ミニマイコンカーVer.2
サーボステアリング 4 輪セット
C 言語プログラム解説
マニュアル

第 1.02 版
2016.02.08
株式会社日立ドキュメントソリューションズ

注 意 事 項 (rev.6.0H)

著作権

・本マニュアルに関する著作権は株式会社日立ドキュメントソリューションズに帰属します。
・本マニュアルは著作権法および、国際著作権条約により保護されています。

禁止事項

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

転載、複製

本マニュアルの転載、複製については、文書による株式会社日立ドキュメントソリューションズの事前の承諾が必要です。

責任の制限

本マニュアルに記載した情報は、正確を期すため、慎重に制作したものですが万一本マニュアルの記述誤りに起因する損害が生じた場合でも、株式会社日立ドキュメントソリューションズ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、株式会社日立ドキュメントソリューションズは、予告なしに、本マニュアルに記載した情報または仕様を変更することがあります。製作に当たりましては、最新の内容を確認いただきますようお願いします。
・すべての商標および登録商標は、それぞれの所有者に帰属します。

連絡先

株式会社 日立ドキュメントソリューションズ
〒135-0016 東京都江東区東陽六丁目 3 番 2 号 イースト 21 タワー
E-mail:himdx.m-carrally.dd@hitachi.com

目　次

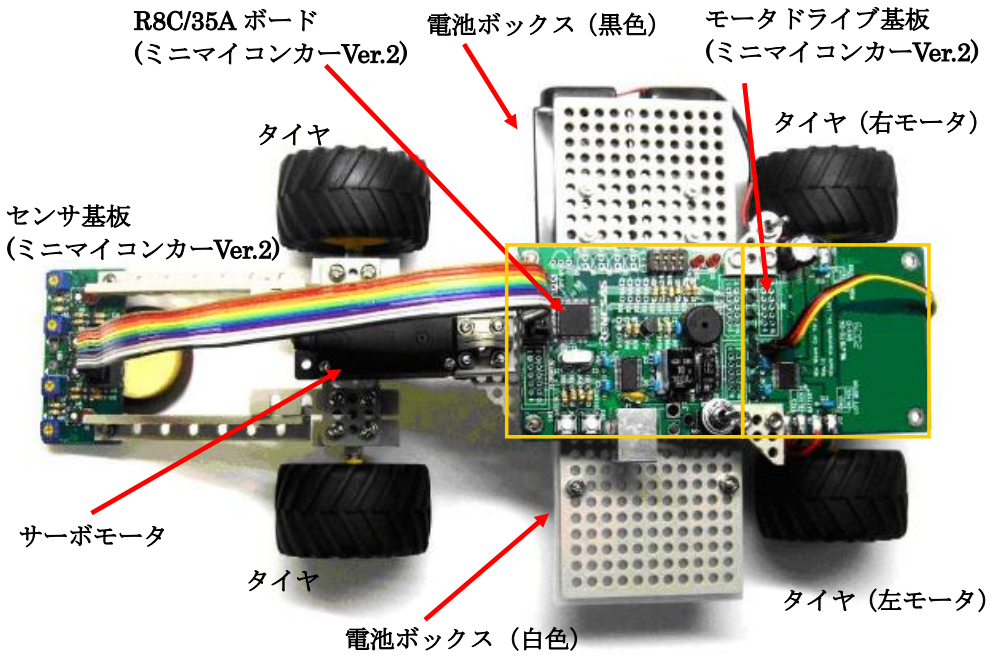
1 概要

本マニュアルでは、サーボステアリング 4 輪セットを追加したミニマイコンカーVer.2(以下、「4 輪ミニマイコンカー」という)の仕様とルネサス エレクトロニクス(株)製マイコン、R8C/35A を使用した 4 輪ミニマイコンカーの制御プログラムについて説明します。

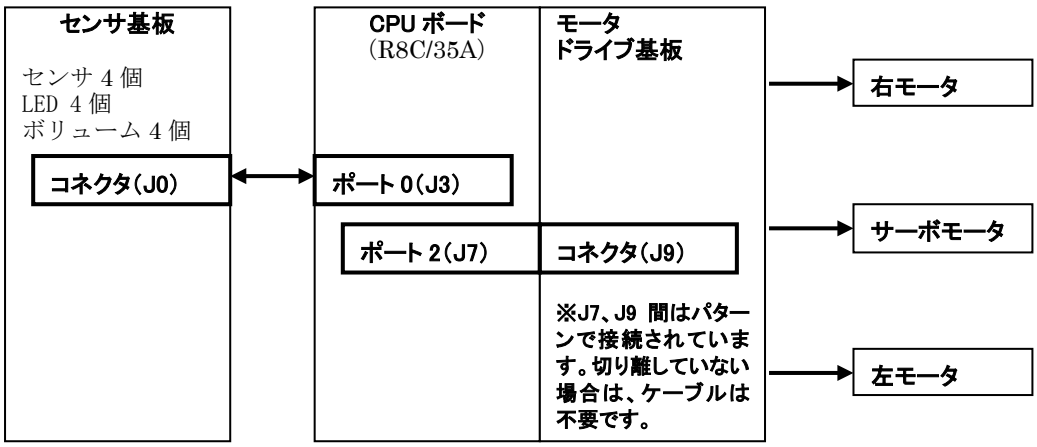
※ミニマイコンカーVer.2 の詳細な説明については、「R8C/35A マイコン実習マニュアル」を参照してください。「R8C/35A マイコン実習マニュアル」は、こちらの URL(http://www2.himdx.net/mcr/)よりダウンロードできます。

2 4 輪ミニマイコンカーの仕様

2.1 外観



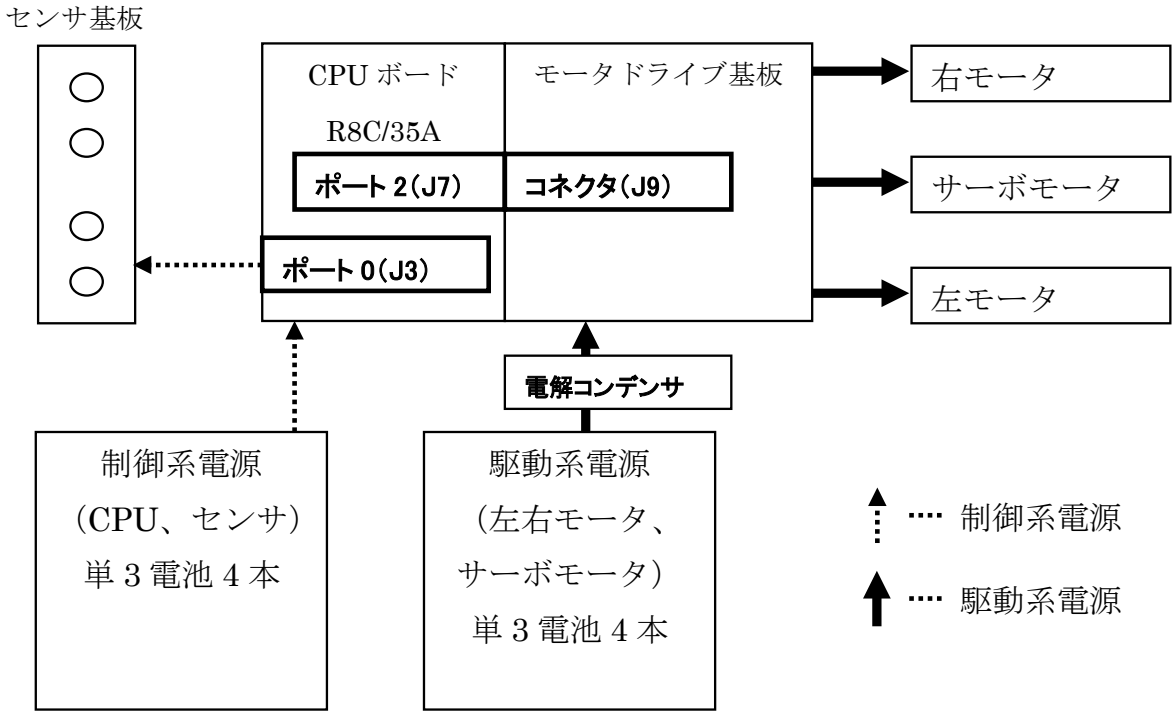
4 輪ミニマイコンカーは、制御系の CPU ボード、センサ基板、モータドライブ基板、駆動系の右モータ、左モータ、サーボモータで構成されています。



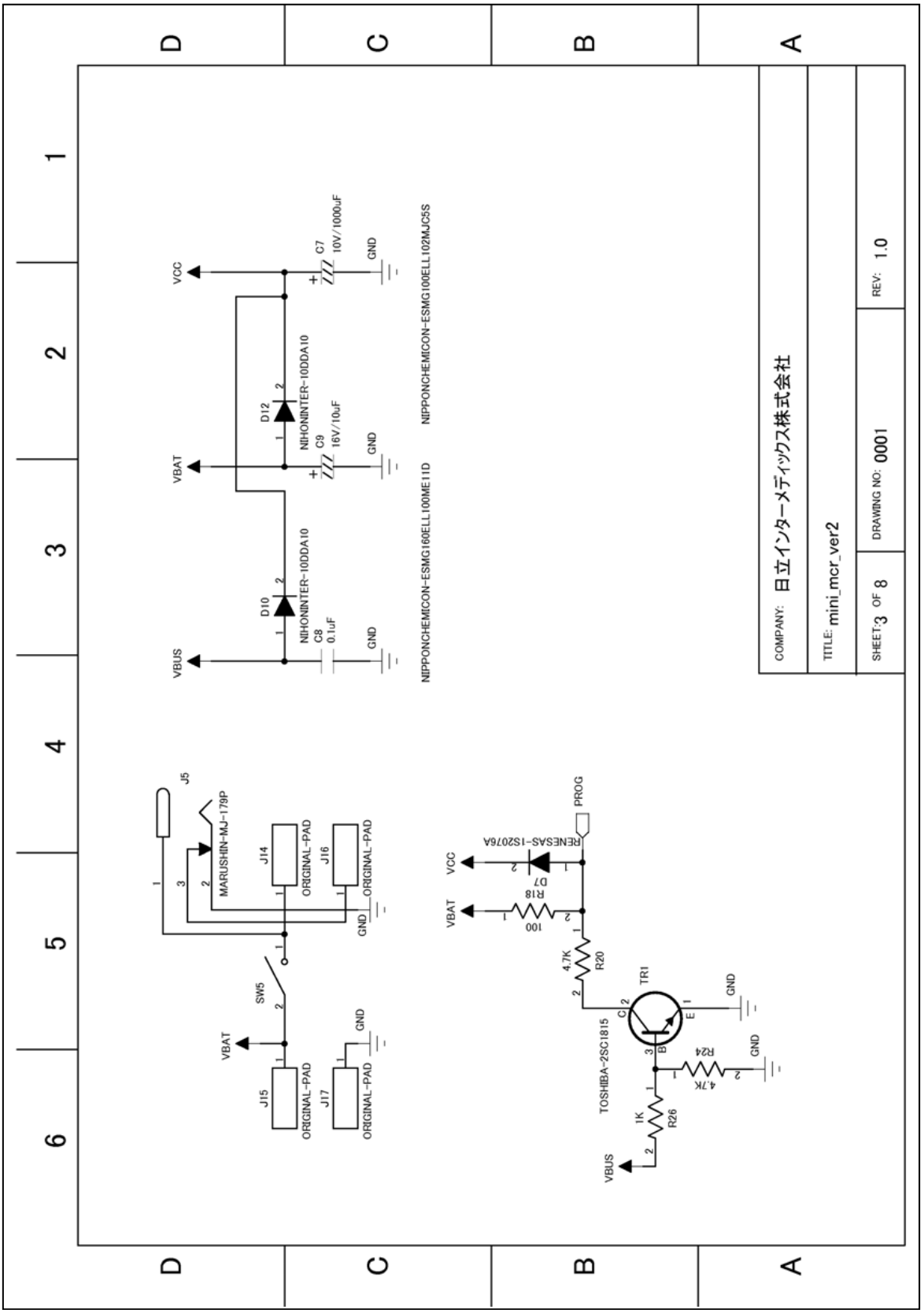
2.2 電源構成

電源構成は、制御系と駆動系を別電源とし、モータ、サーボモータ駆動時の電源電圧変動やノイズがCPUの動作に影響を与えないようにしています。

以下に電源構成を示します。

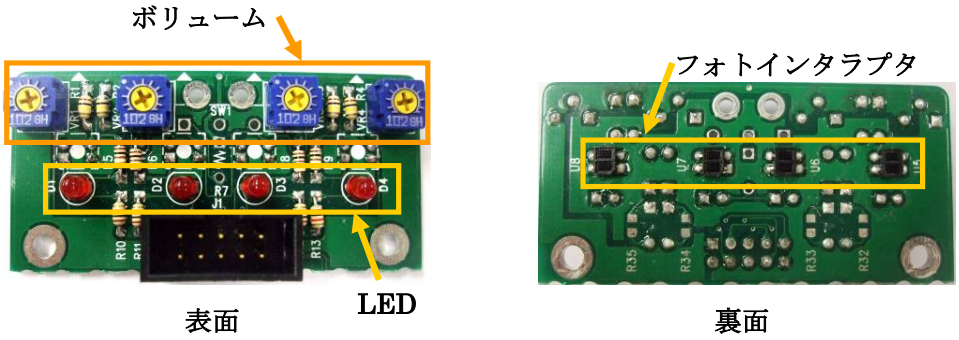


2.3 電源の回路図


COMPANY: 日立インターメディックス株式会社
TITLE: mini_mcr_ver2
SHEET: 3 OF 8
DRAWING NO: 0001
REV: 1.0
VCC
VBAT
VBUS
GND
C7 10V/1000uF
C9 16V/10uF
C8 0.1uF
D12 NIHONINTER-10DDA10
D10 NIHONINTER-10DDA10
NIPPONCHEMICON-ESMG160ELL100ME11D
NIPPONCHEMICON-ESMG100ELL102MJC5S
J5 MARUSHIN-MJ-179P
J14 ORIGINAL-PAD
J16 ORIGINAL-PAD
J15 ORIGINAL-PAD
J17 ORIGINAL-PAD
SW5
PROG
RENESAS-IS2076A
D7
R18 100
R20 4.7K
TR1 TOSHIBA-2SC1815
R24 4.7K
R26 1K

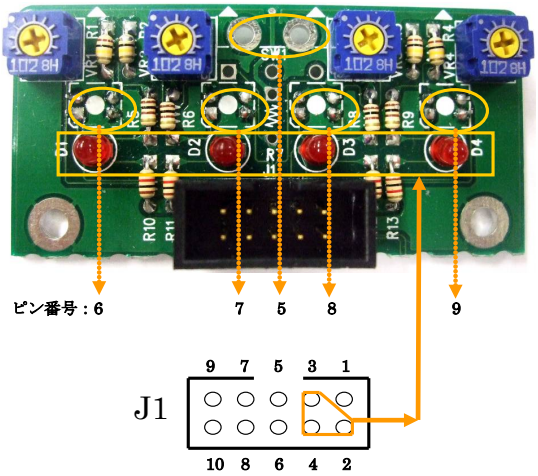
3 センサ基板について

3.1 外観



3.2 コネクタピン配置

○で囲んだセンサの信号が、10P コネクタ(J1)から出力されます。



ピン番号	詳細	"0"	"1"	マイコンの接続先	備考
1	VCC(+5V)				
2	LED_C	LED_A、LED_B、LED_C の信号レベルにより、D1～D4 の LED をどう点灯させるか選択します。		P0_7	
3	LED_B			P0_6	
4	LED_A			P0_5	
5	マイクロスイッチ(SW1)	ON	OFF	P0_4	マイクロスイッチはオプションです
6	赤外線フォトインタラプタ(U5)	白	黒(未反応)	P0_3	赤外線フォトインタラプタからは、0～5V の信号が出力されます。マイコンは、2.5V 以下なら白、以上なら黒と判断します。
7	赤外線フォトインタラプタ(U6)	白	黒(未反応)	P0_2	
8	赤外線フォトインタラプタ(U7)	白	黒(未反応)	P0_1	
9	赤外線フォトインタラプタ(U8)	白	黒(未反応)	P0_0	
10	GND				

3.3 センサ基板の LED 点灯、消灯

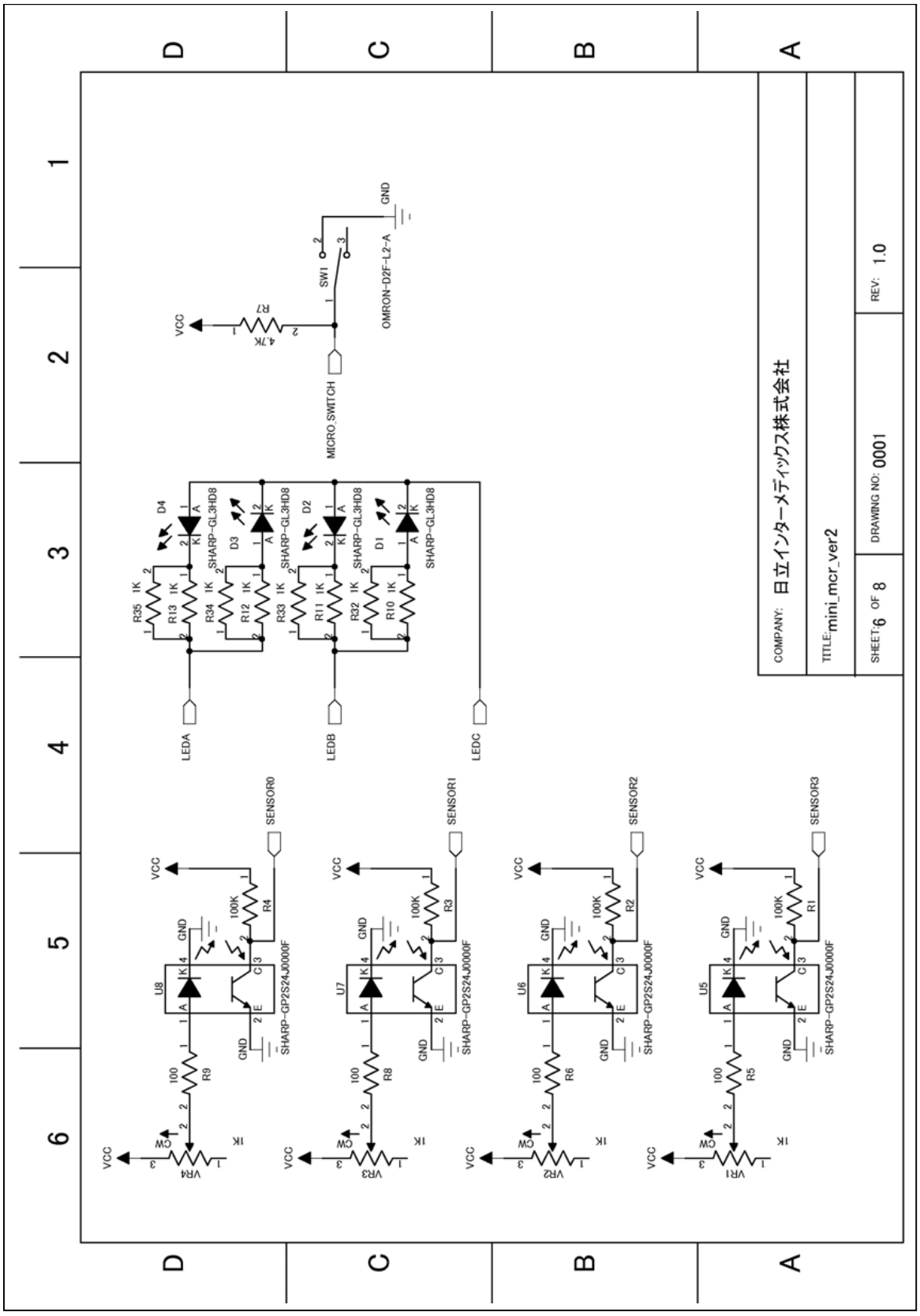
J1 の残っているピン数の関係から、D1～D4 の 4 個の LED を 3 本(P0_7～P0_5)の信号線で制御しています。3 本の信号レベルによって LED の点灯パターンが変わります。下表に点灯パターンを示します。

P0_7 LEDC	P0_6 LEDB	P0_5 LEDA	D1	D2	D3	D4
0V(″0″)	0V(″0″)	0V(″0″)	消灯	消灯	消灯	消灯
0V(″0″)	0V(″0″)	5V(″1″)	消灯	消灯	**点灯**	消灯
0V(″0″)	5V(″1″)	0V(″0″)	**点灯**	消灯	消灯	消灯
0V(″0″)	5V(″1″)	5V(″1″)	**点灯**	消灯	**点灯**	消灯
5V(″1″)	0V(″0″)	0V(″0″)	消灯	**点灯**	消灯	**点灯**
5V(″1″)	0V(″0″)	5V(″1″)	消灯	**点灯**	消灯	消灯
5V(″1″)	5V(″1″)	0V(″0″)	消灯	消灯	消灯	**点灯**
5V(″1″)	5V(″1″)	5V(″1″)	消灯	消灯	消灯	消灯

プログラムでは、次のように時間を分けて LED を制御します。

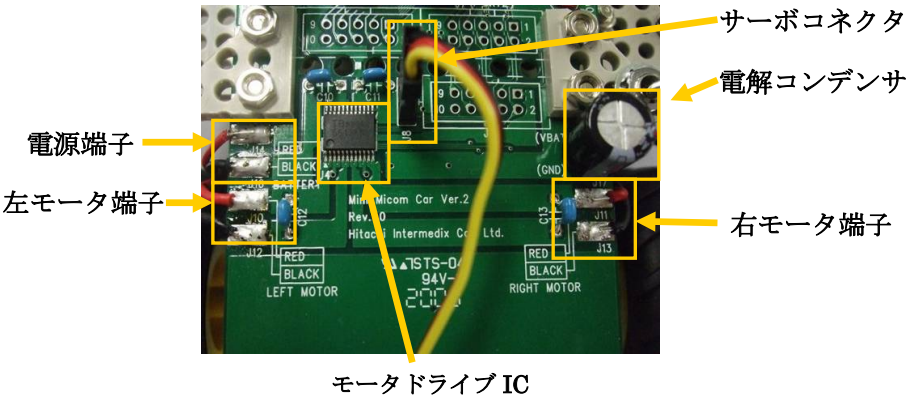
・1ms の間、P0_7 を 0V にして、P0_6 で D1 を、P0_5 で D3 を点灯、または消灯させる。
・1ms の間、P0_7 を 5V にして、P0_6 で D2 を、P0_5 で D4 を点灯、または消灯させる。

3.4 回路図


COMPANY: 日立インターメディックス株式会社
TITLE: mini_mcr_ver2
SHEET: 6 OF 8
DRAWING NO: 0001
REV: 1.0
SENSOR0
SENSOR1
SENSOR2
SENSOR3
MICRO_SWITCH
OMRON-D2F-L2-A
SHARP-GP2S24J0000F
SHARP-GL3HD8
LEDA
LEDB
LEDC
VCC
GND

4 モータドライブ基板

4.1 外観

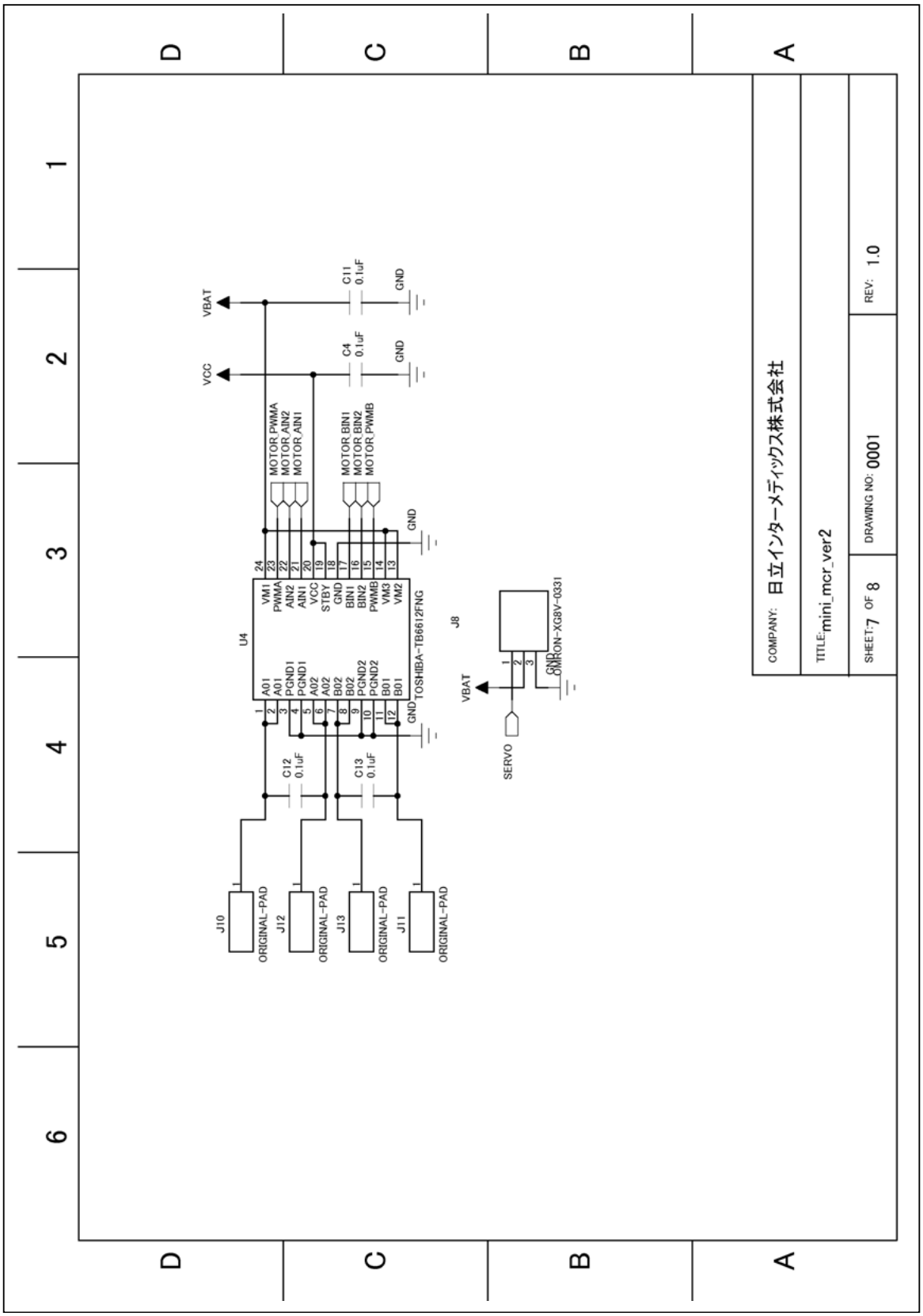


4.2 コネクタピン配置

ピン番号	詳細	マイコンの接続先	備考
1	VCC(+5V)		
2	モータ右 2	P2_7	モータ右 1 との組み合わせで右モータの正転、逆転を決めます
3	モータ左 2	P2_6	モータ左 1 との組み合わせで左モータの正転、逆転を決めます
4	サーボ	P2_5(PWM 出力)	PWM 信号により、サーボの角度を決めます
5	モータ右 PWM	P2_4(PWM 出力)	右モータの回転する速さを決めます
6	モータ右 1	P2_3	モータ右 2 との組み合わせで右モータの正転、逆転を決めます
7	モータ左 PWM	P2_2(PWM 出力)	左モータの回転する速さを決めます
8	モータ左 1	P2_1	モータ左 2 との組み合わせで左モータの正転、逆転を決めます
9	未接続		
10	GND		

※マイコンの接続先は、モータドライブ部とマイコン部を分離していない場合、パターンで接続されています。
モータドライブ基板を分離した場合は、J9 と J7(ポート 2)をフラットケーブルなどで接続してください。

4.3 回路図


COMPANY: 日立インターメディックス株式会社
TITLE: mini_mcr_ver2
DRAWING NO: 0001
REV: 1.0
SHEET: 7 OF 8
U4
TOSHIBA-TB6612FNG
VM1
PWMA
AIN2
AIN1
VCC
STBY
GND
BIN1
BIN2
PWMB
VM3
VM2
A01
A01
PGND1
PGND1
A02
A02
B02
B02
PGND2
PGND2
B01
B01
GND
MOTOR_PWMA
MOTOR_AIN2
MOTOR_AIN1
MOTOR_BIN1
MOTOR_BIN2
MOTOR_PWMB
VBAT
VCC
C11 0.1uF
C4 0.1uF
C12 0.1uF
C13 0.1uF
GND
J8
OMRON-XG8V-0331
SERVO
J10
J12
J13
J11
ORIGINAL-PAD

5 サンプルプログラム

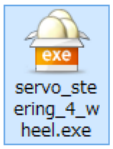
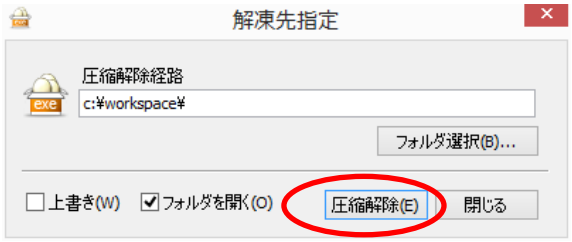
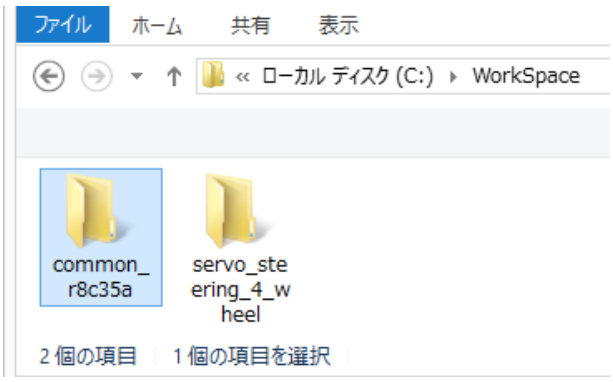
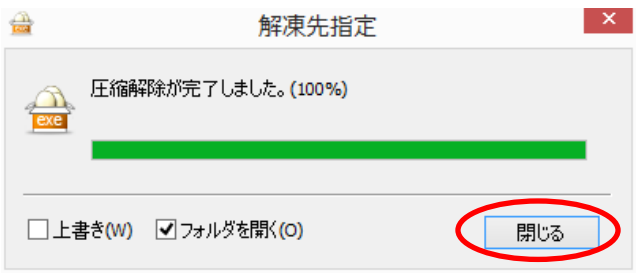
5.1 ルネサス統合開発環境

