To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

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

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

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.

Introducing I/O Script Sample Programs

1. Outline of I/O Script Sample Programs

I/O script sample programs necessary to simulate the internal devices of the M16C/60 series of microcomputers are included with Simulator Debugger. These sample programs are listed below:

1. Timer sample
2. AD conversion sample
3. Serial I/O sample
4. CRC arithmetic circuit sample
5. DMAC sample
6. MR30 timer sample

These sample programs are detailed in a section, "Details of I/O Script Sample Programs."

2. Method for Using I/O Script Sample Programs

Before the I/O script sample programs can be used, they must first be set up in Simulator Debugger. Please follow the procedure described below to set up.
Note that the I/O script sample programs are stored in the directory shown below:

Install directory

¥Tools¥Renesas¥DebugComp¥Platform¥PDTarget¥PD30SIM¥Samples

1. Start up Simulator Debugger and open the I/O Timing Setting window.
2. Choose LOAD menu from the I/O Timing Setting window. When this menu is selected, a file selection dialog box appears.
3. In this dialog box, choose the I/O script sample programs you want to be set up in Simulator Debugger.

3. Details of I/O Script Sample Programs

3.1. Timer A0 (Timer Mode) Sample Program (timerA0_TMode.ios)

This sample program simulates the operation of timer A0 in timer mode.

[Timer function]

Count source: Number of cycles is counted.
Pulse output function: Not available
Gate function: Not available

[Operation]

1. When the count start flag is set to 1, the content of timer A0 register is counted down every 16 cycles, 16 counts at a time.
2. When the timer underflows (counted down below 0), the content of the reload register is reloaded into the timer A0 register and the timer starts counting over again.
   At the same time, the timer A0 interrupt request bit is set to 1.
3. When the count start flag is set to 0, the timer stops counting down, holding the count in progress when stopped.
4. When the interrupt is accepted, the timer A0 interrupt request bit is cleared to 0.

[Differences with the actual chip]

1. When the value of the timer A0 register is changed in a program, the new value is written to the timer A0 register and reload register immediately.
   -> In the actual chip, the new value is held in the reload register without being written to the timer A0 register immediately.
2. Even when the timer A0 interrupt request bit is cleared to 0 in a program, the timer interrupt request cannot be canceled.
3. The count source is fixed to f1 (divide-by-1 clock)

3.2. Timer A0(Timer Mode, Gate Function Selected) Sample Program (timerA0_TMod_Gate.ios)

This sample program simulates the operation of timer A0 in timer mode (gate selected).

[Timer function]

Count source: Number of cycles is counted.
Pulse output function: Not available
Gate function: Counted only when the TA0IN pin level is high (= 1)

[Operation]

1. When the count start flag is 1 and the input level on the TA0IN pin is high (= 1), the content of the timer A0 register is counted down every 16 cycles, 16 counts at a time.
2. When the input level on the TA0IN pin is low (= 0), the timer stops counting down, holding the count in progress when stopped.
3. When the timer underflows (counted down below 0), the content of the reload register is reloaded into the timer A0 register and the timer starts counting over again.
   At the same time, the timer A0 interrupt request bit is set to 1.
4. When the count start flag is set to 0, the timer stops counting down, holding the count in progress when stopped.
5. When the interrupt is accepted, the timer A0 interrupt request bit is cleared to 0.

[Method for creating TA0IN pin input signal]

To create the input signal, modify the following part of I/O script in the timerA0_TMod_Gate.ios file. In this example, the input signal to the TA0IN pin is switched between high and low every 10,000 cycles.

```
{
  while(1){
    set [0x3ed] = [0x3ed] & 0xfd    ; Pulls the TA0IN pin low.
    waitc 10000                     ; Holds the TA0IN pin low for 10000 cycles.
    set [0x3ed] = [0x3ed] | 0x02    ; Drives the TA0IN pin high.
    waitc 10000             ; Holds the TA0IN pin high for 10000 cycles.
  }
}
```

[Differences with the actual chip]

1. When the value of the timer A0 register is changed in a program, the new value is written to the timer A0 register and reload register immediately.
   -> In the actual chip, the new value is held in the reload register without being written to the timer A0 register immediately.
2. Even when the timer A0 interrupt request bit is cleared to 0 in a program, the timer interrupt request cannot be canceled.
3. The count source is fixed to f1 (divide-by-1 clock)

3.3. Timer A0 (Timer Mode, Pulse Output Function Selected) Sample Program (timerA0_TMod_Pulse.ios)

This sample program simulates the operation of timer A0 in timer mode (pulse output function selected).

