AN11607

LPC5410x CoreMark Cortex-M4 Porting Guide

Rev. 2.0 — 14 July 2015 Application note

Document information

Info	Content
Keywords	LPC54102, LPC54101, CoreMark, Cortex-M4 Porting Guide, Keil MDK, IAR EWARM, LPCXpresso
Abstract	This application note describes how to adapt the provided framework projects to run CoreMark benchmark tests on the Cortex-M4 core of the LPC5410x. Different project configuration options for best CoreMark number and µA/MHz are discussed.

Revision history

Rev	Date	Description
2	20150714	Added support for IAR EWARM and LPCXpresso.
1	20141104	Initial version.

Contact information

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

1. Introduction

The LPC54100 series of microcontrollers introduces a breakthrough in ultra-low power performance for "always-on" sensor processing. In an “always on” application, the LPC54100 is in a power down mode “listening” for sensor data. In this low power state, the LPC54100 only draws 3 uA of current. Developers can optimize power efficiency and throughput by choosing between a power-efficient 55 µA/MHz Cortex-M0+ core for sensor data collection, aggregation, and external communications, and a Cortex-M4 processor at 100 µA/MHz to complete math-intensive algorithms, such as sensor fusion, more quickly while saving power.

The LPC54100 was designed from the ground up for ultimate low power efficiency. The low power flash is writable at 1.62V. Core and peripheral voltages are automatically scaled for reduced power consumption at any frequency. An asynchronous peripheral bus reduces peripheral clock speed without affecting CPU clock to minimize peripheral power contribution. Low power serial interfaces can wake up the CPU from Power-Down mode upon receiving data from synchronous serial interfaces or wakeup pins. The 12 bit, 12 channel ADC at 4.8Msps can operate at full spec including 1.62V and can perform conversion while the CPU is asleep.

CoreMark is a simple, yet sophisticated benchmark that is designed specifically to test the functionality of a processor core. Running CoreMark produces a single-number score allowing users to make quick comparisons between processors.

ARM released “DAI 0350A CoreMark Benchmarking for ARM Cortex processors” http://infocenter.arm.com/help/topic/com.arm.doc.dai0350a/DAI0350A_coremark_benchmarking.pdf. This is also a useful reference to review and get familiar with the benchmark.

This application note shows the process of setting up the LPC5410x device and code including memory partitioning, build options and compiler flags. It also shows how to measure CoreMark scores with the Cortex-M4 and presents the Cortex-M4 µA/MHz results that can be achieved. Separate CoreMark frameworks executing CoreMark code from flash and SRAM is provided for Keil MDK, IAR EWARM, and LPCXpresso.

2. Downloading and integrating CoreMark to the framework

The zip files associated with this application note contains LPCOpen based project frameworks that allows developers to drop in the CoreMark sources and quickly get up and running with benchmarking the LPC5410x.

To get started first go to https://www.eembc.org/coremark/. Click the download link and follow the instructions on that page as shown in Figure 1.

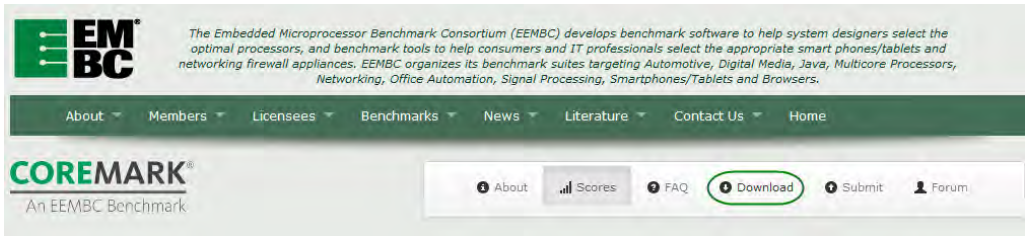

Fig 1. EEMBC download link

After reviewing the license terms, look through the readme and documentation file downloads. The readme gives some 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 Porting CoreMark into the CoreMark framework

There are two variants of the CoreMark framework provided with this application note; one for each of the three toolchains that NXP supports for a total of six CoreMark frameworks. One variant executes the entire application out of the internal flash memory while the other modifies the linker script in order to execute the CoreMark code out of internal SRAM. For clarity, the absolute path for each of the six projects are given below.

The flash-based CoreMark framework on Keil MDK is located here:

lpc5410x_keil_iar_cm_flash\applications\lpc5410x\keil\lpcxpresso_54102\coremark_framework.uvmpw

The flash-based CoreMark framework on IAR EWARM is located here:

lpc5410x_keil_iar_cm_flash\applications\lpc5410x\iar\lpcxpresso_54102\coremark_framework.eww

The flash-based CoreMark framework on LPCXpresso is located inside this archive:

lpc5410x_lpcxpresso_cm_flash.zip

The SRAM-based CoreMark framework on Keil MDK is located here:

lpc5410x_keil_iar_cm_sram\applications\lpc5410x\keil\lpcxpresso_54102\coremark_framework.uvmpw

The SRAM-based CoreMark framework on IAR EWARM is located here:

lpc5410x_keil_iar_cm_sram\applications\lpc5410x\iar\lpcxpresso_54102\coremark_framework.eww

The SRAM-based CoreMark framework on LPCXpresso is located inside this archive:

lpc5410x_lpcxpresso_cm_sram.zip

Extract and open the project file from the IDE you wish to work with.

For Keil MDK, it is recommended to extract the archive contents as close to the hard drive's root folder, such as C:\nxp or a similar folder (long paths names can otherwise be a problem for Keil MDK).

Depending on the toolchain chosen, the workspace should look like one of the following three figures. The CoreMark framework requires the addition of the CoreMark files from EEMBC. For Keil MDK and IAR EWARM, the coremark_m4 project must be set as active before the CoreMark files can be added.

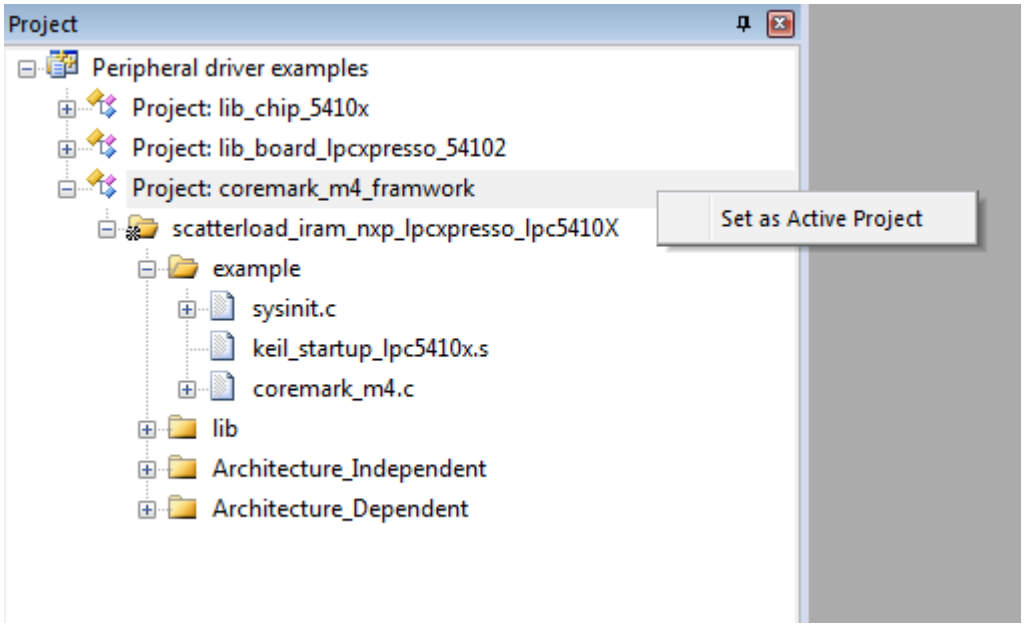

Fig 2. Keil MDK workspace

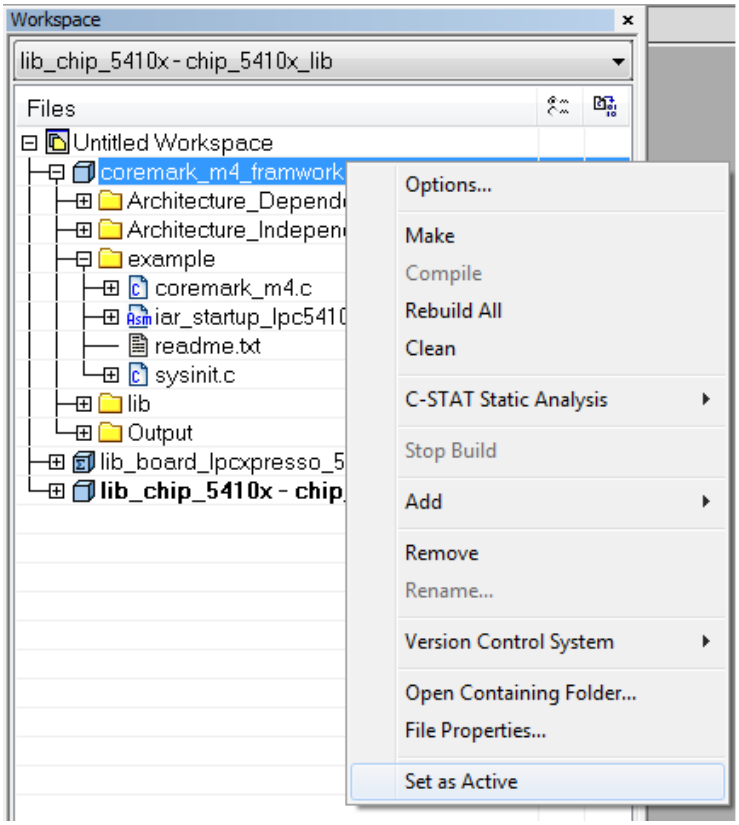

Fig 3. IAR EWARM workspace

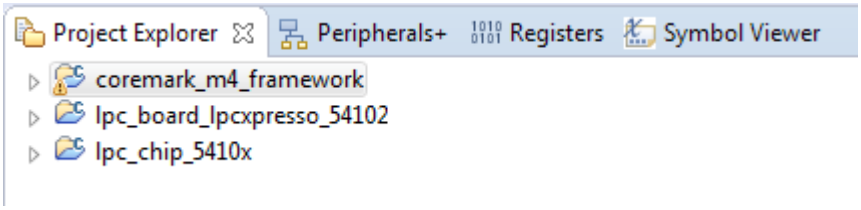

Fig 4. LPCXpresso workspace

Copy the files as shown in Figure 5 from the coremark.zip download.

