マイコンカーラリーキット Vol.3 対応

プログラム
解説マニュアル
kit06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 1.20 版

2007.05.09

ジャパンマイコンカーラリー実行委員会

注 意 事 項(rev.1.1)

著作権

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

禁止事項

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

転載、複製

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

責任の制限

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

その他

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

連絡先

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

目　次

1. 概要

本マニュアルでは、
・マイコンカーラリーキット(Vol.3)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。

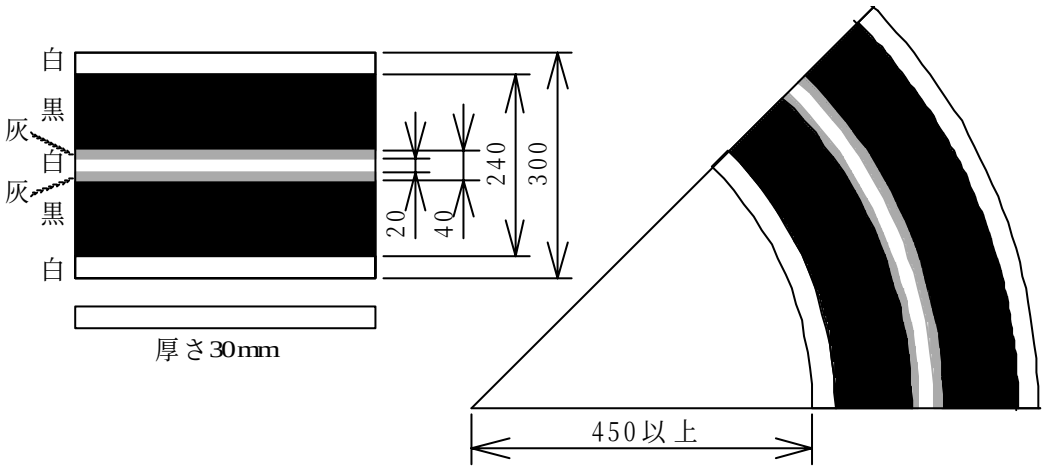
	車体の変遷	プログラムの変遷	主なルール変更
1998 年頃	**マイコンカーキット Vol.1** を開発しました。	プログラム名「tmc4.c」 マイコンカーキット Vol.1 に対応したプログラムです。	
2002 年	**マイコンカーキット Vol.2** を開発しました。モータドライブ基板に電池 8 本分の電圧を加えられる回路になりました。センサ基板が小型化されました。	プログラム名「kit2.c」 マイコンカーキット Vol.2 に対応したプログラムです。	駆動モータが、ジャパンマイコンカーラリー指定モータのみの使用となりました。
2004 年	特に変化はありません。	プログラム名「kit04.c」 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「**マイコンカーキット Vol.3**」としました。	プログラム名「kit05.c」 モータドライブ基板 Vol.3 に対応したプログラムです。	
2006 年	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、レーンチェンジが導入されました。

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

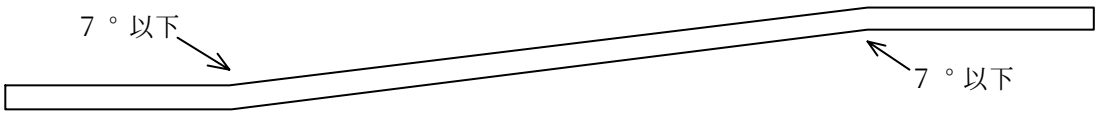
マイコンカーラリーのコースは、直線、カーブ、クランク、坂、レーンチェンジで構成されています。マイコンカーラリーのコースは、幅 300mm の中に、黒、灰、白色があります。カーブは内径が 450mm 以上となっています。マイコンカーに取り付けているセンサによりコースとマイコンカーのずれを検出し、コースに沿って走るように制御します。

マイコンカーキットには、白色か黒色かを判断できるセンサが 8 個取り付けられています。灰色は、センサ基板のボリュームを調整することにより白色か黒色かどちらかと同じ反応にします。これから説明するプログラムは、灰色と白色を同じように判断すると走行できるようになっています。



2.2 上り、下り坂

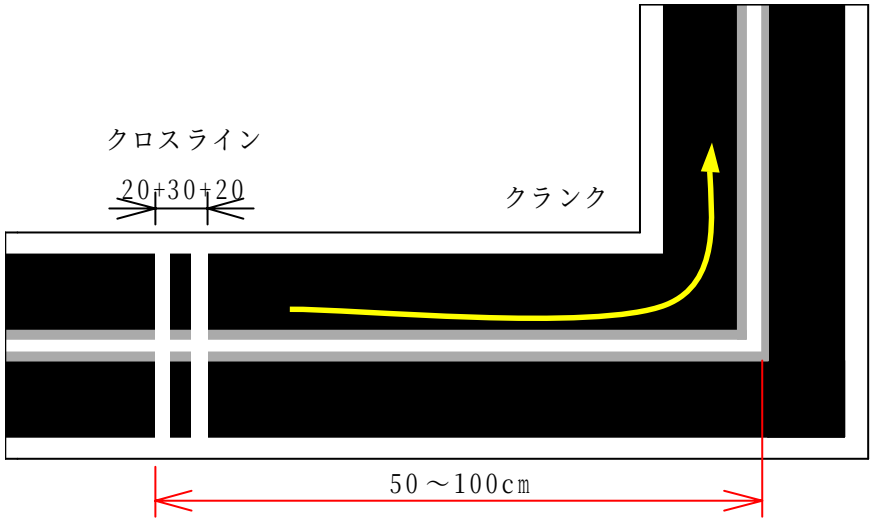
角度が 7 度以下の上り坂、下り坂があります。上り初め、上り終わり、下り初め、下り終わりでマイコンカーのシャーシやセンサ部分がコースとこすらないように製作する必要があります。



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のコースの特徴は、このクランク（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。

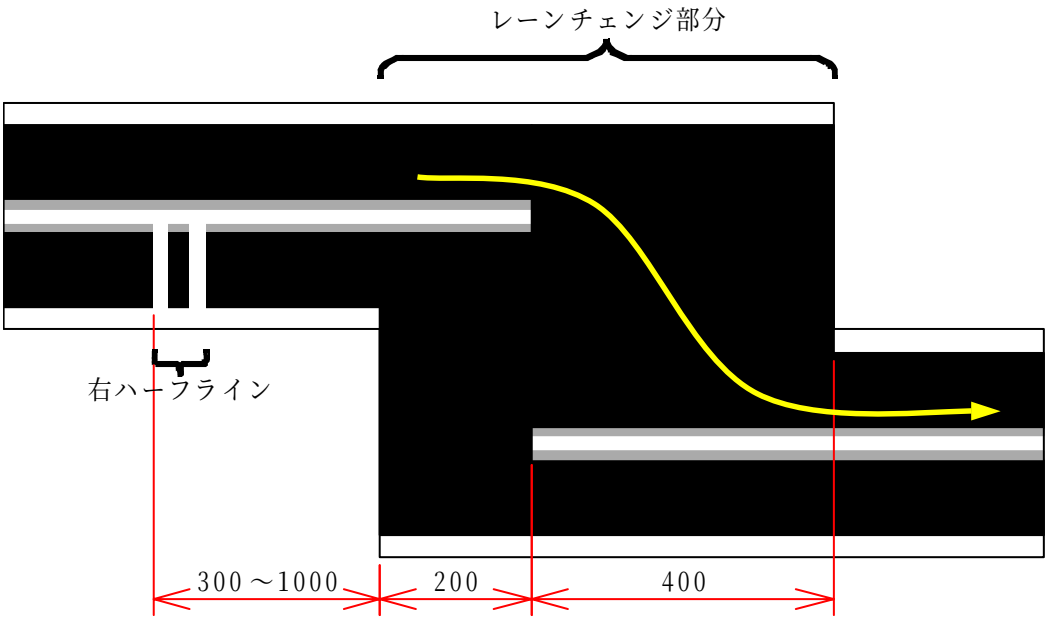
クランク手前の 50～100cm にはクロスラインと呼ばれる2本の白ラインが引かれています。マイコンカーはこのラインを検出すると、直角を曲がれるスピードまで減速します。直角を発見すると曲がり、通常走行に戻ります。



2.4 レーンチェンジ部分

ジャパンマイコンカーラリー2007 大会（2006 年度）より追加されたコースです。今までは中心の白線は連続していました。初めて中心の白線が途切れるコースとなっています。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。

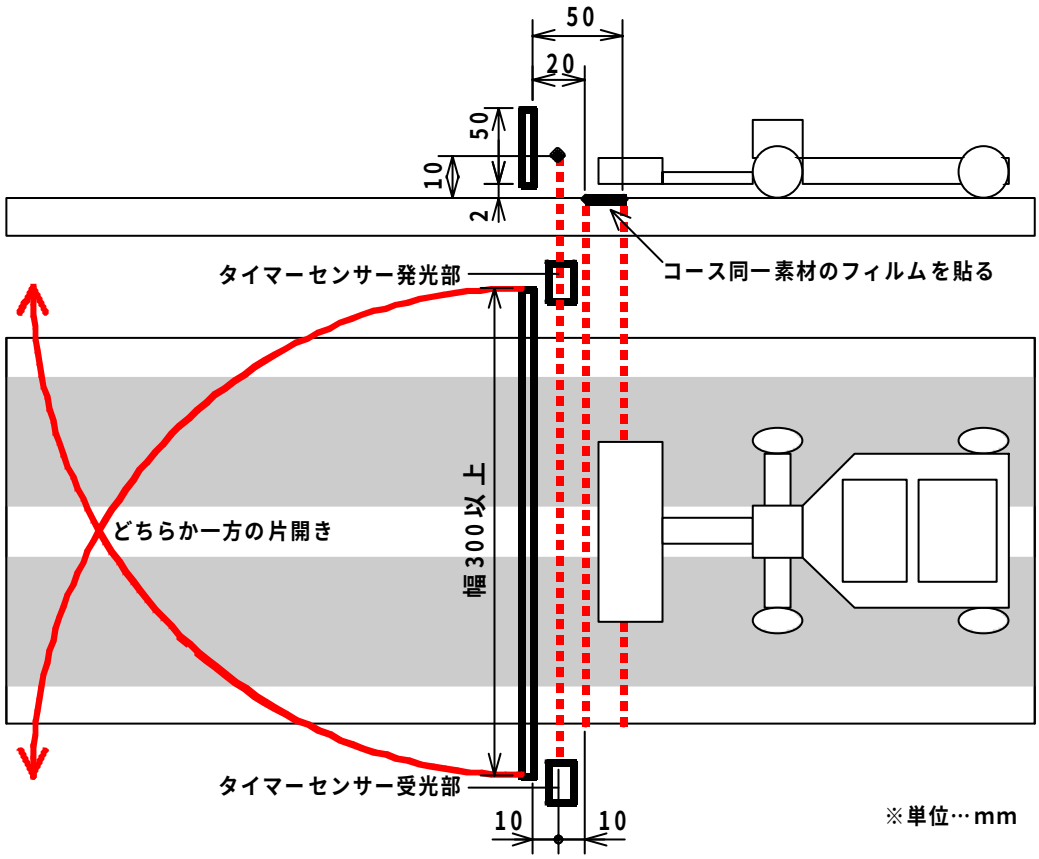
レーンチェンジは右へのレーンチェンジ、左へのレーンチェンジの 2 通りあります。右なら、レーンチェンジ部分 300mm～1000mm 手前にコース中心から右端まで白色2本のハーフラインがあり、それを発見すると右レーンチェンジと判断します。左レーンチェンジなら、コース中心から左端まで白色 2 本のハーフラインがあります。マイコンカーは中心線が無くなるとレーンチェンジを開始し、新しい中心線を見つけるとレーンチェンジ完了と判断し、通常走行に戻ります。下記に右レーンチェンジのコースを示します。



2.5 スタートバー部分

ジャパンマイコンカーラリー2007 大会（2006 年度）からスタート時、競馬のようにバーが開くと同時に計測を開始する方式に変更になりました。今までは、スタートセンサを通過して計測開始、再度通過してゴールでした。下記に変更されたスタート手順を示します。

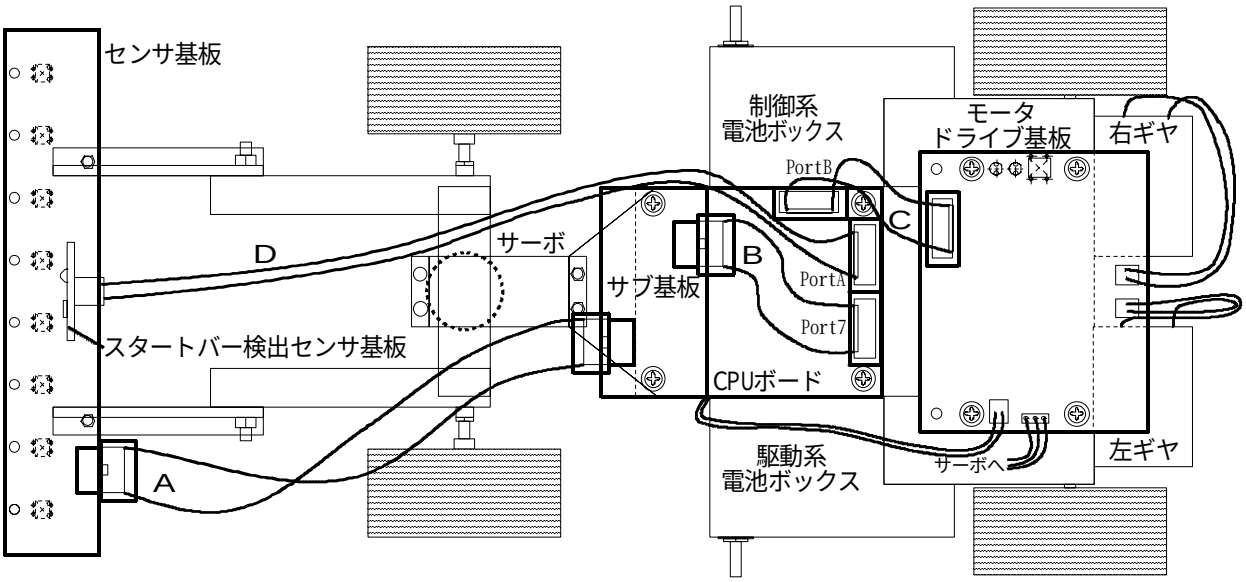
1. 選手は、マイコンカーの先頭とスタートバーの距離を 20～50mm 離してマイコンカーをセットします(コースと同一素材の継ぎテープを貼り、段差を目印とします)。
2. 審判が、マイコンカーをセットできたか確認します。選手は、準備ができたら審判にセットできた旨を伝えます（一般的に手を挙げて合図します）。セット後は、マイコンカーに触れることはできません。
3. スタートバー（表面はコース材質の白色のシールが貼ってあります）が進行方向に開きます（押し扉のイメージ）。
4. マイコンカーは、スタートバーが開いたことを自動で検出してスタートします。
5. スタートバーが開くと同時に、タイム計測が開始されます。
6. 選手が、スタートバーが開いたことを確認してからスタートさせることもできます。
7. スタートバーが開いた後マイコンカーがスタートしない場合は、スイッチの入れ忘れやコネクタの差込確認など、短時間でできる作業を行うことができます（今まで同様）。ただし、タイマーセンサーを通過したマイコンカーに触れた場合は失格となります。



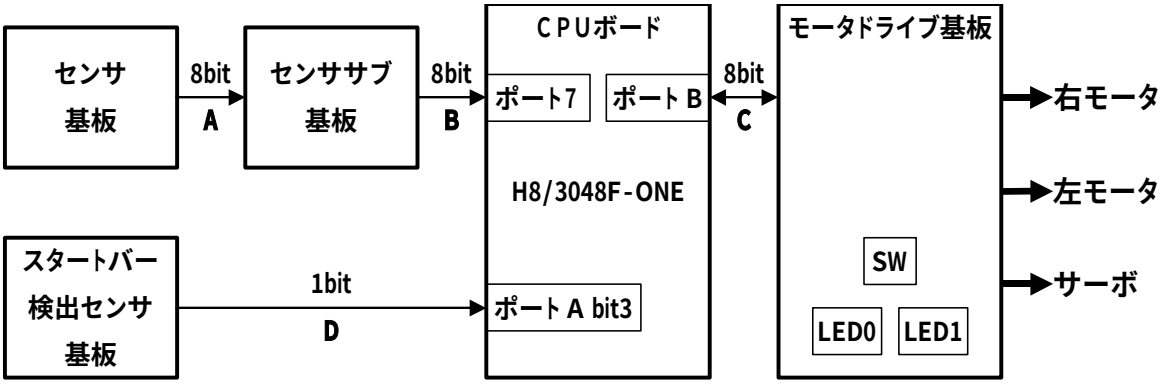
スタートバーがあるか、無いか検出する「スタートバー検出センサ基板」がキット化されています。詳しくは35ページの「**6. スタートバー検出センサ基板**」を参照して下さい。

3. マイコンカーキットの仕様

3.1 外観



マイコンカーは制御系の、CPU ボード、センサ基板、センササブ基板、モータドライブ基板、スタートバー検出センサ基板、駆動系の右モータ、左モータ、サーボで構成されています。

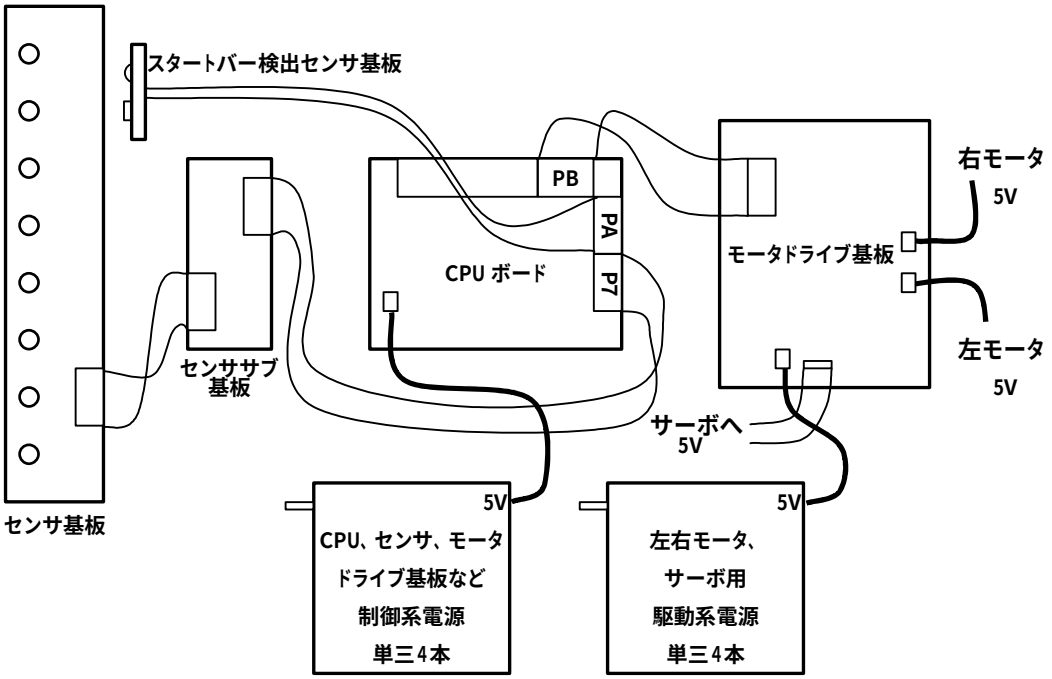


センサ基板	コースの白黒状態を読み込み、デジタル信号に変換して出力します。出力信号は白色:"0"、黒色:"ハイインピーダンス"（サブ基板で"1"にします）です。センサは、8 つ付いています。
センサ サブ基板	プルアップ抵抗で"0"と"ハイインピーダンス"の信号を、"0"と"1"信号に変換します。その後、NOT 回路(74HC04)で信号を反転させます。 結果、センササブ基板の出力は、白色"1"、黒色"0"となります。この信号を、CPU ボードへ出力します。
スタートバー 検出センサ 基板	JMCR2007(2006 年度)からスタート時、スタートバーが開くことをマイコンカーが自動検出してスタートする方式となりました。そのスタートバーが開くことを検出するセンサです。スタートバーは白色なので、白色反応("0")＝スタートバーあり、黒色反応("1")＝スタートバーなしと判断できます。
CPU ボード	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、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。 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。 ポート A のビット 3 には、スタートバー検出センサを接続します。その他のビットは使っていません。そのため、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。
モータドライブ 基板	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。 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 つ付いており、デバッグに使用できます。
電池	・制御系（CPU）電源　　…単三 2 次電池 4 本(1.2V×4 本＝4.8V)を使用 ・駆動系（モータ・サーボ）電源…単三 2 次電池 4 本か 単三アルカリ電池 4 本(1.5V×4 本＝6.0V)を使用 ※CPU 電源は、5.0V±10%の電圧にしてください。

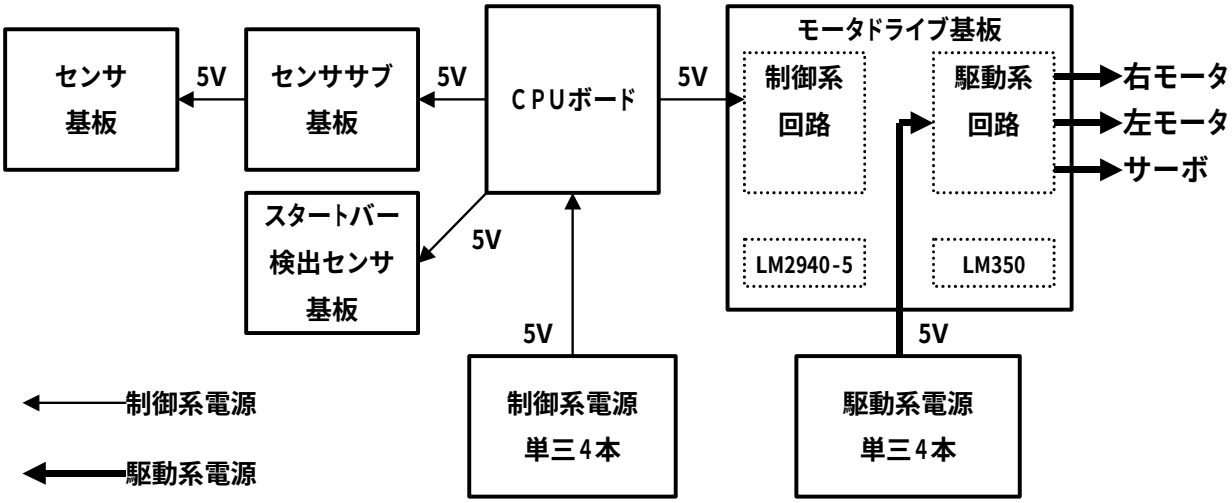
3.2 標準キットの電源構成

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。

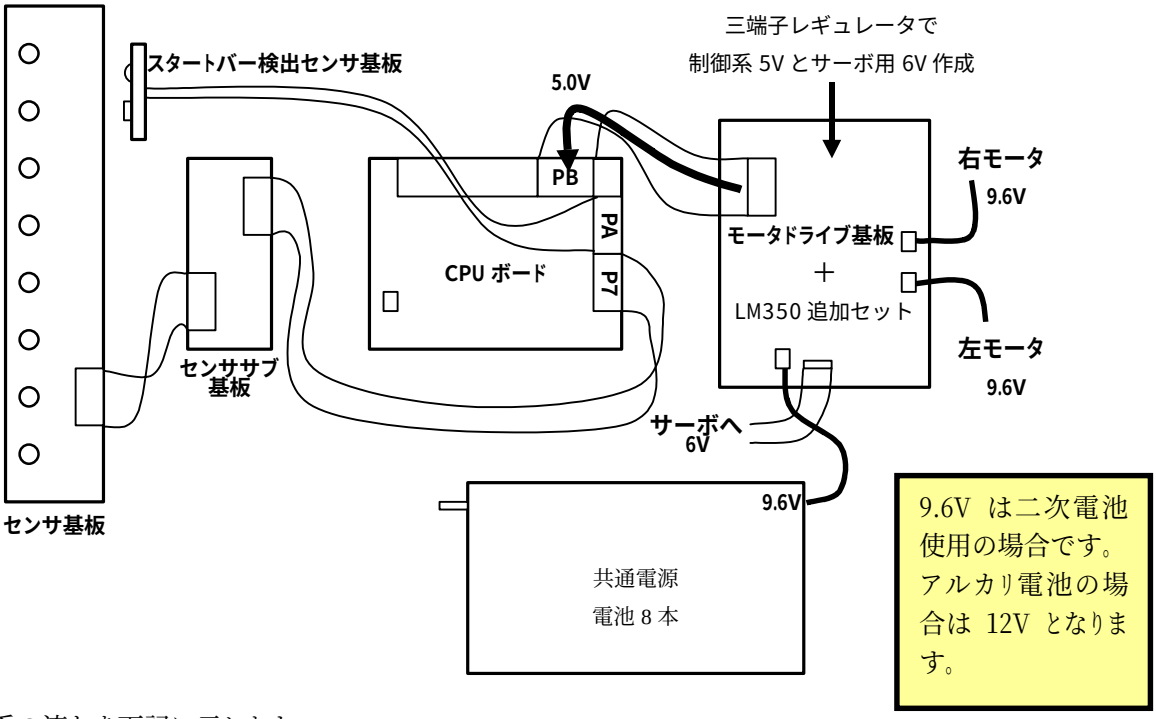


3.3 駆動系電圧を上げた電源構成

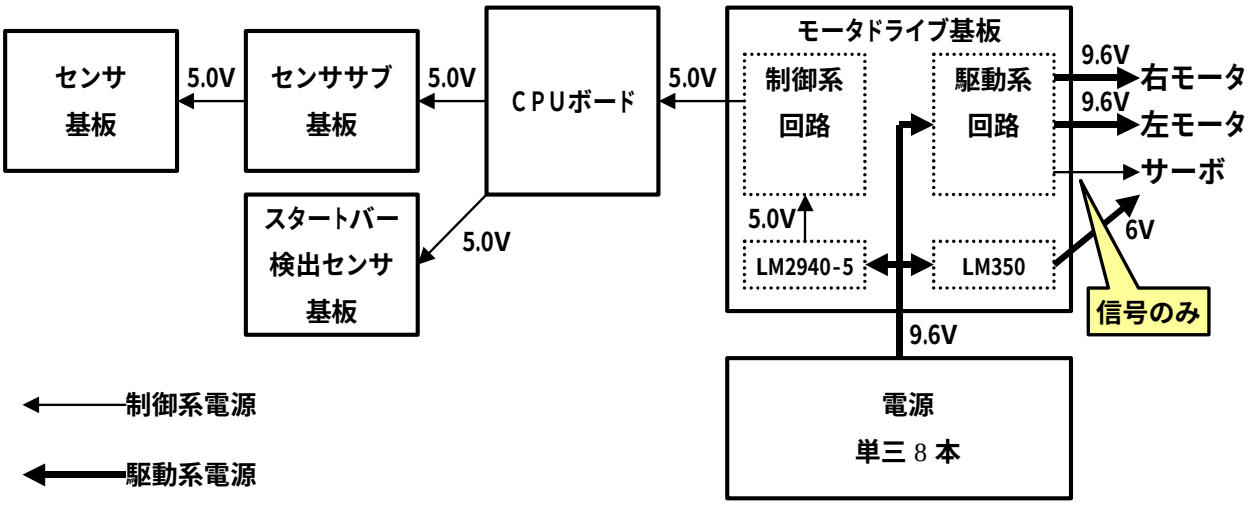
駆動系の電圧を上げれば（電池を増やせば）モータの回転数を上げることが可能です。モータ電源用に 6 本の電池を使えば 7.2V、8 本なら 9.6V となります。しかし、電池の使用本数は 8 本以内と決まっています。そこで、電池を制御系、駆動系共通にします。そのとき、モータは 9.6V でも問題ありませんが、CPU の動作保証電圧は 4.5～5.5V なので 5.5V を超えた電圧をかけると壊れてしまいます。サーボも同様に 6V 以上の電圧をかけられません。そのため、三端子レギュレータを取り付け CPU やサーボの電圧を定格にします。

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。

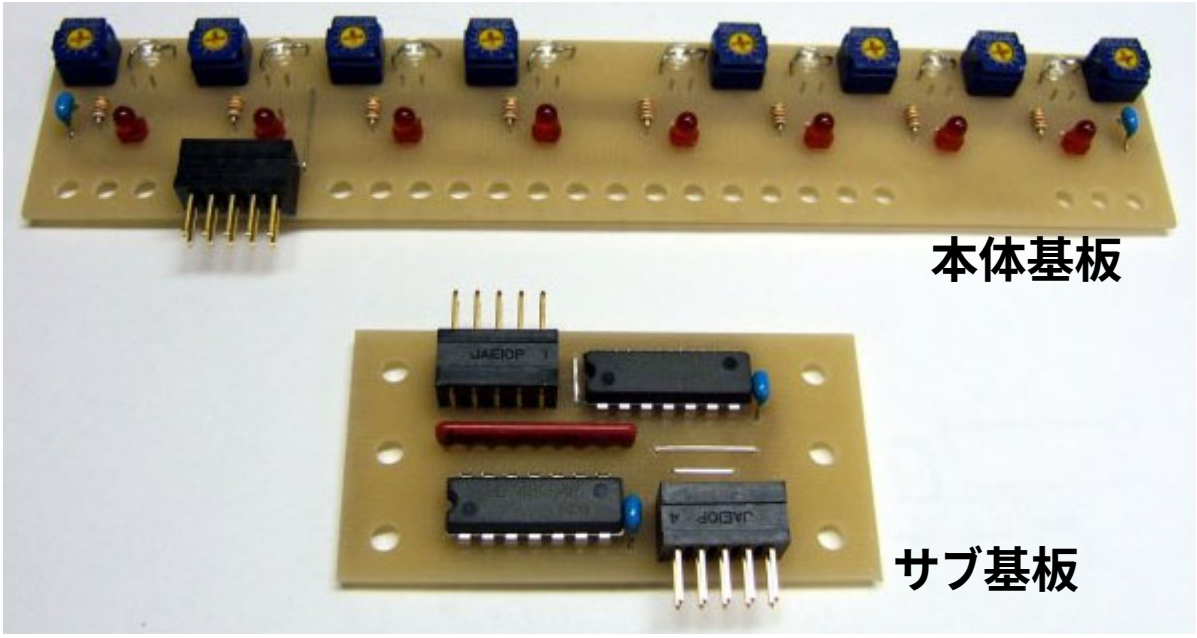


※キットの電池ボックスは、バネの押しが弱いため、マイコンカーの加速によって、電池の端子が電池ボックスから離れてしまい電源が切れて、マイコンがリセットすることがあります。押しつけの強い電池ボックスを使うか、電池を直接半田付けしてこのようなことが起こらないようにしてください。

4. センサ基板

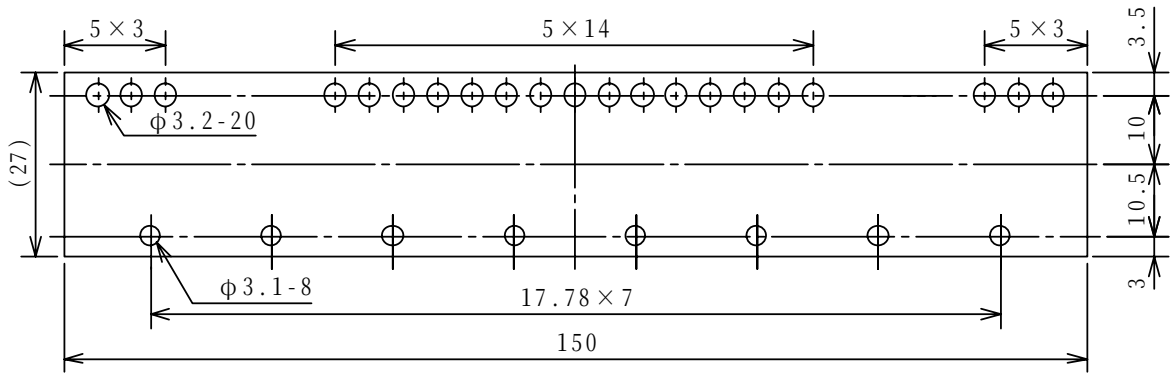
4.1 仕様

内容	本体基板	サブ基板
機能	センサ下の白、黒情報を"0"、"1"（ハイインピーダンス）のデジタル信号に変換する	本体基板の信号を入力し、74HC04 (NOT 回路)で信号を反転、CPU ボードへ信号を出力する
動作電圧	DC5.0V±5%	DC5.0V±5%
寸法	最大 W150×D33×H10mm (実測)	最大 W60×D37×H10mm (実測)
コネクタ	センサ信号出力用10ピンコネクタ×1	本体基板からの信号入力用10ピンコネクタ×1 CPU ボードへの信号出力用10ピンコネクタ×1
信号	・センサ下が黒なら、ハイインピーダンス出力で LED は消灯 ・センサ下が白なら、0V 出力("0")で LED は点灯	・基板内のプルアップ抵抗でハイインピーダンス信号を 5V("1")に変換 ・入力された信号を反転して出力する
重量	約 20g（完成品の実測） ※リード線や半田の量で変わります	約 10g（完成品の実測） ※リード線や半田の量で変わります

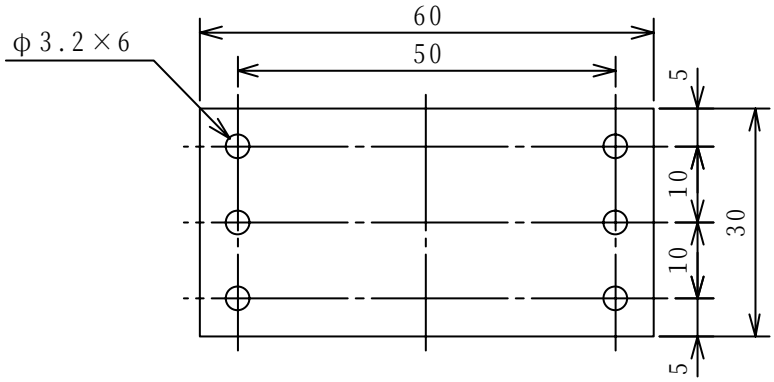


4.2 基板寸法

4.2.1 本体基板



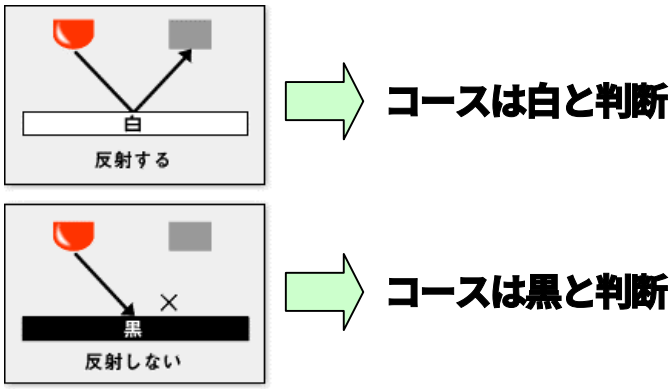
4.2.2 サブ基板



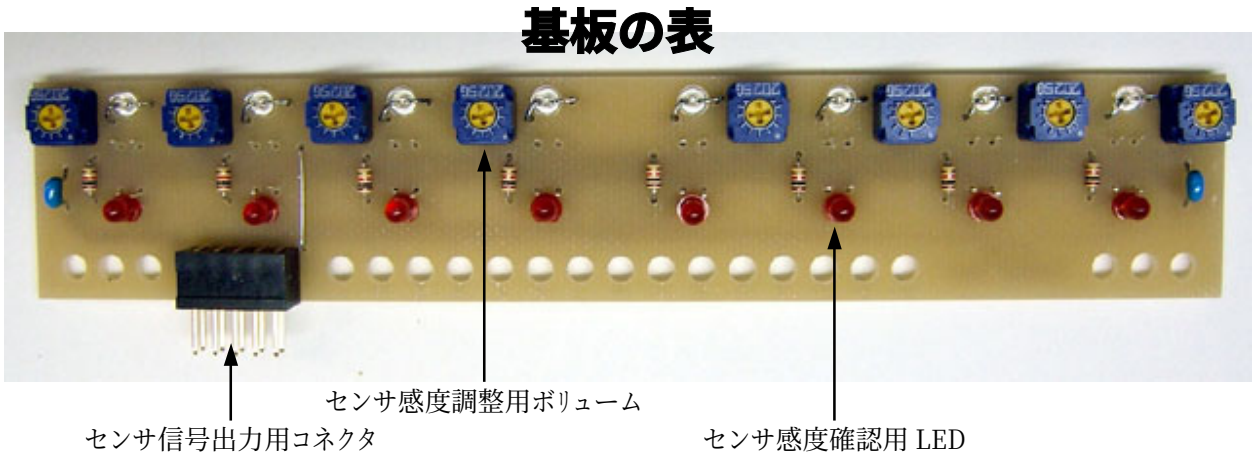
4.3 機能

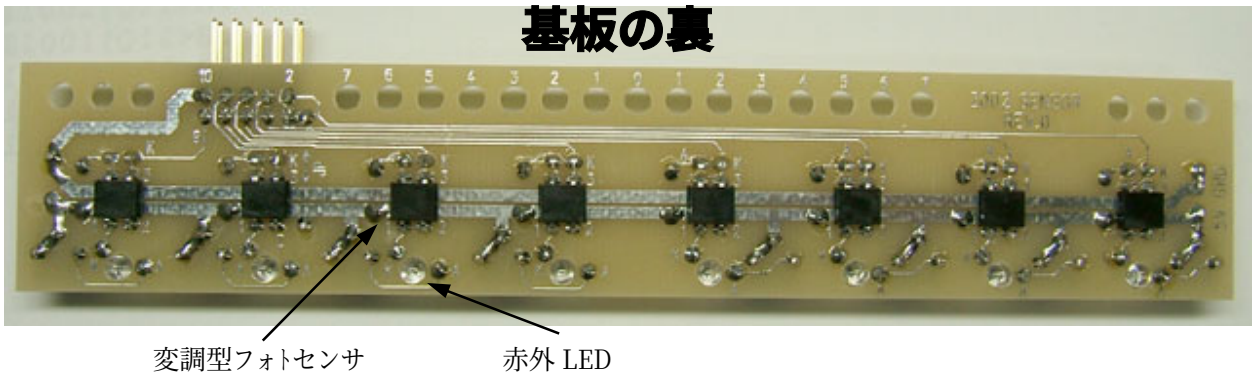
4.3.1 本体基板

本体基板には、光を出す素子と受ける素子が8組付いています。「白は光を反射する」、「黒は光を吸収する」ことを利用して、光を出す素子から出した光をマイコンカーのコースに当てます。その光が、光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。



光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。**標準のプログラムでは、"白"と判断させると良いようになっています**。

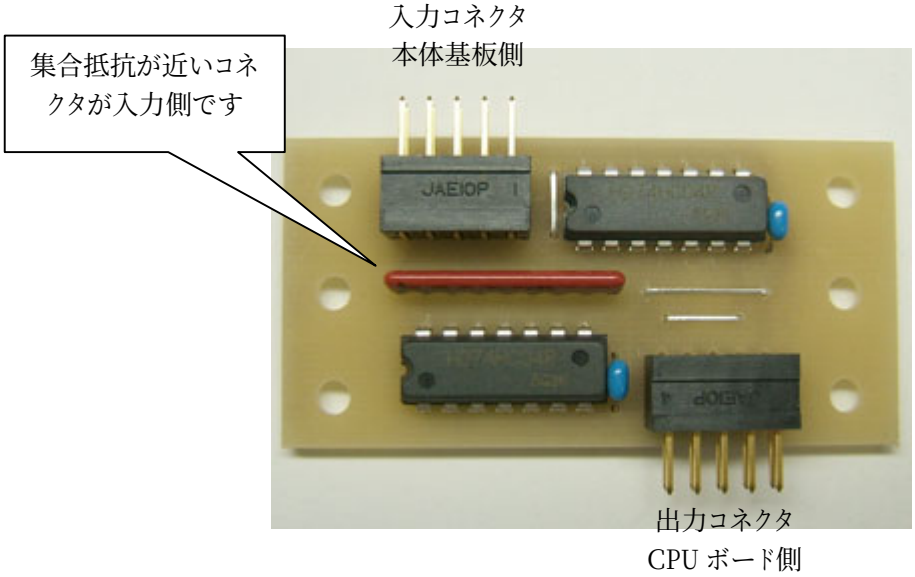




内容	詳細
赤外 LED	TLN119 という素子を使用しています。この素子から赤外線の光を出します。赤外線なので人間の目には見えません。8 個あります。
変調型フォトセンサ	浜松フォトニクス（株）の S7136 という素子を使用しています。赤外 LED が出した光をこの素子で受けます。光が受信できればコースは白、できなければコースは黒と判断します。
センサ信号 出力用コネクタ	センサの下部が白なら"0"(0V)、黒ならハイインピーダンス(プルアップ抵抗を付けることにより"1"になる)の信号がこのコネクタから出力されます。
センサ感度 調整用ボリューム	赤外 LED から出力する光の量を調整します。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にこの LED を確認しながら調整します。

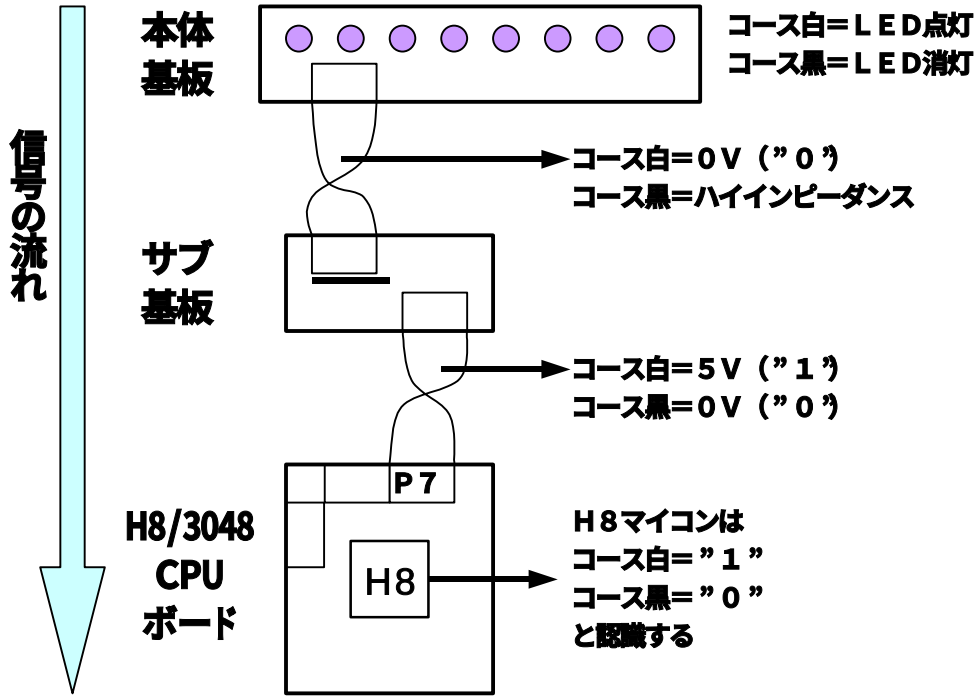
4.3.2 サブ基板

サブ基板は、
・センサ本体基板からの信号を入力コネクタで受けます。
・センサ本体基板からの信号は、黒色はハイインピーダンス（何も繋がっていないのと同じ状態）、白色は"0"出力なので、サブ基板のプルアップ抵抗で、黒色は"1"、白色は"0"に変換します。
・NOT 回路(74HC04)にて論理反転をします。
・出力コネクタより信号を出力します。出力先は、CPU ボードです。



4.4 信号の流れ

本体基板、サブ基板、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、センサが左から「白黒黒黒白白黒黒」なら、

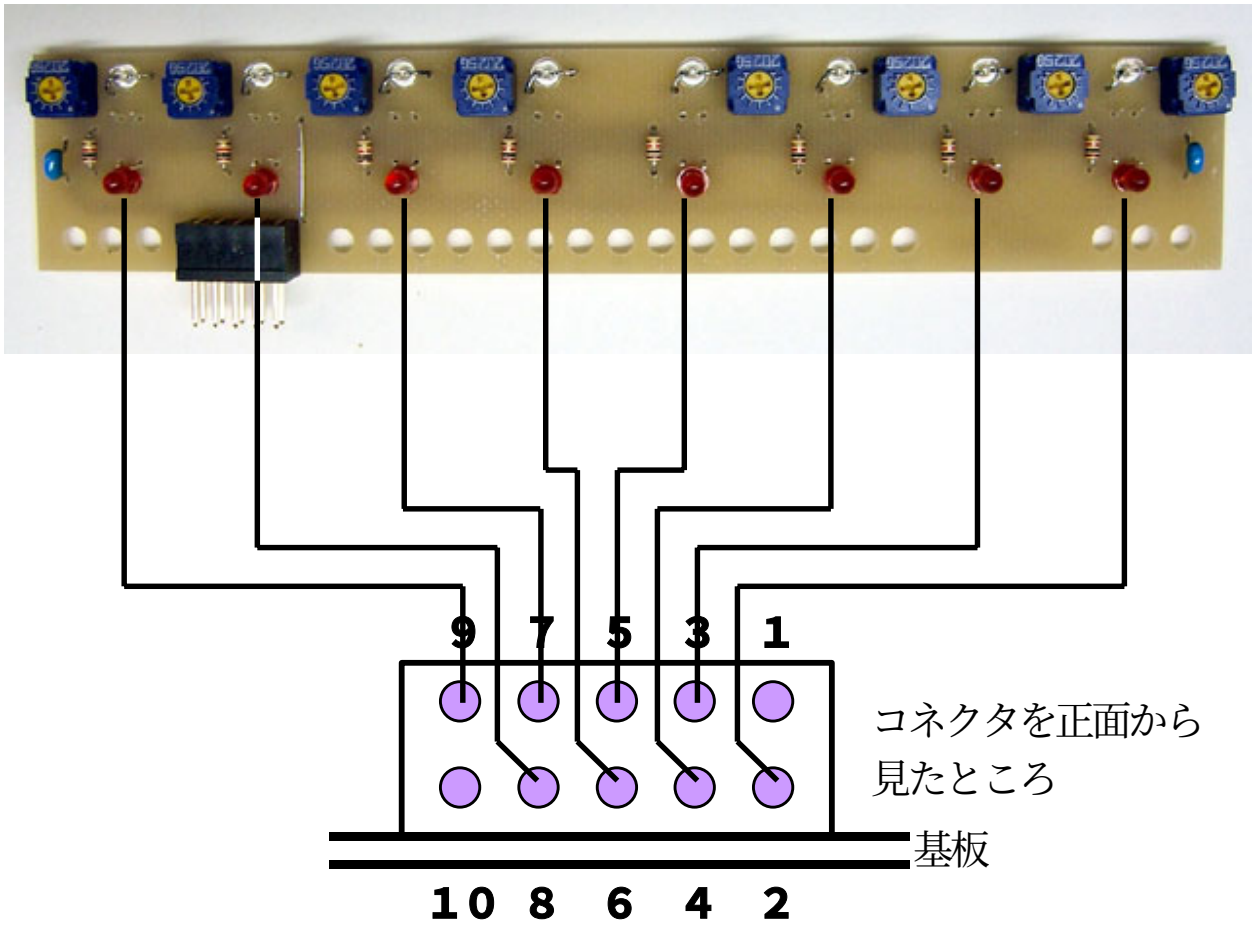
```
unsigned char c;
…
c = P7DR;
```

とすると、変数 c には
c ＝ $(0011\ 0001)_2$ ＝ 0x31
が代入されます。

※「白黒黒黒白白黒黒」はデジタル値に変換すると「10001100」ですが、マイコンがポートからデータを読み込むと左右が入れ代わるため、「00110001」となります（配線の都合上）。左右が入れ代わると非常に分かりづらいです。そのため、**マイコンカーの場合は、プログラムで左右を入れ替えて「10001100」と変換します**。詳しくは、「**9.12 センサ状態読み込み：sensor_inp 関数**」を参照してください。

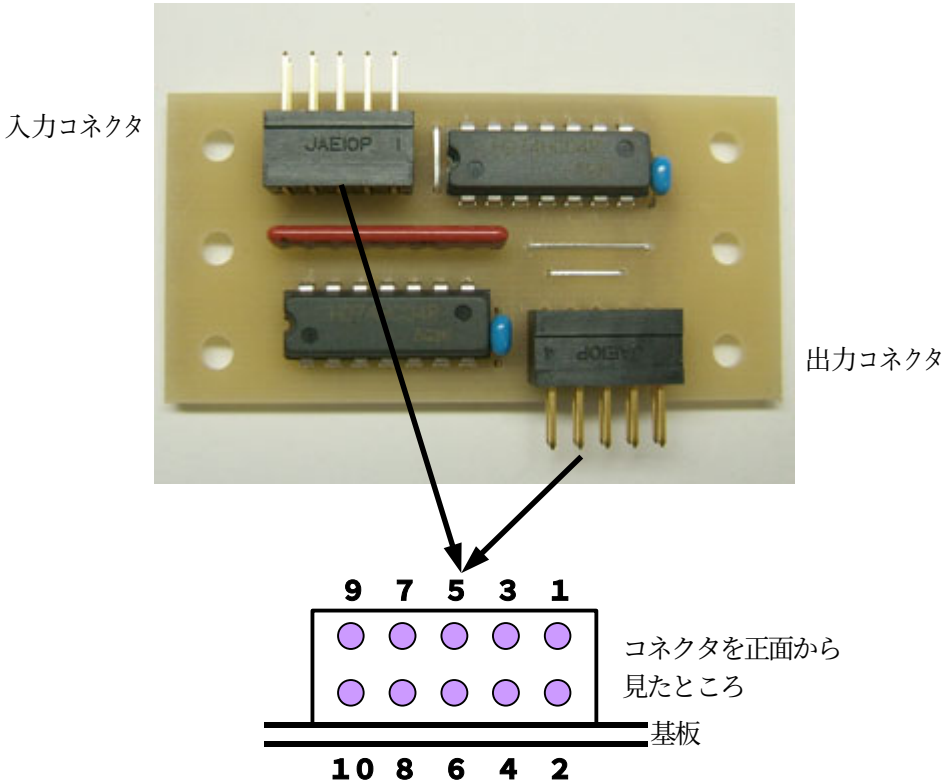
4.5 コネクタ

4.5.1 本体基板の 10 ピンコネクタ



番号	方向	詳細	“0” (0V)	“1” (ハイインピーダンス)
1	―	+5V		
2	OUT	右から 1 番目のセンサ信号出力	白色	黒色
3	OUT	右から 2 番目のセンサ信号出力	白色	黒色
4	OUT	右から 3 番目のセンサ信号出力	白色	黒色
5	OUT	右から 4 番目のセンサ信号出力	白色	黒色
6	OUT	右から 5 番目のセンサ信号出力	白色	黒色
7	OUT	右から 6 番目のセンサ信号出力	白色	黒色
8	OUT	右から 7 番目のセンサ信号出力	白色	黒色
9	OUT	右から 8 番目のセンサ信号出力	白色	黒色
10	―	GND		

4.5.2 サブ基板のコネクタ



入力コネクタ

入力側番号	方向	入力信号
1	+5V	センサ本体基板からの信号
2	IN7	
3	IN6	
4	IN5	
5	IN4	
6	IN3	
7	IN2	
8	IN1	
9	IN0	
10	GND	

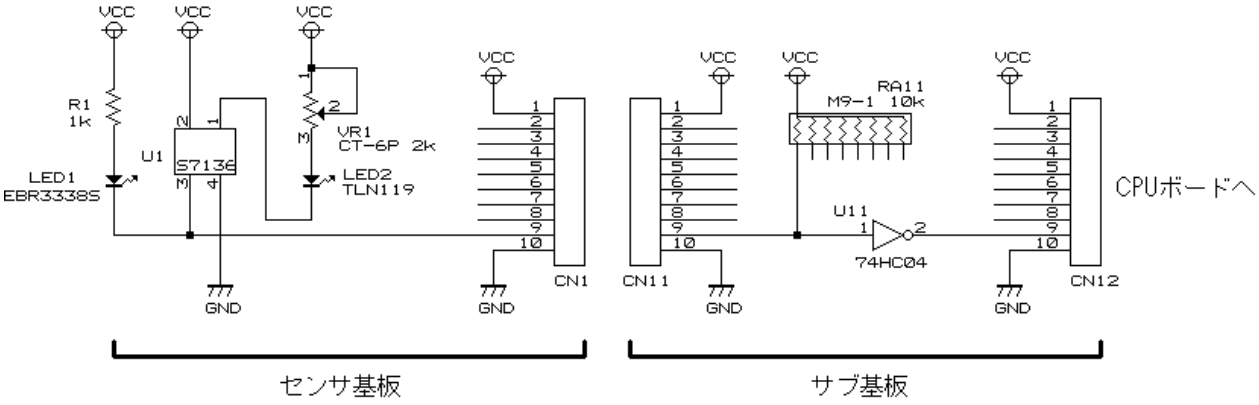
⇒

出力コネクタ

出力側番号	方向	出力信号	CPU ボード接続先
1	+5V		+5V
2	OUT7	入力信号の反転信号	P77
3	OUT6	入力信号の反転信号	P76
4	OUT5	入力信号の反転信号	P75
5	OUT4	入力信号の反転信号	P74
6	OUT3	入力信号の反転信号	P73
7	OUT2	入力信号の反転信号	P72
8	OUT1	入力信号の反転信号	P71
9	OUT0	入力信号の反転信号	P70
10	GND		GND

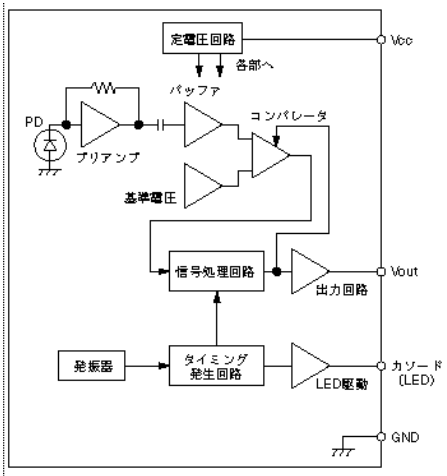
※P70 とは、H8 マイコンのポート 7 ビット 0 という意味です。P71～P77 も同様です。

4.6 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 の出力調整はボリューム 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 U1 に届かなければ（コースは黒）"1"が出力されます（詳しくは次の6番参照）。LED のアノード側が＋、カソード側が＋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。通常デジタルは"0"＝0V、"1"＝5V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。そのため、サブ基板のプルアップ抵抗(RA11)でフォトセンサの出力信号がオープンのときは"1"になるようにしています。
7. U1 の 3 ピンの出力は、コースが白で LED1 が点灯したとき”0”、コースが黒で LED1 が消灯したとき”1”出力となります。しかし、感覚的に白＝”1”、黒＝”0”が分かりやすいので NOT 回路(U11)にて反転させています。CPU に取り込むときは、**「白＝”1”、黒＝”0”」**となります。

※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、入力光の微少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。

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 Vol.3 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…	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
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 (2006.05 現在)

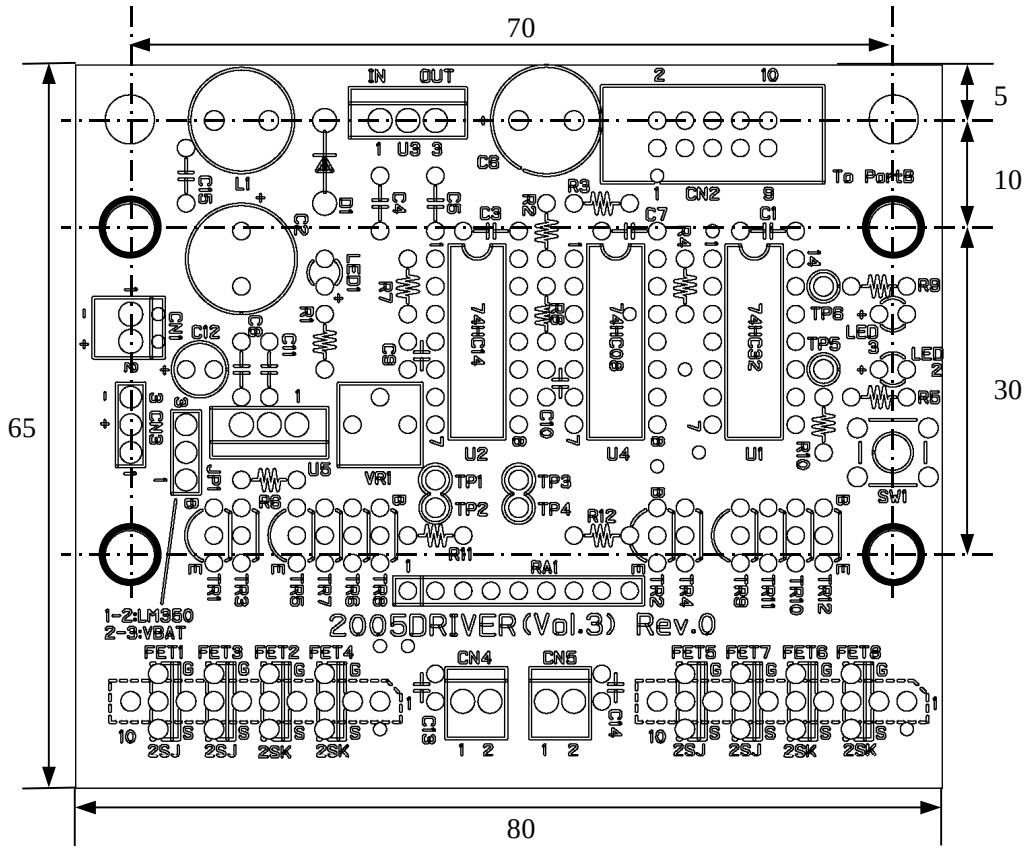
ドライブ基板1ではリセット同期 PWM モードというモードでモータ、サーボを制御していました。これはプログラムが比較的簡単なのですが、右モータ、左モータ、サーボに加える PWM 周期を同じにしかできないという制限がありました。サーボには 16[ms]の周期を加えること、と規格で決まっておりモータに加える PWM も 16[ms]となります。しかし、モータ制御に周期 16[ms]では間隔が長すぎ、モータがガクガクした動きとなってしまいました。

