ADuCM4050 Reference Manual

UG-1161

One Technology Way • P.O. Box 9106 • Norwood, MA 02062-9106, U.S.A. • Tel: 781.329.4700 • Fax: 781.461.3113 • www.analog.com

How to Reproduce the EEMBC CoreMark and ULPMark Core Profile Score for the ADuCM4050

INTRODUCTION

This reference manual describes how to reproduce the Embedded Microprocessor Benchmark Consortium (EEMBC®) CoreMark® and ULPMark™-Core Profile (CP) score for the ADuCM4050 microcontroller.

This reference manual describes the steps necessary to install the software and to set up the hardware for measuring the CoreMark and ULPMark-CP score.

This reference manual helps the user reproduce the EEMBC CoreMark and ULPMark-CP score on the EV-COG-AD4050LZ or the EV-COG-AD4050WZ board.

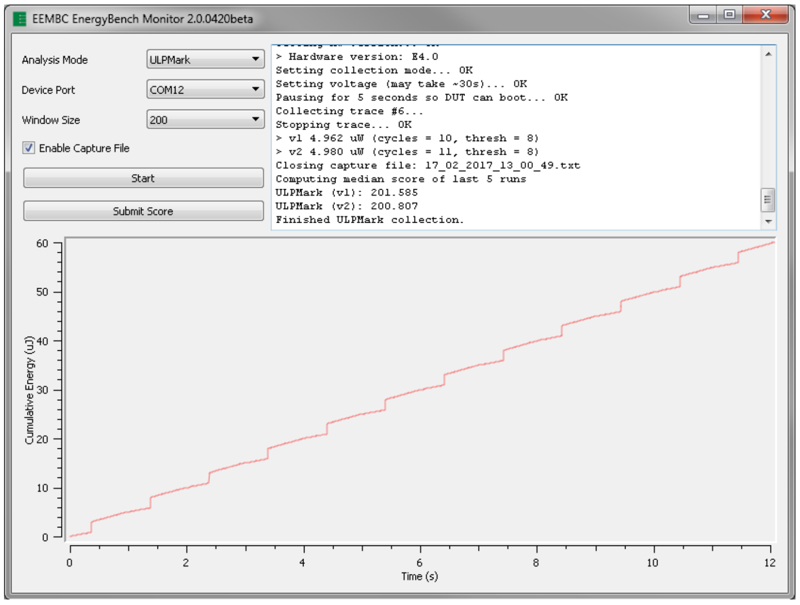
This reference manual provides details of the performance and the energy consumed by the ADuCM4050 microcontroller in the different power modes used on the benchmark, confirming the data sheet power specifications.

ABOUT THE ADuCM4050

The ADuCM4050 processor is an ultralow power, integrated, mixed-signal, microcontroller system used for processing, control, and connectivity. The microcontroller unit (MCU) subsystem is based on the ARM® Cortex™-M4F processor, a collection of digital peripherals, cache embedded SRAM and flash memory, and an analog subsystem that provides clocking, reset, and power management capabilities along with the analog-to-digital converter (ADC).

The ADuCM4050 processor provides a collection of power modes and features, such as dynamic and software controlled clock gating and power gating, to support extremely low dynamic and hibernate power management.

Full specifications on the ADuCM4050 are available in the product data sheet. Consult the data sheet in conjunction with this reference manual when working with the EV-COG-AD4050LZ and the EV-COG-AD4050WZ.



Analysis Mode: ULPMark
Device Port: COM12
Window Size: 200
☑ Enable Capture File

```
> Hardware version: E4.0
Setting collection mode... OK
Setting voltage (may take ~30s)... OK
Pausing for 5 seconds so DUT can boot... OK
Collecting trace #6...
Stopping trace... OK
> v1 4.962 uW (cycles = 10, thresh = 8)
> v2 4.980 uW (cycles = 11, thresh = 8)
Closing capture file: 17_02_2017_13_00_49.txt
Computing median score of last 5 runs
ULPMark (v1): 201.585
ULPMark (v2): 200.807
Finished ULPMark collection.
```

Figure 1. EEMBC ULPMark-CP Score

TABLE OF CONTENTS

REVISION HISTORY

6/2018—Revision 0: Initial Version

ABOUT EEMBC

EEMBC is a nonprofit industry association that detected the need for a joint, democratic effort involving the leading suppliers in the embedded industry to make new benchmarks a reality.

EEMBC members represent more than 40 of the world's leading semiconductor, intellectual property, compiler, real-time operating system, and system companies. Furthermore, EEMBC is licensed by more than 80 companies and more than 100 universities worldwide. Through the combined efforts of its members, EEMBC benchmarks have become an industry standard for evaluating the capabilities of embedded processors and systems according to objective, clearly defined, application-based criteria.

EEMBC has benchmark suites targeting cloud and big data, mobiles devices (for phones and tablets), networking, ultralow power microcontrollers, the Internet of Things (IoT), digital media, automotive, and other application areas. EEMBC also has benchmarks for general-purpose performance analysis including CoreMark, MultiBench (multicore), and FPMark (floating point).

This reference manual focuses on the CoreMark and ultralow power microcontrollers benchmarks, targeted to measure the power processing and the MCU energy efficiency ,respectively, because these aspects are key features of the ADuCM4050 processor.

CoreMark

To select an MCU for a specific application, the user must know if the MCU has enough processing power to meet the application requirements. Several benchmarking options are available. Dhrystone is the most widely used benchmarking option; however, this option has a few problems, such as requiring library calls to be within a timed portion and susceptibility to the ability of a compiler to optimize work. To address these problems and to provide a simple, open source benchmark, EEMBC created the CoreMark.

CoreMark is a benchmark that measures the performance of central processing units (CPUs) used in embedded systems. CoreMark was developed in 2009 at EEMBC and is intended as an industry standard, replacing the antiquated Dhrystone benchmark. Written in C, the code contains implementations of the following algorithms:

- List processing (find and sort)
- Matrix manipulation (common matrix operations)
- State machine (determines if an input stream contains valid numbers)
- Cyclic redundancy check (CRC)

ULPMark

Whether the target is edge nodes for the IoT or any other type of battery-powered application, the implications of ultralow power (ULP) varies. The lowest active current is required when the power source is severely limited (for example, energy harvesting). The lowest sleep current is required when the system spends most of its time in standby or sleep mode, waking up infrequently (periodically or asynchronously) to process a task. ULP also implies great energy efficiency, whereby the most work is performed in a limited time. Overall, the application requires a combination of trade-offs on all the previously mentioned criteria. To ensure ULP operation over periods of months, years, and decades, application developers face a number of optimization challenges. There are an increasing number of microcontrollers claiming ULP capabilities; however, developers cannot rely on data sheet parameters alone. The EEMBC ULPMark standardizes data sheet parameters and provides a methodology to reliably and equitably measure MCU energy efficiency.

The foundations of ULPMark are as follows:

- Comparability, making it easy to compare devices.
- Transparency, making all measurements and setup processes transparent.
- Reproducibility, making it easy for any user to reproduce the benchmark scores.

IAR SETUP

IAR TOOLS INSTALLATION

The IAR Embedded Workbench® and the included IAR C/C++ Compiler™ generates the fastest performing, most compact code in the industry for ARM-based applications. Therefore, Analog Devices, Inc., provides the device family package (DFP) for the ADuCM4050.

Support for the ADuCM4050 is provided in the DFP.

The IAR KickStart Kit™ is a free starter kit and evaluation version of IAR™. This edition has limitations both in code size (32 kB) and in the service and support provided.

Download the IAR Embedded Workbench from the IAR website.

IAR PROJECT CONFIGURATION

This section describes the IAR configuration for proper operation. Only the sections that must be modified from the default values are mentioned.

