RX100 Series

Running CoreMark on the RX100

R01AN1543EU0100
Rev.1.00
Mar 06, 2013

Introduction

This document details the steps needed to obtain, configure, and run the EEMBC CoreMark Benchmark. The RSK RX111 was the platform chosen for the examples throughout this document and results for this setup are shown. The IAR Embedded Workbench IDE is used along with the IAR RX Toolchain.

For more information on the EEMBC CoreMark Benchmark please follow the link below:

http://www.coremark.org/

Target Device

The RX111 is used as an example in this document but the same methods can be used to properly configure CoreMark for any MCU.

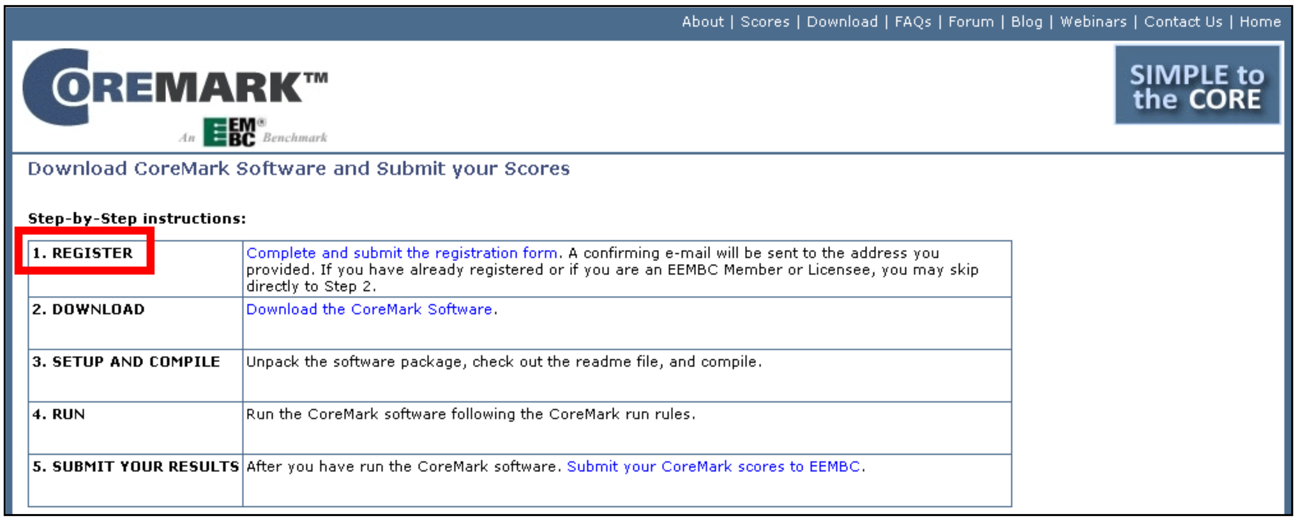
Contents

1. Overview

When looking for an MCU to fit a certain application users need to know if a MCU has enough processing power to meet their needs. There are a number of benchmarking options available with the most widely known probably being the Dhrystone. There are inherent problems with the Dhrystone though that are discussed in the ‘Background’ section of the ‘Coremark-requirements.doc’ document that comes packaged with the CoreMark Software. To address these problems and provide a “simple, open source benchmark” EEMBC created the CoreMark.

2. Obtaining the CoreMark Benchmark

The CoreMark Software is free to download from the CoreMark website. Start by going to http://www.coremark.org/download.


About | Scores | Download | FAQs | Forum | Blog | Webinars | Contact Us | Home
COREMARK™
An EEMBC Benchmark
SIMPLE to the CORE

