M16C/26

Interfacing with 1-Wire™ Devices

1.0 Abstract

The following article introduces and shows an example of how to interface Renesas' 16-bit microcontrollers (MCU) to a 1-wire device. A demo program developed for the Mini 26 board is available.

2.0 Introduction

This article describes the hardware connectivity and software used in this demo for interfacing Renesas' M16C/26 Flash MCU to a 1-wire device, the DS1822 temperature sensor.

3.0 1-Wire Interface

A 1-wire interface is a 'bus' that requires only one data line not including ground. An MCU or a microprocessor communicates with a 1-wire device using this line. If the device supports "parasite power", the 1-wire device can be powered with the same data line. Hence, it became known as 1-wire bus (not including ground).

The 1-wire bus can support multiple devices because of its open drain output configuration. Having multiple devices on one bus requires some form of identification for each device. This comes in the form of a unique 64-bit ROM code (which translates to 264 devices can be connected to the bus) that the bus master uses to communicate or address a specific device on the bus.

As 1-wire devices are mostly passive, a bus master is required for control. And having only one signal line, the bus master initiates all half-duplex communications, i.e. only one device can 'talk' at a time. For the demo, the bus master is the Renesas M16C/26 MCU on the Mini 26 board.

3.1 Hardware

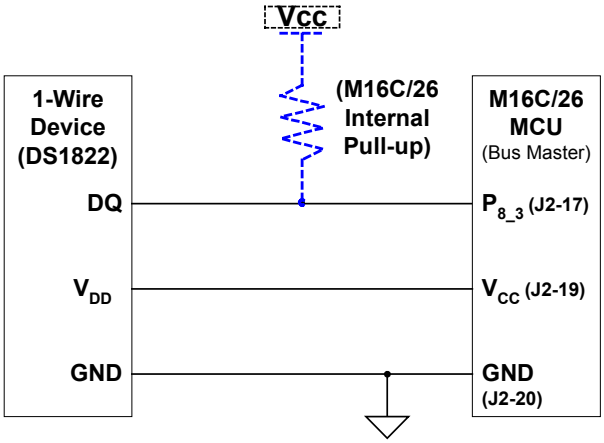
This section describes how to connect a 1-wire device to the M16C/26 MCU and how it was connected for the demo on the Mini 26 board.

3.1.1 1-Wire Device Power

As mentioned earlier, the 1-wire interface uses only one data line. This assumes, however, that the 1-wire device can get its power from this data line. If the device does not support this feature, an external power must be supplied to the device through another line.

The 1-wire device, DS1822, used in this demo can be powered either through the data line ("parasite power") or externally using its Vdd pin. How the DS1822 was connected in this demo is shown on Figure 1. Power is provided externally to the Vdd pin of the DS1822 using a port pin of the MCU. With a 5mA rating, the M16C/26 I/O port can adequately supply power to the DS1822, which can draw current up to 1.5mA (during temperature conversion).

Please see DS1822 datasheet on how to connect hardware in parasite power mode.



Note: The 1-wire device was mounted on the Mini 26's J2 connector.

Figure 1 DS1822 Connection to an M16C/26 MCU (on the Mini 26 Board)

3.1.2 Data/Signal Line

The data line of the 1-wire device is an open drain output. A pull-up resistor is required to bring the bus high, which is the default signal level when the bus is not used. A typical connection of an open drain pin to an output pin using a N-FET is shown on Figure 2. This kind of connectivity, however, will require two port pins: one for input and one for output to drive the N-FET.

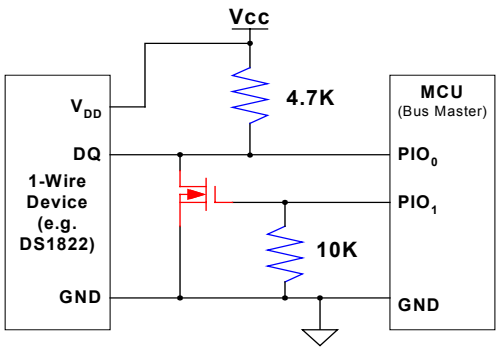


Figure 2 A Typical Open Drain Output MCU Connection using an N-FET

The M16C/26 MCU has two open drain output port pin and can be used when sending data to the 1-wire device. No external FET necessary (but an external pull-up will be required). To be able to use only one pin, the

M16C/26 firmware controls whether the pin is an input or an output. This can be done easily by controlling the M16C/26 I/O port direction register. To emulate an open drain connection, the firmware sets this pin as an input, which is also the default state. When the MCU needs to communicate to the 1-wire device, the pin direction is set to an output. After writing data to the bus, the firmware changes the port direction back to an input.
For the Mini 26 demo, port 8_3 is used with an internal pull-up. The open drain output was not used because an LED is connected to the port pin.