Use the following procedure to configure the IAR:

1. Right click the name of the project and click **Options…**, as shown in Figure 2.

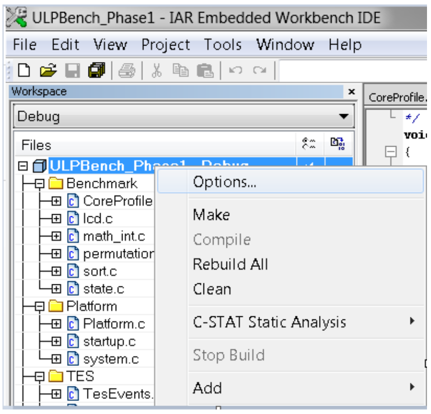
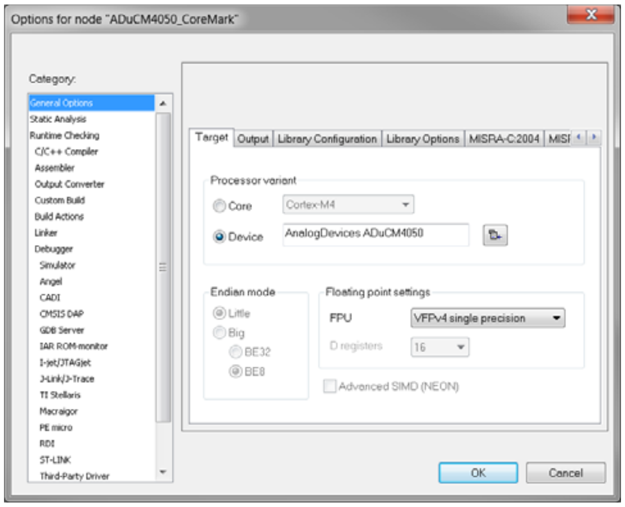


Figure 2. Project Options

2. Under **General Options**, ensure that **AnalogDevices ADuCM4050** is selected as the target, depending on the microcontroller used.



Processor variant
Core [Cortex-M4]
[✓ Device] AnalogDevices ADuCM4050

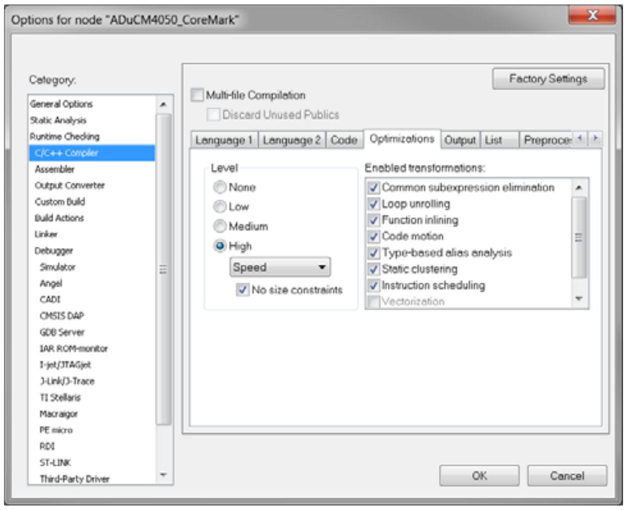
Endian mode
[✓ Little]
Big
BE32
[✓ BE8]

Floating point settings
FPU: VFPv4 single precision
D registers: 16
Advanced SIMD (NEON)

Figure 3. **General Options**—Target Configuration

3. Under **C/C++ Compiler**, ensure that the optimization for high speed is chosen (see Figure 4). Also, check the **No size constraints** option. Some functions, such as **pltInitialize**, are protected to ensure that these functions are not optimized. The following code protects a function and prevents the compiler from modifying the function code:

```
#pragma optimize=none
```



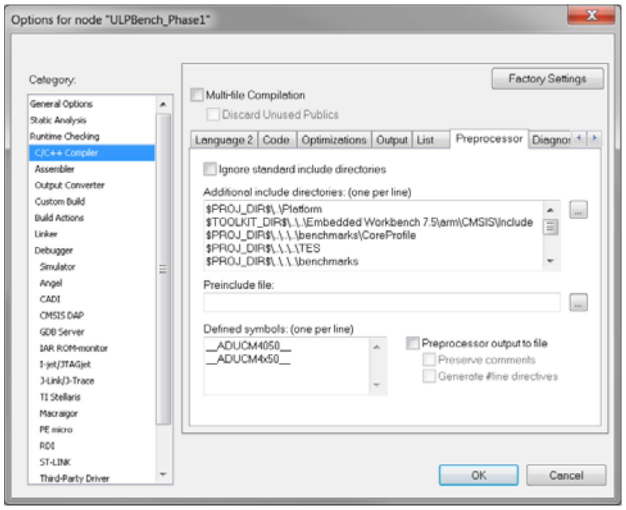
Multi-file Compilation
Discard Unused Publics

Level
None
Low
Medium
[✓ High]
Speed
[✓ No size constraints]

Enabled transformations:
[✓ Common subexpression elimination]
[✓ Loop unrolling]
[✓ Function inlining]
[✓ Code motion]
[✓ Type-based alias analysis]
[✓ Static clustering]
[✓ Instruction scheduling]
Vectorization

Figure 4. **C/C++ Compiler**—Optimizations Configuration

Figure 5 shows the included directories path and the defined symbols settings necessary for a proper compilation of the ULPMark-CP project.


Options for node "ULPBench_Phase1"

Multi-file Compilation: unchecked
Ignore standard include directories: unchecked

Additional include directories: (one per line)
```
$PROJ_DIR$\..\Platform
$TOOLKIT_DIR$\..\..\Embedded Workbench 7.5\arm\CMSIS\Include
$PROJ_DIR$\..\..\..\benchmarks\CoreProfile
$PROJ_DIR$\..\..\..\TES
$PROJ_DIR$\..\..\..\benchmarks
```
Preinclude file:

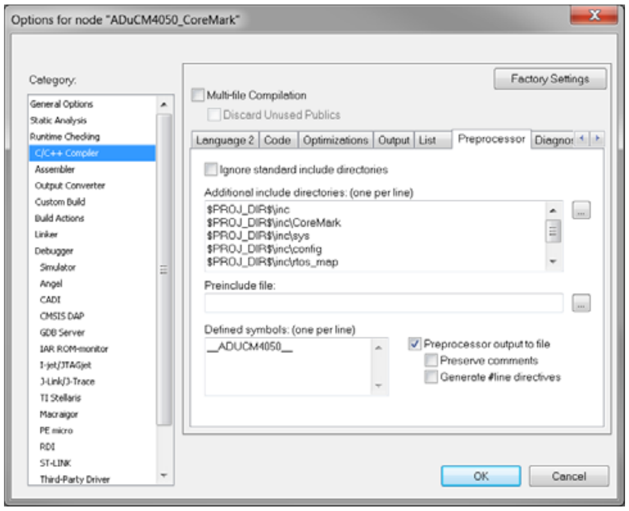
Defined symbols: (one per line)
```
__ADUCM4050__
__ADUCM4x50__
```
Preprocessor output to file: unchecked

*Figure 5. **C/C++ Compiler**—Preprocessor ULPMark-CP Configuration*

The full list of necessary paths for the ULPMark-CP project is as follows:

```
$PROJ_DIR$\..\Platform
$TOOLKIT_DIR$\..\..\Embedded Workbench
7.5\arm\CMSIS\Include
$PROJ_DIR$\..\..\..\benchmarks\CoreProfil
e
$PROJ_DIR$\..\..\..\TES
$PROJ_DIR$\..\..\..\benchmarks
$PROJ_DIR$\..\inc
$PROJ_DIR$\..\inc\config
$PROJ_DIR$\..\inc\drivers
$PROJ_DIR$\..\inc\rtos_map
$PROJ_DIR$\..\inc\sys
```

Figure 6 shows the included directories path and the defined symbol settings necessary for a proper compilation of the CoreMark project.


