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.

H8/38024F

Example Code for H8/38024F Peripherals

Introduction

This application note provides example C code to initialise and configure Timer A, LCD controller and shows how to produce an IIC interface using a standard IO port.

Timer A is configured to be in clock time base operation. In this example an interrupt is generated by this peripheral every second. The LCD initialisation code shows how to setup the LCD controller to control and output data for a Trident 7 segment display or equivalent, part number VIM 838. The next example uses a combination of the timer A code and the LCD initialisation, using these it is possible to generate code that produces a software real time clock and outputs the time on the 7-segment display. The remaining examples are based around the idea of implementing an IIC interface in software using a standard IO port. The first example shows software example to generate the IIC waveforms on the IO port. The following examples then use the IIC software to produce code that can communicate with an external EEPROM and RTC. The RTC example also displays the clock information on the aforementioned display.

Contents

1. H8/38024F Introduction

The H8/38024F microcontroller from Renesas is the first of a new series of single chip SLP devices with FLASH on board. This device is clocked at a maximum of 10 MHz and features 32 k of FLASH and 1 k of RAM. The peripherals include 3x8-bit timers, 1x16-bit timer, 1 watchdog timer, 1 x Asynchronous event counter, 1 serial communications channel (which can be synchronous or asynchronous), 8x10-bit ADC channels, LCD controller, 10-bit PWM, 32 kHz sub-clock operation and a host of internal and external interrupts. The 38024F can be obtained in three sizes of QFP package. The H8/38024F hardware manual, available from www.renesas.com should be consulted for further details on the device and its peripherals.

2. Development Environment

The E10T is a low cost emulation tool for the H8/38024F. It is important when developing with the E10T not to set the whole of port 3 to be an output port as this will interfere with the debug interface on pins 3, 4 and 5 of Port 3. The HDI (Hitachi Debugging Interface) provides a Windows ® based debugging enviroment for the EDK allowing code to be downloaded, run, stepped and examined easily. The HEW (High Performance Embedded Workshop) provides a GUI for the compiler and is also a project management tool.

Configuring HEW

Before starting a new workspace and project in HEW the user needs to add map files and include files so that HEW will ‘recognise’ the H8/38024F. The files in Appendix A (*.seg) should be copied into the 300 folder (path given below).

C:\…….\HEW\System\Pg\IAR\ICCH8300\System\Hew-Maps\300

The file in Appendix B (Ioh838024.h) should be copied into the inc folder (path given below).

C:\…….\HEW\Tools\IAR\ICCH8300\v4_20c\inc

Now, when a new project is started the 38024 should appear as a CPU option.

3. Device Block Diagram

This application note provides example C code to initialise and configure Timer A, LCD controller and shows how to produce an IIC interface using a standard IO port, shown in the block diagram below.

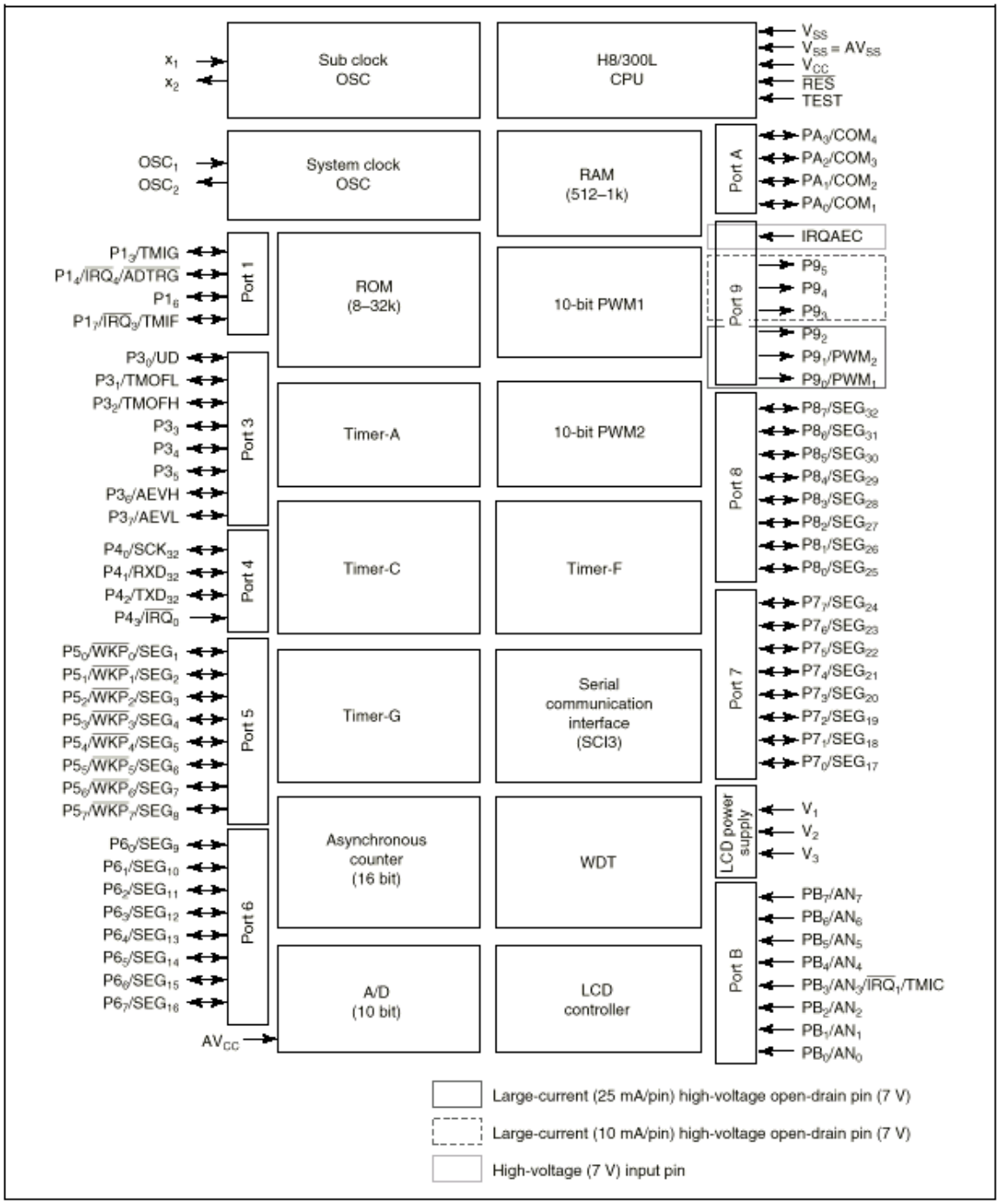

Figure 3.1 Device Block Diagram

4. Timer A

Timer A - Clock Time Base Operation

When bit TMA3 in TMA is set to 1, timer A functions as a clock time base by counting clock signals output by prescaler W. The overflow period of timer A is set by bits TMA1 and TMA0 in TMA. A choice of four periods (1 s, 0.5 s, 0.25 s, 31.25 ms) are available. In time base operation (TMA3 = 1), setting bit TMA2 to 1 clears both TCA and prescaler W to their initial values of H'00.

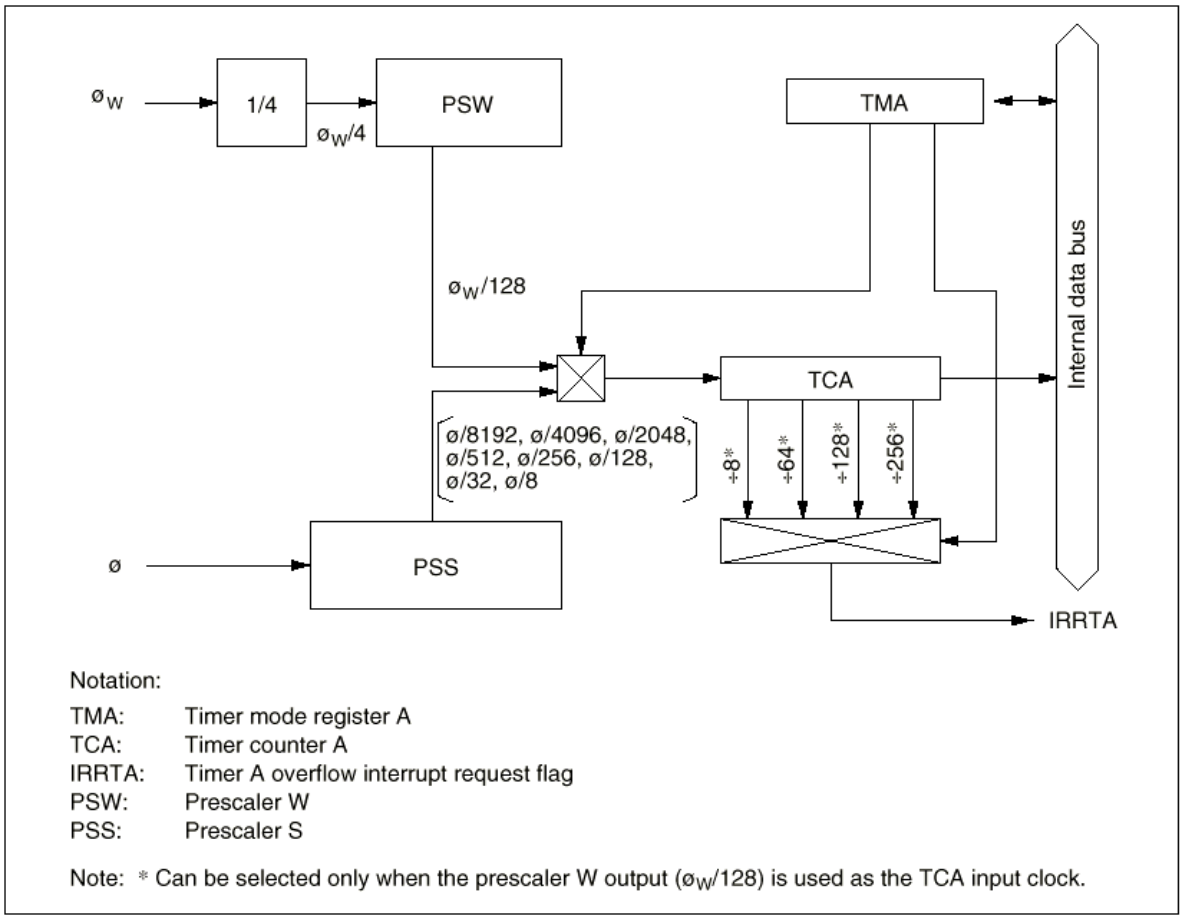

Figure 4.1 Block Diagram of Timer A

The code below gives an example of Timer A used in clock time base operation. An interrupt is requested when timer A overflows. The interrupt service routine clears the interrupt request register, re-sets the counter and toggles two high current output pins (10 mA/pin).

