

マイコンカーラリー用
ロータリエンコーダ
実習マニュアル
kit06版

第 1.10 版以降では、ヘッダファイルや共通の C ソースファイルは「c:¥workspace¥common」フォルダに置き、そのファイルを参照するよう変更しました。

第 1.20 版
2007.02.27
ジャパンマイコンカーラリー実行委員会

注意事項 (rev.1.1)

著作権

- ・本マニュアルに関する著作権はジャパンマイコンカーラリー実行委員会に帰属します。
- ・本マニュアルは著作権法および、国際著作権条約により保護されています。

禁止事項

ユーザーは以下の内容を行うことはできません。

- ・第三者に対して、本マニュアルを販売、販売を目的とした宣伝、使用、営業、複製などを行うこと
- ・第三者に対して、本マニュアルの使用権を譲渡または再承諾すること
- ・本マニュアルの一部または全部を改変、除去すること
- ・本マニュアルを無許可で翻訳すること
- ・本マニュアルの内容を使用しての、人命や人体に危害を及ぼす恐れのある用途での使用

転載、複製

本マニュアルの転載、複製については、文章によるジャパンマイコンカーラリー実行委員会のこと前の承諾が必要です。

責任の制限

本マニュアルに記載した情報は、正確を期すため、慎重に制作したのですが万一本マニュアルの記述誤りに起因する損害が生じた場合でも、ジャパンマイコンカーラリー実行委員会はその責任を負いません。

その他

本マニュアルに記載の情報は本マニュアル発行時点のものであり、ジャパンマイコンカーラリー実行委員会は、予告なしに、本マニュアルに記載した情報または仕様を変更することがあります。製作に当たりましては、こと前にマイコンカー公式ホームページ(<http://www.mcr.gr.jp/>)などを通じて公開される情報に常にご注意ください。

連絡先

ルネサステクノロジ マイコンカーラリー事務局
〒162-0824 東京都新宿区揚場町 2-1 軽子坂MNビル
TEL (03)-3266-8510
E-mail:official@mcr.gr.jp

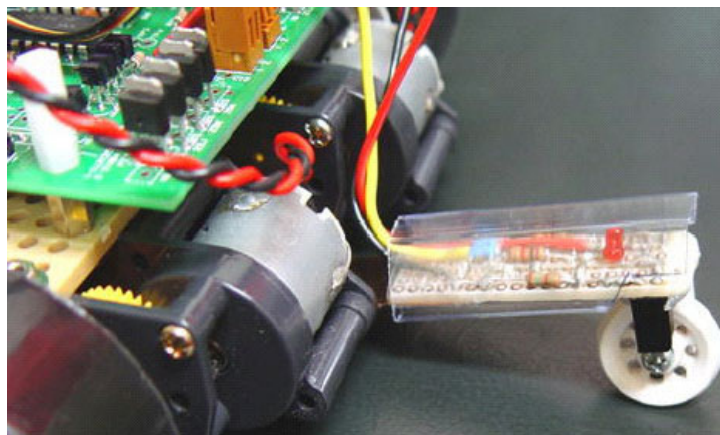
目 次

1. ロータリエンコーダを使う	1
1.1 ロータリエンコーダとは	1
1.2 原理	1
1.3 2相出力のエンコーダ	2
2. マイコンカーへの取り付け	3
2.1 マイコンカーで使えるエンコーダの条件	3
2.2 市販されているエンコーダを使う	3
2.2.1 エンコーダの例	3
2.2.2 回路	4
2.2.3 簡易回路	4
2.2.4 回転部分の加工	6
2.2.5 マイコンカーへの取り付け	7
2.2.6 即席の取り付け例	8
2.2.7 パルス数とスピード(距離)の関係	9
2.3 フォトセンサを使った自作	12
2.3.1 フォトインタラプタとは	12
2.3.2 透過型フォトインタラプタの例	13
2.3.3 回路	13
2.3.4 回転部分の加工	14
2.3.5 フォトインタラプタとプーリーの取り付け	14
2.3.6 マイコンカーへの取り付け	15
2.3.7 パルス数とスピード(距離)の関係	15
3. サンプルプログラム	18
3.1 ルネサス統合開発環境	18
3.2 サンプルプログラムのインストール	18
3.3 ワーススペース「kit06enc」を開く	19
3.4 プロジェクト	20
4. プロジェクト「kit06enc_01」 kit06.c をエンコーダが使用できるように改造	21
4.1 プロジェクトの構成	21
4.2 プログラム	21
4.3 ロータリエンコーダの接続	24
4.3.1 標準キット kit06 の接続の確認	24
4.3.2 ロータリエンコーダを接続	25
4.3.3 確認用 LED の接続	27
4.4 プログラムの解説	29
4.4.1 エンコーダ関連の変数の宣言	29
4.4.2 パターン 0: エンコーダ値をポート 4 へ出力	29
4.4.3 入出力設定の変更	30
4.4.4 外部パルス入力設定	30
4.4.5 パルスカウントを 2 倍にする方法	32
4.4.6 ITU0 割り込み処理	33
4.4.7 更新する間隔について	34
4.4.8 ITU2_CNT が 65535 から 0 になったとき	34

4.4.9	なぜ、バッファを使うのか	35
5.	プロジェクト「kit06enc_02」 速度の調整	36
5.1	プログラム	36
5.2	プログラムの解説	39
5.2.1	パターンの表示	39
5.2.2	パターン 12 右大曲げときの処理	39
5.2.3	speed2 関数	40
5.2.4	パターン 13 左大曲げときの処理	41
5.2.5	パターン 23 クロスライン後のトレース、クランク検出ときの処理	42
5.3	エンコーダの回転数が違う場合の変更点	43
6.	プロジェクト「kit06enc_03」 距離の検出(パターンの区分けを距離で行う)	44
6.1	プログラム	44
6.2	プログラムの解説	46
6.2.1	変数の追加	46
6.2.2	積算値のクリア	46
6.2.3	パターン 21 クロスライン検出ときの積算値を取得	47
6.2.4	パターン 22 2本目を読み飛ばす	48
6.2.5	パターン 51 右ハーフライン検出ときの積算値を取得	49
6.2.6	パターン 52 2本目を読み飛ばす	50
6.2.7	パターン 61~62 左ハーフライン部分の処理	52
6.3	エンコーダの回転数が違う場合の変更点	52
7.	プーリーを使用した自作エンコーダのプログラム	53
7.1	プーリーを使用したときの回転数計算	53
7.2	速度のチェック	55
7.3	距離のチェック	55
7.4	kit06enc_01.c を改造して、自作エンコーダに対応させる場合の変更	56
7.5	kit06enc_02.c を改造して、自作エンコーダに対応させる場合の変更	56
7.6	kit06enc_03.c を改造して、自作エンコーダに対応させる場合の変更	56
8.	参考文献	57

1. ロータリエンコーダを使う

マイコンカーの中には、本体の後ろにタイヤが付いているマシンがあります。これがロータリエンコーダと呼ばれる装置です。



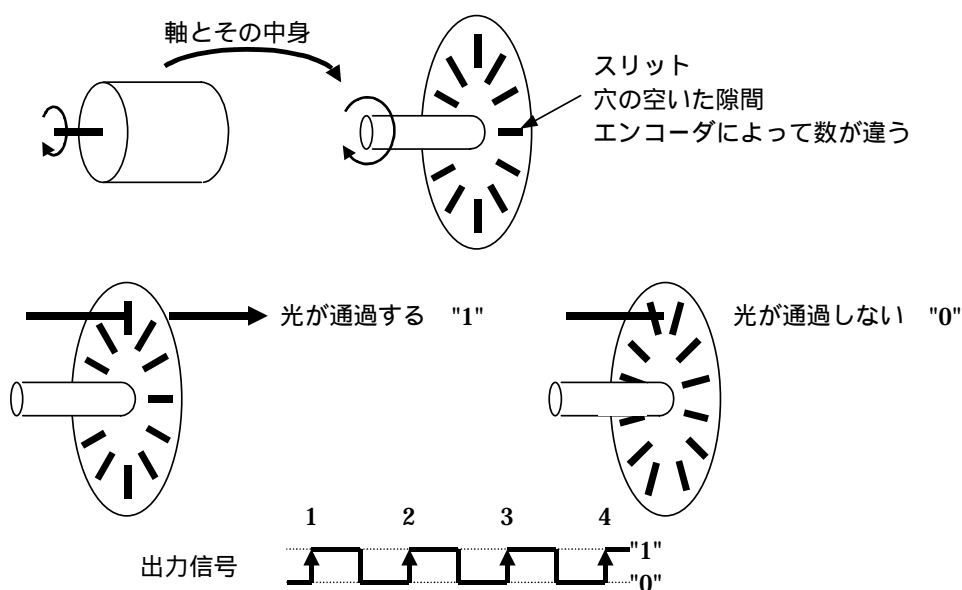
マイコンカーに取り付けたロータリエンコーダ(自作)

1.1 ロータリエンコーダとは

ロータリエンコーダとは、どのような物でしょうか。「ロータリ(rotary)」は、「回転する」という意味です。「エンコーダ(encoder)」は、電気でよく使われる言葉で「符号化する装置」という意味です。この頃、パソコンに映像を取り込んだり、音声を取り込んだりすることが流行っていますが、ビデオ信号を MPEG データに変換したり、音声信号を PCM データに変換することをエンコード(符号化)と言います。これらから、「ロータリエンコーダ」は、回転を符号化(数値化)する装置ということになります。

1.2 原理

原理は、回転軸に薄い円盤が付いています。その円盤にはスリットと呼ばれる小さい隙間を空けておきます。円盤のある一点に光を通して、通過すれば"1"、しなければ"0"とします。スリットの数は、1つの円盤に10個程度から数千個程度まで様々あります。当然スリット数の多い方が、値段が高くなります。

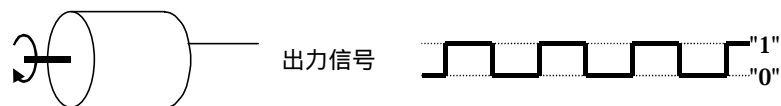


"0"から"1"になる回数を数えれば、距離が分かります。また、ある一定時間、例えば1秒間の回数をカウントして、多ければ回転が速い(=スピードが速い)、少なければ回転が遅い(=スピードが遅い)と判断できます。

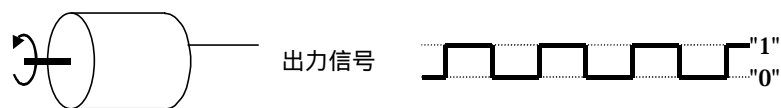
1.3 2相出力のエンコーダ

エンコーダには、1相出力と2相出力があります。先ほどの説明は、1相出力の場合です。1相の場合、回転が正転か逆転が分かりません。どちらも"1"と"0"の信号でしかないので、

正転時の波形



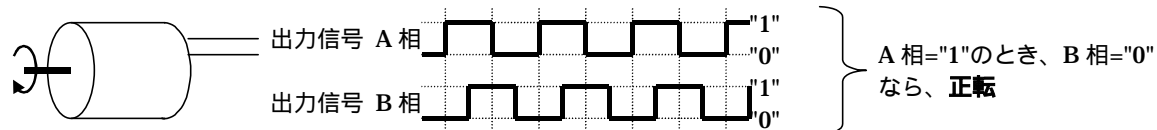
逆転時の波形



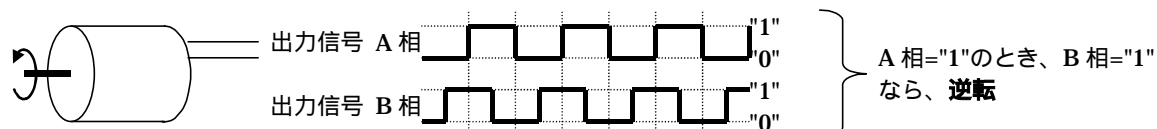
同じ！見分けが付かない

そこで、出力をA相という名前と、B相という名前の2つ出力します。同じ信号を出力しても意味が無いので、B相の光検出を90度分ずらして、A相より90度分ずれるようにしています。

正転時の波形



逆転時の波形



身近な例では、パソコンのマウス(光学式ではなくボール式)には2相のエンコーダが2つ付いています。1つが左右の検出、もう一つで上下の検出をしています。

2. マイコンカーへの取り付け

2.1 マイコンカーで使えるエンコーダの条件

エンコーダを探すといろいろな種類があります。どのようなエンコーダがマイコンカーに使えるのでしょうか。

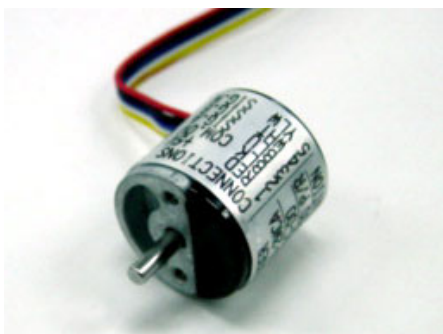
項目	内容
大きさ	走行に影響しない程度の大きさとして。小さければ小さいほど良いですが、高くなります。直径 20 ~ 30mm くらいまでが実用範囲内です。
重さ	軽いエンコーダを選びます。
出力信号	CPU は、基本的には"0"か"1"かのデジタル信号しか扱えないので、エンコーダから出力される信号もデジタル信号が理想です。出力電圧は、CPU に合わせて"0"=0V、"1"=5V だとポートに直結、もしくは 74HC14 などのゲートをかませるだけで簡単に接続できます。正弦波などのデジタル信号ではない場合は、増幅回路やコンパレータなどの回路を外付けしてデジタル信号に変換する必要があります。
動作電圧	CPUと同様の5Vで動作するのが理想です。マイコンカーで使用できる電源は、電池8本までなので、上限は9.6Vの電圧となります。
パルス数	多いにこしたことはありません。1回転20パルス以上あればマイコンカーで使用可能です。

2.2 市販されているエンコーダを使う

2.2.1 エンコーダの例

市販されているエンコーダでマイコンカーに使用できそうなエンコーダを以下に示します。他にもたくさんありますので、調べてみると良いでしょう。

メーカー	型式	特徴
日本電産ネミコン(株)	OME-100-1CA-105-015-00	デジタル信号が出力されるので、マイコンで扱いやすいです。プルアップ抵抗の追加だけで使用可能です。
日本電産コパル(株)	RE12D-100-101-1	デジタル信号が出力されるので、マイコンで扱いやすいです。プルアップも不要です。12mmと小型です。



OME-100-1CA-105-015-00



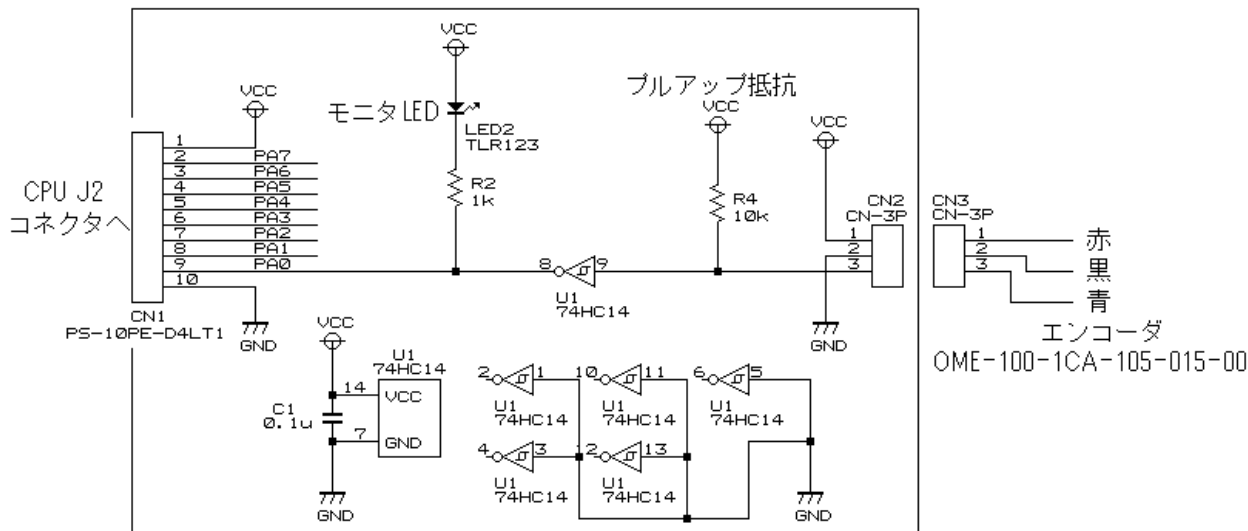
RE12D-100-101-1

2.2.2 回路

日本電産ネミコン(株)「OME-100-1CA-105-015-00」を例に説明します。

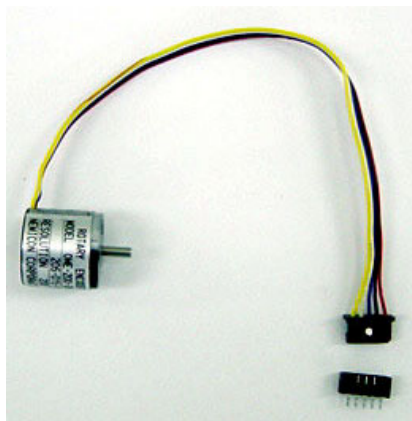
「OME-100-1CA-105-015-00」の出力信号は、デジタル信号のため、そのままポートに接続可能です。ただし、オープンコレクタ出力なのでプルアップは必要です。一応、74HC14 で波形整形すると良いでしょう。

CPU ボードのポート A の bit0 にエンコーダ信号を接続する回路を下記に示します。モニタ LED は、信号が来ているか確認するのに便利です。付けるスペースがあるなら、付けましょう。



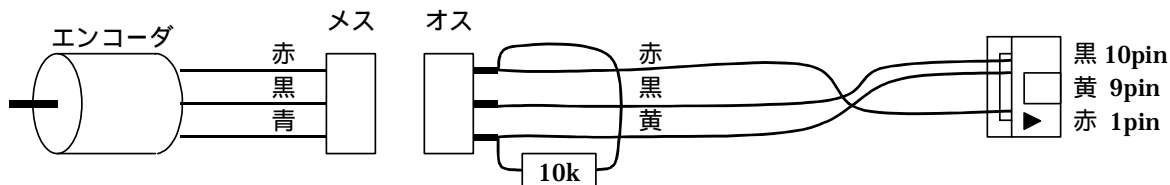
2.2.3 簡易回路

日本電産ネミコン(株)「OME-100-1CA-105-015-00」を例に説明します。



写真は、「OME-100-2MCA-105-015-00」のエンコーダのため、5 ピンコネクタですが、「OME-100-1CA-105-015-00」は 3 ピンコネクタになります。

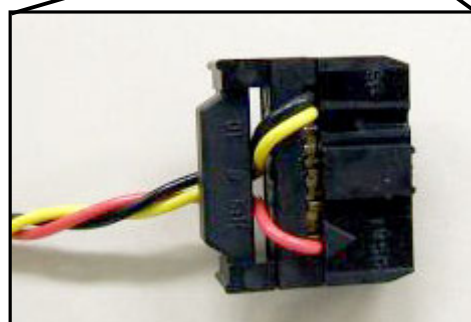
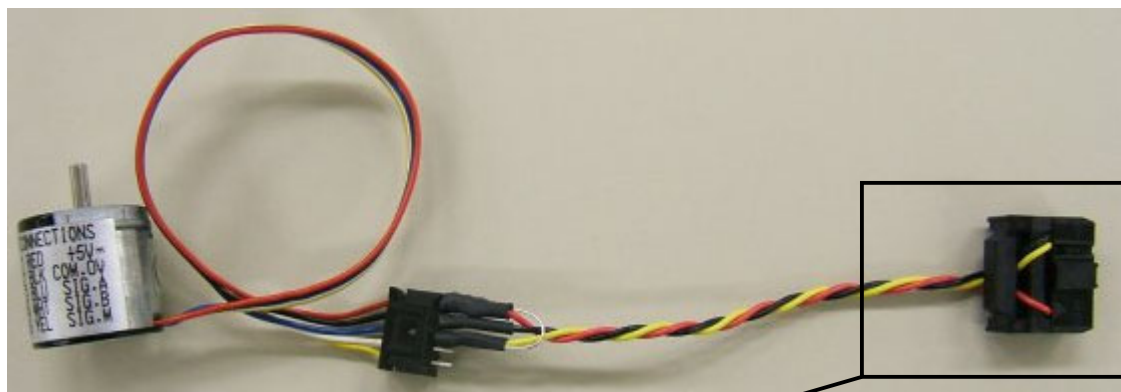
下記のように配線します。