Options for node "ADuCM4050_CoreMark"

Multi-file Compilation: unchecked
Ignore standard include directories: unchecked

Additional include directories: (one per line)
```
$PROJ_DIR$\inc
$PROJ_DIR$\inc\CoreMark
$PROJ_DIR$\inc\sys
$PROJ_DIR$\inc\config
$PROJ_DIR$\inc\rtos_map
```
Preinclude file:

Defined symbols: (one per line)
```
__ADUCM4050__
```
Preprocessor output to file: ✓

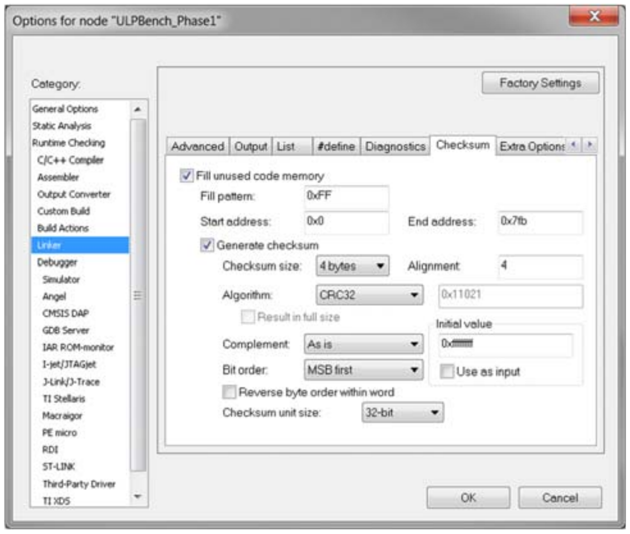
*Figure 6. **C/C++ Compiler**—Preprocessor CoreMark Configuration*

The full list of necessary paths for the CoreMark project is as follows:

```
$PROJ_DIR$\inc
$PROJ_DIR$\inc\CoreMark
$PROJ_DIR$\inc\sys
$PROJ_DIR$\inc\config
$PROJ_DIR$\inc\rtos_map
$TOOLKIT_DIR$\inc\AnalogDevices
$TOOLKIT_DIR$\CMSIS\Include
```

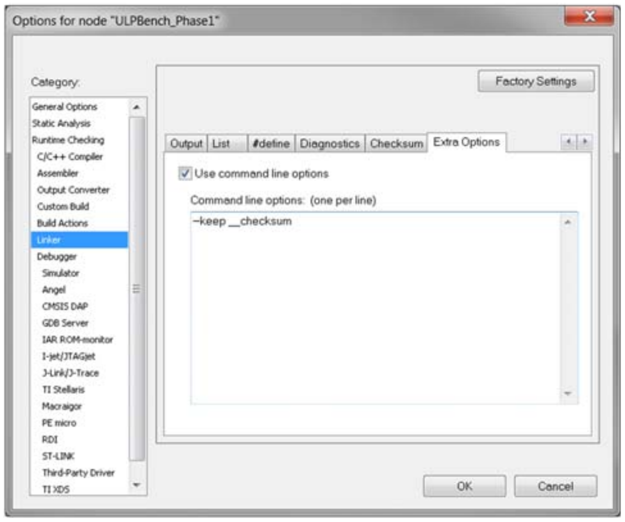
To avoid undesired warnings, add the following diagnostics to the **Suppress following diagnostics** option: Pa050 and Pa082.

4. A 32-bit cyclic redundancy check (CRC) checksum stored in the **Signature** field enables user code to request an integrity check of user space. The user can configure the checksum as shown in Figure 7 and Figure 8 (both configurations can be found in the **Linker** menu).


Options for node "ULPBench_Phase1"

Fill unused code memory: ✓
Fill pattern: 0xFF
Start address: 0x0
End address: 0x7fb
Generate checksum: ✓
Checksum size: 4 bytes
Alignment: 4
Algorithm: CRC32
Complement: As is
Bit order: MSB first
Initial value: 0xffffffff
Use as input: unchecked
Reverse byte order within word: unchecked
Checksum unit size: 32-bit

*Figure 7. **Linker** Menu—**Checksum** Tab*

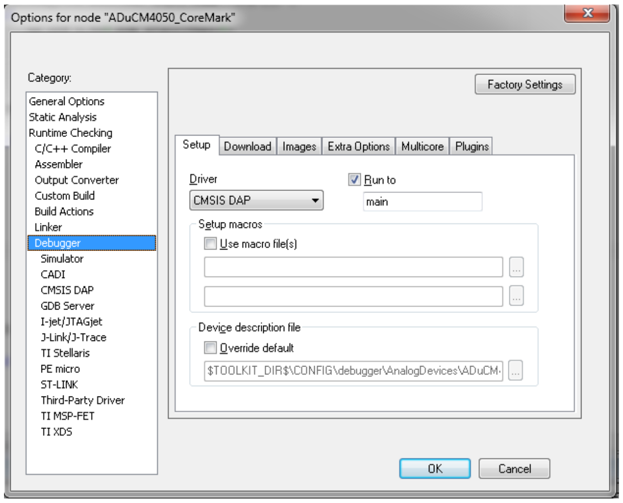

Options for node "ULPBench_Phase1"

Use command line options: ✓

Command line options: (one per line)
```
--keep __checksum
```

*Figure 8. **Linker** Menu—**Extra Options** Tab*

5. The debugger used is **CMSIS DAP** (see Figure 9). Verify that both **Verify download** and **Use flash loader(s)** are checked on the **Debugger** > **Download** menu, as shown in Figure 10.



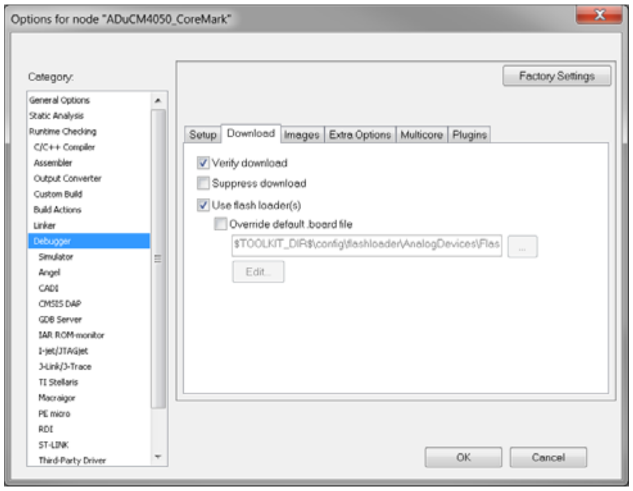
Options for node "ADuCM4050_CoreMark"

Category: General Options, Static Analysis, Runtime Checking, C/C++ Compiler, Assembler, Output Converter, Custom Build, Build Actions, Linker, Debugger (selected), Simulator, CADI, CMSIS DAP, GDB Server, I-jet/JTAGjet, J-Link/J-Trace, TI Stellaris, PE micro, ST-LINK, Third-Party Driver, TI MSP-FET, TI XDS

Tabs: Setup | Download | Images | Extra Options | Multicore | Plugins

Driver: CMSIS DAP
[✓] Run to: main
Setup macros: [ ] Use macro file(s)
Device description file: [ ] Override default
$TOOLKIT_DIR$\CONFIG\debugger\AnalogDevices\ADuCM-

Figure 9. **Debugger**—Setup Configuration



Options for node "ADuCM4050_CoreMark"

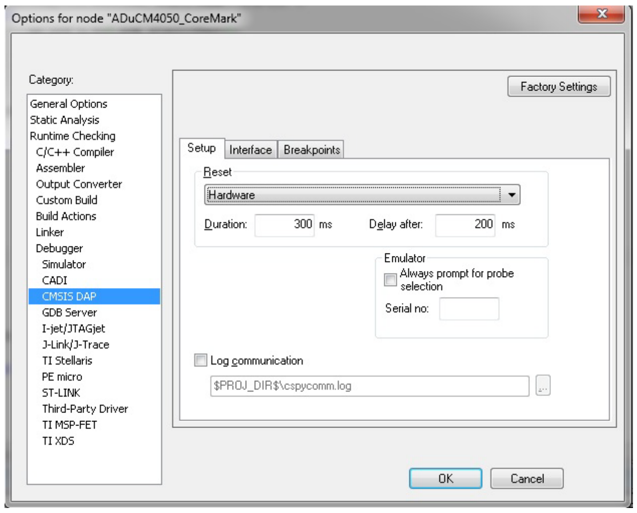
Category: General Options, Static Analysis, Runtime Checking, C/C++ Compiler, Assembler, Output Converter, Custom Build, Build Actions, Linker, Debugger (selected), Simulator, Angel, CADI, CMSIS DAP, GDB Server, IAR ROM-monitor, I-jet/JTAGjet, J-Link/J-Trace, TI Stellaris, Macraigor, PE micro, RDI, ST-LINK, Third-Party Driver

Tabs: Setup | Download | Images | Extra Options | Multicore | Plugins

[✓] Verify download
[ ] Suppress download
[✓] Use flash loader(s)
[ ] Override default .board file
$TOOLKIT_DIR$\config\flashloader\AnalogDevices\Flas