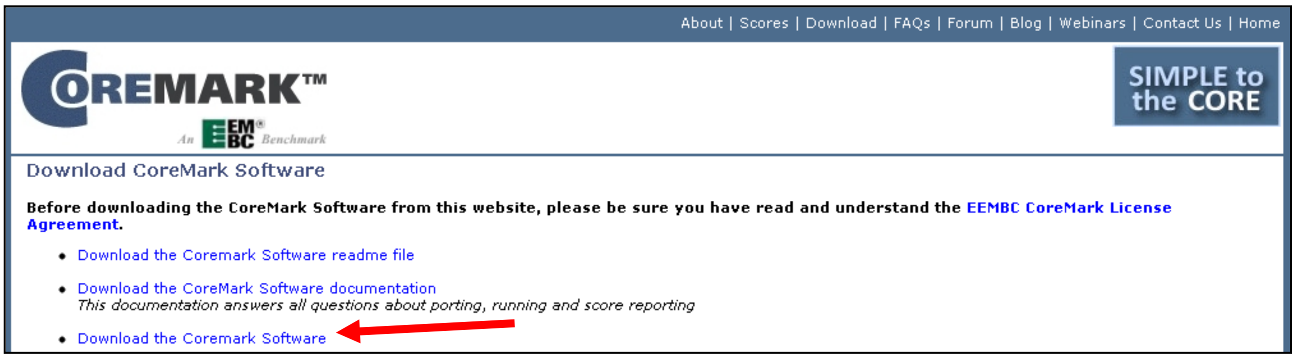
Download CoreMark Software and Submit your Scores

Step-by-Step instructions:

1. REGISTER	Complete and submit the registration form. A confirming e-mail will be sent to the address you provided. If you have already registered or if you are an EEMBC Member or Licensee, you may skip directly to Step 2.
2. DOWNLOAD	Download the CoreMark Software.
3. SETUP AND COMPILE	Unpack the software package, check out the readme file, and compile.
4. RUN	Run the CoreMark software following the CoreMark run rules.
5. SUBMIT YOUR RESULTS	After you have run the CoreMark software. Submit your CoreMark scores to EEMBC.

Figure 1 : Register & Download CoreMark

In order to download the software users first have to register with EEMBC. This can be done by following the instructions underneath the ‘REGISTER’ section on the previously referenced webpage. After the registration process is finished follow the instructions under the ‘DOWNLOAD’ section to get to the ‘Download CoreMark Software’ webpage. From this webpage download the CoreMark Software shown below in Figure 2. The CoreMark documentation is included with this download.


About | Scores | Download | FAQs | Forum | Blog | Webinars | Contact Us | Home
COREMARK™
An EEMBC Benchmark
SIMPLE to the CORE

Download CoreMark Software

Before downloading the CoreMark Software from this website, please be sure you have read and understand the EEMBC CoreMark License Agreement.

- Download the Coremark Software readme file
- Download the CoreMark Software documentation
  This documentation answers all questions about porting, running and score reporting
- Download the Coremark Software

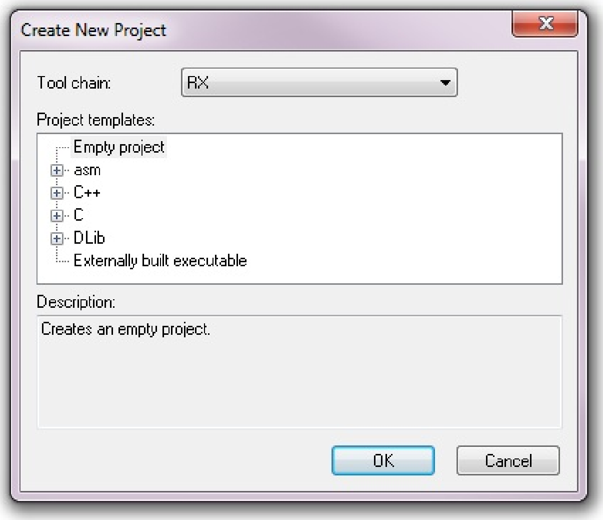
Figure 2 : Download CoreMark Software & Documentation

3. Creating a CoreMark IAR Project

This section assumes that the IAR IDE is already installed. If it is not installed, please go the IAR website and download and install the latest IAR RX Toolchain. The RX100 is supported from v2.41 onward.

Now that we have the CoreMark source files and the IAR RX Toolchain we can create an IAR workspace. Start off by opening Embedded Workbench and creating a new folder in Windows for the workspace.

Now, in Embedded Workbench, create a new project from the Project menu. Choose the RX Toolchain and Use the Empty Project template as shown below:



Tool chain: RX

Creates an empty project.

Figure 3 : Create New IAR project

After clicking OK, give the project a name and save it as a .ewp file in the workspace folder created earlier.

In Embedded Workbench, right click on the project and select options. Under Category, highlight General Options and then under the Target tab choose the RX111 group>>R5F1115. This is the MCU on the RX111 RSK.

Now under Category, highlight the Debugger and under the Setup tab change the Driver from Simulator to E1/E20 emulator and click OK.