CPU ボードのポート A のコネクタへ直接接続します。10 ピンメスコネクタに赤、黒、黄色の線を配線します。

エンコーダ線	接続先	10 ピンコネクタピン番号
赤	+5V	1 ピン
黒	GND	10 ピン
青	PA0	9 ピン

製作例



2.2.4 回転部分の加工

エンコーダの軸にタイヤを取り付け、コース上に接地しながら回転するようにします。



ホイールとして、タミヤの「プーリー(S)セット」を使用します。直径 10mm のプーリーが 4 個、20mm が 2 個、30mm が 2 個入っています。10mm は径が小さすぎて使えませんので、直径 20mm が 2 個、30mm が 2 個使えます。

タイヤとして付属の輪ゴムを使うと、結び目でガタガタしてしまいます。そのため、今回はホームセンターなどで売っているオリングを選びました。写真は東急ハンズで売っていたオリングです。「1A P15」と書いてある袋には、20mm 径のオリングが 10 個入っています。「1A P25」と書いてある袋には、30mm 径のオリングが 10 個入っています。共に 315 円でした。



30mm のプーリーにオリングをはめたところでは、ロータリエンコーダの軸の直径は 2.5mm です。プーリー(S)セットには 2mm と 3mm 径のブッシュ(プーリーの中心の黒い部品)しかありません。そのため、2mm 径のブッシュに 2.5mm のドリルで穴を開けて、エンコーダに取り付けます。

オリングをはめたタイヤの直径は、実測で 33mm になりました。



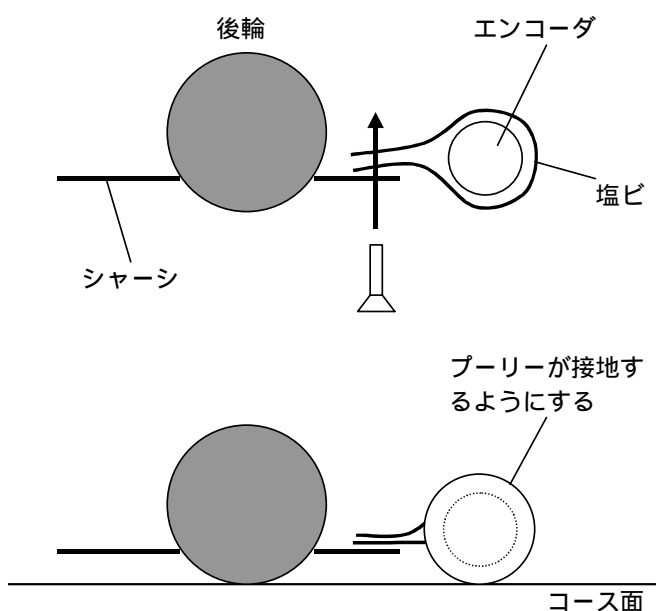
エンコーダにプーリーを取り付けました。軸とプーリーをボンドで固定すれば、はずれる心配がありません。

2.2.5 マイコンカーへの取り付け

タイヤを付けたエンコーダを、マイコンカーに取り付けます。

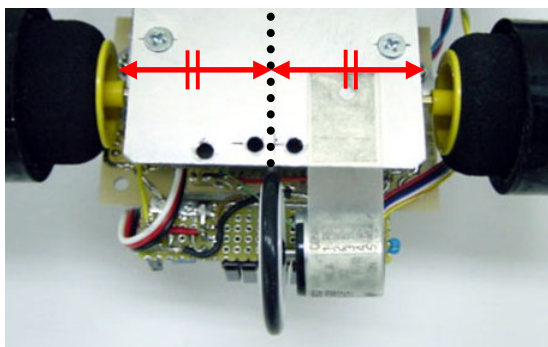


0.5mm 厚の塩ビ板です。薄く弾力性のある素材であれば何でも構いません。



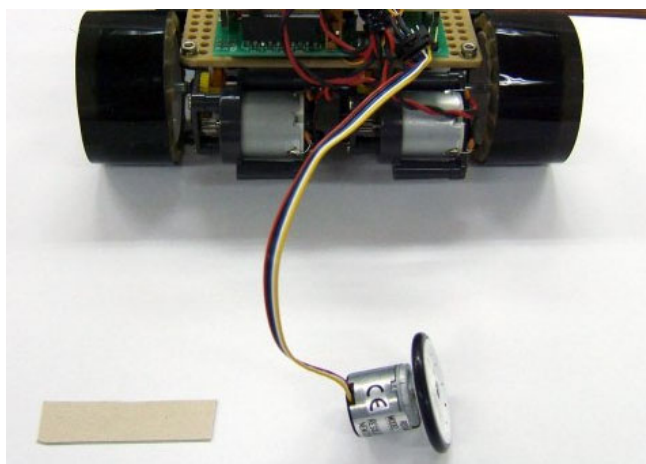
塩ビなどの弾力のある素材で、エンコーダを巻くようにします。エンコーダと塩ビは、両面テープで止めます。塩ビの両端を合わせて、マイコンカー本体のシャーシにネジ止めします。1箇所だとゆるみやすいので、2箇所以上で止めます。

マイコンカーをコースに置いたとき、接地するようにします。圧力が強すぎると、走行に影響するので軽く圧力がかかるようにしてください。

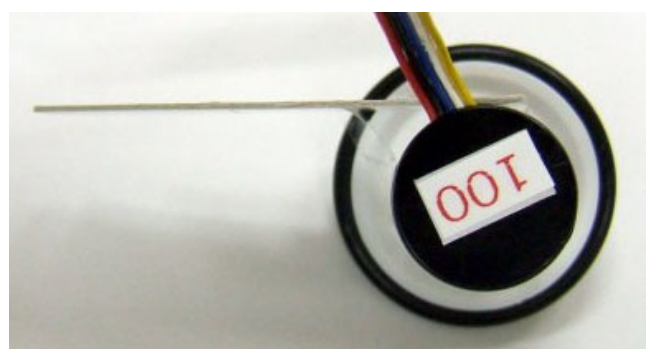


両面テープで簡単に取り付けた例です。ちょうど中心にくるように貼り付けます。

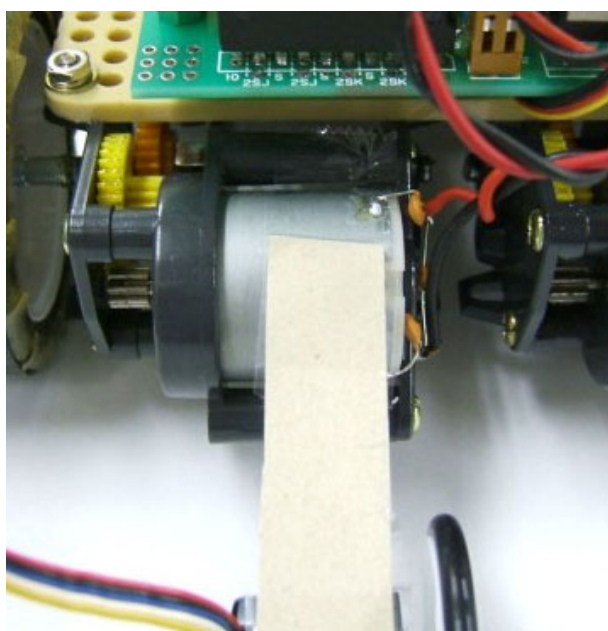
2.2.6 即席の取り付け例



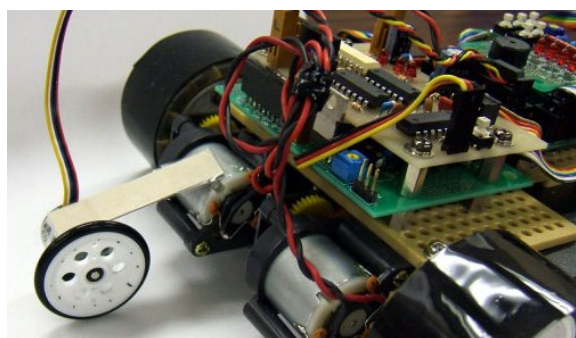
取り付けるマイコンカー、エンコーダ、15 × 50mm 程度の厚紙(硬めの板)とセロテープを用意します。



セロテープでエンコーダと厚紙を止めます。がっちり止めます。



マイコンカー側もセロテープで厚紙を止めます。こちらは上り坂、下り坂でもエンコーダが接地するように、多少上下するようしておきます。



斜め横から見たところですが、エンコーダのタイヤがコースに設置するようにします。

この方法では、すぐに取りれてしまうので、実験のみの使用にしましょう。

2.2.7 パルス数とスピード(距離)の関係

どのくらい進むと何パルスの信号がエンコーダから出力されるのかわからなければ、プログラムできません。下記条件のエンコーダ、タイヤとします。

項目	内容
エンコーダの1回転のパルス数	100 パルス / 回転
タイヤの直径(実寸)	33mm

(1) タイヤが1回転したときのパルス数の計算

タイヤの直径から、円周がわかります。

$$\text{円周} = 2 \times r = 33 \times 3.14 = 103.62\text{mm}$$

エンコーダは 100 パルス / 回転なので、

103.62mm 進むと 100 パルス

となります。

(2) 1m 進んだときのパルス数の計算

パルス数は、距離と比例します。(1)より、1m 進んだときのパルス数は、

$$100 \text{ パルス} : 103.62\text{mm} = x \text{ パルス} : 1000\text{mm}$$

$$x = 965 \text{ パルス}$$

1m (1000mm)進むと、965 パルス

となります。

(3) 秒速 1m で進んだとき 1 秒間のパルス数の計算

(2)より、

1m/s の速さで進んだとき、1 秒間のパルス数は 965 パルス

となります。

(4) 秒速 1m で進んだとき、10ms 間のパルス数の計算

(3)より、

$$1 \text{ 秒} : 965 \text{ パルス} = 0.01 \text{ 秒} : x \text{ パルス}$$

$$x = 9.65 \text{ パルス}$$

1m/s の速さで進んだとき、10ms 間のパルス数は 9.65 パルス

となります。

(5) プログラムで速度を検出する

(4)より、1m/s で進んだとき、10ms 間のパルス数は 9.65 パルスです。
現在の速度は、下記で求めることができます。

$$\text{現在の速度 [m/s]} = 10\text{ms 間のパルス数} \div 9.65 \quad \dots \boxed{A}$$

「10ms 間のパルス数」部分が、マイコンカーの実際の走行スピードにより変化します。

例えば、秒速 2m/s 以上ならモータの PWM を 0%、それ以下なら PWM を 70%にするなら、下記のようになります。

```
if( 現在の速度 >= 2m/s ) {  
    PWM を 0%にする  
} else {  
    PWM を 70%にする  
}
```

プログラムで記述します。「10ms 間のパルス数」は、変数「iEncoder」とします。

\boxed{A} より

$$10\text{ms 間のパルス数} = \text{現在の速度} \times 9.65$$

$$i\text{Encoder} = 2 \times 9.65$$

$$i\text{Encoder} = 19.3$$

変数は、整数しか扱えないので四捨五入します。

$$i\text{Encoder} = 19$$

プログラムは、下記のようになります。

```
if( iEncoder >= 19 ) {  
    speed( 0, 0 );  
} else {  
    speed( 70, 70 );  
}
```

変数 iEncoder について

10ms 間ごとに iEncoder 変数を更新する作業は、プログラムで行います。更新する間隔が短いほど最新のスピードが分かりますが、パルス数が少なくなるため精度が悪くなります。更新する間隔が長いほど精度が良くなりますが、最新の速度が分かりません。10ms ごとにカウントするのが、経験上良いかと思います。

(6) プログラムで距離を検出する

(2)より、1m 進んだときのパルス数は、965 パルスです。
進んだ距離は、下記で求めることができます。

$$\text{進んだ距離 [m]} = \text{合計パルス数} \div 965 \quad \dots \text{B}$$

「合計パルス数」部分が、マイコンカーの進んだ距離によって変化します。

例えば、10m 進んだならモータの PWM を 0%、それ以下なら PWM を 100%にするなら、下記のようになります。

```
if( 進んだ距離 >= 10m ) {
    PWM を 0%にする
} else {
    PWM を 100%にする
}
```

プログラムで記述します。「進んだ距離」は、変数「lEncoderTotal」とします。

Bより

```
合計パルス数 = 進んだ距離 × 965
lEncoderTotal = 10 × 965
lEncoderTotal = 9650
```

プログラムは、下記のようになります。

```
if( lEncoderTotal >= 9650 ) {
    speed( 0, 0 );
} else {
    speed( 100, 100 );
}
```

(5) 1 回転 100 パルス以外、直径 33mm 以外の場合

エンコーダのパルス数が 1 回転 100 パルスと 1 回転 200 パルス、タイヤの直径 33mm と 21mm のとき、計算した結果をまとめると下表のようになります。

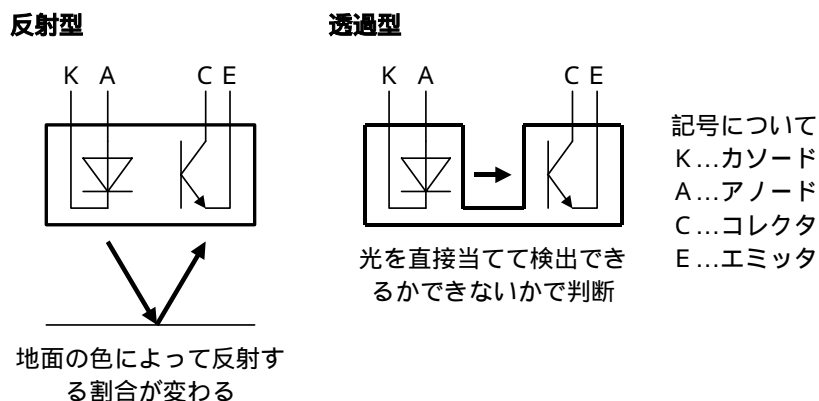
パルス数	100	100	200	200
タイヤ径[mm]	21	33	21	33
1m 進んだときのパルス数	1516	965	3032	1930
1m/s で進んだときの 10ms 間に カウントするパルス数	15.16	9.65	30.32	19.3

2.3 フォトセンサを使った自作

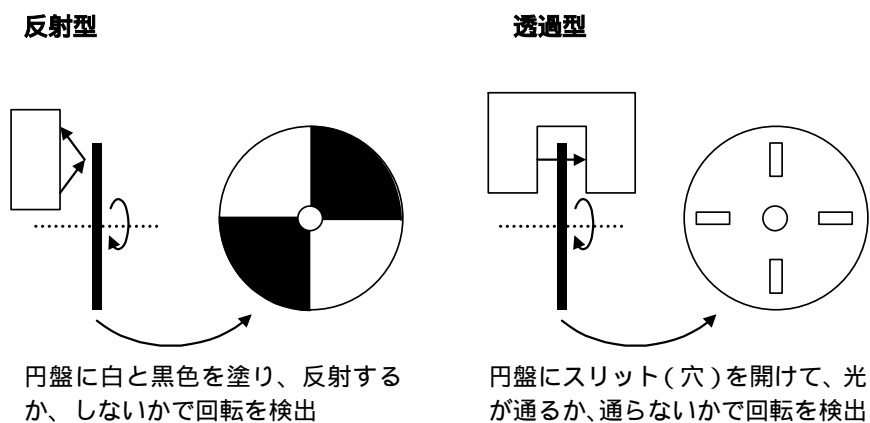
市販されているエンコーダは1回転100パルス以上と性能は申し分ありません。しかし値段が高いのが難点です。そこで、パルス数が少なくなりますが、安くできる方法を紹介します。

2.3.1 フォトインタラプタとは

フォトインタラプタとは、発光、受光が一体化した素子で、発光側には赤外発光LED、受光にはフォトランジスタなどが使われます。フォトインタラプタには、反射型と透過型と呼ばれるタイプがあります。



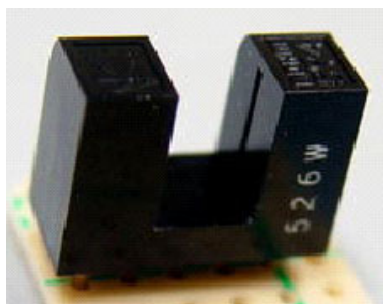
反射型、透過型のフォトインタラプタをエンコーダとして使用したときの例を下記に示します。それぞれ、取り付け方、円盤の加工の仕方が変わります。



2.3.2 透過型フォトインタラプタの例

市販されている透過型フォトインタラプタでマイコンカーに使用できそうなフォトインタラプタを以下に示します。他にもたくさんありますので、調べてみると良いでしょう。

メーカー	型式	特徴
ローム(株)	RPI-574	溝幅は 5mm あります。間にプーリーを入れることができます。フォトトランジスタ出力なので、デジタル信号に変換する回路が必要です。
シャープ(株)	GP1A51HRJ00F	溝幅は 3mm あります。間にプーリーを入れることができません。デジタル出力なので、直結可能です。



RPI-574

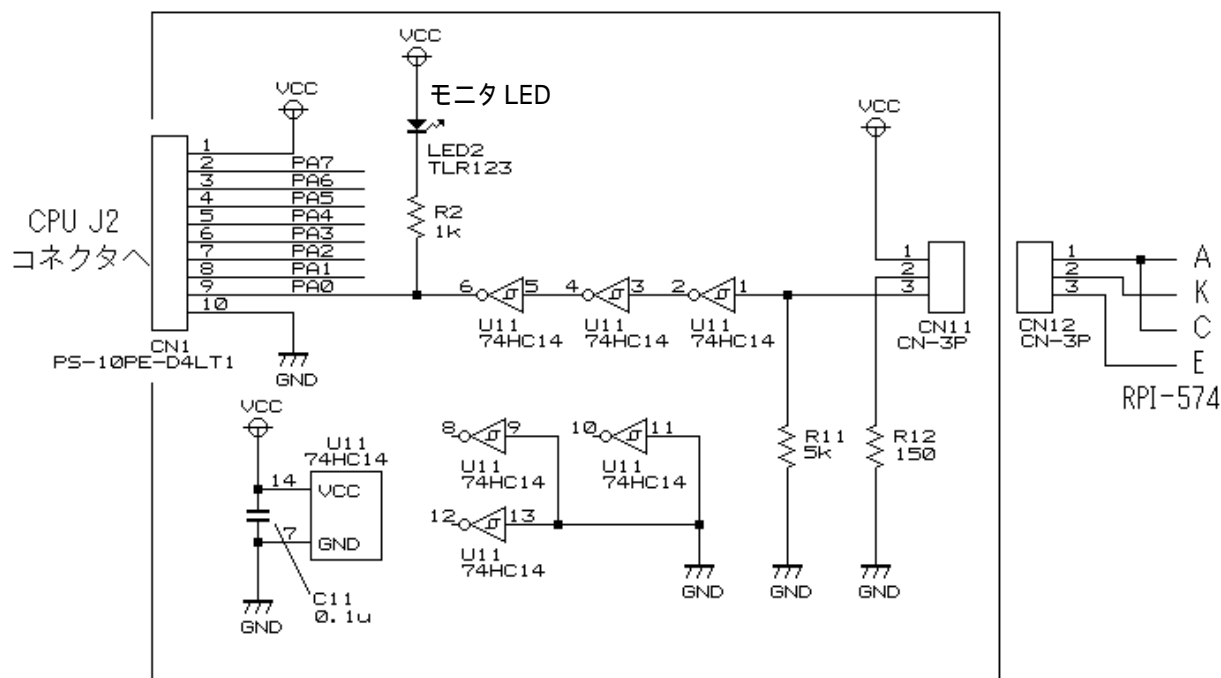


GP1A51HRJ00F

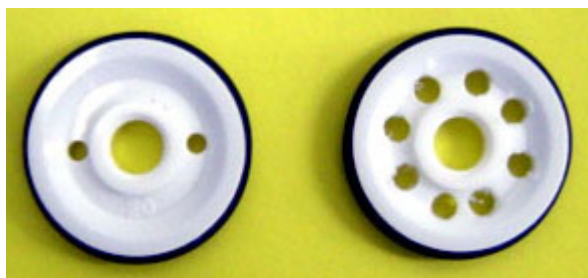
2.3.3 回路

ローム(株)「RPI-574」を例に説明します。

「RPI-574」の出力信号は、フォトトランジスタ出力なので、デジタル信号に変換する必要があります。といっても下記のような簡単な回路です。モニタLEDは、信号が来ているか確認するのに便利です。付けるスペースがあるなら、付けましょう。



