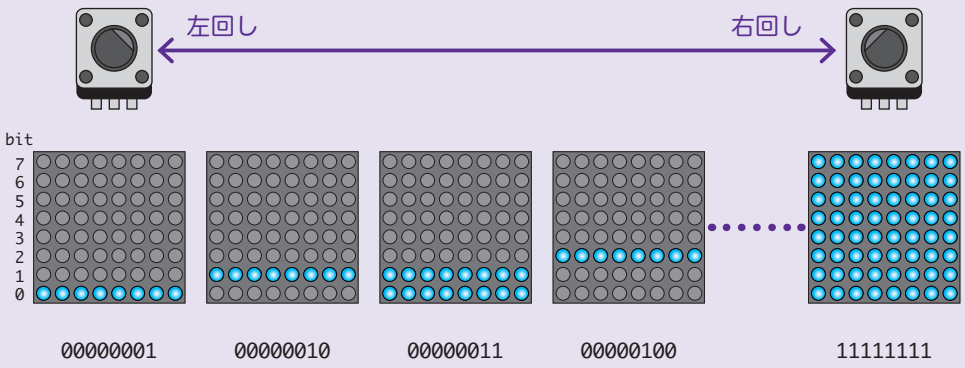
18 STEP

A/D 変換しよう

STEP18 では、STEP17 で学習した A/D 変換を実際行ってみます。
アナログ電圧をデジタル値にして出力させてみましょう。

課題　18-1

可変抵抗器の回転角度を A/D 変換し、入力値をドットマトリクス LED に 2 進数表示で出力させましょう。



STEP 18

A/D 変換しよう

18.1　可変抵抗器とは

可変抵抗器は、物理的変位に対して電気的な抵抗値が変化する入力デバイスです。昔からオーディオ機器のボリューム調整に多く使われていたことから「ボリューム」とも呼ばれます。本キットの可変抵抗器はツマミの回転によって 0 ～ 10kΩ まで変化します。

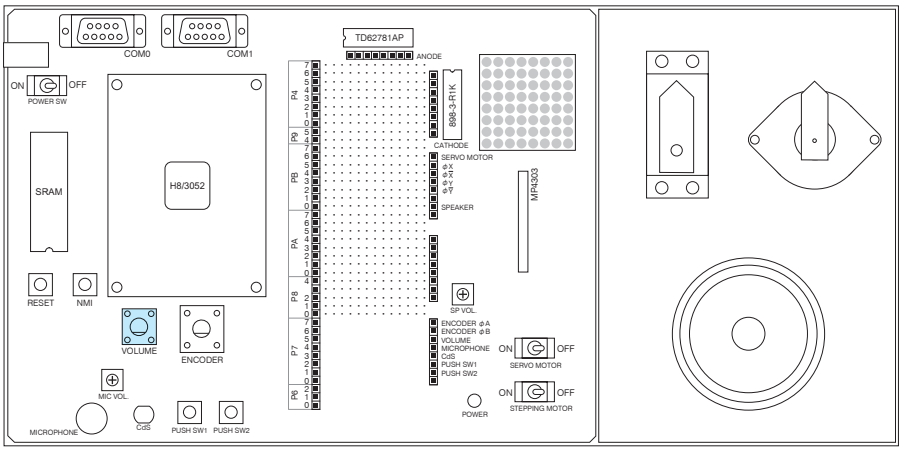


図 18-1　可変抵抗器

可変抵抗器のツマミを回すと、マイコン端子の入力電圧が変化します。
可変抵抗器を右回転すると、抵抗値は小さくなり、入力電圧は高くなります (最大 5V)。
可変抵抗器を左回転すると、抵抗値は大きくなり、入力電圧は低くなります (最小 0V)。

