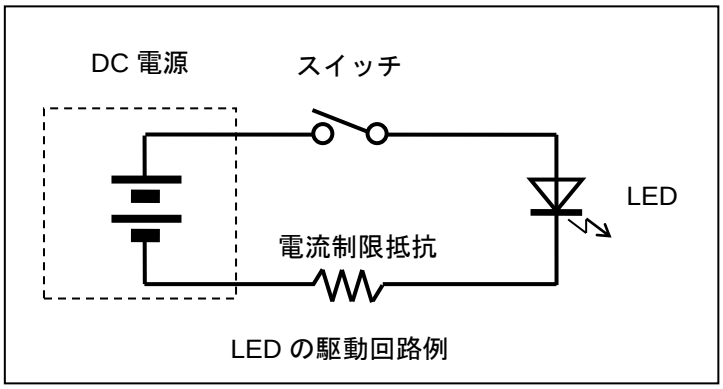
マイコンのポート機能（出力拡張）
マイコンのポートを使って、何か制御する場合には、通常は制御対象一つに対して１本のポートを使用します。
しかし、表示やキー入力のように、人間を対象とした入出力の場合には、マトリクス構成にすることで、必要なポートの数を削減することができます。
逆に言うと、少ないポートでより多くの制御が可能です。

ここでは、ピン数が一番少ない RL78/G10 の 10 ピンの製品でどこまでの LED を制御できるかにチャレンジしてみます。実際には、P40 はオンチップ・デバッグ用に使用するので、P00～P04 の 5 本のポートを使用した例となります。

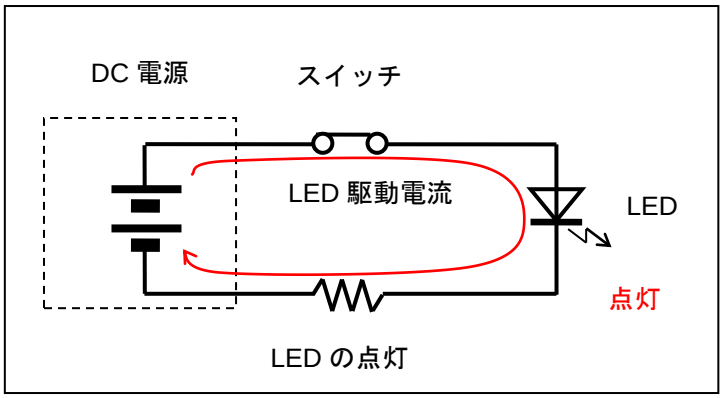
(1)　LED の駆動方法
最初に、LED を駆動する方法について、まとめてみます。

LED はアノード（プラス）側からカソード（マイナス）側に電流を流すことで点灯させることができます。LED を駆動する回路例を示します。この回路例で，スイッチが開いていると，電流が流れないので，LED は点灯しません。