2.3.4 回転部分の加工

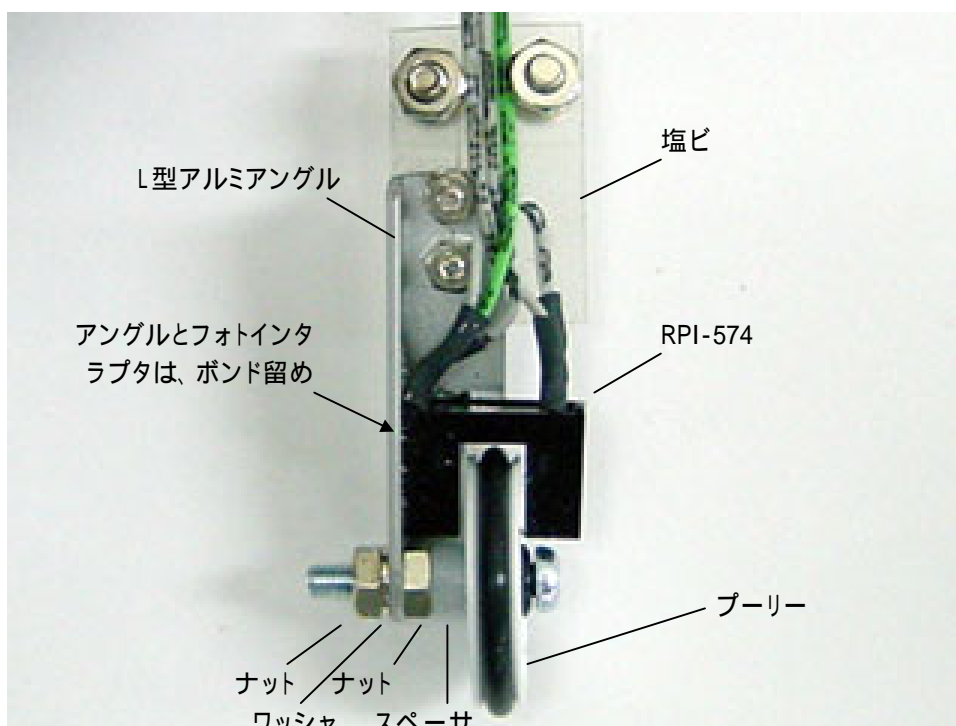


フォトインタラプタの間に入れる円盤として、先ほど紹介したタミヤの「プーリー(S)セット」を使用します。直径は小さい方が1周するのが早いので、直径の小さいプーリーを使用します。ただ、小さすぎると穴を開けられないので、今回は20mmのプーリーを使います。

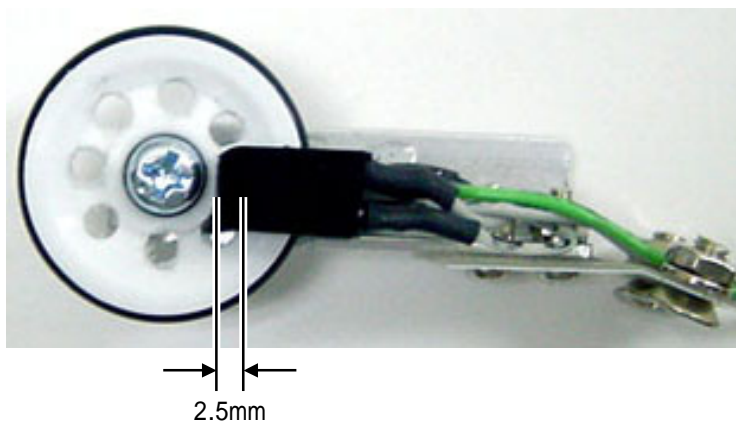
左写真の左が加工前、右が加工後です。2.5mm径のドリルで8箇所穴開けしました。このプーリーは、8パルス/回転ということになります。

2.3.5 フォトインタラプタとプーリーの取り付け

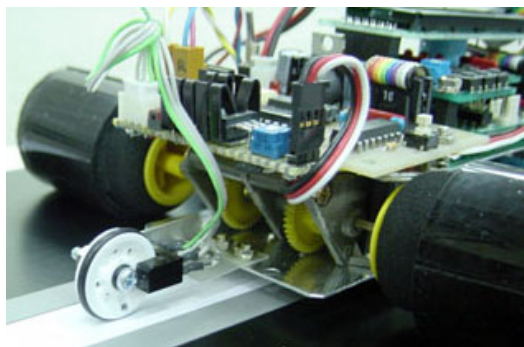
L型アルミアングルを使って、フォトインタラプタとプーリーをうまく配置します。スペーサを使って、フォトインタラプタの間にプーリーを入れます。



プーリーとフォトインタラプタの横の位置関係は、フォトインタラプタの発光素子部分に穴を開けた部分に来るようにします。RPI-574のデータシートより、2.5mmの位置になります。フォトインタラプタによって違いますので各データシートで確認してください。



2.3.6 マイコンカーへの取り付け



後部に取り付けます。プーリーが中心に来るようにします。中心でないと、右に曲がっているときと左に曲がっているときでは内外輪差が生じて回転数が変わります。

コース上に置いたときに、プーリーに軽く圧力がかかるようにしてください。圧力が軽すぎると空転してしまいます。圧力が強すぎると負荷になってしまい、走行に影響します。

2.3.7 パルス数とスピード(距離)の関係

どのくらい進むと何パルスの信号がエンコーダから出力されるのか分からなければ、プログラムできません。下記条件のエンコーダ、タイヤとします。

項目	内容
エンコーダの1回転のパルス数	16パルス / 回転
タイヤの直径(実寸)	21mm

プーリーの穴は8個ですが、プログラムで2倍のカウントにすることができます。後述します。

(1) タイヤが1回転したときのパルス数の計算

タイヤの直径から、円周が分かります。

$$\text{円周} = 2 \times r = 21 \times 3.14 = 65.94\text{mm}$$

エンコーダは 16 パルス / 回転なので、

65.94mm 進むと 16 パルス

となります。

(2) 1m 進んだときのパルス数の計算

パルス数は、距離と比例します。(1)より、1m 進んだときのパルス数は、

$$16 \text{ パルス} : 65.94\text{mm} = x \text{ パルス} : 1000\text{mm}$$

$$x = 242.6 \text{ パルス}$$

1m (1000mm)進むと、242.6 パルス

となります。

(3) 秒速 1m で進んだとき 1 秒間のパルス数の計算

(2)より、

1m/s の速さで進んだとき、1 秒間のパルス数は 242.6 パルス

となります。

(4) 秒速 1m で進んだとき、10ms 間のパルス数の計算

(3)より、

1 秒 : 242.6 パルス = 0.01 秒 : x パルス

x = 2.43 パルス

1m/s の速さで進んだとき、10ms 間のパルス数は 2.43 パルス

となります。

(5) プログラムで速度を検出する

(4)より、1m/s で進んだとき、10ms 間のパルス数は 2.43 パルスです。

現在の速度は、下記で求めることができます。

現在の速度 [m/s] = 10ms 間のパルス数 ÷ 2.43

…A

「10ms 間のパルス数」部分が、マイコンカーの実際の走行スピードにより変化します。

例えば、秒速 2m/s 以上ならモータの PWM を 0%、それ以下なら PWM を 70%にするなら、下記のようになります。

```
if( 現在の速度 >= 2m/s ) {  
    PWM を 0%にする  
} else {  
    PWM を 70%にする  
}
```

プログラムで記述します。「10ms 間のパルス数」は、変数「iEncoder」とします。

Aより

10ms 間のパルス数 = 現在の速度 × 2.43

iEncoder = 2 × 2.43

iEncoder = 4.86

変数は、整数しか扱えないので四捨五入します。

iEncoder = 5

プログラムは、下記のようになります。

```

if( iEncoder >= 5 ) {
    speed( 0, 0 );
} else {
    speed( 70, 70 );
}

```

変数 iEncoder について

10ms 間ごとに iEncoder 変数を更新する作業は、プログラムで行います。更新する間隔が短いほど最新のスピードが分かりますが、パルス数が少なくなるため精度が悪くなります。更新する間隔が長いほど精度が良くなりますが、最新の速度が分かりません。10ms ごとにカウントするのが、経験上良いかと思います。

(6) プログラムで距離を検出する

(2)より、1m 進んだときのパルス数は、242.6 パルスです。
進んだ距離は、下記で求めることができます。

$$\text{進んだ距離 [m]} = \text{合計パルス数} \div 242.6 \quad \dots \text{B}$$

「合計パルス数」部分が、マイコンカーの進んだ距離によって変化します。

例えば、10m 進んだならモータの PWM を 0%、それ以下なら PWM を 100%にするなら、下記のようになります。

```

if( 進んだ距離 >= 10m ) {
    PWM を 0%にする
} else {
    PWM を 100%にする
}

```

プログラムで記述します。「進んだ距離」は、変数「lEncoderTotal」とします。

Bより

合計パルス数 = 進んだ距離 × 242.6

lEncoderTotal = 10 × 242.6

lEncoderTotal = 2426

プログラムは、下記のようになります。

```

if( lEncoderTotal >= 2426 ) {
    speed( 0, 0 );
} else {
    speed( 100, 100 );
}

```

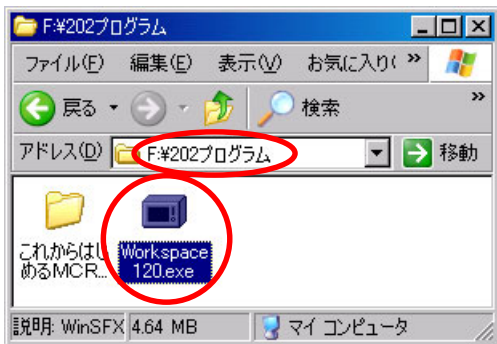
3. サンプルプログラム

3.1 ルネサス統合開発環境

サンプルプログラムは、ルネサス統合開発環境 (High-performance Embedded Workshop) を使用して開発するように作っています。ルネサス統合開発環境についてのインストール、開発方法は、「ルネサス統合開発環境操作マニュアル」を参照してください。

3.2 サンプルプログラムのインストール

サンプルプログラムをインストールします。



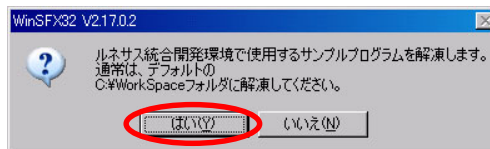
1. 講習会 CD の「CD ドライブ 202 プログラム」フォルダにある、Workspace120.exe を実行します。数字の120は、バージョン 1.20 のことです。バージョンにより数字は異なります。
2. または、マイコンカーラリーサイト「<http://www.mcr.gr.jp/>」の技術情報 ダウンロード内のページへ行きます。

マイコンカーキットプログラム、開発環境の資料

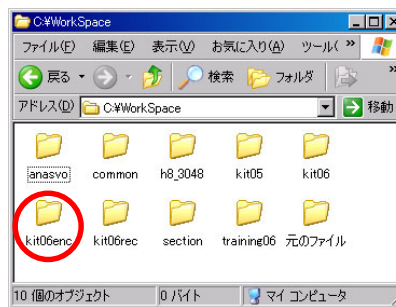
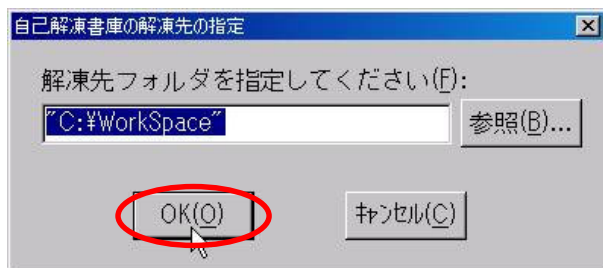
●ルネサス統合開発環境 操作マニュアル 第1.02版 2006.07.13 **NEW!**
ルネサス統合開発環境のダウンロード方法、インストール方法、操作方法、及び設定方法が載っています。ルネサス統合開発環境は、ルネサステクノロジサイトよりダウンロードして下さい(マニュアル内に方法が記載されています)。
→[DOWNLOAD](#) (PDF 約19.3MB)

●ルネサス統合開発環境用その他ソフト Ver1.00 2006.07.06 **NEW!**
ルネサス統合開発環境以外で使用するソフトをインストールします。
→[DOWNLOAD](#) (EXE 約0.3MB)

●ルネサス統合開発環境用マイコンカー関連プログラム Ver1.00 2006.07.06 **NEW!**
ルネサス統合開発環境用のマイコンカー関係プログラムです。自己解凍方式で、実行すると自動でプログラムがインストールされます。
→[DOWNLOAD](#) (EXE 約3.68MB)



3. 「ルネサス統合開発環境用マイコンカー関連プログラム」をダウンロードします。
4. CD またはダウンロードした「Workspace120.exe」を実行します。「はい」をクリックします。

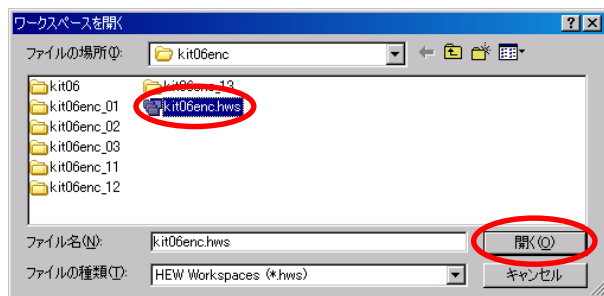
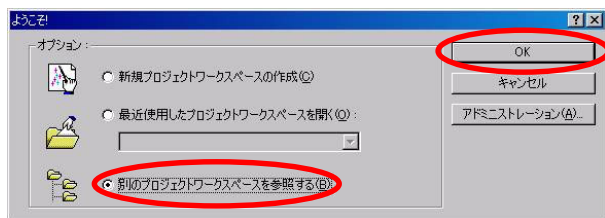


5. ファイルの解凍先を選択します。フォルダの変更はできません。OKをクリックします。
6. 解凍が終わったら、エクスプローラで「C ドライブ WorkSpace」フォルダを開いてみてください。複数のフォルダがあります。今回使用するのは、「kit06enc」です。

3.3 ワークスペース「kit06enc」を開く

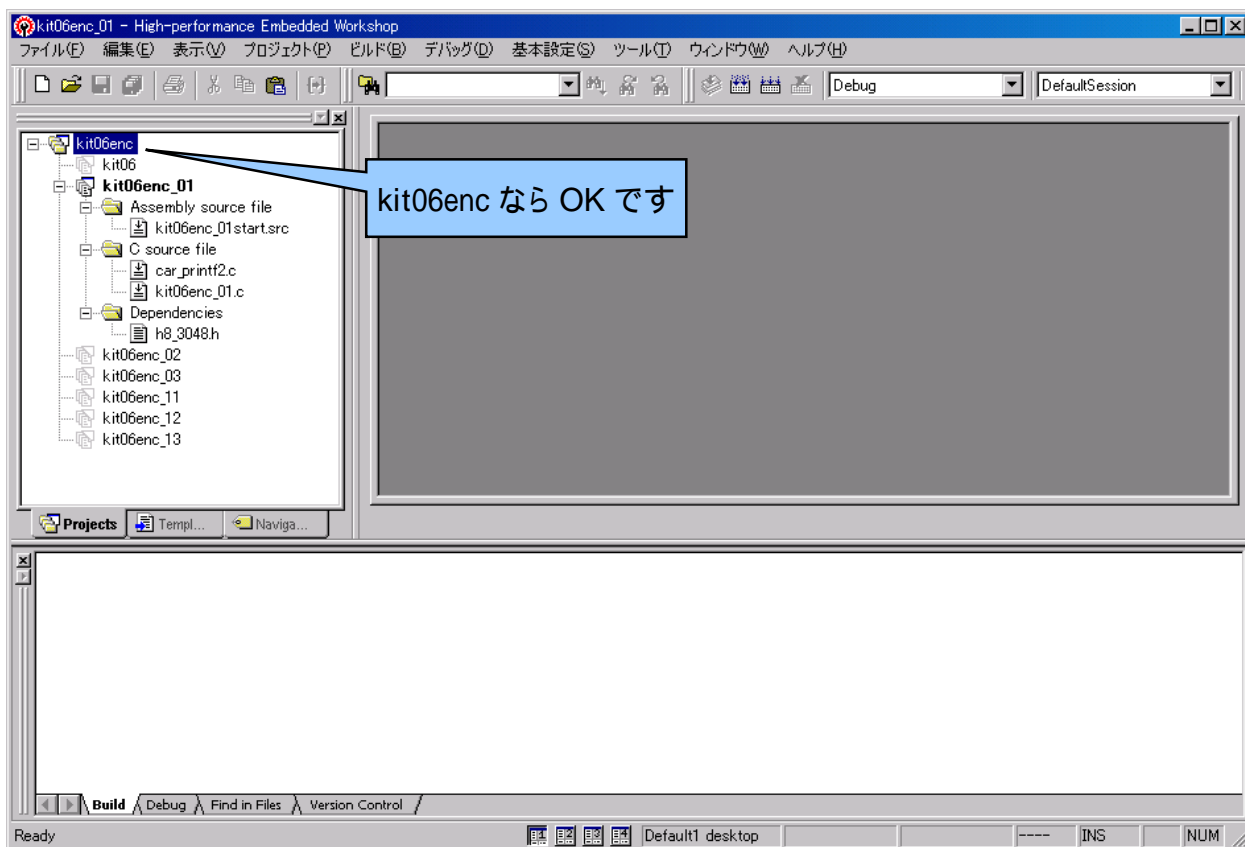


1. ルネサス統合開発環境を実行します。



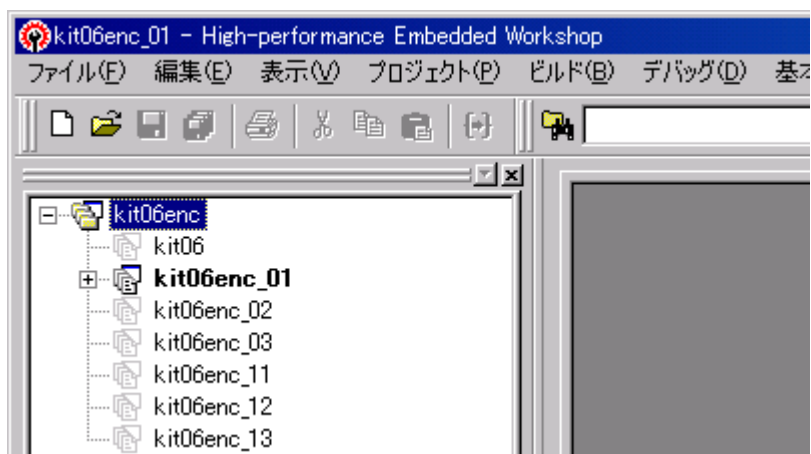
2. 「別のプロジェクトワークスペースを参照する」を選択し、**OK**をクリックします。