3.2 Software

This section describes how the 1-wire interface was implemented using the M16C/26 MCU. The whole project can be requested from your Renesas representative.

3.2.1 1-Wire Transactions

Communications on the 1-wire bus are handled in the form of transactions, which are initiated by the bus master (M16C/26). A transaction for the DS1822 consists of three steps:

- Initialization Sequence
- ROM Command
- Function Command

3.2.1.1 Initialization Sequence

All transactions start with an initialization. An initialization sequence consists of a reset signal and a presence signal. A reset signal brings all 1-wire devices to attention and that the bus master wants to talk. As an acknowledgement to this, 1-wire devices send a presence signal.
The reset signal for the DS1822 should be at least 480us long. The M16C/26 MCU firmware changes the port direction to an output, brings the data line down for 480us, and then changes port direction back to an input. The data line is pulled high through the resistor after changing the port back to an input.
As a response, the DS1822 will send a presence pulse by bringing the data line low for 60-240us after a timeout period of 15-60us max that started when bus master released the bus (brought data line high). After this low pulse, the DS1822 releases the bus, which brings the data line high. The bus master reads this presence pulse as an indication that a 1-wire device exists and ready to operate.
The initialization sequence for the DS1822 takes at least 960us (480us for reset and 480us for presence pulse plus timeout period). The initialization sequence routine on the one_wire.c program is shown below. The routine returns 0 if it does not detect the presence pulse (no 1-wire device exists) and 1 if it detected one. If a 0 is returned at the time the 64-bit ROM code is being retrieved, future 1-wire bus processing will be stopped since there are 1-wire devices on the bus.

```
/*************************************************************************
Name:           init_trans_1wire
Parameters:
Returns:        one_wire_exist (0 - no device, 1 - device present)
Description:    Initializes communications with 1-wire device. All transactions for
                1-wire communications starts with an initialization sequence so the
                bus master(MCU) knows all slave devices are ready to operate.
*************************************************************************/
char init_trans_1wire(void){

        int wait_time;

        // initialization routine starts with a master reset pulse:
        //    1-wire bus pulled low for at least 480us

        one_wr_dir = wrt_dir;           // 1-wire port - output
        one_wr_port = 0;                // bring 1-wire bus low

        wait_time = 480;                // 480us min reset pulse
        usec_cntr(wait_time);
        one_wr_dir = rd_dir;            // 1-wire port back to an input

        wait_time = 30;                 // wait period for 1-wire device to send
        usec_cntr(wait_time);           // presence pulse - 15-60us

        if (!one_wr_port)               // read presence pulse - if low then
                one_wire_exist = 1;     // 1-wire device exists

        wait_time = 450;                // timeout required :
        usec_cntr(wait_time);           // 480us (reqd) - 30 (wait period) = 450

        return one_wire_exist;
}
```

List 1 1-Wire Transaction Initialization Sequence

3.2.1.2 ROM Command

The bus master issues ROM commands to inform 1-wire devices what it wants to do. A list of the ROM commands, hexadecimal code, and brief description pertinent to the DS1822 are shown on the table below. For details about ROM commands, please see the DS1822 datasheet.

Table 1 DS1822 ROM Commands

ROM Command	Hex Code	Description
Search ROM	F0H	Command used to identify 64-bit ROM code when there are multiple 1-wire devices on the bus.
Read ROM	33H	Command used to identify 64-bit ROM code when only one 1-wire device on the bus.
Match ROM	55H	Command used to address a specific 1-wire device on the bus using a 64-bit ROM code.
Skip ROM	CCH	Command used to address all 1-wire devices on the bus (no 64-bit ROM required).
Alarm Search	ECH	Command used to search for all 1-wire devices with alarm set. Similar to Search ROM command but only 1-wire devices that an alarm is set will respond.

Depending on the ROM command used, a data exchange may follow. For example, when the M16C/26 bus master issues a Read ROM command, the single 1-wire device will respond by sending it's unique 64-bit ROM code. ID. However, when a Skip ROM command is used, no data exchange will occur as the command is for all the 1-wire devices on the bus.