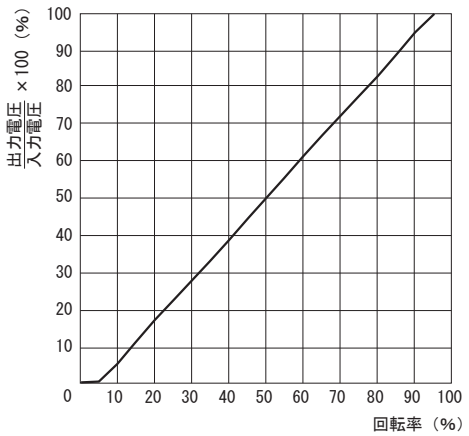


図 18-2　特性グラフ

※ 特性グラフは本キットに使われている可変抵抗器のものです。

本キットの可変抵抗器は図 18-3 のように配線して使います。
マイコン端子はアナログ入力端子です。

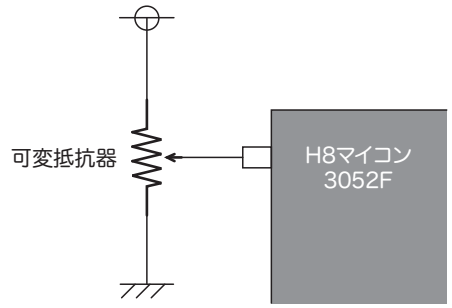


図 18-3　可変抵抗器の接続回路

STEP18 では、アナログ入力端子 AN_2 を使って A/D 変換を行うことにします。

しかし、アナログ入力端子 AN_0 ～ AN_7 はポート 7 の入力専用端子 $P7_0$ ～ $P7_7$ と共用になっています。

初期状態でポート 7 は入力専用端子 $P7_0$ ～ $P7_7$ なので、ADCSR のチャネル設定で、アナログ入力端子として使えるようにする必要があります。

