

マイコンカーラリー用

ロータリエンコーダ kit12_38a プログラム 解説マニュアル (R8C/38A 版)

2013年度から、RY_R8C38ボードに搭載されているマイコンがR8C/38AからR8C/38Cに変更されました。R8C/38AマイコンとR8C/38Cマイコンは、機能的にほぼ互換で、マイコンカーで使う範囲においてはプログラムの変更はほとんどありません。よって、本マニュアルではマイコンの名称を『R8C/38A』で統一します。

第 1.01A 版
2015.04.27
ジャパンマイコンカーラリー実行委員会
株式会社日立ドキュメントソリューションズ

注意事項 (rev.6.0J)

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、最新の内容を確認いただきますようお願いいたします。
- ・すべての商標および登録商標は、それぞれの所有者に帰属します。

連絡先

株式会社 日立ドキュメントソリューションズ

〒135-0016 東京都江東区東陽六丁目 3 番 2 号 イースト 21 タワー

E-mail:himdx.m-carrally.dd@hitachi.com

目次

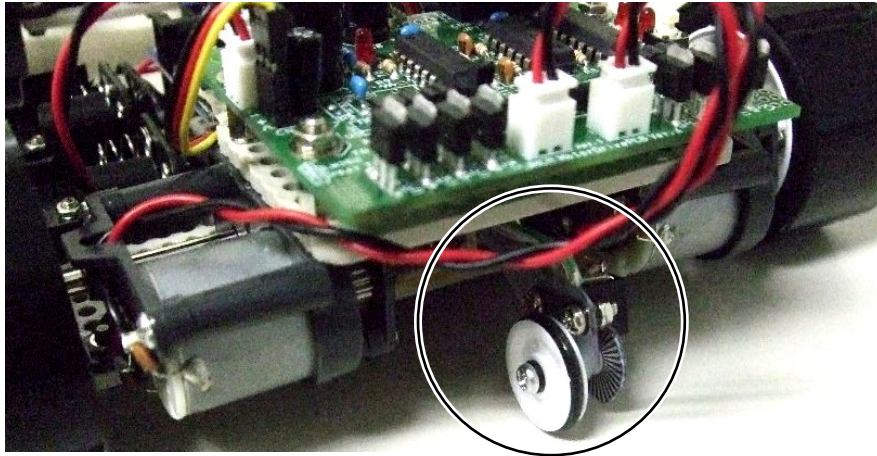
1. ロータリエンコーダを使う	1
1.1 ロータリエンコーダとは.....	1
1.2 原理.....	1
1.3 2相出力のロータリエンコーダ.....	2
1.4 RY_R8C38 ボードの接続先.....	3
2. マイコンカーへの取り付け	4
2.1 マイコンカーで使えるロータリエンコーダの条件.....	4
2.2 市販されているロータリエンコーダを使う.....	4
2.2.1 ロータリエンコーダの例.....	4
2.2.2 回路.....	5
2.2.3 簡易回路.....	5
2.2.4 回転部分の加工.....	7
2.2.5 マイコンカーへの取り付け.....	8
2.2.6 即席の取り付け例.....	9
2.3 ロータリエンコーダ Ver.2を使う.....	10
2.3.1 フォトインタラプタとは.....	10
2.3.2 ロータリエンコーダ Ver.2.....	11
2.3.3 GP1A51HRJ00F を使った回路例.....	11
2.3.4 回転部分.....	12
2.4 パルス数とスピード(距離)の関係.....	13
2.4.1 タイヤが1回転したときのパルス数の計算.....	13
2.4.2 1m進んだときのパルス数の計算.....	13
2.4.3 秒速1mで進んだとき1秒間のパルス数の計算.....	13
2.4.4 秒速1mで進んだとき、10ms間のパルス数の計算.....	14
2.4.5 プログラムで速度を検出する.....	14
2.4.6 プログラムで距離を検出する.....	15
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.....	16
2.6 RY_R8C38 ボードへの接続.....	17
2.6.1 RY_R8C38 ボードコネクタ変換基板を使う.....	17
2.6.2 液晶・microSD 基板を使う.....	18
3. サンプルプログラム	19
3.1 プログラムの開発環境.....	19
3.2 サンプルプログラムのインストール.....	19
3.2.1 ホームページからダウンロードする.....	19
3.2.2 インストール.....	20
3.3 ワーススペース「kit12enc_38a」を開く.....	21
3.4 プロジェクト.....	22
4. プロジェクト「kit12enc01_38a」速度の調整	23
4.1 プロジェクトの構成.....	23
4.2 プログラム.....	23
4.3 ロータリエンコーダの接続.....	31
4.4 プログラムの解説.....	32
4.4.1 ロータリエンコーダ関連の変数の宣言.....	32

4.4.2	パターン 0:ロータリエンコーダの状態をモータドライブ基板の LED へ出力	32
4.4.3	入出力設定の変更	33
4.4.4	外部パルス入力設定	34
4.4.5	円盤の黒、透明(白)の間隔が違うとき	38
4.4.6	タイマ RB 割り込み処理	38
4.4.7	更新する間隔について	39
4.4.8	タイマ RG カウンタ(TRG)が 65535 から 0 になったとき	40
4.4.9	なぜ、バッファを使うのか	41
4.4.10	パターン 12 右大曲げ時の処理	42
4.4.11	motor2 関数	43
4.4.12	パターン 13 左大曲げ時の処理	44
4.4.13	パターン 23 クロスライン後のトレース、クランク検出時の処理	45
4.5	プログラムの調整	46
5.	プロジェクト「kit12enc02_38a」 距離の検出(パターンの区分けを距離で行う)	47
5.1	プロジェクトの構成	47
5.2	プログラム	47
5.3	プログラムの解説	49
5.3.1	変数の追加	49
5.3.2	積算値のクリア	49
5.3.3	パターン 21 クロスライン検出時の積算値を取得	50
5.3.4	パターン 22 クロスラインを読み飛ばす	51
5.3.5	パターン 51 右ハーフライン検出時の積算値を取得	52
5.3.6	パターン 52 右ハーフラインを読み飛ばす	53
5.3.7	パターン 61~62 左ハーフライン部分の処理	54
5.4	プログラムの調整	55
6.	参考文献	56

1. ロータリエンコーダを使う

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。



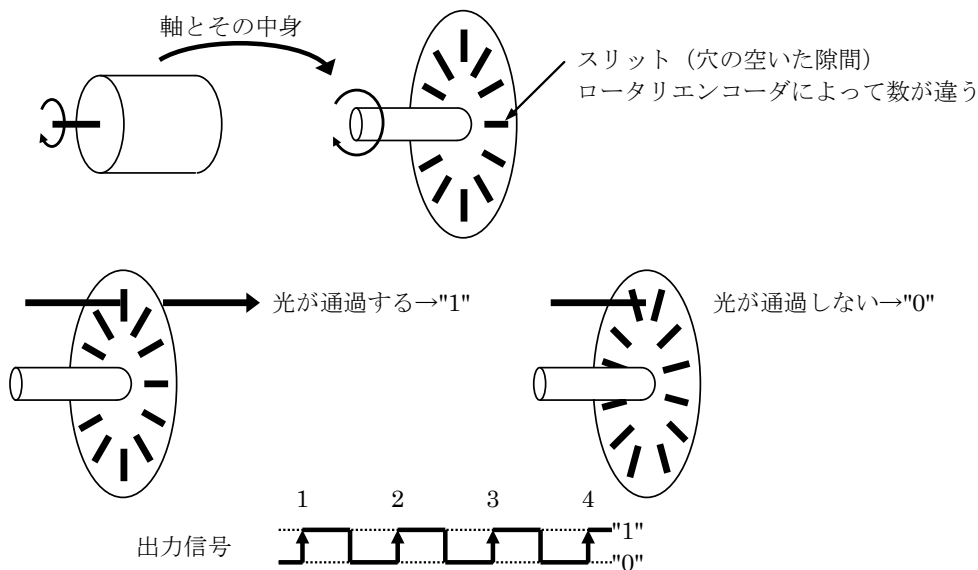
▲マイコンカーに取り付けたロータリエンコーダ (ロータリエンコーダ Ver.2)

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。



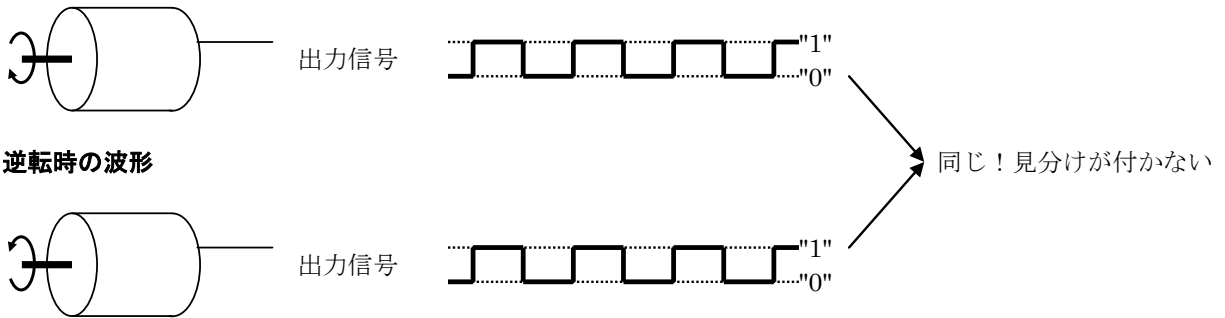
"0"から"1"になる回数を数えれば、距離が分かります。また、ある一定時間、例えば1秒間の回数をカウントして、多ければ回転が速い(=スピードが速い)、少なければ回転が遅い(=スピードが遅い)と判断できます。

1. ロータリエンコーダを使う

1.3 2相出力のロータリエンコーダ

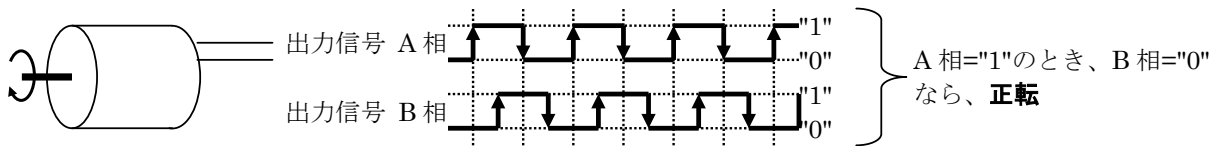
ロータリエンコーダには、1相出力と2相出力があります。先ほどの説明は、1相出力の場合です。1相の場合、回転が正転か逆転か分かりません。どちらも"1"と"0"の信号でしかないためです。

正転時の波形

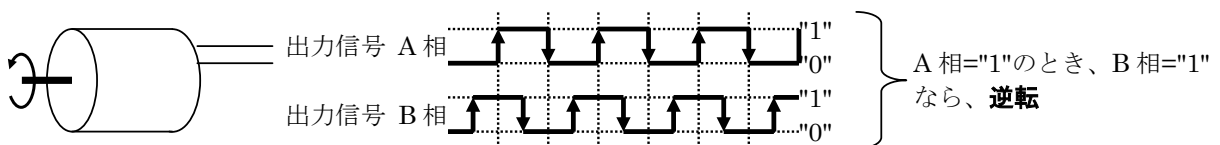


そこで、出力をA相という名前と、B相という名前で2つ出力します。同じ信号を出力しても意味が無いので、B相の光検出をA相の出力より90度分ずれるようにします。このとき、下図のようにA相の信号が"0"から"1"になったとき、B相の信号レベルを調べることで回転方向が分かります。

正転時の波形



逆転時の波形



最近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相のロータリエンコーダが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。

1. ロータリエンコーダを使う

1.4 RY_R8C38 ボードの接続先

RY_R8C38 ボード(R8C/38A マイコン)でマイコンカーを制御するときの内蔵周辺モジュールの一覧を下記に示します。参考として、H8/3048F-ONE でマイコンカーを制御するときの内蔵周辺モジュールと比較します。

項目	R8C/38A でマイコンカーを制御するときの内蔵周辺モジュール	H8/3048F-ONE でマイコンカーを制御するときの内蔵周辺モジュール(参考)
1ms ごとの割り込み	タイマ RB	ITU0
左モータ、右モータ サーボの制御	タイマ RD によるリセット同期 PWM モード	ITU3,4 によるリセット同期 PWM モード
モータを4輪独立制御 する場合の 前輪モータ制御	タイマ RC	ITU0,ITU1
ロータリエンコーダ	タイマ RG	ITU2
赤外線を受光制御	タイマ RA	

今回、ロータリエンコーダのパルスカウントはタイマ RG を使います。

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ロータリエンコーダの条件



ロータリエンコーダを探すといろいろな種類があります。どのようなロータリ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します。小さければ小さいほど良いですが、値段が高くなります。直径 20~30mm くらいまでが実用範囲内です。
重さ	軽いロータリエンコーダを選びます。
出力信号	マイコンは、デジタル信号しか扱えないので、ロータリ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、マイコンに合わせて"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	マイコンと同様に 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	多ければ多いほど速度や距離を細かく知ることができます。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 市販されているロータリエンコーダを使う

2.2.1 ロータリエンコーダの例

市販されているロータリエンコーダでマイコンカーに使用できそうなロータリエンコーダを下記に示します。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。

メーカー	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。このロータリ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。プルアップ抵抗も不要です。φ12mm と小型です。
写真		

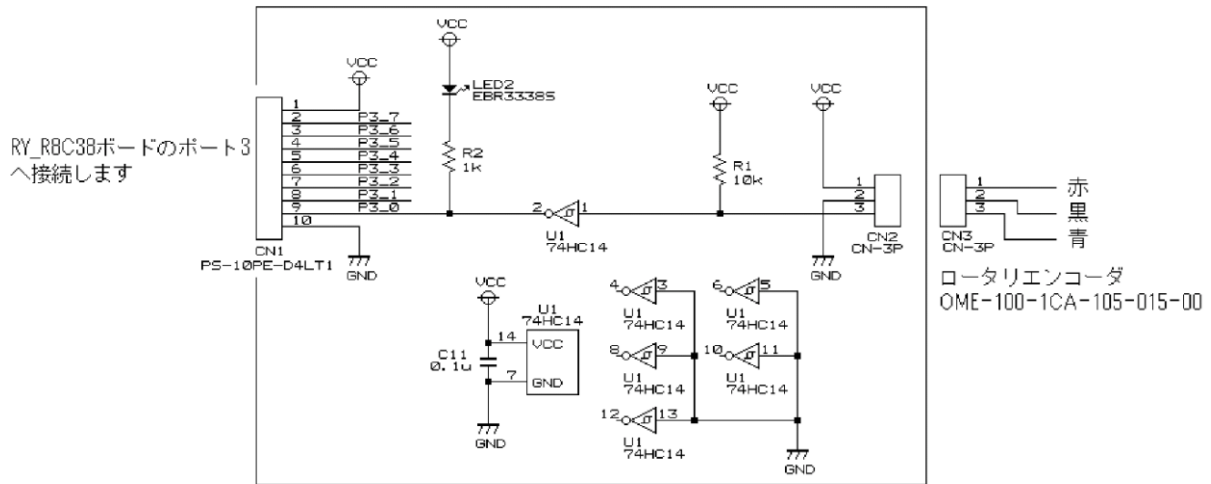
2. マイコンカーへの取り付け

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。

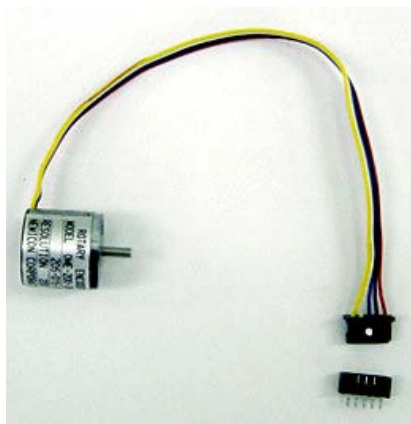
「OME-100-1CA-105-015-00」の出力信号は、デジタル信号のため、そのままポートに接続可能です。ただし、オープンコレクタ出力なのでプルアップ抵抗が必要です。一応、74HC14などで波形整形すると良いでしょう。

RY_R8C38 ボードのポート 3 の bit0 にロータリ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。



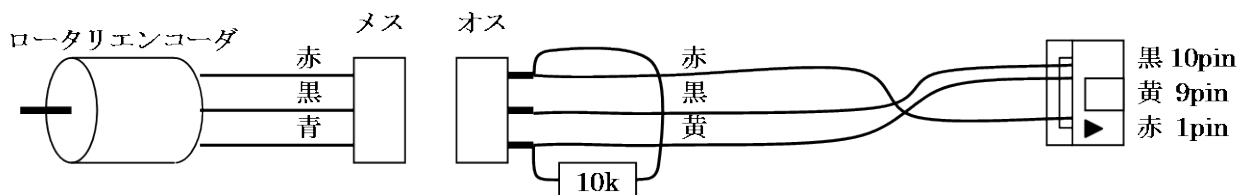
2.2.3 簡易回路

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」のロータリエンコーダのため、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。

