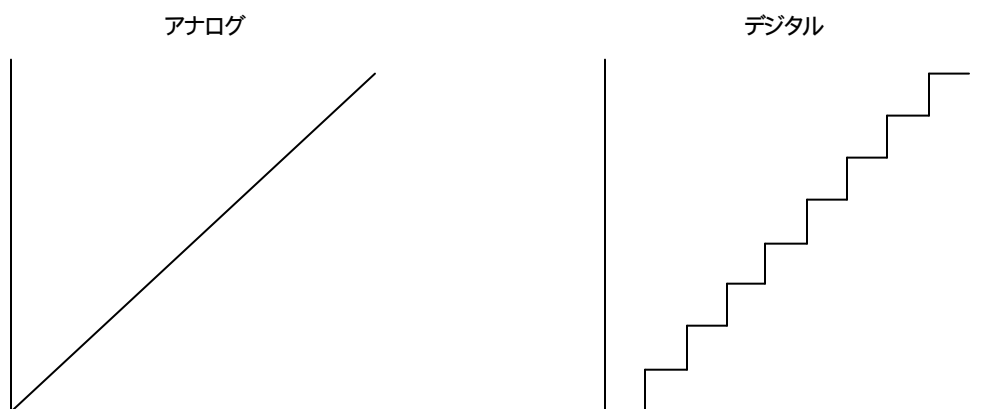


第16章 ADコンバーターを使う（温度計測）

ほとんどのH8マイコンは、ADコンバーターを内蔵しています。ADコンバーターは、アナログ量をデジタル量に変換するものです。

自然界のほとんどの物理量はアナログ量です。温度、光度、音量。これらはすべて連続量でありアナログ値ですから、そのままではマイコンで扱えません。

これらの連続量を1,2,3,4と数えられるデジタル量に変換してやらなければなりません。下図にその様子を示します。アナログ量が連続的な曲線であるのに対し、これをデジタルに変換すると階段状になります。



分解能

階段のステップが細かいほど、アナログに近い値に変換できます。この階段のステップの細かさを表す量として分解能があります。分解能は通常、8ビット、10ビット、12ビット、16ビット、というようにビット数で表されます。ビット数が大きいほど分解能は高く高性能です。

例えば2ビットの分解能のADコンバーターがあったとします。この場合2ビットで表されるデジタル量は0、1、2、3の4段階しかありません。つまり、階段はたったの4つです。温度で言えば、冷たい、ぬるい、暖かい、熱いと同じレベルです。しかし4ビットの分解能なら2の4乗=16段階になりますから、もっと細かく表現できるでしょう。

H8マイコン内蔵のADコンバーターは、10ビットの分解です。2の10乗は1024ですから、アナログ量を0から1023までの1024段階に置き換えることができます。これは実用上十分な分

解能と言えます。分解能がいくら高くてもノイズの影響が大きければ意味がありませんし、値段も高価になります。

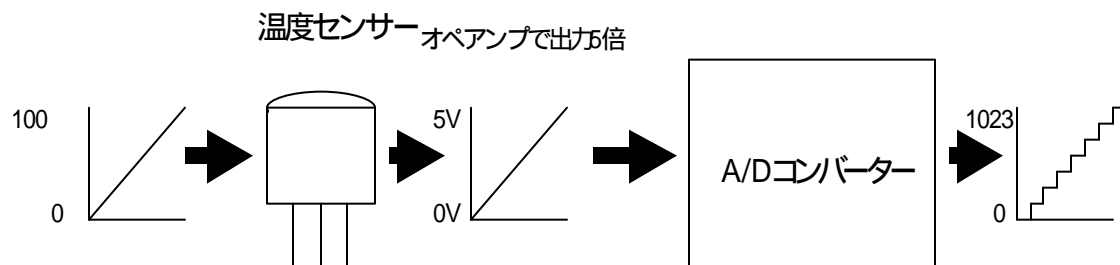
センサー

アナログ量には温度や光量などがありますが、ADコンバーターにこれらのアナログ量を入力してやるには、電圧に変換してやる必要があります。それをするのが、センサーと呼ばれるものです。

