RX600 Series

How to Setup and Run CoreMark on a MCU

R01AN0757EU0100
Rev.1.00
Oct 25, 2011

Introduction

This document details the steps needed to obtain, configure, and run the EEMBC CoreMark Benchmark. The RDK RX62N was the platform chosen for the examples throughout this document and results for this setup are shown. The Renesas High-performance Embedded Workshop (HEW) IDE is used along with the KPIT GNU RX Toolchain.

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

http://www.coremark.org/

Target Device

The RX62N 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 a 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 inherit 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.

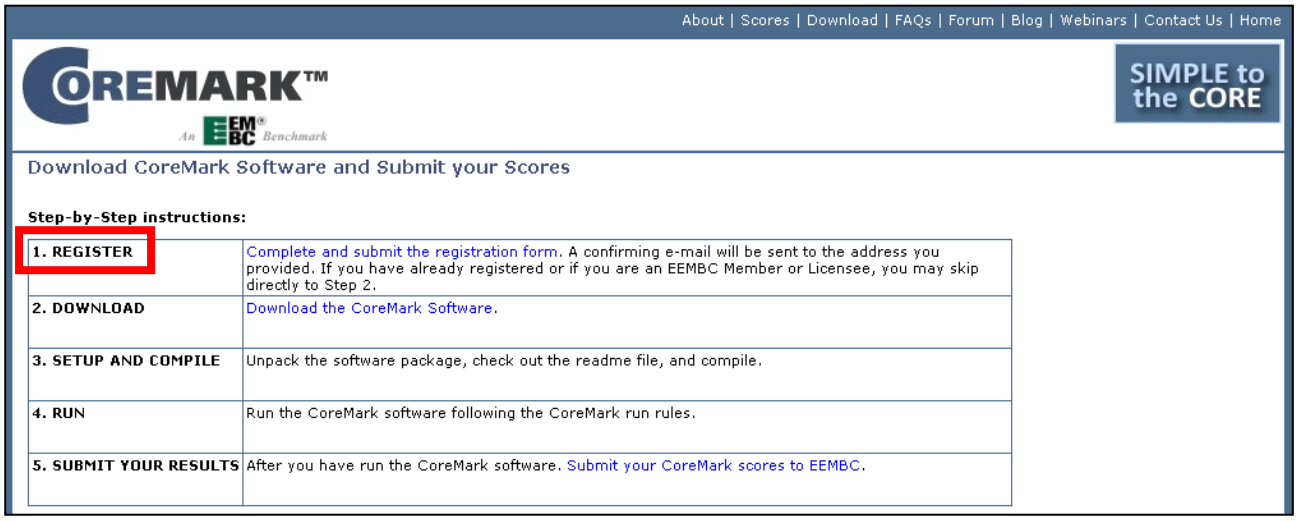


Figure 1 : Register & Download CoreMark

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.

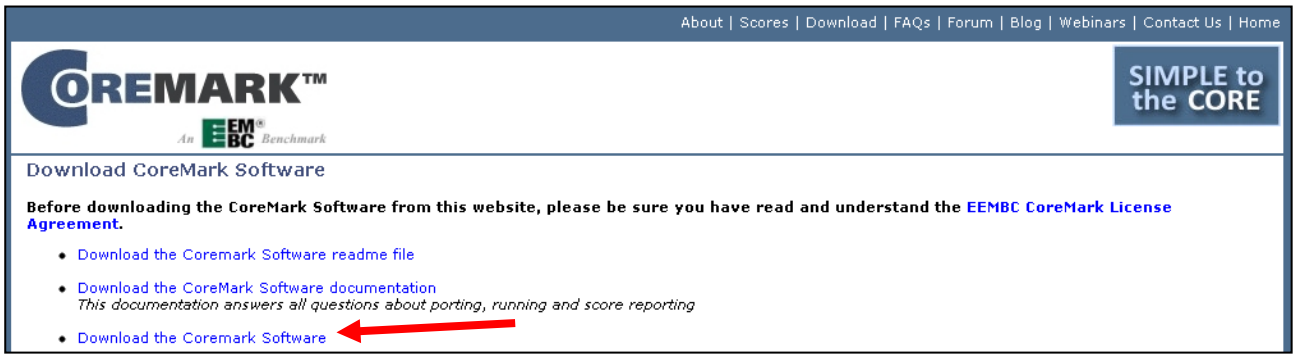


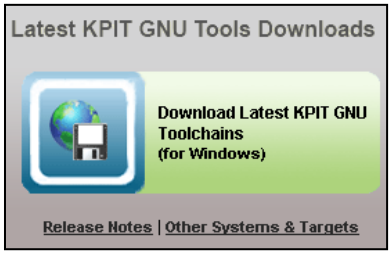
Figure 2 : Download CoreMark Software & Documentation

