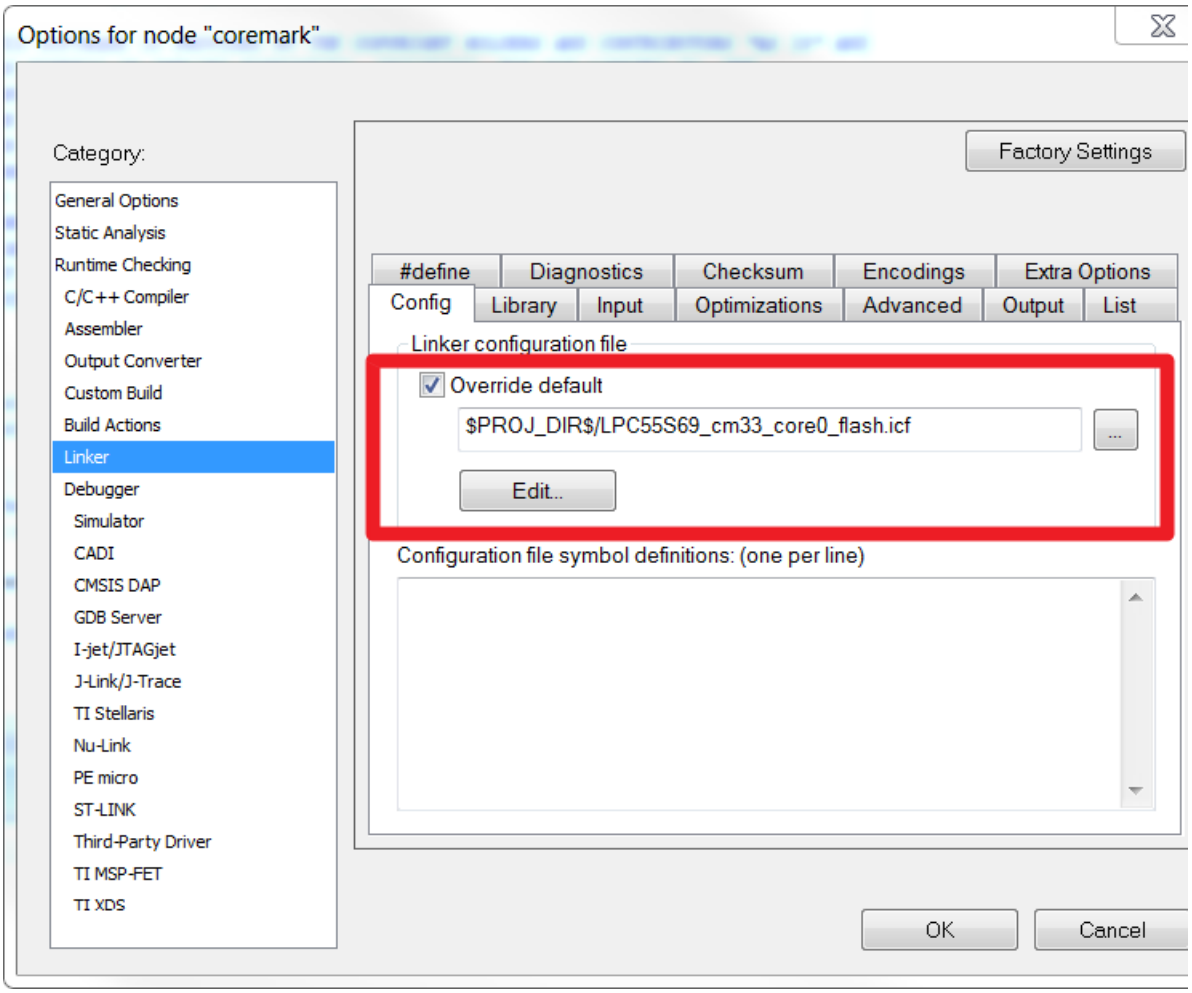


Figure 17. Linker script in IAR EWARM

Added below code in the files header area

```
#if RUN_IN_RAMX
#pragma default_function_attributes = @ ".critical_code"
#endif
```

Added below code in the files end

```
#if RUN_IN_RAMX
#pragma default_function_attributes =
#endif
```

Figure 18. IAR EWARM #pragma command

For MCUXpresso to execute CoreMark in Internal SRAM, just selected the linker file as “lpc5500_coremark_RUN_IN_SRAMX.ld” in “Managed Linker Script”,as shown in Fig 19..

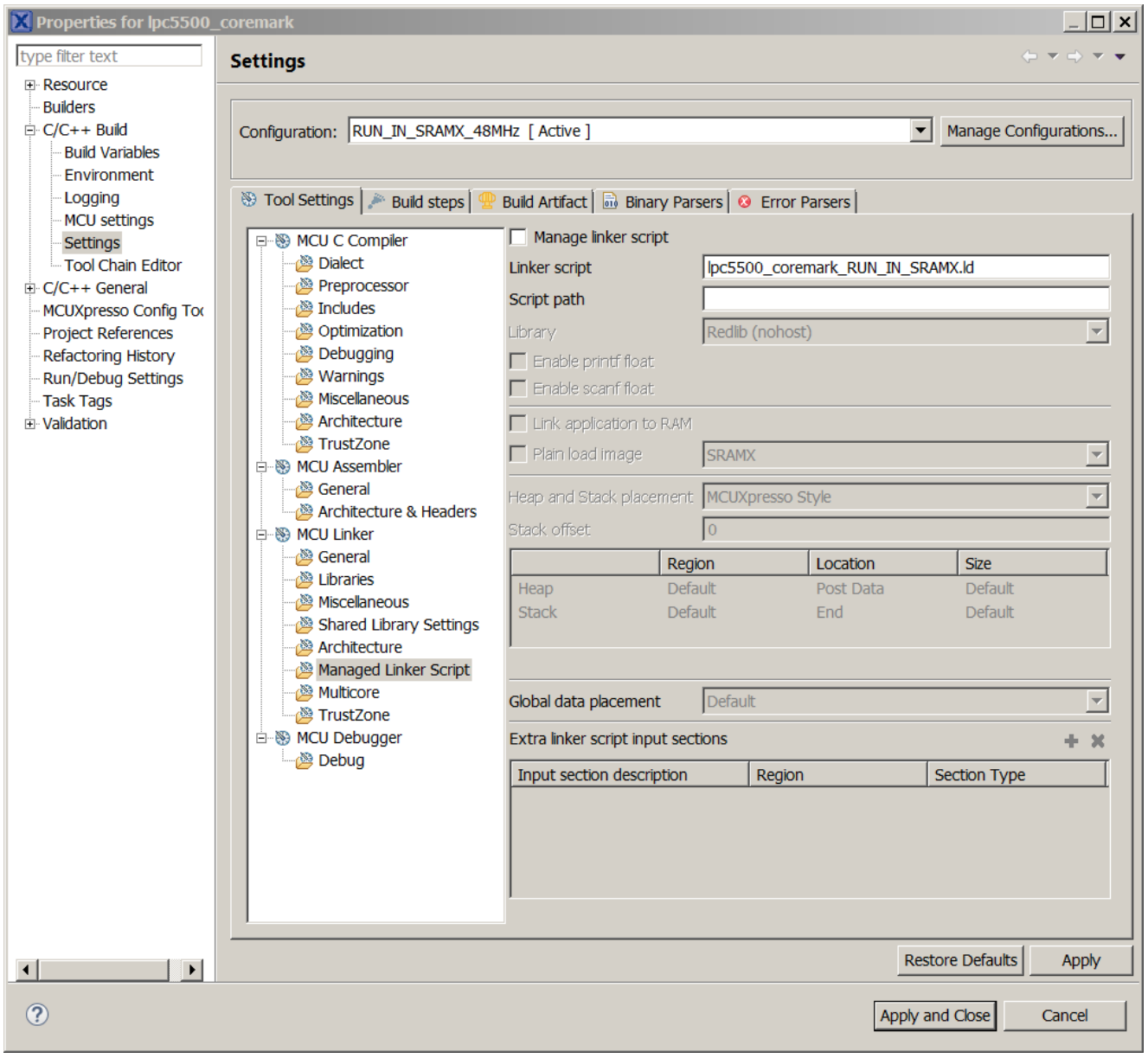


Figure 19. MCUXpresso allocate Code to SRAM area

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 be able 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 following are the IDE versions that are applicable to this application note:

Keil MDK v5.28

IAR EWARM 8.40.2

MCUXpresso 11.0.1_2563

2.2.1 Memory considerations

Due to the inherent architecture of SRAM and flash, CoreMark executes faster when running out of SRAM. The LPC55xx internal memory uses a multilayer AHB matrix system that provides a separate instruction and data bus for Cortex-M33 and SRAMX bank. See Fig 20. SRAM0 to SRAM4 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 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 prefetch ability of flash. The LPC55xx user manual contains more information on configuring the flash memory, such as the minimum number 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.

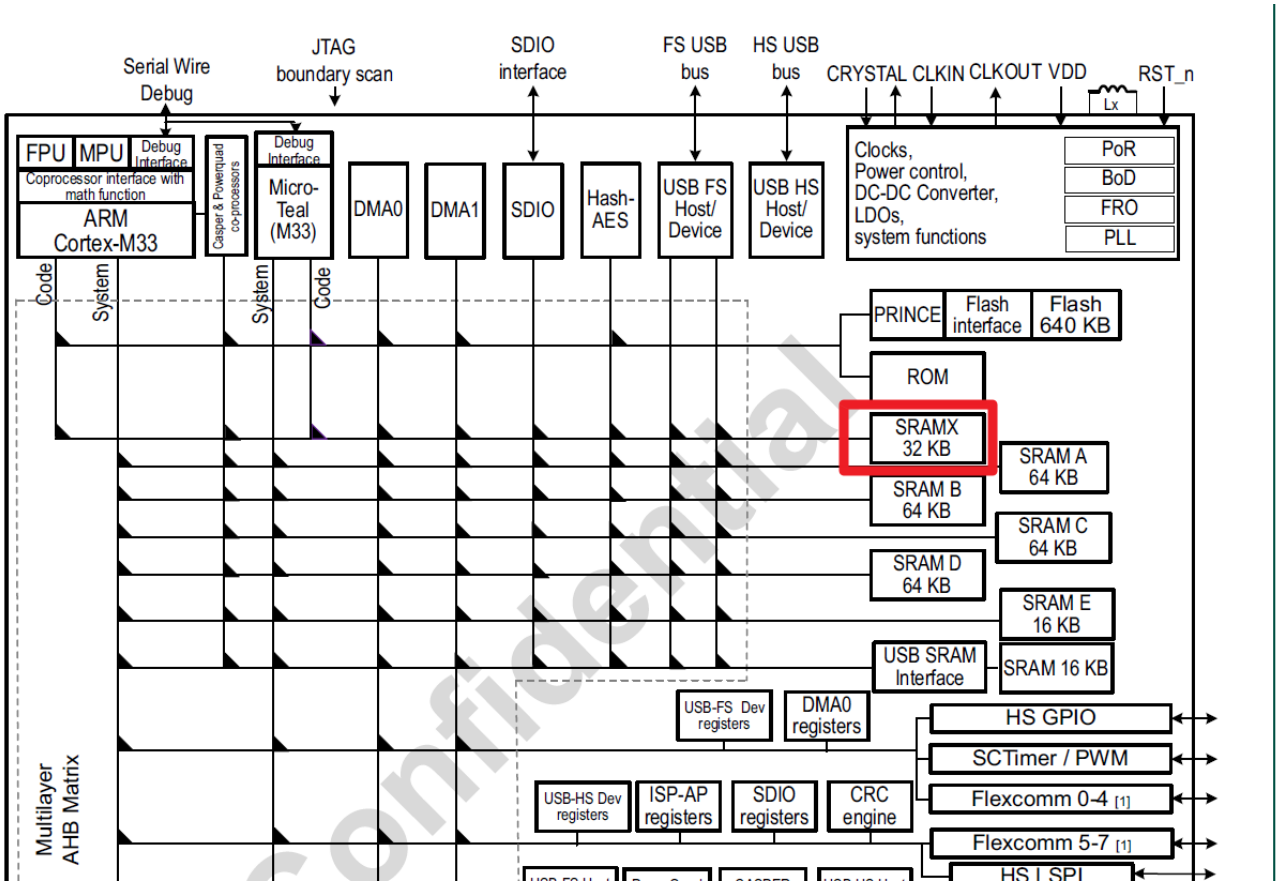


Figure 20. LPC55Sxx AHB matrix

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 runs. If this macro is commented out, the µA/MHz test runs. Use this macro to switch between the two benchmarks.

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 Project->Options and under 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 Misc Ctonrols.