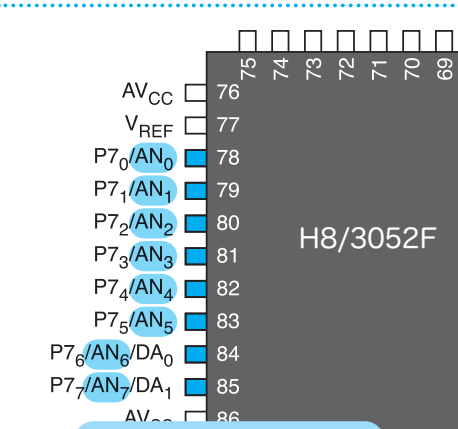


図 18-4　マイコン端子

それでは、可変抵抗器と A/D 変換器を追加配線しましょう。AN_2 は $P7_2$ と共用でしたね。

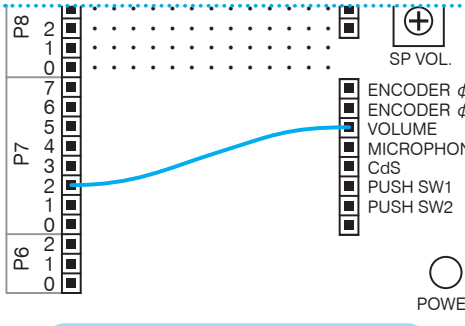


図 18-5　可変抵抗器の配線図

STEP 18 A/D 変換しよう

次にフローチャートを考えて、プログラムの流れを整理してみましょう。

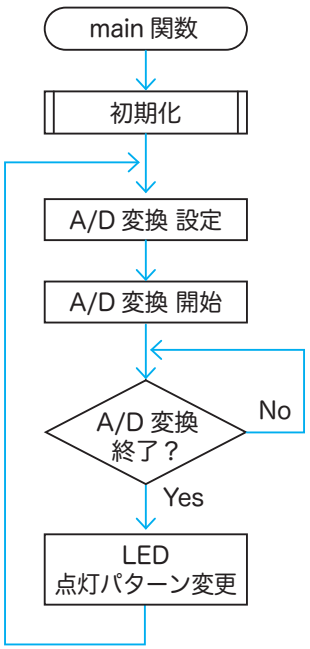


図 18-6　フローチャート

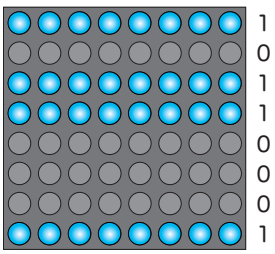
可変抵抗器によるアナログ電圧を A/D 変換してデジタル値にし、ドットマトリクス LED のポート 4 に書き込んで点灯させるという流れです。
LED の横行単位の点灯制御は、カソード側の PB を全列 L（0）にしておき、P4 の H/L（1/0）で制御すればいいでしょう。P4 の入力は、A/D 変換後の有効桁を 8 ビットにすれば 0x00 ～ 0xFF になるのでそのまま使えます。

プログラム例　18-1

```
/************************************************************
    製作者　アドウィン
    解説　　可変抵抗による端子電圧の変化
            →　A/D 変換し ドットマトリクス LED のアノードに格納
            →　点灯パターンによるデジタル値の確認
************************************************************/
#include <3052f.h> // 3052F 固有の定数

/*
 *  初期化関数
 */
void initIO(void)
{
    P4.DDR = 0xFF; // 出力 LED 横行
    PB.DDR = 0xFF; // 出力 LED 縦行

    P4.DR.BYTE = 0x00; // L アノード
    PB.DR.BYTE = 0x00; // L カソード
}

/*
 *  main 関数
 */
int main(void)
{
    initIO();             // 初期化関数の呼び出し
    AD.CSR.BYTE = 0x02;  // A/D 変換設定　AN2 使用

    while (1)
    {
        AD.CSR.BIT.ADST = 1;        // A/D 変換開始
        while (AD.CSR.BIT.ADF == 0) // A/D 変換終了待ちループ
            ;
        P4.DR.BYTE = (AD.DRC >> 8); // A/D 変換値を LED 点灯行に反映
    }

    return 0;
}
```

STEP 18 A/D 変換しよう

プログラム 18-1 を組み終わったら、実行してみてください。
可変抵抗器を回すと、ドットマトリクス LED の点灯パターンが変化していくと思います。ドットマトリクス LED の点灯パターンが A/D 変換値そのものです。点灯しているビットは 1、消灯しているビットは 0 です。



例えば、左図の点灯パターンは、1011 0001 と読み取れます。A/D 変換値 10 ビットの下位 2 ビットを誤差として切り捨てて 8 ビットを使っています。$2^8 = 256$ なので、0 ～ 5V が 0 ～ 255 に変換されていることになります。
1011 0001 は 10 進数で 177 ですから、このときのアナログ電圧値は 3.47V 程度と計算できます。

では、main 関数内の処理を説明していきます。

```
int main(void)
{
    initIO();             // 初期化関数の呼び出し
    AD.CSR.BYTE = 0x02;  // A/D 変換設定   AN2 使用

    while (1)
    {
        AD.CSR.BIT.ADST = 1;         // A/D 変換開始
        while (AD.CSR.BIT.ADF == 0) // A/D 変換終了待ちループ
            ;
        P4.DR.BYTE = (AD.DRC >> 8); // A/D 変換値を LED 点灯行に反映
    }

    return 0;
}
```

```
AD.CSR.BYTE = 0x02;  // A/D 変換設定　AN2 使用
```

単一モードで AN_2 を使うので、ADCSR の CH2、1、0 に「０１０」を設定します。
その他のビットは 0 のままでいいので、ADCSR には 0000 0010 = 0x02 を書き込んでいます。

CH2	CH1	CH0	単一モード
0	0	0	AN_0
		1	AN_1
	1	0	AN_2
		1	AN_3
1	0	0	AN_4
		1	AN_5
	1	0	AN_6
		1	AN_7

ADCSR

ビット：	7	6	5	4	3	2	1	0
	ADF	ADIE	ADST	SCAN	CKS	CH2	CH1	CH0
設定値：	0	0	0	0	0	0	1	0

```
AD.CSR.BIT.ADST = 1;  // A/D 変換開始
```

