画像処理マイコンカーキット
kit20_gr-peach
プログラム
解説マニュアル

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

第 1.00 版
2021.08.15
ジャパンマイコンカーラリー実行委員会
株式会社日立ドキュメントソリューションズ

注 意 事 項 (rev.6.0J)

著作権

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

禁止事項

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

転載、複製

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

責任の制限

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

その他

・本マニュアルに記載の情報は本マニュアル発行時点のものであり、ジャパンマイコンカーラリー実行委員会は、予告なしに、本マニュアルに記載した情報または仕様を変更することがあります。製作に当たりましては、最新の内容を確認いただきますようお願いします。
・すべての商標および登録商標は、それぞれの所有者に帰属します。

連絡先

株式会社 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. マイコンカーラリー 1
1.1 マイコンカーラリー大会の部門 1
1.2 マイコンカーの規定 1
1.3 マイコンカーコース、スタートバーの仕様 2
1.3.1 コースの材質 2
1.3.2 基本的なコース 2
1.3.3 上り坂、下り坂 3
1.3.4 横線からクランク部分 3
1.3.5 レーンチェンジ部分 4
1.3.6 スタートバーの規格 5
1.3.7 コースレイアウト 6
2. 画像処理マイコンカーキットVer.1.0のハードウェア 7
3. GR-MCR基板Rev.1.0 8
3.1 仕様 9
3.2 回路図 10
3.3 寸法 12
3.4 外観 13
4. モータドライブ基板Ver.5 14
4.1 仕様 14
4.2 回路図 15
4.3 寸法 16
4.4 外観 17
4.5 モータドライブ基板Ver.5のCN2とGR-PEACHボードとの関係 19
4.6 モータ制御 20
4.6.1 モータドライブ基板の役割 20
4.6.2 スピード制御の原理 20
4.6.3 正転、逆転の原理 21
4.6.4 ブレーキとフリー 22
4.6.5 Hブリッジ回路 23
4.6.6 Hブリッジ回路のスイッチをFETにする 23
4.6.7 PチャネルFETとNチャネルFETの短絡防止回路 26
4.6.8 フリー回路 29
4.6.9 実際の回路 30
4.6.10 左モータの動作 32
4.6.11 右モータの動作 32
4.7 サーボ制御 33
4.7.1 原理 33
4.7.2 回路 34
4.8 LED制御 34
4.9 スイッチ制御 35
5. サンプルプログラムのダウンロード、インポート 36
5.1 ダウンロード 36

5.2 インストール 37
5.3 インポート 38

6. プログラム解説「kit20_gr-peach.cpp」 42

6.1 プログラムリスト 42
6.2 プログラムの解説 61
6.2.1 スタート 61
6.2.2 外部ファイルの取り込み（インクルード） 61
6.2.3 シンボル定義 62
6.2.4 プロトタイプ宣言 64
6.2.5 グローバル変数の宣言 65
6.2.6 init_MTU2_PWM_Motor関数 67
6.2.7 init_MTU2_PWM_Servo関数 68
6.2.8 Start_Video_Camera関数 69
6.2.1 IntCallbackFunc_Vfield関数 70
6.2.2 intTimer関数（1msごとの割り込み） 71
6.2.3 intImagePro関数 72
6.2.4 sensor_inp関数（センサ状態の読み込み） 75
6.2.5 check_crossline関数（クロスラインチェック） 78
6.2.6 check_rightling関数（右ハーフライン検出処理） 79
6.2.7 check_leftline関数（左ハーフライン検出処理） 80
6.2.8 dipsw_get関数（ディップスイッチ値読み込み） 81
6.2.9 pushsw_get関数（プッシュスイッチ値読み込み） 82
6.2.10 led_out関数（LED制御） 82
6.2.11 motor関数（モータ速度制御） 84
6.2.12 handle関数（サーボハンドル操作） 87
6.2.13 スタート 88
6.2.14 パターン方式について 90
6.2.15 プログラムの作り方 90
6.2.16 パターンの内容 92
6.2.17 パターン方式の最初while、switch部分 93
6.2.18 パターン0：スイッチ入力待ち 94
6.2.19 パターン1：スタートバーが開いたかチェック 95
6.2.20 パターン11：通常トレース 96
6.2.21 パターン12：右へ大曲げの終わりのチェック 105
6.2.22 パターン13：左へ大曲げの終わりのチェック 109
6.2.23 クランク概要 114
6.2.24 パターン21：クロスライン検出時の処理 115
6.2.25 パターン23：クロスライン後のトレース、クランク検出 117
6.2.26 パターン31、32：左クランククリア処理 120
6.2.27 パターン41、42：右クランククリア処理 124
6.2.28 右レーンチェンジ概要 128
6.2.29 パターン51：右ハーフライン検出時の処理 129
6.2.30 パターン53：右ハーフライン後のトレース 131
6.2.31 パターン54：右レーンチェンジ終了のチェック 133
6.2.32 左レーンチェンジ概要 135
6.2.33 パターン61：左ハーフライン検出時の処理 136
6.2.34 パターン63：左ハーフライン後のトレース 138
6.2.35 パターン64：左レーンチェンジ終了のチェック 140

1. マイコンカーラリー

1.1 マイコンカーラリー大会の部門

ジャパンマイコンカーラリー大会は、高等学校在籍者、または特別支援学校の高等部に在籍している生徒(以下、高校生)が参加対象です。2021 年現在、競技は下記の 3 部門が行われます。

●Advanced Class

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

●Basic Class

マイコンカーラリー初心者を対象とした部門で、初めてマイコンカーラリーの大会に参加する高校生を対象としています(大会に参加した年度のみ参加可能です)。1 年生はもとより、2 年生、3 年生であっても初めてマイコンカーラリーに取り組む生徒は参加できます。Advanced Class と比べ、使える部品が限定されています。
もちろん、初めて参加するからといって Basic Class に参加しなければいけない訳ではありません。Advanced Class、Camera Class への参加も可能です。

●Camera Class

2019 年度から新しく新設された部門です。Camera Class は、コース検出センサにイメージセンサ(カメラ)を使った部門です。Basic Class と同様に使える部品が限定されています。

※一般の部について

一般の部は、2009 年度にジャパンマイコンカーラリーから分離しました。詳しくは、マイコンカーラリーホームページ(https://www2.himdx.net/mcr/general/index.html)をご覧ください。

1.2 マイコンカーの規定

最新の Advanced Class、Basic Class、Camera Class の規定については、マイコンカーラリー公式ホームページ(https://www2.himdx.net/mcr/jmcr/index.html)に掲載されている競技規則を参照してください。

1.3 マイコンカーコース、スタートバーの仕様

1.3.1 コースの材質

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

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

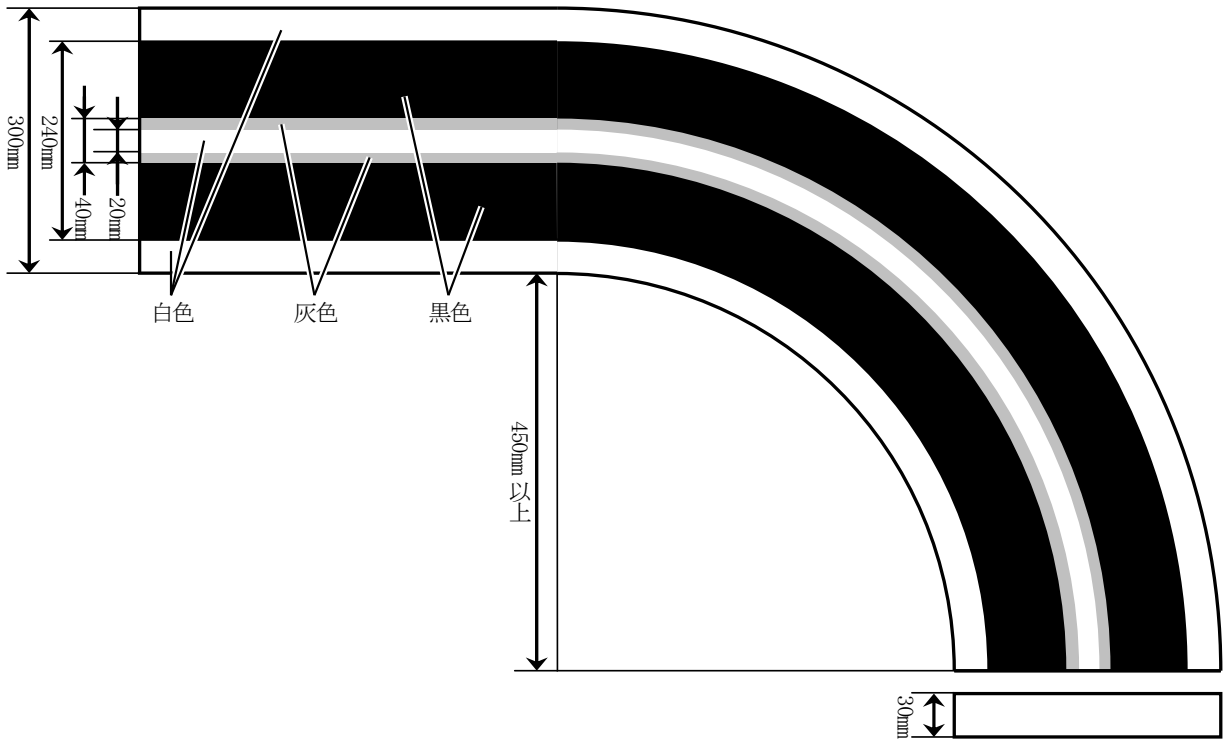
●灰色
セキスイハルカラーHC-050、エコパレットハルカラーHKC-057、中川ケミカル 735(ミディアムグレー)
のいずれか

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

※2021 年 4 月現在、販売されているコースは、中川ケミカルのシールを使っています(予告無く変更の可能性があります)。

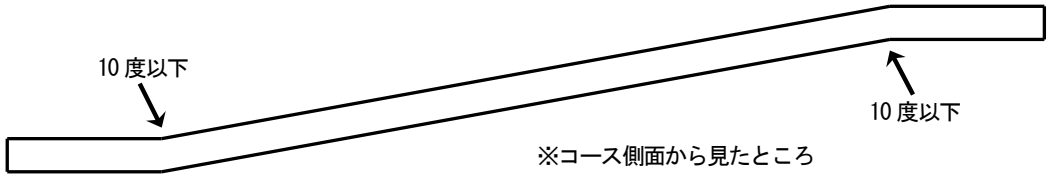
1.3.2 基本的なコース

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



1.3.3 上り坂、下り坂

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



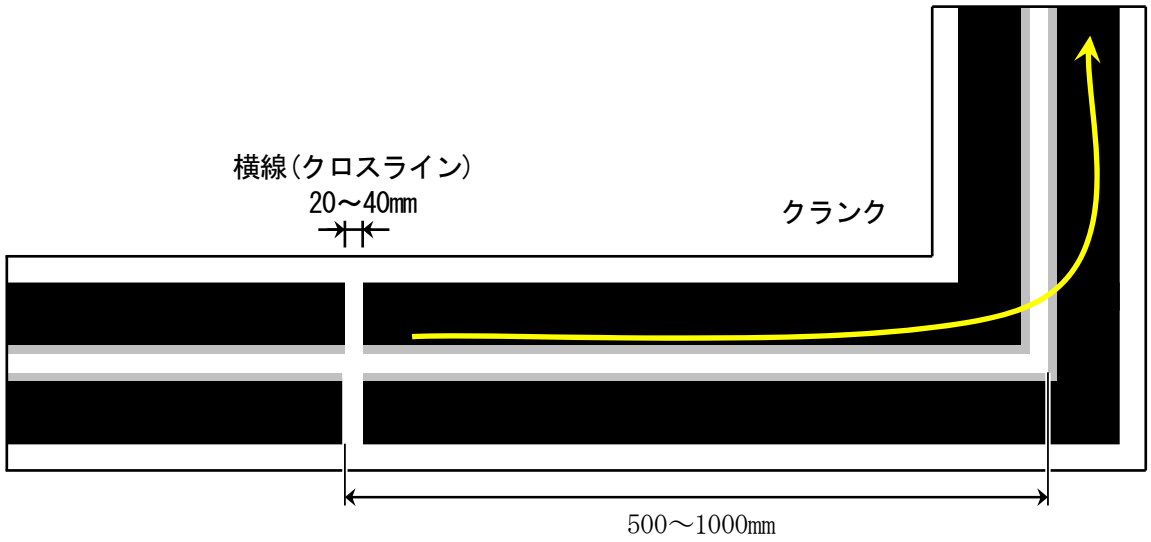
※車検は、上り下りコースパーツ部（10度以内の傾斜がついた坂道コースの一部）を使用して、マイコンカーを手動で通過させます。このとき、センサ類（タイヤ、アースを含む）以外はコースに接触してはいけません。
2 輪タイプでコース接触部にコース保護材をつけたものはタイヤの一部と見なします。
車検時に、センサ部においてコースを損傷させる可能性が確認された場合は、保護材などで対処をお願いします（エンコーダやリミットスイッチ、センサ含む）。

上り坂、下り坂を組み合わせた立体交差

1.3.4 横線からクランク部分

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

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

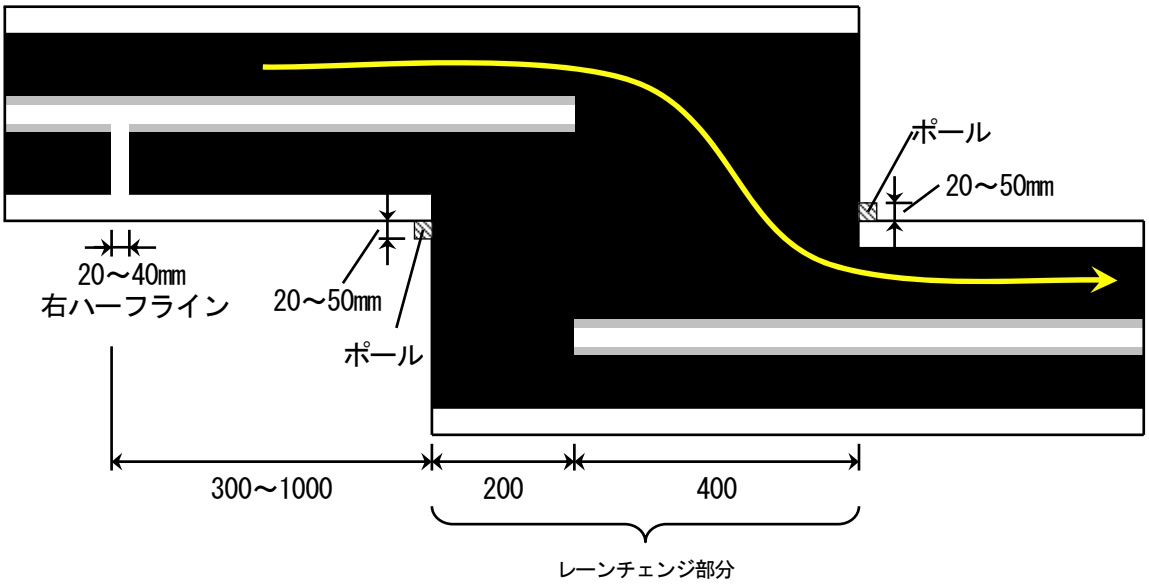


1.3.5 レーンチェンジ部分

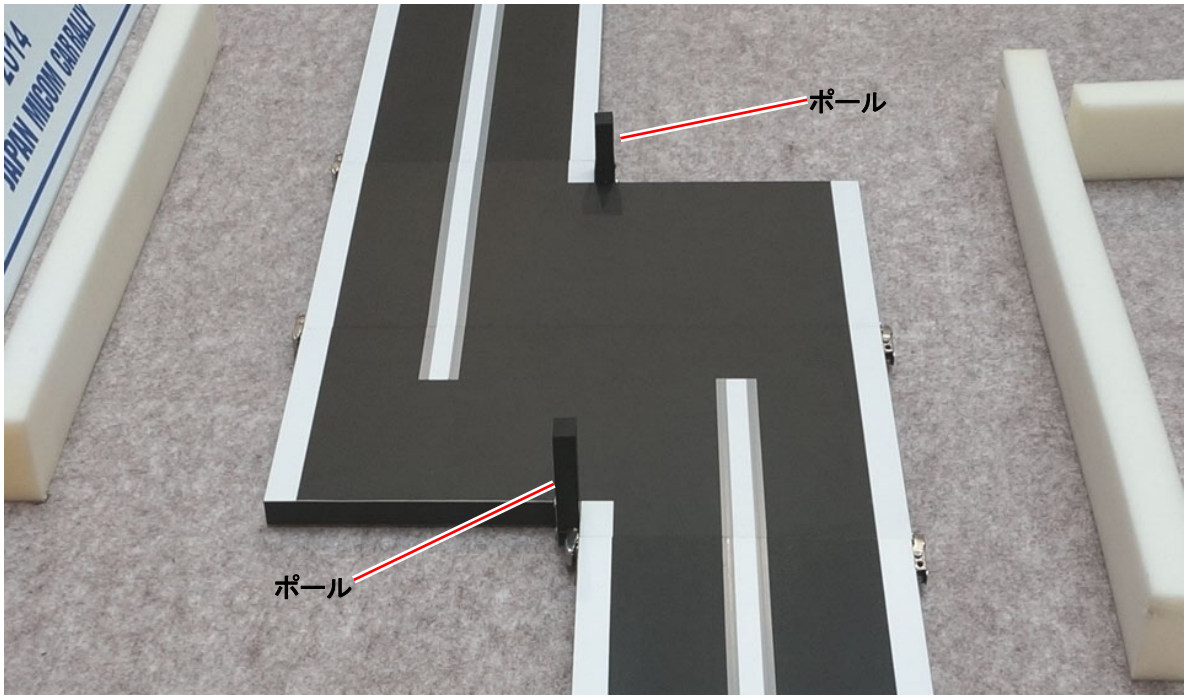
レーンチェンジは、チェンジ区間長さ600mm、幅600mmの部分があり、チェンジ区間より手前300～1000mmの地点に、幅20mm～40mmの白線をチェンジ方向に合わせ(左右片側に)1本引いている部分です。

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

右レーンチェンジのコースを、下記に示します。



※2013年度よりガードレールが、ポールという名称に変更になりました。



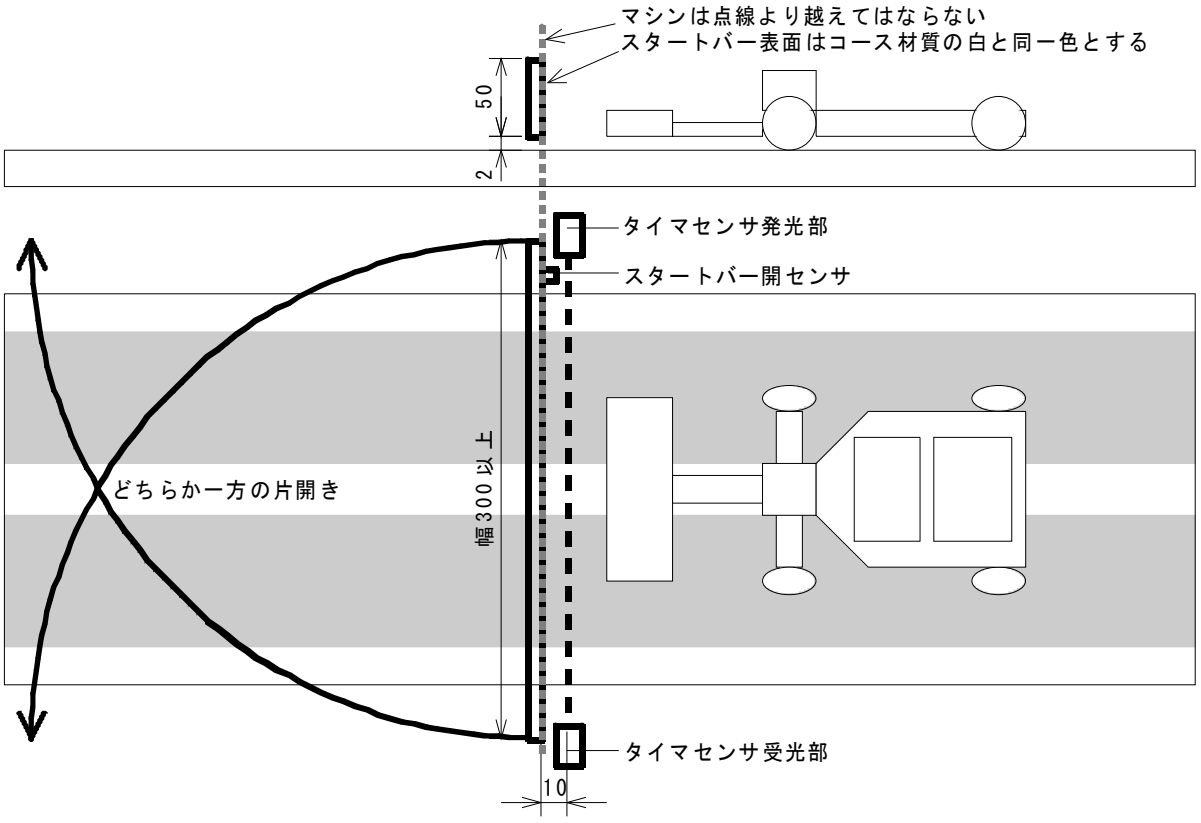
▲ポールの設置例

1.3.6 スタートバーの規格

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

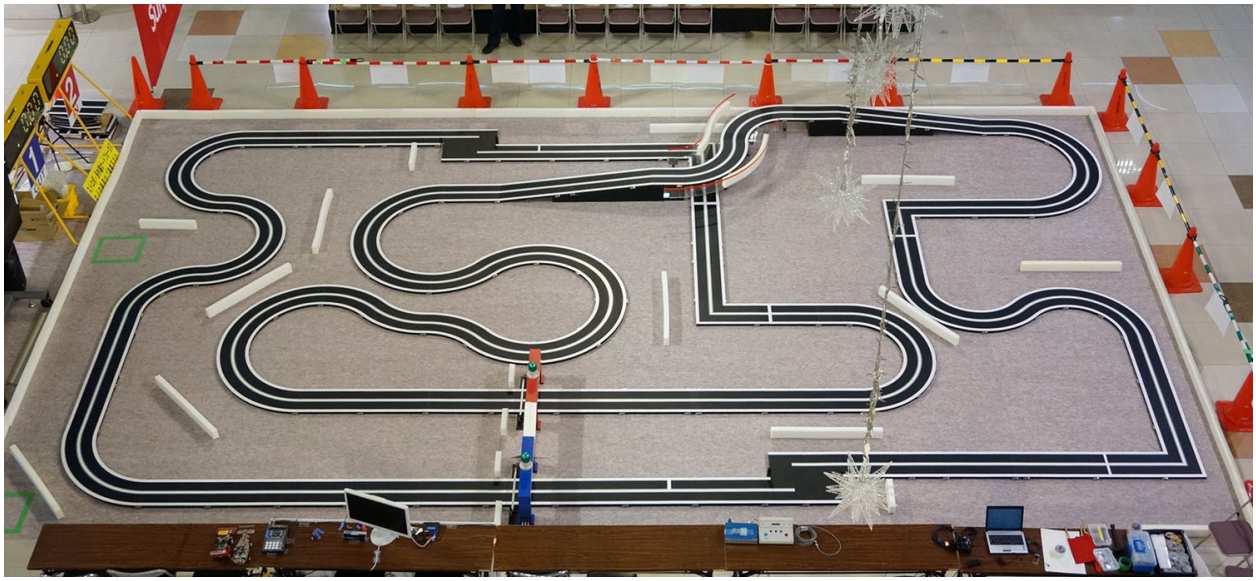
1. 選手は、マイコンカーをスタートバーに触れないように、かつスタートバーを越えないようにセットします。
2. 審判が、マイコンカーをセットできたか確認します。選手は、準備ができたら審判にセットできた旨を伝えます（一般的には手を挙げて合図しますが、各大会で異なります。各大会のルールを確認してください）。セット後は、マイコンカーに触れることはできません。
3. スタートバー（表面はコース材質の白色のシールが貼ってあります）が進行方向に開きます（押し扉のイメージ）。
4. マイコンカーは、スタートバーが開いたことを自動で検出してスタートします（センサ基板 Ver.5 には、スタートバー検出用のセン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