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。ただし、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。

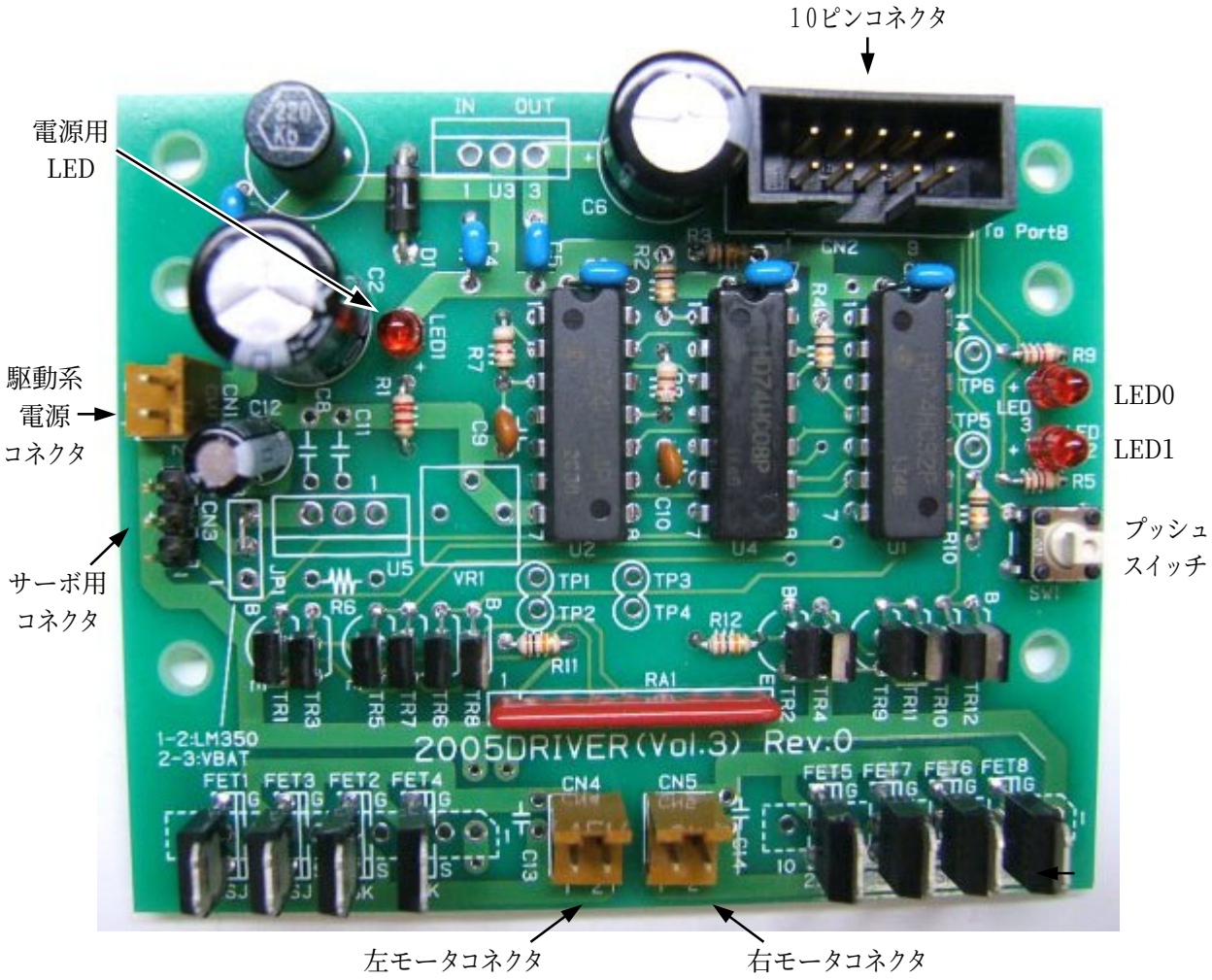
5.2 寸法

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。



5.3 機能

ドライブ基板3は、部品を実装すると下図のようになります。



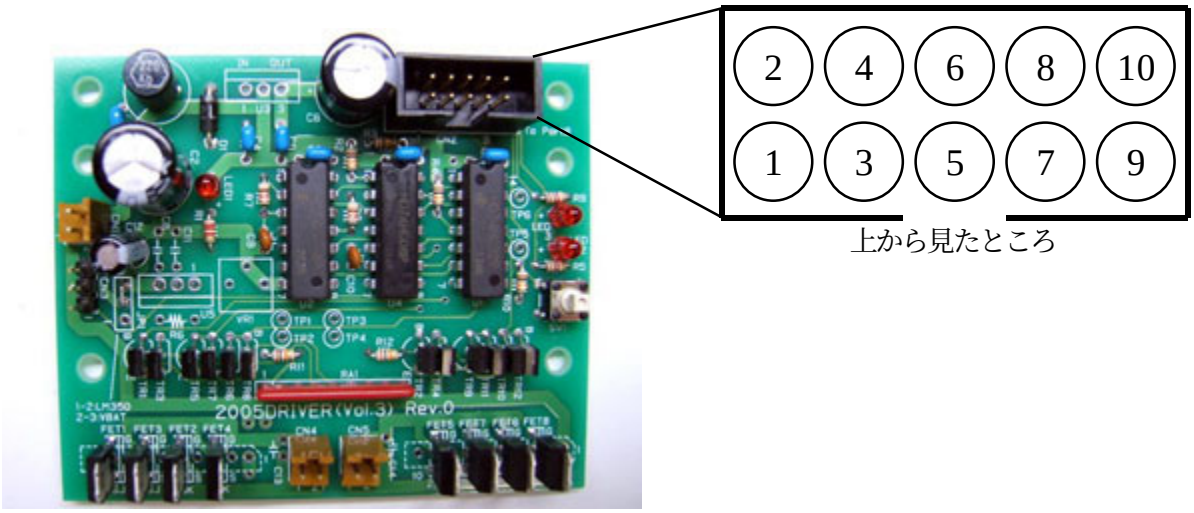
10 ピンコネクタ	フラットケーブルで CPU ボードと接続します。ポート B(J3)のコネクタに接続します。
駆動系 電源コネクタ	モータとサーボに供給する電源です。74HC08、74HC14、74HC32 などの制御系部品は、10 ピンコネクタから供給される 5 V で動作します。標準キットでは入力電圧 6V までに対応していますが、それ以上の電圧にするときは、**サーボに加える電圧を 6 V一定にする必要があります**。LM350追加セットの部品を追加すると、サーボ電圧を一定にすることができます。
右モータコネクタ	右モータと接続します。
左モータコネクタ	左モータと接続します。
サーボ用コネクタ	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。

5.4 コネクタ

5.4.1 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。

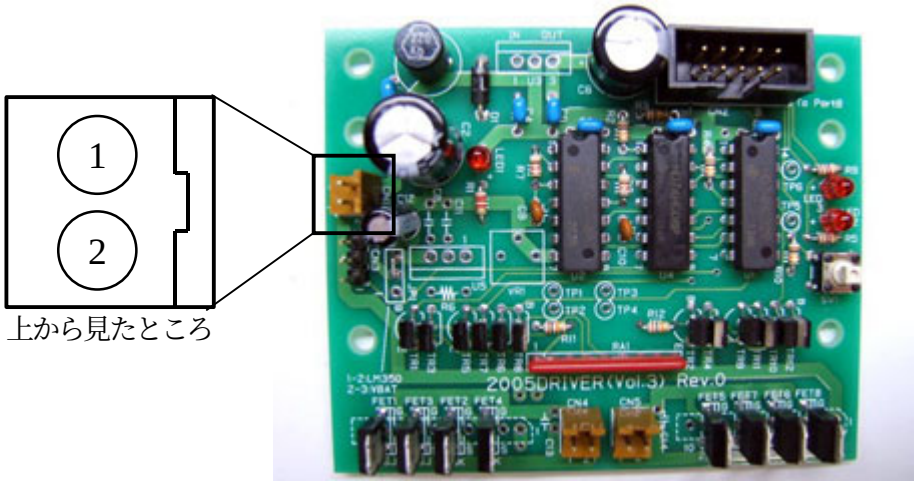


ピン番	信号、方向※	詳細	“0”	“1”	備考
1	―	+5V			
2	基板←PB7	LED1	点灯	消灯	
3	基板←PB6	LED0	点灯	消灯	
4	基板←PB5	サーボ信号	PWM 信号		PWM 信号 ITU4_BRB でデューティ比設定
5	基板←PB4	右モータ PWM	停止	動作	PWM 信号 ITU4_BRA でデューティ比設定
6	基板←PB3	右モータ回転方向	正転	逆転	
7	基板←PB2	左モータ回転方向	正転	逆転	
8	基板←PB1	左モータ PWM	停止	動作	PWM 信号 ITU3_BRB でデューティ比設定
9	基板→PB0	プッシュスイッチ	押された	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

※PB7 とは、H8 マイコンのポート B, ビット7の意味です。
「基板→PB〇」 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←PB〇」 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。

5.4.2 駆動系電源コネクタ

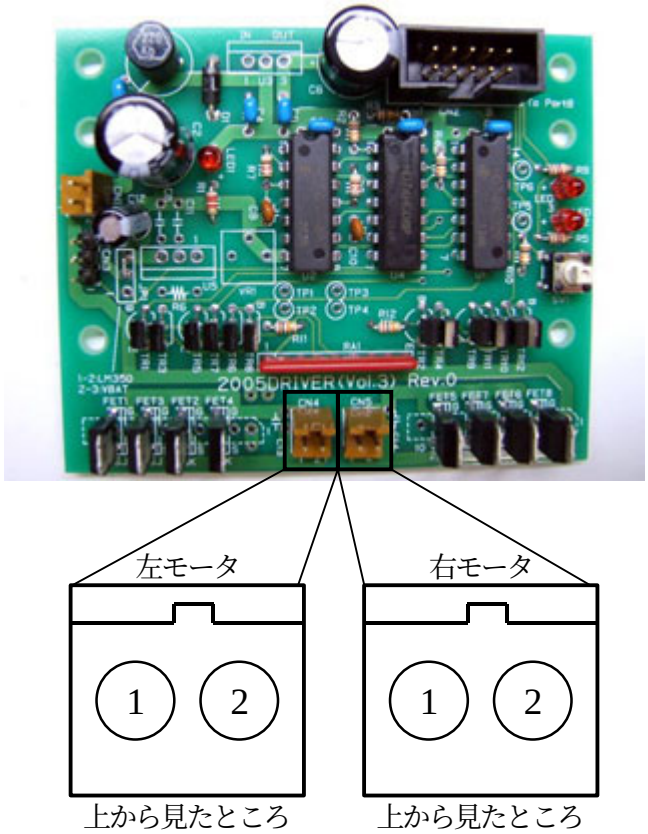
モータ、サーボ用の電源と接続します。



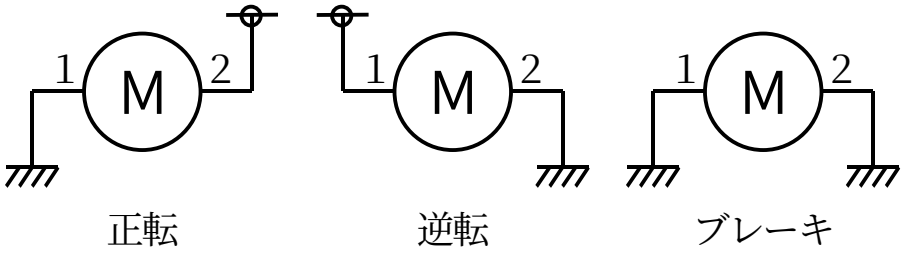
番号	方向	詳細
1	－	GND
2	IN	電源入力 5～15V

5.4.3 モータコネクタ

モータと接続します。

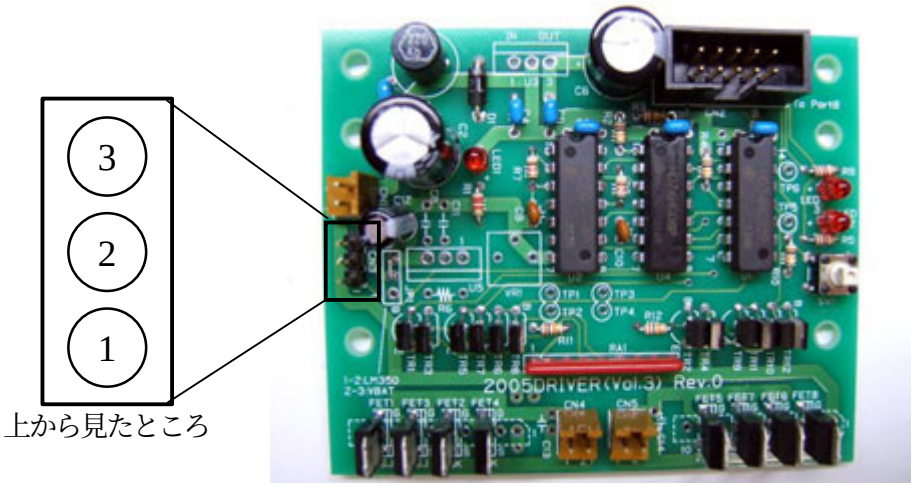


番号	方向	正転	逆転	ブレーキ
1	OUT	0V	電源電圧	0V
2	OUT	電源電圧	0V	0V



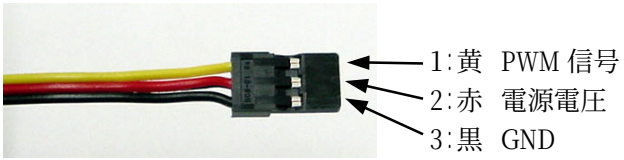
5.4.4 サーボコネクタ

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



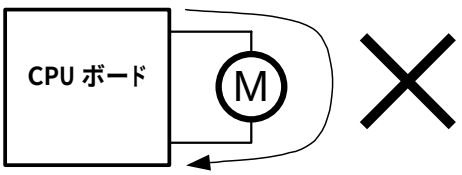
※2 ピンの電源は、JP1 で切り替えます。

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

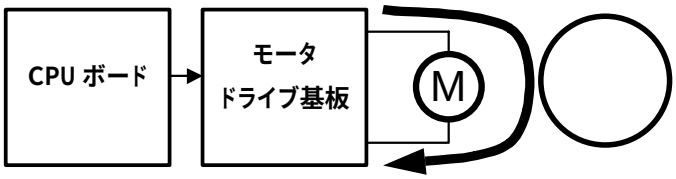
5.5 モータ制御

5.5.1 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。



信号が弱いため、モータは回らない



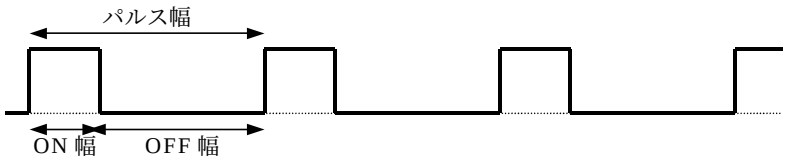
大電流に変換、モータ駆動！

5.5.2 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。

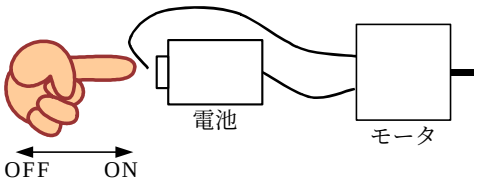
ボリュームを使えば電圧を落とすことができます。しかし、モータへは大電流が流れるため、非常に大きな抵抗が必要です。また、モータに加えなかった分は、抵抗の熱となってしまいます。

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。

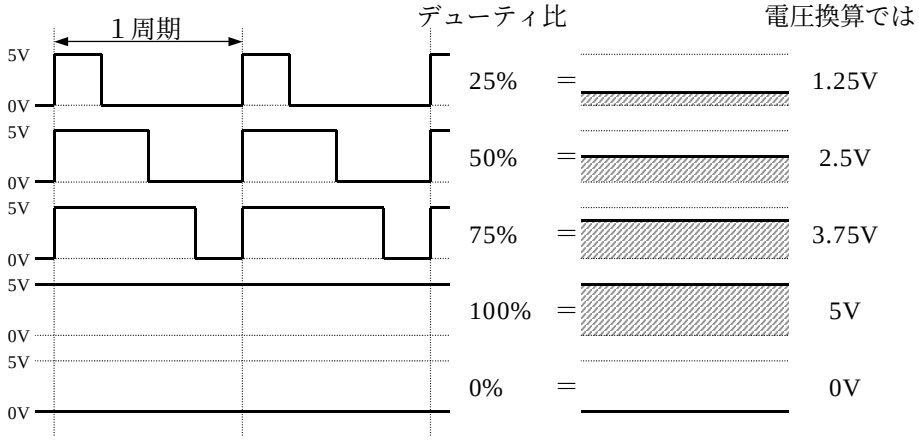
「PWM」と聞くと、何か難しく感じてしまいますが、下記のように手でモータと電池の線を「繋ぐ」、「離す」の繰り返し、それも PWM と言えます。繋いでいる時間が長いとモータは速く回ります。離している時間が長いとモータは少ししか回りません。人なら「繋ぐ」、「離す」動作をコンマ数秒でしか行えませんがマイコンなら数ミリ秒で行えます。



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×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。

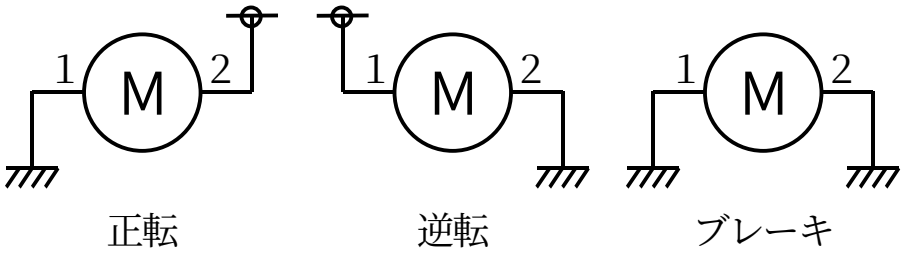


なぜ電圧制御ではなくパルス幅制御でモータのスピードを制御するのでしょうか。CPU は”0”か”1”かのデジタル値の取り扱いは大変得意ですが、何 V というアナログ的な値は不得意です。そのため、"0"と"1"の幅を変えて、あたかも電圧制御しているように振る舞います。これが PWM 制御です。

5.5.3 正転、逆転、ブレーキの原理

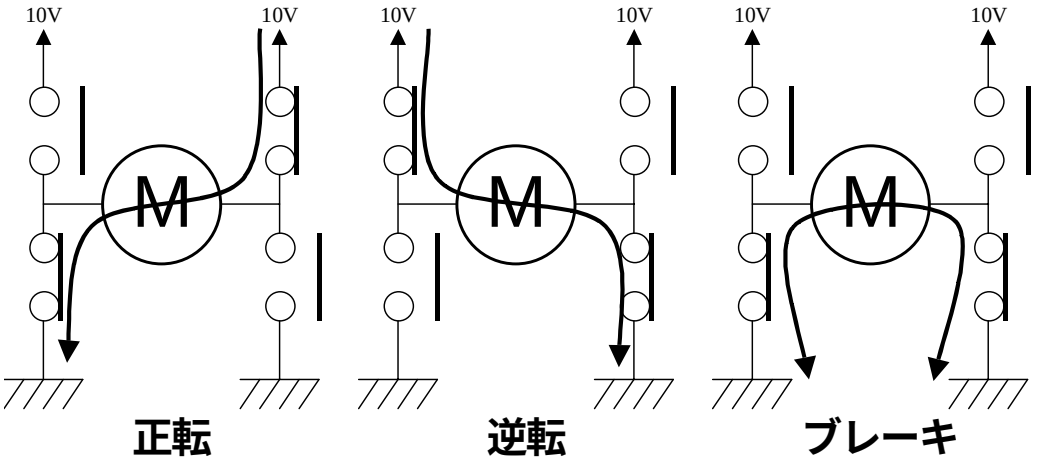
モータドライブ基板(Vol.3)では、モータを「正転、逆転、ブレーキ」制御することができます。これは、モータの端子に加える電圧を下表のように変えることにより、制御しています。

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



5.5.4 Hブリッジ回路

では、実際はどのようにするのでしょうか。下図のように、モータを中心としてH型に4つのスイッチを付けます。この4つのスイッチをそれぞれ ON／OFF することにより、正転、逆転、ブレーキ制御を行います。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。

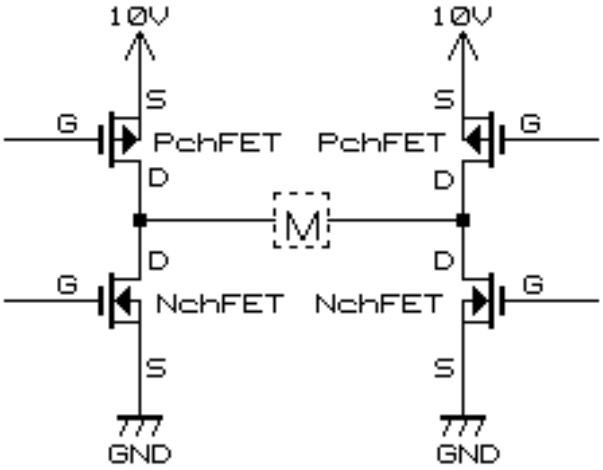


5.5.5 Hブリッジ回路のスイッチをFETにする

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。

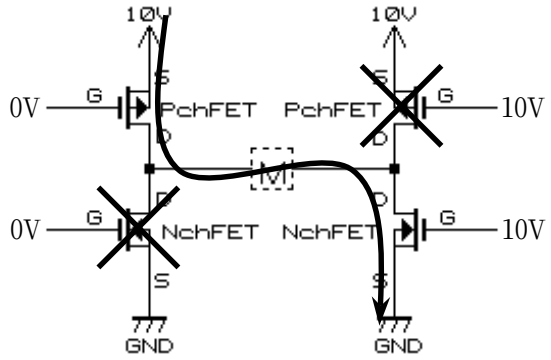
P チャネル FET は、V_G (ゲート電圧) ＜V_S (ソース電圧) のとき、D-S (ドレイン－ソース) 間に電流が流れます。

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。

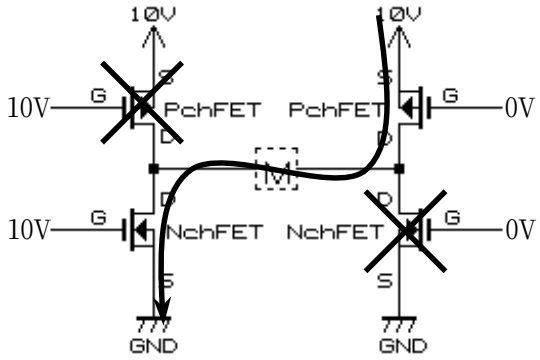


これら 4 つの FET のゲートに加える電圧を変えることにより、正転、逆転、ブレーキの動作を行います。

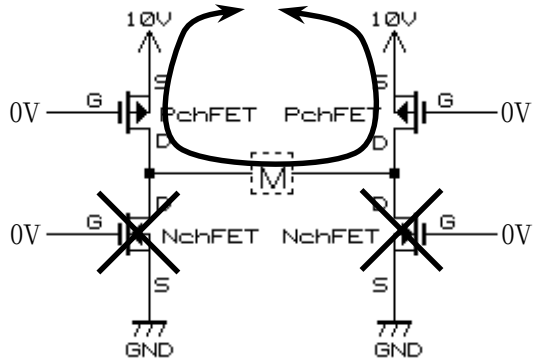
逆転



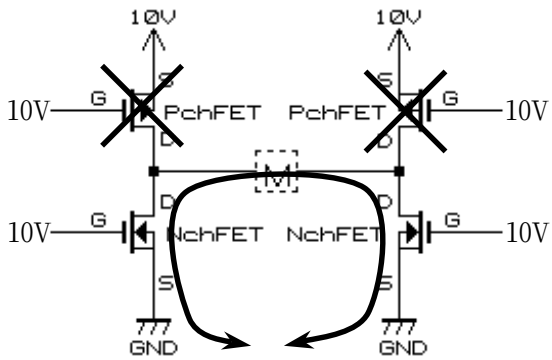
正転



ブレーキ1

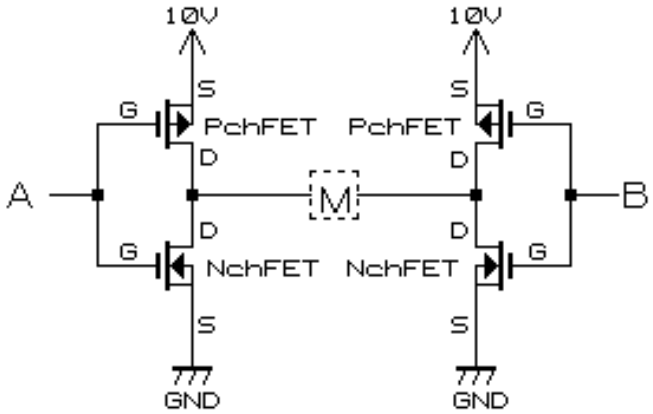


ブレーキ2



注意点は、絶対に左側もしくは右側の FET を同時に ON させてはいけないと言うことです。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

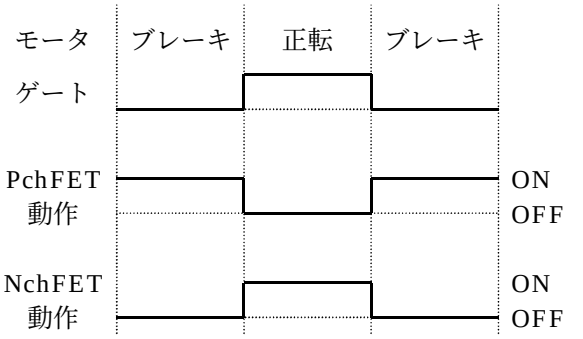
※G（ゲート）端子にはモータ用の電源電圧が 10V であったとすれば、その電圧がそのまま加えられたり 0V が加えられたりします。"0"、"1"の制御信号とは異なるので注意しましょう。

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。

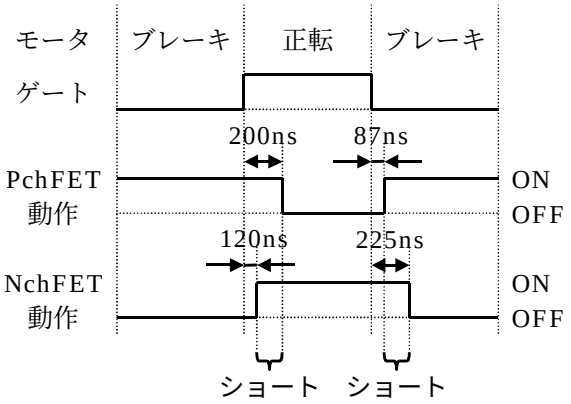
FET のゲートから信号を入力し、ドレイン･ソース間が ON／OFF するとき、事項の左図「理想的な波形」のように、P チャネル FET と N チャネル FET がすぐに反応してブレーキと正転がスムーズに切り替わりるように思えま

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 チャネル)

電気的特性

(Ta=25℃)

項目	記号	Min	Typ	Max	単位	測定条件
ドレイン・ソース破壊電圧	$V_{(BR)DSS}$	-60	—	—	V	I_D =-10mA, V_{GS} = 0
ゲート・ソース破壊電圧	$V_{(BR)GSS}$	±20	—	—	V	I_G =±100μA, V_{DS} = 0
ドレイン遮断電流	I_{DSS}	—	—	-10	μA	V_{DS} =-60V, V_{GS} = 0
ゲート遮断電流	I_{GSS}	—	—	±10	μA	V_{GS} = ±16V, V_{DS} = 0
ゲート・ソース遮断電圧	$V_{GS(off)}$	-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_{DS}=-10V 注4
ドレイン・ソースオン抵抗	$R_{DS(on)}$	—	0.08	0.10	Ω	I_D=-8A, V_{GS}=-10V 注4
ドレイン・ソースオン抵抗	$R_{DS(on)}$	—	0.11	0.16	Ω	I_D=-8A, V_{GS} =-4V 注4
入力容量	Ciss	—	850	—	pF	V_{DS} =-10V, V_{GS} = 0
出力容量	Coss	—	420	—	pF	f = 1MHz
帰還容量	Crss	—	110	—	pF	
ターン・オン遅延時間	td(on)	—	12	—	ns	V_{GS} =-10V, I_D=-8A
上昇時間	tr	—	75	—	ns	R_L =3.75Ω
ターン・オフ遅延時間	td(off)	—	125	—	ns	
下降時間	tf	—	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	trr	—	70	—	ns	I_F =-15A, V_{GS} = 0 diF/dt = 50A/μs

注) 4. パルス測定

OFF→ON は 87ns 遅れる

ON→OFF は 200ns 遅れる

2SK2869 (Nチャネル)

電気的特性

(Ta=25℃)

項目	記号	Min	Typ	Max	単位	測定条件
ドレイン・ソース破壊電圧	$V_{(BR)DSS}$	60	—	—	V	I_D = 10mA, V_{GS} = 0
ゲート・ソース破壊電圧	$V_{(BR)GSS}$	±20	—	—	V	I_G = ±100μA, V_{DS} = 0
ドレイン遮断電流	I_{DSS}	—	—	10	μA	V_{DS} = 60V, V_{GS} = 0
ゲート遮断電流	I_{GSS}	—	—	±10	μA	V_{GS} = ±16V, V_{DS} = 0
ゲート・ソース遮断電圧	$V_{GS(off)}$	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	Ω	I_D = 10A, V_{GS} = 10V*1
ドレイン・ソースオン抵抗	$R_{DS(on)}$	—	0.055	0.07	Ω	I_D = 10A, V_{GS} = 4V*1
入力容量	Ciss	—	740	—	pF	V_{DS} = 10V, V_{GS} = 0
出力容量	Coss	—	380	—	pF	f = 1MHz
帰還容量	Crss	—	140	—	pF	
ターン・オン遅延時間	td(on)	—	10	—	ns	V_{GS} = 10V, I_D= 10A
上昇時間	tr	—	110	—	ns	R_L = 3Ω
ターン・オフ遅延時間	td(off)	—	105	—	ns	
下降時間	tf	—	120	—	ns	
ダイオード順電圧	VDF	—	1.0	—	V	I_F = 20A, V_{GS} = 0
逆回復時間	trr	—	40	—	ns	I_F = 20A, V_{GS} = 0 diF/dt = 50A/μs

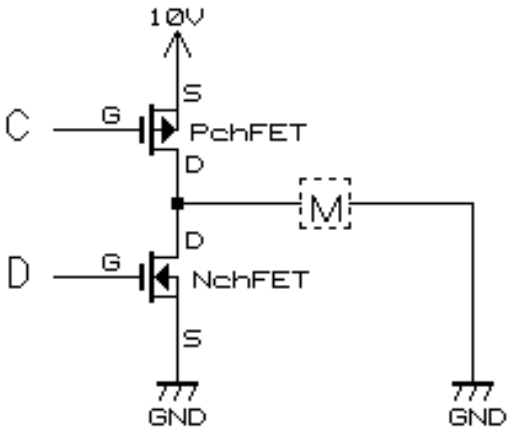
注) 1. パルス測定

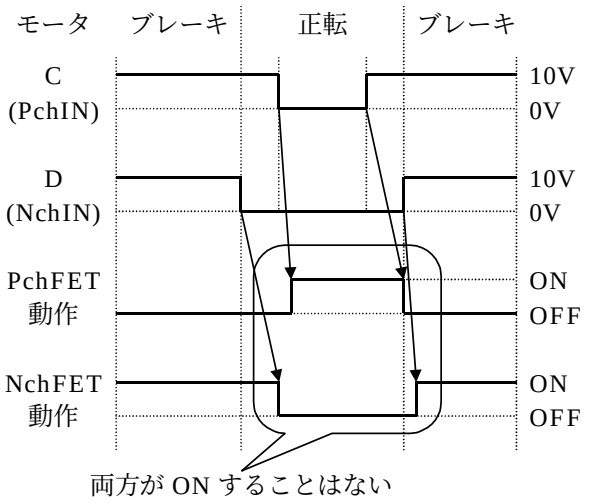
OFF→ON は 120ns 遅れる

ON→OFF は 225ns 遅れる

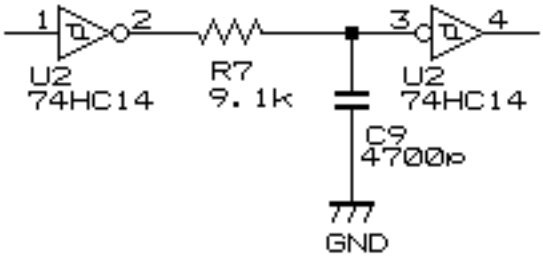
5.5.6 PチャネルとNチャネルの短絡防止回路

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

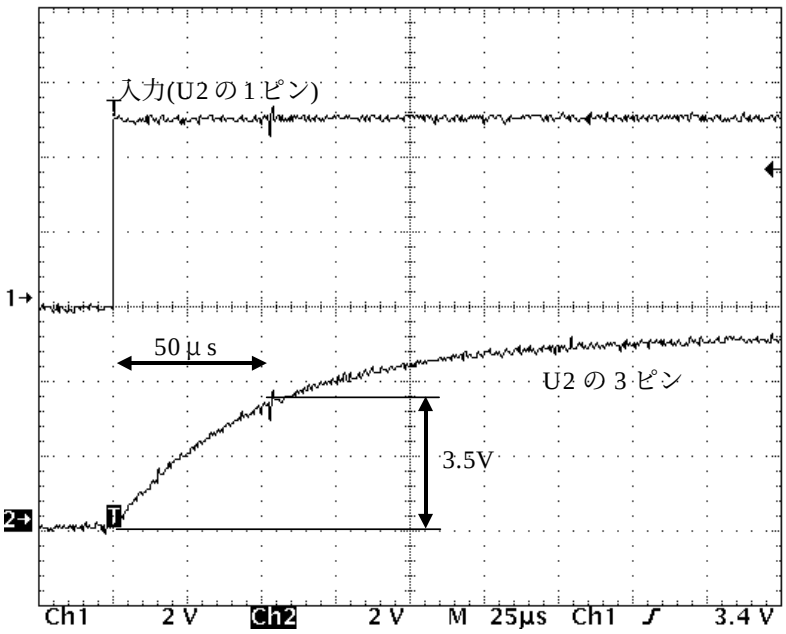




この時間をずらす部分を、積分回路で作ります。積分回路については、多数の専門書があるので、そちらを参照して下さい。

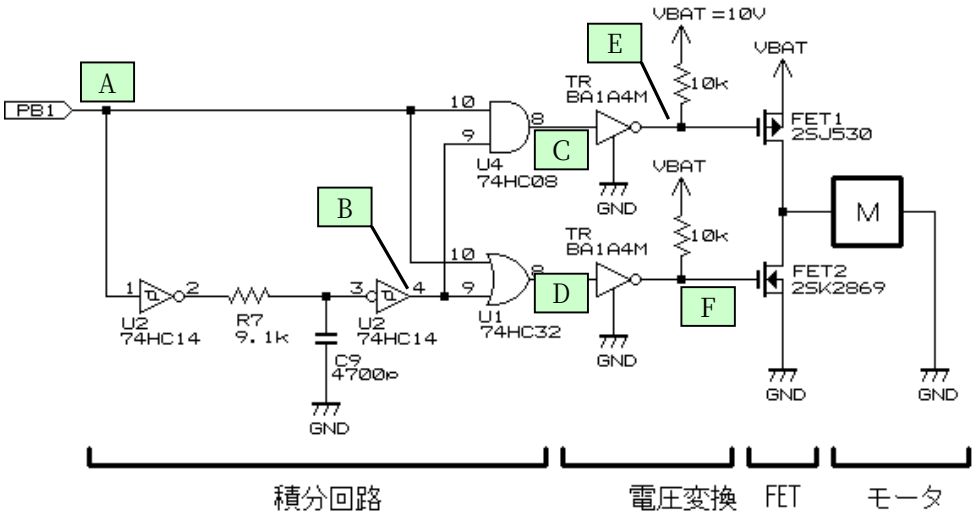


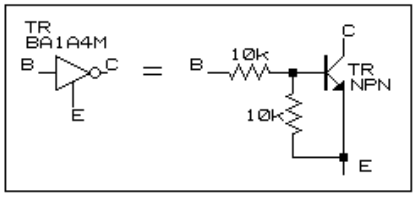
遅延時間はだいたい
時定数T＝CR [s]
で計算することができます。
今回は 9.1kΩ、4700pF なので、計算すると
$T = 9.1 \times 10^{3} \times 4700 \times 10^{-12}$
$= 42.77[\mu s]$
となります。



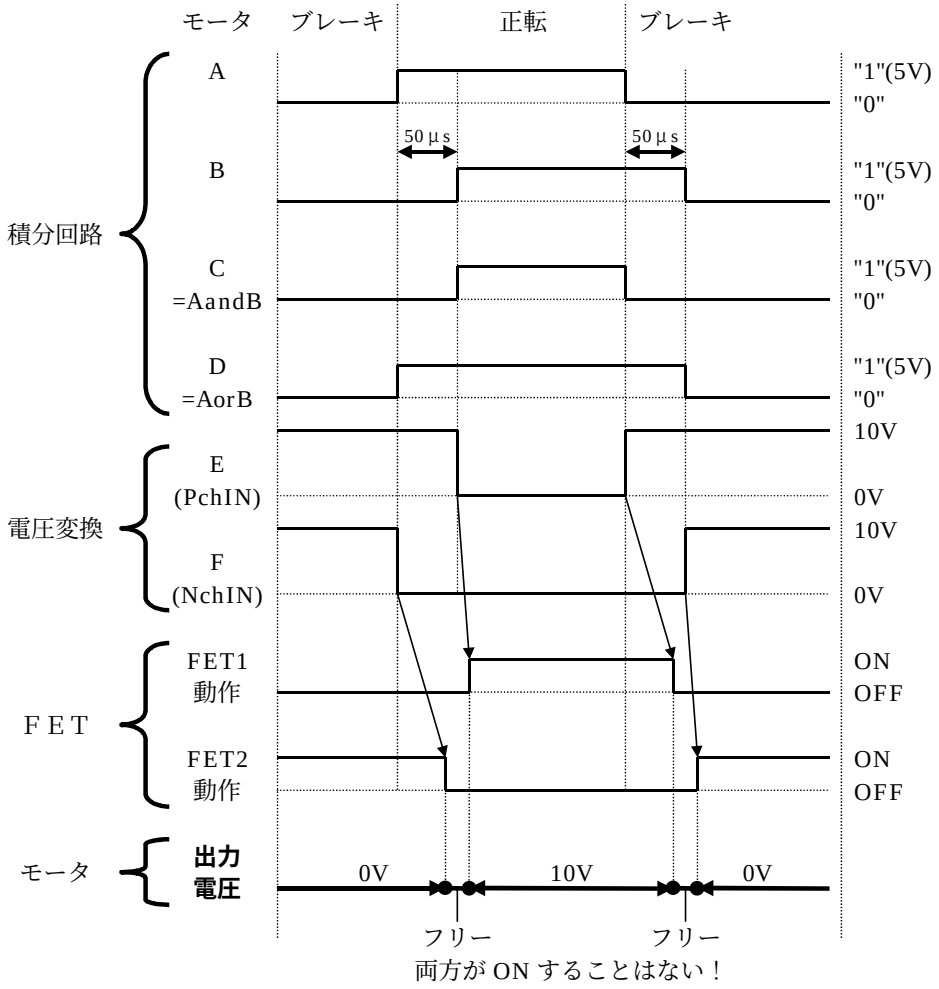
74HC シリーズは 3.5V 以上の入力電圧があると"1"とみなします。実際に波形を観測し、3.5V になるまでの時間を計ると約 50µs になりました。
先ほどの「実際の波形」の図では最高でも 225ns のずれしかありませんが、積分回路では 50µs もの遅延時間を作っています。これは、FET 以外にも、電圧変換用のデジタルトランジスタの遅延時間、FET のゲートのコンデンサ成分による遅れなどを含めたためです。

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 50μs 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 0V－5V 信号が、10V－0V 信号へと変換されます。
6. F点も同様にD点の 0V－5V 信号が、10V－0V 信号へと変換されます。
7. A点の信号を"0"→"1 にかえると、FET2 のゲートが 10V→0V となり FET2 は OFF になります。ただし、遅延時間があるため遅れて OFF になります。この時点では、FET1 も FET2 も OFF 状態のため、モータはフリー状態となります。
8. A点の信号を変えてから 50μs 後、今度は FET1 のゲートが 0V→10V となり ON します。10V がモータに加えられ正転します。

(b) 正転→ブレーキに変えるとき

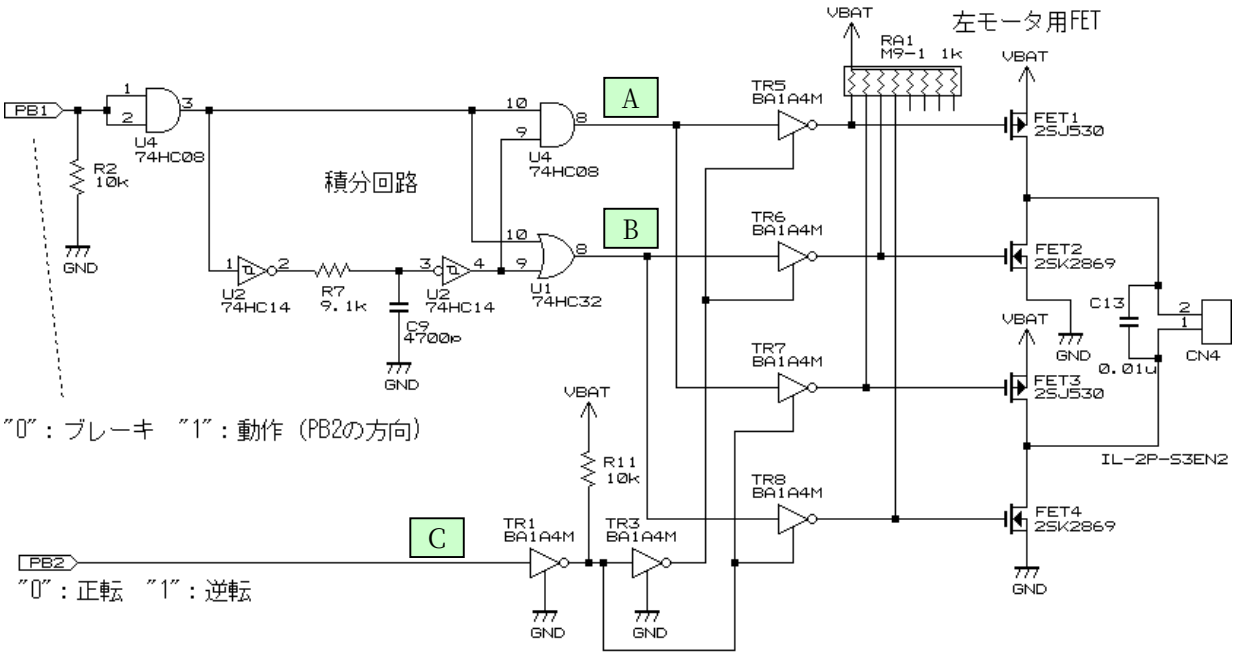
1. A点の信号を"1"→"0"にかえると、FET1 のゲートが 10V→0V となり FET1 は OFF になります。ただし、遅延時間があるため遅れて OFF になります。この時点では、FET1 も FET2 も OFF 状態のため、モータはフリー状態となります。
2. A点の信号を変えてから 50μs 後、今度は FET2 のゲートが 0V→10V となり ON します。0V がモータに加えられ、両端子 0V なのでブレーキ動作になります。

このように、動作を切り替えるときはいったん、両 FET とも 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。

5.5.7 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

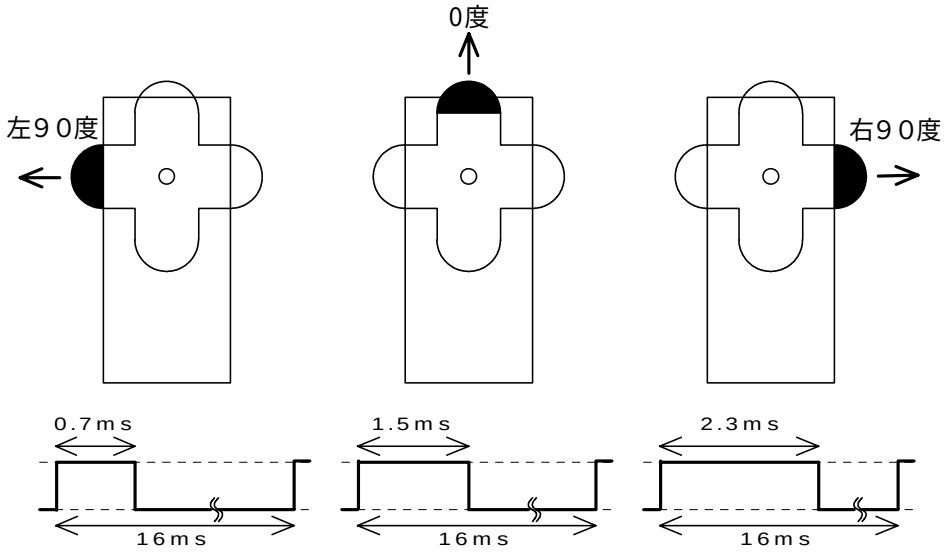
※A,B,C："0"=0V、"1"=5V

※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。

5.6 サーボ制御

5.6.1 原理

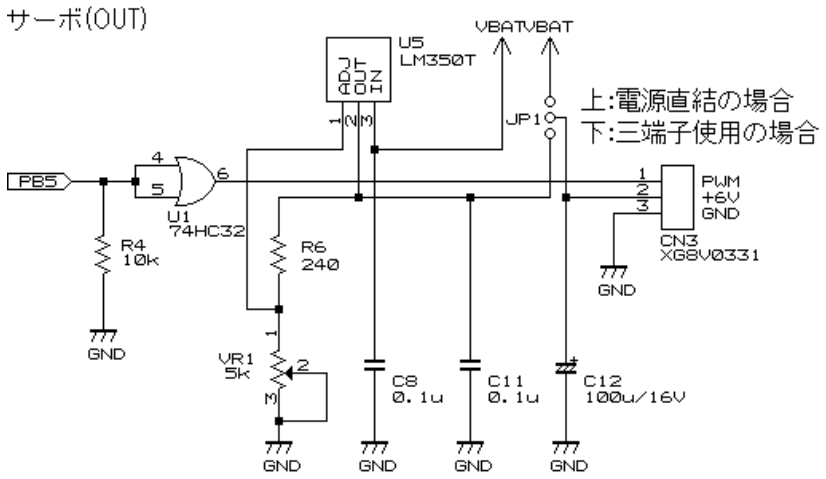
サーボは周期 16[ms]のパルスを加え、そのパルスの ON 幅でサーボの角度が決まります。
サーボの回転角度と ON のパルス幅の関係は、サーボのメーカや個体差によって多少の違いがありますが、ほとんどが下図のような関係になります。



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

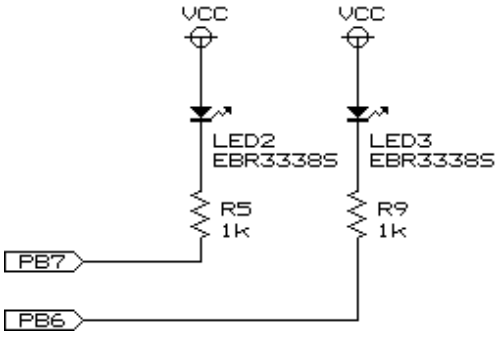
H8 マイコンのリセット同期式 PWM モードで PWM 信号を生成して、サーボを制御します。

5.6.2 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ピンの間に 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。

5.7 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

抵抗＝(電源電圧－LED に加える電圧)／LED に流したい電流

＝(5－1.7)／0.02

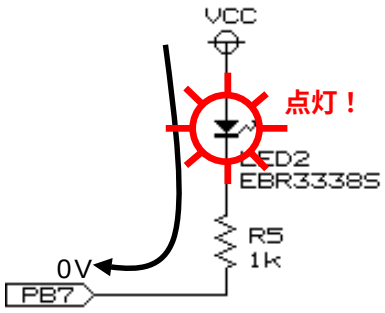
＝165Ω

となります。

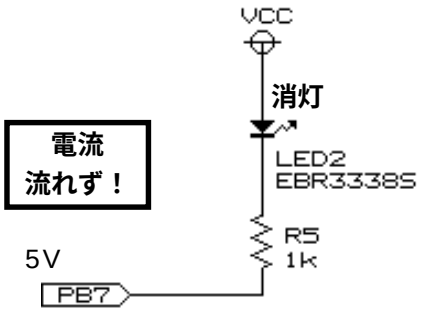
実際は、電池の消費電流を減らすのとポートの電流制限により、1kΩの抵抗を接続しています。電流は、

電流＝(電源電圧－LED に加える電圧)／抵抗

＝(5－1.7)／1000＝3.3[mA]となります。



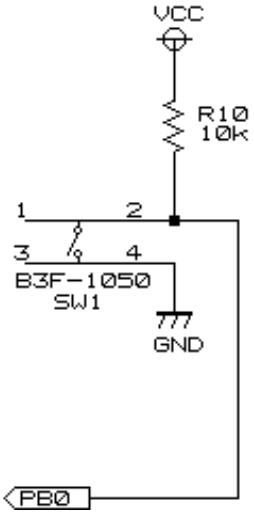
PB7 に"0"を出力すると、LED のカソード側が 0Vになり、電流が流れ、LED は光ります。



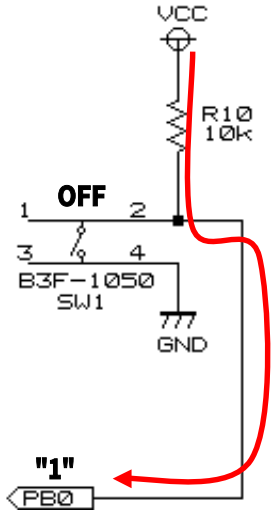
PB7 に"1"を出力すると、LED のカソード側が 5Vになり、LED の両端の電位差は0Vとなり、LED は光りません。

5.8 スイッチ制御

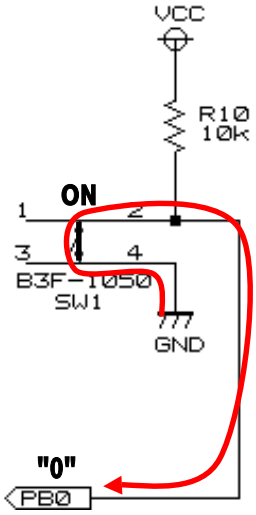
モータドライブ基板には、プッシュスイッチが1個付いています。



スイッチは、10kΩでプルアップされ、ポートBのビット0に繋がっています。



スイッチが押されていなければ、プルアップ抵抗を通して"1"が PB0 に入力されます。

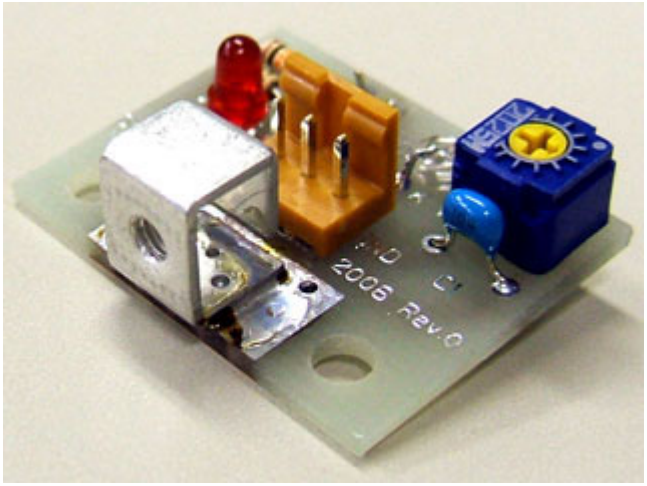


スイッチが押されると、GND を通して"0"が PB0 に入力されます。

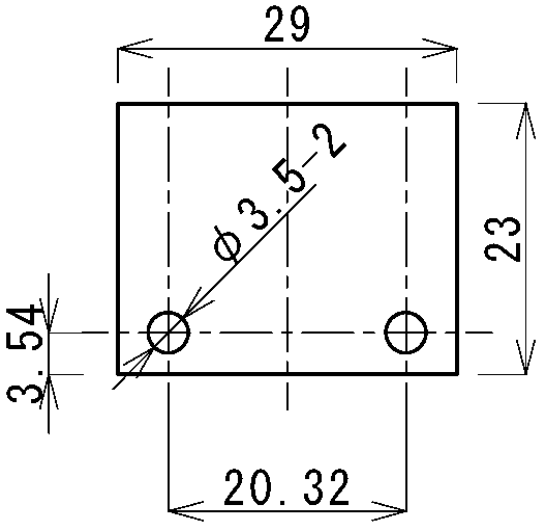
6. スタートバー検出センサ基板

6.1 仕様

内容	本体基板
機能	ジャパンマイコンカーラリー2007(2006 年度)より、スタートに関わる競技ルールが変わりました。コースのスタート位置にはスタートバーが設置され、マイコンカーがスタートバーの開閉を判断し、開くと自動でスタートするようになりました。スタートバーが閉じているか、開いているかを判断するのがスタートバー検出センサ基板です。 実は、コースの白黒を検出するセンサと回路は全く同じです。コースのセンサは下に向いていますが、スタートバー検出センサは前に向けます。スタートバーは白色なので、 ・反射あり→スタートバーあり→スタートバー閉→待機 ・反射なし→スタートバーなし→スタートバー開→スタート と判断できます。
動作電圧	DC5.0V±10%
寸法	最大 W29×D22×H15mm (実測)
コネクタ	センサ信号出力用 3 ピンコネクタ×1
信号	・スタートバーが閉じている場合、センサが反応し LED は点灯、出力信号は"0" ・スタートバーが開いた場合、センサが未反応になり LED は消灯、出力信号は"1"
重量	基板部分　約 5g (完成品の実測) ケーブル、コネクタ部分　約 6g (ケーブル長 60cm 時の目安)

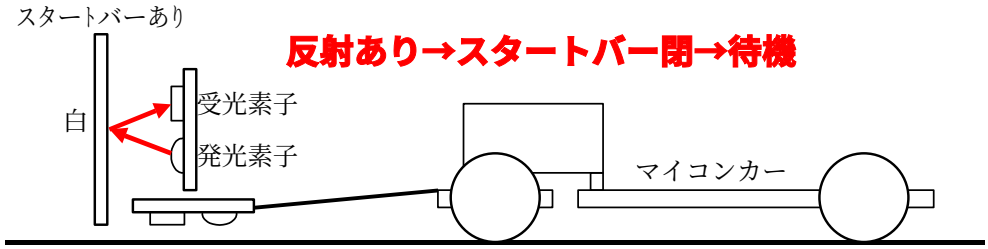


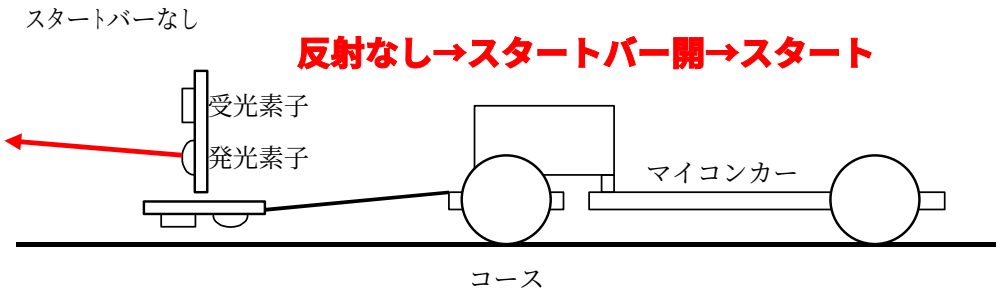
6.2 基板寸法



6.3 原理

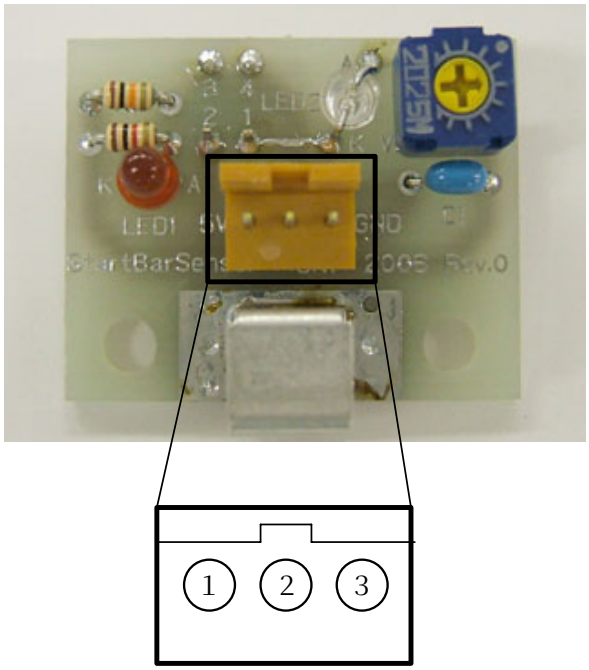
スタート時、白色のスタートバーが閉じています。コースの白黒を判定するセンサ1組を、前方向に取り付けます。センサの状況によって下記のように判断できます。