To do this Port 9 is initialised so that pins 3 and 4 (P93 & P94) are set as outputs. These pins are set up with a low o/p.

Then Timer A is initialised, the ‘default’ of the main clock divided by 32 is not changed (it will not actually output on a pin as the TMOW bit in PMR1 would need to be set to 1). The timer is set to give a clock time base of 1 second. Please note that to use this Timer function a 32.768 kHz clock is needed.

Interrupts are then set so that an interrupt will be generated every second. The interrupt service routine clears the interrupt request register, clears the timer counter to 0 and toggles the output pins on Port 9 to make two LED flash.

Code Example

```
/*
**---------------------------------------------------------------------
**
**      main.c - contains C entry point main()
**
**  This file was generated by HEW IAR Icch8 project generator
**
**---------------------------------------------------------------------
*/


#include <inh8300.h>
#include <ioh838024.h>

#pragma language=extended

void init_TimerA(void);
void init_Port9(void);
void main(void);


void main(void)
{
        init_lcd();
        init_Port9();
        init_TimerA();

        set_interrupt_mask(0);          // enable interrupts

        while(1);
}


void init_TimerA (void)
{
     /* set up timer A for a clock time base of 1 second */

     TMA |= 0x18;
     IENR1 |= 0x80;
}


void init_Port9 (void)
{
      PMR9 |= 0x18;
      PDR9 &= 0x04;
}


interrupt [TIMER_A] void interval (void)
{
        IRR1 &= ~(0x80); /* clear the interrupt request register */
        TCA &= 0x00;/*clear timer counter */
        TMA |= 0x18;
        PDR9 ^= 0x18;       /* toggle the output pin */
}
```

5. LCD Code

The LCD Controller/Driver is set up for 1/3 duty cycle (COM1-3) and SEG 9 to 32 are set up as LCD segment. The LCD supply is turned on, the LCD controller/driver operates and the LCD RAM data is displayed. The Operating clock is set to ϕ/32 which gives a frame frequency of 650.7 Hz (given that ϕ is 8 MHz and the duty ratio is 1/3). Lastly waveform A is used as the LCD drive waveform. As a precaution we clear LCD module standby mode in the module stop register which is the default on after device reset.

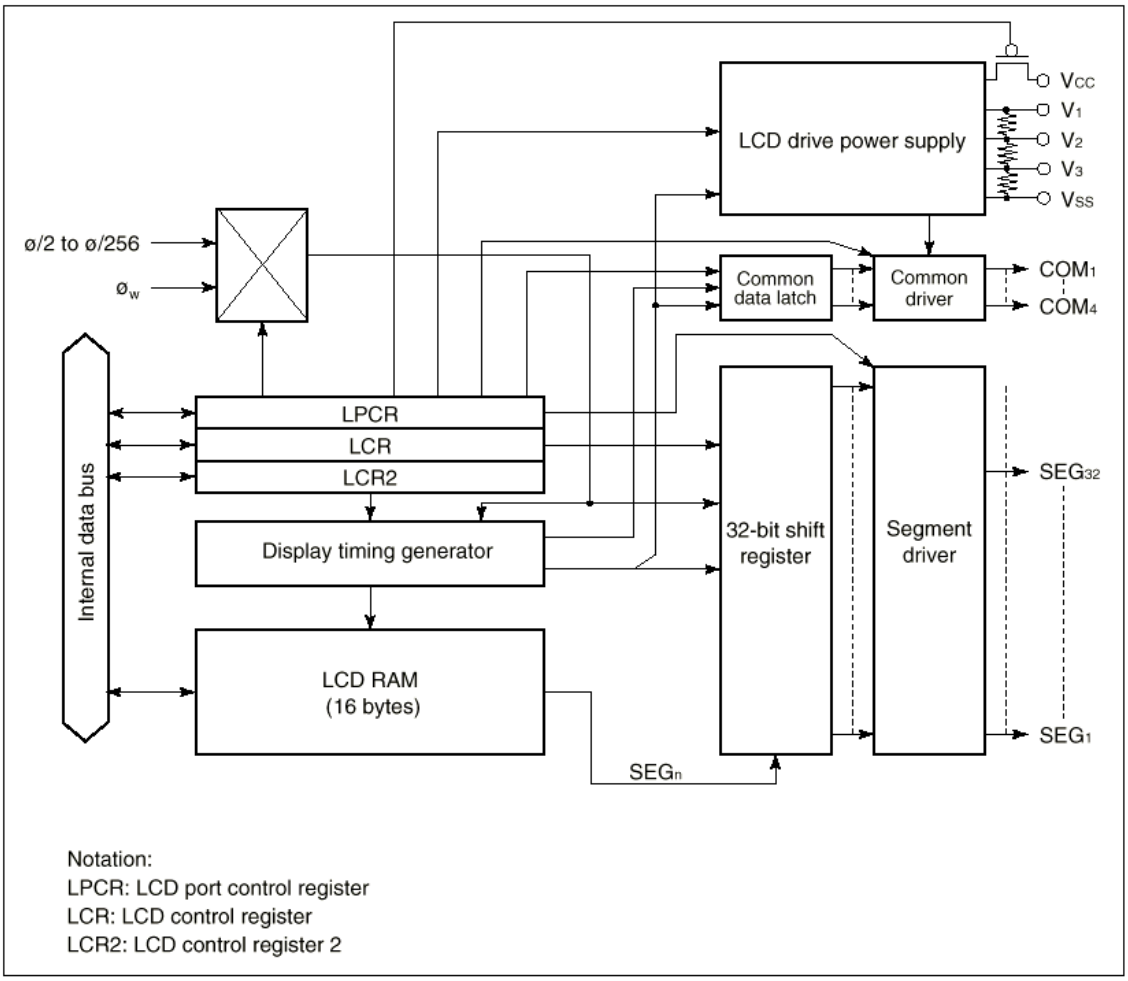

Figure 5.1 Block Diagram of LCD Controller/Driver

Timer A is then set up to interrupt every second. In the interrupt routine the LCD displays 0 to 9 depending on the value of ‘temp’.

Display used is a numeric display from Trident – part no. VIM-838.

Code Example

```
/*
**---------------------------------------------------------------------
**
**      main.c - contains C entry point main()
**
**  This file was generated by HEW IAR Icch8 project generator
**
**---------------------------------------------------------------------
*/



#include <inh8300.h>
#include <ioh838024.h>
#include "lcd.h"
#pragma language=extended

void init_TimerA(void);
void init_lcd(void);
void init_Port9(void);
void main(void);

unsigned char temp=0;
unsigned int read_digit, i, j = 0;

#pragma memory = dataseg(LCD_RAM)
unsigned char digit_two_one[3];
unsigned char digit_four_three[3];
unsigned char digit_six_five[3];
unsigned char digit_eight_seven[3];
#pragma memory = default

const unsigned char l_blank[]    = {0x00, 0xF0, 0xFF};
const unsigned char l_zero[]     = {0x53, 0xF3, 0xFF};      //(_,1d,1g,1a)(_,1DP,1c,1b) (EMPTY)
(_,1AN,1e,1f) (EMPTY) (EMPTY)
const unsigned char l_one[]      = {0x03, 0xF0, 0xFF};
const unsigned char l_two[]      = {0x71, 0xF2, 0xFF};
const unsigned char l_three[]    = {0x73, 0xF0, 0xFF};
const unsigned char l_four[]     = {0x23, 0xF1, 0xFF};
const unsigned char l_five[]     = {0x72, 0xF1, 0xFF};
const unsigned char l_six[]      = {0x72, 0xF3, 0xFF};
const unsigned char l_seven[]    = {0x13, 0xF1, 0xFF};
const unsigned char l_eight[]    = {0x73, 0xF3, 0xFF};
const unsigned char l_nine[]     = {0x73, 0xF1, 0xFF};

const unsigned char u_blank[]    = {0xFF, 0x0F, 0x00};
const unsigned char u_zero[]     = {0xFF, 0x3F, 0x35};      // (EMPTY) (EMPTY) (_,2DP,2c,2b) (EMPTY)
(_,2AN,2e,2f) (_,2d,2g,2a)
const unsigned char u_one[]      = {0xFF, 0x3F, 0x00};
const unsigned char u_two[]      = {0xFF, 0x1F, 0x27};
const unsigned char u_three[]    = {0xFF, 0x3F, 0x07};
const unsigned char u_four[]     = {0xFF, 0x3F, 0x12};
const unsigned char u_five[]     = {0xFF, 0x2F, 0x17};
const unsigned char u_six[]      = {0xFF, 0x2F, 0x37};
const unsigned char u_seven[]    = {0xFF, 0x3F, 0x11};
const unsigned char u_eight[]    = {0xFF, 0x3F, 0x37};
const unsigned char u_nine[]     = {0xFF, 0x3F, 0x17};

void main(void)
{
       init_lcd();
       init_Port9();
       init_TimerA();

       set_interrupt_mask(0);     // enable interrupts
```

```
        while(1);
}



void init_TimerA (void)
{
     /* set up timer A for a clock time base of 1 second */

         TMA |= 0x18;
     IENR1 |= 0x80;
}


void init_Port9 (void)
{
PMR9 |= 0x18;
PDR9 &= 0x04;
}


void init_lcd(void)
{
   LPCR     |= 0x8A;   /* 1/3 duty cycle (COM1-3), SEG 9 to 32 are LCD segment IOs */
   LCR      |= 0xFC;   /* turn on LCD supply, cont/driver + disp ram data, clk=sys/32 (244Hz with
4MHz clk) */
   LCR2     &= 0x80;   /* use waveform A */
   CHSTPR2  |= 0x1;    /* clear LCD module standby mode in module stop register*/
   SEG10_9  = (0x00);
   SEG12_11 = (0x00);
   SEG14_13 = (0x00);
   SEG16_15 = (0x00);
   SEG18_17 = (0x00);
   SEG20_19 = (0x00);
   SEG22_21 = (0x00);
   SEG24_23 = (0x00);
   SEG26_25 = (0x00);
   SEG28_27 = (0x00);
   SEG30_29 = (0x00);
   SEG32_31 = (0x00);

}

interrupt [TIMER_A] void interval (void)
{
        IRR1 &= ~(0x80);          /* clear the interrupt request register */
        TCA &= 0x00;              /*clear timer counter */
        TMA |= 0x18;
        PDR9 ^= 0x18;
        if (temp == 0)
                  {
                                  digit_two_one[0] = (l_zero[0] & u_zero[0]);
                                  digit_two_one[1] = (l_zero[1] & u_zero[1]);
                                  digit_two_one[2] = (l_zero[2] & u_zero[2]);
                  }
        else if (temp == 1)
                  {
                                  digit_two_one[0] &= (l_one[0]);
                                  digit_two_one[1] &= (l_one[1]);
                                  digit_two_one[2] &= (l_one[2]);
                  }
        else if (temp == 2)
                  {
                                  digit_two_one[0] = (l_two[0] & u_two[0]);
                                  digit_two_one[1] = (l_two[1] & u_two[1]);
                                  digit_two_one[2] = (l_two[2] & u_two[2]);

                  }
        else if (temp == 3)
                  {
```

```
                                digit_two_one[0] = (l_three[0] & u_three[0]);
                                digit_two_one[1] = (l_three[1] & u_three[1]);
                                digit_two_one[2] = (l_three[2] & u_three[2]);
                        }
        else if (temp == 4)
                        {
                                digit_two_one[0] = (l_four[0] & u_four[0]);
                                digit_two_one[1] = (l_four[1] & u_four[1]);
                                digit_two_one[2] = (l_four[2] & u_four[2]);
                        }
        else if (temp == 5)
                        {
                                digit_two_one[0] = (l_five[0] & u_five[0]);
                                digit_two_one[1] = (l_five[1] & u_five[1]);
                                digit_two_one[2] = (l_five[2] & u_five[2]);
                        }
        else if (temp == 6)
                        {
                                digit_two_one[0] = (l_six[0] & u_six[0]);
                                digit_two_one[1] = (l_six[1] & u_six[1]);
                                digit_two_one[2] = (l_six[2] & u_six[2]);
                        }
        else if (temp == 7)
                        {
                                digit_two_one[0] = (l_seven[0] & u_seven[0]);
                                digit_two_one[1] = (l_seven[1] & u_seven[1]);
                                digit_two_one[2] = (l_seven[2] & u_seven[2]);
                        }
        else if (temp == 8)
                        {
                                digit_two_one[0] = (l_eight[0] & u_eight[0]);
                                digit_two_one[1] = (l_eight[1] & u_eight[1]);
                                digit_two_one[2] = (l_eight[2] & u_eight[2]);
                        }
        else if (temp == 9)
                        {
                                digit_two_one[0] = (l_nine[0] & u_nine[0]);
                                digit_two_one[1] = (l_nine[1] & u_nine[1]);
                                digit_two_one[2] = (l_nine[2] & u_nine[2]);
                                temp = 0xff;
                        }
temp++;
}
```

