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
kit07版

第 1.25 版
2009.06.01
ジャパンマイコンカーラリー実行委員会

注 意 事 項 (rev.1.4)

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、文書によるジャパンマイコンカーラリー実行委員会の事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、慎重に制作したも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、事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。

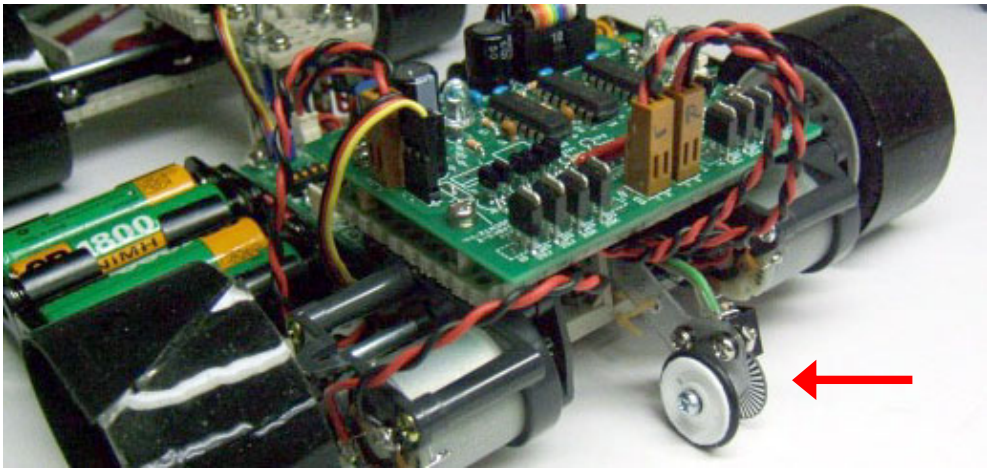
連絡先

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

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。



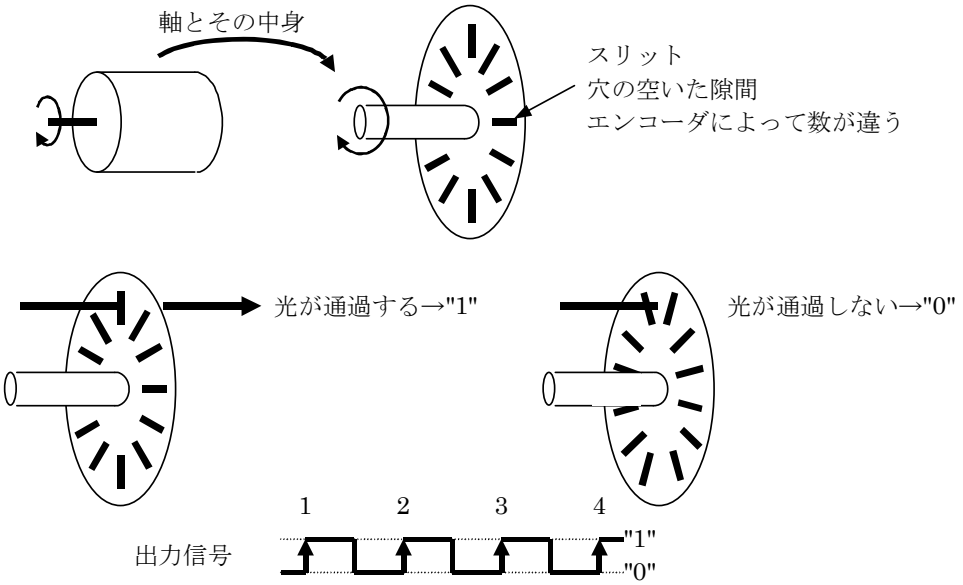
▲マイコンカーに取り付けたロータリエンコーダ（ロータリエンコーダキット Ver.2）

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。合わせると「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

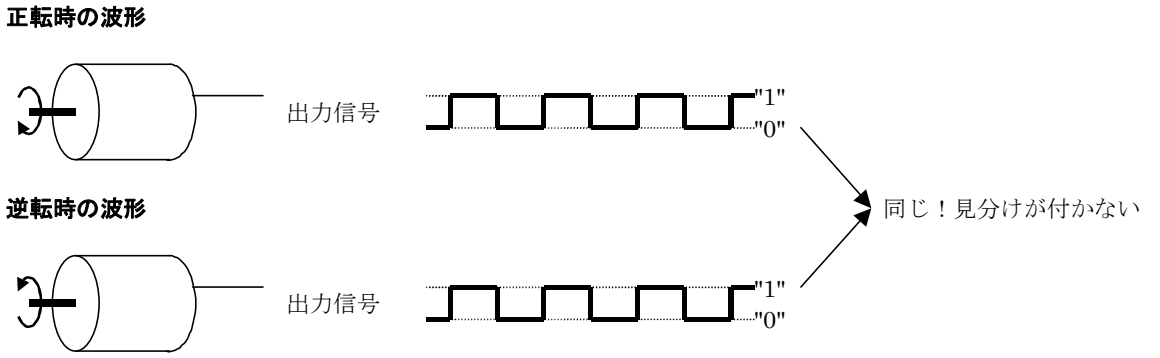
原理は、回転軸に薄い円盤が付いています。その円盤にはスリットと呼ばれる小さい隙間を空けておきます。円盤のある一点に光を通して、通過すれば″1″、しなければ″0″とします。スリットの数は、1 つの円盤に 10 個程度から数千個程度まで様々あります。当然スリット数の多い方が、値段が高くなります。



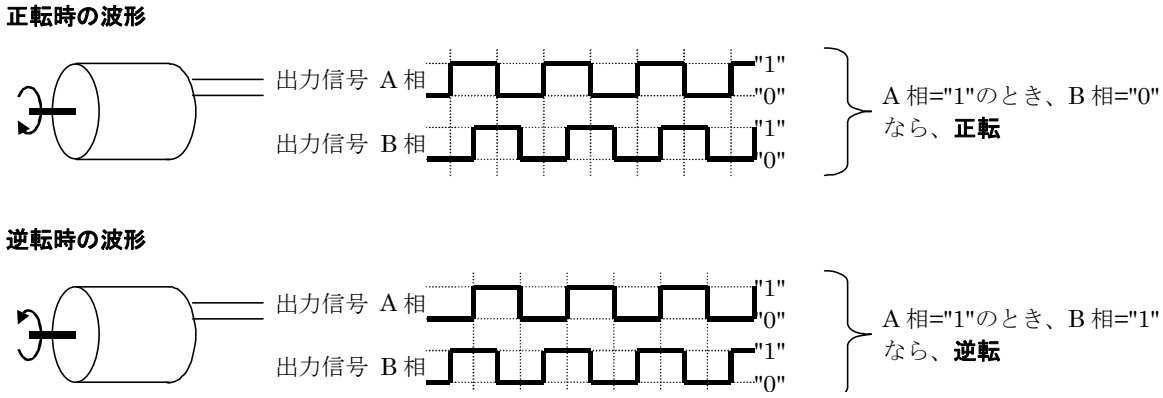
″0″から″1″になる回数を数えれば、距離が分かります。また、ある一定時間、例えば 1 秒間の回数をカウントして、多ければ回転が速い（＝スピードが速い）、少なければ回転が遅い（＝スピードが遅い）と判断できます。

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、回転が正転か逆転か分かりません。どちらも"1"と"0"の信号でしかないためです。



そこで、出力を A 相という名前と、B 相という名前で 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 A 相の出力より 90 度分ずれるようにします。このとき、下図のように A 相の信号が"0"から"1"になったとき、B 相の信号レベルを調べることにより回転方向が分かります。



最近はあまり見られなくなりましたが、パソコンのボール式マウス（光学式ではない）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

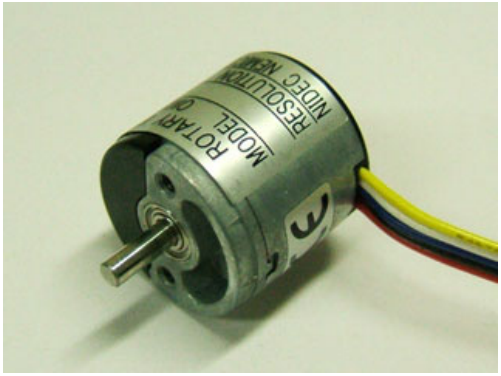
エンコーダを探すといろいろな種類があります。どのようなエンコーダがマイコンカーに使えるのでしょうか。

項目	内容
大きさ	走行に影響しない程度の大きさとします。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 などのロジック IC を入れ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。

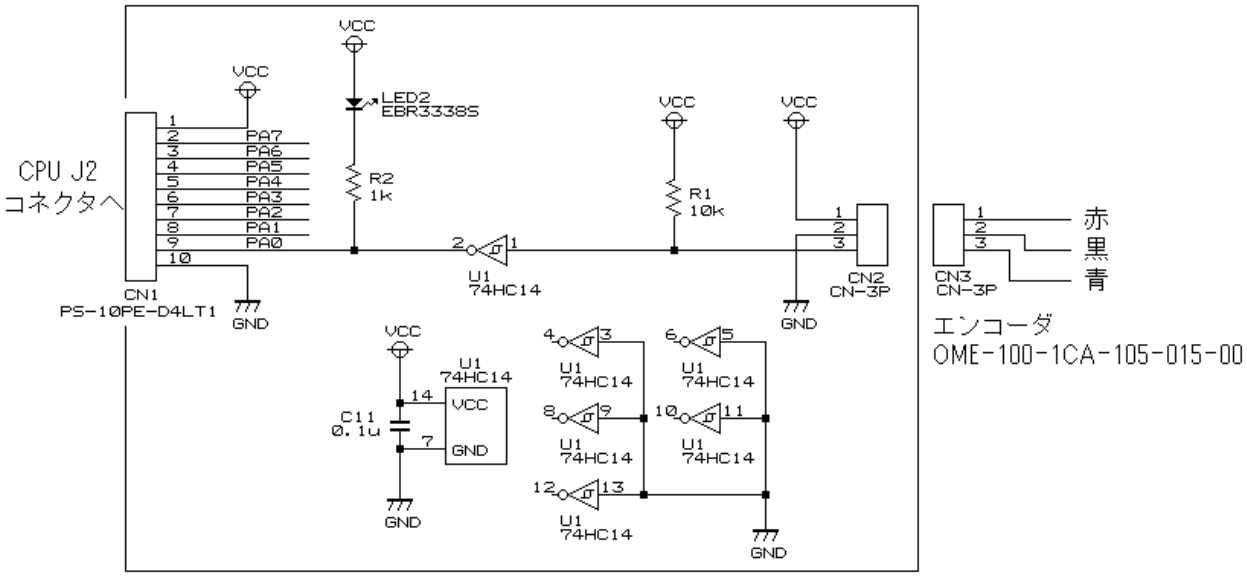
メーカ	日本電産ネミコン（株）	日本電産コパル（株）
型式	OME-100-1CA-105-015-00	RE12D-100-101-1
特徴	デジタル信号が出力されるので、マイコンで扱いやすいです。このエンコーダはオープンコレクタ出力なので、プルアップ抵抗の追加だけで使用可能です。	デジタル信号が出力されるので、マイコンで扱いやすいです。プルアップも不要です。φ12mm と小型です。
写真		

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。

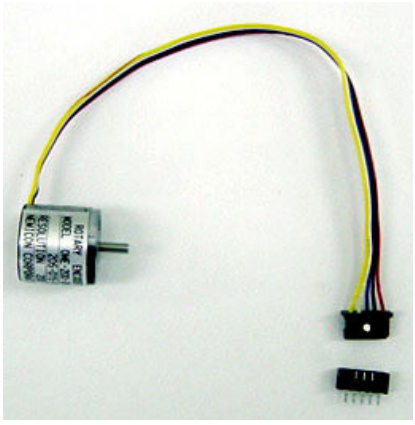
「OME-100-1CA-105-015-00」の出力信号は、デジタル信号のため、そのままポートに接続可能です。ただし、オープンコレクタ出力なのでプルアップ抵抗が必要です。一応、74HC14 で波形整形すると良いでしょう。

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。



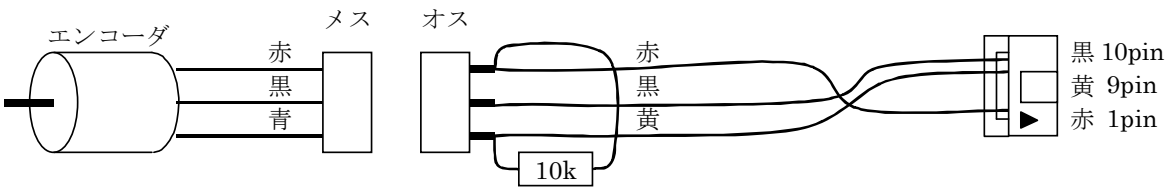
2.2.3 簡易回路

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-**2M**CA-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。

