ANALOG DEVICES

ADuCM3027/ADuCM3029 EEMBC Manual

UG-944

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 Scores for the ADuCM3027/ADuCM3029

INTRODUCTION

This reference manual describes how to reproduce the Embedded Microprocessor Benchmark Consortium (EEMBC) ULPBench™ Core Profile score and the CoreMark® score for the ADuCM3027/ ADuCM3029 microcontrollers.

This reference manual describes the steps necessary to install the software and to set up the hardware for measuring both scores.

Working through this document brings the user to a stage where they are able to reproduce the EEMBC ULPBench Core Profile score and the CoreMark score in the ADuCM3029 EZ-KIT® board.

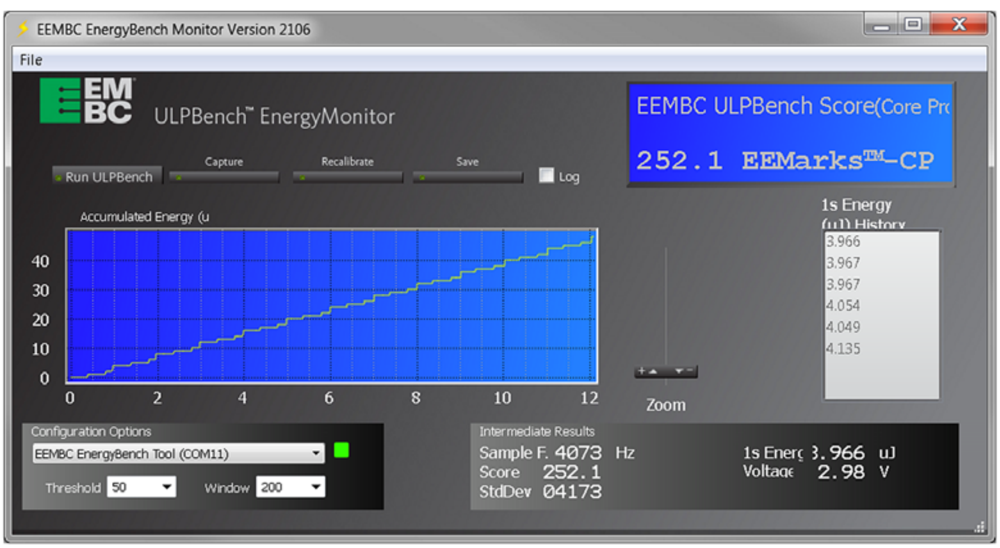
This reference manual provides details of the energy consumed by the ADuCM3027/ADuCM3029 microcontroller in the different power modes used on the benchmark, confirming the data sheet power numbers.

ABOUT THE ADUCM3027/ADUCM3029

The ADuCM3027/ADuCM3029 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™-M3 processor, a collection of digital peripherals, cache embedded SRAM and flash memory, and an analog subsystem, which provides clocking, reset, and power management capabilities along with the analog-to-digital converter (ADC).

The ADuCM3027/ADuCM3029 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 ADuCM3027/ADuCM3029 are available in the product data sheet. Consult the data sheet in conjunction with this reference manual when working with the EZ-KIT.



14302-001

Figure 1. EEMBC ULPBench Score

TABLE OF CONTENTS

REVISION HISTORY

3/2017—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, RTOS, 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 ADuCM3027/ ADuCM3029 processor.

COREMARK

To select an MCU for a particular application, the user needs to know if the MCU has enough processing power to meet the requirement. Several benchmarking options are available. Dhrystone is the most widely used benchmarking option; however, it has a few inherit problems such as having library calls within the timed portion and being susceptible to the ability of a compiler to optimize work, among others. To address these problems and to provide a simple, open source benchmark, EEMBC created the CoreMark.

CoreMark is a benchmark that aims to measure the performance of central processing units (CPU) used in embedded systems. It was developed in 2009 at EEMBC and is intended to become 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 (determine if an input stream contains valid numbers)
- CRC

ULPBENCH

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 can also imply great energy efficiency, whereby the most work is performed in a limited time period. Overall, the application requires a combination of tradeoffs on all of 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 ULPBench standardizes on data sheet parameters and provides a methodology to reliably and equitably measure MCU energy efficiency.

The foundations of ULPBench are as follows:

- Comparability, to make it easy to compare devices.
- Transparency, to make all measurements and the setup process transparent.
- Reproducibility, to make it easy to for anyone 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 provides the board support package (BSP) for the ADuCM3027/ADuCM3029 for the IAR Workbench.

Support for the ADuCM3027/ADuCM3029 is provided within the BSP.

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

The IAR Embedded Workbench can be downloaded from the IAR website.