サンプルプログラムは、**ルネサス統合開発環境(High-performance Embedded Workshop)**を使用して開発します。ルネサス統合開発環境についてのインストール、開発方法については、**「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。**

※「ルネサス統合開発環境操作マニュアル」は、下記の URL よりダウンロードできます。

(URL:https://www2.himdx.net/mcr/product/download.html)

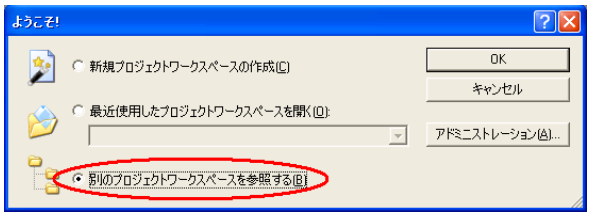
5.2 サンプルプログラムのインストール

	マイコンカーラリー販売サイト (URL:https://www2.himdx.net/mcr/product/download.html) からダウンロードした「servo_steering_4_wheel.zip」ファイルを解凍します。 解凍した「servo_steering_4_wheel.exe」を実行します。
	「圧縮解除」をクリックします。 **※「解凍先指定」の解凍先は変更しないでください。解凍先を変更しない場合は、「C:¥WorkSpace」に解凍されます。**
	解凍が終わると「C:¥WorkSpace」フォルダが自動で開きます。 このフォルダの中に「servo_steering_4_wheel」と「common_r8c35a」のフォルダがあります。 「servo_steering_4_wheel」がサンプルプログラムになります。 **※「common_r8c35a」フォルダには、「servo_steering_4_wheel」で使用する「sfr_r835a.h」ファイルが入っています。**
	「閉じる」をクリックします。

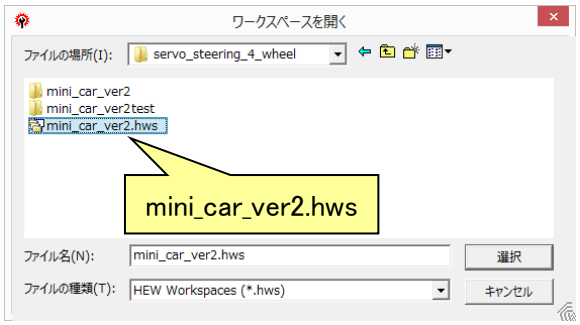
5.3 ワークスペース「servo_steering_4_wheel」を開く

ルネサス統合開発環境(HEW)を実行します。

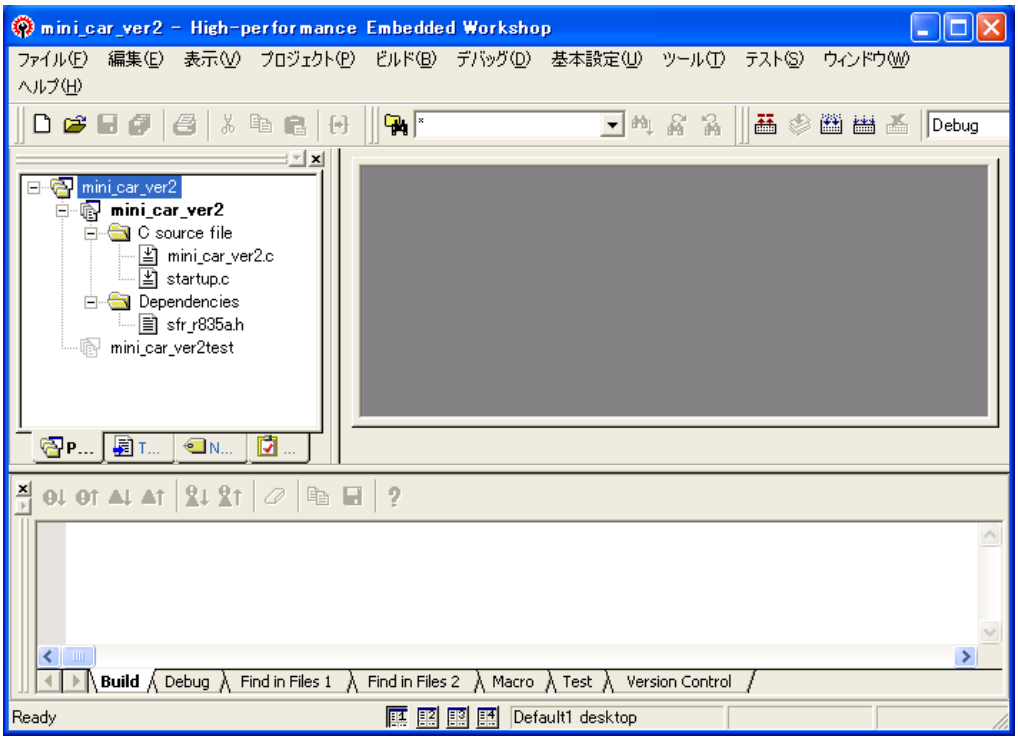
左の図のアイコンをダブルクリックします。



「ようこそ」の画面から
「別のプロジェクトワークスペースを参照する」を選択して、「OK」をクリックします。



C ドライブ→Workspace→servo_steering_4_wheel の「**mini_car_ver2.hws**」を選択します。

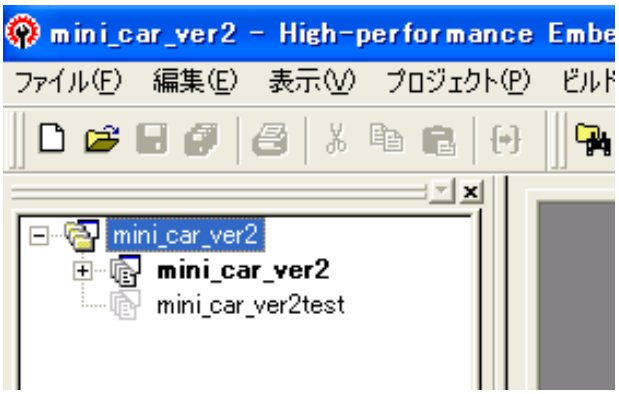


「servo_steering_4_wheel」のワークスペースが開かれます。

6 動作確認用プログラムの書き込み

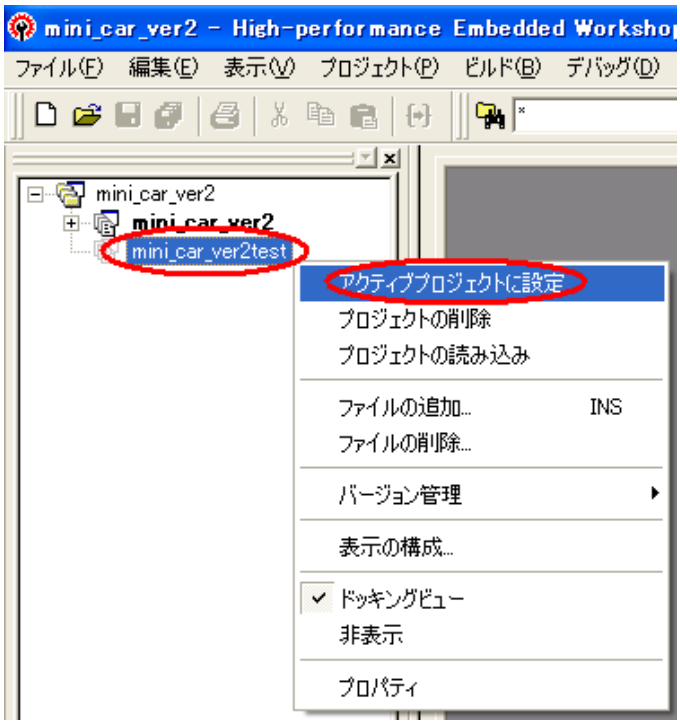
6.1 プロジェクトの変更

5.3 で「servo_steering_4_wheel」のワークスペースを開きました。ワークスペースを開いた状態から説明します。



ワークスペース「servo_steering_4_wheel」には、2 つのプロジェクトが登録されています。

プロジェクト名	内容
mini_car_ver2	4 輪ミニマイコンカーの走行プログラムです。
mini_car_ver2test	製作、組み立てをした 4 輪ミニマイコンカーのモータドライブ基板、センサ基板、サーボモータが正しく動作するかの確認をするプログラムです。



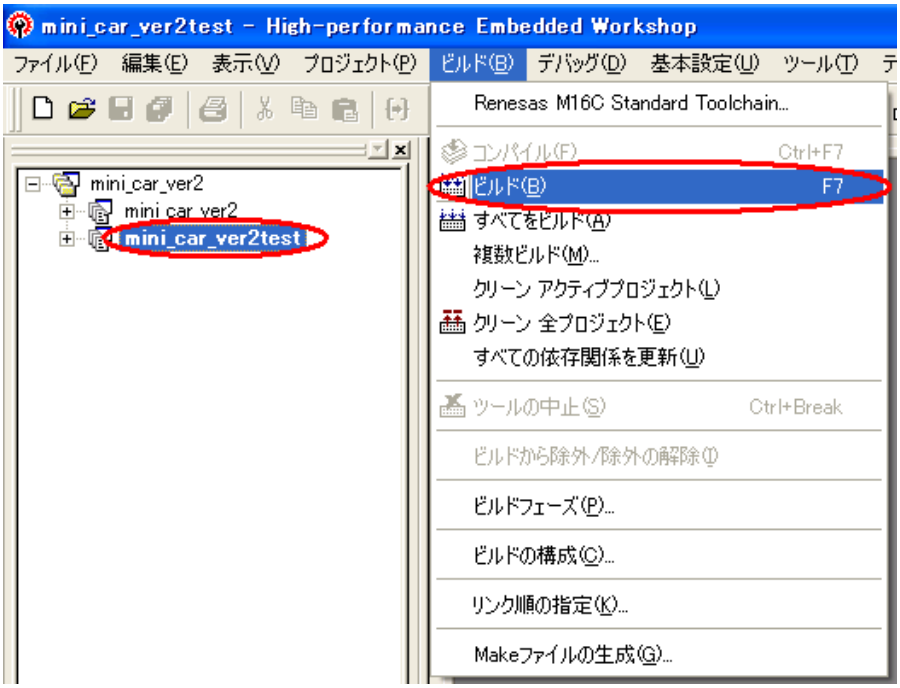
4 輪ミニマイコンカーの動作確認をするため、**「mini_car_ver2test」**をアクティブプロジェクトに設定します。

「mini_car_ver2test」上で右クリックをします。
「アクティブプロジェクトに設定」を選択します。

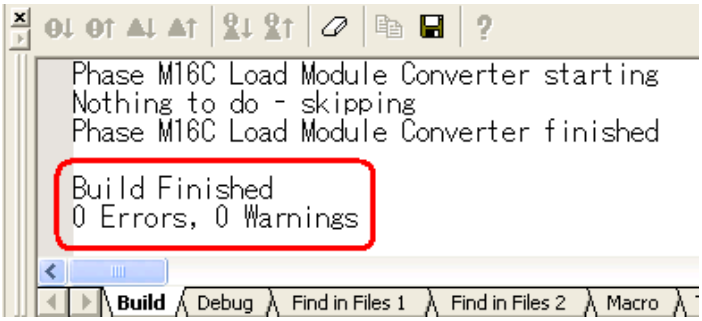
選択すると「mini_car_ver2test」の文字が太字になります。

6.2 動作確認プログラムの書き込み

(1)プログラムのビルド



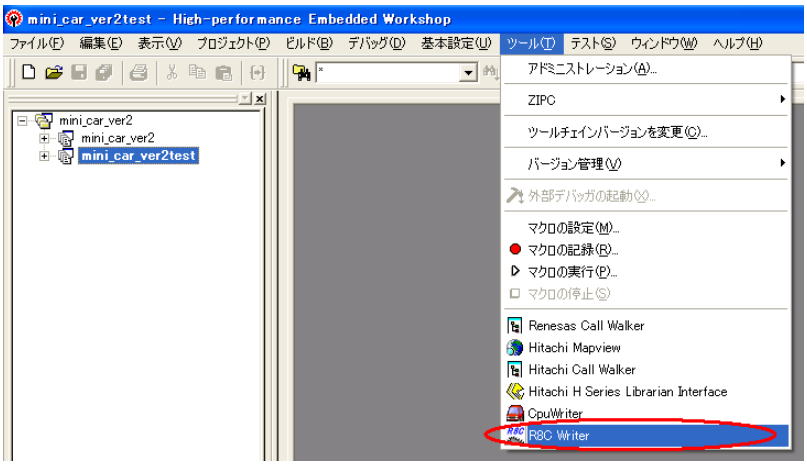
「mini_car_ver2test」が有効(太字)になっていることを確認して、メニューバーにある「ビルド→ビルド」を実行します。



●Error
誤りのことです。Error が出た場合は必ずプログラムやツールチェインの設定を修正します。

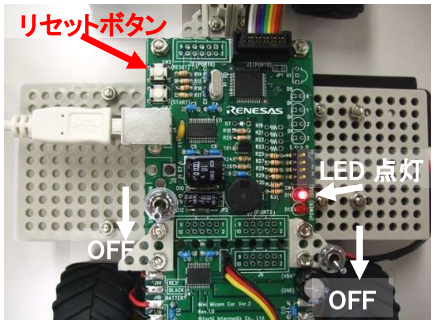
●Warning
警告です。必ずしも誤っているとは言い切れないが、間違っている可能性があるので確認してくださいというメッセージです。こちらも必ず修正します。

(2)プログラムの書き込み

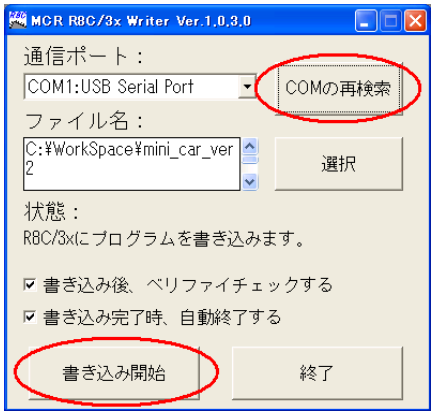


メニューバーにある「ツール→R8C Writer」で書き込みソフトを起動します。もし、R8C Writer コマンドが無い場合は、R8C/35A マイコン実習マニュアルの**「3.5 R8C Writer のダウンロード ～ 3.7 R8C Writer をルネサス統合開発環境に登録をする」**を参照して、登録をしてください。

※「R8C/35A マイコン実習マニュアル」は、こちらの URL（http://www2.himdx.net/mcr/）よりダウンロードできます。



①電源スイッチを OFF にします。
②パソコンとマイコンボードを USB ケーブルで接続します。このとき、マイコンボードの LED（USB）が点灯します。
※LED（POWER）が消灯していることを確認してください。
③リセットボタン（SW2）を押します。

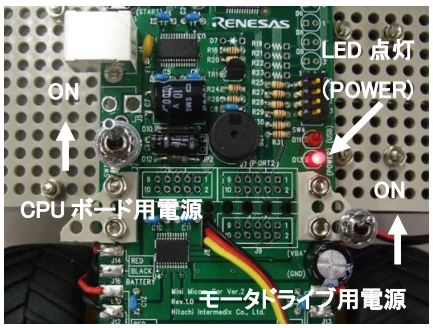


「通信ポート」の番号をミニマイコンカーVer.2 基板のポート番号に設定します。**「COM の再検索」**をクリックします。通信ポートの部分に「COM○：USB Serial Port」（○部分には番号が入ります。）と表示されます。

※通信ポートの番号が分からない場合は、R8C/35A マイコン実習マニュアルの「5.5 プログラムの書き込み」を参照してください。

「書き込み開始」をクリックします。プログラムの書き込みが行われます。

正常にプログラムの書き込みが終わると、R8C Writer は自動終了します。マイコンボードから USB ケーブルを外してください。



電池ボックスに電池を入れます。CPU ボード用電源とモータドライブ用電源のスイッチを「ON」にすると、書き込んだプログラムが実行され、ディップスイッチの状態によって、センサ基板、サーボモータ、タイヤ、ブザーの動作確認ができます。

このとき、マイコンボードの LED(POWER)が点灯します。

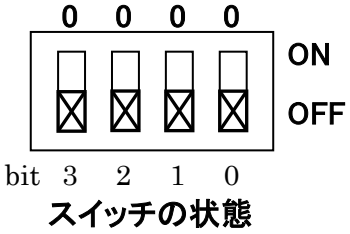
7 動作テスト

7.1 動作テスト一覧

CPUボードのディップスイッチの状態を変更することにより、4輪ミニマイコンカーのどの部分の動作確認をするか選択し、動作の確認をします。

ディップスイッチ				内容
4(3bit)	3(2bit)	2(1bit)	1(0bit)	
0	0	0	0	**LED のテストをします。**LED が 0.5 秒間隔で交互に点灯します
0	0	0	1	**プッシュスイッチのテストをします。**スイッチ OFF で LED が全て消灯、スイッチ ON で LED が全て点灯します。
0	0	1	0	**サーボモータのテストをします。** サーボモータが、「0° →右 30° →左 30° の繰り返し」の動作をします。
0	0	1	1	何もしません。
0	1	0	0	**右モータのテストをします。** 「正転→ブレーキ」を繰り返します。
0	1	0	1	**右モータのテストをします。** 「逆転→ブレーキ」を繰り返します。
0	1	1	0	**左モータのテストをします。** 「正転→ブレーキ」を繰り返します。
0	1	1	1	**左モータのテストをします。** 「逆転→ブレーキ」を繰り返します。
1	0	0	0	**センサとブザーのテストをします。**
1	0	0	1	何もしません。
1	0	1	0	何もしません。
1	0	1	1	何もしません。
1	1	0	0	**直進テストをします。** PWM50%で前進、2 秒後にストップします。
1	1	0	1	**直進テストをします。** PWM50%で前進、5 秒後にストップします。
1	1	1	0	**直進テストをします。** PWM100%で前進、2 秒後にストップします。
1	1	1	1	**直進テストをします。** PWM100%で前進、5 秒後にストップします。

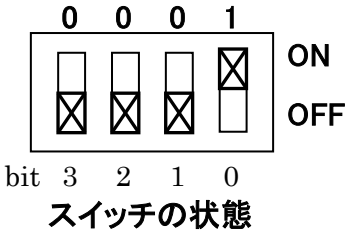
7.2 LED のテスト

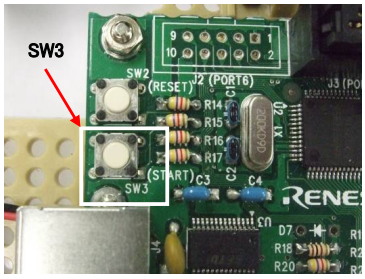


ディップスイッチ“0000”の状態です。CPU ボード用電源を ON にします。ディップスイッチは ON が“1”、OFF“0”です。
センサ基板の LED が 0.5 秒間隔で点滅します。

> 点灯しない場合は、センサ基板と CPU ボードを接続しているフラットケーブルの不良、LED の半田付け不良、半田ブリッジ（ショート）、LED の向きが逆などが考えられます。

7.3 プッシュスイッチのテスト

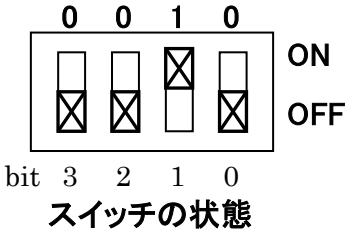




ディップスイッチ“0001”の状態で CPU ボード用電源を ON にします。マイコンボード上のプッシュスイッチ（SW3）が押されていない状態ならセンサ基板の LED が全て消灯、押されたら全て点灯します。
※SW2 のプッシュスイッチはリセットスイッチです。マイコンのポートには接続されていません。

> センサ基板の LED が点灯しない場合は、スイッチまでの回路の半田付け不良、LED が点灯しっぱなしの場合は半田ブリッジなどが考えられます。

7.4 サーボモータのテスト



ディップスイッチ“0010”の状態で CPU ボード用電源、モータドライブ用電源を ON にします。
サーボモータが 1 秒ごとに、「0° →右 30° →左 30° 」の動作を繰り返します。

> サーボモータが動作しない場合は、サーボまでの回路の半田付け不良、サーボコネクタの差し込む向きが逆などが考えられます。

キットに付属していないサーボに交換したとき、**「0° →左 30° →右 30° 」**の動作になる場合、左右の回転が逆のサーボです。その場合は、handle 関数内の 505 行を下記のように変更すると左右が入れ代わり、動作が「0° →右 30° →左 30° 」の動作となります。

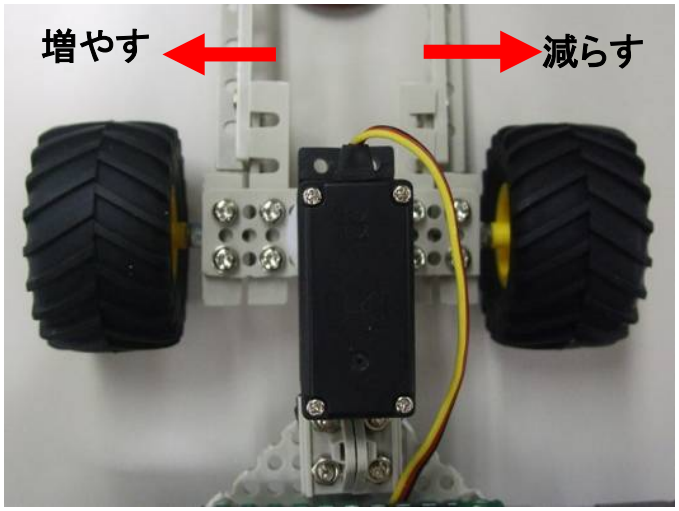
505 行	修正前	trdgrd1 = SERVO_CENTER − angle ∗ HANDLE_STEP;
	修正後	trdgrd1 = SERVO_CENTER **+** angle ∗ HANDLE_STEP;

※サーボのセンタ位置の調整について

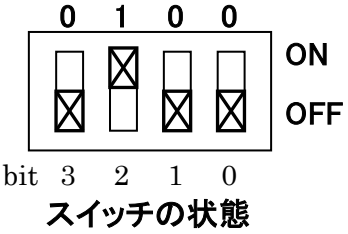
ほとんどの場合、電源を ON にしても前輪がまっすぐ向いていません。これはサーボのセンタ値がそれぞれのサーボによって値が違うため 4 輪ミニマイコンカー1 台 1 台違う値になります。「mini_car_ver2test」の SERVO_CENTER の「3749」という値を変えてまっすぐになるように調整します。23 行目の HANDLE_STEP 値の「22」がサーボ 1° あたりの数値となります。
※1° あたりの数値（HANDLE_STEP 値）の求め方については「**9.4 その他のシンボル定義**」を参照してください。

```
022 #define     SERVO_CENTER    3749    /* サーボのセンタ値    */
023 #define     HANDLE_STEP     22      /* 1° 分の値          */
```

値を増やすと、進行方向に向かって左側、減らすと右側に動きます。

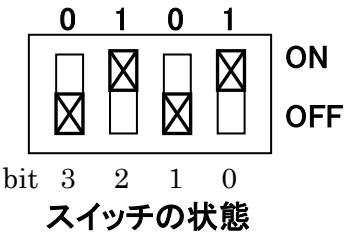


7.5 右モータのテスト



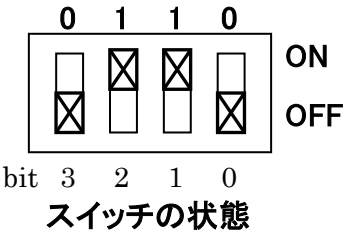
ディップスイッチ“0100”の状態で CPU ボード用電源、モータドライブ用電源を ON にします。
右モータが 1 秒ごとに「正転→ブレーキ」を繰り返します。

> 右モータが正転しない場合は、右モータの制御回路の半田付け不良が考えられます。回転し続ける場合は、半田ブリッジしている可能性があります。また、タイヤが逆転した場合は、モータの赤と黒のリード線の半田付けが逆です。



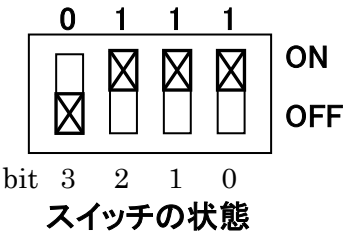
CPU ボード用電源を OFF にします。ディップスイッチを“0101”の状態で CPU ボード用電源を ON にします。
右モータが 1 秒ごとに「逆転→ブレーキ」を繰り返します。

7.6 左モータのテスト



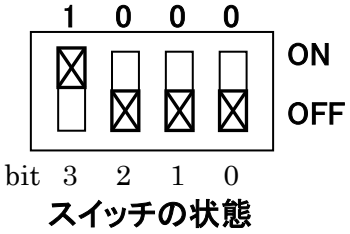
ディップスイッチ“0110”の状態で CPU ボード用電源、モータドライブ用電源を ON にします。
左モータが 1 秒ごとに「正転→ブレーキ」を繰り返します。

> 左モータが正転しない場合は、左モータの制御回路の半田付け不良が考えられます。回転し続ける場合は、半田ブリッジしている可能性があります。また、タイヤが逆転した場合は、モータの赤と黒のリード線の半田付けが逆です。

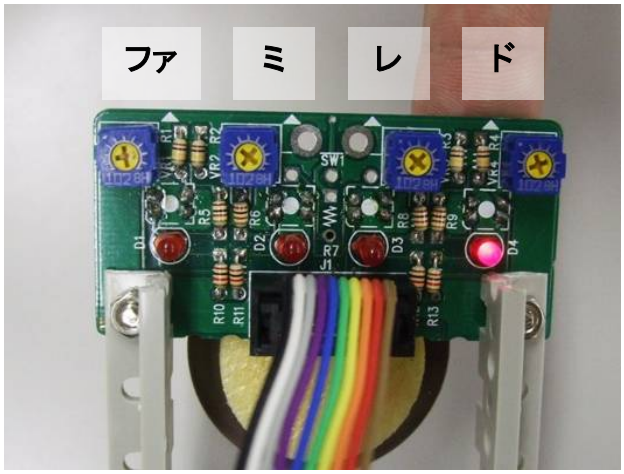


CPU ボード用電源を OFF にします。ディップスイッチを“0111”の状態で CPU ボード用電源を ON にします。
左モータが 1 秒ごとに「逆転→ブレーキ」を繰り返します。

7.7 センサのテスト



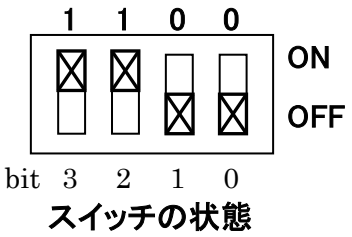
ディップスイッチ“1000”の状態で CPU ボード用電源を ON にします。
下図のように、センサ基板のセンサに指を当てて反応するかを確認します。反応するとセンサに対応するLED が点灯し、ブザーがなります。



ブザーは、左図の右端のセンサから「4 オクターブ目のド、レ、ミ、ファ」の音が鳴ります。

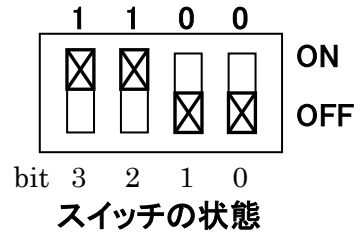
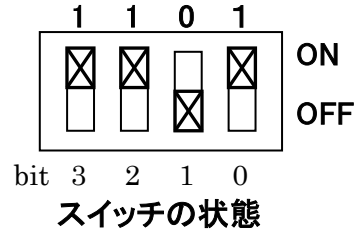
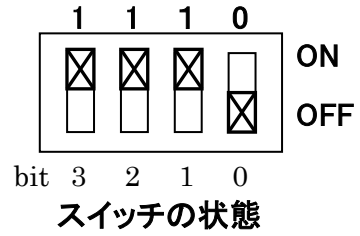
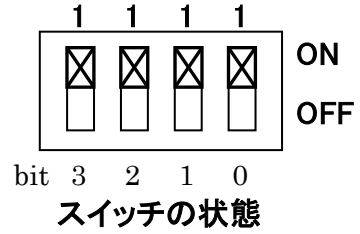
センサ基板の LED が点灯しない場合は、センサ基板の半田付け不良、半田ブリッジ、部品の向きが逆などが考えられます。ブザーが鳴らない場合は、ブザーの回路部分の半田付け不良が考えられます。

7.8 直進テスト



ディップスイッチ“1100”の状態でCPUボード用電源、モータドライブ用電源を ON にします。
2秒後にPWM50%で2秒間直進します。屋内の平らで直線の長い場所で 4 輪ミニマイコンカーを走らせ、まっすぐに進むかをテストします。曲がってしまう場合は、**SERVO_CENTER の値を調整**して直進するようにします。直進性は 4 輪ミニマイコンカーのスピードが速くなった場合、非常に重要になります。