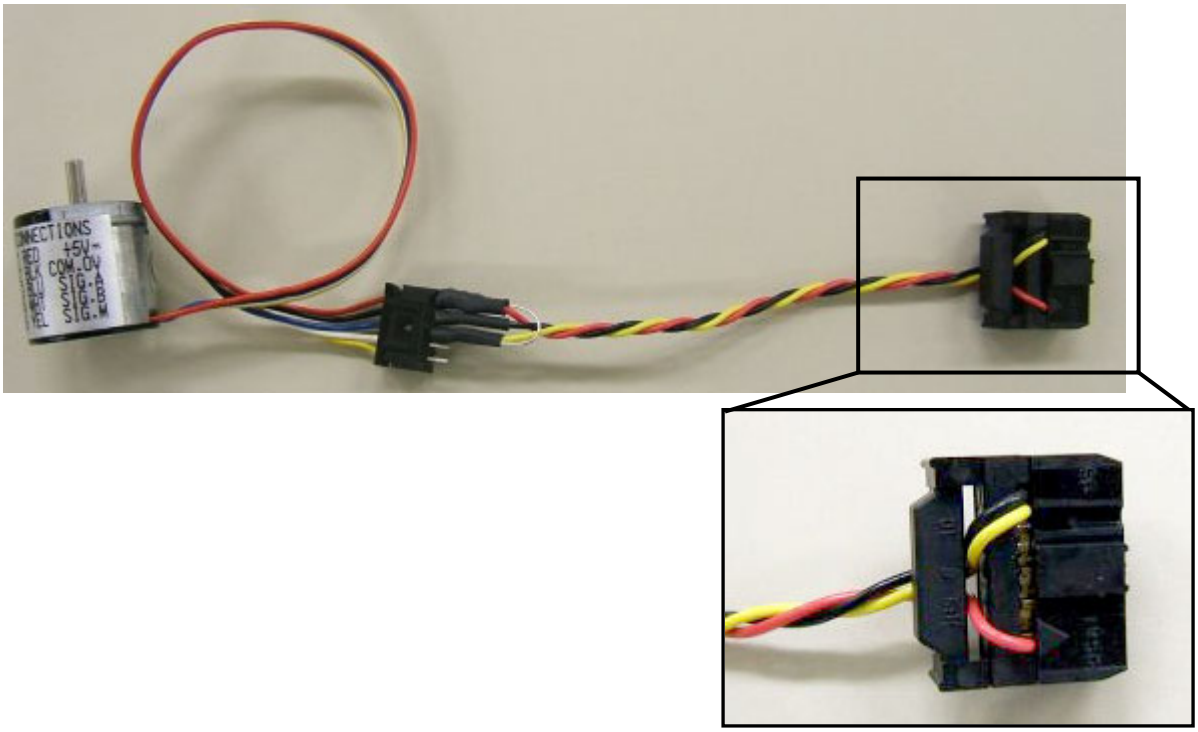
下記のように配線します。



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

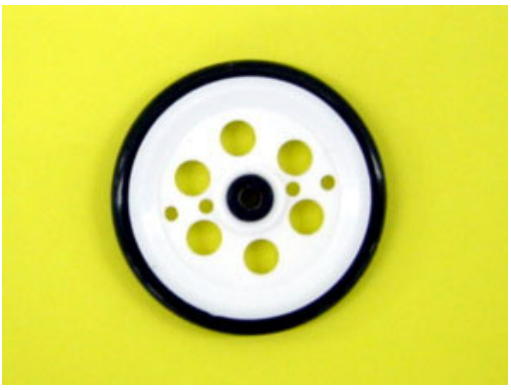


2.2.4　回転部分の加工

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などで売っているOリングを選びました。写真は東急ハンズで売っていた O リングです。「1A P15」と書いてある袋には、直径 20mm の O リングが 10 個入っています。。「1A　P25」と書いてある袋には、直径 30mm の O リングが 10 個入っています。



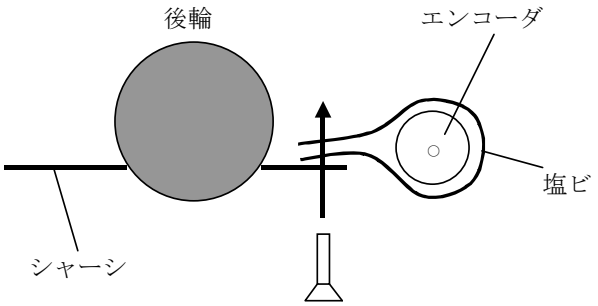
30mm のプーリーに O リングをはめたところです。ロータリエンコーダの軸の直径は 2.5mm です。プーリー(S)セットには 2mm と 3mm 径のブッシュ(プーリーの中心の黒い部品)しかありません。そのため、2mm 径のブッシュに 2.5mm のドリルで穴を開けて、エンコーダに取り付けます。

O リングをはめたタイヤの直径は、実測で 33mm になりました。

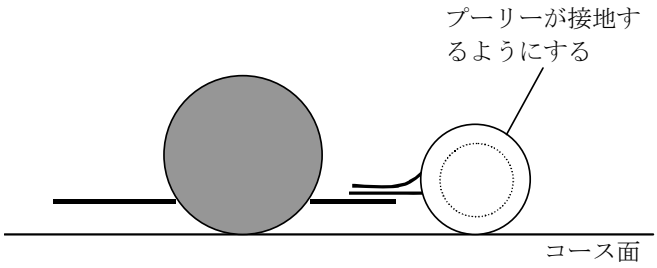
エンコーダにプーリーを取り付けました。軸とプーリーをボンドで固定すれば、はずれる心配がありません。

2.2.5 マイコンカーへの取り付け

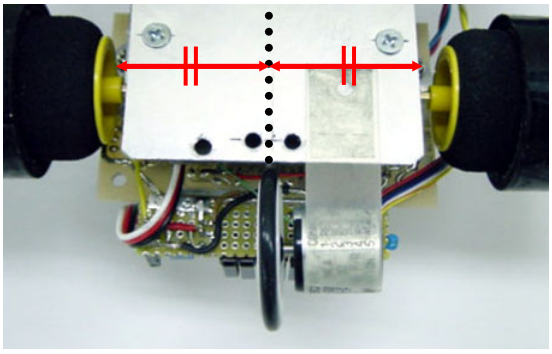
マイコンカーとロータリエンコーダを取り付ける素材として 0.5mm 厚の塩ビ板を用意しました。薄く弾力性のある素材であれば何でも構いません。



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。

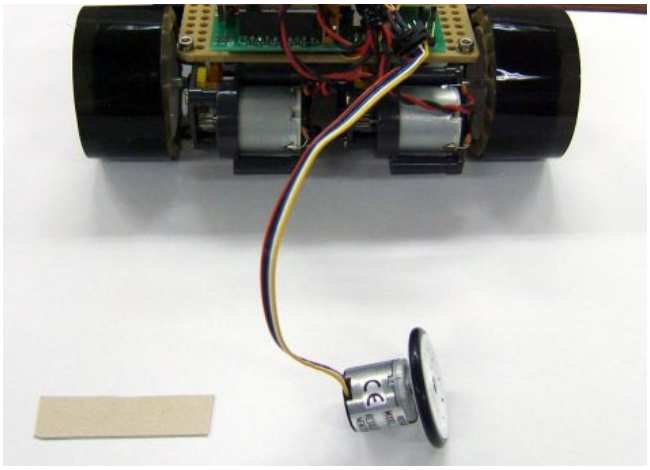


マイコンカーをコースに置いたとき、接地するようにします。圧力が強すぎると、走行に影響するので軽く圧力がかかるようにしてください。

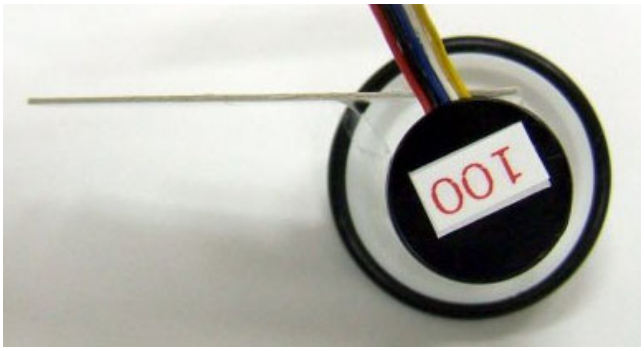


両面テープで簡単に取り付けた例です。ちょうど中心にくるように貼り付けます。

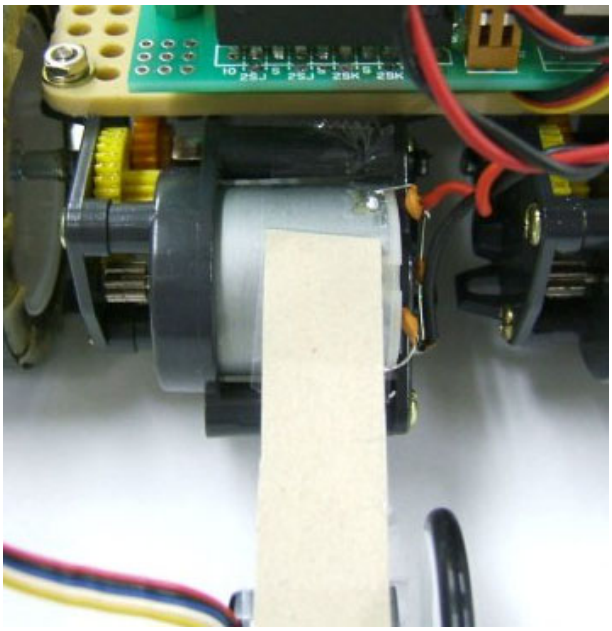
2.2.6　即席の取り付け例



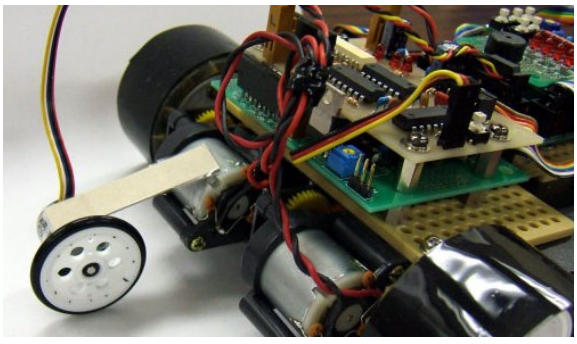
　取り付けるマイコンカー、エンコーダ、15×50mm 程度の厚紙（硬めの板）とセロテープを用意します。



　セロテープでエンコーダと厚紙を止めます。がっちりと止めます。



　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、多少上下するようにしておきます。



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

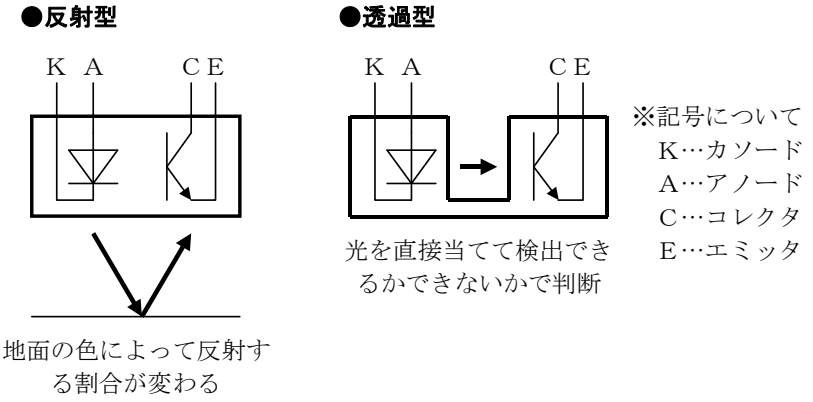
　この方法はすぐに取れてしまうので、実験のみの使用にしましょう。

2.3 フォトセンサを使った自作

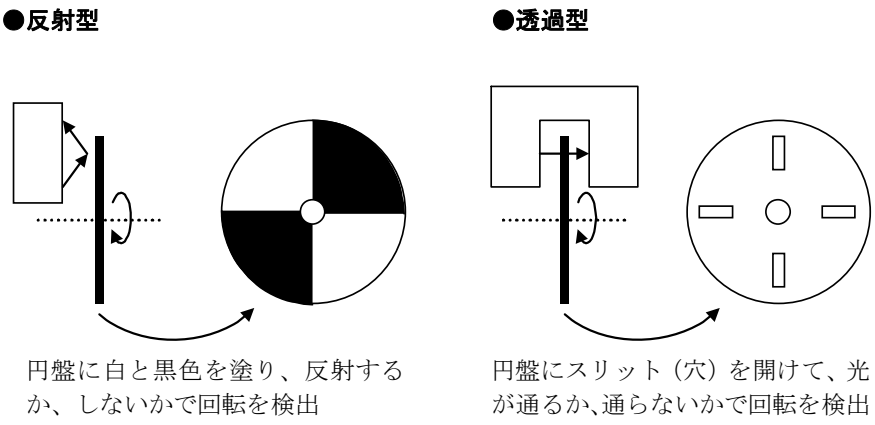
市販されているエンコーダは 1 回転 100 パルス以上と性能は申し分ありません。しかし値段が高いのが難点です。そこで、パルス数が少なくなりますが、安くできる方法を紹介します。

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 LED、受光側にはフォト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。

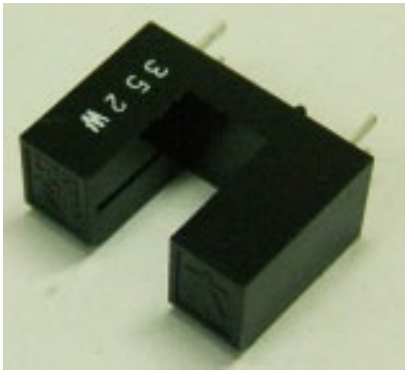


反射型、透過型のフォトインタラプタをエンコーダとして使用したときの例を下記に示します。それぞれ、取り付け方、円盤の加工の仕方が変わります。



2.3.2 透過型フォトインタラプタの例

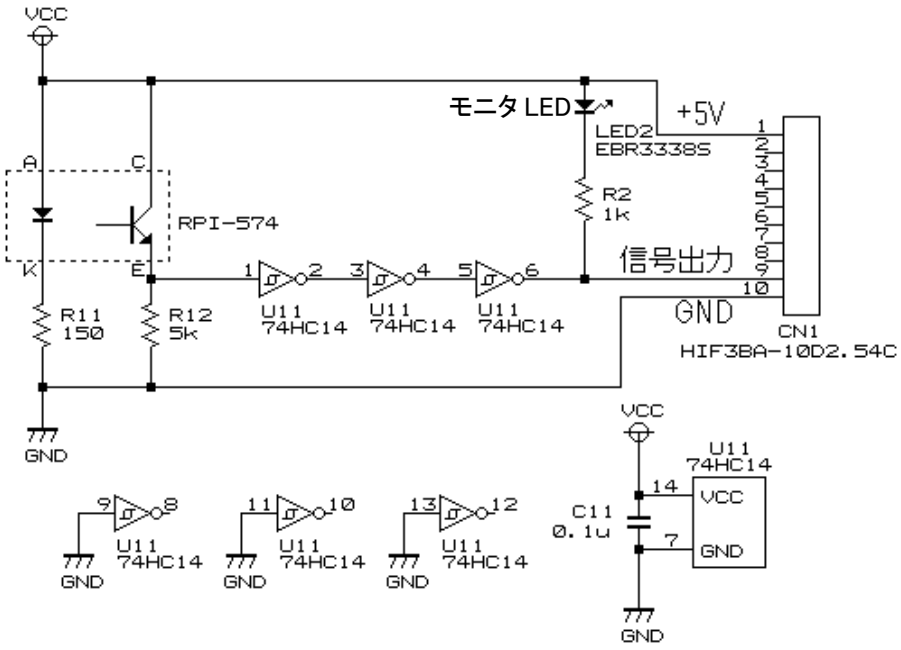
市販されている透過型フォトインタラプタでマイコンカーに使用できそうなフォトインタラプタを下記に示します。他にもたくさんありますので、調べてみると良いでしょう。

