

マイコンカーキットVer.5.1

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。

本マニュアルでは『R8C/38A』のマイコン名称が残っていますが、すべて『R8C/38A および R8C/38C』の意味になります。

本プログラムは、マイコンカーがコースを走らせるための基本的なプログラムになっています。大会で使用するには、それぞれのマイコンカーに合わせてスピード、サーボの調整が必要です。さらに、スピードが変わったり、ちょっとしたぶれにより想定していないセンサ状態になり、脱輪することがあります。それらを解析、調整しながら大会に臨むようにしてください。

第 1.11 版
2014.06.02
ジャパンマイコンカーラリー実行委員会

注 意 事 項 (rev.4.0J)

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、最新の内容を確認いただきますようお願いします。
- ・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)ルネサスソリューションズ ルネサス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
E-mail:mcr.official@lm.renesas.com

目 次

1. マイコンカーラリー	1
1.1 マイコンカーラリー大会の部門	1
1.2 マイコンカーの規定	1
1.3 マイコンカーコース、スタートバーの仕様	4
1.3.1 コースの材質	4
1.3.2 基本的なコース	4
1.3.3 上り坂、下り坂	5
1.3.4 横線からクランク部分	5
1.3.5 レーンチェンジ部分	6
1.3.6 スタートバーの規格	7
1.3.7 コースレイアウト	8
2. マイコンカーキット Ver.5.1 のハードウェア	9
2.1 マイコンカーキットの変遷	9
2.2 マイコンカーキットのバージョン	11
2.3 マイコンカーキット Ver.5.1 の外観	12
2.4 標準キットの電源構成	14
2.5 駆動系電圧を上げた電源構成	15
3. センサ基板 Ver.5	16
3.1 仕様	16
3.2 回路図	17
3.3 寸法	18
3.4 センサの固定位置	18
3.5 外観	19
3.6 センサ基板 Ver.5 の CN1 と RY_R8C38 ボードとの関係	20
3.7 コースの白と黒を判断する仕組み	21
3.8 スタートバーの開閉を判断する仕組み	21
3.9 U8 と U9 の出力信号	22
3.10 センサ回路の原理	23
3.11 センサの調整方法	24
4. モータドライブ基板 Ver.5	27
4.1 仕様	27
4.2 回路図	28
4.3 寸法	29
4.4 外観	30
4.5 モータドライブ基板 Ver.5 の CN2 と RY_R8C38 ボードとの関係	32
4.6 モータ制御	33
4.6.1 モータドライブ基板の役割	33
4.6.2 スピード制御の原理	33
4.6.3 正転、逆転の原理	34
4.6.4 ブレーキとフリー	35
4.6.5 H ブリッジ回路	36
4.6.6 H ブリッジ回路のスイッチを FET にする	36
4.6.7 P チャネル FET と N チャネル FET の短絡防止回路	39

4.6.8 フリー回路.....	42
4.6.9 実際の回路.....	43
4.6.10 左モータの動作.....	45
4.6.11 右モータの動作.....	45
4.7 サーボ制御.....	46
4.7.1 原理.....	46
4.7.2 回路.....	47
4.8 LED 制御.....	47
4.9 スイッチ制御	48
5. サンプルプログラム.....	49
5.1 プログラムの開発環境	49
5.2 サンプルプログラムのインストール	49
5.2.1 ホームページからダウンロードする	49
5.2.2 インストール	50
5.3 ワーススペース「kit12_38a」を開く.....	51
5.4 プロジェクト	52
6. プログラム解説「kit12_38a.c」.....	53
6.1 プログラムリスト	53
6.2 「kit07_38a.c」と「kit12_38a.c」プログラムの相違点.....	62
6.3 R8C/38A マイコンで使用する内蔵周辺機能.....	62
6.4 プログラムの解説.....	63
6.4.1 スタート.....	63
6.4.2 外部ファイルの取り込み(インクルード)	63
6.4.3 シンボル定義.....	64
6.4.4 プロトタイプ宣言	66
6.4.5 グローバル変数の宣言.....	67
6.4.6 init 関数(クロックの切り替え)	68
6.4.7 init 関数(ポートの入出力設定)	68
6.4.8 init 関数(端子のプルアップ)	71
6.4.9 init 関数(タイマ RB の設定)	71
6.4.10 init 関数(タイマ RD の設定)	72
6.4.11 intTRB 関数(1ms ごとの割り込み)	73
6.4.12 timer 関数(時間稼ぎ)	74
6.4.13 sensor_inp 関数(センサ状態の読み込み)	75
6.4.14 check_crossline 関数(クロスラインチェック)	81
6.4.15 check_rightling 関数(右ハーフライン検出処理)	82
6.4.16 check_leftline 関数(左ハーフライン検出処理)	83
6.4.17 dipsw_get 関数(ディップスイッチ値読み込み)	84
6.4.18 pushsw_get 関数(プッシュスイッチ値読み込み)	85
6.4.19 startbar_get 関数(スタートバー検出センサ読み込み)	86
6.4.20 led_out 関数(LED 制御)	87
6.4.21 motor 関数(モータ速度制御)	88
6.4.22 handle 関数(サーボハンドル操作)	94
6.4.23 スタート	96
6.4.24 パターン方式について	97
6.4.25 プログラムの作り方	97
6.4.26 パターンの内容	99
6.4.27 パターン方式の最初 while、switch 部分	100

6.4.28 パターン 0:スイッチ入力待ち.....	101
6.4.29 パターン 1:スタートバーが開いたかチェック.....	103
6.4.30 パターン 11:通常トレース	105
6.4.31 パターン 12:右へ大曲げの終わりのチェック.....	114
6.4.32 パターン 13:左へ大曲げの終わりのチェック.....	118
6.4.33 クランク概要	123
6.4.34 パターン 21:クロスライン検出時の処理	124
6.4.35 パターン 23:クロスライン後のトレース、クランク検出.....	126
6.4.36 パターン 31、32:左クランククリア処理.....	129
6.4.37 パターン 41、42:右クランククリア処理.....	133
6.4.38 右レーンチェンジ概要	137
6.4.39 パターン 51:右ハーフライン検出時の処理	138
6.4.40 パターン 53:右ハーフライン後のトレース	140
6.4.41 パターン 54:右レーンチェンジ終了のチェック	142
6.4.42 左レーンチェンジ概要	144
6.4.43 パターン 61:左ハーフライン検出時の処理	145
6.4.44 パターン 63:左ハーフライン後のトレース	147
6.4.45 パターン 64:左レーンチェンジ終了のチェック	149
7. サーボセンタと最大切れ角の調整.....	151
7.1 概要	151
7.2 通信ソフト「Tera Term」をインストールする	152
7.3 サーボのセンタを調整する	157
7.4 サーボの最大切れ角を見つける	164
7.5 「kit12_38a.c」プログラムを書き換える	168
8. プログラムの改造ポイント.....	170
8.1 概要	170
8.2 脱輪事例.....	171
8.2.1 クロスラインの検出がうまくいかない	171
8.2.2 クランクの検出がうまくいかない.....	172
8.2.3 ハーフラインの検出がうまくいかない.....	174
8.2.4 クランククリア時、外側の白線を中心と勘違いして脱輪してしまう	175
8.2.5 レーンチェンジ終了の判断ができない	178
8.3 まとめ	179
9. モータの左右回転差の計算方法.....	180
9.1 計算方法.....	180
9.2 エクセルを使った内輪の自動計算.....	181
9.3 内輪を自動で計算する関数を追加する.....	182
9.3.1 プロトタイプの追加.....	182
9.3.2 大域変数の追加	182
9.3.3 handle 関数内の追加	183
9.3.4 配列の追加	183
9.3.5 diff 関数の追加	186
9.3.6 プログラムでの使い方	187
10. フリー動作を追加する	188
10.1 概要	188
10.2 接続	188

10.3 プログラムの追加	189
10.3.1 .関数の追加	189
10.3.2 .使い方	190
10.3.3 .ブレーキとフリーの違いを確認する	191
10.3.4 .走行プログラムに motor_mode 関数を組み込む	192
11. RMC-R8C35A ボードを使う	193
11.1 RMC-R8C35A ボード	193
11.2 RMC-R8C35A ボードと RY_R8C38 ボード	194
11.3 RMC-R8C35A ボードの搭載	195
11.4 RMC-R8C35A ボードの電源スイッチをコネクタに変更する	196
11.5 サンプルプログラムのインストール	197
11.5.1 ホームページからソフトを取得する	197
11.5.2 インストール	198
11.6 ワーススペース「kit07r8c」を開く	199
11.7 プロジェクト	200
12. 参考文献	201
13. 改訂記録	202

1. マイコンカーラリー

1.1 マイコンカーラリー大会の部門

ジャパンマイコンカーラリー大会は、高等学校在籍者、または特別支援学校の高等部に在籍している生徒(以下、高校生)が参加対象です。2014 年現在、競技は下記の 2 部門が行われます。

●Advanced Class

高校生全員が参加できます。ただし、Basic Class との重複登録はできません。

●Basic Class

マイコンカーラリー初心者を対象とした部門で、初めてマイコンカーラリーの大会に参加する高校生を対象としています(大会に参加した年度のみ参加可能です)。1年生はもとより、2年生、3年生であっても初めてマイコンカーラリーに取り組む生徒は参加できます。Advanced Class と比べ、使える部品が限定されています。

もちろん、初めて参加するからといって Basic Class に参加しなければいけない訳ではありません。Advanced Class への参加も可能です。

※一般の部について

一般の部は、2009 年度にジャパンマイコンカーラリーから分離しました。詳しくは、マイコンカーラリーホームページ(<http://www.mcr.gr.jp/>)をご覧ください。

1.2 マイコンカーの規定

2014 年度の Advanced Class、Basic Class の規定を、下表に示します。詳しくは競技規則を参照してください。ゴシック体赤色部分は、2014 年度に変更、追加になった部分です。

部門	Basic Class	Advanced Class
外形など 共通仕様	<ul style="list-style-type: none"> マシンの外形は幅 300mm、高さ 150mm 以内とし、全長、重量、材質等については制限しない。 タイマセンサを遮ることのできる構造とする。 スタート後、タイムを有利にするため故意に全長を変えることは不可。 マシンのタイヤ(同等の機能を有するものを含む)はコース面上に接触しながら走行するものとし、接触部分に粘着性物質を使用することは不可(車検に於いて、コースに貼り付くと確認されるものも含む)。 タイヤ幅 30 mm未満、4 輪以内とする。 ※タイヤ幅とは、マシンの進行方向に対する横方向の寸法である。 ※タイヤ幅の確認は、幅 30mm のコの字型のタイヤゲージで行います。一部でも触れると車検はとおりません。 吸引機能を用いたマシンは不可。 電気二重層コンデンサの使用は不可。 ※バックアップ電源等の用途で販売されている電気二重層コンデンサ等の大容量キャパシタは、使用不可とする(公称容量がF[ファラド]で標記されているものは不可)。 走行時にコースを損傷させたり汚したりするおそれのある構造は不可。 レギュレーション検査合格後の改造は禁止とする。ただし、モード変更用機器(液晶など)の脱着は認めるが、レギュレーション車検時と同じ状態にすること。 	

1. マイコンカーラリー

部門	Basic Class	Advanced Class
マイコン ボード	マイコンボードは右に同じ。 他に、マイコンボードの改造はコネクタの追加などの基本性能を変えない加工のみ認め る。	実行委員会承認のマイコンボードを使うこと。 ※マシンに搭載した状態で型式が確認できるこ と。
駆動 モータ	モータの型式、改造の有無は右に同じ。 他に、モータの個数は、2 個とする。	実行委員会承認のモータを使うこと、使用個数は制限なし。分解、改造は不可。ただし、ノイズ除去コンデンサ等のケースへの半田付けは除く。モータの個数は、4 個以内とする。 ※マシンに搭載し た状態ですべてのモータの MCR 刻印が確認でき ること。
ギヤ ボックス	実行委員会承認のギヤボックスを 2 個使うこと。ケースの改造は次の3つ以外認めない。 ①ピニオンギア(8T)への交換 ②シャーシ取り付けネジを避けるための逃 げ加工 ③シャフトの切断	規定なし(自由)
センサ	センサには、実行委員会承認の基板を 1 枚使用すること。代替え部品への交換は認め るが、改造は認めない。ただし、コース検出 センサ(発光部、受光部)の型式の変更は認め ない。	規定なし(自由)
モータ ドライバ	モータドライバには、実行委員会承認の基 板を 1 枚使用すること。代替え部品への交 換は認めるが、改造は認めない。	規定なし(自由)
追加 基板	マイコンボード、センサ基板、モータドライブ 基板以外の基板(部品単体を含む)は、実行 委員会承認のものを使用すること。代替え 部品への交換、マニュアルに掲載している 追加は認めるが、その他の改造は認めな い。	規定なし(自由)
電池	使用できる電池は右に同じ。本数は、制御 系に 4 本、駆動系に 4 本使うこと。取り付け は、電池ボックスを使用し電圧値の確認が でき、電池を容易に取り外すことができる構 造であること。電池のパック化は認めない。 ※制御系の上限電圧は 5.5V までのため、 実際は単三 2 次電池を 4 本加えること になります。	単三アルカリ電池、または単三 2 次電池(1.2V 仕 様)8 本以内。アルカリと 2 次電池の混在は可能。 タブ付き電池は使用可能。マンガン電池、ニッケ ル一次電池、オキシライド電池など上記の 2 種類 以外は使用不可。 ※マシンに搭載した状態で、すべての電池の単三 アルカリ電池記号(「LR6(JIS)」、「AM3(国内旧 称)」、「単三形(国内通称)」、「AA(米国内通 称)」)または単三 2 次電池記号(「AA(米国内通 称)」)が確認できること。 ※電池極面の半田付けはルール上禁止され ていませんが、メーカーの注意事項として禁 止されていることがほとんどなので、電池メー ターの使用上の注意に従ってください。
変圧	三端子レギュレータなどによる降圧、 DC-DC コンバータによる昇圧を含め、変圧 禁止 ※実装されていると、使っているか使っ ていないか判断できません。変圧部品 を実装しないようにしてください。	規定なし(自由)

部門	Basic Class	Advanced Class
サーボ	実行委員会承認のサーボを1個使うこと。サーボモータの基本性能を変える加工は認めない。 ※マシンに搭載した状態で型式が確認できること。	規定なし(自由)
速度を測る機器	使用禁止	規定なし(自由)
表示する機器	規定なし(自由)	規定なし(自由)
走行状態を記録	規定なし(自由) ※ロータリエンコーダはセンサですので、本規定と関係なく、使用できません	規定なし(自由)

※承認部品、指定部品について

内容	型式	BClass	AClass
マイコンボード	①H8/RY3048F ②RY3048Fone(TypeH 含) ③MS304CP01 ④MS304CP02 ⑤RY3687 ⑥RY3687N ⑦RY_R8C38 ⑧RMC-R8C35A	1枚 使用すること	使用すること (数量制限なし)
ギヤボックス	ハイスピードギヤーボックス HE	2個 使用すること	— (規定適用外)
モータ	RC-260RA-18130 (MCR 刻印付)	2個 使用すること	使用すること ただし4個まで
サーボモータ	①ハイテック製 HS-430BH (6.0V 時 0.16s/60 度 4.1kg・cm) ②フタバ製 S3003 (4.8V 時 0.23s/60 度 3.2kg・cm) ③サンワ製 SRM-102Z (4.8V 時 0.20s/60 度 3.0kg・cm) ④JR(日本遠隔制御株式会社)製 S-519 (4.8V 時 0.23s/60 度 3.3kg・cm) ⑤ハイテック製 HS-425BB (4.8V 時 0.21s/60 度 3.3kg・cm) ※変更があった場合は変更した年から3年間は使用できる こととする。⑤の HS-425BB は、平成 26 年度まで使用可 能とする。	1個 使用すること	— (規定適用外)
センサ	①ミニマイコンカーVer.2 のセンサ部(スタートバー検出セ ンサ基板と合わせて1枚とする) ②センサ基板 TLN113 版(スタートバー検出センサ基板と合 わせて1枚とする) ③センサ基板 TLN119 版(スタートバー検出センサ基板と合 わせて1枚とする) ④センサ基板 Ver.4(Ver.4.1 も含む) ⑤センサ基板 Ver.5	1枚 使用すること	— (規定適用外)
モータ ドライバ	①ミニマイコンカーVer.2 のモータドライブ部 ②モータドライブ基板 Vol.2(拡張基板を含む) ③モータドライブ基板 Vol.3 ④モータドライブ基板 Ver.4 ⑤モータドライブ基板 Ver.5	1枚 使用すること	— (規定適用外)
マイコンボード 、センサ基板、 モータドライブ基 板以外の基板	①RY_R8C38 ボード コネクター変換基板 ②フリー追加セット ③EEP-ROM 基板 ④液晶・microSD 基板 ⑤液晶・microSD 基板 Ver.2 ⑥RY_R8C38 ボード DIP スイッチ基板	使用すること (数の制限は なし)	— (規定適用外)

1.3 マイコンカーコース、スタートバーの仕様

1.3.1 コースの材質

コースの黒色、灰色、白色は、下記の材質のシールを使用しています。シールを使っていない部分はつや消し白のアクリル材となります。

● 黒色

セキスイハルカラーHC-015、エコパレットハルカラーHKC-011、中川ケミカル 793(ブラックマット)のいずれか

● 灰色

セキスイハルカラーHC-050、エコパレットハルカラーHKC-057、中川ケミカル 735(ミディアムグレー)のいずれか

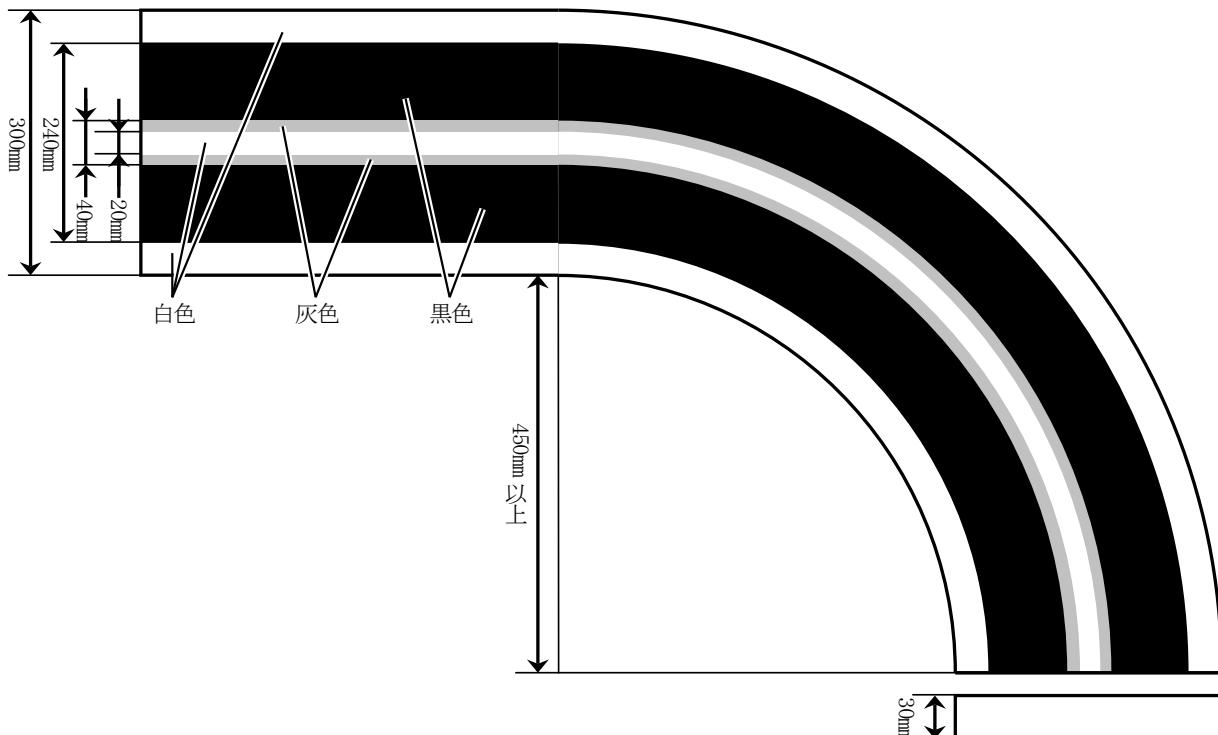
● 白色

セキスイハルカラーHC-095、エコパレットハルカラーHKC-097、中川ケミカル 711(ホワイト)のいずれか

※2014年4月現在、販売されているコースは、中川ケミカルのシールを使っています(予告無く変更の可能性があります)。

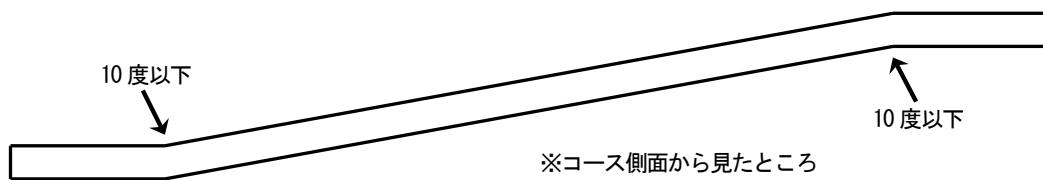
1.3.2 基本的なコース

マイコンカーラリーのコースは、直線、カーブ、クランク、坂、レーンチェンジで構成されています。マイコンカーラリーのコースは、幅 300mm の中に、黒、灰、白色があります。カーブは内径が 450mm 以上となっています。マイコンカーに取り付けているセンサでコースとマイコンカーのズれを検出し、コースに沿って走るように制御します。



1.3.3 上り坂、下り坂

角度が 10 度以下の上り坂、下り坂があります。上り初め、上り終わり、下り初め、下り終わりでマイコンカーのシャーシなどがコースとこすらないように製作する必要があります。



※車検は、上り下りコースパート部(10度以内の傾斜がついた坂道コースの一部)を使用して、マイコンカーを手動で通過させます。このとき、センサ類(タイヤ、アースを含む)以外はコースに接触してはいけません。

2輪タイプでコース接触部にコース保護材をつけたものはタイヤの一部と見なします。

車検時に、センサ部においてコースを損傷させる可能性が確認された場合は、保護材などで対処をお願いします(エンコーダーやリミットスイッチ、センサ含む)。

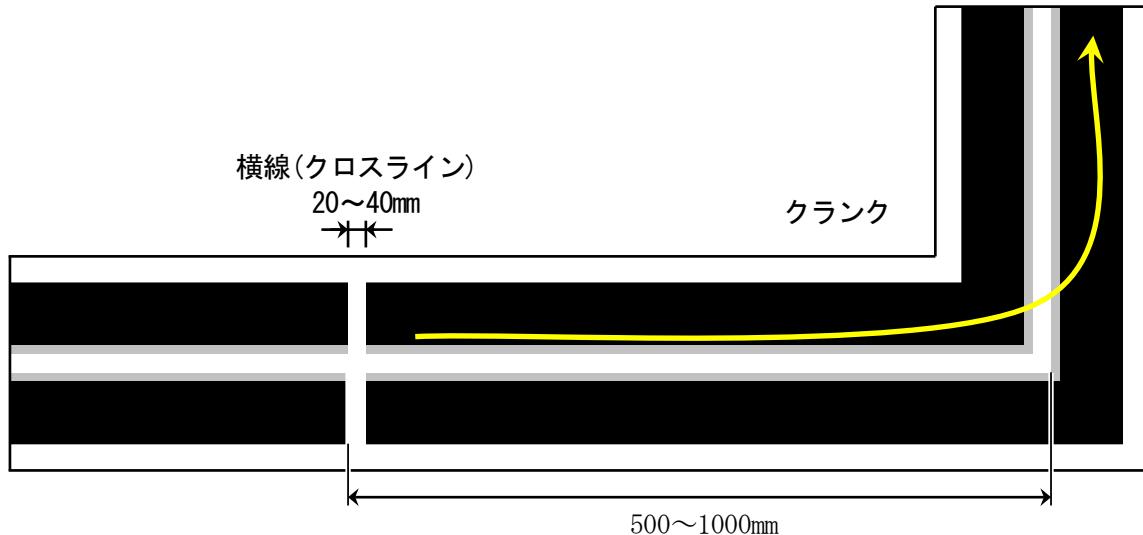


上り坂、下り坂を組み合わせた立体交差

1.3.4 横線からクランク部分

マイコンカーラリーコースのいちばんの特徴は、クランク(直角)です。クランクは最大の難所ですが、腕の見せ所でもあります。

クランク手前の 500~1000mm には幅 20~40mm の白線(1本)が引かれています。マイコンカーはこのラインを検出すると、直角を曲がれるスピードまで減速します。直角を発見すると曲がり、通常走行に戻ります。



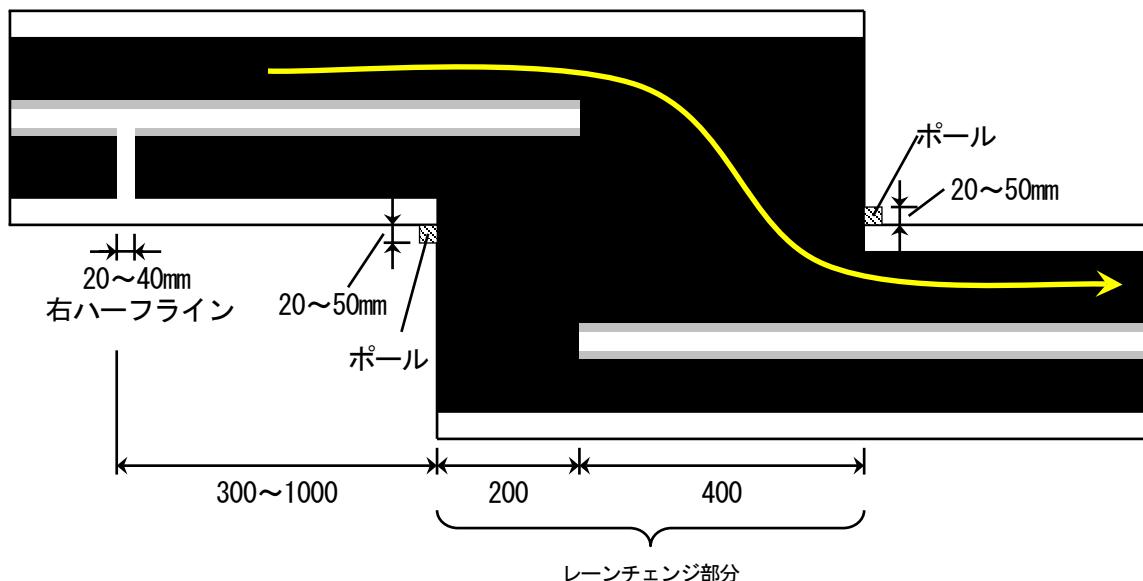
1. マイコンカーラリー

1.3.5 レーンチェンジ部分

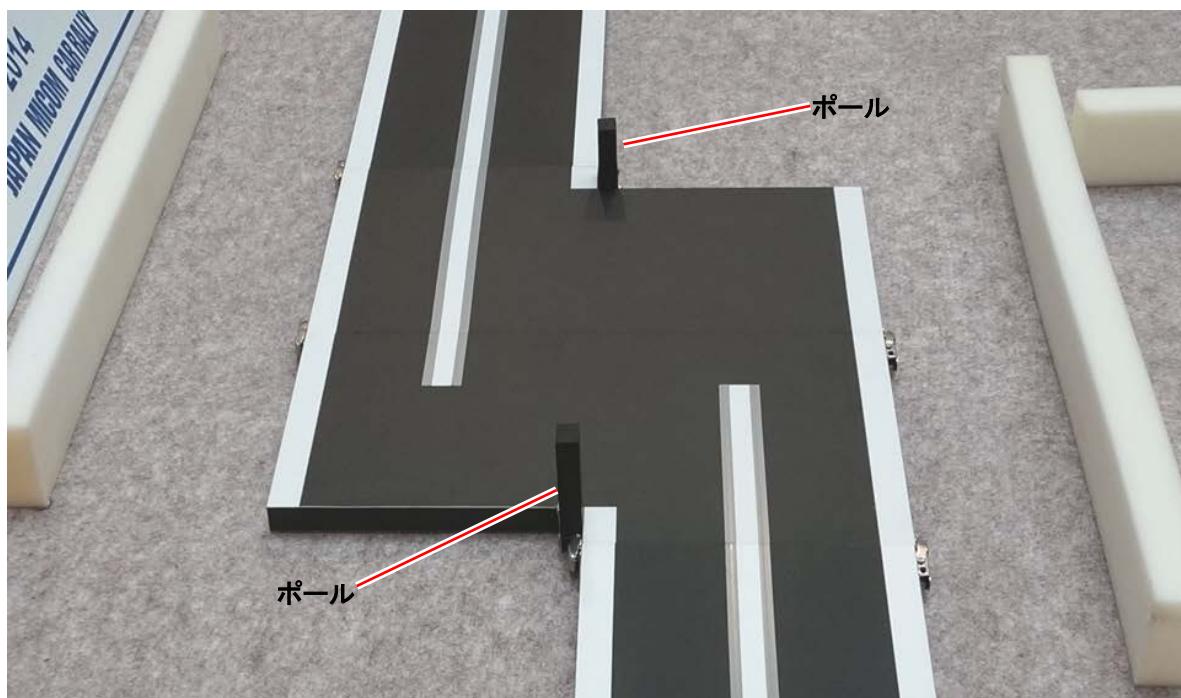
レーンチェンジは、チェンジ区間長さ 600mm、幅 600mm の部分があり、チェンジ区間より手前 300~1000mm の地点に、幅 20mm~40mm の白線をチェンジ方向に合わせ(左右片側に)1 本引いている部分です。

例えば、マイコンカーは右ハーフラインを検出すると、そこから 300mm~1000mm 後に右レーンチェンジがあると判断し、スピードを落として進ませます。中心線が無くなると右にハンドルを切りレーンチェンジを開始し、新しい中心線を見つけるとレーンチェンジ完了と判断し、通常走行に戻ります。

右レーンチェンジのコースを、下記に示します。



※2013 年度よりガードレールが、ポールという名称に変更になりました。



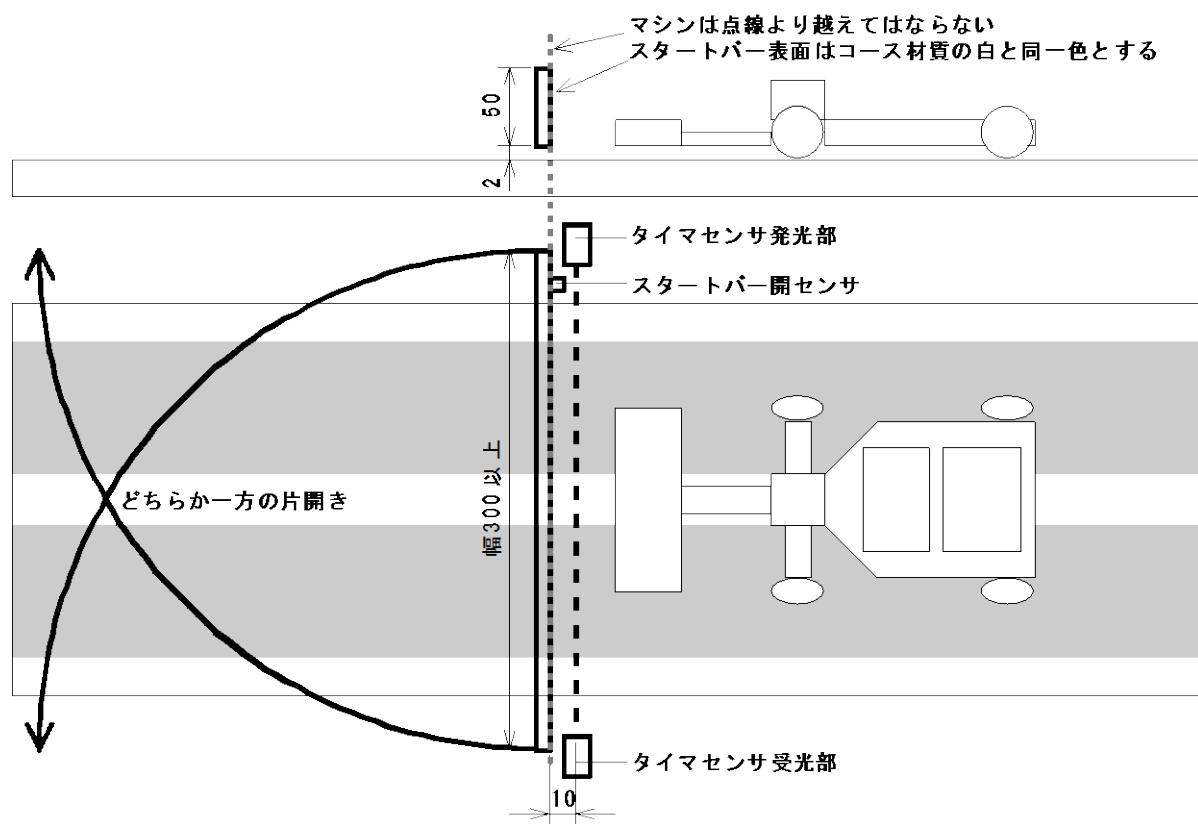
▲ポールの設置例

1.3.6 スタートバーの規格

マイコンカーのスタートは、スタートバーと呼ばれるゲートが開くと同時に計測を開始する方式です。スタート手順を下記に示します。

1. 選手は、マイコンカーをスタートバーに触れないように、かつスタートバーを越えないようにセットします。
2. 審判が、マイコンカーをセットできたか確認します。選手は、準備ができたら審判にセットできた旨を伝えます（一般的には手を挙げて合図しますが、各大会で異なります。各大会のルールを確認してください）。セット後は、マイコンカーに触れることはできません。
3. スタートバー（表面はコース材質の白色のシールが貼ってあります）が進行方向に開きます（押し扉のイメージ）。
4. マイコンカーは、スタートバーが開いたことを自動で検出してスタートします（センサ基板 Ver.5 には、スタートバー検出用のセンサが付属しています）。
5. スタートバーが開くと同時に、タイム計測が開始されます。
6. スタートバーが開いた後マイコンカーがスタートしない場合は、手動スイッチによるスタートも認められています。ただし、タイマーセンサーを通過したマイコンカーに触れた場合は失格となります。※

※2013 年度まではマイコンカーを持ち上げて確認することを認めていましたが、2014 年度からは持ち上げ禁止（持ち上げると記録なし）になりました。



1. マイコンカーラリー

1.3.7 コースレイアウト

コースレイアウトは毎年変わり、直前まで非公開です。地区大会では全長約 50m、全国大会では約 60～65m にもなります。どの様なコースレイアウトにも対応するマイコンカーを作る、そこが腕の見せ所です。



JMCR2014 全国大会 予選コースレイアウト

2. マイコンカーキット Ver.5.1 のハードウェア

2.1 マイコンカーキットの変遷

年度とマイコンカーキットの車体の変遷、プログラムの変遷、主なルール変更について、下表に示します。

	車体の変遷	プログラムの変遷	主なルール変更
1998 年頃	マイコンカーキット Vol.1 を開発しました。	プログラム名「tmc4.c」 マイコンカーキット Vol.1 に対応したプログラムです。	
2002 年度	マイコンカーキット Vol.2 を開発しました。モータドライブ基板 Vol.2 を開発し、電池 8 本分の電圧を加えられる回路になりました。センサ基板 TLN113 版を開発し、前回より小型化しました。	プログラム名「kit2.c」 マイコンカーキット Vol.2 に対応したプログラムです。	駆動モータが、ジャパンマイコンカーラリー指定モータのみの使用となりました。 (高校生の部のみ)
2004 年度		プログラム名「kit04.c」 パターン方式を使用した制御方式に変更しました。	クロスラインからクランクまでの距離が 1m 固定から、50cm～1m 可変となりました。
2005 年度	モータドライブ基板 Vol.3 を開発し、逆転できるようになりました(今までできませんでした)。それに合わせてマイコンカーキットを「マイコンカーキット Vol.3」としました。	プログラム名「kit05.c」 モータドライブ基板 Vol.3 に対応したプログラムです。	
2006 年度	スタートバー検出センサがオプションとして開発されました。	プログラム名「kit06.c」 自動スタート方式、レンチエンジコースに対応したプログラムです。	スタート時、スタートバーが開くことをマイコンカーが自動検出してスタートする方式となりました。また、レンチエンジコースが導入されました。
2007 年度	マイコンカーキット Ver.4 を開発しました。センサ基板 Ver.4 を開発しました。モータドライブ基板は Vol.3 のままでです。	プログラム名「kit07.c」 センサ基板 Ver.4 に対応したプログラムです。	Basic Class(高校生の初心者部門)がプレ大会として導入されました。
2008 年度			Basic Class が正式部門となり、 ・高校生 Advanced Class の部 ・高校生 Basic Class の部 ・一般の部 の 3 部門となりました。
2009 年度	10 度の坂道に対応した、「マイコンカーキット Ver.4.1」を開発しました。このキットは、マイコンカーキット Ver.4 のシャーシ下のネジを標準で皿ネジとして(Ver.4 セットは鍋ネジでした)、10 度車検でシャーシ底部分がぶつからないようにしました。		一般の部が独立し、 ・Advanced Class の部 ・Basic Class の部 の 2 部門となりました。

2. マイコンカーキット Ver.5.1 のハードウェア

	車体の変遷	プログラムの変遷	主なルール変更
2010 年度	R8C/38A マイコンボード (RY_R8C38)、R8C/35A マイコンボード (RMC-R8C35A)が承認されました。	R8C/38A マイコンでマイコンカーを走行させる「kit07_38a.c」、R8C/35A マイコンでマイコンカーを走行させる「kit07_35a.c」を公開しました。	部門名が、 ・Advanced Class ・Basic Class となりました。
2011 年度	モータドライブ基板 Ver.4 を開発しました。センサ基板 Ver.4 にポリパイルテープを追加してセンサ基板 Ver.4.1 としました。それに合わせて、マイコンカーキットを「マイコンカーキット Ver.5」としました。		下記のように改訂されました。 ・タイヤ幅 30mm 以内 ・AClass の駆動モータは 4 個以内とする ・BClass のセンサ、モータドライバーは承認基板とする。
2012 年度	モータドライブ基板 Ver.5、センサ基板 Ver.5 を開発しました。それに合わせて、マイコンカーキットを「マイコンカーキット Ver.5.1」としました。	センサ基板の改訂に合わせて、「kit12_38a.c」を公開しました。 ※モータドライブ基板 Ver.5 に変更になった事によるプログラム変更はありません。	Basic Class で、表示する機器の搭載、走行状態の記録が、禁止から規定なしになりました。
2013 年度	RY_R8C38 マイコンボードのマイコンが、R8C/38A から R8C/38C に変更になりました。 RMC-R8C35A マイコンボードのマイコンが R8C/35A から R8C/35C に変更になりました。		・クランクとレーンチェンジ手前の幅 20mm の 2 本線が、幅 20 ~ 40mm の 1 本線になりました。 ・Basic Class で追加基板が指定になりました。 ・ガードレールがポールになりました。
2014 年度			・電池の型式について、「マシンに搭載した状態で、すべての電池の単三アルカリ電池記号、または単三 2 次電池記号)が確認できること」が追加になりました。

2.2 マイコンカーキットのバージョン

マイコンカーキットのバージョンを、下表に示します。

開発年	キットのバージョン	車体	センサ基板	モータドライブ基板	マイコンボード（マイコン）	プログラム
1998度頃	Vol.1	A	センサ基板 (バージョンなし)	モータドライブ基板 (バージョンなし)	RY3048F (H8/3048F)	tmc4.c
2002年度	Vol.2	B	センサ基板 TLN113 版	モータドライブ基板 Vol.2	RY3048Fone (H8/3048F-ONE)	kit2.c kit04.c
2005年度	Vol.3	B	センサ基板 TLN113 版 途中から センサ基板 TLN119 版	モータドライブ基板 Vol.3	RY3048Fone (H8/3048F-ONE)	kit05.c kit06.c
2007年度	Ver.4	C	センサ基板 Ver.4	モータドライブ基板 Vol.3	RY3048Fone (H8/3048F-ONE)	kit07.c
2009年度	Ver.4.1	C'	センサ基板 Ver.4	モータドライブ基板 Vol.3	RY3048Fone (H8/3048F-ONE)	kit07.c
2011年度	Ver.5	C''	センサ基板 Ver.4.1	モータドライブ基板 Ver.4	RY_R8C38 (R8C/38A)	kit07_38a.c
2012年度	Ver.5.1	C''	センサ基板 Ver.5	モータドライブ基板 Ver.5	RY_R8C38 (R8C/38A)	kit12_38a.c
2013年度	Ver.5.1	C''	センサ基板 Ver.5	モータドライブ基板 Ver.5	RY_R8C38 (R8C/38C)	kit12_38a.c

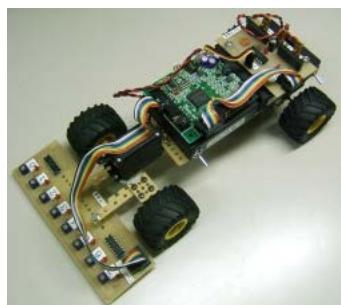
※C'…シャーシ下の鍋ネジを皿ネジに変更しました。

※C''…モータドライブ基板の固定用スタットの下側をメス、上側をオスにしました。

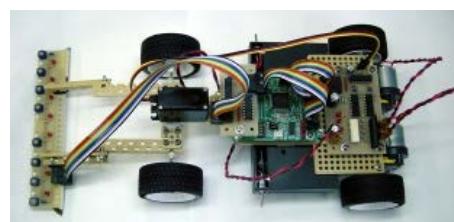
「Vol.」は「volume」を略した記述です。訳すと「巻」という意味で、主に本で使用します。

「Ver.」は「version」を略した記述です。訳すと「版」という意味で、改訂した回数を示します。

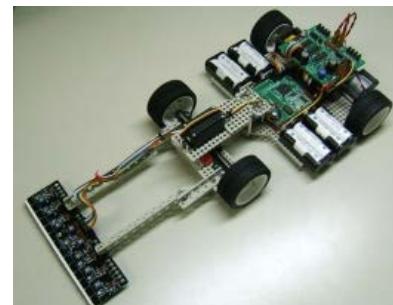
マイコンカーの改訂数は、3 回目まで「Vol.」を使っていましたが、前記のとおり誤った使い方です。4 回目から本来の意味である「Ver.」を使っています。「Vol.1～3」は誤った使い方ですが、今までのマニュアルの記述を考え、あえて直していません。



▲マイコンカーキット Vol.1
(車体 A バージョン)

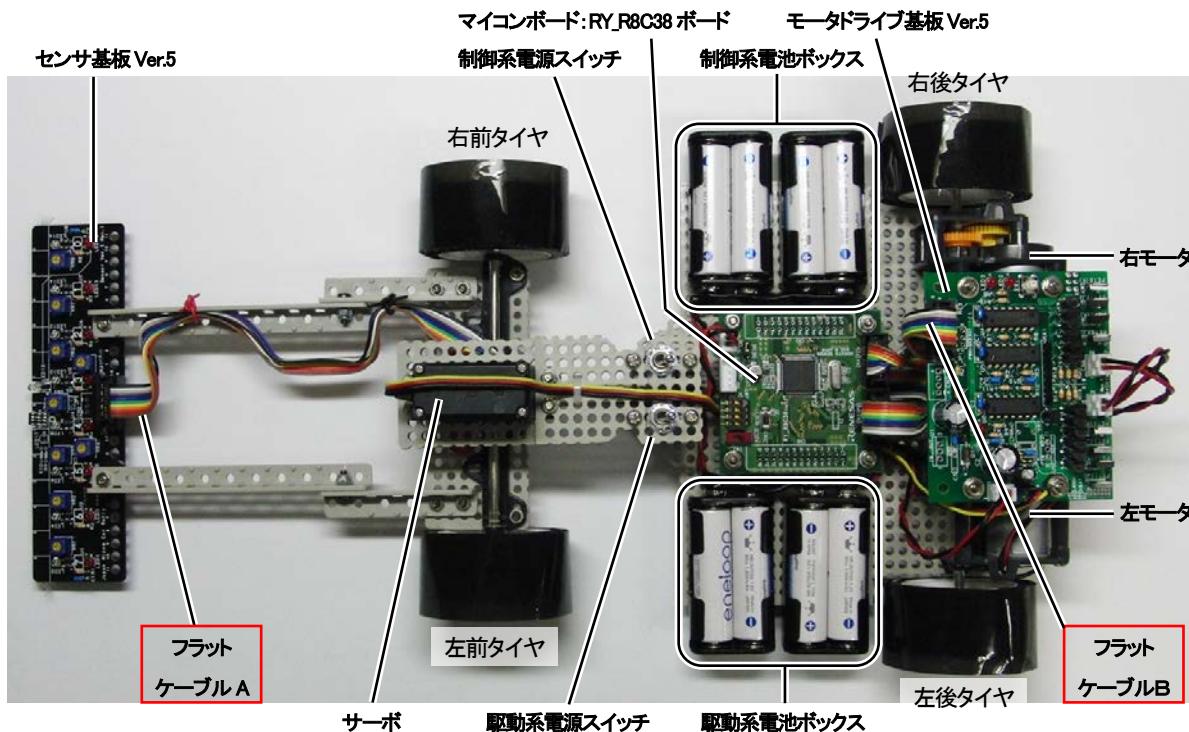


▲マイコンカーキット Vol.2、Vol.3
(車体 B バージョン)

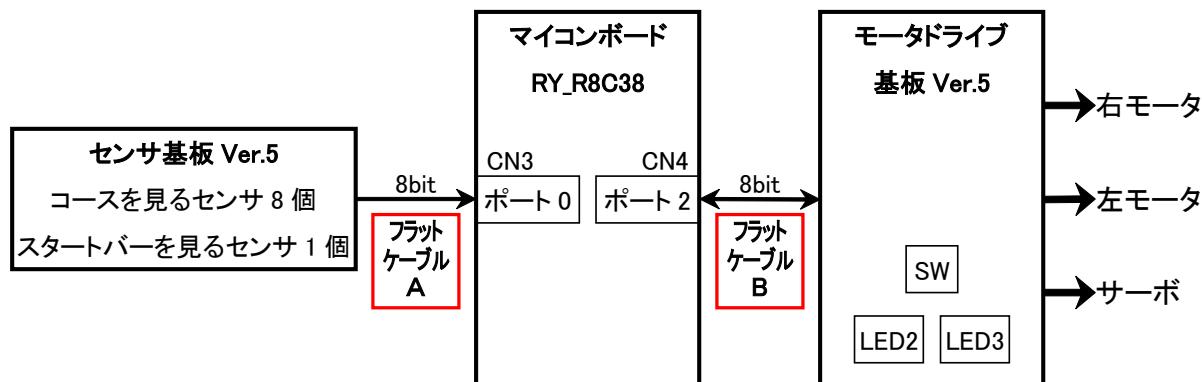


▲マイコンカーキット Ver.4、Ver.5
(車体 C バージョン)

2.3 マイコンカーキットVer.5.1 の外観



マイコンカーキット Ver.5.1 は制御系の RY_R8C38 ボード(R8C/38A マイコン搭載のマイコンボード)、センサ基板 Ver.5、モータドライブ基板 Ver.5、駆動系の右モータ、左モータ、サーボで構成されています。

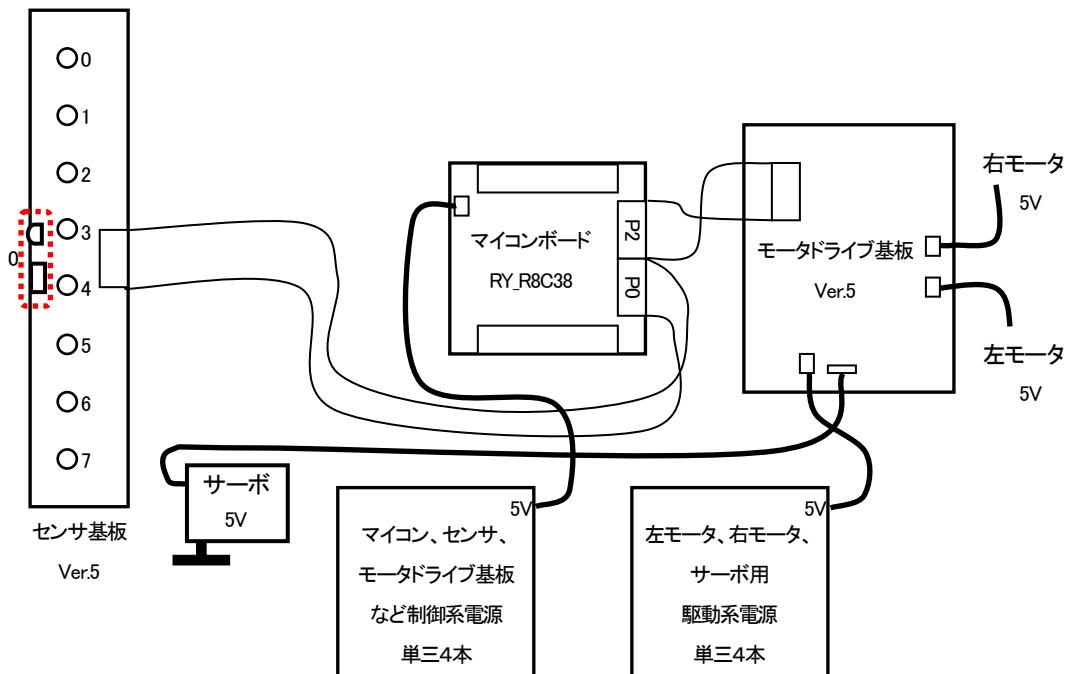


マイコンボード RY_R8C38	センサの状態をポート0から読み込み、左モータ、右モータのPWM出力値、サーボの切れ角を計算して、ポート2に接続されているモータドライバ基板へ出力します。 センサの状態を基に、どのようにモータ、サーボの出力値を決めるかをプログラムすることになります。
センサ基板 Ver.5	コースの状態を検出するセンサが8個あります。センサの下部が白色なら"0"を出力、黒色なら"1"を出力します。 ※プログラムは信号を反転させ、白色を"1"、黒色を"0"と判断しています。 スタートバーがあるかどうか検出するセンサが1個あります。スタートバーがあれば"0"を出力、無ければ"1"を出力します。 ※コース状態検出センサのいちばん右側とスタートバー検出センサは、bit0でOR接続になっています。スタート時、いちばん右側のセンサはコースが黒色なので反応がないので、このときに反応した場合はスタートバーの状態と判断します。スタート後は、コースの状態と判断します。
モータドライバ 基板 Ver.5	マイコンボードからの弱電信号(10mA程度)を、モータを動作させるための強電信号(数A程度)に変換しモータを制御します。サーボもモータ用電源を使用します。 プッシュスイッチが接続されており、このスイッチを押すことによりマイコンカーがスタートするようにプログラムされています。さらに、LEDが2個付いており、デバッグに使用できます。
電池	<ul style="list-style-type: none"> 制御系(マイコン)電源 … 単三2次電池4本($1.2V \times 4$本 = 4.8V)を使用 駆動系(モータ・サーボ)電源 … 単三2次電池4本か … 単三アルカリ電池4本($1.5V \times 4$本 = 6.0V)を使用 <p>※制御系の電圧は必ず、4.5~5.5Vの電圧にしてください。</p>

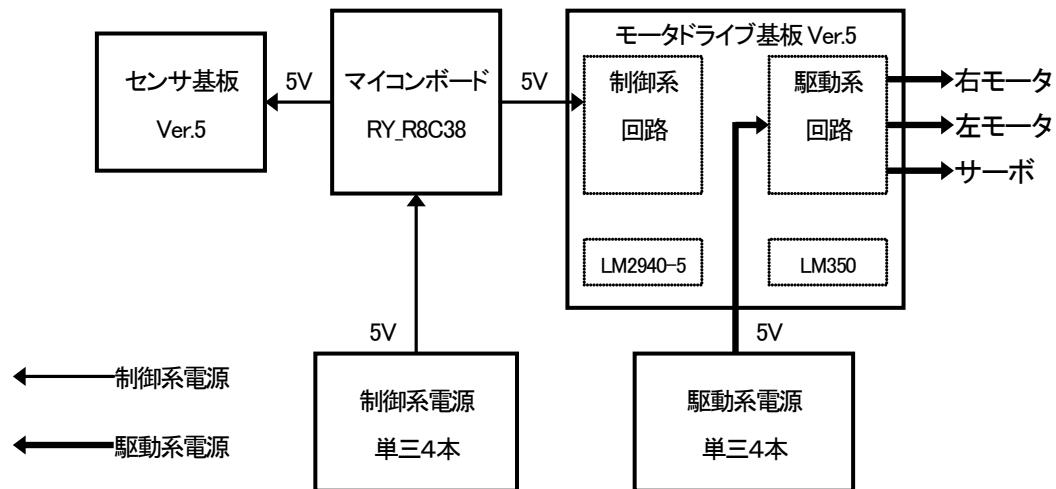
2.4 標準キットの電源構成

標準キットでは、制御系と駆動系で電源系統を切り離して、モータ・サーボ側でどれだけ電流を消費してもマイコンがリセットしないようにしています。

標準キットの電源構成を下記に示します。



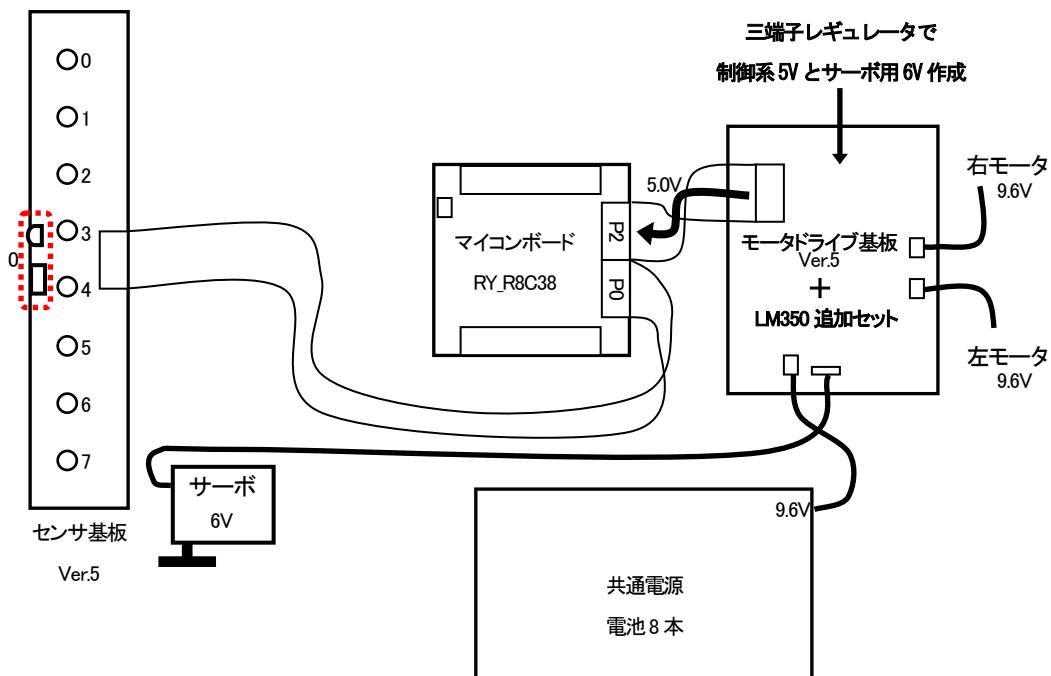
電源系の流れを下記に示します。



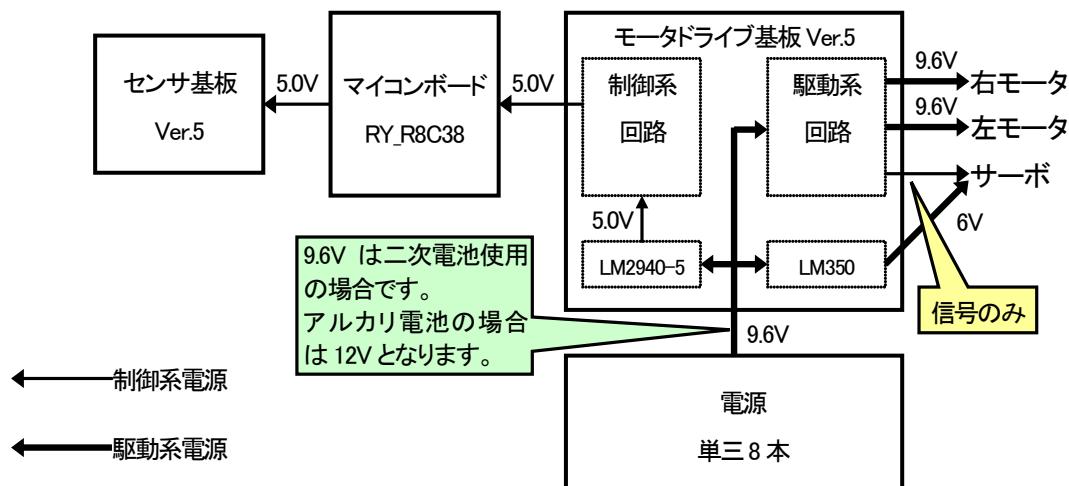
2.5 駆動系電圧を上げた電源構成

駆動系の電圧を上げれば(電池を増やせば)モータの回転数を上げることが可能ですが。モータ電源用に6本の電池を使えば7.2V、8本なら9.6Vとなります。しかし、電池の使用本数はルールで8本以内と決まっています。そこで、電池を制御系、駆動系共通にします。このとき、モータに9.6Vの電圧を加えても壊れませんが(定格は6Vなので定格オーバーですが)、マイコンの動作保証電圧は2.7~5.5V(20MHz動作時)なので5.5Vを超えた電圧をかけると動作しなくなるおそれがあります(電圧の絶対最大定格は6.5Vです、6.5V以上加えると壊れます)。サーボも同様に6V以上の電圧をかけられません。そのため、三端子レギュレータを取り付けマイコンやサーボの電圧を定格にします。ただし、電池を共通にした場合はモータなどが電流を大量に消費し、2.7V以下になるとマイコンがリセットしてしまいます。電池を共通化した場合、マイコンのリセットに気をつけなければいけません。

モータドライブ基板 Ver.5 に「LM350 追加セット」の部品を追加すると、6V以上の電圧を利用して LM2940-5 がマイコンなどの制御系で使用する電圧5Vを生成、LM350がサーボで使用する電圧6Vを生成します。



電源系の流れを下記に示します。



※キットの電池ボックスは、バネの押しが弱いため、マイコンカーの加減速によって、電池の端子が電池ボック
スから離れてしまい電源が切れて、マイコンがリセットすることがあります(数十 ms の単位です)。押しつけの
強い電池ボックスを使うなどして、このようなことが起こらないようにしてください。

3. センサ基板 Ver.5

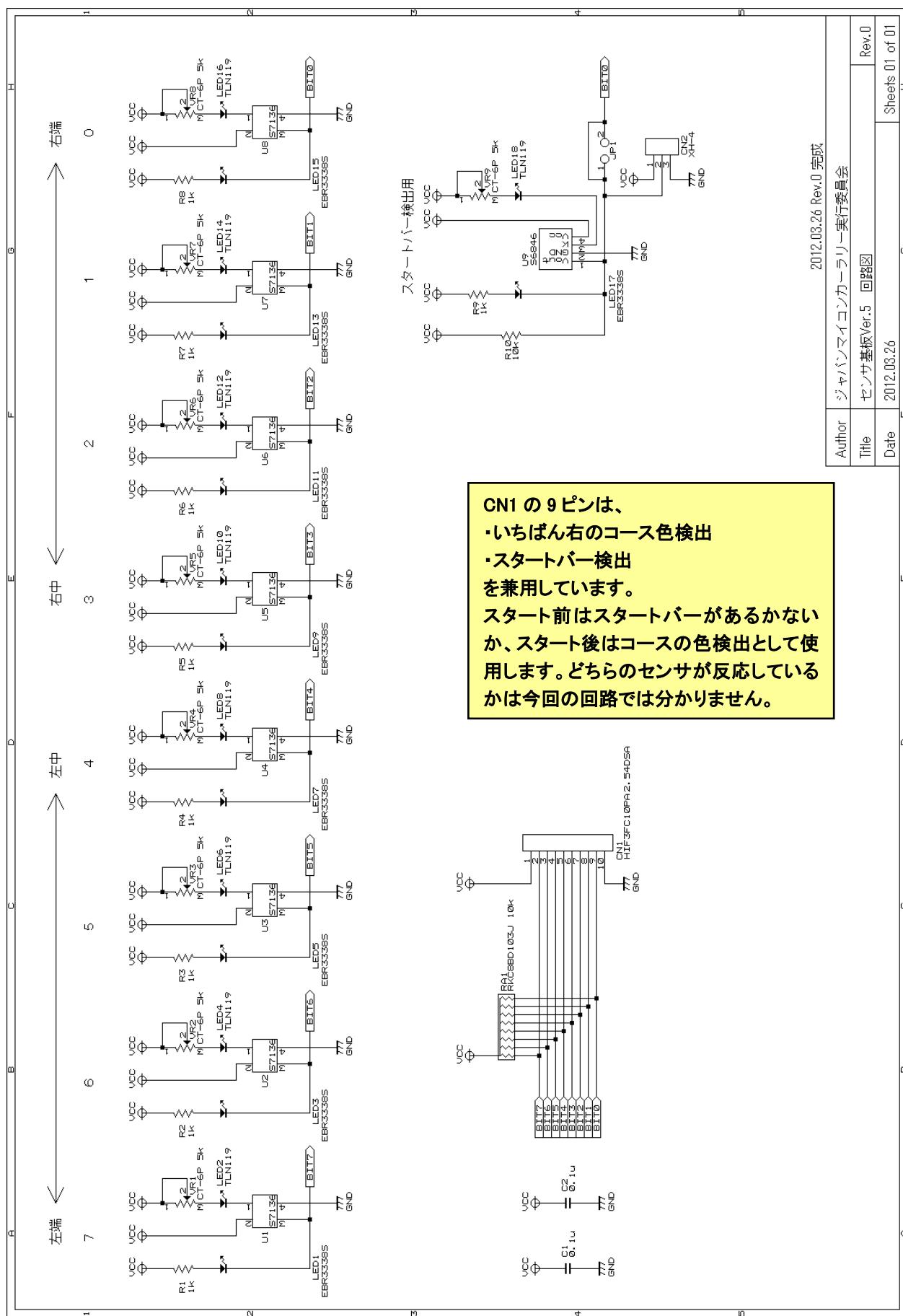
3.1 仕様

センサ基板 Ver.5 の仕様を、下表に示します。

名称	センサ基板 Ver.5	センサ基板 Ver.4.1(参考)
略称	センサ基板 5	センサ基板 4
付属キット	マイコンカーキット Ver.5.1	マイコンカーキット Ver.4 マイコンカーキット Ver.5
販売開始時期	2012 年 6 月～	2007 年 5 月～2012 年 6 月
基板枚数	1 枚	←
コースを見るセンサの個数	8 個	7 個
スタートバーを見るセンサの個数	1 個	←
信号反転回路	なし(プログラムで反転)	←
マイコンボードとの接続	R8C/38A:ポート 0 H8/3048F-ONE:ポート 7	←
電圧	DC5.0V±10%	←
重量(完成品の実測)	約 23g ※ポリパイルテープ含む	約 21g ※ポリパイルテープ含む
レジスト(基板色)	黒色	←
基板寸法	W140×D38×厚さ 1.2mm	←
寸法(実測)	最大 W140×D38×H14mm	←
標準プログラム	R8C/38A マイコン:kit12_38a.c H8/3048F-ONE マイコン:kit07.c	R8C/38A マイコン:kit07_38a.c H8/3048F-ONE マイコン:kit07.c

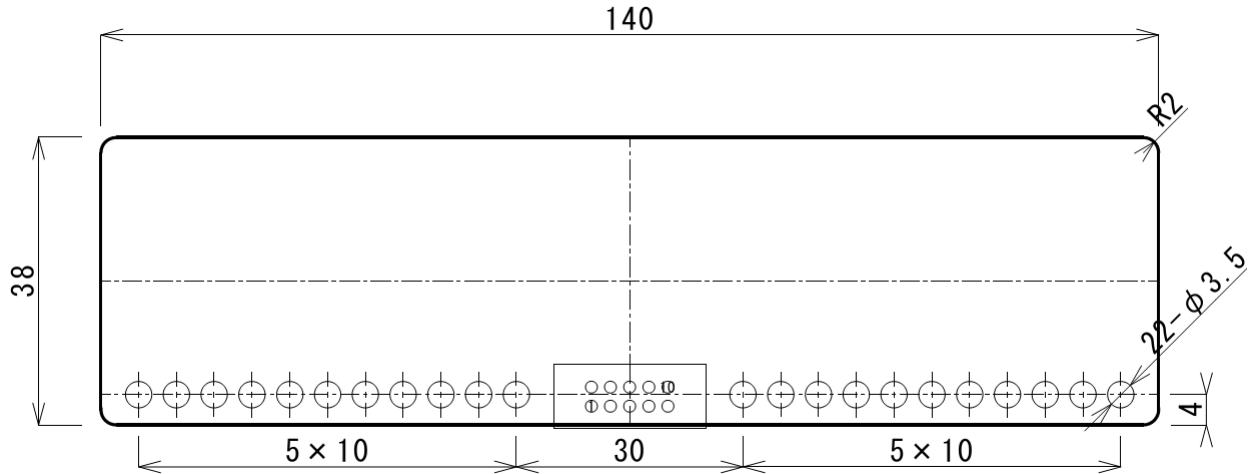
※重量は、リード線の長さや半田の量で変わります

3.2 回路図



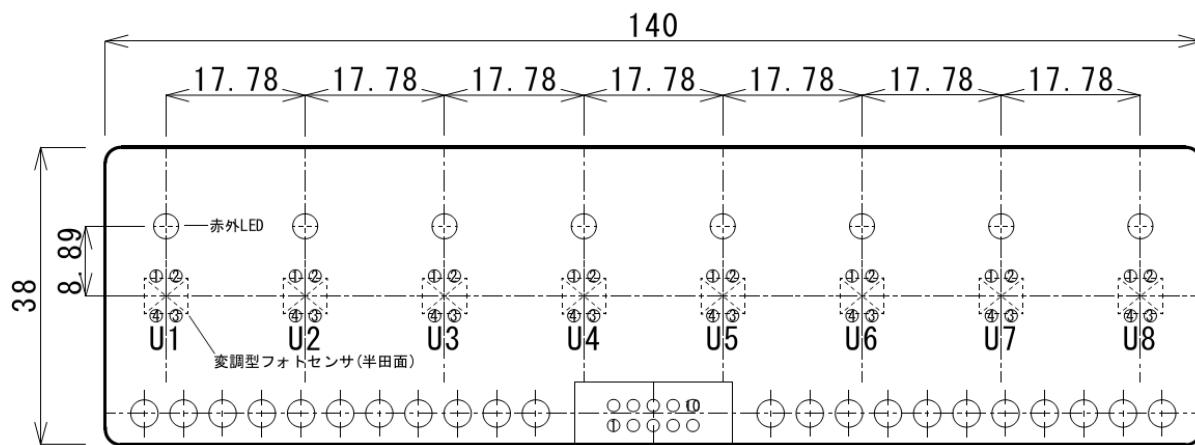
3.3 寸法

センサ基板の取り付け用の穴として、左右 11 個、合計 22 個の穴があります。この穴を使ってセンサ基板を固定してください。

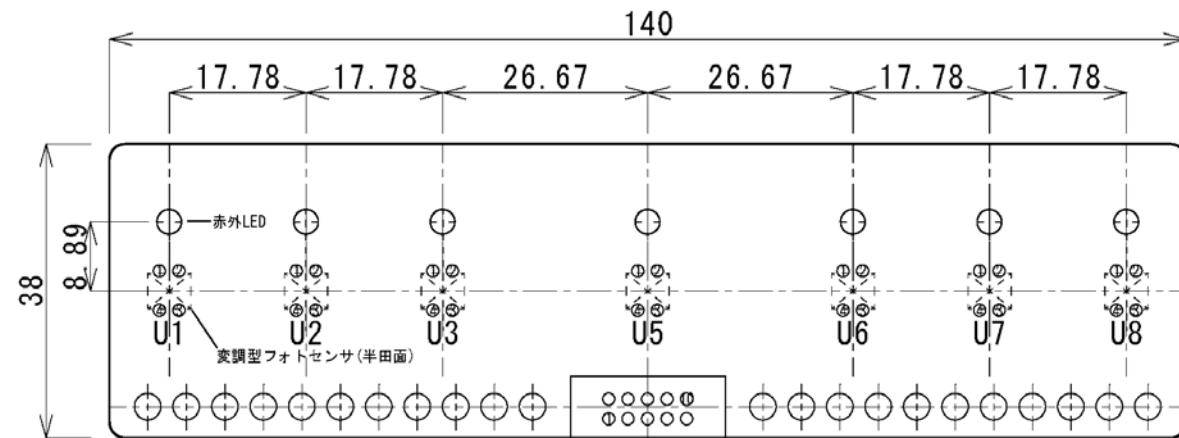


3.4 センサの固定位置

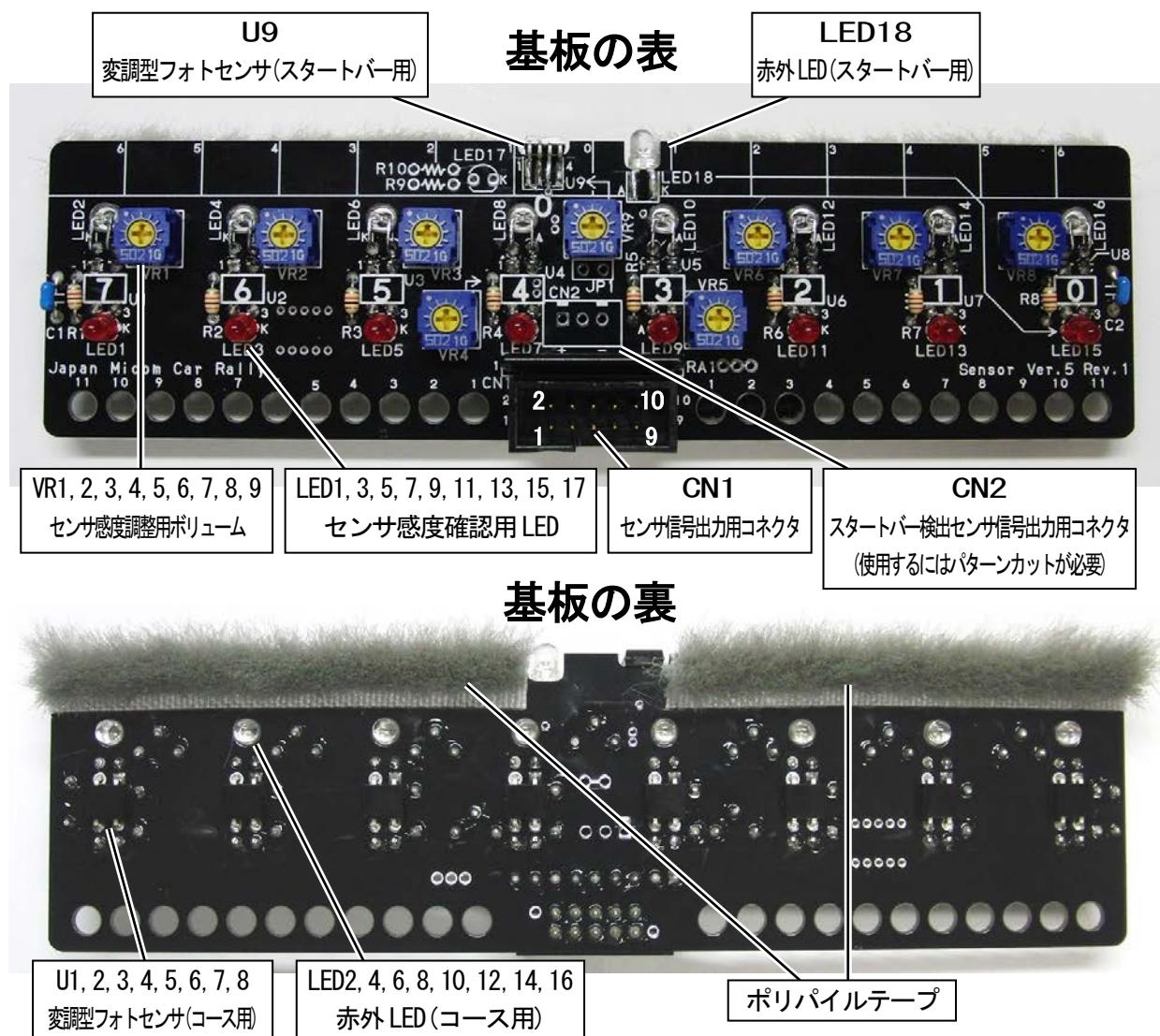
コースの白黒を検出するセンサは、8 個あります。センサを基板へ取り付けている位置を、下記に示します。



※参考—センサ基板 Ver.4.1 のセンサの位置



3.5 外観



コネクタの接続先、信号の内容を下表に示します。

部品番号	内容	詳細
CN1	コネクタ(マイコンボードと接続)	事項参照
CN2	コネクタ (オプション)	コース検出センサのいちばん右側とスタートバー検出センサの信号は兼用です。スタートバー検出センサの信号を CN2 に独立させて出力させることができます。 このとき、スタートバー検出センサ用を独立させるための部品 (R9、R10、LED17) を、追加で実装する必要があります。 詳しくはセンサ基板 Ver.5 製作マニュアルを参照してください。
LED2,4,6,8, 10,12,14,16	赤外 LED	TLN119を使用しています。この素子から赤外線の光を出します。赤外線なので人間の目には見えません。コース検出用に 8 個あります。
LED18	赤外 LED	TLN119を使用しています。スタートバー検出用に 1 個あります。
U1,2,3,4, 5,6,7,8	変調型 フォトセンサ	浜松フォトニクス(株)の S7136 という変調型フォトセンサを使用しています。赤外 LED が出した赤外線をこの素子で受けます。光が受信できればコースは白、できなければコースは黒と判断します。8 個あります。

3. センサ基板 Ver.5

U9	変調型 フォトセンサ	浜松フォトニクス(株)の S6846 という変調型フォトセンサ(縦型)を使用しています。赤外 LED が出した赤外線をこの素子で受けます。光が受信できればスタートバーあり、できなければスタートバーなしと判断します。1 個あります。
VR1,2,3,4, 5,6,7,8	センサ感度 調整用ボリューム (コース用)	これらのボリュームで、赤外 LED から出力する光の量を調整します。マイコンカーのコースには、灰色の線があります。ボリュームで感度を変えることにより、灰色を“白”と判断させるか、“黒”と判断させるか調整することができます。標準のプログラムでは、“白”と判断させると良いようになっています。
VR9	センサ感度 調整用ボリューム (スタートバー用)	このボリュームで、LED18 から出力する光の量を調整します。スタートバーがあると白色、無いなら反射無しとなります。スタートバーがあるときに反応するように(LED15 が点灯するように)このボリュームを調整してください。
LED1,3,5,7, 9,11,13,15	センサ感度 確認用 LED	この LED が点灯で“白”、消灯で“黒”と判断しています。ボリュームで感度を調整するときは、この LED を確認しながら調整します。
—	ポリパイルテープ	ポリパイルテープをセンサ基板の半田面に取り付け、コースとセンサが直接擦らないように、またセンサが適切に反応するように高さを一定にします。

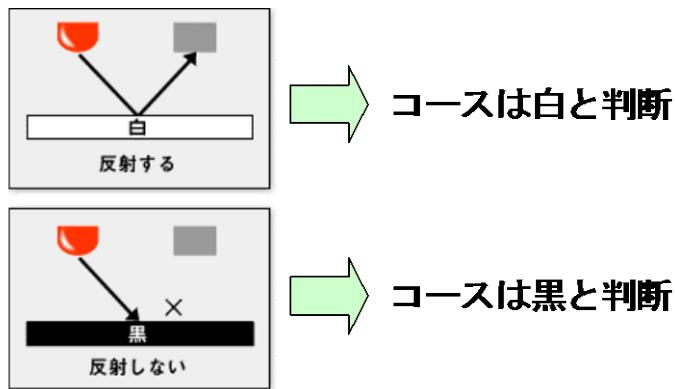
3.6 センサ基板 Ver.5 の CN1 と RY_R8C38 ボードとの関係

センサ基板 Ver.5 の CN1 と RY_R8C38 ボードの CN3(ポート 0)を 10 ピンフラットケーブルで接続します。信号の内容を、下表に示します。

RY_R8C38 ボード の CN3	信号の 方向	センサ基板 5 の CN1	内容
1 ピン(+5V)	—	1 ピン	センサ基板 Ver.5 の回路に+5V±10%(4.5~5.5V)を供給します。
2 ピン(P0_7)	←	2 ピン	U1(左から 1 番目のセンサ)からの信号を入力します。 "0":白色(LED1 点灯) "1":黒色(LED1 消灯)
3 ピン(P0_6)	←	3 ピン	U2(左から 2 番目のセンサ)からの信号を入力します。 "0":白色(LED3 点灯) "1":黒色(LED3 消灯)
4 ピン(P0_5)	←	4 ピン	U3(左から 3 番目のセンサ)からの信号を入力します。 "0":白色(LED5 点灯) "1":黒色(LED5 消灯)
5 ピン(P0_4)	←	5 ピン	U4(左から 4 番目のセンサ)からの信号を入力します。 "0":白色(LED7 点灯) "1":黒色(LED7 消灯)
6 ピン(P0_3)	←	6 ピン	U5(右から 4 番目のセンサ)からの信号を入力します。 "0":白色(LED9 点灯) "1":黒色(LED9 消灯)
7 ピン(P0_2)	←	7 ピン	U6(右から 3 番目のセンサ)からの信号を入力します。 "0":白色(LED11 点灯) "1":黒色(LED11 消灯)
8 ピン(P0_1)	←	8 ピン	U7(右から 2 番目のセンサ)からの信号を入力します。 "0":白色(LED13 点灯) "1":黒色(LED13 消灯)
9 ピン(P0_0)	←	9 ピン	U8(右から 1 番目のセンサ)からの信号と、U9(スタートバー検出センサ)の信号を入力します。 "0":白色(LED15 点灯) "1":黒色(LED15 消灯) または、 "0":スタートバーあり(LED15 点灯) "1":なし(LED15 消灯) です。スタート時、U8(右から 1 番目のセンサ)部分のコースは黒色なので、このときは U9(スタートバー)の反応になります。スタート後はスタートバー検出センサが反応することはないので、U8(コース)の反応になります。
10 ピン(GND)	—	10 ピン	GND