直進テストは、PWM 値と停止するまでの時間の違いで4パターンあります。

スイッチ	PWM 値	停止するまでの時間
	50%	2秒
	50%	5秒
	100%	2秒
	100%	5秒

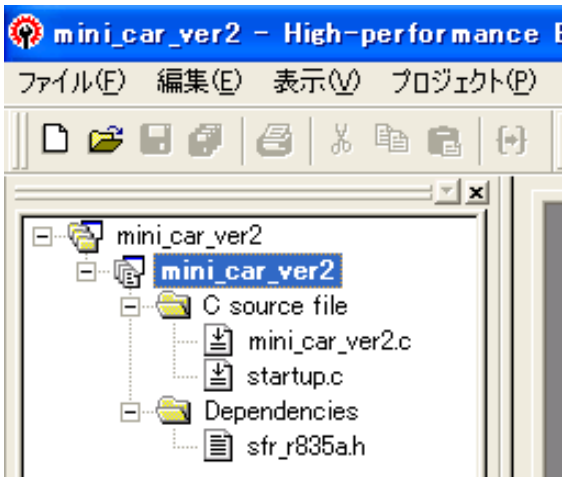
以上で、動作確認は終わりです。

8 プロジェクト内のファイルの構成

8.1 概要

ワークスペース「servo_steering_4_wheel」のプロジェクト「mini_car_ver2」のプロジェクト内にあるファイルについて説明します。

8.2 プロジェクトのファイル構成



プロジェクト「mini_car_ver2」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	mini_car_ver2.c	C 言語ソースファイルです。このファイルは、周辺機能の初期化、4 輪ミニマイコンカーの制御をするメインプログラムが含まれています。
2	startup.c	スタートアッププログラムを記述したファイルです。
3	sfr_r835a.h	SFR(スペシャルファンクションレジスタ)の定義をしたファイルです。

※SFR(Special Function Register)とは、周辺機能の制御レジスタの総称です。

9 プログラム解説「mini_car_ver2.c」

9.1 プログラムリスト

```
001 /************************************************************************/
002 /* 対象マイコン R8C/35A                                                 */
003 /* ﾌｧｲﾙ内容     ミニマイコンカーVer.2 サーボステアリング4輪セット       */
004 /* バージョン   Ver.1.00                                                */
005 /* Date         2010.12.15                                              */
006 /* Copyright    ルネサスエレクトロニクス ルネサスマイコンカーラリー事務局 */
007 /*              日立インターメディックス株式会社                        */
008 /************************************************************************/
009
010 /*====================================*/
011 /* インクルード                       */
012 /*====================================*/
013 #include "sfr_r835a.h"                  /* R8C/35A SFRの定義ファイル    */
014
015 /*====================================*/
016 /* シンボル定義                       */
017 /*====================================*/
018
019 /* 定数設定                 */
020 #define     PWM_CYCLE       39999       /* TRD周期16ms                  */
021
022 #define     SERVO_CENTER    3749        /* サーボのセンタ値             */
023 #define     HANDLE_STEP     22          /* 1° 分の値                    */
024
025 /* マスク値設定 ×：マスクあり(無効) ○：マスク無し(有効) */
026 #define         MASK_0001   0x01        /* ×××○                        */
027 #define         MASK_0010   0x02        /* ××○×                        */
028 #define         MASK_0011   0x03        /* ××○○                        */
029 #define         MASK_0100   0x04        /* ×○××                        */
030 #define         MASK_0101   0x05        /* ×○×○                        */
031 #define         MASK_0110   0x06        /* ×○○×                        */
032 #define         MASK_0111   0x07        /* ×○○○                        */
033 #define         MASK_1000   0x08        /* ○×××                        */
034 #define         MASK_1001   0x09        /* ○××○                        */
035 #define         MASK_1010   0x0a        /* ○×○×                        */
036 #define         MASK_1011   0x0b        /* ○×○○                        */
037 #define         MASK_1100   0x0c        /* ○○××                        */
038 #define         MASK_1101   0x0d        /* ○○×○                        */
039 #define         MASK_1110   0x0e        /* ○○○×                        */
040 #define         MASK_1111   0x0f        /* ○○○○                        */
041
042 /* 音階 ブザー用            */
043 /* 3オクターブ目の音階      */
044 #define         DO_3        38226       /* ド                           */
045 #define         DOU_3       36081       /* ド＃                         */
046 #define         RE_3        34056       /* レ                           */
047 #define         REU_3       32144       /* レ＃                         */
048 #define         MI_3        30340       /* ミ                           */
049 #define         FA_3        28637       /* ファ                         */
050 #define         FAU_3       27030       /* ファ＃                       */
051 #define         SO_3        25513       /* ソ                           */
052 #define         SOU_3       24081       /* ソ＃                         */
053 #define         RA_3        22729       /* ラ                           */
054 #define         RAU_3       21454       /* ラ＃                         */
055 #define         SI_3        20250       /* シ                           */
056
057 /* 4オクターブ目の音階      */
058 #define         DO_4        19113       /* ド                           */
059 #define         DOU_4       18040       /* ド＃                         */
060 #define         RE_4        17028       /* レ                           */
061 #define         REU_4       16072       /* レ＃                         */
062 #define         MI_4        15170       /* ミ                           */
063 #define         FA_4        14319       /* ファ                         */
064 #define         FAU_4       13515       /* ファ＃                       */
065 #define         SO_4        12756       /* ソ                           */
066 #define         SOU_4       12041       /* ソ＃                         */
067 #define         RA_4        11365       /* ラ                           */
068 #define         RAU_4       10727       /* ラ＃                         */
069 #define         SI_4        10125       /* シ                           */
070
071 /* 5オクターブ目の音階      */
072 #define         DO_5        9557        /* ド                           */
073 #define         DOU_5       9020        /* ド＃                         */
074 #define         RE_5        8514        /* レ                           */
075 #define         REU_5       8036        /* レ＃                         */
076 #define         MI_5        7585        /* ミ                           */
077 #define         FA_5        7159        /* ファ                         */
078 #define         FAU_5       6758        /* ファ＃                       */
079 #define         SO_5        6378        /* ソ                           */
080 #define         SOU_5       6020        /* ソ＃                         */
081 #define         RA_5        5682        /* ラ                           */
```

```
#define         RAU_5       5363        /* ラ#                          */
#define         SI_5        5062        /* シ                           */

/*====================================*/
/* プロトタイプ宣言                   */
/*====================================*/
void init( void );
void timer( unsigned int time );
int check_crossline( void );
int check_rightline( void );
int check_leftline( void );
unsigned char sensor_get( unsigned char mask );
void sensorled_out( unsigned char led );
unsigned char dipsw_get( void );
unsigned char pushsw_get( void );
void led_out( unsigned char led );
void motor( int data1, int data2 );
void handle( int angle );
void beep( unsigned int tone );

/*====================================*/
/* グローバル変数                     */
/*====================================*/
unsigned long   cnt0;                   /* timer関数用変数              */
unsigned long   cnt1;                   /* main関数内用                 */

/* sensorled_out関数用変数 */
unsigned char   sensorled_outdata;      /* 出力データ保存用変数         */
int             s_led_out_on;           /* 関数のON/OFF切り替え         */

int             pattern;                /* パターン番号                 */

/************************************************************************/
/* メインプログラム                                                     */
/************************************************************************/
void main( void )
{
    init();                             /* 初期化                       */
    asm("fset I");                      /* 全体割り込み許可             */

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

    while( 1 ){
        switch( pattern ){

        /*****************************************************************
        パターンについて
        0：待機モード（リセット直後の動作）
        1：センサボリューム調整モード
        2：カウントダウンスタート
        11：通常トレース
        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
        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：右クランククリア処理　安定するまで少し待つ
        32：右クランククリア処理　曲げ終わりのチェック
        41：左クランククリア処理　安定するまで少し待つ
        42：左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
        52：２本目を読み飛ばす
        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
        62：２本目を読み飛ばす
        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
        *****************************************************************/

            case 0:
                /* 待機モード（リセット直後の動作）          */
                s_led_out_on = 1;       /* sensorled_out関数使用許可    */
                if( pushsw_get() ){
                    while( pushsw_get() );
                    beep( SI_3 );       /* 3オクターブ目のシの音をセット*/
                    timer( 200 );
                    beep( 0 );          /* ブザーの停止                 */
                    pattern = 1;
                    cnt1 = 0;
                    break;
                }

                /* センサLED点滅処理                        */
                if( cnt1 > 200 ){
                    sensorled_out( 0x5 );
                    cnt1 = 0;
                }else if( cnt1 > 100 ){
                    sensorled_out( 0xa );
```

```
173                 }
174                 break;
175
176             case 1:
177                 /* センサボリューム調整モード                */
178                 /* スイッチ入力待ち                          */
179                 s_led_out_on = 0;       /* sensorled_out関数使用禁止    */
180                 if( pushsw_get() ){
181                     while( pushsw_get() );
182                     pattern = 2;
183                     cnt1 = 0;
184                     break;
185                 }
186                 break;
187
188             case 2:
189                 /* カウントダウンスタート                    */
190                 s_led_out_on = 1;       /* sensorled_out関数使用許可    */
191                 sensorled_out( 0x0c );
192                 beep( RA_3 );           /* 3オクターブ目のラの音をセット*/
193                 timer( 500 );
194                 beep( 0 );              /* ブザーの停止                 */
195                 timer( 500 );
196
197                 sensorled_out( 0x0e );
198                 beep( RA_3 );           /* 3オクターブ目のラの音をセット*/
199                 timer( 500 );
200                 beep( 0 );              /* ブザーの停止                 */
201                 timer( 500 );
202
203                 sensorled_out( 0x0f );
204                 beep( RA_3 );           /* 3オクターブ目のラの音をセット*/
205                 timer( 500 );
206                 beep( 0 );              /* ブザーの停止                 */
207                 timer( 500 );
208
209                 sensorled_out( 0x00 );
210                 beep( RA_5 );           /* 5オクターブ目のラの音をセット*/
211                 timer( 1200 );
212                 beep( 0 );              /* ブザーの停止                 */
213
214                 s_led_out_on = 0;       /* sensorled_out関数使用禁止    */
215                 pattern = 11;
216                 cnt1 = 0;
217                 break;
218
219             case 11:
220                 /* 通常トレース                              */
221                 if( check_crossline() ){/* クロスラインチェック         */
222                     pattern = 21;
223                     break;
224                 }
225                 if( check_rightline() ){/* 右ハーフラインチェック       */
226                     pattern = 51;
227                     break;
228                 }
229                 if( check_leftline() ){ /* 左ハーフラインチェック       */
230                     pattern = 61;
231                     break;
232                 }
233
234                 switch( (sensor_get( MASK_1111 )) ){
235                     case 0x06:
236                         /* センターまっすっぐ                */
237                         handle( 0 );
238                         motor( 100, 100 );
239                         break;
240
241                     case 0x02:
242                         /* 少し左寄り→右へ小曲げ            */
243                         handle( 10 );
244                         motor( 90, 80 );
245                         break;
246
247                     case 0x03:
248                         /* 中くらい左寄り→右へ中曲げ        */
249                         handle( 15 );
250                         motor( 80, 60 );
251                         break;
252
253                     case 0x01:
254                         /* 大きく左寄り→右へ大曲げ          */
255                         handle( 20 );
256                         motor( 60, 40 );
257                         pattern = 12;
258                         break;
259
260                     case 0x04:
261                         /* 少し右寄り→左へ小曲げ            */
262                         handle( -10 );
263                         motor( 80, 90 );
```

```
264                         break;
265
266                     case 0x0c:
267                         /* 中くらい右寄り→左へ中曲げ          */
268                         handle( -15 );
269                         motor( 60, 80 );
270                         break;
271
272                     case 0x08:
273                         /* 大きく右寄り→左へ大曲げ            */
274                         handle( -20 );
275                         motor( 40, 60 );
276                         pattern = 13;
277                         break;
278
279                     default:
280                         break;
281                 }
282                 break;
283
284             case 12:
285                 /* 右へ大曲げの終わりのチェック                */
286                 if( check_crossline() ){/* クロスラインチェック          */
287                     pattern = 21;
288                     break;
289                 }
290                 if( check_rightline() ){/* 右ハーフラインチェック        */
291                     pattern = 51;
292                     break;
293                 }
294                 if( check_leftline() ){ /* 左ハーフラインチェック        */
295                     pattern = 61;
296                     break;
297                 }
298                 if( sensor_get( MASK_0010 ) == 0x02 ){
299                     pattern = 11;
300                 }
301                 break;
302
303             case 13:
304                 /* 左へ大曲げの終わりのチェック                */
305                 if( check_crossline() ){/* クロスラインチェック          */
306                     pattern = 21;
307                     break;
308                 }
309                 if( check_rightline() ){/* 右ハーフラインチェック        */
310                     pattern = 51;
311                     break;
312                 }
313                 if( check_leftline() ){ /* 左ハーフラインチェック        */
314                     pattern = 61;
315                     break;
316                 }
317                 if( sensor_get( MASK_0100 ) == 0x04 ){
318                     pattern = 11;
319                 }
320                 break;
321
322             case 21:
323                 /* １本目のクロスライン検出時の処理            */
324                 beep( DO_5 );           /* 5オクターブ目のドの音をセット*/
325                 handle( 0 );
326                 motor( 0, 0 );
327                 pattern = 22;
328                 cnt1 = 0;
329                 break;
330
331             case 22:
332                 /* ２本目を読み飛ばす                          */
333                 if( cnt1 > 500 ){
334                     beep( 0 );          /* ブザー停止                    */
335                     pattern = 23;
336                     cnt1 = 0;
337                 }
338                 break;
339
340             case 23:
341                 /* クロスライン後のトレース、クランク検出       */
342                 if( sensor_get( MASK_1111 ) == 0x07 ){
343                     /* 右クランクと判断→右クランククリア処理へ */
344                     handle( 35 );
345                     motor( 60, 20 );
346                     pattern = 31;
347                     cnt1 = 0;
348                     break;
349                 }
350                 if( sensor_get( MASK_1111 ) == 0x0e ){
351                     /* 左クランクと判断→左クランククリア処理へ */
352                     handle( -35 );
353                     motor( 20, 60 );
354                     pattern = 41;
```

```
355                     cnt1 = 0;
356                     break;
357                 }
358                 switch( sensor_get( MASK_1111 ) ){
359                     case 0x06:
360                         handle( 0 );
361                         motor( 60, 60 );
362                         break;
363                     case 0x02:
364                     case 0x03:
365                     case 0x01:
366                         handle( 10 );
367                         motor( 60, 50 );
368                         break;
369                     case 0x04:
370                     case 0x0c:
371                     case 0x08:
372                         handle( -10 );
373                         motor( 50, 60 );
374                         break;
375                     default:
376                         break;
377                 }
378                 break;
379
380             case 31:
381                 /* 右クランククリア処理  安定するまで少し待つ */
382                 if( cnt1 > 500 ){
383                     pattern = 32;
384                     cnt1 = 0;
385                 }
386                 break;
387
388             case 32:
389                 /* 右クランククリア処理  曲げ終わりのチェック */
390                 if( sensor_get( MASK_0100 ) == 0x04 ){
391                     pattern = 11;
392                     cnt1 = 0;
393                 }
394                 break;
395
396             case 41:
397                 /* 左クランククリア処理  安定するまで少し待つ */
398                 if( cnt1 > 500 ){
399                     pattern = 42;
400                     cnt1 = 0;
401                  }
402                  break;
403
404             case 42:
405                 /* 左クランククリア処理  曲げ終わりのチェック */
406                 if( sensor_get( MASK_0010 ) == 0x02 ){
407                     pattern = 11;
408                     cnt1 = 0;
409                 }
410                 break;
411
412             case 51:
413                 /* １本目の右ハーフライン検出時の処理         */
414                 beep( RE_5 );           /* 5オクターブ目のレの音をセット*/
415                 handle( 0 );
416                 motor( 70, 70 );
417                 pattern = 52;
418                 cnt1 = 0;
419                 break;
420
421             case 52:
422                 /* ２本目を読み飛ばす                         */
423                 if( cnt1 > 500 ){
424                     beep( 0 );          /* ブザー停止                 */
425                     pattern = 53;
426                     cnt1 = 0;
427                 }
428                 break;
429
430             case 53:
431                 /* 右ハーフライン後のトレース、レーンチェンジ */
432                 if( sensor_get( MASK_1111 ) == 0x00 ){
433                     handle( 15 );
434                     motor( 70, 60 );
435                     pattern = 54;
436                     cnt1 = 0;
437                     break;
438                 }
439                 switch( sensor_get( MASK_1111 ) ){
440                     case 0x06:
441                         handle( 0 );
442                         motor( 70, 70 );
443                         break;
444                     case 0x02:
445                     case 0x03:
```

```
446                     case 0x01:
447                         handle( 10 );
448                         motor( 70, 60 );
449                         break;
450                     case 0x04:
451                     case 0x0c:
452                     case 0x08:
453                         handle( -10 );
454                         motor( 60, 70 );
455                         break;
456                     default:
457                         break;
458                 }
459                 break;
460
461             case 54:
462                 /* 右レーンチェンジ終了のチェック              */
463                 if( sensor_get( MASK_0100 ) == 0x04 ){
464                     pattern = 11;
465                     cnt1 = 0;
466                 }
467                 break;
468
469             case 61:
470                 /* １本目の右ハーフライン検出時の処理          */
471                 beep( MI_5 );           /* 5オクターブ目のミの音をセット*/
472                 handle( 0 );
473                 motor( 70, 70 );
474                 pattern = 62;
475                 cnt1 = 0;
476                 break;
477
478             case 62:
479                 /* ２本目を読み飛ばす                          */
480                 if( cnt1 > 500 ){
481                     beep( 0 );          /* ブザー停止                   */
482                     pattern = 63;
483                     cnt1 = 0;
484                 }
485                 break;
486
487             case 63:
488                 /* 左ハーフライン後のトレース、レーンチェンジ */
489                 if( sensor_get( MASK_1111 ) == 0x00 ){
490                     handle( -15 );
491                     motor( 60, 70 );
492                     pattern = 64;
493                     cnt1 = 0;
494                     break;
495                 }
496                 switch( sensor_get( MASK_1111 ) ){
497                     case 0x06:
498                         handle( 0 );
499                         motor( 70, 70 );
500                         break;
501                     case 0x02:
502                     case 0x03:
503                     case 0x01:
504                         handle( 10 );
505                         motor( 70, 60 );
506                         break;
507                     case 0x04:
508                     case 0x0c:
509                     case 0x08:
510                         handle( -10 );
511                         motor( 60, 70 );
512                         break;
513                     default:
514                         break;
515                 }
516                 break;
517
518             case 64:
519                 /* 左レーンチェンジ終了のチェック              */
520                 if( sensor_get( MASK_0010 ) == 0x02 ){
521                     pattern = 11;
522                     cnt1 = 0;
523                 }
524                 break;
525
526             default:
527                 /* どれでもない場合は待機状態に戻す            */
528                 pattern = 0;
529                 break;
530         }
531     }
532 }
533
534 /************************************************************************/
535 /* R8C/35A スペシャルファンクションレジスタ(SFR)の初期化                */
536 /************************************************************************/
```

```
537 void init( void )
538 {
539     int i;
540
541     /* クロックをXINクロック(20MHz)に変更 */
542     prc0  = 1;                          /* プロテクト解除                */
543     cm13  = 1;                          /* P4_6,P4_7をXIN-XOUT端子にする*/
544     cm05  = 0;                          /* XINクロック発振               */
545     for(i=0; i<50; i++ );               /* 安定するまで少し待つ(約10ms) */
546     ocd2  = 0;                          /* システムクロックをXINにする  */
547     prc0  = 0;                          /* プロテクトON                  */
548
549     /* ポートの入出力設定                */
550     prc2 = 1;                           /* PD0のプロテクト解除           */
551     pd0 = 0xe0;                         /* 7-5:LED 4:MicroSW 3-0:Sensor */
552     p1  = 0x0f;                         /* 3-0:LEDは消灯                 */
553     pd1 = 0xdf;                         /* 5:RXD0 4:TXD0 3-0:LED         */
554     pd2 = 0xfe;                         /* 0:PushSW                      */
555     pd3 = 0xfb;                         /* 4:Buzzer 2:IR                 */
556     pd4 = 0x80;                         /* 7:XOUT 6:XIN 5-3:DIP SW 2:VREF*/
557     pd5 = 0x40;                         /* 7:DIP SW                      */
558     pd6 = 0xff;
559
560     /* タイマRBの設定                    */
561     /* 割り込み周期 = 1 / 20[MHz]    * (TRBPRE+1) * (TRBPR+1)
562                     = 1 / (20*10^-6) * 200        * 100
563                     = 0.001[s] = 1[ms]
564     */
565     trbmr  = 0x00;                      /* 動作モード、分周比設定        */
566     trbpre = 200-1;                     /* プリスケーラレジスタ          */
567     trbpr  = 100-1;                     /* プライマリレジスタ            */
568     trbic  = 0x07;                      /* 割り込み優先レベル設定        */
569     trbcr  = 0x01;                      /* カウント開始                  */
570
571     /* タイマRC PWMの設定(ﾌﾞｻﾞｰ用)        */
572     trcmr   = 0x0a;                     /* TRCIOC端子はPWM出力           */
573     trccr1  = 0xa0;                     /* カウントソース,初期出力の設定*/
574     trccr2  = 0x00;                     /* 出力レベルの設定              */
575     trcoer  = 0x0b;                     /* TROIOC出力許可                */
576     trcpsr1 = 0x02;                     /* TRCIOC,D端子の設定            */
577     trcgra  = 0;                        /* 周期設定                      */
578     trcgrc  = 0;                        /* ON幅設定                      */
579
580     /* タイマRD PWMの設定                */
581     trdfcr  = 0x01;                     /* リセット同期PWMモードに設定  */
582     trdmr   = 0xf0;                     /* バッファレジスタ設定          */
583     trdoer1 = 0xcd;                     /* 出力端子の選択                */
584     trdpsr0 = 0x08;                     /* TRDIOB0,C0,D0端子設定         */
585     trdpsr1 = 0x05;                     /* TRDIOA1,B1,C1,D1端子設定      */
586     trdcr0  = 0x23;                     /* ソースカウントの選択:f8       */
587     trdgra0 = trdgrc0 = PWM_CYCLE;      /* 周期                          */
588     trdgrb0 = trdgrd0 = 0;              /* P2_2端子のON幅設定(左モータ) */
589     trdgra1 = trdgrc1 = 0;              /* P2_4端子のON幅設定(右モータ) */
590     trdgrb1 = trdgrd1 = 0;              /* P2_5端子のON幅設定(サーボモータ)*/
591     trdstr  = 0x0d;                     /* TRD0カウント開始              */
592
593 }
594
595 /************************************************************************/
596 /* タイマRB 割り込み処理                                                */
597 /************************************************************************/
598 #pragma interrupt intTRB(vect=24)
599 void intTRB( void )
600 {
601     unsigned char   sensorled_data;
602
603     /*タイマカウント用変数              */
604     cnt0++;                             /* timer関数用変数               */
605     cnt1++;                             /* main関数用変数                */
606
607     if( !s_led_out_on ){
608         /* センサ値  読み込み                */
609         sensorled_data = sensor_get( MASK_1111 );
610     }else{
611         /* sensorled_out関数のデータ読み込み  */
612         sensorled_data = sensorled_outdata;
613     }
614
615     /* 実際にセンサ部のLEDへ出力する      */
616     if( p0 & 0x80 ) {
617         /* P0_7が"1"なら                  */
618         p0 &= 0x1f;                     /* P0_7～P0_5を"0"にして全消灯  */
619         if( sensorled_data & 0x8 ) p0 |= 0x40;      /* D1点灯           */
620         if( sensorled_data & 0x2 ) p0 |= 0x20;      /* D3点灯           */
621     } else {
622         /* P0_7が"0"なら                  */
623         p0 |= 0xe0;                     /* P0_7～P0_5を"1"にして全消灯  */
624         if( sensorled_data & 0x4 ) p0 &= 0xbf;      /* D2点灯           */
625         if( sensorled_data & 0x1 ) p0 &= 0xdf;      /* D4点灯           */
626     }
627
```

```
628 }
629
630 /*****************************************************************************/
631 /* タイマ(1ms)                                                               */
632 /* 引数 1秒:1000                                                             */
633 /*****************************************************************************/
634 void timer( unsigned int time )
635 {
636     cnt0 = 0;
637     while( time > cnt0 );
638 }
639
640 /*****************************************************************************/
641 /* クロスライン検出処理                                                      */
642 /* 戻り値 0:なし 1:あり                                                      */
643 /*****************************************************************************/
644 int check_crossline( void )
645 {
646     int ret;
647
648     ret = 0;
649     if( sensor_get( MASK_1111 ) == 0x0f ) {
650         ret = 1;
651     }
652
653     return ret;
654 }
655
656 /*****************************************************************************/
657 /* 右ハーフライン検出処理                                                    */
658 /* 戻り値 0:なし 1:あり                                                      */
659 /*****************************************************************************/
660 int check_rightline( void )
661 {
662     int ret;
663
664     ret = 0;
665     if( sensor_get( MASK_1111 ) == 0x07 ) {
666         ret = 1;
667     }
668
669     return ret;
670 }
671
672 /*****************************************************************************/
673 /* 左ハーフライン検出処理                                                    */
674 /* 戻り値 0:なし 1:あり                                                      */
675 /*****************************************************************************/
676 int check_leftline( void )
677 {
678     int ret;
679
680     ret = 0;
681     if( sensor_get( MASK_1111 ) == 0x0e ) {
682         ret = 1;
683     }
684     return ret;
685 }
686
687 /*****************************************************************************/
688 /* センサ値の読み込み                                                        */
689 /* 戻り値 センサ値 0～15                                                     */
690 /*****************************************************************************/
691 unsigned char sensor_get( unsigned char mask )
692 {
693     unsigned char sensor;
694
695     sensor = ~p0;
696     sensor &= 0x0f;
697     sensor &= mask;
698
699     return sensor;
700 }
701
702 /*****************************************************************************/
703 /* センサ部のLED出力                                                         */
704 /* 引数 LEDへの出力値 0～15                                                  */
705 /*****************************************************************************/
706 void sensorled_out( unsigned char led )
707 {
708     /* この関数では、LED出力値を保存するだけ */
709     sensorled_outdata = led;
710 }
711
712 /*****************************************************************************/
713 /* ディップスイッチ値読み込み                                                */
714 /* 戻り値 スイッチ値 0～15                                                   */
715 /*****************************************************************************/
716 unsigned char dipsw_get( void )
717 {
718     unsigned char sw, sw1, sw2;
```

```
719
720     sw1 = (p5>>4) & 0x08;               /* ディップスイッチ読み込み3    */
721     sw2 = (p4>>3) & 0x07;               /* ディップスイッチ読み込み2,1,0*/
722     sw  = ( sw1 | sw2 );                /* P5とP4の値を合わせる          */
723
724     return  sw;
725 }
726
727 /************************************************************************/
728 /* プッシュスイッチ値読み込み                                           */
729 /* 戻り値 プッシュスイッチの値 0:OFF 1:ON                               */
730 /************************************************************************/
731 unsigned char pushsw_get( void )
732 {
733     unsigned char sw;
734
735     sw = ~p2;                           /* プッシュスイッチ読み込み     */
736     sw &= 0x01;                         /* 不要ビットを"0"にする        */
737
738     return sw;
739 }
740
741 /************************************************************************/
742 /* マイコン部のLED出力(LEDを実装している場合、使用可)                   */
743 /* 引数 LEDへの出力値 0～15                                             */
744 /************************************************************************/
745 void led_out( unsigned char led )
746 {
747     unsigned char data;
748
749     led = ~led;
750     led &= 0x0f;
751     data = p1 & 0xf0;
752     p1 = data | led;
753 }
754
755 /************************************************************************/
756 /* モータ速度制御                                                       */
757 /* 引数    左モータ:-100～100、右モータ:-100～100                       */
758 /*         0で停止、100で正転100%、-100で逆転100%                       */
759 /* 戻り値 なし                                                          */
760 /************************************************************************/
761 void motor( int data1, int data2 )
762 {
763     int    motor_r, motor_l, sw_data;
764
765     sw_data = dipsw_get() + 5;
766     motor_l = data1 * sw_data / 20;
767     motor_r = data2 * sw_data / 20;
768
769     /* 左モータ制御 */
770     if( motor_l >= 0 ) {
771         p2 &= 0xfd;
772         p2 |= 0x40;
773         trdgrd0 = (long)( PWM_CYCLE - 1 ) * motor_l / 100;
774     } else {
775         p2 |= 0x02;
776         p2 &= 0xbf;
777         trdgrd0 = (long)( PWM_CYCLE - 1 ) * ( -motor_l ) / 100;
778     }
779
780     /* 右モータ制御 */
781     if( motor_r >= 0 ) {
782         p2 &= 0xf7;
783         p2 |= 0x80;
784         trdgrc1 = (long)( PWM_CYCLE - 1 ) * motor_r / 100;
785     } else {
786         p2 |= 0x08;
787         p2 &= 0x7f;
788         trdgrc1 = (long)( PWM_CYCLE - 1 ) * ( -motor_r ) / 100;
789     }
790 }
791
792 /************************************************************************/
793 /* サーボハンドル操作                                                   */
794 /* 引数    サーボ角度：-90～90                                          */
795 /*         -90で左へ90度、0でまっすぐ、90で右へ90度                      */
796 /************************************************************************/
797 void handle( int angle )
798 {
799     /* サーボが左右逆に動く場合は、「-」を「+」に替えてください */
800     trdgrd1 = SERVO_CENTER - angle * HANDLE_STEP;
801 }
802
803 /************************************************************************/
804 /* ブザーを鳴らす                                                       */
805 /* 引数  音階のPWM値                                                    */
806 /************************************************************************/
807 void beep( unsigned int tone )
808 {
809     if( tone ) {
```

```
810         trcmr  &= 0x7f;                 /* TRCのカウント停止           */
811         trc     = tone - 1;
812         trcgra  = tone - 1;             /* 周期設定                   */
813         trcgrc  = tone / 2 - 1;         /* ON幅設定                   */
814         trcmr  |= 0x80;                 /* TRCのカウント開始           */
815     } else {
816         trcmr  &= 0x7f;                 /* TRCのカウント停止           */
817         trc     = 0;
818         trcgra  = 1;                    /* 周期設定                   */
819         trcgrc  = 1;                    /* ON幅設定                   */
820         trcmr  |= 0x80;                 /* TRCのカウント開始           */
821     }
822 }
823
824 /************************************************************************/
825 /* end of file                                                          */
826 /************************************************************************/
```

9.2 コメント文

```
001 /************************************************************************************/
002 /* 対象マイコン R8C/35A                                                             */
003 /* ﾌｧｲﾙ内容      ミニマイコンカーVer.2  サーボステアリング4輪セット                  */
004 /* バージョン    Ver.1.00                                                           */
005 /* Date          2010.04.01                                                         */
006 /* Copyright     ルネサスエレクトロニクス  ルネサスマイコンカーラリー事務局         */
007 /*               日立インターメディックス株式会社                                   */
008 /************************************************************************************/
```