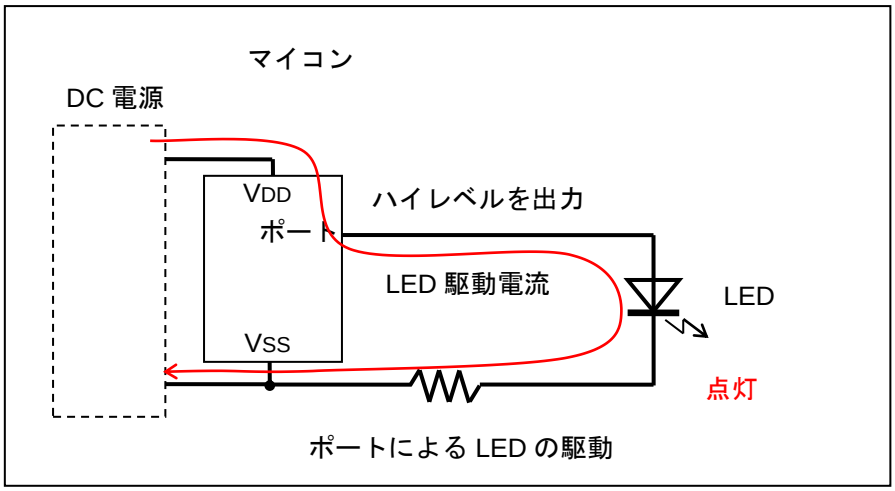
LED の駆動回路例

スイッチを閉じると電流の流れる経路が構成され，LED に電流が流れて点灯します。

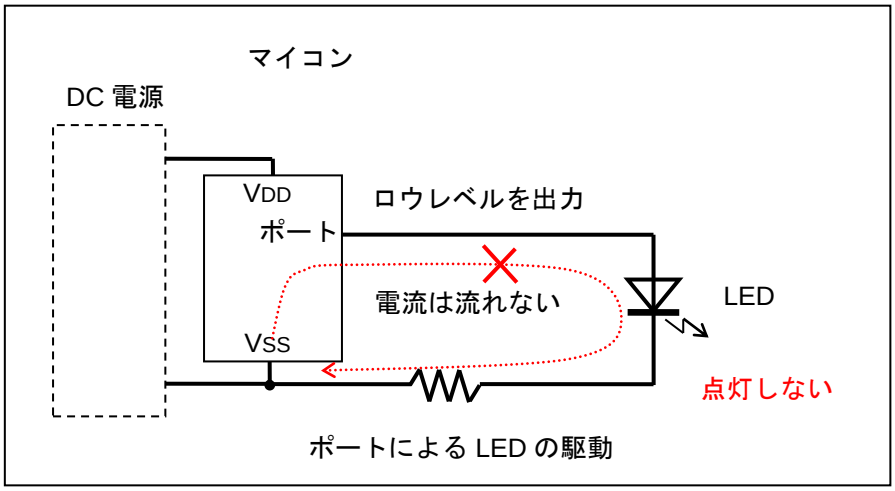


LED の点灯

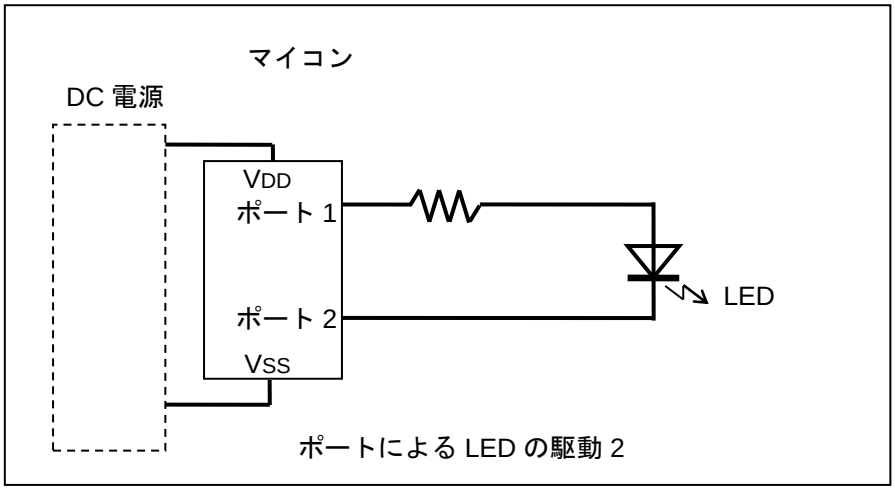
このようにスイッチを操作することで，LED に電流を流すことで，LED を点灯させることができます。
同じことは，スイッチの代わりにマイコンのポートを使って実現することもできます。
マイコンのポートをハイレベルに設定すると，LED のアノードにプラスの電圧が印加され，LED に電流が流れます。LED に電流が流れると LED は点灯します。



ポートをロウレベルに設定すると，LED のアノードには十分な電圧がかからなくなります。そのため，電流が流れなくなり，LED は点灯しません。

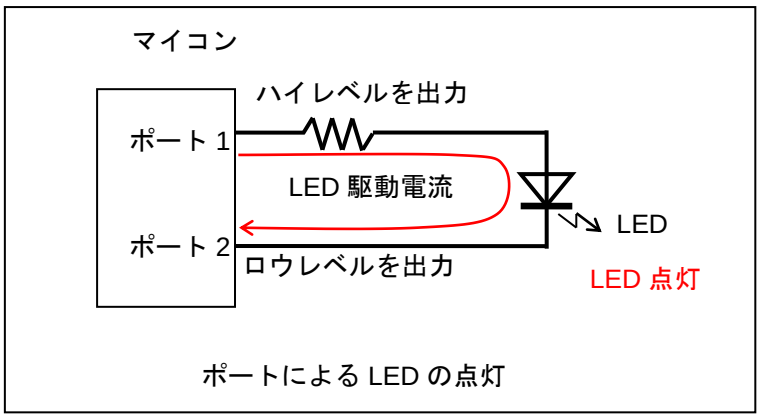


回路を少し変更して"ポートによる LED の駆動 2"のようにしてみます。これでも，LED のアノードにカソードよりも高い電圧をかけることで LED を点灯させることが可能です。