3. Cドライブ Workspace kit06enc の「kit06enc.hws」を選択、**開く**をクリックします。



4. kit06enc というワークスペースが開かれます。

3.4 プロジェクト



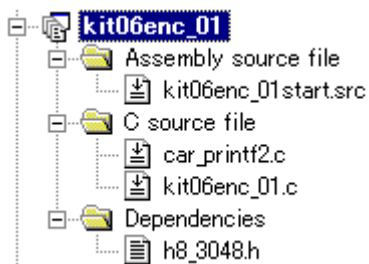
ワークスペース「kit06enc」には、7つのプロジェクトが登録されています。

プロジェクト名	内容
kit06	標準のマイコンカー走行プログラムです。このプログラムが基本となります。 (エンコーダのプログラムは入っていません)
kit06enc_01	標準走行プログラム「kit06.c」を改造して、エンコーダのパルスをカウントできるようにします。このプログラムは、エンコーダのパルスカウントができるように改造しただけで、マイコンカーのスピード制御は行っていません。プログラムの説明用です。
kit06enc_02	速度の調整を行うプログラムです。スピードが速ければマイコンカーを減速させる、遅ければ加速させるなどの制御を行うことができます。
kit06enc_03	距離の検出を行うプログラムです。例えば、クロスライン検出後、2本目の横線を読み飛ばすために、10cm進ませなさい、1周で止めるよう50mで止めなさい、などの制御を行うことができます。
kit06enc_11	kit06enc_01.cのプログラムを、1回転16パルスの自作エンコーダを使用したプログラムに改造しました。
kit06enc_12	kit06enc_02.cのプログラムを、1回転16パルスの自作エンコーダを使用したプログラムに改造しました。
kit06enc_13	kit06enc_03.cのプログラムを、1回転16パルスの自作エンコーダを使用したプログラムに改造しました。

4. プロジェクト「kit06enc_01」 kit06.c をエンコーダが使用できるように改造

標準走行プログラム「kit06.c」を改造して、エンコーダのパルスをカウントできるようにします。

4.1 プロジェクトの構成



·kit06enc_01start.src
 ·kit06enc_01.c
 ·car_printf2.c
 の3ファイルあります。
 h8_3048.h は kit06enc_01.c、car_printf2.c でインクルードされているファイルです。

4.2 プログラム

プログラムのゴシック体部分が追加、変更した部分です。

```

1 :  /*****
2 :  /* エンコーダ搭載マイコンカートレース基本プログラム「kit06enc_01.c」 */
3 :  /*                               2006.04 ジャパンマイコンカーラリー実行委員会 */
4 :  /*****
5 :  /*
6 :  本プログラムはkit06.cをベースにエンコーダを搭載したプログラムです。
7 :
8 :  kit06enc_01.cは、エンコーダが使用できるように改造しただけで、
9 :  エンコーダを使用したマイコンカーの制御はこのプログラムでは行っていません。
10 :  説明用のプログラムです。
11 :
12 :  */
13 :
14 :  /*=====*/
15 :  /* インクルード */
16 :  /*=====*/
17 :  #include <machine.h>
18 :  #include "h8_3048.h"
19 :
20 :  /*=====*/
21 :  /* シンボル定義 */
22 :  /*=====*/
23 :
24 :  /* 定数設定 */
25 :  #define TIMER_CYCLE 3071 /* タイマのサイクル 1ms */
26 :                          /* /8で使用する場合、 */
27 :                          /* /8 = 325.5[ns] */
28 :                          /* TIMER_CYCLE = */
29 :                          /* 1[ms] / 325.5[ns] */
30 :                          /* = 3072 */
31 :  #define PWM_CYCLE 49151 /* PWMのサイクル 16ms */
32 :                          /* PWM_CYCLE = */
33 :                          /* 16[ms] / 325.5[ns] */
34 :                          /* = 49152 */
35 :  #define SERVO_CENTER 5000 /* サーボのセンタ値 */
36 :  #define HANDLE_STEP 26 /* 1°分の値 */
37 :
38 :  /* マスク値設定 x : マスクあり(無効) : マスク無し(有効) */
39 :  #define MASK2_2 0x66 /* x x x x */
40 :  #define MASK2_0 0x60 /* x x x x x */
41 :  #define MASK0_2 0x06 /* x x x x x */
42 :  #define MASK3_3 0xe7 /* x x x x x */
43 :  #define MASK0_3 0x07 /* x x x x x */
44 :  #define MASK3_0 0xe0 /* x x x x x */
45 :  #define MASK4_0 0xf0 /* x x x x x */
46 :  #define MASK0_4 0x0f /* x x x x x */
47 :  #define MASK4_4 0xff /* x x x x x */
48 :
49 :  /*=====*/
50 :  /* プロトタイプ宣言 */
51 :  /*=====*/
52 :  void init( void );
    
```

自分のマイコンカーに合わせて設定します。

ロータリエンコーダ 実習マニュアル kit06 版

```

53 : void timer( unsigned long timer_set );
54 : int check_crossline( void );
55 : int check_rightline( void );
56 : int check_leftline( void );
57 : unsigned char sensor_inp( unsigned char mask );
58 : unsigned char dipsw_get( void );
59 : unsigned char pushsw_get( void );
60 : unsigned char startbar_get( void );
61 : void led_out( unsigned char led );
62 : void speed( int accele_l, int accele_r );
63 : void handle( int angle );
64 : char unsigned bit_change( char unsigned in );
65 :
66 : /*=====*/
67 : /* グローバル変数の宣言 */
68 : /*=====*/
69 : unsigned long cnt0; /* timer関数用 */
70 : unsigned long cnt1; /* main内で使用 */
71 : int pattern; /* パターン番号 */
72 :
73 : /* エンコーダ関連 */
74 : int iTimer10; /* エンコーダ取得間隔 */
75 : long iEncoderTotal; /* 積算値 */
76 : int iEncoderMax; /* 現在最大値 */
77 : int iEncoder; /* 現在値 */
78 : unsigned int uEncoderBuff; /* 前回値保存 */
79 :
80 : /*=====*/
81 : /* メインプログラム */
82 : /*=====*/
83 : void main( void )
84 : {
85 :     int i;
86 :
87 :     /* マイコン機能の初期化 */
88 :     init(); /* 初期化 */
89 :     set_ccr( 0x00 ); /* 全体割り込み許可 */
90 :
91 :     /* マイコンカーの状態初期化 */
92 :     handle( 0 );
93 :     speed( 0, 0 );
94 :
95 :     while( 1 ) {
96 :         switch( pattern ) {
97 :
98 :             /*=====*/
99 :             パターンについて
100 :             0 : スイッチ入力待ち
101 :             1 : スタートバーが開いたかチェック
102 :             11 : 通常トレース
103 :             12 : 右へ大曲げの終わりのチェック
104 :             13 : 左へ大曲げの終わりのチェック
105 :             21 : 1本目のクロスライン検出ときの処理
106 :             22 : 2本目を読み飛ばす
107 :             23 : クロスライン後のトレース、クランク検出
108 :             31 : 左クランククリア処理 安定するまで少し待つ
109 :             32 : 左クランククリア処理 曲げ終わりのチェック
110 :             41 : 右クランククリア処理 安定するまで少し待つ
111 :             42 : 右クランククリア処理 曲げ終わりのチェック
112 :             51 : 1本目の右ハーフライン検出ときの処理
113 :             52 : 2本目を読み飛ばす
114 :             53 : 右ハーフライン後のトレース
115 :             54 : 右レーンチェンジ終了のチェック
116 :             61 : 1本目の左ハーフライン検出ときの処理
117 :             62 : 2本目を読み飛ばす
118 :             63 : 左ハーフライン後のトレース
119 :             64 : 左レーンチェンジ終了のチェック
120 :             /*=====*/
121 :
122 :             case 0:
123 :                 /* スイッチ入力待ち */
124 :                 P4DR = ~iEncoder; /* エンコーダ値出力 */
125 :
126 :                 if( pushsw_get() ) {
127 :                     pattern = 1;
128 :                     cnt1 = 0;
129 :                     break;
130 :                 }
131 :                 if( cnt1 < 100 ) { /* LED点滅処理 */
132 :                     led_out( 0x1 );
133 :                 } else if( cnt1 < 200 ) {
134 :                     led_out( 0x2 );
135 :                 } else {
136 :                     cnt1 = 0;
137 :                 }
138 :                 break;
139 :

```

中略

ロータリエンコーダ 実習マニュアル kit06 版

```

500 : /*****/
501 : /* H8/3048F-ONE 内蔵周辺機能 初期化 */
502 : /*****/
503 : void init( void )
504 : {
505 :     /* I/Oポートの入出力設定 */
506 :     P1DDR = 0xff;
507 :     P2DDR = 0xff;
508 :     P3DDR = 0xff;
509 :     P4DDR = 0xff;
510 :     P5DDR = 0xff;
511 :     P6DDR = 0xf0;          /* CPU基板上的DIP SW */
512 :     P8DDR = 0xff;
513 :     P9DDR = 0xf7;          /* 通信ポート */
514 :     PADDR = 0xf6;          /* 3:スタートパルス 0:Encoder */
515 :     PBDR = 0xc0;
516 :     PBDDR = 0xfe;          /* モータドライブ基板Vol.3 */
517 :     /* センサ基板のP7は、入力専用なので入出力設定はありません */
518 :
519 :     /* ITU0 1msごとの割り込み */
520 :     ITU0_TCR = 0x23;
521 :     ITU0_GRA = TIMER_CYCLE;
522 :     ITU0_IER = 0x01;
523 :
524 :     /* ITU2 パルス入力の設定 */
525 :     ITU2_TCR = 0x04;          /* PA0端子のパルスでカウント*/
526 :
527 :     /* ITU3,4 リセット同期PWMモード 左右モータ、サーボ用 */
528 :     ITU3_TCR = 0x23;
529 :     ITU_FCR = 0x3e;
530 :     ITU3_GRA = PWM_CYCLE;          /* 周期の設定 */
531 :     ITU3_GRB = ITU3_BRB = 0;          /* 左モータのPWM設定 */
532 :     ITU4_GRA = ITU4_BRA = 0;          /* 右モータのPWM設定 */
533 :     ITU4_GRB = ITU4_BRB = SERVO_CENTER; /* サーボのPWM設定 */
534 :     ITU_TOER = 0x38;
535 :
536 :     /* ITUのカウンタスタート */
537 :     ITU_STR = 0x0d;
538 : }
539 :
540 : /*****/
541 : /* ITU0 割り込み処理 */
542 : /*****/
543 : #pragma interrupt( interrupt_timer0 )
544 : void interrupt_timer0( void )
545 : {
546 :     unsigned int i;
547 :
548 :     ITU0_TSR &= 0xfe;          /* フラグクリア */
549 :     cnt0++;
550 :     cnt1++;
551 :
552 :     /* エンコーダ関連 */
553 :     iTimer10++;
554 :     if( iTimer10 >= 10 ) {
555 :         iTimer10 = 0;
556 :         i = ITU2_CNT;
557 :         iEncoder = i - uEncoderBuff;
558 :         iEncoderTotal += iEncoder;
559 :         if( iEncoder > iEncoderMax )
560 :             iEncoderMax = iEncoder;
561 :         uEncoderBuff = i;
562 :     }
563 : }

```

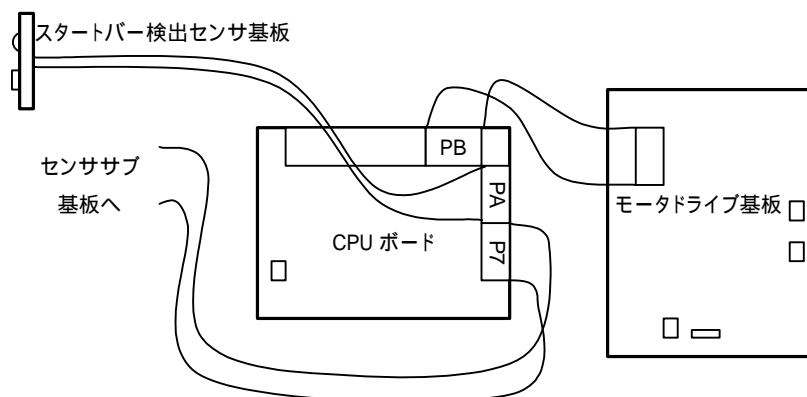
以下、略

4.3 ロータリエンコーダの接続

4.3.1 標準キット kit06 の接続の確認

kit06.c のポートと各基板の接続は、下記のようになっています。

- ・ポート7...すべてのビットをセンサ入力として使用しています。
- ・ポートB...モータドライブ基板と接続しています。
- ・ポートA...スタートバー検出センサ基板が接続されています。



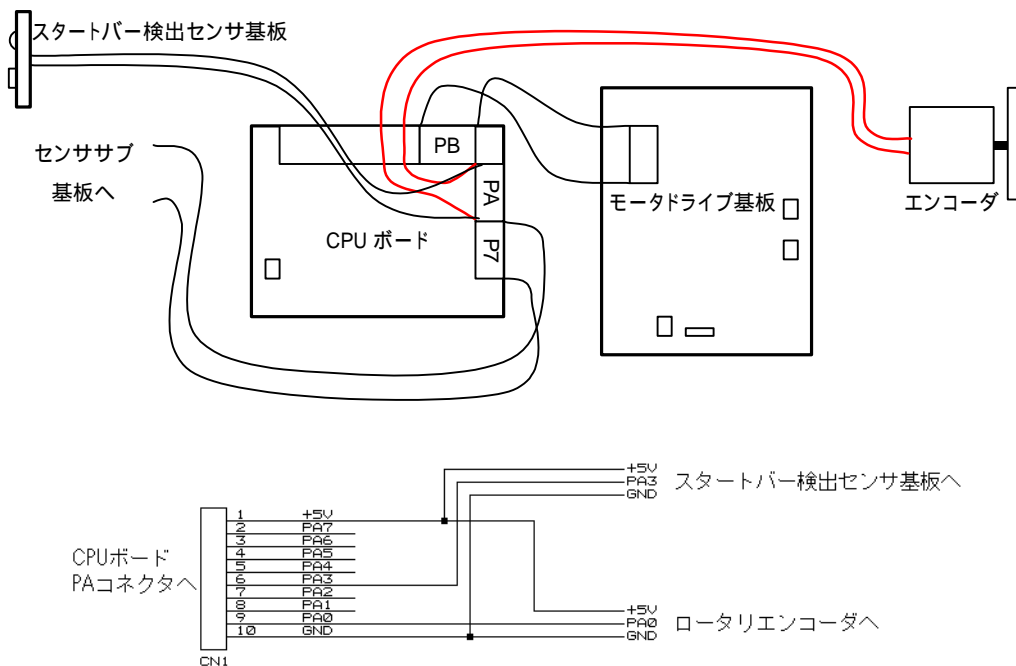
ポートAの各ビットの接続は、下記のようになっています。

ピン番号	信号名	接続先	CPU から見た方向
1	+5V	+5V	
2	PA7		出力
3	PA6		出力
4	PA5		出力
5	PA4		出力
6	PA3	スタートバー検出センサ基板	入力
7	PA2		出力
8	PA1		出力
9	PA0		出力
10	GND	GND	

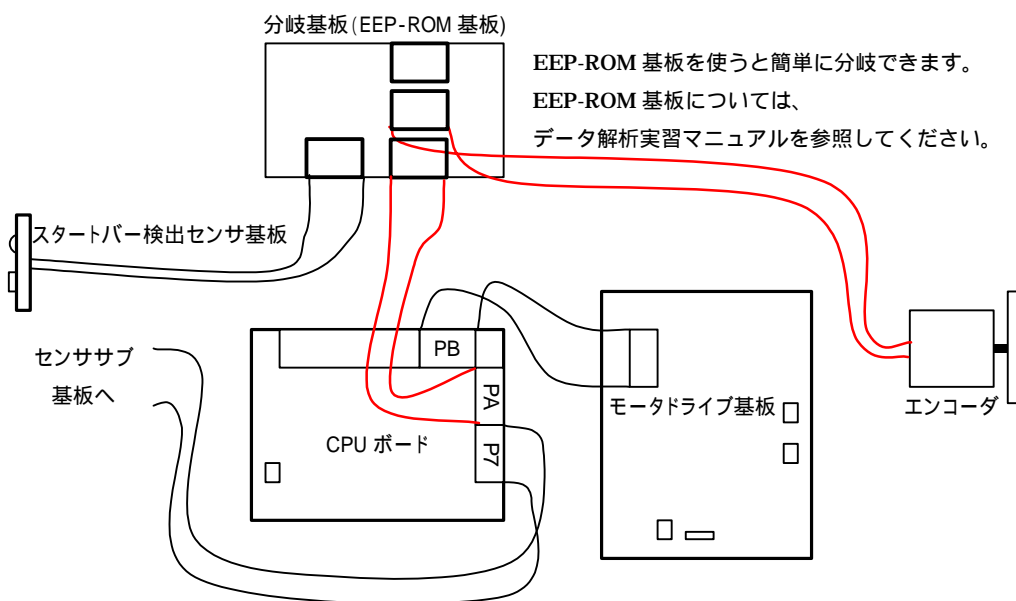
4.3.2 ロータリエンコーダを接続

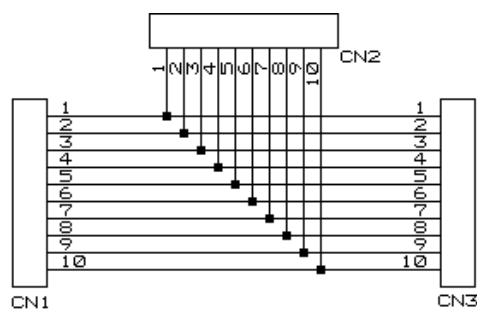
ポートAの各ビットの接続を見てみると、使用しているのはbit3のみです。そこでポートAのbit0にエンコーダを接続します。エンコーダ側のコネクタが10ピンコネクタの場合、スタートバー検出センサ基板をCPUボードに接続できなくなりますので、分岐ケーブルや分岐基板を作り、対応してください。

分岐ケーブルを作ったときの結線例を下図に示します。



分岐基板を作ったときの結線例を下図に示します。





ポート A の接続は下記ようになります。

ピン番号	信号名	接続先	CPU から見た方向
1	+5V	+5V	
2	PA7		出力
3	PA6		出力
4	PA5		出力
5	PA4		出力
6	PA3	スタートパ-検出センサ基板	入力
7	PA2		出力
8	PA1		出力
9	PA0	ロータリエンコーダ	入力
10	GND	GND	