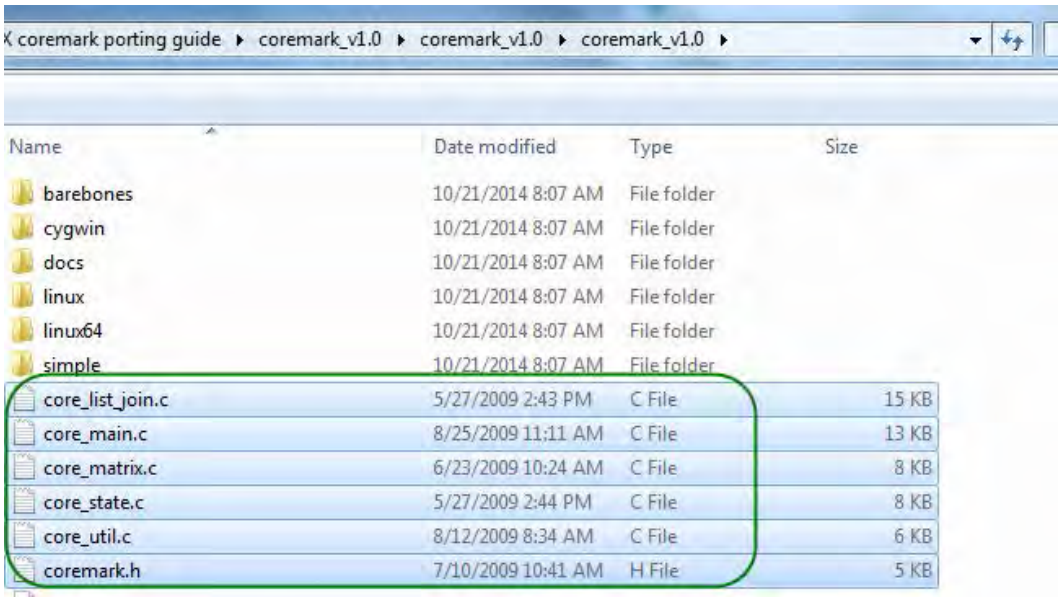

Fig 5. CoreMark files to copy

Place the files in the following project directory for Keil MDK and IAR EWARM:

\applications\lpc5410x\examples\coremark

For LPCXpresso:

\coremark_m4_framework\example\Architecture_Independent

The file ee_printf.c in the \barebones directory of the EEMBC archive also needs to be copied to the same folder for Keil and IAR. For LPCXpresso, it should be copied to the following directory:

\coremark_m4_framework\example\Architecture_Dependent

Add these files into the project. In Keil MDK, double click on Architecture Independent folder and browse to directory of the CoreMark files:

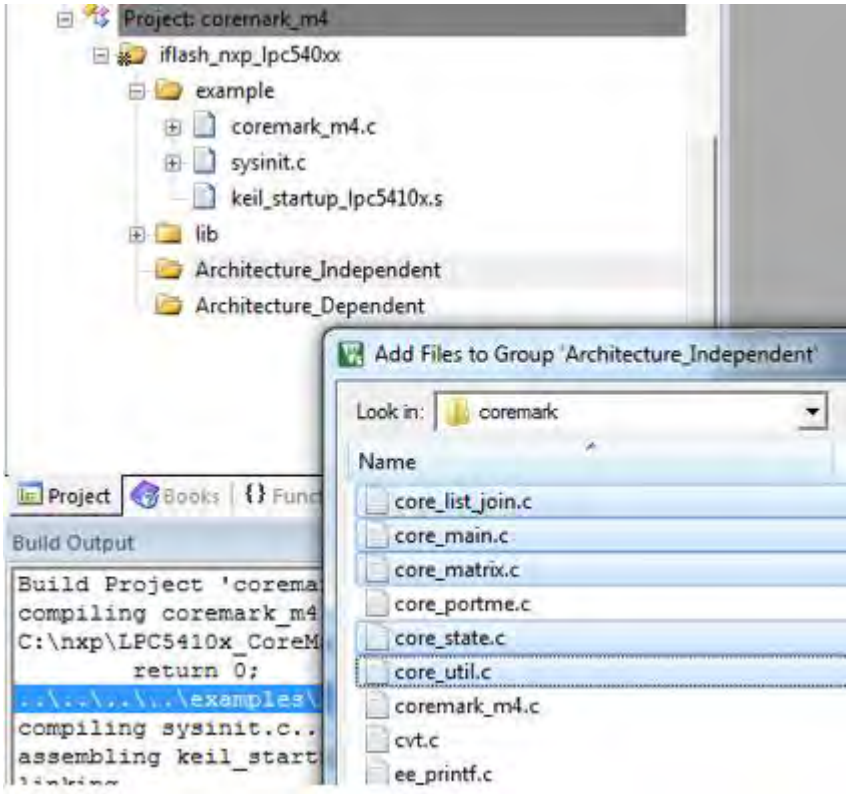

Fig 6. Adding files in Keil MDK

For IAR, right click the Architecture Independent folder and select Add and then “Add Files...”:

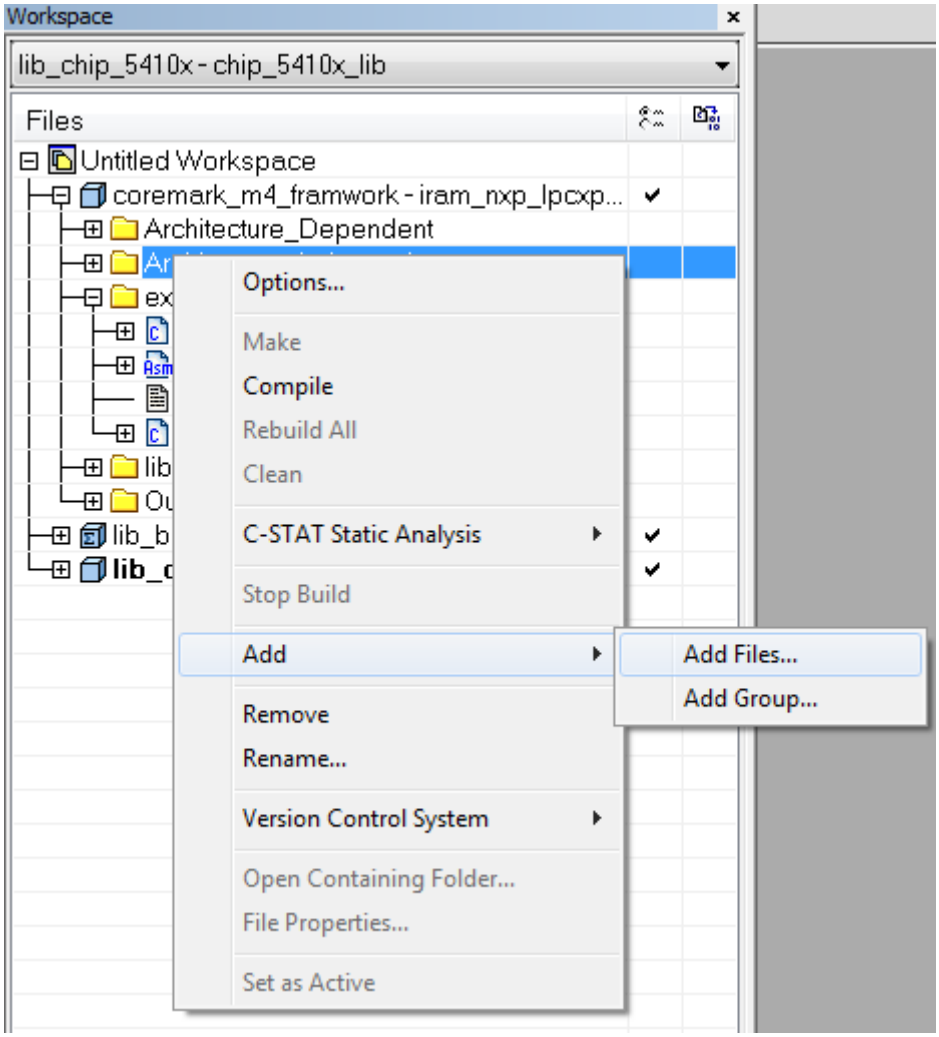

Fig 7. Adding files in IAR EWARM

For LPCXpresso, right click inside the “Project Explorer” tab and select Refresh. LPCXpresso should detect the newly added files and add them to the Project Explorer automatically:

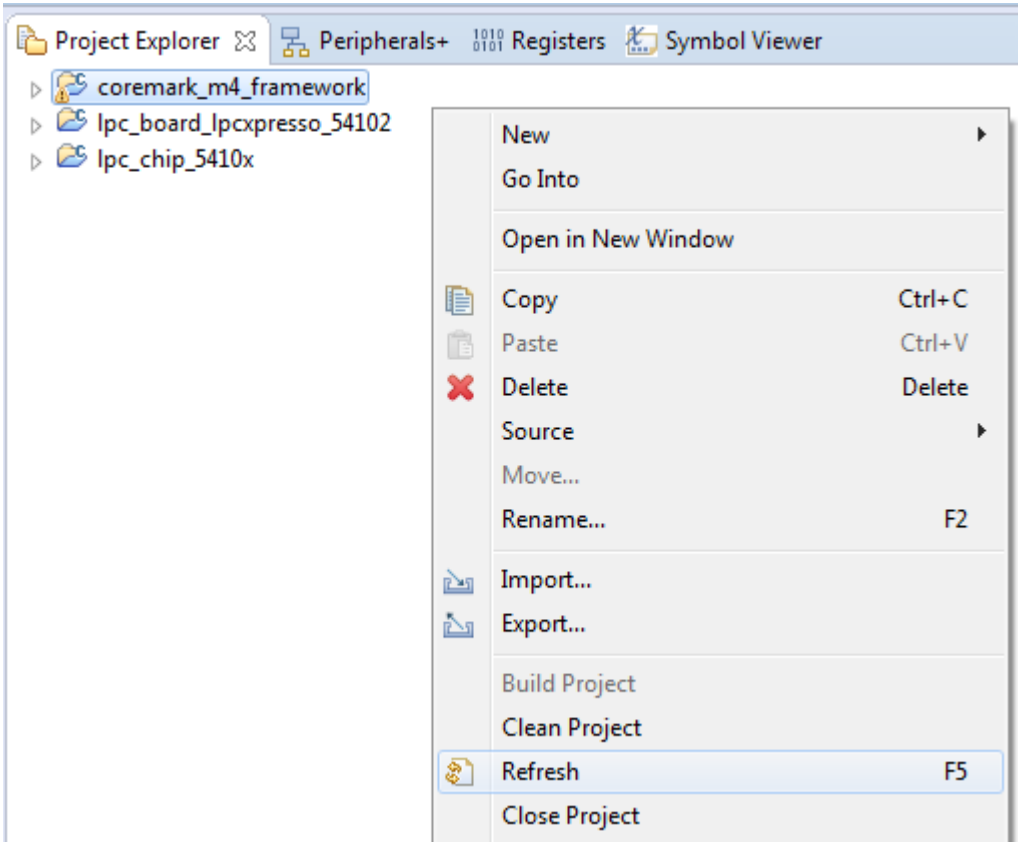

Fig 8. Refreshing LPCXpresso workspace

Add the file eeprinf.c into the ‘Architecture Dependent’ folder. Architecture dependent files are shown below. The cvt.c file requires no change. Use the core_portme.c file supplied in the project not the one from the coremark.zip. For convenience this file has the required porting changes ready for use.

The workspace should look like the following in all three IDEs:

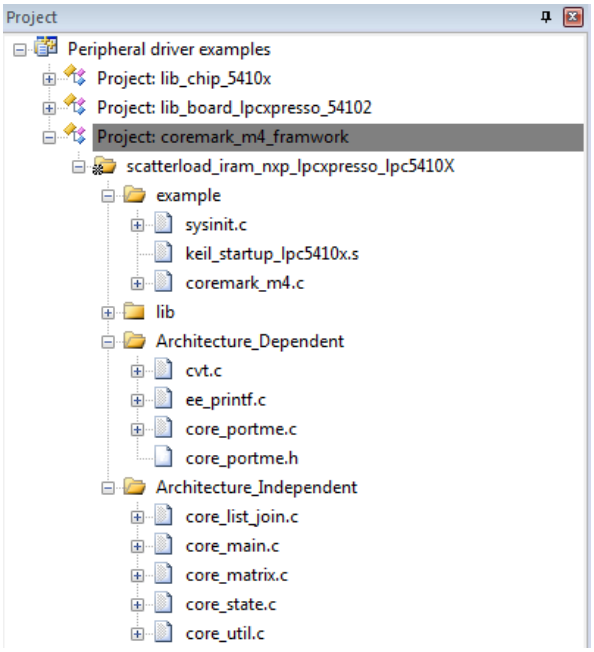

Fig 9. Keil MDK workspace after adding CoreMark files

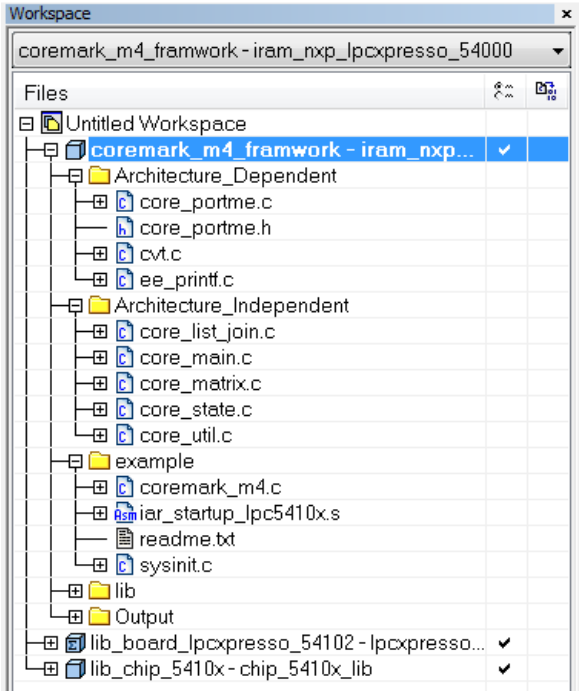

Fig 10. IAR EWARM workspace after adding CoreMark files

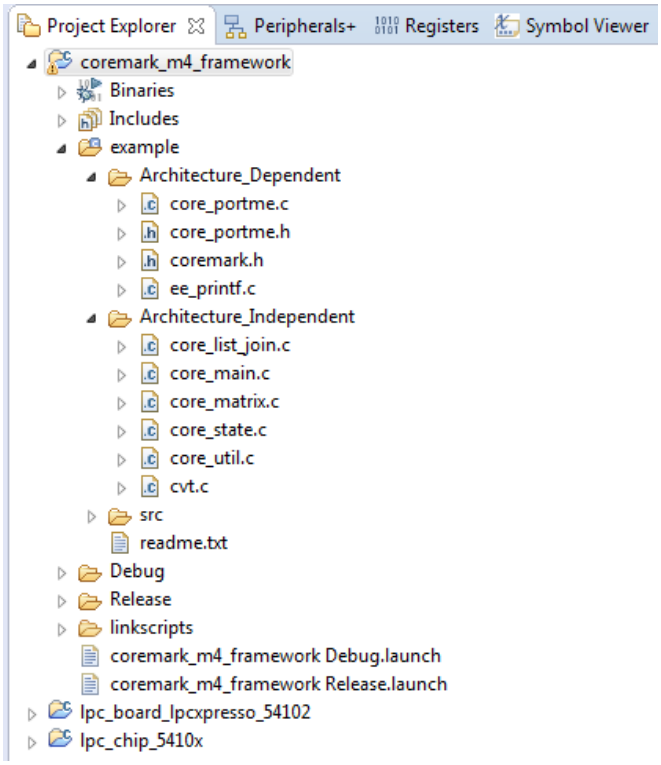

Fig 11. LPCXpresso workspace after adding CoreMark files

A few edits will need to be made to integrate the CoreMark files into the framework. In the file 'core_main.c' make the following modifications:

Replace:

```
#if MAIN_HAS_NOARGC
MAIN_RETURN_TYPE main(void) {
      int argc=0;
      char *argv[1];
#else
```

Fig 12. core_main.c before edits

With:

```
#if MAIN_HAS_NOARGC
      #if defined RENAME_MAIN
MAIN_RETURN_TYPE core_main(void) {
      #else
MAIN_RETURN_TYPE main(void) {
      #endif
      int argc=0;
      char *argv[1];
#else
```

Fig 13. core_main.c after edits

The file ‘eeprinf.c’ needs the LPCOpen UART call to be added to support ‘printf’ to the terminal. To do this, add the following line of code in uart_send_char(char c), see Figure 14.

```
void uart_send_char(char c) {
    Board_UARTPutChar(c);
}
```

Fig 14. Adding printf support to CoreMark

Also, in eeprintf(), add an #if defined(COREMARK_SCORE_TEST). See Figure 15.

```
int ee_printf(const char *fmt, ...)
{
  char buf[256],*p;
  va_list args;
  int n=0;
  #if defined(COREMARK_SCORE_TEST)
  va_start(args, fmt);
  ee_vsprintf(buf, fmt, args);
  va_end(args);
  p=buf;
  while (*p) {
   uart_send_char(*p);
   n++;
   p++;
   }
   #endif
  return n;
}
```

Fig 15. Disabling printf support when executing µA/MHz test

This is added so that printf code will be optimized out 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. Commenting out this #define to run µA/MHz test will also optimize out these unnecessary print statements.

Double check that the include path is correct for the CoreMark files.