A/D スタートビット（ADCSR の bit5）を 1 にして A/D 変換を開始します。

```
while (AD.CSR.BIT.ADF == 0)  // A/D 変換終了待ちループ
    ;
```

A/D 変換は高速で処理されますが時間 0 ではないため、値を読み取る前に A/D 終了待ちを入れておくと確実です。A/D 変換の終了を示すビット ADF をループで確認し、A/D 変換が完了する（ADF が 1 になる）まで何もしない処理を繰り返すようにしています。

STEP 18 A/D 変換しよう

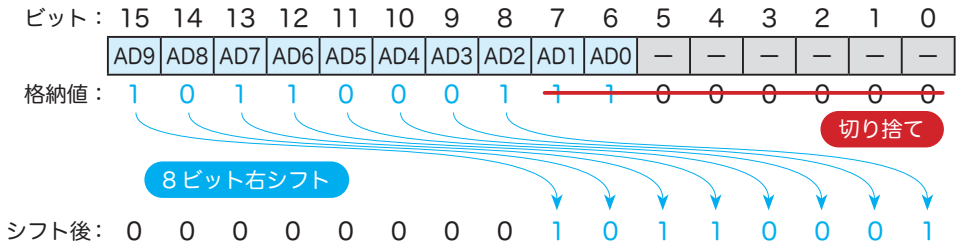
```
P4.DR.BYTE = (AD.DRC >> 8);  // A/D 変換値を LED 点灯行に反映
```

ここが少し頭をひねるところだと思います。
処理自体は、A/D 変換された値を８ビット右にシフトしてポート４に出力しているのですが「８ビット右にシフトする」意味を説明します。

AN2 に印加されたアナログ電圧は、A/D 変換され ADDRC に格納されます。例えば ADDRC に格納されているデジタル値が 1011000111 の 10 ビットとします。そのとき ADDRC は下図のようになります。

ビット：	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	AD9	AD8	AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0	—	—	—	—	—	—
格納値：	1	0	1	1	0	0	0	1	1	1	0	0	0	0	0	0

この課題では、P4DR の８ビットにするため、ADDRC 全 16 ビットの下位８ビットを切り捨てています。具体的にはプログラムで AD.DRC を８ビット右にシフトすれば、下位８ビットは消去され、空欄の上位ビットには０が入ります。
その結果、P4DR に書き込む値は 1011 0001 の８ビット、16 進数で 0xB1 になります。



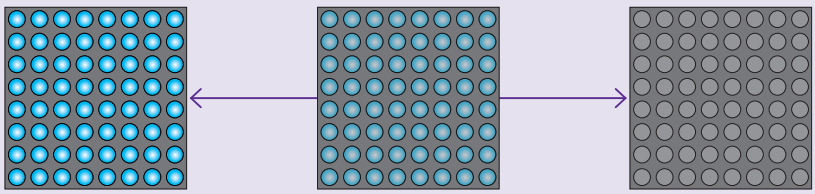
【 右シフト演算子 >> 】

`x >> n;`	x のビットパターンを右に n ビットずらす
`x >>= n;`	x = x >> n と同じ

プログラム 18-1 について理解できましたか？
では、このプログラム 18-1 を応用して、今度は A/D 変換したデジタル値をデューティ比に利用してみましょう。

課題　18-2

可変抵抗器の回転角度に連動して、ドットマトリクス LED の明るさを変化させましょう。



「ドットマトリクス LED の明るさを変化」させるには、H と L の時間を変化させるということです。分からなくなったら、STEP12 を読み返してみてください。どのようにドットマトリクス LED の明暗を変化させたか復習しましょう。

8 ビットの A/D 変換値は 0 ～ 255 の値をとるので、これを待ち時間に利用しましょう。
ms 単位の待ち時間関数では長過ぎて点滅が見えてしまいます。ループ回数の 1560 を減らすか、前 STEP のスピーカ再生で使った「us 単位」の待ち時間関数を使うなどしてください。