例えば、温度を電圧に変換するのが温度センサーです。

学習ボードの例で言えば、温度センサーが搭載されています。この温度センサーは0度～100度の値を0から1Vの電圧に変換します。マイコン内蔵のADコンバーターは、0～5Vの電圧を0から1023の数値に変換します。そこでオペアンプでセンサーの出力を5倍にします。

そうすると、例えば、マイコンがADコンバーターから1023という値を読み込んだとすれば、温度は100度ということが言えます。



プログラムの実行方法

YellowIDEを起動し、メニューの（ファイル）（プロジェクトを開く）を選択します。

[YellowIDE6]フォルダの中の[GSAMPLE2]フォルダの中の[TEMP]フォルダの[TEMP.YIP]を開いて下さい。

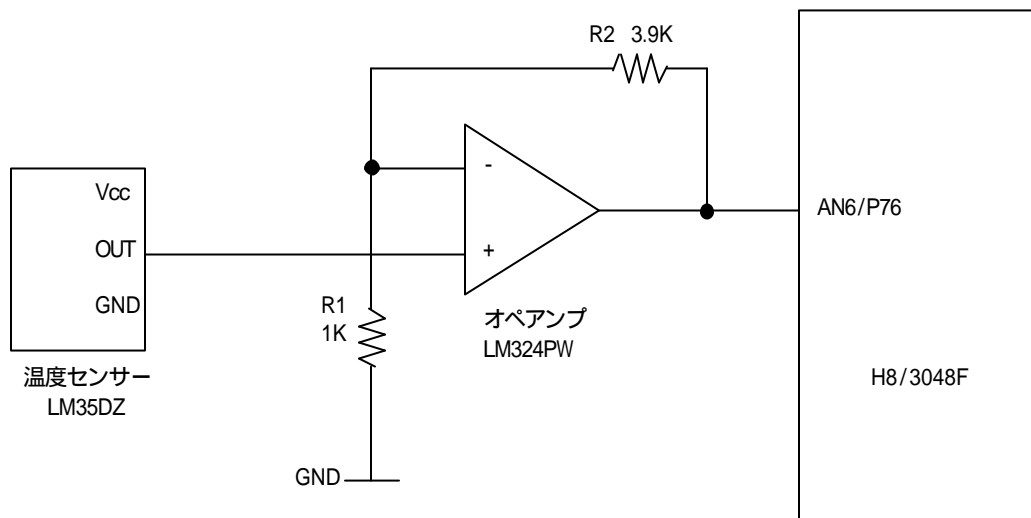
このプログラムを実行すると画面に

"リターンキーで現在の温度を表示します。"

と表示されますので、リターンキーを入力して下さい。

画面に温度が表示されます。温度センサーを手でつまんだり、ドライヤーで暖めるなどして温度変化を確認して下さい。

簡略回路図



回路の簡単な説明

温度センサーには、3端子のトランジスタのような形をしたLM35DZを使用しています。温度センサーはオペアンプを通して、マイコンのアナログ入力端子AN6に接続されています。AN7はポートのP76と兼用です。

オペアンプは、アナログ回路ではなくてはならない重要な素子です。アナログ信号をマイコンに入力するときも、必ずオペアンプを通して入力させます。オペアンプの働きとしては次の二つがあります。

- 1 電圧増幅用
- 2 バッファ、入力の保護

電圧増幅機能ですが、マイコンのアナログ入力電圧は0~5Vですから、センサーの出力をそれに合わせるために使用します。増幅率は、上記の回路では以下のように計算されます。

$$\text{増幅率} = (R1+R2) / R1 = (1K + 3.9K)/1K = 4.9 \quad \text{約5倍}$$

温度センサーの出力電圧は0 ~ 100 で0V~1Vですから、それを5倍して0V~5Vにしています。

オペアンプのもう一つの働きとして、入力の保護があります。マイコンのアナログ入力は0V~5Vの範囲ですので、それを大きく超える電圧が入力された場合にマイコンを壊す可能性

があります。オペアンプはそれ自身の電源電圧を超えた電圧は出力しませんので、マイコンを保護する働きがあります。

ADコンバーターの使い方

それでは、このマイコン内蔵のADコンバーターの使い方を学習します。やはり内蔵機能を利用する場合は、I/Oレジスタを使います。I/Oレジスタは、内蔵機能にアクセスするための窓です。

