RX610 Group

How to Setup and Run Dhrystone on a MCU

REU05B0134-0121
Rev.1.21
Jul 14, 2010

Abstract

This document discusses how to setup the Dhrystone benchmark for your MCU and toolchain. The RX610 was the MCU chosen for the examples throughout this document including a complete description of the measurements and results. The benefits and shortcomings of the Dhrystone benchmark are also briefly discussed.

Introduction

The Dhrystone benchmark is probably the most well known benchmark in the MCU industry. While there are newer benchmark suites that seem to offer performance numbers for every imaginable MCU task, many companies continue to use Dhrystone. One of the main reasons for this is that Dhrystone is so easy to use. You add a couple of files to your project, configure it to run on your system, and you are good to go. Even better, after it runs you get one number for the result. You don't have to dig through pages of output, you just take that number and now you know that everyone with a number less that that is inferior to you. But is this true? Not exactly. Dhrystone was originally designed to give users a point of reference for performance on 'typical' applications. Since comparing MIPS performance on a RISC to a CISC is hard to do, the one number result of Dhrystone was very convenient. What Dhrystone does not do is give you a definite answer on which MCU will be the best performer for you.

At the time, Dhrystone was put together to be a 'typical' application by studying real programs. While this is good for an overview of a MCU, it is not at all an exhaustive test. And the designers of Dhrystone knew this; throughout the 'documentation' that comes packaged with Dhrystone (by documentation I am referring to the text files that are included in the Dhrystone archive) the authors talk about the extent of Dhrystone's usefulness. In particular, some things that are not measured include:

- I/O
- OS Performance
- Floating Point

While your application might not use an RTOS or FPU, it's highly unlikely it doesn't use some form of I/O. So what do you do in this case? You can read the hardware manual on the product or run your own benchmarks. The point is that Dhrystone is just one of the things you should look at when comparing performance across MCUs. Much more expansive and descriptive analysis of Dhrystone is available on the internet if you want more information.

Target Device

The RX610 is used as an example in this document but the same methods can be used to properly configure Dhrystone on any MCU.

Contents

1. Obtaining Dhrystone

There is no official Dhrystone site where the source code can be found. Instead if you search on the internet you will find many sites hosting source code, precompiled binaries, results, and more. For this application note the following archive was used because it had valid Dhrystone benchmark source files:

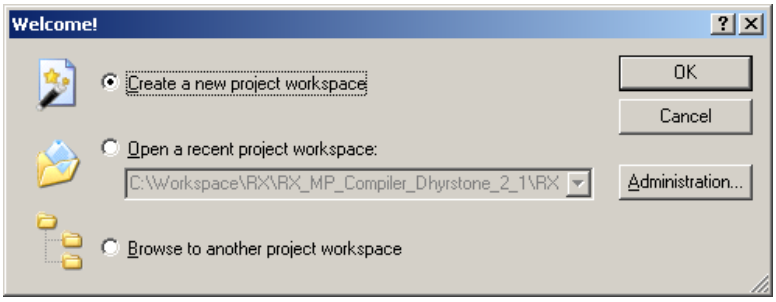
http://ftp.unicamp.br/pub/unix-c/benchmark/system/dhrystone-2.1.tar.gz

After extracting the archive you should have the following files:

- bymanuf
- byperf
- cc_dry2
- cc_dry2reg
- clarify.doc
- dhry-2.1.p
- dhry.h
- dhry.p
- dhry_1.c
- dhry_2.c
- dhry_c.dif
- doit
- Makefile
- pure2_1.dif
- RATIONALE
- README
- README.RER
- results
- submit.frm

2. Creating a Dhrystone HEW Project

Now that we have the Dhrystone source files we can put them in a HEW workspace. Start off by opening HEW and creating a new workspace for the MCU you are using.



Once you have created the new project make sure to add any files that are needed for configuring the MCU. Examples would be files to setup your vector table, configure the system clocks, initialize data sections, etc.

The next step will be to add the Dhrystone source files to your project. The files that you need to add to your workspace are:

- dhry.h
- dhry_1.c
- dhry_2.c

There are also some files that are included for this RX example. These files should also be included in your project and they are:

- rx_dhry.c
- rx_dhry.h
- simple_printf.c

You add source files to your workspace by going to Project >> Add Files within HEW. If the project generator generated a file with the main() function in it when your made your project then you will need to remove the file or exclude it from the build. The reason for this is that ‘dhry_1.c’ has its own main() function and we will be using it. You can exclude a source file from the build by right clicking on its icon in the Navigation pane and clicking ‘Exclude Build …’.