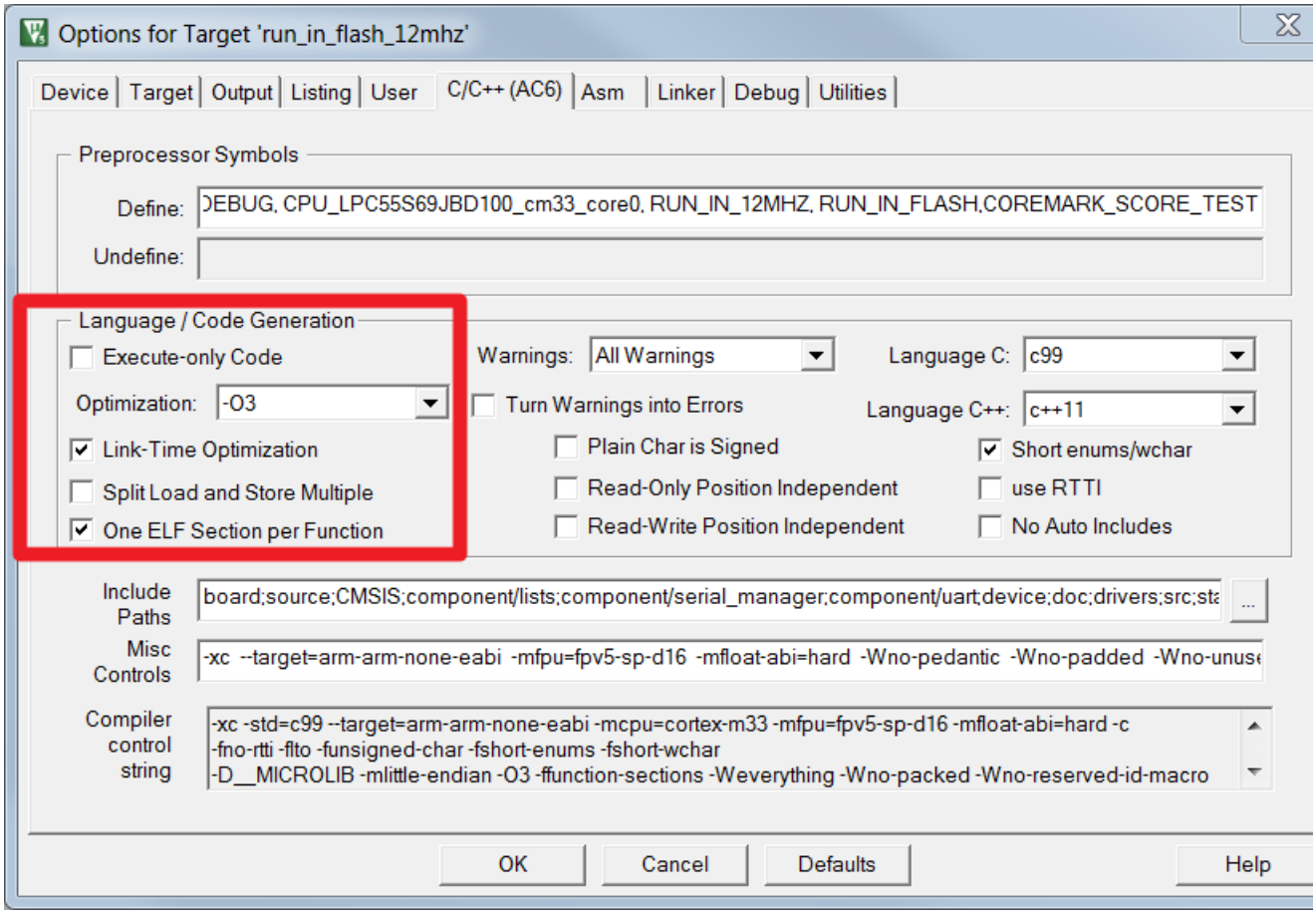


Figure 21. Keil MDK CoreMark score optimization

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.

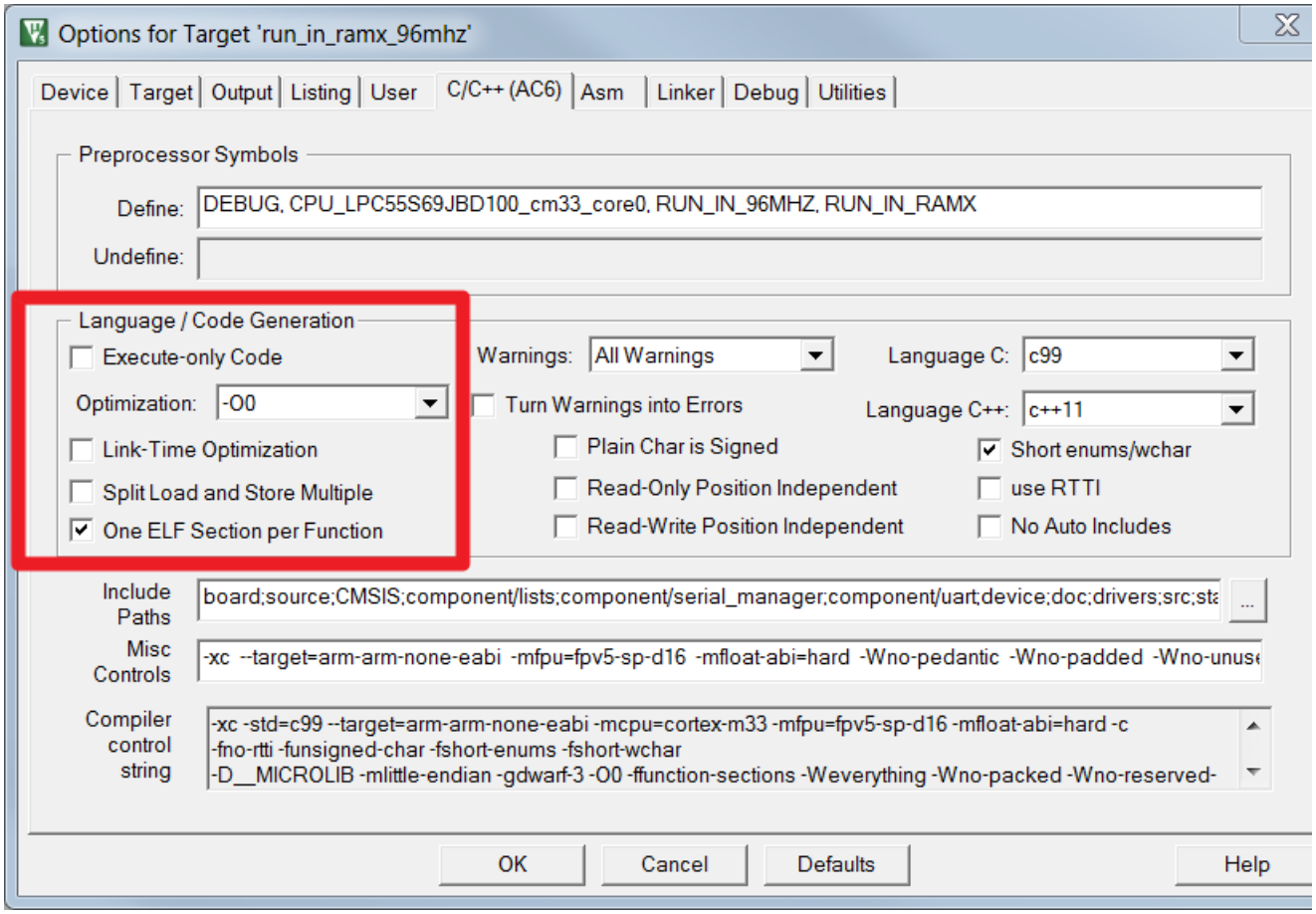


Figure 22. Keil MDK µA/MHz optimization

2.2.2.2 IAR Optimization

There are two compiler optimizations that 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” checkbox

