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.

H8/300L

Usage of ADC for Temperature Reading (ADC_Temp)

Introduction

This application note aims to explore the A/D converter function of the H8/38024F SLP(Super Low Power) MCU in measuring a range of temperatures. The characteristics of the temperature sensor are discussed and its corresponding source code implementation is explained.

Temperature is an analog quantity, but digital systems often use temperature to implement measurement, control and protection functions. Reading temperature with a microcontroller (MCU) is simple in concept. There are 8-channel; 10-bit ADCs build into the SLP Microcontroller which can be used to read the output voltage from a temperature sensor (analog-output type).

Various types of sensors can be used to measure temperature. Examples are thermistor, temperature-sensitive resistor. Most thermistors have a negative temperature coefficient (NTC), meaning the resistance goes up as temperature goes down. Of all passive temperature measurement sensors, thermistors have the highest sensitivity (resistance change per degree of temperature change). However, thermistors do not have a linear temperature/resistance curve.

Target Device

H8/300L Super Low Power (SLP) Series – H8/38024F

Contents

1. Thermistor Characteristics

Data for a typical NTC thermistor family is shown in Table 1. This data is for a BC Components thermistor, a typical NTC thermistor. The resistance is given as a ratio (R/R_{25}). Many thermistors in a family will have similar characteristics and identical temperature/resistance curves. A thermistor from this family with a resistance at 25ºC (R_{25}) of 10K would have a resistance of 28.1K at 0ºC and a resistance of 4.086K at 60ºC. Similarly, a thermistor with R_{25} of 5K would have a resistance of 14.050K at 0ºC.

Temp ºC	R/R_{25}	Temp ºC	R/R_{25}
-40	33.210	40	0.5330
-30	17.520	50	0.3605
-20	9.6360	60	0.2490
-10	5.5050	70	0.1753
0	3.2550	80	0.1256
10	1.9870	90	0.0915
20	1.2490	100	0.0677
25	1.0000	110	0.0508
30	0.8059	120	0.0386

Table 1: Typical NTC thermistor data

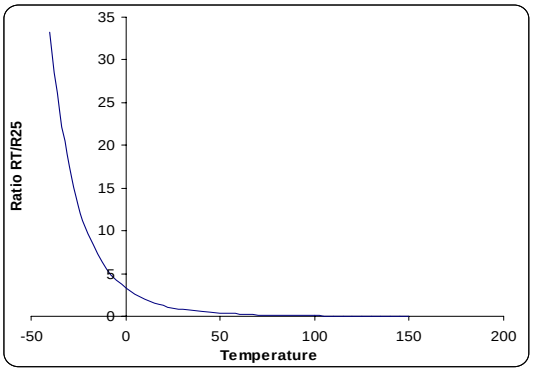


Figure 1: Ratio R_T/R_{25} Vs Temperature (NTC)

Figure 1 shows this thermistor curve graphically. You can see that the resistance/temperature curve is not linear. While the data for this thermistor is given in 10-degree increments, some thermistor tables have five-degree or even one-degree increments. In some cases, the temperature between two points on the table are to be measured. You can estimate this by using the curve, or you can calculate the resistance directly. The formula for resistance looks like this:

$$\frac{R_T}{R_{25}} = \exp\left(A + \frac{B}{T} + \frac{C}{T^2} + \frac{D}{T^3}\right)$$

where *T* is the temperature in degrees Kelvin and *A*, *B*, *C*, and *D* are constants that depend on the characteristics of the thermistor. These parameters must be supplied by the thermistor manufacturer.

Thermistors have a tolerance that limits their repeatability from one sample to the next. This tolerance typically ranges from 1% to 10%, depending on the specific part used. Some thermistors are designed to be interchangeable in applications where it is impractical to have an adjustment. Such an application might include an instrument where the user or a field engineer has to replace the thermistor and has no independent means to calibrate it. These thermistors are much more accurate than ordinary parts, but considerably more expensive.

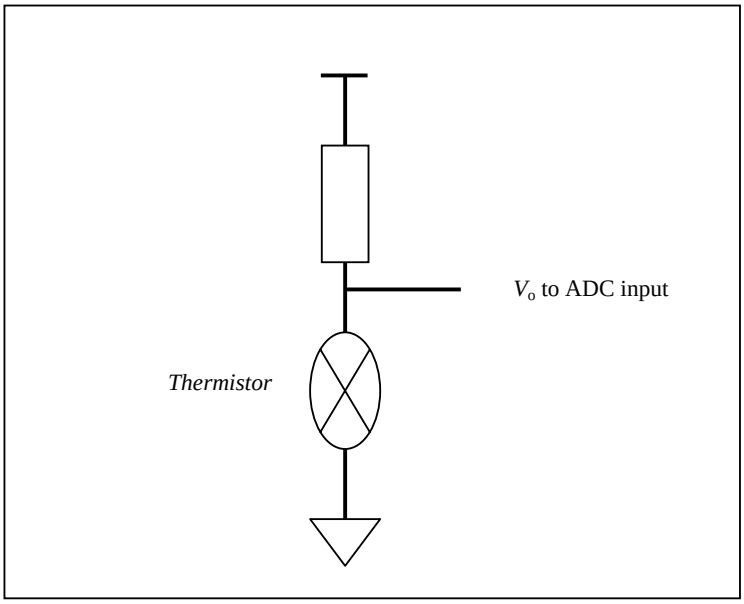


Figure 2: Thermistor circuit

Figure 2 shows a typical circuit that could be used to allow a microprocessor to measure temperature using a thermistor. A resistor (R_1) pulls the thermistor up to a reference voltage. This is typically the same as the ADC reference, so V_{ref} would be 5V if the ADC reference were 5V. The thermistor/resistor combination makes a voltage divider, and the varying thermistor resistance results in a varying voltage at the junction. The accuracy of this circuit depends on the thermistor tolerance, resistor tolerance, and reference accuracy. Since a thermistor is a resistor, passing current through it will generate some heat. The circuit designer must ensure that the pull-up resistor is large enough to prevent excessive self-heating, or the system will end up measuring the thermistor dissipation instead of the ambient temperature. The amount of power that the thermistor has to dissipate to affect the temperature is called the dissipation constant, and is the number of milliwatts needed to raise the thermistor temperature 1ºC above ambient. The dissipation constant varies with the package in which the thermistor is provided, the lead gauge (if a leaded device), type of encapsulating material (if the thermistor is encapsulated), and other factors.