[Timer function]

Count source: Number of cycles is counted.
Pulse output function: Available
Gate function: Not available

[Operation]

1. When the count start flag is set to 1, the content of timer A0 register is counted down every 16 cycles, 16 counts at a time.
2. When the timer underflows (counted down below 0), the content of the reload register is reloaded into the timer A0 register and the timer starts counting over again.
   At the same time, the timer A0 interrupt request bit is set to 1. Also, the output polarity of the TA0OUT pin is inverted.
3. When the count start flag is set to 0, the timer stops counting down, holding the count in progress when stopped. Also, the TA0OUT pin outputs a low (= 0).
4. When the interrupt is accepted, the timer A0 interrupt request bit is cleared to 0.

[Differences with the actual chip]

1. When the value of the timer A0 register is changed in a program, the new value is written to the timer A0 register and reload register immediately.
   -> In the actual chip, the new value is held in the reload register without being written to the timer A0 register immediately.
2. Even when the timer A0 interrupt request bit is cleared to 0 in a program, the timer interrupt request cannot be canceled.
3. The count source is fixed to f1 (divide-by-1 clock)

[Caution]

Changes in the output of the TA0OUT pin cannot be referenced using the I/O Timing Setting window's virtual port output function.

3.4. Timer A0 (One-shot Timer Mode) Sample Program (timerA0_OneShotTM.ios)

This sample program simulates the operation of timer A0 in one-shot timer mode.

[Timer function]

Count source: Number of cycles is counted.
Pulse output function: Not available
Gate function: One-shot start flag is set by writing a 1

[Operation]

1. When the one-shot start flag is set to 1 while the count start flag is 1, the content of timer A0 register is counted down every 16 cycles, 16 counts at a time.
2. When the timer underflows (counted down below 0), the content of the reload register is reloaded into the timer A0 register and the timer stops counting.
   At the same time, the timer A0 interrupt request bit is set to 1.
3. When the one-shot start flag is set to 1 by writing a 1 while counting, the value of the reload register is reloaded into the timer and the timer starts counting again.
4. When the count start flag is set to 0, the timer stops counting down and the content of the reload register is reloaded into the timer.
5. When the interrupt is accepted, the timer A0 interrupt request bit is cleared to 0.

[Differences with the actual chip]

1. When the value of the timer A0 register is changed in a program, the new value is written to the timer A0 register and reload register immediately.
   -> In the actual chip, the new value is held in the reload register without being written to the timer A0 register immediately.
2. Even when the timer A0 interrupt request bit is cleared to 0 in a program, the timer interrupt request cannot be canceled.
3. The count source is fixed to f1 (divide-by-1 clock)

3.5.A-D Converter (One-shot Mode) Sample Program (AD.ios)

This sample program simulates the operation of the A-D converter in one-shot mode. Note that in this sample program, input to A-D register 0 is simulated.

[A-D converter in Simulator Debugger (Differences with the actual chip)]

Simulator Debugger does not support simulation of analog input. For this reason, the functions of the A-D converter are implemented artificially by entering the A-D converted input value into the A-D register.

[Operation]

1. When the A-D conversion start flag is set to 1, an A-D conversion interrupt is generated. At the same time, data is input to the A-D register.
2. When the above is done, the A-D converter interrupt request bit is set to 1. Also, the A-D conversion start flag is set to 0, causing the A-D converter to stop operating.
3. When the interrupt is accepted, the A-D converter interrupt request bit is cleared to 0.
4. After that, when the A-D conversion start flag is set to 1 again, operations 1 to 3 above are repeated until no more input data exists in the A-D register.