3.7 コースの白と黒を判断する仕組み

センサ基板には、コースへ赤外線を出す素子と、反射した赤外線を受ける素子が 8 組付いています。「白は光を反射する」、「黒は光を吸収する」ことを利用します。赤外線を出す素子を使って、コースへ赤外線を当てます。その赤外線が、赤外線を受ける素子で検出できれば“白”、できなければ“黒”と判断します。

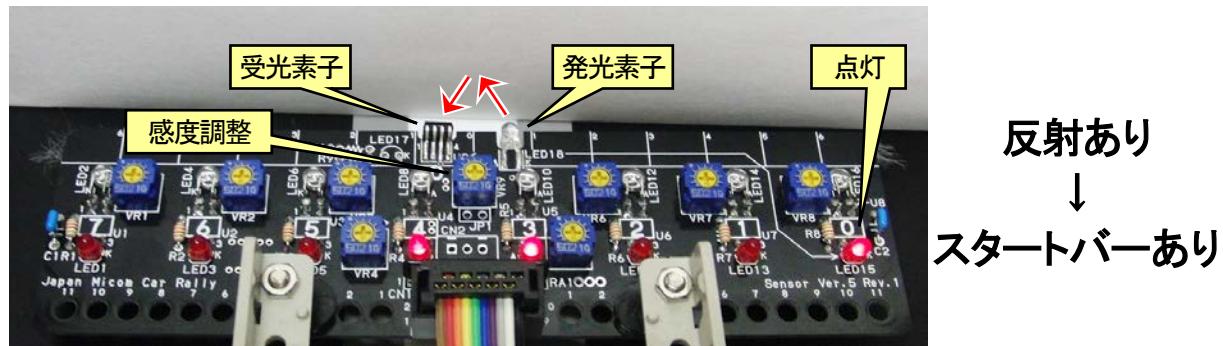


赤外線を出す量をボリュームで調整することができます。マイコンカーのコースには灰色があります。ボリュームの感度を変えることにより、灰色を“白”と判断させるか、“黒”と判断させるか調整することができます。**標準のプログラム**は、灰色を白色と判断させると良いようになっています。

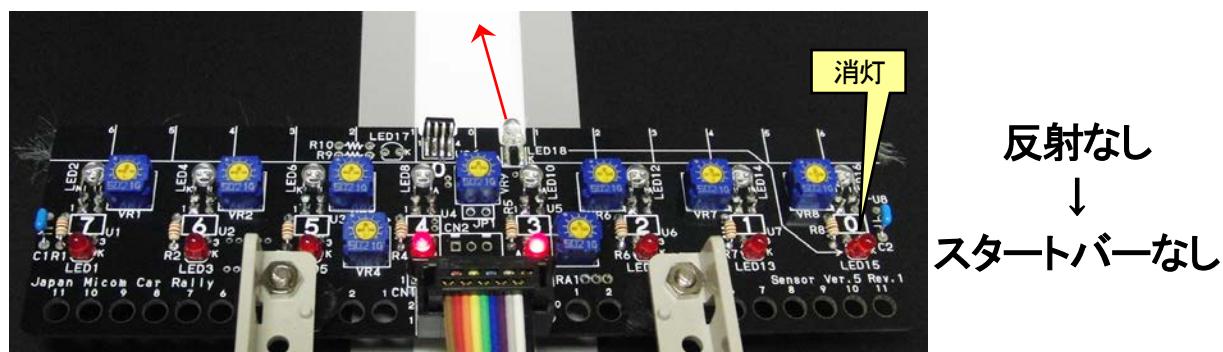
3.8 スタートバーの開閉を判断する仕組み

スタート時、白色のスタートバーが閉じています。センサ基板は赤外 LED と S6846(変調型フォトセンサ)を前方に取り付けており、センサの状況によって下記のように判断します。

●スタートバーが閉じているとき

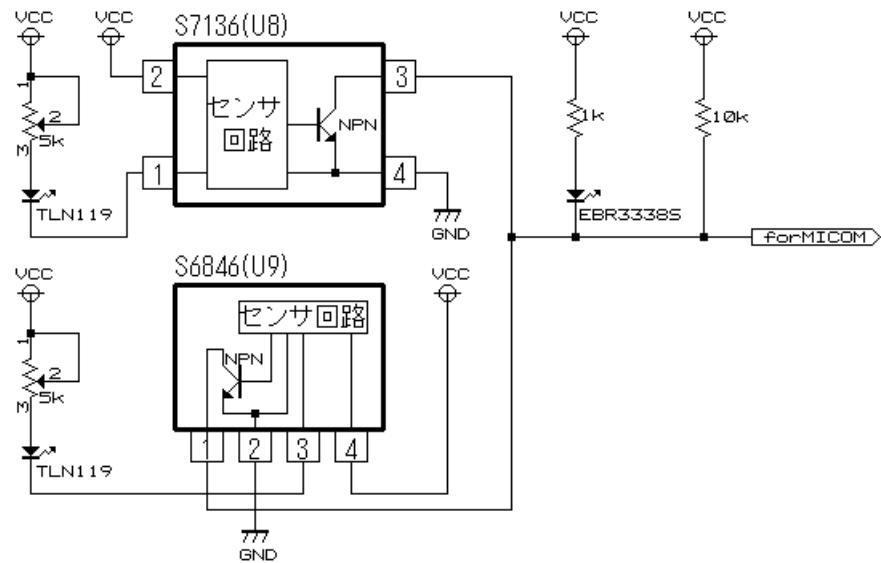


●スタートバーが開いているとき



3.9 U8 と U9 の出力信号

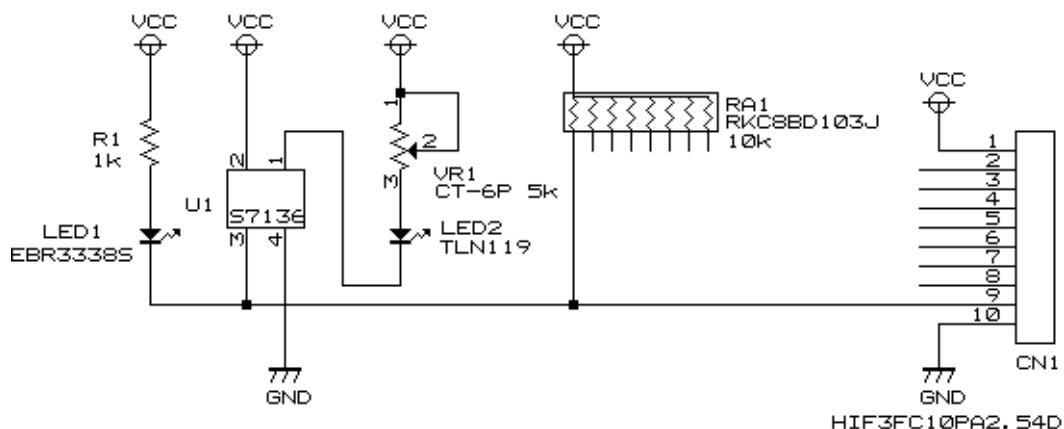
センサの出力は、オープンコレクタ出力で、NPN 型トランジスタ(2SC タイプ)が接続されています。CN1 の 9 ピンは、いちばん右のコース検出センサ(U8)とスタートバー検出センサ(U9)の出力を兼用しており、下記のような回路になっています。



2 つのセンサの反応と、出力信号を下表に示します。

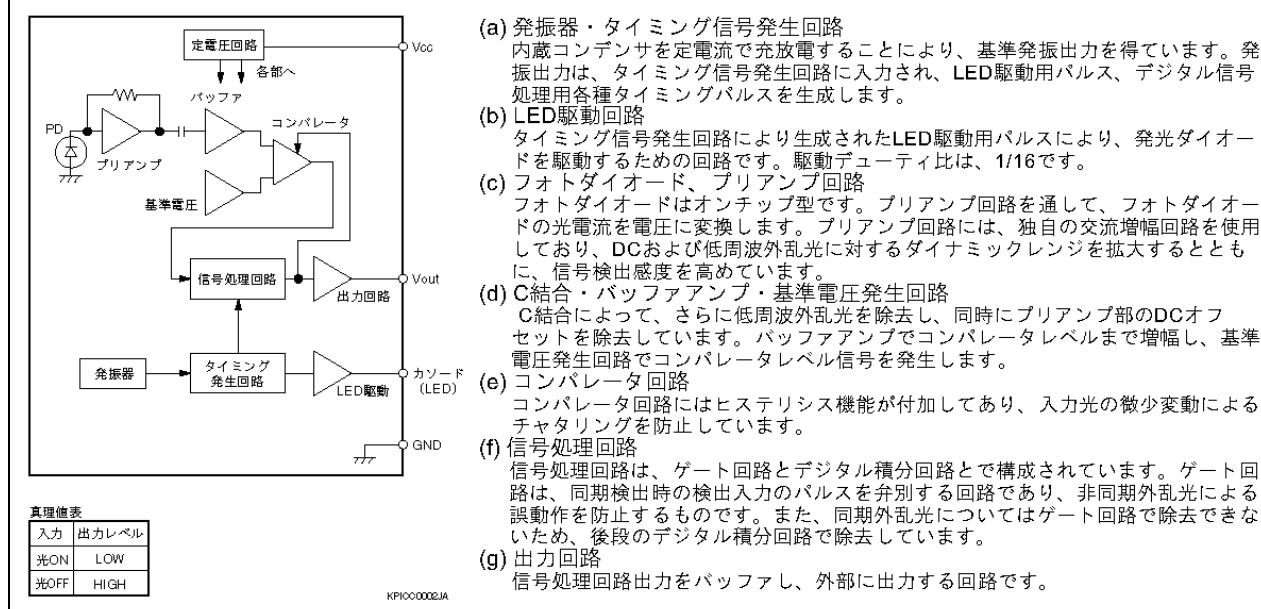
コース 検出センサ	スタートバー 検出センサ	回路	出力
			考え方
黒	なし		開放 (ハイインピーダンス) 実際は、プルアップ抵抗を付けているので 5V 出力になります。
黒	あり		0V スタート前に 0V のときは、スタートバー検出センサがあると判断します。
白	なし		0V スタート後に 0V のときは、コース検出センサが白と判断します。
白	あり		0V 両方が ON の状態は 考え方としては あり得ません。

3.10 センサ回路の原理

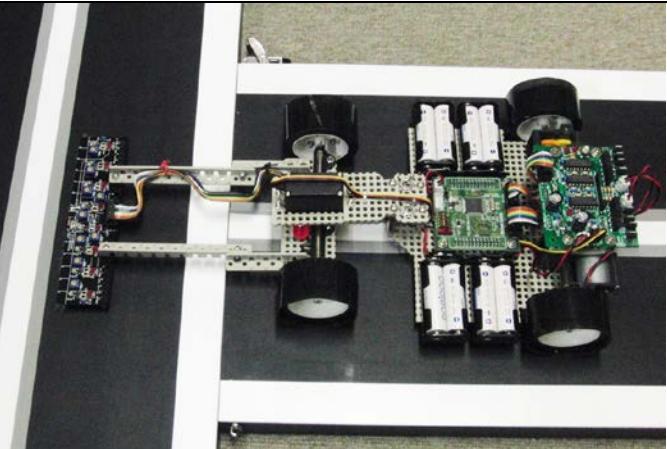


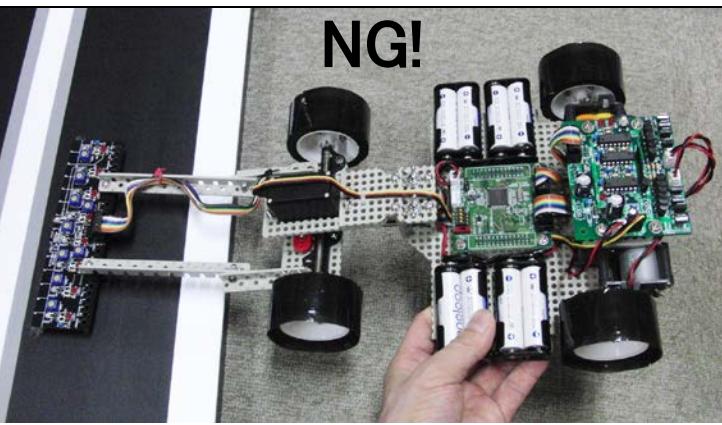
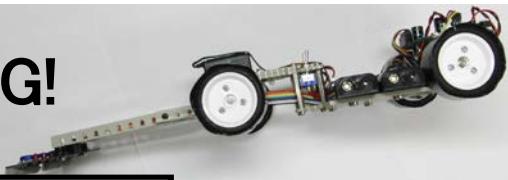
1. U1 がフォトセンサです。受光部と赤外 LED の発振回路を兼ね備えています。
2. U1 の 1 ピンに赤外 LED(LED2)が接続されています。ここで発光した光を U1 で受けます。赤外 LED の明るさ調整はポリューム VR1 で行います。
3. 光を受けたか受けないかを出力するのが U1 の 3 ピンです。LED(LED1)が接続されており "0" か "1" かを目で確かめることができます。
4. 赤外 LED の光が U1 に届くと(コースは白) "0" が出力されます。LED のアノード側が +、カソード側が - になるので LED は光ります。
5. 赤外 LED の光が U1 に届かなければ(コースは黒) "1" が出力されます(詳しくは次を参照)。LED のアノード側が +、カソード側も + になるので LED は光りません。
6. 先ほど、光が届かなければ "1" といいましたが、実は U1 の 3 ピンは、オープンコレクタ出力です。オープンコレクタ出力とは、"0" = 0V、それ以外はオープン、何処とも繋がっていない状態をいいます。デジタルの世界では、"0" でもない "1" でもない値はありません。そのため、抵抗(RA1)で信号をプルアップして、フォトセンサがオープンのときは "1" になるようにしています。

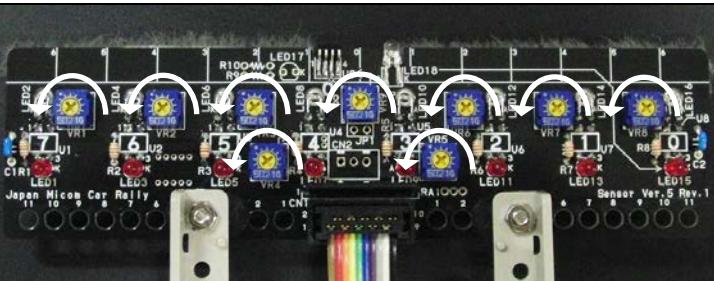
※参考資料－変調型フォトセンサ(S7136)の動作原理(データシートより)

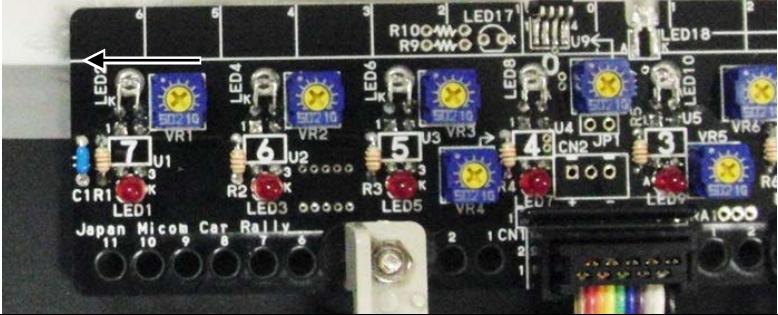
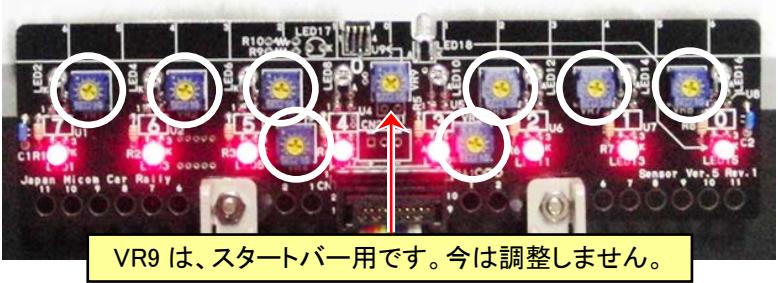


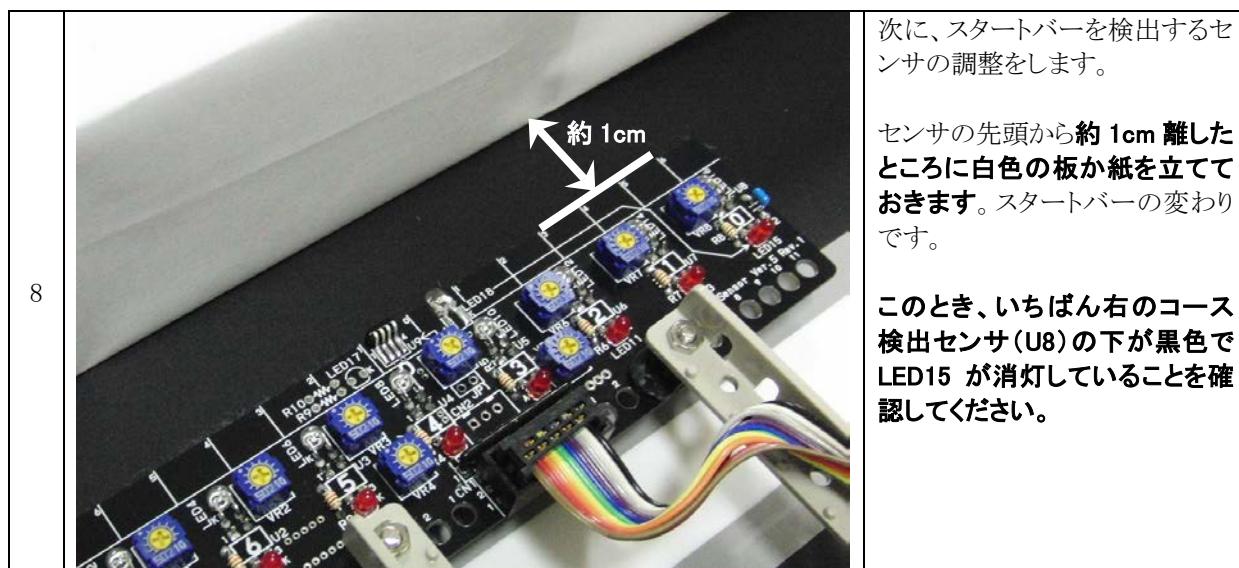
3.11 センサの調整方法

1	 <p>※横から見たところ</p> 	<p>写真のようにコース中心の灰色線とセンサ基板を平行に置きます。このときマイコンカーは、コース同一面上に置き、走っている状態と同じになります。</p>
---	--	--

2	 <p>※横から見たところ</p> 	<p>このように、手で持ちながらセンサの調整をしようとしてもセンサとコースとの間隔が一定にならないため、きちんと調整できません。必ずコース同一面上に置いてください。</p>
---	--	---

3	 <p>9 個のボリュームをすべて、反時計回りに回します。</p>	
---	--	--

		基板の横線とコースの白色と灰色の切り替わり部分を合わせます。真上から見て合わせるようしてください。
4		
5		8 個のボリュームを時計回りに回して LED が点くようにします。一つ一つゆっくりと回して、LED が点いた瞬間回すのを止めます。今回の調整で灰色も反応するように調整します。マイコンカーキットは、白色・灰色で反応するよう調整します。 VR9 は、スタートバー用です。今は調整しません。
6		センサを少し下げます(手前に引きます)。すべて消えます。
7		もう一度、センサを灰色の位置にゆっくりと平行に近づけます。点かない LED は感度を上げます(時計回り)。他の LED より先に点く場合、感度を下げます(反時計回り)。ほぼ同時に LED が点くように何度も調整します。



4. モータドライブ基板 Ver.5

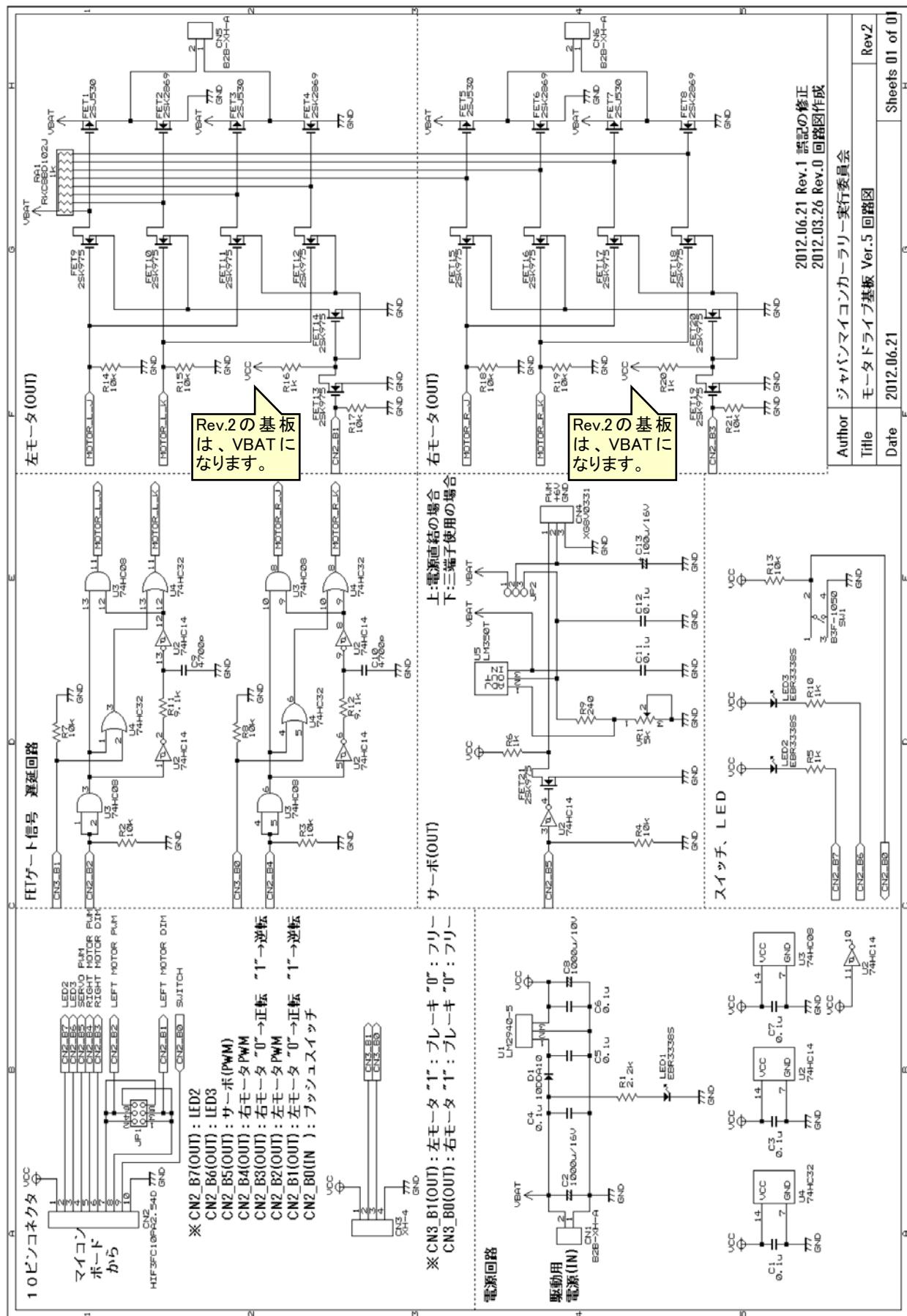
4.1 仕様

モータドライブ基板 Ver.5 の仕様を、下表に示します。

	モータドライブ基板 Ver.5	モータドライブ基板 Ver.4(参考)
略称	ドライブ基板 5	ドライブ基板 4
部品数	リード線のある部品:約 66 個 部品のピンの間隔は 2.54mm 以上	リード線のある部品:約 60 個 部品のピンの間隔は 2.54mm 以上
RY_R8C38 ボードとの接続方法	ポート 2 と接続	←
RY3048Fone ボードとの接続方法	ポート B と接続 ただし、JP1 のパターンカット、ショートの加工あり	←
制御できるモータ	2 個(左モータ、右モータ)	←
制御できるサーボ	1 個	←
プログラムで点灯、消灯できる LED	2 個	←
プッシュスイッチ	1 個	←
制御系電圧 (CN2 に加えることの出来る電圧)	DC5.0V±10%	←
駆動系電圧 (CN1 に加えることの出来る電圧)	4.5～5.5V、または 7～15V ただし 7V 以上の場合、「LM350 追加セット」によりマイコンボードに加える電圧を 5V、サーボに加える電圧を 6V にする必要あり	←
サーボ、モータ制御周期	モータ:16ms サーボ:16ms 個別設定不可	←
モータのフリー制御	フリー追加セットの追加で対応 ※モータの停止にはブレーキとフリーがあります。詳しくは、「フリー追加セット」部分を参照してください。	←
基板外形	幅 80×奥行き 65×厚さ 1.6mm	←
完成時の寸法(実寸)	幅 80×奥行き 65×高さ 20mm	←
重量	約 35g ※リード線の長さや半田の量で変わります	約 33g ※リード線の長さや半田の量で変わります
標準プログラム	R8C/38A マイコン:kit12_38a.c ※	R8C/38A マイコン:kit07_38a.c H8/3048F-ONE マイコン:kit07.c

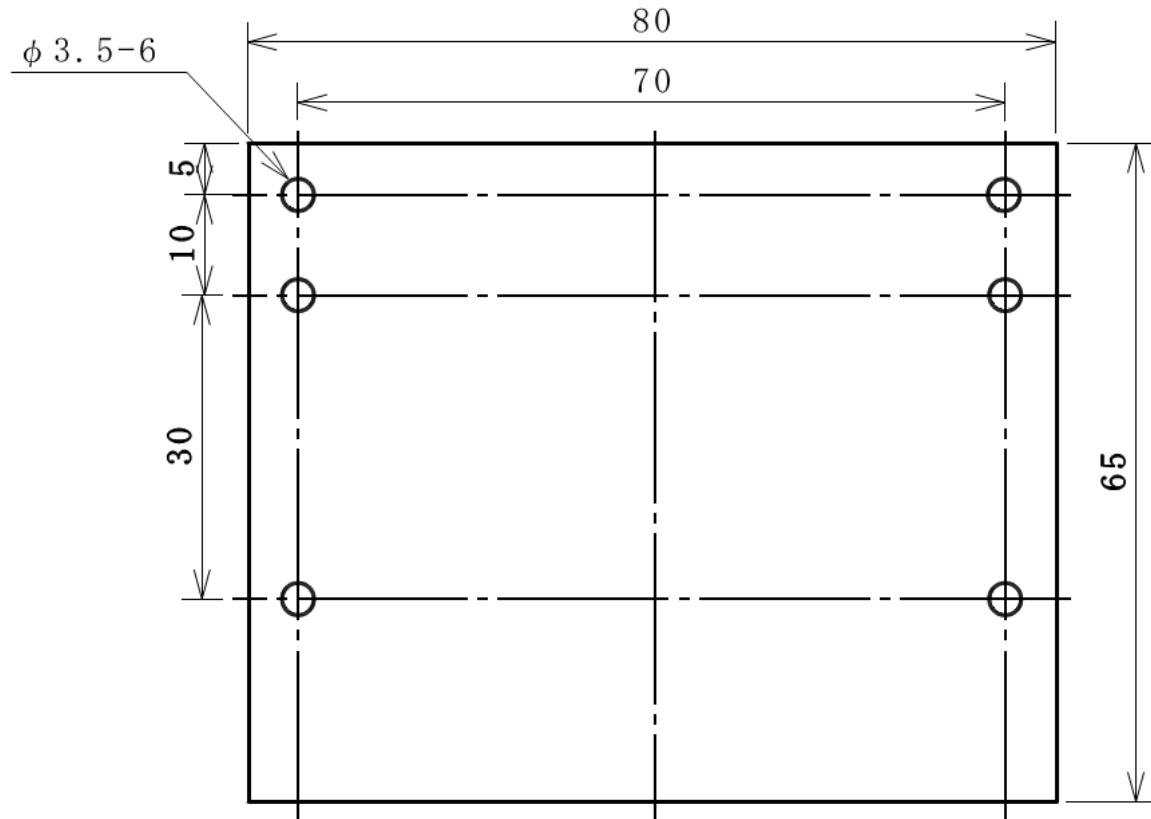
※モータドライブ基板がVer.4からVer.5に変更されたことによるプログラム変更はありません。「kit12_38a.c」はセンサ基板が Ver.4 から Ver.5 に変更されたことによるプログラム変更です。

4.2 回路図



4.3 寸法

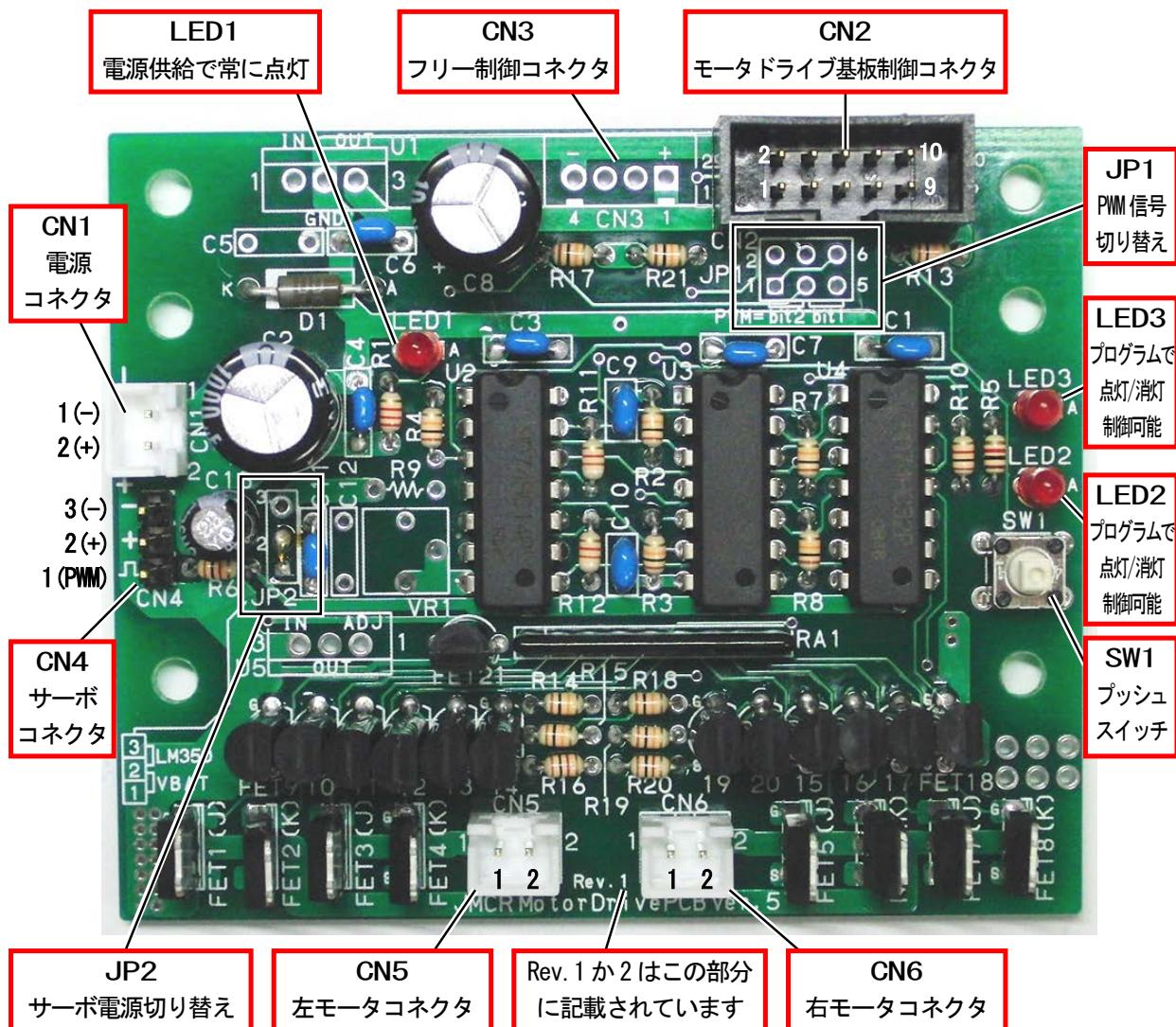
モータドライブ基板 Ver.5 には、取り付け用の穴が 6 個あります。この穴を使って、モータドライブ基板 Ver.5 をマイコンカーキットに固定してください。



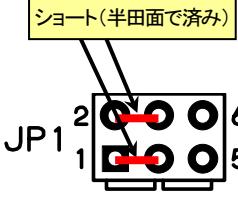
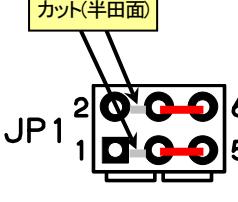
モータドライブ基板 Vol.3、Ver.4、Ver.5 は同じ外形です。

4.4 外観

モータドライブ基板 Ver.5 の外観を、下図に示します。



モータドライブ基板 Ver.5 のコネクタの接続先、信号の内容を下表に示します。

部品番号	接続先	pin	詳細
CN1	電源入力	1	GND
		2	+電源 (4.5~5.5V、または 7V~15V 入力) ※7V 以上の場合、別売りの「LM350 追加セット」の部品を取り付ける必要があります。
CN2	マイコンボードと接続	1~10	次項参照
CN3	マイコンボードと接続	1	+5V
		2	左モータの停止状態選択 1:フリー 0:ブレーキ
		3	右モータの停止状態選択 1:フリー 0:ブレーキ
		4	GND
CN4	サーボ	1	サーボ PWM 信号出力
		2	サーボ電源 (6V 出力)
		3	GND
CN5	左モータ	1、2	左モータ出力
CN6	右モータ	1、2	右モータ出力
JP1	左モータの PWM 信号切り替え	1~6	<p>左モータの PWM 出力端子、方向切り替え端子を切り替えるジャンパです。</p> <p>● RY_R8C38 ボードの場合</p>  <ul style="list-style-type: none"> 1-3 ピン間をショート 2-4 ピン間をショート 3-5 ピン間は無接続 4-6 ピン間は無接続 <p>※半田面でショート済みです。特に何もする必要はありません。</p> <p>● RY3048Fone ボードの場合</p>  <ul style="list-style-type: none"> 1-3 ピン間のパターンカット(半田面) 2-4 ピン間のパターンカット(半田面) 3-5 ピン間をショート 4-6 ピン間をショート

4. モータドライブ基板 Ver.5

JP2	サーボ電源切り替え	1~3	<p>JP2 は、サーボ電源ピン(CN2 の 2 ピン)への電源供給元を切り替える端子です。</p> <ul style="list-style-type: none"> ●CN1 に供給されている電源電圧が 6V 以下の場合 1-2 ピン間をショートさせます。CN1 の電源が直接、CN4 の 2 ピンに接続されます。 ●CN1 に供給されている電源電圧が 6V 以上の場合 サーボに加えることのできる電圧を超えていませんので、別売りの「LM350 追加セット」の部品を取り付けて、2-3 ピン間をショートさせます。LM350(三端子レギュレータ)を通して、6V の電圧が CN4 の 2 ピンに供給されます。
-----	-----------	-----	--

4.5 モータドライブ基板Ver.5 のCN2 とRY_R8C38 ボードとの関係

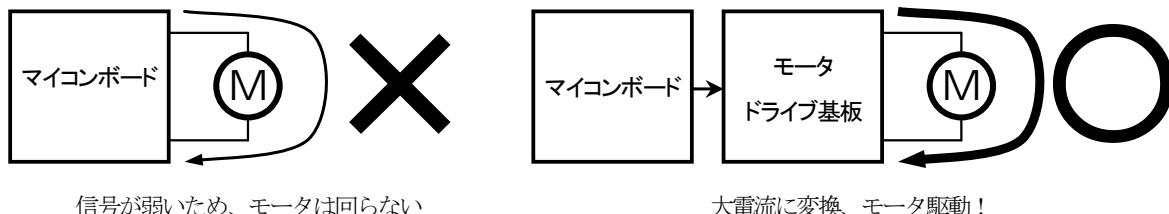
モータドライブ基板 Ver.5 の CN2 と RY_R8C38 ボードの CN4(ポート 2)を 10 ピンフラットケーブルで接続します。信号の内容を、下表に示します。

RY_R8C38 ボード の CN4	信号の 方向	モータドライブ 基板の CN2	内容
1 ピン(+5V)	—	1 ピン	<p>モータドライブ基板 Ver.5 のロジック IC などの制御系回路に供給する+5V です。LM350 追加セットを使う場合と使わない場合で 5V の供給元が変わります。</p> <ul style="list-style-type: none"> ●制御系と駆動系の電源を分けた場合 (LM350 追加セットを使わない場合) RY_R8C38 ボードからモータドライブ基板 Ver.5 の制御系回路 5V が供給されます。 ●制御系と駆動系の電源を共通にした場合 (LM350 追加セットを使う場合) モータドライブ基板 Ver.5 の LM2940-5 (5V を出力する三端子レギュレータ)からモータドライブ基板 Ver.5 の制御系回路と RY_R8C38 ボードへ供給します。
2 ピン(P2_7)	→	2 ピン	LED2 と接続されています。 "0": LED 点灯 "1": LED 消灯
3 ピン(P2_6)	→	3 ピン	LED3 と接続されています。 "0": LED 点灯 "1": LED 消灯
4 ピン(P2_5)	→	4 ピン	サーボへ PWM 信号を出力します。
5 ピン(P2_4)	→	5 ピン	右モータへ PWM 信号を出力します。
6 ピン(P2_3)	→	6 ピン	右モータの回転方向を制御します。 "0": 正転 "1": 逆転
7 ピン(P2_2)	→	7 ピン	左モータへ PWM 信号を出力します。
8 ピン(P2_1)	→	8 ピン	左モータの回転方向を制御します。 "0": 正転 "1": 逆転
9 ピン(P2_0)	←	9 ピン	プッシュスイッチ(SW1)の状態を検出します。 "0": 押されているとき "1": 離されているとき
10 ピン(GND)	—	10 ピン	GND

4.6 モータ制御

4.6.1 モータドライブ基板の役割

モータドライブ基板は、マイコンからの命令によってモータを動かします。マイコンからの「モータを回せ、止める」という信号は非常に弱く、その信号線に直接モータをつないでもモータは動きません。この弱い信号をモータが動くための数 A(アンペア)という大きな電流が流せる信号に変換します。

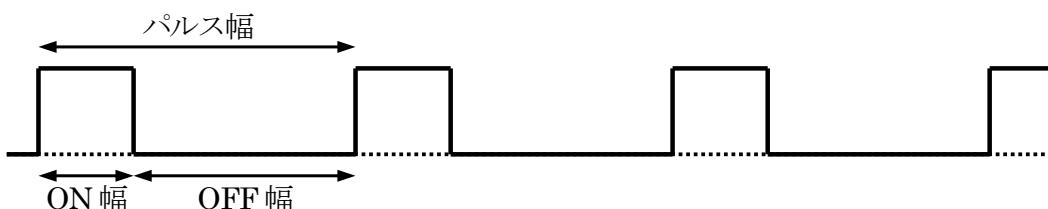


4.6.2 スピード制御の原理

モータを回したければ、電圧を加えれば回ります。止めたければ加えなければよいだけです。では、その中間のスピードや 10%、20%…など、細かくスピード調整したいときはどうすればよいのでしょうか。

ボリューム(半固定抵抗)を使えば電圧を落とすことができます。しかし、モータへは大電流が流れるため、許容電流の大きなボリュームが必要です。また、抵抗で分圧した分は、抵抗の熱となってしまいます。

そこで、スイッチで ON、OFF を高速に繰り返して、あたかも中間的な電圧が出ているような制御を行います。ON/OFF 信号は、周期を一定にして ON と OFF の比率を変える制御を行います。これを、「パルス幅変調 (Pulse Width Modulation)」といい、略して「PWM」といいます。パルス幅に対する ON の割合のことを「デューティ比」といいます。周期に対する ON 幅を 50%にするとき、デューティ比 50%といいます。他にも PWM50%とか、単純にモータ 50%といいます。



デューティ比の計算式を下記に示します。

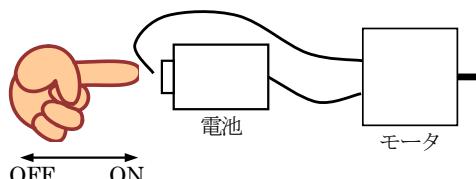
$$\text{デューティ比} = \text{ON 幅} / \text{パルス幅} (\text{ON 幅} + \text{OFF 幅})$$

例えば、100ms のパルスに対して、ON 幅が 60ms なら、

$$\text{デューティ比} = 60\text{ms} / 100\text{ms} = 0.6 = 60\%$$

となります。すべて ON なら、100%、すべて OFF なら 0%となります。

「PWM」と聞くと、何か難しく感じてしまいますが、下記のように手でモータと電池の線を「繋ぐ」、「離す」の繰り返し、それも PWM と言えます。繋いでいる時間が長いとモータは速く回ります。離している時間が長いとモータは少ししか回りません。人なら「繋ぐ」、「離す」動作をコンマ数秒でしか行えませんがマイコンなら数ミリ秒で行えます。

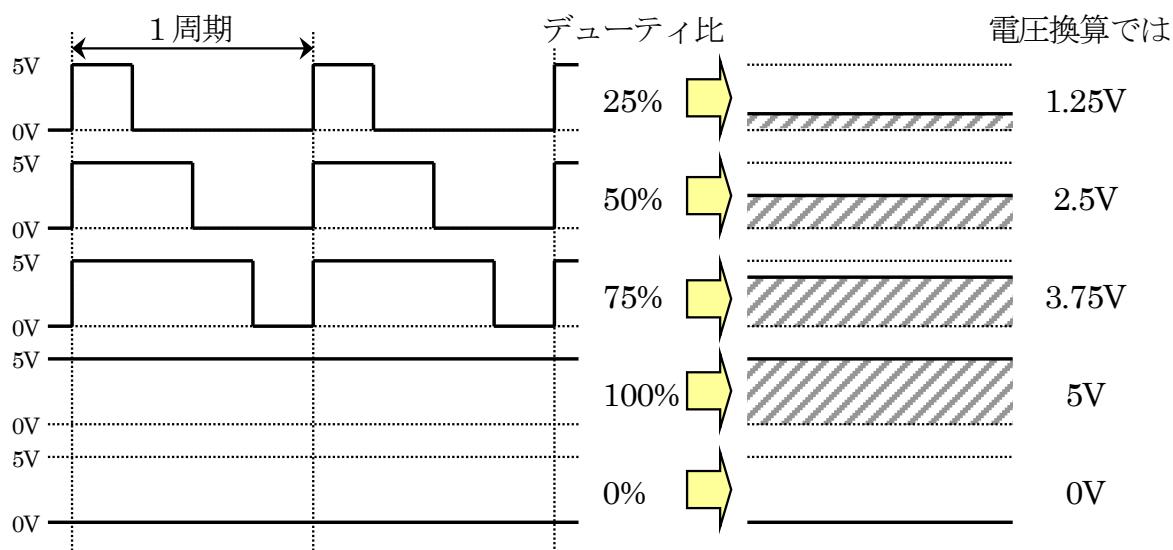


4. モータドライブ基板 Ver.5

下図のように、0Vと5Vを出力するような波形で考えてみます。1周期に対してONの時間が長ければ長いほど平均化した値は大きくなります。すべて5Vにすればもちろん平均化しても5V、これが最大の電圧です。ONの時間を半分の 50%にするとどうでしょうか。平均化すると $5V \times 0.5 = 2.5V$ と、あたかも電圧が変わったようになります。

このようにONにする時間を1周期の 90%, 80%, ..., 0%にすると徐々に平均した電圧が下がっていき最後には0Vになります。

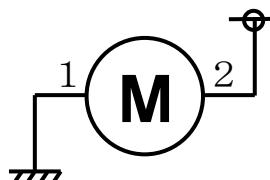
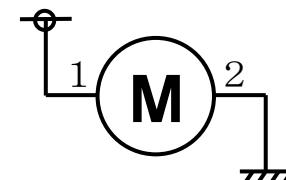
この信号をモータに接続すれば、モータの回転スピードも少しずつ変化させることができ、微妙なスピード制御が可能です。LEDに接続すれば、LEDの明るさを変えることができます。マイコンを使えばこの作業をマイクロ秒、ミリ秒単位で行うことができます。このオーダーでの制御になると、非常にスムーズなモータ制御が可能です。



なぜ電圧制御ではなくパルス幅制御でモータのスピードを制御するのでしょうか。マイコンは”0”か”1”かのデジタル値の取り扱いは大得意ですが、何 V というアナログ的な値は不得意です。そのため、”0”と”1”的幅を変えて、あたかも電圧制御しているように振る舞います。これが PWM 制御です。

4.6.3 正転、逆転の原理

モータドライブ基板 Ver.5 では、モータを「正転、逆転、停止」制御することができます。正転、逆転させるとき、モータの端子に加える電圧を下表に示します。

	正転	逆転
モータの端子に加える電圧	 1 ピンは GND (0V)、2 ピンはプラスに接続します。	 1 ピンはプラス、2 ピンは GND (0V)に接続します。

4.6.4 ブレーキとフリー

モータードライブ基板 Ver.5 の標準回路の停止は、ブレーキです。「フリー追加セット」の部品を追加すると、停止をブレーキとフリーの 2 種類にすることができます。

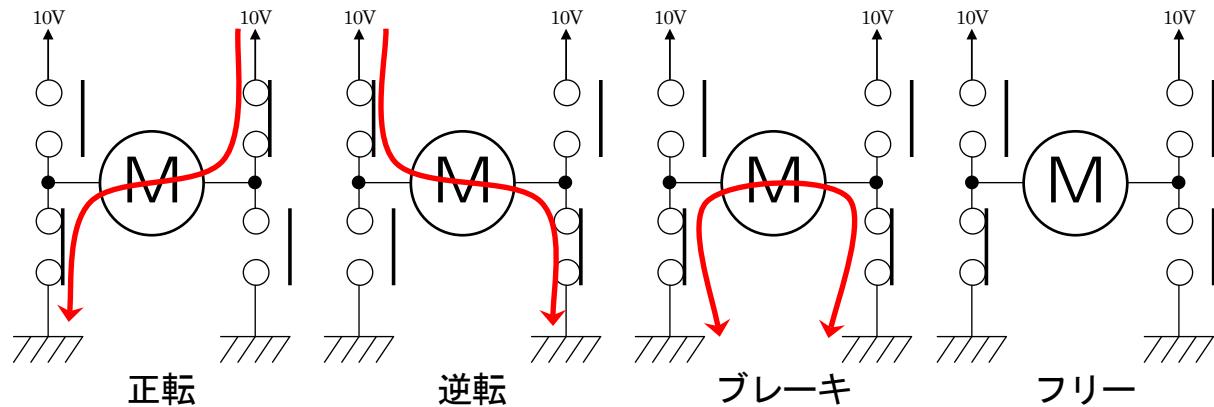
ブレーキとフリーの違いを、下表に示します。

	ブレーキ	フリー
モータに 加える電圧	<p>ブレーキ</p> <p>両端子を GND に接続します。結果、両端子がショートした状態になります。</p>	<p>フリー</p> <p>片側を無接続にします。結果、どこにも繋いでいない状態になります。</p>
スピードの 落ち方 (イメージ)	<p>正転 → ブレーキ</p>	<p>正転 → フリー</p>

フリーはブレーキと比べ、停止の減速が緩やかです。フリーは、スピードをゆっくり落としたい場合などに使用します。

4.6.5 Hブリッジ回路

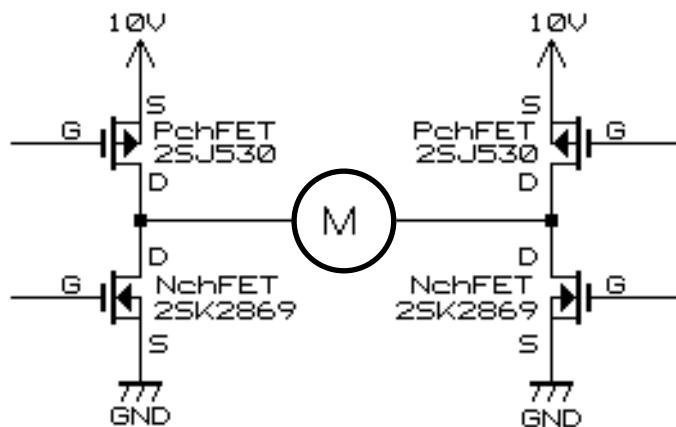
では、実際はどのようにするのでしょうか。下図のように、モータを中心として H 型に4つのスイッチを付けます。この4つのスイッチをそれぞれ ON/OFF することにより、正転、逆転、ブレーキ、フリー制御を行います。H 型をしていることから「H ブリッジ回路」と呼ばれています。



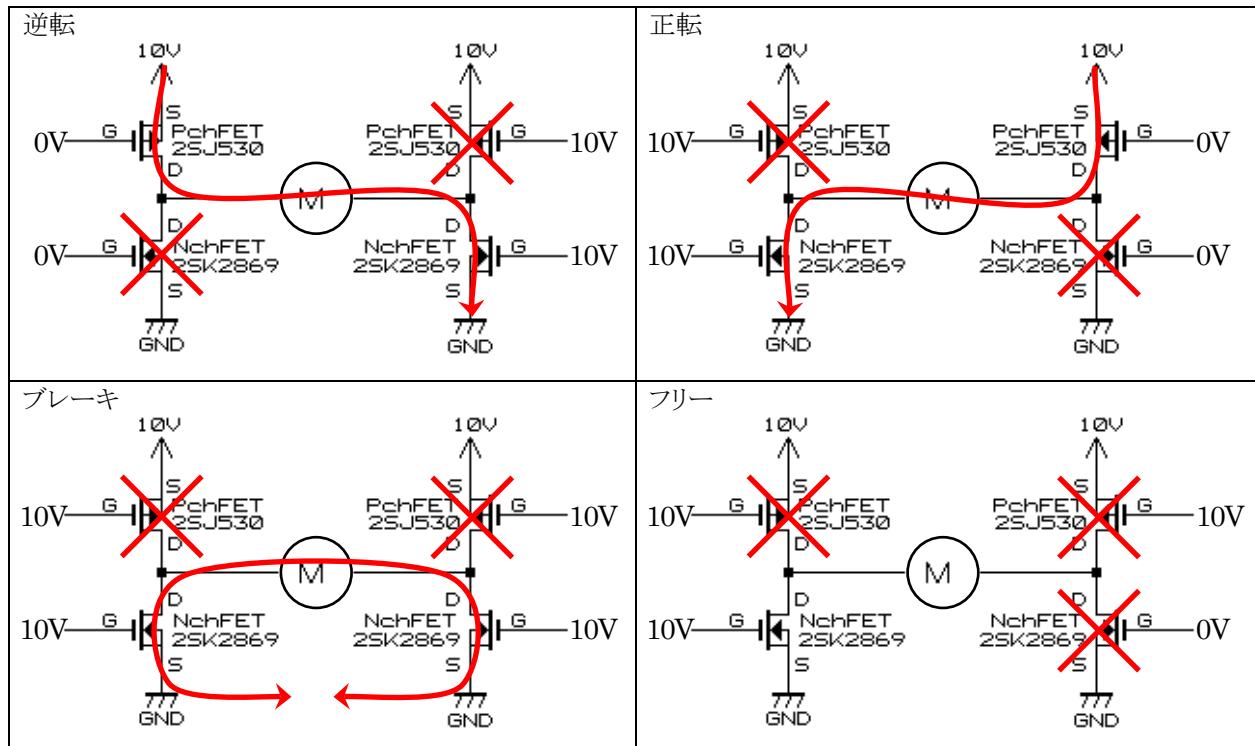
4.6.6 Hブリッジ回路のスイッチをFETにする

スイッチ部分を FET にします。電源のプラス側に P チャネル FET (2SJ タイプ)、マイナス側に N チャネル FET (2SK タイプ) を使用します。

P チャネル FET は、 V_G (ゲート電圧) $< V_s$ (ソース電圧) のとき、D-S(ドレイン-ソース) 間に電流が流れます。
N チャネル FET は、 V_G (ゲート電圧) $> V_s$ (ソース電圧) のとき、D-S(ドレイン-ソース) 間に電流が流れます。

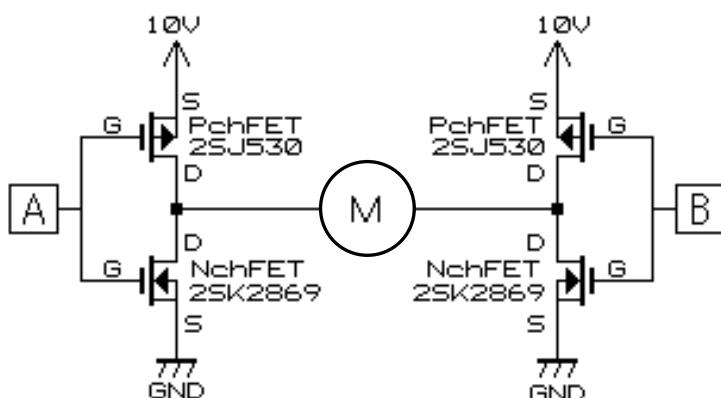


これら 4 つの FET のゲートに加える電圧を変えることにより、正転、逆転、ブレーキの動作を行います。



注意点は、絶対に左側 2 個もしくは右側 2 個の FET を同時に ON させてはいけないということです。同時に ON にすると、10V から GND へ何の負荷もないまま繋がりますのでショートと同じです。FET が燃えるかパタンが燃えるか…いずれにせよ危険です。

4 つのゲート電圧を見ると、左側の P チャネル FET と N チャネル FET、右側の P チャネル FET と N チャネル FET に加える電圧が共通であることが分かります。そのため、下記のような回路にしてみました。

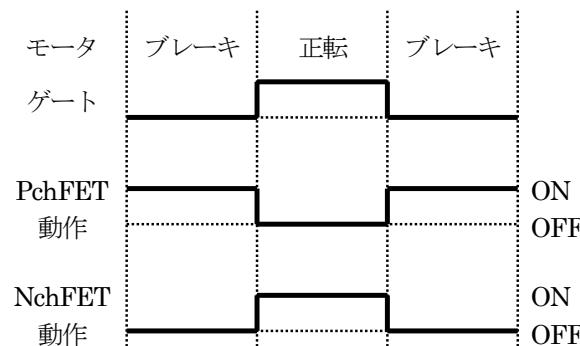


A	B	動作
0V	0V	ブレーキ
0V	10V	逆転
10V	0V	正転
10V	10V	ブレーキ

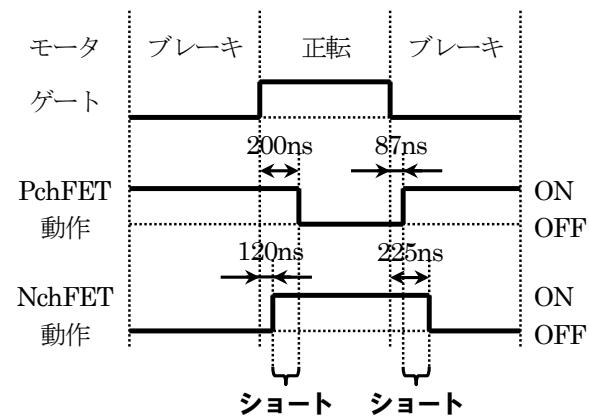
※G(ゲート)端子にはモータ用の電源電圧が 10V であったとすれば、その電圧がそのまま加えられたり 0V が加えられたりします。“0”、“1”的制御信号とは異なるので注意しましょう。

この回路を実際に組んで PWM 波形を加え動作させると、FET が非常に熱くなりました。どうしてでしょうか。FET のゲートから信号を入力し、ドレイン・ソース間が ON/OFF するとき、次ページの左図「理想的な波形」のように、P チャネル FET と N チャネル FET がすぐに反応してブレーキと正転がスムーズに切り替わりるように思えます。しかし、実際にはすぐには動作せず遅延時間があります。この遅延時間は FET が OFF→ON のときより、ON→OFF のときの方が長くなっています。そのため、次ページの右図「実際の波形」のように、短い時間ですが両 FET が ON 状態となり、ショートと同じ状態になってしまいます。

理想的な波形



実際の波形



ON してから実際に反応し始めるまでの遅延を「ターン・オン遅延時間」、ON になり始めてから実際に ON するまでを「上昇時間」、OFF してから実際に反応し始めるまでの遅延を「ターン・オフ遅延時間」、OFF になり始めてから実際に OFF するまでを「下降時間」といいます。

実際に OFF→ON するまでの時間は「ターン・オン遅延時間 + 上昇時間」、ON→OFF するまでの時間は「ターン・オフ遅延時間 + 下降時間」となります。上右図に出ている遅れの時間は、これらの時間のことです。

モータドライブ基板で使用しているルネサス エレクトロニクス製の FET「2SJ530」と「2SK2869」の電気的特性を下記に示します。

2SJ530(P チャネル)

電気的特性						
項目	記号	Min	Typ	Max	単位	測定条件
ドレイン・ソース破壊電圧	$V_{BDS(oss)}$	-60	—	—	V	$I_D = 10mA, V_{GS} = 0$
ゲート・ソース破壊電圧	V_{BGSOSS}	± 20	—	—	V	$I_G = \pm 100\mu A, V_{GS} = 0$
ドレイン遮断電流	I_{DSS}	—	—	-10	μA	$V_{GS} = -60V, V_{DS} = 0$
ゲート遮断電流	I_{GSS}	—	—	± 10	μA	$V_{GS} = \pm 16V, V_{DS} = 0$
ゲート・ソース遮断電圧	$V_{GS(OFF)}$	-1.0	—	-2.0	V	$V_{GS} = -10V, I_G = 1mA$
順伝導アドミタンス	$ Y_{ds} $	6.5	11	—	S	$I_D = 8A, V_{GS} = 10V^{*4}$
ドレイン・ソースオン抵抗	$R_{DS(on)}$	—	0.08	0.10	Ω	$I_D = 8A, V_{GS} = 10V^{*4}$
ドレイン・ソースオフ抵抗	$R_{DS(off)}$	—	0.11	0.16	Ω	$I_D = 8A, V_{GS} = 4V^{*4}$
入力容量	C_{iss}	—	850	—	pF	$V_{GS} = -10V, V_{DS} = 0$
出力容量	C_{oss}	—	420	—	pF	$f = 1MHz$
帰還容量	C_{rss}	—	110	—	pF	
ターン・オン遅延時間	$t_{d(on)}$	—	12	—	ns	$V_{GS} = -10V, I_D = 8A$
上昇時間	t_r	—	75	—	ns	$R_L = 3.75\Omega$
ターン・オフ遅延時間	$t_{d(off)}$	—	125	—	ns	
下降時間	t_f	—	75	—	ns	
ダイオード順電圧	V_{DF}	—	-1.1	—	V	$I_F = -15A, V_{GS} = 0$
逆回復時間	t_{rr}	—	70	—	ns	$I_F = -15A, V_{GS} = 0$ $dIF/dt = 50A/\mu s$

注) 4. パルス測定

OFF→ON は
87ns 遅れる

ON→OFF は
200ns 遅れる

2SK2869(N チャネル)

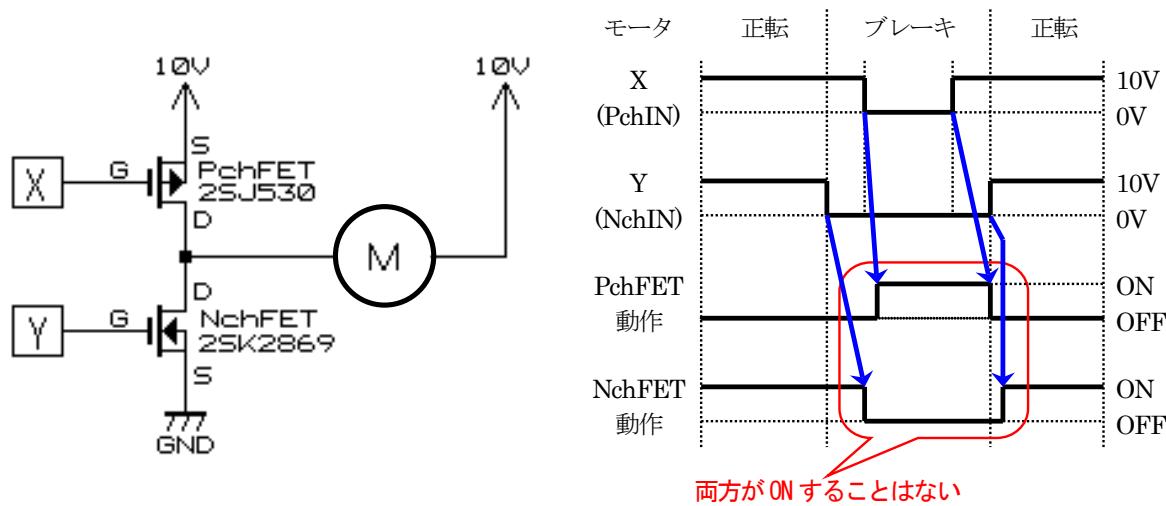
電気的特性						
項目	記号	Min	Typ	Max	単位	測定条件
ドレイン・ソース破壊電圧	$V_{BDS(oss)}$	60	—	—	V	$I_D = 10mA, V_{GS} = 0$
ゲート・ソース破壊電圧	V_{BGSOSS}	± 20	—	—	V	$I_G = \pm 100\mu A, V_{GS} = 0$
ドレイン遮断電流	I_{DSS}	—	—	10	μA	$V_{GS} = 60V, V_{DS} = 0$
ゲート遮断電流	I_{GSS}	—	—	± 10	μA	$V_{GS} = \pm 16V, V_{DS} = 0$
ゲート・ソース遮断電圧	$V_{GS(OFF)}$	1.5	—	2.5	V	$V_{GS} = 10V, I_G = 1mA$
順伝導アドミタンス	$ Y_{ds} $	10	16	—	S	$I_D = 10A, V_{GS} = 10V^{*4}$
ドレイン・ソースオン抵抗	$R_{DS(on)}$	—	0.033	0.045	Ω	$I_D = 10A, V_{GS} = 10V^{*4}$
ドレイン・ソースオフ抵抗	$R_{DS(off)}$	—	0.055	0.07	Ω	$I_D = 10A, V_{GS} = 4V^{*4}$
入力容量	C_{iss}	—	740	—	pF	$V_{GS} = 10V, V_{DS} = 0$
出力容量	C_{oss}	—	380	—	pF	$f = 1MHz$
帰還容量	C_{rss}	—	140	—	pF	
ターン・オン遅延時間	$t_{d(on)}$	—	10	—	ns	$V_{GS} = 10V, I_D = 10A$
上昇時間	t_r	—	110	—	ns	$R_L = 3\Omega$
ターン・オフ遅延時間	$t_{d(off)}$	—	105	—	ns	
下降時間	t_f	—	120	—	ns	
ダイオード順電圧	V_{DF}	—	1.0	—	V	$I_F = 20A, V_{GS} = 0$
逆回復時間	t_{rr}	—	40	—	ns	$I_F = 20A, V_{GS} = 0$ $dIF/dt = 50A/\mu s$

OFF→ON は
120ns 遅れる

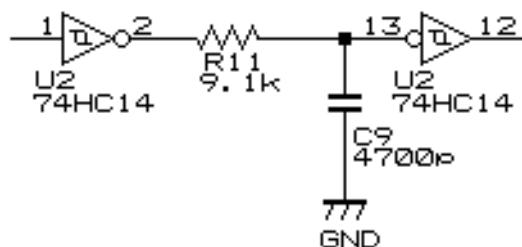
ON→OFF は
225ns 遅れる

4.6.7 PチャネルFETとNチャネルFETの短絡防止回路

短絡の解決策としては、先ほどの回路図にある A 側の P チャネル FET と N チャネル FET を同時に ON、OFF するのではなく、少し時間をずらしてショートさせないようにします。



この時間をずらす部分を、積分回路で作ります。積分回路については、多数の専門書があるので、そちらを参考してください。下記に積分回路を示します。

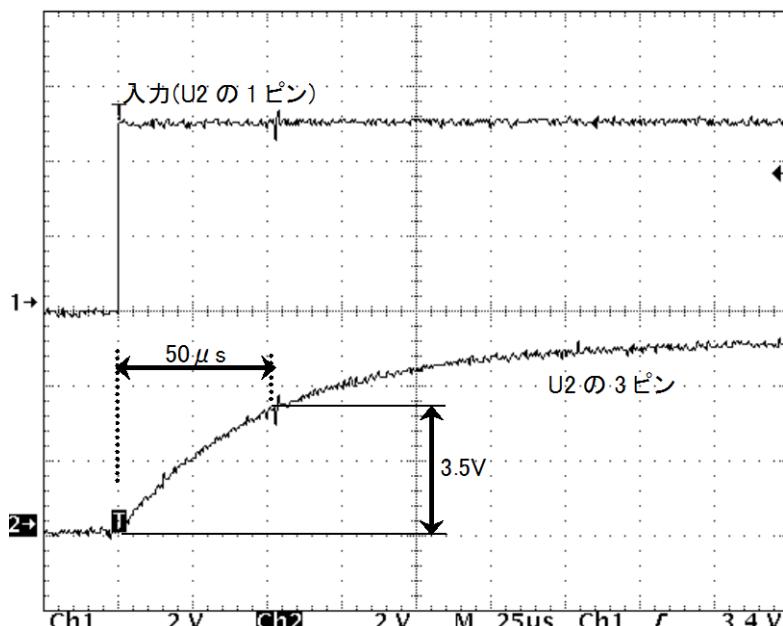


遅延時間は下記の式で計算することができます。

$$\text{時定数 } T = CR \text{ [s]}$$

今回は $9.1\text{k}\Omega$ 、 4700pF なので、計算すると下記の時間になります。

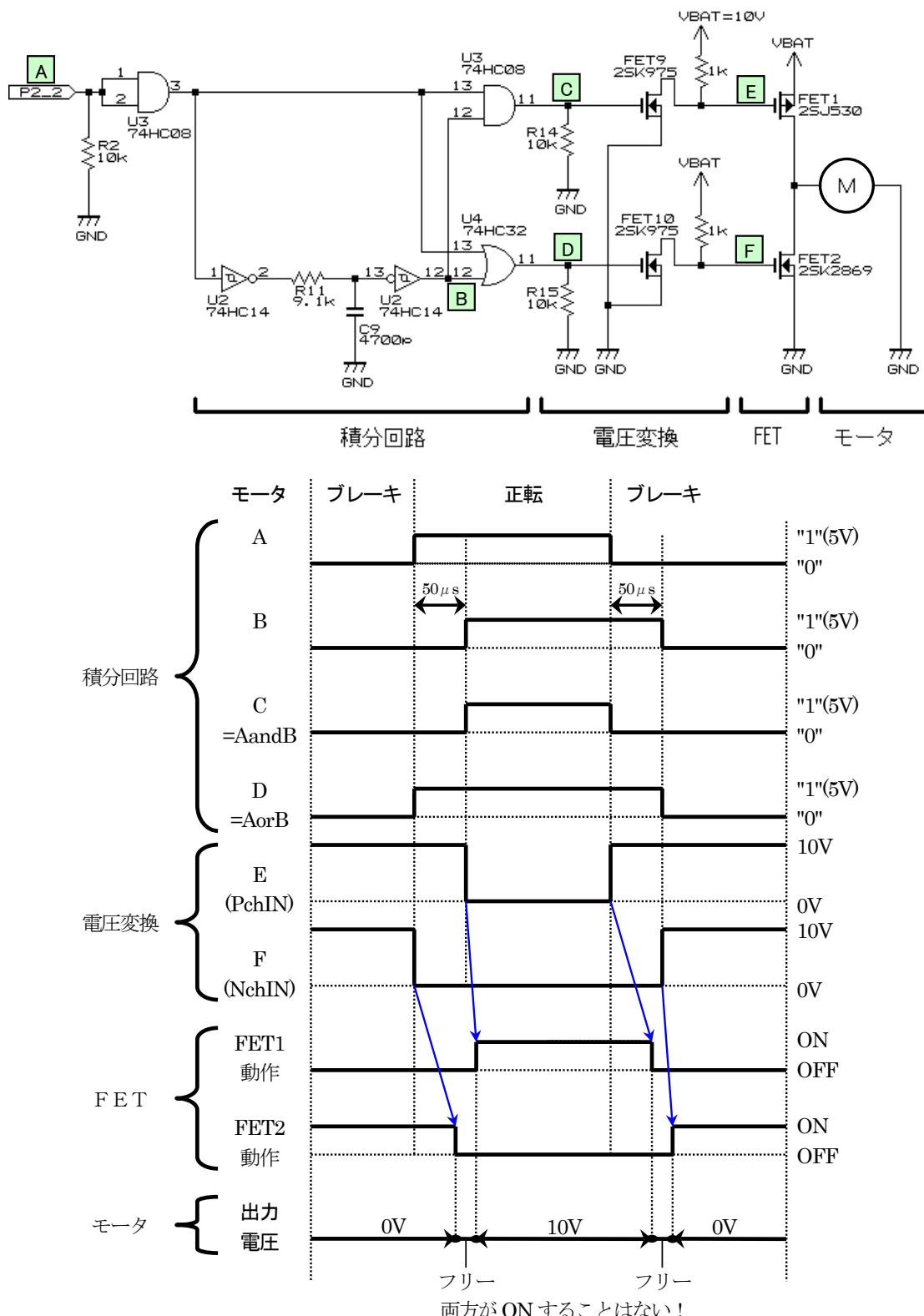
$$\begin{aligned} T &= (9.1 \times 10^3) \times (4700 \times 10^{-12}) \\ &= 42.77[\mu\text{s}] \end{aligned}$$



74HC シリーズは 3.5V 以上の入力電圧があると "1" とみなします。実際に波形を観測し、3.5V になるまでの時間を計ると約 $50\mu\text{s}$ になりました。

先ほどの「実際の波形」の図では最高でも 225ns のずれしかありませんが、積分回路では $50\mu\text{s}$ の遅延時間を作っています。これは、FET 以外にも、その前段にある電圧変換用の FET の遅延時間、FET のゲートのコンデンサ成分による遅れなどを含めたためです。

積分回路と FET を合わせた回路は下記のようになります。



(1) ブレーキ→正転に変えるとき

1. **A**点の信号は"0"でブレーキ、"1"で正転です。**A**点の出力を"0"(ブレーキ)から"1"(正転)へ変えます。
2. **B**点は積分回路により、 $50\ \mu s$ 遅れた波形が出力されます。
3. **C**点は、A and B の波形が出力されます。
4. **D**点は、A or B の波形が出力されます。
5. **E**点は、FET9 で電圧変換された信号が出力されます。**C**点の 0V–5V 信号が、10V–0V 信号へと変換されます。
6. **F**点も同様に**D**点の 0V–5V 信号が、10V–0V 信号へと変換されます。
7. **A**点の信号を"0"→"1"にかえると、FET2 のゲートが 10V→0V となり FET2 は OFF になります。ただし、遅延時間があるため遅れて OFF になります。この時点では、FET1 も FET2 も OFF 状態のため、モータはフリー状態となります。
8. **A**点の信号を変えてから $50\ \mu s$ 後、今度は FET1 のゲートが 0V→10V となり ON します。10V がモータに加えられ正転します。

(2) 正転→ブレーキに変えるとき

1. **A**点の信号を"1"(正転)から"0"(ブレーキ)にかえると、FET1 のゲート電圧が 0V から VBAT となり FET1 は OFF になります。ただし、遅延時間があるため遅れて OFF になります。この時点では、FET1 も FET2 も OFF 状態のため、モータはフリー状態となります。
2. **A**点の信号を変えてから $50\ \mu s$ 後、今度は FET2 のゲートが 0V→10V となり ON します。0V がモータに加えられ、両端子 0V なのでブレーキ動作になります。

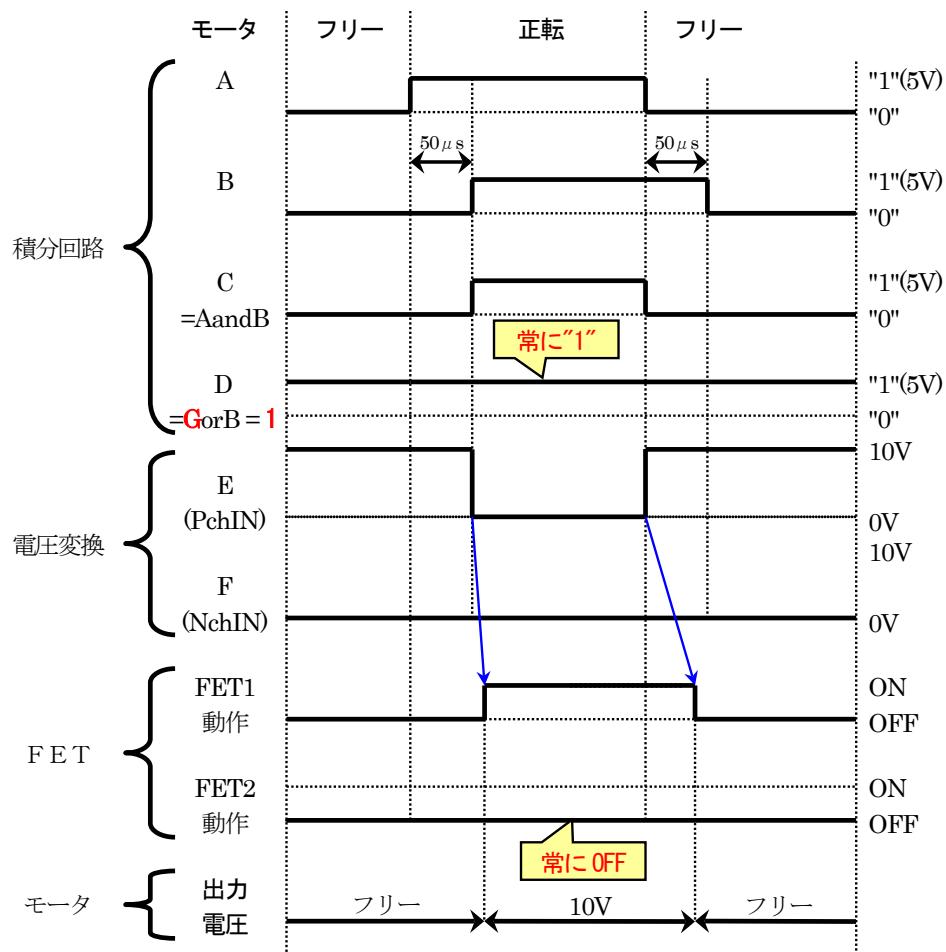
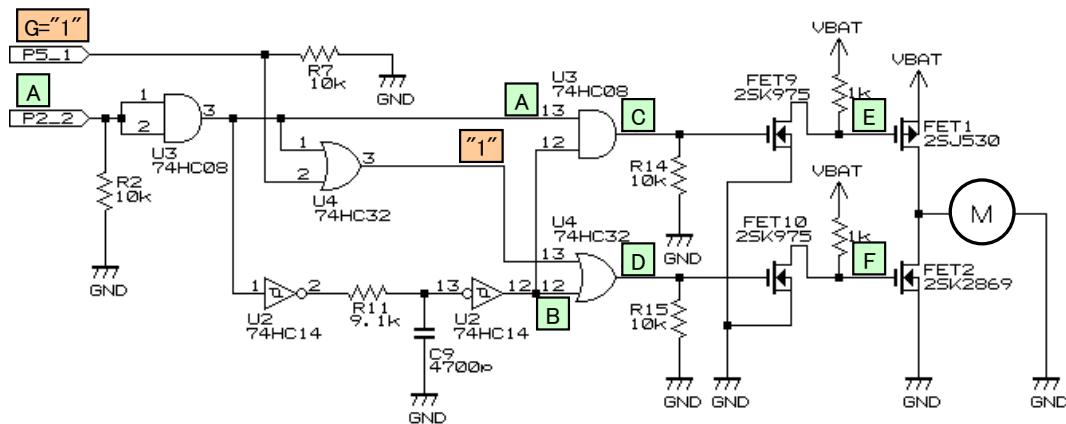
このように、動作を切り替えるときはいったん、両 FET ともフリー状態を作つて、短絡するのを防いでいます。

※ゲートに加える電圧の 10V は例です。実際は電源電圧(VBAT)になります。

4.6.8 フリー回路

ここでいうフリー回路は、P チャネル FET と N チャネル FET の短絡防止ではなく、モータの停止状態をブレーキとフリーにすることです。

モータドライブ基板 Ver.5 は「フリー追加セット」を追加することにより、停止動作をブレーキとフリーにすることができます。G 点が“1”的ときの様子を下図に示します。