The two ROM commands used for the demo is the Read ROM and Match ROM commands. Read ROM command is used to read the 64-bit ROM code and scratch data of the DS1822. The Match ROM command is used to address the DS1822. However, a commented out routine when reading the DS1822 scratchpad using Skip ROM command can also be found in the source code. The two routines using ROM commands are shown below.

```
/*****************************************************************************
Name:         get_1wire_addr
Parameters:
Returns:        0 - no device, 1 - 1-wire exist and address read
Description:  Called by main to get 1-wire device address ROM code using READ ROM
        command if device exists. Address ROM code is stored in addr_1wire array.

        If device does not exist, returns and stops future 1-wire processing.
*****************************************************************************/
int get_1wire_addr(void){

        unsigned char data_cnt = 0;
        /* Initialize 1-wire port */
        one_wr_dir = rd_dir;    // set to read direction
        pu20 = 1;               // P8_3-0 internal pull-up enabled - 1-wire must pull
                                // data port high

        if (!(init_trans_1wire())) // 1-wire bus initialization transaction and
                return 0;       // check if 1-wire device exists; if not return

        write_byte(rom_cmds[rdrom_cmd]);       // send READ ROM command
```

```
        while(data_cnt != 8){   // we need to read 8 bytes from ds1822 scratchpad
                addr_1wire[data_cnt] = read_byte();     // read byte from ds1822
                ++data_cnt;                             // increment data counter
        }
        return 1;
}
```

List 2 Read ROM Command in get_1wire_addr Routine to Get 64-bit ROM Code

```
// Match ROM Command Routine
        if (!t_conv_flg){                       // send temp conversion command - conv_t
                write_byte(rom_cmds[mtch_cmd]); // send Match ROM command
                send_dev_addr();                // send 1-wire device ROM code

                write_byte(rom_cmds[conv_cmd]); // send convert T function command

                t_conv_flg = 1; // set conversion flag for temp reading next second
                return;
        }
        else{                                   // read temp. measurement
                write_byte(rom_cmds[mtch_cmd]); // send Match ROM command
                send_dev_addr();                // send 1-wire device ROM code

                write_byte(rom_cmds[rd_cmd]);   // send Read Scratchpad command
                while(data_cnt != 9){           // read 9 bytes from ds1822 scratchpad

                        scratch_data[data_cnt] = read_byte();  // store read byte to memory
                        ++data_cnt;                            // increment data counter
                }
                t_conv_flg = 0;         // data read - next second do temp conversion
}
```

List 3 Match ROM Command Routine to Address A Specific 1-Wire Device

3.2.1.3 Function Commands

After ROM commands, the M16C/26 bus master instructs the device what to do next using device-specific commands, which are called Function Commands. A list of function commands for the DS1822 is shown on the table below.

Table 2 DS1822 Function Commands

Function Command	Hex Code	Description
Convert T	44H	Command used to initiate temperature conversion.
Write Scratchpad	4EH	Command used to write 2-bytes temperature data to the scratchpad memory.
Read Scratchpad	BEH	Command used to read the 9-byte scratchpad memory.
Copy Scratchpad	48H	Command used to copy 2-bytes of data to the EEPROM memory.
Recall E^2	B8H	Command used to recall 2-bytes of data from EEPROM memory.
Read Power Supply	B4H	Command used to determine if the device is externally powered or in ‘parasite’ power mode.

The function commands used in this demo are Convert T and Read Scratchpad commands and are highlighted in blue on the previous code listing, List 3.

As temperature conversion takes about 750ms, the newly converted temperature data is read on the next interrupt. What the routine will do during one interrupt is based on the value of t_conv_flag. If the t_conv_flag is 0, the M16C/26 bus master will issue a Convert T command and a Read Scratchpad command when the flag is set to 1. The initial value of t_conv_flag is 0.

3.2.2 Miscellaneous 1-Wire Firmware Routines

This section describes the other routines used to communicate with a 1-wire device.

3.2.2.1 Write_byte

The write_byte routine is used when the bus master needs to send data to DS1822. The M16C/26 bus master issues commands, whether ROM commands or function commands, in byte units. The command is the input parameter or argument of this routine.

Since the 1-wire bus is actually a serial bus, a byte-to-bit conversion is required. To send a byte, this routine gets the bit data that needs to be sent, and then calls write_bit function to send the bit information. LSB (least significant bit) is sent first.

```
/*******************************************************************************
Name:         write_byte
Parameters:   send data (byte)
Returns:
Description:  Converts byte data to bit format before writing to 1-wire bus.
              LSB first format.
*******************************************************************************/
void write_byte(unsigned char byte_data){

       unsigned char wr_byte = byte_data;
       unsigned char bit_cntr = 1;

       while (bit_cntr <= 8){              // we need to send 8 bits
              write_bit((wr_byte & 0x01)); // write bit data LSB first
              wr_byte >>= 1;               // shift right one bit to get next bit
              ++bit_cntr;                  // decrement bit counter
       }
}
```

List 4 write_byte Routine to Send Commands to 1-Wire Bus

3.2.2.2 Write_bit

The write_bit routine is used to 'serially' send bit information, which are called write time slots. Write time slots should be at least 60us in width and in intervals of at least 1us to allow the DS1822 to recover. Bit data is the input parameter or argument of this routine.