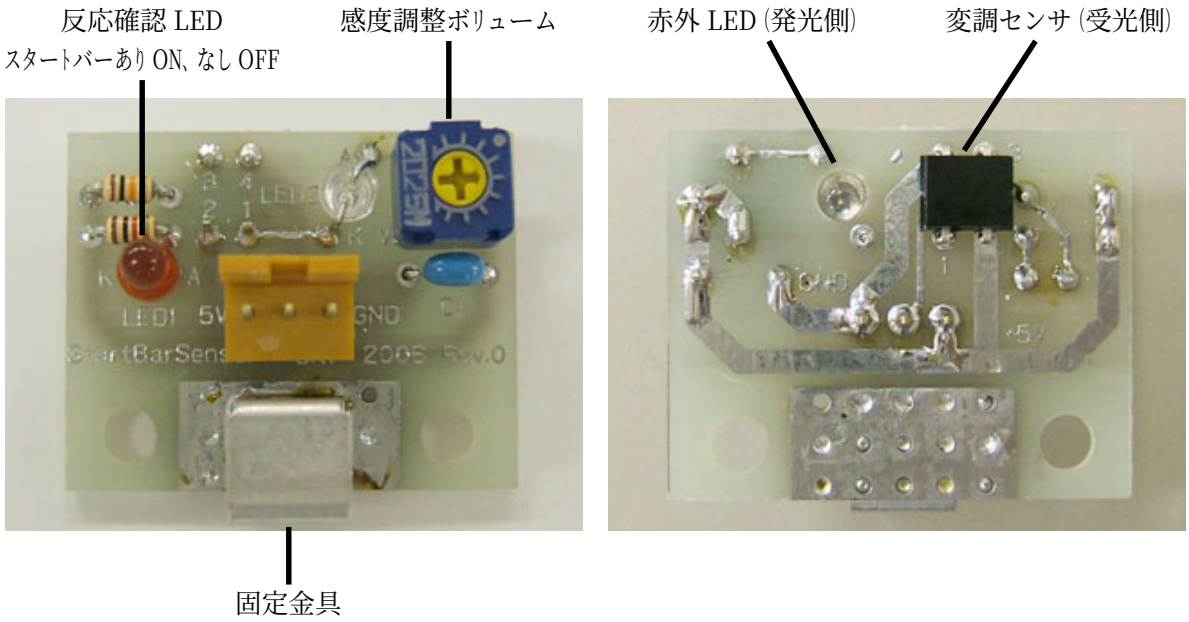
発光素子が出す光の量は、ボリュームで調整することができます。

6.4 コネクタ信号

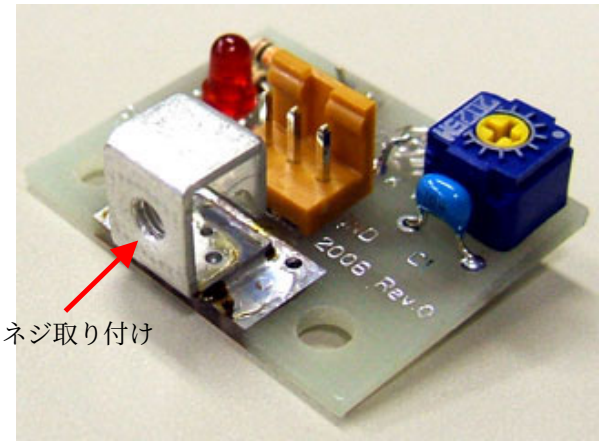


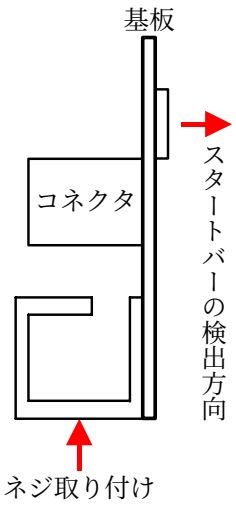
番号	方向	詳細
1	―	電源入力 5V
2	OUT	センサ信号 スタートバーあり："0"　スタートバー無し："1"
3	―	GND

6.5 部品位置

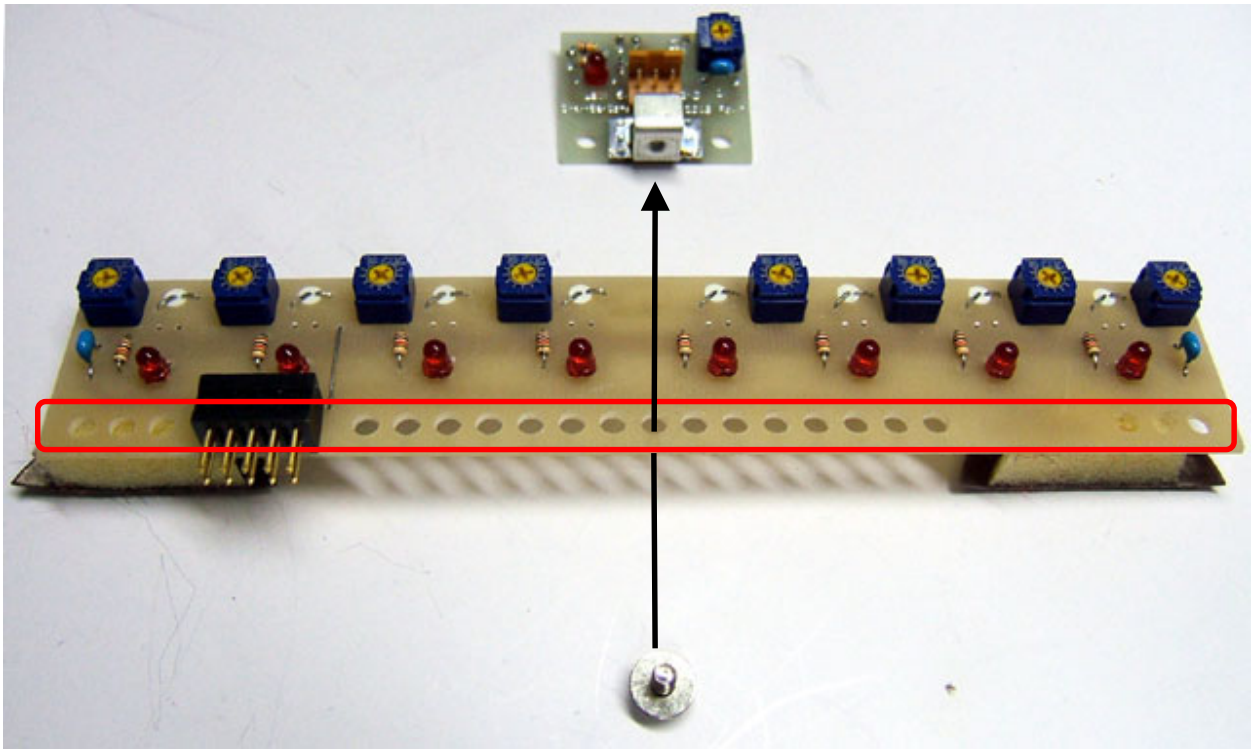


6.6 取り付け方

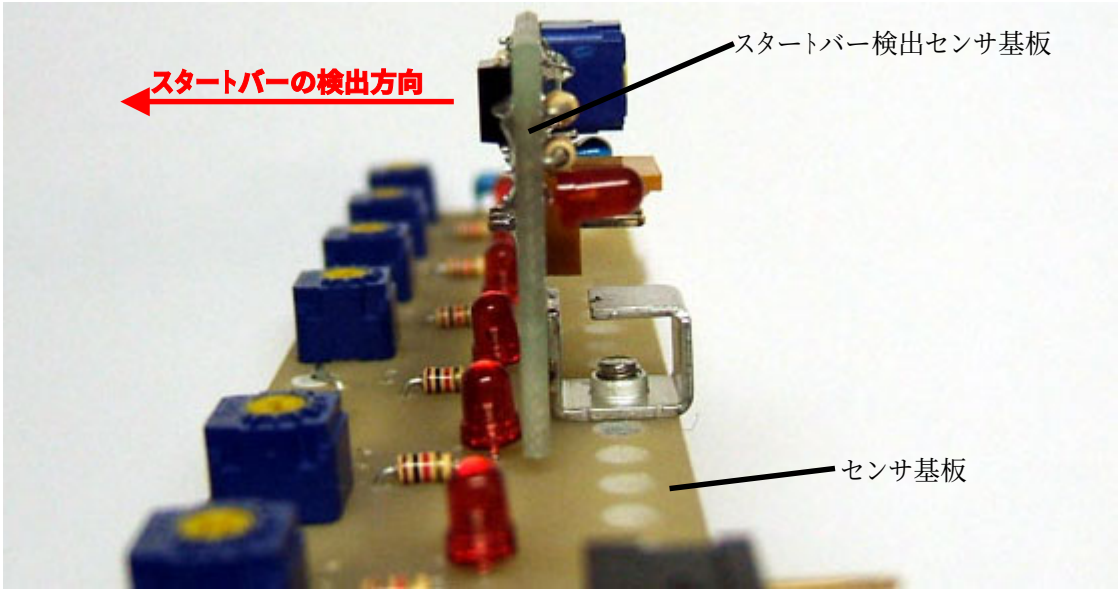




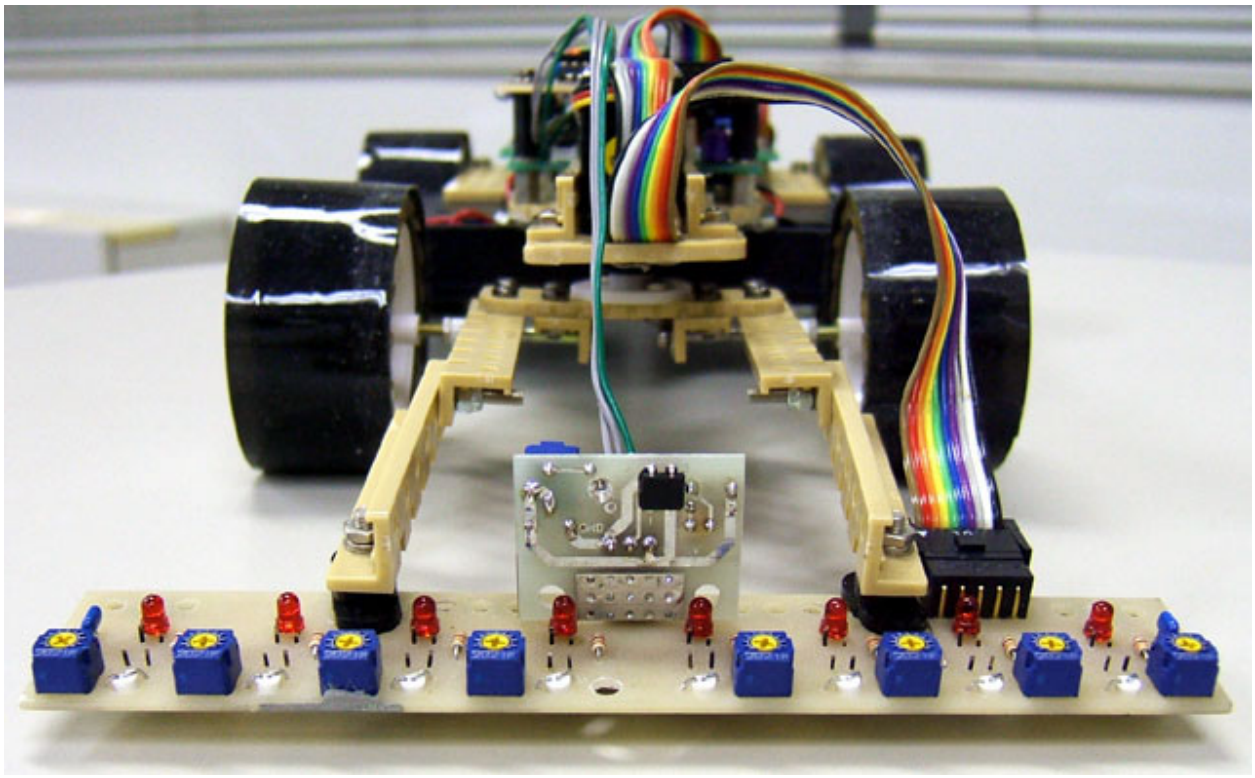
基板が立つように、ネジ取り付け金具が付いています。



センサ基板には、□部分に固定用の穴が多数空いています。この部分を利用して、スタートバー検出センサを取り付けます。ここでは例として、中心に取り付けてみます。ネジを基板の下から通して、スタートバー検出センサを止めます。ネジには必ずゆるみ防止のスプリングワッシャなどを入れます。



スタートバー検出センサ基板が、センサ基板に取り付けられました。



前から見ると、上の写真のようになります。

7. サンプルプログラム

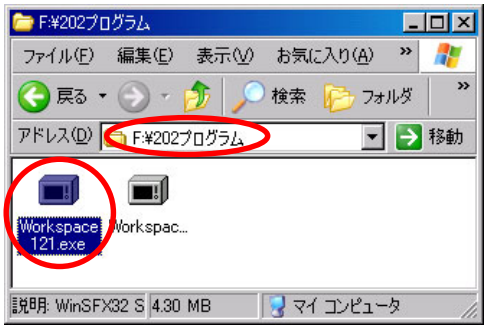
7.1 ルネサス統合開発環境

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

7.2 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。

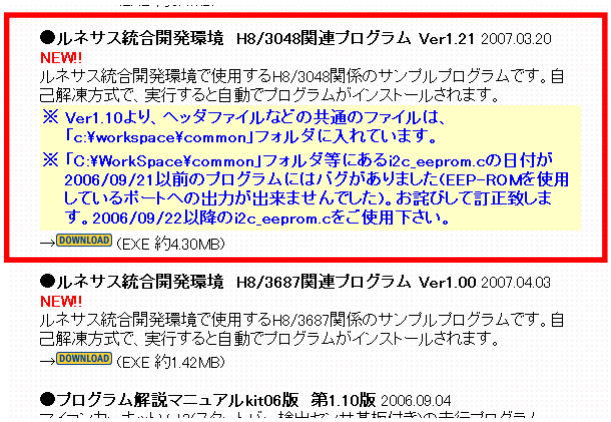
7.2.1 CD からソフトを取得する



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。

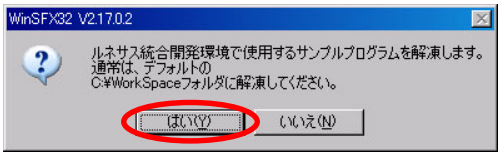
7.2.2 ホームページからソフトを取得する

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

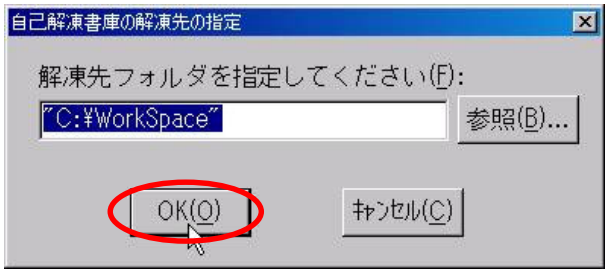


2.「ルネサス統合開発環境 H8/3048 関連プログラム」をダウンロードします。

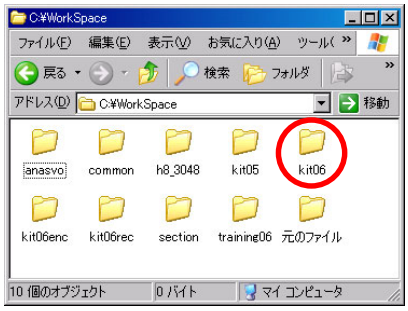
7.2.3 インストール



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。



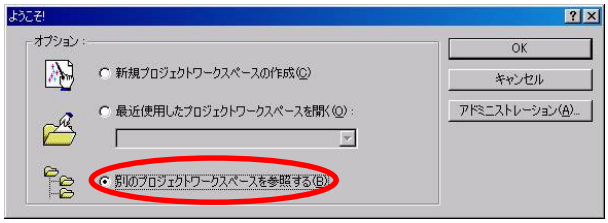
2.ファイルの解凍先を選択します。OK をクリックします。**このフォルダは変更出来ません。**



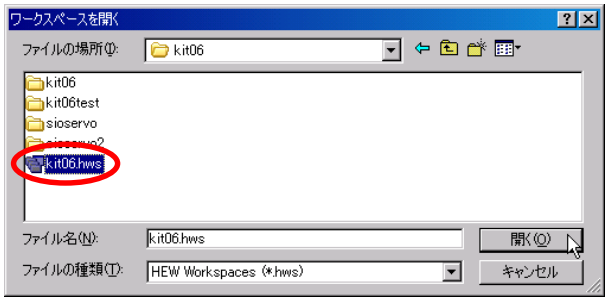
3. 解凍が終わったら、エクスプローラで「C ドライブ→Workspace」フォルダを開いてみてください。複数のフォルダがあります。今回使用するのは、「**kit06**」です。

7.3 ワーススペース「kit06」を開く

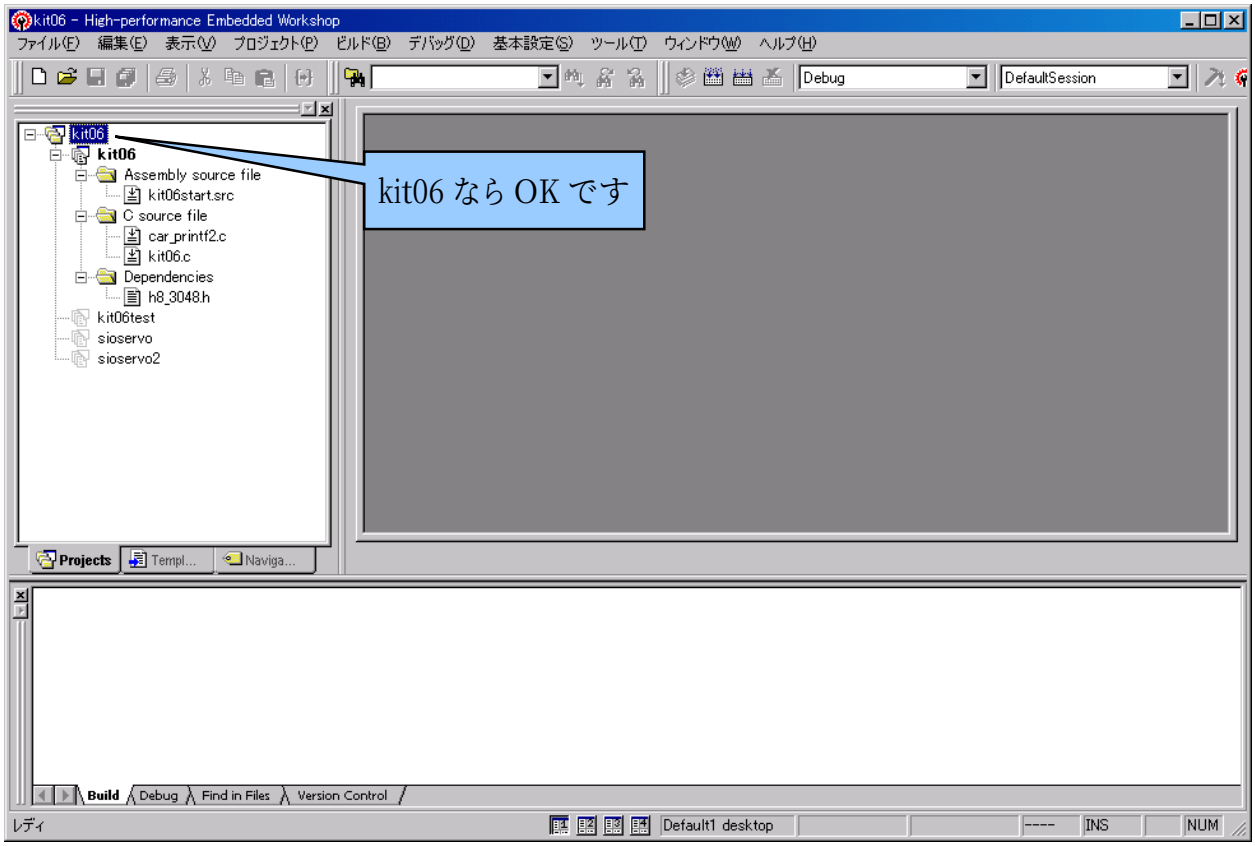
ルネサス統合開発環境を実行します。



「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。

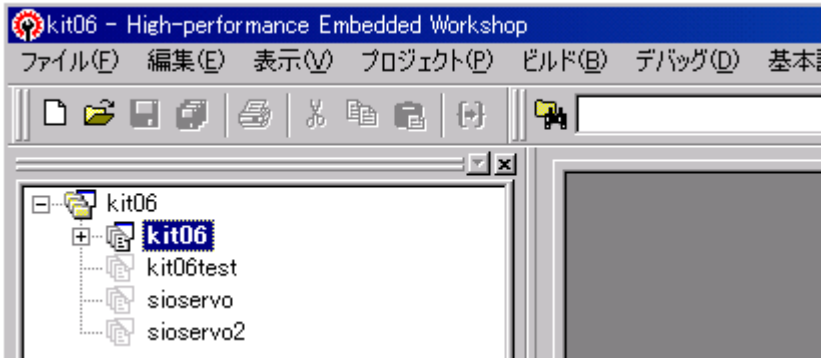


Cドライブ→Workspace→kit06 の「kit06.hws」を選択します。



kit06 というワークスペースが開かれます。

7.4 プロジェクト



ワークスペース「kit06」には、4つのプロジェクトが登録されています。

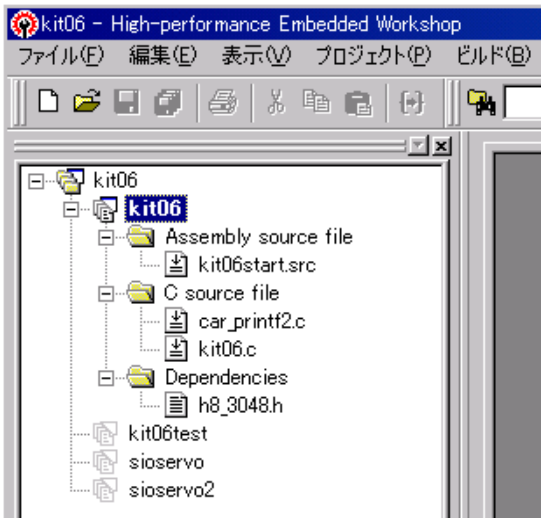
プロジェクト名	内容
kit06	マイコンカー走行プログラムです。 次章からプログラムの解説をします。
kit06test	製作したマイコンカーのモータドライブ基板やセンサ基板、スタートバー検出センサ基板が正しく動作するかテストします。詳しくは「動作テストマニュアル」を参照して下さい。
sioservo	サーボのセンタを調整するプログラムです。後述します。
sioservo2	サーボの最大切れ角を見つけるためのプログラムです。後述します。

8. プロジェクト内のファイルの関わりと実行順

8.1 概要

ここでは、ワークスペース「kit06」のプロジェクト「kit06」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、どの様な順番で実行されていくのか

8.2 プロジェクトのファイル構成



プロジェクト「kit06」は、下記のファイルから構成されています。

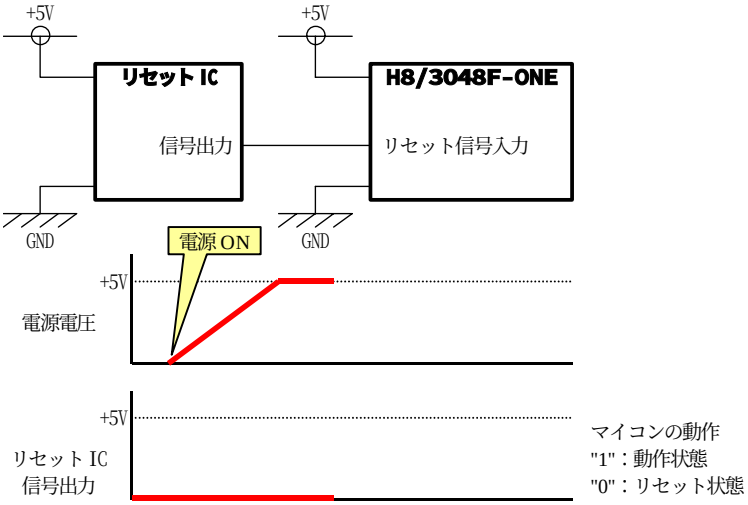
	ファイル名	内容
1	kit06start.src	アセンブリ言語で記述されたアセンブリソースファイルです。このファイルの構造は下記のようになっています。 kit06start.src ＝ ベクタアドレス ＋ スタートアップルーチン
2	kit06.c	C 言語ソースファイルです。このファイルは、H8/3048F-ONE の内蔵周辺機能の初期化、マイコンカーを制御するメインプログラムが含まれています。
3	car_printf2.c	C 言語ソースファイルです。初期値のないグローバル変数（セクション B 領域）、初期値のあるグローバル変数（セクション R 領域）の初期化用です。また、kit06.c 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。

8.3 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。

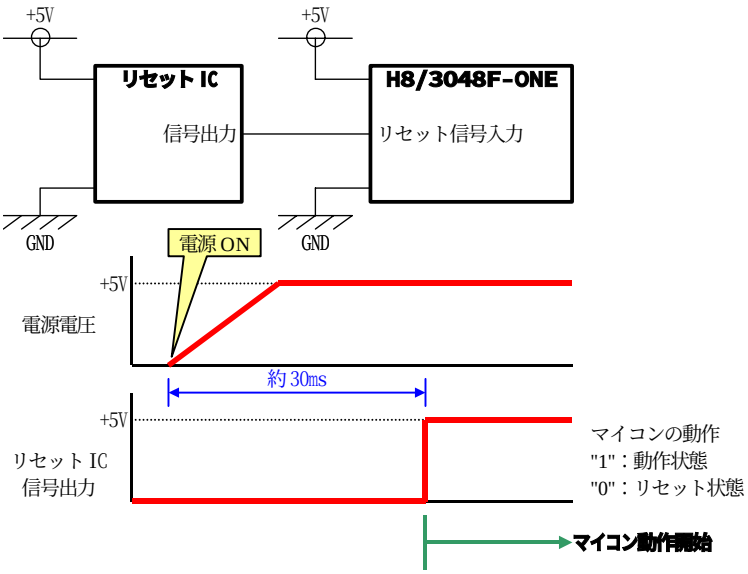
8.3.1 電源を入れたときの動作

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。



8.3.2 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。



8.3.3 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_0 割り込み発生時のジャンプ先アドレス
13 番：　IRQ_1 割り込み発生時のジャンプ先アドレス
・・・

とあらかじめ、「○○の割り込みが発生したときは、○○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、割り込みの発生元の関係は、次の表のようになっています。

割込み要因	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_0		12	H'0030 ～ H'0033	IPRA7	
IRQ_1		13	H'0034 ～ H'0037	IPRA6	
IRQ_2		14	H'0038 ～ H'003B	IPRA5	
IRQ_3		15	H'003C ～ H'003F		
IRQ_4		16	H'0040 ～ H'0043	IPRA4	
IRQ_5		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 (ｺﾝﾍﾟｱﾏｯﾁ／ｲﾝﾌﾟｯﾄｷｬﾌﾟﾁｬ A0)	ITU チャネル0	24	H'0060 ～ H'0063	IPRA2	
IMIB0 (ｺﾝﾍﾟｱﾏｯﾁ／ｲﾝﾌﾟｯﾄｷｬﾌﾟﾁｬ B0)		25	H'0064 ～ H'0067		
OVI0 (オーバフロー0)		26	H'0068 ～ H'006B		
ﾘｻﾞｰﾌﾞ	――――	27	H'006C ～ H'006F		
IMIA1 (ｺﾝﾍﾟｱﾏｯﾁ／ｲﾝﾌﾟｯﾄｷｬﾌﾟﾁｬ A1)	ITU チャネル1	28	H'0070 ～ H'0073	IPRA1	
IMIB1 (ｺﾝﾍﾟｱﾏｯﾁ／ｲﾝﾌﾟｯﾄｷｬﾌﾟﾁｬ B1)		29	H'0074 ～ H'0077		
OVI1 (オーバフロー1)		30	H'0078 ～ H'007B		
ﾘｻﾞｰﾌﾞ	――――	31	H'007C ～ H'007F		
IMIA2 (ｺﾝﾍﾟｱﾏｯﾁ／ｲﾝﾌﾟｯﾄｷｬﾌﾟﾁｬ A2)	ITU チャネル2	32	H'0080 ～ H'0083	IPRA0	
IMIB2 (ｺﾝﾍﾟｱﾏｯﾁ／ｲﾝﾌﾟｯﾄｷｬﾌﾟﾁｬ B2)		33	H'0084 ～ H'0087		
OVI2 (オーバフロー2)		34	H'0088 ～ H'008B		
ﾘｻﾞｰﾌﾞ	――――	35	H'008C ～ H'008F		
IMIA3 (ｺﾝﾍﾟｱﾏｯﾁ／ｲﾝﾌﾟｯﾄｷｬﾌﾟﾁｬ A3)	ITU チャネル3	36	H'0090 ～ H'0093	IPRB7	
IMIB3 (ｺﾝﾍﾟｱﾏｯﾁ／ｲﾝﾌﾟｯﾄｷｬﾌﾟﾁｬ B3)		37	H'0094 ～ H'0097		
OVI3 (オーバフロー3)		38	H'0098 ～ H'009B		
ﾘｻﾞｰﾌﾞ	――――	39	H'009C ～ H'009F		
IMIA4 (ｺﾝﾍﾟｱﾏｯﾁ／ｲﾝﾌﾟｯﾄｷｬﾌﾟﾁｬ A4)	ITU チャネル4	40	H'00A0 ～ H'00A3	IPRB6	
IMIB4 (ｺﾝﾍﾟｱﾏｯﾁ／ｲﾝﾌﾟｯﾄｷｬﾌﾟﾁｬ B4)		41	H'00A4 ～ H'00A7		
OVI4 (オーバフロー4)		42	H'00A8 ～ H'00AB		
ﾘｻﾞｰﾌﾞ	――――	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		
ﾘｻﾞｰﾌﾞ	――――	48 49 50 51	H'00C0 ～ H'00C3 H'00C4 ～ H'00C7 H'00C8 ～ H'00CB H'00CC ～ H'00CF	――	
ERI0 (受信エラー0)	SCI チャネル0	52	H'00D0 ～ H'00D3	IPRB3	
RXI0 (受信完了 0)		53	H'00D4 ～ H'00D7		
TXI0 (送信データエンプティ 0)		54	H'00D8 ～ H'00DB		
TEI0 (送信終了 0)		55	H'00DC ～ H'00DF		
ERI1 (受信エラー1)	SCI チャネル1	56	H'00E0 ～ H'00E3	IPRB2	
RXI1 (受信完了 1)		57	H'00E4 ～ H'00E7		
TXI1 (送信データエンプティ 1)		58	H'00E8 ～ H'00EB		
TEI1 (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、kit06start.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 :          .DATA.L RESET_START          ;  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v
 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
```

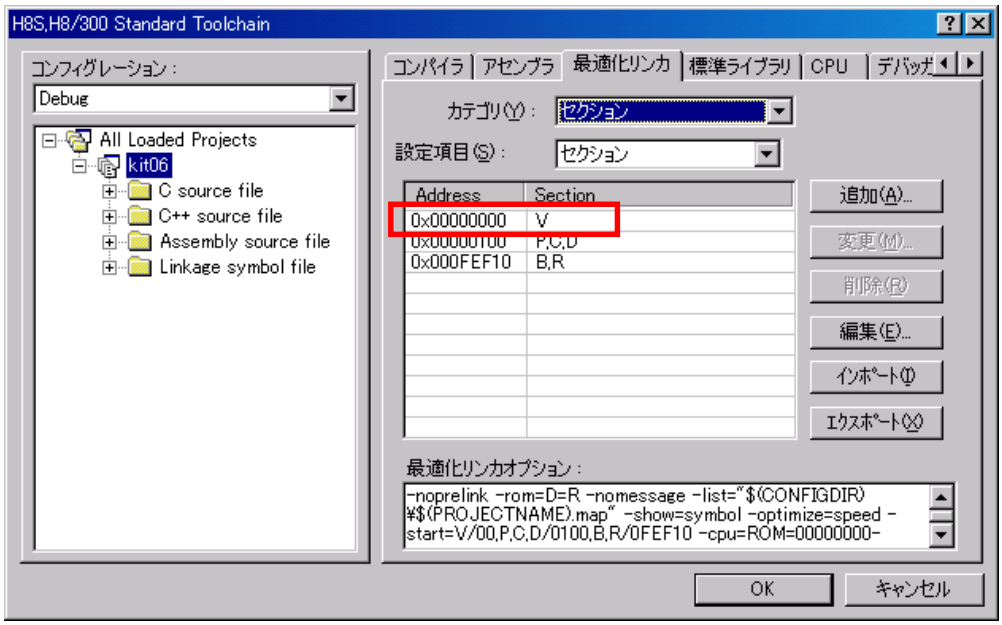
と登録しています。.EQU は、「ｲｺｰﾙ」命令です。「RESERVE」という単語が出てきたら「H'FFFFFFFF」に置き換えなさい、という意味です。登録する必要のない場所は、アドレスが分かりませんので、ダミーとして「H'FFFFFFFF」を入れておきます。結果的には、

```
 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インの設定によって、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インの設定です。ルネサス統合開発環境の「ビルド→H8S,H8/300 Standard Toolchain」を選択、「最適化リンカ、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 番地にしなさい」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。ツールチェイ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。

8.3.4　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。

kit06start.src

```
ベクタアドレス
・ジャンプ先アドレス 0：RESET_START
・ジャンプ先アドレス 1：…
・ジャンプ先アドレス 2：…
        ⋮
・ジャンプ先アドレス 60：…

スタートアップルーチン
(アセンブリソースプログラム)
RESET_START
        MOV.L   H'FFF10,ER7 ←ここから実行
        JSR     @_INITSCT
        JSR     @_main
```

car_printf2.c

```
C言語のプログラム
void INITSCT( void )
{
   セクション D→R へコピー
   セクション B のクリア
}
```

kit06.c

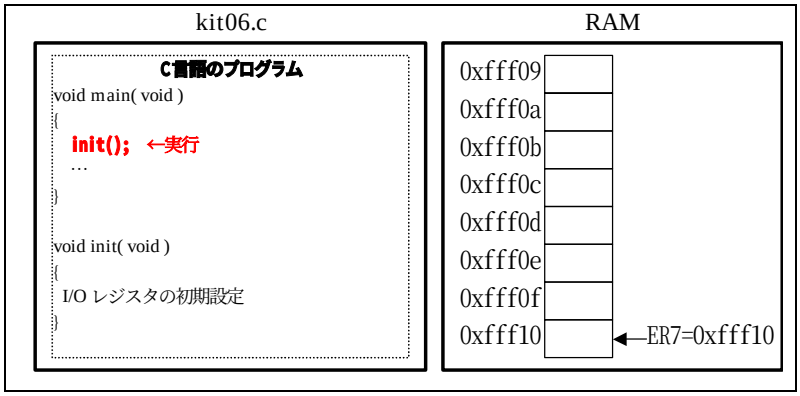
```
C言語のプログラム
void main( void )
{
  init();
  …
}

void init( void )
{
 I/O レジスタの初期設定
}
その他の関数
```

8.3.5 スタックポインタの設定

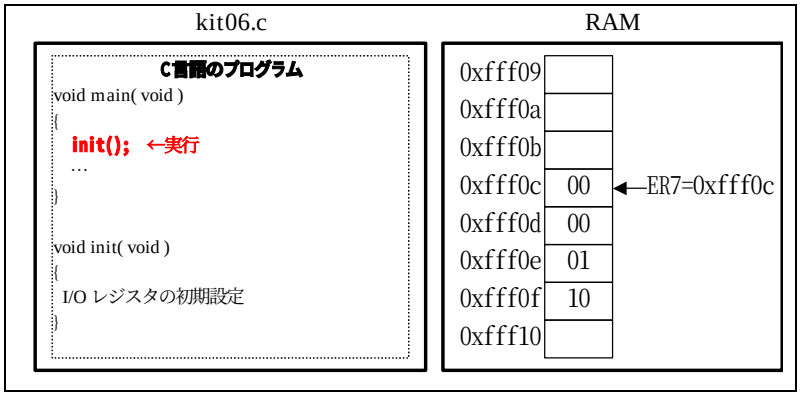
まず、スタックポインタの設定を行います。

```
83 :  RESET_START:
84 :          MOV.L   #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
```

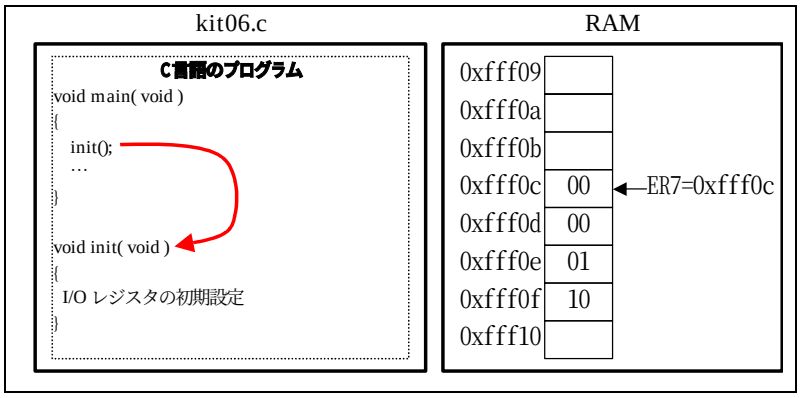
スタックポインタとは、番地やデータを一時的に待避させるアドレスのことです。
例えば、init 関数を呼んだとします（下図）。



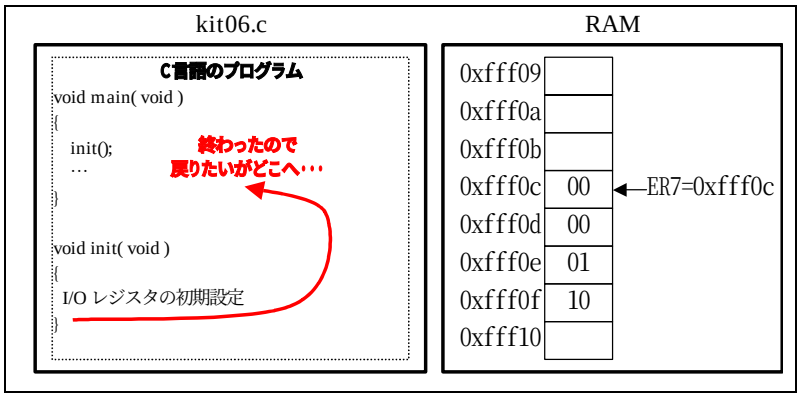
マイコンは、スタックポインタ(ER7)の値を-4 します。その番地である 0xfff0c 番地に、今実行しているプログラムの次のプログラムがある番地を書き込みます。例えば、その番地が 0x000110 番地なら、0xfff0c 番地には 0x00000110 と保存されます（下図）。



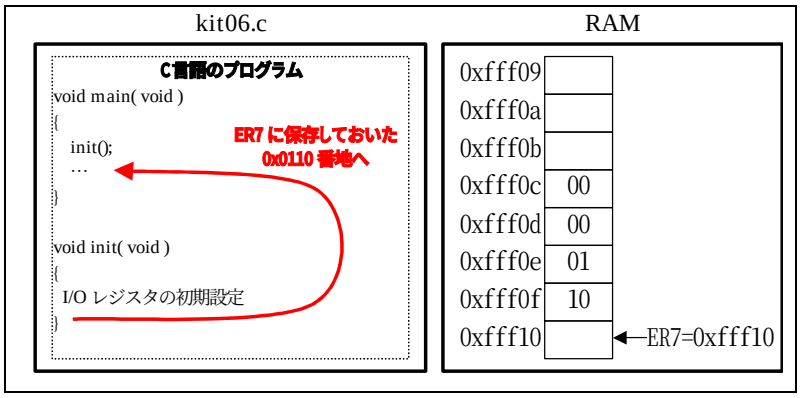
保存後、init 関数へジャンプして、init 関数を実行します（下図）。



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）。

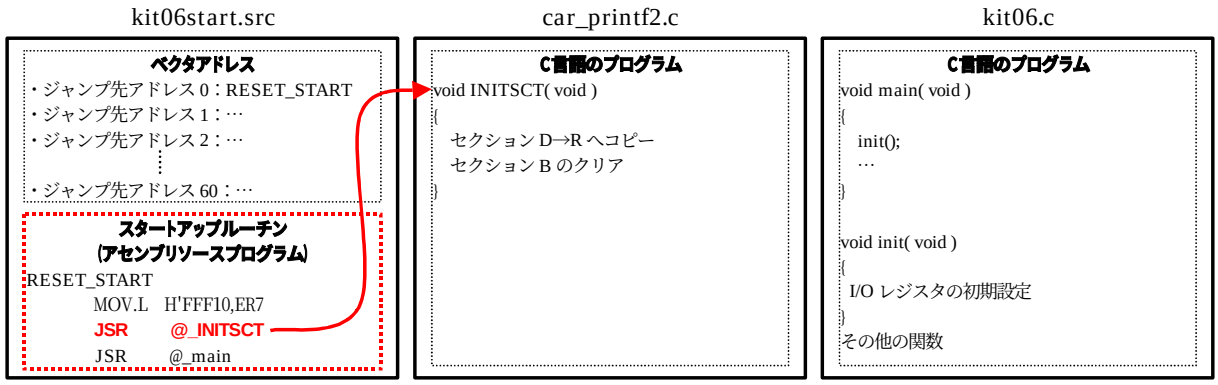


このように、スタックポインタは戻る番地や値を保存しておきます。**スタックポインタを設定しないと、戻り先が分からなくなるので、プログラムが暴走します。そのため、プログラムを実行するとき、一番最初にスタックポインタを設定しなければ行けないのです**。

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。

8.3.6 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。



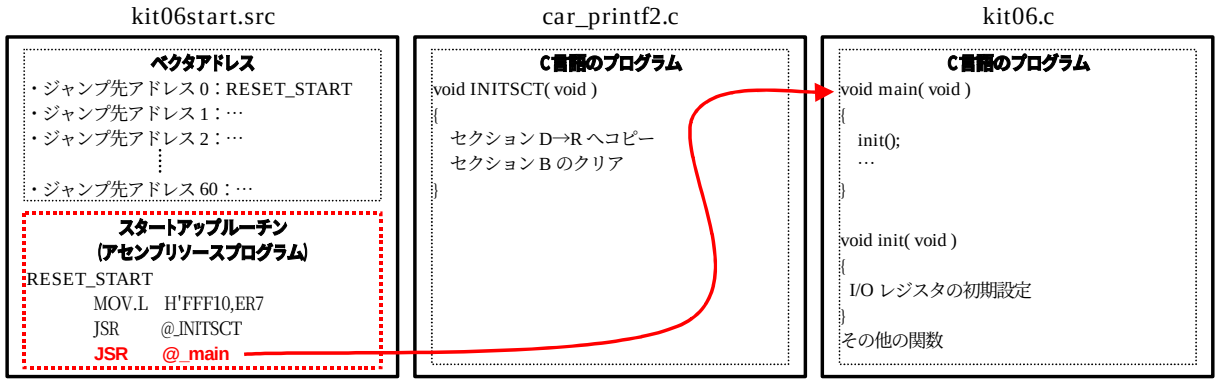
INITSCT 関数を実行します。この関数は何をしているのでしょうか。下記のような役割です。

・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。

8.3.7 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。



8.3.8 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 iostart.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、リンケージエ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
```

9. プログラム解説「kit06.c」

9.1 プログラムリスト

太字部分が、kit05.c から kit06.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「kit06.c」                             */
 3 :  /*                          2006.04 ジャパンマイコンカーラリー実行委員会     */
 4 :  /*****************************************************************************/
 5 :  /*
 6 :  本プログラム「kit06.c」は、
 7 :  ●モータドライブ基板(Vol.3)に対応
 8 :  ●スタートバーセンサ対応(PA3にセンサ信号)
 9 :  ●レーンチェンジ対応
10 :  しています。
11 :  */
12 :
13 :  /*====================================*/
14 :  /* インクルード                       */
15 :  /*====================================*/
16 :  #include    <machine.h>
17 :  #include    "h8_3048.h"
18 :
19 :  /*====================================*/
20 :  /* シンボル定義                       */
21 :  /*====================================*/
22 :
23 :  /* 定数設定 */
24 :  #define         TIMER_CYCLE     3071    /* タイマのサイクル 1ms      */
25 :                                          /* φ/8で使用する場合、      */
26 :                                          /* φ/8 = 325.5[ns]          */
27 :                                          /* ∴TIMER_CYCLE =            */
28 :                                          /*       1[ms] / 325.5[ns]   */
29 :                                          /*                 = 3072    */
30 :  #define         PWM_CYCLE       49151   /* PWMのサイクル 16ms        */
31 :                                          /* ∴PWM_CYCLE =              */
32 :                                          /*       16[ms] / 325.5[ns]  */
33 :                                          /*                 = 49152   */
34 :  #define         SERVO_CENTER    5000    /* サーボのセンタ値          */
35 :  #define         HANDLE_STEP     26      /* 1° 分の値                 */
36 :
37 :  /* マスク値設定 ×：マスクあり(無効)  ○：マスク無し(有効) */
38 :  #define         MASK2_2         0x66    /* ×○○××○○×          */
39 :  #define         MASK2_0         0x60    /* ×○○××××××          */
40 :  #define         MASK0_2         0x06    /* ××××××○○×          */
41 :  #define         MASK3_3         0xe7    /* ○○○××○○○          */
42 :  #define         MASK0_3         0x07    /* ××××××○○○          */
43 :  #define         MASK3_0         0xe0    /* ○○○××××××          */
44 :  #define         MASK4_0         0xf0    /* ○○○○××××          */
45 :  #define         MASK0_4         0x0f    /* ××××○○○○          */
46 :  #define         MASK4_4         0xff    /* ○○○○○○○○          */
47 :
48 :  /*====================================*/
49 :  /* プロトタイプ宣言                   */
50 :  /*====================================*/
51 :  void init( void );
52 :  void timer( unsigned long timer_set );
53 :  int check_crossline( void );
54 :  int check_rightline( void );
55 :  int check_leftline( void );
56 :  unsigned char sensor_inp( unsigned char mask );
57 :  unsigned char dipsw_get( void );
58 :  unsigned char pushsw_get( void );
59 :  unsigned char startbar_get( void );
60 :  void led_out( unsigned char led );
61 :  void speed( int accele_l, int accele_r );
62 :  void handle( int angle );
63 :  char unsigned bit_change( char unsigned in );
64 :
65 :  /*====================================*/
66 :  /* グローバル変数の宣言               */
67 :  /*====================================*/
68 :  unsigned long   cnt0;                   /* timer関数用               */
69 :  unsigned long   cnt1;                   /* main内で使用              */
70 :  int             pattern;                /* パターン番号              */
71 :
```

```
 72 : /************************************************************************/
 73 : /* メインプログラム                                                     */
 74 : /************************************************************************/
 75 : void main( void )
 76 : {
 77 :     int     i;
 78 :
 79 :     /* マイコン機能の初期化 */
 80 :     init();                         /* 初期化                   */
 81 :     set_ccr( 0x00 );                /* 全体割り込み許可         */
 82 :
 83 :     /* マイコンカーの状態初期化 */
 84 :     handle( 0 );
 85 :     speed( 0, 0 );
 86 :
 87 :     while( 1 ) {
 88 :     switch( pattern ) {
 89 :
 90 :     /*****************************************************************
 91 :     パターンについて
 92 :      0：スイッチ入力待ち
 93 :      1：スタートバーが開いたかチェック
 94 :     11：通常トレース
 95 :     12：右へ大曲げの終わりのチェック
 96 :     13：左へ大曲げの終わりのチェック
 97 :     21：１本目のクロスライン検出時の処理
 98 :     22：２本目を読み飛ばす
 99 :     23：クロスライン後のトレース、クランク検出
100 :     31：左クランククリア処理　安定するまで少し待つ
101 :     32：左クランククリア処理　曲げ終わりのチェック
102 :     41：右クランククリア処理　安定するまで少し待つ
103 :     42：右クランククリア処理　曲げ終わりのチェック
104 :     51：１本目の右ハーフライン検出時の処理
105 :     52：２本目を読み飛ばす
106 :     53：右ハーフライン後のトレース
107 :     54：右レーンチェンジ終了のチェック
108 :     61：１本目の左ハーフライン検出時の処理
109 :     62：２本目を読み飛ばす
110 :     63：左ハーフライン後のトレース
111 :     64：左レーンチェンジ終了のチェック
112 :     *****************************************************************/
113 :
114 :     case 0:
115 :         /* スイッチ入力待ち */
116 :         if( pushsw_get() ) {
117 :             pattern = 1;
118 :             cnt1 = 0;
119 :             break;
120 :         }
121 :         if( cnt1 < 100 ) {              /* LED点滅処理              */
122 :             led_out( 0x1 );
123 :         } else if( cnt1 < 200 ) {
124 :             led_out( 0x2 );
125 :         } else {
126 :             cnt1 = 0;
127 :         }
128 :         break;
129 :
130 :     case 1:
131 :         /* スタートバーが開いたかチェック */
132 :         if( !startbar_get() ) {
133 :             /* スタート！！ */
134 :             led_out( 0x0 );
135 :             pattern = 11;
136 :             cnt1 = 0;
137 :             break;
138 :         }
139 :         if( cnt1 < 50 ) {               /* LED点滅処理              */
140 :             led_out( 0x1 );
141 :         } else if( cnt1 < 100 ) {
142 :             led_out( 0x2 );
143 :         } else {
144 :             cnt1 = 0;
145 :         }
146 :         break;
147 :
148 :     case 11:
149 :         /* 通常トレース */
150 :         if( check_crossline() ) {       /* クロスラインチェック     */
151 :             pattern = 21;
152 :             break;
153 :         }
154 :         if( check_rightline() ) {       /* 右ハーフラインチェック   */
155 :             pattern = 51;
156 :             break;
157 :         }
158 :         if( check_leftline() ) {        /* 左ハーフラインチェック   */
159 :             pattern = 61;
160 :             break;
161 :         }
162 :         switch( sensor_inp(MASK3_3) ) {
```

```
163 :               case 0x00:
164 :                   /* センタ→まっすぐ */
165 :                   handle( 0 );
166 :                   speed( 100 ,100 );
167 :                   break;
168 :
169 :               case 0x04:
170 :                   /* 微妙に左寄り→右へ微曲げ */
171 :                   handle( 5 );
172 :                   speed( 100 ,100 );
173 :                   break;
174 :
175 :               case 0x06:
176 :                   /* 少し左寄り→右へ小曲げ */
177 :                   handle( 10 );
178 :                   speed( 80 ,69 );
179 :                   break;
180 :
181 :               case 0x07:
182 :                   /* 中くらい左寄り→右へ中曲げ */
183 :                   handle( 15 );
184 :                   speed( 50 ,40 );
185 :                   break;
186 :
187 :               case 0x03:
188 :                   /* 大きく左寄り→右へ大曲げ */
189 :                   handle( 25 );
190 :                   speed( 30 ,21 );
191 :                   pattern = 12;
192 :                   break;
193 :
194 :               case 0x20:
195 :                   /* 微妙に右寄り→左へ微曲げ */
196 :                   handle( -5 );
197 :                   speed( 100 ,100 );
198 :                   break;
199 :
200 :               case 0x60:
201 :                   /* 少し右寄り→左へ小曲げ */
202 :                   handle( -10 );
203 :                   speed( 69 ,80 );
204 :                   break;
205 :
206 :               case 0xe0:
207 :                   /* 中くらい右寄り→左へ中曲げ */
208 :                   handle( -15 );
209 :                   speed( 40 ,50 );
210 :                   break;
211 :
212 :               case 0xc0:
213 :                   /* 大きく右寄り→左へ大曲げ */
214 :                   handle( -25 );
215 :                   speed( 21 ,30 );
216 :                   pattern = 13;
217 :                   break;
218 :
219 :               default:
220 :                   break;
221 :           }
222 :           break;
223 :
224 :       case 12:
225 :           /* 右へ大曲げの終わりのチェック */
226 :           if( check_crossline() ) {       /* 大曲げ中もクロスラインチェック */
227 :               pattern = 21;
228 :               break;
229 :           }
230 :           if( check_rightline() ) {       /* 右ハーフラインチェック   */
231 :               pattern = 51;
232 :               break;
233 :           }
234 :           if( check_leftline() ) {        /* 左ハーフラインチェック   */
235 :               pattern = 61;
236 :               break;
237 :           }
238 :           if( sensor_inp(MASK3_3) == 0x06 ) {
239 :               pattern = 11;
240 :           }
241 :           break;
242 :
243 :       case 13:
244 :           /* 左へ大曲げの終わりのチェック */
245 :           if( check_crossline() ) {       /* 大曲げ中もクロスラインチェック */
246 :               pattern = 21;
247 :               break;
248 :           }
249 :           if( check_rightline() ) {       /* 右ハーフラインチェック   */
250 :               pattern = 51;
251 :               break;
252 :           }
```

```
253 :           if( check_leftline() ) {        /* 左ハーフラインチェック   */
254 :               pattern = 61;
255 :               break;
256 :           }
257 :           if( sensor_inp(MASK3_3) == 0x60 ) {
258 :               pattern = 11;
259 :           }
260 :           break;
261 :
262 :       case 21:
263 :           /* １本目のクロスライン検出時の処理 */
264 :           led_out( 0x3 );
265 :           handle( 0 );
266 :           speed( 0 ,0 );
267 :           pattern = 22;
268 :           cnt1 = 0;
269 :           break;
270 :
271 :       case 22:
272 :           /* ２本目を読み飛ばす */
273 :           if( cnt1 > 100 ) {
274 :               pattern = 23;
275 :               cnt1 = 0;
276 :           }
277 :           break;
278 :
279 :       case 23:
280 :           /* クロスライン後のトレース、クランク検出 */
281 :           if( sensor_inp(MASK4_4)==0xf8 ) {
282 :               /* 左クランクと判断→左クランククリア処理へ */
283 :               led_out( 0x1 );
284 :               handle( -38 );
285 :               speed( 10 ,50 );
286 :               pattern = 31;
287 :               cnt1 = 0;
288 :               break;
289 :           }
290 :           if( sensor_inp(MASK4_4)==0x1f ) {
291 :               /* 右クランクと判断→右クランククリア処理へ */
292 :               led_out( 0x2 );
293 :               handle( 38 );
294 :               speed( 50 ,10 );
295 :               pattern = 41;
296 :               cnt1 = 0;
297 :               break;
298 :           }
299 :           switch( sensor_inp(MASK3_3) ) {
300 :               case 0x00:
301 :                   /* センタ→まっすぐ */
302 :                   handle( 0 );
303 :                   speed( 40 ,40 );
304 :                   break;
305 :               case 0x04:
306 :               case 0x06:
307 :               case 0x07:
308 :               case 0x03:
309 :                   /* 左寄り→右曲げ */
310 :                   handle( 8 );
311 :                   speed( 40 ,36 );
312 :                   break;
313 :               case 0x20:
314 :               case 0x60:
315 :               case 0xe0:
316 :               case 0xc0:
317 :                   /* 右寄り→左曲げ */
318 :                   handle( -8 );
319 :                   speed( 36 ,40 );
320 :                   break;
321 :           }
322 :           break;
323 :
324 :       case 31:
325 :           /* 左クランククリア処理　安定するまで少し待つ */
326 :           if( cnt1 > 200 ) {
327 :               pattern = 32;
328 :               cnt1 = 0;
329 :           }
330 :           break;
331 :
332 :       case 32:
333 :           /* 左クランククリア処理　曲げ終わりのチェック */
334 :           if( sensor_inp(MASK3_3) == 0x60 ) {
335 :               led_out( 0x0 );
336 :               pattern = 11;
337 :               cnt1 = 0;
338 :           }
339 :           break;
340 :
341 :       case 41:
342 :           /* 右クランククリア処理　安定するまで少し待つ */
343 :           if( cnt1 > 200 ) {
```

```
344 :              pattern = 42;
345 :              cnt1 = 0;
346 :          }
347 :          break;
348 :
349 :      case 42:
350 :          /* 右クランククリア処理　曲げ終わりのチェック */
351 :          if( sensor_inp(MASK3_3) == 0x06 ) {
352 :              led_out( 0x0 );
353 :              pattern = 11;
354 :              cnt1 = 0;
355 :          }
356 :          break;
357 :
358 :      case 51:
359 :          /* １本目の右ハーフライン検出時の処理 */
360 :          led_out( 0x2 );
361 :          handle( 0 );
362 :          speed( 0 ,0 );
363 :          pattern = 52;
364 :          cnt1 = 0;
365 :          break;
366 :
367 :      case 52:
368 :          /* ２本目を読み飛ばす */
369 :          if( cnt1 > 100 ) {
370 :              pattern = 53;
371 :              cnt1 = 0;
372 :          }
373 :          break;
374 :
375 :      case 53:
376 :          /* 右ハーフライン後のトレース、レーンチェンジ */
377 :          if( sensor_inp(MASK4_4) == 0x00 ) {
378 :              handle( 15 );
379 :              speed( 40 ,32 );
380 :              pattern = 54;
381 :              cnt1 = 0;
382 :              break;
383 :          }
384 :          switch( sensor_inp(MASK3_3) ) {
385 :              case 0x00:
386 :                  /* センタ→まっすぐ */
387 :                  handle( 0 );
388 :                  speed( 40 ,40 );
389 :                  break;
390 :              case 0x04:
391 :              case 0x06:
392 :              case 0x07:
393 :              case 0x03:
394 :                  /* 左寄り→右曲げ */
395 :                  handle( 8 );
396 :                  speed( 40 ,36 );
397 :                  break;
398 :              case 0x20:
399 :              case 0x60:
400 :              case 0xe0:
401 :              case 0xc0:
402 :                  /* 右寄り→左曲げ */
403 :                  handle( -8 );
404 :                  speed( 36 ,40 );
405 :                  break;
406 :              default:
407 :                  break;
408 :          }
409 :          break;
410 :
411 :      case 54:
412 :          /* 右レーンチェンジ終了のチェック */
413 :          if( sensor_inp( MASK4_4 ) == 0x3c ) {
414 :              led_out( 0x0 );
415 :              pattern = 11;
416 :              cnt1 = 0;
417 :          }
418 :          break;
419 :
420 :      case 61:
421 :          /* １本目の左ハーフライン検出時の処理 */
422 :          led_out( 0x1 );
423 :          handle( 0 );
424 :          speed( 0 ,0 );
425 :          pattern = 62;
426 :          cnt1 = 0;
427 :          break;
428 :
429 :      case 62:
430 :          /* ２本目を読み飛ばす */
431 :          if( cnt1 > 100 ) {
432 :              pattern = 63;
433 :              cnt1 = 0;
434 :          }
```

```
435 :          break;
436 :
437 :      case 63:
438 :          /* 左ハーフライン後のトレース、レーンチェンジ */
439 :          if( sensor_inp(MASK4_4) == 0x00 ) {
440 :              handle( -15 );
441 :              speed( 32 ,40 );
442 :              pattern = 64;
443 :              cnt1 = 0;
444 :              break;
445 :          }
446 :          switch( sensor_inp(MASK3_3) ) {
447 :              case 0x00:
448 :                  /* センタ→まっすぐ */
449 :                  handle( 0 );
450 :                  speed( 40 ,40 );
451 :                  break;
452 :              case 0x04:
453 :              case 0x06:
454 :              case 0x07:
455 :              case 0x03:
456 :                  /* 左寄り→右曲げ */
457 :                  handle( 8 );
458 :                  speed( 40 ,36 );
459 :                  break;
460 :              case 0x20:
461 :              case 0x60:
462 :              case 0xe0:
463 :              case 0xc0:
464 :                  /* 右寄り→左曲げ */
465 :                  handle( -8 );
466 :                  speed( 36 ,40 );
467 :                  break;
468 :              default:
469 :                  break;
470 :          }
471 :          break;
472 :
473 :      case 64:
474 :          /* 左レーンチェンジ終了のチェック */
475 :          if( sensor_inp( MASK4_4 ) == 0x3c ) {
476 :              led_out( 0x0 );
477 :              pattern = 11;
478 :              cnt1 = 0;
479 :          }
480 :          break;
481 :
482 :      default:
483 :          /* どれでもない場合は待機状態に戻す */
484 :          pattern = 0;
485 :          break;
486 :      }
487 :      }
488 :  }
489 :
490 :  /************************************************************************/
491 :  /* H8/3048F-ONE 内蔵周辺機能　初期化                                    */
492 :  /************************************************************************/
493 :  void init( void )
494 :  {
495 :      /* I/Oポートの入出力設定 */
496 :      P1DDR = 0xff;
497 :      P2DDR = 0xff;
498 :      P3DDR = 0xff;
499 :      P4DDR = 0xff;
500 :      P5DDR = 0xff;
501 :      P6DDR = 0xf0;                       /* CPU基板上のDIP SW        */
502 :      P8DDR = 0xff;
503 :      P9DDR = 0xf7;                       /* 通信ポート               */
504 :      PADDR = 0xf7;                       /* スタートバー検出センサ   */
505 :      PBDR  = 0xc0;
506 :      PBDDR = 0xfe;                       /* モータドライブ基板Vol.3  */
507 :      /* ※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     */
508 :
509 :      /* ITU0 1ms毎の割り込み */
510 :      ITU0_TCR = 0x23;
511 :      ITU0_GRA = TIMER_CYCLE;
512 :      ITU0_IER = 0x01;
513 :
514 :      /* ITU3,4 リセット同期PWMモード 左右モータ、サーボ用 */
515 :      ITU3_TCR = 0x23;
516 :      ITU_FCR  = 0x3e;
517 :      ITU3_GRA = PWM_CYCLE;               /* 周期の設定               */
518 :      ITU3_GRB = ITU3_BRB = 0;            /* 左モータのPWM設定        */
519 :      ITU4_GRA = ITU4_BRA = 0;            /* 右モータのPWM設定        */
520 :      ITU4_GRB = ITU4_BRB = SERVO_CENTER; /* サーボのPWM設定          */
521 :      ITU_TOER = 0x38;
522 :
523 :      /* ITUのカウントスタート */
524 :      ITU_STR = 0x09;
525 :  }
```

```
526 :
527 :  /************************************************************************/
528 :  /* ITU0 割り込み処理                                                    */
529 :  /************************************************************************/
530 :  #pragma interrupt( interrupt_timer0 )
531 :  void interrupt_timer0( void )
532 :  {
533 :      ITU0_TSR &= 0xfe;                   /* フラグクリア             */
534 :      cnt0++;
535 :      cnt1++;
536 :  }
537 :
538 :  /************************************************************************/
539 :  /* タイマ本体                                                           */
540 :  /* 引数   タイマ値 1=1ms                                                */
541 :  /************************************************************************/
542 :  void timer( unsigned long timer_set )
543 :  {
544 :      cnt0 = 0;
545 :      while( cnt0 < timer_set );
546 :  }
547 :
548 :  /************************************************************************/
549 :  /* センサ状態検出                                                       */
550 :  /* 引数   マスク値                                                      */
551 :  /* 戻り値 センサ値                                                      */
552 :  /************************************************************************/
553 :  unsigned char sensor_inp( unsigned char mask )
554 :  {
555 :      unsigned char sensor;
556 :
557 :      sensor  = P7DR;
558 :
559 :      /* 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     */
560 :      /* 逆なので、互換を保つためビットを入れ替える                       */
561 :      sensor  = bit_change( sensor );     /* ビット入れ替え               */
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(MASK2_2);
578 :      if( b==0x66 || b==0x64 || b==0x26 || b==0x62 || b==0x46 ) {
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==0x0f || 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==0xf0 || b==0xf8 ) {
613 :          ret = 1;
614 :      }
615 :      return ret;
616 :  }
```

```
617 :
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 : }
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  = ~PADR;                         /* スタートバー信号読み込み  */
655 :     b &= 0x08;
656 :     b >>= 3;
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;         /* ディ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 :  /* ビット入れ替え                                                        */
722 :  /* 引数    入れ替えする値                                                */
723 :  /* 戻り値 入れ替え後の値                                                 */
724 :  /*************************************************************************/
725 :  char unsigned bit_change( char unsigned in )
726 :  {
727 :      unsigned char ret;
728 :      int i;
729 :
730 :      for( i = 0;  i < 8; i++ ) {
731 :          ret >>= 1;                  /* 戻り値の右シフト              */
732 :          ret |= in & 0x80;           /* ret bit7 = in bit7            */
733 :          in  <<= 1;                  /* 引数の左シフト                */
734 :      }
735 :      return ret;
736 :  }
737 :
738 :  /*************************************************************************/
739 :  /* end of file                                                           */
740 :  /*************************************************************************/
```