The amount of self-heating allowed, and, therefore, the size of the limiting resistor, depends on the measurement accuracy needed. A system that require an accuracy of ±5ºC can tolerate more thermistor self-heating than a system that must be accurate to ±0.1ºC. Note that the pull-up resistor must be calculated to limit self-heating dissipation over the entire measurement temperature range. For a given resistor, the thermistor dissipation will change at different temperatures because the thermistor resistance changes.

1.1 Thermistor Scaling

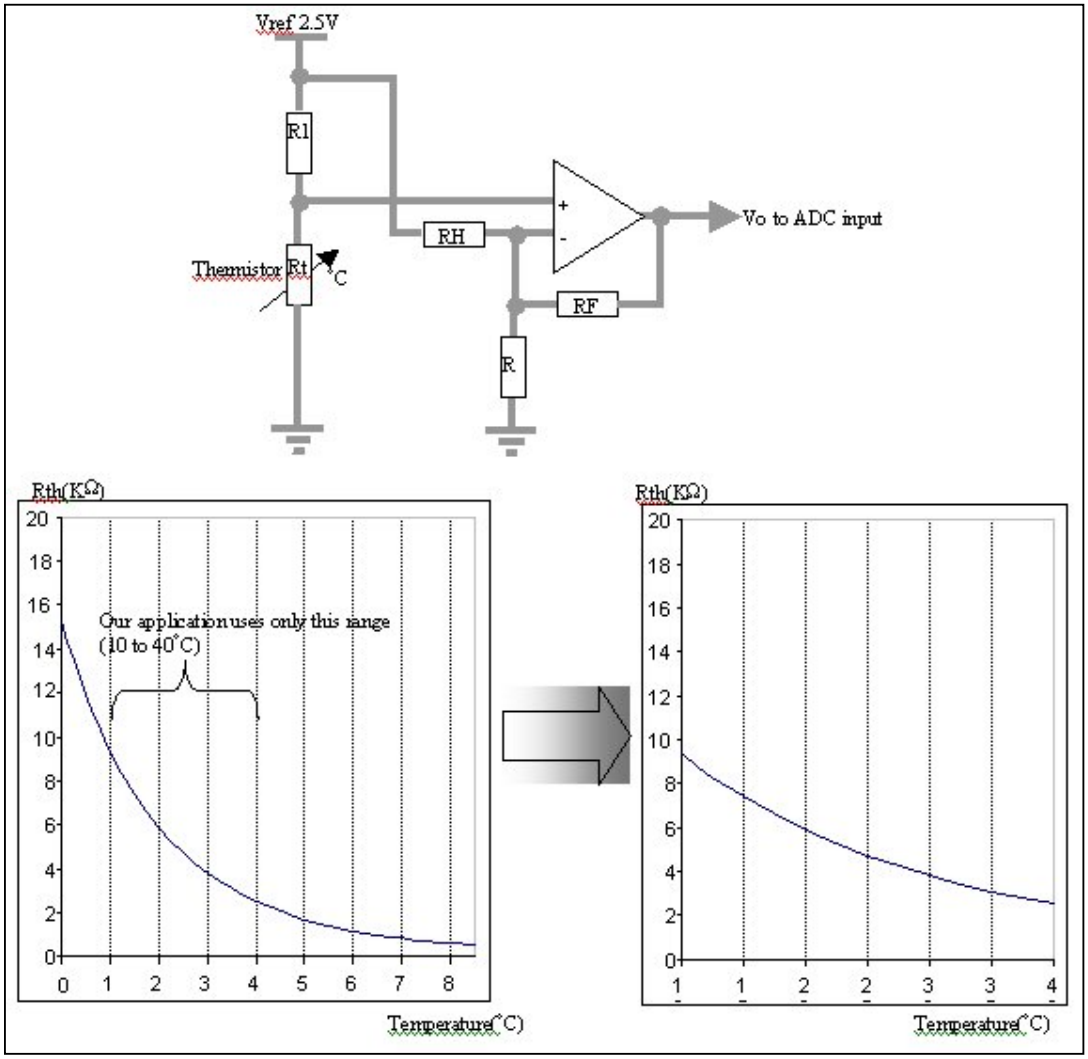


Figure 3: Thermistor scaling

Sometimes you need to scale a thermistor input to get the proper resolution. Figure 3 shows a typical circuit that expands the 10-40ºC range to span the 0-3.3V input of the ADC. The formula for the output of the op amp is as follows:

$$V_o = V_1\left(1+\frac{R_f}{R_l}+\frac{R_f}{R_h}\right)-\frac{V_r R_f}{R_h}$$

Once you have a thermistor scaled (if needed), you can make a chart showing the actual resistance vs. temperature values. You need the chart because the thermistor isn't linear, so the software needs to know what ADC value to expect for each given temperature. The accuracy of the table-one-degree increments or five-degree increments-depends on the accuracy your application requires.