The routine determines whether a '0' or '1' will be sent and executes a write '0' or '1' slot. The difference between these two slots is the time the bus is released (brought back to a high level) by changing the 1-wire data port from an output to an input.

For a write '0' slot, the 1-wire bus is held low for the whole time slot and released only after timing out. For a write '1' slot, the 1-wire bus must be released within a 1us-15us range. The timing to release the 1-wire bus will depend on how fast the 1-wire bus goes to a high level. This may require some hardware evaluation and tweaking.

```
/*************************************************************************
Name:         write_bit
Parameters:   bit data
Returns:
Description:  Writes bit data to 1-wire bus. Write time slots are 60us in
              width and written with 1us intervals.

*************************************************************************/
void write_bit(unsigned char bit_data){

       int wait_time = 1;
       usec_cntr(wait_time);                    // 1us interval between writes

/* to write a 1: low pulse ( > 1us) + high pulse (60us - low pulse (in us) */
/* to write a 0: low pulse for 60us                                        */

       if (!bit_data){                          // bit data == 0
              one_wr_dir = wrt_dir;             // 1-wire port - output
              one_wr_port = 0;                  // bring port low
              wait_time = 60;                   // wait for 60us
              usec_cntr(wait_time);
              one_wr_dir = rd_dir;              // 1-wire port - input
       }
       else{                                    // bit data == 1
              one_wr_dir = wrt_dir;             // 1-wire port - output
              one_wr_port = 0;                  // bring port low
              wait_time = 5;                    // wait for > 1us
              usec_cntr(wait_time);
              one_wr_dir = rd_dir;              // 1-wire port - input
              wait_time = 55;                   // wait for 55us (= 60us - 5us)
              usec_cntr(wait_time);
       }
}
```

List 5 write_bit Routine

3.2.2.3 Read_byte

This routine is used by the M16C/26 (bus master) to read data from the DS1822. The routine returns a byte data to the calling routine after converting the bit data to a byte. LSB is bit received first. In addition, whether the bit data is a '0' or a '1' is determined in this routine.

```
/*************************************************************************
Name:         read_byte
Parameters:
Returns:      received byte
Description:  Converts bit data read from 1-wire bus to byte format. Data from
              DS1822 comes LSB first so some bit manipulation is required.
*************************************************************************/
unsigned char read_byte(void){

       unsigned char rd_byte = 0;
       unsigned char i;

       for (i = 1; i <= 8; i++){          // we need to read 8 bits
              if (read_bit())             // read bit data
                     rd_byte |= 0x80;     // change MSB from 1 to 0
              if (i < 8)                  // only shift 7 times
                     rd_byte >>= 1;       // shift right one bit to get next bit
       }
       return(rd_byte);
}
```

List 6 read_byte Routine to Read Data from 1-Wire Device

3.2.2.4 Read_bit

This routine is used to read/sample bit data from the 1-wire bus. It returns bit data to the read_byte routine. The routine samples the 1-wire bus within a read time slot. Like write time slots, the width of a read time slot should be at least 60us and in 1us (min) intervals.

To get a bit sample, the 1-wire bus is brought low for at least 1us to inform the 1-wire device that the M16C/26 bus master is ready to read bit data. The bus is then released (change from output back to input) and the routine waits a several us (15us max) before sampling the 1-wire bus. The sample is then sent back to read_byte routine.

```
/*************************************************************************
Name:         read_bit
Parameters:
Returns:      bit data
Description:  Reads bit data from 1-wire bus. Read time slots are 60us in
              width and written with 1us intervals.

              Master must read bit within 15us after bringing port low.

*************************************************************************/
```

```
unsigned char read_bit(void){

       int wait_time = 1;
       unsigned char bit_data = 0;        // bit data initialize to 0

       usec_cntr(wait_time);              // 1us interval between reads

/* To read bit data from DS1822, a low pulse ( > 1us) is required to initiate process.
The bit is read within a 15us window.                          */

       one_wr_dir = wrt_dir;              // 1-wire port - output
       one_wr_port = 0;                   // bring port low
       wait_time = 1;                     // wait for 1us
       usec_cntr(wait_time);
       one_wr_dir = rd_dir;               // 1-wire port - input

       wait_time = 1;                     // wait for > 1us but < 15us
       usec_cntr(wait_time);

       // read bit data from 1-wire bus
       if (one_wr_port)                   // if a 1, change bit variable to 1
              bit_data = 1;

       wait_time = 58;                    // wait for 58us (= 60us - 1us - 1us)
       usec_cntr(wait_time);

       return(bit_data);                  // return bit data
}
```