9.2 スタート

```
1 :  /*************************************************************************/
2 :  /* マイコンカートレース基本プログラム「kit06.c」                         */
3 :  /*                              2006.04 ジャパンマイコンカーラリー実行委員会 */
4 :  /*************************************************************************/
5 :  /*
6 :  本プログラム kit06.c は、
7 :  ●モータドライブ基板(Vol.3)に対応
8 :  ●スタートバーセンサ対応(PA3 にセンサ信号)
9 :  ●レーンチェンジ対応
10 :  しています。
11 :  */
```

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。

9.3 外部ファイルの取り込み（インクルード）

```
13 :  /*====================================*/
14 :  /* インクルード                       */
15 :  /*====================================*/
16 :  #include    <machine.h>
17 :  #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。

9.4 その他のシンボル定義

```
19 :  /*====================================*/
20 :  /* シンボル定義                       */
21 :  /*====================================*/
22 :
23 :  /* 定数設定 */
24 :  #define         TIMER_CYCLE     3071    /* タイマのサイクル 1ms      */
25 :                                          /* φ/8で使用する場合、      */
26 :                                          /* φ/8 = 325.5[ns]          */
27 :                                          /* ∴TIMER_CYCLE =           */
28 :                                          /*      1[ms] / 325.5[ns]    */
29 :                                          /*              = 3072       */
30 :  #define         PWM_CYCLE       49151   /* PWMのサイクル 16ms        */
31 :                                          /* ∴PWM_CYCLE =             */
32 :                                          /*      16[ms] / 325.5[ns]   */
33 :                                          /*              = 49152      */
34 :  #define         SERVO_CENTER    5000    /* サーボのセンタ値          */
35 :  #define         HANDLE_STEP     26      /* 1°分の値                 */
36 :
37 :  /* マスク値設定 ×：マスクあり(無効)  ○：マスク無し(有効) */
38 :  #define         MASK2_2         0x66    /* ×○○××○○×               */
39 :  #define         MASK2_0         0x60    /* ×○○××××××               */
40 :  #define         MASK0_2         0x06    /* ××××××○○×               */
41 :  #define         MASK3_3         0xe7    /* ○○○××○○○                 */
42 :  #define         MASK0_3         0x07    /* ××××××○○○                */
43 :  #define         MASK3_0         0xe0    /* ○○○××××××                */
44 :  #define         MASK4_0         0xf0    /* ○○○○××××××                */
45 :  #define         MASK0_4         0x0f    /* ××××○○○○                 */
46 :  #define         MASK4_4         0xff    /* ○○○○○○○○                 */
```

TIMER_CYCLE	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、 $(1\times10^{-3})\div(325.52\times10^{-9})-1=$**3,071** となります。詳しい説明は、ITU0 部分を参照してください。
PWM_CYCLE	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。 $(16\times10^{-3})\div(325.52\times10^{-9})-1=$**49,151** 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。
SERVO_CENTER	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、サーボに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。 $(1.5\times10^{-3})\div(325.52\times10^{-9})-1=4,607$ となります。しかし、サーボのセンタは**サーボ自体の誤差、サーボホーンに挿すギザギザのかみ合わせ方などの影響ですべてのマイコンカーで違う値になります**。例えるなら、人間の指紋のようなものでしょうか。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**、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**。  ▲サーボホーン
HANDLE_STEP	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、 $(2.3\times10^{-3})\div(325.52\times10^{-9})=7,065$ サーボが、左に 90 度向くときは 0.7ms のパルスなので、 $(0.7\times10^{-3})\div(325.52\times10^{-9})=2,150$ （右 90 度）－（左 90 度）＝7,065-2,150＝4,915 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。 4,915÷180＝27.31 正確に計算すると 27 がサーボ 1 度分の値です。ただし、前のプログラム kit05.c では 26 としていたので、過去との互換を考えて、**26** とします。
MASK2_2	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「**0110 0110**」と bit6,5,2,1 をそのままに、他は強制的に”0”にしています。
MASK2_0	「MASK2_0」は、「0x60」と定義していますので、2進数で「**0110 0000**」と bit6,5 をそのままに、他は強制的に”0”にしています。
MASK0_2	「MASK0_2」は、「0x06」と定義していますので、2進数で「**0000 0110**」と bit2,1 をそのままに、他は強制的に”0”にしています。
MASK3_3	「MASK3_3」は、「0xe7」と定義していますので、2進数で「**1110 0111**」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。
MASK0_3	「MASK0_3」は、「0x07」と定義していますので、2進数で「**0000 0111**」と bit2,1,0 をそのままに、他は強制的に”0”にしています。
MASK3_0	「MASK3_0」は、「0xe0」と定義していますので、2進数で「**1110 0000**」と bit7,6,5 をそのままに、他は強制的に”0”にしています。
MASK4_0	「MASK4_0」は、「0xf0」と定義していますので、2進数で「**1111 0000**」と bit7,6,5,4 をそのままに、他は強制的に”0”にしています。
MASK0_4	「MASK0_4」は、「0x0f」と定義していますので、2進数で「**0000 1111**」と bit3,2,1,0 をそのままに、他は強制的に”0”にしています。
MASK4_4	「MASK4_4」は、「0xff」と定義していますので、2進数で「**1111 1111**」とすべてのビットを有効にしています。

※マスクについては後述します。

※kit05.c と比べ、MASK1_1 を削除、MASK4_4 を追加しています。kit05.c では、MASK1_1 を使用していませんでしたのでキットのプログラムではエラーになりませんが、プログラムを改造して MASK1_1 を使っていた場合は、

```
#define         MASK1_1         0x81    /* ○××××××○          */
```

を MASK4_4 の後に追加してください。

9.5 プロトタイプ宣言

```
 48 :  /*====================================*/
 49 :  /* プロトタイプ宣言                   */
 50 :  /*====================================*/
 51 :  void init( void );
 52 :  void timer( unsigned long timer_set );
 53 :  int check_crossline( void );
 54 :  int check_rightline( void );
 55 :  int check_leftline( void );
 56 :  unsigned char sensor_inp( unsigned char mask );
 57 :  unsigned char dipsw_get( void );
 58 :  unsigned char pushsw_get( void );
 59 :  unsigned char startbar_get( void );
 60 :  void led_out( unsigned char led );
 61 :  void speed( int accele_l, int accele_r );
 62 :  void handle( int angle );
 63 :  char unsigned bit_change( char unsigned in );
```

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。

9.6 グローバル変数の宣言

```
 65 :  /*====================================*/
 66 :  /* グローバル変数の宣言                */
 67 :  /*====================================*/
 68 :  unsigned long   cnt0;                  /* timer関数用              */
 69 :  unsigned long   cnt1;                  /* main内で使用             */
 70 :  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;
}
```

kit06.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○○をしなさい、たっていないなら□□をしなさい、というように使用します。main 関数部分で詳しく説明します。
pattern	int	パターン番号です。main 関数部分で詳しく説明します。

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

9.7 メインプログラムを説明する前に

72～488 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。

9.8 H8/3048F-ONE 内蔵周辺機能の初期化：init 関数

9.8.1 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。

```
490 :  /************************************************************************/
491 :  /* H8/3048F-ONE 内蔵周辺機能　初期化                                    */
492 :  /************************************************************************/
493 :  void init( void )
494 :  {
495 :      /* I/0ポートの入出力設定 */
496 :      P1DDR = 0xff;
497 :      P2DDR = 0xff;
498 :      P3DDR = 0xff;
499 :      P4DDR = 0xff;
500 :      P5DDR = 0xff;
501 :      P6DDR = 0xf0;                   /* CPU基板上のDIP SW         */
502 :      P8DDR = 0xff;
503 :      P9DDR = 0xf7;                   /* 通信ポート                */
504 :      PADDR = 0xf7;                   /* スタートバー検出センサ    */
505 :      PBDR  = 0xc0;
506 :      PBDDR = 0xfe;                   /* モータドライブ基板Vol.3   */
507 :      /* ※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      */
```

kit06.c は、「0xff」です。元からあるプログラムを改造する場合は、この部分の変更を忘れがちなので気をつけてください。

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。

9.8.2 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**	**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**	**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	スタートバーセンサ基板	未接続	未接続	未接続	未接続	**スタートバーセンサ入力**	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**

9.8.3 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	出力端子は出力を設定します。
入力	入力端子は入力を設定します。
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。**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。** **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。**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。
—	端子のないビットです。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。

9.8.4 実際の設定

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	**入力**	出力	出力	出力	**0xf7**
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。

この表を基に、ポートの入出力設定を行います。

9.8.5 ポート A の詳細

ポート A にはスタートバー検出センサ基板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←PA7				**出力**
3	基板←PA6				**出力**
4	基板←PA5				**出力**
5	基板←PA4				**出力**
6	基板→PA3	スタートバー検出 センサ基板	スタートバー あり	スタートバー なし	**入力**
7	基板←PA2				**出力**
8	基板←PA1				**出力**
9	基板←PA0				**出力**
10	―	GND			

9.8.6 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	押されてない	**入力**
10	―	GND			

9.8.7 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。

ビット	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。

ビット	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。

```
505 :       PBDR  = 0xc0;
506 :       PBDDR = 0xfe;                    /* モータドライブ基板 Vol.3  */
```

となります。

9.8.8 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。このようなミスを無くすために、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。

9.9 ITU0 1ms ごとの割り込み設定

ITU のチャネル 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。

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

割り込みの概要、ITU を使用した割り込みについて詳しくは、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**」(P79)
を参照してください。

9.9.1 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、**0x23** 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\times10^{-3}) \div (325.52\times10^{-9}) - 1 = $**3071** となります。
ITU0_IER	割り込みを許可します。**0x01** 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。 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** よって、**0x09** を設定します。

9.9.2 割り込みプログラム

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

interrupt_timer0 関数が、1ms ごとに割り込みで実行されます。

533 行…割り込みが発生したことにより ITU0_TSR のビット 0 が "1"になりました。次回の割り込みに備えて bit0 を"0"にしておきます。

534 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。

535 行…534 行と同様、cnt1 変数を＋1しています。

9.9.3 「#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。

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

9.9.4 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
```

9.9.5 ベクタアドレスの設定 (src ファイル)

kit06start.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
```

9.9.6 「.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
```

9.10. 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、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。

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

ITU3_GRA	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
	$(16\times10^{-3})\div(325.52\times10^{-9})-1=49151$
	プログラムは、
	`517 :　　ITU3_GRA = PWM_CYCLE;　　/* 周期の設定　　*/`
	としています。PWM_CYCLE は、
	`30 :　#define　　PWM_CYCLE　　49151`
	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。

ITU3_BRB ITU3_GRB	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、**ITU3_BRB へ値を設定すると言うことは、左モータの ON 幅を設定していると言うことになります**。 最初、左モータは停止にしたいので、PWM0%にします。 `518 :　　ITU3_GRB = ITU3_BRB = 0;　　/* 左モータの PWM 設定　　*/`
ITU4_BRA ITU4_GRA	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、**ITU4_BRA へ値を設定すると言うことは、右モータの ON 幅を設定していると言うことになります**。 最初、右モータは停止にしたいので、PWM0%にします。 `519 :　　ITU4_GRA = ITU4_BRA = 0;　　/* 右モータの PWM 設定　　*/`

ITU4_BRB
ITU4_GRB
	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、**ITU4_BRB へ値を設定すると言うことは、サーボの ON 幅を設定していると言うことになります**。
	最初、サーボは0度にしたいので、0 度になるよう PWM 幅を設定します。
	その PWM 幅は、
	`34 : #define SERVO_CENTER 5000 /* サーボのセンタ値 */`
	と定義しています。そのため、設定は、
	`520 : ITU4_GRB = ITU4_BRB = SERVO_CENTER; /* サーボの PWM 設定 */`
	とします。

ITU_TOER	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 bit 7 6 5 4 3 2 1 0 0固定 0固定 PB5 PB4 PB1 PB3 PB2 PB0 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。 bit 7 6 5 4 3 2 1 0 値 0 0 1 1 1 0 0 0 0011 1000→**0x38** を設定します。 `521 : ITU_TOER = 0x38;`
ITU_STR	それぞれの ITU のチャネルで 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 bit 7 6 5 4 3 2 1 0 0固定 0固定 0固定 ITU4 ITU3 ITU2 ITU1 ITU0 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。 bit 7 6 5 4 3 2 1 0 値 0 0 0 0 1 0 0 1 0000 1001 →**0x09** を設定します。 `524 : ITU_STR = 0x09;`

9.11 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。

```
538 :  /*************************************************************************/
539 :  /* タイマ本体                                                            */
540 :  /* 引数    タイマ値 1=1ms                                                */
541 :  /*************************************************************************/
542 :  void timer( unsigned long timer_set )
543 :  {
544 :      cnt0 = 0;
545 :      while( cnt0 < timer_set );
546 :  }
```

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。544 行で cnt0 を 0 にしているので

```
545 :      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、

```
545 :      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、

```
545 :      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、次の行へ行きます。結果、545 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。

9.12 センサ状態読み込み：sensor_inp 関数

センサ基板からの”白”、”黒”情報を読み込む関数です。引数は、センサをマスクする値です。

```
548 :  /*************************************************************************/
549 :  /* センサ状態検出                                                        */
550 :  /* 引数    マスク値                                                      */
551 :  /* 戻り値 センサ値                                                       */
552 :  /*************************************************************************/
553 :  unsigned char sensor_inp( unsigned char mask )
554 :  {
555 :      unsigned char sensor;
556 :
557 :      sensor  = P7DR;
558 :
559 :      /* 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      */
560 :      /* 逆なので、互換を保つためビットを入れ替える                          */
561 :      sensor  = bit_change( sensor );      /* ビット入れ替え                 */
562 :      sensor &= mask;
563 :
564 :      return sensor;
565 :  }
```

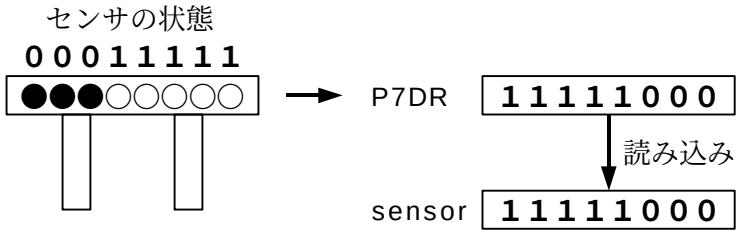
9.12.1 プログラムの解説

```
555 :      unsigned char sensor;
```

入力するセンサ値を保存するためのローカル変数です。ポートの値を保存する変数は、unsigned char 型（符号無し 8 ビット幅）にします。

```
557 :      sensor  = P7DR;
```

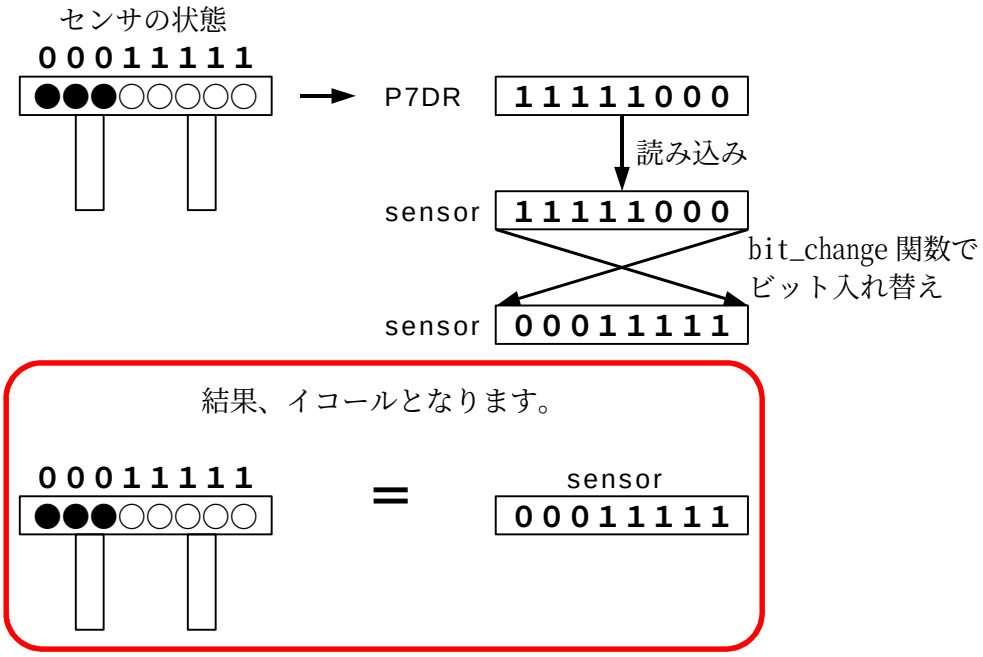
センサの出力は CPU ボードのポート 7 コネクタに接続されていますので、ポート 7 からセンサ値を入力し、sensor 変数に保存します。例えば、センサの状態が「●●●○○○○○」（●：黒(0)、○：白(1) ）なら、sensor=0xf8 となります。



通常、値は左から右へ読みます。しかし、センサ基板の配線の関係で、実際には右から左の方向に読み込まれます。読み込まれた値は「11111000」となります。

```
561 :      sensor  = bit_change( sensor );      /* ビット入れ替え               */
             ↑                    ↑
        変換後の値            変換前の値
```

センサの状態が左から右へ読んで「00011111」であるなら、プログラムで読み込んだ値も「00011111」になったほうが考えやすいのは明白です。そのため、プログラムで左右を入れ替えます。この入れ替える作業を行うのが bit_change 関数です。




```
562 :       sensor &= mask;
```

sensor 値とマスク値を AND 演算して、sensor 値をマスク処理しています。「sensor &= mask;」は、「sensor = sensor & mask」と同じです。

9.12.2 マスク

マスクとは、「覆う」ことです。チェックに不要なビットを覆って”0”にする、それがマスク処理です。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。

1ポートの単位は8ビットのため、**1ビットだけチェックすることはできません**（ビットフィールドという方法を使えばできますが、ここでは無しにします）。**必ず8ビットまとめてのチェックとなります**。
例えば、センサの左端であるビット7が“1”かどうかチェックしたい場合、

```
if( センサの値==0x80 ) {
    /* ビット７が“１”ならこの中を実行 */
}
```

とすればいいように思えます。しかし、ビット6～0がどのような値になっているか分かりません。例えば、ビット7が“1”、ビット0も”1”なら
センサ値＝10000001（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ビット7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ないので、「マスク」という作業をします。マスクというと風邪をひいたときに口元に付けるマスクを連想します。風邪用マスクは風邪の菌をまき散らさないためにつけますが、ここでいうマスクはその見た目をいいます。マスクを付けると口が見えません。隠しています。そうです。ここでいうマスクは『覆い隠す』という意味になります。

では、どのようにマスクするのでしょうか。実際の制御では、強制的に”0”にするだけのことです。
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。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。

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

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。次に、Bが1のときに注目します。

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

Bが1なら、結果はA（センサ値）の値そのものとなります。
実際に置き換えると、Aがセンサの値、Bがマスク値にあたります。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。

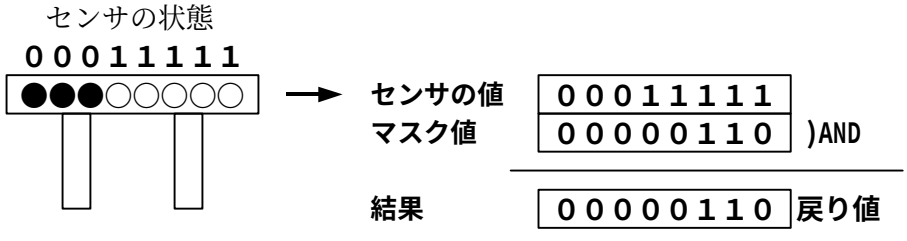
このように、**マスクとは AND 処理して不必要なビットを強制的に"0"にすることです**。
例えば、センサの状態が「黒黒黒白白白白白」のとき、センサ値は「00011111」です。ビット2，1のみチェックに必要で、他は必要なしとします。

bit	7	6	5	4	3	2	1	0
	不要	不要	不要	不要	不要	**必要**	**必要**	不要

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

bit	7	6	5	4	3	2	1	0
マスク値	0	0	0	0	0	**1**	**1**	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、ビット2＝”1”、ビット1＝”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 ) {
    /* ビット２＝”１”、ビット１＝”０”ならこの中を実行 */
}
```

ビット2，1以外はマスクによって強制的に”0”になっていることが分かっているので、安心してビット2，1のみ調べることができます。

9.12.3 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と説明しました。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。

```
 37 :  /* マスク値設定 ×：マスクあり(無効)  ○：マスク無し(有効) */
 38 :  #define        MASK2_2         0x66    /* ×○○××○○×        */
 39 :  #define        MASK2_0         0x60    /* ×○○××××××        */
 40 :  #define        MASK0_2         0x06    /* ××××××○○×        */
 41 :  #define        MASK3_3         0xe7    /* ○○○××○○○        */
 42 :  #define        MASK0_3         0x07    /* ××××××○○○        */
 43 :  #define        MASK3_0         0xe0    /* ○○○××××××        */
 44 :  #define        MASK4_0         0xf0    /* ○○○○××××        */
 45 :  #define        MASK0_4         0x0f    /* ××××○○○○        */
 46 :  #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。
先ほどのビット 2,1 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 ) {
    /* ビット２＝”１”、ビット１＝”０”ならこの中を実行 */
}
```

と書き換えることができます。

9.13 ビットを入れ替える：bit_change 関数

sensor_inp 関数内で使用している、ビットを入れ替える「bit_change 関数」も一緒に説明しておきます。引数「in」は、ビットを入れ替える前の 8 ビット値です。戻り値は、ビットを入れ替えたあとの値です。

```
720 :  /*************************************************************************/
721 :  /* ビット入れ替え                                                        */
722 :  /* 引数    入れ替えする値                                                */
723 :  /* 戻り値  入れ替え後の値                                                */
724 :  /*************************************************************************/
725 :  char unsigned bit_change( char unsigned in )
726 :  {
727 :      unsigned char ret;
728 :      int i;
729 :
730 :      for( i = 0;  i < 8; i++ ) {
731 :          ret >>= 1;                  /* 戻り値の右シフト              */
732 :          ret |= in & 0x80;           /* ret bit7 = in bit7            */
733 :          in  <<= 1;                  /* 引数の左シフト                */
734 :      }
735 :      return ret;
736 :  }
```

for で繰り返す範囲

```
727 :      unsigned char ret;
```

戻り値を保存する変数「ret」です。

```
728 :      int i;
```

作業用変数「i」です。

```
730 :      for( i = 0;  i < 8; i++ ) {
```

i を 0 にして for{ }の中を処理、処理後 i を＋1 して、i<8 なら繰り返しなさい、ということです。i は 0 からスタートするので 0～7 まで 8 回繰り返します。8 回繰り返して 1 バイト分処理します。

```
731 :          ret >>= 1;                  /* 戻り値の右シフト              */
```

戻り値「ret」を 1bit 分右にシフトします。これは、2で割るのと同じです。右にずれることによって無くなる bit0 の値は捨てられ、bit7 には必ず”0”が入ります。

```
732 :          ret |= in & 0x80;           /* ret bit7 = in bit7            */
```

元データ「in」の bit7 の値と「ret」の bit7 を OR 演算します。ret の bit7 は必ず”0”のため、in の bit7 の値が ret の bit7 へ転送されることになります。

```
733 :          in  <<= 1;                  /* 引数の左シフト                */
```

元データ「in」を 1bit 分左にシフトします。これは、2 倍するのと同じです。

```
734 :      }
```

for のカッコ閉じです。731 行～733 行目を繰り返します。

例えば、0xf8 を左右入れ替えしたい場合、

```
c = bit_change( 0xf8 );
```

と関数を実行すると、変数 c には 0x1f が代入されます。計算過程は下図のようなイメージになります。

i	in	ret	説　明
0		０００００００００ →	ret を右にシフトするが最初は 0 なので何も変化なし
	１１１１１０００	１００００００００	in の bit7 と ret の bit7 を OR 演算する
	１１１１０００００ ←		in を 1bit 分左にシフト、bit0 には”0”が入る
1		０１０００００００ →	ret を右にシフト、先ほどの”1”が bit6 へ移動、bit7 には”0”が挿入される
	１１１１０００００	１１０００００００	in の bit7 と ret の bit7 を OR 演算する
	１１１００００００ ←		in を 1bit 分左にシフト、bit0 には”0”が入る
2		０１１００００００ →	ret を右にシフト、先ほどの”1”が bit6 へ移動、bit7 には”0”が挿入される
	１１１００００００	１１１００００００	in の bit7 と ret の bit7 を OR 演算する
	１１０００００００ ←		in を 1bit 分左にシフト、bit0 には”0”が入る
3		０１１１００００００ →	ret を右にシフト、先ほどの”1”が bit6 へ移動、bit7 には”0”が挿入される
	１１０００００００	１１１１００００００	in の bit7 と ret の bit7 を OR 演算する
	１００００００００ ←		in を 1bit 分左にシフト、bit0 には”0”が入る
⋮	⋮	⋮	
7		０００１１１１１１ →	ret を右にシフト、先ほどの”0”が bit6 目へ移動、bit7 には”0”が挿入される
	０００００００００	０００１１１１１１	in の bit7 と ret の bit7 を OR 演算する 0 なので何もならない
	０００００００００ ←		in を 1bit 分左にシフト、bit0 は”0”が入る
8		０００１１１１１１	終了、ビット入れ替え値を戻り値とする

ビット入れ替えが完了して、変数 c には戻り値の「0x1f」が代入されます。

9.14 クロスライン検出処理：check_crossline 関数

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。

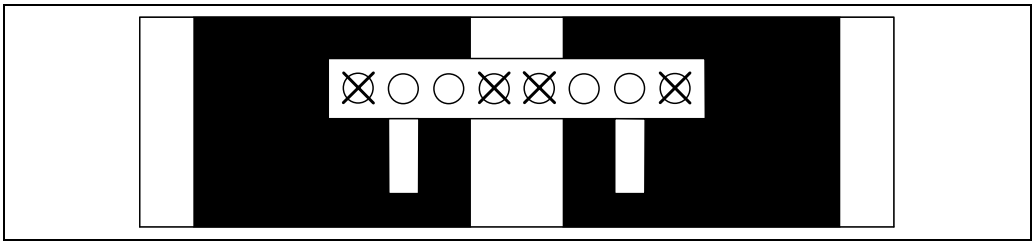
```
567 :  /************************************************************************/
568 :  /* クロスライン検出処理                                                 */
569 :  /* 戻り値 0:クロスラインなし 1:あり                                     */
570 :  /************************************************************************/
571 :  int check_crossline( void )
572 :  {
573 :      unsigned char b;
574 :      int ret;
575 :
576 :      ret = 0;
577 :      b = sensor_inp(MASK2_2);
578 :      if( b==0x66 || b==0x64 || b==0x26 || b==0x62 || b==0x46 ) {
579 :          ret = 1;
580 :      }
581 :      return ret;
582 :  }
```

```
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(MASK2_2);
```

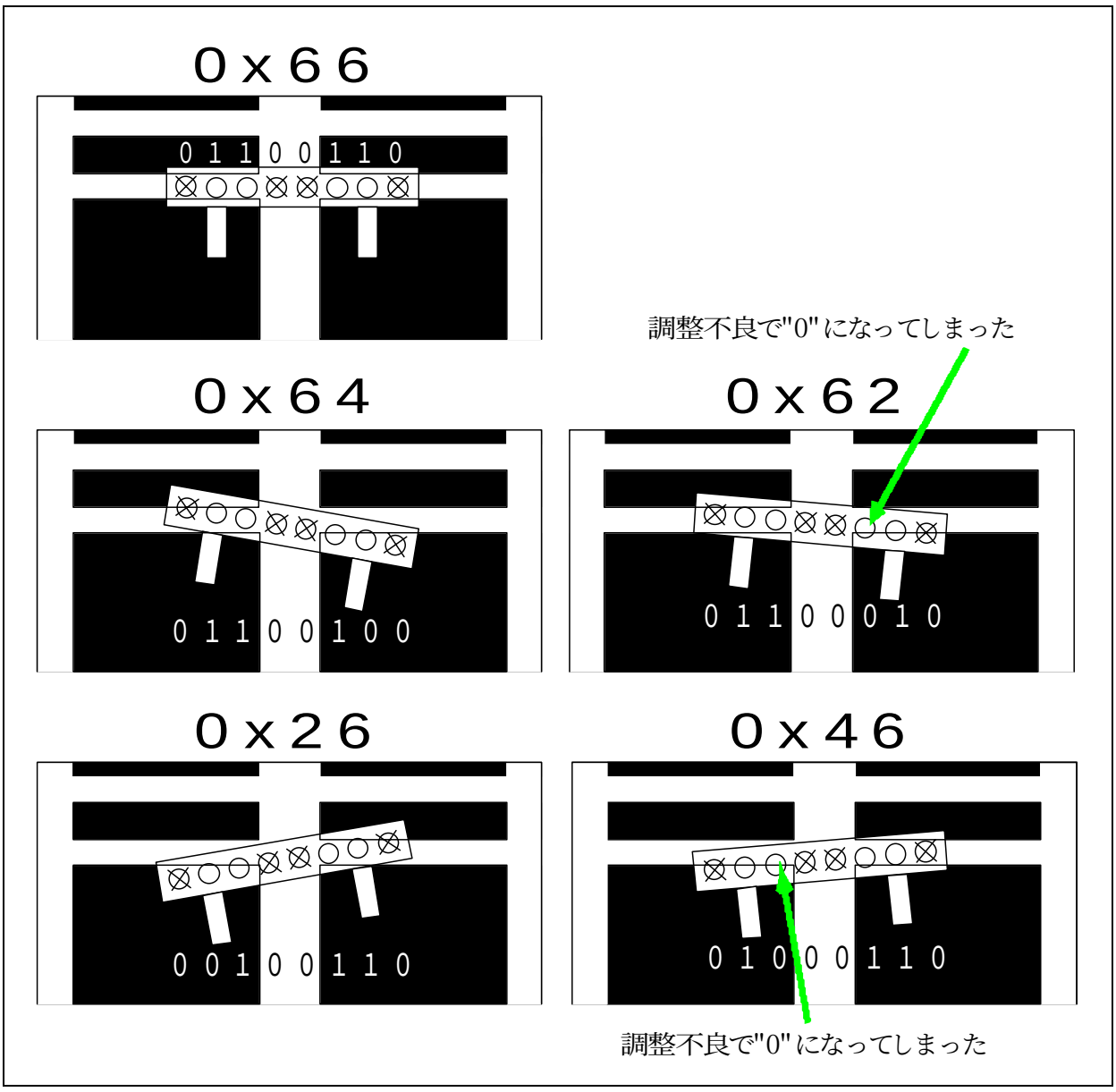
センサを読み込み、変数 b へ保存します。センサのマスク値は、「MASK2_2＝0x66」なので、



のように、左中2つと右中2つの4つのセンサを読み込みます。

```
578 :      if( b==0x66 || b==0x64 || b==0x26 || b==0x62 || b==0x46 ) {
```

センサの状態をチェックします。センサが、0x66 か 0x64 か 0x26 か 0x62 か 0x46 のどれかなら if の条件が成立、どれでもないなら不成立となります。それぞれの状態を図解すると次図のようになります。



0x66 は真っ正面から進入して4つとも"1"となったとき、これは文句なくクロスラインです。0x64、0x26 は左右のどちらからか斜めに進入した場合、3 つ以上"1"となっているのでクロスラインと判断します。0x62 は、図を見ただけでは 0x64 の状態です。センサはボリュームで感度調整を行うことができます。すべてのセンサが同時に点くように調整していれば問題ないのですが、調整不足で右から 2 つ目のセンサの感度が弱かった場合、センサの状態は 0x64 ではなく、0x62 と判断します。しかし、図のようにクロスラインと判断しても問題ありませんので、0x62 も含めます。0x46 も同様です。

9.15 右ハーフライン検出処理：check_rightline 関数

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。

```
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==0x0f || b==0x1f ) {
596 :          ret = 1;
597 :      }
598 :      return ret;
599 :  }
```

```
593 :      ret = 0;
```

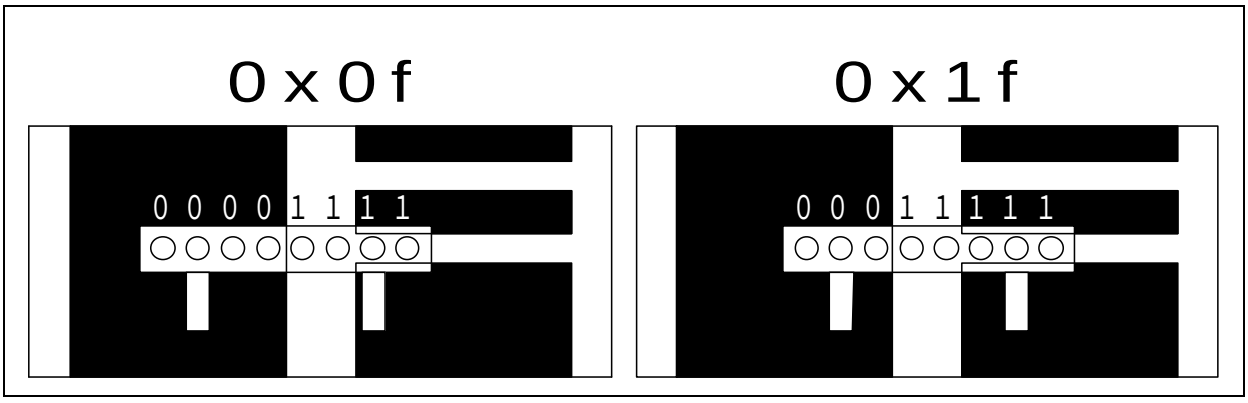
戻り値を保存する ret 変数を初期化しています。右ハーフラインと判断すると 1、違うと判断すると 0 をこの変数に入れます。今のところどちらか分からないので、とりあえず違うと判断して 0 を入れておきます。

```
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 つのセンサすべて読み込みます。

```
595 :      if( b==0x0f || b==0x1f ) {
```

センサの状態をチェックします。センサが、0x0f か 0x1f なら if の条件が成立、どれでもないなら不成立となります。それぞれの状態を図解すると次図のようになります。



右 4 つ、または 5 つが"1"なら右ハーフラインと判断します。

9.16 左ハーフライン検出処理：check_leftline 関数

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==0xf0 || b==0xf8 ) {
613 :          ret = 1;
614 :      }
615 :      return ret;
616 :  }
```

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

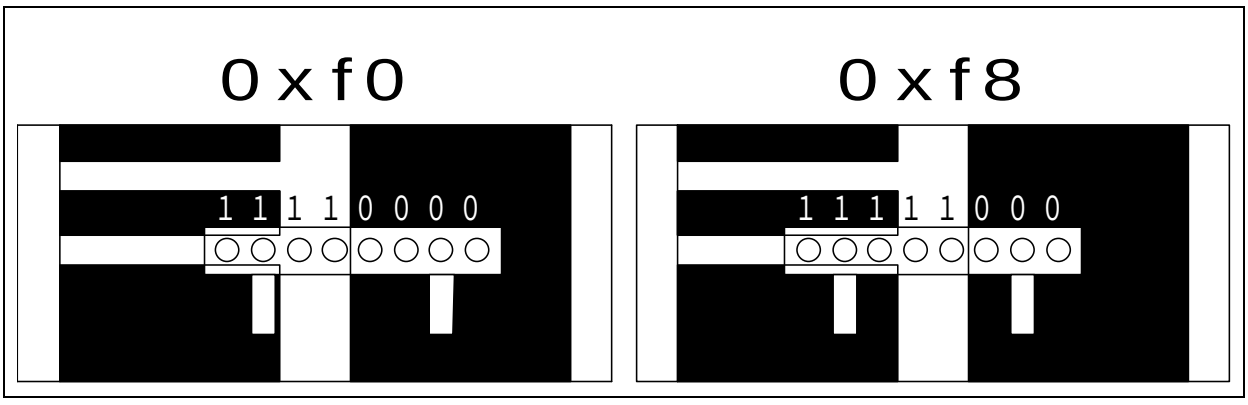
戻り値を保存する ret 変数を初期化しています。左ハーフラインと判断すると 1、違うと判断すると 0 をこの変数に入れます。今のところどちらか分からないので、とりあえず違うと判断して 0 を入れておきます。

```
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 つのセンサすべて読み込みます。

```
612 :      if( b==0xf0 || b==0xf8 ) {
```

センサの状態をチェックします。センサが、0xf0 か 0xf8 なら if の条件が成立、どれでもないなら不成立となります。それぞれの状態を図解すると次図のようになります。



左 4 つ、または 5 つが"1"なら左ハーフラインと判断します。

9.17 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、「~」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。

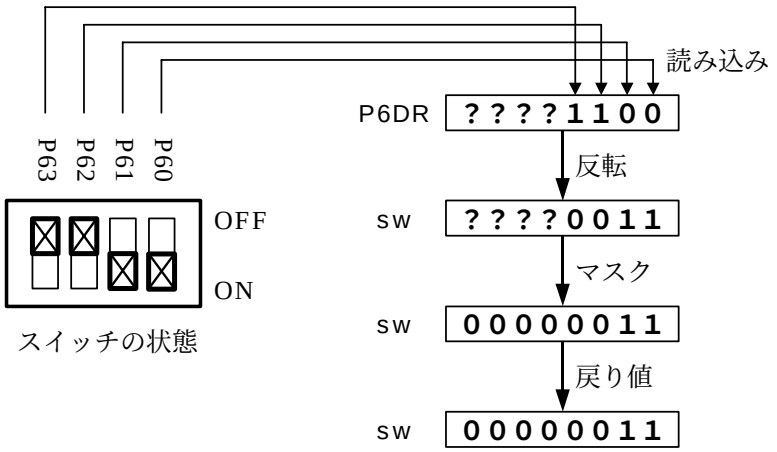
```
627 :      sw &= 0x0f;
```

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

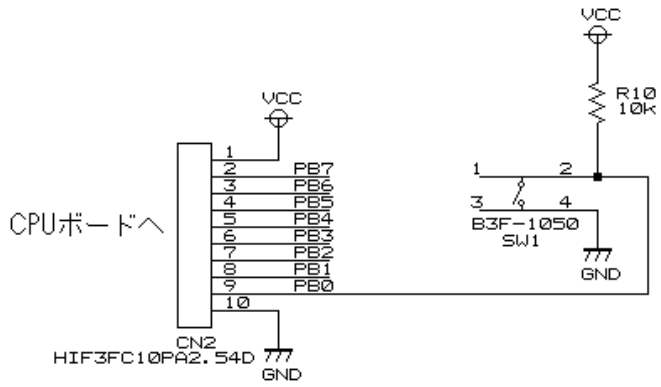
```
        c = dipsw_get();
```

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。



9.18 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。



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 :  }
```

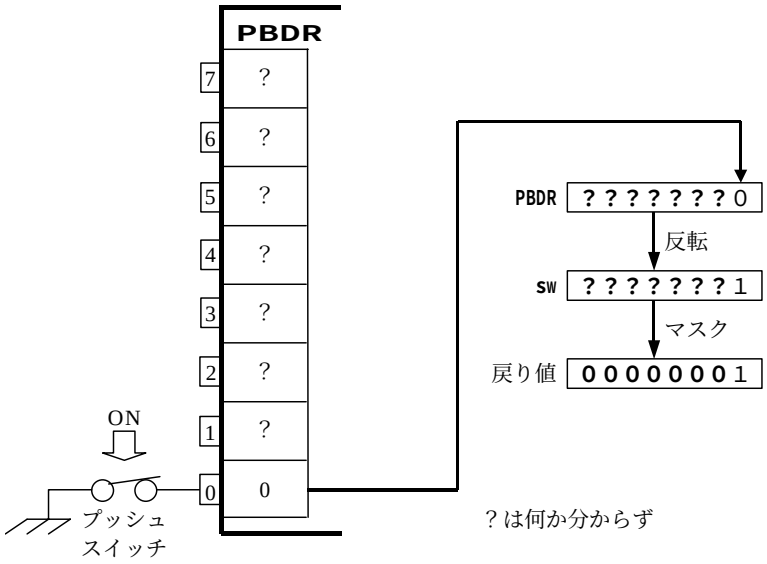
```
640 :      sw  = ~PBDR;                        /* スイッチのあるポート読み込み */
```

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。



9.19 スタートバー検出センサ読み込み：startbar_get 関数

スタートバー検出センサ基板からの信号を、読み込む関数です。スタートバー検出センサの信号は、ポート A の bit3 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  = ~PADR;                         /* スタートバー信号読み込み   */
655 :      b &= 0x08;
656 :      b >>= 3;
657 :
658 :      return  b;
659 :  }
```

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

スタートバー検出センサ基板と接続されているポート A 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。

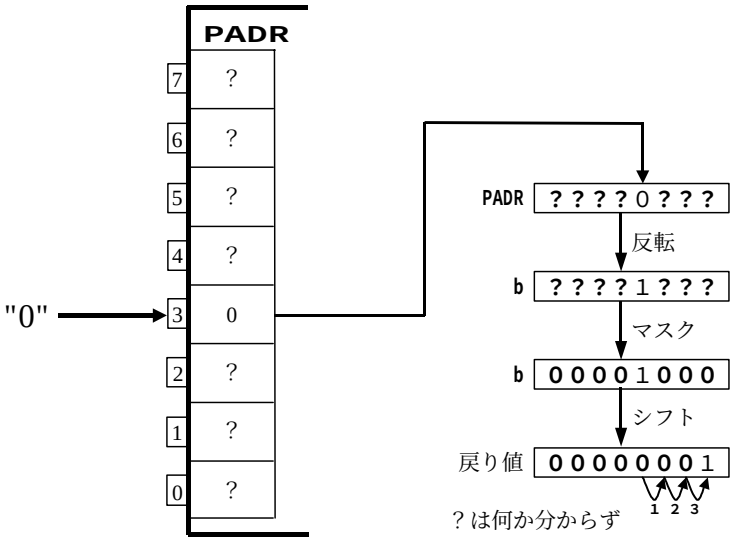
```
655 :      b &= 0x08;
```

スタートバー検出センサの信号は、bit3 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'0000 1000'、16 進数で 0x08 になります。

```
656 :      b >>= 3;
```

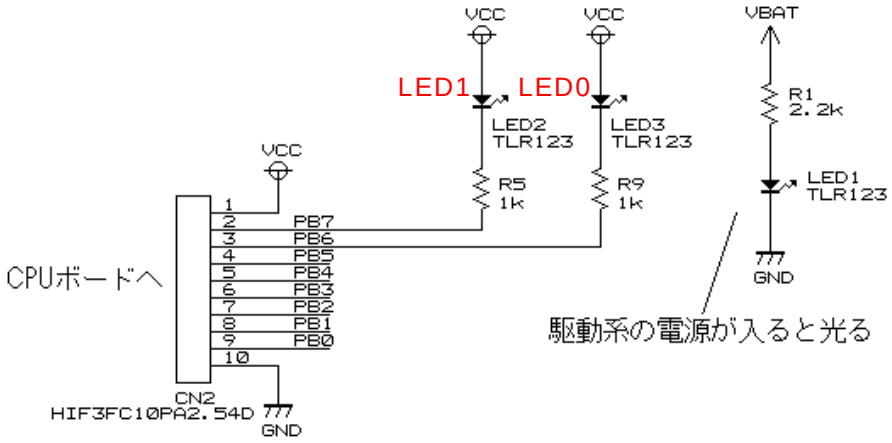
bit3 にある信号値を bit0 に移動させるため、3 ビット右へ移動します。結果、スタートバーありで 1、なしで 0 になります。

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。



9.20 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	0 0	消灯	消灯
1	0 1	消灯	点灯
2	1 0	点灯	消灯
3	1 1	点灯	点灯

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;
```

引数はビット 1 と 0 の値ですが、実際の LED はビット7と6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。

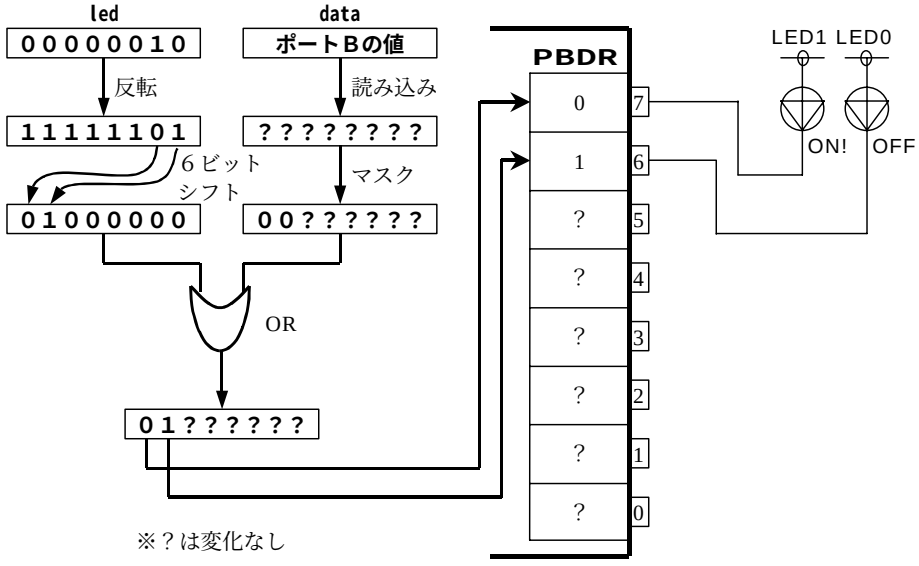
```
673 :      PBDR = data | led;
```

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。

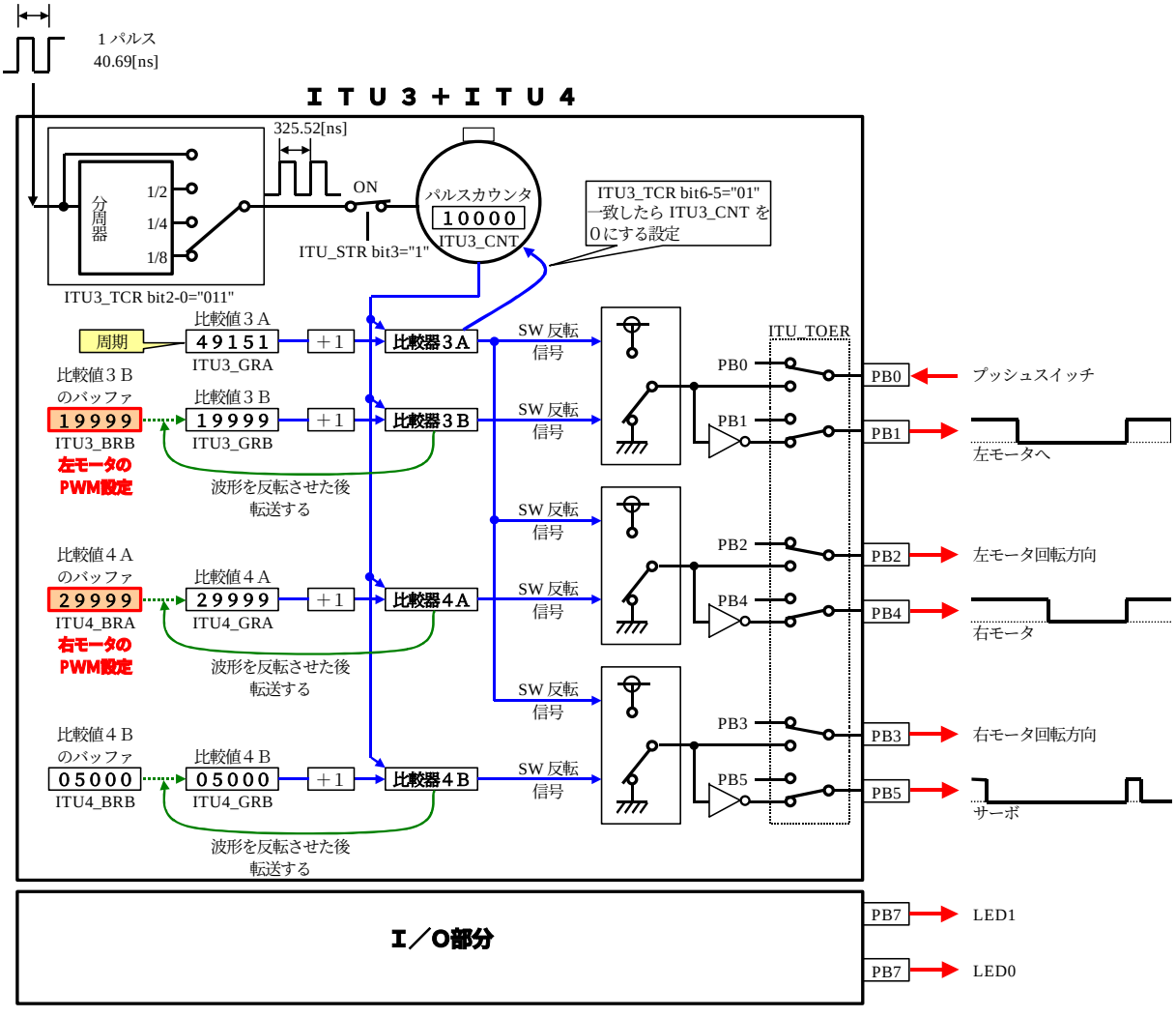


9.21 モータ速度制御：speed 関数

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;          /* ディ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;          /* ディップスイッチ読み込み */
```

dipsw_get 関数は、CPU ボードのディップスイッチの値が返ってきます。0～15 です。＋5 していますのでディ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。

$$\text{speed\_max} = \underset{①}{\text{(unsigned long)}} \quad \underset{②}{\text{100\%のときの値}} \times \underset{③}{\frac{\text{sw\_data}(5\sim20)}{20}}$$

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_l >= 0 ) {                    0以上かチェック
691 :           PBDR &= 0xfb;                        正転にする
692 :           ITU3_BRB = speed_max * accele_l / 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_l 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ビット2を”0”にします。

ビット	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	→0xfb
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。

ITU3_BRB ＝（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）×（speed 関数の%）÷100
＝ speed_max × accele_l ÷100

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ビット2を”1”にします。ビット2のみを"1"にするには OR 演算を使います。

ビット	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	→0x04
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ビット3を”0”にします。

ビット	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	→0xf7
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。

ITU4_BRA ＝（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）×（speed 関数の%）÷100
＝ speed_max × accele_r ÷100

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ビット3を”1”にします。ビット3のみを"1"にするには OR 演算を使います。

ビット	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	→０ｘ０８
PBDR 変更後	※	※	※	※	1	※	※	※	

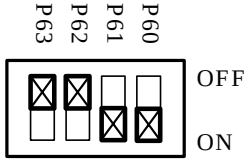
※＝変化無し

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**



スイッチの状態

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%

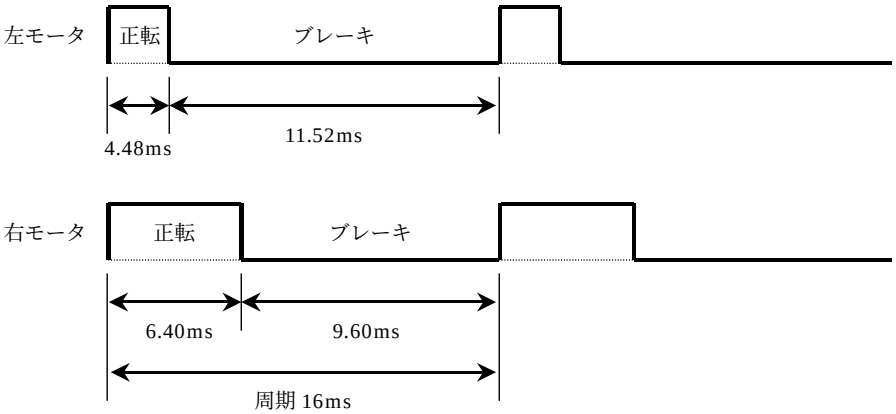
となります。これは、

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のみです。フリー（開放）状態はありません。**

9.22 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 度程度だと思います。