Figure 10. **Debugger**—Download Configuration

6. Figure 11 and Figure 12 show the **CMSIS DAP** configuration. Use the **Hardware** target reset.



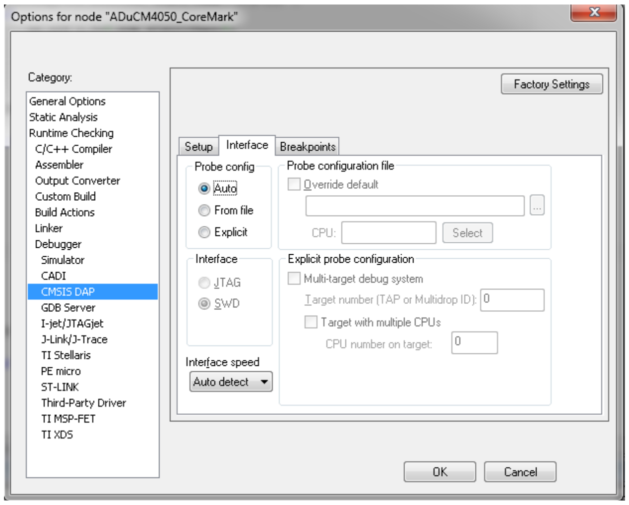
Options for node "ADuCM4050_CoreMark"

Category: General Options, Static Analysis, Runtime Checking, C/C++ Compiler, Assembler, Output Converter, Custom Build, Build Actions, Linker, Debugger, Simulator, CADI, CMSIS DAP (selected), GDB Server, I-jet/JTAGjet, J-Link/J-Trace, TI Stellaris, PE micro, ST-LINK, Third-Party Driver, TI MSP-FET, TI XDS

Tabs: Setup | Interface | Breakpoints

Reset: Hardware
Duration: 300 ms Delay after: 200 ms
Emulator: [ ] Always prompt for probe selection
Serial no:
[ ] Log communication
$PROJ_DIR$\cspycomm.log

Figure 11. **CMSIS DAP**—Setup Configuration



Options for node "ADuCM4050_CoreMark"

Category: General Options, Static Analysis, Runtime Checking, C/C++ Compiler, Assembler, Output Converter, Custom Build, Build Actions, Linker, Debugger, Simulator, CADI, CMSIS DAP (selected), GDB Server, I-jet/JTAGjet, J-Link/J-Trace, TI Stellaris, PE micro, ST-LINK, Third-Party Driver, TI MSP-FET, TI XDS

Tabs: Setup | Interface | Breakpoints

Probe config: [✓ Auto] From file Explicit
Probe configuration file: [ ] Override default
CPU:
Interface: JTAG [✓ SWD]
Explicit probe configuration: [ ] Multi-target debug system
Target number (TAP or Multidrop ID): 0
[ ] Target with multiple CPUs
CPU number on target: 0
Interface speed: Auto detect

Figure 12. **CMSIS DAP**—Connection Configuration

CoreMark

LOADING THE CoreMark PROJECT

The project with the CoreMark source files and **core_portme.*** files are tuned to the Analog Devices platform. The following steps describe how to add the CoreMark project on IAR.

1. Open the IAR workbench.
2. Open the project in IAR.
3. Under **Project**, click **Add Existing Project…**, as shown in Figure 13.

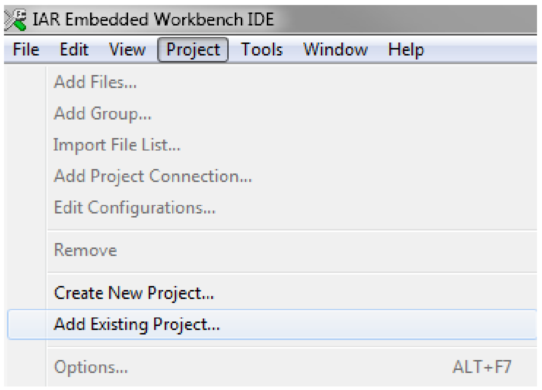


Figure 13. ***Adding an Existing Project***

4. Browse through the project obtained and open the .ewp extension file. The files available in the workspace are shown in Figure 14.

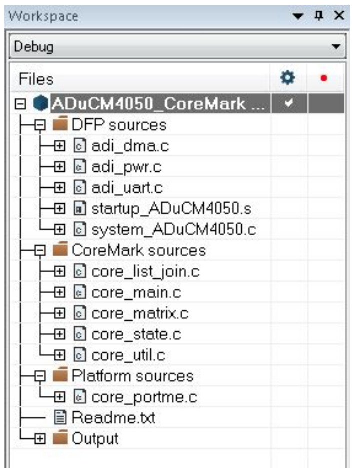


Figure 14. Project Files

The **DFP sources** folder includes the DFP files for configuring the device properly. The **CoreMark sources** folder includes the source files given by the EEMBC. The **Platform sources** folder contains the **core_portme.c** file given by the EEMBC but tuned to configure properly the tested device (in this case, the ADuCM4050 processor).

EEMBC does not restrict changing the **core_portme.*** files to suit the Analog Devices platform. The differences between the **core_portme.c** file and the file given by the EEMBC are as follows:

- Code for UART printing.
- Code for calculating the ticks of execution using the oscillator and crystal.
- Code to configure the microcontroller properly.
- Header files to support these codes.
- Device configuration. Note that the high power buck is enabled to reduce the power consumption; this is useful during power measurement when CoreMark is running (see the Power Measurements section).

The **core_portme.h** file has three definitions:

- The UART_PRINT definition prints the result through the UART. If this definition is commented, the results are printed only on the terminal input and output.
- The crystal definition decides whether to measure the ticks using an external crystal oscillator or the internal resistor capacitor oscillator. If this definition is commented, the internal oscillator is used; otherwise, the crystal oscillator is used.
- The phase-locked loop (PLL) definition enables the PLL. The processor runs at 52 MHz. By default, this definition is commented and the processor runs at 26 MHz.

RUNNING THE CoreMark APPLICATION

Building the CoreMark Application

To build or compile the application code, take one of the following steps:

- Click **Project** and select **Rebuild All** (as shown in Figure 15).
- Right click the project name and click **Rebuild All**, as shown in Figure 16. The user is then prompted to save the workspace in the .eww extension. The project must build without any errors.

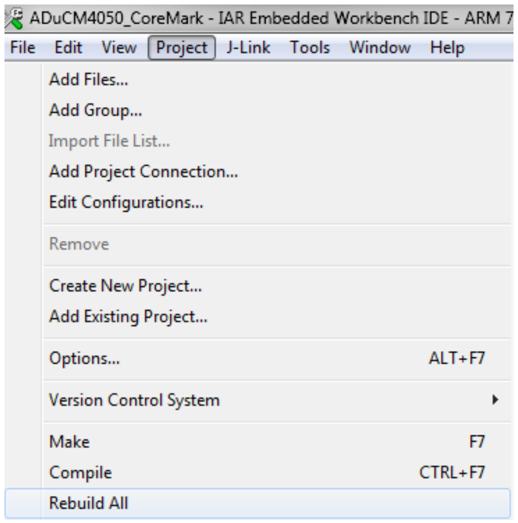


Figure 15. Start to Build the Project

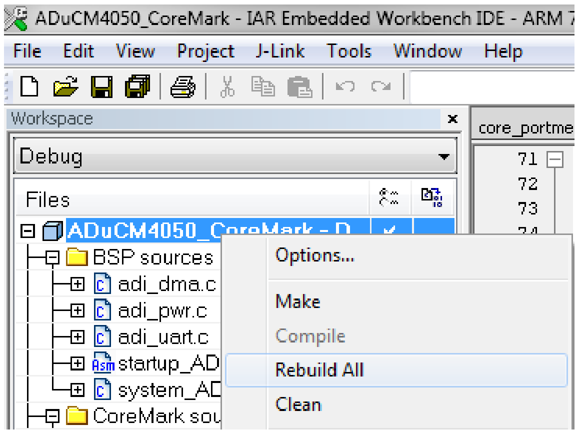
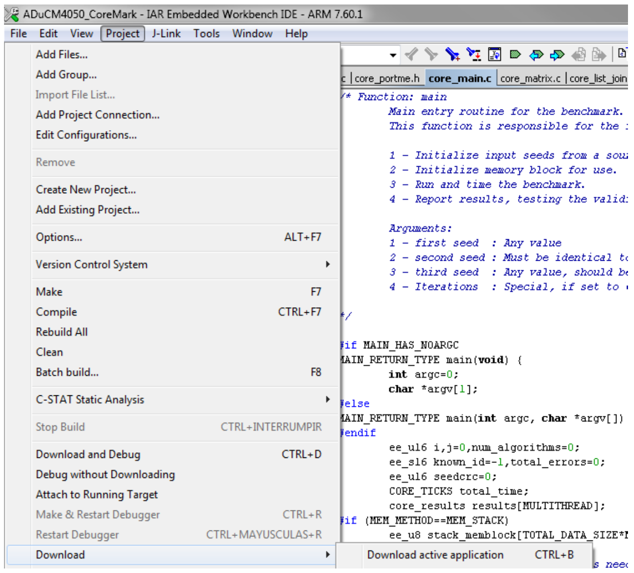


Figure 16. Building the Project

Downloading the CoreMark Code

To load the code onto the EV-COG-AD4050LZ or the EV-COG-AD4050WZ board, take one of the following steps:

- Under **Project**, click **Download** and select **Download active application**, as show in Figure 17.
- Click **Download and Debug**, as shown in Figure 18.


