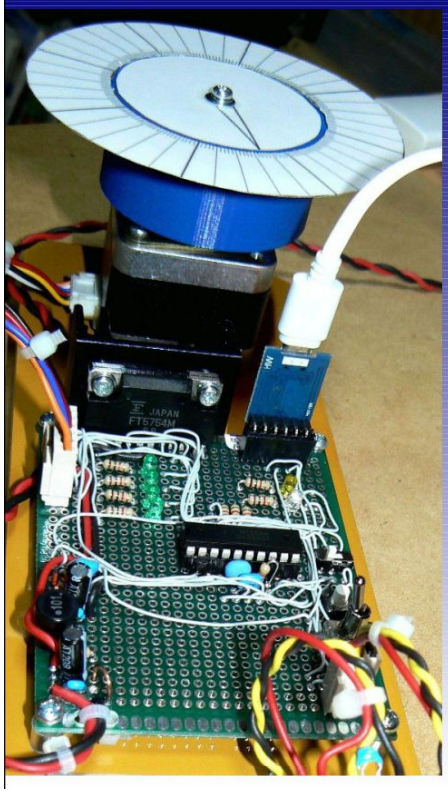
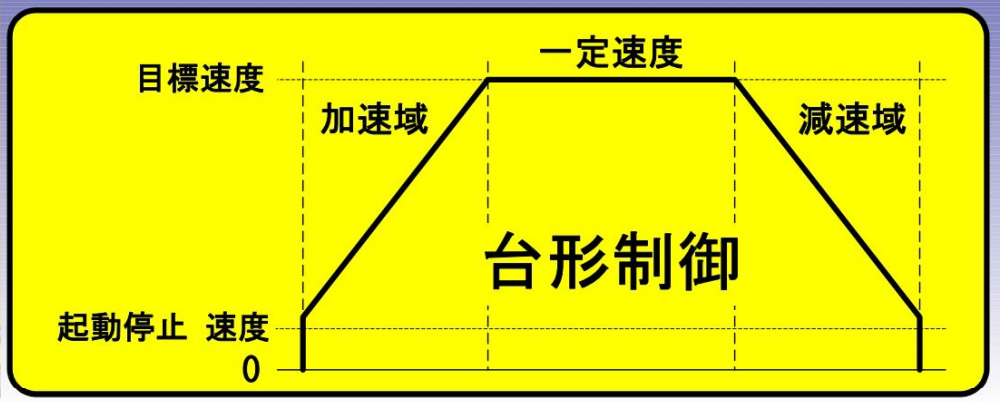
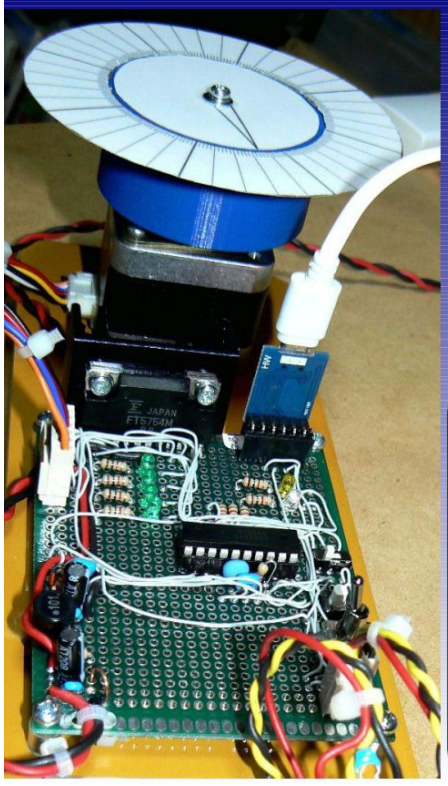
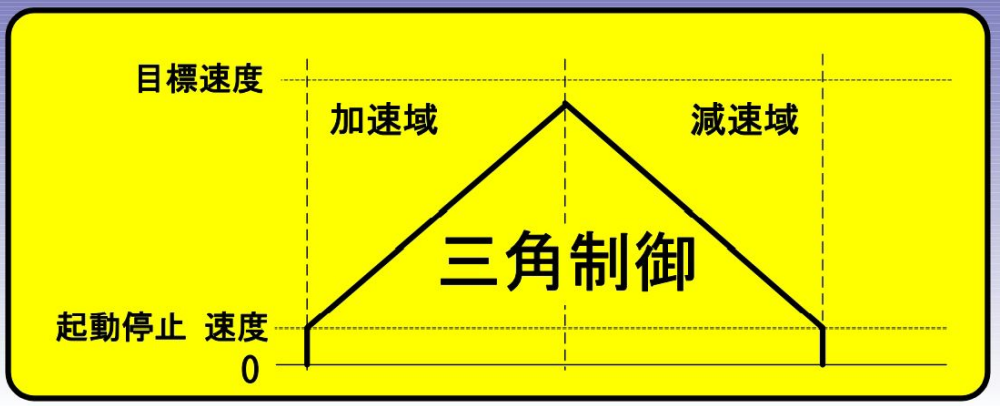