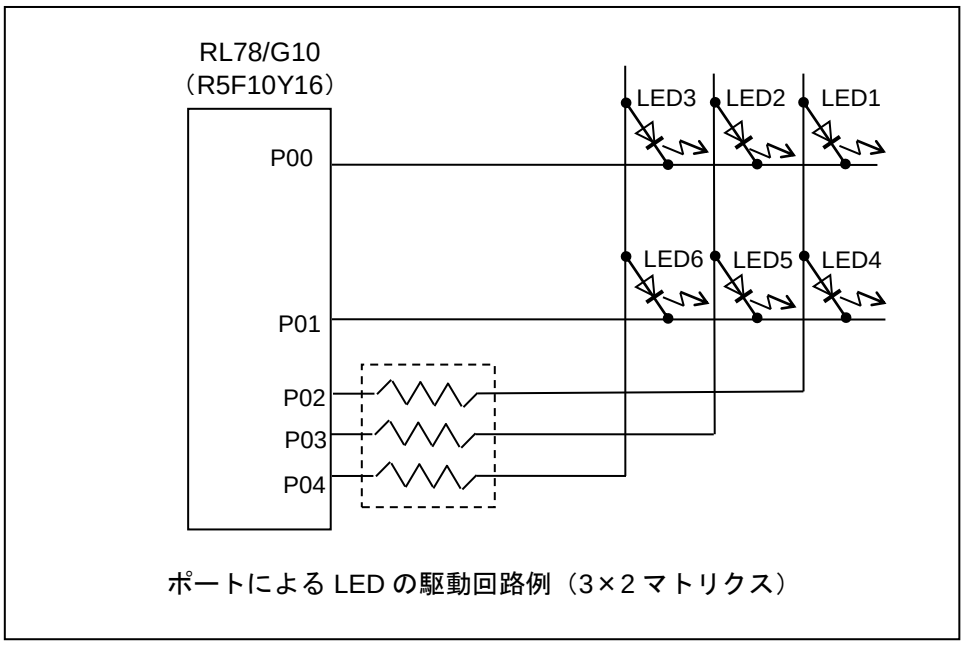
以降，面倒なので，マイコンの電源は省略します。

LED を点灯させるには，ポート 1 をハイレベル，ポート 2 をロウレベルに設定します（そうすると，LED のアノードにはカソードより高い電圧が印加され，電流が流れます）。ポート 1 及びポート 2 のそれ以外の組み合わせでは LED に電流は流れないので，LED は点灯しません。



ポートによる LED の点灯

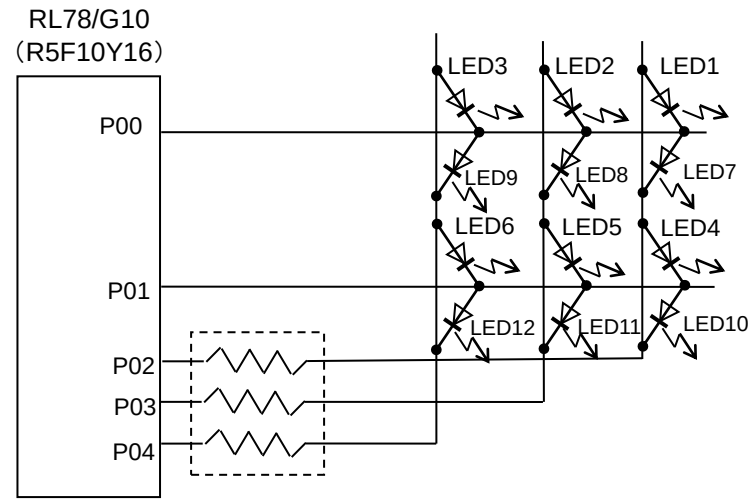
(2) LED マトリクス（ダイナミック点灯）
以上を踏まえて、LED マトリクスの構成です。
5 本のポートを使用する場合には，3×2 のマトリクスにすることで，6 個の LED が制御可能です。本当なら、LED ごとに電流制限抵抗を入れるのですが、ここでは、手を抜いて、3 本だけ入れています。