6. RTC using Timer A

The example code below utilises the one second interrupt of Timer A to provide a software RTC. The time is then displayed on the LCD.

Code Example

```
**---------------------------------------------------------------------
**
**      main.c - contains C entry point main()
**
**  This file was generated by HEW IAR Icch8 project generator
**
**---------------------------------------------------------------------
*/


#include <inh8300.h>
#include <ioh838024.h>
#include "lcd.h"
#pragma language=extended

void init_TimerA(void);
void init_lcd(void);
void init_Port9(void);
void main(void);


const unsigned char clock_lookup[60][3] = {
  {0x57,0x33,0x35},   {0x07,0x30,0x35}, {0x75,0x32,0x35},   {0x77,0x30,0x35}, {0x27,0x31,0x35},
  {0x76,0x31,0x35},   {0x76,0x33,0x35}, {0x17,0x31,0x35},   {0x77,0x33,0x35}, {0x77,0x31,0x35},
  {0x57,0x33,0x00},   {0x07,0x30,0x00}, {0x75,0x32,0x00},   {0x77,0x30,0x00}, {0x27,0x31,0x00},
  {0x76,0x31,0x00},   {0x76,0x33,0x00}, {0x17,0x31,0x00},   {0x77,0x33,0x00}, {0x77,0x31,0x00},
  {0x57,0x13,0x27},   {0x07,0x10,0x27}, {0x75,0x12,0x27},   {0x77,0x10,0x27}, {0x27,0x11,0x27},
  {0x76,0x11,0x27},   {0x76,0x13,0x27}, {0x17,0x11,0x27},   {0x77,0x13,0x27}, {0x77,0x11,0x27},
  {0x57,0x33,0x07},   {0x07,0x30,0x07}, {0x75,0x32,0x07},   {0x77,0x30,0x07}, {0x27,0x31,0x07},
  {0x76,0x31,0x07},   {0x76,0x33,0x07}, {0x17,0x31,0x07},   {0x77,0x33,0x07}, {0x77,0x31,0x07},
  {0x57,0x33,0x12},   {0x07,0x30,0x12}, {0x75,0x32,0x12},   {0x77,0x30,0x12}, {0x27,0x31,0x12},
  {0x76,0x31,0x12},   {0x76,0x33,0x12}, {0x17,0x31,0x12},   {0x77,0x33,0x12}, {0x77,0x31,0x12},
  {0x57,0x23,0x17},   {0x07,0x20,0x17}, {0x75,0x22,0x17},   {0x77,0x20,0x17}, {0x27,0x21,0x17},
  {0x76,0x21,0x17},   {0x76,0x23,0x17}, {0x17,0x21,0x17},   {0x77,0x23,0x17}, {0x77,0x21,0x17}
};

/*
const unsigned char clock_lookup[60][3] = {
  {zero_zero[]},    {zero_one[]},     {zero_two[]},       {zero_three[]},       {zero_four[]},
  {zero_five[]},    {zero_six[]},     {zero_seven[]},     {zero_eight[]},       {zero_nine[]},
  {one_zero[]},     {one_one[]},      {one_two[]},        {one_three[]},        {one_four[]},
  {one_five[]},     {one_six[]},      {one_seven[]},      {one_eight[]},        {one_nine[]},
  {two_zero[]},     {two_one[]},      {two_two[]},        {two_three[]},        {two_four[]},
  {two_five[]},     {two_six[]},      {two_seven[]},      {two_eight[]},        {two_nine[]},
  {three_zero[]},   {three_one[]},    {three_two[]},      {three_three[]},      {three_four[]},
  {three_five[]},   {three_six[]},    {three_seven[]},    {three_eight[]},      {three_nine[]},
  {four_zero[]},    {four_one[]},     {four_two[]},       {four_three[]},       {four_four[]},
  {four_five[]},    {four_six[]},     {four_seven[]},     {four_eight[]},       {four_nine[]},
  {five_zero[]},    {five_one[]},     {five_two[]},       {five_three[]},       {five_four[]},
  {five_five[]},    {five_six[]},     {five_seven[]},     {five_eight[]},       {five_nine[]}
};
*/

unsigned char seconds;
unsigned char minutes;
unsigned char hours;

#pragma memory = dataseg(LCD_RAM)
unsigned char digit_two_one[3];
```

```
unsigned char digit_four_three[3];
unsigned char digit_six_five[3];
unsigned char digit_eight_seven[3];
#pragma memory = default



void main(void)
{
   init_lcd();
   init_Port9();
   init_TimerA();

   set_interrupt_mask(0);// enable interrupts


   while(1)
   {
//      seconds
        digit_two_one[0] = clock_lookup[seconds][0];
        digit_two_one[1] = clock_lookup[seconds][1];
        digit_two_one[2] = clock_lookup[seconds][2];

//      minutes
        digit_four_three[0] = clock_lookup[minutes][0];
        digit_four_three[1] = clock_lookup[minutes][1];
        digit_four_three[2] = clock_lookup[minutes][2];

//      hours
        digit_six_five[0] = clock_lookup[hours][0];
        digit_six_five[1] = clock_lookup[hours][1];
        digit_six_five[2] = clock_lookup[hours][2];
   }
}



void init_TimerA (void)
{
/* set up timer A for a clock time base of 1 second */

   TMA |= 0x18;
IENR1 |= 0x80;
}


void init_Port9 (void)
{
PMR9 |= 0x18;
PDR9 &= 0x04;
}


void init_lcd(void)
{
     LPCR      |= 0x8A;     /* 1/3 duty cycle (COM1-3), SEG 9 to 32 are LCD segment IOs */
     LCR       |= 0xFC;     /* turn on LCD supply, cont/driver + disp ram data, clk=sys/32 (244Hz
with 4MHz clk) */
     LCR2      &= 0x80;     /* use waveform A */
     CHSTPR2   |= 0x1;      /* clear LCD module standby mode in module stop register*/
     SEG10_9   = (0x00);
     SEG12_11  = (0x00);
     SEG14_13  = (0x00);
     SEG16_15  = (0x00);
     SEG18_17  = (0x00);
     SEG20_19  = (0x00);
     SEG22_21  = (0x00);
     SEG24_23  = (0x00);
     SEG26_25  = (0x00);
     SEG28_27  = (0x00);
```

```
    SEG30_29   = (0x00);
    SEG32_31   = (0x00);
}

interrupt [TIMER_A] void interval (void)
{
    IRR1 &= ~(0x80);              /* clear the interrupt request register */
    TCA &= 0x00;                                              /*clear timer counter */
    TMA |= 0x18;
    PDR9 ^= 0x18;
    seconds++;
       if (seconds == 60)
       {
                          seconds = 0;
                          minutes ++;
                          if (minutes == 60)
                          {
                                              minutes = 0;
                                              hours ++;
                                              if (hours == 24)
                                              {
                                                           hours = 0;
                                              }
                          }
       }

}
```

7. Software Controlled IIC

The IIC bus is a low-cost, bi-directional two-wire serial bus. Only two bus signals are required: a Serial Data Line (SDA) and a Serial Clock Line (SCL). The IIC bus is a multi-master bus, although only one master can control the bus at a time through an arbitration process. Each slave device connected to the bus has its own unique address and can transmit or receive data.

