

マイコンカーラリーキット Ver.4 対応

# プログラム 解説マニュアル kit07版

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下さい。

第 1.04 版

2007.09.14

ジャパンマイコンカーラリー実行委員会

# 注意事項 (rev.1.2)

## 著作権

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

## 禁止事項

ユーザーは以下の内容を行うことはできません。

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

## 転載、複製

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

## 責任の制限

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

## その他

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

## 連絡先

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

# 目 次

<b>1. 概要</b> .....	<b>1</b>
<b>2. マイコンカーコース、スタートバーの仕様</b> .....	<b>2</b>
2.1 基本的なコース .....	2
2.2 上り、下り坂 .....	2
2.3 クロスラインからクランク部分 .....	3
2.4 レーンチェンジ部分 .....	3
2.5 スタートバー部分 .....	4
<b>3. マイコンカーキットの仕様</b> .....	<b>5</b>
3.1 外観 .....	5
3.2 標準キットの電源構成 .....	7
3.3 駆動系電圧を上げた電源構成 .....	8
<b>4. センサ基板</b> .....	<b>9</b>
4.1 仕様 .....	9
4.2 回路図 .....	10
4.3 基板寸法 .....	11
4.4 センサ取り付け寸法 .....	11
4.5 コースの白と黒を判断する仕組み .....	12
4.6 スタートバーの開閉を判断する仕組み .....	13
4.7 10ピンコネクタ .....	14
4.8 信号の流れ .....	15
4.9 回路の原理 .....	16
4.10 センサの調整方法 .....	17
<b>5. モータドライブ基板</b> .....	<b>20</b>
5.1 仕様 .....	20
5.2 回路図 .....	21
5.3 寸法 .....	22
5.4 機能 .....	23
5.5 コネクタ .....	24
5.5.1 10ピンコネクタ .....	24
5.5.2 駆動系電源コネクタ .....	25
5.5.3 モータコネクタ .....	25
5.5.4 サーボコネクタ .....	26
5.6 モータ制御 .....	27
5.6.1 モータドライブ基板の役割 .....	27
5.6.2 スピード制御の原理 .....	27
5.6.3 正転、逆転、ブレーキの原理 .....	28
5.6.4 Hブリッジ回路 .....	29
5.6.5 Hブリッジ回路のスイッチをFETにする .....	29
5.6.6 PチャネルとNチャネルの短絡防止回路 .....	32
5.6.7 モータドライブ基板の回路 .....	35
5.7 サーボ制御 .....	36
5.7.1 原理 .....	36

5.7.2 回路.....	37
5.8 LED 制御.....	37
5.9 スイッチ制御.....	38
<b>6. サンプルプログラム.....</b>	<b>39</b>
6.1 ルネサス統合開発環境.....	39
6.2 サンプルプログラムのインストール.....	39
6.2.1 CD からソフトを取得する.....	39
6.2.2 ホームページからソフトを取得する.....	39
6.2.3 インストール.....	40
6.3 ワーススペース「kit07」を開く.....	41
6.4 プロジェクト.....	42
<b>7. プロジェクト内のファイルの関わりと実行順.....</b>	<b>43</b>
7.1 概要.....	43
7.2 プロジェクトのファイル構成.....	43
7.3 プログラムの実行順.....	44
7.3.1 電源を入れたときの動作.....	44
7.3.2 マイコンの動作開始.....	44
7.3.3 ベクタアドレスからジャンプ先アドレスを取り出す.....	45
7.3.4 スタートアップルーチンの実行.....	48
7.3.5 スタックポインタの設定.....	49
7.3.6 INIT_SCT 関数の実行.....	51
7.3.7 main 関数の実行.....	51
7.3.8 IMPORT 宣言.....	52
<b>8. プログラム解説「kit07.c」.....</b>	<b>53</b>
8.1 プログラムリスト.....	53
8.2 スタート.....	61
8.3 外部ファイルの取り込み(インクルード).....	62
8.4 その他のシンボル定義.....	62
8.5 プロトタイプ宣言.....	64
8.6 グローバル変数の宣言.....	65
8.7 メインプログラムを説明する前に.....	66
8.8 H8/3048F-ONE 内蔵周辺機能の初期化:init 関数.....	66
8.8.1 プログラム.....	66
8.8.2 ポートの接続.....	67
8.8.3 入出力を決める.....	67
8.8.4 実際の設定.....	68
8.8.5 ポート B の詳細.....	68
8.8.6 ポート B の初期出力値.....	69
8.8.7 PBDR と PBDDR の設定する順番.....	69
8.9 ITU0 1ms ごとの割り込み設定.....	70
8.9.1 ITU0 レジスタの設定.....	70
8.9.2 割り込みプログラム.....	71
8.9.3 「#pragma interrupt」の設定.....	71
8.9.4 全体の割り込みを許可する.....	71
8.9.5 ベクタアドレスの設定(src ファイル).....	71
8.9.6 「.IMPORT」の設定(src ファイル).....	72
8.10 リセット同期 PWM モードの設定.....	72

8.11	時間稼ぎ:timer 関数 .....	75
8.12	コースのセンサ状態読み込み:sensor_inp 関数(センサ基板 Ver.4 仕様) .....	76
8.12.1	センサ基板 4 以前の信号変換 .....	76
8.12.2	センサ基板 4 の信号変換 .....	77
8.12.3	プログラム .....	78
8.12.4	マスク .....	80
8.12.5	まとめ .....	83
8.12.6	注意点 .....	84
8.13	クロスライン検出処理:check_crossline 関数(kit07.c で変更) .....	85
8.14	右ハーフライン検出処理:check_rightline 関数(kit07.c で変更) .....	86
8.15	左ハーフライン検出処理:check_leftline 関数(kit07.c で変更) .....	87
8.16	ディップスイッチの読み込み:dipsw_get 関数 .....	89
8.17	プッシュスイッチの読み込み:pushsw_get 関数 .....	90
8.18	スタートバー検出センサ読み込み:startbar_get 関数(kit07.c で変更) .....	91
8.19	LED の制御:led_out 関数 .....	92
8.20	モータ速度制御:speed 関数 .....	94
8.21	サーボハンドル操作:handle 関数 .....	99
8.22	メインプログラム .....	100
8.22.1	スタート .....	100
8.22.2	パターン方式について .....	100
8.22.3	プログラムの作り方 .....	101
8.22.4	パターンの内容 .....	103
8.22.5	パターン方式の最初 while、switch 部分 .....	105
8.22.6	パターン 0:スイッチ入力待ち .....	106
8.22.7	パターン 1:スタートバーが開いたかチェック .....	108
8.22.8	パターン11:通常トレース .....	109
8.22.9	パターン12:右へ大曲げの終わりのチェック .....	118
8.22.10	パターン13:左へ大曲げの終わりのチェック .....	121
8.22.11	パターン 21:1 本目のクロスライン検出時の処理 .....	124
8.22.12	パターン 23:クロスライン後のトレース、クランク検出 .....	127
8.22.13	パターン 31、32:左クランククリア処理 .....	130
8.22.14	パターン 41、42:右クランククリア処理 .....	133
8.22.15	右レーンチェンジ概要 .....	136
8.22.16	パターン 51:1 本目の右ハーフライン検出時の処理 .....	137
8.22.17	パターン 53:右ハーフライン後のトレース .....	140
8.22.18	パターン 54:右レーンチェンジ終了のチェック .....	142
8.22.19	左レーンチェンジ概要 .....	144
8.22.20	パターン 61:1 本目の左ハーフライン検出時の処理 .....	145
8.22.21	パターン 63:左ハーフライン後のトレース .....	148
8.22.22	パターン 64:左レーンチェンジ終了のチェック .....	150
8.22.23	どれでもないパターン .....	152
8.23	センサ基板 Ver.4 にしたときのプログラム変更点 .....	152
<b>9.</b>	<b>プログラム解説「kit07start.src」 .....</b>	<b>153</b>
9.1	プログラムリスト .....	153
9.2	概要 .....	154
<b>10.</b>	<b>プログラム解説「car_printf2.c」 .....</b>	<b>155</b>
10.1	プログラムリスト .....	155
10.2	概要 .....	158

10.3	宣言されている関数 .....	158
10.4	シンボル定義 .....	159
<b>11.</b>	<b>プロジェクト「kit07」のツールチェーンの設定 .....</b>	<b>160</b>
11.1	ツールチェーン .....	160
11.2	コンパイラの設定 .....	160
11.3	アセンブラの設定 .....	162
11.4	最適化リンカの設定 .....	163
11.5	標準ライブラリの設定 .....	164
11.6	CPU の設定 .....	165
<b>12.</b>	<b>モータの左右回転差の計算方法 .....</b>	<b>166</b>
12.1	計算方法 .....	166
12.2	内輪を計算するエクセルシートの作成 .....	167
12.3	サンプルエクセルシートを使った内輪の計算 .....	169
<b>13.</b>	<b>サーボセンタと最大切れ角の調整 .....</b>	<b>170</b>
13.1	概要 .....	170
13.2	通信ソフトをインストールする .....	171
13.2.1	Tera Term Pro のインストール .....	171
13.2.2	Tera Term Pro の使い方 .....	173
13.2.3	COM 番号の増やし方 .....	175
13.3	サーボのセンタを調整する .....	176
13.4	サーボの最大切れ角を見つける .....	180
13.5	「kit07.c」プログラムを書き換える .....	183
13.6	プロジェクト「sioservo」 サーボセンタの調整のプログラム解説 .....	186
13.6.1	プロジェクトの構成 .....	186
13.6.2	変数の宣言 .....	186
13.6.3	プログラムスタートのメッセージ .....	186
13.6.4	メイン関数 .....	187
13.7	プロジェクト「sioservo2」 サーボの切れ角を確かめるプログラムの解説 .....	188
13.7.1	プロジェクトの構成 .....	188
13.7.2	変数の宣言 .....	188
13.7.3	プログラムスタートのメッセージ .....	188
13.7.4	メイン関数 .....	189
<b>14.</b>	<b>プログラムの改造ポイント .....</b>	<b>190</b>
14.1	概要 .....	190
14.2	脱輪事例 .....	191
14.2.1	クロスラインの検出がうまくいかない .....	191
14.2.2	クランクの検出がうまくいかない .....	192
14.2.3	ハーフラインの検出がうまくいかない .....	194
14.2.4	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 .....	195
14.2.5	レーンチェンジ終了の判断ができない .....	198
14.3	まとめ .....	199
<b>15.</b>	<b>Basic Class (高校生初心者部門) .....</b>	<b>200</b>
15.1	概要 .....	200
15.2	参加資格 .....	200
15.3	レギュレーション .....	201

<b>16. RY3687N ボードを使用したマイコンカー .....</b>	<b>202</b>
16.1 外観.....	202
16.2 標準キットの電源構成.....	203
<b>17. サンプルプログラム(H8/3687F 版) .....</b>	<b>204</b>
17.1 サンプルプログラムのインストール .....	204
17.1.1 CD からソフトを取得する.....	204
17.1.2 ホームページからソフトを取得する .....	204
17.1.3 インストール .....	205
17.2 ワーススペース「kit07_3687」を開く .....	206
17.3 プロジェクト.....	207
17.4 サーボのセンタを調整する .....	208
17.5 サーボの最大切れ角を見つける.....	211
17.6 「kit07_3687.c」プログラムを書き換える .....	214
<b>18. 参考文献.....</b>	<b>217</b>

# 1. 概要

本マニュアルでは、

- ・マイコンカーラリーキット Ver.4 の仕様、回路
- ・RY3048Fone ボードの使い方
- ・ルネサスマイコン H8/3048F-ONE の内蔵周辺機能(PWM 機能、タイマ機能)の使い方
- ・マイコンカー制御プログラム

について解説しています。

年度と車体、プログラム、ルールの変遷を下記に示します。

	車体の変遷	プログラムの変遷	主なルール変更
1998 年頃	<b>マイコンカーキット Vol.1</b> 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	<b>マイコンカーキット Vol.2</b> を開発しました。モータドライブ基板に電池 8 本分の電圧を加えられる回路になりました。センサ基板が小型化されました。	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)
2003 年			
2004 年		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	モータドライブ基板が Vol.3 となり、逆転できるようになりました(今まではできませんでした)。キットの内容はモータドライブ基板の変更のみですが、混乱を避けるため名称を「 <b>マイコンカーキット Vol.3</b> 」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	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、レーンチェ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、レーンチェンジコースが導入されました。
2007 年	<b>マイコンカーキット Ver.4</b> 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。	<b>Basic Class</b> (高校生の初心者部門) がプレ大会として導入されました。

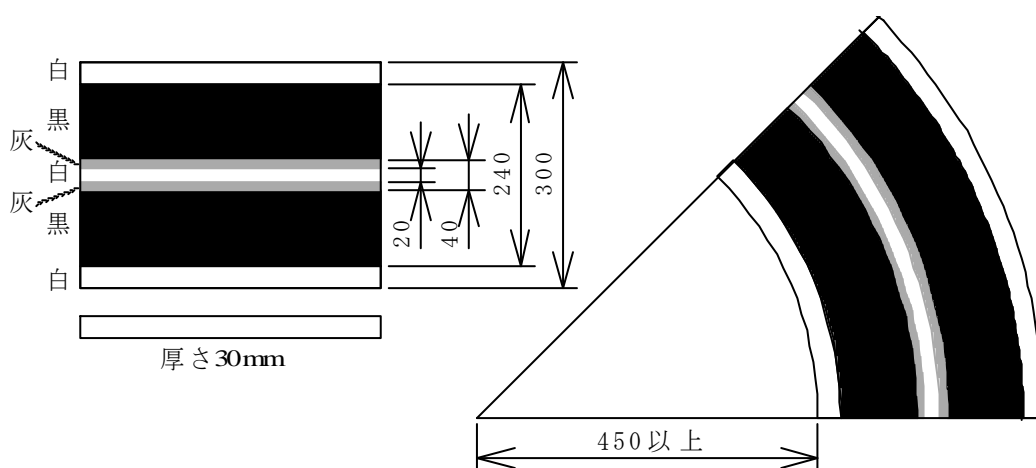


## 2. マイコンカーコース、スタートバーの仕様

### 2.1 基本的なコース

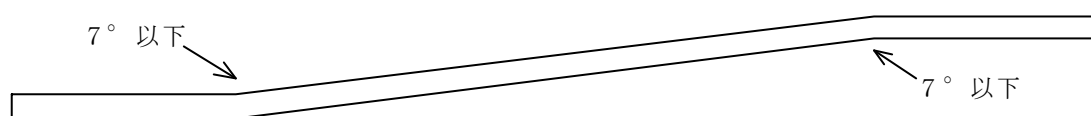
マイコンカーラリーのコースは、直線、カーブ、クランク、坂、レーンチェンジで構成されています。マイコンカーラリーのコースは、幅 300mm の中に、黒、灰、白色があります。カーブは内径が 450mm 以上となっています。マイコンカーに取り付けているセンサによりコースとマイコンカーのずれを検出し、コースに沿って走るように制御します。

マイコンカーキットには、コースの色が白色か黒色か判断することができるセンサが 7 個取り付けられています (センサ基板 Ver.4 の場合)。灰色はセンサ基板のボリュームにより、反応させるか (白色とみるか)、させないか (黒色とみるか) を調整することができます。これから説明するプログラムは、灰色を白色と同じ反応になるよう調整するとうまく走行できるようになっています。



### 2.2 上り、下り坂

角度が 7 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ン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ツ部 (7 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、アースは含む) 以外はコースに接触してはいけません。

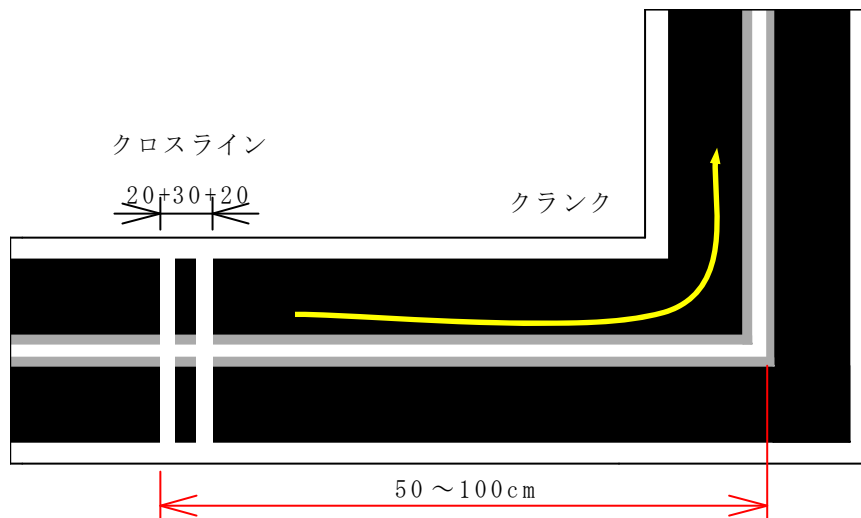
2 輪タイプでコース接触部にコース保護材をつけたものはタイヤの一部と見なします。

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、保護材等で対処をお願いします (エンコーダやリミットスイッチ、センサ含)。

## 2.3 クロスラインからクランク部分

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。

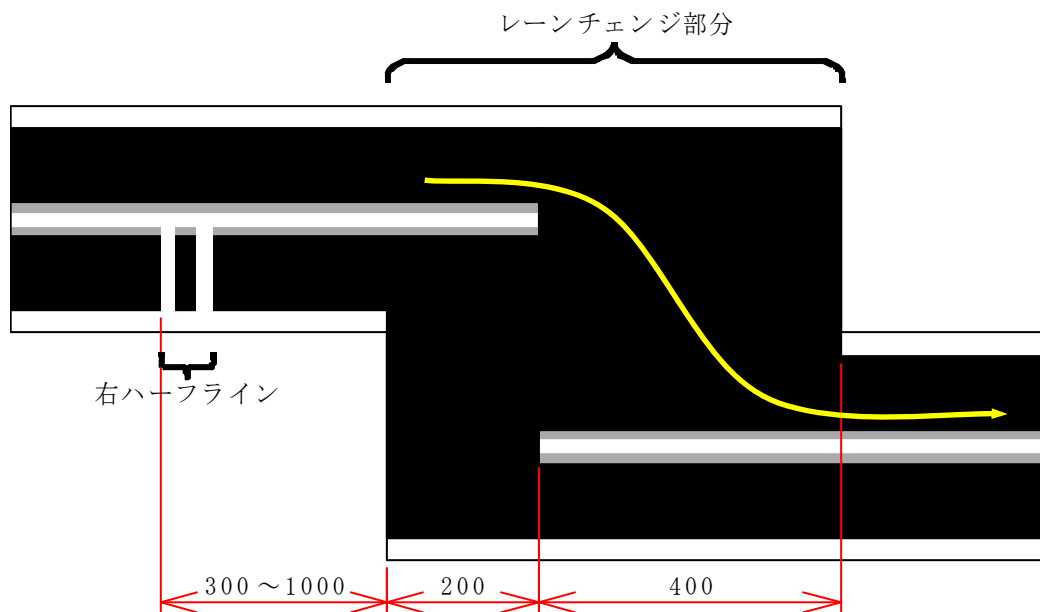
クランク手前の 50~100cm にはクロスラインと呼ばれる 2 本の白ラインが引かれています。マイコンカーはこのラインを検出すると、直角を曲がれるスピードまで減速します。直角を発見すると曲がり、通常走行に戻ります。



## 2.4 レーンチェンジ部分

レーンチェンジコースは、ジャパンマイコンカーラリー2007 大会(2006 年度)より追加されました。今までのコースは中心の白線が無くなることはありませんでしたが、初めて中心の白線が途切れるコースになっています。レーンチェンジは、プログラム技術の向上を目的として作られました。

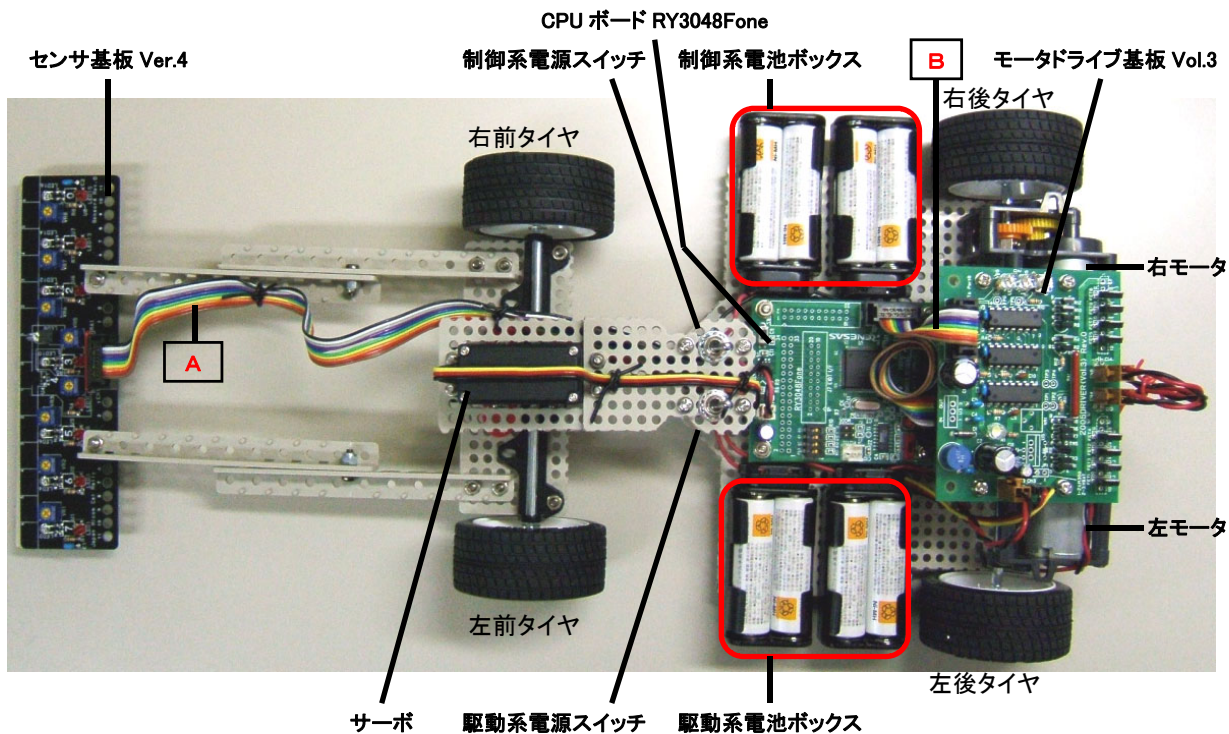
レーンチェンジは右へのレーンチェンジ、左へのレーンチェンジの 2 通りあります。右なら、レーンチェンジ部分 300mm~1000mm 手前にコース中心から右端まで白色2本のハーフラインがあり、それを発見すると右レーンチェンジと判断します。左レーンチェンジなら、コース中心から左端まで白色 2 本のハーフラインがあります。マイコンカーは中心線が無くなるとレーンチェンジを開始し、新しい中心線を見つけるとレーンチェンジ完了と判断し、通常走行に戻ります。下記に右レーンチェンジのコースを示します。



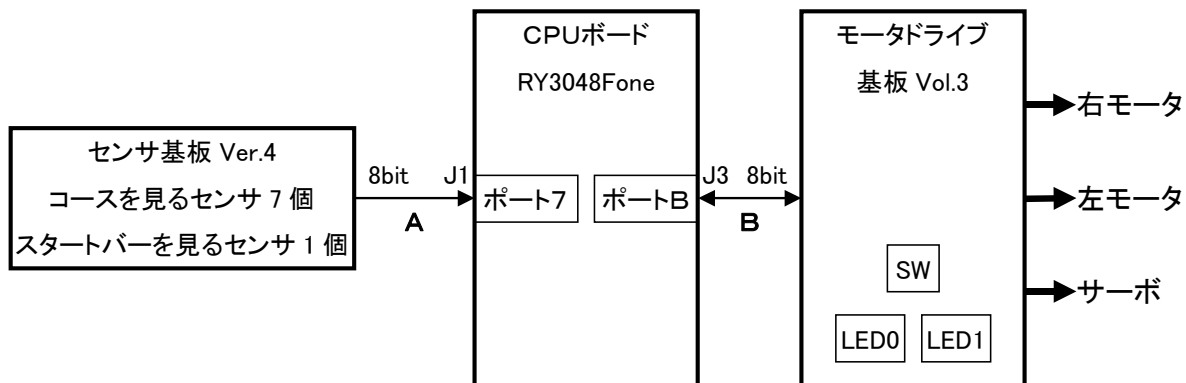


### 3. マイコンカーキットの仕様

#### 3.1 外観



マイコンカーキットは制御系のCPUボード、センサ基板(Ver.4)、モータドライブ基板(Vol.3)、駆動系の右モータ、左モータ、サーボで構成されています。

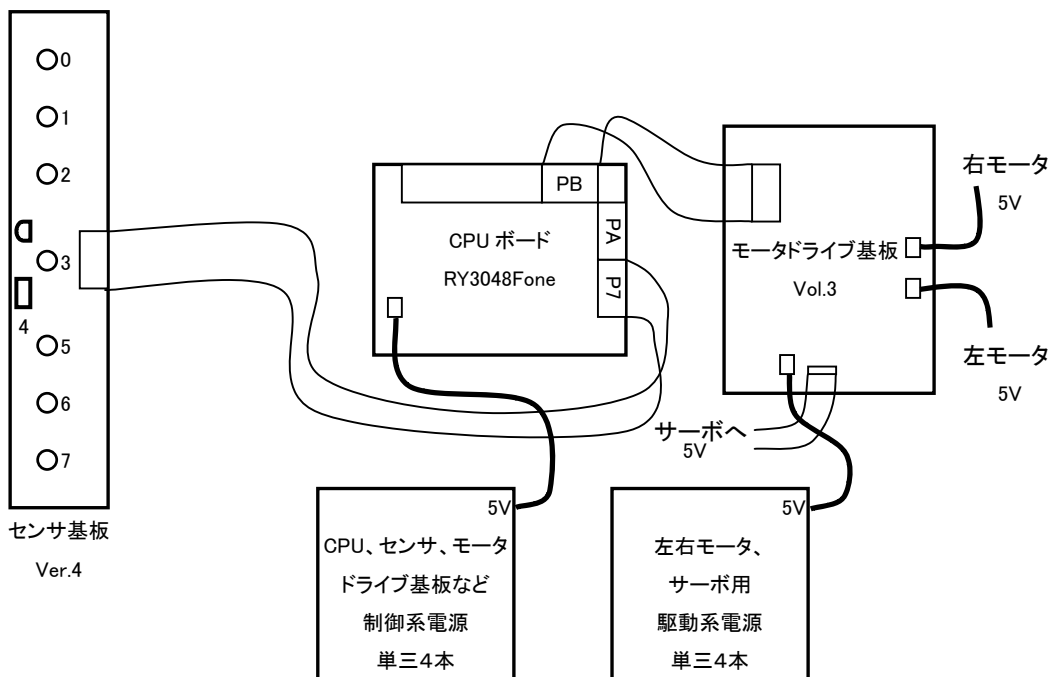


CPU ボード	<p>センサの状態をポート 7 から読み込み、左右モータの出力値、サーボの切れ角を計算して、ポート B に接続されているモータドライブ基板へ出力します。</p> <p>この、センサの状態を基にどのようにモータ、サーボの出力値を決めるか、これをプログラムすることになります。</p> <p>ポート A はキットでは使用していません。ロータリエンコーダ、EEP-ROM など、チューンナップするための機器を接続することができます。</p>
センサ基板 Ver.4	<p>コースの状態を検出するセンサが 7 個あります。センサの下部が白色なら"0"を出力、黒色なら"1"を出力します。</p> <p>スタートバーがあるかどうかを検出するセンサが 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。</p>
モータドライブ 基板 Vol.3	<p>CPU ボードからの弱電信号を、モータを動作させるための強電信号に変換します。サーボの駆動もモータ用電源を使用します。</p> <p>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。</p>
電池	<ul style="list-style-type: none"> <li>・制御系 (CPU) 電源 …単三 2 次電池 4 本(1.2V×4 本=4.8V)を使用</li> <li>・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×4 本=6.0V)を使用</li> </ul> <p><b>※CPU ボードの電圧は必ず、5.0V±10%の電圧にしてください。</b></p>

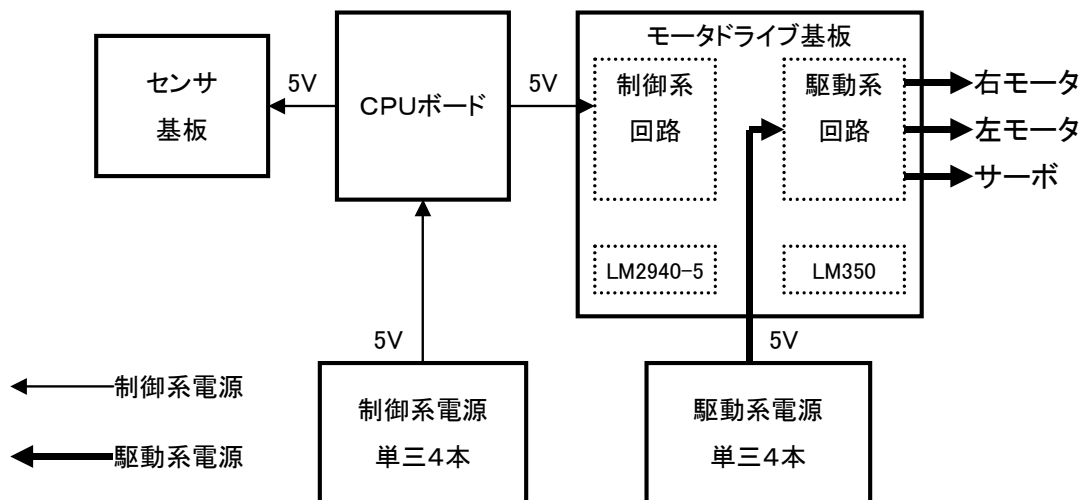
### 3.2 標準キットの電源構成

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CPU 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。

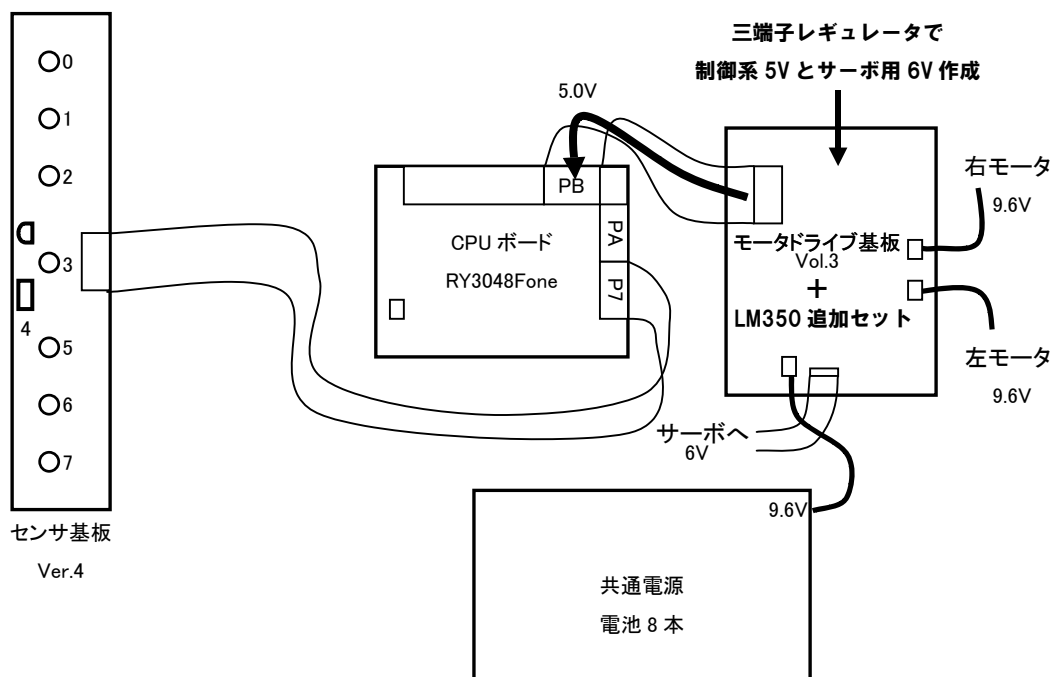


### 3.3 駆動系電圧を上げた電源構成

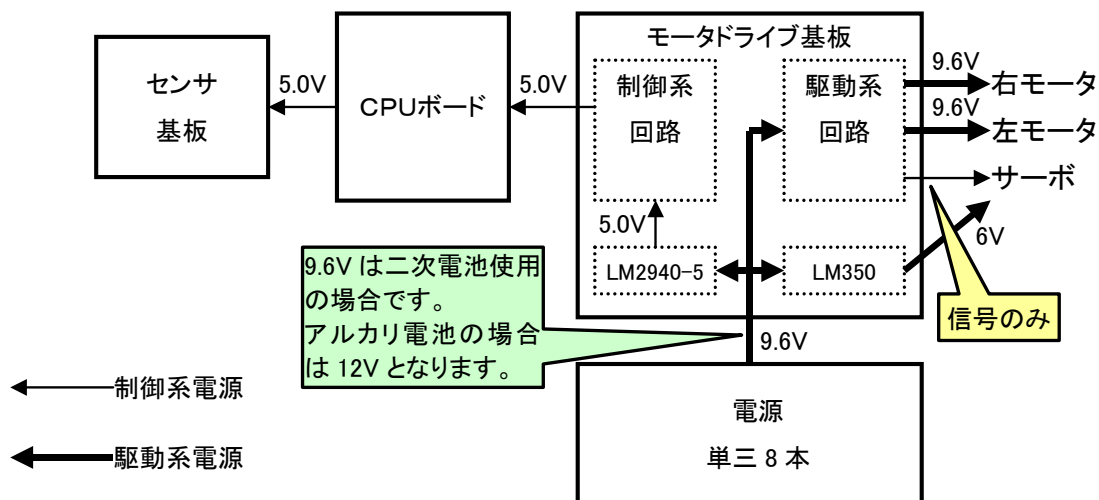
駆動系の電圧を上げれば(電池を増やせば)モータの回転数を上げることが可能です。モータ電源用に 6 本の電池を使えば 7.2V、8 本なら 9.6V となります。しかし、電池の使用本数は 8 本以内と決まっています。そこで、電池を制御系、駆動系共通にします。このとき、モータに 9.6V の電圧を加えても壊れません(定格は 6V なので好ましいことではありません)、CPU の動作保証電圧は 4.5~5.5V なので 5.5V を超えた電圧をかけると壊れてしまいます。サーボも同様に 6V 以上の電圧をかけられません。そのため、三端子レギュレータを取り付け CPU やサーボの電圧を定格にします。

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、4.5V 以下になると CPU がリセットしてしまいます。電池を共通化した場合、CPU のリセットに気をつけなければいけません。

「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 CPU 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、バネの押しが弱いので、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。

## 4. センサ基板

### 4.1 仕様

下記に、各センサ基板の仕様を示します。センサ基板 Ver.4 は、マイコンカーキット Ver.4 に付属のセンサ基板です。

名称	センサ基板	センサ基板 TLN119 版	センサ基板 Ver.4
略称	センサ基板	センサ基板 3	センサ基板 4
付属キット	初期マイコンカーキット	マイコンカーキット Vol.2 マイコンカーキット Vol.3	マイコンカーキット Ver.4
販売開始時期	1998 年頃 (販売終了)	2002 年 4 月頃 (2007/5 現在販売中)	2007 年 5 月 (2007/5 現在販売中)
基板枚数	1 枚	本体基板とサブ基板の 2枚	1 枚
コースを見る センサの個数	8 個	8 個	7 個
スタートバーを見る センサの個数	0 個	0 個	1 個
信号反転 回路	74HC04 による反転	74HC04 による反転	なし(プログラムで反転)
電圧	DC5.0V±10%	DC5.0V±10%	DC5.0V±10%
重量 (完成品の実測)		本体基板:約 20g サブ基板:約 10g	約 18g
レジスト (基板色)	なし(基板の地の色)	なし(基板の地の色)	黒色
基板寸法	W150×D50×厚さ 1.6mm	本体基板: W150×D33×厚さ 1.6mm サブ基板: W60×D37×厚さ 1.6mm	W140×D38×厚さ 0.8mm
寸法 (実測)	最大 W150×D50×H20mm	本体基板: 最大 W150×D33×H10mm サブ基板: 最大 W60×D37×H10mm	最大 W140×D38×H14mm

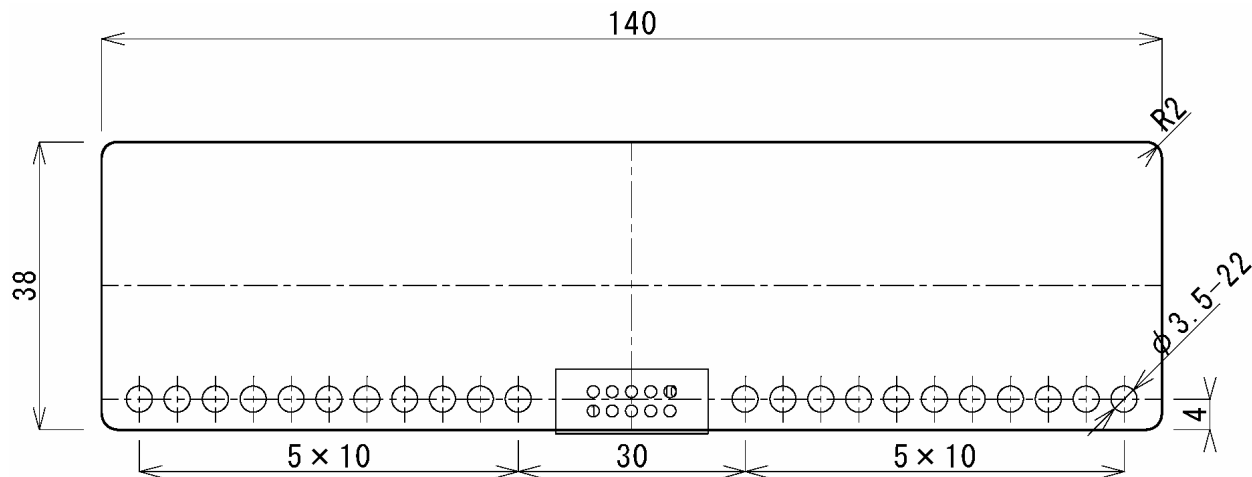
※重量は、リード線の長さや半田の量で変わります





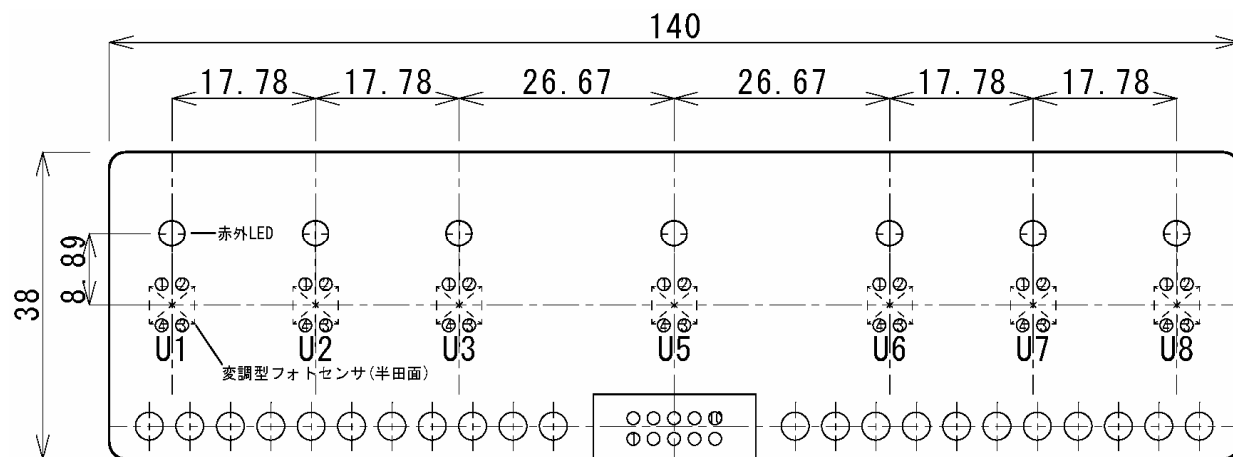
### 4.3 基板寸法

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。



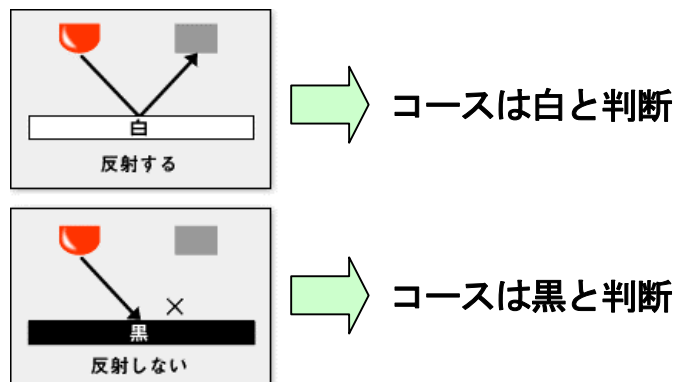
### 4.4 センサ取り付け寸法

コースを見るセンサは、7 個あります。基板の下記のような位置に取り付けています。

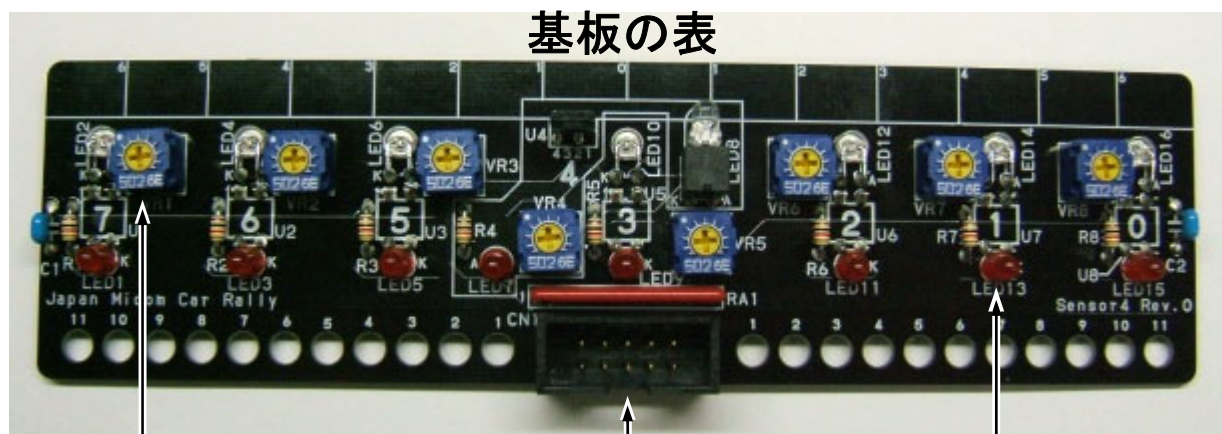


## 4.5 コースの白と黒を判断する仕組み

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7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。



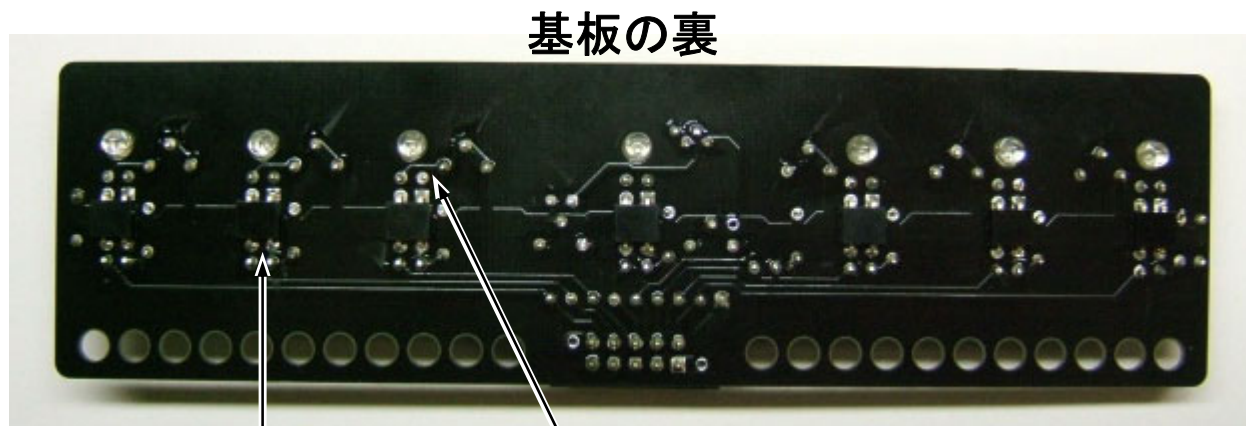
赤外線を出す量をボリュームで調整することができます。マイコンカーのコースには灰色があります。ボリュームの感度を変えることにより、灰色を”白”と判断させるか、”黒”と判断させるか調整することができます。**標準のプログラムは、灰色を白色と判断させると良いようになっています。**



センサ感度調整用ボリューム

センサ信号出力用コネクタ

センサ感度確認用 LED



変調型フォトセンサ

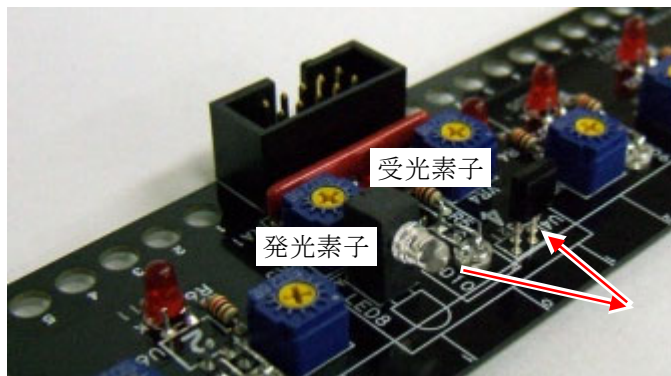
赤外 LED

内容	詳細
赤外 LED	TLN119 という素子を使用しています。この素子から赤外線を出します。赤外線なので人間の目には見えません。7 個あります。
変調型フォトセンサ	浜松フォトニクス(株)の S7136 という素子を使用しています。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。7 個あります。
センサ感度調整用ボリューム	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。標準のプログラムでは、“白”と判断させると良いようになっています。
センサ感度確認用 LED	LED 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この LED を確認しながら調整します。
センサ信号出力用コネクタ	センサの下部が白なら“0”(0V)、黒なら“1”(5V)の信号がこのコネクタから出力されます。

#### 4.6 スタートバーの開閉を判断する仕組み

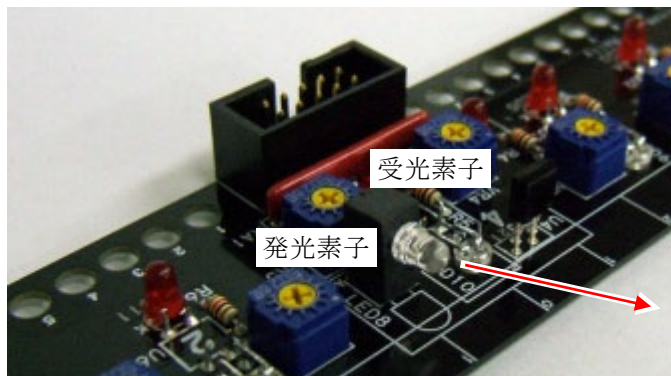
スタート時、白色のスタートバーが閉じています。赤外 LED と S6846(変調型フォトセンサ)を前方向に取り付けます。センサの状況によって下記のように判断できます。

●スタートバーが閉じているとき



反射あり→スタートバーあり

●スタートバーが開いているとき

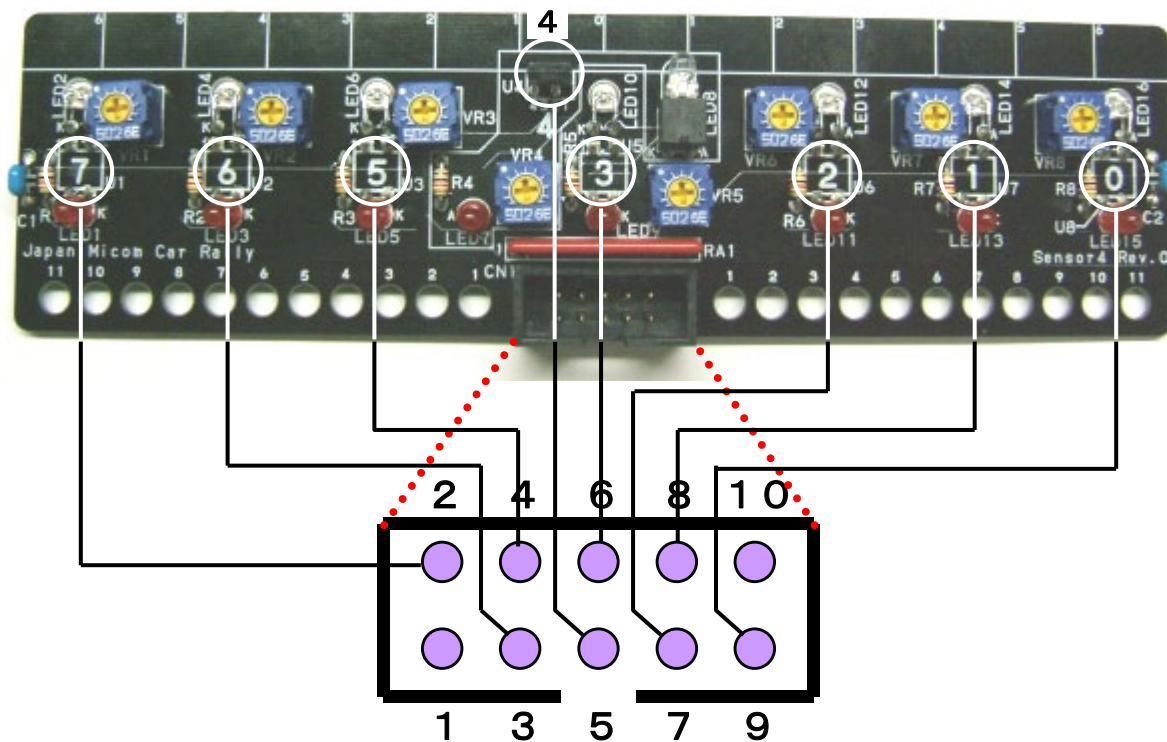


反射無し→スタートバーなし

発光素子が出す光の量は、ボリュームで調整することができます。

## 4.7 10ピンコネクタ

○で囲ったセンサの信号が、10ピンコネクタから出力されます。

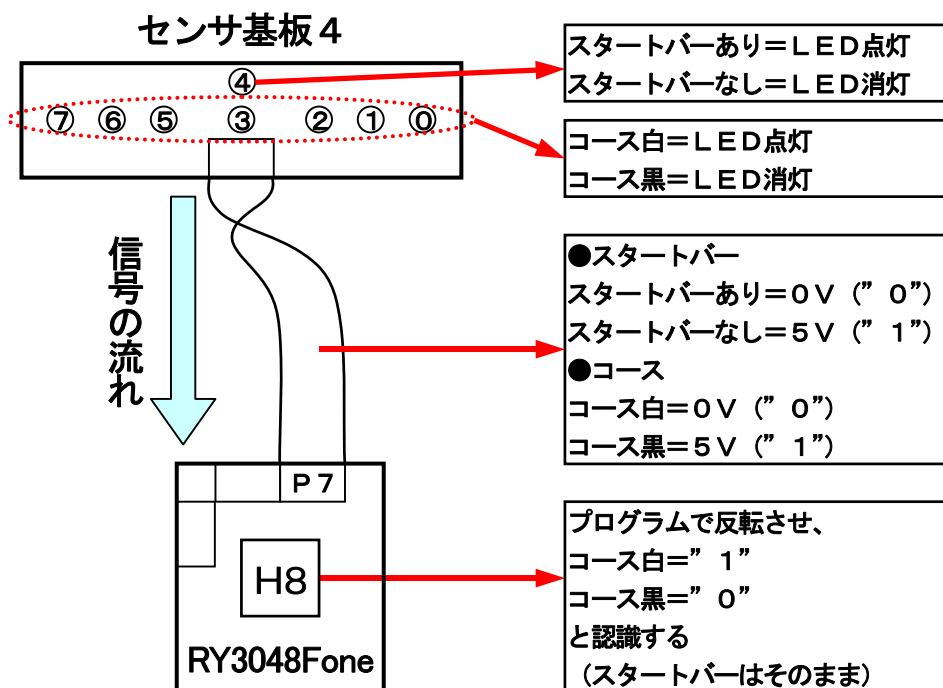


コネクタを上から見たところ

番号	方向	詳細	“0”(0V)	“1”(5V)
1	—	+5V		
2	OUT	7 センサ信号の出力 (左から1 番目)	白色	黒色
3	OUT	6 センサ信号の出力 (左から2 番目)	白色	黒色
4	OUT	5 センサ信号の出力 (左から3 番目)	白色	黒色
5	OUT	4 センサ信号の出力 (スタートバー)	バーあり	バーなし
6	OUT	3 センサ信号の出力 (中心)	白色	黒色
7	OUT	2 センサ信号の出力 (右から3 番目)	白色	黒色
8	OUT	1 センサ信号の出力 (右から2 番目)	白色	黒色
9	OUT	0 センサ信号の出力 (右から1 番目)	白色	黒色
10	—	GND		

## 4.8 信号の流れ

センサ基板から CPU ボードへの信号の流れは、下図のようになります。



例えば、コースセンサが左から「白黒黒 黒 白白黒」の状態、下記プログラムを実行したとします(スタートバーのセンサは無視します)。

```
unsigned char c;
...
c = ~P7DR;
```

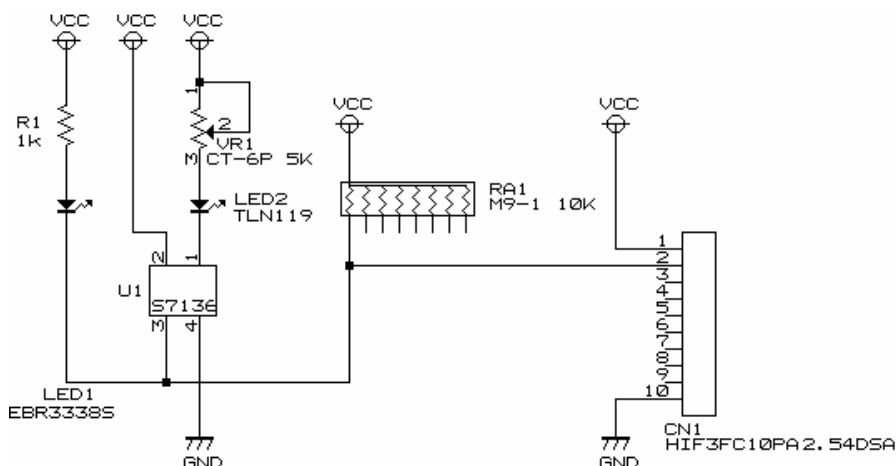
変数 c には下記の値が代入されます。

変数 c の値							
7	6	5	4	3	2	1	0
左から 1 番目のセンサ	左から 2 番目のセンサ	左から 3 番目のセンサ	スタートバーセンサ	中心センサ	右から 3 番目のセンサ	右から 2 番目のセンサ	右から 1 番目のセンサ
白	黒	黒		黒	白	白	黒
↓	↓	↓	↓	↓	↓	↓	↓
1	0	0	0	0	1	1	0

$c = (1000\ 0110)_2 = 0x86$

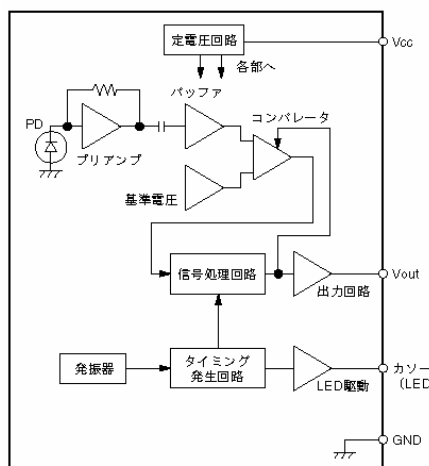
が代入されます。実際のプログラムでは、コースの状態とスタートバーの状態は、別々に判定します。

### 4.9 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 の明るさ調整はボリューム 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”でもない値はあり得ません。その為、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際は“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)

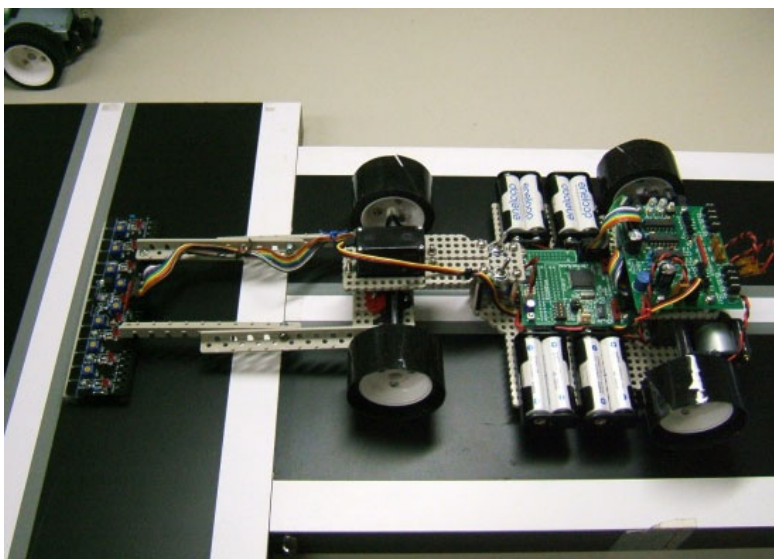


真理値表

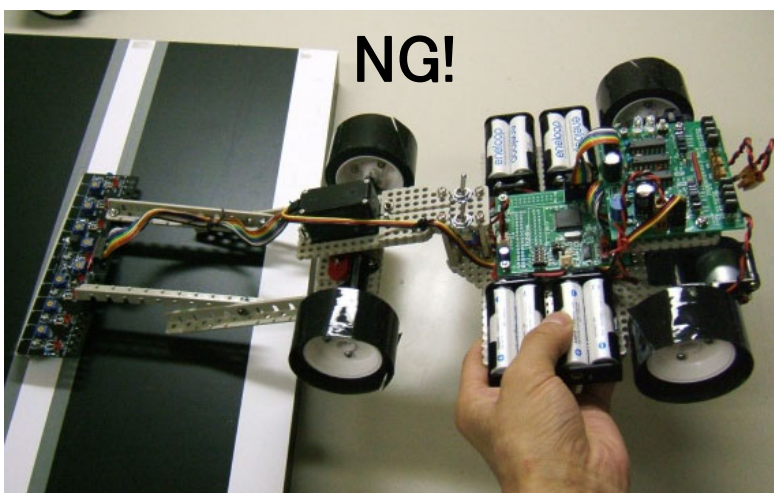
入力	出力レベル
光ON	LOW
光OFF	HIGH

- (a) 発振器・タイミング信号発生回路  
内蔵コンデンサを定電流で充放電することにより、基準発振出力を得ています。発振出力は、タイミング信号発生回路に入力され、LED駆動用パルス、デジタル信号処理用各種タイミングパルスを生成します。
- (b) LED駆動回路  
タイミング信号発生回路により生成されたLED駆動用パルスにより、発光ダイオードを駆動するための回路です。駆動デューティ比は、1/16です。
- (c) フォトダイオード、プリアンプ回路  
フォトダイオードはオンチップ型です。プリアンプ回路を通して、フォトダイオードの光電流を電圧に変換します。プリアンプ回路には、独自の交流増幅回路を使用しており、DCおよび低周波外乱光に対するダイナミックレンジを拡大するとともに、信号検出感度を高めています。
- (d) C結合・バッファアンプ・基準電圧発生回路  
C結合によって、さらに低周波外乱光を除去し、同時にプリアンプ部のDCオフセットを除去しています。バッファアンプでコンパレータレベルまで増幅し、基準電圧発生回路でコンパレータレベル信号を発生します。
- (e) コンパレータ回路  
コンパレータ回路にはヒステリシス機能が付加してあり、入力光の微小変動によるチャタリングを防止しています。
- (f) 信号処理回路  
信号処理回路は、ゲート回路とデジタル積分回路とで構成されています。ゲート回路は、同期検出時の検出入力のパルスを弁別する回路であり、非同期外乱光による誤動作を防止するものです。また、同期外乱光についてはゲート回路で除去できないため、後段のデジタル積分回路で除去しています。
- (g) 出力回路  
信号処理回路出力をバッファし、外部に出力する回路です。

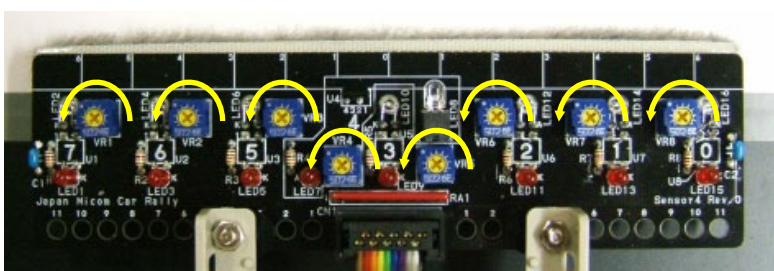
#### 4.10 センサの調整方法



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します。

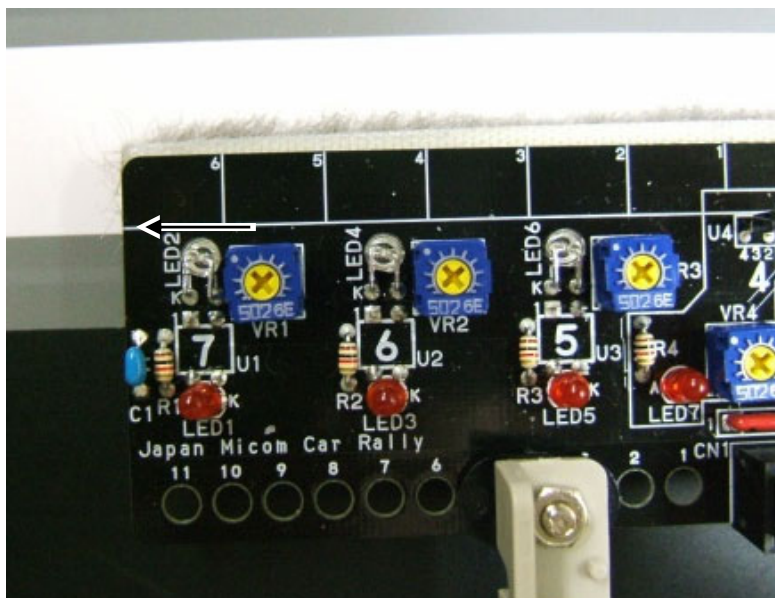


このように、手で持ちながらセンサの調整をしようとしてもセンサとコースとの間隔が一定にならないため、きちんと調整できません。**必ずコース同一面上に置いてください。**



8 個のボリュームをすべて、反時計回りに回します。

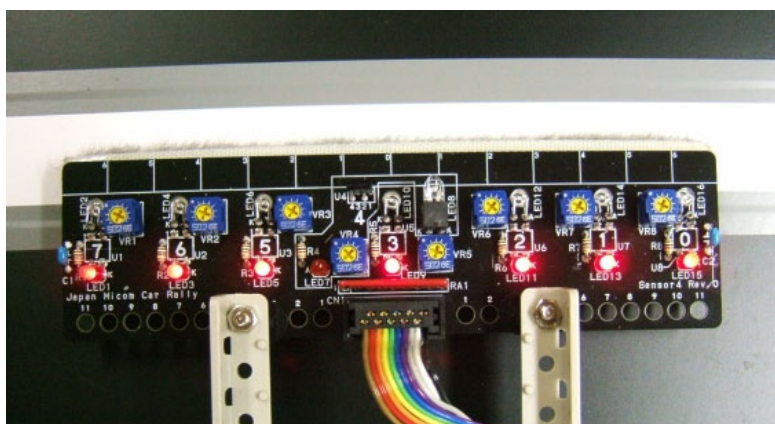




基板の横線とコースの白色と灰色の切り替わり部分を合わせます。真上から見て合わせるようにしてください。



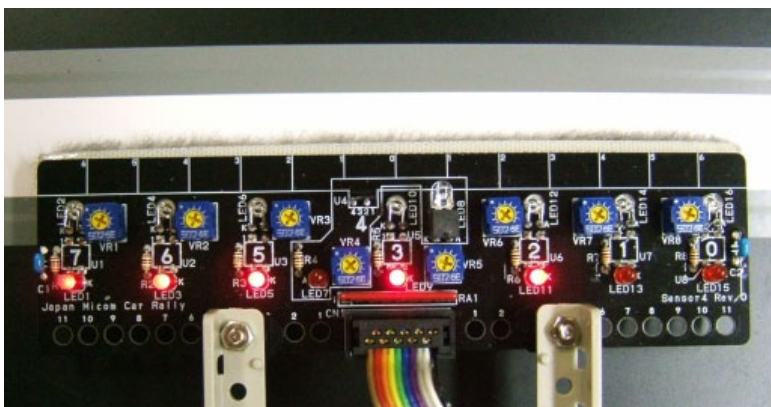
7 個のボリュームを時計回りに回して LED が点くようにします。一つ一つゆっくりと回して、LED が点いた瞬間回すのを止めます。今回の調整で灰色も反応するように調整します。マイコンカーキットは、白色・灰色で反応するよう調整します。



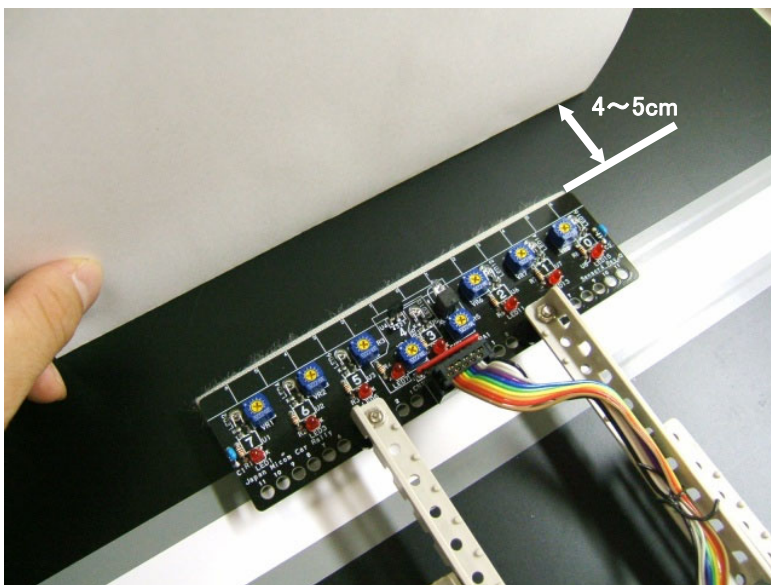
7 個の LED が点きました。LED7 はスタートバー用なのでここでは調整しません。



センサを少し下げます。すべて消えます。

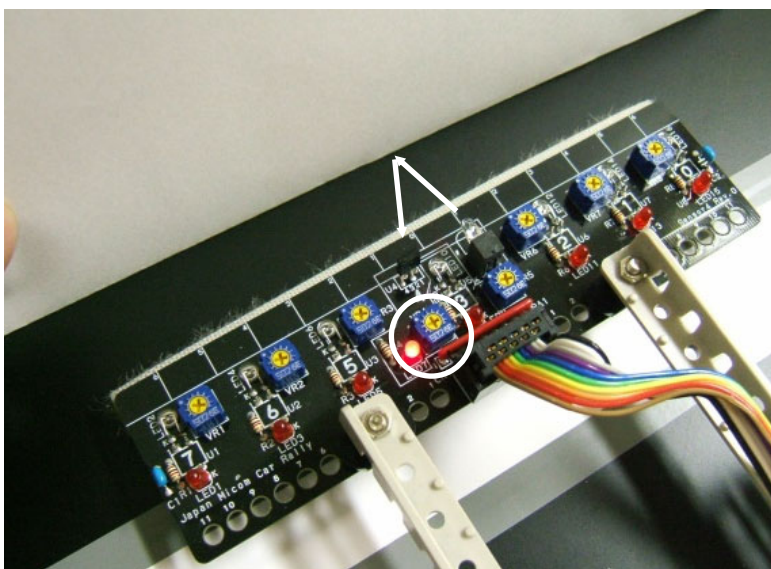


再度センサを灰色の位置に**ゆっくりと平行に近づけます**。他の LED より先に点く場合、感度を下げます(反時計回り)。点かない LED は感度を上げます(時計回り)。ほぼ同時に LED が点くように何度も調整します。



次に、スタートバーを検出するセンサの調整をします。

センサの先頭から **4~5cm 離れたところに白色の板か紙を立てておきます**。スタートバーの変わりです(ルールは 2~5cm 離してセットします)。



○の VR4 をゆっくりと時計回りに回し、LED7 が点く位置で止めます。このとき、センサ下のコースの色は関係ありません。

板や紙などを外したときに、LED が消えれば完了です。

## 5. モータドライブ基板

### 5.1 仕様

下記に、モータドライブ基板 Vol.1～3 の仕様をまとめます。マイコンカーキット Ver.4 で使用されているモータドライブ基板は Vol.3 です。

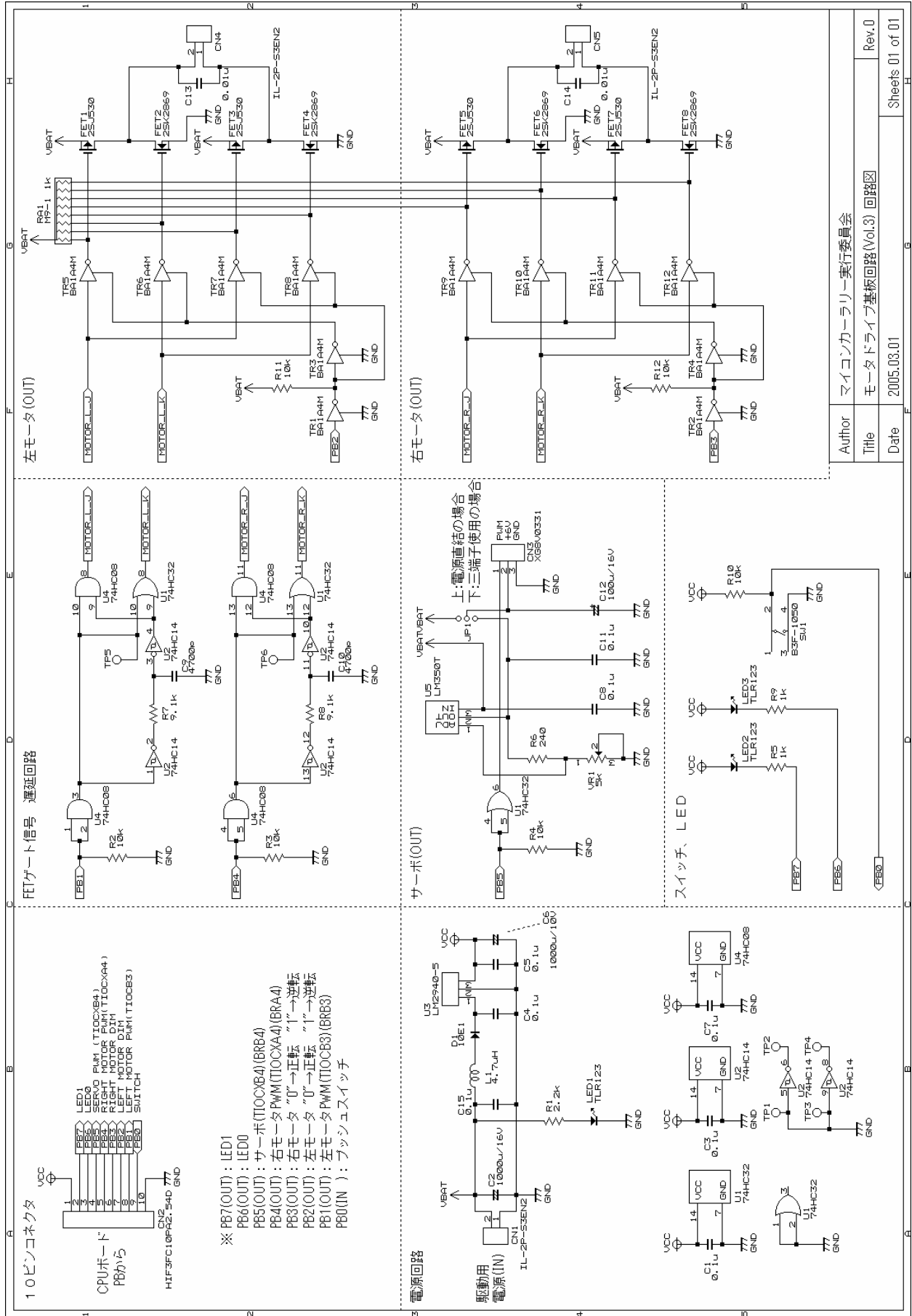
名称	モータドライブ 基板(Vol.1)	モータドライブ 基板(Vol.2)	モータドライブ 基板(Vol.3)
略称	ドライブ基板1	ドライブ基板2	ドライブ基板3
販売開始 時期	1998 年ごろ	2002 年 4 月	2005 年 4 月
モータの 動作	正転、逆転、ブレーキ	正転、フリー、ブレーキ	正転、逆転、ブレーキ
CPU ボード との接続	ポート B	ポート A	ポート B
PWM	リセット同期 PWM モード使用	1 チャンネルごとの PWM 使用	リセット同期 PWM モード使用
周期	モータ:16ms サーボ:16ms 個別設定不可	モータ:1ms サーボ:16ms 個別に設定可能	モータ:16ms サーボ:16ms 個別設定不可
使用する FET	4AM12×2 個	4AM12×1 個	2SJ タイプ×4 個+2SK タイプ×4 個 または 4AM12×2 個
制御系 電圧	DC5.0V±10%	DC5.0V±10%	DC5.0V±10%
駆動系 電圧	5V	5～15V	5～15V
駆動系電圧 6V 以上のとき、 サーボは…	回路的にできない	6V 一定にする回路を外付け する必要有り	基板にパターン有り LM350 追加セットの部品を 取り付け
標準 プログラム	tmc4.c	kit2.c または kit04.c	kit05.c または kit06.c または kit07.c
寸法	最大 W76×D49×H15mm (実測)	最大 W80×D50×H15mm (実測)	最大 W80×D65×H20mm (実測)
その他	販売終了	販売終了	販売中 (2007.05 現在)

ドライブ基板1ではリセット同期 PWM モードというモードでモータ、サーボを制御していました。これはプログラムが比較的簡単なのですが、右モータ、左モータ、サーボに加える PWM 周期を同じにしかできないという制限がありました。サーボには 16[ms]の周期を加えることと規格で決まっており、モータに加える PWM も 16[ms]となります。しかし、モータ制御に周期 16[ms]では間隔が長すぎ、モータがガクガクした動きとなってしまいました。

ドライブ基板2では、1チャンネルごとの PWM を使用して、右モータ、左モータの周期を 1[ms]、サーボの周期を 16[ms]と、それぞれ独立して周期を設定できるようにしました。モータの制御は滑らかになりましたが、プログラムが複雑になってしまい非常に理解しづらくなってしまいました。

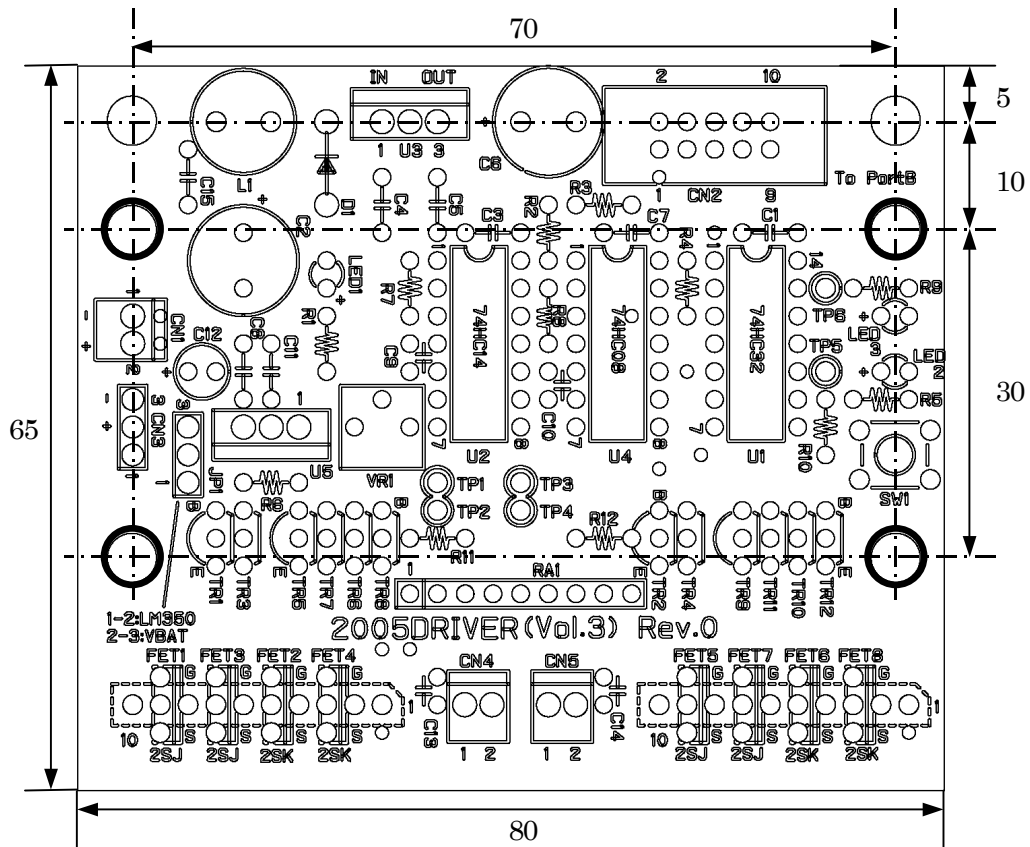
そこで、ドライブ基板3では、リセット同期 PWM モードに戻しました。ドライブ基板1の説明のとおり、モータがガクガクしてしまいます。しかし、サーボの周期は規格では 16[ms]ですが、実験で周期を短くしても動作することが分かりました。そこで、どうしてもガクガクが気になる場合は、サーボが動作する範囲内で周期を短くして対応します。

