11 STEP

チャタリング

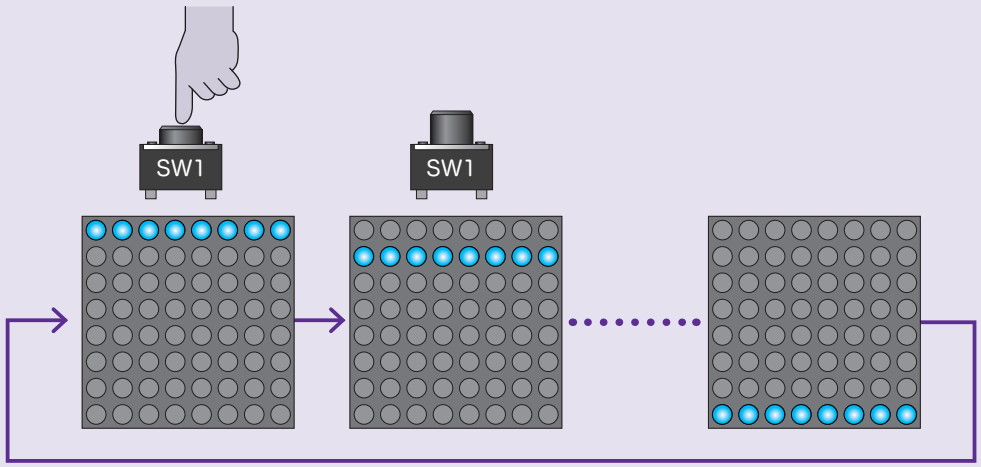
SW を使うと、チャタリングという現象が起きています。
チャタリング対策を考慮しなければ、SW をうまく扱えません。
この STEP では、チャタリングという現象とその対策を学びましょう。

11.1　チャタリングを実感しよう

チャタリングとは、SW の ON/OFF を切り替える際、金属の接点部分がバウンドする現象で、ハード的な問題と言えます。チャタリングは、メカニカルスイッチでは必ず起きる現象です。
チャタリングを防止するには、ハードウェア、ソフトウェアで回避する方法がありますが、本書ではソフトウェアで回避する方法を学習しましょう。
まずは、チャタリングがどんな現象か実感してみましょう。

課題　11-1

SW1 を押して、離すたびに LED の点灯行を下げる。8 行目が点灯した後は、1 行目に戻す。



STEP 11 チャタリング

SW を放したときに動作させるには、以下のようなフローチャートで実現できそうです。
では、図 11-1 のフローチャートを基に、プログラム 11.1 を組んでみましょう。
②の条件分岐は、SW1 が ON の間はループしているように while 文を使うといいでしょう。