List 7 read_bit Routine

3.2.2.5 Send_dev_addr

This routine is used to send the 64-bit ROM code of a 1-wire device. The M16C/26 bus master addresses a specific 1-wire device by sending the 64-bit ROM code of the 1-wire device. The 64-bit code, read in the get_1wire_addr routine, is stored in an array. This routine reads the 64-bit (8-byte) code from the array and sends it in byte increments.

```
/*****************************************************************************
Name:         send_dev_addr
Parameters:
Returns:
Description:  Sends the 64-bit ROM code of the 1-wire device.

*****************************************************************************/
void send_dev_addr(void){

       int data_cnt = 0;

       while (data_cnt != 8){             // there are 8 bytes (64 bits) to send
```

```
                write_byte(addr_1wire[data_cnt]);  // bring 1-wire bus low
                ++data_cnt;
        }
}
```

List 8 send_dev_addr Routine

3.2.2.6 Usec_cntr

The timing parameters for handling transactions or sending/reading from the 1-wire bus are in microseconds (us). This routine is given the amount of time (in microseconds, us) to count and returns to calling routine after counter expires. Timer A1, configured as a 1us timer in timer mode, start and stop are controlled inside this routine. Timer A1 is configured in mcu_init routine of main.c.

```
/*************************************************************************
Name:         usec_cntr
Parameters:   time period - no. of us
Returns:
Description:  usec counter function. Calling routine provides the amount of time,
              in usec. no_of_usec is multiplied by 16 because the clock source for
              timer A1 is 16MHz and not 1MHz.

*************************************************************************/
void usec_cntr(int no_of_usec){

       ta1 = no_of_usec * 16;   // no. of us * 16 because timer A1 clock is 16MHz
       ta1s = 1;                // start timer A1
       while (!ir_ta1ic){}      // wait for Timer A1 to expire
       ta1s = 0;                // stop Timer A1
       ir_ta1ic = 0;            // reset Timer A1 irq flag to 0

}
```

List 9 usec_cntr Routine

```
       /* Configure Timer A1 - us (microsecond) counter */
       ta1mr = 0x00; // Timer mode, f1
       ta1 = 0x0;    // initial value - set by usec_cntr function

       ta1s = 0;     // timer A1 will be started/stopped by usec_cntr function
```

List 10 Timer A1 Initialization Snippet from mcu_init Routine in main.c

4.0 Conclusions

1-wire devices provide flexibility in various applications using very few signal lines. Connecting these 1-wire devices and implementing the interface for reading data and control are easily accomplished using the Renesas M16C/26 MCU.

5.0 Reference

Renesas Technology Corporation Semiconductor Home Page

http://www.renesas.com

E-mail Support

support_apl@renesas.com

Data Sheets

- M16C/26 datasheets, M30262eds.pdf

User's Manual

- M16C/20/60 C Language Programming Manual, 6020c.pdf
- M16C/20/60 Software Manual, 6020software.pdf
- Interrupt Handler App Note, M16C26_Interrupt_Handlers_in_C.doc
- Mini 26 Users Manual, Users_Manual_Mini26B.pdf

For more information on 1-Wire devices, device datasheets, application notes, please visit:

http://www.maxim-ic.com/1-Wire.cfm

6.0 Software Code

The 1-wire routines for this demo can be found on one_wire.c, which is listed below. The project, written for the Mini 26 Board, can be requested from your Renesas representative.

```
/*****************************************************************************
*
*       File Name:   one_wire.c
*
*       Content:  Code for interfacing with a 1-wire device. The MCU is the
*                 bus master and all 1-wire devices connected to it are slaves.
*
*       Copyright (c) 2003 Renesas Technology America, Inc.
*       All rights reserved
*
*=============================================================================
*       $Log:$
*============================================================================*/

#include "..\common\sfr262.h"
#include "one_wire.h"
```

```
int get_1wire_addr(void);
void get_1wire_samp(void);
char init_trans_1wire(void);
void send_dev_addr(void);
void usec_cntr(unsigned int);
unsigned char read_byte(void);
unsigned char read_bit(void);
void write_byte(unsigned char);
void write_bit(unsigned char);

unsigned char rom_cmds[] = {   // ROM Commands to 1-wire device
        0x00,
        0x44,           // Convert T - initiates temp conversion
        0xBE,           // Read scratchpad including CRC
        0x4E,           // Write scratchpad bytes 2 and 3 - Th and Tl
        0x48,           // Copy Th and Tl from scratchpad to EEPROM
        0xB8,           // Recall Th and Tl from EEPROM to scratchpad
        0xB4,           // Read Power Supply Mode of 1-Wire device
        0xF0,           // Search/identify ROM codes of all slave devices
        0x33,           // Read ROM - if only one slave device
        0x55,           // Match ROM - identify which slave device to address
        0xCC,           // Skip ROM - address all slave device w/o ROM code
        0xEC            // Alarm Search - identify slave with alarm flag set
};

char one_wire_exist = 0;              // 0 - no 1-wire device, 1 - 1-wire device exists
char t_conv_flg = 0;                  // 0 - temp conversion, 1 - read temp - due to 750ms
time for temp conversion

unsigned char scratch_data[9] = { 0, 0, 0, 0, 0, 0, 0, 0, 0};  // array to store ds1822
                                                               // scratchpad
unsigned char addr_1wire[8] = { 0, 0, 0, 0, 0, 0, 0, 0};       // array for 64-bit ROM code
                                                               // address of 1-wire device
unsigned temp_data;               // C to F converted temperature data

/*************************************************************************
Name:           get_1wire_addr
Parameters:
Returns:        0 - no device, 1 - 1-wire exist and address read
Description:    Called by main to get 1-wire device address ROM code using READ ROM
                command if device exists. Address ROM code is stored in addr_1wire array.

                If device does not exist, returns and stops future 1-wire processing.
*************************************************************************/
```

```
int get_1wire_addr(void){

        unsigned char data_cnt = 0;

        /* Initialize 1-wire port */
        one_wr_dir = rd_dir;    // set to read direction
        pu20 = 1;       // P8_3-0 internal pull-up enabled - 1-wire must pull data port high

        if (!(init_trans_1wire()))      // 1-wire bus initialization transaction and check if
                return 0;               // 1-wire device exists; if not return

        write_byte(rom_cmds[rdrom_cmd]);        // send READ ROM command
        while(data_cnt != 8){           // we need to read 8 bytes from ds1822 scratchpad

                addr_1wire[data_cnt] = read_byte();     // read byte from ds1822
                ++data_cnt;                             // increment data counter
        }
        return 1;
}

/*************************************************************************
Name:         get_1wire_samp
Parameters:
Returns:
Description:  Called by main to get temp sample from the 1-wire device, ds1822.
                Bus transaction sequence always consists of:
                   a. Initialization
                   b. ROM Command and any required data exchange
                   c. 1-Wire Function Command

                Doing temp conversion ds1822 takes 750ms so we break it into two steps:
                  1. Send a temp conversion command.
                  2. Read temp (/scratch) data

                Which step gets executed depends on the t_conv_flag:
                  t_conv_flag = 0: temp conversion
                  t_conv_flag = 1: read temp data
*************************************************************************/
void get_1wire_samp(void){

        unsigned char data_cnt = 0;
        unsigned int temp_var;
        float temp_value;

        init_trans_1wire();             // 1-wire bus initialization transaction

// Skip ROM Command Routine
/*      write_byte(rom_cmds[skip_cmd]);         // send Skip ROM command
        write_byte(rom_cmds[rd_cmd]);           // send Read Scratchpad command
        while(data_cnt != 9){           // we need to read 9 bytes from ds1822 scratchpad

                scratch_data[data_cnt] = read_byte();     // read byte from ds1822
                ++data_cnt;                     // increment data counter
        }
*/
```

```
// Match ROM Command Routine
        if (!t_conv_flg){              // send temp conversion command - conv_t

                write_byte(rom_cmds[mtch_cmd]);        // send Match ROM command
                send_dev_addr();               // send 1-wire device ROM code

                write_byte(rom_cmds[conv_cmd]);        // send convert T function command

                t_conv_flg = 1;        // set conversion flag for temp reading next second
                return;
        }
        else{                          // read temp. measurement

                write_byte(rom_cmds[mtch_cmd]);        // send Match ROM command
                send_dev_addr();                       // send 1-wire device ROM code

                write_byte(rom_cmds[rd_cmd]);          // send Read Scratchpad command
                while(data_cnt != 9){                  // read 9 bytes from ds1822 scratchpad

                        scratch_data[data_cnt] = read_byte();  // store read byte to memory
                        ++data_cnt;                    // increment data counter
                }
                t_conv_flg = 0;        // data read - next second do temp conversion
        }

        if (scratch_data[1] & 0xF0){           // negative temp ?
                temp_var = ((unsigned int) scratch_data[1]) << 8;   // get MSB temp data and
                                                       // shift 8 bits to the left
                temp_var &= 0xFF00;                    // zero out lower 8 bits
                temp_var |= (unsigned int) scratch_data[0];  // get LSB temp data ORed with
                                                             // MSB data
                temp_var = (0xFFFF - temp_var) >> 3;   // shift left 3x after reading negative
                                                       // temp
                temp_var &= 0xFF;                      // get temp data in 8 bit form
        }
        else{
                temp_var = ((unsigned int) scratch_data[1]) << 8;   // get MSB temp data and
                                / /shift 8 bits to the left
                temp_var &= 0xFF00;                                 // zero out lower 8 bits
                temp_var = (temp_var | (unsigned int) scratch_data[0]) >> 3; // get LSB temp
                                // data ORed with MSB data and then shift left 3x