配線を下図に示します。

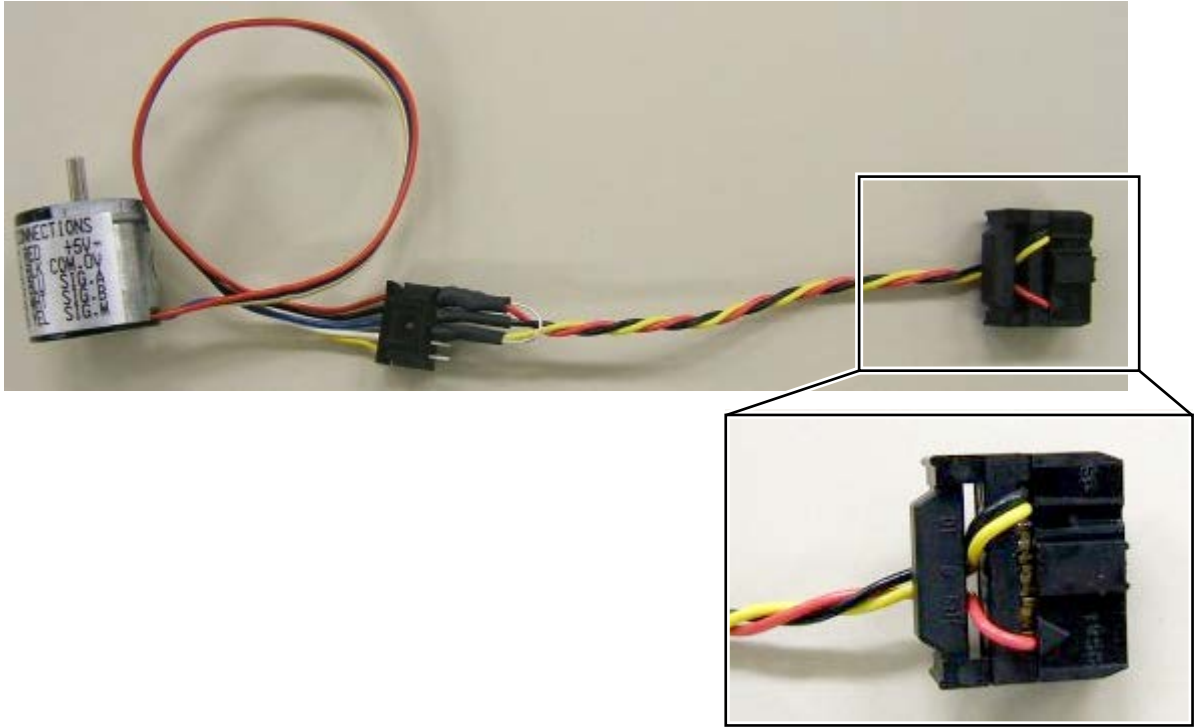


2. マイコンカーへの取り付け

RY_R8C38 ボードのポート 3 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。

ロータリ 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(赤)
黒	GND	10ピン(黒)
青	P3_0	9ピン(黄)

▼製作例



※RY_R8C38 ボードのポート 3 への接続方法については、「2.6 RY_R8C38 ボードへの接続」を参照してください。

2. マイコンカーへの取り付け


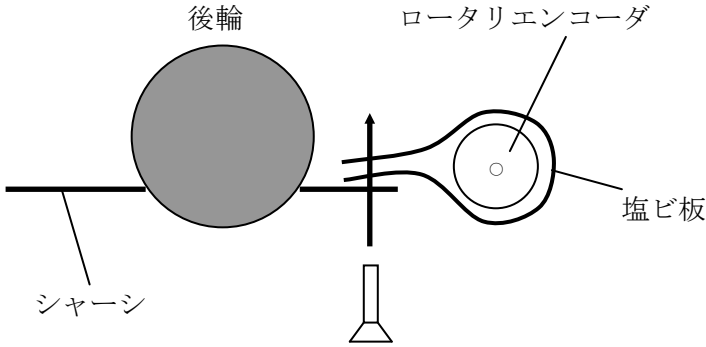
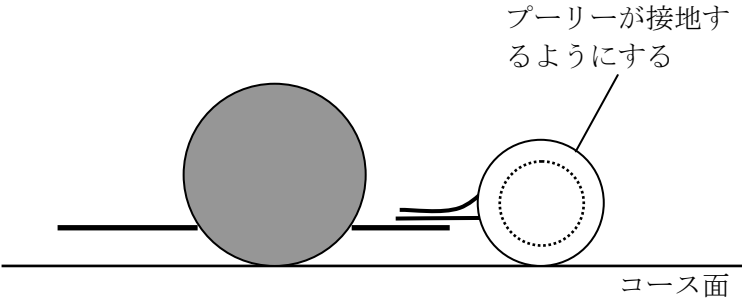
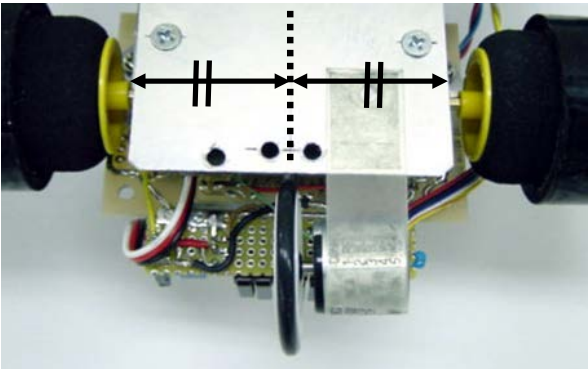
2.2.4 回転部分の加工

ロータリ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。

<p>1</p>		<p>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。</p> <p>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 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。</p>
<p>2</p>		<p>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、ロータリエンコーダに取り付けます。</p> <p>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。</p>
<p>3</p>		<p>ロータリ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。</p>

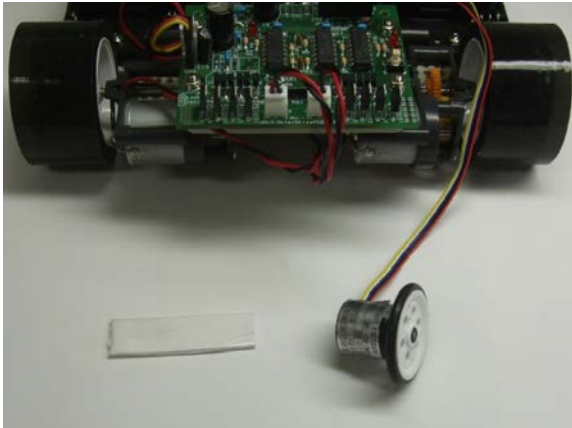

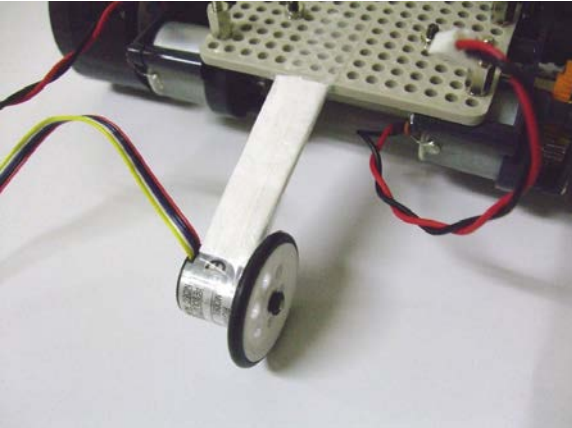
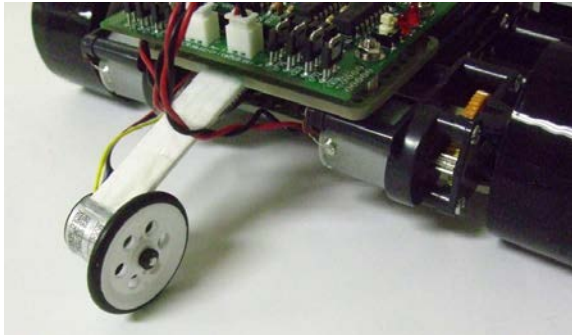
2. マイコンカーへの取り付け

2.2.5 マイコンカーへの取り付け

<p>1</p>		<p>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 厚の塩化ビニール板を用意しました。薄く弾力性のある素材であれば何でも構いません。</p>
<p>2</p>		<p>塩ビ板などの弾力のある素材で、ロータリエンコーダを巻くようにします。ロータリエンコーダと塩ビ板は、両面テープで止めます。塩ビ板の両端を合わせて、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。</p>
<p>3</p>		<p>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。</p>
<p>4</p>		<p>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。</p>

2. マイコンカーへの取り付け

2.2.6 即席の取り付け例

1	 A photograph showing a microcontroller board with a rotary encoder and a small piece of white paper. The encoder is connected to the board with colored wires.	<p>取り付けるマイコンカー、ロータリエンコーダ、15×50mm 程度の厚紙(硬めの板)とセロテープを用意します。</p>
2	 A close-up photograph of the rotary encoder with a piece of white paper and a strip of clear tape attached to its top surface.	<p>セロテープでロータリエンコーダと厚紙を止めます。がっちり止めます。</p>
3	 A photograph showing the encoder assembly mounted on a breadboard. The white paper is positioned so that the encoder's pulley is centered on the breadboard's surface.	<p>マイコンカー側もロータリエンコーダのプーリーが中心にくるように、セロテープで厚紙を止めます。上り坂、下り坂でもロータリ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。</p>
4	 A photograph showing the encoder assembly mounted on the microcontroller board. The white paper is secured with tape, and the pulley is positioned to make contact with the ground plane of the board.	<p>この方法はすぐにとれてしまうので、実験のみの使用にしましょう。</p>

2. マイコンカーへの取り付け

2.3 ロータリエンコーダ Ver.2 を使う

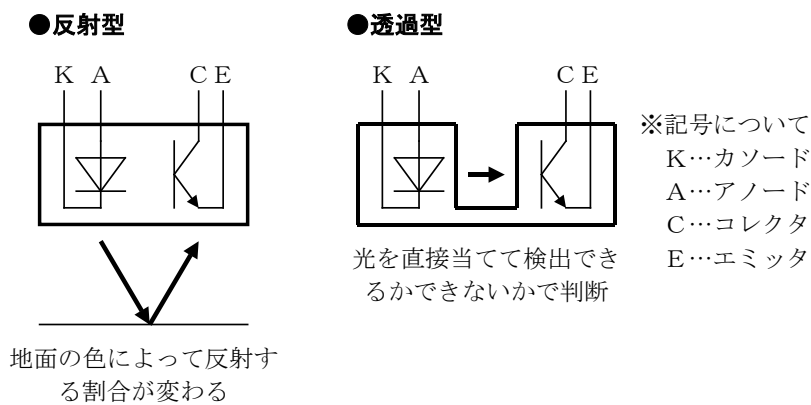
市販されているロータリエンコーダは1回転100パルス以上と性能は申し分ありません。しかし値段が高いのが難点です。そこで、パルス数が少なくなりますが、フォトインタラプタを使った安価なロータリエンコーダ Ver.2 を紹介します。

※ロータリエンコーダ Ver.2 は、

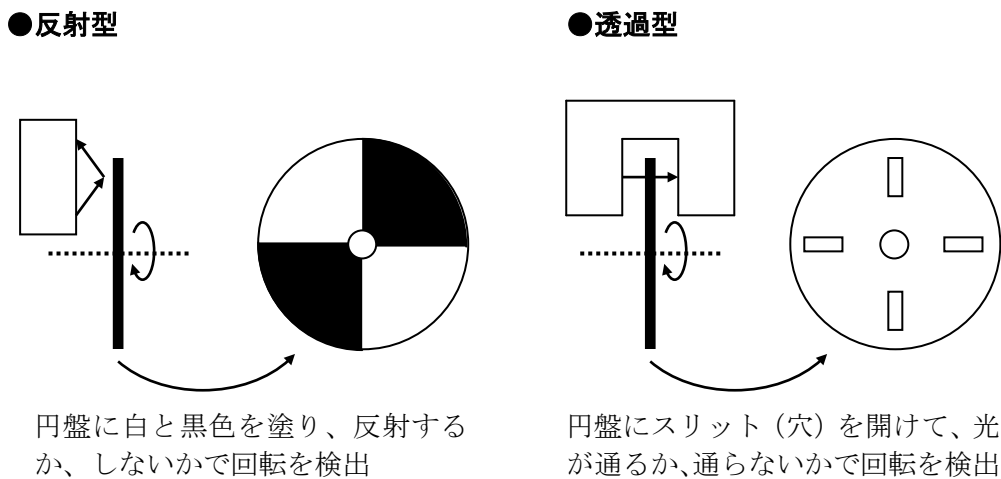
マイコンカーラリー販売サイト(URL:<http://www2.himdx.net/mcr/>)で販売しています。

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ロータリ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。




ロータリエンコーダ Ver.2 は、透過型のフォトインタラプタを使用しています。

2. マイコンカーへの取り付け

2.3.2 ロータリエンコーダ Ver.2

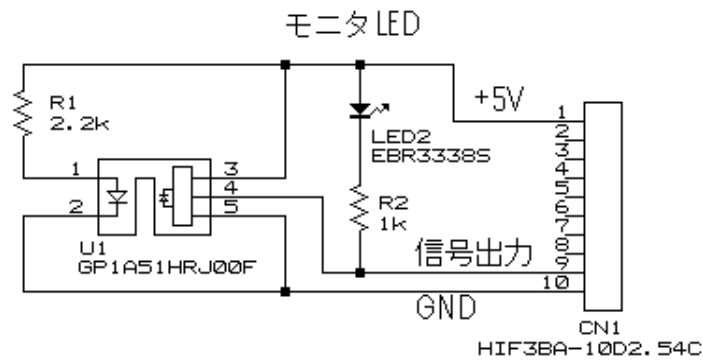
ロータリエンコーダ Ver.2 で使用しているフォトインタラプタを下記に示します。

写真	メーカー	シャープ(株)
	型式	GP1A51HRJ00F
	特徴	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。

2.3.3 GP1A51HRJ00F を使った回路例

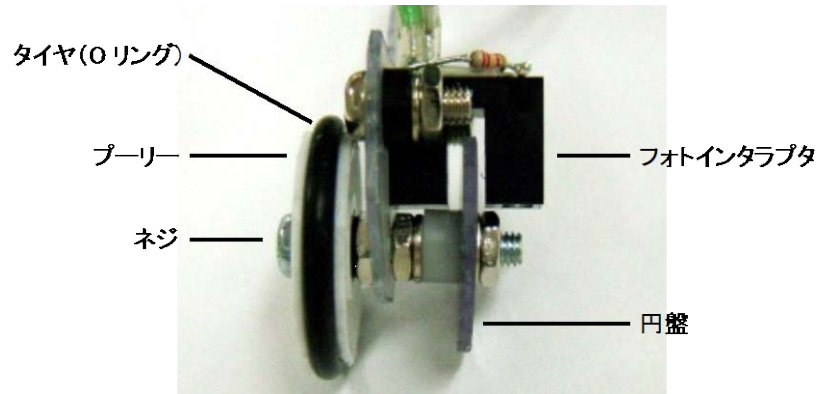
「GP1A51HRJ00F」の出力信号は、0V か 5V なのでそのままポートに接続することができます。モニタ LED は、信号が来ているか確認するのに便利です。付けるスペースがあるなら、付けましょう。

ロータリエンコーダ Ver.2 は、下記回路の LED が付いていない構成です。今回、信号が来ているかどうかの確認は、プログラムでロータリエンコーダの信号を読み込み、モータドライブ基板の LED に出力しています。

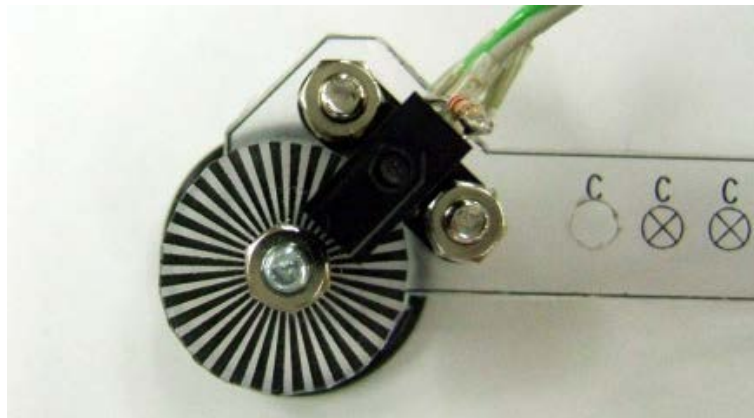


2.3.4 回転部分

フォトインタラプタの間に入れる円盤とコース面に接地しているタイヤ(Oリング)と一緒に回るようにします。



円盤は、下写真のように黒と透明が交互にあります。円盤が回転することによりフォトインタラプタの赤外LED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ロータリ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、ロータリエンコーダ Ver.2 を使用することにして計算します。ロータリエンコーダ Ver.2 の仕様を下表に示します。

項目	内容
ロータリエンコーダ 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。

$$\text{円周} = 2 \pi r = 2 \times 10.5 \times 3.14 = 65.94\text{mm}$$

ロータリエンコーダは 72 パルス/回転なので、

65.94mm 進むと 72 パルス(1)

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、

$$72 \text{ パルス} : 65.94\text{mm} = x \text{ パルス} : 1000\text{mm}$$

$$x = 1091.9 \text{ パルス}$$

1m (1000mm)進むと、1091.9 パルス(2)