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



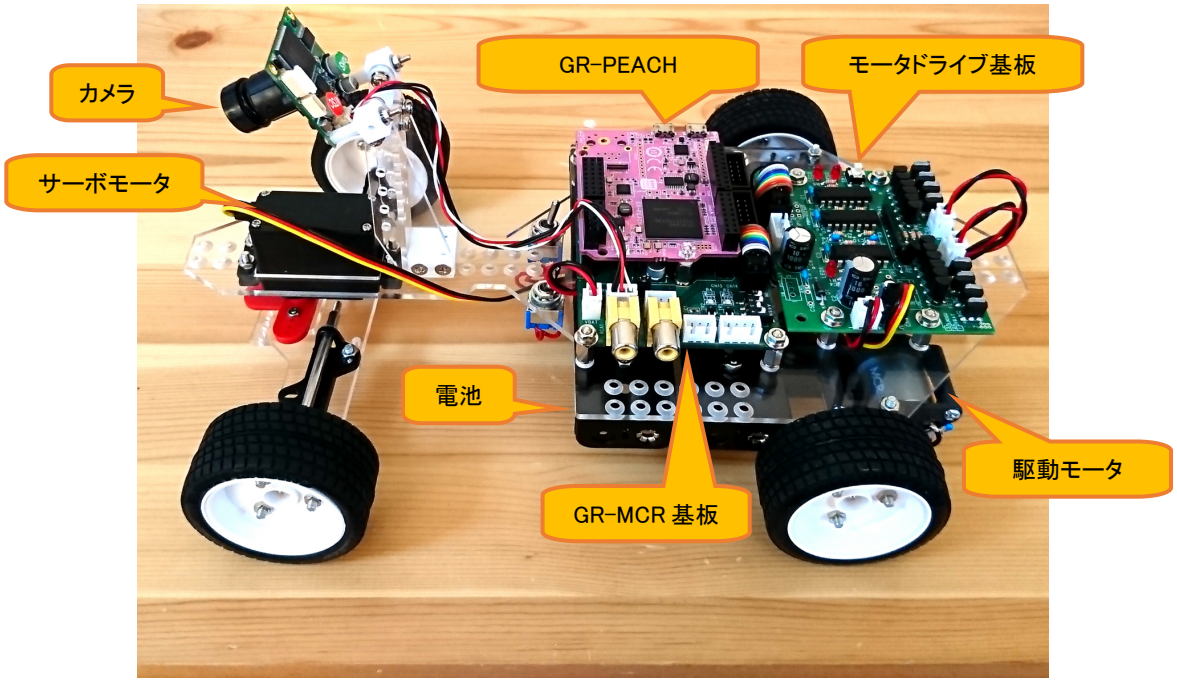
1.3.7 コースレイアウト

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



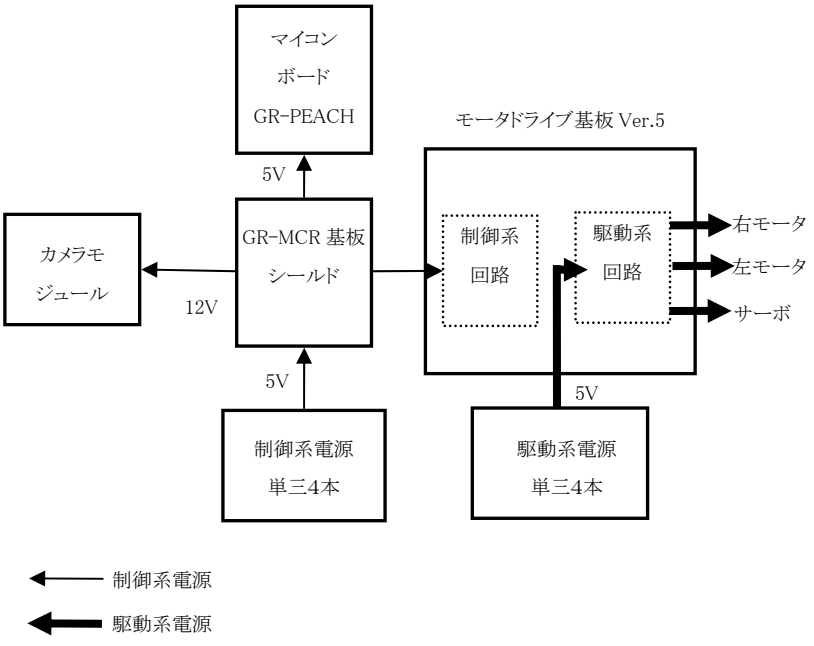
JMCR2015 全国大会　予選コースレイアウト

2. 画像処理マイコンカーキット Ver.1.0 のハードウェア



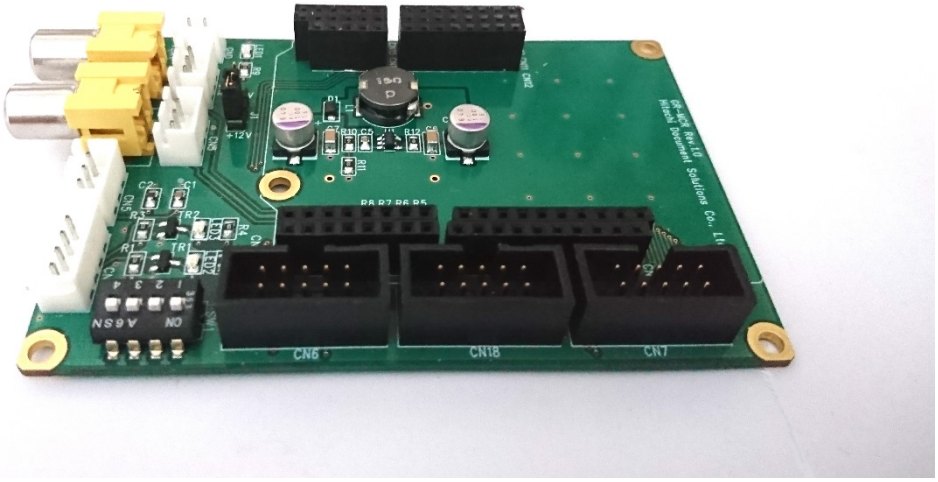
画像処理マイコンカーキットは、制御系の GR-PEACH ボード（RZ/A1H マイコン搭載のマイコンボード）、カメラモジュール、GR-MCR 基板 Rev.1.0、モータドライブ基板 Ver.5、駆動系の左モータ、右モータ、サーボモータで構成されています。

電源系の流れを下記に示します。

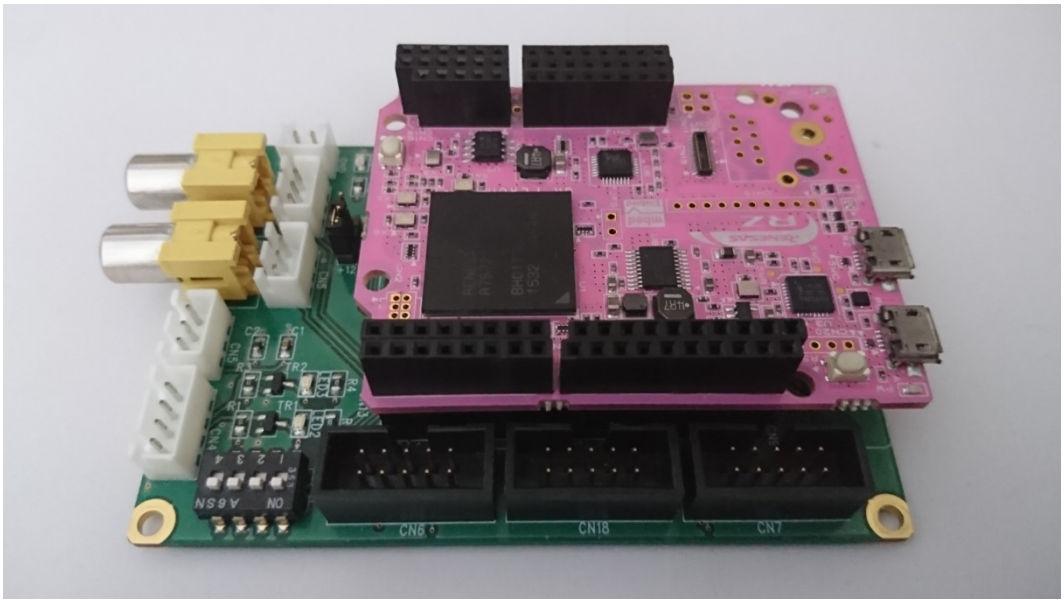


3. GR-MCR 基板 Rev.1.0

GR-MCR 基板 Rev.1.0 は、カメラ入力（NTSC）、カメラモジュール用電源（12V または、5V）、アナログ入力、ディップスイッチ、ロータリエンコーダ入力、GR-PEACH のコネクタを 10 ピンコネクタに変換し、モータドライブ基板 Ver.5 に接続するための基板です。



▲完成例



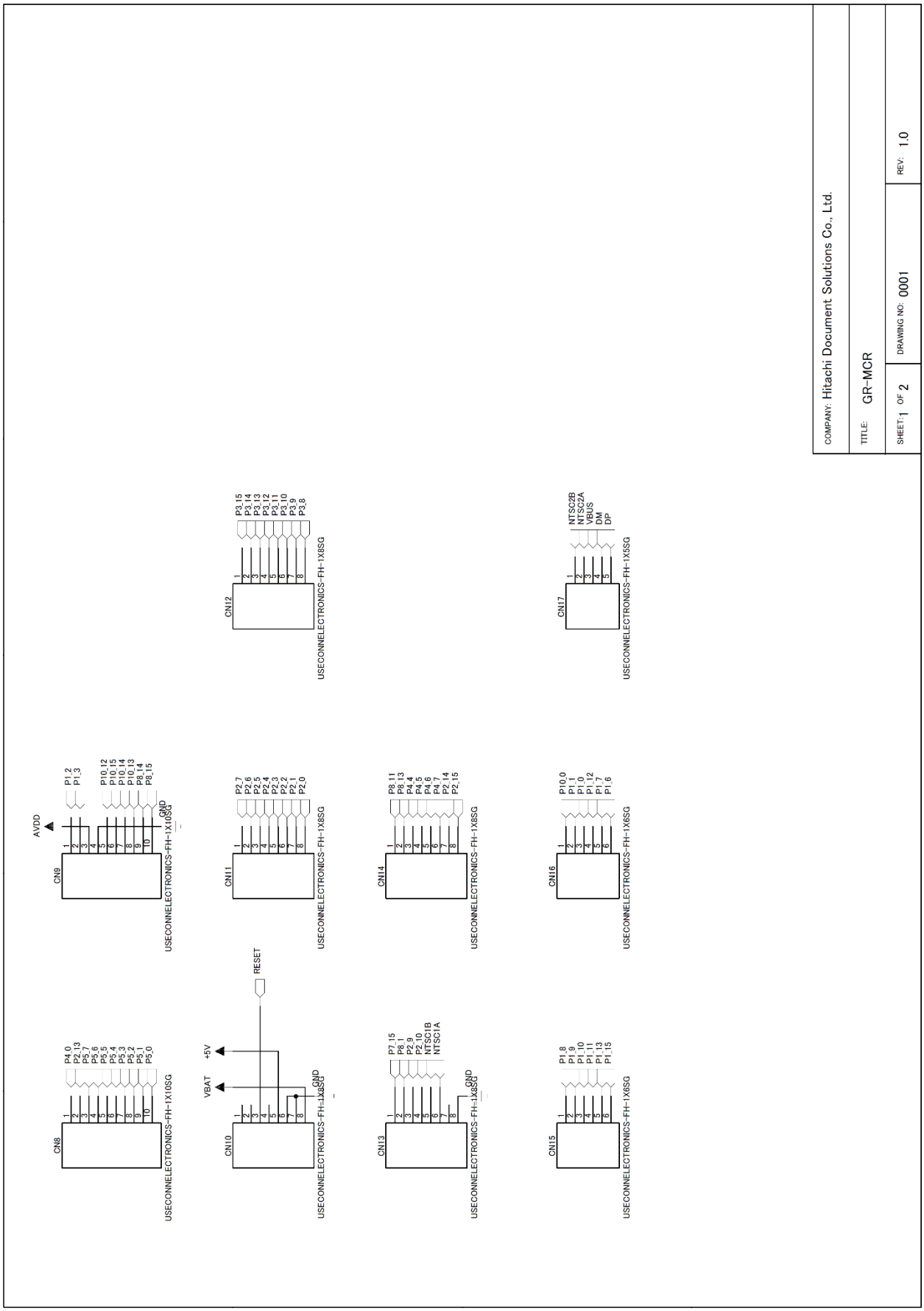
▲GR-MCR 基板 Rev.1.0 に GR-PEACH ボードを取り付けたところ

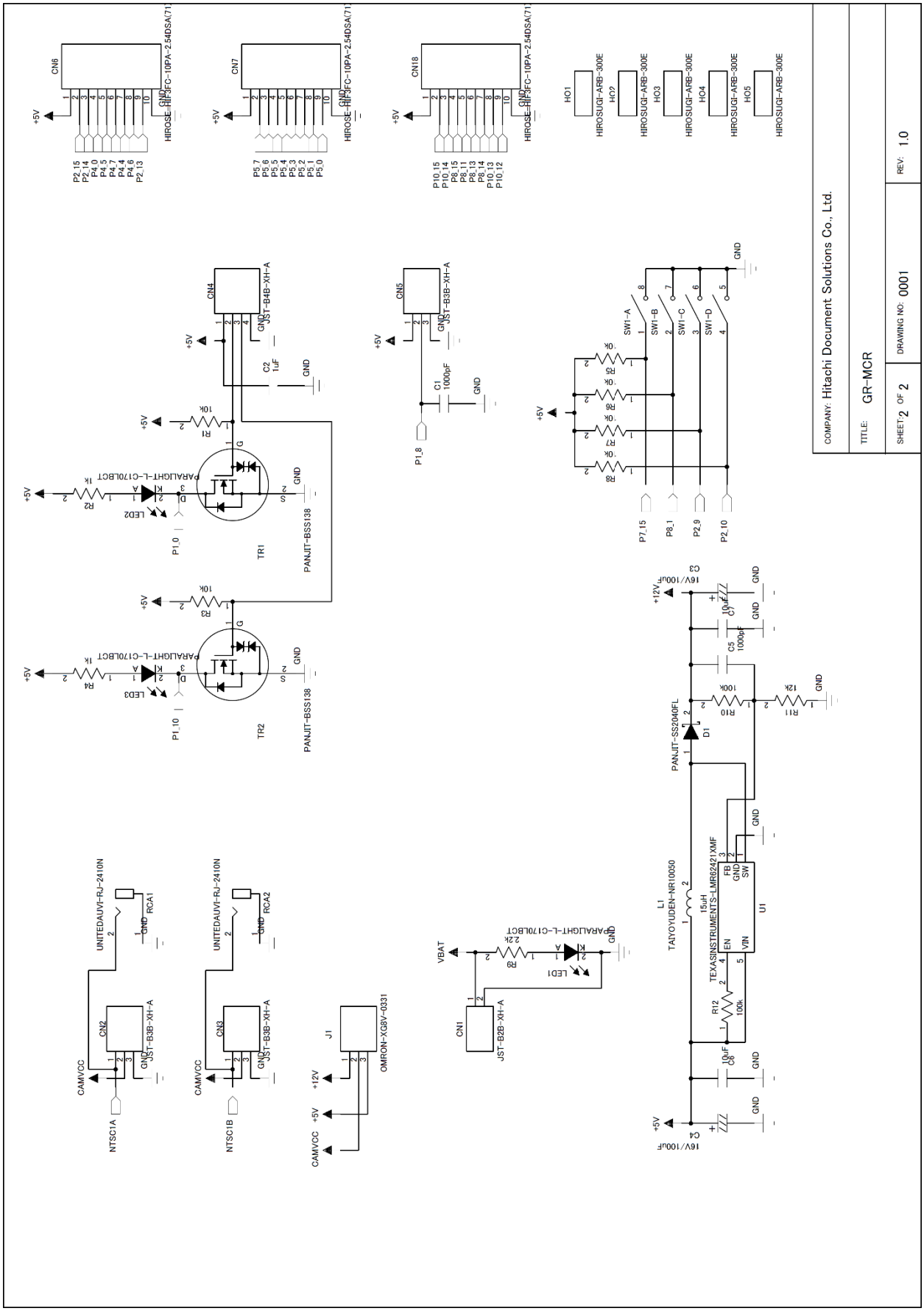
3.1 仕様

GR-MCR 基板 Rev.1.0 の仕様を下表に示します。

	GR-MCR 基板 Rev.1.0
部品数	リード線のある部品:約 21 個 面実装部品:約 28 個
GR-PEACH ボードとの接続方法	本基板の上に重ねる
入力電圧	DC5.0V±10%
カメラ入力回路	2 個
ボリューム入力回路	1 個
エンコーダ入力回路	1 個
カメラ用電源	5V or 12V(J1 のジャンパーピンで切り替え)
ディップスイッチ(4bit)	1 個
拡張コネクタ	10 ピンコネクタを 3 個搭載
基板外形	幅 95×奥行き 65×厚さ 1.2mm

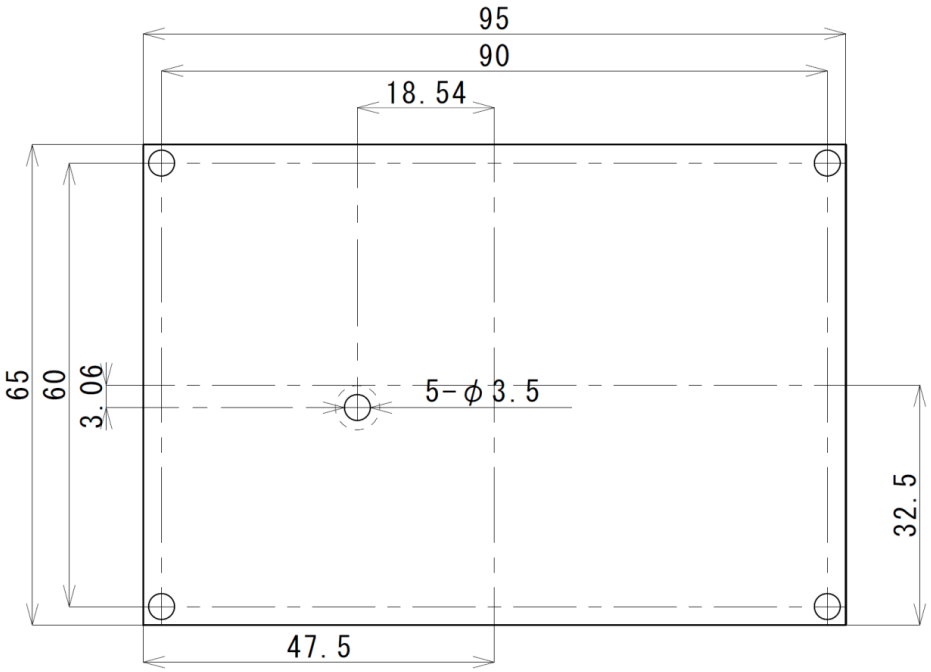
3.2 回路図


COMPANY: Hitachi Document Solutions Co., Ltd.
TITLE: GR-MCR
SHEET: 1 OF 2
DRAWING NO: 0001
REV: 1.0
CN8
CN9
CN10
CN11
CN12
CN13
CN14
CN15
CN16
CN17
USECONNELECTRONICS-FH-1X10SG
USECONNELECTRONICS-FH-1X8SG
USECONNELECTRONICS-FH-1X5SG
AVDD
VBAT
+5V
RESET
GND
P4_0
P2_13
P5_7
P5_6
P5_5
P5_4
P5_3
P5_2
P5_1
P5_0
P1_2
P1_3
P10_12
P10_15
P10_14
P10_13
P8_14
P8_15
P2_7
P2_6
P2_5
P2_4
P2_3
P2_2
P2_1
P2_0
P8_11
P8_13
P4_4
P4_5
P4_6
P4_7
P2_14
P2_15
P10_0
P1_1
P1_0
P1_12
P1_7
P1_6
P7_15
P8_1
P2_9
P2_10
NTSC1B
NTSC1A
P1_8
P1_9
P1_10
P1_11
P1_13
P1_15
P3_15
P3_14
P3_13
P3_12
P3_11
P3_10
P3_9
P3_8
NTSC2B
NTSC2A
VBUS
DM
DP

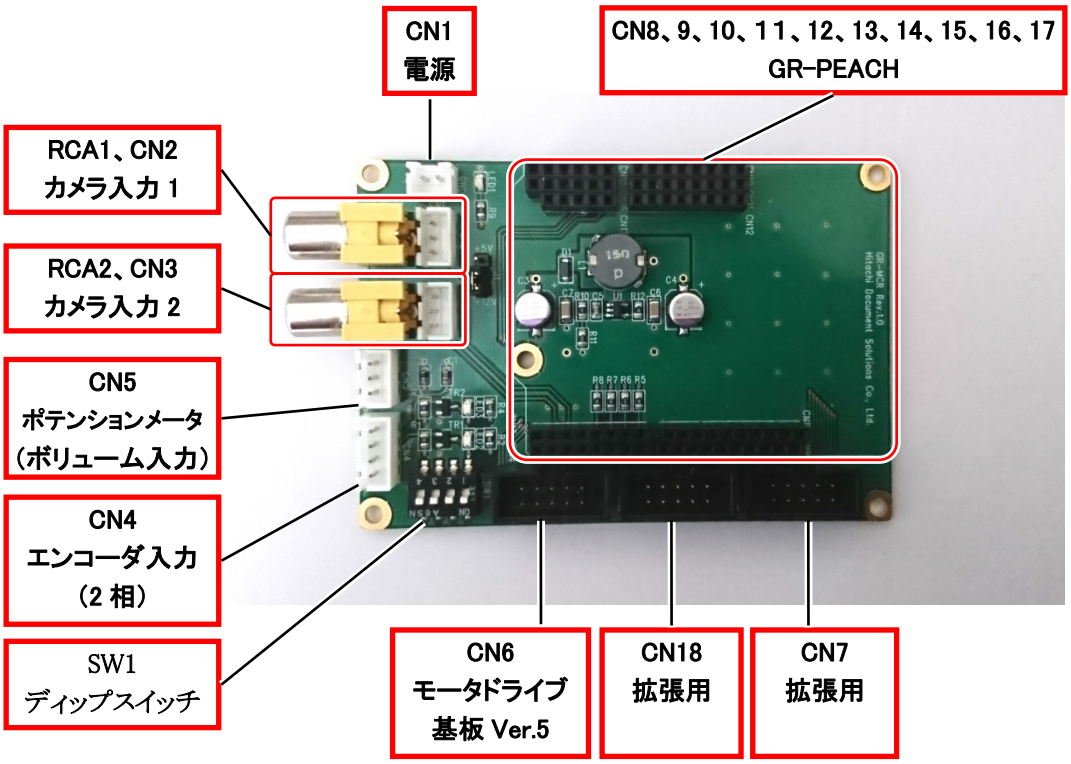

COMPANY: Hitachi Document Solutions Co., Ltd.
TITLE: GR-MCR
SHEET 2 OF 2
DRAWING NO: 0001
REV: 1.0
CN6
CN7
CN18
HIROSE-HIF3FC-10PA-2.54DSA(71)
HO1
HO2
HO3
HO4
HO5
HIROSUGI-ARB-300E
CN4
CN5
JST-B4B-XH-A
JST-B3B-XH-A
TR1
TR2
PANJIT-BSS138
LED2
LED3
PARALIGHT-L-C170LBGT
CN2
CN3
CN1
JST-B2B-XH-A
UNITEDAUVI-RJ-2410N
RCA1
RCA2
NTSC1A
NTSC1B
CAMVCC
J1
OMRON-XG8V-0331
LED1
VBAT
L1
TAIYOYUDEN-NR10050
U1
TEXASINSTRUMENTS-LMR62421XMF
D1
PANJIT-SS2040FL
SW1-A
SW1-B
SW1-C
SW1-D
P7_15
P8_1
P2_9
P2_10
P1_0
P1_10
P1_8
+5V
+12V
GND

3.3 寸法

GR-MCR 基板 Rev.1.0 には、取り付け用の穴が 5 個あります。この穴を使って、GR-MCR 基板 Rev.1.0 を固定してください。



3.4 外観


CN1
電源
CN8、9、10、11、12、13、14、15、16、17
GR-PEACH
RCA1、CN2
カメラ入力 1
RCA2、CN3
カメラ入力 2
CN5
ポテンションメータ
（ボリューム入力）
CN4
エンコーダ入力
（2 相）
SW1
ディップスイッチ
CN6
モータドライブ
基板 Ver.5
CN18
拡張用
CN7
拡張用

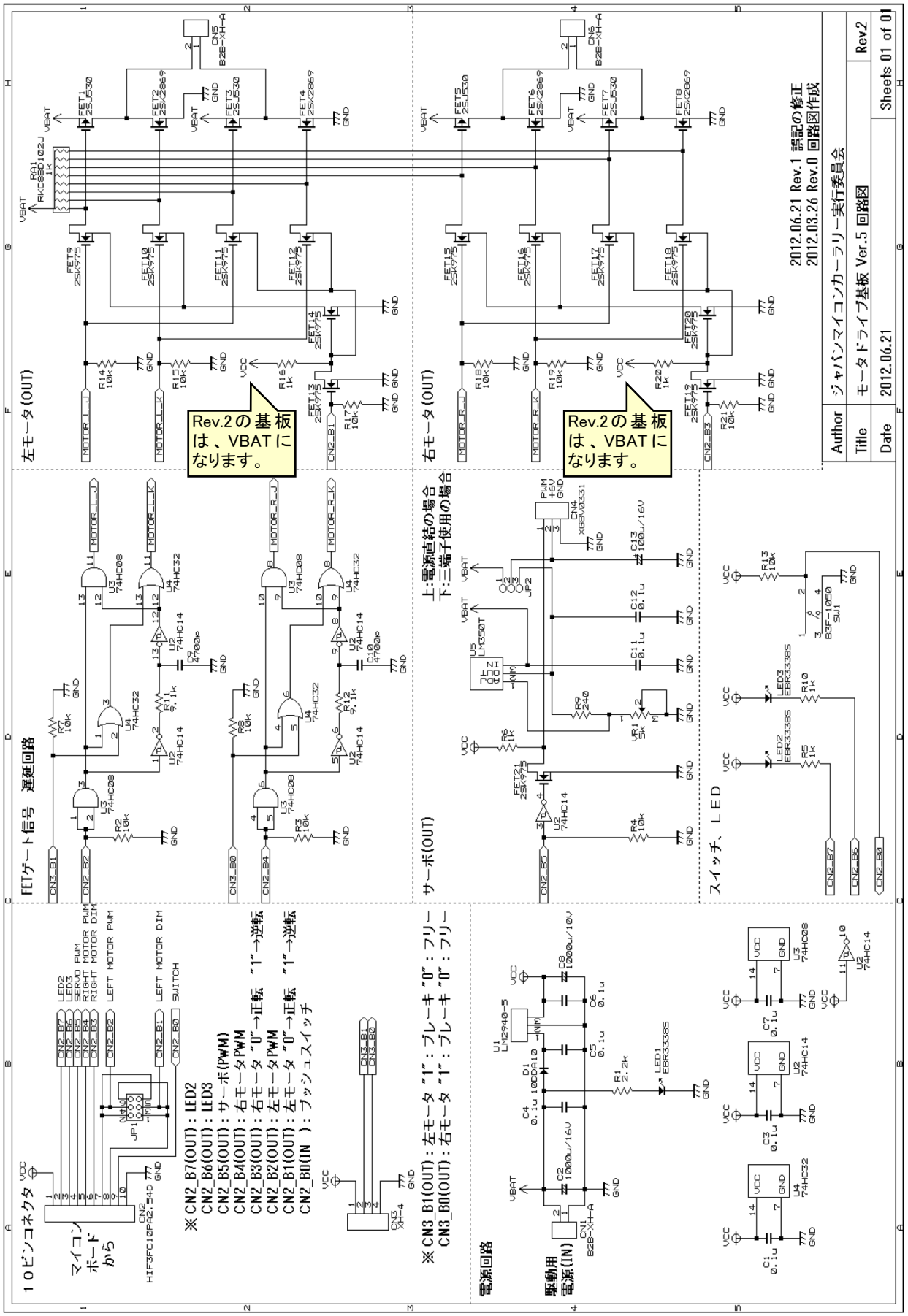
4. モータドライブ基板 Ver.5

4.1 仕様

モータドライブ基板 Ver.5 の仕様を、下表に示します。

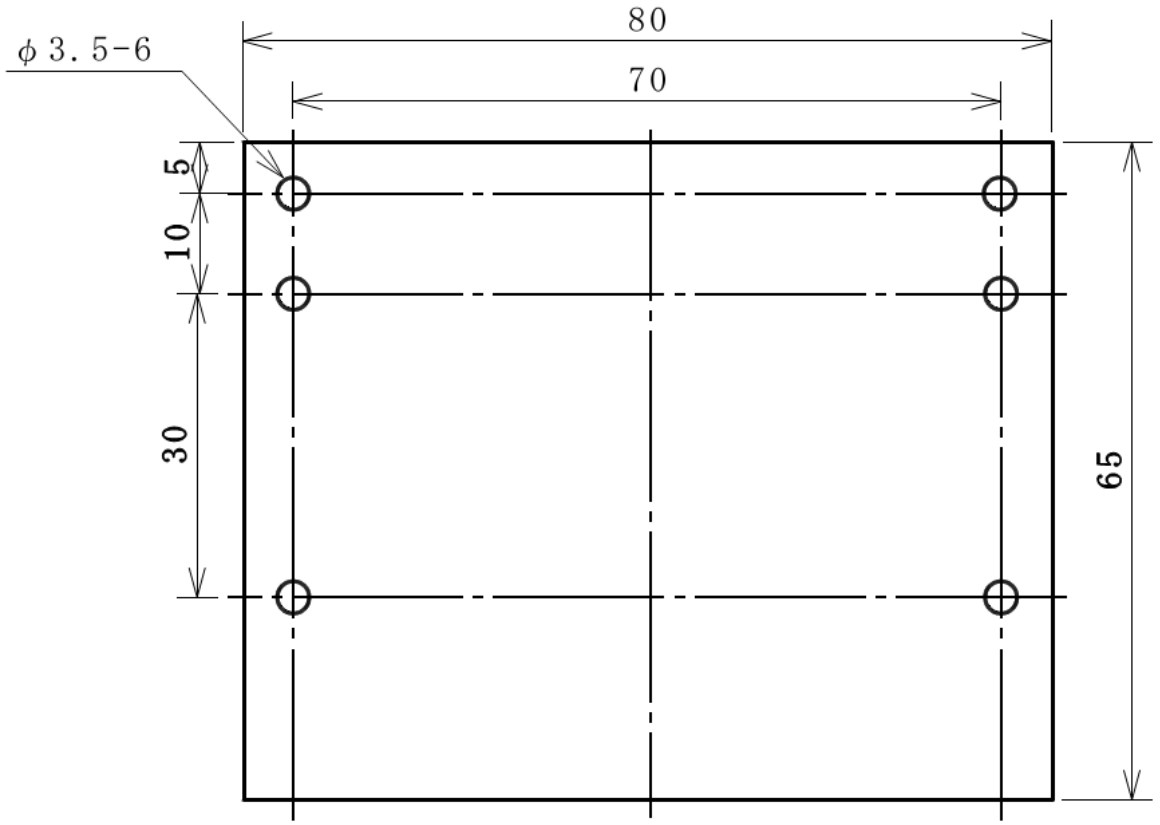
	モータドライブ基板 Ver.5
略称	ドライブ基板 5
部品数	リード線のある部品:約 66 個 部品のピンの間隔は 2.54mm 以上
RY_R8C38 ボード、 または、 GR-MCR 基板 Rev.1.0 (GR-PEACH) との接続方法	ポート 2 と接続
RY3048Fone ボードと の接続方法	ポート B と接続 **ただし、JP1 のパターンカット、ショートの加工あり**
制御できるモータ	2 個(左モータ、右モータ)
制御できるサーボ	1 個
プログラムで 点灯、消灯できる LED	2 個
プッシュスイッチ	1 個
制御系電圧 (CN2 に加えることの 出来る電圧)	DC5.0V±10%
駆動系電圧 (CN1 に加えることの 出来る電圧)	4.5～5.5V、または 7～15V ただし 7V 以上の場合、「LM350 追加セット」 によりマイコンボードに加える電圧を 5V、 サーボに加える電圧を 6V にする必要あり
サーボ、モータ 制御周期	モータ:16ms サーボ:16ms 個別設定不可
モータのフリー制御	フリー追加セットの追加で対応 ※モータの停止にはブレーキとフリーが あります。詳しくは、「フリー追加セット」 部分を参照してください。
基板外形	幅 80×奥行き 65×厚さ 1.6mm
完成時の寸法(実寸)	幅 80×奥行き 65×高さ 20mm
重量	約 35g ※リード線の長さや半田の量で変わります
標準プログラム	RZ/A1H マイコン:kit20_gr-peach ※