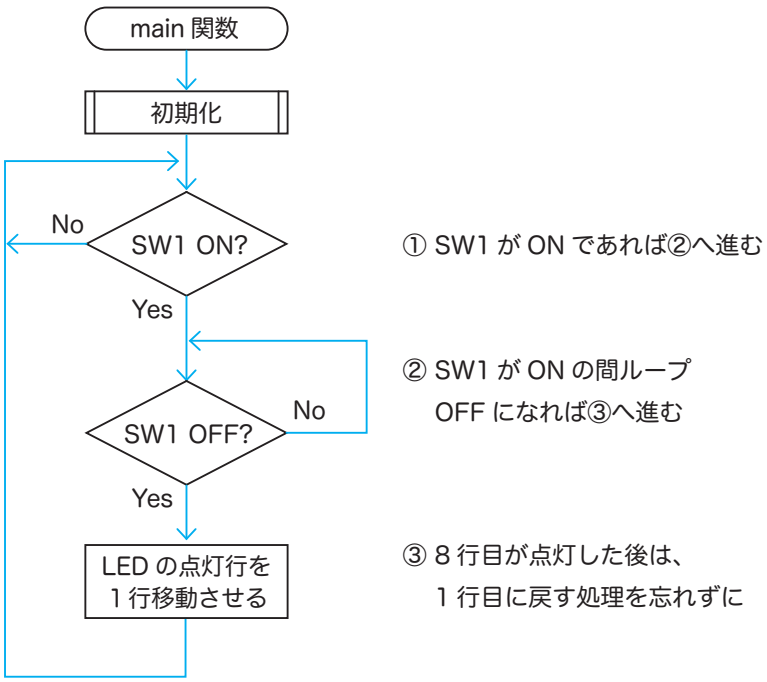


図 11-1　フローチャート

プログラム例　11-1

```
/*
 *  main 関数
 */
int main(void)
{
    int index = 1; // 配列に使う添え字の変数

    initIO(); // 初期化関数の呼び出し

    // 1 行目を点灯
    PB.DR.BYTE = 0x00;
    P4.DR.BYTE = 0x80;

    while (1)
    {
        // SW1 を押して離すと点灯行が 1 行下がる
        if (SW1_ON) // SW1 が ON
        {
            while (SW1_ON) // SW1 が ON の間ループ
                ;

            P4.DR.BYTE = a_p4[index]; // 点灯

            index++; // 点灯行の変更

            // 点灯が一番下まできたら戻す
            if (index > 7)
                index = 0;
        }
    }

    return 0;
}
```

プログラムを実行した直後は 1 行目が点灯し、SW を操作すると 2 行目が点灯するようにしたいので、配列「a_p4」の添え字である index の初期値は 2 行目を 1 にしています。
index の初期値は 0 のままで、a_p4 配列を 0x40 から始まるように並びを変更する方法もあります。

STEP 11 チャタリング

プログラム 11-1 を実行してみてください。
何度も SW の ON/OFF を繰り返していると、2、3 行飛んで点灯行が下がったりしませんか？
それが、チャタリングのために起きる現象です。

では、チャタリングについて詳しく説明していきます。

11.2　チャタリングとは

チャタリングは、SW の ON/OFF が切り替わる際、金属の接点部分がバウンドしてしまう現象です。私たちが SW を 1 回 ON したつもりでも、実は何度も ON/OFF を繰り返してるのです。図は SW がマイコン端子にアクティブ L で接続されているときの信号電圧レベルを表した波形です。

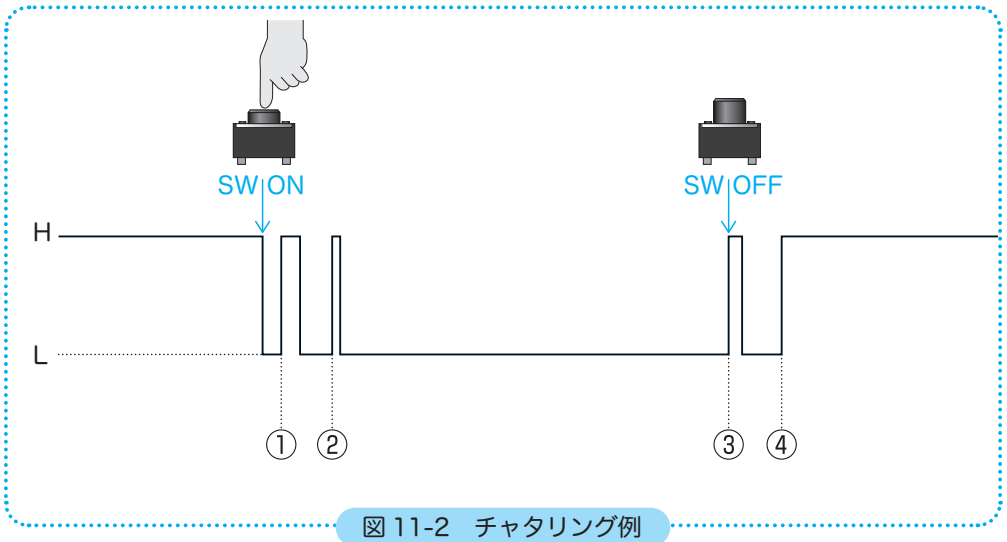


図 11-2　チャタリング例

チャタリングは、数 n 秒〜数 m 秒という非常に短い時間発生します。
図 11-2 のようなチャタリングが起きた場合、①②③④のタイミングで点灯行が下がります。
点灯行を 1 行下げたいだけなのに、上図の場合では 4 行も下がってしまうことになります。

11.3　待ち時間を入れたチャタリングの対策

チャタリング対策として、具体的に何をすればいいのでしょうか？
結論を言ってしまうと、チャタリングが起きている間、何もしなければいいのです。チャタリングが終わるまで待てば、後の処理には影響が出ないはずですね。チャタリングが起きている間、「何もしない」ようにするには、「待ち時間関数」を利用します。