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)

2. マイコンカーへの取り付け

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、

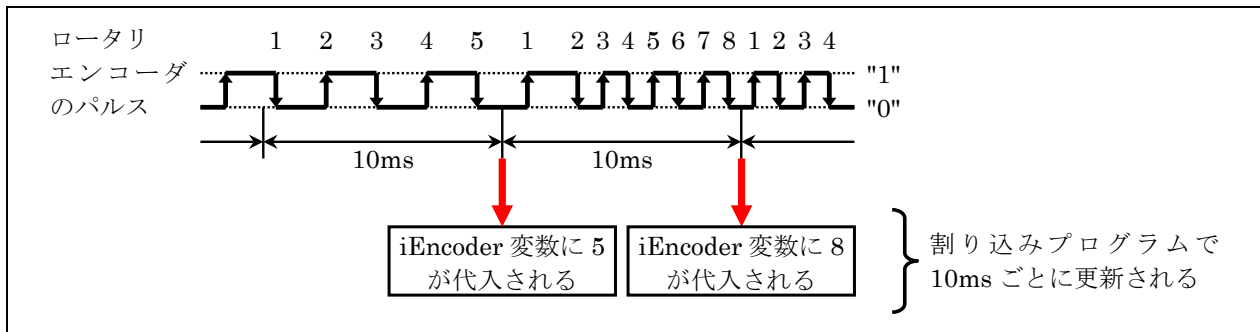
$$1 \text{ 秒} : 1091.9 \text{ パルス} = 0.01 \text{ 秒} : x \text{ パルス}$$

$$x = 10.919 \approx 10.92 \text{ パルス}$$

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)

2.4.5 プログラムで速度を検出する

今回のロータリエンコーダを使ったプログラムでは、「iEncoder」という変数に 10ms 間のロータリエンコーダのパルス数が 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 で走っていることになります。

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%にする場合の考え方を下記に示します。

```

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

この考え方を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。今回のロータリエンコーダは秒速 1m/s で 10.92 パルスなので、2m/s は、

$$\begin{aligned}
 \text{秒速 2m/s のパルス数} &= \text{秒速 1m/s のパルス数} \times 2 \\
 &= 10.92 \times 2 \\
 &= 21.84 \\
 &\approx 22 \quad \text{※iEncoder 変数は整数型なので四捨五入して整数にします}
 \end{aligned}$$

プログラムを下記に示します。

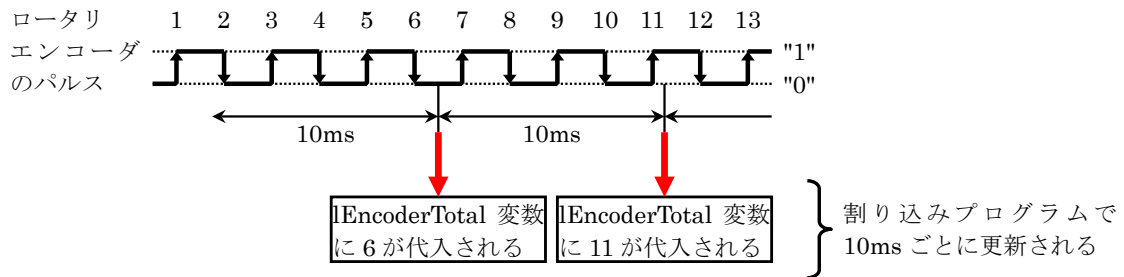
```

if( iEncoder >= 22 ) {
    motor2( 0, 0 );
} else {
    motor2( 70, 70 );
}
    
```

2. マイコンカーへの取り付け

2.4.6 プログラムで距離を検出する

今回のロータリエンコーダを使ったプログラムでは、「lEncoderTotal(エルエンコーダトータル)」という変数にロータリエンコーダの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 進んだということです。

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%にする場合の考え方を下記に示します。

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

この考え方を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。今回のロータリエンコーダは 1m 進むと 1091.9 パルスなので、10m は、

$$\begin{aligned} \text{距離 10m のパルス数} &= \text{1m のパルス数} \times 10 \\ &= 1091.9 \times 10 \\ &= 10919 \end{aligned}$$

プログラムを下記に示します。

```
if( lEncoderTotal >= 10919 ) {
    motor2( 0, 0 );
} else {
    motor2( 100, 100 );
}
```

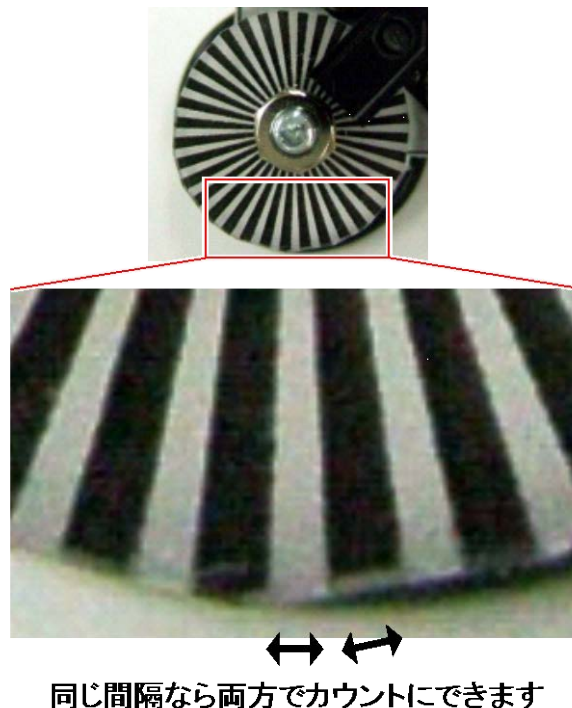
2.5 自分のマイコンカーのパルス数とスピード(距離)の関係

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、立ち下がりの両方でカウントする設定です。	パルス (B)
円周 = $2\pi \times (A)$	mm (C)
1000mm 進んだときのパルス数は、 $1000 : x = (C) : (B) \therefore x = 1000 \times (B) \div (C)$	パルス (D)
100mm 進んだときのパルス数は、 $(E) = (D) \times 0.1$	パルス (E) ※四捨五入した整数
1m/s で進んだとき、10ms 間のパルス数は、 $(F) = (D) \times 0.01$	パルス (F) ※四捨五入した整数
2m/s で進んだとき、10ms 間のパルス数は、 $(G) = (D) \times 0.02$	パルス (G) ※四捨五入した整数

(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(下写真)。間隔が違う場合は、両方でカウントはできません。



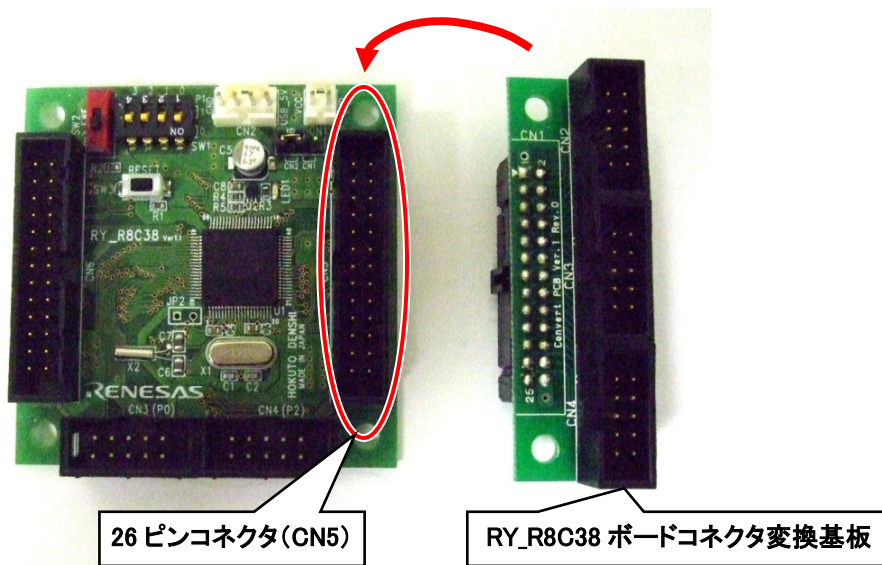
詳しくは、「4.4.5 円盤の黒、透明(白)の間隔が違うとき」を参照してください。

2.6 RY_R8C38 ボードへの接続

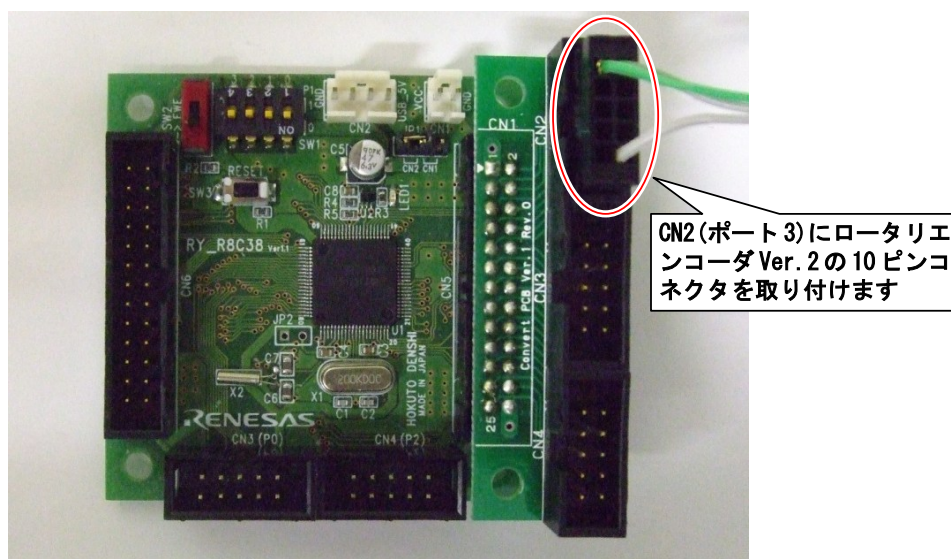
RY_R8C38 ボードのポート3は26ピンコネクタなので、ロータリエンコーダ Ver.2 の10ピンメスコネクタを直接接続することができません。ここでは、ロータリエンコーダの信号を RY_R8C38 ボードのポート3へ接続する方法について説明します。

2.6.1 RY_R8C38 ボードコネクタ変換基板を使う

「RY_R8C38 ボードコネクタ変換基板」は、RY_R8C38 ボードの26ピンコネクタ1個を10ピンコネクタ3個に変換する基板です。

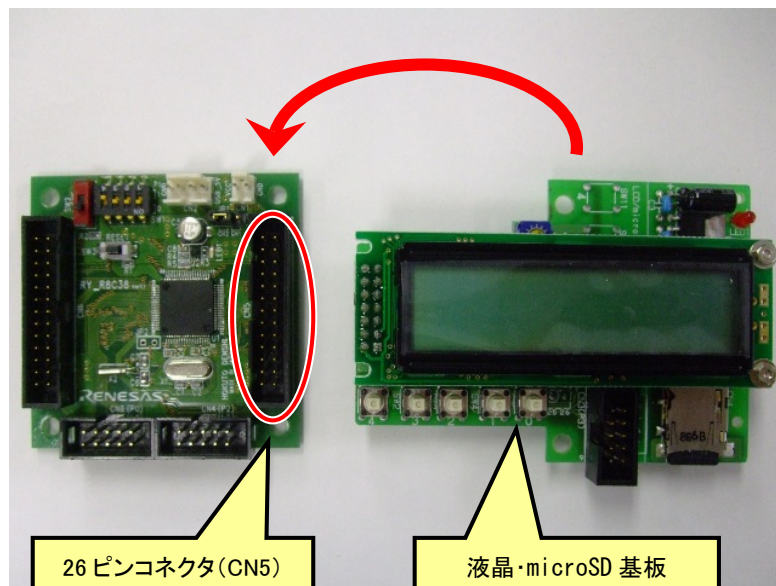


ロータリエンコーダ Ver.2 の10ピンメスコネクタを RY_R8C38 ボードコネクタ変換基板の CN2 に接続します。接続した様子を下写真に示します。RY_R8C38 ボードコネクタ変換基板(右側)の CN2 は、R8C/38A マイコンのポート3に接続されています。

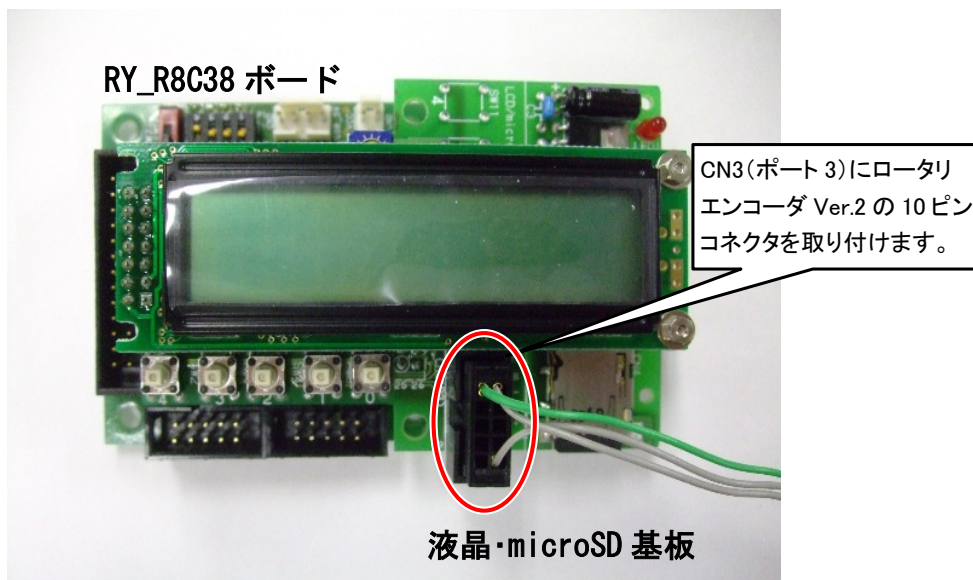


2.6.2 液晶・microSD 基板を使う

液晶・microSD 基板は、RY_R8C38 ボードの CN5 に接続します。この基板を使用するときは、RY_R8C38 ボードコネクタ変換基板を接続することができません。



液晶・microSD 基板には、ポート 3 につながっている 10 ピンコネクタ(CN3)が取り付けられています。ロータリエンコーダ Ver.2 の 10 ピンコネクタは、液晶・microSD 基板の CN3(ポート 3)に接続します。接続した様子を下写真に示します。



3. サンプルプログラム

3.1 プログラムの開発環境

プログラムの開発環境は、ルネサス統合開発環境(High-performance Embedded Workshop)を使います。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(R8C/38A版)」を参照してください。


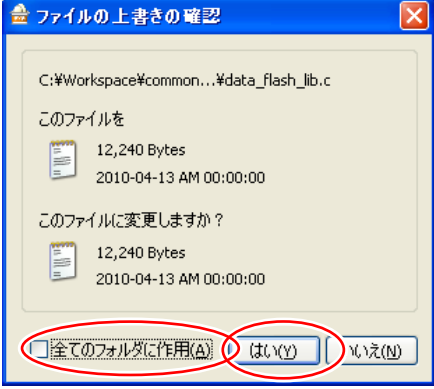
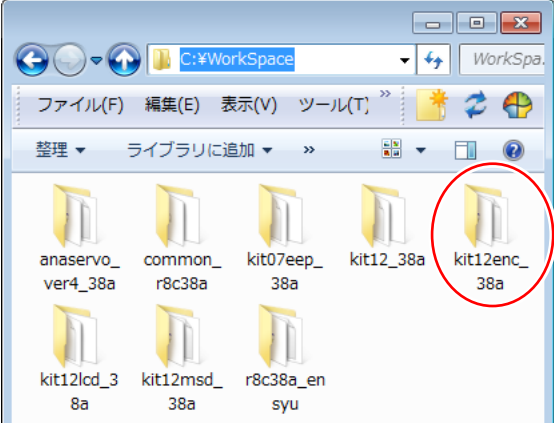
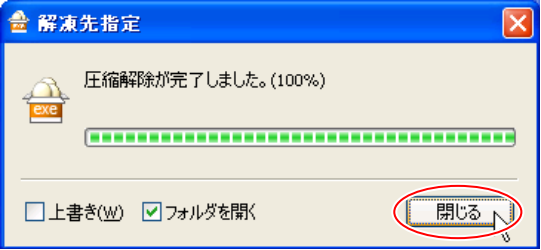
3.2 サンプルプログラムのインストール

サンプルプログラムをインストールします。


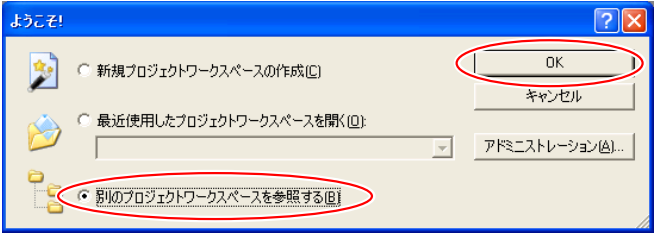
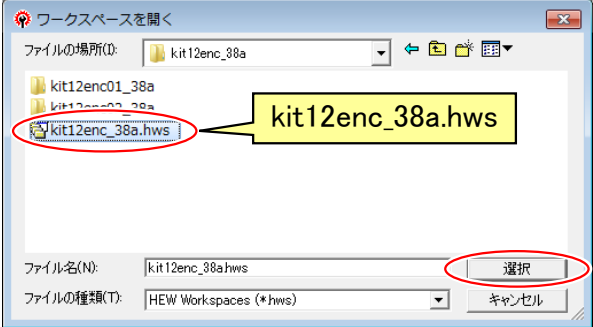
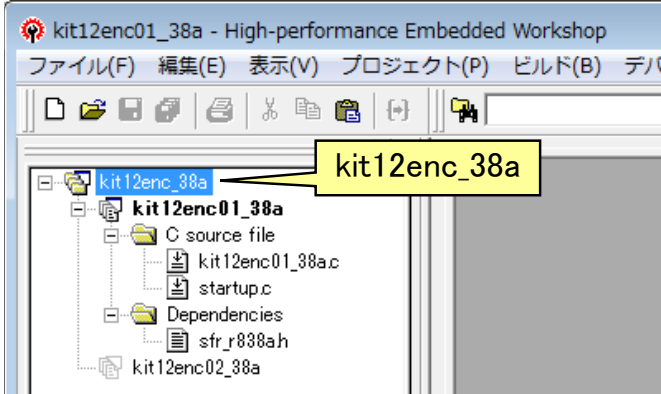
3.2.1 ホームページからダウンロードする

<p>1</p>		<p>マイコンカーラリーサイト 「http://www.mcr.gr.jp/index2.html」 の技術情報→ダウンロード内のページをアクセスします。</p>												
<p>2</p>	<p>免責事項</p> <p>「マニュアル」、「ソフトウェア」は万全な体制で制作されており、通常の使用環境においては正常に動作するように作成されていますが、万が一「マニュアル」、「ソフトウェア」による損失・損害が発生した時には、『ジャパンマイコンカーラリー実行委員会』はいかなる場合も責任を負いません。個人の免責が取れる範囲内であらかじめ了承した上でご使用くださるようお願いいたします。</p> <table border="1"> <thead> <tr> <th>対象マイコン</th> <th>内容</th> <th>更新日</th> </tr> </thead> <tbody> <tr> <td>R8C/38A</td> <td>R8C/38Aマイコン(RY_R8C38ボード)に関する資料</td> <td>2013.06.03 NEW!!</td> </tr> <tr> <td>H8/3048F-ONE</td> <td>H8/3048F-ONEマイコン(RY3048Foneボード)用のサンプルプログラム、書き込みソフト</td> <td>2010.10.07</td> </tr> <tr> <td>H8/3048F</td> <td>H8/3048F-ONEマイコン(RY3048Foneボード)に</td> <td>~~~~~</td> </tr> </tbody> </table>	対象マイコン	内容	更新日	R8C/38A	R8C/38Aマイコン(RY_R8C38ボード)に関する資料	2013.06.03 NEW!!	H8/3048F-ONE	H8/3048F-ONEマイコン(RY3048Foneボード)用のサンプルプログラム、書き込みソフト	2010.10.07	H8/3048F	H8/3048F-ONEマイコン(RY3048Foneボード)に	~~~~~	<p>「R8C/38A マイコン(RY_R8C38 ボード)に関する資料」をクリックします。</p>
対象マイコン	内容	更新日												
R8C/38A	R8C/38Aマイコン(RY_R8C38ボード)に関する資料	2013.06.03 NEW!!												
H8/3048F-ONE	H8/3048F-ONEマイコン(RY3048Foneボード)用のサンプルプログラム、書き込みソフト	2010.10.07												
H8/3048F	H8/3048F-ONEマイコン(RY3048Foneボード)に	~~~~~												
<p>3</p>	<table border="1"> <tr> <td>子2個を追加することのできる基板です。</td> <td>第1.00版 2012.11.15</td> <td></td> <td></td> </tr> <tr> <td>ロータリエンコーダ Ver.2 マイコンカーの走行速度を計測し、制御することのできる機器です。</td> <td>ロータリエンコーダVer.2 製作マニュアル 第1.01版 2011.07.09</td> <td>ロータリエンコーダ kit12_38a プログラム解説マニュアル 第1.00版 2013.05.28</td> <td>kit12enc_38a.exe</td> </tr> <tr> <td>液晶・microSD基板 「液晶・microSD基板 液晶・スイッチャット」</td> <td></td> <td>液晶・microSD基板 kit12_38a プログラム解説</td> <td>kit12enc_38a.exe</td> </tr> </table>	子2個を追加することのできる基板です。	第1.00版 2012.11.15			ロータリエンコーダ Ver.2 マイコンカーの走行速度を計測し、制御することのできる機器です。	ロータリエンコーダVer.2 製作マニュアル 第1.01版 2011.07.09	ロータリエンコーダ kit12_38a プログラム解説マニュアル 第1.00版 2013.05.28	kit12enc_38a.exe	液晶・microSD基板 「液晶・microSD基板 液晶・スイッチャット」		液晶・microSD基板 kit12_38a プログラム解説	kit12enc_38a.exe	<p>「kit12enc_38a.exe」をダウンロードします。</p>
子2個を追加することのできる基板です。	第1.00版 2012.11.15													
ロータリエンコーダ Ver.2 マイコンカーの走行速度を計測し、制御することのできる機器です。	ロータリエンコーダVer.2 製作マニュアル 第1.01版 2011.07.09	ロータリエンコーダ kit12_38a プログラム解説マニュアル 第1.00版 2013.05.28	kit12enc_38a.exe											
液晶・microSD基板 「液晶・microSD基板 液晶・スイッチャット」		液晶・microSD基板 kit12_38a プログラム解説	kit12enc_38a.exe											

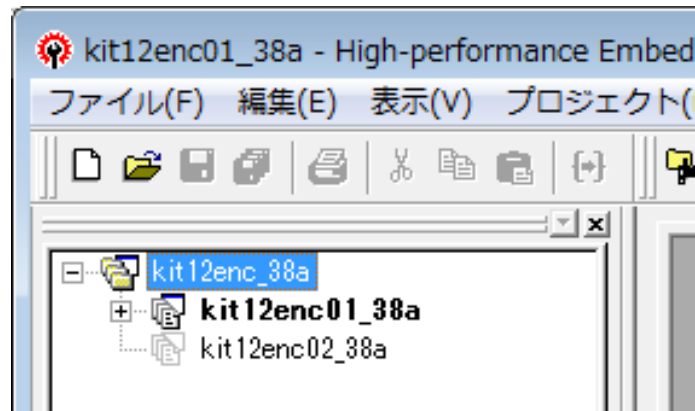
3.2.2 インストール

<p>1</p>		<p>「kit12enc_38a.exe」を実行します。</p> <p>圧縮解除をクリックします。 ※解凍先フォルダは変更しないでください。</p>
<p>2</p>		<p>ファイルの上書き確認の画面が出てきた場合、「全てのファイルに作用」のチェックを付けて、はいをクリックします。</p> <p>※上書きしたくない場合は、元々あるファイルを保存してから実行してください。</p>
<p>3</p>		<p>解凍が終わったら、「C ドライブ → Workspace」フォルダが開かれます。今回使用するのは、「kit12enc_38a」です。</p>
<p>4</p>		<p>閉じるをクリックします。</p>

3.3 ワークスペース「kit12enc_38a」を開く

<p>1</p>		<p>ルネサス統合開発環境を実行します。</p>
<p>2</p>		<p>「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。</p>
<p>3</p>		<p>Cドライブ→Workspace→kit12enc_38a の「kit12enc_38a.hws」を選択し、選択をクリックします。</p>
<p>4</p>		<p>ワークスペース「kit12enc_38a」が開かれます。</p>

3.4 プロジェクト



ワークスペース「kit12enc_38a」には、2つのプロジェクトが登録されています。

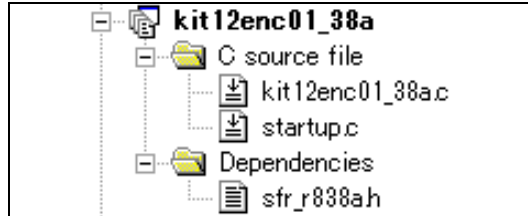
プロジェクト名	内容
kit12enc01_38a	標準走行プログラム「kit12_38a.c」を改造して、ロータリエンコーダのパルスをカウントできるようにしたプログラムです。このプログラムは、マイコンカーのスピード制御を行います。
kit12enc02_38a	このプログラムはスピード制御に加え、距離制御を行います。

プログラムのスピードや距離の設定は、ロータリエンコーダ 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。違うロータリ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てください。

4. プロジェクト「kit12enc01_38a」速度の調整

標準走行プログラムは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。

4.1 プロジェクトの構成



	ファイル名	内容
1	kit12enc01_38a.c	実際に制御するプログラムが書かれています。R8C/38A の内蔵周辺機能(SFR)の初期化も行います。 ファイルの位置→C:\¥Workspace¥kit12enc_38a¥kit12enc01_38a¥kit12enc01_38a.c
2	startup.c	固定割り込みベクタアドレスの設定、スタートアッププログラム、RAM の初期化(初期値のないグローバル変数、初期値のあるグローバル変数の設定)などを行います。 ファイルの位置→C:\¥Workspace¥kit12enc_38a¥kit12enc01_38a¥startup.c
3	sfr_r838a.h	R8C/38A マイコンの内蔵周辺機能を制御するためのレジスタ(Special Function Register)を定義したファイルです。 ファイルの位置→C:\¥Workspace¥common_r8c38a¥sfr_r838a.h

4.2 プログラム

プログラムのゴシック体部分が、「kit12_38a.c」から追加、変更した部分です。