G 点が“1”なら、D 点は A 点や B 点の状態に関わらず“1”になります。よって、FET2 は常に OFF になり、モータは正転とフリーを繰り返します。

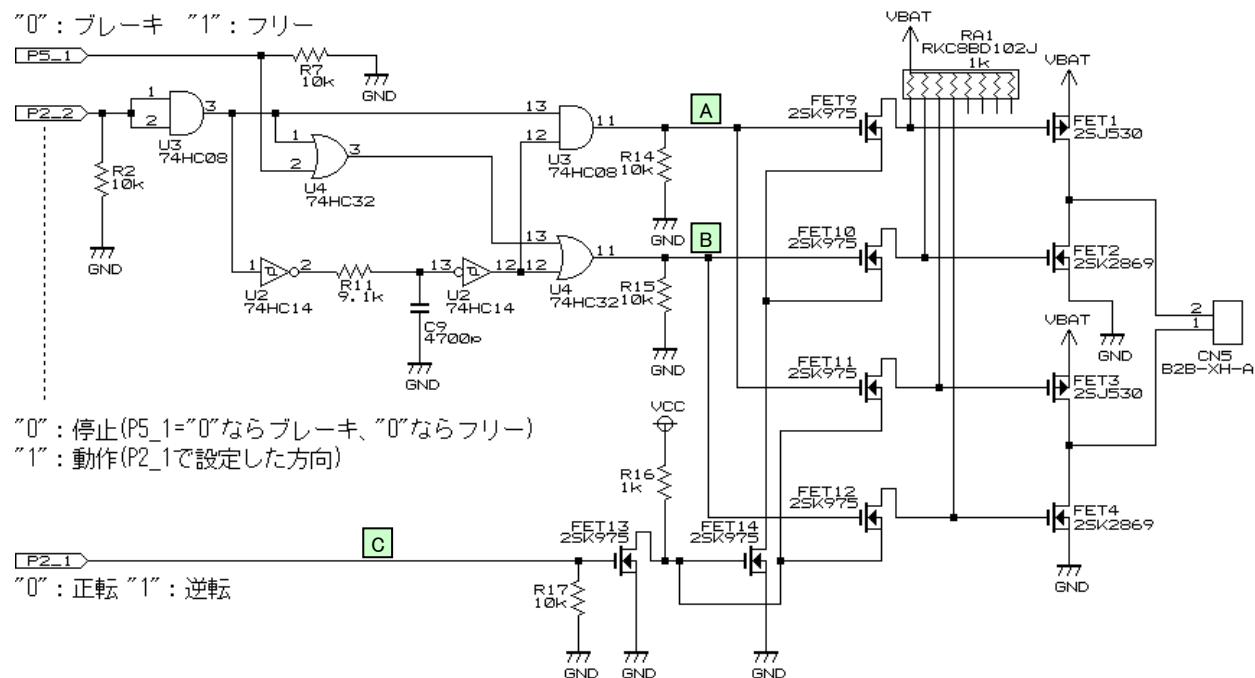
G 点が“0”的ときは前記のとおり、正転とブレーキを繰り返します。

4.6.9 実際の回路

実際の回路は、積分回路、FET 回路、フリー回路の他、正転／逆転切り替え用回路が付加されています。下記回路は、左モータ用の回路です。端子は、下記の 3 本あります。

- P2_2…PWM を加える端子
- P2_1…正転／逆転を切り替える端子
- P5_1…ブレーキ／フリーを切り替える端子です。

(1) 回路図



(2) 方向: 正転、停止: ブレーキのときの信号のレベルとモータの動作

A	B	C	FET1 のゲート	FET2 のゲート	FET3 のゲート	FET4 のゲート	CN5 2 ピン	CN5 1 ピン	モータ動作
0	0	0	10V (OFF)	10V (ON)	10V (OFF)	10V (ON)	0V	0V	ブレーキ
0	1		10V (OFF)	0V (OFF)	10V (OFF)	10V (ON)	フリー (開放)	0V	フリー
1	1		0V (ON)	0V (OFF)	10V (OFF)	10V (ON)	10V	0V	正転
0	1		10V (OFF)	0V (OFF)	10V (OFF)	10V (ON)	フリー (開放)	0V	フリー
0	0		10V (OFF)	10V (ON)	10V (OFF)	10V (ON)	0V	0V	ブレーキ

※A,B,C:"0"=0V、"1"=5V

(3) 方向:逆転、停止:ブレーキのときの信号のレベルとモータの動作

A	B	C	FET1 の ゲート	FET2 の ゲート	FET3 の ゲート	FET4 の ゲート	CN5 2 ピン	CN5 1 ピン	モータ動作
0	0	1	10V (OFF)	10V (ON)	10V (OFF)	10V (ON)	0V	0V	ブレーキ
0	1		10V (OFF)	10V (ON)	10V (OFF)	0V (OFF)	0V	フリー (開放)	フリー
1	1		10V (OFF)	10V (ON)	0V (ON)	0V (OFF)	0V	10V	逆転
0	1		10V (OFF)	10V (ON)	10V (OFF)	0V (OFF)	0V	フリー (開放)	フリー
0	0		10V (OFF)	10V (ON)	10V (OFF)	10V (ON)	0V	0V	ブレーキ

(4) 方向:正転、停止:フリーのときの信号のレベルとモータの動作

A	B	C	FET1 の ゲート	FET2 の ゲート	FET3 の ゲート	FET4 の ゲート	CN5 2 ピン	CN5 1 ピン	モータ動作
0	1	0	10V (OFF)	0V (OFF)	10V (OFF)	10V (ON)	フリー (開放)	0V	フリー
0	1		10V (OFF)	0V (OFF)	10V (OFF)	10V (ON)	フリー (開放)	0V	フリー
1	1		0V (ON)	0V (OFF)	10V (OFF)	10V (ON)	10V	0V	正転
0	1		10V (OFF)	0V (OFF)	10V (OFF)	10V (ON)	フリー (開放)	0V	フリー
0	1		10V (OFF)	0V (OFF)	10V (OFF)	10V (ON)	フリー (開放)	0V	フリー

4.6.10 左モータの動作

左モータは、P2_1、P2_2、P5_1 の 3 端子で制御します。フリー追加セットが無い場合、P5_1 の部分は常に "0" になります。

左モータ方向 P2_1	左モータ PWM P2_2	左モータ停止動作 P5_1	モータ動作
0	PWM	0	PWM="1"なら正転、"0"ならブレーキ
0	PWM	1	PWM="1"なら正転、"0"ならフリー
1	PWM	0	PWM="1"なら逆転、"0"ならブレーキ
1	PWM	1	PWM="1"なら逆転、"0"ならフリー

左モータを正転とブレーキで動作させたい場合、P2_1="0"、P5_1="0"にして P2_2 から PWM 波形を出力すると、左モータが PWM の割合に応じて正転します。例えば、PWM が 0%ならモータの回転は停止、PWM が 50%ならモータの回転は正転 50%、PWM100%ならモータの回転は正転 100%になります。このときの停止は、ブレーキ動作になります。

4.6.11 右モータの動作

右モータは、P2_3、P2_4、P5_0 の 3 端子で制御します。フリー追加セットが無い場合、P5_0 の部分は常に "0" になります。

右モータ方向 P2_3	右モータ PWM P2_4	右モータ停止動作 P5_0	モータ動作
0	PWM	0	PWM="1"なら正転、"0"ならブレーキ
0	PWM	1	PWM="1"なら正転、"0"ならフリー
1	PWM	0	PWM="1"なら逆転、"0"ならブレーキ
1	PWM	1	PWM="1"なら逆転、"0"ならフリー

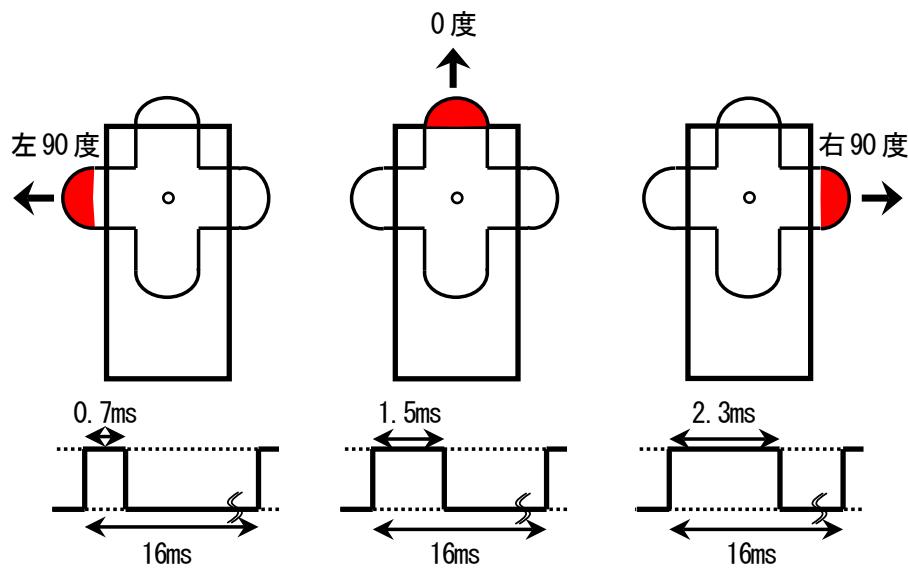
右モータを正転とフリーで動作させたい場合、P2_3="0"、P5_0="1"にして P2_4 から PWM 波形を出力すると、右モータが PWM の割合に応じて正転します。例えば、PWM が 0%ならモータの回転は停止、PWM が 50%ならモータの回転は正転 50%、PWM100%ならモータの回転は正転 100%になります。このときの停止は、フリー動作になります。

4.7 サーボ制御

4.7.1 原理

サーボは周期 16[ms]のパルスを加え、そのパルスの ON 幅でサーボの角度が決まります。

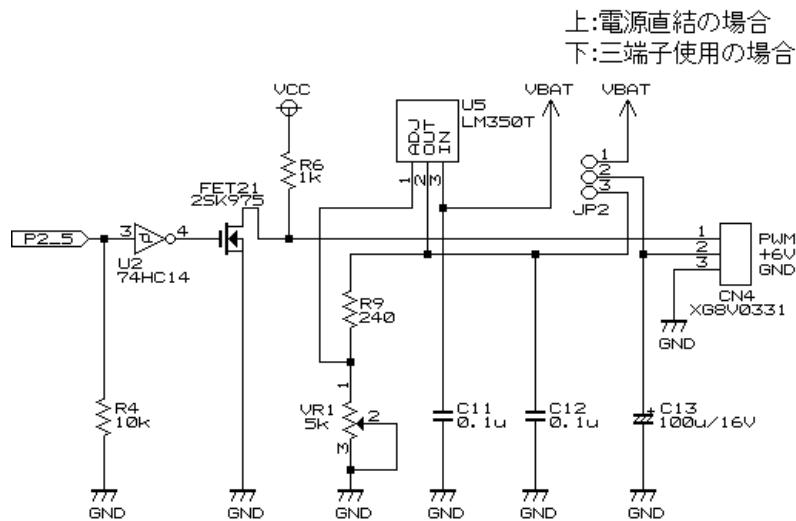
サーボの回転角度と ON のパルス幅の関係は、サーボのメーカや個体差によって多少の違いがありますが、ほとんどが下図のような関係になります。



- ・周期は 16[ms]
- ・中心は 1.5[ms]の ON パルス、 $\pm 0.8[ms]$ で ± 90 度のサーボ角度変化

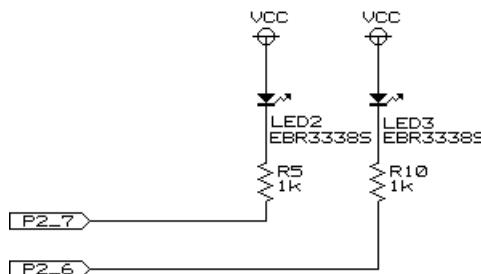
R8C/38A マイコンのリセット同期式 PWM モードで PWM 信号を生成して、サーボを制御します。

4.7.2 回路



- ポート 2 の bit5 から PWM 信号を出力します。
- ポートとサーボの1ピンの間にFETを入れてバッファとします。ポート2のbit5とサーボの1ピンが直結のとき、CN4 の 1 ピンに間違って + 電源を接続したりノイズが混入した場合、P2_5 端子を壊してしまう可能性があります。壊れてしまったらマイコンボードを交換しなければいけません。FET が壊れたなら、簡単に交換できます。
- CN4 の 2 ピンには、サーボ用の電源を供給します。モータ用電源が電池 4 本の場合、JP2 の 1-2 ピン間をショートして電源と直結します。それ以上の電圧の場合、サーボの定格を超えますので LM350 という 3A の電流を流せる三端子レギュレータを使って電圧を 6V 一定にします。JP2 は 2-3 ピン間をショートさせます。

4.8 LED制御



モータドライブ基板には 3 個の LED が付いています。そのうち、2 個がマイコンで ON/OFF 可能です。LED のカソードは、マイコンのポートに直結されています。電流制限抵抗は、1kΩです。EBR3338S は順電圧 1.7V、電流は 20mA 流すことができます。電流制限抵抗は下記のようになります。

$$\begin{aligned} \text{抵抗} &= (\text{電源電圧} - \text{LED に加える電圧}) / \text{LED に流したい電流} \\ &= (5 - 1.7) / 0.02 \\ &= 165\Omega \end{aligned}$$

実際は、電池の消費電流を減らすのとポートの電流制限により、1kΩの抵抗を接続しています。電流は、下記のようになります。

$$\begin{aligned} \text{電流} &= (\text{電源電圧} - \text{LED に加える電圧}) / \text{抵抗} \\ &= (5 - 1.7) / 1000 = 3.3[\text{mA}] \end{aligned}$$

<p>点灯！</p>	<p>消灯</p> <p>電流流れず！</p>
<p>P2_7 に "0" を出力すると、LED のカソード側が 0V になり、電流が流れ、LED は光ります。</p>	<p>P2_7 に "1" を出力すると、LED のカソード側が 5V になり、LED の両端の電位差は 0V となり、LED は光りません。</p>

4.9 スイッチ制御

モータドライブ基板には、プッシュスイッチが1個付いています。

	<p>"1"</p>	<p>"0"</p>
<p>スイッチは、10kΩでプルアップされ、ポート2のbit0に繋がっています。</p>	<p>スイッチが押されていない場合は、プルアップ抵抗を通して "1" が P2_0 に入力されます。</p>	<p>スイッチが押されると GND とつながり、"0" が P2_0 に入力されます。</p>

5. サンプルプログラム

5.1 プログラムの開発環境

プログラムの開発は、ルネサス統合開発環境を使います。ルネサス統合開発環境のインストール、使い方については、「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。

5.2 サンプルプログラムのインストール

5.2.1 ホームページからダウンロードする

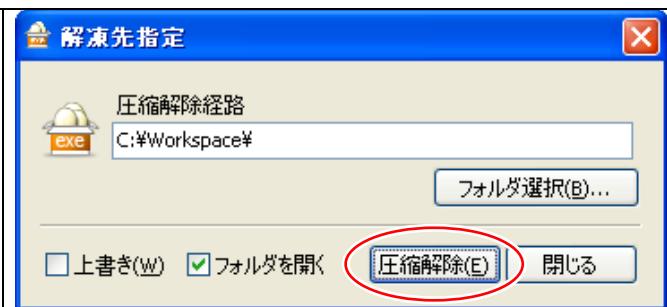
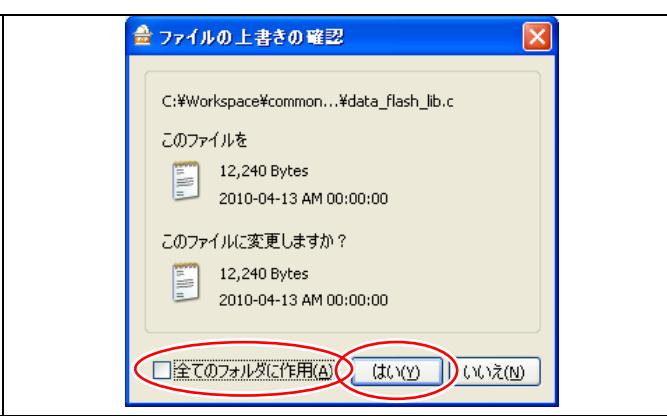
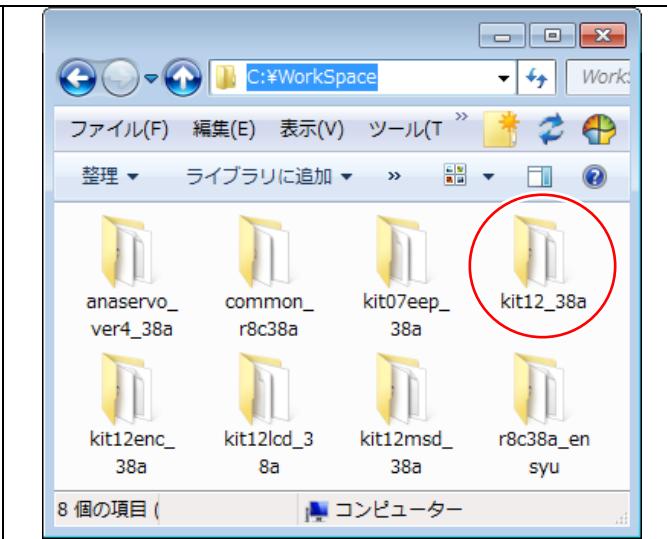
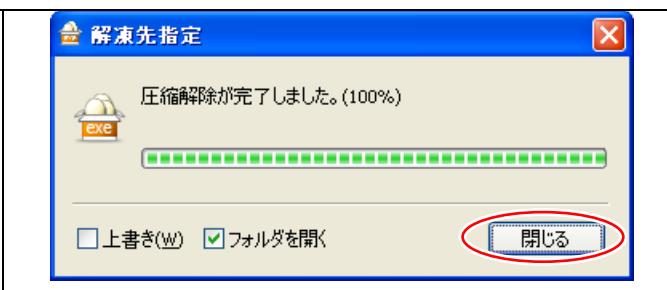
1		<p>マイコンカーラリーサイト 「http://www.mcr.gr.jp/」の技術情報→ダウンロード内のページへ行きます。</p>
---	--	--

2	<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なりますが、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をいたします。</p>	<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>												
	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center; padding: 5px;">対象</th> <th style="text-align: center; padding: 5px;">内容</th> <th style="text-align: center; padding: 5px;">更新日</th> </tr> </thead> <tbody> <tr> <td style="text-align: center; padding: 5px;">R8C/ 38A</td> <td style="text-align: center; padding: 5px;">R8C/38Aマイコン(RY_R8C38ボード)に関する資料</td> <td style="text-align: center; padding: 5px;">2014.06.17</td> </tr> <tr> <td style="text-align: center; padding: 5px;">H8/3048F -ONE</td> <td style="text-align: center; padding: 5px;">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 style="text-align: center; padding: 5px;">2010.10.07</td> </tr> <tr> <td style="text-align: center; padding: 5px;">H8/3048C</td> <td style="text-align: center; padding: 5px;">H8/3048Cマイコン(RY3048Cボード)用</td> <td></td> </tr> </tbody> </table>	対象	内容	更新日	R8C/ 38A	R8C/38Aマイコン(RY_R8C38ボード)に関する資料	2014.06.17	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	H8/3048C	H8/3048Cマイコン(RY3048Cボード)用		
対象	内容	更新日												
R8C/ 38A	R8C/38Aマイコン(RY_R8C38ボード)に関する資料	2014.06.17												
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												
H8/3048C	H8/3048Cマイコン(RY3048Cボード)用													

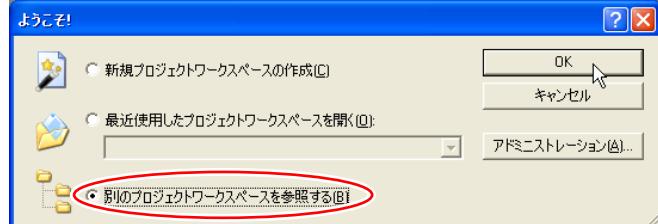
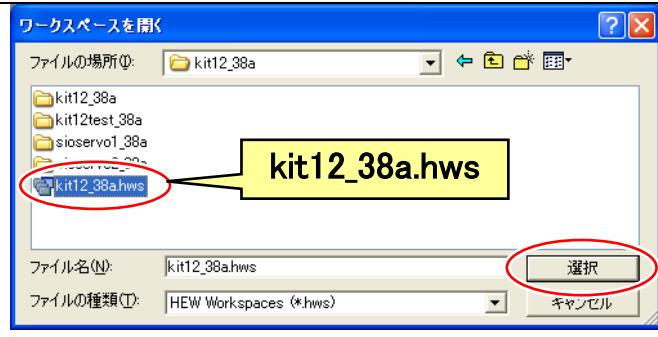
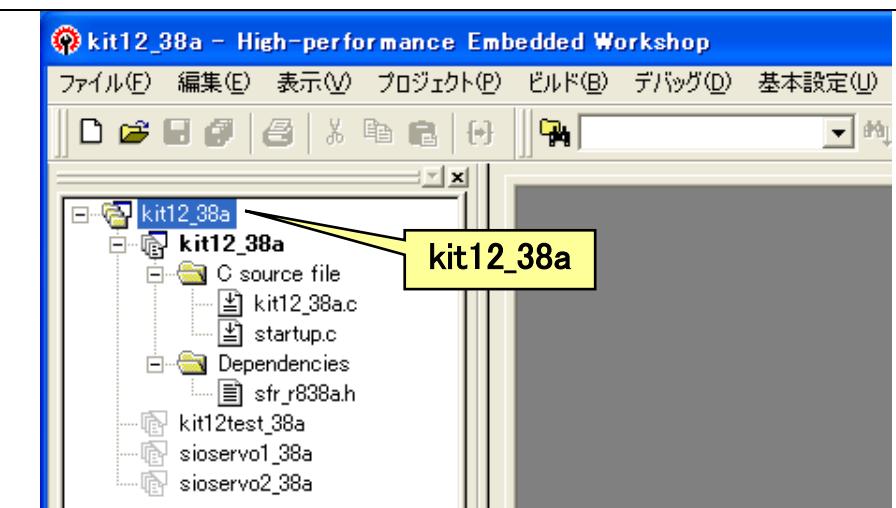
3	<p>■マイコンカーキットに関する資料</p> <p>マイコンカーキットVer.5.1に関する資料は、下記からダウンロードしてください。</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center; padding: 5px;">本体製作</th><th style="text-align: center; padding: 5px;">センサ基板</th><th style="text-align: center; padding: 5px;">モータドライバ基板</th><th style="text-align: center; padding: 5px;">動作確認</th><th style="text-align: center; padding: 5px;">プログラム解説</th><th style="text-align: center; padding: 5px;">プログラム</th></tr> </thead> <tbody> <tr> <td style="text-align: center; padding: 5px;">マイコンカーキットVer.5.1 本体組み立て製作マニュアル アル 第1.02版 2014.04.10 部品リストに写真を追加して見やすくしました!!</td><td style="text-align: center; padding: 5px;">センサ基板Ver.5製作マニュアル 第2.02版 2014.05.13</td><td style="text-align: center; padding: 5px;">モータドライバ基板Ver.5製作マニュアル 第1.04版 2014.05.13</td><td style="text-align: center; padding: 5px;">※「LM350追加セット」、「フリー追加セット」についても本マニュアルをご覧ください。</td><td style="text-align: center; padding: 5px;">マイコンカーキットVer.5.1動作確認マニュアル 第1.01版 2013.05.28</td><td style="text-align: center; padding: 5px;">kit12_38aプロ グラム解説 第1.11版 2014.06.02</td><td style="text-align: center; padding: 5px;">kit12_38a zip 2013.04.01</td></tr> <tr> <td colspan="6" style="text-align: center; padding: 10px;"> kit12_38a.zip </td><td></td></tr> </tbody> </table>	本体製作	センサ基板	モータドライバ基板	動作確認	プログラム解説	プログラム	マイコンカーキットVer.5.1 本体組み立て製作マニュアル アル 第1.02版 2014.04.10 部品リストに写真を追加して見やすくしました!!	センサ基板Ver.5製作マニュアル 第2.02版 2014.05.13	モータドライバ基板Ver.5製作マニュアル 第1.04版 2014.05.13	※「LM350追加セット」、「フリー追加セット」についても本マニュアルをご覧ください。	マイコンカーキットVer.5.1動作確認マニュアル 第1.01版 2013.05.28	kit12_38aプロ グラム解説 第1.11版 2014.06.02	kit12_38a zip 2013.04.01	kit12_38a.zip							<p>マイコンカーキット Ver.5.1 に関する資料の「kit12_38a.zip」をダウンロードします。 このファイルに圧縮されている「kit12_38a.exe」ファイルを、解凍ソフトで解凍します。</p>
本体製作	センサ基板	モータドライバ基板	動作確認	プログラム解説	プログラム																	
マイコンカーキットVer.5.1 本体組み立て製作マニュアル アル 第1.02版 2014.04.10 部品リストに写真を追加して見やすくしました!!	センサ基板Ver.5製作マニュアル 第2.02版 2014.05.13	モータドライバ基板Ver.5製作マニュアル 第1.04版 2014.05.13	※「LM350追加セット」、「フリー追加セット」についても本マニュアルをご覧ください。	マイコンカーキットVer.5.1動作確認マニュアル 第1.01版 2013.05.28	kit12_38aプロ グラム解説 第1.11版 2014.06.02	kit12_38a zip 2013.04.01																
kit12_38a.zip																						

5. サンプルプログラム

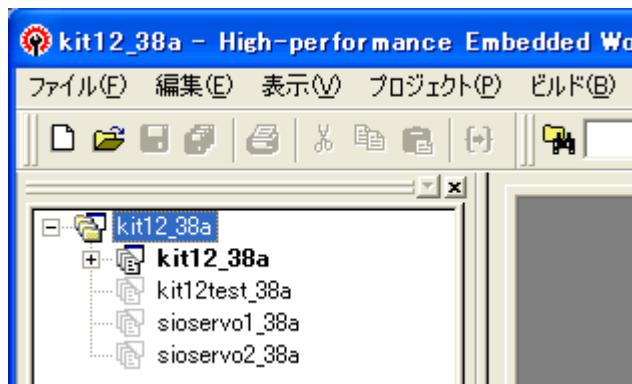
5.2.2 インストール

1		<p>ダウンロードしたファイルを実行します。</p> <p>「圧縮解除」をクリックします。</p> <p>※フォルダは替えないでください。替えた場合は、ルネサス統合開発環境のツールチエインの設定変更が必要です。</p>
2		<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>
3		<p>「C:\Workspace」のフォルダが自動で開きます。</p> <p>「kit12_38a」フォルダの内容が、今回使用するプログラムが入っているフォルダです。</p>
4		<p>「閉じる」をクリックします。</p>

5.3 ワーススペース「kit12_38a」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」を選択します。
3		C ドライブ → Workspace → kit12_38a の「kit12_38a.hws」を選択します。
4		ワークスペース「kit12_38a」が開かれます。

5.4 プロジェクト



ワークスペース「kit12_38a」には、4 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
1	kit12_38a	マイコンカー走行プログラムです。 次章からプログラムの解説をします。
2	kit12test_38a	製作したマイコンカーのモータドライブ基板やセンサ基板が正しく動作するかテストします。詳しくは、「マイコンカーキット Ver.5.1 動作テストマニュアル」を参照してください。
3	sioservo1_38a	サーボのセンタを調整するプログラムです。 詳しくは、「7. サーボセンタと最大切れ角の調整」を参照してください。
4	sioservo2_38a	サーボの最大切れ角を見つけるためのプログラムです。 詳しくは、「7. サーボセンタと最大切れ角の調整」を参照してください。

6. プログラム解説「kit12_38a.c」

6.1 プログラムリスト

R8C/38A マイコンでマイコンカーを制御するプログラムを、下記に示します。

```

1 : /***** *****/
2 : /* 対象マイコン R8C/38A */ *
3 : /* ファイル内容 マイコンカーキットVer.5.1 トレース基本プログラム(R8C/38A版) */
4 : /* バージョン Ver.1.01 */ *
5 : /* Date 2013.04.18 */ *
6 : /* Copyright ジャパンマイコンカーラリー実行委員会 */ *
7 : /***** *****/
8 : /*
9 : このプログラムは、下記基板に対応しています。
10:  * RV_R8C38ボード
11:  * モータドライブ基板Ver.5
12:  * センサ基板Ver.5
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:
29: /* マスク値設定 X : マスクあり(無効) ○ : マスク無し(有効) */
30: #define MASK2_2        0x66       /* X○○×○○× */
31: #define MASK2_0        0x60       /* X○○×××× */
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 handle( int angle );
55:
56: /*****
57: /* グローバル変数の宣言 */
58: /*****
59: unsigned long cnt0;           /* timer関数用 */
60: unsigned long cnt1;           /* main内で使用 */
61: int          pattern;         /* パターン番号 */
62:
63: /***** *****/
64: /* メインプログラム */
65: /***** *****/
66: void main( void )
67: {
68:     int i;
69:
70:     /* マイコン機能の初期化 */
71:     init();                  /* 初期化 */
72:     asm(" fset I ");        /* 全体の割り込み許可 */
73:
74:     /* マイコンカーの状態初期化 */
75:     handle( 0 );
76:     motor( 0, 0 );
77:

```

6. プログラム解説「kit12_38a.c」

```

78 :     while( 1 ) {
79 :         switch( pattern ) {
80 :             ****
81 :             // パターンについて
82 :             0 : スイッチ入力待ち
83 :             1 : スタートバーが開いたかチェック
84 :             11 : 通常トレース
85 :             12 : 右へ大曲げの終わりのチェック
86 :             13 : 左へ大曲げの終わりのチェック
87 :             21 : クロスライン検出時の処理
88 :             22 : クロスラインを読み飛ばす
89 :             23 : クロスライン後のトレース、クランク検出
90 :             31 : 左クランククリア処理 安定するまで少し待つ
91 :             32 : 左クランククリア処理 曲げ終わりのチェック
92 :             41 : 右クランククリア処理 安定するまで少し待つ
93 :             42 : 右クランククリア処理 曲げ終わりのチェック
94 :             51 : 右ハーフライン検出時の処理
95 :             52 : 右ハーフラインを読み飛ばす
96 :             53 : 右ハーフライン後のトレース、レーンチェンジ
97 :             54 : 右レーンチェンジ終了のチェック
98 :             61 : 左ハーフライン検出時の処理
99 :             62 : 左ハーフラインを読み飛ばす
100 :            63 : 左ハーフライン後のトレース、レーンチェンジ
101 :            64 : 左レーンチェンジ終了のチェック
102 : ****
103 : ****
104 :
105 :     case 0:
106 :         /* スイッチ入力待ち */
107 :         if( pushsw_get() ) {
108 :             pattern = 1;
109 :             cntl = 0;
110 :             break;
111 :         }
112 :         if( cntl < 100 ) { /* LED点滅処理 */          led_out( 0x1 );
113 :         } else if( cntl < 200 ) {
114 :             led_out( 0x2 );
115 :         } else {
116 :             cntl = 0;
117 :         }
118 :         break;
119 :
120 :
121 :     case 1:
122 :         /* スタートバーが開いたかチェック */
123 :         if( !startbar_get() ) {
124 :             /* スタート！！ */
125 :             led_out( 0x0 );
126 :             pattern = 11;
127 :             cntl = 0;
128 :             break;
129 :         }
130 :         if( cntl < 50 ) { /* LED点滅処理 */          led_out( 0x1 );
131 :         } else if( cntl < 100 ) {
132 :             led_out( 0x2 );
133 :         } else {
134 :             cntl = 0;
135 :         }
136 :         break;
137 :
138 :
139 :     case 11:
140 :         /* 通常トレース */
141 :         if( check_crossline() ) { /* クロスラインチェック */          pattern = 21;
142 :         } else {
143 :             break;
144 :         }
145 :         if( check_rightline() ) { /* 右ハーフラインチェック */          pattern = 51;
146 :         } else {
147 :             break;
148 :         }
149 :         if( check_leftline() ) { /* 左ハーフラインチェック */          pattern = 61;
150 :         } else {
151 :             break;
152 :         }
153 :         switch( sensor_inp(MASK3_3) ) {
154 :             case 0x00:
155 :                 /* センター→まっすぐ */
156 :                 handle( 0 );
157 :                 motor( 100, 100 );
158 :                 break;
159 :
160 :             case 0x04:
161 :                 /* 微妙に左寄り→右へ微曲げ */
162 :                 handle( 5 );
163 :                 motor( 100, 100 );
164 :                 break;
165 :         }

```

```

166 :         case 0x06:
167 :             /* 少し左寄り→右へ小曲げ */
168 :             handle( 10 );
169 :             motor( 80 , 67 );
170 :             break;
171 :
172 :         case 0x07:
173 :             /* 中くらい左寄り→右へ中曲げ */
174 :             handle( 15 );
175 :             motor( 50 , 38 );
176 :             break;
177 :
178 :         case 0x03:
179 :             /* 大きく左寄り→右へ大曲げ */
180 :             handle( 25 );
181 :             motor( 30 , 19 );
182 :             pattern = 12;
183 :             break;
184 :
185 :         case 0x20:
186 :             /* 微妙に右寄り→左へ微曲げ */
187 :             handle( -5 );
188 :             motor( 100 , 100 );
189 :             break;
190 :
191 :         case 0x60:
192 :             /* 少し右寄り→左へ小曲げ */
193 :             handle( -10 );
194 :             motor( 67 , 80 );
195 :             break;
196 :
197 :         case 0xe0:
198 :             /* 中くらい右寄り→左へ中曲げ */
199 :             handle( -15 );
200 :             motor( 38 , 50 );
201 :             break;
202 :
203 :         case 0xc0:
204 :             /* 大きく右寄り→左へ大曲げ */
205 :             handle( -25 );
206 :             motor( 19 , 30 );
207 :             pattern = 13;
208 :             break;
209 :
210 :     default:
211 :         break;
212 :     }
213 :     break;
214 :
215 : case 12:
216 :     /* 右へ大曲げの終わりのチェック */
217 :     if( check_crossline() ) { /* 大曲げ中もクロスラインチェック */
218 :         pattern = 21;
219 :         break;
220 :     }
221 :     if( check_rightline() ) { /* 右ハーフラインチェック */
222 :         pattern = 51;
223 :         break;
224 :     }
225 :     if( check_leftline() ) { /* 左ハーフラインチェック */
226 :         pattern = 61;
227 :         break;
228 :     }
229 :     if( sensor_inp(MASK3_3) == 0x06 ) {
230 :         pattern = 11;
231 :     }
232 :     break;
233 :
234 : case 13:
235 :     /* 左へ大曲げの終わりのチェック */
236 :     if( check_crossline() ) { /* 大曲げ中もクロスラインチェック */
237 :         pattern = 21;
238 :         break;
239 :     }
240 :     if( check_rightline() ) { /* 右ハーフラインチェック */
241 :         pattern = 51;
242 :         break;
243 :     }
244 :     if( check_leftline() ) { /* 左ハーフラインチェック */
245 :         pattern = 61;
246 :         break;
247 :     }
248 :     if( sensor_inp(MASK3_3) == 0x60 ) {
249 :         pattern = 11;
250 :     }
251 :     break;
252 :

```

6. プログラム解説「kit12_38a.c」

```

253 :     case 21:
254 :         /* クロスライン検出時の処理 */
255 :         led_out( 0x3 );
256 :         handle( 0 );
257 :         motor( 0 ,0 );
258 :         pattern = 22;
259 :         cntl = 0;
260 :         break;
261 :
262 :     case 22:
263 :         /* クロスラインを読み飛ばす */
264 :         if( cntl > 100 ) {
265 :             pattern = 23;
266 :             cntl = 0;
267 :         }
268 :         break;
269 :
270 :     case 23:
271 :         /* クロスライン後のトレース、クランク検出 */
272 :         if( sensor_inp(MASK4_4)==0xf8 ) {
273 :             /* 左クランクと判断→左クランククリア処理へ */
274 :             led_out( 0x1 );
275 :             handle( -38 );
276 :             motor( 10 ,50 );
277 :             pattern = 31;
278 :             cntl = 0;
279 :             break;
280 :         }
281 :         if( sensor_inp(MASK4_4)==0x1f ) {
282 :             /* 右クランクと判断→右クランククリア処理へ */
283 :             led_out( 0x2 );
284 :             handle( 38 );
285 :             motor( 50 ,10 );
286 :             pattern = 41;
287 :             cntl = 0;
288 :             break;
289 :         }
290 :         switch( sensor_inp(MASK3_3) ) {
291 :             case 0x00:
292 :                 /* センタ→まっすぐ */
293 :                 handle( 0 );
294 :                 motor( 40 ,40 );
295 :                 break;
296 :             case 0x04:
297 :             case 0x06:
298 :             case 0x07:
299 :             case 0x03:
300 :                 /* 左寄り→右曲げ */
301 :                 handle( 8 );
302 :                 motor( 40 ,35 );
303 :                 break;
304 :             case 0x20:
305 :             case 0x60:
306 :             case 0xe0:
307 :             case 0xc0:
308 :                 /* 右寄り→左曲げ */
309 :                 handle( -8 );
310 :                 motor( 35 ,40 );
311 :                 break;
312 :             }
313 :             break;
314 :
315 :         case 31:
316 :             /* 左クランククリア処理 安定するまで少し待つ */
317 :             if( cntl > 200 ) {
318 :                 pattern = 32;
319 :                 cntl = 0;
320 :             }
321 :             break;
322 :
323 :         case 32:
324 :             /* 左クランククリア処理 曲げ終わりのチェック */
325 :             if( sensor_inp(MASK3_3) == 0x60 ) {
326 :                 led_out( 0x0 );
327 :                 pattern = 11;
328 :                 cntl = 0;
329 :             }
330 :             break;
331 :
332 :         case 41:
333 :             /* 右クランククリア処理 安定するまで少し待つ */
334 :             if( cntl > 200 ) {
335 :                 pattern = 42;
336 :                 cntl = 0;
337 :             }
338 :             break;
339 :

```

```

340 :     case 42:
341 :         /* 右クランククリア処理 曲げ終わりのチェック */
342 :         if( sensor_inp(MASK3_3) == 0x06 ) {
343 :             led_out( 0x0 );
344 :             pattern = 11;
345 :             cnt1 = 0;
346 :         }
347 :         break;
348 :
349 :     case 51:
350 :         /* 右ハーフライン検出時の処理 */
351 :         led_out( 0x2 );
352 :         handle( 0 );
353 :         motor( 0 , 0 );
354 :         pattern = 52;
355 :         cnt1 = 0;
356 :         break;
357 :
358 :     case 52:
359 :         /* 右ハーフラインを読み飛ばす */
360 :         if( cnt1 > 100 ) {
361 :             pattern = 53;
362 :             cnt1 = 0;
363 :         }
364 :         break;
365 :
366 :     case 53:
367 :         /* 右ハーフライン後のトレース、レーンチェンジ */
368 :         if( sensor_inp(MASK4_4) == 0x00 ) {
369 :             handle( 15 );
370 :             motor( 40 , 31 );
371 :             pattern = 54;
372 :             cnt1 = 0;
373 :             break;
374 :         }
375 :         switch( sensor_inp(MASK3_3) ) {
376 :             case 0x00:
377 :                 /* センタ→まっすぐ */
378 :                 handle( 0 );
379 :                 motor( 40 , 40 );
380 :                 break;
381 :             case 0x04:
382 :             case 0x06:
383 :             case 0x07:
384 :             case 0x03:
385 :                 /* 左寄り→右曲げ */
386 :                 handle( 8 );
387 :                 motor( 40 , 35 );
388 :                 break;
389 :             case 0x20:
390 :             case 0x60:
391 :             case 0xe0:
392 :             case 0xc0:
393 :                 /* 右寄り→左曲げ */
394 :                 handle( -8 );
395 :                 motor( 35 , 40 );
396 :                 break;
397 :             default:
398 :                 break;
399 :         }
400 :         break;
401 :
402 :     case 54:
403 :         /* 右レーンチェンジ終了のチェック */
404 :         if( sensor_inp( MASK4_4 ) == 0x3c ) {
405 :             led_out( 0x0 );
406 :             pattern = 11;
407 :             cnt1 = 0;
408 :         }
409 :         break;
410 :
411 :     case 61:
412 :         /* 左ハーフライン検出時の処理 */
413 :         led_out( 0x1 );
414 :         handle( 0 );
415 :         motor( 0 , 0 );
416 :         pattern = 62;
417 :         cnt1 = 0;
418 :         break;
419 :
420 :     case 62:
421 :         /* 左ハーフラインを読み飛ばす */
422 :         if( cnt1 > 100 ) {
423 :             pattern = 63;
424 :             cnt1 = 0;
425 :         }
426 :         break;
427 :

```

6. プログラム解説「kit12_38a.c」

```

428 :     case 63:
429 :         /* 左ハーフライン後のトレース、レーンチェンジ */
430 :         if( sensor_inp(MASK4_4) == 0x00 ) {
431 :             handle( -15 );
432 :             motor( 31 , 40 );
433 :             pattern = 64;
434 :             cntl = 0;
435 :             break;
436 :         }
437 :         switch( sensor_inp(MASK3_3) ) {
438 :             case 0x00:
439 :                 /* センタ→まっすぐ */
440 :                 handle( 0 );
441 :                 motor( 40 , 40 );
442 :                 break;
443 :             case 0x04:
444 :             case 0x06:
445 :             case 0x07:
446 :             case 0x03:
447 :                 /* 左寄り→右曲げ */
448 :                 handle( 8 );
449 :                 motor( 40 , 35 );
450 :                 break;
451 :             case 0x20:
452 :             case 0x60:
453 :             case 0xe0:
454 :             case 0xc0:
455 :                 /* 右寄り→左曲げ */
456 :                 handle( -8 );
457 :                 motor( 35 , 40 );
458 :                 break;
459 :             default:
460 :                 break;
461 :         }
462 :         break;
463 :
464 :     case 64:
465 :         /* 左レーンチェンジ終了のチェック */
466 :         if( sensor_inp( MASK4_4 ) == 0x3c ) {
467 :             led_out( 0x0 );
468 :             pattern = 11;
469 :             cntl = 0;
470 :         }
471 :         break;
472 :
473 :     default:
474 :         /* どれでもない場合は待機状態に戻す */
475 :         pattern = 0;
476 :         break;
477 :     }
478 : }
479 : }
480 :
481 : /*****
482 :  * R8C/38A スペシャルファンクションレジスタ(SFR)の初期化
483 : *****/
484 : void init( void )
485 : {
486 :     int i;
487 :
488 :     /* クロックをXINクロック(20MHz)に変更 */
489 :     prc0 = 1;                                /* プロテクト解除 */
490 :     cm13 = 1;                                /* P4_6,P4_7をXIN-XOUT端子にする */
491 :     cm05 = 0;                                /* XINクロック発振 */
492 :     for(i=0; i<50; i++ );                   /* 安定するまで少し待つ(約10ms) */
493 :     ocd2 = 0;                                /* システムクロックをXINにする */
494 :     prc0 = 0;                                /* プロテクトON */
495 :
496 :     /* ポートの入出力設定 */
497 :     prc2 = 1;                                /* PD0のプロテクト解除 */
498 :     pd0 = 0x00;                               /* 7-0:センサ基板Ver.5 */
499 :     pd1 = 0xd0;                               /* 5:RXD0 4:TXD0 3-0:DIP SW */
500 :     p2 = 0xc0;                               /* 7-0:モータドライブ基板Ver.5 */
501 :     pd2 = 0xfe;                               /* */
502 :     pd3 = 0xff;                               /* */
503 :     p4 = 0x20;                               /* P4_5のLED:初期は点灯 */
504 :     pd4 = 0xb8;                               /* 7:XOUT 6:XIN 5:LED 2:VREF */
505 :     pd5 = 0xff;                               /* */
506 :     pd6 = 0xff;                               /* */
507 :     pd7 = 0xff;                               /* */
508 :     pd8 = 0xff;                               /* */
509 :     pd9 = 0x3f;                               /* */
510 :     pur0 = 0x04;                             /* P1_3~P1_0のプルアップON */
511 :

```

```

512 : /* タイマRBの設定 */
513 : /* 割り込み周期 = 1 / 20[MHz] * (TRBPRE+1) * (TRBPR+1)
514 :           = 1 / (20*106) * 200          * 100
515 :           = 0.001[s] = 1[ms]
516 : */
517 : trbmr  = 0x00;           /* 動作モード、分周比設定 */
518 : trbpre = 200-1;         /* ブリッスケーラレジスタ */
519 : trbpr  = 100-1;         /* プライマリレジスタ */
520 : trbic  = 0x07;         /* 割り込み優先レベル設定 */
521 : trbcr  = 0x01;         /* カウント開始 */
522 :
523 : /* タイマRD リセット同期PWMモードの設定*/
524 : /* PWM周期 = 1 / 20[MHz] * カウントソース * (TRDGRA0+1)
525 :           = 1 / (20*106) * 8          * 40000
526 :           = 0.016[s] = 16[ms]
527 : */
528 : trdpsr0 = 0x08;        /* TRDIOB0, C0, D0端子設定 */
529 : trdpsr1 = 0x05;        /* TRDIOA1, B1, C1, D1端子設定 */
530 : trdmr  = 0xf0;         /* バッファレジスタ設定 */
531 : trdfcr = 0x01;         /* リセット同期PWMモードに設定 */
532 : trdcr0 = 0x23;         /* ソースカウントの選択:f8 */
533 : trdgra0 = trdgrc0 = PWM_CYCLE; /* 周期 */
534 : trdgrb0 = trdgrd0 = 0;   /* P2_2端子のON幅設定 */
535 : trdgrb1 = trdgrc1 = 0;   /* P2_4端子のON幅設定 */
536 : trdgrb1 = trdgrd1 = SERVO_CENTER; /* P2_5端子のON幅設定 */
537 : trdoer1 = 0xcd;        /* 出力端子の選択 */
538 : trdstr = 0x0d;         /* TRDOカウント開始 */
539 : }
540 :
541 : /****** */
542 : /* タイマRB 割り込み処理 */
543 : /****** */
544 : #pragma interrupt intTRB(vect=24)
545 : void intTRB( void )
546 : {
547 :     cnt0++;
548 :     cnt1++;
549 : }
550 :
551 : /****** */
552 : /* タイマ本体 */
553 : /* 引数 タイマ値 1=lms */
554 : /****** */
555 : void timer( unsigned long timer_set )
556 : {
557 :     cnt0 = 0;
558 :     while( cnt0 < timer_set );
559 : }
560 :
561 : /****** */
562 : /* センサ状態検出 */
563 : /* 引数 マスク値 */
564 : /* 戻り値 センサ値 */
565 : /****** */
566 : unsigned char sensor_inp( unsigned char mask )
567 : {
568 :     unsigned char sensor;
569 :
570 :     sensor = ~p0;
571 :
572 :     sensor &= mask;
573 :
574 :     return sensor;
575 : }
576 :
577 : /****** */
578 : /* クロスライン検出処理 */
579 : /* 戻り値 0:クロスラインなし 1:あり */
580 : /****** */
581 : int check_crossline( void )
582 : {
583 :     unsigned char b;
584 :     int ret;
585 :
586 :     ret = 0;
587 :     b = sensor_inp(MASK3_3);
588 :     if( b==0xe7 ) {
589 :         ret = 1;
590 :     }
591 :     return ret;
592 : }
593 :

```

6. プログラム解説「kit12_38a.c」

```

594 : /*****
595 : /* 右ハーフライン検出処理 */
596 : /* 戻り値 0:なし 1:あり */
597 : *****/
598 : int check_rightline( void )
599 : {
600 :     unsigned char b;
601 :     int ret;
602 :
603 :     ret = 0;
604 :     b = sensor_inp(MASK4_4);
605 :     if( b==0x1f ) {
606 :         ret = 1;
607 :     }
608 :     return ret;
609 : }
610 :
611 : /*****
612 : /* 左ハーフライン検出処理 */
613 : /* 戻り値 0:なし 1:あり */
614 : *****/
615 : int check_leftline( void )
616 : {
617 :     unsigned char b;
618 :     int ret;
619 :
620 :     ret = 0;
621 :     b = sensor_inp(MASK4_4);
622 :     if( b==0xf8 ) {
623 :         ret = 1;
624 :     }
625 :     return ret;
626 : }
627 :
628 : /*****
629 : /* ディップスイッチ値読み込み */
630 : /* 戻り値 スイッチ値 0~15 */
631 : *****/
632 : unsigned char dipsw_get( void )
633 : {
634 :     unsigned char sw;
635 :     sw = p1 & 0x0f;           /* P1_3~P1_0読み込み */
636 :     return sw;
637 : }
638 :
639 : /*****
640 : /* ブッシュスイッチ値読み込み */
641 : /* 戻り値 スイッチ値 ON:1 OFF:0 */
642 : *****/
643 : unsigned char pushsw_get( void )
644 : {
645 :     unsigned char sw;
646 :
647 :     sw = ~p2;                /* スイッチのあるポート読み込み */
648 :     sw &= 0x01;
649 :     return sw;
650 : }
651 :
652 : /*****
653 : /* スタートバー検出センサ読み込み */
654 : /* 戻り値 センサ値 ON(バーあり):1 OFF(なし):0 */
655 : *****/
656 : unsigned char startbar_get( void )
657 : {
658 :     unsigned char b;
659 :
660 :     b = ~p0;                 /* スタートバー信号読み込み */
661 :     b &= 0x01;
662 :
663 :     return b;
664 : }
665 :
666 : /*****
667 : /* LED制御 */
668 : /* 引数 スイッチ値 LED2:bit1 LED3:bit0 "0":消灯 "1":点灯 */
669 : /* 例)0x3→LED2:ON LED3:ON 0x2→LED2:ON LED3:OFF */
670 : *****/
671 : void led_out( unsigned char led )
672 : {
673 :     unsigned char data;
674 :
675 :     led = ~led;
676 :     led <= 6;
677 :     data = p2 & 0x3f;
678 :     p2 = data | led;
679 : }
680 :
681 : /*****
682 : *****/
683 :

```

```

684 : *****/
685 : /* モータ速度制御 */
686 : /* 引数 左モータ:-100～100、右モータ:-100～100 */
687 : /* 0で停止、100で正転100%、-100で逆転100% */
688 : /* 戻り値 なし */
689 : *****/
690 : void motor( int accele_l, int accele_r )
691 : {
692 :     int     sw_data;
693 :
694 :     sw_data = dipsw_get() + 5;
695 :     accele_l = accele_l * sw_data / 20;
696 :     accele_r = accele_r * sw_data / 20;
697 :
698 :     /* 左モータ制御 */
699 :     if( accele_l >= 0 ) {
700 :         p2 &= 0xfd;
701 :         trdgrd0 = (long)( PWM_CYCLE - 1 ) * accele_l / 100;
702 :     } else {
703 :         p2 |= 0x02;
704 :         trdgrd0 = (long)( PWM_CYCLE - 1 ) * ( -accele_l ) / 100;
705 :     }
706 :
707 :     /* 右モータ制御 */
708 :     if( accele_r >= 0 ) {
709 :         p2 &= 0xf7;
710 :         trdgrc1 = (long)( PWM_CYCLE - 1 ) * accele_r / 100;
711 :     } else {
712 :         p2 |= 0x08;
713 :         trdgrc1 = (long)( PWM_CYCLE - 1 ) * ( -accele_r ) / 100;
714 :     }
715 : }
716 :
717 : *****/
718 : /* サーボハンドル操作 */
719 : /* 引数 サーボ操作角度 : -90～90 */
720 : /* -90で左～90度、0でまっすぐ、90で右～90度回転 */
721 : *****/
722 : void handle( int angle )
723 : {
724 :     /* サーボが左右逆に動く場合は、「-」を「+」に替えてください */
725 :     trdgrd1 = SERVO_CENTER - angle * HANDLE_STEP;
726 : }
727 :
728 : *****/
729 : /* end of file */
730 : *****/
731 :
732 : /*
733 : 改訂内容
734 :
735 : Ver. 1.00 2012.05.07 作成(センサ基板Ver.4→Ver.5用に変更)
736 : Ver. 1.01 2013.04.18 クランク、レーンチェンジ手前の横線が2本から1本になった
737 : ことによる、コメントの修正(プログラムの変更はなし)
738 :
739 : */

```

6.2 「kit07_38a.c」と「kit12_38a.c」プログラムの相違点

「kit07_38a.c」と「kit12_38a.c」プログラムの相違点を下表に示します。

関数	kit07_38a.c(センサ基板 Ver.4 を使用するとき)	kit12_38a.c(センサ基板 Ver.5 を使用するとき)
sensor_inp 関数	<pre>unsigned char sensor_inp(unsigned char mask) { unsigned char sensor; sensor = ~p0; sensor &= 0xef; if(sensor & 0x08) sensor = 0x10; sensor &= mask; return sensor; }</pre>	<pre>unsigned char sensor_inp(unsigned char mask) { unsigned char sensor; sensor = ~p0; // この行は削除 // この行は削除 sensor &= mask; return sensor; }</pre>
startbar_get 関数	<pre>unsigned char startbar_get(void) { unsigned char b; b = ~p0; b &= 0x10; b >>= 4; return b; }</pre>	<pre>unsigned char startbar_get(void) { unsigned char b; b = ~p0; b &= 0x01; // 0x10→0x01 に変更 // この行は削除 return b; }</pre>

6.3 R8C/38A マイコンで使用する内蔵周辺機能

RY_R8C38 ボード(R8C/38A マイコン)でマイコンカーキット Ver.5 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※(パルスカウント)	タイマ RG	ITU2
赤外線の受光制御※	タイマ RA	

※は、標準のマイコンカーキット(本マニュアル)では扱いません。

6.4 プログラムの解説

6.4.1 スタート

```

1 : /*****
2 : /* 対象マイコン R8C/38A
3 : /* ファイル内容 マイコンカーキットVer. 5.1 トレース基本プログラム(R8C/38A版) */
4 : /* バージョン Ver. 1.01
5 : /* Date 2013. 04. 18
6 : /* Copyright ジャパンマイコンカーラリー実行委員会
7 : /*****
8 : /*
9 : このプログラムは、下記基板に対応しています。
10: ・RY_R8C38ボード
11: ・モータドライブ基板Ver. 5
12: ・センサ基板Ver. 5
13: */

```

最初はコメント部分です。「/*」がコメントの開始、「*/」がコメントの終了です。コメント開始から終了までの間の文字は無視されるので、メモ書きとして利用します。

6.4.2 外部ファイルの取り込み(インクルード)

```

15 : /*****
16 : /* インクルード */
17 : /*****
18 : #include "sfr_r838a.h"          /* R8C/38A SFR の定義ファイル */

```

「#include」がインクルード命令です。インクルード命令は、外部からファイルを呼び出す命令です。。

ファイル名	内容
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 Registers)を定義したファイルです。ちなみにこのファイルは、次のフォルダにあります。 C:\Workspace\common_r8c38a

6.4.3 シンボル定義

```

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 :
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       /* ○○○○○○○○ */

```

<p>PWM_CYCLE</p> <p>右モータ、左モータ、およびサーボに加える PWM 周期を「PWM_CYCLE」という名前で定義します。今回、PWM 周期は 16ms にします。 PWM_CYCLE に設定する値は下記のようになります。</p> $\begin{aligned} \text{PWM_CYCLE} &= \text{設定したい周期} / \text{タイマ RD 制御レジスタ 0(TRDCR0)のカウントソースー1} \\ &= 16\text{ms} / 400\text{ns} - 1 \\ &= (16 \times 10^{-3}) / (400 \times 10^{-9}) - 1 \\ &= 40000 - 1 = \textbf{39999} \end{aligned}$ <p>詳しい説明は、リセット同期 PWM モード部分を参照してください。</p>	<p>SERVO_CENTER</p> <p>サーボが 0 度(まっすぐ向く角度)のときの値を「SERVO_CENTER」という名前で定義します。標準的なサーボは、1.5[ms]のパルス幅を加えるとまっすぐ向きます。よって、パルスの ON 幅を 1.5ms に設定します。SERVO_CENTER に設定する値は下記のようになります。</p> $\begin{aligned} \text{SERVO_CENTER} &= \text{パルスの ON 幅} / \text{タイマ RD 制御レジスタ 0(TRDCR0)カウントソース選択ビットの設定ー1} \\ &= 1.5\text{ms} / 400\text{ns} - 1 \\ &= (1.5 \times 10^{-3}) / (400 \times 10^{-9}) - 1 \\ &= 3750 - 1 = 3749 \end{aligned}$ <p>しかし、サーボのセンタは サーボ自体の誤差、サーボホーンに挿すギザギザのかみ合わせ方などの影響ですべてのマイコンカーで違う値になります。例えるなら、人間の指紋のようなものです。そのため、この値はプログラムでハンドルを 0 度にしたときに、マイコンカーがまっすぐ走るようにマイコンカー 1 台ごとに調整、変更します。</p>
--	--



▲ サーボホーン

	<p>サーボが 1 度分動く値を「HANDLE_STEP」という名前で定義します。 左 90 度の PWM の ON 幅は、0.7ms です。右 90 度の PWM の ON 幅は、2.3ms です。この差分を 180 で割ると、1 度当たりの値が計算できます。</p> <ul style="list-style-type: none"> • 左 90 度の PWM の ON 幅 $\begin{aligned} \text{TRDGRB1} &= \text{PWM 波形の ON 幅} / \text{タイマ RD カウンタ 0 のカウントソース - 1} \\ &= (0.7 \times 10^{-3}) / (400 \times 10^{-9}) - 1 \\ &= 1750 - 1 = 1749 \end{aligned}$ • 右 90 度の PWM の ON 幅 $\begin{aligned} \text{TRDGRB1} &= \text{PWM 波形の ON 幅} / \text{タイマ RD カウンタ 0 のカウントソース - 1} \\ &= (2.3 \times 10^{-3}) / (400 \times 10^{-9}) - 1 \\ &= 5750 - 1 = 5749 \end{aligned}$ • 1 度当たりの値 $(右 - 左) / 180 = (5749 - 1749) / 180 = 22.22 \approx 22$ <p>よって、HANDLE_STEP を 22 として定義します。1 度当たりの値を変更する場合は、この値を変更してください。</p>
MASK2_2 MASK2_0 MASK0_2 MASK3_3 MASK0_3 MASK3_0 MASK4_0 MASK0_4 MASK4_4	<p>sensor_inp 関数でセンサの値をマスクするとき、よく使うマスク値を定義しています。 「MASK」+「A」+「_（アンダーバー）」+「B」として定義しています。意味は下記のようになります。</p> <ul style="list-style-type: none"> • A…左のセンサ 4 個中 A 個を有効にする • B…右のセンサ 4 個中 B 個を有効にする • その他をマスクする <p>詳しくは、「6.4.13 sensor_inp 関数（センサ状態の読み込み）」を参照してください。</p>

6.4.4 プロトタイプ宣言

```

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 handle( int angle );

```

プロトタイプ宣言とは、自作した関数の引数の型と個数をチェックするために、関数を使用する前に宣言することです。関数プロトタイプは、関数に「;」を付加したものです。

プログラム例を下記に示します。

```

void motor( int accele_l, int accele_r );           /* プロトタイプ宣言 */

void main( void )
{
    int a, b;

    a = 50;
    b = 100;

    motor( a, b ); ←プロトタイプ宣言をしたので、1つ目の引数がint型か、2つ目がint型か
                    チェックする。もし、引数がint型でなければコンパイルエラーになる
}

/* モータ制御関数 */
void motor( int accele_l, int accele_r )
{
    プログラム
}

```

6.4.5 グローバル変数の宣言

```

56 : /*=====
57 : /* グローバル変数の宣言 */
58 : =====*/
59 : unsigned long cnt0;           /* timer関数用 */
60 : unsigned long cnt1;           /* main内で使用 */
61 : int pattern;                /* パターン番号 */

```

グローバル変数とは、関数の外で定義されどその関数からも参照できる変数のことです。ちなみに、関数内で宣言されている通常の変数は、ローカル変数といって、その関数の中のみで参照できる変数のことです。

プログラム例を下記に示します。

```

void a( void );                      /* プロトタイプ宣言 */

int timer;                           /* グローバル変数 */

void main( void )
{
    int i;

    timer = 0;
    i = 10;
    printf( "%d\n" , timer );          ←0 を表示
    a();
    printf( "%d\n" , timer );          ←timer はグローバル変数なので、
                                       a 関数内でセットした 20 を表示
    printf( "%d\n" , i );              ←a 関数でも変数 i を使っているがローカル
                                       変数なので、a 関数内の i 変数は無関係
                                       この関数でセットした 10 が表示される
}

void a( void )
{
    int i;
    i = 20;
    timer = i;
}

```

kit12_38a.c プログラムでは、3 つのグローバル変数を宣言しています。

変数名	型	使用方法
cnt0	unsigned long	この変数は、1ms ごとに+1 する変数です。timer 関数で 1ms を数えるのに使用します。timer 関数部分で詳しく説明します。
cnt1	unsigned long	この変数は、1ms ごとに+1 する変数です。この変数は、プログラム作成者が自由に使って、時間を計ります。例えば、300ms たつたら○○をしなさい、たっていないなら□□をしなさい、というように使用します。main 関数部分で詳しく説明します。
pattern	int	パターン番号です。main 関数部分で詳しく説明します。

ANSI C 規格(C言語の規格)で未初期化データは初期値が0x00でなければいけないと決まっています。そのため、これらの変数は 0 になっています。

6.4.6 init関数(クロックの切り替え)

init 関数は、R8C/38A マイコンの内蔵周辺機能を初期化する関数です。「init」とは、「initialize(イニシャライズ)」の略で、初期化の意味です。

init 関数では、何種類かの内蔵周辺機能の初期化をしています。それぞれの機能に分けて、説明します。

まず、動作クロックを変更します。R8C/38A は立ち上がったとき、低速オンチップオシレータ(内蔵クリスタル)で動作しています。このクロックは約 125kHz と遅いので、下記の 6 行のプログラムを実行して XIN クロック入力(外付けクリスタル)に切り替えます。RY_R8C38 ボードの XIN クロック入力は、X1 の 20MHz のクリスタルになります。

```

488 :     /* クロックをXINクロック(20MHz)に変更 */
489 :     prc0 = 1;                      /* プロテクト解除 */
490 :     cm13 = 1;                     /* P4_6,P4_7をXIN-XOUT端子にする */
491 :     cm05 = 0;                     /* XINクロック発振 */
492 :     for(i=0; i<50; i++);        /* 安定するまで少し待つ(約10ms) */
493 :     ocd2 = 0;                     /* システムクロックをXINにする */
494 :     prc0 = 0;                     /* プロテクトON */

```

プログラムの詳細については、「マイコン実習マニュアル(R8C/38A 版)」の「プロジェクト:io」を参照してください。

6.4.7 init関数(ポートの入出力設定)

次に、ポートの入出力設定を行います。

ポートの入出力設定の詳細については、「マイコン実習マニュアル(R8C/38A 版)」の「プロジェクト:io」を参照してください。

```

496 :     /* ポートの入出力設定 */
497 :     prc2 = 1;                      /* PD0のプロテクト解除 */
498 :     pd0 = 0x00;                   /* 7-0:センサ基板Ver.5 */
499 :     pd1 = 0xd0;                   /* 5:RXD0 4:TXD0 3-0:DIP SW */
500 :     p2 = 0xc0;
501 :     pd2 = 0xfe;                  /* 7-0:モータドライブ基板Ver.5 */
502 :     pd3 = 0xff;
503 :     p4 = 0x20;                   /* P4_5のLED:初期は点灯 */
504 :     pd4 = 0xb8;                   /* 7:XOUT 6:XIN 5:LED 2:VREF */
505 :     pd5 = 0xff;
506 :     pd6 = 0xff;
507 :     pd7 = 0xff;
508 :     pd8 = 0xff;
509 :     pd9 = 0x3f;
510 :     pur0 = 0x04;                 /* P1_3～P1_0のプルアップON */

```

R8C/38A にはポート 0 からポート 9 まで 10 個のポートがあります。マイコンカーキットの接続状態を下表に示します。

ポート	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
0	センサ基板 コース状態 入力	センサ基板 コース状態 入力	センサ基板 コース状態 入力	センサ基板 コース状態 入力	センサ基板 コース状態 入力	センサ基板 コース状態 入力	センサ基板 コース状態 入力	センサ基板 コース状態 兼 スタートバー状態 入力
1	未接続	未接続	RxD0 入力	TxD0 出力	RY_R8C38 ボ ード上の SW3 入力	RY_R8C38 ボ ード上の SW2 入力	RY_R8C38 ボ ード上の SW1 入力	RY_R8C38 ボ ード上の SW0 入力
2	モータドライブ 基板 LED2 出力	モータドライブ 基板 LED3 出力	モータドライブ 基板 サーボ 出力	モータドライブ 基板 右モータ PWM 出力	モータドライブ 基板 右モータ方向 出力	モータドライブ 基板 左モータ PWM 出力	モータドライブ 基板 左モータ方向 出力	モータドライブ 基板 プッシュスイッチ 入力
3	未接続	未接続	未接続	未接続	未接続	未接続	未接続	未接続
4	クリスタル 出力	クリスタル 入力	RY_R8C38 ボ ード上の LED 出力	時計用 クリスタル ※未接続 出力	時計用 クリスタル ※未接続 出力	Vcc 入力		
5	未接続	未接続	未接続	未接続	未接続	未接続	未接続	未接続
6	未接続	未接続	未接続	未接続	未接続	未接続	未接続	未接続
7	未接続	未接続	未接続	未接続	未接続	未接続	未接続	未接続
8	未接続	未接続	未接続	未接続	未接続	未接続	未接続	未接続
9			未接続	未接続	未接続	未接続	未接続	未接続

※リセット後は、全て入力ポートです。

※表の斜線の bit は、端子がない bit です。

※太線枠は、RY_R8C38 ボード上で使っている端子です。

※P4_2 のみ入力専用端子です。他の端子は、入出力可能です。

※P4_2 は、A/D コンバータ用基準電圧を入力するか、通常の入力端子として使用します。今回は、A/D コンバータ用基準電圧として Vcc を接続しています。

※P4_3、P4_4 に時計用クリスタルが接続されている場合は、両ビットとも入力にしてください。

入出力方向の設定は、ポート P0 方向レジスタ(PD0)～ポート P9 方向レジスタ(PD9)で行います。これらのレジスタは、下記の内容に則って入出力方向を設定します。

- ①外部へ信号を出力する端子は、“1”を設定する
- ②外部から信号を入力する端子は、“0”を設定する
- ③未接続の端子は、プルアップ抵抗、またはプルダウン抵抗を接続して入力(“0”)にするか、何も接続せずに出力(“1”)にする。今回は、後者で設定する
- ④端子が無い場合(表の斜線部分)は、“0”にする

6. プログラム解説「kit12_38a.c」

①～④の考え方を基に、"1"、"0"で書き換えた表を下記に示します。

ポート	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	16進数
0	0	0	0	0	0	0	0	0	0x00
1	1	1	0	1	0	0	0	0	0xd0
2	1	1	1	1	1	1	1	1	0xff
3	1	1	1	1	1	1	1	1	0xff
4	1	0	1	1	1	0	0	0	0xb8
5	1	1	1	1	1	1	1	1	0xff
6	1	1	1	1	1	1	1	1	0xff
7	1	1	1	1	1	1	1	1	0xff
8	1	1	1	1	1	1	1	1	0xff
9	0	0	1	1	1	1	1	1	0x3f

C 言語は2進数表記はできないので、10進数か16進数に変換します。2進数を16進数に変換する方が簡単なため、通常は16進数に変換します。

表より、ポートP0 方向レジスタ(PD0)～ポートP9 方向レジスタ(PD9)の設定を下記に示します。

ポート	方向レジスタ名	設定値
0	PD0	0x00
1	PD1	0xd0
2	PD2	0xff
3	PD3	0xff
4	PD4	0xb8
5	PD5	0xff
6	PD6	0xff
7	PD7	0xff
8	PD8	0xff
9	PD9	0x3f

※PD0 を設定するときの注意点

R8C/38A マイコンには、プログラムが暴走したときに備え重要なレジスタは簡単に書き換えられないように保護するレジスタがあります。

PD0～PD9 の中で、PD0だけは保護(プロテクト)されており、保護を解除しないと書き換えることができません。PD0を書き換える前に、プロテクトレジスタ(PRCR)の bit2 を"1"にします。このビットは「sfr_r838a.h (R8C/38A のレジスタ定義ファイル)」で「PRC2」と定義されており、「PRC2」と記述すると、PRCRのbit2の意味になります。プログラムを下記に示します。

```
prc2 = 1;      ←PD0の書き換えを許可 (PRC2=PRCRのbit2の意味です)
pd0 = 0x00;    ←その後、書き換える
```

PRC2を"1"にした後、次に何かの命令を実行するとPRC2は自動的に"0"(書き換え不可)になります。そのため、必ず PRC2 を"1"にした次の行で PD0 を設定してください。プログラムで PRC2 を"0"に戻す必要はありません。

6.4.8 init関数(端子のプルアップ)

R8C/38A マイコンのポートのほとんどに、プルアップ抵抗が内蔵されています。入力ポートの場合、マイコン内蔵のプルアップ抵抗を ON にすれば、外付けでプルアップ抵抗を付ける必要がありません。

内蔵プルアップの抵抗値は、25~100kΩです。標準は 50kΩです。

今回、ポート 0 の bit3~bit0 に接続されているディップスイッチには、プルアップ抵抗が接続されていません。ディップスイッチは、下記の電圧を出力します。

- スイッチを下側にしたとき:0V 出力
- スイッチを上側にしたとき:無接続なので、電圧は出力されない

そのため、内蔵プルアップを ON にします。ディップスイッチに接続されている P1_3~P1_0 端子のプルアップを ON にするプログラムを下記に示します。

```
510 :     pur0 = 0x04;          /* P1_3~P1_0のプルアップON */
```

6.4.9 init関数(タイマRBの設定)

タイマ RB を使って、1msごとに割り込みを発生させます。詳しくは、「マイコン実習マニュアル(R8C/38A 版) プロジェクト:timer2」を参照してください。

```
512 :     /* タイマRBの設定 */
513 :     /* 割り込み周期 = 1 / 20[MHz] * (TRBPRE+1) * (TRBPR+1)
514 :           = 1 / (20*10^6) * 200      * 100
515 :           = 0.001[s] = 1[ms]
516 :     */
517 :     trbmr = 0x00;             /* 動作モード、分周比設定 */
518 :     trbpre = 200-1;          /* プリスケーラレジスタ */
519 :     trbpr = 100-1;           /* プライマリレジスタ */
520 :     trbic = 0x07;            /* 割り込み優先レベル設定 */
521 :     trbcr = 0x01;            /* カウント開始 */
```

TRBPRE と TRBPR の値を計算する式を、下記に示します。

$$(TRBPRE + 1) \times (TRBPR + 1) = \text{タイマ RB 割り込み要求周期} / \text{タイマ RB カウントソース}$$

今回、設定する割り込み周期は 1ms です。タイマ RB カウントソースとは、タイマ RB モードレジスタ(TRBMR)のカウントソース選択ビット(bit5,4)で設定している内容で、今回は f1 (50ns) です。よって、

$$\begin{array}{ccc} (TRBPRE + 1) \times (TRBPR + 1) & = & \text{タイマ RB 割り込み要求周期} / \text{タイマ RB カウントソース} \\ (TRBPRE + 1) \times (TRBPR + 1) & = & (1 \times 10^{-3}) / (50 \times 10^{-9}) \\ \underline{(TRBPRE + 1)} \times \underline{(TRBPR + 1)} & = & \underline{20,000} \\ B & C & A \end{array}$$

A…1~65,536 にする必要があります。65,537 以上の場合、カウントソースを長い時間に設定し直してください。

B…1~256 になるよう、値を設定してください。値は整数です。

C…1~256 になるよう、値を設定してください。値は整数です。

今回、A は 20,000 なので、A の条件は満たしています。

6. プログラム解説「kit12_38a.c」

B、C を決める公式はありません。B×C が、20,000 になるような数字を適宜見つけてください。

例えば、B=200 とすると、

$$200 \times C = 20,000$$

$$\therefore C = 100$$

となります。

$$B = \text{TRBPRE} + 1 \quad \therefore \text{TRBPRE} = 200 - 1 = \mathbf{199}$$

$$C = \text{TRBPR} + 1 \quad \therefore \text{TRBPR} = 100 - 1 = \mathbf{99}$$

を今回設定します。

6.4.10 init関数(タイマRDの設定)

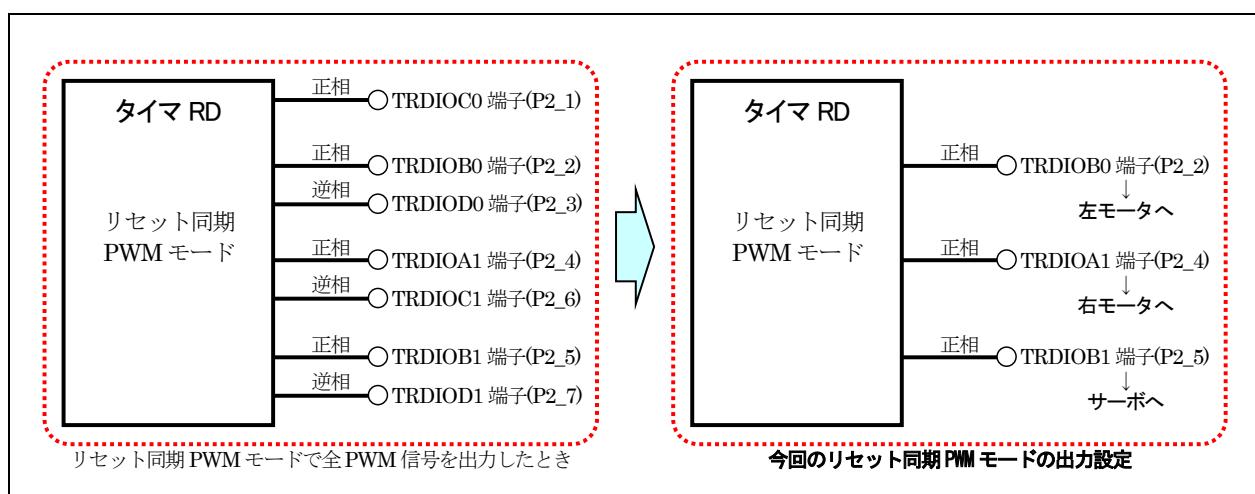
タイマ RD をリセット同期 PWM モードで使って、左モータ、右モータ、サーボへの PWM 信号を出力します。詳しくは、「マイコン実習マニュアル(R8C/38A 版)」の「プロジェクト:timer_rd_doukipwm」を参照してください。

```

523 :     /* タイマRD リセット同期PWMモードの設定*/
524 :     /* PWM周期 = 1 / 20[MHz] * カウントソース * (TRDGRA0+1)
525 :             = 1 / (20*10^6) * 8           * 40000
526 :             = 0.016[s] = 16[ms]
527 : */
528 :     trdpsr0 = 0x08;                  /* TRDIOB0, C0, D0端子設定      */
529 :     trdpsr1 = 0x05;                  /* TRDIOA1, B1, C1, D1端子設定  */
530 :     trdmr   = 0xf0;                  /* バッファレジスタ設定          */
531 :     trdfcr  = 0x01;                  /* リセット同期PWMモードに設定    */
532 :     trdcr0  = 0x23;                  /* ソースカウントの選択:f8       */
533 :     trdgca0 = trdgcc0 = PWM_CYCLE;  /* 周期                          */
534 :     trdgrb0 = trdgrd0 = 0;          /* P2_2端子のON幅設定          */
535 :     trdgca1 = trdgcc1 = 0;          /* P2_4端子のON幅設定          */
536 :     trdgrb1 = trdgrd1 = SERVO_CENTER; /* P2_5端子のON幅設定          */
537 :     trdoer1 = 0xcd;                  /* 出力端子の選択                */
538 :     trdstr  = 0x0d;                  /* TRD0カウント開始              */

```

今回のリセット同期 PWM モードの設定をしたときの出力端子を、下記に示します。



6.4.11 intTRB関数(1msごとの割り込み)

intTRB 関数は、タイマ RB で設定した関数で 1ms ごとに実行されます。

```

541 : /*****
542 : /* タイマRB 割り込み処理
543 : *****/
544 : #pragma interrupt intTRB(vect=24)
545 : void intTRB( void )
546 : {
547 :     cnt0++;
548 :     cnt1++;
549 : }
```

544 行	<p>#pragma interrupt 割り込み処理関数名(vect=ソフトウェア割り込み番号) とすることで、ソフトウェア割り込み番号の割り込みが発生したとき、割り込み処理関数名を実行します。 タイマ RB 割り込みのソフトウェア割り込み番号は表より、24 番です。 今回は、24 番の割り込みが発生したとき、intTRB 関数を実行するよう、「#pragma interrupt」で設定します。</p>
545 行	タイマ RB 割り込みにより実行する関数です。割り込み関数は、引数、戻り値ともに指定することはできません。すなわち、「void 関数名(void)」である必要があります。
547 行	cnt0 変数を+1 します。この関数は 1ms ごとに実行されるので、cnt0 変数は 1ms ごとに+1 されることになります。
548 行	cnt1 変数を+1 します。この関数は 1ms ごとに実行されるので、cnt1 変数は 1ms ごとに+1 されることになります。

6.4.12 timer関数(時間稼ぎ)

timer 関数は、時間稼ぎをする関数です。

```

551 : /***** *****/
552 : /* タイマ本体 */
553 : /* 引数 タイマ値 1=1ms */
554 : /***** *****/
555 : void timer( unsigned long timer_set )
556 : {
557 :     cnt0 = 0;
558 :     while( cnt0 < timer_set );
559 : }
```

(1) timer関数の使い方

timer 関数の使い方を下記に示します。

```
timer( 時間稼ぎをしたい時間 );
```

引数は、時間稼ぎをしたい時間をミリ秒単位で設定します。使用例を下記に示します。

```
motor( 50, 100 ); ←数100μs
timer( 1000 ); ←1000ms
```

モータを制御してから、timer 関数で 1000ms 待ちます。

(2) プログラムの解説

「timer(1000);」を実行したとして、説明します。

557 行	cnt0 を 0 にクリアします。
	<p>まず下記プログラムが実行されます。</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <pre>while(cnt0 < timer_set);</pre> </div> <p>① cnt0 変数は 557 行で 0 にクリアされています。timer_set 変数は関数の引数です。よって、</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <pre>while(0 < 1000);</pre> </div> <p>となります。カッコの中が成り立ちませんので、この命令を繰り返し続けます。</p>
558 行	<p>1ms 後(cnt0 をクリア後、1 回目は 1ms 以内に)、intRB 割り込みが発生します。現在実行している while 文の実行を止めて、割り込みプログラムである命令が実行されます。</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p>547 : cnt0++; ← 0++なので、1になる</p> </div> <p>よって、cnt0 は 1 になります。割り込みプログラムが終了すると、割り込みが発生した行に戻ります。</p>

