マイコンのポート機能（入力）:
（G10_PORT0_2 と G10_PORT0_3 の二つのプロジェクトが関連）

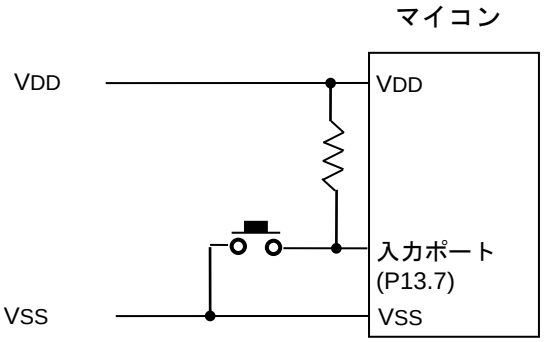
ここまではポートからの出力について説明しましたが，今回は入力について説明します。
マイコンに搭載されているポートには，データを出力する機能と，データを入力する機能，入出力を切り替えることができるものがあります。

ポートからの出力では出力したデータを保持する機能（ラッチ機能）がありましたが，ポートからの入力にはそのような機能はありません。

これは，ポートからの入力では，入力の変化のスピードに比べて，CPU の動作速度が速いので，必要なタイミングでポートを読むことで，入力信号の状態の変化を知ることができるからです。

一番簡単な入力装置（？）はスイッチです。
スイッチにもいろんなタイプがありますが，ここでは，押したときに導通（オン）し，放すと導通しなくなる（オフする）スイッチを考えます。

マイコンにスイッチを接続するときに最も一般的な方法に入力ポートを抵抗でプルアップしておき，スイッチをポートとグランドの間に接続する方法です。

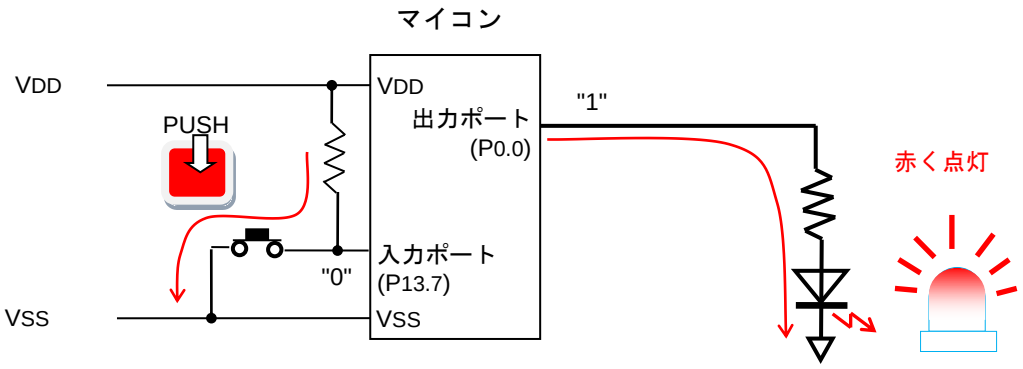


RL78 では，P13.7 は共通的に外部割り込み入力兼用の入力ポートとなっています。
この端子を使ったポート入力の方法を「評価ボード 2」を用いて動作確認します。

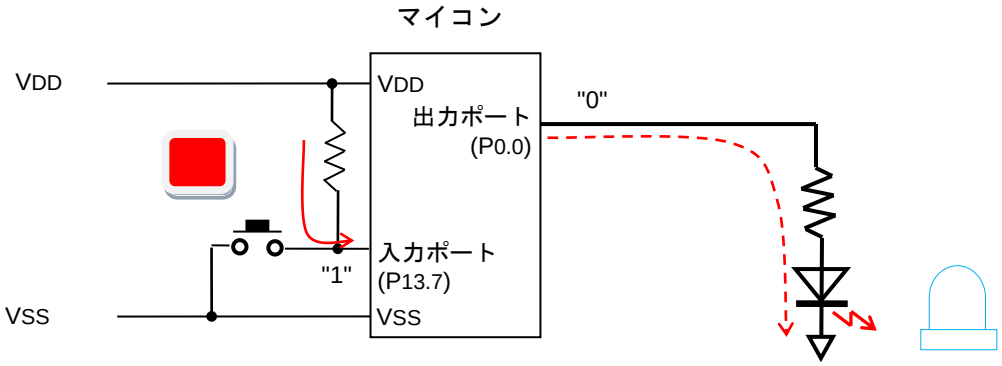
プログラムの例としては，下記のようになります。

```
P0.0 = ~P13.7;                    /* switch red LED   */
```