3. Downloading & Installing the KPIT GNU Tools

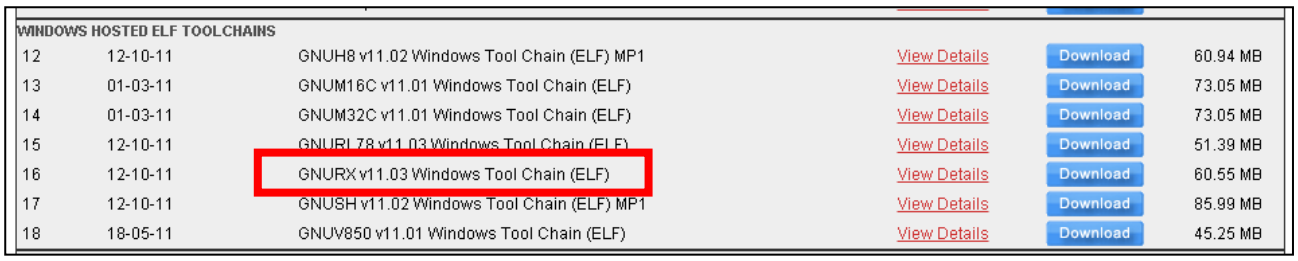
This section assumes that the Renesas High-performance Embedded Workshop (HEW) is already installed. If it is not installed, please go the Renesas website and download and install the latest Renesas RX Toolchain. This will install the Renesas RX Toolchain and HEW.

The example project described in this document uses the KPIT GNU RX Toolchain. KPIT provides free GNU toolchains for Renesas MCUs with free customer support. To download and install the KPIT GNU RX Toolchain follow these steps:

1. Go to http://www.kpitgnutools.com/ in a web browser.
2. If this is your first time using KPIT tools then click the Register button and obtain a username and password.
3. Click the button that says 'Download Latest KPIT GNU Toolchains (for Windows)'



4. Log in with your KPIT username and password.
5. Under the list of tools download the latest GNU RX Toolchain. As of this writing it was v11.03.



6. After downloading, install the GNU RX Toolchain. It will automatically integrate the toolchain with HEW.

4. Creating a CoreMark HEW Project

Now that we have the CoreMark source files and the KPIT GNU RX Toolchain we can create a HEW workspace. Start off by opening HEW and creating a new workspace for the MCU you are using.

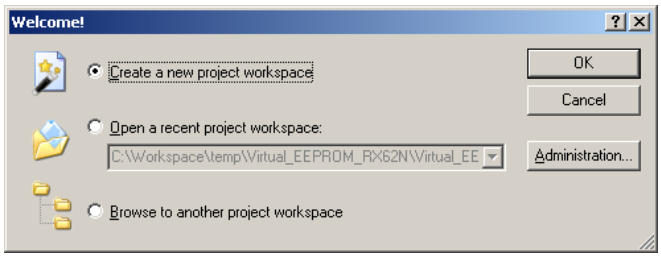


Figure 3 : Create New HEW Workspace

The following steps are related to the example used with this application note. Choose 'RX' for the CPU family and 'KPIT GNURX [ELF]' for the toolchain. Make sure 'C Application' is chosen in the left pane and enter a name in the 'Workspace Name' input box. Click OK.