メーカ	ローム(株)	シャープ(株)
型式	RPI-574	GP1A51HRJ00F
特徴	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。	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。
写真		

2.3.3 RPI-574 を使った回路例

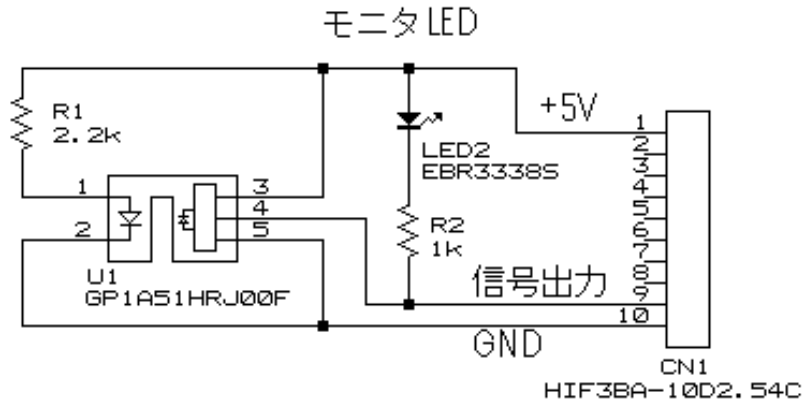
ローム(株)「RPI-574」の回路例を示します。

「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。



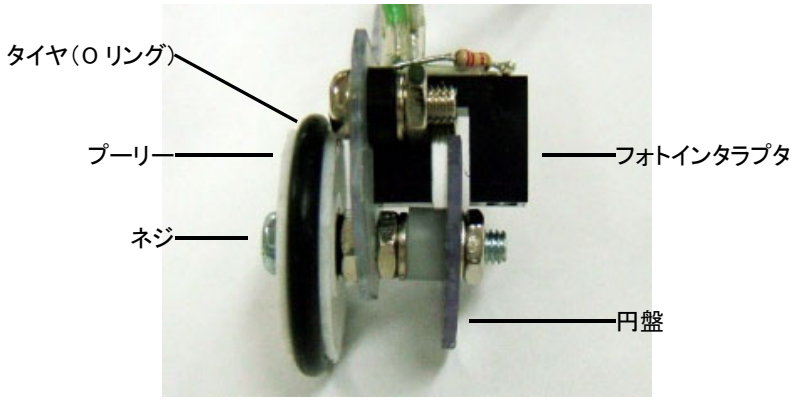
2.3.4　GP1A51HRJ00Fを使った回路例

シャープ(株)「GP1A51HRJ00F」の回路例を示します。
「GP1A51HRJ00F」の出力信号は、0V か 5V なのでそのままポートに出力することができます。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。ロータリエンコーダキット Ver.2 は、下記回路に LED の付いていない構成です。信号が来ているかどうかの確認は、モータドライブ基板の LED を使用しています。

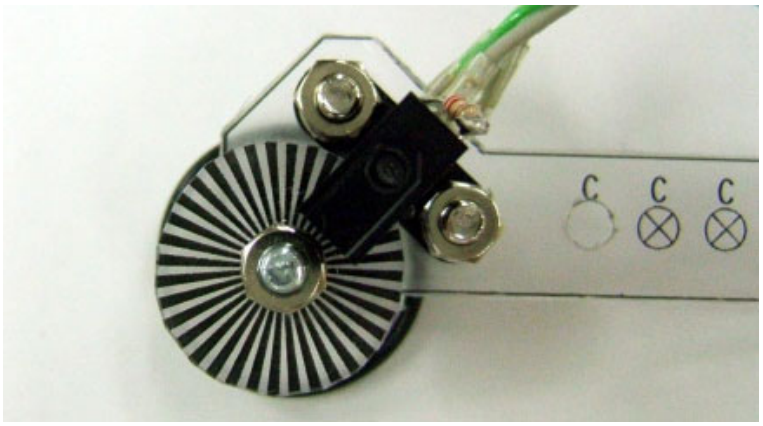


2.3.5　回転部分

シャープ(株)「GP1A51HRJ00F」を使った例を示します。
コース面に接地しているタイヤ(O リング)といっしょに円盤が回ります。



円盤は、下写真のように黒と透明が交互にあり、回転することによりフォトインタラプタの赤外 LED からの光が受光部分に届く、届かないを繰り返し、その信号がパルスとしてマイコンへ出力されます。



2.4 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。今回は例として、ロータリエンコーダキット Ver.2 を使用することとして計算します。ロータリエンコーダキット Ver.2 の仕様は下表のようになっています。

項目	内容
エンコーダの 1 回転のパルス数	72 パルス／回転　※
タイヤの半径（実寸）	10.5mm

この 2 項目が分かれば、プログラムすることができます。

※通常は黒部分の数(36 個)ですが、黒部分、透明部分の両方でパルスカウントすることができます。パルス数は 2 倍の 72 パルスとなります。プログラムの設定については後述します。

2.4.1 タイヤが 1 回転したときのパルス数の計算

タイヤの半径から、円周が分かります。
円周＝2πr＝2×10.5×3.14＝65.94mm

エンコーダは 72 パルス／回転なので、

65.94mm 進むと 72 パルス　　・・・(1)

となります。

2.4.2 1m進んだときのパルス数の計算

(1)より、1m 進んだときのパルス数は、
72 パルス:65.94mm＝xパルス:1000mm
x＝1091.9 パルス

1m (1000mm)進むと、1091.9 パルス　　・・・(2)

となります。

2.4.3 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 1091.9 パルス　　・・・(3)

となります。

2.4.4　秒速 1mで進んだとき、10ms間のパルス数の計算

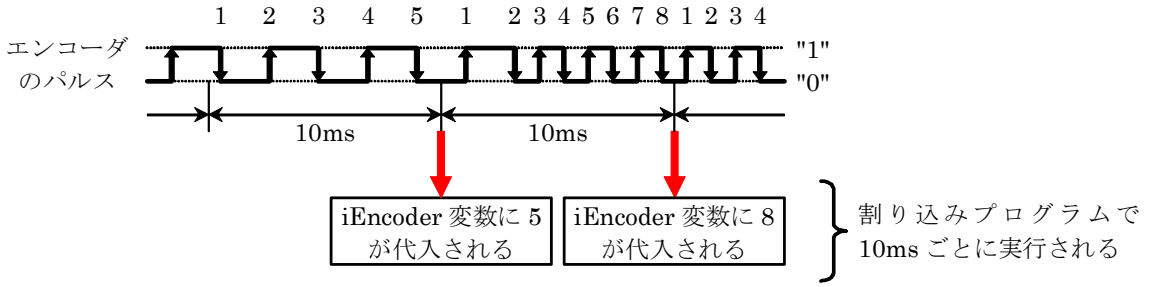
(3)より、
1 秒:1091.9 パルス＝0.01 秒:xパルス
x＝10.919≒10.92 パルス

1m/s の速さで進んだとき、10ms 間のパルス数は 10.92 パルス　　・・・(4)

となります。

2.4.5　プログラムで速度を検出する

今回のエンコーダを使ったプログラムでは、「**iEncoder**」という変数に 10ms 間のエンコーダパルス数が 10ms ごとに代入されます(下図)。プログラムについて、詳しくは後述します。



(4)より、1m/s で進んだとき、10ms 間のパルス数は 10.92 パルスです。要は、iEncoder 変数の値が 11 のとき(この変数は整数型なので四捨五入した値にします)、マイコンカーが秒速 1m/s で走っていると言うことです。
よって、iEncoder 変数の値をチェックすることにより、スピード制御することができます。

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
}
```

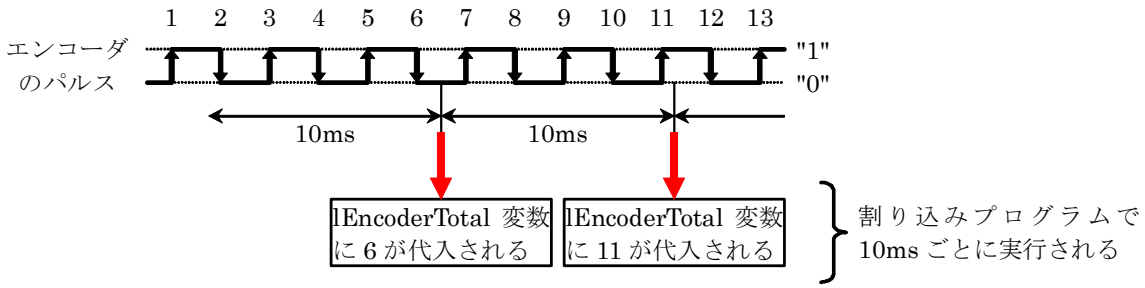
プログラムで記述します。「現在の速度」部分が、「iEncoder」になります。
今回のエンコーダは秒速 1m/s で 10.92 パルスなので、2m/s は、
秒速 2m/s のパルス数　＝　秒速 1m/s のパルス数×2
　　＝　10.92×2
　　＝　21.84
　　≒　22　　※iEncoder 変数は整数型なので四捨五入して整数にします

プログラムは、下記のようになります。

```
if( iEncoder >= 22 ) {
    speed( 0, 0 );
} else {
    speed( 70, 70 );
}
```

2.4.6 プログラムで距離を検出する

今回のエンコーダを使ったプログラムでは、「**lEncoderTotal**」という変数にエンコーダパルス数の合計値が 10ms ごとに代入されます(プログラムについて、詳しくは後述します)。



(2)より、1m 進んだときのパルス数は 1091.9 パルスです。要は、lEncoderTotal 変数の値が 1092 のとき(この変数は整数型なので四捨五入した値にします)、マイコンカーが 1m 走った(動いた)と言うことです。
よって、lEncoderTotal 変数の値をチェックすることにより、走行距離を知ることができます。

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」部分が、「lEncoderTotal」になります。
今回のエンコーダは秒速 1m 進むとで 1091.9 パルスなので、10m は、
距離 10m のパルス数　＝ 1m のパルス数×10
　　＝ 1091.9×10
　　＝ 10919

プログラムは、下記のようになります。