4.2 回路図


左モータ(OUT)
右モータ(OUT)
FETゲート信号 遅延回路
サーボ(OUT)
スイッチ、LED
電源回路
１０ピンコネクタ
マイコンボードから
Rev.2の基板は、VBATになります。
Rev.2の基板は、VBATになります。
上:電源直結の場合
下:三端子使用の場合
※ CN2_B7(OUT) : LED2
CN2_B6(OUT) : LED3
CN2_B5(OUT) : サーボ(PWM)
CN2_B4(OUT) : 右モータPWM
CN2_B3(OUT) : 右モータ "0"→正転 "1"→逆転
CN2_B2(OUT) : 左モータPWM
CN2_B1(OUT) : 左モータ "0"→正転 "1"→逆転
CN2_B0(IN) : プッシュスイッチ
※ CN3_B1(OUT) : 左モータ "1" : ブレーキ "0" : フリー
CN3_B0(OUT) : 右モータ "1" : ブレーキ "0" : フリー
駆動用電源(IN)
2012.06.21 Rev.1 誤記の修正
2012.03.26 Rev.0 回路図作成
Author ジャパンマイコンカーラリー実行委員会
Title モータドライブ基板 Ver.5 回路図
Rev2
Date 2012.06.21
Sheets 01 of 01

4.3 寸法

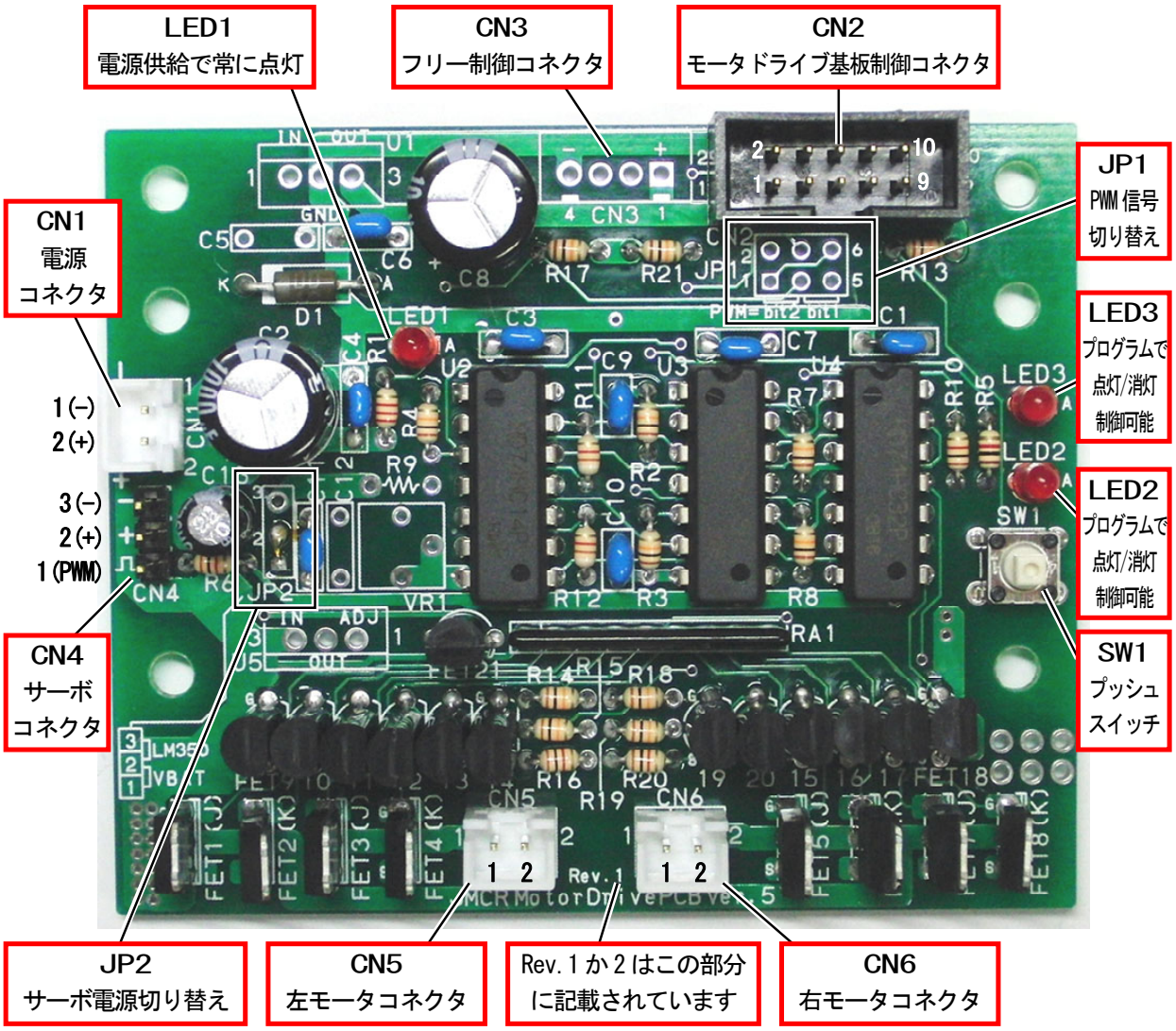
モータドライブ基板 Ver.5 には、取り付け用の穴が 6 個あります。この穴を使って、モータドライブ基板 Ver.5 をマイコンカーキットに固定してください。



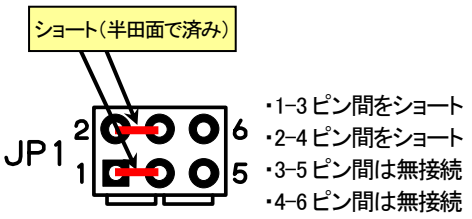
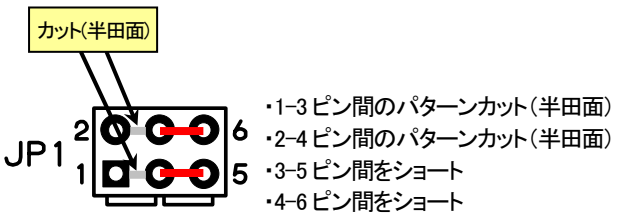
モータドライブ基板 Vol.3、Ver.4、Ver.5 は同じ外形です。

4.4 外観

モータドライブ基板 Ver.5 の外観を、下図に示します。



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

部品番号	接続先	pin	詳細
CN1	電源入力	1	GND
		2	＋電源（4.5～5.5V、または 7V～15V 入力） ※7V 以上の場合、別売りの「LM350 追加セット」の部品を取り付ける必要があります。
CN2	マイコンボードと接続	1～10	次項参照
CN3	マイコンボードと接続	1	+5V
		2	左モータの停止状態選択 1:フリー 0:ブレーキ
		3	右モータの停止状態選択 1:フリー 0:ブレーキ
		4	GND
CN4	サーボ	1	サーボ PWM 信号出力
		2	サーボ電源（6V 出力）
		3	GND
CN5	左モータ	1、2	左モータ出力
CN6	右モータ	1、2	右モータ出力
JP1	左モータの PWM 信号切り替え	1～6	左モータの PWM 出力端子、方向切り替え端子を切り替えるジャンパです。 ●RY_R8C38 ボード、GR-MCR 基板 Rev.1.0（GR-PEACH）の場合  ※半田面でショート済みです。特に何もする必要はありません。 ●RY3048Fone ボードの場合 

JP2	サーボ電源切り替え	1～3	JP2 は、サーボ電源ピン(CN2 の 2 ピン)への電源供給元を切り替える端子です。 ●CN1 に供給されている電源電圧が 6V 以下の場合 1-2 ピン間をショートさせます。CN1 の電源が直接、CN4 の 2 ピンに接続されます。 ●CN1 に供給されている電源電圧が 6V 以上の場合 サーボに加えることのできる電圧を超えていますので、別売りの「LM350 追加セット」の部品を取り付けて、2-3 ピン間をショートさせます。LM350(三端子レギュレータ)を通して、6V の電圧が CN4 の 2 ピンに供給されます。

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

GR-PEACH ボードとモータドライブ基板 Ver.5 の接続を下表に示します。
GR-PEACH ボードに接続されている GR-PEACH シールドとモータドライブ基板をフラットケーブルで接続すると、サーボ端子が P4_0 と接続されます。MTU2_0 による PWM モード 1 を使用してこの端子から PWM 波形を出力、サーボを制御します。

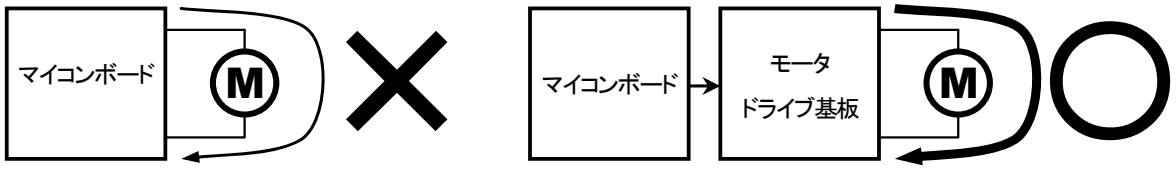
モータドライブ基板 Ver.5			GR-MCR 基板 Rev.1.0(GR-PEACH) CN4			
ピン番号	信号名		ピン番号	信号名	入出力設定	説明
1	VCC(+5V)	―	1	VCC(+5V)		
2	LED2	―	2	P2_15	出力	P2_15 は通常の I/O ポートです。
3	LED3	―	3	P2_14	出力	P2_14 は通常の I/O ポートです。
4	サーボ	―	4	P4_0(TIOC0A)	出力(PWM 波形出力)	この端子は PWM 出力許可にします。MTU2TGRD_0 で ON 幅を設定します。
5	右モータ PWM	―	5	P4_5	出力(PWM 波形出力)	この端子は PWM 出力許可にします。MTU2TGRD_4 で ON 幅を設定します。
6	右モータ回転方向	―	6	P4_7	出力	P4_7 は通常の I/O ポートです。
7	左モータ PWM	―	7	P4_4	出力(PWM 波形出力)	この端子は PWM 出力許可にします。MTU2TGRC_4 で ON 幅を設定します。
8	左モータ回転方向	―	8	P4_6	出力	P4_6 は通常の I/O ポートです。
9	プッシュスイッチ	―	9	P2_13	入力	プッシュスイッチの状態を入力します。
10	GND	―	10	GND		

↑
フラットケーブルの結線

4.6 モータ制御

4.6.1 モータドライブ基板の役割

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



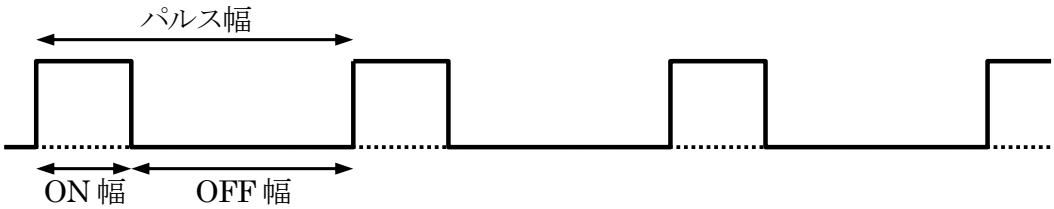
信号が弱いため、モータは回らない　　大電流に変換、モータ駆動！

4.6.2 スピード制御の原理

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

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

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



デューティ比の計算式を下記に示します。

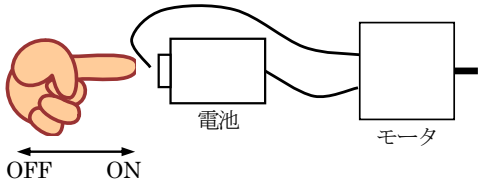
デューティ比＝ON 幅／パルス幅(ON幅＋OFF幅)

例えば、100ms のパルスに対して、ON 幅が 60ms なら、

デューティ比＝60ms/100ms＝0.6＝60%

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

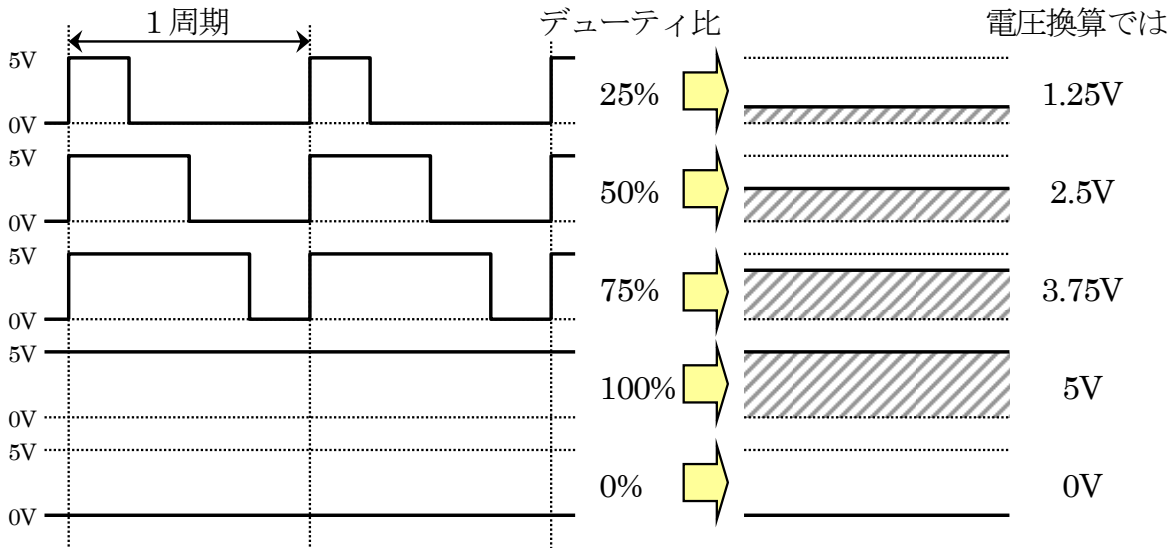
「PWM」と聞くと、何か難しく感じてしまいますが、下記のように手でモータと電池の線を「繋ぐ」、「離す」の繰り返し、それも PWM と言えます。繋いでいる時間が長いとモータは速く回ります。離している時間が長いとモータは少ししか回りません。人なら「繋ぐ」、「離す」動作をコンマ数秒でしか行えませんがマイコンなら数ミリ秒で行えます。



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

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

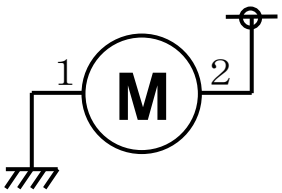
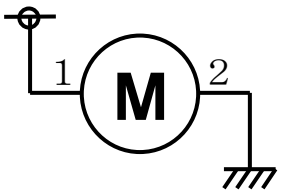
この信号をモータに接続すれば、モータの回転スピードも少しずつ変化させることができ、微妙なスピード制御が可能です。LEDに接続すれば、LEDの明るさを変えることができます。マイコンを使えばこの作業をマイクロ秒、ミリ秒単位で行うことができます。このオーダでの制御になると、非常にスムーズなモータ制御が可能です。



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

4.6.3 正転、逆転の原理

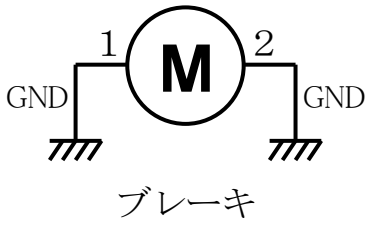
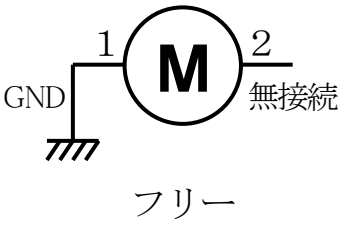
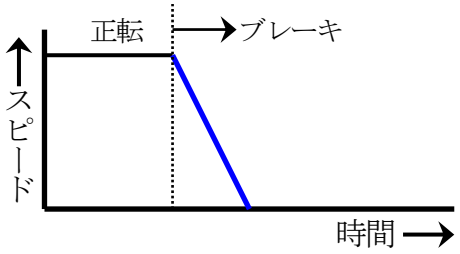
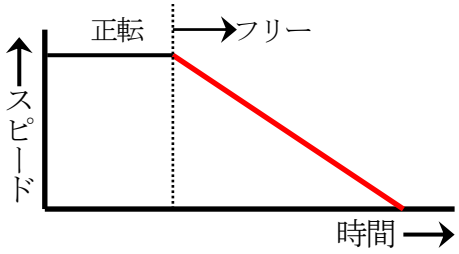
モータドライブ基板 Ver.5 では、モータを「正転、逆転、停止」制御することができます。正転、逆転させるとき、モータの端子に加える電圧を下表に示します。

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

4.6.4 ブレーキとフリー

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

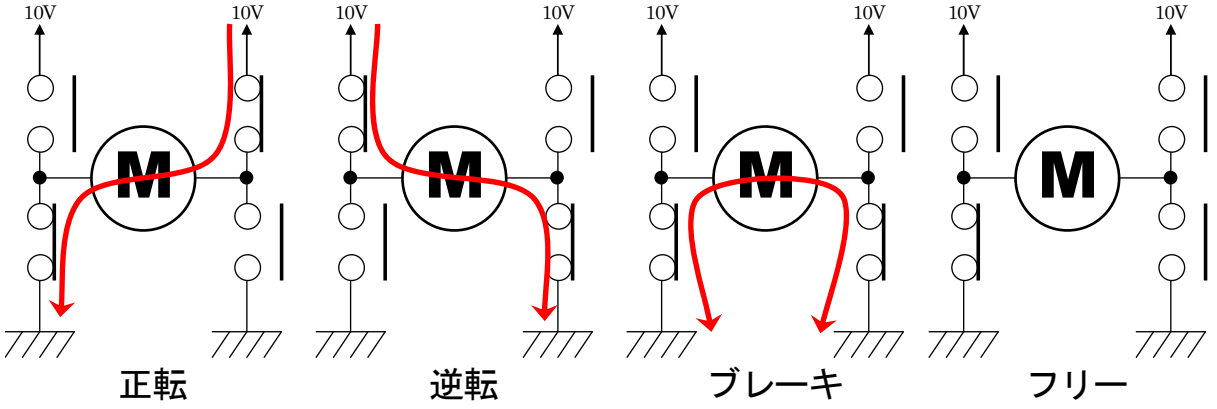
ブレーキとフリーの違いを、下表に示します。

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

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

4.6.5 Hブリッジ回路

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

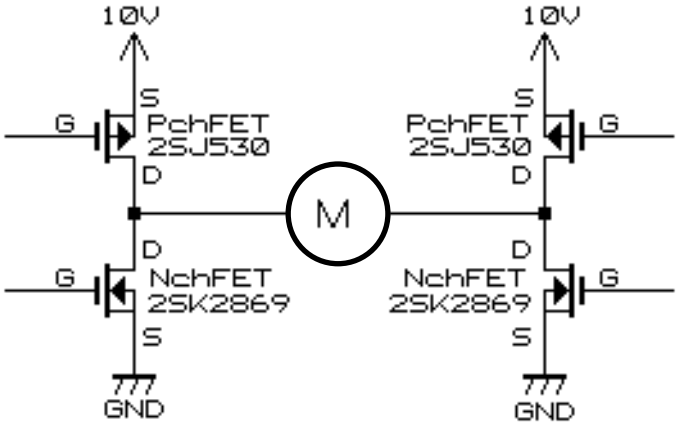


4.6.6 Hブリッジ回路のスイッチをFETにする

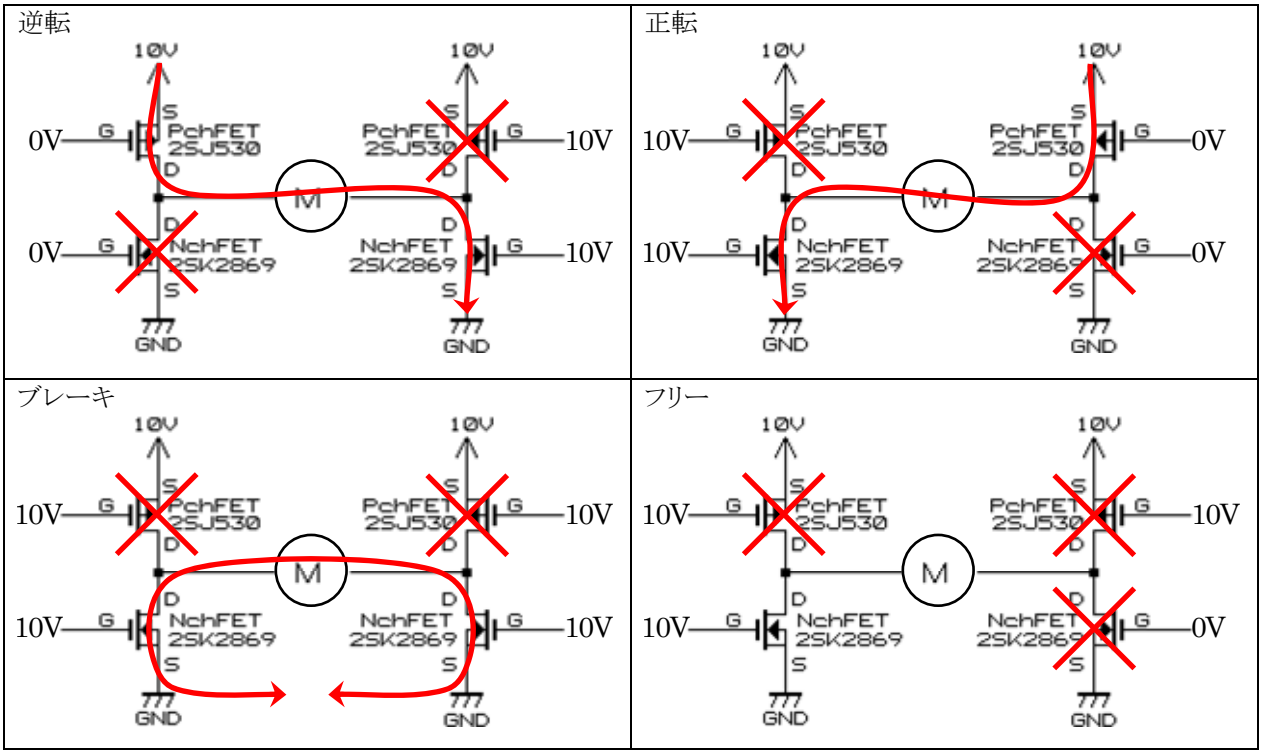
スイッチ部分を FET にします。電源のプラス側に P チャネル FET（2SJ タイプ）、マイナス側に N チャネル FET（2SK タイプ）を使用します。

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

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

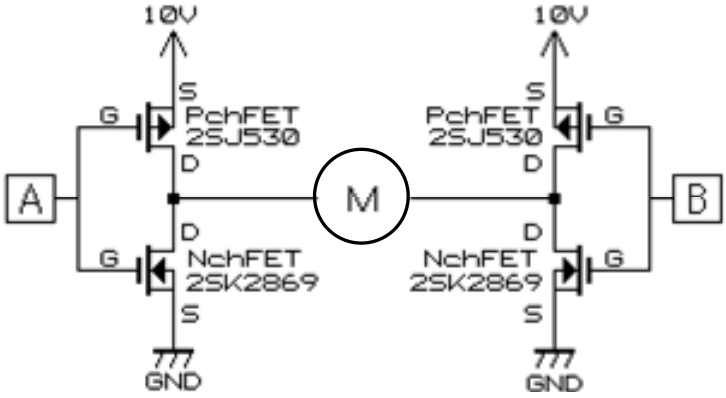


これら 4 つの FET のゲートに加える電圧を変えることにより、正転、逆転、ブレーキの動作を行います。



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

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



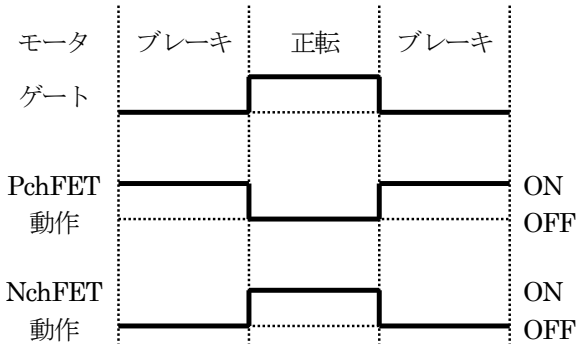
A	B	動作
0V	0V	ブレーキ
0V	10V	逆転
10V	0V	正転
10V	10V	ブレーキ