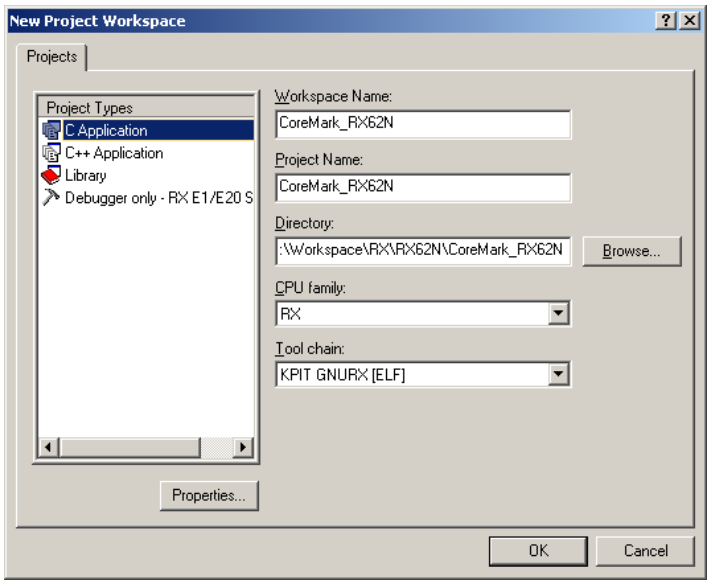


Figure 4 : Choosing Toolchain

On the next screen choose the latest toolchain version and the RX62N.

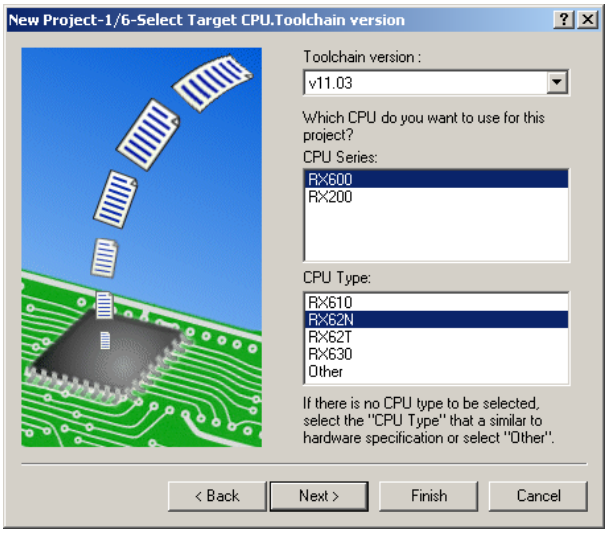


Figure 5 : Choose RX62N & Toolchain Version

Continue to click ‘Next’ until you get to the screen where you choose the target system for debugging. On this screen select the ‘RX600 Segger J-Link’. The Segger J-Link debugger comes installed on the RX62N RDK board.

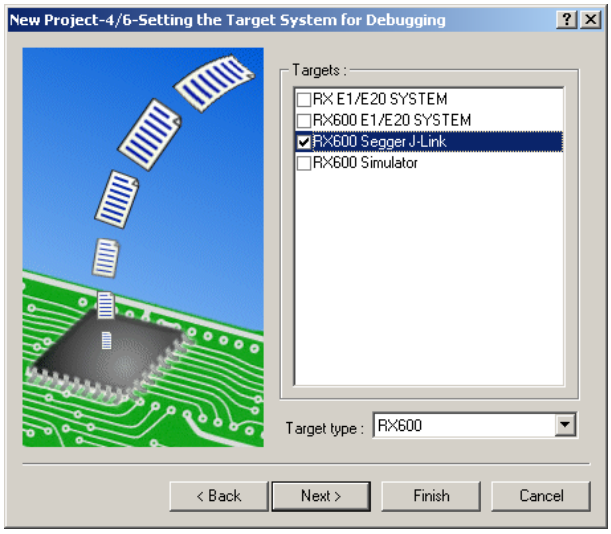


Figure 6 : Choose J-Link for Debugging

Click ‘Finish’ and then ‘OK’ in the next pop-up window. At this point the HEW workspace has been created. The CoreMark software and board support files can now be added.

5. Adding CoreMark Source

Once the HEW workspace has been setup the CoreMark source files can be added. Find the CoreMark Software package that was downloaded earlier and copy the following files to your HEW project:

- 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 HEW project by going to Project >> Add Files. After adding the files you may wish to group them in the HEW navigation pane by right-clicking on ‘C source file’ and choosing ‘Add Folder’. After creating the folder drag the CoreMark files into it.

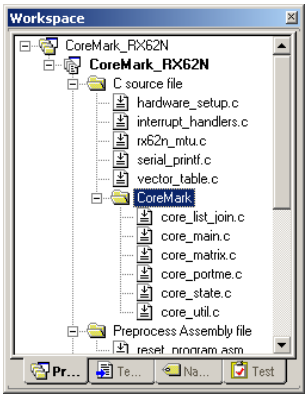


Figure 7 : Add & Group CoreMark Files

Since CoreMark has its own main() function the user will need to remove the one that comes by default with the HEW project. The source file that needs to be removed will have the same name that was given to the project. The example project used in this document is named 'CoreMark_RX62N' so the *CoreMark_RX62N.c* file needs to be removed from the project. This is done in HEW by clicking Project >> Remove Files. From the window that pops up, select *CoreMark_RX62N.c* and click 'Remove'.