```

1 : /******
2 : /* 対象マイコン R8C/38A */
3 : /* ファイル内容 ロータリエンコーダ搭載マイコンカートレース基本プログラム1 */
4 : /* バージョン Ver. 1. 00 */
5 : /* Date 2013. 04. 22 */
6 : /* Copyright ジャパンマイコンカーラリー実行委員会 */
7 : /******
8 : /*
9 : 本ワークスペースのプログラムは、kit12_38a.cをベースにロータリエンコーダを
10: 搭載し、スピード制御、距離の検出を行うプログラムです。
11: 本プログラムは、マイコンカーのスピードを検出し、
12: 速ければ減速させ、遅ければ加速させます。
13: */
14:
15: /*=====*/
16: /* インクルード */
17: /*=====*/
18: #include "sfr_r838a.h" /* R8C/38A SFRの定義ファイル */
19:
20: /*=====*/
21: /* シンボル定義 */
22: /*=====*/
23:
24: /* 定数設定 */
25: #define PWM_CYCLE 39999 /* モータPWMの周期 */
26: #define SERVO_CENTER 3750 /* サーボのセンタ値 */
27: #define HANDLE_STEP 22 /* 1° 分の値 */
28:

```

4. プロジェクト「kit12enc01_38a」速度の調整

```

29 : /* マスク値設定 × : マスクあり(無効) ○ : マスク無し(有効) */
30 : #define MASK2_2      0x66      /* ×○○××○○×      */
31 : #define MASK2_0      0x60      /* ×○○××××××      */
32 : #define MASK0_2      0x06      /* ×××××○○×      */
33 : #define MASK3_3      0xe7      /* ○○○××○○○      */
34 : #define MASK0_3      0x07      /* ×××××○○○      */
35 : #define MASK3_0      0xe0      /* ○○○×××××      */
36 : #define MASK4_0      0xf0      /* ○○○○××××      */
37 : #define MASK0_4      0x0f      /* ××××○○○○      */
38 : #define MASK4_4      0xff      /* ○○○○○○○○      */
39 :
40 : /*=====*/
41 : /* プロトタイプ宣言 */
42 : /*=====*/
43 : void init( void );
44 : void timer( unsigned long timer_set );
45 : int check_crossline( void );
46 : int check_rightline( void );
47 : int check_leftline( void );
48 : unsigned char sensor_inp( unsigned char mask );
49 : unsigned char dipsw_get( void );
50 : unsigned char pushsw_get( void );
51 : unsigned char startbar_get( void );
52 : void led_out( unsigned char led );
53 : void motor( int accele_l, int accele_r );
54 : void motor2( int accele_l, int accele_r );
55 : void handle( int angle );
56 :
57 : /*=====*/
58 : /* グローバル変数の宣言 */
59 : /*=====*/
60 : unsigned long cnt0; /* timer関数用 */
61 : unsigned long cnt1; /* main内で使用 */
62 : int pattern; /* パターン番号 */
63 :
64 : /* エンコーダ関連 */
65 : int iTimer10; /* エンコーダ取得間隔 */
66 : long lEncoderTotal; /* 積算値 */
67 : int iEncoder; /* 現在値 */
68 : unsigned int uEncoderBuff; /* 前回値保存 */
69 :
70 : /*=====*/
71 : /* メインプログラム */
72 : /*=====*/
73 : void main( void )
74 : {
75 :     int i;
76 :
77 :     /* マイコン機能の初期化 */
78 :     init(); /* 初期化 */
79 :     asm( " fset I " ); /* 全体の割り込み許可 */
80 :
81 :     /* マイコンカーの状態初期化 */
82 :     handle( 0 );
83 :     motor( 0, 0 );
84 :
85 :     while( 1 ) {
86 :         switch( pattern ) {
87 :
88 :             /*=====*/
89 :             パターンについて
90 :             0 : スイッチ入力待ち
91 :             1 : スタートバーが開いたかチェック
92 :             11 : 通常トレース
93 :             12 : 右へ大曲げの終わりのチェック
94 :             13 : 左へ大曲げの終わりのチェック
95 :             21 : クロスライン検出時の処理
96 :             22 : クロスラインを読み飛ばす
97 :             23 : クロスライン後のトレース、クランク検出
98 :             31 : 左クランククリア処理 安定するまで少し待つ
99 :             32 : 左クランククリア処理 曲げ終わりのチェック
100 :             41 : 右クランククリア処理 安定するまで少し待つ
101 :             42 : 右クランククリア処理 曲げ終わりのチェック
102 :             51 : 右ハーフライン検出時の処理
103 :             52 : 右ハーフラインを読み飛ばす
104 :             53 : 右ハーフライン後のトレース、レーンチェンジ
105 :             54 : 右レーンチェンジ終了のチェック
106 :             61 : 左ハーフライン検出時の処理
107 :             62 : 左ハーフラインを読み飛ばす
108 :             63 : 左ハーフライン後のトレース、レーンチェンジ
109 :             64 : 左レーンチェンジ終了のチェック
110 :             /*=====*/
111 :

```

4. プロジェクト「kit12enc01_38a」速度の調整

```

112 :     case 0:
113 :         /* スイッチ入力待ち */
114 :         if( pushsw_get() ) {
115 :             pattern = 1;
116 :             cnt1 = 0;
117 :             break;
118 :         }
119 :
120 :         if( cnt1 >= 200 ) cnt1 = 0;      /* カウンタの上限チェック */
121 :
122 :         if( cnt1 < 100 ) {              /* LED点滅処理 */
123 :             led_out( p3_0 );
124 :         } else {
125 :             led_out( 0x2 | p3_0 );
126 :         }
127 :         break;
128 :
129 :     case 1:
130 :         /* スタートバーが開いたかチェック */
131 :         if( !startbar_get() ) {
132 :             /* スタート!! */
133 :             led_out( 0x0 );
134 :             pattern = 11;
135 :             cnt1 = 0;
136 :             break;
137 :         }
138 :         if( cnt1 < 50 ) {                /* LED点滅処理 */
139 :             led_out( 0x1 );
140 :         } else if( cnt1 < 100 ) {
141 :             led_out( 0x2 );
142 :         } else {
143 :             cnt1 = 0;
144 :         }
145 :         break;
146 :
147 :     case 11:
148 :         /* 通常トレース */
149 :         if( check_crossline() ) {        /* クロスラインチェック */
150 :             pattern = 21;
151 :             break;
152 :         }
153 :         if( check_rightline() ) {        /* 右ハーフラインチェック */
154 :             pattern = 51;
155 :             break;
156 :         }
157 :         if( check_leftline() ) {         /* 左ハーフラインチェック */
158 :             pattern = 61;
159 :             break;
160 :         }
161 :         switch( sensor_inp(MASK3_3) ) {
162 :             case 0x00:
163 :                 /* センタ→まっすぐ */
164 :                 handle( 0 );
165 :                 motor( 100 , 100 );
166 :                 break;
167 :
168 :             case 0x04:
169 :                 /* 微妙に左寄り→右へ微曲げ */
170 :                 handle( 5 );
171 :                 motor( 100 , 100 );
172 :                 break;
173 :
174 :             case 0x06:
175 :                 /* 少し左寄り→右へ小曲げ */
176 :                 handle( 10 );
177 :                 motor( 80 , 67 );
178 :                 break;
179 :
180 :             case 0x07:
181 :                 /* 中くらい左寄り→右へ中曲げ */
182 :                 handle( 15 );
183 :                 motor( 50 , 38 );
184 :                 break;
185 :
186 :             case 0x03:
187 :                 /* 大きく左寄り→右へ大曲げ */
188 :                 handle( 25 );
189 :                 motor( 30 , 19 );
190 :                 pattern = 12;
191 :                 break;
192 :
193 :             case 0x20:
194 :                 /* 微妙に右寄り→左へ微曲げ */
195 :                 handle( -5 );
196 :                 motor( 100 , 100 );
197 :                 break;
198 :

```

4. プロジェクト「kit12enc01_38a」速度の調整