[Method for creating A-D input data]

To create the A-D input data, modify the following part of I/O script in the AD.ios file:

```
{
  ; Sets AD data.
  ; Inputs data in order of 0x0, 0x1, and 0x2 each time an A-D conversion
  ; interrupt is generated.

  set #isint:14, [0x3c0] = 0x0, 0x1, 0x2, 0x3, -> Define the input data here.

  set %data_exist = 0       ; Finishes entering AD data.
}
```

[Caution]

Even when the A-D converter interrupt request bit is cleared to 0 in a program, the A-D converter interrupt request cannot be canceled.

3.6. Serial I/O (Receive) Sample Program (SIO.ios)

This sample program simulates serial I/O (receive) operation. Note that in this sample program, input to UART0 is simulated.

[Serial I/O in sample program (Differences with the actual chip)]

In the sample program, serial I/O operation is simulated in a simple way by entering data for serial I/O input directly into the UART receive buffer instead of latching the input signal from RxD.

Note that the sample program can be used in both clock-synchronous and clock-asynchronous serial modes. However, the receive data in the clock-asynchronous serial mode is 8 bits long.

[Operation]

1. When the receive enable bit is set to 1, a UART0 receive interrupt is generated. At the same time, data is input to the UART0 receive buffer.
2. When the above is done, the receive complete flag and the UART0 interrupt request bit are set to 1.
3. The receive complete flag is cleared to 0 when the lower byte of the UART0 receive buffer register is read out.
4. When the interrupt is accepted, the UART0 interrupt request bit is cleared to 0.
5. After that, operations 1 to 4 above are performed every 10,000 cycles until no more data exists in the UART0 receive buffer.

[Method for creating receive data]

To create the receive data, modify the following part of I/O script in the SIO.ios file:

```
{
  ; Sets receive data.
  ; Inputs data in order of 0x0, 0x1, and 0x2 each time a UART0 interrupt is
  ; generated.

  set #isint:18, [0x3a6] = 0x0, 0x1, 0x2, 0x3, -> Define the receive data here.

  set %data_exist = 0       ; Finishes entering receive data.
}
```

[Method for changing receive data input timing]

To change the receive data input timing, modify the following part of I/O script in the SIO.ios file. In the sample program, data is input every 10,000 cycles.

```
if(%data_exist == 1){       ; Checks to see if receive data exists.
        waitc 10000   -> Change this part to modify the sample program so
                         that data can be input at any time you want.
```

[Caution]

1. Even when the UART0 interrupt request bit is cleared to 0 in a program, the UART0 interrupt request cannot be canceled.
2. For error detection, only an overrun error can be detected.

3.7. CRC Arithmetic Circuit Sample Program (CRC.ios)

This sample program simulates the operation of the CRC arithmetic circuit.

[Operation]

1. The initial value 0 is set in the CRC data register.
2. When 1-byte data is written to the CRC input register, CRC code is generated in the CRC data register according to the said written data and the content of the CRC register.
3. If you want CRC for consecutive bytes to be calculated, write the next data to the CRC input register after CRC code is generated above.
4. The content of the CRC data register after all data have been written into it constitutes the CRC code.

[Differences with the actual chip]

1. In the actual chip, CRC code generation requires two machine cycles; in the sample program, no cycles are required (not added).

3.8. DMAC Sample Program (DMAC.ios)

This sample program simulates the operation of repeat transfers by DMAC. The operation of both DMA0 and DMA1 can be simulated.

[Functions of DMA0 and DMA1]

Transfer space: Following three

(1) From selected 1-Mbyte space to fixed address
(2) From fixed address to selected 1-Mbyte space
(3) From fixed address to fixed address

Cause of DMA transfer: Timer A0

Channel priority: If DMA0 transfer request and DMA1 transfer request occur simultaneously, DMA0 transfer is given priority.