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コンパイラに無視されるので、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）

```
010 /*======================================*/
011 /* インクルード                         */
012 /*======================================*/
013 #include "sfr_r835a.h"                  /* R8C/35A SFRの定義ファイル     */
```

「#include」はインクルードと読み、外部ファイルを取り込む命令です。

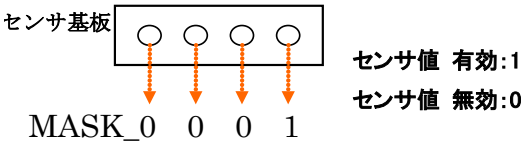
sfr_r835a.h	R8C/35A マイコンの周辺機能を制御するためのレジスタ(Special Function Registers)を定義したファイルです。

9.4 その他シンボル定義

```
015 /*======================================*/
016 /* シンボル定義                         */
017 /*======================================*/
018
019 /* 定数設定                  */
020 #define     PWM_CYCLE       39999       /* TRD周期16ms                  */
021
022 #define     SERVO_CENTER    3749        /* サーボのセンタ値             */
023 #define     HANDLE_STEP     22          /* 1°分の値                    */
```

PWM_CYCLE	PWM サイクルは、R8C/35A 内蔵周辺機能であるタイマ RD の PWM 周期を設定します。 今回は周期を 16ms に設定します。 計算は、 $(16\times10^{-3})\div(400\times10^{-9})-1=39999$ **※(400×10^{-9})は、CPU の動作周波数（20MHz）を 8 分周した値です。**
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、サーボがまっすぐ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。 $(1.5\times10^{-3})\div(400\times10^{-9})-1=3749$ となります。 **※サーボ自体の個体差やサーボホーンの固定の仕方などの影響で、4 輪ミニマイコンカーの車体それぞれ違う値になります。 この値は、プログラムでサーボモータを 0 度にしたときに、4 輪ミニマイコンカーがまっすぐ走るように調整、変更します。**
HANDLE_STEP	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、 $(2.3\times10^{-3})\div(400\times10^{-9})=5750$ サーボが、左に 90 度向くときは 0.7ms のパルスなので、 $(0.7\times10^{-3})\div(400\times10^{-9})=1750$ （右 90 度）－（左 90 度）＝4000 が 180 度分動く値です。これを 180 で割れば 1 度当たりの増減量が分かります。 4000÷180＝22.22 正確に計算すると 22 がサーボ 1 度分の値です。小数点以下は切り捨てます。

```
025 /* マスク値設定 ×：マスクあり(無効)  ○：マスク無し(有効)  */
026 #define          MASK_0001    0x01       /* ×××○                  */
027 #define          MASK_0010    0x02       /* ××○×                  */
028 #define          MASK_0011    0x03       /* ××○○                  */
029 #define          MASK_0100    0x04       /* ×○××                  */
030 #define          MASK_0101    0x05       /* ×○×○                  */
031 #define          MASK_0110    0x06       /* ×○○×                  */
032 #define          MASK_0111    0x07       /* ×○○○                  */
033 #define          MASK_1000    0x08       /* ○×××                  */
034 #define          MASK_1001    0x09       /* ○××○                  */
035 #define          MASK_1010    0x0a       /* ○×○×                  */
036 #define          MASK_1011    0x0b       /* ○×○○                  */
037 #define          MASK_1100    0x0c       /* ○○××                  */
038 #define          MASK_1101    0x0d       /* ○○×○                  */
039 #define          MASK_1110    0x0e       /* ○○○×                  */
040 #define          MASK_1111    0x0f       /* ○○○○                  */
```

MASK_xxxx	センサの状態をマスクする値です。センサ基板にセンサが 4 個付いています。 「MASK_xxxx」のx部分は、センサ 4 個を表しています。下図のような関係です。  「MASK_0001」を例に説明します。「MASK_0001」は右端のセンサ1個を有効にし、左側のセンサ 3 個は“0”を検出しても、“1”を検出しても強制的に“0”としてセンサ値を読み込みます。これがマスクです。 サンプルプログラムでは、15 パターンのマスクを用意しています。

```
042 /* 音階 ブザー用            */
043 /* 3オクターブ目の音階      */
044 #define         DO_3        38226       /* ド                   */
045 #define         DOU_3       36081       /* ド＃                 */
046 #define         RE_3        34056       /* レ                   */
047 #define         REU_3       32144       /* レ＃                 */
048 #define         MI_3        30340       /* ミ                   */
049 #define         FA_3        28637       /* ファ                 */
050 #define         FAU_3       27030       /* ファ＃               */
051 #define         SO_3        25513       /* ソ                   */
052 #define         SOU_3       24081       /* ソ＃                 */
053 #define         RA_3        22729       /* ラ                   */
054 #define         RAU_3       21454       /* ラ＃                 */
055 #define         SI_3        20250       /* シ                   */
056
057 /* 4オクターブ目の音階      */
058 #define         DO_4        19113       /* ド                   */
059 #define         DOU_4       18040       /* ド＃                 */
060 #define         RE_4        17028       /* レ                   */
061 #define         REU_4       16072       /* レ＃                 */
062 #define         MI_4        15170       /* ミ                   */
063 #define         FA_4        14319       /* ファ                 */
064 #define         FAU_4       13515       /* ファ＃               */
065 #define         SO_4        12756       /* ソ                   */
066 #define         SOU_4       12041       /* ソ＃                 */
067 #define         RA_4        11365       /* ラ                   */
068 #define         RAU_4       10727       /* ラ＃                 */
069 #define         SI_4        10125       /* シ                   */
070
071 /* 5オクターブ目の音階      */
072 #define         DO_5        9557        /* ド                   */
073 #define         DOU_5       9020        /* ド＃                 */
074 #define         RE_5        8514        /* レ                   */
075 #define         REU_5       8036        /* レ＃                 */
076 #define         MI_5        7585        /* ミ                   */
077 #define         FA_5        7159        /* ファ                 */
078 #define         FAU_5       6758        /* ファ＃               */
079 #define         SO_5        6378        /* ソ                   */
080 #define         SOU_5       6020        /* ソ＃                 */
081 #define         RA_5        5682        /* ラ                   */
082 #define         RAU_5       5363        /* ラ＃                 */
083 #define         SI_5        5062        /* シ                   */
```

ブザー用音階	R8C/35A の周辺機能であるタイマ RC の PWM を使用して、音階を鳴らします。 音階とは、「ドレミファソラシド」のことです。この音階の周波数が分かれば周期が分かりますので、デューティ比 50%の PWM をブザーに出力すれば、「ドレミファソラシド」と音を鳴らすことができます。 音階は、「ド」から次の高い「ド」まで「ド、ド＃、レ、レ＃、ミ、ファ、ファ＃、ソ、ソ＃、ラ、ラ＃、シ」の 12 段階あります。12 段階を1オクターブといいます。 4オクターブ目のドの音の周波数は 261.1Hz です。12 段階の周波数は、次の式で求めることができます。 $周波数 = 261.6 \times 2^{(x/12)}[Hz]$ xは、ドが 0、ド＃が 1‥‥、シが 11 というように一つずつ増えていきます。 1 つ高い 5 オクターブ目のドの周波数は、2 倍の 523.2Hz となります。xに当たる部分は、12 になります。 1 つ低い 3 オクターブ目のドの周波数は、1/2 の 130.8Hz となります。xに当たる部分は、-12 になります。 4 オクターブ目のドの音を例に、周波数からタイマ RC の PWM の値を計算してみます。 $(1 \div 261.6) \div (200 \times 10^{-9}) = 19113.15$ となります。4 オクターブ目のドに設定する値は 19113 となります。

9.5 プロトタイプ宣言

```
085 /*====================================*/
086 /* プロトタイプ宣言                   */
087 /*====================================*/
088 void init( void );
089 void timer( unsigned int time );
090 int check_crossline( void );
091 int check_rightline( void );
092 int check_leftline( void );
093 unsigned char sensor_get( unsigned char mask );
094 void sensorled_out( unsigned char led );
095 unsigned char dipsw_get( void );
096 unsigned char pushsw_get( void );
097 void led_out( unsigned char led );
098 void motor( int data1, int data2 );
099 void handle( int angle );
100 void beep( unsigned int tone );
```

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。

9.6 グローバル変数の宣言

```
102 /*====================================*/
103 /* グローバル変数                     */
104 /*====================================*/
105 unsigned long   cnt0;                  /* timer関数用変数            */
106 unsigned long   cnt1;                  /* main関数内用               */
107
108 /* sensorled_out関数用変数 */
109 unsigned char   sensorled_outdata;     /* 出力データ保存用変数       */
110 int             s_led_out_on;          /* 関数のON/OFF切り替え       */
111
112 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、ローカル変数といい、その関数の中でのみ参照できる変数のことです。

「mini_car_ver2.c」のプログラムでは、5 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。
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。
sensorled_outdata	unsigned char	この変数は、sensorled_out 関数内で出力データ保存用として使用します。
s_led_out_on	int	sensorled_out 関数を使用する場合は、1 を書き込み、 sensorled_out 関数を使用しない場合は、0 を書き込みます。
pattern	int	パターン番号です。

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、これらの変数の初期値は 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

メインプログラム(114～533 行)に 4 輪ミニマイコンカーの制御の中心となる 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分かりやすいので、main 関数は最後に説明します。

9.8 R8C/35A 周辺機能の初期化：init 関数

9.8.1 プログラム

R8C/35A マイコンの周辺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「init」関数です。

```
534 /************************************************************************/
535 /* R8C/35A スペシャルファンクションレジスタ(SFR)の初期化                */
536 /************************************************************************/
537 void init( void )
538 {
539     int i;
540
541     /* クロックをXINクロック(20MHz)に変更 */
542     prc0  = 1;                          /* プロテクト解除                */
543     cm13  = 1;                          /* P4_6,P4_7をXIN-XOUT端子にする*/
544     cm05  = 0;                          /* XINクロック発振               */
545     for(i=0; i<50; i++ );               /* 安定するまで少し待つ(約10ms) */
546     ocd2  = 0;                          /* システムクロックをXINにする  */
547     prc0  = 0;                          /* プロテクトON                  */
548
549     /* ポートの入出力設定                 */
550     prc2 = 1;                           /* PD0のプロテクト解除           */
551     pd0 = 0xe0;                         /* 7-5:LED 4:MicroSW 3-0:Sensor */
552     p1  = 0x0f;                         /* 3-0:LEDは消灯                 */
553     pd1 = 0xdf;                         /* 5:RXD0 4:TXD0 3-0:LED         */
554     pd2 = 0xfe;                         /* 0:PushSW                      */
555     pd3 = 0xfb;                         /* 4:Buzzer 2:IR                 */
556     pd4 = 0x80;                         /* 7:XOUT 6:XIN 5-3:DIP SW 2:VREF*/
557     pd5 = 0x40;                         /* 7:DIP SW                      */
558     pd6 = 0xff;
559
560     /* タイマRBの設定                     */
561     /* 割り込み周期 = 1 / 20[MHz]    * (TRBPRE+1) * (TRBPR+1)
562                     = 1 / (20*10^-6) * 200        * 100
563                     = 0.001[s] = 1[ms]
564     */
565     trbmr  = 0x00;                      /* 動作モード、分周比設定        */
566     trbpre = 200-1;                     /* プリスケーラレジスタ          */
567     trbpr  = 100-1;                     /* プライマリレジスタ            */
568     trbic  = 0x07;                      /* 割り込み優先レベル設定        */
569     trbcr  = 0x01;                      /* カウント開始                  */
570
571     /* タイマRC PWMの設定(ﾌﾞｻﾞｰ用)        */
572     trcmr   = 0x0a;                     /* TRCIOC端子はPWM出力           */
573     trccr1  = 0xa0;                     /* カウントソース,初期出力の設定*/
574     trccr2  = 0x00;                     /* 出力レベルの設定              */
575     trcoer  = 0x0b;                     /* TROIOC出力許可                */
576     trcpsr1 = 0x02;                     /* TRCIOC,D端子の設定            */
577     trcgra  = 0;                        /* 周期設定                      */
578     trcgrc  = 0;                        /* ON幅設定                      */
579
```

```
580     /* タイマRD PWMの設定                    */
581     trdfcr  = 0x01;                         /* リセット同期PWMモードに設定  */
582     trdmr   = 0xf0;                         /* バッファレジスタ設定         */
583     trdoer1 = 0xcd;                         /* 出力端子の選択               */
584     trdpsr0 = 0x08;                         /* TRDIOB0,C0,D0端子設定        */
585     trdpsr1 = 0x05;                         /* TRDIOA1,B1,C1,D1端子設定     */
586     trdcr0  = 0x23;                         /* ソースカウントの選択:f8      */
587     trdgra0 = trdgrc0 = PWM_CYCLE;          /* 周期                         */
588     trdgrb0 = trdgrd0 = 0;                  /* P2_2端子のON幅設定(左モータ) */
589     trdgra1 = trdgrc1 = 0;                  /* P2_4端子のON幅設定(右モータ) */
590     trdgrb1 = trdgrd1 = 0;                  /* P2_5端子のON幅設定(サーボモータ)*/
591     trdstr  = 0x0d;                         /* TRD0カウント開始             */
592
593 }
```

9.8.2 レジスタ設定一覧表

機能	レジスタ	設定値	詳細
クロックを XIN クロック (20MHz)に変更	prc0	1	プロテクト解除
	cm13	1	P4_6,P4_7 を XIN-XOUT 端子にする
	cm05	0	XIN クロック発振 発振させた後、安定するまで少し待ちます(約 10ms)
	ocd2	0	システムクロックを XIN にする
	prc0	0	プロテクト ON
ポートの入出力設定	prc2	1	PD0 のプロテクト解除
	pd0	0xe0	7-5:LED 4:MicroSW 3-0:Sensor
	p1	0x0f	3-0:LED は消灯
	pd1	0xdf	5:RXD0 4:TXD0 3-0:LED
	pd2	0xfe	0:PushSW
	pd3	0xfb	4:Buzzer 2:IR
	pd4	0x80	7:XOUT 6:XIN 5-3:DIP SW 2:VREF
	pd5	0x40	7:DIP SW
	pd6	0xff	7:DIP SW
タイマ RB の設定	trbmr	0x00	動作モード、分周比設定
	trbpre	200-1	プリスケーラレジスタとプライマリレジスタをペアで使い割り込みの周期を設定
	trbpr	100-1	
	trbic	0x07	割り込み優先レベル設定
	trbcr	0x01	タイマ RB の動作を開始

機能	レジスタ	設定値	詳細
タイマ RC PWM の設定(ﾌﾞｻﾞｰ用)	trcmr	0x0a	TRCIOC 端子は PWM 出力
	trccr1	0xa0	カウントソース,初期出力の設定
	trccr2	0x00	出力レベルの設定
	trcoer	0x0b	TROIOC 出力許可
	trcpsr1	0x02	TRCIOC,D 端子の設定
	trcgra	0	周期設定
	trcgrc	0	ON 幅設定
タイマ RD PWM の設定(左右モータ、サーボモータ駆動用)	trdfcr	0x01	リセット同期 PWM モードに設定
	trdmr	0xf0	バッファレジスタ設定
	trdoer1	0xcd	出力端子の選択
	trdpsr0	0x08	TRDIOB0,C0,D0 端子設定
	trdpsr1	0x05	TRDIOA1,B1,C1,D1 端子設定
	trdcr0	0x23	ソースカウントの選択:f8
	trdgra0 trdgrc0	PWM_CYCLE (39999)	周期
	trdgrb0 trdgrd0	0	P2_2 端子の ON 幅設定(左モータ)
	trdgra1 trdgrc1	0	P2_4 端子の ON 幅設定(右モータ)
	trdgrb1 trdgrd1	0	P2_5 端子の ON 幅設定(サーボモータ)
	trdstr	0x0d	TRD0 カウント開始

詳しくは、「R8C/35Aマイコン実習マニュアル」を参照してください。

※「R8C/35A マイコン実習マニュアル」は、こちらの URL(http://www2.himdx.net/mcr/)よりダウンロードできます。

9.8.3 ポートの接続

R8C/35A ボードにはポート 0～6 まであります。4 輪ミニマイコンカーは、下記のように接続されています。

ポート	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
0	**LEDC 出力**	**LEDB 出力**	**LEDA 出力**	**リミットスイッチ入力**	**センサ 3 入力**	**センサ 2 入力**	**センサ 1 入力**	**センサ 0 入力**
1	未接続	未接続	**RxD0 入力**	**TxD0 出力**	**マイコンボード上の LED3 出力**	**マイコンボード上の LED2 出力**	**マイコンボード上の LED1 出力**	**マイコンボード上の LED0 出力**
2	**モータ右 2 出力**	**モータ左 2 出力**	**サーボ PWM 出力**	**モータ右 PWM 出力**	**モータ右 1 出力**	**モータ左 PWM 出力**	**モータ左 1 出力**	**マイコンボード上のタクトスイッチ入力**
3	未接続	未接続	未接続	**マイコンボード上のブザー出力**	未接続	**マイコンボード上の赤外線受光 IC 入力**	未接続	未接続
4	**クリスタル出力**	**クリスタル入力**	**マイコンボード上の SW2 入力**	**マイコンボード上の SW1 入力**	**マイコンボード上の SW0 入力**	**Vcc 入力**		
5	**マイコンボード上の SW3 入力**	未接続						
6	**未接続**	**未接続**	**未接続**	**未接続**	**未接続**	**未接続**	**未接続**	**未接続**

※表の斜線の bit は、端子がない bit です。
※リセット後は、全て入力ポートです。

9.8.4 入出力を決める

ポートの入出力設定は下記のようにします。

出力	出力端子は出力に設定します。外部へ信号を出力する端子は、“1”に設定します。
入力	入力端子は入力に設定します。外部から信号を入力する端子は、“0”に設定します。
未接続	未接続端子の端子は、プルアップ抵抗、またはプルダウン抵抗を接続して入力端子にするか、何も接続せずに出力端子にします。今回は、出力端子に設定します。
斜線	端子のないビットです。表の斜線部は、“0”に設定します。

9.8.5 実際の設定

ポートの入出力設定は、**「ポート Pi 方向レジスタ」(i は 0～6 の数字が入ります)で行い、レジスタ名は「pd0～pd6」の 7 個あります**。最後の数字がポート番号です。例えば、ポート 0 は「pd0」となります。

ポート Pi 方向レジスタの設定方法を下記に示します。

ポート	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	**16 進数**
PD0	1	1	1	0	0	0	0	0	**0xe0**
PD1	1	1	0	1	1	1	1	1	**0xdf**
PD2	1	1	1	1	1	1	1	0	**0xfe**
PD3	1	1	1	1	1	0	1	1	**0xfb**
PD4	1	0	0	0	0	0	0	0	**0x80**
PD5	0	1	0	0	0	0	0	0	**0x40**
PD6	1	1	1	1	1	1	1	1	**0xff**

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

プログラムを下記に示します。

```
549     /* ポートの入出力設定                    */
550     prc2 = 1;                          /* PD0のプロテクト解除          */
551     pd0 = 0xe0;                        /* 7-5:LED 4:MicroSW 3-0:Sensor */
552     p1  = 0x0f;                        /* 3-0:LEDは消灯                */
553     pd1 = 0xdf;                        /* 5:RXD0 4:TXD0 3-0:LED        */
554     pd2 = 0xfe;                        /* 0:PushSW                     */
555     pd3 = 0xfb;                        /* 4:Buzzer 2:IR                */
556     pd4 = 0x80;                        /* 7:XOUT 6:XIN 5-3:DIP SW 2:VREF*/
557     pd5 = 0x40;                        /* 7:DIP SW                     */
558     pd6 = 0xff;
```

9.9 タイマ RB による割り込み関数

9.9.1 intTRB 関数（1ms ごとに実行される関数）

intTRB 関数は、割り込みが発生したときに実行される関数です。

```
595 /************************************************************************/
596 /* タイマRB 割り込み処理                                                */
597 /************************************************************************/
598 #pragma interrupt intTRB(vect=24)
599 void intTRB( void )
600 {
601     unsigned char   sensorled_data;
602
603     /*タイマカウント用変数                */
604     cut0++;                               /* timer関数用変数              */
605     cut1++;                               /* main関数用変数               */
606
607     if( !s_led_out_on ){
608         /* センサ値　読み込み                  */
609         sensorled_data = sensor_get( MASK_1111 );
610     }else{
611         /* sensorled_out関数のデータ読み込み  */
612         sensorled_data = sensorled_outdata;
613     }
614
615     /* 実際にセンサ部のLEDへ出力する       */
616     if( p0 & 0x80 ) {
617         /* P0_7が"1"なら                   */
618         p0 &= 0x1f;                       /* P0_7～P0_5を"0"にして全消灯  */
619         if( sensorled_data & 0x8 ) p0 |= 0x40;      /* D1点灯           */
620         if( sensorled_data & 0x2 ) p0 |= 0x20;      /* D3点灯           */
621     } else {
622         /* P0_7が"0"なら                   */
623         p0 |= 0xe0;                       /* P0_7～P0_5を"1"にして全消灯  */
624         if( sensorled_data & 0x4 ) p0 &= 0xbf;      /* D2点灯           */
625         if( sensorled_data & 0x1 ) p0 &= 0xdf;      /* D4点灯           */
626     }
627
628 }
```

598 行	#pragma interrupt **割り込み処理関数名**(vect=**ソフトウェア割り込み番号**) とすることで、ソフトウェア割り込み番号の割り込みが発生したとき、割り込み処理関数名を実行します。今回は、ソフトウェア割り込み番号 24 番の割り込みが発生したとき、intTRB 関数を実行します。
599 行	タイマ RB 割り込みにより実行する関数です。割り込み関数は、引数、戻り値ともに指定することはできません。要は、「void 関数名(void)」である必要があります。

※ソフトウェア割り込み番号

割り込み要因とソフトウェア割り込み番号の関係は、下記のとおりです。

割り込み要因	ベクタ番地(注1) 番地(L)～番地(H)	ソフトウェア割り込み番号	割り込み制御レジスタ	参照先
BRK命令(注3)	+0～+3(0000h～0003h)	0	—	R8C/Tinyシリーズ ソフトウェアマニュアル
フラッシュメモリレディ	+4～+7(0004h～0007h)	1	FMRDYIC	32. フラッシュメモリ
—(予約)		2～5	—	—
$\overline{\text{INT4}}$	+24～+27(0018h～001BFh)	6	INT4IC	11.4 $\overline{\text{INT}}$割り込み
タイマRC	+28～+31(001Ch～001Fh)	7	TRCIC	19. タイマRC
タイマRD0	+32～+35(0020h～0023h)	8	TRD0IC	20. タイマRD
タイマRD1	+36～+39(0024h～0027h)	9	TRD1IC	
タイマRE	+40～+43(0028h～002Bh)	10	TREIC	21. タイマRE
UART2送信/NACK2	+44～+47(002Ch～002Fh)	11	S2TIC	23. シリアルインタフェース(UART2)
UART2受信/ACK2	+48～+51(0030h～0033h)	12	S2RIC	
キー入力	+52～+55(0034h～0037h)	13	KUPIC	11.5 キー入力割り込み
A/D変換	+56～+59(0038h～003Bh)	14	ADIC	28. A/Dコンバータ
シンクロナスシリアルコミュニケーションユニット/I^2Cバスインタフェース(注2)	+60～+63(003Ch～003Fh)	15	SSUIC/ IICIC	25. シンクロナスシリアルコミュニケーションユニット(SSU)、 26. I^2Cバスインタフェース
—(予約)		16	—	—
UART0送信	+68～+71(0044h～0047h)	17	S0TIC	22. シリアルインタフェース(UARTi (i=0～1))
UART0受信	+72～+75(0048h～004Bh)	18	S0RIC	
UART1送信	+76～+79(004Ch～004Fh)	19	S1TIC	
UART1受信	+80～+83(0050h～0053h)	20	S1RIC	
$\overline{\text{INT2}}$	+84～+87(0054h～0057h)	21	INT2IC	11.4 $\overline{\text{INT}}$割り込み
タイマRA	+88～+91(0058h～005Bh)	22	TRAIC	17. タイマRA
—(予約)		23	—	—
タイマRB	+96～+99(0060h～0063h)	24	TRBIC	18. タイマRB
$\overline{\text{INT1}}$	+100～+103(0064h～0067h)	25	INT1IC	11.4 $\overline{\text{INT}}$割り込み
$\overline{\text{INT3}}$	+104～+107(0068h～006Bh)	26	INT3IC	
—(予約)		27	—	—
—(予約)		28	—	—
$\overline{\text{INT0}}$	+116～+119(0074h～0077h)	29	INT0IC	11.4 $\overline{\text{INT}}$割り込み
UART2バス衝突検出	+120～+123(0078h～007Bh)	30	U2BCNIC	23. シリアルインタフェース(UART2)
—(予約)		31	—	—
ソフトウェア(注3)	+128～+131(0080h～0083h)～ +164～+167(00A4h～00A7h)	32～41	—	R8C/Tinyシリーズ ソフトウェアマニュアル
—(予約)		42～49	—	—
電圧監視1/コンパレータA1	+200～+203(00C8h～00CBh)	50	VCMP1IC	6. 電圧検出回路 30. コンパレータA
電圧監視2/コンパレータA2	+204～+207(00CCh～00CFh)	51	VCMP2IC	
—(予約)		52～55	—	—
ソフトウェア(注3)	+224～+227(00E0h～00E3h)～ +252～+255(00FCh～00FFh)	56～63		R8C/Tinyシリーズ ソフトウェアマニュアル

注1. INTBレジスタが示す番地からの相対番地です。
注2. SSUIICSRレジスタのIICSELビットで選択できます。
注3. Iフラグによる禁止はできません。

今回は、タイマRBを使用して割り込みを発生させるので、表より番号は24番となります。

次のように、#pragma interrupt 命令を記述して、「ソフトウェア割り込み番号 24 番が発生したときに、実行する関数は割り込み処理関数名です」ということを、宣言します。

```
#pragma interrupt 割り込み処理関数名(vect=24)
```

関数名のプログラムを記述して、割り込みが発生したときに実行するプログラムを作成します。

```
void 割り込み処理関数名( void )
{
    プログラム
}
```

9.9.2 全体の割り込み許可

init 関数内でタイマ RB の割り込み設定と割り込み関数を作成しました。これだけでは割り込みは発生しません。init 関数を実行した後に「全体の割り込み許可」の設定をします。

```
114 /************************************************************************/
115 /* メインプログラム                                                     */
116 /************************************************************************/
117 void main( void )
118 {
119     init();                         /* 初期化                           */
120     asm("fset I");                  /* 全体割り込み許可                 */
121
```

120 行	全体の割り込みを許可する命令です。 init 関数内でタイマ RB の割り込みを許可していますが、全体の割り込みを許可しなければ割り込みは発生しません。全体の割り込みを許可する命令は、C 言語で記述することができないため、asm 命令を使ってアセンブリ言語で割り込みを許可する命令を記述しています。

9.10 時間稼ぎ：timer 関数

この関数を実行すると時間稼ぎとして、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。

```
630 /*********************************************************************/
631 /* タイマ(1ms)                                                       */
632 /* 引数 1秒:1000                                                     */
633 /*********************************************************************/
634 void timer( unsigned int time )
635 {
636     cnt0 = 0;
637     while( time > cnt0 );
638 }
```

timer 関数を使用し、1000ms の時間稼ぎした場合の動作を説明します。

```
timer( 1000 );
```

引数 time には、1000 が入ります。
636 行目で cnt0 を 0 にしているので、637 行目は、

```
while( 1000 > 0 );
```

となります。これではカッコの中が常に成り立ってしまいますが、cnt0 は 1ms ごとに割り込みが発生して実行される「intTRB」関数内で＋1しています。
そのため、1ms 後には、

```
while( 1000 > 1 );
```

となります。どんどん時間がたっていき、きっかり 1000ms 後に、

```
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、次の行へ進みます。その結果、637 行目で 1000ms 待つ、関数の名前の通り、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。4 輪ミニ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、4 輪ミニ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。

9.11 センサ値の読み込み：sensor_get 関数

この関数を実行すると、センサ値の読み込みを行います。

```
687 /*****************************************************************************/
688 /* センサ値の読み込み                                                        */
689 /* 戻り値 センサ値 0～15                                                     */
690 /*****************************************************************************/
691 unsigned char sensor_get( unsigned char mask )
692 {
693     unsigned char sensor;
694
695     sensor = ~p0;
696     sensor &= 0x0f;
697     sensor &= mask;
698
699     return sensor;
700 }
```

4 輪ミニマイコンカーのセンサ信号は、CPU ボードの P0_0～P0_3 に接続されていますので P0 レジスタから読み込みます。

P0 レジスタは、ラインが白で“0”(0V)、黒で“1”(5V)になります。そこで、白のときにプログラム上で“1”にしたいので、695 行目で P0 レジスタのデータを[~](NOT)で反転させます。

```
695     sensor = ~p0;
```

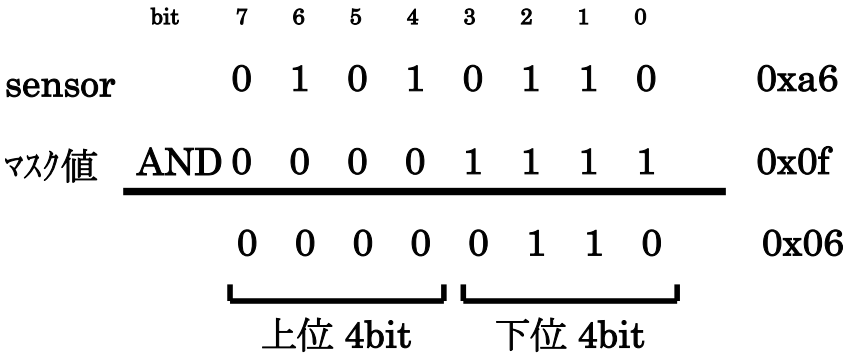
次に、センサが接続されている bit は、下位 4bit(P0_0～P0_3)です。上位 4bit のデータは不要ですので、696 行目で強制的に“0”にします。

```
696     sensor &= 0x0f;
```