```

199 :         case 0x60:
200 :             /* 少し右寄り→左へ小曲げ */
201 :             handle( -10 );
202 :             motor( 67 ,80 );
203 :             break;
204 :
205 :         case 0xe0:
206 :             /* 中くらい右寄り→左へ中曲げ */
207 :             handle( -15 );
208 :             motor( 38 ,50 );
209 :             break;
210 :
211 :         case 0xc0:
212 :             /* 大きく右寄り→左へ大曲げ */
213 :             handle( -25 );
214 :             motor( 19 ,30 );
215 :             pattern = 13;
216 :             break;
217 :
218 :         default:
219 :             break;
220 :     }
221 :     break;
222 :
223 : case 12:
224 :     /* 右へ大曲げの終わりのチェック */
225 :     if( check_crossline() ) { /* 大曲げ中もクロスラインチェック */
226 :         pattern = 21;
227 :         break;
228 :     }
229 :     if( check_rightline() ) { /* 右ハーフラインチェック */
230 :         pattern = 51;
231 :         break;
232 :     }
233 :     if( check_leftline() ) { /* 左ハーフラインチェック */
234 :         pattern = 61;
235 :         break;
236 :     }
237 :     if( iEncoder >= 11 ) {
238 :         motor2( 0 ,0 );
239 :     } else {
240 :         motor2( 60 ,37 );
241 :     }
242 :     if( sensor_inp(MASK3_3) == 0x06 ) {
243 :         pattern = 11;
244 :     }
245 :     break;
246 :
247 : case 13:
248 :     /* 左へ大曲げの終わりのチェック */
249 :     if( check_crossline() ) { /* 大曲げ中もクロスラインチェック */
250 :         pattern = 21;
251 :         break;
252 :     }
253 :     if( check_rightline() ) { /* 右ハーフラインチェック */
254 :         pattern = 51;
255 :         break;
256 :     }
257 :     if( check_leftline() ) { /* 左ハーフラインチェック */
258 :         pattern = 61;
259 :         break;
260 :     }
261 :     if( iEncoder >= 11 ) {
262 :         motor2( 0 ,0 );
263 :     } else {
264 :         motor2( 37 ,60 );
265 :     }
266 :     if( sensor_inp(MASK3_3) == 0x60 ) {
267 :         pattern = 11;
268 :     }
269 :     break;
270 :
271 : case 21:
272 :     /* クロスライン検出時の処理 */
273 :     led_out( 0x3 );
274 :     handle( 0 );
275 :     motor( 0 ,0 );
276 :     pattern = 22;
277 :     cnt1 = 0;
278 :     break;
279 :
280 : case 22:
281 :     /* クロスラインを読み飛ばす */
282 :     if( cnt1 > 100 ) {
283 :         pattern = 23;
284 :         cnt1 = 0;
285 :     }
286 :     break;
287 :

```

4. プロジェクト「kit12enc01_38a」速度の調整

```

288 :     case 23:
289 :         /* クロスライン後のトレース、クランク検出 */
290 :         if( sensor_inp(MASK4_4)==0xf8 ) {
291 :             /* 左クランクと判断→左クランククリア処理へ */
292 :             led_out( 0x1 );
293 :             handle( -38 );
294 :             motor( 10 ,50 );
295 :             pattern = 31;
296 :             cnt1 = 0;
297 :             break;
298 :         }
299 :         if( sensor_inp(MASK4_4)==0x1f ) {
300 :             /* 右クランクと判断→右クランククリア処理へ */
301 :             led_out( 0x2 );
302 :             handle( 38 );
303 :             motor( 50 ,10 );
304 :             pattern = 41;
305 :             cnt1 = 0;
306 :             break;
307 :         }
308 :         if( iEncoder >= 11 ) {      /* クロスライン後のスピード制御 */
309 :             motor2( 0 ,0 );
310 :         } else {
311 :             motor2( 70 ,70 );
312 :         }
313 :         switch( sensor_inp(MASK3_3) ) {
314 :             case 0x00:
315 :                 /* センタ→まっすぐ */
316 :                 handle( 0 );
317 :                 break;
318 :             case 0x04:
319 :             case 0x06:
320 :             case 0x07:
321 :             case 0x03:
322 :                 /* 左寄り→右曲げ */
323 :                 handle( 8 );
324 :                 break;
325 :             case 0x20:
326 :             case 0x60:
327 :             case 0xe0:
328 :             case 0xc0:
329 :                 /* 右寄り→左曲げ */
330 :                 handle( -8 );
331 :                 break;
332 :             }
333 :         break;
334 :
335 :     case 31:
336 :         /* 左クランククリア処理 安定するまで少し待つ */
337 :         if( cnt1 > 200 ) {
338 :             pattern = 32;
339 :             cnt1 = 0;
340 :         }
341 :         break;
342 :
343 :     case 32:
344 :         /* 左クランククリア処理 曲げ終わりのチェック */
345 :         if( sensor_inp(MASK3_3) == 0x60 ) {
346 :             led_out( 0x0 );
347 :             pattern = 11;
348 :             cnt1 = 0;
349 :         }
350 :         break;
351 :
352 :     case 41:
353 :         /* 右クランククリア処理 安定するまで少し待つ */
354 :         if( cnt1 > 200 ) {
355 :             pattern = 42;
356 :             cnt1 = 0;
357 :         }
358 :         break;
359 :
360 :     case 42:
361 :         /* 右クランククリア処理 曲げ終わりのチェック */
362 :         if( sensor_inp(MASK3_3) == 0x06 ) {
363 :             led_out( 0x0 );
364 :             pattern = 11;
365 :             cnt1 = 0;
366 :         }
367 :         break;
368 :
369 :     case 51:
370 :         /* 右ハーフライン検出時の処理 */
371 :         led_out( 0x2 );
372 :         handle( 0 );
373 :         motor( 0 ,0 );
374 :         pattern = 52;
375 :         cnt1 = 0;
376 :         break;
377 :

```

4. プロジェクト「kit12enc01_38a」速度の調整

```

378 :     case 52:
379 :         /* 右ハーフラインを読み飛ばす */
380 :         if( cnt1 > 100 ) {
381 :             pattern = 53;
382 :             cnt1 = 0;
383 :         }
384 :         break;
385 :
386 :     case 53:
387 :         /* 右ハーフライン後のトレース、レーンチェンジ */
388 :         if( sensor_inp(MASK4_4) == 0x00 ) {
389 :             handle( 15 );
390 :             motor( 40 , 31 );
391 :             pattern = 54;
392 :             cnt1 = 0;
393 :             break;
394 :         }
395 :         if( iEncoder >= 11 ) {      /* ハーフライン後のスピード制御 */
396 :             motor2( 0 , 0 );
397 :         } else {
398 :             motor2( 70 , 70 );
399 :         }
400 :         switch( sensor_inp(MASK3_3) ) {
401 :             case 0x00:
402 :                 /* センタ→まっすぐ */
403 :                 handle( 0 );
404 :                 motor( 40 , 40 );
405 :                 break;
406 :             case 0x04:
407 :             case 0x06:
408 :             case 0x07:
409 :             case 0x03:
410 :                 /* 左寄り→右曲げ */
411 :                 handle( 8 );
412 :                 motor( 40 , 35 );
413 :                 break;
414 :             case 0x20:
415 :             case 0x60:
416 :             case 0xe0:
417 :             case 0xc0:
418 :                 /* 右寄り→左曲げ */
419 :                 handle( -8 );
420 :                 motor( 35 , 40 );
421 :                 break;
422 :             default:
423 :                 break;
424 :         }
425 :         break;
426 :
427 :     case 54:
428 :         /* 右レーンチェンジ終了のチェック */
429 :         if( sensor_inp( MASK4_4 ) == 0x3c ) {
430 :             led_out( 0x0 );
431 :             pattern = 11;
432 :             cnt1 = 0;
433 :         }
434 :         break;
435 :
436 :     case 61:
437 :         /* 左ハーフライン検出時の処理 */
438 :         led_out( 0x1 );
439 :         handle( 0 );
440 :         motor( 0 , 0 );
441 :         pattern = 62;
442 :         cnt1 = 0;
443 :         break;
444 :
445 :     case 62:
446 :         /* 左ハーフラインを読み飛ばす */
447 :         if( cnt1 > 100 ) {
448 :             pattern = 63;
449 :             cnt1 = 0;
450 :         }
451 :         break;
452 :

```


4. プロジェクト「kit12enc01_38a」速度の調整

```

453 :     case 63:
454 :         /* 左ハーフライン後のトレース、レーンチェンジ */
455 :         if( sensor_inp(MASK4_4) == 0x00 ) {
456 :             handle( -15 );
457 :             motor( 31 , 40 );
458 :             pattern = 64;
459 :             cnt1 = 0;
460 :             break;
461 :         }
462 :         if( iEncoder >= 11 ) { /* ハーフラインライン後のスピード制御 */
463 :             motor2( 0 , 0 );
464 :         } else {
465 :             motor2( 70 , 70 );
466 :         }
467 :         switch( sensor_inp(MASK3_3) ) {
468 :             case 0x00:
469 :                 /* センタ→まっすぐ */
470 :                 handle( 0 );
471 :                 motor( 40 , 40 );
472 :                 break;
473 :             case 0x04:
474 :             case 0x06:
475 :             case 0x07:
476 :             case 0x03:
477 :                 /* 左寄り→右曲げ */
478 :                 handle( 8 );
479 :                 motor( 40 , 35 );
480 :                 break;
481 :             case 0x20:
482 :             case 0x60:
483 :             case 0xe0:
484 :             case 0xc0:
485 :                 /* 右寄り→左曲げ */
486 :                 handle( -8 );
487 :                 motor( 35 , 40 );
488 :                 break;
489 :             default:
490 :                 break;
491 :         }
492 :         break;
493 :
494 :     case 64:
495 :         /* 左レーンチェンジ終了のチェック */
496 :         if( sensor_inp( MASK4_4 ) == 0x3c ) {
497 :             led_out( 0x0 );
498 :             pattern = 11;
499 :             cnt1 = 0;
500 :         }
501 :         break;
502 :
503 :     default:
504 :         /* どれでもない場合は待機状態に戻す */
505 :         pattern = 0;
506 :         break;
507 :     }
508 : }
509 : }
510 :
511 : /*****
512 : /* R8C/38A スペシャルファンクションレジスタ(SFR)の初期化 */
513 : *****/
514 : void init( void )
515 : {
516 :     int i;
517 :
518 :     /* クロックをXINクロック(20MHz)に変更 */
519 :     prc0 = 1; /* プロテクト解除 */
520 :     cm13 = 1; /* P4_6, P4_7をXIN-XOUT端子にする*/
521 :     cm05 = 0; /* XINクロック発振 */
522 :     for( i=0; i<50; i++ ); /* 安定するまで少し待つ(約10ms) */
523 :     ocd2 = 0; /* システムクロックをXINにする */
524 :     prc0 = 0; /* プロテクトON */
525 :
526 :     /* ポートの入出力設定 */
527 :     prc2 = 1; /* PD0のプロテクト解除 */
528 :     pd0 = 0x00; /* 7-0:センサ基板Ver. 5 */
529 :     pd1 = 0xd0; /* 5:RXD0 4:TXD0 3-0:DIP SW */
530 :     p2 = 0xc0;
531 :     pd2 = 0xfe; /* 7-0:モータドライブ基板Ver. 5 */
532 :     pd3 = 0xfe; /* 0:ロータリエンコーダ */
533 :     p4 = 0x20; /* P4_5のLED:初期は点灯 */
534 :     pd4 = 0xb8; /* 7:XOUT 6:XIN 5:LED 2:VREF */
535 :     pd5 = 0xff; /*
536 :     pd6 = 0xff; /*
537 :     pd7 = 0xff; /*
538 :     pd8 = 0xff; /*
539 :     pd9 = 0x3f; /*
540 :     pur0 = 0x04; /* P1_3~P1_0のプルアップON */
541 :

```

4. プロジェクト「kit12enc01_38a」速度の調整

```

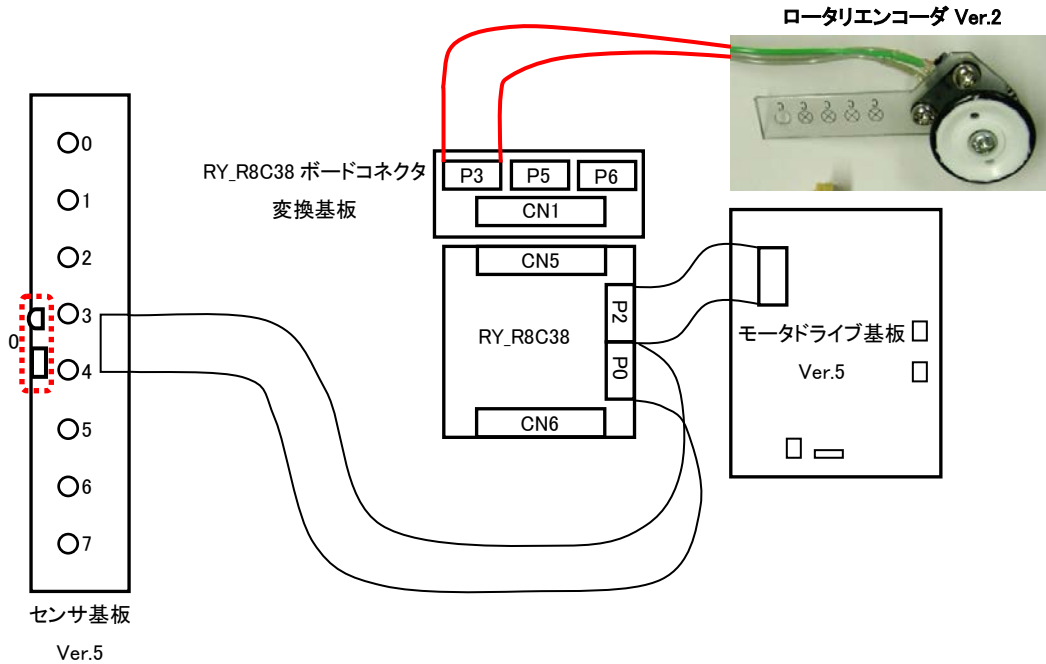
542 :      /* タイマRBの設定 */
543 :      /* 割り込み周期 = 1 / 20[MHz] * (TRBPRE+1) * (TRBPR+1)
544 :                  = 1 / (20*106) * 200 * 100
545 :                  = 0.001[s] = 1[ms]
546 :      */
547 :      trbmr = 0x00;          /* 動作モード、分周比設定 */
548 :      trbpre = 200-1;      /* プリスケーラレジスタ */
549 :      trbpr = 100-1;      /* プライマリレジスタ */
550 :      trbic = 0x07;       /* 割り込み優先レベル設定 */
551 :      trbcr = 0x01;       /* カウント開始 */
552 :
553 :      /* タイマRD リセット同期PWMモードの設定*/
554 :      /* PWM周期 = 1 / 20[MHz] * カウントソース * (TRDGRA0+1)
555 :                  = 1 / (20*106) * 8 * 40000
556 :                  = 0.016[s] = 16[ms]
557 :      */
558 :      trdpsr0 = 0x08;      /* TRDIOB0, C0, D0端子設定 */
559 :      trdpsr1 = 0x05;      /* TRDIOA1, B1, C1, D1端子設定 */
560 :      trdmr = 0xf0;        /* バッファレジスタ設定 */
561 :      trdfcr = 0x01;       /* リセット同期PWMモードに設定 */
562 :      trdcr0 = 0x23;       /* ソースカウン트의選択:f8 */
563 :      trdgra0 = trdgrc0 = PWM_CYCLE; /* 周期 */
564 :      trdgrb0 = trdgrd0 = 0; /* P2_2端子のON幅設定 */
565 :      trdgral = trdgrcl = 0; /* P2_4端子のON幅設定 */
566 :      trdgrbl = trdgrdl = SERVO_CENTER; /* P2_5端子のON幅設定 */
567 :      trdoerl = 0xcd;      /* 出力端子の選択 */
568 :      trdstr = 0x0d;       /* TRD0カウント開始 */
569 :
570 :      /* タイマRG タイマモード(両エッジでカウント)の設定 */
571 :      timsr = 0x40;        /* TRGCLKA端子 P3_0に割り当てる */
572 :      trgcr = 0x15;        /* TRGCLKA端子の両エッジでカウント*/
573 :      trgmr = 0x80;        /* TRGのカウント開始 */
574 : }
575 :
576 : /*****
577 : /* タイマRB 割り込み処理
578 : /*****
579 : #pragma interrupt intTRB(vect=24)
580 : void intTRB( void )
581 : {
582 :     unsigned int i;
583 :
584 :     cnt0++;
585 :     cnt1++;
586 :
587 :     /* エンコーダ関連処理 */
588 :     iTimer10++;
589 :     if( iTimer10 >= 10 ) {
590 :         iTimer10 = 0;
591 :         i = trg;
592 :         iEncoder = i - uEncoderBuff;
593 :         lEncoderTotal += iEncoder;
594 :         uEncoderBuff = i;
595 :     }
596 : }

```

以下、略

4.3 ロータリエンコーダの接続

ポート3のbit0にロータリエンコーダ Ver.2を接続します。ポート3は26ピンコネクタなので、RY_R8C38 ボードコネクタ変換基板などを使用して接続します。



ポート3の接続は下記のようになります。

ピン番号	信号名	接続先	マイコンから見た方向
1	+5V	+5V	
2	P3_7		出力
3	P3_6		出力
4	P3_5		出力
5	P3_4		出力
6	P3_3		出力
7	P3_2		出力
8	P3_1		出力
9	P3_0	ロータリエンコーダ	入力
10	GND	GND	

PD3の入出力設定を下表に示します。

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

PD3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3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 ロータリエンコーダ関連の変数の宣言

64 :	/* エンコーダ関連 */		
65 :	int	iTimer10;	/* エンコーダ取得間隔 */
66 :	long	lEncoderTotal;	/* 積算値 */
67 :	int	iEncoder;	/* 現在値 */
68 :	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	ロータリエンコーダ値の更新は、intTRB 関数内で行います。intTRB 関数は 1ms ごとに実行されますが、ロータリエンコーダ処理は 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	積算値	スタートしてからのロータリ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。
iEncoder	10ms ごとの現在値	10ms ごとに更新されるロータリ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	タイマ RG カウンタ (TRG) の前回の値を保存しています。main 関数では使用しません。

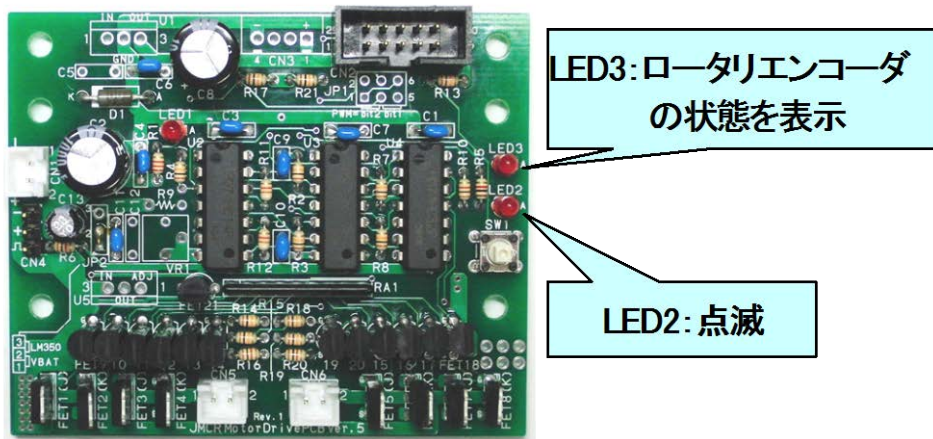
これらの変数は、割り込みプログラム内で、10ms ごとに更新されます。詳しくは割り込みで説明します。ちなみに、これらの変数は初期値のないグローバル変数なので、初期値は 0 です。

4.4.2 パターン 0: ロータリエンコーダの状態をモータドライブ基板の LED へ出力

112 :	case 0:		
113 :	/* スイッチ入力待ち */		
114 :	if(pushsw_get()) {		
115 :	pattern = 1;		
116 :	cnt1 = 0;		
117 :	break;		
118 :	}		
119 :			
120 :	if(cnt1 >= 200) cnt1 = 0;	/* カウンタの上限チェック	*/
121 :			
122 :	if(cnt1 < 100) {	/* LED点滅処理	*/
123 :	led_out(p3_0);		
124 :	} else {		
125 :	led_out(0x2 p3_0);		
126 :	}		
127 :	break;		

モータドライブ基板 Ver.5 の LED3 にロータリエンコーダの状態を出力します。LED2 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. プロジェクト「kit12enc01_38a」速度の調整



4.4.3 入出力設定の変更