3. Modifying Dhrystone Files for Your MCU

In order for Dhrystone to work correctly you will have to make some modifications to the source files.

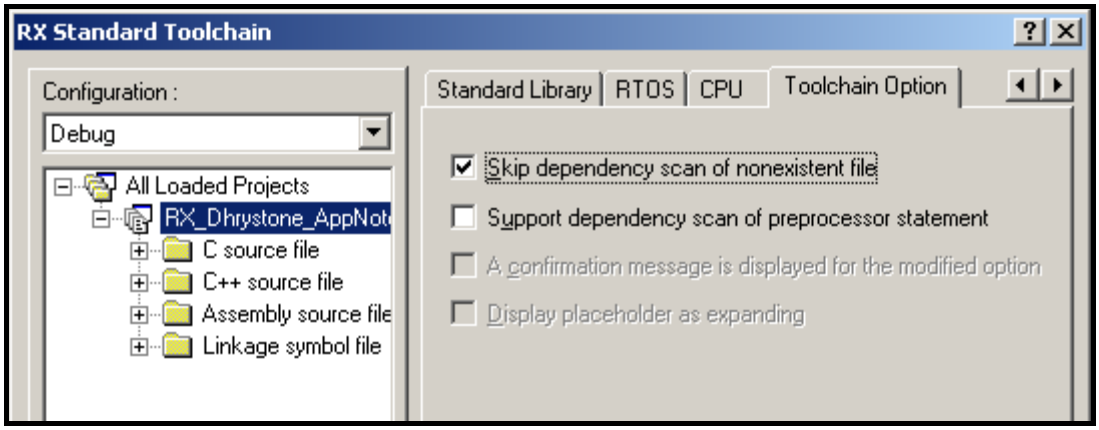
3.1 Modifying dhry.h

The only change we are going to make in this file is to make sure that 'TIMES' is not defined. This is done by commenting out the following lines of code:

```
/* Commented out because we do not have an OS function call to get the time */
/*
#ifndef TIME
#undef  TIMES
#define TIMES
#endif
*/
```

The reason this is done is because by default the Dhrystone code makes OS calls to get the time before and after the benchmark. The times are then subtracted to get the benchmark time. While code could be written to use a timer to perform this function, in this example we obtain the benchmark time by flipping a port pin and measuring the time on an oscilloscope. This method is used for its simplicity.

Even though we commented out the DEFINE statement for TIMES an error could still be generated saying that the header files 'time.h', 'sys\types.h', and 'sys\times.h' do not exist. This error is generated because the toolchain looks for the header file before it sees if it really needs it. To get rid of this error go to Build >> RX Standard Toolchain (this should work the same for other toolchains). In the window that comes up click the right arrow close to the top right until the 'Toolchain Option' tab is visible. Now check the box next to 'Skip dependency scan of nonexistent file'. Click OK and compile again. The errors should be gone.



3.2 Modifying dhry_1.c

3.2.1 Initialize Hardware and Output printf() on UART

The first thing we will add to this file is code to initialize our hardware. In this example we are calling two functions at the beginning of the main() function which are listed below. Remember when calling external functions to also include the appropriate header files ('hwsetup.h' and 'rx_dhry.h' in this example).

1. **HardwareSetup()** - This function takes care of setting up the clock tree and port pins on our MCU. The file 'hwsetup.c' is usually included when you create a new workspace in HEW and that is where this function is located. If you have your own hardware initialization code then replace this function call with your own.
2. **InitSerial()** - This function configures the UART that I want to use with the printf() function

The Dhrystone benchmark uses printf() to output information. Most importantly in our case it is used after the benchmark to print out validation information. If the results are not valid then you know the compiler did something it was not supposed to. The easiest way to configure the printf() function for Renesas MCU UARTs is to use sprintf() (simple printf). For more information on using sprintf() you can download the archive below that includes the application note and source code:

http://america.renesas.com/fmwk.jsp?cnt=/an_reu05b0076_mcu_code_example.jsp&fp=/support/downloads/download_results/C2014201-C2014300