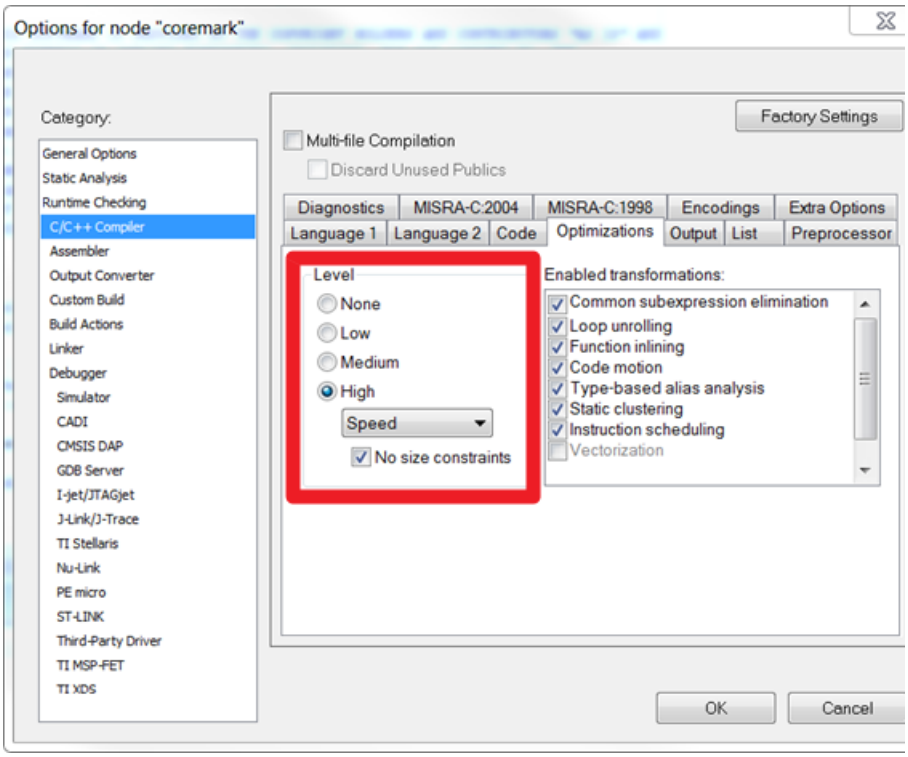


Figure 23. IAR EWARM CoreMark score optimization

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

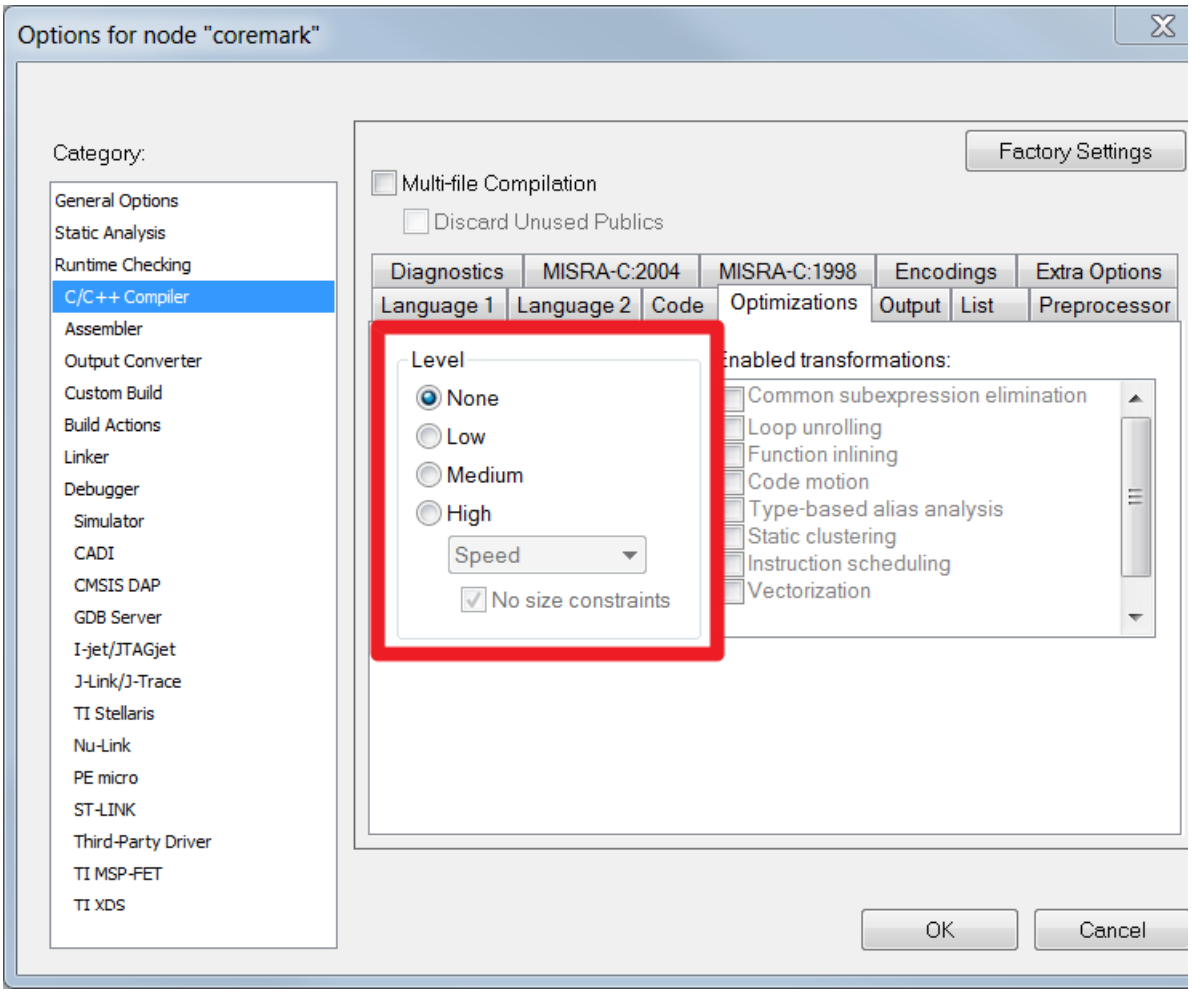


Figure 24. IAR EWARM µA/MHz optimization

2.2.2.3 MCUXpresso Optimization

There are two compiler optimizations that can be done to improve CoreMark score. Set the optimization level to “-O3” so please select “Optimize most(-O3)” from the drop down menu.

