お客様各位

カタログ等資料中の旧社名の扱いについて

2010年4月1日を以ってNECエレクトロニクス株式会社及び株式会社ルネサステクノロジが合併し、両社の全ての事業が当社に承継されております。従いまして、本資料中には旧社名での表記が残っておりますが、当社の資料として有効ですので、ご理解の程宜しくお願い申し上げます。

ルネサスエレクトロニクス ホームページ（http://www.renesas.com）

2010年4月1日
ルネサスエレクトロニクス株式会社

【発行】ルネサスエレクトロニクス株式会社（http://www.renesas.com）

【問い合わせ先】http://japan.renesas.com/inquiry

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   標準水準： コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせないようお客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

注 1. 本資料において使用されている「当社」とは、ルネサスエレクトロニクス株式会社およびルネサスエレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

H8/300L Super Low Power (SLP)シリーズ

ADC を使用した温度計測

内容

本アプリケーションノートでは、温度計測で用いる H8/38024F SLP（Super Low Power）MCU の A/D 変換機能について説明します。温度センサの特性について述べるとともに、対応するソースプログラムの実行についても説明します。

はじめに

温度はアナログ量ですが、デジタルシステムにおいても、計測、制御、保護機能を実現するために温度を用いることがよくあります。マイクロコンピュータ（MCU）を用いて温度を計測することは、考え方としてはシンプルです。SLP マイクロコンピュータは 10 ビットの ADC を 8 チャネル内蔵しており、これを用いて温度センサ（アナログ出力型）の出力電圧を読み取ることができます。

温度を計測するセンサには様々なタイプがあります。一例はサーミスタ（温度に感応する抵抗）です。ほとんどのサーミスタは負の温度係数（NTC）をもち、温度が下がると抵抗値が上がります。受動温度計測センサのなかで、サーミスタが最も高い感度（変化温度に対する抵抗値の変化）を備えています。しかし、サーミスタの温度-抵抗値の特性は直線的ではありません。

動作確認デバイス

H8/38024F

目次

1. サーミスタの特性

代表的な NTC サーミスタファミリのデータを表 1 に示します。これは、BC Components 社のサーミスタ（代表的な NTC サーミスタ）のデータです。抵抗値は割合（R/R_{25}）で表されています。同一ファミリのサーミスタの多くは同じような特性で同一の温度-抵抗値特性を示します。25ºC での抵抗値（R_{25}）が 10K のこのファミリのサーミスタは、0ºC では抵抗値 28.1K、60ºC では抵抗値 4.086K となります。同様に R_{25} が 5K のサーミスタは、0ºC では抵抗値 14.050K となります。

表 1　代表的な NTC サーミスタのデータ

温度 ºC	R/R_{25}	温度 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

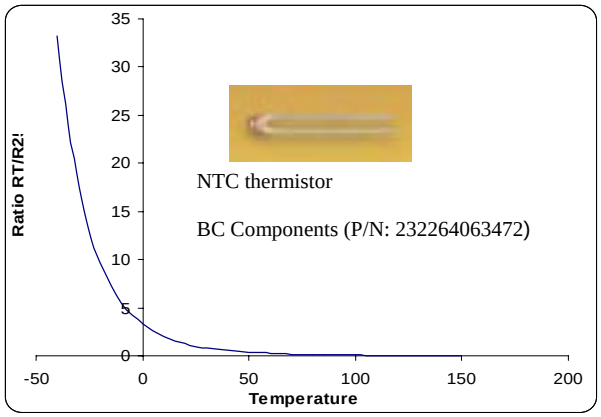


図 1　R_T/R_{25} 値と温度の関係 (NTC)

図 1 にサーミスタ特性のグラフを示します。抵抗値と温度の関係が直線的でないことがわかります。このサーミスタのデータは 10 度単位で示されていますが、サーミスタによっては 5 度単位あるいは 1 度単位で示されているものもあります。特性表の特定の 2 点間における温度の見当をつけなければならない場合もあります。特性グラフからこれを概算することも、抵抗値を直接計算することもできます。抵抗値の計算式は次のようになります。

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

ここで、T は絶対温度、A, B, C, D はサーミスタの特性によって異なる定数です。これらのパラメータは、サーミスタの製造元から提供を受けてください。

サーミスタにはサンプル間で抵抗値の再現性に一定幅の許容差を持たせています。その許容差は一般的に 1%から 10%の範囲で、使用する部品により異なります。調整が事実上不可能なアプリケーションにおいて部品交換できるよう設計されたサーミスタもあります。例えば、ユーザやフィールドエンジニアがサーミスタを交換しなければならないが、それを較正する手段がないようなアプリケーションでは（計測）機器を内蔵する必要があります。このタイプのサーミスタは通常のサーミスタより正確ですが、かなり高価になります。