```

511 : /*****/
512 : /* R8C/38A スペシャルファンクションレジスタ (SFR) の初期化 */
513 : /*****/
514 : void init( void )
515 : {
516 :     int i;
517 :
518 :     /* クロックをXINクロック (20MHz)に変更 */
519 :     prc0 = 1; /* プロテクト解除 */
520 :     cm13 = 1; /* P4_6, P4_7をXIN-XOUT端子にする*/
521 :     cm05 = 0; /* XINクロック発振 */
522 :     for(i=0; i<50; i++ ); /* 安定するまで少し待つ(約10ms) */
523 :     ocd2 = 0; /* システムクロックをXINにする */
524 :     prc0 = 0; /* プロテクトON */
525 :
526 :     /* ポートの入出力設定 */
527 :     prc2 = 1; /* PD0のプロテクト解除 */
528 :     pd0 = 0x00; /* 7-0:センサ基板Ver. 5 */
529 :     pd1 = 0xd0; /* 5:RXD0 4:TXD0 3-0:DIP SW */
530 :     p2 = 0xc0;
531 :     pd2 = 0xfe; /* 7-0:モータドライブ基板Ver. 5 */
532 :     pd3 = 0xfe; /* 0:ロータリエンコーダ */
533 :     p4 = 0x20; /* P4_5のLED:初期は点灯 */
534 :     pd4 = 0xb8; /* 7:XOUT 6:XIN 5:LED 2:VREF */
535 :     pd5 = 0xff; /* */
536 :     pd6 = 0xff; /* */
537 :     pd7 = 0xff; /* */
538 :     pd8 = 0xff; /* */
539 :     pd9 = 0x3f; /* */
540 :     pur0 = 0x04; /* P1_3~P1_0のプルアップON */

```

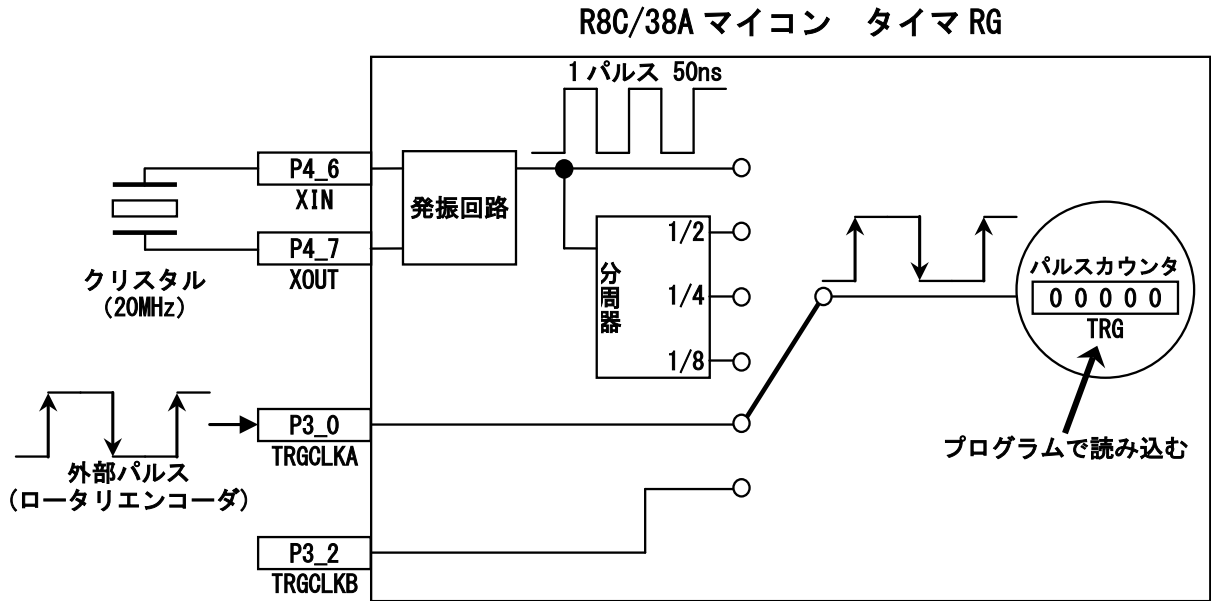
PD3 の bit0 は、ロータリエンコーダのパルス入力になったので、0xff から 0xfe へ変更します。

4.4.4 外部パルス入力設定

```

570 : /* タイマRG タイマモード(両エッジでカウント)の設定 */
571 : timsr = 0x40; /* TRGCLKA端子 P3_0に割り当てる */
572 : trgcr = 0x15; /* TRGCLKA端子の両エッジでカウント*/
573 : trgmr = 0x80; /* TRGのカウント開始 */
    
```

タイマ RG 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。タイマ RG を使ったパルスカウントのイメージを下図に示します。



- ① 外部からのパルスは、P3_0 端子か P3_2 端子から入力することができます。今回は、P3.0 端子を使用します。
- ② パルスが入力されると、TRG というレジスタの値が増えていきます。このときのパルスは、立ち上がりでカウントするか、立ち下がりでカウントするか、両方でカウントするか選ぶことができます。今回は両方でカウントするようにします。

これから、レジスタの設定について説明します。

4. プロジェクト「kit12enc01_38a」速度の調整

①タイマ端子選択レジスタ(TIMSR:Timer Pin Select Register)の設定

タイマを使うときの端子を設定します。今回は、TRGCLKA 端子を P3_0 に割り当てます。

設定 bit	上:ビット名 下:シンボル	内容	今回の内容
bit7	TRGCLKB 端子選択ビット trgclkbsel	0:割り当てない 1:P3_2 に割り当てる 今回は、割り当てません。P3_2 端子でパルスカウントする場合は、この bit を設定してください。	0
bit6	TRGCLKA 端子選択ビット trgclkasel	0:割り当てない 1:P3_0 に割り当てる 今回は、P3_0 を使用するので"1"を設定します。	1
bit5	TRGIOB 端子選択ビット trgiobsel	0:割り当てない 1:P5_7 に割り当てる 使いません。	0
bit4	TRGIOA 端子選択ビット trgioasel	0:割り当てない 1:P5_6 に割り当てる 使いません。	0
bit3		"0"を設定	0
bit2	TRFI 端子選択ビット trfisel0	0:割り当てない 1:P8_3 に割り当てる 使いません。	0
bit1		"0"を設定	0
bit0	TREO 端子選択ビット treosel0	0:P0_4 に割り当てる 1:P6_0 に割り当てる 使いません。	0

タイマ端子選択レジスタ(TIMSR)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	1	0	0	0	0	0	0
16進数	4				0			

4. プロジェクト「kit12enc01_38a」速度の調整

②タイマ RG 制御レジスタ(TRGCR: Timer RG Control Register)の設定

カウントソースやエッジのカウント方法を設定します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7		"0"を設定	0
bit6,5	TRG レジスタクリア要因選択 ビット bit6:cclr1_trgcr bit5:cclr0_trgcr	00:クリア禁止 01:インプットキャプチャまたは TRGGRA のコンペアー 致で TRG レジスタをクリア 10:インプットキャプチャまたは TRGGRB のコンペアー 致で TRG レジスタをクリア 11:設定しないでください クリアしません。"00"を設定します。	00
bit4,3	外部クロック有効エッジ選択 ビット(注 1) bit4:ckeg1_trgcr bit3:ckeg0_trgcr	00:立ち上がりエッジでカウント 01:立ち下がりエッジでカウント 10:立ち上がり/立ち下がりの両エッジでカウント 11:設定しないでください 両エッジでカウントします。"10"を設定します。	10
bit2~0	カウントソース選択ビット(注 1) bit2:tck2_trgcr bit1:tck1_trgcr bit0:tck0_trgcr	000:f1 (1/20MHz=50ns) 001:f2 (2/20MHz=100ns) 010:f4 (4/20MHz=200ns) 011:f8 (8/20MHz=400ns) 100:f32 (32/20MHz=1600ns) 101:TRGCLKA 入力(P3_0 端子から入力) 110:fOCO40M 111:TRGCLKB 入力(P3_2 端子から入力) P3_0 端子から入力するので"101"を設定します。	101

注 1. 位相計数モードのとき、TCK0~TCK2ビット、および CKEG0~CKEG1ビット設定は無効になり、位相計数モードの動作が優先されます。

タイマ RG 制御レジスタ(TRGCR)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	1	0	1	0	1
16 進数	1				5			

4. プロジェクト「kit12enc01_38a」速度の調整

③タイマ RG モードレジスタ(TRGMR:Timer RG Mode Register)の設定

タイマ RG の使用モード、カウントの開始を設定します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7	TRG カウント開始ビット tstart_trgmr	0:カウント停止 1:カウント開始 TRG(タイマ RG カウンタ)のカウント開始します。"1"を設定します。	1
bit6		"0"を設定	0
bit5,4	デジタルフィルタ機能で使用するクロック選択ビット bit5:dfck1_trgmr bit4:dfck0_trgmr	00:f32 01:f8 10:f1 11:TRGCR レジスタの TCK0~2 で選択したクロック デジタルフィルタ機能は使用しませんのでどれを設定しても構いませんが、"00"を設定しておきます。	00
bit3	TRGIOB 端子のデジタルフィルタ機能選択ビット dfb_trgmr	0:デジタルフィルタ機能なし 1:デジタルフィルタ機能あり デジタルフィルタ機能は使いません。	0
bit2	TRGIOA 端子のデジタルフィルタ機能選択ビット dfa_trgmr	0:デジタルフィルタ機能なし 1:デジタルフィルタ機能あり デジタルフィルタ機能は使いません。	0
bit1	位相計数モード選択ビット mdf_trgmr	0:アップカウント 1:位相計数モード アップカウントです。"0"を設定します。	0
bit0	PWM モード選択ビット pwm_trgmr	0:タイマモード 1:PWM モード タイマモードで使用します。"0"を設定します。	0

タイマ RG モードレジスタ(TRGMR)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	1	0	0	0	0	0	0	0
16 進数	8				0			

4.4.5 円盤の黒、透明(白)の間隔が違うとき

立ち上がり／立ち下がり両エッジ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「trgcr = 0x15」にできます

間隔が違う場合は、立ち上がりのみにします。

```

570 :      /* タイマRG タイマモード(両エッジでカウント)の設定 */
571 :      timsr = 0x40;                               /* TRGCLKA端子 P3_0に割り当てる */
572 :      trgcr = 0x05;                             /* TRGCLKA端子の立ち上がりエッジカウント*/
573 :      trgmr = 0x80;                               /* TRGのカウント開始          */
    
```

4.4.6 タイマ RB 割り込み処理

```

579 : #pragma interrupt intTRB(vect=24)
580 : void intTRB( void )
581 : {
582 :     unsigned int i;
583 :
584 :     cnt0++;
585 :     cnt1++;
586 :
587 :     /* エンコーダ関連処理 */
588 :     iTimer10++;
589 :     if( iTimer10 >= 10 ) {
590 :         iTimer10 = 0;
591 :         i = trg;
592 :         iEncoder      = i - uEncoderBuff;
593 :         lEncoderTotal += iEncoder;
594 :         uEncoderBuff = i;
595 :     }
596 : }
    
```

588 行	iTimer10 変数を増加させます。
589 行	iTimer10 変数が 10 以上なら次の行を実行します。TRB 割り込みは、1ms ごとに実行されますが、ロータリ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ロータリエンコーダ処理を行います。それ以下なら 595 行へ移りロータリエンコーダ処理をしません。
590 行	iTimer10 変数を 0 にして、実行回数を数え直します。
591 行	現在のカウント値 TRG を変数 i に代入します。なぜ、TRG の値を直接使わないのでしょうか。TRG の値は、ロータリ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 i 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。

4. プロジェクト「kit12enc01_38a」速度の調整

592 行	<p>最新の 10ms 間のロータリエンコーダのカウンタ数を計算しています。計算は、</p> $10\text{ms 間のロータリエンコーダのカウンタ数} = i - u\text{EncoderBuff}$ <p>としています。i は現在のカウンタ値、uEncoderBuff は前回(10ms 前)のカウンタ値です。言い換えれば、10ms 間のロータリエンコーダのカウンタ数 = 現在のカウンタ値 - 1 回前のカウンタ値となります。タイマ RG カウンタ (TRG) 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。</p> <p>図解すると下記のようなイメージです。</p> <div style="text-align: center;"> </div>
593 行	<p>ロータリエンコーダの積算値を計算しています。計算は、</p> $\text{積算値} = \text{積算値} + \text{最新の 10ms 間のロータリエンコーダ値}$ <p>です。積算値は、long 型ですので、21 億回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。</p>
594 行	<p>i には現在のタイマRGカウンタ (TRG) 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 変数の値は最新値を代入したことになりますが、次にロータリ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 変数は 10ms 前の値となります。</p>

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、下記のようなプログラムで回数を数えて 10 回目で実行します。

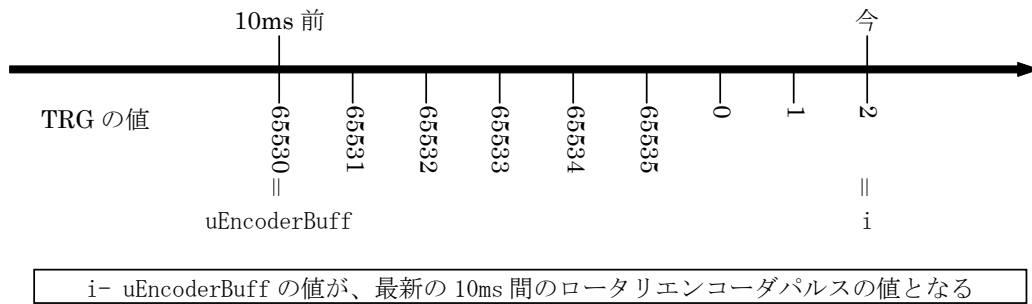
```