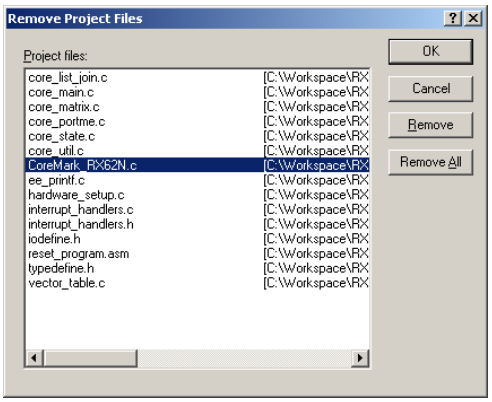


Figure 8 : Remove Default main()

6. 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 RX62N RDK and the board support files can be found in the 'bsp' (Board Support Package) folder that was packaged with this application note. Copy the files listed below to your own project workspace and then add them to your project in HEW. Adding files to your project in HEW is done by clicking Project >> Add Files.

- *rx62n_mtu.c & rx62n_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.

7. 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 (located here: <CoreMark Software>/docs/index.html) can be consulted.

7.1 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 RX62N example setup.

- HAS_FLOAT
  - Should be defined as '1' since the RX62N has a single-precision FPU
- 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 KPIT GNU toolchain does support stdio.h
- HAS_PRINTF
  - Should be defined as '1' since KPIT GNU 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`	`#ifndef CLOCKS_PER_SEC`
	`/* Timer clock frequency: PCLK=48MHz */`
	`#define CLOCKS_PER_SEC 48000000`
	`#endif`
	
	`#define NSECS_PER_SEC CLOCKS_PER_SEC`

7.2 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 discussed in Section 8.3. 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. This is done by following these steps:

1. Go to Build >> KPIT GNURX [ELF] Toolchain.
2. Make sure the 'C/C++' tab is chosen.
3. Change the 'Show entries for' dropdown to 'Defines'
4. Click 'Add'
5. Put the name of the definition in the 'Macro' input box.
6. Put the value of the definition in the 'Replacement' input box.

Use these steps to define ITERATIONS to be '6000'.

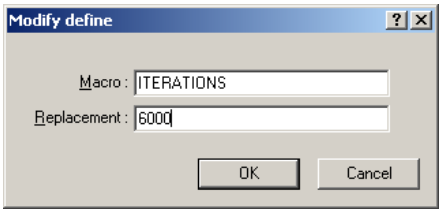


Figure 9 : Defining ITERATIONS

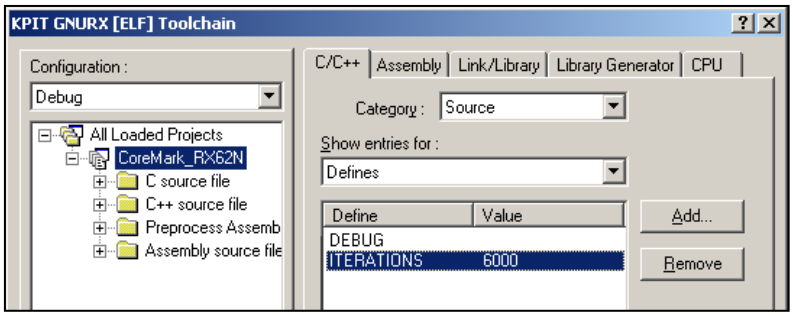


Figure 10 : ITERATIONS Definition

7.3 Increasing Stack Size

The default memory allocation method used by CoreMark is to use the stack. Using the KPIT GNU 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. Go to Build >> KPIT GNURX [ELF] Toolchain.
2. Make sure the 'Link/Library' tab is chosen.
3. Change the 'Category' dropdown to 'Sections'.
4. Scroll down the list until '.ustack' is shown.
5. Double click on '.ustack' and when the window pops up change the address to 0x5000. Click OK.
6. Double click on '.istack' and change its address to 0x5100.
7. Click OK.