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

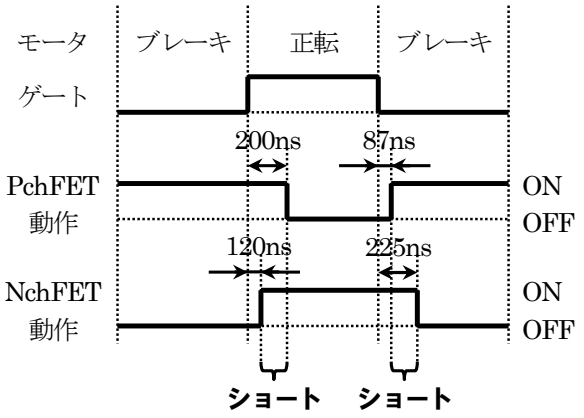
この回路を実際に組んで PWM 波形を加え動作させると、FET が非常に熱くなりました。どうしてでしょうか。

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

理想的な波形



実際の波形



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

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

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

2SJ530(P チャネル)

電気的特性

(Ta=25℃)

項目	記号	Min	Typ	Max	単位	測定条件
ドレイン・ソース破壊電圧	$V_{(BR)DSS}$	-60	—	—	V	I_D =-10mA, V_{GS} = 0
ゲート・ソース破壊電圧	$V_{(BR)GSS}$	±20	—	—	V	I_G = ±100μA, V_{DS} = 0
ドレイン遮断電流	I_{DSS}	—	—	-10	μA	V_{DS} =-60V, V_{GS} = 0
ゲート遮断電流	I_{GSS}	—	—	±10	μA	V_{GS} = ±16V, V_{DS} = 0
ゲート・ソース遮断電圧	$V_{GS(off)}$	-1.0	—	-2.0	V	V_{DS} =-10V, I_D=-1mA
順伝達アドミタンス	$\|y_{fs}\|$	6.5	11	—	S	I_D=-8A,V_{DS}=-10V 注4
ドレイン・ソースオン抵抗	$R_{DS(on)}$	—	0.08	0.10	Ω	I_D=-8A, V_{GS}=-10V 注4
ドレイン・ソースオン抵抗	$R_{DS(on)}$	—	0.11	0.16	Ω	I_D=-8A, V_{GS} =-4V 注4
入力容量	Ciss	—	850	—	pF	V_{DS} =-10V, V_{GS} = 0
出力容量	Coss	—	420	—	pF	f = 1MHz
帰還容量	Crss	—	110	—	pF	
ターン・オン遅延時間	td(on)	—	12	—	ns	V_{GS} =-10V, I_D=-8A
上昇時間	tr	—	75	—	ns	R_L =3.75Ω
ターン・オフ遅延時間	td(off)	—	125	—	ns	
下降時間	tf	—	75	—	ns	
ダイオード順電圧	V_{DF}	—	-1.1	—	V	I_F =-15A, V_{GS} = 0
逆回復時間	trr	—	70	—	ns	I_F =-15A, V_{GS} = 0 diF/dt = 50A/μs

注) 4. パルス測定

OFF→ON は 87ns 遅れる

ON→OFF は 200ns 遅れる

2SK2869(Nチャネル)

電気的特性

(Ta=25℃)

項目	記号	Min	Typ	Max	単位	測定条件
ドレイン・ソース破壊電圧	$V_{(BR)DSS}$	60	—	—	V	I_D = 10mA, V_{GS} = 0
ゲート・ソース破壊電圧	$V_{(BR)GSS}$	±20	—	—	V	I_G = ±100μA, V_{DS} = 0
ドレイン遮断電流	I_{DSS}	—	—	10	μA	V_{DS} = 60V, V_{GS} = 0
ゲート遮断電流	I_{GSS}	—	—	±10	μA	V_{GS} = ±16V, V_{DS} = 0
ゲート・ソース遮断電圧	$V_{GS(off)}$	1.5	—	2.5	V	V_{DS} = 10V, I_D = 1mA
順伝達アドミタンス	$\|y_{fs}\|$	10	16	—	S	I_D = 10A, V_{DS} = 10V※1
ドレイン・ソースオン抵抗	$R_{DS(on)}$	—	0.033	0.045	Ω	I_D = 10A, V_{GS}= 10V※1
ドレイン・ソースオン抵抗	$R_{DS(on)}$	—	0.055	0.07	Ω	I_D = 10A, V_{GS} = 4V※1
入力容量	Ciss	—	740	—	pF	V_{DS} = 10V, V_{GS} = 0
出力容量	Coss	—	380	—	pF	f = 1MHz
帰還容量	Crss	—	140	—	pF	
ターン・オン遅延時間	td(on)	—	10	—	ns	V_{GS} = 10V, I_D= 10A
上昇時間	tr	—	110	—	ns	R_L = 3Ω
ターン・オフ遅延時間	td(off)	—	105	—	ns	
下降時間	tf	—	120	—	ns	
ダイオード順電圧	VDF	—	1.0	—	V	I_F = 20A, V_{GS} = 0
逆回復時間	trr	—	40	—	ns	I_F = 20A, V_{GS} = 0 diF/dt = 50A/μs

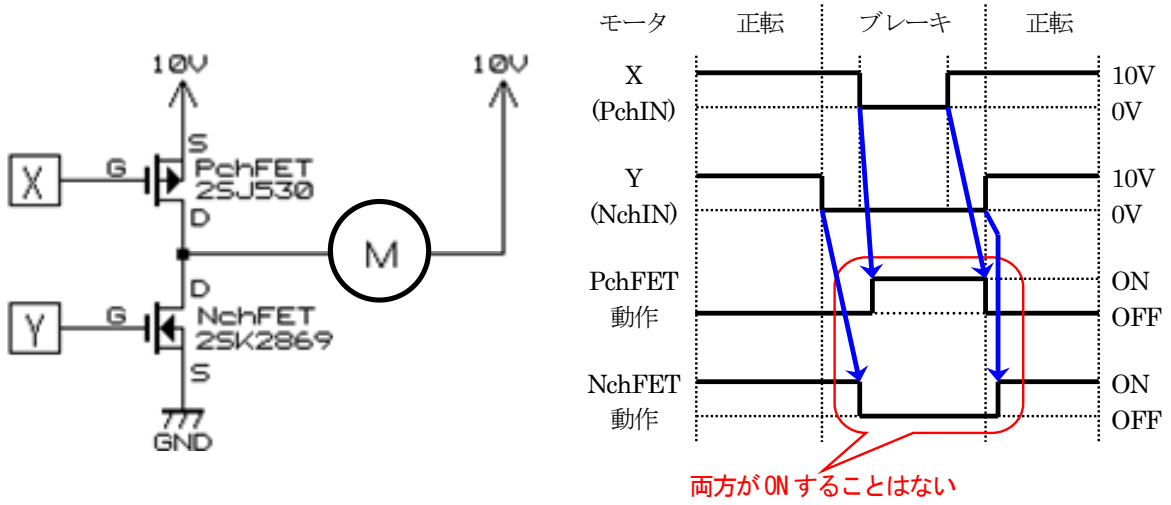
注) 1. パルス測定

OFF→ON は 120ns 遅れる

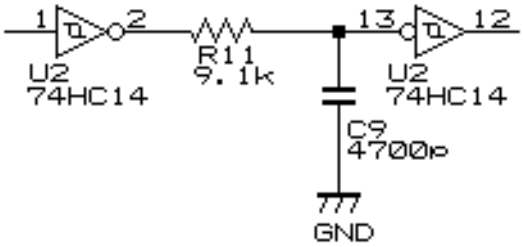
ON→OFF は 225ns 遅れる

4.6.7 PチャネルFETとNチャネルFETの短絡防止回路

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



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



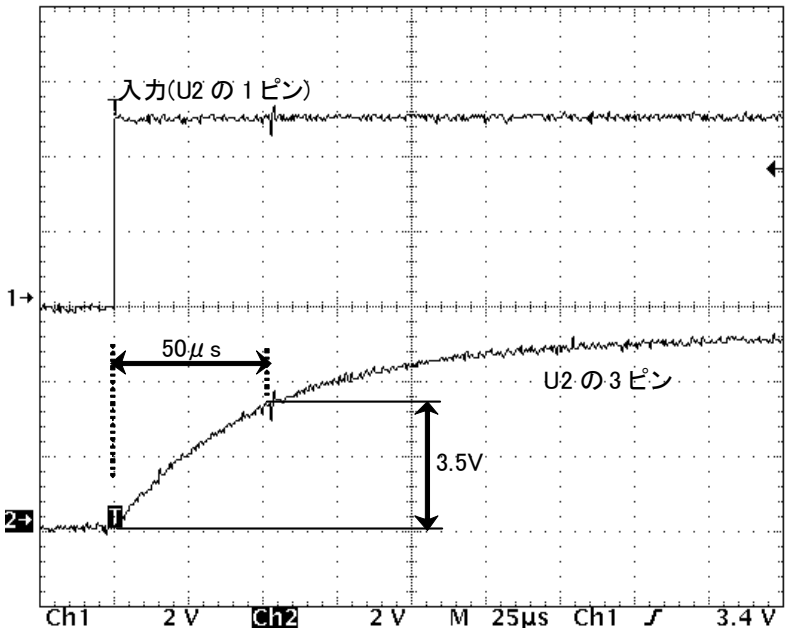
遅延時間は下記の式で計算することができます。

時定数T＝CR［s］

今回は9.1kΩ、4700pFなので、計算すると下記の時間になります。

$T=(9.1\times10^{3})\times(4700\times10^{-12})$

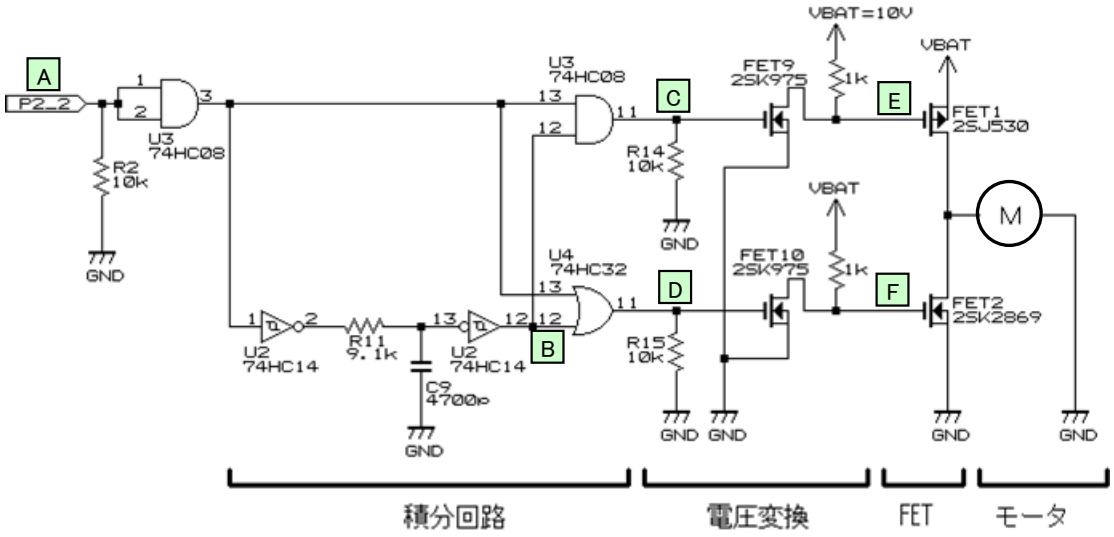
$=42.77[\mu s]$

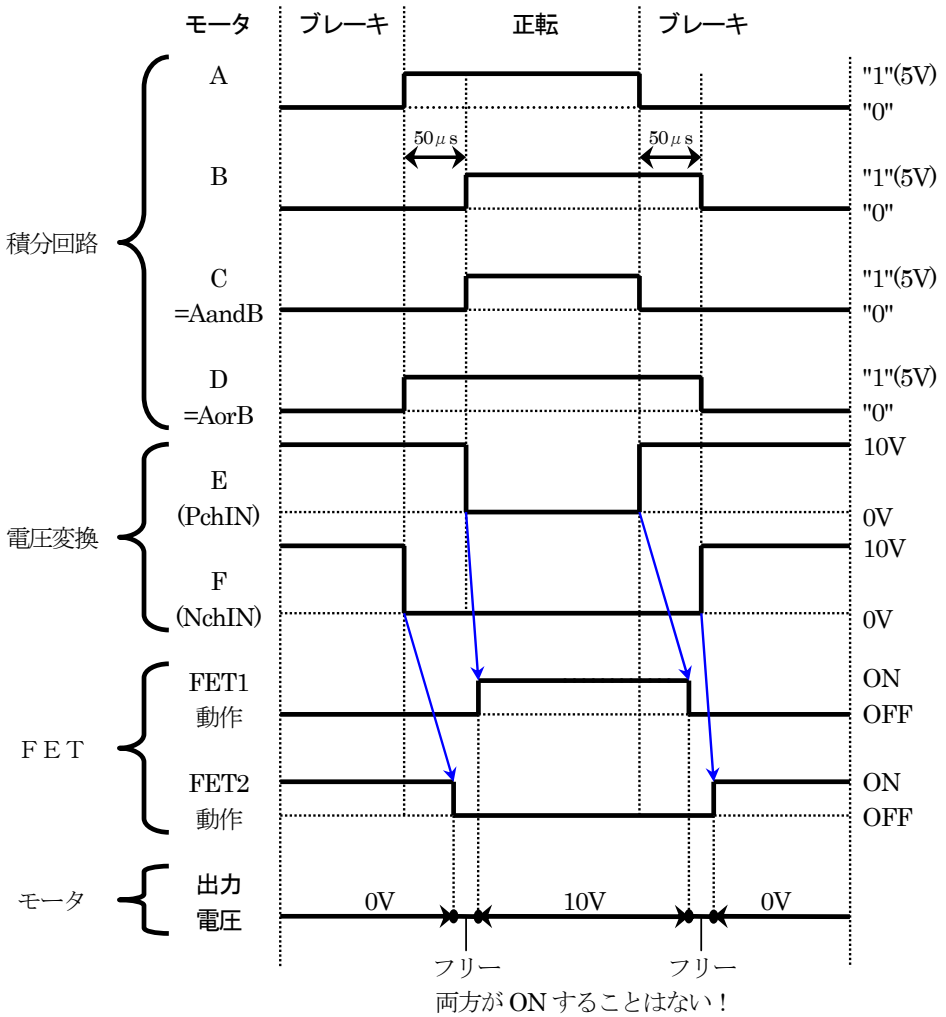


74HCシリーズは3.5V以上の入力電圧があると"1"とみなします。実際に波形を観測し、3.5Vになるまでの時間を計ると約50μsになりました。

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

積分回路とFETを合わせた回路は下記のようになります。





(1) ブレーキ→正転に変えるとき

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

(2) 正転→ブレーキに変えるとき

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

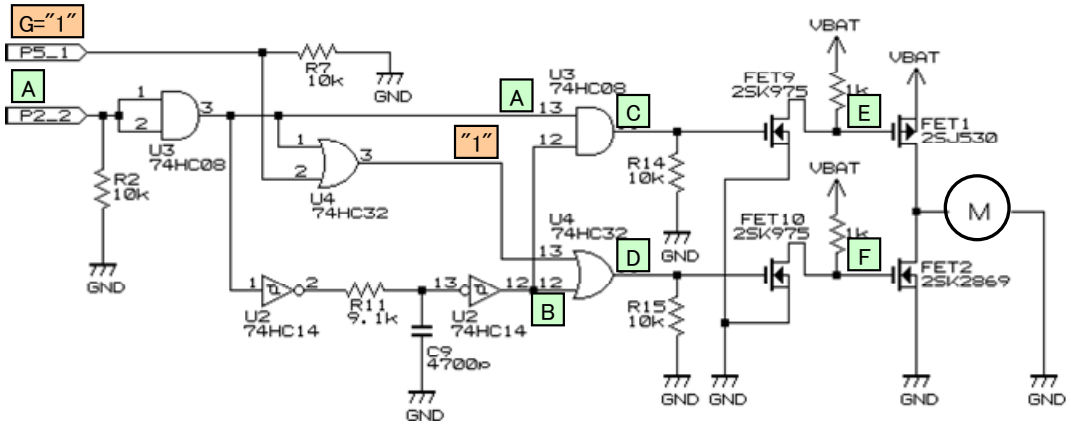
このように、動作を切り替えるときはいったん、両 FET ともフリー状態を作って、短絡するのを防いでいます。

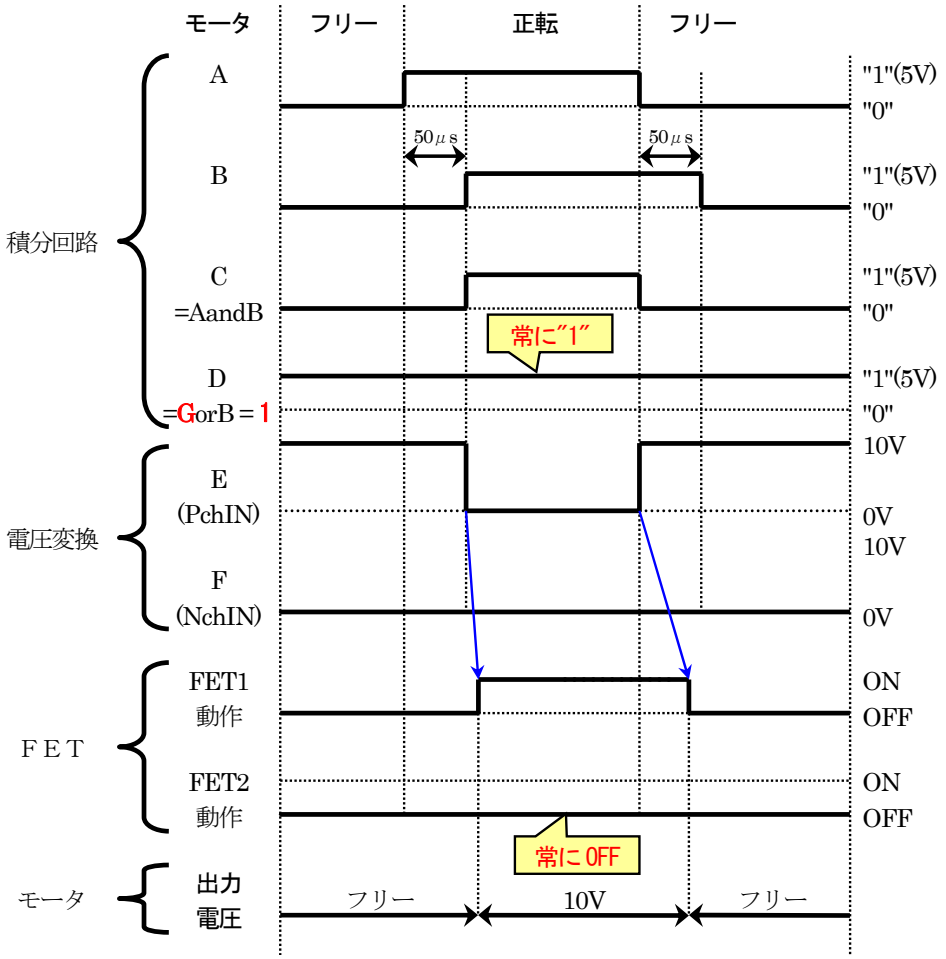
※ゲートに加える電圧の 10V は例です。実際は電源電圧(VBAT)になります。

4.6.8 フリー回路

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

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





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

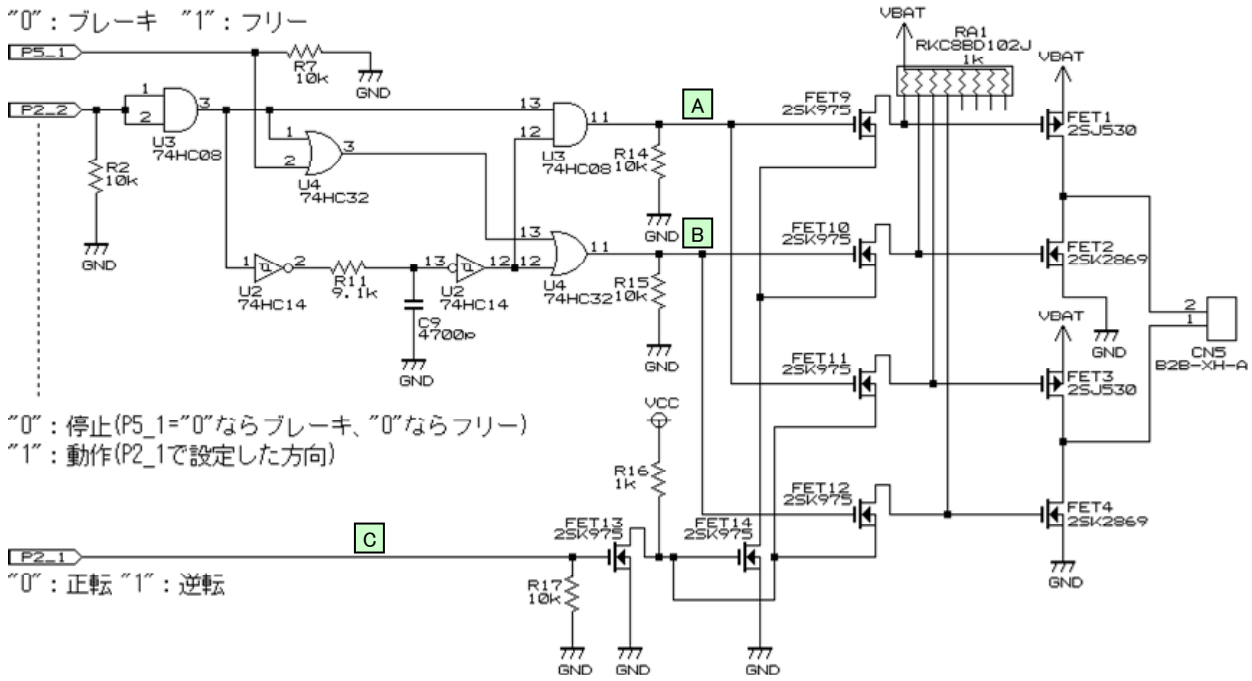
G 点が"0"のときは前記のとおり、正転とブレーキを繰り返します。

4.6.9 実際の回路

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

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

(1) 回路図



(2) 方向：正転、停止：ブレーキのときの信号のレベルとモータの動作

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

※A,B,C：″0″=0V、″1″=5V

(3) 方向:逆転、停止:ブレーキのときの信号のレベルとモータの動作

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

(4) 方向:正転、停止:フリーのときの信号のレベルとモータの動作

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

4.6.10 左モータの動作

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

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

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

4.6.11 右モータの動作

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

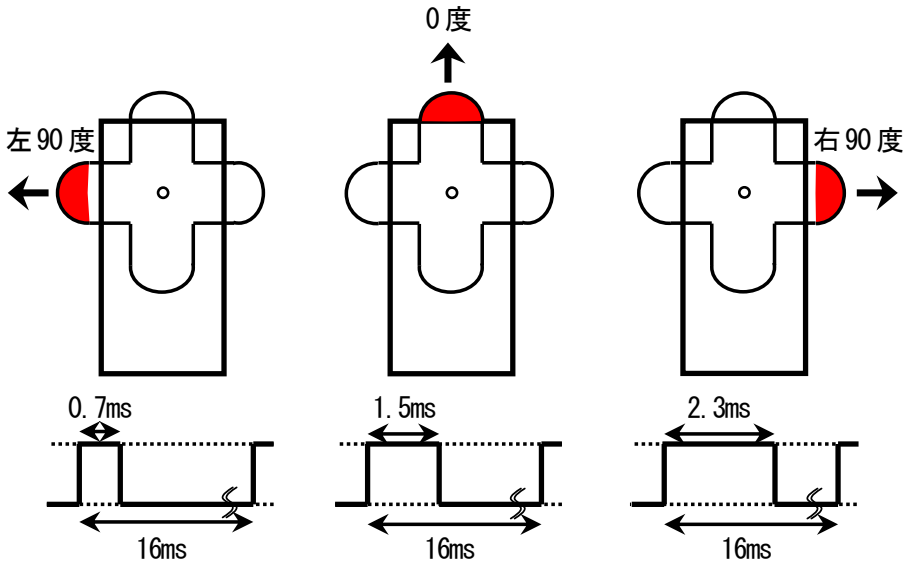
右モータ方向 P2_3	右モータ PWM P2_4	右モータ停止動作 P5_0	モータ動作
0	PWM	0	PWM="1"なら正転、"0"ならブレーキ
0	PWM	1	PWM="1"なら正転、"0"ならフリー
1	PWM	0	PWM="1"なら逆転、"0"ならブレーキ
1	PWM	1	PWM="1"なら逆転、"0"ならフリー

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

4.7 サーボ制御

4.7.1 原理

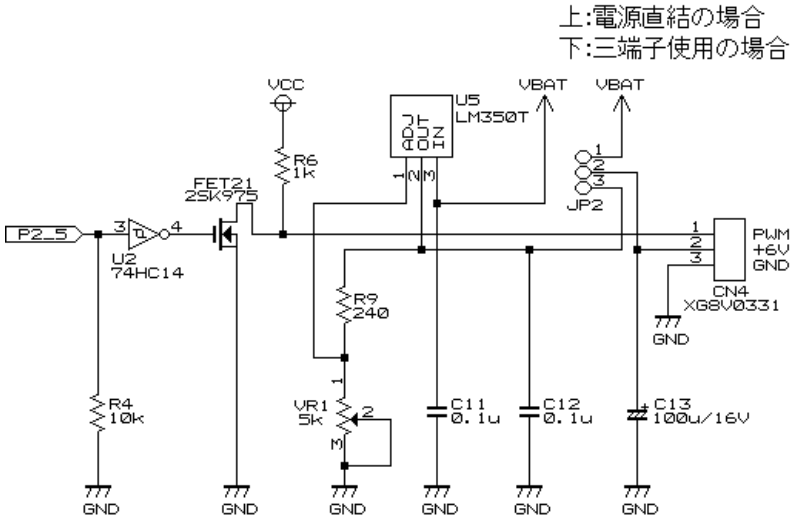
サーボは周期 16[ms]のパルスを加え、そのパルスの ON 幅でサーボの角度が決まります。
サーボの回転角度と ON のパルス幅の関係は、サーボのメーカや個体差によって多少の違いがありますが、ほとんどが下図のような関係になります。