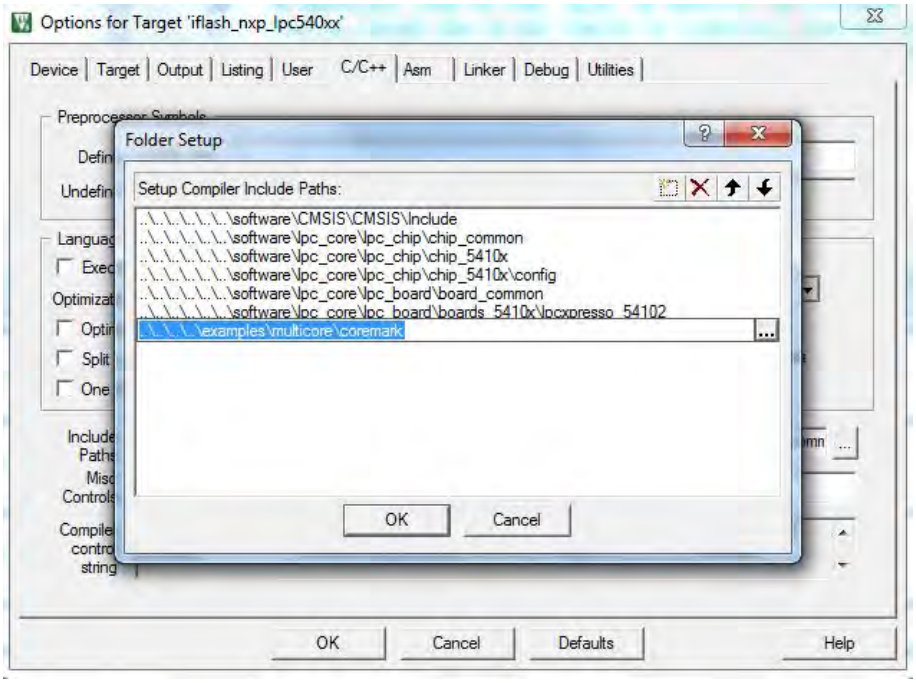

Fig 16. Keil MDK compiler include paths

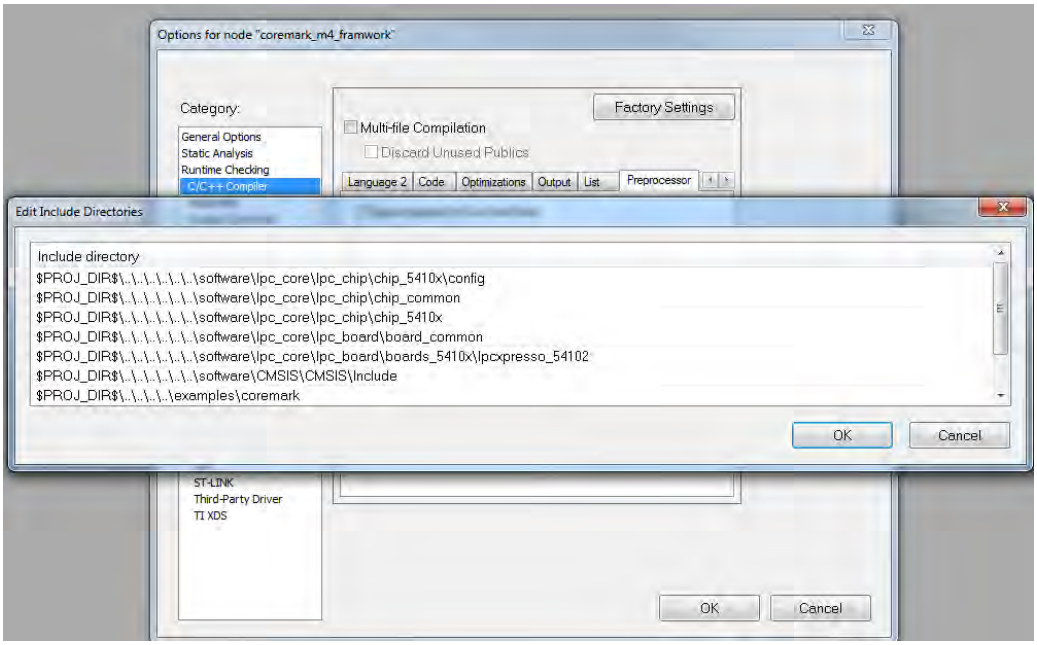

Fig 17. IAR EWARM compiler include paths

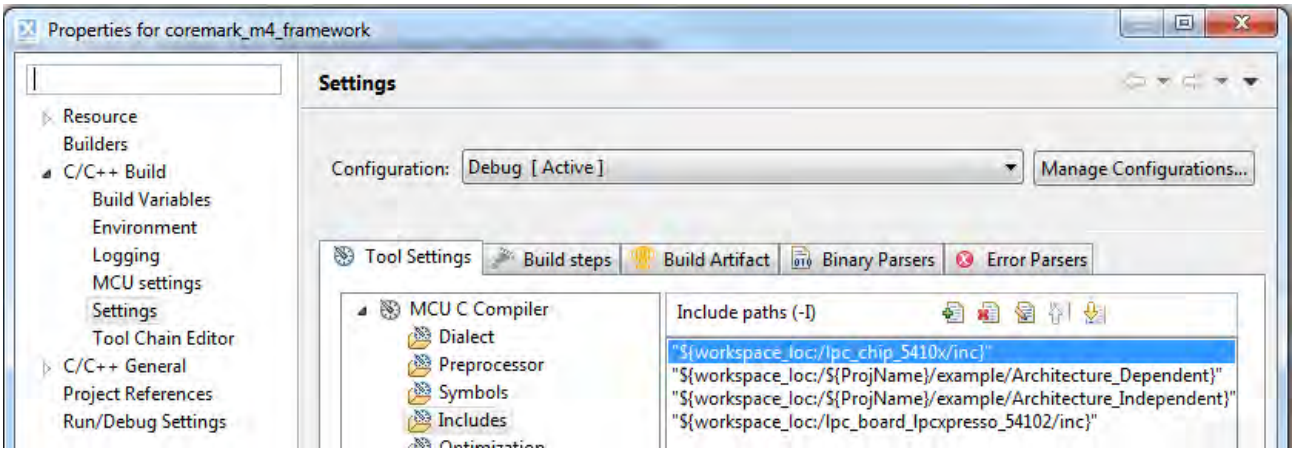

Fig 18. LPCXpresso compiler include paths

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

If the IDE chosen is IAR EWARM and your goal is to store the CoreMark code in SRAM for faster execution, a line of code will need to be added to some CoreMark files. In any other scenario, this line of code is not needed. The files that need to be modified are:

core_main.c, core_util.c, core_state.c, core_matrix.c and core_list_join.c

To execute CoreMark from SRAM with IAR EWARM, these CoreMark files will need to be labeled as their own IAR EWARM linker “section”. The provided .icf linker file will then place this section, which is called “critical_text”, into SRAM. To do this, add the following line of code shown in Figure 19 above includes in all five of the files.

```
/* CoreMark source files should contain this #pragma command to be put into SRAM */
#pragma default_function_attributes = @ "critical_text"
```

Fig 19. IAR EWARM #pragma command

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 5.13.0.0

IAR EWARM 7.40.1.8472

LPCXpresso 7.6.2

2.2.1 Memory considerations

Due to the inherent architecture of SRAM and flash, CoreMark will execute faster when running out of SRAM. As shown in Figure 20, the LPC5410x internal memory uses a multilayer AHB matrix system that provides a separate instruction and data bus for the Cortex-M4 as well as individual buses for each of the three SRAM banks. Placing the CoreMark code and data sections into different SRAM banks minimizes bus contention and improves instruction and data parallelism.

It is important to minimize the flash wait states according to the MCUs 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 on correctly configuring the flash memory, such as the minimum amount of wait states allowed at a given core frequency, read the LPC5410x user manual:

http://www.nxp.com/documents/user_manual/UM10850.pdf

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

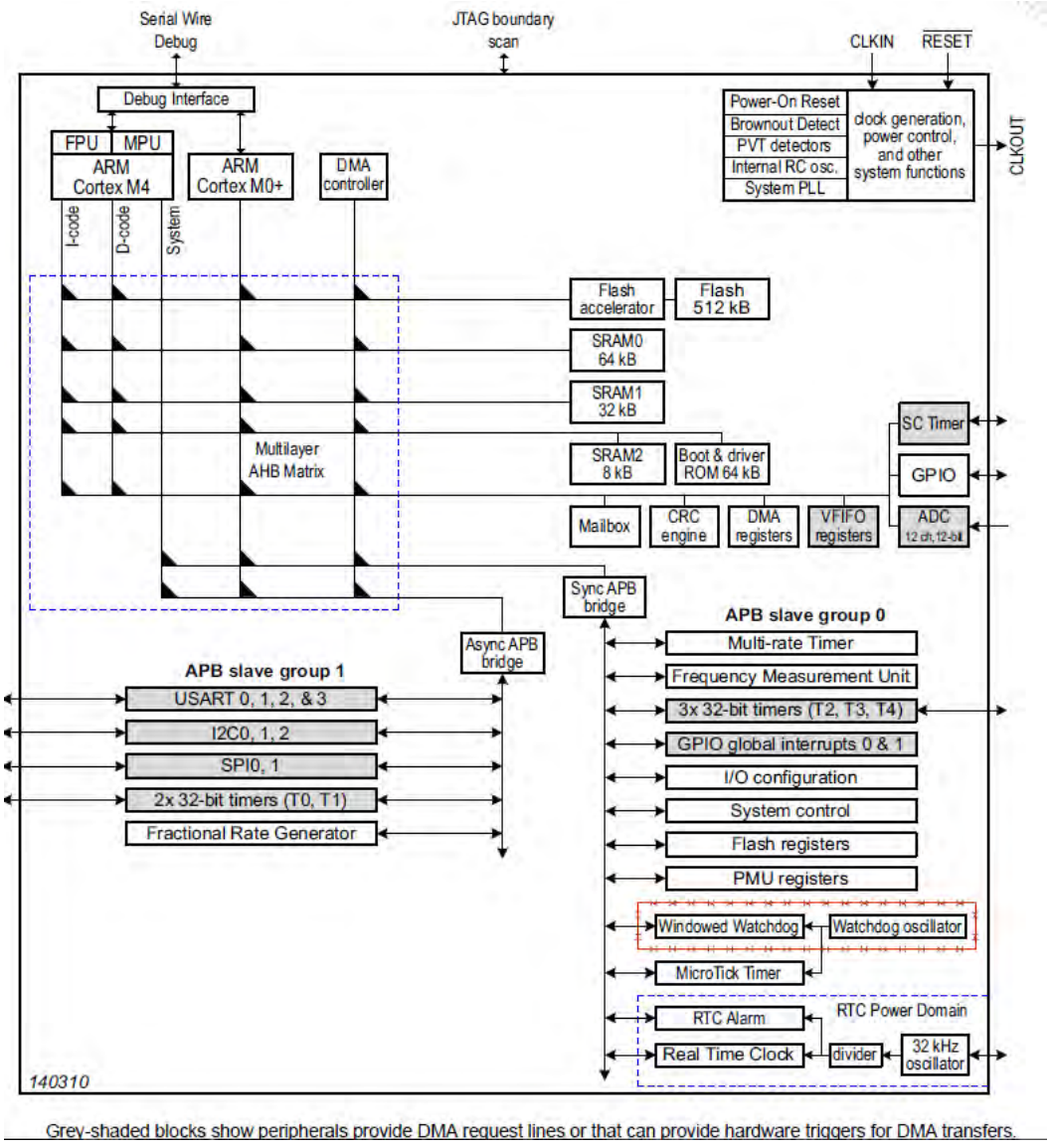

Fig 20. LPC5410x AHB matrix

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.

2.2.2 IDE Optimizations

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 MDK Optimizations

There are two compiler optimizations that can be done to improve the CoreMark score. As shown in Figure 21, in the project options and under the C/C++ tab, the optimization level needs to be set to Level 3 (-O3) and “Optimize for Time” should be ticked.