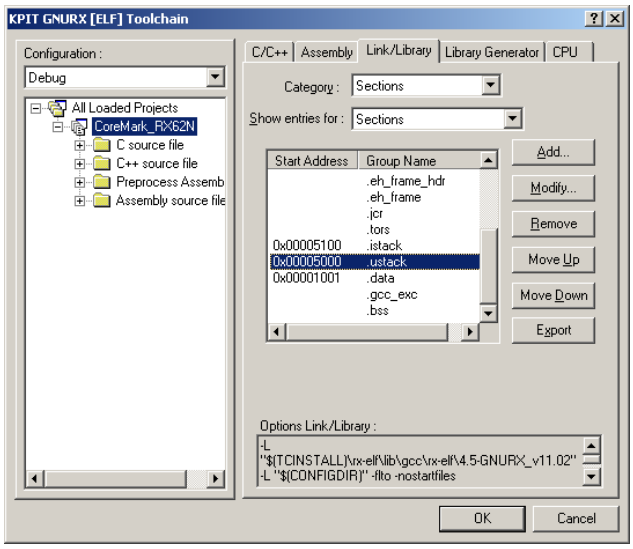


Figure 11 : Increasing Stack Size

7.4 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 RX62N 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 48MHz this gives a maximum time of around 89.5 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 *rx62n_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 SCI2-B which is connected to the serial port on the RDK board. 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. The user should add the *rx62n_mtu.c* and *serial_printf.c* files to their project before making these changes.

Find:	Change to:

`#include <stdlib.h>`
`#include "coremark.h"`
	`#include <stdlib.h>`
	`#include “rx62n_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

8. 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 10.

8.1 Compiler Optimizations

There are two optimizations that need to be turned on with the compiler. The reason for this will be discussed later.

1. Go to Build >> KPIT GNURX [ELF] Toolchain.
2. Make sure the 'C/C++' tab is chosen.
3. Change the 'Category' dropdown to 'Optimize'
4. Make sure the 'Optimize C/C++ Source Code' box is checked
5. Change 'Optimization Type' to 'Optimize for speed'.
6. Make sure the 'Enable Link-time optimizations (-flto)' box is checked.
7. Click OK.

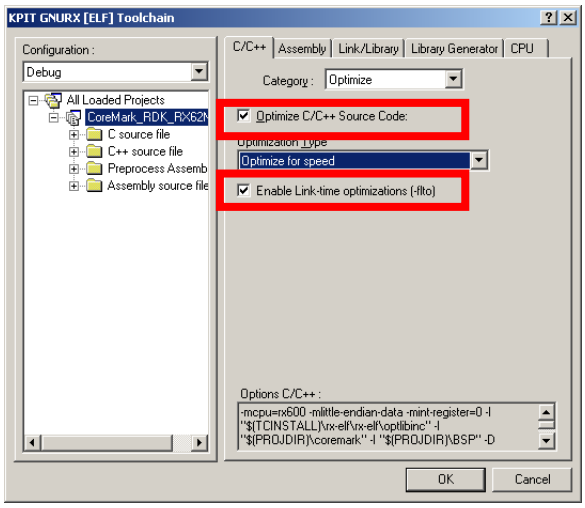


Figure 12: Enable Link-time Optimizations for Compiler

8.2 Linker Optimizations

The linker settings are where the optimization settings that make a large performance impact will be defined.

8. Go to Build >> KPIT GNURX [ELF] Toolchain.
9. Make sure the 'Link/Library' tab is chosen.
10. Change the 'Category' dropdown to 'Output'.
11. Make sure the 'Enable link-time optimizations (-flto)' box is checked.