4. Adding CoreMark Source

Once the IAR project has been setup the CoreMark source files can be added. In Windows, create a new folder in the workspace folder and name it CoreMark. Find the CoreMark Software package that was downloaded earlier and copy the following files to the CoreMark folder that was just created:

- core_list_join.c
- core_main.c
- core_matrix.c
- core_state.c
- core_util.c
- coremark.h
- simple/core_portme.c
- simple/core_portme.h

Files can be added to your IAR project by going to Project >> Add Files. After adding the files from the CoreMark folder into your project, you may wish to group them in the Embedded Workbench by right-clicking anywhere in the left hand pane and choosing Add>>Add Group and name it CoreMark. After creating the folder drag the CoreMark files into it.

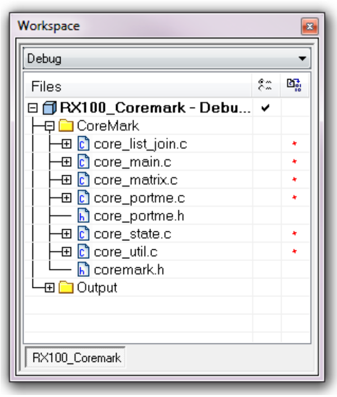

Workspace
Debug
Files
RX100_Coremark - Debu...
CoreMark
core_list_join.c
core_main.c
core_matrix.c
core_portme.c
core_portme.h
core_state.c
core_util.c
coremark.h
Output
RX100_Coremark

Figure 4 : Add & Group CoreMark Files

5. Adding Board Support Files

Along with the CoreMark Software the user will also need to add files to support their development system. This application note uses the RX111 RSK and the board support files can be found in the 'bsp' (Board Support Package) folder that was packaged with this application note. In Windows, copy the bsp folder to the workspace folder and then add them to your project in Embedded Workbench. Adding files to your project is done by clicking Project >> Add Files.

- *rx111_mtu.c & rx111_mtu.h*
  - Contains code to use the MTU timer peripheral. This is used for measuring performance.
- *serial_printf.c & serial_printf.h*
  - Replaces low-level functions to redirect printf() to a serial port. Also initializes SCI peripheral.
- *hardware_setup.c*
  - Sets up the clocks on the RX62N MCU. This replaces the default *hardware_setup.c* file that comes from the project generators. This means the user will first need to remove the previous file and then add the new one. The user could also just copy the new file over the old one.
- *low_level_init.c*
  - Reset processing function required by IAR
- *iorx111.h*
  - RX111 header file for IAR

Your project window should look like this now:

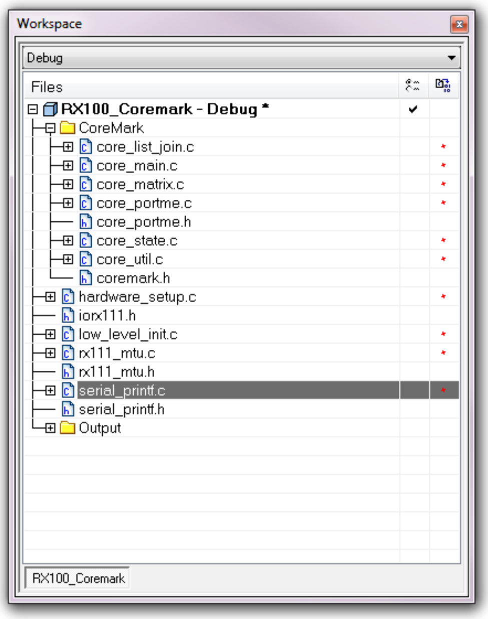

Workspace
Debug
Files
RX100_Coremark - Debug *
CoreMark
core_list_join.c
core_main.c
core_matrix.c
core_portme.c
core_portme.h
core_state.c
core_util.c
coremark.h
hardware_setup.c
iorx111.h
low_level_init.c
rx111_mtu.c
rx111_mtu.h
serial_printf.c
serial_printf.h
Output
RX100_Coremark

Figure 5 : Add bsp files

Save the workspace into your workspace folder in Windows by going to File>>Save Workspace and giving it a name.

6. Configuring CoreMark

After the CoreMark source has been added, it needs to be configured. This is done through the *core_portme.c* and *core_portme.h* files. These files are commented well and if more information is needed then the CoreMark documentation (part of the package downloaded from coremark.org) can be consulted.