IAR PROJECT CONFIGURATION

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

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

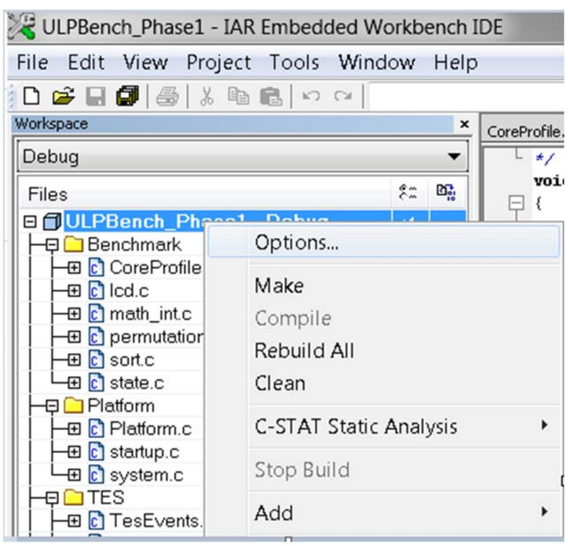


Figure 2. Project Options

2. Under **General Options**, ensure that **AnalogDevices ADUCM3027** or **AnalogDevices ADUCM3029** is selected as the target, depending on the microcontroller used.

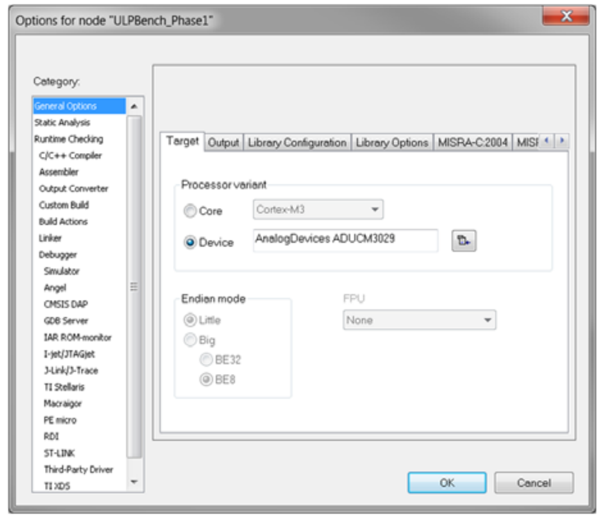


*Figure 3. **General Options**—Target Configuration*

3. Under **C/C++ Compiler**, ensure that the optimization for high speed is chosen (see Figure 4). Some functions, such as **pltInitialize**, are protected to ensure that they are not optimized. The following code protects a function and prevents the compiler from modifying the function code:

```
#pragma optimize=none
```

Figure 5 shows the included directories path and the **define** settings necessary for a proper compilation of the ULPBench Core Profile project.