In order to use the sprintf() function you just need to add the 'simple_printf.c' file to your project. Once this is done you will need to override the fputc() and putc() functions to make the output go through the UART. An example of this done for the RX610 is shown below.

```
/*****************************************************************
   stdio functions
   This is needed in order to remove the inclusion of the full
   stdio library which will added un-needed (and un used) code to
   your project.
*****************************************************************/
int fputc(int c, FILE *st)
{
  /* Wait for TDR to be empty */
  while(TXI_IR == 0);

  /* Clear TXI IR bit */
  TXI_IR = 0;

  /* Write the character out */
  SCI1_TDR = (unsigned char) c;

  return c;
}

int putc(int c, FILE *st)
{
  /* Wait for TDR to be empty */
  while(TXI_IR == 0);

  /* Clear TXI IR bit */
  TXI_IR = 0;

  /* Write the character out */
  SCI1_TDR = (unsigned char) c;

  return c;
}
```

After making sure these functions are in your project you call the printf() function as you normally would and the output will go wherever you specified in the fputc() and putc() functions. While the output goes to the UART in this example, that is not a requirement.

3.2.2 Remove scanf()

The next change that we will make is removing the scanf() function call. You have the option of implementing this function for your MCU but it is simpler to just remove the function call and give the variable a constant value. For this example 10,000 iterations were chosen but you can change this to however many runs you want.

```
/* Replace scanf() with giving 'Number_Of_Runs' a constant value */
/*
int n;
scanf ("%d", &n);
Number_Of_Runs = n;
*/
Number_Of_runs = 10000;
```

3.2.3 Removing Time Calculation

After the actual Dhrystone benchmark code has been executed the validation and elapsed time portions are output using printf(). Since we are not measuring the time internally to the MCU and the sprintf() function does not support float data types, we are going to comment this section of code out. The code that needs to be commented out is shown below. We also add an infinite loop at the end of the code.

```
  /* Comment out this section because we did not measure time with
     the MCU; also the simple printf function does not support floats */

  /*

  User_Time = End_Time - Begin_Time;

  if (User_Time < Too_Small_Time)
  {
    printf ("Measured time too small to obtain meaningful results\n");
    printf ("Please increase number of runs\n");
    printf ("\n");
  }
  else
  {
#ifdef TIME
    Microseconds = (float) User_Time * Mic_secs_Per_Second
                         / (float) Number_Of_Runs;
    Dhrystones_Per_Second = (float) Number_Of_Runs / (float) User_Time;
#else
    Microseconds = (float) User_Time * Mic_secs_Per_Second
                         / ((float) HZ * ((float) Number_Of_Runs));
    Dhrystones_Per_Second = ((float) HZ * (float) Number_Of_Runs)
                         / (float) User_Time;
#endif
    printf ("Microseconds for one run through Dhrystone: ");
    printf ("%6.1f \n", Microseconds);
    printf ("Dhrystones per Second:                      ");
    printf ("%6.1f \n", Dhrystones_Per_Second);
    printf ("\n");
  }

  */

  while(1){};
```

3.2.4 Flip Port Pin for Oscilloscope

Since we are measuring the benchmark time by flipping a port pin and watching it on an oscilloscope we need to add this code right before and after the benchmark. To do this add the code SCOPE_HI() right before the main loop and then add SCOPE_LO() right after the loop has finished. It will look like below:

```
/* Set port pin high when benchmark begins */
SCOPE_HI();

for (Run_Index = 1; Run_Index <= Number_Of_Runs; ++Run_Index)
{

  Proc_5();
  ...
  ...

} /* loop "for Run_Index" */

/**************/
/* Stop timer */
/**************/

/* Set port pin low when benchmark finishes */
SCOPE_LO();
```

These are not real function calls. They are really defines that are included in ‘rx_dhry.h’. The define statements can be seen below.

```
/* Defines used to signal when the benchmark starts and when it ends */
/* Turn LED0 on RX610 RSK off (1) */
#define SCOPE_HI()PORT8_DR |= 0x08;
/* Turn LED0 on RX610 RSK on (0) */
#define SCOPE_LO()PORT8_DR &= 0xF7;
```