6.1 System Definitions and Data Types

The following information details the settings that are used for the provided example project. Open the file *core_portme.h*. This file has a list of definitions that need to be changed to meet the requirements of the system being developed for. These definitions are listed below with the values that should be assigned to them for the RX111 example setup.

- HAS_FLOAT
    - Should be defined as '0' since the RX111 does not have an FPU. However, defining this as a 1 only results in better resolution when printing out the results and does not cause a significant difference in the final result.
- HAS_TIME_H
    - Should be defined as '0' since the time functions are not implemented
- USE_CLOCK
    - Should be defined as '0' since the time functions are not implemented
- HAS_STDIO
    - Should be defined as '1' since the IAR toolchain does support stdio.h
- HAS_PRINTF
    - Should be defined as '1' since the IAR toolchain does support printf()
- MAIN_HAS_NOARGC
    - Should be set to '1' since arguments to main() are not used

The data types defined in *core_portme.h* should be fine by default. The user will need to include *stddef.h* at the top of the file to make sure the *size_t* typedef is defined.

```
#include <stddef.h>
```

Since *time.h* is not supported the use of the *clock_t* type will need to be changed to *unsigned long*. This should be done in two places. In *core_portme.h* find the following code and make the changes shown below.

Find:	Change to:

`#include <time.h>`
`typedef clock_t CORE_TICKS;`
	`#if 0`
	`#include <time.h>`
	`typedef clock_t CORE_TICKS;`
	`#else`
	`typedef unsigned long CORE_TICKS;`
	`#endif`

In *core_portme.c* find the following code and the make the changes shown below.

Find:	Change to:
`#define CORETIMETYPE clock_t`	`#define CORETIMETYPE` **`unsigned long`**

The CLOCKS_PER_SEC definition will also be missing. To fix this make the changes shown below in *core_portme.c*.

Find:	Change to:

`#define NSECS_PER_SEC CLOCKS_PER_SEC`	**`/* Timer clock frequency: PCLK=32MHz */`**
	`#define NSECS_PER_SEC` **`32000000`**

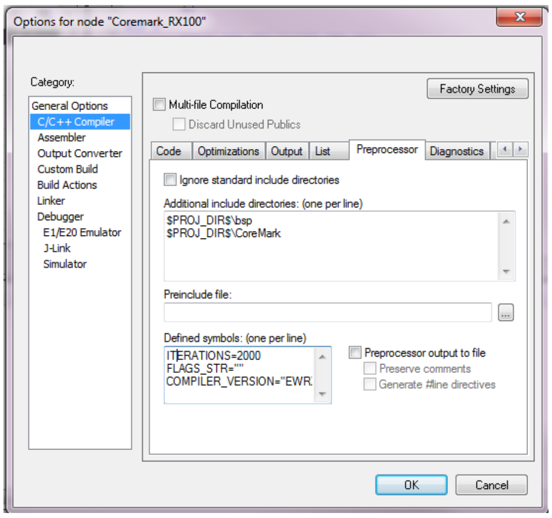
6.2 Run Configuration Definitions

There are two definitions that need to be defined that have to do with the CoreMark run. The first definition is ITERATIONS. This definition defines the number of CoreMark iterations that will be run. The second definition is FLAGS_STR which is set to a 0 length string. These definitions may be placed inside of *core_portme.h* or can be defined in the toolchain options. For this example the definitions will be put in the toolchain options.

We also need to configure the include paths for the project.

This is done by following these steps:

1. In Embedded Workbench, right click on the project and select options.
2. Under Category, highlight “C/C++ Compiler”.
3. Make sure the ‘Preprocessor’ tab is chosen.
4. In the Additional Include Directories box, enter $PROJ_DIR$\bsp and $PROJ_DIR$\CoreMark on separate lines.
5. In the Defined Symbols box, make the following entries on separate lines:
    a. ITERATIONS=2000
    b. FLAGS_STR=""
    c. COMPILER_VERSION="EWRX V.2.41.1". If you are using a newer version of the compiler, modify this string accordingly.
6. The entries should look like this:



☐ Multi-file Compilation
☐ Discard Unused Publics
☐ Ignore standard include directories

Additional include directories: (one per line)

```
$PROJ_DIR$\bsp
$PROJ_DIR$\CoreMark
```

Preinclude file:

Defined symbols: (one per line)

```
ITERATIONS=2000
FLAGS_STR=""
COMPILER_VERSION="EWR
```

☐ Preprocessor output to file
☐ Preserve comments
☐ Generate #line directives

Figure 6 : Configuration defintions

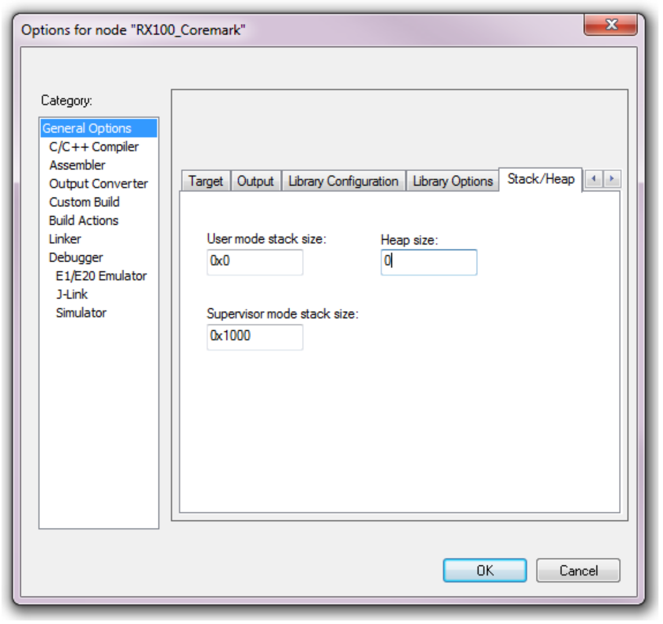
7. Click OK

6.3 Increasing Stack Size

The default memory allocation method used by CoreMark is to use the stack. Using the IAR RX project generators the number of bytes allocated for the user and interrupts stacks is 256 bytes by default. The size of the user stack needs to be increased to avoid stack overflow. Follow the steps below to increase the stack size.

1. In Embedded Workbench, right click on the project and select options.
2. Under Category, highlight “General Options”.
3. Make sure the “Stack/Heap” tab is chosen.
4. Set the Supervisor mode stack to 0x1000.

5. Set the Heap size to 0.
6. Click OK.



User mode stack size: 0x0
Heap size: 0
Supervisor mode stack size: 0x1000

Figure 7 : Increasing Stack Size

6.4 Getting Execution Cycles and Outputting Results

In order to get performance results from the CoreMark benchmark we must know how many cycles it took to execute the CoreMark program. This is done in *core_portme.c* using the start_time() and stop_time() functions. For this example the MTU timer on the RX111 was used to count cycles. Each MTU channel is 16-bits and two channels can be cascaded to make a 32-bit timer. If the MTU is run at the maximum speed of 32MHz this gives a maximum time of around 134 seconds. According to the CoreMark documentation the benchmark must be run for a minimum of 10 seconds which means there is no reason to divide the clock any further. The high resolution of the timer combined with a large number of CoreMark iterations makes the resulting cycle count very accurate. The functions to initialize, start, and stop the MTU are provided in *rx111_mtu.c* which is provided with this application note.

After the measured section of the CoreMark has finished the results will be calculated and output using the printf() function. For this example the output of printf() will be directed to SCI1 which is connected to the serial port on the RSK. The user redirects the output of printf() by writing their own putchar() and getchar() functions. These functions and a function to initialize the SCI peripheral are included with this application note in *serial_printf.c*.

Table 1 shows the changes that need to be made to *core_portme.c* to use the MTU timer and initialize the SCI peripheral.

Find:	Change to:

#include <stdlib.h>
#include "coremark.h"
	#include <stdlib.h>
	#include “rx111_mtu.h”
	#include “serial_printf.h”
	#include “coremark.h”

#define CORETIMETYPE unsigned long #define GETMYTIME(_t) (*_t=clock())	**#if 0** #define GETMYTIME(_t) (*_t=clock()) **#endif**

void start_time(void) {
   GETMYTIME(&start_time_val);
}
	void start_time(void) {
	   start_time_val = timer_start();
	}

void stop_time(void) {
   GETMYTIME(&stop_time_val);
}
	void stop_time(void) {
	   stop_time_val = timer_stop();
	}

void portable_init(core_portable *p, int *argc, char *argv[])
{
   ...
   p->portable_id=1;
}
	void portable_init(core_portable *p, int *argc, char *argv[])
	{
	    ...
	    p->portable_id=1;
	    timer_init();
	    sci_init();
	}