5.2 回路図



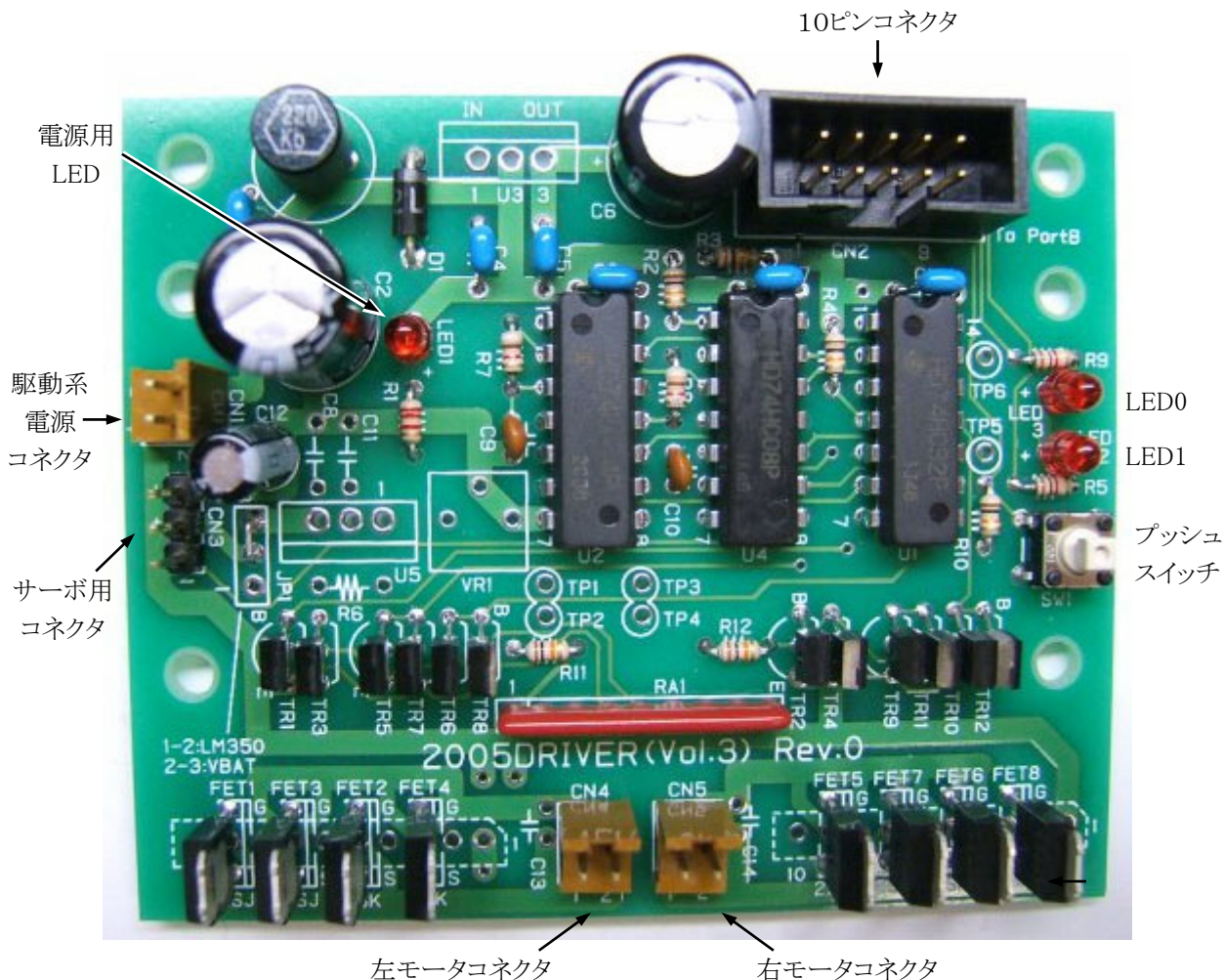
### 5.3 寸法

基板取り付け用の穴として、6つあります。キットでは、太い○の4つの穴を使用してキットと基板を固定します。キットへの取り付けは、ドライブ基板2と同じです。



## 5.4 機能

ドライブ基板3は、部品を実装すると下記写真のようになります。



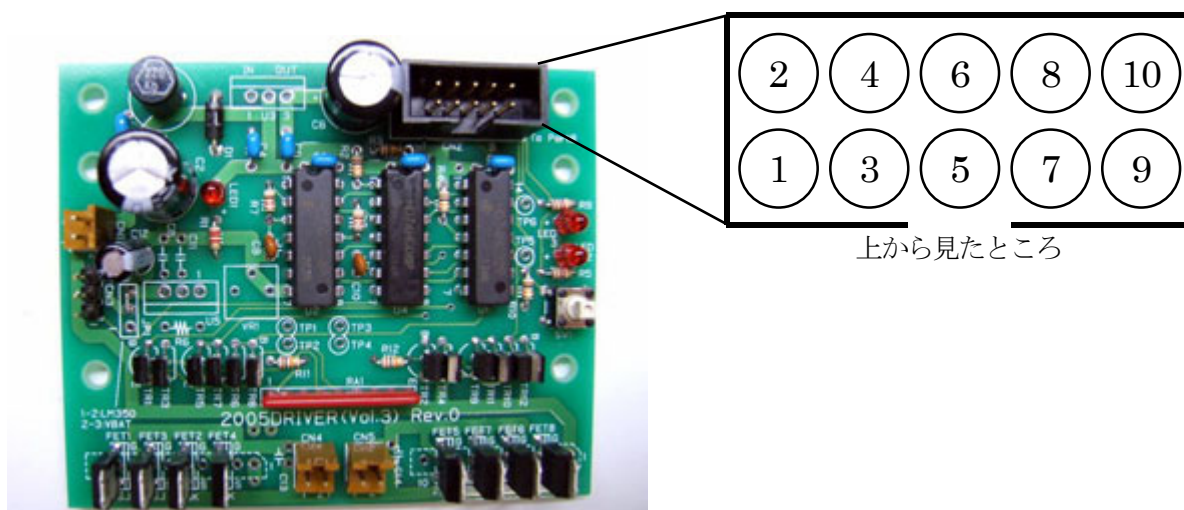
10ピンコネクタ	フラットケーブルで CPU ボードと接続します。ポート B(J3)のコネクタに接続します。
駆動系電源コネクタ	モータとサーボに供給する電源です。74HC08、74HC14、74HC32 などの制御系部品は、10ピンコネクタから供給される5Vで動作します。標準キットでは入力電圧6Vまでに対応していますが、それ以上の電圧にするときは、 <b>サーボに加える電圧を6V一定にする必要があります</b> 。LM350 追加セットの部品を追加すると、サーボ電圧を一定にすることができます。
右モータコネクタ	右モータと接続します。
左モータコネクタ	左モータと接続します。
サーボ用コネクタ	サーボと接続します。3ピンで信号の順番が、「1:サーボ信号、2:+電源、3:GND」となっています。この順番でないメーカーのサーボは、サーボ側のピンを入れ替える必要があります。
電源用 LED	電源コネクタに電圧が供給されていると光ります。

LED0	10 ピンコネクタに接続した CPU ボードのポート B の bit6 と接続されています。この bit を出力用に設定して、LED0 を点灯／消灯させます。
LED1	10 ピンコネクタに接続した CPU ボードのポート B の bit7 と接続されています。この bit を出力用に設定して、LED0 を点灯／消灯させます。
プッシュスイッチ	10 ピンコネクタに接続した CPU ボードのポート B の bit0 と接続されています。この bit を入力用に設定して、状態を読み込むことによりスイッチが押されているかどうかチェックします。

## 5.5 コネクタ

### 5.5.1 10 ピンコネクタ

フラットケーブルで CPU ボードと接続します。



上から見たところ

ピン番	信号、方向※	詳細	“0”	“1”	備考
1	—	+5V			
2	基板←PB7	LED1	点灯	消灯	
3	基板←PB6	LED0	点灯	消灯	
4	基板←PB5	サーボ信号	PWM 信号		PWM 信号 ITU4_BRB でデューティ比設定
5	基板←PB4	右モータ PWM	停止	動作	PWM 信号 ITU4_BRA でデューティ比設定
6	基板←PB3	右モータ回転方向	正転	逆転	
7	基板←PB2	左モータ回転方向	正転	逆転	
8	基板←PB1	左モータ PWM	停止	動作	PWM 信号 ITU3_BRB でデューティ比設定
9	基板→PB0	プッシュスイッチ	押された	押されていない	
10	—	GND			

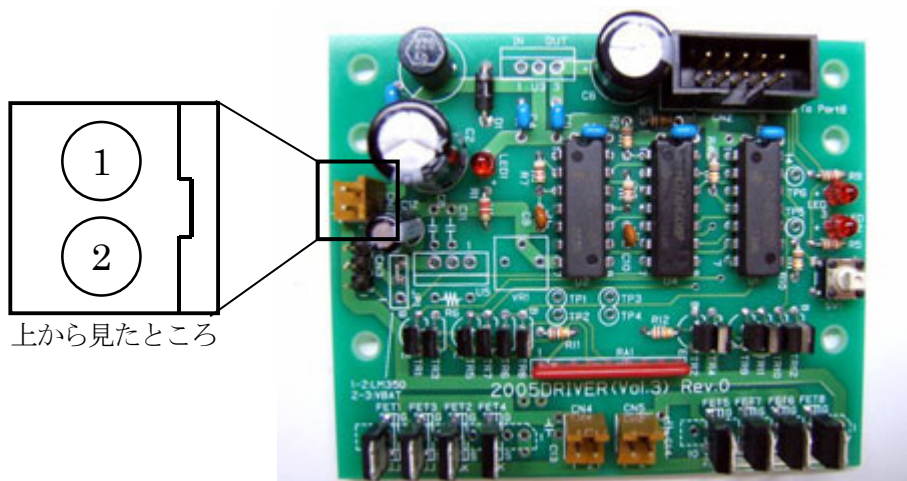
※PB7 とは、H8 マイコンのポート B の bit7 の意味です。

「基板→PBO」 は、ドライブ基板からの出力信号をマイコンのポートで読み込みます (ポートは入力)。

「基板←PBO」 は、マイコンからの出力信号をドライブ基板が入力し動作します。

### 5.5.2 駆動系電源コネクタ

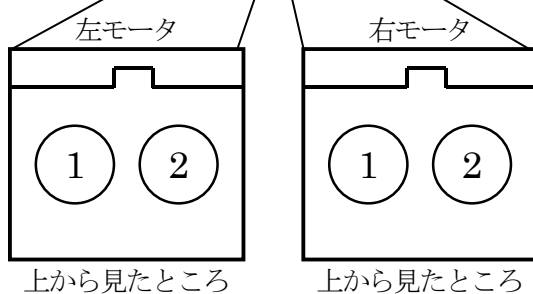
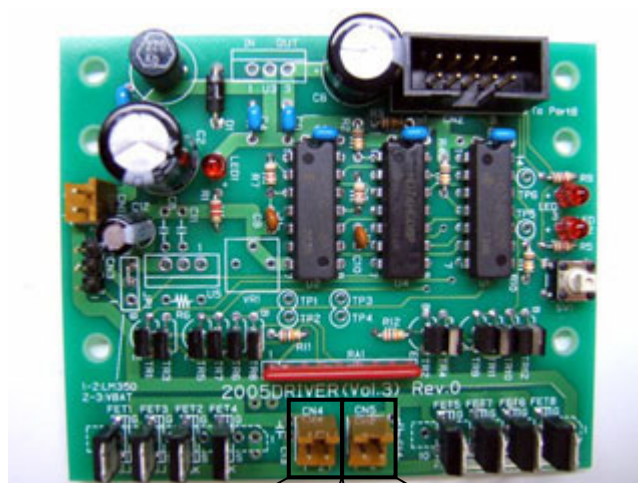
モータ、サーボ用の電源と接続します。



番号	方向	詳細
1	—	GND
2	IN	電源入力 5~15V

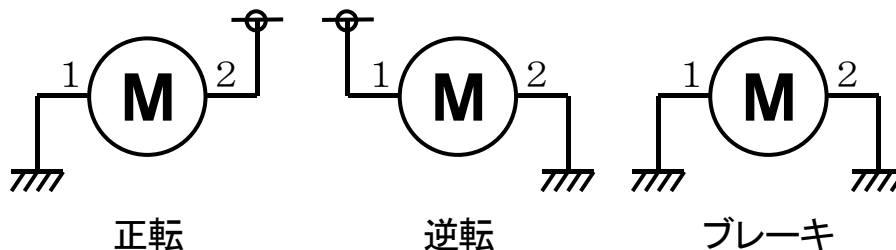
### 5.5.3 モータコネクタ

モータと接続します。



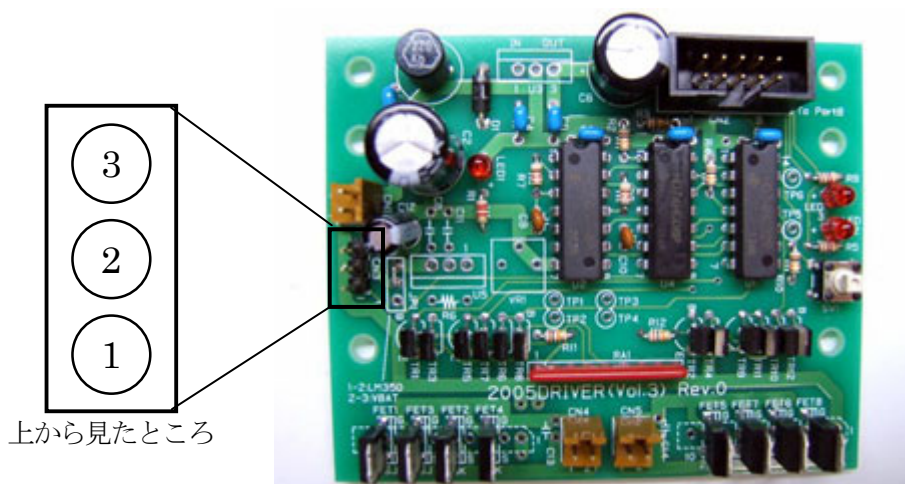
番号	方向	正転	逆転	ブレーキ
1	OUT	0V	電源電圧	0V
2	OUT	電源電圧	0V	0V





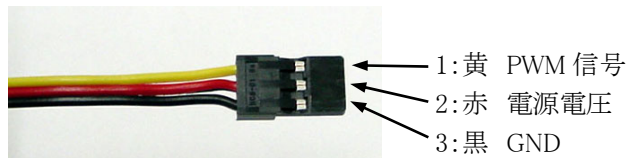
### 5.5.4 サーボコネクタ

サーボと接続します。サーボ側のコネクタが表のようなピン割り当てになっていればそのまま接続できますが、なっていない場合はピンを入れ替えます。一般的なサーボは、黒色が GND、赤色が電源、白または黄色が PWM 信号用となっています。



番号	方向	詳細
1	OUT	PWM 信号出力
2	OUT	電源電圧 ※下記参照
3	OUT	GND

サーボ側コネクタの例



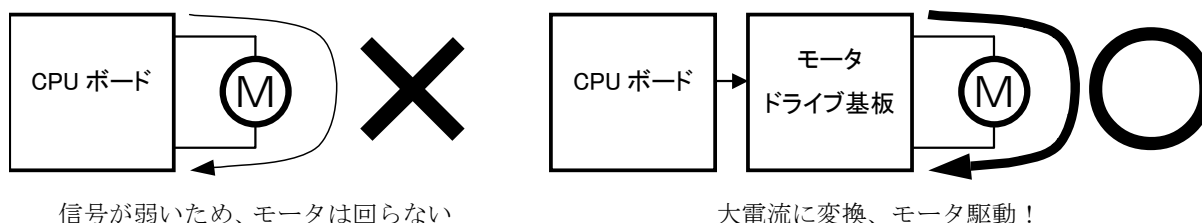
※2 ピンの電源は、JP1 で切り替えます。

JP1	詳細
1-2 間ショート	駆動系電源電圧が 6V 以上ならこちらを選択 LM350 を通して 6V 一定にした電圧が出力される (LM350 追加セットの部品を追加する必要有り)
2-3 間ショート	駆動系電源電圧が 6V 以下ならこちらを選択 駆動系電源と直結される

## 5.6 モータ制御

### 5.6.1 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、止めろ」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百～数千 mA という大きな電流が流せる信号に変換します。

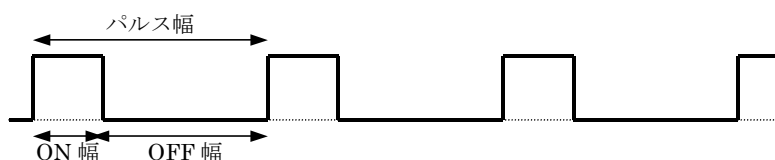


### 5.6.2 スピード制御の原理

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、非常に大きな抵抗が必要です。また、モータに加えなかった分は、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」と呼び、英語では「Pulse Width Modulation」となります。略して **PWM 制御** 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。



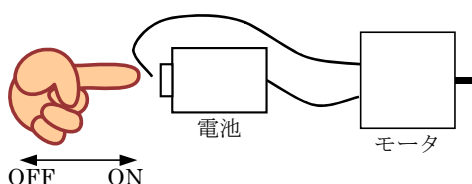
デューティ比 = ON 幅 / パルス幅 (ON 幅 + OFF 幅)

です。例えば、100ms のパルスに対して、ON 幅が 60ms なら、

デューティ比 = 60ms / 100ms = 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。

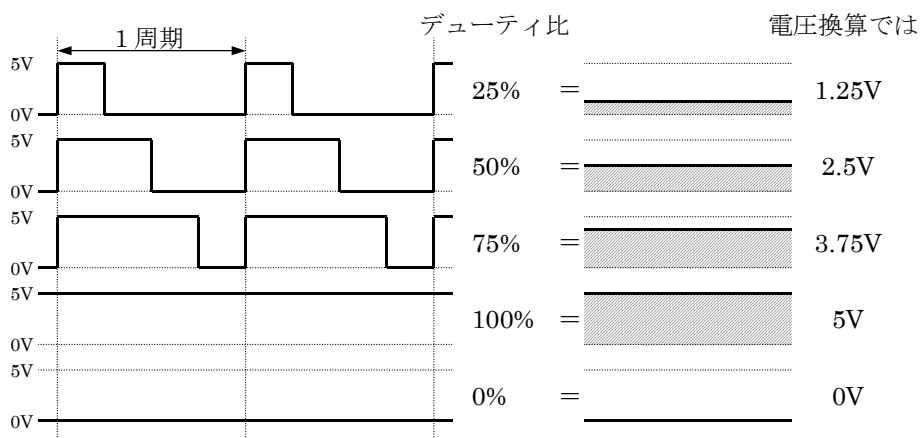
「PWM」と聞くと、何か難しく感じてしまいますが、下記のように手でモータと電池の線を「繋ぐ」、「離す」の繰り返し、それも PWM と言えます。繋いでいる時間が長いとモータは速く回ります。離している時間が長いとモータは少ししか回りません。人なら「繋ぐ」、「離す」動作をコンマ数秒でしか行えませんがマイコンなら数ミリ秒で行えます。



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。CPU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。

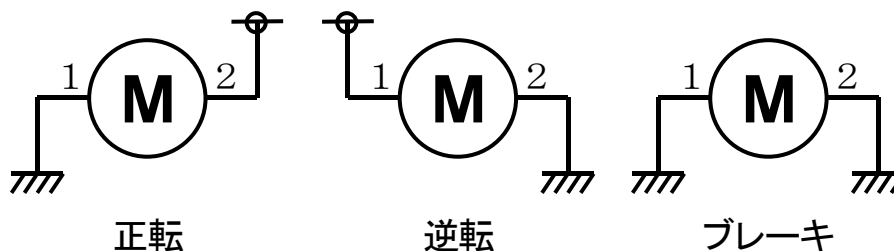


なぜ電圧制御ではなくパルス幅制御でモータのスピードを制御するのでしょうか。CPUは"0"か"1"かのデジタル値の取り扱いは大変得意ですが、何Vというアナログ的な値は不得意です。そのため、"0"と"1"の幅を変えて、あたかも電圧制御しているように振る舞います。これがPWM制御です。

### 5.6.3 正転、逆転、ブレーキの原理

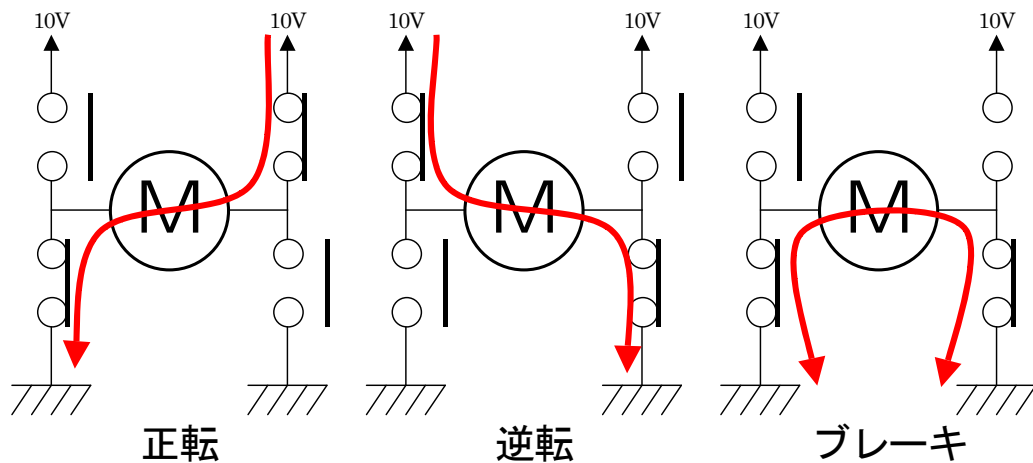
モータドライブ基板(Vol.3)では、モータを「正転、逆転、ブレーキ」制御することができます。これは、モータの端子に加える電圧を下表のように変えることにより、制御しています。

動作	端子1	端子2
正転	GND接続	+接続
逆転	+接続	GND接続
ブレーキ	GND接続	GND接続



### 5.6.4 Hブリッジ回路

では、実際はどのようにするのでしょようか。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。

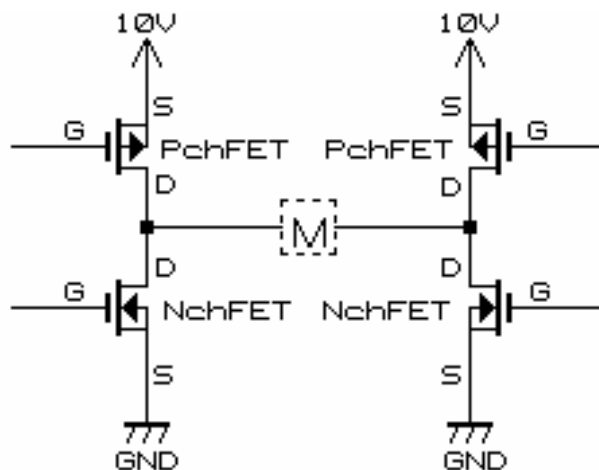


### 5.6.5 Hブリッジ回路のスイッチをFETにする

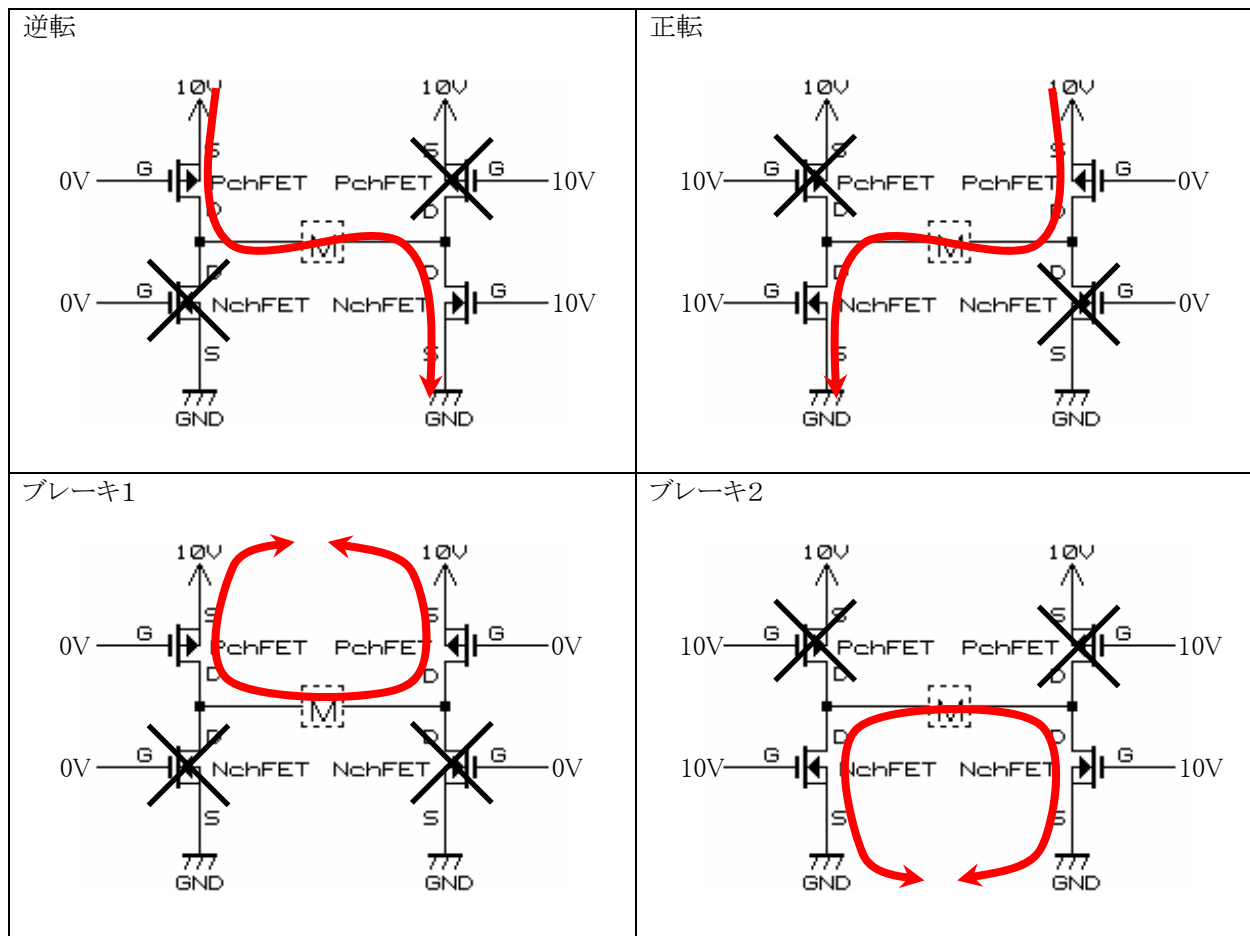
スイッチ部分を FET にします。電源のプラス側に P チャネル FET (2SJ タイプ)、マイナス側に N チャネル FET (2SK タイプ) を使用します。

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。

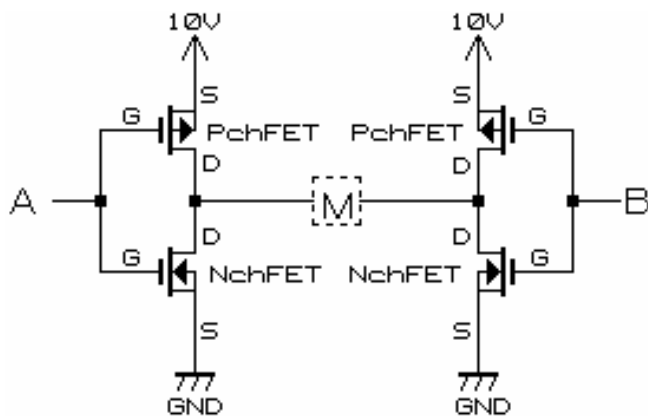


これら 4 つの FET のゲートに加える電圧を変えることにより、正転、逆転、ブレーキの動作を行います。



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 させてはいけません。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 が燃えるかパターンが燃えるか…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。



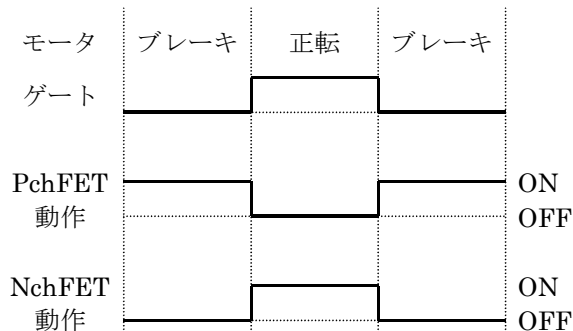
A	B	動作
0V	0V	ブレーキ
0V	10V	逆転
10V	0V	正転
0V	0V	ブレーキ

※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”の制御信号とは異なるので注意しましょう。

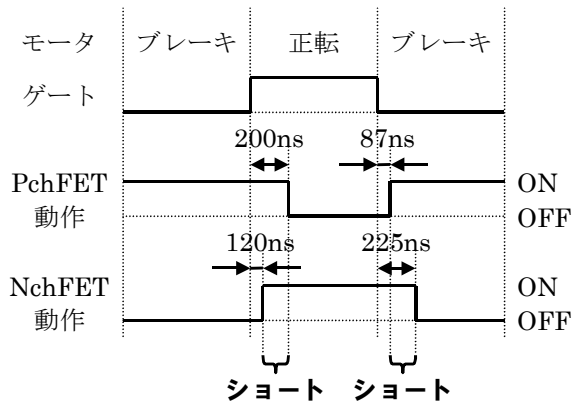
この回路を実際に組んで PWM 波形を加え動作させると、FET が非常に熱くなりました。どうしてでしょうか。FET のゲートから信号を入力し、ドレイン・ソース間が ON/OFF するとき、事項の左図「理想的な波形」のように、P チャネル FET と N チャネル FET がすぐに反応してブレーキと正転がスムーズに切り替わりるように思えま

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、下の右図「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になり初めてから実際に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になり初めてから実際に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。

参考までにFETの電気的特性を下記に示します。

2SJ530(Pチャンネル)

電気的特性						
(Ta=25°C)						
項目	記号	Min	Typ	Max	単位	測定条件
ドレイン・ソース破壊電圧	$V_{BRDSS}$	-60	—	—	V	$I_D = 10mA, V_{GS} = 0$
ゲート・ソース破壊電圧	$V_{BRSSS}$	$\pm 20$	—	—	V	$I_G = \pm 100\mu A, V_{DS} = 0$
ドレイン遮断電流	$I_{DSS}$	—	—	-10	$\mu A$	$V_{GS} = -60V, V_{DS} = 0$
ゲート遮断電流	$I_{GSS}$	—	—	$\pm 10$	$\mu A$	$V_{DS} = \pm 16V, V_{GS} = 0$
ゲート・ソース遮断電圧	$V_{GSOFF}$	-1.0	—	-2.0	V	$V_{DS} = 10V, I_D = 1mA$
順伝達アドミタンス	$ y_{fs} $	6.5	11	—	S	$I_D = 8A, V_{GS} = 10V^{24}$
ドレイン・ソースオン抵抗	$R_{DS(on)}$	—	0.08	0.10	$\Omega$	$I_D = 8A, V_{GS} = 10V^{24}$
ドレイン・ソースオン抵抗	$R_{S(on)}$	—	0.11	0.16	$\Omega$	$I_D = 8A, V_{GS} = 4V^{24}$
入力容量	$C_{iss}$	—	850	—	pF	$V_{DS} = 10V, V_{GS} = 0$
出力容量	$C_{oss}$	—	420	—	pF	$f = 1MHz$
掃蕩容量	$C_{rss}$	—	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	$I_F = 15A, V_{GS} = 0$ $dI_F/dt = 50A/\mu s$

注) 4. パルス測定

OFF→ONは  
87ns 遅れる

ON→OFFは  
200ns 遅れる

2SK2869(Nチャンネル)

電気的特性						
(Ta=25°C)						
項目	記号	Min	Typ	Max	単位	測定条件
ドレイン・ソース破壊電圧	$V_{BRDSS}$	60	—	—	V	$I_D = 10mA, V_{GS} = 0$
ゲート・ソース破壊電圧	$V_{BRSSS}$	$\pm 20$	—	—	V	$I_G = \pm 100\mu A, V_{DS} = 0$
ドレイン遮断電流	$I_{DSS}$	—	—	10	$\mu A$	$V_{GS} = 60V, V_{DS} = 0$
ゲート遮断電流	$I_{GSS}$	—	—	$\pm 10$	$\mu A$	$V_{DS} = \pm 16V, V_{GS} = 0$
ゲート・ソース遮断電圧	$V_{GSOFF}$	1.5	—	2.5	V	$V_{DS} = 10V, I_D = 1mA$
順伝達アドミタンス	$ y_{fs} $	10	16	—	S	$I_D = 10A, V_{DS} = 10V^{*1}$
ドレイン・ソースオン抵抗	$R_{DS(on)}$	—	0.033	0.045	$\Omega$	$I_D = 10A, V_{GS} = 10V^{*1}$
ドレイン・ソースオン抵抗	$R_{S(on)}$	—	0.055	0.07	$\Omega$	$I_D = 10A, V_{GS} = 4V^{*1}$
入力容量	$C_{iss}$	—	740	—	pF	$V_{DS} = 10V, V_{GS} = 0$
出力容量	$C_{oss}$	—	380	—	pF	$f = 1MHz$
掃蕩容量	$C_{rss}$	—	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	$I_F = 20A, V_{GS} = 0$ $dI_F/dt = 50A/\mu s$

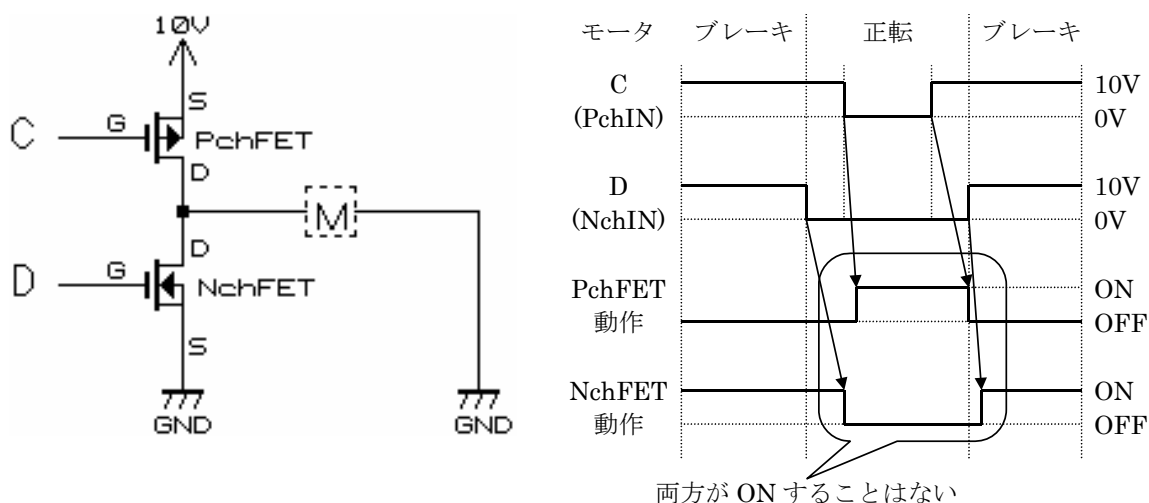
注) 1. パルス測定

OFF→ONは  
120ns 遅れる

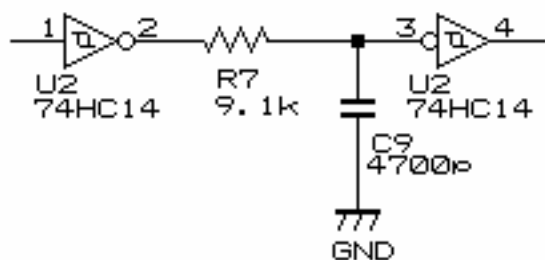
ON→OFFは  
225ns 遅れる

### 5.6.6 PチャンネルとNチャンネルの短絡防止回路

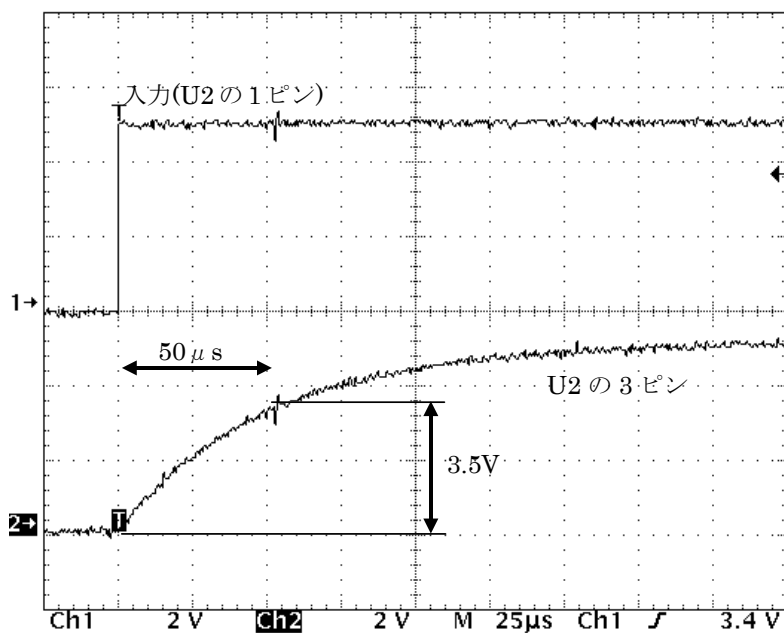
解決策としては、先ほどの回路図にある A 側の P チャンネル FET と N チャンネル FET を同時に ON、OFF するのではなく、少し時間をずらしてショートさせないようにします。



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、そちらを参照して下さい。

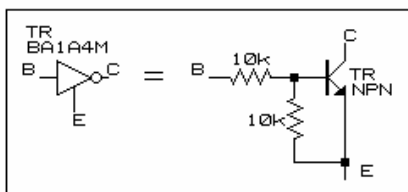
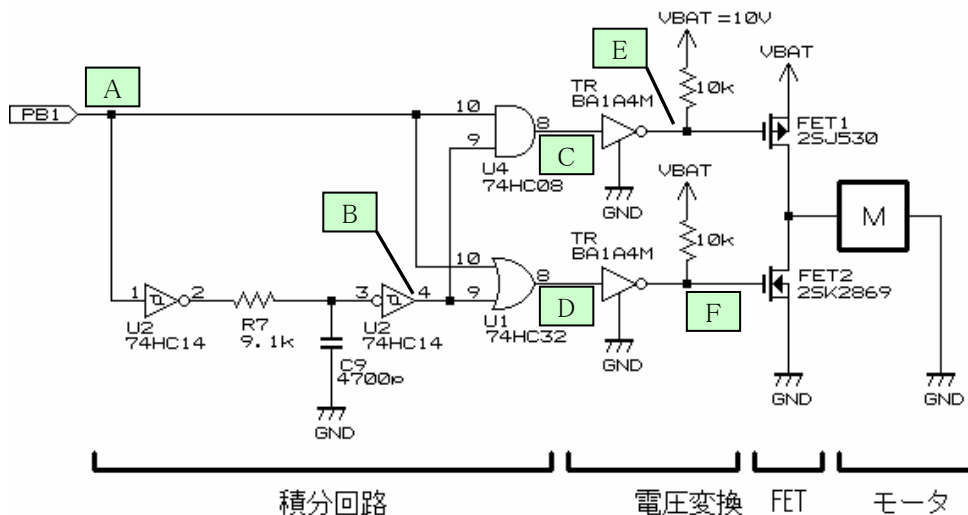


遅延時間はだいたい  
 時定数  $T = CR$  [s]  
 で計算することができます。  
 今回は  $9.1k\Omega$ 、 $4700pF$  なので、計算すると  
 $T = 9.1 \times 10^3 \times 4700 \times 10^{-12}$   
 $= 42.77 [\mu s]$   
 となります。

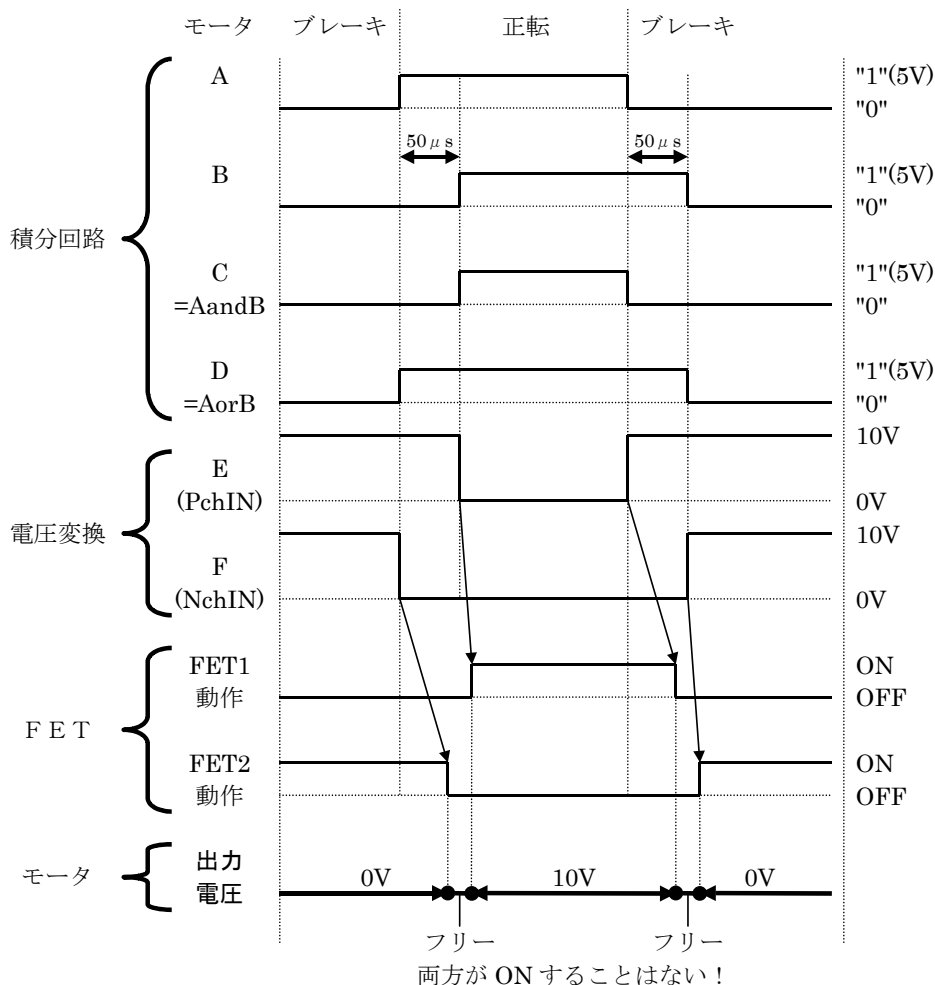


74HC シリーズは 3.5V 以上の入力電圧があると“1”とみなします。実際に波形を観測し、3.5V になるまでの時間を計ると約  $50\mu s$  になりました。  
 先ほどの「実際の波形」の図では最高でも 225ns のずれしかありませんが、積分回路では  $50\mu s$  もの遅延時間を作っています。これは、FET 以外にも、電圧変換用のデジタルトランジスタの遅延時間、FET のゲートのコンデンサ成分による遅れなどを含めたためです。

積分回路とFETを合わせた回路は下記のようになります。



デジタルトランジスタで  
 入力0V→出力10V(オープンコレクタ)  
 入力5V→出力0V  
 に変換します





(a) ブレーキ→正転に変えるとき

1. ポートからの信号は“0”でブレーキ、“1”で正転です。ブレーキから正転へ変えます (A点)。
2. 積分回路により  $50\ \mu\text{s}$  遅れた波形がB点より出力されます。
3. C点は、AandB の波形が出力されます。
4. D点は、AorB の波形が出力されます。
5. E点は、デジタルトランジスタで電圧変換された信号が出力されます。C点の  $0\text{V}-5\text{V}$  信号が、 $10\text{V}-0\text{V}$  信号へと変換されます。
6. F点も同様にD点の  $0\text{V}-5\text{V}$  信号が、 $10\text{V}-0\text{V}$  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  $10\text{V}\rightarrow 0\text{V}$  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  $50\ \mu\text{s}$  後、今度は FET1 のゲートが  $0\text{V}\rightarrow 10\text{V}$  となり ON します。10V がモータに加えられ正転します。

(b) 正転→ブレーキに変えるとき

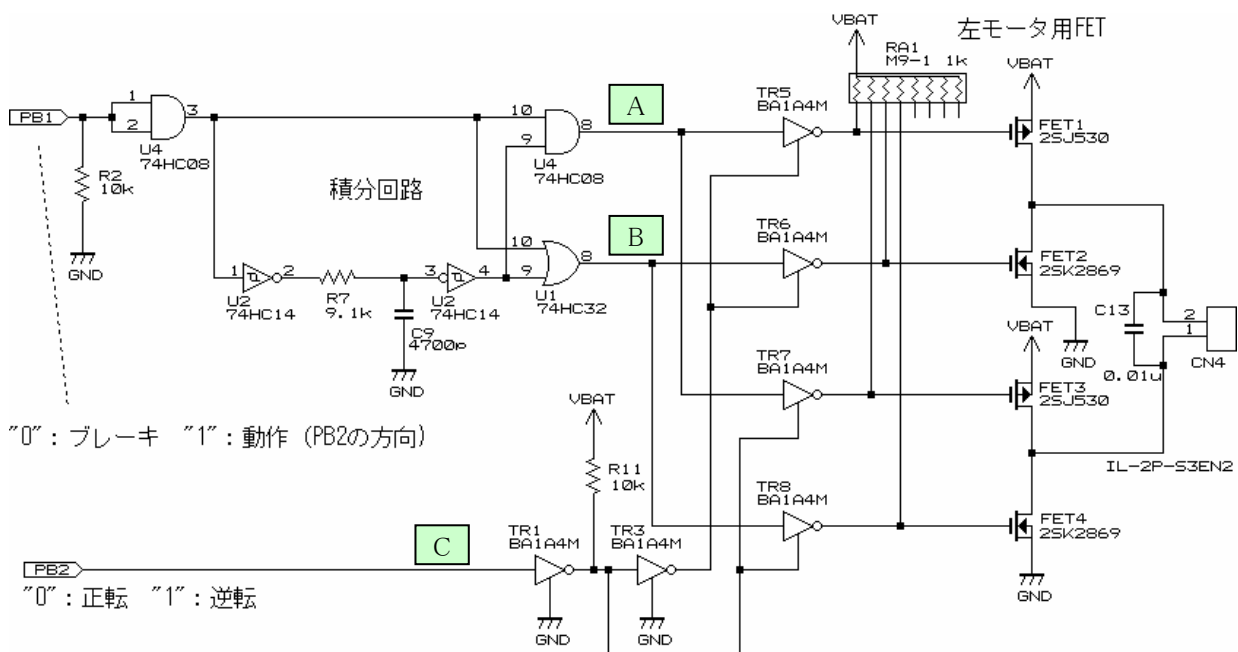
1. A点の信号を“1”→“0”にかえると、FET1 のゲートが  $10\text{V}\rightarrow 0\text{V}$  となり FET1 は OFF になります。ただし、遅延時間があるため遅れて OFF になります。この時点では、FET1 も FET2 も OFF 状態のため、モータはフリー状態となります。
2. A点の信号を変えてから  $50\ \mu\text{s}$  後、今度は FET2 のゲートが  $0\text{V}\rightarrow 10\text{V}$  となり ON します。0V がモータに加えられ、両端子 0V なのでブレーキ動作になります。

このように、動作を切り替えるときはいったん、両 FET とも OFF のフリー状態を作って、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)と同じにします。

### 5.6.7 モータドライブ基板の回路

実際の回路は、積分回路、FET回路の他、正転／逆転切り換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。PB1 が PWM を加える端子、PB2 が正転／逆転を切り替える端子です。



A	B	C	FET1 のゲート	FET2 のゲート	FET3 のゲート	FET4 のゲート	CN4 2ピン	CN4 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	FET1 のゲート	FET2 のゲート	FET3 のゲート	FET4 のゲート	CN4 2ピン	CN4 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

※A,B,C: “0”=0V、“1”=5V

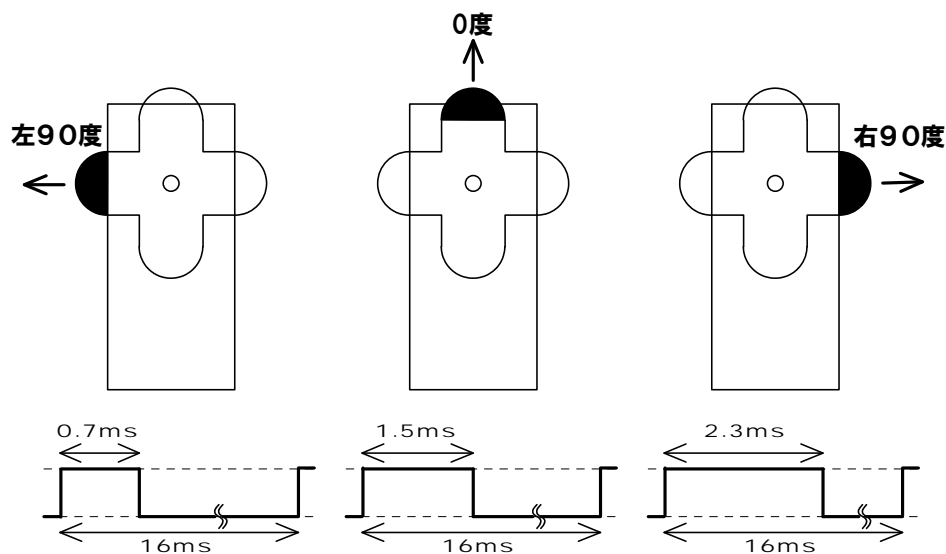
※フリーについて

フリーは、PchFET と NchFET のショートを避けるために積分回路で作っています。そのため、プログラムでフリーにすることはできません。モータドライブ基板 Vol.3 の停止はすべてブレーキです。  
フリーの時間を変えたい場合は、積分回路の C と R の値を変えます。

## 5.7 サーボ制御

### 5.7.1 原理

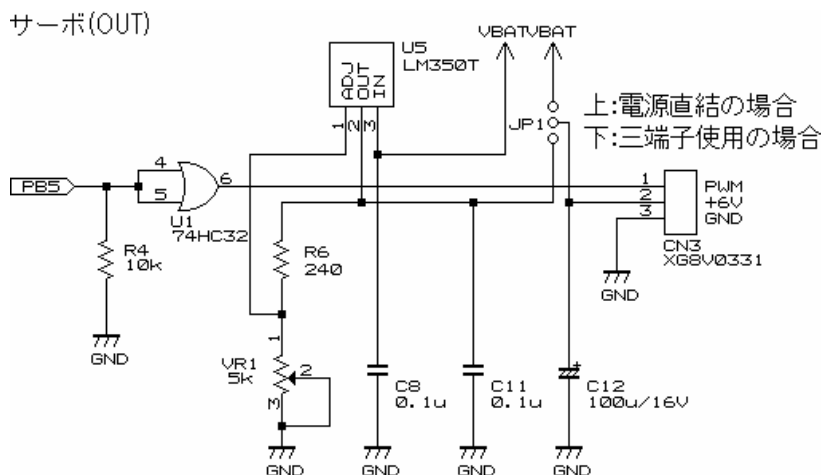
サーボは周期 16[ms]のパルスを加え、そのパルスの ON 幅でサーボの角度が決まります。  
サーボの回転角度と ON のパルス幅の関係は、サーボのメーカーや個体差によって多少の違いがありますが、ほとんどが下図のような関係になります。



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

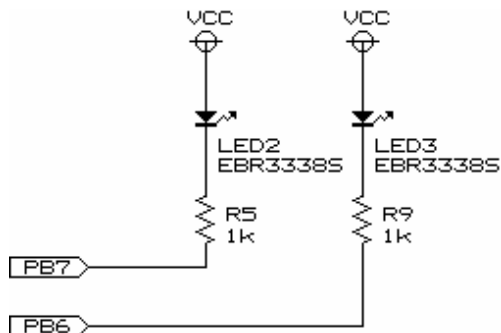
H8 マイコンのリセット同期式 PWM モードで PWM 信号を生成して、サーボを制御します。

### 5.7.2 回路



1. ポート B の bit5 から PWM 信号を出力します。プログラムは、ITU4\_BRB の値を変えると ON 幅が変わります。
2. ポートとサーボの1ピンの中に OR 回路(74HC32)を入れてバッファとします。例えば、1ピンに間違っって電源を接続したりノイズが混入して端子が壊れてしまう場合、ポート B の bit5 とサーボの 1ピンが直結ならマイコンのポートを壊します。これは致命的です。74HC32 なら 14ピン IC なので簡単に交換できます。
3. 2ピンは、サーボ用電源です。モータ用電源が電池4本以下の場合、JP1 の上側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 電流を流せる三端子レギュレータにて電圧を 6V 一定にします。JP1 は下側をショートさせます。

### 5.8 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 には順電圧 1.7V、電流 20mA 流すことができます。

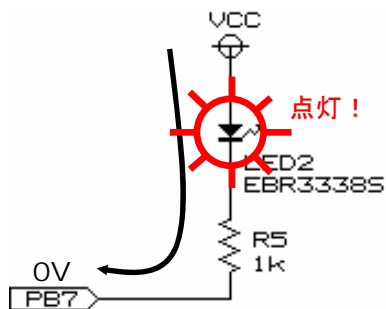
電流制限抵抗は

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

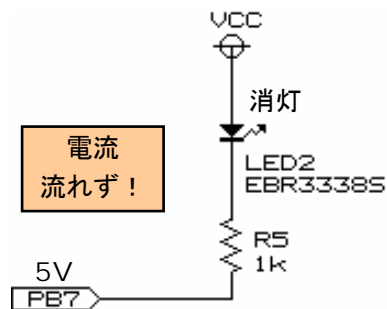
となります。

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、

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



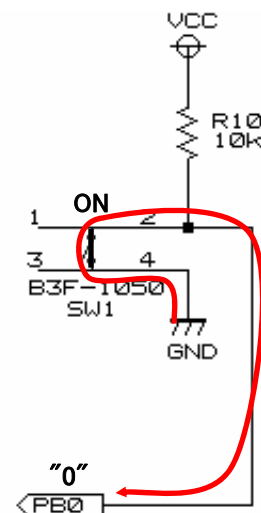
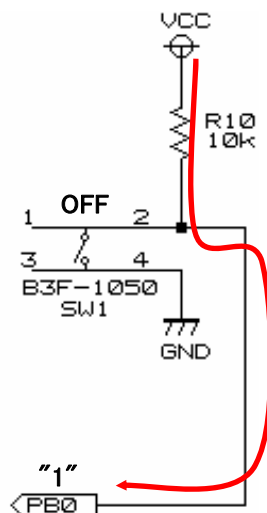
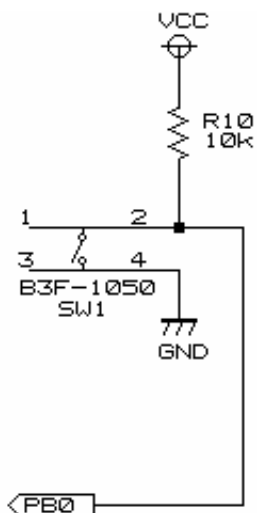
PB7 に"0"を出力すると、LED のカソード側が 0V になり、電流が流れ、LED は光ります。



PB7 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。

## 5.9 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。



スイッチは、10kΩ でプルアップされ、ポートBのbit0に繋がっています。

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PB0に入力されます。

スイッチが押されると、GND を通して"0"がPB0に入力されます。

## 6. サンプルプログラム

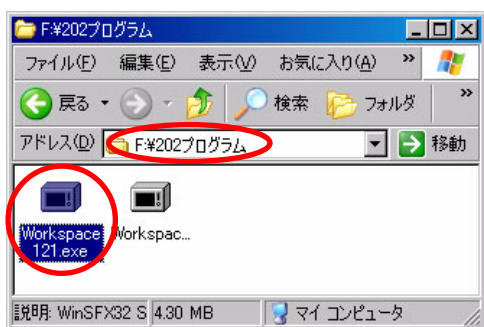
### 6.1 ルネサス統合開発環境

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

### 6.2 サンプルプログラムのインストール

サンプルプログラムをインストールします。

#### 6.2.1 CDからソフトを取得する



2007 年以降の講習会 CD がある場合、「CD ドライブ→202 プログラム」フォルダにある、「Workspace121.exe」を実行します。数字の 121 は、バージョンにより異なります。

#### 6.2.2 ホームページからソフトを取得する



●ルネサス統合開発環境 H8/3048関連プログラム Ver1.21 2007.03.20  
**NEW!!**  
ルネサス統合開発環境で使用するH8/3048関係のサンプルプログラムです。自己解凍方式で、実行すると自動でプログラムがインストールされます。  
※ Ver1.10より、ヘッダファイルなどの共通のファイルは、「c:\workspace\common」フォルダに入れています。  
※ 「C:\WorkSpace\common」フォルダ等にあるi2c\_eeprom.cの日付が2006/09/21以前のプログラムにはバグがありました(EEP-ROMを使用しているポートへの出力が出来ませんでした)。お詫びして訂正致します。2006/09/22以降のi2c\_eeprom.cをご使用下さい。  
→ [DOWNLOAD](#) (EXE 約4.30MB)

●ルネサス統合開発環境 H8/3687関連プログラム Ver1.00 2007.04.03  
**NEW!!**  
ルネサス統合開発環境で使用するH8/3687関係のサンプルプログラムです。自己解凍方式で、実行すると自動でプログラムがインストールされます。  
→ [DOWNLOAD](#) (EXE 約1.42MB)

●プログラム解説マニュアルkit06版 第1.10版 2006.09.04  
ダウンロード

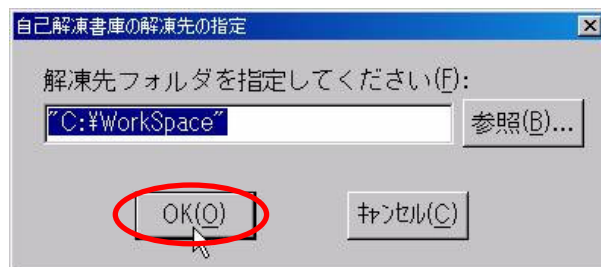
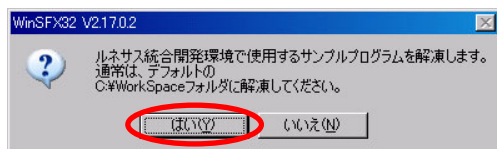
#### 1. マイコンカーラリーサイト

「<http://www.mcr.gr.jp/>」の技術情報→ダウンロード内のページへ行きます。

#### 2. 「ルネサス統合開発環境 H8/3048 関連プログラム」

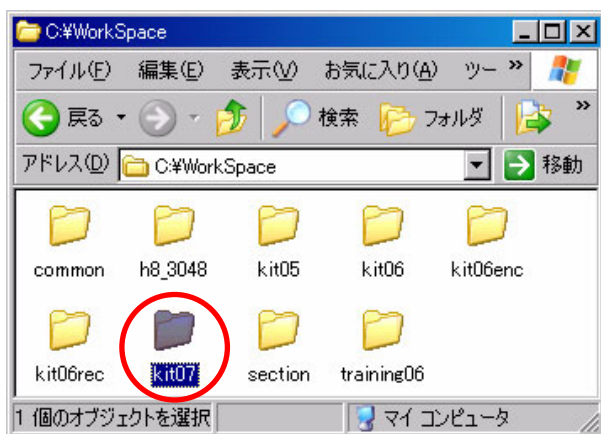
をダウンロードします。

### 6.2.3 インストール



1. CD またはダウンロードした「Workspace121.exe」を実行します。「はい」をクリックします。

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

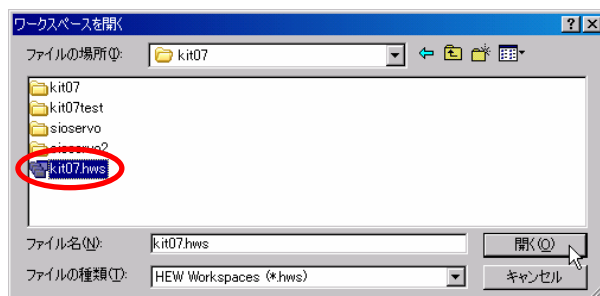
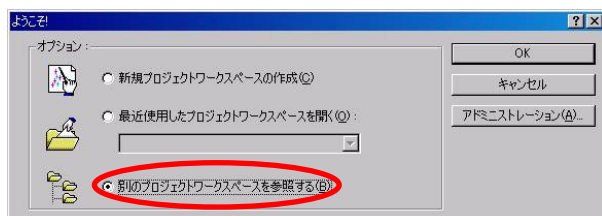


3. 解凍が終わったら、エクスプローラで「C ドライブ→Workspace」フォルダを開いてみてください。複数のフォルダがあります。今回使用するのは、「kit07」です。

### 6.3 ワークスペース「kit07」を開く

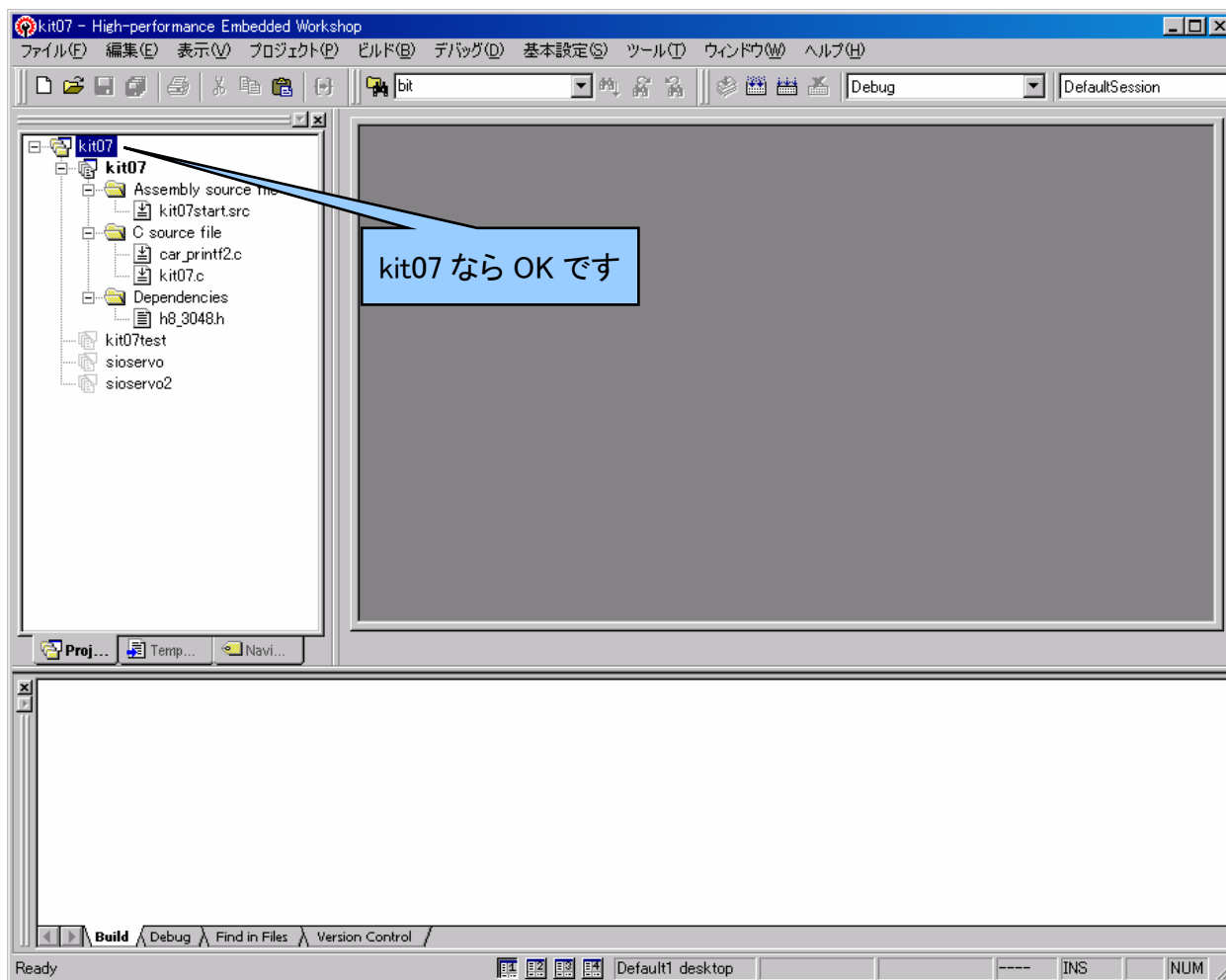


ルネサス統合開発環境を実行します。



「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。

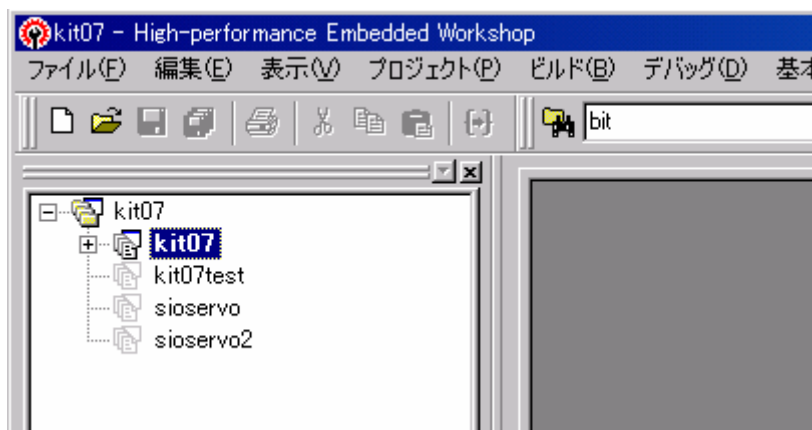
Cドライブ→Workspace→kit07 の「kit07.hws」を選択します。



kit07 というワークスペースが開かれます。



## 6.4 プロジェクト



ワークスペース「kit07」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
kit07	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。
kit07test	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「動作テストマニュアル」を参照して下さい。
sioservo	サーボのセンタを調整するプログラムです。後述します。
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。後述します。

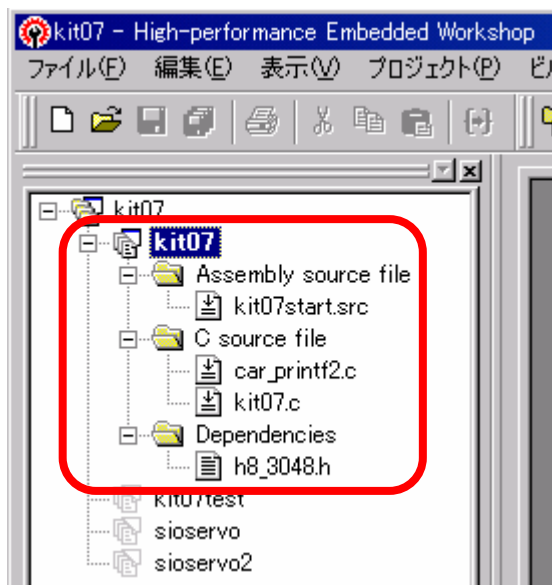
## 7. プロジェクト内のファイルの関わりと実行順

### 7.1 概要

ここでは、ワークスペース「kit07」のプロジェクト「kit07」を例にして、下記について説明します。

- ・プロジェクト内にあるファイルがどう関わっているのか
- ・電源を入れてから、どのような順番で実行されていくのか

### 7.2 プロジェクトのファイル構成



プロジェクト「kit07」は、下記のファイルから構成されています。

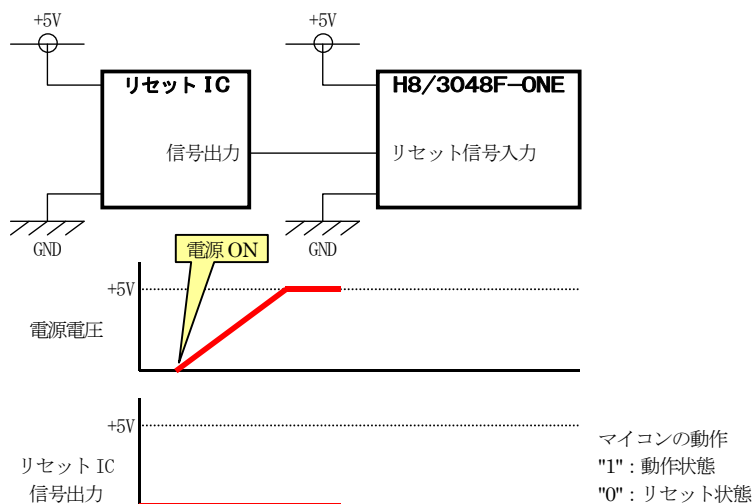
	ファイル名	内容
1	kit07start.src	アセンブリ言語で記述されたアセンブリソースファイルです。このファイルの構造は下記のようになっています。 kit07start.src = <span style="border: 1px solid black; padding: 2px;">ベクタアドレス</span> + <span style="border: 1px solid black; padding: 2px;">スタートアップルーチン</span>
2	kit07.c	C 言語ソースファイルです。このファイルは、H8/3048F-ONE の内蔵周辺機能の初期化、マイコンカーを制御するメインプログラムが含まれています。
3	car_printf2.c	C 言語ソースファイルです。初期値のないグローバル変数(セクション B 領域)、初期値のあるグローバル変数(セクション R 領域)の初期化用です。また、kit07.c プログラムでは使用していませんが、printf、scanf 文を使用するとき、このファイルが必要です。
4	h8_3048.h	H8/3048F-ONE の内蔵周辺機能の I/O レジスタを定義したファイルです。

### 7.3 プログラムの実行順

どのような順番でプログラムが実行されていくのか、説明していきます。

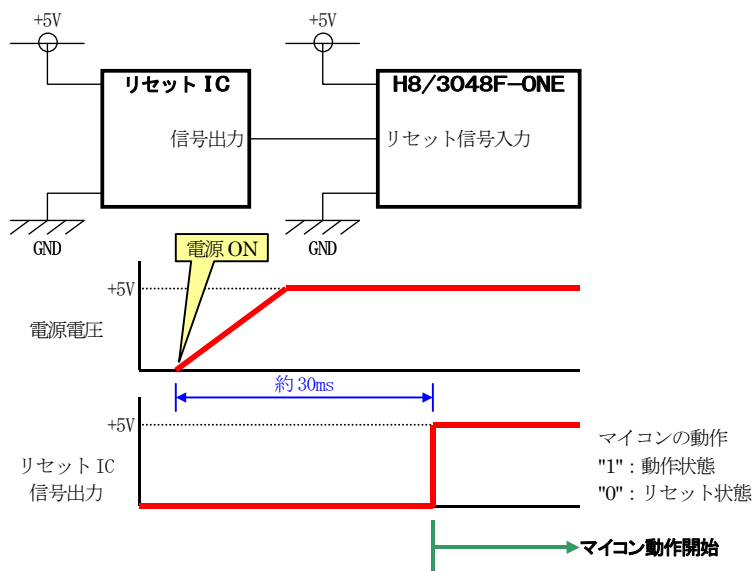
#### 7.3.1 電源を入れたときの動作

マイコンボードの電源を入れます。電源を入れる瞬間は、電圧が安定しません。そのため、RY3048Fone ボードに取り付けているリセット IC の出力信号がしばらくの間、“0”を出力します。出力先は H8 マイコンのリセット端子です。マイコンはリセット端子=“0”でリセット状態となるため、何もしません。



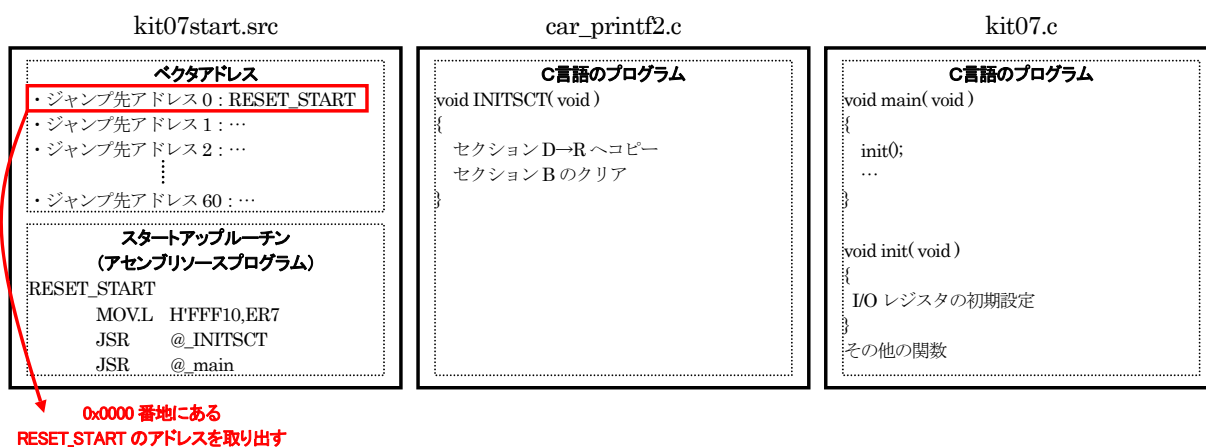
#### 7.3.2 マイコンの動作開始

約 30ms たつとリセット IC は電源が安定しマイコンの準備ができたと判断して、“1”を出力します。H8 マイコンのリセット信号入力端子が“1”になります。この瞬間から、マイコンは動き出します。ちなみに 30ms というのは、RY3048Fone ボードに取り付けているリセット IC の機能により、電源を入れてから 30ms 後に“0”→“1”になるためです。時間はリセット IC の種類によって違います。今回のリセット IC はたまたま 30ms だったと言うだけです。



### 7.3.3 ベクタアドレスからジャンプ先アドレスを取り出す

マイコンが動作を開始すると、ベクタアドレスと呼ばれる領域の「リセットが解除されたときのジャンプ先アドレス」が書いている部分(0番)から値を取り出します。



ベクタアドレスは、0番～60番まであります。

- 0番: リセット解除後のジャンプ先アドレス
- 7番: NMI 割り込み発生時のジャンプ先アドレス
- 12番: IRQ<sub>0</sub> 割り込み発生時のジャンプ先アドレス
- 13番: IRQ<sub>1</sub> 割り込み発生時のジャンプ先アドレス
- ...