ポートによる LED の駆動回路例（3×2 マトリクス）

この回路例では、P02 にハイレベルを出力し、P00 にロウレベルを出力すると、LED1 には電流が流れるので、LED1 が点灯します。P03 にハイレベルを出力し，P00 にロウレベルを出力すると，LED2 が点灯します。
このように、P02～P04 と P00、P01 の出力を組み合わせることで、LED1～LED6 のどれかを点灯させることができます。
この駆動回路は，同時に LED1 と LED6 だけを点灯させることはできません。しかし，誤動作する欠陥回路ではありません。この回路を使うには，プログラムが大きな働きをします。
人の目には残像効果があります。これを利用して，あるタイミングでは LED1 だけを点灯させ，次に LED6 だけを点灯させ，これを目にも止まらぬ速さで繰り返します。すると，人の目には LED1 と LED6 が同時に点灯しているように見えます。
これは、ダイナミック点灯と呼ばれている方法です。

(3) LED マトリクスその２
3×2 マトリクスでは、5 本のポートを使って 6 個の LED しか制御できないので、マトリクスはあまりメリットがないようにみえます。
そこで、ポートからの出力にはハイレベル、ロウレベル以外にハイインピーダンスがあることを利用して、制御する LED の数を倍の 12 個にしてみます。



ポートによる LED の駆動回路例（3×2×2 マトリクス）

上記回路が 5 本のポートで 12 個の LED を制御する回路です。回路自体は LED を逆向きに追加しただけと単純な回路です。
この駆動回路でも時分割での点灯を行いますが，単にロウレベルとハイレベルだけの組み合わせでは実現できません。
RL78/G10 のポートにはロウレベルとハイレベル以外の状態が存在します。それは，入力ポート状態で，外部から見ると，ハイインピーダンス状態です。この状態では，入力ポートなので，外部を駆動することはできません。このことを積極的に活用しています。
具体的な制御としては，点灯させたい LED の接続されたポートだけを出力ポートに設定し，ドライブできるようにします。これだけでは，例えば，LED1 と LED7 のどちらが点灯するか特定できません。
そこで，ポートのデータを同時に制御します。P00 をロウレベル，P02 をハイレベルにします。すると，LED1 ではアノードがプラス，カソードがマイナスとなり，順方向にバイアスされるので，電流が流れて LED1 は点灯します。しかし，LED7 では，カソードがプラス，アノードがマイナスと逆方向にバイアスされるために電流は流れません。従って，LED1 は点灯するが，LED7 は点灯しない状態になります。
この回路の制御タイミグを下記の表に示します。

タイミング	P00	P01	備考
タイミング 1	ロウレベル	Hi-Z	LED1～LED3 の点灯（ハイレベルで点灯）
タイミング 2	Hi-Z	ロウレベル	LED4～LED6 の点灯（ハイレベルで点灯）
タイミング 3	ハイレベル	Hi-Z	LED7～LED9 の点灯（ロウレベルで点灯）
タイミング 4	Hi-Z	ハイレベル	LED10～LED12 の点灯（ロウレベルで点灯）

3×2 マトリクスの「×2」部分が「×2×2」となったと考えられます。

RL78/G10（R5F10Y16）では、オンチップ・デバッグ用の P40/TOOL0 端子を出力ポートとして利用することは可能です。P40 も使用すると最大で 18 個の LED を制御可能になります。18 個の LED を準備して並べるのが大変なのと、オンチップ・デバッグができなくなるのが面倒なので 12 個までの制御にとどめます。

(4) 簡単な制御
12 個の LED を完全に独立して制御するのは大変なので、最初は 1 個ずつ点灯させることにします。
それではこの 12 個の LED を点灯させる条件をまとめてみましょう。