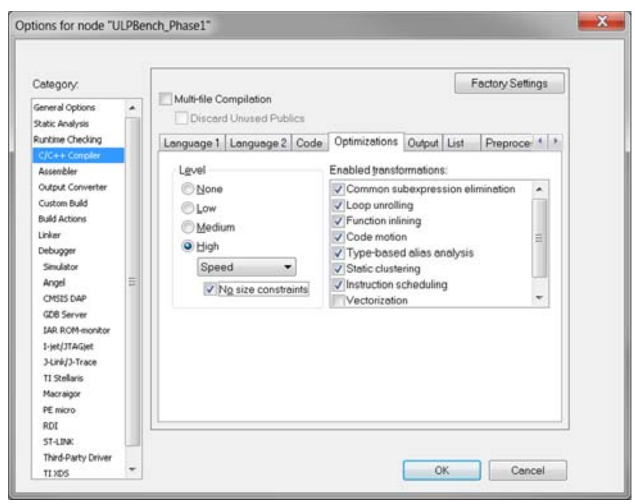


*Figure 4. **C/C++ Compiler**—Optimizations Configuration*

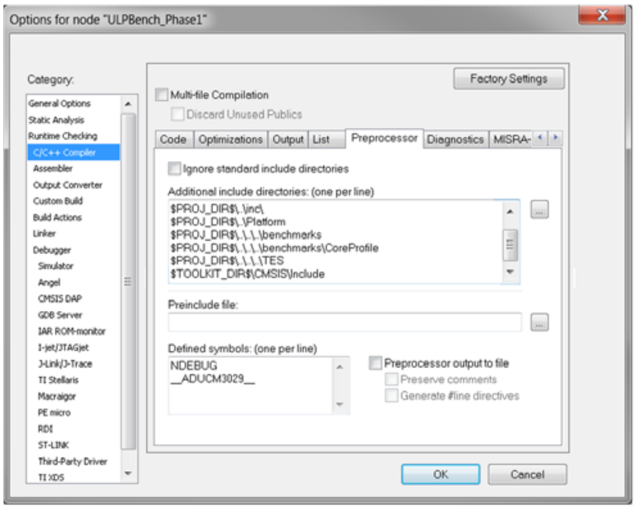


Figure 5. ***C/C++ Compiler****—Preprocessor ULPBench Configuration*

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

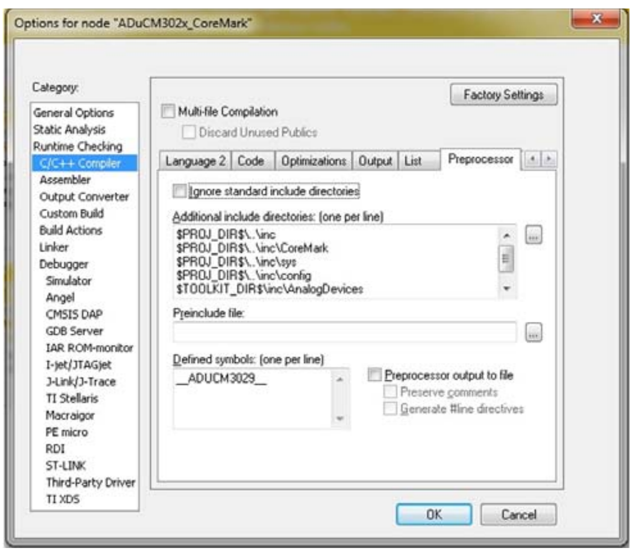


Figure 6. ***C/C++ Compiler****—Preprocessor CoreMark Configuration*

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. Therefore, it is necessary to configure the checksum as shown in Figure 7 and Figure 8 (both configurations can be found under **Linker**).

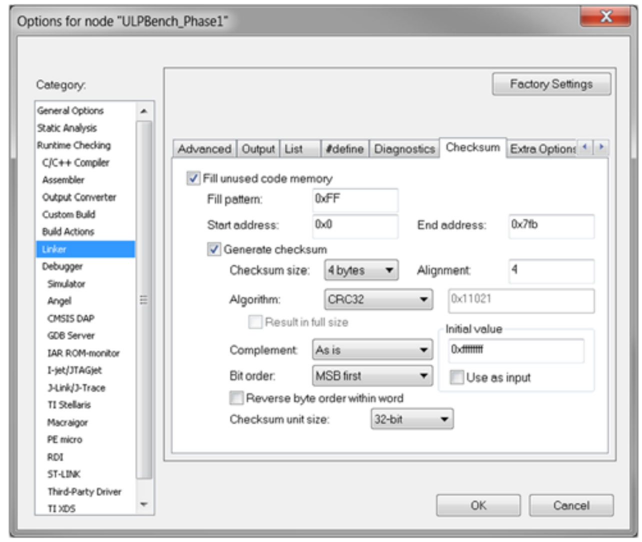


Figure 7. ***Linker****—Checksum Configuration*

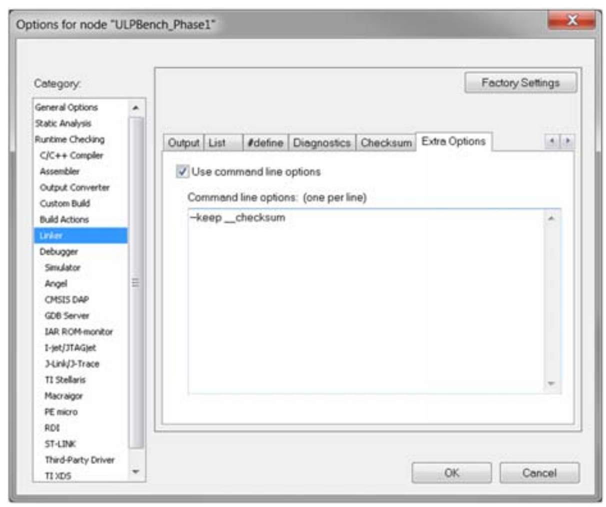


Figure 8. ***Linker****—Extra Options Configuration*

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

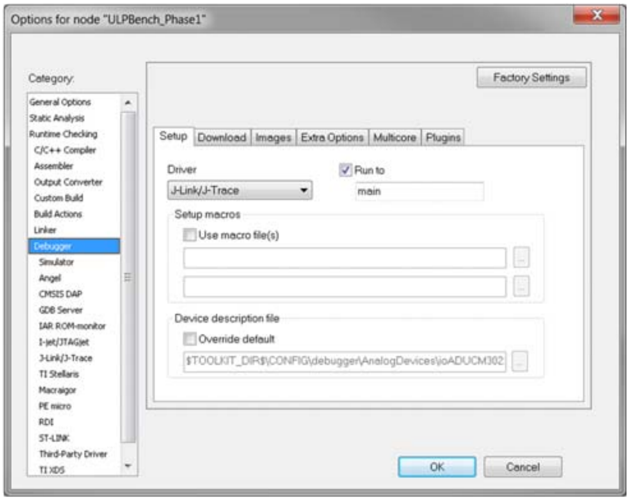


*Figure 9. **Debugger**—Setup Configuration*

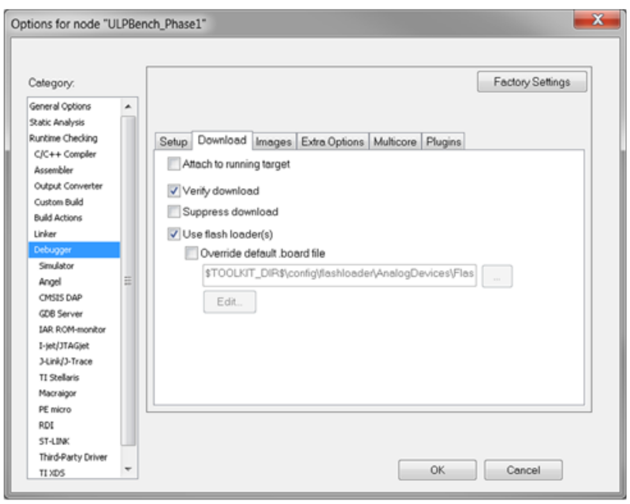


*Figure 10. **Debugger**—Download Configuration*

6. Figure 11 and Figure 12 show the **J-Link/J-Trace** configuration. Be sure to use the **Halt after bootloader** target reset strategy; otherwise, a kernel corruption is incurred, and the device locks down. To recover from device lockdown, the user must reflash the kernel.