```
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	
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。

9.23 メインプログラム

9.23.1 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。

```
 72 :  /*****************************************************************************/
 73 :  /* メインプログラム                                                          */
 74 :  /*****************************************************************************/
 75 :  void main( void )
 76 :  {
 77 :      int     i;
 78 :
 79 :      /* マイコン機能の初期化 */
 80 :      init();                             /* 初期化                   */
 81 :      set_ccr( 0x00 );                    /* 全体割り込み許可         */
 82 :
 83 :      /* マイコンカーの状態初期化 */
 84 :      handle( 0 );
 85 :      speed( 0, 0 );
```

```
 80 :      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。

```
 81 :      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。

```
 83 :      /* マイコンカーの状態初期化 */
 84 :      handle( 0 );
 85 :      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。

9.23.2 パターン方式について

kit06.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。

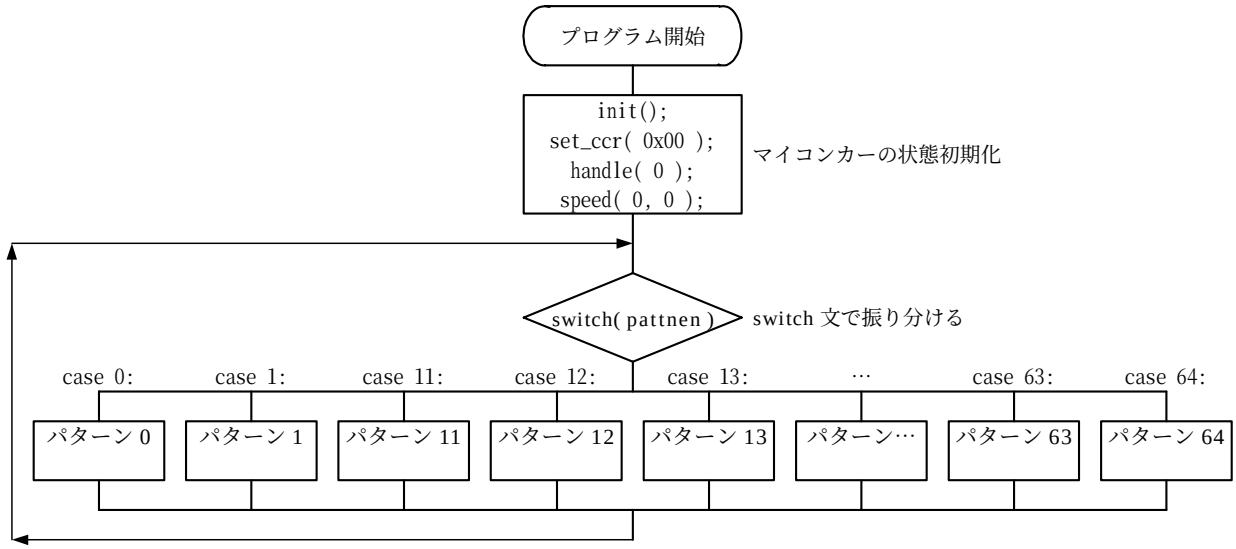
次に、pattern という変数を作ります。この変数に設定した値により、どのプログラムを実行するか選択します。

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。

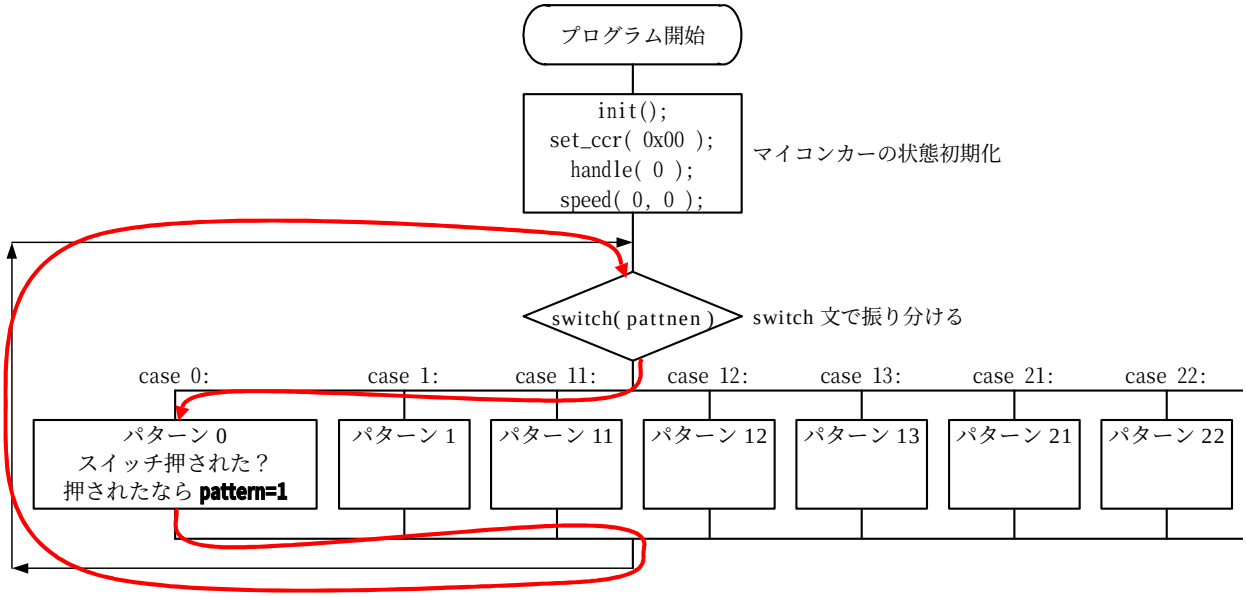
この方式を使うと、パターンごとに処理を分けられるため、プログラムが見やすくなります。パターン方式は、**「プログラムのブロック化」**と言うこともできます。

9.23.3 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。

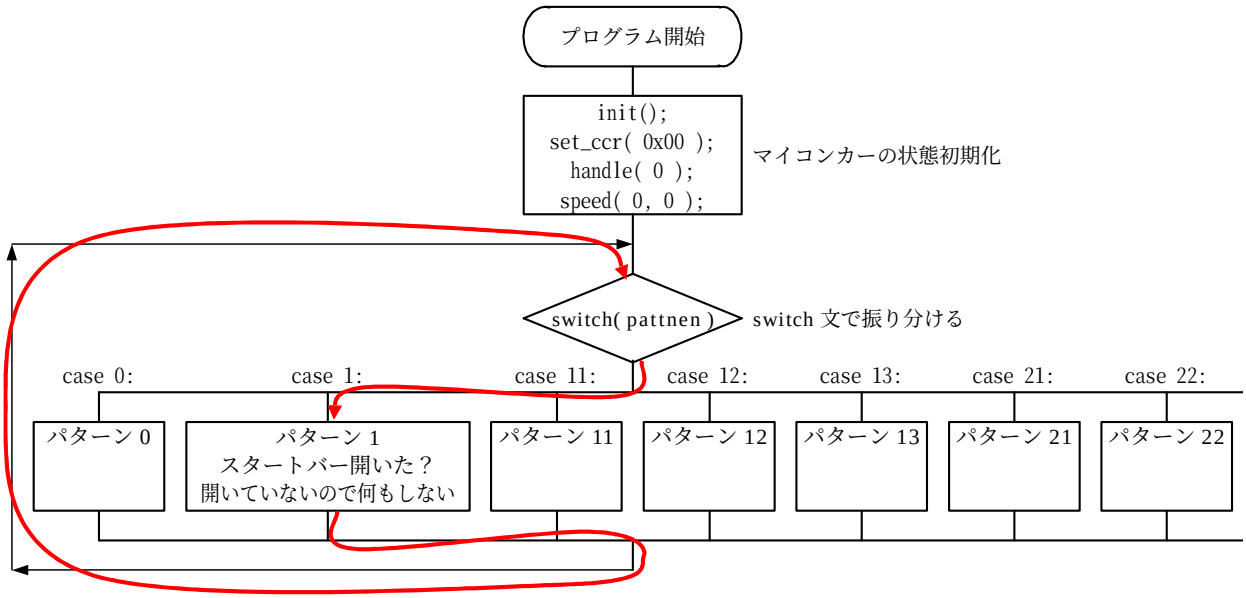


起動時、pattern 変数は 0 です。switch 文により case 0 部分のプログラム「パターン 0 プログラム」を実行し続けます。後ほど説明しますが、パターン 0 はスイッチ入力待ちです。スイッチが押されると、「pattern=1」が実行されます。下図のようです。



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。

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

pattern と各 case 文の定数を比較して、一致したらその case 位置にジャンプして処理を行う。

その処理は、break 文に出会うか switch 文の終端に出会うと終了する。

どの case 文の定数にも当てはまらない場合は、default 文を実行。ちなみに、default 文も無い場合は何も実行しない。

9.23.4 パターンの内容

kit06.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。

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	**スタートバーが開いたかチェック**	
11	**通常トレース**	11～20 は、通常走行中の処理を行っています
12	**右へ大曲げの終わりのチェック**	
13	**左へ大曲げの終わりのチェック**	
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	**1本目の右ハーフライン検出時の処理**	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	**2本目の右ハーフラインを読み飛ばす**	
53	**右ハーフライン後のトレース**	
54	**右レーンチェンジ終了のチェック**	
61	**1本目の左ハーフライン検出時の処理**	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	**2本目の左ハーフラインを読み飛ばす**	
63	**左ハーフライン後のトレース**	
64	**左レーンチェンジ終了のチェック**	

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。

太字部分が、kit05.c から kit06.c に変更するに当たって追加、または変更した部分です。

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	内容
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	・中心線がセンサの中心に来たならパターン 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	・中心線がセンサの中心に来たならパターン 11 へ
	状態	内容

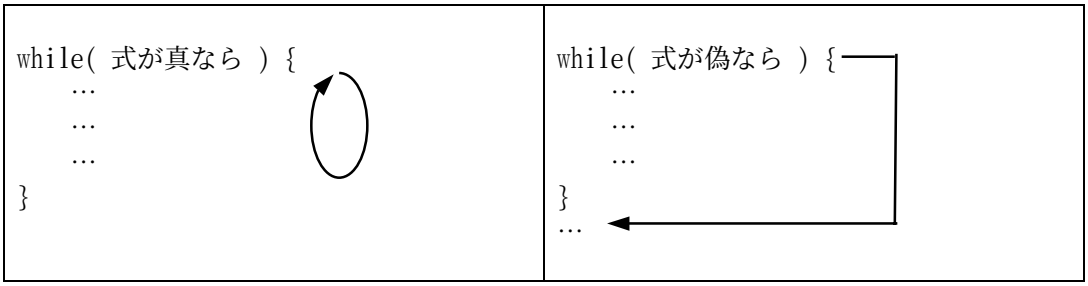
9.23.5 パターン方式の最初 while、switch 部分

```
 87 :      while( 1 ) {
 88 :      switch( pattern ) {
114 :      case 0:
               パターン0時の処理
128 :          break;
129 :
130 :      case 1:
               パターン1時の処理
146 :          break;
                                                          対      対
           それぞれのパターン処理

482 :      default:
483 :          /* どれでもない場合は待機状態に戻す */
484 :          pattern = 0;
485 :          break;
486 :      }
487 :      }
```

87 行の「while(1) {」と 487 行の「}」が対、88 行の「switch(pattan) {」と 486 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、88～486 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だけ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ばいいのです。逆にマイコンは、適切に終了処理をせずにプログラムを終わらせてしまうと、プログラムが書かれていない領域にまで行ってしまい暴走してしまいます。マ

イコンは、何も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。

9.23.6 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。

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

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。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**、117～119 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。

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

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。kit06.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。

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

128 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マになると 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**。

9.23.7 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。

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

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。

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

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。kit06.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。

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

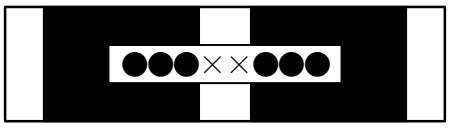
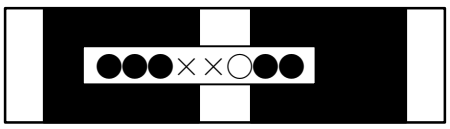
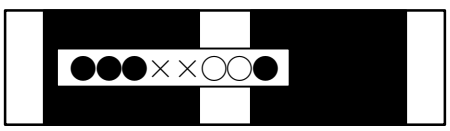
9.23.8 パターン11：通常トレース

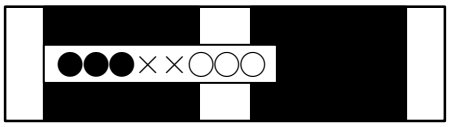
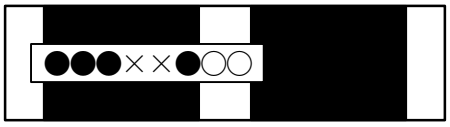
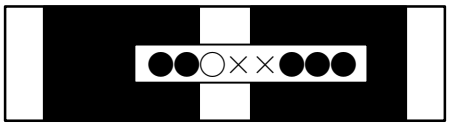
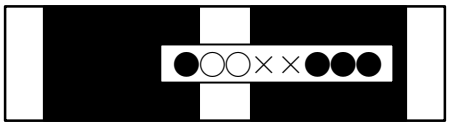
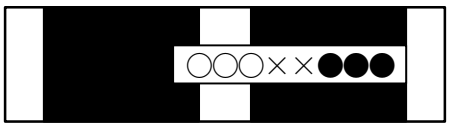
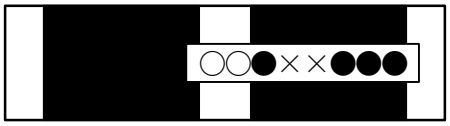
パターン 11 は、コース上をトレースする状態です。

まず、想定されるセンサの状態を考えます。センサは 8 つありますが、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つ、左3つ、合計6つのセンサでコースの状態を検出するようにします。

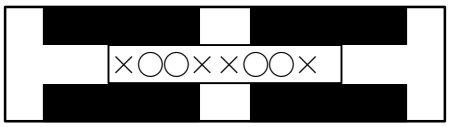
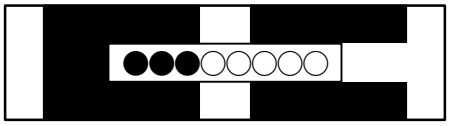
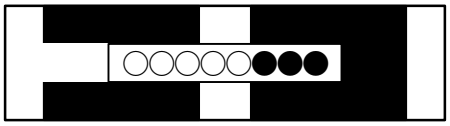
次に、そのときのハンドル角度と左モータ、右モータの PWM 値を考えます。センサが中心のときはスピードを上げます。センサが中心よりずれればずれるほど、ハンドルを大きく曲げてスピードを落とします。

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	69

4		00000111	0x07	15	50	40
5		00000011	0x03	25	30	21
6		00100000	0x20	-5	100	100
7		01100000	0x60	-10	69	80
8		11100000	0xe0	-15	40	50
9		11000000	0xc0	-25	21	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4つ使用	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つ使用	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つ使用	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

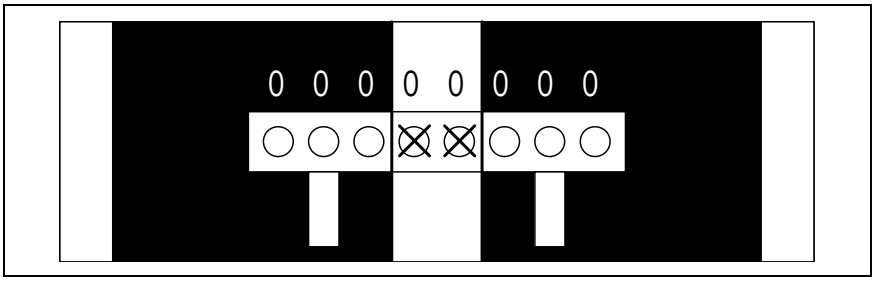
```
162 :          switch( sensor_inp(MASK3_3) ) {
```

センサの状態を読み込みます。右3つ、左3つのセンサを読み込むので MASK3_3 を使います。

(2) 直進

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

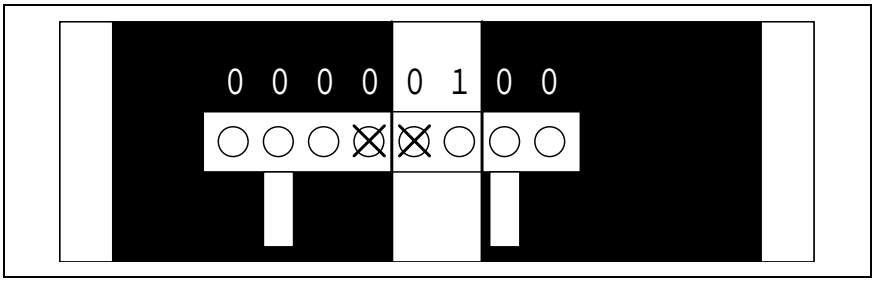
センサが「0x00」の状態です。この状態は下図のように、まっすぐ進んでいる状態です。サーボ角度 0 度、左モータ 100%、右モータ 100%で進みます。



(3) 左寄り

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

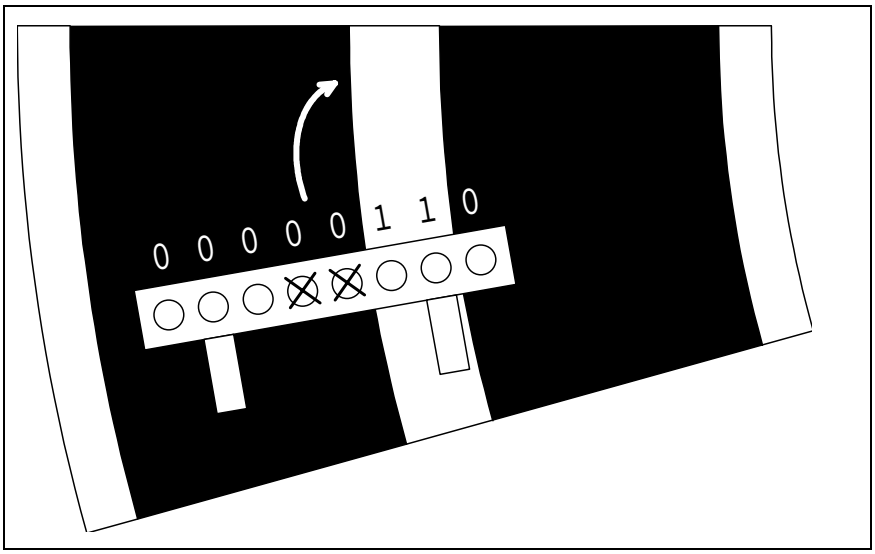
センサが「0x04」の状態です。この状態は下図のように、マイコンカーが微妙に左に寄っている状態です。サーボを右に 5 度、左モータ 100%、右モータ 100%で進み、中心に寄るようにします。



(4) 少し左寄り

```
175 :                case 0x06:
176 :                    /* 少し左寄り→右へ小曲げ */
177 :                    handle( 10 );
178 :                    speed( 80 ,69 );
179 :                    break;
```

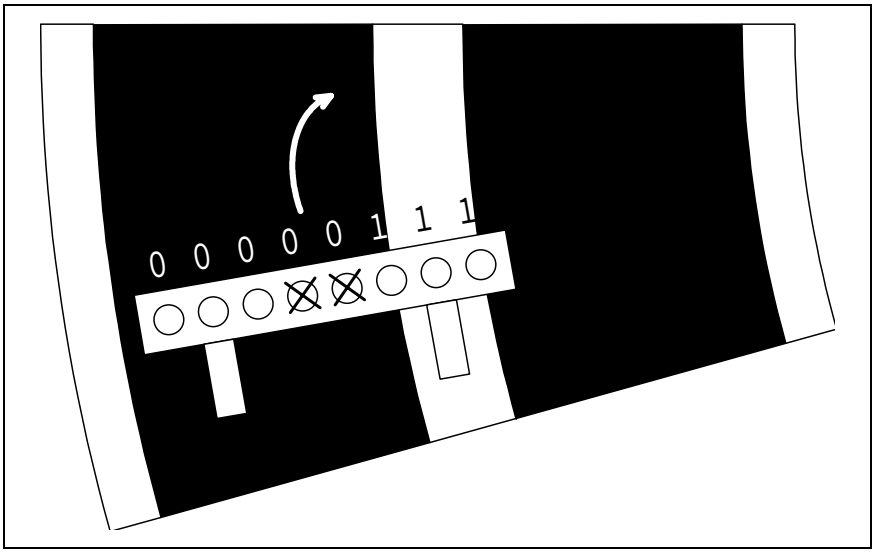
センサが「0x06」の状態です。この状態は下図のように、マイコンカーが少し左に寄っている状態です。サーボを右に 10 度、左モータ 80%、右モータ 69%で進み、減速しながら中心に寄るようにします。



(5) 中くらい左寄り

```
181 :                case 0x07:
182 :                    /* 中くらい左寄り→右へ中曲げ */
183 :                    handle( 15 );
184 :                    speed( 50 ,40 );
185 :                    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 40%で進み、減速しながら中心に寄るようにします。

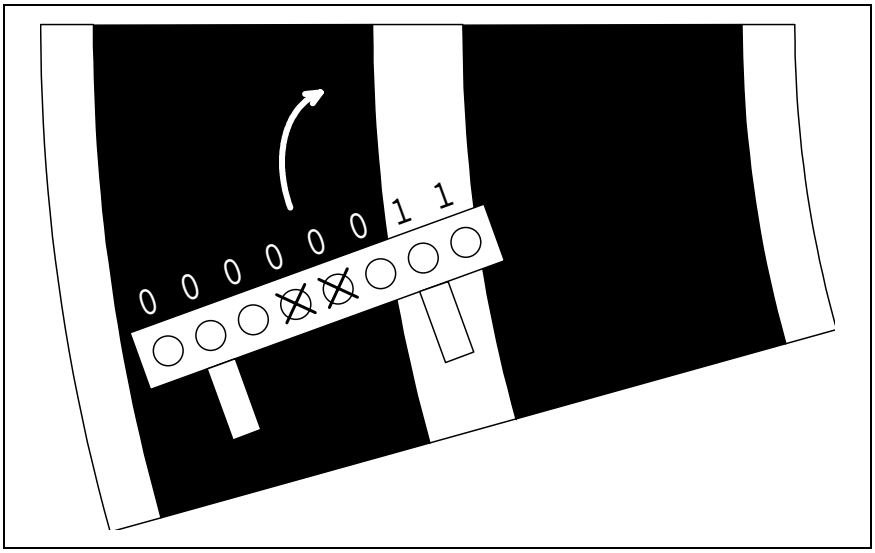


(6) 大きく左寄り

```
187 :                case 0x03:
188 :                    /* 大きく左寄り→右へ大曲げ */
189 :                    handle( 25 );
190 :                    speed( 30 ,21 );
192 :                    break;
```

※本当のプログラムは 191 行が入っています。後述します。

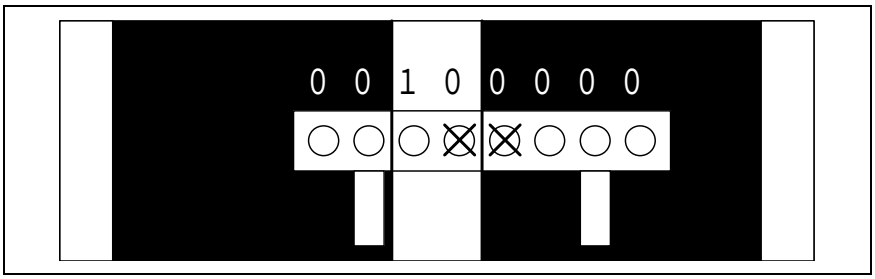
センサが「0x03」の状態です。この状態は下図のように、マイコンカーが大きく左に寄っている状態です。サーボを右に 25 度、左モータ 30%、右モータ 21%で進み、かなり減速しながら中心に寄るようにします。



(7) 微妙に右寄り

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

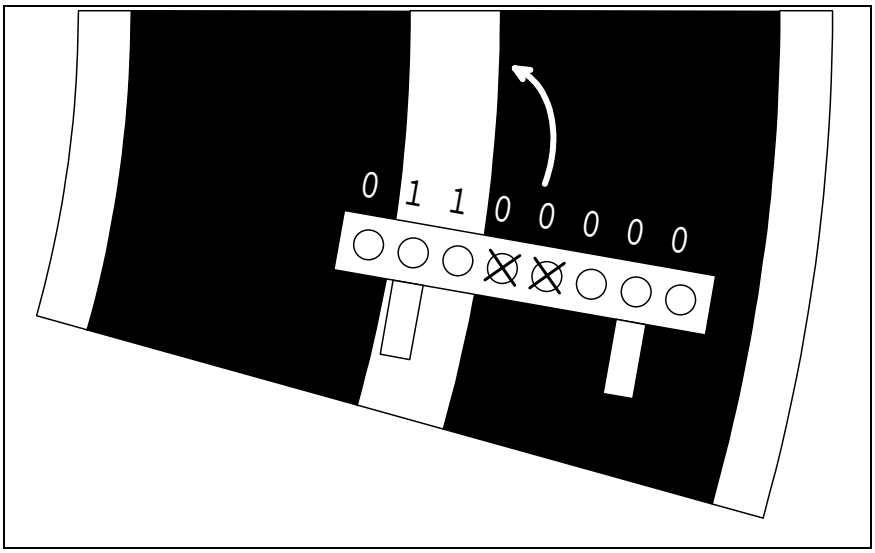
センサが「0x20」の状態です。この状態は下図のように、マイコンカーが微妙に右に寄っている状態です。サーボを左に 5 度、左モータ 100%、右モータ 100%で進み、中心に寄るようにします。



(8) 少し右寄り

```
200 :               case 0x60:
201 :                   /* 少し右寄り→左へ小曲げ */
202 :                   handle( -10 );
203 :                   speed( 69 ,80 );
204 :                   break;
```

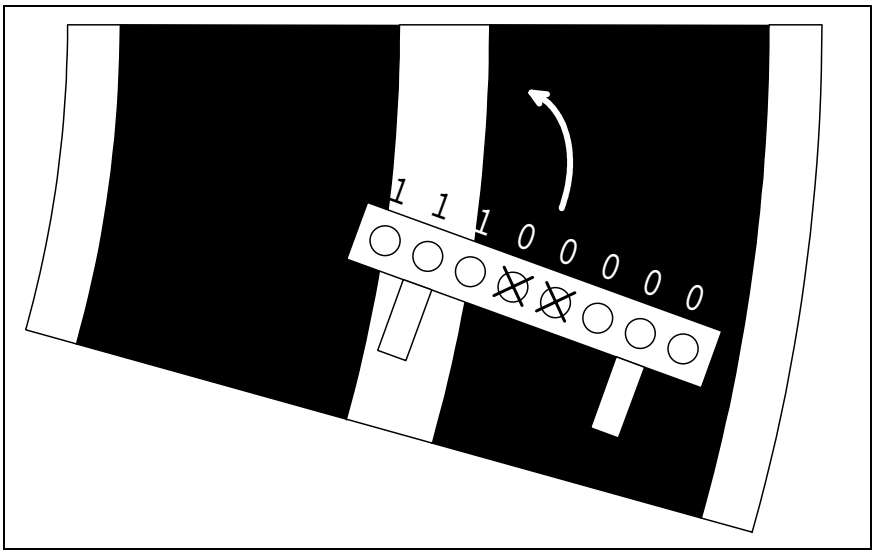
センサが「0x60」の状態です。この状態は下図のように、マイコンカーが少し右に寄っている状態です。サーボを左に 10 度、左モータ 69%、右モータ 80%で進み、減速しながら中心に寄るようにします。



(9) 中くらい右寄り

```
206 :               case 0xe0:
207 :                   /* 中くらい右寄り→左へ中曲げ */
208 :                   handle( -15 );
209 :                   speed( 40 ,50 );
210 :                   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 40%、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。

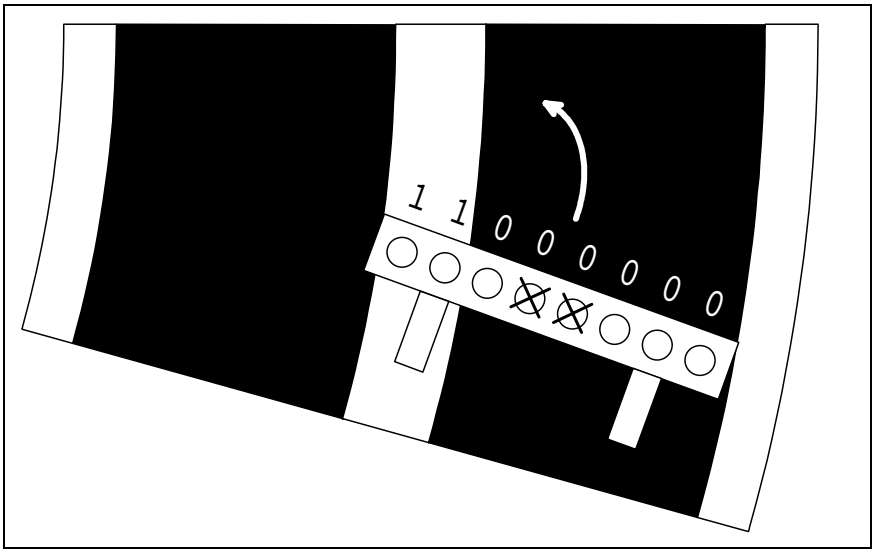


(10) 大きく右寄り

```
212 :               case 0xc0:
213 :                   /* 大きく右寄り→左へ大曲げ */
214 :                   handle( -25 );
215 :                   speed( 21 ,30 );
217 :                   break;
```

※本当のプログラムは 216 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 21%、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

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

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

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

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

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

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

```
219 :              default:
220 :                  break;
```

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

(15) break 文で抜ける位置

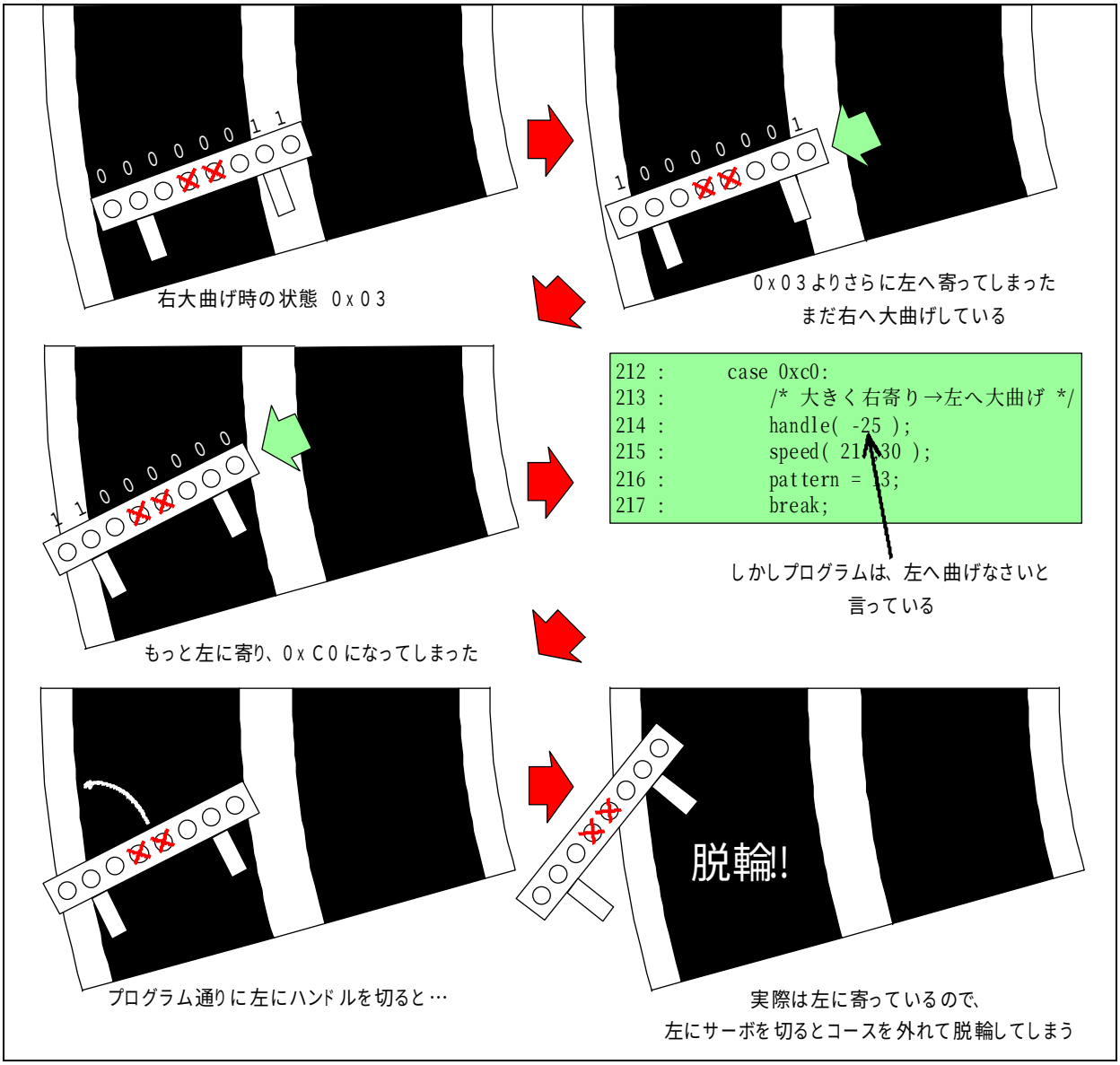
break 文は、switch 文または for、while、do～while のループから脱出させるための文です。**多重ループの中で用いられると、その break 文の存在するループを<u>ひとつだけ打ち切り</u>、すぐ外側のループに処理を移します。**「**<u>ひとつだけ打ち切り</u>**」が重要です。

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 を抜けるのではない!!
        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
      2}
        break; ←――――――――――――――――――――― switch( pattern )を抜ける break、1へ

        中略

  1}
    }
```

9.23.9 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。



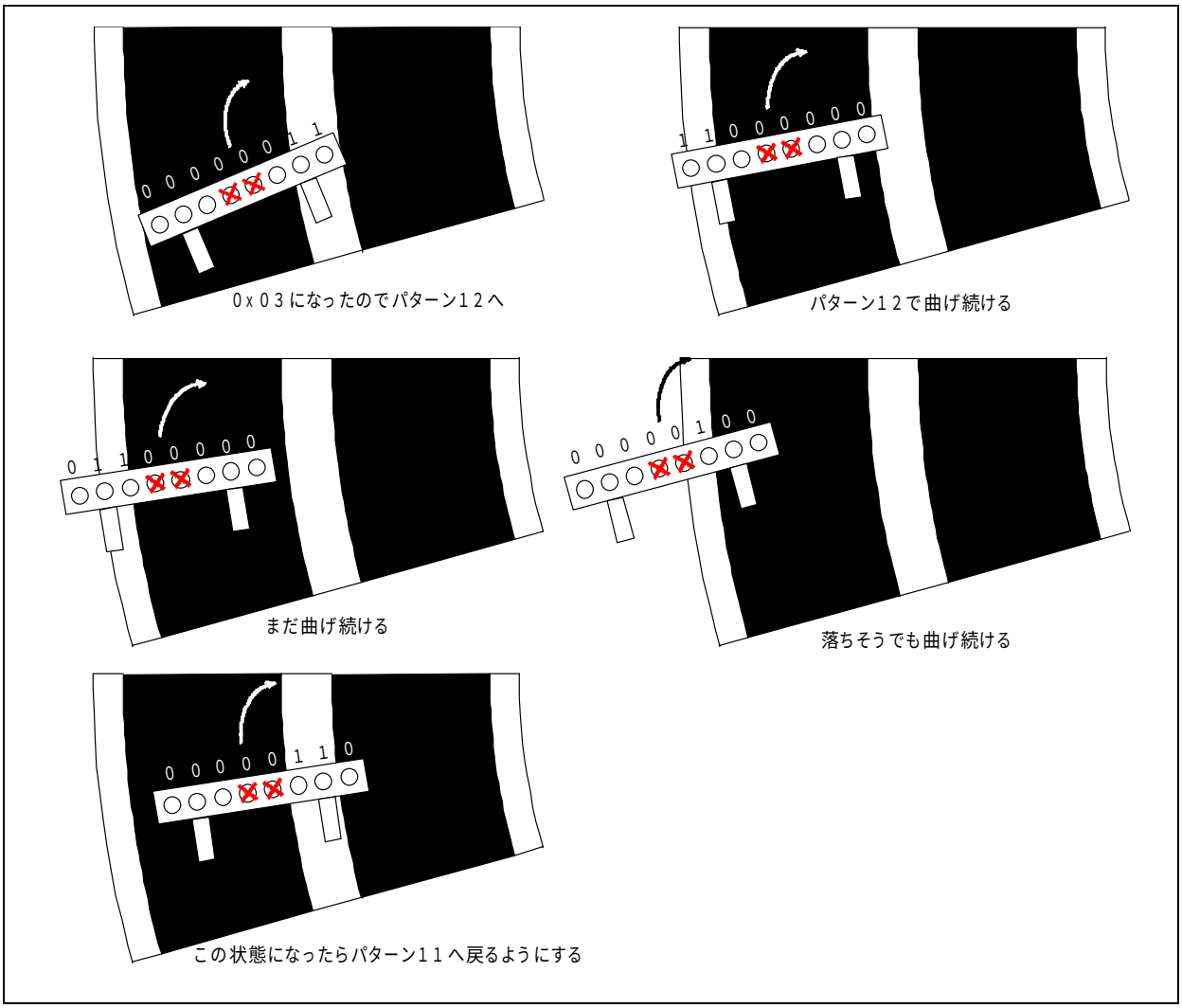
本当は左に大きく寄っている場合でもプログラムでは、「右に大きく寄っている」と誤った判断をすることがあります。もちろん、誤った判断をするとサーボを逆に切るのですぐさま脱輪します。

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

```
187 :                case 0x03:
188 :                    /* 大きく左寄り→右へ大曲げ */
189 :                    handle( 25 );
190 :                    speed( 30 ,21 );
191 :                    pattern = 12;     ←追加　パターン１２へ移る
192 :                    break;
```

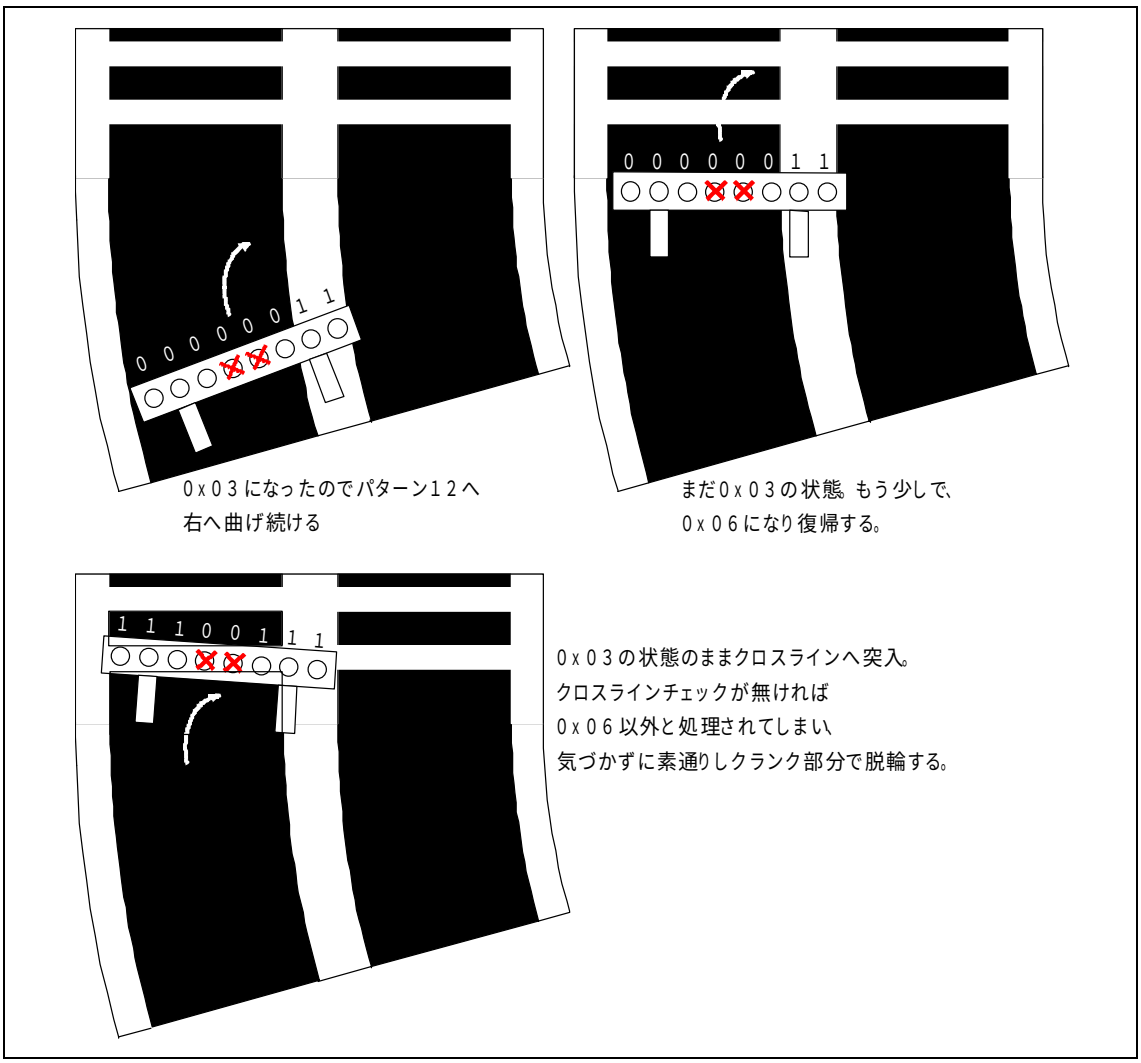
センサが 0x03 になるとパターン 12 へ移ります。
パターン 12 では、どのようになったら通常走行のパターン 11 へ戻るか考えてみます。



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。



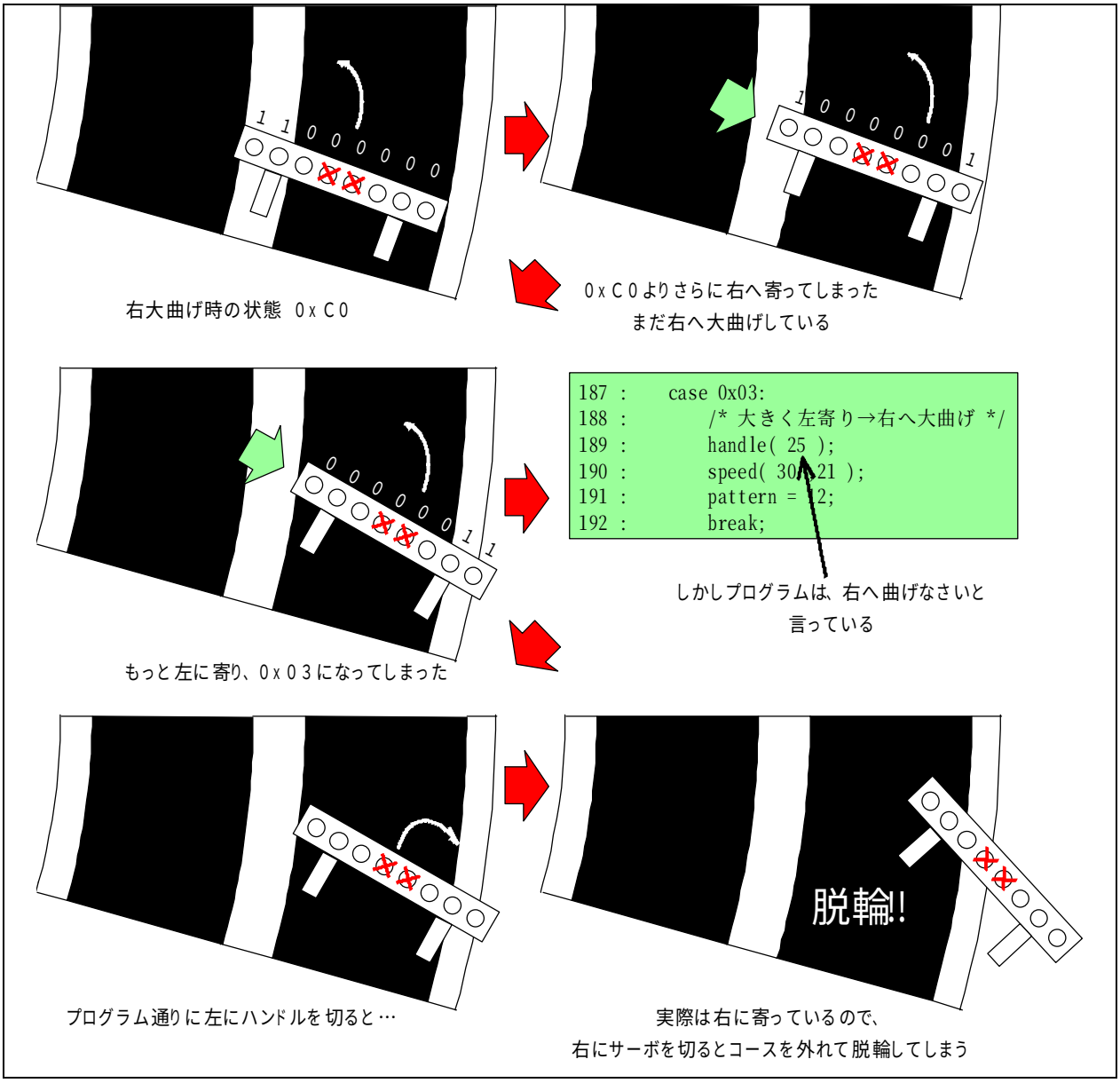
このようにパターン 12 を処理中でも、クロスラインを検出することがありそうです。同様に右ハーフライン、左ハーフラインもあり得ます。そこで、パターン 12 にも 3 種類のチェックを追加します。

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

パターン 12 のプログラムはこれで完成です。

9.23.10 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。



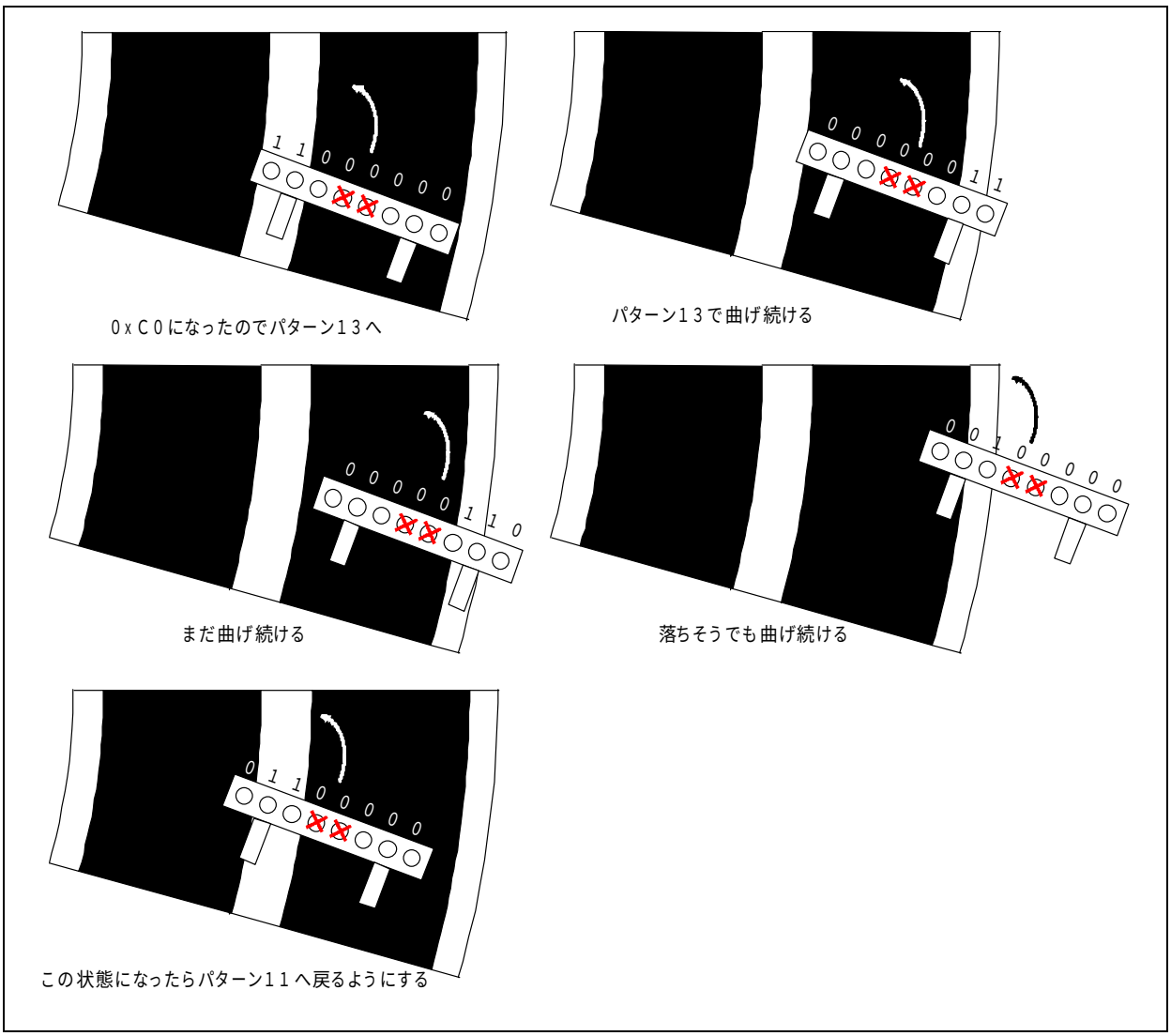
本当は右に大きく寄っている場合でもプログラムでは、「左に大きく寄っている」と誤った判断をすることがあります。もちろん、誤った判断をするとサーボを逆に切るのですぐさま脱輪します。

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

```
212 :           case 0xc0:
213 :               /* 大きく右寄り→左へ大曲げ */
214 :               handle( -25 );
215 :               speed( 21 ,30 );
216 :               pattern = 13;     ←追加　パターン１３へ移る
217 :               break;
```

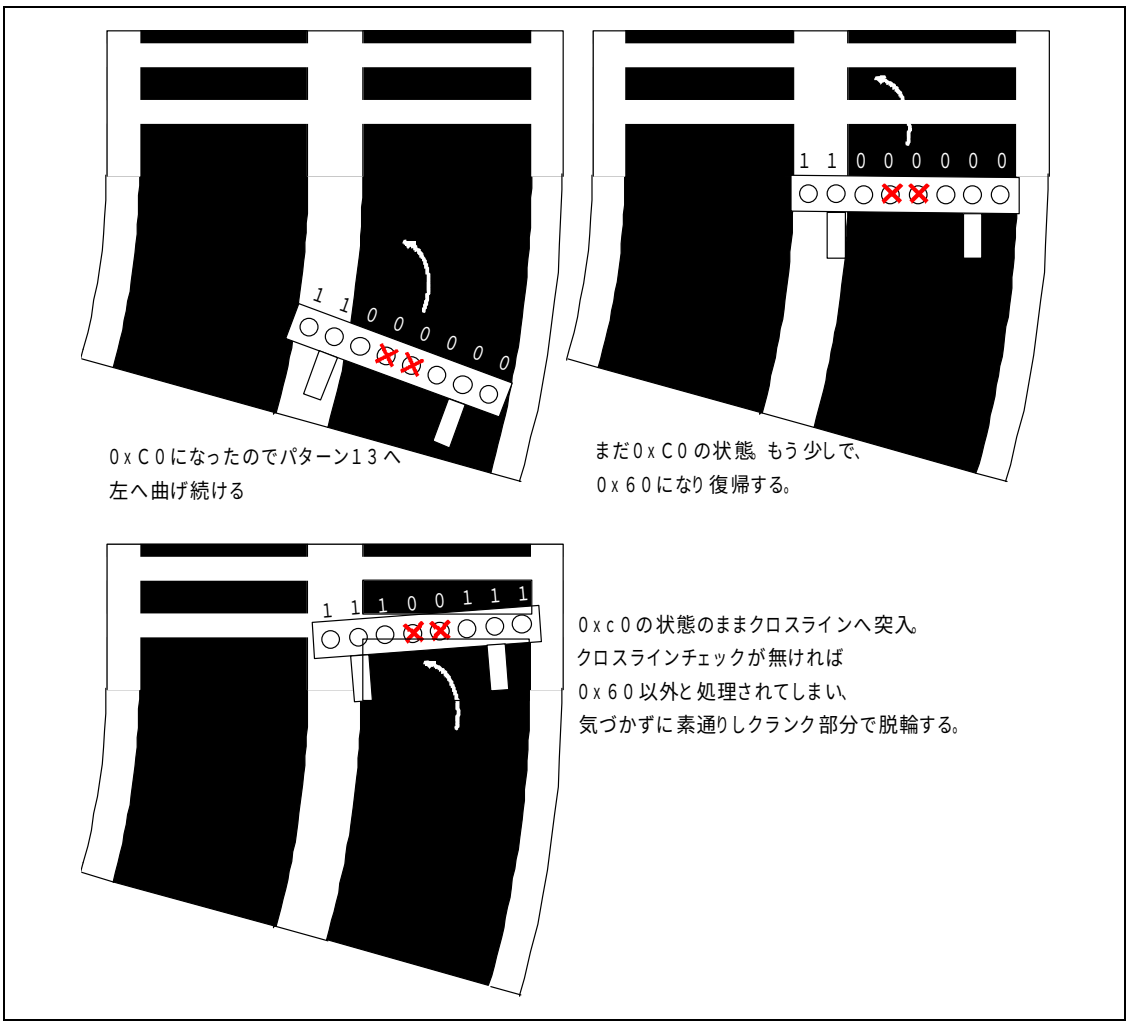
センサが 0xc0 になるとパターン 13 に移ります。
パターン 13 では、どのようになったら通常走行のパターン 11 へ戻るか考えてみます。



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。



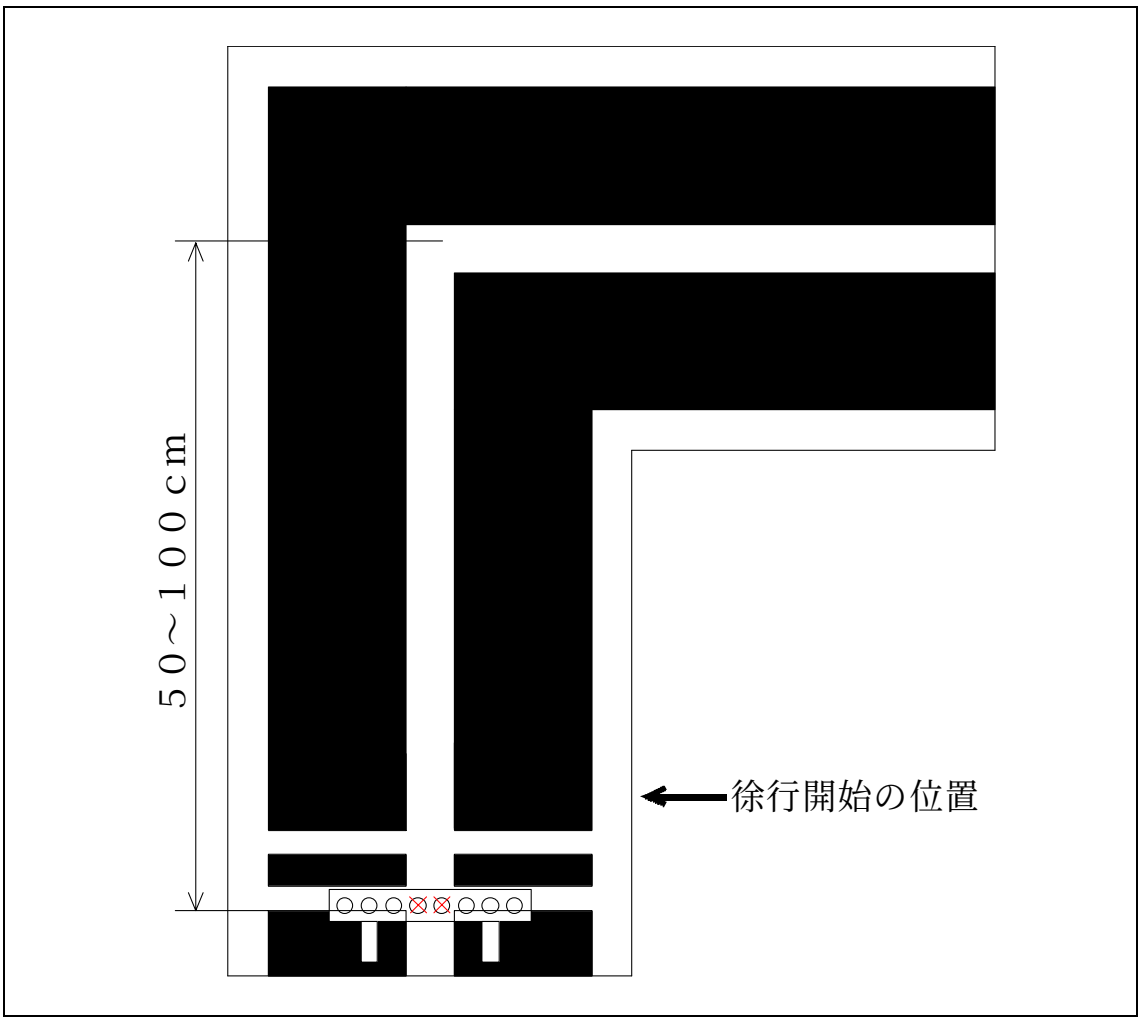
このようにパターン 13 を処理中でも、クロスラインを検出することがありそうです。同様に右ハーフライン、左ハーフラインもあり得ます。そこで、パターン 13 にも 3 種類のチェックを追加します。