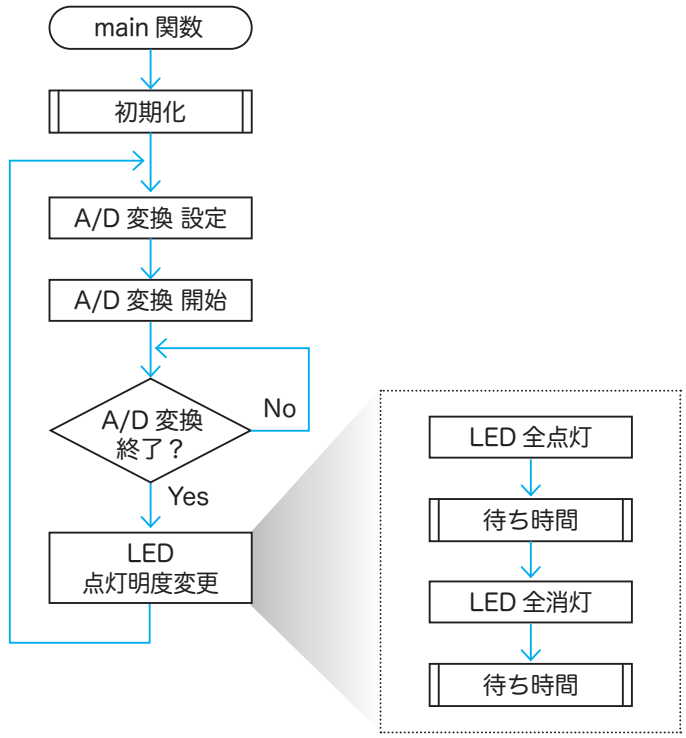


図 18-7　フローチャート

プログラム例　18-2

```
/************************************************************
    製作者     アドウィン
    解説       可変抵抗による端子電圧の変化
               →  A/D 変換し LED ドットマトリクスの点滅パルスに
************************************************************/
#include <3052f.h> // 3052F 固有の定数

/*
 *  初期化関数
 */
void initIO(void)
{
    P4.DDR = 0xFF; // 出力 LED 横行
    PB.DDR = 0xFF; // 出力 LED 縦行

    P4.DR.BYTE = 0x00; // L アノード
    PB.DR.BYTE = 0x00; // L カソード
}

/*
 *  待ち時間関数 [us]
 */
void waitUs(int us)
{
    int i, k;
    for (i = 0; i < us; i++)
        k++;
}

/*
 *  main 関数
 */
int main(void)
{
    initIO();           // 初期化関数の呼び出し
    AD.CSR.BYTE = 0x02; // A/D 変換設定  AN2 使用

    while (1)
    {
        AD.CSR.BIT.ADST = 1;        // A/D 変換開始
        while (AD.CSR.BIT.ADF == 0) // A/D 変換終了待ちループ
            ;

        P4.DR.BYTE = 0xFF; // 点灯
        waitUs(AD.DRC >> 8);
        P4.DR.BYTE = 0x00; // 消灯
        waitUs(255 - (AD.DRC >> 8));
    }

    return 0;
}
```

```
waitUs(255 - (AD.DRC >> 8));
```

この演算はどういう意味でしょうか？「AD.DRC >> 8」はビットシフトしているので２進数のようですが・・・
H8 用コンパイラ「h8300-hms-gcc」は、0x を付けた数値を 16 進数、0x を付けなければ 10 進数として扱うだけで、どちらも２進数で処理されます。つまり 10 進数と 16 進数の混在演算が可能なのです。
例えば、(AD.DRC >> 8) が２進数で 1011 0001 とすると、10 進数で 177、16 進数で B1 なので、255 − 177　や　0xFF − 0xB1 でも変わらないということです。

残念ながら H8 用コンパイラ「h8300-hms-gcc」は、0b 接頭辞を付けた２進数表記には対応していません。