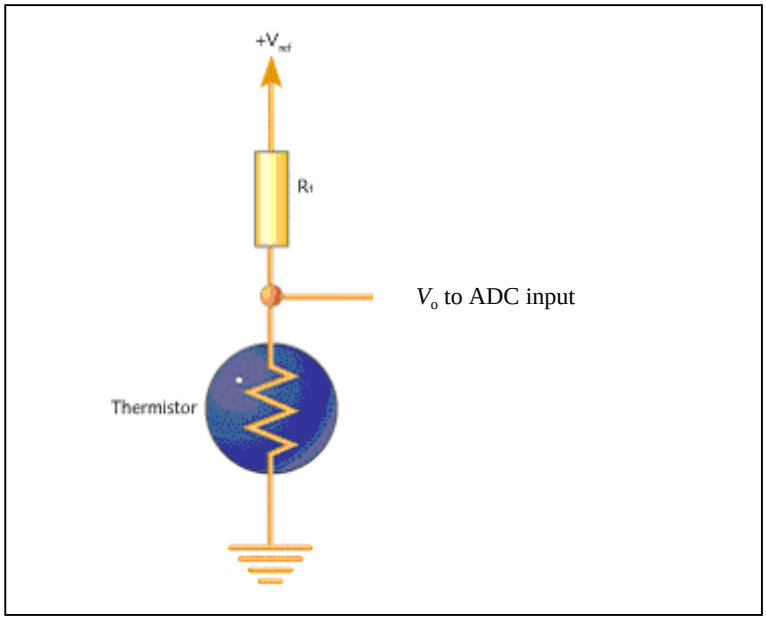


図 2　サーミスタ回路

図 2 に、マイクロコンピュータがサーミスタを用いて温度を計測する代表的な回路を示します。抵抗（R_1）はサーミスタを基準電圧にプルアップします。これは、一般に ADC 基準電圧と同じであり、ADC 基準電圧が 5V であれば V_{ref} も 5V になります。サーミスタと抵抗の組み合わせにより分圧器が構成され、サーミスタの抵抗値を変化させると接続部の電圧が変化します。この回路の精度はサーミスタの許容差、抵抗の許容差、基準電圧の精度で決まります。サーミスタは抵抗であるため、電流を流すと熱を発します。回路を設計する際には、必ずプルアップ抵抗を十分に大きくして過度な自己発熱を防止してください。これを怠ると、システムは周囲の温度ではなく、サーミスタが発する熱を計測してしまいます。温度に影響を与えるような熱をサーミスタが発するのに消費する電力量は、熱放散定数と呼ばれ、サーミスタ温度を周囲温度より 1ºC 上げるのに必要な電力はミリワット数です。熱放散定数は、サーミスタのパッケージ、リード寸法（リード付きデバイスの場合）、封止材料（サーミスタが封止されている場合）などの要因により異なります。

許容される自己発熱と、発熱を抑える抵抗の大きさは、要求する計測精度によって決まります。±5ºC 精度しか要求しないシステムは±0.1ºC の精度を要求するシステムよりも多くの自己発熱量を許容できます。全計測温度範囲にわたって自己発熱放散を抑えられるようプルアップ抵抗値を求めるようにしてください。一定の抵抗に対し、温度が変わるとサーミスタ抵抗値が変化するため熱放散値も変化します。