とあらかじめ、「〇〇の割り込みが発生したときは、〇〇番からジャンプ先アドレスを読み込む」と決まっています。それらの割り込みを使うときは、ジャンプ先アドレスを登録しておきます。**リセットを解除したときは、0番からジャンプ先アドレスを読み込みます。**

ベクタアドレスは、ROMの0x00000～0x000f3番地の範囲で、ベクタ番号0番は0x00000番地、ベクタ番号1番は0x00004番地・・・と決められています。ベクタ番号とベクタアドレス、割り込みの発生元との関係は、次の表のようになっています。

割り込み要因	要因発生元	ベクタ番号	ベクタアドレス	IPR	優先順位
リセット	外部端子	0	H' 0000 ~ H' 0003	—	高 ↑
NMI	外部端子	7	H' 001C ~ H' 001F	—	
IRQ <sub>0</sub>		12	H' 0030 ~ H' 0033	IPRA7	
IRQ <sub>1</sub>		13	H' 0034 ~ H' 0037	IPRA6	
IRQ <sub>2</sub>		14	H' 0038 ~ H' 003B		
IRQ <sub>3</sub>		15	H' 003C ~ H' 003F	IPRA5	
IRQ <sub>4</sub>		16	H' 0040 ~ H' 0043		
IRQ <sub>5</sub>		17	H' 0044 ~ H' 0047		
リザーブ		—	18	H' 0048 ~ H' 004B	
		19	H' 004C ~ H' 004F		
WOVI (インターバルタイマ)	ウォッチドッグタイマ	20	H' 0050 ~ H' 0053	IPRA3	
CMI (コンペアマッチ)	リフレッシュコントローラ	21	H' 0054 ~ H' 0057		
リザーブ	—	22	H' 0058 ~ H' 005B		
		23	H' 005C ~ H' 005F		

IMIA0 (コンペアマッチ/インプットキャプチャA0)	ITU チャンネル0	24	H' 0060 ~ H' 0063	IPRA2
IMIB0 (コンペアマッチ/インプットキャプチャB0)		25	H' 0064 ~ H' 0067	
OVI0 (オーバフロー0)		26	H' 0068 ~ H' 006B	
リザーブ	—————	27	H' 006C ~ H' 006F	
IMIA1 (コンペアマッチ/インプットキャプチャA1)	ITU チャンネル1	28	H' 0070 ~ H' 0073	IPRA1
IMIB1 (コンペアマッチ/インプットキャプチャB1)		29	H' 0074 ~ H' 0077	
OVI1 (オーバフロー1)		30	H' 0078 ~ H' 007B	
リザーブ	—————	31	H' 007C ~ H' 007F	
IMIA2 (コンペアマッチ/インプットキャプチャA2)	ITU チャンネル2	32	H' 0080 ~ H' 0083	IPRA0
IMIB2 (コンペアマッチ/インプットキャプチャB2)		33	H' 0084 ~ H' 0087	
OVI2 (オーバフロー2)		34	H' 0088 ~ H' 008B	
リザーブ	—————	35	H' 008C ~ H' 008F	
IMIA3 (コンペアマッチ/インプットキャプチャA3)	ITU チャンネル3	36	H' 0090 ~ H' 0093	IPRB7
IMIB3 (コンペアマッチ/インプットキャプチャB3)		37	H' 0094 ~ H' 0097	
OVI3 (オーバフロー3)		38	H' 0098 ~ H' 009B	
リザーブ	—————	39	H' 009C ~ H' 009F	
IMIA4 (コンペアマッチ/インプットキャプチャA4)	ITU チャンネル4	40	H' 00A0 ~ H' 00A3	IPRB6
IMIB4 (コンペアマッチ/インプットキャプチャB4)		41	H' 00A4 ~ H' 00A7	
OVI4 (オーバフロー4)		42	H' 00A8 ~ H' 00AB	
リザーブ	—————	43	H' 00AC ~ H' 00AF	
DEDN0A	DMAC	44	H' 00B0 ~ H' 00B3	IPRB5
DEDN0B		45	H' 00B4 ~ H' 00B7	
DEDN1A		46	H' 00B8 ~ H' 00BB	
DEDN1B		47	H' 00BC ~ H' 00BF	
リザーブ	—————	48	H' 00C0 ~ H' 00C3	———
		49	H' 00C4 ~ H' 00C7	
		50	H' 00C8 ~ H' 00CB	
		51	H' 00CC ~ H' 00CF	
ERI0 (受信エラー0)	SCI チャンネル0	52	H' 00D0 ~ H' 00D3	IPRB3
RX10 (受信完了0)		53	H' 00D4 ~ H' 00D7	
TX10 (送信データエンプティ0)		54	H' 00D8 ~ H' 00DB	
TE10 (送信終了0)		55	H' 00DC ~ H' 00DF	
ERI1 (受信エラー1)	SCI チャンネル1	56	H' 00E0 ~ H' 00E3	IPRB2
RX11 (受信完了1)		57	H' 00E4 ~ H' 00E7	
TX11 (送信データエンプティ1)		58	H' 00E8 ~ H' 00EB	
TE11 (送信終了1)		59	H' 00EC ~ H' 00EF	
ADI (A/D エンド)	A/D	60	H' 00F0 ~ H' 00F3	IPRB1

↓  
低

例えば、リセット後、「RESET\_START」ラベルのある場所からプログラムを開始したいとします。その場合、kit07start.src プログラムに、下記のようにベクタアドレスを記述します。

16 :	.SECTION V			
17 :	<b>.DATA.L RESET_START</b>	;	0 H' 000000	リセット ←0番のジャンプ先アドレス
18 :	.DATA.L RESERVE	;	1 H' 000004	システム予約 ←1番のジャンプ先アドレス
19 :	.DATA.L RESERVE	;	2 H' 000008	システム予約 ←2番のジャンプ先アドレス
中略				
76 :	.DATA.L RESERVE	;	59 H' 0000ec	SCI1 TEI1 ←59番のジャンプ先アドレス
77 :	.DATA.L RESERVE	;	60 H' 0000f0	A/D ADI ←60番のジャンプ先アドレス

「.DATA.L」はロングサイズ(4 バイト)でデータを作りなさいという命令です。17 行目は、「RESET\_START」ラベルがある番地を数値にします。例えば、「RESET\_START」のラベルがある場所が 0x00100 番地なら、下記と同じことです。

17 :	.DATA.L H' 00000100	;	0 H' 000000	リセット
------	---------------------	---	-------------	------

登録する必要のない番号には「RESERVE」を入れておきます。

18 :	.DATA.L RESERVE	;	1 H' 000004	システム予約
19 :	.DATA.L RESERVE	;	2 H' 000008	システム予約
.....				
76 :	.DATA.L RESERVE	;	59 H' 0000ec	SCI1 TEI1
77 :	.DATA.L RESERVE	;	60 H' 0000f0	A/D ADI

「RESERVE」は、

4 :	RESERVE: .EQU H' FFFFFFFF	;	未使用領域のアドレス	
-----	---------------------------	---	------------	--

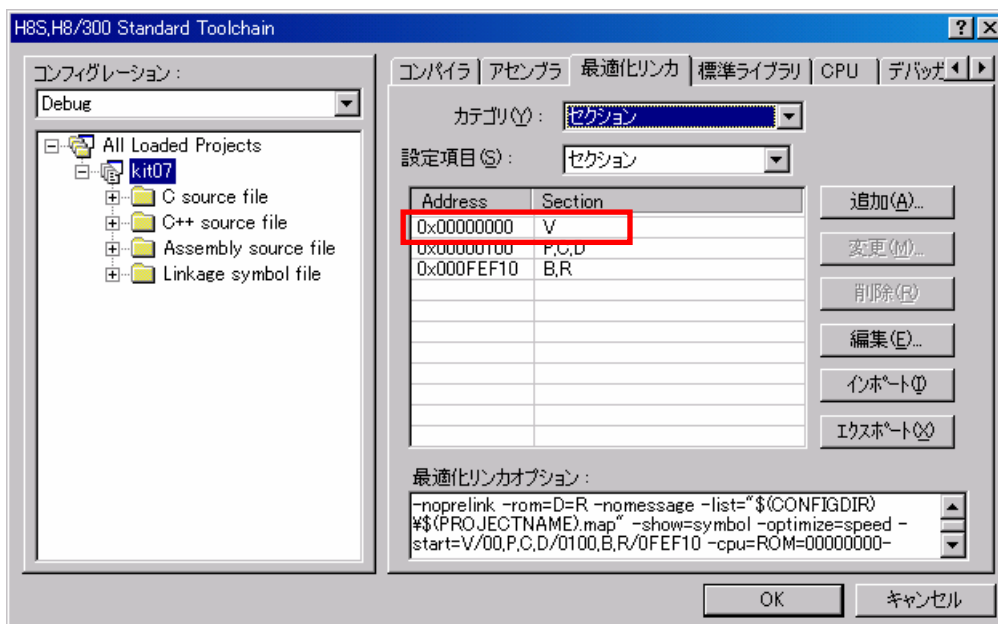
と登録しています。.EQU は、「イコール」命令です。「RESERVE」という単語が出てきたら「H' FFFFFFFF」に置き換えなさい、という意味です。登録する必要のない場所は、アドレスが分かりませんので、ダミーとして「H' FFFFFFFF」を入れておきます。結果的には、

18 :	.DATA.L H' FFFFFFFF	;	1 H' 000004	システム予約
------	---------------------	---	-------------	--------

ということになります。

ちなみに、16 行目の記述は「ここからはセクション V という領域ですよ」とルネサス統合開発環境(リンカ)に知らせている命令です。ルネサス統合開発環境はツールチェーンの設定によって、セクション V を何番地にするか決めます。

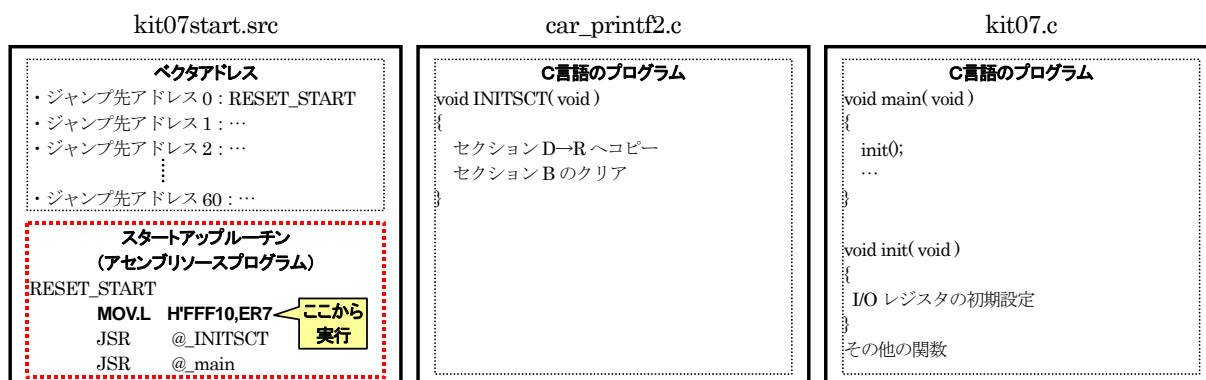
下記が実際のツールチェーンの設定です。ルネサス統合開発環境の「ビルド→H8S,H8/300 Standard Toolchain」を選択、「最適化リンク、カテゴリ:セクション、設定項目:セクション」を選択すると確認できます。



ここで、「セクション V は、0x00000 番地にしない」と設定されているので、ベクタアドレスであるセクション V 部分は 0x0000 番地から配置されるのです。「ベクタアドレスは 0x0000 番地から開始する」というのは、決まり事なので変更することはできません。ツールチェーンやセクションについての詳しい説明は、「ルネサス統合開発環境操作マニュアル 応用編」を参照してください。

### 7.3.4 スタートアップルーチンの実行

マイコンは、ベクタ番号 0 番に書かれている番地、すなわち「RESET\_START」部分から実行します。ここには、初期設定を行うプログラムを用意しておきます。実際のプログラムは下記のようになっています。



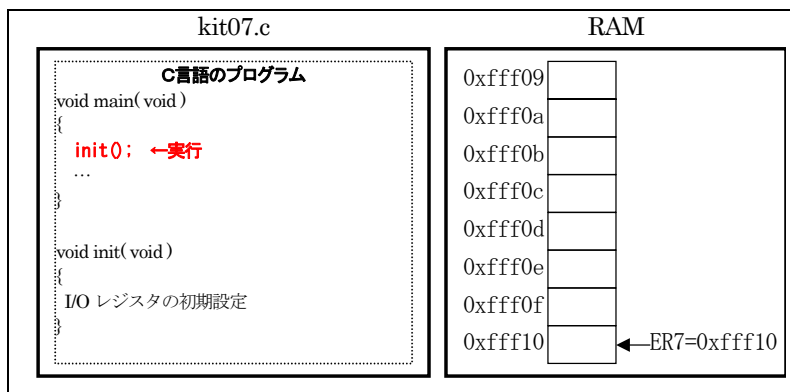
### 7.3.5 スタックポインタの設定

まず、スタックポインタの設定を行います。

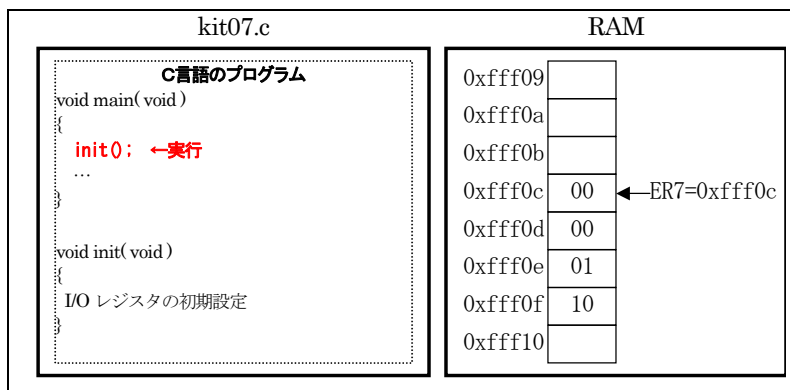
```

83 : RESET_START:
84 :      MOV.L  #'FFF10, ER7      ; スタックの設定
85 :      JSR   @_INITSCT         ; セクション D, R, B の設定
86 :      JSR   @_main            ; C 言語の main() 関数へジャンプ
87 : OWARI:
88 :      BRA   OWARI
    
```

スタックポインタとは、番地やデータを一時的に待避させるアドレスのことです。  
 例えば、init 関数を呼んだとします(下図)。

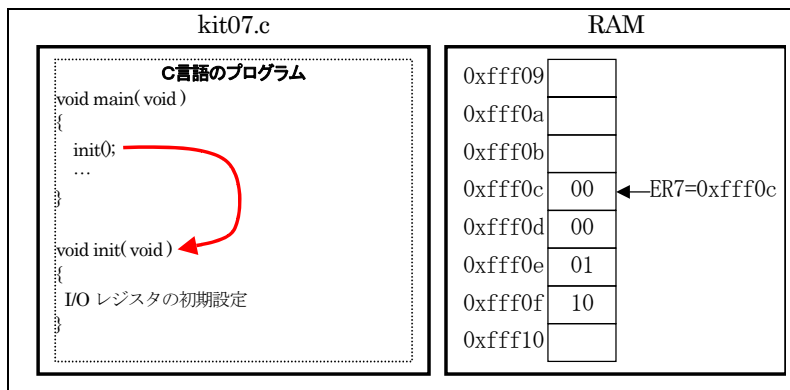


マイコンは、スタックポインタ(ER7)の値を-4 します。その番地である 0xffff0c 番地に、今実行しているプログラムの次のプログラムがある番地を書き込みます。例えば、その番地が 0x000110 番地なら、0xffff0c 番地には 0x00000110 と保存されます(下図)。

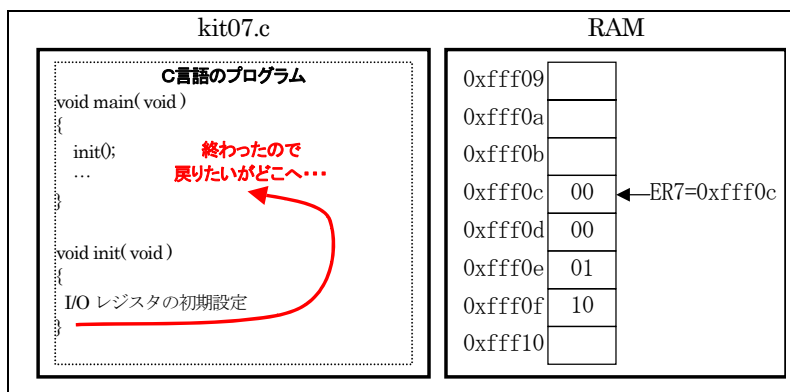




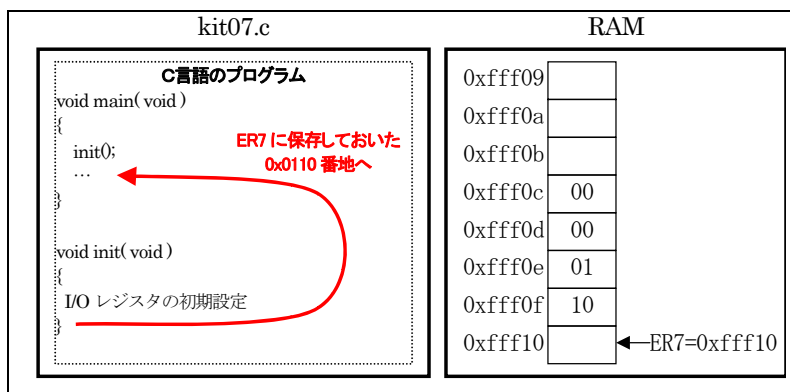
保存後、init 関数へジャンプして、init 関数を実行します(下図)。



実行が終わったら、呼ばれた部分へ戻ります。しかし、それは何処だったでしょうか？(下図)



ここで、スタックポインタの意味があるのです。init 関数を呼んだとき、戻り先をスタックポインタで示している番地に保存しました。スタックポインタ(ER7)が示している番地のデータを読み込み、その番地を実行すればよいのです。スタックポインタ(ER7)の値は+4しておきます(下図)。

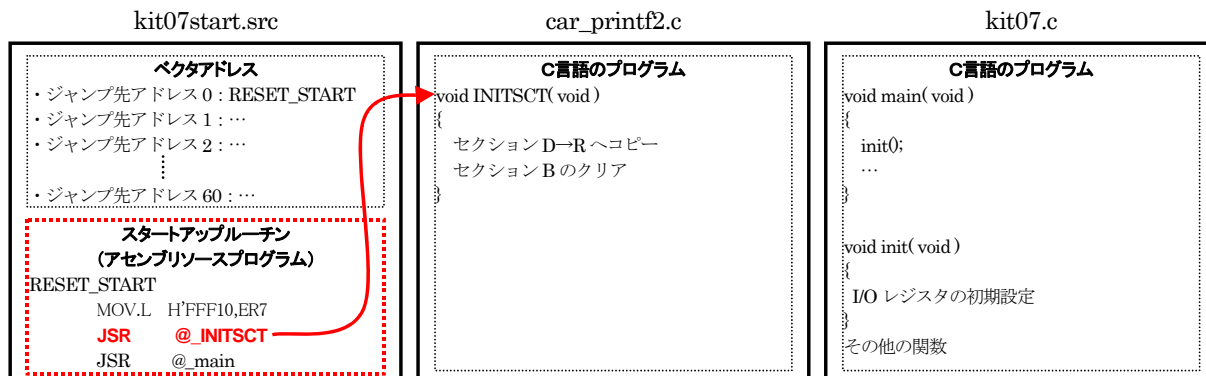


このように、スタックポインタは戻る番地や値を保存しておきます。**スタックポインタを設定しないと、戻り先が分からなくなるので、プログラムが暴走します。そのため、プログラムを実行するとき、一番最初にスタックポインタを設定しなければ行けないのです。**

それ以外にも、割り込み発生時の戻り先や CCR を保存したりします。詳しくは「スタックポインタ」という単語でインターネットを検索するか、制御の教科書を参照してください。

### 7.3.6 INITSCT関数の実行

次に「JSR @\_INITSCT」を実行します。JSR とは、「ジャンプサブルーチン」の意味です。INITSCT へジャンプして、その処理が終わったら戻ってきなさい、という意味です。ちなみに「INITSCT」の前の「\_」(アンダーバー)は、アセンブリソースプログラムからC言語ソースプログラムの関数を呼び出す場合は、「\_」を付けなければいけないという決まりがあるためです。



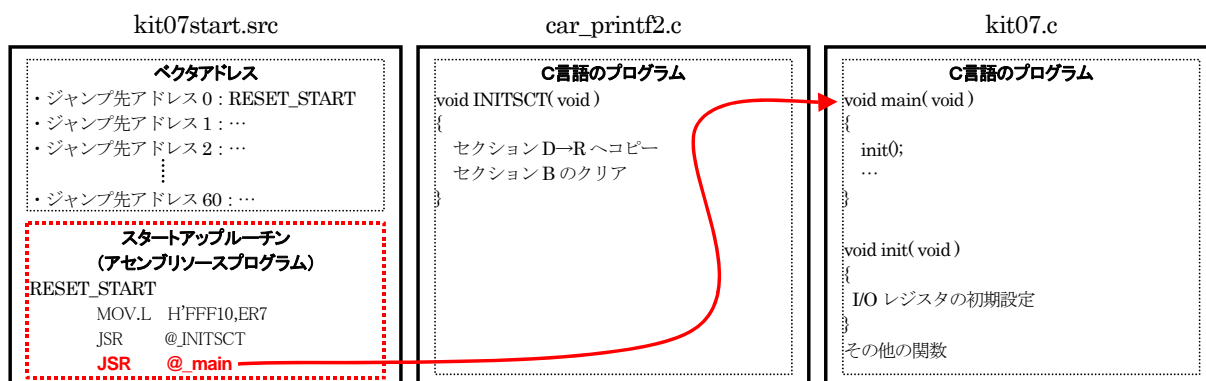
INITSCT 関数を実行します。この関数は何をしているのでしょうか。下記のような役割です。

- ・初期値のあるグローバル変数(セクション R 領域)の値を設定
- ・初期値のないグローバル変数(セクション B 領域)をクリア

詳しくは、「ルネサス統合開発環境操作マニュアル 応用編」のセクションを参照してください。

### 7.3.7 main関数の実行

初期値のあるグローバル変数の設定、初期値のないグローバル変数のクリアが終わったら、main 関数を実行します。main 関数内には、読んで字の如く、メインのプログラムを入れておきます。



### 7.3.8 IMPORT宣言

ただ、ちょっとした問題があります。アセンブルやコンパイルはファイルごとに行います。

```
86 :          JSR      @_main          ; C言語の main() 関数へジャンプ
```

をアセンブルするとき、「\_main」を探します。しかし、「\_main」は kit07start.src 内にはありません。「\_main」は、io.c ファイル内にあります。そのため、「\_main」は他の場所にあるので、他を探してください、とアセンブラに知らせる必要があります。その命令が、「IMPORT」命令なのです。

```
9 :          .IMPORT  _main
```

という記述で、アセンブラは「\_main」が他のファイルにあることを理解して「\_main」というラベル名があれば、予約だけしておきます。その場合、リンカージェディタがリンク時に実際のアドレスを入れます。

同様に、「\_INITSCT」や、割り込みで使うプログラムも IMPORT 宣言しておきます。

```
10 :         .IMPORT  _INITSCT
11 :         .IMPORT  _interrupt_timer0
```

## 8. プログラム解説「kit07.c」

### 8.1 プログラムリスト

太字部分が、kit06.c から kit07.c に改造するに当たって追加、変更になった部分です。

```

1 : /*****
2 : /* マイコンカートレース基本プログラム "kit07.c" */
3 : /* 2007.05 ジャパンマイコンカーラリー実行委員会 */
4 : /*****
5 : /*
6 : このプログラムは、下記基板に対応しています。
7 : ・モータドライブ基板 (Vol. 3)
8 : ・センサ基板Ver. 4
9 :
10 : このプログラムは、下記レギュレーションに対応しています。
11 : ・レーンチェンジ
12 : ・スタートバーによるスタート方式
13 : */
14 :
15 : /*=====*/
16 : /* インクルード */
17 : /*=====*/
18 : #include <machine.h>
19 : #include "h8_3048.h"
20 :
21 : /*=====*/
22 : /* シンボル定義 */
23 : /*=====*/
24 :
25 : /* 定数設定 */
26 : #define TIMER_CYCLE 3071 /* タイマのサイクル 1ms */
27 : /* φ/8で使用する場合、 */
28 : /* φ/8 = 325.5[ns] */
29 : /* ∴TIMER_CYCLE = */
30 : /* 1[ms] / 325.5[ns] */
31 : /* = 3072 */
32 : #define PWM_CYCLE 49151 /* PWMのサイクル 16ms */
33 : /* ∴PWM_CYCLE = */
34 : /* 16[ms] / 325.5[ns] */
35 : /* = 49152 */
36 : #define SERVO_CENTER 5000 /* サーボのセンタ値 */
37 : #define HANDLE_STEP 26 /* 1° 分の値 */
38 :
39 : /* マスク値設定 × : マスクあり (無効) ○ : マスク無し (有効) */
40 : #define MASK2_2 0x66 /* ×○○××○○× */
41 : #define MASK2_0 0x60 /* ×○○××××× */
42 : #define MASK0_2 0x06 /* ×××××○○× */
43 : #define MASK3_3 0xe7 /* ○○○××○○○ */
44 : #define MASK0_3 0x07 /* ×××××○○○ */
45 : #define MASK3_0 0xe0 /* ○○○××××× */
46 : #define MASK4_0 0xf0 /* ○○○○×××× */
47 : #define MASK0_4 0x0f /* ××××○○○○ */
48 : #define MASK4_4 0xff /* ○○○○○○○○ */
49 :
50 : /*=====*/
51 : /* プロトタイプ宣言 */
52 : /*=====*/
53 : void init( void );
54 : void timer( unsigned long timer_set );
55 : int check_crossline( void );
56 : int check_rightline( void );
57 : int check_leftline( void );
58 : unsigned char sensor_inp( unsigned char mask );
59 : unsigned char dipsw_get( void );
60 : unsigned char pushsw_get( void );
61 : unsigned char startbar_get( void );
62 : void led_out( unsigned char led );
63 : void speed( int accele_l, int accele_r );
64 : void handle( int angle );
65 :
66 : /*=====*/
67 : /* グローバル変数の宣言 */
68 : /*=====*/
69 : unsigned long cnt0; /* timer関数用 */
70 : unsigned long cnt1; /* main内で使用 */
71 : int pattern; /* パターン番号 */
72 :
73 : /*****
74 : /* メインプログラム */
75 : /*****
76 : void main( void )
77 : {
78 : int i;

```

```

79 :
80 :     /* マイコン機能の初期化 */
81 :     init();                               /* 初期化 */
82 :     set_ccr( 0x00 );                       /* 全体割り込み許可 */
83 :
84 :     /* マイコンカーの状態初期化 */
85 :     handle( 0 );
86 :     speed( 0, 0 );
87 :
88 :     while( 1 ) {
89 :         switch( pattern ) {
90 :
91 :             /***/
92 :             パターンについて
93 :             0 : スイッチ入力待ち
94 :             1 : スタートバーが開いたかチェック
95 :             11 : 通常トレース
96 :             12 : 右へ大曲げの終わりのチェック
97 :             13 : 左へ大曲げの終わりのチェック
98 :             21 : 1本目のクロスライン検出時の処理
99 :             22 : 2本目を読み飛ばす
100 :            23 : クロスライン後のトレース、クランク検出
101 :            31 : 左クランククリア処理 安定するまで少し待つ
102 :            32 : 左クランククリア処理 曲げ終わりのチェック
103 :            41 : 右クランククリア処理 安定するまで少し待つ
104 :            42 : 右クランククリア処理 曲げ終わりのチェック
105 :            51 : 1本目の右ハーフライン検出時の処理
106 :            52 : 2本目を読み飛ばす
107 :            53 : 右ハーフライン後のトレース
108 :            54 : 右レーンチェンジ終了のチェック
109 :            61 : 1本目の左ハーフライン検出時の処理
110 :            62 : 2本目を読み飛ばす
111 :            63 : 左ハーフライン後のトレース
112 :            64 : 左レーンチェンジ終了のチェック
113 :            *****/
114 :
115 :         case 0:
116 :             /* スイッチ入力待ち */
117 :             if( pushsw_get() ) {
118 :                 pattern = 1;
119 :                 cnt1 = 0;
120 :                 break;
121 :             }
122 :             if( cnt1 < 100 ) {                /* LED点滅処理 */
123 :                 led_out( 0x1 );
124 :             } else if( cnt1 < 200 ) {
125 :                 led_out( 0x2 );
126 :             } else {
127 :                 cnt1 = 0;
128 :             }
129 :             break;
130 :
131 :         case 1:
132 :             /* スタートバーが開いたかチェック */
133 :             if( !startbar_get() ) {
134 :                 /* スタート!! */
135 :                 led_out( 0x0 );
136 :                 pattern = 11;
137 :                 cnt1 = 0;
138 :                 break;
139 :             }
140 :             if( cnt1 < 50 ) {                /* LED点滅処理 */
141 :                 led_out( 0x1 );
142 :             } else if( cnt1 < 100 ) {
143 :                 led_out( 0x2 );
144 :             } else {
145 :                 cnt1 = 0;
146 :             }
147 :             break;
148 :
149 :         case 11:
150 :             /* 通常トレース */
151 :             if( check_crossline() ) {        /* クロスラインチェック */
152 :                 pattern = 21;
153 :                 break;
154 :             }
155 :             if( check_rightline() ) {        /* 右ハーフラインチェック */
156 :                 pattern = 51;
157 :                 break;
158 :             }
159 :             if( check_leftline() ) {         /* 左ハーフラインチェック */
160 :                 pattern = 61;
161 :                 break;
162 :             }
163 :             switch( sensor_inp(MASK3_3) ) {
164 :                 case 0x00:
165 :                     /* センタ→まっすぐ */
166 :                     handle( 0 );
167 :                     speed( 100, 100 );
168 :                     break;
169 :

```

```

170 :         case 0x04:
171 :             /* 微妙に左寄り→右へ微曲げ */
172 :             handle( 5 );
173 :             speed( 100 , 100 );
174 :             break;
175 :
176 :         case 0x06:
177 :             /* 少し左寄り→右へ小曲げ */
178 :             handle( 10 );
179 :             speed( 80 , 67 );
180 :             break;
181 :
182 :         case 0x07:
183 :             /* 中くらい左寄り→右へ中曲げ */
184 :             handle( 15 );
185 :             speed( 50 , 38 );
186 :             break;
187 :
188 :         case 0x03:
189 :             /* 大きく左寄り→右へ大曲げ */
190 :             handle( 25 );
191 :             speed( 30 , 19 );
192 :             pattern = 12;
193 :             break;
194 :
195 :         case 0x20:
196 :             /* 微妙に右寄り→左へ微曲げ */
197 :             handle( -5 );
198 :             speed( 100 , 100 );
199 :             break;
200 :
201 :         case 0x60:
202 :             /* 少し右寄り→左へ小曲げ */
203 :             handle( -10 );
204 :             speed( 67 , 80 );
205 :             break;
206 :
207 :         case 0xe0:
208 :             /* 中くらい右寄り→左へ中曲げ */
209 :             handle( -15 );
210 :             speed( 38 , 50 );
211 :             break;
212 :
213 :         case 0xc0:
214 :             /* 大きく右寄り→左へ大曲げ */
215 :             handle( -25 );
216 :             speed( 19 , 30 );
217 :             pattern = 13;
218 :             break;
219 :
220 :         default:
221 :             break;
222 :     }
223 :     break;
224 :
225 : case 12:
226 :     /* 右へ大曲げの終わりのチェック */
227 :     if( check_crossline() ) { /* 大曲げ中もクロスラインチェック */
228 :         pattern = 21;
229 :         break;
230 :     }
231 :     if( check_rightline() ) { /* 右ハーフラインチェック */
232 :         pattern = 51;
233 :         break;
234 :     }
235 :     if( check_leftline() ) { /* 左ハーフラインチェック */
236 :         pattern = 61;
237 :         break;
238 :     }
239 :     if( sensor_inp(MASK3_3) == 0x06 ) {
240 :         pattern = 11;
241 :     }
242 :     break;
243 :
244 : case 13:
245 :     /* 左へ大曲げの終わりのチェック */
246 :     if( check_crossline() ) { /* 大曲げ中もクロスラインチェック */
247 :         pattern = 21;
248 :         break;
249 :     }
250 :     if( check_rightline() ) { /* 右ハーフラインチェック */
251 :         pattern = 51;
252 :         break;
253 :     }
254 :     if( check_leftline() ) { /* 左ハーフラインチェック */
255 :         pattern = 61;
256 :         break;
257 :     }

```

```

258 :         if( sensor_inp(MASK3_3) == 0x60 ) {
259 :             pattern = 11;
260 :         }
261 :         break;
262 :
263 :     case 21:
264 :         /* 1本目のクロスライン検出時の処理 */
265 :         led_out( 0x3 );
266 :         handle( 0 );
267 :         speed( 0, 0 );
268 :         pattern = 22;
269 :         cnt1 = 0;
270 :         break;
271 :
272 :     case 22:
273 :         /* 2本目を読み飛ばす */
274 :         if( cnt1 > 100 ) {
275 :             pattern = 23;
276 :             cnt1 = 0;
277 :         }
278 :         break;
279 :
280 :     case 23:
281 :         /* クロスライン後のトレース、クランク検出 */
282 :         if( sensor_inp(MASK4_4)==0xf8 ) {
283 :             /* 左クランクと判断→左クランククリア処理へ */
284 :             led_out( 0x1 );
285 :             handle( -38 );
286 :             speed( 10, 50 );
287 :             pattern = 31;
288 :             cnt1 = 0;
289 :             break;
290 :         }
291 :         if( sensor_inp(MASK4_4)==0x1f ) {
292 :             /* 右クランクと判断→右クランククリア処理へ */
293 :             led_out( 0x2 );
294 :             handle( 38 );
295 :             speed( 50, 10 );
296 :             pattern = 41;
297 :             cnt1 = 0;
298 :             break;
299 :         }
300 :         switch( sensor_inp(MASK3_3) ) {
301 :             case 0x00:
302 :                 /* センタ→まっすぐ */
303 :                 handle( 0 );
304 :                 speed( 40, 40 );
305 :                 break;
306 :             case 0x04:
307 :             case 0x06:
308 :             case 0x07:
309 :             case 0x03:
310 :                 /* 左寄り→右曲げ */
311 :                 handle( 8 );
312 :                 speed( 40, 35 );
313 :                 break;
314 :             case 0x20:
315 :             case 0x60:
316 :             case 0xe0:
317 :             case 0xc0:
318 :                 /* 右寄り→左曲げ */
319 :                 handle( -8 );
320 :                 speed( 35, 40 );
321 :                 break;
322 :         }
323 :         break;
324 :
325 :     case 31:
326 :         /* 左クランククリア処理 安定するまで少し待つ */
327 :         if( cnt1 > 200 ) {
328 :             pattern = 32;
329 :             cnt1 = 0;
330 :         }
331 :         break;
332 :
333 :     case 32:
334 :         /* 左クランククリア処理 曲げ終わりのチェック */
335 :         if( sensor_inp(MASK3_3) == 0x60 ) {
336 :             led_out( 0x0 );
337 :             pattern = 11;
338 :             cnt1 = 0;
339 :         }
340 :         break;
341 :
342 :     case 41:
343 :         /* 右クランククリア処理 安定するまで少し待つ */
344 :         if( cnt1 > 200 ) {
345 :             pattern = 42;
346 :             cnt1 = 0;
347 :         }
348 :         break;

```

```

349 :
350 :     case 42:
351 :         /* 右クランククリア処理 曲げ終わりのチェック */
352 :         if( sensor_inp(MASK3_3) == 0x06 ) {
353 :             led_out( 0x0 );
354 :             pattern = 11;
355 :             cnt1 = 0;
356 :         }
357 :         break;
358 :
359 :     case 51:
360 :         /* 1本目の右ハーフライン検出時の処理 */
361 :         led_out( 0x2 );
362 :         handle( 0 );
363 :         speed( 0, 0 );
364 :         pattern = 52;
365 :         cnt1 = 0;
366 :         break;
367 :
368 :     case 52:
369 :         /* 2本目を読み飛ばす */
370 :         if( cnt1 > 100 ) {
371 :             pattern = 53;
372 :             cnt1 = 0;
373 :         }
374 :         break;
375 :
376 :     case 53:
377 :         /* 右ハーフライン後のトレース、レーンチェンジ */
378 :         if( sensor_inp(MASK4_4) == 0x00 ) {
379 :             handle( 15 );
380 :             speed( 40, 31 );
381 :             pattern = 54;
382 :             cnt1 = 0;
383 :             break;
384 :         }
385 :         switch( sensor_inp(MASK3_3) ) {
386 :             case 0x00:
387 :                 /* センター→まっすぐ */
388 :                 handle( 0 );
389 :                 speed( 40, 40 );
390 :                 break;
391 :             case 0x04:
392 :             case 0x06:
393 :             case 0x07:
394 :             case 0x03:
395 :                 /* 左寄り→右曲げ */
396 :                 handle( 8 );
397 :                 speed( 40, 35 );
398 :                 break;
399 :             case 0x20:
400 :             case 0x60:
401 :             case 0xe0:
402 :             case 0xc0:
403 :                 /* 右寄り→左曲げ */
404 :                 handle( -8 );
405 :                 speed( 35, 40 );
406 :                 break;
407 :             default:
408 :                 break;
409 :         }
410 :         break;
411 :
412 :     case 54:
413 :         /* 右レーンチェンジ終了のチェック */
414 :         if( sensor_inp( MASK4_4 ) == 0x3c ) {
415 :             led_out( 0x0 );
416 :             pattern = 11;
417 :             cnt1 = 0;
418 :         }
419 :         break;
420 :
421 :     case 61:
422 :         /* 1本目の左ハーフライン検出時の処理 */
423 :         led_out( 0x1 );
424 :         handle( 0 );
425 :         speed( 0, 0 );
426 :         pattern = 62;
427 :         cnt1 = 0;
428 :         break;
429 :
430 :     case 62:
431 :         /* 2本目を読み飛ばす */
432 :         if( cnt1 > 100 ) {
433 :             pattern = 63;
434 :             cnt1 = 0;
435 :         }
436 :         break;
437 :

```



```

438 :     case 63:
439 :         /* 左ハーフライン後のトレース、レーンチェンジ */
440 :         if( sensor_inp(MASK4_4) == 0x00 ) {
441 :             handle( -15 );
442 :             speed( 31 , 40 );
443 :             pattern = 64;
444 :             cnt1 = 0;
445 :             break;
446 :         }
447 :         switch( sensor_inp(MASK3_3) ) {
448 :             case 0x00:
449 :                 /* センタ→まっすぐ */
450 :                 handle( 0 );
451 :                 speed( 40 , 40 );
452 :                 break;
453 :             case 0x04:
454 :             case 0x06:
455 :             case 0x07:
456 :             case 0x03:
457 :                 /* 左寄り→右曲げ */
458 :                 handle( 8 );
459 :                 speed( 40 , 35 );
460 :                 break;
461 :             case 0x20:
462 :             case 0x60:
463 :             case 0xe0:
464 :             case 0xc0:
465 :                 /* 右寄り→左曲げ */
466 :                 handle( -8 );
467 :                 speed( 35 , 40 );
468 :                 break;
469 :             default:
470 :                 break;
471 :         }
472 :         break;
473 :
474 :     case 64:
475 :         /* 左レーンチェンジ終了のチェック */
476 :         if( sensor_inp( MASK4_4 ) == 0x3c ) {
477 :             led_out( 0x0 );
478 :             pattern = 11;
479 :             cnt1 = 0;
480 :         }
481 :         break;
482 :
483 :     default:
484 :         /* どれでもない場合は待機状態に戻す */
485 :         pattern = 0;
486 :         break;
487 :     }
488 : }
489 :
490 :
491 : /*****
492 : /* H8/3048F-ONE 内蔵周辺機能 初期化 */
493 : *****/
494 : void init( void )
495 : {
496 :     /* I/Oポートの入出力設定 */
497 :     P1DDR = 0xff;
498 :     P2DDR = 0xff;
499 :     P3DDR = 0xff;
500 :     P4DDR = 0xff;
501 :     P5DDR = 0xff;
502 :     P6DDR = 0xf0; /* CPU基板上のDIP SW */
503 :     P8DDR = 0xff;
504 :     P9DDR = 0xf7; /* 通信ポート */
505 :     PADDR = 0xff;
506 :     PBDR = 0xc0;
507 :     PBDDR = 0xfe; /* モータドライブ基板Vol.3 */
508 :     /* ※センサ基板のP7は、入力専用なので入出力設定はありません */
509 :
510 :     /* ITU0 1msごとの割り込み */
511 :     ITU0_TCR = 0x23;
512 :     ITU0_GRA = TIMER_CYCLE;
513 :     ITU0_IER = 0x01;
514 :
515 :     /* ITU3,4 リセット同期PWMモード 左右モータ、サーボ用 */
516 :     ITU3_TCR = 0x23;
517 :     ITU_FCR = 0x3e;
518 :     ITU3_GRA = PWM_CYCLE; /* 周期の設定 */
519 :     ITU3_GRB = ITU3_BRB = 0; /* 左モータのPWM設定 */
520 :     ITU4_GRA = ITU4_BRA = 0; /* 右モータのPWM設定 */
521 :     ITU4_GRB = ITU4_BRB = SERVO_CENTER; /* サーボのPWM設定 */
522 :     ITU_TOER = 0x38;
523 :
524 :     /* ITUのカウントスタート */
525 :     ITU_STR = 0x09;
526 : }
527 :

```

```

528 : /******
529 : /* ITU0 割り込み処理 */
530 : /******
531 : #pragma interrupt( interrupt_timer0 )
532 : void interrupt_timer0( void )
533 : {
534 :     ITU0_TSR &= 0xfe;          /* フラグクリア */
535 :     cnt0++;
536 :     cnt1++;
537 : }
538 :
539 : /******
540 : /* タイマ本体 */
541 : /* 引数 タイマ値 1=1ms */
542 : /******
543 : void timer( unsigned long timer_set )
544 : {
545 :     cnt0 = 0;
546 :     while( cnt0 < timer_set );
547 : }
548 :
549 : /******
550 : /* センサ状態検出 */
551 : /* 引数 マスク値 */
552 : /* 戻り値 センサ値 */
553 : /******
554 : unsigned char sensor_inp( unsigned char mask )
555 : {
556 :     unsigned char sensor;
557 :
558 :     sensor = ~P7DR;
559 :     sensor &= 0xef;
560 :     if( sensor & 0x08 ) sensor |= 0x10;
561 :
562 :     sensor &= mask;
563 :
564 :     return sensor;
565 : }
566 :
567 : /******
568 : /* クロスライン検出処理 */
569 : /* 戻り値 0:クロスラインなし 1:あり */
570 : /******
571 : int check_crossline( void )
572 : {
573 :     unsigned char b;
574 :     int ret;
575 :
576 :     ret = 0;
577 :     b = sensor_inp(MASK3_3);
578 :     if( b==0xe7 ) {
579 :         ret = 1;
580 :     }
581 :     return ret;
582 : }
583 :
584 : /******
585 : /* 右ハーフライン検出処理 */
586 : /* 戻り値 0:なし 1:あり */
587 : /******
588 : int check_rightline( void )
589 : {
590 :     unsigned char b;
591 :     int ret;
592 :
593 :     ret = 0;
594 :     b = sensor_inp(MASK4_4);
595 :     if( b==0x1f ) {
596 :         ret = 1;
597 :     }
598 :     return ret;
599 : }
600 :
601 : /******
602 : /* 左ハーフライン検出処理 */
603 : /* 戻り値 0:なし 1:あり */
604 : /******
605 : int check_leftline( void )
606 : {
607 :     unsigned char b;
608 :     int ret;
609 :
610 :     ret = 0;
611 :     b = sensor_inp(MASK4_4);
612 :     if( b==0xf8 ) {
613 :         ret = 1;
614 :     }
615 :     return ret;
616 : }
617 :

```

```

618 : /******
619 : /* デイップスイッチ値読み込み */
620 : /* 戻り値 スイッチ値 0~15 */
621 : /******
622 : unsigned char dipsw_get( void )
623 : {
624 :     unsigned char sw;
625 :
626 :     sw = ^P6DR;          /* デイップスイッチ読み込み */
627 :     sw &= 0x0f;
628 :
629 :     return sw;
630 : }
631 :
632 : /******
633 : /* プッシュスイッチ値読み込み */
634 : /* 戻り値 スイッチ値 ON:1 OFF:0 */
635 : /******
636 : unsigned char pushsw_get( void )
637 : {
638 :     unsigned char sw;
639 :
640 :     sw = ^PBDR;          /* スイッチのあるポート読み込み */
641 :     sw &= 0x01;
642 :
643 :     return sw;
644 : }
645 :
646 : /******
647 : /* スタートバー検出センサ読み込み */
648 : /* 戻り値 センサ値 ON(バーあり):1 OFF(なし):0 */
649 : /******
650 : unsigned char startbar_get( void )
651 : {
652 :     unsigned char b;
653 :
654 :     b = ^P7DR;          /* スタートバー信号読み込み */
655 :     b &= 0x10;
656 :     b >>= 4;
657 :
658 :     return b;
659 : }
660 :
661 : /******
662 : /* LED制御 */
663 : /* 引数 スイッチ値 LED0:bit0 LED1:bit1 "0":消灯 "1":点灯 */
664 : /* 例)0x3→LED1:ON LED0:ON 0x2→LED1:ON LED0:OFF */
665 : /******
666 : void led_out( unsigned char led )
667 : {
668 :     unsigned char data;
669 :
670 :     led = ^led;
671 :     led <<= 6;
672 :     data = PBDR & 0x3f;
673 :     PBDR = data | led;
674 : }
675 :
676 : /******
677 : /* 速度制御 */
678 : /* 引数 左モータ:-100~100 , 右モータ:-100~100 */
679 : /* 0で停止、100で正転100%、-100で逆転100% */
680 : /******
681 : void speed( int accele_l, int accele_r )
682 : {
683 :     unsigned char sw_data;
684 :     unsigned long speed_max;
685 :
686 :     sw_data = dipsw_get() + 5;          /* デイップスイッチ読み込み */
687 :     speed_max = (unsigned long)(PWM_CYCLE-1) * sw_data / 20;
688 :
689 :     /* 左モータ */
690 :     if( accele_l >= 0 ) {
691 :         PBDR &= 0xfb;
692 :         ITU3_BRB = speed_max * accele_l / 100;
693 :     } else {
694 :         PBDR |= 0x04;
695 :         accele_l = -accele_l;
696 :         ITU3_BRB = speed_max * accele_l / 100;
697 :     }
698 :
699 :     /* 右モータ */
700 :     if( accele_r >= 0 ) {
701 :         PBDR &= 0xf7;
702 :         ITU4_BRA = speed_max * accele_r / 100;
703 :     } else {
704 :         PBDR |= 0x08;
705 :         accele_r = -accele_r;
706 :         ITU4_BRA = speed_max * accele_r / 100;
707 :     }
708 : }

```

```

709 :
710 : /*****
711 : /* サーボハンドル操作 */
712 : /* 引数   サーボ操作角度：-90～90 */
713 : /*      -90で左へ90度、0でまっすぐ、90で右へ90度回転 */
714 : /*****
715 : void handle( int angle )
716 : {
717 :     ITU4_BRB = SERVO_CENTER - angle * HANDLE_STEP;
718 : }
719 :
720 : /*****
721 : /* end of file */
722 : /*****

```

## 8.2 スタート

```

1 : /*****
2 : /* マイコンカートレース基本プログラム "kit07.c" */
3 : /*      2007.05 ジャパンマイコンカーラリー実行委員会 */
4 : /*****
5 : /*
6 : このプログラムは、下記基板に対応しています。
7 : ・モータドライブ基板 (Vol. 3)
8 : ・センサ基板 Ver. 4
9 :
10 : このプログラムは、下記レギュレーションに対応しています。
11 : ・レーンチェンジ
12 : ・スタートバーによるスタート方式
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。

### 8.3 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 <machine.h>
19 : #include "h8_3048.h"
    
```

「#include」がインクルード命令です。2つのファイルをインクルード命令で取り込んでいます。

machine.h	C 言語で記述できない CPU に特化した機能を提供する組み込み関数です。
h8_3048.h	H8/3048F-ONE 用の内蔵周辺機能の I/O レジスタを定義したファイルです。

詳しくは、H8/3048F-ONE 実習マニュアルの

「6.8.1 「machine.h」ファイルの取り込み」(P52)

「6.8.2 「h8\_3048.h」ファイルの取り込み」(P52)


「6.8.3 「h8\_3048.h」ファイルの内容」(P53)

を参照してください。

### 8.4 その他のシンボル定義

```

21 : /*=====*/
22 : /* シンボル定義 */
23 : /*=====*/
24 :
25 : /* 定数設定 */
26 : #define TIMER_CYCLE 3071 /* タイマのサイクル 1ms */
27 : /* φ/8で使用する場合、 */
28 : /* φ/8 = 325.5[ns] */
29 : /* ∴TIMER_CYCLE = */
30 : /* 1[ms] / 325.5[ns] */
31 : /* = 3072 */
32 : #define PWM_CYCLE 49151 /* PWMのサイクル 16ms */
33 : /* ∴PWM_CYCLE = */
34 : /* 16[ms] / 325.5[ns] */
35 : /* = 49152 */
36 : #define SERVO_CENTER 5000 /* サーボのセンタ値 */
37 : #define HANDLE_STEP 26 /* 1° 分の値 */
38 :
39 : /* マスク値設定 × : マスクあり(無効) ○ : マスク無し(有効) */
40 : #define MASK2_2 0x66 /* ×○○××○○× */
41 : #define MASK2_0 0x60 /* ×○○×××××× */
42 : #define MASK0_2 0x06 /* ×××××○○× */
43 : #define MASK3_3 0xe7 /* ○○○××○○○ */
44 : #define MASK0_3 0x07 /* ×××××○○○ */
45 : #define MASK3_0 0xe0 /* ○○○×××××× */
46 : #define MASK4_0 0xf0 /* ○○○○××××× */
47 : #define MASK0_4 0x0f /* ×××××○○○○ */
48 : #define MASK4_4 0xff /* ○○○○○○○○○ */
    
```

TIMER_CYCLE	<p>タイマサイクルは、ITU0 で割り込みを発生させる間隔を設定します。今回は、1[ms]とします。計算は、  <math>(1 \times 10^{-3}) \div (325.52 \times 10^{-9}) - 1 = \mathbf{3,071}</math>                  となります。詳しい説明は、ITU0 部分を参照してください。</p>
PWM_CYCLE	<p>PWM サイクルは、右モータ、左モータ、およびサーボに加える PWM 周期を設定します。今回は、16[ms]を PWM の周期とします。  <math>(16 \times 10^{-3}) \div (325.52 \times 10^{-9}) - 1 = \mathbf{49,151}</math>                  となります。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>
SERVO_CENTER	<p>サーボに加えるパルス幅で、サーボがどの角度になるか決まります。サーボセンタは、サーボがまっすぐを向くときの値を設定します。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。  <math>(1.5 \times 10^{-3}) \div (325.52 \times 10^{-9}) - 1 = 4,607</math>                  となります。しかし、サーボのセンタは<b>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</b>。例えるなら、人間の指紋のようなものでしょうか。そのため、ここではきりのいい 5,000 にしています。この値は、<b>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調整、変更します</b>。</p> <div style="text-align: right;">  <p>▲サーボホーン</p> </div>
HANDLE_STEP	<p>サーボのハンドルステップは、サーボが1度分移動するときの増減分の値です。サーボが、右に 90 度向くときは 2.3ms のパルスなので、  <math>(2.3 \times 10^{-3}) \div (325.52 \times 10^{-9}) = 7,065</math>                  サーボが、左に 90 度向くときは 0.7ms のパルスなので、  <math>(0.7 \times 10^{-3}) \div (325.52 \times 10^{-9}) = 2,150</math>  <math>(右 90 度) - (左 90 度) = 7,065 - 2,150 = 4,915</math>                  が 180 度分動く値です。これを 180 で割れば1度当たりの移動量が分かります。  <math>4,915 \div 180 = 27.31</math>                  正確に計算すると 27 がサーボ 1 度分の値です。ただし、過去のプログラムでは 26 としていたので、過去との互換を考慮して、<b>26</b> とします。</p>
MASK2_2	<p>センサの状態をマスクする値です。「MASK○_◎」として、左のセンサ○個、右のセンサ◎個を有効にして他をマスクする、という意味です。                  「MASK2_2」は、「0x66」と定義していますので、2進数で「<b>0110 0110</b>」と bit6,5,2,1 をそのままに、他は強制的に”0”にしています。</p>
MASK2_0	<p>「MASK2_0」は、「0x60」と定義していますので、2進数で「<b>0110 0000</b>」と bit6,5 をそのままに、他は強制的に”0”にしています。</p>
MASK0_2	<p>「MASK0_2」は、「0x06」と定義していますので、2進数で「<b>0000 0110</b>」と bit2,1 をそのままに、他は強制的に”0”にしています。</p>
MASK3_3	<p>「MASK3_3」は、「0xe7」と定義していますので、2進数で「<b>1110 0111</b>」と bit7,6,5,2,1,0 をそのままに、他は強制的に”0”にしています。</p>
MASK0_3	<p>「MASK0_3」は、「0x07」と定義していますので、2進数で「<b>0000 0111</b>」と bit2,1,0 をそのままに、他は強制的に”0”にしています。</p>
MASK3_0	<p>「MASK3_0」は、「0xe0」と定義していますので、2進数で「<b>1110 0000</b>」と bit7,6,5 をそのままに、他は強制的に”0”にしています。</p>
MASK4_0	<p>「MASK4_0」は、「0xf0」と定義していますので、2進数で「<b>1111 0000</b>」と bit7,6,5,4 をそのままに、他は強制的に”0”にしています。</p>
MASK0_4	<p>「MASK0_4」は、「0x0f」と定義していますので、2進数で「<b>0000 1111</b>」と bit3,2,1,0 をそのままに、他は強制的に”0”にしています。</p>
MASK4_4	<p>「MASK4_4」は、「0xff」と定義していますので、2進数で「<b>1111 1111</b>」とすべてのビットを有効にしています。</p>

※マスクについては後述します。

## 8.5 プロトタイプ宣言

```
50 : /*=====*/
51 : /* プロトタイプ宣言 */
52 : /*=====*/
53 : void init( void );
54 : void timer( unsigned long timer_set );
55 : int check_crossline( void );
56 : int check_rightline( void );
57 : int check_leftline( void );
58 : unsigned char sensor_inp( unsigned char mask );
59 : unsigned char dipsw_get( void );
60 : unsigned char pushsw_get( void );
61 : unsigned char startbar_get( void );
62 : void led_out( unsigned char led );
63 : void speed( int accele_l, int accele_r );
64 : 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。

詳しくは、H8/3048F-ONE 実習マニュアルの  
「**6.8.4 プロトタイプ宣言**」(P56)  
を参照してください。

## 8.6 グローバル変数の宣言

```

66 : /*=====*/
67 : /* グローバル変数の宣言          */
68 : /*=====*/
69 : unsigned long   cnt0;           /* timer関数用          */
70 : unsigned long   cnt1;           /* main内で使用        */
71 : int             pattern;        /* パターン番号         */
    
```

グローバル変数とは、関数の外で定義され、ど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。次のサンプルプログラムはその例を示したものです。

```

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 );    ←もちろん 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;
}
    
```

kit07.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	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、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 たったなら〇〇をしないで、たっていないなら□□をしないで、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。



## 8.7 メインプログラムを説明する前に

76～489 行がマイコンカーを制御するメインとなる main 関数が記載されています。しかし、main 関数は、main 関数の後に記載されている細かい関数を組み合わせてプログラムしています。そのため、先に細かい関数を解説した方が説明しやすいので、main 関数は一番最後に説明します。

## 8.8 H8/3048F-ONE内蔵周辺機能の初期化:init関数

### 8.8.1 プログラム

H8/3048F-ONE マイコンに内蔵されている機能の初期化を行います。「init」とは、「initialize(イニシャライズ)」の略で、初期化の意味です。下記が、I/O ポートに関する設定です。

```

491 : /*******/
492 : /* H8/3048F-ONE 内蔵周辺機能 初期化 */
493 : /*******/
494 : void init( void )
495 : {
496 :     /* I/Oポートの入出力設定 */
497 :     P1DDR = 0xff;
498 :     P2DDR = 0xff;
499 :     P3DDR = 0xff;
500 :     P4DDR = 0xff;
501 :     P5DDR = 0xff;
502 :     P6DDR = 0xf0;          /* CPU基板上的DIP SW */
503 :     P8DDR = 0xff;
504 :     P9DDR = 0xf7;        /* 通信ポート */
505 :     PADDR = 0xff;
506 :     PBDR = 0xc0;
507 :     PBDDR = 0xfe;        /* モータドライブ基板Vol.3 */
508 :     /* ※センサ基板のP7は、入力専用なので入出力設定はありません */

```

はじめに、I/O ポートの入出力設定を行います。

ポートの入出力設定について詳しくは、H8/3048F-ONE 実習マニュアルの

「**6.8.5 init 関数(I/O ポートの入出力設定)**」(P57)

「**6.8.6 データを出力する、読み込む**」(P60)

を参照してください。

### 8.8.2 ポートの接続

H8/3048F-ONE にはポート 1 からポート B まであります。マイコンカーキットは、下記のように接続されています。

ポート	説明	7	6	5	4	3	2	1	0
1	未接続	未接続	未接続	未接続	未接続	未接続	未接続	未接続	未接続
2	未接続	未接続	未接続	未接続	未接続	未接続	未接続	未接続	未接続
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	未接続
4	未接続	未接続	未接続	未接続	未接続	未接続	未接続	未接続	未接続
5	未接続	—	—	—	—	未接続	未接続	未接続	未接続
6	ディップ スイッチ	—	未接続	未接続	未接続	スイッチ 入力	スイッチ 入力	スイッチ 入力	スイッチ 入力
7	センサ 基板	コース センサ 入力	コース センサ 入力	コース センサ 入力	スタート バーセン サ入力	コース センサ 入力	コース センサ 入力	コース センサ 入力	コース センサ 入力
8	未接続	—	—	—	未接続	未接続	未接続	未接続	未接続
9	通信	未接続	未接続	未接続	未接続	通信(RxD) 入力	未接続	通信(TxD) 出力	未接続
A	未接続	未接続	未接続	未接続	未接続	未接続	未接続	未接続	未接続
B	モータド ライブ基板	LED1 出力	LED0 出力	サーボ 出力	右モータ 出力	右モータ 出力	左モータ 出力	左モータ 出力	スイッチ 入力

### 8.8.3 入出力を決める

ポートの入出力設定は下記のようになります。

出力	出力端子は出力に設定します。
入力	入力端子は入力に設定します。
未接続	未接続端子は出力に設定します。 何も接続されていない状態で入力設定にすると、外部からの雑音(ノイズ)が端子に入ってきて最悪の場合には、H8 マイコン内部の入力回路部分が壊れてしまいます。 <b>何も接続されていない端子は、プルアップ抵抗かプルダウン抵抗を接続して入力端子とするか、何も接続せずに出力設定とします。</b> <b>ポート7は常に入力のため、接続しない場合は必ず、プルアップ抵抗かプルダウン抵抗を接続しなければいけません。</b> 今回はセンサ基板が繋がっているため、プルアップやプルダウン抵抗は必要ありません。
—	端子のないピンです。入力でも出力でも構いませんが、「未接続は出力に設定」を適用して出力にしておきます。

### 8.8.4 実際の設定

それぞれの端子の接続が、入力か出力かまとめると下表のようになります。この表を基に、ポートの入出力設定を行います。

ポート	7	6	5	4	3	2	1	0	DDR
1	出力	出力	出力	出力	出力	出力	出力	出力	0xff
2	出力	出力	出力	出力	出力	出力	出力	出力	0xff
3	出力	出力	出力	出力	出力	出力	出力	出力	0xff
4	出力	出力	出力	出力	出力	出力	出力	出力	0xff
5	出力	出力	出力	出力	出力	出力	出力	出力	0xff
6	出力	出力	出力	出力	入力	入力	入力	入力	0xf0
7	入力	入力	入力	入力	入力	入力	入力	入力	—
8	出力	出力	出力	出力	出力	出力	出力	出力	0xff
9	出力	出力	出力	出力	入力	出力	出力	出力	0xf7
A	出力	出力	出力	出力	出力	出力	出力	出力	0xff
B	出力	出力	出力	出力	出力	出力	出力	入力	0xfe

※ポート7は常に入力のため、P7DDRはありません。

### 8.8.5 ポートBの詳細

ポートBにはモータドライブ基板 Vol.3 が接続されています。入出力方向は下表のようになります。

ピン番	信号、方向	詳細	“0”	“1”	入出力
1	—	+5V			
2	基板←PB7	LED1	点灯	消灯	出力
3	基板←PB6	LED0	点灯	消灯	出力
4	基板←PB5	サーボ信号	PWM 信号		出力
5	基板←PB4	右モータ PWM	停止	動作	出力
6	基板←PB3	右モータ回転方向	正転	逆転	出力
7	基板←PB2	左モータ回転方向	正転	逆転	出力
8	基板←PB1	左モータ PWM	停止	動作	出力
9	基板→PB0	プッシュスイッチ	押された	押されていない	入力
10	—	GND			

### 8.8.6 ポートBの初期出力値

LEDは消灯させたいので、“1”を出力します。モータは停止させたいので“0”、サーボはどちらでも良いのですが、とりあえず“0”としておきます。

bit	7	6	5	4	3	2	1	0
ポートBへ出力する値	1	1	0	0	0	0	0	入力端子

入力端子へは、何を書き込んでも何も起こりません。ただ、“1”にすると、何か意味があるのかと思われるため、“0”にします。まとめると、下記のようになります。

bit	7	6	5	4	3	2	1	0
ポートBへ出力する値	1	1	0	0	0	0	0	0

2進数を16進数に変換してPBDRへ設定します。

<b>506 :</b>	<b>PBDR = 0xc0;</b>	
507 :	PBDDR = 0xfe;	/* モータドライブ基板 Vol.3 */

となります。

### 8.8.7 PBDRとPBDDRの設定する順番

プログラムでは最初にPBDRへ0xc0をセットして、次にPBDDRへ0xfeをセットしています。なぜ入出力設定の前にデータをセットするのでしょうか？これは、リセットした直後のレジスタの値がどのようになっているかに関係してきます。

PBDDRの初期値は、ハードウェアマニュアルを見ると「0x00」です。端子は入力になっています。PBDRの初期値は、「0x00」です。そのため、先にPBDDRでポートを出力にセットすると、PBDRの値「0x00」が出力されてしまいます。LEDは“0”で点灯するのでLEDが点灯します。次のPBDRへの書き込みですぐに消灯するので問題ないと言えば無いのですが、仮に他のデバイスに接続されていた場合、一瞬有効になったことにより誤動作するかもしれません。このようなミス無くするために、先にPBDRを設定してLEDを消灯状態にしておいてから、PBDDRの設定を行っています。

## 8.9 ITU0 1msごとの割り込み設定

ITU のチャンネル 0 という機能を使って、1ms ごとに割り込みを発生させます。下記が init 関数内の ITU0 に関する部分です。ITU0 割り込みの設定、割り込みの許可を行っています。

```

510 :      /* ITU0 1msごとの割り込み */
511 :      ITU0_TCR = 0x23;
512 :      ITU0_GRA = TIMER_CYCLE;
513 :      ITU0_IER = 0x01;
中略
524 :      /* ITUのカウントスタート */
525 :      ITU_STR = 0x09;
    
```

割り込みの概要、ITU を使用した割り込みについては、H8/3048F-ONE 実習マニュアルの

「**8.6 割り込みの概要**」(P76)

「**8.7 割り込みを使う設定、割り込みを許可する**」(P78)

を参照してください。

### 8.9.1 ITU0 レジスタの設定

設定するレジスタ	詳細																																
ITU0_TCR	ITU0_CNT の +1 する時間、クリア要因の設定をします。 0x20→40.69[ns]でカウント    0x21→81.38[ns]でカウント 0x22→162.76[ns]でカウント    0x23→325.52[ns]でカウント 今回は、 <b>0x23</b> を選択します。																																
ITU0_GRA	割り込み周期を設定します。「設定したい周期 ÷ ITU0_CNT のカウント時間 - 1」です。 1ms ごとに割り込みを発生させたいので周期 1ms、 カウント時間は前の設定より 325.52[ns]なので $(1 \times 10^{-3}) \div (325.52 \times 10^{-9}) - 1 = \mathbf{3071}$ となります。																																
ITU0_IER	割り込みを許可します。 <b>0x01</b> を設定します。																																
ITU_STR	ITU のカウンタ (CNT) を動作させる設定です。ITU_STR は ITU0~4 共通です。 <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <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>—</td> <td>—</td> <td>—</td> <td>ITU4</td> <td>ITU3</td> <td>ITU2</td> <td>ITU1</td> <td>ITU0</td> </tr> <tr> <td></td> <td></td> <td></td> <td>未使用</td> <td>リセット同期 PWM モード で使用</td> <td>未使用</td> <td>未使用</td> <td>1ms ごとの割 り込みで使用</td> </tr> <tr> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>1</td> <td>0</td> <td>0</td> <td>1</td> </tr> </tbody> </table> よって、 <b>0x09</b> を設定します。	7	6	5	4	3	2	1	0	—	—	—	ITU4	ITU3	ITU2	ITU1	ITU0				未使用	リセット同期 PWM モード で使用	未使用	未使用	1ms ごとの割 り込みで使用	0	0	0	0	1	0	0	1
7	6	5	4	3	2	1	0																										
—	—	—	ITU4	ITU3	ITU2	ITU1	ITU0																										
			未使用	リセット同期 PWM モード で使用	未使用	未使用	1ms ごとの割 り込みで使用																										
0	0	0	0	1	0	0	1																										

## 8.9.2 割り込みプログラム

```

532 : void interrupt_timer0( void )
533 : {
534 :     ITU0_TSR &= 0xfe;           /* フラグクリア           */
535 :     cnt0++;
536 :     cnt1++;
537 : }

```

interrupt\_timer0 関数が、1ms ごとに割り込みで実行されます。

534 行…割り込みが発生したことにより ITU0\_TSR の bit0 が “1”になります。次回の割り込みに備えて bit0 を “0” にしておきます。

535 行…cnt0 変数を +1 しています。cnt0 はグローバル変数なので、どの関数からもアクセスできます。interrupt\_timer0 関数で 1ms ごとに +1 して、main 関数などで cnt0 の値をチェックすることにより正確に時間を計測することができます。

536 行…535 行と同様、cnt1 変数を +1 しています。

## 8.9.3 「#pragma interrupt」の設定

interrupt\_timer0 関数は、割り込み関数なので「#pragma interrupt」宣言しておきます。

```

531 : #pragma interrupt( interrupt_timer0 )   ←割り込み関数であることを宣言する
532 : void interrupt_timer0( void )
533 : {
534 :     ITU0_TSR &= 0xfe;           ←フラグのクリアを必ず行う
535 :     cnt0++;                     ←割り込みプログラム
536 :     cnt1++;                     ←割り込みプログラム
537 : }

```

## 8.9.4 全体の割り込みを許可する

init 関数終了後、全体の割り込みを許可します。

```

void main( void )
{
    init();           /* 初期化           */   ←初期化
    set_ccr( 0x00 ); /* 全体割り込み許可 */ ←全体の割り込み許可、必ず行う
}

```

## 8.9.5 ベクタアドレスの設定(srcファイル)

kit07start.src ソースファイルにベクタアドレスを設定します。今回は ITU0 の IMIA0 割り込みが発生するので、ベクタアドレスの表より 24 番部分 (0x0060 番地)に「\_interrupt\_timer0」を記述します。関数名を記述するとき、先頭に「\_\_ (アンダーバー)」を追加することを忘れないようにします。

```

.DATA.L __interrupt_timer0 ; 24 h'000060 ITU0 IMIA0

```

### 8.9.6 「.IMPORT」の設定(srcファイル)

「.IMPORT+関数名」で、別のファイルにこの関数があることを伝えます。

```
.IMPORT _interrupt_timer0 ; 外部参照
```

### 8.10 リセット同期PWMモードの設定

リセット同期 PWM モードを使用して PWM 信号を出力し、左モータ、右モータ、サーボを制御します。PWM 周期は、サーボ周期の 16[ms]にします。モータの PWM 周期は、1[ms]くらいでも良いのですが、周期は共通にしかできないのでサーボに合わせます。

```
515 : /* ITU3,4 リセット同期 PWM モード 左右モータ、サーボ用 */
516 : ITU3_TCR = 0x23;
517 : ITU_FCR = 0x3e;
518 : ITU3_GRA = PWM_CYCLE; /* 周期の設定 */
519 : ITU3_GRB = ITU3_BRB = 0; /* 左モータの PWM 設定 */
520 : ITU4_GRA = ITU4_BRA = 0; /* 右モータの PWM 設定 */
521 : ITU4_GRB = ITU4_BRB = SERVO_CENTER; /* サーボの PWM 設定 */
522 : ITU_TOER = 0x38;
523 :
524 : /* ITU のカウントスタート */
525 : ITU_STR = 0x09;
```

リセット同期 PWM モードについて詳しくは、H8/3048F-ONE 実習マニュアルの「**14.5.1 リセット同期 PWM モードの設定**」(P153)を参照してください。

レジスタ設定内容は、下記のようになります。

設定するレジスタ	詳細
ITU3_TCR	<p>ITU3_CNT の値が+1する時間、クリア要因の設定をします。                      0x20…周期が 2.666ms 以下の場合(+1する時間は 40.69[ns]ごと)                      0x21…周期が 5.333ms 以下の場合(+1する時間は 81.38[ns]ごと)                      0x22…周期が 10.67ms 以下の場合(+1する時間は 162.76[ns]ごと)                      0x23…周期が 21.33ms 以下の場合(+1する時間は 325.52[ns]ごと)                      今回は、周期 16ms なので <b>0x23</b> を設定します。</p> <pre>516 :      ITU3_TCR = 0x23;</pre>
ITU_FCR	<p>リセット同期PWMモードの設定と、バッファレジスタを使用するかの設定をします。  <b>0x3e</b> を設定します。PB0～PB5 の端子からは PWM 波形が出力されます。出力したくない場合は、ITU_TOERを設定することにより通常の I/O ポートとして使用できます。</p> <pre>517 :      ITU_FCR = 0x3e;</pre>
ITU3_GRA	<p>周期を設定します。計算は、「設定したい周期÷(ITU3_CNT の値が+1する時間)−1」です。                      周期 16ms、+1する時間 325.52ns なら  <math>(16 \times 10^{-3}) \div (325.52 \times 10^{-9}) - 1 = 49151</math>                      プログラムは、</p> <pre>518 :      ITU3_GRA = PWM_CYCLE;      /* 周期の設定 */</pre> <p>としています。PWM_CYCLE は、</p> <pre>32 : #define      PWM_CYCLE      49151</pre> <p>と定義しています。これは、49151 という数値だけでは意味が分かりづらいため、「PWM_CYCLE」と文字列で定義することにより、PWM のサイクルだと言うことを明確化しています。</p>
ITU3_BRB ITU3_GRB	<p>PB0 の OFF 幅、または PB1 の ON 幅を設定します。                      計算は、「設定したい幅÷(ITU3_CNT の値が+1する時間)−1」です。                      ITU3_GRB は、最初だけ ITU3_BRB と同じ値を設定します。その後は、バッファレジスタ ITU3_BRB に設定します。                      今回、PB0 は PWM を OFF にしています。PB1 には、左モータが接続されています。そのため、<b>ITU3_BRB へ値を設定するということは、左モータの ON 幅を設定しているということになります。</b>                      最初、左モータは停止にしたいので、PWM0%にします。</p> <pre>519 :      ITU3_GRB = ITU3_BRB = 0;      /* 左モータの PWM 設定 */</pre>
ITU4_BRA ITU4_GRA	<p>PB2 の OFF 幅、または PB4 の ON 幅を設定します。                      計算は、「設定したい幅÷(ITU3_CNT の値が+1する時間)−1」です。                      ITU4_GRA は、最初だけ ITU4_BRA と同じ値を設定します。その後は、バッファレジスタ ITU4_BRA に設定します。                      今回、PB2 は PWM を OFF にしています。PB4 には、右モータが接続されています。そのため、<b>ITU4_BRA へ値を設定するということは、右モータの ON 幅を設定しているということになります。</b>                      最初、右モータは停止にしたいので、PWM0%にします。</p> <pre>520 :      ITU4_GRA = ITU4_BRA = 0;      /* 右モータの PWM 設定 */</pre>



<p>ITU4_BRB ITU4_GRB</p>	<p>PB3 の OFF 幅、または PB5 の ON 幅を設定します。                  計算は、「設定したい幅 ÷ (ITU3_CNT の値が +1 する時間) - 1」です。                  ITU4_GRB は、最初だけ ITU4_BRB と同じ値を設定します。その後は、バッファレジスタ ITU4_BRB に設定します。                  今回、PB3 は PWM を OFF にしています。PB5 には、サーボが接続されています。そのため、<b>ITU4_BRB へ値を設定する</b>ということは、<b>サーボの ON 幅を設定しているということになります</b>。                  最初、サーボは 0 度にしたいため、0 度になるよう PWM 幅を設定します。                  その PWM 幅は、</p> <pre style="border: 1px solid black; padding: 2px;">36 : #define    SERVO_CENTER    5000    /* サーボのセンタ値    */</pre> <p>と定義しています。そのため、設定は、</p> <pre style="border: 1px solid black; padding: 2px;">521 : ITU4_GRB = ITU4_BRB = SERVO_CENTER; /* サーボの PWM 設定    */</pre> <p>とします。</p>																																				
<p>ITU_TOER</p>	<p>PWM 波形を出力するかしないか選択します。                  "1": PWM 波形出力 "0": 通常の I/O ポートとして使用</p> <table border="1" data-bbox="395 752 1433 853"> <tr> <td>bit</td> <td>7</td> <td>6</td> <td>5</td> <td>4</td> <td>3</td> <td>2</td> <td>1</td> <td>0</td> </tr> <tr> <td></td> <td>0 固定</td> <td>0 固定</td> <td>PB5</td> <td>PB4</td> <td>PB1</td> <td>PB3</td> <td>PB2</td> <td>PB0</td> </tr> </table> <p>例) PB5, PB4, PB1 を PWM として使用、他は I/O ポートなら                  PB5, PB4, PB1 の部分を "1" に、PB4, PB2, PB0 の部分を "0" にします。</p> <table border="1" data-bbox="395 952 1433 1052"> <tr> <td>bit</td> <td>7</td> <td>6</td> <td>5</td> <td>4</td> <td>3</td> <td>2</td> <td>1</td> <td>0</td> </tr> <tr> <td>値</td> <td>0</td> <td>0</td> <td>1</td> <td>1</td> <td>1</td> <td>0</td> <td>0</td> <td>0</td> </tr> </table> <p>0011 1000 → <b>0x38</b> を設定します。  <pre style="border: 1px solid black; padding: 2px;">522 : ITU_TOER = 0x38;</pre></p>	bit	7	6	5	4	3	2	1	0		0 固定	0 固定	PB5	PB4	PB1	PB3	PB2	PB0	bit	7	6	5	4	3	2	1	0	値	0	0	1	1	1	0	0	0
bit	7	6	5	4	3	2	1	0																													
	0 固定	0 固定	PB5	PB4	PB1	PB3	PB2	PB0																													
bit	7	6	5	4	3	2	1	0																													
値	0	0	1	1	1	0	0	0																													
<p>ITU_STR</p>	<p>それぞれの ITU のチャンネルで ITU_CNT をカウント動作させるかどうか選択します。要は、ITU を使うか使わないかの設定です。                  "1": 使用する "0": 使用しない</p> <table border="1" data-bbox="395 1294 1433 1395"> <tr> <td>bit</td> <td>7</td> <td>6</td> <td>5</td> <td>4</td> <td>3</td> <td>2</td> <td>1</td> <td>0</td> </tr> <tr> <td></td> <td>0 固定</td> <td>0 固定</td> <td>0 固定</td> <td>ITU4</td> <td>ITU3</td> <td>ITU2</td> <td>ITU1</td> <td>ITU0</td> </tr> </table> <p>リセット同期 PWM モードでは ITU3 と 4 を使用します。しかしカウンタは ITU3 のみしか使用しません。また、割り込みで ITU0 を使用しています。結果、ITU3 と ITU0 のカウンタ動作をさせ、他はカウントさせません。</p> <table border="1" data-bbox="395 1529 1433 1630"> <tr> <td>bit</td> <td>7</td> <td>6</td> <td>5</td> <td>4</td> <td>3</td> <td>2</td> <td>1</td> <td>0</td> </tr> <tr> <td>値</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>1</td> <td>0</td> <td>0</td> <td>1</td> </tr> </table> <p>0000 1001 → <b>0x09</b> を設定します。  <pre style="border: 1px solid black; padding: 2px;">525 : ITU_STR = 0x09;</pre></p>	bit	7	6	5	4	3	2	1	0		0 固定	0 固定	0 固定	ITU4	ITU3	ITU2	ITU1	ITU0	bit	7	6	5	4	3	2	1	0	値	0	0	0	0	1	0	0	1
bit	7	6	5	4	3	2	1	0																													
	0 固定	0 固定	0 固定	ITU4	ITU3	ITU2	ITU1	ITU0																													
bit	7	6	5	4	3	2	1	0																													
値	0	0	0	0	1	0	0	1																													

## 8.11 時間稼ぎ:timer関数

この関数を実行すると時間稼ぎをして、他は何もしません。使い方としては、timer 関数の引数にミリ秒単位で数値を入れます。

```

539 : /*****/
540 : /* タイマ本体 */
541 : /* 引数 タイマ値 1=1ms */
542 : /*****/
543 : void timer( unsigned long timer_set )
544 : {
545 :     cnt0 = 0;
546 :     while( cnt0 < timer_set );
547 : }
    
```

例えば、下記のように timer 関数を実行します。

```

timer( 1000 );           この行で1000ms の時間稼ぎをする
    
```

引数は、1000 です。要は、timer\_set が 1000 になります。545 行で cnt0 を 0 にしているので

```

546 :     while( 0 < 1000 );
    
```

となります。これではカッコの内が常に成り立ってしまい、次に進むことは無いように思えます。

cnt0 を操作している場所を思い出してみます。そうです。cnt0 は 1ms ごとに割り込みが発生して実行される interrupt\_timer0 関数内で+1しています。そのため、1ms 後には、

```

546 :     while( 1 < 1000 );
    
```

となります。どんどん時間がたっていき、きっかり 1000ms 後に、

```

546 :     while( 1000 < 1000 );           ←成り立たなくなる！
    
```

となり、カッコは偽(成り立たない)と判断され、次の行へいきます。結果、546 行で 1000ms 待つ、関数の名前とおり、タイマの役割をします。

例えば、10 秒の時間稼ぎをしたいなら、10 秒=10000ms なので、

```

timer( 10000 );
    
```

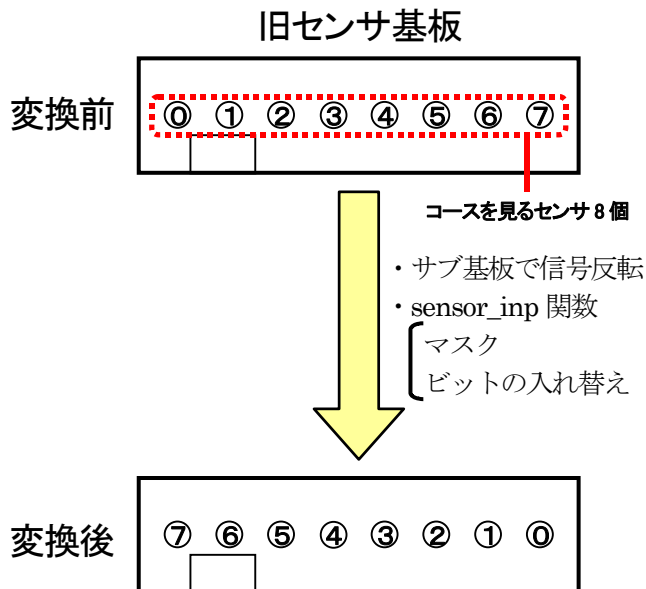
となります。

ただし、timer 関数は、「**何もせずに指定時間待つ**」関数です。マイコンカーの場合は長い時間センサを見ずに時間稼ぎをしていたら、脱輪してしまいます。そのため、マイコンカーのプログラムでは timer 関数は使わずに別な方法で時間を計ります。詳しくは後述します。

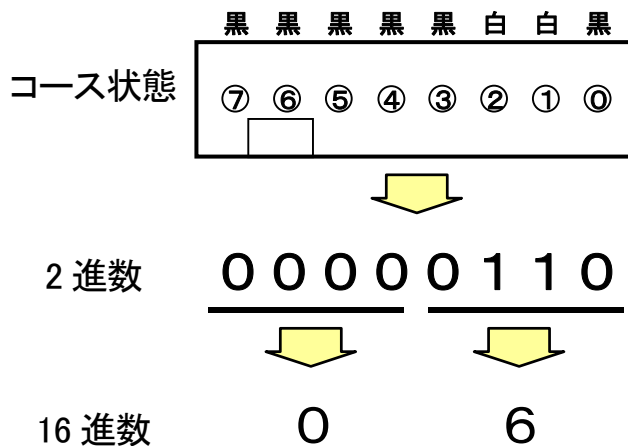
## 8.12 コースのセンサ状態読み込み: sensor\_inp関数(センサ基板Ver.4 仕様)

### 8.12.1 センサ基板 4 以前の信号変換

センサ基板 4 以前のセンサ基板(以後、旧センサ基板といいます)は、コースを見るセンサが 8 個ありました(下図)。サブ基板や sensor\_inp 関数で信号を変換して、変換後のビット配列にしました。

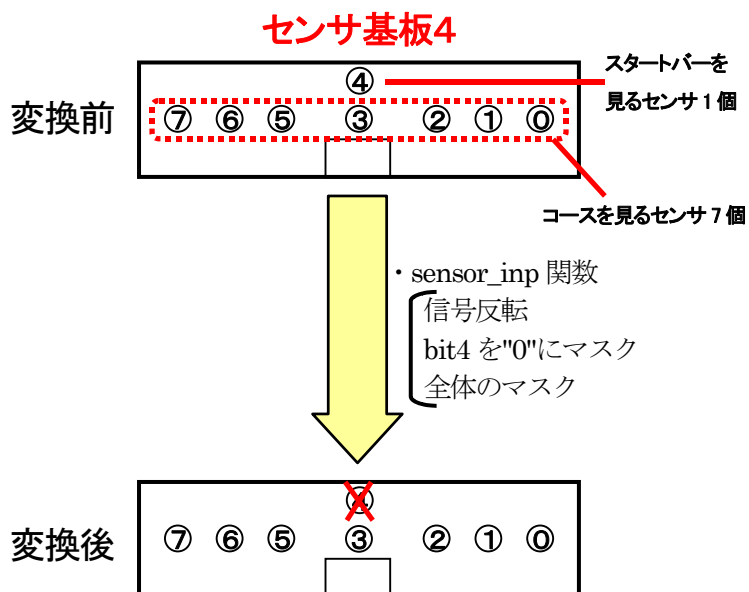


例えば、コースの状態が「黒黒黒黒 黒白白黒」になっていたとします。16 進数データは下記のようになります。

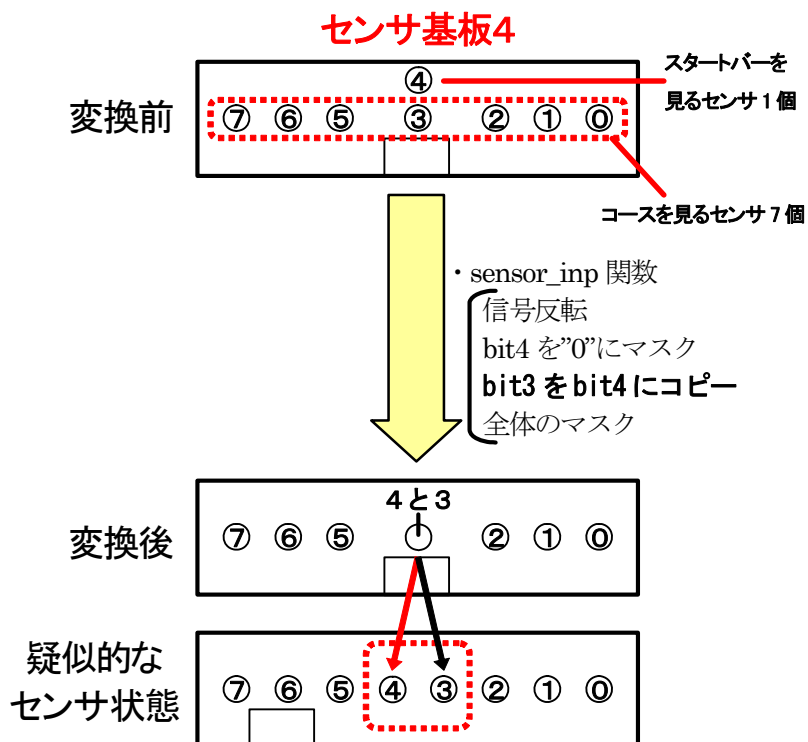


### 8.12.2 センサ基板 4 の信号変換

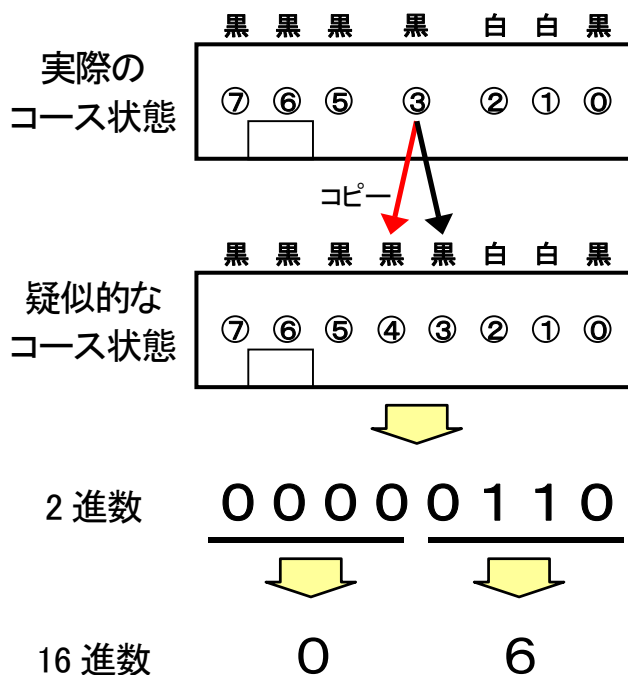
センサ基板 Ver.4 は、コースを見るセンサが7個、スタートバーを見るセンサが1個あります(下図)。sensor\_inp 関数はコースの状態を見る関数なので、スタートバーを見るセンサのビットを強制的に"0"にします。



結果、コースを見るセンサが7個となります。ただし、旧センサ基板はコースを見るセンサが8個でした。そのため、プログラムに互換性が無くなり大幅に変更しなければいけません。さらに16進数は、2進数を4桁ごとに区切って変換します。7bitだと、非常に変換しづらいです。そのため、次の図のように変換することにしました。



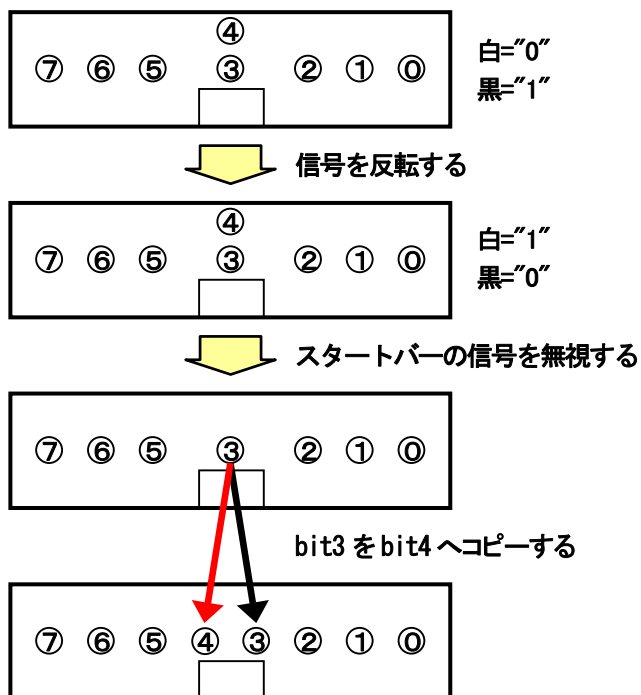
例えば、コース状態が「黒黒黒 黒 白白黒」になっていたとします。bit3の値をbit4へコピーしていますので、**擬似的にセンサ信号は「黒黒黒黒 黒白白黒」となります。**16進数データは下記のようになります。



16進数は、旧センサ基板と同様の値となりました。メインプログラムでは、「**センサの値は bit4 と bit3 が同じ値になる**」ということだけ気をつけてプログラムします。マイコンカーの旧センサ基板を、**センサ基板 4 に交換したときは、センサ信号の bit4 と bit3 の値が同じになるということに気をつけてプログラムを見直せば、変更は少なくなります。**

### 8.12.3 プログラム

今までの考え方をまとめると、次のようになります。



まず、センサ信号を読み込み、反転します。センサはポート 7 に繋がっていますので、P7DR から読み込み、「~」(チルダ)で反転します。

```
sensor = ~P7DR;
```

次に、スタートバーの信号が入力される bit4 を強制的に“0”にして無視します。2 進数で「1110 1111」なので、16 進数に直すと「0xef」の値で AND 処理します。

```
sensor &= 0xef;
```

次に、bit3 の値が“1”なら bit4 の値も“1”にします。先の処理で bit4 は“0”ということが分かっているので、bit3 が“0”なら何もする必要はありません。

```
if( sensor & 0x08 ) sensor |= 0x10;
```

まとめると、sensor\_inp 関数は下記のようにになります。

```
unsigned char sensor_inp( void )
{
    unsigned char sensor;

    sensor = ~P7DR;
    sensor &= 0xef;
    if( sensor & 0x08 ) sensor |= 0x10;
}
```

※チルダは、キーボードの☒キー左横の☒キーを、シフトを押しながら押すと☒になります。

## 8.12.4 マスク

### (a) マスクとは？

562 行で sensor 変数の値と mask 変数の値を AND 演算しています。

562 :	sensor &= mask;	この命令は、	<code>sensor = sensor &amp; mask;</code>	と同じです。
-------	-----------------	--------	--	--------

**チェックに不要なビットを強制的に“0”にすること「マスク処理」といいます。**マスクは制御で非常に重要です。マイコンカー制御でも頻繁に使用します。

### (b) 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 ビットまとめた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、

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

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

センサ値=1000 0001 (2 進数)=0x81 (16 進数)

となります。プログラムで 0x80 かどうかチェックしただけでは bit7 が“1”かどうか判断できません。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。

### (c) マスク処理を行うと

先ほどの方法では、bit7 が“1”かどうか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(“0”なら値の変化はありません)。

<pre>if( bit6~bit0 を“0”にしたセンサの値==0x80 ) {     /* bit7 が “1” ならこの中を実行 */ }</pre>	<div style="border: 1px solid black; padding: 2px; display: inline-block;">bit7=“1”なら 0x80、bit7=“0”なら 0x00 になる</div>
---	--

(d) マスク処理を行うプログラム

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。AND 演算とは、2 つの変数 A と B があるとき(それぞれ 0 か 1 の数値)、ともに 1 であるときのみ 1 になる演算を言います(下表)。A をセンサの値として考えてみます。

A(センサ値)	B	A and B
0	0	0
0	1	0
1	0	0
1	1	1

ここで、Bが0のときに注目します。Bが0ならA(センサ値)がどの値でも結果は必ず0になります(下表)。

A(センサ値)	B	A and B
0	0	0
1	0	0

次に、Bが1のときに注目します。Bが1なら、結果はA(センサ値)の値そのものとなります(下表)。

A(センサ値)	B	A and B
0	1	0
1	1	1

実際に置き換えると、Aがセンサの値、**Bがマスクをする値にあたります(以後、マスク値)**。マスク値は、必要なビットは"1"に、必要のないビットは"0"にします。そして AND 演算を行うと不必要なビットは必ず"0"になるので、**プログラムでは必要ないビットは"0"ということを前提にして作成することができます。**

(e) 例

例えば、センサの状態が「黒黒黒白白白白」のとき、センサ値は「00011111」です。bit2、bit1 のみチェックに必要で、他は必要なしとします。

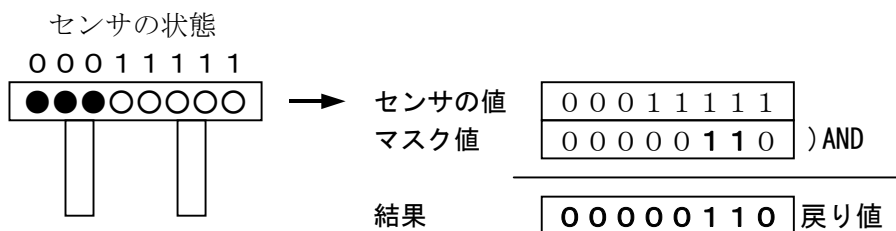
bit	7	6	5	4	3	2	1	0
	不要	不要	不要	不要	不要	<b>必要</b>	<b>必要</b>	不要

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	0	0	0	0	0	<b>1</b>	<b>1</b>	0



2進数で「00000110」、16進数に直すと「0x06」となります。下表はその計算方法と結果です。



例えば、bit2="1"、bit1="0"かどうかチェックしたい場合、下記のようになります。