```
if( lEncoderTotal >= 10919 ) {
    speed( 0, 0 );
} else {
    speed( 100, 100 );
}
```

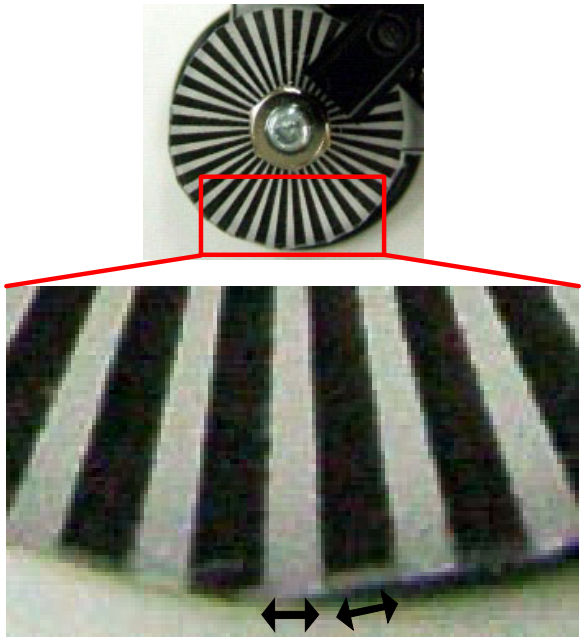
2.5　自分のマイコンカーのパルス数とスピード（距離）の関係

自分のマイコンカーのロータリエンコーダに関わる値を計算しておきましょう。

ロータリエンコーダのタイヤの半径	mm **(A)**
1回転のパルス数（ロータリエンコーダ Ver.2 は 72） ※標準は立ち上がり、立ち下がりでカウントする設定です。	パルス **(B)**
円周＝2π×(A)	mm **(C)**
1000mm 進んだときのパルス数は、 1000：x＝(C)：(B)　∴x＝1000×(B)÷(C)	パルス **(D)**
100mm 進んだときのパルス数は、 (E)＝(D)×0.1　※四捨五入した整数	パルス **(E)**
1m/s で進んだとき、10ms 間のパルス数は、 (F)＝(D)×0.01　※四捨五入した整数	パルス **(F)**
2m/s で進んだとき、10ms 間のパルス数は、 (G)＝(D)×0.02　※四捨五入した整数	パルス **(G)**

(A)～**(G)**の値は、後でプログラム修正時に使用します。

※立ち上がり、立ち下がりの両方でカウントする場合、黒い部分と透明部分の間隔が同じである必要があります。
間隔が違う場合は、両方でカウントはできません（下写真）。



同じ間隔なら両方でカウントにできます

詳しくは、「**4.4.5　円盤の黒、透明（白）の間隔が違うとき**」を参照してください。

3. サンプルプログラム

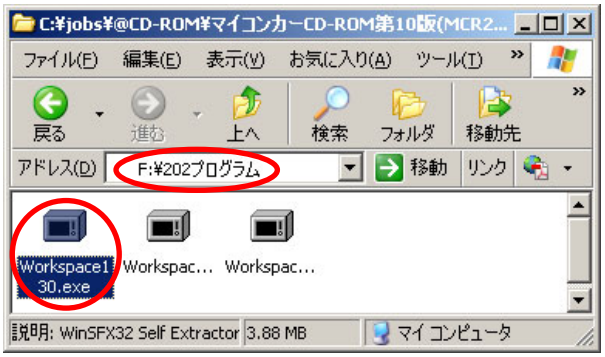
3.1 ルネサス統合開発環境

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　導入編」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。

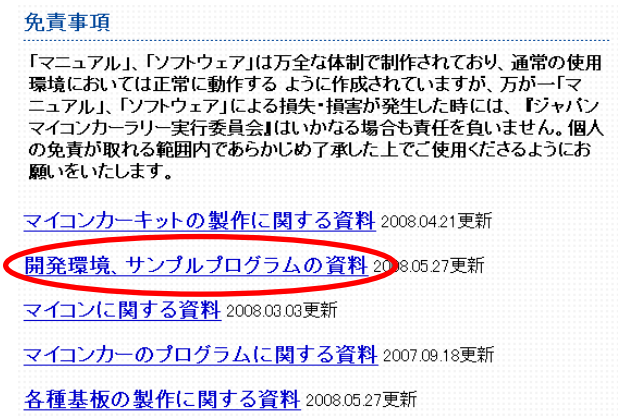
3.2.1 CDからソフトを取得する



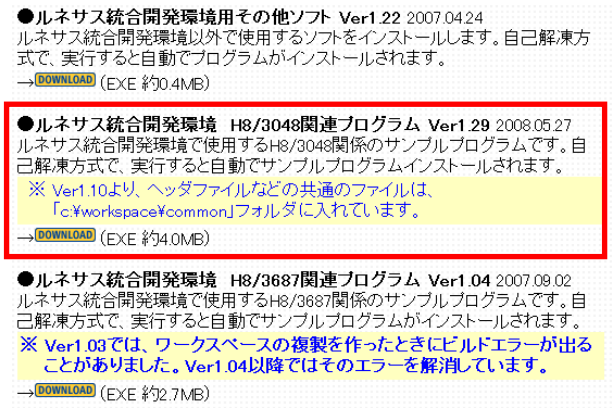
2007 年以降の講習会 CD がある場合、「CD ドライブ→202 プログラム」フォルダにある、「Workspace130.exe」を実行します。数字の 130 は、バージョンにより異なります。

3.2.2 ホームページからソフトを取得する

1.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。

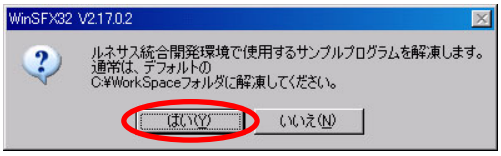


2.「開発環境、サンプルプログラムの資料」をクリックします。

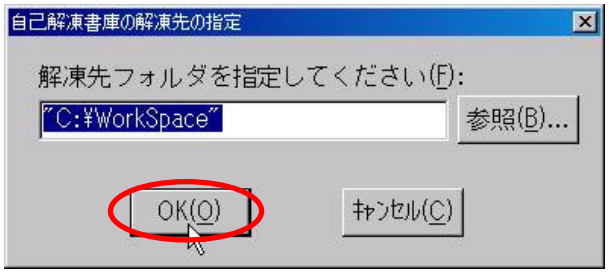


3.「ルネサス統合開発環境　H8/3048 関連プログラム」をダウンロードします。

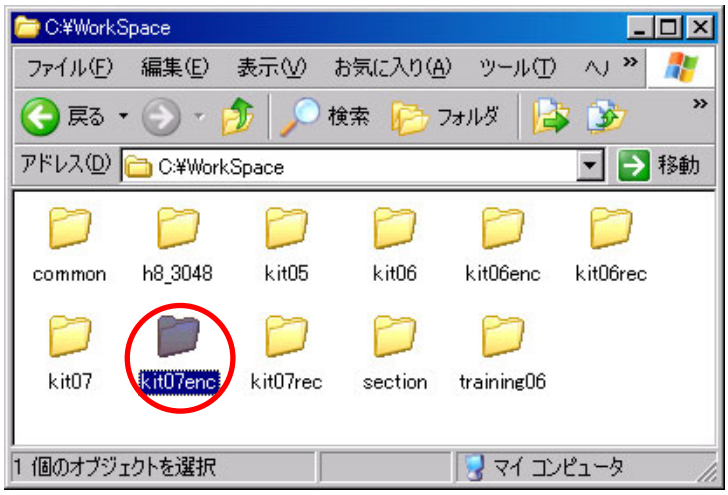
3.2.3　インストール



1.CD またはダウンロードした「Workspace130.exe」を実行します。はいをクリックします。



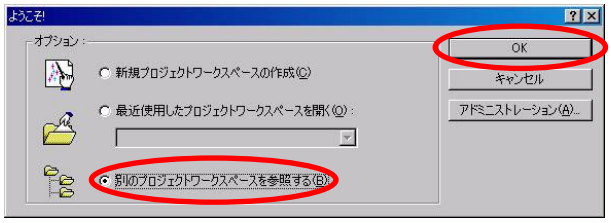
2.ファイルの解凍先を選択します。OK をクリックします。**このフォルダは変更できません。**



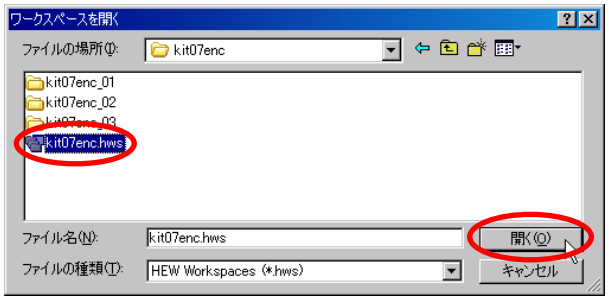
3. 解凍が終わったら、エクスプローラで「C ドライブ→Workspace」フォルダが開かれます。複数のフォルダがあります。今回使用するのは、「**kit07enc**」です。

3.3　ワーススペース「kit07enc」を開く

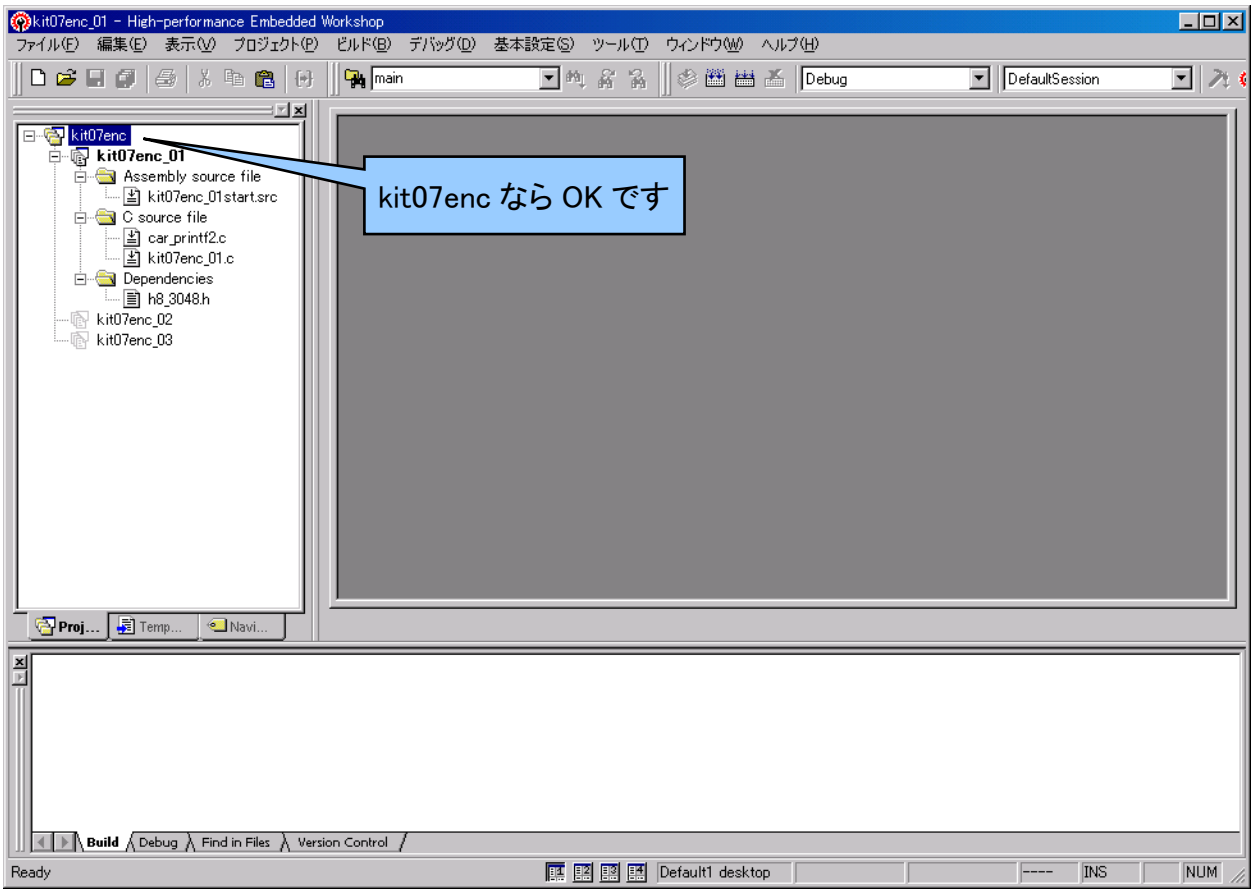
1. ルネサス統合開発環境を実行します。



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。

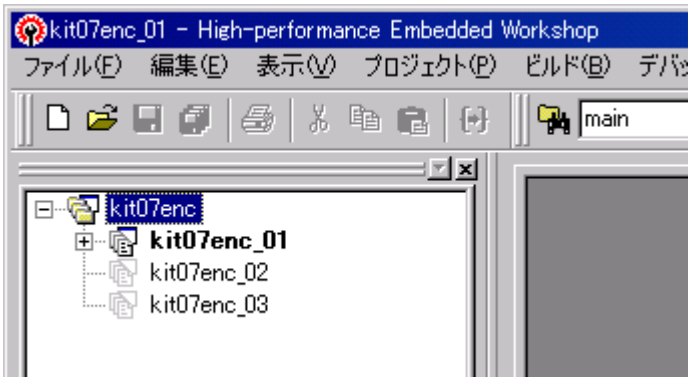


3. Cドライブ→Workspace→kit07enc の「kit07enc.hws」を選択、開く をクリックします。



4. kit07enc 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「kit07enc」には、3 つのプロジェクトが登録されています。

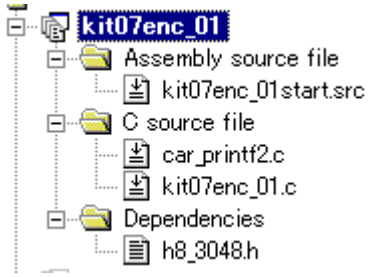
プロジェクト名	内容
kit07enc_01	標準走行プログラム「kit07.c」を改造して、エンコーダのパルスをカウントできるようにします。このプログラムは、エンコーダのパルスカウントができるように改造しただけで、マイコンカーのスピード制御は行っていません。プログラムの説明用です。
kit07enc_02	速度の調整を行うプログラムです。スピードが速ければマイコンカーを減速させる、遅ければ加速させるなどの制御を行うことができます。
kit07enc_03	距離の検出を行うプログラムです。例えば、クロスライン検出後、2 本目の横線を読み飛ばすために、10cm 進ませなさい、50m 進んだら止めなさい、などの制御を行うことができます。

プログラムのスピードや距離の設定値は、ロータリエンコーダキット Ver.2（72 パルス/回転、半径 10.5mm のタイヤ）の値に設定しています。違うエンコーダを使用している場合は、スピードや距離の値を設定し直します。

4. プロジェクト「kit07enc_01」 kit07.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「kit07.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。このプロジェクトは、プログラムの説明用でマイコンカーのスピード制御は行っていません。

4.1 プロジェクトの構成



・kit07enc_01start.src
・kit07enc_01.c
・car_printf2.c
の 3 ファイルあります。
h8_3048.h は kit07enc_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「kit07enc_01.c」      */
 3 :  /*                          2007.05 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kit07.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 :  kit07enc_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 :                                          /* φ/8で使用する場合、      */
27 :                                          /* φ/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 ×：マスクあり(無効)  ○：マスク無し(有効) */
39 :  #define         MASK2_2         0x66    /* ××○○××○○×          */
40 :  #define         MASK2_0         0x60    /* ××○○××××××          */
41 :  #define         MASK0_2         0x06    /* ××××××○○×          */
42 :  #define         MASK3_3         0xe7    /* ○○○××○○○          */
43 :  #define         MASK0_3         0x07    /* ××××××○○○          */
44 :  #define         MASK3_0         0xe0    /* ○○○××××××          */
45 :  #define         MASK4_0         0xf0    /* ○○○○××××××          */
46 :  #define         MASK0_4         0x0f    /* ××××○○○○          */
47 :  #define         MASK4_4         0xff    /* ○○○○○○○○          */
48 :
49 :  /*====================================*/
50 :  /* プロトタイプ宣言                   */
```

```
 51 :  /*====================================*/
 52 :  void init( void );
 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            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              */
 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時の処理
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時の処理
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時の処理
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 :          if( pushsw_get() ) {
125 :              pattern = 1;
126 :              cnt1 = 0;
127 :              break;
128 :          }
129 :          if( cnt1 < 100 ) {             /* LED点滅処理             */
130 :              led_out( PADR & 0x01 );
131 :          } else if( cnt1 < 200 ) {
132 :              led_out( PADR & 0x01 | 2 );
133 :          } else {
134 :              cnt1 = 0;
135 :          }
136 :          break;
```