1.1 サーミスタの調整

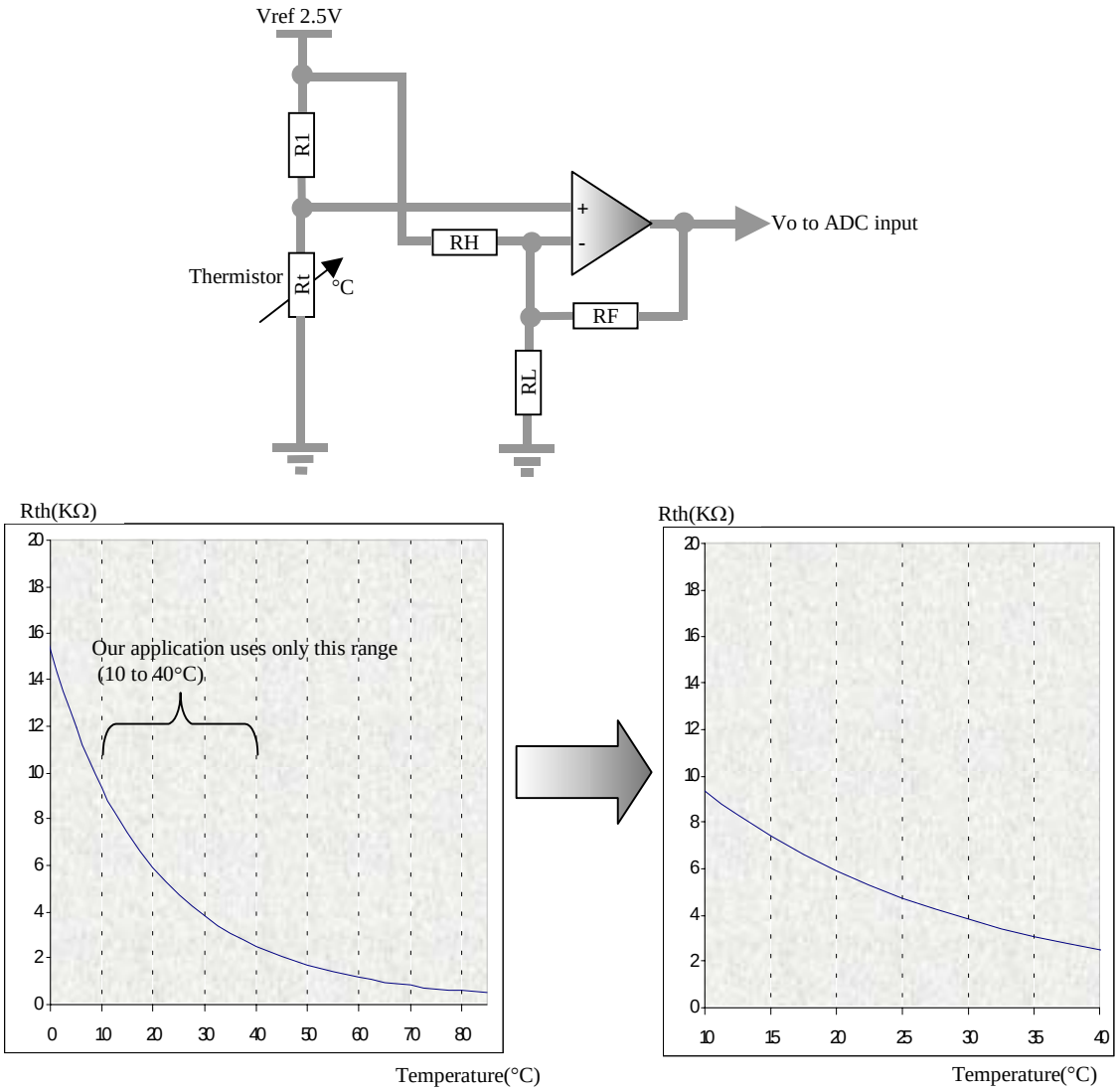


図 3　サーミスタの調整

適切な分解能を得るために、サーミスタ入力を調整しなければならないことがあります。図 3 に、温度範囲 10-40ºC を ADC 入力電圧範囲 0-3.3V に対応させる代表的な回路を示します。オペアンプの出力の計算式は次のようになります。

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

必要に応じてサーミスタを調整すれば、実際の抵抗値と温度の関係を示すグラフを作成することができます。サーミスタ特性が直線的でないため、各温度に対応する ADC 値をあらかじめソフトウェアが知っておく必要があり、そのためにこのグラフが必要です。グラフの精度（1 度刻みか 5 度刻み）は、アプリケーションが必要とする精度に依存します。

1.2 誤差の蓄積

いずれのサーミスタのアプリケーションでも、必要な精度にあわせてセンサなどの入力回路の部品を選ぶ必要があります。1%の抵抗しか必要としないアプリケーションもありますが、0.1%の抵抗が必要なアプリケーションもあります。いずれの場合も、抵抗、基準、サーミスタ自身も含んだ全部品における蓄積誤差の影響を示す計算表を作る必要があります。

手頃に入手できる部品から得られる精度よりも高い精度が必要な場合は、システムを作成したあとで較正しなくてはならないことがあります。アプリケーションによっては、この方法は選択できません。というのは回路ボードやサーミスタが現場で交換可能でなければならないからです。しかし、現場での交換が不可能な場合、または現場の技術者が温度を計測する別の手段をもっている場合は、温度と ADC 値の関係を示す表をソフトウェアで作成することができます。ソフトウェアが表を作成できるように、実際の温度（別のツールで計測した温度）を入力する手段が必要です。サーミスタが現場で交換可能でなければならないシステムでは、交換可能な部品（センサまたはアナログのフロントエンド全体）を工場で較正し、較正データをディスクなどの記憶媒体で提供することもできます。もちろん、部品を交換したときに較正データを適用する手段をソフトウェアで提供しなくてはなりません。概して、サーミスタは温度を計測するためにはコスト効率が良く、しかも使いやすい手段です。状況によっては、以下に示すような RTD や熱電対センサを選択することもできます。

表 2　各温度センサの特性

	サーミスタ	RTD	熱電対
温度範囲	-100°C〜450°C	-250°C〜900°C	-270°C〜1800°C
感度	**数**Ω/Ω/°C	0.00385 Ω/Ω/°C	**数十**μV/°C
精度	± 0.1°C	± 0.01°C	± 0.5°C
直線性	**三次以上の多項式あるいはそれに相当するルックアップテーブルが必要**	**二次以上の多項式あるいはそれに相当するルックアップテーブルが必要**	**四次以上の多項式あるいはそれに相当するルックアップテーブルが必要**
耐久性	**各種外装により耐久性向上。衝撃・振動による影響小。**	**振動に弱い**	**良好な絶縁体を用いると、耐久性が高まる**
応答性	1〜5 **秒**	1〜10 **秒**	1 **秒未満**
励起源	**電圧**	**電流**	**なし**
出力形態	**抵抗値**	**抵抗値**	**電圧**
寸法	0.1 x 0.1 **インチ**	0.25 x 0.25 **インチ**	**外径**∅ = **導線**∅ x 5
価格	$2〜$10	$25〜$1000	$1〜$50