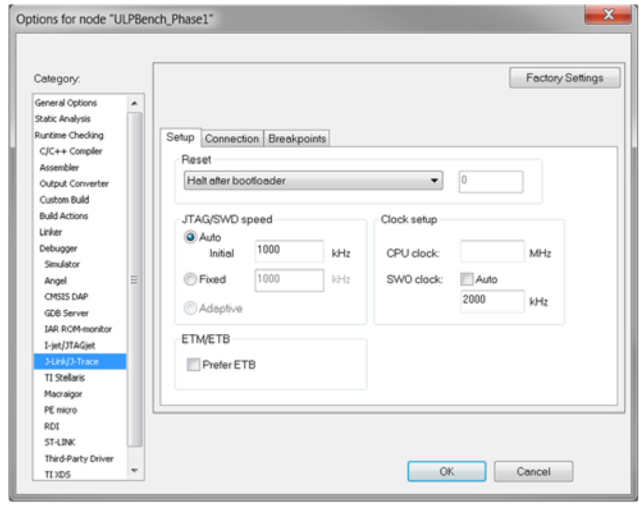


*Figure 11. **J-Link/J-Trace**—Setup Configuration*

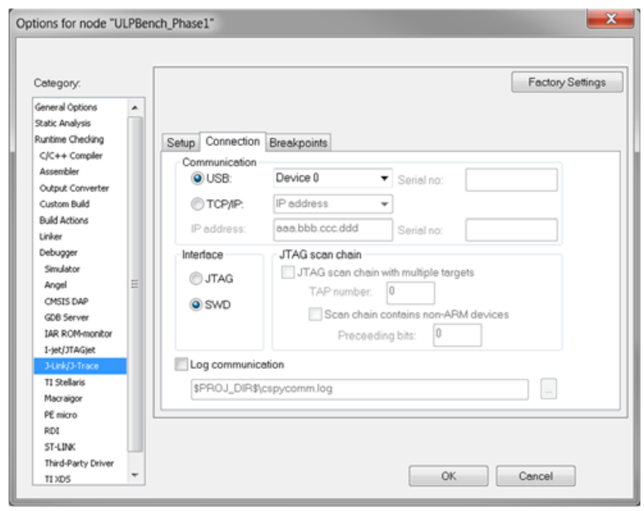


*Figure 12. **J-Link/J-Trace**—Connection Configuration*

COREMARK

LOADING COREMARK PROJECT

The project with the CoreMark source files and the core_portme.c and core_portme.h 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.

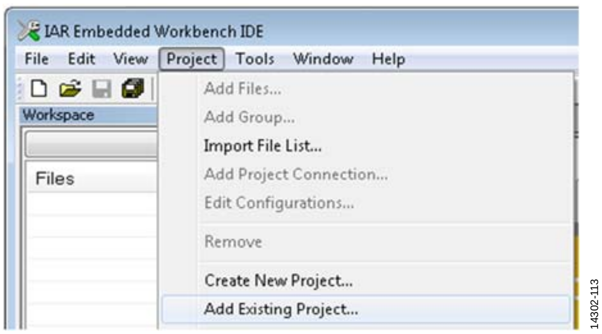


Figure 13. Adding an Existing Project

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

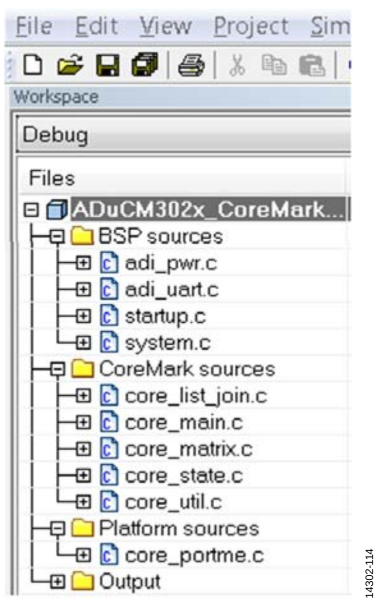


Figure 14. Project Files

The **BSP sources** folder includes the BSP 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 ADuCM3027/ADuCM3029 processor.

EEMBC does not restrict on changing the core_portme.c and core_portme.h files to suit the Analog Devices platform. The following are the differences between the core_portme.c file and the one given by the EEMBC:

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

The core_portme.h file has two defines:

- UART_PRINT define is used to print the result through the UART. If this is commented, the results are printed only on the terminal input/output.
- XTAL define is used to decide whether measuring the ticks using an external crystal oscillator or the internal RC oscillator. If this is commented, the internal oscillator is used; otherwise, the crystal oscillator is used.