ADuCM4050_CoreMark - IAR Embedded Workbench IDE - ARM 7.60.1
File Edit View Project J-Link Tools Window Help
Add Files...
Add Group...
Import File List...
Add Project Connection...
Edit Configurations...
Remove
Create New Project...
Add Existing Project...
Options... ALT+F7
Version Control System
Make F7
Compile CTRL+F7
Rebuild All
Clean
Batch build... F8
C-STAT Static Analysis
Stop Build CTRL+INTERRUMPIR
Download and Debug CTRL+D
Debug without Downloading
Attach to Running Target
Make & Restart Debugger CTRL+R
Restart Debugger CTRL+MAYUSCULAS+R
Download
Download active application CTRL+B
core_portme.h core_main.c core_matrix.c core_list_join
16075-017

```
/* Function: main
        Main entry routine for the benchmark.
        This function is responsible for the

        1 - Initialize input seeds from a sou
        2 - Initialize memory block for use.
        3 - Run and time the benchmark.
        4 - Report results, testing the valid

        Arguments:
        1 - first seed  : Any value
        2 - second seed : Must be identical to
        3 - third seed  : Any value, should be
        4 - Iterations  : Special, if set to

*/

#if MAIN_HAS_NOARGC
MAIN_RETURN_TYPE main(void) {
        int argc=0;
        char *argv[1];
#else
MAIN_RETURN_TYPE main(int argc, char *argv[])
#endif
        ee_u16 i,j=0,num_algorithms=0;
        ee_s16 known_id=-1,total_errors=0;
        ee_u16 seedcrc=0;
        CORE_TICKS total_time;
        core_results results[MULTITHREAD];
#if (MEM_METHOD==MEM_STACK)
        ee_u8 stack_memblock[TOTAL_DATA_SIZE*M
                                            s need
```

Figure 17. Downloading the Code

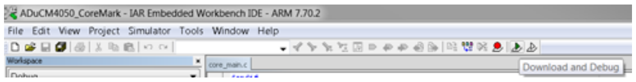

ADuCM4050_CoreMark - IAR Embedded Workbench IDE - ARM 7.70.2
File Edit View Project Simulator Tools Window Help
Workspace
Debug
core_main.c
Download and Debug
16075-018

*Figure 18. **Download and Debug** Button*

Running the CoreMark Project

To run the code, click **Go**, as shown in Figure 19.

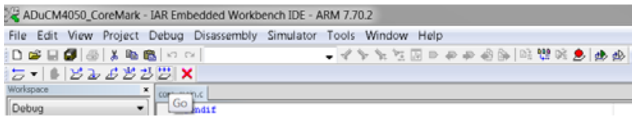
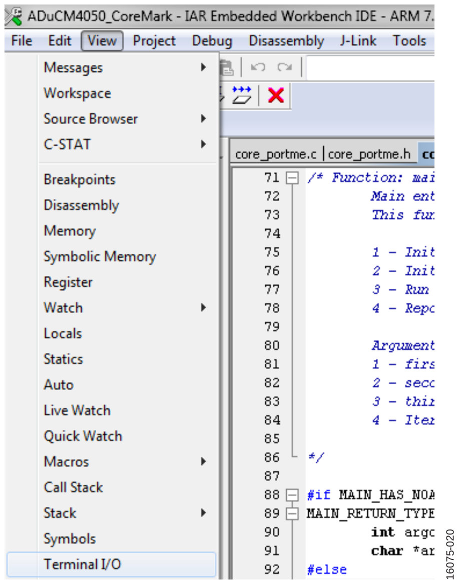

ADuCM4050_CoreMark - IAR Embedded Workbench IDE - ARM 7.70.2
File Edit View Project Debug Disassembly Simulator Tools Window Help
Workspace
Debug
Go
16075-019

Figure 19. Running the Project

CoreMark RESULTS

The requirement is that the CoreMark code must run for at least 10 seconds. The provided code has 10,000 iterations set up, resulting in approximately 2 minutes to complete the execution.

The results are printed out on the terminal input and output. To view the terminal input and output, click **View** and then select **Terminal I/O**. Note that users must be in debug mode for this option to be active.



```
/* Function: mai
        Main ent
        This fur

        1 - Init
        2 - Init
        3 - Run
        4 - Repo

        Argument
        1 - firs
        2 - secc
        3 - thiı
        4 - Iteı

*/

#if MAIN_HAS_NOA
MAIN_RETURN_TYPE
        int argc
        char *ar
#else
```

Figure 20. View Terminal Input and Output

By default, the UART_PRINT definition in the **core_portme.h** file is commented. To print the results through UART, uncomment the UART_PRINT definition.

The following steps describe how to print the results through UART:

1. Uncomment the UART_PRINT definition.
2. Connect the UART port of the EV-COG-AD4050LZ or the EV-COG-AD4050WZ to the PC using a USB cable.

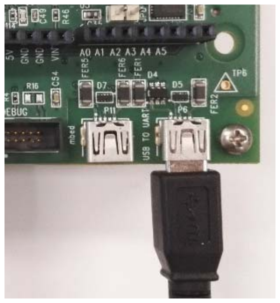
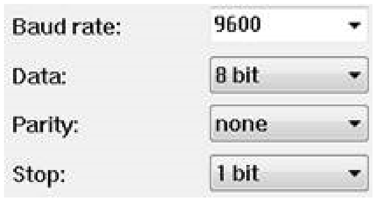


Figure 21. USB to UART Connection

3. From the **Control Panel**, click **Device Manager**.
4. Check the COM port number to which the UART is connected.
5. Open a terminal that can connect to the UART port (PuTTY is used here).
6. Set the **Connection** type to **Serial** and input the corresponding COM port number. The other settings are shown in Figure 22.



Baud rate:	9600
Data:	8 bit
Parity:	none
Stop:	1 bit

Figure 22. Additional UART Configuration

Rebuild the project, following the instructions in the Running the CoreMark Project section. The results are printed through UART (see Figure 24).

The CoreMark number shows the raw horsepower, and the CoreMark/MHz number shows the efficiency of the core. To calculate the CoreMark/MHz number, the CoreMark number must be divided by the clock speed used when the benchmark is performed.

CoreMark (MHz) = *CoreMark Score*/*Clock Frequency*

Results at 26 MHz

In this project, the ADuCM4050 processors run at 26 MHz.

CoreMark (MHz) = 91.59/26 MHz

The CoreMark/MHz score is 3.52.

This score is almost equal to the CoreMark/MHz score of the ARM Cortex-M4F processor, whose score is 3.40.