ポート A の入出力設定は、

ビット	7	6	5	4	3	2	1	0
ポート A の入出力設定	出力	出力	出力	出力	入力	出力	出力	入力

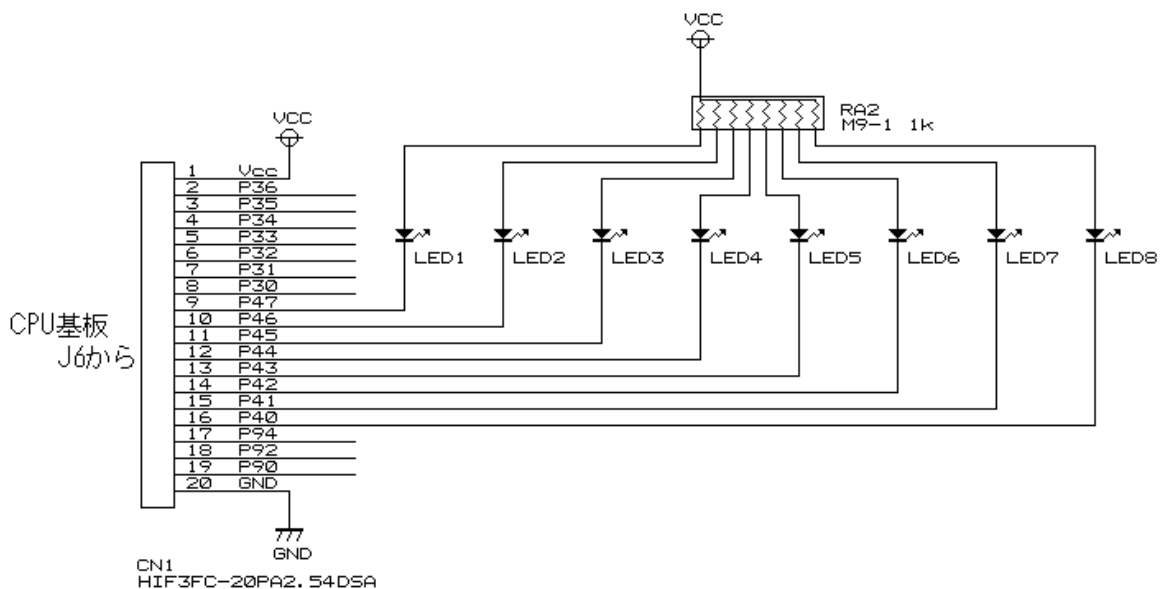
PADDR への設定値は、出力"1"、入力"0"にすれば良いだけです。

ビット	7	6	5	4	3	2	1	0
ポート A の入出力設定	1	1	1	1	0	1	1	0

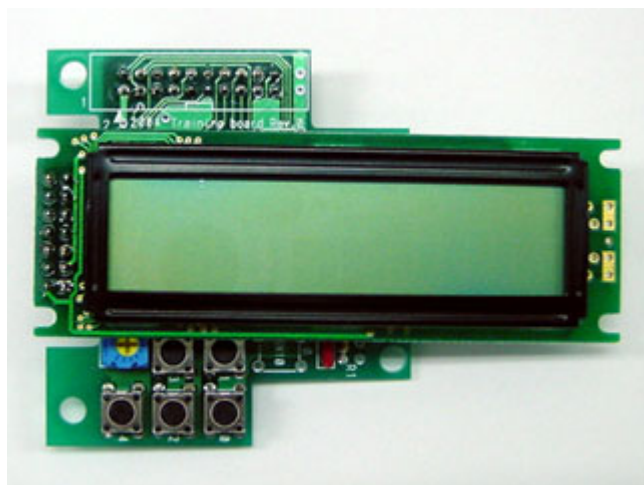
16進数に直すと、1111 0110 0xf6 となります。

4.3.3 確認用 LED の接続

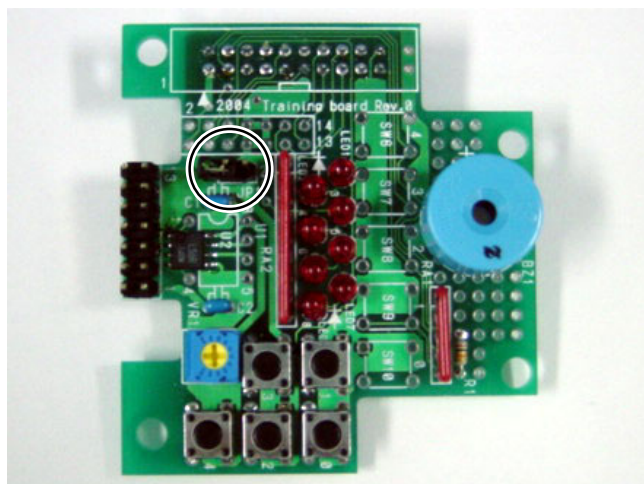
CPU ボードの J6 コネクタ (20 ピンコネクタ) が空いているので、確認用の LED を接続します。ポート 4 が 8 ビット分あるので、ポート 4 に LED を 8 個接続します。



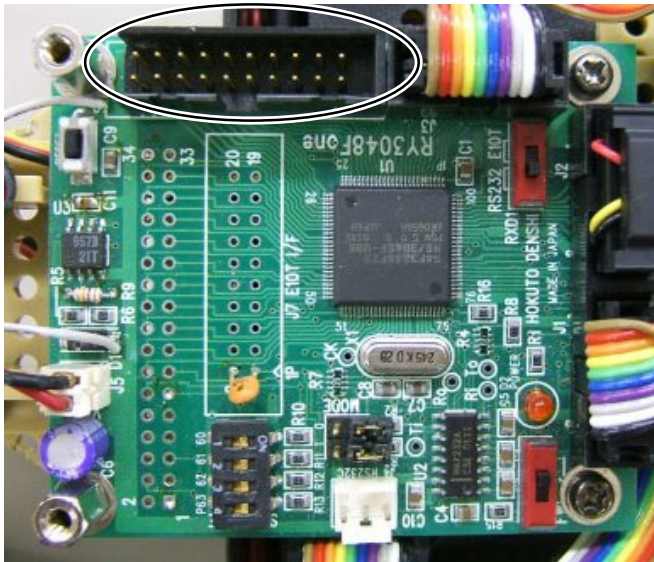
トレーニングボードには、ポート 4 に LED が 8 個接続されています。トレーニングボードがある場合は、これを接続します。



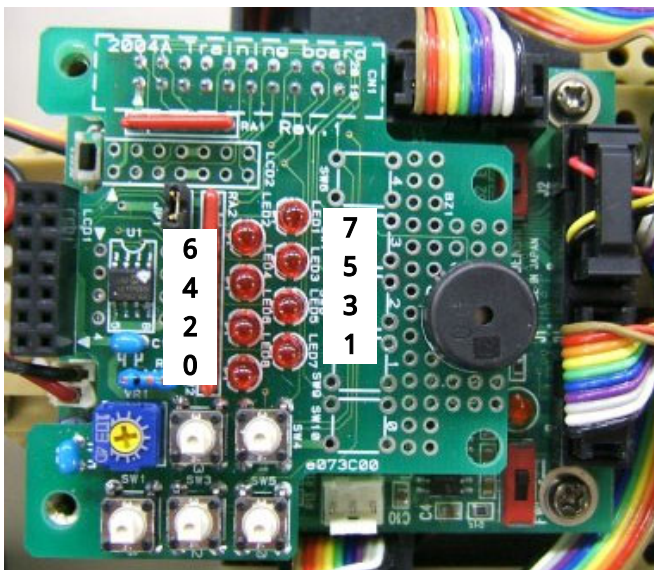
トレーニングボードです。



液晶を外します。LED が 8 個あります。部分のジャンパはショートしておきます。



CPU ボードは、20 ピンコネクタが付いていないので、あらかじめ実装しておきます。



トレーニングボードをCPUボードに接続します。これで、確認用LEDが取り付けられました。

ちなみにトレーニングボードのLEDは、左の写真のようにビット配列されています。

4.4 プログラムの解説

4.4.1 エンコーダ関連の変数の宣言

```

73 : /* エンコーダ関連 */
74 : int          iTimer10;          /* エンコーダ取得間隔      */
75 : long         lEncoderTotal;     /* 積算値                  */
76 : int          iEncoderMax;       /* 現在最大値              */
77 : int          iEncoder;          /* 現在値                  */
78 : unsigned int uEncoderBuff;     /* 前回値保存              */

```

ロータリエンコーダを使用するに当たって、新たに変数を宣言しています。

変数名	意味	内容
iTimer10	10ms タイマ	エンコーダ値の更新は、interrupt_timer0 関数内で行います。interrupt_timer0 関数は 1ms ごとに実行されますが、エンコーダ処理は 10ms ごとです。そこで、この変数を 1ms ごとに + 1 して 10 になったかどうかチェックしています。
lEncoderTotal	エンコーダ積算値	スタートしてからのエンコーダパルスの積算値を保存しています。long 型変数ですので、21 億回までカウントできます。
iEncoderMax	10ms ごとの最大値	10ms ごとに更新されるエンコーダ値の最大値を保存しています。走行後、この値をチェックすれば最速値が分かります。
iEncoder	10ms ごとの現在値	10ms ごとに更新されるエンコーダ値の現在値を保存しています。この値をチェックすれば、現在のスピードが分かります。
uEncoderBuff	前回値保存用バッファ	ITU2_CNT の前回の値を保存しています。main 関数では使用しません。

これらの変数は、割り込みプログラム内で、10ms ごとに更新されます。詳しくは割り込みで説明します。ちなみに、これらの変数は初期値のないグローバル変数なので、初期値 0 です。

4.4.2 パターン 0:エンコーダ値をポート 4 へ出力

```

122 :     case 0:
123 :         /* スイッチ入力待ち */
124 :         P4DR = ~iEncoder;          /* エンコーダ値出力      */
125 :
126 :         if( pushsw_get() ) {
127 :             pattern = 1;
128 :             cnt1 = 0;
129 :             break;
130 :         }
131 :         if( cnt1 < 100 ) {          /* LED 点滅処理          */
132 :             led_out( 0x1 );
133 :         } else if( cnt1 < 200 ) {
134 :             led_out( 0x2 );
135 :         } else {
136 :             cnt1 = 0;
137 :         }
138 :         break;

```

「iEncoder」変数は、10ms 間のエンコーダパルス値を格納する変数です。ポート 4 には、10ms 間に数えたパルス数が出力されます。10ms ごとに最新の値に更新されます。

トレーニングボードの LED は、「0」で点灯、「1」で消灯のため、「~ (チルダ)」で反転させています。

4.4.3 入出力設定の変更

```

500 : /*****/
501 : /* H8/3048F-ONE 内蔵周辺機能 初期化 */
502 : /*****/
503 : void init( void )
504 : {
505 :     /* I/O ポートの入出力設定 */
506 :     P1DDR = 0xff;
507 :     P2DDR = 0xff;
508 :     P3DDR = 0xff;
509 :     P4DDR = 0xff;
510 :     P5DDR = 0xff;
511 :     P6DDR = 0xf0;          /* CPU 基板上的 DIP SW */
512 :     P8DDR = 0xff;
513 :     P9DDR = 0xf7;        /* 通信ポート */
514 :     PADDR = 0xf6;        /* 3:スタートセンサ 0:Encoder */
515 :     PBDR = 0xc0;
516 :     PBDDR = 0xfe;        /* モータドライブ基板 Vol.3 */
517 :     /* センサ基板の P7 は、入力専用なので入出力設定はありません */

```

ポート A の bit0 は、ロータリエンコーダ入力になったので、0xf7 から 0xf6 へ変更します。

4.4.4 外部パルス入力設定

```

524 :     /* ITU2 パルス入力の設定 */
525 :     ITU2_TCR = 0x04;          /* PA0 端子のパルスでカウント*/
中略
536 :     /* ITU のカウントスタート */
537 :     ITU_STR = 0x0d;

```

ITU2 を外部パルス入力用として、ロータリエンコーダのパルスをカウントします。

レジスタの設定について説明します。

ITU2_TCR(タイマコントロールレジスタ)の設定内容

ビット:	7	6	5	4	3	2	1	0
ITU2_TCR:	-	CCLR1	CCLR0	CKEG1	CKEG0	TPSC2	TPSC1	TPSC0
設定値:	0	0	0	0	0	1	0	0
16進数:	0				4			

・ビット 6,5:カウンタクリア 1,0

CNT のカウンタクリア要因を選択します。

CCLR1	CCLR0	説明
0	0	CNT のクリア禁止
0	1	GRA のコンペアマッチ / インพุットキャプチャで CNT をクリア
1	0	GRB のコンペアマッチ / インพุットキャプチャで CNT をクリア
1	1	同期クリア

今までは、「GRAのコンペアマッチ / インพุットキャプチャでCNTをクリア」の設定でしたが、今回は ITU2_CNT をクリアする必要はないので、クリアしません。

・ビット 4,3:クロックエッジ 1,0

外部クロック選択時に、外部クロックの入力エッジを選択します。

CKEG1	CKEG0	説明
0	0	立ち上がりエッジでカウント
0	1	立ち下がりエッジでカウント
1	0	立ち上がり / 立ち下がり両エッジでカウント
1	1	立ち上がり / 立ち下がり両エッジでカウント

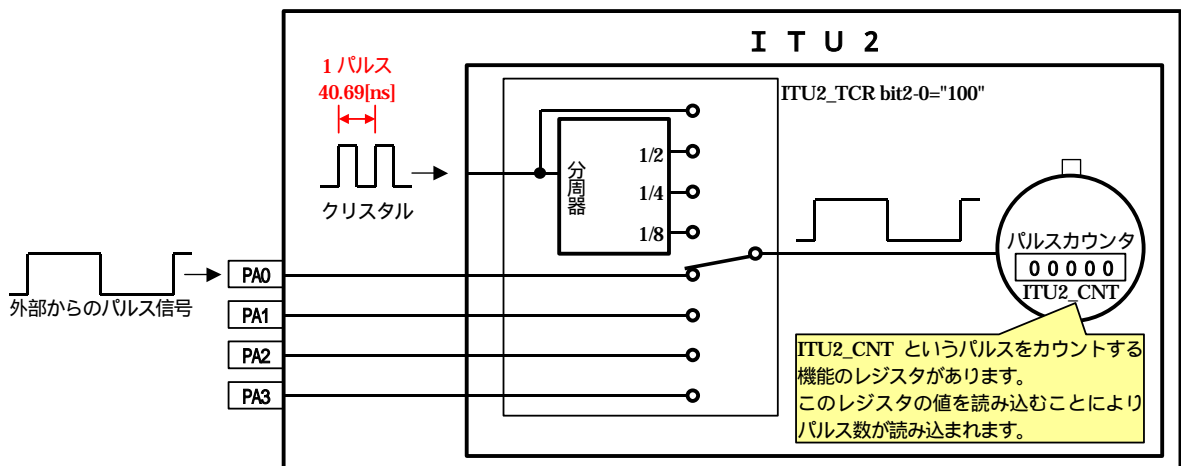
外部パルスの立ち上がりで ITU2_CNT が + 1 します。

・ビット 2~0:タイマプリスケラ 2~0

CNT のカウントクロックを選択します。

TPSC2	TPSC1	TPSC0	説明
0	0	0	内部クロック: でカウント
0	0	1	内部クロック: / 2 でカウント
0	1	0	内部クロック: / 4 でカウント
0	1	1	内部クロック: / 8 でカウント
1	0	0	外部クロック A: TCLKA 端子 (PA0) でカウント
1	0	1	外部クロック B: TCLKB 端子 (PA1) でカウント
1	1	0	外部クロック C: TCLKC 端子 (PA2) でカウント
1	1	1	外部クロック D: TCLKD 端子 (PA3) でカウント

イメージとしては下図のようになります。



ITU2_CNT は、エンコーダから出力されるパルス数をカウントします。入力端子は、ポート A の bit0 ~ 3 のどれかを選ぶことができます。今回は、PA0 に接続します。

外部パルスを実カウントする場合、ポート A の bit0 ~ 3 の端子以外でカウントすることはできません。

ITU_STR(タイマスタートレジスタ)の設定内容

ビット:	7	6	5	4	3	2	1	0
ITU_STR:	-	-	-	STR4	STR3	STR2	STR1	STR0
設定値:	0	0	0	0	1	1	0	1
16進数:	0				d			

・ビット 4-0: カウンタスタート 4 ~ 0

タイマカウンタ x の動作 / 停止を選択します。

STR4-0	説明
0	ITUx の CNT のカウント動作は停止
1	ITUx の CNT はカウント動作

x は 0 ~ 4

ITU は下記のように使用します。

・ITU0...1ms 割り込み ・ITU1...未使用 ・ITU2...パルスカウンタ ・ITU3...リセット同期 PWM モード
設定値は 0x0d となります。ここで、ITU2 のみカウントする設定にしてしまうと、ITU2 以外は動作しなくなってしまうので注意が必要です。

4.4.5 パルスカウントを 2 倍にする方法

因みに、525 行の ITU2_TCR の値を 0x14 (bit4="1") にすると、立ち上がり、立ち下がりの両エッジでカウントする設定となり、2 倍でカウントすることができます。エンコーダを自作して、パルス数の少ない場合は有効です。詳しくは、「7.1 プーリーを使用したときの回転数計算」を参照してください。

```
524 : /* ITU2 パルス入力の設定 */
525 : ITU2_TCR = 0x14; /* PA0 端子のパルスでカウント*/
```

4.4.6 ITU0 割り込み処理

```

543 : #pragma interrupt( interrupt_timer0 )
544 : void interrupt_timer0( void )
545 : {
546 :     unsigned int i;
547 :
548 :     ITU0_TSR &= 0xfe;          /* フラグクリア          */
549 :     cnt0++;
550 :     cnt1++;
551 :
552 :     /* エンコーダ関連 */
553 :     iTimer10++;
554 :     if( iTimer10 >= 10 ) {
555 :         iTimer10 = 0;
556 :         i = ITU2_CNT;
557 :         iEncoder      = i - uEncoderBuff;
558 :         lEncoderTotal += iEncoder;
559 :         if( iEncoder > iEncoderMax )
560 :             iEncoderMax = iEncoder;
561 :         uEncoderBuff = i;
562 :     }
563 : }

```

553 行...iTimer10 変数を増加させます。

554 行...iTimer10 変数が 10 以上なら次の行を実行します。ITU0 割り込みは、1ms ごとに実行されますが、エンコーダ関連処理は 10ms ごとに処理します。そのため、回数を数えて 10 回以上なら次の行に移りエンコーダ処理を、それ以下なら 562 行へ移りエンコーダ処理をしません。

555 行...iTimer10 変数を 0 にして、次回 10 回目かどうかチェックするのに備えます。

556 行...現在のカウンタ値 ITU2_CNT を変数 i に代入します。なぜ、ITU2_CNT の値を直接使わないのでしょうか。ITU2_CNT の値は、エンコーダからのパルスが入力されるたびに増加していきます。プログラムが 1 行進むと違う値になっているかもしれません。そのため、いったん別な変数に代入して、この値をプログラムでは最新値として使います。

557 行...最新の 10ms 間のエンコーダ値を計算しています。計算は、

エンコーダ値 = $i - uEncoderBuff$

としています。i は、現在のカウンタ値、uEncoderBuff のカウンタ値です。言い換えれば、

エンコーダ値 = 現在のカウンタ値 - 1 回前のカウンタ値

となります。ITU2_CNT は 16 ビット幅の符号無し int 型の大きさなので、0 ~ 65,535 までカウントされます。65,535 の次は 0 に戻ってカウントを続けます。そのため、前の値を覚えておき、現在の値を引くことにより前回と今回の差分が得られます。これが 10ms 間のパルス数です。

図解すると下記のようなイメージです。



i - uEncoderBuff の値が、最新の 10ms 間のエンコーダパルス値となる

558 行...エンコーダの積算値を計算しています。計算は、

積算値 = 積算値 + 最新の 10ms 間のエンコーダ値

です。積算値は、long 型ですので、21 億までカウントできます。1m で 1000 カウントとすると、約 2,100,000m(=2,100km)まで計算できます。