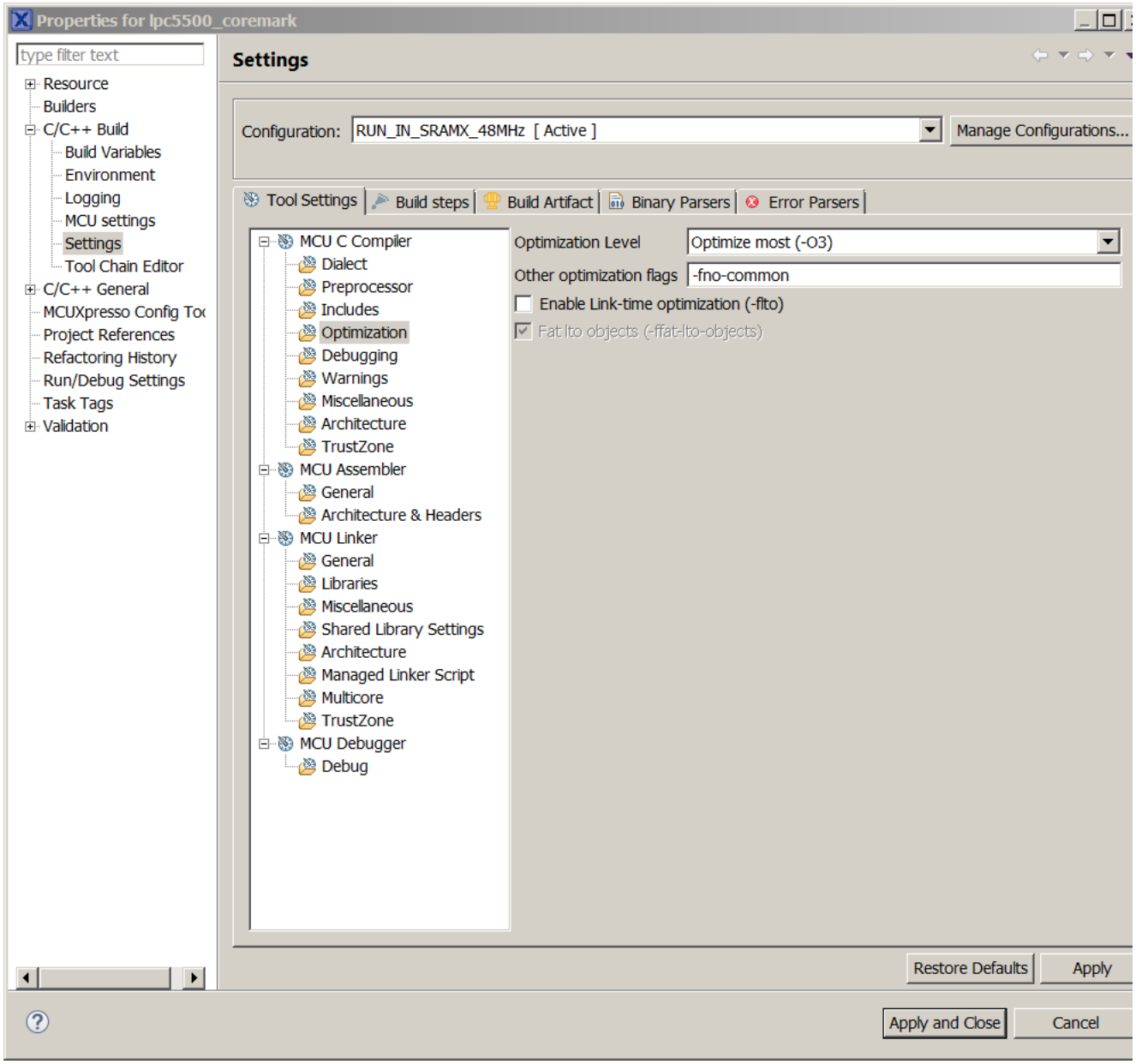


Figure 25. MCUXpresso CoreMark score optimization

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

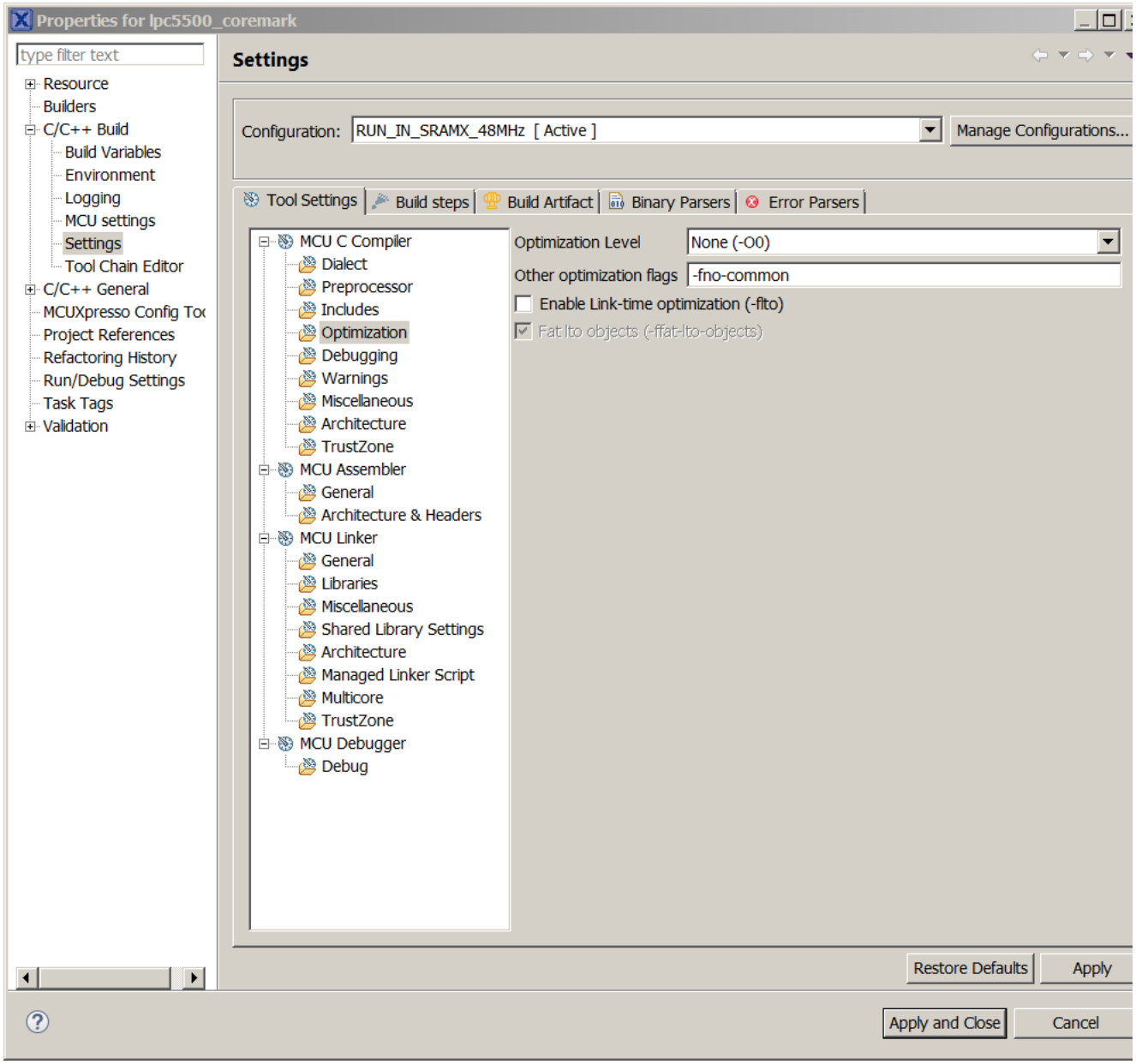


Figure 26. MCUXpresso µA/MHz optimization

3 Measuring CoreMark on board

3.1 LPC55S69Xpresso board

The LPC55S69Xpresso board supports a VCOM serial port connection via **P6**. 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, the new line receive setting should be set to auto.