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

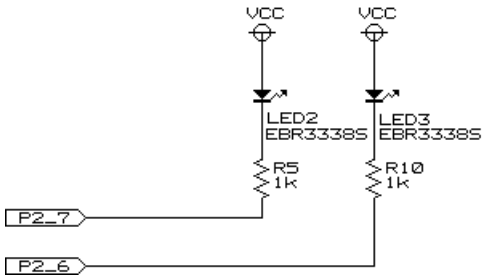
RZ/A1H マイコンの PWM モードで PWM 信号を生成して、サーボを制御します。

4.7.2 回路



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

4.8 LED 制御

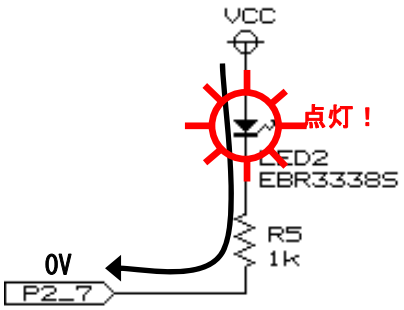
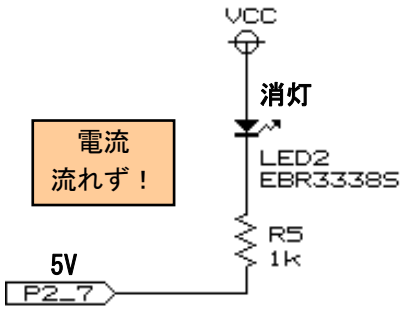


モータドライブ基板には 3 個の LED が付いています。そのうち、2 個がマイコンで ON／OFF 可能です。
LED のカソードは、マイコンのポートに直結されています。電流制限抵抗は、1kΩです。
EBR3338S は順電圧 1.7V、電流は 20mA 流すことができます。
電流制限抵抗は下記のようになります。

抵抗＝(電源電圧－LED に加える電圧)／LED に流したい電流
＝(5 － 1.7)／ 0.02
＝165Ω

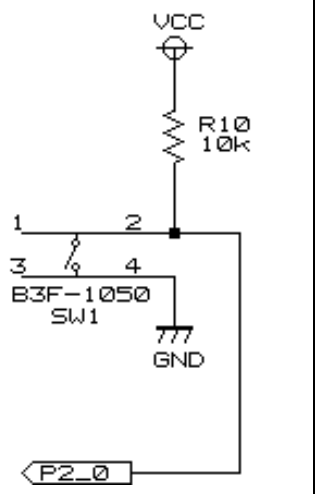
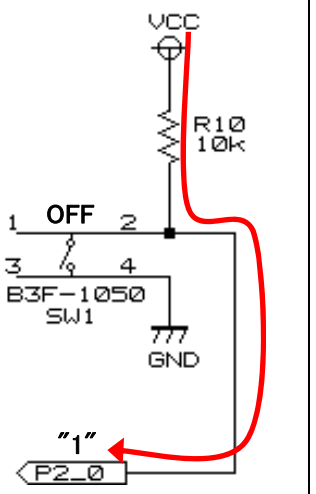
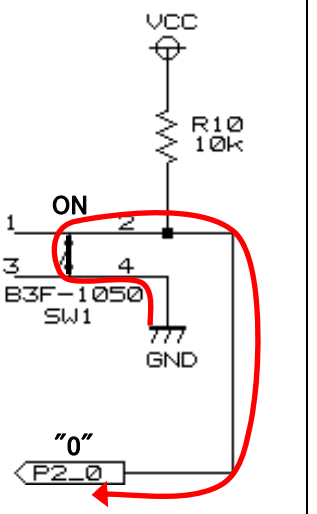
実際は、電池の消費電流を減らすのとポートの電流制限により、1kΩの抵抗を接続しています。電流は、下記のようになります。

電流＝ (電源電圧－LED に加える電圧)／抵抗
＝ (5 － 1.7)／1000 ＝ 3.3[mA]

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

4.9 スイッチ制御

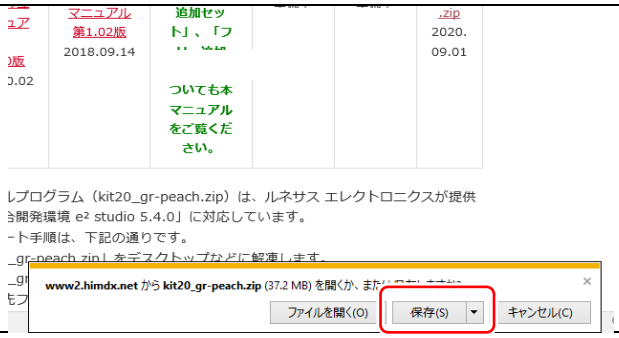
モータドライブ基板には、プッシュスイッチが1個付いています。

		
スイッチは、10k Ω でプルアップされ、ポート 2 の bit0 に繋がっています。	スイッチが押されていなければ、プルアップ抵抗を通して"1"が P2_0 に入力されます。	スイッチが押されると GND とつながり、"0"が P2_0 に入力されます。

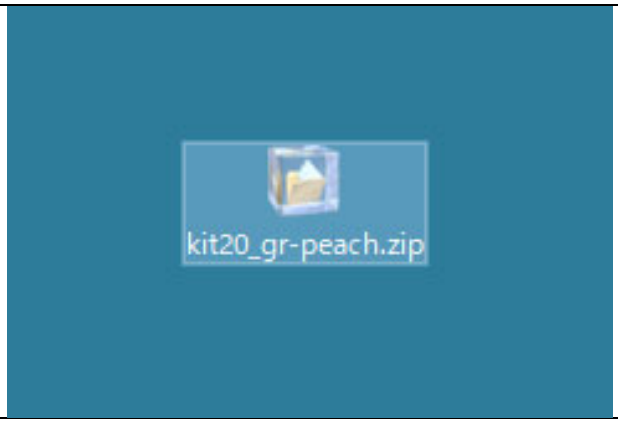
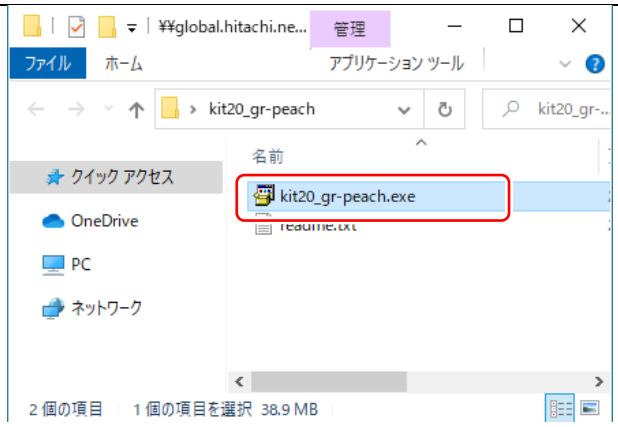
5. サンプルプログラムのダウンロード、インポート

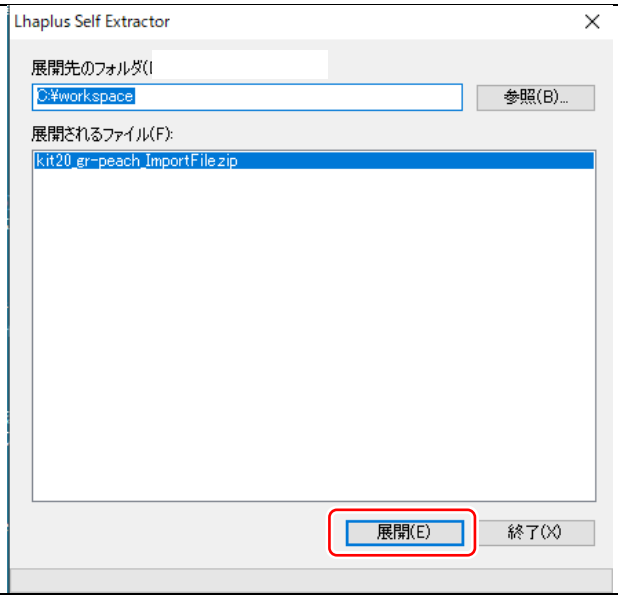
5.1 ダウンロード

1		マイコンカーラリー販売のホームページ(https://www2.himdx.net/mcr/index.asp)を開き、「ダウンロード」をクリックします。 ※または、下記のアドレスから直接ダウンロードできます。
2		「画像処理マイコンカーキットに関する資料」をクリックします。
3		「kit20_gr-peach.zip」をクリックします。

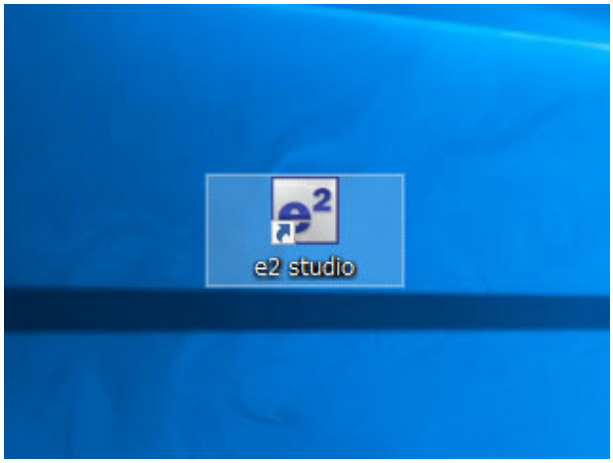
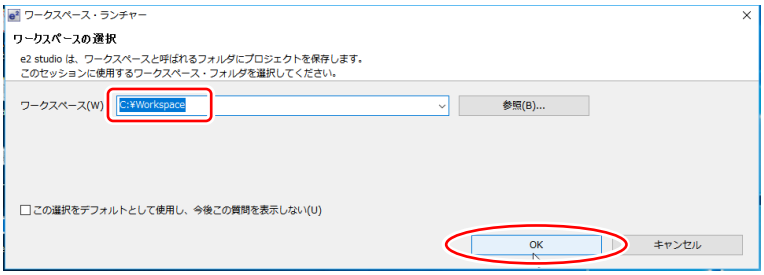
4		保存をクリックし、「kit20_gr-peach.zip」をダウンロードします。

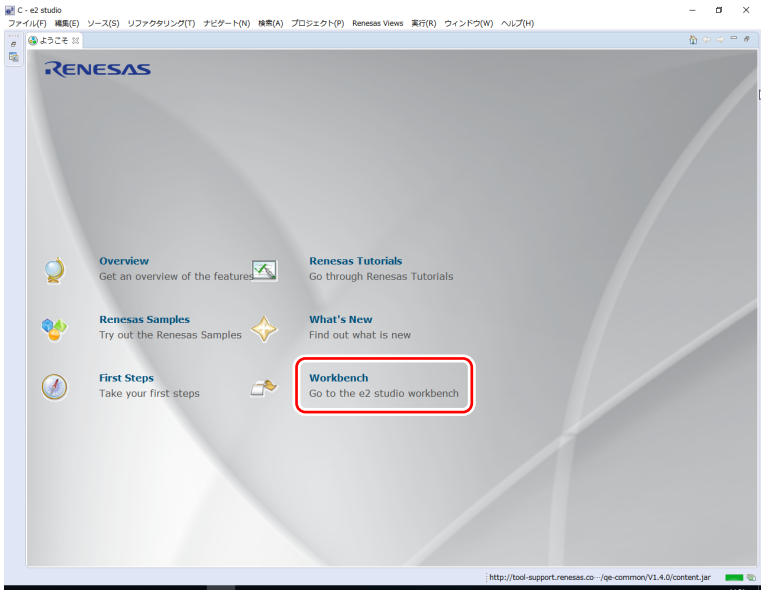
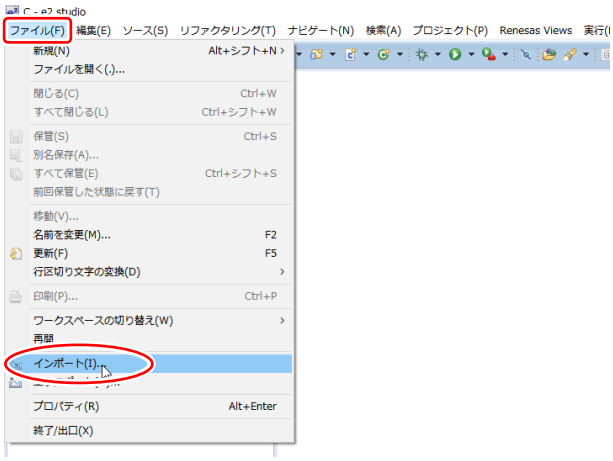
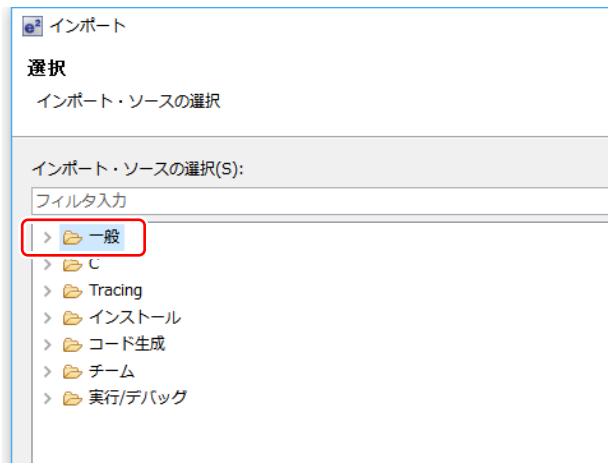
5.2 インストール

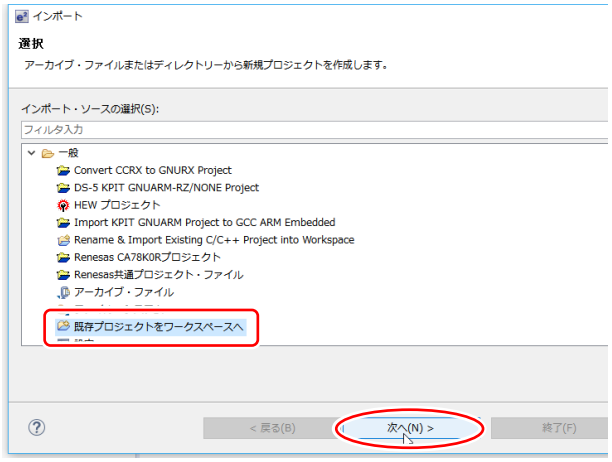
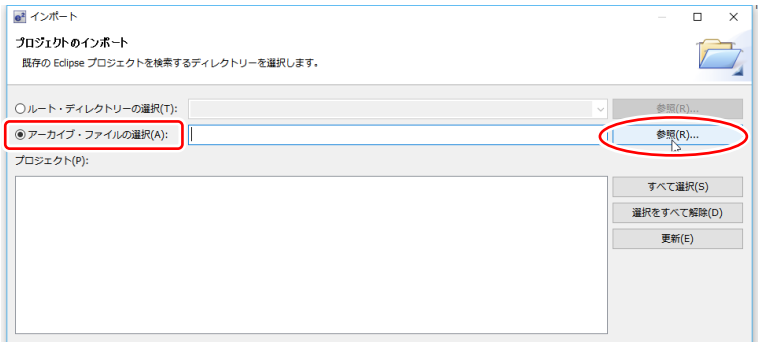
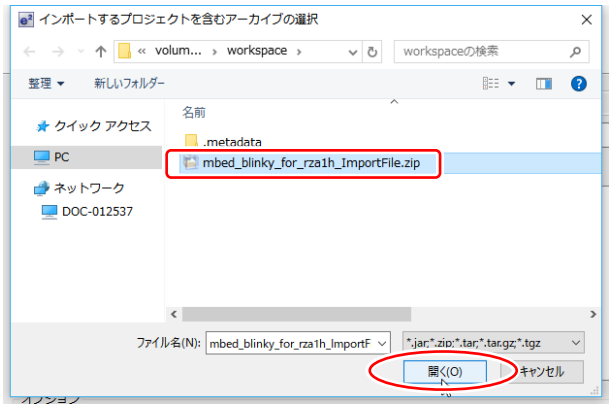
1		「kit20_gr-peach.zip」を任意の場所に解凍します。
2		「kit20_gr-peach.exe」を実行します。

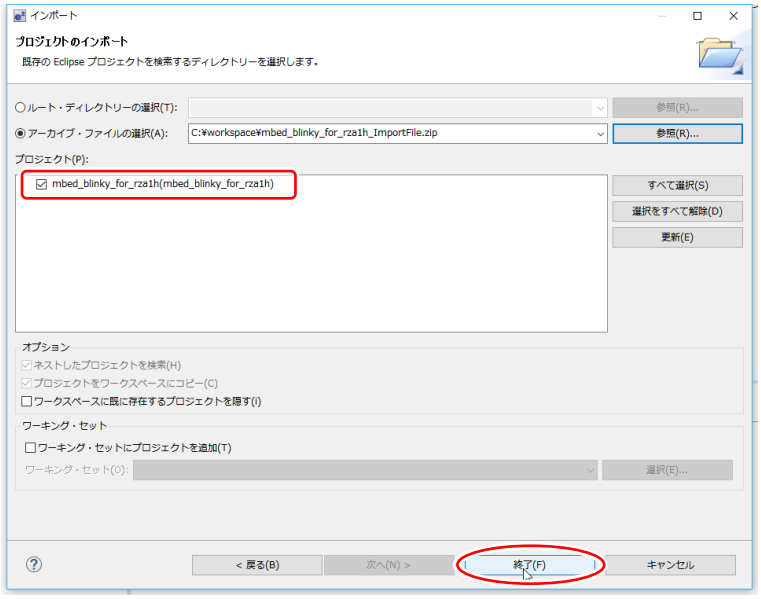
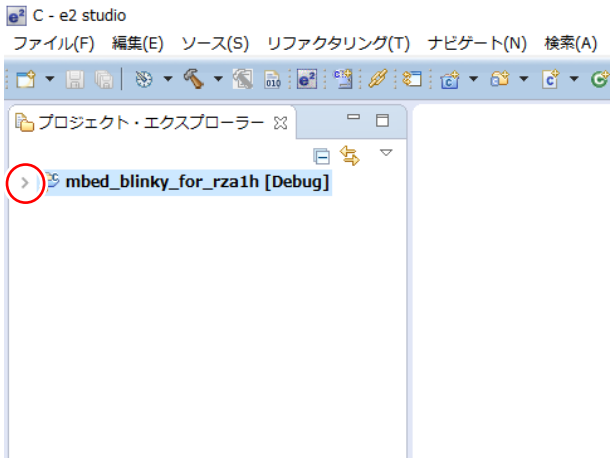
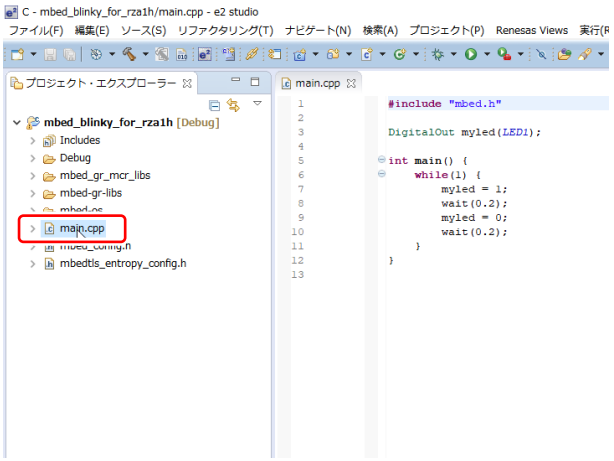
3		展開をクリックします。 ※展開をクリックすると、統合開発環境e2studioでインポートする「kit20_gr-peach_ImportFile.zip」が「C:¥workspace」フォルダに展開されます。

5.3 インポート

1		「e2 studio」を実行します。
2		ワークスペースのディレクトリが「C:¥workspace」フォルダになっていることを確認し、OKをクリックします。

3		「Workbench」をクリックします。
4		「ファイル → インポート」をクリックします。
5		「 > 一般」を開きます。

6		「既存プロジェクトをワークスペースへ」を選択し、次へをクリックします。
7		「アーカイブ・ファイルの選択」を選択し、参照をクリックします。
8		参照をクリックすると「C:¥workspace」フォルダが開きます。 「4.2 インストール」でインストールしたファイル「kit20_gr-peach_ImportFile.zip」を選択し、開くをクリックします。 ※「C:¥workspace」フォルダが開かない場合は、直接ディレクトリを指定して開いてください。

9		アーカイブファイルを選択すると、プロジェクトに、インポートするプロジェクトが表示されます。 終了をクリックし、インポートします。 ※プロジェクトに、インポートするプロジェクトが表示されない場合は、選択するアーカイブファイルが間違っています。再度ご確認ください。
10		プロジェクト「kit20_gr-peach」のインポートができました。 「>」をクリックすると、プロジェクトの中が表示されます。
11		「kit20_gr-peach.cpp」ファイルが C ソースファイルです。 「kit20_gr-peach.cpp」を開くと、エディタウィンドウにプログラムが表示されます。