1.2 Tolerance stackup

In any thermistor application, you have to select the sensor and any other components in the input circuit to match your required accuracy. Some applications may only need 1% resistors, but others may require 0.1% resistors. In any event, you should make a spreadsheet showing the effects of tolerance stackup in all the components, including the resistors and references, and the thermistor itself.

If you need more accuracy than you can get with affordable components, you may have to calibrate the system after it is built. In some applications, this is not an option since the circuit boards and/or thermistor must be field-replaceable. However, in cases where the equipment is not field-replaceable, or where the field technicians have an independent means to monitor the temperature, it is possible to let the software build a table of temperature vs. ADC values. There must be some means to input the actual temperature (measured with the independent tool) so the software can construct the table. In some systems, where the thermistor must be field-replaceable, you may be able to calibrate the replaceable component (sensor or entire analog front end) at the factory and provide the calibration data on disk or other storage media. Of course, the software must provide a means to apply the calibration data when the components are changed. In general, thermistors provide a cost-effective means to measure temperature, while still remaining easy to use. Depending on situation, users may opt for RTD and thermocouple sensors as shown below.

	Thermistor	**RTD**	**Thermocouple**
Temperature Range	-100°C to 450°C	-250°C to 900°C	-270°C to 1800°C
Sensitivity	Several Ω/Ω/°C	0.00385 Ω/Ω/°C	10s of μV/°C
Accuracy	± 0.1°C	± 0.01°C	± 0.5°C
Linearity	Require at least 3rd order polynomial or equivalent look up table	Require at least 2nd order polynomial or equivalent look up table	Require at least 4th order polynomial or equivalent look up table
Ruggedness	Various encasements solution. Not easily affected by vibration of shock	Vulnerable to vibration	More rugged and more sturdy with good insulation material
Responsiveness	1 to 5 secs	1 to 10 secs	Less than 1 sec
Excitation	Voltage source	Current source	None
Output Form	Resistance	Resistance	Voltage
Physical Size	0.1 x 0.1 inch	0.25 x 0.25 inch	Bead ∅ = 5 x wire ∅
Price	$2 to $10	$25 to $1000	$1 to $50

Table 2: Characteristics of various temperature sensors

2. Hardware Overview

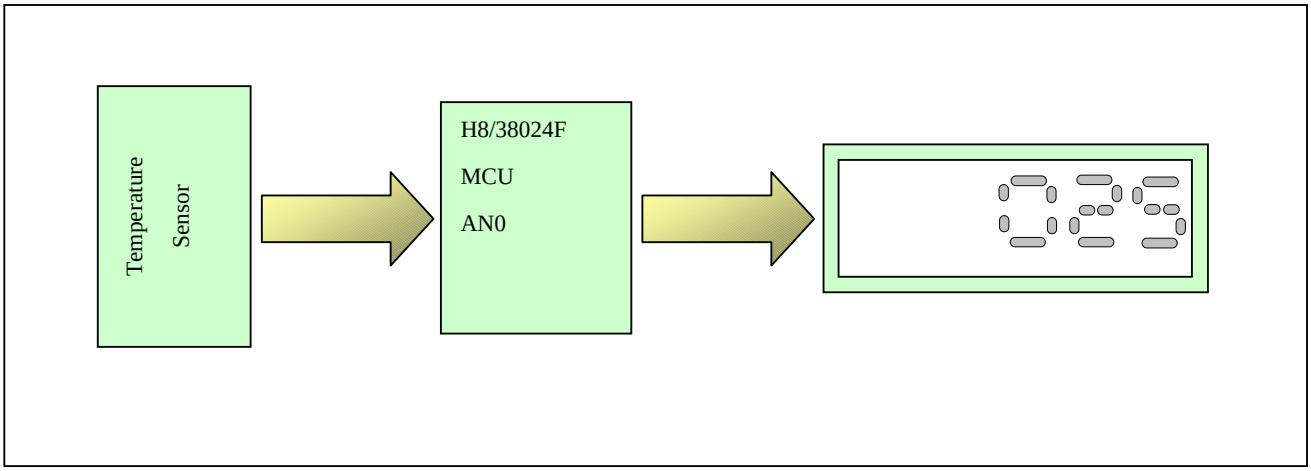


Figure 4: Hardware Block Diagram

Basically, there are 3 components in the temperature sensor circuit:

a. Micro-controller - used to read temperature sensor voltage via build-in ADC port