Transfer mode: Repeat transfer only

Unit of transfer: Transferred in 8 bits or 16 bits

[Operation]

1. If a timer A0 interrupt is generated when the DMA enable bit = 1, a DMA request can be accepted.
2. When the DMA request is accepted, DMA transfers are performed in a specified transfer space in a specified unit of transfer.
3. Even when the DMA0/1 transfer counter underflows, the DMA enable bit remains set. The DMA0/1 interrupt request bit is set to 1 when the DMA0/1 transfer counter underflows.
4. When the DMA0/1 interrupt is accepted, the DMA0/1 interrupt request bit is cleared to 0.
5. If a second DMA request occurs after the DMA0/1 transfer counter underflows, control returns to 1, from which DMA transfers are repeated.

[Differences with the actual chip]

1. Even when the DMA0/1 interrupt request bit is cleared to 0 in a program, the DMA0/1 interrupt request cannot be canceled.
2. For the transfer mode, you can specify only the repeat transfer.
3. Since the DMA request bit is not simulated, a DMA request is always accepted if a timer A0 interrupt occurs when the DMA enable bit = 1.

[Method for modifying program to use other interrupt as the cause of DMA transfer]

Here, explanation is made by assuming the use of timer A1 as an example. Modify following two parts of the DMA.ios file:

-Location to change for DMA0

```
if(%dma0_start == 1){
        pass #isint:21, 1    -> Change this to pass#isint:22,1.
```

-Location to change for DMA1

```
if(%dma1_start == 1){
        pass #isint:21, 1    -> Change this to pass#isint:22,1.
```

Specify the interrupt vector number you want to be the cause of DMA transfer by using a #isint statement as shown above.

3.9. MR30 Timer Sample Program (MR.ios)

This sample program simulates the operation of timer A0 when using it in the MR30.

For the application programs that use MR30 to be executed in Simulator Debugger, the following setting is required.
Since MR30 sets the system clock that is referenced by the cyclic handler, alarm handler, get_tim system call, and dly_tsk system call, specify in the configuration file the timer used by MR30 and the time interval at which timer interrupts are generated.

```
clock{
        mpu_clock       = 10MHz;
        timer           = A0;
        IPL             = 4;
        unit_time       = 100ms;        // ms
        initial_time    = 0:0:0;
};
```

In the example above, timer A0 is used for timer interrupts to the system, and the interval at which timer A0 interrupts are generated is set to 100 ms.
For the application programs that use MR30 to be executed in Simulator Debugger, the timer used by MR30 (e.g., timer A0 in this case) must be set.
The following shows a sample program (mr.ios) for this setup:

```
; Example of timer A0 definition
{
    while(1){                                   -> while statement
        if( ([0x380].b & 0x01) == 0x01){        -> Checks timer A0 count start flag.
            waitc [0x386].w + 1                 -> Keeps I/O script execution waiting for the
                                                   number of cycles equal to the divide-by ratio set in
                                                   timer A0.
            int 21, [0x55] & 0x7                -> Generates timer A0 virtual interrupt.
                                                    (For priority, refer to the interrupt control register.)
        }else{
            waiti 100                           -> Keeps I/O script execution waiting for 100
                                                   instructions.
        }
    }
}
```

By registering this I/O script in Simulator Debugger from the I/O Timing Setting window menus [Load], you can simulate the MR30 applications.

[Method for modifying program to use other timer]

Here, explanation is made by assuming the use of timer A3 as an example.

-Timer A0 sample program

```
{
        while(1){
          if( ([0x380].b & 0x01) == 0x01){
            waitc [0x386].w + 1
             int 21, [0x55] & 0x07
          }else{
            waiti 100
          }
        }
}
```

Modify the parts shown below so that timer A3 can be used.

-Timer A3 sample program

```
{
        while(1){
          if( ([0x380].b & 0x08) == 0x08){   -> Changed to reference timer A3 count start flag.
            waitc [0x38C].w + 1              -> Changed to reference the divide-by ratio of timer A0
                                                register.
             int 21, [0x58] & 0x07           -> Changed to reference timer A3 interrupt control
                                                register's interrupt priority select bit.
          }else{
            waiti 100
          }
        }
}
```