588 :     iTimer10++;
589 :     if( iTimer10 >= 10 ) {

```

結果、10ms ごとにロータリ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。今回のプログラムは 10ms ごとにカウントさせています。

4.4.8 タイマ RG カウンタ(TRG)が 65535 から 0 になったとき



タイマ RG カウンタ(TRG)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、

$$(\text{現在の TRG}) - (\text{10ms 前の TRG})$$

です。図のように、10ms 前の TRG 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、

$$(\text{現在の TRG}) - (\text{10ms 前の TRG}) = 2 - 65530 = -65528$$

となり、とんでもない値になります。

16 進数に直すと、

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

となります。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ため、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

タイマ RG カウンタ(TRG)がロータリ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 intTRB(vect=24)
void intTRB( void )
{
    unsigned int i;

    cnt0++;
    cnt1++;

    /* エンコーダ関連処理 */
    iTimer10++;
    if( iTimer10 >= 10 ) {
        iTimer10 = 0;
        iEncoder = trg;
        trg = 0;
    }
}
```

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。ただしこの場合は、入力されたパルスを見逃してしまう場合があります。iEncoder という変数にパルスを代入して、すぐにタイマ RG カウンタ(TRG)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 = trg;          /* 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 trg の値が1つ増えた
    trg = 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**。パルス1つ分ですが、もしタイマ RG カウンタ(TRG)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、少しわかりづらいですが、バッファを使用したプログラムで処理しています。

4.4.10 パターン 12 右大曲げ時の処理

```

223 :     case 12:
224 :         /* 右へ大曲げの終わりのチェック */
225 :         if( check_crossline() ) {          /* 大曲げ中もクロスラインチェック */
226 :             pattern = 21;
227 :             break;
228 :         }
229 :         if( check_rightline() ) {          /* 右ハーフラインチェック      */
230 :             pattern = 51;
231 :             break;
232 :         }
233 :         if( check_leftline() ) {          /* 左ハーフラインチェック      */
234 :             pattern = 61;
235 :             break;
236 :         }
237 :         if( iEncoder >= 11 ) {
238 :             motor2( 0 , 0 );
239 :         } else {
240 :             motor2( 60 , 37 );
241 :         }
242 :         if( sensor_inp(MASK3_3) == 0x06 ) {
243 :             pattern = 11;
244 :         }
245 :         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、

$$\begin{aligned}
 10\text{ms 間のパルス数} &= \text{現在の速度} \times 10.92 \\
 &= 2[\text{m/s}] \times 10.92 \\
 &= 21.84 \\
 &\simeq 22 \quad \text{※小数点は使えないので四捨五入}
 \end{aligned}$$

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	タイムロスが少ない	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。

4.4.11 motor2 関数

motor2 関数を良く見ると… 「motor~~2~~」と、2 が付いています。

```

767 : /*****/
768 : /* モータ速度制御2(ディップスイッチは関係なし) */
769 : /* 引数 左モータ:-100~100、右モータ:-100~100 */
770 : /*      0で停止、100で正転100%、-100で逆転100% */
771 : /* 戻り値 なし */
772 : /*****/
773 : void motor2( int accele_l, int accele_r )
774 : {
775 :     /* 左モータ制御 */
776 :     if( accele_l >= 0 ) {
777 :         p2 &= 0xfd;
778 :         trdgrd0 = (long)( PWM_CYCLE - 1 ) * accele_l / 100;
779 :     } else {
780 :         p2 |= 0x02;
781 :         trdgrd0 = (long)( PWM_CYCLE - 1 ) * ( -accele_l ) / 100;
782 :     }
783 :
784 :     /* 右モータ制御 */
785 :     if( accele_r >= 0 ) {
786 :         p2 &= 0xf7;
787 :         trdgrc1 = (long)( PWM_CYCLE - 1 ) * accele_r / 100;
788 :     } else {
789 :         p2 |= 0x08;
790 :         trdgrc1 = (long)( PWM_CYCLE - 1 ) * ( -accele_r ) / 100;
791 :     }
792 : }

```

motor 関数を実行したとき、実際にモータに出力される PWM 値は、

実際にモータに出力される PWM 値 = motor 関数の引数の割合 × (ディップスイッチの値 + 5) ÷ 20

でした。ロータリ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、motor 関数の引数そのものがモータに出力される motor2 関数を作りました**。ロータリエンコーダ値を比較してスピード制御する部分には、motor2 関数を使用します。

motor2 関数を実行したとき、実際にモータに出力される PWM 値は、

実際にモータに出力される PWM 値 = motor 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。

4.4.12 パターン 13 左大曲げ時の処理

```
247 :     case 13:
248 :         /* 左へ大曲げの終わりのチェック */
249 :         if( check_crossline() ) {          /* 大曲げ中もクロスラインチェック */
250 :             pattern = 21;
251 :             break;
252 :         }
253 :         if( check_rightline() ) {          /* 右ハーフラインチェック      */
254 :             pattern = 51;
255 :             break;
256 :         }
257 :         if( check_leftline() ) {          /* 左ハーフラインチェック      */
258 :             pattern = 61;
259 :             break;
260 :         }
261 :         if( iEncoder >= 11 ) {
262 :             motor2( 0 , 0 );
263 :         } else {
264 :             motor2( 37 , 60 );
265 :         }
266 :         if( sensor_inp(MASK3_3) == 0x60 ) {
267 :             pattern = 11;
268 :         }
269 :         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 motor2 関数を使用します。

4.4.13 パターン 23 クロスライン後のトレース、クランク検出時の処理

```

288 :     case 23:
289 :         /* クロスライン後のトレース、クランク検出 */
290 :         if( sensor_inp(MASK4_4)==0xf8 ) {
291 :             /* 左クランクと判断→左クランククリア処理へ */
292 :             led_out( 0x1 );
293 :             handle( -38 );
294 :             motor( 10 ,50 );
295 :             pattern = 31;
296 :             cnt1 = 0;
297 :             break;
298 :         }
299 :         if( sensor_inp(MASK4_4)==0x1f ) {
300 :             /* 右クランクと判断→右クランククリア処理へ */
301 :             led_out( 0x2 );
302 :             handle( 38 );
303 :             motor( 50 ,10 );
304 :             pattern = 41;
305 :             cnt1 = 0;
306 :             break;
307 :         }
308 :         if( iEncoder >= 11 ) {      /* クロスライン後のスピード制御 */
309 :             motor2( 0 ,0 );
310 :         } else {
311 :             motor2( 70 ,70 );
312 :         }
313 :         switch( sensor_inp(MASK3_3) ) {
314 :             case 0x00:
315 :                 /* センタ→まっすぐ */
316 :                 handle( 0 );
317 :                 break;
318 :             case 0x04:
319 :             case 0x06:
320 :             case 0x07:
321 :             case 0x03:
322 :                 /* 左寄り→右曲げ */
323 :                 handle( 8 );
324 :                 break;
325 :             case 0x20:
326 :             case 0x60:
327 :             case 0xe0:
328 :             case 0xc0:
329 :                 /* 右寄り→左曲げ */
330 :                 handle( -8 );
331 :                 break;
332 :         }
333 :         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のですが、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。

モータドライブ基板 Ver.5 は逆転も可能です。ブレーキ(PWM0%)だけでスピードが落ちきらない場合は、逆転ブレーキで急減速すると良いでしょう。ただし、ロータリ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 63 でのスピード調整も同様です。

4.5 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、タイヤ直径 21mm のロータリエンコーダを使用した場合です。条件が違ふとき、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
26	3750	それぞれのマイコンカーのサーボセンタ値にします。
237	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。
261	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。
293	-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。
302	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。
308	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。
395	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。
462	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。
572	0x15	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、「 0x05 」にします。

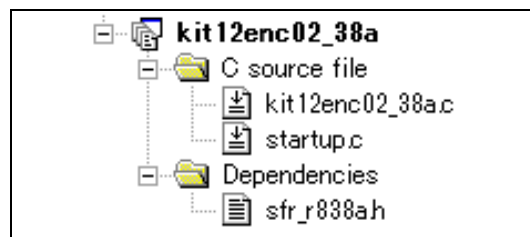
5. プロジェクト「kit12enc02_38a」 距離の検出(パターンの区分けを距離で行う)

kit12_38a 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 間センサを見ません。100ms 後は、クロスラインが終わった直後と仮定しています。しかし、マイコンカーのスピードの違いで進む距離が変わってしまいます。パターン 42、パターン 52 も同様です。

そこで、距離を検出できるロータリ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 進む**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ことがありませんので、安定して走行することができます。

5.1 プロジェクトの構成



	ファイル名	内容
1	kit12enc02_38a.c	実際に制御するプログラムが書かれています。R8C/38A の内蔵周辺機能(SFR)の初期化も行います。 ファイルの位置→C:\¥WorkSpace¥kit12enc_38a¥kit12enc02_38a¥kit12enc02_38a.c
2	startup.c	固定割り込みベクタアドレスの設定、スタートアッププログラム、RAMの初期化(初期値のないグローバル変数、初期値のあるグローバル変数の設定)などを行います。 ファイルの位置→C:\¥WorkSpace¥kit12enc_38a¥kit12enc02_38a¥startup.c
3	sfr_r838a.h	R8C/38A マイコンの内蔵周辺機能を制御するためのレジスタ(Special Function Register)を定義したファイルです。 ファイルの位置→C:\¥WorkSpace¥common_r8c38a¥sfr_r838a.h

5.2 プログラム

プログラムのゴシック体部分が、「kit12enc01_38a.c」から追加した部分です。

前略

```

57 : /*=====*/
58 : /* グローバル変数の宣言 */
59 : /*=====*/
60 : unsigned long cnt0; /* timer関数用 */
61 : unsigned long cnt1; /* main内で使用 */
62 : int pattern; /* パターン番号 */
63 :
64 : /* エンコーダ関連 */
65 : int iTimer10; /* エンコーダ取得間隔 */
66 : long lEncoderTotal; /* 積算値 */
67 : int iEncoder; /* 現在値 */
68 : unsigned int uEncoderBuff; /* 前回値保存 */
69 : long lEncoderLine; /* ライン検出時の積算値 */

```

中略

5. プロジェクト「kit12enc02_38a」 距離の検出(パターンの区分けを距離で行う)

```

130 :     case 1:
131 :         /* スタートバーが開いたかチェック */
132 :         if( !startbar_get() ) {
133 :             /* スタート!! */
134 :             IEncoderTotal = 0;
135 :             led_out( 0x0 );
136 :             pattern = 11;
137 :             cnt1 = 0;
138 :             break;
139 :         }
140 :         if( cnt1 < 50 ) {             /* LED点滅処理 */
141 :             led_out( 0x1 );
142 :         } else if( cnt1 < 100 ) {
143 :             led_out( 0x2 );
144 :         } else {
145 :             cnt1 = 0;
146 :         }
147 :         break;

```

中略

```

273 :     case 21:
274 :         /* クロスライン検出時の処理 */
275 :         IEncoderLine = IEncoderTotal;
276 :         led_out( 0x3 );
277 :         handle( 0 );
278 :         motor( 0, 0 );
279 :         pattern = 22;
280 :         cnt1 = 0;
281 :         break;
282 :
283 :     case 22:
284 :         /* クロスラインを読み飛ばす */
285 :         if( IEncoderTotal-IEncoderLine >= 109 ) { /* 約10cmたったか? */
286 :             pattern = 23;
287 :             cnt1 = 0;
288 :         }
289 :         break;

```

中略

```

372 :     case 51:
373 :         /* 右ハーフライン検出時の処理 */
374 :         IEncoderLine = IEncoderTotal;
375 :         led_out( 0x2 );
376 :         handle( 0 );
377 :         motor( 0, 0 );
378 :         pattern = 52;
379 :         cnt1 = 0;
380 :         break;
381 :
382 :     case 52:
383 :         /* 右ハーフラインを読み飛ばす */
384 :         if( IEncoderTotal-IEncoderLine >= 109 ) { /* 約10cmたったか? */
385 :             pattern = 53;
386 :             cnt1 = 0;
387 :         }
388 :         break;

```

中略

```

440 :     case 61:
441 :         /* 左ハーフライン検出時の処理 */
442 :         IEncoderLine = IEncoderTotal;
443 :         led_out( 0x1 );
444 :         handle( 0 );
445 :         motor( 0, 0 );
446 :         pattern = 62;
447 :         cnt1 = 0;
448 :         break;
449 :
450 :     case 62:
451 :         /* 左ハーフラインを読み飛ばす */
452 :         if( IEncoderTotal-IEncoderLine >= 109 ) { /* 約10cmたったか? */
453 :             pattern = 63;
454 :             cnt1 = 0;
455 :         }
456 :         break;

```

以下、略

5.3 プログラムの解説

5.3.1 変数の追加

```

64 : /* エンコーダ関連 */
65 : int          iTimer10;          /* エンコーダ取得間隔      */
66 : long         lEncoderTotal;     /* 積算値                  */
67 : int          iEncoder;          /* 現在値                  */
68 : unsigned int uEncoderBuff;     /* 前回値保存              */
69 : long         lEncoderLine;     /* ライン検出時の積算値    */

```

69 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情報(積算値)を記憶させておきます。

5.3.2 積算値のクリア

```

130 :     case 1:
131 :         /* スタートバーが開いたかチェック */
132 :         if( !startbar_get() ) {
133 :             /* スタート!! */
134 :                 lEncoderTotal = 0;
135 :             led_out( 0x0 );
136 :             pattern = 11;
137 :             cnt1 = 0;
138 :             break;
139 :         }
140 :         if( cnt1 < 50 ) {          /* LED点滅処理          */
141 :             led_out( 0x1 );
142 :         } else if( cnt1 < 100 ) {
143 :             led_out( 0x2 );
144 :         } else {
145 :             cnt1 = 0;
146 :         }
147 :         break;

```

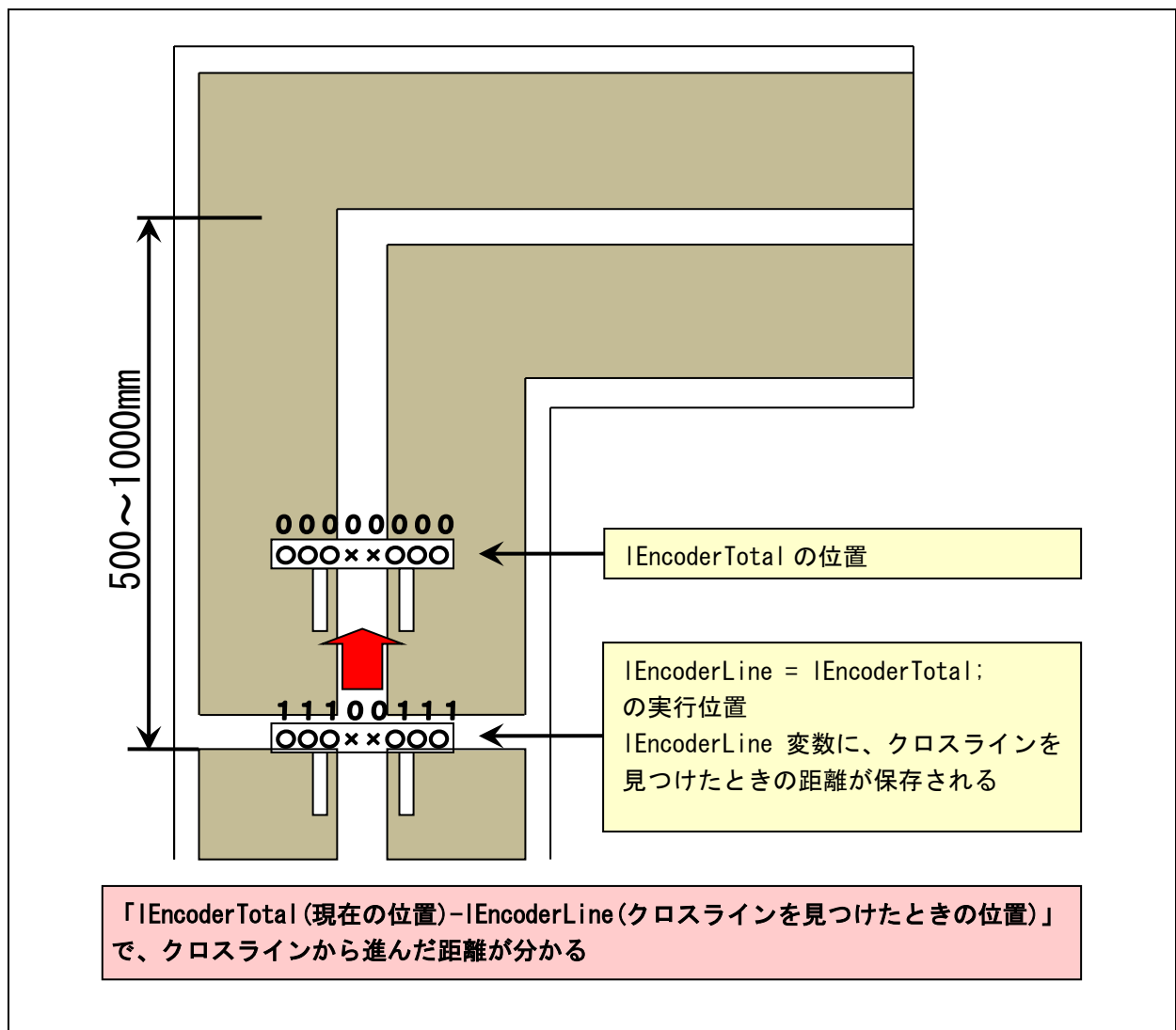
134 行の lEncoderTotal 変数は、電源を入れてから積算を開始します。そのため、スタート前もカウントしていません。lEncoderTotal 変数は、コースを走行した距離を測るのが目的ですので、走行前からカウントされると距離が変わってしまいます。そこで、スタート直前に lEncoderTotal 変数をクリアします。

5.3.3 パターン 21 クロスライン検出時の積算値を取得