中略

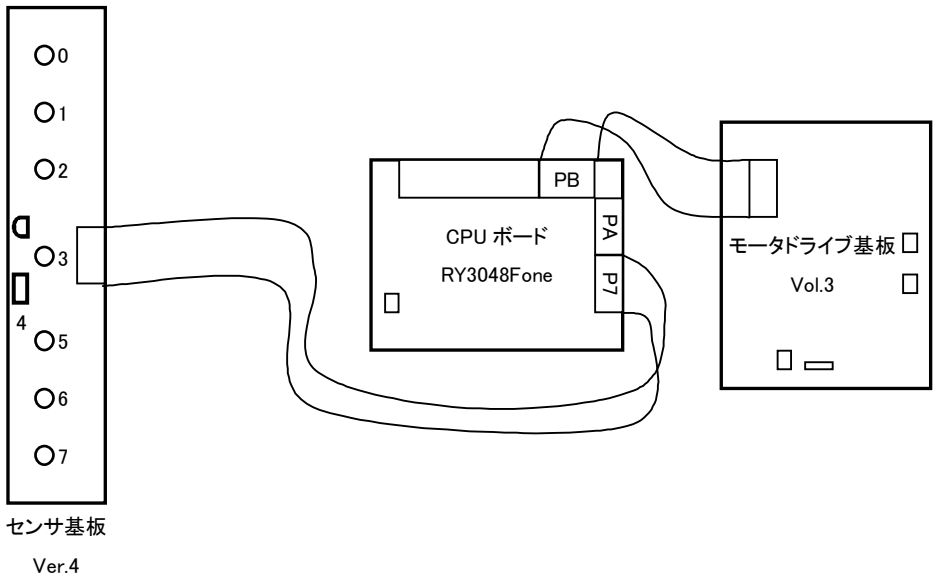
```
498 :  /************************************************************************/
499 :  /* H8/3048F-ONE 内蔵周辺機能　初期化                                    */
500 :  /************************************************************************/
501 :  void init( void )
502 :  {
503 :      /* I/Oポートの入出力設定 */
504 :      P1DDR = 0xff;
505 :      P2DDR = 0xff;
506 :      P3DDR = 0xff;
507 :      P4DDR = 0xff;
508 :      P5DDR = 0xff;
509 :      P6DDR = 0xf0;                       /* CPU基板上のDIP SW        */
510 :      P8DDR = 0xff;
511 :      P9DDR = 0xf7;                       /* 通信ポート               */
512 :      PADDR = 0xfe;                       /* 0:Encoder                */
513 :      PBDR  = 0xc0;
514 :      PBDDR = 0xfe;                       /* モータドライブ基板Vol.3  */
515 :      /* ※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     */
516 :
517 :      /* ITU0 1msごとの割り込み */
518 :      ITU0_TCR = 0x23;
519 :      ITU0_GRA = TIMER_CYCLE;
520 :      ITU0_IER = 0x01;
521 :
522 :      /* ITU2 パルス入力の設定 */
523 :      ITU2_TCR = 0x14;                    /* PA0端子のパルスでカウント*/
524 :
525 :      /* ITU3,4 リセット同期PWMモード 左右モータ、サーボ用 */
526 :      ITU3_TCR = 0x23;
527 :      ITU_FCR  = 0x3e;
528 :      ITU3_GRA = PWM_CYCLE;               /* 周期の設定               */
529 :      ITU3_GRB = ITU3_BRB = 0;            /* 左モータのPWM設定        */
530 :      ITU4_GRA = ITU4_BRA = 0;            /* 右モータのPWM設定        */
531 :      ITU4_GRB = ITU4_BRB = SERVO_CENTER; /* サーボのPWM設定          */
532 :      ITU_TOER = 0x38;
533 :
534 :      /* ITUのカウントスタート */
535 :      ITU_STR = 0x0d;
536 :  }
537 :
538 :  /************************************************************************/
539 :  /* ITU0 割り込み処理                                                    */
540 :  /************************************************************************/
541 :  #pragma interrupt( interrupt_timer0 )
542 :  void interrupt_timer0( void )
543 :  {
544 :      unsigned int i;
545 :
546 :      ITU0_TSR &= 0xfe;                   /* フラグクリア             */
547 :      cnt0++;
548 :      cnt1++;
549 :
550 :      /* エンコーダ関連 */
551 :      iTimer10++;
552 :      if( iTimer10 >= 10 ) {
553 :          iTimer10 = 0;
554 :          i = ITU2_CNT;
555 :          iEncoder       = i - uEncoderBuff;
556 :          lEncoderTotal += iEncoder;
557 :          if( iEncoder > iEncoderMax )
558 :                      iEncoderMax = iEncoder;
559 :          uEncoderBuff = i;
560 :      }
561 :  }
```

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kit07 の接続の確認

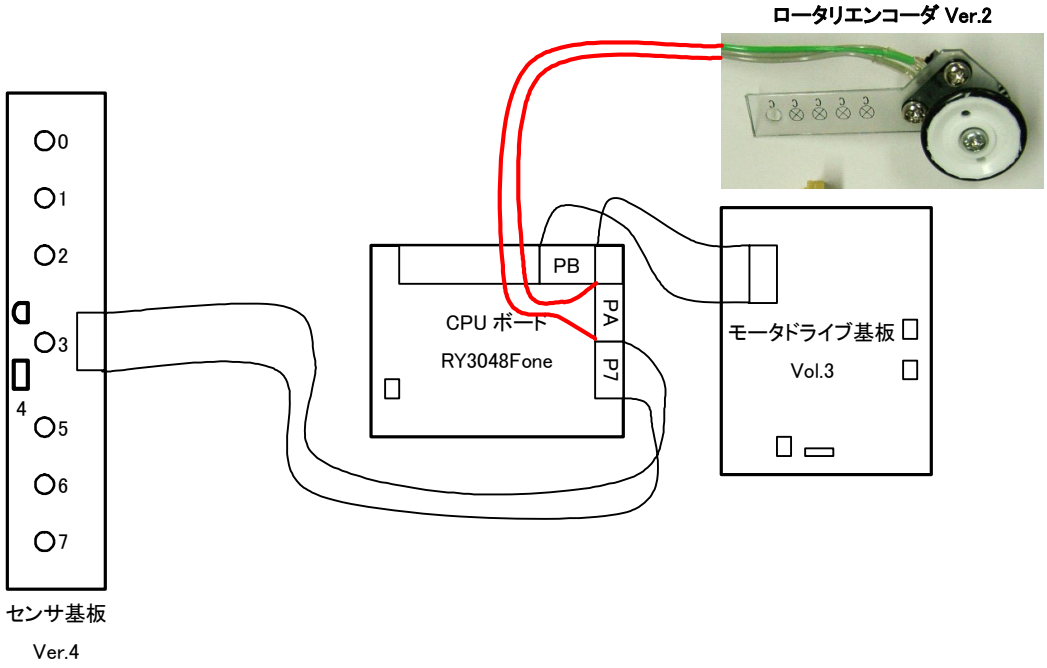
kit07.c のポートと各基板の接続は、下記のようになっています。
・ポート 7…センサ基板 Ver.4 と接続しています。
・ポート B…モータドライブ基板 Vol.3 と接続しています。



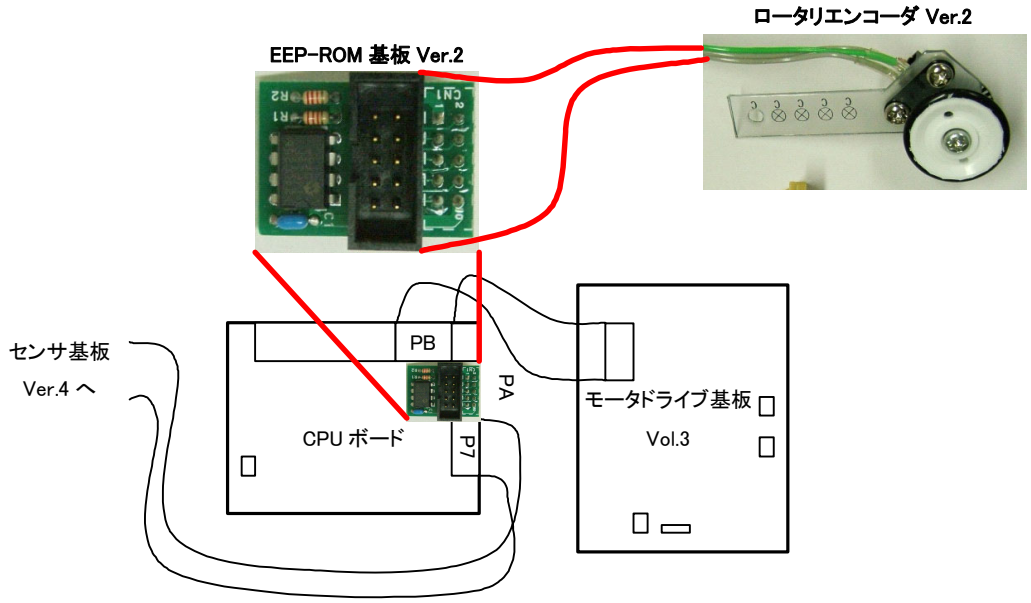
ポート A は使用していません。ここにロータリエンコーダを接続します。

4.3.2　ロータリエンコーダを接続

ポート A の bit0 にロータリエンコーダ Ver.2 を接続します。



EEP-ROM 基板 Ver.2 を接続するときは、EEP-ROM 基板 Ver.2 上のコネクタにロータリエンコーダ Ver.2 を接続します（下図）。



※EEP-ROM 基板については、データ解析実習マニュアルを参照してください。

ポートＡの接続は下記のようになります。

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		出力
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	出力	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	1	1	1	**0**

16 進数に直すと、1111 1110→0xfe となります。

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。

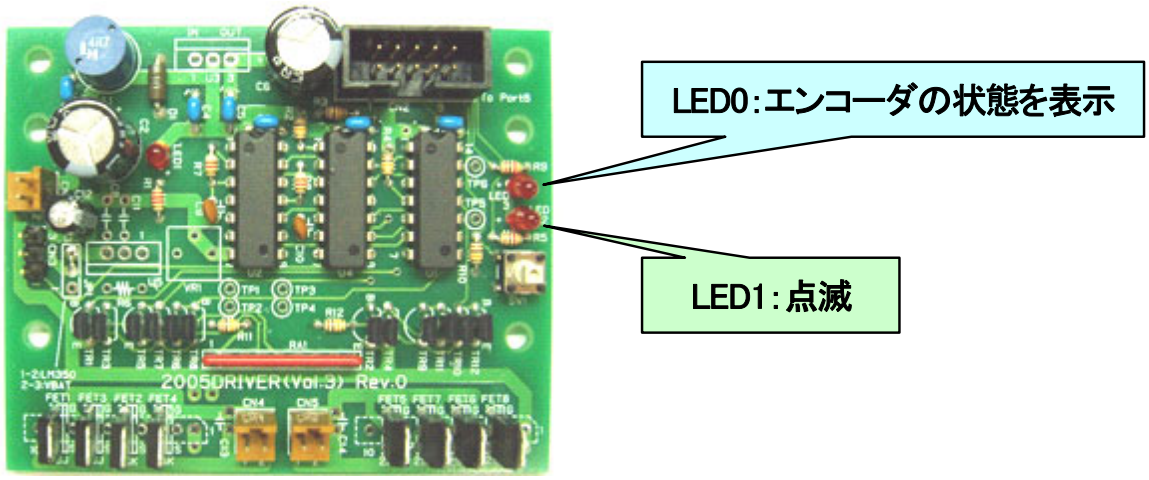
変数名	意味	内容
iTimer10	10ms タイマ	エンコーダ値の更新は、interrupt_timer0 関数内で行います。interrupt_timer0 関数は 1ms ごとに実行されますが、エンコーダ処理は 10ms ごとです。そこで、この変数を 1ms ごとに＋1して 10 になったかどうかチェックしています。
lEncoderTotal	エンコーダ積算値	スタートしてからのエンコーダパルスの積算値を保存しています。long 型変数なので、21 億回までカウントできます。
iEncoderMax	10ms ごとの最大値	10ms ごとに更新されるエンコーダ値の最大値を保存しています。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:エンコーダの状態をモータドライブ基板のLEDへ出力

```
122 :      case 0:
123 :          /* スイッチ入力待ち */
124 :          if( pushsw_get() ) {
125 :              pattern = 1;
126 :              cnt1 = 0;
127 :              break;
128 :          }
129 :          if( cnt1 < 100 ) {              /* LED 点滅処理             */
130 :              led_out( PADR & 0x01 );
131 :          } else if( cnt1 < 200 ) {
132 :              led_out( PADR & 0x01 | 2 );
133 :          } else {
134 :              cnt1 = 0;
135 :          }
136 :          break;
```

モータドライブ基板の LED0 にエンコーダの状態を出力します。LED1 は 0.1 秒ごとに点滅させてスイッチ入力待ちであることを知らせます。



4.4.3 入出力設定の変更

```
498 :  /************************************************************************/
499 :  /* H8/3048F-ONE 内蔵周辺機能　初期化                                    */
500 :  /************************************************************************/
501 :  void init( void )
502 :  {
503 :      /* I/O ポートの入出力設定 */
504 :      P1DDR = 0xff;
505 :      P2DDR = 0xff;
506 :      P3DDR = 0xff;
507 :      P4DDR = 0xff;
508 :      P5DDR = 0xff;
509 :      P6DDR = 0xf0;                    /* CPU 基板上の DIP SW        */
510 :      P8DDR = 0xff;
511 :      P9DDR = 0xf7;                    /* 通信ポート                 */
512 :      PADDR = 0xfe;                    /* 0:Encoder                  */
513 :      PBDR  = 0xc0;
514 :      PBDDR = 0xfe;                    /* モータドライブ基板 Vol.3   */
515 :      /* ※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のパルス入力になったので、0xff から 0xfe へ変更します。**

4.4.4 外部パルス入力設定

```
 522 :      /* ITU2 パルス入力の設定 */
 523 :      ITU2_TCR = 0x14;                    /* PA0 端子のパルスでカウント*/
中略
 534 :      /* ITU のカウントスタート */
 535 :      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	1	0	1	0	0
16 進数:	1				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／インプットキャプチャで CNT をクリア
1	0	GRB のコンペアマッチ／インプットキャプチャで CNT をクリア
1	1	同期クリア

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／立ち下がりの両エッジでカウント
1	1	立ち上がり／立ち下がりの両エッジでカウント

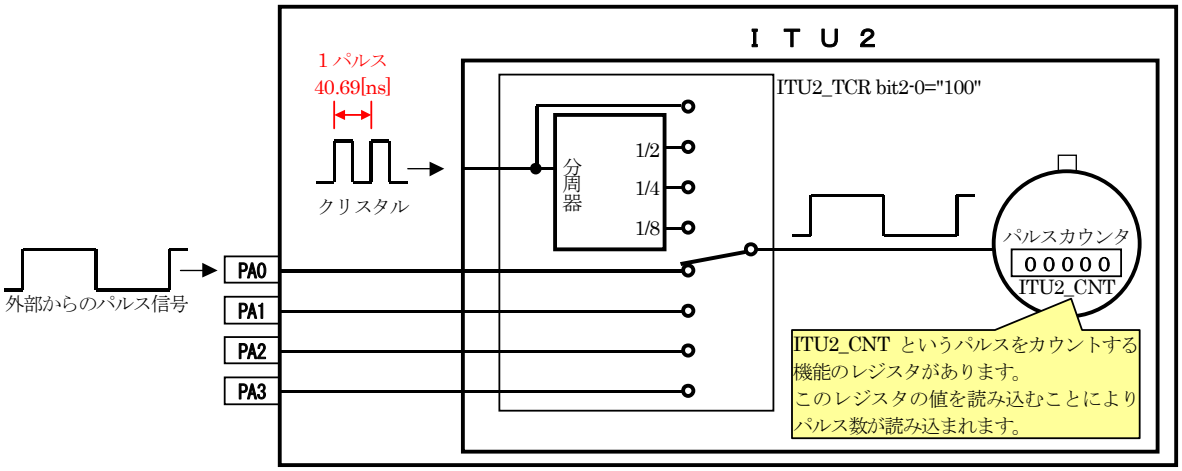
外部パルスの立ち上がり、立ち下がりで ITU2_CNT が+1 します。

・ビット 2～0：タイマプリスケーラ 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：φでカウント
0	0	1	内部クロック：φ／2でカウント
0	1	0	内部クロック：φ／4でカウント
0	1	1	内部クロック：φ／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を数えます。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