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

パターン 13 のプログラムはこれで完成です。

9.23.11 パターン 21：１本目のクロスライン検出時の処理

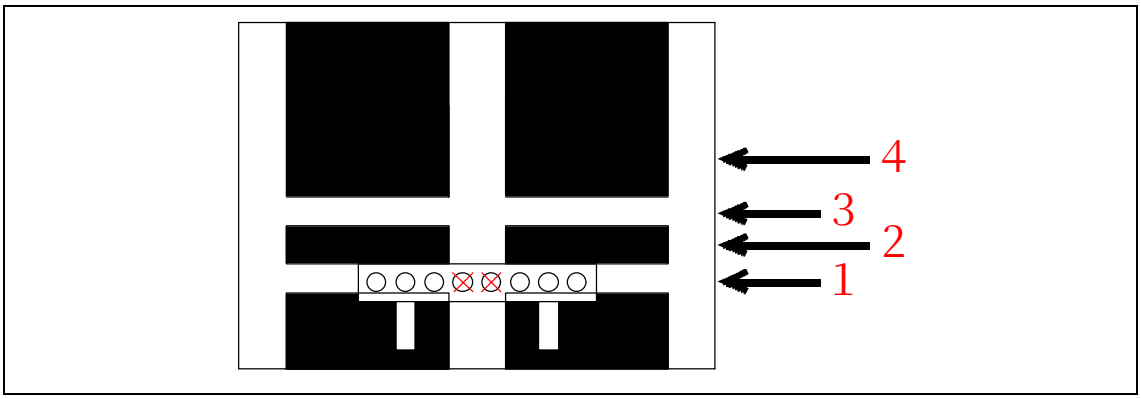
パターン 21 には、下図のような状態になった瞬間に移ってきます。



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。

ちょっと考え方を変えてみます。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、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 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。

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

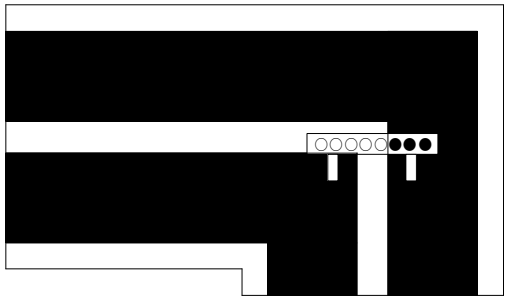
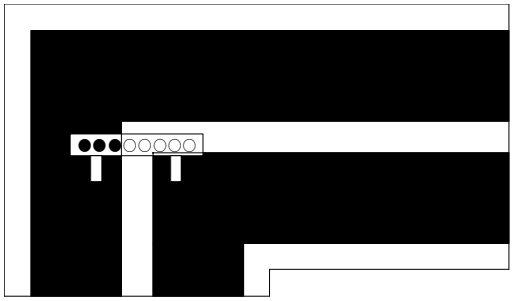
完成しました。本当にこれでよいか見直してみます。パターン 21 でブレーキさせ、時間の基準である cnt1 をクリアしパターン 22 へ。パターン 22 では 0.1 秒たったかチェック。たったならパターン 23 へ。
これでクロスラインを検出してから徐行開始までのプログラムが完成しました。

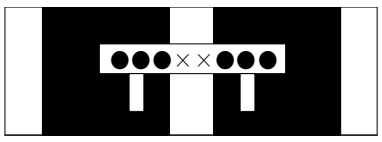
9.23.12　パターン 23：クロスライン後のトレース、クランク検出

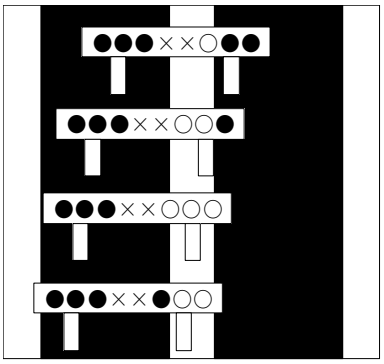
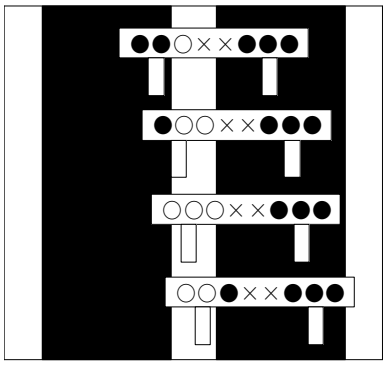
パターン 21、22 では、クロスラインを検出後、ブレーキを 0.1 秒かけ 2 本のクロスラインを通過させました。パターン 23 では、その後の処理を行います。

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。

<table>
<tr>
<td> 
→0xf8
（8つすべてチェック）</td>
<td>左クランク部分では、8つ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。
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脹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。
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。まとめると下記のようになります。
ハンドル：-38 度
左モータ：10%　右モータ：50%
その後、パターン 31 へ移ります。</td>
</tr>
<tr>
<td> 
→0x1f
（8つすべてチェック）</td>
<td>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。
ハンドル：38 度
左モータ：50%　右モータ：10%

その後、パターン 41 へ移ります。</td>
</tr>
<tr>
<td> 
→0x00</td>
<td>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します。まとめると下記のようになります。
ハンドル：0 度
左モータ：40%　右モータ：40%</td>
</tr>
</table>

	マイコンカーが左に寄ったときを考えています。 中心から少しずつ左へずらしていくと左図のように 4 つの状態になりました。センサの状態はもっと左へ寄ることも考えられますが、クロスラインの後は直線しかないと分かっているので、これ以上センサ状態は増やさないでおきます。 動作は、左へ寄っているので右へハンドルを切ります。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。 **ハンドル：8 度** **左モータ：40%　右モータ：36%**
	マイコンカーが右に寄ったときを考えています。 中心から少しずつ右へずらしていくと左図のように 4 つの状態になりました。センサの状態はもっと右へ寄ることも考えられますが、クロスラインの後は直線しかないと分かっているので、これ以上センサ状態は増やさないでおきます。 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。 **ハンドル：-8 度** **左モータ：36%　右モータ：40%**

ポイントは、クランクチェックは8つ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つ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。

```
279 :       case 23:
280 :           /* クロスライン後のトレース、クランク検出 */
281 :           if( sensor_inp(MASK4_4)==0xf8 ) {
282 :               /* 左クランクと判断→左クランククリア処理へ */
283 :               led_out( 0x1 );
284 :               handle( -38 );
285 :               speed( 10 ,50 );
286 :               pattern = 31;
287 :               cnt1 = 0;
288 :               break;
289 :           }
290 :           if( sensor_inp(MASK4_4)==0x1f ) {
291 :               /* 右クランクと判断→右クランククリア処理へ */
292 :               led_out( 0x2 );
293 :               handle( 38 );
294 :               speed( 50 ,10 );
295 :               pattern = 41;
296 :               cnt1 = 0;
297 :               break;
298 :           }
299 :           switch( sensor_inp(MASK3_3) ) {
300 :               case 0x00:
301 :                   /* センタ→まっすぐ */
302 :                   handle( 0 );
303 :                   speed( 40 ,40 );
304 :                   break;
305 :               case 0x04:
306 :               case 0x06:
307 :               case 0x07:
308 :               case 0x03:
309 :                   /* 左寄り→右曲げ */
310 :                   handle( 8 );
311 :                   speed( 40 ,36 );
312 :                   break;
313 :               case 0x20:
314 :               case 0x60:
315 :               case 0xe0:
316 :               case 0xc0:
317 :                   /* 右寄り→左曲げ */
318 :                   handle( -8 );
319 :                   speed( 36 ,40 );
320 :                   break;
321 :           }
322 :           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。

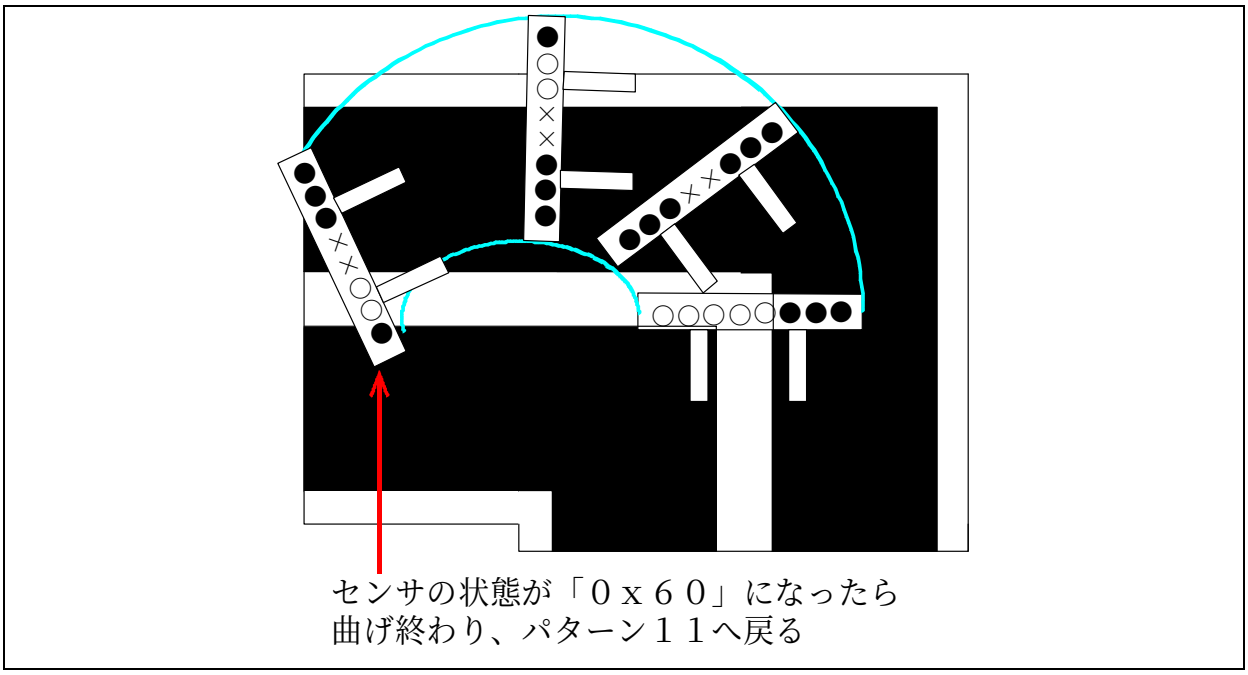
左クランク、右クランクは、if 文で判定しています。その他は、if 文で「sensor_inp(MASK3_3)」とセンサ値を一回一回比較すると長くなるので、switch 文を使いました。

9.23.13 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 つが「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膨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下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。

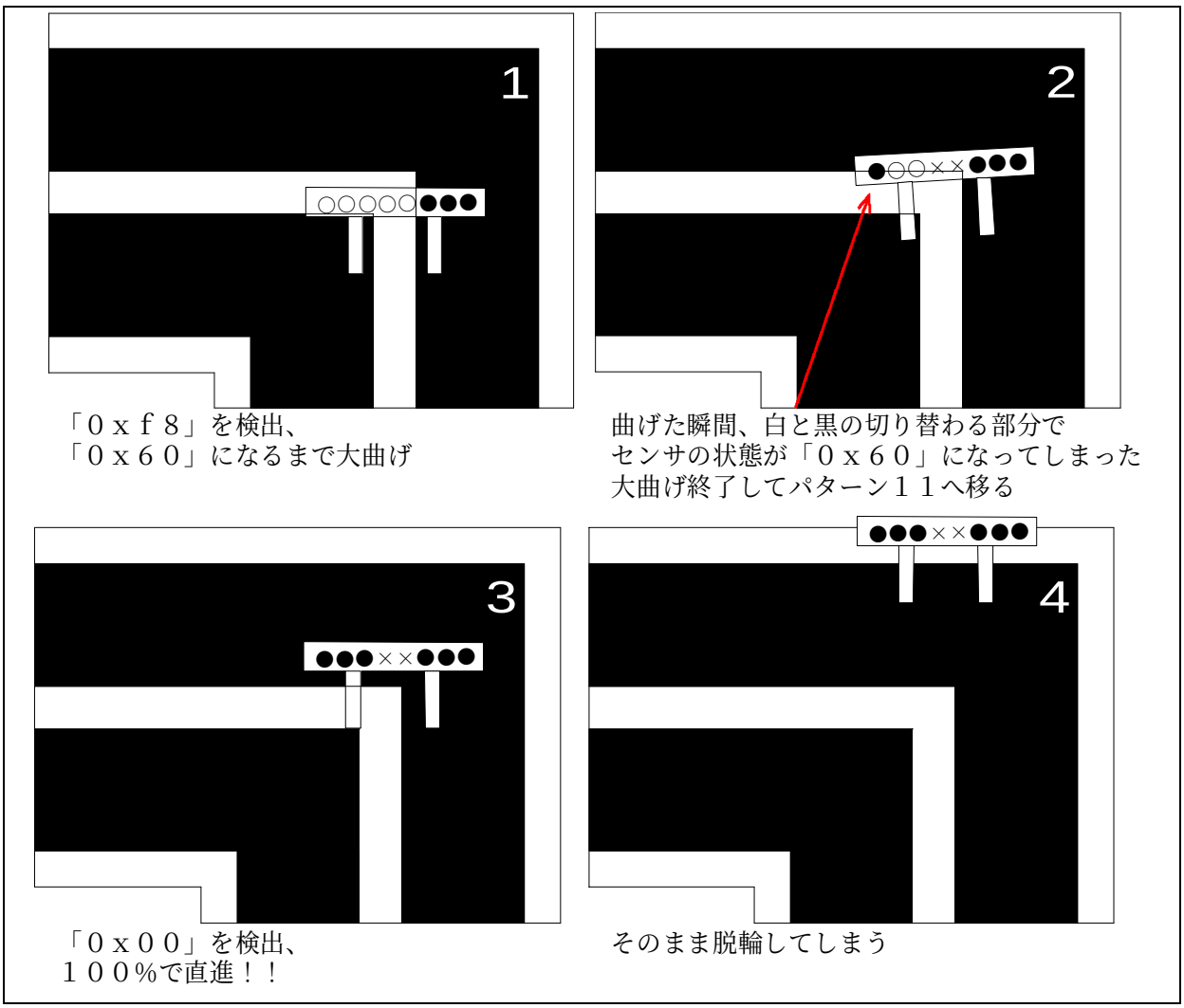
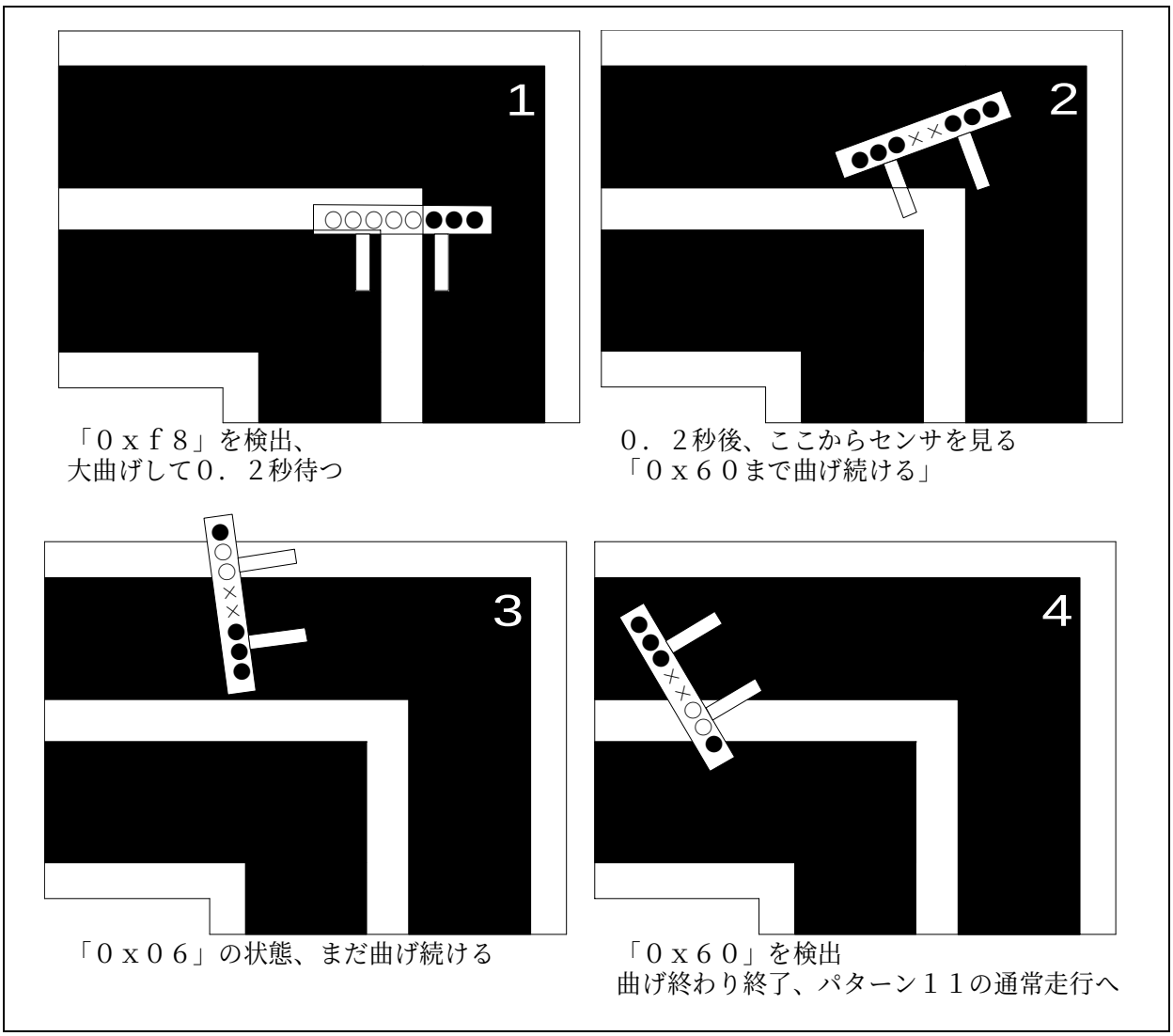


図2のように、白と黒の変わり目で（本当は白と灰と黒色ですが、灰色は白と見なします）センサの状態が「0x60」になっていることが分かりました。いちばん左のセンサ調整がうまくいっておらず先に”0”になっています。いちばん左のセンサ感度をもう少し強くすれば良いのですが、センサのちょっとした感度で脱輪するのは嫌なのでプログラムで何とかできないでしょうか。

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。

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

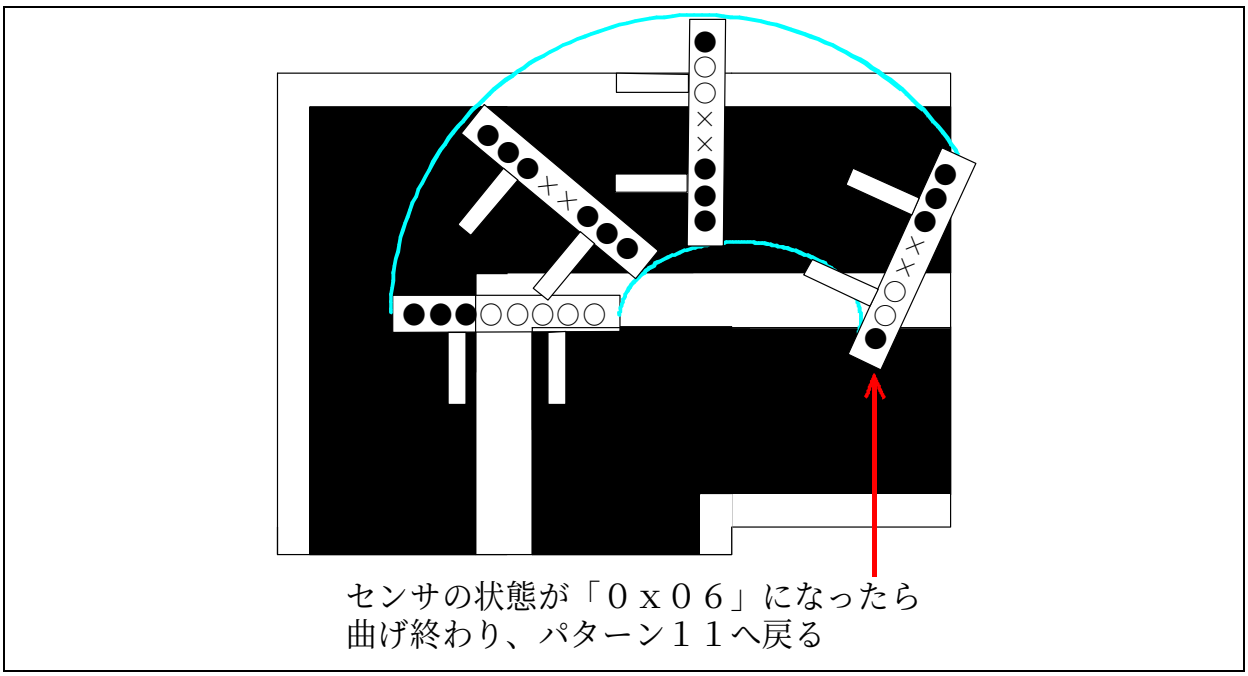
326 行で cnt1 が 200 以上かチェックしています。200 以上なら、すなわち 0.2 秒たったならパターン 32 へ移ります。ちなみに cnt1 のクリアは、パターン 31 へ移る前の 287 行で行っています。

9.23.14 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膨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。

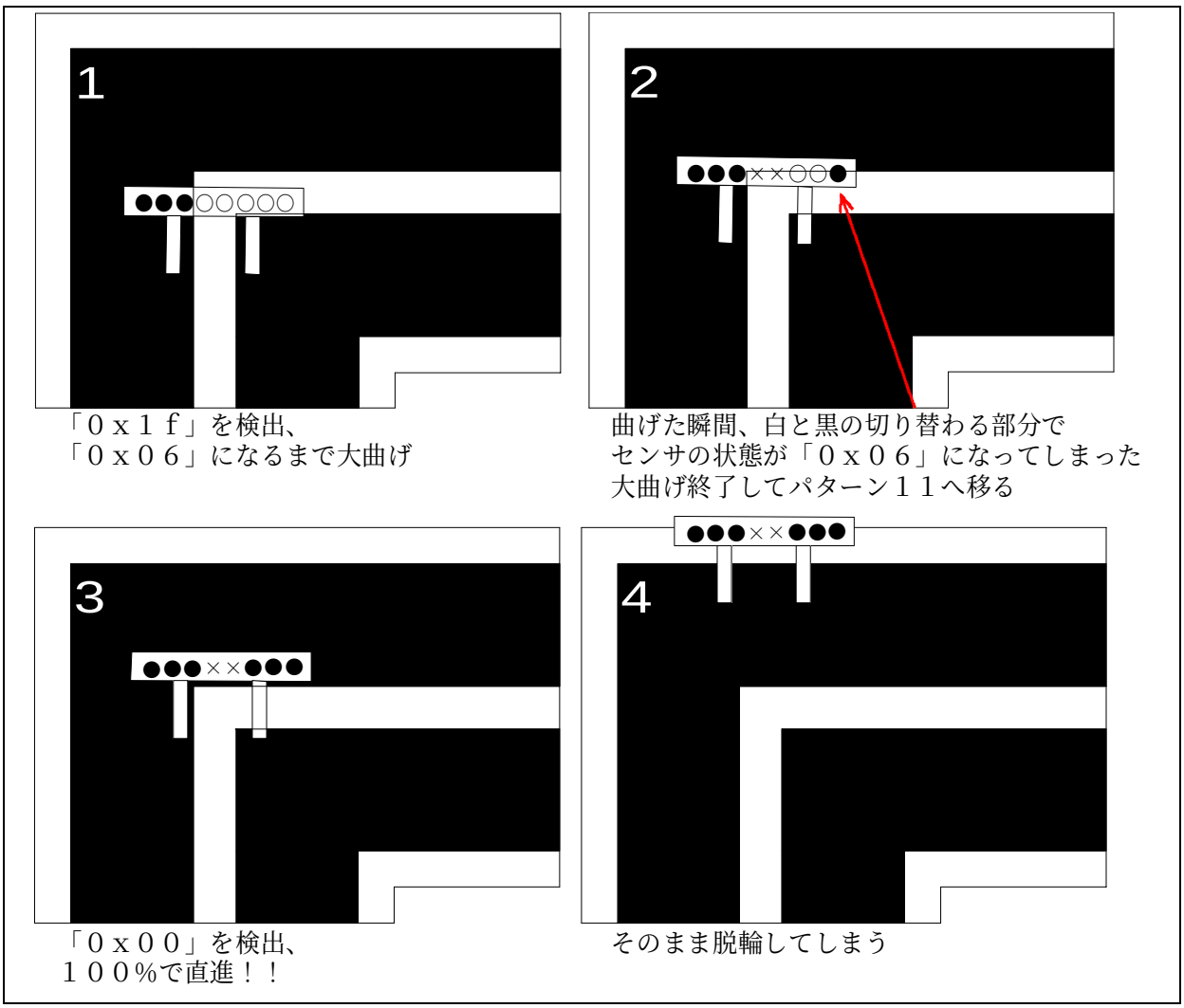
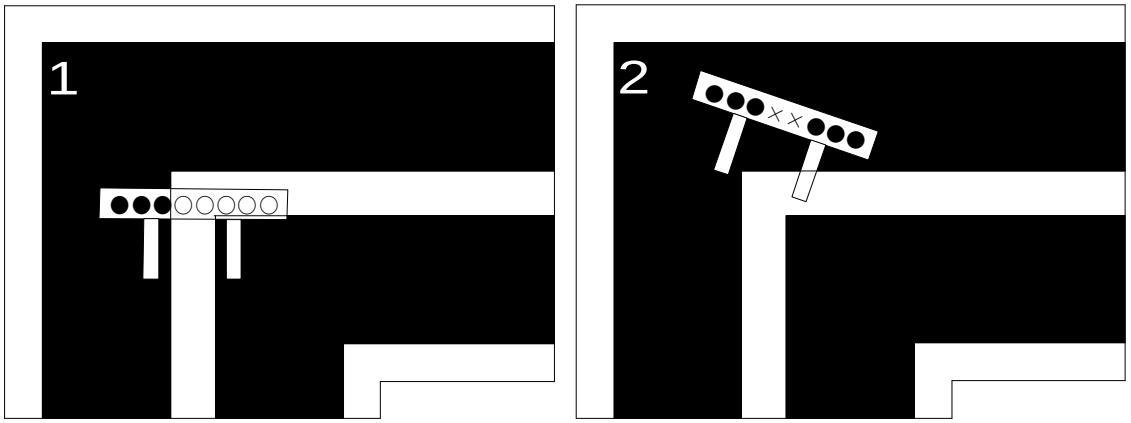


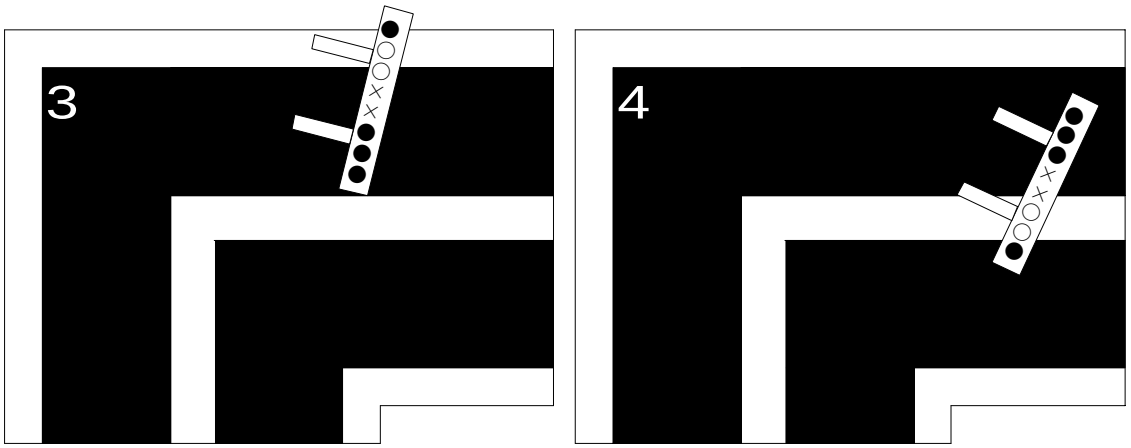
図2のように、白と黒の変わり目で（本当は白と灰と黒色ですが、灰色は白と見なします）センサの状態が「0x06」になっていることが分かりました。いちばん右のセンサ調整がうまくいっておらず先に”0”になっています。いちばん右のセンサ感度をもう少し強くすれば良いのですが、センサのちょっとした感度で脱輪するのは嫌なのでプログラムで何とかできないでしょうか。

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。



「０ｘ１ｆ」を検出、
大曲げして０．２秒待つ

０．２秒後、ここからセンサを見る
「０ｘ０６まで曲げ続ける」



「０ｘ６０」の状態、まだ曲げ続ける

「０ｘ０６」を検出
曲げ終わり終了、パターン１１の通常走行へ

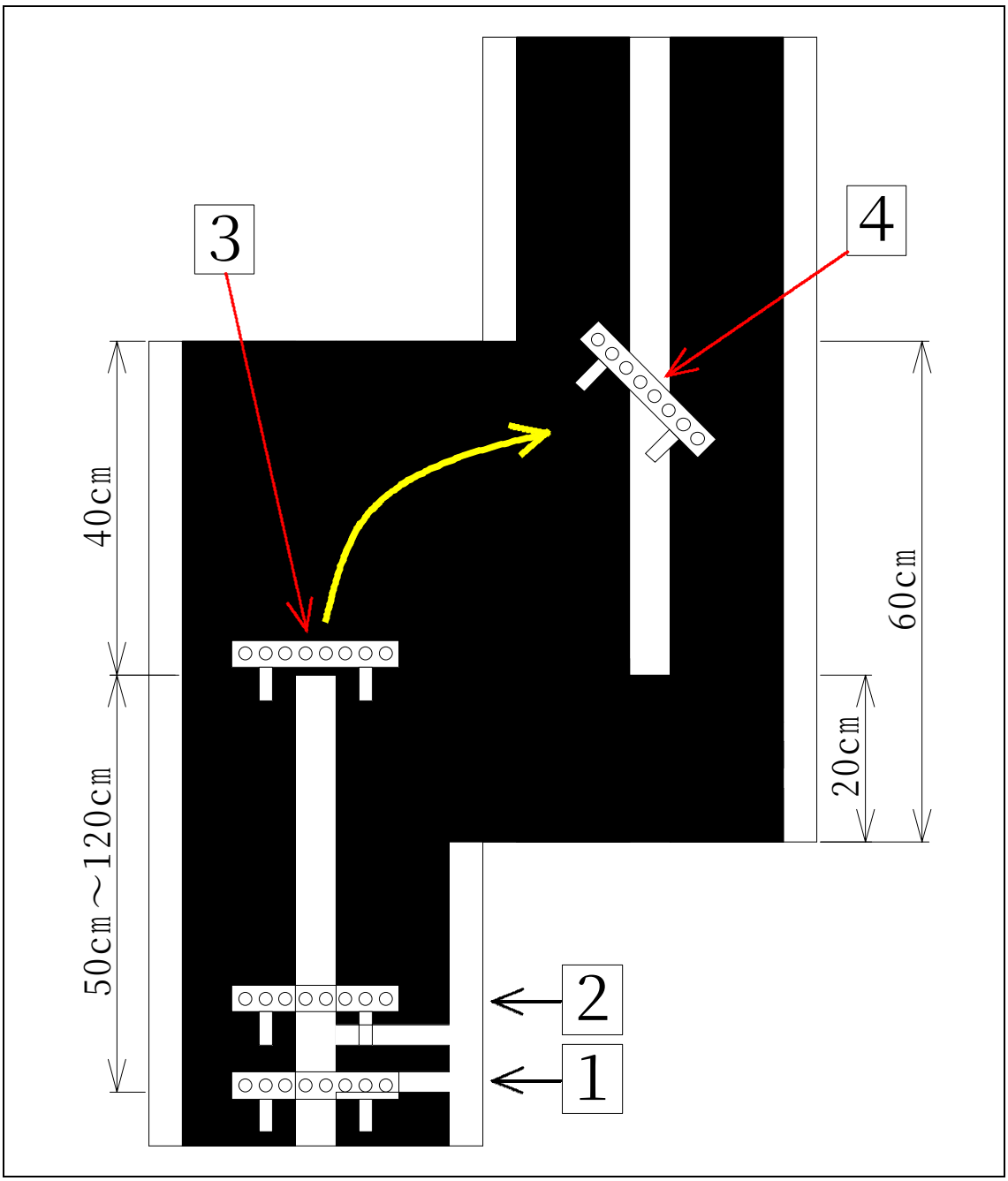
0.2 秒後、図2のように白と黒の変わり目を越えた位置にあります。その後は「0x06」になるまで安心して曲げ続けるだけです。これなら良さそうです。プログラム化します。

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

343 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 296 行で行っています。

9.23.15 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。

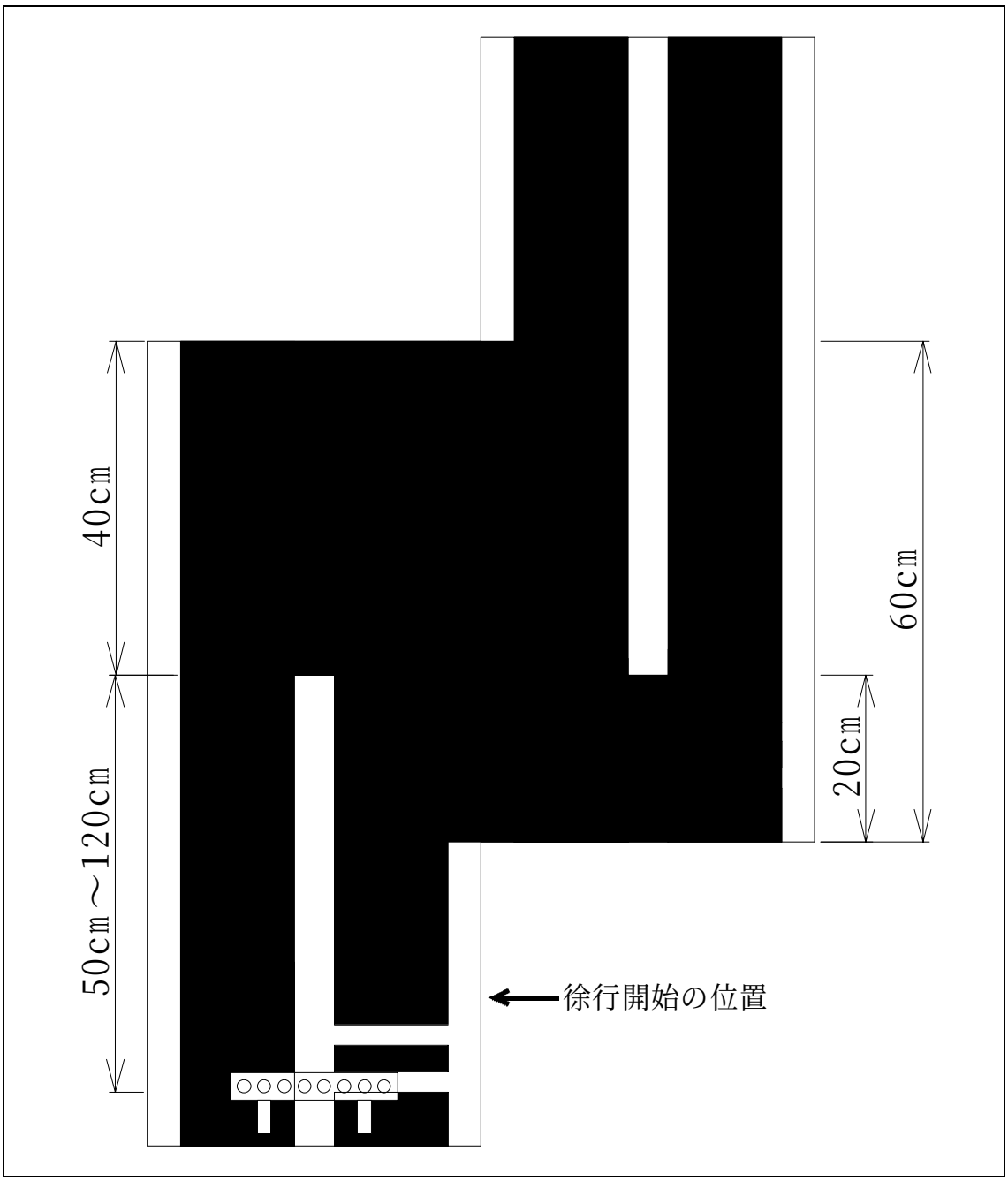


1 check_rightline 関数で右ハーフラインを検出します。50cm～120cm 先で、右レーンに移動するために右に曲がらなければいけないのでブレーキをかけます。また、2 本目の右ハーフラインでセンサが誤検出しないよう 2 の位置までセンサは見ません。
2 この位置から徐行開始します。中心線をトレースしながら進んでいきます。
3 中心線が無くなると、右へハンドルを切ります。
4 新しい中心線を検出すると、今度はこの中心線でライントレースを再開します。

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。

9.23.16 パターン 51：１本目の右ハーフライン検出時の処理

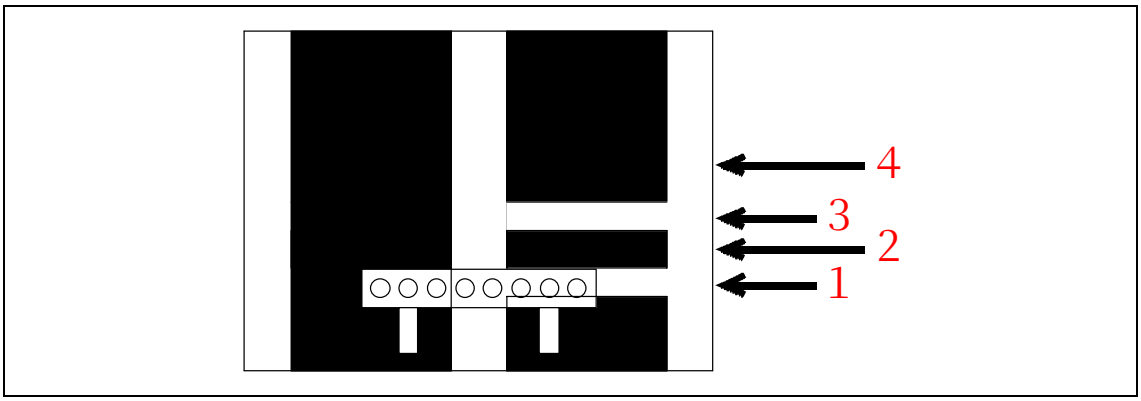
パターン 51 には、下図のような状態になった瞬間に移ってきます。



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。

ちょっと考え方を変えてみます。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。

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

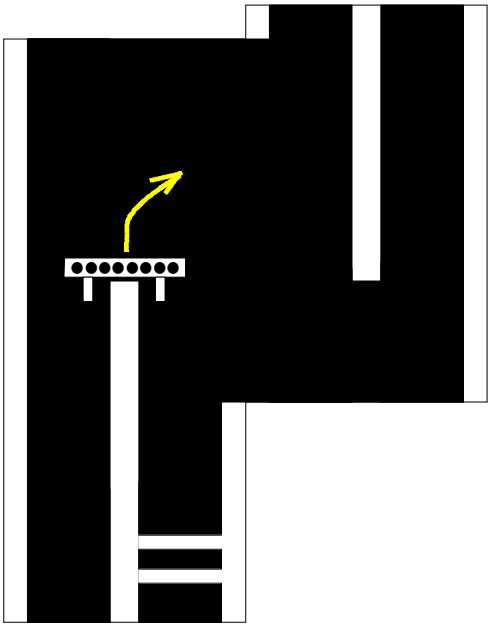
本当にこれでよいか見直してみます。パターン 51 でブレーキさせ、時間の基準である cnt1 をクリアしパターン 52 へ。パターン 52 では 0.1 秒たったかチェック。たったならパターン 53 へ。
これで右ハーフラインを検出してから徐行開始までのプログラムが完成しました。

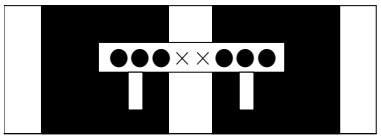
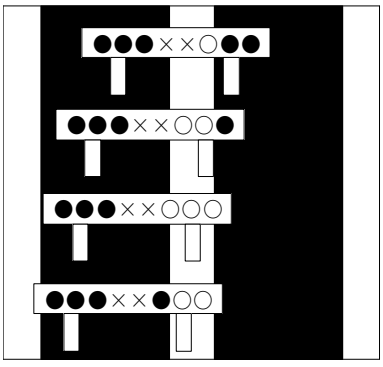
9.23.17　パターン 53：右ハーフライン後のトレース

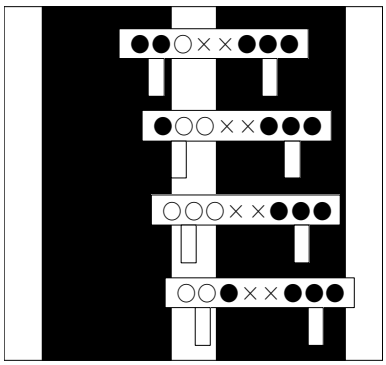
パターン 51、52 では、右ハーフラインを検出後、ブレーキを 0.1 秒かけ 2 本の右ハーフラインを通過させました。パターン 53 では、その後の処理を行います。

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。

<table>
<tr>
<td> 
→0x00
（8つすべてチェック）</td>
<td>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つのセンサ状態が左図のように「0x00」になりました。この状態を検出すると、右曲げを開始します。
このとき、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ことは予想できます。実際に何%にすればよいかは、マイコンカーのスピードやタイヤの滑り具合、サーボの反応速度によって変わるので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。
ハンドル：15 度
左モータ：40%　右モータ：32%

その後、パターン 54 へ移ります。</td>
</tr>
<tr>
<td> 
→0x00</td>
<td>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します。まとめると下記のようになります。
ハンドル：0 度
左モータ：40%　右モータ：40%</td>
</tr>
<tr>
<td> 
→0x04
→0x06
→0x07
→0x03</td>
<td>マイコンカーが左に寄ったときを考えています。中心から少しずつ左へずらしていくと左図のように 4 つの状態になりました。センサの状態はもっと左へ寄ることも考えられますが、右ハーフラインの後は直線しかないと分かっているので、これ以上センサ状態は増やさないでおきます。
動作は、左へ寄っているので右へハンドルを切ります。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。
ハンドル：8 度
左モータ：40%　右モータ：36%</td>
</tr>
</table>



マイコンカーが右に寄ったときを考えています。中心から少しずつ、右へずらしていくと左図のように 4 つの状態になりました。センサの状態はもっと右へ寄ることも考えられますが、右ハーフラインの後は直線しかないと分かっているので、これ以上センサ状態は増やさないでおきます。
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。
ハンドル：-8 度
左モータ：36%　右モータ：40%

ポイントは、中心線があるかどうかのチェックは8つ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つ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。

```
375 :       case 53:
376 :           /* 右ハーフライン後のトレース、レーンチェンジ */
377 :           if( sensor_inp(MASK4_4) == 0x00 ) {
378 :               handle( 15 );
379 :               speed( 40 ,32 );
380 :               pattern = 54;
381 :               cnt1 = 0;
382 :               break;
383 :           }
384 :           switch( sensor_inp(MASK3_3) ) {
385 :               case 0x00:
386 :                   /* センタ→まっすぐ */
387 :                   handle( 0 );
388 :                   speed( 40 ,40 );
389 :                   break;
390 :               case 0x04:
391 :               case 0x06:
392 :               case 0x07:
393 :               case 0x03:
394 :                   /* 左寄り→右曲げ */
395 :                   handle( 8 );
396 :                   speed( 40 ,36 );
397 :                   break;
398 :               case 0x20:
399 :               case 0x60:
400 :               case 0xe0:
401 :               case 0xc0:
402 :                   /* 右寄り→左曲げ */
403 :                   handle( -8 );
404 :                   speed( 36 ,40 );
405 :                   break;
406 :               default:
407 :                   break;
408 :           }
409 :           break;
```

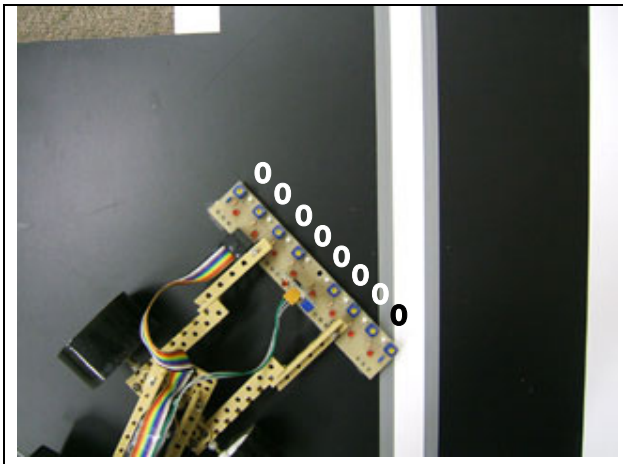
case を続けて書くと
0x04 または 0x06 または 0x07 または 0x03 のとき
という意味になります。

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。

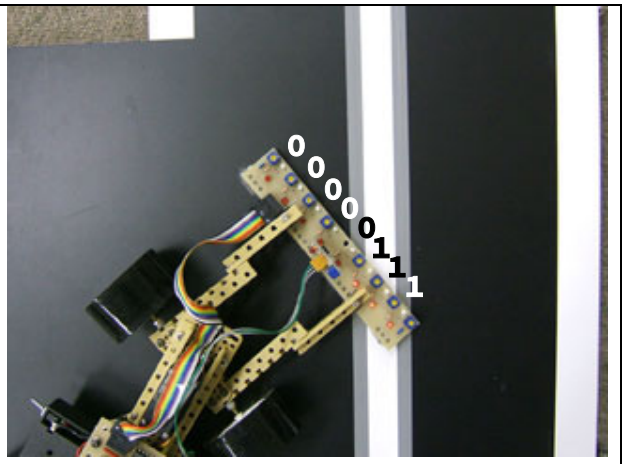
9.23.18 パターン 54：右レーンチェンジ終了のチェック

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

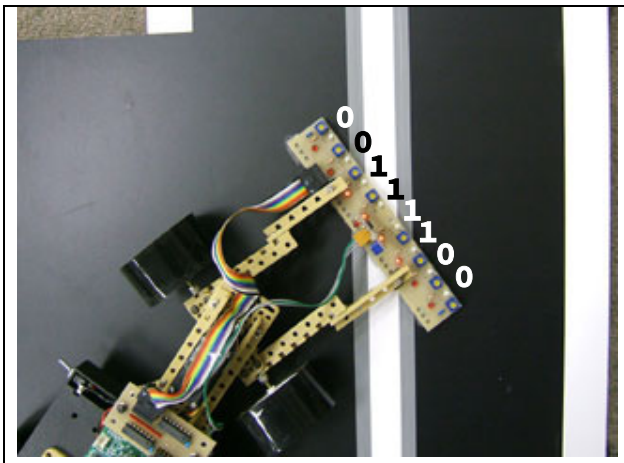
処理を行いました。次は、右側にある新しい中心線まで行きます。新しい中心線を見つけたら、その中心線をトレースしていきます。これで右レーンチェンジ処理は完了です。では、新しい中心線と見なすセンサ状態はどのような状態でしょうか。



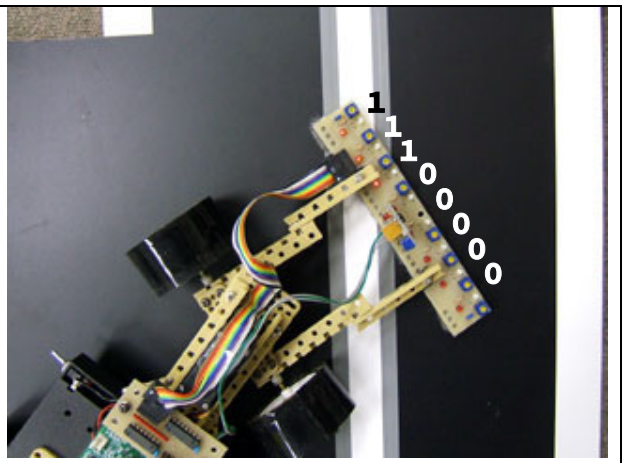
新しい中心線を見つける直前です。



センサの右端で検出しました。
センサの状態は、「0000　0111」です。



センサの中心まで来ました。
センサの状態は、「0011　1100」です。



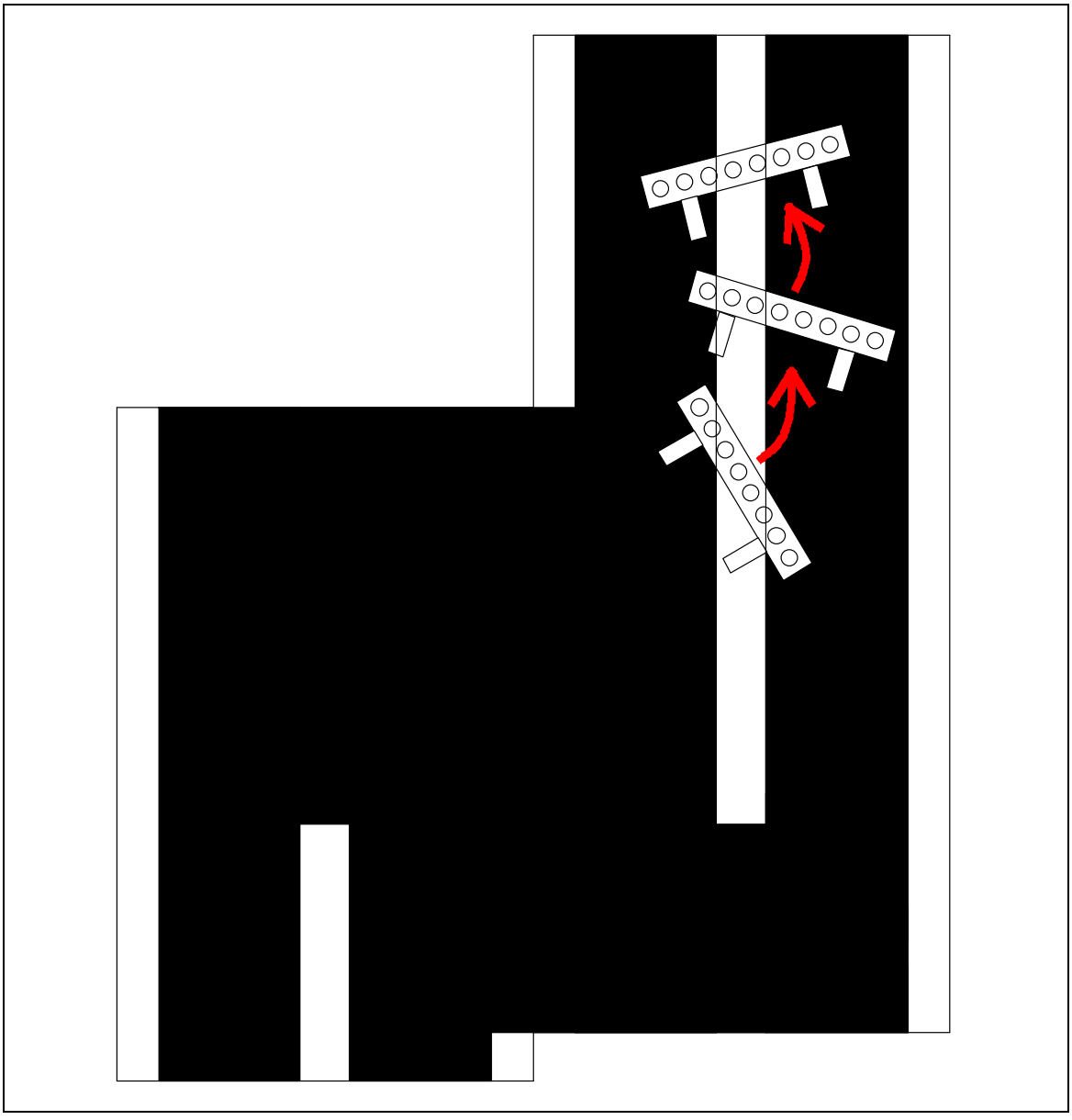
センサの左端まで来ました。
センサの状態は、「1110　0000」です。

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つチェックしたとき、「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。

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

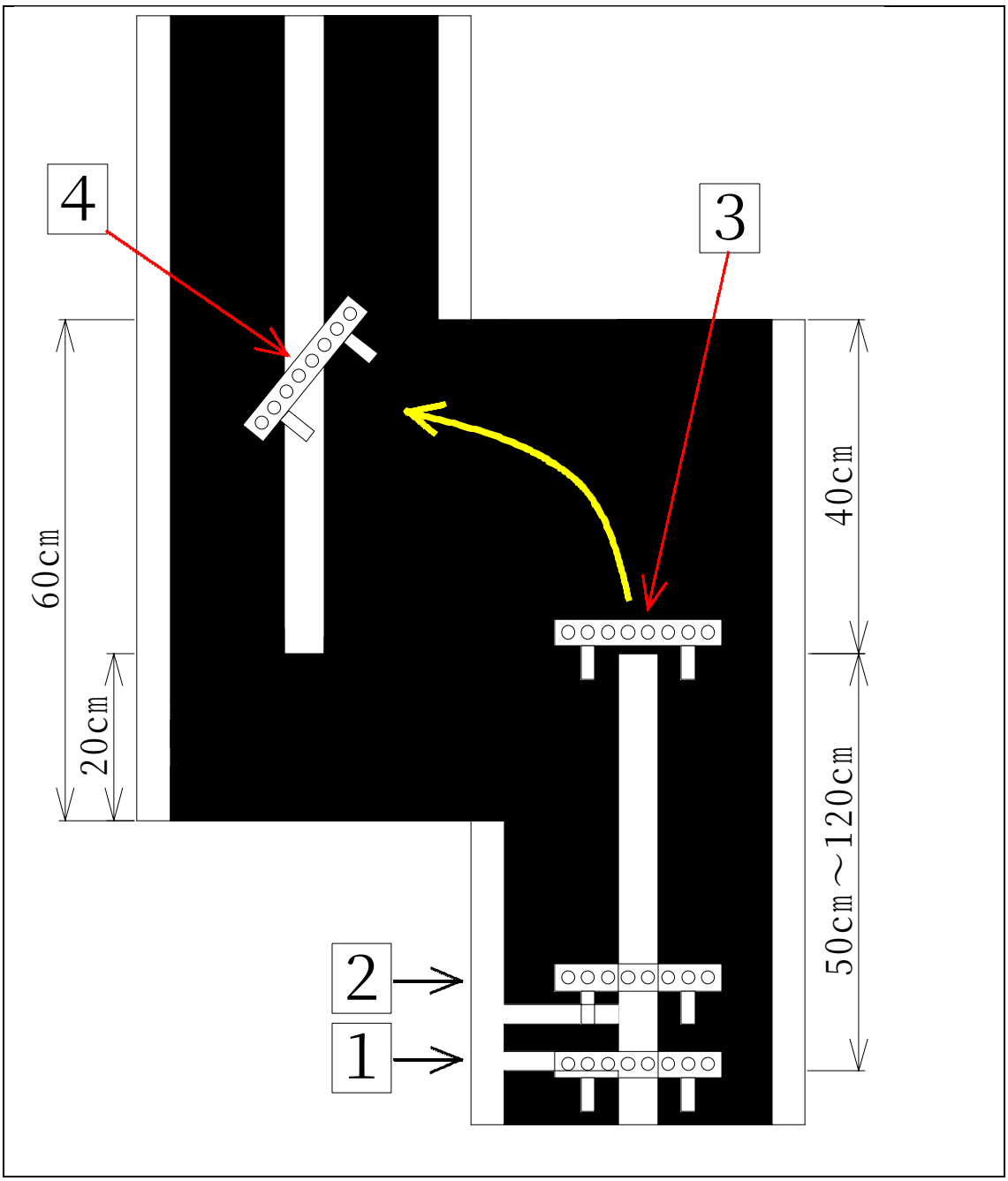
右ハーフラインを検出したときに LED を点灯させたので、414 行で消灯させてからパターン 11 へ移るようにします。



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。

9.23.19 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。

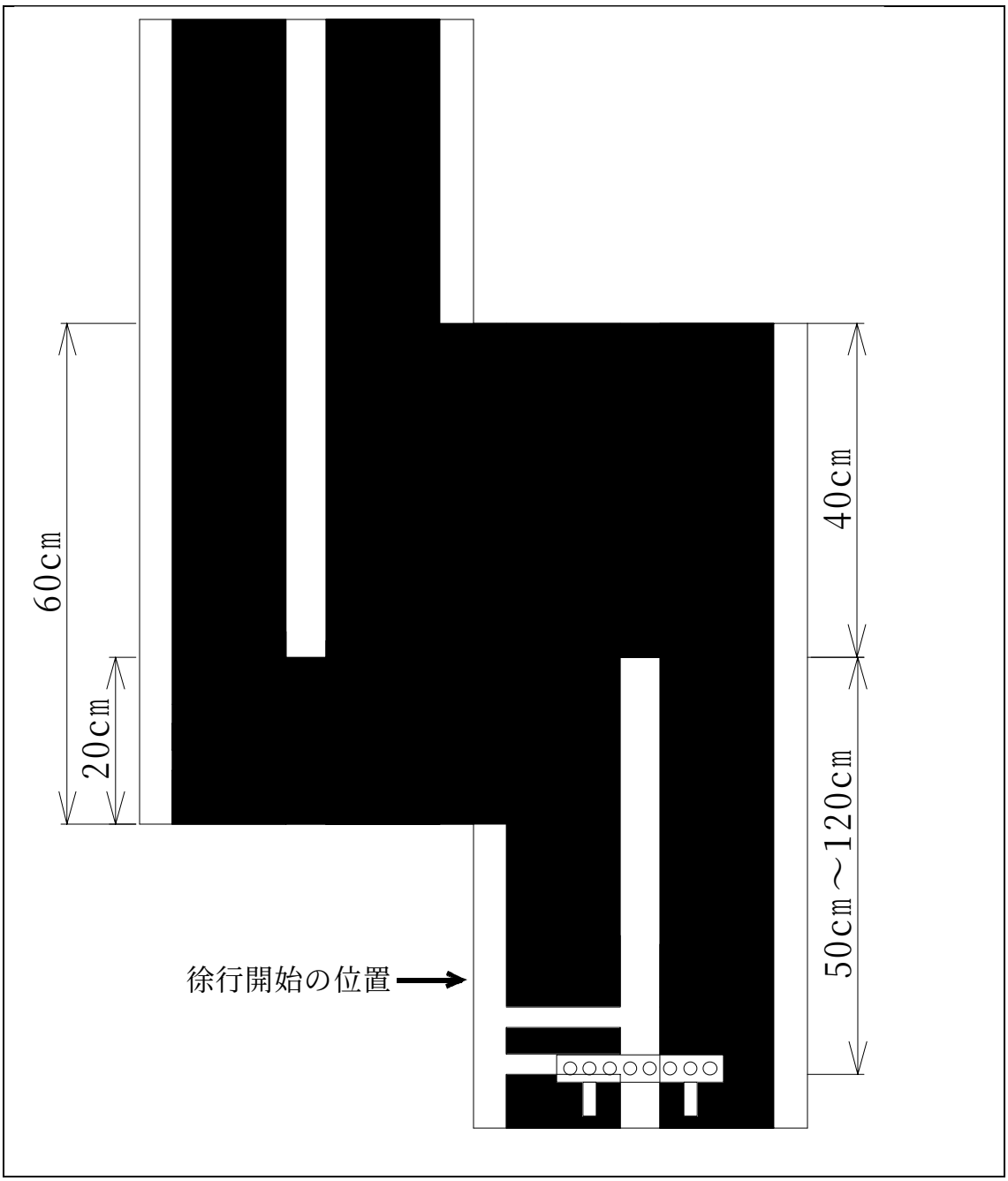


1 check_leftline 関数で左ハーフラインを検出します。50cm～120cm 先で、左レーンに移動するために左に曲がらなければいけないのでブレーキをかけます。また、2 本目の左ハーフラインでセンサが誤検出しないよう 2 の位置までセンサは見ません。
2 この位置から徐行開始します。中心線をトレースしながら進んでいきます。
3 中心線が無くなると、左へハンドルを切ります。
4 新しい中心線を検出すると、今度はこの中心線でライントレースを再開します。

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。

9.23.20 パターン 61：１本目の左ハーフライン検出時の処理

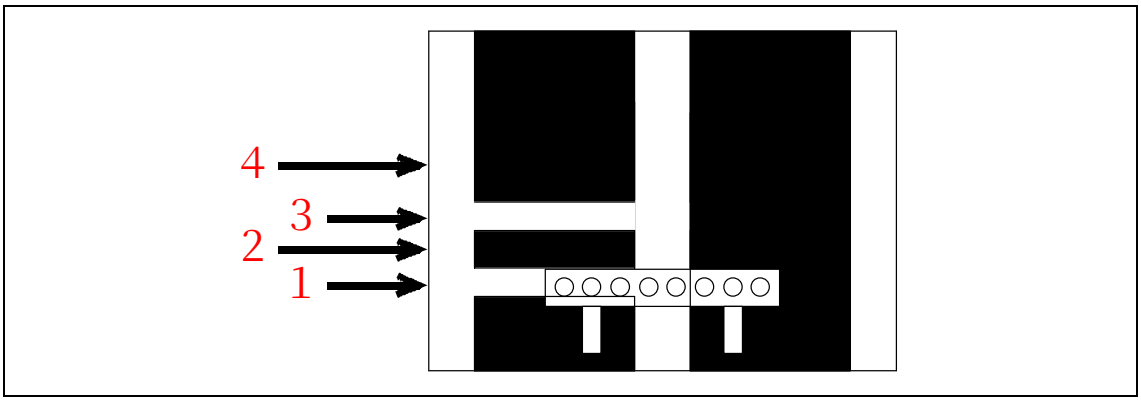
パターン 61 には、下記のような状態になった瞬間に移ってきます。



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。

ちょっと考え方を変えてみます。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。

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

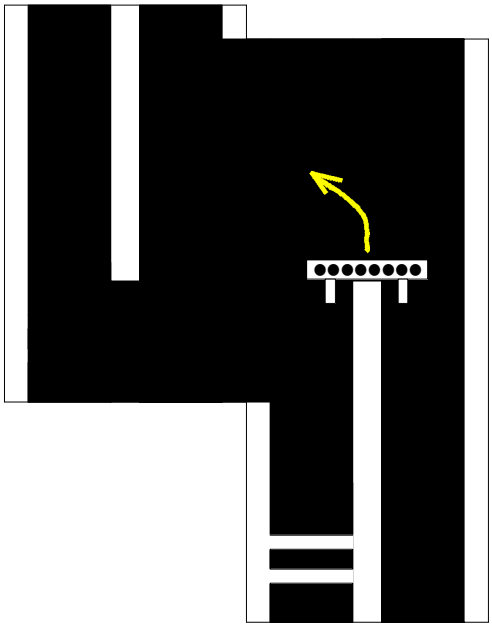
本当にこれでよいか見直してみます。パターン 61 でブレーキさせ、時間の基準である cnt1 をクリアしパターン 62 へ。パターン 62 では 0.1 秒たったかチェック。たったならパターン 63 へ。良さそうです。
これで左ハーフラインを検出してから徐行開始までのプログラムが完成しました。

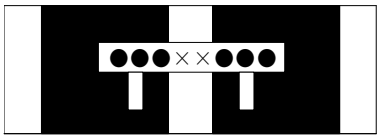
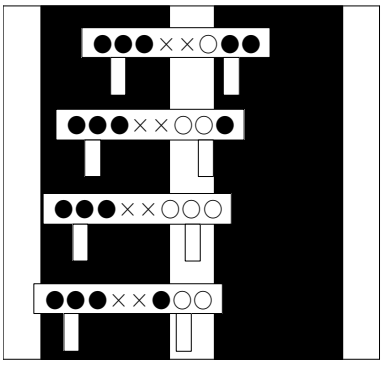
9.23.21 パターン 63：左ハーフライン後のトレース

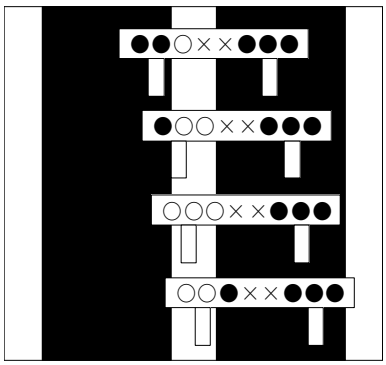
パターン 61、62 では、左ハーフラインを検出後、ブレーキを 0.1 秒かけ 2 本の左ハーフラインを通過させました。パターン 63 では、その後の処理を行います。

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。

<table>
<tr>
<td> 
→0x00
（8つすべてチェック）</td>
<td>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つのセンサ状態が左図のように「0x00」になりました。この状態を検出すると、左曲げを開始します。
このとき、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ことは予想できます。実際に何%にすればよいかは、マイコンカーのスピードやタイヤの滑り具合、サーボの反応速度によって変わるので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。
ハンドル：-15 度
左モータ：32%　右モータ：40%

その後、パターン 64 へ移ります。</td>
</tr>
<tr>
<td> 
→0x00</td>
<td>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します。まとめると下記のようになります。
ハンドル：0 度
左モータ：40%　右モータ：40%</td>
</tr>
<tr>
<td> 
→0x04
→0x06
→0x07
→0x03</td>
<td>マイコンカーが左に寄ったときを考えています。中心から少しずつ左へずらしていくと左図のように 4 つの状態になりました。センサの状態はもっと左へ寄ることも考えられますが、右ハーフラインの後は直線しかないと分かっているので、これ以上センサ状態は増やさないでおきます。
動作は、左へ寄っているので右へハンドルを切ります。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。
ハンドル：8 度
左モータ：40%　右モータ：36%</td>
</tr>
</table>



マイコンカーが右に寄ったときを考えています。中心から少しずつ、右へずらしていくと左図のように 4 つの状態になりました。センサの状態はもっと右へ寄ることも考えられますが、右ハーフラインの後は直線しかないと分かっているので、これ以上センサ状態は増やさないでおきます。
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。
ハンドル：-8 度
左モータ：36%　右モータ：40%

ポイントは、中心線があるかどうかのチェックは8つ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つ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。