目標速度
一定速度
加速域
減速域
台形制御
起動停止 速度
0




目標速度
加速域
減速域
三角制御
起動停止 速度
0

発生した3つの障害

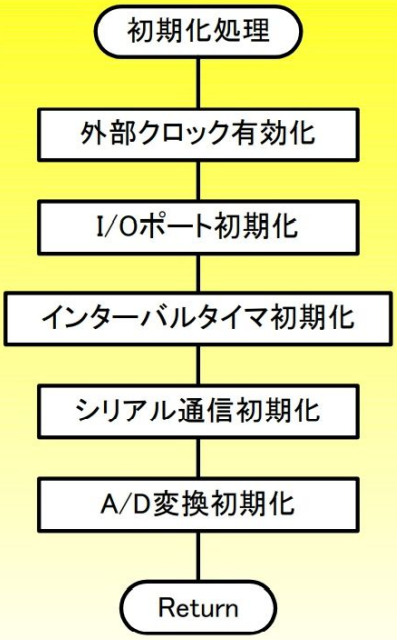
(1) **R8C/M120A マイコンにて、外部発振子を使おうとすると、動作しない。**
(外部発振子が、発振してない。)
前回、発生した障害で、応急的な回避策として、外部発振子を使わずに、
CPU内部 RC発振器を使用する事で、前回は動かしました。

(2) **シリアル通信のトラブル1:** ターミナルソフトで 通信できるソフトと
通信出来ないソフトがある。
これは、今回パソコンから、マイコンに各種コマンドを 送る事により
マイコンを動かそうと考えたから、気付いた障害で、(3)の障害もそうです。

(3) **シリアル通信のトラブル2:** マイコンの電源ON直後、通信が出来ない。
電源ON後、数分経過すると通信出来るようになる。

(1) R8C/M120A マイコンにて、外部発振子を使おうとすると、動作しない。

この障害の原因は、A/Dコンバータの初期化ルーチンの影響でした。
R8C/M120Aの A/D入力に設定できる端子は、6本ありますが、
その6本全てを、一括 A/D端子にすると、一部シリアル通信と
重なる端子が発生し、そのシリアル端子を、別の端子に
変更する場合に、発振子を接続するXIN端子と重なっていた。
という事でした。
初期化処理の順番として、最初に クロックの設定を行います。
次に、I/Oポート初期化、インターバルタイマー、シリアル通信、
最後に A/D変換の 初期化処理を行っていました。
初期化で、**機能が重なっているピンが、あると 後に実行した**
初期化処理ルーチンで、機能が上書きされます。
調べる方法は、インターバルタイマー、シリアル通信、A/D変換の
初期化処理を、一つづつ止めて動作を確認しました。
対応策として、A/D変換 初期化処理は、6本あるA/Dピンの
全てを独立してばらばらに初期化出来るようにしました。
そして必要な、1本だけ A/D端子として初期化しました。
これにより、XIN端子とぶつかる端子の発生を回避しました。



(2) シリアル通信のトラブル1:　シリアル通信ターミナルソフトで通信できるソフトと、通信出来ないソフトがある

これは、多分 Windows側で発生している障害と、判断しました。

最初、USBシリアル通信ICモジュールとして、CP2102を使用してましたが、
その、CP210xの Windows用ドライバーモジュールに、何か問題があるようで、
CP2102を、 以前実績のあるFTDI社の FT1234X に変えたら
どの、ターミナルソフトでも使えるように なりました。

FTDI社の USBシリアル通信IC用のWindows用ドライバは、Windows に
標準で入ってるようで、Microsoft 社の お墨付きという事でしょうか。？

CP2102も、用途によっては使えると思います。
FTDI社以外の USBシリアル通信ICモジュールは、Windows側のドライバーソフトの
問題で、相性の悪い通信ソフトがある。という事です。
(私が思うに、PC側で、CS信号(送信許可)の 処理がまずいのではと 思います。)

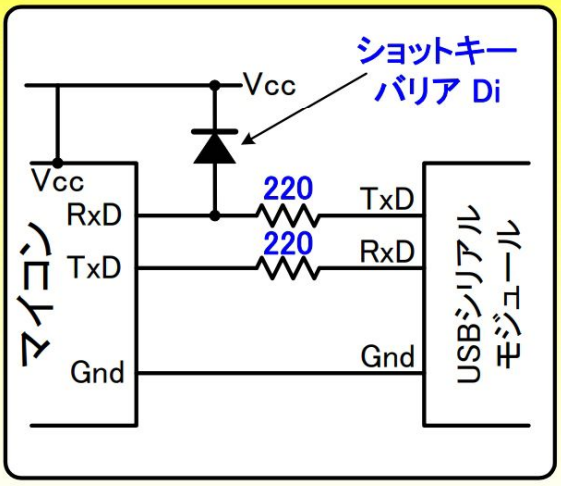
(3) シリアル通信のトラブル2：マイコンの電源ON直後、通信が出来ない。電源ON後、数分経過すると通信出来るようになる。