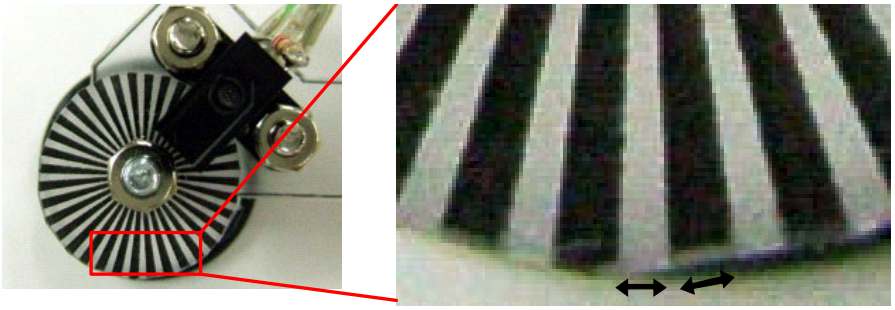
ITU は下記のように使用します。

・ITU0…1ms 割り込み　・ITU1…未使用　**・ITU2…パルスカウント**　・ITU3…リセット同期 PWM モード

設定値は 0x0d となります。

4.4.5 円盤の黒、透明(白)の間隔が違うとき

立ち上がり／立ち下がりの両エッジでカウントの設定にする場合は、円盤の黒い部分と透明部分の間隔が同じである必要があります(下写真)。



同じ間隔なら 0x14 にできます

間隔が違う場合は、立ち上がりのみにします。

```
522 :       /* ITU2 パルス入力の設定 */
523 :       ITU2_TCR = 0x04;                    /* PA0 端子のパルスでカウント*/
```

4.4.6 ITU0 割り込み処理

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

551 行…iTimer10 変数を増加させます。

552 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、実行回数を数えて 10 回目なら次の行に移りエンコーダ処理を行います。それ以下なら 560 行へ移りエンコーダ処理をしません。

553 行…iTimer10 変数を 0 にして、実行回数を数え直します。

554 行…現在のカウント値 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。

555 行…最新の 10ms 間のエンコーダのカウント数を計算しています。計算は、

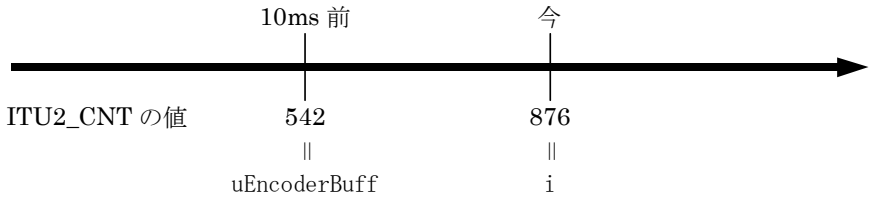
10ms 間のエンコーダのカウント数 = i - uEncoderBuff

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、

10ms 間のエンコーダのカウント数 = 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でます。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

556 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。

557 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。

558 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。

559 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

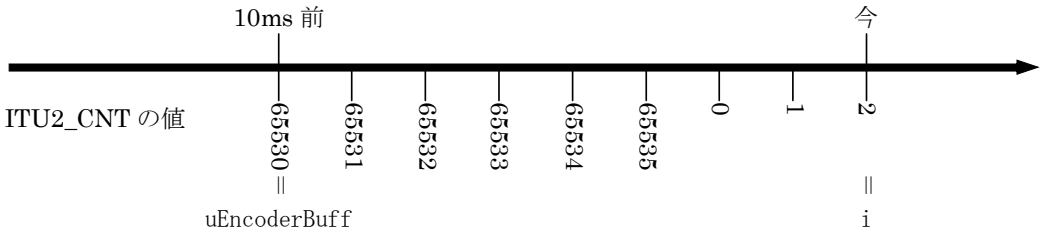
このプログラムでは割り込み内にあるため、1ms ごとに実行されます。そこで、

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

で、回数を数えて 10 回目で実行、ようは 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、符号無しの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のには、

（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、

0x0002－0xfffa＝0xffff0008

ただし、**計算結果も符号無し 16 ビット幅**なので、

0x0002－0xfffa＝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で、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か。

```
#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 をクリアしています。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。

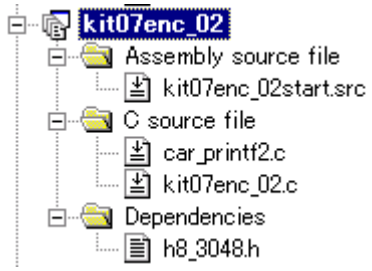
```
    if( iTimer10 >= 10 ) {
        iTimer10 = 0;
        iEncoder = ITU2_CNT;            /* カウンタの値を iEncoder にコピーして */
        ここでパルスが入力されて ITU2_CNT の値が 1 つ増えた
        ITU2_CNT = 0;                   /* カウンタの値をクリア */
        中略
    }
```

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 1 カウント分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「kit07enc_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、クロスライン検出後、右ハーフライン検出後、左ハーフライン検出後の速度を検出して、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 プロジェクトの構成



・kit07enc_02start.src
・kit07enc_02.c
・car_printf2.c
の 3 ファイルあります。
h8_3048.h は kit07enc_02.c、car_printf2.c でインクルードされているファイルです。

5.2 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。

前略

```
233 :      case 12:
234 :          /* 右へ大曲げの終わりのチェック */
235 :          if( check_crossline() ) {       /* 大曲げ中もクロスラインチェック */
236 :              pattern = 21;
237 :              break;
238 :          }
239 :          if( check_rightline() ) {       /* 右ハーフラインチェック   */
240 :              pattern = 51;
241 :              break;
242 :          }
243 :          if( check_leftline() ) {        /* 左ハーフラインチェック   */
244 :              pattern = 61;
245 :              break;
246 :          }
247 :          if( iEncoder >= 11 ) {
248 :              speed2( 0 ,0 );
249 :          } else {
250 :              speed2( 60 ,37 );
251 :          }
252 :          if( sensor_inp(MASK3_3) == 0x06 ) {
253 :              pattern = 11;
254 :          }
255 :          break;
256 :
257 :      case 13:
258 :          /* 左へ大曲げの終わりのチェック */
259 :          if( check_crossline() ) {       /* 大曲げ中もクロスラインチェック */
260 :              pattern = 21;
261 :              break;
262 :          }
263 :          if( check_rightline() ) {       /* 右ハーフラインチェック   */
264 :              pattern = 51;
265 :              break;
266 :          }
267 :          if( check_leftline() ) {        /* 左ハーフラインチェック   */
268 :              pattern = 61;
269 :              break;
270 :          }
271 :          if( iEncoder >= 11 ) {
272 :              speed2( 0 ,0 );
273 :          } else {
274 :              speed2( 37 ,60 );
275 :          }
276 :          if( sensor_inp(MASK3_3) == 0x60 ) {
277 :              pattern = 11;
278 :          }
279 :          break;
```

中略

```
298 :      case 23:
299 :          /* クロスライン後のトレース、クランク検出 */
300 :          if( sensor_inp(MASK4_4)==0xf8 ) {
301 :              /* 左クランクと判断→左クランククリア処理へ */
302 :              led_out( 0x1 );
303 :              handle( -38 );
304 :              speed( 10 ,50 );
305 :              pattern = 31;
306 :              cnt1 = 0;
307 :              break;
308 :          }
309 :          if( sensor_inp(MASK4_4)==0x1f ) {
310 :              /* 右クランクと判断→右クランククリア処理へ */
311 :              led_out( 0x2 );
312 :              handle( 38 );
313 :              speed( 50 ,10 );
314 :              pattern = 41;
315 :              cnt1 = 0;
316 :              break;
317 :          }
318 :          if( iEncoder >= 11 ) {      /* クロスライン後のスピード制御 */
319 :              speed2( 0 ,0 );
320 :          } else {
321 :              speed2( 70 ,70 );
322 :          }
323 :          switch( sensor_inp(MASK3_3) ) {
324 :              case 0x00:
325 :                  /* センタ→まっすぐ */
326 :                  handle( 0 );
327 :                  break;
328 :              case 0x04:
329 :              case 0x06:
330 :              case 0x07:
331 :              case 0x03:
332 :                  /* 左寄り→右曲げ */
333 :                  handle( 8 );
334 :                  break;
335 :              case 0x20:
336 :              case 0x60:
337 :              case 0xe0:
338 :              case 0xc0:
339 :                  /* 右寄り→左曲げ */
340 :                  handle( -8 );
341 :                  break;
342 :          }
343 :          break;
```

中略

```
396 :      case 53:
397 :          /* 右ハーフライン後のトレース、レーンチェンジ */
398 :          if( sensor_inp(MASK4_4) == 0x00 ) {
399 :              handle( 15 );
400 :              speed( 40 ,31 );
401 :              pattern = 54;
402 :              cnt1 = 0;
403 :              break;
404 :          }
405 :          if( iEncoder >= 11 ) {      /* ハーフラインライン後のスピード制御 */
406 :              speed2( 0 ,0 );
407 :          } else {
408 :              speed2( 70 ,70 );
409 :          }
410 :          switch( sensor_inp(MASK3_3) ) {
411 :              case 0x00:
412 :                  /* センタ→まっすぐ */
413 :                  handle( 0 );
414 :                  break;
415 :              case 0x04:
416 :              case 0x06:
417 :              case 0x07:
418 :              case 0x03:
419 :                  /* 左寄り→右曲げ */
420 :                  handle( 8 );
421 :                  break;
422 :              case 0x20:
423 :              case 0x60:
424 :              case 0xe0:
425 :              case 0xc0:
426 :                  /* 右寄り→左曲げ */
427 :                  handle( -8 );
428 :                  break;
429 :              default:
430 :                  break;
431 :          }
432 :          break;
```

中略

```
460 :       case 63:
461 :           /* 左ハーフライン後のトレース、レーンチェンジ */
462 :           if( sensor_inp(MASK4_4) == 0x00 ) {
463 :               handle( -15 );
464 :               speed( 31 ,40 );
465 :               pattern = 64;
466 :               cnt1 = 0;
467 :               break;
468 :           }
469 :           if( iEncoder >= 11 ) {      /* ハーフラインライン後のスピード制御 */
470 :               speed2( 0 ,0 );
471 :           } else {
472 :               speed2( 70 ,70 );
473 :           }
474 :           switch( sensor_inp(MASK3_3) ) {
475 :               case 0x00:
476 :                   /* センタ→まっすぐ */
477 :                   handle( 0 );
478 :                   break;
479 :               case 0x04:
480 :               case 0x06:
481 :               case 0x07:
482 :               case 0x03:
483 :                   /* 左寄り→右曲げ */
484 :                   handle( 8 );
485 :                   break;
486 :               case 0x20:
487 :               case 0x60:
488 :               case 0xe0:
489 :               case 0xc0:
490 :                   /* 右寄り→左曲げ */
491 :                   handle( -8 );
492 :                   break;
493 :               default:
494 :                   break;
495 :           }
496 :           break;
```

中略

```
752 :   /************************************************************************/
753 :   /* 速度制御2                                                            */
754 :   /* 引数    左モータ:-100～100 , 右モータ:-100～100                      */
755 :   /*         0で停止、100で正転100%、-100で逆転100%                       */
756 :   /*         ディップスイッチは関係なし                                   */
757 :   /************************************************************************/
758 :   void speed2( int accele_l, int accele_r )
759 :   {
760 :       unsigned long   speed_max;
761 :
762 :       speed_max = PWM_CYCLE - 1;
763 :
764 :       /* 左モータ */
765 :       if( accele_l >= 0 ) {
766 :           PBDR &= 0xfb;
767 :           ITU3_BRB = speed_max * accele_l / 100;
768 :       } else {
769 :           PBDR |= 0x04;
770 :           accele_l = -accele_l;
771 :           ITU3_BRB = speed_max * accele_l / 100;
772 :       }
773 :
774 :       /* 右モータ */
775 :       if( accele_r >= 0 ) {
776 :           PBDR &= 0xf7;
777 :           ITU4_BRA = speed_max * accele_r / 100;
778 :       } else {
779 :           PBDR |= 0x08;
780 :           accele_r = -accele_r;
781 :           ITU4_BRA = speed_max * accele_r / 100;
782 :       }
783 :   }
```

以下、略

5.3 プログラムの解説

5.3.1 パターン 12 右大曲げときの処理

```
233 :       case 12:
234 :           /* 右へ大曲げの終わりのチェック */
235 :           if( check_crossline() ) {       /* 大曲げ中もクロスラインチェック */
236 :               pattern = 21;
237 :               break;
238 :           }
239 :           if( check_rightline() ) {       /* 右ハーフラインチェック    */
240 :               pattern = 51;
241 :               break;
242 :           }
243 :           if( check_leftline() ) {        /* 左ハーフラインチェック    */
244 :               pattern = 61;
245 :               break;
246 :           }
247 :           if( iEncoder >= 11 ) {
248 :               speed2( 0 ,0 );
249 :           } else {
250 :               speed2( 60 ,37 );
251 :           }
252 :           if( sensor_inp(MASK3_3) == 0x06 ) {
253 :               pattern = 11;
254 :           }
255 :           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 37%にします。

「**2.4 パルス数とスピード（距離）の関係**」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 11 パルス**」でした。ここでは現在のパルス値iEncoderが 11 以下か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、

10ms間のパルス数 ＝ 現在の速度 × 10.92
＝ 2[m/s] × 10.92
＝ 21.84
≒ 22　※小数点は使えないので四捨五入

iEncoder が 22 以上かどうかチェックすると、速度が 2m/s 以上かどう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しないように調整します。