「&」は、論理演算の AND です。AND 演算とは、2 つの変数 A と B があるとき(それぞれ 0 か 1 の数値)、ともに 1 であるときのみ 1 になる演算を言います。下表に真理値表を示します。

A(変数:sensor)	B	A and B
0	0	0
1	0	0
0	1	0
1	1	1

696 行目のプログラムを見てみます。変数「sensor」に「0xa6」という値が入っているとします。変数「sensor」に「0x0f」で AND 演算を行っています。下図のようになります。



そうすると、変数「sensor」の上位 4bit は、“0”になり、下位 4bit は「0110」の値だけを取り出すことができました。このように**不要な bit を強制的に“0”にすることを「マスク処理」といいます。**

ここまでで、センサの値を取り出すことができました。

```
687 /*********************************************************************/
688 /* センサ値の読み込み                                                */
689 /* 戻り値 センサ値 0～15                                             */
690 /*********************************************************************/
691 unsigned char sensor_get( unsigned char mask )
692 {
693     unsigned char sensor;
694
695     sensor = ~p0;
696     sensor &= 0x0f;
697     sensor &= mask;
698
699     return sensor;
700 }
```

引数

次に 697 行目では、センサの見たい bit だけを取り出します。プログラムを作成していると何ビット目のセンサ値だけを見たいということが良くあります。そのためマスク処理しやすいように、sensor_get 関数の引数にマスク値を入れられるようにしました。

そこで、下記のようにマスク値をあらかじめ定義しておきます。

```
025 /* マスク値設定 ×：マスクあり(無効)  ○：マスク無し(有効)  */
026 #define     MASK_0001   0x01    /* ×××○                    */
027 #define     MASK_0010   0x02    /* ××○×                    */
028 #define     MASK_0011   0x03    /* ××○○                    */
029 #define     MASK_0100   0x04    /* ×○××                    */
030 #define     MASK_0101   0x05    /* ×○×○                    */
031 #define     MASK_0110   0x06    /* ×○○×                    */
032 #define     MASK_0111   0x07    /* ×○○○                    */
033 #define     MASK_1000   0x08    /* ○×××                    */
034 #define     MASK_1001   0x09    /* ○××○                    */
035 #define     MASK_1010   0x0a    /* ○×○×                    */
036 #define     MASK_1011   0x0b    /* ○×○○                    */
037 #define     MASK_1100   0x0c    /* ○○××                    */
038 #define     MASK_1101   0x0d    /* ○○×○                    */
039 #define     MASK_1110   0x0e    /* ○○○×                    */
040 #define     MASK_1111   0x0f    /* ○○○○                    */
```

sensor_get 関数の使用方法は、下記のようになります。

```
        sensor_get( MASK_1111 );
```

と書き換えることができます。
ちなみに「MASK_1111」は、4 つのセンサ全て有効になります。

9.12 クロスライン検出処理：check_crossline 関数

クランク(直角カーブ)の手前には、必ず 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です。戻り値は、クロスラインと判断すると“1”、クロスラインでなければ“0”とします。

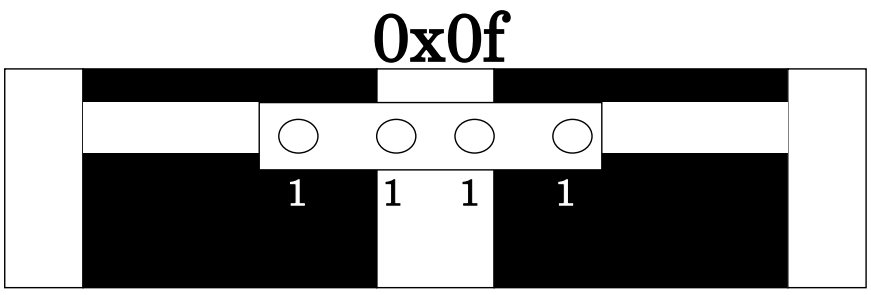
```
640 /*************************************************************************/
641 /* クロスライン検出処理                                                  */
642 /* 戻り値 0:なし 1:あり                                                  */
643 /*************************************************************************/
644 int check_crossline( void )
645 {
646     int ret;
647
648     ret = 0;
649     if( sensor_get( MASK_1111 ) == 0x0f ) {
650         ret = 1;
651     }
652
653     return ret;
654 }
```

```
648     ret = 0;
```

648 行目では戻り値を保存する ret 変数の初期化をしています。クロスラインと判断すると“1”、違うと判断すると“0”を代入します。今のところどちらか分からないので、とりあえず違うと判断して“0”を入れておきます。

```
649     if( sensor_get( MASK_1111 ) == 0x0f ) {
650         ret = 1;
651     }
```

649 行目の sensor_get 関数でセンサ値を読み込みます。マスク値は「MASK_1111」を使用していますので、センサ 4 つ全ての反応を見ます。センサ値と「0x0f」を比較した結果、一致していたらクロスラインと判断して ret 変数に“1”を代入します。



9.13 右ハーフライン検出処理：check_rightline 関数

右レーンチェンジの手前に 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です。戻り値は、右ハーフラインと判断すると“1”、右ハーフラインでなければ“0”とします。

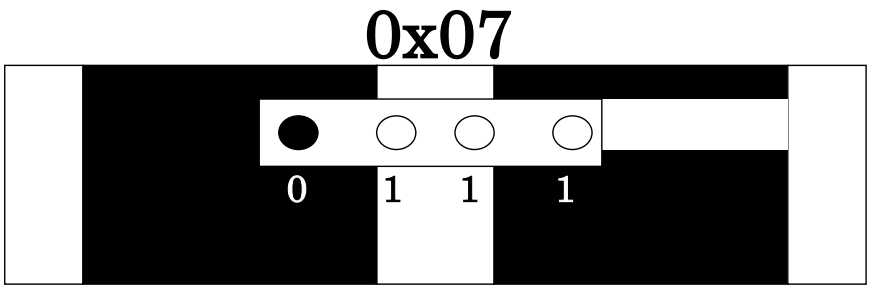
```
656 /*****************************************************************************/
657 /* 右ハーフライン検出処理                                                    */
658 /* 戻り値 0:なし 1:あり                                                      */
659 /*****************************************************************************/
660 int check_rightline( void )
661 {
662     int ret;
663
664     ret = 0;
665     if( sensor_get( MASK_1111 ) == 0x07 ) {
666         ret = 1;
667     }
668
669     return ret;
670 }
```

```
664     ret = 0;
```

664 行目では戻り値を保存する ret 変数の初期化をしています。右ハーフラインと判断すると“1”、違うと判断すると“0”を代入します。今のところどちらか分からないので、とりあえず違うと判断して“0”を入れておきます。

```
665     if( sensor_get( MASK_1111 ) == 0x07 ) {
666         ret = 1;
667     }
```

665 行目の sensor_get 関数でセンサ値を読み込みます。マスク値は「MASK_1111」を使用していますので、センサ 4 つ全ての反応を見ます。センサ値と「0x07」を比較した結果、一致していたら右ハーフラインと判断して ret 変数に“1”を代入します。



9.14 左ハーフライン検出処理：check_leftline 関数

左レーンチェンジの手前に 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です。戻り値は、左ハーフラインと判断すると“1”、左ハーフラインでなければ“0”とします。

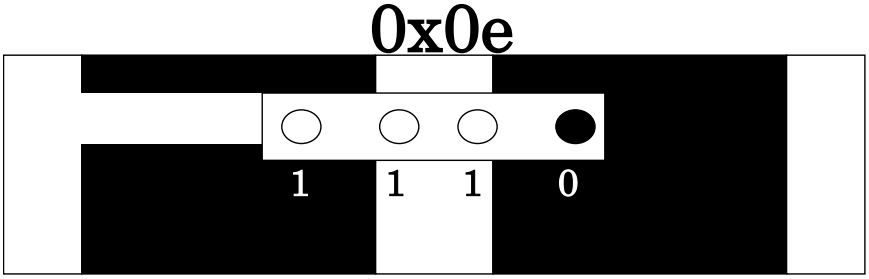
```
672 /*********************************************************************/
673 /* 左ハーフライン検出処理                                            */
674 /* 戻り値 0:なし 1:あり                                              */
675 /*********************************************************************/
676 int check_leftline( void )
677 {
678     int ret;
679
680     ret = 0;
681     if( sensor_get( MASK_1111 ) == 0x0e ) {
682         ret = 1;
683     }
684     return ret;
685 }
```

```
680     ret = 0;
```

680 行目では戻り値を保存する ret 変数の初期化をしています。左ハーフラインと判断すると“1”、違うと判断すると“0”を代入します。今のところどちらか分からないので、とりあえず違うと判断して“0”を入れておきます。

```
681     if( sensor_get( MASK_1111 ) == 0x0e ) {
682         ret = 1;
683     }
```

681 行目の sensor_get 関数でセンサ値を読み込みます。マスク値は「MASK_1111」を使用していますので、センサ 4 つ全ての反応を見ます。センサ値と「0x0e」を比較した結果、一致していたら左ハーフラインと判断して ret 変数に“1”を代入します。



9.15 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。ディップスイッチが ON のとき“1”、OFF のとき“0”を返します。

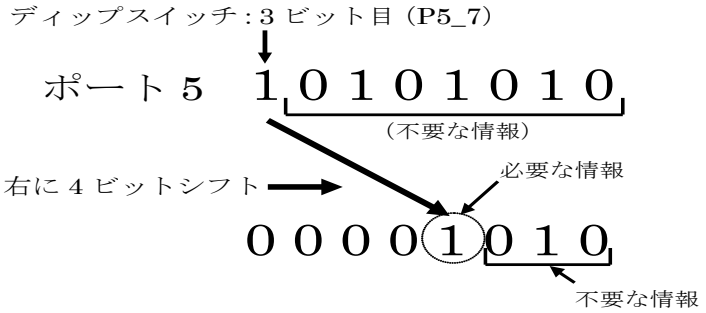
```
712 /*************************************************************************/
713 /* ディップスイッチ値読み込み                                            */
714 /* 戻り値 スイッチ値 0～15                                               */
715 /*************************************************************************/
716 unsigned char dipsw_get( void )
717 {
718     unsigned char sw, sw1, sw2;
719
720     sw1 = (p5>>4) & 0x08;               /* ディップスイッチ読み込み3     */
721     sw2 = (p4>>3) & 0x07;               /* ディップスイッチ読み込み2,1,0*/
722     sw  = ( sw1 | sw2 );                /* P5とP4の値を合わせる          */
723
724     return  sw;
725 }
```

CPU ボードのディップスイッチは、P4_3～P4_5、P5_7 ビットに接続されています。1 つのポートから 4 ビット分読み込めれば良いのですが、今回はディップスイッチの 0～2 ビット目はポート 4、3 ビット目はポート 5 とポートが別々になっています。

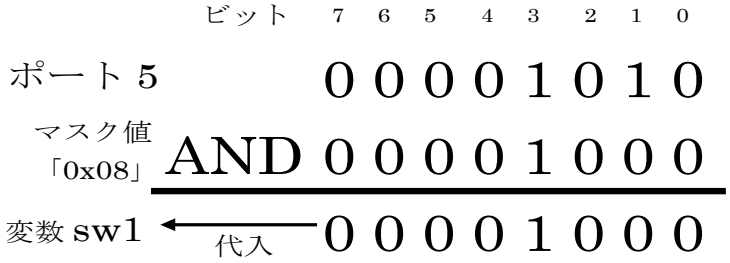
ポート 5、ポート 4 でディップスイッチの接続されていないビットの値は不定です。

```
720     sw1 = (p5>>4) & 0x08;               /* ディップスイッチ読み込み3     */
```

720 行目では、ポート 5 の7ビット目に接続されているディップスイッチ(3 ビット目のスイッチ)の値を取り出します。



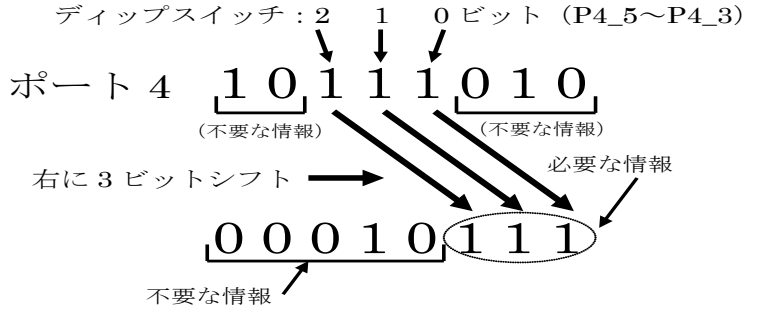
右に 4 ビットシフトした値には、まだ不要な情報が入っていますので、不要なビットを全て“0”にします。不要なビットをマスク処理で強制的に“0”にします。



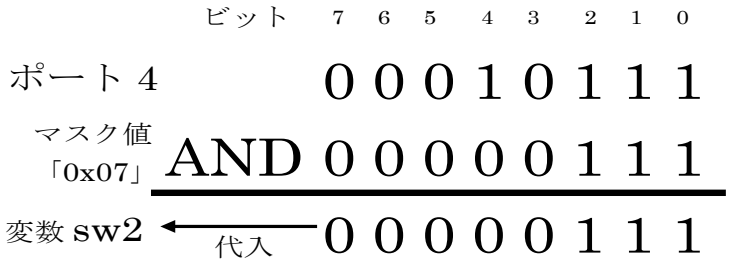
必要な情報は、ポート 5 の 3 ビット目にありますので、マスク値は「0x08」にします。結果、ビット 3 以外のビットは“0”になり、変数 sw1 に代入されます。

```
721     sw2 = (p4>>3) & 0x07;                   /* ディップスイッチ読み込み2,1,0*/
```

721 行目では、ポート 4 の 3～5 ビットに接続されているディップスイッチ(0～2 ビット目のスイッチ)の値を取り出します。



右に3ビットシフトした値には、まだ不要な情報が入っていますので、不要なビットを全て“0”にします。不要なビットをマスク処理で“0”にします。



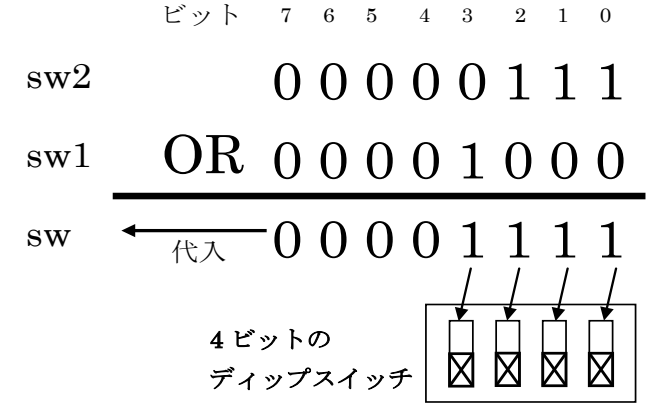
必要な情報は、ポート 4 の 2～0 ビット目にありますので、マスク値は「0x07」にします。結果、2～0 ビット以外のビットは“0”になり、変数 sw2 に代入されます。

```
722     sw  = ( sw1 | sw2 );                   /* P5とP4の値を合わせる          */
```

722 行目では、ディップスイッチの 3 ビット目の値を取り出した変数 sw1 とディップスイッチの 2～0 ビットの値を取り出した変数 sw2 を論理演算の OR で合わせます。

OR 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)、A と B どちらか 1 つでも“1”のとき“1”になる演算を言います。下表に OR の真理値表を示します。

A	B	A OR B
0	0	0
1	0	1
0	1	1
1	1	1

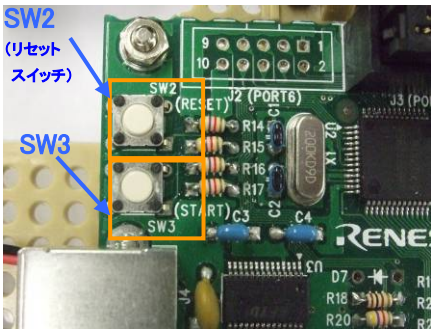


変数sw1 と変数 sw2 を合わせた結果を変数swへ代入します。下位 4 ビットが 4 ビットのディップスイッチに対応します。

9.16 プッシュスイッチの読み込み：pushsw_get 関数

マイコンボードにはプッシュスイッチが 2 個あります。機能を下記に示します。

- SW2：マイコンのリセットスイッチです。マイコンのポートには接続されていません。
- SW3：マイコンの P2_0 に接続されています。



pushsw_get 関数は、プッシュスイッチ SW3 の値を読み込む関数です。

```
727 /************************************************************************/
728 /* プッシュスイッチ値読み込み                                           */
729 /* 戻り値 プッシュスイッチの値 0:OFF 1:ON                               */
730 /************************************************************************/
731 unsigned char pushsw_get( void )
732 {
733     unsigned char sw;
734
735     sw = ~p2;                           /* プッシュスイッチ読み込み     */
736     sw &= 0x01;                         /* 不要ビットを"0"にする        */
737
738     return sw;
739 }
```

プッシュスイッチは ON のとき“0”(0V)、OFF のとき“1”(5V)が P2_0 に入力されます。

```
735     sw = ~p2;                           /* プッシュスイッチ読み込み     */
```

プログラムでは、ON のとき“1”、OFF のとき“0”とした方が分かりやすいため、735 行目では P2 レジスタの値を「~」(NOT)で反転させます。

```
736     sw &= 0x01;                         /* 不要ビットを"0"にする        */
```

P2_0 以外は不要なビットです。何の値が入っている分からないため、「0x01」でマスクします。0x01 は「0000 0001」なので bit0 のみ有効になります。他のビットは強制的に“0”にします。

9.17 センサ LED の制御：sensorled_out 関数

4 輪ミニマイコンカーのセンサ部にある LED（D1～D4）4 個を点灯させます。センサ部にある LED は、3 本の線で 4 個の LED を点灯／消灯させています。どのように制御しているか説明します。

センサ基板の LED は P0_7～P0_5 の信号レベルによって点灯／消灯を制御しています。
P0_7(LEDC)、P0_6(LEDB)、P0_5(LEDA)の信号レベルと D1～D4 の関係を、下表に示します。

P0_7 LEDC	P0_6 LEDB	P0_5 LEDA	D1	D2	D3	D4
0V(″0″)	0V(″0″)	0V(″0″)	消灯	消灯	消灯	消灯
0V(″0″)	0V(″0″)	**5V(″1″)**	消灯	消灯	**点灯**	消灯
0V(″0″)	**5V(″1″)**	0V(″0″)	**点灯**	消灯	消灯	消灯
0V(″0″)	**5V(″1″)**	**5V(″1″)**	**点灯**	消灯	**点灯**	消灯
5V(″1″)	**0V(″0″)**	**0V(″0″)**	消灯	**点灯**	消灯	**点灯**
5V(″1″)	**0V(″0″)**	5V(″1″)	消灯	**点灯**	消灯	消灯
5V(″1″)	5V(″1″)	**0V(″0″)**	消灯	消灯	消灯	**点灯**
5V(″1″)	5V(″1″)	5V(″1″)	消灯	消灯	消灯	消灯

P0_7 が″0″のときは、D1 と D3 を点灯させることができます。P0_7 が″1″のときは、D2 と D4 を点灯させることができます。プログラムでは、下記のように制御して 4 個の LED を点灯させます。

①	P0_7、P0_6、P0_5 を″0″にして、すべての LED を消灯します。
②	D1 を点灯させたければ P0_6 を″1″にします。D3 を点灯させたければ P0_5 を″1″にします。
③	1ms 間待ちます。
④	P0_7、P0_6、P0_5 を″1″にして、すべての LED を消灯します。
⑤	D2 を点灯させたければ P0_6 を″0″にします。D4 を点灯させたければ P0_5 を″0″にします。
⑥	1ms 間待ちます。

これを、割り込みプログラムで実行、繰り返し続けて D1～D4 を点灯／消灯処理を行います。

変数 sensorled_outdata をグローバル変数で宣言します。

```
102 /*======================================*/
103 /* グローバル変数                       */
104 /*======================================*/
 :
途中省略
 :
108 /* sensorled_out関数用変数 */
109 unsigned char   sensorled_outdata;      /* 出力データ保存用変数          */
110 int             s_led_out_on;           /* 関数のON/OFF切り替え          */
```

変数 sensorled_outdata は、センサ部の LED へ出力する値を保存するだけの変数です。LED 出力は、割り込みプログラムで行っています。割り込みプログラムから sensorled_outdata を読み込んでポート 0 を制御しています。

変数 s_led_out_on は、sensorled_out 関数の値を LED に出力するか、sensor_get 関数のセンサ値を出力するかを切り替える変数です。

sensorled_out 関数を下記に示します。

```
/************************************************************************/
/* センサ部のLED出力                                                    */
/* 引数 LEDへの出力値 0～15                                             */
/************************************************************************/
void sensorled_out( unsigned char led )
{
    /* この関数では、LED出力値を保存するだけ */
    sensorled_outdata = led;
}
```

この関数では、LED の制御をしていません。実際に LED の制御をしているのは割り込み関数の中です。ここでは、グローバル変数 sensorled_outdata に LED への出力値 0～15 を保存しています。

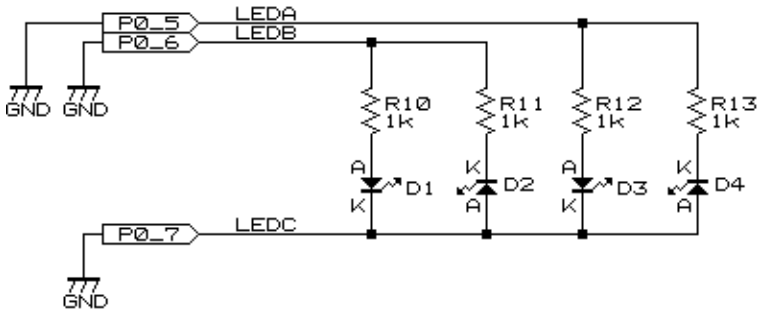
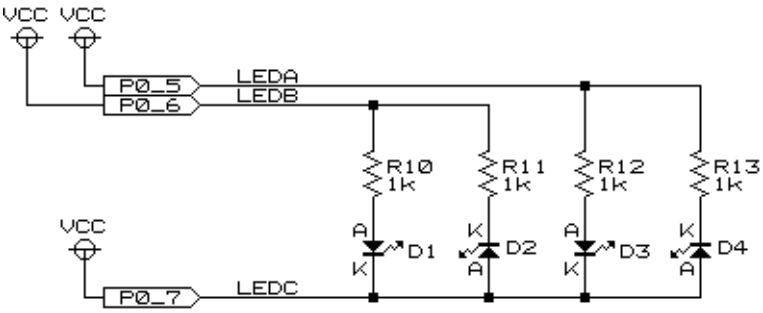
割り込み関数を下記に示します。

```
/************************************************************************/
/* タイマRB 割り込み処理                                                */
/************************************************************************/
#pragma interrupt intTRB(vect=24)
void intTRB( void )
{
    unsigned char   sensorled_data;

    /*タイマカウント用変数              */
    cnt0++;                           /* timer関数用変数              */
    cnt1++;                           /* main関数用変数               */

    if( !s_led_out_on ){
        /* センサ値  読み込み                */
        sensorled_data = sensor_get( MASK_1111 );
    }else{
        /* sensorled_out関数のデータ読み込み  */
        sensorled_data = sensorled_outdata;
    }

    /* 実際にセンサ部のLEDへ出力する      */
    if( p0 & 0x80 ) {
        /* P0_7が"1"なら                  */
        p0 &= 0x1f;                       /* P0_7～P0_5を"0"にして全消灯  */
        if( sensorled_data & 0x8 ) p0 |= 0x40;      /* D1点灯           */
        if( sensorled_data & 0x2 ) p0 |= 0x20;      /* D3点灯           */
    } else {
        /* P0_7が"0"なら                  */
        p0 |= 0xe0;                       /* P0_7～P0_5を"1"にして全消灯  */
        if( sensorled_data & 0x4 ) p0 &= 0xbf;      /* D2点灯           */
        if( sensorled_data & 0x1 ) p0 &= 0xdf;      /* D4点灯           */
    }

}
```

607行目～613行目の間で、sensorled_out 関数の値とsensor_get 関数のセンサ値を出力するか判断しています。グローバル変数 s_led_out_on が“1”のときは、sensorled_out 関数の値を出力します。グローバル変数 s_led_out_on が“0”のときは、sensor_get 関数のセンサ値を出力します。

※sensorled_out 関数を使用する場合は、グローバル変数 s_led_out_on の値を“1”にしてください。sensorled_out 関数の使用が終わったら、変数 s_led_out_on の値を“0”に戻してください。

割り込み関数は、1ms ごとに実行されます。この関数では 616 行目～626 行目までの間でグローバル変数 sensorled_outdata の値を読み込んで、センサ部の LED へ値を出力しています。

616 行	現在、P0_7 に出力している信号レベルが"1"なら 618～620 行を、"0"なら 623～625 行を実行します。
618 行 ～ 620 行	618 行で P0_7、P0_6、P0_5 を"0"にして、すべての LED を消灯させます（下図）。  D1 を点灯させたければ P0_6 を"1"に、D3 を点灯させたければ P0_5 を"1"にします。消灯のままであれば、何もしません。 P0_7="0"なので、D2 と D4 は P0_6、P0_5 が"0"、"1"のどちらでも消灯のままです。 この状態を次の割り込みがかかるまで、すなわち 1ms 後まで保持します。
623 行 ～ 625 行	623 行で P0_7、P0_6、P0_5 を"1"にして、すべての LED を消灯させます（下図）。  D2 を点灯させたければ P0_6 を"0"に、D4 を点灯させたければ P0_5 を"0"にします。消灯のままであれば、何もしません。 P0_7="1"なので、D1 と D3 は P0_6、P0_5 が"0"、"1"のどちらでも消灯のままです。 この状態を次の割り込みがかかるまで、すなわち 1ms 後まで保持します。

9.18 マイコンボード上の LED 制御：led_out 関数

マイコンボード上の LED の点灯／消灯を制御する関数です。
（※マイコンボード上の LED はオプションです。）

```
741 /*************************************************************************/
742 /* マイコン部のLED出力(LEDを実装している場合、使用可)                    */
743 /* 引数 LEDへの出力値 0～15                                              */
744 /*************************************************************************/
745 void led_out( unsigned char led )
746 {
747     unsigned char data;
748
749     led = ~led;
750     led &= 0x0f;
751     data = p1 & 0xf0;
752     p1 = data | led;
753 }
```

```
749     led = ~led;
```

led_out 関数の引数には、プログラムを作成する上で分かりやすいように、LED の点灯させたいビットは“1”、消灯させたいビットは“0”にして引数部分に値を入れます。
実際の LED は、“0”(0V)で点灯、“1”(5V)で消灯になります。そのため 749 行目では、「~」(NOT)で反転させます。

```
750     led &= 0x0f;
```

750 行目では、変数 led の上位 4 ビットは不要なため、「0x0f」でマスクをして値を強制的に“0”にします。

```
751     data = p1 & 0xf0;
```

P1 の上位ビットを他の何かで使用している場合、値を変えてしまうと大変困ります。そのため 751 行目では、上位 4 ビットに入っている値を「0xf0」でマスクします。そうすると、P1 の上位 4 ビットの値だけを取り出すことができます。その値を変数 data に代入します。

```
752     p1 = data | led;
```

752 行目は、変数 led の下位 4 ビットと変数 data の上位 4 ビットを論理演算の OR で合わせて、P1レジスタに書き込みます。

9.19 モータ速度制御：motor 関数

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

9.19.1 左モータの動作

左モータは、P2_1、P2_2、P2_6 の 3 端子で制御します。

モータ左 1 P2_1	モータ左 2 P2_6	モータ左 PWM P2_2	モータ動作
1	1	x	ブレーキ
0	0	x	フリー
0	1	PWM	PWM=″1″なら正転、″0″ならブレーキ
1	0	PWM	PWM=″1″なら逆転、″0″ならブレーキ

左モータを正転させたい場合、P2_1=″0″、P2_6=″1″にして、P2_2 から PWM 波形を出力すると、左モータが PWM の割合に応じて正転します。例えば、PWM が 0%ならモータの回転は停止、PWM が 50%ならモータの回転は正転 50%、PWM100%ならモータの回転は正転 100%になります。
左モータを逆転させたい場合、P2_1=″1″、P2_6=″0″にして、P2_2 から PWM 波形を出力すると、左モータが PWM の割合に応じて逆転します。例えば、PWM が 0%ならモータの回転は停止、PWM が 50%ならモータの回転は逆転 50%、PWM100%ならモータの回転は逆転 100%になります。

9.19.2 右モータの動作

右モータは、P2_3、P2_4、P2_7 の 3 端子で制御します。

モータ右 1 P2_3	モータ右 2 P2_7	モータ右 PWM P2_4	モータ動作
1	1	x	ブレーキ
0	0	x	フリー
0	1	PWM	PWM=″1″なら正転、″0″ならブレーキ
1	0	PWM	PWM=″1″なら逆転、″0″ならブレーキ

右モータを正転させたい場合、P2_3=″0″、P2_7=″1″にして、P2_4 から PWM 波形を出力すると、右モータが PWM の割合に応じて正転します。例えば、PWM が 0%ならモータの回転は停止、PWM が 50%ならモータの回転は正転 50%、PWM100%ならモータの回転は正転 100%になります。
右モータを逆転させたい場合、P2_3=″1″、P2_7=″0″にして、P2_4 から PWM 波形を出力すると、右モータが PWM の割合に応じて逆転します。例えば、PWM が 0%ならモータの回転は停止、PWM が 50%ならモータの回転は逆転 50%、PWM100%ならモータの回転は逆転 100%になります。