USBシリアル通信ICモジュールの　基本的な使い方として、USBケーブルからの
5Vまたは、USBシリアル通信ICモジュール内で分圧した、3.3Vの電源を使って
マイコンを動かす事を、想定しているのではと思います。

仮に、USBケーブルから 5Vを引き出す場合、最大 200[mA]までという条件が ありますが
USBで提供される電源でマイコンを動かす場合は、何の問題も生じません。
　今回のように、5Vで STEPモーター(1相当たり 1[A])を使う事は、電流容量の問題で、
USBからの電源は、使用出来ません。 よって容量の大きい、別の 5[V]電源を
使用する事になります。
この際に、別電源の電源スイッチを切った状態で、パソコンのUSBが生きていると
USBシリアル通信ICモジュールのTxD(送信データ端子)から、3.3Vが 出てるのです。
流せる電流は、少ないと思いますが、USBシリアル通信ICモジュールのTxD端子から
マイコンの、RxD 端子に電流が流れ込みます。
そして、マイコンの電源端子から電流が流れ出ているのです。
これは、通常の状態と逆に電流が、流れている訳で、マイコンを壊しかねません。

これに気づいたのは、外部電源ユニットの電源を切っている状態で、マイコンのVcc-GND間に、0.7Vの電圧が出ていました。 試しに電源ユニットの＋5V側のリード線を外したらマイコンの Vcc-GND間の電圧が、**2.5V** に、跳ね上がりました。 ビックリしました。
他に、電流が流れ込む場所としては、USBシリアルモジュールしかないと思い調べたらやっぱり、USBシリアルモジュールの TxD側から電流が流れ出ていました。

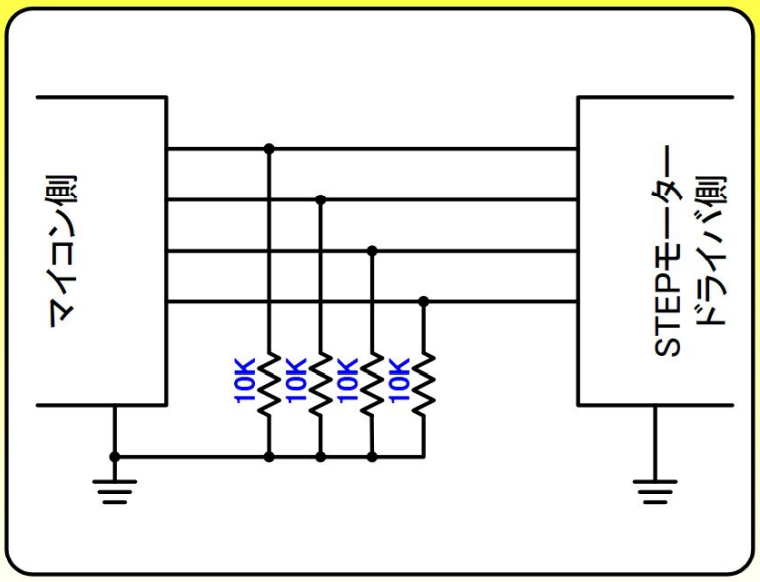
対策としてマイコンと、USBシリアルモジュールの2本の通信線の間に 220Ωの抵抗を入れ、特にUSBシリアルモジュールの Txd側から、220Ωを通ってマイコンの RxDに入る側に、ショットキーバリアダイオードのアノード側を接続しダイオードのカソード側をマイコンのVcc側に接続しました。
（ マイコン側電源OFF時の、電流の流れ道のバイパスを作ったという事です。）



この処置を行ってからマイコンの調子が良くなってソフト開発が、スムーズに出来るようになりました。

(4) 今回は、必要無かったかもしれませんが、4素子オープンコレクタのドライバICの入力端子に、10KΩの PullDown抵抗を入れました。

マイコンの、入出力ピンの初期化（各ピンの入力、出力の方向を設定）を行う前は、全てのピンが、入力になっています。
出力に設定するピンは、相手側が入力ピンなので、出力に設定する前は、線の両側が入力ピンでハイインピーダンス状態となりノイズが乗りやすいです。
特に、ドライバICが、MOSの場合ノイズ誤動作のリスクが高いと思います。 Hiレベルで、ONする場合は、ONしないように、Low側に抵抗で、弱く引っ張る対策をします。
このような用途の抵抗をプルダウン抵抗といいます。



(5) 台形制御(三角制御)の必要性:

STEPモーターは、本来 位置決め用のモーターなので、高速で回る事が苦手なモーターです。(高速で回せば、回すほどトルクが弱くなります。)
高速で回すという事は、STEPモーターの励磁パルスの周波数が高くなる事を意味します。
モーターコイルは、インダクタンス性負荷なので周波数が高くなるほど インピーダンスが上昇し電流が流れにくくなるので、高速で回るほど トルクが弱くなります。
また、STEPモーターに接続する機械的負荷(摩擦負荷や 回転系の慣性質量)の影響で、急激な加速、減速を行うと、脱調(入力励磁パルスに同期して動けなくなる現象)が、発生する事があります。