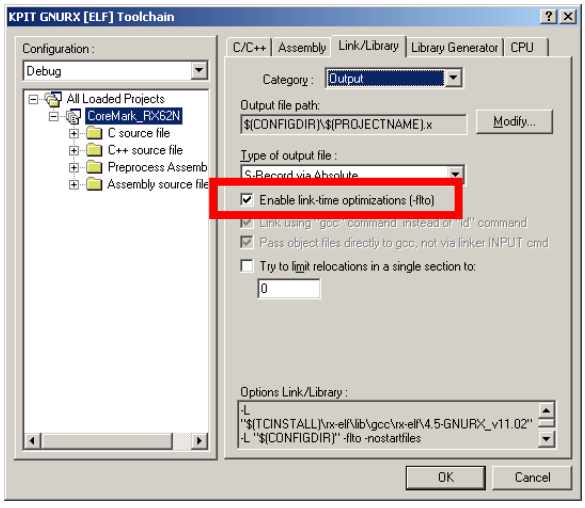
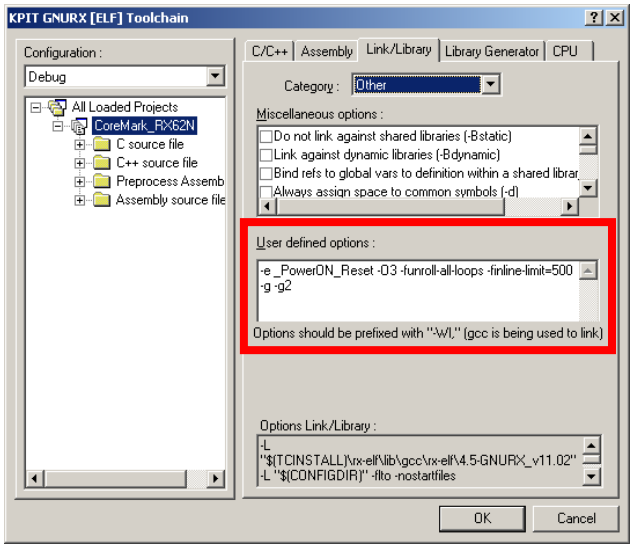


Figure 13 : Enable Link-time Optimizations for Linker

12. Change the ‘Category’ dropdown to ‘Other’.
13. Append the following string to the ‘User defined options’ input box: “-O3 -funroll-all-loops -finline-limit=500 -g -g2” (without the quotation marks).



14. Click OK.

Some users might see the options that were used for the linker optimizations and think that they appear to be optimizations that would normally be used for the compiler. The reason the optimizations are setup this way is because Link-Time Optimizations (-flto) are enabled. When this setting is enabled the link-time optimizer is used at the link stage. The –flto option should be used both when compiling and when linking. When compiling with the –flto option the compiler will store a special bytecode representation (used internally by GCC tools) of the source in the object file. When the linker is called with the –flto option it will extract these separate bytecode images and will merge them all into a single internal image. The resulting image will then be compiled with the optimizations specified. Since the bytecode images were combined before being optimized inter-module optimizations will be performed.

For a full description of this option users can reference http://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html#Optimize-Options.

8.3 Reporting Optimization Settings

When reporting the CoreMark results the optimization settings used are included. Using the same steps as detailed in Section 7.2 add a definition for **FLAGS_STR** to be \"**-O3_-flto_-finline-limit-500_-funroll-all-loops**\". The (\") are included for compilation purposes. These are the settings that were detailed earlier in this section. If the user changes these settings, then they will need to change this definition as well for accurate reporting.

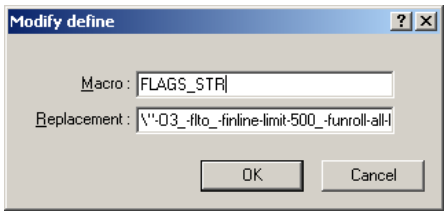


Figure 14 : Defining Optimization Settings

9. Run the CoreMark Benchmark

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

1. Connect the RDK to your PC using the supplied USB cable.
2. Change the debugging session from 'DefaultSession' to 'SessionRX600_Segger_Jlink' using the session drop down box.