5.3.2　speed2 関数

speed 関数を良く見ると…　speed2 関数？　2 が付いています。

```
752 :  /*********************************************************************/
753 :  /* 速度制御2                                                         */
754 :  /* 引数    左モータ:-100～100 ，右モータ:-100～100                   */
755 :  /*         0で停止、100で正転100%、-100で逆転100%                    */
756 :  /*         ディップスイッチは関係なし                                */
757 :  /*********************************************************************/
758 :  void speed2( int accele_l, int accele_r )
759 :  {
760 :      unsigned long   speed_max;
761 :
762 :      speed_max = PWM_CYCLE - 1;
763 :
764 :      /* 左モータ */
765 :      if( accele_l >= 0 ) {
766 :          PBDR &= 0xfb;
767 :          ITU3_BRB = speed_max * accele_l / 100;
768 :      } else {
769 :          PBDR |= 0x04;
770 :          accele_l = -accele_l;
771 :          ITU3_BRB = speed_max * accele_l / 100;
772 :      }
773 :
774 :      /* 右モータ */
775 :      if( accele_r >= 0 ) {
776 :          PBDR &= 0xf7;
777 :          ITU4_BRA = speed_max * accele_r / 100;
778 :      } else {
779 :          PBDR |= 0x08;
780 :          accele_r = -accele_r;
781 :          ITU4_BRA = speed_max * accele_r / 100;
782 :      }
783 :  }
```

speed 関数は、

実際にモータに出力される PWM 値＝speed 関数の引数の割合×ディップスイッチの割合

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。speed2 関数は、

実際にモータに出力される PWM 値＝speed 関数の引数の割合

となります。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。

5.3.3 パターン 13 左大曲げときの処理

```
257 :      case 13:
258 :          /* 左へ大曲げの終わりのチェック */
259 :          if( check_crossline() ) {       /* 大曲げ中もクロスラインチェック */
260 :              pattern = 21;
261 :              break;
262 :          }
263 :          if( check_rightline() ) {       /* 右ハーフラインチェック    */
264 :              pattern = 51;
265 :              break;
266 :          }
267 :          if( check_leftline() ) {        /* 左ハーフラインチェック    */
268 :              pattern = 61;
269 :              break;
270 :          }
271 :          if( iEncoder >= 11 ) {
272 :              speed2( 0 ,0 );
273 :          } else {
274 :              speed2( 37 ,60 );
275 :          }
276 :          if( sensor_inp(MASK3_3) == 0x60 ) {
277 :              pattern = 11;
278 :          }
279 :          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 37%、右 60%にします。こちらも speed2 関数を使用します。

5.3.4　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

```
298 :      case 23:
299 :          /* クロスライン後のトレース、クランク検出 */
300 :          if( sensor_inp(MASK4_4)==0xf8 ) {
301 :              /* 左クランクと判断→左クランククリア処理へ */
302 :              led_out( 0x1 );
303 :              handle( -38 );
304 :              speed( 10 ,50 );
305 :              pattern = 31;
306 :              cnt1 = 0;
307 :              break;
308 :          }
309 :          if( sensor_inp(MASK4_4)==0x1f ) {
310 :              /* 右クランクと判断→右クランククリア処理へ */
311 :              led_out( 0x2 );
312 :              handle( 38 );
313 :              speed( 50 ,10 );
314 :              pattern = 41;
315 :              cnt1 = 0;
316 :              break;
317 :          }
318 :          if( iEncoder >= 11 ) {      /* クロスライン後のスピード制御 */
319 :              speed2( 0 ,0 );
320 :          } else {
321 :              speed2( 70 ,70 );
322 :          }
323 :          switch( sensor_inp(MASK3_3) ) {
324 :              case 0x00:
325 :                  /* センタ→まっすぐ */
326 :                  handle( 0 );
327 :                  break;
328 :              case 0x04:
329 :              case 0x06:
330 :              case 0x07:
331 :              case 0x03:
332 :                  /* 左寄り→右曲げ */
333 :                  handle( 8 );
334 :                  break;
335 :              case 0x20:
336 :              case 0x60:
337 :              case 0xe0:
338 :              case 0xc0:
339 :                  /* 右寄り→左曲げ */
340 :                  handle( -8 );
341 :                  break;
342 :          }
343 :          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 1m 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 50cm～1m 後にクランクがあることと変更になりました。

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 1m 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しきれない場合は、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。

パターン 53、パターン 64 でのスピード調整も同様です。

5.4 プログラムの調整

このサンプルプログラムは、72 パルス／回転、エンコーダのタイヤ直径 21mm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。「**2.5 自分のマイコンカーのパルス数とスピード（距離）の関係**」を参照しながら変更してください。

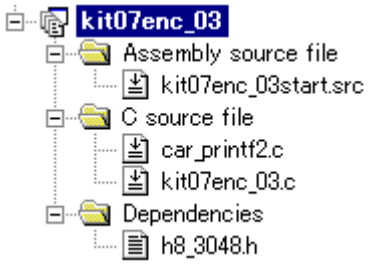
行番号	元の数値	変更後の数値
35	5000	それぞれのマイコンカーのサーボセンタ値にします。
247	11	右へ大曲げの終わりのチェック中のスピードを設定します。 1m/s にするなら、**(F)**の値にします。
271	11	左へ大曲げの終わりのチェック中のスピードを設定します。 1m/s にするなら、**(F)**の値にします。
303	-38	左クランクを曲がるときの角度です。 左最大角度を設定します。
312	38	右クランクを曲がるときの角度です。 右最大角度を設定します。
318	11	クロスラインを検出後、徐行して進むスピードを設定します。 1m/s にするなら、**(F)**の値にします。
405	11	右ハーフラインを検出後、徐行して進むスピードを設定します。 1m/s にするなら、**(F)**の値にします。2m/s にするなら、**(G)** の値にします。 最初は**(G)**の値にして、脱輪するようなら**(F)**の値にします。
469	11	左ハーフラインを検出後、徐行して進むスピードを設定します。 1m/s にするなら、**(F)**の値にします。2m/s にするなら、**(G)**の値にします。 最初は**(G)**の値にして、脱輪するようなら**(F)**の値にします。
540	0x14	立ち上がり、立ち下がりでカウントアップする設定です。立ち上がりのみでカウントアップする場合、「**0x04**」にします。

6. プロジェクト「kit07enc_03」 距離の検出(パターンの区分けを距離で行う)

kit07 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、2 本目のクロスラインが終わった直後と仮定しています。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によって位置が変わると言うことがありません。

6.1 プロジェクトの構成



・kit07enc_03start.src
・kit07enc_03.c
・car_printf2.c
の 3 ファイルあります。
h8_3048.h は kit07enc_03.c、car_printf2.c でインクルードされているファイルです。

6.2 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;         /* ライン検出時の積算値     */
```

中略

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

中略

```
283 :       case 21:
284 :           /* １本目のクロスライン検出時の処理 */
285 :           lEncoderLine = lEncoderTotal;
286 :           led_out( 0x3 );
287 :           handle( 0 );
288 :           speed( 0 ,0 );
289 :           pattern = 22;
290 :           cnt1 = 0;
291 :           break;
292 :
293 :       case 22:
294 :           /* ２本目を読み飛ばす */
295 :           if( lEncoderTotal-lEncoderLine >= 109 ) {   /* 約10cmたったか？ */
296 :               pattern = 23;
297 :               cnt1 = 0;
298 :           }
299 :           break;
```

中略

```
382 :       case 51:
383 :           /* １本目の右ハーフライン検出時の処理 */
384 :           lEncoderLine = lEncoderTotal;
385 :           led_out( 0x2 );
386 :           handle( 0 );
387 :           speed( 0 ,0 );
388 :           pattern = 52;
389 :           cnt1 = 0;
390 :           break;
391 :
392 :       case 52:
393 :           /* ２本目を読み飛ばす */
394 :           if( lEncoderTotal-lEncoderLine >= 109 ) {   /* 約10cmたったか？ */
395 :               pattern = 53;
396 :               cnt1 = 0;
397 :           }
398 :           break;
```

中略

```
447 :       case 61:
448 :           /* １本目の左ハーフライン検出時の処理 */
449 :           lEncoderLine = lEncoderTotal;
450 :           led_out( 0x1 );
451 :           handle( 0 );
452 :           speed( 0 ,0 );
453 :           pattern = 62;
454 :           cnt1 = 0;
455 :           break;
456 :
457 :       case 62:
458 :           /* ２本目を読み飛ばす */
459 :           if( lEncoderTotal-lEncoderLine >= 109 ) {   /* 約10cmたったか？ */
460 :               pattern = 63;
461 :               cnt1 = 0;
462 :           }
463 :           break;
```

以下、略

6.3 プログラムの解説

6.3.1 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;       /* ライン検出時の積算値     */
```

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瞬間の位置の積算値を記憶させておきます。

6.3.2 積算値のクリア

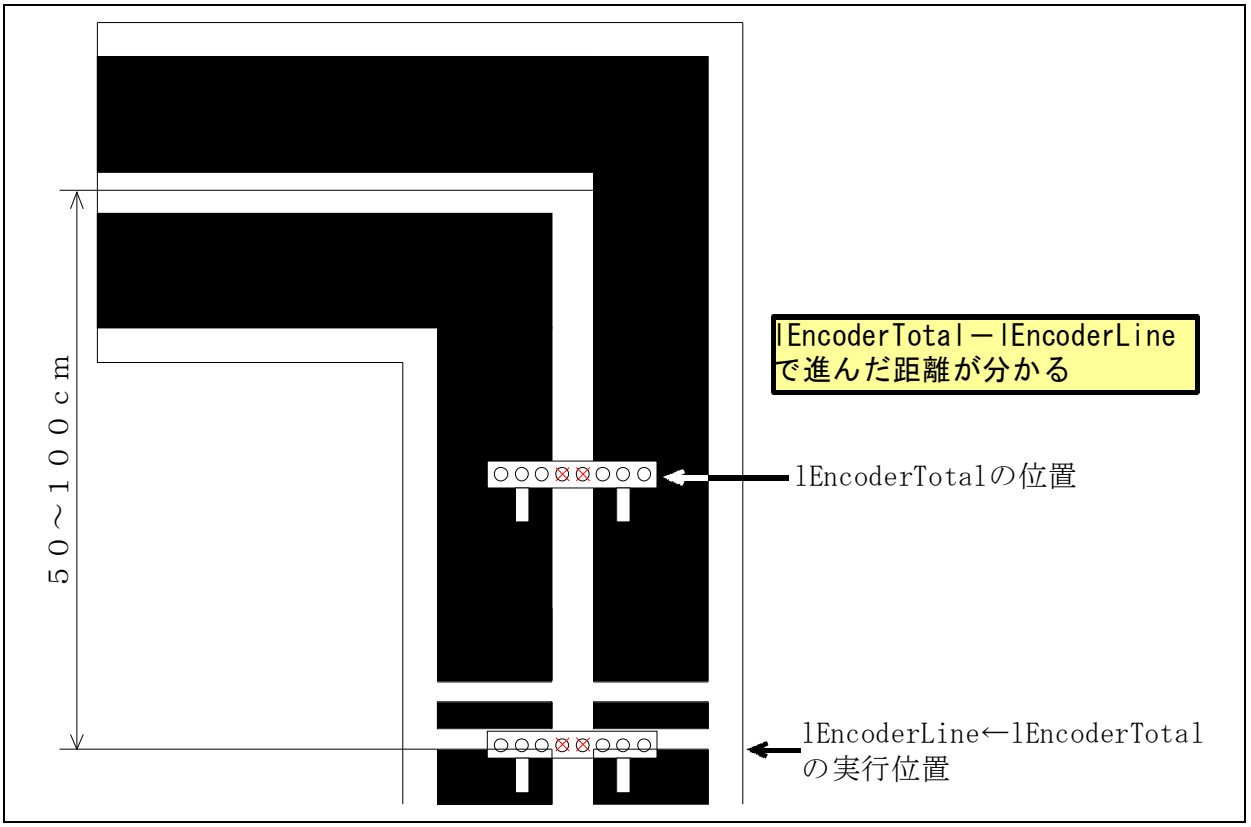
```
140 :      case 1:
141 :          /* スタートバーが開いたかチェック */
142 :          if( !startbar_get() ) {
143 :              /* スタート！！ */
144 :              lEncoderTotal = 0;
145 :              led_out( 0x0 );
146 :              pattern = 11;
147 :              cnt1 = 0;
148 :              break;
149 :   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

lEncoderTotal 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。

6.3.3 パターン 21 クロスライン検出ときの積算値を取得

```
283 :       case 21:
284 :           /* １本目のクロスライン検出時の処理 */
285 :           lEncoderLine = lEncoderTotal;
286 :           led_out( 0x3 );
287 :           handle( 0 );
288 :           speed( 0 ,0 );
289 :           pattern = 22;
290 :           cnt1 = 0;
291 :           break;
```

クロス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、クロスラインを検出してからのパルス数が分かります。ようは、**クロスラインから進んだ距離が分かります。**



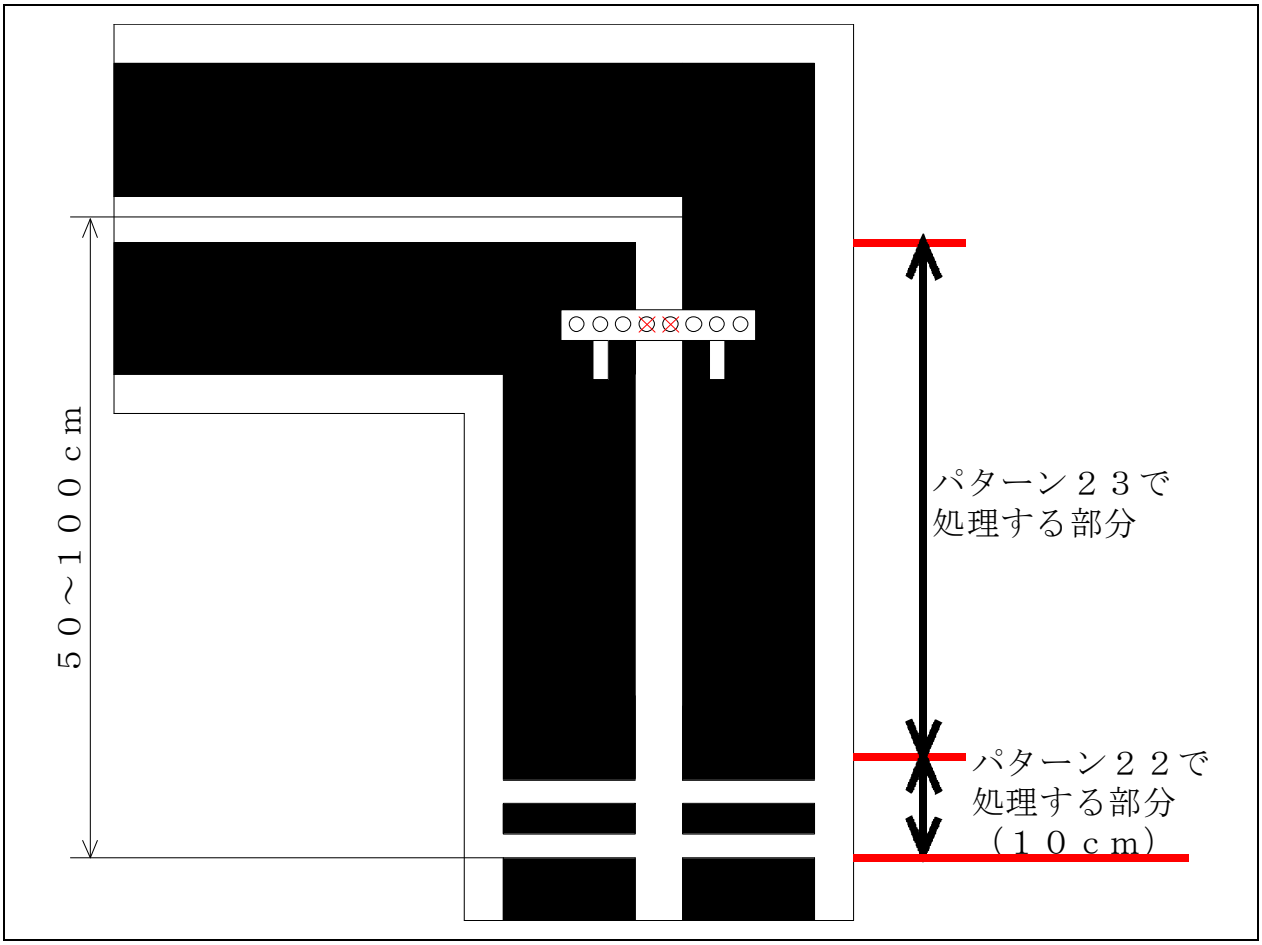
6.3.4 パターン 22 ２本目を読み飛ばす