脱調を起こさないために、停止した状態から、徐々に加速して目標速度に達するように制御したり、停止位置の手前から徐々に減速して、スムーズに停止位置に停止させる制御を、台形制御といいます。
(移動距離が短くて目標速度に達する前に、減速を始めないといけない場合を、三角制御といいます。)
今回、この台形制御のプログラムを作って、
実際に、モーターを回してみようと思います。

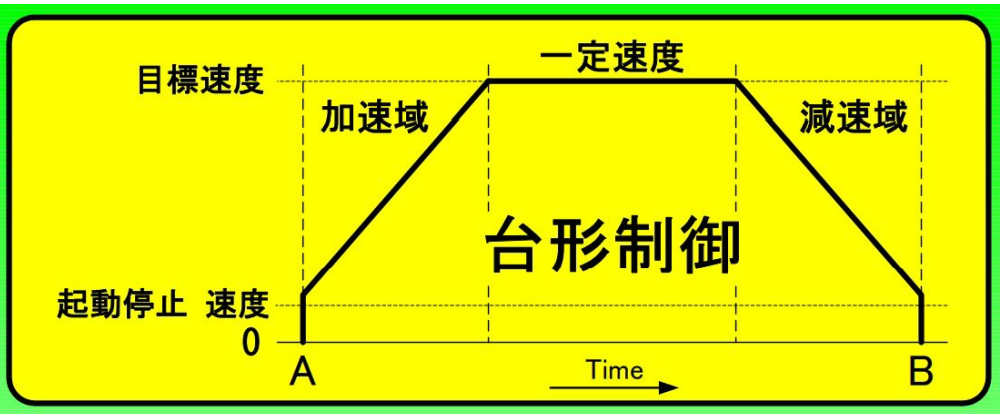
(6) 台形制御 (三角制御) 動作のグラフ:

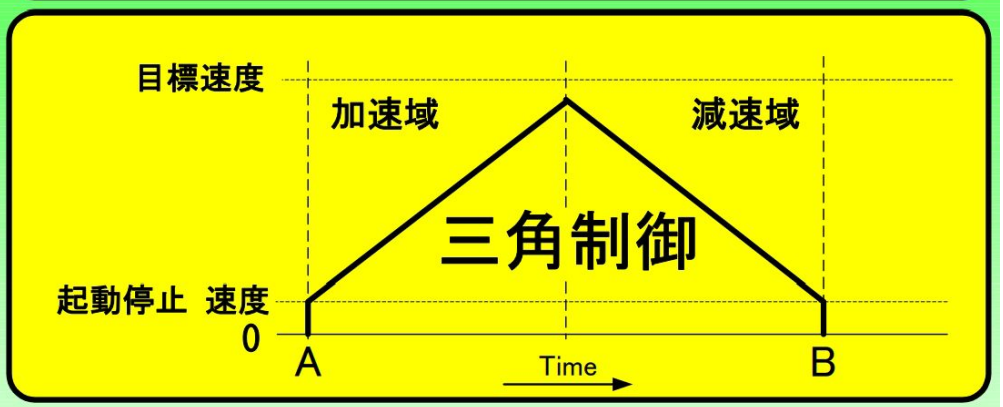
出発点Aから、目標点Bまで STEPモーターを動かすのに、台形制御の場合:
1. 加速域
2. 一定速度域
3. 減速域

の3つの段階を通って目標点Bに、移動します。

Aから、Bの距離が短い三角制御の場合:
1. 加速域
2. 減速域

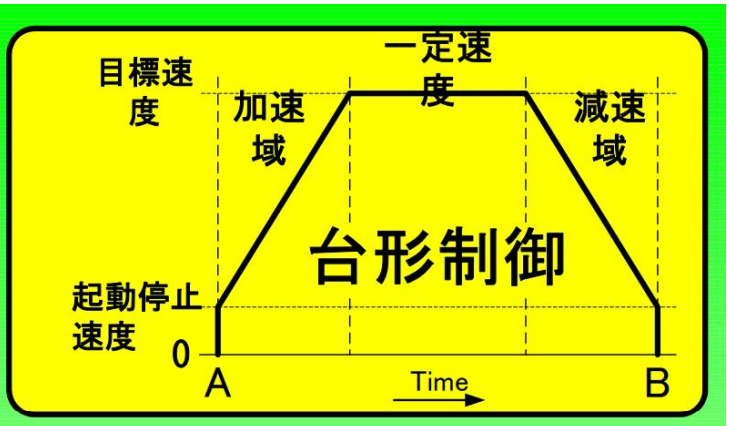
の2段階で、移行します。





右のグラフに表示されているパラメータ:

1. 起動停止速度:
   STEPモーターは、低速域にて段階的にカクカク動くので、滑らかに周り出すパルス周波数から回し始めます。
2. 目標速度:
   加速後、一定速度で移動する速度です。
3. 加速域(加速フェーズ):
   モーター起動直後は、加速フェーズ(加速段階)になります。
4. 一定速度域(一定速度フェーズ):
   指定された速度で、一定に駆動される段階です。
5. 減速域(減速フェーズ):
   目標点Bから、現在の地点を逆算して減速フェーズ(減速段階)になります。



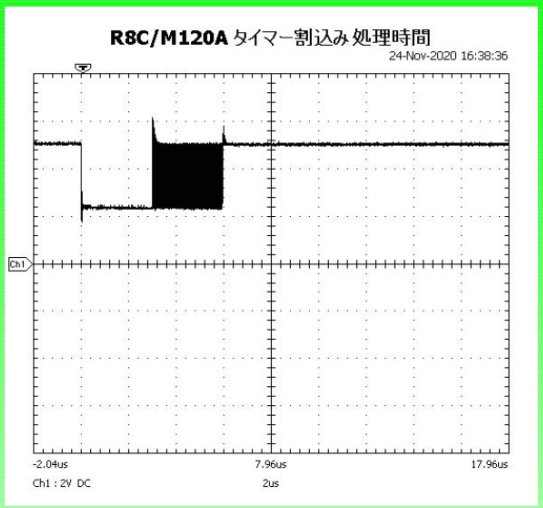
台形制御の要素として、加速、(等側)、減速の各フェーズがある事を、まず理解して下さい。

台形制御では、STEPパルスを実時間で出すプログラムにおいて、この3つの状態をスイッチする必要があるからです。

(7) STEPモーター駆動パルスを、どのように台形制御で出していくのか:

まず、STEPモーターの駆動方法ですが、前回の動画で、ユニポーラのSTEPモーターで1相駆動、2相駆動、1-2駆動の話をしました。 実際回している時は、4つの相のコイルに順番に、1, 2, 3, 4, 1, 2, 3, 4 と繰り返し 駆動電流を流していました。 この繰り返すタイミングを作り出すパルスを、駆動パルスと呼ぶ事にします。 駆動パルスを作るには、[ms]オーダーで、正確に時間を刻むタイマーが必要になります。 今回は、R8Cの RB2タイマー周辺回路を使用して、100[us](1/10000秒)分解能の タイマー割込みをCPUにかけて、インターバルタイマーを実現しました。 実際、STEPモーターを回すのは、最大 1[KHz]の駆動パルスで回す事にしています。 じゃ何故、10倍の 100[us]のタイマーを使うのかというと、分周カウンターて、分かりますでしょうか。? 1KHzを 1/2にしたら 500Hz、1/3にしたら、333Hz、1/4にしたら、250Hz と 分周する数字が大きくなると、周波数はその分低くなります。
そして、1/100 と 1/101 とかだったら、値は、僅かな差です。 ところが、早い周波数が欲しくて、1=1[kHz] 、その下の 1/2分周の 500[Hz]では、倍の差がありますよね。
滑らかな台形制御をしたいのに、速度の差が、2倍ではダメですよね。
だから、10倍の 100[us]を 用いて、1/10分周して 1[KHz]を作っているのです。
その下の周波数は、1/11分周で 909[Hz]になります。 これなら何とか滑らかといえます。

マイコンのハード寄りの話になって、ちょっと疲れたかもしれませんが、もうちょっと 頑張って下さい。 中には、じゃ、もっと早い割り込み周波数で回せばもっと滑らかになると思われた方もいるかもしれません。理論的には、合ってます。
但し、CPUの処理能力の限界があり、やたら早くできる訳ではありません。 割り込み処理の入口でLEDを点灯させ、出口で消灯させます。 それをオシロスコープで観測すれば、割込み処理にかかる時間が分かります。 右のグラフは実際に観測したグラフです。まだ、かなり余裕はあります。白いところ黒いところ合わせて 6[us]ぐらいです。黒いところは、1[ms]のカウント処理で、10回に1回しか入ってこないため、このようなグラフになるようです。 凡その指標として、割込み処理時間の最悪の長さが、割込み周期の半分未満を想定して下さい。 他にも割込み要因が複数ある場合は、あまり無理しない方がいいと思います。



割込みの話とか横道にそれたので戻します。タイマー処理でSTEPモーターの駆動パルスの周期を作るという話をしました。
今回、100[us]分解能タイマーに関わる関数を2つ用意しました。

[1] void set_tamer_100u(int n);
100[us]単位の時間待ち処理の時間設定関数です。
int n は、100[us]単位の待ち時間です。