This application note describes a software implementation of the IIC serial bus for the Renesas H8-series microcontrollers. In the examples, the H8 microcontroller acts as the sole master of the bus. Two of its I/O pins function as the I2C bus, and communication to the slave devices is controlled by software. The software device driver is written in C; however, knowledge of the I2C specification is not required in order to apply these C routines in applications.

For further information on hardware interfacing an IIC network, the IIC protocol and the IIC data format please refer to the 'The I^2C-Bus Specification' v2.1 (January 2000) from Philips Semiconductors.

There are two examples shown below which illustrate how to implement these C routines on the Renesas H8/38024. The first example shows communication between the H8 and a Holtek CMOS 16 K 2-Wire serial EEPROM HT24LC16. The second shows communication to a Philips low power clock/calendar, PCF8593.

The low level functions in this example have been written in a modular fashion. These are then called in the following examples to make the complete IIC interface. The following paragraphs discuss these functions in more detail.

Due to the nature of communication via the IIC interface, it is not possible to configure the IO port to one direction. This is because the data line is bi-directional; were the device to be used in slave mode the clock line would be input from an external device. However in these examples the H8 is always the master. To allow the data and clock lines to be bi-directional and ease the understanding of this code some defines have been made which convert the physical name to the relevant IO port or data value. These can be found in the header file iic.h. These examples use port 5 as the IO port for the data and clock lines, port 5 bit 6 is used for the data line and bit 5 for the clock. N.B. this file has defines set-up to use the on board pull-up resistors for this port, although these are included, the hardware used had external resistors and therefore these defines are not used in the examples beneath.

One common pitfall on the Renesas H8 microcontroller is that in general the port control register, which determines the operation (input or output) of the port, is a write only register. This means that read modify write instructions can not be used with this register, instead this example uses a shadow register. All operations can be performed on this shadow register and then the result can be written directly to the PCR, port control register. The first four functions of this example are relatively intuitive, these are sdain(), sclin(), sdaout() and sclout(). Functions sdain() and sclin() are used to read the current status of the data and clock lines, this sets them to inputs and returns whether the input is high or low. Functions sdaout() and sclout() are used to output the high or low

values onto the data and clock lines, here the ports are set to outputs and the level that is passed to the function is output on the corresponding line.

There are two delay functions; these delays could be implemented using a number of methods including using one of the many onboard timers that the H8/38024 has to offer, instead, to make this code more portable, these have been created in software using a while loop. The maximum clock frequency of the Philips low power clock is 100 KHz; this gives a clock period of 10us. This corresponds to a pulse width of 5us. So that it is possible to interface to slower IIC devices, this example has been written with a delay of approximately 6us. This can also vary within the code since in the sendstartbit() function has a clock period of around 48us. There is also a delay of 12us, which is simply a function which calls the 6us delay twice. If using this example to interface to other IIC devices it is recommended that these timing delays are re-visited and checked such that the timings match those of the new device.

Also included in this example is a function that checks the bus state of the IIC lines. This function can then be used before commencing communication on the IIC line to check that there is no traffic already on this line and that the buses are in their default state. The start bit of the IIC bus is defined as a high to low transition on the data line with the clock line high. The sendstartbit() function described here allows some compensation for the acknowledge bit if in some cases this bit is particularly slow, this should then reset the device into the default condition, this also sets up the buses so that the start condition can begin, i.e. sets both lines high. Once this is finished the data line is changed to a low state and the start condition has been completed, however this function also sets the clock line low to leave the bus in known condition.

The next two functions are for sending and receiving bits. The send bit has a small delay and then outputs onto the data line the value that is passed to it as the data bit to be sent. After another small delay the clock line is pulled high and tested. To receive the bit the data and clock lines are set high, the clock line is tested to ensure that it is high and then the data line is read. The value found on this line is then returned. There is also a get acknowledge bit function, getack(), which is the same as the function for receiving a bit on the data line.

Once we have the above functions we can build on these to create the send byte and receive byte functions. The send byte function is passed a byte that is to be sent on to the IIC data line, the IIC format is such that the MSB is sent first. To do this we can test the bit that is to be sent, then call the send bit function with the bit value. When all of the bits are sent we can wait for the acknowledge. The bytes are also received in the same way. The send stop bit function then terminates the transmission of the last byte and issues the stop command which is a low to high transition of the data line while the clock is high. Also included in the source code is some example software on how to use the above functions to create an IIC read and write functions. These can be used to perform sequential read and writing by passing the slave address, a pointer to the data (stored or to be written) and the number of bytes to be sent. This has not been included since the EEPROM and RTC code examples use other methods to pass data to and call the above functions.

Code Example

```
/*
        FILE:           IIC.C
        DATE:           26-6-98

                        DESCRIPTION:       IIC driver routines
```

```
*/

/* Software controlled IIC approch used to implement the IIC protocol */
/* H8 is the only master on the network */
/* To act as an IIC master, 5 steps are needed to Rx or Tx data */
/* 1.   poll the bus to make sure bus is idle */
/* 2.   generate START bit */
/* 3.   send out the slave address and wait for ACK */
/* 4.   TX or Rx data byte(s) */
/* 5.   generate STOP bit */

/* IO ports used for SCL and SDA are set up in iic.h */


#include "iic.h"

volatile unsigned char dummy_iic_port_ddr;        /* value used to store DDR value as */
                                                  /* DDR is a write only reg and cannot be read */

unsigned char sclin (void)
{

        /* check SCL signal level */

        dummy_iic_port_ddr &= SCL_IO_RESET_BIT;
        SCL_IO_REG = dummy_iic_port_ddr;                /* make SCL an input */
//      SCL_PULL_UP |= SCL_PULL_UP_ON;                  /* turn the internal pull up on     */

        if (SCL_DATA_REG & SCL_DATA_SET_BIT)
                        return HIGH;
        else
                        return LOW;
}

unsigned char sdain (void)
{
        /* check SDA signal level */

        dummy_iic_port_ddr &= SDA_IO_RESET_BIT;
        SDA_IO_REG = dummy_iic_port_ddr;          /* make SDA an input */
//      SDA_PULL_UP |= SDA_PULL_UP_ON;            /* turn the internal pull up on     */

        if (SDA_DATA_REG & SDA_DATA_SET_BIT)
                        return HIGH;
        else
                        return LOW;
}

void sclout (unsigned char status)
{
        /* drive the SCL bus */

        if (status == LOW)
        {
                        dummy_iic_port_ddr |= SCL_IO_SET_BIT;
                        SCL_IO_REG = dummy_iic_port_ddr; /* make SCL output */

                        SCL_DATA_REG &= SCL_DATA_RESET_BIT;     /* drive port LOW */
        }
        else
        {
                        /* port is an input, use external pullup resister to go high */
                        dummy_iic_port_ddr &= SCL_IO_RESET_BIT;
                        SCL_IO_REG = dummy_iic_port_ddr;
        }
}

void sdaout (unsigned char status)
{
```

```
        /* drive the SDA bus */

        if (status == LOW)
        {
                        dummy_iic_port_ddr |= SDA_IO_SET_BIT;
                        SDA_IO_REG = dummy_iic_port_ddr; /* make port an output */

                        SDA_DATA_REG &= SDA_DATA_RESET_BIT;
        }
        else
        {
                        /* port is input and using external pullup resistor to go high*/
                        dummy_iic_port_ddr &= SDA_IO_RESET_BIT;
                        SDA_IO_REG = dummy_iic_port_ddr; /* make port an input */
        }
}

void delay6us (void)
{
        /* function taking approx 6us to call     */
        /* H8/38024F @ 10MHz                                        */

        volatile unsigned char i;

        i=0;
        while (i<25)
        {
                        i++;
        }
}

void delay12us (void)
{
        delay6us();
        delay6us();
}

unsigned char checkbusstate (void)
{
        if ( (sclin() == HIGH) && (sdain() == HIGH) )
        {
                        return TRUE;
        }
        else
                        return FALSE;
}

void sendstartbit (void)
{
        sclout(LOW);         /* allow ACK to complete */
        delay12us();
        sdaout(HIGH);           /* release SDA (goes high) */
        delay12us();
        sclout(HIGH);
        delay12us();
        sdaout(LOW);
        delay12us();
        sclout(LOW);
}

void sendbit (unsigned char data_byte)
{
        /* it is beneficial to send out at the middle of the clock being low */

        sclout(LOW);
        delay6us();

        if (data_byte != 0)
                        sdaout(HIGH);
        else
```

```
                        sdaout (LOW);

        delay6us();
        sclout(HIGH);

        while (sclin() != HIGH);    /* wait for slow device to release clock */

        delay12us();
}

unsigned char getbit (void)
{
        /* it is beneficial to sample the data input at the middle of the clock being high */

        unsigned char temp;

        sclout(LOW);
        delay12us();
        sdaout(HIGH);
        delay12us();
        sclout(HIGH);

        while (sclin() != HIGH);    /* wait for slow device to release clock */

        delay6us();
        temp = sdain();
        delay6us();

        return temp;
}

unsigned char getack (void)
{
        /* similar to getbit, but this is a critical operation since the master must pull */
        /* SDA high before it finds out whether there is ACK (SDA low) or not */

        unsigned char temp;

        sclout(LOW);
        delay6us();
        sdaout(HIGH);
        delay6us();
        sclout(HIGH);

        while (sclin() != HIGH);    /* wait for slow device to release clock */

        delay6us();
        temp = sdain();
        delay6us();

        return temp;
}

unsigned char sendbyte (unsigned char data_byte)
{
        /* note that the master needs to get ACK after sending out every byte */

        unsigned char i;
        unsigned char mask;

        mask = 0x80;         /* send out msb first */

        for (i=0; i<8; i++)
        {
                        sendbit(data_byte & mask);
                        mask /= 2;
        }

        return getack();
}
```

```
unsigned char getbyte (void)
{
        unsigned char temp;
        unsigned char mask, i;

        mask = 0x80;
        temp = 0;

        for (i=0; i<8; i++)
        {
                        if (getbit())
                             temp |= mask;

                        mask /= 2;
        }

        return temp;
}

void sendstopbit (void)
{
        sclout(LOW);
        delay6us();
        sdaout(LOW);
        delay6us();
        sclout(HIGH);
        delay12us();
        sdaout(HIGH);
        delay12us();
}

/*unsigned char i2cwrite (unsigned char slave_addr, unsigned char *buf_ptr, unsigned char length)
{
        unsigned char i;

        if (checkbusstate() != TRUE)
        {
                        return BUS_BUSY;
        }

        sendstartbit();

        // send address and write command
        if (sendbyte(slave_addr << 1) != LOW)
        {
                        return NO_RESPONSE;
        }

        for (i=0; i<length; i++)
        {
                        if (sendbyte(*buf_ptr++) != LOW)
                        {
                             return ERR_RESPONSE;
                        }
        }

        sendstopbit();

        return OP_DONE;
}           */
/*
unsigned char i2cread (unsigned char slave_addr, unsigned char *buf_ptr, unsigned char length)
{
        unsigned char i;

        if (checkbusstate() != TRUE)
        {
                        return BUS_BUSY;
        }
```

```
        sendstartbit();

        // send address and read command
        if (sendbyte((slave_addr << 1) | 0x01) != LOW)
        {
                        return NO_RESPONSE;
        }

        for (i=0; i<length-1; i++)
        {
                        *buf_ptr++ = getbyte();          // read data
                        sendbit(LOW);                    // ack it low
        }

        *buf_ptr = getbyte();                            // get last data and ack high
        sendbit(HIGH);

        sendstopbit();

        return OP_DONE;
}                       */
```