	<p>戻った行は、while 文です。</p> <div style="border: 1px solid black; padding: 5px; text-align: center;">while(1 < 1000);</div> <p>cnt0 部分が割り込みプログラム内でプラスされたので 1 になりました。まだ、カッコの中が成り立ちませんので、この命令を繰り返し続けます。</p>
④	<p>②を 1000 回実行すると、cnt0 は 1000 になります。</p> <div style="border: 1px solid black; padding: 5px; text-align: center;">while(1000 < 1000);</div> <p>while 文の条件式が成り立たなくなりました。while 文を終えて次の行に進みます。次の行は無いので、timer 関数を終了します。intRB 関数は、タイマ RB 割り込みで 1ms ごとに実行されます。1000 になるまで while 文を繰り返し続けたと言うことは、1000ms の間、while 文を繰り返し続けたということになります。</p> <p>このように、タイマ RB による 1ms ごとの割り込みを実行した回数を timer 関数で数えて、時間を計ります。</p>

6.4.13 sensor_inp 関数(センサ状態の読み込み)

sensor_inp 関数は、センサ基板からセンサの状態を読み込む関数です。

```

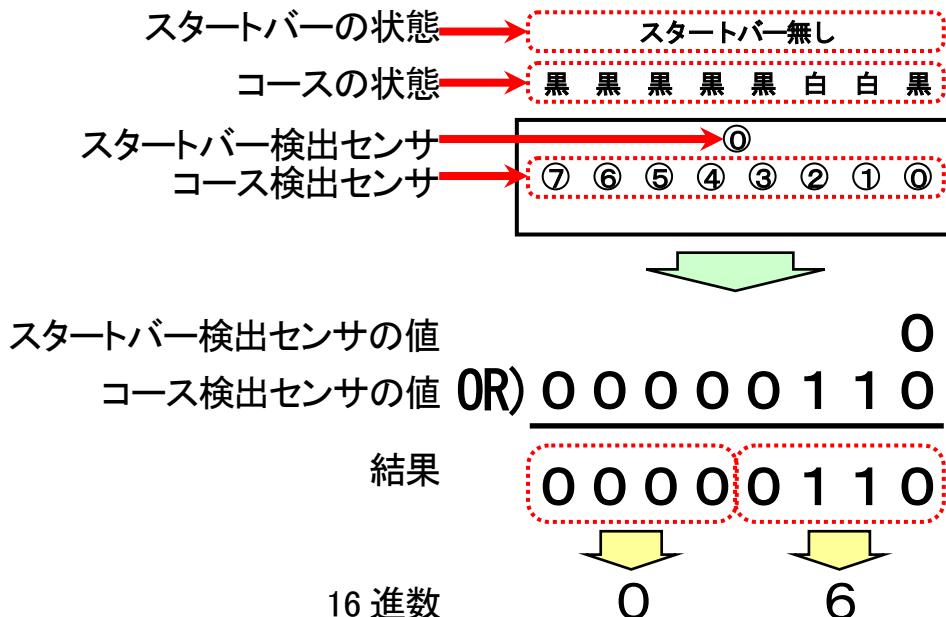
561 : /*****
562 : /* センサ状態検出
563 : /* 引数 マスク値
564 : /* 戻り値 センサ値
565 : *****/
566 : unsigned char sensor_inp( unsigned char mask )
567 : {
568 :     unsigned char sensor;
569 :
570 :     sensor = ~p0;
571 :
572 :     sensor &= mask;
573 :
574 :     return sensor;
575 : }
```

568 行	sensor 変数を unsigned char 型で宣言します。この変数は、sensor_inp 関数内でセンサの状態を加工する変数です。センサ基板の情報は、ポート 0 から 8bit 幅で読み込みます。よって、8bit 幅である char 型とし、符号無しで宣言しています。
570 行	ポート 0 からセンサ基板の情報を読み込みます。センサからの出力信号は、白="0"、黒="1"と人間の感覚とは逆で分かりづらいので、「~(チルダ)」で反転させ、白="1"、黒="0"に変換しています。
572 行	センサの情報 sensor 変数と sensor_inp 関数の引数であるマスク値を AND 演算し、sensor 変数に代入します。不要なビットを強制的に"0"にしています。
574 行	sensor 変数を戻り値にして、関数を終了します。

(1) bit0 の扱い

センサ基板Ver.5のいちばん右のコース検出センサとスタートバー検出センサは、ポート0のbit0の端子にOR接続された状態で入力されます。その様子を下図に示します。

※値は、プログラムで反転後の状態です。



bit0 が”1”になったとき、スタートバー検出センサが反応したのか、コース検出センサの bit0 が反応したのか分かりません。ただし、次の表のとおり、2 つのセンサを同時に使うことはありません。

状態	写真	説明
スタート前	<p>スタートバー検出センサのモニタ</p>	<p>センサ基板をコース中心に置けば、いちばん右のコース検出センサは必ず黒色なので、 bit0=“1”：スタートバーあり “0”：スタートバーなし と判断できます。</p>
スタート後	<p>いちばん右のコース検出センサのモニタ</p>	<p>スタート後はスタートバー検出センサの反応は無いので、 bit0=“1”：いちばん右のコースは白色 “0”：いちばん右のコースは黒色 と判断できます。</p>

(2) マスク処理

572 行でマスクという処理を行っています。マスクとは、チェックに不要なビットを強制的に”0”にすることで、AND 演算を行います。

①なぜマスク処理が必要か

1ポートの単位は8ビットのため、**1ビットだけチェックすることはできません**（ビットフィールドという方法を使えばできますが、ここでは無しにします）。**必ず 8ビットまとめてのチェックとなります。**

例えば、センサの左端である bit7 が”1”かどうかチェックしたい場合、

```
if( センサの値 == 0x80 ) {
    /* bit7 が “1” ならこの中を実行 */
}
```

とすればいいように思えます。しかし、bit6～0 がどのような値になっているか分かりません。例えば、bit7 が”1”、bit0 も”1”なら

```
if( センサの値 0x81 == 0x80 ) {           ←センサの値=1000 0001=0x81
    /* bit7 が “1” ならこの中を実行 */
}
```

となります。bit7 は”1”ですが、bit0 も”1”的め、0x80 ではないと判断されて、カッコの中は実行されません。これでは、うまくチェックできません。そのため、マスクという処理を行います。

②マスク処理を行うと

先ほどの方法では、bit7 以外の値が”0”か”1”か分からなかつたため、うまくチェックできませんでした。そこでマスク処理を行い、bit6～bit0 の値を強制的に”0”にします。

bit6～bit0 は必ず”0”だと分かっているので、bit6～bit0 は”0”という前提でセンサ値をチェックします。もし、bit7 が”1”かどうかチェックしたい場合は、

```
if( bit7 はそのまま bit6～bit0 を“0”にしたセンサの値==0x80 ) {
    /* bit7 が “1” ならこの中を実行 */
}
```

とすれば、bit7 が”1”か”0”か判断することができ、bit7 が”1”ならカッコの中を実行します。

③マスク値の決め方と演算方法

プログラムでは、論理演算の論理積、すなわち AND 演算を行います。

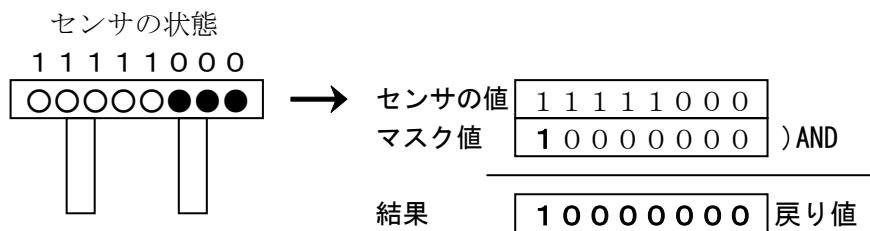
例えば、センサの状態が「白白白白黒黒黒」のとき、センサ値は「1111 1000」です。bit7 をチェックする場合、bit6～0 はチェックするときに不要です。

bit	7	6	5	4	3	2	1	0
必要	必要	不要						

不要な部分を“0”にするためには、不要ビットのマスク値を“0”にして AND 演算を行います。したがって、**マスク値は上表の不要部分を“0”に、必要部分を“1”に書き換えれば良いことになります。**

bit	7	6	5	4	3	2	1	0
マスク値	1	0	0	0	0	0	0	0

センサの値とマスク値を AND 演算し、その結果とチェックしたい値を比較します（下図）。



上記をプログラムで表すと、下記のようになります。

```
if( (センサの値 & 0x80) == 0x80 ) {
    /* bit7="1"ならこの中を実行 */
}
```

bit6～bit0 はマスク処理によって強制的に“0”になっていることが分かっているので、bit6～bit0 は“0”として比較します。

④sensor_inp 関数の構造

sensor_inp 関数では、センサの状態取得とマスク処理を行います。
センサ値の反転をした後、最後にマスク処理を行います。

```

561 : /*****
562 : /* センサ状態検出
563 : /* 引数 マスク値
564 : /* 戻り値 センサ値
565 : *****/
566 : unsigned char sensor_inp( unsigned char mask )
567 : {
568 :     unsigned char sensor;
569 :
570 :     sensor = ~p0;
571 :
572 :     sensor &= mask; ←
573 :
574 :     return sensor;
575 : }
```

センサ値を加工したいいちばん最後で
マスク処理する

⑤マスク値の定義

kit12_38a.c では、よく使うマスク値を define 定義しています。プログラムでは、30～38 行の部分です。
下記のように定義しています。

#define MASK[A][B]	マスク値
--------------------	------

定義のルールは、「MASK」+「[A]」+「_(アンダバー)」+「[B]」として、マスク値を定義しています。意味は下記のようになります。

- ・[A]…左のセンサ 4 個中 [A] 個を有効にする
- ・[B]…右のセンサ 4 個中 [B] 個を有効にする
- ・その他をマスクする

今回定義した内容を、一覧表で示します。

定義したマスク文字列	マスク値	2進数	意味
MASK2_2	0x66	0110 0110	左の真ん中 2 個、右の真ん中 2 個を有効に、他をマスクする。
MASK2_0	0x60	0110 0000	左の真ん中 2 個を有効に、他をマスクする。
MASK0_2	0x06	0000 0110	右の真ん中 2 個を有効に、他をマスクする。
MASK3_3	0xe7	1110 0111	左 3 個、右 3 個を有効に、他をマスクする。
MASK0_3	0x07	0000 0111	右 3 個を有効に、他をマスクする。
MASK3_0	0xe0	1110 0000	左 3 個を有効に、他をマスクする。

6. プログラム解説「kit12_38a.c」

MASK4_0	0xf0	1111 0000	左 4 個を有効に、他をマスクする。
MASK0_4	0x0f	0000 1111	右 4 個を有効に、他をマスクする。
MASK4_4	0xff	1111 1111	右 4 個を有効に、左 4 個を有効に、他をマスクする。 ※結果、MASK4_4 はマスクせずにすべてのセンサを有効にしています。これは、sensor_inp 関数のカッコの中には必ず値を入れなければいけないので、マスクする必要がなくても「0xff」として全ビット有効にする値を入れています。

sensor_inp 関数のカッコの中に設定するマスク値は、上記のマスク文字列を設定します。

マスクしたいマスク文字列が無ければ、自分で定義して増やしてください。または、マスク文字列を使わずに、直接数値をいれても構いません。

(3) sensor_inp 関数の使い方

```
if( sensor_inp( [マスク値] ) == [センサチェック値] ) {
    式が成り立つならこの中を実行
} else {
    式が成り立たないなら、この中を実行
}
```

[マスク値]には、センサの値をマスクする値、[センサチェック値]はマスク後にチェックしたい値を入れます。

例えば、センサ値は 0x1f、マスク値は MASK0_2、センサチェック値は 0x04 のときの動作を下記に示します。

```
if( sensor_inp( [MASK0_2] ) == [0x04] ) {
    式が成り立つならこの中を実行
} else {
    式が成り立たないなら、この中を実行
}
```

①	センサの値は「0001 1111」、sensor_inp 関数のマスク値は「0000 0110」です。AND 演算を行うと結果は下記のようになります。 <table style="margin-left: 100px;"> <tr> <td>センサの値</td><td>0001 1111</td></tr> <tr> <td>マスク値</td><td>0000 0110 (AND)</td></tr> <tr> <td colspan="2"><hr/></td></tr> <tr> <td>結果</td><td>0000 0110 → 16進数で 0x06</td></tr> </table>	センサの値	0001 1111	マスク値	0000 0110 (AND)	<hr/>		結果	0000 0110 → 16進数で 0x06
センサの値	0001 1111								
マスク値	0000 0110 (AND)								
<hr/>									
結果	0000 0110 → 16進数で 0x06								
②	結果の 0x06 とセンサチェック値の 0x04 をチェックします。一致しないので、「式が成り立たないなら、この中を実行」部分を実行します。								

6.4.14 check_crossover 関数(クロスラインチェック)

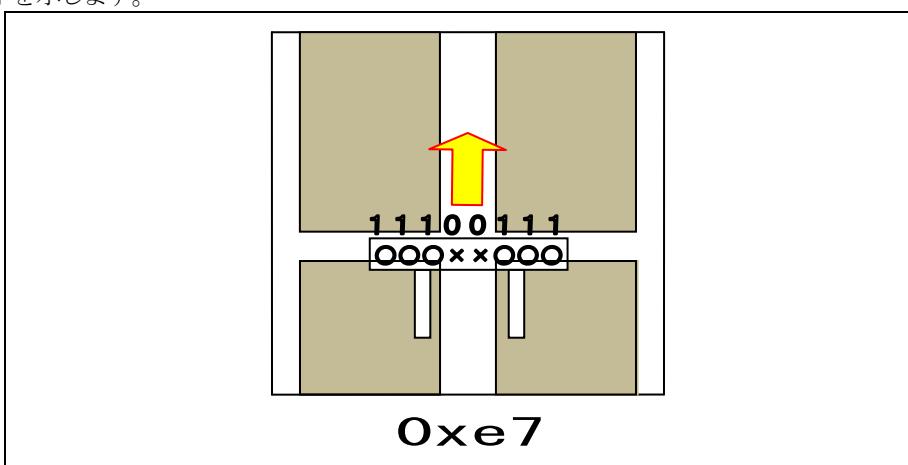
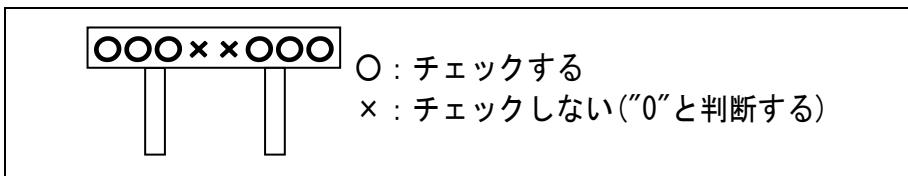
クランクの 500mm～1000mm 手前に横線があります。これをクロスラインと呼びます。check_crossover 関数は、クロスラインの検出を行う関数です。

戻り値は、クロスラインと判断すると 1、クロスラインでなければ 0 とします。

```

577 : /*****
578 : /* クロスライン検出処理
579 : /* 戻り値 0:クロスラインなし 1:あり
580 : *****/
581 : int check_crossover( void )
582 : {
583 :     unsigned char b;
584 :     int ret;
585 :
586 :     ret = 0;
587 :     b = sensor_inp(MASK3_3);
588 :     if( b==0xe7 ) {
589 :         ret = 1;
590 :     }
591 :     return ret;
592 : }
```

586 行	戻り値を保存する ret 変数を初期化しています。ret 変数には、クロスラインなら 1、違うなら 0 を代入します。今のところどちらか分からないので、とりあえず違うと判断して 0 を入れておきます。
587 行	センサを読み込み、変数 b へ保存します。センサのマスク値は、「MASK3_3(0xe7)」なので、左 3 個、右 3 個の合計 6 個のセンサを読み込みます。 読み込むセンサを下図に示します。
588 行	センサの状態が 0xe7 かどうかチェックします。0xe7 は、クロスラインを検出したと判断します。下図にその様子を示します。



センサが、0xe7 なら if の条件が成立るので ret 変数は 1、違うなら ret 変数は変化せず 0 のままとなります。

ret 変数が戻り値なので、"1" = クロスラインあり、"0" = クロスラインなし、ということになります。

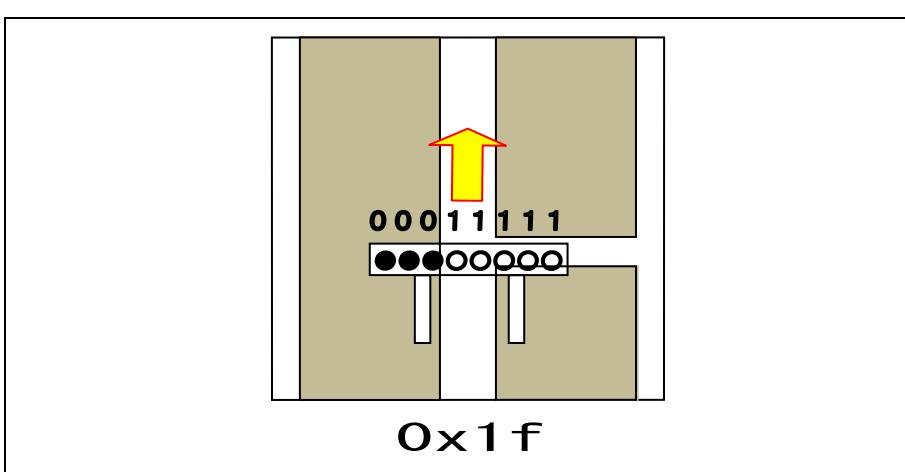
6.4.15 check_rightline 関数(右ハーフライン検出処理)

右レーンチェンジの 300mm～1000mm 手前に右ハーフラインがあります。check_rightline 関数は、右ハーフラインの検出を行う関数です。

戻り値は、右ハーフラインと判断すると 1、右ハーフラインでなければ 0 とします。

```

594 : /*****
595 : /* 右ハーフライン検出処理
596 : /* 戻り値 0:なし 1:あり
597 : *****/
598 : int check_rightline( void )
599 : {
600 :     unsigned char b;
601 :     int ret;
602 :
603 :     ret = 0;
604 :     b = sensor_inp(MASK4_4);
605 :     if( b==0x1f ) {
606 :         ret = 1;
607 :     }
608 :     return ret;
609 : }
```

603 行	戻り値を保存する ret 変数を初期化しています。ret 変数には、右ハーフラインなら 1、違うなら 0 を代入します。今のところどちらか分からないので、とりあえず違うと判断して 0 を入れておきます。
604 行 ～ 607 行	<p>センサの状態をチェックします。マスク値は、MASK4_4 なのでセンサの状態を全て読み込みます。このとき、0x1fなら右ハーフラインを検出したと判断します。下図にその様子を示します。</p>  <p>センサが、0x1f なら if の条件が成立るので ret 変数は 1、違うなら ret 変数は変化せず 0 のままとなります。 ret 変数が戻り値なので、"1" = 右ハーフラインあり、"0" = 右ハーフラインなし、ということになります。</p>

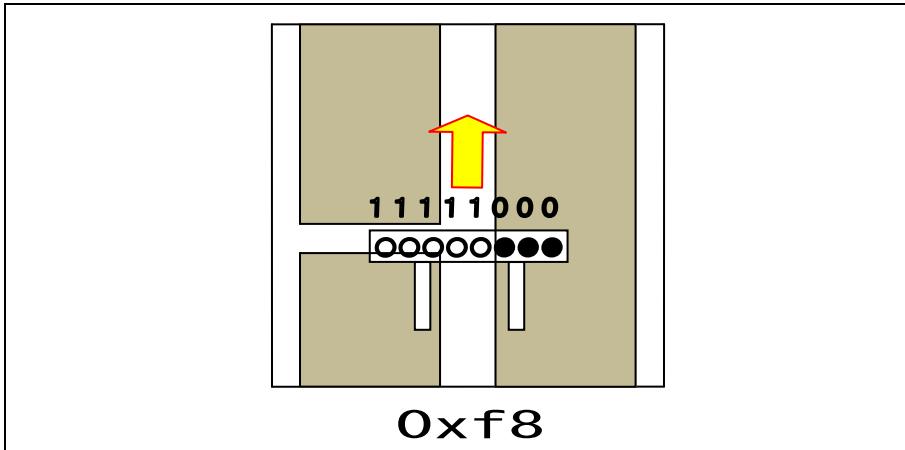
6.4.16 check_leftline関数(左ハーフライン検出処理)

左レーンチェンジの300mm～1000mm手前に左ハーフラインがあります。check_leftline 関数は、左ハーフラインの検出を行う関数です。

戻り値は、左ハーフラインと判断すると 1、左ハーフラインでなければ 0 とします。

```

611 : /*****
612 : /* 左ハーフライン検出処理
613 : /* 戻り値 0:なし 1:あり
614 : *****/
615 : int check_leftline( void )
616 : {
617 :     unsigned char b;
618 :     int ret;
619 :
620 :     ret = 0;
621 :     b = sensor_inp(MASK4_4);
622 :     if( b==0xf8 ) {
623 :         ret = 1;
624 :     }
625 :     return ret;
626 : }
```

620 行	戻り値を保存する ret 変数を初期化しています。ret 変数には、左ハーフラインなら 1、違うなら 0 を代入します。今のところどちらか分からないので、とりあえず違うと判断して 0 を入れておきます。
621 行 ～ 624 行	<p>センサの状態をチェックします。マスク値は、MASK4_4 なのでセンサの状態を全て読み込みます。このとき、0xf8 なら左ハーフラインを検出したと判断します。下図にその様子を示します。</p>  <p>センサが、0xf8 なら if の条件が成立つので ret 変数は 1、違うなら ret 変数は変化せず 0 のままとなります。 ret 変数が戻り値なので、"1" = 左ハーフラインあり、"0" = 左ハーフラインなし、ということになります。</p>

6.4.17 dipsw_get関数(ディップスイッチ値読み込み)

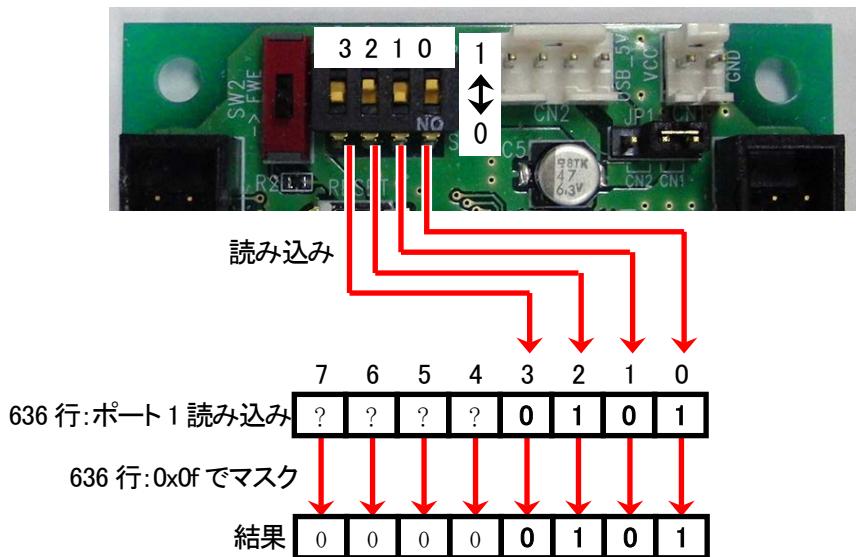
dipsw_get 関数は、RY_R8C38 ボードのディップスイッチの状態を読み込む関数です。
戻り値は、ディップスイッチの値によって 0~15 の値が返ってきます。

```
628 : /*****  
629 : /* ディップスイッチ値読み込み */  
630 : /* 戻り値 スイッチ値 0~15 */  
631 : /*****  
632 : unsigned char dipsw_get( void )  
633 : {  
634 :     unsigned char sw;  
635 :  
636 :     sw = p1 & 0x0f;           /* P1_3~P1_0読み込み */  
637 :  
638 :     return  sw;  
639 : }
```

RY_R8C38 ボードには 4bit のディップスイッチが搭載されています。左から順にマイコンの P1_3、P1_2、P1_1、P1_0 の各 bit に接続されています。

ポート 1 の bit7~4 は関係ないので、ビット演算を使ってディップスイッチの bit だけ取り込みます。dipsw_get 関数を呼ぶと、0~15 の値が返ってきます。ディップスイッチを上にすると"1"、下(ON と書いてある側)にすると"0"です。

ディップスイッチが"0101"(下上下上)のときの、dipsw_get 関数の動きを下図に示します。



6.4.18 pushsw_get関数(プッシュスイッチ値読み込み)

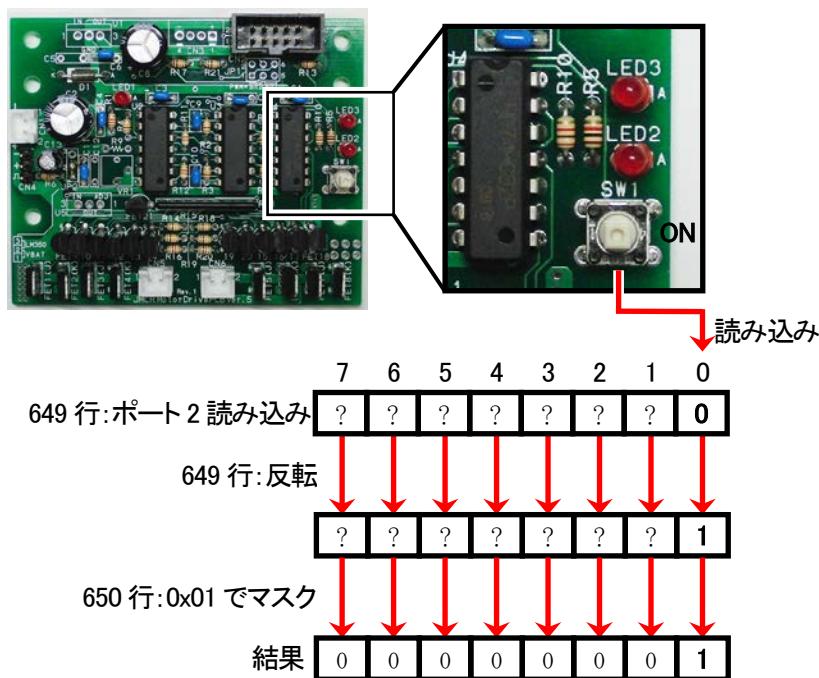
pushsw_get 関数は、モータドライブ基板のプッシュスイッチの値を読み込む関数です。
戻り値は、プッシュスイッチが押されると 1、離されていると 0 が返ってきます。

```

641 : /*****
642 : /* プッシュスイッチ値読み込み
643 : /* 戻り値 スイッチ値 ON:1 OFF:0
644 : *****/
645 : unsigned char pushsw_get( void )
646 : {
647 :     unsigned char sw;
648 :
649 :     sw = ~p2;           /* スイッチのあるポート読み込み */
650 :     sw &= 0x01;
651 :
652 :     return sw;
653 : }
```

プッシュスイッチは、ポート 2 の bit0 に接続されています。ポート 2 の bit7~1 は関係ないので、ビット演算を使ってプッシュスイッチの bitだけ取り込みます。

プッシュスイッチを押したときの動きを下図に示します。



6.4.19 startbar_get関数(スタートバー検出センサ読み込み)

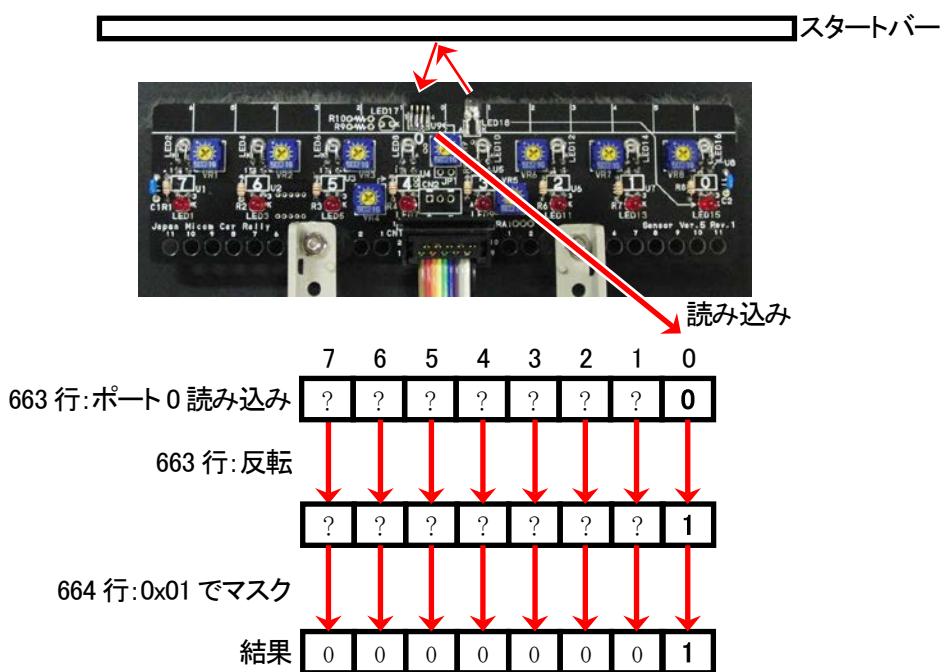
startbar_get 関数は、スタートバーがある(閉じている)かない(開いている)か、状態を読み込む関数です。戻り値は、スタートバーがあると 1、なければ 0 が返ってきます。

```

655 : /*****
656 : /* スタートバー検出センサ読み込み */
657 : /* 戻り値 センサ値 ON(バーあり):1 OFF(なし):0 */
658 : *****/
659 : unsigned char startbar_get( void )
660 : {
661 :     unsigned char b;
662 :
663 :     b = ~p0;                      /* スタートバー信号読み込み */
664 :     b &= 0x01;
665 :
666 :     return b;
667 : }
```

スタートバー検出センサは、ポート 0 の bit0 に接続されています。ポート 0 の bit0 以外は関係ないので、ビット演算を使ってスタートバー検出センサの bit だけ取り込みます。

スタートバーがあるときの動きを下図に示します。



6.4.20 led_out関数(LED制御)

led_out 関数は、モータドライブ基板の LED2、LED3 を消灯／点灯させる関数です。

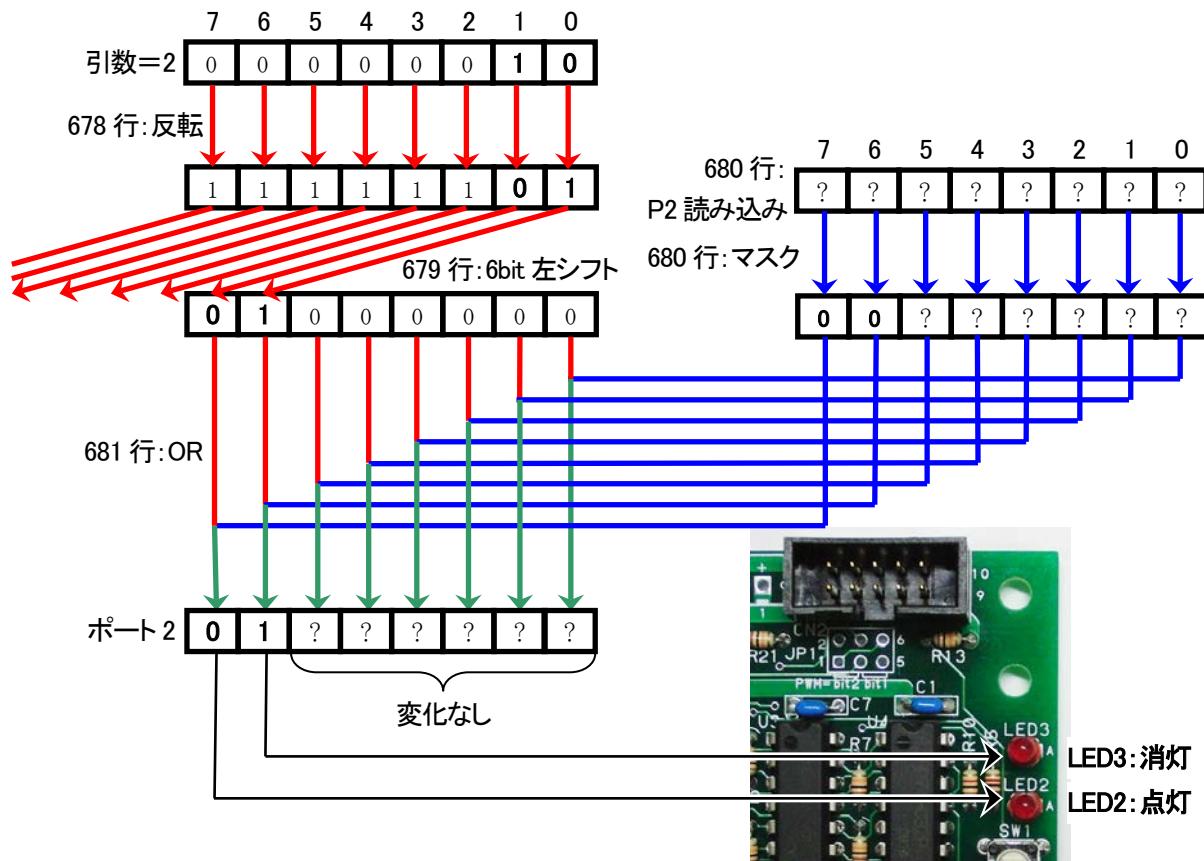
引数と LED2、LED3 の関係を下記に示します。

引数	2進数	LED2	LED3
0	0 0	消灯	消灯
1	0 1	消灯	点灯
2	1 0	点灯	消灯
3	1 1	点灯	点灯

```

669 : /*****
670 : /* LED制御
671 : /* 引数 スイッチ値 LED2:bit1 LED3:bit0 "0":消灯 "1":点灯
672 : /* 例)0x3→LED2:ON LED3:ON 0x2→LED2:ON LED3:OFF
673 : *****/
674 : void led_out( unsigned char led )
675 : {
676 :     unsigned char data;
677 :
678 :     led = ~led;
679 :     led <= 6;
680 :     data = p2 & 0x3f;
681 :     p2 = data | led;
682 : }
```

引数が 2 のときの動きを下図に示します。



6.4.21 motor関数(モータ速度制御)

左モータ、右モータへ PWM を出力する関数です。また、引数の符号により正転、逆転の制御も行います。

```

684 : /*****
685 : /* モータ速度制御
686 : /* 引数 左モータ:-100～100、右モータ:-100～100
687 : /* 0で停止、100で正転100%、-100で逆転100%
688 : /* 戻り値 なし
689 : *****/
690 : void motor( int accele_l, int accele_r )
691 : {
692 :     int     sw_data;
693 :
694 :     sw_data = dipsw_get() + 5;
695 :     accele_l = accele_l * sw_data / 20;
696 :     accele_r = accele_r * sw_data / 20;
697 :
698 :     /* 左モータ制御 */
699 :     if( accele_l >= 0 ) {
700 :         p2 &= 0xfd;
701 :         trdgrd0 = (long)( PWM_CYCLE - 1 ) * accele_l / 100;
702 :     } else {
703 :         p2 |= 0x02;
704 :         trdgrd0 = (long)( PWM_CYCLE - 1 ) * ( -accele_l ) / 100;
705 :     }
706 :
707 :     /* 右モータ制御 */
708 :     if( accele_r >= 0 ) {
709 :         p2 &= 0xf7;
710 :         trdgrc1 = (long)( PWM_CYCLE - 1 ) * accele_r / 100;
711 :     } else {
712 :         p2 |= 0x08;
713 :         trdgrc1 = (long)( PWM_CYCLE - 1 ) * ( -accele_r ) / 100;
714 :     }
715 : }
```

(1) motor関数の使い方

motor 関数の使い方を下記に示します。

```
motor( 左モータの PWM 値 , 右モータの PWM 値 );
```

引数は、左モータの PWM 値と右モータの PWM 値をカンマで代入します。値とモータの回転の関係を下記に示します。

値	説明
-100～-1	逆転します。-100 で 100%逆転です。-100 以上の値は設定できません。また、整数のみの設定になります。
0	モータが停止します。
1～100	正転します。100 で 100%正転です。100 以上の値は設定できません。また、整数のみの設定になります。

実際にモータに出力される割合を、下記に示します。

$$\text{左モータに出力される PWM} = \text{motor 関数で設定した左モータの PWM 値} \times \frac{\text{ディップスイッチの値} + 5}{20}$$

$$\text{右モータに出力される PWM} = \text{motor 関数で設定した右モータの PWM 値} \times \frac{\text{ディップスイッチの値} + 5}{20}$$

例えば、motor 関数で左モータに 80 を設定した場合、左モータは正転で 80% の回転をするかというと実はそうではありません。マイコンボード上にあるディップスイッチの値により、実際にモータへ出力される PWM の割合が変化します。

ディップスイッチが、”1100”(10 進数で 12) のとき、下記プログラムを実行したとします。

```
motor( -70 , 100 );
```

実際のモータに出力される PWM 値は、下記のようになります。

$$\text{左モータに出力される PWM} = -70 \times (12+5) \div 20 = -70 \times 0.85 = -59.5 = -59\%$$

$$\text{右モータに出力される PWM} = 100 \times (12+5) \div 20 = 100 \times 0.85 = 85\%$$

左モータの計算結果は-59.5%ですが、小数点は計算できないので切り捨てられ整数になります。よって左モータに出力される PWM 値は逆転 59%、右モータに出力される PWM 値は正転 85%となります。

これから、上記の内容がどのように実行されるのか説明します。

(2) ディップスイッチの割合に応じて、PWM 値を変更

694 :	<code>sw_data = dipsw_get() + 5;</code>	<code>dipsw_get()</code> = ディップスイッチの値 0~15
695 :	<code>accele_l = accele_l * sw_data / 20;</code>	
696 :	<code>accele_r = accele_r * sw_data / 20;</code>	

694 行	sw_data 変数にディップスイッチの値+5 の値を代入します。ディップスイッチの値は 0~15 なので、sw_data 変数の値は、5~20 になります。
695 行	左辺の accele_l 変数は、左モータに加える PWM 値の割合を代入する変数です。右辺の accele_l が motor 関数に設定した左モータの PWM 値です。 よって、次の計算を行って、左モータに加える PWM 値を設定します。 $\text{accele_l} = \text{accele_l} (\text{motor 関数で設定した左モータの PWM}) \times \text{sw_data} / 20$ accele_l 変数の範囲は、-100~100 の値です。
696 行	左辺の accele_r 変数は、右モータに加える PWM 値の割合を代入する変数です。右辺の accele_r が motor 関数に設定した右モータの PWM 値です。 よって、次の計算を行って、右モータに加える PWM 値を設定します。 $\text{accele_r} = \text{accele_r} (\text{motor 関数で設定した右モータの PWM}) \times \text{sw_data} / 20$ accele_r 変数の範囲は、-100~100 の値です。

(3) 左モータ制御

左モータを制御する部分です。左モータの PWM は P2_2 端子から出力します。P2_2 端子から出力する PWM の設定は、タイマ RD ジェネラルレジスタ B0(TRDGRB0)に PWM 値を設定します。ただし、直接設定するのではなく、バッファレジスタであるタイマ RD ジェネラルレジスタ D0(TRDGRD0)に設定します。

```

698 : /* 左モータ制御 */
699 : if( accele_1 >= 0 ) {
700 :     p2 &= 0xfd;           p2 = p2 AND 0xfdという意味です
701 :     trdgrd0 = (long)( PWM_CYCLE - 1 ) * accele_1 / 100;
702 : } else {
703 :     p2 |= 0x02;          p2 = p2 OR 0x02という意味です
704 :     trdgrd0 = (long)( PWM_CYCLE - 1 ) * ( -accele_1 ) / 100;
705 : }
```

699 行	左モータの PWM 値の割合が正の数か負の数かをチェックします。 正の数なら 700～701 行、負の数なら 703～704 行を実行します。																																				
700 行 ～ 701 行	<p>正の数なら 700～701 行を実行します。 P2_1="0"を設定し、P2_2端子からPWM出力すると、PWMの割合に応じて左モータが正転します。 700 行目で下表のビット演算を行い、P2_1 端子を"0"にします。</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>bit</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr> </thead> <tbody> <tr> <td>元の値 (ポート 2)</td><td>P2_7</td><td>P2_6</td><td>P2_5</td><td>P2_4</td><td>P2_3</td><td>P2_2</td><td>P2_1</td><td>P2_0</td></tr> <tr> <td>AND 値</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td></tr> <tr> <td>結果</td><td>P2_7</td><td>P2_6</td><td>P2_5</td><td>P2_4</td><td>P2_3</td><td>P2_2</td><td>0</td><td>P2_0</td></tr> </tbody> </table> <p>701 行目で次の計算を行って、タイマ RD ジェネラルレジスタ D0(TRDGRD0)～PWM 値を設定します。小数点が出た場合は、切り捨てられます。</p> $\begin{aligned} \text{TRDGRD0} &= (\text{PWM_CYCLE} - 1) \times \frac{\text{accele_1 (0~100)}}{100} \\ &= 39998 \times \frac{\text{accele_1 (0~100)}}{100} \end{aligned}$ <p>例えば accele_1=80 なら、TRDGRD0 は次のように計算されます。</p> $\text{TRDGRD0} = 39998 \times 80 / 100 = 31998.4 = 31998$	bit	7	6	5	4	3	2	1	0	元の値 (ポート 2)	P2_7	P2_6	P2_5	P2_4	P2_3	P2_2	P2_1	P2_0	AND 値	1	1	1	1	1	1	0	1	結果	P2_7	P2_6	P2_5	P2_4	P2_3	P2_2	0	P2_0
bit	7	6	5	4	3	2	1	0																													
元の値 (ポート 2)	P2_7	P2_6	P2_5	P2_4	P2_3	P2_2	P2_1	P2_0																													
AND 値	1	1	1	1	1	1	0	1																													
結果	P2_7	P2_6	P2_5	P2_4	P2_3	P2_2	0	P2_0																													

bit	7	6	5	4	3	2	1	0
元の値 (ボルト2)	P2_7	P2_6	P2_5	P2_4	P2_3	P2_2	P2_1	P2_0
OR 値	0	0	0	0	0	0	1	0
結果	P2_7	P2_6	P2_5	P2_4	P2_3	P2_2	1	P2_0

703 行

~
704 行

704 行目で次の計算を行って、タイマ RD ジェネラルレジスタ D0(TRDGRD0)へ PWM 値を設定します。小数点が出た場合は、切り捨てられます。

$$\begin{aligned} \text{TRDGRDO} &= (\text{PWM_CYCLE} - 1) \times \frac{-\text{accele_l} (-1 \sim -100)}{100} \\ &= 39998 \times \frac{-\text{accele_l} (-1 \sim -100)}{100} \end{aligned}$$

ポイントは、`accele_1` 変数が負の数ということです。回路的には `P2_1="1"` で逆転の設定になっているので `accele_1` は、正の数に直して計算します。直し方は、計算式の中で「`-accele_1`」としています。例えば `accele_1=-50` なら、`TRDGRD0` は次のように計算されます。

$$\text{TRDGRD0} = 39998 \times \{-(-50)\} / 100 = 39998 \times 50 / 100 = 19999$$

(4) 右手一夕制御

右モータを制御する部分です。右モータの PWM は P2_4 端子から出力します。P2_4 端子から出力する PWM の設定は、タイマ RD ジェネラルレジスタ A1(TRDGRA1)に PWM 値を設定します。ただし、直接設定するのではなく、バッファレジスタであるタイマ RD ジェネラルレジスタ C1(TRDGRC1)に設定します。

```
707 :     /* 右モータ制御 */
708 :     if( accele_r >= 0 ) {
709 :         p2 &= 0xf7;           p2 = p2 AND 0xf7という意味です
710 :         trdgrc1 = (long)( PWM_CYCLE - 1 ) * accele_r / 100;
711 :     } else {
712 :         p2 |= 0x08;          p2 = p2 OR 0x08という意味です
713 :         trdgrc1 = (long)( PWM_CYCLE - 1 ) * ( -accele_r ) / 100;
714 :     }
```

6. プログラム解説「kit12_38a.c」

708 行	右モータの PWM 値の割合が正の数か負の数かをチェックします。 正の数なら 709~710 行、負の数なら 712~713 行を実行します。																																				
709 行 ～ 710 行	<p>正の数なら 709~710 行を実行します。 P2_3="0"を設定し、P2_4 端子から PWM 出力すると、PWM の割合に応じて右モータが正転します。 709 行目で下表のビット演算を行い、P2_3 端子を"0"にします。</p> <table border="1"> <thead> <tr> <th>bit</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr> </thead> <tbody> <tr> <td>元の値 (ポート 2)</td><td>P2_7</td><td>P2_6</td><td>P2_5</td><td>P2_4</td><td>P2_3</td><td>P2_2</td><td>P2_1</td><td>P2_0</td></tr> <tr> <td>AND 値</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td></tr> <tr> <td>結果</td><td>P2_7</td><td>P2_6</td><td>P2_5</td><td>P2_4</td><td>0</td><td>P2_2</td><td>P2_1</td><td>P2_0</td></tr> </tbody> </table> <p>710 行目で次の計算を行って、タイマ RD ジェネラルレジスタ C1(TRDGRC1)～PWM 値を設定しています。小数点が出た場合は、切り捨てられます。</p> $\text{TRDGRC1} = (\text{PWM_CYCLE} - 1) \times \frac{\text{accele_r } (0\sim100)}{100}$ $= 39998 \times \frac{\text{accele_r } (0\sim100)}{100}$ <p>例えば accele_r=20 なら、TRDGRC1 は次のように計算されます。</p> $\text{TRDGRC1} = 39998 \times 20 / 100 = 7999.6 = 7999$	bit	7	6	5	4	3	2	1	0	元の値 (ポート 2)	P2_7	P2_6	P2_5	P2_4	P2_3	P2_2	P2_1	P2_0	AND 値	1	1	1	1	0	1	1	1	結果	P2_7	P2_6	P2_5	P2_4	0	P2_2	P2_1	P2_0
bit	7	6	5	4	3	2	1	0																													
元の値 (ポート 2)	P2_7	P2_6	P2_5	P2_4	P2_3	P2_2	P2_1	P2_0																													
AND 値	1	1	1	1	0	1	1	1																													
結果	P2_7	P2_6	P2_5	P2_4	0	P2_2	P2_1	P2_0																													
712 行 ～ 713 行	<p>負の数なら 712～713 行を実行します。 P2_3="1"を設定し、P2_4 端子から PWM 出力すると、PWM の割合に応じて右モータが逆転します。 712 行目で下表のビット演算を行い、P2_3 端子を"1"にします。</p> <table border="1"> <thead> <tr> <th>bit</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr> </thead> <tbody> <tr> <td>元の値 (ポート 2)</td><td>P2_7</td><td>P2_6</td><td>P2_5</td><td>P2_4</td><td>P2_3</td><td>P2_2</td><td>P2_1</td><td>P2_0</td></tr> <tr> <td>OR 値</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td></tr> <tr> <td>結果</td><td>P2_7</td><td>P2_6</td><td>P2_5</td><td>P2_4</td><td>1</td><td>P2_2</td><td>P2_1</td><td>P2_0</td></tr> </tbody> </table> <p>713 行目で次の計算を行って、タイマ RD ジェネラルレジスタ C1(TRDGRC1)～PWM 値を設定します。小数点が出た場合は、切り捨てられます。</p> $\text{TRDGRC1} = (\text{PWM_CYCLE} - 1) \times \frac{-\text{accele_r } (-1\sim-100)}{100}$ $= 39998 \times \frac{-\text{accele_r } (-1\sim-100)}{100}$ <p>ポイントは、accele_r 変数が負の数だということです。回路的には P2_3="1"で逆転の設定になっているので accele_r は、正の数に直して計算します。直し方は、計算式の中で「-accele_r」としています。例えば accele_r=-90 なら、TRDGRC1 は次のように計算されます。</p> $\text{TRDGRC1} = 39998 \times \{-(-90)\} / 100 = 39998 \times 90 / 100 = 35998.2 = 35998$	bit	7	6	5	4	3	2	1	0	元の値 (ポート 2)	P2_7	P2_6	P2_5	P2_4	P2_3	P2_2	P2_1	P2_0	OR 値	0	0	0	0	1	0	0	0	結果	P2_7	P2_6	P2_5	P2_4	1	P2_2	P2_1	P2_0
bit	7	6	5	4	3	2	1	0																													
元の値 (ポート 2)	P2_7	P2_6	P2_5	P2_4	P2_3	P2_2	P2_1	P2_0																													
OR 値	0	0	0	0	1	0	0	0																													
結果	P2_7	P2_6	P2_5	P2_4	1	P2_2	P2_1	P2_0																													

(5) ディップスイッチの値とモータ出力

motor 関数に 100%を設定したとき、ディップスイッチの値と実際に出力される PWM の関係を、下記に示します。

ディップスイッチ				10進数	計算	実際に出力される PWM の割合
P1_3	P1_2	P1_1	P1_0			
0	0	0	0	0	5/20	25%
0	0	0	1	1	6/20	30%
0	0	1	0	2	7/20	35%
0	0	1	1	3	8/20	40%
0	1	0	0	4	9/20	45%
0	1	0	1	5	10/20	50%
0	1	1	0	6	11/20	55%
0	1	1	1	7	12/20	60%
1	0	0	0	8	13/20	65%
1	0	0	1	9	14/20	70%
1	0	1	0	10	15/20	75%
1	0	1	1	11	16/20	80%
1	1	0	0	12	17/20	85%
1	1	0	1	13	18/20	90%
1	1	1	0	14	19/20	95%
1	1	1	1	15	20/20	100%

6.4.22 handle関数(サーボハンドル操作)

```
717 : /*****  
718 : /* サーボハンドル操作 */  
719 : /* 引数 サーボ操作角度 : -90～90 */  
720 : /* -90で左～90度、0でまっすぐ、90で右～90度回転 */  
721 : /*****  
722 : void handle( int angle )  
723 : {  
724 :     /* サーボが左右逆に動く場合は、「-」を「+」に替えてください */  
725 :     trdgrd1 = SERVO_CENTER - angle * HANDLE_STEP;  
726 : }
```

(1) handle関数の使い方

handle 関数の使い方を下記に示します。

```
handle( サーボの角度 );
```

引数は、サーボの角度を設定します。値とサーボの角度の関係を下記に示します。

値	説明
マイナス	指定した角度分、左へサーボを曲げます。
0	サーボが 0 度(まっすぐ)を向きます。0 を設定してサーボがまっすぐ向かない場合、「SERVO_CENTER」の値がずれています。この値を調整してください。
プラス	指定した角度分、右へサーボを曲げます。

プログラム例を、下記に示します。

```
handle( 0 );          0 度  
handle( 30 );         右 30 度  
handle( -45 );        左 45 度
```

(2) プログラムの内容

725 :	trdgrd1 = SERVO_CENTER - angle * HANDLE_STEP;
	① ② ③ ④

①	サーボに接続されている P2_5 端子の PWM の ON 幅を設定するのは、TRDGRB1 です。ただ、直接設定するのではなくバッファレジスタである TRDGRD1 に設定します。
②	0 度のときの値です。
③	handle 関数で指定した角度が代入されている変数です。
④	1 度当たりの増分です。

TRDGRD1 に代入される値の計算例を下記に示します。

※SERVO_CENTER=3749、HANDLE_STEP=22 とします。

• 0 度のとき

$$\begin{aligned} \text{TRDGRD1} &= \text{SERVO_CENTER} - \text{angle} * \text{HANDLE_STEP} \\ &= 3749 - \boxed{0} * 22 \\ &= 3749 \end{aligned}$$

• 30 度のとき

$$\begin{aligned} \text{TRDGRD1} &= \text{SERVO_CENTER} - \text{angle} * \text{HANDLE_STEP} \\ &= 3749 - \boxed{30} * 22 \\ &= 3749 - 660 \\ &= 3089 \end{aligned}$$

• -45 度のとき

$$\begin{aligned} \text{TRDGRD1} &= \text{SERVO_CENTER} - \text{angle} * \text{HANDLE_STEP} \\ &= 3749 - \boxed{(-45)} * 22 \\ &= 3749 - (-990) \\ &= 4739 \end{aligned}$$

6.4.23 スタート

メイン関数です。スタートアップルーチンから呼ばれて最初に実行する C 言語のプログラムはここからです。

```

63 : /*****
64 : /* メインプログラム
65 : ****/
66 : void main( void )
67 : {
68 :     int      i;
69 :
70 :     /* マイコン機能の初期化 */
71 :     init();           /* 初期化 */
72 :     asm(" fset I "); /* 全体の割り込み許可 */
73 :
74 :     /* マイコンカーの状態初期化 */
75 :     handle( 0 );
76 :     motor( 0, 0 );

```

71 行	R8C/38A マイコンの内蔵周辺機能を初期化する関数です。
72 行	全体の割り込みを許可する命令です。 init 関数内でタイマ RB の割り込みを許可していますが、全体の割り込みを許可しなければ割り込みは発生しません。全体の割り込みを許可する命令は、C 言語で記述することができないため、asm 命令を使ってアセンブリ言語で割り込みを許可する命令を記述しています。
75 行 76 行	マイコンカーの状態を初期化します。 handle 関数でサーボの角度を 0 度にします。 motor 関数で左モータの回転を 0%、右モータの回転を 0% にします。

6.4.24 パターン方式について

kit12_38a.c では、パターン方式という方法でプログラムを実行します。

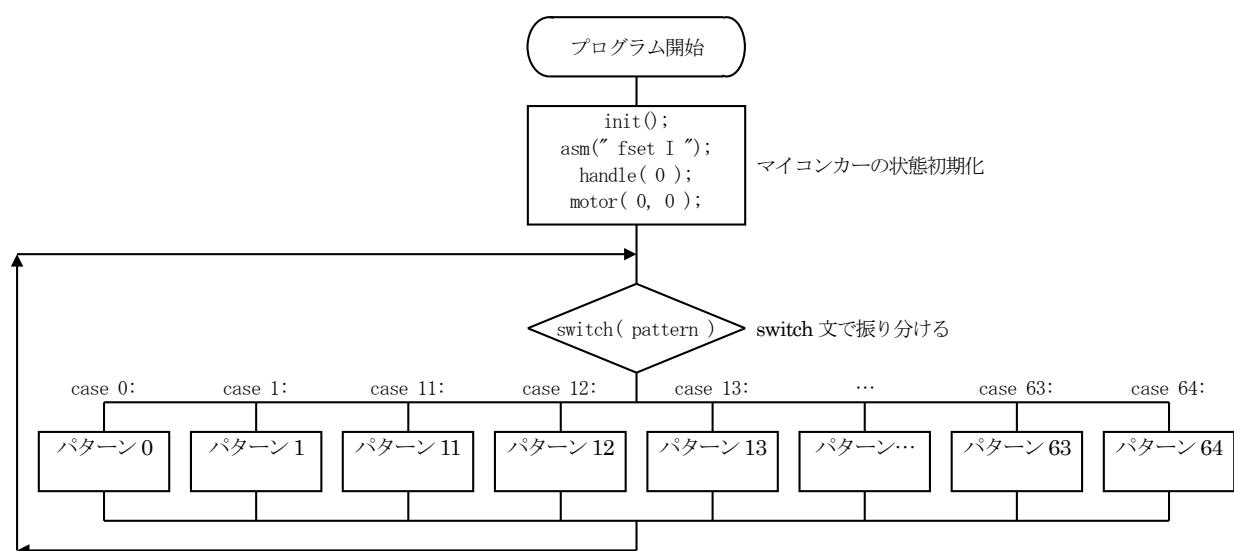
仕組みは、あらかじめプログラムを細かく分けておきます。例えば、「スイッチ入力待ちの処理を行うプログラム」、「スタートバーが開いたかチェックする処理を行うプログラム」、などです。

次に、pattern という変数を作ります。この変数に設定した値により、どのプログラムを実行するか選択します。例えば、パターン変数が 0 のときはスイッチ入力待ちの処理、パターン変数が 1 のときはスタートバーが開いたかチェックする処理...などです。

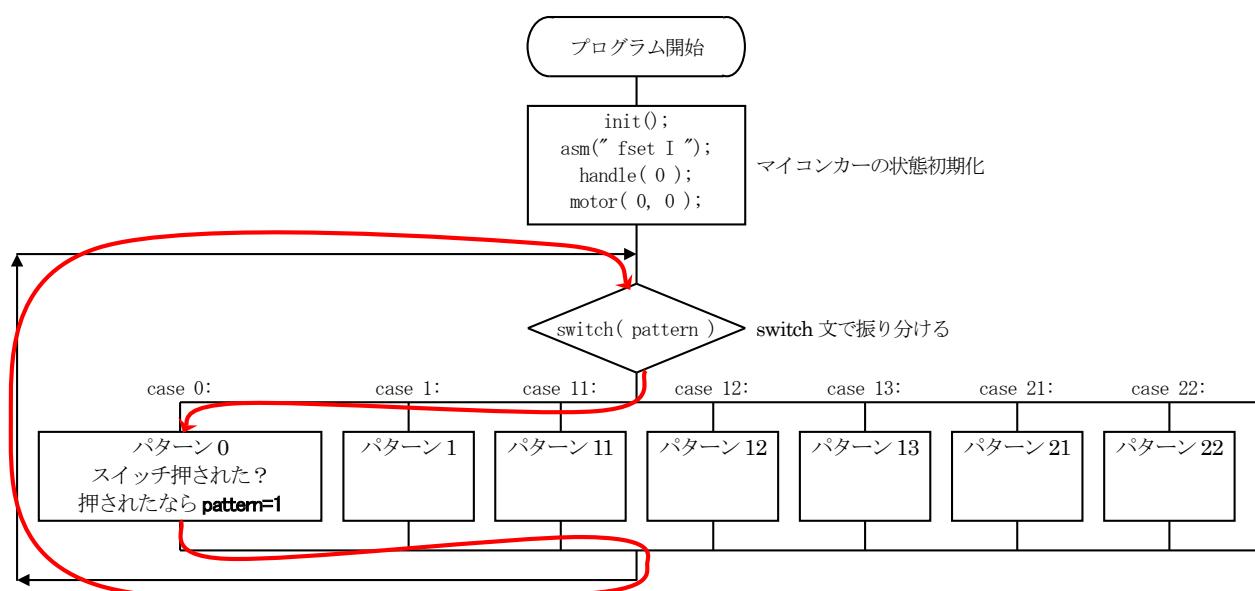
この方式を使うと、パターンごとに処理を分けられるため、プログラムが見やすくなります。パターン方式は、「**プログラムのブロック化**」と言うこともできます。

6.4.25 プログラムの作り方

パターン方式を C 言語で行うには、switch 文で分岐させます。フローチャートを下図に示します。

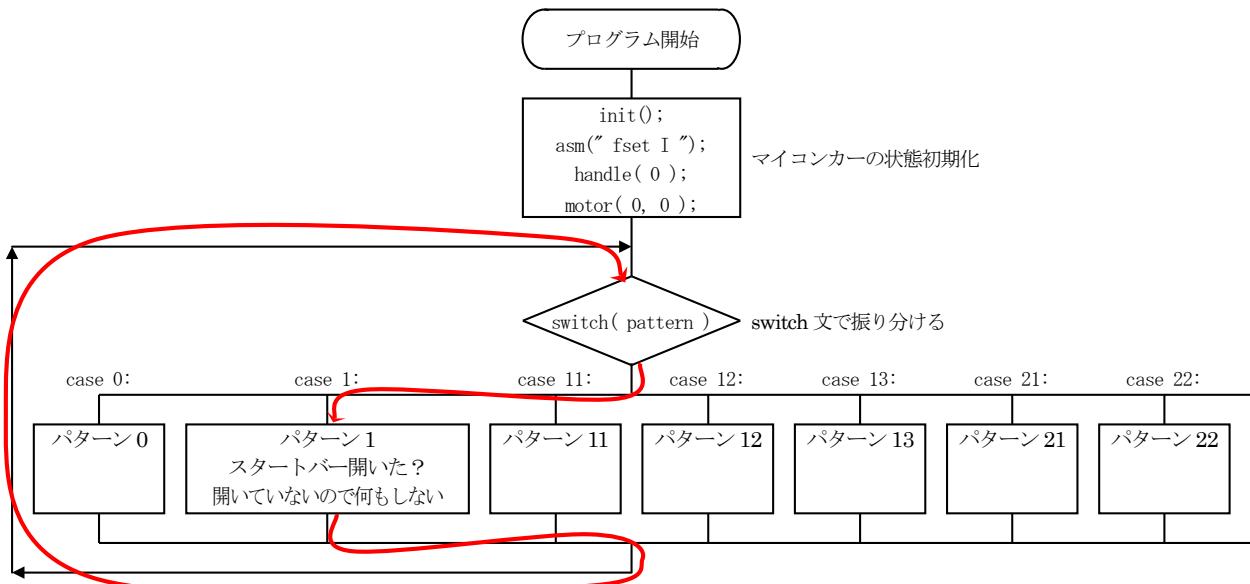


起動時、pattern 変数は 0 です。switch 文により case 0 部分のプログラム「パターン 0 プログラム」を実行し続けます。後ほど説明しますが、パターン 0 はスイッチ入力待ちです。スイッチが押されると、「pattern=1」が実行されます。フローチャートを下図に示します。



6. プログラム解説「kit12_38a.c」

次に switch 文を実行したとき、pattern 変数の値が 1 になっているので、case 1 部分のプログラムが実行されます。本プログラムでは、switch(pattern)の case 1 のプログラムを、「パターン 1 のプログラム」と言います。パターン 1 は、スタートバーが開いたかどうかチェックする部分です。フローチャートを下図に示します。



このように、プログラムをブロック化します。ブロック化したプログラムでは、「スタートスイッチが押されたか」、「スタートバーが開いたか」など簡単なチェックを行い、条件を満たすとパターン番号(pattern 変数の値)を変えます。

プログラムを下記に示します。通常の switch～case 文です。

```

switch( pattern ) {
    case 0:
        /* pattern=0 の処理 */
        break;
    case 1:
        /* pattern=1 の処理 */
        break;
    default:
        /* どれでもないなら */
        break;
}

```

pattern と各 case 文の定数を比較して、一致したらその case 位置にジャンプして処理を行う。
その処理は、break 文に出会うか switch 文の終端に出会うと終了する。
どの case 文の定数にも当てはまらない場合は、default 文を実行。default 文も無い場合は何も実行しない。

6.4.26 パターンの内容

kit12_38a.c のパターン番号と、プログラムの処理内容、パターンが変わる条件を下記に示します。

現在のパターン	処理内容	パターンが変わる条件
0	スイッチ入力待ち	•スイッチを押したらパターン 1 へ
1	スタートバーが開いたか チェック	•スタートバーが開いたことを検出したらパターン 11 へ
11	通常トレース	•右大曲げになつたらパターン 12 へ •左大曲げになつたらパターン 13 へ •クロスラインを検出したらパターン 21 へ •右ハーフラインを検出したらパターン 51 へ •左ハーフラインを検出したらパターン 61 へ
12	右へ大曲げの終わりの チェック	•右大曲げが終わったらパターン 11 へ •クロスラインを検出したらパターン 21 へ •右ハーフラインを検出したらパターン 51 へ •左ハーフラインを検出したらパターン 61 へ
13	左へ大曲げの終わりの チェック	•左大曲げが終わったらパターン 11 へ •クロスラインを検出したらパターン 21 へ •右ハーフラインを検出したらパターン 51 へ •左ハーフラインを検出したらパターン 61 へ
21	クロスライン 検出時の処理	•サーボ、スピードの設定を終えたらパターン 22 へ
22	クロスラインを読み飛ばす	•100msたつたらパターン 23 へ
23	クロスライン後のトレース、 クランク検出	•左クランクを見つけたらパターン 31 へ •右クランクを見つけたらパターン 41 へ
31	左クランクリア処理 安定するまで少し待つ	•200msたつたらパターン 32 へ
32	左クランクリア処理 曲げ終わりのチェック	•左クランクをクリアしたらパターン 11 へ
41	右クランクリア処理 安定するまで少し待つ	•200msたつたらパターン 42 へ
42	右クランクリア処理 曲げ終わりのチェック	•右クランクをクリアしたらパターン 11 へ
51	右ハーフライン 検出時の処理	•サーボ、スピードの設定を終えたらパターン 52 へ
52	右ハーフラインを 読み飛ばす	•100msたつたらパターン 53 へ
53	右ハーフライン後の トレース、レーンチェンジ	•中心線が無くなつたなら右へハンドルを曲げてパターン 54 へ
54	右レーンチェンジ終了の チェック	•新しい中心線がセンサの中心に来たならパターン 11 へ
61	左ハーフライン検出時の 処理	•サーボ、スピードの設定を終えたらパターン 62 へ
62	左ハーフラインを読み飛 ばす	•100msたつたらパターン 63 へ
63	左ハーフライン後のトレー ス、レーンチェンジ	•中心線が無くなつたなら左へハンドルを曲げてパターン 64 へ
64	左レーンチェンジ終了の チェック	•新しい中心線がセンサの中心に来たならパターン 11 へ
現在のパターン	処理内容	内容

6.4.27 パターン方式の最初while、switch部分

```

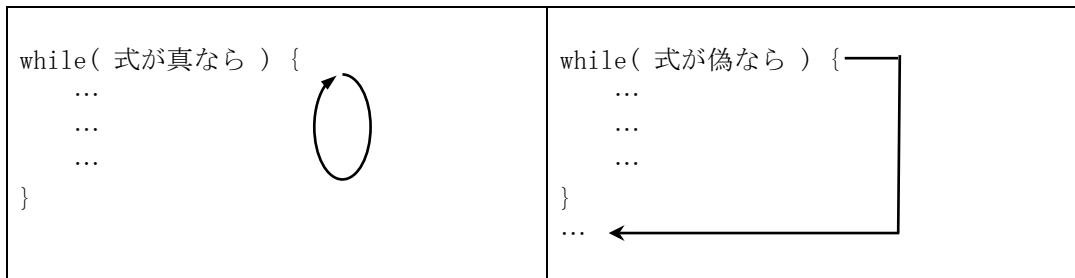
78 :     while( 1 ) { —————
79 :         switch( pattern ) { —————
105 :             case 0:
                  パターン0時の処理
119 :                 break;
120 :
121 :             case 1:
                  パターン1時の処理
137 :                 break;
                      対      対
                      それぞれのパターン処理
473 :         default:
474 :             /* どれでもない場合は待機状態に戻す */
475 :             pattern = 0;
476 :             break;
477 :     } —————
478 : } —————

```

78 行の「while(1) {」と 478 行の「}」が対、79 行の「switch(pattern) {」と 477 行の「}」が対となります。

通常、中カッコ「{」があるとそれと対になる中カッコ閉じ「}」が来るまで、くぐられた中は 4 文字右にずらして分かりやすくなります。このプログラムも 4 文字右にずらしています。しかし、while 部分と switch 部分は同列に書いています。これは、プログラムが複雑になった場合に画面の右端を超えて 2 行になり、見づらくなるのを防ぐためです。人々、人間に分かりやすくするために列をずらしているわけですから、コンパイル上はまったく問題ありません。どうしても気になってしまふ場合は、79~477 行を 4 文字分、右にずらすと良いでしょう。

「while(式)」は、式の中が「真」なら{ }でくくった命令を繰り返し、「偽」なら{ }でくくった次の命令から実行するという制御文です。



「真」、「偽」とは、

	説明	例
真	正しいこと、0 以外	3<5 3==3 1 2 3 -1 -2 -3
偽	正しくないこと、0	5<3 3==6 0

と定義されています。プログラムは、「while(1)」となっています。1は常に「真」ですので while の{ }内を無限に繰り返します。Windows プログラムなどで無限ループを行うと、アプリケーションが終了できなくなり困りますが、今回マイコンカーを動かすためだけのプログラムなのでこれで構いません。マイコンカーがゴール(または脱輪)すれば、人が取り上げてスイッチを切ればいいのです。逆にマイコンは、適切に終了処理をせずにプログラムを終わらせてしまうと、プログラムが書かれていない領域にまで行ってしまい暴走してしまいます。マイコンは、何もないことを繰り返して(無限ループ)終了させないか、スリープモードと呼ばれる低消費電力モードに移り動作を停止させて次に復帰するタイミングを待つののが普通です。

6.4.28 パターン 0:スイッチ入力待ち

パターン 0 は、スイッチが押されたかチェックする部分です。チェック中、動作しているのか止まっているのか分かりません。そのため、モータドライブ基板の LED2 と LED3 を交互に光らせます。

まず、スイッチ検出部分です。pushsw_get 関数でスイッチをチェックします。押されると 1 が返ってくるので、カッコの中が実行されパターンを 1 にします。

```

105 :     case 0:
106 :         /* スイッチ入力待ち */
107 :         if( pushsw_get() ) {           ←スイッチが押されたなら(戻り値が 0 以外なら)
108 :             pattern = 1;            ←パターンを1に
109 :             cnt1 = 0;              ←cnt1を0に
110 :             break;                ←switch 文を終了
111 :         }

```

	<p>プッシュスイッチが押されたら、カッコの中のプログラム(108~110 行)を実行します。押されていなければ実行しません。</p> <p>if 文は比較です。次のプログラムは、pushsw_get 関数の戻り値が 1 かどうか比較しています。</p> <pre>if(pushsw_get() == 1) { pushsw_get 関数の戻り値が 1 なら</pre> <p>今回のプログラムは、次のようになっています。</p> <pre>if(pushsw_get()) {</pre> <p>比較していません。C 言語では次のような意味になります。</p> <p>107 行</p> <pre>if(値) { 値が0以外なら成り立つと判断し、この部分を実行 } else { 値が0なら成り立たないと判断し、この部分を実行 }</pre> <p>pushsw_get 関数の戻り値は、スイッチが押されたら 1、押されていなければ 0 なので、押されたら下記のように動作します。</p> <pre>if(1) { pushsw_get() の部分が 1 になります 値は 0 以外なので、カッコの中を実行 }</pre> <p>108 行</p> <p>pattern 変数に 1 を代入しています。次に switch-case 文を実行するときは「case 1:」部分を実行します。</p>
--	---

6. プログラム解説「kit12_38a.c」

次にLEDを点滅させるプログラムを追加します。点滅は、0.1秒間LED3が点灯、次の0.1秒間LED2が点灯、それを繰り返す処理にします。

```

112 :         if( cnt1 < 100 ) {           ←cnt1 が 0～99 か
113 :             led_out( 0x1 );        ←なら LED3 のみ点灯
114 :         } else if( cnt1 < 200 ) {   ←cnt1 が 100～199 か
115 :             led_out( 0x2 );        ←なら LED2 のみ点灯
116 :         } else {                ←それ以外なら(200 以上)
117 :             cnt1 = 0;            ←cnt1 を0にする
118 :         }

```

通常の変数、例えば pattern 変数は、一度設定すると次に変更するまで値は変わりません。kit12_38a.c では、**cnt0 変数と cnt1 変数だけは例外です**。cnt0 と cnt1 は割り込み関数内で 1msごとに+1しています。そのため、この変数を使って時間を計ることができます。

119 行の break 文は、case 0 を終えるための break です。

```

78 :     while( 1 ) { ←
79 :         switch( pattern ) {
105 :             case 0:
中略
119 :             break; ←①
中略
477 :         }
478 :     } ←②

```

①	119 行の break 文で、switch-case 文の中カッコ閉じである 477 行の次の行へ移ります。
②	次に 478 行を実行しますが、ここにある中カッコ閉じは 78 行にある while 文の中カッコ閉じなので、78 行へ戻ります。
③	79 行の switch-case 文を実行して、pattern 変数に応じて、case 文に移ります。

※cnt1 変数を使わなかった場合

もし、cnt1 を使わずに、timer 関数を使ったらどうなるでしょうか。

```
if( pushsw_get() ) {
    pattern = 1;
    cnt1 = 0;
    break;
}
timer( 100 );           ←この行で、100ms 止まってしまう！！
led_out( 0x1 );
timer( 100 );           ←この行で、100ms 止まってしまう！！
led_out( 0x2 );
break;
```

プログラムがシンプルになりました。こちらの方がいい気がします。しかし、timer 関数は**待つこと以外は何もしません**。そのため、timer 関数実行中にプッシュスイッチを押して離した場合、pushsw_get 関数が実行されたときにはすでにスイッチは押されていないので検出もれがおこります。このプログラムの timer 関数の実行は 0.1 秒なので、かなり素早くスイッチを押さなければ検出もれは起こりませんが、もしこれが何秒間というように長い時間になるとスイッチをチェックしない時間が長すぎ、検出もれをおこします。このため、**cnt1 変数を使って時間をチェックしながら、スイッチのチェックも行っています**。

6.4.29 パターン 1: スタートバーが開いたかチェック

パターン 1 は、スタートバーが開いたかどうかチェックする部分です。チェック中、本当に動作しているのか止まっているのか分かりません。そのため、LED2 と LED3 を交互に光らせます。

まず、スタートバーの開閉を検出する部分です。

```
121 :     case 1:
122 :         /* スタートバーが開いたかチェック */
123 :         if( !startbar_get() ) {
124 :             /* スタート！！ */
125 :             led_out( 0x0 );
126 :             pattern = 11;
127 :             cnt1 = 0;
128 :             break;
129 : }
```

6. プログラム解説「kit12_38a.c」

	<p>スタートバーが開いたら、カッコの中のプログラム(125～128 行)を実行します。閉じたままだと実行しません。</p> <p>startbar_get 関数は、スタートバーがあれば(センサに反応があれば)1 が、スタートバーがなければ(センサに反応がなければ)0 が戻ってくる関数です。</p> <p>先ほどの pushsw_get 関数と同じようにプログラムしてみます。</p> <pre>if(startbar_get()) { [スタートバーがあればこの部分を実行] }</pre> <p>123 行</p> <p>スタートバーがあればカッコの中を実行してしまいます。本当は、スタートバーがなくなればカッコの中を実行したいです。そのため、スタートバーがなければカッコの中を実行するように「!」を付けて、否定します。「!」をつけると、「でないなら」となります。</p> <pre>if(!値) { [値が0以外でないならこの部分を実行→0ならこの部分を実行] } else { [値が0でないならこの部分を実行→0以外ならこの部分を実行] }</pre> <p>よって、次のような動作になります。</p> <pre>if(!startbar_get()) { [スタートバーがなければ、この部分を実行] }</pre>
126 行	pattern 変数に 11 を代入しています。次に switch-case 文を実行するときは「case 11:」部分を実行します。
128 行	break 文がくると、switch-case 文の終わりの中カッコ閉じに移ります。

次に、LED を点滅させるプログラムを実行します。点滅は、0.05 秒間 LED3 が点灯、次の 0.05 秒間 LED2 が点灯、それを繰り返す処理にします。パターン 0 より、速く点滅させて、スタートバーが開くことを待っていることを分かりやすくしています。

130 :	if(cnt1 < 50) {	←cnt1 が 0～49 か
131 :	led_out(0x1);	←なら LED3 のみ点灯
132 :	} else if(cnt1 < 100) {	←cnt1 が 50～99 か
133 :	led_out(0x2);	←なら LED2 のみ点灯
134 :	} else {	←それ以外なら(100 以上)
135 :	cnt1 = 0;	←cnt1 を 0 にする
136 :	}	
137 :	break;	

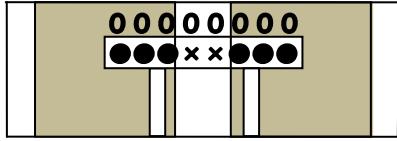
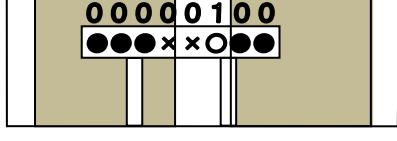
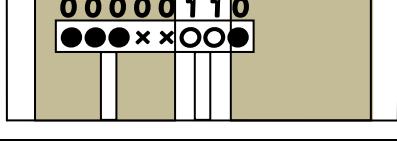
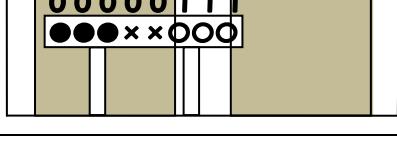
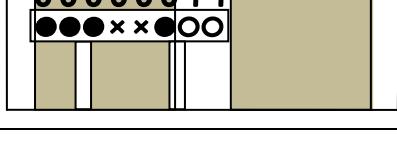
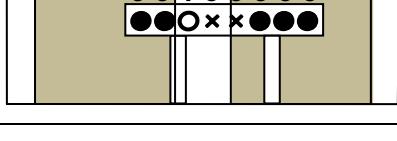
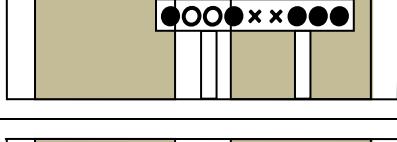
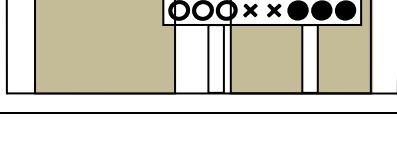
6.4.30 パターン 11:通常トレース

パターン 11 は、センサを読み込んで、左モータ、右モータ、サーボを制御する状態です。

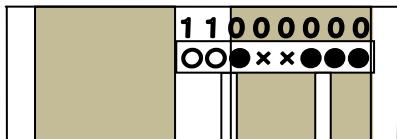
まず、想定されるセンサの状態を考えます。センサは 8 個ありますが、すべて使うとセンサの検出状態が多くプログラムが複雑になることが考えられます。そこで「MASK3_3」でマスクをかけて、右 3 個、左 3 個、合計 6 個のセンサでコースの状態を検出するようにします。

次に、そのときの左モータの PWM 値、右モータの PWM 値、ハンドル切れ角を考えます。考え方は、センサが中心のときはハンドルをまっすぐにしてスピードを上げます。センサのずれが大きくなればなるほど、ハンドルの曲げ角を大きくして、左モータ、右モータのスピードを落とします。