b. Temperature Sensor - used to convert temperature to voltage

c. LCD Panel - used to display ADC conversion result

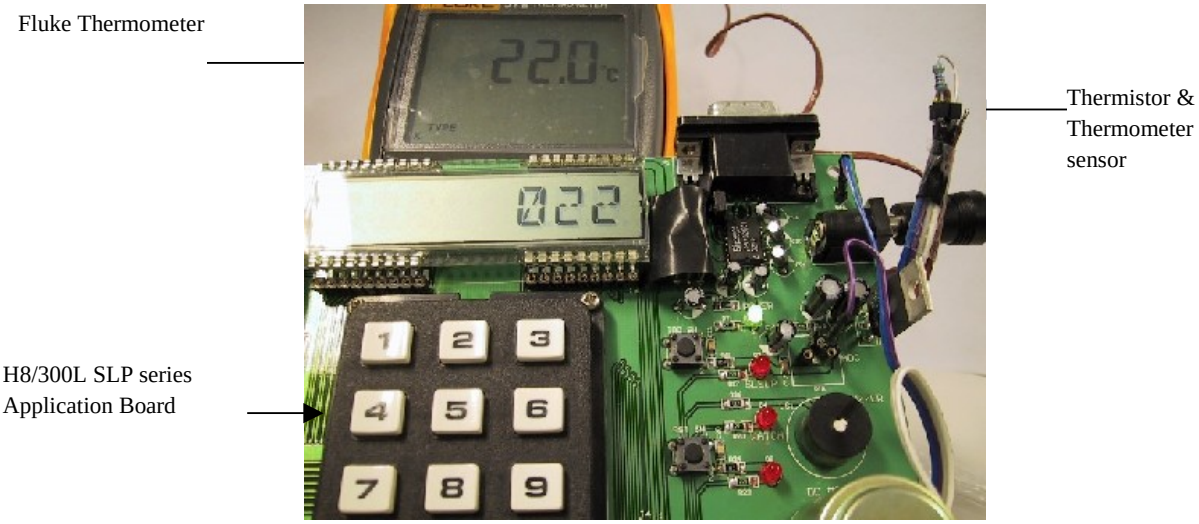


Figure 5: Actual Hardware Implementation

HARDWARE CIRCUIT (Resistor -Thermistor voltage divider):

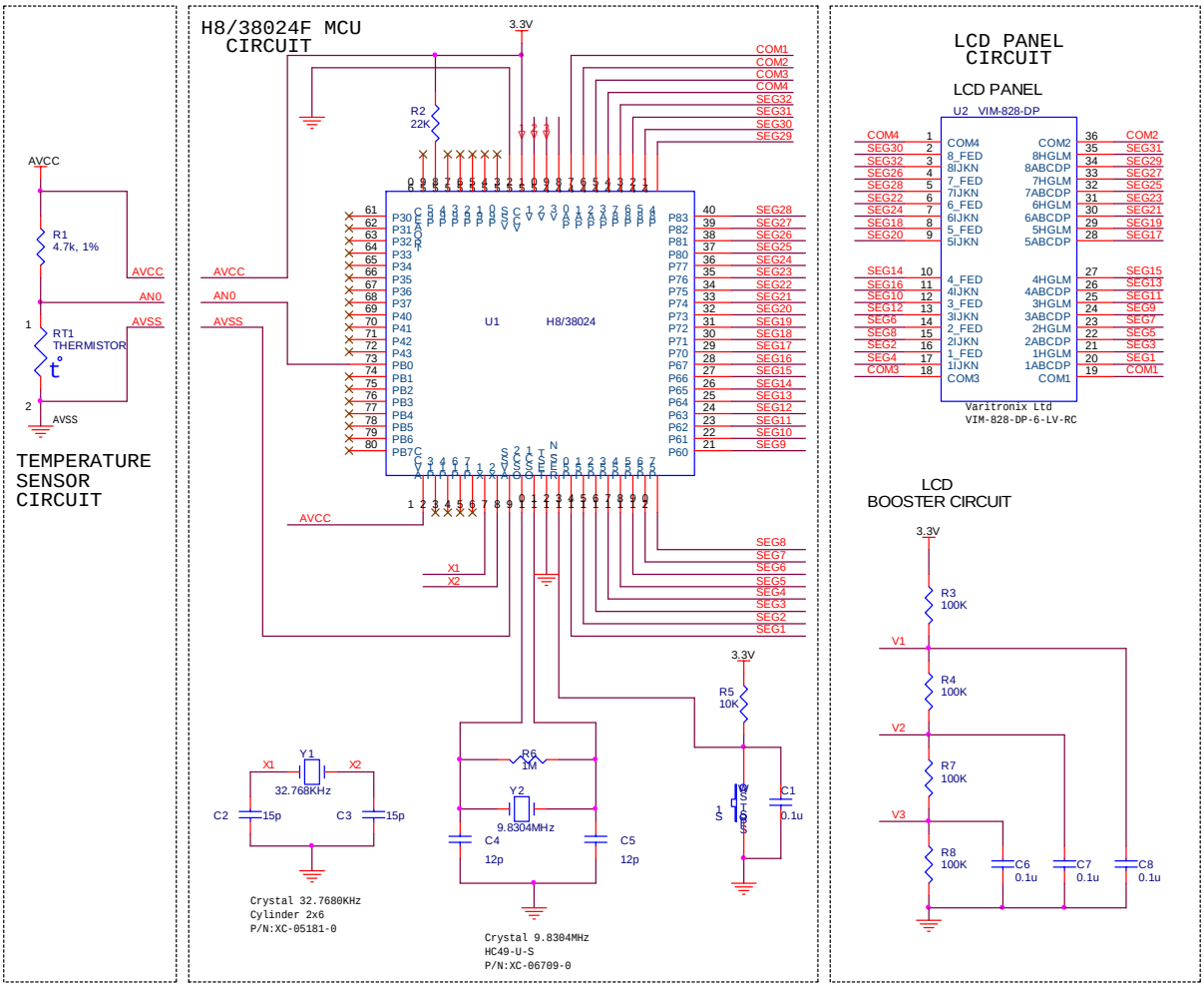


Figure 6: Schematic Diagram for Temperature Reader using Resister-Thermistor voltage divider

Figure 6 shows the schematic diagram for a simple temperature reader. Referring to the figure, towards the left, we can see the circuit for the thermistor/resistor voltage divider. The changing thermistor's resistance, according to ambience temperature, will cause the junction voltage to change also. The ADC input channel zero (AN0) will then capture this junction voltage. AVCC and AVSS, to be used by the A/D converter, refer to the analog power supply and analog ground respectively (please refer to the H8/38024 hardware manual for details).

In the middle, it shows the schematic of the H8/38024 MCU with the necessary connections required for our application. Firstly, AN0 is a multiplexed pin with Port B pin 0. Port B is an 8-bit input-only port configured for the in-built A/D converter 8 input channels. At the bottom, from left to right, you see the sub-clock generator, system-clock generator and the system reset circuits. Ports 5, 6, 7, 8 (SEG0 to SEG32) and Port A (COM1 to COM4) are used for interfacing with the 32x4 LCD display segments and common pins.