559 行...iEncoder と iEncoderMax 変数を比較しています。iEncoder 変数の方の値が大きければ次の行へ進みます。

560 行...iEncoderMax 変数に、iEncoder 変数の値を代入します。iEncoderMax 変数には 10ms 間に計測したパルス数の最大値が代入されます。走行後、この変数をチェックすればマイコンカーの瞬間最大速度が分かります。

561 行...i には現在の ITU2_CNT の値が入っています。最後に uEncoderBuff 変数に i の値を代入します。今は uEncoderBuff の値は最新値を代入したことになりますが、次にエンコーダ関連処理をするのは 10ms 後なので、そのときの uEncoderBuff は 10ms 前の値となります。

4.4.7 更新する間隔について

このプログラムでは割り込み内にあるため、1ms ごとに実行されます。そこで、

```
553 :     iTimer10++;
554 :     if( iTimer10 >= 10 ) {
```

で、回数を数えて 10 回目で行、要は 10ms ごとにエンコーダ処理が行われます。

更新する間隔が短いほど最新のスピードが分かりますが、パルス数が少なくなるため精度が悪くなります。更新する間隔が長いほど精度が良くなりますが、最新の速度が分かりません。10ms ごとにカウントするのが、経験上良いかと思います。

4.4.8 ITU2_CNT が 65535 から 0 になったとき



i - uEncoderBuff の値が、最新の 10ms 間のエンコーダパルス値となる

ITU2_CNT は、符号無し 16 ビット幅です。上限は 65535 で、次が 0 に戻ります。

10ms 間のパルス値を計算するには、
(現在の ITU2_CNT) - (10ms 前の ITU2_CNT)

です。図のように、10ms 前の ITU2_CNT の値が 65530、現在の値が 0 に戻って 2 になった場合、どのようになるのでしょうか。

普通に考えると、

$$(現在の ITU2_CNT) - (10ms 前の ITU2_CNT) = 2 - 65530 = -65528$$

となり、とんでもない値になります。

16 進数に直すと、

$$0x0002 - 0xffffa = 0xffff0008$$

ただし、計算結果も符号無し 16 ビット幅なので、

$$0x0002 - 0xffffa = 0x0008$$

となり、結果は 8 になります。カウント分を数えると、65531,65532,65533,65534,65535,0,1,2 と 8 カウント分になり

計算は合います。

このように、符号無し16ビット幅で計算しているの、いったん0に戻ってもきちんと計算されます。

4.4.9 なぜ、バッファを使うのか

ITU2_CNT がエンコーダのパルスによって増えていきます。下記のようなプログラムではどうなのでしょう。

```
#pragma interrupt( interrupt_timer0 )
void interrupt_timer0( void )
{
    unsigned int i;

    ITU0_TSR &= 0xfe;          /* フラグクリア          */
    cnt0++;
    cnt1++;

    /* エンコーダ関連 */
    iTimer10++;
    if( iTimer10 >= 10 ) {
        iTimer10 = 0;
        iEncoder = ITU2_CNT;    /* カウンタの値を iEncoder にコピーして */
        ITU2_CNT = 0;          /* カウンタの値をクリア */
        中略
    }
}
```

このようにすれば、uEncoderBuffという変数を使用しないで、シンプルに計測ができます。実は、これではパルスカウントされない場合があります。iEncoder という変数にパルスを代入して、すぐに ITU2_CNT をクリアしていません。代入してから0にするまでの短い間でも、パルスが入力されてしまうことがあります。

```
if( iTimer10 >= 10 ) {
    iTimer10 = 0;
    iEncoder = ITU2_CNT;    /* カウンタの値を iEncoder にコピーして */
    ここでパルスが入力されて ITU2_CNT の値が1つ増えた
    ITU2_CNT = 0;          /* カウンタの値をクリア */
    中略
}
```

この場合、1カウント分が無効になってしまいます。たったのパルス1つ分ですが、もしITU2_CNTをクリアするたびに無効になればかなりのパルス数になってしまいます。そのため、バッファを使用した複雑なプログラムで処理しています。

5. プロジェクト「kit06enc_02」 速度の調整

急カーブになり大曲げするとき、スピードを落とします。しかし、スピードを落としすぎるとタイムロスにつながり、速すぎると脱輪します。そこで、大曲げ中の現在の速度を検出して、設定スピード以上ならブレーキ、以下なら走行させるようにします。

5.1 プログラム

プログラムのゴシック体部分が追加、変更した部分です。

前略

```

84 : void main( void )
85 : {
86 :     int    i;
87 :
88 :     /* マイコン機能の初期化 */
89 :     init();                               /* 初期化          */
90 :     set_ccr( 0x00 );                       /* 全体割り込み許可 */
91 :
92 :     /* マイコンカーの状態初期化 */
93 :     handle( 0 );
94 :     speed( 0, 0 );
95 :
96 :     while( 1 ) {
97 :
98 :         P4DR = ~pattern;
99 :
100 :        switch( pattern ) {

```

中略

```

236 :        case 12:
237 :            /* 右へ大曲げの終わりのチェック */
238 :            if( check_crossline() ) {      /* 大曲げ中もクロスラインチェック */
239 :                pattern = 21;
240 :                break;
241 :            }
242 :            if( check_rightline() ) {      /* 右ハーフラインチェック */
243 :                pattern = 51;
244 :                break;
245 :            }
246 :            if( check_leftline() ) {      /* 左ハーフラインチェック */
247 :                pattern = 61;
248 :                break;
249 :            }
250 :            if( iEncoder >= 10 ) {
251 :                speed2( 0 ,0 );
252 :            } else {
253 :                speed2( 60 ,41 );
254 :            }
255 :            if( sensor_inp(MASK3_3) == 0x06 ) {
256 :                pattern = 11;
257 :            }
258 :            break;
259 :
260 :        case 13:
261 :            /* 左へ大曲げの終わりのチェック */
262 :            if( check_crossline() ) {      /* 大曲げ中もクロスラインチェック */
263 :                pattern = 21;
264 :                break;
265 :            }
266 :            if( check_rightline() ) {      /* 右ハーフラインチェック */
267 :                pattern = 51;
268 :                break;
269 :            }
270 :            if( check_leftline() ) {      /* 左ハーフラインチェック */
271 :                pattern = 61;
272 :                break;
273 :            }
274 :            if( iEncoder >= 10 ) {
275 :                speed2( 0 ,0 );
276 :            } else {
277 :                speed2( 41 ,60 );
278 :            }
279 :            if( sensor_inp(MASK3_3) == 0x60 ) {
280 :                pattern = 11;
281 :            }
282 :            break;

```

中略

ロータリエンコーダ 実習マニュアル kit06 版

```

301 :     case 23:
302 :         /* クロスライン後のトレース、クランク検出 */
303 :         if( sensor_inp(MASK4_4)==0xf8 ) {
304 :             /* 左クランクと判断 左クランククリア処理へ */
305 :             led_out( 0x1 );
306 :             handle( -38 );
307 :             speed( 10 ,50 );
308 :             pattern = 31;
309 :             cnt1 = 0;
310 :             break;
311 :         }
312 :         if( sensor_inp(MASK4_4)==0x1f ) {
313 :             /* 右クランクと判断 右クランククリア処理へ */
314 :             led_out( 0x2 );
315 :             handle( 38 );
316 :             speed( 50 ,10 );
317 :             pattern = 41;
318 :             cnt1 = 0;
319 :             break;
320 :         }
321 :         if( iEncoder >= 10 ) {      /* クロスライン後のスピード制御 */
322 :             speed2( 0 ,0 );
323 :         } else {
324 :             speed2( 70 ,70 );
325 :         }
326 :         switch( sensor_inp(MASK3_3) ) {
327 :             case 0x00:
328 :                 /* センタ まっすぐ */
329 :                 handle( 0 );
330 :                 break;
331 :             case 0x04:
332 :             case 0x06:
333 :             case 0x07:
334 :             case 0x03:
335 :                 /* 左寄り 右曲げ */
336 :                 handle( 8 );
337 :                 break;
338 :             case 0x20:
339 :             case 0x60:
340 :             case 0xe0:
341 :             case 0xc0:
342 :                 /* 右寄り 左曲げ */
343 :                 handle( -8 );
344 :                 break;
345 :         }
346 :         break;

```

中略

```

399 :     case 53:
400 :         /* 右ハーフライン後のトレース、レーンチェンジ */
401 :         if( sensor_inp(MASK4_4) == 0x00 ) {
402 :             handle( 15 );
403 :             speed( 40 ,32 );
404 :             pattern = 54;
405 :             cnt1 = 0;
406 :             break;
407 :         }
408 :         if( iEncoder >= 10 ) {      /* ハーフライン後のスピード制御 */
409 :             speed2( 0 ,0 );
410 :         } else {
411 :             speed2( 70 ,70 );
412 :         }
413 :         switch( sensor_inp(MASK3_3) ) {
414 :             case 0x00:
415 :                 /* センタ まっすぐ */
416 :                 handle( 0 );
417 :                 break;
418 :             case 0x04:
419 :             case 0x06:
420 :             case 0x07:
421 :             case 0x03:
422 :                 /* 左寄り 右曲げ */
423 :                 handle( 8 );
424 :                 break;
425 :             case 0x20:
426 :             case 0x60:
427 :             case 0xe0:
428 :             case 0xc0:
429 :                 /* 右寄り 左曲げ */
430 :                 handle( -8 );
431 :                 break;
432 :             default:
433 :                 break;
434 :         }
435 :         break;

```

中略

```

463 :     case 63:
464 :         /* 左ハーフライン後のトレース、レーンチェンジ */
465 :         if( sensor_inp(MASK4_4) == 0x00 ) {
466 :             handle( -15 );
467 :             speed( 32 ,40 );
468 :             pattern = 64;
469 :             cnt1 = 0;
470 :             break;
471 :         }
472 :         if( iEncoder >= 10 ) { /* ハーフラインライン後のスピード制御 */
473 :             speed2( 0 ,0 );
474 :         } else {
475 :             speed2( 70 ,70 );
476 :         }
477 :         switch( sensor_inp(MASK3_3) ) {
478 :             case 0x00:
479 :                 /* センタ まっすぐ */
480 :                 handle( 0 );
481 :                 break;
482 :             case 0x04:
483 :             case 0x06:
484 :             case 0x07:
485 :             case 0x03:
486 :                 /* 左寄り 右曲げ */
487 :                 handle( 8 );
488 :                 break;
489 :             case 0x20:
490 :             case 0x60:
491 :             case 0xe0:
492 :             case 0xc0:
493 :                 /* 右寄り 左曲げ */
494 :                 handle( -8 );
495 :                 break;
496 :             default:
497 :                 break;
498 :         }
499 :         break;

```

中略

```

755 : /******
756 : /* 速度制御2 */
757 : /* 引数 左モータ:-100~100 , 右モータ:-100~100 */
758 : /* 0で停止、100で正転100%、-100で逆転100% */
759 : /* ティップスイッチは関係なし */
760 : /******
761 : void speed2( int accele_l, int accele_r )
762 : {
763 :     unsigned long speed_max;
764 :
765 :     speed_max = PWM_CYCLE - 1;
766 :
767 :     /* 左モータ */
768 :     if( accele_l >= 0 ) {
769 :         PBDR &= 0xfb;
770 :         ITU3_BRB = speed_max * accele_l / 100;
771 :     } else {
772 :         PBDR |= 0x04;
773 :         accele_l = -accele_l;
774 :         ITU3_BRB = speed_max * accele_l / 100;
775 :     }
776 :
777 :     /* 右モータ */
778 :     if( accele_r >= 0 ) {
779 :         PBDR &= 0xf7;
780 :         ITU4_BRA = speed_max * accele_r / 100;
781 :     } else {
782 :         PBDR |= 0x08;
783 :         accele_r = -accele_r;
784 :         ITU4_BRA = speed_max * accele_r / 100;
785 :     }
786 : }

```

以下、略

5.2 プログラムの解説

5.2.1 パターンの表示

```

96 :      while( 1 ) {
97 :
98 :      P4DR = ~pattern;
99 :
100 :      switch( pattern ) {

```

現在、どのパターンを実行しているか、マイコンカーをただけでは分かりません。もしかすると、自分が予期していないパターンのプログラムを実行しているかもしれません。そこで、ポート4に接続されているLEDに現在のパターンを出力して、分かるようにしています。デバッグ用です。

5.2.2 パターン 12 右大曲げときの処理

```

236 :      case 12:
237 :          /* 右へ大曲げの終わりのチェック */
238 :          if( check_crossline() ) {          /* 大曲げ中もクロスラインチェック */
239 :              pattern = 21;
240 :              break;
241 :          }
242 :          if( check_rightline() ) {          /* 右ハーフラインチェック */
243 :              pattern = 51;
244 :              break;
245 :          }
246 :          if( check_leftline() ) {          /* 左ハーフラインチェック */
247 :              pattern = 61;
248 :              break;
249 :          }
250 :          if( iEncoder >= 10 ) {
251 :              speed2( 0 ,0 );
252 :          } else {
253 :              speed2( 60 ,41 );
254 :          }
255 :          if( sensor_inp(MASK3_3) == 0x06 ) {
256 :              pattern = 11;
257 :          }
258 :          break;

```

パターン 12 はコース左に寄り、右に大曲げしているときの処理です。

ここで、現在のスピードをチェックして、設定スピード以上ならモータを左右 0%、設定スピード以下なら左 60%、右 41%にします。

前章の計算結果は、「1m/s の速さで進んだとき、10ms 間のパルス数は 9.65 パルス」でした。ここでは現在のパルス値 iEncoder が 10 以下がチェックしていますので、約 1m/s かどうかチェックしています。もし、2m/s かどうかチェックしたいときは、

$$\begin{aligned}
 10\text{ms間のパルス数} &= \text{現在の速度} \times 9.65 \\
 &= 2[\text{m/s}] \times 9.65 \\
 &= 19.30 \\
 &19 \quad \text{小数点は使えないので四捨五入}
 \end{aligned}$$

iEncoder が 19 以上かどうかチェックすると、速度が 2m/s 以上かチェックすることになります。
一般的に、下記のような関係になります。

	特徴	長所	短所
設定値が小さい場合	ブレーキを多くかける	カーブで脱輪しづらい	タイムロスが多くなる
設定値が大きい場合	ブレーキを余りかけない	タイムロスは少ない	カーブで脱輪しやすい

各自のマイコンカーに合わせて、一番きついカーブで脱輪ないように調整します。

5.2.3 speed2 関数

speed 関数を良く見ると... speed2 関数？ 2 が付いています。

```

755 : /****** /
756 : /* 速度制御2 */
757 : /* 引数 左モータ:-100~100 , 右モータ:-100~100 */
758 : /*      0で停止、100で正転100%、-100で逆転100% */
759 : /*      ディップスイッチは関係なし */
760 : /****** /
761 : void speed2( int accele_l, int accele_r )
762 : {
763 :     unsigned long  speed_max;
764 :
765 :     speed_max = PWM_CYCLE - 1;
766 :
767 :     /* 左モータ */
768 :     if( accele_l >= 0 ) {
769 :         PBDR &= 0xfb;
770 :         ITU3_BRB = speed_max * accele_l / 100;
771 :     } else {
772 :         PBDR |= 0x04;
773 :         accele_l = -accele_l;
774 :         ITU3_BRB = speed_max * accele_l / 100;
775 :     }
776 :
777 :     /* 右モータ */
778 :     if( accele_r >= 0 ) {
779 :         PBDR &= 0xf7;
780 :         ITU4_BRA = speed_max * accele_r / 100;
781 :     } else {
782 :         PBDR |= 0x08;
783 :         accele_r = -accele_r;
784 :         ITU4_BRA = speed_max * accele_r / 100;
785 :     }
786 : }
```

speed 関数は、

speed 関数の引数の割合 × ディップスイッチの割合

が実際にモータに出力される PWM 値でした。エンコーダを使えば、パルス数によってスピードを制御するのでディップスイッチでスピードを落とす必要がありません。そこで**ディップスイッチには関係なく、speed 関数の引数そのものがモータに出力される speed2 関数を作りました。**エンコーダ値を比較してスピード制御する部分には、

speed2 関数を使用します。

関数を追加したときは、忘れずにプロトタイプ宣言も追加してください。

5.2.4 パターン 13 左大曲げときの処理

```

260 :     case 13:
261 :         /* 左へ大曲げの終わりのチェック */
262 :         if( check_crossline() ) {      /* 大曲げ中もクロスラインチェック */
263 :             pattern = 21;
264 :             break;
265 :         }
266 :         if( check_rightline() ) {      /* 右ハーフラインチェック */
267 :             pattern = 51;
268 :             break;
269 :         }
270 :         if( check_leftline() ) {      /* 左ハーフラインチェック */
271 :             pattern = 61;
272 :             break;
273 :         }
274 :         if( iEncoder >= 10 ) {
275 :             speed2( 0 ,0 );
276 :         } else {
277 :             speed2( 41 ,60 );
278 :         }
279 :         if( sensor_inp(MASK3_3) == 0x60 ) {
280 :             pattern = 11;
281 :         }
282 :         break;

```

パターン 13 はコース右に寄り、左に大曲げしているときの処理です。

ここで、現在のスピードをチェックして、設定スピード以上ならモータを左右 0%、設定スピード以下なら左 41%、右 60%にします。こちらも speed2 関数を使用します。

5.2.5 パターン 23 クロスライン後のトレース、クランク検出ときの処理

```
301 :     case 23:
302 :         /* クロスライン後のトレース、クランク検出 */
303 :         if( sensor_inp(MASK4_4)==0xf8 ) {
304 :             /* 左クランクと判断 左クランククリア処理へ */
305 :             led_out( 0x1 );
306 :             handle( -38 );
307 :             speed( 10 ,50 );
308 :             pattern = 31;
309 :             cnt1 = 0;
310 :             break;
311 :         }
312 :         if( sensor_inp(MASK4_4)==0x1f ) {
313 :             /* 右クランクと判断 右クランククリア処理へ */
314 :             led_out( 0x2 );
315 :             handle( 38 );
316 :             speed( 50 ,10 );
317 :             pattern = 41;
318 :             cnt1 = 0;
319 :             break;
320 :         }
321 :         if( iEncoder >= 10 ) {      /* クロスライン後のスピード制御 */
322 :             speed2( 0 ,0 );
323 :         } else {
324 :             speed2( 70 ,70 );
325 :         }
326 :         switch( sensor_inp(MASK3_3) ) {
327 :             case 0x00:
328 :                 /* センタ まっすぐ */
329 :                 handle( 0 );
330 :                 break;
331 :             case 0x04:
332 :             case 0x06:
333 :             case 0x07:
334 :             case 0x03:
335 :                 /* 左寄り 右曲げ */
336 :                 handle( 8 );
337 :                 break;
338 :             case 0x20:
339 :             case 0x60:
340 :             case 0xe0:
341 :             case 0xc0:
342 :                 /* 右寄り 左曲げ */
343 :                 handle( -8 );
344 :                 break;
345 :         }
346 :         break;
```

パターン 23 は、直前にクランクがある状態です。この時点でスピードが遅ければ良いですが、速すぎればクランクを曲がり切れません。そこで、パターン 23 でもスピードをチェックし、速すぎればブレーキをかけます。

第 9 回大会までは、クロスラインの 100cm 後にクランクがありました。第 10 回大会から、クロスラインの 50～100cm 後にクランクがあることとなりました。

プログラムは、50cm 後にクランクがあると仮定して調整します。50cm 進んだときにスピードが落とし切れていれば、後はそのスピードを保って進めば 60cm だろうが 100cm だろうが対応できます。