2. ハードウェアの概要

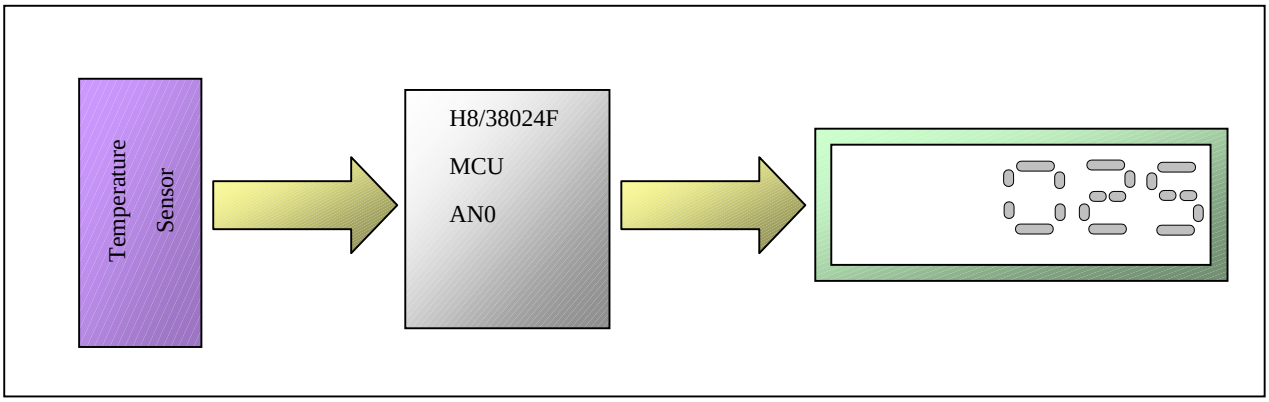


図 4　ハードウェアブロック図

基本的に、温度センサ回路には 3 つの部品があります。

a. マイクロコンピュータ　— 内蔵 ADC のポートを通じて温度センサの電圧を読みとる
b. 温度センサ　— 温度を電圧に変換する
c. LCD パネル　— ADC 変換結果を表示する