```
if( (センサの値 & 0x06) == 0x04 ) {
    /* bit2="1"、bit1="0"ならこの中を実行 */
}
```

bit2、bit1 以外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、安心して bit2、bit1 のみ調べることができます。

(f) マスク値を引数にした sensor\_inp 関数

センサの値をマスクすることはよくあります。逆にセンサの値をマスクせずにそのまま8ビット分使うことの方が少ないくらいです。そのためマスク処理しやすいように、sensor\_inp 関数の引数にマスク値を入れられるよう改造します。この関数が実際の sensor\_inp 関数になります。

```
554 : unsigned char sensor_inp( unsigned char mask )
555 : {
556 :     unsigned char sensor;
557 :
558 :     sensor = ~P7DR;
559 :     sensor &= 0xef;
560 :     if( sensor & 0x08 ) sensor |= 0x10;
561 :
562 :     sensor &= mask;
563 :
564 :     return sensor;
565 : }
```

センサ値を加工した一番最後でマスク処理する

(g) センサのマスクパターン

sensor\_inp 関数の引数にマスク値を入れるようにしました。実際、マイコンカーのプログラムを作っていく上で、マスク値は限られてきます。それをあらかじめ定義しておきます。

```

39 : /* マスク値設定 × : マスクあり(無効) ○ : マスク無し(有効) */
40 : #define MASK2_2 0x66 /* ×○○××○○× */
41 : #define MASK2_0 0x60 /* ×○○××××× */
42 : #define MASK0_2 0x06 /* ×××××○○× */
43 : #define MASK3_3 0xe7 /* ○○○××○○○ */
44 : #define MASK0_3 0x07 /* ×××××○○○ */
45 : #define MASK3_0 0xe0 /* ○○○××××× */
46 : #define MASK4_0 0xf0 /* ○○○○×××× */
47 : #define MASK0_4 0x0f /* ××××○○○○ */
48 : #define MASK4_4 0xff /* ○○○○○○○○ */

```

「MASK○\_◎」として、左のセンサ○個、右のセンサ◎個を残して他をマスクする、という意味です。先ほどの bit2、bit1 をチェックするマスクパターンは、上表より「MASK0\_2」なので、

```

if( sensor_inp( MASK0_2 ) == 0x04 ) {
    /* bit2="1"、bit1="0"ならこの中を実行 */
}

```

と書き換えることができます。

8.12.5 まとめ

下記プログラムの太字部分が、kit06.c から kit07.c に変更するに当たり、変わった部分です。旧センサ基板をセンサ基板 4 に変更したときは、sensor\_inp 関数を入れ替えてください。








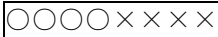
```

549 : /******
550 : /* センサ状態検出 */
551 : /* 引数 マスク値 */
552 : /* 戻り値 センサ値 */
553 : /******
554 : unsigned char sensor_inp( unsigned char mask )
555 : {
556 :     unsigned char sensor;
557 :
558 :     sensor = ~P7DR;
559 :     sensor &= 0xef;
560 :     if( sensor & 0x08 ) sensor |= 0x10;
561 :
562 :     sensor &= mask;
563 :
564 :     return sensor;
565 : }

```

### 8.12.6 注意点

sensor\_inp 関数の値は bit4 と bit3 は同じと言うことを説明しました。ただし、マスク値によっては sensor\_inp 関数の戻り値の bit4 と bit3 が異なる場合があります。sensor\_inp 関数のマスク値にも気をつけてプログラムしてください。

if( sensor_inp(MASK4_4) == 0x1f ) { }		<b>あり得る</b>
if( sensor_inp(MASK4_4) == 0x07 ) { }		<b>あり得る</b>
if( sensor_inp(MASK4_4) == 0x0f ) { }		<b>0x0f はあり得ない</b>
if( sensor_inp(MASK4_4) == 0xf8 ) { }		<b>あり得る</b>
if( sensor_inp(MASK4_4) == 0xe0 ) { }		<b>あり得る</b>
if( sensor_inp(MASK4_4) == 0xf0 ) { }		<b>0xf0はあり得ない</b>
if( sensor_inp(MASKO_4) == 0x0f ) { }		<b>マスク値によっては 0x0f はあり得る！</b>
if( sensor_inp(MASK4_0) == 0xf0 ) { }		<b>マスク値によっては 0xf0はあり得る！</b>

### 8.13 クロスライン検出処理: check\_crossline関数(kit07.cで変更)

クランクの 50cm~100cm 手前に 2 本の横線があります。これをクロスラインと呼びます。このクロスラインの検出を行う専用の関数を作りました。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。太字部分が、kit06.c から kit07.c に変更するに当たり、変わった部分です。

```

567 : /*****
568 : /* クロスライン検出処理
569 : /* 戻り値 0:クロスラインなし 1:あり
570 : *****/
571 : int check_crossline( void )
572 : {
573 :     unsigned char b;
574 :     int ret;
575 :
576 :     ret = 0;
577 :     b = sensor_inp(MASK3_3);
578 :     if( b==0xe7 ) {
579 :         ret = 1;
580 :     }
581 :     return ret;
582 : }

```

参考: kit06.c のプログラム

```

ret = 0;
b = sensor_inp(MASK2_2);
if( b==0x66 || b==0x64 || b==0x26 || b==0x62 || b==0x46 ) {
    ret = 1;
}

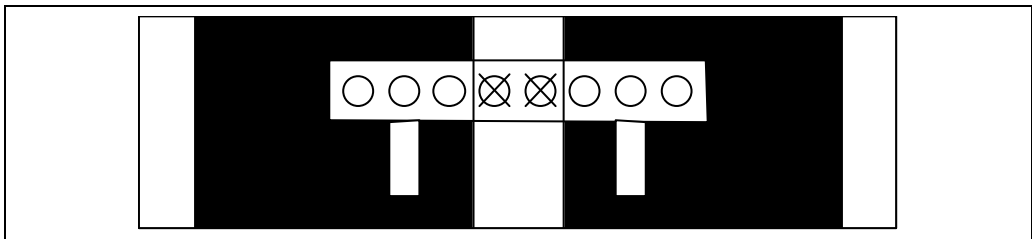
```

```
576 :     ret = 0;
```

戻り値を保存する ret 変数を初期化しています。クロスラインと判断すると 1、違うと判断すると 0 をこの変数に入れます。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。

```
577 :     b = sensor_inp(MASK3_3);
```

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、



のように、左 3 個、右 3 個の合計 6 個のセンサを読み込みます。

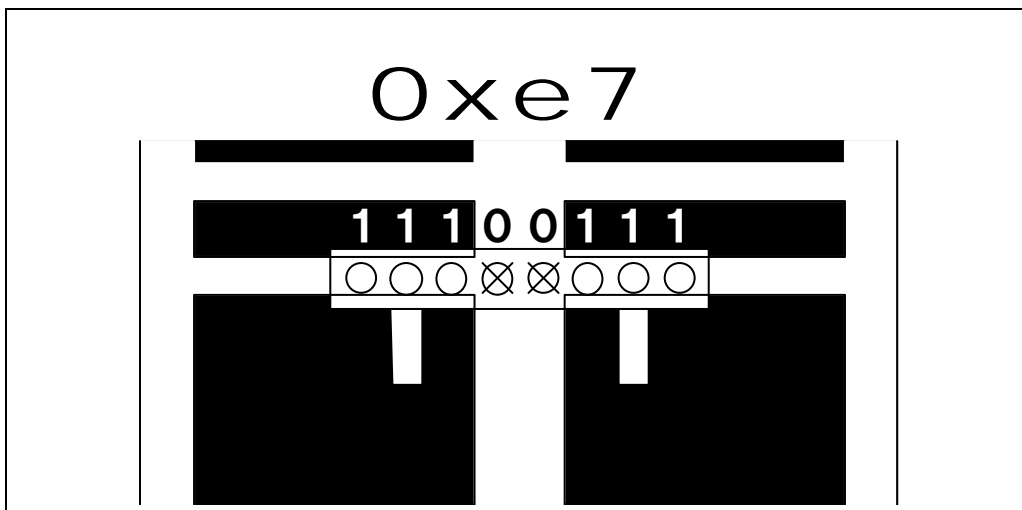
```

578 :         if( b==0xe7 ) {
579 :             ret = 1;
580 :         }

```

センサの状態をチェックします。センサが、0xe7 なら if の条件が成立するので ret 変数は 1、違うなら不成立なので 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、成立=クロスラインあり、不成立=クロスラインなし、ということになります。



### 8.14 右ハーフライン検出処理: check\_rightline関数(kit07.cで変更)

右レーンチェンジの50cm~120cm手前に2本の右ハーフラインがあります。右ハーフラインの検出を行う専用の関数を作りました。戻り値は、右ハーフラインと判断すると"1"、でなければ"0"とします。太字部分が、kit06.cからkit07.cに変更するに当たり、変わった部分です。

```

584 : /*****
585 : /* 右ハーフライン検出処理
586 : /* 戻り値 0:なし 1:あり
587 : /*****
588 : int check_rightline( void )
589 : {
590 :     unsigned char b;
591 :     int ret;
592 :
593 :     ret = 0;
594 :     b = sensor_inp(MASK4_4);
595 :     if( b==0x1f ) {
596 :         ret = 1;
597 :     }
598 :     return ret;
599 : }
    
```

参考: kit06.c のプログラム

```

ret = 0;
b = sensor_inp(MASK4_4);
if( b==0x0f || b==0x1f ) {
    ret = 1;
}
    
```

```

593 :     ret = 0;
    
```

戻り値を保存するret変数を初期化しています。右ハーフラインと判断すると1、違うと判断すると0をこの変数に入れます。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。

```

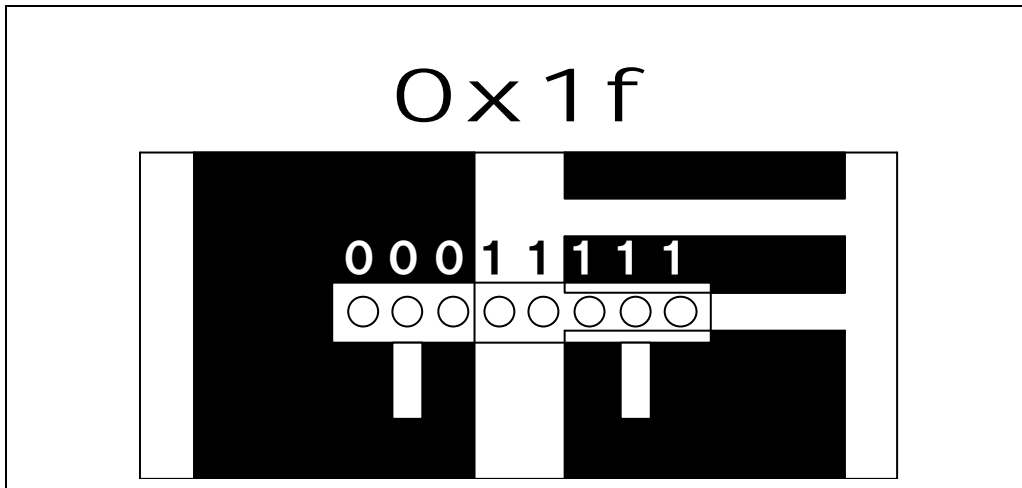
594 :     b = sensor_inp(MASK4_4);
    
```

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、「MASK4\_4=0xff」なので、8個のセンサすべて読み込みます。

```
595 :     if( b==0x1f ) {
596 :         ret = 1;
597 :     }
```

センサの状態をチェックします。センサが、0x1f なら if の条件が成立するので ret 変数は 1、違うなら不成立なので 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、成立 = 右ハーフラインあり、不成立 = 右ハーフラインなし、ということになります。



右 5 つが "1" なら右ハーフラインと判断します。

センサ基板 4 は、中心 1 個のセンサを擬似的にプログラムで 2 個にしています。そのため、中心 2 個 (bit4 と bit3) のセンサ値は必ず同じ値になります。**センサ値が "0000 1111" になることは、ありません。**プログラムでは注意してください。

### 8.15 左ハーフライン検出処理 : check\_leftline 関数 (kit07.c で変更)

左レーンチェンジの 50cm ~ 120cm 手前に 2 本の左ハーフラインがあります。左ハーフラインの検出を行う専用の関数を作りました。戻り値は、左ハーフラインと判断すると "1"、でなければ "0" とします。

```
601 :  /*****
602 :  /* 左ハーフライン検出処理
603 :  /* 戻り値 0:なし 1:あり
604 :  *****/
605 :  int check_leftline( void )
606 :  {
607 :      unsigned char b;
608 :      int ret;
609 :
610 :      ret = 0;
611 :      b = sensor_inp(MASK4_4);
612 :      if( b==0xf8 ) {
613 :          ret = 1;
614 :      }
615 :      return ret;
616 :  }
```

参考: kit06.c のプログラム

```
ret = 0;
b = sensor_inp(MASK4_4);
if( b==0xf0 || b==0xf8 ) {
    ret = 1;
}
```

```
610 :     ret = 0;
```

戻り値を保存する ret 変数を初期化しています。左ハーフラインと判断すると1、違うと判断すると0をこの変数に入れます。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。

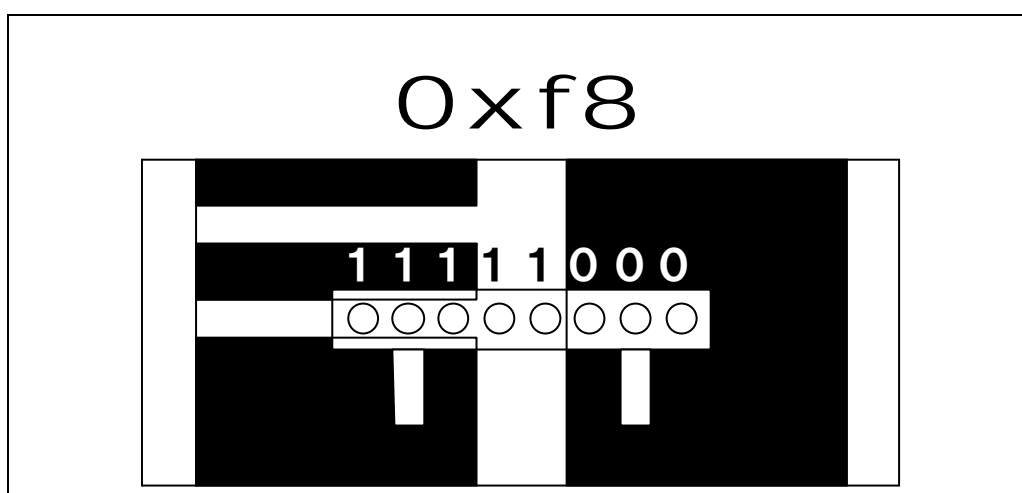
```
611 :     b = sensor_inp(MASK4_4);
```

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、「MASK4\_4=0xff」なので、8 個のセンサすべて読み込みます。

```
612 :     if( b==0xf8 ) {
613 :         ret = 1;
614 :     }
```

センサの状態をチェックします。センサが、0xf8 なら if の条件が成立するので ret 変数は1、違うなら不成立なので 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、成立=左ハーフラインあり、不成立=左ハーフラインなし、ということになります。



左5個が“1”なら左ハーフラインと判断します。

センサ基板4は、中心1個のセンサを擬似的にプログラムで2個にしています。そのため、中心2個(bit4とbit3)のセンサ値は必ず同じ値になります。**センサ値が“1111 0000”になることは、ありません。**プログラムでは注意してください。

## 8.16 ディップスイッチの読み込み:dipsw\_get関数

CPU ボードにある 4 ビットのディップスイッチの値を読み込む関数です。

```

618 : /*****/
619 : /* ディップスイッチ値読み込み */
620 : /* 戻り値 スイッチ値 0~15 */
621 : /*****/
622 : unsigned char dipsw_get( void )
623 : {
624 :     unsigned char sw;
625 :
626 :     sw = ~P6DR;          /* ディップスイッチ読み込み */
627 :     sw &= 0x0f;
628 :
629 :     return sw;
630 : }
    
```

```

626 :     sw = ~P6DR;          /* ディップスイッチ読み込み */
    
```

ディップスイッチのあるポート 6 からデータを読み込みます。ディップスイッチは、ポートから読み込む値は OFF で”1”、ON で”0”です。ただ、感覚的には OFF が”0”、ON が”1”の方が分かりやすいので、「~」(チルダ)にて反転させています。

```

627 :     sw &= 0x0f;
    
```

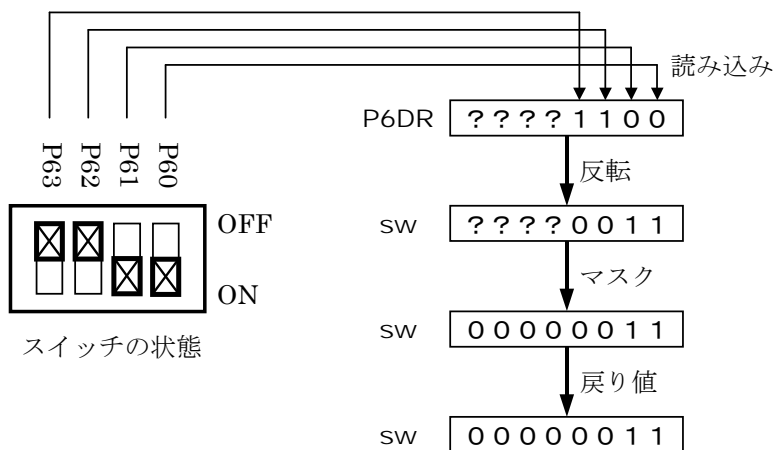
ディップスイッチは、bit3~0 のみなので、bit7~4 部分はマスクして強制的に”0”にします。

例えば、ディップスイッチが、OFF、OFF、ON、ON で

```

c = dipsw_get();
    
```

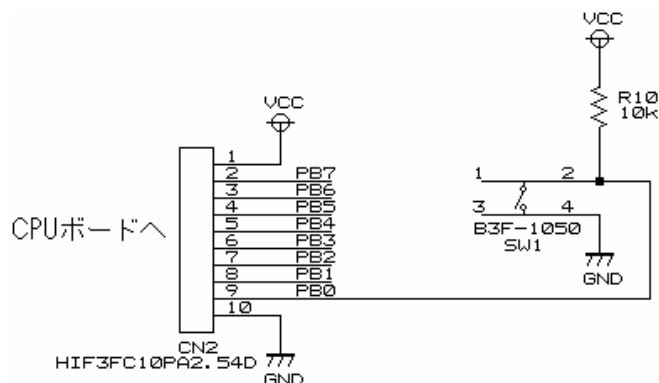
と関数を呼んだとします。この場合、下図のようなイメージになります。変数 c には、0x03 が入ります。





## 8.17 プッシュスイッチの読み込み: pushsw\_get関数

モータドライブ基板のプッシュスイッチの状態を読み込む関数です。プッシュスイッチ回路は下記のようになっています。



プッシュスイッチは、ポート B の bit0 に繋がっています。回路的にはスイッチ ON で“0”、OFF で“1”が入力されます。この関数では、スイッチ ON で 1、OFF で 0 が戻り値になるようにします。

```

632 : /*****/
633 : /* プッシュスイッチ値読み込み */
634 : /* 戻り値 スイッチ値 ON:1 OFF:0 */
635 : /*****/
636 : unsigned char pushsw_get( void )
637 : {
638 :     unsigned char sw;
639 :
640 :     sw = ~PBDR; /* スイッチのあるポート読み込み */
641 :     sw &= 0x01;
642 :
643 :     return sw;
644 : }
    
```

```

640 :     sw = ~PBDR; /* スイッチのあるポート読み込み */
    
```

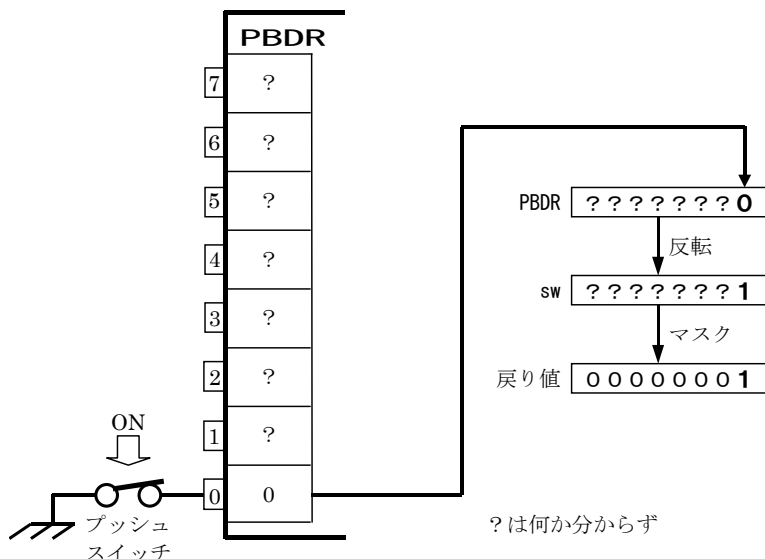
プッシュスイッチのあるポート B からデータを読み込みます。プッシュスイッチ OFF で“1”、ON で“0”となっているので、「~」にて反転させています。「~」は、チルダと読み、キーボードの「」キー左横の「」キーを、シフトを押しながら押すと「」になります。

```

641 :     sw &= 0x01;
    
```

プッシュスイッチは、bit0 のみなので、bit7～1 部分はマスクして強制的に“0”にします。マスク値は、2 進数で‘0000 0001’、16 進数で 0x01 になります。

スイッチを押した例を下図に示します。



### 8.18 スタートバー検出センサ読み込み: startbar\_get関数(kit07.cで変更)

センサ基板4からのスタートバー検出信号を、読み込む関数です。スタートバー信号は、ポート7のbit4から読み込みます。**戻り値は、スタートバーありで1、なしで0が返ってきます。**

```

646 : /*****
647 : /* スタートバー検出センサ読み込み */
648 : /* 戻り値 センサ値 ON(バーあり):1 OFF(なし):0 */
649 : /*****
650 : unsigned char startbar_get( void )
651 : {
652 :     unsigned char b;
653 :
654 :     b = ~P7DR;          /* スタートバー信号読み込み */
655 :     b &= 0x10;
656 :     b >>= 4;
657 :
658 :     return b;
659 : }
    
```

```

654 :     b = ~P7DR;          /* スタートバー信号読み込み */
    
```

センサ基板が接続されているポート7からデータを読み込みます。スタートバーありで"0"、なしで"1"となっているので、「~」にて反転させています。「~」は、チルダと読み、キーボードの「」キー左横の「」キーを、シフトを押しながら押すと「」になります。

```

655 :     b &= 0x10;
    
```

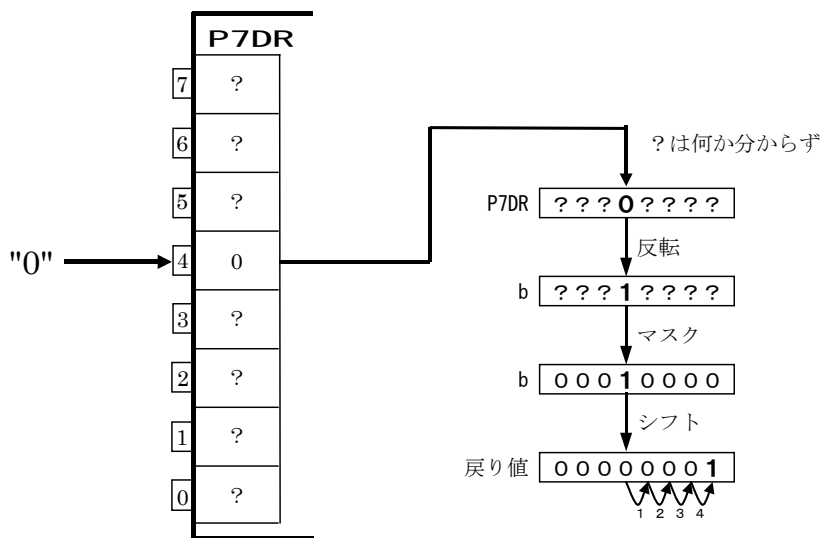
スタートバー検出センサの信号は、bit4のみなので、それ以外のビットはマスクして強制的に"0"にします。マスク値は、2進数で'0001 0000'、16進数で0x10になります。

```

656 :     b >>= 4;
    
```

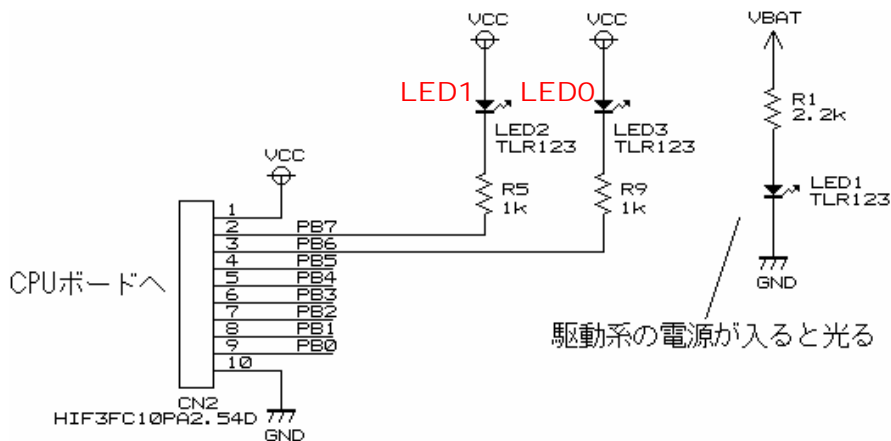
bit4にある信号値をbit0に移動させるため、4ビット右へ移動します。結果、スタートバーありで1、なしで0になります。

スタートバー検出センサから"0"信号(スタートバーあり)を入力したときの様子を下図に示します。



### 8.19 LEDの制御:led\_out関数

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。



ポートBのbit7とbit6に繋がれています。bit7に繋がれているLEDがプログラムで言う「LED1」、bit6に繋がれているLEDがプログラムで言う「LED0」です。

この関数では、引数とLEDの点灯の仕方を下記の表のようにします。

引数	2進数では	LED1	LED0
0	00	消灯	消灯
1	01	消灯	点灯
2	10	点灯	消灯
3	11	点灯	点灯

引数を2進数にしたとき、1桁目をLED0、2桁目をLED1の制御用にして"0"で消灯、"1"で点灯にします。

```

661 : /*****
662 : /* LED 制御
663 : /* 引数 スイッチ値 LED0:bit0 LED1:bit1 "0":消灯 "1":点灯
664 : /* 例)0x3→LED1:ON LED0:ON 0x2→LED1:ON LED0:OFF
665 : *****/
666 : void led_out( unsigned char led )
667 : {
668 :     unsigned char data;
669 :
670 :     led = ~led;
671 :     led <<= 6;
672 :     data = PBDR & 0x3f;
673 :     PBDR = data | led;
674 : }
    
```

```

670 :     led = ~led;
    
```

引数は”1”が点灯、”0”が消灯ですが、実際の LED は、”0”で点灯、”1”で消灯です。そのため、反転させて論理が合うようにしています。

```

671 :     led <<= 6;
    
```

引数は bit ト 1 と bit0 の値ですが、実際の LED は bit7 と bit6 に有るので、左シフトして移動します。

```

672 :     data = PBDR & 0x3f;
    
```

ポート B の現在の出力値を読み込み、LED 出力である bit7,6 をマスクして強制的に”0”にしています。

```

673 :     PBDR = data | led;
    
```

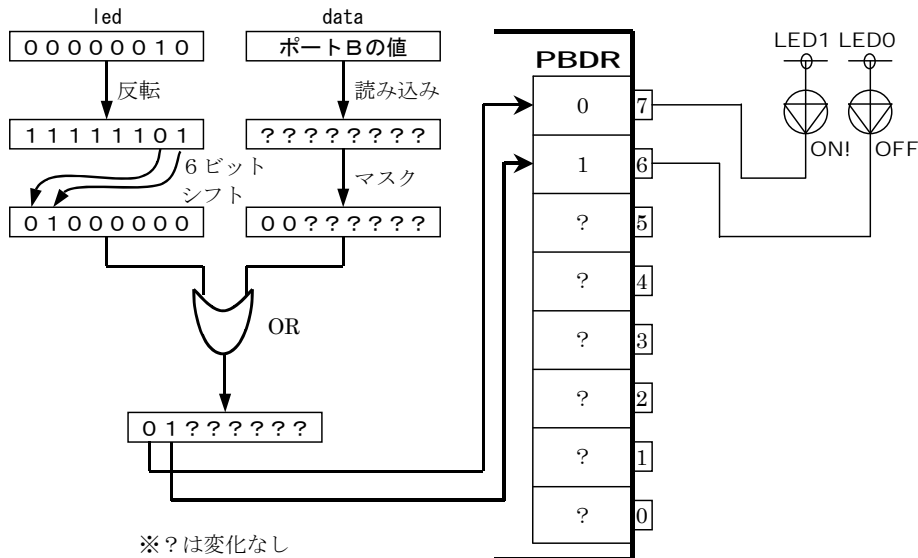
新たに、今回制御したい LED データをポートBに書き込みます。

例えば、

```

led_out ( 0x02 );
    
```

と関数を実行したとします。この場合、下図のようなイメージになります。LED1 が点灯、LED0 は消灯します。



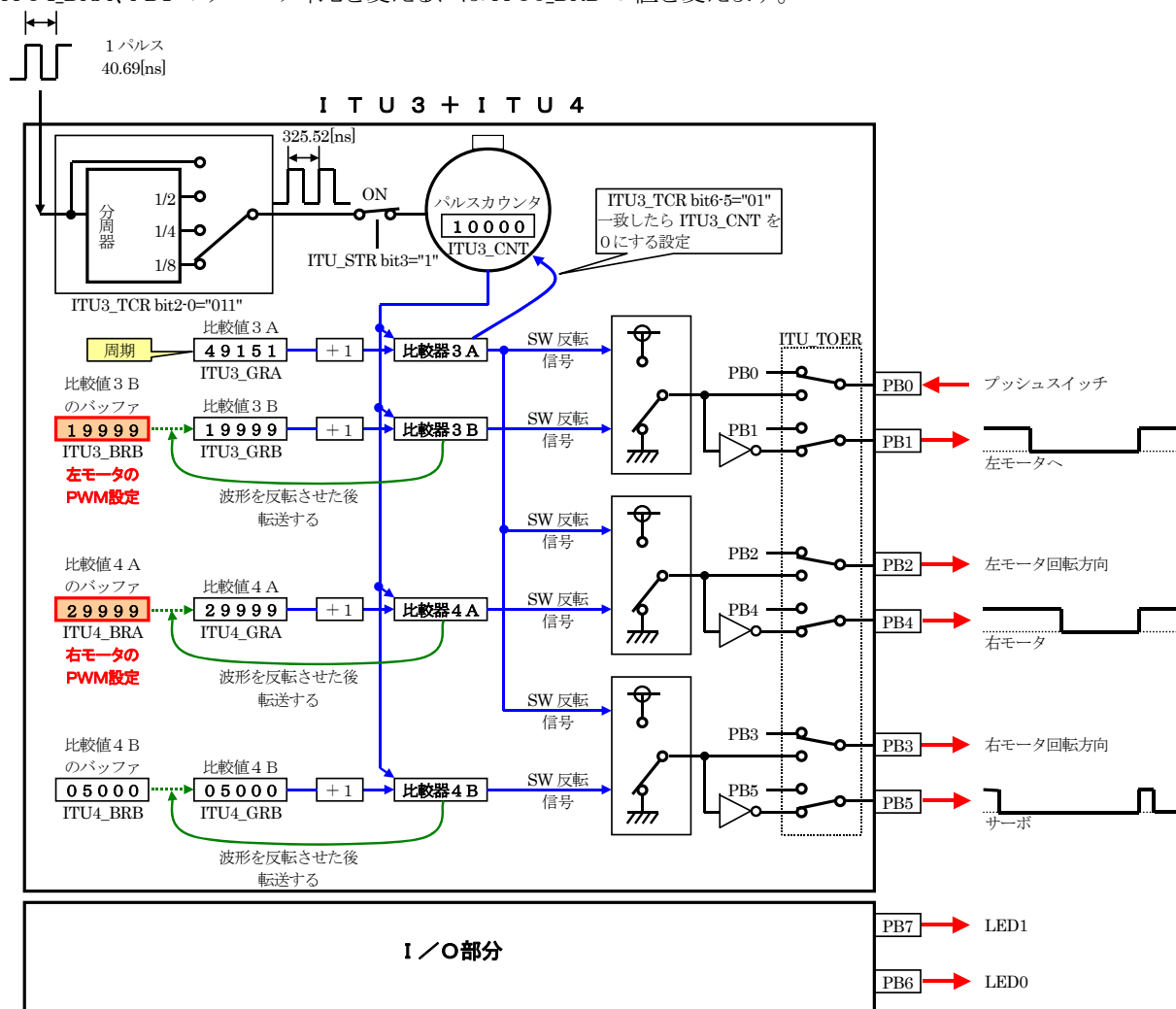
## 8.20 モータ速度制御: speed関数

左モータ、右モータを制御する関数です。引数は100～-100まで設定でき、100で正転100%、-100で逆転100%、0でブレーキとなります。

モータドライブ基板の接続をもう一度確認しておきます。

ピン番	信号、方向※	詳細	“0”	“1”	詳細
1	—	+5V			
2	基板←PB7	LED1	点灯	消灯	
3	基板←PB6	LED0	点灯	消灯	
4	基板←PB5	サーボ信号	PWM 信号		ITU4_BRB でデューティ比設定
5	基板←PB4	右モータ PWM	停止	動作	ITU4_BRA でデューティ比設定
6	基板←PB3	右モータ回転方向	正転	逆転	
7	基板←PB2	左モータ回転方向	正転	逆転	
8	基板←PB1	左モータ PWM	停止	動作	ITU3_BRB でデューティ比設定
9	基板→PB0	プッシュスイッチ	押された	押されていない	
10	—	GND			

表のとおり、PB4 が右モータ PWM、PB1 が左モータ PWM、PB3 が右モータの回転方向制御、PB2 が左モータの回転方向制御です。リセット同期 PWM モードを使用していますので PB4 のデューティ比を変えるには ITU4\_BRA、PB1 のデューティ比を変えるには ITU3\_BRB の値を変えます。



```

676 : /*****
677 : /* 速度制御
678 : /* 引数 左モータ:-100~100 , 右モータ:-100~100
679 : /*      0で停止、100で正転100%、-100で逆転100%
680 : *****/
681 : void speed( int accele_l, int accele_r )
682 : {
683 :     unsigned char    sw_data;
684 :     unsigned long    speed_max;
685 :
686 :     sw_data = dipsw_get() + 5;          /* デイップスイッチ読み込み */
687 :     speed_max = (unsigned long)(PWM_CYCLE-1) * sw_data / 20;
688 :
689 :     /* 左モータ */
690 :     if( accele_l >= 0 ) {
691 :         PBDR &= 0xfb;
692 :         ITU3_BRB = speed_max * accele_l / 100;
693 :     } else {
694 :         PBDR |= 0x04;
695 :         accele_l = -accele_l;
696 :         ITU3_BRB = speed_max * accele_l / 100;
697 :     }
698 :
699 :     /* 右モータ */
700 :     if( accele_r >= 0 ) {
701 :         PBDR &= 0xf7;
702 :         ITU4_BRA = speed_max * accele_r / 100;
703 :     } else {
704 :         PBDR |= 0x08;
705 :         accele_r = -accele_r;
706 :         ITU4_BRA = speed_max * accele_r / 100;
707 :     }
708 : }

```

```

686 :     sw_data = dipsw_get() + 5;          /* デイップスイッチ読み込み */

```

dipsw\_get 関数は、CPU ボードのデイップスイッチの値が返ってきます。0~15 です。+5 していますのでデイップスイッチの値に応じて  
 sw\_data = 5~20  
 になります。

```

687 :     speed_max = (unsigned long)(PWM_CYCLE-1) * sw_data / 20;

```

書き直すと、次のようになります。

$$\text{speed\_max} = \text{(unsigned long)} \quad \text{①} \quad \text{100\%のときの値} \times \frac{\text{sw\_data(5~20)}}{\text{20}}$$

① ②部分はPWMが 100%のときの値です。この値は、(PWM\_CYCLE-1)→(49151-1)→49150 となります。③の分子は最大 20 なので、49150×20=983000 となります。(unsigned int)型の最大値である 65535 の値を超えるので、正しい計算ができません。そのため、一時的にunsigned long型に変換します。計算は、常に一番大きい(小さい)値がどうなるかを計算します。型を超える場合は型変換して、必ず範囲を超えないようにします。  
 ②リセット同期PWMモードの設定で説明したとおり、100%はITU3\_GRA(周期)より1小さい値を設定します。PWM\_CYCLEがITU3\_GRAに設定している値なので、PWM\_CYCLEから1引いた値が 100%の値となります。

③ディップスイッチにより最大値の割合を変えます。割合は下記のようになります。

ディップスイッチ				10進数	計算	モータスピードの割合
P63	P62	P61	P60			
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%

結果、speed\_max にはディップスイッチの割合に応じたバッファレジスタに代入する値が入ります。

```

689 :      /* 左モータ */
690 :      if( accele_1 >= 0 ) {                0以上かチェック
691 :          PBDR &= 0xfb;                    正転にする
692 :          ITU3_BRB = speed_max * accele_1 / 100;  PWM値の設定
693 :      } else {

```

左モータ制御です。まず、speed 関数の左モータ PWM の引数である accele\_1 が 0 以上かチェックします。0 以上なら正転なので左モータ回転方向制御の bit2 を"0"にします。

bit	7	6	5	4	3	2	1	0	
PBDR 変更前	?	?	?	?	?	?	?	1	
AND する値	1	1	1	1	1	0	1	1	→ 0 x f b
PBDR 変更後	※	※	※	※	※	0	※	※	

※=変化無し

次に、デューティ比を設定します。設定は ITU3\_GRB ですが、プログラムで変更するのはバッファレジスタです。ITU3\_GRB のバッファレジスタは ITU3\_BRB となります。

$$\begin{aligned}
 \text{ITU3\_BRB} &= (\text{ディップスイッチの割合に応じたバッファレジスタに代入する値}) \times (\text{speed 関数の}\%) \div 100 \\
 &= \text{speed\_max} \times \text{accele\_1} \div 100
 \end{aligned}$$

となります。ポイントは、speed 関数で設定した割合がモータに出力されるのではなく、「**ディップスイッチの割合 × speed 関数で設定した割合**」がモータに出力されるということです。

```

690 :     if( accele_l >= 0 ) {
中略
693 :     } else {                               逆転の場合
694 :         PBDR |= 0x04;                       逆転にする
695 :         accele_l = -accele_l;                PWM値を正の数にする
696 :         ITU3_BRB = speed_max * accele_l / 100;  PWM値の設定
697 :     }

```

else 文以降です。この部分は、690 行の判定が当てはまらない場合にこの部分が実行されます。判定は、「accele\_l が 0 以上か」なので、else 文を実行するときは「**accele\_l が 0 以下**」のときになります。

逆転させるので左モータ回転方向制御の bit2 を“1”にします。bit2 のみを“1”にするには OR 演算を使います。

bit	7	6	5	4	3	2	1	0	
PBDR 変更前	?	?	?	?	?	?	?	1	
OR する値	0	0	0	0	0	1	0	0	→ 0 x 0 4
PBDR 変更後	※	※	※	※	※	1	※	※	

※=変化無し

次に、デューティ比を設定します。accele\_l はマイナスなので、計算前に 695 行で符号をプラスに変えています。後は、ITU3\_BRB にデューティ比を設定します。

```

699 :     /* 右モータ */
700 :     if( accele_r >= 0 ) {                   0以上かチェック
701 :         PBDR &= 0xf7;                       正転にする
702 :         ITU4_BRA = speed_max * accele_r / 100;  PWM値の設定
703 :     } else {

```

左モータの次は、右モータ制御です。まず、speed 関数の右モータ PWM の引数である accele\_r が 0 以上かチェックします。0 以上なら正転なので右モータ回転方向制御の bit3 を“0”にします。

bit	7	6	5	4	3	2	1	0	
PBDR 変更前	?	?	?	?	?	?	?	1	
AND する値	1	1	1	1	0	1	1	1	→ 0 x f 7
PBDR 変更後	※	※	※	※	0	※	※	※	

※=変化無し

次に、デューティ比を設定します。設定は ITU4\_GRA ですが、プログラムで変更するのはバッファレジスタです。ITU4\_GRA のバッファレジスタは ITU4\_BRA となります。

$$\begin{aligned}
 \text{ITU4\_BRA} &= (\text{ディップスイッチの割合に応じたバッファレジスタに代入する値}) \times (\text{speed 関数の}\%) \div 100 \\
 &= \text{speed\_max} \times \text{accele\_r} \div 100
 \end{aligned}$$

となります。ポイントは、speed 関数で設定した割合がモータに出力されるのではなく、「**ディップスイッチの割合 × speed 関数で設定した割合**」がモータに出力されるということです。

```

700 :     if( accele_r >= 0 ) {
中略
703 :     } else {                               逆転の場合
704 :         PBDR |= 0x08;                       逆転にする
705 :         accele_r = -accele_r;                PWM値を正の数にする
706 :         ITU4_BRA = speed_max * accele_r / 100;  PWM値の設定
707 :     }

```



else 文以降です。この部分は、700 行の判定が当てはまらない場合にこの部分が実行されます。判定は、「`accele_r` が 0 以上か」なので、else 文を実行するときは「**accele\_r が 0 以下**」のときになります。

逆転させるので右モータ回転方向制御の bit3 を“1”にします。bit3 のみを“1”にするには OR 演算を使います。

bit	7	6	5	4	3	2	1	0	
PBDR 変更前	?	?	?	?	?	?	?	1	
OR する値	0	0	0	0	1	0	0	0	→ 0 x 0 8
PBDR 変更後	※	※	※	※	1	※	※	※	

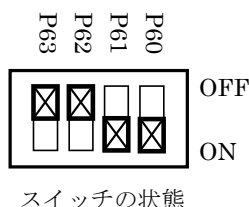
※=変化無し

次に、デューティ比を設定します。`accele_r` はマイナスなので、計算前に 705 行で符号をプラスに変えています。後は、ITU4\_BRA にデューティ比を設定します。

例えば、以下のプログラムを実行したとします。

```
speed( 70, 100 );           /* 左の割合 70%           右の割合 100%           */
```

0 0 1 1 → **3**      ディップスイッチが左図の場合、スイッチから読み出した値は 3 ですので、



左モータ PWM 値 = (ディップスイッチの値+5)/20 × speed 関数の割合  
 = (3+5)/20 × 70%  
 = 0.4 × 70%  
 = 28%

右モータ PWM 値 = (ディップスイッチの値+5)/20 × speed 関数の割合  
 = (3+5)/20 × 100%  
 = 0.4 × 100%  
 = 40%

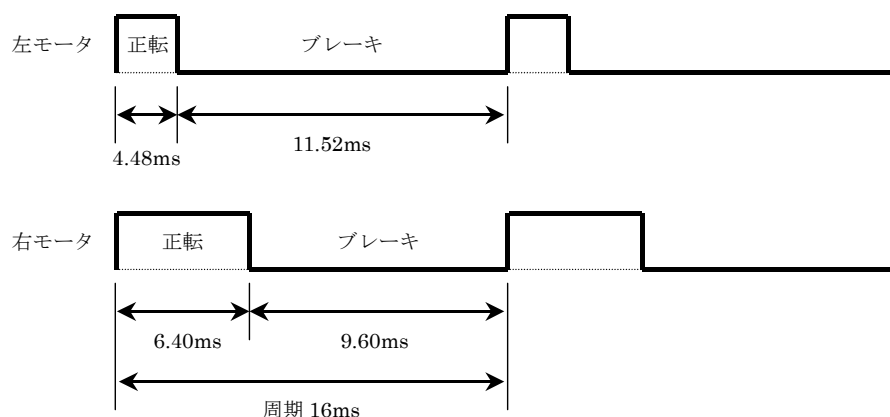
となります。これは、

左モータ→28%は正転、72%はブレーキ  
 右モータ→40%は正転、60%はブレーキ

という回転になります。1 周期は 16[ms]ですので、時間でいうと

左モータ→4.48[ms]は正転、11.52[ms]はブレーキ  
 右モータ→6.40[ms]は正転、9.60[ms]はブレーキ

という動作になります。



なお、モータドライブ基板 Vol.3 では、停止時の動作はブレーキ(モータ端子間は短絡)のみです。フリー(開放)状態はありません。

## 8.21 サーボハンドル操作 : handle関数

サーボの角度を制御する関数です。引数は、サーボ回転角度の-90~90 度の範囲ですが、実際そこまでサーボを回転させないので実質-45~45 度程度だと思えます。

```

710 :  /*****/
711 :  /* サーボハンドル操作 */
712 :  /* 引数   サーボ操作角度 : -90~90 */
713 :  /*      -90で左へ90度、0でまっすぐ、90で右へ90度回転 */
714 :  /*****/
715 :  void handle( int angle )
716 :  {
717 :      ITU4_BRB = SERVO_CENTER - angle * HANDLE_STEP;
718 :  }
    
```

モータドライブ基板の接続をもう一度確認しておきます。

ピン番	信号、方向	詳細	“0”	“1”	詳細
1	—	+5V			
2	基板←PB7	LED1	点灯	消灯	
3	基板←PB6	LED0	点灯	消灯	
<b>4</b>	<b>基板←PB5</b>	<b>サーボ信号</b>	<b>PWM 信号</b>		<b>ITU4_BRB でデューティ比設定</b>
5	基板←PB4	右モータ PWM	停止	動作	ITU4_BRA でデューティ比設定
6	基板←PB3	右モータ回転方向	正転	逆転	
7	基板←PB2	左モータ回転方向	正転	逆転	
8	基板←PB1	左モータ PWM	停止	動作	ITU3_BRB でデューティ比設定
9	基板→PB0	プッシュスイッチ	押された	押されていない	
10	—	GND			

表のとおり、PB5 がサーボ制御です。リセット同期 PWM モードを使用していますので PB5 のデューティ比を変えるには ITU4\_BRB の値を変えます。

```

717 :      ITU4_BRB = SERVO_CENTER - angle * HANDLE_STEP;
           ↑           ↑           ↑
           サーボの   角度   1度あたりの増分
           センタ値
    
```

SERVO\_CENTER がサーボの中心です。その値から angle 度分、角度を変えます。ただし、「ITU4\_BRB の値1 = 1度ではない」ため、1度分の増減量である HANDLE\_STEP を angle にかけています。

## 8.22 メインプログラム

### 8.22.1 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。

```

73 : /*****
74 : /* メインプログラム */
75 : /*****
76 : void main( void )
77 : {
78 :     int    i;
79 :
80 :     /* マイコン機能の初期化 */
81 :     init();                /* 初期化          */
82 :     set_ccr( 0x00 );      /* 全体割り込み許可 */
83 :
84 :     /* マイコンカーの状態初期化 */
85 :     handle( 0 );
86 :     speed( 0, 0 );

```

```

81 :     init();                /* 初期化          */

```

init 関数を実行し、H8/3048F-ONE の内蔵周辺機能の I/O レジスタを初期化します。

```

82 :     set_ccr( 0x00 );      /* 全体割り込み許可 */

```

CPU 全体の割り込みを許可します。

```

84 :     /* マイコンカーの状態初期化 */
85 :     handle( 0 );
86 :     speed( 0, 0 );

```

次に、マイコンカーの状態を初期化します。

- ・ハンドルは 0 度
- ・スピードは左 0%、右 0%  
しています。

### 8.22.2 パターン方式について

kit07.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。

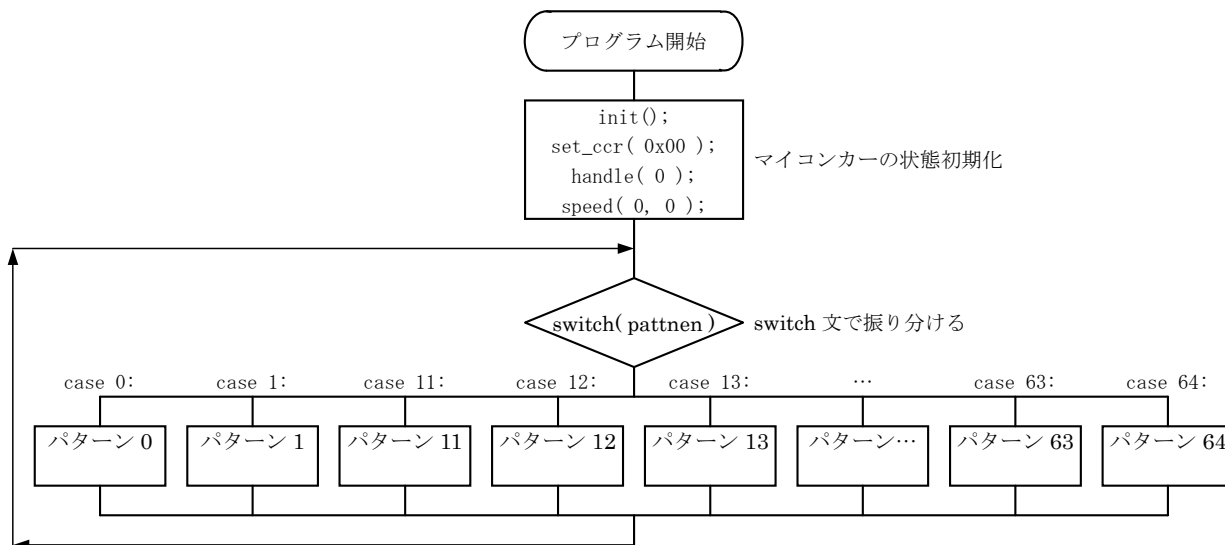
次に、pattern という変数を作ります。この変数に設定した値により、どのプログラムを実行するか選択します。

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。

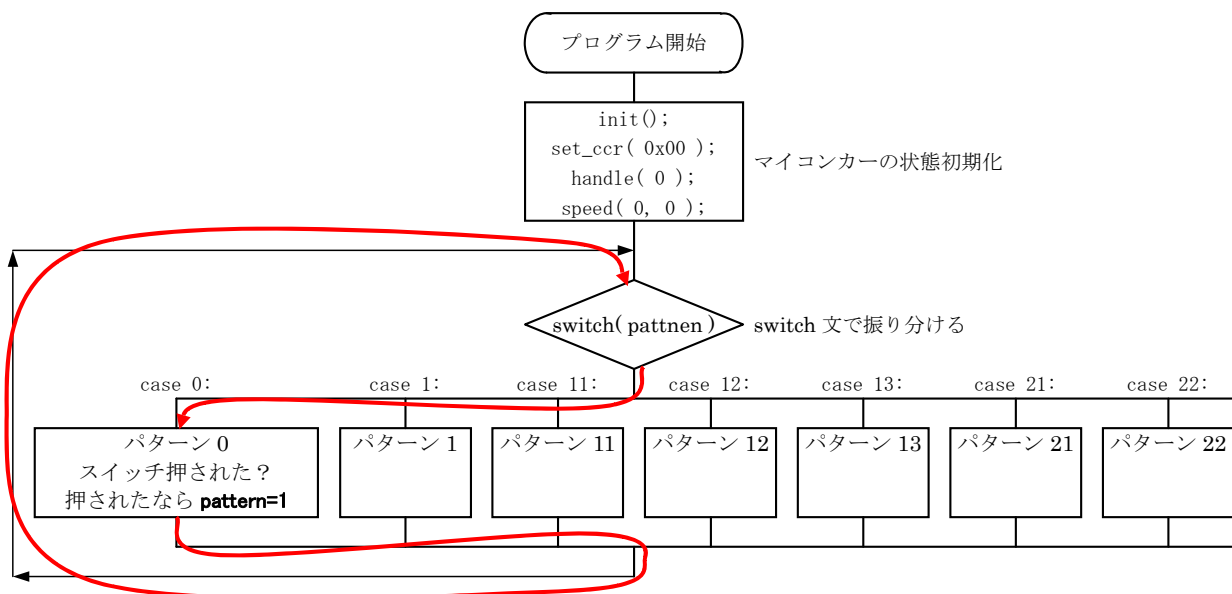
この方式を使うと、パターンごとに処理を分けられるため、プログラムが見やすくなります。パターン方式は、「プログラムのブロック化」と言うこともできます。

### 8.22.3 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。フローチャートは下図のようになります。

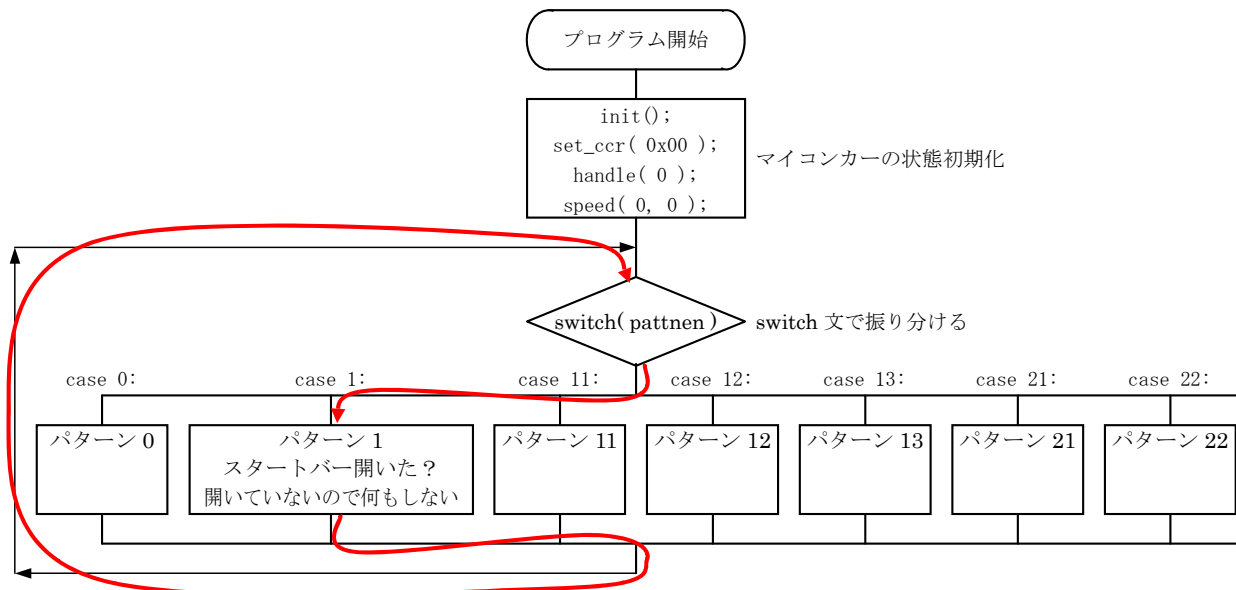


起動時、pattern 変数は0です。switch 文により case 0 部分のプログラム「パターン0 プログラム」を実行し続けます。後ほど説明しますが、パターン 0 はスイッチ入力待ちです。スイッチが押されると、「pattern=1」が実行されます。下図のようです。



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 部分にある「パターン 1 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 で実行されるプログラムを、パターン 1 を実行すると言うことにします。

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。下図のようです。



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。

プログラムは下記のようになります。通常の switch~case 文です。

<pre> switch( pattern ) { case 0:     /* pattern=0 の処理 */     break; case 1:     /* pattern=1 の処理 */     break; default:     /* どれも無いなら */     break; }                 </pre>	<p>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。</p> <p>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。</p> <p>どの case 文の定数にも当てはまらない場合は、default 文を実行。ちなみに、default 文も無い場合は何も実行しない。</p>
--	---

### 8.22.4 パターンの内容

kit07.c のパターン番号と、プログラムの内容は下記のようになっています。

太字部分が、スタートバー、レーンチェンジコースが追加になったことにより、プログラムを追加、または変更した部分です。

パターン	処理内容	備考
0	スイッチ入力待ち	0～10 は、走行前の処理を行っています
1	<b>スタートバーが開いたかチェック</b>	
11	<b>通常トレース</b>	11～20 は、通常走行中の処理を行っています
12	<b>右へ大曲げの終わりのチェック</b>	
13	<b>左へ大曲げの終わりのチェック</b>	
21	1本目のクロスライン検出時の処理	21～30 は、クロスラインを見つけてから直角までの処理を行っています
22	2本目を読み飛ばす	
23	クロスライン後のトレース、クランク検出	
31	左クランククリア処理 安定するまで少し待つ	31～40 は、左クランク処理を行っています
32	左クランククリア処理 曲げ終わりのチェック	
41	右クランククリア処理 安定するまで少し待つ	41～50 は、右クランク処理を行っています
42	右クランククリア処理 曲げ終わりのチェック	
51	<b>1本目の右ハーフライン検出時の処理</b>	51～60 は、右ハーフラインを見つけてから右レーンチェンジをクリアするまでの処理を行っています
52	<b>2本目の右ハーフラインを読み飛ばす</b>	
53	<b>右ハーフライン後のトレース</b>	
54	<b>右レーンチェンジ終了のチェック</b>	
61	<b>1本目の左ハーフライン検出時の処理</b>	61～70 は、左ハーフラインを見つけてから左レーンチェンジをクリアするまでの処理を行っています
62	<b>2本目の左ハーフラインを読み飛ばす</b>	
63	<b>左ハーフライン後のトレース</b>	
64	<b>左レーンチェンジ終了のチェック</b>	

このように、パターンの番号と処理内容を決めて、プログラムを作っていきます。

パターン番号は、0、1、11、12…と値が飛び飛びになっています。これは、備考のように 0 番台を走行前の処理、10 番台を通常走行処理というように、10 ごとに大まかな処理内容を決めて分類しているためです。

パターンの流れについて、下表にまとめます。常にパターンの流れを意識しながらプログラムを作ったり解析したりすると、理解しやすいかと思います。

現在のパターン	状態	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	・右大曲げになったらパターン 12 へ ・左大曲げになったらパターン 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	1本目のクロスライン検 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	2本目を読み飛ばす	・100ms たったらパターン 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	左クランククリア処理 安定するまで少し待つ	・200ms たったならパターン 32 へ
32	左クランククリア処理 曲げ終わりのチェック	・左クランクをクリアしたならパターン 11 へ
41	右クランククリア処理 安定するまで少し待つ	・200ms たったならパターン 42 へ
42	右クランククリア処理 曲げ終わりのチェック	・右クランクをクリアしたならパターン 11 へ
51	1本目の右ハーフライン 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	2本目を読み飛ばす	・100ms たったならパターン 53 へ
53	右ハーフライン後の トレース	・中心線が無くなったならパターン 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らパターン 11 へ
61	1本目の左ハーフライン 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	2本目を読み飛ばす	・100ms たったならパターン 63 へ
63	左ハーフライン後の トレース	・中心線が無くなったならパターン 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らパターン 11 へ
現在のパターン	状態	内容

### 8.22.5 パターン方式の最初while、switch部分

```

88 :   while( 1 ) {
89 :       switch( pattern ) {
115 :           case 0:
                パターン0時の処理
129 :               break;
130 :
131 :           case 1:
                パターン1時の処理
146 :               break;

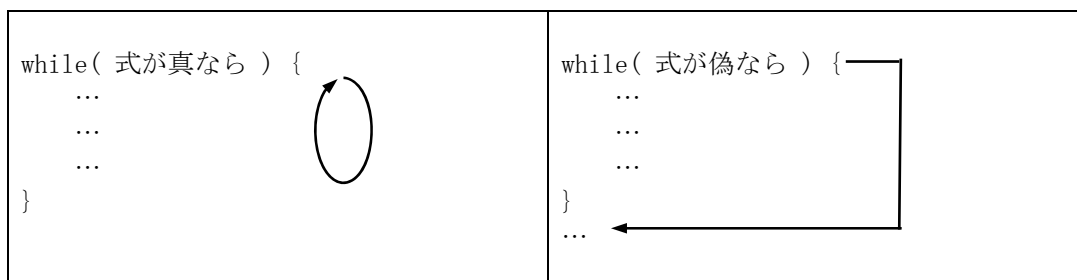
                それぞれのパターン処理

483 :       default:
484 :           /* どれも無い場合は待機状態に戻す */
485 :           pattern = 0;
486 :           break;
487 :       }
488 :   }
    
```

88 行の「while( 1 ) {」と 488 行の「}」が対、89 行の「switch( pattern ) {」と 487 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、くられた中は 4 文字右にずらして分かりやすくします。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。元々、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有りません。どうしても気になってしまう場合は、89~487 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、このプログラムはマイコンカーを動かすためのプログラムです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、人間が取り上げてスイッチを切れればいいのです。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



アイコンは、何もし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が普通です。

### 8.22.6 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、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、LED0 と LED1 を交互に光らせます。

まず、スイッチ検出部分です。

```

115 :     case 0:
116 :         /* スイッチ入力待ち */
117 :         if( pushsw_get() ) {           ←スイッチが押されたなら(戻り値が 0 以外なら)
118 :             pattern = 1;              ←パターンを1に
119 :             cnt1 = 0;                  ←cnt1 を 0 に
120 :             break;                     ←そして switch 文を終了
121 :         }

```

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。

ここで if 文のカッコの中を注目します。

```

if( pushsw_get() == 1 ) {

```

なら、「pushsw\_get 関数の戻り値が1ならカッコの中を実行しなさい」と、意味が分かります。

しかし、今回のプログラムは、

```

if( pushsw_get() ) {

```

となっています。どの値とも比較していません。これはどういう意味でしょうか。C言語は、このようなプログラムもきちんとした意味があります。

```

if( 値 ) {
    0以外の値なら成り立つと判断し、この部分を実行
} else {
    0なら成り立たないと判断し、この部分を実行
}

```

となります。

pushsw\_get 関数の戻り値は1がスイッチが押されている状態です。そのため、スイッチが押されたら**0以外の値と判断され**、118～120行が実行されます。0なら、何も実行しません。

この後に、LEDを点滅させるプログラムを追加します。点滅は、0.1秒LED0が点灯、次の0.1秒LED1が点灯、それを繰り返す処理にします。

```

122 :     if( cnt1 < 100 ) {                ←cnt1 が 0～99 か
123 :         led_out( 0x1 );               ←なら LED0 のみ点灯
124 :     } else if( cnt1 < 200 ) {         ←cnt1 が 100～199 か
125 :         led_out( 0x2 );               ←なら LED1 のみ点灯
126 :     } else {                           ←それ以外なら(200 以上)
127 :         cnt1 = 0;                       ←cnt1 を0にする
128 :     }

```

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。kit07.c では、**cnt0 変数と cnt1 変数だけは例外です**。cnt0 と cnt1 は interrupt\_timer0 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。

スイッチ検出とLED点滅のプログラムを合わせると、次のようなプログラムになります。

```

115 :     case 0:
116 :         /* スイッチ入力待ち */
117 :         if( pushsw_get() ) {
118 :             pattern = 1;
119 :             cnt1 = 0;
120 :             break;
121 :         }
122 :         if( cnt1 < 100 ) {                /* LED点滅処理                */
123 :             led_out( 0x1 );
124 :         } else if( cnt1 < 200 ) {
125 :             led_out( 0x2 );
126 :         } else {
127 :             cnt1 = 0;
128 :         }
129 :         break;

```

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

パターンが変わる条件

- ・スイッチが押されたら、パターン1へ

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;

```

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**待つ以外何もしません**。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もうスイッチは押されていません。検出もれしてしまいます。このプログラムでは 0.2 秒ですので、かなり素早くスイッチを押さなければ検出できないということはありませんが、もしこれが何秒間というように更に長いタイムになると timer 関数を使用したのでは、スイッチをチェックしない時間が多すぎ、検出できなくなります。そのため、**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**。

### 8.22.7 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、LED0 と LED1 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。

```

131 :     case 1:
132 :         /* スタートバーが開いたかチェック */
133 :         if( !startbar_get() ) {
134 :             /* スタート!! */
135 :             led_out( 0x0 );
136 :             pattern = 11;
137 :             cnt1 = 0;
138 :             break;
139 :         }

```

startbar\_get 関数でスタートバーをチェックします。スタートバーが開いたら(無くなったら)0 が返ってくるので、カッコの中が実行されパターンを 11 にします。

ここで if 文のカッコの中を注目します。

```

if( !startbar_get() ) {

```

「!」が付いています。これは否定です。したがって、「startbar\_get 関数の戻り値が 0 以外でないならカッコの中を実行しなさい」という意味になります。要は、「startbar\_get 関数の戻り値が 0 なら」ということです。

```

if( !値 ) {
    0ならこの部分を実行
} else {
    0でないならこの部分を実行
}

```

となります。

startbar\_get 関数の戻り値は 1 がスタートバーあり、0 がスタートバーなしです。スタートバーが開いたら(スタートバーが無くなったら)スタートさせたいので、「!」マークを付けて「0になったら実行する」ようにしています。

この後に、LED を点滅させるプログラムを追加します。点滅は、0.05 秒 LED0 が点灯、次の 0.05 秒 LED1 が点灯、それを繰り返す処理にします。

```

140 :         if( cnt1 < 50 ) {                ←cnt1 が 0~49 か
141 :             led_out( 0x1 );              ←なら LED0 のみ点灯
142 :         } else if( cnt1 < 100 ) {        ←cnt1 が 50~99 か
143 :             led_out( 0x2 );              ←なら LED1 のみ点灯
144 :         } else {                          ←それ以外なら(100 以上)
145 :             cnt1 = 0;                      ←cnt1 を 0 にする
146 :         }

```

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。kit07.c では、**cnt0 変数と cnt1 変数だけは例外です**。cnt0 と cnt1 は interrupt\_timer0 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。

スタートバー検出と LED 点滅のプログラムを合わせると、次のようなプログラムになります。

```

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

```

パターンが変わる条件

・スタートバー検出センサがスタートバーが開いたことを検出したら、パターン11へ

### 8.22.8 パターン11:通常トレース

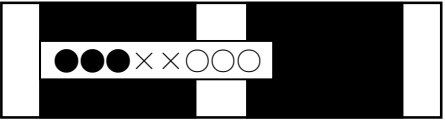
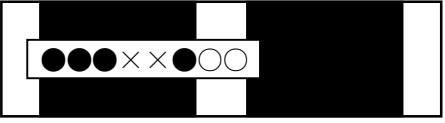
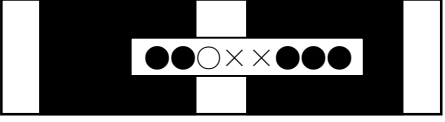
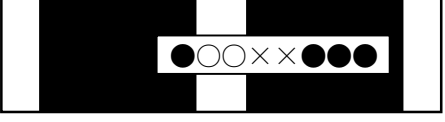
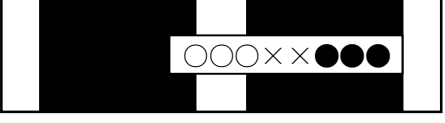
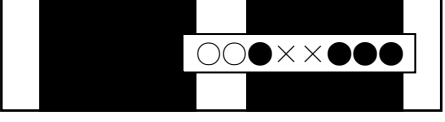
パターン 11 は、コース上をトレースする状態です。

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。

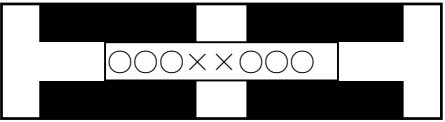
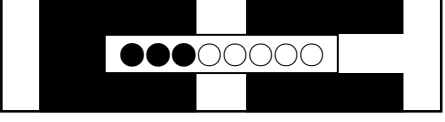
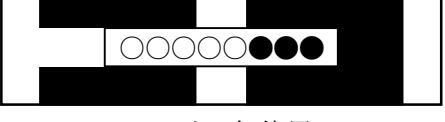
次に、そのときのハンドル角度と左モータ、右モータの PWM 値を考えます。センサが中心のときはスピードを上げます。センサが中心よりずればずれるほど、ハンドルを大きく曲げてスピードを落とします。

とりあえず下記のように考えました。

	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
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 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 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 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

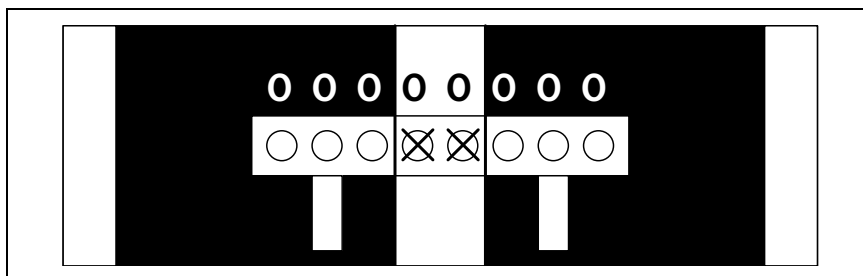
```
163 :          switch( sensor_inp(MASK3_3) ) {
```

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。

(2) 直進

```
164 :          case 0x00:
165 :              /* センタ→まっすぐ */
166 :              handle( 0 );
167 :              speed( 100 ,100 );
168 :              break;
```

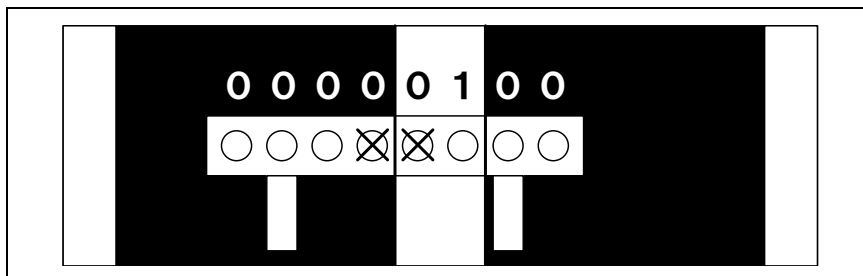
センサが「0x00」の状態です。この状態は下図のように、まっすぐ進んでいる状態です。サーボ角度 0 度、左モータ 100%、右モータ 100%で進みます。



(3) 左寄り

```
170 :          case 0x04:
171 :              /* 微妙に左寄り→右へ微曲げ */
172 :              handle( 5 );
173 :              speed( 100 ,100 );
174 :              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。

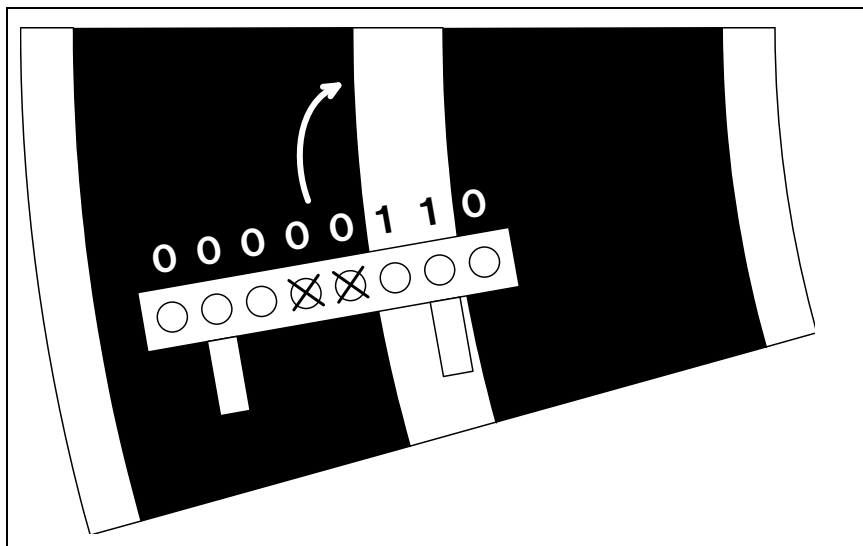


(4) 少し左寄り

```

176 :          case 0x06:
177 :             /* 少し左寄り→右へ小曲げ */
178 :             handle( 10 );
179 :             speed( 80 ,67);
180 :             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。

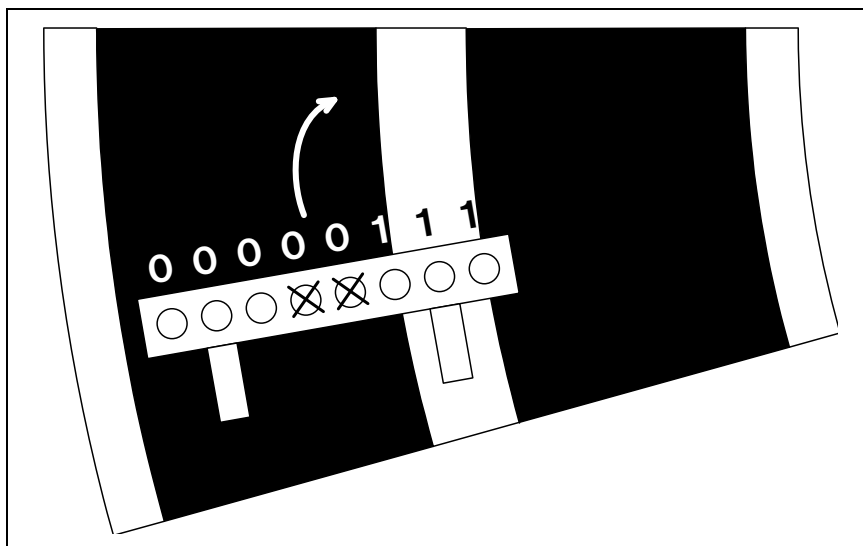


(5) 中くらい左寄り

```

182 :          case 0x07:
183 :             /* 中くらい左寄り→右へ中曲げ */
184 :             handle( 15 );
185 :             speed( 50 ,38 );
186 :             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。



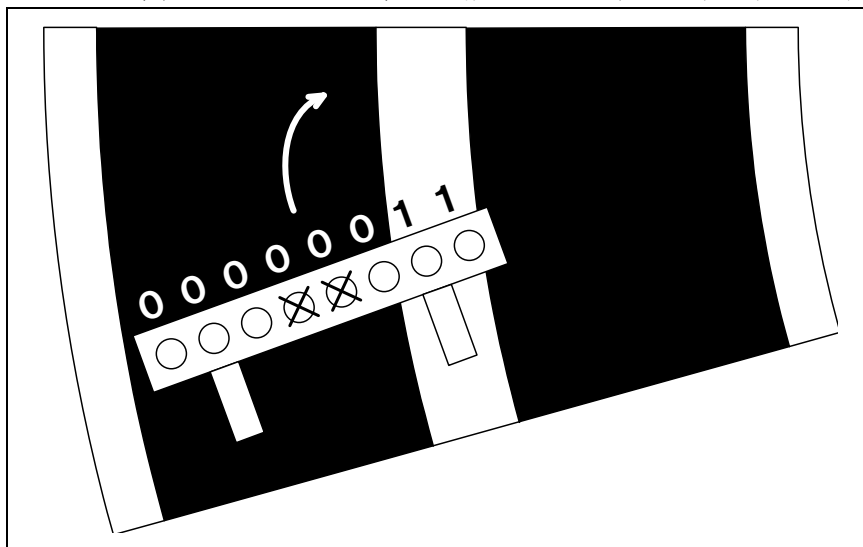
(6) 大きく左寄り

```

188 :          case 0x03:
189 :             /* 大きく左寄り→右へ大曲げ */
190 :             handle( 25 );
191 :             speed( 30 , 19 );
193 :             break;
    
```

※本当のプログラムは 192 行が入っています。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。

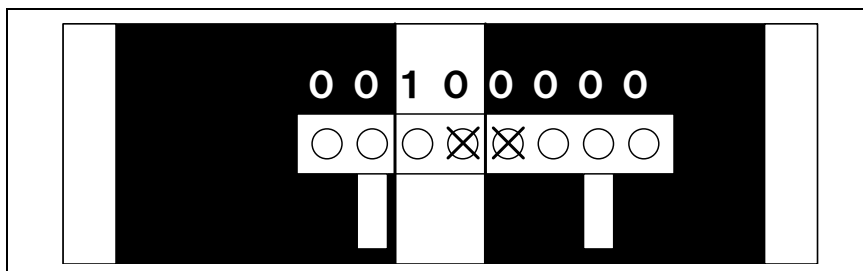


(7) 微妙に右寄り

```

195 :          case 0x20:
196 :             /* 微妙に右寄り→左へ微曲げ */
197 :             handle( -5 );
198 :             speed( 100 , 100 );
199 :             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。



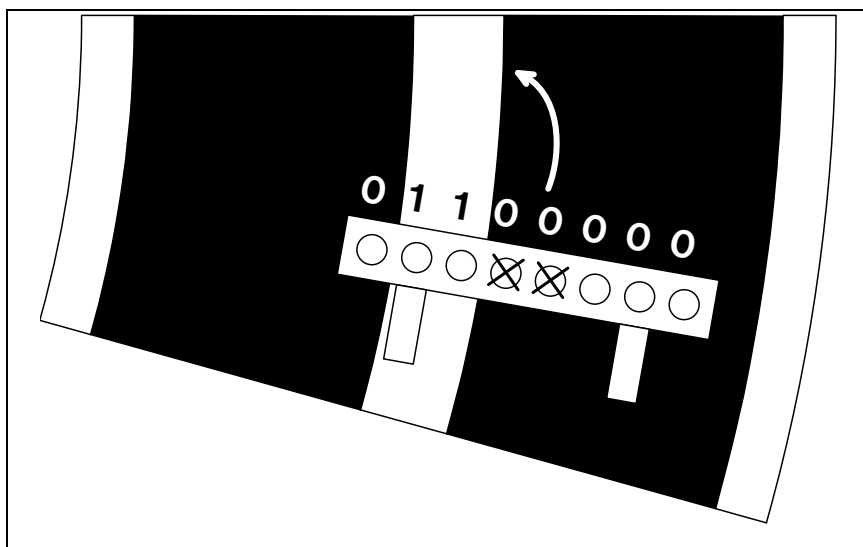


(8) 少し右寄り

```

201 :          case 0x60:
202 :             /* 少し右寄り→左へ小曲げ */
203 :             handle( -10 );
204 :             speed( 67 ,80 );
205 :             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。

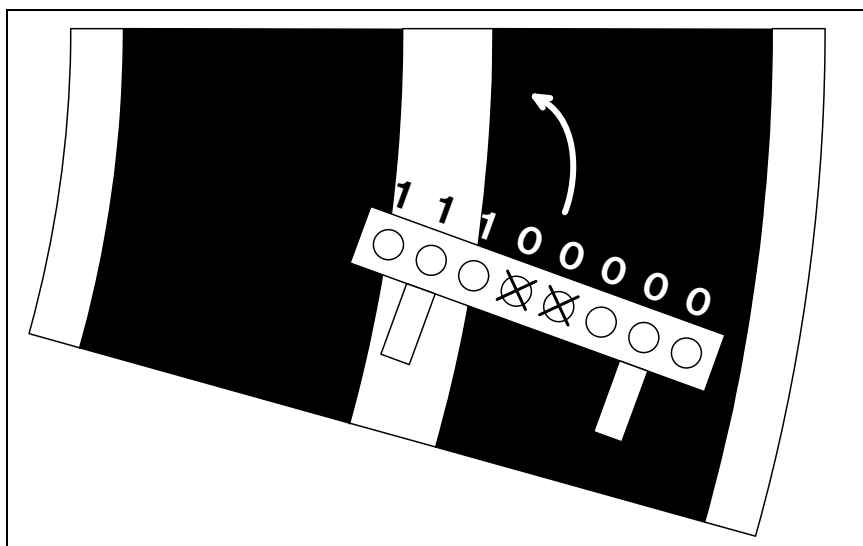


(9) 中くらい右寄り

```

207 :          case 0xe0:
208 :             /* 中くらい右寄り→左へ中曲げ */
209 :             handle( -15 );
210 :             speed( 38 ,50 );
211 :             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。



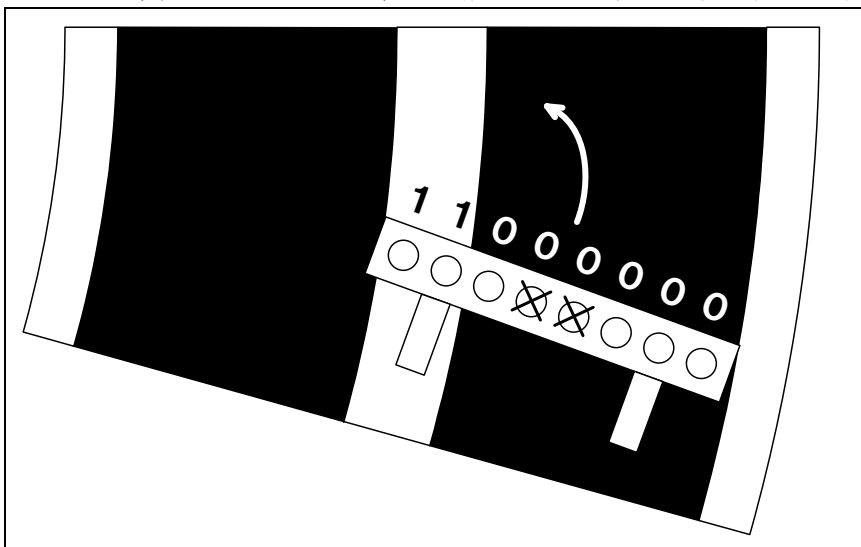
(10) 大きく右寄り

```

213 :          case 0xc0:
214 :              /* 大きく右寄り→左へ大曲げ */
215 :              handle( -25 );
216 :              speed( 19 , 30 );
218 :              break;
    
```