この処理は，入力ポート"P13.7"の状態を読み込み，論理を反転して出力ポート P0.0 に出力するものです。
具体的なイメージは P13.7 に接続されたスイッチが押されると，値として，"0"が読み込まれます。その論理を反転（"1"に）して P0.0 に"1"を出力すると，その先に接続された LED（赤）が点灯します。



スイッチが放されると，P13.7 で読み込む
値は"1"になります。その値の論理を反転して，P0.0 に"0"を出力すると，LED は
消灯します。



ポートからの入力は，このような簡単なデータの代入処理で使用するだけでなく，
プログラムの判断分岐でも使用できます。上記の代入処理と同じことを判断分岐
を使って記述すると以下のようになります。

```
if(P13.7 == 0)
{
        P0.0 = 1;               /* turn on red LED      */
}
else
{
        P0.0 = 0;               /* turn off red LED     */
}
```

また，スイッチが押されている間は何かの処理をする場合には以下のように記述
することが可能です。

```
while(P13.7 == 0)
{
        何かの処理
}
```

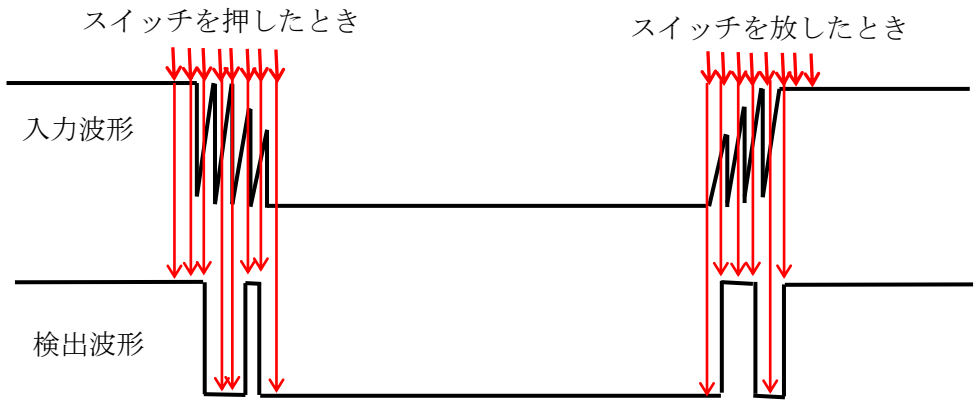
スイッチが押されるのを待つ場合には，以下のようなプログラムとなります。
最初に押されていない状態になるのを待ち，その後，押された状態となるのを
待ちます。

```
while(P13.7 == 0)               /* wait for high level  */
{
        NOP();
}
while(P13.7 == 1)               /* wait for low level   */
{
        NOP();
}
```

このように，代入式の右辺や判断分岐の条件として，入力ポートの状態を簡単に
使用することができます。
なお，押されるのを待つ（検出）や，逆に放されるのを待つは，ハードウェアで
信号の変化（エッジ）を検出することも可能です。検出したことは，割り込みや
フラグで確認することができます。

マイコンのポート機能（入力の 2）:

スイッチのような機械的部品では，状態が変わるときに機械的な振動が発生します。この振動で，スイッチを押したときや，放したときにスイッチの接点が接触したり離れたりして，入力ポートで読み出す値が変化します。この状態は，数十 ms 程度続くことがあります。これは，人からみると短い時間ですが，マイコンからみると，非常に長い時間です。1 回押したつもりが，数回押されたことを検出することがあります。また，押したことを検出したつもりが，放されたときにも検出してしまうこともあります（下図の赤い矢印がマイコンでチェックしたタイミングの例）。このようなスイッチの振動現象は"チャタリング"と呼ばれます。