```
437 :       case 63:
438 :           /* 左ハーフライン後のトレース、レーンチェンジ */
439 :           if( sensor_inp(MASK4_4) == 0x00 ) {
440 :               handle( -15 );
441 :               speed( 32 ,40 );
442 :               pattern = 64;
443 :               cnt1 = 0;
444 :               break;
445 :           }
446 :           switch( sensor_inp(MASK3_3) ) {
447 :               case 0x00:
448 :                   /* センタ→まっすぐ */
449 :                   handle( 0 );
450 :                   speed( 40 ,40 );
451 :                   break;
452 :               case 0x04:
453 :               case 0x06:
454 :               case 0x07:
455 :               case 0x03:
456 :                   /* 左寄り→右曲げ */
457 :                   handle( 8 );
458 :                   speed( 40 ,36 );
459 :                   break;
460 :               case 0x20:
461 :               case 0x60:
462 :               case 0xe0:
463 :               case 0xc0:
464 :                   /* 右寄り→左曲げ */
465 :                   handle( -8 );
466 :                   speed( 36 ,40 );
467 :                   break;
468 :               default:
469 :                   break;
470 :           }
471 :           break;
```

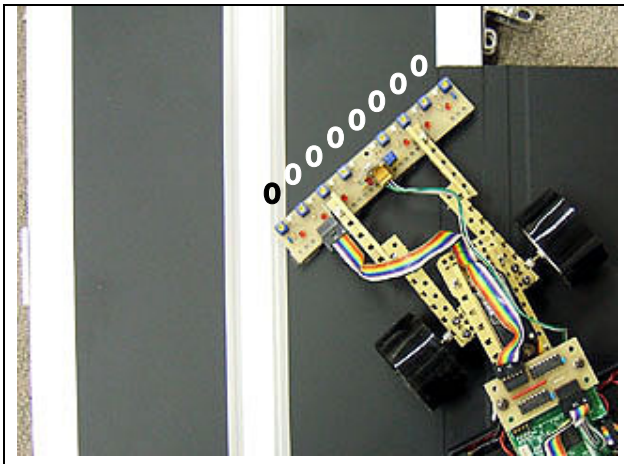
case を続けて書くと
0x04 または 0x06 または 0x07 または 0x03 のとき
という意味になります。

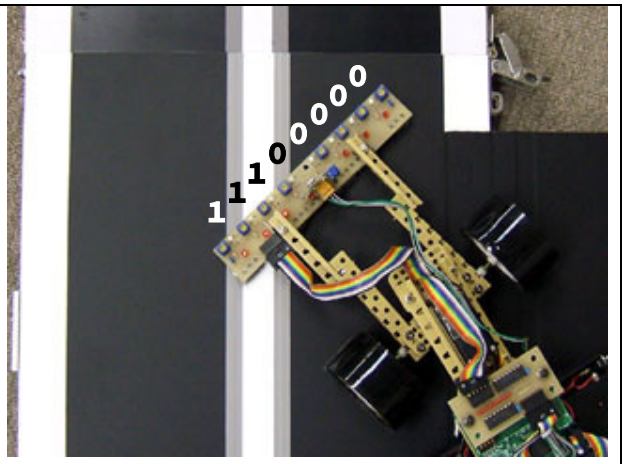
case を続けて書くと
0x20 または 0x60 または 0xe0 または 0xc0 のとき
という意味になります。

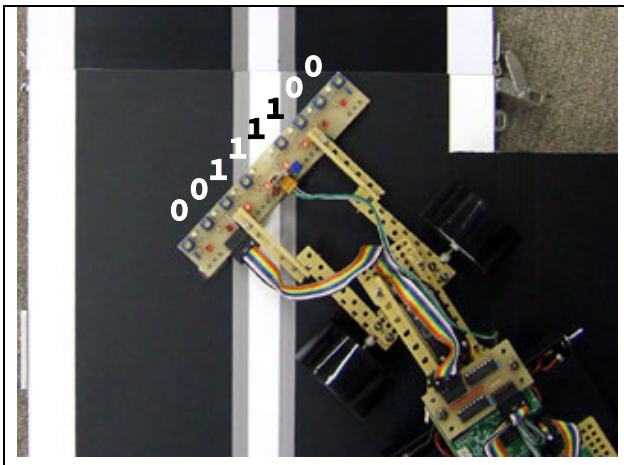
9.23.22 パターン 64：左レーンチェンジ終了のチェック

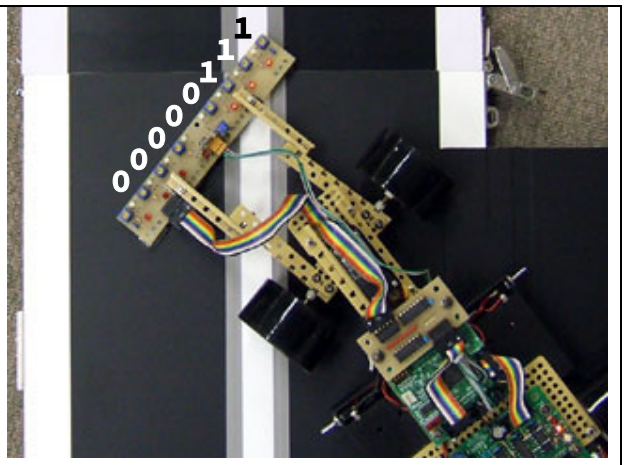
ここまで、
1. 左ハーフライン検出
2. 徐行して進む
3. 中心線が無くなったことを検出して左へ曲げる

処理を行いました。次は、左側にある新しい中心線まで行きます。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。

<table>
<tr>
<td> 

新しい中心線を見つける直前です。</td>
<td> 

センサの左端で検出しました。
センサの状態は、「1110　0000」です。</td>
</tr>
<tr>
<td> 

センサの中心まで来ました。
センサの状態は、「0011　1100」です。</td>
<td> 

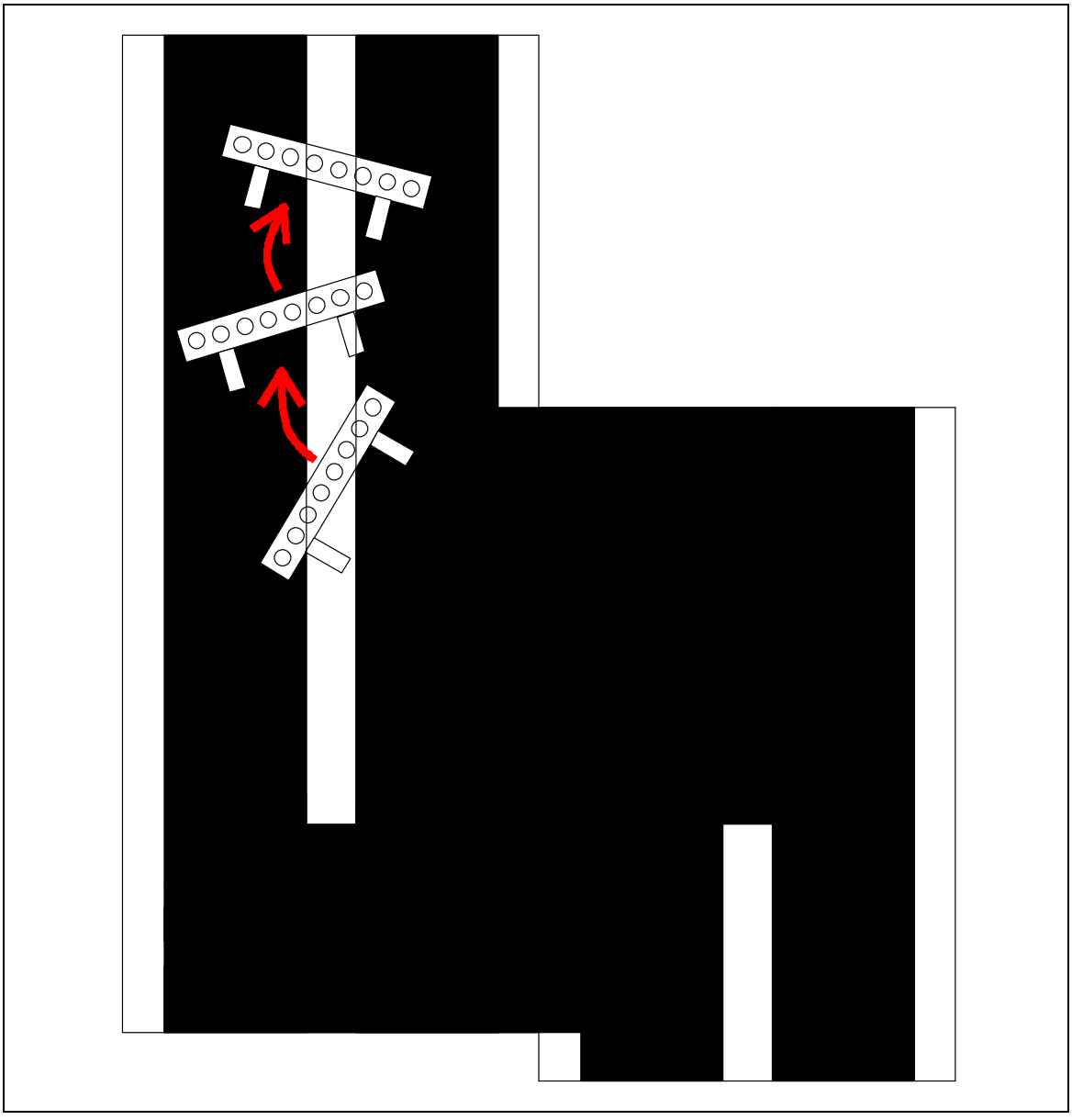
センサの右端まで来ました。
センサの状態は、「0000　0111」です。</td>
</tr>
</table>

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つチェックしたとき、「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。

```
473 :       case 64:
474 :           /* 左レーンチェンジ終了のチェック */
475 :           if( sensor_inp( MASK4_4 ) == 0x3c ) {
476 :               led_out( 0x0 );
477 :               pattern = 11;
478 :               cnt1 = 0;
479 :           }
480 :           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、476 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。

9.23.23 どれでもないパターン

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

10. プログラム解説「kit06start.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。

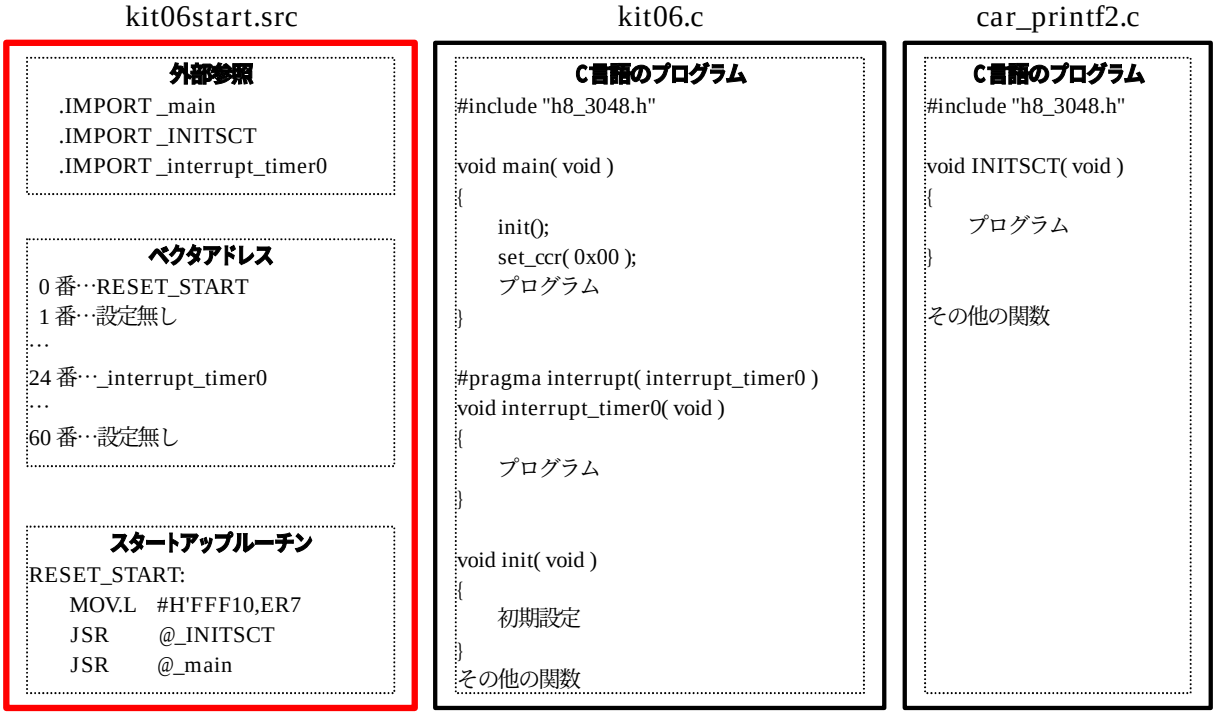
10.1 プログラムリスト

```
 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   ﾄﾗｯﾌﾟ命令
26 :          .DATA.L RESERVE               ;  9 H'000024   ﾄﾗｯﾌﾟ命令
27 :          .DATA.L RESERVE               ; 10 H'000028   ﾄﾗｯﾌﾟ命令
28 :          .DATA.L RESERVE               ; 11 H'00002c   ﾄﾗｯﾌﾟ命令
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 MOVI
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 OVI0
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 OVI1
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 OVI2
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 OVI3
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 OVI4
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   SCI0 ERI0
70 :          .DATA.L RESERVE               ; 53 H'0000d4   SCI0 RXI0
71 :          .DATA.L RESERVE               ; 54 H'0000d8   SCI0 TXI0
72 :          .DATA.L RESERVE               ; 55 H'0000dc   SCI0 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
```

10.2 概要

プロジェクト「kit06」の構成を簡単に書くと下記のようになります。



・kit06start.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
・kit06.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

kit06start.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。

11. プログラム解説「car_printf2.c」

11.1 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。

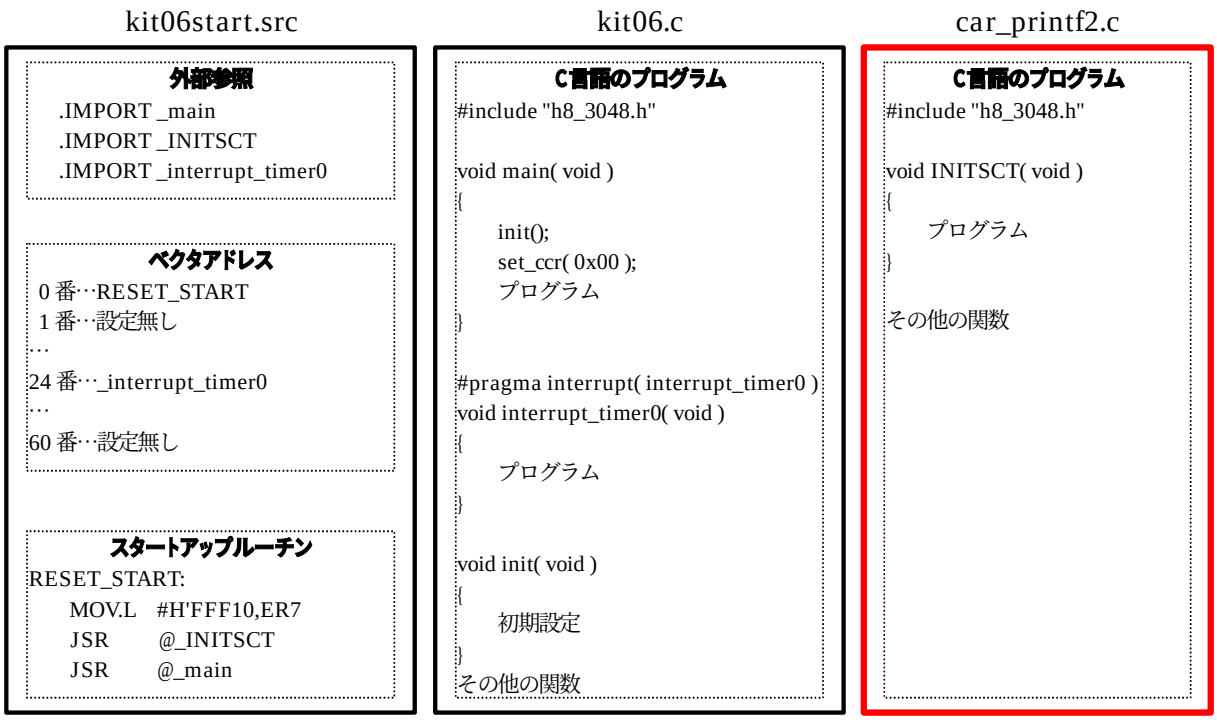
```
  1 :  /************************************************************************/
  2 :  /* マイコンカー用printf,scanf使用プログラム Ver2                        */
  3 :  /*                          2006.04 ジャパンマイコンカーラリー実行委員会 */
  4 :  /************************************************************************/
  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    ボーレートレジスタ設定値                                     */
 55 :  /* 戻り値 なし                                                          */
 56 :  /************************************************************************/
 57 :  void init_sci1( 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 :   /************************************************************************/
```

11.2 概要

プロジェクト「kit06」の構成を簡単に書くと下記のようになります。



･kit06start.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
･kit06.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。

11.3 宣言されている関数

関数名	内容
init_sci1	void init_sci1(int smr, int brr) printf、scanf 関数を使用するために通信の設定を行います。 例） init_sci1(0x00, 79); 詳しくは、H8/3048F-ONE 実習マニュアルの「**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**」(P230)を参照してください。
INITSCT	void INITSCT(void) 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） INITSCT();

intTXI1	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

あと、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。

11.4 シンボル定義

設定行数	内容
24 行	24 : #define SEND_BUFF_SIZE 64 /* 送信バッファサイズ */ 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。
25 行	25 : #define RECV_BUFF_SIZE 32 /* 受信バッファサイズ */ 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。

●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。kit06.c など、main 関数のある C 言語ソースファイルといっしょに使用します。

kit06.c では、セクション B を使用しています。そのため、kit06start.src ファイルで、INITSCT 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。

12. プロジェクト「kit06」のツールチェインの設定

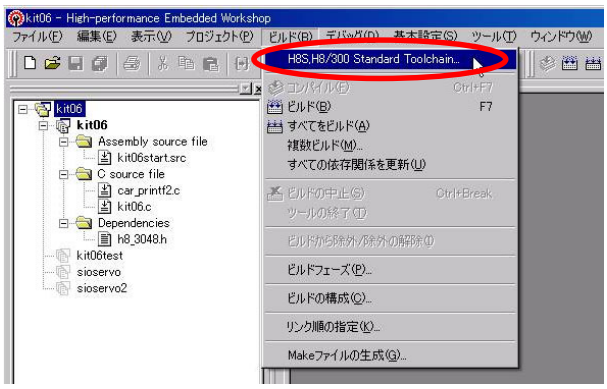
12.1 ツールチェイン

ツールチェイ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。

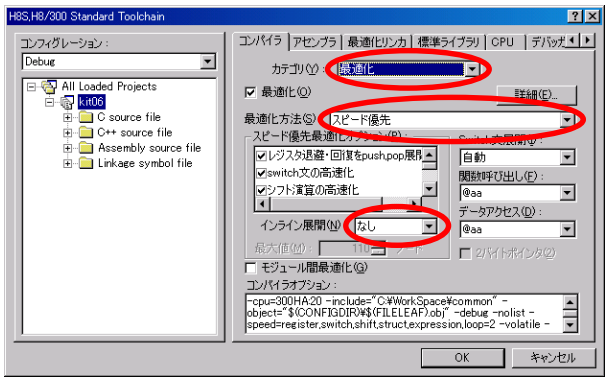
```
input kit05start,kit05
lib c:\h8n_win\3048\c\c38hae.lib
output kit05
print kit05
start V(000000)
start P,C(000100)
start B(0fef10)
exit
```

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インになります。ここでは、主に変えなければいけない設定を紹介します。

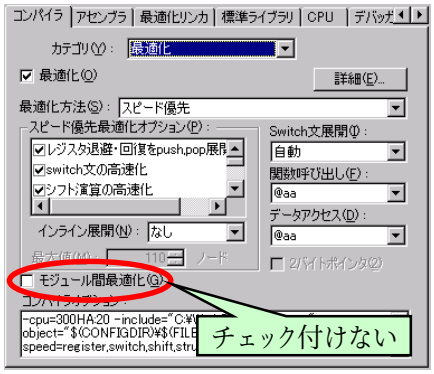
12.2 コンパイラの設定



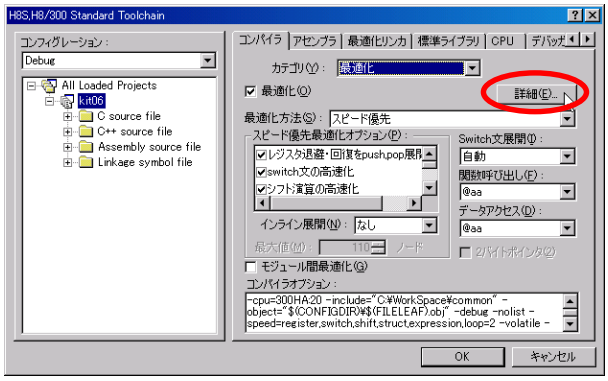
1.「ビルド→H8S,H8/300 Standard Toolchain」（ツールチェイン）を選択します。



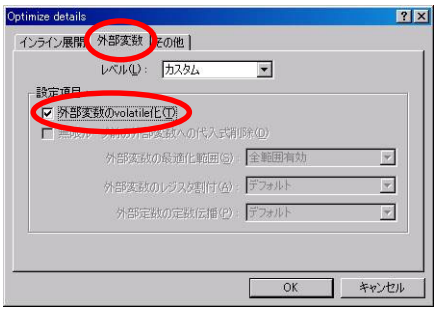
2.「カテゴリ:**最適化**」、「最適化方法:**スピード優先**」、「インライン展開:**なし**」を選択します。



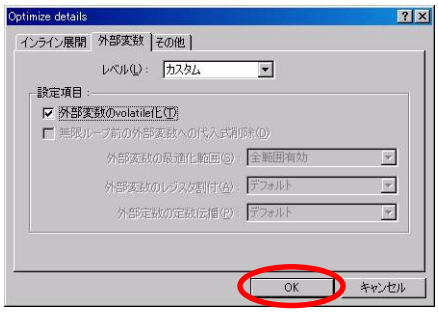
3.「モジュール間最適化」のチェックは**付けません**。最適化という文字に惑わされてチェックを付けると良くなりそうですが、チェックをつけるとプログラムが暴走することがあります（チェックをつけた場合は、さらに設定が必要です）。



4.詳細をクリックします。



5.「外部変数の volatile 化」のチェックを付けます。
※volataile についての詳しい説明は下記を参照してください。



6.OK をクリックします。

※参考資料ー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;
```

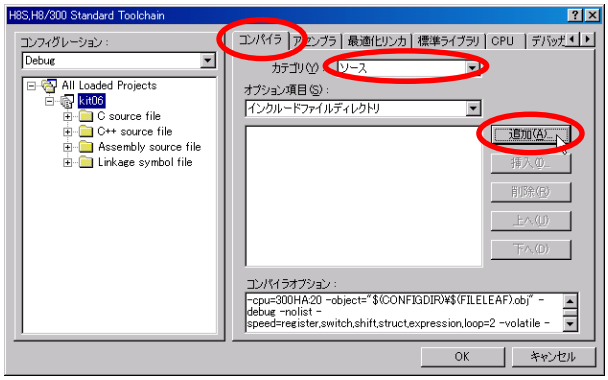
ポート A にデータを3回連続して書き込んでします。コンパイラは、同じ場所に連続して値を書き込んでいるので、最終的な結果は PADR に 0x00 を書き込めばいいだろう、と勝手に解釈してしまいます。本来は、ポート A に出力する信号を連続して変化させたいのですが、そうなりません。
そこで、コンパイルのオプションで「外部変数の volatile 化」にチェックを付けます。**このチェックを付けることにより最適化を行わないようにします**。

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

コンパイル →

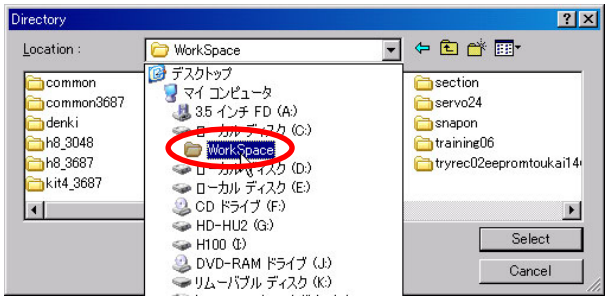
```
PADR = 0x55;
PADR = 0xaa;
PADR = 0x00;
```

と、プログラムしたとおりの動きになります。

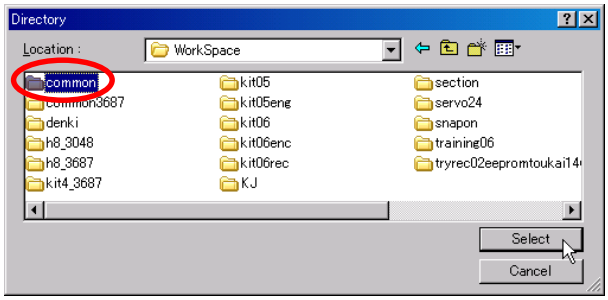


7.「コンパイラ」を選択します。「カテゴリ:**ソース**」、追加をクリックします。

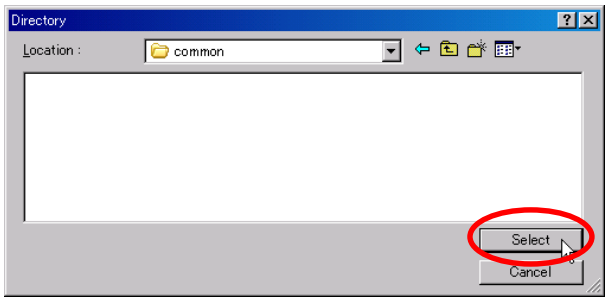
8.「相対パス:**Custom directory**」にします。続けて、ブラウズをクリックします。



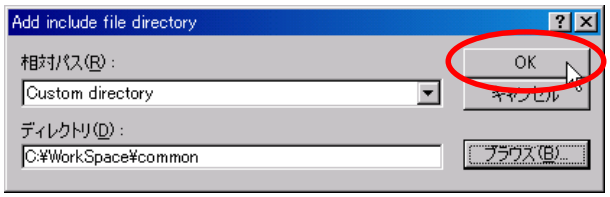
9.「C ドライブの WorkSpace」フォルダを選択します。



10.「common」フォルダをダブルクリックします。

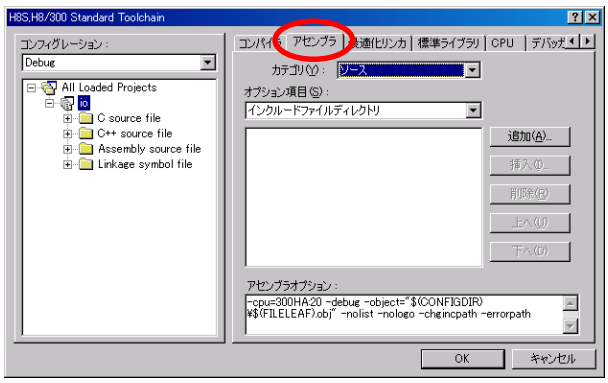


11.Select をクリックします。



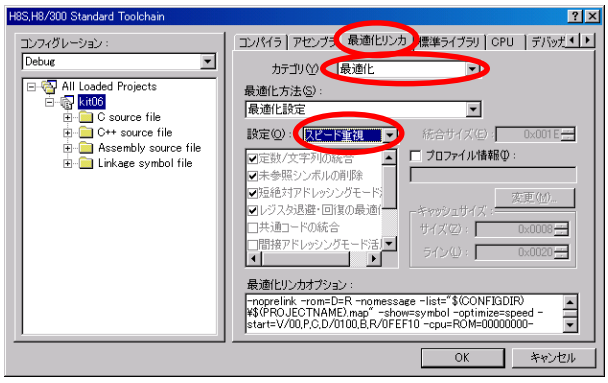
12.OK をクリックします。

12.3 アセンブラの設定

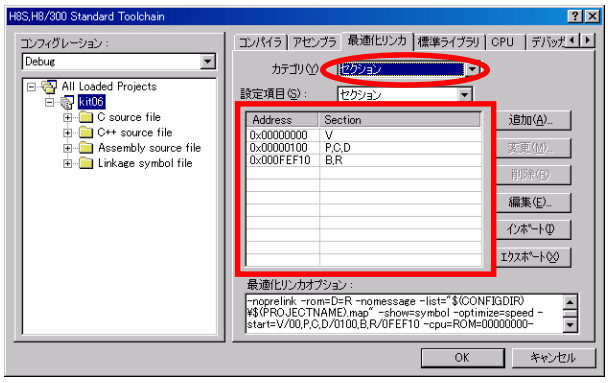


1.アセンブラで設定する項目はありません。

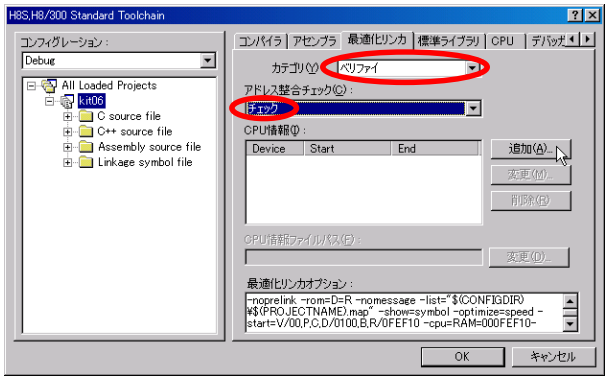
12.4 最適化リンカの設定



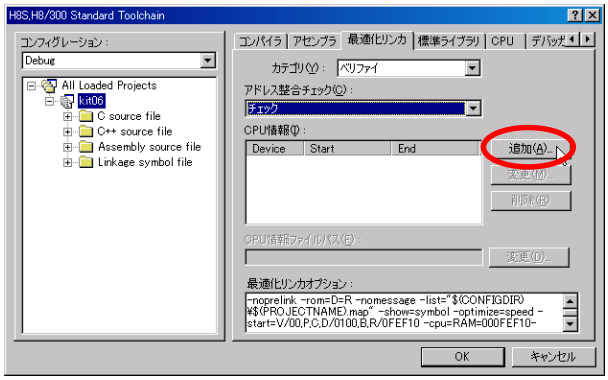
1.「最適化リンカ」を選択します。「カテゴリ：**最適化**」、「設定：**スピード重視**」にします。



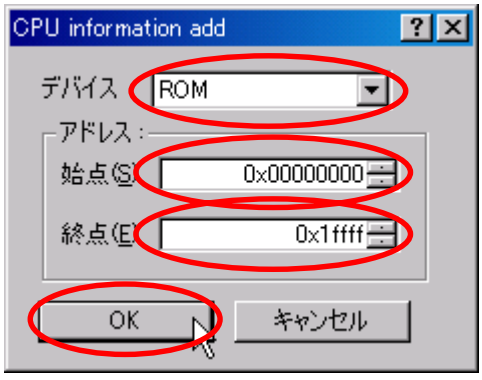
2.「カテゴリ：**セクション**」にします。□欄はプログラムに合わせてセクションを設定します。プロジェクト「kit06」は画面のようにします。



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。



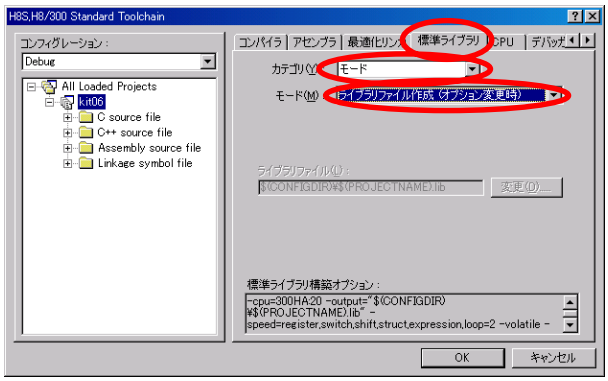
4. 追加をクリックします。
次に ROM の容量と RAM の容量を入力しますが、CPU の種類によってアドレスが変わります。



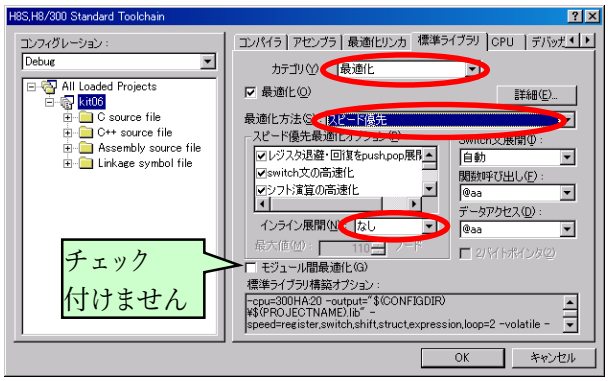
5.「デバイス：**ROM**」を選択、
「始点：**0x0000**」、「終点：**0x1ffff**」
を入力して OK をクリックします。

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。

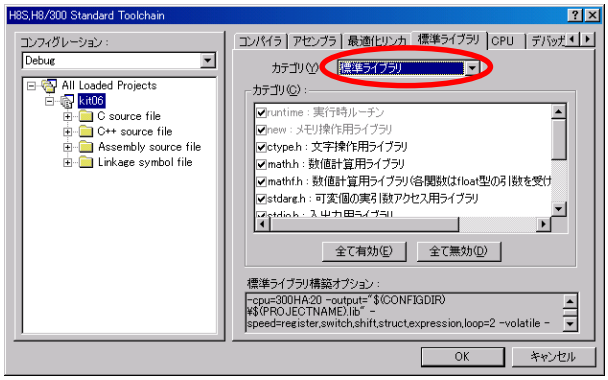
12.5 標準ライブラリの設定



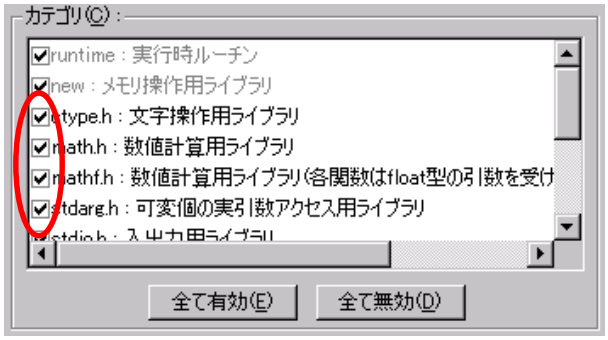
1.「標準ライブラリ」を選択します。「カテゴリ：**モード**」、「モード：**ライブラリファイル作成（オプション変更時）**」にします。



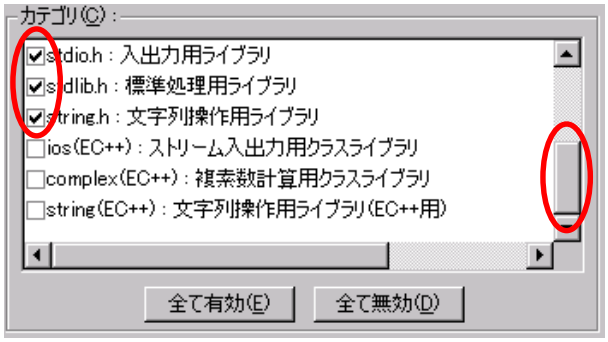
2.「カテゴリ：**最適化**」、「最適化方法：**スピード優先**」、「インライン展開：**なし**」を選択します。「モジュール間最適化」のチェックは**付けません**。



3.「カテゴリ：**標準ライブラリ**」を選択します。

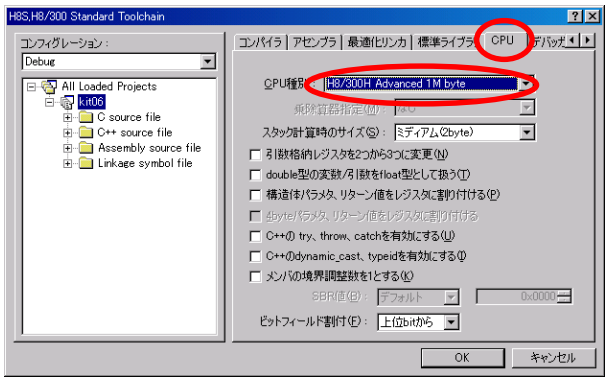


4.カテゴリ欄のファイルにチェックを付けます。

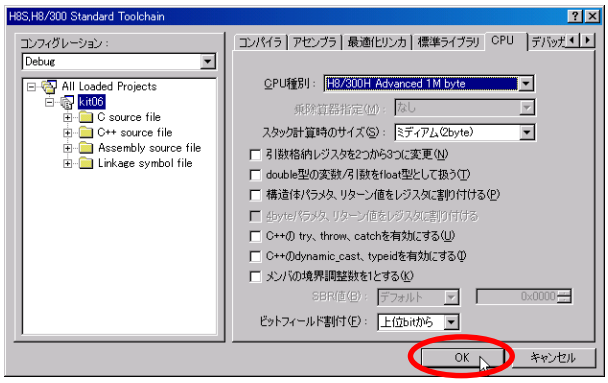


5.スクロールバーを下ろして、更にチェックを付けます。いちばん下の 3 ファイル(EC++と書かれたファイル)にはチェックを付けません。

12.6 CPU の設定



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

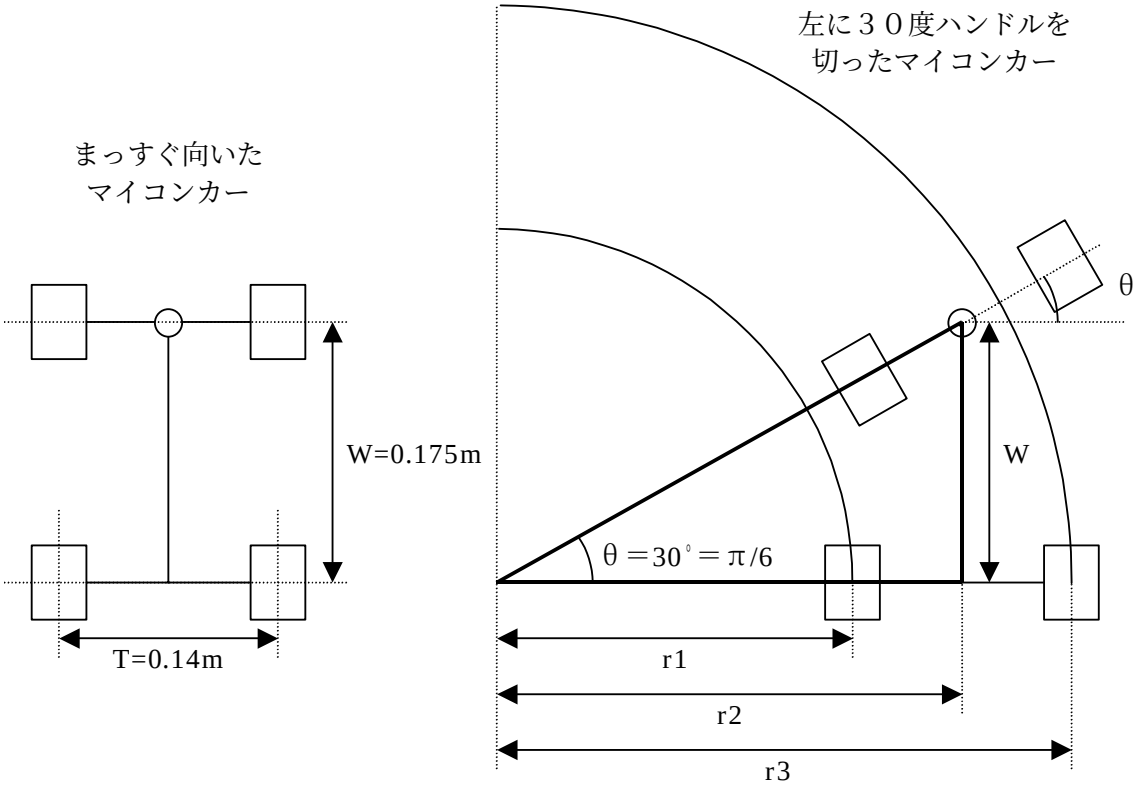


2.OK をクリックして、ツールチェインの設定を完了します。

13. モータの左右回転差の計算方法

13.1 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 0.14[m]です。
W＝ホイールベース…前輪と後輪の間隔　キットでは 0.175[m]です。

図のように、底辺 r2、高さ W、角度θの三角形の関係は次のようです。
tanθ＝W／r2

角度θ、W が分かっていますので、r2 が分かります。
r2＝W／tanθ＝0.175／tan(π/6)＝0.303[m]

内輪の半径は、
r1＝r2－T/2＝0.303-0.07＝0.233

外輪の半径は、
r3＝r2＋T/2＝0.303+0.07＝0.373

よって、外輪を 100 とすると内輪の回転数は、
r1／r3×100＝0.233／0.373×100＝62
となります。

左に 30 ° ハンドルを切ったとき、右タイヤ 100 に対して、左タイヤ 62 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( 62, 100 );
```

13.2 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	0.175	m ←ホイールベースを入力してください			
2		T	0.14	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	10.031	9.961	10.101	99
7		2	0.035	5.014	4.944	5.084	97
8		3	0.052	3.341	3.271	3.411	96
9		4	0.070	2.504	2.434	2.574	95
10		5	0.087	2.001	1.931	2.071	93
11		6	0.105	1.666	1.596	1.736	92
12		7	0.122	1.426	1.356	1.496	91
13		8	0.140	1.246	1.176	1.316	89
14		9	0.157	1.105	1.035	1.175	88
15		10	0.174	0.993	0.923	1.063	87
16		11	0.192	0.901	0.831	0.971	86
17		12	0.209	0.824	0.754	0.894	84
18		13	0.227	0.758	0.688	0.828	83
19		14	0.244	0.702	0.632	0.772	82
20		15	0.262	0.653	0.583	0.723	81
21		16	0.279	0.611	0.541	0.681	79