6. プログラム解説「kit20_gr-peach.cpp」

6.1 プログラムリスト

RZ/A1H マイコンで画像処理マイコンカーを制御するプログラムを下記に示します。

```
 1 :  //-----------------------------------------------------------------------//
 2 :  //Supported MCU :  RZ/A1H
 3 :  //File Contents :  kit20_gr-peach ( Trace Program )
 4 :  //Version number:  Ver.1.00
 5 :  //Date:            2020.09.01
 6 :  //Copyright:       Renesas Electronics Corporation
 7 :  //                 Hitachi Document Solutions Co., Ltd.
 8 :  //-----------------------------------------------------------------------//
 9 :
10 :  //This program supports the following kit:
11 :  //* M-S348 Image processing micon car production kit
12 :
13 :  //-----------------------------------------------------------------------//
14 :  //Include
15 :  //-----------------------------------------------------------------------//
16 :  #include "mbed.h"
17 :  #include "iodefine.h"
18 :  #include "DisplayBace.h"
19 :  #include "ImageProcessFunc.h"
20 :  #include "EasyAttach_CameraAndLCD.h"
21 :
22 :  //-----------------------------------------------------------------------//
23 :  //Define
24 :  //-----------------------------------------------------------------------//
25 :  //Motor PWM cycle
26 :  #define     MOTOR_PWM_CYCLE     33332   /* Motor PWM period         */
27 :                                          /* 1ms    P0φ/1  = 0.03us   */
28 :  //Servo PWM cycle
29 :  #define     SERVO_PWM_CYCLE     33332   /* SERVO PWM period         */
30 :                                          /* 16ms   P0φ/16 = 0.48us   */
31 :  #define     SERVO_CENTER        3124    /* 1.5ms / 0.48us - 1 = 3124*/
32 :  #define     HANDLE_STEP         18      /* 1 degree value           */
33 :
34 :  #define     THRESHOLD           128     /* Set 0 to 255 Black:0 White:255 */
35 :
36 :  //Masked value settings X:masked (disabled) 0:not masked (enabled)
37 :  #define     MASK2_2             0x66    /* X 0 0 X  X 0 0 X         */
38 :  #define     MASK2_0             0x60    /* X 0 0 X  X X X X         */
39 :  #define     MASK0_2             0x06    /* X X X X  X 0 0 X         */
40 :  #define     MASK3_3             0xe7    /* 0 0 0 X  X 0 0 0         */
41 :  #define     MASK0_3             0x07    /* X X X X  X 0 0 0         */
42 :  #define     MASK3_0             0xe0    /* 0 0 0 X  X X X X         */
43 :  #define     MASK4_0             0xf0    /* 0 0 0 0  X X X X         */
44 :  #define     MASK0_4             0x0f    /* X X X X  0 0 0 0         */
45 :  #define     MASK4_4             0xff    /* 0 0 0 0  0 0 0 0         */
46 :
47 :  //LED Color on GR-PEACH
48 :  #define     LED_OFF             0x00
49 :  #define     LED_RED             0x01
50 :  #define     LED_GREEN           0x02
51 :  #define     LED_YELLOW          0x03
52 :  #define     LED_BLUE            0x04
53 :  #define     LED_PURPLE          0x05
54 :  #define     LED_SKYBLUE         0x06
55 :  #define     LED_WHITE           0x07
56 :
57 :  //led_m_set Function only
58 :  #define     RUN                 0x00
59 :  #define     STOP                0x01
60 :  #define     ERROR               0x02
61 :  #define     DEBUG               0x03
62 :  #define     CRANK               0x04
63 :  #define     LCHANGE             0x05
64 :
65 :  //SerialOut_CsvJpg Function only
66 :  #define     COLOR               0x00
67 :  #define     BINARY              0x01
68 :  #define     COLOR_BINARY        0x02
69 :
70 :  //-----------------------------------------------------------------------//
71 :  //Define(NTSC-Video)
72 :  //-----------------------------------------------------------------------//
73 :  #define VIDEO_INPUT_CH          (DisplayBase::VIDEO_INPUT_CHANNEL_0)
74 :  #define VIDEO_INT_TYPE          (DisplayBase::INT_TYPE_S0_VFIELD)
75 :
76 :  // VIDEO_FORMAT Select ( VIDEO_YCBCR422 or VIDEO_RGB888 )
77 :  #define VIDEO_FORMAT            (VIDEO_YCBCR422)
78 :
```

```
 79 :  #if(VIDEO_FORMAT == VIDEO_YCBCR422)
 80 :  #define DATA_SIZE_PER_PIC      (2u)
 81 :  #endif
 82 :  #if(VIDEO_FORMAT == VIDEO_RGB888)
 83 :  #define DATA_SIZE_PER_PIC      (4u)
 84 :  #endif
 85 :
 86 :  // Frame buffer stride: Frame buffer stride should be set to a multiple of 32 or 128
 87 :  // in accordance with the frame buffer burst transfer mode.
 88 :  #define PIXEL_HW               (160u)   /* QQVGA                     */
 89 :  #define PIXEL_VW               (120u)   /* QQVGA                     */
 90 :  #define VIDEO_BUFFER_STRIDE    (((PIXEL_HW * DATA_SIZE_PER_PIC) + 31u) & ~31u)
 91 :  #define VIDEO_BUFFER_HEIGHT    (PIXEL_VW)
 92 :
 93 :  #define TOP                     0       /* Even                      */
 94 :  #define BOTTOM                  1       /* Odd                       */
 95 :
 96 :  // Image Copy Size Select ( HALF or FULL )
 97 :  #define ICS                    (HALF)   /* Image Copy Function Only */
 98 :
 99 :  //---------------------------------------------------------------------//
100 :  // Struct
101 :  //---------------------------------------------------------------------//
102 :  typedef struct {
103 :      volatile int            x[8];       /* X-axis                    */
104 :      volatile int            y;          /* Y-axis                    */
105 :      volatile unsigned char  v;          /* Value                     */
106 :  } SENSOR;
107 :
108 :  //---------------------------------------------------------------------//
109 :  // Constructor
110 :  //---------------------------------------------------------------------//
111 :  // Create DisplayBase object
112 :  DisplayBase Display;
113 :
114 :  Ticker      interrput;
115 :  Serial      pc(USBTX, USBRX);
116 :
117 :  DigitalOut  LED_R(P6_13);               /* LED1 on the GR-PEACH board */
118 :  DigitalOut  LED_G(P6_14);               /* LED2 on the GR-PEACH board */
119 :  DigitalOut  LED_B(P6_15);               /* LED3 on the GR-PEACH board */
120 :  DigitalOut  USER_LED(P6_12);           /* USER_LED on the GR-PEACH board */
121 :  DigitalIn   user_botton(P6_0);          /* SW1 on the GR-PEACH board */
122 :
123 :  BusIn       dipsw( P7_15, P8_1, P2_9, P2_10 ); /* SW1 on Shield board */
124 :
125 :  DigitalOut  Left_motor_signal(P4_6);    /* Used by motor Function   */
126 :  DigitalOut  Right_motor_signal(P4_7);   /* Used by motor Function   */
127 :  DigitalIn   push_sw(P2_13);             /* SW1 on the Motor Drive board */
128 :  DigitalOut  LED_3(P2_14);               /* LED3 on the Motor Drive board */
129 :  DigitalOut  LED_2(P2_15);               /* LED2 on the Motor Drive board */
130 :
131 :  //---------------------------------------------------------------------//
132 :  // Prototype ( NTSC-video )
133 :  //---------------------------------------------------------------------//
134 :  static void Start_Video_Camera( void );
135 :  static void IntCallbackFunc_Vfield(DisplayBase::int_type_t int_type);
136 :
137 :  //---------------------------------------------------------------------//
138 :  // Prototype
139 :  //---------------------------------------------------------------------//
140 :  // Peripheral Functions
141 :  void init_MTU2_PWM_Motor( void );       /* Initialize PWM Functions */
142 :  void init_MTU2_PWM_Servo( void );       /* Initialize PWM Functions */
143 :
144 :  // Interrupt Function
145 :  void intTimer( void );                  /* 1ms period               */
146 :  void intImagePro( void );
147 :
148 :  // Micon board ( GR-PEACH )
149 :  void led_m_rgb(int led);
150 :  void led_m_user( int led );
151 :  unsigned char user_button_get( void );
152 :  void led_m_set( int set );
153 :  void led_m_pro( void );                 /* Only Function for interrupt */
154 :
155 :  // Motor drive board
156 :  void led_out(int led);
157 :  unsigned char pushsw_get( void );
158 :  void motor( int accele_l, int accele_r );
159 :  void handle( int angle );
160 :
161 :  // Shield board
162 :  unsigned char dipsw_get( void );
163 :
164 :  //---------------------------------------------------------------------//
165 :  // Prototype( Sensor Functions )
166 :  //---------------------------------------------------------------------//
167 :  int initSensor( mcrMat *ImData, SENSOR *S, int PL );
168 :  void SensorPro(  mcrMat *ImData, SENSOR *S ); /* Only Function for interrupt */
169 :  unsigned char sensor_inp( SENSOR *S, unsigned char mask );
```

```
170 :  int startbar_get( void );
171 :  int check_crossline( void );
172 :  int check_rightline( void );
173 :  int check_leftline( void );
174 :
175 :  //-------------------------------------------------------------------//
176 :  // Prototype( Debug Functions )
177 :  //-------------------------------------------------------------------//
178 :  void SerialOut_TeraTerm( mcrMat *ImData, int Mode );
179 :  void SerialOut_Excel( mcrMat *ImData, int Mode );
180 :  void SerialOut_CsvJpg( mcrMat *ImData, int Format, int ColorPattern );
181 :  void DebugPro( void );
182 :
183 :  //-------------------------------------------------------------------//
184 :  // Global Variable ( NTSC-video )
185 :  //-------------------------------------------------------------------//
186 :  static uint8_t FrameBuffer_Video_A[VIDEO_BUFFER_STRIDE *
                  VIDEO_BUFFER_HEIGHT]__attribute((section("NC_BSS"),aligned(16)));  //16 bytes aligned!;
187 :  uint8_t * write_buff_addr = FrameBuffer_Video_A;
188 :  static volatile int32_t field      = BOTTOM;
189 :  static volatile int32_t field_buff = BOTTOM;
190 :
191 :  //-------------------------------------------------------------------//
192 :  // Global Variable ( Image Functions )
193 :  //-------------------------------------------------------------------//
194 :  mcrMat        ImgInp0;                /* Screen Size ( 80 * 60 )  */
195 :  mcrMat        ImgInp1;                /* Screen Size ( 40 * 30 )  */
196 :  mcrMat        ImgBuff;                /* Screen Size ( 80 * 60 )  */
197 :  double        Rate = 0.5;             /* Reduction rate           */
198 :
199 :  //-------------------------------------------------------------------//
200 :  // Global Variable for Sensor Function
201 :  //-------------------------------------------------------------------//
202 :  SENSOR        sensor0;                /* Start bar sensor         */
203 :  SENSOR        sensor1;                /* Line trace sensor        */
204 :  int           senError;
205 :
206 :  //-------------------------------------------------------------------//
207 :  // Global Variable for Trace Program
208 :  //-------------------------------------------------------------------//
209 :  volatile unsigned long  PNumber;      /* intTimer Function Only   */
210 :  volatile int            DebugMode;    /* intTimer Function Only   */
211 :
212 :  volatile unsigned long  cnt0;         /* Used by timer Function   */
213 :  volatile unsigned long  cnt1;         /* Used within main         */
214 :  volatile int            pattern;      /* Pattern numbers          */
215 :
216 :  volatile int            ThresholdBuff;  /* ImageBinary Function only*/
217 :  volatile int            ThresholdMax;   /* ImageBinary Function only*/
218 :  volatile int            ThresholdMin;   /* ImageBinary Function only*/
219 :
220 :  volatile int            LedPattern;     /* led_m_pro Function only  */
221 :
222 :  // Led Set ( on GR-Peach board ) Function only
223 :  const int led_data[   ][ 3 ] = {
224 :          /* Color    , onTime, offTime */
225 :      { LED_GREEN ,    500,      500 },// 0:RUN
226 :      { LED_RED   ,    500,        0 },// 1:STOP
227 :      { LED_RED   ,    100,      100 },// 2:ERROR
228 :      { LED_BLUE  ,     50,       50 },// 3:DEBUG
229 :      { LED_YELLOW,    500,      500 },// 4:CRANK
230 :      { LED_BLUE  ,    500,      500 } // 5:LCHANGE
231 :  };
232 :
233 :  //*******************************************************************//
234 :  // Main Function
235 :  //*******************************************************************/
236 :  int main( void )
237 :  {
238 :      volatile int    SL;               /* Sensor Line Pixel        */
239 :
240 :      // Initialize MCU Functions
241 :      init_MTU2_PWM_Motor();
242 :      init_MTU2_PWM_Servo();
243 :      pc.baud(230400);
244 :
245 :      // Interrupt Function( intTimer )( 1ms )
246 :      interrput.attach(&intTimer, 0.001);
247 :      DebugMode = 0;
248 :
249 :      // Initialize Micon Car state
250 :      handle( 0 );
251 :      motor( 0, 0 );
252 :      led_out( 0x0 );
253 :      led_m_set( STOP );
254 :
255 :      // Initialize threshold
256 :      ThresholdBuff = THRESHOLD;
257 :
258 :      // Camera Start
259 :      EasyAttach_Init(Display, PIXEL_HW, PIXEL_VW);
```

```
260 :      Start_Video_Camera();
261 :
262 :      // wait to stabilize NTSC signal (about 200ms)
263 :      wait(0.2);
264 :
265 :      // Screen clear
266 :      pc.printf( "¥033[2J" );
267 :      pc.printf( "¥033[50F" );
268 :
269 :      // Initialize Sensor
270 :      // start bar sensor
271 :      SL = ( ( PIXEL_VW >> ICS ) * Rate ) * (double)0.7;
272 :      senError  = initSensor( &ImgInp1, &sensor1, SL );
273 :      // line trace sensor
274 :      SL = ( ( PIXEL_VW >> ICS ) * Rate ) * (double)0.8;
275 :      senError += initSensor( &ImgInp1, &sensor0, SL );
276 :      if( senError !=0 ) {
277 :          pc.printf( "initSensor Function Error ¥n¥r" );
278 :          pc.printf( "¥n¥r" );
279 :          led_m_set( ERROR );
280 :          cnt1 = 0;
281 :          while( cnt1 < 3000 ){
282 :              if( cnt1 % 200 < 100 ) {
283 :                  led_out( 0x3 );
284 :              } else {
285 :                  led_out( 0x0 );
286 :              }
287 :          }
288 :          led_m_set( STOP );
289 :      }
290 :
291 :      // Debug Program
292 :      if( user_button_get() ) {
293 :          led_m_set( DEBUG );
294 :          wait(0.1);  // ON  Time
295 :          while( user_button_get() );
296 :          wait(0.5);  // OFF Time
297 :          DebugPro();
298 :      }
299 :
300 :      // Trace Program
301 :      while( 1 ) {
302 :
303 :      /* pattern, sensor_inp() display */
304 :      if( cnt0 > 250 ) {
305 :          cnt0 = 0;
306 :          pc.printf( "pattern = %2d, ",         pattern                          );
307 :          pc.printf( "startbar_get() = %1d ",  startbar_get()                   );
308 :          pc.printf( "sensor_inp() = 0x%02x ", sensor_inp( &sensor0, MASK4_4 ) );
309 :          pc.printf( "¥r" );
310 :      }
311 :
312 :      switch( pattern ) {
313 :      /*****************************************************************
314 :      About patern
315 :        0:wait for switch input
316 :        1:check if start bar is open
317 :       11:normal trace
318 :       12:Check end of large turn to right
319 :       13:Check end of large turn to left
320 :       21:Processing at 1st cross line
321 :       22:Read but ignore 2nd line
322 :       23:Trace, crank detection after cross line
323 :       31:Left crank clearing processing ? wait until stable
324 :       32:Left crank clearing processing ? check end of turn
325 :       41:Right crank clearing processing ? wait until stable
326 :       42:Right crank clearing processing ? check end of turn
327 :       51:Processing at 1st right half line detection
328 :       52:Read but ignore 2nd time
329 :       53:Trace, lane change after right half line detection
330 :       54:Right lane change end check
331 :       61:Processing at 1st left half line detection
332 :       62:Read but ignore 2nd time
333 :       63:Trace, lane change after left half line detection
334 :       64:Left lane change end check
335 :      *****************************************************************/
336 :      case 0:
337 :          /* wait for switch input */
338 :          if( pushsw_get() ) {
339 :              led_out( 0x0 );
340 :              pattern = 1;
341 :              while( pushsw_get() );
342 :              cnt1 = 0;
343 :              break;
344 :          }
345 :          if( !startbar_get() ) {
346 :              if( cnt1 < 100 ) { /* LED flashing processing */
347 :                  led_out( 0x1 );
348 :              } else if( cnt1 < 200 ) {
349 :                  led_out( 0x2 );
350 :              } else {
```

```
351 :                   cnt1 = 0;
352 :               }
353 :           } else {
354 :               if( cnt1 < 50 ) { /* LED flashing processing */
355 :                   led_out( 0x1 );
356 :               } else if( cnt1 < 100 ) {
357 :                   led_out( 0x2 );
358 :               } else {
359 :                   cnt1 = 0;
360 :               }
361 :           }
362 :           break;
363 :
364 :       case 1:
365 :           /* Check if start bar is open */
366 :           if( !startbar_get() ) {
367 :               /* Start!! */
368 :               led_out( 0x0 );
369 :               pattern = 11;
370 :               cnt1 = 0;
371 :               break;
372 :           }
373 :           if( cnt1 < 50 ) { /* LED flashing processing */
374 :               led_out( 0x1 );
375 :           } else if( cnt1 < 100 ) {
376 :               led_out( 0x2 );
377 :           } else {
378 :               cnt1 = 0;
379 :           }
380 :           break;
381 :
382 :       case 11:
383 :           /* normal trace */
384 :           if( check_crossline() ) { /* Cross line check */
385 :               pattern = 21;
386 :               break;
387 :           }
388 :           if( check_rightline() ) { /* Right half line detection check */
389 :               pattern = 51;
390 :               break;
391 :           }
392 :           if( check_leftline() ) { /* Left half line detection check */
393 :               pattern = 61;
394 :               break;
395 :           }
396 :           switch( sensor_inp( &sensor0, MASK3_3 ) ) {
397 :               case 0x00:
398 :                   /* Center -> straight */
399 :                   handle( 0 );
400 :                   motor( 100, 100 );
401 :                   break;
402 :
403 :               case 0x04:
404 :                   /* Slight amount left of center -> slight turn to right */
405 :                   handle( 5 );
406 :                   motor( 100, 100 );
407 :                   break;
408 :
409 :               case 0x06:
410 :                   /* Small amount left of center -> small turn to right */
411 :                   handle( 10 );
412 :                   motor( 80, 67 );
413 :                   break;
414 :
415 :               case 0x07:
416 :                   /* Medium amount left of center -> medium turn to right */
417 :                   handle( 15 );
418 :                   motor( 50, 38 );
419 :                   break;
420 :
421 :               case 0x03:
422 :                   /* Large amount left of center -> large turn to right */
423 :                   handle( 25 );
424 :                   motor( 30, 19 );
425 :                   pattern = 12;
426 :                   break;
427 :
428 :               case 0x20:
429 :                   /* Slight amount right of center -> slight turn to left */
430 :                   handle( -5 );
431 :                   motor( 100, 100 );
432 :                   break;
433 :
434 :               case 0x60:
435 :                   /* Small amount right of center -> small turn to left */
436 :                   handle( -10 );
437 :                   motor( 67, 80 );
438 :                   break;
439 :
440 :               case 0xe0:
441 :                   /* Medium amount right of center -> medium turn to left */
```

```
442 :                   handle( -15 );
443 :                   motor( 38, 50 );
444 :                   break;
445 :
446 :               case 0xc0:
447 :                   /* Large amount right of center -> large turn to left */
448 :                   handle( -25 );
449 :                   motor( 19, 30 );
450 :                   pattern = 13;
451 :                   break;
452 :
453 :               default:
454 :                   break;
455 :           }
456 :           break;
457 :
458 :       case 12:
459 :           /* Check end of large turn to right */
460 :           if( sensor_inp( &sensor0, MASK3_3 ) == 0x06 ) {
461 :               pattern = 11;
462 :           }
463 :           if( check_crossline() ) { /* Cross line check */
464 :               pattern = 21;
465 :               break;
466 :           }
467 :           if( check_rightline() ) { /* Right half line detection check */
468 :               pattern = 51;
469 :               break;
470 :           }
471 :           if( check_leftline() ) { /* Left half line detection check */
472 :               pattern = 61;
473 :               break;
474 :           }
475 :           break;
476 :
477 :       case 13:
478 :           /* Check end of large turn to left */
479 :           if( sensor_inp( &sensor0, MASK3_3 ) == 0x60 ) {
480 :               pattern = 11;
481 :           }
482 :           if( check_crossline() ) { /* Cross line check */
483 :               pattern = 21;
484 :               break;
485 :           }
486 :           if( check_rightline() ) { /* Right half line detection check */
487 :               pattern = 51;
488 :               break;
489 :           }
490 :           if( check_leftline() ) { /* Left half line detection check */
491 :               pattern = 61;
492 :               break;
493 :           }
494 :           break;
495 :
496 :       case 21:
497 :           /* Processing at 1st cross line */
498 :           led_out( 0x3 );
499 :           handle( 0 );
500 :           motor( 0 ,0 );
501 :           pattern = 22;
502 :           cnt1 = 0;
503 :           break;
504 :
505 :       case 22:
506 :           /* Read but ignore 2nd line */
507 :           if( cnt1 > 100 ) {
508 :               pattern = 23;
509 :               cnt1 = 0;
510 :           }
511 :           break;
512 :
513 :       case 23:
514 :           /* Trace, crank detection after cross line */
515 :           if( sensor_inp( &sensor0, MASK4_4 ) == 0xf8 ) {
516 :               /* Left crank determined -> to left crank clearing processing */
517 :               led_out( 0x1 );
518 :               handle( -38 );
519 :               motor( 10 ,50 );
520 :               pattern = 31;
521 :               cnt1 = 0;
522 :               break;
523 :           }
524 :           if( sensor_inp( &sensor0 , MASK4_4 ) == 0x1f ) {
525 :               /* Right crank determined -> to right crank clearing processing */
526 :               led_out( 0x2 );
527 :               handle( 38 );
528 :               motor( 50 ,10 );
529 :               pattern = 41;
530 :               cnt1 = 0;
531 :               break;
532 :           }
```

```
533 :           switch( sensor_inp( &sensor0, MASK3_3 ) ) {
534 :               case 0x00:
535 :                   /* Center -> straight */
536 :                   handle( 0 );
537 :                   motor( 40 ,40 );
538 :                   break;
539 :               case 0x04:
540 :               case 0x06:
541 :               case 0x07:
542 :               case 0x03:
543 :                   /* Left of center -> turn to right */
544 :                   handle( 8 );
545 :                   motor( 40 ,35 );
546 :                   break;
547 :               case 0x20:
548 :               case 0x60:
549 :               case 0xe0:
550 :               case 0xc0:
551 :                   /* Right of center -> turn to left */
552 :                   handle( -8 );
553 :                   motor( 35 ,40 );
554 :                   break;
555 :           }
556 :           break;
557 :
558 :       case 31:
559 :           /* Left crank clearing processing ? wait until stable */
560 :           if( cnt1 > 200 ) {
561 :               pattern = 32;
562 :               cnt1 = 0;
563 :           }
564 :           break;
565 :
566 :       case 32:
567 :           /* Left crank clearing processing ? check end of turn */
568 :           if( sensor_inp( &sensor0, MASK3_3 ) == 0x60 ) {
569 :               led_out( 0x0 );
570 :               pattern = 11;
571 :               cnt1 = 0;
572 :           }
573 :           break;
574 :
575 :       case 41:
576 :           /* Right crank clearing processing ? wait until stable */
577 :           if( cnt1 > 200 ) {
578 :               pattern = 42;
579 :               cnt1 = 0;
580 :           }
581 :           break;
582 :
583 :       case 42:
584 :           /* Right crank clearing processing ? check end of turn */
585 :           if( sensor_inp( &sensor0, MASK3_3 ) == 0x06 ) {
586 :               led_out( 0x0 );
587 :               pattern = 11;
588 :               cnt1 = 0;
589 :           }
590 :           break;
591 :
592 :       case 51:
593 :           /* Processing at 1st right half line detection */
594 :           led_out( 0x2 );
595 :           handle( 0 );
596 :           motor( 0 ,0 );
597 :           pattern = 52;
598 :           cnt1 = 0;
599 :           break;
600 :
601 :       case 52:
602 :           /* Read but ignore 2nd time */
603 :           if( cnt1 > 100 ){
604 :               pattern = 53;
605 :               cnt1 = 0;
606 :           }
607 :           break;
608 :
609 :       case 53:
610 :           /* Trace, lane change after right half line detection */
611 :           if( sensor_inp( &sensor0, MASK4_4 ) == 0x00 ) {
612 :               handle( 15 );
613 :               motor( 40 ,31 );
614 :               pattern = 54;
615 :               cnt1 = 0;
616 :               break;
617 :           }
618 :           switch( sensor_inp( &sensor0, MASK3_3) ) {
619 :               case 0x00:
620 :                   /* Center -> straight */
621 :                   handle( 0 );
622 :                   motor( 40 ,40 );
623 :                   break;
```

```
624 :             case 0x04:
625 :             case 0x06:
626 :             case 0x07:
627 :             case 0x03:
628 :                 /* Left of center -> turn to right */
629 :                 handle( 8 );
630 :                 motor( 40 ,35 );
631 :                 break;
632 :             case 0x20:
633 :             case 0x60:
634 :             case 0xe0:
635 :             case 0xc0:
636 :                 /* Right of center -> turn to left */
637 :                 handle( -8 );
638 :                 motor( 35 ,40 );
639 :                 break;
640 :             default:
641 :                 break;
642 :         }
643 :         break;
644 :
645 :     case 54:
646 :         /* Right lane change end check */
647 :         if( sensor_inp( &sensor0, MASK4_4 ) == 0x3c ) {
648 :             led_out( 0x0 );
649 :             pattern = 11;
650 :             cnt1 = 0;
651 :         }
652 :         break;
653 :
654 :     case 61:
655 :         /* Processing at 1st left half line detection */
656 :         led_out( 0x1 );
657 :         handle( 0 );
658 :         motor( 0 ,0 );
659 :         pattern = 62;
660 :         cnt1 = 0;
661 :         break;
662 :
663 :     case 62:
664 :         /* Read but ignore 2nd time */
665 :         if( cnt1 > 100 ){
666 :             pattern = 63;
667 :             cnt1 = 0;
668 :         }
669 :         break;
670 :
671 :     case 63:
672 :         /* Trace, lane change after left half line detection */
673 :         if( sensor_inp( &sensor0, MASK4_4) == 0x00 ) {
674 :             handle( -15 );
675 :             motor( 31 ,40 );
676 :             pattern = 64;
677 :             cnt1 = 0;
678 :             break;
679 :         }
680 :         switch( sensor_inp( &sensor0, MASK3_3) ) {
681 :             case 0x00:
682 :                 /* Center -> straight */
683 :                 handle( 0 );
684 :                 motor( 40 ,40 );
685 :                 break;
686 :             case 0x04:
687 :             case 0x06:
688 :             case 0x07:
689 :             case 0x03:
690 :                 /* Left of center -> turn to right */
691 :                 handle( 8 );
692 :                 motor( 40 ,35 );
693 :                 break;
694 :             case 0x20:
695 :             case 0x60:
696 :             case 0xe0:
697 :             case 0xc0:
698 :                 /* Right of center -> turn to left */
699 :                 handle( -8 );
700 :                 motor( 35 ,40 );
701 :                 break;
702 :             default:
703 :                 break;
704 :         }
705 :         break;
706 :
707 :     case 64:
708 :         /* Left lane change end check */
709 :         if( sensor_inp( &sensor0, MASK4_4 ) == 0x3c ) {
710 :             led_out( 0x0 );
711 :             pattern = 11;
712 :             cnt1 = 0;
713 :         }
714 :         break;
```

```
715 :
716 :      default:
717 :          break;
718 :      }
719 :      }
720 :  }
721 :
722 :  //*****************************************************************//
723 :  // Initialize Functions
724 :  //*****************************************************************/
725 :  //-----------------------------------------------------------------//
726 :  // Initialize MTU2 PWM Functions
727 :  //-----------------------------------------------------------------//
728 :  // MTU2_3, MTU2_4
729 :  // Reset-Synchronized PWM mode
730 :  // TIOC4A(P4_4) :Left-motor
731 :  // TIOC4B(P4_5) :Right-motor
732 :  //-----------------------------------------------------------------//
733 :  void init_MTU2_PWM_Motor( void )
734 :  {
735 :      // Port setting for S/W I/O Control
736 :      // alternative mode
737 :
738 :      // MTU2_4 (P4_4)(P4_5)
739 :      GPIOPBDC4   = 0x0000;               /* Bidirection mode disabled*/
740 :      GPIOPFCAE4 &= 0xffcf;               /* The alternative Function of a pin */
741 :      GPIOPFCE4  |= 0x0030;               /* The alternative Function of a pin */
742 :      GPIOPFC4   &= 0xffcf;               /* The alternative Function of a pin */
743 :                                          /* 2nd altemative Function/output    */
744 :      GPIOP4     &= 0xffcf;               /*                                   */
745 :      GPIOPM4    &= 0xffcf;               /* p4_4,P4_5:output                  */
746 :      GPIOPMC4   |= 0x0030;               /* P4_4,P4_5:double                  */
747 :
748 :      // Module stop 33(MTU2) canceling
749 :      CPGSTBCR3  &= 0xf7;
750 :
751 :      // MTU2_3 and MTU2_4 (Motor PWM)
752 :      MTU2TCR_3   = 0x20;                 /* TCNT Clear(TGRA), Pφ/1  */
753 :      MTU2TOCR1   = 0x04;                 /*                         */
754 :      MTU2TOCR2   = 0x40;                 /* N L>H  P H>L            */
755 :      MTU2TMDR_3  = 0x38;                 /* Buff:ON Reset-Synchronized PWM mode */
756 :      MTU2TMDR_4  = 0x30;                 /* Buff:ON                 */
757 :      MTU2TOER    = 0xc6;                 /* TIOC3B,4A,4B enabled output */
758 :      MTU2TCNT_3  = MTU2TCNT_4 = 0;       /* TCNT3,TCNT4 Set 0       */
759 :      MTU2TGRA_3  = MTU2TGRC_3 = MOTOR_PWM_CYCLE;
760 :                                          /* PWM-Cycle(1ms)          */
761 :      MTU2TGRA_4  = MTU2TGRC_4 = 0;       /* Left-motor(P4_4)        */
762 :      MTU2TGRB_4  = MTU2TGRD_4 = 0;       /* Right-motor(P4_5)       */
763 :      MTU2TSTR   |= 0x40;                 /* TCNT_4 Start            */
764 :  }
765 :
766 :  //-----------------------------------------------------------------//
767 :  // Initialize MTU2 PWM Functions
768 :  //-----------------------------------------------------------------//
769 :  // MTU2_0
770 :  // PWM mode 1
771 :  // TIOC0A(P4_0) :Servo-motor
772 :  //-----------------------------------------------------------------//
773 :  void init_MTU2_PWM_Servo( void )
774 :  {
775 :      // Port setting for S/W I/O Control
776 :      // alternative mode
777 :
778 :      // MTU2_0 (P4_0)
779 :      GPIOPBDC4   = 0x0000;               /* Bidirection mode disabled*/
780 :      GPIOPFCAE4 &= 0xfffe;               /* The alternative Function of a pin */
781 :      GPIOPFCE4  &= 0xfffe;               /* The alternative Function of a pin */
782 :      GPIOPFC4   |= 0x0001;               /* The alternative Function of a pin */
783 :                                          /* 2nd alternative Function/output   */
784 :      GPIOP4     &= 0xfffe;               /*                                   */
785 :      GPIOPM4    &= 0xfffe;               /* p4_0:output                       */
786 :      GPIOPMC4   |= 0x0001;               /* P4_0:double                       */
787 :
788 :      // Module stop 33(MTU2) canceling
789 :      CPGSTBCR3  &= 0xf7;
790 :
791 :      // MTU2_0 (Motor PWM)
792 :      MTU2TCR_0   = 0x22;                 /* TCNT Clear(TGRA), Pφ/16 */
793 :      MTU2TIORH_0 = 0x52;                 /* TGRA L>H, TGRB H>L      */
794 :      MTU2TMDR_0  = 0x32;                 /* TGRC and TGRD = Buff-mode*/
795 :                                          /* PWM-mode1               */
796 :      MTU2TCNT_0  = 0;                    /* TCNT0 Set 0             */
797 :      MTU2TGRA_0  = MTU2TGRC_0 = SERVO_PWM_CYCLE;
798 :                                          /* PWM-Cycle(16ms)         */
799 :      MTU2TGRB_0  = MTU2TGRD_0 = 0;       /* Servo-motor(P4_0)       */
800 :      MTU2TSTR   |= 0x01;                 /* TCNT_0 Start            */
801 :  }
802 :
803 :  //-----------------------------------------------------------------//
804 :  // Initialize Camera Function
805 :  //-----------------------------------------------------------------//
```

```
806 :  static void Start_Video_Camera( void )
807 :  {
808 :      // Initialize the background to black
809 :      for (uint32_t i = 0; i < sizeof(FrameBuffer_Video_A); i += 2) {
810 :          FrameBuffer_Video_A[i + 0] = 0x10;
811 :          FrameBuffer_Video_A[i + 1] = 0x80;
812 :      }
813 :      // Interrupt callback Function setting (Field end signal for recording Function in scaler 0)
814 :      Display.Graphics_Irq_Handler_Set(VIDEO_INT_TYPE, 0, IntCallbackFunc_Vfield);
815 :      // Video capture setting (progressive form fixed)
816 :      Display.Video_Write_Setting(
817 :          VIDEO_INPUT_CH,
818 :          DisplayBase::COL_SYS_NTSC_358,
819 :          write_buff_addr,
820 :          VIDEO_BUFFER_STRIDE,
821 :          #if(VIDEO_FORMAT == VIDEO_YCBCR422)
822 :          DisplayBase::VIDEO_FORMAT_YCBCR422,
823 :          DisplayBase::WR_RD_WRSWA_32_16BIT,
824 :          #endif
825 :          #if(VIDEO_FORMAT == VIDEO_RGB888)
826 :          DisplayBase::VIDEO_FORMAT_RGB888,
827 :          DisplayBase::WR_RD_WRSWA_32BIT,
828 :          #endif
829 :          PIXEL_VW,
830 :          PIXEL_HW
831 :      );
832 :      EasyAttach_CameraStart(Display, VIDEO_INPUT_CH);
833 :  }
834 :
835 :  //------------------------------------------------------------------//