図 5　実際のハードウェア実装

ハードウェア回路（抵抗 -サーミスタ分圧器）：

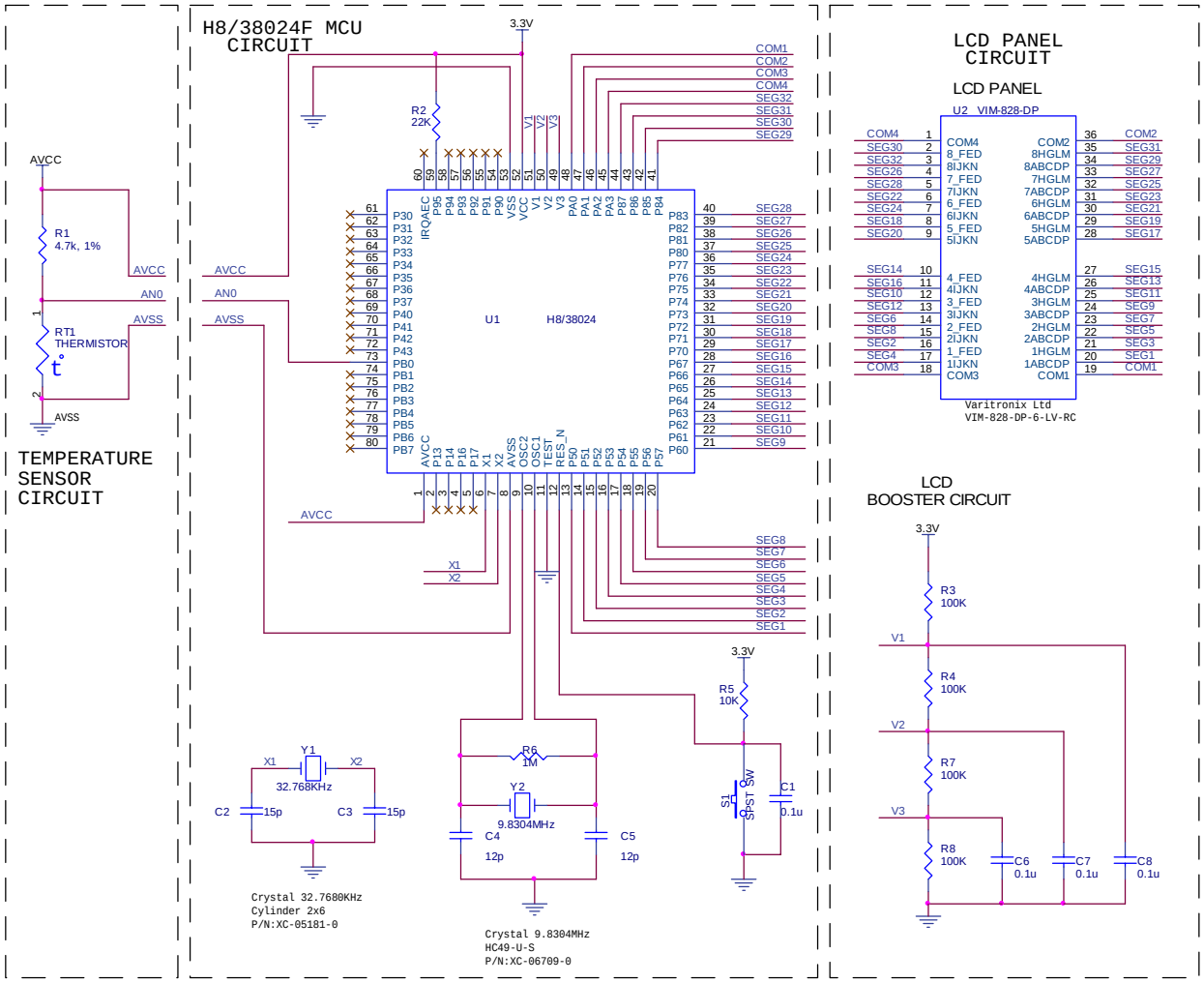


図 6　抵抗-サーミスタ分圧器を用いた温度計測器の概略図

図 6 に単純な温度計測器の概略図を示します。図の左に、サーミスタ-抵抗分圧器の回路があります。周囲温度にしたがってサーミスタの抵抗値が変化し、接続部の電圧も変化します。ADC 入力チャネル 0（AN0）が接続部の電圧を取り込みます。A/D 変換器に供給される AVCC と AVSS は、それぞれアナログ電源とアナロググランドです（詳細は H8/38024 ハードウェアマニュアルを参照してください）。

図の中央に、このアプリケーションに必要な接続を施した H8/38024 MCU があります。AN0 はポート B の端子 0 とのマルチプレクスピンです。ポート B は 8 ビットの入力専用ポートで、内蔵 A/D 変換器の 8 つの入力チャネルに対応しています。図の下部に、サブクロック発振器、システムクロック発振器、システムリセット回路が左から右に向かって並んでいます。ポート 5, 6, 7, 8（SEG0～SEG32）とポート A（COM1～COM4）は、32x4 LCD のセグメント端子とコモン端子とのインタフェースに用いています。

右側に、LCD パネル接続の詳細があり、下部には LCD 駆動用の昇圧回路があります。英数字を表示するには、LCD RAM（アドレス H’F740～H’F74F）にワードデータを書き込んで必要なセグメントをオンにしなくてはなりません。昇圧回路は、チップの電源供給能力では足りないような大きなパネルを用いる場合に必要です。

3. ソフトウェアの概要

必要な接続が完了したら、アプリケーションのプログラムを作成します。まず、ADC 値をデコードするためのルックアップテーブルを作成する必要があります。サーミスタ値は、サーミスタのデータシートから 5°C 単位で抽出します。次に、RT1 と AVcc 値を一定にして、0°C～20°C の温度範囲で表 3 を作成します。Vin（AN0 の電圧）を分圧器の法則を用いて計算し、対応する ADC 値を求めます（0V を 0_{10} に、3.3V を 1023_{10} に対応させる）。

表 3　ADC 値に対する温度マップ

温度 (°C)	サーミスタ R1 (Ohms)	RT1 (Ohms)	AVcc (V)	AN0 の Vin (V)	ADC 値 (10 進)	ADC 値 (丸め後)
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

しかし、5°C 単位でなく 1°C 単位のデータが必要です。実際に近い概算値を得るために、5°C 毎の ADC 値を 1°C 単位の ADC 値に分解配置する必要があります。これを、直線補間と呼びます。これによって、以下の値が得られます。

表 4　温度マップの拡張

温度 (°C)	サーミスタ R1 (Ohms)	RT1 (Ohms)	AVcc (V)	AN0 の Vin (V)	ADC 値 (10 進)	ADC 値 (丸め後)
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

図 7 に、このアプリケーションソースプログラムの始めの部分を示します。上記のルックアップテーブルを配列 **temperature[]**（100 要素）に実現しました。配列のインデックスが実際の温度（°C）で、配列要素が対応する変換後の ADC 値です。残りの部分は、関数プロトタイプと変数の宣言です。自己記述的なものもありますが、プログラムの他の部分を見せることでより明解になるものもあるでしょう。図 7 からわかるように、変数はすべて最初に 0 に初期化して、不定値が入らないようにします。

```
#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()
```

図 7　関数プロトタイプと宣言

```
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
    }
}
```