[2] int get_timer_100u(void);
100[us]単位の時間待ち処理の経過時間を返す関数です。
関数値は、[1]で設定した値を減算していき0になったら減算を止めます。

```
//  STEPモーター駆動処理 メインループ（ サンプル ）
//=============================================
    set_timer_100u( pw );    // 初期パルス間隔時間設定
    for( i=0; i<nn; i++ )     // 目標値までの STEP数ループ
    {
       while( get_timer_100u() > 0 );  // 時間待ち
       if( MtrPm.Trp.sw == 1 ) // 台形制御 ON,OFF確認
         {
           trape_ctrl( i );      // 台形制御 有り
         }
       else
         {
            set_timer_100u( pw );  // 台形制御 無し
         }

     if( pol == 0 )  // モーター回転方向確認
       MtrPm.step_cnt++;  // STEPカウント（正転）
     else
       MtrPm.step_cnt--;  // STEPカウント（逆転）

       kudou( MtrPm.step_cnt );  // モーターコイル励磁更新
    }

//  台けい制御なしの場合、駆動パルス更新周期は、最初から
//  最後まで、同じで pw という変数に値が入ってます。
//  台形制御有りの場合は、trape_ctrl()関数内で、次の
//  駆動パルス更新周期を設定します。
```

では、台形制御で核心となる加速、減速の制御というか、連続した滑らかな分周タイマーの更新方法ですが、今回は 100円マイコンで複雑な演算処理は行わず、事前に計算した値を、テーブルデータとしてマイコンに持たせる事にします。 その方が、プログラムも簡単で処理速度も 早く出来ます。
今回、2種類ほどテーブルを作り実装しました。テーブルデータを作るのは、パソコン側で行いました。 実際の加速データを右に示します。
全部で、データが 118個あるので中間のデータ(11～100)を 削りました。
Tbl_Dat_B40: 下の .word 500 が、20Hzを表します。周波数:20[Hz]を 周期で表すと50[ms] となります。 50ms/100us = 500 になります。 一番下の 10は、1ms/100us = 10 で 1[KHz]です。

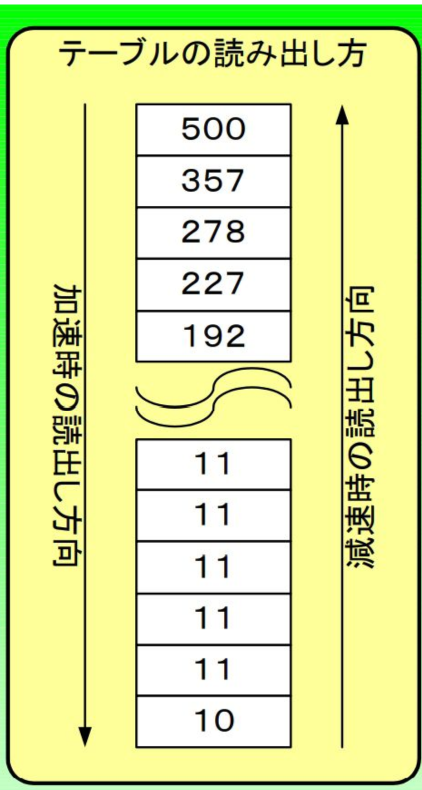
```
;   開始倍率：1.40   開始周波数： 20.0Hz
;------------------------------------------------
Tbl_Cnt_B40:                                    ; 1.40テーブル データ数
                .word    118
Tbl_Dat_B40:                                    ; テーブルデータ先頭
        .word   500   :   1
        .word   357   :   2
        .word   278   :   3
        .word   227   :   4
        .word   192   :   5

        .word   167   :   6
        .word   147   :   7
        .word   132   :   8
        .word   119   :   9
        .word   109   :  10
- - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        .word    12   : 101
        .word    12   : 102
        .word    12   : 103
        .word    12   : 104
        .word    12   : 105

        .word    12   : 106
        .word    12   : 107
        .word    11   : 108
        .word    11   : 109
        .word    11   : 110

        .word    11   : 111
        .word    11   : 112
        .word    11   : 113
        .word    11   : 114
        .word    11   : 115

        .word    11   : 116
        .word    11   : 117
        .word    10   : 118
```

字が小さくて
ごめんなさい。

.word 500 ; 1 (1番目のデータの意味)
(これより右はコメント)
は、周期 50[ms](20Hz)で 起動周波数です。

.word 10 ; 118 (118番目のデータの意味)
は、周期 1[ms](1KHz)で 最高速度周波数です。

よって先頭から順番に読み出し、値を 100usタイマーに設定する事により、滑らかに加速させる事が出来ます。 目標速度直前になったら、加速テーブル読み出し(加速フェーズ)から抜け出し、(一定速度フェーズ)に移行します。
減速する場合は、(一定速度フェーズ)直前に読み出したテーブル位置から、テーブルを先頭に遡るように読み出し、タイマーを設定すれば、減速状態になります。 テーブルを入れ替える事により、いろいろな加速、減速パターンを用意する事も可能です。
気付いた人もいると思いますが、アセンブラのインクルードファイルとして取り込んでいます。 Cでも、定数の配列は、宣言出来ますが、Cの場合は、データは基本的に RAMに置く事を前提としているので、テーブルを何本も作ると、RAMを、圧迫します。
アセンブラであれば、ROMにデータを置く事が出来ます。