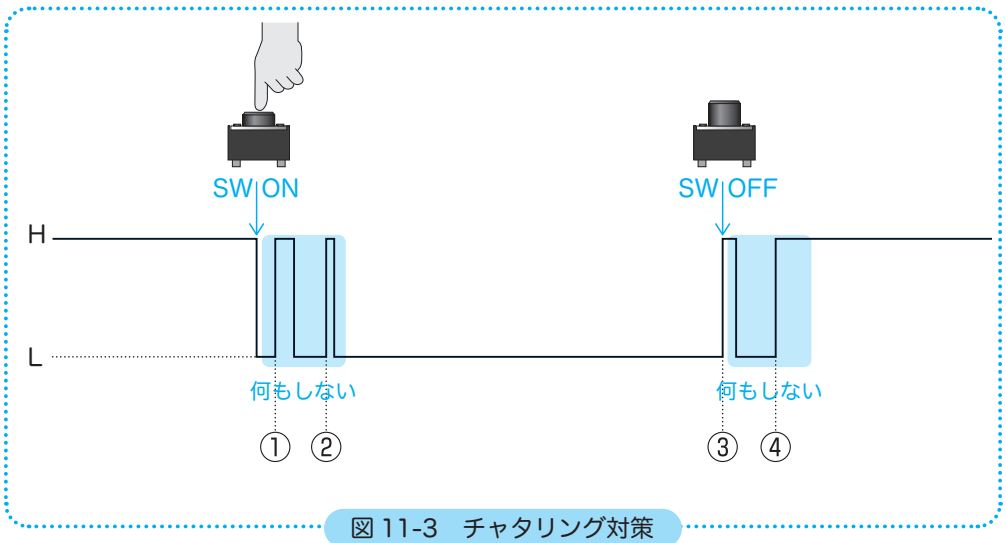


図 11-3　チャタリング対策

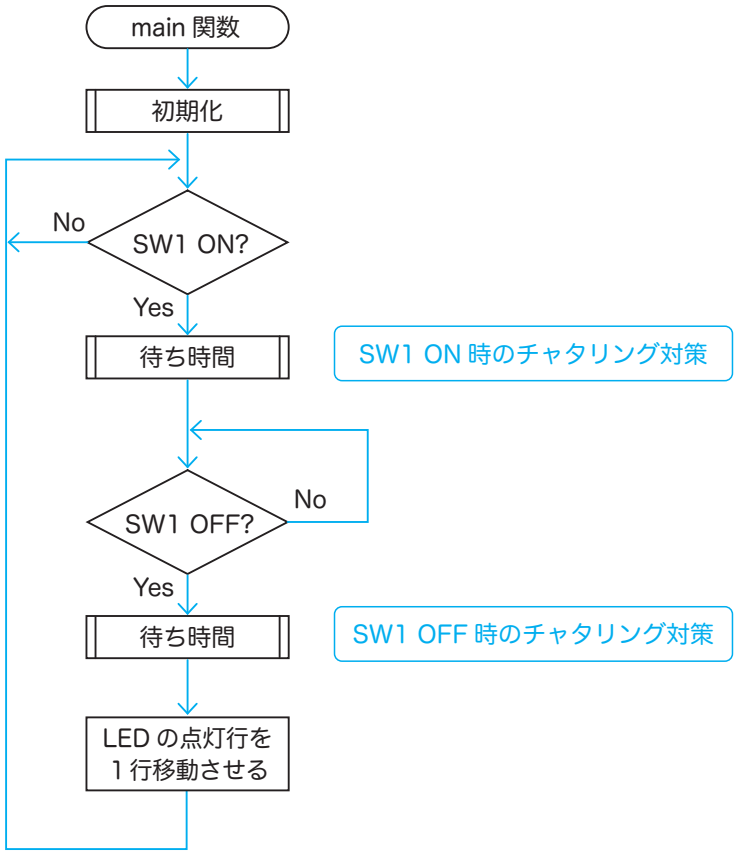


図 11-4　チャタリング対策したフローチャート

待ち時間処理で気を付けなければならないことは、待ち時間処理を行う分、処理がわずかながら遅くなることです。また、チャタリングの発生時間は、SW の仕様によって変わったり、SW が劣化してくると長くなる傾向があるので調整が必要です。
今回は、試しに 10m 秒の待ち時間を入れたプログラム 11-2 を組んでみましょう。