※本当のプログラムは 217 行が入っています。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

```

151 :          if( check_crossline() ) {          /* クロスラインチェック */
152 :              pattern = 21;
153 :              break;
154 :          }
    
```

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

```

155 :          if( check_rightline() ) {          /* 右ハーフラインチェック */
156 :              pattern = 51;
157 :              break;
158 :          }
    
```

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

```
159 :         if( check_leftline() ) {           /* 左ハーフラインチェック */
160 :             pattern = 61;
161 :             break;
162 :         }
```

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

```
220 :             default:
221 :                 break;
```

いままでのパターン以外るとき、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、その break 文の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 が抜けた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: ← switch( sensor_inp(MASK3_3) )に対応する default
        break; ← switch( sensor_inp(MASK3_3) )を抜ける break、2へ
    }
    break; ← switch( pattern )を抜ける break、1へ

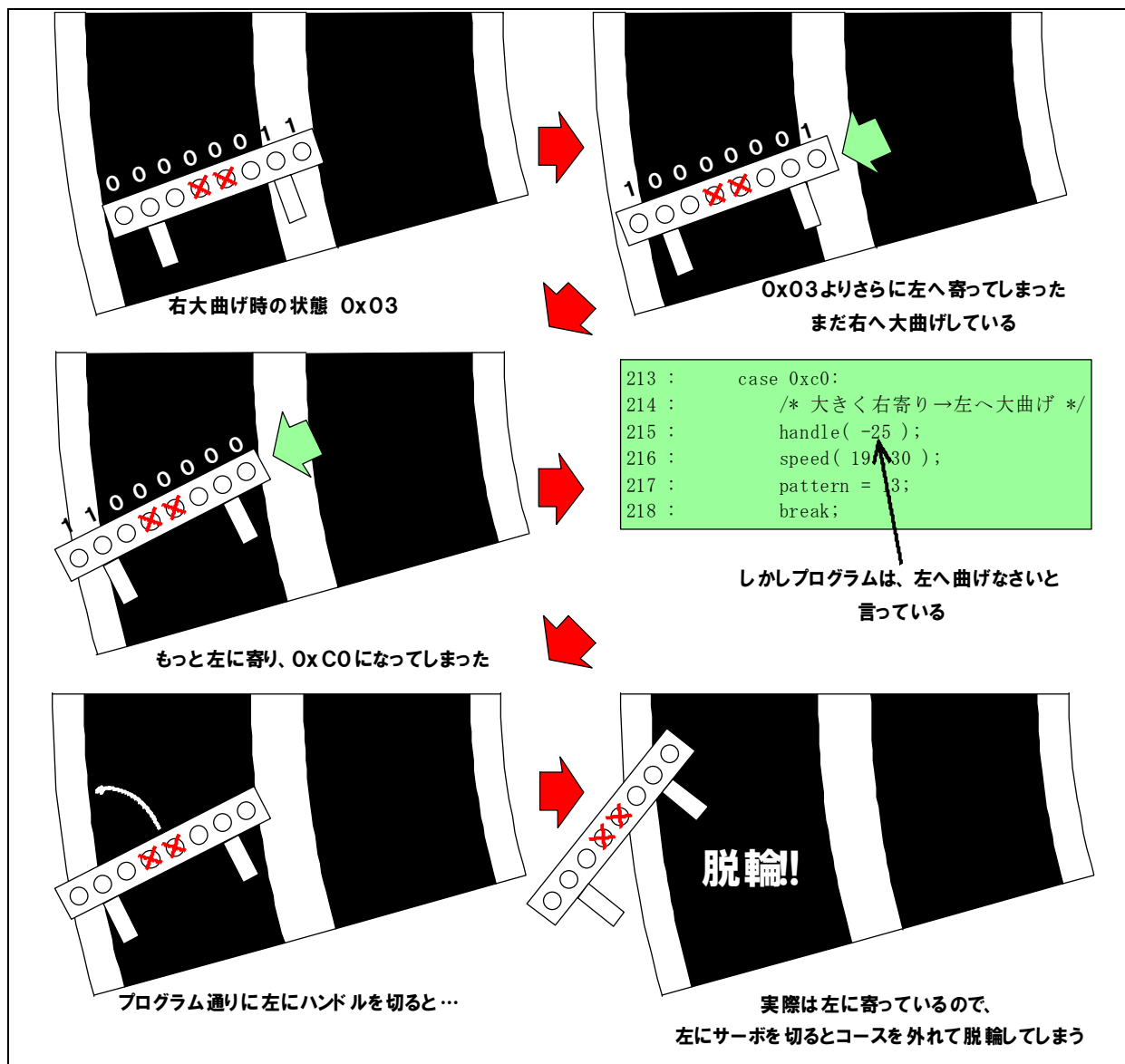
    中略

  }
}

```

### 8.22.9 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。



本当は左に大きく寄っている場合でもプログラムでは、「右に大きく寄っている」と誤った判断をすることがあります。もちろん、誤った判断をするとサーボを逆に切るのですぐさま脱輪します。

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ます。この“あるセンサ状態”を判定するのが、パターン 12 になります。

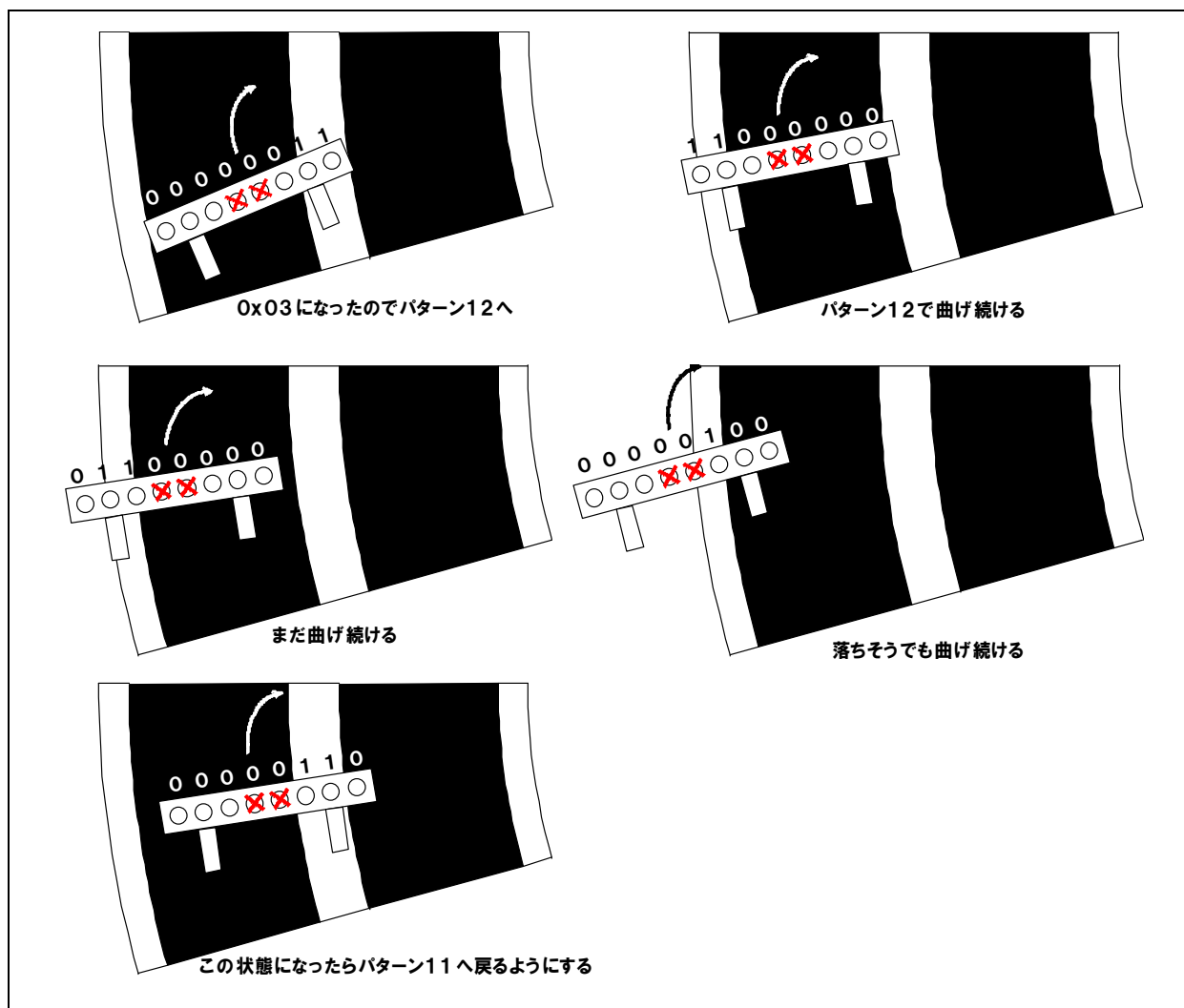
パターン 11 の case 0x03 部分

```

188 :         case 0x03:
189 :             /* 大きく左寄り→右へ大曲げ */
190 :             handle( 25 );
191 :             speed( 30 , 19 );
192 :             pattern = 12; ←追加 パターン12へ移る
193 :             break;
    
```

センサが 0x03 になるとパターン 12 へ移ります。

パターン 12 では、どのようになったら通常走行のパターン 11 へ戻るか考えてみます。



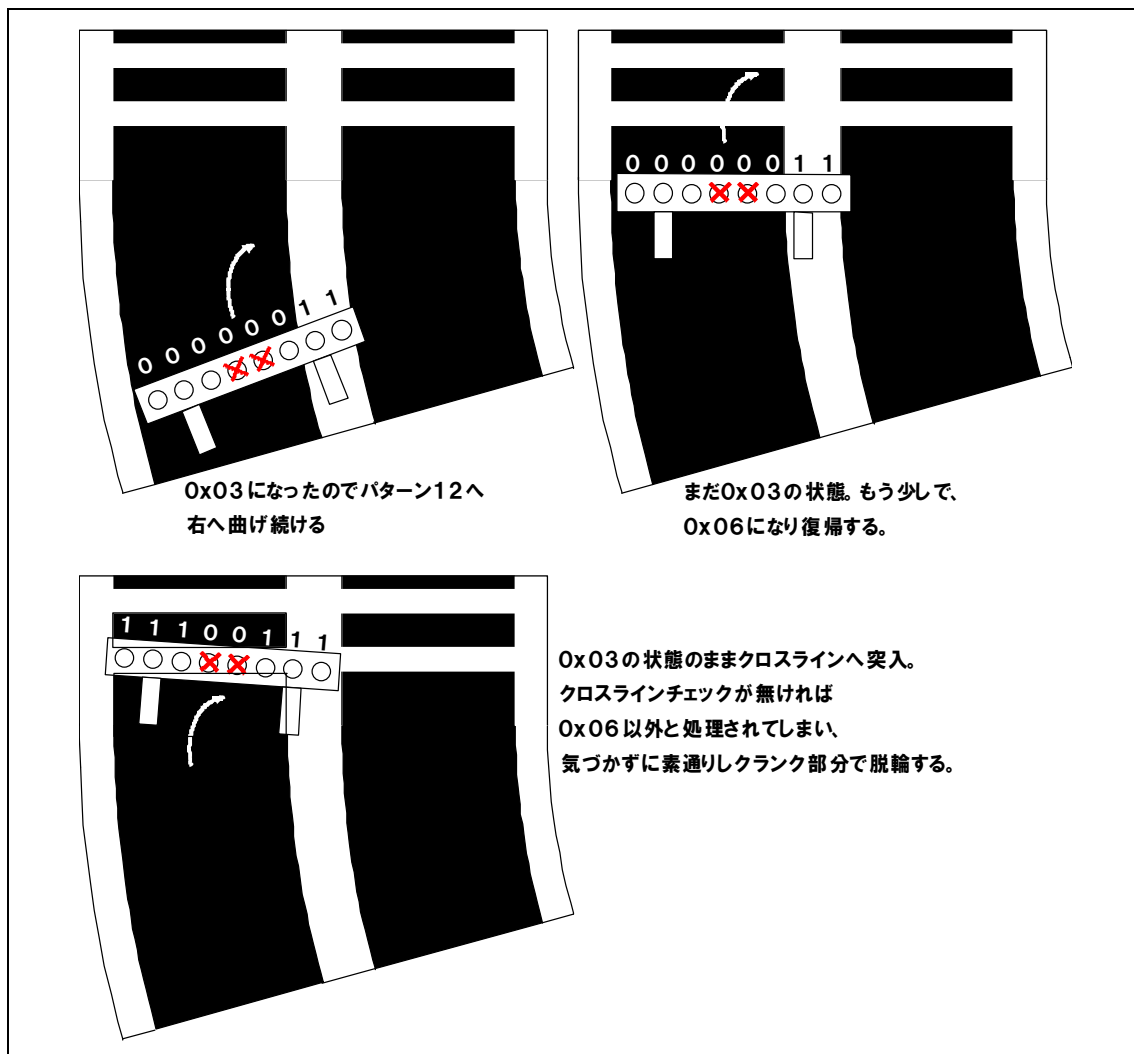
センサ状態が 0x06 になったらパターン 11 へ戻るようにすれば、先ほどの勘違いをなくせそうです。

```

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。



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 12 にも 3 種類のチェックを追加します。

```

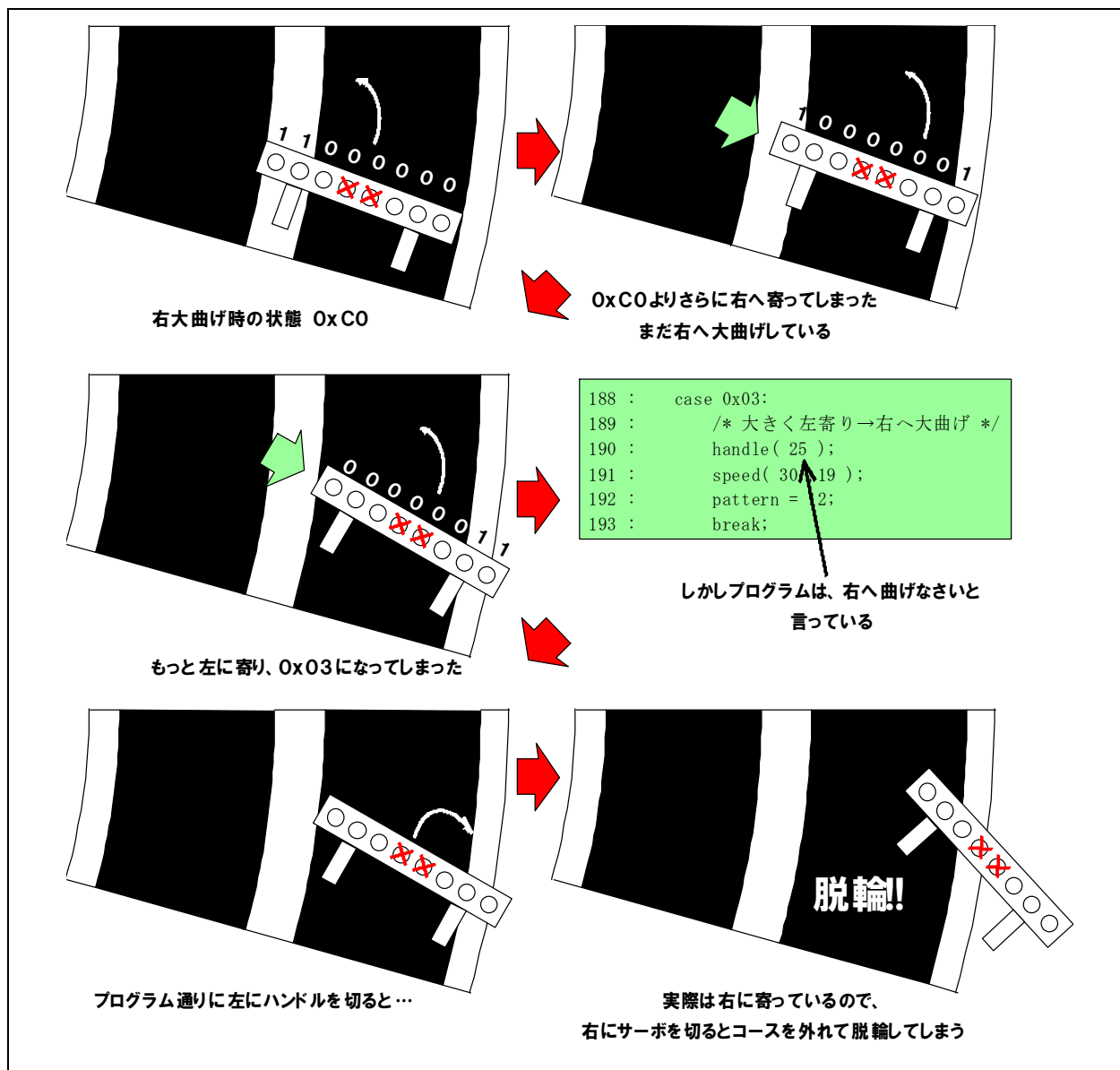
225 :     case 12:
226 :         /* 右へ大曲げの終わりのチェック */
227 :         if( check_crossline() ) {          /* 大曲げ中もクロスラインチェック */
228 :             pattern = 21;
229 :             break;
230 :         }
231 :         if( check_rightline() ) {          /* 右ハーフラインチェック */
232 :             pattern = 51;
233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

```

パターン 12 のプログラムはこれで完成です。

### 8.22.10 パターン13:左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。



本当は右に大きく寄っている場合でもプログラムでは、「左に大きく寄っている」と誤った判断をすることがあります。もちろん、誤った判断をするとサーボを逆に切るのですぐさま脱輪します。

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ます。この“あるセンサ状態”を判定するのが、パターン 13 になります。

パターン 11 の case 0xc0 部分

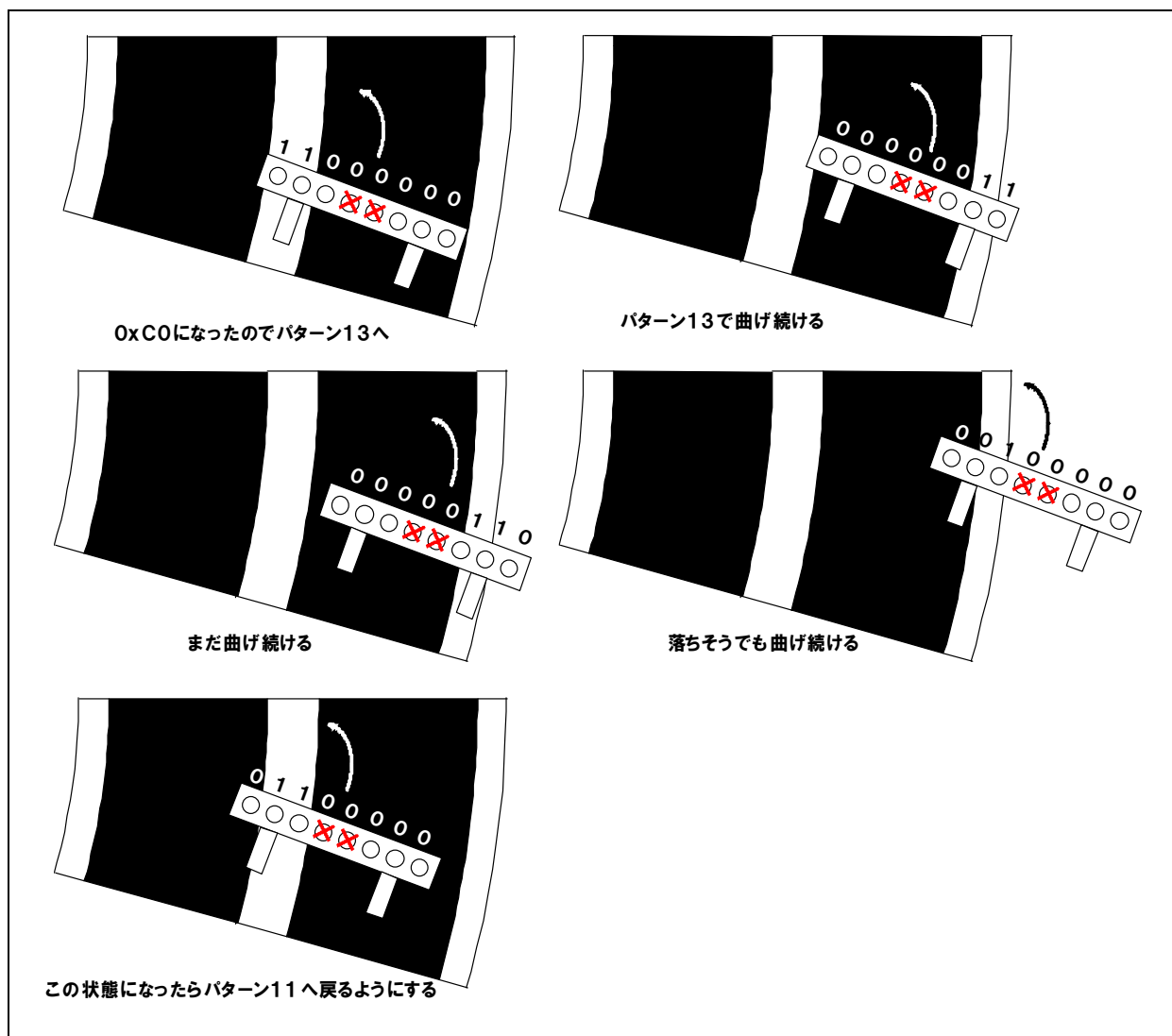
```

213 :     case 0xc0:
214 :         /* 大きく右寄り→左へ大曲げ */
215 :         handle( -25 );
216 :         speed( 19, 30 );
217 :         pattern = 13; ←追加 パターン13へ移る
218 :         break;
    
```



センサが 0xc0 になるとパターン 13 に移ります。

パターン 13 では、どのようになったら通常走行のパターン 11 へ戻るか考えてみます。



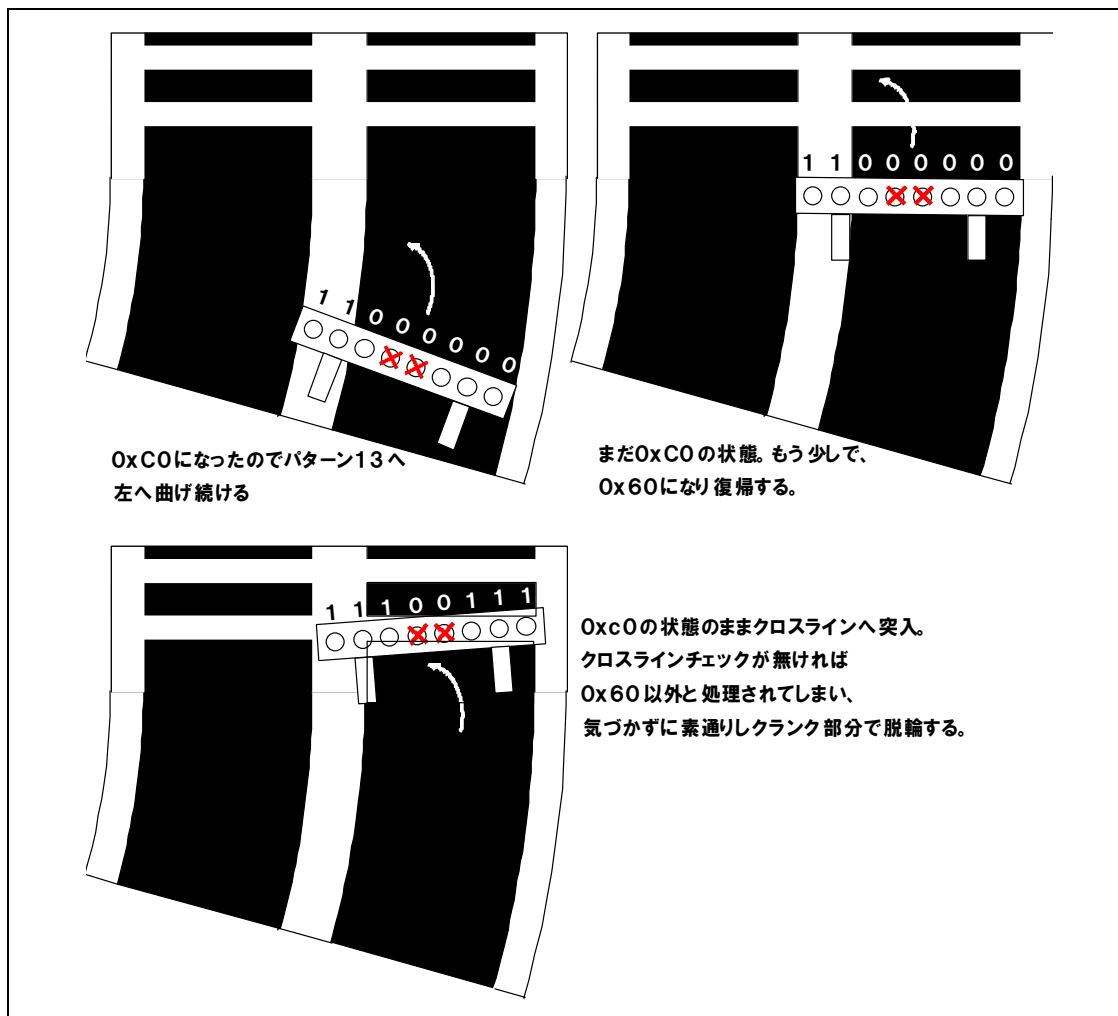
センサ状態が 0x60 になったらパターン 11 へ戻るようにすれば、先ほどの勘違いをなくせそうです。

```

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。



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。

```

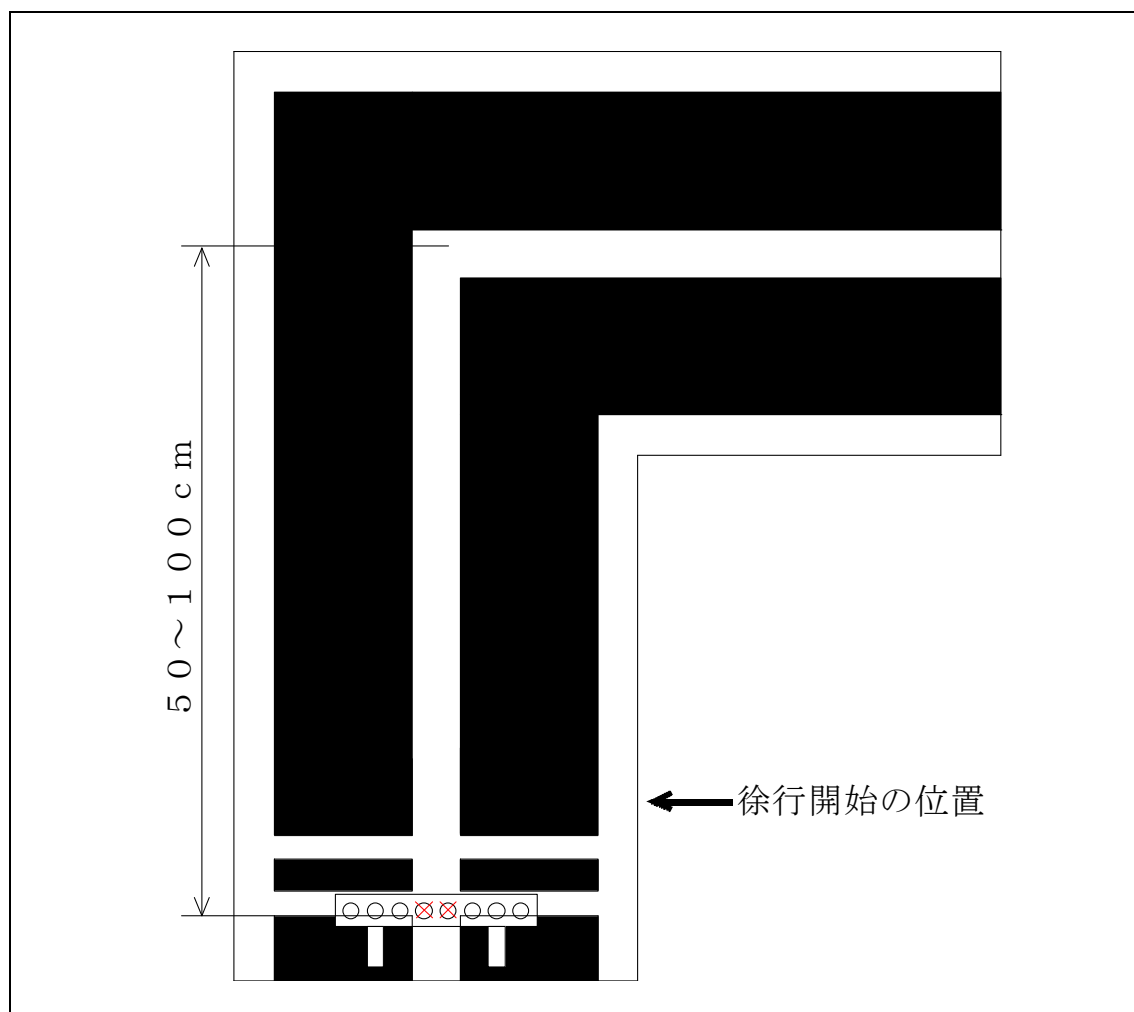
244 :     case 13:
245 :         /* 左へ大曲げの終わりのチェック */
246 :         if( check_crossline() ) { /* 大曲げ中もクロスラインチェック */
247 :             pattern = 21;
248 :             break;
249 :         }
250 :         if( check_rightline() ) { /* 右ハーフラインチェック */
251 :             pattern = 51;
252 :             break;
253 :         }
254 :         if( check_leftline() ) { /* 左ハーフラインチェック */
255 :             pattern = 61;
256 :             break;
257 :         }
258 :         if( sensor_inp(MASK3_3) == 0x60 ) {
259 :             pattern = 11;
260 :         }
261 :         break;

```

パターン 13 のプログラムはこれで完成です。

### 8.22.11 パターン 21: 1 本目のクロスライン検出時の処理

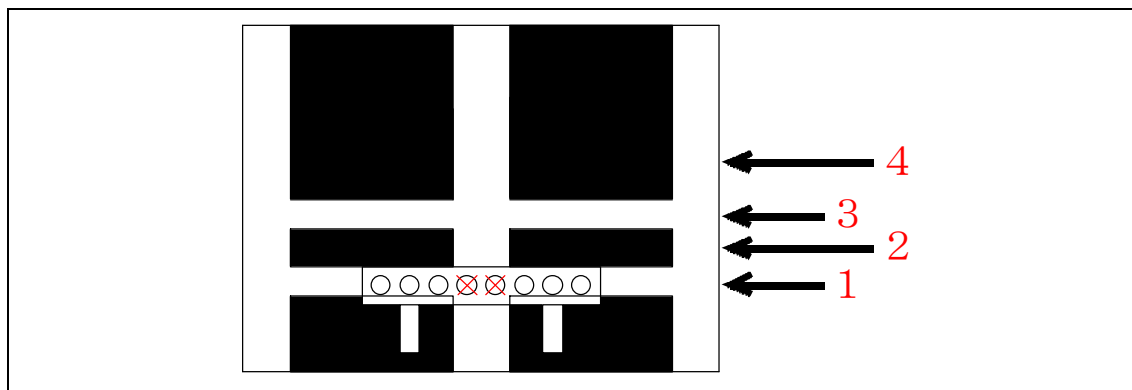
パターン 21 には、下図のような状態になった瞬間に移ってきます。



クロスラインの後、50~100cm 先には、コースでもっとも難関であるクランク(直角)があることを示しています。まず、何をすべきか。マイコンカーは、かなりのスピードがついています。そのままのスピードで直角を曲がるのは無理な話です。まずブレーキをかけます。クロスライン後は、直線ということが分かっていますのでハンドルを0度にしします。

ブレーキは、「徐行開始の位置」までかけます。その後、徐行して進ませようと考えました。

最初のクロスラインを見つけてから徐行部分へ進むまでに下図のような状態があります。



1 が1本目のクロスライン、2 が黒、3 が2 本目のクロスライン、そして4 が黒で徐行開始の部分です。コースが白→黒→白→黒と変化したことを検出して、4 まで進んだか判別します。そして、4 部分でブレーキを解除するプログラムが必要です。なんだか複雑そうです。

ちょっと考え方を考えてみます。クロスラインを検出して4 の位置まで進ませるとします。10cm くらいでしょうか。10cm くらいならタイマで少し時間稼ぎをすれば惰性で進み難いセンサ判断をせずに済みそうです。その時間は… これは実験してみないと何とも言えません。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とします。いっしょに、クロスラインを検出したとき、LED を点灯させパターン21 に入ったよ！ ということを外部に知らせるようにします。

まとめると、

- LED0,1 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
- 0.1 秒待つ
- 時間がたっ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 以上になったら(100 ミリ秒たったら)、パターン22 へ移るようにしています。それはパターン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 であつたら、1 回目で cnt1 は100 以上と判断してしまい、0.1 秒どころかほとんどパターン21 が実行されません。cnt1 が0 である保証はどこにもありません。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 は0.1 秒たったかチェックする部分に分けます。

再度まとめると、

●パターン 21 で行うこと

- ・LED0,1 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

- ・cnt1 が 100 以上になったかチェック
- ・なったら、パターンを次へ移す

ゴシック体(赤)が変更した部分です。

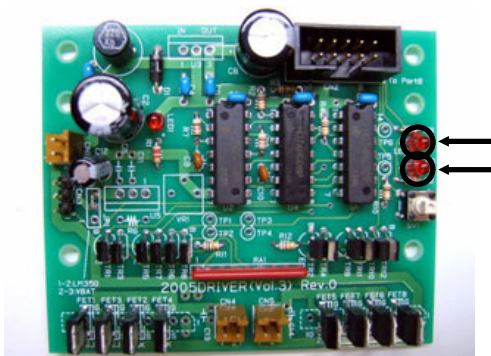
上記にしたがって再度プログラムを作ってみます。

```
263 :     case 21:
264 :         /* 1 本目のクロスライン検出時の処理 */
265 :         led_out( 0x3 );
266 :         handle( 0 );
267 :         speed( 0 , 0 );
268 :         pattern = 22;
269 :         cnt1 = 0;
270 :         break;
271 :
272 :     case 22:
273 :         /* 2 本目を読み飛ばす */
274 :         if( cnt1 > 100 ) {
275 :             pattern = 23;
276 :             cnt1 = 0;
277 :         }
278 :         break;
```

完成しました。本当にこれでよいか見直してみます。パターン 21 でブレーキさせ、時間の基準である cnt1 をクリアしパターン 22 へ。パターン 22 では 0.1 秒たったかチェック。たったならパターン 23 へ。

これでクロスラインを検出してから徐行開始までのプログラムが完成しました。

ポイント



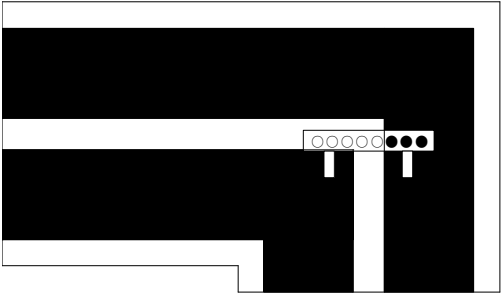
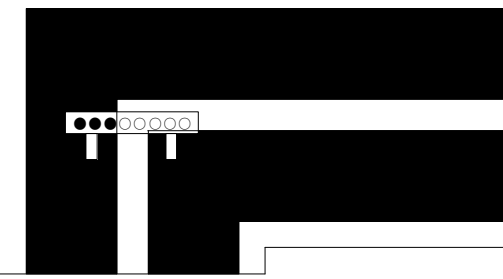
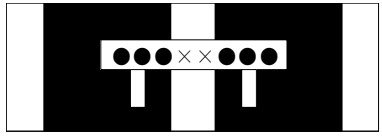
クロスラインを検出すると、モータドライブ基板の LED が 2 個点きます。点いていなければ、クロスラインを検出していません。

8.22.12 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2本の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、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、下図のように考えました。

 <p style="text-align: center;">→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膨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。ハンドルをどんどん切っていくとシャーシにぶつかると思います。そのときが最大の角度です。分度器で何度か測ってみます。大体40度くらいでした。ぶつからないように、2 度くらい余裕を見てプログラムでは 38 度にします。もし 60 度までいってもぶつからない場合は、60 度くらいで止めておいた方がよいでしょう。それ以上の角度だと、進む方向に対してタイヤの角度がきつすぎタイヤがスリップしてうまく曲がれない可能性があります。</p> <p>モータの回転はどうしましょうか。左に曲がるので、左モータを少なく、右モータを多めにすることは予想できます。実際に何%にするか、やってみないと分からないのでとりあえず、左モータ 10%、右モータ 50%にしておきます。まとめると下記ようになります。</p> <p><b>ハンドル:-38 度</b>  <b>左モータ:10% 右モータ:50%</b>  <b>その後、パターン 31 へ移ります。</b></p>
 <p style="text-align: center;">→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><b>ハンドル:38 度</b>  <b>左モータ:50% 右モータ:10%</b>  <b>その後、パターン 41 へ移ります。</b></p>
 <p style="text-align: center;">→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。クランクを見つけたとき直角を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なります。まとめると下記ようになります。</p> <p><b>ハンドル:0 度</b>  <b>左モータ:40% 右モータ:40%</b></p>

<p>→ 0 x 0 4</p> <p>→ 0 x 0 6</p> <p>→ 0 x 0 7</p> <p>→ 0 x 0 3</p>	<p>マイコンカーが左に寄ったときを考えています。</p> <p>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、これ以上センサ状態は増やさないでおきます。</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。小さすぎるとず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><b>ハンドル: 8 度</b>  <b>左モータ: 40% 右モータ: 35%</b></p>
<p>→ 0 x 2 0</p> <p>→ 0 x 6 0</p> <p>→ 0 x e 0</p> <p>→ 0 x c 0</p>	<p>マイコンカーが右に寄ったときを考えています。</p> <p>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。小さすぎるとず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><b>ハンドル: -8 度</b>  <b>左モータ: 35% 右モータ: 40%</b></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。

プログラム化すると下記のようになります。

```

280 :     case 23:
281 :         /* クロスライン後のトレース、クランク検出 */
282 :         if( sensor_inp(MASK4_4)==0xf8 ) {
283 :             /* 左クランクと判断→左クランククリア処理へ */
284 :             led_out( 0x1 );
285 :             handle( -38 );
286 :             speed( 10 ,50 );
287 :             pattern = 31;
288 :             cnt1 = 0;
289 :             break;
290 :         }
291 :         if( sensor_inp(MASK4_4)==0x1f ) {
292 :             /* 右クランクと判断→右クランククリア処理へ */
293 :             led_out( 0x2 );
294 :             handle( 38 );
295 :             speed( 50 ,10 );
296 :             pattern = 41;
297 :             cnt1 = 0;
298 :             break;
299 :         }
300 :         switch( sensor_inp(MASK3_3) ) {
301 :             case 0x00:
302 :                 /* センタ→まっすぐ */
303 :                 handle( 0 );
304 :                 speed( 40 ,40 );
305 :                 break;
306 :             case 0x04:
307 :             case 0x06:
308 :             case 0x07:
309 :             case 0x03:
310 :                 /* 左寄り→右曲げ */
311 :                 handle( 8 );
312 :                 speed( 40 ,35 );
313 :                 break;
314 :             case 0x20:
315 :             case 0x60:
316 :             case 0xe0:
317 :             case 0xc0:
318 :                 /* 右寄り→左曲げ */
319 :                 handle( -8 );
320 :                 speed( 35 ,40 );
321 :                 break;
322 :         }
323 :         break;

```

case を続けて書くと  
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。

case を続けて書くと  
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。

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、if 文で「sensor\_inp(MASK3\_3)」とセンサ値を一回一回比較すると長くなるので、switch 文を使いました。

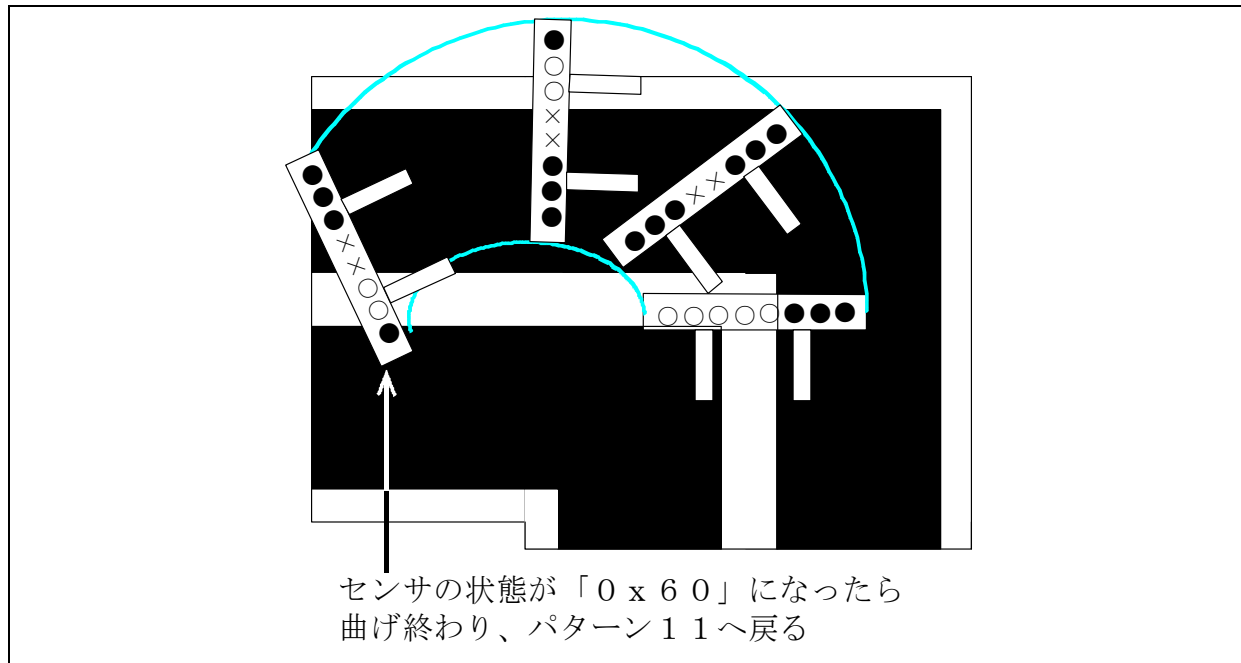


### 8.22.13 パターン 31、32: 左クランククリア処理

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。「いつまで左に大曲げをさせるか」ということです。

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」と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;
```

プログラムができました。実際に走らせてみました。そうすると、センサ状態が「0xf8」になった瞬間、ハンドルが左へ曲がり始めました。このまま曲げ続けるかと思いきや、すぐにまっすぐ向いてそのまま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手で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。

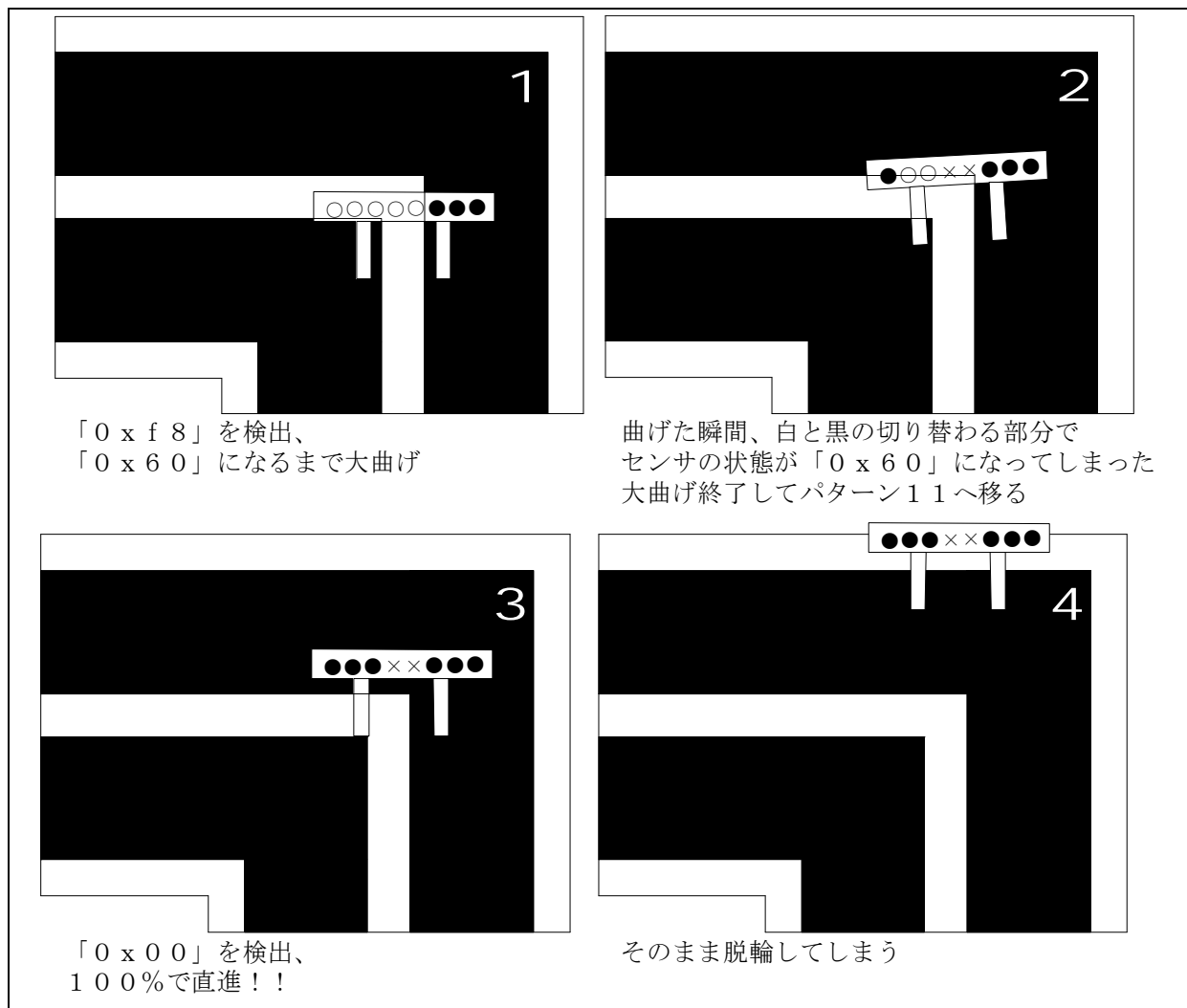
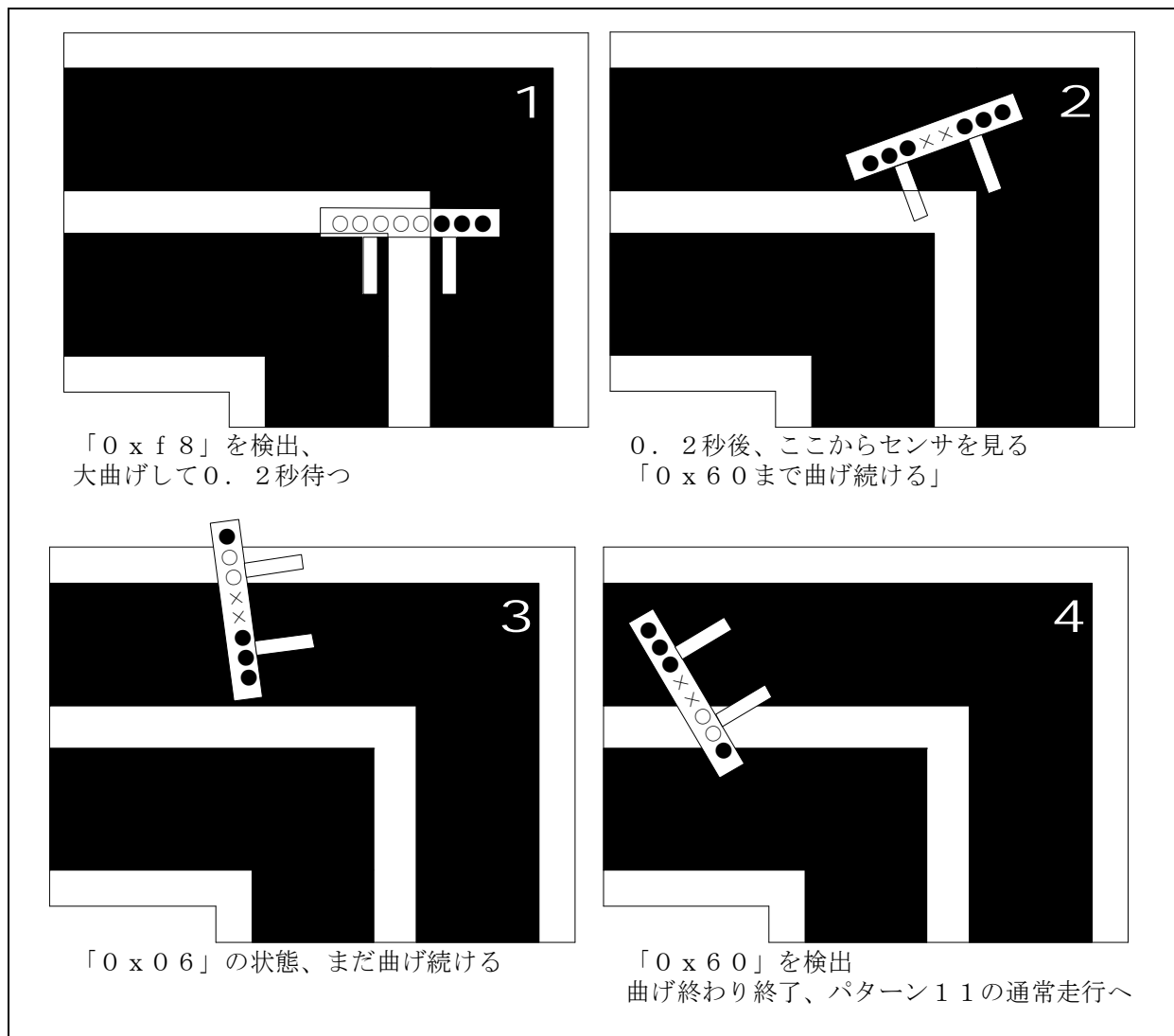


図 2 のように、白と黒の変わり目で(本当は白と灰と黒色ですが、灰色は白と見なします)センサの状態が「0x60」になっていることが分かりました。いちばん左のセンサ調整がうまくいっておらず先に”0”になっています。いちばん左のセンサ感度をもう少し強くすれば良いのですが、センサのちょっとした感度で脱輪するのは嫌なのでプログラムで何とかできないでしょうか。

考えてみると、「0xf8」を検出してから、終わりのセンサ状態「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た後、0.2 秒間は何もセンサを見ないように、ここでもタイマを使って少し進むのを待ちます。その後、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。



0.2秒後、図2のように白と黒の変わり目を越えた位置にあります。その後は「0x60」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。

```

325 :     case 31:
326 :         /* 左クランククリア処理 安定するまで少し待つ */
327 :         if( cnt1 > 200 ) {
328 :             pattern = 32;
329 :             cnt1 = 0;
330 :         }
331 :         break;
332 :
333 :     case 32:
334 :         /* 左クランククリア処理 曲げ終わりのチェック */
335 :         if( sensor_inp(MASK3_3) == 0x60 ) {
336 :             led_out( 0x0 );
337 :             pattern = 11;
338 :             cnt1 = 0;
339 :         }
340 :         break;

```

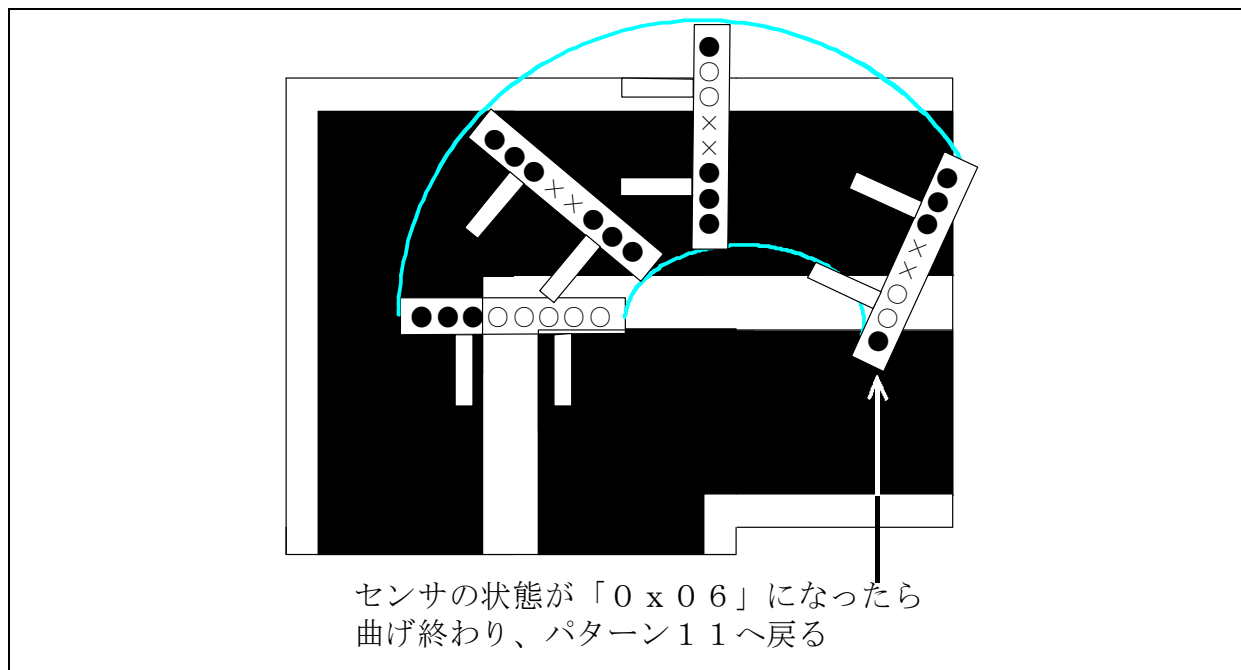
327行で cnt1 が 200 以上かチェックしています。200 以上なら、すなわち 0.2 秒たったならパターン 32 へ移ります。ちなみに cnt1 のクリアは、パターン 31 へ移る前の 288 行で行っています。

### 8.22.14 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てします。

次の問題が出てきます。「いつまで右に大曲げをさせるか」ということです。

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」と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;
```

実際に走らせてみました。そうすると、センサ状態が「0x1f」になった瞬間、ハンドルが右へ曲がり始めました。このまま曲げ続けるかと思いきや、すぐにまっすぐ向いてそのまま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手で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図のようになっていました。

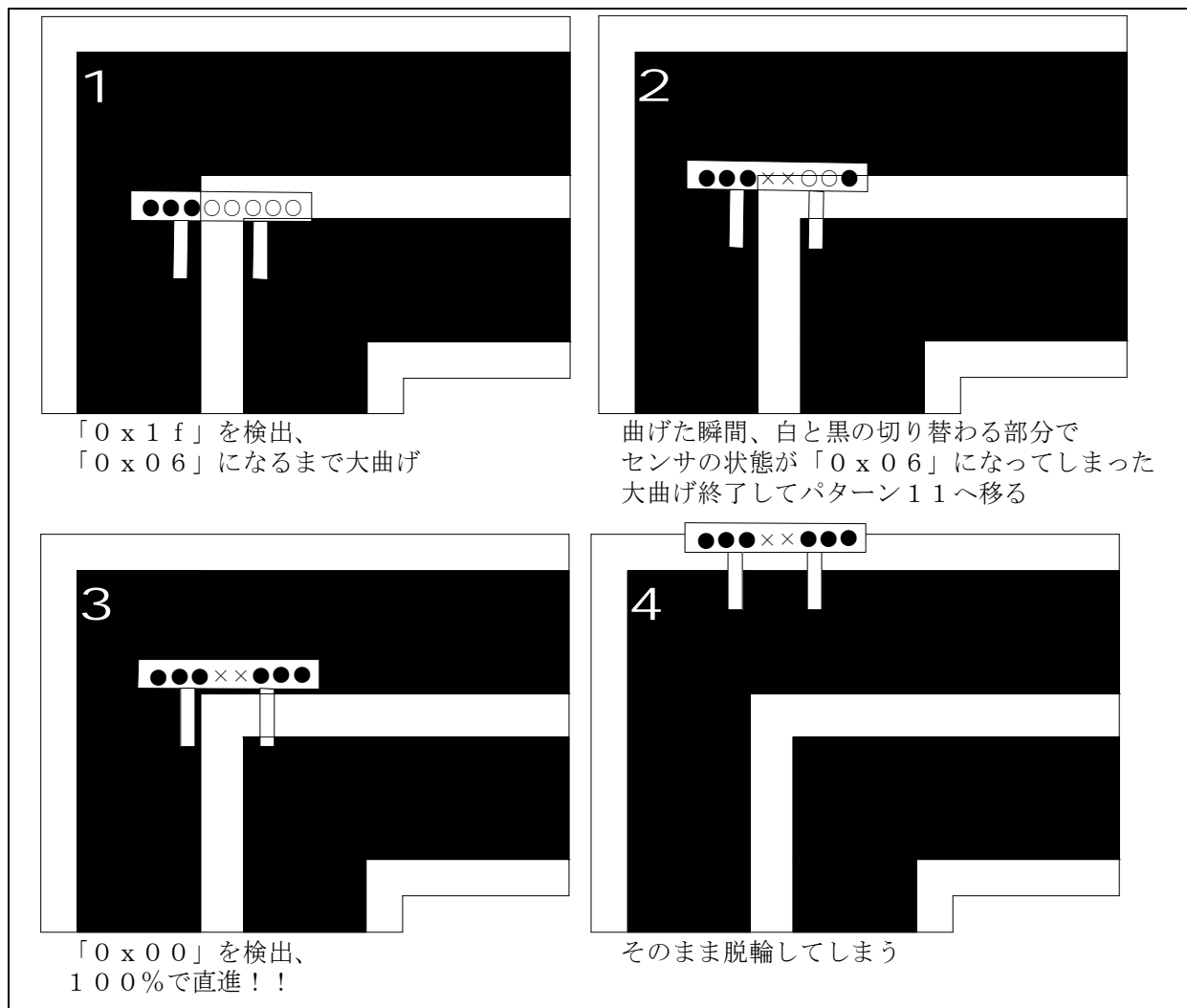
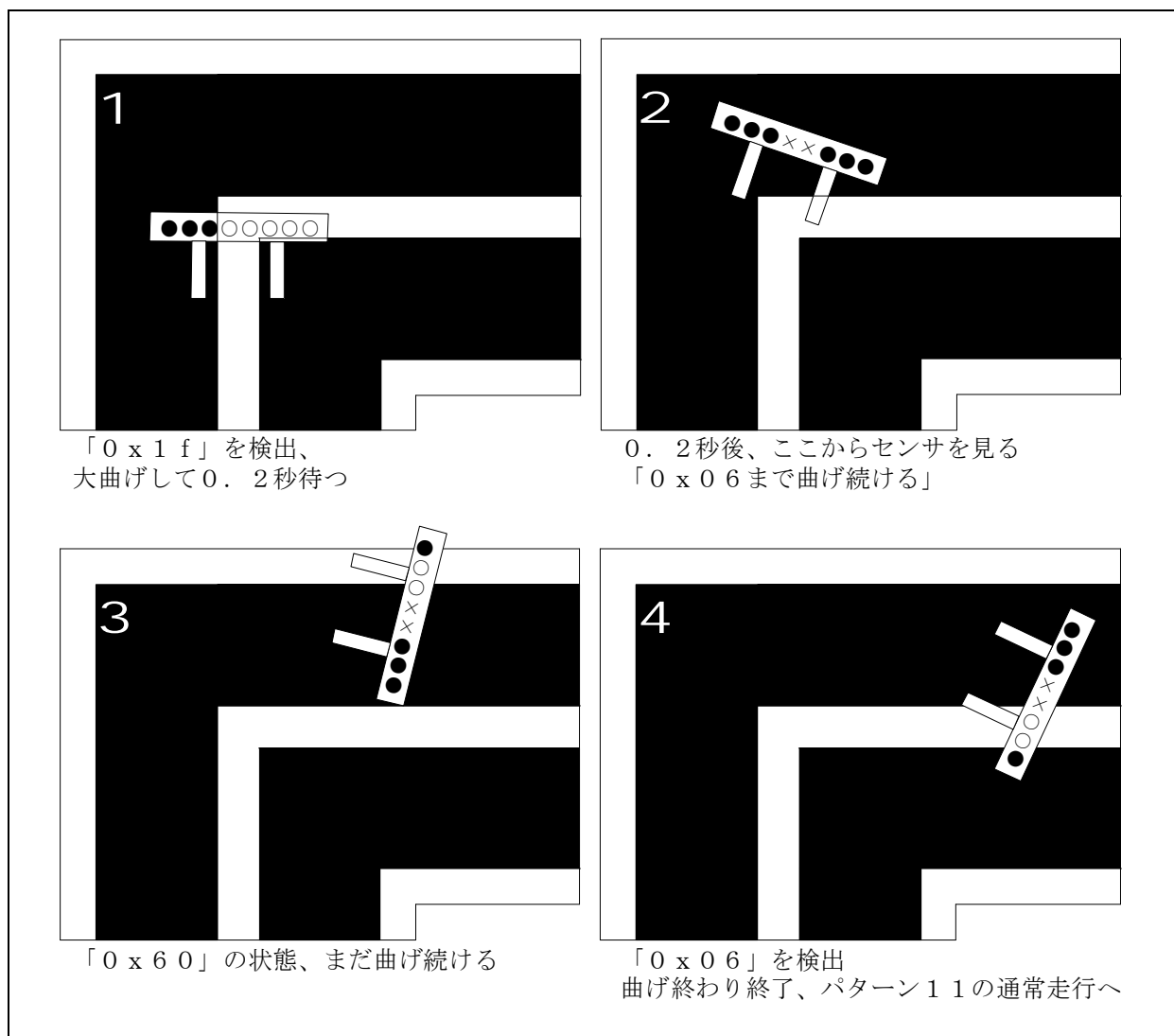


図 2 のように、白と黒の変わり目で(本当は白と灰と黒色ですが、灰色は白と見なします)センサの状態が「0x06」になっていることが分かりました。いちばん右のセンサ調整がうまくいっておらず先に”0”になっています。いちばん右のセンサ感度をもう少し強くすれば良いのですが、センサのちょっとした感度で脱輪するのは嫌なのでプログラムで何とかできないでしょうか。

考えてみると、「0x1f」を検出してから、終わりのセンサ状態「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た後、0.2 秒間は何もセンサを見ないように、ここでもタイマを使って少し進むのを待ちます。その後、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。



0.2秒後、図2のように白と黒の変わり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。

```

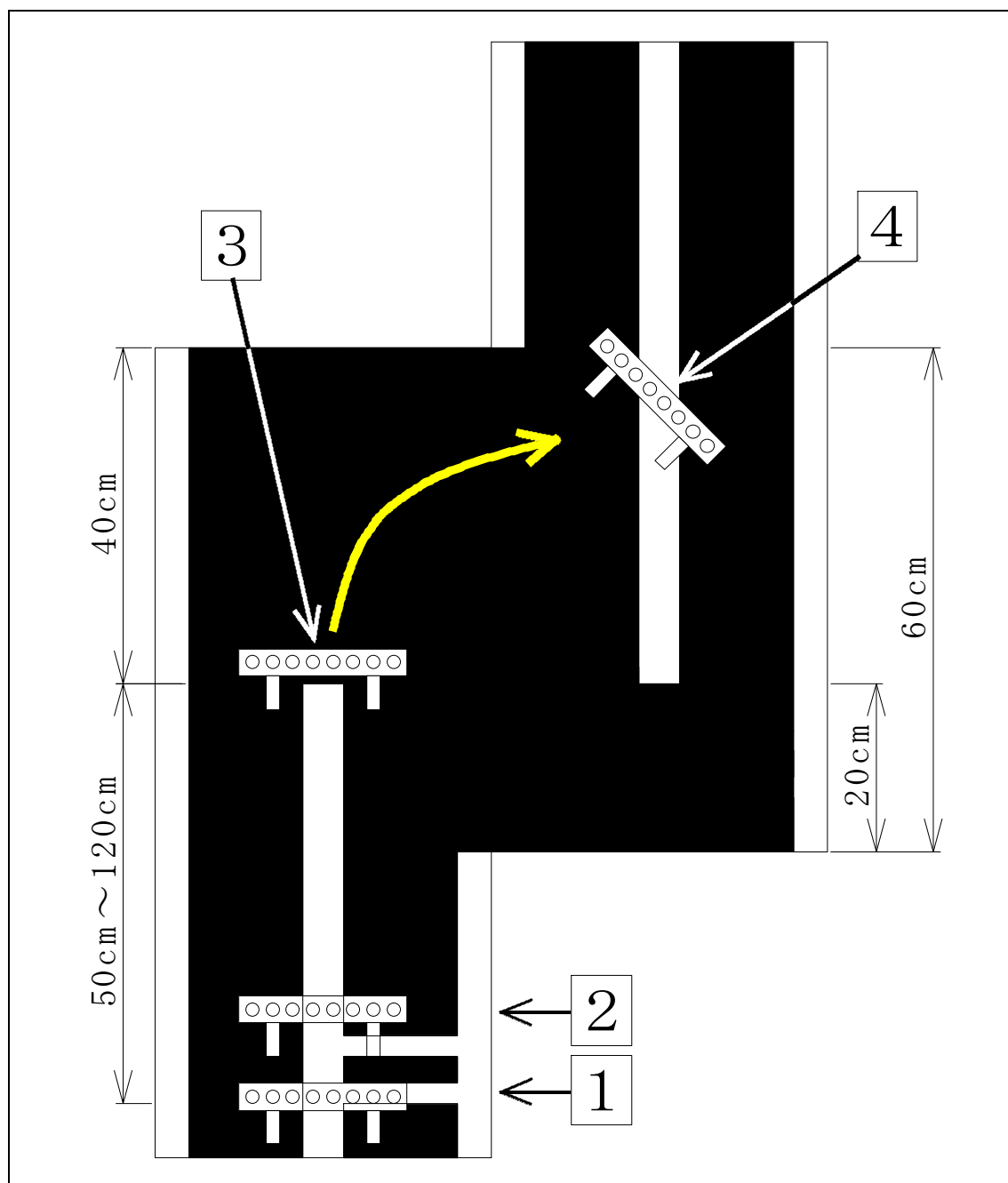
342 :     case 41:
343 :         /* 右クランククリア処理 安定するまで少し待つ */
344 :         if( cnt1 > 200 ) {
345 :             pattern = 42;
346 :             cnt1 = 0;
347 :         }
348 :         break;
349 :
350 :     case 42:
351 :         /* 右クランククリア処理 曲げ終わりのチェック */
352 :         if( sensor_inp(MASK3_3) == 0x06 ) {
353 :             led_out( 0x0 );
354 :             pattern = 11;
355 :             cnt1 = 0;
356 :         }
357 :         break;

```

344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297行で行っています。

### 8.22.15 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になっています。処理の概要は下図のようです。

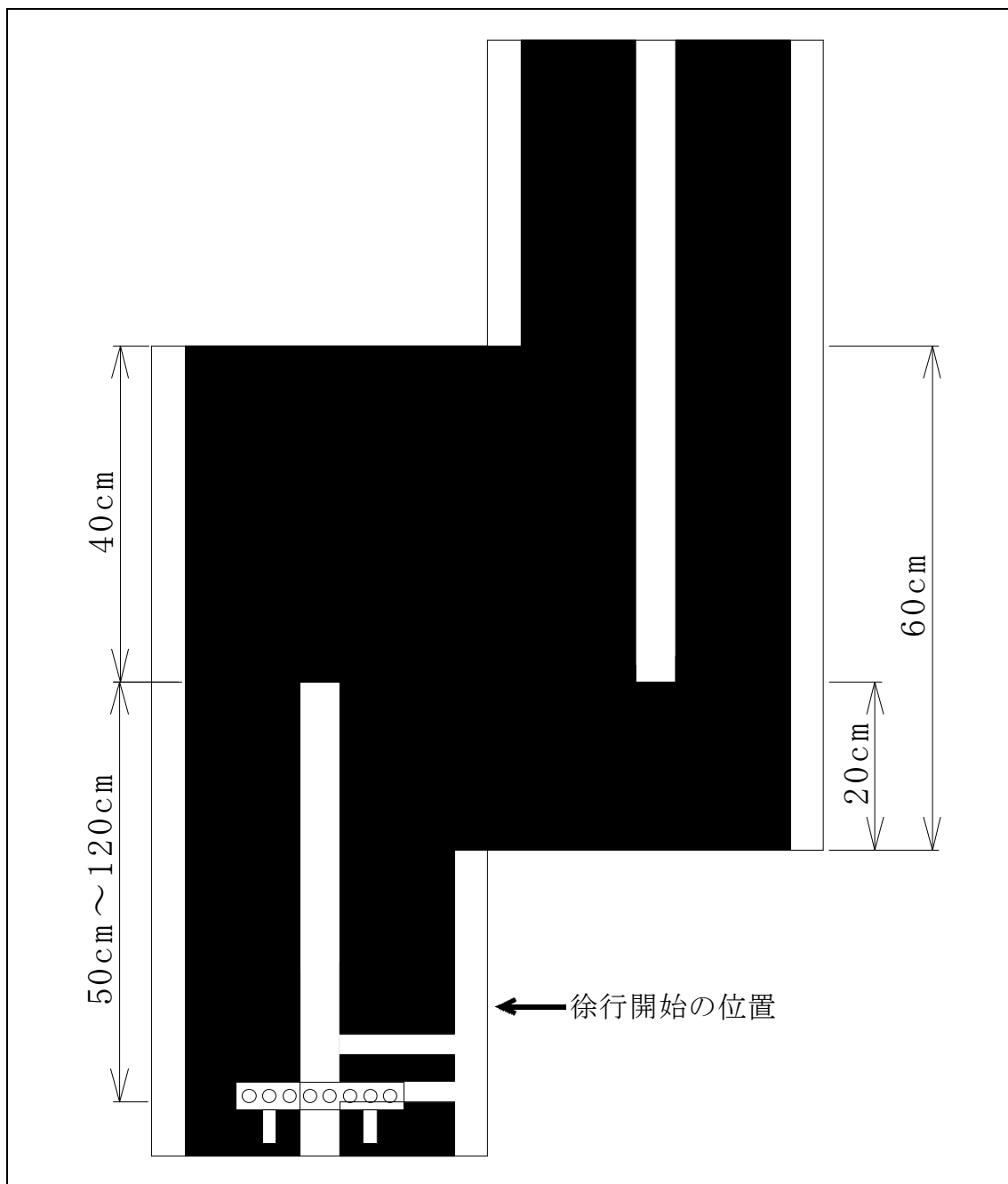


- 1 check\_rightline 関数で右ハーフラインを検出します。50cm~120cm 前で、右レーンに移動するために右に曲がらなければいけないのでブレーキをかけます。また、2本目の右ハーフラインでセンサが誤検出しないよう2の位置までセンサは見ません。
- 2 この位置から徐行開始します。中心線をトレースしながら進んでいきます。
- 3 中心線が無くなると、右へハンドルを切ります。
- 4 新しい中心線を検出すると、今度はこの中心線でライントレースを再開します。

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。

### 8.22.16 パターン 51: 1 本目の右ハーフライン検出時の処理

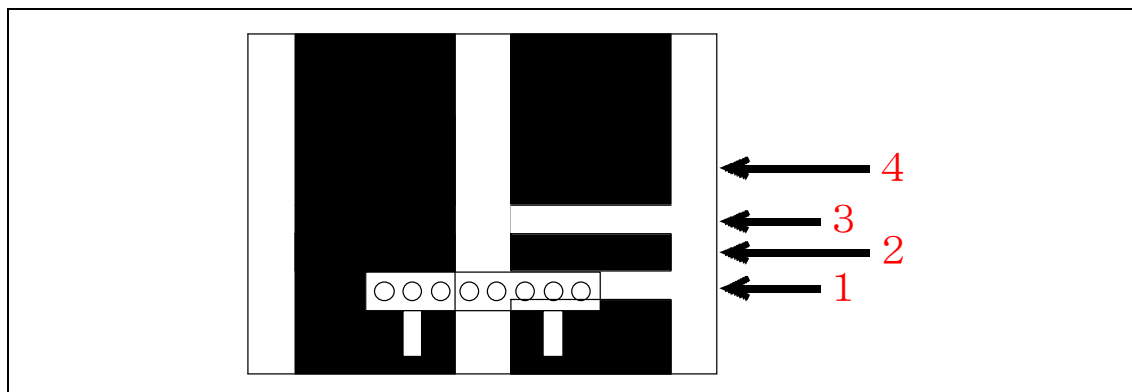
パターン 51 には、下図のような状態になった瞬間に移ってきます。



右ハーフライン後、50cm~120cm 先には、右レーンチェンジがあることを示しています。まず、何をすべきか。マイコンカーは、かなりスピードがついています。そのままのスピードでレーンチェンジするには無理な話です。まずブレーキをかけます。右ハーフライン後は、直線ということが分かっていますのでハンドルを0度にします。ブレーキは、「徐行開始の位置」までかけます。その後、徐行して進ませようと考えました。



最初の右ハーフラインを見つけてから徐行部分へ進むまでに下図のような状態があります。



1で1本目の右ハーフライン、2が黒、3が2本目の右ハーフライン、そして4が黒で徐行開始の部分です。コースが白→黒→白→黒と変化したことを検出して、4まで進んだか判別します。そして、4部分でブレーキを解除するプログラムが必要です。

ちょっと考え方を考えてみます。右ハーフラインを検出して4の位置まで進ませるとします。10cmくらいでしょうか。10cmくらいならタイマで少し時間稼ぎをすれば惰性で進み、難しいセンサ判断をせずに済みそうです。その時間は…これは実験してみないと何とも言えません。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とします。いっしょに、右ハーフラインを検出したとき、LEDを点灯させパターン51に入ったよ！ということを外部に知らせるようにします。

まとめると、

- LED1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
- 時間がたっ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以上になったら(100ミリ秒たったら)、パターン52へ移るようにしています。それはパターン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回目でcnt1は100以上と判断してしまい、0.1秒どころかほとんどパターン51が実行されません。cnt1が0である保証はどこにもありません。そこで、パターンをもう一つ増やします。パターン51はブレーキをかけcnt1をクリア、パターン52は0.1秒たったかチェックする部分に分けます。

再度まとめると、

●パターン 51 で行うこと

- ・LED1 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

- ・**cnt1 が 100 以上になったかチェック**
- ・**なったら、パターンを次へ移す**

**ゴシック体(赤)**が変更した部分です。

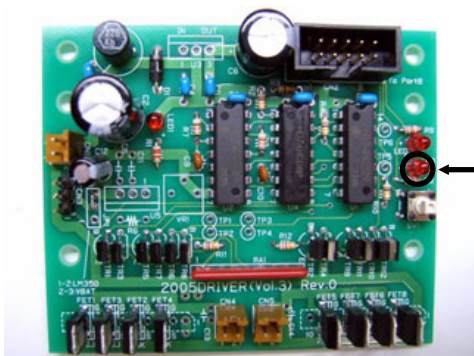
上記にしたがって再度プログラムを作ってみます。

```
359 :      case 51:
360 :          /* 1 本目の右ハーフライン検出時の処理 */
361 :          led_out( 0x2 );
362 :          handle( 0 );
363 :          speed( 0 , 0 );
364 :          pattern = 52;
365 :          cnt1 = 0;
366 :          break;
367 :
368 :      case 52:
369 :          /* 2 本目を読み飛ばす */
370 :          if( cnt1 > 100 ) {
371 :              pattern = 53;
372 :              cnt1 = 0;
373 :          }
374 :          break;
```

本当にこれでよいか見直してみます。パターン 51 でブレーキさせ、時間の基準である cnt1 をクリアしパターン 52 へ。パターン 52 では 0.1 秒たったかチェック。たったならパターン 53 へ。

これで右ハーフラインを検出してから徐行開始までのプログラムが完成しました。

**ポイント**



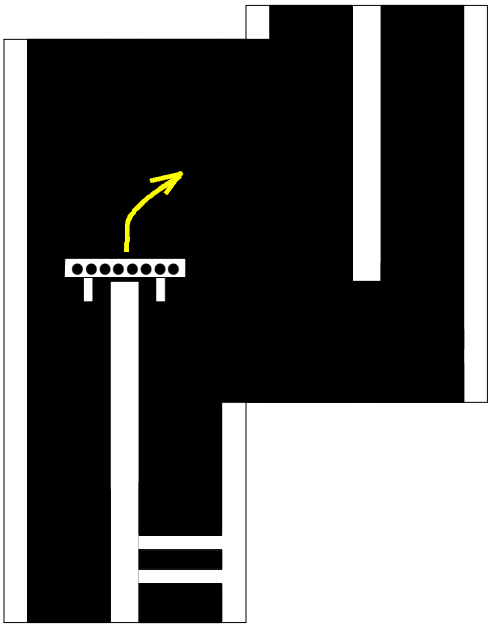
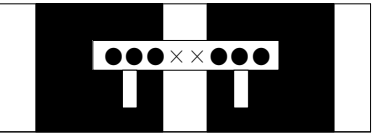
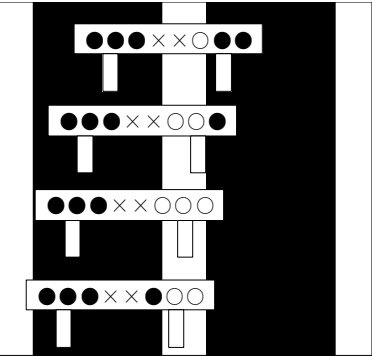
右ハーフラインを検出すると、モータドライブ基板の LED が 1 個点きます(下側)。  
点いていなければ、右ハーフラインを検出していません。

8.22.17 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 2 本の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っ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 style="text-align: center;">→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。</p> <p>このとき、サーボの切れ角、モータの回転はどうしましょうか。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のは予想できます。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やってみないと分かりません。とりあえず、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><b>ハンドル: 15 度</b> <b>左モータ: 40% 右モータ: 31%</b></p> <p><b>その後、パターン 54 へ移ります。</b></p>
 <p style="text-align: right;">→ 0 x 0 0</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。中心線が無くなっ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なります。まとめると下記ようになります。</p> <p><b>ハンドル: 0 度</b> <b>左モータ: 40% 右モータ: 40%</b></p>
 <p style="text-align: right;">→ 0 x 0 4 → 0 x 0 6 → 0 x 0 7 → 0 x 0 3</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ため、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、左へ寄っているため右へハンドルを切ります。小さすぎるとず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。まとめると下記ようになります。</p> <p><b>ハンドル: 8 度</b> <b>左モータ: 40% 右モータ: 35%</b></p>

	<p>→ 0 x 2 0</p> <p>→ 0 x 6 0</p> <p>→ 0 x e 0</p> <p>→ 0 x c 0</p>	<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分かっているの、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、右へ寄っているの左へハンドルを切ります。小さすぎるとず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 度します。まとめると下記のようになります。</p> <p><b>ハンドル:-8 度</b>  <b>左モータ:35% 右モータ:40%</b></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ようになります。