実際にスイッチに入力をプログラムに取り込んでみます。出力ポートでの LED 制御で使用したプログラムを一部変更してみます。

```
P0.0 = 1;                       /* turn on red LED          */
HALT();
NOP();
```

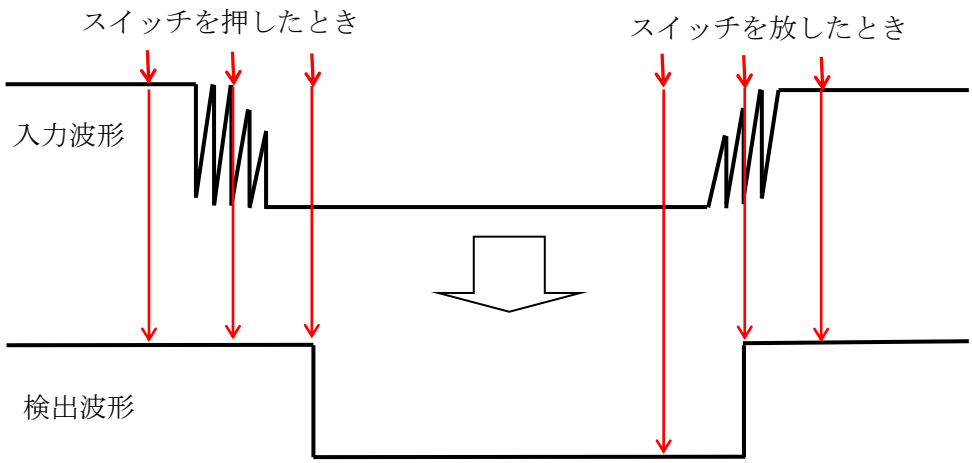
これを以下のように，"HALT();"と"NOP();"の 2 つを" wait_SW();"に変更します。

```
P0.0 = 1;                   /* turn on red LED          */
wait_SW();                  /* wait for SW is pushed    */
```

ここで，"wait_SW()"関数は前回示した SW が押されたことを検出する処理プログラムです。
プログラム全体としては，①の状態からスイッチを押す度に次に移っていくはずです。
①左の LED を赤で点灯
②左の LED を消灯
③左の LED を青で点灯
④左の LED を消灯
⑤右の LED を赤で点灯
⑥右の LED を消灯
⑦右の LED を青で点灯
⑧右の LED を消灯
⑨左の LED を赤と青で点灯（紫になる）
⑩右の LED を赤で点灯
⑪右の LED で青も点灯（紫点灯），左で赤を消灯（青点灯）
⑫左の LED の青を消灯（左は消灯）
⑬すべての LED が消灯
⑭①に戻る。

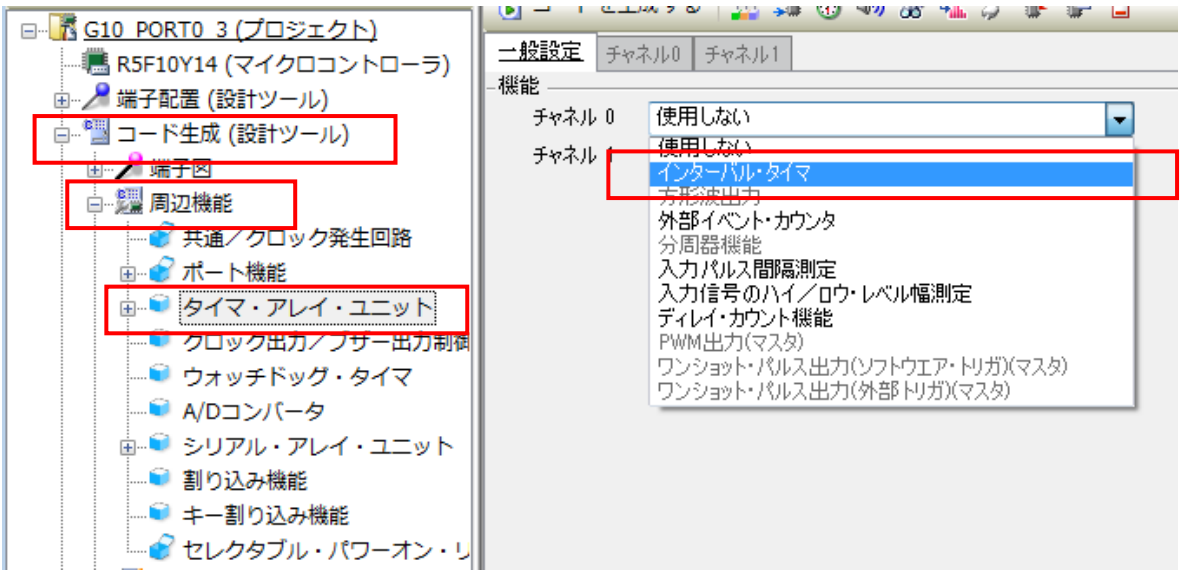
実際に動作させてみると，たまにどこかが抜けてしまうのが分かります。これがチャタリングの影響です。