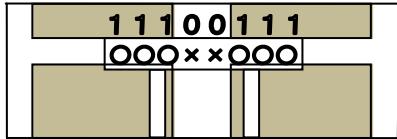
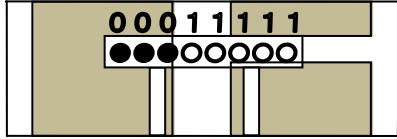
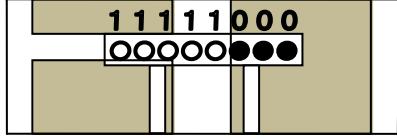
kit12_38a.c では、下記のようにしました。

	コースとセンサの状態	センサを読み込んだときの値	16進数	ハンドル角度	左モータ PWM	右モータ PWM
1		00000000	0x00	0	100	100
2		00000100	0x04	5	100	100
3		00000110	0x06	10	80	67
4		00000111	0x07	15	50	38
5		00000011	0x03	25	30	19
6		00100000	0x20	-5	100	100
7		01100000	0x60	-10	67	80
8		11100000	0xe0	-15	38	50

6. プログラム解説「kit12_38a.c」

	コースとセンサの状態	センサを読み込んだときの値	16進数	ハンドル角度	左モータ PWM	右モータ PWM
9		11000000	0xc0	-25	19	30

また、マイコンカーのコースにはクロスラインや右ハーフライン、左ハーフラインがあります。それぞれ、検出する関数があるのでそれを使います。

	コースとセンサの状態	コースの特徴、処理	チェックする関数名
10	 センサは 6 個使用	横線 (クロスライン) ↓ 検出したならクランク処理へ (パターン 21)	check_crossline
11	 センサは 8 個使用	中心から右側のみの横線 (右ハーフライン) ↓ 検出したなら右ハーフライン 処理へ(パターン 51)	check_rightline
12	 センサは 8 個使用	中心から左側のみの横線 (左ハーフライン) ↓ 検出したなら左ハーフライン 処理へ(パターン 61)	check_leftline

この表に基づいて、プログラムを書いていきます。

(1) センサ読み込み

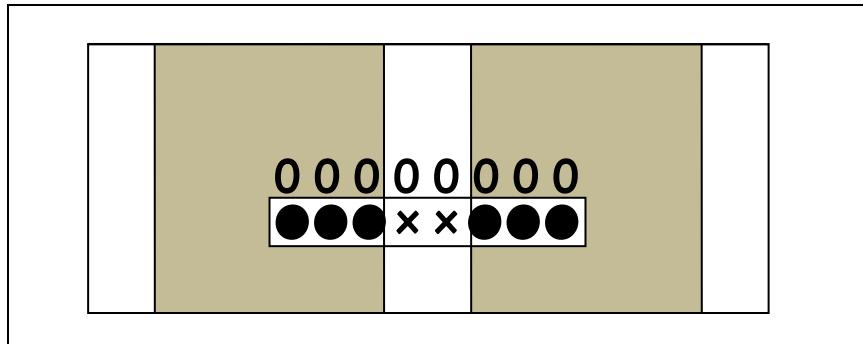
```
153 :         switch( sensor_inp(MASK3_3) ) {
```

センサの状態を読み込みます。右3個、左3個のセンサを読み込むので MASK3_3 を使います。センサの状態によりプログラムを分岐させる部分は、switch-case 文を使います。

(2) 直進

```
154 :             case 0x00:  
155 :                 /* センタ→まっすぐ */  
156 :                 handle( 0 );  
157 :                 motor( 100 ,100 );  
158 :                 break;
```

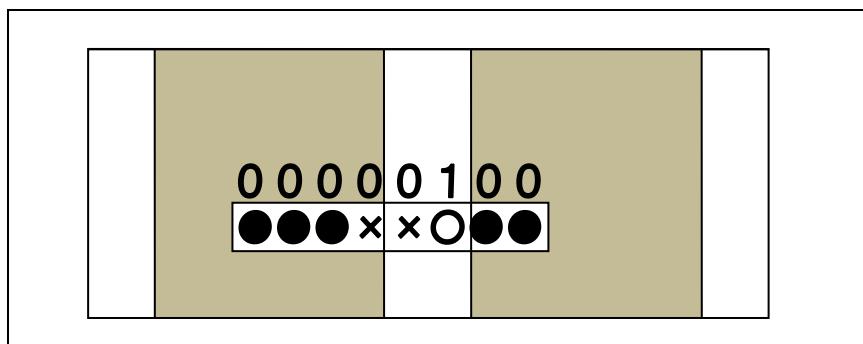
センサが「0x00」の状態です。この状態は下図のように、まっすぐ進んでいる状態です。サーボ角度 0 度、左モータ 100%、右モータ 100%で進みます。



(3) 左寄り

```
160 :             case 0x04:  
161 :                 /* 微妙に左寄り→右へ微曲げ */  
162 :                 handle( 5 );  
163 :                 motor( 100 ,100 );  
164 :                 break;
```

センサが「0x04」の状態です。この状態は下図のように、マイコンカーが微妙に左に寄っている状態です。サーボを右に 5 度、左モータ 100%、右モータ 100%で進み、中心に寄るようにします。



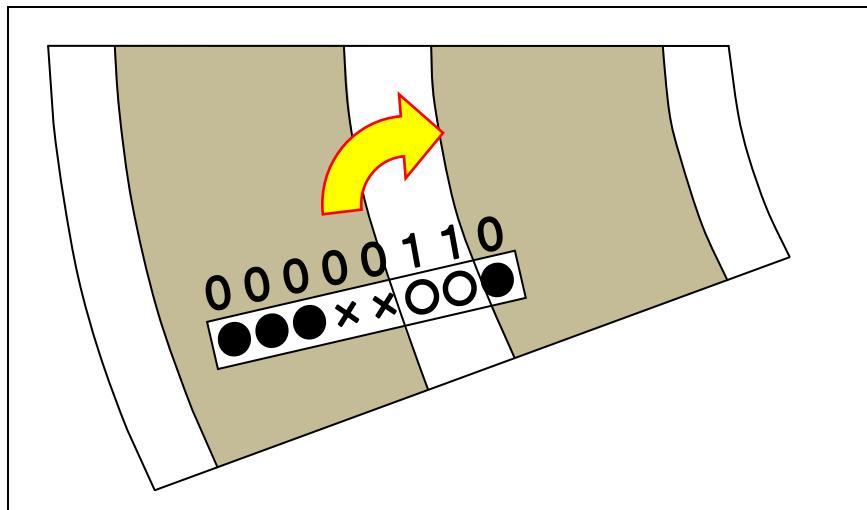
(4) 少し左寄り

```

166 :         case 0x06:
167 :             /* 少し左寄り→右へ小曲げ */
168 :             handle( 10 );
169 :             motor( 80 , 67 );
170 :             break;

```

センサが「0x06」の状態です。この状態は下図のように、マイコンカーが少し左に寄っている状態です。サーボを右に 10 度、左モータ 80%、右モータ 67%で進み、減速しながら中心に寄るようにします。



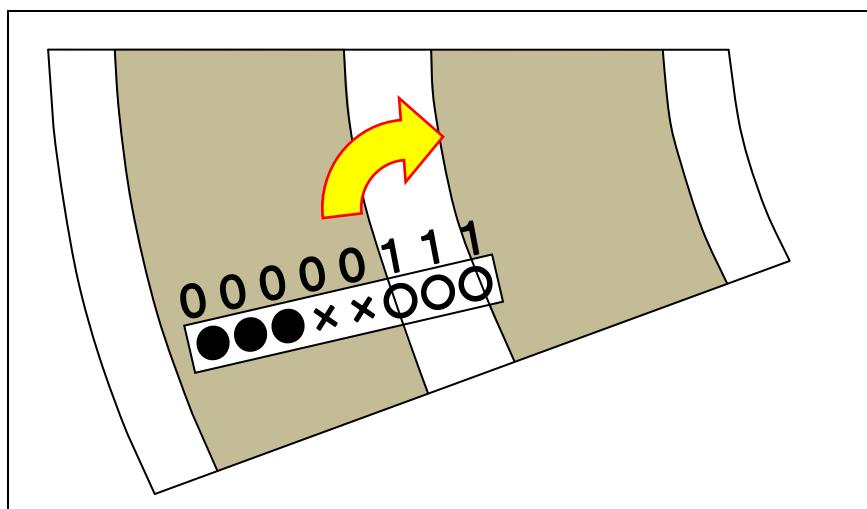
(5) 中くらい左寄り

```

172 :         case 0x07:
173 :             /* 中くらい左寄り→右へ中曲げ */
174 :             handle( 15 );
175 :             motor( 50 , 38 );
176 :             break;

```

センサが「0x07」の状態です。この状態は下図のように、マイコンカーが中くらい左に寄っている状態です。サーボを右に 15 度、左モータ 50%、右モータ 38%で進み、減速しながら中心に寄るようにします。



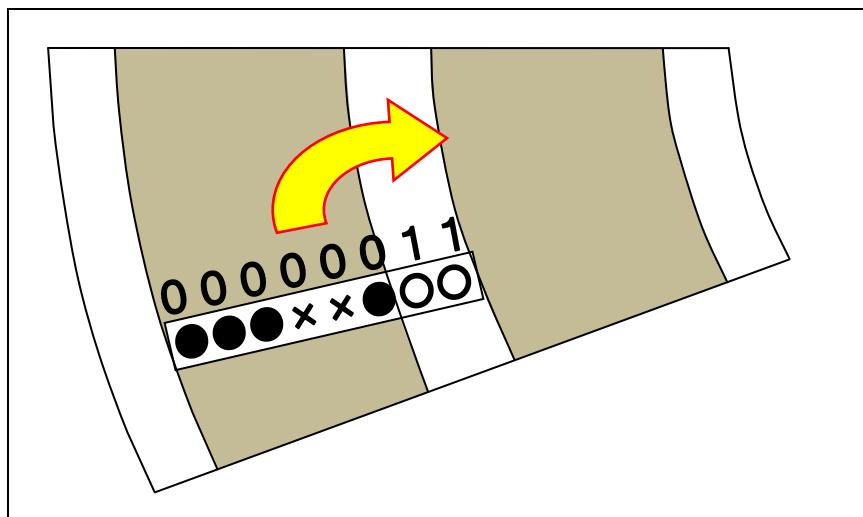
(6) 大きく左寄り

```

178 :         case 0x03:
179 :             /* 大きく左寄り→右へ大曲げ */
180 :             handle( 25 );
181 :             motor( 30 , 19 );
183 :             break;
    
```

※本当は 182 行がありますが、ここでは省略して説明します。後述します。

センサが「0x03」の状態です。この状態は下図のように、マイコンカーが大きく左に寄っている状態です。サーボを右に 25 度、左モータ 30%、右モータ 19%で進み、かなり減速しながら中心に寄るようにします。

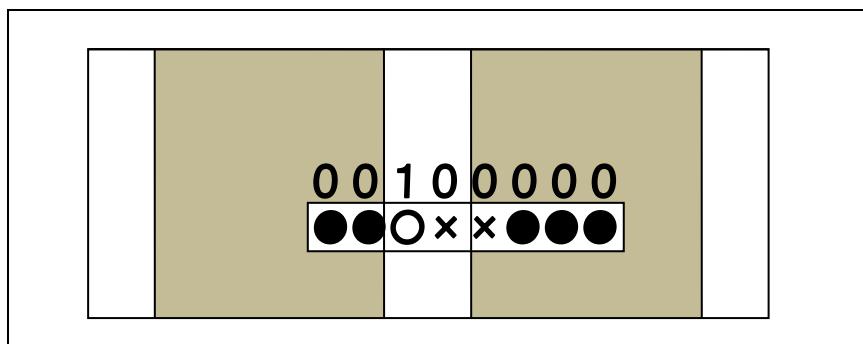


(7) 微妙に右寄り

```

185 :         case 0x20:
186 :             /* 微妙に右寄り→左へ微曲げ */
187 :             handle( -5 );
188 :             motor( 100 , 100 );
189 :             break;
    
```

センサが「0x20」の状態です。この状態は下図のように、マイコンカーが微妙に右に寄っている状態です。サーボを左に 5 度、左モータ 100%、右モータ 100%で進み、中心に寄るようにします。



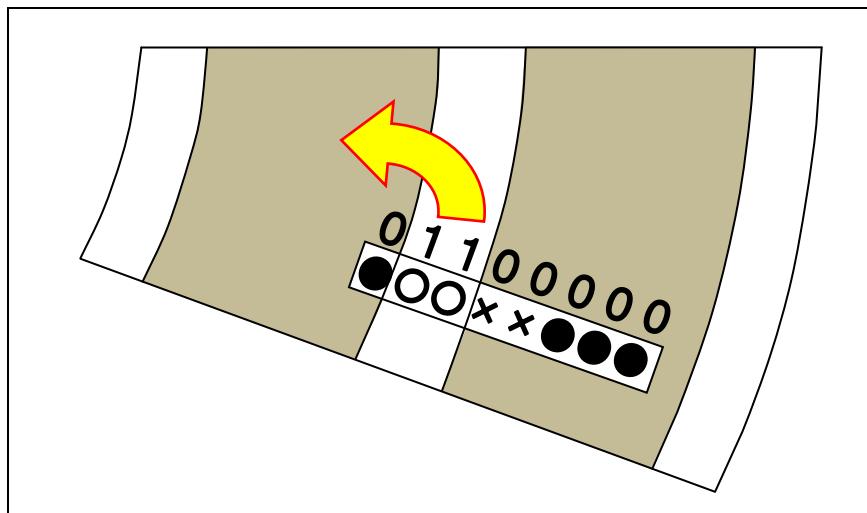
(8) 少し右寄り

```

191 :         case 0x60:
192 :             /* 少し右寄り→左へ小曲げ */
193 :             handle( -10 );
194 :             motor( 67 , 80 );
195 :             break;

```

センサが「0x60」の状態です。この状態は下図のように、マイコンカーが少し右に寄っている状態です。サーボを左に 10 度、左モータ 67%、右モータ 80%で進み、減速しながら中心に寄るようにします。



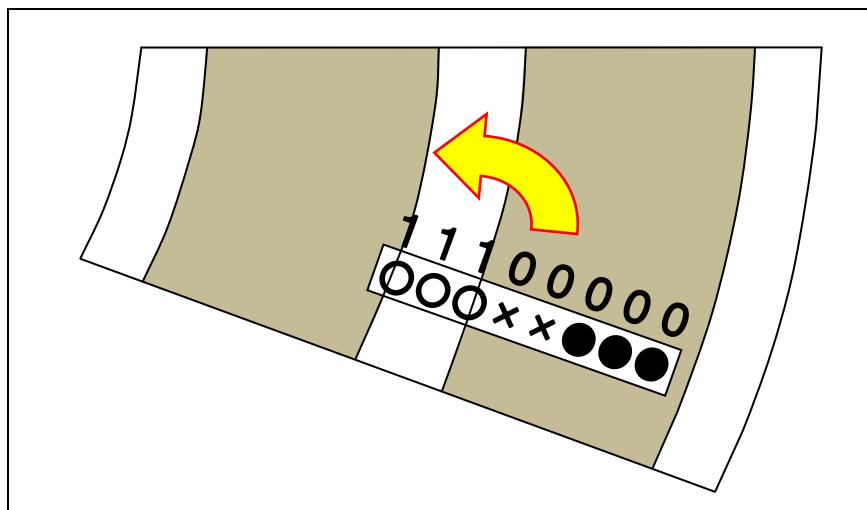
(9) 中くらい右寄り

```

197 :         case 0xe0:
198 :             /* 中くらい右寄り→左へ中曲げ */
199 :             handle( -15 );
200 :             motor( 38 , 50 );
201 :             break;

```

センサが「0xe0」の状態です。この状態は下図のように、マイコンカーが少し右に寄っている状態です。サーボを左に 15 度、左モータ 38%、右モータ 50%で進み、減速しながら中心に寄るようにします。

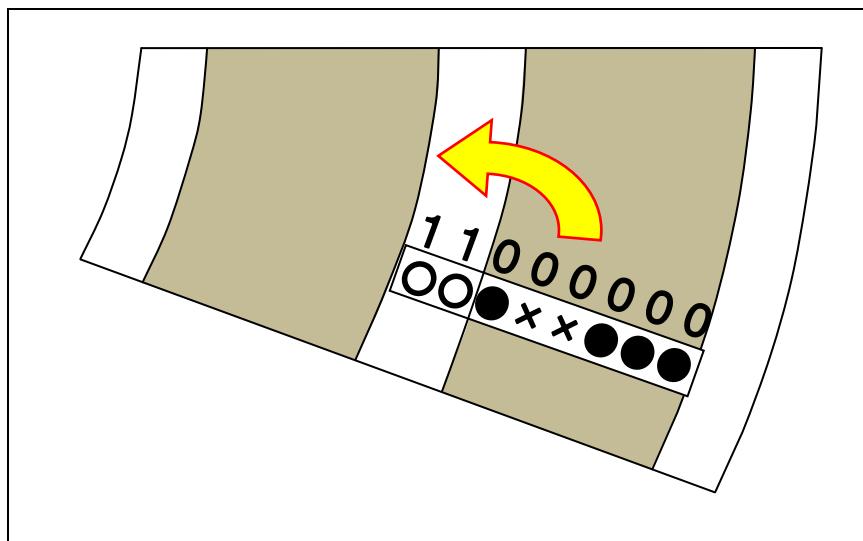


(10) 大きく右寄り

```
203 :         case 0xc0:  
204 :             /* 大きく右寄り→左へ大曲げ */  
205 :             handle( -25 );  
206 :             motor( 19 , 30 );  
208 :             break;
```

※本当は 207 行がありますが、ここでは省略して説明します。後述します。

センサが「0xc0」の状態です。この状態は下図のように、マイコンカーが大きく右に寄っている状態です。サーボを左に 25 度、左モータ 19%、右モータ 30%で進み、かなり減速しながら中心に寄るようにします。



(11) クロスラインチェック

```

141 :         if( check_crossline() ) {      /* クロスラインチェック */
142 :             pattern = 21;
143 :             break;
144 :         }

```

check_crossline 関数の戻り値は、0 でクロスラインではない、1 でクロスライン検出状態となります。クロスラインを検出したらパターンを 21 にして、break 文で switch-case 文を終わります。クロスラインチェックは重要なので、通常トレースプログラムより前に実行するようにします。

(12) 右ハーフライン

```

145 :         if( check_rightline() ) {      /* 右ハーフラインチェック */
146 :             pattern = 51;
147 :             break;
148 :         }

```

check_rightline 関数の戻り値は、0 で右ハーフラインではない、1 で右ハーフライン検出状態となります。右ハーフラインを検出したらパターンを 51 にして、break 文で switch-case 文を終わります。右ハーフラインチェックは重要なので、通常トレースプログラムより前に実行するようにします。

(13) 左ハーフライン

```

149 :         if( check_leftline() ) {      /* 左ハーフラインチェック */
150 :             pattern = 61;
151 :             break;
152 :         }

```

check_leftline 関数の戻り値は、0 で左ハーフラインではない、1 で左ハーフライン検出状態となります。左ハーフラインを検出したらパターンを 61 にして、break 文で switch-case 文を終わります。左ハーフラインチェックは重要なので、通常トレースプログラムより前に実行するようにします。

(14) それ以外

```

210 :         default:
211 :             break;

```

今までのパターン以外のとき、この default 部分へジャンプしてきます。何もしません。

(15) break文で抜ける位置

break 文は、switch 文または for、while、do～while のループから脱出させるための文です。**多重ループの中で用いられると、break 文が存在するループをひとつだけ打ち切り、すぐ外側のループに処理を移します。「ひとつだけ打ち切り」が重要です。**

パターン 11 の break で抜ける位置は下記のようになります。抜ける位置が違いますので、どのループの中で break 文が使われているか見極めて判断してください。

```

while( 1 ) {
    switch( pattern ) {

        中略

        case 11: ← switch( pattern )に対応する case
            /* 通常トレース */
            if( check_crossline() ) {
                pattern = 21;
                break; ← switch( pattern )を抜ける break、[1]へ
            }
            if( check_rightline() ) {
                pattern = 51;
                break; ← switch( pattern )を抜ける break、[1]へ
            }
            if( check_leftline() ) {
                pattern = 61;
                break; ← switch( pattern )を抜ける break、[1]へ
            }
            switch( sensor_inp(MASK3_3) ) {
                case 0x00: ← switch( sensor_inp(MASK3_3) )に対応する case
                    /* センタ→まっすぐ */
                    handle( 0 );
                    speed( 100 ,100 );
                    break; ← switch( sensor_inp(MASK3_3) )を抜ける break、[2]へ

                case 0x04: ← switch( sensor_inp(MASK3_3) )に対応する case
                    /* 微妙に左寄り→右へ微曲げ */
                    handle( 5 );
                    speed( 100 ,100 );
                    break; ← switch( sensor_inp(MASK3_3) )を抜ける break、[2]へ

                中略

                default:
                    break; ← switch( sensor_inp(MASK3_3) )に対応する default
                } [2]
                break; ← switch( pattern )を抜ける break、[1]へ

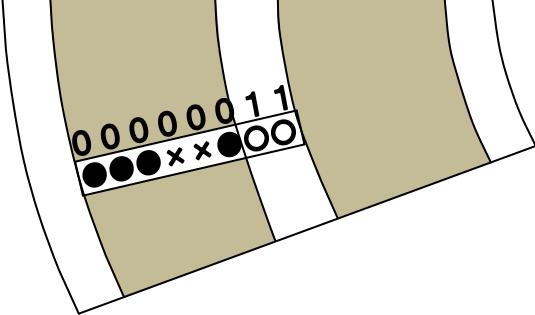
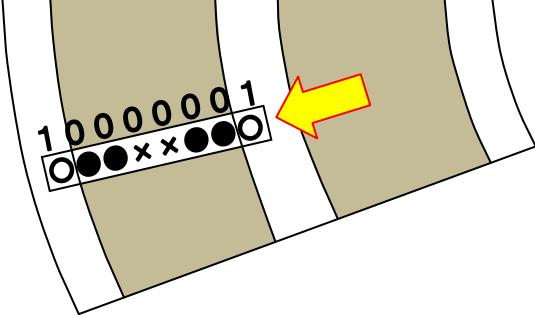
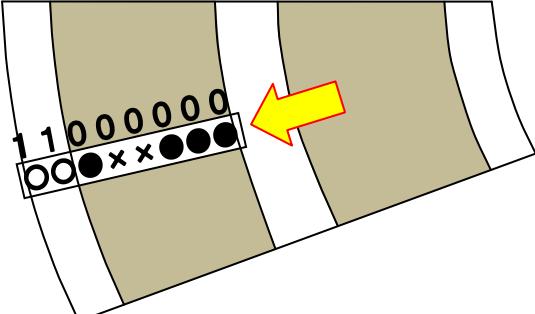
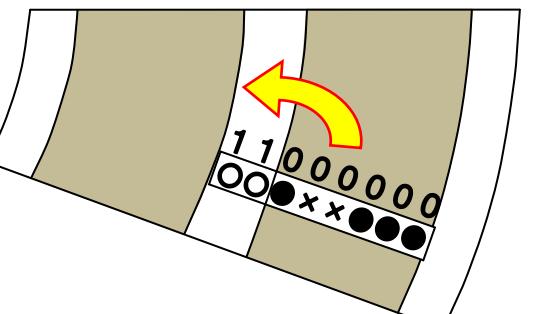
                中略

            } [1]
        }
    }
}

```

6.4.31 パターン 12: 右へ大曲げの終わりのチェック

センサ状態 0x03 は、一番大きく左に寄ったときのセンサ状態です。そのため、これ以上カーブで脹らんだ場合、下図のようになる可能性があります。

1		<p>大きく左に寄ったときのセンサの状態です。 センサは、「0000 0011」です。 この状態になったなら、下記プログラムが実行されます。</p> <pre>handle(25); motor(30 , 19);</pre> <p>よって、右に 25 度ハンドルを切って左モータ 30%、右モータ 19%で回します。</p>
2		<p>さらに左に寄りました。 センサは、「1000 0001」です。 この状態のとき、実行するプログラムの記述はありません。この場合、前の状態を保持します。 前の状態は「0000 0011」なので、このセンサ値のモータ回転数、ハンドル角度になります。</p>
3		<p>さらに左に寄りました。 センサは、「1100 0000」です。</p>
4		<p>センサ状態「1100 0000」は、本プログラムでは、左図のように、マイコンカーが右に寄っているので、左にハンドルを曲げる状態を想定しています。この状態になったなら、下記プログラムが実行されます。</p> <pre>handle(-25); motor(19, 30);</pre> <p>よって、左に 25 度ハンドルを切って左モータ 19%、右モータ 30%で回します。</p>

5		実際は、左に大きく寄っているので、この状態で左にハンドルを曲げると、脱輪してしまいます。
---	--	--

そこで、右に大曲げしたら、あるセンサ状態に戻るまで右に大曲げし続けるようにします。この“あるセンサ状態”を判定する部分を、パターン 12 に作ります。

パターン 11 の case 0x03 部分

```

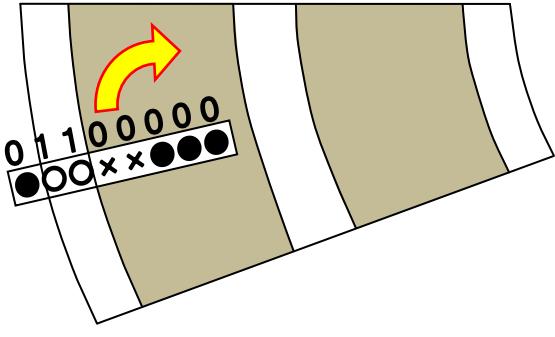
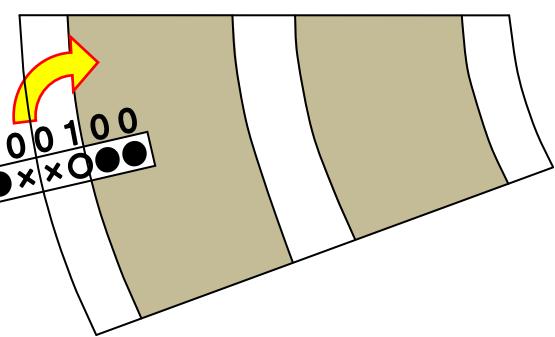
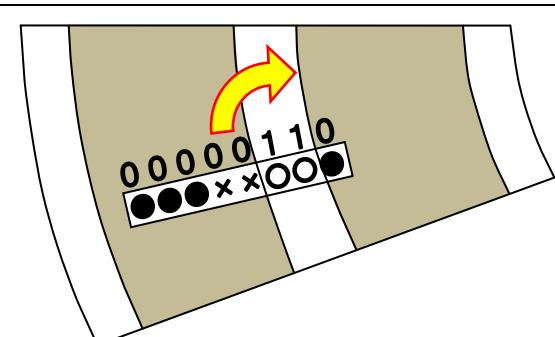
178 :           case 0x03:
179 :             /* 大きく左寄り→右へ大曲げ */
180 :             handle( 25 );
181 :             speed( 30 ,19 );
182 :             pattern = 12;  ←追加 パターン 12へ移る
183 :             break;

```

6		センサが「0000 0011」になったなら、パターン 12 に移ります。 今回パターン 12 では、この 1 つ内側のセンサ状態である「0000 0110」になったなら、パターン 11 に戻るプログラムを考えました。 この考え方で良いか検証してみます。
---	--	--

7		センサが「1100 0000」になりました。「0000 0110」ではないので、まだ右に曲げ続けます。 先ほどは右に寄っていると勘違いしましたが、今回はパターン 12 なので勘違いしません。
---	--	--

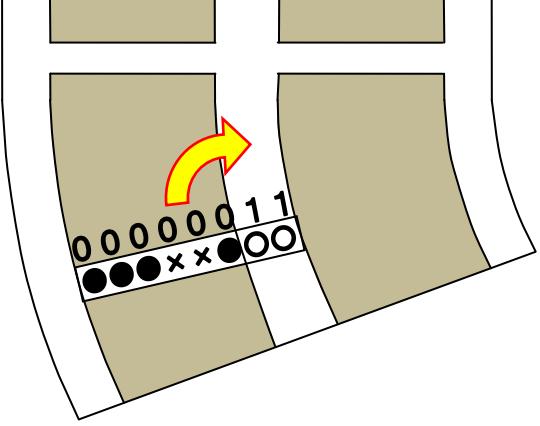
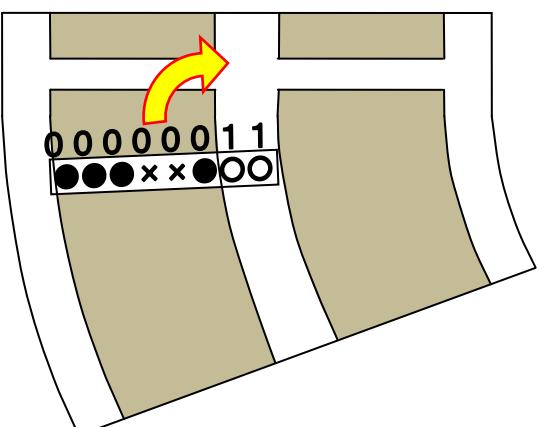
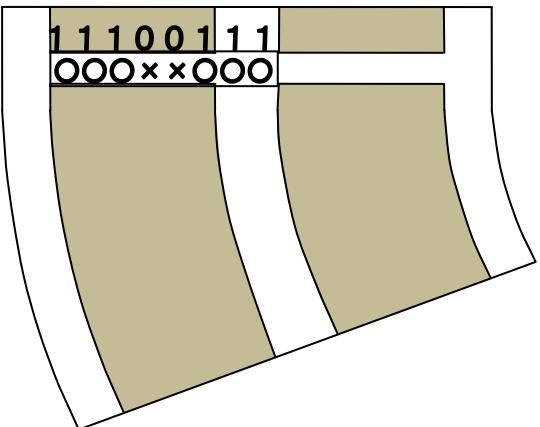
6. プログラム解説「kit12_38a.c」

8		センサが「0110 0000」になりました。 「0000 0110」ではないので、まだ右に曲げ続けます。
9		センサが「0000 0100」になりました。 落ちそうですが、プログラムとしては 「0000 0110」ではないので、まだ右に曲げ続けます。 ただ、車体は落ちているかもしれません。
10		左に寄っていたのが右に戻り始め、 センサが「0000 0110」になりました。 この状態でパターン 11 に戻ります。

この考え方でプログラムします。

```
case 12:
/* 右へ大曲げの終わりのチェック */
if( sensor_inp(MASK3_3) == 0x06 ) {
    pattern = 11;
}
break;
```

これで完成と思いきや、パターン 11 ではクロスライン、右ハーフライン、左ハーフラインのチェックを行っていました。パターン 12 では必要ないのでしょうか。

			クロスラインの手前で、センサが「0000 0011」になりました。パターン 12 に移ります。
11			
12			センサは「0000 0110」ではないので、右に曲げ続けます。
13			<p>クロスラインなのでクランク検出処理に移らなければいけません。 ただ、パターン 12 では、センサ状態が「0000 0110」かどうかしか調べていないので、クロスラインと判断できずそのまま通過してしまいます。</p> <p>このように、パターン 12 を処理中でも、クロスラインを検出するがあります。同様に右ハーフライン、左ハーフラインもあり得ます。そこで、パターン 12 にも 3 種類のチェックを追加します。</p>

6. プログラム解説「kit12_38a.c」

最終的なプログラムを下記に示します。

```

215 :     case 12:
216 :         /* 右へ大曲げの終わりのチェック */
217 :         if( check_crossline() ) {      /* 大曲げ中もクロスラインチェック */
218 :             pattern = 21;
219 :             break;
220 :         }
221 :         if( check_rightline() ) {      /* 右ハーフラインチェック */
222 :             pattern = 51;
223 :             break;
224 :         }
225 :         if( check_leftline() ) {      /* 左ハーフラインチェック */
226 :             pattern = 61;
227 :             break;
228 :         }
229 :         if( sensor_inp(MASK3_3) == 0x06 ) {
230 :             pattern = 11;
231 :         }
232 :         break;

```

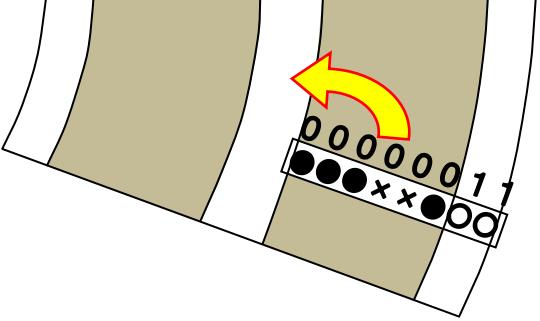
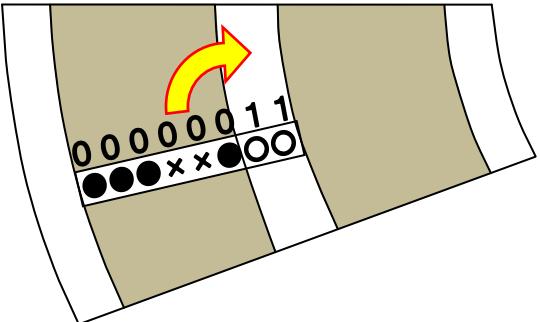
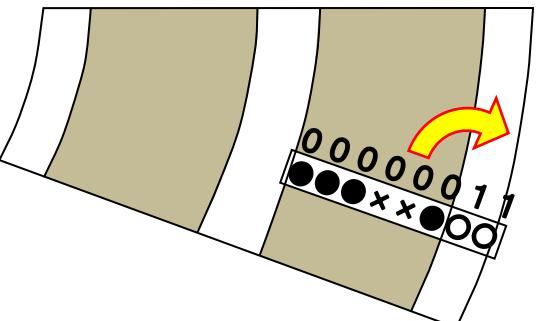
パターン 12 のプログラムはこれで完成です。

6.4.32 パターン 13: 左へ大曲げの終わりのチェック

センサ状態 0xc0 は、一番大きく右に寄ったときのセンサ状態です。そのため、これ以上カーブで脹らんだ場合、下図のようになる可能性があります。

1		<p>大きく左に寄ったときのセンサの状態です。 センサは、「1100 0000」です。 この状態になったなら、下記プログラムが実行されます。</p> <pre> handle(-25); motor(19 , 30); </pre> <p>よって、左に 25 度ハンドルを切って左モータ 19%、右モータ 30%で回します。</p>
---	--	--

2		<p>さらに右に寄りました。 センサは、「1000 0001」です。 この状態のとき、実行するプログラムの記述はありません。この場合、前の状態を保持します。 前の状態は「1100 0000」なので、このセンサ値のモータ回転数、ハンドル角度になります。</p>
---	--	---

3		さらに右に寄りました。 センサは、「0000 0011」です。
4		センサ状態「0000 0011」は、本プログラムでは、左図のように、マイコンカーが左に寄っているので、右にハンドルを曲げる状態を想定しています。この状態になったなら、下記プログラムが実行されます。 <pre>handle(25); motor(30, 19);</pre> よって、右に 25 度ハンドルを切って左モータ 30%、右モータ 19%で回します。
5		実際は、右に大きく寄っているので、この状態で右にハンドルを曲げると、脱輪してしまいます。

そこで、左に大曲げしたら、あるセンサ状態に戻るまで左に大曲げし続けるようにします。この“あるセンサ状態”を判定する部分を、パターン 13 に作ります。

パターン 11 の case 0xc0 部分

```

203 :           case 0xc0:
204 :             /* 大きく右寄り→左へ大曲げ */
205 :             handle( -25 );
206 :             motor( 19 , 30 );
207 :             pattern = 13;   ←追加 パターン 13 へ移る
208 :             break;

```

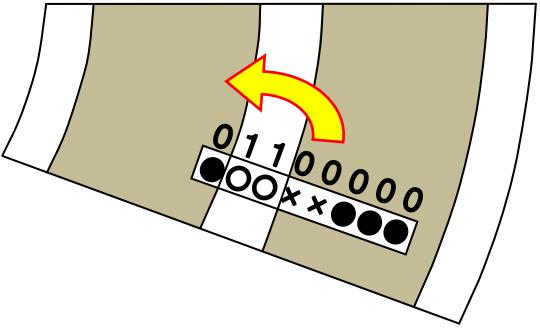
6. プログラム解説「kit12_38a.c」

6		<p>センサが「1100 0000」になったなら、パターン 13 に移ります。</p> <p>今回パターン 13 では、この 1 つ内側のセンサ状態である「0110 0000」になったなら、パターン 11 に戻るプログラムを考えました。</p> <p>この考え方で良いか検証してみます。</p>
---	--	---

7		<p>センサが「0000 0011」になりました。「0110 0000」ではないので、まだ左に曲げ続けます。</p> <p>先ほどは左に寄っていると勘違いしましたが、今回はパターン 13 なので勘違いしません。</p>
---	--	---

8		<p>センサが「0000 0110」になりました。 「0110 0000」ではないので、まだ左に曲げ続けます。</p>
---	--	---

9		<p>センサが「0010 0000」になりました。 落ちそうですが、プログラムとしては 「0110 0000」ではないので、まだ左に曲げ続けます。</p> <p>ただ、車体は落ちているかもしれません。</p>
---	--	--

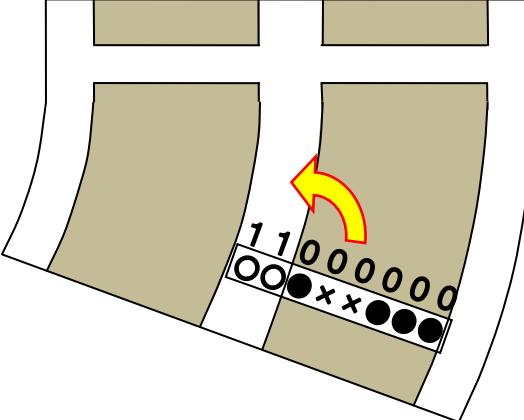
10		右に寄っていたのが左に戻り始め、センサが「0110 0000」になりました。 この状態でパターン 11 に戻ります。
----	---	---

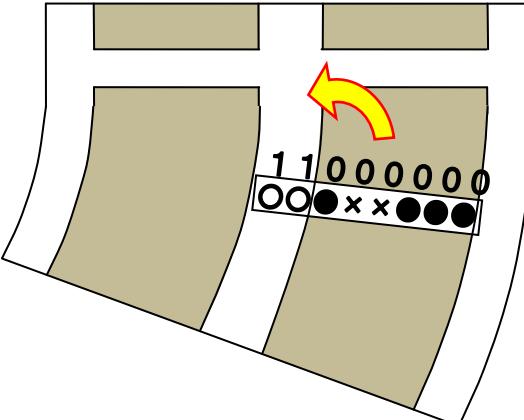
この考え方でプログラムします。

case 13:

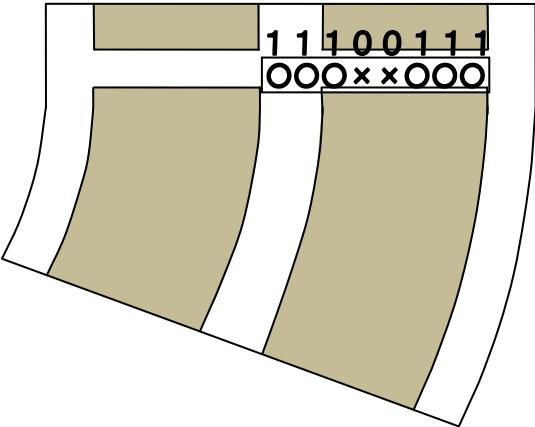
```
/* 左へ大曲げの終わりのチェック */
if( sensor_inp(MASK3_3) == 0x60 ) {
    pattern = 11;
}
break;
```

これで完成と思いきや、パターン 11 ではクロスライン、右ハーフライン、左ハーフラインのチェックを行っていました。パターン 13 では必要ないのでしょうか。

11		クロスラインの手前で、センサが「1100 0000」になりました。パターン 13 に移ります。
----	---	---

12		センサは「0110 0000」ではないので、左に曲げ続けます。
----	---	---------------------------------

6. プログラム解説「kit12_38a.c」

13		<p>クロスラインなのでクランク検出処理に移らなければいけません。 ただ、パターン 13 では、センサ状態が「0110 0000」かどうかしか調べていないので、クロスラインと判断できずそのまま通過します。</p> <p>このように、パターン 13 を処理中でも、クロスラインを検出することがあります。同様に右ハーフライン、左ハーフラインもあり得ます。そこで、パターン 13 にも 3 種類のチェックを追加します。</p>
----	---	--

最終的なプログラムを下記に示します。

```

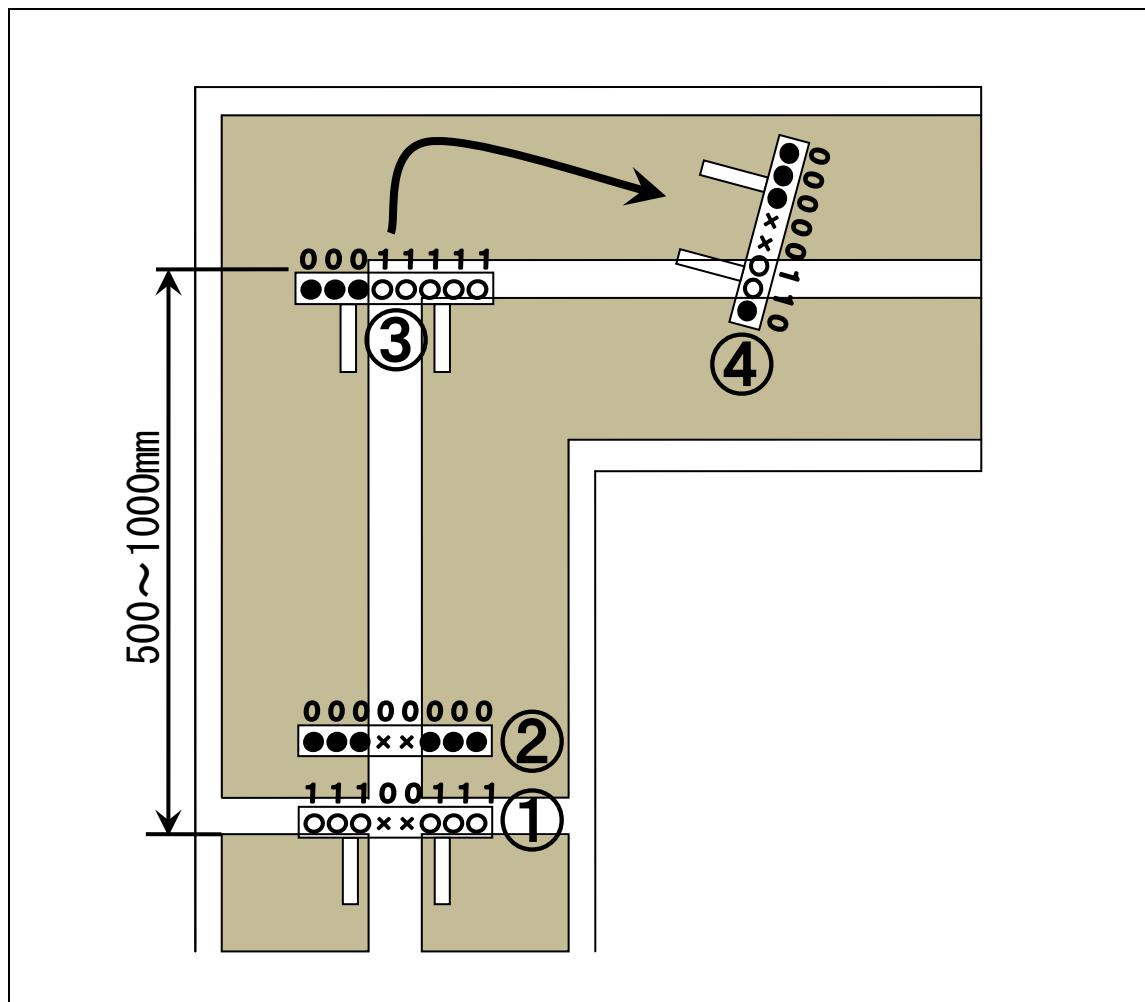
234 :     case 13:
235 :         /* 左へ大曲げの終わりのチェック */
236 :         if( check_crossline() ) {          /* 大曲げ中もクロスラインチェック */
237 :             pattern = 21;
238 :             break;
239 :         }
240 :         if( check_rightline() ) {        /* 右ハーフラインチェック */
241 :             pattern = 51;
242 :             break;
243 :         }
244 :         if( check_leftline() ) {        /* 左ハーフラインチェック */
245 :             pattern = 61;
246 :             break;
247 :         }
248 :         if( sensor_inp(MASK3_3) == 0x60 ) {
249 :             pattern = 11;
250 :         }
251 :         break;

```

パターン 13 のプログラムはこれで完成です。

6.4.33 クランク概要

パターン 21 から 42 は、クランク(直角)に関するプログラムです。処理の概要を下図に示します。

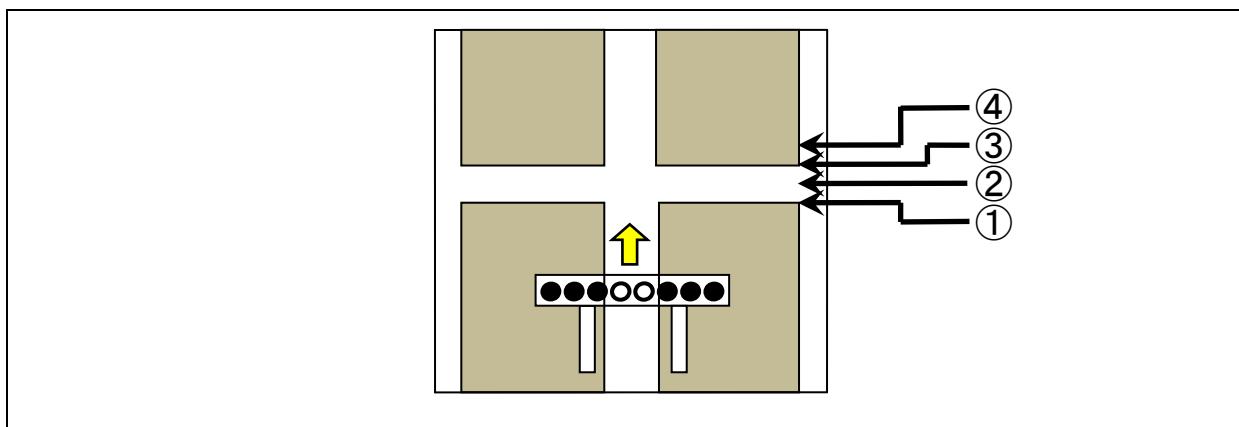


①	check_crossline 関数でクロスラインを検出します。500~1000mm 先で、右クランクか左クランクがあるので、クリアできるスピードにするためブレーキをかけます。また、クロスライン通過中にセンサが誤検出しないよう②の位置までセンサは見ません。
②	この位置から徐行開始します。中心線をトレースしながら進んでいきます。
③	クランクを検出すると、クランクがある方向にハンドルを切れます。
④	中心線を検出すると、パターン 11 に戻りライントレースを再開します。

このように、クランクをクリアします。次から具体的なプログラムの説明をしていきます。

6.4.34 パターン 21:クロスライン検出時の処理

パターン 21 は、クロスラインを見つけた瞬間に移ってきます。まず、クロスラインを通過し終わるまで、下図のような状態があります。



- ①クロスラインの始めの境目
- ②クロスラインの白色部分
- ③クロスラインの終わりの境目
- ④通常コース、ここから徐行しながらトレース

④に行くまでにコースが、クロスラインの始めの境目→白→終わりの境目→黒と変化します。それをプログラムで検出して……、とかなり複雑なプログラムになりそうです。

ちょっと考え方を変えてみます。①の位置から④まで、余裕を見て約 100mm あります(正確にはクロスラインの幅は 20~40mm です)。ほぼ中心線の位置にいるとして 100mm くらいならセンサの状態を見ずに進めても大きくなれなさそうです。マイコンカーキットは距離の検出はできないので、タイマでセンサを読まない時間を作ります。時間については、走行スピードによって変わるので、何とも言えません。とりあえず 0.1 秒として、細かい時間は走らせて微調整することにします。同時にモータドライブ基板の LED を点灯させ、パターン 21 に入ったことを外部に知らせるようにします。

まとめると、

- LED2,3 を点灯させる
- ハンドルを 0 度にする
- 左モータ、右モータの PWM を 0%にしてブレーキをかける
- 0.1 秒待つ
- 0.1 秒たつたら次のパターンへ移る

これをパターン 21 でプログラム化します。

```
case 21:
    led_out( 0x3 );
    handle( 0 );
    speed( 0 , 0 );
    if( cnt1 > 100 ) {
        pattern = 22; /* 0.1 秒後パターン 22 へ */
    }
    break;
```

完成しました。本当にこれでよいか見直してみます。cnt1 が 100 以上になつたら(100 ミリ秒たつたら)、パターン 22 へ移るようにしています。これはパターン 21 を開始したときに、cnt1 が 0 になっている必要があります。例えば

パターン 21 にプログラムが移ってきた時点で cnt1 が 1000 なら、1 回目の比較で cnt1 は 100 以上と判断して、すぐにパターン 22 に移ってしまいます。0.1 秒どころか、1 回しかパターン 21 が実行しません(約数 $10 \mu\text{s}$)。そこで、パターンをもう一つ増やします。パターン 21 はブレーキをかけ cnt1 をクリア、パターン 22 で 0.1 秒たつたかチェックするようにします。

再度まとめると、下記のようになります。

パターン 21 で行うこと	<ul style="list-style-type: none"> LED2,3 を点灯 ハンドルを 0 度に 左右モータ PWM を 0%にしてブレーキをかける パターンを次へ移す cnt1 をクリア
パターン 22 で行うこと	<ul style="list-style-type: none"> cnt1 が 100 以上になったなら、次のパターンへ移す

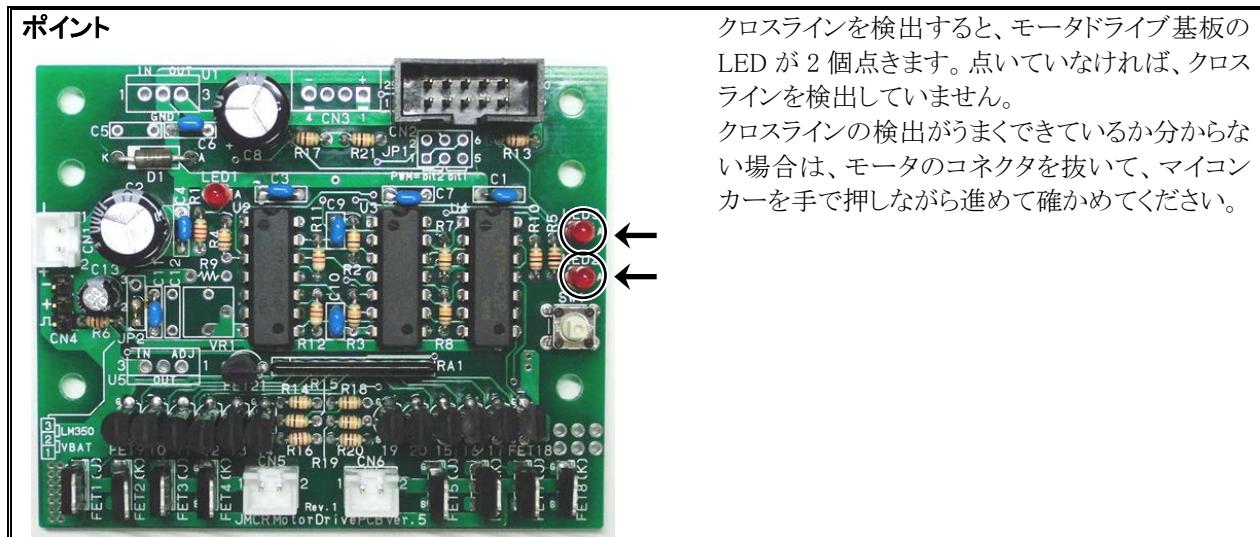
上記にしたがって再度プログラムを作成します。

```

253 :     case 21:
254 :         /* クロスライン検出時の処理 */
255 :         led_out( 0x3 );
256 :         handle( 0 );
257 :         motor( 0 , 0 );
258 :         pattern = 22;
259 :         cnt1 = 0;
260 :         break;
261 :
262 :     case 22:
263 :         /* クロスラインを読み飛ばす */
264 :         if( cnt1 > 100 ) {
265 :             pattern = 23;
266 :             cnt1 = 0;
267 :         }
268 :         break;

```

クロスラインを検出してから徐行して、トレース開始部分までのプログラムが完成しました。

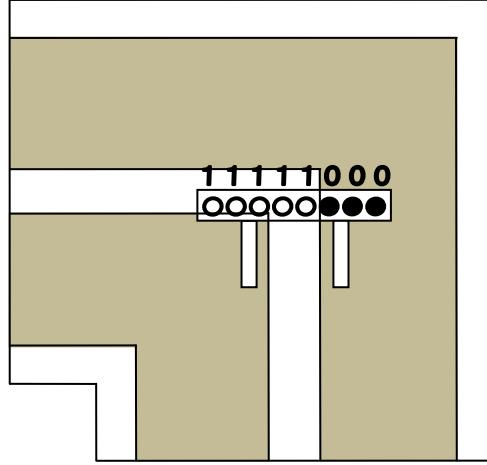
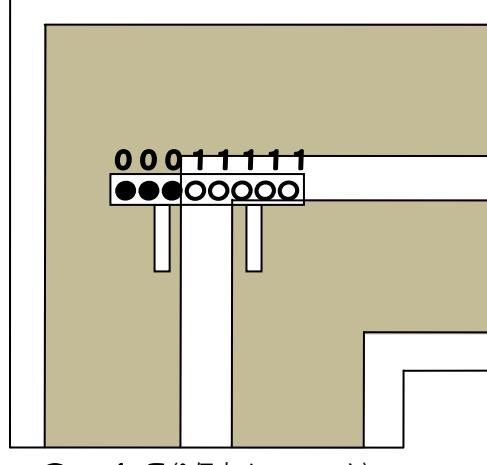
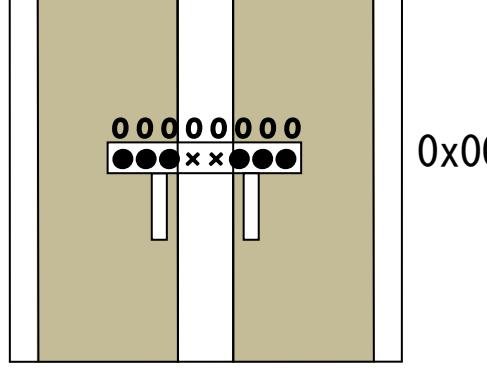


6.4.35 パターン 23:クロスライン後のトレース、クランク検出

パターン 21、22 では、クロスラインを検出後、ブレーキを 0.1 秒かけクロスラインを通過させました。パターン 23 では、その後の処理を行います。

クロスラインを過ぎたので、後はクランク(直角)の検出です。クランクを見つけたらすぐに曲げなければいけませんので徐行して進んでいきます。またクランクまでの間、中心線をトレースしながら進んでいく処理も必要です。

今回、下図のように考えました。

 <p>0xf8 (8個すべてチェック)</p>	<p>左クランク部分では、8 個のセンサ状態が左図のように「0xf8」になりました。そこで、「0xf8」の状態を左クランクと判断するようにします。</p> <p>このとき、ハンドルを左いっぱいまで曲げなければ外側へ張らんで脱輪してしまいます。何度曲げるか… それはマイコンカーの作りによって違いますので、実際のマイコンカーを見て何処までハンドルが切れるか確かめる必要があります。プロジェクト「sioservo2_38a」でハンドルの最大切れ角を調べるとキットは大体 40 度くらいです。2 度くらい余裕を見て、プログラムでは 38 度にします。</p> <p>左モータ、右モータの回転数は、左に大きく曲げる所以、左モータを少なく、右モータを多めにします。実際に何%にするかは、走らせて確かめるとして、ここでは、左モータ 10%、右モータ 50%にしておきます。まとめると下記のようになります。</p> <p>ハンドル:-38 度 左モータ:10% 右モータ:50%</p> <p>その後、パターン 31 へ移ります。</p>
 <p>0x1f (8個すべてチェック)</p>	<p>右クランク部分です。考え方は左クランクと同様です。 まとめると下記のようになります。</p> <p>ハンドル:38 度 左モータ:50% 右モータ:10%</p> <p>その後、パターン 41 へ移ります。</p>
 <p>0x00</p>	<p>直進時、センサの状態は「0x00」です。マイコンカーは中心にあると判断します。ハンドルは0度です。問題はモータのPWM値です。モータのPWM値は、クランクを見つけたときに直角を曲がれるスピードにしなければいけません。ここでは 40%にしておき、実際に走らせて微調整することにします。まとめると下記のようになります。</p> <p>ハンドル:0 度 左モータ:40% 右モータ:40%</p>

 0x04 0x06 0x07 0x03	<p>マイコンカーが左に寄ったときを考えています。 中心から少しずつ左へずらしていくと左図のように 4 つの状態になりました。センサの状態はもっと左へ寄ることも考えられますが、クロスラインの後は直線しかないと分かっているので、これ以上左に寄らないと判断します。</p> <p>動作は、左へ寄っているので右へハンドルを切ります。切り角が小さすぎるとそれが大きいときに戻りきれなくなり、大きすぎるとセンサが左右にばたばた振れてしまいます。ちょうど良い角度に調整するのは難しいです。今回は、どの状態も 8 度にします。右に曲げますので右モータの PWM を左モータより少なめにしておきます。まとめると下記のようになります。</p> <p>ハンドル: 8 度 左モータ: 40% 右モータ: 35%</p>
 0x20 0x60 0xe0 0xc0	<p>マイコンカーが右に寄ったときを考えています。 中心から少しずつ右へずらしていくと左図のように 4 つの状態になりました。センサの状態はもっと右へ寄ることも考えられますが、クロスラインの後は直線しかないと分かっているので、これ以上右に寄らないと判断します。</p> <p>動作は、右へ寄っているので左へハンドルを切ります。切り角が小さすぎるとそれが大きいときに戻りきれなくなり、大きすぎるとセンサが左右にばたばた振れてしまいます。ちょうど良い角度に調整するのは難しいです。今回は、どの状態も -8 度にします。左に曲げますので左モータの PWM を右モータより少なめにしておきます。まとめると下記のようになります。</p> <p>ハンドル: -8 度 左モータ: 35% 右モータ: 40%</p>

ポイントは、クランクチェックは 8 個のセンサすべてを使用することです。他は「MASK3_3」でマスクして中心の 2 個のセンサは使用しません。

6. プログラム解説「kit12_38a.c」

プログラム化すると下記のようになります。

```

270 :     case 23:
271 :         /* クロスライン後のトレース、クランク検出 */
272 :         if( sensor_inp(MASK4_4)==0xf8 ) {
273 :             /* 左クランクと判断→左クランククリア処理へ */
274 :             led_out( 0x1 );
275 :             handle( -38 );
276 :             motor( 10 ,50 );
277 :             pattern = 31;
278 :             cnt1 = 0;
279 :             break;
280 :         }
281 :         if( sensor_inp(MASK4_4)==0x1f ) {
282 :             /* 右クランクと判断→右クランククリア処理へ */
283 :             led_out( 0x2 );
284 :             handle( 38 );
285 :             motor( 50 ,10 );
286 :             pattern = 41;
287 :             cnt1 = 0;
288 :             break;
289 :         }
290 :     switch( sensor_inp(MASK3_3) ) {
291 :         case 0x00:
292 :             /* センタ→まっすぐ */
293 :             handle( 0 );
294 :             motor( 40 ,40 );
295 :             break;
296 :         case 0x04:
297 :         case 0x06:
298 :         case 0x07:
299 :         case 0x03: } case を続けて書くと
299 :             0x04 または 0x06 または 0x07 または 0x03 のとき
299 :             という意味になります。
300 :             /* 左寄り→右曲げ */
301 :             handle( 8 );
302 :             motor( 40 ,35 );
303 :             break;
304 :         case 0x20:
305 :         case 0x60:
306 :         case 0xe0:
307 :         case 0xc0: } case を続けて書くと
307 :             0x20 または 0x60 または 0xe0 または 0xc0 のとき
307 :             という意味になります。
308 :             /* 右寄り→左曲げ */
309 :             handle( -8 );
310 :             motor( 35 ,40 );
311 :             break;
312 :     }
313 :     break;

```

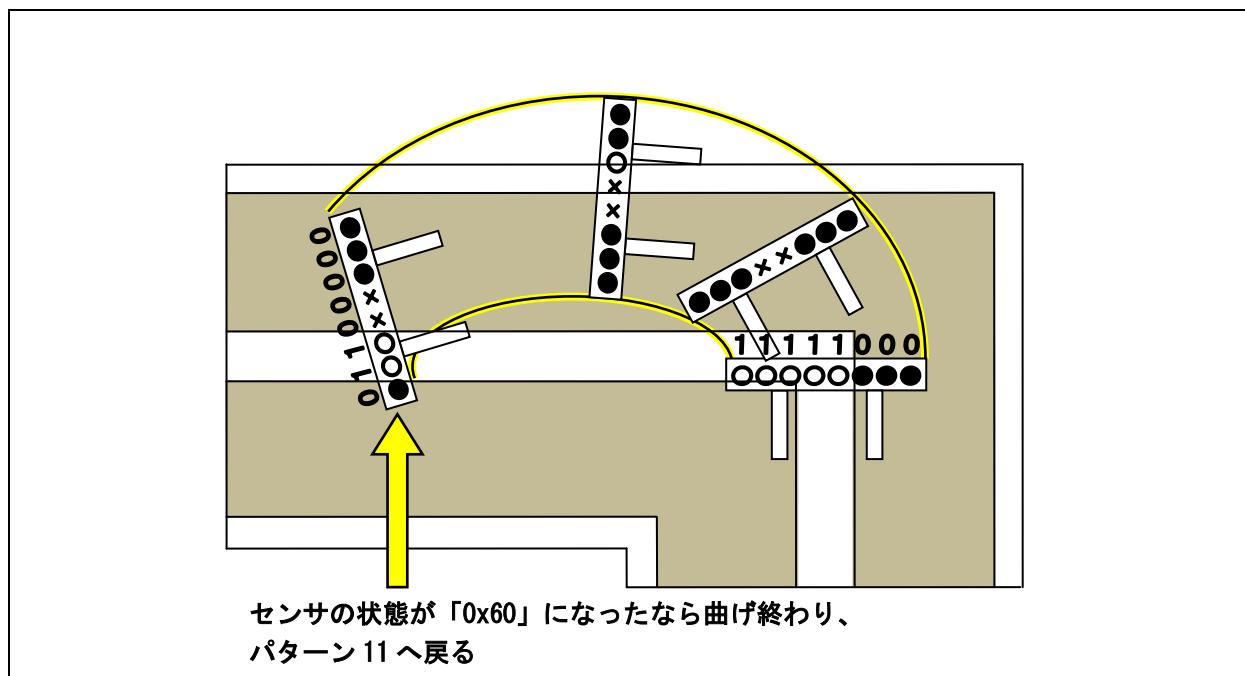
左クランク、右クランクは、if 文で判定しています。その他は、switch 文を使って「sensor_inp(MASK3_3)」の値で case へ分岐するようにしました。

6.4.36 パターン 31、32: 左クランククリア処理

パターン 23 でセンサ 8 個が「0xf8」になると、左クランクと判断してハンドルを左に大きく曲げ、クランクをクリアしようとします。

次の問題が出てきます。「いつまで左に曲げ続けるか」ということです。この部分のプログラムが、パターン 31、32 です。

下図のように考えました。



「0xf8」と判断すると左へ大曲げしますが、スピードがついているので脹らみ気味に曲がっていきます。センサが中心線付近に来て「0x60」になったときを曲げ終わりと判断してパターン 11 へ戻ります。

これをプログラム化してみます。

```
case 31:
    if( sensor_inp(MASK3_3) == 0x60 ) {
        pattern = 11;
    }
    break;
```

上記プログラムで実際に走らせてみました。左クランクでセンサの状態が「0xf8」になった瞬間、ハンドルが左へ曲がり始めました。想定では、センサが「0x60」になるまで曲げ続けるはずでしたが、すぐにハンドルがまっすぐ向いてしまい、クランク部分を直進し脱輪してしまいました。動作が早すぎてよく分からないので、モータとサーボのコネクタを抜いて左クランク部分を手で押しながらゆっくりと進ませていきます。センサの状態をじっくりと観察すると次の図のようになっていました。

参考

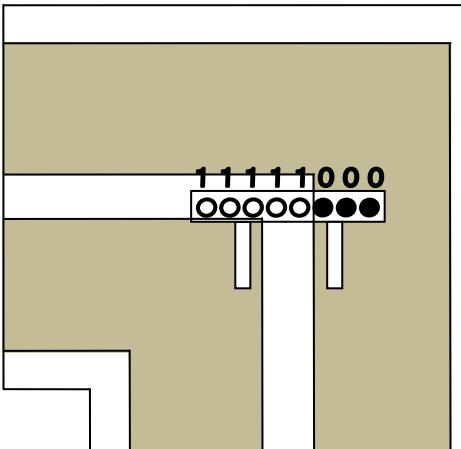
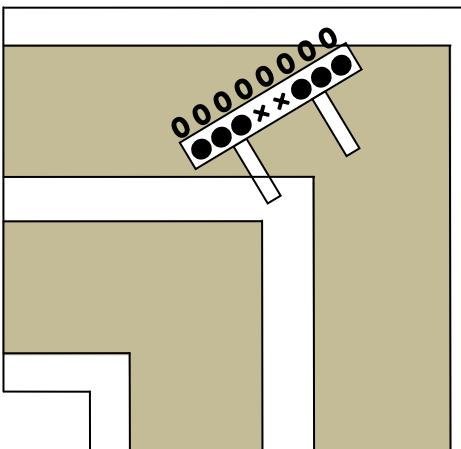
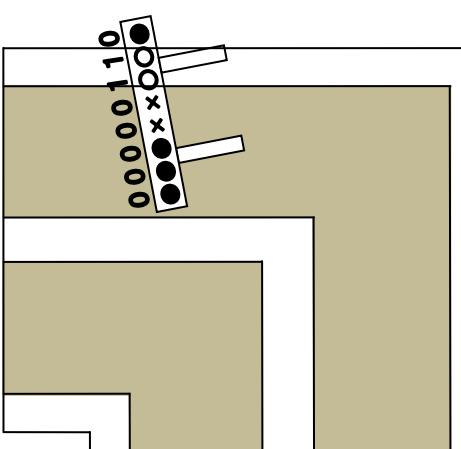
マイコンカーの動きがよく分からない場合は、モータのコネクタを抜いて、
手で押しながらセンサ状態を確認することをお勧めします。

6. プログラム解説「kit12_38a.c」

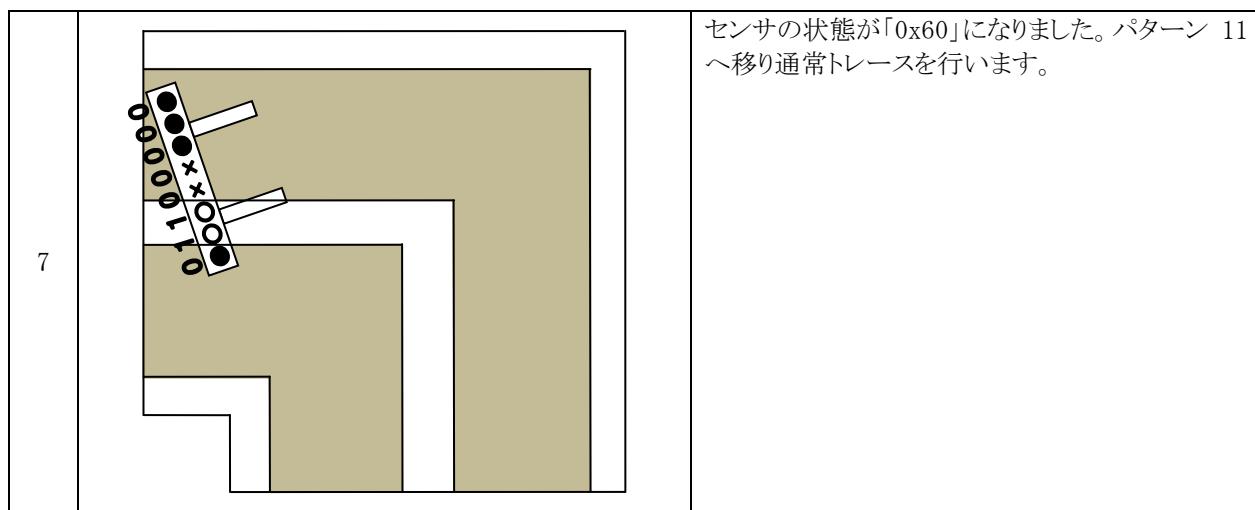
			センサの状態が「0xf8」になったので、センサの状態が「0x60」になるまで左に 38 度曲げます。
1			ハンドルを曲げ始めた瞬間、白と黒の境目で(本当は白と灰と黒色ですが、灰色は白と見なします)センサの状態が「0x60」になりました。プログラムでは、「0x60」になったので、この状態でパターン 11 に移ります。
2			パターン 11 では、センサ状態「0x00」はセンサが中心にあると判断して、ハンドル 0 度、モータは 100%で走行します。まっすぐ進んでしまい脱輪します。

センサの感度を調べてみるといちばん左のセンサの調整が弱めになっており、左から 2 番目、3 番目のセンサより先に "0" になっていました。そのため、白と黒の境目で「0x60」になり誤動作していました。いちばん左のセンサ感度をもう少し強くすれば良いのですが、センサのちょっとした感度で脱輪するのは嫌なのでプログラムで対処してみます。

考えてみると、「0xf8」を検出してから中心線を検出するセンサ状態「0x60」になるまで、ちょっと時間がかかることに気がつきました。そこで左クランクを見つけてサーボ、モータの回転数を設定した後、0.2 秒間はセンサを見ないで境目を通過するのを待って、0.2 秒後からセンサをチェックするようにしてはどうでしょうか。図を書いてイメージしてみます。

4		センサの状態が「0xe8」になったので、ハンドルを右に 38 度曲げます。その後、センサを見ずに 0.2 秒進ませます。
5		0.2 秒後です。ここからセンサの状態が「0x60」かどうかチェックします。
6		まだセンサの状態が「0x60」ではないので曲げ続けます。

6. プログラム解説「kit12_38a.c」



クランクを検出した 0.2 秒後、図 5 のように白と黒の境目を越えた位置にあります。その後は「0x60」になるまで曲げ続けるだけです。これなら良さそうです。プログラム化します。

```

315 :     case 31:
316 :         /* 左クランククリア処理 安定するまで少し待つ */
317 :         if( cnt1 > 200 ) {
318 :             pattern = 32;
319 :             cnt1 = 0;
320 :         }
321 :         break;
322 :
323 :     case 32:
324 :         /* 左クランククリア処理 曲げ終わりのチェック */
325 :         if( sensor_inp(MASK3_3) == 0x60 ) {
326 :             led_out( 0x0 );
327 :             pattern = 11;
328 :             cnt1 = 0;
329 :         }
330 :         break;

```

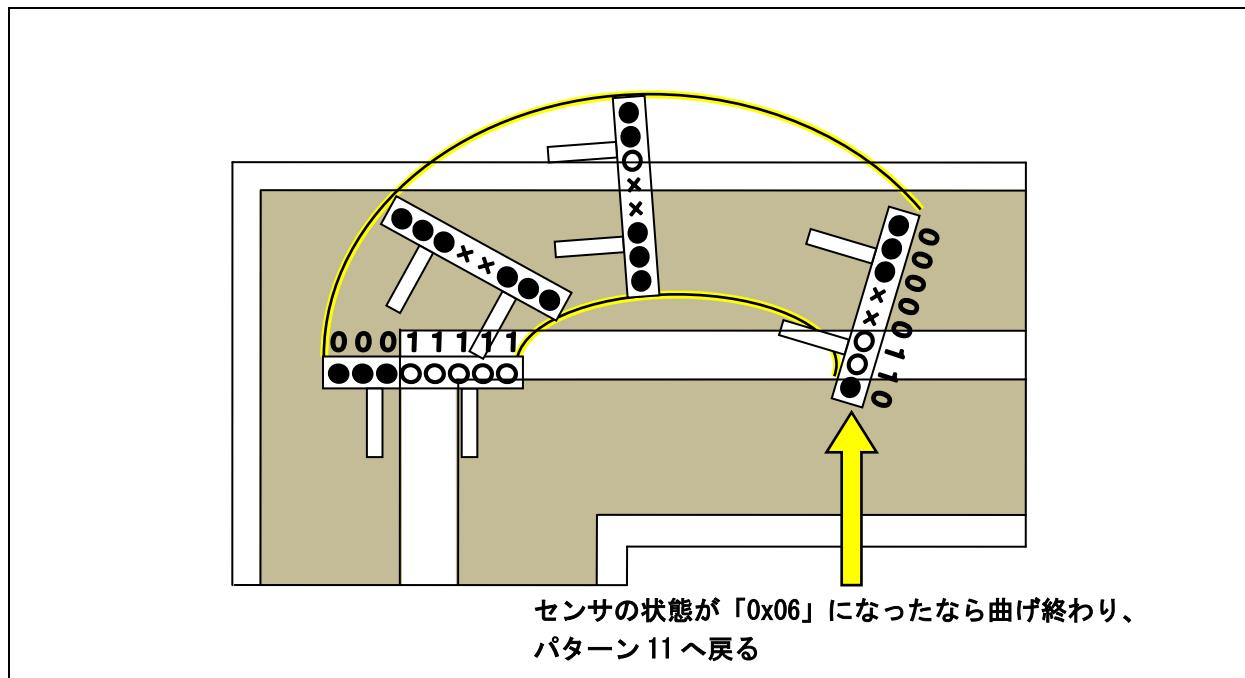
317 行で cnt1 が 200 以上かチェックしています。200 以上なら、すなわち 0.2 秒たったならパターン 32 へ移ります。ちなみに cnt1 のクリアは、パターン 31 へ移る前の 278 行で行っています。

6.4.37 パターン 41、42:右クランククリア処理

パターン 23 でセンサ 8 個が「0x1f」になると、右クランクと判断してハンドルを右に大きく曲げクランクをクリアしようとします。

次の問題が出てきます。「いつまで右に曲げ続けるか」ということです。この部分のプログラムが、パターン 41、42 です。

下図のように考えました。



「0x1f」と判断すると右へ大曲げしますが、スピードがついているので脹らみ気味に曲がっていきます。センサが中心線付近に来て「0x06」になったときを曲げ終わりと判断してパターン 11 へ戻ります。

これをプログラム化してみます。

```
case 41:
    if( sensor_inp(MASK3_3) == 0x06 ) {
        pattern = 11;
    }
    break;
```

上記プログラムで実際に走らせてみました。右クランクでセンサの状態が「0x1f」になった瞬間、ハンドルが右へ曲がり始めました。想定では、センサが「0x06」になるまで曲げ続けるはずでしたが、すぐにハンドルがまっすぐに向いてしまい、クランク部分を直進し脱輪してしまいました。動作が早すぎてよく分からないので、モータとサーボのコネクタを抜いて右クランク部分を手で押しながらゆっくりと進ませていきます。センサの状態をじっくりと観察すると次の図のようになっていました。