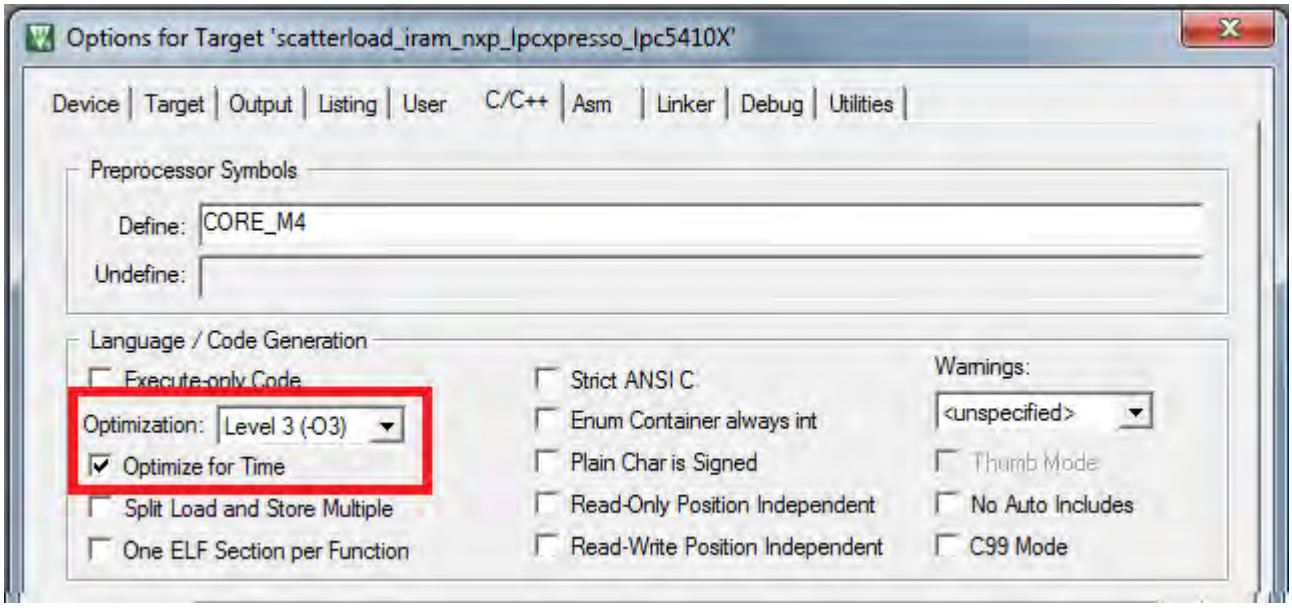

Fig 21. Keil MDK CoreMark score optimizations

When benchmarking the power consumption of the MCU, the optimization level should be set to Level 0 (-O0) and “Optimized for Time” should be unchecked. See Figure 22.

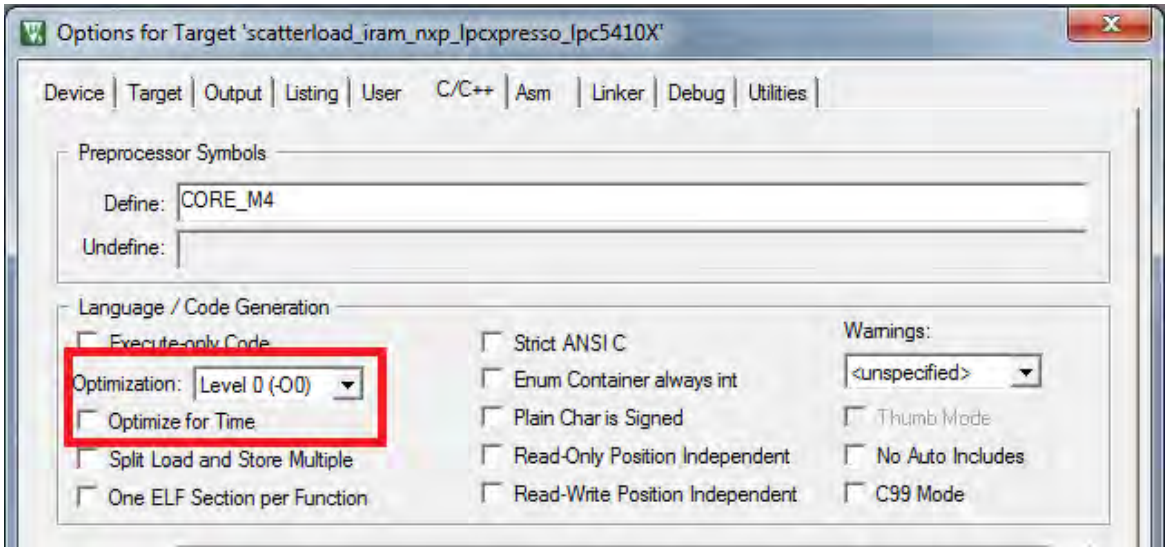

Fig 22. Keil MDK µA/MHz optimizations

2.2.2.2 IAR EWARM Optimizations

There are two compiler optimizations that can be done to improve CoreMark score. Set the optimization level to “High” and select “Speed” from the drop down menu. See Figure 23.

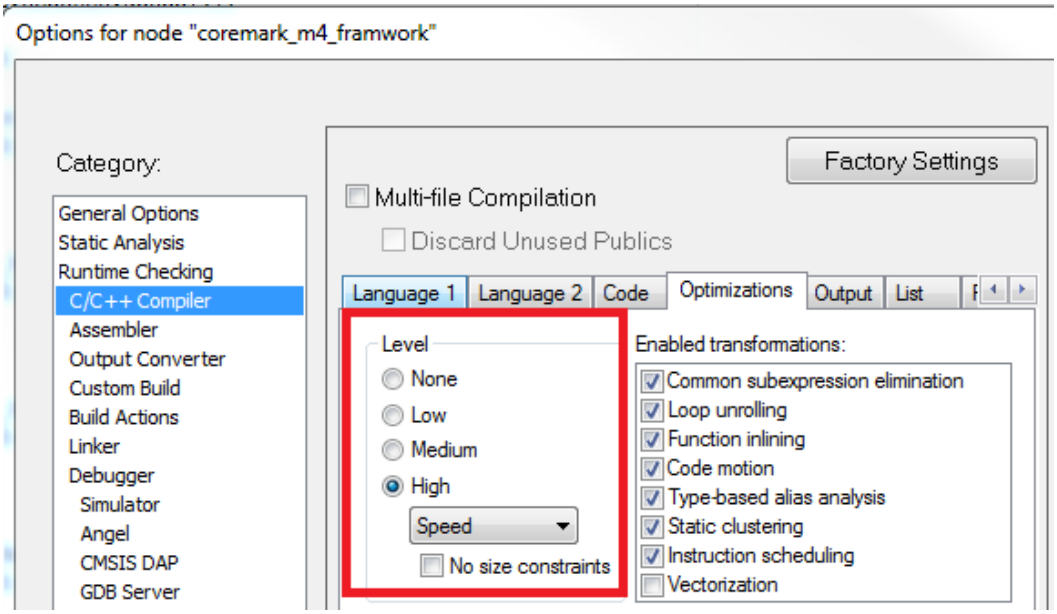

Fig 23. IAR EWARM CoreMark score optimizations

When benchmarking the power consumption of the MCU, the optimization level should be set to “None”. See Figure 24.

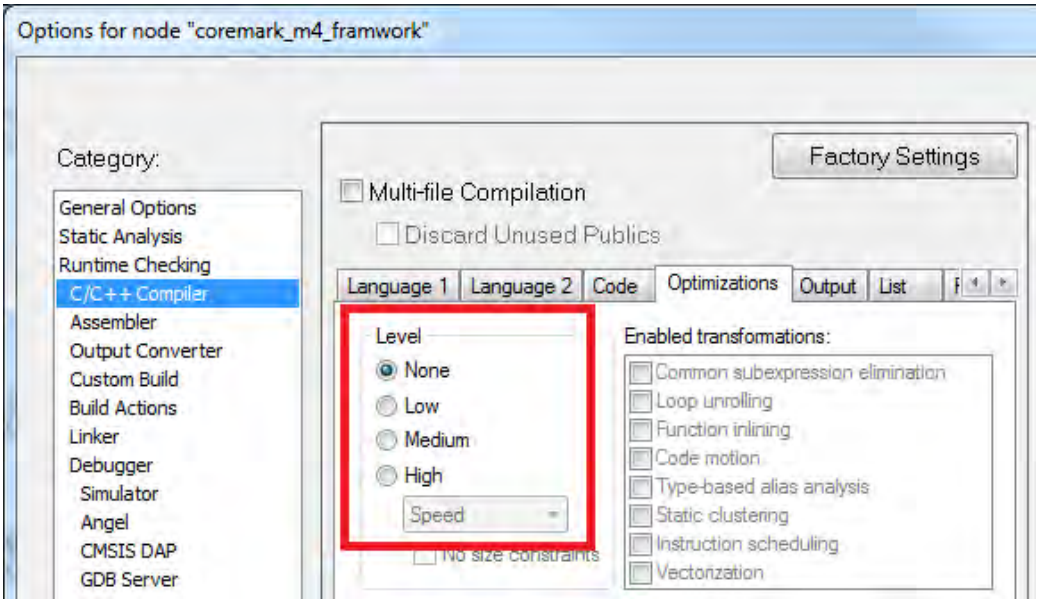

Fig 24. IAR EWARM µA/MHz optimizations

2.2.2.3 LPCXpresso Optimizations

There are two compiler optimizations that can be done to improve CoreMark score. Set the optimization level to “Optimize most (-O3)” with the optimization flag “-ftree-switch-shortcut”. See Fig 25