On the left, you see the detailed LCD panel connections and below, the boosters circuit to drive the LCD. To display Alphanumeric characters, users are required to write word value to the LCD RAM (address H'F740 to H'F74F) to turn on a particular segment. The booster circuit is required when we need to drive such a large panel, as the on-chip power supply capacity is insufficient.

3. Software Overview

With the appropriate connections made, we are now ready to write our application's code. Firstly, we will need to generate a lookup table for decoding each ADC values. The thermistor values are extracted , from the thermistor data sheet, in steps of 5°C. Next, Table 3 is generated for temperatures ranging from 0°C to 20°C for fixed RT1 & AVcc values. Vin (voltage at AN0) is calculated using voltage-divider rules and the corresponding ADC value is thus obtained (mapping 0V to 0_{10} & 3.3V to 1023_{10}).

Temp (°C)	Thermistor R1 (Ohms)	RT1 (Ohms)	AVcc (V)	Vin at AN0 (V)	ADC value (decimal)	ADC value (round off)
0	15300	4700	3.3	2.524500	783.36	783
5	11910	4700	3.3	2.366225	734.24684	734
10	9340	4700	3.3	2.195299	681.20798	681
15	7378	4700	3.3	2.015847	625.52343	626
20	5869	4700	3.3	1.832501	568.63052	569

Table 3: Temperatures Mapping to ADC value

However, we are more interested in 1°C measurement than a lump 5°C. In order to obtain a closer-to-actual estimate, we need to distribute the intermediate difference between these ADC values to that of 1°C reading. What we are doing here is called linear interpolation. And this you will get:

Temp (°C)	Thermistor R1 (Ohms)	RT1 (Ohms)	AVcc (V)	Vin at AN0 (V)	ADC value (decimal)	ADC value (round off)
0	15300	4700	3.3	2.524500	783.36	783
1					773.53737	774
2					763.71474	764
3					753.8921	754
4					744.06947	744
5	11910	4700	3.3	2.366225	734.24684	734
6					723.63907	724
7					713.03129	713
8					702.42352	702
9					691.81575	692
10	9340	4700	3.3	2.195299	681.20798	681
11					670.07107	670
12					658.93416	659
13					647.79725	648
14					636.66034	637
15	7378	4700	3.3	2.015847	625.52343	626
16					614.14485	614
17					602.76627	603
18					591.38769	591
19					580.0091	580
20	5869	4700	3.3	1.832501	568.63052	569

Table 4: Expanded temperature mapping

Figure 7 shows the beginning of our application source codes. We have implemented the above lookup table into an array called **`temperature[]`**(100 elements), with its array's index as the actual temperature in °C and its element as the corresponding converted ADC value. The rest are function prototypes and variables declaration. Some of them are self-describing and whereas the rest will become clearer as we introduce the rest of the codes. As you can see, all the variables are initialized to hold zero first to prevent garbage values.

```
#include "iodefine.h"
#include "lcd.h"
#include <machine.h>

/************************************************************/
/*  Function define                                         */
/************************************************************/
void init_adc(unsigned char, char, char);
void init_lcd(void);
void display_number(unsigned char, unsigned char, unsigned char);
void taint(void);
void init_timerA(unsigned char);

/************************************************************/
/*  RAM define                                              */
/************************************************************/
// Temperature vs ADC-value lookup table
const unsigned short temperature[101] =
{783,774,764,754,744,734,724,713,702,692,
 681,670,659,648,637,626,614,603,591,580,
 569,557,546,535,523,512,501,490,479,468,
 457,447,436,426,415,405,395,385,376,366,
 356,347,338,329,320,312,303,295,287,279,
 271,264,257,250,243,236,229,223,217,210,
 204,199,193,188,182,177,172,167,162,158,
 153,149,144,140,136,132,129,125,121,118,
 114,111,108,105,102,99,96,94,91,89,
 86,84,81,79,77,75,73,71,69,67,
 65};   // 10x10 table
unsigned char goRead=0;                  //Global variable goRead is use by main() and taint()
```

Figure 7: Function Prototypes & Declarations

```
void main(void)
{
       unsigned short ADC_value=0;
       unsigned char temp1=0, temp2=0, temp3=0, deg=0, i=0, near1=0, near2=0;
    init_lcd();        // Intialize LCD display
    init_adc(0,0,0);       // Intialize ADC
    init_timerA(0x18);     // Intitialize TimerA, default interrupt
    while(1)
    {
        while (goRead == 0);                // Wait for status to start reading temp
        while (P_AD.ADSR.BYTE & 0x80);      // If ADSR = 1, A/D conversion in progress
        ADC_value = P_AD.ADRR >> 6;         // Capture the ADC value
        goRead=0;

        for ( i=0; i<101; i++)              // loop through the whole table, linear search
        {
            // within range?
                    if( (temperature[i]>=ADC_value) && (ADC_value>=temperature[i+1]) )
            {
                        near1 = temperature[i]-ADC_value;       // Calculate to the nearest deg
                near2 = ADC_value-temperature[i+1];
               if(near1<near2) deg=i;       // Round off to the nearest deg
               else  deg=i+1;
            }
        }

        if(deg==100) {temp1=1;}                     // check if reading==100 deg
        else if(deg<100 && deg>9) {temp2=deg/10; temp3=deg%10;}
        else if(deg<10 && deg>=0) temp3=deg;
        else {temp1=16; temp2=29; temp3=29;}        // display 'ERR' if out of range

        display_number(2, temp1, 0);        // Display 100th digit
        display_number(1, temp2, 0);        // Display 10th digit
        display_number(0, temp3, 0);        // Display lowest digit
        temp1=0;temp2=0;temp3=0;            // Reset all
    }
}
```

Figure 8: Function '`main()`'

Figure 8 shows the **`main()`** function where LCD, ADC and Timer A are initialized. Since we are using Timer A to achieve periodic reading of temperatures, due to its overflow timing, we have to calculate the number of loops to do in order to achieve the time period. This is carried out by function **`nloopTA()`**. In our application, we are using a time period of 1 second because the thermistor requires about 1 second to stabilize. Since our focus here is not about using Timers, by default, Timer A uses interrupt request for overflow.