参考

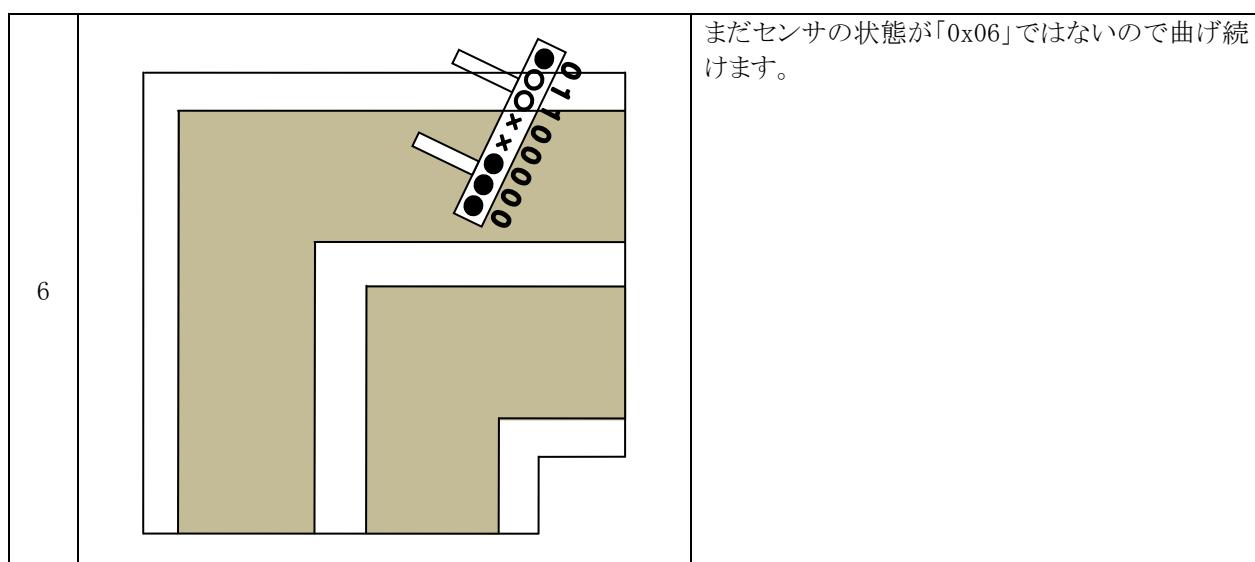
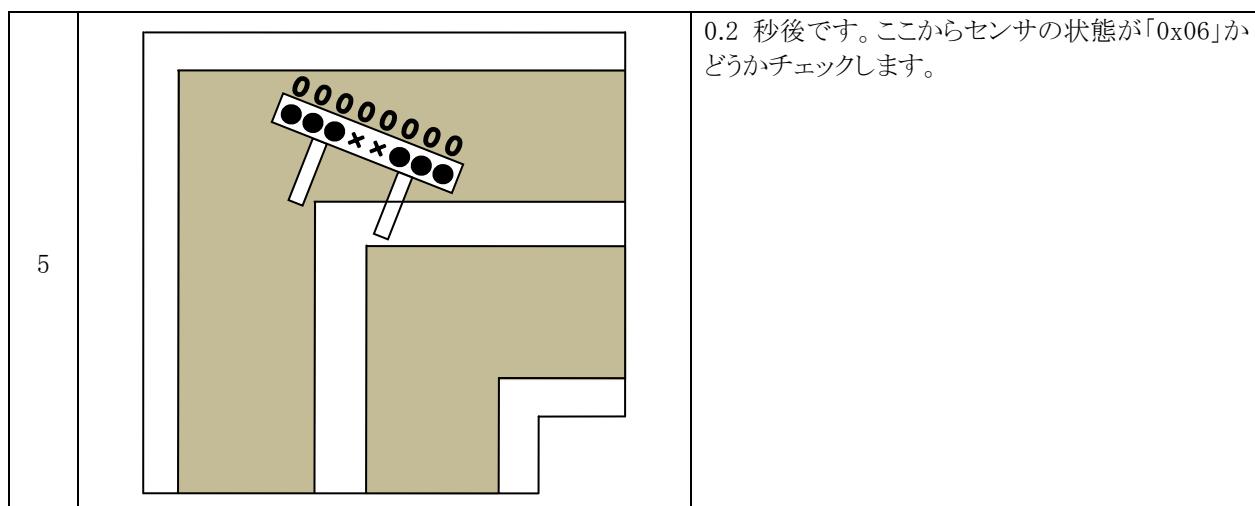
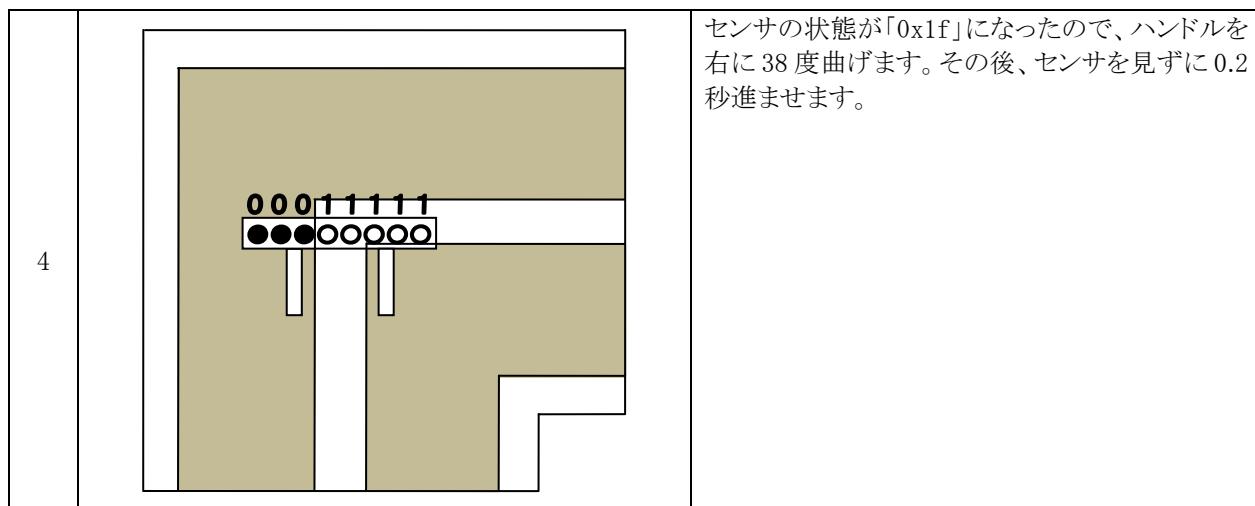
マイコンカーの動きがよく分からない場合は、モータのコネクタを抜いて、手で押しながらセンサ状態を確認することをお勧めします。

6. プログラム解説「kit12_38a.c」

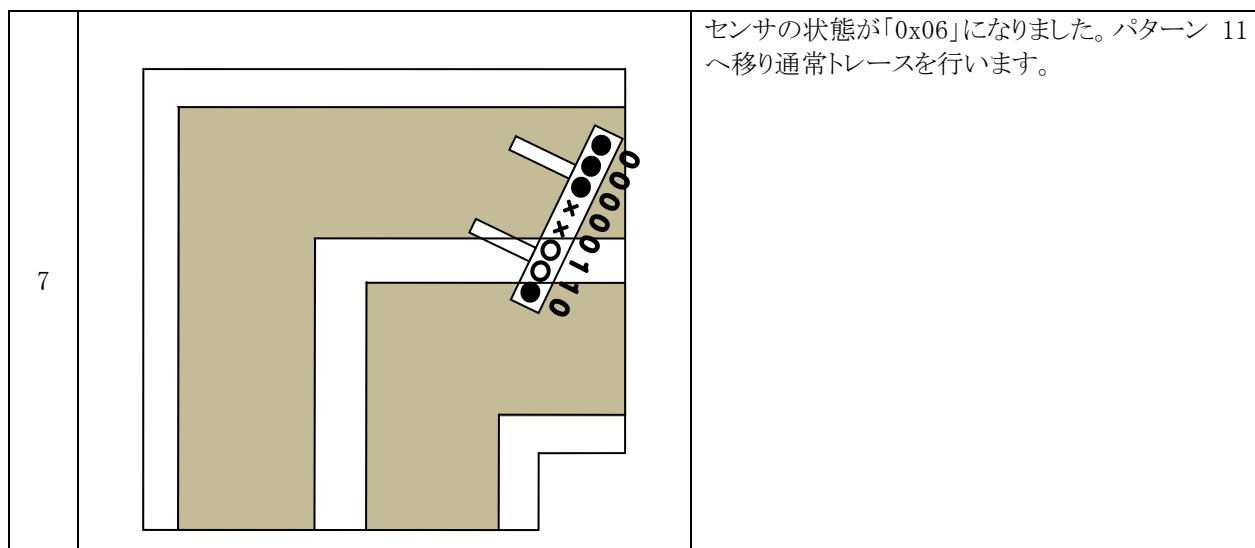
			センサの状態が「0x1f」になったので、センサの状態が「0x06」になるまで右に 38 度曲げます。
1		00011111 ●●○○○○○○	
2		00000110 ●●●××○○●	ハンドルを曲げ始めた瞬間、白と黒の境目で(本当は白と灰と黒色ですが、灰色は白と見なします)センサの状態が「0x06」になりました。プログラムでは、「0x06」になったので、この状態でパターン 11 に移ります。
3		00011111 ●●○○○○○○	パターン 11 では、センサ状態「0x00」はセンサが中心にあると判断して、ハンドル 0 度、モータは 100%で走行します。まっすぐ進んでしまい脱輪します。

センサの感度を調べてみるといちばん右のセンサの調整が弱めになっており、右から 2 番目、3 番目のセンサより先に“0”になっていました。そのため、白と黒の境目で「0x06」になり誤動作していました。いちばん右のセンサ感度をもう少し強くすれば良いのですが、センサのちょっとした感度で脱輪するのは嫌なのでプログラムで対処してみます。

考えてみると、「0x1f」を検出してから中心線を椰出するセンサ状態「0x06」になるまで、ちょっと時間がかかることに気がつきました。そこで右クランクを見つけてサーボ、モータの回転数を設定した後、0.2 秒間はセンサを見ないで境目を通過するのを待って、0.2 秒後からセンサをチェックするようにしてはどうでしょうか。図を書いてイメージしてみます。



6. プログラム解説「kit12_38a.c」



クランク検出を検出した 0.2 秒後、図 5 のように白と黒の境目を越えた位置にあります。その後は「0x06」になるまで安心して曲げ続けるだけです。これなら良さそうです。プログラム化します。

```

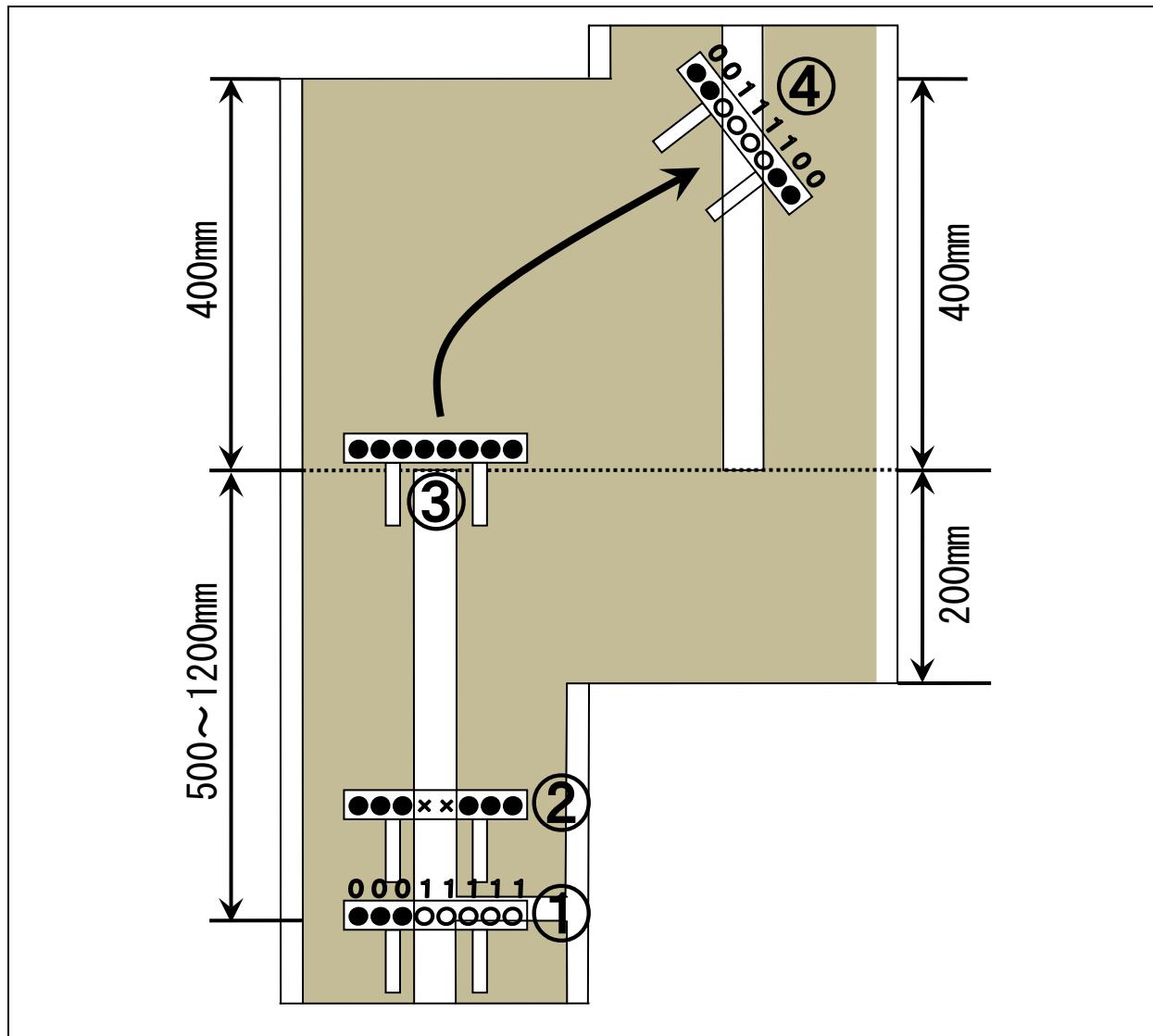
332 :     case 41:
333 :         /* 右クランククリア処理 安定するまで少し待つ */
334 :         if( cnt1 > 200 ) {
335 :             pattern = 42;
336 :             cnt1 = 0;
337 :         }
338 :         break;
339 :
340 :     case 42:
341 :         /* 右クランククリア処理 曲げ終わりのチェック */
342 :         if( sensor_inp(MASK3_3) == 0x06 ) {
343 :             led_out( 0x0 );
344 :             pattern = 11;
345 :             cnt1 = 0;
346 :         }
347 :         break;

```

334 行で cnt1 が 200 以上かチェックしています。200 以上なら、すなわち 0.2 秒たったならパターン 42 へ移ります。ちなみに cnt1 のクリアは、パターン 41 へ移る前の 287 行で行っています。

6.4.38 右レーンチェンジ概要

パターン 51 から 54 は、右レーンチェンジに関するプログラムです。処理の概要を下図に示します。

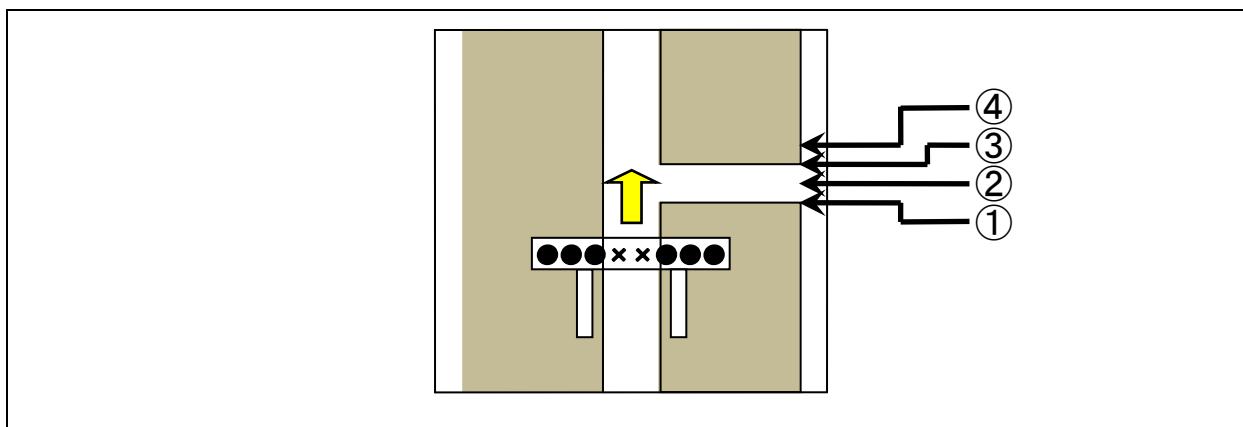


①	check_rightline 関数で右ハーフラインを検出します。500~1200mm 先で、右レーンに移動するために右に曲がらなければいけないのでブレーキをかけます。また、右ハーフライン通過時にセンサが誤検出しないよう②の位置までセンサは見ません。
②	この位置から徐行開始します。中心線をトレースしながら進んでいきます。
③	中心線が無くなると、右へハンドルを切ります。
④	新しい中心線を検出すると、今度はこの中心線でライントレースを再開します。

このように、右レーンチェンジをクリアします。次から具体的なプログラムの説明をしていきます。

6.4.39 パターン 51: 右ハーフライン検出時の処理

パターン 51 は、右ハーフラインを見つけた瞬間に移ってきます。まず、右ハーフラインを通過し終わるまで、下図のような状態があります。



- ①右ハーフラインの始めの境目
- ②右ハーフラインの白色部分
- ③右ハーフラインの終わりの境目
- ④通常コース、ここから徐行しながらトレース

④に行くまでにコースが、右ハーフラインの始めの境目→白→終わりの境目→黒と変化します。それをプログラムで検出して……、とかなり複雑なプログラムになりそうです。

ちょっと考え方を変えてみます。①の位置から④まで、余裕を見て約 100mm あります（正確にはハーフラインの幅は 20~40mm です）。ほぼ中心線の位置にいるとして 100mm くらいならセンサの状態を見ずに進めても大きくなれなさそうです。マイコンカーキットは距離の検出はできないので、タイマでセンサを読まない時間を作ります。時間については、走行スピードによって変わるので、何とも言えません。とりあえず 0.1 秒として、細かい時間は走らせて微調整することにします。同時にモータドライブ基板の LED を点灯させ、パターン 51 に入ったことを外部に知らせるようにします。

まとめると、

- LED2 を点灯（クロスラインとは違う点灯にして区別させます）
- ハンドルを 0 度に
- 左右モータ PWM を 0% にしてブレーキをかける
- 0.1 秒待つ
- 0.1 秒たつたら次のパターンへ移る

これをパターン 51 でプログラム化します。

```
case 51:
    led_out( 0x2 );
    handle( 0 );
    speed( 0 , 0 );
    if( cnt1 > 100 ) {
        pattern = 52; /* 0.1 秒後パターン 52 へ */
    }
    break;
```

完成しました。本当にこれでよいか見直してみます。cnt1 が 100 以上になつたら（100 ミリ秒たつたら）、パターン 52 へ移るようにしています。これはパターン 51 を開始したときに、cnt1 が 0 になっている必要があります。例えば

パターン 51 にプログラムが移ってきた時点で cnt1 が 1000 であったら、1 回目の比較で cnt1 は 100 以上と判断して、すぐにパターン 52 に移ってしまいます。0.1 秒どころか、1 回しかパターン 51 を実行しません(約数 $10 \mu s$)。そこで、パターンをもう一つ増やします。パターン 51 でブレーキをかけ cnt1 をクリア、パターン 52 で 0.1 秒たつたかチェックするようにします。

再度まとめると、下記のようになります。

パターン 51 で行うこと	<ul style="list-style-type: none"> LED2 を点灯 ハンドルを 0 度に 左右モータ PWM を 0%にしてブレーキをかける パターンを次へ移す cnt1 をクリア
パターン 52 で行うこと	<ul style="list-style-type: none"> cnt1 が 100 以上になったなら、次のパターンへ移す

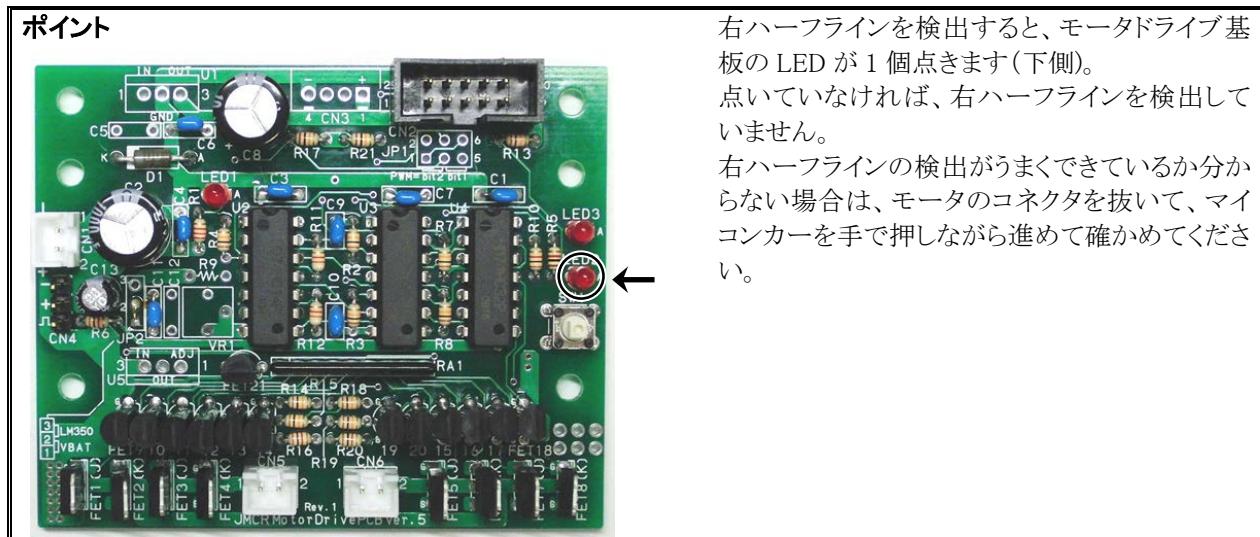
上記にしたがって再度プログラムを作ってみます。

```

349 :     case 51:
350 :         /* 右ハーフライン検出時の処理 */
351 :         led_out( 0x2 );
352 :         handle( 0 );
353 :         motor( 0 , 0 );
354 :         pattern = 52;
355 :         cnt1 = 0;
356 :         break;
357 :
358 :     case 52:
359 :         /* 右ハーフラインを読み飛ばす */
360 :         if( cnt1 > 100 ) {
361 :             pattern = 53;
362 :             cnt1 = 0;
363 :         }
364 :         break;

```

右ハーフラインを検出してから徐行して、トレイス開始部分までプログラムが完成しました。

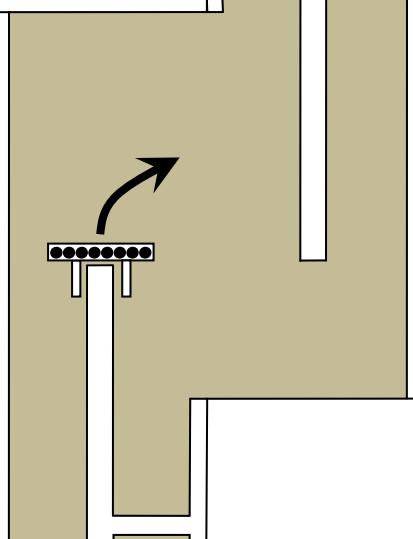
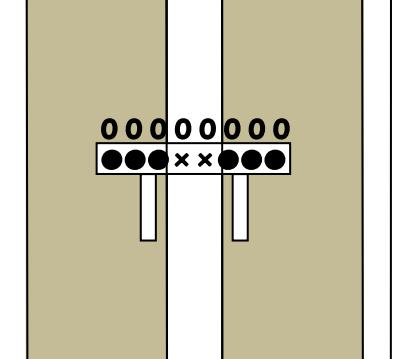
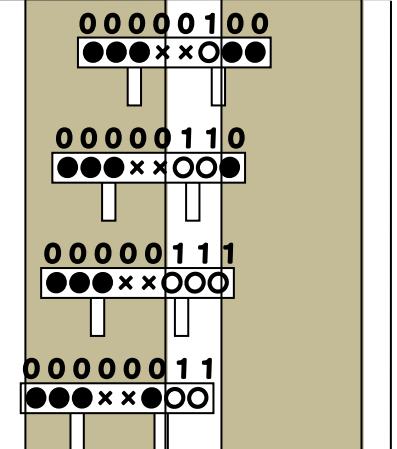


6.4.40 パターン 53:右ハーフライン後のトレース

パターン 51、52 では、右ハーフラインを検出後、ブレーキを 0.1 秒かけ右ハーフラインを通過させました。パターン 53 では、その後の処理を行います。

右ハーフラインを過ぎたので、後は中心線が無くなつたかどうかチェックしながら進んでいきます。また中心線が無くなるまでの間、直線をトレースしなければいけませんので通常トレースも必要です。

今回は、下図のように考えました。

 <p>0x00 (8個すべてチェック)</p>	<p>右ハーフライン検出後、トレースしていき中心線が無くなると、8 個のセンサ状態が左図のように「0x00」になりました。この状態を検出すると、右曲げを開始します。 このとき、右に曲がるので、右モータの回転数を少なく、左モータの回転数を多めにすることは予想できます。実際に何%にすればよいかは、マイコンカーのスピードやタイヤの滑り具合、サーボの反応速度によって変わるので、実際のマイコンカーを見て決めます。ここでは下記のように決め、後は実際に走らせて決めたいと思います。</p> <p>ハンドル: 15 度 左モータ: 40% 右モータ: 31%</p> <p>その後、パターン 54 へ移ります。</p>
 <p>0x00</p>	<p>直進時、センサの状態は「0x00」です。これを直進状態と判断します。ハンドルをまっすぐにしなければいけないのは、疑いの余地がありません。問題はモータの PWM 値です。こちらは実際に走らせなればどのくらいのスピードになるのか何とも言えません。中心線が無くなつたら曲がれるスピードにしなければいけません。とりあえず 40%にしておき、実際に走らせて微調整することにします。まとめると下記のようになります。</p> <p>ハンドル: 0 度 左モータ: 40% 右モータ: 40%</p>
 <p>0x04 0x06 0x07 0x03</p>	<p>マイコンカーが左に寄ったときを考えています。中心から少しずつ左へずらしていくと左図のように 4 つの状態になりました。センサの状態はもっと左へ寄ることも考えられますが、右ハーフラインの後は直線しかないと分かっているので、これ以上センサ状態は増やさないでおきます。 動作は、左へ寄っているので右へハンドルを切れます。切り角が小さすぎるとそれが大きいときに戻りきれなくなり、大きすぎるとセンサが左右にばたばた振れてしまいます。ちょうど良い角度調整は難しいです。今回は、とりあえずどの状態も 8 度にします。まとめると下記のようになります。</p> <p>ハンドル: 8 度 左モータ: 40% 右モータ: 35%</p>

 00100000 01100000 11100000 11000000	<p>マイコンカーが右に寄ったときを考えています。中心から少しずつ、右へずらしていくと左図のようになります。センサの状態はもっと右へ寄ることも考えられますが、右ハーフラインの後は直線しかないと分かっているので、これ以上センサ状態は増やさないでおきます。</p> <p>動作は、右へ寄っているので左へハンドルを切ります。切り角が小さすぎるとそれが大きいときに戻りきれなくなり、大きすぎるとセンサが左右にばたばた振れてしまいます。ちょうど良い角度調整は難しいです。今回は、とりあえずどの状態も-8 度にします。まとめると下記のようになります。</p> <p>ハンドル:-8 度 左モータ:35% 右モータ:40%</p>
--	--

ポイントは、中心線があるかどうかのチェックは 8 個のセンサすべてを使用することです。他は「MASK3_3」でマスクして中心の 2 個のセンサは使用しません。

プログラム化すると下記のようになります。

```

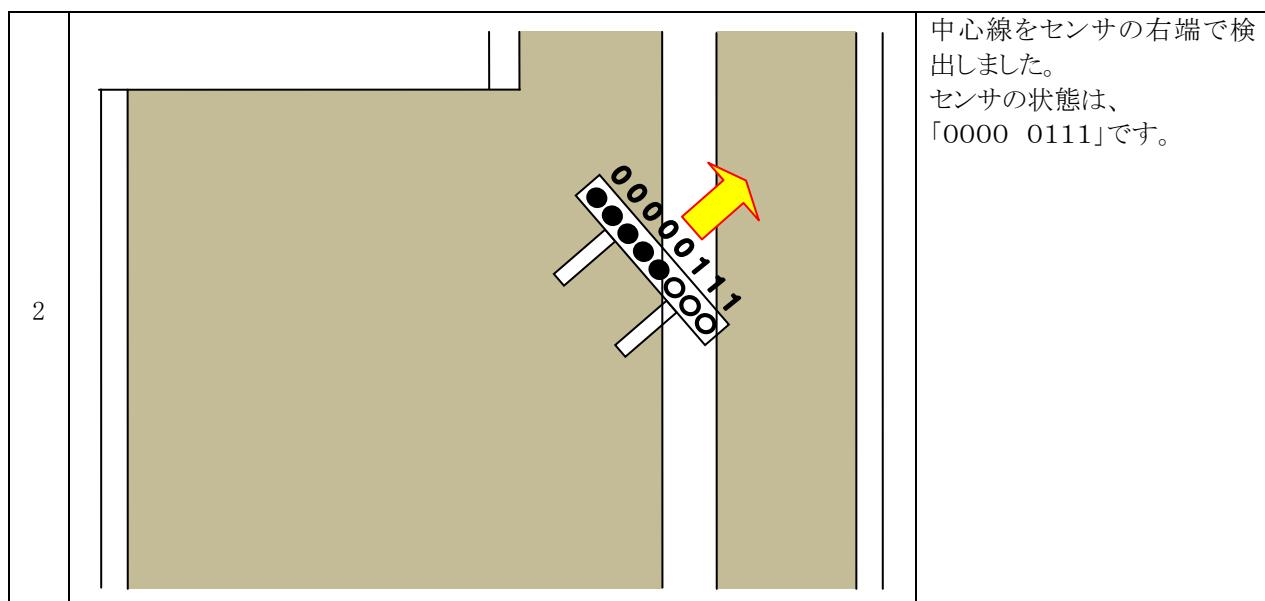
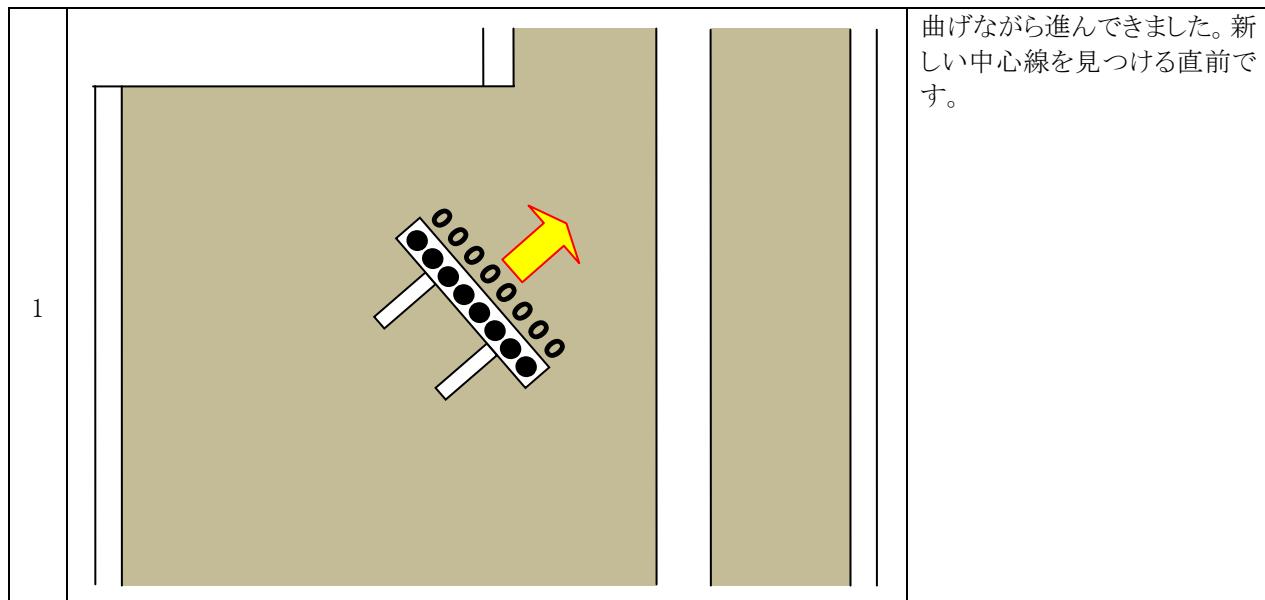
366 :     case 53:
367 :         /* 右ハーフライン後のトレース、レーンチェンジ */
368 :         if( sensor_inp(MASK4_4) == 0x00 ) {
369 :             handle( 15 );
370 :             motor( 40 ,31 );
371 :             pattern = 54;
372 :             cnt1 = 0;
373 :             break;
374 :         }
375 :         switch( sensor_inp(MASK3_3) ) {
376 :             case 0x00:
377 :                 /* センタ→まっすぐ */
378 :                 handle( 0 );
379 :                 motor( 40 ,40 );
380 :                 break;
381 :             case 0x04: } case を続けて書くと
382 :             case 0x06: } 0x04 または 0x06 または 0x07 または 0x03 のとき
383 :             case 0x07: } という意味になります。
384 :             case 0x03: }
385 :                 /* 左寄り→右曲げ */
386 :                 handle( 8 );
387 :                 motor( 40 ,35 );
388 :                 break;
389 :             case 0x20: } case を続けて書くと
390 :             case 0x60: } 0x20 または 0x60 または 0xe0 または 0xc0 のとき
391 :             case 0xe0: } という意味になります。
392 :             case 0xc0: }
393 :                 /* 右寄り→左曲げ */
394 :                 handle( -8 );
395 :                 motor( 35 ,40 );
396 :                 break;
397 :             default:
398 :                 break;
399 :         }
400 :         break;

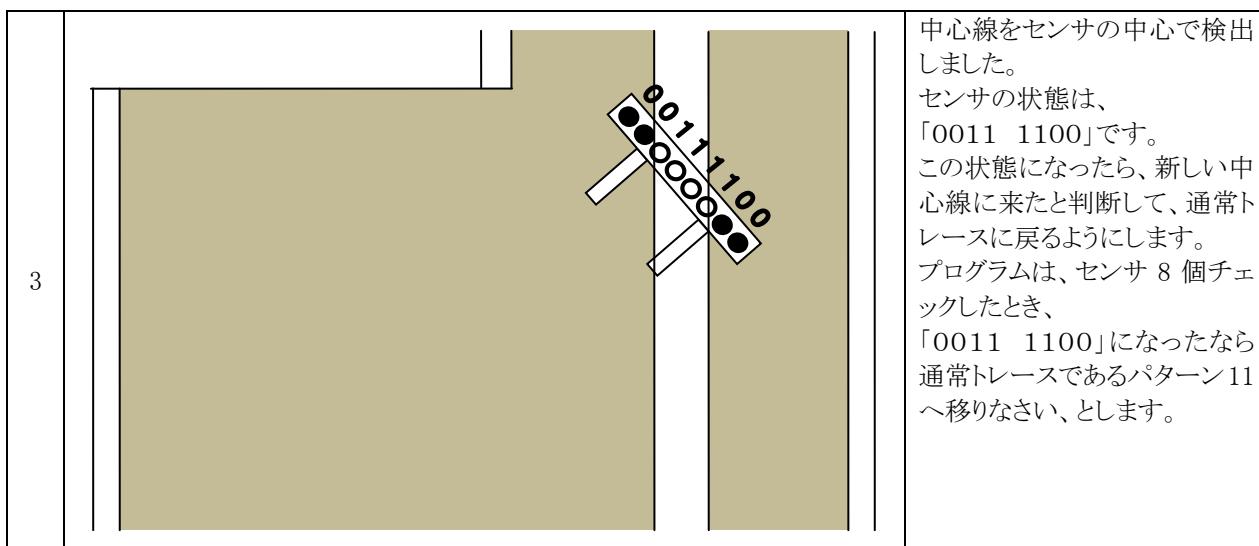
```

6.4.41 パターン 54: 右レーンチェンジ終了のチェック

パターン 53 でセンサ 8 個が「0x00」になると、右レーンチェンジ開始と判断してハンドルを右に 15 度曲げて進みます。次の問題が出てきます。「いつまで右に曲げ続けるか」ということです。この部分のプログラムが、パターン 54 です。

パターン 54 は、右側にある新しい中心線まで進みます。新しい中心線を見つけたら、その中心線をトレースしていきます。これで右レーンチェンジ処理は完了です。では、新しい中心線と見なすセンサ状態はどのような状態でしょうか。



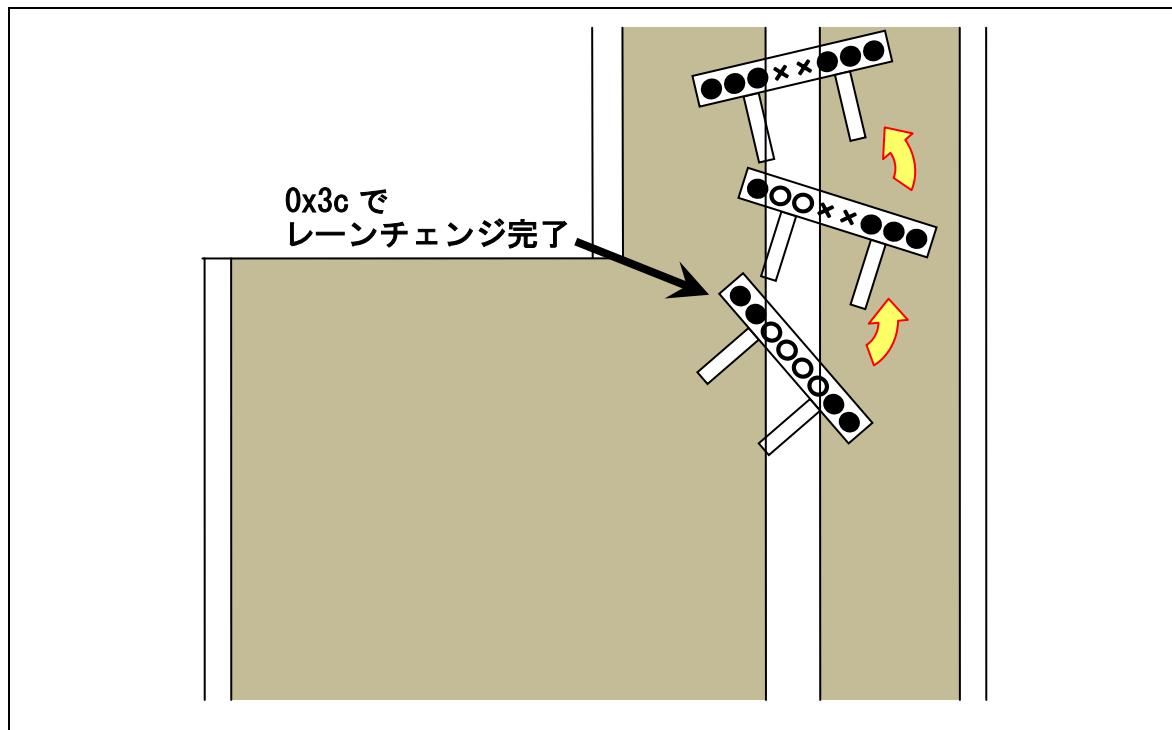


上記の考え方で、プログラムします。

```

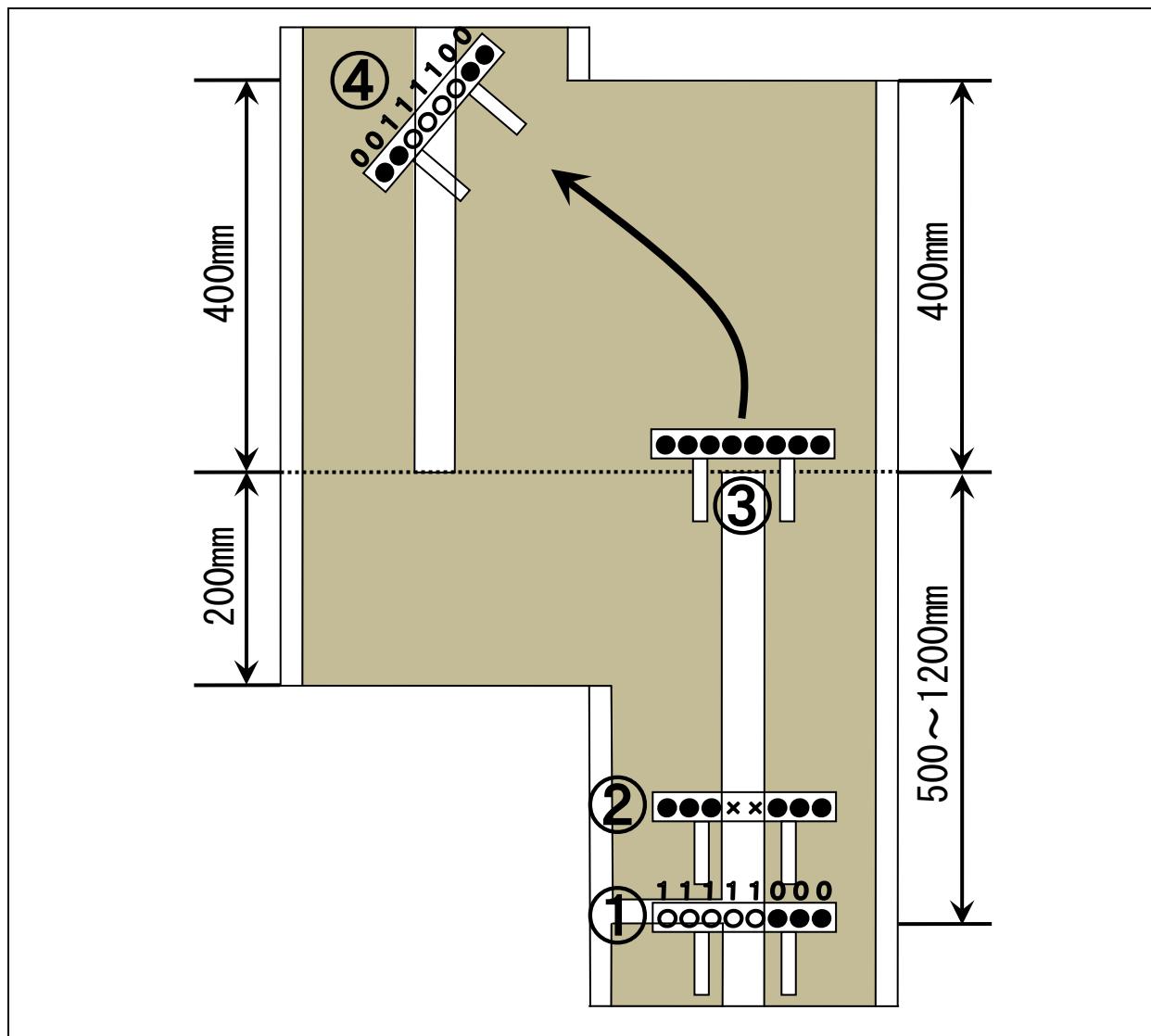
402 :     case 54:
403 :         /* 右レーンチェンジ終了のチェック */
404 :         if( sensor_inp( MASK4_4 ) == 0x3c ) {
405 :             led_out( 0x0 );
406 :             pattern = 11;
407 :             cnt1 = 0;
408 :         }
409 :         break;
    
```

右ハーフラインを検出したときに LED を点灯させたので、405 行で消灯させてからパターン 11 へ移るようにします。「0x3c」を検出して、パターン 11 に移った後の状態を下記に示します。右に角度がついた状態ですが、パターン 11 の処理で中心に復帰していきます。



6.4.42 左レーンチェンジ概要

パターン 61 から 64 は、左レーンチェンジに関するプログラムになっています。処理の概要を下図に示します。

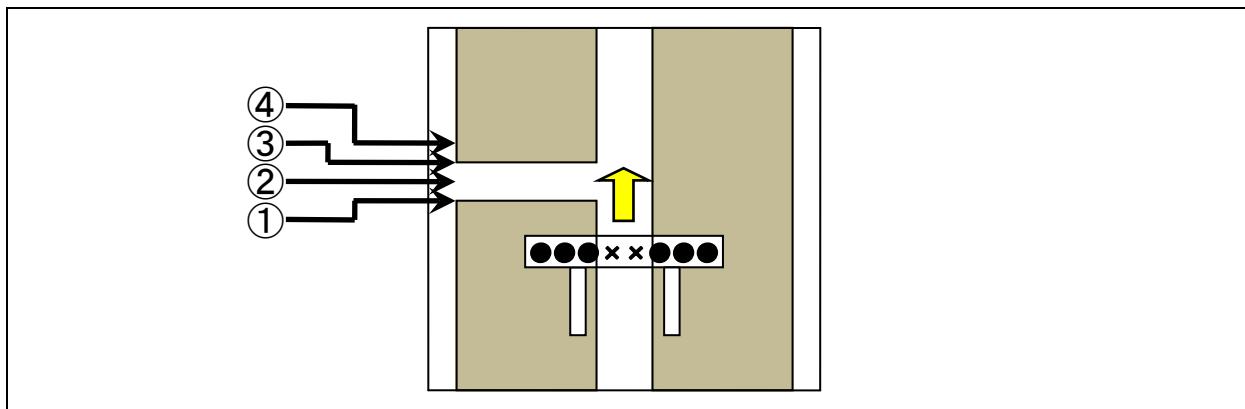


①	check_leftline 関数で左ハーフラインを検出します。500~1200mm 先で、左レーンに移動するために左に曲がらなければいけないのでブレーキをかけます。また、左ハーフライン通過時にセンサが誤検出しないよう②の位置までセンサは見ません。
②	この位置から徐行開始します。中心線をトレースしながら進んでいきます。
③	中心線が無くなると、左へハンドルを切れます。
④	新しい中心線を検出すると、今度はこの中心線でライントレースを再開します。

このように、左レーンチェンジをクリアします。次から具体的なプログラムの説明をしていきます。

6.4.43 パターン 61: 左ハーフライン検出時の処理

パターン 61 は、左ハーフラインを見つけた瞬間に移ってきます。まず、左ハーフラインを通過させます。左ハーフラインを見つけてから、左ハーフラインを終えるまで、下図のような状態があります。



- ①左ハーフラインの始めの境目
- ②左ハーフラインの白色部分
- ③左ハーフラインの終わりの境目
- ④通常コース、ここから徐行しながらトレース

④に行くまでにコースが、左ハーフラインの始めの境目→白→終わりの境目→黒と変化します。それをプログラムで検出して……、とかなり複雑なプログラムになりそうです。

ちょっと考え方を変えてみます。①の位置から④まで、余裕を見て約 100mm あります（正確にはハーフラインの幅は 20~40mm です）。ほぼ中心線の位置にいるとして 100mm くらいならセンサの状態を見ずに進めても大きくなればなさそうです。マイコンカーキットは距離の検出はできないので、タイマでセンサを読まない時間を作ります。時間については、走行スピードによって変わるので、何とも言えません。とりあえず 0.1 秒として、細かい時間は走らせて微調整することにします。同時に、モータドライブ基板の LED を点灯させ、パターン 61 に入ったことを外部に知らせるようにします。

まとめると、

- LED3 を点灯（クロスラインとは違う点灯にして区別させます）
- ハンドルを 0 度に
- 左右モータ PWM を 0% にしてブレーキをかける
- 0.1 秒待つ
- 0.1 秒たったら次のパターンへ移る

これをパターン 61 でプログラム化します。

```
case 61:
    led_out( 0x1 );
    handle( 0 );
    speed( 0 ,0 );
    if( cnt1 > 100 ) {
        pattern = 62; /* 0.1 秒後パターン 62 ～*/
    }
    break;
```

6. プログラム解説「kit12_38a.c」

完成しました。本当にこれでよいか見直してみます。cnt1 が 100 以上になつたら(100 ミリ秒たつたら)、パターン 62 へ移るようになっています。それはパターン 61 を開始したときに、cnt1 が 0 になっている必要があります。例えばパターン 61 にプログラムが移ってきた時点で cnt1 が 1000 であつたら、1 回目の比較で cnt1 は 100 以上と判断して、すぐにパターン 62 に移ってしまいます。0.1 秒どころか、1 回しかパターン 61 を実行しません(約数 10 μ s)。そこで、パターンをもう一つ増やします。パターン 61 でブレーキをかけ cnt1 をクリア、パターン 62 で 0.1 秒たつたかチェックするようにします。

再度まとめると、下記のようになります。

パターン 61 で行うこと	<ul style="list-style-type: none"> LED3 を点灯 ハンドルを 0 度に 左右モータ PWM を 0%にしてブレーキをかける パターンを次へ移す cnt1 をクリア
パターン 62 で行うこと	<ul style="list-style-type: none"> cnt1 が 100 以上になつたら、次のパターンへ移す

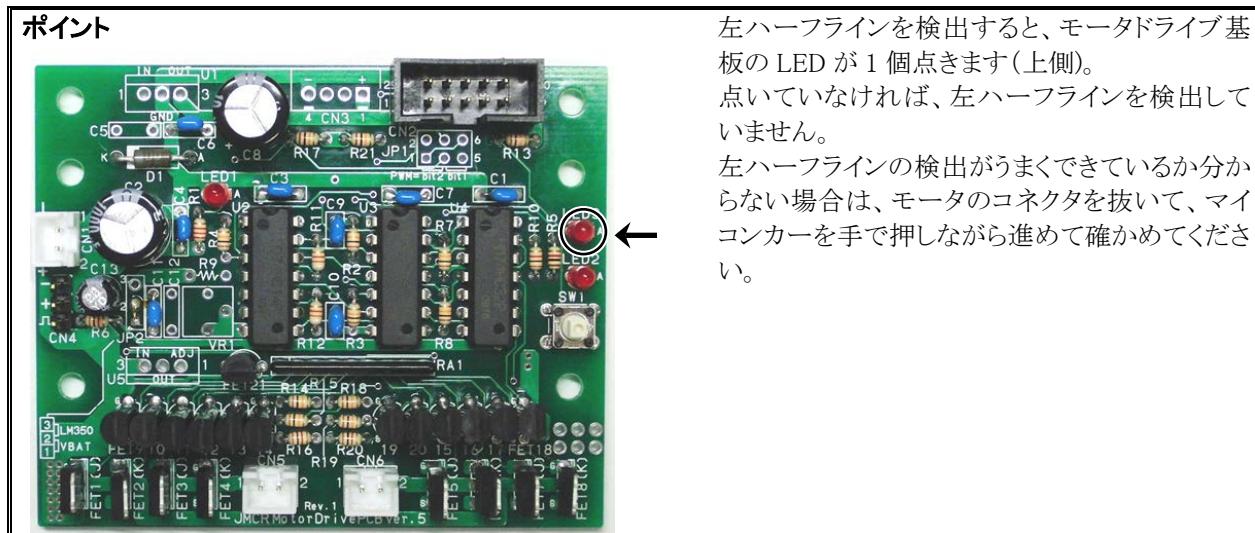
上記にしたがって再度プログラムを作つてみます。

```

411 :     case 61:
412 :         /* 左ハーフライン検出時の処理 */
413 :         led_out( 0x1 );
414 :         handle( 0 );
415 :         motor( 0 ,0 );
416 :         pattern = 62;
417 :         cnt1 = 0;
418 :         break;
419 :
420 :     case 62:
421 :         /* 左ハーフラインを読み飛ばす */
422 :         if( cnt1 > 100 ) {
423 :             pattern = 63;
424 :             cnt1 = 0;
425 :         }
426 :         break;

```

左ハーフラインを検出してから徐行して、トレイス開始部分までプログラムが完成しました。

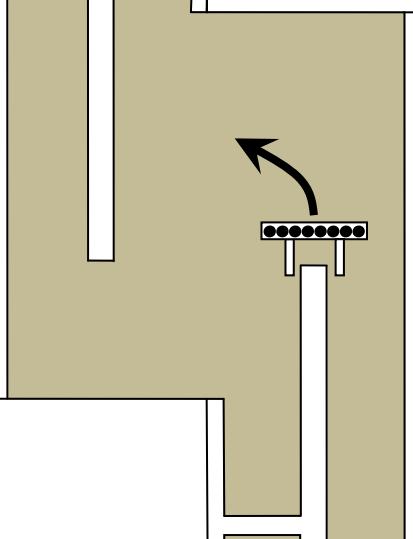
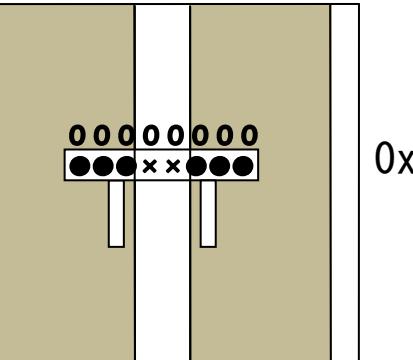
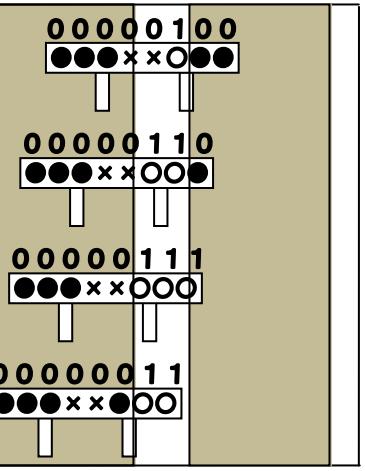


6.4.44 パターン 63: 左ハーフライン後のトレース

パターン 61、62 では、左ハーフラインを検出後、ブレーキを 0.1 秒かけ左ハーフラインを通過させました。パターン 63 では、その後の処理を行います。

左ハーフラインを過ぎたので、後は中心線が無くなつたかどうかチェックしながら進んでいきます。また中心線が無くなるまでの間、直線をトレースしなければいけませんので通常トレースも必要です。

今回は、下図のように考えました。

 <p>0x00 (8個すべてチェック)</p>	<p>左ハーフライン検出後、トレースしていく中心線が無くなると、8 個のセンサ状態が左図のように「0x00」になりました。この状態を検出すると、左曲げを開始します。 このとき、左に曲がるので、左モータの回転数を少なく、右モータの回転数を多めにすることは予想できます。実際に何%にすればよいかは、マイコンカーのスピードやタイヤの滑り具合、サーボの反応速度によって変わるので、実際のマイコンカーを見て決めます。ここでは下記のように決め、後は実際に走らせて決めたいと思います。</p> <p>ハンドル:-15 度 左モータ:31% 右モータ:40%</p> <p>その後、パターン 64 へ移ります。</p>
 <p>0x00</p>	<p>直進時、センサの状態は「0x00」です。これを直進状態と判断します。ハンドルをまっすぐにしなければいけないのは、疑いの余地がありません。問題はモータの PWM 値です。こちらは実際に走らせなければどのくらいのスピードになるのか何とも言えません。中心線が無くなつたら曲がれるスピードにしなければいけません。とりあえず 40%にしておき、実際に走らせて微調整することにします。まとめると下記のようになります。</p> <p>ハンドル:0 度 左モータ:40% 右モータ:40%</p>
 <p>0x04 0x06 0x07 0x03</p>	<p>マイコンカーが左に寄ったときを考えています。中心から少しずつ左へずらしていくと左図のように 4 つの状態になりました。センサの状態はもっと左へ寄ることも考えられますが、右ハーフラインの後は直線しかないと分かっているので、これ以上センサ状態は増やさないであります。 動作は、左へ寄っているので右へハンドルを切れます。切り角が小さすぎるとそれが大きいときに戻りきれなくなり、大きすぎるとセンサが左右にばたばた振れてしまいます。ちょうど良い角度調整は難しいです。今回は、とりあえずどの状態も 8 度にします。まとめると下記のようになります。</p> <p>ハンドル:8 度 左モータ:40% 右モータ:35%</p>

6. プログラム解説「kit12_38a.c」

	<p>マイコンカーが右に寄ったときを考えています。中心から少しずつ、右へずらしていくと左図のように 4 つの状態になりました。センサの状態はもっと右へ寄ることも考えられますが、右ハーフラインの後は直線しかないと分かっているので、これ以上センサ状態は増やさないでおきます。</p> <p>動作は、右へ寄っているので左へハンドルを切ります。切り角が小さすぎるとそれが大きいときに戻りきれなくなり、大きすぎるとセンサが左右にばたばた振れてしまいます。ちょうど良い角度調整は難しいです。今回は、とりあえずどの状態も -8 度にします。まとめると下記のようになります。</p> <p>ハンドル: -8 度 左モータ: 35% 右モータ: 40%</p>
--	---

ポイントは、中心線があるかどうかのチェックは 8 個のセンサすべてを使用することです。他は「MASK3_3」でマスクして中心の 2 個のセンサは使用しません。

プログラム化すると下記のようになります。

```

428 :     case 63:
429 :         /* 左ハーフライン後のトレース、レーンチェンジ */
430 :         if( sensor_inp(MASK4_4) == 0x00 ) {
431 :             handle( -15 );
432 :             motor( 31 , 40 );
433 :             pattern = 64;
434 :             cnt1 = 0;
435 :             break;
436 :         }
437 :         switch( sensor_inp(MASK3_3) ) {
438 :             case 0x00:
439 :                 /* センタ→まっすぐ */
440 :                 handle( 0 );
441 :                 motor( 40 , 40 );
442 :                 break;
443 :             case 0x04:
444 :             case 0x06:
445 :             case 0x07:
446 :             case 0x03: } case を続けて書くと
447 :                 /* 左寄り→右曲げ */
448 :                 handle( 8 );
449 :                 motor( 40 , 35 );
450 :                 break;
451 :             case 0x20:
452 :             case 0x60:
453 :             case 0xe0:
454 :             case 0xc0: } case を続けて書くと
455 :                 /* 右寄り→左曲げ */
456 :                 handle( -8 );
457 :                 motor( 35 , 40 );
458 :                 break;
459 :             default:
460 :                 break;
461 :         }
462 :         break;

```

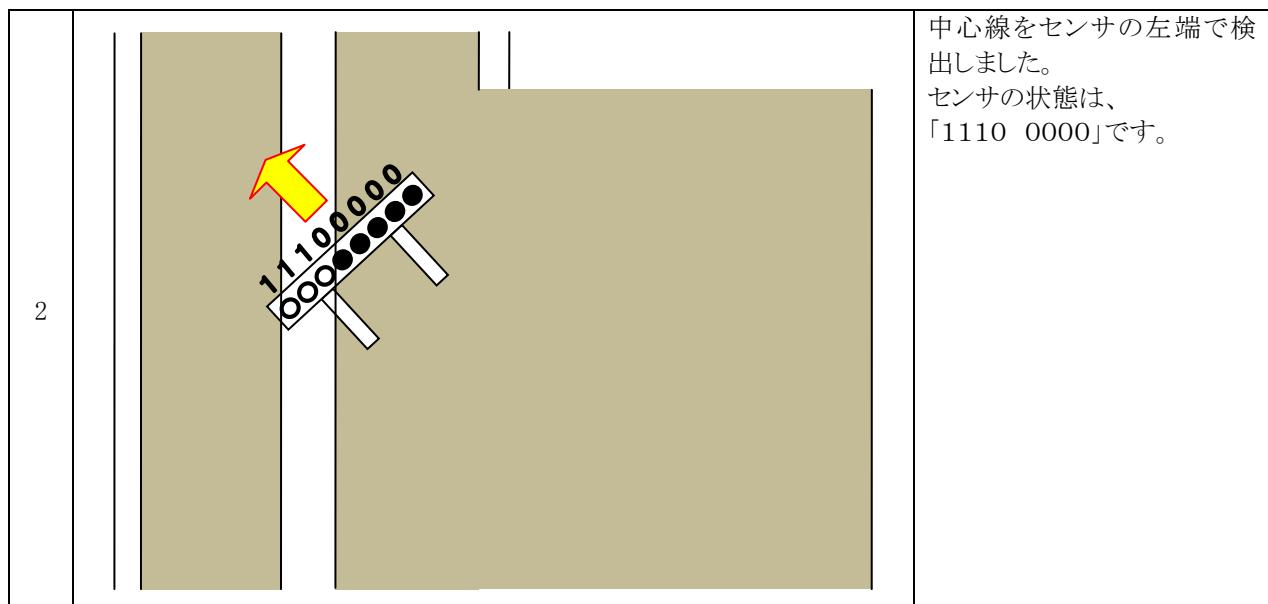
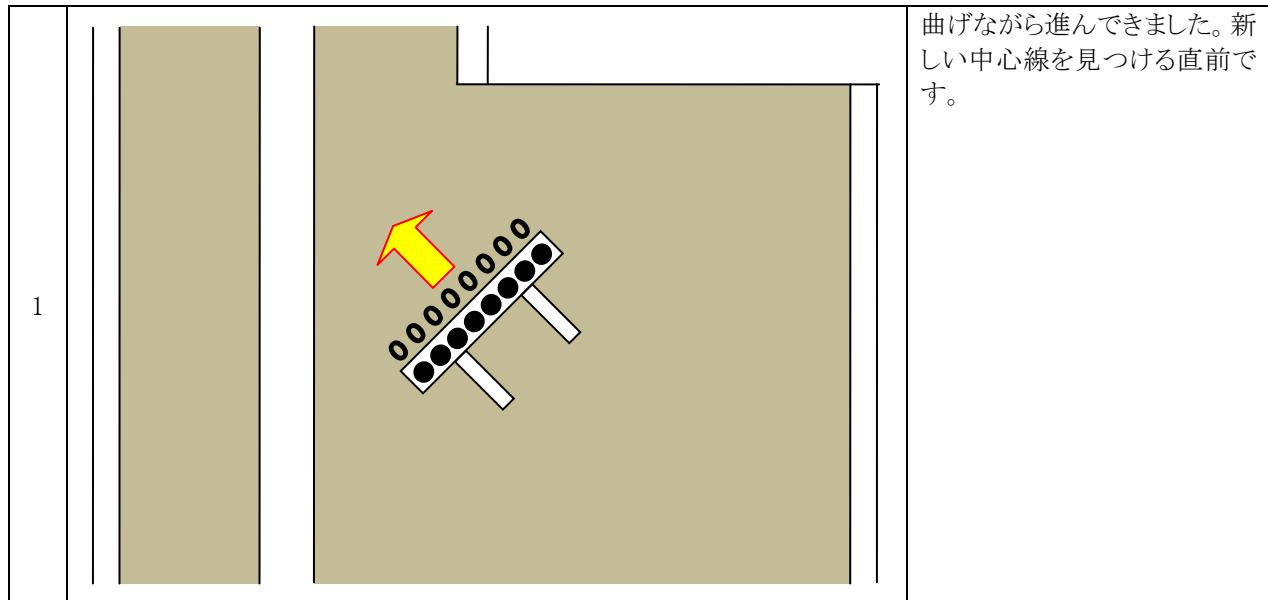
} 0x04 または 0x06 または 0x07 または 0x03 のとき
という意味になります。

} 0x20 または 0x60 または 0xe0 または 0xc0 のとき
という意味になります。

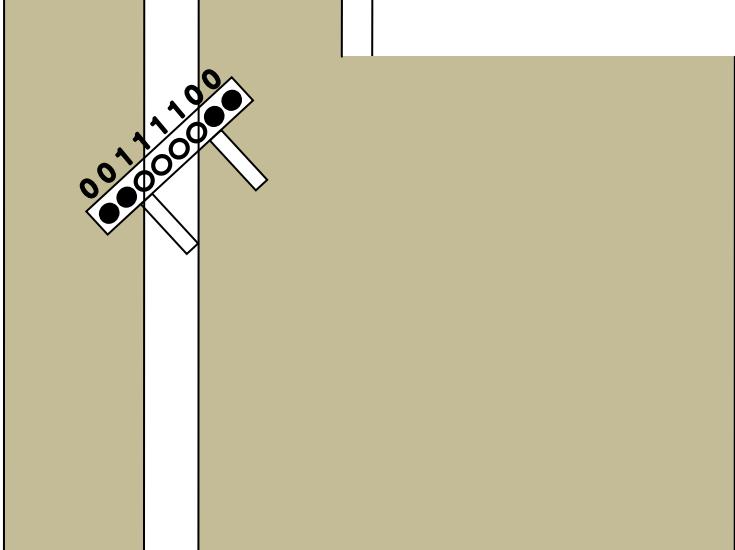
6.4.45 パターン 64: 左レーンチェンジ終了のチェック

パターン 63 でセンサ 8 個が「0x00」になると、左レーンチェンジ開始と判断してハンドルを左に 15 度曲げて進みます。次の問題が出てきます。「いつまで左に曲げ続けるか」ということです。この部分のプログラムが、パターン 64 です。

パターン 64 は、左側にある新しい中心線まで進みます。新しい中心線を見つけたら、その中心線をトレースしていきます。これで左レーンチェンジ処理は完了です。では、新しい中心線と見なすセンサ状態はどのような状態でしょうか。



6. プログラム解説「kit12_38a.c」

3		<p>中心線をセンサの中心で検出しました。 センサの状態は、「0011 1100」です。 この状態になったら、新しい中心線に来たと判断して、通常トレースに戻るようにします。 プログラムは、センサ 8 個チェックしたとき、「0011 1100」になったなら通常トレースであるパターン 11 へ移りなさい、とします。</p>
---	--	--

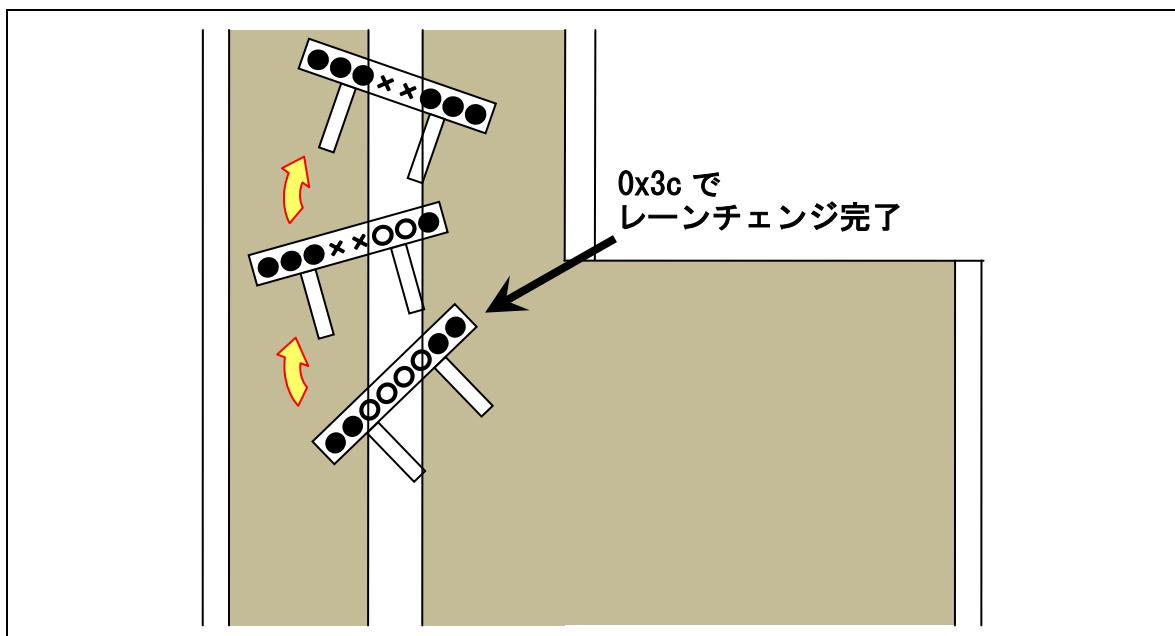
上記の考え方で、プログラムします。

```

464 :     case 64:
465 :         /* 左レーンチェンジ終了のチェック */
466 :         if( sensor_inp( MASK4_4 ) == 0x3c ) {
467 :             led_out( 0x0 );
468 :             pattern = 11;
469 :             cnt1 = 0;
470 :         }
471 :         break;

```

左ハーフラインを検出したときに LED を点灯させたので、467 行で消灯させてからパターン 11 へ移るようにします。「0x3c」を検出して、パターン 11 に移った後の状態を下記に示します。左に角度がついた状態ですが、パターン 11 の処理で中心に復帰していきます。

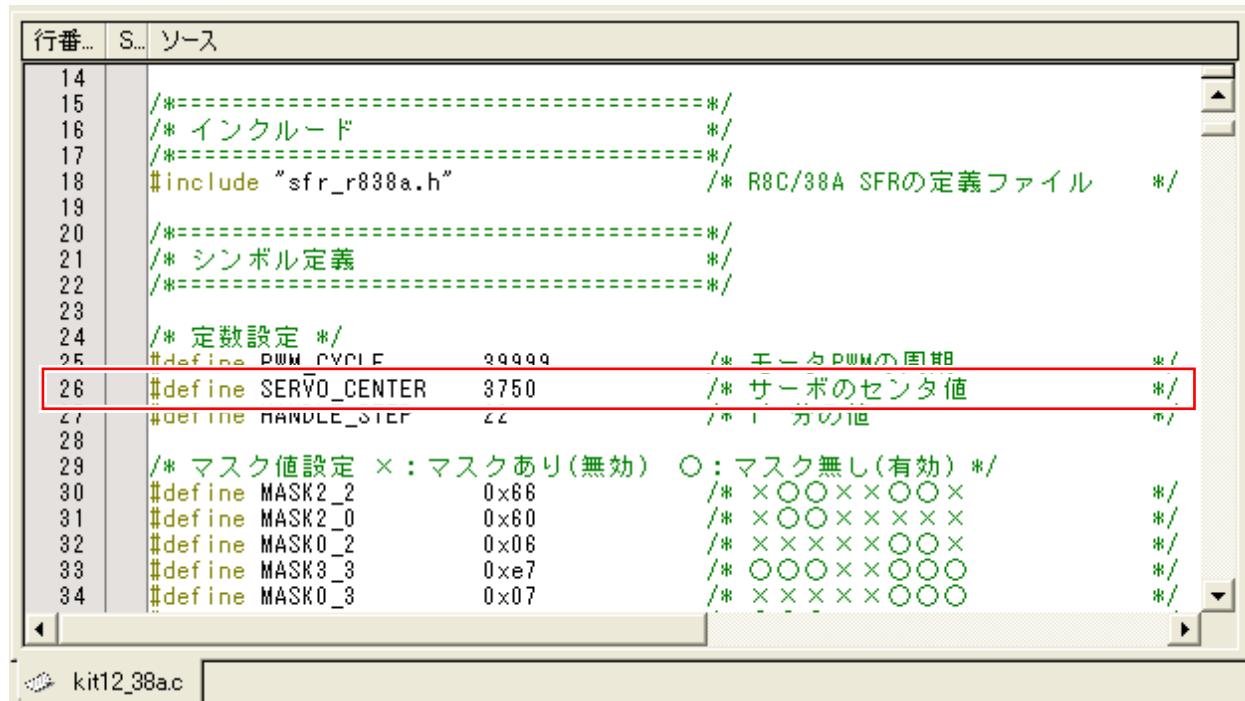


7. サーボセンタと最大切れ角の調整

7.1 概要

kit12_38a.mot を書き込んでマイコンカーの電源を入れても、ほとんどの場合ハンドルが 0 度(まっすぐ)になります。これは、人の指紋が一人一人違うのと同じで、サーボを「まっすぐにしなさい」という数値がサーボ 1 個 1 個違うためです。

そこで、サーボセンタの調整を行います。「kit12_38a.c」の 26 行



```

行番... S... ソース
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 99999           /* エータ PWM の周波 */
26 #define SERVO_CENTER 3750          /* サーボのセンタ値 */
27 #define HANDLE_STER 22             /* ハンドル */
28
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              /* ×××××○○○ */

```

が、サーボセンタの値です。調整は、

- ・ずれに応じて値を調整する(1度あたり 22、値を減らすと左へ、増やすと右へサーボが動きます)
 - ・ビルドする
 - ・RY_R8C38 ボードにプログラムを書き込む
 - ・0 度か確かめる
 - ・0 度でなければやり直し
- という作業を何度も繰り返さなければきっちとした中心になりません。

そこで、パソコンとマイコンカーを通信ケーブルで繋ぎます。調整は、

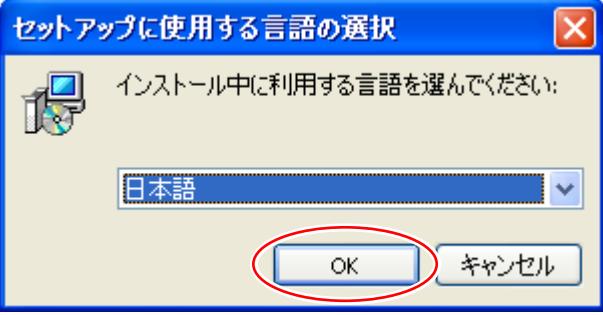
- ・パソコンのキーボードを使いながらサーボのセンタを調整、0 度の値を見つける
- ・値をプログラムに書き込む
- ・ビルドする
- ・RY_R8C38 ボードにプログラムを書き込む

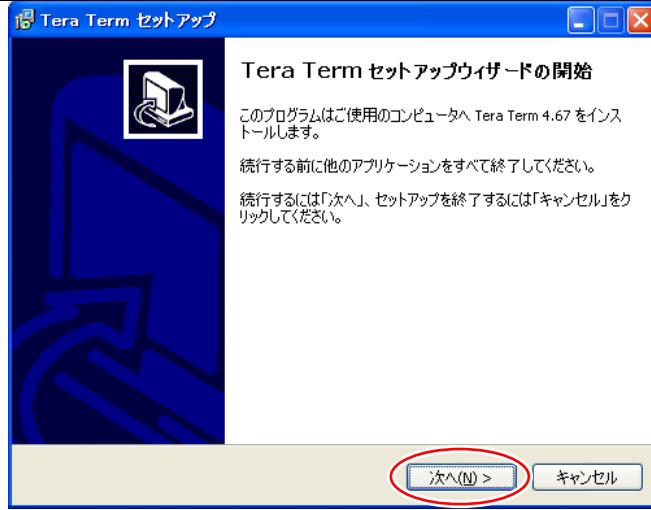
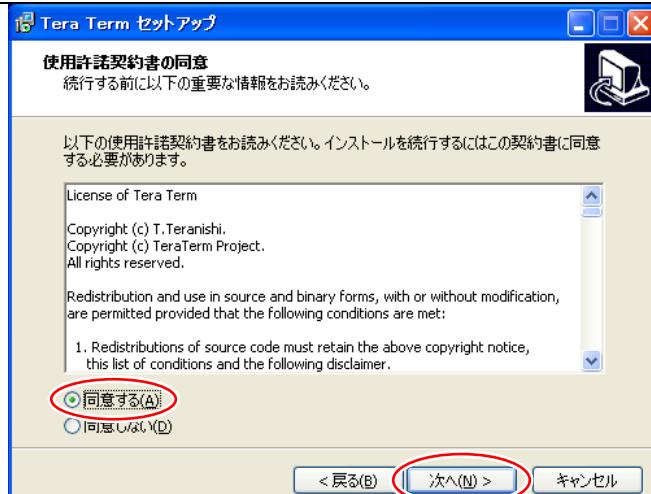
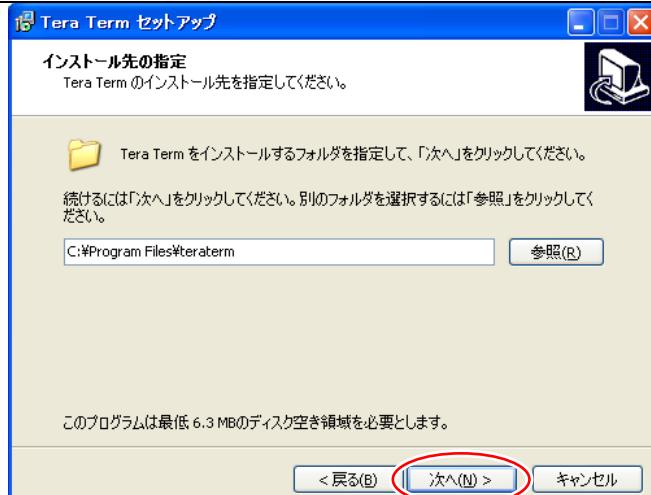
という作業のみで OK です。先ほどより、簡単になりました。本章では、パソコンのキーボードで下記の調整を行います。

- ①サーボセンタの値を簡単に調整しましょう(sioservo1_38a プロジェクトで実施)
- ②右に何度も切れるか、右最大切れ角を見つけましょう(sioservo2_38a プロジェクトで実施)
- ③左に何度も切れるか、左最大切れ角を見つけましょう(sioservo2_38a プロジェクトで実施)

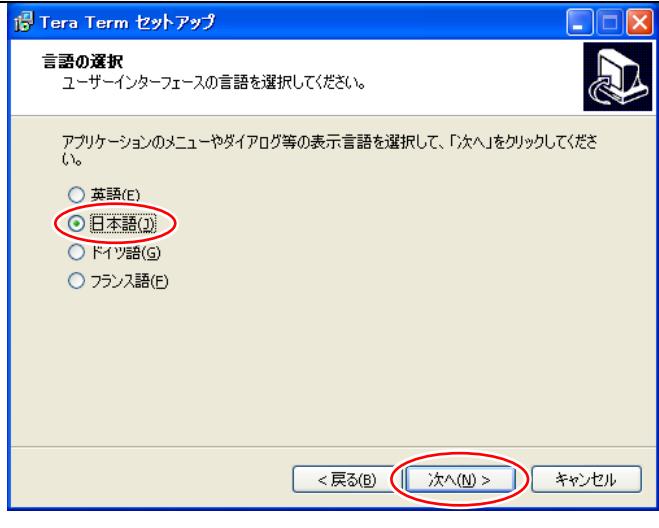
7.2 通信ソフト「Tera Term」をインストールする

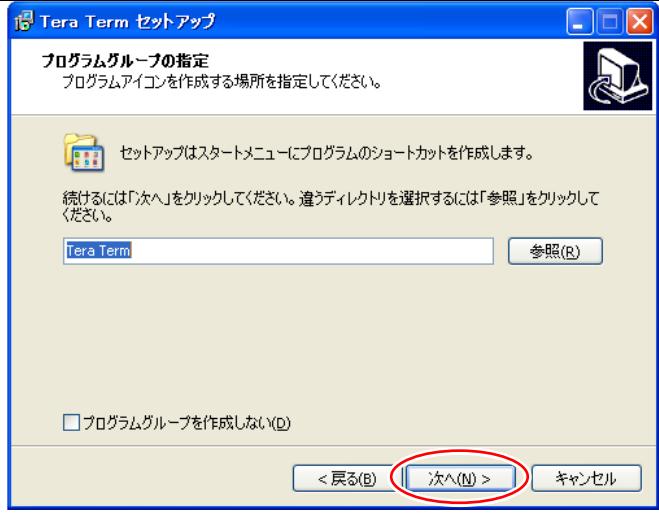
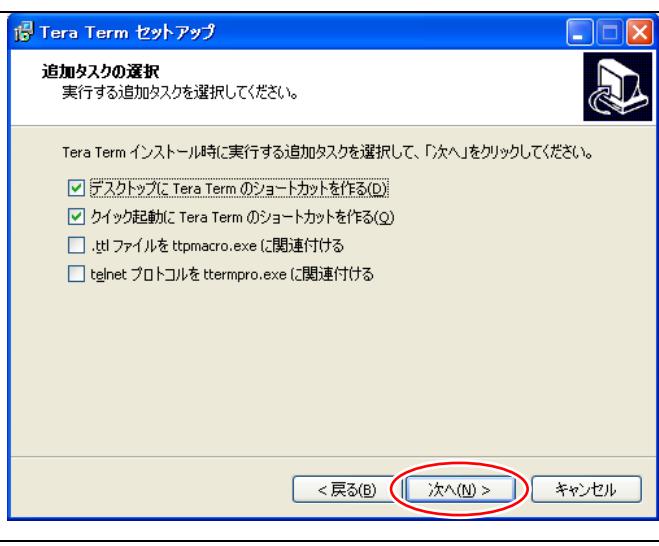
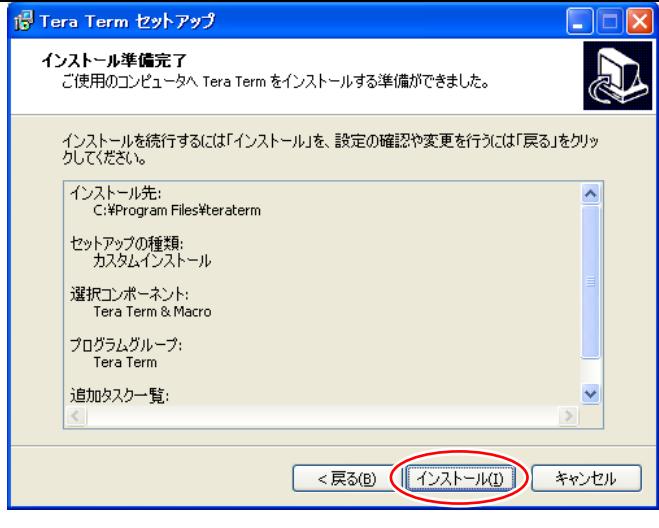
フリーの通信ソフト「**Tera Term**」を使用して、マイコンと通信をおこないます。ここではインストール手順を説明します。既に Tera Term をインストールしている場合は、インストールする必要はありません。