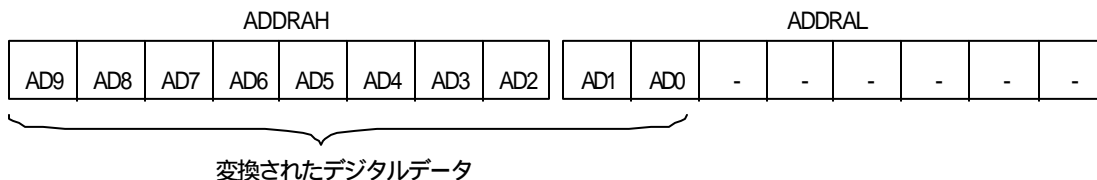
ADコンバーターの内蔵レジスタは以下のものがあります。ハードウェアマニュアルからレジスタ構成表を抜粋します。

アドレス	名称	略称	R/W	初期値
H'FFE0	A/DデータレジスタAH	ADDRAH	R	H'00
H'FFE1	A/DデータレジスタAL	ADDRAL	R	H'00
H'FFE2	A/DデータレジスタBH	ADDRBH	R	H'00
H'FFE3	A/DデータレジスタBL	ADDRBL	R	H'00
H'FFE4	A/DデータレジスタCH	ADDRCH	R	H'00
H'FFE5	A/DデータレジスタCL	ADDRCL	R	H'00
H'FFE6	A/DデータレジスタDH	ADDRDH	R	H'00
H'FFE7	A/DデータレジスタDL	ADDRDL	R	H'00
H'FFE8	A/Dコントロール/ステータスレジスタ	ADCSR	R/(W)	H'00
H'FFE9	A/Dコントロールレジスタ	ADCR	R/W	H'7F

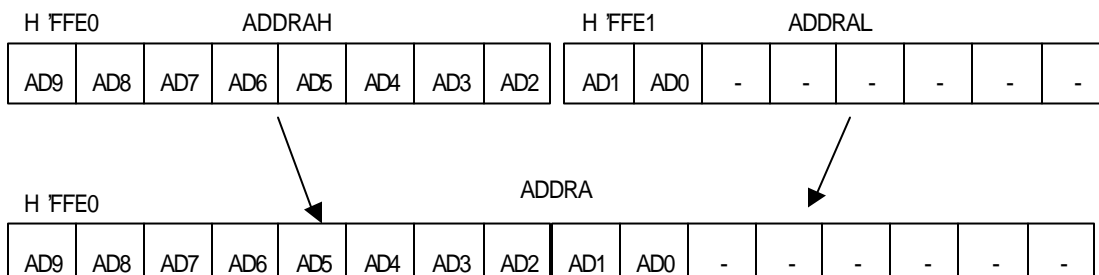
上から8つのレジスタはA/Dデータレジスタで、アナログから変換されたデジタルデータが格納されるレジスタです。なぜいっぱいあるかというと、内蔵ADコンバーターは8チャンネルを有しているからです。つまり、最大8本のアナログ入力端子からデータを取り込むことができます。

また、ADコンバーターは10ビットですから、8ビットのI/Oレジスタでは格納できません。そこで上位と下位に分けて、上位側がHの付くレジスタに格納され、下位側がLの付くレジスタに格納されます。

その分け方ですが、以下のように上位側に詰めた配置となっています。



A/Dデータレジスタは、このように8ビットレジスタ2本を使います。しかし上位側と下位側の番地が連続していますので、下図のように1本の16ビットレジスタとして扱うことができます。



8ビットレジスタADDRHとADDRLを一つの16ビットレジスタADDRとして扱うことができます。この場合、16ビットレジスタの番地は上位側の8ビットレジスタの番地に等しくなります。つまりH'FFE0です。

さて、内蔵ADコンバーターは8チャンネルあると言いましたが、ADデータレジスタは、16ビットのものが4本しかありません。実は、8チャンネルを二つのグループに分けて使うことになっているのです。

アナログ入力チャンネル		A/Dデータレジスタ
グループ0	グループ1	
AN0	AN4	ADDRA
AN1	AN5	ADDRB
AN2	AN6	ADDRC
AN3	AN7	ADDRD