3.2 Board Setup

The LPC55S69 Rev A1 development board is used for benchmarking

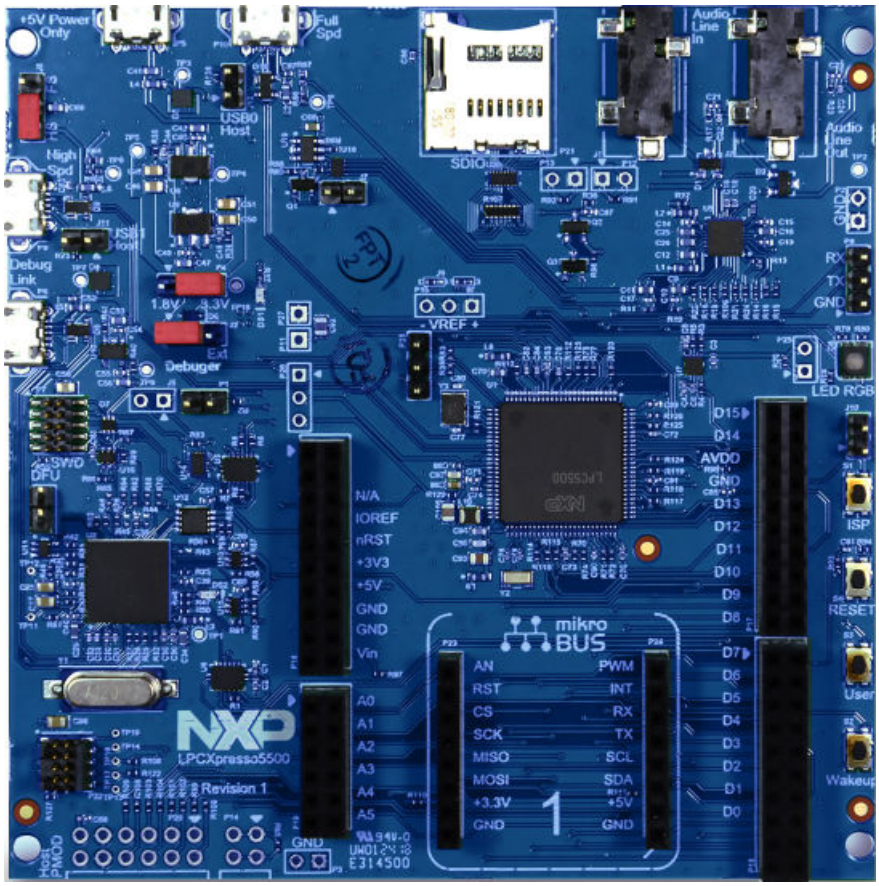


Figure 27. LPC55S69Xpresso board

The board ships with CMSIS-DAP debug firmware programmed. Visit the following FAQ for more information on CMSIS_DAP debug firmware: https://www.nxp.com/downloads/en/software/lpc_driver_setup.exe For debugging and terminal debug messages, connect a USB cable to P6 USB connector. Board schematics are available on www.nxp.com.

3.2.1 µA/MHz measurement setup

To measure the LPC5500 power consumption, remove R92, install header at P13, and connect ammeter across P13 as shown in Figure 28.

NOTE

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

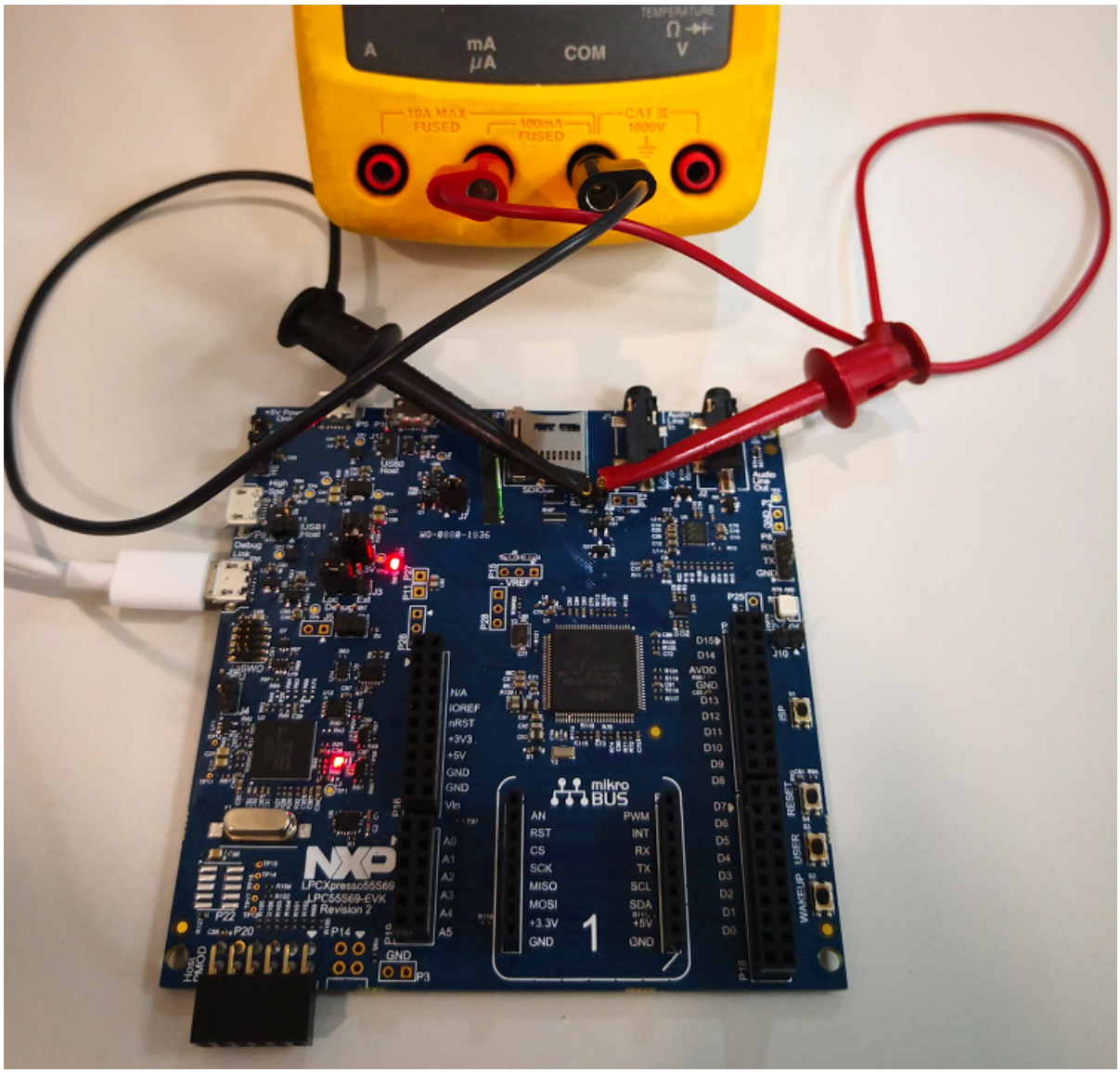


Figure 28. μA/MHz measurement setup

If we need measurement the MCU core current, we need rework the board by removing the R92. Then we can measure the current through P13 by multimeter.

While performing the μA/MHz benchmark, use P6 USB connector to provide power to the board. After the μA/MHz benchmark project has been downloaded, power cycling the board by removing the USB cable, and reinsert to make sure that the debug probe is not connected.

The baud rate setting for debug messages is 115200. It can be changed in core_potme.c file.

Line209 config.baudRate_Bps = 115200;

Similarly, by selecting different configuration projects in workspace window, the core clock frequency can be changed. Each of the configuration may enable below defined project configuration settings:

RUN_IN_12MHZ

RUN_IN_48MHZ

RUN_IN_96MHZ

RUN_IN_150MHZ

3.3 Run CoreMark code

The first step to get CoreMark result is to connect the connector P6 of the board with PC. Then the PC recognizes the LPC-Link2 debugger with a Simulate Serial Port as shown in Fig 29 .