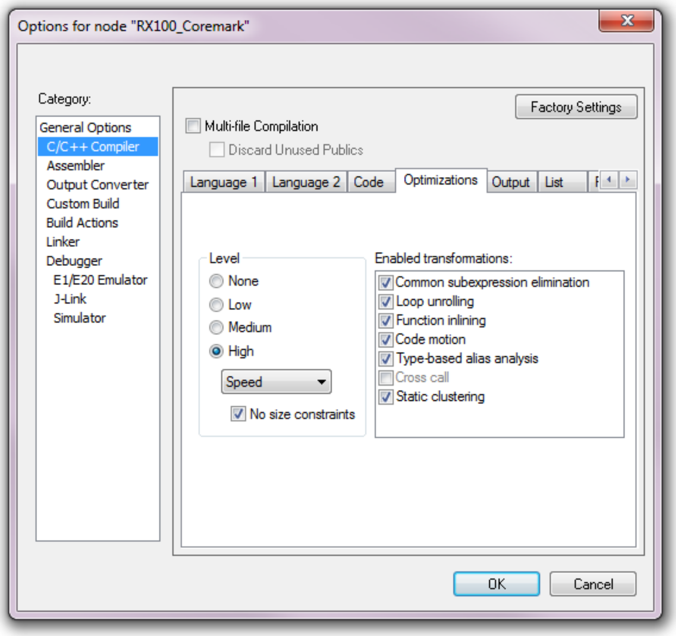
Table 1 : Changes to core_portme.c

7. Optimizing CoreMark

To obtain the best benchmark results optimizations should be turned on. This section will guide the user through using the same optimizations that were used to obtain the CoreMark number shown in Section 9.

7.1 Compiler Optimizations

1. In Embedded Workbench, right click on the project and select options.
2. Under Category, highlight “C/C++ Compiler”.
3. Make sure the “Code” tab is chosen.
4. Set Align functions to 4.
5. Select the Optimizations tab.
6. Set Level to High and change from Balanced to Speed.
7. Check the “No Size Constraints” option.
8. Click OK.



☐ 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
☐ Cross call
☑ Static clustering

Figure 8: Enable Optimizations for Compiler

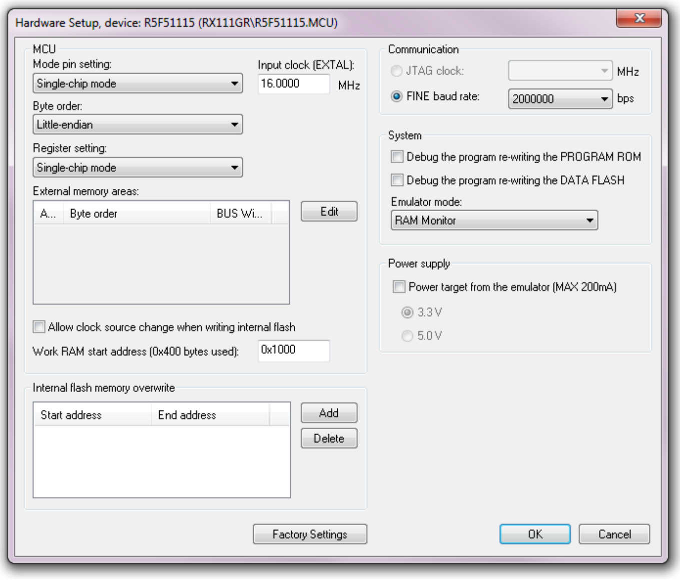
7.2 Linker Optimizations

1. In Embedded Workbench, right click on the project and select options.
2. Under Category, highlight “Linker”.
3. Make sure the “Optimizations” tab is chosen.
4. Check the “Inline small routines” option.

8. Run the CoreMark Benchmark

With the changes made from the previous sections go ahead and build the project by going to Project >> Make. After this process has finished, we can connect to the RSK board and run the code. Follow the steps below to do this.

1. Connect the E1 debugger to your PC using the supplied cable.
2. Connect the RSK to the E1.
3. Bring up the emulator setup window by going to E1/E20 Emulator>>Hardware Setup
    a. Change the Input clock to 16.0000 MHz
    b. If the RSK does not have external power connected, check the “Power target from emulator” option, otherwise uncheck the option. For details on configuring the RSK for internal or emulator power, refer to the RSK schematic.