8. Software IIC interfacing to an EEPROM

Using the previous example's software controlled IIC code it is possible to easily generate some code that can be used to communicate with other IIC devices. In this example, the code is used to store and retrieve data on an external EEPROM. The EEPROM used is the Holtek HT24LC16, this is a CMOS 16K 2-wire serial EEPROM, please refer to the data sheet for this part for any more information on the device. This example only supports byte write and random read operations. Since the HT24LC16 does not use the device address pins, this limits the number of devices that can be used on a single bus to one, the device address for this part is therefore H'A0 which is defined in the header file and used for all accesses.

There are only two functions which are necessary to complete the EEPROM code, these are write and read. The write function needs the device address, the word address and the data. To keep this re-usable the device address is not hard coded into the read and write functions. The first task of this function is to determine the new device address and word address. Since the EEPROM is internally organised with 2048 8-bit words, the 16 K requires an 11-bit data word address for random word addressing. This is done by using 3 bits of the device address as extra bits in the data word address, the remaining address bits are then the word address. This means that the first 3 bits of the word address that are passed to this function must be taken and added to the device address in the correct position. Thus creating the new device address and new word address, the extra bit determining a read or write operation must also be added to the device address. In this example the word address is passed as a 16-bit address and converted into the 8-bit word address. The transmission then follows the following procedure:

- Check the IIC bus state condition.
- Calculate the new device address and the new word address.
- Send the start bit.

- Send the device address.
- Send the word address.
- Send the data.
- Send the stop bit.
- Send start bit and device address, poll the acknowledge (ACK).
- Send start bit, stop bit.

Typically, EEPROM devices require additional time for programming. The HT24LC16 requires a maximum of 5 ms to program the data. Since the device will not acknowledge during a write cycle, this can be used to determine when the cycle is complete. Once the stop condition for a write command has been issued from the master, the device initiates the internally timed write cycle. Acknowledge polling can be initiated immediately. This involves the master sending a start condition followed by the control byte for a write command. If the device is still busy with the write cycle, then no ACK will be returned. Once this is completed, sending the start bit followed by the stop bit will return the devices and the IIC bus to the default state.

The read function works in a very similar way. In this example random reads are performed, using this method the master must send the device address, word address, issue a start condition, the device address and then read the data. Depending on whether the acknowledge is present will determine whether the read is a sequential read or random read is performed. In this example the ACK is not present which terminates the read operation.

The main function is used to produce some simple communication between the two devices. In this case there are two buffers created. The first buffer is filled with data, which is then written into the EEPROM. It is possible to check for any errors that may have occurred at this point. Once this is completed the code then reads back the data from the EEPROM, the two buffers can then be compared to check that the data is the same in both.

Code Example

```
/*
    FILE:      EEPROM.C
    DATE:      4-8-98

                DESCRIPTION:Code to extend IIC functionality to read/write from a serial
                                        EEPROM (HT24LC16 - 2KB).
*/
#include "eeprom.h"

unsigned char i2c_eeprom_write (unsigned char slave_addr, unsigned short write_addr, unsigned char
*buf_ptr)
{
        unsigned char i, new_slave_addr, new_write_addr;

        union {
                unsigned char c[2];/* union to convert short to 2 chars */
                unsigned short s;  /* c[0] = ms byte */
        } data;                    /* c[1] = ls byte */

        if (checkbusstate() != TRUE)
        {
                return BUS_BUSY;
        }


        /* calculate the slave address from the device code and the write_address     */
        /* note that the new_slave_addr value will be shifted left by one bit         */
        /* when passed to the 'send_byte' routine                                     */
        data.s = write_addr;
        data.c[0] &= 0x07;   /* clear all bits 3 to 7 */
        data.c[0] << 1;             /* we shift data.c to the left by 1 bit to allow the addtion of
the write bit   */
        new_slave_addr = slave_addr | data.c[0];  /* bits  8, 9 and 10 of write_addr         */
                                    /* = bits 2, 1 and 0 of slave_addr                */
                                    /* before shifting left when                      */
                                    /* they become bits 3, 2 and 1 of                 */
                                    /* device select byte                     */
        new_write_addr = data.c[1];


        sendstartbit();

        /* send address and write command - ACK is tested in the sendbyte function*/
        if (sendbyte (new_slave_addr ) != LOW)
        {
                return NO_RESPONSE;
        }

        /* send byte address */
        if (sendbyte (new_write_addr) != LOW)
        {
                return NO_RESPONSE;
        }

        if (sendbyte(*buf_ptr++) != LOW)
        {
                return ERR_RESPONSE;
        }

        sendstopbit();

        /* poll on ACK to test when the device has programmed the byte */
        i = HIGH;

        while (i == HIGH)    /* loop until ACK (low) is received */
        {
                sendstartbit();
                i = sendbyte(new_slave_addr );
```

```
        }

        sendstartbit();

        sendstopbit();

        return OP_DONE;
}

unsigned char i2c_eeprom_read (unsigned char slave_addr, unsigned short read_addr, unsigned char
*buf_ptr)
{
        unsigned char i, new_slave_addr, new_read_addr;

        union {
                 unsigned char c[2];/* union to convert short to 2 chars */
                 unsigned short s;  /* c[0] = ms byte */
        } data;                     /* c[1] = ls byte */

        if (checkbusstate() != TRUE)
        {
                 return BUS_BUSY;
        }


        /* calculate the slave address from the device code and the read_addr  */
        /* note that the new_slave_addr value will be shifted left by one bit  */
        /* when passed to the 'send_byte' routine                               */
        data.s = read_addr;
        data.c[0] &= 0x07;   /* clear all bits 3 to 7 */
        data.c[0] << 1;             /* we shift data.c to the left by 1 bit to allow the addtion of
the write bit    */
        new_slave_addr = slave_addr | data.c[0];  /* bits  8, 9 and 10 of write_addr  */
                                          /* = bits 1 and 0 of slave_addr            */
                                          /* before shifting left when               */
                                          /* they become bits 3, 2 and 1 of          */
                                          /* device select byte                      */
        new_read_addr = data.c[1];

        sendstartbit();

        /* send address and write command */
        if (sendbyte(new_slave_addr) != LOW)
        {
                 return NO_RESPONSE;
        }

        /* send byte address */
        if (sendbyte (new_read_addr) != LOW)
        {
                 return NO_RESPONSE;
        }

        sendstartbit();

        /* send address and read command */
        if (sendbyte((new_slave_addr ) | 0x01) != LOW)
        {
                 return NO_RESPONSE;
        }

        *buf_ptr = getbyte();                     /* get last data and ack high */
        sendbit(HIGH);

        sendstopbit();

        return OP_DONE;
}
```

```
void dummy (void)
{
}

void main (void)
{
        unsigned char i;
        unsigned short t;
        static unsigned char buf[25];
        static unsigned char buf_in[25];


        for (i=0; i<25; i++)
                 buf[i] = 0x44;

        buf[0] = 0x44;


        t = 0;
        i = OP_DONE;


        while ( (t < 25) && (i == OP_DONE) )
        {
                 i = i2c_eeprom_write (EEPROM_1, t, &buf[0]);
                 t++;
        }

/*      switch(i)
        {
                 case OP_DONE:
                      dummy();
                 break;

                 case BUS_BUSY:
                      dummy();
                 break;

                 case NO_RESPONSE:
                      dummy();
                 break;

                 case ERR_RESPONSE:
                      dummy();
                 break;
        } */


        t = 0;
        i = OP_DONE;

        while ( (t < 25) && (i == OP_DONE) )
        {
                 i = i2c_eeprom_read (EEPROM_1, t, &buf_in[t]);
                 t++;
        }

        switch(i)
        {
                 case OP_DONE:
                      dummy();
                 break;

                 case BUS_BUSY:
                      dummy();
                 break;

                 case NO_RESPONSE:
                      dummy();
                 break;
```

```
                case ERR_RESPONSE:
                     dummy();
                break;
        }
}
```

9. Software IIC interfacing to a RTC (Real Time Clock)

As mentioned in the previous example, using the software controlled IIC code it is possible to generate code that can interface with other IIC devices. This example extends this and uses the code to interface to a Philips low power clock, PCF8593, for more information on this device please consult the data sheet for this part.

The RTC device is configured such that first eight registers are designed as addressable 8-bit parallel registers, the first register is used as a control/status register and memory addresses 01 to 07 are used for the counters for the clock function. This code gives an example of how to enter the data into the RTC and read a value, i.e. the time, from it. This example then outputs the time that has been read from the IIC RTC on to an LCD. The interface to the LCD has been discussed previously in this application note.