If PC cannot find the serial port driver, download the LPCScrypt from below link, and install 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

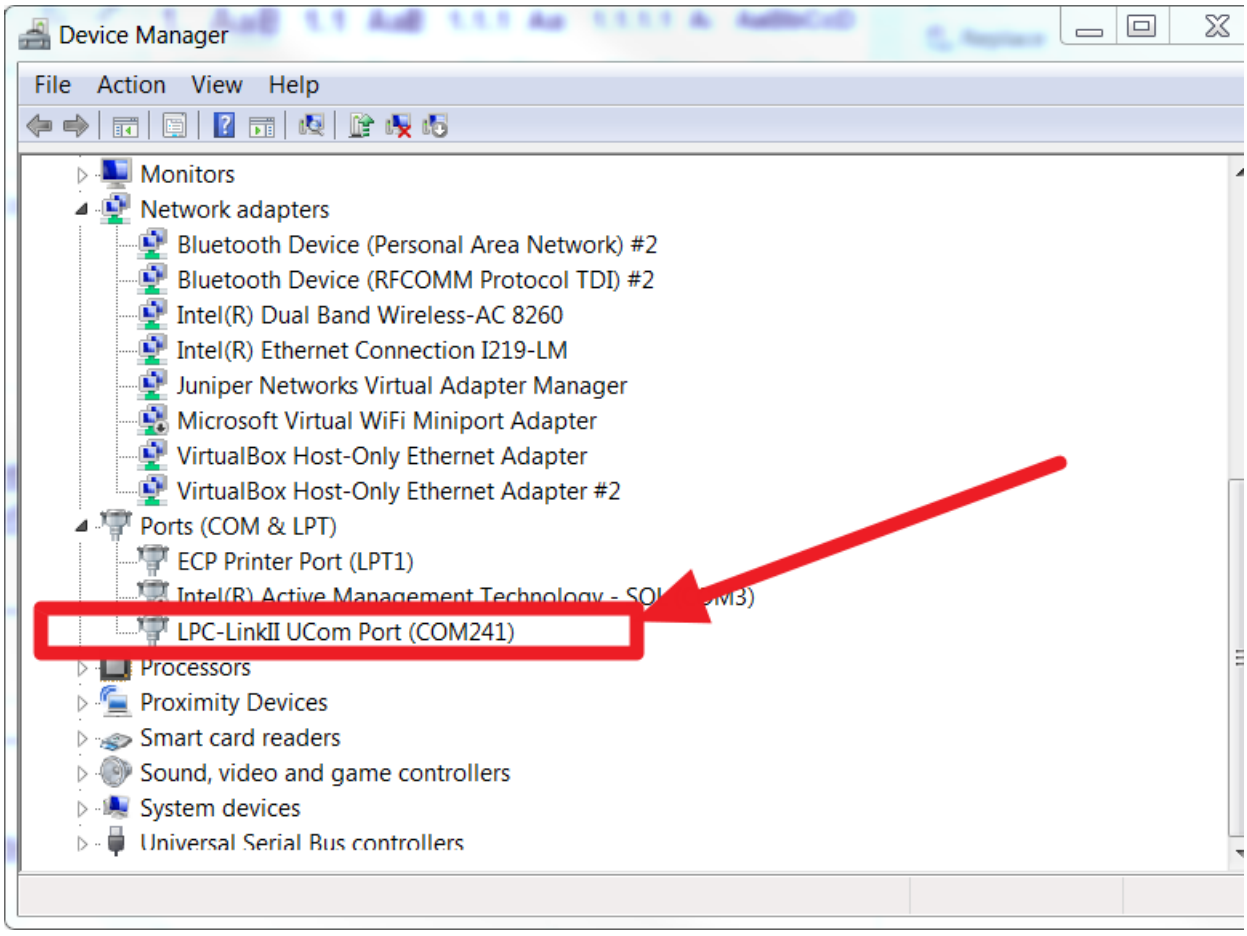


Figure 29. LPC-LinkII UCom Port

Open a UART debug terminal (like Tera Term, putty, etc.), and configure as 115200, 8 data bits, no parity, 1 stop bit, refer Fig 30.

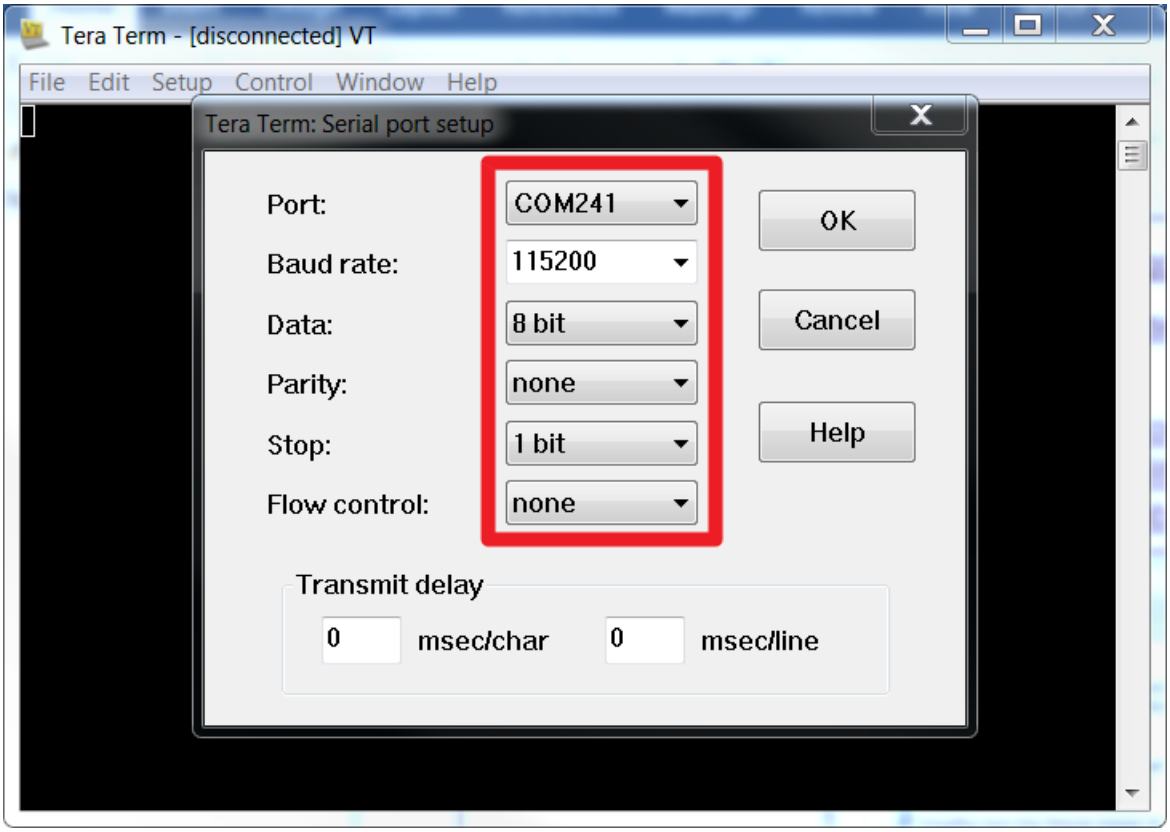


Figure 30. UART debug terminal configuration

Once the CoreMark necessary files are added into the project (by following Chapter 2.1 instructions), compile the project and download to the LPC5500Xpresso board.

Click reset button, the CoreMark benchmark prints on the terminal after a few seconds, like Fig 31 in Chapter 4.

4 Result

Figure 31 shows the CoreMark benchmark result when running LPC5500 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 372.786580/96 MHz = 3.883 CoreMark/MHz .