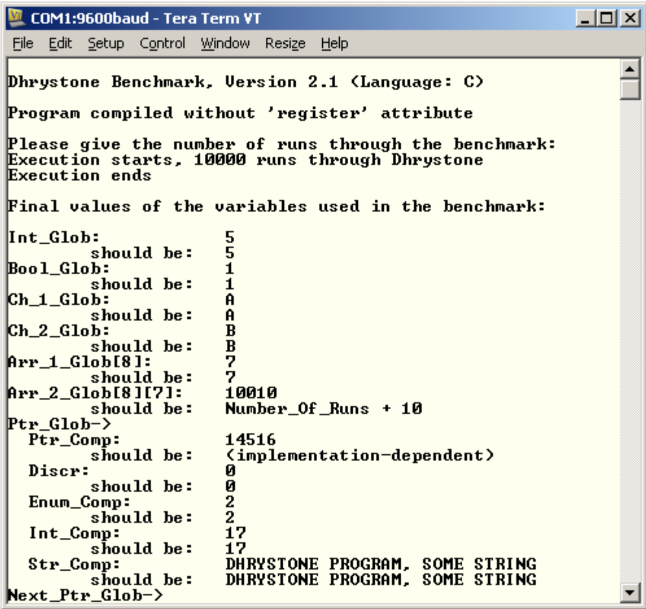
The LEDs also have to be designated as output ports which is done in the HardwareSetup() function that is called before the benchmark loop.

4. Running Dhrystone and Calculating DMIPS

We can now run the benchmark by programming the flash or by using a debugger. For the RX example we are using the E1 debugger. It should be noted that some debuggers are not as transparent as others and could possibly cause a slow down in performance. In this case it is recommended to program the flash and run the code without using a debugger.

We are flipping a port pin to signal when the benchmark has started and ended so you will need to connect an oscilloscope to your system to measure the time. Also, the printf() statements are being sent out of the UART so you will need to connect a DB9 cable between your computer and development system. Then on your computer you can open up a terminal program, like HyperTerminal, and configure it for your settings.

After running the benchmark you will have the execution time on your oscilloscope and the validation information in your terminal program. Look through the data that is output at the end of the benchmark and make sure that all the information is valid. An example of the data that will be output is shown below.



With the execution time we can now determine the Dhrystone MIPS number (DMIPS). This is first done by figuring out how many Dhrystone runs you can complete in 1 second, also called the Dhrystone score in some cases. The formula for this is below.

Dhrystones per Second = (1 second / Measured Execution Time) * Number_of_Runs

Number_of_Runs is the number of loop iterations you chose in Section 3.2.2 (this example uses 10,000).

You then take the Dhrystones per Second value and divide it by the constant, 1,757. This number was chosen because it was the Dhrystone score of the VAX 11/780 which is considered a 1 MIPS machine.

DMIPS = Dhrystones per Second / 1,757

Since MCUs vary much in speed and efficiency it is usually desirable to see the Dhrystone score as a function of the MCU frequency. This helps normalize the value across MCUs and can help show the effectiveness of the MCUs core rather than how fast it can run. To find this number, called DMIPS/MHz, just divide the DMIPS number you just calculated by the operating frequency that your MCU was tested under.

DMIPS/MHz = DMIPS / Frequency of MCU in MHz

All together the formula looks like this:

$$\text{DMIPS/MHz} = \frac{(1 \,/\, \text{Execution Time}) * \text{Number\_of\_Runs}}{1{,}757 * \text{MCU Frequency in MHz}}$$

5. Optimizing Dhrystone

You can find many articles on the internet on ways to optimize Dhrystone. The following methods are not exhaustive, but in practice have shown to offer the most benefit. One thing that should be mentioned prior to turning on all these optimizations is that Dhrystone does have some loose standards on what optimizations should not be allowed. These 'rules' are included with some of the Dhrystone packages you find online. The fact that the word 'some' was used is what makes the rules loose and is one of the main detractors from being able to compare Dhrystone scores. If there is no committee to set the rules and validate the scores then the reported results are in the hands of individuals to confirm. Since confirming each Dhrystone score would take huge amounts of time, this is impractical. The Dhrystone designers recognize this as an issue and this is just another reason why they state that Dhrystone should never be the only measurement you use when choosing an MCU. In general, a Dhrystone number is meaningless without compiler settings and overall benchmarking conditions. Even when compiler settings are given, the burden still rests upon the reviewer to see what those settings actually mean. An example is if the optimization settings are reported as "-O3". To actually find out what "-O3" does inside the compiler, the reviewer will have to read the compiler manual.