プログラム例　11-2

```
/*
 *  main 関数
 */
int main(void)
{
    int index = 1; // 配列に使う添え字の変数

    initIO(); // 初期化関数の呼び出し

    // 1 行目を点灯
    P4.DR.BYTE = 0x80; // 1000 0000 アノード
    PB.DR.BYTE = 0x00; // L カソード

    while (1)
    {
        // SW1 を押して離すと点灯行が 1 行下がる
        if (SW1_ON) // SW1 が ON
        {
            waitMs(10);   // チャタリング対策　SW ON 時
            while (SW1_ON) // SW1 が ON の間ループ
                ;
            waitMs(10);   // チャタリング対策　SW OFF 時

            P4.DR.BYTE = a_p4[index]; // 点灯

            index++; // 点灯行の変更

            // 点灯が一番下まできたら戻す
            if (index > 7)
                index = 0;
        }
    }

    return 0;
}
```

待ち時間 10ms でチャタリングがおさまらないようなら、待ち時間を長くして調節してみてください。

STEP 11 チャタリング

ソフトウェアによるチャタリング対策には色々な方法が考えられます。

例えば、ポートの状態を繰り返し読み込んで、ある回数（1000 回くらい）連続した状態が続いた場合に限り、スイッチが押された（あるいは放された）と見なす方法があります。図 11-5 にフローチャート例を示しますので、参考にしてください。

他にも簡単で確実な方法がないか考えてみるのもいいでしょう。

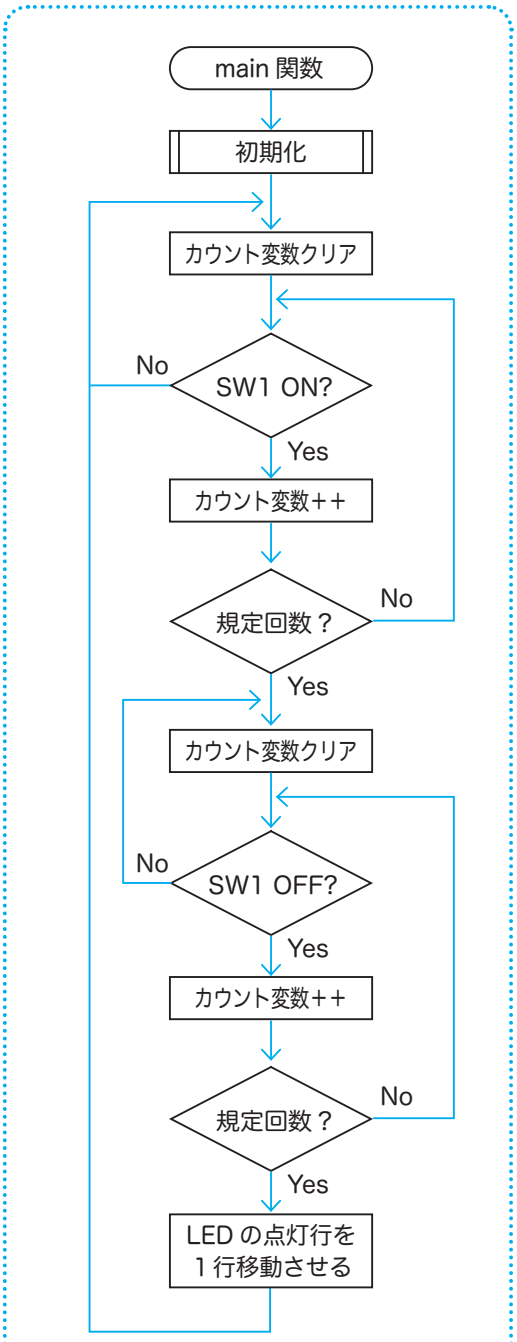


図 11-5　チャタリング対策 別案

図 11-5 フローチャートの「規定回数？」の条件分岐は、do ～ while 文を使うといいでしょう。

【 do ～ While 文 】

```
do
{
    処理 ;
}
while ( 継続条件式 );
```

while 文と異なり、まず処理を実行してから、継続条件の判定を行います。継続条件式が真で処理を繰り返し実行します。
while 文は条件によって処理が一度も実行されないことがありますが、do ～ while 文では 1 回は処理が実行されます。
継続条件式の後のセミコロン（;）を忘れないよう注意してください。

例)

```
count = 0;
do
{
    while (P8.DR.BIT.B0 == 0)
        count = 0;
    count++;
} while (count < 1000);
```