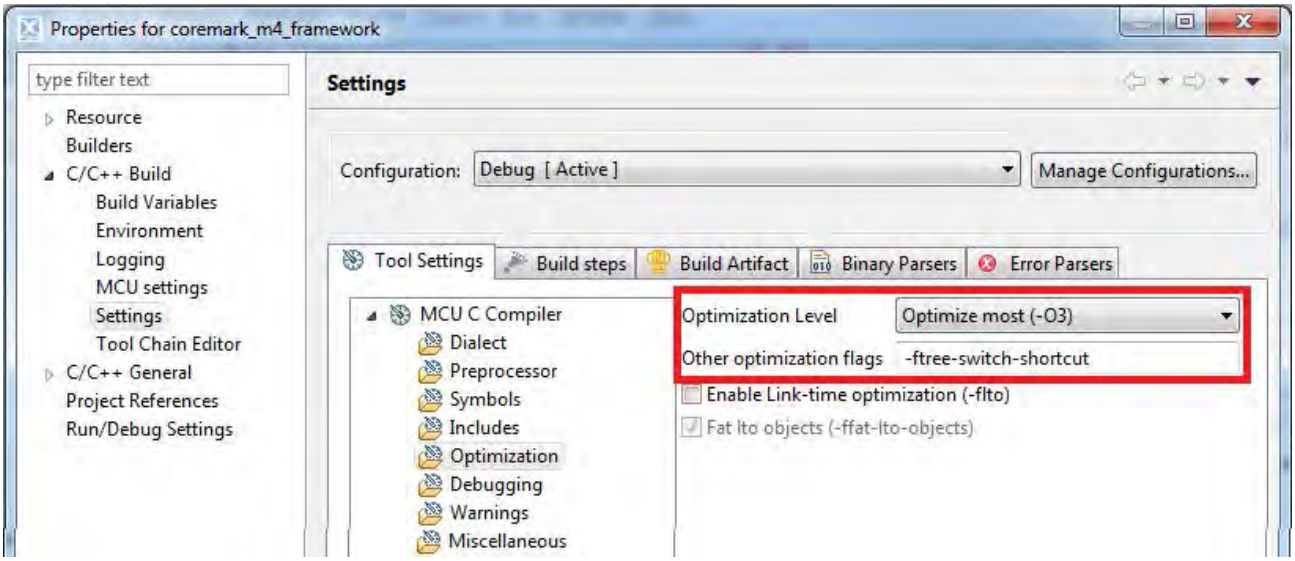

Fig 25. LPCXpresso CoreMark score optimizations

When benchmarking the power consumption of the MCU, set the optimization level to “None (-O0)”. The optimization flag does not have a meaningful effect on the power consumption and can be left unchanged. See Figure 26.

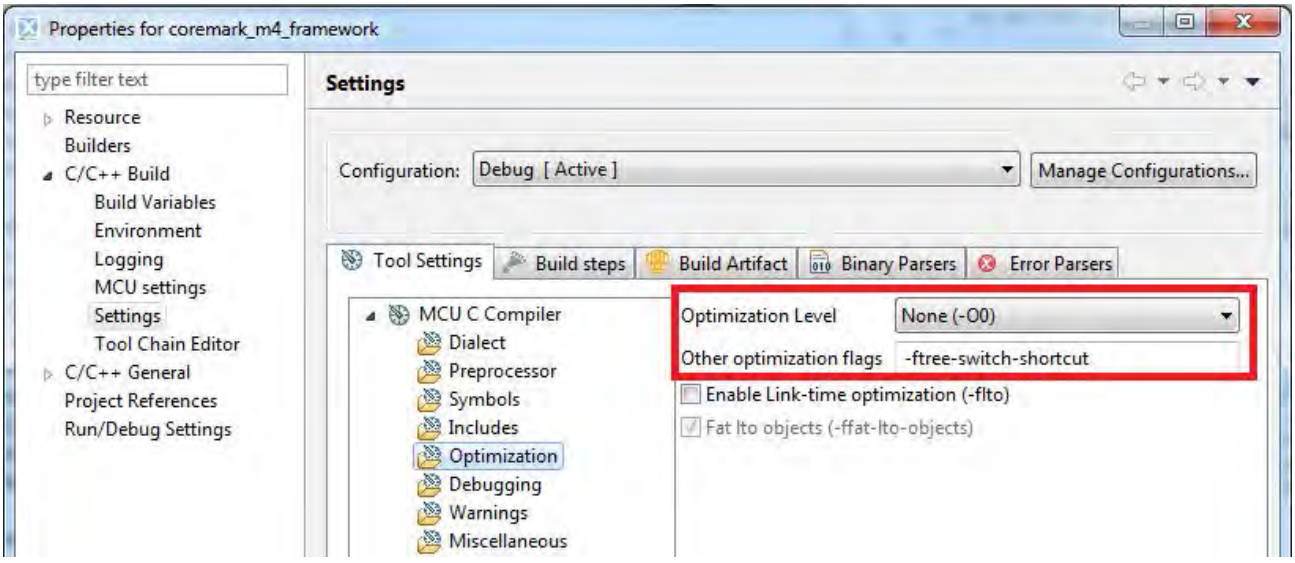

Fig 26. LPCXpresso µA/MHz optimizations

3. LPCXpresso LPC54102 board setup and macro defines

The LPC54102 LPCXpresso board supports a VCOM serial port connection via J6. 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.

3.1.1 Board setup

The LPCXpresso LPC54102 development board is used for the benchmarking.

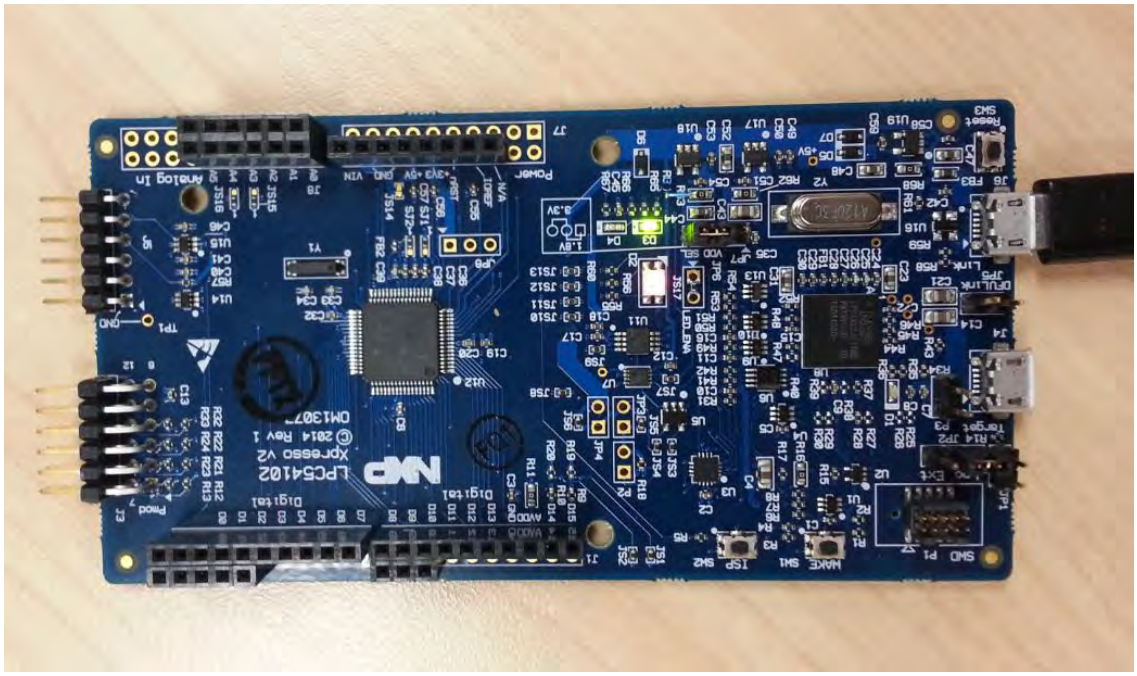

Fig 27. LPC54102 LPCXpresso development board

The board ships with CMSIS-DAP preprogrammed. For debugging and terminal debug messages, connect a USB cable to J6. Board schematics are available on LPCWare.com.

To benchmark any of the SRAM projects, be sure to follow Section 2.2.2 to pick the correct linker script and produce the best score.

3.2 µA/MHz Measurement setup

To measure the LPC54102 power consumption, remove JS6, install connector at JP4 and insert ammeter at JP4 as shown in Figure 28.

When performing the µA/MHz benchmark from the SRAM projects, NXP recommends using the J4 USB connector to only provide power to the necessary components on the LPC54102 board. Additionally, after the µA/MHz benchmark has been downloaded, power cycling the board by removing the USB cable and reinserting it is recommended to make sure the debug block is not left on by the debugger. When trying to flash a new program to the LPC54102, put the MCU into ISP mode. This may be necessary because

the flash is turned off by the µA/MHz benchmark, which may be problematic the next time the IDE tries to program the flash memory.