図 8　'main()'関数

図 8 に関数 **main()**を示します。LCD、ADC、タイマ A が初期化されます。周期的に温度計測をするためにタイマ A を使用するので、オーバフローのタイミングによって必要なループの数を計算し周期を決めなくてはなりません。これは、関数 **nloopTA()**で行ないます。このアプリケーション例では、サーミスタが安定するのに 1 秒かかるため、周期を 1 秒とします。ここでの説明の主眼はタイマではないので、タイマ A はデフォルトでオーバフロー割り込みを用いることにします。

ステータスフラグ **goRead** を用いて、1 秒ごとの温度計測の合図をします。ステータスフラグ **goRead** は、あらかじめ決められたループ数に達したときに、タイマ A 割り込み処理ルーチン **taint()**で設定します（図 10）。同時に、**start_adc()**が呼び出され、ADC 変換を開始します。変換中には、変換が終了したことによって ADSF フラグがクリアされたかどうかを、以下の文によりビットごとの AND 演算子（&）を用いてチェックします。

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

‘0x80’は 10 進の 128 を表す 16 進数で、ADSR レジスタのビット 7（MSB）に対応します。次に、ADRRH と ADRRL レジスタ値を 6 ビット左に論理シフトして、ADC 値を変数 **ADC_value** に格納します。

次に、‘for’ループで、取り込んだ ADC 値が当てはまる温度範囲を配列全体から検索します。そして、最も近い摂氏温度（配列インデックス）に丸めます。次に、適切なフォーマット変換を行なって、温度を LCD に表示します。

```
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;
}
```

図 9　‘init_adc()’関数

図 9 に関数 **init_adc()**を示します。クロックを選択する **conv_period**、外部トリガ信号により A/D 変換の開始を許可／禁止する **Ext_TRG**、8 つのチャネルを選択する **Input_CH** の 3 つのパラメータリストがあります。これらに論理 OR を行なって、AMR レジスタに設定します。詳細は、H8/38024 ハードウェアマニュアルを参照してください。

```
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;
}
```

図 10　関数‘init_lcd()’

図 10 に、LCD パネルを初期化するための関数 **init_lcd()**を示します。パラメータは受け取りません。LCD RAM の先頭アドレスを指すようにポインタ***dest** を設定します。‘for’ループで LCD RAM をクリアします。LCD の詳細については、アプリケーションボードマニュアル APPBD-3800 と H8/38024 ハードウェアマニュアルを参照してください。

```
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
}
```

図 11　関数'taint()'

図 11 に割り込み処理ルーチン **taint()**を示します。**main()**と同じファイルで定義されています。このサブルーチンは、タイマ A の割り込みを用いるときに必要で、タイマ A がオーバフローするたびにこのサブルーチンが呼び出されます。最初に、割り込み要求フラグ **IRRTA** をクリアします。**nOVF** はオーバフローが発生した回数をカウントし、計算した必要なループ（**count**）と比較します。必要な場合は、ステータスフラグ **goRead** を設定し、A/D 変換を開始します。実際は、全ての割り込み処理ルーチン組み込み用のファイルがすでに用意されています。これは、**intprg.c** ファイルです。このファイルの外で割り込み処理ルーチンを定義する場合は、図 12 に示すプログラム文 2 行を追加する必要があります。

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
}
```

図 12　割り込み処理ルーチンの組み込み

```
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
}
```

図 13　関数'start_adc()'

図 13 に関数 **start_adc()**を示します。関数 **taint()**に呼び出されると、ADSF をセットして A/D 変換を開始するか、または ADSF をクリアして変換を停止します。

```
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
}
```

図 14　関数‘init_timerA()’

図 14 に関数 **init_timerA()**を示します。タイマ A を初期化して、アプリケーションに最適なサブクロックタイミングを用いるようにします（1 秒、0.5 秒、0.25 秒、31.25 ミリ秒）。IENR1 レジスタの IENTA ビットをセットして、タイマ A の割り込み要求が受け付けられるようにしてください（H8/38024 ハードウェアマニュアルを参照してください）。関数 **set_imask_ccr()**は **machine.h** ライブラリでデフォルトで提供されるもので、定義する必要はありません。割り込みフラグを変更・許可しているときに、システム割り込みを一時的に禁止・マスクすることができます。これを行なわないと、割り込みを許可する前にシステム割り込みが発生し、一連の割り込み反応が発生する可能性があります。これにより、システムが不安定になり、データの不一致が起きる場合があります。