A status flag **goRead** is used to signal the start of temperature reading at every 1 seconds. The status flag **goRead** is set in the Timer A interrupt service routine **taint()** (Figure 10), when the predetermined looping number has been reached. And at the same time, **start_adc()** is called to start the ADC conversion. Then while during its conversion, the statement

while (P_AD.ADSR.BYTE & 0x80);

tests for ADSF flag to clear to signify end of conversion by using bitwise AND operator (&). '0x80' is a hexadecimal representation of decimal value 128 which is bit-7 (MSB) of ADSR register. Next, the ADC value is stored in variable **ADC_value,** by performing a logical left-shift by 6 positions of the ADRRH & ADRRL register value.

Next, the 'for' loop will search the whole array for the temperature range for which the captured ADC value falls in and thus round it off to the nearest degree celsius (array index). Then proper formatting is carried out to display the temperature onto the LCD display.

```
void init_adc(unsigned char conv_period, char Ext_TRG, char Input_CH )
{
    //conv_period = 0  => fast (but OSC1 <= 10MHz)
    //conv_period = 1  => slow (but OSC1 <> 10MHz)
    //Ext_TRG    = 0   => Disable start of A/D conversion by external trigger
    //Ext_TRG    = 1   => Enable start of A/D conversion by external trigger
    //Input_CH   = 0-7 => select ADC input channel
    conv_period = (conv_period & 0x01)<<7;
    Ext_TRG  = (Ext_TRG & 0x01) << 6;
    Input_CH = (Input_CH & 0x07) + 4;
    P_AD.AMR.BYTE = conv_period | Ext_TRG | Input_CH;
}
```

Figure 9: Function 'init_adc()'

Figure 9 shows the function **init_adc()** with a three parameter list, **conv_period** for clock selection, **Ext_TRG** for enabling/disabling of start of A/D conversion by external trigger signal and **Input_CH** for selecting the 8 channels. Then they are all logically ORed together and set into register AMR. For details, please refer to the H8/38024 hardware manual.

```
void init_lcd(void)
{
    unsigned char  temp_a;
    unsigned char  *dest;
    dest = (unsigned char *)0xF740;
    for (temp_a=0 ; temp_a<16 ; temp_a++){*dest++ = 0;}
    P_LCD.LPCR.BYTE = 0xC8;  //1/4 duty cycle
    P_LCD.LCR.BYTE = 0xFF; //display is faint
    P_LCD.LCR2.BYTE = 0x60;
}
```

Figure 10: Function 'init_lcd()'

Figure 10 shows the function **init_lcd()** to initialize the LCD panel for display. It takes in no parameters. Pointer ***dest** is set to point to the starting address of the LCD RAM. The 'for' loop is used to clear the LCD RAM memory. For details on the LCD, please refer to the application board manual APPBD-3800 and H8/38024 hardware manual.

```
void taint( void )
{
    P_SYSCR.IRR1.BIT.IRRTA = 0;     // Clear IRRTA flag
    goRead =1;     // Set status to start to read temp
    start_adc(1);      // start ADC conversion
}
```

Figure 11: Function 'taint()'

Figure 11 shows the interrupt service routine **taint()**, defined inside the same file together with the **main()** . This subroutine is necessary when using interrupts with Timer A because whenever Timer A overflows, this subroutine will be called. First of all, the interrupt request flag **IRRTA** is clear. **nOVF** counts the number of overflows that had occurred and compares with the calculated loop (**count**) required, then if necessary, status flag **goRead** is set and A/D conversion is started. In fact, already there has been allocated a file for all interrupt service routine implementation. This file is **intprg.c**. Thus if we are to define the interrupt service routine outside of this file, you will need to add these two lines of code to this file as shown in Figure 12.

intprg.c file

```
#include <machine.h>
#pragma section IntPRG
extern void taint ( void );         // Add this line
    :
    :
    :
    :
// vector 11 Timer A Overflow
__interrupt(vect=11) void INT_TimerA(void)
{
    taint();              // Add this line
}
```

Figure 12: Implementation of Interrupt Service Routines

```
void start_adc(unsigned char start)
{
    //start = 1 , start ADC
    //start = 0 , stop ADC
    if(start==1) P_AD.ADSR.BYTE |= 0x80;    //Set ADSF : start A/D conversion
    else P_AD.ADSR.BYTE &= 0x7F;                    //Set ADSF : stop A/D conversion
}
```

Figure 13: Function '`start_adc()`'

Figure 13 shows the function **`start_adc()`** when called by function **`taint()`** , will start the A/D conversion by setting ADSF or otherwise clears it to stop the conversion.

```
void init_timerA(unsigned char clkSel)
{
    set_imask_ccr(1);               // Interrupt Disable
    //TMA : |---|---|---|---|TMA3|TMA2|TMA1|TMA0|
    //Bits 7 to 5 are reserved; only 0 can be written to these bits
    //Bit 4 is reserved; it is always read as 1 and cannot be modified
    //Bits 3 to 0 : TMA3 to TMA0 : Internal Clock Select
    P_TMRA.TMA.BYTE = clkSel;
    // 0x18, 1 sec
    // 0x19, 0.5 sec
    // 0x1A, 0.25 sec
    // 0x1B, 0.03125 sec
        P_SYSCR.IRR1.BIT.IRRTA = 0;         // Clear IRRTA flag
    P_SYSCR.IENR1.BIT.IENTA = 1;            // Timer A Interrupt, 1-Enable, 0-Disable
    set_imask_ccr(0);       // Interrupt,0-Enable,1-Disable
    // set_imask_ccr() comes as a pair
}
```