駆動する LED	P00	P01	P02	P03	P04	備考
LED1	ロウレベル	Hi-Z	ハイレベル	Hi-Z	Hi-Z	
LED2	ロウレベル	Hi-Z	Hi-Z	ハイレベル	Hi-Z	
LED3	ロウレベル	Hi-Z	Hi-Z	Hi-Z	ハイレベル	
LED4	Hi-Z	ロウレベル	ハイレベル	Hi-Z	Hi-Z	
LED5	Hi-Z	ロウレベル	Hi-Z	ハイレベル	Hi-Z	
LED6	Hi-Z	ロウレベル	Hi-Z	Hi-Z	ハイレベル	
LED7	ハイレベル	Hi-Z	ロウレベル	Hi-Z	Hi-Z	
LED8	ハイレベル	Hi-Z	Hi-Z	ロウレベル	Hi-Z	
LED9	ハイレベル	Hi-Z	Hi-Z	Hi-Z	ロウレベル	
LED10	Hi-Z	ハイレベル	ロウレベル	Hi-Z	Hi-Z	
LED11	Hi-Z	ハイレベル	Hi-Z	ロウレベル	Hi-Z	
LED12	Hi-Z	ハイレベル	Hi-Z	Hi-Z	ロウレベル	

この条件の中で PM0 レジスタでの出力／ハイインピーダンス制御は制御を簡単にするために，LED の番号に対応した PM0 レジスタの設定値をテーブル（配列）として準備しておき，LED 番号でテーブル参照を行います。テーブルの例を示します。

```
static const uint8_t dataTBL[]=
{
    0b11111010,                 /* LED1,7 ON    */
    0b11110110,                 /* LED2,8 ON    */
    0b11101110,                 /* LED3,9 ON    */
    0b11111001,                 /* LED4,10 ON   */
    0b11110101,                 /* LED5,11 ON   */
    0b11101101,                 /* LED6,12 ON   */
    0b11110110,                 /* LED2,8 ON    */
    0b11101110,                 /* LED3,9 ON    */
    0b11111001,                 /* LED4,10 ON   */
    0b11110101,                 /* LED5,11 ON   */
    0b11101101,                 /* LED6,12 ON   */
    0b11111010                  /* LED1,7 ON
      |||||||+----- PM0.0 (LED1-3,7-9)
      ||||||+------ PM0.1 (LED4-6,10-12)
      |||||+------- PM0.2 (LED1,4,7,10)
      ||||+-------- PM0.3 (LED2,5,8,11)
      |||+--------- PM0.4 (LED3,6,9,12)
      +++---------- not used must be 1
*/
};
```

このテーブルの内容を参照して，以下のようにします。ここで，INDEX は対象とする LED の番号（1～12）です。

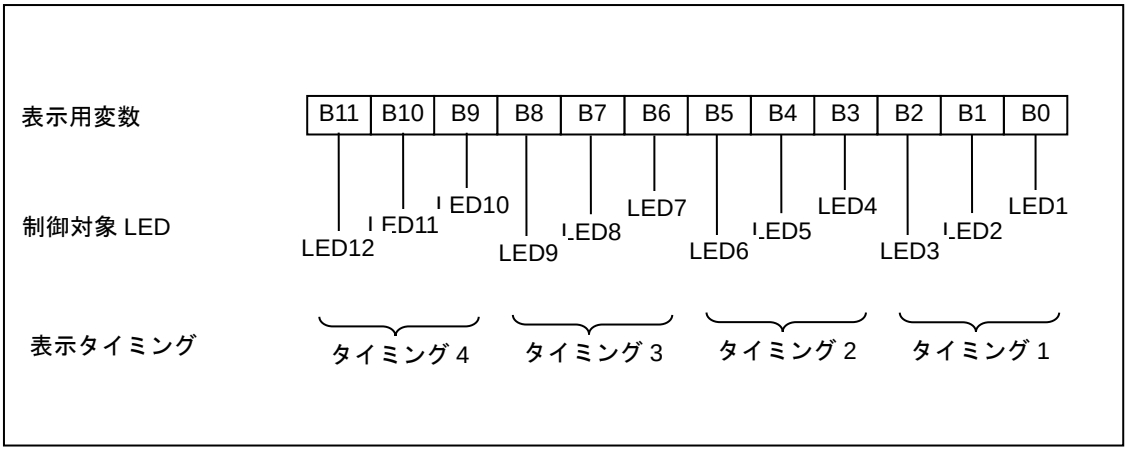
```
        PM0 = dataTBL[INDEX-1]
```

このとき，P00,P01 がロウレベルで，P02～P04 がハイレベルなら，LED1～6 のどれか 1 個が点灯し，P00,P01 がハイレベルで，P02～P04 がロウレベルなら，LED7～12 のどれか 1 個が点灯します。
これだけでも、結構使えると考えられます。