```
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];
}
```

図 15　関数‘display_number()’

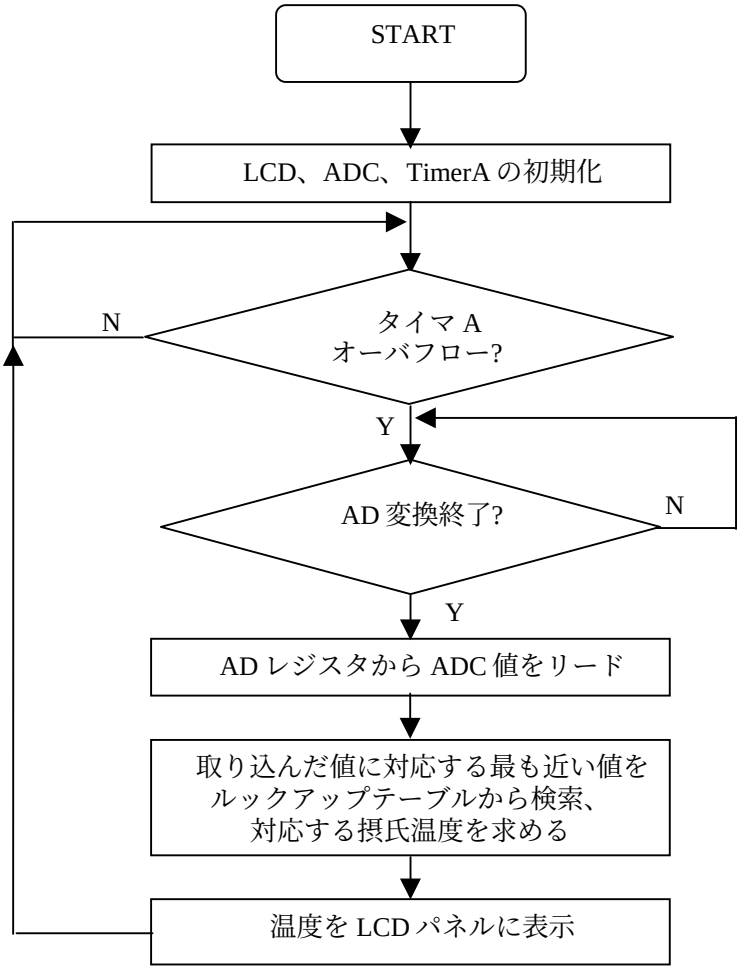
図 15 に関数 **display_number()**を示します。この関数は点灯する表示セグメント（ハードウェアの端子ではない）を選択する **digit**、表示する文字を格納した **number**、特定のセグメントの小数点を表示／非表示にする **decimal_point** の 3 つのパラメータを必要とします。最初に、**digit** が示すセグメント番号にしたがい、ポインタ***dest** が該当する LCD RAM のアドレスを指します。次に、**decimal_point** が正のゼロ以外の整数の場合は、セグメントの小数点を、配列 **lcd_number_data[]**が格納している文字とともに表示します。配列 **lcd_number_data[]**は、このアプリケーション例のファイル **lcd.h** で定義されているグローバル変数です。この値は LCD の仕様にしたがって求めるもので、算出法を知りたい場合は、ハードウェアのデータシートを参照する必要があります。新しいファイルを作成して **lcd.h** という名前をつけ、図 16 に示すセクションをこのファイルに追加する必要があります。このファイルをワークスペースディレクトリに追加した後、必ずすべての依存関係を更新してください。

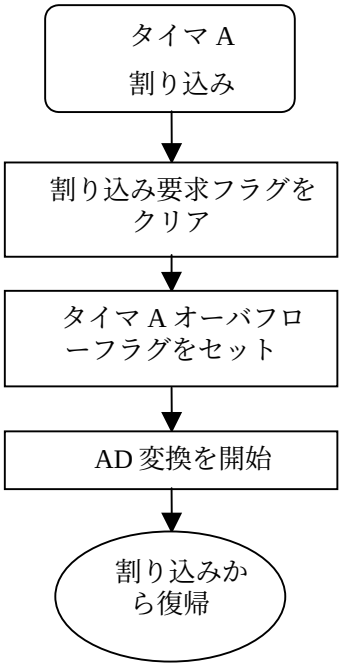
```
const short  lcd_number_data[39] = {0xE724,     //0. '0' : ABCDEF
                    0x0600,             //1. '1' : BC
                    0xC342,             //2. '2' : ABDEGK
                    0x8742,             //3. '3' : ABCDGK
                    0x2642,             //4. '4' : BCFGK
                    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'
                    };
```

図 16　LCD 値のデコード

4. ソフトウェアの動作

アプリケーションを実行したときの、ソフトウェアの動作の流れを示すフローチャートを示します。





5. その他の検討事項

5.1 システム性能

計測システムで重要な検討事項のひとつに、さまざまな誤差要因に起因するシステム全体の誤差があります。最初に、システム全体の性能の必要事項から考えてみます。システムの各部品の誤差は相互に関連しますが、一定の範囲で最小限に抑えられています。信号経路では ADC が重要な部品であるため、ADC による誤差の検討が必要です。システム全体の性能の解析を始める前に、ADC において、変換速度、インタフェース、電源供給、消費電力、入力範囲、チャネル数の必要事項が容認できる値であると仮定します。

ADC の精度は、非線型誤差（INL）、オフセットとゲインの誤差、基準電圧の精度、温度の影響、AC 性能などのさまざまな重要な特性によって決まります。AC 電源については関わらないので、ここでは AC 性能については考えません。DC 性能は、一般に AC 性能より良好です。

5.2 解析方法

システム全体の誤差を求める方法として、2 つの方法がよく使われています。二乗和平方根（root-sum-square：RSS）法と、ワーストケース法です。RSS 法では、誤差の項を個別に二乗し、加算し、次に平方根を求めます。RSS 法による誤差の概算は次のようになります。

$$\text{システム全体の誤差} = \text{sqrt}(E_1^2 + E_2^2 + E_3^2 + \ldots + E_N^2)$$

ここで、E_Nは特定の回路部品またはパラメータの項を表します。この方法で精度を高めるには、全ての誤差の項が相関関係を持たない（不変の関係を持たない）ことが必要ですが、常にそうなるとは限りません。ワーストケース法の誤差解析では、全ての誤差の項を加算します。この方法では、誤差は考えられ得る最大値となり、実際の誤差がこの値を決して越すことはありません。実際の誤差は、常にこの値より小さくなります。実測すると誤差は、ほとんどの場合、この 2 つの方法で求めた値の間におさまりますが、RSS 値に近い場合が多くなります。