(8) 加速テーブルの作り方：

R8Cマイコンに取り込んだ加速テーブルは、パソコン上で、Delphi(Object PASCAL)でテーブル生成プログラムを、作成しました。
　演算処理のソースを、右に示します。
今までの説明を凡そ理解していれば、何をやっているのか分ると思います。
えっ　言語が、Delphi !! と 言われそうですが、ネィティブコンパイラなので、VBのように手軽に作れて、且つ奥が深く、C++なみに実行速度は早いです。　私は気に入ってます。
余談：
　最近気付いたのですが、Linux環境で、Lazarusという Delphiに よく似た開発環境が、あります。
　パソコンのLinux上で、以前動かしました。
(現状 漢字入力に難があります。)
これを、RaspberryPiで 使ってみたいです。
制御系のプログラムが、作りやすい気がします。
(私だけかな。？)

字が小さくてごめんなさい。

```
procedure TForm1.calc_proc;
var
    a, aa, spd, mga, mg, k:     double;
    i, iv, ivl, n:     Integer;
    tx:          String;
    sw:          BYTE;
begin
    spd := Spm.freq_min;           // spd に起動周波数を設定
    n := 1;       i := 0;      sw := 0;
    iv := Round( Spm.freq_max );    // 最高速度を整数化して iv に入れる
    mga := Spm.magni - 1;           // 初期加速度 1.4から 0.4を取り出す

    while spd < Spm.freq_max  do    // 速度が最高速度に達しているか確認
    begin
        inc( i );                  // データ個数カウンタ＋＋
        if i>1000  then  break;    // 無限ループ対策 1000個 超えたら中断

        a := 1/ spd;               // 周波数の逆数：周期（時間）
        if sw = 0  then            // 最初に１回だけ実行
        begin
            aa := a;     sw := 1;  // aa に 最初の a の値( 周期を設定 )
        end;
        ivl := Round( a / Spm.resolu );  // a / 0.0001 (タイマー分解能 100us)
                                         // を、整数化して ivl に入れる
        pr_memo( n, ivl );         // Meemoコンポーネントに表示出力

        k := a / aa;               // パルス周期に従いだんだん小さくなる倍率
        mg := 1 + mga * k;         // 小数点以下の初期倍率に、かける
        spd := spd * mg;           // スピード変数に倍率をかける
        inc( n );                 // nは、表示上の１から始まる個数＋＋
    end;
end;
```

(8) 加速、減速のプログラム部分 1：

　右は、R8Cマイコンにて、モーター回転処理ループを、行う前に、加速フェーズと一定速度フェーズの境界(step数)を bound_1というメンバー変数に入れてます。
一定速度フェーズから、減速フェーズに入る境界(step数)を、bound_2というメンバー変数に入れてます。
hn = nn / 2; は、目標点までの移動step数を2で割った値を、hnに入れてます。
id = search_tspeed_index(MtrPm.speed);
は、テーブル内をサーチして、目標速度を超えない範囲で、最大値の加速テーブルのIndexを返します。　これにより目標速度手前まで加速フェーズで扱い、その後は一定速度フェーズに移行します。hn と id の大小関係で、台形制御か、三角制御かを判断してます。

```
//*********************************************
//**  台形制御／次の周期インターバルの設定  **
//** ------------------------------------- **
//**  nn :  目的位置までの ステップ数        **
//*********************************************
static  void trape_setup( int nn )
{
    int id, hn;

    MtrPm.Trp.mov_cnt = nn;
    hn = nn / 2;
    id = search_tspeed_index( MtrPm.speed );
    if( hn > id )
    {   // 加速、減速間に 一定速度のSTEPがある ( 台形制御 )
        MtrPm.Trp.bound_1 = id;
        MtrPm.Trp.bound_2 = nn - id;
    }
    else
    {   // 加速、減速間に 一定速度のSTEP 無し ( 三角制御 )
        MtrPm.Trp.bound_1 = hn;
        MtrPm.Trp.bound_2 = nn - hn;
    }
}
```

(8) 加速、減速のプログラム部分 2:

右は、R8Cマイコンにて、モーター回転処理ループ内で、加速フェーズ、一定速度フェーズ、減速フェーズを判断して、周期インターバルの設定(次の駆動パルスを出すタイミング)を、行っています。

前ページの、trape_setup()と組み合わせて見ると分かると思います。

余談:

台形制御のプログラムは、今回初めて作りました。 プログラム開発に関して、ここに至るまでのステップは、長かったのですが、今にして思うとこんな簡単なロジックでよく出来たな。
と、思います。

```
//*****************************************
//**  台形制御／次の周期インターバルの設定  **
//** ------------------------------------- **
//**  step :  目的位置までの、現在位置を     **
//**          示すステップ数                 **
//*****************************************
static  void    trape_ctrl( int step )
{
    inti, pw;
    char       tx[40];

    pw = MtrPm.speed;  // 仮に定速走行の速度（周期）を設定する
    if( step < MtrPm.Trp.bound_1 ) // 加速位置の範囲内か？
    {                  // 加速フェーズ
        pw = get_step_interval( step );    // 加速テーブル読出し
    }
    else
    {
        if( step >= MtrPm.Trp.bound_2 )    // 減速位置に入ったか？
        {                       // 減速フェーズ
            i = MtrPm.Trp.mov_cnt - step -1;  // 減速読出し位置計算
            if( i < 0 )    i = 0;
            pw = get_step_interval( i );   // 加速テーブル減速読出し
        }
    }
    set_timer_100u( pw );     // 周期インターバル設定
}
```