5.1 String Operations

One of the characteristics of Dhrystone is that it is relatively heavy in string operations. This means that MCUs with dedicated string operation instructions can have an advantage. In this example we are using the RX toolchain which uses SMOVU instruction by default when calling the strcpy() function. It is worth the effort to check and make sure you are using these instructions if your MCU supports them. Also, making use of intrinsics is a standard way of optimizing performance and should be considered.

Another thing to mention about the string operations is that it is allowed to inline the string operation functions (example: strcpy). While inlining Dhrystone functions is disallowed, inlining string functions is seen as acceptable. The inlining of string functions is worth implementing since it can give a considerable performance boost. To make sure the strcpy and strcmp functions are inlined in the Dhrystone benchmark for the RX the user should make sure to include 'string.h' (#include <string.h>) in both 'dhry_1.c' and 'dhry_2.c'.

5.2 Function Inlining

The Dhrystone documentation states that procedure merging or function inlining is disallowed. If these methods are used then it should be noted when stating the Dhrystone score. For further reasoning behind this decision refer to the RATIONALE document that comes with the Dhrystone archive.

To turn on function inlining with the RX toolchain do the following:

1. Go to Build >> RX Standard Toolchain
2. Select the 'C/C++' tab
3. Choose 'Optimize' from the Category drop down.
4. If 'Optimize for speed' is selected then function inlining will be done by default when an 'Optimize level' of 2 (inline = 100) or Max (inline = 250) is chosen. If 'Optimize for size' is selected then no function inlining will be done. Refer to the User's Manual for the RX toolchain for further information on how to use custom levels of function inlining.

To use an 'Optimize Level' of 2 or Max with speed priority without using function inlining you will need to do the following:

1. Go to Build >> RX Standard Toolchain
2. Select the 'C/C++' tab
3. Click the 'Details' button
4. Select the 'Inline' tab
5. In the drop down box choose 'None'

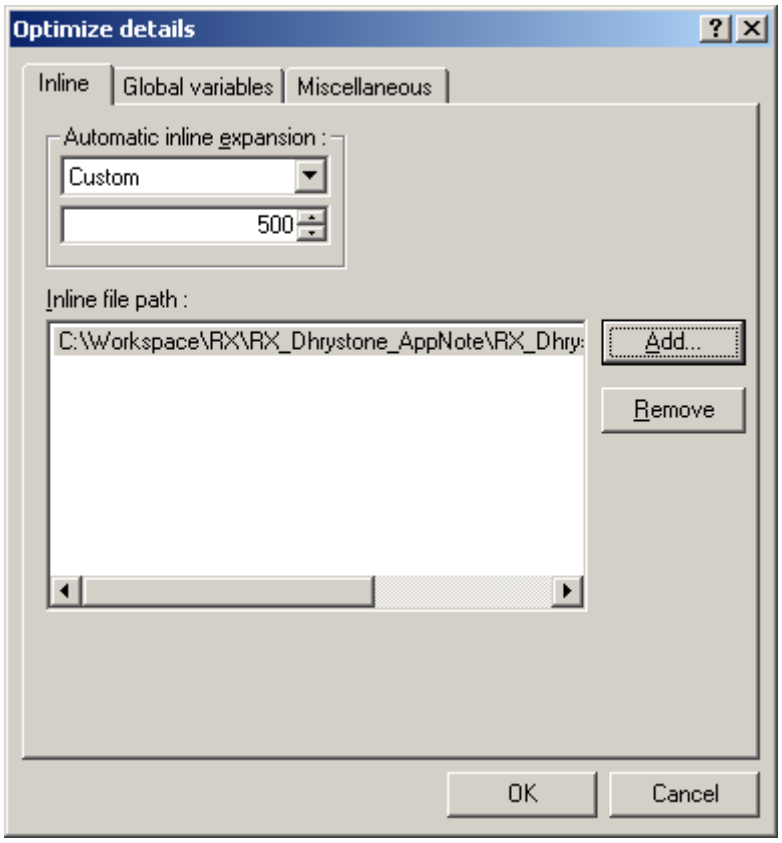
This window is also where you can select a custom level of inline expansion.