この表を見ますと、チャンネル0(AN0)とチャンネル4(AN4)は同じADDRAレジスタを使うことがわかります。したがって、チャンネル4のデータを読み込むには、ADDRAレジスタを読み込む必要があります。

次にADCSRレジスタについて説明します。

ADCSR

ビット	7	6	5	4	3	2	1	0
	ADF	ADIE	ADST	SCAN	CKS	CH2	CH1	CH0
初期値	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

ビット7 ADF	説明
0	AD変換が終了すると1にセットされる。0にクリアするにはADFをリード後、0にライトする
1	AD変換が終了すると1にセットされる。

ビット6 ADIE	説明
0	AD変換終了による割込み(ADI)要求を禁止
1	AD変換終了による割込み(ADI)要求を許可

ビット5 ADST	説明
0	AD変換を停止
1	AD変換を開始。(AD変換が終了すると自動的に0になる)

ビット4 SCAN	説明
0	単一モード
1	スキャンモード

ビット3 CKS	説明
0	変換時間=266ステート(MAX)
1	変換時間=134ステート(MAX)

ビット2 CH2	ビット1 CH1	ビット0 CH0	説明
0	0	0	AN0
		1	AN1
	1	0	AN2
		1	AN3
1	0	0	AN4
		1	AN5
	1	0	AN6
		1	AN7

ADCSRレジスタは、ADコンバーターの様々な動作を制御するレジスタです。ADCSRレジスタで重要なのは以下のビットです。

ADST

AD変換を開始させるビットです。ここを'1'にセットすればAD変換が開始されます。

ADF

AD変換が終了すると、このビットが'1'にセットされます。

CH2 ~ CH0

チャンネルを選択するビットです。3ビットで8個のチャンネルを指定します。

AD変換の手順は以下の通りです。

- 1 CH2 ~ CH0でチャンネルを選択する
- 2 ADSTを1にセットして、AD変換を開始させる
- 3 ADFが1になるまで待つ
- 4 ADデータレジスタを読んで、AD変換データを取り出す。
- 5 ADFを0にクリアする

1と2は一つの命令で実現できます。

5の「ADFを0にクリアする」ですが、これはAD変換が終了した合図のADFが自動で0にクリアされないので、プログラムで0を書き込む必要があるためです。

サンプルプログラム解説

それでは、ADコンバーターの接続された温度センサーから温度を計測するプログラムを見てみましょう。

```
/* ヘッダファイル */
#include <stdio.h>           //puts、printf、fgetcのプロトタイプ宣言

/* レジスタ定義 */
/* AD変換器 アドレス定義 */
#define ADDRA (*(volatile unsigned short *)0xFFFFE0)
#define ADDRb (*(volatile unsigned short *)0xFFFFE2)
#define ADDRc (*(volatile unsigned short *)0xFFFFE4)
#define ADDRd (*(volatile unsigned short *)0xFFFFE6)
#define ADCSR (*(volatile unsigned char *)0xFFFFE8)
```

最初はヘッダファイルのインクルードと、レジスタの番地定義です。ADデータレジスタは本来8ビットのレジスタ2つですが、16ビットレジスタとしてアクセスできますので、そのように定義しています。16ビットレジスタとしてアクセスする場合は、番地は必ず偶数になることに注意して下さい。H8マイコンのプログラムでは、10レジスタにしてもメモリにしても、奇数番地に16ビットアクセスすることはあり得ません。もしあったら、それは間違いです。このように、16ビットアクセスする場合は偶数番地でなくてはならないというような条件を「境界条件」と言います。

```

/*****
****  温度入力関数                               ****
****          引数   なし                       ****
****          戻り値 現在の温度                 ****
*****/
int in_temp(void)
{
    unsigned int temp;

    ADCSR = 6;                                //チャンネル選択
    ADCSR |= 0x20;                             //変換開始
    while ((ADCSR & 0x80) == 0)               //変換終了まで待つ
        ;
    ADCSR &= ~0x80;                            //ADFクリア

    temp = ADIRC >> 6;                        //AD変換データ
    temp = 500L*(long)temp / 1024L;           //電圧×100
    temp = temp/5;                             //温度
    return temp;
}