                temp_var &= 0xFF;                      // get temp data in 8 bit form
        }

        temp_var /= 2;                                 // calculate temp
        temp_var  = ((temp_var * 9) / 5) + 32;         // convert C to F
        temp_data = temp_var;
}
```

```
/*************************************************************************
Name:         init_trans_1wire
Parameters:
Returns:
Description:  Initializes communications with 1-wire device. All transactions for
               1-wire communications starts with an initialization sequence so the
               bus  master(MCU)  knows  all  slave  devices  are  ready  to  operate.
*************************************************************************/
char init_trans_1wire(void){

        unsigned int wait_time;

        // initialization routine starts with a master reset pulse:
        //    1-wire bus pulled low for at least 480us

        one_wr_dir = wrt_dir;          // 1-wire port - output
        one_wr_port = 0;               // bring 1-wire bus low

        wait_time = 480;               // 480us min reset pulse
        usec_cntr(wait_time);
        one_wr_dir = rd_dir;           // 1-wire port - input

        wait_time = 30;                // wait period for 1-wire device to send
        usec_cntr(wait_time);          // presence pulse - 15-60us

        if (!one_wr_port)              // read presence pulse - if low then
                one_wire_exist = 1;    // 1-wire device exists

        wait_time = 450;               // timeout required: 480us(reqd)-30(wait period)=450
        usec_cntr (wait_time);

        return one_wire_exist;
}

/*************************************************************************
Name:         send_dev_addr
Parameters:
Returns:
Description:  Sends the 64-bit ROM code of the 1-wire device.
*************************************************************************/
void send_dev_addr(void){

        int data_cnt = 0;

        while (data_cnt != 8){                        // there are 8 bytes (64 bits) to send
                write_byte(addr_1wire[data_cnt]);     // bring 1-wire bus low
                ++data_cnt;
        }
}
```

```
/*****************************************************************************
Name:         usec_cntr
Parameters:   time period - no. of us
Returns:
Description:  usec counter function. Calling routine provides the amount of time,
               in usec. no_of_usec is multiplied by 20 because the clock source for
               timer A1 is 20MHz.
*****************************************************************************/
void usec_cntr(unsigned int no_of_usec){

        ta1 = no_of_usec * 20;          // no. of us * 20 because timer A1 clock is 20MHz
        ta1s = 1;                       // start timer A1
        while (!ir_ta1ic){}             // wait for Timer A1 to expire
        ta1s = 0;                       // stop Timer A1
        ir_ta1ic = 0;                   // reset Timer A1 irq flag to 0
}

/*****************************************************************************
Name:         read_byte
Parameters:
Returns:      received byte
Description:  Converts bit data read from 1-wire bus to byte format. Data from
                 ds1822 comes LSB first so some bit manipulation is required.
*****************************************************************************/
unsigned char read_byte(void){

        unsigned char rd_byte = 0;
        unsigned char i;

        for (i = 1; i <= 8; i++){               // we need to read 8 bits
                if (read_bit())                 // read bit data
                        rd_byte |= 0x80;        // change MSB from 1 to 0
                if (i < 8)                      // only shift 7 times
                        rd_byte >>= 1;          // shift right one bit to get next data bit
        }
        return(rd_byte);
}

/*****************************************************************************
Name:         read_bit
Parameters:
Returns:      bit data
Description:  Reads bit data from 1-wire bus. Read time slots are 60us in
          width and written with 1us intervals.

          Master must read bit within 15us after bringing port low.
*****************************************************************************/
unsigned char read_bit(void){

        int wait_time = 1;
        unsigned char bit_data = 0;             // bit data initialize to 0

        usec_cntr(wait_time);                   // 1us interval between reads

/* To read bit data from DS1822, a low pulse ( > 1us) is required to initiate process. */
```

```
/* The bit is read within a 15us window.                                    */

        one_wr_dir = wrt_dir;                   // 1-wire port - output
        one_wr_port = 0;                        // bring port low
        wait_time = 1;                          // wait for 1us
        usec_cntr(wait_time);
        one_wr_dir = rd_dir;                    // 1-wire port - input

        wait_time = 1;                          // wait for > 1us but < 15us
        usec_cntr(wait_time);

        // read bit data from 1-wire bus
        if (one_wr_port)                        // is it a 1, then change bit variable to 1
                bit_data = 1;

        wait_time = 58;                         // wait for 58us (= 60us - 1us - 10us)
        usec_cntr(wait_time);

        return(bit_data);                       // return bit data
}

/*************************************************************************
Name:         write_byte
Parameters:   send data (byte)
Returns:
Description:  Converts byte data to bit format before writing to 1-wire bus.
                 LSB first format.
*************************************************************************/
void write_byte(unsigned char byte_data){

        unsigned char wr_byte = byte_data;
        unsigned char bit_cntr = 1;

        while (bit_cntr <= 8){                  // we need to send 8 bits
                write_bit((wr_byte & 0x01));    // write bit data LSB first
                wr_byte >>= 1;                  // shift right one bit to get next data bit
                ++bit_cntr;                     // decrement bit counter
        }
}

/*************************************************************************
Name:         write_bit
Parameters:   bit data
Returns:
Description:  Writes bit data to 1-wire bus. Write time slots are 60us in
                 width and written with 1us intervals.
*************************************************************************/
void write_bit(unsigned char bit_data){

        int wait_time = 1;

        usec_cntr(wait_time);                   // 1us interval between writes

/* to write a 1: low pulse ( > 1us) + high pulse (60us - low pulse (in us) */
/* to write a 0: low pulse for 60us                                        */
```

```
        if (!bit_data){                         // bit data == 0
                one_wr_dir = wrt_dir;           // 1-wire port - output
                one_wr_port = 0;                // bring port low
                wait_time = 60;                 // wait for 60us
                usec_cntr(wait_time);
                one_wr_dir = rd_dir;            // 1-wire port - input
        }
        else{                                   // bit data == 1
                one_wr_dir = wrt_dir;           // 1-wire port - output
                one_wr_port = 0;                // bring port low
                wait_time = 5;                  // wait for > 1us
                usec_cntr(wait_time);
                one_wr_dir = rd_dir;            // 1-wire port - input
                wait_time = 55;                 // wait for 55us (= 60us - 5us)
                usec_cntr(wait_time);
        }
}
```