RUNNING THE APPLICATION

Building the Application

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

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

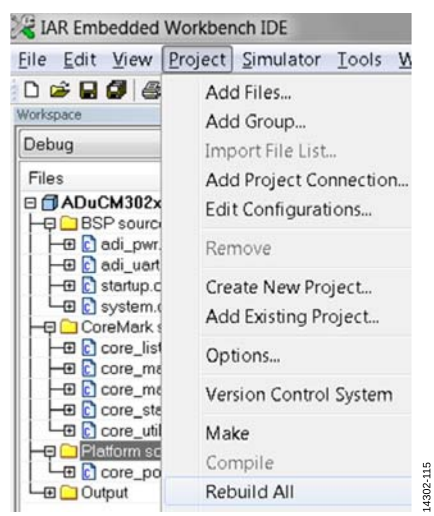


Figure 15. Build the Project

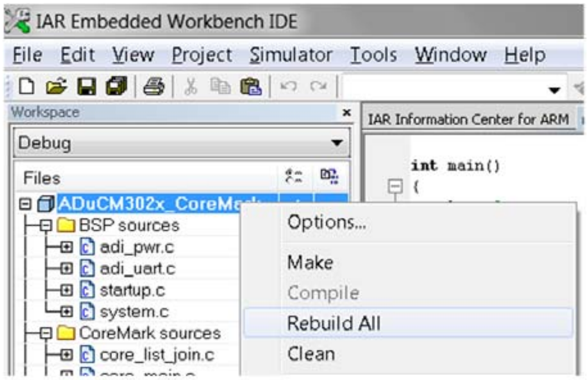


Figure 16. Building the Project

Downloading the Code

To load the code onto the EZ-KIT board, take one of the following steps:

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

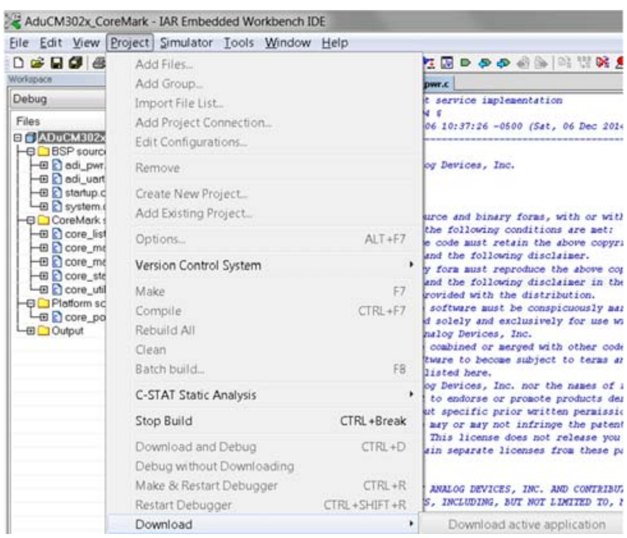


Figure 17. Downloading the Code

Figure 18. **Download and Debug** Button

Running the Project

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

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 set up 10,000 iterations, resulting approximately 2 minutes to complete the execution.

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

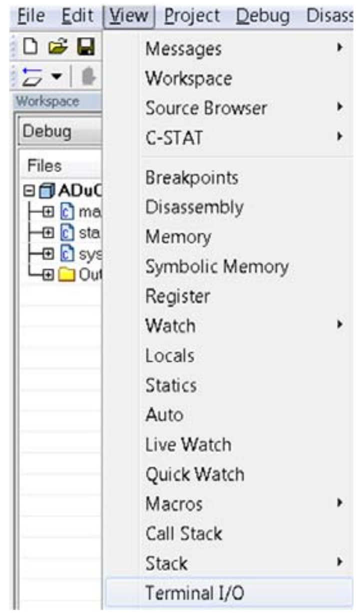


Figure 20. View **Terminal I/O**

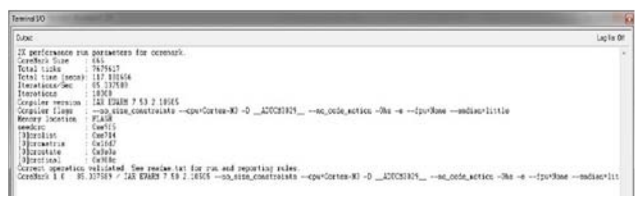


Figure 21. Terminal Results

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

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

1. Uncomment UART_PRINT define.
2. Connect the UART port of the EZ-KIT to the PC using a USB cable.

Figure 22. 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.
7. The other settings are as shown in Figure 23.

Figure 23. UART Configuration

Rebuild the project, following the instructions in the Running the 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.

$$CoreMark\ /\ MHz = \frac{CoreMark\ Score}{Clock\ Frequency}$$

In this project, the ADuCM3027/ADuCM3029 processors runs at 26 MHz.

$$CoreMark/\ MHz = \frac{85.337589}{26\ \mathrm{MHz}}$$

The CoreMark/MHz score is 3.2822.