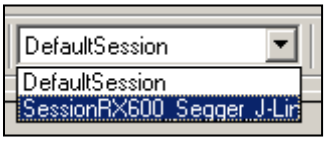


Figure 15 : Select JLink Debugging Session

3. Choose 'RX62N Group' for the 'MCU Group'.
4. Choose 'R5F562N8' for the 'Device'. Click Next.

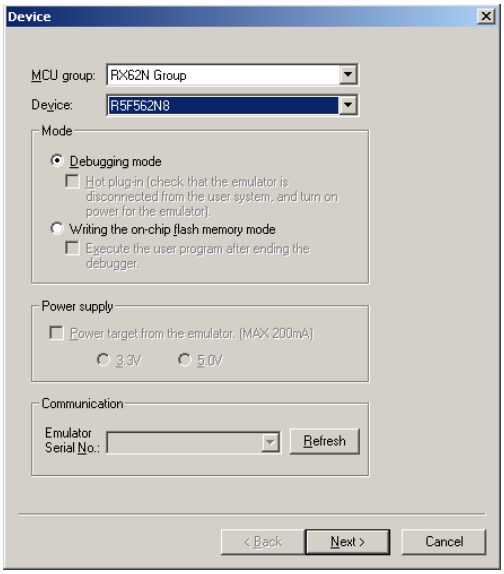


Figure 16 : Choose MCU to Debug

5. Choose '16.5Mhz' for the JTAG Clock and hit Finish.

6. In the next window input ‘12’ for the ‘Input clock (EXTAL)’. Click OK.

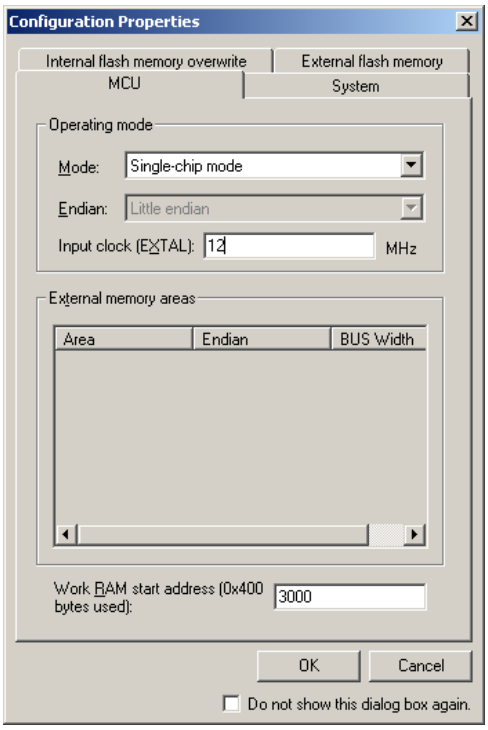


Figure 17 : JLink Configuration Properties

7. When the connection process has completed double-click on the icon underneath the ‘Download modules’ folder in the left pane. This will download the program to the RX62N MCU.

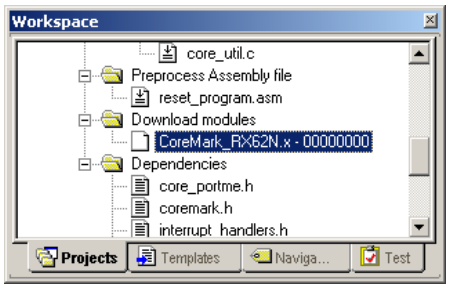


Figure 18 : Download Program to MCU

8. Connect a serial cable between the RDK board and your PC.
9. 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
10. Click Debug >> Reset Go.
11. After approximately 27 seconds the results should show up in the terminal window. Example output is shown in Section 10.

10. Results for RX62N

This section shows the results for the example project that was built throughout this document. The system used was a RDK RX62N. These results will be the same for all RX610, RX62x, and RX63x devices since they all share the same core.

CoreMark Output

```
2K performance run parameters for coremark.
CoreMark Size    : 666
Total ticks      : 1281394099
Total time (secs): 26.695709
Iterations/Sec   : 224.755239
Iterations       : 6000
Compiler version : GCC4.5-GNURX_v11.03
Compiler flags   : -O3_-flto_-finline-limit-500_-funroll-all-loops
Memory location  : STACK
seedcrc          : 0xe9f5
[0]crclist       : 0xe714
[0]crcmatrix     : 0x1fd7
[0]crcstate      : 0x8e3a
[0]crcfinal      : 0xa14c
Correct operation validated. See readme.txt for run and reporting rules.
CoreMark 1.0 : 224.755239 / GCC4.5-GNURX_v11.03 -O3_-flto_-finline-limit-500_-
funroll-all-loops / 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 RX62N was run at 96MHz.

CoreMark/MHz = CoreMark Score / Clock Speed

CoreMark/MHz = 224.755239 / 96MHz

CoreMark/MHz = 2.341

To report the score, CoreMark recommends using the same format as before. This is shown below.

CoreMark/MHz

CoreMark/MHz 1.0 : 2.34 / GCC4.5-GNURX_v11.03 -O3 –flto -finline-limit=500 -funroll-all-loops / STACK

11. 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 of 230 and MCU-B has a CoreMark of 220 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	Oct.25.11	—	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.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.
   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

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: +44-1628-585-100, Fax: +44-1628-585-900

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.
1 harbourFront Avenue, #06-10, keppel Bay Tower, Singapore 098632
Tel: +65-6213-0200, Fax: +65-6278-8001

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

Colophon 1.1