5.3 File Merging

Dhrystone documentation also states that each module should be compiled separately which means no module should have knowledge of the procedures and variables located in other modules. If this method is used then it should be indicated when stating the Dhrystone score. This method can increase the Dhrystone score because it allows the compiler to inline the functions located in the 'dhry_2.c' file. To implement this on the RX do the following:

1. Go to Build >> RX Standard Toolchain
2. Select the 'C/C++' tab
3. Click the 'Details' button
4. Select the 'Inline' tab
5. Click the 'Add' button.
6. Select 'Custom directory' from the 'Relative to' drop down box.
7. Click the 'Browse' button and locate the file 'dhry_2.c' and click OK.

Note that if you receive a warning or error about 'dhry_2.c' being included twice then you can remove this message by excluding 'dhry_2.c' from your project. You can do this by right-clicking on the file in the Navigation pane and choosing 'Exclude build dhry_2.c'.



5.4 Register Declaration

While the Dhrystone document states that a good compiler should be able to optimally put variables in registers itself, you are allowed to use register declarations. This did not have an effect with the RX toolchain since registers were already being used; but it might with other compilers. Here is how you would enable it on the RX toolchain.

1. Go to Build >> RX Standard Toolchain
2. Select the 'C/C++' tab
3. Click the 'Details' button
4. Select the 'Miscellaneous' tab
5. Check the box next to 'Enable register declaration'
6. Go back to the HEW editor and add the following line to the top of the file:
   #define REG register
7. This define is needed because the Dhrystone uses 'REG' for register declarations while the RX toolchain uses 'register'. If your toolchain uses another directive then replace 'register' with the name your toolchain uses.

6. Results of RX Dhrystone Benchmark

This section will show and discuss results found from running the Dhrystone benchmark on the RX610. The results are shown for different compiler settings discussed in Section 5.

Setup	
MCU	RX610 @ 100MHz
Toolchain	RX v1.0
Development Board	RX610 RSK
Code Running From	On Board Flash
Base Optimization Setting	Max Optimization for Speed
Inline Setting for Base Settings	inline = 250, loop=32, schedule instructions

RX610 RSK	**RX @ 100MHz**		
Dhrystone 2.1 (10,000 iterations)	**Speed (ms)**	**Dhrystones/Sec**	**DMIPS/MHz**
Base	33.90	294985.3	1.68
Base, no inline	41.30	242130.8	1.38
Base, no inline, 'register' used	41.30	242130.8	1.38
Base, dhry_2.c file inline	25.60	390625.0	2.22

These results show how much variance there can be from changing the compiler settings. When turning on function inlining the time is reduced by around 18%. When combining function inlining with file merging the execution time drops by over 38%.

7. Summary

A detailed description and instructions were provided for replicating the measurements and results of the Dhrystone v2.1 benchmark for the RX610 MCU. These steps can be used to run the benchmark and report results for any MCU/MPU.

The results in Section 6 show the importance of well documented Dhrystone results. Compiler settings are clearly integral to providing useful performance data. DMIPS and DMIPS/MHz numbers are meaningless without specific compiler information, settings and instructions for repeating the measurements.

The RX610 was shown to achieve a 1.68 DMIPS/MHz rating with the Renesas v1.0 compiler set to max speed optimization which includes function in-lining. This is just slightly higher than the advertised value of 1.65 DMIPS/MHz.

Website and Support

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

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

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Jan.25.10	—	First edition issued
1.10	Feb.18.10	—	Added ‘string.h’ information, commented on “-O3”, and updated Dhrystone results
1.11	Feb.22.10	11	Fixed an incorrect result number
1.12	Feb.23.10	4, 6	Changed code examples
1.21	Jul.14.10	—	Changes and fixes after review

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 manual, refer to the relevant sections of the manual. If the descriptions under General Precautions in the Handling of MPU/MCU Products and in the body of the manual differ from each other, the description in the body of the manual takes precedence.

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 one with a different type number, confirm that the change will not lead to problems.
   — The characteristics of MPU/MCU in the same group but having different type numbers may differ because of the differences in internal memory capacity and layout pattern. When changing to products of different type numbers, implement a system-evaluation test for each of the products.

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.
7F, 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