モータドライブ基板 Vol.3 は逆転も可能です。ブレーキ(PWM0%)だけでスピードを落とすきれない場合は、逆転ブレーキで急減速すると良いでしょう。ただし、エンコーダ値をきちんと見ないとバックしてしまうので注意が必要です。

ほかのパターン 53、パターン 64 でのスピード調整も同様です。

5.3 エンコーダの回転数が違う場合の変更点

このサンプルプログラムは、100 パルス / 回転、エンコーダのタイヤ直径 33mm のエンコーダを使用した場合です。条件が違うとき、プログラムを変更しなければいけない部分を下記に示します。

行番号	元の数値	変更後の数値
250	10	秒速 1m/s のときのパルス数を入れます。 例)200 パルス / 回転、直径 33mm なら 19
274	10	秒速 1m/s のときのパルス数を入れます。 例)200 パルス / 回転、直径 33mm なら 19
321	10	秒速 1m/s のときのパルス数を入れます。 例)200 パルス / 回転、直径 33mm なら 19
408	10	秒速 1m/s のときのパルス数を入れます。 例)200 パルス / 回転、直径 33mm なら 19
472	10	秒速 1m/s のときのパルス数を入れます。 例)200 パルス / 回転、直径 33mm なら 19

6. プロジェクト「kit06enc_03」 距離の検出(パターンの区分けを距離で行う)

kit06 標準プログラムは、クロスラインを検出後のパターン 22 では、100ms の間はセンサを見ません。100ms 後は、クロスラインが終わった直前と仮定しています。しかし、スピードが速いと 100ms 間でかなり進んでしまいます。進む距離が多いほどセンサを見ていない訳ですから中心からのずれが大きくなってしまいます。パターン 42、パターン 52 も同様です。そこで、距離を検出できるエンコーダがあるので**クロスラインを検出してからパターン 22 を 10cm、右ハーフラインを検出してからパターン 52 を 10cm、左ハーフラインを検出してからパターン 62 を 10cm 実行**するように改造します。

距離にすれば、マイコンカーのスピードが速いので進む距離が長くなってしまふということがありません。

6.1 プログラム

プログラムのゴシック体部分が追加した部分です。

前略

```

67 : /*=====*/
68 : /* グローバル変数の宣言 */
69 : /*=====*/
70 : unsigned long cnt0; /* timer関数用 */
71 : unsigned long cnt1; /* main内で使用 */
72 : int pattern; /* パターン番号 */
73 :
74 : /* エンコーダ関連 */
75 : int iTimer10; /* エンコーダ取得間隔 */
76 : long lEncoderTotal; /* 積算値 */
77 : int iEncoderMax; /* 現在最大値 */
78 : int iEncoder; /* 現在値 */
79 : unsigned int uEncoderBuff; /* 前回値保存 */
80 : long lEncoderLine; /* ライン検出ときの積算値 */
81 :
82 : /*=====*/
83 : /* メインプログラム */
84 : /*=====*/
85 : void main( void )
86 : {
87 :     int i;
88 :
89 :     /* マイコン機能の初期化 */
90 :     init(); /* 初期化 */
91 :     set_ccr( 0x00 ); /* 全体割り込み許可 */
92 :
93 :     /* マイコンカーの状態初期化 */
94 :     handle( 0 );
95 :     speed( 0, 0 );
96 :
97 :     while( 1 ) {
98 :
99 :         P4DR = ~pattern;
100 :
101 :         switch( pattern ) {

```

中略

```

143 :     case 1:
144 :         /* スタートバーが開いたかチェック */
145 :         if( !startbar_get() ) {
146 :             /* スタート!! */
147 :             lEncoderTotal = 0;
148 :             led_out( 0x0 );
149 :             pattern = 11;
150 :             cnt1 = 0;
151 :             break;
152 :         }
153 :         if( cnt1 < 50 ) { /* LED点滅処理 */
154 :             led_out( 0x1 );
155 :         } else if( cnt1 < 100 ) {
156 :             led_out( 0x2 );
157 :         } else {
158 :             cnt1 = 0;
159 :         }
160 :         break;

```

中略

ロータリエンコーダ 実習マニュアル kit06 版

```
286 :     case 21:
287 :         /* 1本目のクロスライン検出ときの処理 */
288 :         IEncoderLine = IEncoderTotal;
289 :         led_out( 0x3 );
290 :         handle( 0 );
291 :         speed( 0 ,0 );
292 :         pattern = 22;
293 :         cnt1 = 0;
294 :         break;
295 :
296 :     case 22:
297 :         /* 2本目を読み飛ばす */
298 :         if( IEncoderTotal-IEncoderLine >= 97 ) { /* 約10cmたったか? */
299 :             pattern = 23;
300 :             cnt1 = 0;
301 :         }
302 :         break;
```

中略

```
385 :     case 51:
386 :         /* 1本目の右ハーフライン検出ときの処理 */
387 :         IEncoderLine = IEncoderTotal;
388 :         led_out( 0x2 );
389 :         handle( 0 );
390 :         speed( 0 ,0 );
391 :         pattern = 52;
392 :         cnt1 = 0;
393 :         break;
394 :
395 :     case 52:
396 :         /* 2本目を読み飛ばす */
397 :         if( IEncoderTotal-IEncoderLine >= 97 ) { /* 約10cmたったか? */
398 :             pattern = 53;
399 :             cnt1 = 0;
400 :         }
401 :         break;
```

中略

```
450 :     case 61:
451 :         /* 1本目の左ハーフライン検出ときの処理 */
452 :         IEncoderLine = IEncoderTotal;
453 :         led_out( 0x1 );
454 :         handle( 0 );
455 :         speed( 0 ,0 );
456 :         pattern = 62;
457 :         cnt1 = 0;
458 :         break;
459 :
460 :     case 62:
461 :         /* 2本目を読み飛ばす */
462 :         if( IEncoderTotal-IEncoderLine >= 97 ) { /* 約10cmたったか? */
463 :             pattern = 63;
464 :             cnt1 = 0;
465 :         }
466 :         break;
```

以下、略

6.2 プログラムの解説

6.2.1 変数の追加

```

74 : /* エンコーダ関連 */
75 : int          iTimer10;          /* エンコーダ取得間隔 */
76 : long         lEncoderTotal;     /* 積算値 */
77 : int          iEncoderMax;       /* 現在最大値 */
78 : int          iEncoder;          /* 現在値 */
79 : unsigned int uEncoderBuff;     /* 前回値保存 */
80 : long         lEncoderLine;      /* ライン検出ときの積算値 */

```

80 行に lEncoderLine 変数を追加しています。この変数には、クロスライン、右ハーフライン、左ハーフラインを検出した位置の積算値を記憶させておきます。

6.2.2 積算値のクリア

```

143 :     case 1:
144 :         /* スタートバーが開いたかチェック */
145 :         if( !startbar_get() ) {
146 :             /* スタート!! */
147 :             lEncoderTotal = 0;
148 :             led_out( 0x0 );
149 :             pattern = 11;
150 :             cnt1 = 0;
151 :             break;
152 :         }
153 :         if( cnt1 < 50 ) {          /* LED 点滅処理 */
154 :             led_out( 0x1 );
155 :         } else if( cnt1 < 100 ) {
156 :             led_out( 0x2 );
157 :         } else {
158 :             cnt1 = 0;
159 :         }
160 :         break;

```

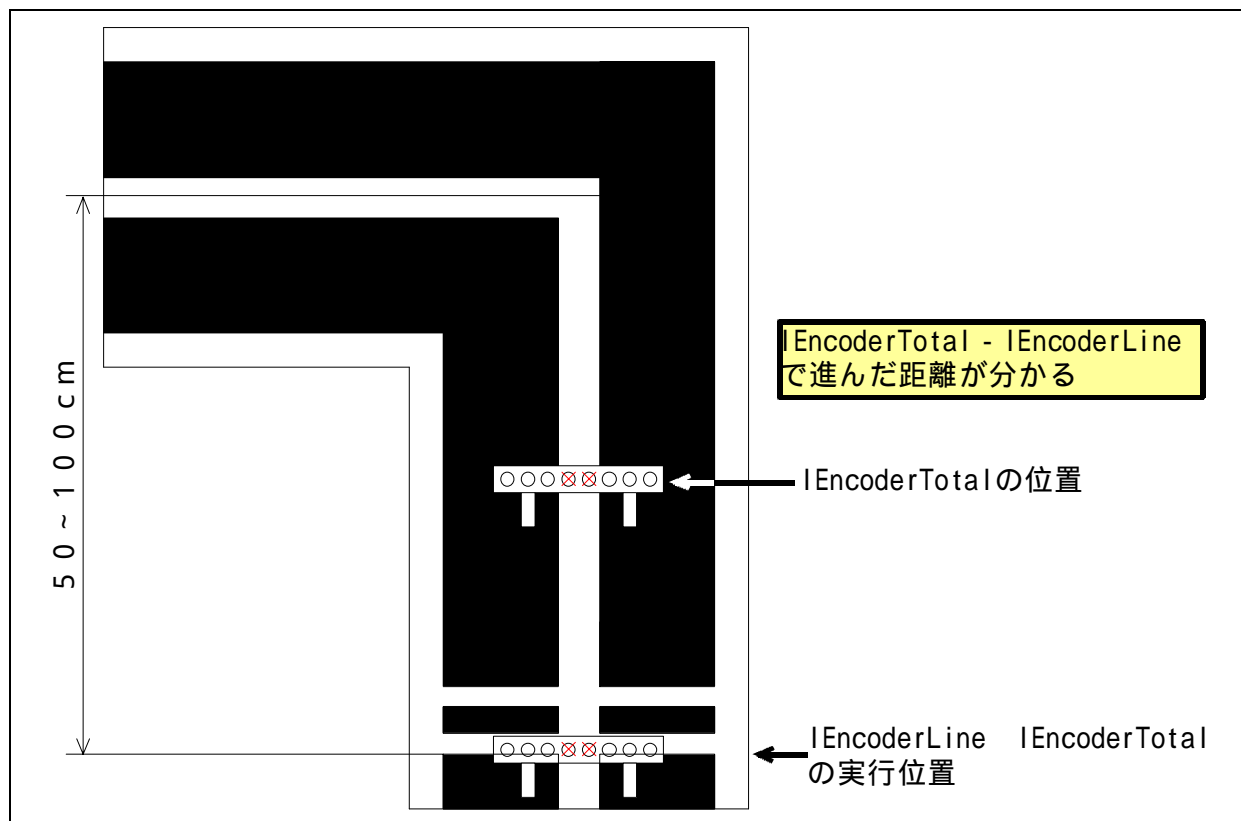
この変数は、電源を入れてから積算を開始します。そのため、スタート前もカウントしています。lEncoderTotal 変数は、コースを走行した距離を測るのが目的ですので、走行前からカウントされると距離が変わってしまいます。そこで、スタート直前に lEncoderTotal 変数をクリアします。

6.2.3 パターン 21 クロスライン検出ときの積算値を取得

```

286 :     case 21:
287 :         /* 1本目のクロスライン検出ときの処理 */
288 :         IEncoderLine = IEncoderTotal;
289 :         led_out( 0x3 );
290 :         handle( 0 );
291 :         speed( 0 ,0 );
292 :         pattern = 22;
293 :         cnt1 = 0;
294 :         break;
    
```

クロスラインを検出した瞬間の積算値 IEncoderTotal の値を、IEncoderLine にコピーしています。IEncoderTotal - IEncoderLine で、クロスラインを検出してからのパルス数が分かります。要は、クロスラインから進んだ距離が分かります。



6.2.4 パターン 22 2本目を読み飛ばす

```

296 :     case 22:
297 :         /* 2本目を読み飛ばす */
298 :         if( IEncoderTotal-IEncoderLine >= 97 ) { /* 約 10cm たったか? */
299 :             pattern = 23;
300 :             cnt1 = 0;
301 :         }
302 :         break;

```

298 行で、10cm 進んだかチェックしています。距離は、1 本目の白線 2cm + 黒部分 3cm + 2 本目の白線 2cm で、合計 7cm です。余裕を見て 10cm としています。次のような意味です。

$$IEncoderTotal - IEncoderLine \geq 10cm$$

$$\text{現在の積算値} - \text{クロスラインを検出したときの積算値} \geq 10cm$$

$$\text{クランク内で進んだパルス数(距離)} \geq 10cm$$

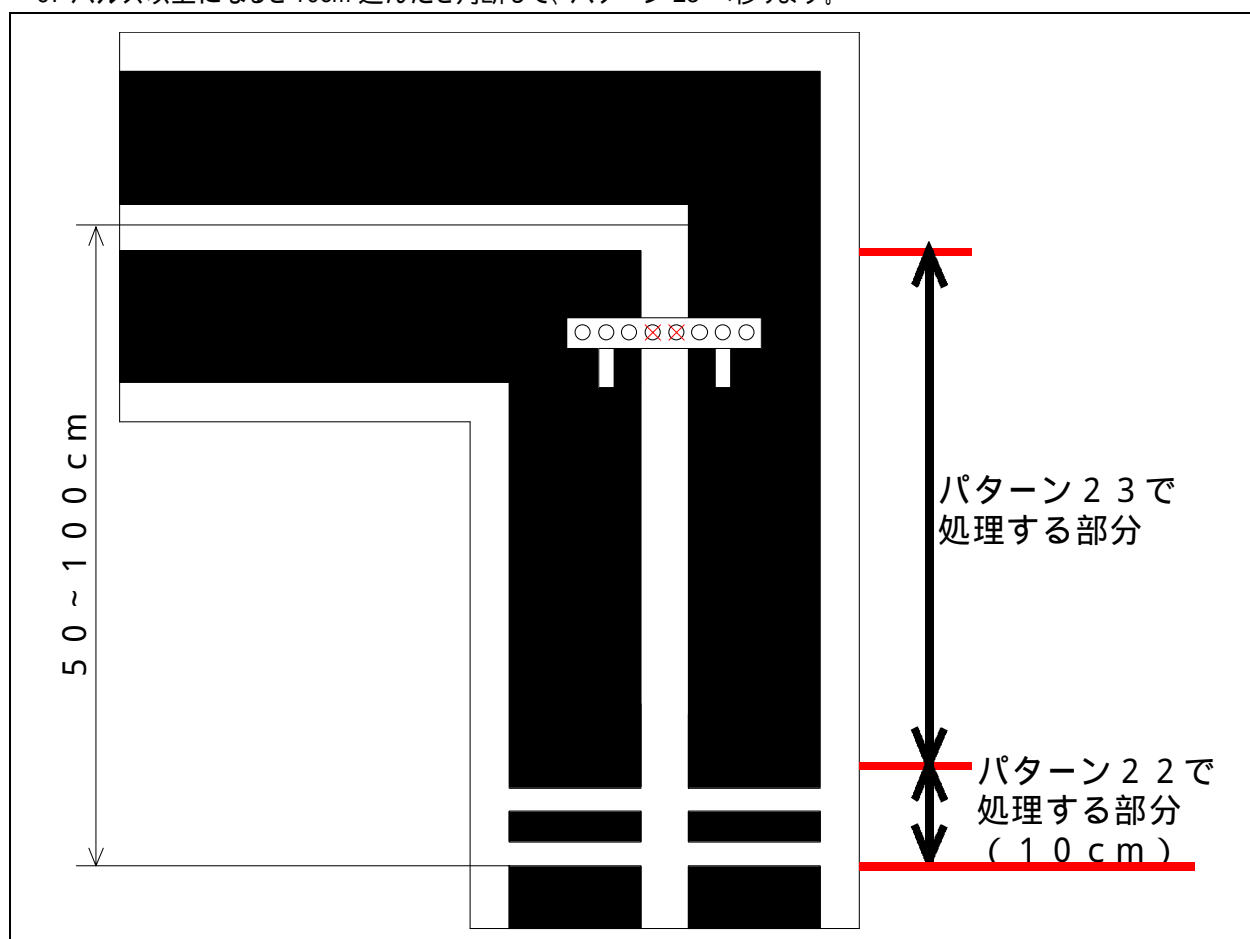
今回のエンコーダは 1m で 965 パルスのエンコーダなので、10cm 進んだかどうかチェックするには、

$$1m : 965 \text{ パルス} = 0.1m : x \text{ パルス}$$

$$x = 96.5 \text{ パルス}$$

と、クロスラインを検出した瞬間から 97 パルス以上になったかプログラムで見れば良いことになります。

97 パルス以上になると 10cm 進んだと判断して、パターン 23 へ移ります。

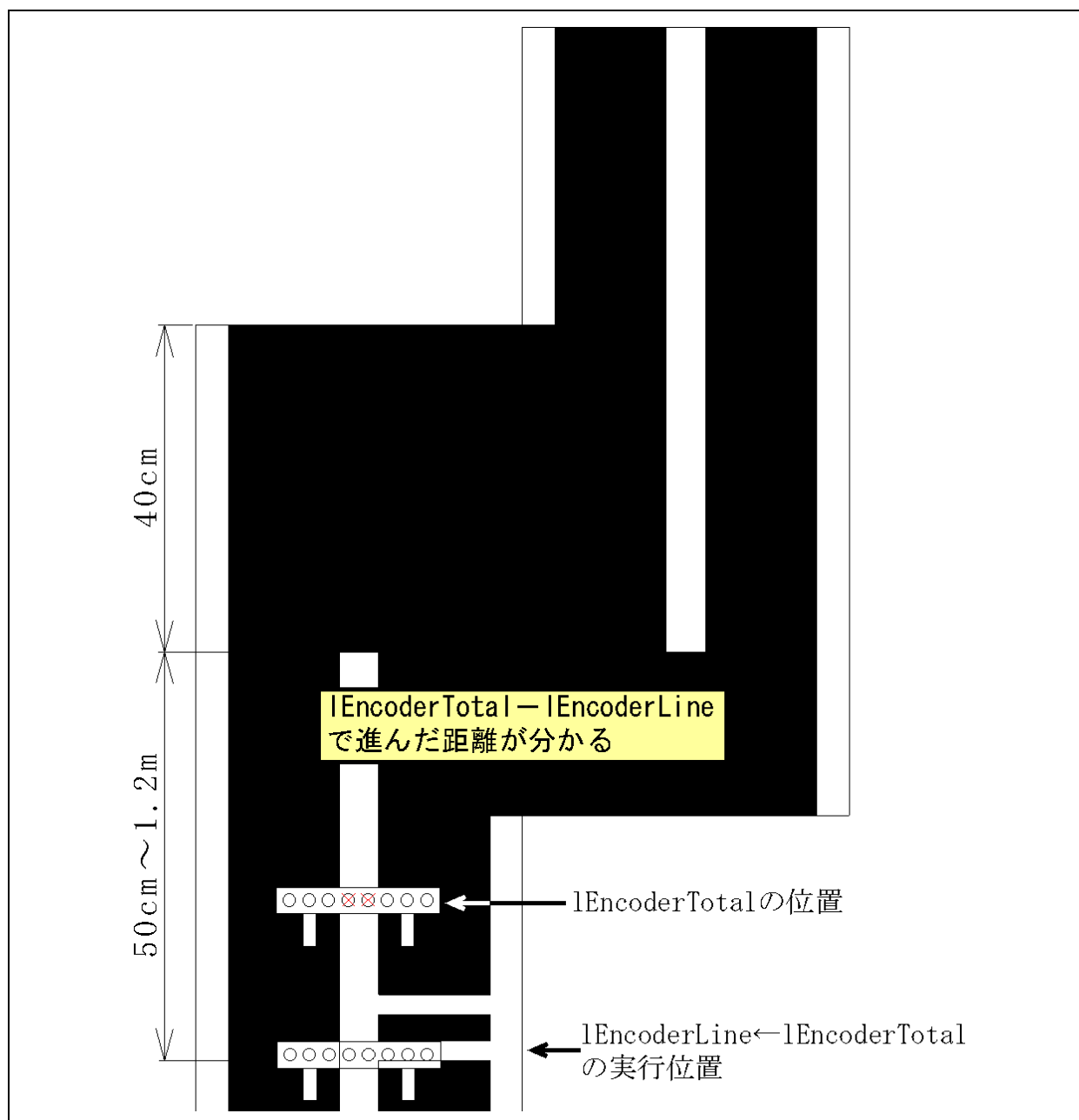


6.2.5 パターン 51 右ハーフライン検出ときの積算値を取得

```

385 :     case 51:
386 :         /* 1本目の右ハーフライン検出ときの処理 */
387 :         lEncoderLine = lEncoderTotal;
388 :         led_out( 0x2 );
389 :         handle( 0 );
390 :         speed( 0 ,0 );
391 :         pattern = 52;
392 :         cnt1 = 0;
393 :         break;
    
```