```
293 :       case 22:
294 :           /* ２本目を読み飛ばす */
295 :           if( lEncoderTotal-lEncoderLine >= 109 ) {   /* 約 10cm たったか？ */
296 :               pattern = 23;
297 :               cnt1 = 0;
298 :           }
299 :           break;
```

295 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。

lEncoderTotal- lEncoderLine　　>= 10cm
↓
現在の積算値 - クロスラインを検出したときの積算値　>= 10cm
↓
クランク内で進んだパルス数(距離)　　>= 10cm

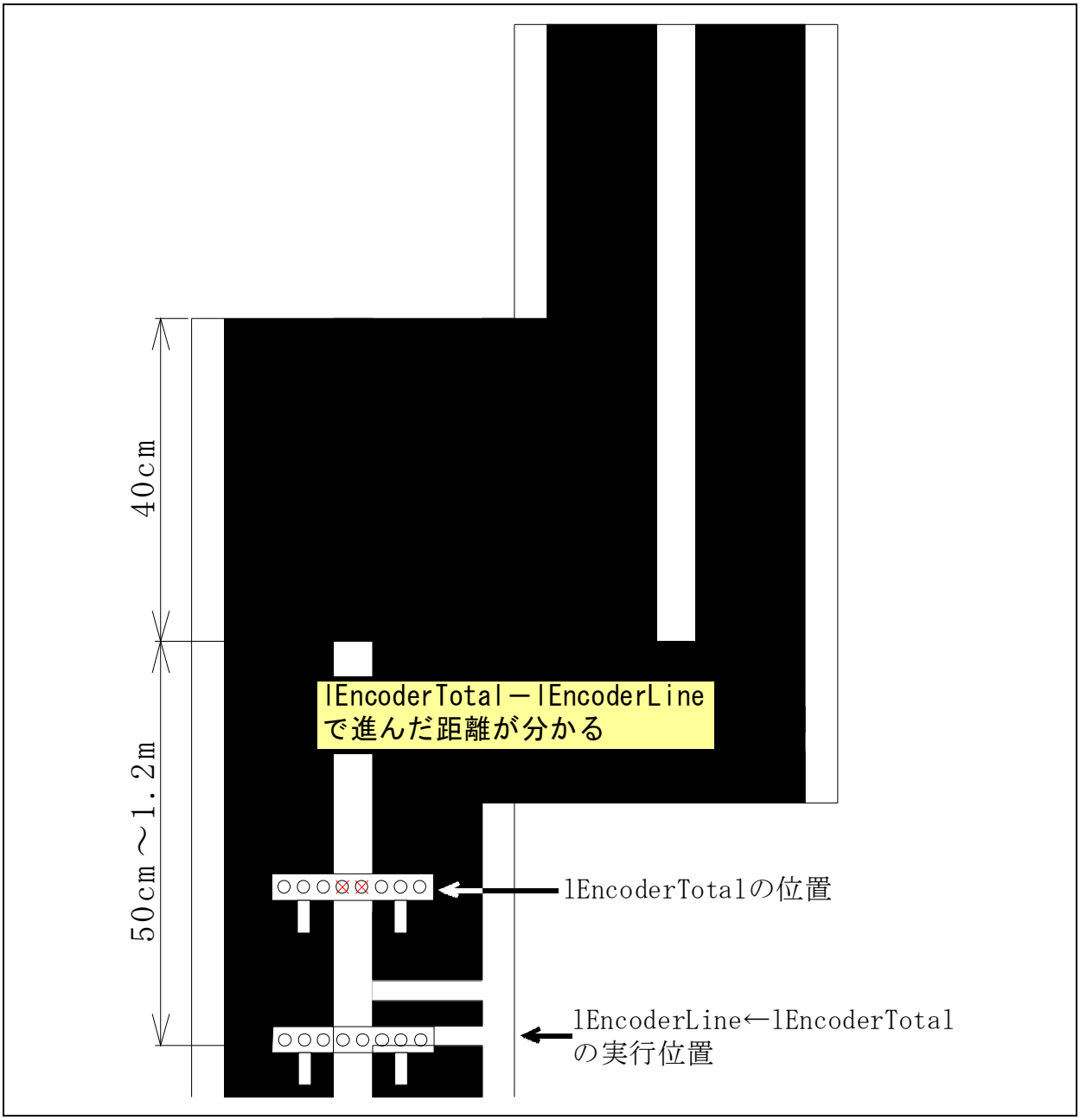
今回のエンコーダは 1m で 1092 パルスのエンコーダなので、10cm 進んだかどうかチェックするには、
1m：1092 パルス＝0.1m：x パルス
x＝109.2 パルス
と、クロスラインを検出した瞬間から 109 パルス以上になったかプログラムで見れば良いことになります。
109 パルス以上になると 10cm 進んだと判断して、パターン 23 へ移ります。



6.3.5 パターン 51 右ハーフライン検出ときの積算値を取得

```
382 :       case 51:
383 :           /* １本目の右ハーフライン検出時の処理 */
384 :           lEncoderLine = lEncoderTotal;
385 :           led_out( 0x2 );
386 :           handle( 0 );
387 :           speed( 0 ,0 );
388 :           pattern = 52;
389 :           cnt1 = 0;
390 :           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。ようは、**右ハーフラインから進んだ距離が分かります**。



6.3.6 パターン 52 ２本目を読み飛ばす

```
392 :       case 52:
393 :           /* ２本目を読み飛ばす */
394 :           if( lEncoderTotal-lEncoderLine >= 109 ) {   /* 約 10cm たったか？ */
395 :               pattern = 53;
396 :               cnt1 = 0;
397 :           }
398 :           break;
```

394 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。

lEncoderTotal- lEncoderLine　　>= 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　>= 10cm

↓

右ハーフライン検出後に進んだパルス数(距離)　　>= 10cm

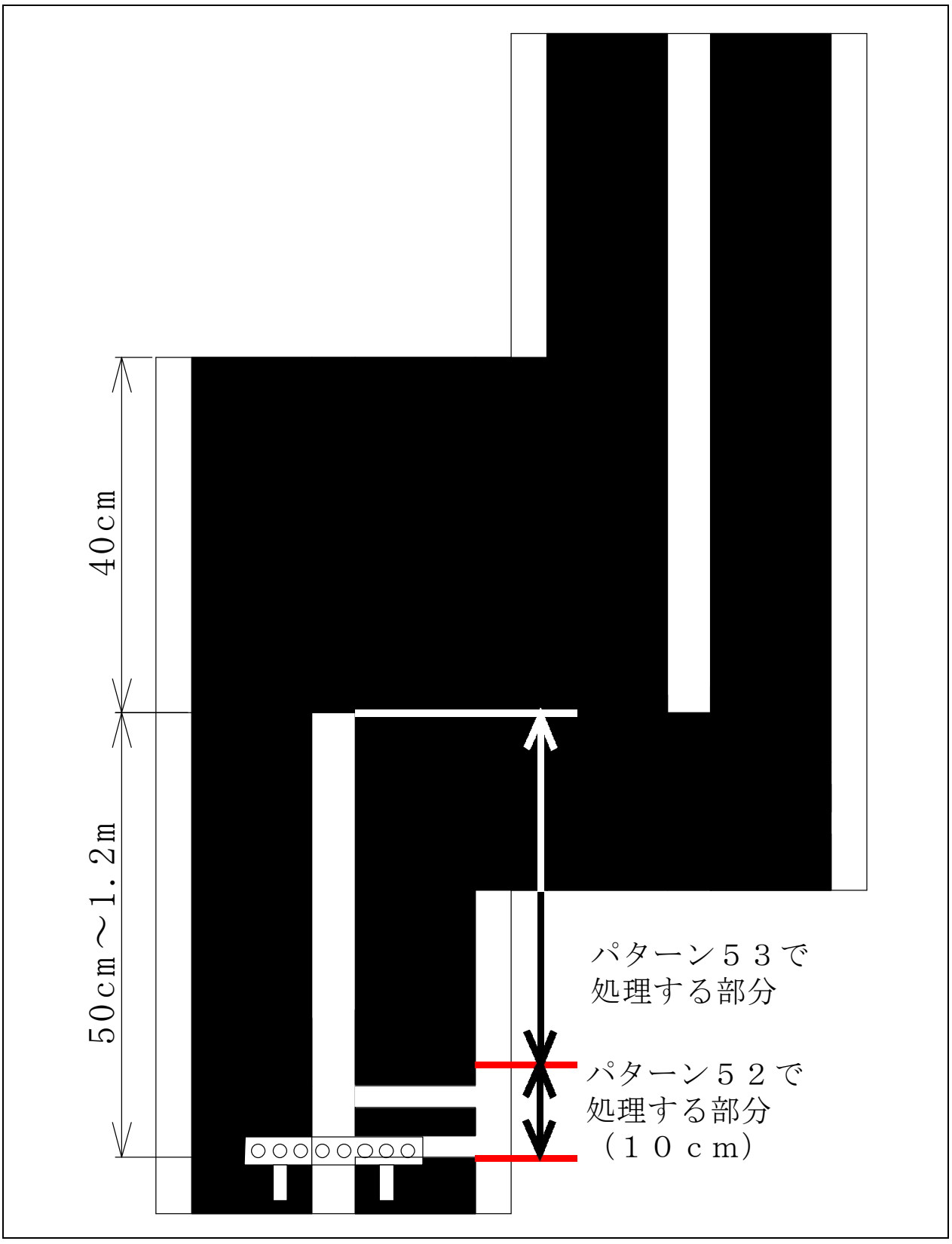
今回のエンコーダは 1m で 1092 パルスのエンコーダなので、10cm 進んだかどうかチェックするには、

1m：1092 パルス＝0.1m：x パルス

x＝109.2 パルス

と、右ハーフラインを検出した瞬間から 109 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。

109 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。


40ｃｍ
50ｃｍ～1．2ｍ
パターン５３で
処理する部分
パターン５２で
処理する部分
（１０ｃｍ）

6.3.7 パターン 61～62 左ハーフライン部分の処理

```
447 :       case 61:
448 :           /* １本目の左ハーフライン検出時の処理 */
449 :           lEncoderLine = lEncoderTotal;
450 :           led_out( 0x1 );
451 :           handle( 0 );
452 :           speed( 0 ,0 );
453 :           pattern = 62;
454 :           cnt1 = 0;
455 :           break;
456 :
457 :       case 62:
458 :           /* ２本目を読み飛ばす */
459 :           if( lEncoderTotal-lEncoderLine >= 109 ) {   /* 約 10cm たったか？ */
460 :               pattern = 63;
461 :               cnt1 = 0;
462 :           }
463 :           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。
449 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。

6.4 プログラムの調整

このサンプルプログラムは、72 パルス／回転、エンコーダのタイヤ直径 21mm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。「**2.5　自分のマイコンカーのパルス数とスピード（距離）の関係**」を参照しながら変更してください。

行番号	元の数値	変更後の数値
35	5000	それぞれのマイコンカーのサーボセンタ値にします。
249	11	右へ大曲げの終わりのチェック中のスピードを設定します。 1m/s にするなら、**(F)**の値にします。
273	11	左へ大曲げの終わりのチェック中のスピードを設定します。 1m/s にするなら、**(F)**の値にします。
295	109	クロスラインを検出後、10cm センサを見ない距離を設定します。 **(E)**の値にします。
306	-38	左クランクを曲がるときの角度です。 左最大角度を設定します。
315	38	右クランクを曲がるときの角度です。 右最大角度を設定します。
321	11	クロスラインを検出後、徐行して進むスピードを設定します。 1m/s にするなら、**(F)**の値にします。
394	109	右ハーフラインを検出後、10cm センサを見ない距離を設定します。 **(E)**の値にします。
409	11	右ハーフラインを検出後、徐行して進むスピードを設定します。 1m/s にするなら、**(F)**の値にします。2m/s にするなら、**(G)**の値にします。 最初は**(G)**の値にして、脱輪するようなら**(F)**の値にします。
459	109	左ハーフラインを検出後、10cm センサを見ない距離を設定します。 **(E)**の値にします。
474	11	左ハーフラインを検出後、徐行して進むスピードを設定します。 1m/s にするなら、**(F)**の値にします。2m/s にするなら、**(G)**の値にします。 最初は**(G)**の値にして、脱輪するようなら**(F)**の値にします。
545	0x14	立ち上がり、立ち下がりでカウントアップする設定です。立ち上がりのみでカウントアップする場合、「**0x04**」にします。

7. 参考文献

・(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
・(株)ルネサス テクノロジ
　H8/3687 シリーズ　ハードウェアマニュアル　第 3 版
・(株)ルネサス テクノロジ
　High-performance Embedded Workshop V.4.00　ユーザーズマニュアル　Rev.3.00
・(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
・(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
・電波新聞社　マイコン入門講座　大須賀威彦著　第 1 版
・(株)オーム社　C言語でH8マイコンを使いこなす　鹿取祐二著　第 1 版
・ソフトバンク(株)　新C言語入門シニア編　林晴比古著　初版
・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

マイコンカーラリーについての詳しい情報は、マイコンカーラリ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」、または「マイコン」→「Tiny」でご覧頂けます

※リンクは、2009 年 6 月現在の情報です。