This score is almost equal to the CoreMark/MHz score of the ARM Cortex-M3 processor, whose score is 3.32.

To report the score, CoreMark recommends the following format:

```
CoreMark/MHz 1.0: 3.2822 / IAR EWARM
7.50.2.10505 --no_size_constraints
--cpu=Cortex-M3 -D __ADUCM3029__
--no_code_motion -Ohs -e --fpu=None
--endian=little/FLASH
```

POWER MEASUREMENTS

The following steps describe how to monitor the current consumption of the ADuCM3027/ADuCM3029 processor when it 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 JP6 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 1165 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 425 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:

$$Slope = \frac{1165 - 425}{26 - 6.5} = 38\ \mu\mathrm{A/MHz}$$

The dynamic current consumption is 38 µA/MHz.

```
2K performance run parameters for coremark.
CoreMark Size    : 666
Total ticks      : 7712537
Total time (secs): 117.683975
Iterations/Sec   : 84.973336
Iterations       : 10000
Compiler version : IAR EWARM 7.50.2.10505
Compiler flags   : --no_size_constraints --cpu=Cortex-M3 -D __ADUCM3029__ --no_code_motion -Ohs -e --fpu=None --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 : 84.973336 / IAR EWARM 7.50.2.10505 --no_size_constraints --cpu=Cortex-M3 -D __ADUCM3029__ --no_code_motion -Ohs -e --fpu=None --endian=little
 / FLASH
```


Figure 24. Results on UART

ULPBENCH CORE PROFILE

THE ENERGYMONITOR

The EEMBC ULPBench EnergyMonitor™ software is an accurate tool for measuring energy. The EEMBC EnergyMonitor hardware (shown in Figure 25) is needed to measure ULPBench scores. This hardware can be purchased from the EEMBC website.

Figure 25 shows the EEMBC EnergyMonitor hardware, and the VCC and GND pins used to power the ADuCM3029 EZ-KIT board.

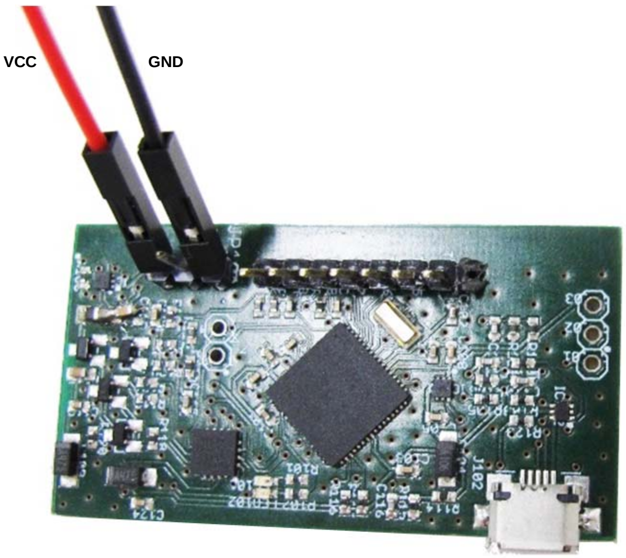


Figure 25. EEMBC EnergyMonitor Hardware

Installing the EnergyMonitor Software Drivers

The first time the EnergyMonitor hardware is connected to the PC, a USB driver message appears because it 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 devices named **EEMBC Application UART1** and **EEMBC Energy Tool V1** 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 appears 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 APPLICATION CODE

Building the Application

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

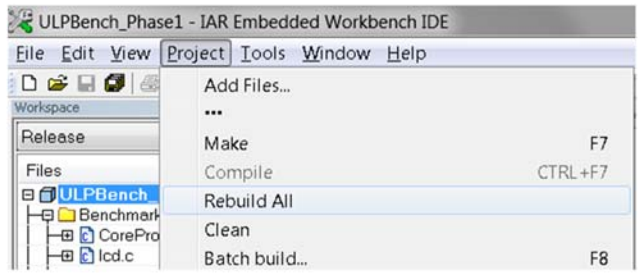


Figure 26. Build Project

Setting Up the Board to Download the Code

To download the code via USB, both JP6 jumpers must be installed.

If the EEMBC ULPBench_Phase1 project has been downloaded previously, the device is in hibernate mode for the majority of the time while in operation. In deep sleep mode (both hibernate and shutdown modes), the serial wire is disabled and it is not possible to download code.

The EEMBC ULPBench_Phase1 project ensures that the user can download code when the application is in flash by adding the following code:

```
while((pADI_GPIO0->IN & (1 << 4)) == (0 << 4));
```

At startup, P0.4 is configured as an input. To be able to download the code, it is necessary to clear the P0.4 pin, because the code is waiting in the **while** loop. To clear the P0.4 pin, shortcut P0.4 (Pin 10 of the Arduino header, J6, labelled as SCL on the silkscreen) and ground (Pin 7 of the Arduino header, J7, labelled as GND on the silkscreen) by placing a cable as shown in Figure 27.