To report the score, CoreMark recommends the following format:

```
CoreMark 1.0 : 91.591873 / IAR EWARM
7.60.1.11216 --no_size_constraints --
cpu=Cortex-M4 -D __ADUCM4050__ --
no_code_motion -Ohs -e --fpu=VFPv4_sp --
endian=little  / FLASH
```

Power Measurements at 26 MHz

The following steps describe how to monitor the current consumption of the ADuCM4050 processor when the device executes the CoreMark code:

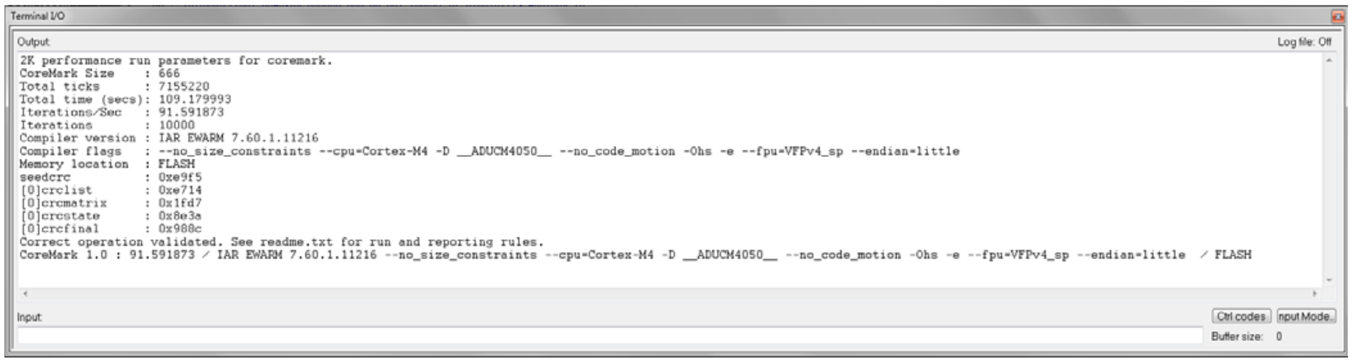
1. Ensure that UART_PRINT define in the **core_portme.h** file is commented so that the UART pins are not floating.
2. Load the code onto the microcontroller.
3. Connect the positive terminal of the source to the JP5 connector.
4. Connect the GND of the source to the GND of the board.
5. Remove all the jumpers.
6. Press the **Reset** button.
7. Monitor the current consumption on the meter. The current consumption must be approximately 1645 µA when the processor executes the code at 26 MHz.
8. For dynamic current consumption, repeat the procedure with a different frequency. Change the CLKDIV definition variable to 4 so that the frequency is divided by 4, yielding a value of 26 MHz/4 = 6.5 MHz.
9. Monitor the current consumption on the meter. The current consumption must be approximately 564 µA.

To obtain the dynamic current consumption value, calculate the slope of the line formed by the two points (frequency and current).

The slope is calculated as follows:

Slope = ((1645 – 564)/(26 – 6.5)) = 55.44 µA/MHz

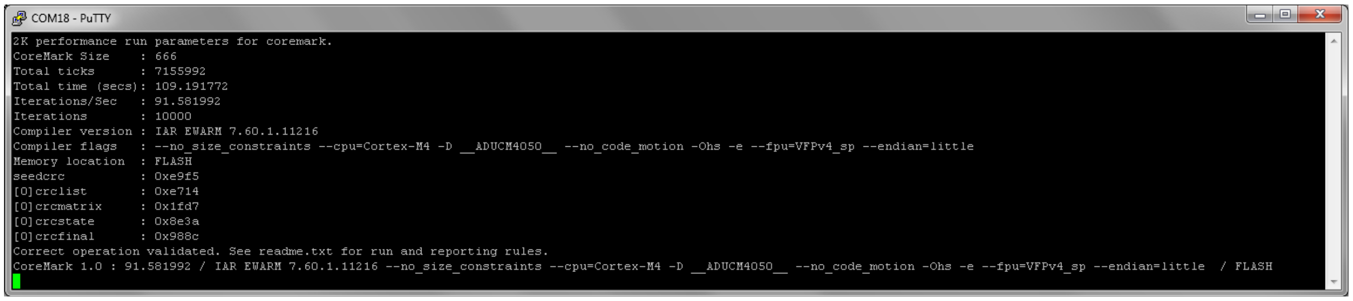
The dynamic current consumption is 55.44 µA/MHz.



```
2K performance run parameters for coremark.
CoreMark Size    : 666
Total ticks      : 7155220
Total time (secs): 109.179993
Iterations/Sec   : 91.591873
Iterations       : 10000
Compiler version : IAR EWARM 7.60.1.11216
Compiler flags   : --no_size_constraints --cpu=Cortex-M4 -D __ADUCM4050__ --no_code_motion -Ohs -e --fpu=VFPv4_sp --endian=little
Memory location  : FLASH
seedcrc          : 0xe9f5
[0]crclist       : 0xe714
[0]crcmatrix     : 0x1fd7
[0]crcstate      : 0x8e3a
[0]crcfinal      : 0x988c
Correct operation validated. See readme.txt for run and reporting rules.
CoreMark 1.0 : 91.591873 / IAR EWARM 7.60.1.11216 --no_size_constraints --cpu=Cortex-M4 -D __ADUCM4050__ --no_code_motion -Ohs -e --fpu=VFPv4_sp --endian=little  / FLASH
```

16075-023

Figure 23. Terminal Results at 26 MHz



```
2K performance run parameters for coremark.
CoreMark Size    : 666
Total ticks      : 7155992
Total time (secs): 109.191772
Iterations/Sec   : 91.581992
Iterations       : 10000
Compiler version : IAR EWARM 7.60.1.11216
Compiler flags   : --no_size_constraints --cpu=Cortex-M4 -D __ADUCM4050__ --no_code_motion -Ohs -e --fpu=VFPv4_sp --endian=little
Memory location  : FLASH
seedcrc          : 0xe9f5
[0]crclist       : 0xe714
[0]crcmatrix     : 0x1fd7
[0]crcstate      : 0x8e3a
[0]crcfinal      : 0x988c
Correct operation validated. See readme.txt for run and reporting rules.
CoreMark 1.0 : 91.581992 / IAR EWARM 7.60.1.11216 --no_size_constraints --cpu=Cortex-M4 -D __ADUCM4050__ --no_code_motion -Ohs -e --fpu=VFPv4_sp --endian=little  / FLASH
```

16075-024

Figure 24. Results on UART at 26 MHz

Results at 52 MHz

Enabling the PLL, the ADuCM4050 processors runs at 52 MHz. To enable the device, uncomment PLL_52MHz define in the **core_portme.h** file. Then, follow the steps explained in the CoreMark Results section to run the CoreMark score.

$$CoreMark/MHz = 183.87/52\ \text{MHz}$$

The CoreMark/MHz score is 3.54.

Power Measurements at 52 MHz

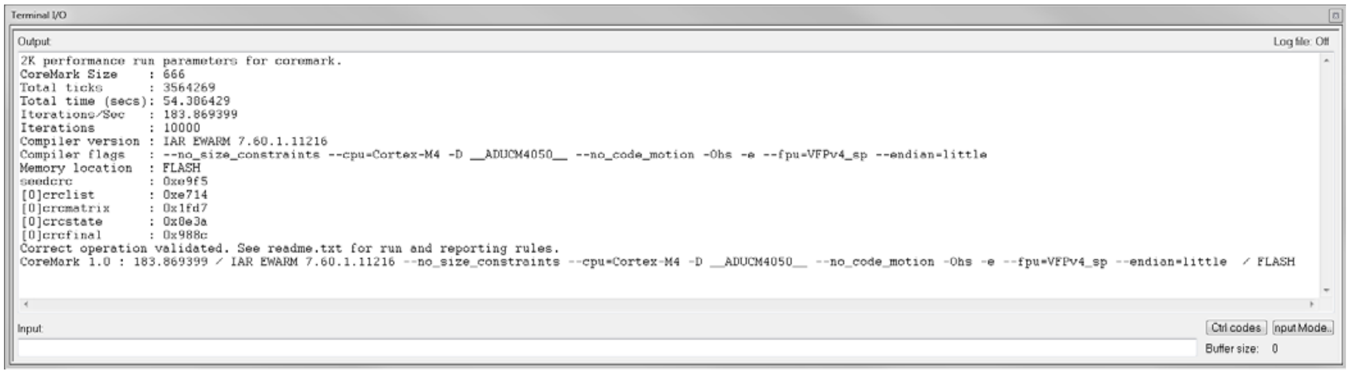
The following steps describe how to monitor the current consumption of the ADuCM4050 processor when the device executes the CoreMark code:

1. Ensure that UART_PRINT definition in the **core_portme.h** file is commented so that the UART pins are not floating.
2. Load the code onto the microcontroller.
3. Connect the positive terminal of the source to the JP5 connector.
4. Connect the ground of the source to the ground of the board.
5. Remove all the jumpers.
6. Press the **Reset** button.
7. Monitor the current consumption on the meter. The current consumption must be approximately 3094 µA when the processor executes the code at 52 MHz.
8. For dynamic current consumption, repeat the procedure with a different frequency. Comment the PLL_52MHz definition and uncomment the PLL_26MHz definition for a frequency of 26 MHz.
9. Monitor the current consumption on the meter. The current consumption must be approximately 1663 µA.

To obtain the dynamic current consumption value, calculate the slope of the line formed by the two points (frequency and current).