```

376 :     case 53:
377 :         /* 右ハーフライン後のトレース、レーンチェンジ */
378 :         if( sensor_inp(MASK4_4) == 0x00 ) {
379 :             handle( 15 );
380 :             speed( 40 , 31 );
381 :             pattern = 54;
382 :             cnt1 = 0;
383 :             break;
384 :         }
385 :         switch( sensor_inp(MASK3_3) ) {
386 :             case 0x00:
387 :                 /* センタ→まっすぐ */
388 :                 handle( 0 );
389 :                 speed( 40 , 40 );
390 :                 break;
391 :             case 0x04:
392 :             case 0x06:
393 :             case 0x07:
394 :             case 0x03:
395 :                 /* 左寄り→右曲げ */
396 :                 handle( 8 );
397 :                 speed( 40 , 35 );
398 :                 break;
399 :             case 0x20:
400 :             case 0x60:
401 :             case 0xe0:
402 :             case 0xc0:
403 :                 /* 右寄り→左曲げ */
404 :                 handle( -8 );
405 :                 speed( 35 , 40 );
406 :                 break;
407 :             default:
408 :                 break;
409 :         }
410 :         break;

```

case を続けて書くと  
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。

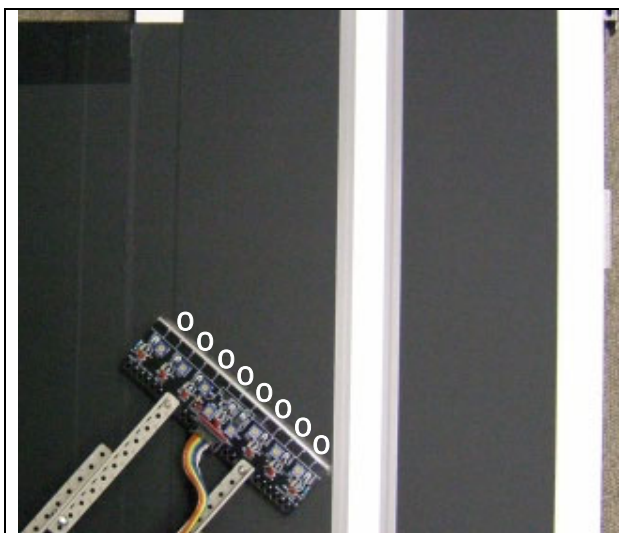
case を続けて書くと  
0x20 または 0x60 または 0xe0 または 0xc0 のとき  
という意味になります。

### 8.22.18 パターン 54: 右レーンチェンジ終了のチェック

ここまで、

1. 右ハーフライン検出
2. 徐行して進む
3. 中心線が無くなったことを検出して右へ曲げる

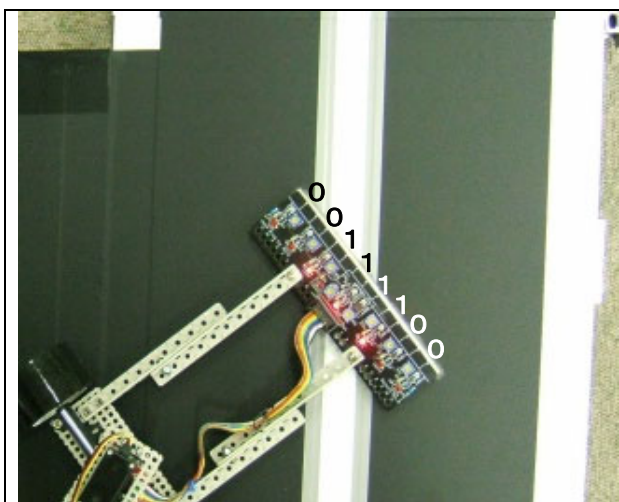
処理を行いました。次は、右側にある新しい中心線までいきます。新しい中心線を見つけたら、その中心線をトレースしていきます。これで右レーンチェンジ処理は完了です。では、新しい中心線と見なすセンサ状態はどのような状態でしょうか。



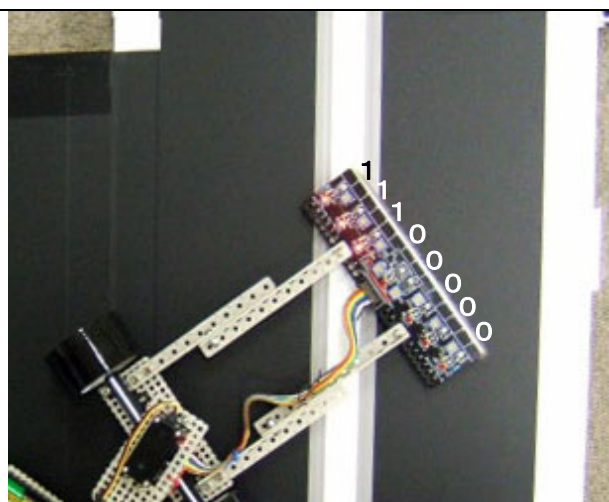
1. 新しい中心線を見つける直前です。



2. センサの右端で検出しました。  
センサの状態は、「0000 0111」です。



3. センサの中心まで来ました。  
センサの状態は、「0011 1100」です。



4. センサの左端まで来ました。  
センサの状態は、「1111 0000」です。

このように、センサの反応が変わっていきます。どの状態で、新しい中心線に来たと判断するのでしょうか。一番考えやすいのはやはりセンサの中心に来たときでしょうか。センサの状態が「0011 1100」になったときです。

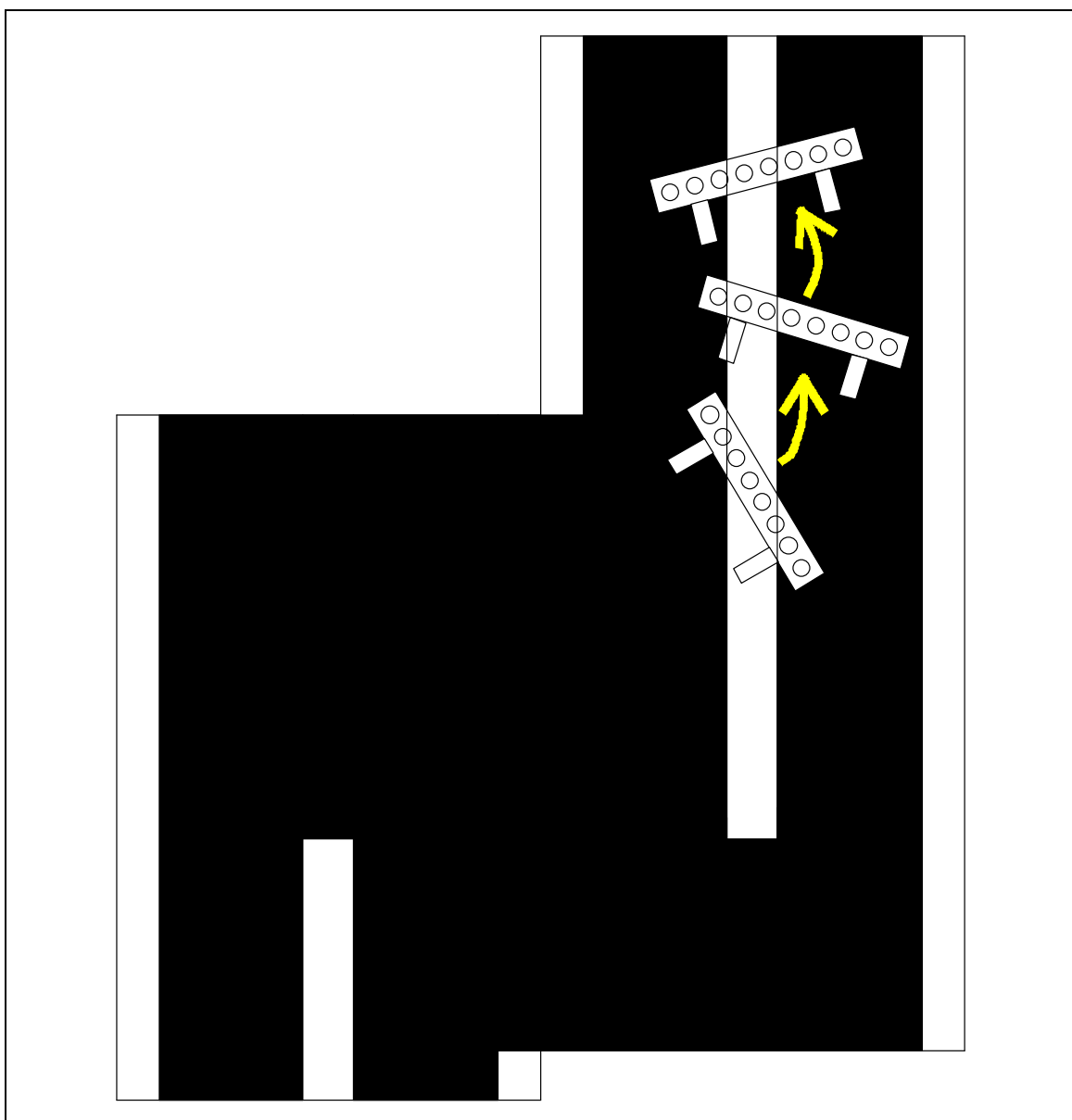
プログラムは、センサ 8 個チェックしたとき、「0011 1100」になったなら通常トレースであるパターン 11 へ移りなさい、とします。

```

412 :     case 54:
413 :         /* 右レーンチェンジ終了のチェック */
414 :         if( sensor_inp( MASK4_4 ) == 0x3c ) {
415 :             led_out( 0x0 );
416 :             pattern = 11;
417 :             cnt1 = 0;
418 :         }
419 :         break;

```

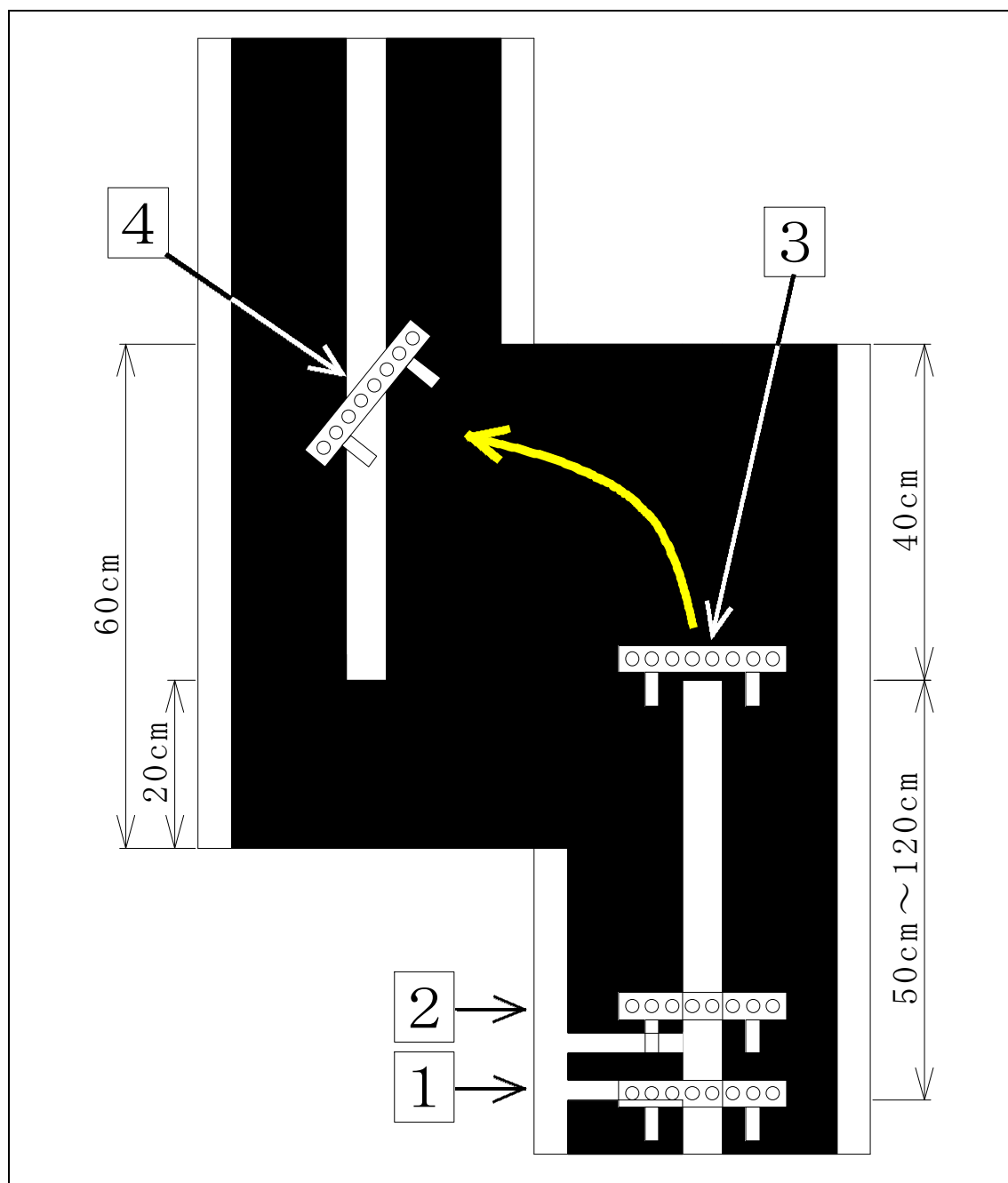
右ハーフラインを検出したときにLEDを点灯させたので、415行で消灯させてからパターン11へ移るようにします。



中心線を見つけたときは角度がついていますが、パターン11の処理で中心に復帰していきます。

### 8.22.19 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。処理の概要は下図のようです。

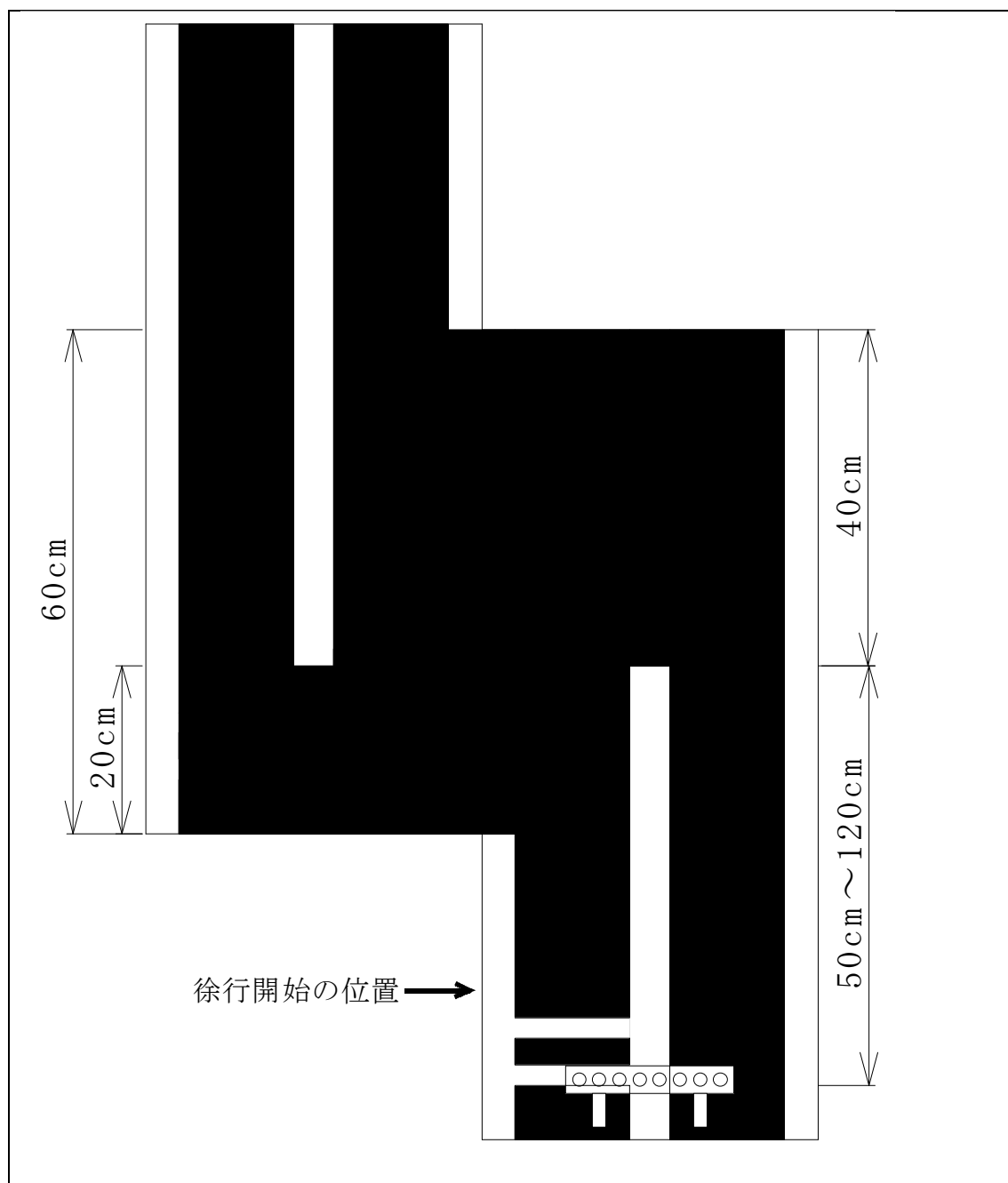


- 1 check\_leftline 関数で左ハーフラインを検出します。50cm~120cm 先で、左レーンに移動するために左に曲がらなければいけないのでブレーキをかけます。また、2本目の左ハーフラインでセンサが誤検出しないよう2の位置までセンサは見ません。
- 2 この位置から徐行開始します。中心線をトレースしながら進んでいきます。
- 3 中心線が無くなると、左へハンドルを切ります。
- 4 新しい中心線を検出すると、今度はこの中心線でライントレースを再開します。

このように、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。

### 8.22.20 パターン 61: 1 本目の左ハーフライン検出時の処理

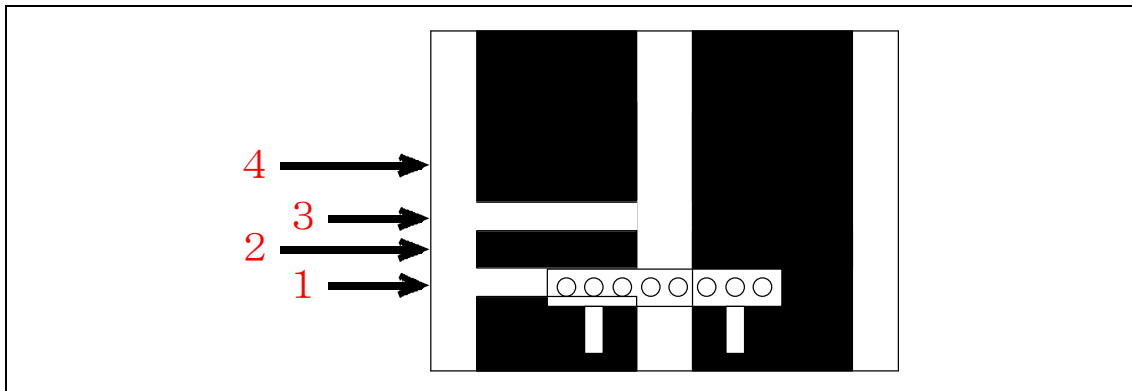
パターン 61 には、下記のような状態になった瞬間に移ってきます。



左ハーフライン後、50cm~120cm 先には、左レーンチェンジがあることを示しています。まず、何をすべきか。マイコンカーは、かなりスピードがついています。そのままのスピードでレーンチェンジするには無理な話です。まずブレーキをかけます。左ハーフライン後は、直線ということが分かっていますのでハンドルを 0 度にします。ブレーキは、「徐行開始の位置」までかけます。その後、徐行して進ませようと考えました。



最初の左ハーフラインを見つけてから徐行部分へ進むまでに下図のような状態があります。



1で1本目の左ハーフライン、2が黒、3が2本目の左ハーフライン、そして4が黒で徐行開始の部分です。コースが白→黒→白→黒と変化したことを検出して、4まで進んだか判別します。そして、4部分でブレーキを解除するプログラムが必要です。なんだか複雑そうです。

ちょっと考え方を考えてみます。左ハーフラインを検出して4の位置まで進ませるとします。10cmくらいでしょうか。10cmくらいならタイマで少し時間稼ぎをすれば惰性で進み、難しいセンサ判断をせずに済みそうです。その時間は…これは実験してみないと何とも言えません。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とします。いっしょに、左ハーフラインを検出したとき、LEDを点灯させパターン61に入ったよ！ということを外部に知らせるようにします。

まとめると、

- LED0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
- 時間が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以上になったら(100ミリ秒たったら)、パターン62へ移るようにしています。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であったら、1回目でcnt1は100以上と判断してしまい、0.1秒どころかほとんどパターン61が実行されません。cnt1が0である保証はどこにもありません。そこで、パターンをもう一つ増やします。パターン61はブレーキをかけcnt1をクリア、パターン62は0.1秒たったかチェックする部分に分けます。

再度まとめると、

●パターン 61 で行うこと

- ・LED0 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

- ・**cnt1 が 100 以上になったかチェック**
- ・**なったら、パターンを次へ移す**

**ゴシック体(赤)**が変更した部分です。

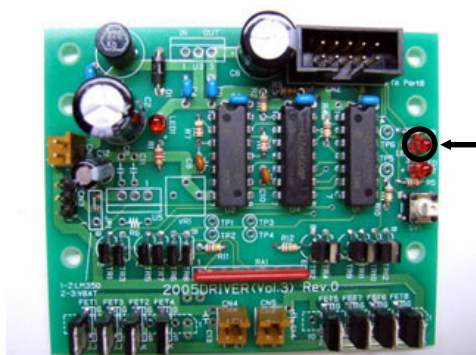
上記にしたがって再度プログラムを作ってみます。

```
421 :      case 61:
422 :          /* 1 本目の左ハーフライン検出時の処理 */
423 :          led_out( 0x1 );
424 :          handle( 0 );
425 :          speed( 0 , 0 );
426 :          pattern = 62;
427 :          cnt1 = 0;
428 :          break;
429 :
430 :      case 62:
431 :          /* 2 本目を読み飛ばす */
432 :          if( cnt1 > 100 ) {
433 :              pattern = 63;
434 :              cnt1 = 0;
435 :          }
436 :          break;
```

本当にこれでよいか見直してみます。パターン 61 でブレーキさせ、時間の基準である cnt1 をクリアしパターン 62 へ。パターン 62 では 0.1 秒たったかチェック。たったならパターン 63 へ。良さそうです。

これで左ハーフラインを検出してから徐行開始までのプログラムが完成しました。

**ポイント**



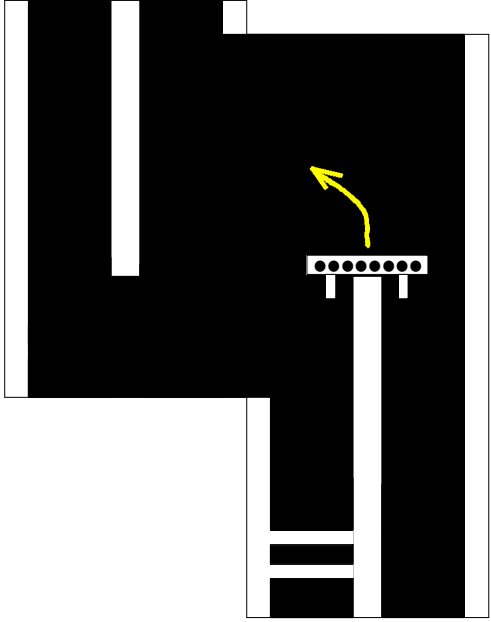
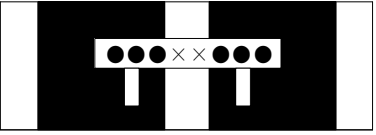
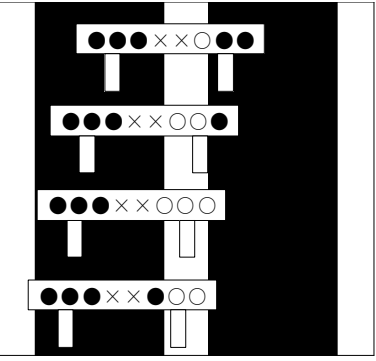
左ハーフラインを検出すると、モータドライブ基板の LED が 1 個点きます(上側)。  
点いていなければ、左ハーフラインを検出していません。

### 8.22.21 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 2 本の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っ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 style="text-align: center;">→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、左曲げを開始します。</p> <p>このとき、サーボの切れ角、モータの回転はどうしましょうか。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のは予想できます。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やってみないと分かりません。とりあえず、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><b>ハンドル:-15 度</b> <b>左モータ:31% 右モータ:40%</b></p> <p><b>その後、パターン 64 へ移ります。</b></p>
 <p style="text-align: right;">→ 0 x 0 0</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。ハンドルを真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。中心線が無くなっ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なります。まとめると下記ようになります。</p> <p><b>ハンドル:0 度</b> <b>左モータ:40% 右モータ:40%</b></p>
 <p style="text-align: right;">→ 0 x 0 4 → 0 x 0 6 → 0 x 0 7 → 0 x 0 3</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ため、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、左へ寄っているため右へハンドルを切ります。小さすぎるとず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。まとめると下記ようになります。</p> <p><b>ハンドル:8 度</b> <b>左モータ:40% 右モータ:35%</b></p>

	<p>→ 0 x 2 0</p> <p>→ 0 x 6 0</p> <p>→ 0 x e 0</p> <p>→ 0 x c 0</p>	<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分かっているの、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。小さすぎるとず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 度になります。</p> <p><b>ハンドル:-8 度</b>  <b>左モータ:35% 右モータ:40%</b></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ようになります。

```

438 :     case 63:
439 :         /* 左ハーフライン後のトレース、レーンチェンジ */
440 :         if( sensor_inp(MASK4_4) == 0x00 ) {
441 :             handle( -15 );
442 :             speed( 31 , 40 );
443 :             pattern = 64;
444 :             cnt1 = 0;
445 :             break;
446 :         }
447 :         switch( sensor_inp(MASK3_3) ) {
448 :             case 0x00:
449 :                 /* センタ→まっすぐ */
450 :                 handle( 0 );
451 :                 speed( 40 , 40 );
452 :                 break;
453 :             case 0x04:
454 :             case 0x06:
455 :             case 0x07:
456 :             case 0x03:
457 :                 /* 左寄り→右曲げ */
458 :                 handle( 8 );
459 :                 speed( 40 , 35 );
460 :                 break;
461 :             case 0x20:
462 :             case 0x60:
463 :             case 0xe0:
464 :             case 0xc0:
465 :                 /* 右寄り→左曲げ */
466 :                 handle( -8 );
467 :                 speed( 35 , 40 );
468 :                 break;
469 :             default:
470 :                 break;
471 :         }
472 :         break;

```

case を続けて書くと 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。

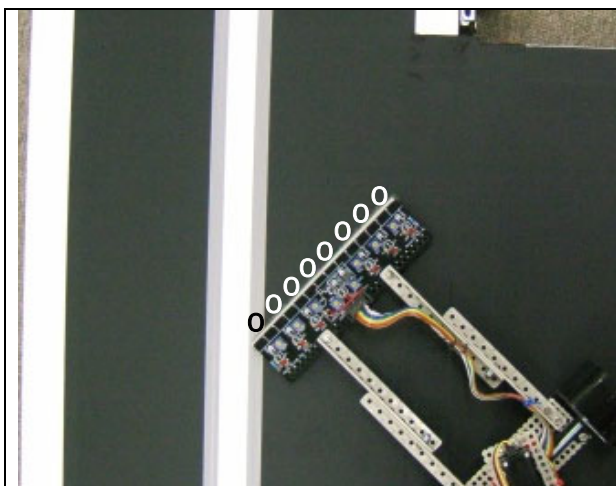
case を続けて書くと 0x20 または 0x60 または 0xe0 または 0xc0 のとき という意味になります。

## 8.22.22 パターン 64: 左レーンチェンジ終了のチェック

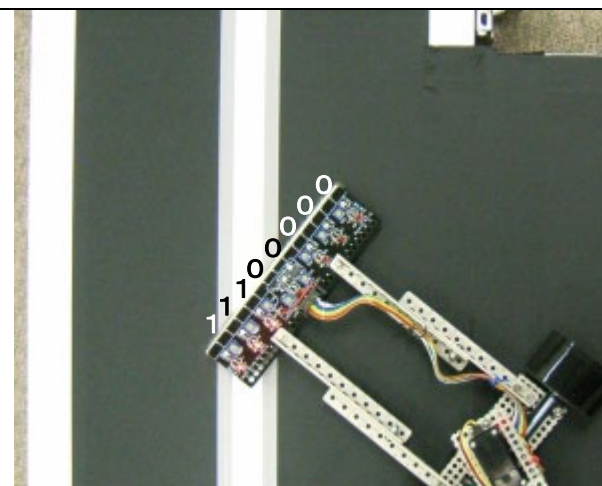
ここまで、

1. 左ハーフライン検出
2. 徐行して進む
3. 中心線が無くなったことを検出して左へ曲げる

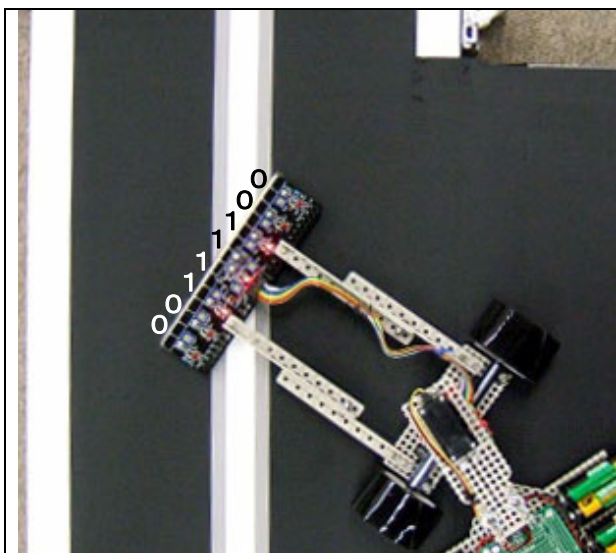
処理を行いました。次は、左側にある新しい中心線までいきます。新しい中心線を見つけたら、その中心線をトレースしていきます。これで左レーンチェンジ処理は完了です。では、新しい中心線と見なすセンサ状態はどのような状態でしょうか。



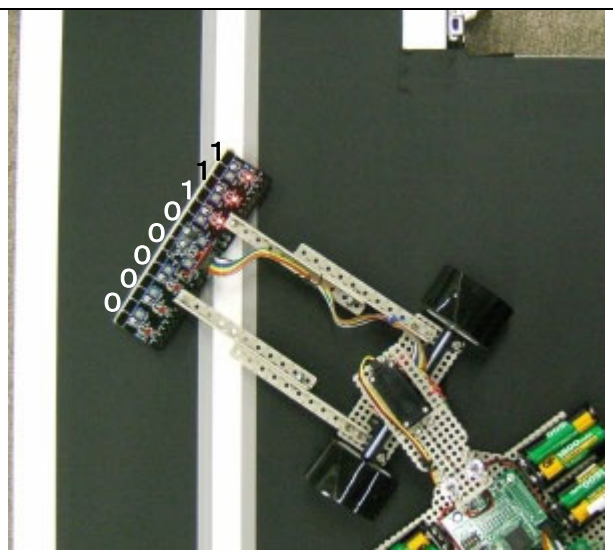
1. 新しい中心線を見つける直前です。



2. センサの左端で検出しました。  
センサの状態は、「1110 0000」です。



3. センサの中心まで来ました。  
センサの状態は、「0011 1100」です。



4. センサの右端まで来ました。  
センサの状態は、「0000 0111」です。

このように、センサの反応が変わっていきます。どの状態で、新しい中心線に来たと判断するのでしょうか。一番考えやすいのはやはりセンサの中心に来たときでしょうか。センサの状態が「0011 1100」になったときです。

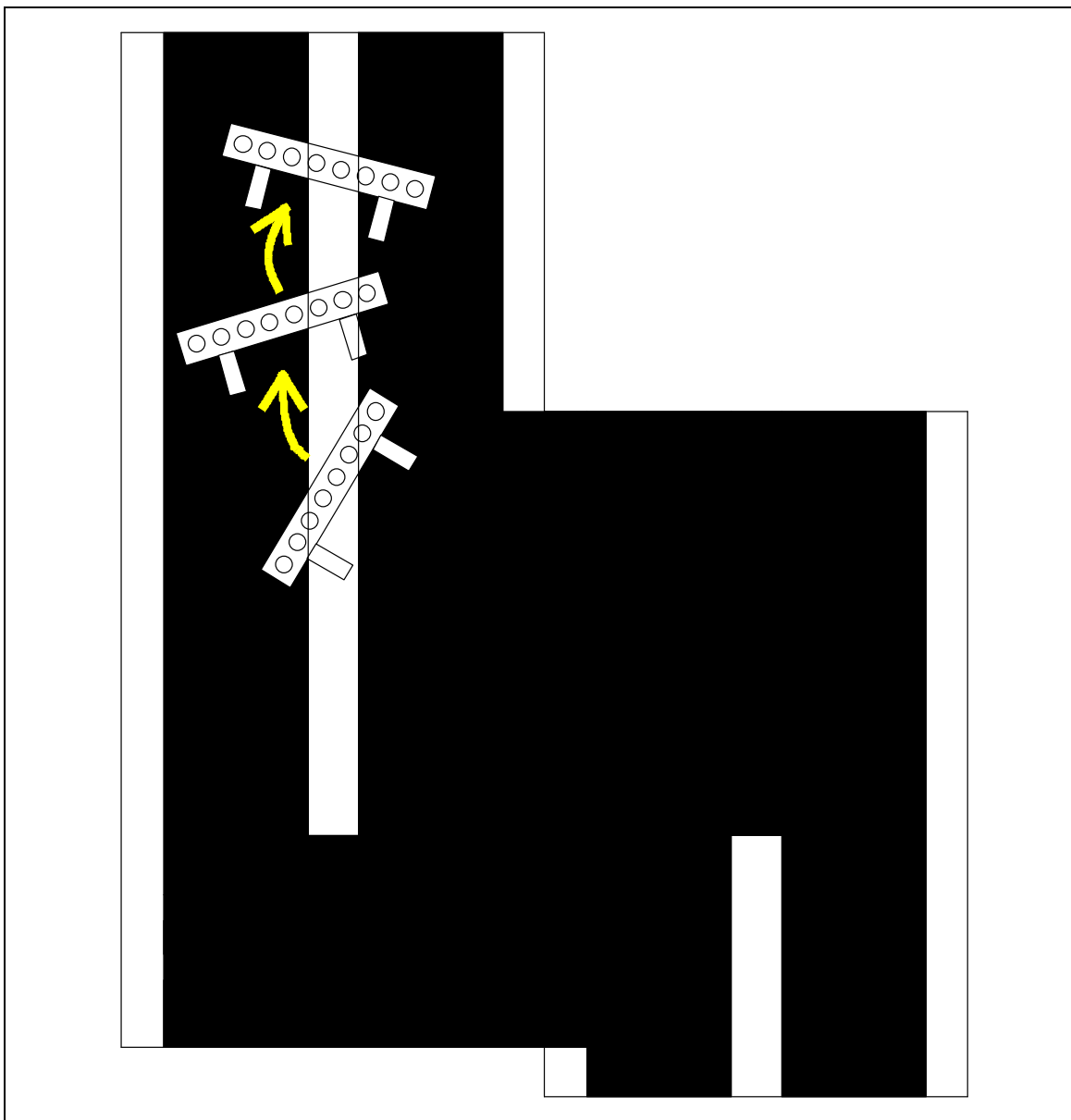
プログラムは、センサ 8 個チェックしたとき、「0011 1100」になったなら通常トレースであるパターン 11 へ移りなさい、とします。

```

474 :     case 64:
475 :         /* 左レーンチェンジ終了のチェック */
476 :         if( sensor_inp( MASK4_4 ) == 0x3c ) {
477 :             led_out( 0x0 );
478 :             pattern = 11;
479 :             cnt1 = 0;
480 :         }
481 :         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、477行で消灯させてからパターン11へ移るようにします。



中心線を見つけたときは角度がついていますが、パターン11の処理で中心に復帰していきます。

### 8.22.23 どれでもないパターン

```

483 :      default:
484 :          /* どれでもない場合は待機状態に戻す */
485 :          pattern = 0;
486 :          break;
    
```

もしパターンがどれでもない場合、default 文を実行します。パターンを 0 にして待機状態にします。default 文が実行されるということは、パターンを変えるときにどこかのプログラムで不定なパターンにしたということなので、その部分を探して直すようにします。

### 8.23 センサ基板Ver.4 にしたときのプログラム変更点

センサ基板をマイコンカーキット Vol.2 や Vol.3 付属の基板から、センサ基板 Ver.4 に差し替えたとき、プログラムは下記の 2 つの関数を差し替えれば対応できます。

関数名	kit06.c のプログラム	kit07.c のプログラム
sensor_inp	<pre> unsigned char sensor_inp( unsigned char mask ) {     unsigned char sensor;      sensor = P7DR;      /* 新センサ基板は、向かって左がbit0、右がbit7と前回センサ基板と */     /* 逆なので、互換を保つためビットを入れ替える */     sensor = bit_change( sensor ); /* ビット入れ替え */     sensor &amp;= mask;      return sensor; }         </pre>	<pre> unsigned char sensor_inp( unsigned char mask ) {     unsigned char sensor;  <b>sensor = ~P7DR;</b> <b>sensor &amp;= 0xef;</b> <b>if( sensor &amp; 0x08 ) sensor  = 0x10;</b>      sensor &amp;= mask;      return sensor; }         </pre>
startbar_get	<pre> unsigned char startbar_get( void ) {     unsigned char b;      b = ~PADR; /* スタートバー信号読み込み */     b &amp;= 0x08;     b &gt;&gt;= 3;      return b; }         </pre>	<pre> unsigned char startbar_get( void ) {     unsigned char b;  <b>b = ~P7DR; /* スタートバー信号読み込み */</b> <b>b &amp;= 0x10;</b> <b>b &gt;&gt;= 4;</b>      return b; }         </pre>

また、「8.12.6 注意点」(84ページ)にあるとおり、あり得ないセンサ状態があるので、sensor\_inp関数を使ってセンサの状態をチェックしている部分は見直しておきます。もし、ありえないセンサ状態があれば修正しておきましょう。

## 9. プログラム解説「kit07start.src」

このアセンブリソースプログラムは、ベクタアドレスやスタートアッププログラムが記述されています。

### 9.1 プログラムリスト

```

1 : ;=====
2 : ; 定義
3 : ;=====
4 : RESERVE: .EQU    H' FFFFFFFF          ; 未使用領域のアドレス
5 :
6 : ;=====
7 : ; 外部参照
8 : ;=====
9 :     .IMPORT _main
10 :     .IMPORT _INITSCT
11 :     .IMPORT _interrupt_timer0
12 :
13 : ;=====
14 : ; ベクタセクション
15 : ;=====
16 :     .SECTION V
17 :     .DATA L RESET_START                ; 0 H' 000000   リセット
18 :     .DATA L RESERVE                    ; 1 H' 000004   システム予約
19 :     .DATA L RESERVE                    ; 2 H' 000008   システム予約
20 :     .DATA L RESERVE                    ; 3 H' 00000c   システム予約
21 :     .DATA L RESERVE                    ; 4 H' 000010   システム予約
22 :     .DATA L RESERVE                    ; 5 H' 000014   システム予約
23 :     .DATA L RESERVE                    ; 6 H' 000018   システム予約
24 :     .DATA L RESERVE                    ; 7 H' 00001c   外部割り込み NMI
25 :     .DATA L RESERVE                    ; 8 H' 000020   トラップ 命令
26 :     .DATA L RESERVE                    ; 9 H' 000024   トラップ 命令
27 :     .DATA L RESERVE                    ; 10 H' 000028  トラップ 命令
28 :     .DATA L RESERVE                    ; 11 H' 00002c  トラップ 命令
29 :     .DATA L RESERVE                    ; 12 H' 000030   外部割り込み IRQ0
30 :     .DATA L RESERVE                    ; 13 H' 000034   外部割り込み IRQ1
31 :     .DATA L RESERVE                    ; 14 H' 000038   外部割り込み IRQ2
32 :     .DATA L RESERVE                    ; 15 H' 00003c   外部割り込み IRQ3
33 :     .DATA L RESERVE                    ; 16 H' 000040   外部割り込み IRQ4
34 :     .DATA L RESERVE                    ; 17 H' 000044   外部割り込み IRQ5
35 :     .DATA L RESERVE                    ; 18 H' 000048   システム予約
36 :     .DATA L RESERVE                    ; 19 H' 00004c   システム予約
37 :     .DATA L RESERVE                    ; 20 H' 000050   WDT MOV1
38 :     .DATA L RESERVE                    ; 21 H' 000054   REF CMI
39 :     .DATA L RESERVE                    ; 22 H' 000058   システム予約
40 :     .DATA L RESERVE                    ; 23 H' 00005c   システム予約
41 :     .DATA L _interrupt_timer0          ; 24 h' 000060   ITU0 IMIA0
42 :     .DATA L RESERVE                    ; 25 H' 000064   ITU0 IMIB0
43 :     .DATA L RESERVE                    ; 26 H' 000068   ITU0 OV10
44 :     .DATA L RESERVE                    ; 27 H' 00006c   システム予約
45 :     .DATA L RESERVE                    ; 28 H' 000070   ITU1 IMIA1
46 :     .DATA L RESERVE                    ; 29 H' 000074   ITU1 IMIB1
47 :     .DATA L RESERVE                    ; 30 H' 000078   ITU1 OV11
48 :     .DATA L RESERVE                    ; 31 H' 00007c   システム予約
49 :     .DATA L RESERVE                    ; 32 H' 000080   ITU2 IMIA2
50 :     .DATA L RESERVE                    ; 33 H' 000084   ITU2 IMIB2
51 :     .DATA L RESERVE                    ; 34 H' 000088   ITU2 OV12
52 :     .DATA L RESERVE                    ; 35 H' 00008c   システム予約
53 :     .DATA L RESERVE                    ; 36 H' 000090   ITU3 IMIA3
54 :     .DATA L RESERVE                    ; 37 H' 000094   ITU3 IMIB3
55 :     .DATA L RESERVE                    ; 38 H' 000098   ITU3 OV13
56 :     .DATA L RESERVE                    ; 39 H' 00009c   システム予約
57 :     .DATA L RESERVE                    ; 40 H' 0000a0   ITU4 IMIA4
58 :     .DATA L RESERVE                    ; 41 H' 0000a4   ITU4 IMIB4
59 :     .DATA L RESERVE                    ; 42 H' 0000a8   ITU4 OV14
60 :     .DATA L RESERVE                    ; 43 H' 0000ac   システム予約
61 :     .DATA L RESERVE                    ; 44 H' 0000b0   DMAC DEND0A
62 :     .DATA L RESERVE                    ; 45 H' 0000b4   DMAC DEND0B
63 :     .DATA L RESERVE                    ; 46 H' 0000b8   DMAC DEND1A
64 :     .DATA L RESERVE                    ; 47 H' 0000bc   DMCA DEND1B
65 :     .DATA L RESERVE                    ; 48 H' 0000c0   システム予約
66 :     .DATA L RESERVE                    ; 49 H' 0000c4   システム予約
67 :     .DATA L RESERVE                    ; 50 H' 0000c8   システム予約
68 :     .DATA L RESERVE                    ; 51 H' 0000cc   システム予約
69 :     .DATA L RESERVE                    ; 52 H' 0000d0   SCIO ERI0
70 :     .DATA L RESERVE                    ; 53 H' 0000d4   SCIO RXI0
71 :     .DATA L RESERVE                    ; 54 H' 0000d8   SCIO TXI0
72 :     .DATA L RESERVE                    ; 55 H' 0000dc   SCIO TEI0
73 :     .DATA L RESERVE                    ; 56 H' 0000e0   SCI1 ERI1
74 :     .DATA L RESERVE                    ; 57 H' 0000e4   SCI1 RXI1
75 :     .DATA L RESERVE                    ; 58 H' 0000e8   SCI1 TXI1
76 :     .DATA L RESERVE                    ; 59 H' 0000ec   SCI1 TEI1

```

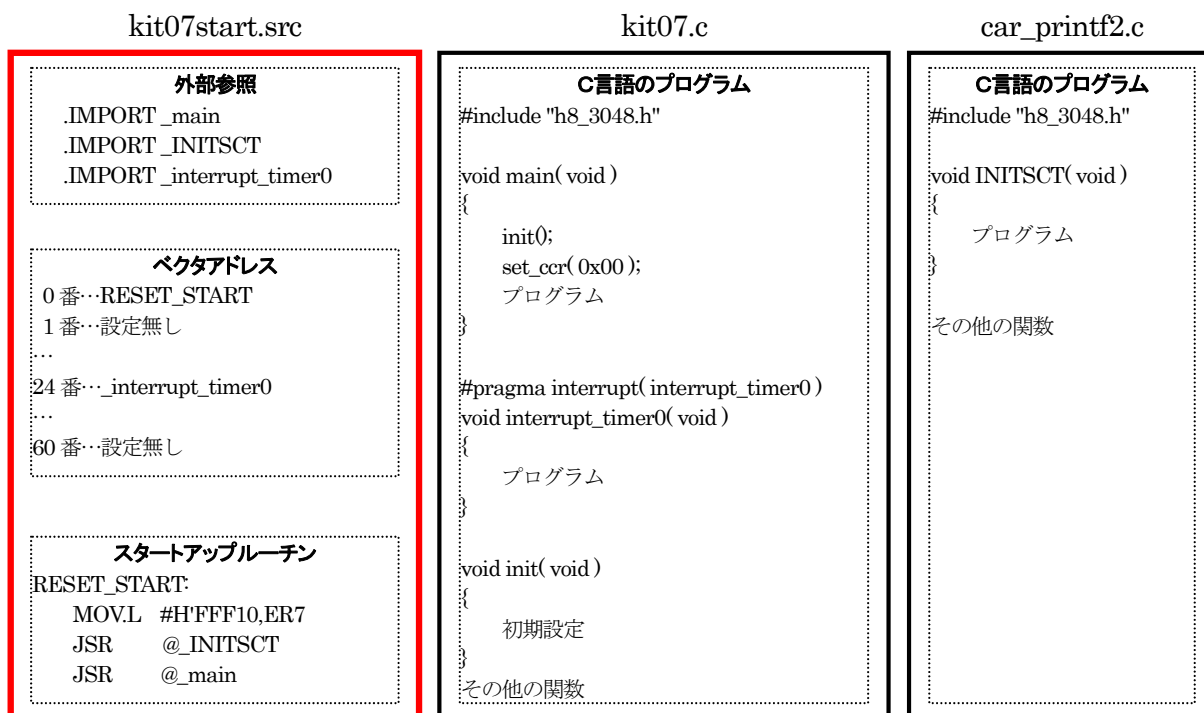


```

77 :          .DATA.L RESERVE          ; 60 H' 0000f0   A/D ADI
78 :
79 : ;=====
80 : ; スタートアッププログラム
81 : ;=====
82 :          .SECTION P
83 : RESET_START:
84 :     MOV.L  #H' FFF10, ER7          ; スタックの設定
85 :     JSR   @_INITSCT                ; セクションD, R, Bの設定
86 :     JSR   @_main                    ; C言語のmain()関数へジャンプ
87 : OWARI:
88 :     BRA   OWARI
89 :
90 :          .END
    
```

## 9.2 概要

プロジェクト「kit07」の構成を簡単に書くと下記ようになります。



- kit07start.src…… 外部参照、ベクタアドレスの設定、スタートアップルーチン
- kit07.c…………… C言語のメインプログラム
- car\_printf2.c …… printf、scanf 関数やセクション D,R,B を使用するために設定するプログラム

kit07start.src は、

- 外部参照
- ベクタアドレス
- スタートアップルーチン

が設定されているファイルです。

詳しくは、「5. プロジェクト内のファイルの関わりと実行順」を参照してください。

## 10. プログラム解説「car\_printf2.c」

### 10.1 プログラムリスト

「car\_printf2.c」ファイルは、「C:¥Workspace¥common」フォルダ内にあります。

```

1 : /******
2 : /* マイコンカー用printf, scanf使用プログラム Ver2 */
3 : /* 2006.04 ジャパンマイコンカーラリー実行委員会 */
4 : /******
5 :
6 : /*=====*/
7 : /* インクルード */
8 : /*=====*/
9 : #include <no_float.h> /* stdioの簡略化 最初に置く*/
10 : /*
11 : printf, scanf文でfloatやdouble型を使わなければ、stdio.hをインクルードする前に
12 : no_float.hをインクルードすることにより、MOTファイルサイズを小さくすることが
13 : 出来ます。もし、double型を使用するのであれば、インクルードしないでください。
14 : no_float.hはルネサス統合開発環境でのみ使用出来ます。
15 : */
16 : #include <stdio.h>
17 : #include <machine.h>
18 : #include "h8_3048.h"
19 :
20 : /*=====*/
21 : /* シンボル定義 */
22 : /*=====*/
23 : #define CHECK_PRINTFSCANF 1 /* printf, scanf使用するなら1*/
24 : #define SEND_BUFF_SIZE 64 /* 送信バッファサイズ */
25 : #define RECV_BUFF_SIZE 32 /* 受信バッファサイズ */
26 :
27 : /*=====*/
28 : /* グローバル変数の宣言 */
29 : /*=====*/
30 : #if CHECK_PRINTFSCANF
31 : /* 送信バッファ */
32 : char send_buff[SEND_BUFF_SIZE];
33 : char *send_w = send_buff;
34 : char *send_r = send_buff;
35 : unsigned int send_count = 0;
36 :
37 : /* 受信バッファ */
38 : char recv_buff[RECV_BUFF_SIZE];
39 : char *recv_w = recv_buff;
40 : char *recv_r = recv_buff;
41 :
42 : /* printf, scanfを使う為の変数設定 */
43 : unsigned char sml_buf[4];
44 : FILE _iob[] = { { &sml_buf[2], 0, &sml_buf[0], 3, _IOREAD, 0, 0 },
45 : { &sml_buf[3], 0, &sml_buf[3], 1, _IOWRITE|_IOUNBUF, 0, 1 } };
46 :
47 : volatile int _errno;
48 :
49 : #endif
50 :
51 : #if CHECK_PRINTFSCANF
52 : /******
53 : /* SCI1の初期化 */
54 : /* 引数 ポーレートレジスタ設定値 */
55 : /* 戻り値 なし */
56 : /******
57 : void init_sc1( int smr, int brr )
58 : {
59 :     int i;
60 :
61 :     SCI1_SCR = 0x00;
62 :     SCI1_SMR = smr;
63 :     SCI1_BRR = brr;
64 :     for( i=0; i<10000; i++ );
65 :     SCI1_SCR = 0x30; /* 送受信許可 */
66 :     SCI1_SSR &= 0x80; /* エラーフラグクリア */
67 : }
68 :
69 : /******
70 : /* 1文字受信 */
71 : /* 引数 受信文字格納アドレス */
72 : /* 戻り値 -1:受信エラー 0:受信なし 1:受信あり 文字は*sに格納 */
73 : /******
74 : int get_sci( char *s )
75 : {
76 :     int i;

```

```

77 :
78 :     if( SCI1_SSR & 0x38 ) {
79 :         /* 受信エラー */
80 :         SCI1_SSR &= 0xc7;
81 :         return -1;
82 :     } else if( SCI1_SSR & 0x40 ) {
83 :         /* 受信有り */
84 :         *s = SCI1_RDR;
85 :         SCI1_SSR &= 0xbf;
86 :         return 1;
87 :     }
88 :     /* 受信なし */
89 :     return 0;
90 : }
91 : /******
92 : /* 送信バッファに保存 */
93 : /* 引数 格納文字 */
94 : /* 戻り値 なし */
95 : /* メモ バッファがフルの場合、空くまで待ちます */
96 : /******
97 : void setSendBuff(char c)
98 : {
99 :     /* バッファが空くまで待つ */
100 :    while( SEND_BUFF_SIZE == send_count );
101 :
102 :    *send_w++ = c;
103 :    if( send_w >= send_buff+SEND_BUFF_SIZE ) send_w = send_buff;
104 :    send_count++;
105 :
106 :    /* 送信割り込み許可 */
107 :    SCI1_SCR |= 0x80;
108 : }
109 :
110 : /******
111 : /* 送信割り込み */
112 : /* 引数 なし */
113 : /* 戻り値 なし */
114 : /******
115 : #pragma interrupt (intTXI1)
116 : void intTXI1( void )
117 : {
118 :     /* 送信データをレジスタにセット */
119 :     SCI1_TDR = *send_r++;
120 :     if( send_r >= send_buff+SEND_BUFF_SIZE ) send_r = send_buff;
121 :
122 :     /* 送信開始 */
123 :     SCI1_SSR &= 0x7f;
124 :
125 :     /* これが最後のデータなら次以降の割り込み禁止 */
126 :     send_count--;
127 :     if( !send_count ) SCI1_SCR &= 0x7f;
128 : }
129 :
130 : /******
131 : /* printfで呼び出される関数 */
132 : /* ユーザーからは呼び出せません */
133 : /******
134 : char *sbrk(size_t size)
135 : {
136 :     return (char *)-1;
137 : }
138 :
139 : /******
140 : /* printfで呼び出される関数 */
141 : /* ユーザーからは呼び出せません */
142 : /******
143 : int write(int fileno, char *buf, unsigned int cnt)
144 : {
145 :     int i;
146 :     static int (*func)(const char *,...) = printf;
147 :
148 :     if( !cnt ) return 0;
149 :
150 :     if( *buf == '\n' ) {
151 :         setSendBuff( '\r' );
152 :     } else if( *buf == '\b' ) {
153 :         setSendBuff( '\b' );
154 :         setSendBuff( ' ' );
155 :     }
156 :     setSendBuff( *buf );
157 :     return cnt;
158 : }
159 :
160 : /******
161 : /* scanfで呼び出される関数 */
162 : /* ユーザーからは呼び出せません */
163 : /******
164 : int read(int fileno, char *buf, unsigned int cnt)
165 : {
166 :     static int (*func)(const char *,...) = scanf;
167 :

```

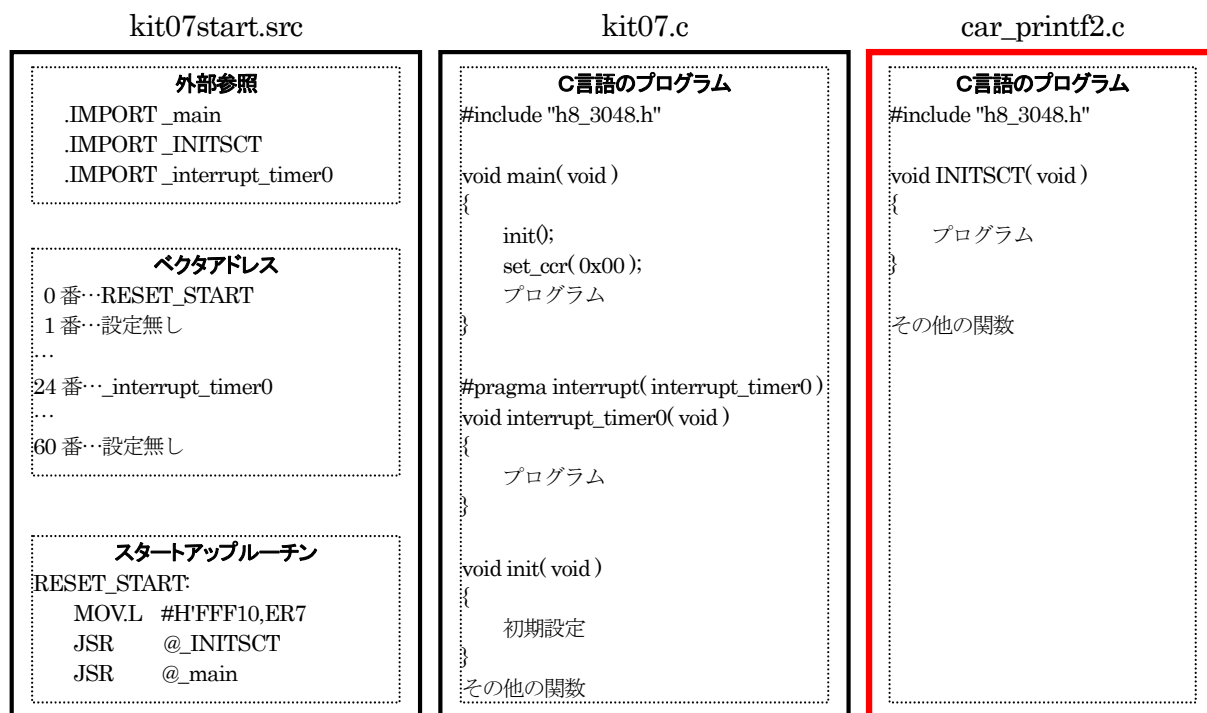
```

168 :     if( !cnt ) return 0;
169 :
170 :     if( recv_r == recv_w ) {
171 :         do {
172 :             /* 受信待ち */
173 :             while( !(SCI1_SSR & 0x40) )
174 :                 SCI1_SSR &= 0xc0;
175 :             *buf = SCI1_RDR;
176 :             SCI1_SSR &= 0xbf;
177 :
178 :             switch( *buf ) {
179 :                 case ' b': /* バックスペース */
180 :                     /* 何もバッファにないならBSは無効 */
181 :                     if( recv_r == recv_w ) continue;
182 :                     /* あるなら一つ戻る */
183 :                     recv_w--;
184 :                     break;
185 :                 case ' r': /* Enterキー */
186 :                     *recv_w++ = *buf = ' n';
187 :                     *recv_w++ = ' r';
188 :                     break;
189 :                 default:
190 :                     if( recv_w >= recv_buff+RECV_BUFF_SIZE-2 ) continue;
191 :                     *recv_w++ = *buf;
192 :                     break;
193 :             }
194 :             /* エコーバック 入力された文字を返す */
195 :             write( fileno, buf, cnt );
196 :         } while( *buf != ' n' );
197 :     }
198 :     *buf = *recv_r++;
199 :     if( recv_r == recv_w ) recv_r = recv_w = recv_buff;
200 :
201 :     return 1;
202 : }
203 :
204 : #endif
205 :
206 : /******
207 : /* RAMエリアの初期化
208 : /* 引数 なし
209 : /* 戻り値 なし
210 : /******
211 : void INITSCT( void )
212 : {
213 :     int *s, *e, *r;
214 :
215 :     r = __sectop("R"); /* Rセクション(RAM)の最初 */
216 :     s = __sectop("D"); /* Dセクション(ROM)の最初 */
217 :     e = __secend("D"); /* Dセクション(ROM)の最後 */
218 :     while( s < e ) {
219 :         *r++ = *s++; /* R ← D コピー */
220 :     }
221 :
222 :     s = __sectop("B"); /* Bセクション(RAM)の最初 */
223 :     e = __secend("B"); /* Bセクション(RAM)の最後 */
224 :     while( s < e ) {
225 :         *s++ = 0x00; /* B ← 0x00 */
226 :     }
227 : }
228 :
229 : /******
230 : /* end of file
231 : /******

```

## 10.2 概要

プロジェクト「kit07」の構成を簡単に書くと下記のようになります。



- kit07start.src…… 外部参照、ベクタアドレスの設定、スタートアップルーチン
- kit07.c…………… C言語のメインプログラム
- car\_printf2.c …… printf 関数、scanf 関数やセクション D,R,B を使用するために設定するプログラム

car\_printf2.c ファイルは

- 通信するための設定(SCI1 の初期設定)
  - printf 関数の出力先、scanf 関数の入力元を通信にするための設定
  - セクション D,R,B を初期化する設定
- が設定されています。

## 10.3 宣言されている関数

関数名	内容
init_sci1	<pre>void init_sci1( int smr, int brr )</pre> <p>printf、scanf 関数を使用するために通信の設定を行います。                  例) <code>init_sci1( 0x00, 79 );</code>                  詳しくは、H8/3048F-ONE 実習マニュアルの「<b>21. プロジェクト「sio」パソコンから数値を入力して LED に出力する</b>」(P230)を参照してください。</p>
INITSCT	<pre>void INITSCT( void )</pre> <p>セクション D,R,B の設定を行います。                  セクション D のデータをセクション R の位置にコピー、セクション B の領域を 0 でクリアします。セクション D,R を使用するプログラムは、必ず実行します。                  例) <code>INITSCT();</code></p>

intTXI1	<pre>void intTXI1( void ) printf 文を使うと、使われる割り込み関数です。直接の関数を呼び出すことはありません。 src ファイルのベクタアドレス 58 番にこの関数名を登録します。          .DATA.L    _intTXI1                ; 58 H' 0000e8    SCI1 TXI1</pre>
---------	--

また、直接関数を呼び出すことはしませんが、write、read という関数が定義されており、printf 関数の出力先、scanf 関数の入力元を通信にするための設定をしています。

## 10.4 シンボル定義

設定行数	内容
24 行	<pre>24 : #define          SEND_BUFF_SIZE  64          /* 送信バッファサイズ */</pre> <p>printf 関数を使用するとき、一度に貯めておける文字数を設定します。 小さいとプログラムの実行速度が遅くなります。大きいと RAM の消費が増えてプログラムが実行できなくなることがあります。問題がなければ、標準値の 64 としておきます。</p>
25 行	<pre>25 : #define          RECV_BUFF_SIZE  32          /* 受信バッファサイズ */</pre> <p>scanf 関数を使用するとき、一度に貯めておける文字数を設定します。 小さいと scanf 文で受信できる文字数が少なくなります。大きいと RAM の消費が増えてプログラムが実行できなくなることがあります。問題がなければ、標準値の 32 としておきます。</p>

### ●まとめ

car\_printf2.c ファイルは、printf 関数、scanf 関数やセクション D,R,B を使用するために設定するプログラムです。kit07.c など、main 関数のある C 言語ソースファイルと一っしょに使用します。

kit07.c では、セクション B を使用しています。そのため、kit07start.src ファイルで、INIT\_SCT 関数を実行します。これで、セクション B を使用する設定ができます。

## 11. プロジェクト「kit07」のツールチェーンの設定

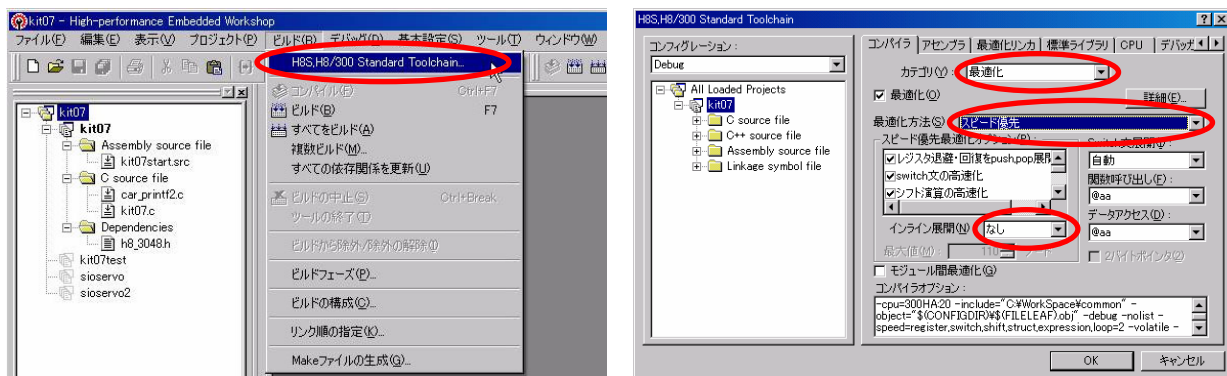
### 11.1 ツールチェーン

ツールチェーンという言葉は馴染みがないと思います。要は、**ビルドするときの設定のこと**です。実行委員会開発環境には sub ファイルと呼ばれるファイルがありました。kit05.sub は下記のようになっています。

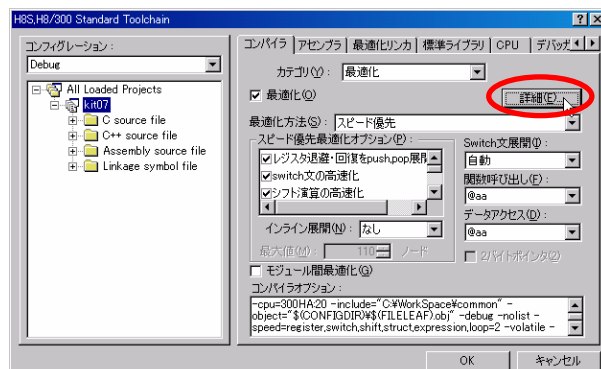
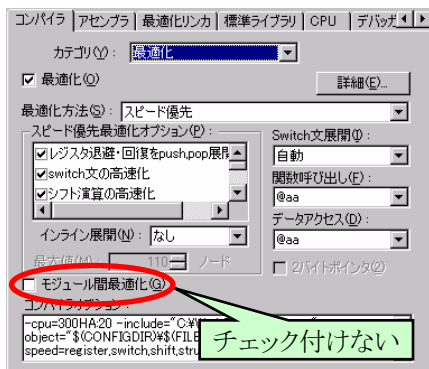
```
input kit05start, kit05
lib c:\¥h8n_win¥3048¥c¥c38hae.lib
output kit05
print kit05
start V(000000)
start P,C(000100)
start B(0fef10)
exit
```

この sub ファイルに当たる部分が、ルネサス統合開発環境ではツールチェーンになります。ここでは、主に変えなければいけない設定を紹介します。

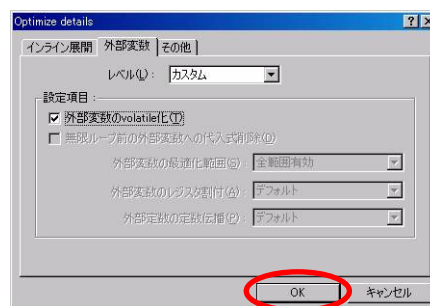
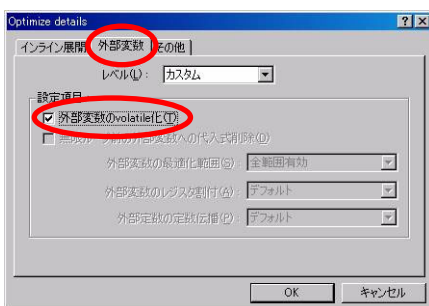
### 11.2 コンパイラの設定



1. 「ビルド→H8S,H8/300 Standard Toolchain」(ツールチェーン)を選択します。
2. 「カテゴリ:最適化」、「最適化方法:スピード優先」、「インライン展開:なし」を選択します。



- 「モジュール間最適化」のチェックは**付けません**。最適化という文字に惑わされてチェックを付けると良くなりそうですが、チェックを付けたらプログラムが暴走することがあります(チェックをつけた場合は、さらに設定が必要です)。
- 詳細**をクリックします。



- 「外部変数の volatile 化」のチェックを付けます。  
※volatile についての詳しい説明は下記を参照してください。
- OK**をクリックします。

**※参考資料—volatile について**

コンパイラはプログラムをコンパイルするとき最適化と呼ばれる作業を行います。これは、無駄な部分を省いてプログラムのサイズを小さく、実行速度を速くするようにします。

例えば、下記のプログラムを作ったとします。コンパイルするとどうなるでしょうか。

```
PADR = 0x55;
PADR = 0xaa;
PADR = 0x00;
```

コンパイル



```
PADR = 0x00;
```

ポート A にデータを3回連続して書き込んでみます。コンパイラは、同じ場所に連続して値を書き込んでいるので、最終的な結果は PADR に 0x00 を書き込めばいいだろう、と勝手に解釈してしまいます。本来は、ポート A に出力する信号を連続して変化させたいのですが、そうなりません。

そこで、コンパイルのオプションで「外部変数の volatile 化」にチェックを付けます。**このチェックを付けることにより最適化を行わないようにします。**

```
PADR = 0x55;
PADR = 0xaa;
PADR = 0x00;
```

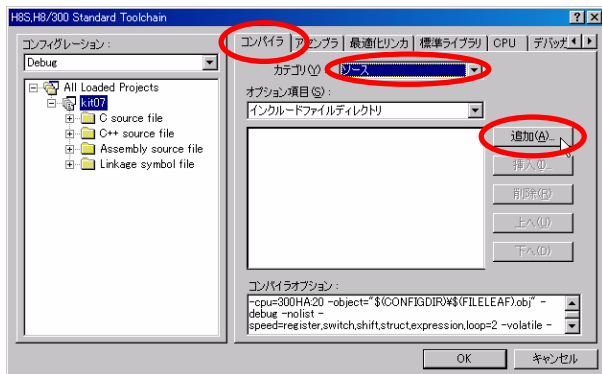
コンパイル



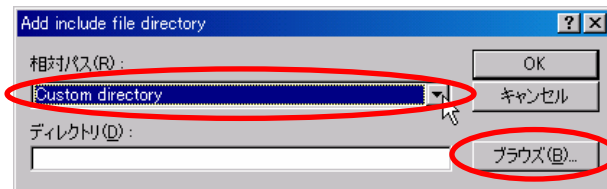
```
PADR = 0x55;
PADR = 0xaa;
PADR = 0x00;
```

と、プログラムしたとおりの動きになります。

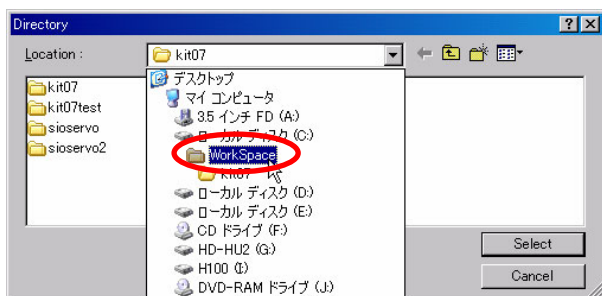




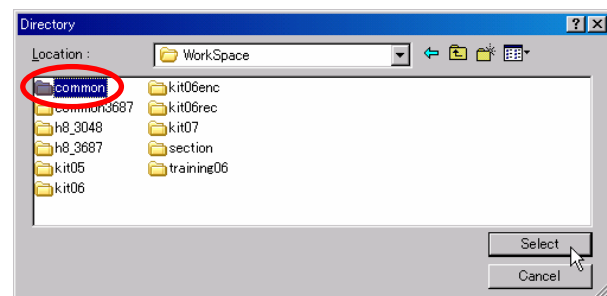
7. 「コンパイラ」を選択します。「カテゴリ: ソース」、**追加**をクリックします。



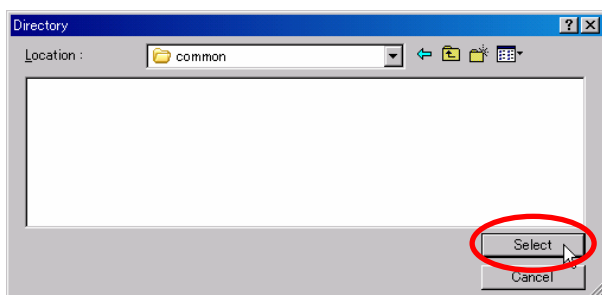
8. 「相対パス: **Custom directory**」にします。続けて、**ブラウズ**をクリックします。



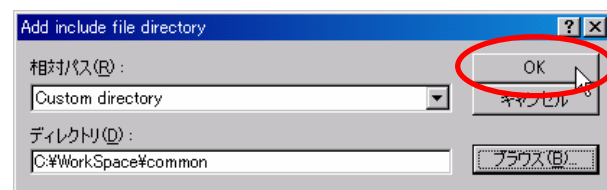
9. 「Cドライブの WorkSpace」フォルダを選択します。



10. 「common」フォルダをダブルクリックします。

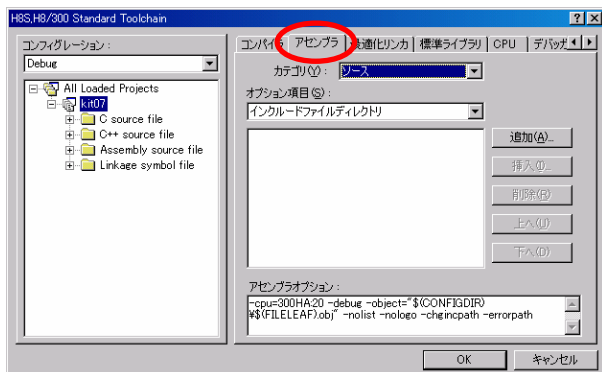


11. **Select**をクリックします。



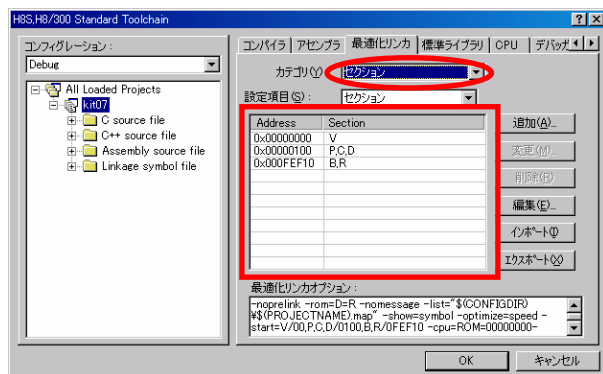
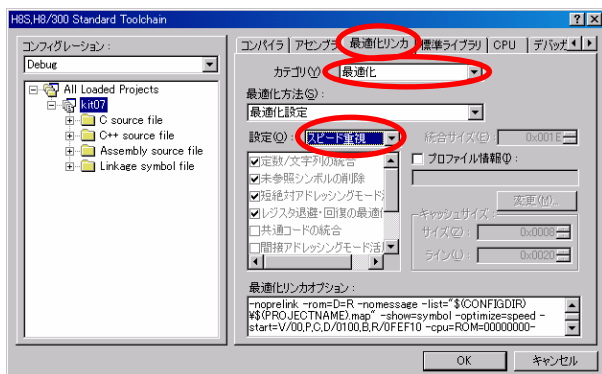
12. **OK**をクリックします。

### 11.3 アセンブラの設定



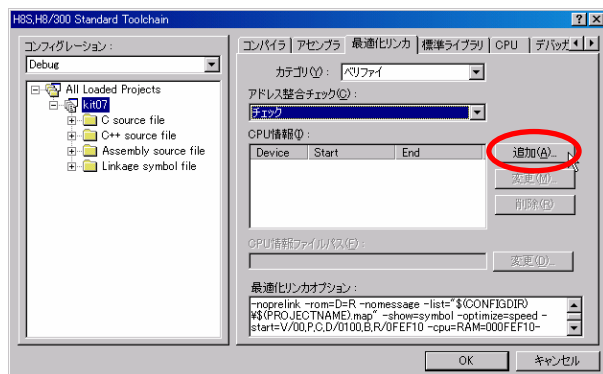
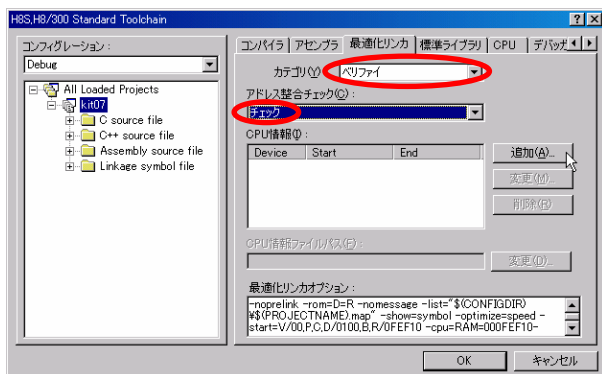
1. アセンブラで設定する項目はありません。

### 11.4 最適化リンカの設定



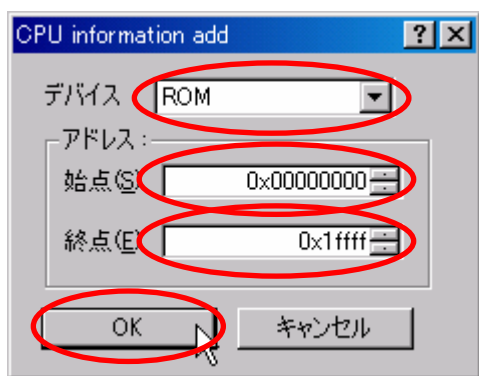
1. 「最適化リンカ」を選択します。「カテゴリ:最適化」、「設定:スピード重視」にします。

2. 「カテゴリ:セクション」にします。□欄はプログラムに合わせてセクションを設定します。プロジェクト「kit07」は画面のようになります。



3. 「カテゴリ:ベリファイ」、「アドレス整合チェック:チェック」にします。これは、CPU の無効なアドレスにプログラムを書き込んだときに、警告するための設定です。

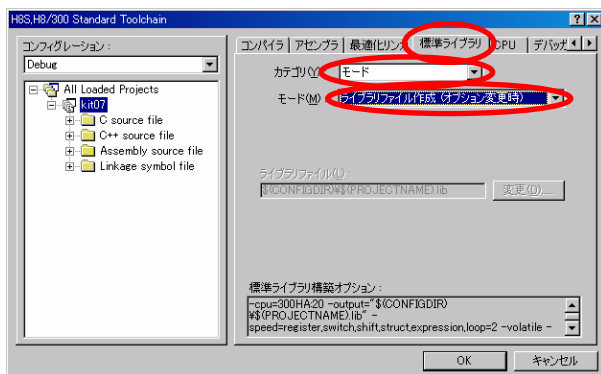
4. 「追加」をクリックします。次に ROM の容量と RAM の容量を入力しますが、CPU の種類によってアドレスが変わります。



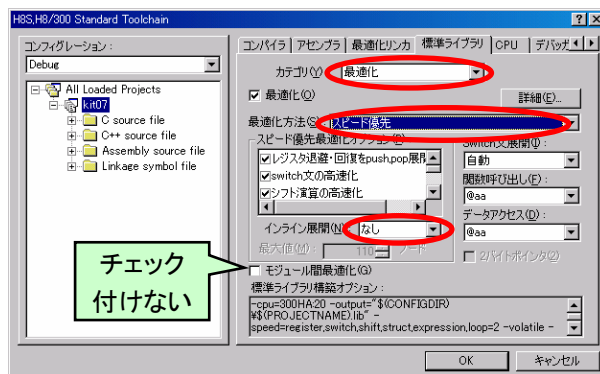
5. 「デバイス:ROM」を選択、「始点:0x0000」、「終点:0x1ffff」を入力して「OK」をクリックします。

6. 再度「追加」をクリックします。「デバイス:RAM」を選択、「始点:0xfef10」、「終点:0xffb0f」を入力して「OK」をクリックします。**RAM は 4KB ありますが、1KB はスタックなどで使用しますので、**  
 $0xfef10 + 3KB = 0xffb0f$  とします。

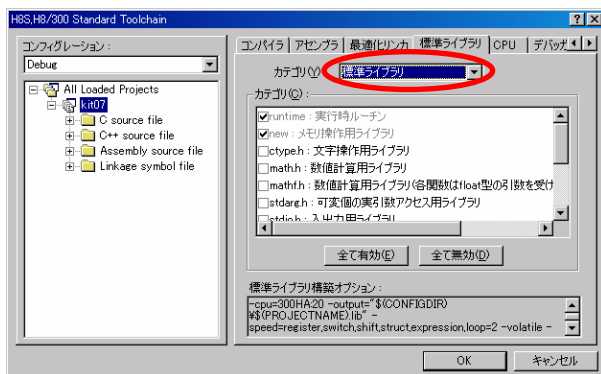
## 11.5 標準ライブラリの設定



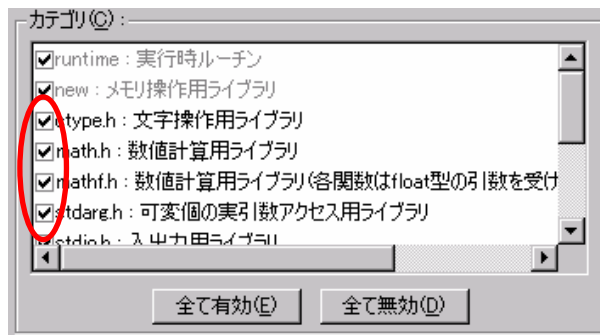
1. 「標準ライブラリ」を選択します。「カテゴリ:モード」、「モード:ライブラリファイル作成(オプション変更時)」にします。



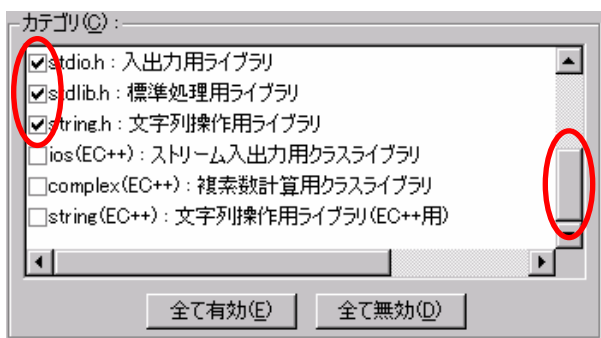
2. 「カテゴリ:最適化」、「最適化方法:スピード優先」、「インライン展開:なし」を選択します。「モジュール間最適化」のチェックは付けません。



3. 「カテゴリ:標準ライブラリ」を選択します。

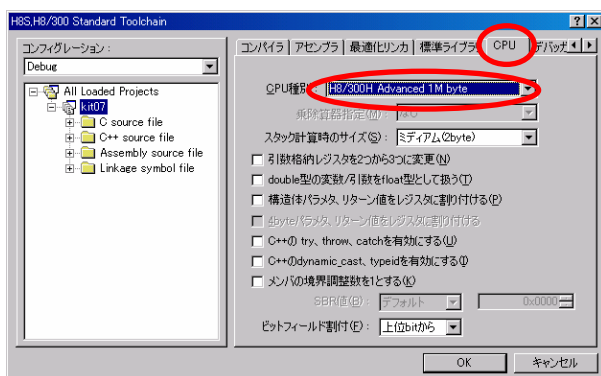


4. カテゴリ欄のファイルにチェックを付けます。

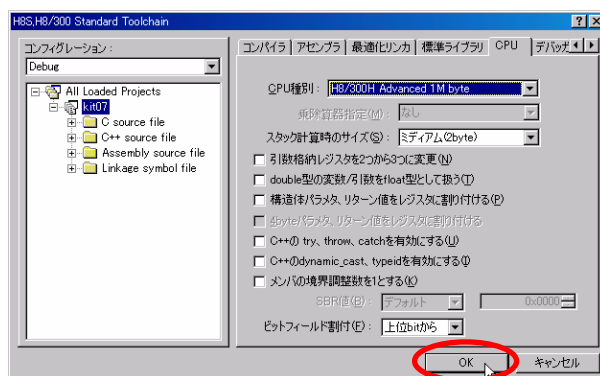


5. スクロールバーを下ろして、更にチェックを付けます。いちばん下の3ファイル(EC++と書かれたファイル)にはチェックを付けません。

## 11.6 CPUの設定



1. 「CPU」を選択します。  
「CPU 種別: **H8/300H Advanced 1M byte**」にします。

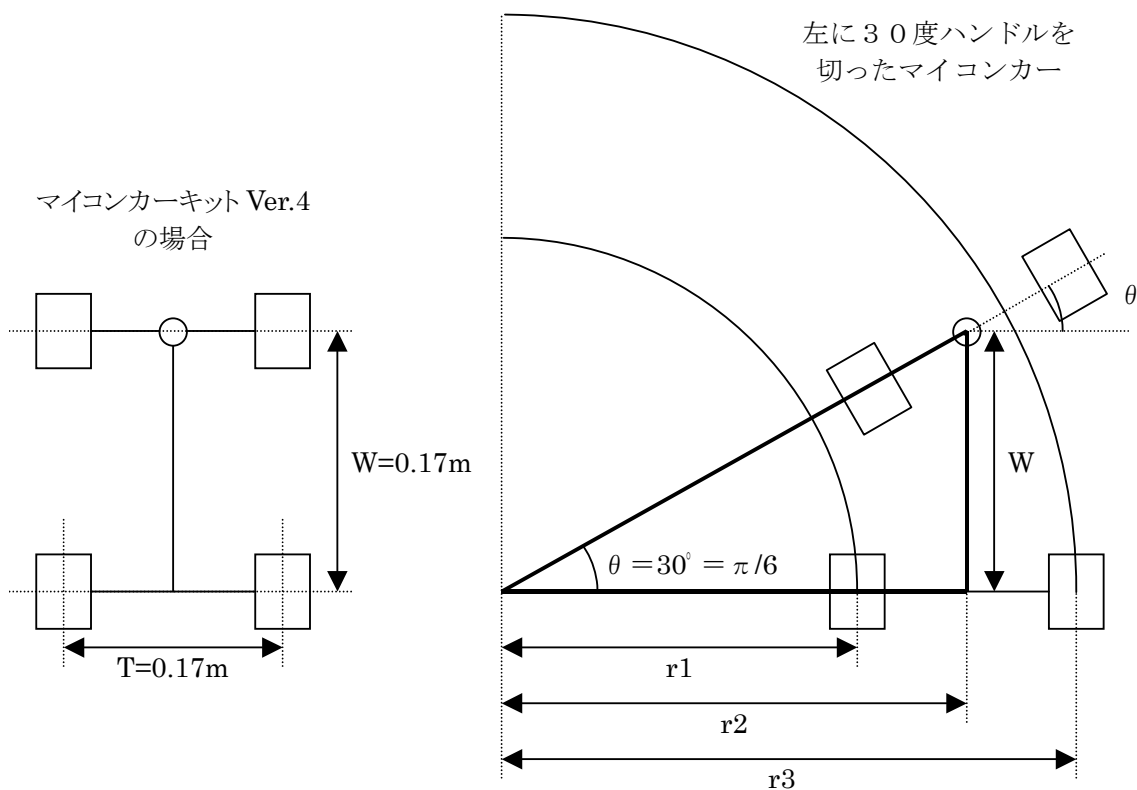


2. 「OK」をクリックして、ツールチェーンの設定を完了します。

## 12. モータの左右回転差の計算方法

### 12.1 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[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[m]です。