836 :  // @brief       Interrupt callback Function
837 :  // @param[in]   int_type    : VDC5 interrupt type
838 :  // @retval      None
839 :  //------------------------------------------------------------------//
840 :  static void IntCallbackFunc_Vfield(DisplayBase::int_type_t int_type)
841 :  {
842 :      // Top or Bottom
843 :      if( field == BOTTOM ) field = TOP;
844 :      else                  field = BOTTOM;
845 :      // Led
846 :      led_m_user( field );
847 :  }
848 :
849 :  //------------------------------------------------------------------//
850 :  // Interrupt Function( intTimer )
851 :  //------------------------------------------------------------------//
852 :  void intTimer( void )
853 :  {
854 :      cnt0++;
855 :      cnt1++;
856 :
857 :      if( !DebugMode ) {
858 :          if( field_buff != field ) {
859 :              field_buff  = field;
860 :              PNumber = 0;
861 :          }
862 :          intImagePro();
863 :      }
864 :
865 :      led_m_pro();
866 :  }
867 :
868 :  //------------------------------------------------------------------//
869 :  // Interrupt Function( intImagePro )
870 :  //------------------------------------------------------------------//
871 :  void intImagePro( void )
872 :  {
873 :      switch( PNumber++ ) {
874 :
875 :      #if(VIDEO_FORMAT == VIDEO_YCBCR422)
876 :      case 0:
877 :          // Size( 160 * 120 ) -> ( 80 * 60 ) Top field or Bottom field
878 :          intImageCopy( write_buff_addr, PIXEL_HW, PIXEL_VW, VIDEO_FORMAT, field, &ImgInp0, ICS,
                                                                                        INTERMITTENT_PRO4 );
879 :          break;
880 :      case 1:
881 :          intImageCopy( write_buff_addr, PIXEL_HW, PIXEL_VW, VIDEO_FORMAT, field, &ImgInp0, ICS,
                                                                                        INTERMITTENT_PRO4 );
882 :          break;
883 :      case 2:
884 :          intImageCopy( write_buff_addr, PIXEL_HW, PIXEL_VW, VIDEO_FORMAT, field, &ImgInp0, ICS,
                                                                                        INTERMITTENT_PRO4 );
885 :          break;
886 :      case 3:
887 :          intImageCopy( write_buff_addr, PIXEL_HW, PIXEL_VW, VIDEO_FORMAT, field, &ImgInp0, ICS,
                                                                                        INTERMITTENT_PRO4 );
888 :          break;
889 :      case 4:
890 :          // Size( 80 * 60 ) -> ( 40 * 30 )
891 :          intImageReduction( &ImgInp0, &ImgInp1, VIDEO_FORMAT, Rate, INTERMITTENT_PRO2 );
892 :          break;
```

```
893 :      case 5:
894 :          intImageReduction( &ImgInp0, &ImgInp1, VIDEO_FORMAT, Rate, INTERMITTENT_PRO2 );
895 :          break;
896 :      case 6:
897 :          // Automatic calculation of threshold value ( YCBCR422 Only function )
898 :          ThresholdBuff = DiscriminantAnalysisMethod( &ImgInp1 );
899 :          // Threshold Max
900 :          ThresholdMax  = ThresholdBuff + 255;
901 :          if( ThresholdMax > 255 ) ThresholdMax = 255;
902 :          // Threshold Min
903 :          ThresholdMin  = ThresholdBuff + 0;
904 :          if( ThresholdMin  <=  50 ) ThresholdMin =  50;
905 :          break;
906 :      case 7:
907 :          // Binary process
908 :          ImageBinarySetY( ThresholdMin, ThresholdMax );
909 :          intImageBinary( &ImgInp1, &ImgInp1, BINARY_Y, INTERMITTENT_PRO1 );
910 :          SensorPro( &ImgInp1, &sensor1 );
911 :          SensorPro( &ImgInp1, &sensor0 );
912 :          break;
913 :      #endif
914 :
915 :      #if(VIDEO_FORMAT == VIDEO_RGB888)
916 :      case 0:
917 :          // Size( 160 * 120 ) -> ( 80 * 60 ) Top field or Bottom field
918 :          intImageCopy( write_buff_addr, PIXEL_HW, PIXEL_VW, VIDEO_FORMAT, field, &ImgInp0, ICS,
                                                                                        INTERMITTENT_PRO4 );
919 :          break;
920 :      case 1:
921 :          intImageCopy( write_buff_addr, PIXEL_HW, PIXEL_VW, VIDEO_FORMAT, field, &ImgInp0, ICS,
                                                                                        INTERMITTENT_PRO4 );
922 :          break;
923 :      case 2:
924 :          intImageCopy( write_buff_addr, PIXEL_HW, PIXEL_VW, VIDEO_FORMAT, field, &ImgInp0, ICS,
                                                                                        INTERMITTENT_PRO4 );
925 :          break;
926 :      case 3:
927 :          intImageCopy( write_buff_addr, PIXEL_HW, PIXEL_VW, VIDEO_FORMAT, field, &ImgInp0, ICS,
                                                                                        INTERMITTENT_PRO4 );
928 :          break;
929 :      case 4:
930 :          // Size( 80 * 60 ) -> ( 40 * 30 )
931 :          intImageReduction( &ImgInp0, &ImgInp1, VIDEO_FORMAT, Rate, INTERMITTENT_PRO2 );
932 :          break;
933 :      case 5:
934 :          intImageReduction( &ImgInp0, &ImgInp1, VIDEO_FORMAT, Rate, INTERMITTENT_PRO2 );
935 :          break;
936 :      case 6:
937 :          // RGB -> YCBCR422
938 :          intRGB_YCBCR_Converter( &ImgInp1, INTERMITTENT_PRO1 );
939 :          break;
940 :      case 7:
941 :          // Automatic calculation of threshold value ( YCBCR422 Only function )
942 :          ThresholdBuff = DiscriminantAnalysisMethod( &ImgInp1 );
943 :          // Threshold Max
944 :          ThresholdMax  = ThresholdBuff + 255;
945 :          if( ThresholdMax > 255 ) ThresholdMax = 255;
946 :          // Threshold Min
947 :          ThresholdMin  = ThresholdBuff + 0;
948 :          if( ThresholdMin  <=  50 ) ThresholdMin =  50;
949 :          break;
950 :      case 8:
951 :          // Binary process
952 :          ImageBinarySetR( ThresholdMin, ThresholdMax );
953 :          ImageBinarySetG( ThresholdMin, ThresholdMax );
954 :          ImageBinarySetB( ThresholdMin, ThresholdMax );
955 :          intImageBinary( &ImgInp1, &ImgInp1, BINARY_RGB, INTERMITTENT_PRO1 );
956 :          SensorPro( &ImgInp1, &sensor1 );
957 :          SensorPro( &ImgInp1, &sensor0 );
958 :          break;
959 :      #endif
960 :
961 :      default:
962 :          PNumber = 16;
963 :          break;
964 :      }
965 :  }
966 :
967 :  //*****************************************************************//
968 :  // Functions ( on Micon board )
969 :  //*****************************************************************/
970 :  //-----------------------------------------------------------------//
971 :  // Led RGB Function
972 :  // led        : 0 ~ 7 ( Select color )
973 :  //-----------------------------------------------------------------//
974 :  void led_m_rgb(int led)
975 :  {
976 :      LED_R = led & 0x1;
977 :      LED_G = (led >> 1 ) & 0x1;
978 :      LED_B = (led >> 2 ) & 0x1;
979 :  }
```

```
 980 :
 981 :  //---------------------------------------------------------------------//
 982 :  // User Button Get Function
 983 :  // Return     : 0:OFF, 1:ON
 984 :  //---------------------------------------------------------------------//
 985 :  unsigned char user_button_get( void )
 986 :  {
 987 :      return (~user_botton) & 0x1;        /* Read ports with switches */
 988 :  }
 989 :
 990 :  //---------------------------------------------------------------------//
 991 :  // Led User ( on Micon board ) Function
 992 :  // led        : 0:OFF, 1:ON
 993 :  //---------------------------------------------------------------------//
 994 :  void led_m_user( int led )
 995 :  {
 996 :      USER_LED = led & 0x01;
 997 :  }
 998 :
 999 :  //---------------------------------------------------------------------//
1000 :  // Led Set ( on Micon board )Function
1001 :  // led       : Led pattern ( RUN, STOP etc )
1002 :  //---------------------------------------------------------------------//
1003 :  void led_m_set( int set )
1004 :  {
1005 :      LedPattern = set;
1006 :  }
1007 :
1008 :  //---------------------------------------------------------------------//
1009 :  // Led Process ( on Micon board ) Function for only interrupt
1010 :  //---------------------------------------------------------------------//
1011 :  void led_m_pro( void )
1012 :  {
1013 :      volatile static long    cnt_led_m;
1014 :
1015 :      cnt_led_m++;
1016 :
1017 :      if( pattern <= 0 ) {
1018 :          /* NONE */
1019 :      } else if( pattern <= 20 ) {
1020 :          led_m_set( RUN );
1021 :      } else if( pattern <= 50 ) {
1022 :          led_m_set( CRANK );
1023 :      } else if( pattern <= 70 ) {
1024 :          led_m_set( LCHANGE );
1025 :      } else {
1026 :          led_m_set( ERROR );
1027 :      }
1028 :
1029 :      if( cnt_led_m <   led_data[ LedPattern ][ 1 ] ) led_m_rgb( led_data[ LedPattern ][ 0 ] );
1030 :      else if( cnt_led_m < ( led_data[ LedPattern ][ 1 ] + led_data[ LedPattern ][ 2 ] ) ) led_m_rgb( LED_OFF );
1031 :      else cnt_led_m = 0;
1032 :  }
1033 :
1034 :  //*********************************************************************//
1035 :  // Functions ( on Motor drive board )
1036 :  //*********************************************************************/
1037 :  //---------------------------------------------------------------------//
1038 :  // Led Out Function
1039 :  // led       : LED2:bit1 LED3:bit0  "0":OFF "1":ON
1040 :  // For example ) 0x3->LED2:ON LED3:ON  0x2->LED2:ON LED3:OFF
1041 :  //---------------------------------------------------------------------//
1042 :  void led_out(int led)
1043 :  {
1044 :      led = ~led;
1045 :      LED_3 = led & 0x1;
1046 :      LED_2 = ( led >> 1 ) & 0x1;
1047 :  }
1048 :
1049 :  //---------------------------------------------------------------------//
1050 :  // Push SW Get Function
1051 :  // Return     : 0:OFF, 1:ON
1052 :  //---------------------------------------------------------------------//
1053 :  unsigned char pushsw_get( void )
1054 :  {
1055 :      return (~push_sw) & 0x1;            /* Read ports with switches */
1056 :  }
1057 :
1058 :  //---------------------------------------------------------------------//
1059 :  // motor speed control(PWM)
1060 :  // Arguments: motor:-100 to 100
1061 :  // Here, 0 is stop, 100 is forward, -100 is reverse
1062 :  //---------------------------------------------------------------------//
1063 :  void motor( int accele_l, int accele_r )
1064 :  {
1065 :      int    sw_data;
1066 :
1067 :      sw_data = dipsw_get() + 5;
1068 :      accele_l = ( accele_l * sw_data ) / 20;
1069 :      accele_r = ( accele_r * sw_data ) / 20;
1070 :
```

```
1071 :      // Left Motor Control
1072 :      if( accele_l >= 0 ) {
1073 :          // forward
1074 :          Left_motor_signal = 0;
1075 :          MTU2TGRC_4 = (long)( MOTOR_PWM_CYCLE - 1 ) * accele_l / 100;
1076 :      } else {
1077 :          // reverse
1078 :          Left_motor_signal = 1;
1079 :          MTU2TGRC_4 = (long)( MOTOR_PWM_CYCLE - 1 ) * ( -accele_l ) / 100;
1080 :      }
1081 :
1082 :      // Right Motor Control
1083 :      if( accele_r >= 0 ) {
1084 :          // forward
1085 :          Right_motor_signal = 0;
1086 :          MTU2TGRD_4 = (long)( MOTOR_PWM_CYCLE - 1 ) * accele_r / 100;
1087 :      } else {
1088 :          // reverse
1089 :          Right_motor_signal = 1;
1090 :          MTU2TGRD_4 = (long)( MOTOR_PWM_CYCLE - 1 ) * ( -accele_r ) / 100;
1091 :      }
1092 :  }
1093 :
1094 :  //-----------------------------------------------------------------------//
1095 :  // handle Function
1096 :  //-----------------------------------------------------------------------//
1097 :  void handle( int angle )
1098 :  {
1099 :      // When the servo move from left to right in reverse, replace "-" with "+"
1100 :      MTU2TGRD_0 = SERVO_CENTER - angle * HANDLE_STEP;
1101 :  }
1102 :
1103 :  //***********************************************************************//
1104 :  // Functions ( on Shield board )
1105 :  //***********************************************************************/
1106 :  //-----------------------------------------------------------------------//
1107 :  // Dipsw get Function
1108 :  // Return     : SW value 0 ~ 15
1109 :  //-----------------------------------------------------------------------//
1110 :  unsigned char dipsw_get( void )
1111 :  {
1112 :      return( dipsw.read() & 0x0f );
1113 :  }
1114 :
1115 :  //***********************************************************************//
1116 :  // Sensor Functions
1117 :  //***********************************************************************/
1118 :  //-----------------------------------------------------------------------//
1119 :  // Initialize Sensor Function
1120 :  // ImData     : mcrMat
1121 :  // S          : SENSOR
1122 :  // PL         : Pixel Line
1123 :  //-----------------------------------------------------------------------//
1124 :  int initSensor( mcrMat *ImData, SENSOR *S, int PL )
1125 :  {
1126 :      volatile int    X, Y, L;
1127 :      volatile int    CX;
1128 :      volatile int    Gap, offset;
1129 :      volatile char   ErrorCode;
1130 :
1131 :      // Initialize
1132 :      offset    = 0;
1133 :      ErrorCode = 0;
1134 :
1135 :      // Center Point X
1136 :      // Left side
1137 :      S->y = ( ImData->h - 1 );
1138 :      L = ( ImData->h - 1 ) * ImData->w;
1139 :      if( ImData->bin[ L + 0 ] == 1 ) S->x[ 5 ] = 0;
1140 :      if( ImData->bin[ L + 1 ] == 1 ) S->x[ 5 ] = 1;
1141 :      if( ImData->bin[ L + 0 ] == 0 &&
1142 :          ImData->bin[ L + 1 ] == 0  ) {
1143 :          for( X = 2; X < ( ImData->w - 1 ); X++ ) {
1144 :              L = ( ( ImData->h - 1 ) * ImData->w ) + X;
1145 :              if( ImData->bin[ L - 2 ] == 0 &&
1146 :                  ImData->bin[ L - 1 ] == 0 &&
1147 :                  ImData->bin[ L - 0 ] == 1 &&
1148 :                  ImData->bin[ L + 1 ] == 1 ) {
1149 :                  S->x[ 5 ] = X;
1150 :                  break;
1151 :              }
1152 :          }
1153 :      }
1154 :      // Right side
1155 :      L = ( ImData->h - 1 ) * ImData->w;
1156 :      if( ImData->bin[ L + S->x[ 5 ] + 0 ] == 1 ) S->x[ 2 ] = S->x[ 5 ] + 0;
1157 :      if( ImData->bin[ L + S->x[ 5 ] + 1 ] == 1 ) S->x[ 2 ] = S->x[ 5 ] + 1;
1158 :      if( ImData->bin[ L + S->x[ 5 ] + 0 ] == 1 &&
1159 :          ImData->bin[ L + S->x[ 5 ] + 1 ] == 1  ) {
1160 :          for( X = ( S->x[ 5 ] + 1 ); X < ( ImData->w - 3 ); X++ ) {
1161 :              L = ( ( ImData->h - 1 ) * ImData->w ) + X;
```

```
            if( ImData->bin[ L - 1 ] == 1 &&
                ImData->bin[ L + 0 ] == 1 &&
                ImData->bin[ L + 1 ] == 0 &&
                ImData->bin[ L + 2 ] == 0 ) {
                S->x[ 2 ] = X;
                break;
            }
        }
    }
    CX = S->x[ 5 ] + ( ( S->x[ 2 ] - S->x[ 5 ] ) >> 1 ) ;

    // Error ( bit0 ) : Not "Micon car" in center of the course.
    if( CX < ( ( ImData->w >> 1 ) - 5 ) || ( ( ImData->w >> 1 ) + 5 ) < CX ) ErrorCode |= 0x01;

    // Error ( bit1 ) : Not white line in center of the screen.
    for( Y = ( ImData->h - 1 ); Y >= ( PL - 1 ); Y-- ) {
        // Left side
        L = Y * ImData->w;
        for( X = CX; X > 1; X-- ) {
            if( ImData->bin[ L + X - 0 ] == 0 &&
                ImData->bin[ L + X + 1 ] == 0 &&
                ImData->bin[ L + X + 2 ] == 1 &&
                ImData->bin[ L + X + 3 ] == 1 ) {
                S->x[ 5 ] = X;
                break;
            }
        }
        // Right side
        for( X = CX; X < ( ImData->h - 2 ); X++ ) {
            if( ImData->bin[ L + X - 3 ] == 1 &&
                ImData->bin[ L + X - 2 ] == 1 &&
                ImData->bin[ L + X - 1 ] == 0 &&
                ImData->bin[ L + X + 0 ] == 0 ) {
                S->x[ 2 ] = X;
                break;
            }
        }
        CX   = S->x[ 5 ] + ( ( S->x[ 2 ] - S->x[ 5 ] ) >> 1 ) ;
        S->y = Y;

        L = ( ( Y - 1 ) * ImData->w ) + CX;
        if( ImData->bin[ L ] == 0 ) {
            ErrorCode |= 0x02;
            break;
        }
    }
    if( !ErrorCode ){
        // Center
        Gap = ( ( S->x[ 2 ] - S->x[ 5 ] ) / 3 ) + offset;
        S->x[ 4 ] = S->x[ 5 ] + Gap;
        S->x[ 3 ] = S->x[ 2 ] - Gap;
        // Left side
        S->x[ 6 ] = S->x[ 5 ] - Gap;
        S->x[ 7 ] = S->x[ 6 ] - Gap;
        // Error ( bit2 ) : Set outside of a screen ( X-axis of left side )
        if( S->x[ 5 ] < 0 || S->x[ 6 ] < 0 || S->x[ 7 ] < 0 ) {
            ErrorCode |= 0x04;
        }
        // Right side
        S->x[ 1 ] = S->x[ 2 ] + Gap;
        S->x[ 0 ] = S->x[ 1 ] + Gap;
        // Error ( bit3 ) : Set outside of a screen. ( X-axis of right side )
        if( S->x[ 2 ] > ( ImData->w - 1 ) || S->x[ 1 ] > ( ImData->w - 1 ) || S->x[ 0 ] > ( ImData->w -
1 ) ) {
            ErrorCode |= 0x08;
        }
    }
    return ErrorCode;
}

//---------------------------------------------------------------------//
// Sensor Process Function ( Interrupt Function )
// ImData     : mcrMat
// S          : SENSOR
//---------------------------------------------------------------------//
void SensorPro( mcrMat *ImData, SENSOR *S )
{
    long            L;
    int             i;
    unsigned char   sensor;

    sensor = 0;
    for( i = 0; i < 8; i++ ) {
        L = ( S->y * ImData->w ) + S->x[ i ];
        sensor |= ( ImData->bin[ L ] & 0x01 ) << i;
        ImData->r[ L ] = 255;
        ImData->g[ L ] =   0;
        ImData->b[ L ] =   0;
    }
    S->v = sensor & 0xff;
}
```

```
1252 :
1253 : //-----------------------------------------------------------------//
1254 : // Sensor Input Function
1255 : // S          : SENSOR
1256 : // Return     : Sensor Information( 8bit )
1257 : //-----------------------------------------------------------------//
1258 : unsigned char sensor_inp( SENSOR *S, unsigned char mask )
1259 : {
1260 :     return ( S->v & 0xff ) & mask;
1261 : }
1262 :
1263 : //-----------------------------------------------------------------//
1264 : // Start bar get Function
1265 : // Return values: Sensor value, ON (bar present):1,
1266 : // OFF (no bar present):0
1267 : //-----------------------------------------------------------------//
1268 : int startbar_get( void )
1269 : {
1270 :     unsigned char b;
1271 :     int ret;
1272 :
1273 :     ret = 0;
1274 :     b = sensor_inp( &sensor1, MASK3_3 ); /* Read start bar signal */
1275 :     if( b == 0xe7 ) {
1276 :         ret = 1;
1277 :     }
1278 :
1279 :     return ret;
1280 : }
1281 :
1282 : //-----------------------------------------------------------------//
1283 : // Check cross line Function
1284 : // Return values: 0: no cross line, 1: cross line
1285 : //-----------------------------------------------------------------//
1286 : int check_crossline( void )
1287 : {
1288 :     unsigned char b;
1289 :     int ret;
1290 :
1291 :     ret = 0;
1292 :     b = sensor_inp( &sensor0, MASK3_3);
1293 :     if( b==0xe7 ) {
1294 :         ret = 1;
1295 :     }
1296 :     return ret;
1297 : }
1298 :
1299 : //-----------------------------------------------------------------//
1300 : // Check right line Function
1301 : // Return values: 0: not detected, 1: detected
1302 : //-----------------------------------------------------------------//
1303 : int check_rightline( void )
1304 : {
1305 :     unsigned char b;
1306 :     int ret;
1307 :
1308 :     ret = 0;
1309 :     b = sensor_inp( &sensor0, MASK4_4);
1310 :     if( b==0x1f ) {
1311 :         ret = 1;
1312 :     }
1313 :     return ret;
1314 : }
1315 :
1316 : //-----------------------------------------------------------------//
1317 : // Check left line Function
1318 : // Return values: 0: not detected, 1: detected
1319 : //-----------------------------------------------------------------//
1320 : int check_leftline( void )
1321 : {
1322 :     unsigned char b;
1323 :     int ret;
1324 :
1325 :     ret = 0;
1326 :     b = sensor_inp( &sensor0, MASK4_4);
1327 :     if( b==0xf8 ) {
1328 :         ret = 1;
1329 :     }
1330 :     return ret;
1331 : }
1332 :
1333 : //*****************************************************************//
1334 : // Debug Functions
1335 : //*****************************************************************/
1336 : //-----------------------------------------------------------------//
1337 : // Serial Out ( TeraTerm )
1338 : // ImData     : mcrMat
1339 : // Mode       : FULL   -> Full screen
1340 : //              TOP    -> Top screen
1341 : //              BOTTOM -> Bottom screen
1342 : //-----------------------------------------------------------------//
```

```
1343 :  void SerialOut_TeraTerm( mcrMat *ImData, int Mode )
1344 :  {
1345 :      volatile long   X, Y, H;
1346 :      volatile int    i;
1347 :
1348 :      switch( Mode ) {
1349 :      case 0:
1350 :          // FULL
1351 :          H = ImData->h;
1352 :          for( Y = 0; Y < ImData->h; Y++ ) {
1353 :              for( X = 0; X < ImData->w; X++ ) {
1354 :                  pc.printf( "%d ", ImData->bin[ ( Y * ImData->w ) + X ] );
1355 :              }
1356 :              if( Y == sensor1.y ) pc.printf( "%2d sensor1 ", sensor1.y );
1357 :              if( Y == sensor0.y ) pc.printf( "%2d sensor0 ", sensor0.y );
1358 :              pc.printf( "¥n¥r" );
1359 :          }
1360 :          break;
1361 :      case 1:
1362 :          // TOP
1363 :          H = ImData->h >> 1;
1364 :          for( Y = 0; Y < ( ImData->h >> 1 ); Y++ ) {
1365 :              for( X = 0; X < ImData->w; X++ ) {
1366 :                  pc.printf( "%d", ImData->bin[ ( Y * ImData->w ) + X ] );
1367 :              }
1368 :              if( Y == sensor1.y ) pc.printf( "%2d sensor1 ", sensor1.y );
1369 :              if( Y == sensor0.y ) pc.printf( "%2d sensor0 ", sensor0.y );
1370 :              pc.printf( "¥n¥r" );
1371 :          }
1372 :          break;
1373 :      case 2:
1374 :          // BOTTOM
1375 :          H = ImData->h >> 1;
1376 :          for( Y = ( ImData->h >> 1 ); Y < ImData->h; Y++ ) {
1377 :              for( X = 0; X < ImData->w; X++ ) {
1378 :                  pc.printf( "%d", ImData->bin[ ( Y * ImData->w ) + X ] );
1379 :              }
1380 :              if( Y == sensor1.y ) pc.printf( "%2d sensor1 ", sensor1.y );
1381 :              if( Y == sensor0.y ) pc.printf( "%2d sensor0 ", sensor0.y );
1382 :              pc.printf( "¥n¥r" );
1383 :          }
1384 :          break;
1385 :      default:
1386 :          break;
1387 :      }
1388 :
1389 :      // Add display( sensor )
1390 :      pc.printf( "¥n¥r" );
1391 :      for( Y = sensor1.y, X = 0, i = 7; i >= 0; i-- ) {
1392 :          for(  ; X < sensor1.x[ i ]; X++ ) pc.printf( "  " );
1393 :          pc.printf( "%1d ", ImData->bin[ ( Y * ImData->w ) + X ] );
1394 :          X++;
1395 :      } pc.printf( "¥n¥r" );
1396 :      for( Y = sensor0.y, X = 0, i = 7; i >= 0; i-- ) {
1397 :          for(  ; X < sensor0.x[ i ]; X++ ) pc.printf( "  " );
1398 :          pc.printf( "%1d ", ImData->bin[ ( Y * ImData->w ) + X ] );
1399 :          X++;
1400 :      } pc.printf( "¥n¥r" );
1401 :      H += 3;
1402 :
1403 :      // Add display
1404 :      pc.printf( "¥n¥r" );
1405 :     pc.printf("ThresholdBuff = %3d, Min = %3d, Max = %3d¥n¥r", ThresholdBuff, ThresholdMin, ThresholdMax );
1406 :      pc.printf( "startbar_get  = %1d ¥n¥r", startbar_get() );
1407 :      pc.printf( "sensor_inp    = 0x%02x ¥n¥r", sensor_inp( &sensor0, 0xff ) );
1408 :      pc.printf( "¥n¥r" );
1409 :      H += 5;
1410 :
1411 :      pc.printf( "¥033[%dA" , H );
1412 :  }
1413 :
1414 :  //---------------------------------------------------------------------//
1415 :  // Serial Out ( Excel )
1416 :  // ImData     : mcrMat
1417 :  // Mode       : 0 -> Y data
1418 :  //              1 -> R data
1419 :  //              2 -> G data
1420 :  //              3 -> B data
1421 :  //              4 -> H data
1422 :  //              5 -> S data
1423 :  //              6 -> L data
1424 :  //---------------------------------------------------------------------//
1425 :  void SerialOut_Excel( mcrMat *ImData, int Mode )
1426 :  {
1427 :      volatile long   X, Y;
1428 :
1429 :      for( Y = 0; Y < ImData->h; Y++ ) {
1430 :          for( X = 0; X < ImData->w; X++ ) {
1431 :              if     ( Mode == 0 ) pc.printf( "%3d,", ImData->y[ ( Y * ImData->w ) + X ] );
1432 :              else if( Mode == 1 ) pc.printf( "%3d,", ImData->r[ ( Y * ImData->w ) + X ] );
1433 :              else if( Mode == 2 ) pc.printf( "%3d,", ImData->g[ ( Y * ImData->w ) + X ] );
```

```
1434 :              else if( Mode == 3 ) pc.printf( "%3d,", ImData->b[ ( Y * ImData->w ) + X ] );
1435 :              else if( Mode == 4 ) pc.printf( "%3d,", ImData->H[ ( Y * ImData->w ) + X ] );
1436 :              else if( Mode == 5 ) pc.printf( "%3d,", ImData->S[ ( Y * ImData->w ) + X ] );
1437 :              else if( Mode == 6 ) pc.printf( "%3d,", ImData->L[ ( Y * ImData->w ) + X ] );
1438 :          }
1439 :          pc.printf("¥n¥r");
1440 :      }
1441 :  }
1442 :
1443 :  //-------------------------------------------------------------------//
1444 :  // Serial Out ( csv -> jpg )
1445 :  // ImData      : mcrMat
1446 :  // Format      : VIDEO_YCBCR422 or VIDEO_RGB888
1447 :  // ColorPattern: COLOR
1448 :  //               BINARY
1449 :  //               COLOR_BINARY
1450 :  //-------------------------------------------------------------------//
1451 :  void SerialOut_CsvJpg( mcrMat *ImData, int Format, int ColorPattern )
1452 :  {
1453 :      volatile long   X, Y, L, D;
1454 :
1455 :      pc.printf( "//,X-Size,Y-Size" );
1456 :      pc.printf( "¥n¥r" );
1457 :      pc.printf( "#SIZE,%3d,%3d", ImData->w, ImData->h );
1458 :      pc.printf( "¥n¥r" );
1459 :      if     ( Format == VIDEO_YCBCR422 ) pc.printf( "//,X-Point,Y-Point,Y,CB,CR" );
1460 :      else if( Format == VIDEO_RGB888   ) pc.printf( "//,X-Point,Y-Point,R,G,B" );
1461 :      pc.printf( "¥n¥r" );
1462 :
1463 :      switch( ColorPattern ) {
1464 :      case COLOR:
1465 :          for( Y = 0; Y < ImData->h; Y++ ){
1466 :              L = ImData->w * Y;
1467 :              for( X = 0; X < ImData->w; X++ ){
1468 :                  D = L + X;
1469 :                  if     ( Format == VIDEO_YCBCR422 ) pc.printf( "#YCbCr," );
1470 :                  else if( Format == VIDEO_RGB888   ) pc.printf( "#RGB," );
1471 :                  pc.printf( "%3d,", X);
1472 :                  pc.printf( "%3d,", Y);
1473 :                  if( Format == VIDEO_YCBCR422 ) {
1474 :                      pc.printf( "%3d,", ImData->y [ D ] );
1475 :                      pc.printf( "%3d,", ImData->cb[ D ] );
1476 :                      pc.printf( "%3d,", ImData->cr[ D ] );
1477 :                  } else if( Format == VIDEO_RGB888 ) {
1478 :                      pc.printf( "%3d,", ImData->r[ D ] );
1479 :                      pc.printf( "%3d,", ImData->g[ D ] );
1480 :                      pc.printf( "%3d,", ImData->b[ D ] );
1481 :                  }
1482 :                  pc.printf( "¥n¥r" );
1483 :              }
1484 :          }
1485 :          break;
1486 :
1487 :      case BINARY:
1488 :          for( Y = 0; Y < ImData->h; Y++ ){
1489 :              L = ImData->w * Y;
1490 :              for( X = 0; X < ImData->w; X++ ){
1491 :                  D = L + X;
1492 :                  if     ( Format == VIDEO_YCBCR422 ) pc.printf( "#YCbCr," );
1493 :                  else if( Format == VIDEO_RGB888   ) pc.printf( "#RGB," );
1494 :                  pc.printf( "%3d,", X);
1495 :                  pc.printf( "%3d,", Y);
1496 :                  if( Format == VIDEO_YCBCR422 ) {
1497 :                      if( ImData->bin[ D ] ) {
1498 :                          pc.printf( "%3d,", 255 );
1499 :                          pc.printf( "%3d,", 128 );
1500 :                          pc.printf( "%3d,", 128 );
1501 :                      } else {
1502 :                          pc.printf( "%3d,",   0 );
1503 :                          pc.printf( "%3d,", 128 );
1504 :                          pc.printf( "%3d,", 128 );
1505 :                      }
1506 :                  } else if( Format == VIDEO_RGB888 ) {
1507 :                      if( ImData->bin[ D ] ) {
1508 :                          pc.printf( "%3d,", 255 );
1509 :                          pc.printf( "%3d,", 255 );
1510 :                          pc.printf( "%3d,", 255 );
1511 :                      } else {
1512 :                          pc.printf( "%3d,",   0 );
1513 :                          pc.printf( "%3d,",   0 );
1514 :                          pc.printf( "%3d,",   0 );
1515 :                      }
1516 :                  }
1517 :                  pc.printf( "¥n¥r" );
1518 :              }
1519 :          }
1520 :          break;
1521 :
1522 :      case COLOR_BINARY:
1523 :          for( Y = 0; Y < ImData->h; Y++ ){
1524 :              L = ImData->w * Y;
```

```
1525 :              for( X = 0; X < ImData->w; X++ ){
1526 :                  D = L + X;
1527 :                  if     ( Format == VIDEO_YCBCR422 ) pc.printf( "#YCbCr," );
1528 :                  else if( Format == VIDEO_RGB888   ) pc.printf( "#RGB," );
1529 :                  pc.printf( "%3d,", X);
1530 :                  pc.printf( "%3d,", Y);
1531 :                  if( Format == VIDEO_YCBCR422 ) {
1532 :                      if( ImData->bin[ D ] ) {
1533 :                          pc.printf( "%3d,", ImData->y [ D ] );
1534 :                          pc.printf( "%3d,", ImData->cb[ D ] );
1535 :                          pc.printf( "%3d,", ImData->cr[ D ] );
1536 :                      } else {
1537 :                          pc.printf( "%3d,",   0 );
1538 :                          pc.printf( "%3d,", 128 );
1539 :                          pc.printf( "%3d,", 128 );
1540 :                      }
1541 :                  } else if( Format == VIDEO_RGB888 ) {
1542 :                      if( ImData->bin[ D ] ) {
1543 :                          pc.printf( "%3d,", ImData->r[ D ] );
1544 :                          pc.printf( "%3d,", ImData->g[ D ] );
1545 :                          pc.printf( "%3d,", ImData->b[ D ] );
1546 :                      } else {
1547 :                          pc.printf( "%3d,",   0 );
1548 :                          pc.printf( "%3d,",   0 );
1549 :                          pc.printf( "%3d,",   0 );
1550 :                      }
1551 :                  }
1552 :                  pc.printf( "¥n¥r" );
1553 :              }
1554 :          }
1555 :          break;
1556 :
1557 :      default:
1558 :          break;
1559 :      }
1560 :      pc.printf( "End¥n¥r" );
1561 :  }
1562 :
1563 :  //-----------------------------------------------------------------------//
1564 :  // Debug Process Functions
1565 :  //-----------------------------------------------------------------------//
1566 :  void DebugPro( void )
1567 :  {
1568 :      volatile int    Number;             /* Serial Debug Mode only   */
1569 :
1570 :      pc.printf( "Serial Debug Mode ¥n¥r" );
1571 :      pc.printf( "¥n¥r" );
1572 :      #if(VIDEO_FORMAT == VIDEO_YCBCR422)
1573 :      pc.printf( "VIDEO_FORMAT = VIDEO_YCBCR422 ¥n¥r" );
1574 :      #endif
1575 :      #if(VIDEO_FORMAT == VIDEO_RGB888)
1576 :      pc.printf( "VIDEO_FORMAT = VIDEO_RGB888 ¥n¥r" );
1577 :      #endif
1578 :      pc.printf( "¥n¥r" );
1579 :      pc.printf( "0:TeraTram Real-time display (Binary)¥n¥r" );
1580 :      pc.printf( "1:Excel(csv) (Y)(R)(G)(B)¥n¥r" );
1581 :      pc.printf( "2:Excel(csv) -> csv_jpg_convert.bat (Color)¥n¥r" );
1582 :      pc.printf( "3:Excel(csv) -> csv_jpg_convert.bat (Binary)¥n¥r" );
1583 :      pc.printf( "4:Excel(csv) -> csv_jpg_convert.bat (Color - Binary)¥n¥r" );
1584 :      pc.printf( "5:Run time for Functions¥n¥r" );
1585 :      pc.printf( "¥n¥r" );
1586 :      pc.printf( "Please Number¥n¥r" );
1587 :      pc.printf( "No = " );
1588 :      pc.scanf( "%d", &Number );
1589 :      pc.printf( "¥n¥r" );
1590 :      pc.printf( "Please push the SW ( on the Motor drive board )¥n¥r" );
1591 :      pc.printf( "¥n¥r" );
1592 :      while( !pushsw_get() );
1593 :      while( PNumber <= 15 );
1594 :      ImgBuff = ImgInp1;
1595 :
1596 :      switch( Number ) {
1597 :      case 0:
1598 :          // TeraTram Real-time display (Binary)
1599 :          while( 1 ) SerialOut_TeraTerm( &ImgInp1, 0 );
1600 :          break;
1601 :      case 1:
1602 :          // Excel(csv)
1603 :          pc.printf("Excel(csv)(Y)¥n¥r");
1604 :          pc.printf("¥n¥r");
1605 :          SerialOut_Excel( &ImgBuff, 0 );
1606 :          pc.printf("¥n¥r");
1607 :
1608 :          pc.printf("Excel(csv)(R)¥n¥r");
1609 :          pc.printf("¥n¥r");
1610 :          SerialOut_Excel( &ImgBuff, 1 );
1611 :          pc.printf("¥n¥r");
1612 :
1613 :          pc.printf("Excel(csv)(G)¥n¥r");
1614 :          pc.printf("¥n¥r");
1615 :          SerialOut_Excel( &ImgBuff, 2 );
```

```
1616 :          pc.printf("¥n¥r");
1617 :
1618 :          pc.printf("Excel(csv)(B)¥n¥r");
1619 :          pc.printf("¥n¥r");
1620 :          SerialOut_Excel( &ImgBuff, 3 );
1621 :          pc.printf("¥n¥r");
1622 :          break;
1623 :      case 2:
1624 :          // Excel(csv) -> csv_jpg_convert.bat (Color)
1625 :          SerialOut_CsvJpg( &ImgBuff, VIDEO_FORMAT, COLOR );
1626 :          break;
1627 :      case 3:
1628 :          // Excel(csv) -> csv_jpg_convert.bat (Binary)
1629 :          SerialOut_CsvJpg( &ImgBuff, VIDEO_FORMAT, BINARY );
1630 :          break;
1631 :      case 4:
1632 :          // Excel(csv) -> csv_jpg_convert.bat (Color Binary)
1633 :          SerialOut_CsvJpg( &ImgBuff, VIDEO_FORMAT, COLOR_BINARY );
1634 :          break;
1635 :      case 5:
1636 :          // Run time for Functions
1637 :          // Change of interrupt time ( 100us )
1638 :          DebugMode = 1;
1639 :          interrput.attach(&intTimer, 0.0001);
1640 :
1641 :          do {
1642 :              if( field_buff != field ) {
1643 :                  field_buff  = field;
1644 :                  PNumber = 0;
1645 :              }
1646 :          } while( PNumber != 0 );
1647 :
1648 :          do {
1649 :              pc.printf( "intImagePro()  case %2d:  RunTime = ", PNumber );
1650 :              cnt0 = 0;
1651 :              intImagePro();
1652 :              pc.printf( "%3d00us ¥n¥r", cnt0 );
1653 :          } while( PNumber < 16 );
1654 :
1655 :          // Change of interrupt time ( 1ms )
1656 :          interrput.attach(&intTimer, 0.001);
1657 :          DebugMode = 0;
1658 :          break;
1659 :      default:
1660 :          break;
1661 :      }
1662 :      while(1);
1663 :  }
1664 :
1665 :  //------------------------------------------------------------------//
1666 :  // End of file
1667 :  //------------------------------------------------------------------//
```