As with the EEPROM example there are two main functions that are used to communicate with the RTC IIC device. These are the “i2c_rtc_read” and “i2c_rtc_write”. The write function works in a very similar way to the previous example except that there is no need to resolve the device address and the word address. Here the read is performed by sending the start bit, sending the slave address, the word address (RTC register), the data byte followed by the stop command. The read function is the same except that an acknowledge bit is sent before the stop command.

There are four main files associated with this example, all of which combine to produce code that reads the RTC device and displays the result onto the previously discussed LCD. The project initialises the RTC to count from a 32 KHz input and so it will send a signal to the H8 every time one second lapses. Once the H8 has received this signal it will read the RTC registers to receive the time, if the corresponding register has changed value then it will update the display. The H8 is configured to receive this input via the IRQ0 pin, Port 4 bit 3, and this is set up to be an IRQ input detecting the falling edge. There are also two switches that are used in this example to set the initial time. These are the WKP pins on the H8, Port 5 bits 0 and 1, these are configured to WKP pins and are interrupt enabled to produce an interrupt on a falling edge, once pressed they will increment the hours and seconds value depending on which switch is activated.

The four files are: IIC.c, this file contains the bit bashing I2C code previously discussed. Switch.c, this file contains the interrupt routine and initialisation code for the switches. Lcd2.c, this file is used to configure the LCD driver and the LCD RAM, please be aware that although the LCD is the same as the previous LCD example the method of controlling the RAM contents differs. RTC_IIC.c, this file contains the main routine and is used to combine the functions. The header files associated with this project can be found in the downloadable files with this application note.

Code Example

```
/*
        FILE:           MAIN.C
        DATE:           13 - 01 - 2003


                         DESCRIPTION:     Code to intialise and demo the IIC/RTC funtionality

*/


#include "rtc.h"         /* This file contains the real time clock specific defines */
#include "lcd2.h"        // This file contains the LCD sepcific defines.
#include "switch.h"      // This file contains the switch sepcific defines.

unsigned char INT_IRQ0;
unsigned char ten_hours=1;
unsigned char hours=1;
unsigned char ten_mins=1;
unsigned char mins=1;
unsigned char ten_secs=1;
unsigned char secs=1;
unsigned char clock_var1, clock_var2;            // These are used to change the LCD display.


const unsigned long clock_lookup[6][10] = {
  {(U_ZERO|L_ZERO), (U_ZERO|L_ONE), (U_ZERO|L_TWO), (U_ZERO|L_THREE), (U_ZERO|L_FOUR),
   (U_ZERO|L_FIVE), (U_ZERO|L_SIX), (U_ZERO|L_SEVEN), (U_ZERO|L_EIGHT), (U_ZERO|L_NINE)} ,
  {(U_ONE|L_ZERO), (U_ONE|L_ONE), (U_ONE|L_TWO), (U_ONE|L_THREE), (U_ONE|L_FOUR),
   (U_ONE|L_FIVE), (U_ONE|L_SIX), (U_ONE|L_SEVEN), (U_ONE|L_EIGHT), (U_ONE|L_NINE) } ,
  {(U_TWO|L_ZERO), (U_TWO|L_ONE), (U_TWO|L_TWO), (U_TWO|L_THREE), (U_TWO|L_FOUR),
   (U_TWO|L_FIVE), (U_TWO|L_SIX), (U_TWO|L_SEVEN), (U_TWO|L_EIGHT), (U_TWO|L_NINE) },
  {(U_THREE|L_ZERO), (U_THREE|L_ONE), (U_THREE|L_TWO), (U_THREE|L_THREE), (U_THREE|L_FOUR),
   (U_THREE|L_FIVE), (U_THREE|L_SIX), (U_THREE|L_SEVEN), (U_THREE|L_EIGHT), (U_THREE|L_NINE)},
  {(U_FOUR|L_ZERO), (U_FOUR|L_ONE), (U_FOUR|L_TWO), (U_FOUR|L_THREE), (U_FOUR|L_FOUR),
   (U_FOUR|L_FIVE), (U_FOUR|L_SIX), (U_FOUR|L_SEVEN), (U_FOUR|L_EIGHT), (U_FOUR|L_NINE) } ,
  {(U_FIVE|L_ZERO), (U_FIVE|L_ONE), (U_FIVE|L_TWO), (U_FIVE|L_THREE), (U_FIVE|L_FOUR),
   (U_FIVE|L_FIVE), (U_FIVE|L_SIX), (U_FIVE|L_SEVEN), (U_FIVE|L_EIGHT), (U_FIVE|L_NINE) } ,
};

void main(void)
{
        /* Add your code here */
        unsigned char i=0;

        /* initialise IIC port */
        init_lcd();
        init_irq0();
    init_rtc();
    init_switches();

        set_interrupt_mask(0);          // enable interrupts

        while(1)
        {
        if (INT_IRQ0 == PENDING)            // If the interrupt has ocurred the flag will be pending.
        {
                rtc_data_array[0] &= 0xFE;          // Clear the Timer flag on the RTC.
                i = i2c_rtc_write (RTC_1, CTRL_STS, &rtc_data_array[0]);        // Write the above to
RTC.
                i = read_rtc_time();        // Read the latest time info.

                if (i != OP_DONE)
                {
                        while(1);
                }

                //      This section of code will update the LCD.
    display_clock();
```

```
                INT_IRQ0 = SERVICED;
        }                                                   // End if (INT_IRQ0 == PENDING)

        if (switch_U10 == SET)              // If the swicth is pressed and debouncing done.
        {
          // If U10 is pressed this will increment the hours that are displayed on the LCD.

                read_rtc_time();            // Ensure the data in the mins/rtc_data array
                ten_hours = (rtc_data_array[HOUR] & 0xF0) >> 4; // is current data.
                hours = (rtc_data_array[HOUR] & 0x0F);

                hours++;                                    // Increment Hours.
                if (hours == 4 && ten_hours == 2)
                {
                        hours = 0;
                        ten_hours = 0;
                }
                 else if (hours == 10)
                         {
                         hours = 0;
                         ten_hours++;
                         }

                 update_time();      // This function updates the RTC I2C device with new time.
                         hours = 10;         // Set the hour to error value this will cause the
                                             // display to change when we enter display_clock();
                 switch_U10 = CLEARED;       // Clear the switch flag ready for next I/P.
                 }                                                  // End if (switch_U10 == SET)

        if (switch_U6 == SET)               // If the swicth is pressed and debouncing done.
                {
                            // If U6 is pressed this will increment the minutes that are displayed
on the LCD.

                read_rtc_time();            // Ensure the data in the mins/rtc_data array
                ten_mins = (rtc_data_array[MIN] & 0xF0) >> 4;    // is current data.
                mins = (rtc_data_array[MIN] & 0x0F);

                mins++;
        // Increment minutes.
                if (mins == 10 && ten_mins == 5)    // Check counts to 59, overflowing at ten.
        {
                mins = 0;
                ten_mins = 0;
        }
                else if (mins == 10)
        {
                mins = 0;
                ten_mins++;
        }

        update_time();              // This function updates the RTC I2C device with new time.
        mins = 10;                  // Set the hour to error value this will cause the
                                    // display to change when we enter display_clock();
        switch_U6 = CLEARED;
                           }                                        // End if (switch_U6 == SET)

        }                                           // End of while(1)
}

void init_irq0(void)
{
        PMR2 |= 0x01;        // Setting bit 0 of the PMR2 sets Port 4.3 as IRQ0 input pin.
        IEGR &= 0xFE;        // Falling edge of IRQ0 pin input is detected
        IENR1 |= 0x01;       // Enables interrupt requests from pin IRQn
        INT_IRQ0 = SERVICED;// Clears the IRQ interrupt function, this is called when IRQ happens.
}

interrupt [IRQ_0] void isr_irq0 (void)
```

```
{
        IRR1 &= 0xFE;     // Clearing conditions: When IRRIn = 1, it is cleared by writing 0
        INT_IRQ0 = PENDING;
}

void display_clock(void)
{
        unsigned long array_shift;

        clock_var1 = (rtc_data_array[HOUR] & 0x30) >> 4;
        clock_var2 = (rtc_data_array[HOUR] & 0x0F);
        if (clock_var1 != ten_hours || clock_var2 != hours)
        {
        ten_hours = clock_var1;
        hours = clock_var2;

        array_shift = clock_lookup[ten_hours][hours];
        digit_six_five[2] = (unsigned char ) array_shift;
        array_shift = array_shift >> 8;
        digit_six_five[1] = (unsigned char ) array_shift;
        array_shift = array_shift >> 8;
        digit_six_five[0] = ((unsigned char ) array_shift) | 0x04;      // The 0x04 adds the DP on the
clock
        }

        clock_var1 = (rtc_data_array[MIN] & 0xF0) >> 4;
        clock_var2 = (rtc_data_array[MIN] & 0x0F);
        if (clock_var1 != ten_mins || clock_var2 != mins)
        {
                        ten_mins = clock_var1;
                        mins = clock_var2;

                        array_shift = clock_lookup[ten_mins][mins];
                        digit_four_three[2] = (unsigned char ) array_shift;
                        array_shift = array_shift >> 8;
                        digit_four_three[1] = (unsigned char ) array_shift;
                        array_shift = array_shift >> 8;
                        digit_four_three[0] = ((unsigned char ) array_shift) | 0x04;       // The
0x04 adds the DP on the clock
        }

        clock_var1 = (rtc_data_array[SEC] & 0xF0) >> 4;
        clock_var2 = (rtc_data_array[SEC] & 0x0F);
        if (clock_var1 != ten_secs || clock_var2 != secs)
        {
                        ten_secs = clock_var1;
                        secs = clock_var2;

                        array_shift = clock_lookup[ten_secs][secs];
                        digit_two_one[2] = (unsigned char ) array_shift;
                        array_shift = array_shift >> 8;
                        digit_two_one[1] = (unsigned char ) array_shift;
                        array_shift = array_shift >> 8;
                        digit_two_one[0] = (unsigned char ) array_shift;
        }

}


void init_rtc (void)
{
        /*      This function is used to initialise the RTC.  This loads all of the data in the RTC
                array into the correct locations in the RTC.      */
        unsigned char t, i;

        t = 0;
        i = OP_DONE;

        while ( (t < 16) && (i == OP_DONE) )
        {
```

```
                        i =i2c_rtc_write (RTC_1, t, &rtc_data_array[t]);
                        t++;
        }
        if (i != OP_DONE)
    {
        while(1);                                   // Error correction code needed here.
    }

    rtc_data_array[0] &= 0x7F;                      // Start the Counter.
    i = i2c_rtc_write (RTC_1, CTRL_STS, &rtc_data_array[0]);
    if (i != OP_DONE)
    {
        while(1);                                   // Error correction code needed here.
    }
}

unsigned char read_rtc_time(void)
{
        /*   This funstion is used to read all of the time registers from the I2C RTC    */
        unsigned char t, i;

        t = SEC;
        i = OP_DONE;

        while ( (t < TIMER) && (i == OP_DONE) )
        {
                        i =i2c_rtc_read (RTC_1, t, &rtc_data_array[t]);
                        t++;
        }
        if (i != OP_DONE)
    {
        while(1);                                   // Error correction code needed here.
    }

    return i;

}

void update_time (void)
{
        /*     This function is used to initialise the RTC.  This loads all of the data in the RTC
               array into the correct locations in the RTC.      */
        unsigned char t, i;

        rtc_data_array[HOUR] = (ten_hours << 4) + hours;
        rtc_data_array[MIN] = (ten_mins << 4) + mins;

        t = MIN;
        i = OP_DONE;

        while ( (t < YEAR) && (i == OP_DONE) )
        {
                        i =i2c_rtc_write (RTC_1, t, &rtc_data_array[t]);
                        t++;
        }

        if (i != OP_DONE)
    {
        while(1);                                   // Error correction code needed here.
    }

}

unsigned char i2c_rtc_write (unsigned char slave_addr, unsigned char rtc_register, unsigned char
*buf_ptr)
{
        unsigned char i;

        if (checkbusstate() != TRUE)
        {
```

```
                        return BUS_BUSY;
        }

        sendstartbit();

        // send address and write command
        if (sendbyte(slave_addr) != LOW)
        {
                        return NO_RESPONSE;
        }

        // Send Register Address (Word address)
        if (sendbyte(rtc_register) != LOW)
        {
                        return NO_RESPONSE;
        }

        // Send data.
        if (sendbyte(*buf_ptr++) != LOW)
        {
                        return ERR_RESPONSE;
        }

        sendstopbit();

        return OP_DONE;
}


unsigned char i2c_rtc_read (unsigned char slave_addr, unsigned char rtc_register, unsigned char
*buf_ptr)
{
        unsigned char i;

        if (checkbusstate() != TRUE)
        {
                        return BUS_BUSY;
        }

        sendstartbit();

        // send address and write command
        if (sendbyte(slave_addr) != LOW)
        {
                        return NO_RESPONSE;
        }

        // send RTC register address (word address)
        if (sendbyte(rtc_register) != LOW)
        {
                        return NO_RESPONSE;
        }

        sendstartbit();

        // send address and read command
        if (sendbyte((slave_addr) | 0x01) != LOW)
        {
                        return NO_RESPONSE;
        }


        *buf_ptr = getbyte();                   // get last data and ack high
        sendbit(HIGH);

        sendstopbit();

        return OP_DONE;
```