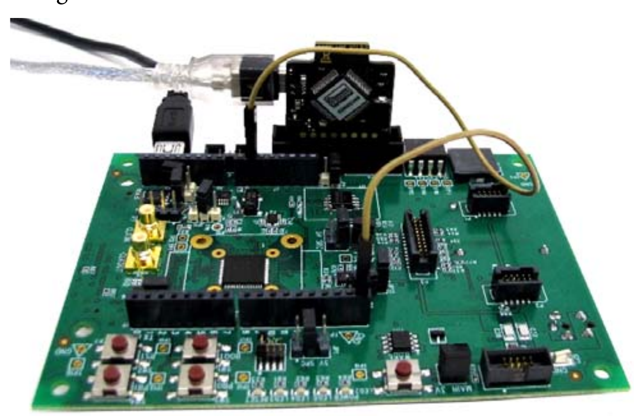


Figure 27. Board Setup for Downloading the Code

Downloading the Code

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

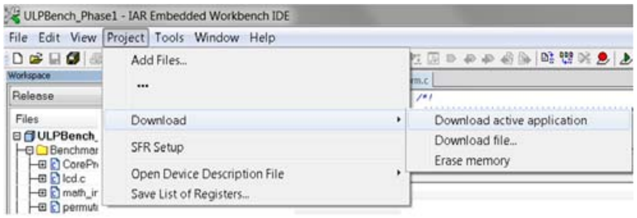


Figure 28. Downloading the Application

After power cycling the device, the code runs on the ADuCM3027/ADuCM3029 device.

RUNNING THE BENCHMARK

Running the ULPBench Core profile

This section provides a step by step account of how to set up the ADuCM3029 EZ-KIT board for measuring the ULPBench Core Profile score.

1. Remove the cable previously placed between P0.4 and GND (described in the previous section).
2. Place the cable previously removed between P0.4 and VBAT (Pin 4 of the Arduino header, J7, labelled as 3.3 V on the silkscreen). With this configuration, the code does not stop on the **while** instruction (see the Setting Up the Board to Download the Code section).
3. Remove the USB cable.
4. Remove the JTAG.
5. Remove the JP6 jumper between Pin 1 and Pin 3. This is the power measurement jumper.
   - Pin 1 powers the external components of the board (LEDs, switches, sensors, and accelerometers).
   - Pin 3 powers the microcontroller.

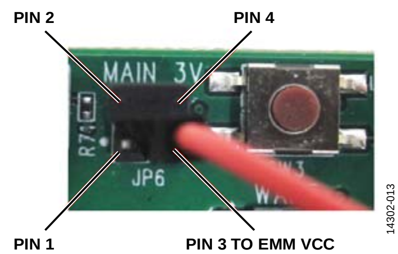


Figure 29. JP6 Connection

6. Connect the VBAT cable coming from the EnergyMonitor hardware to Pin 3 of JP6.
7. Connect the GND cable coming from the EnergyMonitor hardware to Pin 7 of J6.

The connection as shown in Figure 30 is now established.

Proceed to measure the score by starting the EnergyMonitor software and clicking **Run ULPBench**. The EnergyMonitor hardware powers the ADuCM3029 EZ-KIT board and measures the energy consumption of the core profile. At the end of the run, the software calculates the EEMBC ULPBench Core Profile score and displays it on screen. The software also displays the average energy consumed for previous cycles in the history window.

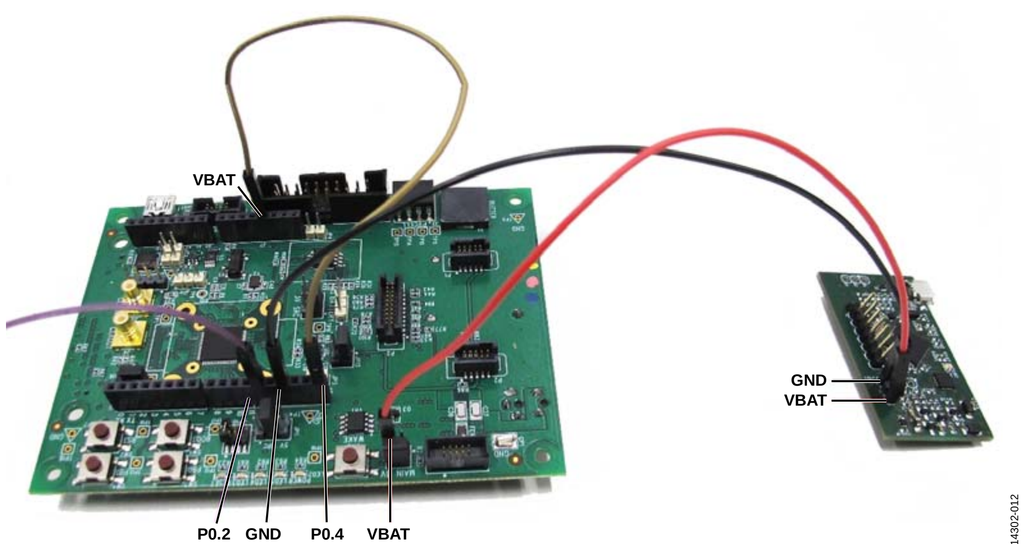
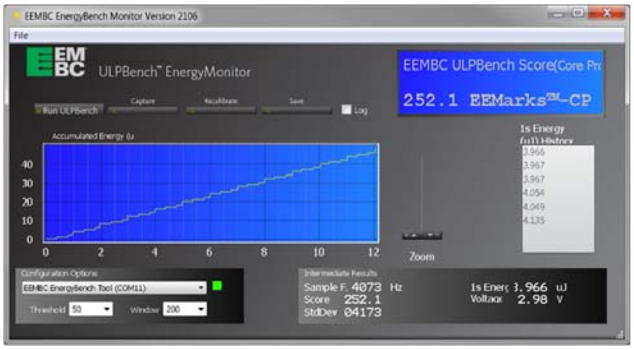