図のように、底辺  $r_2$ 、高さ  $W$ 、角度  $\theta$  の三角形の関係は次のようです。

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

角度  $\theta$ 、 $W$  が分かっていますので、 $r_2$  が分かります。

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

内輪の半径は、

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

外輪の半径は、

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

よって、外輪を 100 とすると内輪の回転数は、

$$r_1 / r_3 \times 100 = 0.209 / 0.379 \times 100 = 55$$

となります。

左に  $30^\circ$  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 );
```

### 12.2 内輪を計算するエクセルシートの作成

エクセルで、表を作って 0~45 度くらいまでの角度と左右タイヤの回転比の表を作っておくと便利です。

	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			
3							
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
21		16	0.279	0.593	0.508	0.678	75

セル	内容	値、式の例
C1	ホイールベースを入力	キット Ver.4 なら 0.17
C2	トレッドを入力	キット Ver.4 なら 0.17
B 列	角度を入力 0 から 45 まで	直接入力
C 列	角度° を rad に変換	C6 セル=B6*3.14/180
D 列	図の r2 の計算	D6 セル=\$C\$1/TAN(C6)
E 列	図の r1 の計算	E6 セル=D6-\$C\$2/2
F 列	図の r3 の計算	F6 セル=D6+\$C\$2/2
G 列	比率の計算	G6 セル=E6/F6*100

W を 0.17[m]、T を 0.17[m]としたときの、表を次ページに示します。  
 例えば、通常走行時、センサ状態が「0x06」のとき、次のようなプログラムでした。

```
176 :          case 0x06:
177 :                /* 少し左寄り→右へ小曲げ */
178 :                handle( 10 );
179 :                speed( 80 , ?? );          ←??が分からない
180 :                break;
```

右へ小曲げにするため、ハンドルを右へ 10 度、左タイヤを 80%にしようと考えます。内輪側の右タイヤの PWM 値が分かりません。

表より、10 度のときは内輪側の左タイヤが 84%であることが分かります。ただし、これは外輪が 100%のときの値です。今回は 80%にしますので、

$$\text{右タイヤ} = \text{左タイヤの PWM} \times \text{比率} / 100 = 80 \times 84 / 100 = 67$$

よって、右タイヤの PWM 値を **67** に設定します。kit07.c では、内輪と外輪の回転差をこのようにして計算しています。

▼内輪側の関係

度	rad	r2	r1	r3	r1/r3*100
0	0				100
1	0.017	9.744	9.659	9.829	98
2	0.035	4.871	4.786	4.956	97
3	0.052	3.245	3.160	3.330	95
4	0.070	2.432	2.347	2.517	93
5	0.087	1.944	1.859	2.029	92
6	0.105	1.618	1.533	1.703	90
7	0.122	1.385	1.300	1.470	88
8	0.140	1.210	1.125	1.295	87
9	0.157	1.074	0.989	1.159	85
10	0.174	0.965	0.880	1.050	84
11	0.192	0.875	0.790	0.960	82
12	0.209	0.800	0.715	0.885	81
13	0.227	0.737	0.652	0.822	79
14	0.244	0.682	0.597	0.767	78
15	0.262	0.635	0.550	0.720	76
16	0.279	0.593	0.508	0.678	75
17	0.297	0.556	0.471	0.641	73
18	0.314	0.523	0.438	0.608	72
19	0.331	0.494	0.409	0.579	71
20	0.349	0.467	0.382	0.552	69
21	0.366	0.443	0.358	0.528	68
22	0.384	0.421	0.336	0.506	66
23	0.401	0.401	0.316	0.486	65
24	0.419	0.382	0.297	0.467	64
25	0.436	0.365	0.280	0.450	62
26	0.454	0.349	0.264	0.434	61
27	0.471	0.334	0.249	0.419	59
28	0.488	0.320	0.235	0.405	58
29	0.506	0.307	0.222	0.392	57
30	0.523	0.295	0.210	0.380	55
31	0.541	0.283	0.198	0.368	54
32	0.558	0.272	0.187	0.357	52
33	0.576	0.262	0.177	0.347	51
34	0.593	0.252	0.167	0.337	50
35	0.611	0.243	0.158	0.328	48
36	0.628	0.234	0.149	0.319	47
37	0.645	0.226	0.141	0.311	45
38	0.663	0.218	0.133	0.303	44
39	0.680	0.210	0.125	0.295	42
40	0.698	0.203	0.118	0.288	41
41	0.715	0.196	0.111	0.281	39
42	0.733	0.189	0.104	0.274	38
43	0.750	0.182	0.097	0.267	36
44	0.768	0.176	0.091	0.261	35
45	0.785	0.170	0.085	0.255	33

※Wを0.17[m]、Tを0.17[m]としたときの場合

WとTの値を自分のマイコンカーの長さに変えると、左右のタイヤの回転比率が分かります。

### 12.3 サンプルエクセルシートを使った内輪の計算

サンプルで、「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。

	A	B	C	D	E	F	G	H	I	J	K	L
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		プログラム例 <b>handle( 15 )</b> <b>speed(50,38)</b>			
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					

このセルの、

- L1セル→ハンドル角度の入力
- L2セル→外輪のスピードを入力

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。

これは便利です。実は、kit07.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。

ただし、ホイールベースとトレッドはきちんと合わせておいてください。



## 13. サーボセンタと最大切れ角の調整

### 13.1 概要

kit07.mot を書き込んでマイコンカーの電源を入れると、ハンドルが0度になっていないと思います。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。「kit07.c」の36行

```

27                                     /* φ/8で使用する場合、 */
28                                     /* φ/8 = 325.5[ns] */
29                                     /* ∴TIMER_CYCLE = */
30                                     /*     1[ms] / 325.5[ns] */
31                                     /*     = 3072 */
32 #define PWM_CYCLE 49151 /* PWMのサイクル 16ms */
33                                     /* ∴PWM_CYCLE = */
34                                     /*     16[ms] / 325.5[ns] */
35                                     /*     = 49152 */
36 #define SERVO_CENTER 5000 /* サーボのセンタ値 */
37 #define HANDLE_STEP 26 /* 1°分の値 */
38
39 /* マスク値設定 ×：マスクあり(無効) ○：マスク無し(有効) */
40 #define MASK2_2 0x66 /* ×○○××○○× */
41 #define MASK2_0 0x60 /* ×○○×××××× */
42 #define MASK0_2 0x06 /* ×××××○○× */
43 #define MASK3_3 0xe7 /* ○○○××○○○ */
44 #define MASK0_3 0x07 /* ×××××○○○ */
45 #define MASK3_0 0xe0 /* ○○○×××××× */
46 #define MASK4_0 0xf0 /* ○○○○××××× */
    
```

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26、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
- CPU ボードに書き込む
- 0度か確かめる
- 0度でなければやり直し

という作業を通常は、最低5回は繰り返さなければき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がります。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
- CPU ボードに書き込む

という作業のみでOKです。先ほどより、簡単になりました。今回は、パソコンのキーボードを調整用として使い、

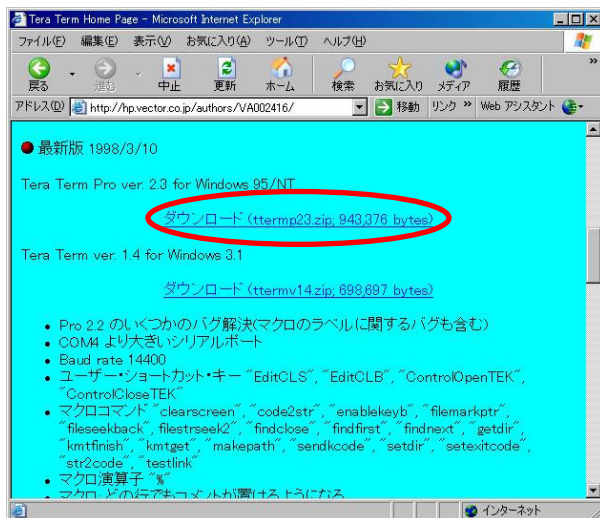
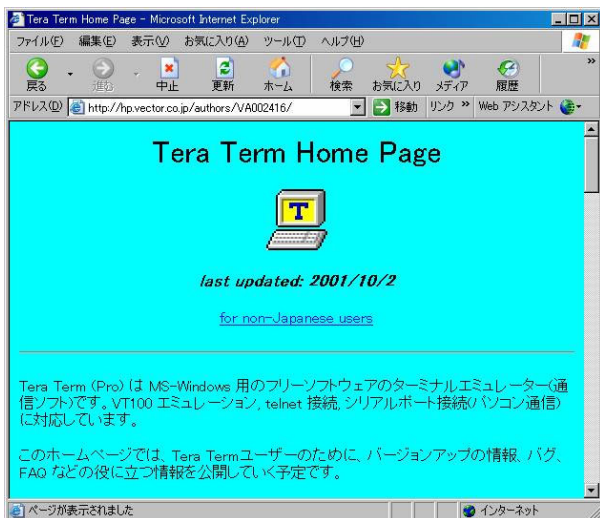
- サーボセンタの値を簡単に調整しましょう
- 最大切れ角もいっしょに見つけましょう

というのが内容です。

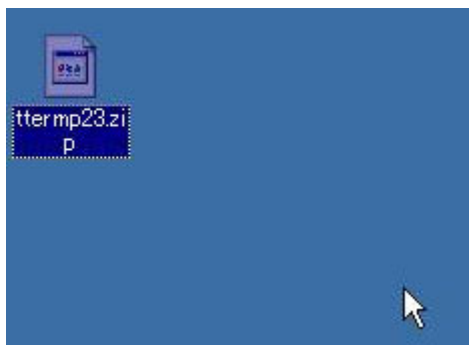
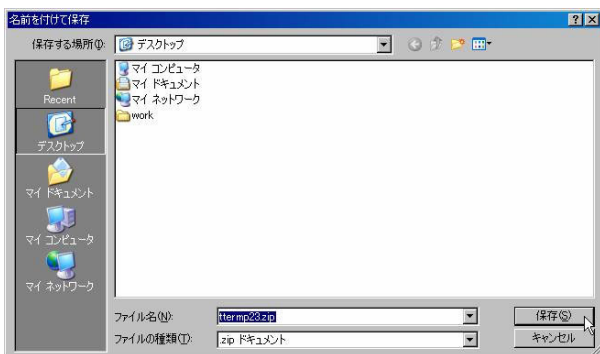
## 13.2 通信ソフトをインストールする

ハイパーターミナルというソフトは、Windows 標準で入っています。しかし、古いWindowsでは入っていないことがある、なぜか調子が悪いなど不具合が発生しやすいソフトでもあります。そこで、フリーソフトで通信のできる「Tera Term Pro」というソフトを使用してみます。

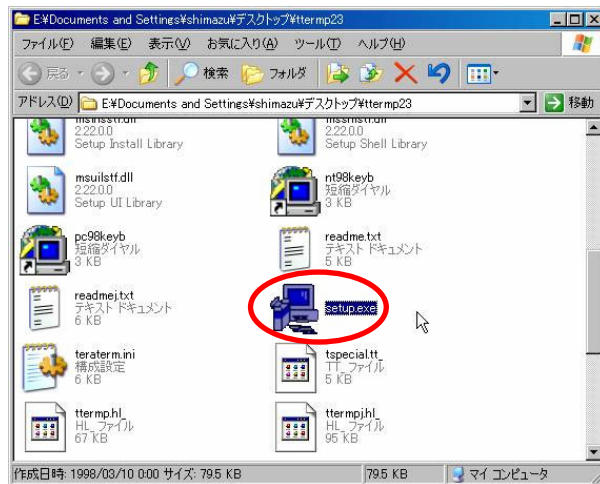
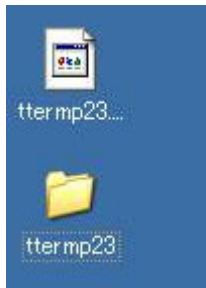
### 13.2.1 Tera Term Proのインストール



1. 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://hp.vector.co.jp/authors/VA002416/>  
 を開きます。  
 または、講習会 CD がある場合は、  
 CD ドライブ → 401 関連ソフト → ttermp23 →  
 setup.exe  
 を実行してください。その場合、7 へ進んでください。
2. 下の方に「ダウンロード (ttermp23.zip; 943,376 bytes)」とあるので、クリックして保存します。



3. 保存は何処でも良いですが、ここではデスクトップに保存します。
4. 保存されました。

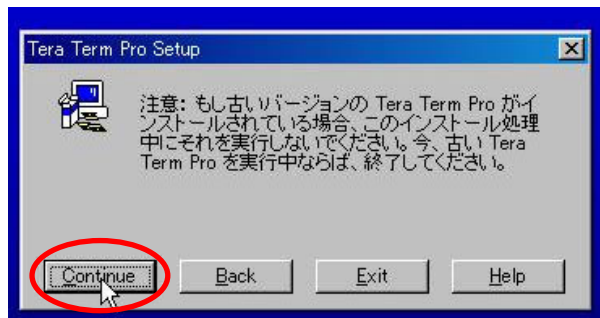


5. ttermp23.zip は ZIP 形式で圧縮された形式なので、解凍します。解凍ソフトは、フリーソフトでたくさんありますので、インターネットなどで探してください。図は、ttermp23 というフォルダに解凍したところです。

6. 解凍後のファイルです。setup.exe を実行します。



7. 言語の選択です。日本になっていますのでそのまま **Continue**(続ける)をクリックします。



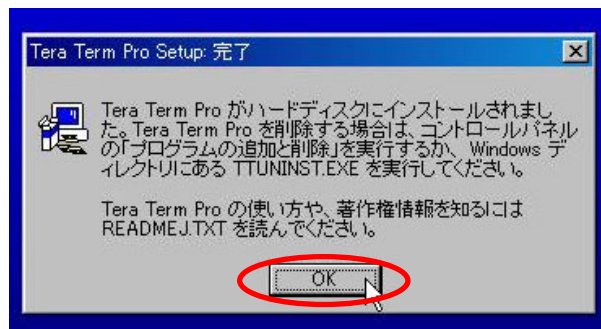
8. 注意が出ます。**Continue**をクリックします。



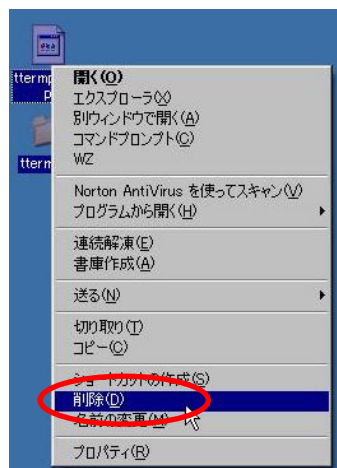
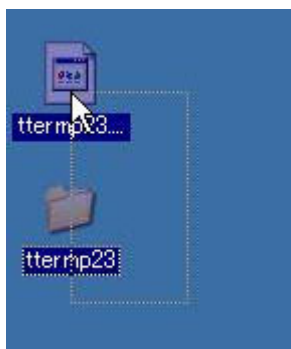
9. キーボードの選択です。そのまま **Continue** をクリックします。



10. インストール先を確認して、問題なければ **Continue** をクリックします。インストールが開始されます。

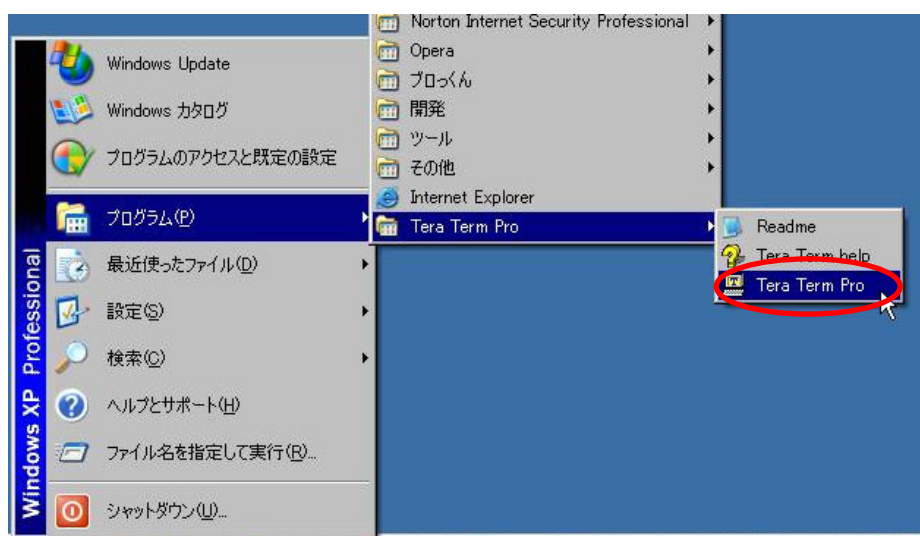


11. メニューが立ち上がります。Xをクリックして閉じます。
12. OKをクリックして、インストールを完了します。

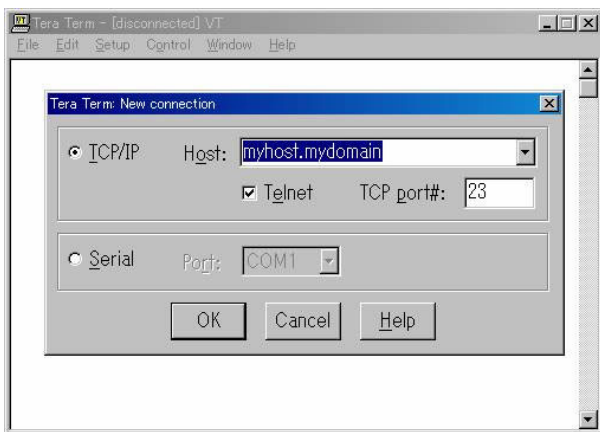


13. インターネットからファイルをダウンロードした場合、削除します。まず、tterm23.zip と tterm23 フォルダを選択します。CD からインストールした場合は不要です。
14. どちらかのファイルの上で右クリックして「削除」を選択、ファイルを削除します。

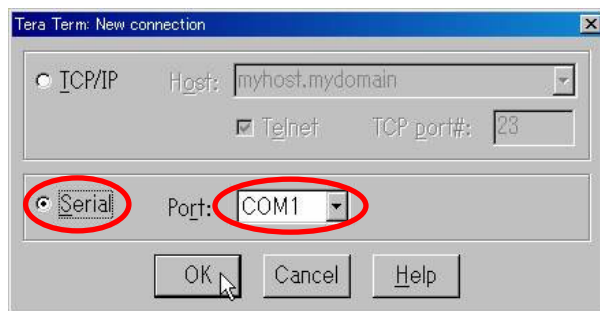
### 13.2.2 Tera Term Proの使い方



1. 「スタート→すべてのプログラム、またはプログラム→Tera Term Pro→Tera Term Pro」で Tera Term Pro が立ち上がります。

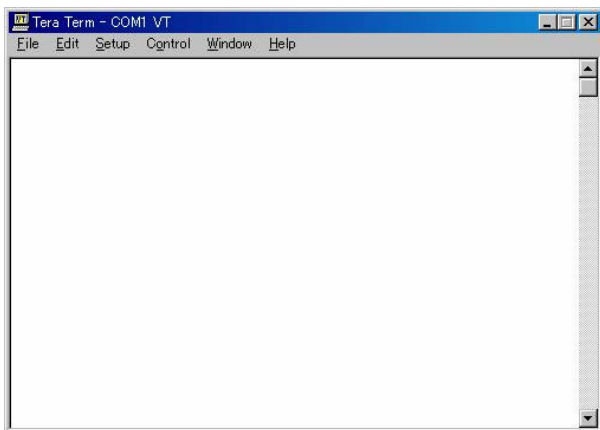


2. 最初にどこと接続するか確認する画面が出てきます。

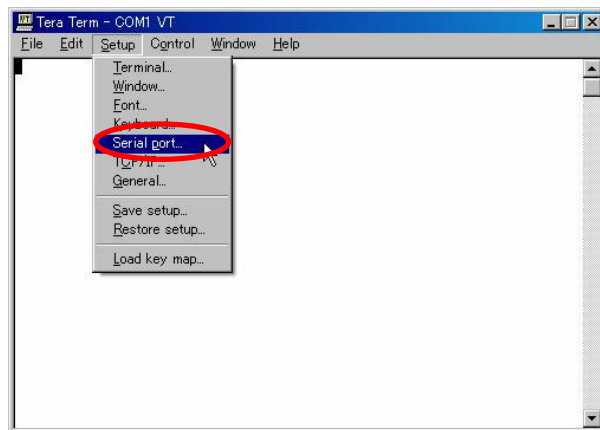


3. 「Serial」を選んで、ポート番号を選びます。選択後、**OK**をクリックします。

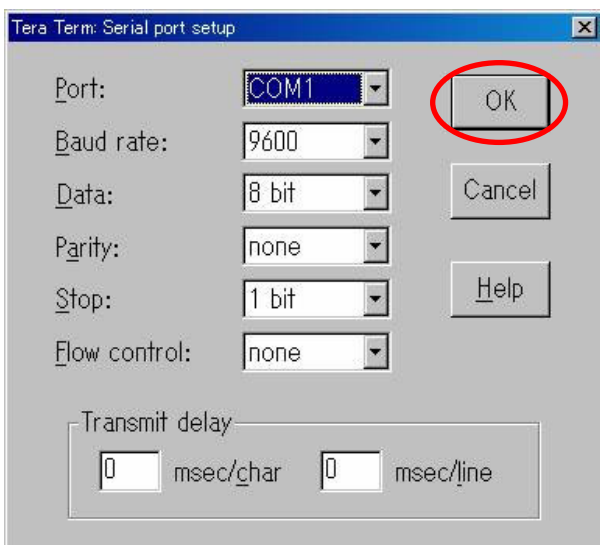
**※標準では COM4 までしか選択できませんが、増やすことができます。後述します。**



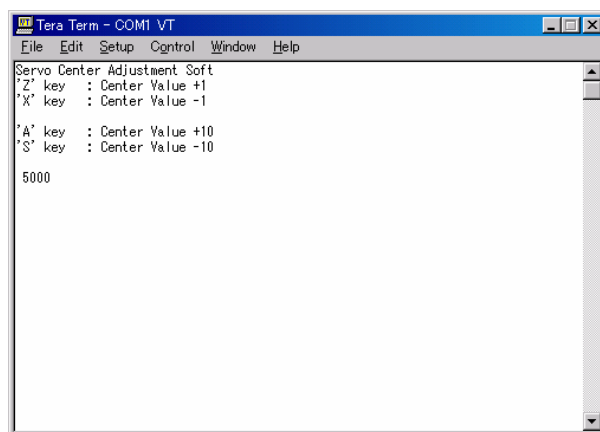
4. 立ち上がりました。詳細設定をします。



5. 「Setup→Serial port」を選択します。

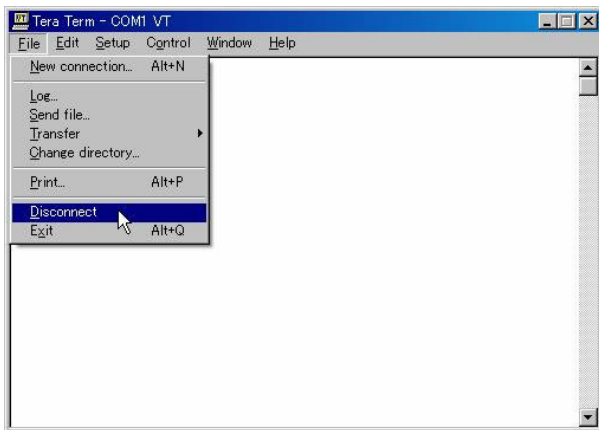


6. 通信設定を確認します。画面のように設定して、**OK**をクリックします。「Port」は、それぞれの通信ポートの番号に合わせてください。

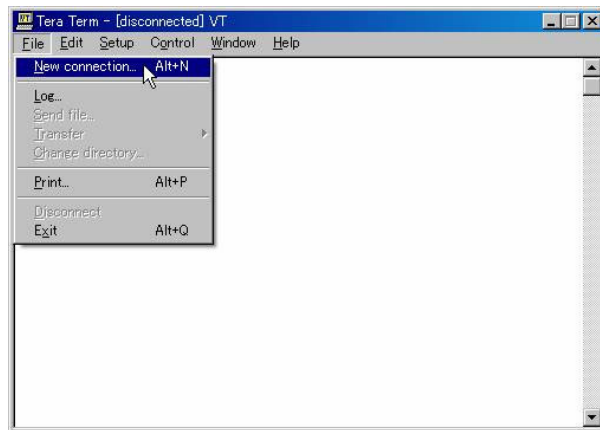


※プロジェクト「sioservo」の例です。

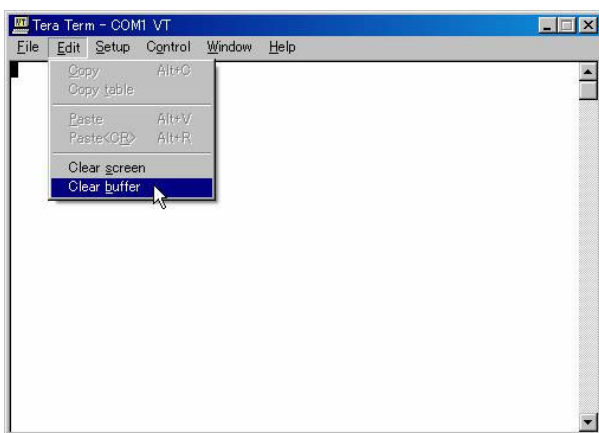
7. 通信するプログラムが入っているマイコンカーの電源を入れたら、マイコンカーからメッセージが送られてきます。表示されれば接続成功です。



8. 一時的な切断は、「File→Disconnect」で切断できます。



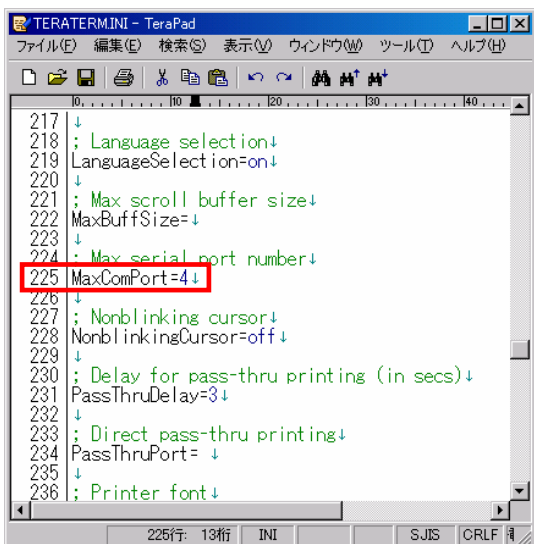
9. 再接続する場合は、「File→New connection」で接続できます。



10. 画面をクリアしたい場合は、「Edit→Clear buffer」です。

### 13.2.3 COM番号の増やし方

Tera Term Pro をインストールしたフォルダ、標準では、  
 Cドライブ→Program Files→TTERMPRO  
 です。このフォルダにある「TERATERM.INI」というファイルを開きます。読み取り専用になっている事があるので、  
**右クリック→プロパティで確認します。読み取り専用ならチェックを外します。**



225 行に

**MaxComPort=4**

とあります。この「4」という数字が COM 番号の最大値です。

この数値を「16」に書き換えておきましょう。COM16 まで開くことができます(TeraTermPro は 16 が最大です)。

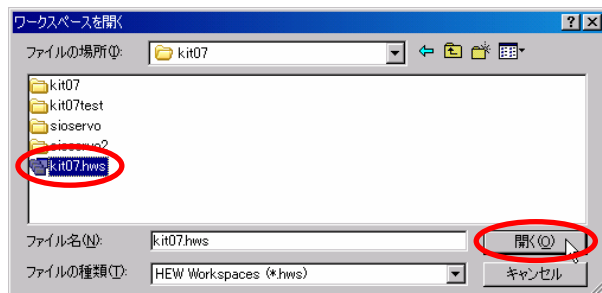
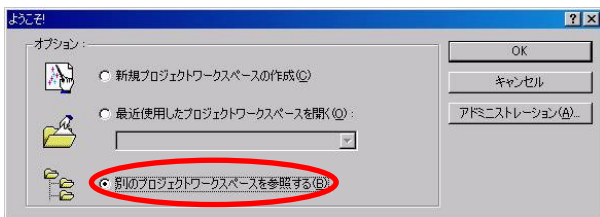
上書き保存で保存して、完了です。

### 13.3 サーボのセンタを調整する

ワークスペース「kit07」を開きます。

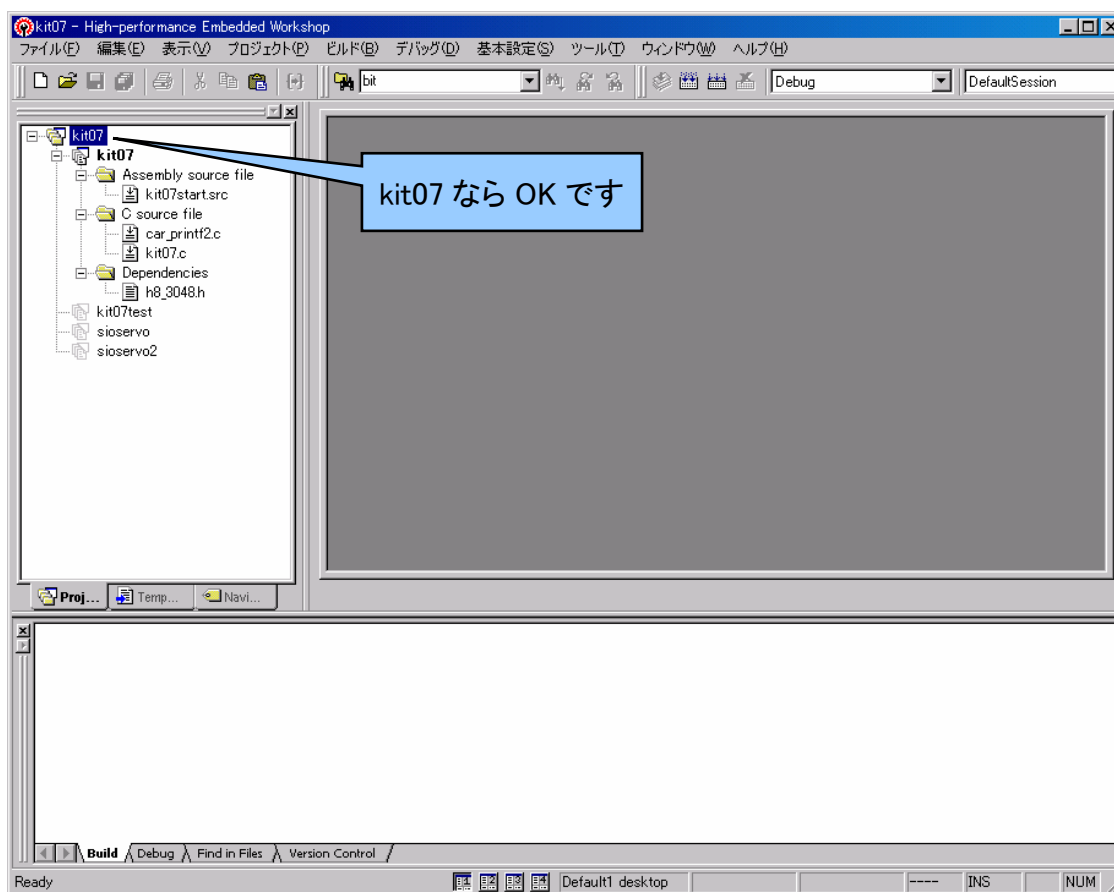


1.ルネサス統合開発環境を実行します。

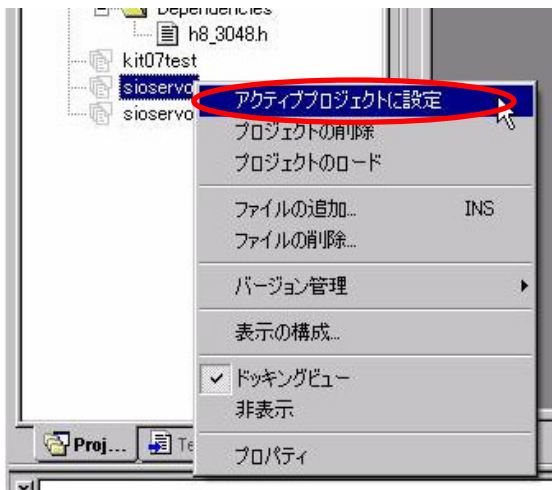


2.「別のプロジェクトワークスペースを参照する」を選択します。

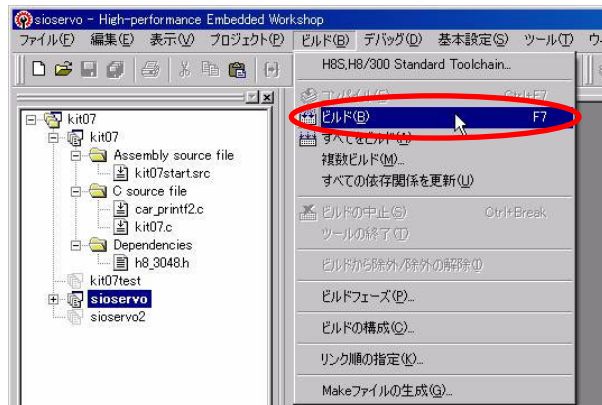
3.Cドライブ→Workspace→kit07 の「kit07.hws」を選択します。



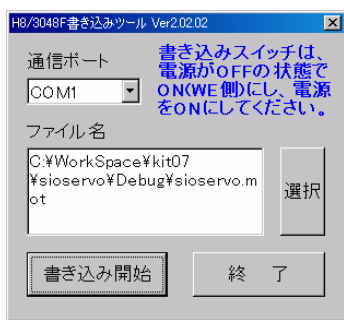
4.kit07 というワークスペースが開かれます。



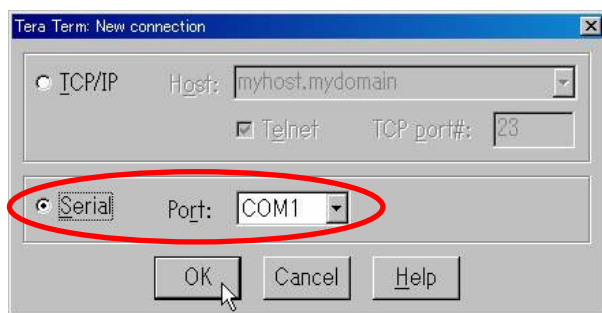
5. プロジェクト「sioservo」をアクティブプロジェクトに設定します。



6. 「ビルド→ビルド」でビルドします。MOT ファイルがで  
き上がります。



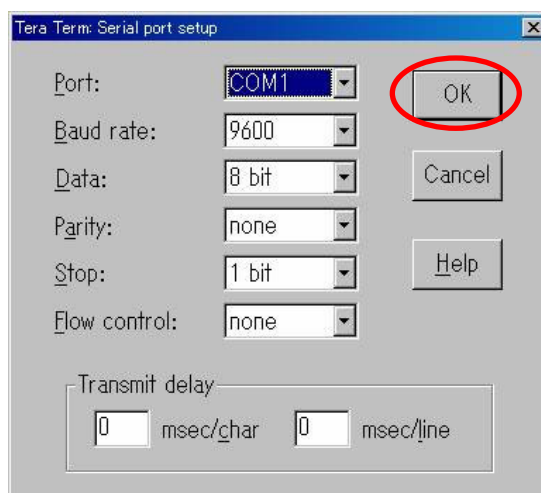
7. ルネサス統合開発環境の「ツール→CpuWrite」で書き込みソフトを立ち上げ、プログラムを書き込みます。転送終了後、CPU ボードの電源を OFF にして書き込みスイッチを内側にしておきます。ケーブルは繋いだままです。



8. 「スタート→プログラム→Tera Term Pro→Tera Term Pro」で Tera Term Pro を立ち上げます。Serial を選択して、ポートを書き込みポートの番号に合わせます。

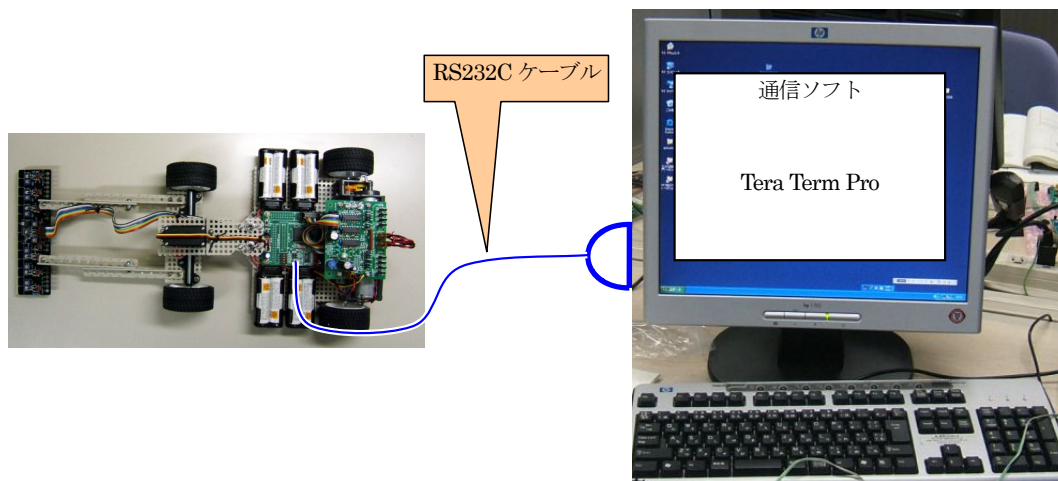


9. 「Setup→Serial port」を選択します。



10. 設定は、上記のようにします。OK をクリックして TeraTermPro の準備が整いました。

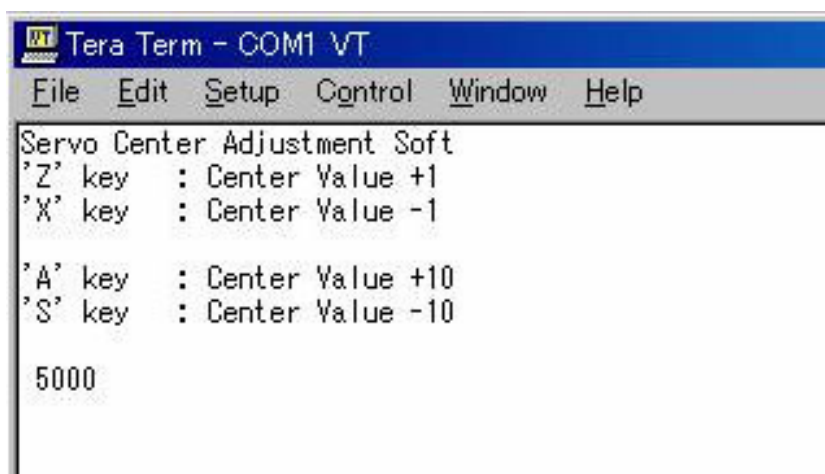




11.接続、設定を確かめます。

- マイコンカーとパソコンを RS232C ケーブルで接続しているか
- TeraTermPro を立ち上げて、ポートを設定しているか

OK なら次に進みます。

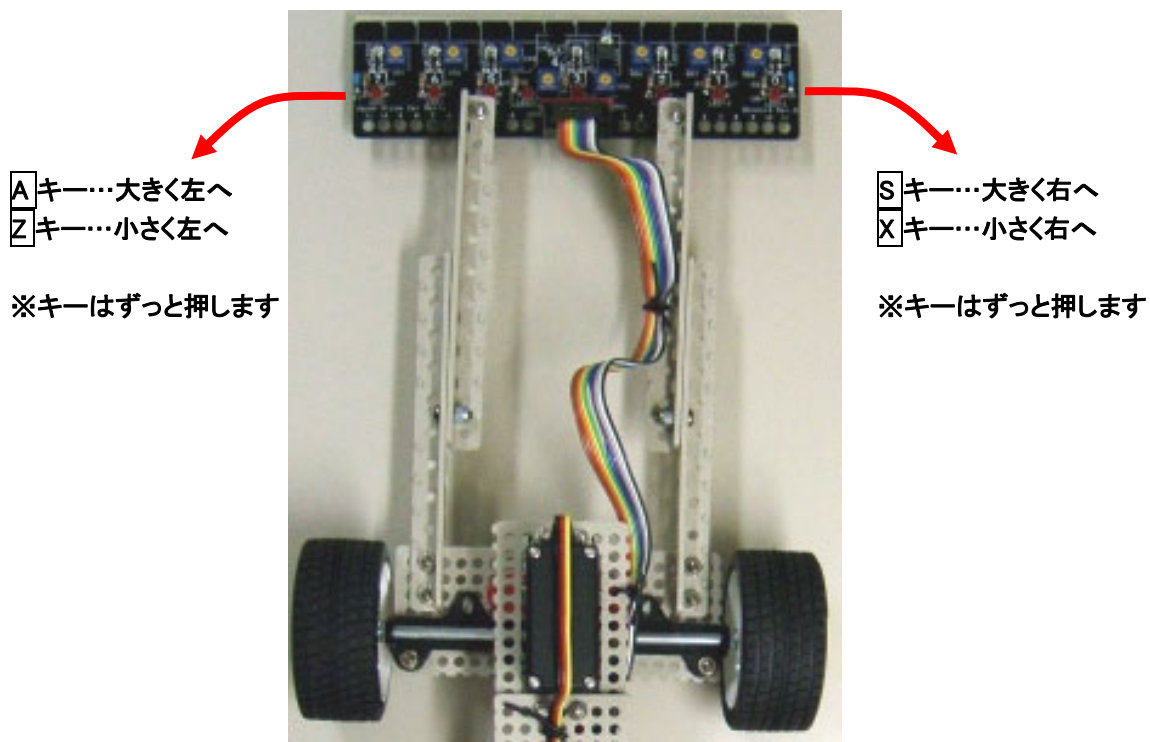


12. マイコンカーの電源を入れるとメッセージが表示されます。表示されない場合は、ケーブルの接続やマイコンカーの電池、書き込みスイッチを戻したか、通信ポートの番号、書き込んだプログラムが本当にプロジェクト「sioservo」のsioservo.mot を書き込んだかなど確かめてください。

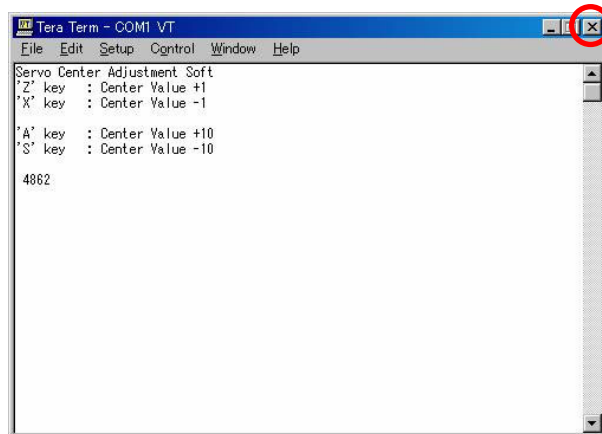
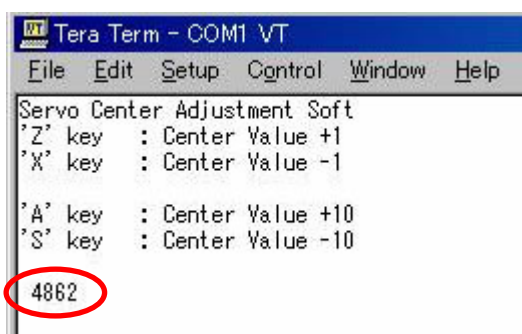
### ポイント

TeraTermPro を立ち上げてから、**一番最後にマイコンカーの電源を入れます。**

電源を入れた瞬間にマイコンカーからメッセージを出力しますので、その後に TeraTermPro を立ち上げても何も表示されません。表示内容は、マイコンカーから送られてきます。



13. **A**、**S**、**Z**、**X** キーをそれぞれ押し続けるとサーボが動きます。キーを使ってサーボがまっすぐ向く角度に調整してください。

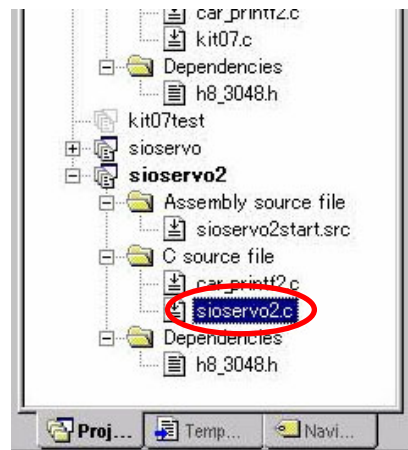
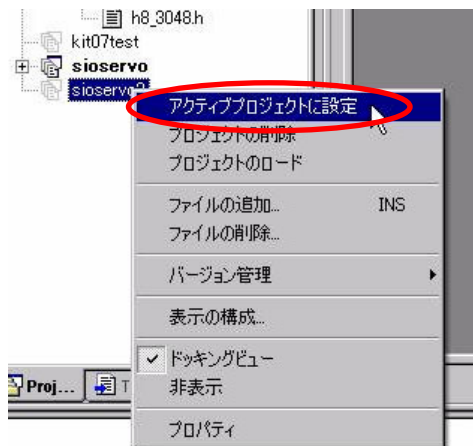


14. まっすぐに調整できたら、Tera Term Pro の数字を見ます。今回は「4862」とでした。これがこのマイコンカーのサーボセンタ (SERVO\_CENTER) の値です。メモしておきます。

15. 次は、プロジェクト「sioservo2」に進みます。**X** をクリックして TeraTermPro を終了しておきます。マイコンカーの電源も切ります。

### 13.4 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」をアクティブプロジェクトに設定します。
2. 「sioservo2.c」をダブルクリックしてエディタウィンドウを開きます。

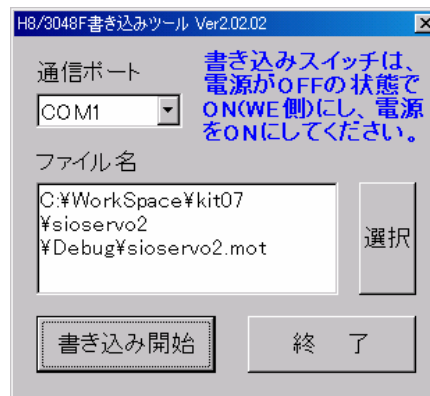
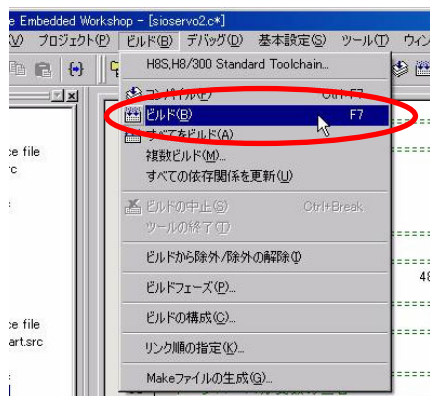
```

24 #include <stdio.h>
25 #include <machine.h>
26 #include "h8_3048.h"
27
28 /*=====*/
29 /* シンボル定義 */
30 /*=====*/
31 #define SERVO_CENTER 5000 /*
32
33 /*=====*/
34 /* プロトタイプ宣言 */
35 /*=====*/
36 void init( void );
37
38 /*=====*/
39 /* グローバル変数の宣言 */
40 /*=====*/
41 int servo_angle;
    
```

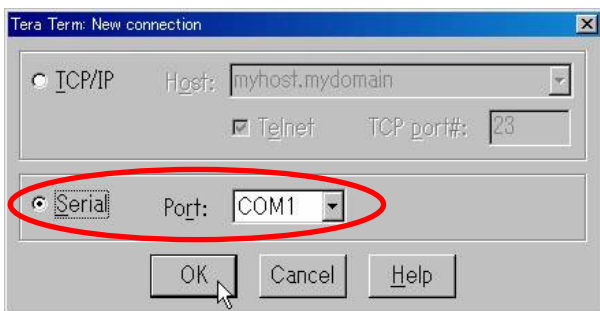
```

24 #include <stdio.h>
25 #include <machine.h>
26 #include "h8_3048.h"
27
28 /*=====*/
29 /* シンボル定義 */
30 /*=====*/
31 #define SERVO_CENTER 4862 /*
32
33 /*=====*/
34 /* プロトタイプ宣言 */
35 /*=====*/
36 void init( void );
37
38 /*=====*/
39 /* グローバル変数の宣言 */
40 /*=====*/
41 int servo_angle;
    
```

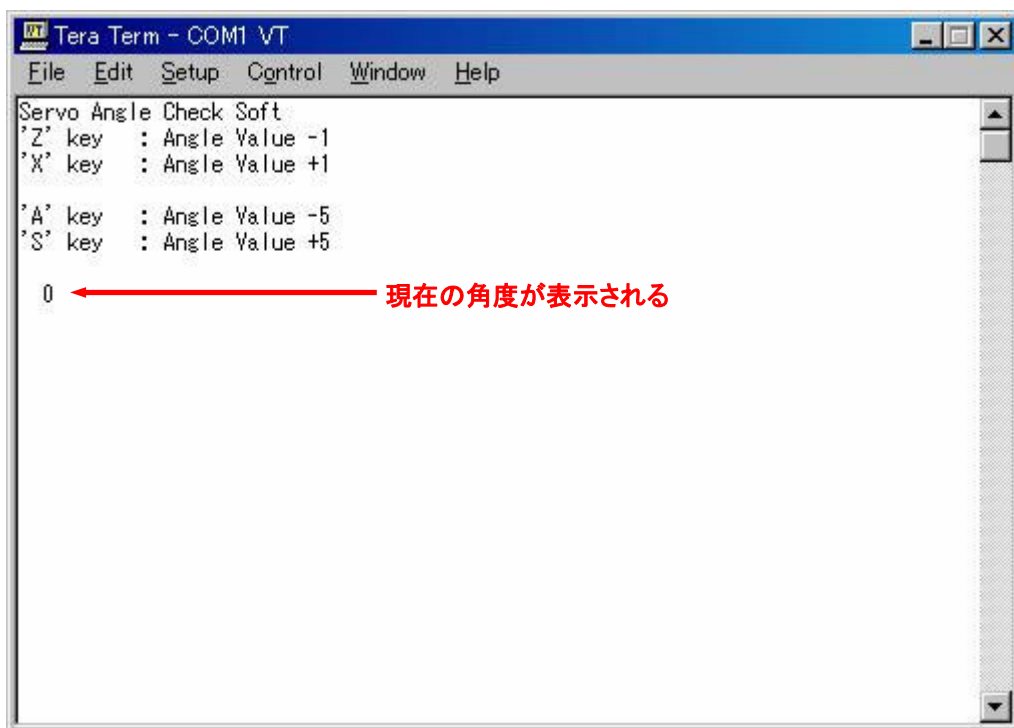
3. 31 行に  
SERVO\_CENTER 5000  
という記述があります。
4. 先ほど見つけたサーボセンタ値に書き換えます。画面は、例として「4862」に書き換えています。



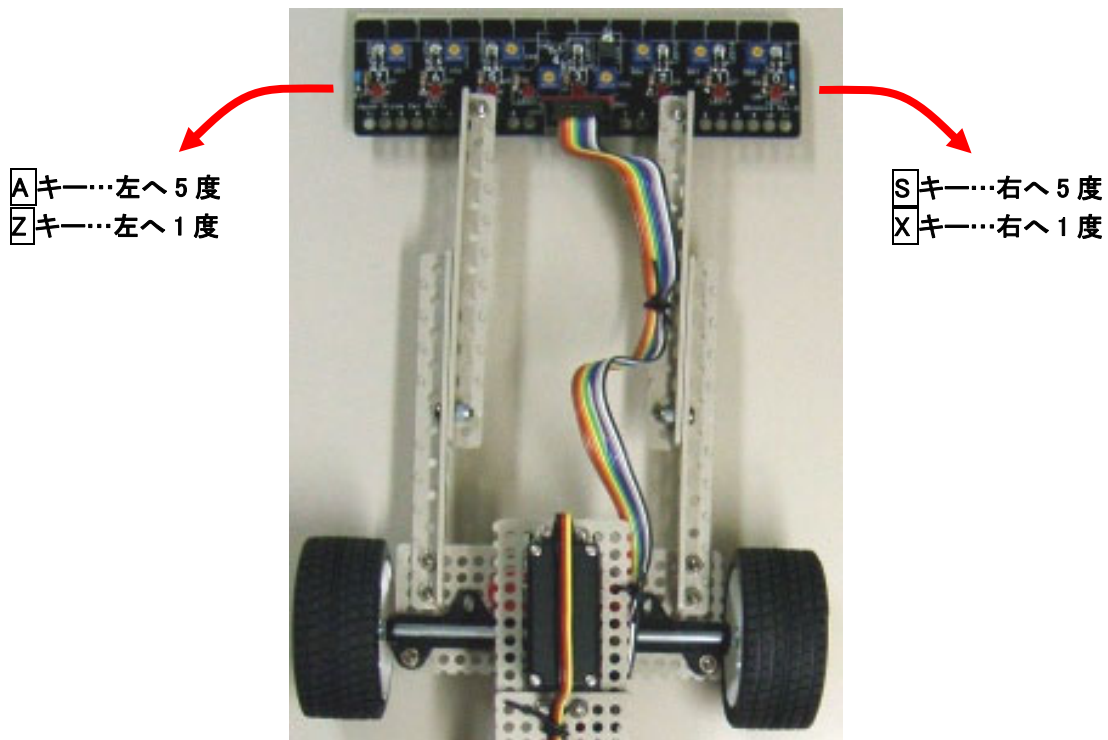
5. 「ビルド→ビルド」で MOT ファイルを作成します。
6. 「ツール→CpuWrite」で書き込みソフトを立ち上げ、プログラムを書き込みます。転送終了後、CPU ボードの電源を OFF にして書き込みスイッチを内側にしておきます。ケーブルは繋いだままです。



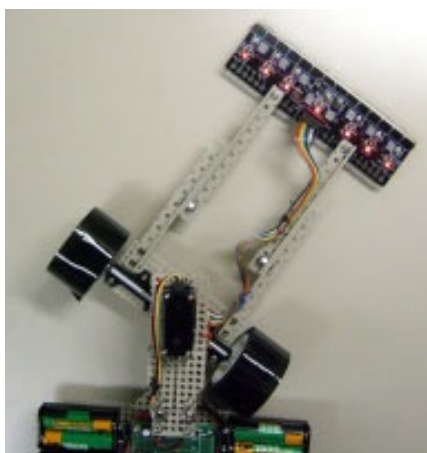
7. 「スタート→プログラム→Tera Term Pro→Tera Term Pro」で Tera Term Pro を立ち上げます。Serial を選択して、ポートを書き込みポートの番号に合わせます。



8. マイコンカーの電源を入れるとメッセージが表示されます。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「sioservo2.mot」かなど確かめてください。



9. **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ることができるかを調べます。左も同様に調べます。



```

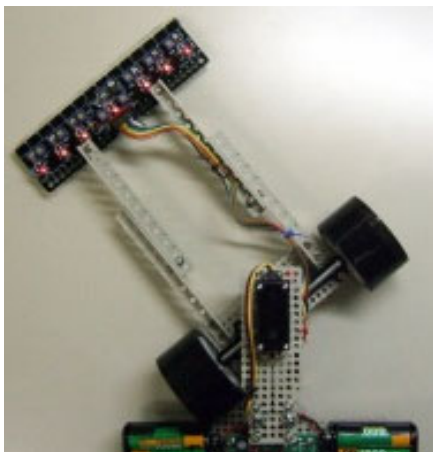
VT Tera Term - COM1 VT
File Edit Setup Control Window Help
Servo Angle Check Soft
'Z' key : Angle Value -1
'X' key : Angle Value +1

'A' key : Angle Value -5
'S' key : Angle Value +5

40
    
```

10. 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 **Z** キーでもう少し小さくしてください。

11. Tera Term Pro 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。**

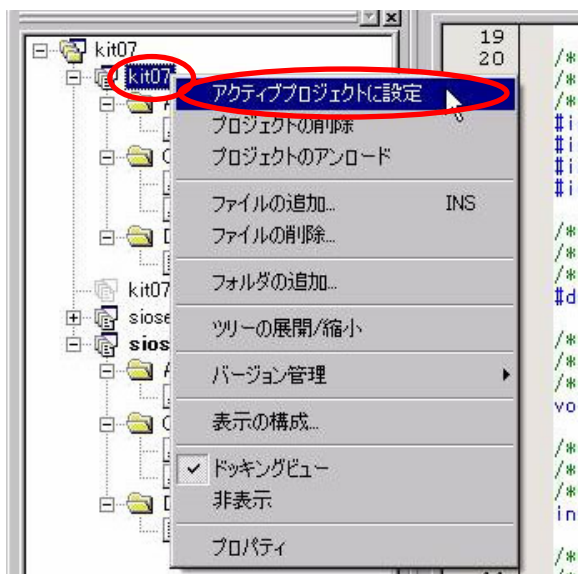


12.今度は **A** キー、**Z** キーで逆の左の限界を見つめます。こちらもタイヤを回して回るか確かめてください。シャーシにぶつかるようなら **X** キーでもう少し小さくしてください。

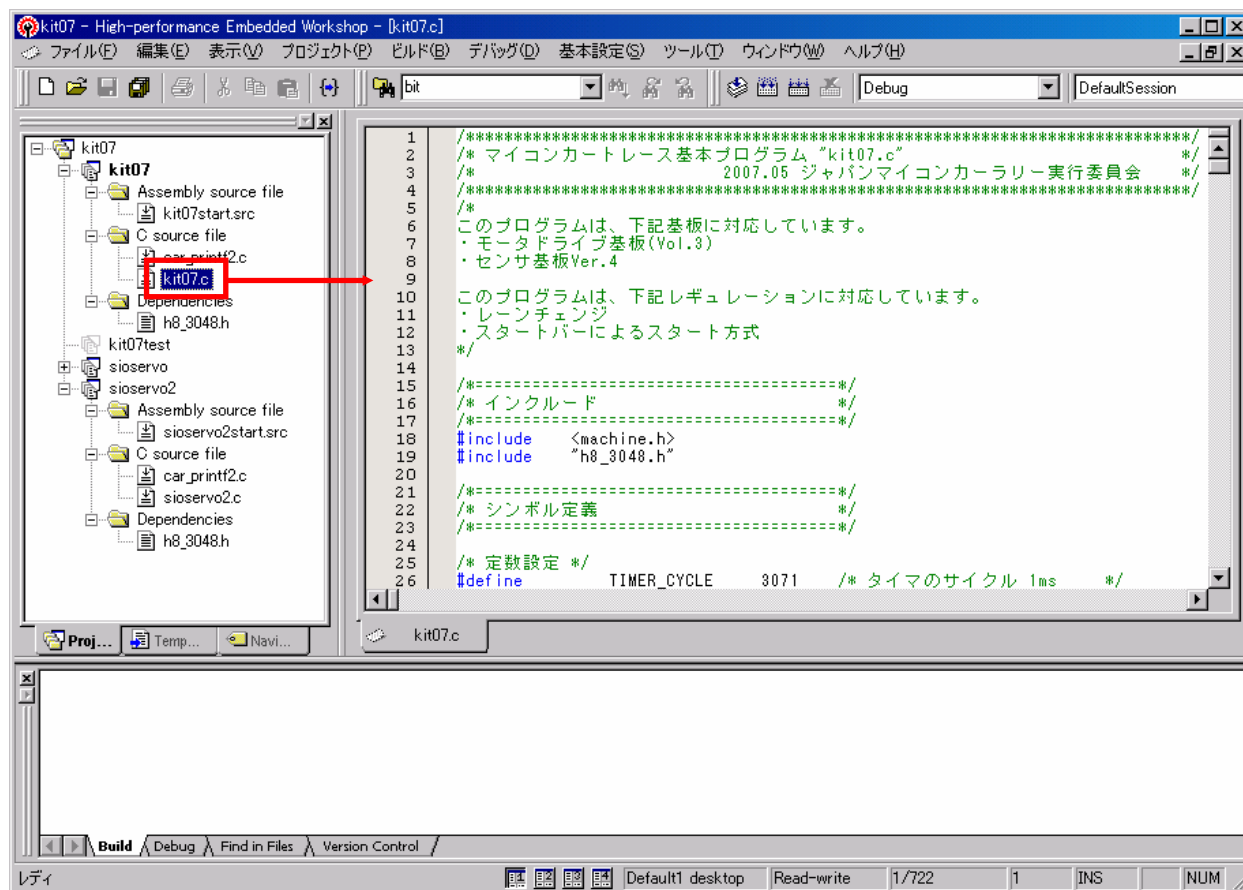
13.Tera Term Pro の数値を見ます。これが現在ハンドルを左へ曲げている角度です。**-41 度曲げていると**言うことが分かりました。

### 13.5 「kit07.c」プログラムを書き換える

プロジェクト「sioservo」、「sioservo2」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である「kit07.c」へ書き込みます。このファイルは、プロジェクト「kit07」内にあります。



1.ワークスペース「kit07」のプロジェクト「kit07」をアクティブプロジェクトに設定します。



2.「kit07.c」をダブルクリックして、エディタウィンドウに表示させます。  
 下記のように変更します。

内容	kit07.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	36 行	5000	4862
左の最大角度	285 行	-38	-41
右の最大角度	294 行	38	40

まず、サーボセンタです。

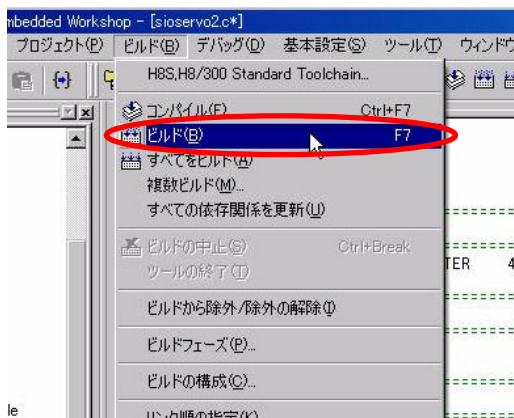
```
36 : #define      SERVO_CENTER    5000    /* サーボのセンタ値      */
↓
36 : #define      SERVO_CENTER    4862  /* サーボのセンタ値      */
```

次に左の最大角度です。この部分は、左クランクを見つけたときにハンドルを切る角度を設定します。

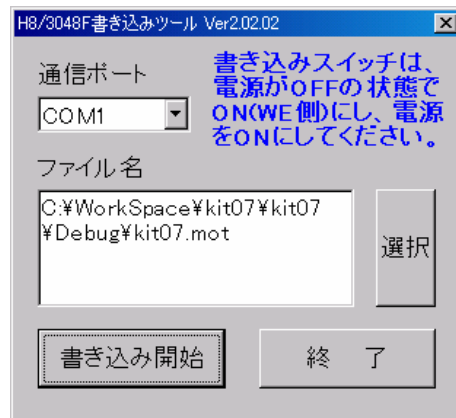
```
285 :          handle( -38 );
↓
285 :          handle( -41 );
```

次に右の最大角度です。この部分は、右クランクを見つけたときにハンドルを切る角度を設定します。

```
294 :          handle( 38 );
↓
294 :          handle( 40 );
```



3.「ビルド→ビルド」で MOT ファイルを作成します。



4.「ツール→CpuWrite」で書き込みソフトを立ち上げ、プログラムを書き込みます。転送終了後、CPU 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。ケーブルは繋いだままです。

**これで、kit07c の調整、書き込みができました。コースを走らせてみましょう！**



## 13.6 プロジェクト「sioservo」 サーボセンタの調整のプログラム解説

### 13.6.1 プロジェクトの構成



•sioservostart.src  
 •sioservo.c  
 •car\_printf2.c  
 の3ファイルです。  
 sioservo.c、car\_printf2.c で h8\_3048.h をインクルードしています。

### 13.6.2 変数の宣言

```

37 : /*=====*/
38 : /* グローバル変数の宣言          */
39 : /*=====*/
40 : unsigned int    servo_offset;      /* サーボオフセット          */
    
```

servo\_offset 変数を宣言します。この変数の値をパソコンのキー操作で増減させます。また、この値を ITU4\_BRB に代入して、サーボを制御します。

### 13.6.3 プログラムスタートのメッセージ

```

56 :     printf(
57 :         "Servo Center Adjustment Soft¥n"
58 :         "'Z' key   : Center Value +1¥n"
59 :         "'X' key   : Center Value -1¥n"
60 :         "¥n"
61 :         "'A' key   : Center Value +10¥n"
62 :         "'S' key   : Center Value -10¥n"
63 :         "¥n"
64 :     );
65 :     printf( "%5d¥r", servo_offset );
    
```

「¥n」は改行です。プログラムリストも「¥n」に合わせて、改行して記述しています。  
 ちなみに、「¥n」は改行、「¥r」は同じ行の先頭へ戻るという意味です。

## 13.6.4 メイン関数

```

67 :     while( 1 ) {
68 :         ITU4_BRB = servo_offset;
69 :
70 :         i = get_sci( &c );
71 :         if( i == 1 ) {
72 :             switch( c ) {
73 :                 case 'Z':
74 :                 case 'z':
75 :                     servo_offset++;
76 :                     if( servo_offset > 10000 ) servo_offset = 10000;
77 :                     printf( "%5d¥r", servo_offset );
78 :                     break;
79 :
80 :                 case 'A':
81 :                 case 'a':
82 :                     servo_offset += 10;
83 :                     if( servo_offset > 10000 ) servo_offset = 10000;
84 :                     printf( "%5d¥r", servo_offset );
85 :                     break;
86 :
87 :                 case 'X':
88 :                 case 'x':
89 :                     servo_offset--;
90 :                     if( servo_offset < 1000 ) servo_offset = 1000;
91 :                     printf( "%5d¥r", servo_offset );
92 :                     break;
93 :
94 :                 case 'S':
95 :                 case 's':
96 :                     servo_offset -= 10;
97 :                     if( servo_offset < 1000 ) servo_offset = 1000;
98 :                     printf( "%5d¥r", servo_offset );
99 :                     break;
100 :
101 :                 default:
102 :                     break;
103 :             }
104 :         }
105 :     }

```

68 行で、servo\_offset の値をバッファに代入してパルスの ON 幅を変えます。

70 行の get\_sci 関数は、

戻り値 -1:受信エラー 0:受信なし 1:受信あり

が返ってくる関数です。カッコの中には、char 型変数のアドレスを入れます。受信があった場合、その変数の中に受信した1文字が入ります。

71 行で戻り値が1かチェックします。1は「受信あり」です。

受信があった場合、受信文字をチェックして、それぞれの文字に応じて servo\_offset 変数の値を変えます。

- Z**キー … servo\_offset 変数を+1します。
- A**キー … servo\_offset 変数を+10します。
- X**キー … servo\_offset 変数を-1します。
- S**キー … servo\_offset 変数を-10します。

## 13.7 プロジェクト「sioservo2」 サーボの切れ角を確かめるプログラムの解説

### 13.7.1 プロジェクトの構成



•sioservo2start.src  
 •sioservo2.c  
 •car\_printf2.c  
 の3ファイルです。  
 sioservo2.c、car\_printf2.c で h8\_3048.h をインクルードしています。

### 13.7.2 変数の宣言

```

38 : /*=====*/
39 : /* グローバル変数の宣言 */
40 : /*=====*/
41 : int          servo_angle;          /* サーボ角度 */
    
```

servo\_angle 変数を宣言します。この変数の値をパソコンのキー操作で増減させます。また、この値を ITU4\_BRB に代入して、サーボを制御します。

### 13.7.3 プログラムスタートのメッセージ

```

56 :     servo_angle = 0;
57 :     printf(
58 :         "Servo Angle Check Soft¥n"
59 :         "'Z' key  : Angle Value -1¥n"
60 :         "'X' key  : Angle Value +1¥n"
61 :         "¥n"
62 :         "'A' key  : Angle Value -5¥n"
63 :         "'S' key  : Angle Value +5¥n"
64 :         "¥n"
65 :     );
66 :     printf( "%3d¥r", servo_angle );
    
```

「¥n」は改行です。プログラムリストも「¥n」に合わせて、改行して記述しています。  
 ちなみに、「¥n」は改行、「¥r」は同じ行の先頭へ戻るとい意味です。

### 13.7.4 メイン関数