Keep safety first in your circuit designs!

- Renesas Technology Corporation puts the maximum effort into making semiconductor products better and more reliable, but there is always the possibility that trouble may occur with them. Trouble with semiconductors may lead to personal injury, fire or property damage.
  Remember to give due consideration to safety when making your circuit designs, with appropriate measures such as (i) placement of substitutive, auxiliary circuits, (ii) use of nonflammable material or (iii) prevention against any malfunction or mishap.

Notes regarding these materials

- These materials are intended as a reference to assist our customers in the selection of the Renesas Technology Corporation product best suited to the customer's application; they do not convey any license under any intellectual property rights, or any other rights, belonging to Renesas Technology Corporation or a third party.
- Renesas Technology Corporation assumes no responsibility for any damage, or infringement of any third-party's rights, originating in the use of any product data, diagrams, charts, programs, algorithms, or circuit application examples contained in these materials.
- All information contained in these materials, including product data, diagrams, charts, programs and algorithms represents information on products at the time of publication of these materials, and are subject to change by Renesas Technology Corporation without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corporation or an authorized Renesas Technology Corporation product distributor for the latest product information before purchasing a product listed herein.
  The information described here may contain technical inaccuracies or typographical errors.
  Renesas Technology Corporation assumes no responsibility for any damage, liability, or other loss rising from these inaccuracies or errors.
  Please also pay attention to information published by Renesas Technology Corporation by various means, including the Renesas Technology Corporation Semiconductor home page (http://www.renesas.com).
- When using any or all of the information contained in these materials, including product data, diagrams, charts, programs, and algorithms, please be sure to evaluate all information as a total system before making a final decision on the applicability of the information and products. Renesas Technology Corporation assumes no responsibility for any damage, liability or other loss resulting from the information contained herein.
- Renesas Technology Corporation semiconductors are not designed or manufactured for use in a device or system that is used under circumstances in which human life is potentially at stake. Please contact Renesas Technology Corporation or an authorized Renesas Technology Corporation product distributor when considering the use of a product contained herein for any specific purposes, such as apparatus or systems for transportation, vehicular, medical, aerospace, nuclear, or undersea repeater use.
- The prior written approval of Renesas Technology Corporation is necessary to reprint or reproduce in whole or in part these materials.
- If these products or technologies are subject to the Japanese export control restrictions, they must be exported under a license from the Japanese government and cannot be imported into a country other than the approved destination.
  Any diversion or reexport contrary to the export control laws and regulations of Japan and/or the country of destination is prohibited.
- Please contact Renesas Technology Corporation for further details on these materials or the products contained therein.