(9) モーター駆動ポートのアクセスについて:

右は、R8Cマイコンにて、モーター回転のための、励磁シーケンス処理です。

前回の動画で、モーターの4相ある電磁石の励磁シーケンスです。整数の下位 2bit または 下位 3bit を使って、STEPモーターの1相励磁、2相励磁では、0, 1, 2, 3, 0, 1, 2, 3 のシーケンスを作っています。
1-2相励磁では、下位 3bit で 0, 1, 2, 3, 4, 5, 6, 7 の繰り返しシーケンスを作っています。
それと、I/Oポートを頻繁にアクセスする場合は、bit単位のアクセスではなく、byte単位でまとめてアクセスした方が、実行速度が向上します。
STEP_MTRの #Define は、Port3を定義してます。
Port3には、4本のモータードライバに接続された4bitがあります。右のプログラムでは、
4bit まとめて Byte単位で 設定しています。

字が小さくてごめんなさい。

```
static    void      kudou( WORD pc )
{
   BYTE  k;

   pc = pc & 0x0007;
   switch( MtrPm.d_mode )
   {
      case  0:  // 停止モード
                STEP_MTR = 0;
                break;
       case  1:  // 1相 励磁駆動モード
                 k = pc % 4;
                if( k == 0 )   STEP_MTR = SBT_X;
                if( k == 1 )   STEP_MTR = SBT_Y;
                if( k == 2 )   STEP_MTR = SBT_IX;
                if( k == 3 )   STEP_MTR = SBT_IY;
                 break;
       case  2:  // 2相 励磁駆動モード
                 k = pc % 4;
                if( k == 0 )   STEP_MTR = SBT_X + SBT_Y;
                if( k == 1 )   STEP_MTR = SBT_Y + SBT_IX;
                if( k == 2 )   STEP_MTR = SBT_IX + SBT_IY;
                if( k == 3 )   STEP_MTR = SBT_IY + SBT_X;
                 break;
       case  3:  // 1-2相 励磁駆動モード
                 k = pc % 8;
                 if( k == 0 )   STEP_MTR = SBT_X;
                 if( k == 1 )   STEP_MTR = SBT_X + SBT_Y;
                 if( k == 2 )   STEP_MTR = SBT_Y;
                 if( k == 3 )   STEP_MTR = SBT_Y + SBT_IX;
                 if( k == 4 )   STEP_MTR = SBT_IX;
                 if( k == 5 )   STEP_MTR = SBT_IX + SBT_IY;
                 if( k == 6 )   STEP_MTR = SBT_IY;
                 if( k == 7 )   STEP_MTR = SBT_IY + SBT_X;
                  break;
   }
}
```

(10) 構造体変数の宣言に ついて:

前ページの制御プログラム内にて構造体変数をいくつか使っていたので、構造体変数になじんでない方には、意味が分かりにくいところが あったかと思います。

プログラム先頭で、構造体宣言している箇所を、右に示します。

上半分の TRAPEZOID_PARAM は、台形制御に関わる変数を、一つにまとめた物です。

MOTOR_DRV_PARAM は、モーター制御全体に関わる変数です。 この中に Trpという名前で、台形制御パラメータが 入ってます。

例えば、台形制御パラメータの ON,OFFスイッチは、MtrPm.Trp.sw でアクセス出来ます。

構造体変数は、一つの機能を実現するために 必要な変数を、1本にまとめたものといえます。

```
//  ***  構造体データ宣言  ***
// ------------------------------------------------------
typedef struct  {
    WORD    cur_tbl; // 現在選択しているテーブル番号
    WORD    t_count; // テーブル内のデータ数
    WORD    mov_cnt; // 今回の移動ステップ数
    WORD    bound_1; // Ph.1から Ph.2 の境界（目標速度 開始位置）
    WORD    bound_2; // Ph.2から Ph.3 の境界（減速 開始位置）

    BYTE    sw;      // 台形制御 ON,OFF スイッチ
    BYTE    pha_sw;  // フェーズスイッチ 1=加速、2=目標速度、3=減速
}  TRAPEZOID_PARAM;    // 台形制御パラメータ構造体

typedef struct  {
    WORD    con_stp;    // 継続した移動ステップ数
    WORD    speed;      //  目標速度
    WORD    step_cnt;   // 継続したステップカウンタ
    BYTE    d_mode;     // ドライブモード 1=1相駆動、
                        //  2=2相駆動、3=1-2相駆動
    TRAPEZOID_PARAM  Trp;   // 台形制御パラメータ
} MOTOR_DRV_PARAM;      // モーター駆動パラメータ構造体

 MOTOR_DRV_PARAM  MtrPm;  // モーター駆動パラメータ変数
```