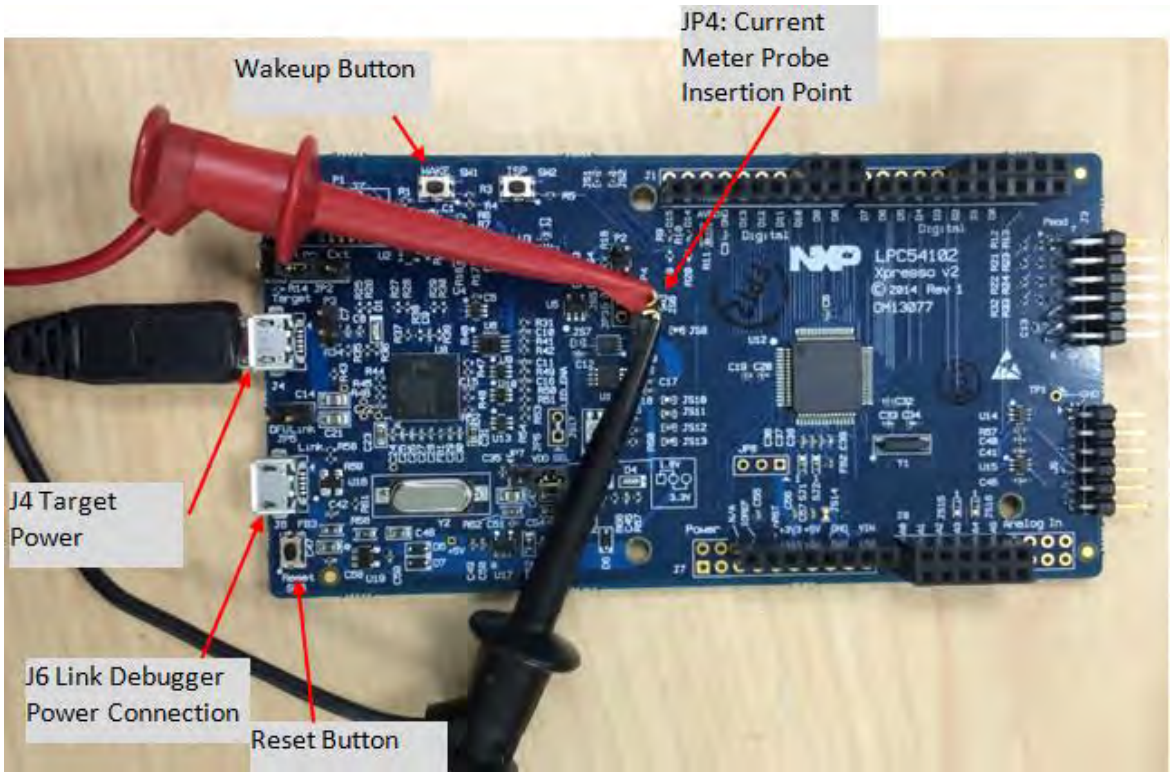

Fig 28. Current Meter Hookup

3.2.1 SRAM Linker Script Usage

For the SRAM projects, follow the instructions in this section to correctly configure the provided linker script and achieve the best uA/MHz score. For flash projects, this section can be skipped. The SRAM projects utilize the IDE's linker in two ways to optimize the CoreMark score benchmark and µA/MHz test. For this reason, there is an extra step that needs to be taken when switching between these two benchmarks.

The SRAM projects are configured to copy the relevant CoreMark code and data into separate SRAM banks. In Keil MDK, this done in by a scatterload file, located here:

\applications\lpc5410x\examples\coremark_m4\ coremark_m4.sct

In IAR EWARM, it is done in an .icf file, located here:

\applications\lpc5410x\iar\lpcxpresso_54102\multicore\coremark_m4\ coremark_m4.icf

In LPCXpresso, it is done by linker scripts in a folder located here:

\coremark_m4_framework\linkscripts\

The µA/MHz test is optimized by saving as much power as possible. In this case, turn off as many things possible, including the second SRAM bank that is utilized in the

CoreMark score benchmark. For Keil MDK, change the scatterload file to the one located here:

\applications\lpc5410x\examples\coremark_m4\ coremark_m4_power.sct

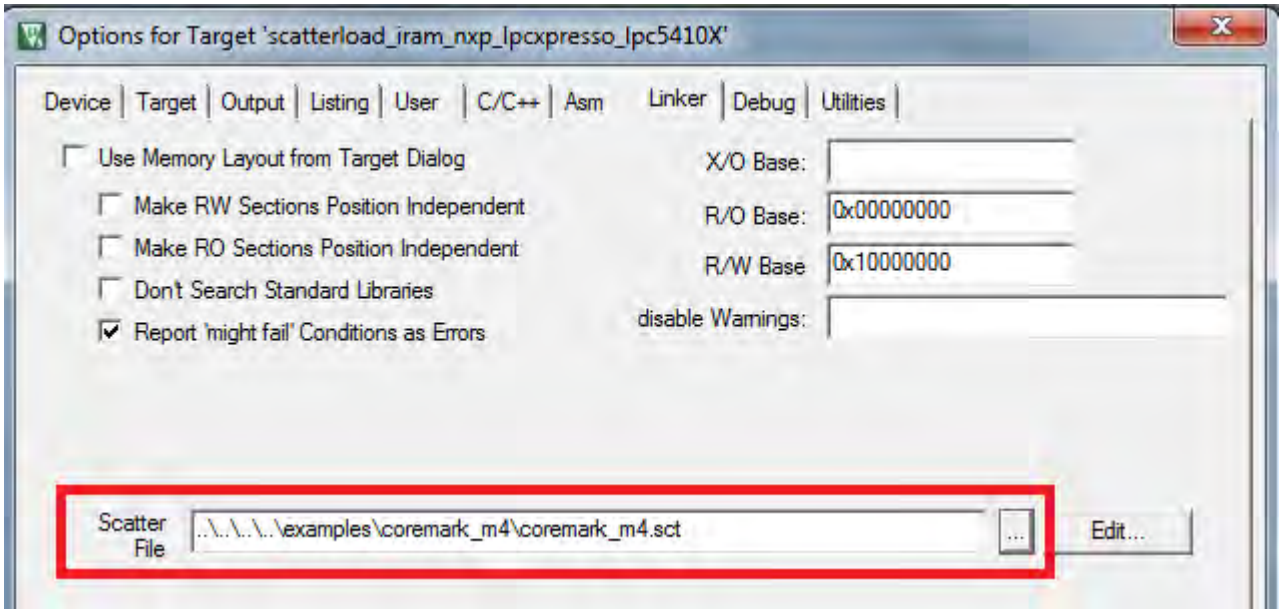

Fig 29. Changing the scatterload file in Keil MDK

In IAR EWARM, change the .icf file to the one located here:

\applications\lpc5410x\iar\lpcxpresso_54102\multicore\coremark_m4\ coremark_m4_power.icf

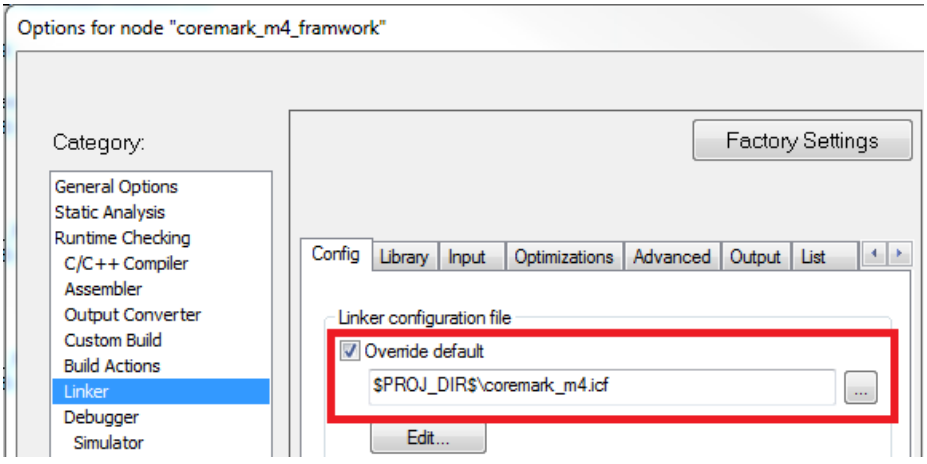

Fig 30. Changing the .icf file in IAR EWARM

In LPCXpresso, change the linker script folder path to the one located here:

\coremark_m4_framework\linkscripts_power\

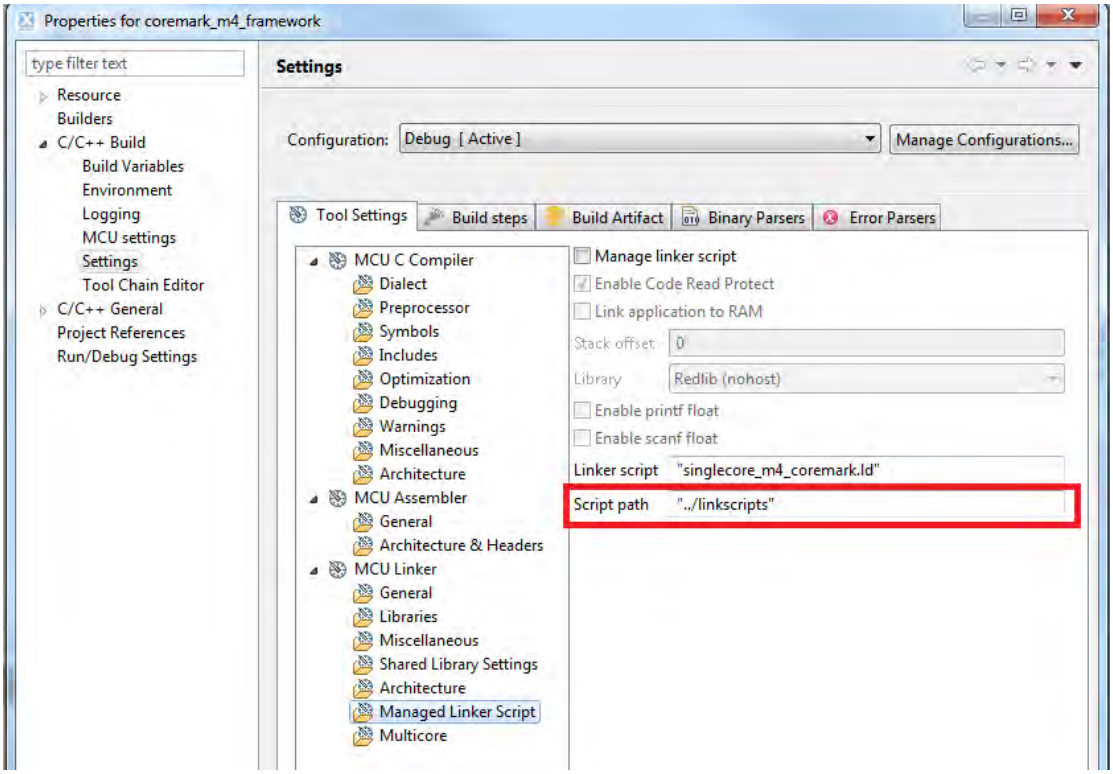

Fig 31. Changing linker script directory in LPCXpresso

It is important to have the appropriate scatterload file, .icf file, or linker script selected depending on the benchmark being executed. Failure to do so will either produce an un-optimized score or the inability to run the benchmark due to different amounts of SRAM used in the two benchmarks.