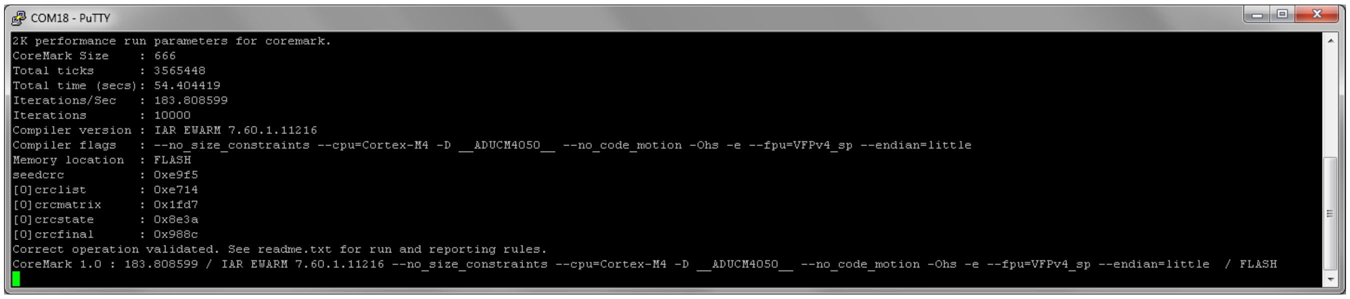
The slope is calculated as follows:

$$Slope = ((3094 - 1663)/(52 - 26)) = 55.04\ \mu\text{A/MHz}$$



```
2K performance run parameters for coremark.
CoreMark Size     : 666
Total ticks       : 3564269
Total time (secs): 54.386429
Iterations/Sec    : 183.869399
Iterations        : 10000
Compiler version : IAR EWARM 7.60.1.11216
Compiler flags    : --no_size_constraints --cpu=Cortex-M4 -D __ADUCM4050__ --no_code_motion -Ohs -e --fpu=VFPv4_sp --endian=little
Memory location   : FLASH
seedcrc           : 0xe9f5
[0]crclist        : 0xe714
[0]crcmatrix      : 0x1fd7
[0]crcstate       : 0x8e3a
[0]crcfinal       : 0x988c
Correct operation validated. See readme.txt for run and reporting rules.
CoreMark 1.0 : 183.869399 / IAR EWARM 7.60.1.11216 --no_size_constraints --cpu=Cortex-M4 -D __ADUCM4050__ --no_code_motion -Ohs -e --fpu=VFPv4_sp --endian=little  / FLASH
```

Figure 25. Terminal Results at 52 MHz



```
2K performance run parameters for coremark.
CoreMark Size     : 666
Total ticks       : 3565448
Total time (secs): 54.404419
Iterations/Sec    : 183.808599
Iterations        : 10000
Compiler version : IAR EWARM 7.60.1.11216
Compiler flags    : --no_size_constraints --cpu=Cortex-M4 -D __ADUCM4050__ --no_code_motion -Ohs -e --fpu=VFPv4_sp --endian=little
Memory location   : FLASH
seedcrc           : 0xe9f5
[0]crclist        : 0xe714
[0]crcmatrix      : 0x1fd7
[0]crcstate       : 0x8e3a
[0]crcfinal       : 0x988c
Correct operation validated. See readme.txt for run and reporting rules.
CoreMark 1.0 : 183.808599 / IAR EWARM 7.60.1.11216 --no_size_constraints --cpu=Cortex-M4 -D __ADUCM4050__ --no_code_motion -Ohs -e --fpu=VFPv4_sp --endian=little  / FLASH
```

Figure 26. Results on UART at 52 MHz

ULPMark-CP

THE EnergyMonitor

The EEMBC ULPMark EnergyMonitor™ software is an accurate tool for measuring energy. The EEMBC EnergyMonitor hardware (shown in Figure 27) is required to measure ULPMark-CP scores. This hardware can be purchased from the EEMBC website.

Figure 27 shows the EEMBC EnergyMonitor hardware, and the VCC and GND pins used to power the EV-COG-AD4050LZ or the EV-COG-AD4050WZ board.

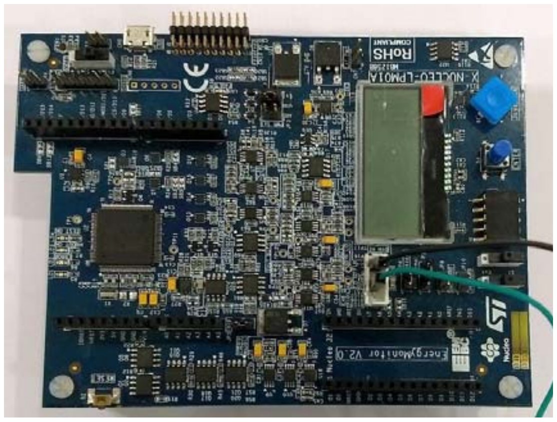


Figure 27. EEMBC EnergyMonitor Hardware

Installing the EnergyMonitor Software Drivers

The first time the EnergyMonitor hardware is connected to a PC, a USB driver message appears because the hardware is an unrecognized USB device.

When the USB driver message appears, click **Next**, and then click **Manually locate USB drivers**.

If the driver message does not appear, go to the **Device Manager** and locate the **EEMBC Application UART1** and **EEMBC Energy Tool V1** devices to install the driver on each of them.

Install the USB drivers, which are located at **/bin/USB_CDC/monitor_driver.inf** and **/bin/USB_CDC/monitor_driver.cat**. A security warning will then appear indicating that the publisher cannot be verified. Click **Install this driver software anyway**.

By default, 64-bit versions of Windows® Vista and later versions of Windows load a kernel mode driver only if the kernel can verify the driver signature. If using one of these versions of Windows, and the drivers cannot be installed, use the appropriate mechanisms to temporarily disable load time enforcement of a valid driver signature (the appropriate mechanism depends on the Windows version).

BUILDING THE ULPMark-CP APPLICATION CODE

Building the ULPMark-CP Application

To build or compile the application code, click **Project**, and click **Rebuild All**.

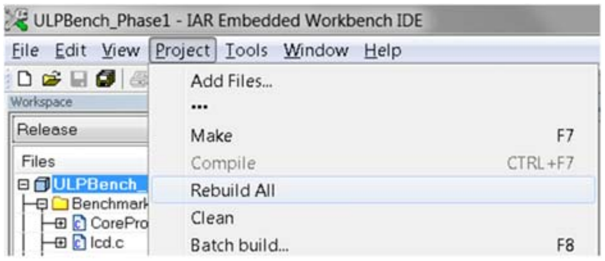


Figure 28. Build or Compile the Project

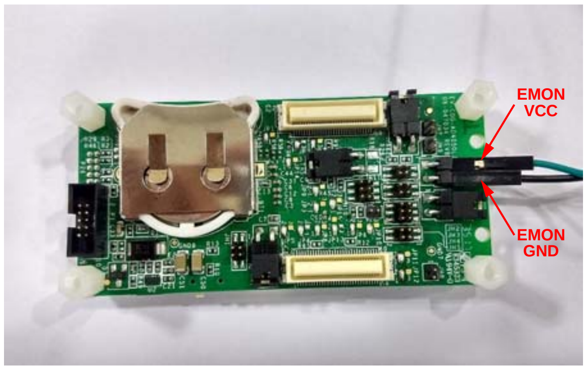


Figure 29. Board Setup for Downloading the Code

Downloading the ULPMark-CP Code

To download the code, click **Project** > **Download** > **Download** active application.

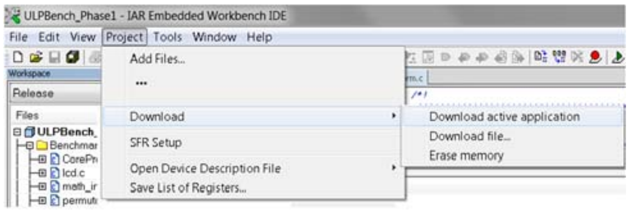

ULPBench_Phase1 - IAR Embedded Workbench IDE
File Edit View Project Tools Window Help
Workspace
Release
Files
ULPBench
Benchmar
CorePr
lcd.c
math_ir
permut
Add Files...
...
Download
SFR Setup
Open Device Description File
Save List of Registers...
Download active application
Download file...
Erase memory

Figure 30. Downloading the Application

After power cycling the device, the code runs on the ADuCM4050 device.