Figure 14: Function '`init_timerA()`'

Figure 14 shows the function **`init_timerA()`** to initialize Timer A to make use of the available subclock timings (i.e. 1sec, 0.5sec, 0.25sec, 31.25msec) as we deem it is most appropriate for our application. Please remember to enable the interrupt request for Timer A to be taken by setting bit IENTA in register IENR1 (please refer to the H8/38024 hardware manual). Function

`set_imask_ccr()` is provided by default in the **`machine.h`** library, so you do not need to define it. It actually helps to temporary disable/mask any system interrupts while you are modifying/enabling any interrupt flags. If not, any system interrupts may interrupt you before you actually enable an interrupt and therefore set off a chain of interrupt reactions. This may cause system unstability and data inconsistency.

```
void display_number(unsigned char digit, unsigned char number, unsigned char decimal_point)
{
    unsigned short  *dest;

    switch(digit)
    {   case 0:
            dest = (unsigned short *)0xF740; break;
        case 1:
            dest = (unsigned short *)0xF742; break;
        case 2:
            dest = (unsigned short *)0xF744; break;
        case 3:
            dest = (unsigned short *)0xF746; break;
        case 4:
            dest = (unsigned short *)0xF748; break;
        case 5:
            dest = (unsigned short *)0xF74A; break;
        case 6:
            dest = (unsigned short *)0xF74C; break;
        case 7:
            dest = (unsigned short *)0xF74E; break;
    }

    if (decimal_point)
        *dest = (unsigned short)(lcd_number_data[number] | 0x0800);
    else
        *dest = lcd_number_data[number];
```

Figure 15: Function 'display_number()'

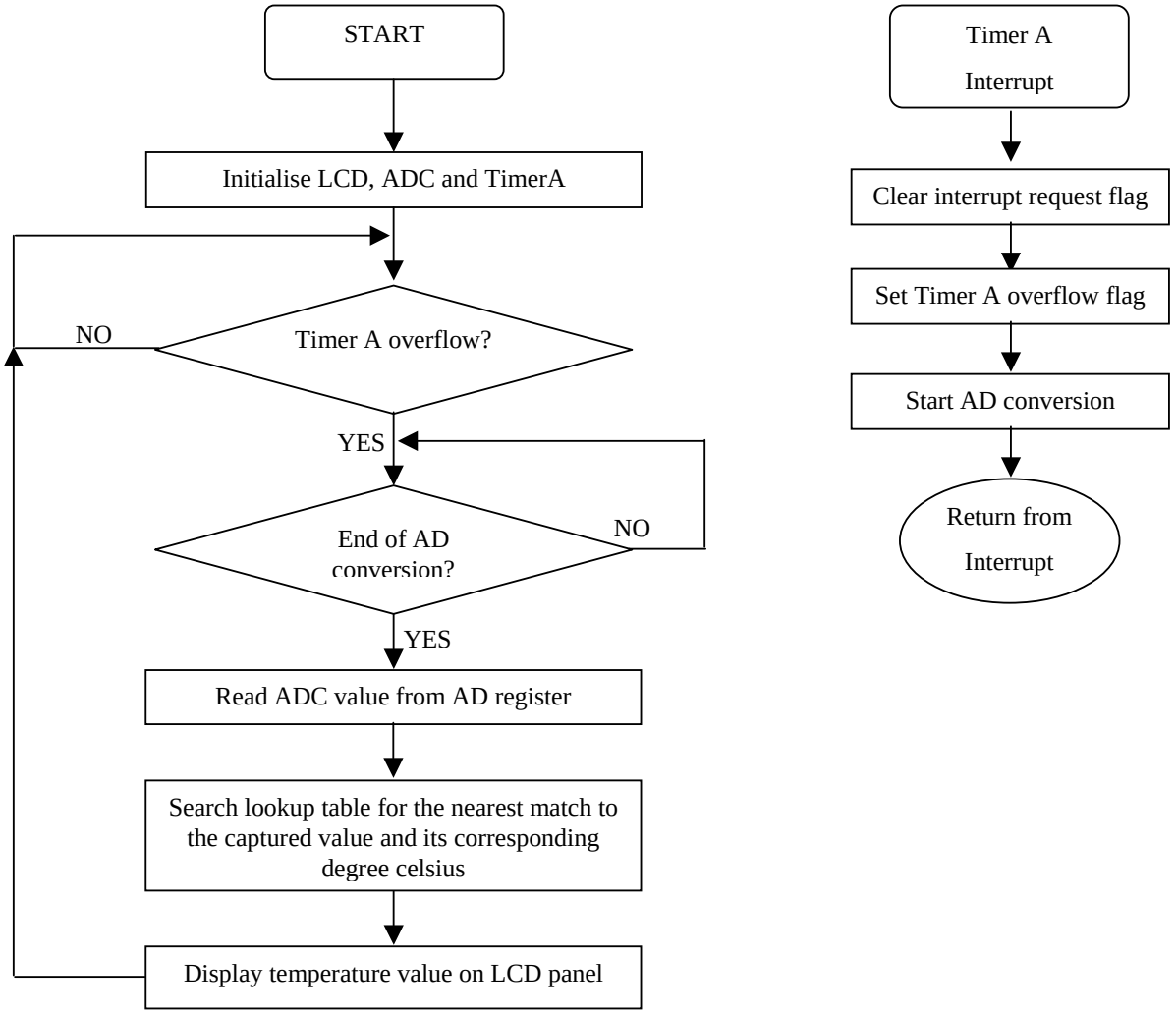
On previous page, Figure 15 shows the function **display_number()**. This function requires 3 parameters, **digit** to select which display segment (not the hardware pins) to turn on, **number** contains the character to be displayed and **decimal_point** to enable or disable that particular segment's decimal point. Firstly, pointer ***dest** will point to the respective LCD RAM's location according to the segment number contained in **digit**. Then if **decimal_point** is a positive non-zero integer, that segment's decimal point will be displayed together with the character contained in array **lcd_number_data[]**. Array **lcd_number_data[]** is a global variable defined in the file **lcd.h** as in our application. These values are worked out according to the LCD's specification, so you are required to search through hardware datasheet in case you want to know how they are derived. You'll need to create on your own a new file and named it as **lcd.h** and add the section in Figure 16 to this file. Remember to update all dependencies after you have added this file to the workspace directory.

```
const short lcd_number_data[39] = {0xE724,  //0. '0' : ABCDEF
                    0x0600,  //1. '1' : BC
                    0xC342,  //2. '2' : ABDEGK
                    0x8742,  //3. '3' : ABCDGK
                    0x2642,  //4. '4' : BCFGK
                    0xA542,  //5. '5' : ACDFGK
                    0xE542,  //6. '6' : ACDEFGK
                    0x0700,  //7. '7' : ABC
                    0xE742,  //8. '8' : ABCDEFGK
                    0x2742,  //9. '9' : ABCFGK
                    0x00E7,  //10. '*' : GHJKLN
                    0x0000,  //11. Blank: All segments OFF
                    0x6742,  //12. 'A' :
                    0xE442,  //13  'B' :CDEFGK
                    0xE100,  //14. 'C'
                    0xC642,  //15. 'D'
                    0xE142,  //16. 'E'
                    0x6142,  //17. 'F'
                    0xE540,  //18. 'G'
                    0x6642,  //19. 'H'
                    0x8118,  //20. 'I'
                    0xC600,  //21. 'J'
                    0x60A2,  //22. 'K'
                    0xE000,  //23. 'L'
                    0x6621,  //24. 'M'
                    0x6681,  //25. 'N'
                    0xE700,  //26. 'O'
                    0x6342,  //27. 'P'
                    0xE780,  //28. 'Q'
                    0x63C2,  //29. 'R'
                    };
```