```

ADコンバーターから、データを入力し、それを電圧に変換し、さらに温度に変換する関数です。このサンプルプログラムの核心部分です。

最初にチャンネルを選択します。温度センサーはチャンネル6に接続されていますので、ADCSRレジスタに6をセットします。

```
ADCSR = 6;
```

次に変換を開始します。ADCSRレジスタのビット5を'1'にセットすると、変換が開始されます。

```
ADCSR |= 0x20;
```

前記2つの文を一つにし、ADCSR = 0x26と書いても良いです。

さて、変換を開始してもすぐにデータを読み出してはいけません。変換には時間がかかりますので、変換が終了するまで待ちます。変換が終了するとADCSRレジスタの最上位ビット(ADFフラグ)が'1'にセットされますので、'0'の間はループするようにします。

```
while ((ADCSR & 0x80) == 0)    //変換終了まで待つ
    ;
```

ループを抜け変換が終了してもADFフラグは'1'のままです。0にクリアします。

```
ADCSR &= ~0x80;                //ADFクリア
```

次に実際にAD変換データを読み込みます。

```
temp = ADDRC >> 6;            //AD変換データ
```

ADチャンネルは6ですから、ADデータレジスタはADDRCになります。16ビットのレジスタにデータが格納されていますが、データ自身は10ビットです。しかも左詰めで格納されていますので、右に6ビットシフトします。

この時点で変数tempの中には0~1023のデータが格納されます。これを最初に電圧に変換します。0~1023の値を0~5Vに変換しますが(正確には0V~4.99V)、0~5Vでは値が小数になってしまいますので、電圧を100倍にした値(0~500)に変換することにします。

```
temp = 500L * (long)temp / 1024L;    //電圧 × 100
```

500をかけて1024で割ります。この演算の順番が大変に重要です。先に1024で割ってはいけません。

```
× temp = (long)temp / 1024L * 500L;
```

先に1024で割ってしまうとtemp/1024の項は0.xxxxxという1未満の小数になります。整数型の演算ですからこの項は0になってしまいます。ですから、この式は常に答えが0になってしまいます。

また、変数tempの前に(long)を付けています。定数の後にはLを付けています。これは型変換です。変数tempはintで定義されています。また、定数は通常はint型です。

イエローソフトのH8用のCコンパイラはintは16ビットになります。したがって格納できる数値は0～65535の範囲です。もし型変換をしなかったらどうなるでしょう。

```
temp = 500 * temp / 1024;
```

この式は16ビットで計算されます。例えば計算の途中であっても一回でも、桁あふれをおこしたら上位のビットは失われてしまいます。この例では

```
500 * temp
```

が桁あふれをおこす可能性があります。なぜなら、tempの最大値は10ビットデータですから1023です。

```
500 * 1023 = 511,500
```

です。したがって65535の範囲に収まりません。上位のビットデータは失われ、間違った答えを出します。そこでlong型に変換して、桁あふれを起こさないように配慮しています。

さて、この段階でtempには電圧を100倍した値が格納されています。つまり0～500の値が格納されています。次にこれを温度に変換します。

この温度センサーは、幸いにも温度と電圧が比例関係にあります。しかも0 のとき0Vで、100 のとき1Vです。したがって、tempを5で割ればそのまま温度になります。非常に分かりやすい関係にあります。温度センサーの中には指数関係であったり、オフセットが付いているものもあります。

```
temp = temp/5;           //温度
```

これですべての演算が終了です。

```

/*****
*****   温度センサーテストプログラム           *****
*****/
void main(void)
{
    puts("温度センサーテストプログラム");

    while (1) {
        int data;
        printf("リターンキーで現在の温度を表示します。");
        fgetc(stdin);
        data = in_temp();           //温度入力
        printf("現在の温度=%d¥n", data);   //温度表示
    }
}

```

最後はメインの関数です。パソコンのリターンキーを入力すると、温度を読みとり温度を表示するようになっています。

課題

このサンプルプログラムでは、リターンキーの入力でもって、温度の表示を更新していません。これをタイマー割り込みによって、1秒間で自動更新するように変更せよ。