右ハーフラインを検出した瞬間の積算値 lEncoderTotal の値を、lEncoderLine にコピーしています。lEncoderTotal - lEncoderLine で、右ハーフラインを検出してからのパルス数が分かります。要は、**右ハーフラインから進んだ距離**が分かります。



6.2.6 パターン 52 2本目を読み飛ばす

```

395 :     case 52:
396 :         /* 2本目を読み飛ばす */
397 :         if( IEncoderTotal-IEncoderLine >= 97 ) { /* 約10cm たったか? */
398 :             pattern = 53;
399 :             cnt1 = 0;
400 :         }
401 :         break;

```

397 行で、10cm 進んだかチェックしています。距離は、1 本目の白線 2cm + 黒部分 3cm + 2 本目の白線 2cm で、合計 7cm です。余裕を見て 10cm としています。次のような意味です。

IEncoderTotal - IEncoderLine \geq 10cm

現在の積算値 - 右ハーフラインを検出したときの積算値 \geq 10cm

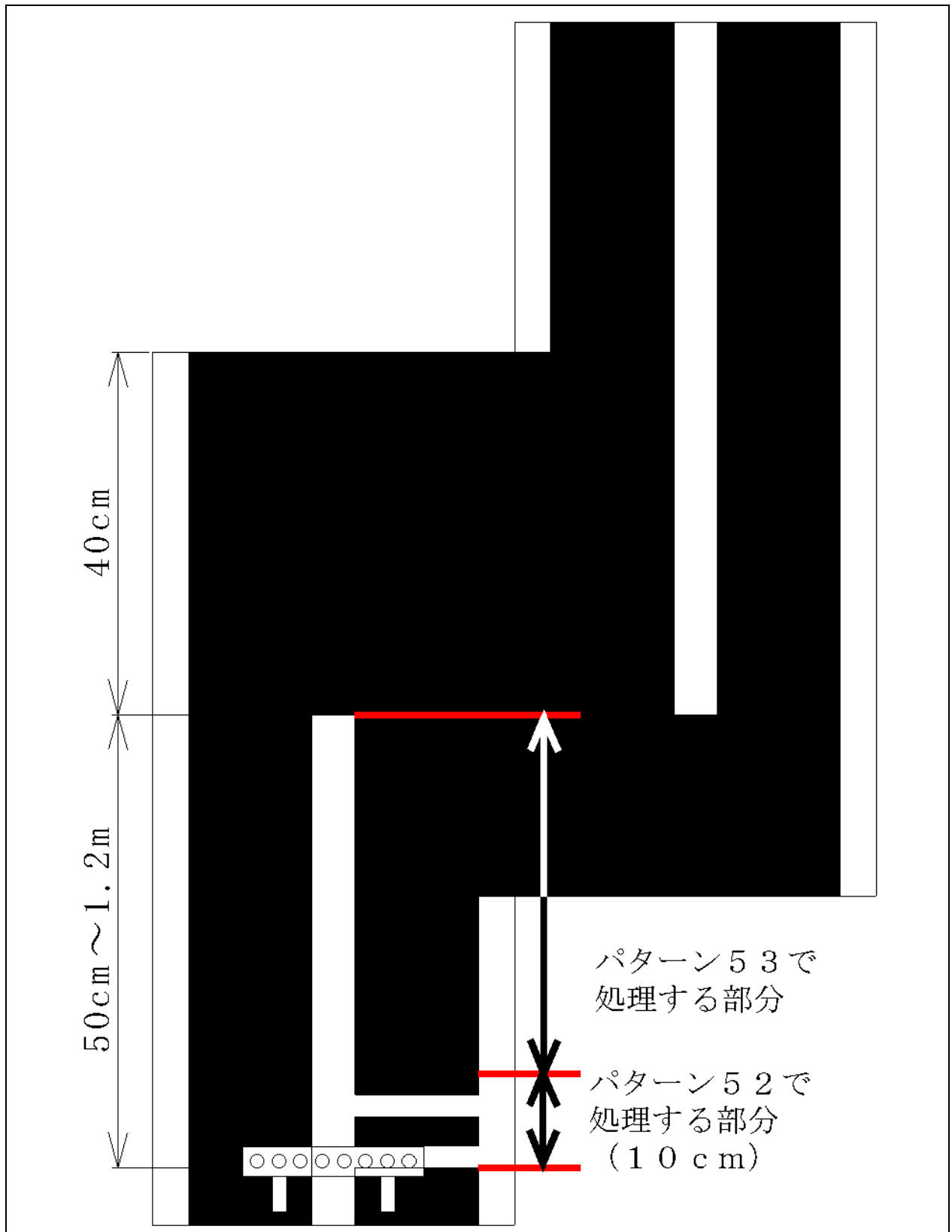
右ハーフライン検出後に進んだパルス数(距離) \geq 10cm

今回のエンコーダは 1m で 965 パルスのエンコーダなので、10cm 進んだかどうかチェックするには、

1m : 965 パルス = 0.1m : x パルス

x = 96.5 パルス

と、右ハーフラインを検出した瞬間から 97 パルス以上になったかプログラムで見れば良いことになります。97 パルス以上になると 10cm 進んだと判断して、パターン 53 へ移ります。



6.2.7 パターン 61～62 左ハーフライン部分の処理

```

450 :     case 61:
451 :         /* 1本目の左ハーフライン検出ときの処理 */
452 :         IEncoderLine = IEncoderTotal;
453 :         led_out( 0x1 );
454 :         handle( 0 );
455 :         speed( 0 ,0 );
456 :         pattern = 62;
457 :         cnt1 = 0;
458 :         break;
459 :
460 :     case 62:
461 :         /* 2本目を読み飛ばす */
462 :         if( IEncoderTotal-IEncoderLine >= 97 ) { /* 約10cm たったか? */
463 :             pattern = 63;
464 :             cnt1 = 0;
465 :         }
466 :         break;
503 :     }
504 :     break;

```

パターン 61、62 は、パターン 51、52 部分と比べ、右ハーフラインが左ハーフラインに変わるだけです。452 行で左ハーフラインを検出したときの距離を記憶します。10cm 進むとパターン 62 へ移ります。

6.3 エンコーダの回転数が違う場合の変更点

このサンプルプログラムは、100 パルス / 回転、エンコーダのタイヤ直径 33mm のエンコーダを使用した場合です。条件が違うとき、プログラムを変更しなければいけない部分を下記に示します。

行番号	元の数値	変更後の数値
252	10	秒速 1m/s ときのパルス数を入れます。 例)200 パルス / 回転、直径 33mm なら 19
276	10	秒速 1m/s ときのパルス数を入れます。 例)200 パルス / 回転、直径 33mm なら 19
298	97	10cm 進んだときのパルス数を入れます。 例)200 パルス / 回転、直径 33mm なら 193
324	10	秒速 1m/s ときのパルス数を入れます。 例)200 パルス / 回転、直径 33mm なら 19
397	97	10cm 進んだときのパルス数を入れます。 例)200 パルス / 回転、直径 33mm なら 193
412	10	秒速 1m/s ときのパルス数を入れます。 例)200 パルス / 回転、直径 33mm なら 19
462	97	10cm 進んだときのパルス数を入れます。 例)200 パルス / 回転、直径 33mm なら 193
477	10	秒速 1m/s ときのパルス数を入れます。 例)200 パルス / 回転、直径 33mm なら 19

7. プーリーを使用した自作エンコーダのプログラム

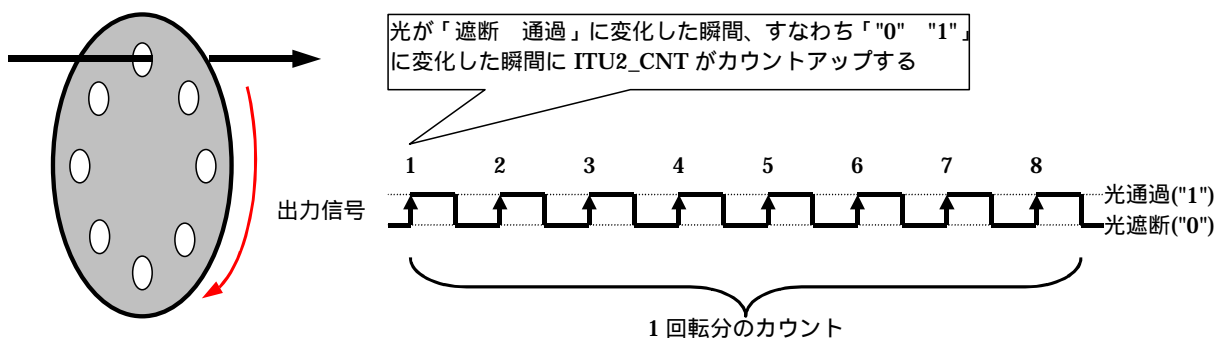
7.1 プーリーを使用したときの回転数計算

プーリーを使用したエンコーダの自作方法を紹介しました。下写真のようにプーリーに 8 つの穴が空いているとします。



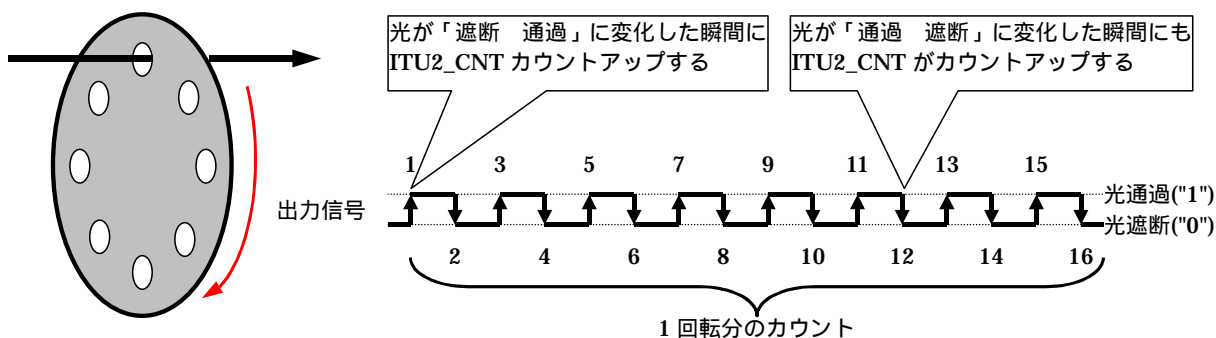
8 つの穴が空いているので、8 パルス / 回転となります。ITU2 を使用してパルスカウントします。パルス入力端子は PA0 とします。ITU2_TCR の設定は下記のようになります。

```
ITU2_TCR = 0x04; /* PA0 端子のパルスでカウント*/
```



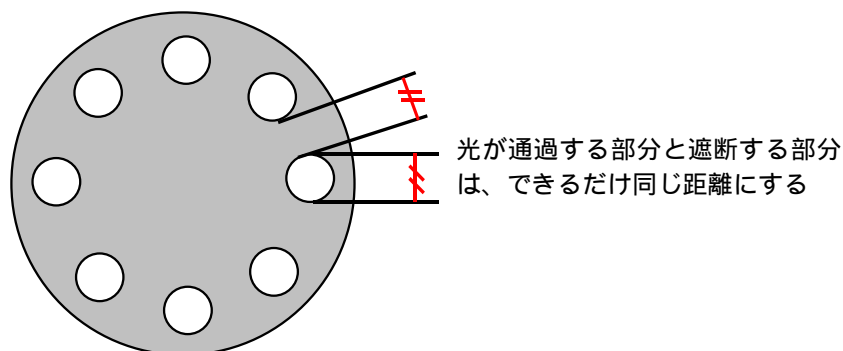
8 パルス/回転では、ちょっとパルス数が少ないですね。ITU2_TCR の設定には、「信号の立ち上がり("0" "1" になった瞬間)でカウントしなさい」という設定の他、「信号の立ち下がり("1" "0" になった瞬間)でカウントしなさい」と、「信号の立ち上がり、立ち下がり両方でカウントしなさい」という設定ができます。そこで、両方でカウントするようにします。プログラムは下記のようになります。

```
ITU2_TCR = 0x14; /* PA0 端子のパルスでカウント*/
```



この設定で 16 パルス / 回転となりました。

ただし、この方法を使用する場合は、光が通過する部分と遮断する部分の距離をほぼ同じにします(下図)。



「2.3.7 パルス数とスピード(距離)の関係」のとおり、1m/s で進んだときの 10ms 間にカウントするパルス数は、2.43 パルスです。プログラムでは 2.43 パルス以上か、または以下かチェックします。しかし、プログラムでは整数しか扱うことができません。そのため、1 パルス出力されたとき、2 パルス出力されたとき・・・というように 1 パルス当たりの速度がどのくらいか逆算しておく、プログラム作成時に便利です。

$$1\text{m/s で進んだときの }10\text{ms 間にカウントするパルス数} : 2.43 \text{ パルス} = x \text{ m/s で進んだときの }10\text{ms 間にカウントするパルス数} : 1 \text{ パルス}$$

$$x = 0.41[\text{m/s}]$$

下表のようになります。

10ms 間計測したときの パルス数	速度[m/s]
0	0.41 * 0 = 0
1	0.41 * 1 = 0.41
2	0.41 * 2 = 0.82
3	0.41 * 3 = 1.23
4	0.41 * 4 = 1.64
5	0.41 * 5 = 2.05
6	0.41 * 6 = 2.46
7	0.41 * 7 = 2.87
8	0.41 * 8 = 3.28
9	0.41 * 9 = 3.69
10	0.41 * 10 = 4.10
11	0.41 * 11 = 4.51
12	0.41 * 12 = 4.92
13	0.41 * 13 = 5.33
14	0.41 * 14 = 5.74
15	0.41 * 15 = 6.15

この設定を使用して、プログラムしていきます。

7.2 速度のチェック

速度制御する場合は、iEncoder 変数を使用します。秒速 1m/s 以上なら PWM0%、以下なら PWM70%にするプログラムを作るとします。1 パルス当たり 0.41m/s なので 1m/s に一番近いパルス数は、2 パルスの 0.82m/s か 3 パルスのときの 1.23m/s です。ここでは 1.23m/s とします。

```

if( iEncoder >= 3 ) { /* 1.23m/s 以上かどうか */
    speed2( 0 ,0 );
} else {
    speed2( 70 ,70 );
}

```

7.3 距離のチェック

距離のチェックには、lEncoderTotal 変数を使用します。プーリーが 1 回転すると、lEncoderTotal 変数には 16 が入ります。2 回転すると 32、3 回転すると…… というように積算され続けていきます。この数値は先ほど計算したとおり、243 で 1m の距離に相当します。例えば、10m 進んだらブレーキをかけるプログラムを作るとします。1m で 243 なので、10m はその 10 倍の 2430 となります。

```

if( lEncoderTotal >= 2430 ) { /* 10m 以上かどうか */
    speed( 0 ,0 );
} else {
    speed( 100 ,100 );
}

```

このように、プーリーを使った自作エンコーダでも、スピード制御することができます。

7.4 kit06enc_01.c を改造して、自作エンコーダに対応させる場合の変更

行番号	元の数値、設定	変更後の数値
525	ITU2_TCR = 0x04;	立ち上がり、立ち下がりでカウントさせるため、ITU2_TCR = 0x14; とします。

変更後のプログラムは、「kit06enc_11.c」です。

7.5 kit06enc_02.c を改造して、自作エンコーダに対応させる場合の変更

行番号	元の数値、設定	変更後の数値
250	10	秒速 1m/s のときのパルス数を入れます。今回は 1m/s は設定できないので 1.23m/s とし、3 を設定します。
274	10	秒速 1m/s のときのパルス数を入れます。今回は 1m/s は設定できないので 1.23m/s とし、3 を設定します。
321	10	秒速 1m/s のときのパルス数を入れます。今回は 1m/s は設定できないので 1.23m/s とし、3 を設定します。
408	10	秒速 1m/s のときのパルス数を入れます。今回は 1m/s は設定できないので 1.23m/s とし、3 を設定します。
472	10	秒速 1m/s のときのパルス数を入れます。今回は 1m/s は設定できないので 1.23m/s とし、3 を設定します。
543	ITU2_TCR = 0x04;	立ち上がり、立ち下がりでカウントさせるため、ITU2_TCR = 0x14; とします。

変更後のプログラムは、「kit06enc_12.c」です。

7.6 kit06enc_03.c を改造して、自作エンコーダに対応させる場合の変更

行番号	元の数値、設定	変更後の数値
252	10	秒速 1m/s のときのパルス数を入れます。今回は 1m/s は設定できないので 1.23m/s とし、3 を設定します。
276	10	秒速 1m/s のときのパルス数を入れます。今回は 1m/s は設定できないので 1.23m/s とし、3 を設定します。
298	97	10cm 進んだときの距離を入れます。1m で 243 なので 0.1m なら $243 \times 0.1 = 24$ を設定します。
324	10	秒速 1m/s のときのパルス数を入れます。今回は 1m/s は設定できないので 1.23m/s とし、3 を設定します。
397	97	10cm 進んだときの距離を入れます。1m で 243 なので 0.1m なら $243 \times 0.1 = 24$ を設定します。
412	10	秒速 1m/s のときのパルス数を入れます。今回は 1m/s は設定できないので 1.23m/s とし、3 を設定します。
462	97	10cm 進んだときの距離を入れます。1m で 243 なので 0.1m なら $243 \times 0.1 = 24$ を設定します。
477	10	秒速 1m/s のときのパルス数を入れます。今回は 1m/s は設定できないので 1.23m/s とし、3 を設定します。
548	ITU2_TCR = 0x04;	立ち上がり、立ち下がりでカウントさせるため、ITU2_TCR = 0x14; とします。

変更後のプログラムは、「kit06enc_13.c」です。

8. 参考文献

1. (株)ルネサス テクノロジ
H8/3048 シリーズ、H8/3048F-ZTAT™ (H8/3048F、H8/3048F-ONE)ハードウェアマニュアル 第7版
2. (株)ルネサス テクノロジ 半導体トレーニングセンター C言語入門コーステキスト 第1版
3. (株)オーム社 H8 マイコン完全マニュアル 藤澤幸穂著 第1版
4. 電波新聞社 マイコン入門講座 大須賀威彦著 第1版
5. 電波新聞社 C言語でH8マイコンを使いこなす 鹿取祐二著 第1版
6. ソフトバンク(株) 新C言語入門シニア編 林晴比古著 初版
7. 共立出版(株) プログラマのためのANSI C全書 L.Ammeraal 著
吉田敬一・竹内淑子・吉田恵美子訳 初版

マイコンカーラーについての詳しい情報は、マイコンカーラー公式ホームページをご覧ください。

<http://www.mcr.gr.jp/>

H8 マイコンについての詳しい情報は、(株)ルネサス テクノロジのホームページをご覧ください。

<http://japan.renesas.com/>

の「マイコン」 「H8ファミリ」 「H8/3048B グループ」でご覧頂けます

リンクは、2007年2月現在の情報です。