左モータ、右モータへ PWM を出力する関数です。

```
755 /************************************************************************/
756 /* モータ速度制御                                                       */
757 /* 引数    左モータ:-100～100、右モータ:-100～100                       */
758 /*         0で停止、100で正転100%、-100で逆転100%                       */
759 /* 戻り値 なし                                                          */
760 /************************************************************************/
761 void motor( int data1, int data2 )
762 {
763     int    motor_r, motor_l, sw_data;
764
765     sw_data = dipsw_get() + 5;
766     motor_l = data1 * sw_data / 20;
767     motor_r = data2 * sw_data / 20;
768
769     /* 左モータ制御 */
770     if( motor_l >= 0 ) {
771         p2 &= 0xfd;
772         p2 |= 0x40;
773         trdgrd0 = (long)( PWM_CYCLE - 1 ) * motor_l / 100;
774     } else {
775         p2 |= 0x02;
776         p2 &= 0xbf;
777         trdgrd0 = (long)( PWM_CYCLE - 1 ) * ( -motor_l ) / 100;
778     }
779
780     /* 右モータ制御 */
781     if( motor_r >= 0 ) {
782         p2 &= 0xf7;
783         p2 |= 0x80;
784         trdgrc1 = (long)( PWM_CYCLE - 1 ) * motor_r / 100;
785     } else {
786         p2 |= 0x08;
787         p2 &= 0x7f;
788         trdgrc1 = (long)( PWM_CYCLE - 1 ) * ( -motor_r ) / 100;
789     }
790 }
```

(1) motor 関数の使い方

motor 関数の使い方を下記に示します。

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

引数は、左モータの PWM 値と右モータの PWM 値をカンマで区切って入れます。値とモータの回転の関係を下記に示します。

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

実際にモータに出力される割合を、下記に示します。

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

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

例えば、motor 関数で左モータに 80 を設定した場合、正転 80%にするかというと実はそうではありません。マイコンボード上にあるディップスイッチの値により、実際にモータに出力される PWM の割合が変化します。
ディップスイッチが、“1100”(10 進数で 12)のとき、下記プログラムを実行したとします。

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

実際のモータに出力される PWM 値は、下記のようになります。

左モータに出力される PWM ＝ -70×(12＋5)÷20 ＝ -70×0.85 ＝ -59.5 ＝ -59%
右モータに出力される PWM ＝ 100×(12＋5)÷20 ＝ 100×0.85 ＝ 85%

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

これから、上記の内容がどのように実行されるのか説明します。

(2) ディップスイッチの割合に応じて、PWM 値を変更

```
765     sw_data = dipsw_get() + 5;          dipsw_get() = ディップスイッチの値0～15
766     motor_l = data1 * sw_data / 20;
767     motor_r = data2 * sw_data / 20;
```

765 行	sw_data 変数にディップスイッチの値＋5の値を代入します。ディップスイッチの値は 0～15 なので、sw_data 変数の値は、5～20 になります。
766 行	motor_l 変数は、左モータに加える PWM 値の割合を代入する変数です。data1 が motor 関数に設定した左モータの PWM 値です。 よって、次の計算を行って motor_l 変数に PWM 値を設定します。 motor_l＝data1(motor 関数で設定した左モータの PWM)×sw_data／20 motor_l 変数の範囲は、-100～100 の値です。
767 行	motor_r 変数は、右モータに加える PWM 値の割合を代入する変数です。data2 が motor 関数に設定した右モータの PWM 値です。 よって、次の計算を行って motor_r 変数に PWM 値を設定します。 motor_r＝data2(motor 関数で設定した右モータの PWM)×sw_data／20 motor_r 変数の範囲は、-100～100 の値です。

9.20 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、サーボの回転角度を設定できます。

```
792 /************************************************************************/
793 /* サーボハンドル操作                                                   */
794 /* 引数    サーボ角度：-90～90                                          */
795 /*         -90で左へ90度、0でまっすぐ、90で右へ90度                      */
796 /************************************************************************/
797 void handle( int angle )
798 {
799     /* サーボが左右逆に動く場合は、「-」を「+」に替えてください */
800     trdgrd1 = SERVO_CENTER - angle * HANDLE_STEP;
801 }
```

(1) handle 関数の使い方

handle 関数の使い方を下記に示します。

```
handle( サーボの角度 );
```

引数は、サーボの角度を設定します。値とサーボの角度の関係を下記に示します。

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

プログラム例を、下記に示します。

```
ハンドルの角度が 0 度 の場合は、handle( 0 );
ハンドルの角度が 右 30 度 の場合は、handle( 30 );
ハンドルの角度が 左 45 度 の場合は、handle( -45 );
```

となります。

(2) プログラムの内容

```
800     trdgrd1 = SERVO_CENTER - angle * HANDLE_STEP;
          ①           ②          ③          ④
```

①	サーボに接続されている P2_5 端子の PWM の ON 幅を設定するのは、TRDGRB1 です。ただ、直接 TRDGRB1 の値を書き換えるとそのタイミングによっては、PWM 波形が 100%出力になってしまうことがあるので、バッファレジスタを使います。TRDGRB1 のバッファレジスタは TRDGRD1 なので、今回は、このレジスタに PWM 値を設定します。
②	0 度のときの値です。
③	handle 関数で指定した角度が代入されている変数です。
④	1 度当たりの増分です。

TRDGRD1 に代入される値の計算例を下記に示します。
※SERVO_CENTER=3749、HANDLE_STEP=22 とします。

• 0 度のとき

```
TRDGRD1 = SERVO_CENTER - angle * HANDLE_STEP
        = 3749         - 0     * 22
        = 3749
```

• 30 度のとき

```
TRDGRD1 = SERVO_CENTER - angle * HANDLE_STEP
        = 3749         - 30    * 22
        = 3749         - 660
        = 3089
```

• -45 度のとき

```
TRDGRD1 = SERVO_CENTER - angle * HANDLE_STEP
        = 3749         - (-45) * 22
        = 3749         - (-990)
        = 4739
```

9.21 ブザーを鳴らす：beep 関数

ブザーを鳴らす関数です。

```
803 /************************************************************************/
804 /* ブザーを鳴らす                                                       */
805 /* 引数  音階のPWM値                                                    */
806 /************************************************************************/
807 void beep( unsigned int tone )
808 {
809     if( tone ) {
810         trcmr  &= 0x7f;                 /* TRCのカウント停止            */
811         trc     = tone - 1;
812         trcgra  = tone - 1;             /* 周期設定                     */
813         trcgrc  = tone / 2 - 1;         /* ON幅設定                     */
814         trcmr  |= 0x80;                 /* TRCのカウント開始            */
815     } else {
816         trcmr  &= 0x7f;                 /* TRCのカウント停止            */
817         trc     = 0;
818         trcgra  = 1;                    /* 周期設定                     */
819         trcgrc  = 1;                    /* ON幅設定                     */
820         trcmr  |= 0x80;                 /* TRCのカウント開始            */
821     }
822 }
```

809 行	引数は、タイマ RC ジェネラルレジスタ A(TRCGRA)に設定する周期のデータです。 このデータが 0 でないならブザーを鳴らすために 810 行へ、ブザーを止めるなら 816 行へジャンプします。
810 行	タイマ RC モードレジスタ(TRCMR)の bit7 を″0″にして、TRC のカウント動作を停止します。PWM 波形出力が止まるので、ブザーが鳴りやみます。
811 行～ 813 行	タイマ RC カウンタ(TRC)にカウンタの初期値、タイマ RC ジェネラルレジスタ A(TRCGRA)に周期、タイマ RC ジェネラルレジスタ C(TRCGRC)に ON 幅を設定します。
814 行	タイマ RC モードレジスタ(TRCMR)の bit7 を″1″にして、TRC のカウント動作を開始します。P3_4 端子から PWM 波形が出力され、ブザーが鳴ります。
816 行	タイマ RC モードレジスタ(TRCMR)の bit7 を″0″にして、TRC のカウント動作を停止します。PWM 波形出力が止まるので、ブザーが鳴りやみます。ただし、PWM 端子は″0″で止まったか、″1″で止まったかは分かりません。
817 行～ 819 行	タイマ RC カウンタ(TRC)に 0、タイマ RC ジェネラルレジスタ A(TRCGRA)に 1、タイマ RC ジェネラルレジスタ C(TRCGRC)に 1 を設定します。
820 行	タイマ RC モードレジスタ(TRCMR)の bit7 を″1″にして、TRC のカウント動作を開始します。P3_4 端子は″0″になり、ブザーは鳴りやみます。

(1) beep 関数の使い方

beep 関数の使い方を下記に示します。

```
beep( 鳴らしたい音の PWM 値 );
```

引数は、鳴らしたい音の PWM 値を設定します。PWM 値を計算で求めるのは大変なため、下記のように PWM 値

```
042 /* 音階 ブザー用          */
043 /* 3オクターブ目の音階     */
044 #define         DO_3        38226       /* ド                  */
045 #define         DOU_3       36081       /* ド＃                */
046 #define         RE_3        34056       /* レ                  */
047 #define         REU_3       32144       /* レ＃                */
048 #define         MI_3        30340       /* ミ                  */
049 #define         FA_3        28637       /* ファ                */
050 #define         FAU_3       27030       /* ファ＃              */
051 #define         SO_3        25513       /* ソ                  */
052 #define         SOU_3       24081       /* ソ＃                */
053 #define         RA_3        22729       /* ラ                  */
054 #define         RAU_3       21454       /* ラ＃                */
055 #define         SI_3        20250       /* シ                  */
056
  :
  :
省略
  :
084
```

プログラム例を、下記に示します。

3 オクターブ目のドの音 の場合は、beep(DO_3);
3 オクターブ目のレの音 の場合は、beep(RE_3);
3 オクターブ目のミの音 の場合は、beep(MI_3);

となります。

9.22 メインプログラム

9.22.1 スタート

メイン関数です。スタートアッププログラムから呼ばれて実行される最初のプログラムです。

```
114 /************************************************************************/
115 /* メインプログラム                                                     */
116 /************************************************************************/
117 void main( void )
118 {
119     init();                             /* 初期化                       */
120     asm("fset I");                      /* 全体割り込み許可             */
121
122     /* マイコンカーの状態初期化 */
123     handle( 0 );
124     motor( 0, 0 );
```

```
119     init();                             /* 初期化                       */
```

init 関数を実行し、R8C/35A の周辺機能の初期化をします。

```
120     asm("fset I");                      /* 全体割り込み許可             */
```

CPU 全体の割り込みを許可します。

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

次に、マイコンカーの状態を初期化します。
・サーボモータは 0 度
・スピードは、左 0%、右 0%
にしています。

9.22.2 パターン方式

サンプルプログラムでは、1つのパターンを1つのサブルーチンとしてプログラミングをしています。仕組みは、あらかじめプログラムを細かく分けて作成します。(例えば、「待機モード」、「センサボリューム調整モード」など)

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。

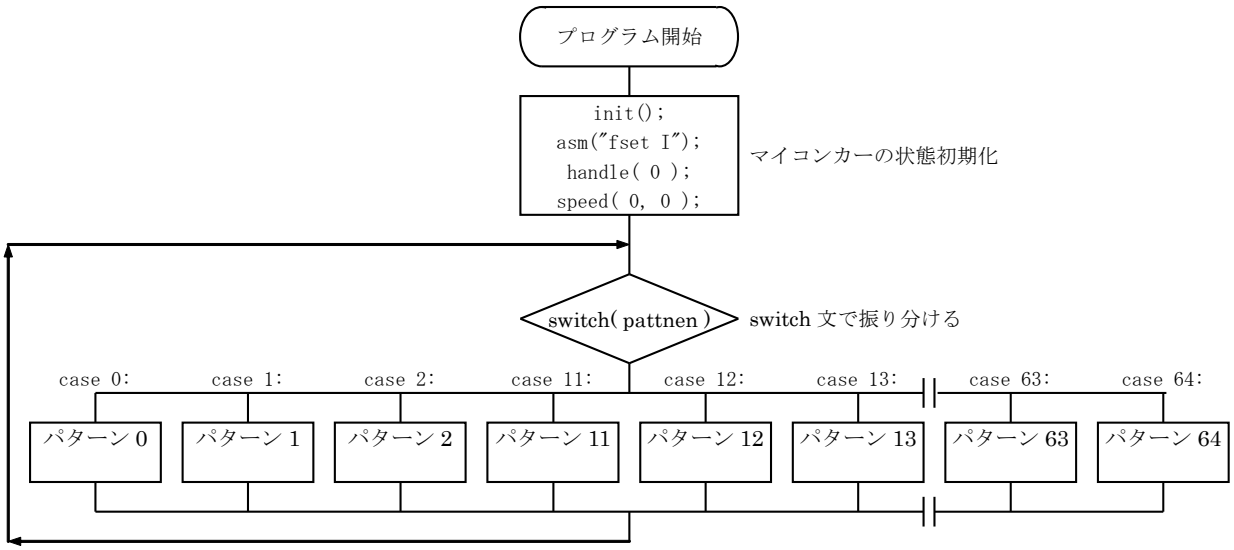
例えば、pattern 変数が 0 のときは「待機モード」、pattern 変数が 1 のきは「センサボリューム調整モード」などです。

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。ここでは、この方式をパターン方式と定めています。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。

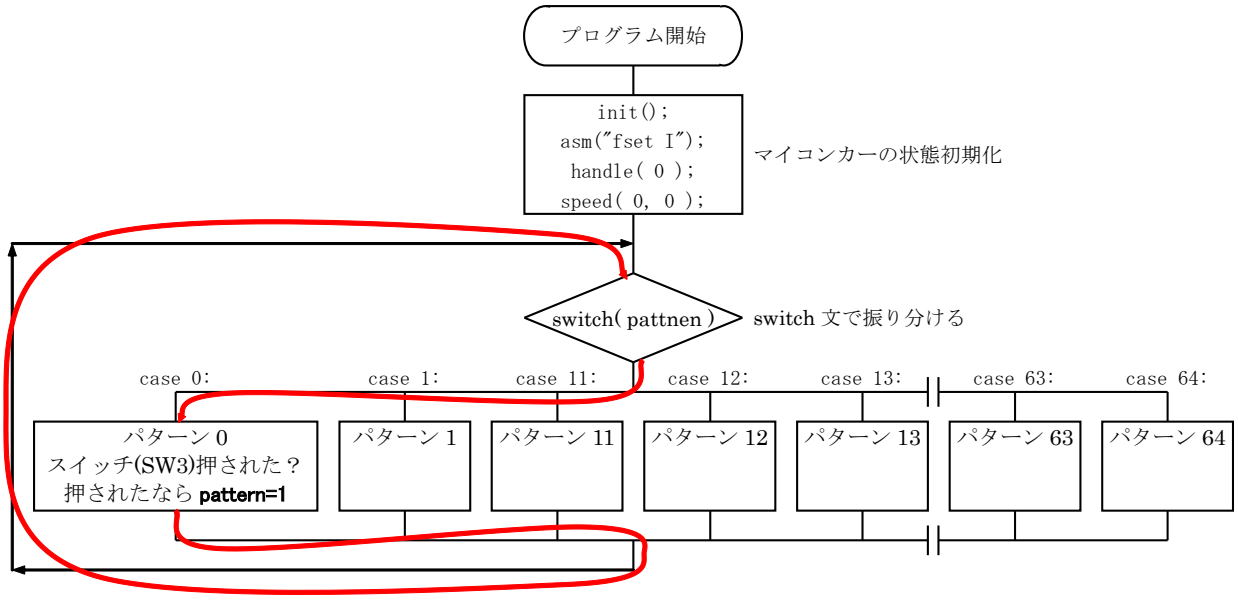
各パターンについては、**「9.22.4 パターンの内容」**に記述しています。

9.22.3 プログラムの作り方

パターン方式をC言語で各パターンに切り替えるには、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。

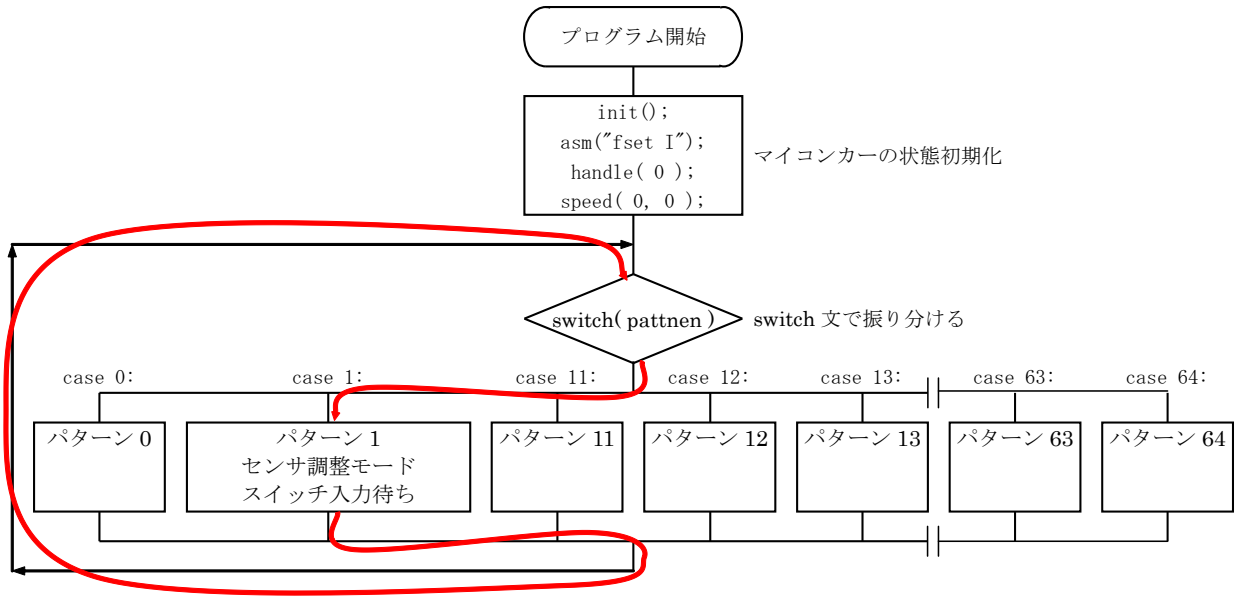


起動時、pattern 変数は 0 です。switch 文により case 0 部分のプログラム「パターン 0 プログラム」を実行し続けます。後ほど説明しますが、パターン 0 は待機モード(リセット直後の動作)です。スイッチ(SW3)が押されると、「pattern=1」が実行されます。下図のようになります。



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、センサボリューム調整モード部分です。下図のようになります。



このように、プログラムをブロック化します。ブロック化したプログラムでは、「スイッチが押されたか」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。

9.22.4 パターンの内容

サンプルプログラム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。

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、2、11、12…と値が飛び飛びになっています。これは、備考のように 0 番台を走行前の処理、10 番台を通常走行処理というように、10 ごとに大まかな処理内容を決めて分類しているためです。

パターン	処理内容	備考
0	待機モード(リセット直後の動作)	パターン 0～10 は、走行前の処理
1	センサボリューム調整モード	
2	カウントダウンスタート	
11	通常トレース	パターン 11～20 は、通常走行中の処理
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	パターン 21～30 は、クロスラインを見つけてから直角までの処理
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	右クランククリア処理 安定するまで少し待つ	パターン 31～40 は、右クランク処理
32	右クランククリア処理 曲げ終わりのチェック	
41	左クランククリア処理 安定するまで少し待つ	パターン 41～50 は、左クランク処理
42	左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	パターン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
52	2本目を読み飛ばす	
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	パターン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
62	2本目を読み飛ばす	
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	

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、理解しやすくなります。

現在のパターン	状態	パターンが変わる条件
0	待機モード(リセット直後の動作)	・スイッチ(SW3)を押したらパターン 1 へ
1	**センサボリューム調整モード**	・スイッチ(SW3)を押したらパターン 2 へ
2	**カウントダウンスタート**	・4.2 秒たったらパターン 11 へ
11	**通常トレース**	・右大曲げになったらパターン 12 へ ・左大曲げになったらパターン 13 へ ・クロスラインを検出したらパターン 21 へ ・右ハーフラインを検出したらパターン 51 へ ・左ハーフラインを検出したらパターン 61 へ
12	**右へ大曲げの終わりのチェック**	・右大曲げが終わったらパターン 11 へ ・クロスラインを検出したらパターン 21 へ ・右ハーフラインを検出したらパターン 51 へ ・左ハーフラインを検出したらパターン 61 へ
13	**左へ大曲げの終わりのチェック**	・左大曲げが終わったらパターン 11 へ ・クロスラインを検出したらパターン 21 へ ・右ハーフラインを検出したらパターン 51 へ ・左ハーフラインを検出したらパターン 61 へ
21	1本目のクロスライン検出時の処理	・サーボ、スピードの設定を終えたらパターン 22 へ
22	2本目を読み飛ばす	・500ms 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	・右クランクを見つけたらパターン 31 へ ・左クランクを見つけたらパターン 41 へ
31	右クランククリア処理　安定するまで少し待つ	・500ms たったならパターン 32 へ
32	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
41	左クランククリア処理　安定するまで少し待つ	・500ms たったならパターン 42 へ
42	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
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**	・500ms たったならパターン 53 へ
53	**右ハーフライン後のトレース**	・中心線が無くなったならパターン 54 へ
54	**右レーンチェンジ終了のチェック**	・新しい中心線がセンサの中心に来たならパターン 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**	・500ms たったならパターン 63 へ
63	**左ハーフライン後のトレース**	・中心線が無くなったならパターン 64 へ
64	**左レーンチェンジ終了のチェック**	・新しい中心線がセンサの中心に来たならパターン 11 へ

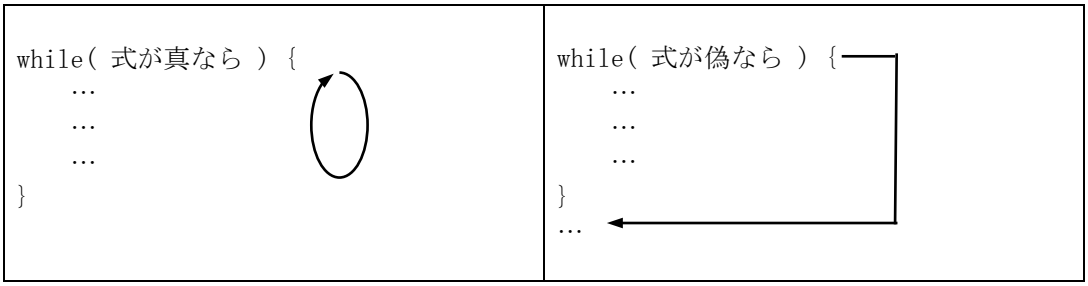
9.22.5 パターン方式の最初 while、switch 部分

```
126     while( 1 ){
127         switch( pattern ){
154             case 0:
                      パターン0時の処理
174                 break;
175
176             case 1:
                      パターン1時の処理
186                 break;

            それぞれのパターン処理

526             default:
527                 /* どれでもない場合は待機状態に戻す          */
528                 pattern = 0;
529                 break;
530         }
531     }
                                                        対      対
```

126 行の「while(1){」と 531 行の「}」が対、127 行の「switch(pattan){」と 530 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。

「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 の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 4 輪ミニマイコンカーを動かすためのプログラムですのでこれで問題ありません。4 輪ミニマイコンカーがゴール(または脱輪)すれば、人間が取り上げて電源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通常の使用方法です。

9.22.6 パターン 0：待機モード（リセット直後の動作）

パターン 0 は、CPU の電源を入れた(リセットボタンを押した)あと、パターン方式の中で一番初めに実行するプログラムです。ここでは、スイッチ(SW3)が押されたかチェックをする部分です。チェック中、本当にマイコンが動作しているのか止まっているのか分からないため、センサ基板についている LED の点滅処理を行います。

```
154             case 0:
155                 /* 待機モード（リセット直後の動作）            */
156                 s_led_out_on = 1;       /* sensorled_out関数使用許可    */
157                 if( pushsw_get() ){
158                     while( pushsw_get() );
159                     beep( SI_3 );       /* 3オクターブ目のシの音をセット*/
160                     timer( 200 );
161                     beep( 0 );          /* ブザーの停止                 */
162                     pattern = 1;
163                     cnt1 = 0;
164                     break;
165                 }
```

157 行目の pushsw_get 関数でスイッチ(SW3)を押されたかチェックします。押されると“1”が返ってくるので、カッコの中が実行され、beep 関数で「3 オクターブ目のシ」の音をセットして 0.2 秒間鳴らします。そして、pattern 変数に“1”を代入します。

ここで if 文のカッコの中に注目します。

```
        if( pushsw_get() == 1 ) {
```

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

しかし、今回のプログラムは、

```
        if( pushsw_get() ) {
```

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

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

となります。

スイッチが押された状態で、pushsw_get 関数の戻り値は“1”になります。そのため、スイッチが押されたら**0 以外の値と判断され**、158～164 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。センサ基板にある 4 つの LED を「0101」と「1010」を 0.1 秒おきに、点滅を繰り返す処理にします。

sensorled_out 関数を使用するため、156 行で変数 s_led_out_on を“1”にします。変数 s_led_out_on が“0”のままだと sensorled_out 関数の引数に値をセットしても、指定した通りに LED が点灯しません。

```
167                 /* センサLED点滅処理                            */
168                 if( cnt1 > 200 ){
169                     sensorled_out( 0x5 );
170                     cnt1 = 0;
171                 }else if( cnt1 > 100 ){
172                     sensorled_out( 0xa );
173                 }
174                 break;
```

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。サンプルプログラムでは、**変数 cnt0 と cnt1 だけは例外です**。cnt0 と cnt1 は割り込み関数「intTRB」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 の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。

```
154             case 0:
155                 /* 待機モード（リセット直後の動作）            */
156                 s_led_out_on = 1;       /* sensorled_out関数使用許可    */
157                 if( pushsw_get() ){
158                     while( pushsw_get() );
159                     beep( SI_3 );       /* 3オクターブ目のシの音をセット*/
160                     timer( 200 );
161                     beep( 0 );          /* ブザーの停止                 */
162                     pattern = 1;
163                     cnt1 = 0;
164                     break;
165                 }
166
167                 /* センサLED点滅処理                          */
168                 if( cnt1 > 200 ){
169                     sensorled_out( 0x5 );
170                     cnt1 = 0;
171                 }else if( cnt1 > 100 ){
172                     sensorled_out( 0xa );
173                 }
174                 break;
```

174 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。

```
154             case 0:
155                 /* 待機モード（リセット直後の動作）            */
156                 s_led_out_on = 1;       /* sensorled_out関数使用許可    */
157                 if( pushsw_get() ){
158                     while( pushsw_get() );
159                     beep( SI_3 );       /* 3オクターブ目のシの音をセット*/
160                     timer( 200 );
161                     beep( 0 );          /* ブザーの停止                 */
162                     pattern = 1;
163                     cnt1 = 0;
164                     break;
165                 }
166
167                 /* センサLED点滅処理                          */
168                 timer(100);
169                 sensorled_out( 0x5 );
170                 timer(100);
171                 sensorled_out( 0xa );
172                 break;
```

シンプルになりました。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てしまい、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**。

9.22.7 パターン 1：センサボリューム調整モード

パターン 1 では、センサのボリュームを回して、赤外 LED の感度を調整するモードです。

パターン方式のプログラムで一番初めに行うプログラムはパターン 0 です。パターン 0 では、sensorled_out 関数を使用しているため、センサ基板の LED が使えません。そのため、センサ調整ができません。そこで、パターン 1 でセンサ調整をできるようにしました。

```
176             case 1:
177                 /* センサボリューム調整モード                    */
178                 /* スイッチ入力待ち                              */
179                 s_led_out_on = 0;       /* sensorled_out関数使用禁止    */
180                 if( pushsw_get() ){
181                     while( pushsw_get() );
182                     pattern = 2;
183                     cnt1 = 0;
184                     break;
185                 }
186                 break;
```

パターン 1 では、sensorled_out 関数を使用できないように、変数 s_led_out_on に“0”を代入します。これで sensor_get 関数（センサ値）の値がセンサ基板の LED に出力されます。これでセンサ調整が行えます。

パターン 0 同様にスイッチが押されたら、pattern 変数に“2”を代入します。

パターンが変わる条件

・スイッチ(SW3)が押されたら、パターン 2 へ

9.22.8 パターン 2：カウントダウンスタート

パターン 2 では、カウトダウンを行います。ここでは、音が「ピ・ピ・ピ・ピーー」となったらパターン 11 へ移ります。

```
188             case 2:
189                 /* カウントダウンスタート                          */
190                 s_led_out_on = 1;       /* sensorled_out関数使用許可     */
191                 sensorled_out( 0x0c );
192                 beep( RA_3 );           /* 3オクターブ目のラの音をセット*/
193                 timer( 500 );
194                 beep( 0 );              /* ブザーの停止                  */
195                 timer( 500 );
196
197                 sensorled_out( 0x0e );
198                 beep( RA_3 );           /* 3オクターブ目のラの音をセット*/
199                 timer( 500 );
200                 beep( 0 );              /* ブザーの停止                  */
201                 timer( 500 );
202
203                 sensorled_out( 0x0f );
204                 beep( RA_3 );           /* 3オクターブ目のラの音をセット*/
205                 timer( 500 );
206                 beep( 0 );              /* ブザーの停止                  */
207                 timer( 500 );
208
209                 sensorled_out( 0x00 );
210                 beep( RA_5 );           /* 5オクターブ目のラの音をセット*/
211                 timer( 1200 );
212                 beep( 0 );              /* ブザーの停止                  */
213
214                 s_led_out_on = 0;       /* sensorled_out関数使用禁止     */
215                 pattern = 11;
216                 cnt1 = 0;
217                 break;
```

190 行～191 行	s_led_out_on 変数の値を“1”にして sensorled_out 関数の使用を許可します。 sensorled_out 関数で LED に「0x0c」(00001100)を出力します。
192 行～195 行 198 行～201 行 204 行～207 行	beep 関数で「3 オクターブ目のラ」の音をセットし、0.5 秒間鳴らします。その後 0.5 秒間止めます。
197 行	sensorled_out 関数で LED に「0x0e」(00001110)を出力します。
203 行	sensorled_out 関数で LED に「0x0f」(00001111)を出力します。
209 行	sensorled_out 関数で LED に「0x00」(00000000)を出力します。
210 行～212 行	beep 関数で「5 オクターブ目のラ」の音をセットし、1.2 秒間鳴らします。 beep 関数の引数の値を 0 にして、ブザーを止めます。
214 行	s_led_out_on 変数の値を“0”にして sensorled_out 関数の使用を禁止します。
215 行	pattern 変数の値を“11”にして、パターン 11 へ移ります。