RUNNING THE ULPMark-CP BENCHMARK

Running the ULPMark-CP

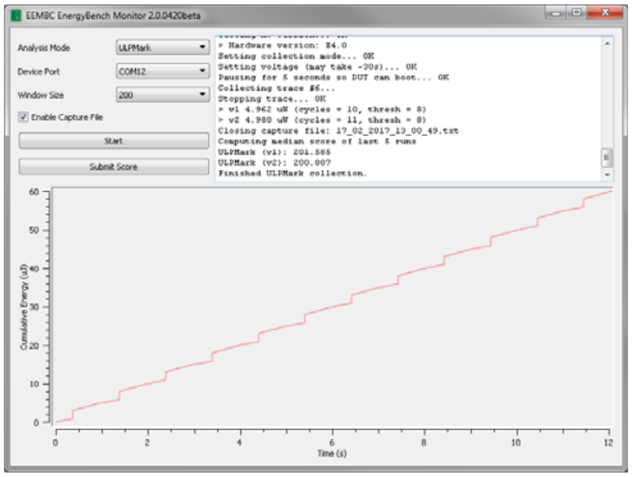
This section provides step by step information on how to set up the EV-COG-AD4050LZ or the EV-COG-AD4050WZ board for measuring the ULPMark-CP score.

1. Remove the USB cable.
2. Remove all the jumpers (J7, J8, J9, and so on) except TH1, TH2, TH4 and TH5. These mostly connect the external components of the board (LEDs, switches, sensors, battery, and so on).
3. Place the VBAT and GND cable of EMON to the TH3 test point, as shown in Figure 32.

The connection as shown in Figure 32 is now established.

Proceed to measure the score by starting the EnergyMonitor software and clicking **Start**. The EnergyMonitor hardware powers the EV-COG-AD4050LZ or the EV-COG-AD4050WZ board and measures the energy consumption of the core profile. At the end of the run, the software calculates the EEMBC ULPMark-CP score and displays the score on screen. The software also displays the average energy consumed for previous cycles in the history window.

The score obtained for typical devices at 52 MHz is around 189 EEMarks™-CP. This value may vary depending on process and temperature conditions. Figure 31 shows an example of a score for a typical device.



```
> Hardware version: E4.0
Setting collection mode... OK
Setting voltage (may take ~30s)... OK
Pausing for 5 seconds so DUT can boot... OK
Collecting trace #6...
Stopping trace... OK
> v1 4.962 uW (cycles = 10, thresh = 8)
> v2 4.980 uW (cycles = 11, thresh = 8)
Closing capture file: 17_02_2017_13_00_49.txt
Computing median score of last 5 runs
ULPMark (v1): 201.585
ULPMark (v2): 200.807
Finished ULPMark collection.
```

Figure 31. ULPMark-CP Score

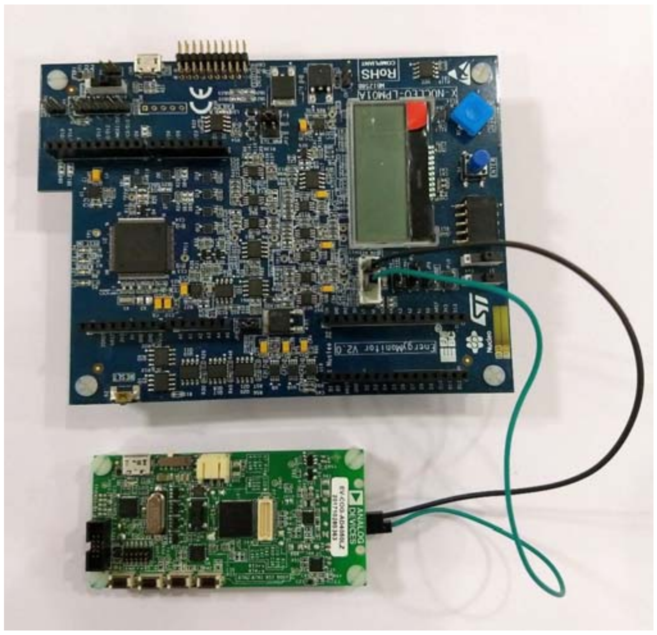


Figure 32. Board Setup for Measuring the Score

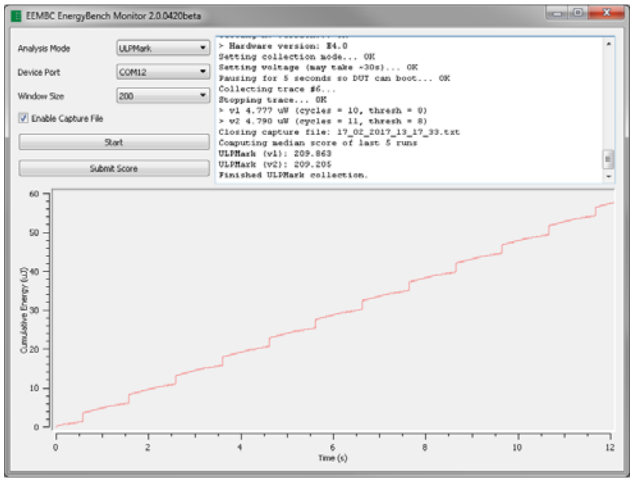
Running the ULPMark-Crystalless Profile

If an application does not require an accuracy as high as the one provided by a crystal, a low frequency oscillator can be used as the source clock of the real-time clock to reduce the energy consumption. Both the low frequency oscillator and crystal frequencies are 32 kHz.

The distributed code includes a **define** directive (Line 51 of the **Platform.c** file) to allow testing of the ULPMark-Crystalless Profile. Uncomment the **define USE_LFOSC** line to use the low frequency oscillator as the real-time clock:

```
#define USE_LFOSC
```

The score obtained for typical devices at 52 MHz is around 189 EEMarks-CP. This value may vary depending on process and temperature conditions. Figure 33 shows an example of a score for a typical device.


EEMBC EnergyBench Monitor 2.0.0420beta

Analysis Mode: ULPMark
Device Port: COM12
Window Size: 200
[✓] Enable Capture File

```
> Hardware version: E4.0
Setting collection mode... OK
Setting voltage (may take ~30s)... OK
Pausing for 5 seconds so DUT can boot... OK
Collecting trace #6...
Stopping trace... OK
> v1 4.777 uW (cycles = 10, thresh = 0)
> v2 4.790 uW (cycles = 11, thresh = 8)
Closing capture file: 17_02_2017_13_17_33.txt
Computing median score of last 5 runs
ULPMark (v1): 209.863
ULPMark (v2): 209.205
Finished ULPMark collection.
```

Figure 33. ULP Crystalless Profile Score

RESULTS ANALYSIS

The ULPMark-CP uses a formula that takes the reciprocal of the energy values (median of 5 times the average energy per second for 10 ULPMark-CP cycles).

$$\mathit{Energy}\ (\mu\text{J}) = 1000/\mathit{EEMarkCP}$$

The consumed energy is obtained as the sum of the energy consumed while the device is executing the workload (in active mode) and while the device is in hibernate.

$$\mathit{Energy} = \mathit{Active\ Energy} + \mathit{Sleep\ Energy}$$

According to the ADuCM4050 data sheet, the typical value for an active current at 52 MHz with cache disabled (similar to the ULPMark-CP behavior) is 3.21 mA, and for a hibernate current, 783 nA with low frequency crystal and real-time clock enabled. The active time duration is 217 µs.

$$\mathit{Energy} = \mathit{Voltage} \times \mathit{Current} \times \mathit{Time}$$

$$\mathit{Active\ Energy} = 3\ \text{V} \times 3210\ \mu\text{A} \times 0.217\ \text{ms} = 2.09\ \mu\text{J}$$

$$\mathit{Sleep\ Energy} = 3\ \text{V} \times 0.783\ \mu\text{A} \times 999.783\ \text{ms} = 2.34\ \mu\text{J}$$

According to the data sheet numbers and the execution time, the energy for the active current is 2.04 µJ, and the energy consumed during the sleep time is 2.34 µJ.

The score according to those values approaches the ones measured with the EEMBC EnergyMonitor software. The difference is due to the energy lost on the wake-up process.

$$\mathit{Energy}\ (\mu\text{J}) = 2.09 + 2.34 = 4.43\ \mu\text{J} \approx (1000/201) = 4.97\ \mu\text{J}$$

NOTES