最終的には、誤差の項に標準値を用いるか最悪値を用いるかは、設計者の判断しだいです。考慮すべき項目は多く、特定のパラメータの重要性、計測値の標準偏差、他の誤差に対する自誤差の大きさなどがあります。したがって、実際、従うべき標準的なルールはありません。このアプリケーション例では、RSS 法で充分と判断し、これを用います。

システム全体では、信号経路にある各回路部品の誤差項目の総和に基づいて計算された予定誤差が存在します。システム全体の誤差を計算するのに、次の通り仮定します。ADC 入力（センサ出力）に関して我々は、緩慢に変化する DC タイプの信号を計測していること、温度範囲は 0°C～70°C で性能を保証するのは-20°C～75°C であること、ADC 用の AVcc は 2.7～5.5V（仕様値）です。すると、システム全体の誤差は次のようになります。

$$\begin{aligned}\text{システム全体の誤差} &= \text{sqrt}[(\text{ADC 誤差の合計})^2 + (\text{プルアップ抵抗誤差})^2 \\ &\quad + (\text{サーミスタ誤差})^2] \\ &= \text{sqrt}((0.78\%)^2 + (5\%)^2 + (5\%)^2) \\ &= 7.11\ \%\end{aligned}$$

参考文献

H8/38024 シリーズ, H8/38024F-ZTAT ハードウェアマニュアル

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2003.09.19	—	初版発行

安全設計に関するお願い

1. 弊社は品質、信頼性の向上に努めておりますが、半導体製品は故障が発生したり、誤動作する場合があります。弊社の半導体製品の故障又は誤動作によって結果として、人身事故、火災事故、社会的損害などを生じさせないような安全性を考慮した冗長設計、延焼対策設計、誤動作防止設計などの安全設計に十分ご留意ください。

本資料ご利用に際しての留意事項

1. 本資料は、お客様が用途に応じた適切なルネサス テクノロジ製品をご購入いただくための参考資料であり、本資料中に記載の技術情報についてルネサス テクノロジが所有する知的財産権その他の権利の実施、使用を許諾するものではありません。
2. 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他応用回路例の使用に起因する損害、第三者所有の権利に対する侵害に関し、ルネサス テクノロジは責任を負いません。
3. 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他全ての情報は本資料発行時点のものであり、ルネサス テクノロジは、予告なしに、本資料に記載した製品または仕様を変更することがあります。ルネサス テクノロジ半導体製品のご購入に当たりましては、事前にルネサス テクノロジ、ルネサス販売または特約店へ最新の情報をご確認頂きますとともに、ルネサス テクノロジホームページ(http://www.renesas.com)などを通じて公開される情報に常にご注意ください。
4. 本資料に記載した情報は、正確を期すため、慎重に制作したものですが万一本資料の記述誤りに起因する損害がお客様に生じた場合には、ルネサス テクノロジはその責任を負いません。
5. 本資料に記載の製品データ、図、表に示す技術的な内容、プログラム及びアルゴリズムを流用する場合は、技術内容、プログラム、アルゴリズム単位で評価するだけでなく、システム全体で十分に評価し、お客様の責任において適用可否を判断してください。ルネサス テクノロジは、適用可否に対する責任は負いません。
6. 本資料に記載された製品は、人命にかかわるような状況の下で使用される機器あるいはシステムに用いられることを目的として設計、製造されたものではありません。本資料に記載の製品を運輸、移動体用、医療用、航空宇宙用、原子力制御用、海底中継用機器あるいはシステムなど、特殊用途へのご利用をご検討の際には、ルネサス テクノロジ、ルネサス販売または特約店へご照会ください。
7. 本資料の転載、複製については、文書によるルネサス テクノロジの事前の承諾が必要です。
8. 本資料に関し詳細についてのお問い合わせ、その他お気付きの点がございましたらルネサス テクノロジ、ルネサス販売または特約店までご照会ください。