チャタリングの影響をなくす最も簡単な方法は，チャタリングが発生する期間よりも長い周期でスイッチの状態をチェックすることです。
下図に示すようにチャタリングが発生している期間に 1 回以上チェックしないようにすることです。

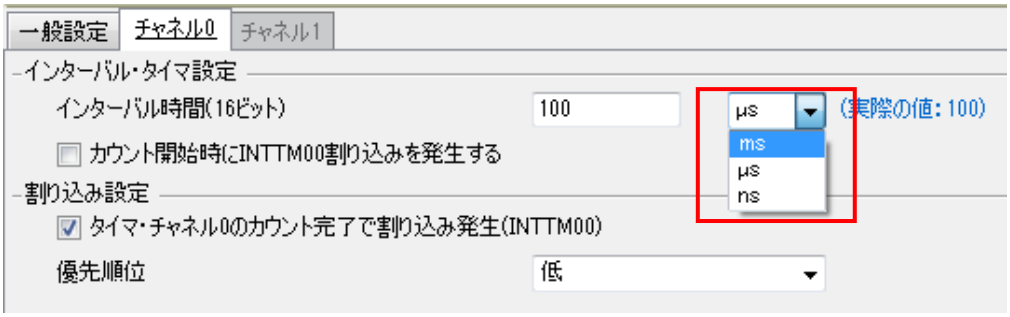


このように，チャタリングの発生している期間よりも長い間隔で入力ポートをチェックすると，それだけでチャタリングの影響をなくすことが可能です。ただし，その分だけスイッチが押されたことを検出するタイミングが遅くなる可能性があります。

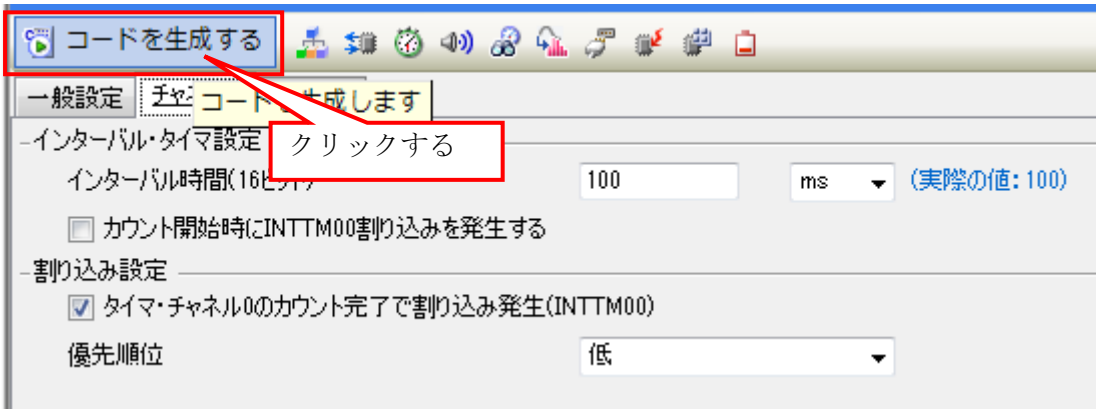
この機能を実現するためには，タイマ機能を使った割り込みで処理するのが簡単です。
使用するタイマ機能は TAU を用いて，インターバル・タイマとして使用します。
TAU のインターバル・タイマを使用するには，「コード生成（設計ツール）」を使い，「周辺機能」の中から「タイマ・アレイ・ユニット」を選択します。右側の画面で，「一般設定」の使用するチャネルのプルダウン・メニュから「インターバル・タイマ」を選択します。（下図参照）