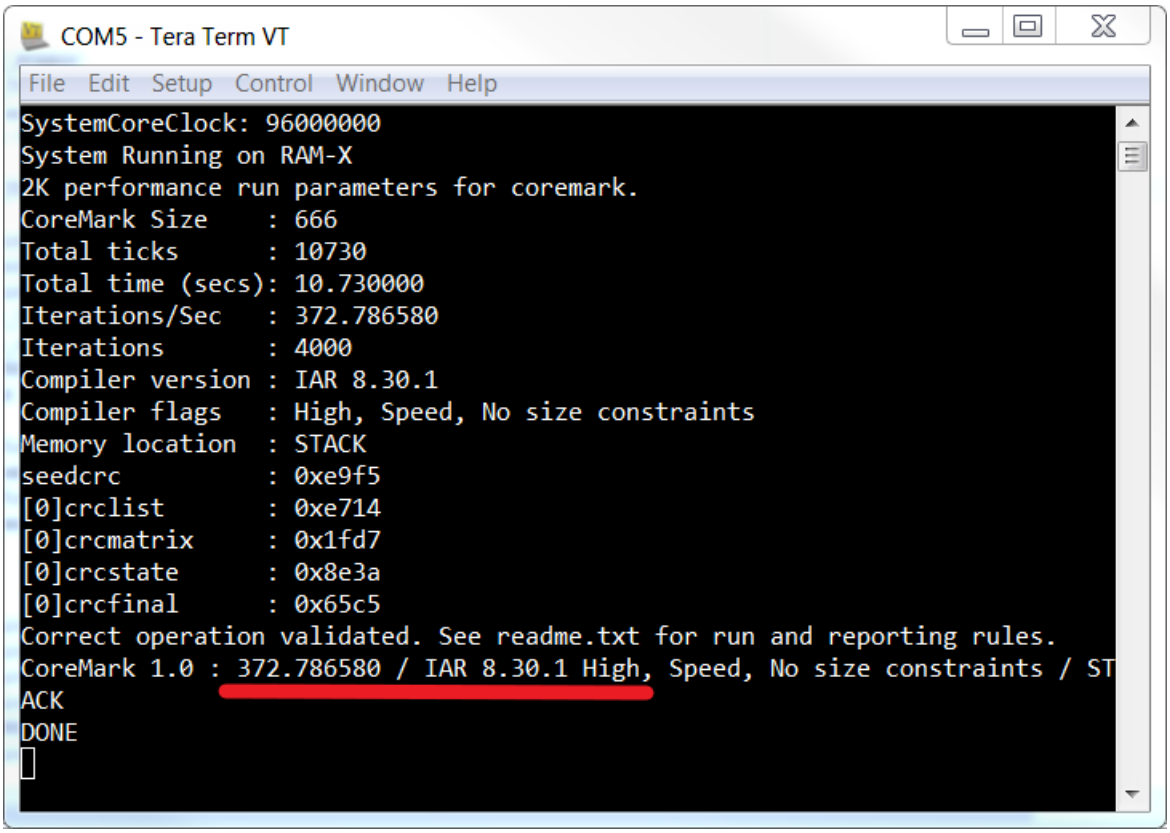


Figure 31. CoreMark result

Table 1 shows 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 1. LPC55S69Xpresso board CoreMark/MHz Score

IDE	CoreMark/MHz Score(SRAMX)	CoreMark/MHz Score(Flash)
KEIL MDK	4.021	2.333
IAR EWARM	3.887	2.435
MCUXpresso	2.843	2.016

NOTE

Test under 96 MHz

For µA/MHz, following tables show typical results when running on the LPCXpresso55S69 board with VDD = 3.3 V at room temperature. Fig 24 compares the three IDEs in terms of power consumption.

NOTE

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

NOTE

The average current in 150MHz will higher than other modes, the reason is 150Mhz will enable PLL, the PLL cost more power.

Table 2. 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	1.34	111.67	1.35	112.50
48 MHz	2.68	55.84	2.72	56.67
96 MHz	3.89	40.53	3.95	41.14
150 MHz	7.24	48.27	6.30	42.00

Table 3. 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	1.48	123.34	1.29	107.50
48 MHz	2.63	54.80	3.32	69.17
96 MHz	3.96	41.25	4.28	44.59
150 MHz	7.63	50.87	7.56	50.40

Table 4. 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	1.33	110.84	1.19	115.84
48 MHz	2.40	50.00	2.41	50.03
96 MHz	3.64	37.92	3.58	37.30
150 MHz	7.19	47.94	6.57	43.80

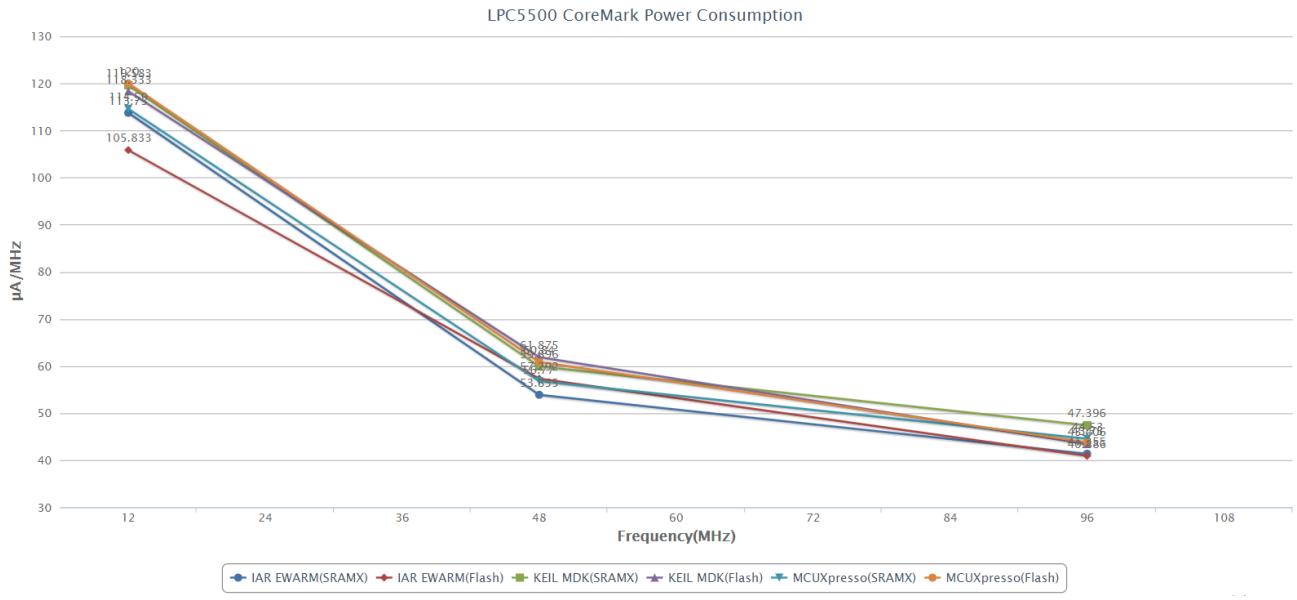


Figure 32. µA/MHz IDE comparison

5 Conclusion

Three types of CoreMark benchmarking on the LPC55xx are presented in this document with different IDEs (Keil, IAR, MCUXpresso):

CoreMark score, power consumption, and 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 LPCXpresso55S69. 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 37.30, achieved by running CoreMark from flash when core frequency is 96 MHz.

6 Reference

1. CoreMark Benchmarking for ARM Cortex Processors, ARM
2. AN11811 LPC5411x CoreMark Cortex-M4 Porting Guide,NXP
3. UM11126_LPC55xx/LPC55Sxx User Manual ,NXP

7 Revision history

Revision history		
Rev.	**Date**	**Substantial changes**
0	25 January, 2019	Initial reversion
1	December, 2019	Updated CoreMark scores on silicon ‘1B’ with SDK2.6.3 and add MHz CoreMark

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: December 2019

Document identifier: AN12284

arm