1	 <p>Tera Term</p> <p>v4.67 (10/08/31) UTF-8やSSH/SSH2に対応した「Tera Term Pro」の拡張版</p> <p>NT 2000 XP 2003 Vista 2008 7 フリーソフト</p> <p>TELNETとシリアル接続に対応したターミナルエミュレーター「Tera Term Pro」を、多くの開発者の手で拡張したバージョン。原作者の許可を得てオープンソースで開発されている。「Tera Term Pro」からの一番の変更点は文字コードUTF-8の表示と、SSH/SSH2プロトコルによる接続への対応だが、</p> 	<p>まず、ソフトをダウンロードします。インターネットブラウザで http://www.forest.impress.co.jp/lib/inet/servernt/remote/utf8teraterm.html にアクセスします。</p> <p>※窓の杜という Windows 用オンラインソフト紹介サイト、または同等のサイトでダウンロードします。</p>
2	 <p>Tera Term</p> <p>v4.67 (10/08/31) UTF-8やSSH/SSH2に対応した「Tera Term Pro」の拡張版</p> <p>NT 2000 XP 2003 Vista 2008 7 フリーソフト</p>	<p>「DOWNLOAD」をクリックし、ファイルをダウンロードします。</p>
3	 <p>teraterm-4.67.exe</p>	<p>ダウンロードした「teraterm-4.67.exe」を実行します。</p> <p>※バージョンにより、「4.67」部分は異なります。</p>
4	 <p>セットアップに使用する言語の選択</p> <p>インストール中にご利用する言語を選んでください:</p> <p>日本語</p> <p>OK キャンセル</p>	<p>言語の選択です。OKをクリックします。</p>

5	 <p>Tera Term セットアップ Tera Term セットアップウィザードの開始 このプログラムはご使用のコンピュータへ Tera Term 4.67 をインストールします。 続行する前に他のアプリケーションをすべて終了してください。 続行するには「次へ」、セットアップを終了するには「キャンセル」をクリックしてください。</p>	<p>次へをクリックします。</p>
6	 <p>使用許諾契約書の同意 続行する前に以下の重要な情報を読みください。</p> <p>以下を使用許諾契約書をお読みください。インストールを続行するにはこの契約書に同意する必要があります。</p> <p>License of Tera Term Copyright (c) T.Teranishi. Copyright (c) TeraTerm Project. All rights reserved.</p> <p>Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:</p> <p>1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.</p> <p><input checked="" type="radio"/> 同意する(A) <input type="radio"/> 同意しない(D)</p>	<p>同意する場合は、「同意する」をクリックして、 次へをクリックします。</p>
7	 <p>インストール先の指定 Tera Term のインストール先を指定してください。</p> <p>Tera Term をインストールするフォルダを指定して、「次へ」をクリックしてください。</p> <p>続けるには「次へ」をクリックしてください。別のフォルダを選択するには「参照」をクリックしてください。</p> <p>C:\Program Files\teraterm <input type="button" value="参照(R)"/></p> <p>このプログラムは最低 6.3 MB のディスク空き領域を必要とします。</p>	<p>次へをクリックします。</p>

7. サーボセンタと最大切れ角の調整

8		<p>次へをクリックします。</p>
9		<p>コンポーネントの選択です。どのコンポーネントも使いませんので、すべてのチェックを外します。次へをクリックします。</p>
10		<p>「日本語」を選択して、次へをクリックします。</p>

11		<p>次へをクリックします。</p>
12		<p>追加するタスクを選択して(通常は変更なしで大丈夫です)、次へをクリックします。</p>
13		<p>インストールをクリックします。</p>

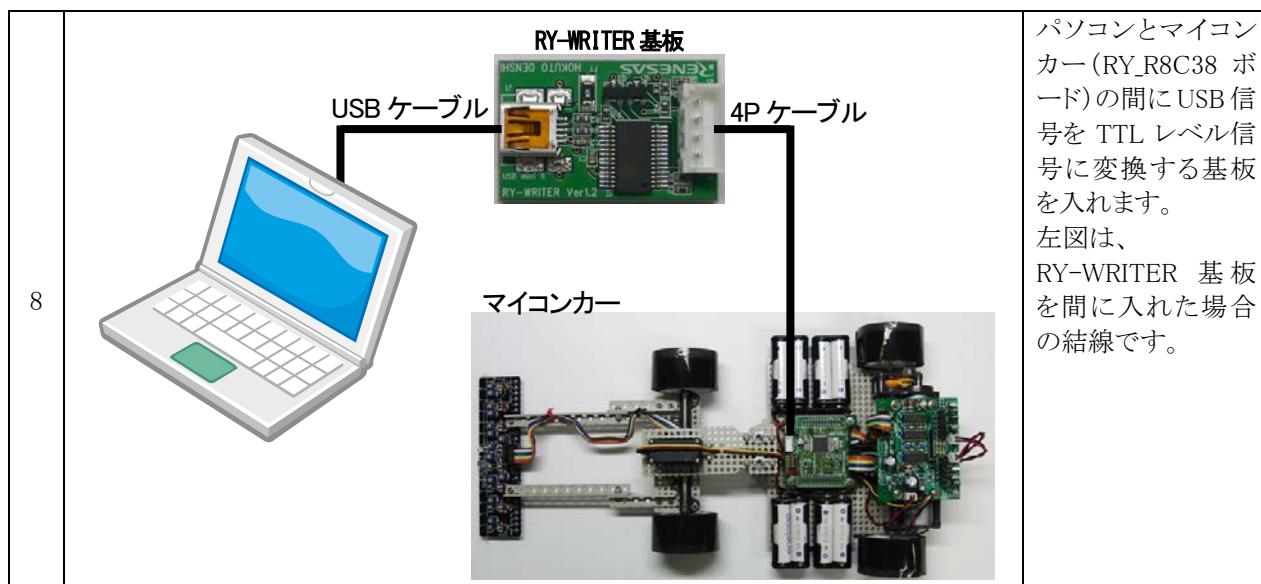
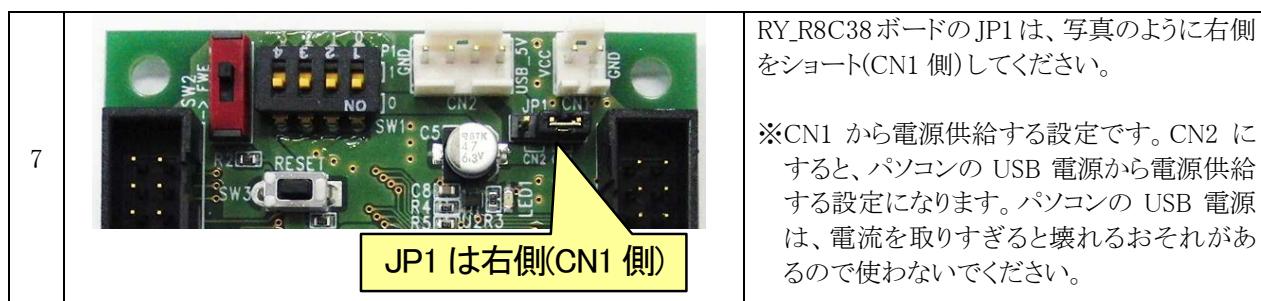
7. サーボセンタと最大切れ角の調整

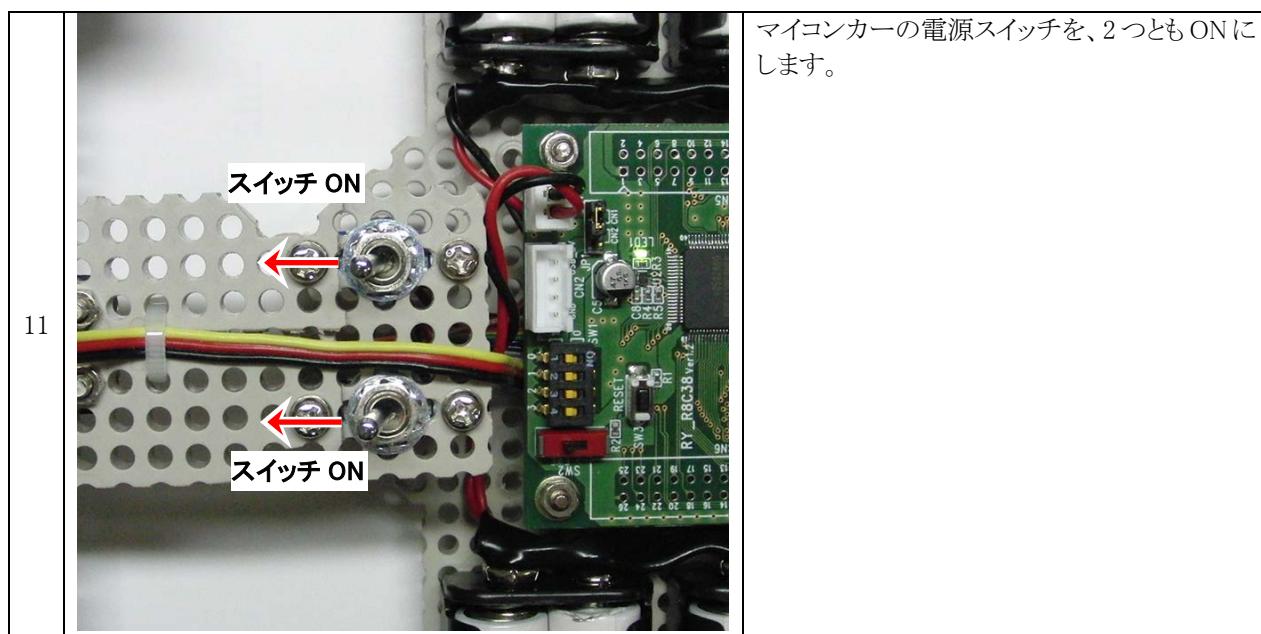
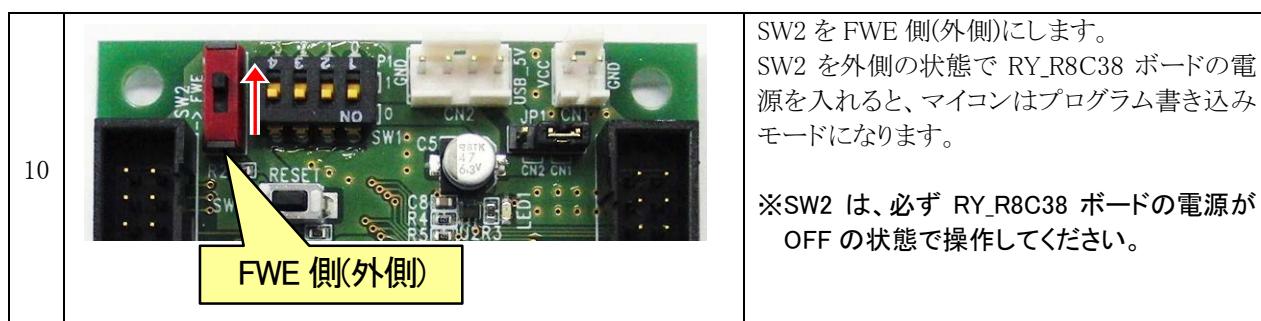
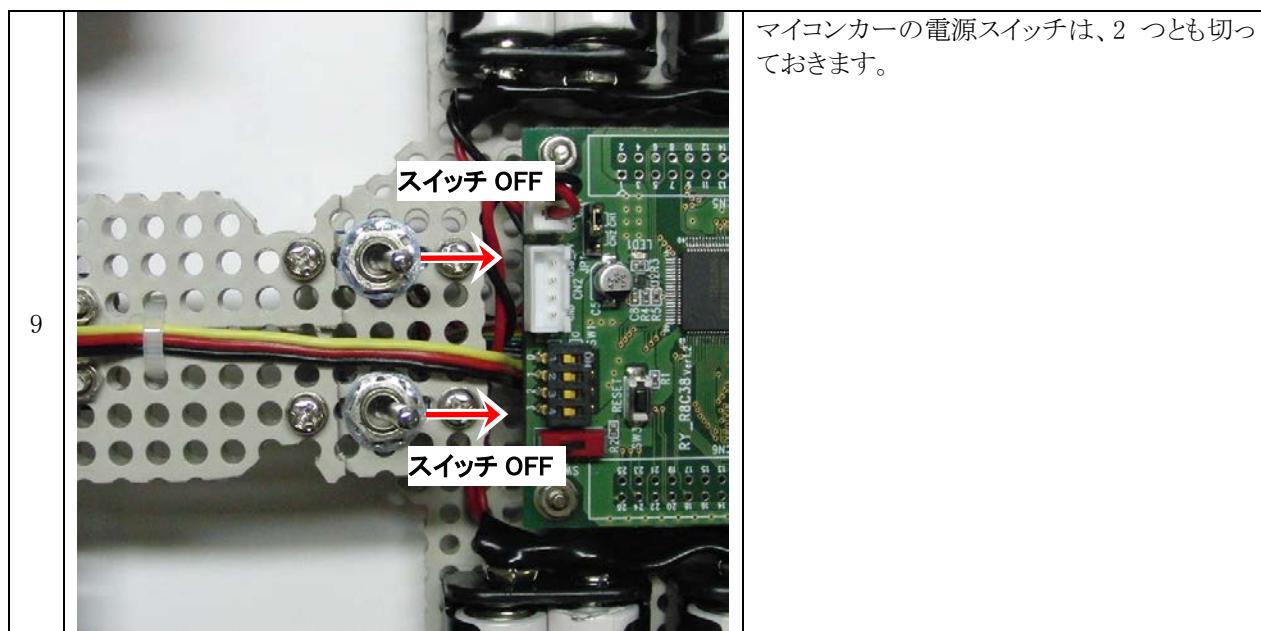
14		インストール中です。
15		完了をクリックして、インストール終了です。

7.3 サーボのセンタを調整する

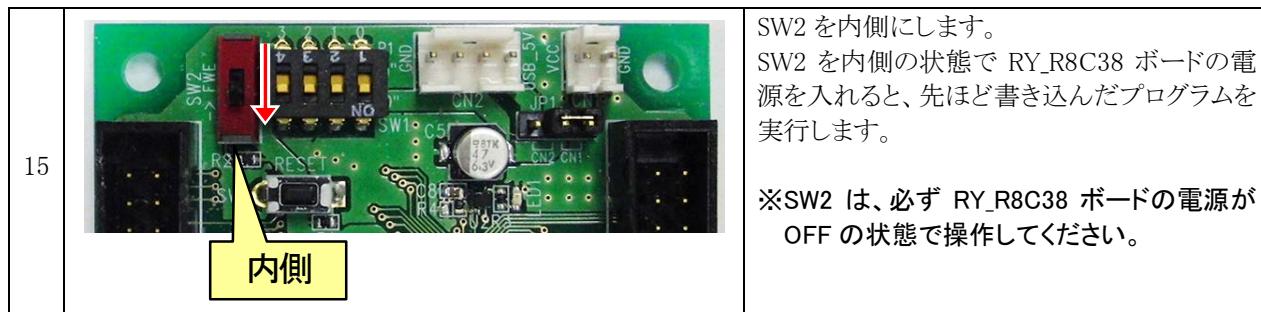
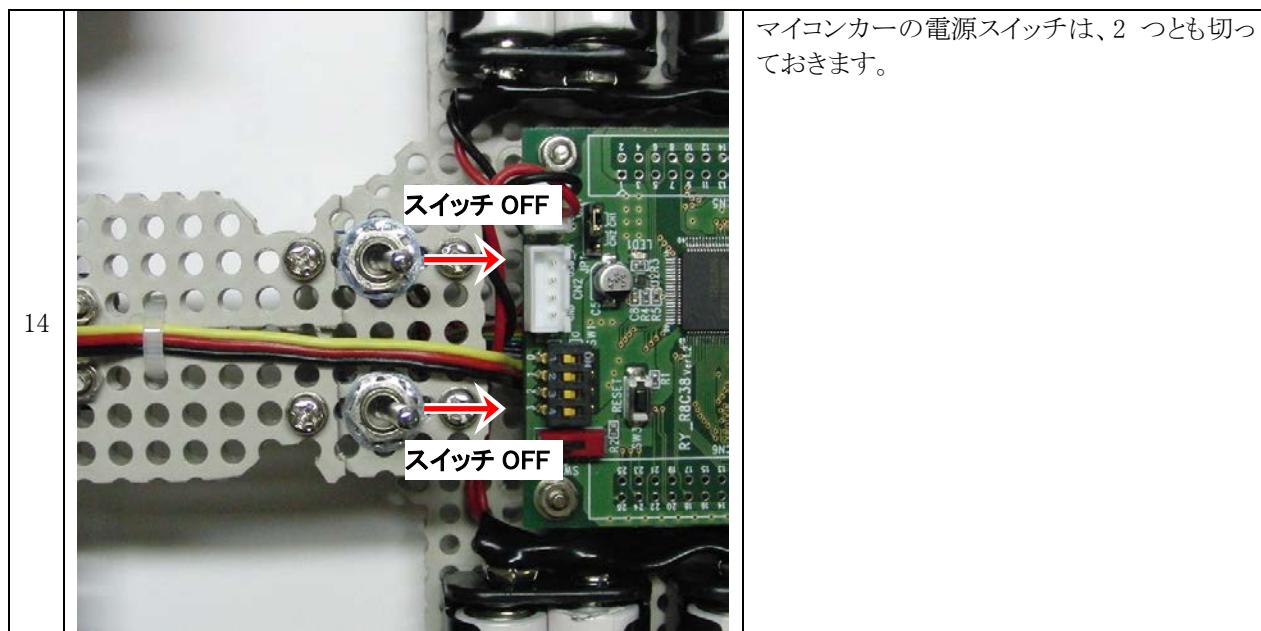
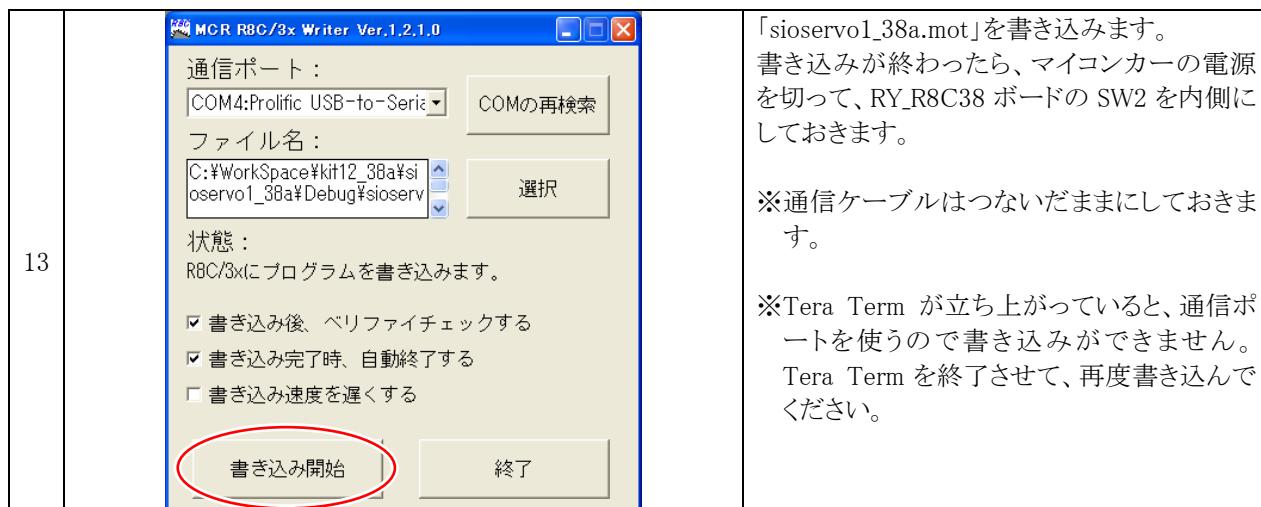
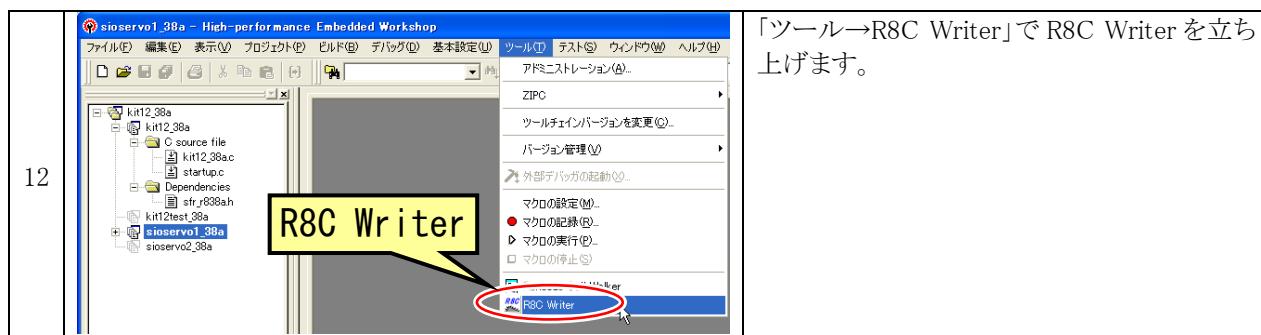
1		ルネサス統合開発環境を実行します。
2		「別のプロジェクトワークスペースを参照する」を選択します。
3		C ドライブ → Workspace → kit12_38a の「kit12_38ahws」を選択します。
4		ワークスペース「kit12_38a」が開きます。
5		プロジェクト「sioservo1_38a」をアクティブプロジェクトに設定します。

7. サーボセンタと最大切れ角の調整





7. サーボセンタと最大切れ角の調整



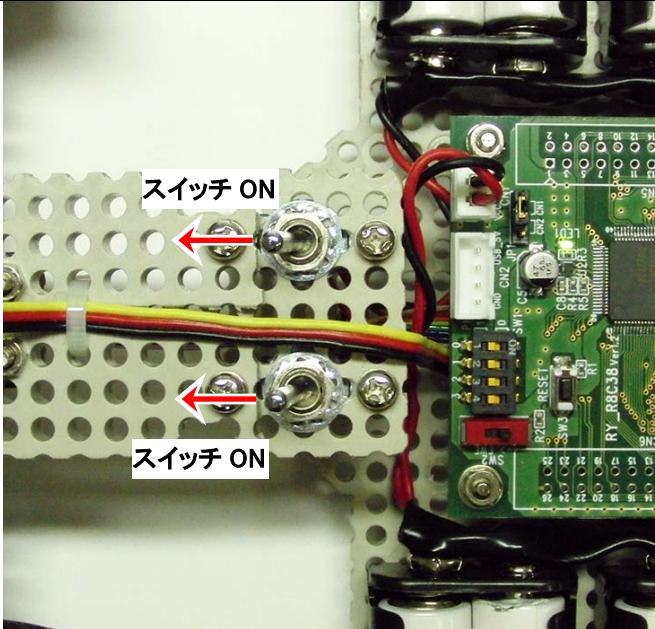
16	 <p>「スタート→すべてのプログラム→Tera Term→Tera Term」を選択します。</p>
----	---

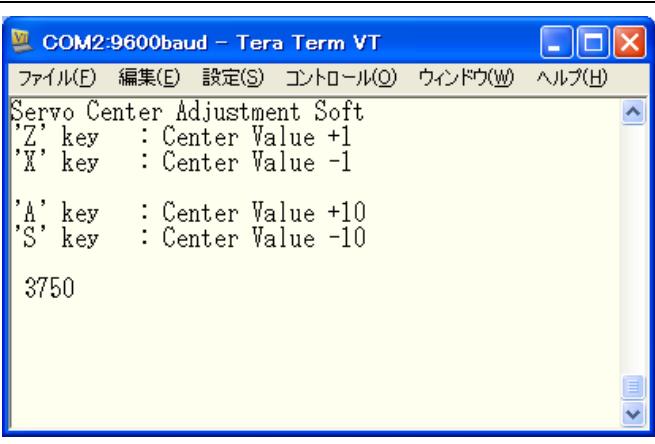
17	 <p>新しい接続のウィンドウが立ち上がります。 「シリアル」を選択します。</p>
----	---

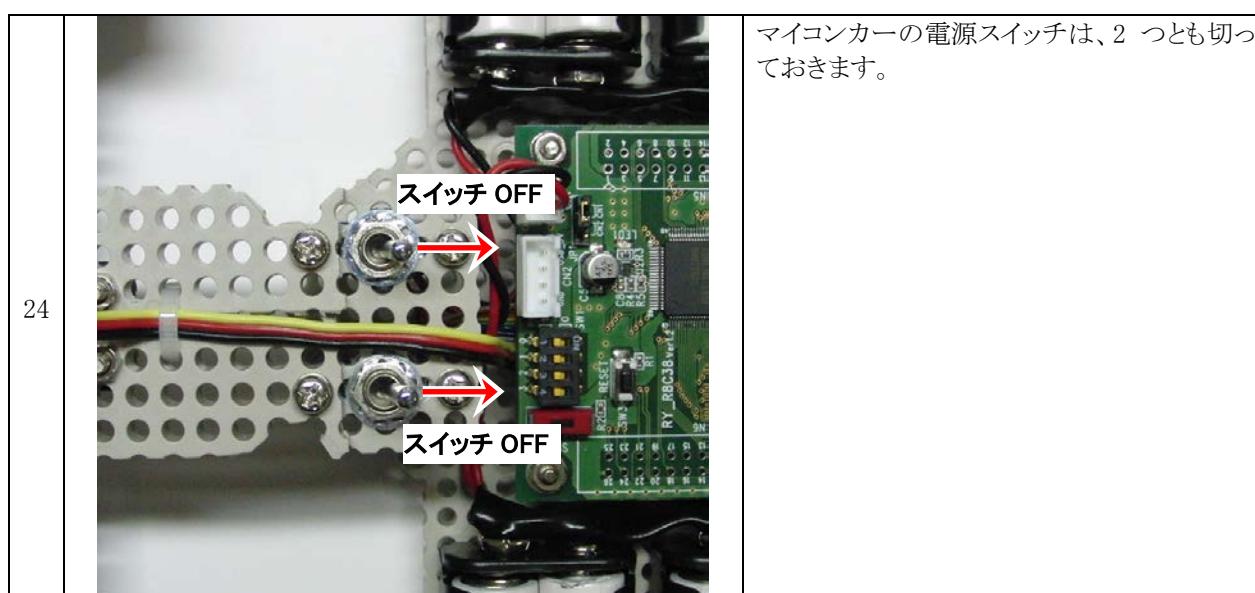
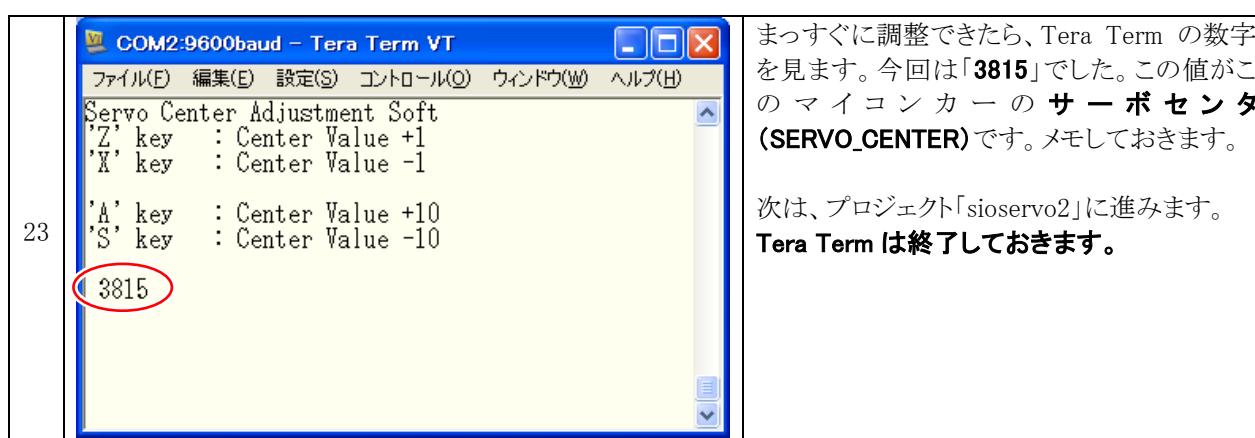
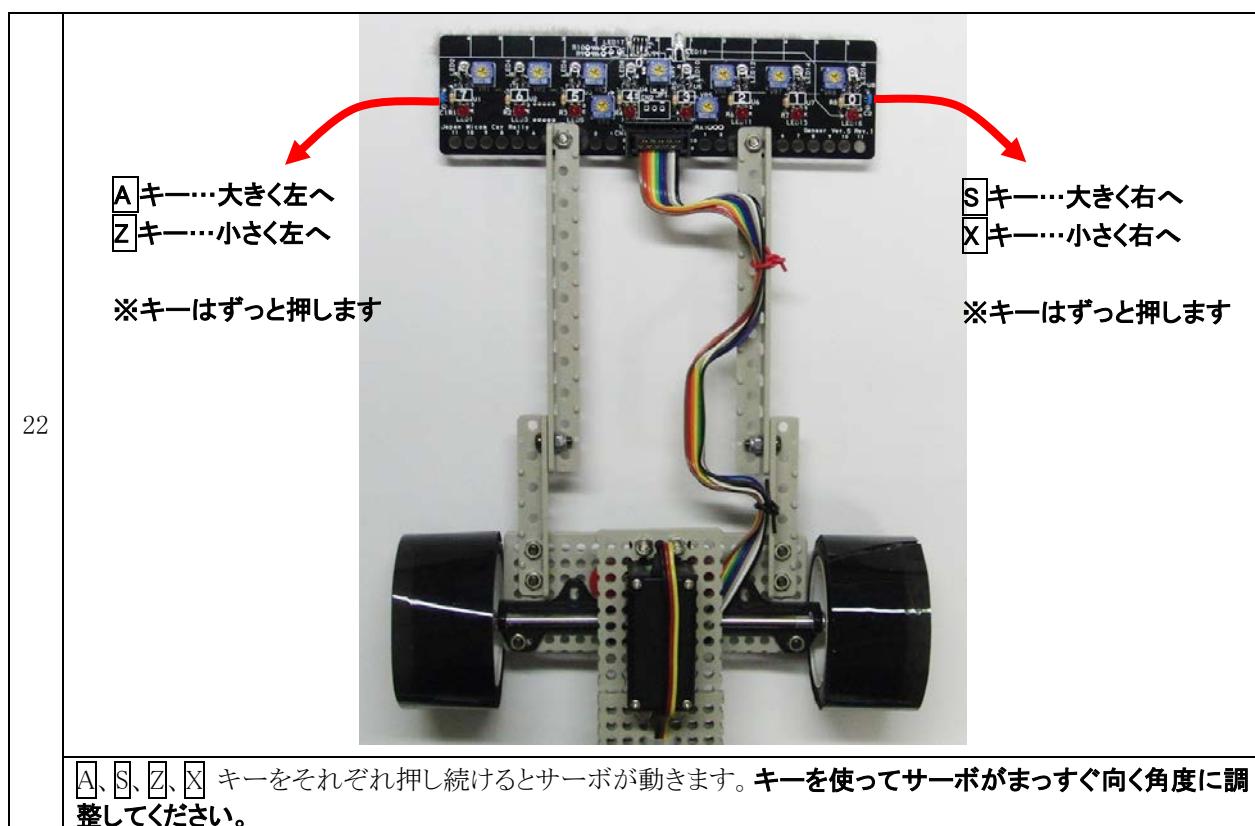
18	 <p>ポートを「Prolific USB-to-Serial Comm Port」がある番号、または接続しているシリアルにします。 OKをクリックします。</p>
----	--

19	 <p>Tera Term の画面が立ち上りました。</p>
----	---

7. サーボセンタと最大切れ角の調整

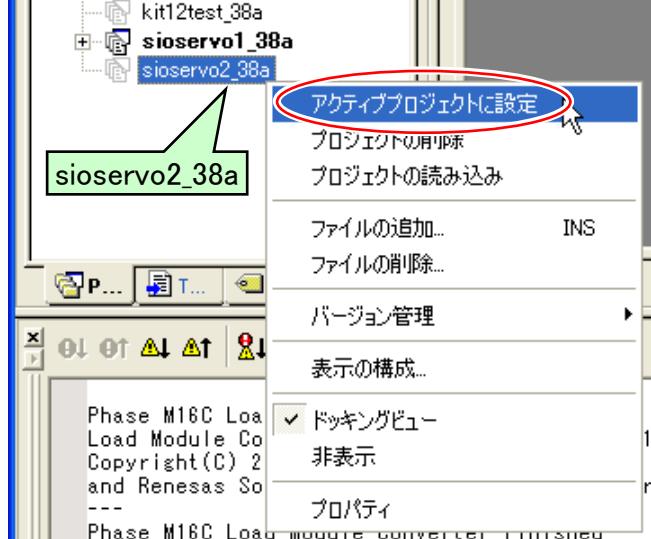
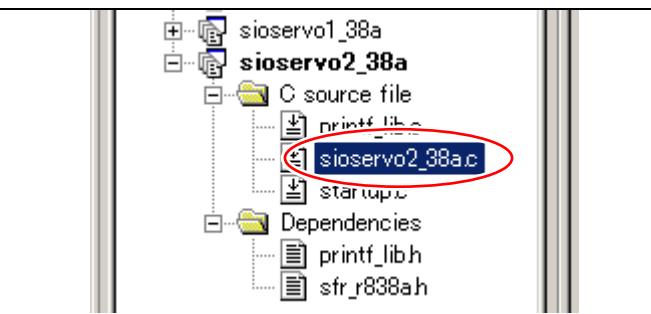
20		<p>マイコンカーの電源を入れてください。スイッチは二つとも ON にします。</p> <p>※マイコンカーの電源をすでに入れていた場合は、いちどマイコンカーの電源を OFF にしてから、再度 ON してください。</p> <p>※マイコンカーの電源を入れた瞬間に、マイコンカーからパソコンへメッセージを出力します。マイコンカーの電源を入れた後で、Tera Term を立ち上げても画面には何も出力されません。</p>
----	---	--

21		<p>マイコンカーの電源を入れると、左画面のようにメッセージが表示されます。</p> <p>いちばん下の「3750」が現在のサーボセンタの値です。</p>
----	--	---

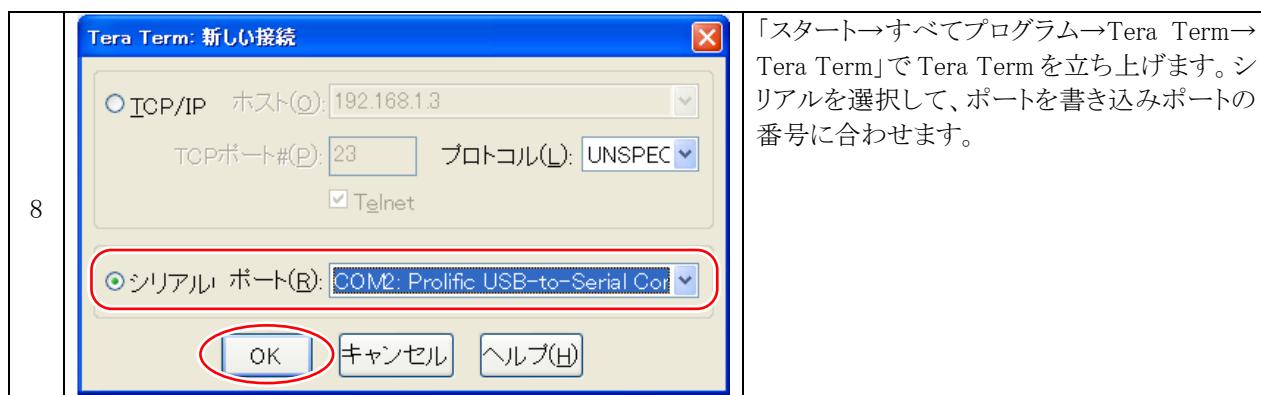
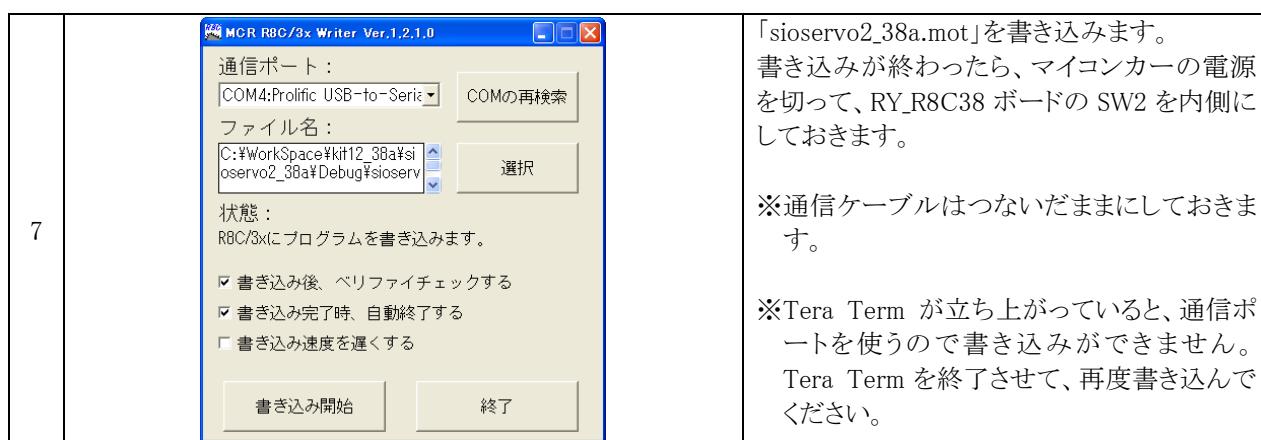
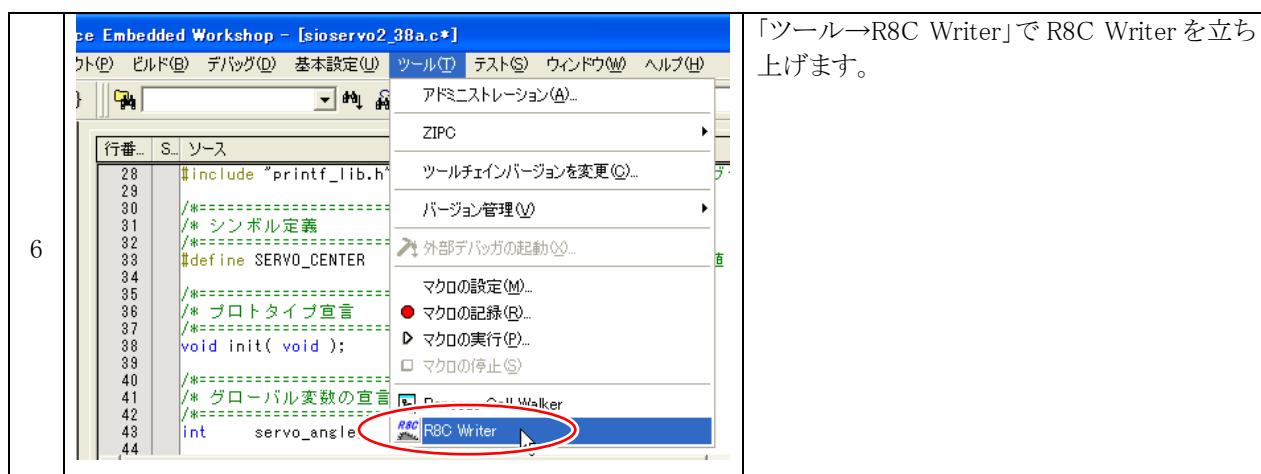
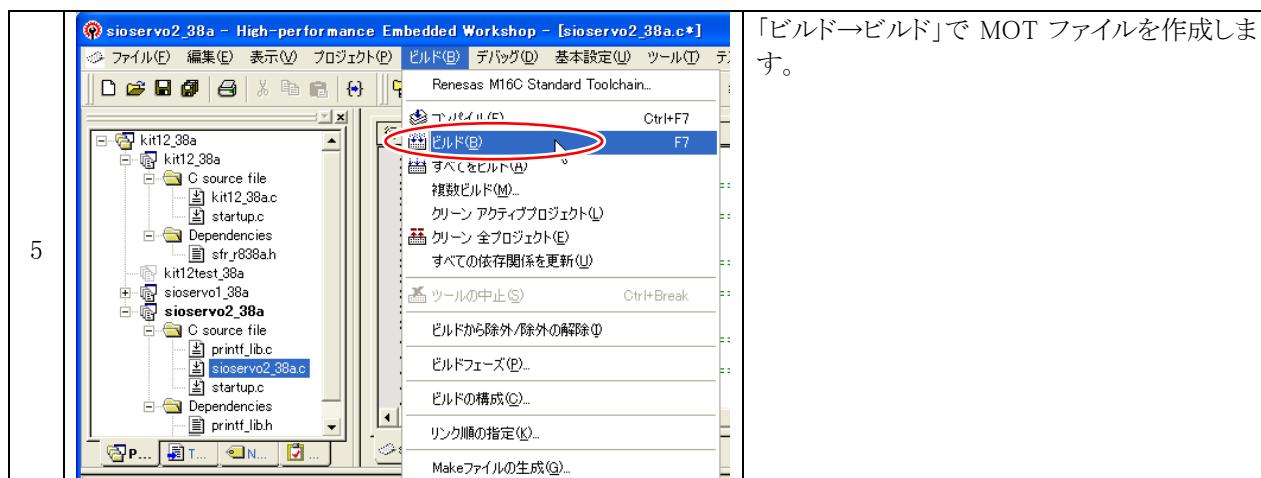


7.4 サーボの最大切れ角を見つける

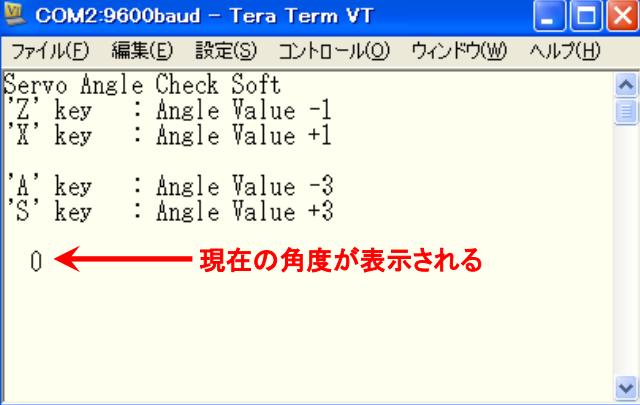
引き続き、サーボの最大切れ角を見つけます。

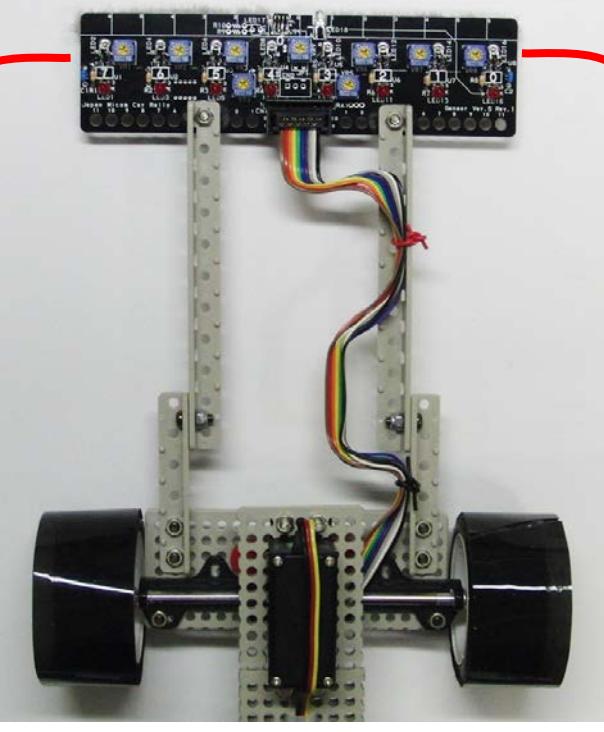
1		プロジェクト「sioservo2_38a」をアクティブプロジェクトに設定します。
2		「sioservo2_38a.c」をダブルクリックしてエディタウンドウを開きます。
3	<pre> 26 #include <stdio.h> 27 #include "sfr_r838a.h" 28 #include "printf_lib.h" /* 29 30 /*===== 31 /* シンボル定義 32 /*===== 33 #define SERVO_CENTER 3750 /* 34 35 /*===== 36 /* プロトタイプ宣言 37 /*===== 38 void init(void); </pre>	33 行に SERVO_CENTER 3750 という記述があります。
4	<pre> 26 #include <stdio.h> 27 #include "sfr_r838a.h" 28 #include "printf_lib.h" /* 29 30 /*===== 31 /* シンボル定義 32 /*===== 33 #define SERVO_CENTER 3815 /* 34 35 /*===== 36 /* プロトタイプ宣言 37 /*===== 38 void init(void); </pre>	先ほど見つけたサーボセンタ値に書き換えます。画面は、例として「3815」に書き換えていきます。

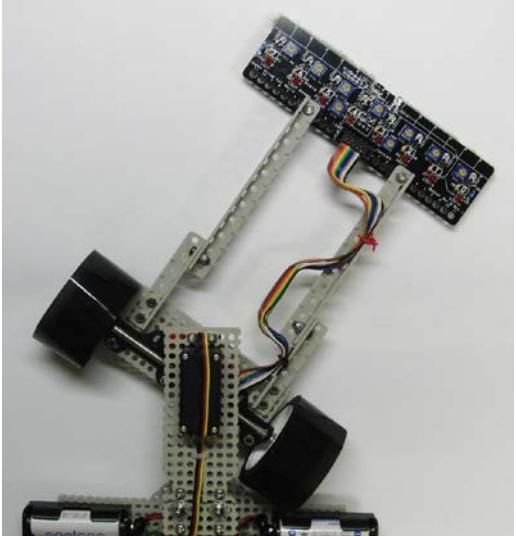
7. サーボセンタと最大切れ角の調整



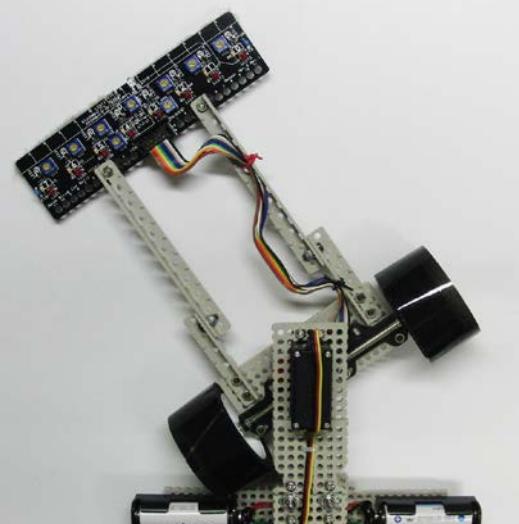
7. サーボセンタと最大切れ角の調整

	 <pre>COM2:9600baud - Tera Term VT ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H) Servo Angle Check Soft 'Z' key : Angle Value -1 'X' key : Angle Value +1 'A' key : Angle Value -3 'S' key : Angle Value +3 0 ← 現在の角度が表示される</pre>	<p>マイコンカーの電源を入れると左画面のメッセージが表示されます。</p> <p>表示されない場合は、ケーブルの接続やマイコンカーの電池、書き込みスイッチを戻したか、通信ポートの番号、書き込んだプログラムが本当にプロジェクト「sioservo2_38a」の sioservo2_38a.mot を書き込んだかなど確かめてください。</p> <p>※マイコンカーのスイッチは、二つとも ON にしてください。</p>
--	--	--

	 <p>A キー…左へ3度 Z キー…左へ1度</p> <p>S キー…右へ3度 X キー…右へ1度</p> <p>□ A、□ S、□ Z、□ X キーをそれぞれ押すとサーボが動きます。右はどこまでハンドルを曲げができるかを調べます。左も同様に調べます。</p>
--	--

		<p>まず S キー、X キーで右の限界を見つけます。タイヤを回して回るか確かめてください。シャーシにぶつかるようなら □ キーでもう少し小さくしてください。</p>
--	---	---

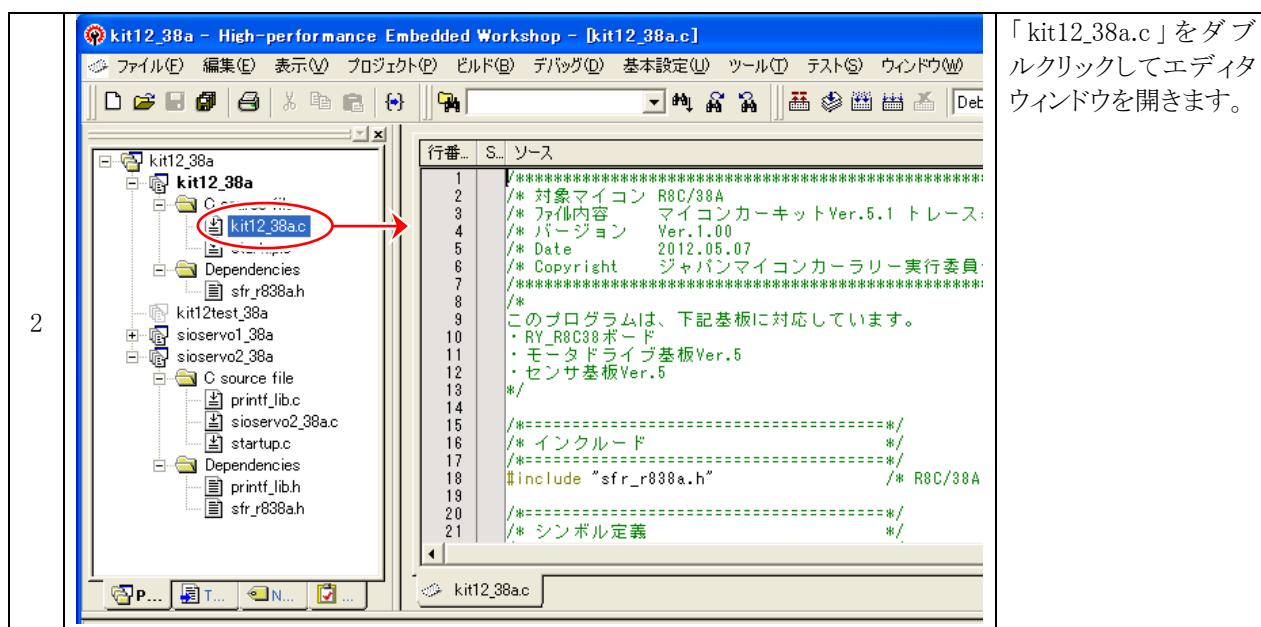
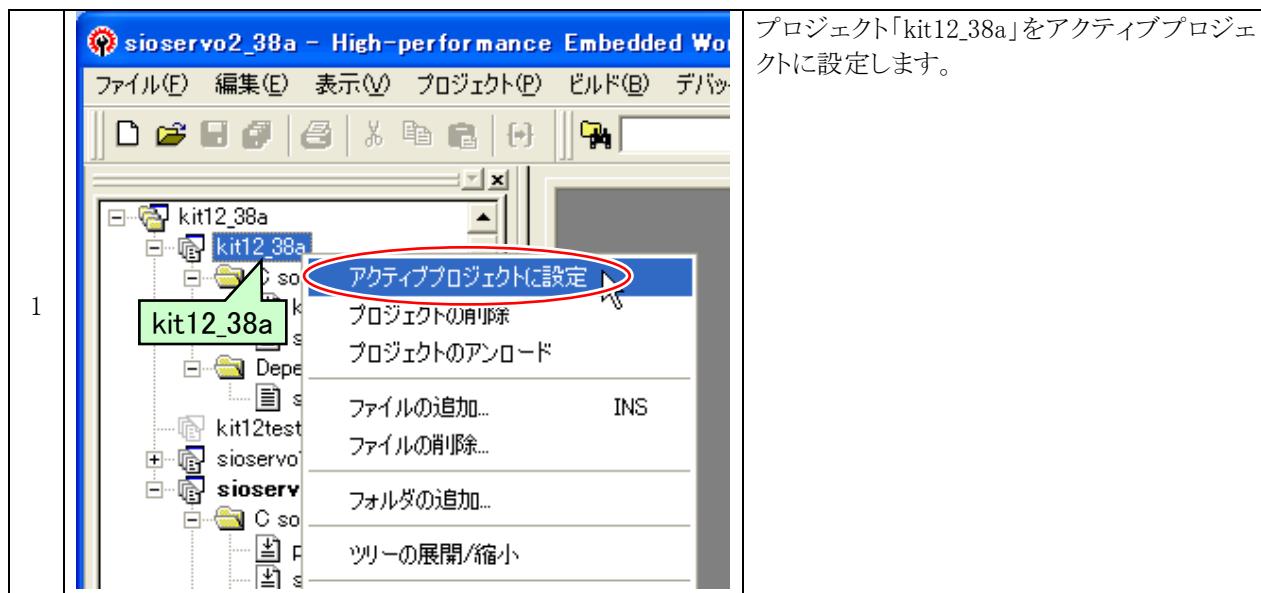
12	 <pre> COM2:9600baud - Tera Term VT [メニュー] [編集] [設定] [コントロール] [ウインドウ] [ヘルプ] Servo Angle Check Soft 'Z' key : Angle Value -1 'X' key : Angle Value +1 'A' key : Angle Value -3 'S' key : Angle Value +3 40 </pre>	<p>Tera Term の数値を見ます。これが現在ハンドルを右へ曲げている角度です。40 度曲げしていると言うことが分かりました。 右が 40 度だからといって左が-40 度とは限りません。必ず左右確かめます。</p>
----	---	---

13		<p>今度は A キー、Z キーで逆の左の限界を見つけます。こちらもタイヤを回して回るか確かめてください。シャーシにぶつかるようなら X キーでもう少し小さくしてください。</p>
----	--	---

14	 <pre> COM2:9600baud - Tera Term VT [メニュー] [編集] [設定] [コントロール] [ウインドウ] [ヘルプ] Servo Angle Check Soft 'Z' key : Angle Value -1 'X' key : Angle Value +1 'A' key : Angle Value -3 'S' key : Angle Value +3 -41 </pre>	<p>Tera Term の数値を見ます。これが現在ハンドルを左へ曲げている角度です。-41 度曲げしていると言うことが分かりました。</p>
----	--	--

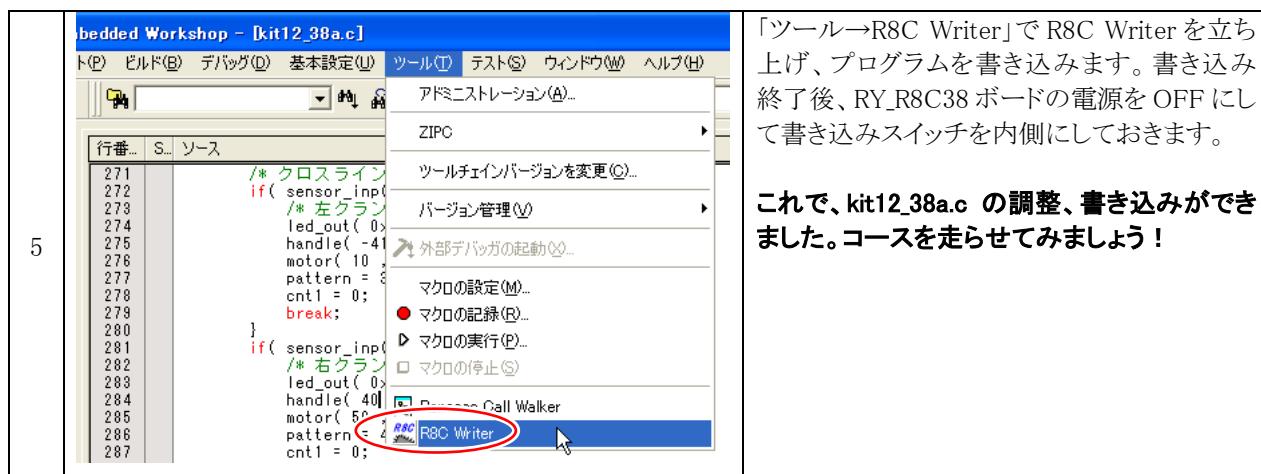
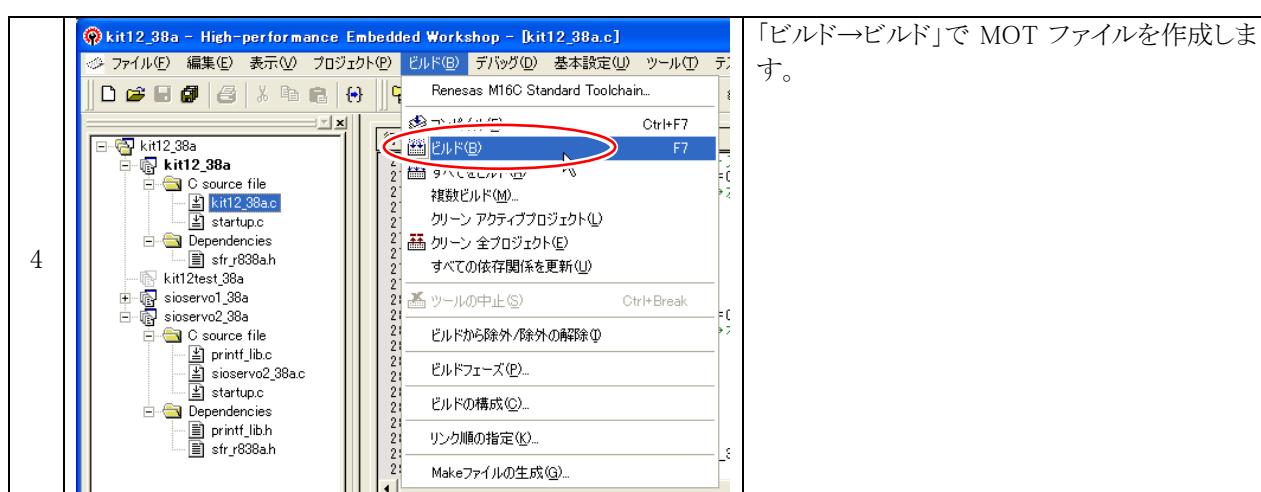
7.5 「kit12_38a.c」プログラムを書き換える

プロジェクト「sioservo1_38a」、「sioservo2_38a」で 3 つの数値が分かりました。それらの数値をマイコンカーを走行させるプログラム「kit12_38a.c」へ書き込みます。このファイルは、プロジェクト「kit12_38a」内にあります。



下記のように 3 つの値を自分のマイコンカーの値に変更します。

内容	kit12_38a.c で書き換える行番号	キットの標準値	今回の例の値
サーボセンタ	26 行	3750	3815
左の最大角度	275 行	-38	-41
右の最大角度	284 行	38	40



「ビルド→ビルド」で MOT ファイルを作成します。

「ツール→R8C Writer」で R8C Writer を立ち上げ、プログラムを書き込みます。書き込み終了後、RY_R8C38 ボードの電源を OFF にして書き込みスイッチを内側にしておきます。

これで、kit12_38a.c の調整、書き込みができます。コースを走らせてみましょう！

8. プログラムの改造ポイント

8.1 概要

kit12_38a.c プログラムをビルド後、RY_R8C38 ボードにプログラムを書き込み、マイコンカーを走らせます。サンプルプログラムを何も改造していないにも関わらず(SERVO_CENTER などのマイコンカー固有の値は除きます)、ほとんどの場合は途中で脱輪してしまうと思います。

今までの説明は、下記の状態を想定して説明しています。

- コースが白から黒色に変化したときのセンサ反応がすべて同じ反応になっている
- クロスラインやハーフラインに対してまっすぐに進入している
- 直線はほぼ中心をトレースしている

しかし、ほとんどの場合、下記のようになってしまいます。

- センサの反応にはらつきがある
- クロスラインやハーフラインに対して、斜めに入ってしまう
- 直線でも、右か左に寄ってしまう

そのため、脱輪してしまうのです。

これからの説明は、「液晶・microSD 基板 kit07_38a プログラム解説マニュアル データ解析(microSD)編」に掲載されている方法で、実際のセンサ状態がどうなっているか確認してみました。具体的には 10ms ごとにセンサの状態を保存、パソコンに転送してパターンとセンサの状態を確認しました。その結果を、次ページから説明します。

なお、説明は、下記の 3 つ 1 組で説明しています。

(a) 現象

どう落ちたのか。

(b) 解析結果

なぜ落ちたのか。

(c) 解決例

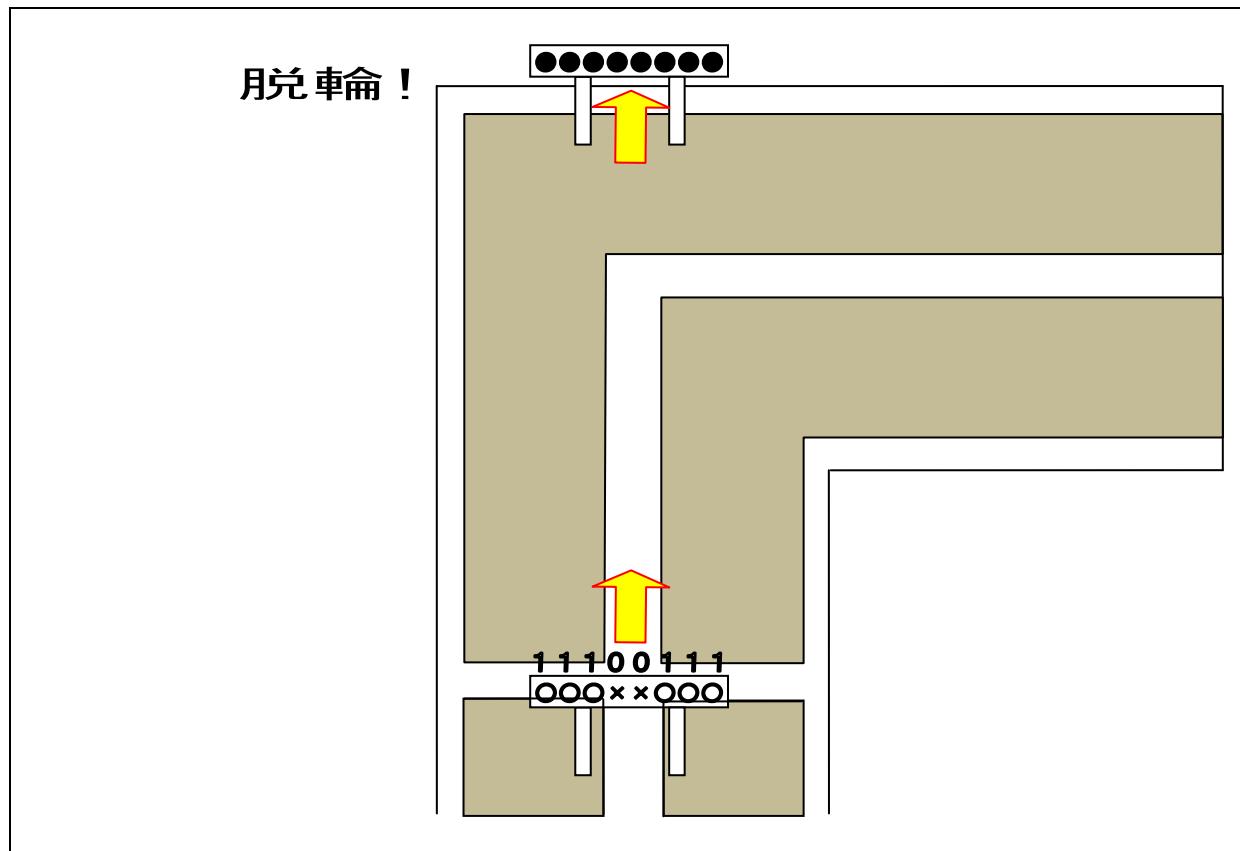
どのすれば解決できるのか

8.2 脱輪事例

8.2.1 クロスラインの検出がうまくいかない

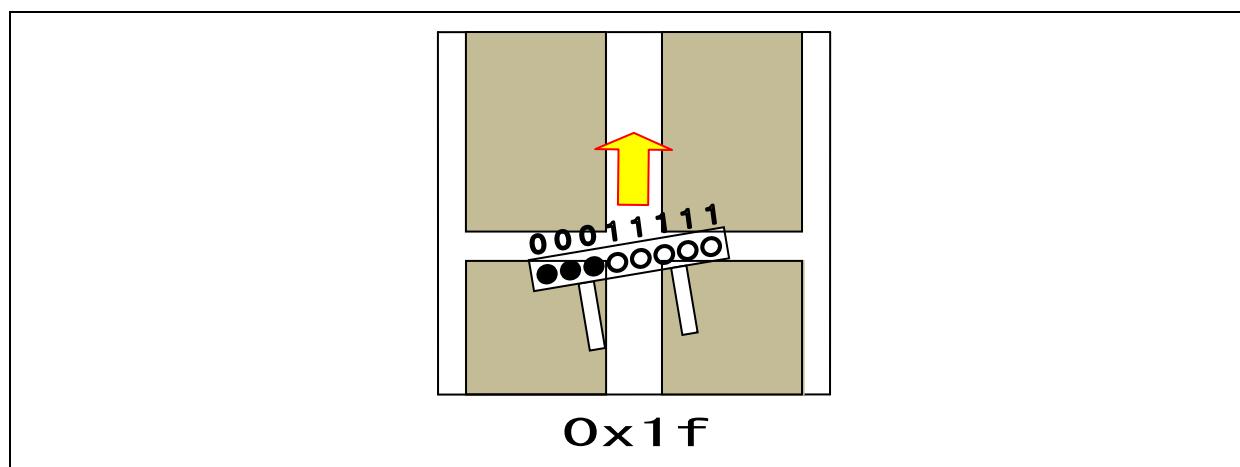
(a) 現象

クロスラインを検出後、クラシクで曲がらずにそのまま進んで脱輪するようになりました。



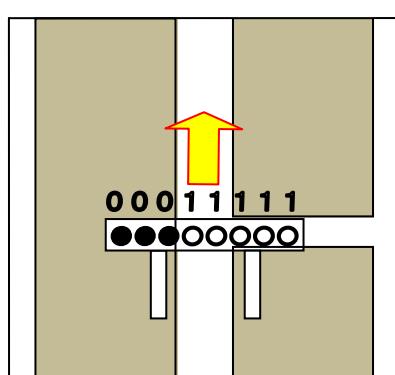
(b) 解析結果

走行データを探って解析すると、クロスラインを検出する瞬間、センサの状態が想定の「0xe7」ではなく、「0x1f」になっていることが分かりました(下図)。



8. プログラムの改造ポイント

「0x1f」というセンサの反応は、どこかで見た状態です。右ハーフラインのセンサ検出は、8 個のセンサをチェックして「0x1f」の時に右ハーフラインと判断します。



想定ではこの状態が 0x1f の状態

このように、本当はクロスラインなのに、右ハーフラインのセンサ検出状態と一致してしまい、誤動作していました。

(c) 解決例

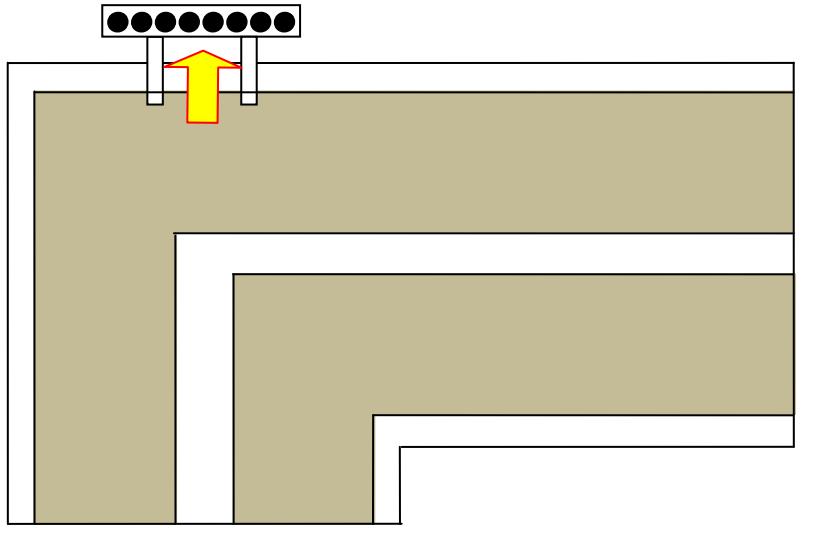
センサが斜めになることはプログラムではどうしようもありません。サーボセンタ値を合わせたとしてもラインを検出する瞬間は、どうしても片側のみになってしまいます。解決例としては、右ハーフラインと判断しても少しの間はセンサのチェックを続けて、クロスラインの状態になったなら **クロスラインと判断し直すと良いでしょう**。左ハーフラインも同様です。

8.2.2 クランクの検出がうまくいかない

(a) 現象

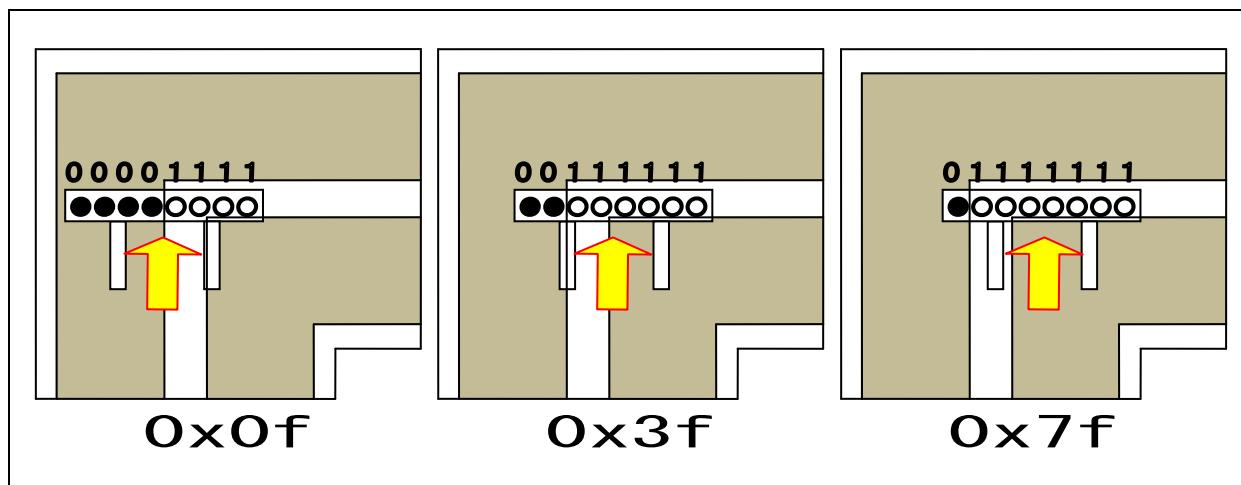
クランクで曲がらずにそのまま進んで脱輪するようになりました。モータドライブ基板の LED は 2 個点いているのでパターン 23 には入っているようです。

月兌車両！



(b) 解析結果

走行データを採って解析すると、右クランクを検出する瞬間、センサの状態が想定の「0x1f」ではなく、「0x0f」、「0x3f」、「0x7f」になつていていました（下図）。



このように、本当は右クランクなのに、プログラムでは右クランクと一致するセンサ状態ではないため、そのまま進んで脱輪してしまいました。

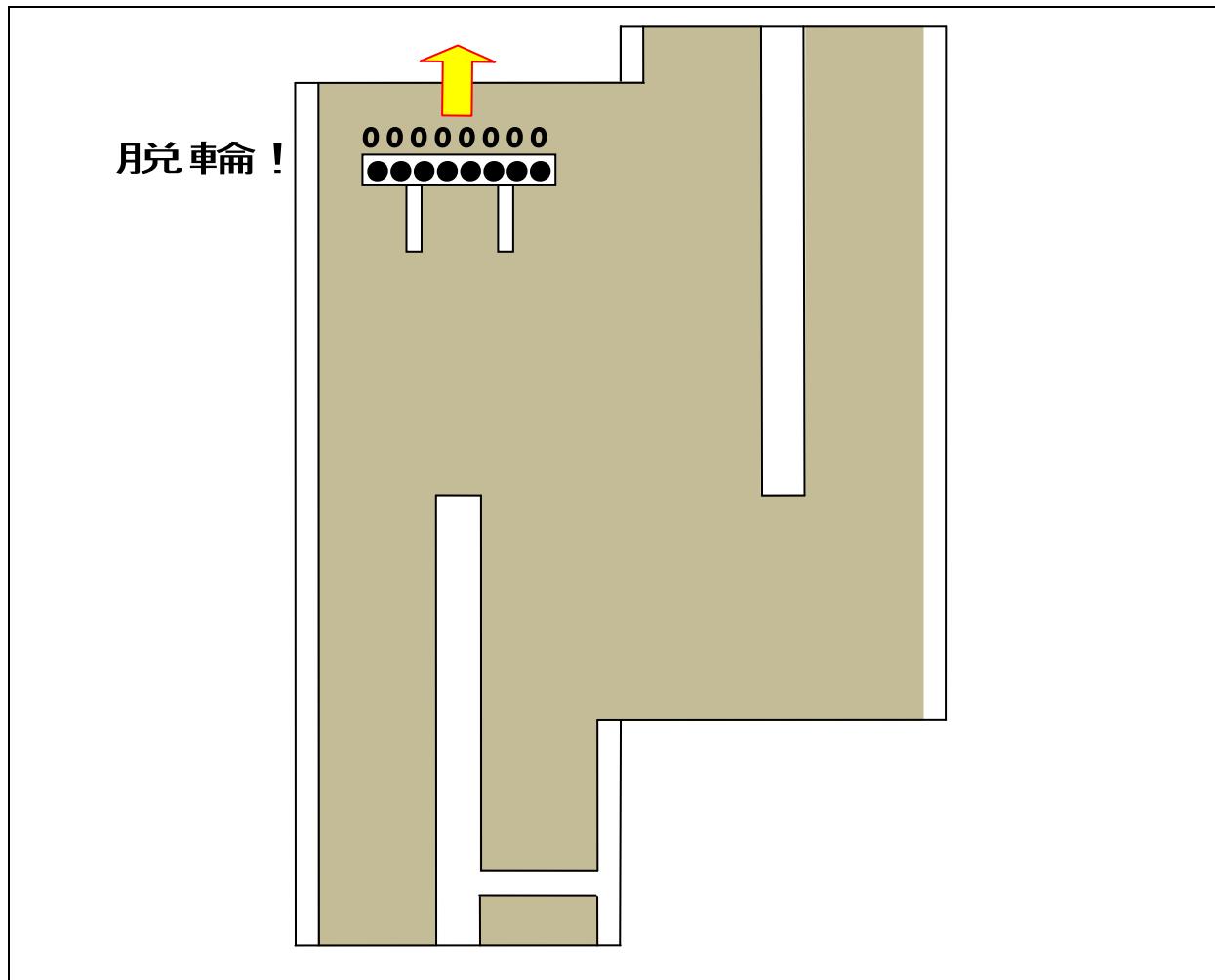
(c) 解決例

キットのプログラムでは、右クランクを検出するセンサ状態は、「0x1f」のみです。実際は、「0x0f」や「0x3f」や「0x7f」のセンサ状態もありました。そのため、これらの状態でも右クランクと判断するように、センサ状態を追加しましょう。同様に、左クランクも誤動作することが考えられますので、センサ状態を追加しておきましょう。

8.2.3 ハーフラインの検出がうまくいかない

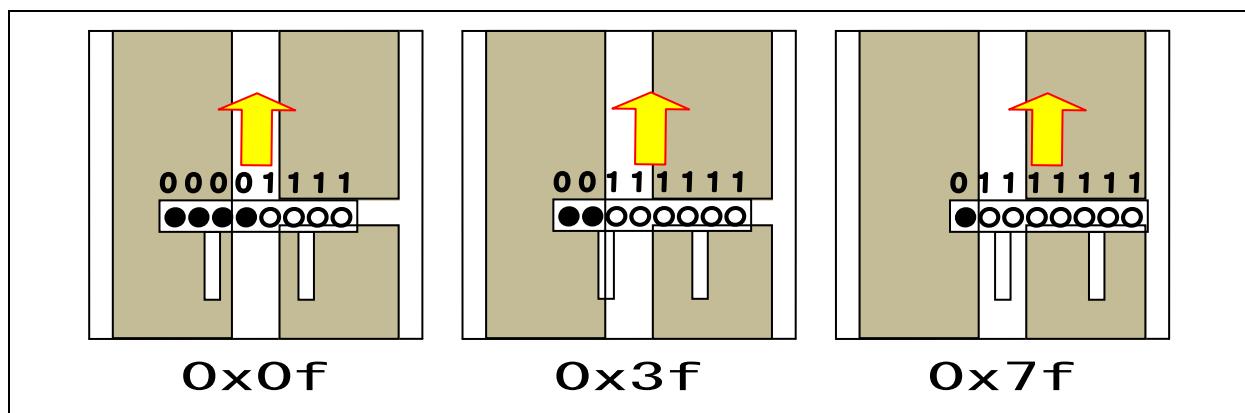
(a) 現象

右レーンチェンジを、そのまま直進して脱輪してしまいました。



(b) 解析結果

走行データを採って解析すると、右ハーフラインを検出する瞬間、センサの状態が想定の「0x1f」ではなく、「0x0f」、「0x3f」、「0x7f」になっていることが分かりました（下図）。