APPENDIX A – SEGMENT DEFINITIONS

38024.seg

-Z(BIT)BITVARS=0
-Z(CODE)INTVEC,IFLIST,FLIST=0-FF
-Z(CODE)RCODE,CODE,CDATA0,CDATA1,CDATA2,CDATA3,ZVECT,CONST,CSTR,CCSTR=100-7FFF
-Z(DATA)DATA2,IDATA2,UDATA2,DATA1,IDATA1,UDATA1,DATA0,IDATA0,UDATA0,SHORTAD=FB80-FEFF
-Z(DATA)DATA3,IDATA3,UDATA3,ECSTR,WCSTR,TEMP=FF00-FF00
-Z(DATA)CSTACK+80=FF00-FF7F

38023.seg

-Z(BIT)BITVARS=0
-Z(CODE)INTVEC,IFLIST,FLIST=0-FF
-Z(CODE)RCODE,CODE,CDATA0,CDATA1,CDATA2,CDATA3,ZVECT,CONST,CSTR,CCSTR =100-5FFF
-Z(DATA)DATA2,IDATA2,UDATA2,DATA1,IDATA1,UDATA1,DATA0,IDATA0,UDATA0,SHORTAD=FB80-FEFF
-Z(DATA)DATA3,IDATA3,UDATA3,ECSTR,WCSTR,TEMP=FF00-FF00
-Z(DATA)CSTACK+80=FF00-FF7F

38022.seg

-Z(BIT)BITVARS=0
-Z(CODE)INTVEC,IFLIST,FLIST=0-FF
-Z(CODE)RCODE,CODE,CDATA0,CDATA1,CDATA2,CDATA3,ZVECT,CONST,CSTR,CCSTR =100-3FFF
-Z(DATA)DATA2,IDATA2,UDATA2,DATA1,IDATA1,UDATA1,DATA0,IDATA0,UDATA0,SHORTAD =FB80-FE7F
-Z(DATA)DATA3,IDATA3,UDATA3,ECSTR,WCSTR,TEMP=FF00-FF00
-Z(DATA)CSTACK+80=FF00-FF7F

38021.seg

-Z(BIT)BITVARS=0
-Z(CODE)INTVEC,IFLIST,FLIST=0-FF
-Z(CODE)RCODE,CODE,CDATA0,CDATA1,CDATA2,CDATA3,ZVECT,CONST,CSTR,CCSTR=100-2FFF
-Z(DATA)DATA2,IDATA2,UDATA2,DATA1,IDATA1,UDATA1,DATA0,IDATA0,UDATA0,SHORTAD =FD80-FE7F
-Z(DATA)DATA3,IDATA3,UDATA3,ECSTR,WCSTR,TEMP=FF00-FF00
-Z(DATA)CSTACK+80=FF00-FF7F