```

68 :     while( 1 ) {
69 :         ITU4_BRB = SERVO_CENTER - servo_angle * 26;
70 :
71 :         i = get_sci( &c );
72 :         if( i == 1 ) {
73 :             switch( c ) {
74 :                 case 'Z':
75 :                 case 'z':
76 :                     servo_angle--;
77 :                     if( servo_angle < -50 ) servo_angle = -50;
78 :                     printf( "%3d\u0322r", servo_angle );
79 :                     break;
80 :
81 :                 case 'X':
82 :                 case 'x':
83 :                     servo_angle++;
84 :                     if( servo_angle > 50 ) servo_angle = 50;
85 :                     printf( "%3d\u0322r", servo_angle );
86 :                     break;
87 :
88 :                 case 'A':
89 :                 case 'a':
90 :                     servo_angle -= 5;
91 :                     if( servo_angle < -50 ) servo_angle = -50;
92 :                     printf( "%3d\u0322r", servo_angle );
93 :                     break;
94 :
95 :                 case 'S':
96 :                 case 's':
97 :                     servo_angle += 5;
98 :                     if( servo_angle > 50 ) servo_angle = 50;
99 :                     printf( "%3d\u0322r", servo_angle );
100 :                     break;
101 :
102 :                 default:
103 :                     break;
104 :             }
105 :         }
106 :     }

```

69 行で、servo\_offset の値をバッファに代入してパルスの ON 幅を変えます。

式は、

$$\text{ITU4\_BRB} = \text{SERVO\_CENTER} - \text{servo\_angle} * 26;$$

↑
↑
↑

サーボセンタ
度
1度当たりの増分

です。

受信があった場合、受信文字をチェックして、それぞれの文字に応じて servo\_angle 変数の値を変えます。

- Zキー … servo\_angle 変数を-1します。
- Aキー … servo\_angle 変数を-5します。
- Xキー … servo\_angle 変数を+1します。
- Sキー … servo\_angle 変数を+5します。

## 14. プログラムの改造ポイント

### 14.1 概要

kit07.c プログラムをビルド後、CPU 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、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。

- センサの反応にばらつきがある
- 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。

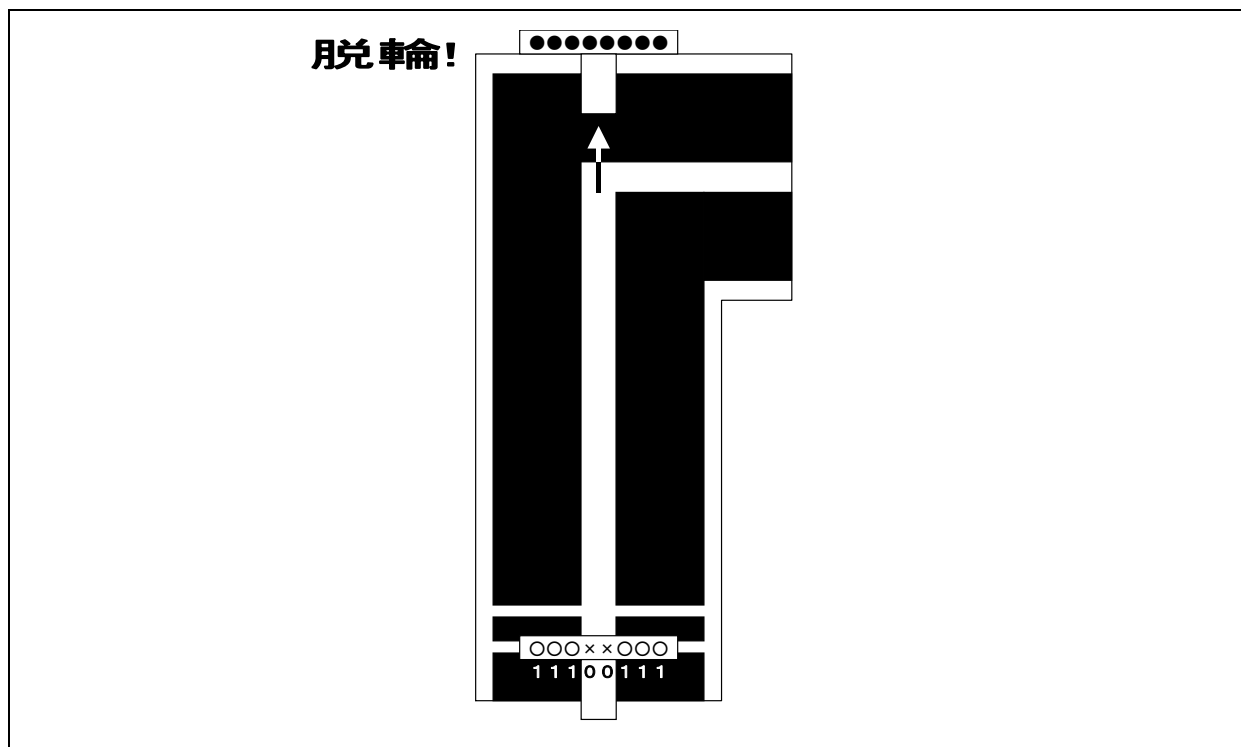
これからの説明は、データ解析実習マニュアルに掲載されている方法で、実際のセンサ状態がどうなっているか確認してみました。具体的には 10ms ごとにセンサの状態を保存、パソコンに転送してパターンとセンサの状態を確認しました。その結果を次節で説明します。

## 14.2 脱輪事例

### 14.2.1 クロスラインの検出がうまくいかない

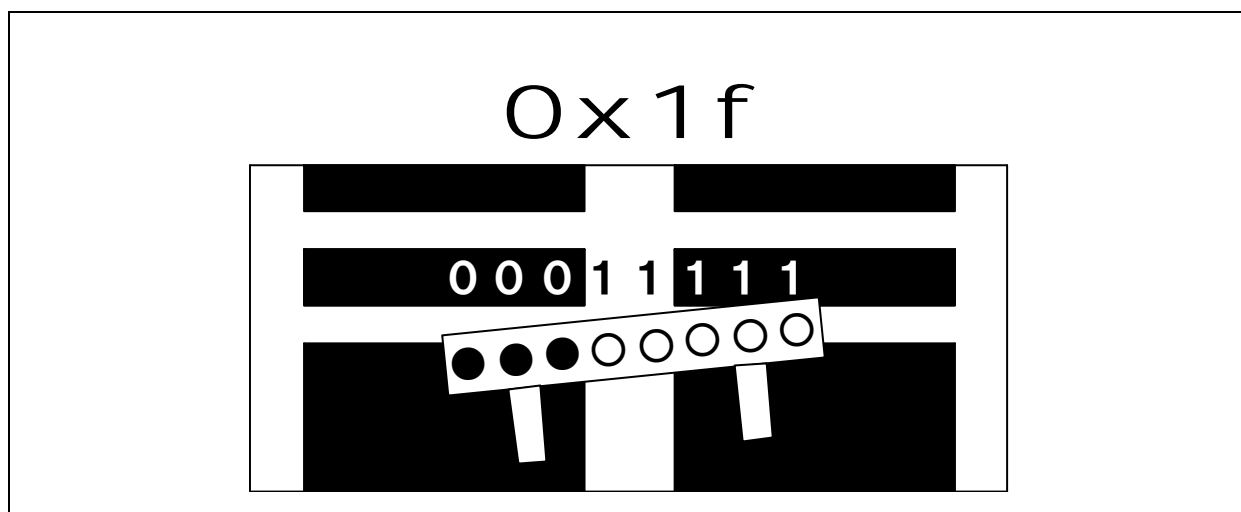
#### (a) 現象

クロスラインを検出後、クランクで曲がらずにそのまま進んで脱輪することがありました。

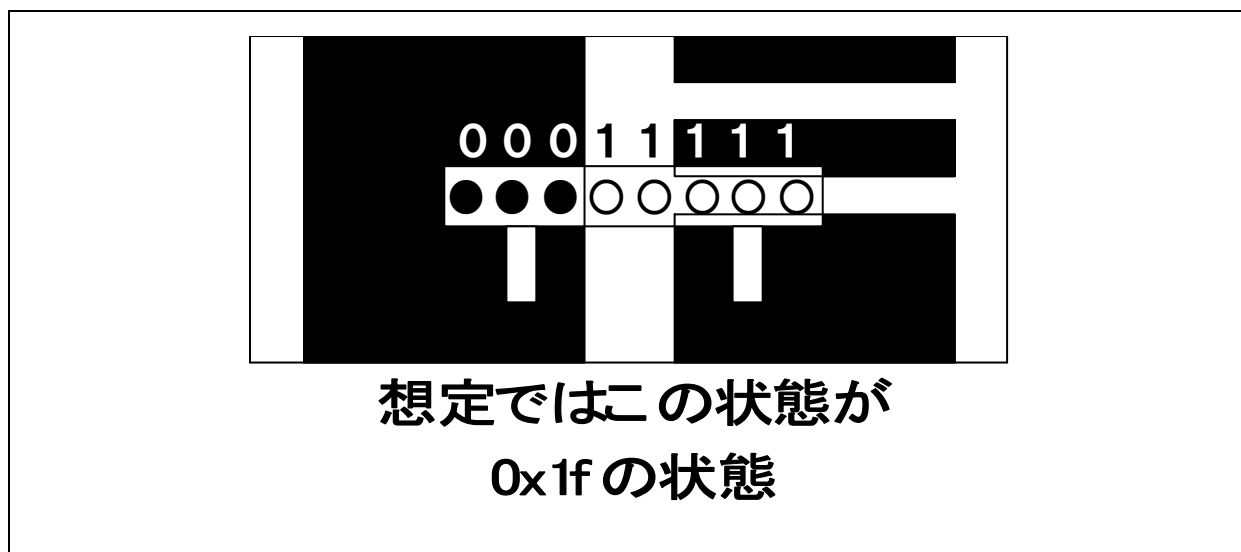


#### (b) 解析結果

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 of 「0xe7」ではなく、「0x1f」になっていることが分かりました(下図)。



「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。



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。

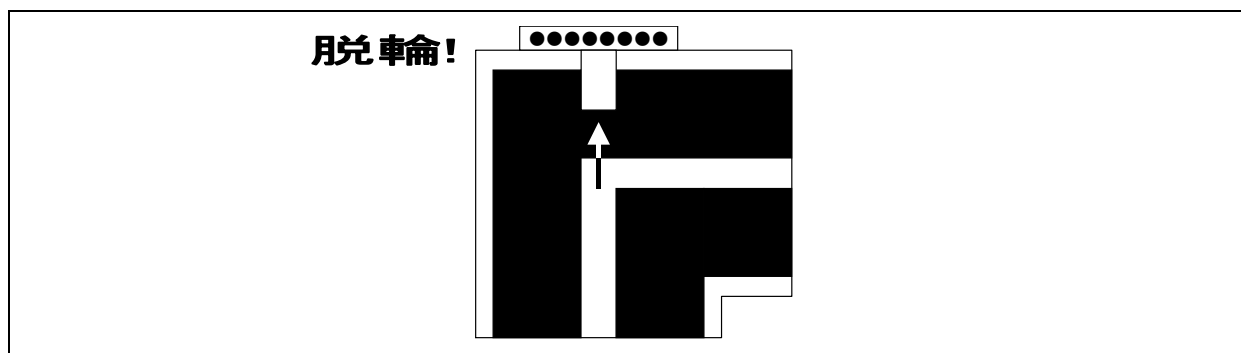
(c) 解決例

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。

### 14.2.2 クランクの検出がうまくいかない

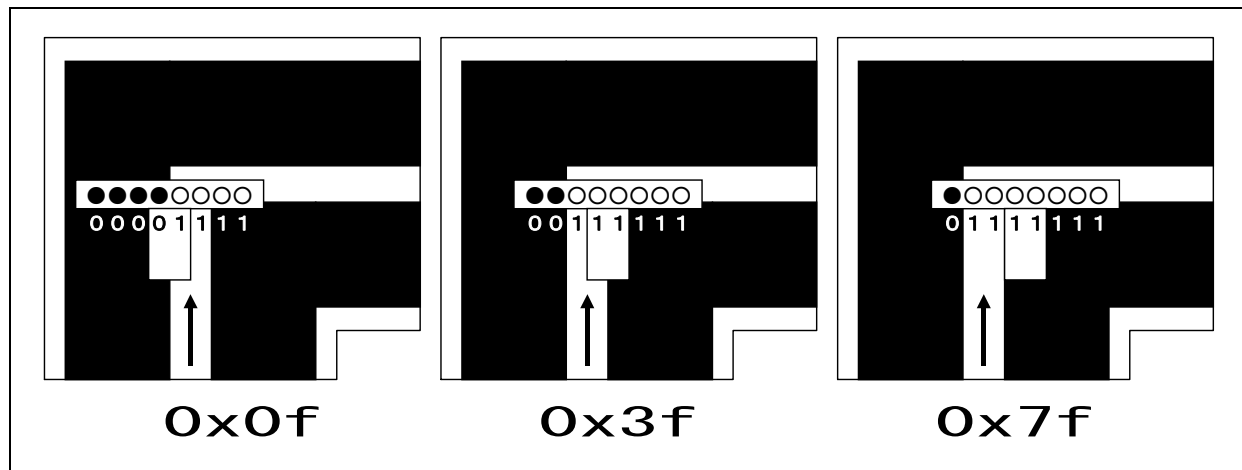
(a) 現象

クランクで曲がらずにそのまま進んで脱輪することがありました。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。



(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「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)。



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

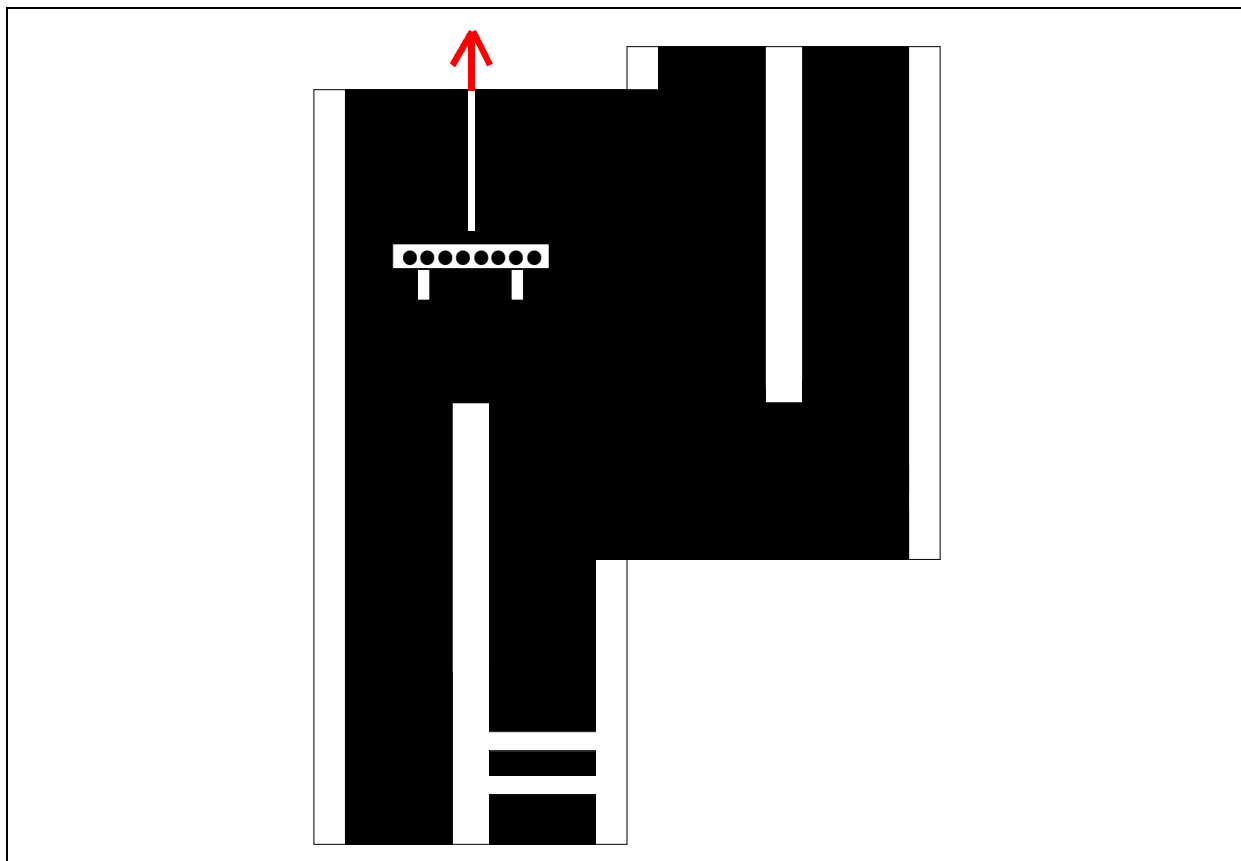
キットのプログラムでは、右クランクを検出するセンサ状態は、「0x1f」のみです。実際は、「0x0f」や「0x3f」や「0x7f」のセンサ状態もありました。そのため、これらの状態でも右クランクと判断するように、センサ状態を追加しましょう。同様に、左クランクも誤動作することが考えられますので、センサ状態を追加しておきましょう。



### 14.2.3 ハーフラインの検出がうまくいかない

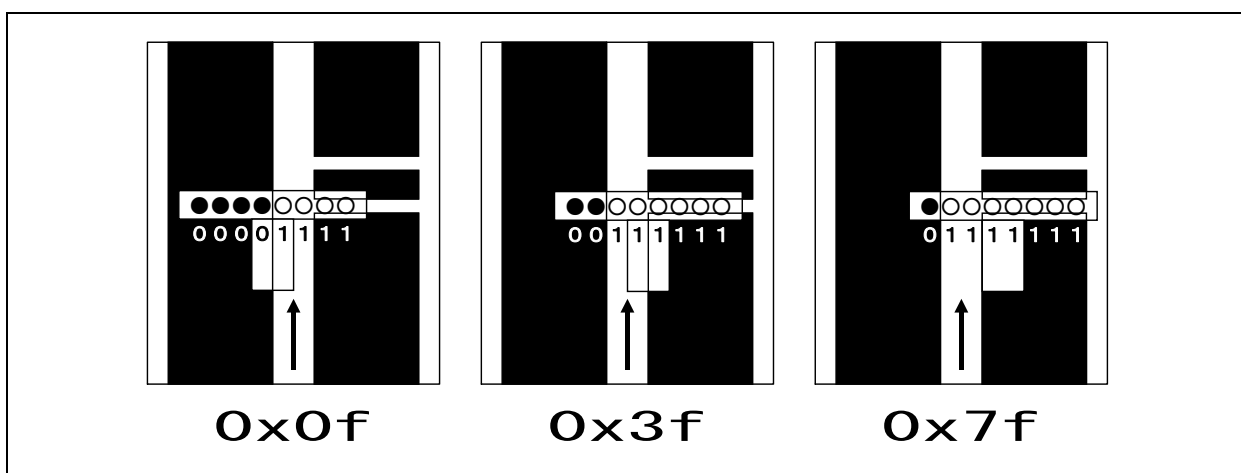
#### (a) 現象

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「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)。



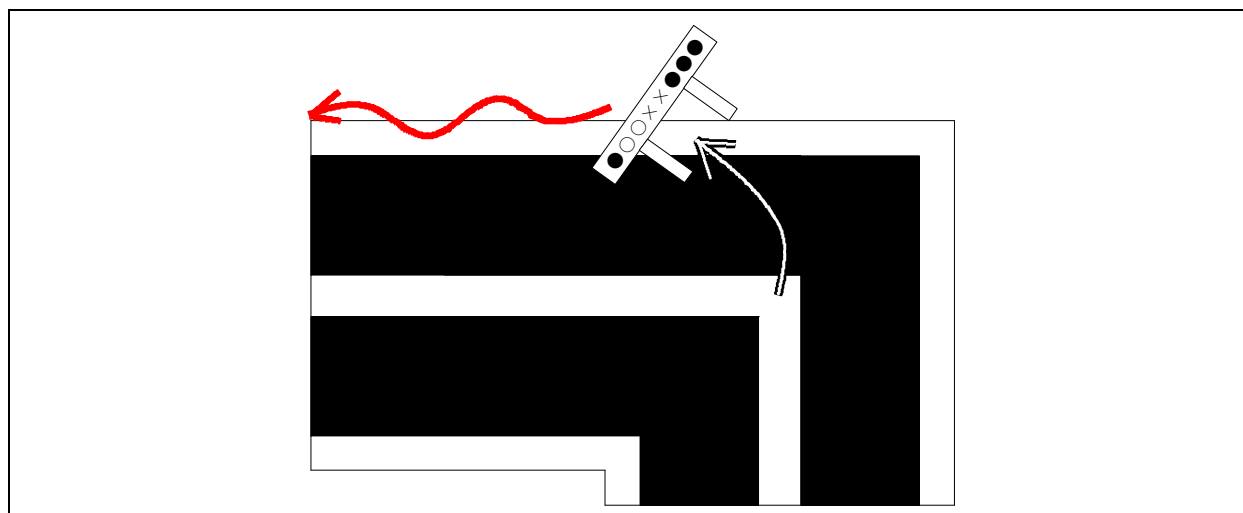
(c) 解決例

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。

14.2.4 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

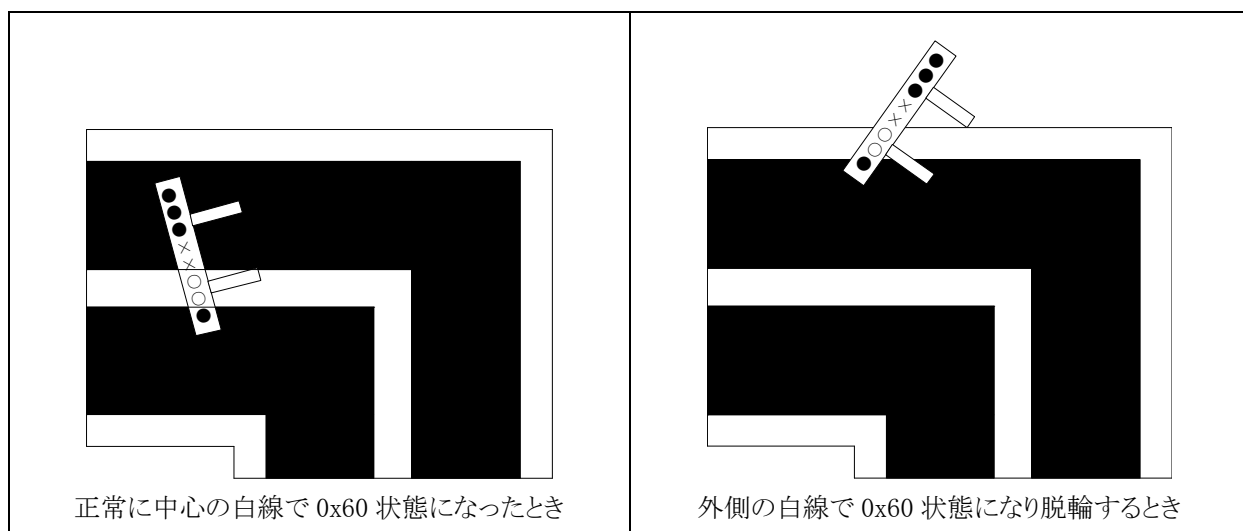
(a) 現象

左クランクを検出し、左へハンドルを切りました。若干膨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。



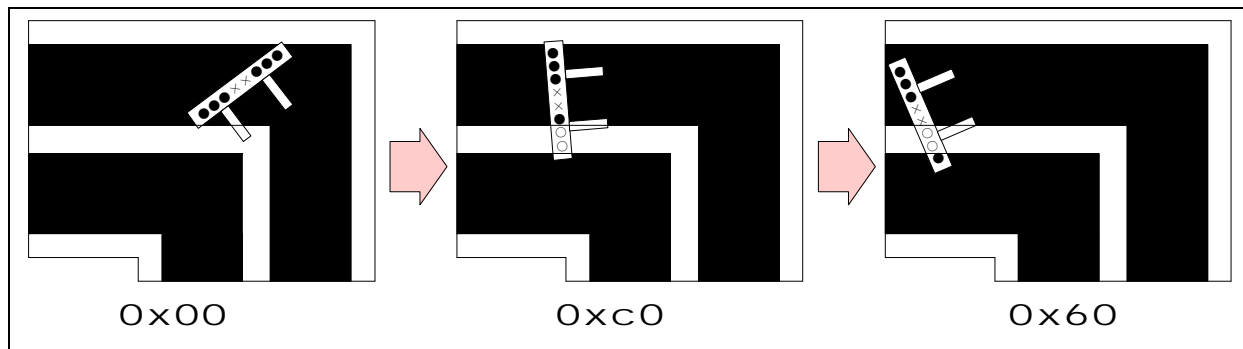
(b) 解析結果

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。

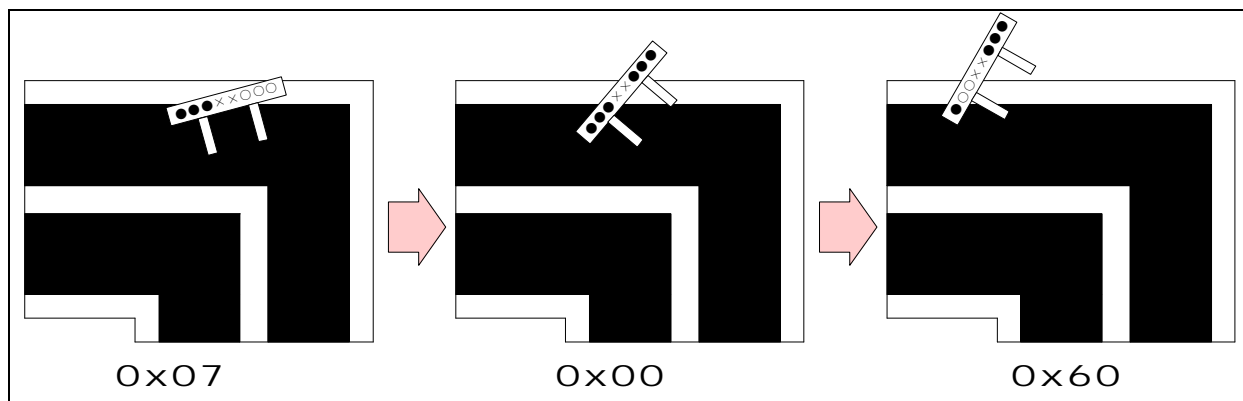


(c) 解決例

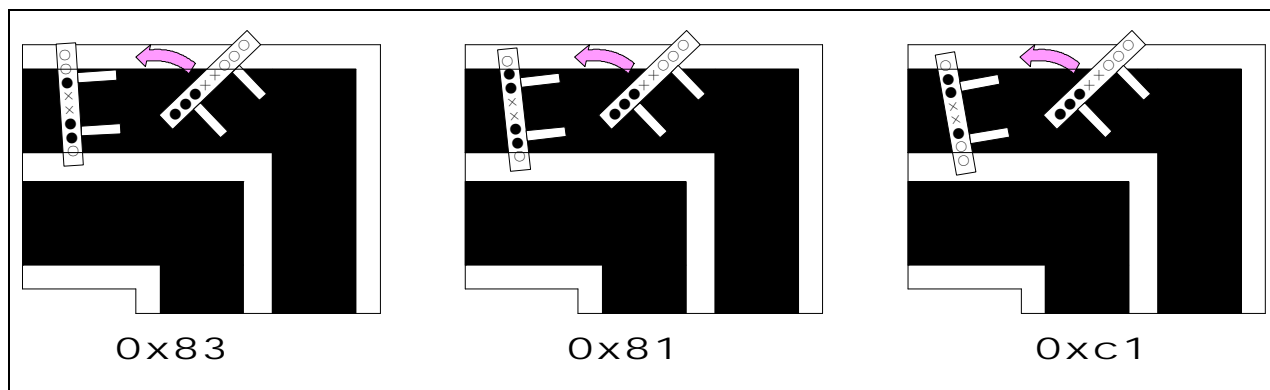
正常に中心線を見つけたときのセンサの移り変わりを確認します。0x00→0xc0→0x60のようにセンサの状態が変わり、通常走行に戻ります(下図)。



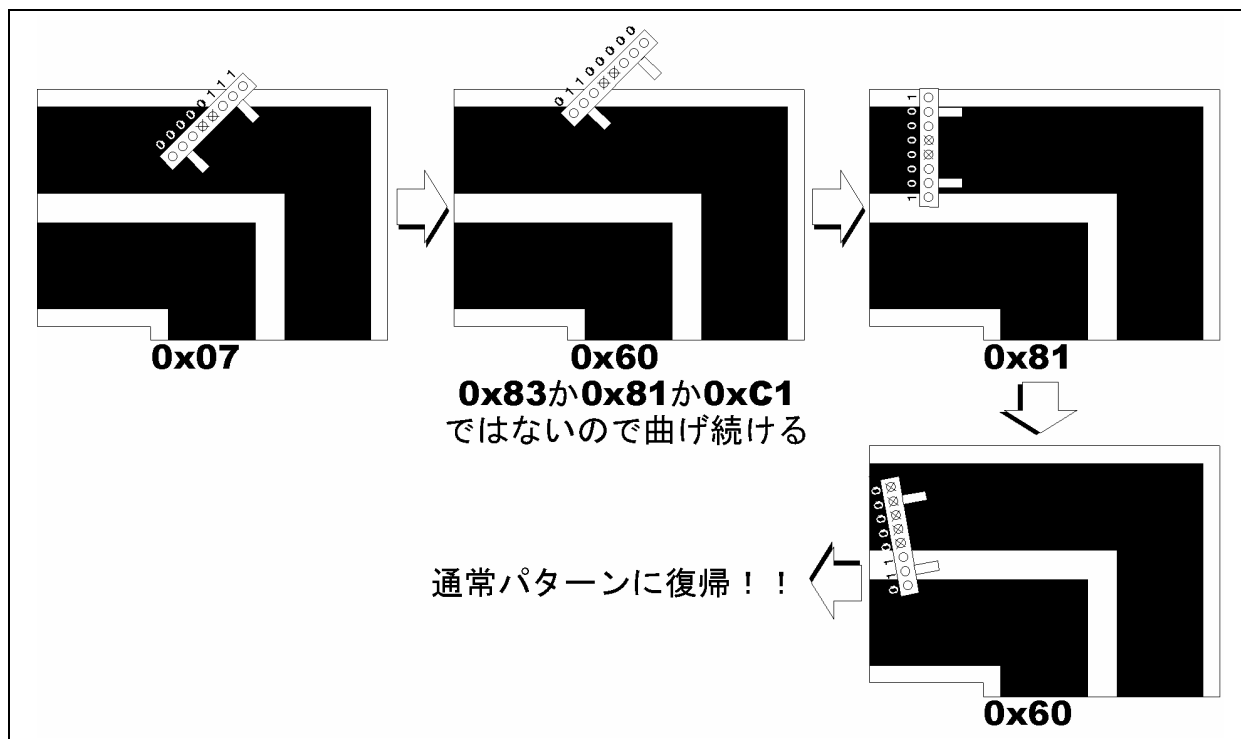
外側の白線を中心と勘違いしたときのセンサの移り変わりを確認します。0x07→0x00→0x60のようにセンサの状態が変わり、通常走行に戻ります(下図)。



2通りを見比べると、誤動作するときはセンサの状態が「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」になったら、「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。

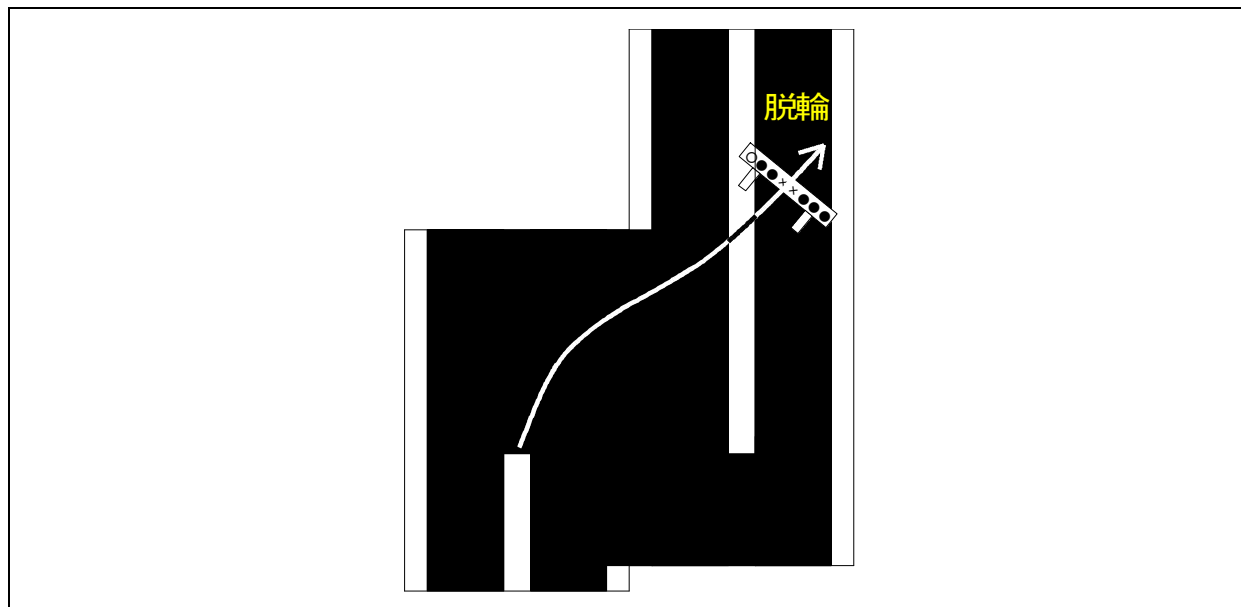


この考え方にに基づきプログラムを作ってみましょう。これで外側の白線を中心線と間違ってしまう誤動作が無くなります。右クランクも同様です。

### 14.2.5 レーンチェンジ終了の判断ができない

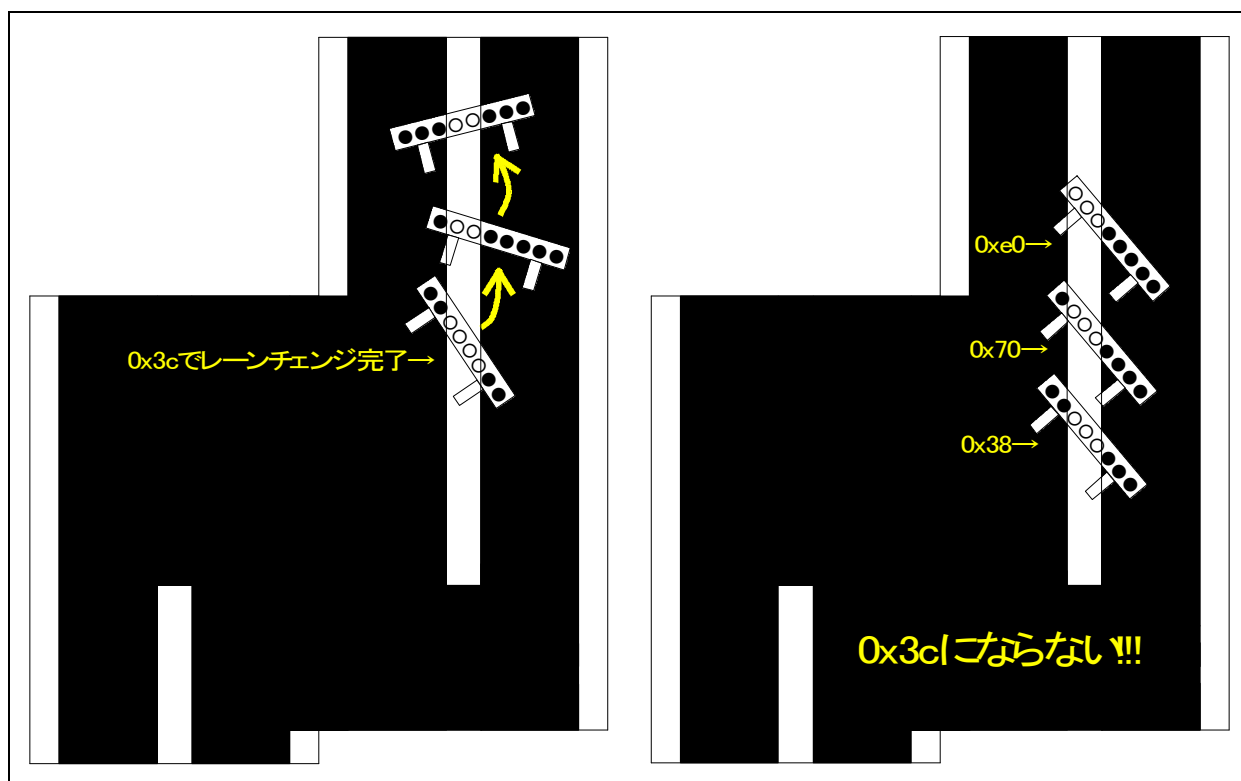
#### (a) 現象

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、そのまま真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」と変化していきましました。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。

## 14.3 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□□にしようと思います。

- 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凶解しましたが、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。

## 15. Basic Class(高校生初心者部門)

### 15.1 概要

2007 年度の大会は、下記の 3 部門が行われます。

- ・高校生の部
- ・一般の部
- ・**Japan Micom Car Rally Basic Class**(以下、Basic Class)

Basic Class は、2007 年度プレ大会として開催される部門です。これは、マイコンカー製作を通してものづくりに興味と関心を持たせると共に、ものづくり初心者にその魅力を喚起し技術者育成の裾野を広げる目的で開かれます。

### 15.2 参加資格

- 1 高等学校に在籍し、地区代表として各学校が参加を認めた者としてします。
- 2 エントリーは一人一台とし、複数台の登録はできないものとします。また、他部門への登録も認めません。
- 3 この部門への参加は、一度のみとし、過去に本大会に出場したことのない者としてします。
- 4 地区代表人数については、実行委員会が定めます。

※2007 年度は、各地区 2 名まで地区代表となります。全国大会は 12 地区から合計 24 名による予選と決勝トーナメントが行われます。

## 15.3 レギュレーション

高校生の部と Basic Class のレギュレーションを下記に示します。

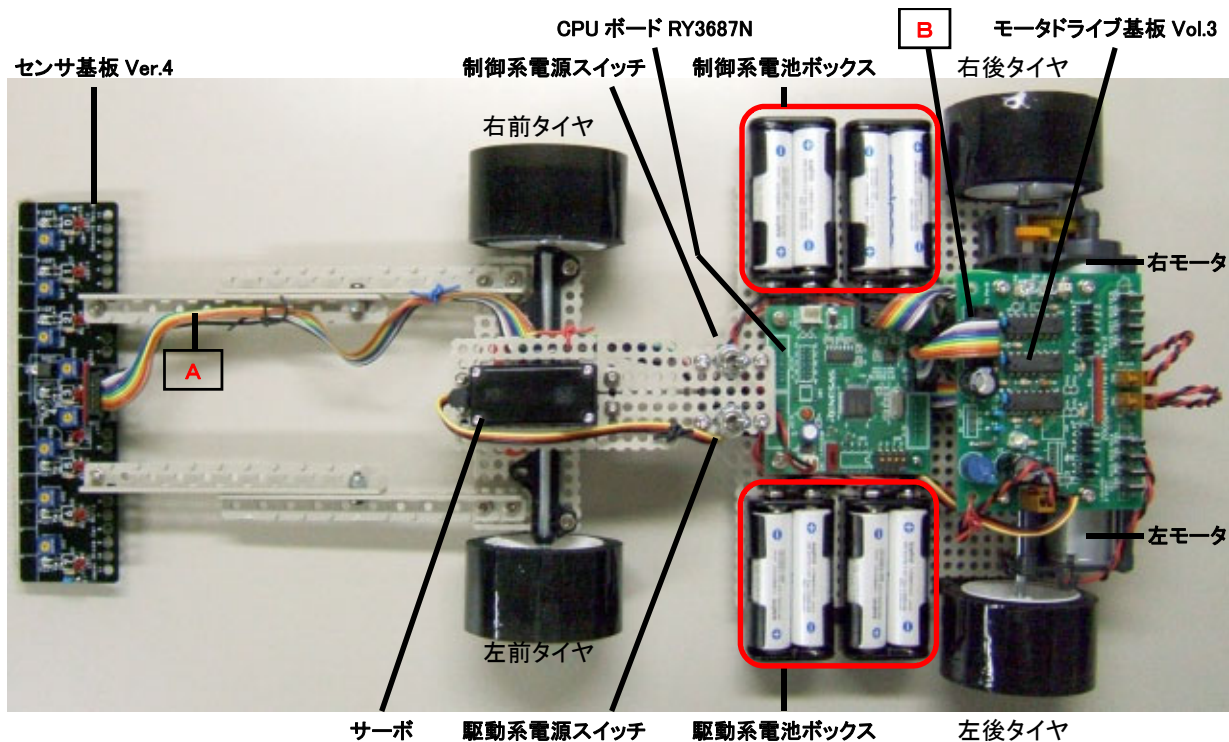
内容	高校生の部	Basic Class
CPU ボード	搭載すること。	<b>1 枚</b> 搭載すること。改造はコネクタの追加のみ認める。
電源	単三アルカリ電池 (LR6) または単三 2 次電池 (1.2V) を 8 本以内使用すること。	単三アルカリ電池 (LR6) または単三 2 次電池 (1.2V) <b>8 本</b> とし、 <b>駆動用には 4 本、CPU 用に 4 本の電池を使用すること(変圧不可)</b> 。
外形	マシンの外形は幅 300mm、高さ 150mm 以内とし全長、重量、材質等については制限しない。ただしタイマーのセンサ(レーザー光)を遮ることのできる構造とする。	
走行	マシンの駆動部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とする。	
吸引	吸引機能を用いたマシンは不可とする。	
コンデンサなど	電気二重層コンデンサ及びバックアップ用などとして用いられている大容量コンデンサ(公称値[F])の使用は不可とする。	
損傷	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とする。	
ギヤボックス	制限なし	<b>ハイスピードギヤ 2 個</b> を使用し、ケースの改造は認めない。ただし、次の点については認める。 ① 金属製のピニオンギヤ(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
モータ	支給モータの RC260RA18130(MCR刻印付き)を使用し、分解、内外部の加工を禁ずる。(ノイズ除去コンデンサ等のケースへの半田付けは除く)	支給モータの RC260RA18130(MCR刻印付き)を <b>2 個</b> 使用し、分解、内外部の加工を禁ずる。(ノイズ除去コンデンサ等のケースへの半田付けは除く)
電池ボックス	制限なし	<b>電池ボックス</b> 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
サーボ	制限なし	<b>サーボモータを 1 個</b> 使用し、その型式の確認ができる構造であること。改造は、 <b>サーボモータの基本性能を変える加工は認めない</b> 。 承認サーボ ハイテック製:HS425BB、フタバ製:S3002、サンワ製: SX-01、JR製: ES-519
距離や速度の測定	制限なし	距離及び速度を測る目的のセンサ又は機器の使用は認めない。
その他		<b>液晶の搭載は認めない</b> 。

マイコンカーキット Vol.3 やマイコンカーキット Ver.4 は Basic Class レギュレーションに適合しています。

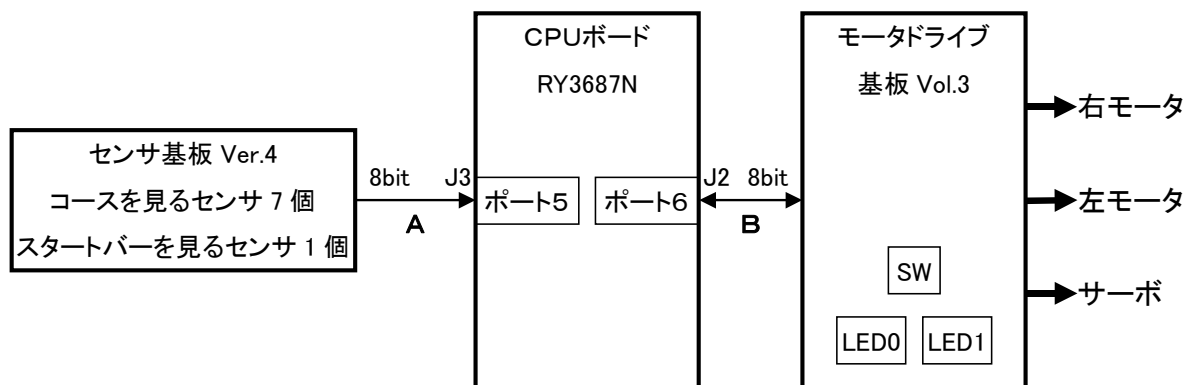


## 16. RY3687Nボードを使用したマイコンカー

### 16.1 外観



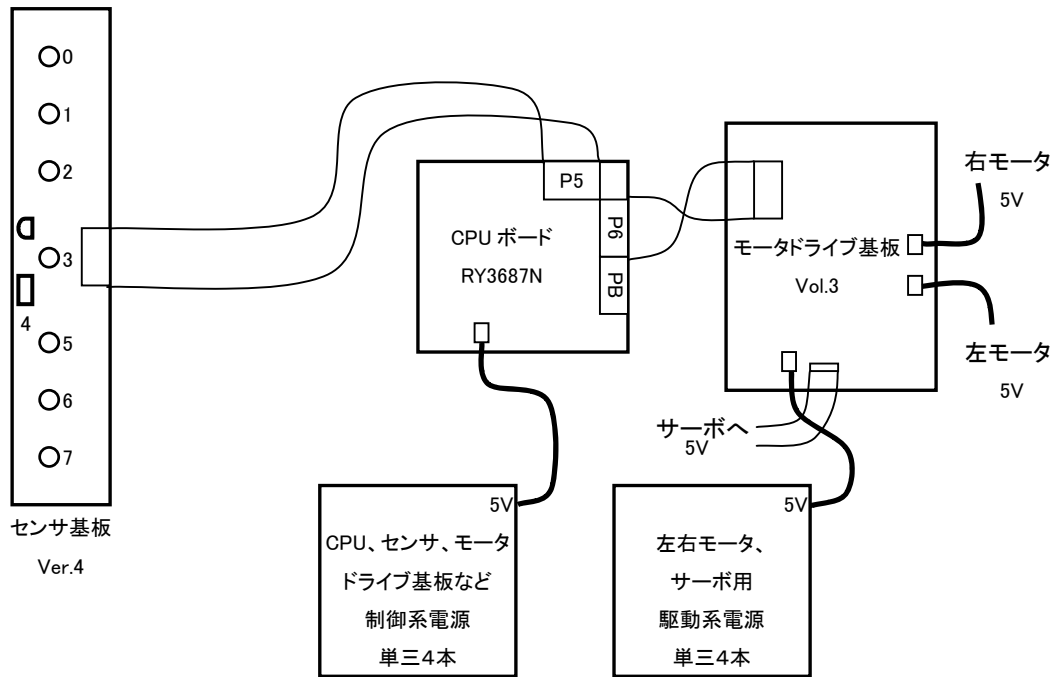
マイコンカーキットは制御系のCPUボード、センサ基板(Ver.4)、モータードライブ基板(Vol.3)、駆動系の右モーター、左モーター、サーボで構成されています。



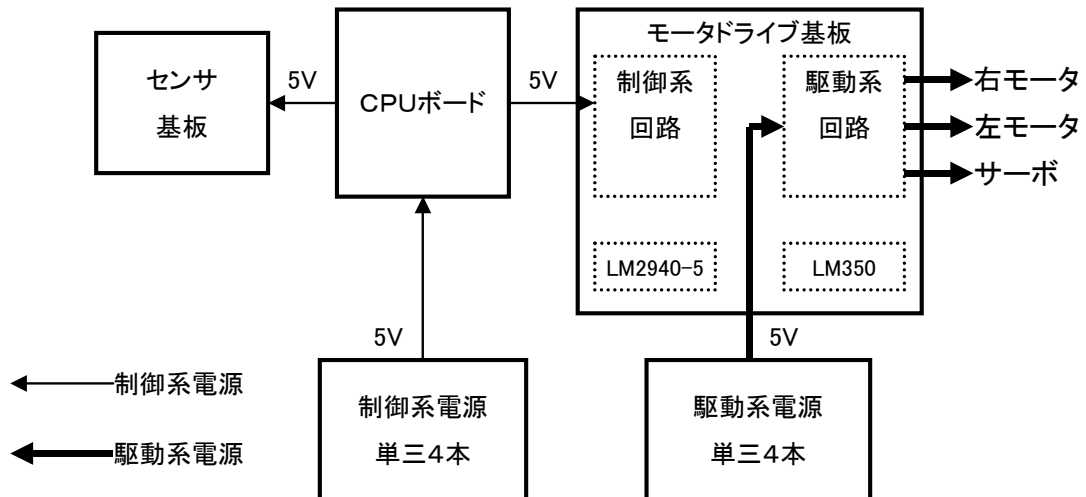
## 16.2 標準キットの電源構成

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CPU 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。

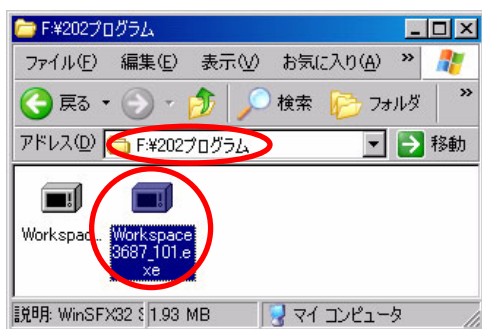


## 17. サンプルプログラム(H8/3687F版)

### 17.1 サンプルプログラムのインストール

サンプルプログラムをインストールします。

#### 17.1.1 CDからソフトを取得する



2007 年以降の講習会 CD がある場合、「CD ドライブ → 202 プログラム」フォルダにある、「Workspace3687\_101.exe」を実行します。数字の 101 は、バージョンにより異なります。

#### 17.1.2 ホームページからソフトを取得する



●ルネサス統合開発環境 H8/3048関連プログラム Ver1.21 2007.03.20  
**NEW!**  
 ルネサス統合開発環境で使用するH8/3048関係のサンプルプログラムです。自己解凍方式で、実行すると自動でプログラムがインストールされます。  
 ※ Ver1.10より、ヘッダファイルなどの共通のファイルは、「C:\workspace\common」フォルダに入れています。  
 ※ 「C:\WorkSpace\common」フォルダ等にあるi2c\_eeprom.cの日付が2006/09/21以前のプログラムにはバグがありました(EEP-ROMを使用しているポートへの出力が出来ませんでした)。お詫びして訂正致します。2006/09/22以降のi2c\_eeprom.cをご使用下さい。  
 → [DOWNLOAD](#) (EXE 約4.30MB)

●ルネサス統合開発環境 H8/3687関連プログラム Ver1.00 2007.04.03  
**NEW!**  
 ルネサス統合開発環境で使用するH8/3687関係のサンプルプログラムです。自己解凍方式で、実行すると自動でプログラムがインストールされます。  
 → [DOWNLOAD](#) (EXE 約1.42MB)

●プログラム解説マニュアルkit06版 第1.10版 2006.09.04  
プログラムのインストール、実行、終了、および、再実行の主なプログラム

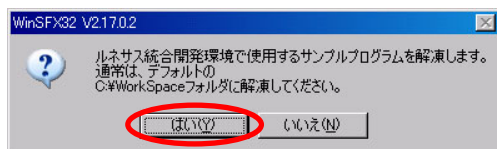
#### 1. マイコンラリーサイト

「<http://www.mcr.gr.jp/>」の技術情報→ダウンロード内のページへ行きます。

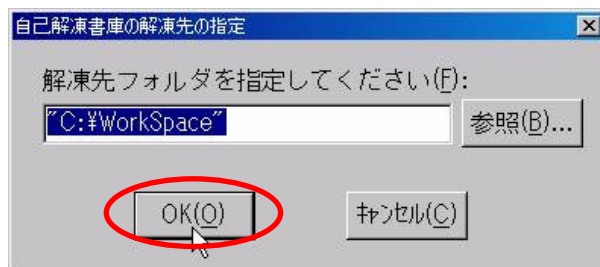
#### 2. 「ルネサス統合開発環境 H8/3687 関連プログラム」

をダウンロードします。

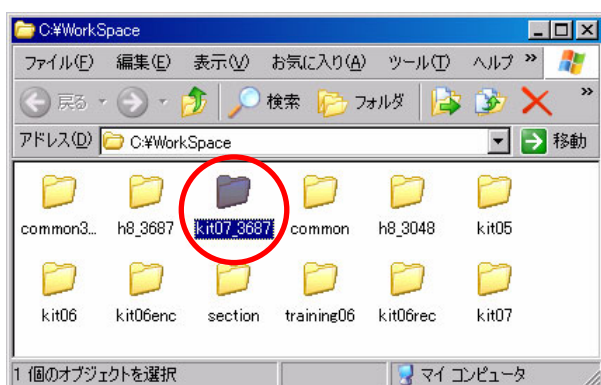
### 17.1.3 インストール



1. CD またはダウンロードした「Workspace3687\_101.exe」を実行します。「はい」をクリックします。



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

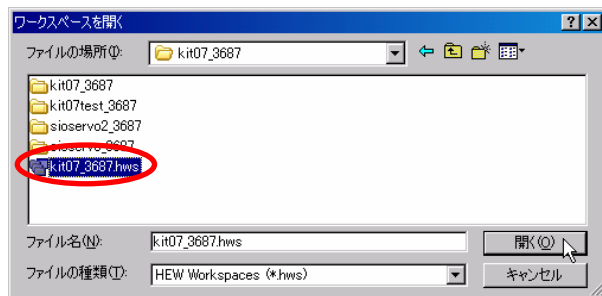
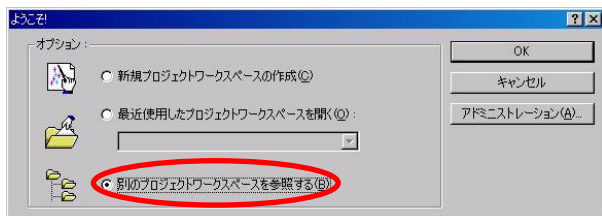


3. 解凍が終わったら、エクスプローラで「C ドライブ→Workspace」フォルダを開いてみてください。複数のフォルダがあります。今回使用するののは、「**kit07\_3687**」です。

## 17.2 ワークスペース「kit07\_3687」を開く

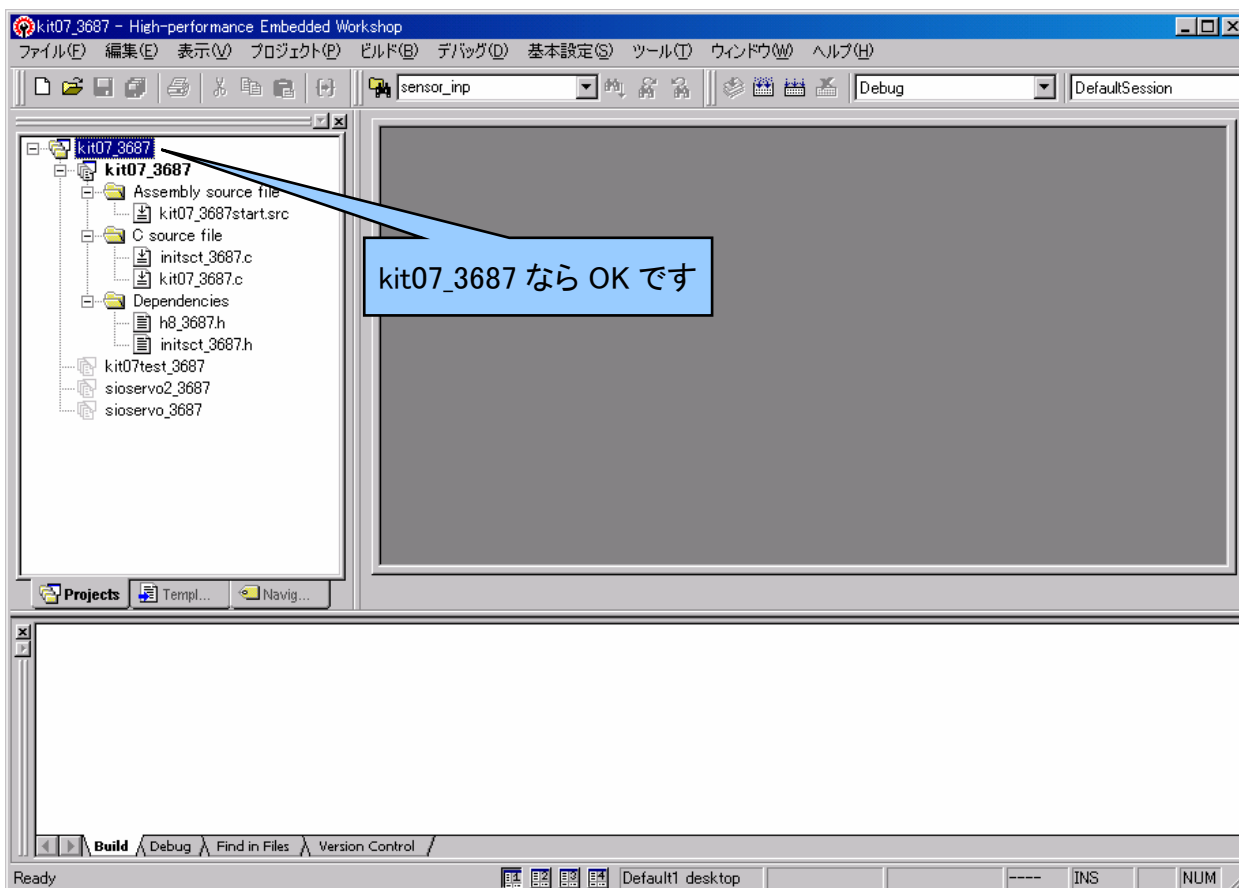


ルネサス統合開発環境を実行します。



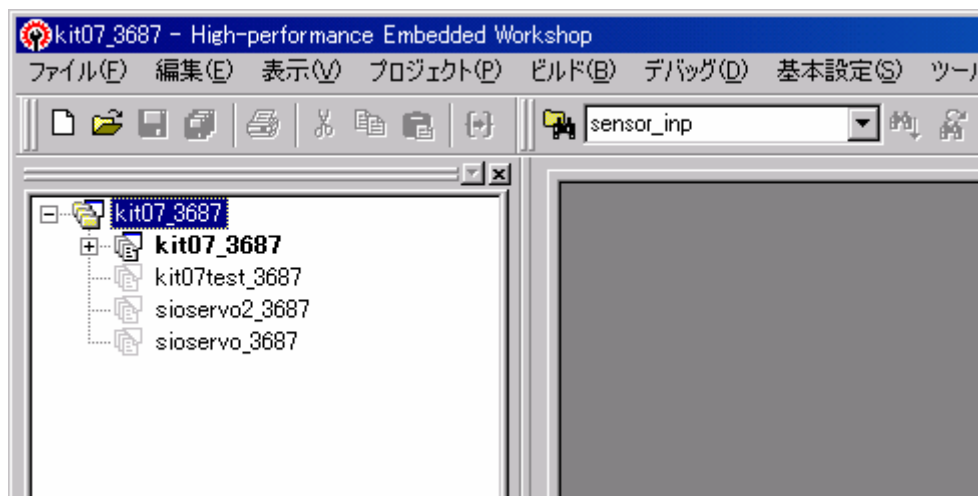
「別のプロジェクトワークスペースを参照する」を選択します。

Cドライブ→Workspace→kit07\_3687 の「kit07\_3687.hws」を選択します。



kit07\_3687 というワークスペースが開かれます。

## 17.3 プロジェクト

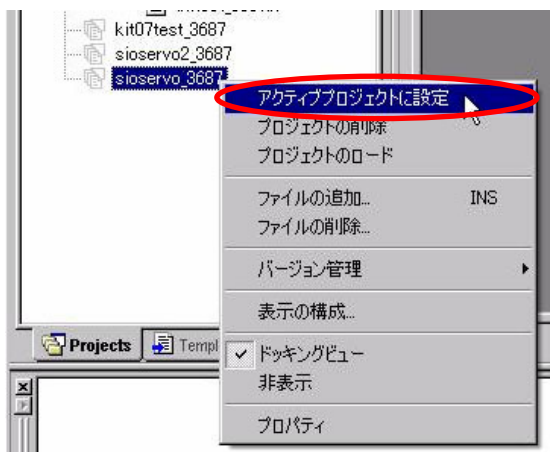


ワークスペース「kit07\_3687」には、4つのプロジェクトが登録されています。

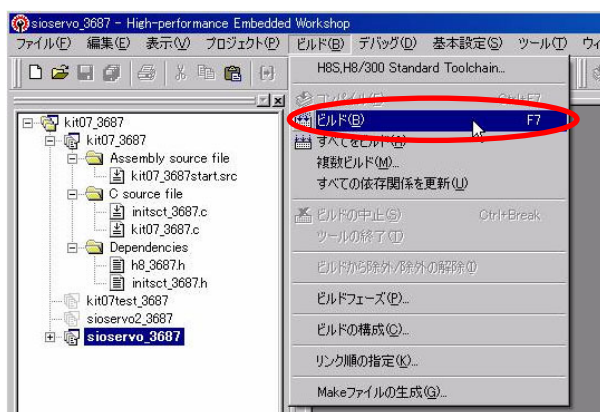
プロジェクト名	内容
kit07_3687	マイコンカー走行プログラムです。H8/3687F 版です。
kit07test_3687	製作したマイコンカーのモータドライブ基板やセンサ基板が正しく動作するかテストします。H8/3687F 版です。
sioservo_3687	サーボのセンタを調整するプログラムです。H8/3687F 版です。
sioservo2_3687	サーボの最大切れ角を見つけるためのプログラムです。H8/3687F 版です。

これらのプログラムは、ワークスペース「kit07」の H8/3048F-ONE に関する部分(主に内蔵周辺機能)を、H8/3687F 用書き換えた内容です。

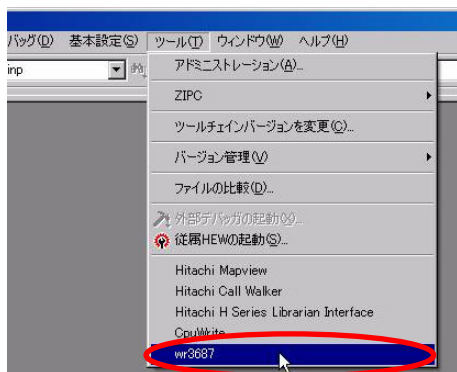
## 17.4 サーボのセンタを調整する



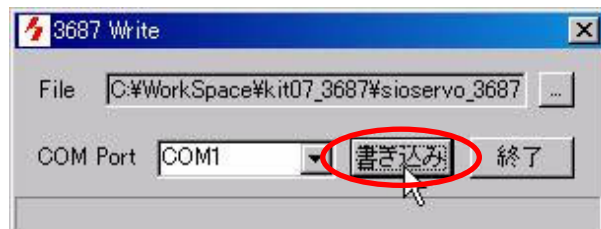
1. プロジェクト「sioservo\_3687」をアクティブプロジェクトに設定します。



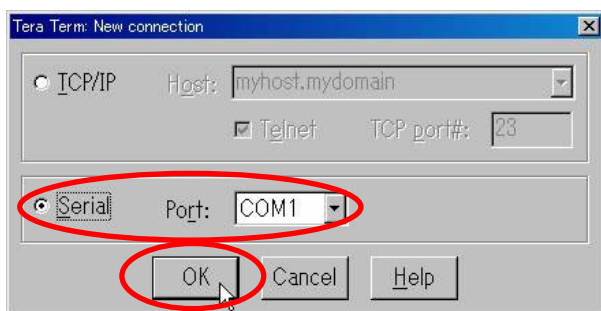
2. 「ビルド→ビルド」でビルドします。MOT ファイルができ上がります。



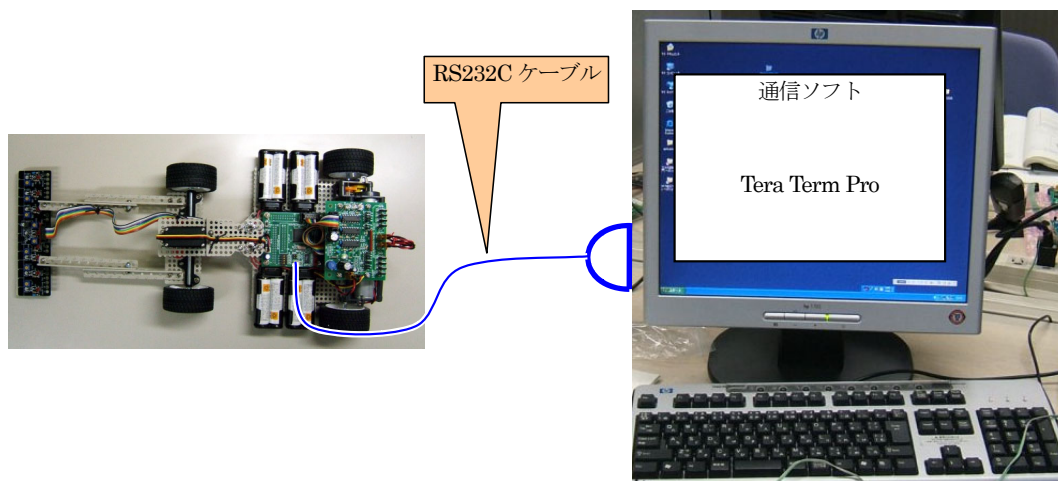
3. ルネサス統合開発環境の「ツール→wr3687」で書き込みソフトを立ち上げます。



4. 「書き込み」をクリックして、プログラムを書き込みます。転送終了後、CPU 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。ケーブルは繋いだままです。



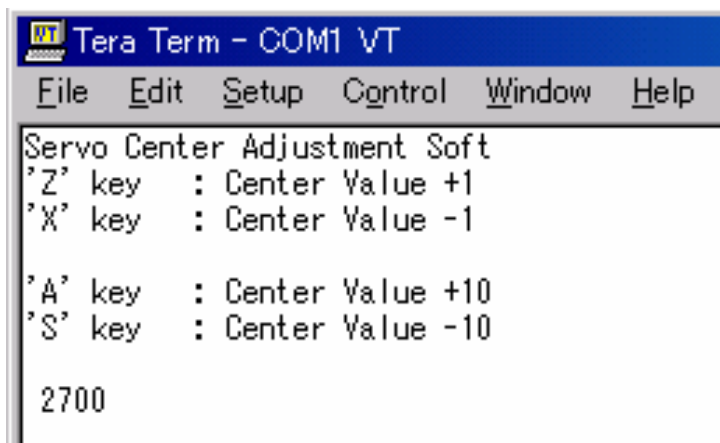
5. 「スタート→プログラム→Tera Term Pro→Tera Term Pro」で Tera Term Pro を立ち上げます。Serial を選択して、ポートを書き込みポートの番号に合わせます。



6.接続、設定を確かめます。

- ・マイコンカーとパソコンを RS232C ケーブルで接続しているか
- ・TeraTermPro を立ち上げて、ポートを設定しているか

OK なら次に進みます。



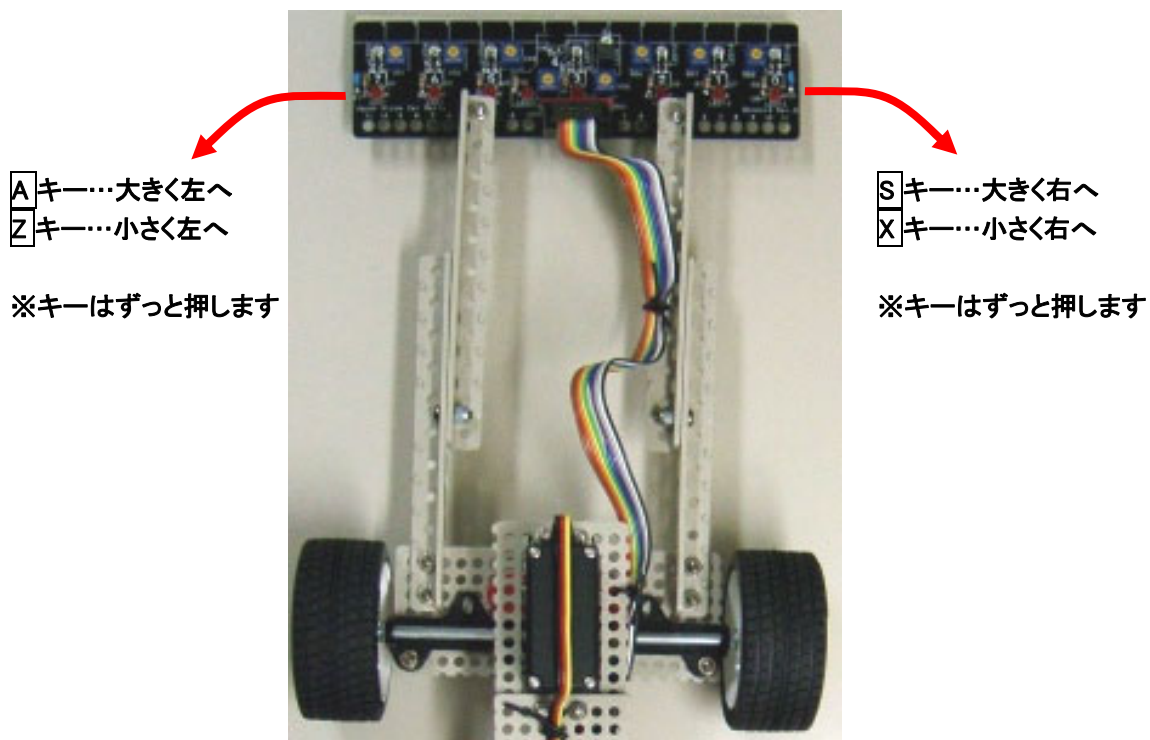
7.マイコンカーの電源を入れるとメッセージが表示されます。表示されない場合は、ケーブルの接続やマイコンカーの電池、書き込みスイッチを戻したか、通信ポートの番号、書き込んだプログラムが本当にプロジェクト「sioservo\_3687」のsioservo\_3687.mot を書き込んだかなど確かめてください。

#### ポイント

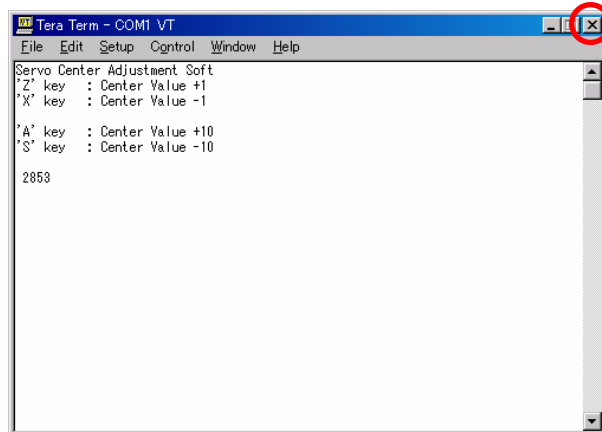
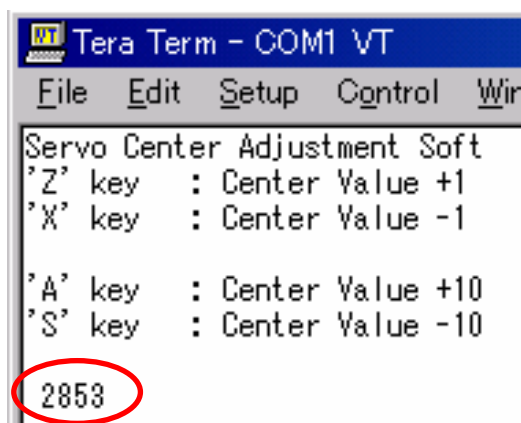
TeraTermPro を立ち上げてから、**一番最後にマイコンカーの電源を入れます。**

電源を入れた瞬間にマイコンカーからメッセージを出力しますので、その後に TeraTermPro を立ち上げてても何も表示されません。表示内容は、マイコンカーから送られてきます。





8. **A**、**S**、**Z**、**X** キーをそれぞれ押し続けるとサーボが動きます。キーを使ってサーボがまっすぐ向く角度に調整してください。

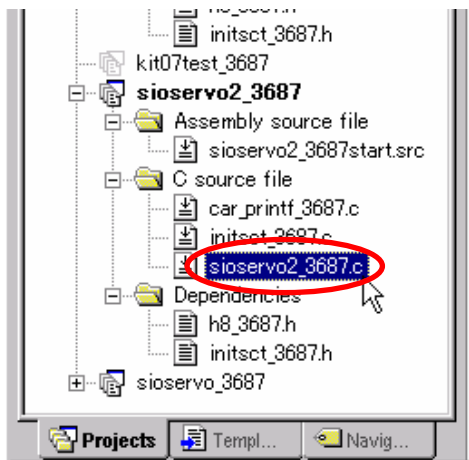
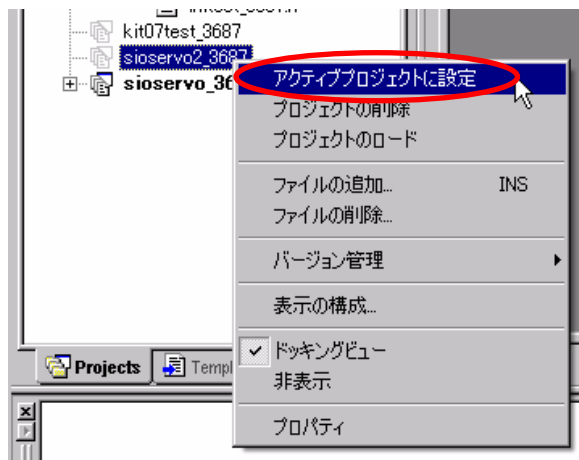


9. まっすぐに調整できたら、Tera Term Pro の数字を見ます。今回は「2853」とでました。これがこのマイコンカーのサーボセンタ(SERVO\_CENTER)の値です。メモしておきます。

10. 次は、プロジェクト「sioservo2\_3687」に進みます。**X**をクリックして TeraTermPro を終了しておきます。マイコンカーの電源も切ります。

## 17.5 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\_3687」をアクティブプロジェクトに設定します。
2. 「sioservo2\_3687.c」をダブルクリックしてエディタウィンドウを開きます。

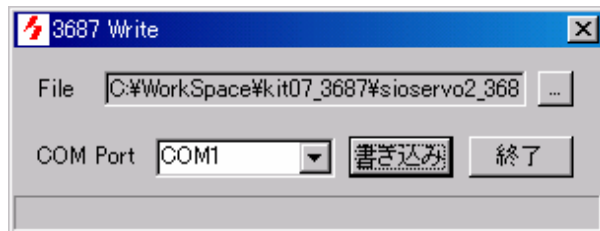
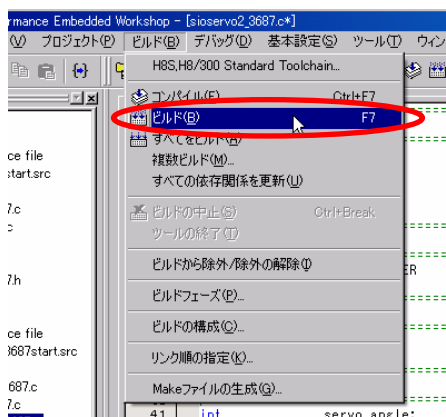
```

24  #include <stdio.h>
25  #include <machine.h>
26  #include "h8_3687.h"
27
28  /*=====*/
29  /* シンボル定義 */
30  /*=====*/
31  #define SERVO_CENTER 2700 /*
32
33  /*=====*/
34  /* プロトタイプ宣言 */
35  /*=====*/
36  void init( void );
37
38  /*=====*/
39  /* グローバル変数の宣言 */
40  /*=====*/
41  int servo_angle; /*
    
```

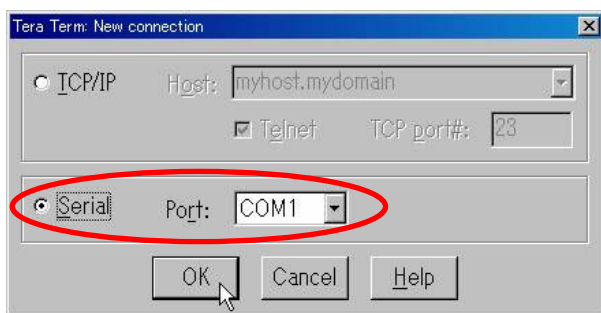
```

24  #include <stdio.h>
25  #include <machine.h>
26  #include "h8_3687.h"
27
28  /*=====*/
29  /* シンボル定義 */
30  /*=====*/
31  #define SERVO_CENTER 2853 /*
32
33  /*=====*/
34  /* プロトタイプ宣言 */
35  /*=====*/
36  void init( void );
37
38  /*=====*/
39  /* グローバル変数の宣言 */
40  /*=====*/
41  int servo_angle; /*
    
```

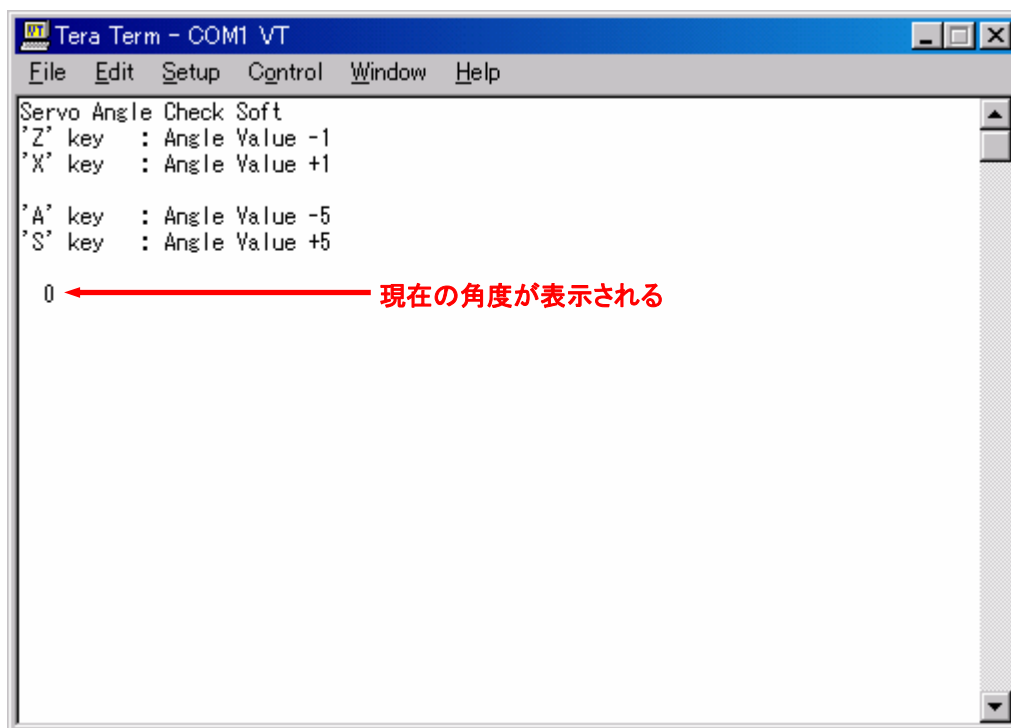
3. 31 行に  
SERVO\_CENTER 2700  
という記述があります。
4. 先ほど見つけたサーボセンタ値に書き換えます。画面は、例として「2853」に書き換えています。



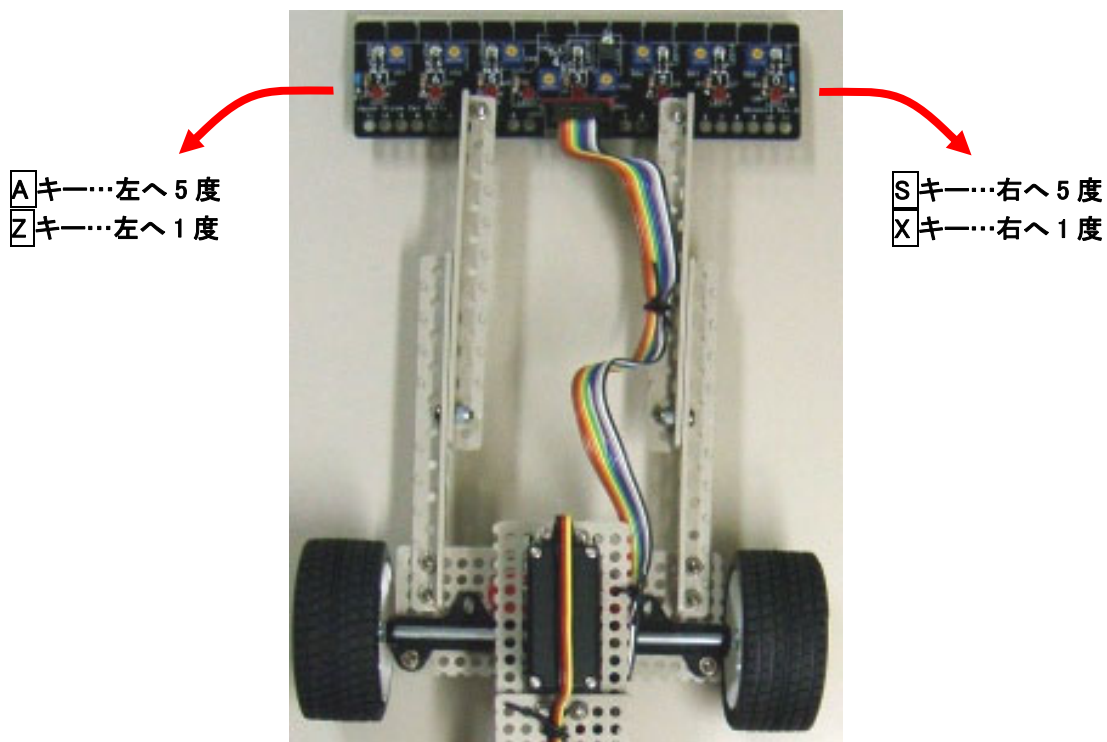
5. 「ビルド→ビルド」で MOT ファイルを作成します。
6. 「ツール→wr3687」で書き込みソフトを立ち上げ、プログラムを書き込みます。転送終了後、CPU ボードの電源を OFF にして書き込みスイッチを内側にしておきます。ケーブルは繋いだままです。



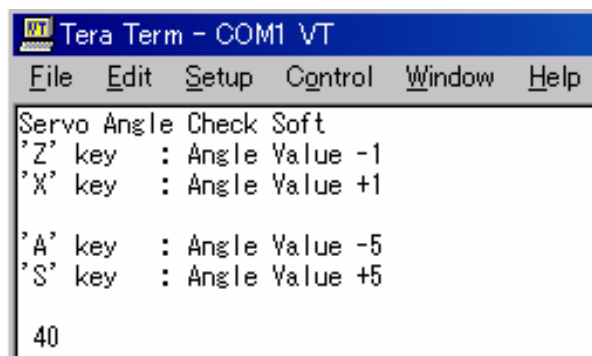
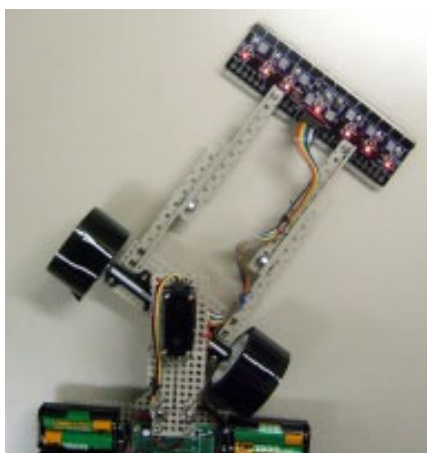
7. 「スタート→プログラム→Tera Term Pro→Tera Term Pro」で Tera Term Pro を立ち上げます。Serial を選択して、ポートを書き込みポートの番号に合わせます。



8. マイコンカーの電源を入れるとメッセージが表示されます。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「sioservo2\_3687.mot」かなど確かめてください。

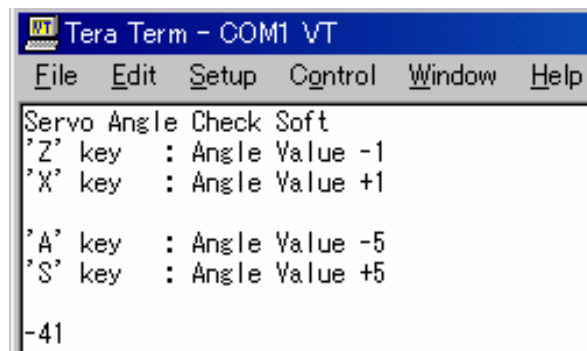
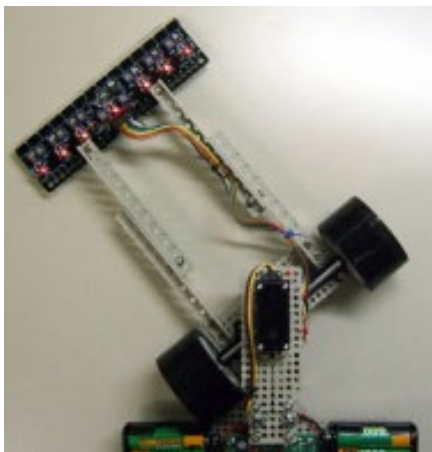


9. **A**、**S**、**Z**、**X** キーをそれぞれ押すとサーボが動きます。右はどこまでハンドルを曲げることができるかを調べます。左も同様に調べます。



10. 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 **Z** キーでもう少し小さくしてください。

11. Tera Term Pro 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。**

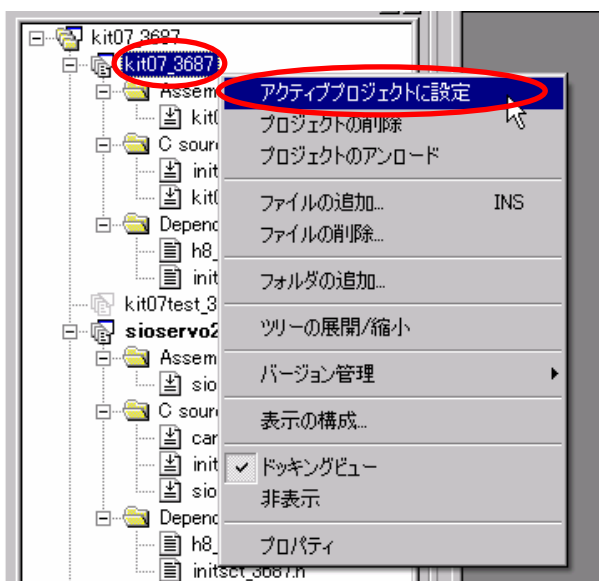


12.今度は [A] キー、[Z] キーで逆の左の限界を見つけます。こちらもタイヤを回して回るか確かめてください。シャーシにぶつかるようなら [X] キーでもう少し小さくしてください。

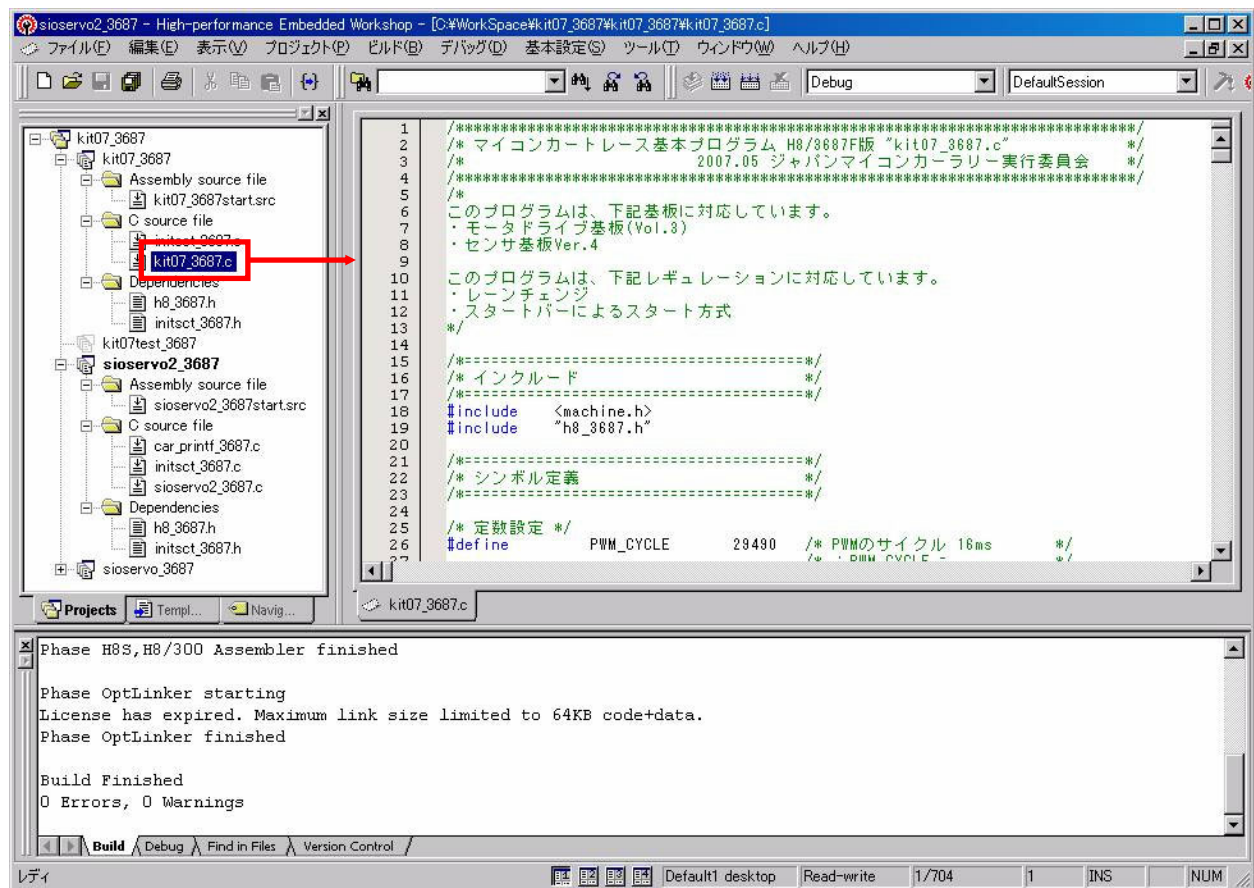
13.Tera Term Pro の数値を見ます。これが現在ハンドルを左へ曲げている角度です。**-41 度曲げていると**言うことが分かりました。

## 17.6 「kit07\_3687.c」プログラムを書き換える

プロジェクト「sioservo\_3687」、「sioservo2\_3687」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である「kit07\_3687.c」へ書き込みます。このファイルは、プロジェクト「kit07\_3687」内にあります。



1.ワークスペース「kit07\_3687」のプロジェクト「kit07\_3687」をアクティブプロジェクトに設定します。



2.「kit07\_3687.c」をダブルクリックして、エディタウィンドウに表示させます。  
 下記のように変更します。

内容	kit07_3687.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	30 行	2700	2853
左の最大角度	279 行	-38	-41
右の最大角度	288 行	38	40

まず、サーボセンタです。

```
30 : #define      SERVO_CENTER    2700    /* サーボのセンタ値      */
```



```
30 : #define      SERVO_CENTER    2853 /* サーボのセンタ値      */
```

次に左の最大角度です。この部分は、左クランクを見つけたときにハンドルを切る角度を設定します。

```
279 :          handle( -38 );
```



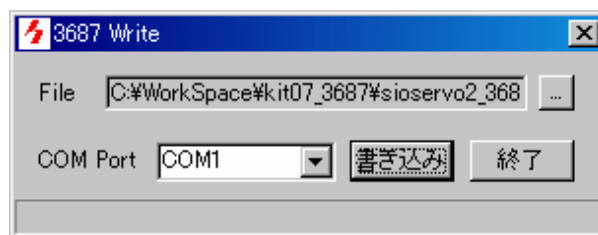
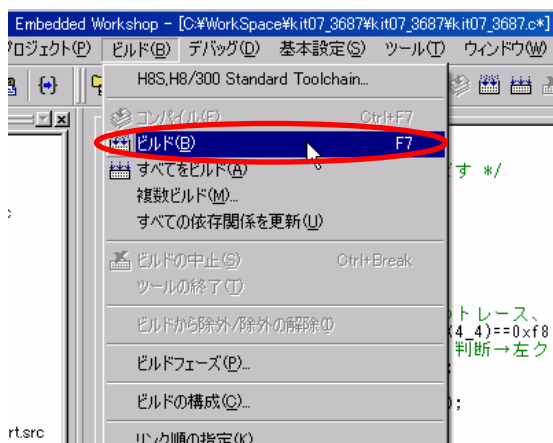
```
279 :          handle( -41 );
```

次に右の最大角度です。この部分は、右クランクを見つけたときにハンドルを切る角度を設定します。

```
288 :          handle( 38 );
```



```
288 :          handle( 40 );
```



3.「ビルド→ビルド」で MOT ファイルを作成します。

4.「ツール→wr3687」で書き込みソフトを立ち上げ、プログラムを書き込みます。転送終了後、CPU ボードの電源を OFF にして書き込みスイッチを内側におきます。ケーブルは繋いだままです。

**これで、kit07\_3687.c の調整、書き込みができました。コースを走らせてみましょう！**

## 18. 参考文献

- (株)ルネサス テクノロジ  
H8/3048 シリーズ、H8/3048F-ZTAT™ (H8/3048F、H8/3048F-ONE)ハードウェアマニュアル 第7版
- (株)ルネサス テクノロジ  
H8/3687 シリーズ ハードウェアマニュアル 第3版
- (株)ルネサス テクノロジ  
High-performance Embedded Workshop V.4.00 ユーザーズマニュアル Rev.3.00
- (株)ルネサス テクノロジ 半導体トレーニングセンター C言語入門コーステキスト 第1版
- (株)オーム社 H8 マイコン完全マニュアル 藤澤幸徳著 第1版
- 電波新聞社 マイコン入門講座 大須賀威彦著 第1版
- 電波新聞社 C言語でH8マイコンを使いこなす 鹿取祐二著 第1版
- 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

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

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

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

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

の「マイコン」→「H8 ファミリ」、または「マイコン」→「Tiny」でご覧頂けます

※リンクは、2007年9月現在の情報です。