```

273 :     case 21:
274 :         /* クロスライン検出時の処理 */
275 :         IEncoderLine = IEncoderTotal;
276 :         led_out( 0x3 );
277 :         handle( 0 );
278 :         motor( 0 , 0 );
279 :         pattern = 22;
280 :         cnt1 = 0;
281 :         break;
    
```

275 行は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(=進んだ距離)が分かります。



5.3.4 パターン 22 クロスラインを読み飛ばす

```
283 : case 22:  
284 :     /* クロスラインを読み飛ばす */  
285 :     if ( lEncoderTotal - lEncoderLine >= 109 ) { /* 約10cmたったか? */  
286 :         pattern = 23;  
287 :         cnt1 = 0;  
288 :     }  
289 :     break;
```

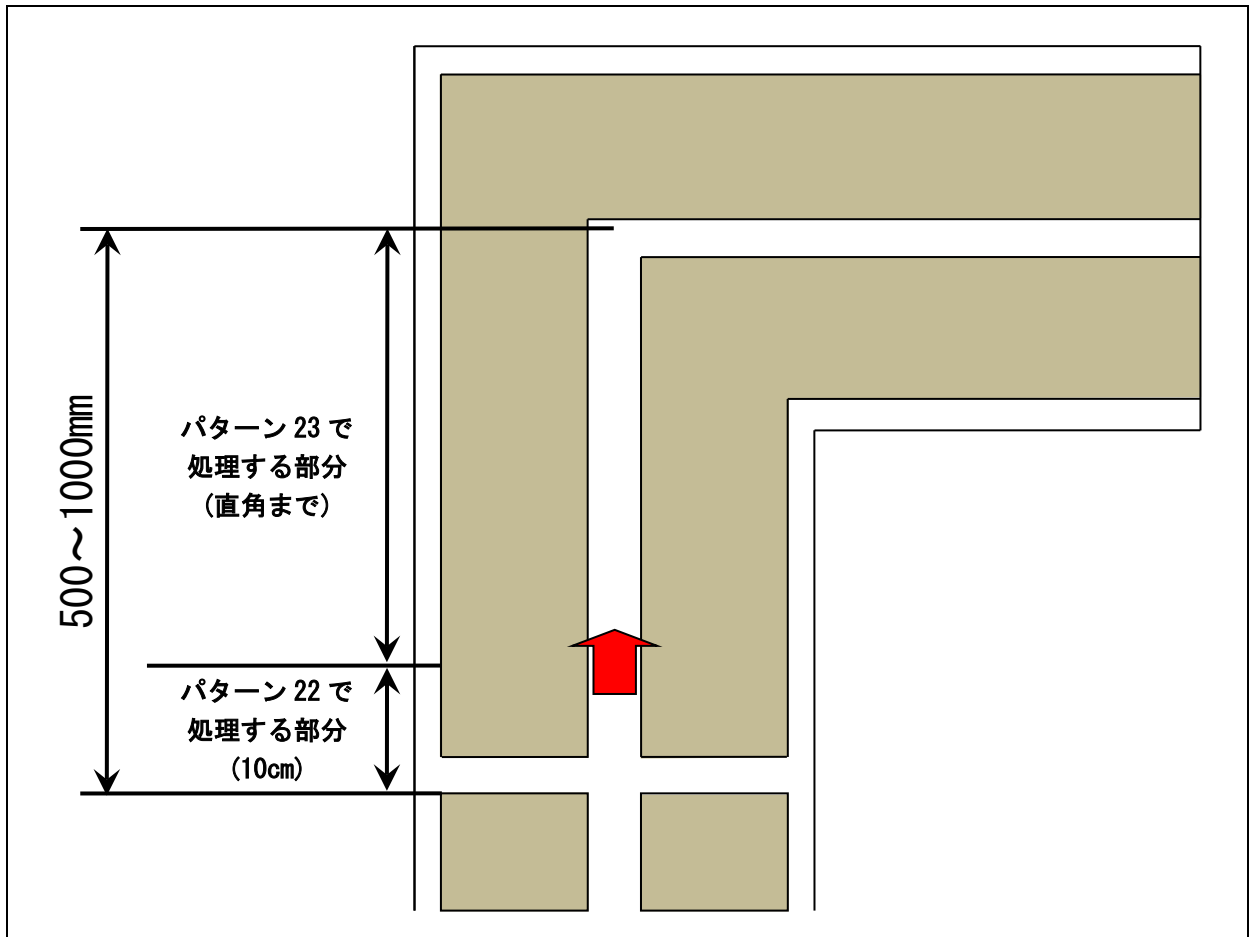
パターン22は、クロスラインを読み飛ばす処理です。クロスラインの幅は最大で4cmなので、クロスラインを見つけた瞬間から4cm進めばいいのですが、余裕を見て10cmとしています。285行目の計算式の意味を下記に示します。

$$\begin{array}{l} lEncoderTotal - lEncoderLine \qquad \qquad \qquad \geq 10cm \\ \downarrow \\ \text{現在の積算値} - \text{クロスラインを検出したときの積算値} \geq 10cm \\ \downarrow \\ \text{クランク内で進んだパルス数(距離)} \qquad \qquad \qquad \geq 10cm \end{array}$$

今回のロータリエンコーダは1mで1092パルスなので、10cm進んだかどうかチェックするには、

$$\begin{array}{l} 1m : 1092 \text{ パルス} = 0.1m : x \text{ パルス} \\ x = 109.2 \text{ パルス} \end{array}$$

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。



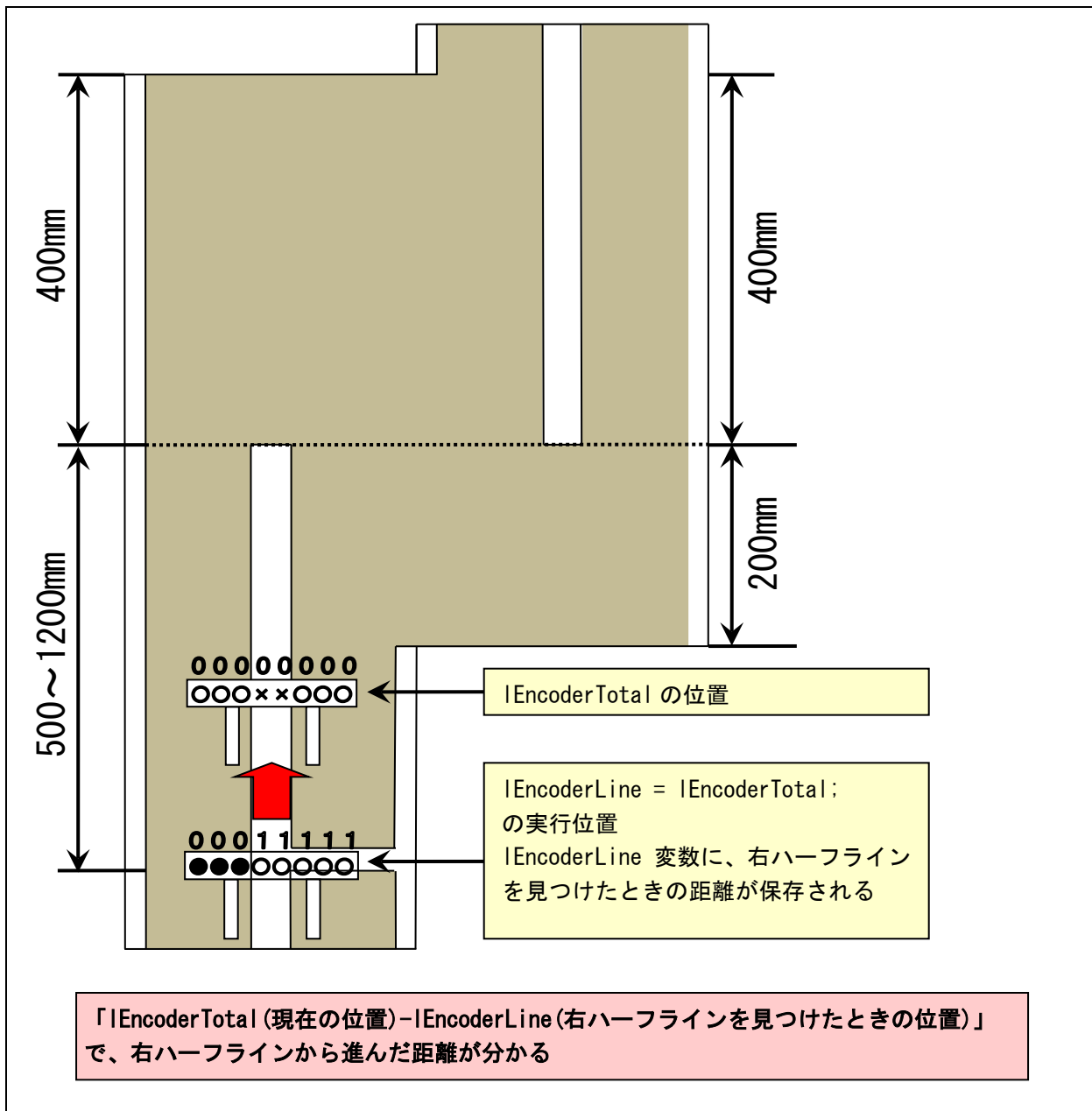
5.3.5 パターン 51 右ハーフライン検出時の積算値を取得

```

372 :     case 51:
373 :         /* 右ハーフライン検出時の処理 */
374 :         IEncoderLine = IEncoderTotal;
375 :         led_out( 0x2 );
376 :         handle( 0 );
377 :         motor( 0 , 0 );
378 :         pattern = 52;
379 :         cnt1 = 0;
380 :         break;

```

374 行は右ハーフ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、右ハーフラインを検出してからのパルス数(=**進んだ距離**)が分かります。



5.3.6 パターン 52 右ハーフラインを読み飛ばす

```
382 :     case 52:  
383 :         /* 右ハーフラインを読み飛ばす */  
384 :         if( lEncoderTotal-lEncoderLine >= 109 ) { /* 約10cmたったか? */  
385 :             pattern = 53;  
386 :             cnt1 = 0;  
387 :         }  
388 :         break;
```

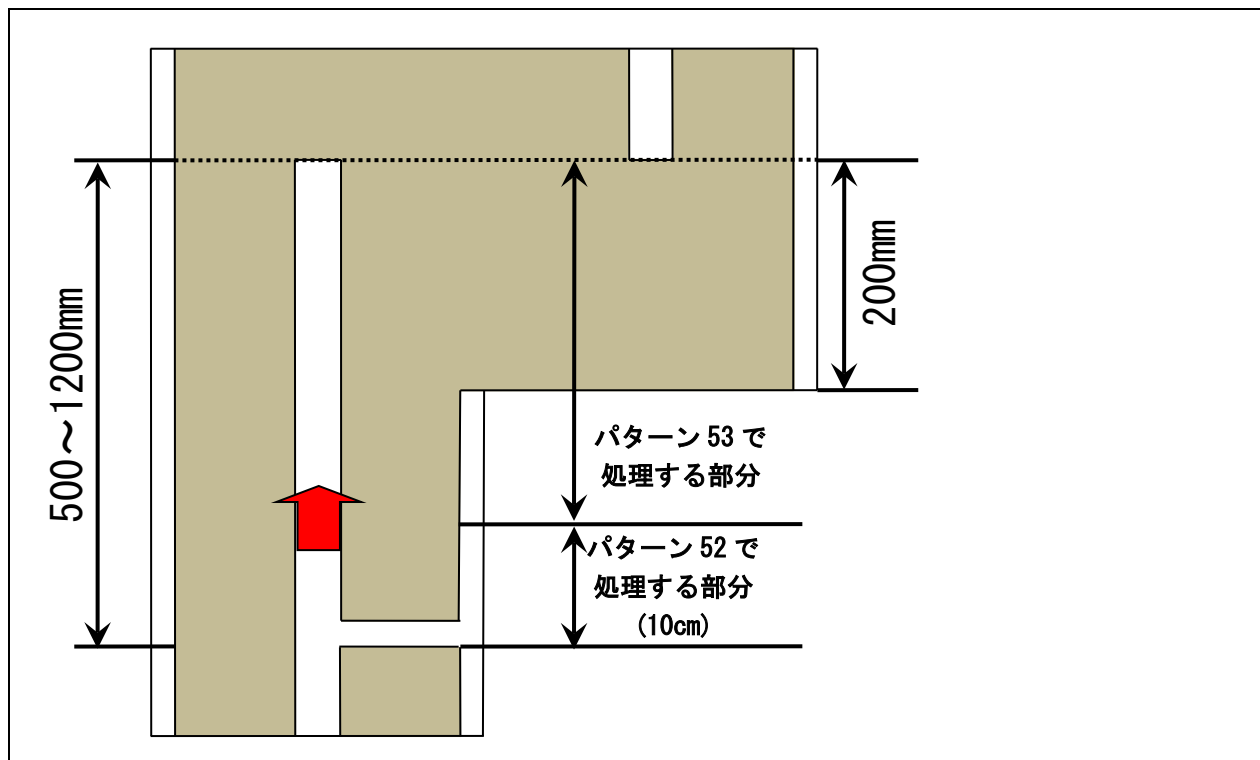
パターン 52 は、右ハーフラインを読み飛ばす処理です。右ハーフラインの幅は最大で 4cm なので、右ハーフラインを見つけた瞬間から 4cm 進めばいいのですが、余裕を見て 10cm としています。384 行目の計算式の意味を下記に示します。

$$\begin{aligned} & lEncoderTotal - lEncoderLine >= 10\text{cm} \\ & \quad \downarrow \\ & \text{現在の積算値} - \text{右ハーフラインを検出したときの積算値} >= 10\text{cm} \\ & \quad \downarrow \\ & \text{右ハーフライン検出後に進んだパルス数(距離)} >= 10\text{cm} \end{aligned}$$

今回のロータリエンコーダは 1m で 1092 パルスなので、10cm 進んだかどうかチェックするには、

$$\begin{aligned} 1\text{m} : 1092 \text{ パルス} &= 0.1\text{m} : x \text{ パルス} \\ x &= 109.2 \text{ パルス} \end{aligned}$$

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。



5.3.7 パターン 61~62 左ハーフライン部分の処理

```
440 :     case 61:
441 :         /* 左ハーフライン検出時の処理 */
442 :         IEncoderLine = IEncoderTotal;
443 :         led_out( 0x1 );
444 :         handle( 0 );
445 :         motor( 0 , 0 );
446 :         pattern = 62;
447 :         cnt1 = 0;
448 :         break;
449 :
450 :     case 62:
451 :         /* 左ハーフラインを読み飛ばす */
452 :         if( IEncoderTotal-IEncoderLine >= 109 ) { /* 約10cmたったか? */
453 :             pattern = 63;
454 :             cnt1 = 0;
455 :         }
456 :         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。442 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。452 行で 10cm 進んだと判断したとき、パターン 63 へ移ります。

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、タイヤ直径 21mm のロータリエンコーダを使用した場合です。条件が違ふとき、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
26	3750	それぞれのマイコンカーのサーボセンタ値にします。
239	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。
263	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。
285	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。
296	-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。
305	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。
311	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。
384	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。
399	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。
452	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。
467	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。
577	0x15	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、「 0x05 」にします。

6. 参考文献

- ルネサス エレクトロニクス(株)
R8C/38C グループ ユーザーズマニュアル ハードウェア編 Rev.1.10
- ルネサス エレクトロニクス(株)
M16C シリーズ,R8C ファミリー用Cコンパイラパッケージ V.5.45Cコンパイラユーザーズマニュアル Rev.1.00
- ルネサス エレクトロニクス(株)
High-performance Embedded Workshop V.4.05 ユーザーズマニュアル Rev.1.00
- ルネサス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
- 電波新聞社 マイコン入門講座 大須賀威彦著 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、販売部品についての詳しい情報は、マイコンカーラリー販売サイトをご覧ください。

<https://www2.himdx.net/mcr/>

R8C マイコンについての詳しい情報は、ルネサス エレクトロニクス(株)のホームページをご覧ください。

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

の製品情報にある「マイコン」→「R8C」でご覧頂けます