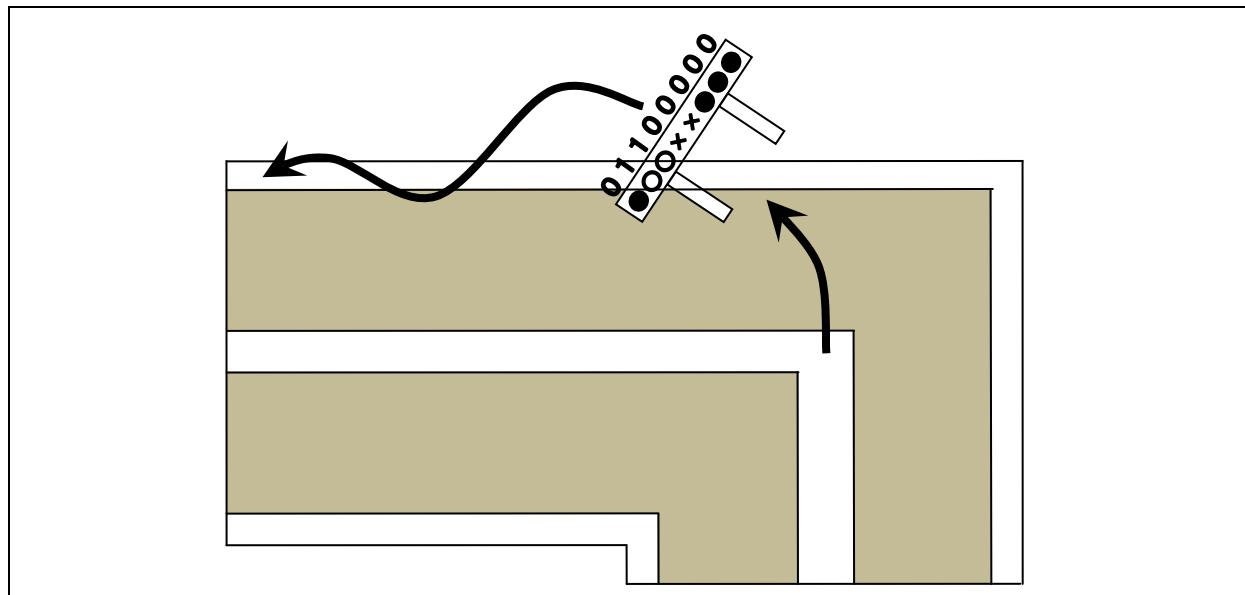
(c) 解決例

キットのプログラムでは、右ハーフラインを検出するセンサ状態は、「0x1f」のみです。実際は、「0x0f」や「0x3f」や「0x7f」のセンサ状態もありました。そのため、これらの状態でも右ハーフラインと判断するように、センサ状態を追加しましょう。同様に左ハーフラインも誤動作することが考えられますので、対策しておきましょう。

8.2.4 クランククリア時、外側の白線を中心と勘違いして脱輪してしまう

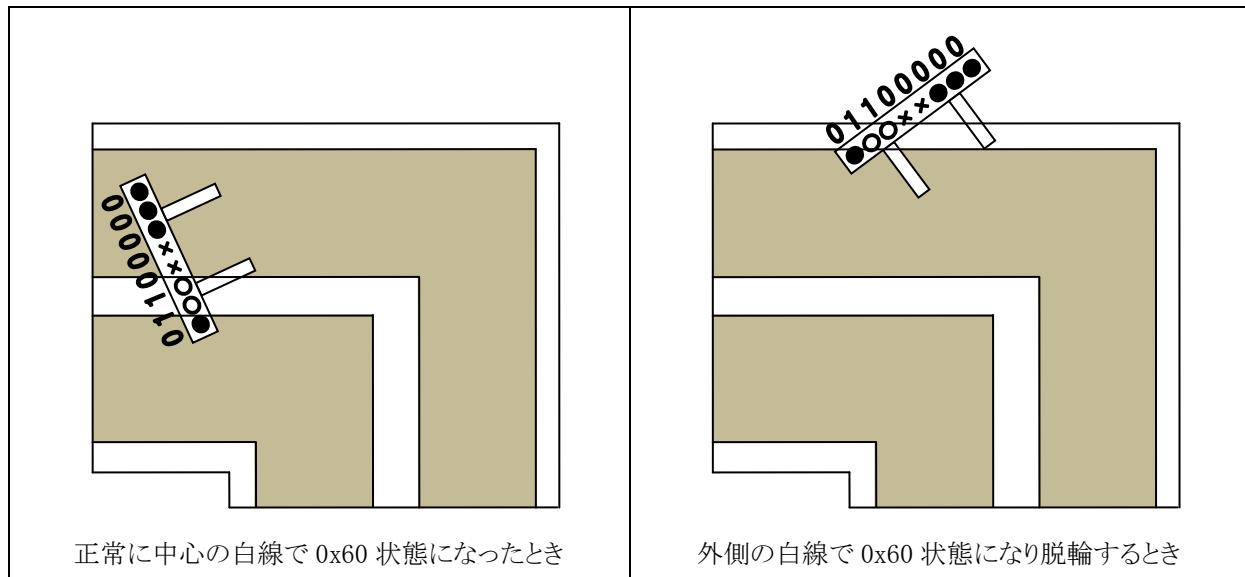
(a) 現象

左クランクを検出し、左へハンドルを切りました。若干脹らみ気味に曲がりながら進んでいくと、外側の白線部分をトレースしながら進み脱輪してしまいました。



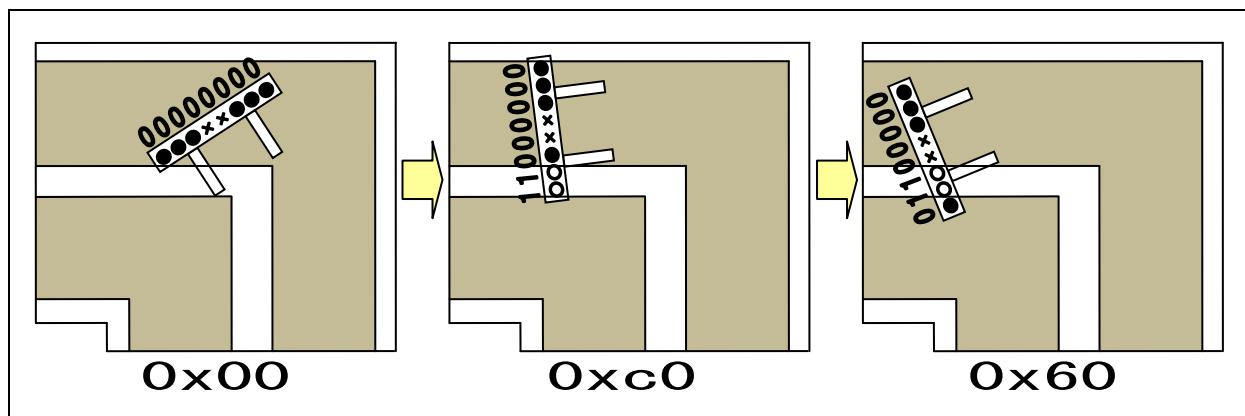
(b) 解析結果

左クランクを検出すると、左に 38 度、左モータ 10%、右モータ 50%にします。その状態をいつ終えて通常パターンに戻るのでしょうか。サンプルプログラムは、センサの状態が「0x60」になったときです。状態は、下左図のように中心の白線を検出して終わることを想定しています。しかし、スピードが速すぎると曲がりきれずに外側に膨らんでしまい、下右図のように外側の白線で「0x60」状態を検出してしまいます。この状態で通常パターンに戻るのでは脱輪してしまうのです。

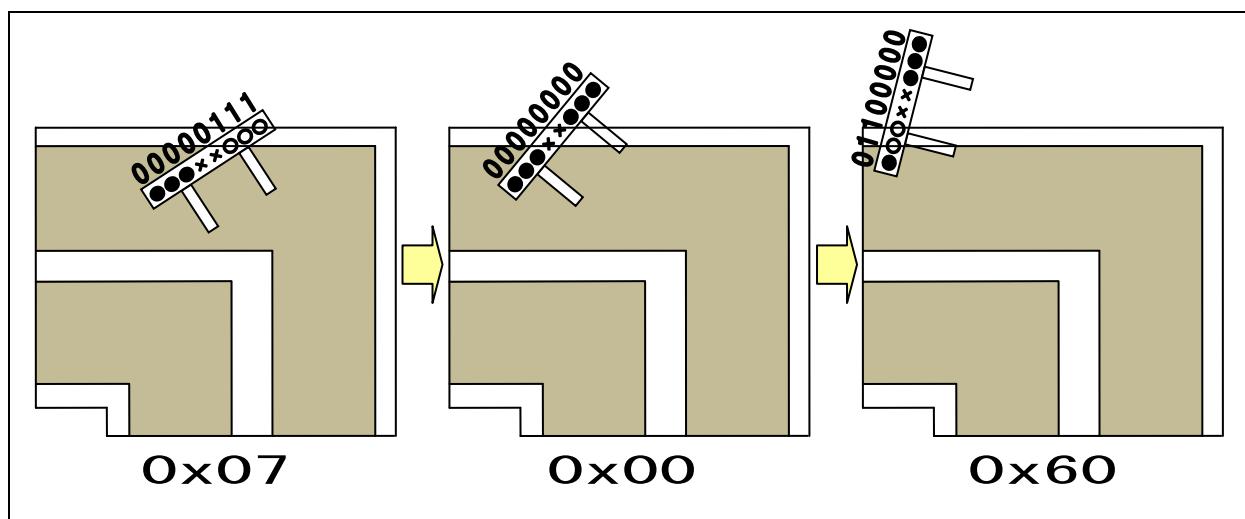


(c) 解決例

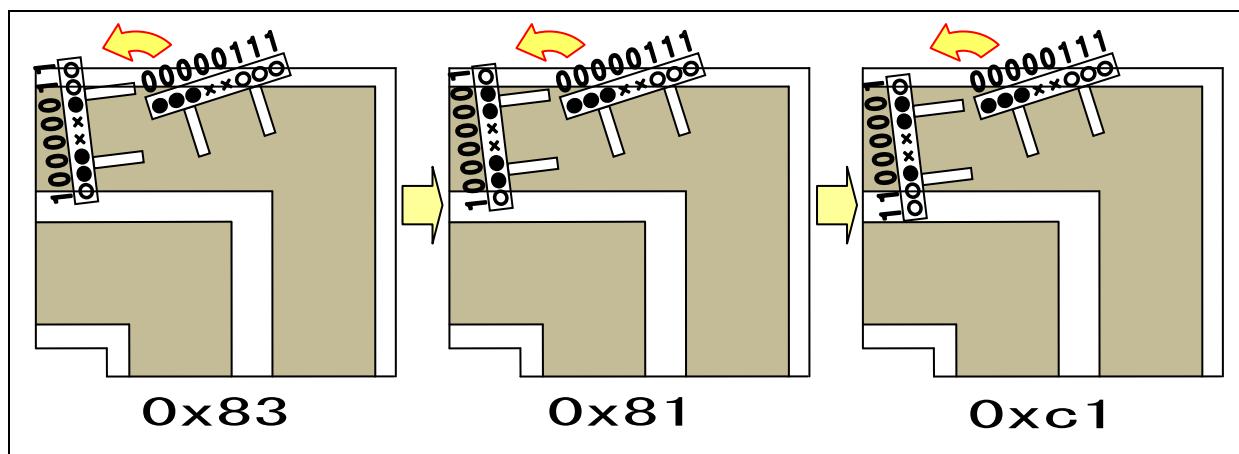
正常に中心線を見つけたときのセンサの移り変わりを確認します。 $0x00 \rightarrow 0xc0 \rightarrow 0x60$ のようにセンサの状態が変わり、通常走行に戻ります(下図)。



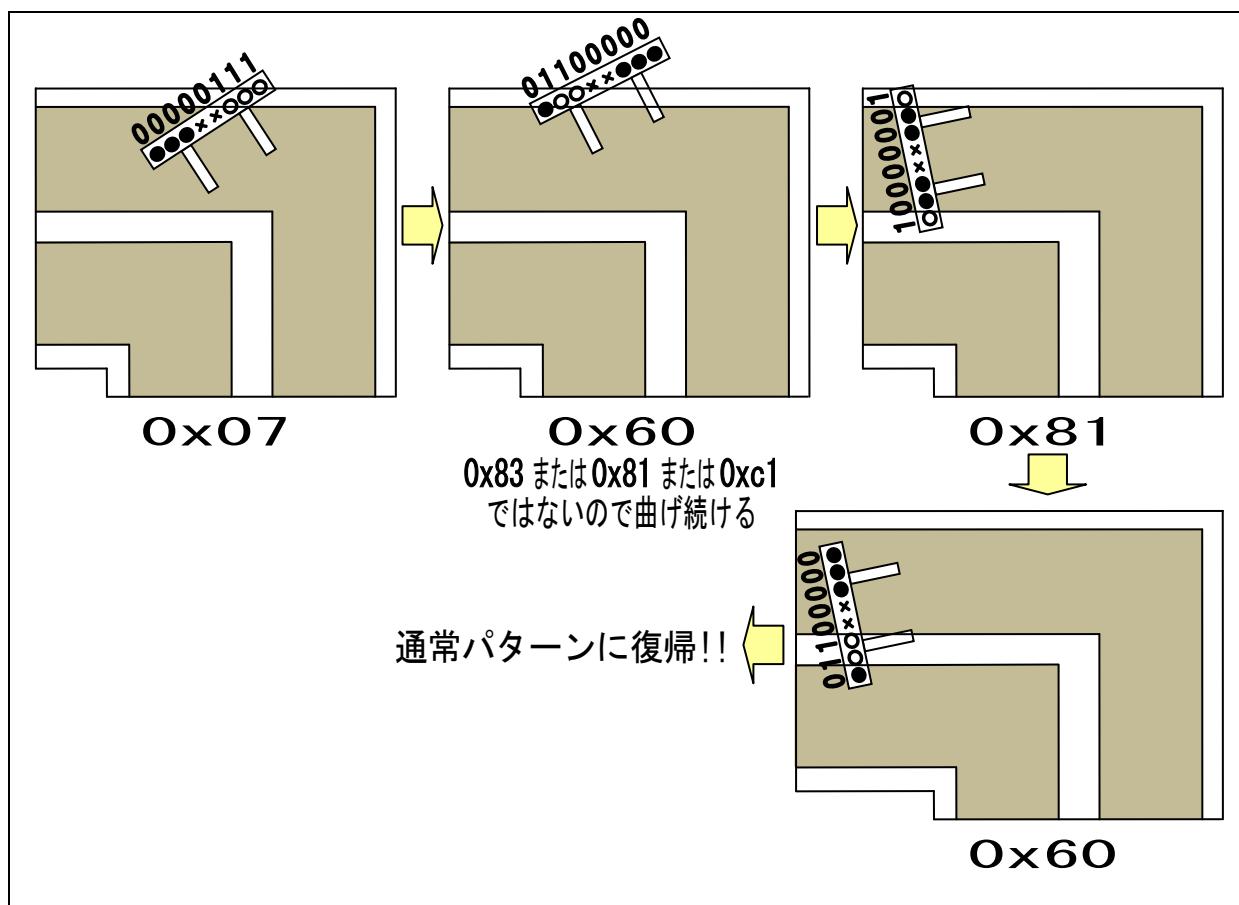
外側の白線を中心と勘違いしたときのセンサの移り変わりを確認します。 $0x07 \rightarrow 0x00 \rightarrow 0x60$ のようにセンサの状態が変わり、通常走行に戻ります(下図)。



2通りを見比べると誤動作するときのセンサ状態は、「**0x07 になってから 0x60 になる**」ということが分かりました。そこで、センサの状態が「0x07」になつたら、「0x83 か 0x81 か 0xc1」になるまで曲げ続けるようにしたらどうでしょうか(下図)。



シミュレーションしてみます。0x07になると、「0x83か0x81か0xc1」になるまで曲げ続けます。そのため、「0x60」になっても曲げ続けます。前までは、ここで通常パターンに戻り脱輪してしまいました。その後、さらに曲げ続け「0x81」になると、0x60をチェックするプログラムへ戻します。その後センサが0x60の状態になると、中心線と判断して通常パターンに戻ります。

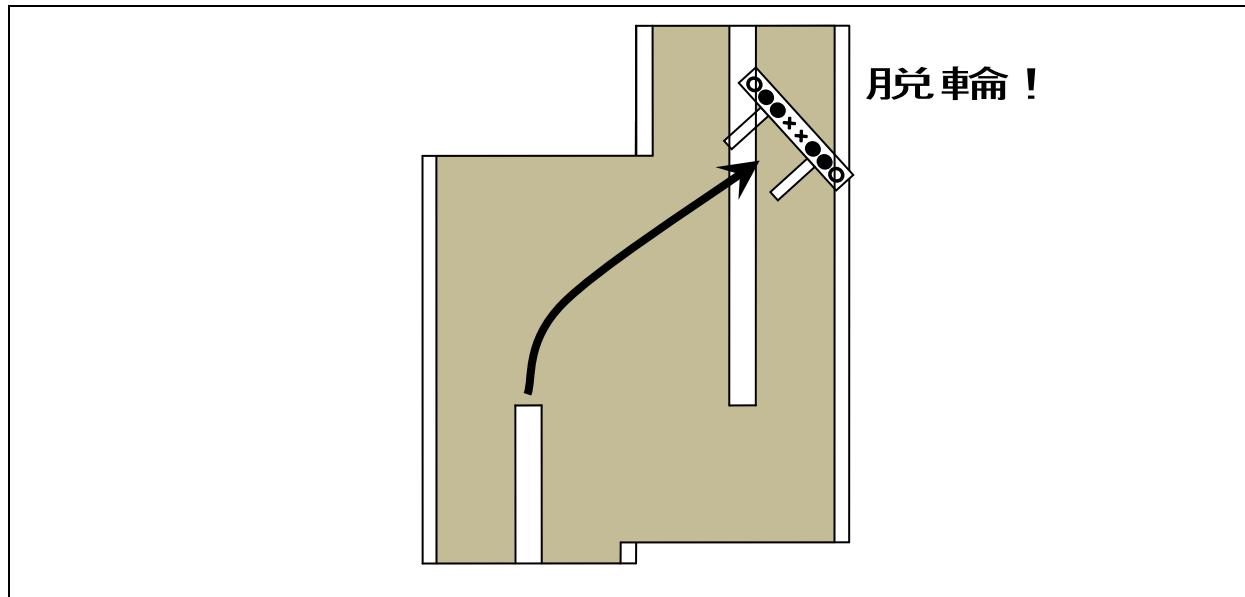


この考え方に基づきプログラムを作ってみましょう。これで外側の白線を中心線と間違ってしまう誤動作が無くなります。右クランクも同様です。

8.2.5 レーンチェンジ終了の判断ができない

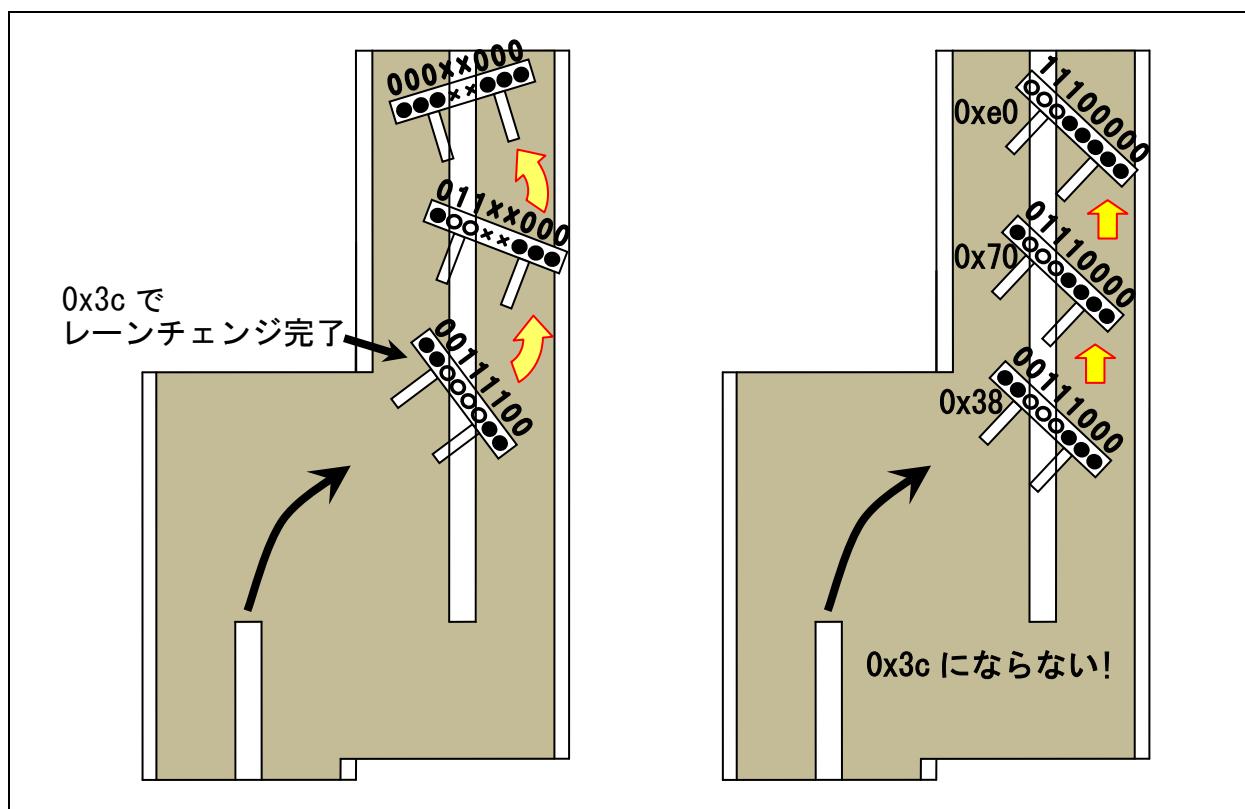
(a) 現象

右レーンチェンジを検出して、右にハンドルを切りました。新しい中心線を検出してそのラインでトレースしていくはずでしたが、そのまままっすぐに進んでしまい脱輪してしまいました(下図)。



(b) 解析結果

センサ状態を解析すると、新しい中心線を見つけたとき、センサの状態が「`0x38→0x70→0xe0`」と変化していました。サンプルプログラムで右レーンチェンジが終わったと判断するのは、センサを 8 個チェックして「`0x3c`」になったときです(左下図)。進入角度によっては、「`0x3c`」にならないことがあります(右下図)。



(c) 解決例

新しい中心線を発見したときのセンサ状態を「0x3c」だけから、解析結果で検出されたセンサ状態を追加します。他には、センサの検出状態をまったく別な値に変える、中心線を検出したらサーボのハンドル角度を浅くしてさらに進んでいくなど、色々考えられます。どうすればレーンチェンジコースを安定して（さらには速く）走行できるか、いろいろ試してみてください。

8.3 まとめ

すべてに言えることですが、例えば、クロスラインのセンサ状態を□□にしようとなります。

- センサ状態□□はクロスライン
- しかし、センサ状態□□はハーフラインでもありえる
- そのため、誤検出して脱輪することがある

というように、プログラム（自分）では想定していない場所でセンサ状態が一致してしまい、脱輪してしまうのです。
そこで、

- センサ状態△△はクロスライン
- 他の状態でセンサ状態△△になることはない
- したがって誤動作しない！！

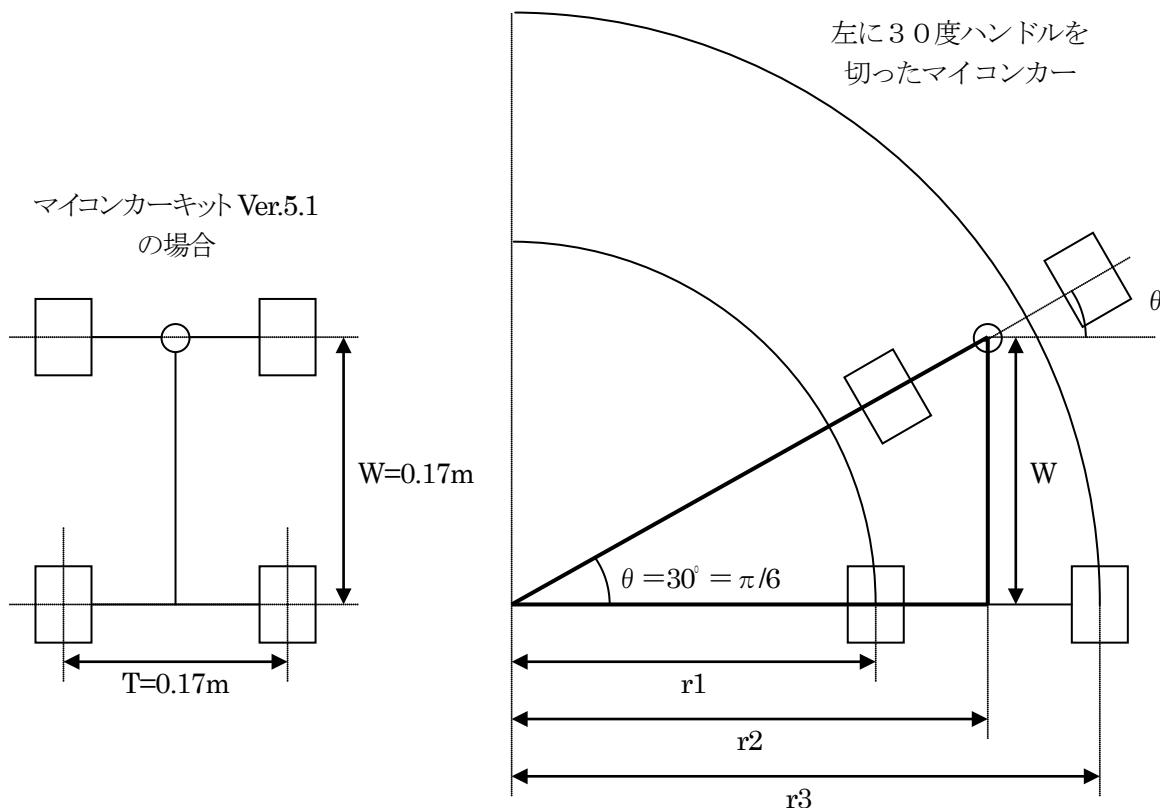
というように、自分オリジナルのセンサ状態を見つけることがポイントです。

今回いくつかの事例を図解しましたが、まだ脱輪してしまう条件があるかもしれません。脱輪したとき、「ああ落ちてしまった…」で終わらせるのではなく、徹底的に原因を究明して対策してください（ハード、ソフトに限らず）。一つ一つの問題を地道に解決させることが、大会で完走させる秘訣です。

9. モータの左右回転差の計算方法

9.1 計算方法

ハンドルを切ったとき、内輪側と外輪側ではタイヤの回転数が違います。その計算方法を下記に示します。



T =トレッド…左右輪の中心線の距離 キットでは $0.17[\text{m}]$ です。

W =ホイールベース…前輪と後輪の間隔 キットでは $0.17[\text{m}]$ です。

図のように、底辺 r_2 、高さ W 、角度 θ の三角形の関係は次のようにです。

$$\tan \theta = W / r_2$$

角度 θ 、 W が分かっていますので、 r_2 が分かれます。

$$r_2 = W / \tan \theta = 0.17 / \tan(\pi/6) = 0.294[\text{m}]$$

内輪の半径 r_1 は、

$$r_1 = r_2 - T/2 = 0.294 - 0.085 = 0.209$$

外輪の半径 r_3 は、

$$r_3 = r_2 + T/2 = 0.294 + 0.085 = 0.379$$

よって、外輪を 100 とすると内輪の回転数は、

$$r1/r3 \times 100 = 0.209 / 0.379 \times 100 = 55$$

となります。

左に 30° ハンドルを切ったとき、右タイヤ 100 に対して、左タイヤ 55 の回転となる。

プログラムでは次のようにすると、内輪と外輪のロスのない回転ができます。

```
handle( -30 );
speed( 55, 100 );
```

9.2 エクセルを使った内輪の自動計算

マイコンカーラリーホームページに、内輪を自動で計算する「角度計算.xls」ファイルがあります。これを開くと、下記のようなファイルが開きます。

※アドレス:<http://www.mcr.gr.jp/tech/download/main01.html> → 「マイコンカー、オプションに関する資料(R8C 編)」→「内輪差、外輪差計算エクセルシート」です。

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	W		0.17	m ←ホイールベースを入力してください					ハンドル角度を入力してください			15	
2	T		0.17	m ←トレッドを入力してください					外輪のスピードを入力してください			50	
3													
4	度	rad	r2	r1	r3	r1/r3*100		内輪のスピード(自動計算)				38	
5	0	0				100							
6	1	0.017	9.744	9.659	9.829	98							
7	2	0.035	4.871	4.786	4.956	97							
8	3	0.052	3.245	3.160	3.330	95							
9	4	0.070	2.432	2.347	2.517	93							
10	5	0.087	1.944	1.859	2.029	92							
11	6	0.105	1.618	1.533	1.703	90							
12	7	0.122	1.385	1.300	1.470	88							
13	8	0.140	1.210	1.125	1.295	87							
14	9	0.157	1.074	0.989	1.159	85							
15	10	0.174	0.965	0.880	1.050	84							
..							

プログラム例
handle(15)
motor(50,38)

下記のセルを入力すると、○部分に自動でハンドル角度とスピード値が入力されます。

- L1セル→ハンドル角度を入力
- L2セル→外輪のスピードを入力

kit12_38a.c の左右回転差計算もこのエクセルシートで行いました。ホイールベースとトレッドの寸法は、合わせておいてください。

9.3 内輪を自動で計算する関数を追加する

ハンドル角度を変えるたびに、内輪の値を再計算するのは大変です。

そこで内輪の値を自動で計算する関数 diff 関数を追加して、自動で計算するようにしてみます。diff とは、「difference」の略で「差」という意味です。

9.3.1 プロトタイプの追加

```
/*=====
/* プロトタイプ宣言 */
=====*/
void init( void );
void timer( unsigned long timer_set );
int check_crossline( void );
int check_rightline( void );
int check_leftline( void );
unsigned char sensor_inp( unsigned char mask );
unsigned char dipsw_get( void );
unsigned char pushsw_get( void );
unsigned char startbar_get( void );
void led_out( unsigned char led );
void motor( int accele_l, int accele_r );
void handle( int angle );
int diff( int pwm );           ←追加
```

diff 関数を追加するので、プロトタイプ宣言します。

9.3.2 大域変数の追加

```
/*=====
/* グローバル変数の宣言 */
=====*/
unsigned long cnt0;          /* timer 関数用 */
unsigned long cnt1;          /* main 内で使用 */
int pattern;                /* パターン番号 */
int angle_buff;           /* 現在ハンドル角度保持用 */
```

angle_buff 変数を追加します。これは、後ほど説明する左右回転差を計算する関数に、現在のハンドル角度を知らせるための変数です。

9.3.3 handle 関数内の追加

```

/*****************/
/* サーボハンドル操作 */
/* 引数 サーボ操作角度 : -90~90 */
/*      -90で左へ90度、0でまっすぐ、90で右へ90度回転 */
/*****************/
void handle( int angle )
{
    angle_buff = angle;           /* 現在の角度保存 */

    /* サーボが左右逆に動く場合は、「-」を「+」に替えてください */
    trdgrd1 = SERVO_CENTER - angle * HANDLE_STEP;
}

```

角度を angle_buff 変数に保存します。diff 関数で使用するための準備です。

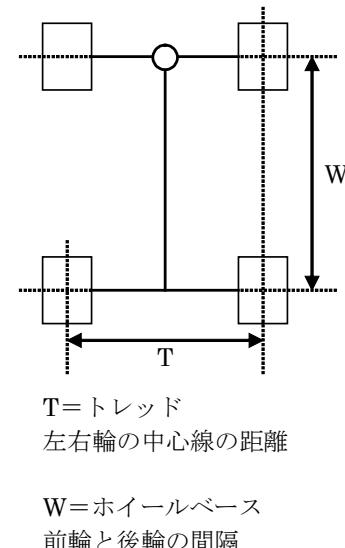
9.3.4 配列の追加

配列データの作成は大変なので、エクセルを使って追加します。

マイコンカーラリーホームページ:<http://www.mcr.gr.jp/tech/download/main01.html> → 「マイコンカー、オプションに関する資料(R8C 編)」→「内輪差、外輪差計算エクセルシート」をダウンロードします。このエクセルファイルを開きます。

W 欄にはホイールベース、T 欄にはトレッドを入れます。自分のマイコンカーのそれぞれの長さを測って、メートル単位で入力します。

	A	B	C	D	E	F	G
1	W		0.17	m ← ホイールベースを入力してください			
2	T		0.17	m ← トレッドを入力してください			
4	度	rad	r2	r1	r3	r1/r3*100	
5	0	0				100	
6	1	0.017	9.744	9.659	9.829	98	
7	2	0.035	4.871	4.786	4.956	97	
8	3	0.052	3.245	3.160	3.330	95	
9	4	0.070	2.432	2.347	2.517	93	
10	5	0.087	1.944	1.859	2.029	92	
11	6	0.105	1.618	1.533	1.703	90	
12	7	0.122	1.385	1.300	1.470	88	
13	8	0.140	1.210	1.125	1.295	87	
14	9	0.157	1.074	0.989	1.159	85	
15	10	0.174	0.965	0.880	1.050	84	
16	11	0.192	0.875	0.790	0.960	82	
17	12	0.209	0.800	0.715	0.885	81	
18	13	0.227	0.737	0.652	0.822	79	
19	14	0.244	0.682	0.597	0.767	78	
20	15	0.262	0.635	0.550	0.720	76	



9. モータの左右回転差の計算方法

あるマイコンカーの長さを測ると、W=0.15m、T=0.15m でした。入力すると、自動で0~45 度にハンドルを切ったときの内輪回転数が計算されます。

	A	B	C	D	E	F	G
1	W	0.15	m ← ホイールベースを入力してください				
2	T	0.15	m ← トレッドを入力してください				
3							
4	度	rad	r2	r1	r3	r1/r3*100	
5	0	0				100	
6	1	0.017	8.598	8.523	8.673	98	
7	2	0.035	4.298	4.223	4.373	97	
8	3	0.052	2.864	2.789	2.939	95	
9	4	0.070	2.146	2.071	2.221	93	
10	5	0.087	1.715	1.640	1.790	92	
11	6	0.105	1.428	1.353	1.503	90	
12	7	0.122	1.222	1.147	1.297	88	
13	8	0.140	1.068	0.993	1.143	87	
14	9	0.157	0.948	0.873	1.023	85	
15	10	0.174	0.851	0.776	0.926	84	
16	11	0.192	0.772	0.697	0.847	82	
17	12	0.209	0.706	0.631	0.781	81	
18	13	0.227	0.650	0.575	0.725	79	
19	14	0.244	0.602	0.527	0.677	78	
20	15	0.262	0.560	0.485	0.635	76	
21	16	0.279	0.523	0.448	0.598	75	
22	17	0.297	0.491	0.416	0.566	73	
23	18	0.314	0.462	0.387	0.537	72	
24	19	0.331	0.436	0.361	0.511	71	
25	20	0.349	0.412	0.337	0.487	69	
26	21	0.366	0.391	0.316	0.466	68	
27	22	0.384	0.371	0.296	0.446	66	
28	23	0.401	0.354	0.279	0.429	65	
29	24	0.419	0.337	0.262	0.412	64	
30	25	0.422	0.322	0.247	0.397	62	

図形の調整(R) オートシェイプ(U) コマンド

上記 ○の「コピーして貼り付け用」タブをクリックします。

```

MS 明朝 11 B I U 三三三三 %
A11 = " & ROUND(表!G50,0)&" 

1 const revolution_difference[] = { /* 角度から内輪、外輪回転差計算 */
2   100, 98, 97, 95, 93,
3   92, 90, 88, 87, 85,
4   84, 82, 81, 79, 78,
5   76, 75, 73, 72, 71,
6   69, 68, 66, 65, 64,
7   62, 61, 59, 58, 57,
8   55, 54, 52, 51, 50,
9   48, 47, 45, 44, 42,
10  41, 39, 38, 36, 35,
11  33 };

```

A1～A11 を選択して、「右クリック→コピー」で内容をコピーします。

プログラムの、グローバル変数の最後の部分に貼り付けます。これで、自分のマイコンカーに合った左右回転差が計算されます。

```
/*=====
/* グローバル変数の宣言 */
=====*/
unsigned long cnt0;           /* timer 関数用 */
unsigned long cnt1;           /* main 内で使用 */
int pattern;                 /* パターン番号 */
int angle_buff;              /* 現在ハンドル角度保持用 */

const revolution_difference[] = {      /* 角度から内輪、外輪回転差計算 */
    100, 99, 97, 96, 95,
    93, 92, 91, 89, 88,
    87, 86, 84, 83, 82,
    81, 79, 78, 77, 76,
    75, 73, 72, 71, 70,
    69, 67, 66, 65, 64,
    62, 61, 60, 59, 58,
    56, 55, 54, 52, 51,
    50, 48, 47, 46, 44,
    43 };

```

今回、revolution_difference という回転の差を計算した配列を追加します。この配列には、const を先頭に付けています。値を変更しない変数や配列は、RAM 上に配置する必要はありません。const 型修飾子を指定すると ROM エリアに配置されます。RAM は ROM より容量が少ないので、RAM の有効活用を考えて const を付けました。

revolution_difference 配列の[]内に数字を入れると、入れた数字番目の数値と同じ意味になります。[]の中に入れる数字を添字といいます。添字は 0 から数えます。

```
revolution_difference[ 0] = 100
revolution_difference[ 1] = 99
revolution_difference[ 2] = 97
|
|
revolution_difference[45] = 43
```

というように、順番に値が返ってきます。ちなみに 46 以上の値を設定してもエラーにはなりませんが、不定な値が返ってきます。46 以上にしないようにプログラマが注意する必要があります。

値の意味は、**外輪の回転を 100 としたとき、添字に現在のハンドル角度を入れると内輪の回転数が返ってくるようにしています**。添字が 2 の時、97 が返ってきます。これは外輪 100、ハンドル角度が 2 度のとき、内輪の回転数は 97 ということです。ハンドル角度が 0 度～45 度の時の内輪の値を、あらかじめエクセルなどで計算しておきます。計算方法は後述します。

9. モータの左右回転差の計算方法

9.3.5 diff 関数の追加

diff 関数を追加します。プログラムのいちばん最後に追加します。

```
/****************************************************************************
 * 外輪のPWMから、内輪のPWMを割り出す ハンドル角度は現在の値を使用 */
/* 引数 外輪PWM */ 
/* 戻り値 内輪PWM */ 
/****************************************************************************
int diff( int pwm )
{
    int ret;

    if( pwm >= 0 ) {
        /* PWM値が正の数なら */
        if( angle_buff < 0 ) {
            angle_buff = -angle_buff;
        }
        ret = revolution_difference[angle_buff] * pwm / 100;
    } else {
        /* PWM値が負の数なら */
        ret = pwm;           /* そのまま返す */
    }
    return ret;
}
```

diff 関数は、引数に外輪(多く回るタイヤ側)の PWM 値を入れて呼び出すると、内輪(少なく回るタイヤ側)の PWM 値が返って来るという関数です。現在のハンドル角度は、angle_buff 変数から読み込みます。

例えば、今のハンドル角度が-25 度、外輪の PWM 値を 60 とするとします。

まず、875 行で角度がマイナスかどうかチェックします。-25 度なのでマイナスです。878 行でプラスに直します。次に回転差を計算します。外輪の回転数を 100 と考えて、内輪の回転数を計算します。

```
ret = revolution_difference[angle_buff]
= revolution_difference[ 25 ]
= 69
```

で内輪の回転数が出ます。

次に、外輪が 100 では無い場合、内輪は外輪の回転に比例しますので、

```
ret = 69 * 外輪の PWM 値 / 100
= 69 * 60 / 100
= 41
```

となります。結果、ハンドル角度が-25 度、外輪(右)の PWM 値を 60 とすると内輪(左)の PWM 値は 41 となります。

9.3.6 プログラムでの使い方

まず、ハンドル角度を設定します。

```
handle( 15 );
```

handle 関数は正の数で右へ、負の数で左へハンドルを切る関数ですので、15 度というと右側へハンドルを切っています。そのため、外輪が左タイヤ、内輪が右タイヤとなります。

次に motor 関数です。PWM 値を 50%にしたい場合、外輪の左タイヤを 50 に、内輪の右タイヤを diff(50) とします。どちらが外輪で、どちらが内輪かは、プログラマが判断して diff 関数を使います。

```
speed( 50 , diff(50) );
      ↑      ↑
    外輪    内輪
```

diff(50)の戻り値は、

```
revolution_difference[angle_buff] * pwm / 100
= revolution_difference[15]           * 50 / 100
= 81 * 50 / 100
= 40
```

となります。

ハンドル角度を変えると、自動で内輪側の PWM 値を計算してくれるので便利です。

10. フリー動作を追加する

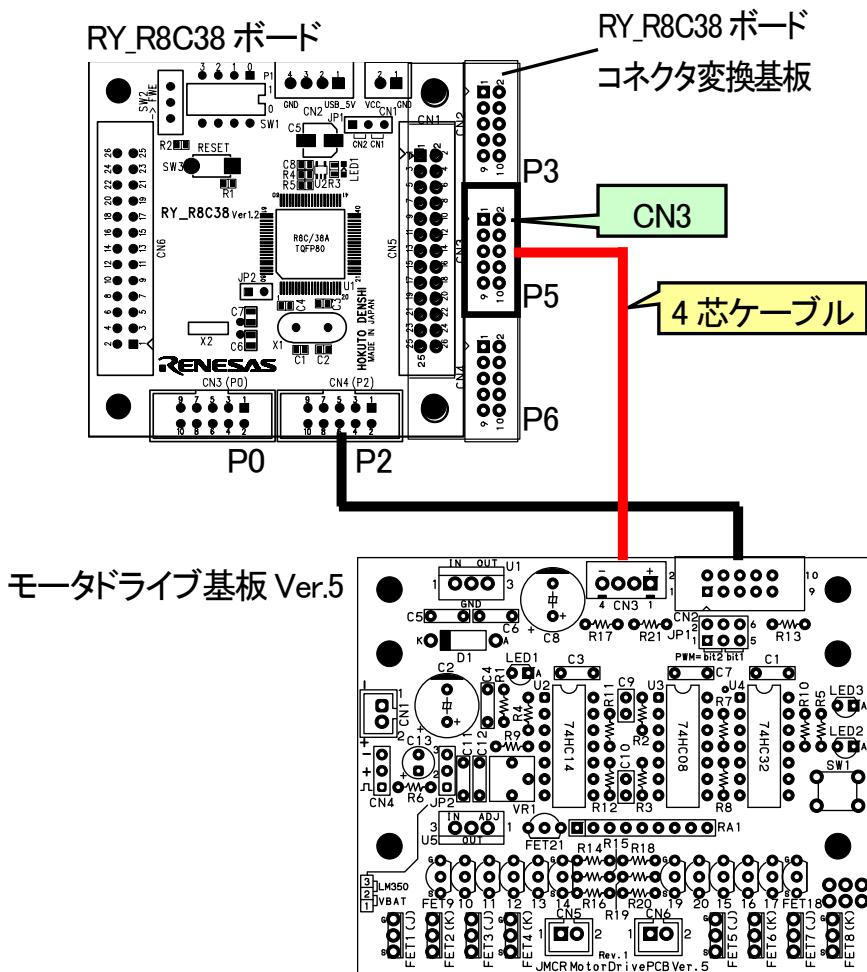
10.1 概要

モータドライブ基板 Ver.5 標準の停止は、ブレーキのみです。モータドライブ基板 Ver.5 に「フリー追加セット」を追加することにより、モータの停止状態をフリーとブレーキのどちらかを選べるようになります。

ここでは、フリー追加セットを追加したときの結線やプログラムの改造について、説明します。回路的な説明は、「4. モータドライブ基板 Ver.5」を参照してください。

10.2 接続

RY_R8C38 ボードの CN5 に「RY_R8C38 ボードコネクタ変換基板」を取り付けます。10 ピンコネクタが 3 個追加されたうちの CN3 にフリー追加セットの 4 芯ケーブルを接続します。このコネクタのポートは、ポート 5 になります。



※「RY_R8C38 ボードコネクタ変換基板」は、26 ピンコネクタを 10 ピンコネクタ 3 個に変換する基板です。詳しくは、「マイコン実習マニュアル(R8C/38A 版)」を参照してください。

10.3 プログラムの追加

10.3.1 関数の追加

(1) motor_mode 関数の追加

左モータの停止状態を設定するのは P5_1、右モータの停止状態を設定するのは P5_0 で行います。設定ごとに P5_1 や P5_0 を操作するのは効率が良くないので、停止時の動作を設定する関数「motor_mode」を作ります。

motor_mode 関数を、下記に示します。プログラムは、kit12_38a.c の handle 関数の次などに入れてください。

```
/*********************************************
/* 停止状態制御 */
/* 引数 左モータの状態 : BRAKE or FREE */
/* 右モータの状態 : BRAKE or FREE */
/********************************************/

void motor_mode( int mode_l, int mode_r )
{
    if( mode_l ) { /* 左モータチェック */          /* フリー動作 */
        p5_1 = 1;                                /* フリー動作 */
    } else {                                     /* ブレーキ動作 */
        p5_1 = 0;                                /* ブレーキ動作 */
    }
    if( mode_r ) { /* 右モータチェック */          /* フリー動作 */
        p5_0 = 1;                                /* フリー動作 */
    } else {                                     /* ブレーキ動作 */
        p5_0 = 0;                                /* ブレーキ動作 */
    }
}
```

(2) プロトタイプ宣言の追加

関数を新しく作った場合、プロトタイプ宣言（使用する関数の宣言）が必要です。kit12_38a.c の初めの方にプロトタイプ宣言をしている部分がありますので、その最後に追加しておきます。

```
/*=====
/* プロトタイプ宣言 */
=====*/
void init( void );
void timer( unsigned long timer_set );
int check_crossline( void );
int check_rightline( void );
int check_leftline( void );
unsigned char sensor_inp( unsigned char mask );
unsigned char dipsw_get( void );
unsigned char pushsw_get( void );
unsigned char startbar_get( void );
void led_out( unsigned char led );
void motor( int accele_l, int accele_r );
void handle( int angle );
void motor_mode( int mode_l, int mode_r ); ←追加
```

(3) シンボル定義の追加

最後に、motor_mode 関数の引数に入れる内容をシンボル定義しておきます。

```
/*=====
/* シンボル定義
=====
/* 停止時の動作 */
#define FREE      1          /* フリー          */
#define BRAKE    0          /* ブレーキ        */
/*=====
```

中略

10.3.2 使い方

motor_mode 関数の使い方を下記に示します。

```
motor_mode( 左モータの停止状態 , 右モータの停止状態 );
```

引数は、左モータ、右モータの停止状態を設定します。設定値を、下記に示します。

引数	説明
FREE	停止時の状態をフリー動作にします。
BRAKE	停止時の状態をブレーキ動作にします。

プログラム例を、下記に示します。

motor_mode(FREE, BRAKE);	左モータはフリー、右モータはブレーキ
----------------------------	--------------------

motor_mode(BRAKE, BRAKE);	左モータはブレーキ、右モータはブレーキ
-----------------------------	---------------------

motor_mode(FREE, FREE);	左モータはフリー、右モータはフリー
---------------------------	-------------------

10.3.3 ブレーキとフリーの違いを確認する

kit12_38a.c プログラムに下記の太字部分のプログラムを追加します。このプログラムを実行すると、右モータ、左モータはPWM50%で回転します(ディップスイッチがすべて"1"の場合)。このとき、モータドライブ基板のプッシュスイッチを押すと停止動作はフリー、離していると停止動作はブレーキになります。

タイヤを浮かせた状態でモータを回し、プッシュスイッチを押しているときと離しているときでモータの回転状態を確認してください。確認が終わったら、追加した部分を消しておきます。

```
/*********************************************
/* メインプログラム
/*********************************************
void main( void )
{
    int      i;

    /* マイコン機能の初期化 */
    init();                                /* 初期化          */
    asm(" fset I ");                      /* 全体の割り込み許可      */

    /* マイコンカーの状態初期化 */
    handle( 0 );
    motor( 0, 0 );

    // 追加 ここから
    while( 1 ) {
        if( cnt0 < 1000) {
            motor( 50, 50 );
        } else {
            motor( 0, 0 );
            if( cnt0 >= 2000 ) cnt0 = 0;
        }
        if( pushsw_get() ) {
            motor_mode( FREE, FREE );    // プッシュスイッチが押されていたらフリー動作
        } else {
            motor_mode( BRAKE, BRAKE ); // プッシュスイッチが離されていたらブレーキ動作
        }
    }
    // 追加 ここまで

    while( 1 ) {
        switch( pattern ) {
```

10.3.4 走行プログラムにmotor_mode関数を組み込む

motor_mode 関数は、motor 関数とペアで使います。下記に使用例を示します。

```

case 11:
    /* 通常トレース */
中略
    switch( sensor_inp(MASK3_3) ) {
        case 0x00:
            /* センタ→まっすぐ */
            handle( 0 );
            motor_mode( FREE, FREE );
            motor( 100 ,100 );
            break;

        case 0x04:
            /* 微妙に左寄り→右へ微曲げ */
            handle( 5 );
            motor_mode( FREE, FREE );
            motor( 100 ,100 );
            break;

        case 0x06:
            /* 少し左寄り→右へ小曲げ */
            handle( 10 );
            motor_mode( BRAKE, FREE );
            motor( 80 ,67 );
            break;

        case 0x07:
            /* 中くらい左寄り→右へ中曲げ */
            handle( 15 );
            motor_mode( BRAKE, FREE );
            motor( 50 ,38 );
            break;

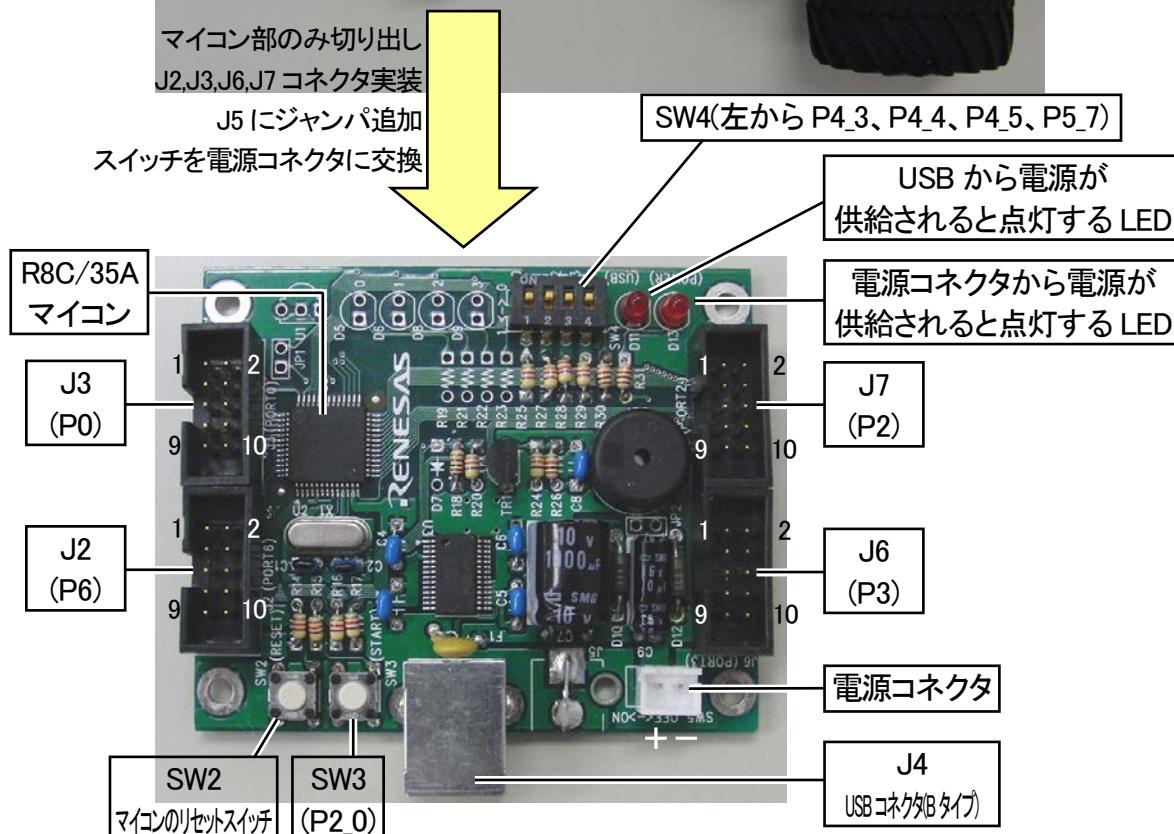
        case 0x03:
            /* 大きく左寄り→右へ大曲げ */
            handle( 25 );
            motor_mode( BRAKE, BRAKE );
            motor( 30 ,19 );
            pattern = 12;
            break;
    }
}

```

11. RMC-R8C35A ボードを使う

11.1 RMC-R8C35A ボード

RMC-R8C35A ボードは、ミニマイコンカーバージョン Ver.2 のマイコン部分を切り出した基板です。マイコン部分単体でも販売されています。RMC-R8C35A ボードは、ルネサス エレクトロニクス製の R8C/35A マイコンを搭載しています。



※ミニマイコンカーバージョン Ver.2 については、(株)日立ドキュメントソリューションズの
マイコンカー販売サイト(<http://www2.himdx.net/mcr/>)を参照してください。

※マイコン部分単体の販売もあります。

※公開されているプログラムは、センサ基板 Ver.4 を使ったプログラムです。センサ基板 Ver.5 を使う場合は、
sensor_inp 関数、startbar_get 関数を修正してください。

11.2 RMC-R8C35AボードとRY_R8C38 ボード

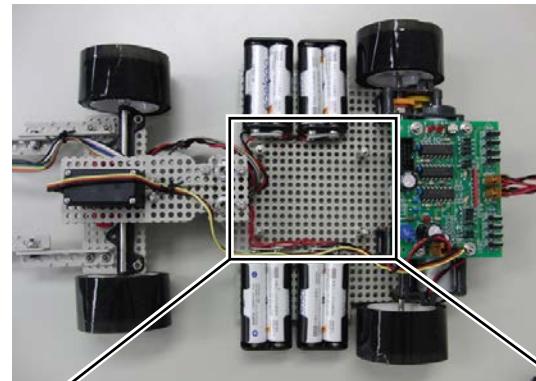
RMC-R8C35A ボードと RY_R8C38 ボードの違いを下表に示します。

「R8C/38A=R8C/35A+ α 」の関係です。

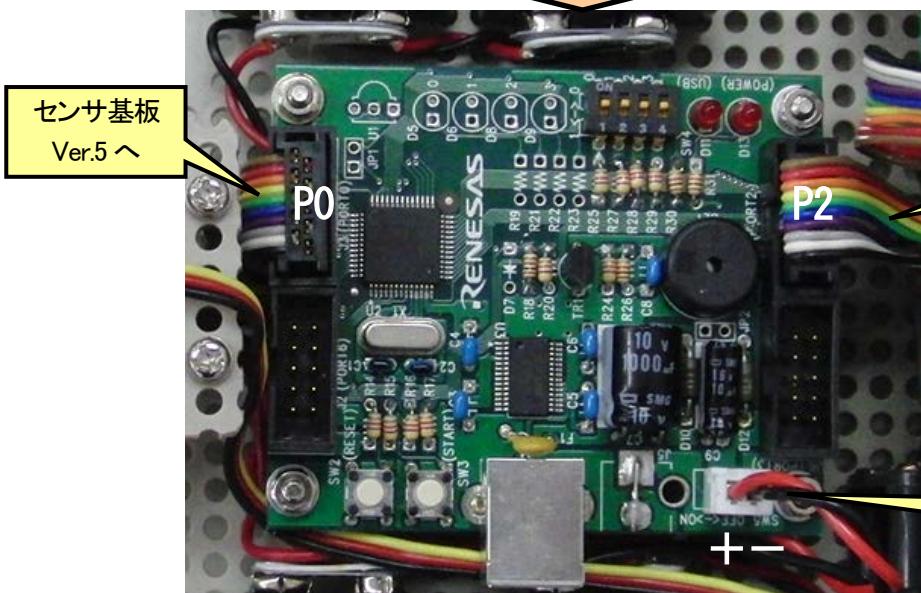
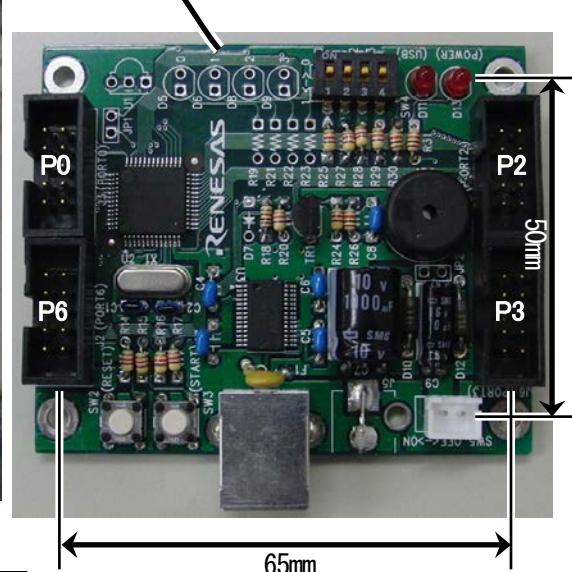
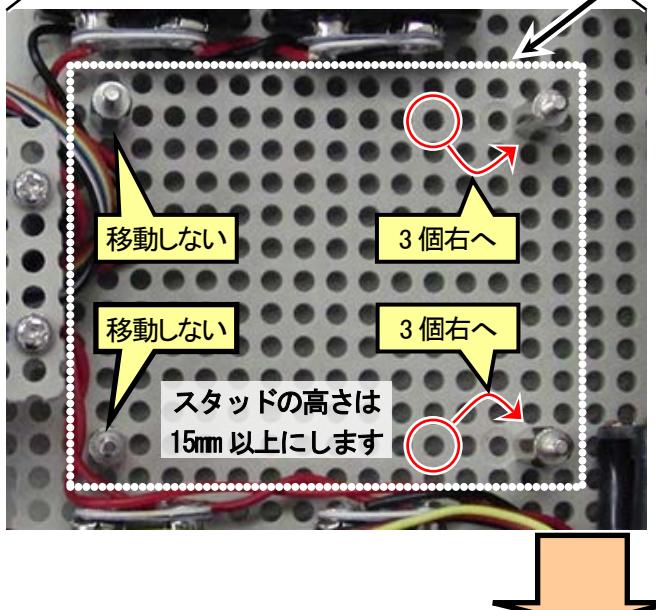
		RMC-R8C35Aボード	RY_R8C38ボード																								
マイコン	ルネサス エレクトロニクス製 R8C/35A (R5F21356ANFP)	ルネサス エレクトロニクス製 R8C/38A (R5F2138CANFP)																									
パッケージ	52ピンLQFP	80ピンLQFP																									
ボードのクリスタル値	20.0MHz	20.0MHz																									
動作電圧	動作周波数が5.0～20.0MHzの場合:2.7～5.5V 動作周波数が2.0～5.0MHzの場合:1.8～5.5V	動作周波数が5.0～20.0MHzの場合:2.7～5.5V 動作周波数が2.0～5.0MHzの場合:1.8～5.5V																									
基板外形	W72×D60×H13mm(実寸)	W60×D60×H13mm(実寸)																									
内蔵メモリ	ROM	32KB : 0x8000～0xffff番地 プログラム、イレーズ(消去)回数:保証回数1000回	128KB : 0x4000～0x23fff番地 プログラム、イレーズ(消去)回数:保証回数1000回																								
	RAM	2.5KB : 0x0400～0x0dff番地	10KB : 0x0400～0x2bff番地																								
	データ フラッシュ	4KB : 0x3000～0x3fff番地 プログラム、イレーズ(消去)回数:保証回数10000回	4KB : 0x3000～0x3fff番地 プログラム、イレーズ(消去)回数:保証回数10000回																								
コネクタ	<table border="1"> <tr><td>コネクタ(ポート)</td><td>I/O数</td></tr> <tr><td>J3(ポート0)</td><td>8</td></tr> <tr><td>J7(ポート2)</td><td>8</td></tr> <tr><td>J6(ポート3)</td><td>8</td></tr> <tr><td>J2(ポート6)</td><td>8</td></tr> <tr><td>合計</td><td>32</td></tr> </table>		コネクタ(ポート)	I/O数	J3(ポート0)	8	J7(ポート2)	8	J6(ポート3)	8	J2(ポート6)	8	合計	32	<table border="1"> <tr><td>コネクタ(ポート)</td><td>I/O数</td></tr> <tr><td>CN3(ポート0)</td><td>8</td></tr> <tr><td>CN4(ポート2)</td><td>8</td></tr> <tr><td>CN5(ポート3, 5, 6)</td><td>24</td></tr> <tr><td>CN6(ポート7, 8など)</td><td>24</td></tr> <tr><td>合計</td><td>64</td></tr> </table>	コネクタ(ポート)	I/O数	CN3(ポート0)	8	CN4(ポート2)	8	CN5(ポート3, 5, 6)	24	CN6(ポート7, 8など)	24	合計	64
コネクタ(ポート)	I/O数																										
J3(ポート0)	8																										
J7(ポート2)	8																										
J6(ポート3)	8																										
J2(ポート6)	8																										
合計	32																										
コネクタ(ポート)	I/O数																										
CN3(ポート0)	8																										
CN4(ポート2)	8																										
CN5(ポート3, 5, 6)	24																										
CN6(ポート7, 8など)	24																										
合計	64																										
※コネクタはすべてオプションです。		※CN5、CN6のコネクタはオプションです。																									
マイコンの内蔵機能	タイマRA:1ch タイマRB:1ch タイマRC:1ch タイマRD:2ch タイマRE:1ch 10ビットA/Dコンバータ:12ch 8ビットD/Aコンバータ:2回路 UART:3ch 外部割り込み入力:9本	タイマRA:1ch タイマRB:1ch タイマRC:1ch タイマRD:2ch タイマRE:1ch タイマRF:1ch タイマRG:1ch 10ビットA/Dコンバータ:20ch 8ビットD/Aコンバータ:2回路 UART:3ch 外部割り込み入力:9本																									
LED	電源モニタ用LED:1個 USB電源用LED:1個 P1_3～P1_0接続LED:4個(オプション)	P4_5接続LED:1個 (P4_5が入力設定の場合は点灯)																									
圧電サウンダ	搭載	未搭載																									
USB変換回路	搭載	未搭載(別売りのRY-WRITER基板などを使用)																									
ディップスイッチ	P5_7、P4_5、P4_4、P4_3に接続(4ビット分)	P1_3～P1_0に接続(4ビット分)																									

11.3 RMC-R8C35Aボードの搭載

RMC-R8C35A ボードは、RY_R8C38 ボードより横方向が 15mm 長くなっています。そのため、マイコンボードを止めるスタッドも、横の長さを 15mm 伸ばします。また USB コネクタ(J4)と電池がぶつからないように、スタッドの高さを 15~20mm 程度にします。

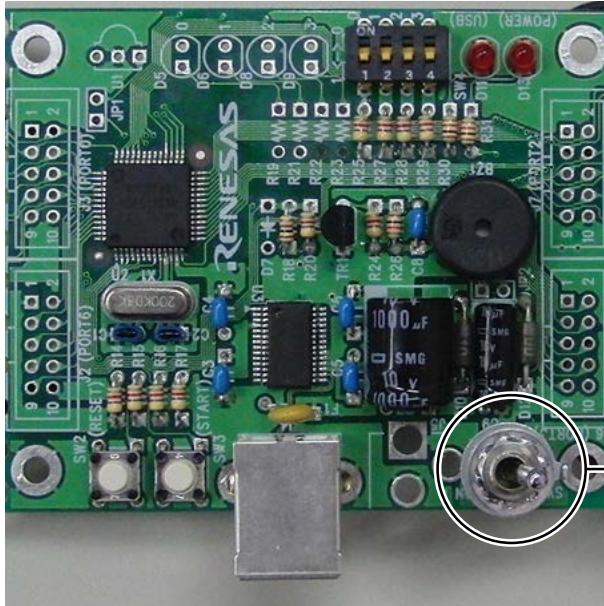
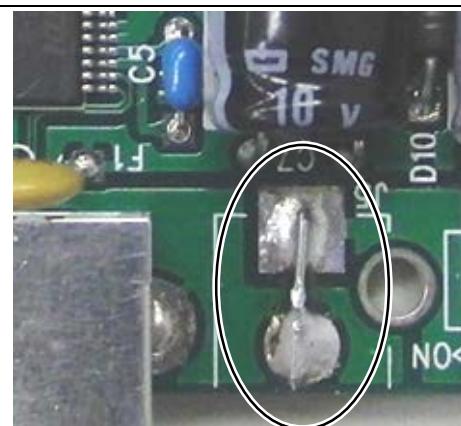
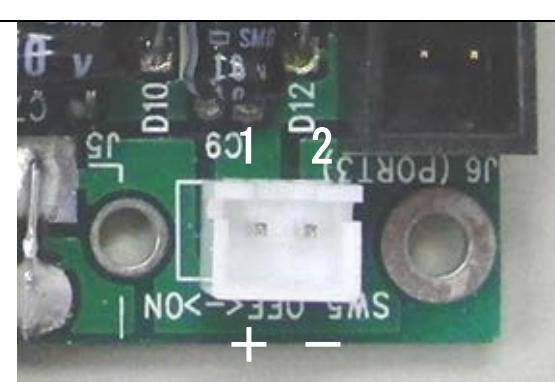


※スタッドの位置は、RY_R8C38 ボードと
比べたときの位置です。



11.4 RMC-R8C35Aボードの電源スイッチをコネクタに変更する

RMC-R8C35A ボードは、DC ジャック(オプション)から電源供給します。今までのマイコンボードは、2 ピンコネクタで電源供給していたので、コネクタ経由で電源供給するよう改造します。

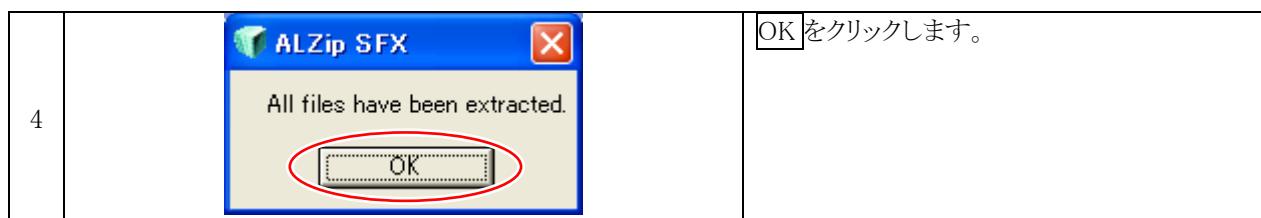
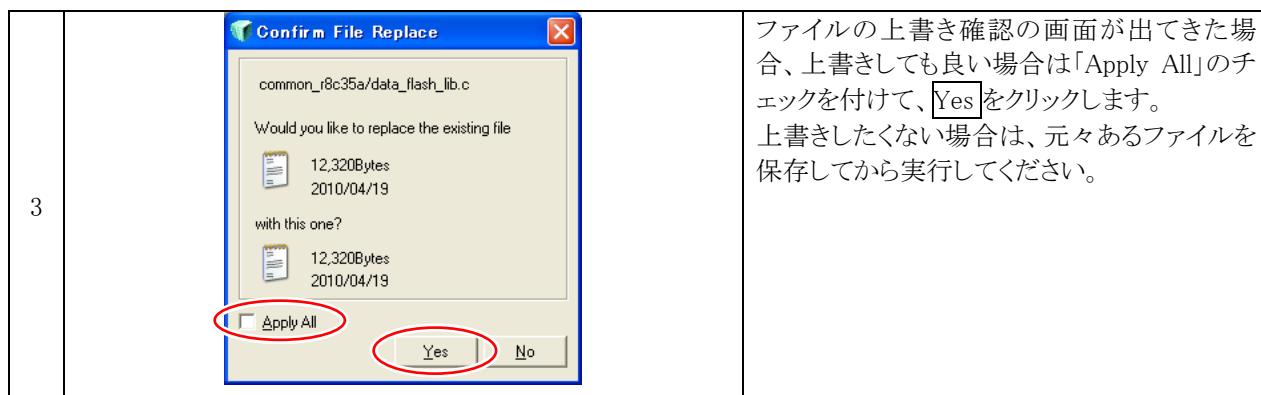
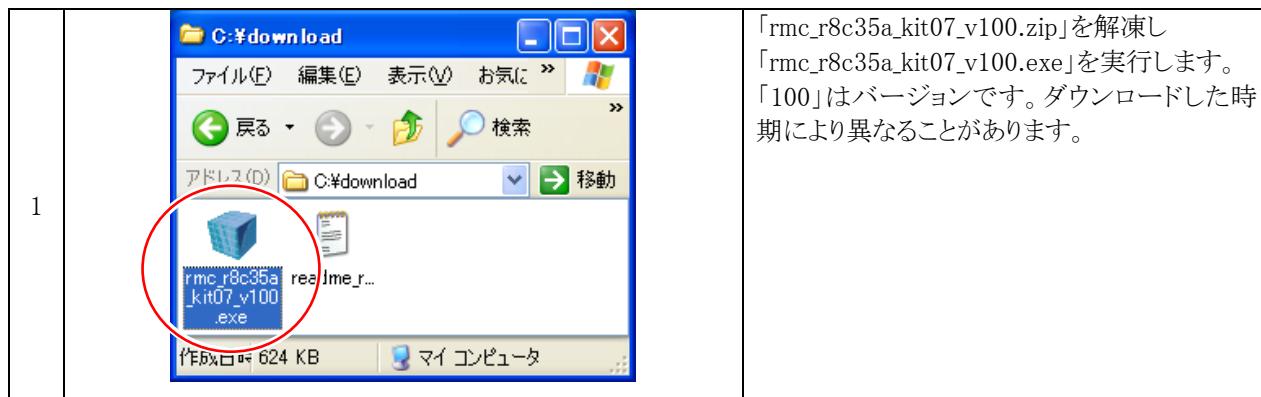
1		スイッチ部分をコネクタに変更します。 スイッチ(SW5)を取ります。
2		J5 にある 3 つのランドの内、縦に並んでいる ランドをジャンパでショートさせます。
3		SW5 の部分に手持ちのコネクタを実装します。 写真は、日本圧着端子製造(株)の XH コネクタ(B2B-XH-A)を 実装したところです。

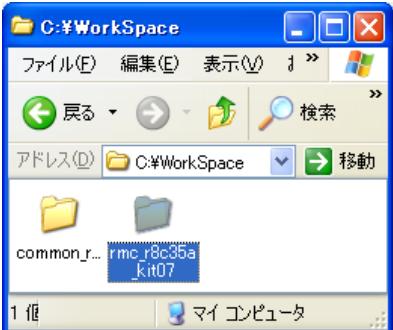
11.5 サンプルプログラムのインストール

11.5.1 ホームページからソフトを取得する

1	 <p>ミニマイコンカー製作キット Ver.2を販売開始しました。 商品の詳細は製品情報ページを参照ください。</p>	<p>(株)日立ドキュメントソリューションズのマイコンカーラリー販売サイト (https://www2.himdx.net/mcr/) にアクセスします。 製品情報をクリックします。</p>
2	<p>マイコンボード</p> <p>RMC-R8C35A</p>  <p>RMC-R8C35Aは、ミニマイコンカー製作キット Ver.2のマイコンボード部分を製品化することにより低価格を実現したマイコンボードです。 ワンボードで、DIPスイッチを使用したLEDの点灯制御やブザーを使用した電子オルゴールなどの制御の学習ができます。</p> <p>→ 詳細はこちら</p> <p>↑ ページの先頭へ</p>	<p>「RMC-R8C35A」をクリックします。</p>
2	<p>プログラム</p> <p>C言語でプログラムを開発するためには、ルネサス エレクトロニクスの統合開発環境 High-performance Embedded Workshop(HEW)のインストールが必要になります。 HEWのインストール方法や設定などについては、</p> <p>ミニマイコンカー製作キット Ver.2 R8C/35Aマイコン実習マニュアルを参照してください。</p> <p>RMC-R8C35A 走行プログラム Ver.1.00</p> <p>(更新日：2011年04月19日) RMC-R8C35Aをマイコンカー製作キットVer.4に搭載して、走行するプログラムのHEWのワークスペースです。 ※zipを解凍して「rmc_r8c35a_kit07_v100.exe」を実行してください。</p> <p>ワークスペース(rmc_r8c35a_kit07_v100.zip(zip形式 592Kバイト))</p> <p>※その他のマニュアル、開発環境、プログラムなどについては、 ミニマイコンカー製作キット Ver.2の製品情報ページの参照をお願いします。</p> <p>↑ ページの先頭へ</p>	<p>「ワークスペース (rmc_r8c35a_kit07_v100).zip (zip 形式 592K バイト)」をクリックして、「rmc_r8c35a_kit07_v100.zip」をダウンロードします。</p> <p>※「100」はバージョンです。ダウンロードした時期により異なることがあります。</p>

11.5.2 インストール

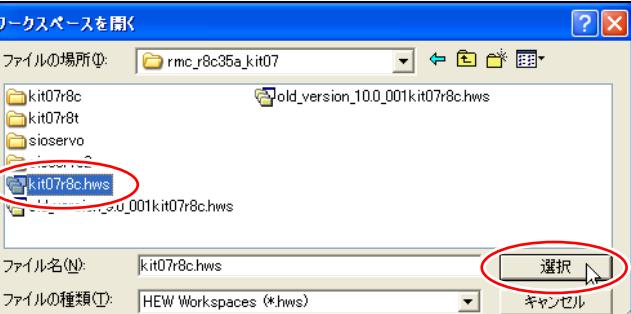


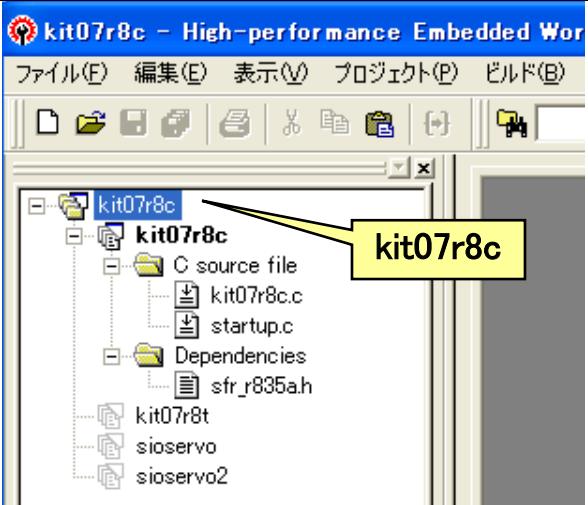
5		<p>「C:\WorkSpace」フォルダに、次の 2 つのフォルダができます。</p> <ul style="list-style-type: none"> • rmc_r8c35a_kit07 • common_r8c35a
---	---	--

11.6 ワーススペース「kit07r8c」を開く

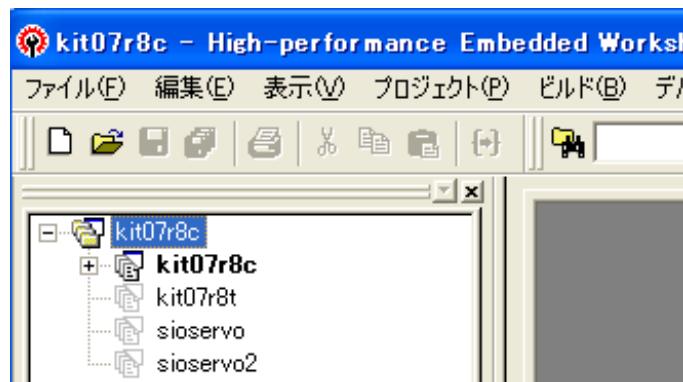
1		<p>ルネサス統合開発環境を実行します。</p>
---	---	--------------------------

2		<p>「別のプロジェクトワークスペースを参照する」を選択します。</p>
---	--	--------------------------------------

3		<p>C ドライブ → Workspace → rmc_r8c35a_kit07 の「kit07r8c.hws」を選択します。</p>
---	---	---

5		<p>ワークスペース「kit07r8c」が開かれます。</p>
---	---	---------------------------------

11.7 プロジェクト



ワークスペース「kit07r8c」には、4 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
kit07r8c	マイコンカー走行プログラムです。 ワークスペース「kit07_38a」のプロジェクト「kit07_38a」の R8C/35A 版です。
kit07r8t	製作したマイコンカーのモータドライブ基板やセンサ基板が正しく動作するかテストします。 ワークスペース「kit07_38a」のプロジェクト「kit07test_38a」の R8C/35A 版です。
sioservo	サーボのセンタを調整するプログラムです。 ワークスペース「kit07_38a」のプロジェクト「sioservo1_38a」の R8C/35A 版です。
sioservo2	サーボの最大切れ角を見つけるためのプログラムです。 ワークスペース「kit07_38a」のプロジェクト「sioservo2_38a」の R8C/35A 版です。

※プログラムは、センサ基板 Ver.4 用に作られています。センサ基板 Ver.5 を使用する場合は、sensor_inp 関数、startbar_get 関数の内容を変更してください。変更内容は、『6.2 「kit07_38a.c」と「kit12_38a.c」プログラムの相違点』部分のそれぞれの関数部分の説明を参照してください。

12. 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 ファミリ用 C/C++コンパイラパッケージ V.6.00
C/C++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.4.09 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/>

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

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

の「製品情報」欄→「マイコン」→「R8C」でご覧頂けます

版	日付	ページ	内容
1.11	2014.06.02	全体	JMCR2015 の内容に更新