38020.seg

-Z(BIT)BITVARS=0

-Z(CODE)INTVEC,IFLIST,FLIST=0-FF

-Z(CODE)RCODE,CODE,CDATA0,CDATA1,CDATA2,CDATA3,ZVECT,CONST,CSTR,CCSTR=100-1FFF

-Z(DATA)DATA2,IDATA2,UDATA2,DATA1,IDATA1,UDATA1,DATA0,IDATA0,UDATA0,SHORTAD =FD80-FE7F

-Z(DATA)DATA3,IDATA3,UDATA3,ECSTR,WCSTR,TEMP=FF00-FF00

-Z(DATA)CSTACK+80=FF00-FF7F

APPENDIX B – IAR HEADER FILE

```
/***************************************************************************
 *          - IOH838024.H -
 *
 *   This file defines internal register addresses for Renesas H8/38024.
 *
 *   Used with ICCH8300 and AH8300.
 *
 *   Copyright:  1999 IAR Systems. All rights reserved.
 *               2003 (c) Copyright Renesas Micro Systems Europe Ltd. (RMSE)
 *
 *   $Revision: 1.2 $
 *
 ***************************************************************************/

#ifndef _IOH838024_INCLUDED
#define _IOH838024_INCLUDED


#ifdef __IAR_SYSTEMS_ICC
#define ICC_UNSIGN_CHAR        *(unsigned char *)
#define ICC_UNSIGN_SHORT       *(unsigned short *)
#else
#define ICC_UNSIGN_CHAR
#define ICC_UNSIGN_SHORT
#endif



                /*--------------------------------*/
                /* FLASH                          */
                /*--------------------------------*/


#define FLMCR1                  (ICC_UNSIGN_CHAR ( 0xFF20 ))
#define FLMCR2                  (ICC_UNSIGN_CHAR ( 0xFF21 ))
#define FLPWCR                  (ICC_UNSIGN_CHAR ( 0xFF22 ))
#define EBR           (ICC_UNSIGN_CHAR ( 0xFF23 ))

#define FENR                    (ICC_UNSIGN_CHAR ( 0xFF2B ))
```

```
                /*---------------------------------------*/
                /* Serial communication interface */
                /*---------------------------------------*/

#define SPCR            (ICC_UNSIGN_CHAR ( 0xFF91 ))


#define SCI3_SMR                (ICC_UNSIGN_CHAR ( 0xFFA8 ))
#define SCI3_BRR                (ICC_UNSIGN_CHAR ( 0xFFA9 ))
#define SCI3_SCR                (ICC_UNSIGN_CHAR ( 0xFFAA ))
#define SCI3_TDR                (ICC_UNSIGN_CHAR ( 0xFFAB ))
#define SCI3_SSR                (ICC_UNSIGN_CHAR ( 0xFFAC ))
#define SCI3_RDR                (ICC_UNSIGN_CHAR ( 0xFFAD ))

                /*------------*/
                /* Timer A */
                /*------------*/

#define TMA                     (ICC_UNSIGN_CHAR ( 0xFFB0 ))
#define TCA             (ICC_UNSIGN_CHAR ( 0xFFB1 ))


                /*----------------------*/
                /* Watchdog Timer */
                /*----------------------*/

#define TCSRW                   (ICC_UNSIGN_CHAR ( 0xFFB4 ))
#define TCC             (ICC_UNSIGN_CHAR ( 0xFFB5 ))
#define TLC             (ICC_UNSIGN_CHAR ( 0xFFB5 ))


                /*------------*/
                /* Timer C */
                /*------------*/

#define TMC             (ICC_UNSIGN_CHAR ( 0xFFB4 ))
#define TCC             (ICC_UNSIGN_CHAR ( 0xFFB5 ))



                /*-----------------------------*/
                /* Asynchronous counter */
                /*-----------------------------*/

#define ECPWCRH                 (ICC_UNSIGN_CHAR ( 0xFF8C ))
#define ECPWCRL                 (ICC_UNSIGN_CHAR ( 0xFF8D ))
```

```
#define ECPWDRH         (ICC_UNSIGN_CHAR ( 0xFF8E ))
#define ECPWDRL         (ICC_UNSIGN_CHAR ( 0xFF8F ))
#define AEGSR                   (ICC_UNSIGN_CHAR ( 0xFF92 ))
#define ECCR                    (ICC_UNSIGN_CHAR ( 0xFF94 ))
#define ECCSR                   (ICC_UNSIGN_CHAR ( 0xFF95 ))
#define ECH             (ICC_UNSIGN_CHAR ( 0xFF96 ))
#define ECL             (ICC_UNSIGN_CHAR ( 0xFF97 ))


        /*-----------*/
        /* Timer F */
        /*-----------*/

#define TCRF                    (ICC_UNSIGN_CHAR ( 0xFFB6 ))
#define TCSRF                   (ICC_UNSIGN_CHAR ( 0xFFB7 ))
#define TCFH                    (ICC_UNSIGN_CHAR ( 0xFFB8 ))
#define TCFL                    (ICC_UNSIGN_CHAR ( 0xFFB9 ))
#define OCRFH                   (ICC_UNSIGN_CHAR ( 0xFFBA ))
#define OCRFL                   (ICC_UNSIGN_CHAR ( 0xFFBB ))


        /*------------*/
        /* Timer G */
        /*------------*/

#define TMG             (ICC_UNSIGN_CHAR ( 0xFFBC ))
#define ICRGF           (ICC_UNSIGN_CHAR ( 0xFFBD ))
#define ICRGR           (ICC_UNSIGN_CHAR ( 0xFFBE ))


        /*---------------------*/
        /* LCD Controller */
        /*---------------------*/

#define LPCR            (ICC_UNSIGN_CHAR ( 0xFFC0 ))
#define LCR         (ICC_UNSIGN_CHAR ( 0xFFC1 ))
#define LCR2            (ICC_UNSIGN_CHAR ( 0xFFC2 ))


        /*--------------------*/
        /* A/D Converter */
        /*--------------------*/

#define ADRRH           (ICC_UNSIGN_CHAR ( 0xFFC4 ))
#define ADRRL           (ICC_UNSIGN_CHAR ( 0xFFC5 ))
```

```
#define AMR             (ICC_UNSIGN_CHAR ( 0xFFC6 ))
#define ADSR            (ICC_UNSIGN_CHAR ( 0xFFC7 ))

        /*-----------*/
        /* I/O Ports */
        /*-----------*/

#define PMR1            (ICC_UNSIGN_CHAR ( 0xFFC8 ))
#define PMR2            (ICC_UNSIGN_CHAR ( 0xFFC9 ))
#define PMR3            (ICC_UNSIGN_CHAR ( 0xFFCA ))

#define PMR5            (ICC_UNSIGN_CHAR ( 0xFFCC ))


          /*--------------------------------*/
        /* Pulse Width Modulator 2 */
        /*--------------------------------*/

#define PWCR2           (ICC_UNSIGN_CHAR ( 0xFFCD ))
#define PWDRU2          (ICC_UNSIGN_CHAR ( 0xFFCE ))
#define PWDRL2          (ICC_UNSIGN_CHAR ( 0xFFCF ))


        /*--------------------------------*/
        /* Pulse Width Modulator 1 */
        /*--------------------------------*/

#define PWCR1           (ICC_UNSIGN_CHAR ( 0xFFD0 ))
#define PWDRU1          (ICC_UNSIGN_CHAR ( 0xFFD1 ))
#define PWDRL1          (ICC_UNSIGN_CHAR ( 0xFFD2 ))


          /*------------*/
        /* I/O Ports */
        /*------------*/

#define PDR1            (ICC_UNSIGN_CHAR ( 0xFFD4 ))

#define PDR3            (ICC_UNSIGN_CHAR ( 0xFFD6 ))
#define PDR4            (ICC_UNSIGN_CHAR ( 0xFFD7 ))
#define PDR5            (ICC_UNSIGN_CHAR ( 0xFFD8 ))
#define PDR6            (ICC_UNSIGN_CHAR ( 0xFFD9 ))
#define PDR7            (ICC_UNSIGN_CHAR ( 0xFFDA ))
#define PDR8            (ICC_UNSIGN_CHAR ( 0xFFDB ))
#define PDR9            (ICC_UNSIGN_CHAR ( 0xFFDC ))
#define PDRA            (ICC_UNSIGN_CHAR ( 0xFFDD ))
#define PDRB            (ICC_UNSIGN_CHAR ( 0xFFDE ))
```

```
#define PUCR1               (ICC_UNSIGN_CHAR ( 0xFFE0 ))
#define PUCR3               (ICC_UNSIGN_CHAR ( 0xFFE1 ))
#define PUCR5               (ICC_UNSIGN_CHAR ( 0xFFE2 ))
#define PUCR6               (ICC_UNSIGN_CHAR ( 0xFFE3 ))
#define PCR1                (ICC_UNSIGN_CHAR ( 0xFFE4 ))

#define PCR3                (ICC_UNSIGN_CHAR ( 0xFFE6 ))
#define PCR4                (ICC_UNSIGN_CHAR ( 0xFFE7 ))
#define PCR5                (ICC_UNSIGN_CHAR ( 0xFFE8 ))
#define PCR6                (ICC_UNSIGN_CHAR ( 0xFFE9 ))
#define PCR7                (ICC_UNSIGN_CHAR ( 0xFFEA ))
#define PCR8                (ICC_UNSIGN_CHAR ( 0xFFEB ))
#define PMR9                (ICC_UNSIGN_CHAR ( 0xFFEC ))
#define PCRA                (ICC_UNSIGN_CHAR ( 0xFFED ))
#define PMRB                (ICC_UNSIGN_CHAR ( 0xFFEE ))

            /*---------------------*/
            /* System Control  */
            /*---------------------*/

#define WEGR                (ICC_UNSIGN_CHAR ( 0xFF90 ))

#define SYSCR1              (ICC_UNSIGN_CHAR ( 0xFFF0 ))
#define SYSCR2              (ICC_UNSIGN_CHAR ( 0xFFF1 ))
#define IEGR                (ICC_UNSIGN_CHAR ( 0xFFF2 ))
#define IENR1               (ICC_UNSIGN_CHAR ( 0xFFF3 ))
#define IENR2               (ICC_UNSIGN_CHAR ( 0xFFF4 ))

#define IRR1        (ICC_UNSIGN_CHAR ( 0xFFF6 ))
#define IRR2        (ICC_UNSIGN_CHAR ( 0xFFF7 ))

#define IWPR                (ICC_UNSIGN_CHAR ( 0xFFF9 ))

#define CHSTPR1             (ICC_UNSIGN_CHAR ( 0xFFFA ))
#define CHSTPR2             (ICC_UNSIGN_CHAR ( 0xFFFB ))


                    /*----------------------------------*/
                    /* Interrupt vector addresses */
                    /*----------------------------------*/

#define RESET               0x00

#define IRQ_0                       0x08
#define IRQ_1                       0x0A
```

```
#define IRQ_AEC         0x0C
#define WKP0            0x12
#define WKP1            0x12
#define WKP2            0x12
#define WKP3            0x12
#define WKP4            0x12
#define WKP5            0x12
#define WKP6            0x12
#define WKP7            0x12

#define TIMER_A         0x16
#define ASYNC_EC        0x18
#define TIMER_C         0x1A
#define TIMER_FL        0x1C
#define TIMER_FH        0x1E
#define TIMER_C         0x20

#define SCI3                    0x24

#define AD_CONV         0x26

#define INT_SLEEP       0x28



#endif

/* EOF */
```

Website and Support

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

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

Notes regarding these materials

1. This document is provided for reference purposes only so that Renesas customers may select the appropriate Renesas products for their use. Renesas neither makes warranties or representations with respect to the accuracy or completeness of the information contained in this document nor grants any license to any intellectual property rights or any other rights of Renesas or any third party with respect to the information in this document.
2. Renesas shall have no liability for damages or infringement of any intellectual property or other rights arising out of the use of any information in this document, including, but not limited to, product data, diagrams, charts, programs, algorithms, and application circuit examples.
3. You should not use the products or the technology described in this document for the purpose of military applications such as the development of weapons of mass destruction or for the purpose of any other military use. When exporting the products or technology described herein, you should follow the applicable export control laws and regulations, and procedures required by such laws and regulations.
4. All information included in this document such as product data, diagrams, charts, programs, algorithms, and application circuit examples, 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 products listed in this document, please confirm the latest product information with a Renesas sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas such as that disclosed through our website. (http://www.renesas.com)
5. Renesas has used reasonable care in compiling the information included in this document, but Renesas assumes no liability whatsoever for any damages incurred as a result of errors or omissions in the information included in this document.
6. When using or otherwise relying on the information in this document, you should evaluate the information in light of the total system before deciding about the applicability of such information to the intended application. Renesas makes no representations, warranties or guaranties regarding the suitability of its products for any particular application and specifically disclaims any liability arising out of the application and use of the information in this document or Renesas products.
7. With the exception of products specified by Renesas as suitable for automobile applications, Renesas products are not designed, manufactured or tested for applications or otherwise in systems the failure or malfunction of which may cause a direct threat to human life or create a risk of human injury or which require especially high quality and reliability such as safety systems, or equipment or systems for transportation and traffic, healthcare, combustion control, aerospace and aeronautics, nuclear power, or undersea communication transmission. If you are considering the use of our products for such purposes, please contact a Renesas sales office beforehand. Renesas shall have no liability for damages arising out of the uses set forth above.
8. Notwithstanding the preceding paragraph, you should not use Renesas products for the purposes listed below:
   (1) artificial life support devices or systems
   (2) surgical implantations
   (3) healthcare intervention (e.g., excision, administration of medication, etc.)
   (4) any other purposes that pose a direct threat to human life

   Renesas shall have no liability for damages arising out of the uses set forth in the above and purchasers who elect to use Renesas products in any of the foregoing applications shall indemnify and hold harmless Renesas Technology Corp., its affiliated companies and their officers, directors, and employees against any and all damages arising out of such applications.
9. You should use the products described herein within the range specified by Renesas, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas shall have no liability for malfunctions or damages arising out of the use of Renesas products beyond such specified ranges.
10. Although Renesas endeavors to improve the quality and reliability of its products, IC products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Please be sure to implement safety measures to guard against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas 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 applicable measures. Among others, since the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
11. In case Renesas products listed in this document are detached from the products to which the Renesas products are attached or affixed, the risk of accident such as swallowing by infants and small children is very high. You should implement safety measures so that Renesas products may not be easily detached from your products. Renesas shall have no liability for damages arising out of such detachment.
12. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written approval from Renesas.
13. Please contact a Renesas sales office if you have any questions regarding the information contained in this document, Renesas semiconductor products, or if you have any other inquiries.