パターンが変わる条件

・4.2 秒たったら、パターン 11 へ

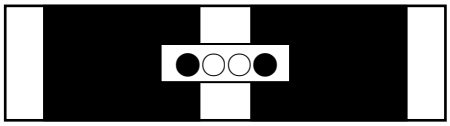
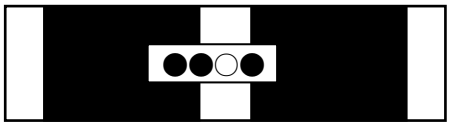
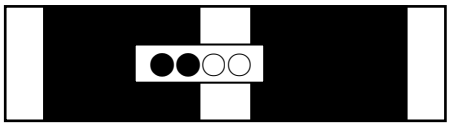
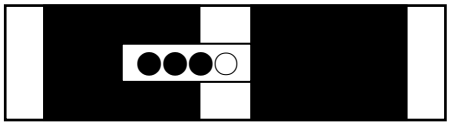
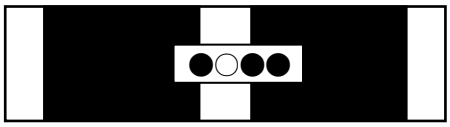
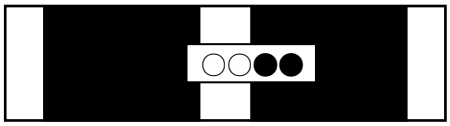
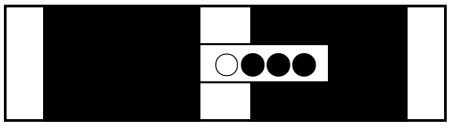
9.22.9 パターン 11：通常トレース

パターン 11 は、コース上をトレースする状態です。

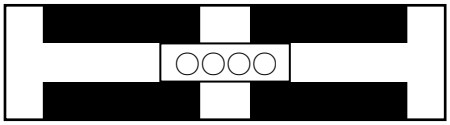
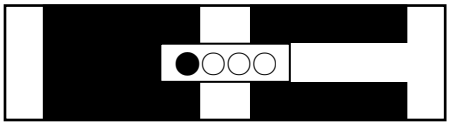
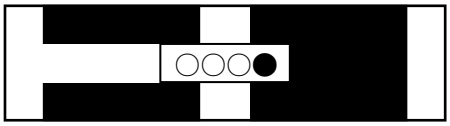
4 輪ミニマイコンカーには、センサが 4 つ付いています。この 4 つのセンサで、想定されるセンサの状態を考えます。

次にそのときのサーボモータの角度と左モータ、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、サーボモータを大きく曲げてスピードを落とします。

下記のようなパターンを考えました。

	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	サーボモータ角度	左モータ PWM	右モータ PWM
1		0110	0x06	0	100	100
2		0010	0x02	10	90	80
3		0011	0x03	15	80	60
4		0001	0x01	20	60	40
5		0100	0x04	−10	80	90
6		1100	0x0c	−15	60	80
7		1000	0x08	−20	40	60

また、マイコンカーラリ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	センサを読み込んだときの値	16進数	チェックする関数
8		1111	0x0f	check_crossline 関数
9		0111	0x07	check_rightline 関数
10		1110	0x0e	check_leftline 関数

(1)センサ読み込み

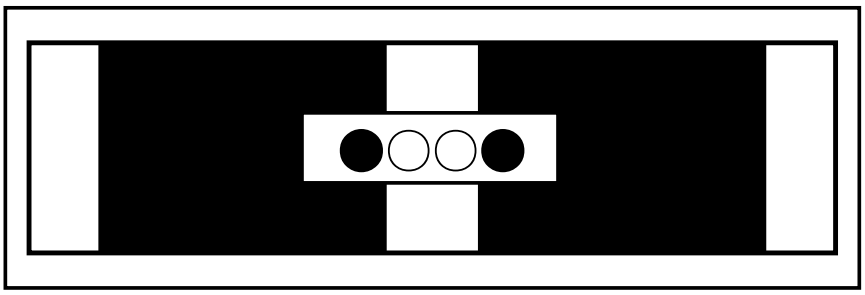
```
234                 switch( (sensor_get( MASK_1111 )) ){
```

センサの状態を読み込みます。センサ 4 個全て読み込みます。

(2)直進

```
235                 case 0x06:
236                     /* センターまっすぐ                    */
237                     handle( 0 );
238                     motor( 100, 100 );
239                     break;
```

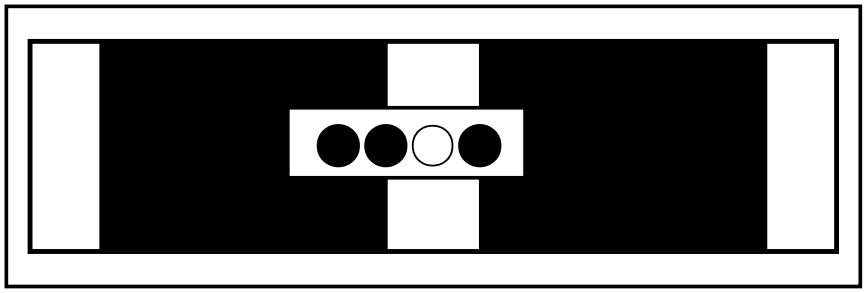
センサが「0x06」の状態です。この状態は下図のように、まっすぐ進んでいる状態です。サーボモータの角度、0 度、左モータ 100%、右モータ 100%で進みます。



(3)少し左寄り

```
241                     case 0x02:
242                         /* 少し左寄り→右へ小曲げ                */
243                         handle( 10 );
244                         motor( 90, 80 );
245                         break;
```

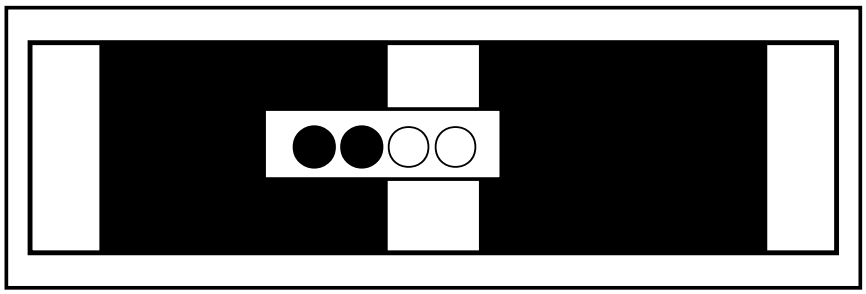
センサが「0x02」の状態です。この状態は下図のように、4 輪ミニマイコンカーが少し左寄りの状態です。サーボモータを右に 10 度、左モータ 90%、右モータ 80%で進み、中心に寄るようにします。



(4)中くらい左寄り

```
247                     case 0x03:
248                         /* 中くらい左寄り→右へ中曲げ            */
249                         handle( 15 );
250                         motor( 80, 60 );
251                         break;
```

センサの状態が「0x03」の状態です。この状態は下図のように、4 輪ミニマイコンカーが中くらい左に寄っている状態です。サーボモータを右に 15 度、左モータ 80%、右モータ 60%で進み、減速しながら中心に寄るようにします。

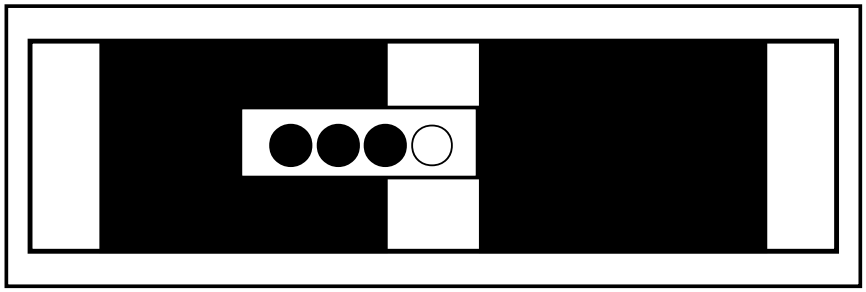


(5)大きく左寄り

```
253                     case 0x01:
254                         /* 大きく左寄り→右へ大曲げ              */
255                         handle( 20 );
256                         motor( 60, 40 );
257                         pattern = 12;
258                         break;
```

※257 行については、後述します。

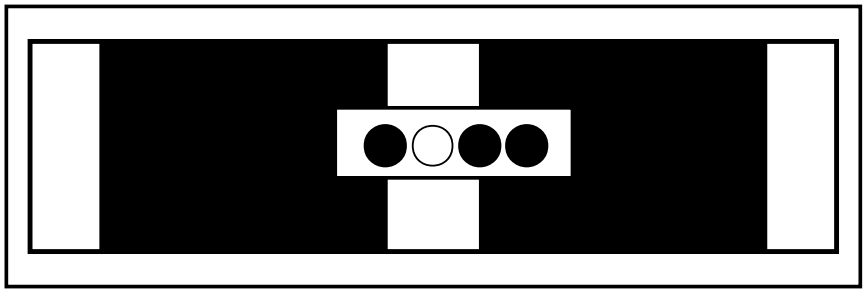
センサが「0x01」の状態です。この状態は下図のように、4 輪ミニマイコンカーが大きく左に寄っている状態です。サーボモータを右に 20 度、左モータ 60%、右モータ 40%で進み、かなり減速しながら中心を寄るようにします。



(6)少し右寄り

```
260                     case 0x04:
261                         /* 少し右寄り→左へ小曲げ              */
262                         handle( -10 );
263                         motor( 80, 90 );
264                         break;
```

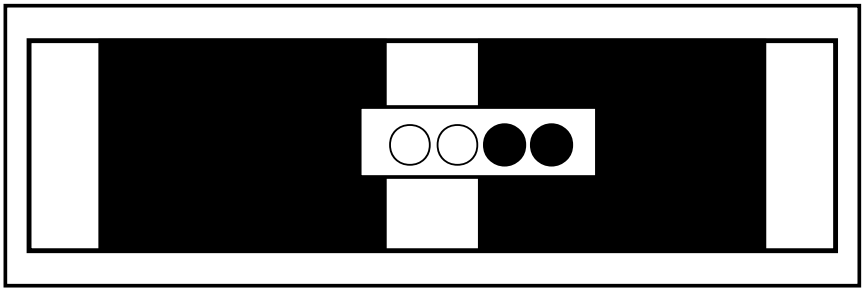
センサが「0x04」の状態です。この状態は下図のように、4 輪ミニマイコンカーが少し右寄りの状態です。サーボモータを左に 10 度、左モータ 80%、右モータ 90%で進み、中心に寄るようにします。



(7) 中くらい右寄り

```
266                     case 0x0c:
267                         /* 中くらい右寄り→左へ中曲げ          */
268                         handle( -15 );
269                         motor( 60, 80 );
270                         break;
```

センサが「0x0c」の状態です。この状態は下図のように、4 輪ミニマイコンカーが少し右に寄っている状態です。サーボモータを左に 15 度、左モータ 60%、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。

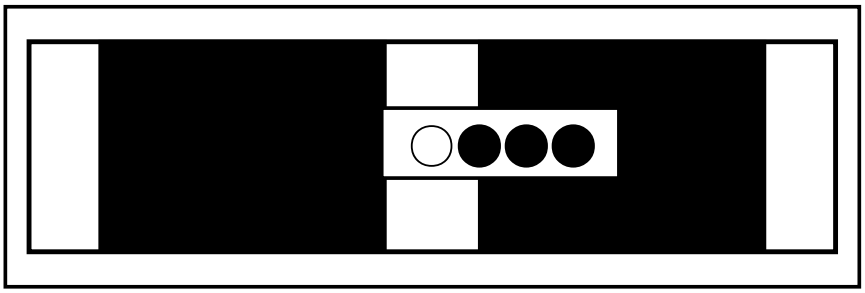


(8) 大きく右寄り

```
272                     case 0x08:
273                         /* 大きく右寄り→左へ大曲げ            */
274                         handle( -20 );
275                         motor( 40, 60 );
276                         pattern = 13;
277                         break;
```

※276 行については、後述します。

センサが「0x08」の状態です。この状態は下図のように、4 輪ミニマイコンカーが大きく右に寄っている状態です。サーボモータを左に 25 度、左モータ 40%、右モータ 60%で進み、かなり減速しながら中心に寄るようにします。



(9)クロスラインチェック

```
221                 if( check_crossline() ){/* クロスラインチェック          */
222                     pattern = 21;
223                     break;
224                 }
```

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 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。

(10)右ハーフラインチェック

```
225                 if( check_rightline() ){/* 右ハーフラインチェック          */
226                     pattern = 51;
227                     break;
228                 }
```

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の前に実行するようにします。

(11)左ハーフラインチェック

```
225                 if( check_rightline() ){/* 右ハーフラインチェック          */
226                     pattern = 51;
227                     break;
228                 }
```

check_leftline 関数の戻り値は“0”で左ハーフラインではない、“1”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 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。

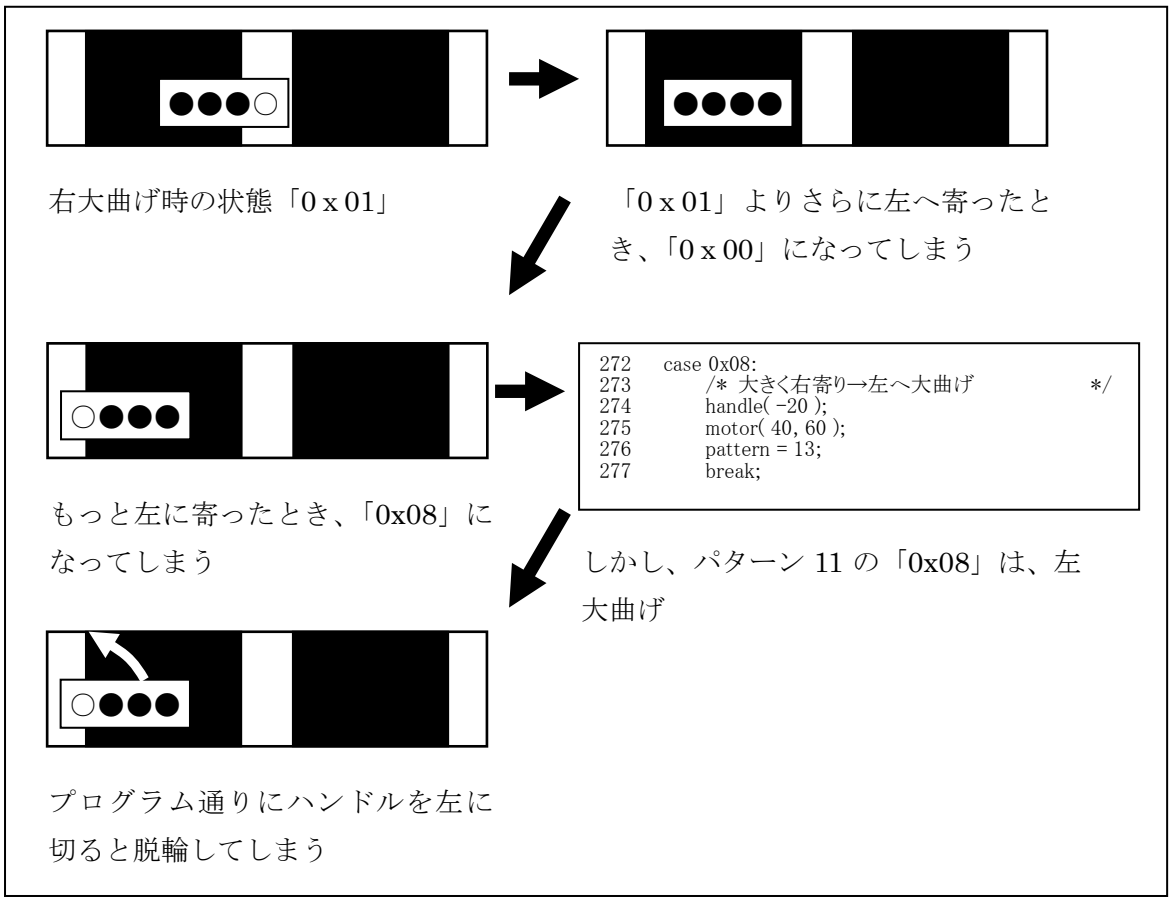
(12)それ以外

```
279                 default:
280                     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。

9.22.10 パターン 12：右へ大曲げの終わりチェック

センサ状態「0x01」は、一番大きく左に寄った状態です。そのため、これ以上カーブで脹らんだ場合、下図のようになります。



実際は、4 輪ミニマイコンカーが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曲げるのですぐに脱輪してしまいます。

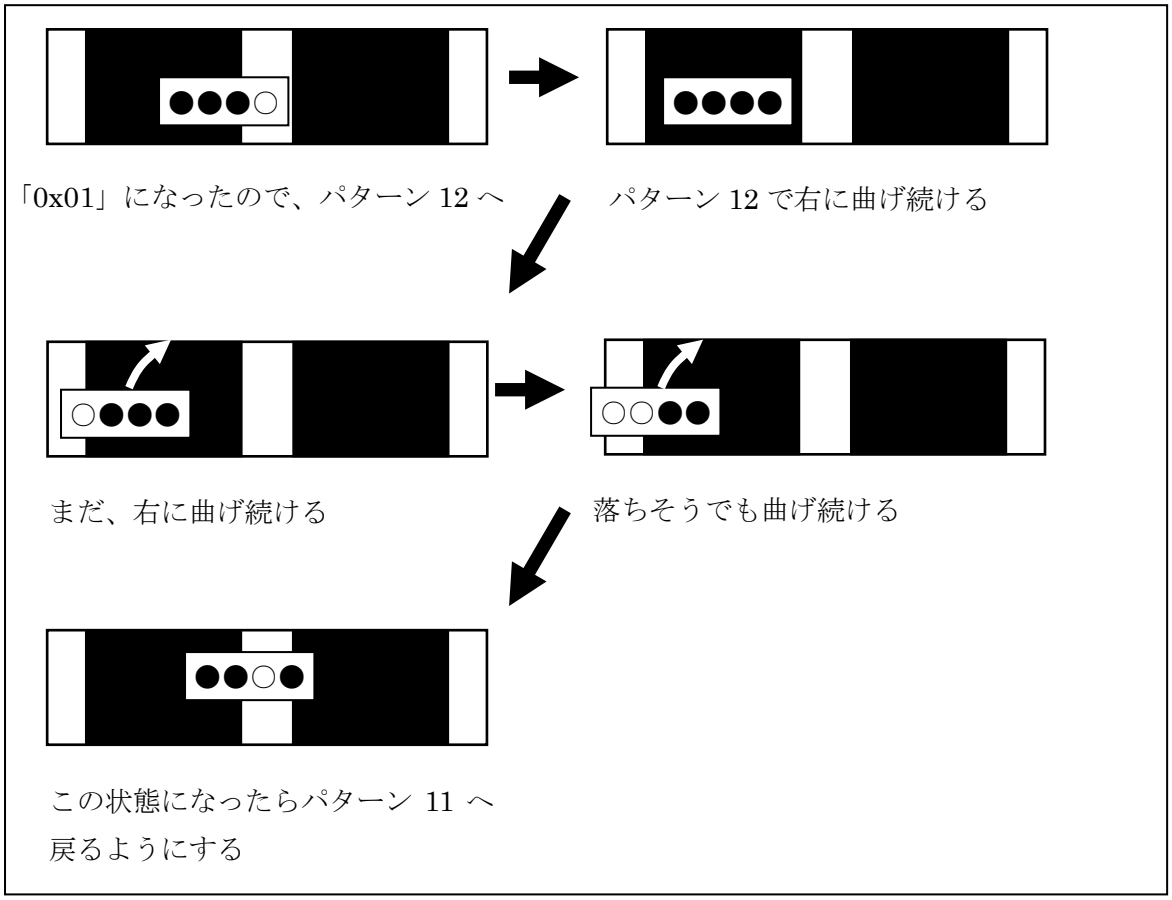
そこで、右に大曲げしたら、センサがある状態に戻るまで右に大曲げし続けます。この状態を判定するのが、パターン 12 になります。

パターン 11 の case 0x01 部分

```
253                     case 0x01:
254                         /* 大きく左寄り→右へ大曲げ          */
255                         handle( 20 );
256                         motor( 60, 40 );
257                         pattern = 12;
258                         break;
```

センサが「0x01」になると、257 行でパターン 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 に戻るか考えてみます。



センサ状態が 0x02 になったらパターン 11 へ戻るようにすれば、先ほどの誤った判断をなくせそうです。

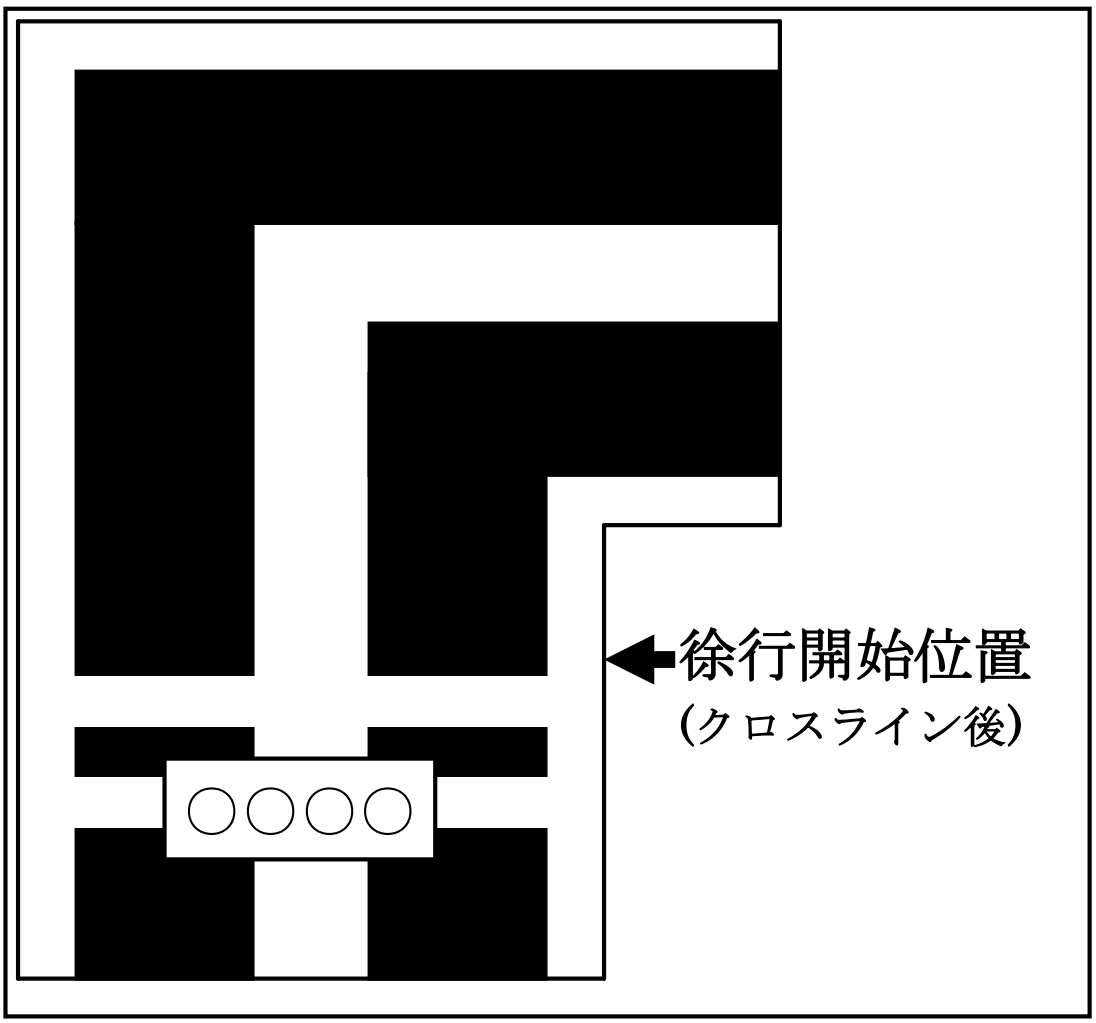
パターン 12 の状態でクロスラインや右ハーフライン、左ハーフラインがあった場合に検出することができません。そこで、パターン 12 にクロスラインチェック、右ハーフラインチェック、左ハーフラインチェックを行います。

```
284                 case 12:
285                     /* 右へ大曲げの終わりのチェック                    */
286                     if( check_crossline() ){/* クロスラインチェック          */
287                         pattern = 21;
288                         break;
289                     }
290                     if( check_rightline() ){/* 右ハーフラインチェック        */
291                         pattern = 51;
292                         break;
293                     }
294                     if( check_leftline() ){/* 左ハーフラインチェック         */
295                         pattern = 61;
296                         break;
297                     }
298                     if( sensor_get( MASK_0010 ) == 0x02 ){
299                         pattern = 11;
300                     }
301                     break;
```

パターン 13 の「左大曲げの終わりチェック」は、パターン 12 の「右大曲げの終わりチェック」の考え方を左右反対にしたものなので、説明は省略します。

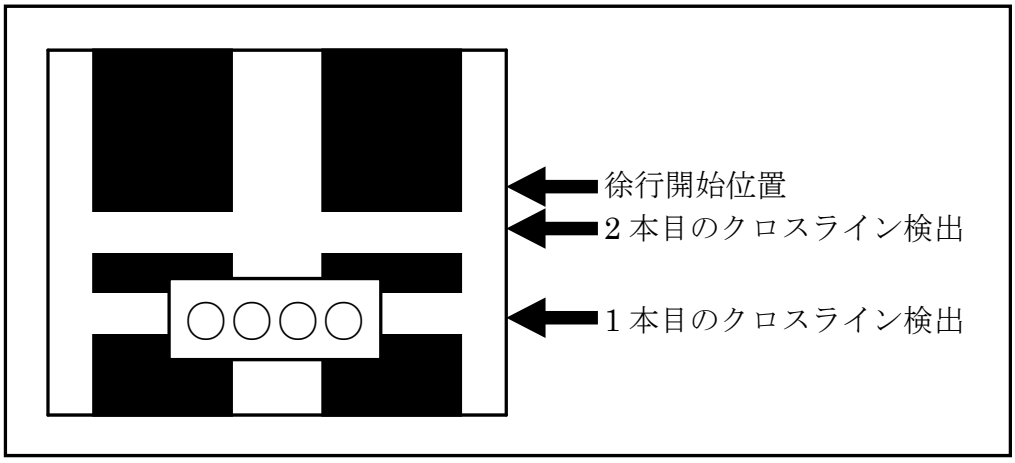
9.22.11 パターン 21：1 本目のクロスライン検出時の処理

パターン 21 は、下図のようにクロスラインを検出した場合の処理になります。



クロスラインの先には、クランク(直角)があります。速度を落とさずに直角を曲がるのは難しいのでクロスライン検知後にブレーキをかけて速度を落とす処理をします。

1本目のクロスラインを検出してから 2 本目のクロスラインを検出し終わるまでに下図のような状態があります。



コースが白→黒→白→黒と変化したことを検出して、徐行開始位置まで進んだか判別します。

考え方を変えてみます。クロスラインを検出して徐行開始位置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の判断をせずに済みそうです。サンプルプログラムでは 0.5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。クロスラインを検出したとき、ブザーで「ド」の音を鳴らして**「パターン 21 に入った」**ということを外部に知らせるようにします。

1 本目のクロスラインを検出すると下記を実行します。
・ブザーで「ド」の音を鳴らす(0.5 秒間)
・サーボモータを0度に
・左右モータ PWM を 0%にしてブレーキをかける

これをパターン 21 でプログラム化します。

```
322             case 21:
323                 /* 1本目のクロスライン検出時の処理            */
324                 beep( DO_5 );           /* 5オクターブ目のドの音をセット*/
325                 handle( 0 );
326                 motor( 0, 0 );
327                 pattern = 22;
328                 cnt1 = 0;
329                 break;
```

次に、0.5 秒待つパターンを作ります。プログラム化すると下記のようになります。