Figure 16: LCD values decode

Software Operation

Below is a flowchart showing the flow of operation implemented by the code when the application is running.



4. Other Considerations

4.1 System Performance

One of the seroius considerations of any measurement system is the overall system error contributed by the various sources of errors. Let us start by establishing our overall system-performance requirements. Each component in the system will have an associated error, but it is to kept to a minimum within a certain limit. Since the ADC is a key component in the signal path, therefore necessary to consider errors contributed by the ADC. For the ADC, let's assume that the conversion-rate, interface, power-supply, power-dissipation, input-range, and channel-count requirements are acceptable before we embark on our analysis of the overall system performance.

Accuracy of the ADC is dependent on several key specs, which include integral nonlinearity error (INL), offset and gain errors, and the accuracy of the voltage reference, temperature effects, and AC performance. Since we are not involved in AC sources, AC performance is not our concern here. The DC performance will commonly be better than the AC performance.

4.2 Methods of Analysis

There are two popular methods for determining the overall system error. They are the root-sum-square (RSS) method and the worst-case method. In the RSS method, the error terms are individually squared, added, and then the square root is taken. The RSS error budget is given by:

$$\text{Total system error} = \text{sqrt}(E_1^2 + E_2^2 + E_3^2 + \ldots + E_N^2)$$

where E_N represents the term for a particular circuit component or parameter. To achieve higher accuracy for this method, all error terms should be uncorrelated (no constant relation), which may or may not always be the case. In the worst-case error analysis, all error terms add. This method will achieve worst case error possible and guarantees the actual error will never exceed a this limit. The actual error is always less than this value. The measured error will most probably lies somewhere between the values given by these two methods, but is often closer to the RSS value.

Ultimately, it is up to the designer's error budget, typical or worst-case values for the error terms can be used. There are many factors to consider, the importance of that particular parameter, the standard deviation of the measurement value, the size of the error in relation to other errors, etc. So there really aren't standard rules that must be obeyed. For our case, we will use RSS method will suffice.

The overall system will have a total-error budget based on the summation of error terms for each circuit component in the signal path. Assumptions we make about the input of ADC(output of sensor) are that we are measuring a slow-changing, DC-type signal, and our operating temperature range is 0°C to 70°C with performance guaranteed from -20°C to 75°C and Avcc 2.7 to 5.5V for ADC(from specs). Therefore,

$$\text{Total system error} = \text{sqrt}[(\text{ADC overall error})^2 + (\text{pullup resistor error})^2 + (\text{thermistor error})^2]$$

$$= \text{sqrt}((0.78\%)^2 + (5\%)^2 + (5\%)^2)$$

$$= 7.11\ \%$$

Reference

1. H8/38024 Series, H8/38024F-ZTAT Hardware Manual

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Sep.03	-	First edition issued

Keep safety first in your circuit designs!

1. 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

1. 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.
2. 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.
3. 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).
4. 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.
5. 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.
6. The prior written approval of Renesas Technology Corporation is necessary to reprint or reproduce in whole or in part these materials.
7. 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.
8. Please contact Renesas Technology Corporation for further details on these materials or the products contained therein.