Hardware Setup, device: R5F51115 (RX111GR\R5F51115.MCU)

MCU
Mode pin setting: Single-chip mode
Input clock (EXTAL): 16.0000 MHz
Byte order: Little-endian
Register setting: Single-chip mode
External memory areas:

A...	Byte order	BUS Wi...

Edit

☐ Allow clock source change when writing internal flash
Work RAM start address (0x400 bytes used): 0x1000

Internal flash memory overwrite

Start address	End address

Add
Delete

Communication
○ JTAG clock: MHz
● FINE baud rate: 2000000 bps

System
☐ Debug the program re-writing the PROGRAM ROM
☐ Debug the program re-writing the DATA FLASH
Emulator mode: RAM Monitor

Power supply
☐ Power target from the emulator (MAX 200mA)
● 3.3 V
○ 5.0 V

Factory Settings OK Cancel

Figure 9 : Configure E1 Debugging Session

   c. Click OK.

4. Build the project by pressing F7
5. Start debugging by pressing “Ctrl+D”.
6. Connect a serial cable between the RSK and your PC.
7. Open up a terminal program with the following settings:
   a. 115200 baud
   b. 8 data bits
   c. 1 stop bit
   d. No parity
   e. No flow control
8. Click Debug >> Go.
9. After approximately 20 seconds the results should show up in the terminal window. Example output is shown in Section 9.

9. Results for RX111

This section shows the results for the example project that was built throughout this document. The system used was a RSK RX111. These results will be the same for all RX100 devices since they all share the same core.

CoreMark Output

```
2K performance run parameters for coremark.
CoreMark Size    : 666
Total ticks      : 650799876
Total time (secs): 20.337496
Iterations/Sec   : 98.340520
Iterations       : 2000
Compiler version : EWRX V.2.41.1
Compiler flags   :
Memory location  : STACK
seedcrc          : 0xe9f5
[0]crclist       : 0xe714
[0]crcmatrix     : 0x1fd7
[0]crcstate      : 0x8e3a
[0]crcfinal      : 0x4983
Correct operation validated. See readme.txt for run and reporting rules.
CoreMark 1.0 : 98.340520 / EWRX V.2.41.1  / STACK
```

When discussing MCUs, many users are interested in seeing the CoreMark per megahertz of the MCU. The CoreMark number helps in showing raw horsepower while the CoreMark/MHz number helps show the efficiency of the core. If you have to run MCU-A at 100MHz and 80mA to match the processing power of MCU-B running at 50MHz and 40mA then most users will select MCU-B.

To calculate CoreMark/MHz the CoreMark number should be divided by the clock speed that was used when the benchmark was performed. For this example the RX111 was run at 32MHz.

CoreMark/MHz = CoreMark Score / Clock Speed

CoreMark/MHz = 98.340520 / 32MHz

CoreMark/MHz = 3.073

Note that you will get a slightly different result (Coremark/MHz = 3.125) if you have defined HAS_FLOAT in section 6.1as 0.This is because of lower resolution when printing the final result.

10. Summary

Detailed instructions were provided to aid users in replicating the CoreMark/MHz number that Renesas provides. As with any benchmark, these results should not be used as the sole reason to choose one MCU over another. Rather, the CoreMark should be used as a general indicator of a MCU’s core processing power. For example, if MCU-A has a CoreMark 95 and MCU-B has a CoreMark of 90 then the conclusion that can be drawn is that both MCUs have similar core capabilities and if one has enough power for you, then so should the other one. Using the same example, if a user says that MCU-A will work but MCU-B will not because it is not powerful enough then that means the margin for error is very small. In this case the user would be better off moving up to the next ‘class’ of MCUs with more processing power.

Website and Support

Renesas Electronics Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/inquiry

Revision Record

		Description	
Rev.	Date	Page	Summary
1.00	March.06.13	—	First edition issued

General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Handling of Unused Pins

   Handle unused pins in accord with the directions given under Handling of Unused Pins in the manual.

   — The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.
2. Processing at Power-on

   The state of the product is undefined at the moment when power is supplied.

   — The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
   In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.
3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.
4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.

   — When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.
5. Differences between Products

   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.

   — The characteristics of an MPU or MCU in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

RENESAS

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700, Fax: +44-1628-651-804

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141