(5) 通常のダイナミック点灯制御

単に LED を 1 個ずつ点灯させるのであれば、12 個ではなく、20 個でも点灯させることは可能（P00 と P01 の間に 2 個、P02～P04 の各間に 2 個ずつの合計 8 個は追加可能）です。しかし、20 時分割表示では表示が暗くなってしまうことが考えられるので、ここではやりません。

ある程度、明るさを確保することを考えると、LED1～LED3 は同時に点灯できるようにし、LED4～LED6、LED7～LED9、LED10～LED12 も同様に動じ点灯できるようにして 4 時分割での点灯を制御します。こうして、点灯可能時間を十分確保することで、ソフトウェア PWM による明るさ制御の可能性を残しておきます。



ここでは LED 表示データ用に 16 ビットの変数 G_LED_DATA を定義し、各ビットを上記のように定義しておきます。

表示タイミングを変数 G_TIMING でカウントし、その値に応じて switch 分で、処理するビット位置と P0 に設定する値を変化させることでダイナミック表示を制御します。switch 文の中を以下に示します。

```
    switch ( G_TIMING )             /* check timing             */
    {
        case    0:                  /* LED1-3 lighting timing   */
            P0_image = 0b00011100;
            PM0_image = (uint8_t)((G_LED_DATA << 2)& 0b00011100);
            PM0_image ^= 0b00011100;
            PM0_image |= 0b00000010; /* P1 output disable       */
            break;

        case    1:                  /* LED4-6 lighting timing   */
            P0_image = 0b00011100;
            PM0_image = (uint8_t)((G_LED_DATA >> 1)& 0b00011100);
            PM0_image ^= 0b00011100;
            PM0_image |= 0b00000001; /* P0 output disable       */
            break;

        case    2:                  /* LED7-9 lighting timing   */
            P0_image = 0b00000011;
            PM0_image = (uint8_t)((G_LED_DATA >> 4)& 0b00011100);
            PM0_image ^= 0b00011100;
            PM0_image |= 0b00000010; /* P1 output disable       */
            break;

        default:                    /* LED10-12 lighting timing */
            P0_image = 0b00000011;
            PM0_image = (uint8_t)((G_LED_DATA >> 7)& 0b00011100);
            PM0_image ^= 0b00011100;
            PM0_image |= 0b00000001; /* P0 output disable       */
            break;
    }
```

このように、各タイミングに応じて、G_LED_DATA（表示データ）の対応したビットを抽出し、その値で変数 PM0_image の表示させたい LED に対応したビットは 0、それ以外は 1 にし、P0 に設定する値を P0_image に作成します。後は、このデータを P0 と PM0 に設定すれば、4 時分割による制御が完了します。
以上の処理はタイマ割り込みの中で行われるため、メイン関数では、どのようなデータを表示させるかを変数 G_LED_DATA に設定するだけで済みます。
なお、G_LED_DATA は、時分割で表示を制御する関数側でグローバル変数として定義されており、メイン関数では外部変数として宣言してあります。
たとえば、下記の例は LED1 から LED12 を順に 1 個ずつ点灯させる例です。

```
        for (G_LED_DATA = 0b000000000001 ; G_LED_DATA < 0b1000000000000 ; )
        {

            Wait_1sec();
            G_LED_DATA = G_LED_DATA << 1;

        }
```

下記の例は LDE1 から順に点灯し、12 個の LED すべてを点灯させるものです。

```
        for (G_LED_DATA = 0b000000000001 ; G_LED_DATA < 0b1000000000000 ; )
        {

            Wait_1sec();
            G_LED_DATA = G_LED_DATA << 1;
            G_LED_DATA++;

        }
```

下記の例は奇数番号の LED と偶数番号の LED を交互に点灯させるものです。

```
        for ( w_count = 10 ; w_count > 0 ; w_count-- )
        {
            G_LED_DATA = 0b010101010101;
            Wait_1sec();
            G_LED_DATA = 0b101010101010;
            Wait_1sec();
        }
```

このように、タイマ関数 Wait_1sec での時間待ちと組み合わせることでいろんな表示パターンを比較的簡単に実現できます。

どのようなパターンがいいかは、LED1～LED12 をどのように配置するかで変化します。上記のパターンは LED1～LED12 をサークル状に配置した場合のものです。

＜ポートの使い方（出力拡張）＞終了