```
331             case 22:
332                 /* 2本目を読み飛ばす                          */
333                 if( cnt1 > 500 ){
334                     beep( 0 );          /* ブザー停止                 */
335                     pattern = 23;
336                     cnt1 = 0;
337                 }
338                 break;
```

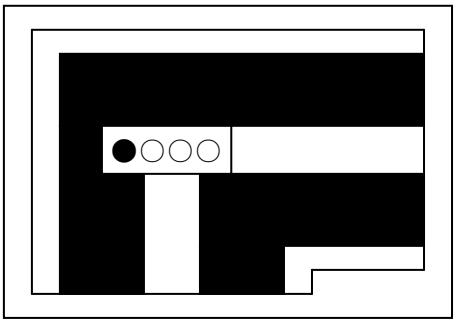
1 本目のクロスラインを検出したら、パターン 21 の 324 行で「ド」の音をセットします。サーボモータの角度は 0 度、左右のモータを 0%(ブレーキ)にします。pattern 変数に“22”を代入します。cnt1 の値を 0 にします。**もし、cnt1 の値に 1000 などの値が入っていた場合、パターン 22 の 333 行の if 文で条件が成り立ち、0.5 秒待たずに次のパターンに移ってしまいます。必ず、cnt1 はパターン 21 で“0”にします。**

9.22.12 パターン 23：クロスライン後のトレース、クランク検出

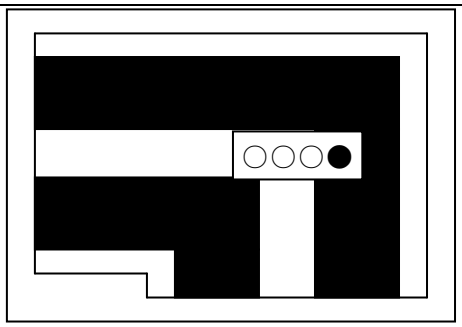
パターン 21、22 では、クロスライン検出後、ブレーキを 0.5 秒かけ 2 本目のクロスラインを通過させました。パターン 23 では、その後の処理を行います。

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サーボモータを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。

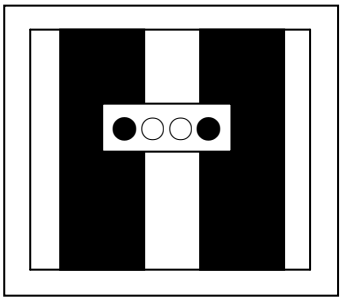
今回は、下図のようにパターンを考えました。

<table>
<tr>
<td> 

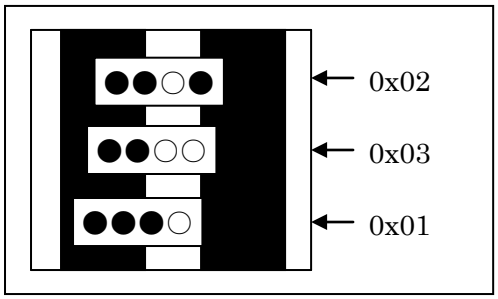
→0x07
（4つすべてチェック）</td>
<td>右クランク部分では、4つのセンサ状態が左図のように「0x07」になりました。そこで、「0x07」の状態を右クランクと判断するようにします。
このとき、サーボモータを右いっぱいまで曲げなければ外側へ脹らんで脱輪してしまいます。4 輪ミニマイコンカーの最大の切れ角は約 45 度です。プログラムでは 35 度に設定しています。45 度いっぱいに曲げることはできますが、進む方向に対してタイヤの角度がきつすぎるとタイヤがスリップしてうまく曲がれない可能性があります。
モータの回転について考えます。今回は右に曲がるので、右モータを少なく、左モータを多めにすることは予想できます。サンプルプログラムでは左モータ 60%、右モータ 20%に設定します。まとめると下記のようになります。

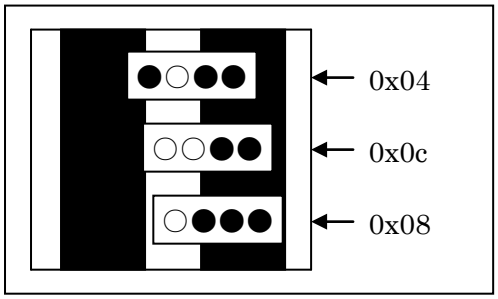
サーボモータ:35 度
左モータ:60% 右モータ:20%
その後、パターン 31 へ移ります。</td>
</tr>
<tr>
<td> 

→0x0e
（4 つすべてチェック）</td>
<td>左クランク部分です。考え方は右クランクと同様です。
まとめると下記のようになります。

サーボモータ:-35 度
左モータ:20% 右モータ:60%
その後、パターン 41 へ移ります。</td>
</tr>
<tr>
<td> 
</td>
<td>直進時、センサの状態は「0x06」です。これを直進状態と判断します。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。サンプルプログラムでは 60%に設定します。まとめると下記のようになります。

サーボモータ:0 度
左モータ:60% 右モータ:60%</td>
</tr>
</table>

<table>
<tr>
<td> 
</td>
<td>徐行走行中に 4 輪ミニマイコンカーが左に寄ったときを考えてみます。中心から少しずつ左へずらしていくと左図のように 3 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サーボモータを曲げます。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振れてしまいます。ちょうど良い角度調整は難しいです。サンプルプログラムではどの状態も10度にしています。まとめると下記のようになります。

サーボモータ:10 度
左モータ:60% 右モータ:50%</td>
</tr>
<tr>
<td> 
</td>
<td>右にずれたときは、左にずれたときの考え方を左右反対にしたものなので、説明は省略します。

サーボモータ:-10 度
左モータ:50% 右モータ:60%</td>
</tr>
</table>

プログラム化すると下記のようになります。

```
340             case 23:
341                 /* クロスライン後のトレース、クランク検出        */
342                 if( sensor_get( MASK_1111 ) == 0x07 ){
343                     /* 右クランクと判断→右クランククリア処理へ */
344                     handle( 35 );
345                     motor( 60, 20 );
346                     pattern = 31;
347                     cnt1 = 0;
348                     break;
349                 }
350                 if( sensor_get( MASK_1111 ) == 0x0e ){
351                     /* 左クランクと判断→左クランククリア処理へ */
352                     handle( -35 );
353                     motor( 20, 60 );
354                     pattern = 41;
355                     cnt1 = 0;
356                     break;
357                 }
358                 switch( sensor_get( MASK_1111 ) ){
359                     case 0x06:
360                         handle( 0 );
361                         motor( 60, 60 );
362                         break;
363                     case 0x02:
364                     case 0x03:
365                     case 0x01:
366                         handle( 10 );
367                         motor( 60, 50 );
368                         break;
369                     case 0x04:
370                     case 0x0c:
371                     case 0x08:
372                         handle( -10 );
373                         motor( 50, 60 );
374                         break;
375                     default:
376                         break;
377                 }
378                 break;
```

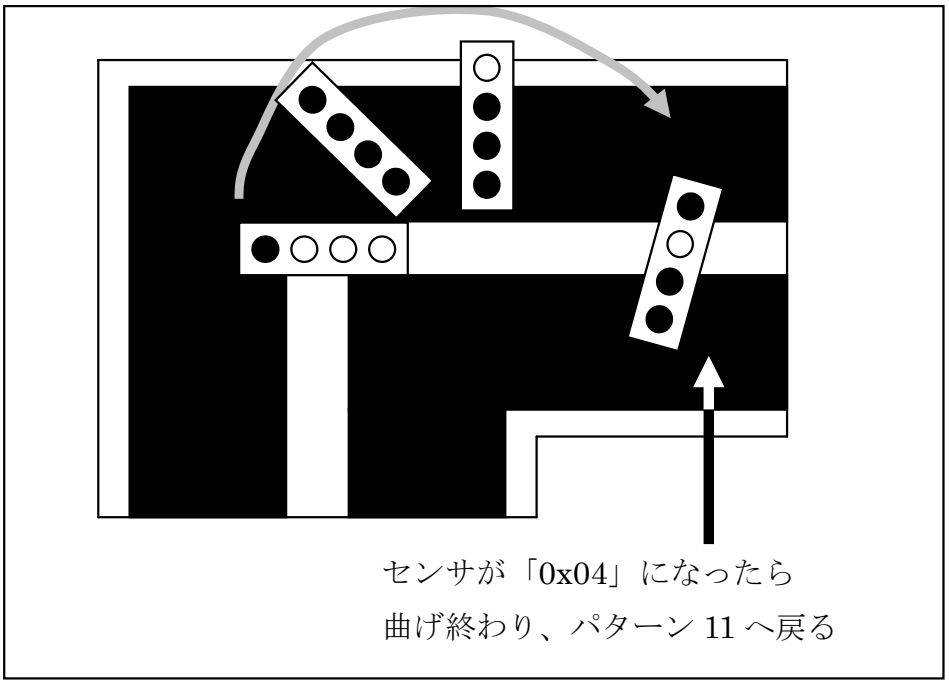
case を続けて書くと
0x02 または 0x03 または 0x01 のとき
という意味になります。

case を続けて書くと
0x04 または 0x0cまたは 0x08 のとき
という意味になります。

9.22.13 パターン 31、32：右クランククリア処理

パターン 23 でセンサ 4 つのうち右側 3 つのセンサが反応(0111)すると、右クランクと判断してサーボモータを右に大きく曲げクランクをクリアしようとします。

下図のように考えました。

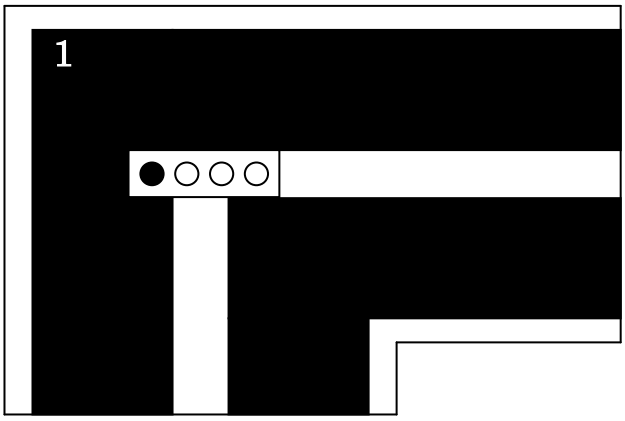


「0x07」と判断すると右へ大曲げしますが、スピードがついているので膨らみ気味に曲がっていきます。センサが中心付近に来て「0x04」となったとき曲げ終わりと判断してパターン 11 の通常トレースに戻ります。

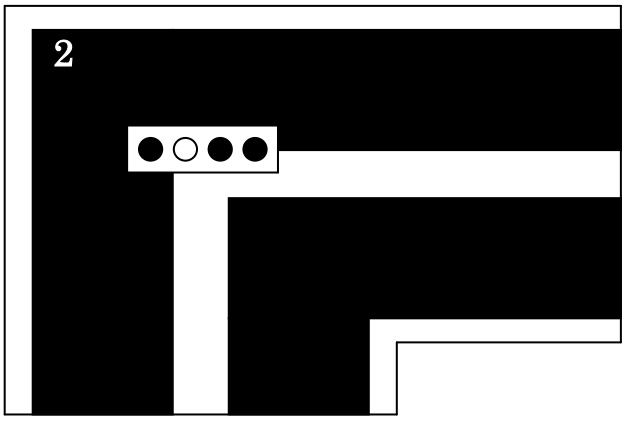
これをプログラム化してみます。

```
            case 31:
                /* 右クランククリア処理　曲げ終わりのチェック */
                if( sensor_get( MASK_0100 ) == 0x04 ){
                    pattern = 11;
                    cnt1 = 0;
                }
                break;
```

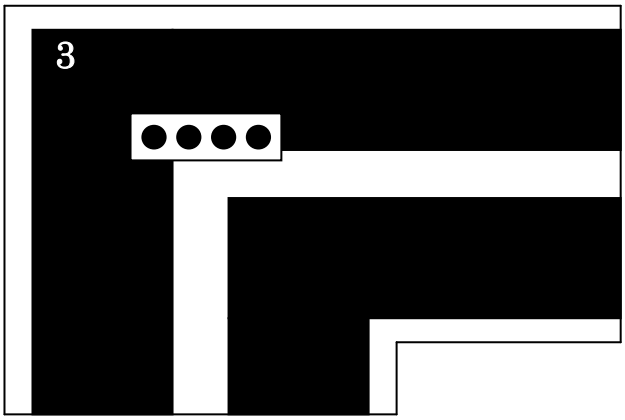
実際に走らせてみます。そうすると、センサ状態が「0x07」になった瞬間。サーボモータを右へ曲げ始めました。このままサーボモータを曲げ続けるかと思いきや、すぐにまっすぐ向いて直進し、そのまま脱輪してしまいました。このときのセンサ状態をじっくりと観察すると次のようになりました。



「0x07」を検出
「0x04」になるまで右に大曲げ



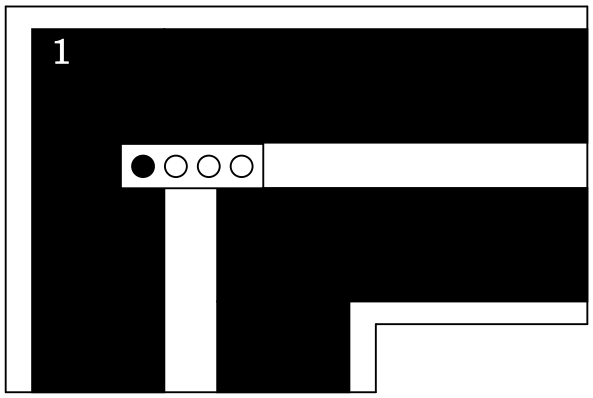
曲げた瞬間、白と黒の切り替わる部分で
センサ状態が「0x04」になってしまったため
大曲げを終了してパターン 11 へ移る



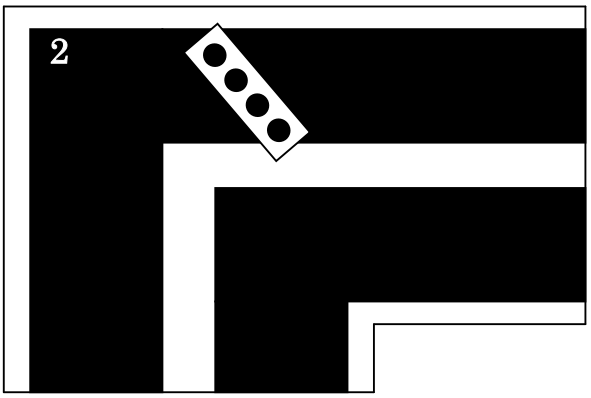
「0x00」を検出、パターン 11 のセンサパターンに「0x00」はないため、default になる

上図の 2 のように、白と黒の変わり目（本当は、白と灰と黒ですが、灰色は黒と見なします）でセンサの状態が「0x04」になっていることが分かりました。

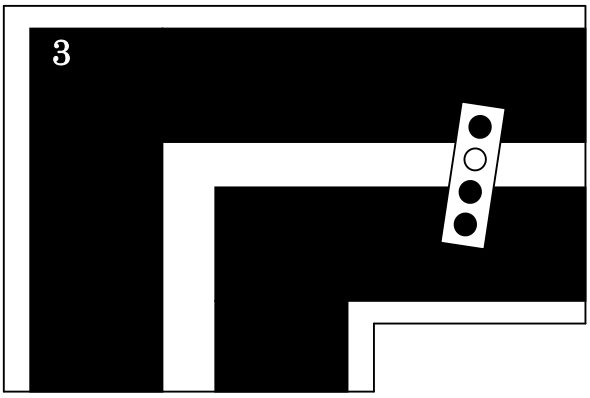
「0x07」を検出してから終わりのセンサ状態が「0x04」になるまで少し時間がかかることに気がつきました。そこで右クランクを検出した後、0.5 秒間はセンサ状態を見ないようにタイマを使って少し進むのを待ちます。その後、センサ状態をチェックするように考えました。図を書いてイメージしてみます。



「0x07」を検出
大曲げして 0.5 秒待つ



0.5 秒後、ここからセンサを見る
「0x04」まで曲げ続ける



「0x04」を検出
曲げ終わりの終了、パターン 11 へ戻る

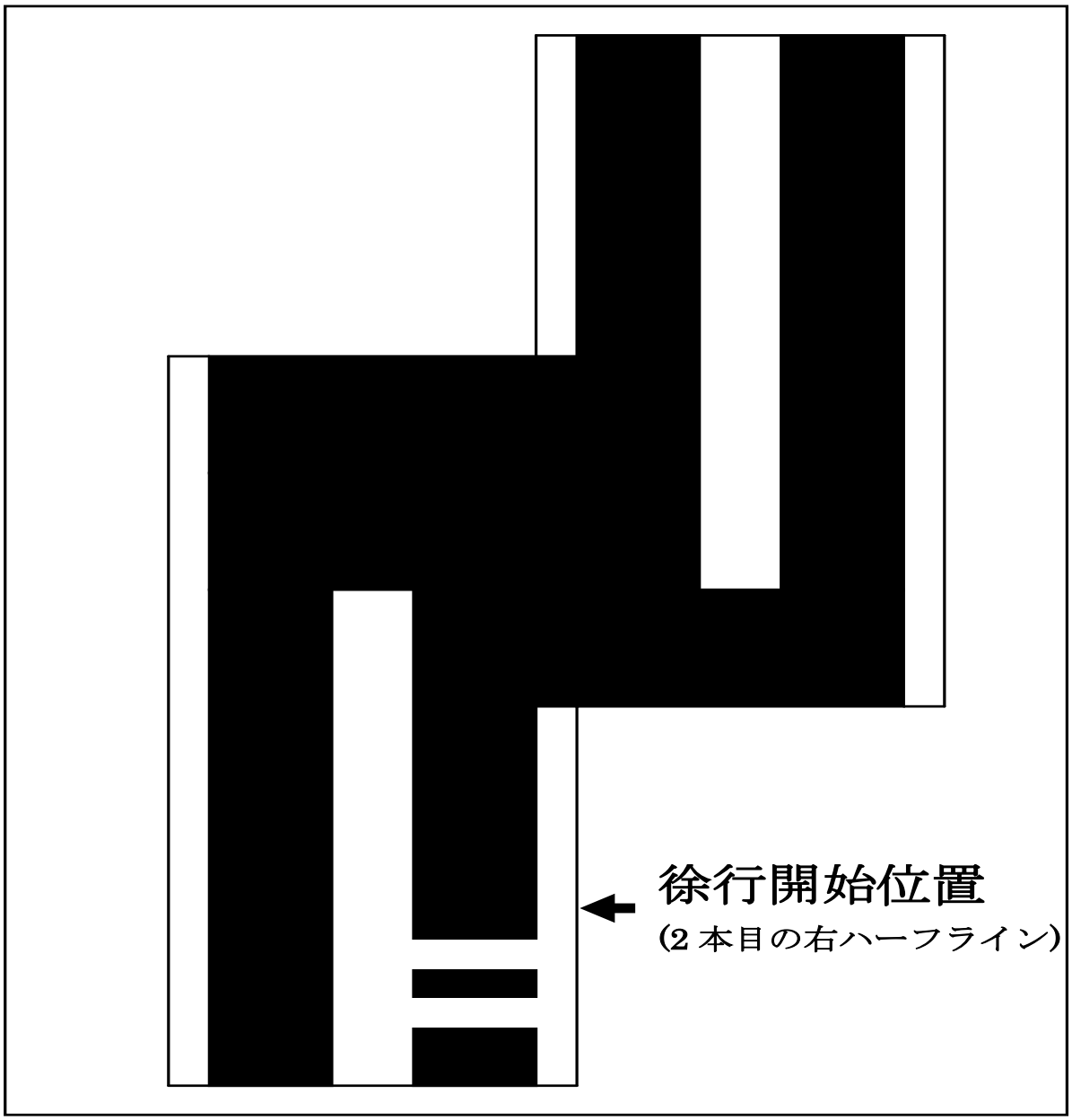
0.5 秒後、上図の 2 のように白と黒の変わり目を越えた位置にあります。その後に「0x04」になるまでサーボモータを曲げ続けるだけです。プログラム化します。

```
380             case 31:
381                 /* 右クランククリア処理　安定するまで少し待つ */
382                 if( cnt1 > 500 ){
383                     pattern = 32;
384                     cnt1 = 0;
385                 }
386                 break;
387
388             case 32:
389                 /* 右クランククリア処理　曲げ終わりのチェック */
390                 if( sensor_get( MASK_0100 ) == 0x04 ){
391                     pattern = 11;
392                     cnt1 = 0;
393                 }
394                 break;
```

382 行で cnt1 変数が 500 以上かチェックしています。500 以上(0.5 秒たった)ならパターン 32 に移ります。ちなみに、cnt1 のクリアは、パターン 31 へ移る前の 347 行で行っています。

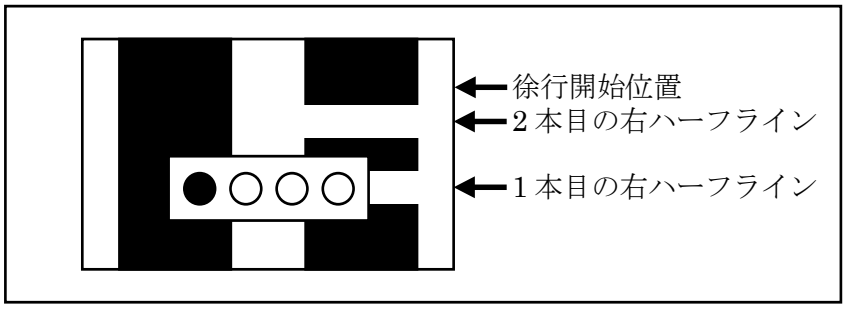
パターン 41、42(左クランククリア処理)については、右クランククリア処理の考え方を左右反対にしたものなので、説明は省略します。

9.22.14 パターン 51：1 本目の右ハーフライン検出時の処理



右ハーフライン後には、右レーンチェンジがあることを示しています。右レーンチェンジを走行するとき、速度を落とさずに走行するのは難しいため、右ハーフラインを検出後にブレーキをかけて速度を落とす処理をします。

1本目の右ハーフラインを検出してから 2 本目の右ハーフラインを検出し終わるまでに下図のような状態があります。



コースが白→黒→白→黒と変化したことを検出して、徐行開始位置まで進んだか判別します。

考え方を変えてみます。クロスラインを検出して徐行開始位置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の判断をせずに済みそうです。サンプルプログラムでは 0.5 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。

クロスライン検出時と同様に右ハーフラインを検出したときは、ブザーで「レ」の音を鳴らして外部に知らせるようにプログラムしています。(※左ハーフライン検出時は、「ミ」の音を鳴らすようにしています。)

1 本目の右ハーフラインを検出すると下記を実行します。
・ブザーで「レ」の音を鳴らす
・サーボモータを0度に
・左右のモータ PWM を 70%にして、少し減速します

これをパターン 51 でプログラム化します。

```
412             case 51:
413                 /* 1本目の右ハーフライン検出時の処理          */
414                 beep( RE_5 );           /* 5オクターブ目のレの音をセット*/
415                 handle( 0 );
416                 motor( 70, 70 );
417                 pattern = 52;
418                 cnt1 = 0;
```

次に、0.5 秒待つパターンを作ります。プログラム化すると下記のようになります。

```
421             case 52:
422                 /* 2本目を読み飛ばす                          */
423                 if( cnt1 > 500 ){
424                     beep( 0 );          /* ブザー停止                 */
425                     pattern = 53;
426                     cnt1 = 0;
427                 }
428                 break;
```

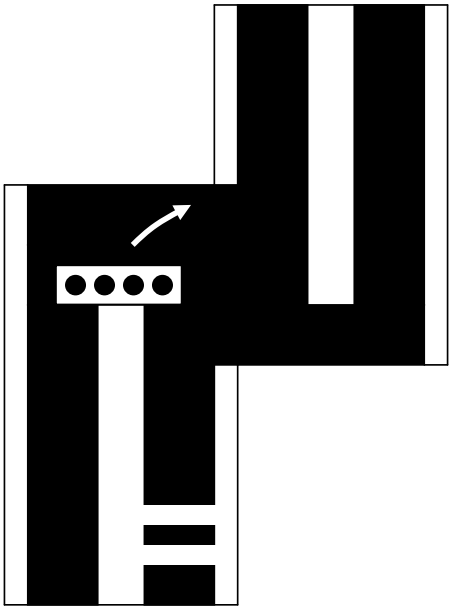
1 本目の右ハーフラインを検出したら、パターン 51 の 414 行で「レ」の音をセットします。サーボモータの角度は 0 度、左右のモータを 70%にして減速します。pattern 変数に“52”を代入します。cnt1 の値を 0 にします。**もし、cnt1 の値に 1000 などの値が入っていた場合、パターン 52 の 423 行の if 文で条件が成り立ち、0.5 秒待たずに次のパターンに移ってしまいます。必ず、cnt1 はパターン 51 で 0 にします。**

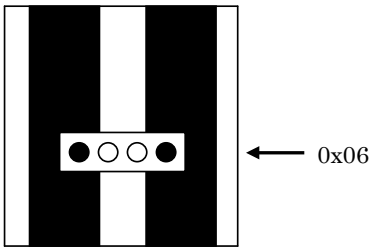
9.22.15 パターン 53：右ハーフライン後のトレース

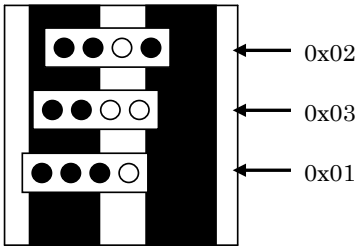
パターン 51、52 では、右ハーフラインを検出後、0.5 秒かけて 2 本目の右ハーフラインを通過させました。パターン 53 では、その後の処理を行います。

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。

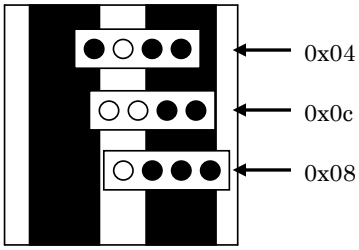
今回は、下図のように考えました。

<table>
<tr>
<td> 
→0x00
（4 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、4 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、サーボモータの曲げ角、モータの回転はどうすればよいのでしょうか。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。サンプルプログラムでは下記のように設定します。

サーボモータ:15 度
左モータ:70%　右モータ:60%
その後、パターン 54 へ移ります。</td>
</tr>
<tr>
<td> 
0x06</td>
<td>直進時、センサの状態は「0x06」です。これを直進状態と判断します。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、サンプルプログラムでは 70%にしておきます。まとめると下記のようになります。

サーボモータ:0 度
左モータ:70%　右モータ:70%</td>
</tr>
<tr>
<td> 
0x02
0x03
0x01</td>
<td>徐行走行中に 4 輪ミニマイコンカーが左に寄ったときを考えてみます。中心から少しずつ左へずらしていくと左図のように 3 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サーボモータを曲げます。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振れてしまいます。サンプルプログラムでは、どの状態も 10 度にしています。まとめると下記のようになります。

サーボモータ:10 度
左モータ:70%　右モータ:60%</td>
</tr>
</table>



右に寄ったときについては、左に寄ったときの考え方を左右反対にしたものなので、説明は省略します。まとめると下記のようになります。

サーボモータ:-10 度
左モータ:60% 右モータ:70%

プログラム化すると下記のようになります。

```
430             case 53:
  431                   /* 右ハーフライン後のトレース、レーンチェンジ */
  432                   if( sensor_get( MASK_1111 ) == 0x00 ){
  433                       handle( 15 );
  434                       motor( 70, 60 );
  435                       pattern = 54;
  436                       cnt1 = 0;
  437                       break;
  438                   }
  439                   switch( sensor_get( MASK_1111 ) ){
  440                       case 0x06:
  441                           handle( 0 );
  442                           motor( 70, 70 );
  443                           break;
  444                       case 0x02:
  445                       case 0x03:
  446                       case 0x01:
  447                           handle( 10 );
  448                           motor( 70, 60 );
  449                           break;
  450                       case 0x04:
  451                       case 0x0c:
  452                       case 0x08:
  453                           handle( -10 );
  454                           motor( 60, 70 );
  455                           break;
  456                       default:
  457                           break;
  458                   }
```

case を続けて書くと
0x02 または 0x03 または 0x01 のとき
という意味になります。

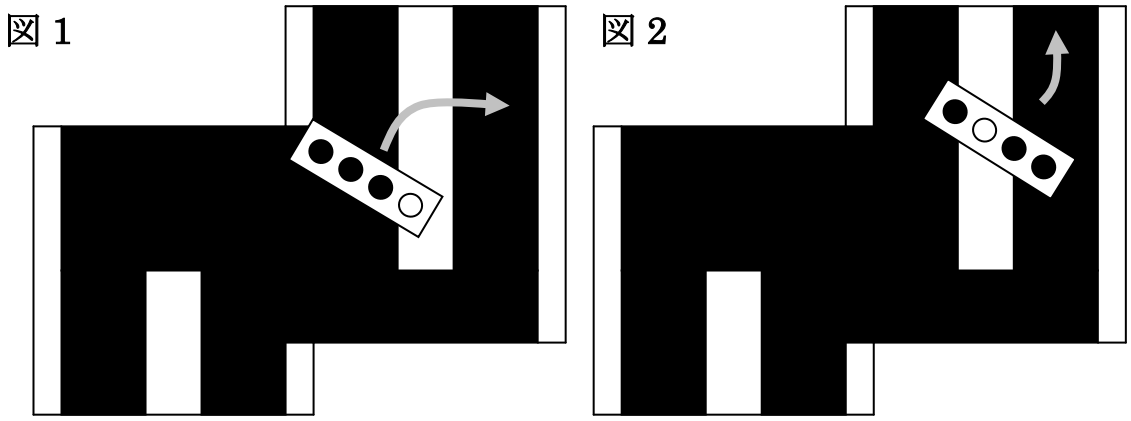
case を続けて書くと
0x04 または 0x0c または 0x08 のとき
という意味になります。

9.22.16 パターン 54：右レーンチェンジ終了のチェック

ここまでで、
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サーボモータを右へ曲げる

処理を行いました。次は、レーンチェンジ後の中心線を検出して通常トレースに戻ります。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。

下図のような状態が考えられます。



どのセンサが反応したときにパターン 11 へ戻したらよいでしょうか。図 1 のように、0x01 の検出でパターン 11 に戻したとします。パターン 11 の 0x01 は、右大曲の状態になり、サーボモータを大きく右に曲げてしまい、脱輪してしまいます。

図 2 ではどうでしょうか。0x04 の状態でパターン 11 に戻ります。パターン 11 の 0x04 は、左小曲げの状態になり、中心線に戻ろうとします。

これをプログラム化してみます。

```
461             case 54:
462                 /* 右レーンチェンジ終了のチェック              */
463                 if( sensor_get( MASK_0100 ) == 0x04 ){
464                     pattern = 11;
465                     cnt1 = 0;
466                 }
467                 break;
```

右レーンチェンジの終了チェックを 463 行で行っています。今回はマスク値を「MASK_0100」に設定し、左から 2 つ目のセンサのみをチェックするようにします。

パターン 61～64（左レーンチェンジ）については、右レーンチェンジの考え方を反対にしたものなので、説明は省略します。

9.22.17 どれでもないパターン

```
526             default:
527                 /* どれでもない場合は待機状態に戻す            */
528                 pattern = 0;
529                 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しょう。

以上で、プログラムの解説は終わりです。

※「startup.c」の内容については、「R8C/35A マイコン実習マニュアル」を参照してください。
「R8C/35A マイコン実習マニュアル」は、こちらの URL（http://www2.himdx.net/mcr/）よりダウンロードできます。

10 参考文献

・ルネサス エレクトロニクス(株)
R8C/35A グループ ハードウェアマニュアル Rev.0.40
・ルネサス エレクトロニクス(株)
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 半導体トレーニングセンター C言語入門コーステキスト 第 1 版
・電波新聞社 マイコン入門講座 大須賀威彦著 第 1 版
・ソフトバンク(株) 新C言語入門シニア編 林晴比古著 初版
・共立出版(株) プログラマのための ANSI C 全書 L.Ammeraal 著
吉田敬一・竹内淑子・吉田恵美子訳 初版

マイコンカーラリー、販売部品についての詳しい情報は、マイコンカーラリー販売サイトをご覧ください。

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

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

http://japan.renesas.com/

の「製品情報」→「マイコン」→「R8C」でご覧頂けます