セル	内容	値、式の例
C1	ホイールベースを入力	キットなら 0.175
C2	トレッドを入力	キットなら 0.14
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 0.175[m]、T を 0.14[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。

```
158 :           case 0x06:
159 :               /* 少し左寄り→右へ小曲げ */
160 :               handle( 10 );
161 :               speed( 80 ,?? );          ←??が分からない
162 :               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 87%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、
右タイヤ＝左タイヤの PWM／比率×100＝80／87×100＝69

よって、右タイヤの PWM 値を **69** に設定します。kit05.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	10.031	9.961	10.101	99
2	0.035	5.014	4.944	5.084	97
3	0.052	3.341	3.271	3.411	96
4	0.070	2.504	2.434	2.574	95
5	0.087	2.001	1.931	2.071	93
6	0.105	1.666	1.596	1.736	92
7	0.122	1.426	1.356	1.496	91
8	0.140	1.246	1.176	1.316	89
9	0.157	1.105	1.035	1.175	88
10	0.174	0.993	0.923	1.063	87
11	0.192	0.901	0.831	0.971	86
12	0.209	0.824	0.754	0.894	84
13	0.227	0.758	0.688	0.828	83
14	0.244	0.702	0.632	0.772	82
15	0.262	0.653	0.583	0.723	81
16	0.279	0.611	0.541	0.681	79
17	0.297	0.573	0.503	0.643	78
18	0.314	0.539	0.469	0.609	77
19	0.331	0.509	0.439	0.579	76
20	0.349	0.481	0.411	0.551	75
21	0.366	0.456	0.386	0.526	73
22	0.384	0.433	0.363	0.503	72
23	0.401	0.413	0.343	0.483	71
24	0.419	0.393	0.323	0.463	70
25	0.436	0.376	0.306	0.446	69
26	0.454	0.359	0.289	0.429	67
27	0.471	0.344	0.274	0.414	66
28	0.488	0.329	0.259	0.399	65
29	0.506	0.316	0.246	0.386	64
30	0.523	0.303	0.233	0.373	62
31	0.541	0.291	0.221	0.361	61
32	0.558	0.280	0.210	0.350	60
33	0.576	0.270	0.200	0.340	59
34	0.593	0.260	0.190	0.330	58
35	0.611	0.250	0.180	0.320	56
36	0.628	0.241	0.171	0.311	55
37	0.645	0.232	0.162	0.302	54
38	0.663	0.224	0.154	0.294	52
39	0.680	0.216	0.146	0.286	51
40	0.698	0.209	0.139	0.279	50
41	0.715	0.201	0.131	0.271	48
42	0.733	0.195	0.125	0.265	47
43	0.750	0.188	0.118	0.258	46
44	0.768	0.181	0.111	0.251	44
45	0.785	0.175	0.105	0.245	43

※W を 0.175[m]、T を 0.14[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。

13.3 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
		W	0.175	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
		T	0.14	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
		度	rad	r2	r1	r3	r1/r3*100		内輪のスピード(自動計算)			40
		0	0				100					
		1	0.017	10.031	9.961	10.101	99		プログラム例			
		2	0.035	5.014	4.944	5.084	97		handle(15)			
		3	0.052	3.341	3.271	3.411	96		speed(50,40)			
		4	0.070	2.504	2.434	2.574	95					
		5	0.087	2.001	1.931	2.071	93					
		6	0.105	1.666	1.596	1.736	92					
		7	0.122	1.426	1.356	1.496	91					
		8	0.140	1.246	1.176	1.316	89					
		9	0.157	1.105	1.035	1.175	88					

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

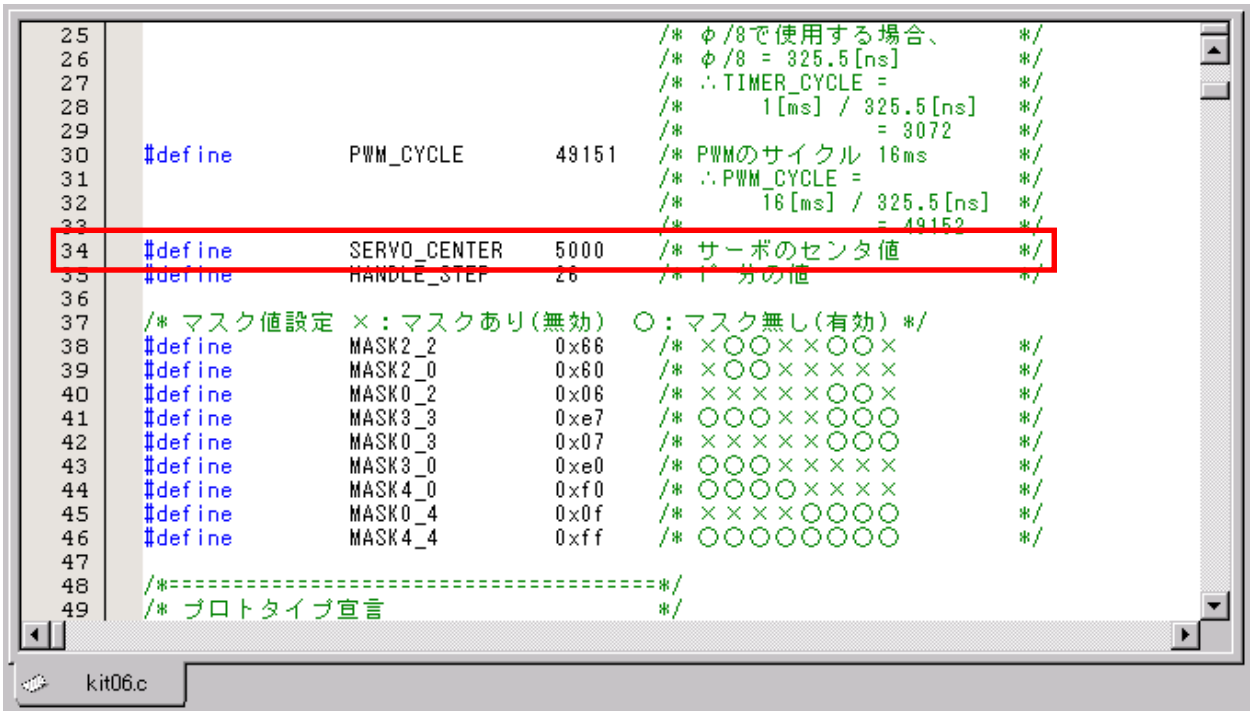
すると、〇部分に自動でハンドル角度とスピード値が入力されます。
これは便利です。実は、kit06.c の左右回転差計算もこのエクセルシートで行いました。
ただし、ホイールベースとトレッドはきちんと合わせておいてください。

14. サーボセンタと最大切れ角の調整

14.1 概要

kit06.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。「kit06.c」の 34 行



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っちとした中心になりません。

そこで、パソコンとマイコンカーを通信ケーブルで繋ぎます。調整は、

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

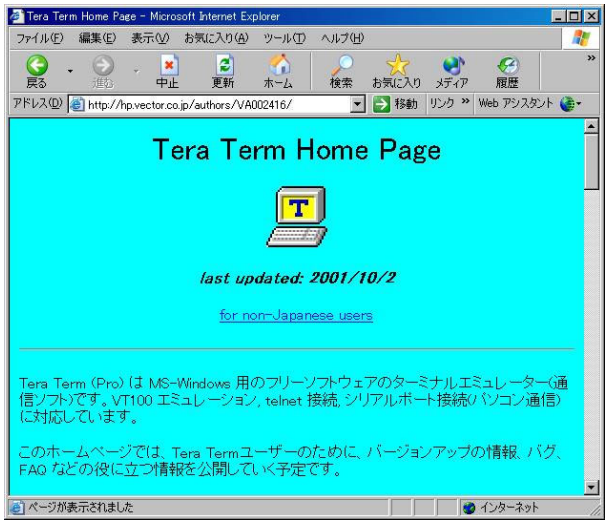
・サーボセンタの値を簡単に調整しましょう
・最大切れ角もいっしょに見つけましょう

というのが内容です。

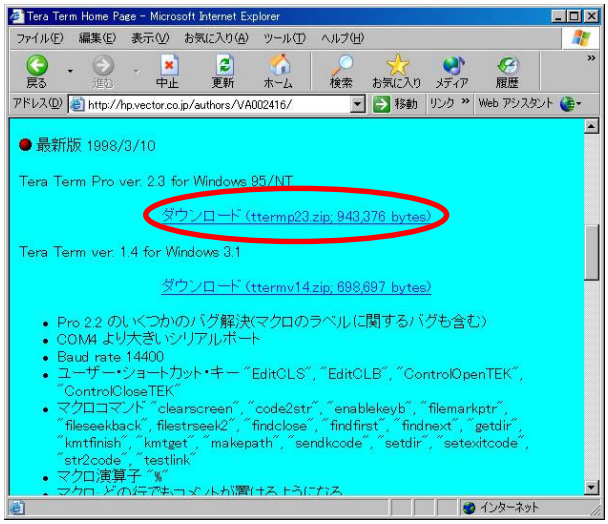
14.2 通信ソフトをインストールする

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。

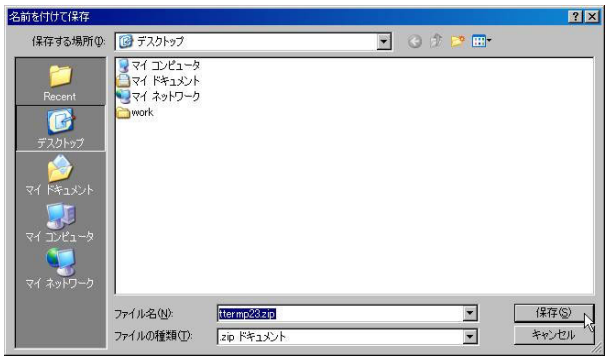
14.2.1 Tera Term Pro のインストール



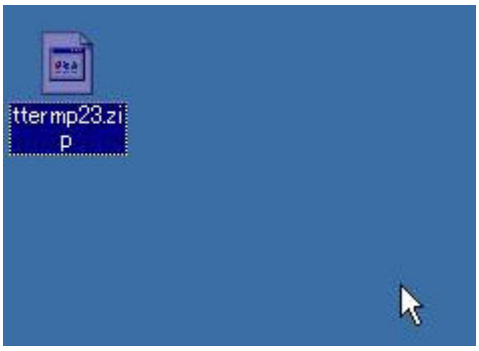
1. まず、ソフトをダウンロードします。インターネットブラウザで
http://hp.vector.co.jp/authors/VA002416/
を開きます。
または、講習会 CD がある場合は、
CD ドライブ→401 関連ソフト→ttermp23→setup.exe
を実行してください。その場合、7 へ進んでください。



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。

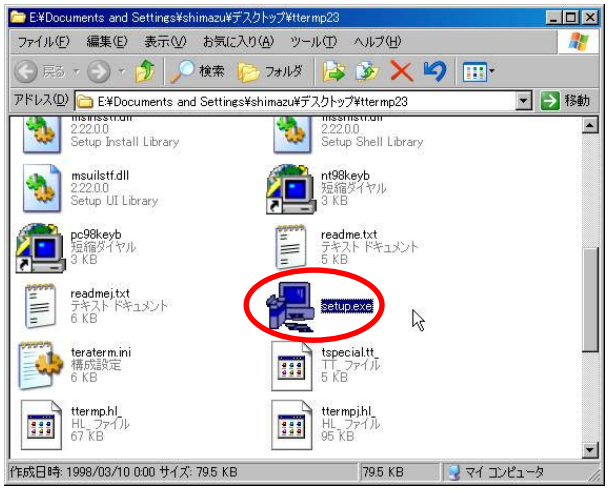


3. 保存は何処でも良いですが、ここではディスクトップに保存します。

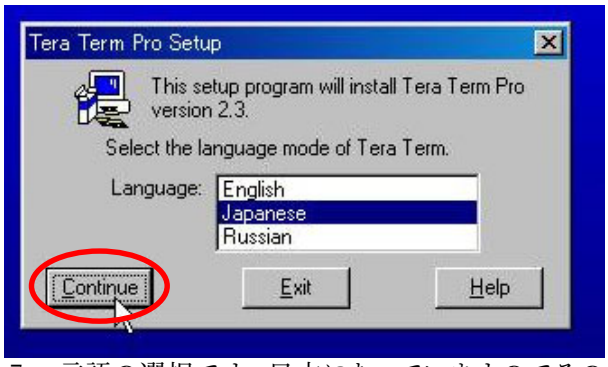


4. 保存されました。

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。



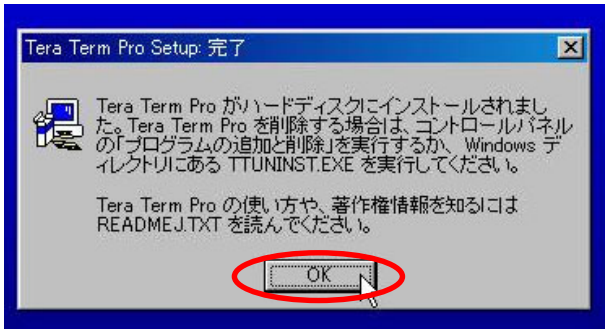
7. 言語の選択です。日本になっていますのでそのまま Continue(続ける)をクリックします。

8. 注意が出ます。Continue をクリックします。

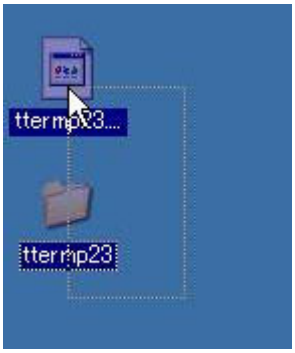
9. キーボードの選択です。そのまま Continue をクリックします。

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。

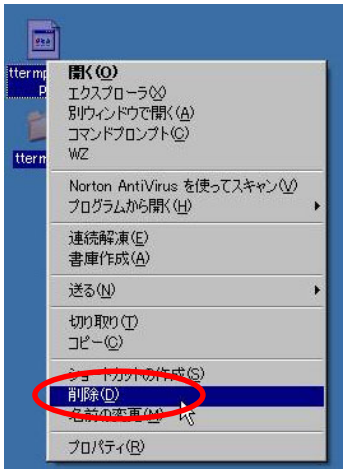
11. メニューが立ち上がります。×をクリックして閉じます。



12. OK をクリックして、インストールを完了します。

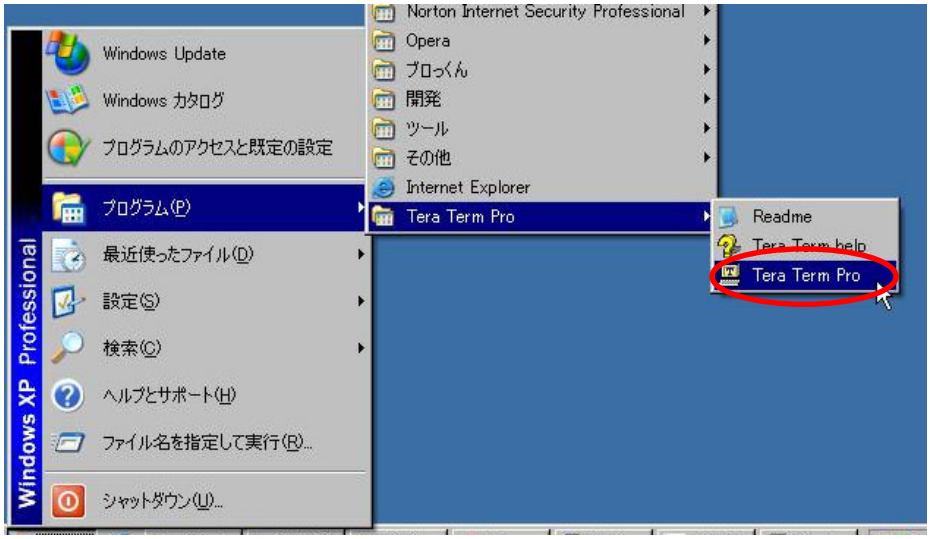


13. インターネットからファイルをダウンロードした場合、削除します。まず、ttermp23.zip と ttermp23 フォルダを選択します。CD からインストールした場合は不要です。

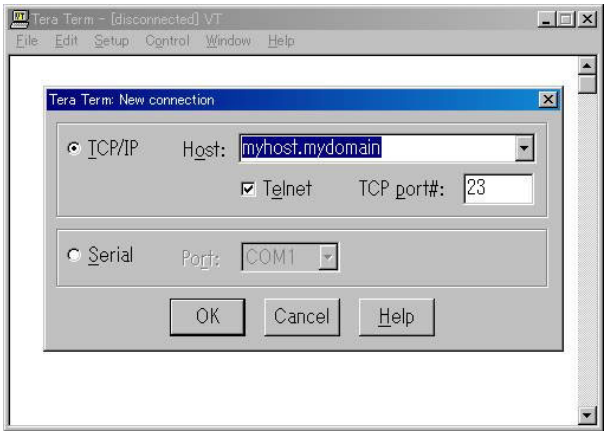


14. どちらかのファイルの上で右クリックして「削除」を選択、ファイルを削除します。

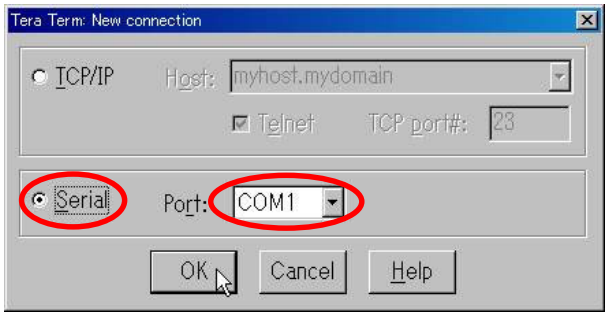
14.2.2 Tera Term Pro の使い方



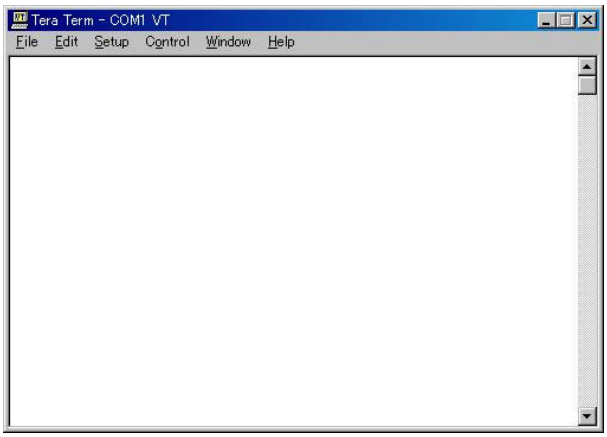
1. 「スタート→すべてのプログラム、またはプログラム→Tera Term Pro→Tera Term Pro」で Tera Term Pro が立ち上がります。



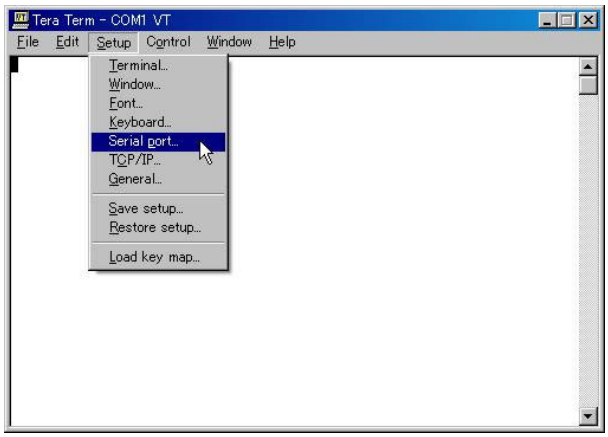
2. 最初にどこと接続するか確認する画面が出てきます。



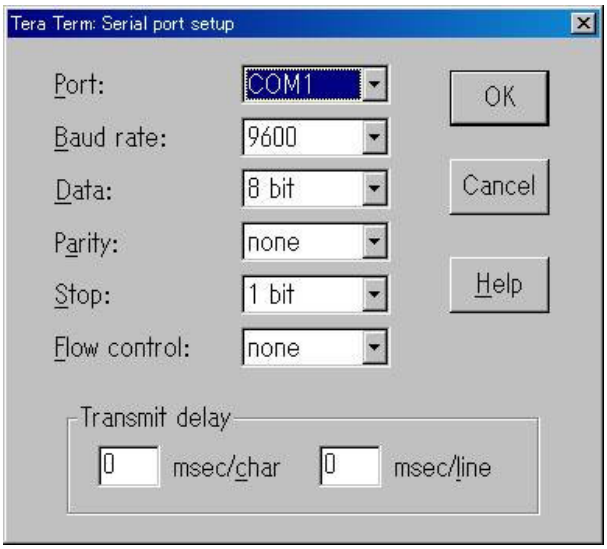
3. 「Serial」を選んで、ポート番号を選びます。選択後、OK をクリックします。
※標準では COM4 までしか選択できませんが、増やすことができます。後述します。



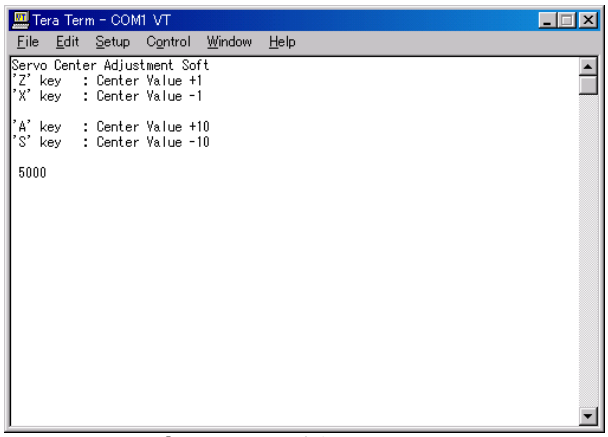
4. 立ち上がりました。
詳細設定をします。



5. 「Setup→Serial port」を選択します。

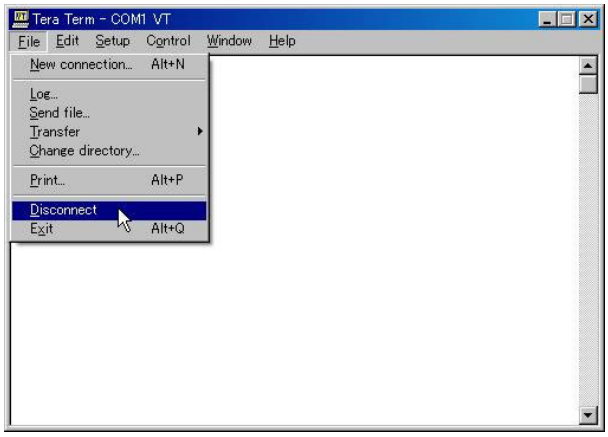


6. 通信設定を確認します。画面のように設定して、OK をクリックします。「Port」は、それぞれの通信ポートの番号に合わせてください。

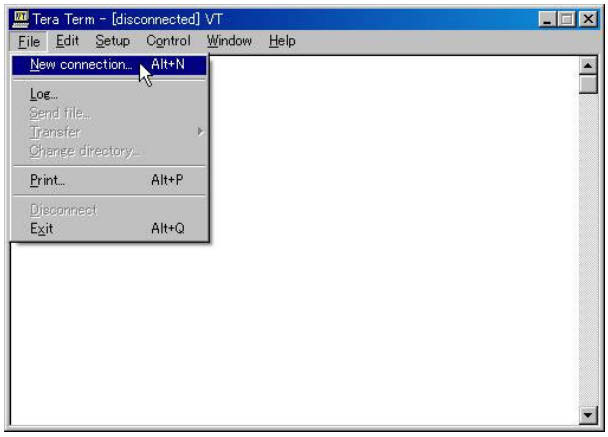


※プロジェクト「sioservo」の例です。

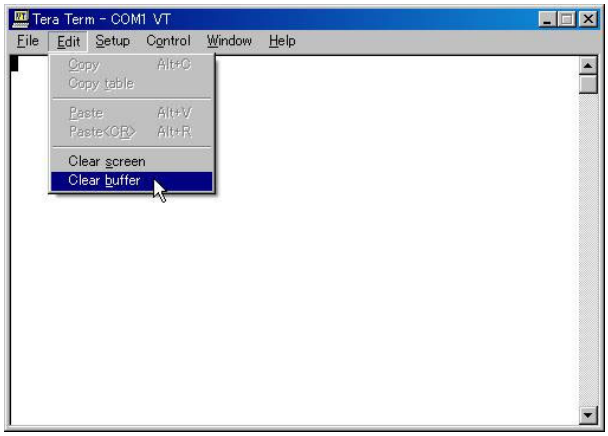
7. 通信するプログラムが入っているマイコンカーの電源を入れると、マイコンカーからメッセージが送られてきます。表示されれば接続成功です。



8. 一時的な切断は、「File→Disconnect」で切断できます。



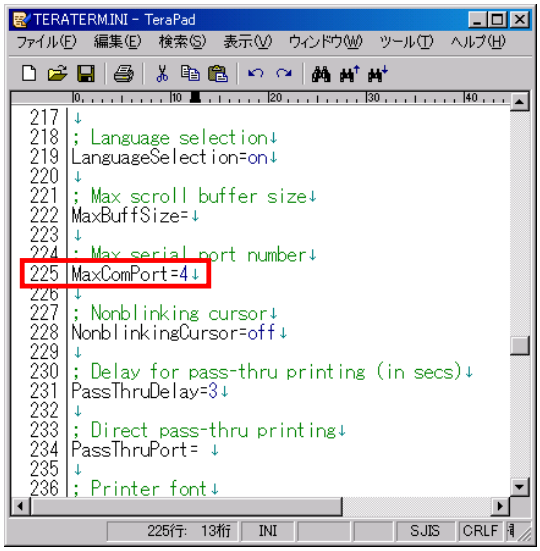
9. 再接続する場合は、「File→New connection」で接続できます。



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

14.2.3 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

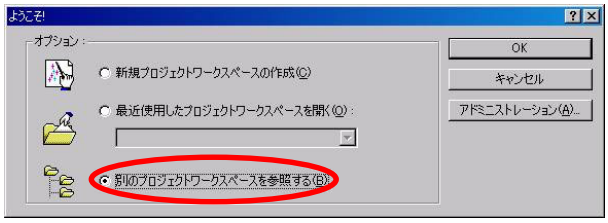
MaxComPort=4

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。

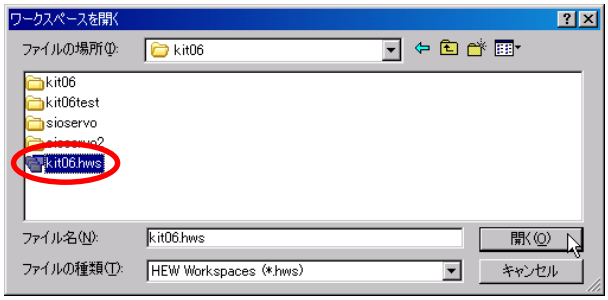
14.3 サーボのセンタを調整する

ワークスペース「kit06」を開きます。

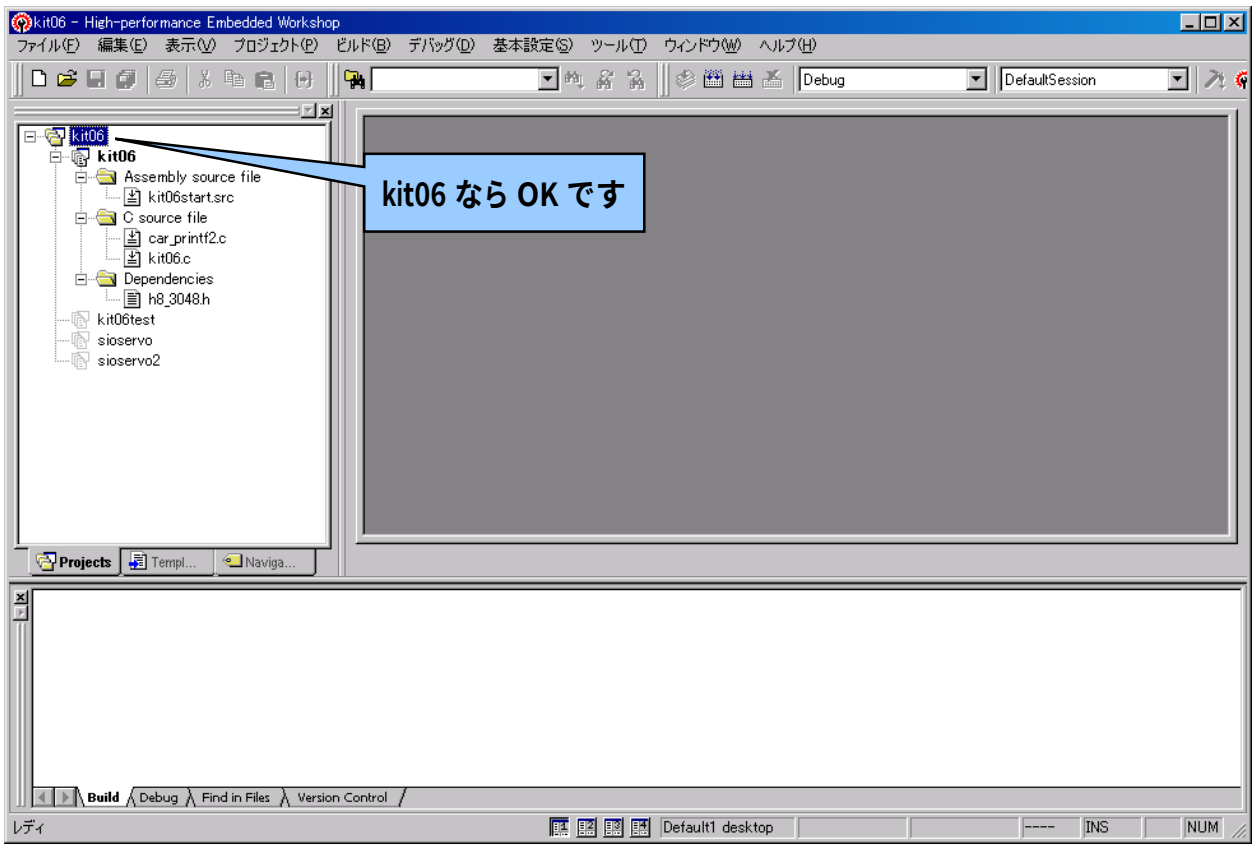
1.ルネサス統合開発環境を実行します。



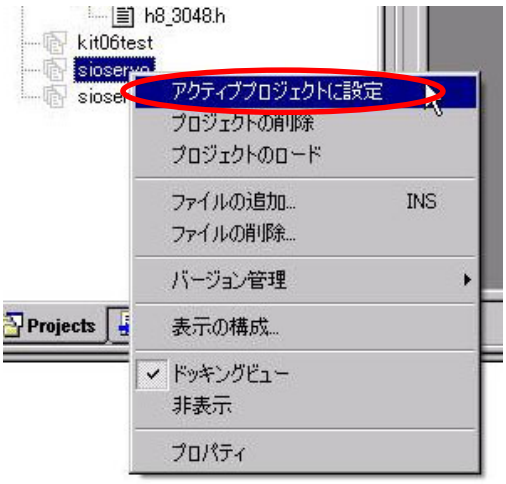
2.「別のプロジェクトワークスペースを参照する」を選択します。



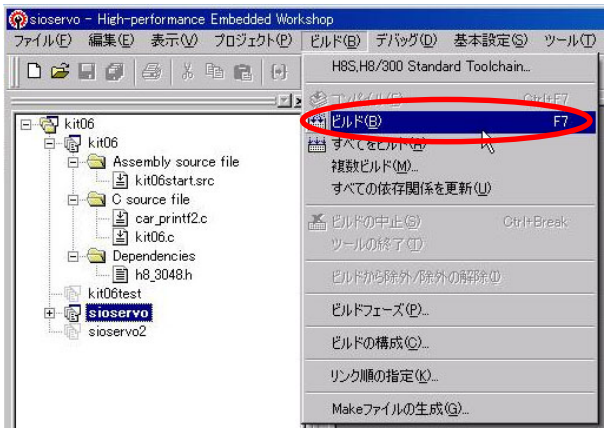
3.Cドライブ→Workspace→kit06 の「kit06.hws」を選択します。



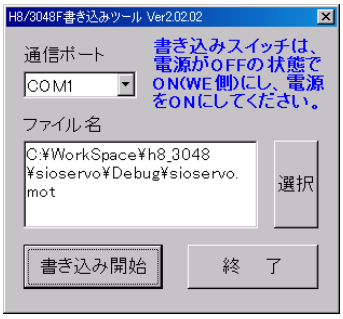
4.kit06 というワークスペースが開かれます。



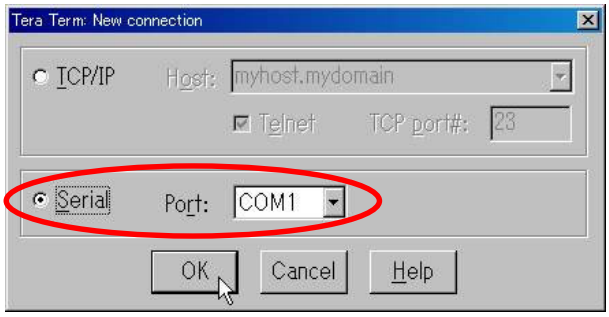
5.プロジェクト「sioservo」をアクティブプロジェクトに設定します。



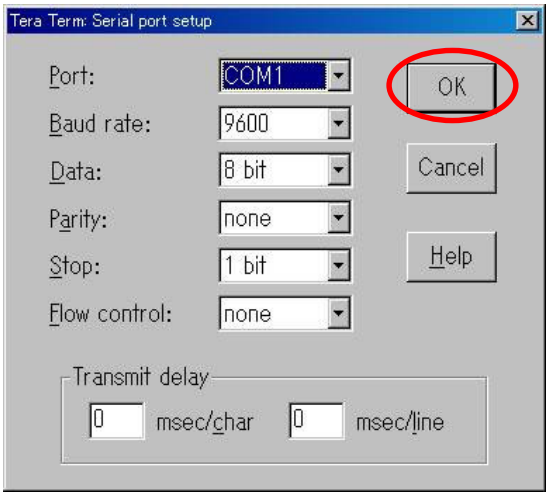
6.「ビルド→ビルド」でビルドします。MOT ファイルができ上がります。



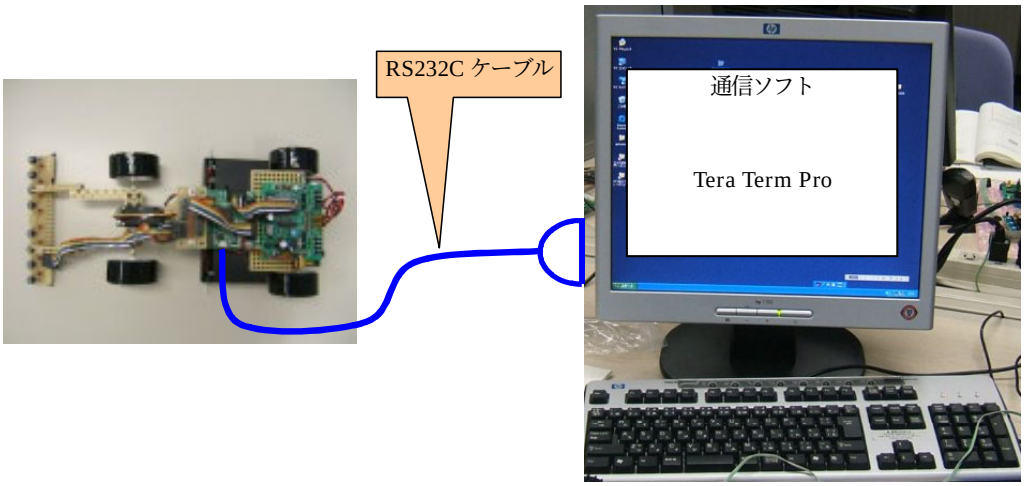
7.ルネサス統合開発環境の「ツール→CpuWrite」で書き込みソフトを立ち上げ、プログラムを書き込みます。転送終了後、CPU ボードの電源を OFF にして書き込みスイッチを内側にしておきます。ケーブルは繋いだままです。



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。



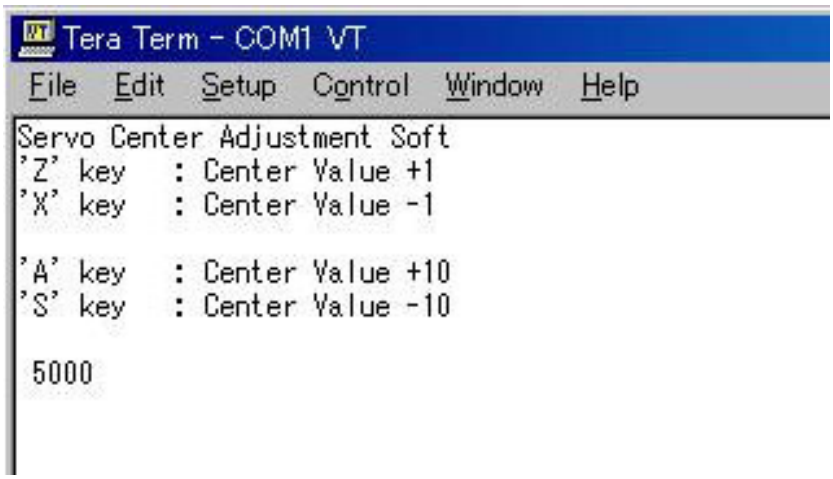
9.設定は、上記のようにします。OK をクリックして TeraTermPro の準備が整いました。



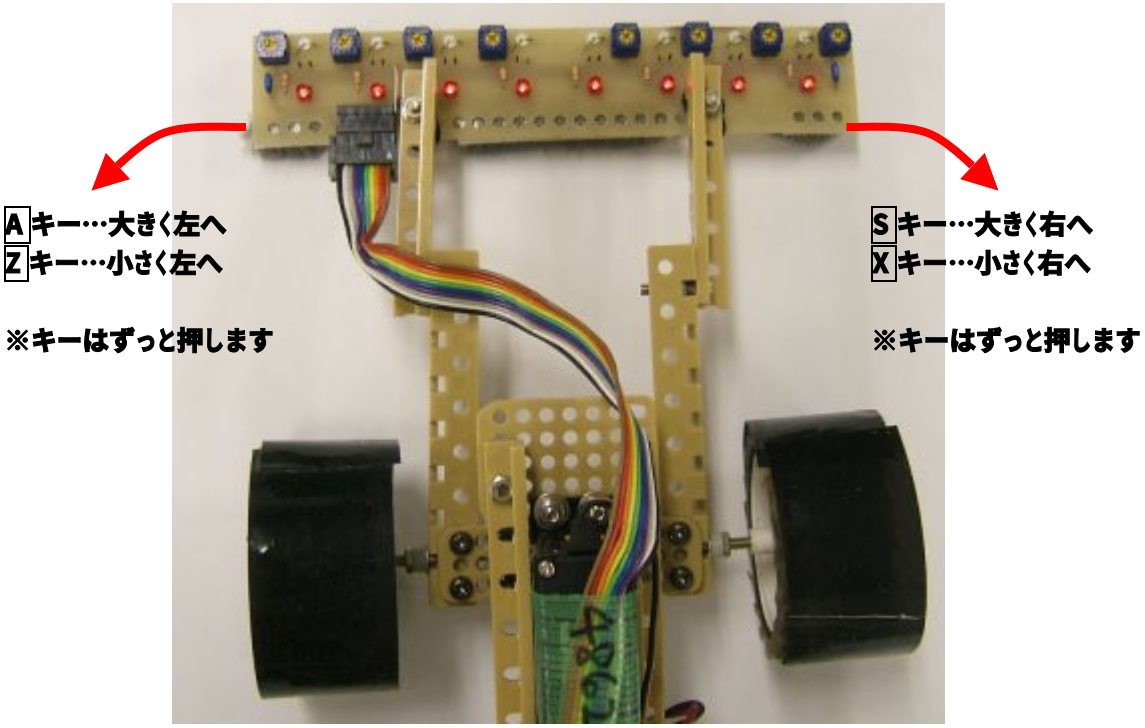
10.接続、設定を確かめます。

・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

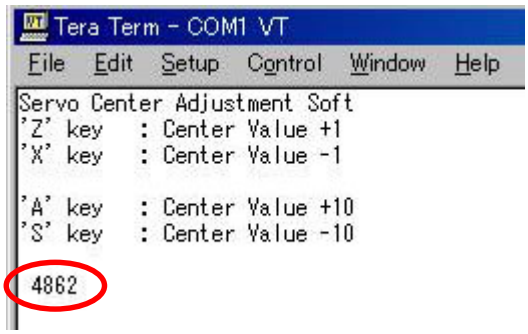
OK なら次に進みます。

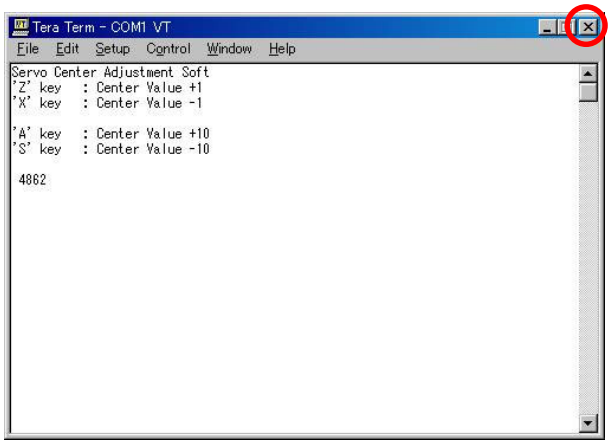


11.マイコンカーの電源を入れるとメッセージが表示されます。表示されない場合は、ケーブルの接続やマイコンカーの電池、書き込みスイッチを戻したか、通信ポートの番号、書き込んだプログラムが本当にプロジェクト「sioservo」の sioservo.mot を書き込んだかなど確かめてください。



12.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。**



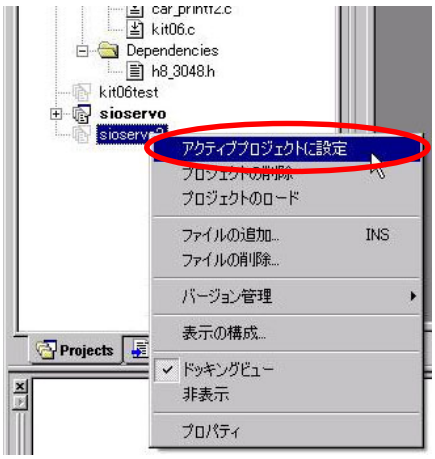


13.まっすぐに調整できたら、Tera Term Pro の数字を見ます。今回は「**4862**」とでました。これがこのマイコンカーの**サーボセンタ（SERVO_CENTER）**の値です。メモしておきます。

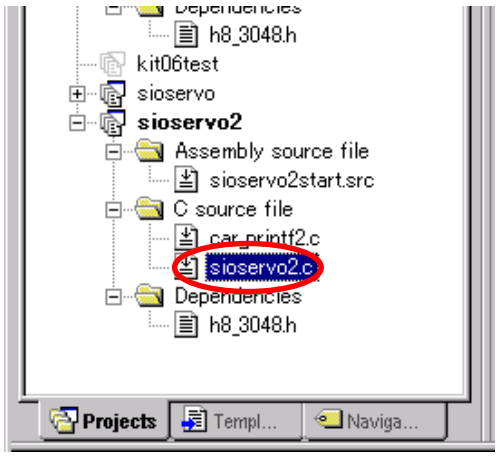
14.次は、プロジェクト「sioservo2」に進みます。
×をクリックしてソフトを終了しておきます。
マイコンカーの電源も切ります。

14.4 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。



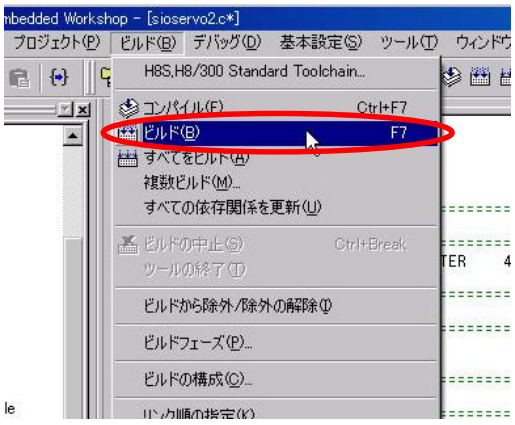
2.「sioservo2.c」をダブルクリックしてエディタウィンドウを開きます。

```
20  #include    <stdio.h>
21  #include    <machine.h>
22  #include    "h8_3048.h"
23
24  /*======================================*/
25  /* シンボル定義                         */
26  /*======================================*/
27  #define         SERVO_CENTER    5000    /*
28
29  /*======================================*/
30  /* プロトタイプ宣言                     */
31  /*======================================*/
32  void init( void );
33
34  /*======================================*/
35  /* グローバル変数の宣言                 */
36  /*======================================*/
37  int             servo_angle;            /*
38
```

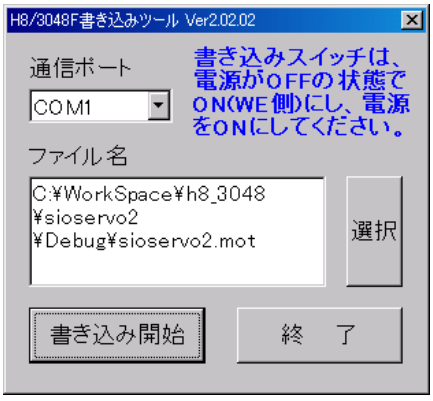
3. 27 行に
SERVO_CENTER　5000
という記述があります。

```
20  #include    <stdio.h>
21  #include    <machine.h>
22  #include    "h8_3048.h"
23
24  /*======================================*/
25  /* シンボル定義                         */
26  /*======================================*/
27  #define         SERVO_CENTER    4862    /*
28
29  /*======================================*/
30  /* プロトタイプ宣言                     */
31  /*======================================*/
32  void init( void );
33
34  /*======================================*/
35  /* グローバル変数の宣言                 */
36  /*======================================*/
37  int             servo_angle;            /*
38
```

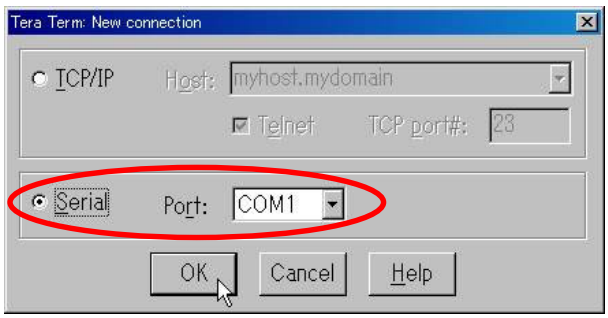
4.先ほど見つけたサーボセンタ値に書き換えます。画面は、例として「**4862**」に書き換えています。

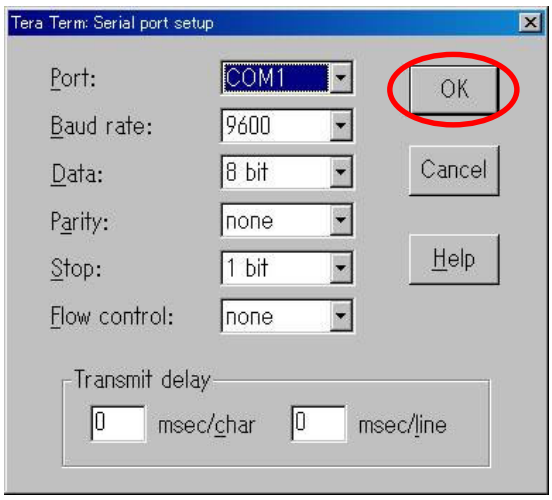


5.「ビルド→ビルド」で MOT ファイルを作成します。



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。

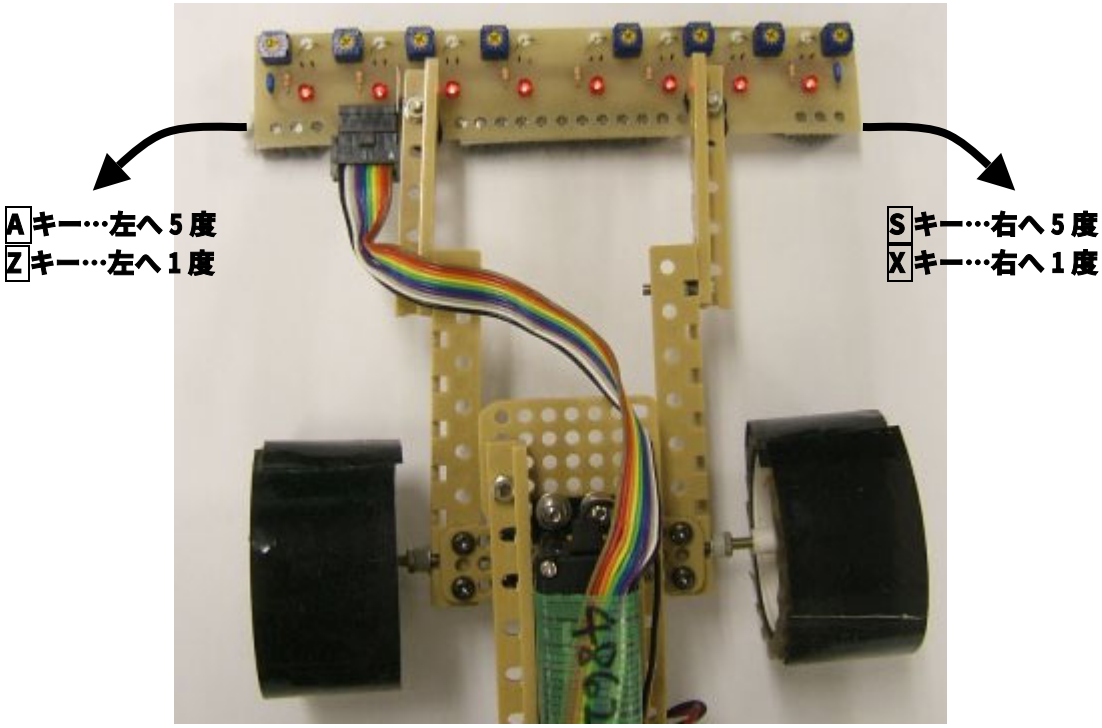




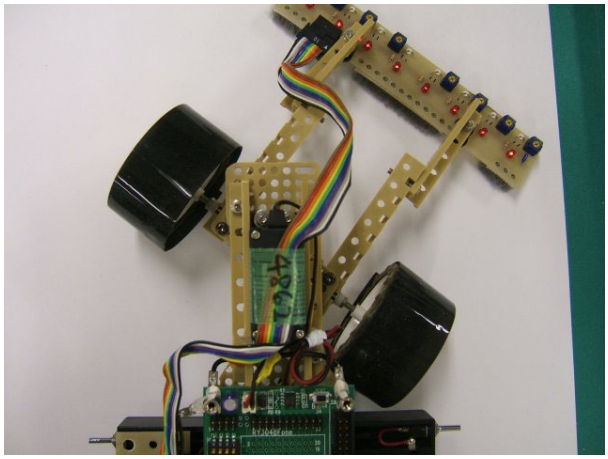
7.「スタート→プログラム→Tera Term Pro→Tera Term Pro」で Tera Term Pro を立ち上げます。Serial を選択して、ポートを書き込みポートの番号に合わせます。

8.設定は、上記のようにします。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。

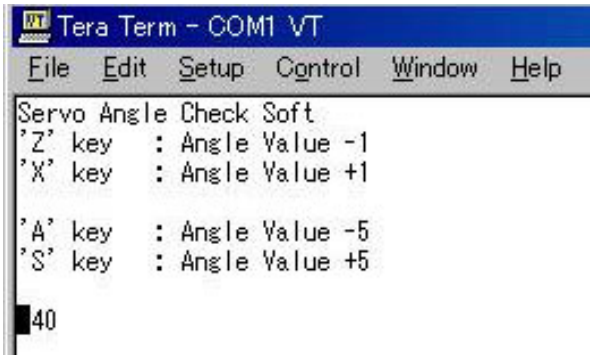
9.マイコンカーの電源を入れるとメッセージが表示されます。表示されない場合は、ケーブルの接続やマイコンカーの電池、書き込みスイッチを戻したか、通信ポートの番号、書き込んだプログラムが本当に「sioservo2.mot」かなど確かめてください。



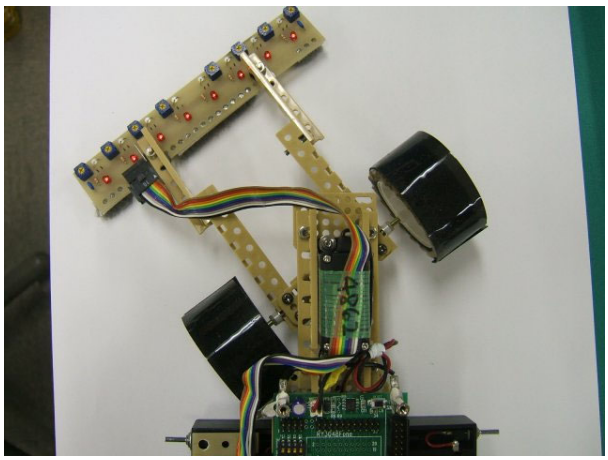
10.A、S、Z、X キーをそれぞれ押すとサーボが動きます。**右はどこまでハンドルを曲げることができるかを調べます。左も同様に調べます。**



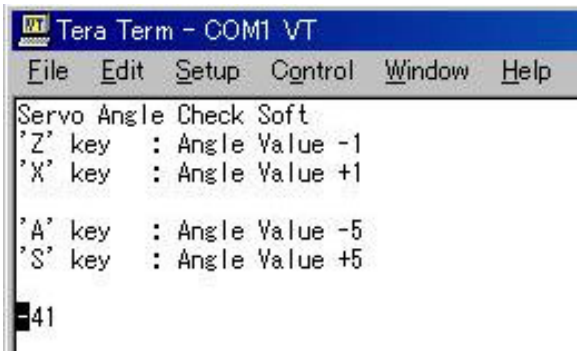
11.まず S キー、X キーで右の限界を見つけます。タイヤを回して回るか確かめてください。シャーシにぶつかるようなら Z キーでもう少し小さくしてください。



12.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。



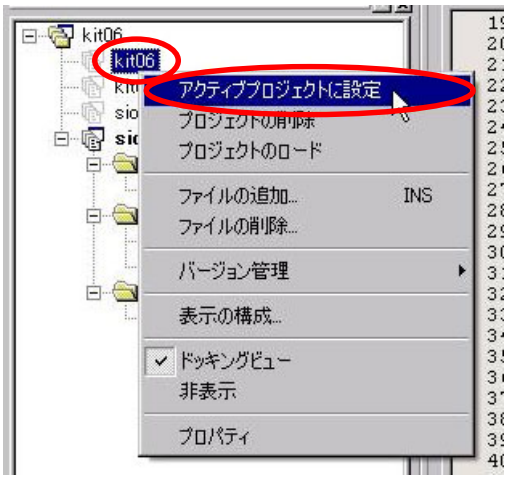
13.今度は A キー、Z キーで逆の左の限界を見つけます。こちらもタイヤを回して回るか確かめてください。シャーシにぶつかるようなら Z キーでもう少し小さくしてください。



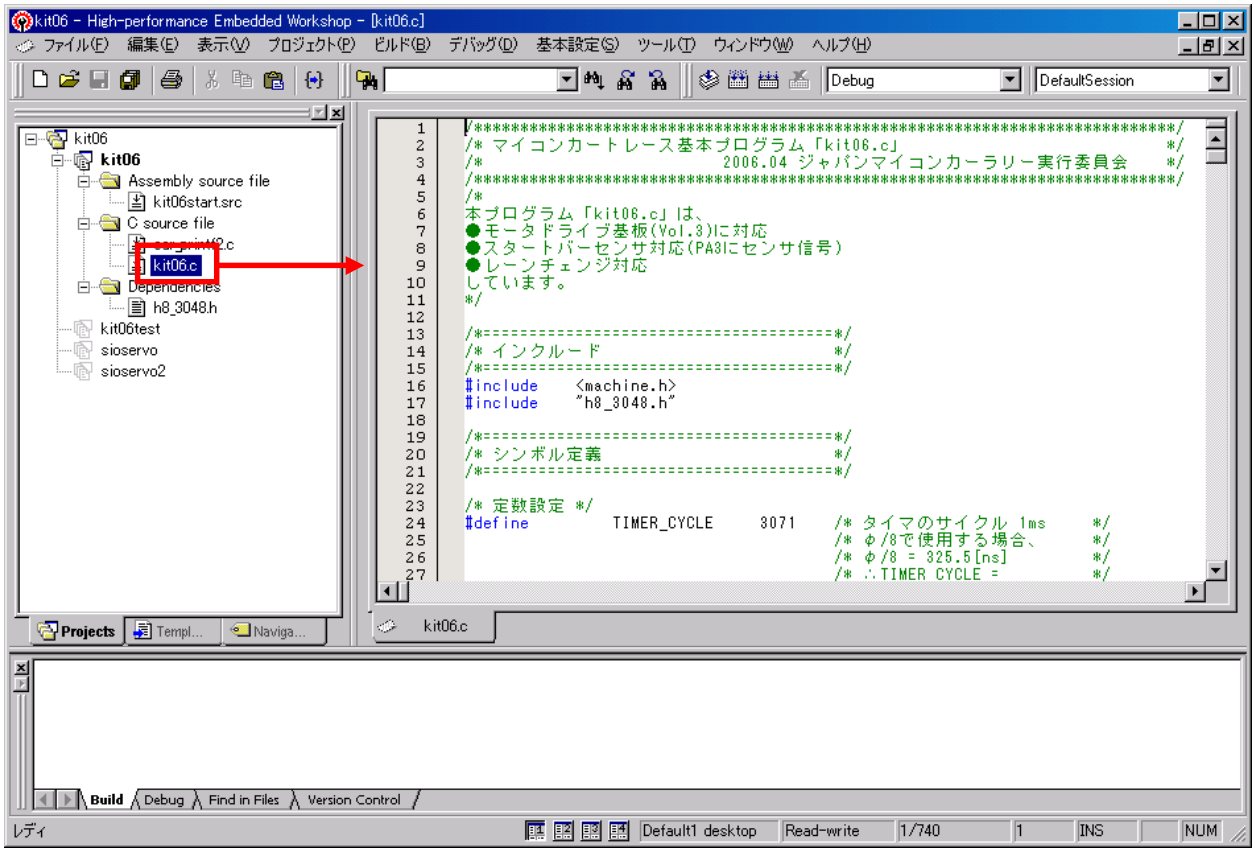
14.Tera Term Pro の数値を見ます。これが現在ハンドルを左へ曲げている角度です。**-41 度曲げていると言うことが分かりました。**

14.5 「kit06.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「kit06.c」へ書き込みます。このファイルは、プロジェクト「kit06」内にあります。



1.ワークスペース「kit06」のプロジェクト「kit06」をアクティブプロジェクトに設定します。



2.「kit06.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	kit06.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	34 行	5000	4862
左の最大角度	284 行	-38	-41
右の最大角度	293 行	38	40

まず、サーボセンタです。

```
34 :  #define        SERVO_CENTER     5000     /* サーボのセンタ値          */
```

↓

```
34 :  #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。

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

↓

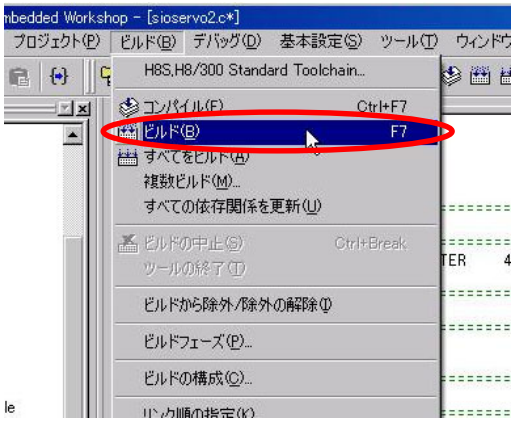
```
284 :                  handle( -41 );
```

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。

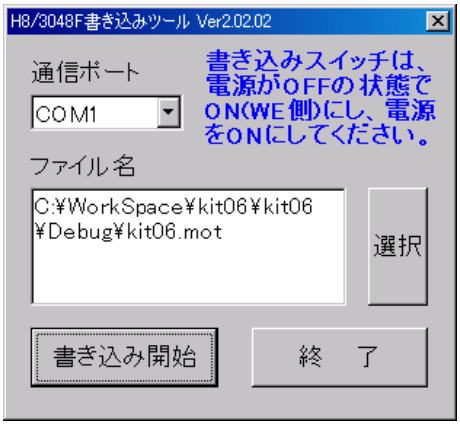
```
293 :                   handle( 38 );
```

↓

```
293 :                   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、kit06.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！

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

14.6.1 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。

14.6.2 変数の宣言

```
33 :  /*====================================*/
34 :  /* グローバル変数の宣言               */
35 :  /*====================================*/
36 :  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。

14.6.3 プログラムスタートのメッセージ

```
52 :      printf(
53 :          "Servo Center Adjustment Soft\n"
54 :          "'Z' key   : Center Value +1\n"
55 :          "'X' key   : Center Value -1\n"
56 :          "\n"
57 :          "'A' key   : Center Value +10\n"
58 :          "'S' key   : Center Value -10\n"
59 :          "\n"
60 :      );
61 :      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。

14.6.4 メイン関数

```
 63 :      while( 1 ) {
 64 :          ITU4_BRB = servo_offset;
 65 :
 66 :          i = get_sci( &c );
 67 :          if( i == 1 ) {
 68 :              switch( c ) {
 69 :              case 'Z':
 70 :              case 'z':
 71 :                  servo_offset++;
 72 :                  if( servo_offset > 10000 ) servo_offset = 10000;
 73 :                  printf( "%5d\r", servo_offset );
 74 :                  break;
 75 :
 76 :              case 'A':
 77 :              case 'a':
 78 :                  servo_offset += 10;
 79 :                  if( servo_offset > 10000 ) servo_offset = 10000;
 80 :                  printf( "%5d\r", servo_offset );
 81 :                  break;
 82 :
 83 :              case 'X':
 84 :              case 'x':
 85 :                  servo_offset--;
 86 :                  if( servo_offset < 1000 ) servo_offset = 1000;
 87 :                  printf( "%5d\r", servo_offset );
 88 :                  break;
 89 :
 90 :              case 'S':
 91 :              case 's':
 92 :                  servo_offset -= 10;
 93 :                  if( servo_offset < 1000 ) servo_offset = 1000;
 94 :                  printf( "%5d\r", servo_offset );
 95 :                  break;

 96 :
 97 :              default:
 98 :                  break;
 99 :              }
100 :          }
101 :      }
```

64 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。

66 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。

67 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。

14.7 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

14.7.1 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。

14.7.2 変数の宣言

```
34 :  /*====================================*/
35 :  /* グローバル変数の宣言               */
36 :  /*====================================*/
37 :  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。

14.7.3 プログラムスタートのメッセージ

```
52 :      servo_angle = 0;
53 :      printf(
54 :          "Servo Angle Check Soft\n"
55 :          "'Z' key   : Angle Value -1\n"
56 :          "'X' key   : Angle Value +1\n"
57 :          "\n"
58 :          "'A' key   : Angle Value -5\n"
59 :          "'S' key   : Angle Value +5\n"
60 :          "\n"
61 :      );
62 :      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う意味です。

14.7.4 メイン関数

```
 64 :      while( 1 ) {
 65 :          ITU4_BRB = SERVO_CENTER - servo_angle * 26;
 66 :
 67 :          i = get_sci( &c );
 68 :          if( i == 1 ) {
 69 :              switch( c ) {
 70 :              case 'Z':
 71 :              case 'z':
 72 :                  servo_angle--;
 73 :                  if( servo_angle < -50 ) servo_angle = -50;
 74 :                  printf( "%3d\r", servo_angle );
 75 :                  break;
 76 :
 77 :              case 'X':
 78 :              case 'x':
 79 :                  servo_angle++;
 80 :                  if( servo_angle > 50 ) servo_angle = 50;
 81 :                  printf( "%3d\r", servo_angle );
 82 :                  break;
 83 :
 84 :              case 'A':
 85 :              case 'a':
 86 :                  servo_angle -= 5;
 87 :                  if( servo_angle < -50 ) servo_angle = -50;
 88 :                  printf( "%3d\r", servo_angle );
 89 :                  break;
 90 :
 91 :              case 'S':
 92 :              case 's':
 93 :                  servo_angle += 5;
 94 :                  if( servo_angle > 50 ) servo_angle = 50;
 95 :                  printf( "%3d\r", servo_angle );
 96 :                  break;
 97 :
 98 :              default:
 99 :                  break;
100 :              }
101 :          }
102 :      }
```

65 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、

```
ITU4_BRB = SERVO_CENTER - 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。

15. 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
High-performance Embedded Workshop V.4.00 ユーザーズマニュアル Rev.3.00
・(株) ルネサス テクノロジ 半導体トレーニングセンター C言語入門コーステキスト 第 1 版
・(株) オーム社 H8 マイコン完全マニュアル 藤澤幸穂著 第1版
・電波新聞社 マイコン入門講座 大須賀威彦著 第 1 版
・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

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

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

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

http://japan.renesas.com/

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

※リンクは、2007 年 4 月現在の情報です。