3.3 LPCOpen Board.h defines

The default baud rate setting for the debug messages is 115200. To change the baud rate, browse to the board.h file and update the DEBUGBAUDRATE define. If this #defines are changed then the board, chip, and project will need to be recompiled.

Changing the core clock frequency can also be implemented by setting this #define BOARD_MAINCLOCKRATE in board.h. If this #define is changed then the board, chip, and project will need to be recompiled.

board.h

```
74
75  /** Bit rate for the debug UART in Hz */
76  #define DEBUGBAUDRATE    (115200)
77
78  /** Main system clock rate in Hz for this board */
79  #define BOARD_MAINCLOCKRATE    (100000000) //100MHz
```

Fig 32. Board.h settings

4. Results

Figure 33 shows the result when running LPC5410x at 100MHz core clock from Keil MDK. The CoreMark benchmark score is the number of iterations per second. The CoreMark/MHz score for this run is 262.24/100MHz = 2.62 CoreMark/MHz.

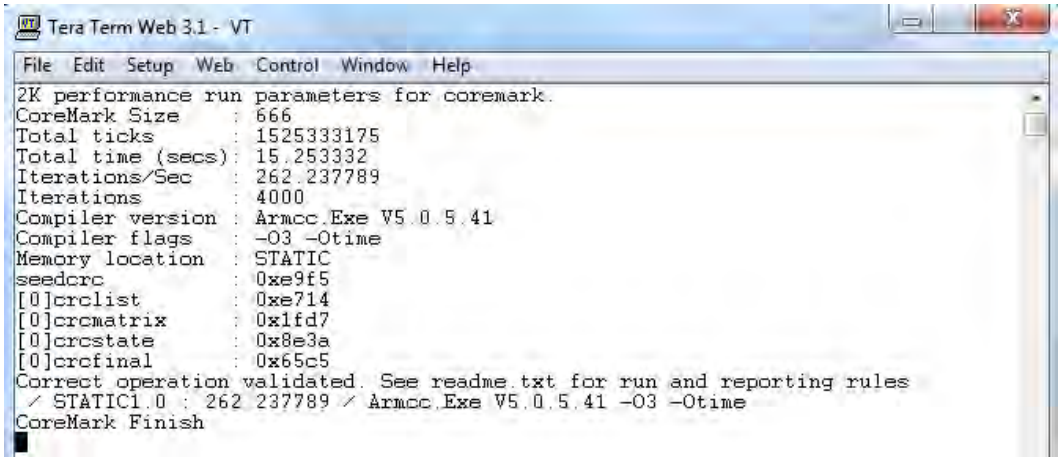

Fig 33. CoreMark result from RAM @100MHz

Table 1 shows typical CoreMark score when benchmarked on Keil MDK, IAR EWARM, and LPCXpresso when running from SRAM and flash at 100 MHz.

Table 1. LPC54102 LPCXpresso board CoreMark/MHz score

IDE	CoreMark/MHz Score (SRAM)	CoreMark/MHz Score (flash)
Keil MDK	2.62	2.21
IAR EWARM	3.33	2.69
LPCXpresso	2.45	2.10

For µA/MHz, the following tables show typical results that can be expected when running on the LPC5410x LPCXpresso board. A graph is also presented, comparing the three IDEs in terms of power consumption.

Table 2. Keil MDK µA/MHz score

Frequency	Average Power Consumption (SRAM)	µA/MHz Score (SRAM)	Average Power Consumption (flash)	µA/MHz Score (flash)
12 MHz	1.51 mA	125.67 µA/MHz	1.97 mA	164.17 µA/MHz
84 MHz	7.87 mA	93.73 µA/MHz	8.80 mA	104.76 µA/MHz
96 MHz	9.20 mA	95.85 µA/MHz	10.20 mA	106.25 µA/MHz
100 MHz	98.61 mA	98.61 µA/MHz	10.71 mA	107.1 µA/MHz

Table 3. IAR EWARM µA/MHz score

Frequency	Average Power Consumption (SRAM)	µA/MHz Score (SRAM)	Average Power Consumption (flash)	µA/MHz Score (flash)
12 MHz	1.542 mA	128.5 µA/MHz	2.16 mA	180 µA/MHz
84 MHz	8.18 mA	97.32 µA/MHz	9.2 mA	108.52 µA/MHz
96 MHz	9.57 mA	99.64 µA/MHz	10.56 mA	110 µA/MHz
100 MHz	10.25 mA	102.5 µA/MHz	11.22 mA	112.2 µA/MHz

Table 4. LPCXpresso µA/MHz score

Frequency	Average Power Consumption (SRAM)	µA/MHz Score (SRAM)	Average Power Consumption (flash)	µA/MHz Score (flash)
12 MHz	1.47 mA	122.67 µA/MHz	2.09 mA	174.17 µA/MHz
84 MHz	7.5 mA	89.29 µA/MHz	10.4 mA	123.81 µA/MHz
96 MHz	8.76 mA	91.26 µA/MHz	11.99 mA	124.9 µA/MHz
100 MHz	9.38 mA	93.83 µA/MHz	12.71 mA	127.1 µA/MHz

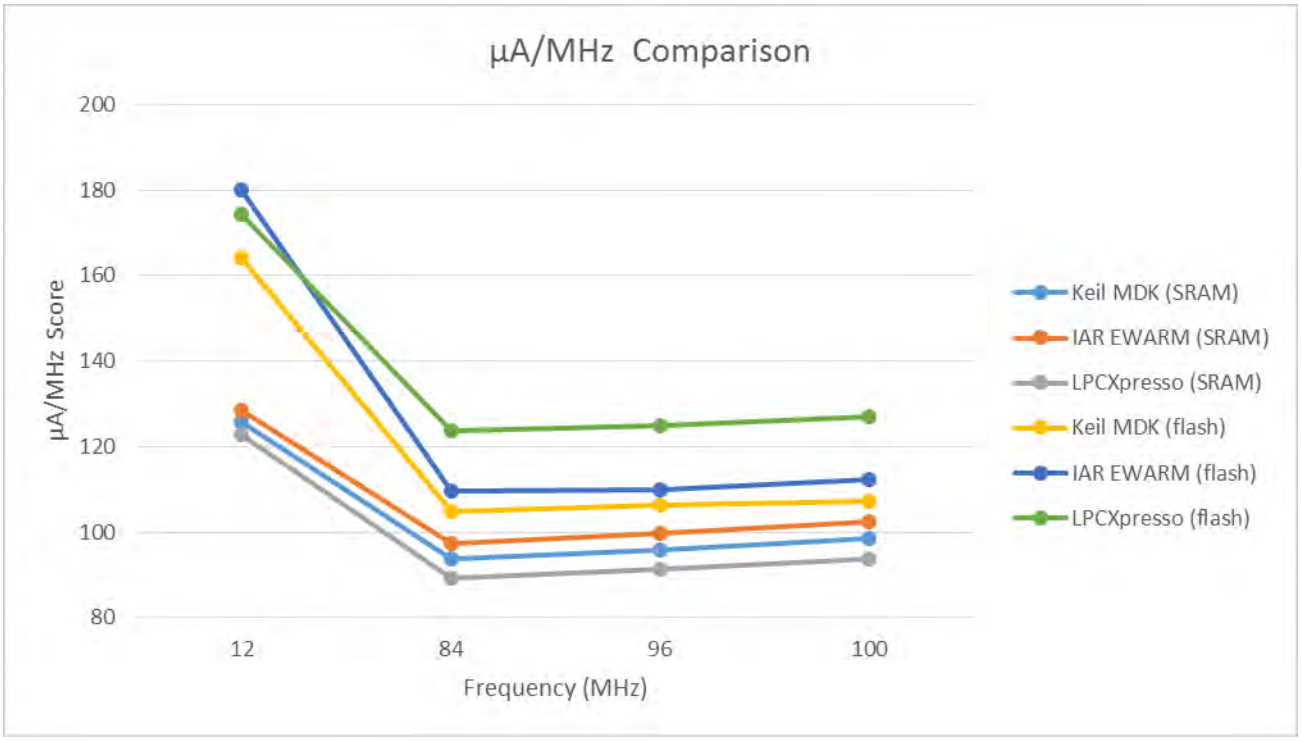

Fig 34. µA/MHz IDE Comparison

5. Conclusion

In this application note, the two types of benchmarks that can be done on the LPC5410x are presented: the CoreMark score benchmark and the µA/MHz benchmark. Details on how to optimize these benchmarks on the LPC5410x when running the benchmark out of SRAM and flash are discussed. These optimizations show how to achieve up to a competitive 3.3 CoreMark/MHz score and a best in class 100 µA/MHz for this device using the Cortex-M4 core.

6. Legal information

6.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

6.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the *Terms and conditions of commercial sale* of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Translations — A non-English (translated) version of a document is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Evaluation products — This product is provided on an "as is" and "with all faults" basis for evaluation purposes only. NXP Semiconductors, its affiliates and their suppliers expressly disclaim all warranties, whether express, implied or statutory, including but not limited to the implied warranties of non-infringement, merchantability and fitness for a particular purpose. The entire risk as to the quality, or arising out of the use or performance, of this product remains with customer.

In no event shall NXP Semiconductors, its affiliates or their suppliers be liable to customer for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out the use of or inability to use the product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages.

Notwithstanding any damages that customer might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP Semiconductors, its affiliates and their suppliers and customer's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by customer based on reasonable reliance up to the greater of the amount actually paid by customer for the product or five dollars (US$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose.

6.3 Licenses

Purchase of NXP <xxx> components
<License statement text>

6.4 Patents

Notice is herewith given that the subject device uses one or more of the following patents and that each of these patents may have corresponding patents in other jurisdictions.

<Patent ID> — owned by <Company name>

6.5 Trademarks

Notice: All referenced brands, product names, service names and trademarks are property of their respective owners.

<Name> — is a trademark of NXP Semiconductors N.V.

.

7. List of figures

8. List of tables

 All information provided in this document is subject to legal disclaimers.

9. Contents

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

Date of release: 14 July 2015

Document identifier: LPC5410x CoreMark Cortex-M4 Porting Guide