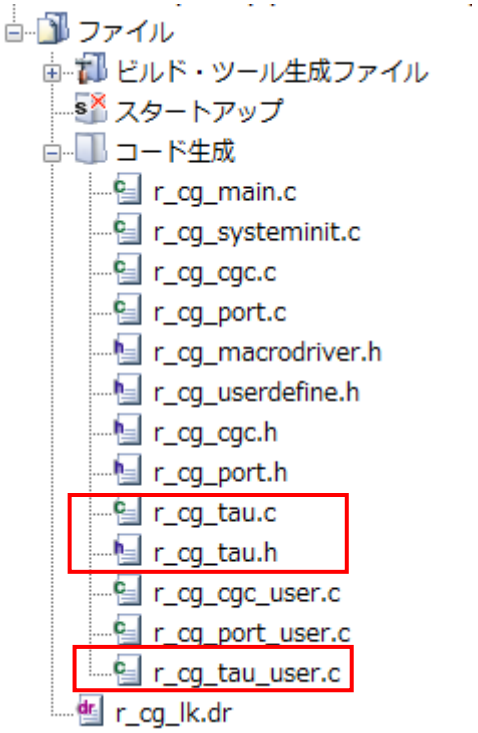
使用するチャネルをインターバル・タイマに設定したら，対象のチャネルのタグを選択して，詳細な設定を行います。
実際にやることは，インターバル時間の設定だけです。ここでは，インターバル時間として 100ms を選択してみます。初期状で，100μs になっているので，単位をプルダウン・メニュから「ms」を選択するだけです。



設定が完了したら，「コード生成する」をクリックすれば，タイマの初期設定の指定は完了です。



これで，「ファイル」の「コード生成」の下に「r_cg_tau.c」「r_cg_tau.h」及び，「r_cg_tau_user.c」の 3 つのファイルが生成されます。



「r_cg_tau.c」には初期設定（これは自動的に処理されます）とタイマの動作開始関数（R_TAU0_Channel0_Start）が生成されています。「r_cg_tau_user.c」は割り込みでの処理内容を記述するための割り込み関数が生成されています。

生成された割り込み処理関数を示します。ここに必要な処理を記述することになります。
生成された割り込み関数は単に入れ物だけがつくられるので，必要な処理を記述します。
ここでは，グローバル変数として，ビットの P137image を定義し，タイマ割り込み処理
ではそこに P13.7 の値を取り込んでいる（サンプリングしている）だけです。

```
/***********************************************************************
Global variables and functions
***********************************************************************/
/* Start user code for global. Do not edit comment generated here */
bit P137image;                          /* image data of P13.7  */
/* End user code. Do not edit comment generated here */

/***********************************************************************
* Function Name: r_tau0_channel0_interrupt
* Description  : This function INTTM00 interrupt service routine.
* Arguments    : None
* Return Value : None
***********************************************************************/
__interrupt static void r_tau0_channel0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    P137image = P13.7;
    /* End user code. Do not edit comment generated here */
}
```

その上で，wait_SW 関数では，P13.7 の代わりにこのビット変数 P137image を参照する
ようにします。

```
    while( P137image == 0 )
    {
        NOP();
    }
    while( P137image == 1 )
    {
        NOP();
    }
```

このためには，r_cg_main.c のグローバル変数の宣言部分に以下のように外部で宣言して
いる変数であることの宣言を追加しておきます。

```
/***********************************************************************
Global variables and functions
***********************************************************************/
/* Start user code for global. Do not edit comment generated here */
void wait_SW(void);
extern bit P137image;
/* End user code. Do not edit comment generated here */
```

これで，チャタリング対策は完了です。

＜ポートからの入力＞終了