Figure 30. Board Setup for Measuring the Score

The score obtained for typical devices is around 250 EEMarks™-CP. This value may vary depending on process and temperature conditions. Figure 31 shows an example of a score for a typical device.



14302-014

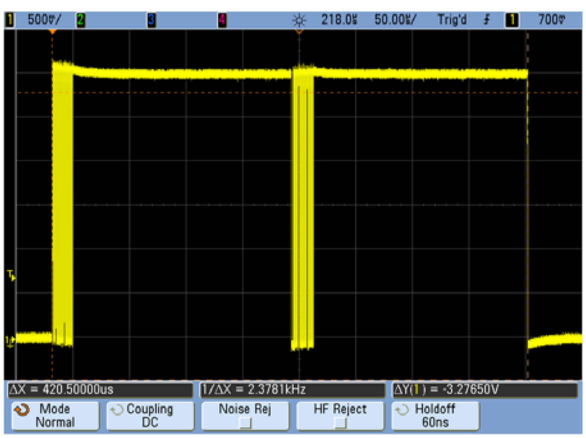
Figure 31. ULPBench Core Profile Score

Verifying the Correct Operation

To ensure that the workload is being executed correctly, a status pin has been defined. P0.2 (accessible on Pin 5 of J6, labelled as 12 on the silkscreen) is configured as the status pin in the certified code (purple cable shown in Figure 30).

According to the benchmark, the workload is executed twice, and at the beginning, the status pin toggles 20 times. When the second workload finishes, the status pin is cleared unless an error is flagged.

Figure 32 shows the P0.2 output while the workload is executed, as described previously.



14302-015

Figure 32. Verification of Correct Operation

Running the ULP Crystalless Profile

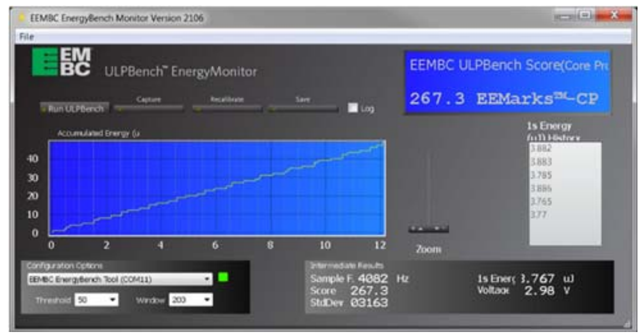
The Core Profile is the only score currently certified by the ULPBench group. However, an agreement for crystalless score is currently under discussion within the ULPBench working group.

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

The distributed code includes a **define** directive (Line 51 of the Platform.c file) to allow testing of the ULPBench Crystalless Profile. Uncomment the **define USE_LFOSC** line to use the LFOSC as the RTC clock:

```
#define USE_LFOSC
```

The score obtained for typical devices is around 265 EEMarks™-CP. This value may vary depending on process and temperature conditions. Figure 33 shows an example of a score for a typical device.



14302-016

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 ULPBench cycles).

$$Energy\ (\mu J) = \frac{1000}{EEMarkCP}$$

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.

$$Energy = Active\ energy + Sleep\ energy$$

According to the ADuCM3027/ADuCM3029 data sheet, the typical value for an active current is 38 μA/MHz, and for a hibernate current, 830 nA with LFXTAL and RTC enabled. Figure 32 shows that the active time duration is 420 μs.

$$Energy = Voltage \times Current \times Time$$

$$Active\ energy = 3\ V \times 1188\ \mu A \times 0.42\ ms = 1.49\ \mu J$$

$$Sleep\ energy = 3\ V \times 830\ nA \times 999.58\ ms = 2.49\ \mu J$$

According to the data sheet numbers and the execution time, the energy for the active current is 1.49 μJ, and the energy consumed during the sleep time is 2.49 μJ. The score according to those values matches the ones measured with the EEMBC EnergyMonitor software.

$$Energy\ (\mu J) = 1.49 + 2.49 = 3.98\ \mu J \cong \frac{1000}{252.1} = 3.96\ \mu J$$

NOTES

ESD Caution

ESD (electrostatic discharge) sensitive device. Charged devices and circuit boards can discharge without detection. Although this product features patented or proprietary protection circuitry, damage may occur on devices subjected to high energy ESD. Therefore, proper ESD precautions should be taken to avoid performance degradation or loss of functionality.

www.analog.com