6.2 プログラムの解説

6.2.1 スタート

```
 1 :  //-------------------------------------------------------------------//
 2 :  //Supported MCU :  RZ/A1H
 3 :  //File Contents :  kit20_gr-peach ( Trace Program )
 4 :  //Version number:  Ver.1.00
 5 :  //Date:            2020.09.01
 6 :  //Copyright:       Renesas Electronics Corporation
 7 :  //                 Hitachi Document Solutions Co., Ltd.
 8 :  //-------------------------------------------------------------------//
 9 :
10 :  //This program supports the following kit:
11 :  //* M-S348 Image processing micon car production kit
```

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

6.2.2 外部ファイルの取り込み（インクルード）

```
13 :  //-------------------------------------------------------------------//
14 :  //Include
15 :  //-------------------------------------------------------------------//
16 :  #include "mbed.h"
17 :  #include "iodefine.h"
18 :  #include "DisplayBace.h"
19 :  #include "ImageProcessFunc.h"
20 :  #include "EasyAttach_CameraAndLCD.h"
```

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

ファイル名	内容
mbed.h	RZ/A1H マイコンの内蔵周辺機能を制御するためのレジスタと定義したファイルです。他に、標準の API が定義されています。
iodefine.h	RZ/A1H マイコンの内蔵周辺機能を制御するための I/O レジスタを定義したファイルです。
DisplayBace.h	カメラ設定用の API を定義したファイルです。
ImageProcessFunc.h	マイコンカーラリー用の画像処理関数を定義したファイルです。
EasyAttach_CameraAndLCD.h	カメラ、LCD 用の関数を定義したファイルです。

6.2.3 シンボル定義

```
22 :  //-------------------------------------------------------------------//
23 :  //Define
24 :  //-------------------------------------------------------------------//
25 :  //Motor PWM cycle
26 :  #define     MOTOR_PWM_CYCLE     33332   /* Motor PWM period         */
27 :                                          /* 1ms    PO φ/1  = 0.03us   */
28 :  //Servo PWM cycle
29 :  #define     SERVO_PWM_CYCLE     33332   /* SERVO PWM period         */
30 :                                          /* 16ms   PO φ/16 = 0.48us   */
31 :  #define     SERVO_CENTER        3124    /* 1.5ms / 0.48us - 1 = 3124*/
32 :  #define     HANDLE_STEP         18      /* 1 degree value           */
33 :
34 :  #define     THRESHOLD           128     /* Set 0 to 255 Black:0 White:255 */
35 :
36 :  //Masked value settings X:masked (disabled) O:not masked (enabled)
37 :  #define     MASK2_2             0x66    /* X O O X  X O O X         */
38 :  #define     MASK2_0             0x60    /* X O O X  X X X X         */
39 :  #define     MASK0_2             0x06    /* X X X X  X O O X         */
40 :  #define     MASK3_3             0xe7    /* O O O X  X O O O         */
41 :  #define     MASK0_3             0x07    /* X X X X  X O O O         */
42 :  #define     MASK3_0             0xe0    /* O O O X  X X X X         */
43 :  #define     MASK4_0             0xf0    /* O O O O  X X X X         */
44 :  #define     MASK0_4             0x0f    /* X X X X  O O O O         */
45 :  #define     MASK4_4             0xff    /* O O O O  O O O O         */
```

MOTOR_PWM_CYCLE	右モータ、左モータに加える PWM 周期を「MOTOR_PWM_CYCLE」という名前で定義します。今回、PWM 周期は 1ms にします。 MOTOR_PWM_CYCLE に設定する値は下記のようになります。 MOTOR_PWM_CYCLE ＝ 周期 ／ カウントソース － 1 MOTOR_PWM_CYCLE ＝ (1×10-3) ／ (30×10-9) - 1 MOTOR_PWM_CYCLE ＝ 333333 - 1 ＝ 33332
SERVO_PWM_CYCLE	サーボに加える PWM 周期を「SERVO_PWM_CYCLE」という名前で定義します。今回、PWM 周期は 16ms にします。 SERVO_PWM_CYCLE に設定する値は下記のようになります。 SERVO_PWM_CYCLE ＝ 周期 ／ カウントソース － 1 SERVO_PWM_CYCLE ＝ (16×10-3) ／ (480×10-9) - 1 SERVO_PWM_CYCLE ＝ 333333 - 1 ＝ 33332

SERVO_CENTER	サーボが 0 度(まっすぐ向く角度)のときの値を「SERVO_CENTER」という名前で定義します。標準的なサーボは、1.5[ms]のパルス幅を加えるとまっすぐ向きます。よって、パルスの ON 幅を 1.5ms に設定します。SERVO_CENTER に設定する値は下記のようになります。 SERVO_CENTER＝パルスの ON 幅 ／ カウントソース － 1 SERVO_CENTER＝(1.5×10-3)／(480×10-9)－1 SERVO_CENTER＝3125－1 ＝ 3124 よって、SERVO_CENTER を 3124 として定義します。実際はサーボに個体差があり、3124 がまっすぐ向く場合はほとんどありません。サーボのセンタ値を調整する場合は、この値を変更してください。 しかし、サーボのセンタは**サーボ自体の誤差、サーボホーンに挿すギザギザのかみ合わせ方などの影響ですべてのマイコンカーで違う値になります**。例えるなら、人間の指紋のようなものです。そのため、この値は**プログラムでハンドルを 0 度にしたときに、マイコンカーがまっすぐ走るようにマイコンカー1 台ごとに調整、変更します**。  ▲サーボホーン
HANDLE_STEP	サーボが 1 度分動く値を「HANDLE_STEP」という名前で定義します。 左 90 度の PWM の ON 幅は、0.7ms です。右 90 度の PWM の ON 幅は、2.3ms です。この差分を 180 で割ると、1 度当たりの値が計算できます。 ● 左 90 度の PWM の ON 幅 MTU2TGRB_0＝P4_0 端子から出力される PWM 波形の ON 幅／タイマカウンタのカウントソース－1 ＝(0.7×10-3)／(480×10-9)－1 ＝1458-1 ＝ 1457 ● 右 90 度の PWM の ON 幅 MTU2TGRB_0＝P4_0 端子から出力される PWM 波形の ON 幅／タイマカウンタのカウントソース－1 ＝(2.3×10-3)／(400×10-9)－1 ＝4791-1 ＝ 4790 ● 1 度当たりの値 (右－左)/180 ＝ (4790－1457)/180 ＝ 18.52 ≒ 18 よって、HANDLE_STEP を 18 として定義します。1 度当たりの値を変更する場合は、この値を変更してください。
THRESHOLD	しきい値を「THRESHOLD」という名前で定義します。設定できる値は、0(黒)～255(白)です。初期値は、128 に設定します。
MASK2_2 MASK2_0 MASK0_2 MASK3_3 MASK0_3 MASK3_0 MASK4_0 MASK0_4 MASK4_4	sensor_inp 関数でセンサの値をマスクするとき、よく使うマスク値を定義しています。 「MASK」＋「A」＋「__(アンダバー)」＋「B」として定義しています。意味は下記のようになります。 ・A…左のセンサ 4 個中 A 個を有効にする ・B…右のセンサ 4 個中 B 個を有効にする ・その他をマスクする

6.2.4 プロトタイプ宣言

```
 137 :  //-----------------------------------------------------------------//
 138 :  // Prototype
 139 :  //-----------------------------------------------------------------//
 140 :  // Peripheral Functions
 141 :  void init_MTU2_PWM_Motor( void );       /* Initialize PWM Functions */
 142 :  void init_MTU2_PWM_Servo( void );       /* Initialize PWM Functions */
 143 :
 144 :  // Interrupt Function
 145 :  void intTimer( void );                  /* 1ms period               */
 146 :  void intImagePro( void );
 147 :
 148 :  // Micon board ( GR-PEACH )
 149 :  void led_m_rgb(int led);
 150 :  void led_m_user( int led );
 151 :  unsigned char user_button_get( void );
 152 :  void led_m_set( int set );
 153 :  void led_m_pro( void );                 /* Only Function for interrupt */
 154 :
 155 :  // Motor drive board
 156 :  void led_out(int led);
 157 :  unsigned char pushsw_get( void );
 158 :  void motor( int accele_l, int accele_r );
 159 :  void handle( int angle );
 160 :
 161 :  // Shield board
 162 :  unsigned char dipsw_get( void );
```

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

プログラム例を下記に示します。

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

void main( void )
{
    int a, b;

    a = 50;
    b = 100;

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

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

6.2.5 グローバル変数の宣言

```
206 :  //-----------------------------------------------------------------//
207 :  // Global Variable for Trace Program
208 :  //-----------------------------------------------------------------//
209 :  volatile unsigned long  PNumber;        /* intTimer Function Only   */
210 :  volatile int            DebugMode;      /* intTimer Function Only   */
211 :
212 :  volatile unsigned long  cnt0;           /* Used by timer Function   */
213 :  volatile unsigned long  cnt1;           /* Used within main         */
214 :  volatile int            pattern;        /* Pattern numbers          */
215 :
216 :  volatile int            ThresholdBuff;  /* ImageBinary Function only*/
217 :  volatile int            ThresholdMax;   /* ImageBinary Function only*/
218 :  volatile int            ThresholdMin;   /* ImageBinary Function only*/
219 :
220 :  volatile int            LedPattern;     /* led_m_pro Function only  */
221 :
222 :  // Led Set ( on GR-Peach board ) Function only
223 :  const int led_data[   ][ 3 ] = {
224 :   /* Color    , onTime, offTime */
225 :       { LED_GREEN ,     500,      500 },// 0:RUN
226 :       { LED_RED   ,     500,        0 },// 1:STOP
227 :       { LED_RED   ,     100,      100 },// 2:ERROR
228 :       { LED_BLUE  ,      50,       50 },// 3:DEBUG
229 :       { LED_YELLOW,     500,      500 },// 4:CRANK
230 :       { LED_BLUE  ,     500,      500 } // 5:LCHANGE
231 :  };
```

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

プログラム例を下記に示します。

```
void a( void );                         /* プロトタイプ宣言 */
int timer;                              /* グローバル変数   */

void main( void )
{
        int i;

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

Kit20_gr-peach.cpp プログラムでは、下記のグローバル変数を宣言しています。

変数名	型	使用方法
PNumber	unsigned long	割り込み関数専用の変数です。割り込み関数を実行する度に実行するプログラムの番号を格納します。
DebugMode	int	デバッグプログラムを実行中は「1」が格納されます。
cnt0	unsigned long	**この変数は、1ms ごとに+1 する変数です。**timer 関数で 1ms を数えるのに使用します。timer 関数部分で詳しく説明します。
cnt1	unsigned long	**この変数は、1ms ごとに+1 する変数です。**この変数は、プログラム作成者が自由に使って、時間を計ります。例えば、300ms たったなら○○をしなさい、たっていないなら□□をしなさい、というように使用します。main 関数部分で詳しく説明します。
pattern	int	パターン番号です。main 関数部分で詳しく説明します。
ThresholdBuff	int	しきい値を格納します。
ThresholdMax	int	しきい値の最大値を格納します。
ThresholdMin	int	しきい値の最小値を格納します。
LedPattern	int	GR-PEACH ボード上の LED 関数専用の変数です。LED の処理パターン番号を格納します。
led_data	const int	GR-PEACH ボード上の LED 関数専用の変数です。LED の色、点滅する速さを設定します。

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

6.2.6 init_MTU2_PWM_Motor 関数

init_MTU2_PWM_Motor 関数は、モータ用の PWM と I/O を設定する関数です。

```
725 :  //-----------------------------------------------------------------//
726 :  // Initialize MTU2 PWM Functions
727 :  //-----------------------------------------------------------------//
728 :  // MTU2_3, MTU2_4
729 :  // Reset-Synchronized PWM mode
730 :  // TIOC4A(P4_4) :Left-motor
731 :  // TIOC4B(P4_5) :Right-motor
732 :  //-----------------------------------------------------------------//
733 :  void init_MTU2_PWM_Motor( void )
734 :  {
735 :      // Port setting for S/W I/O Control
736 :      // alternative mode
737 :
738 :      // MTU2_4 (P4_4)(P4_5)
739 :      GPIOPBDC4   = 0x0000;              /* Bidirection mode disabled*/
740 :      GPIOPFCAE4 &= 0xffcf;              /* The alternative Function of a pin */
741 :      GPIOPFCE4  |= 0x0030;              /* The alternative Function of a pin */
742 :      GPIOPFC4   &= 0xffcf;              /* The alternative Function of a pin */
743 :                                         /* 2nd altemative Function/output    */
744 :      GPIOP4     &= 0xffcf;              /*                                   */
745 :      GPIOPM4    &= 0xffcf;              /* p4_4,P4_5:output                  */
746 :      GPIOPMC4   |= 0x0030;              /* P4_4,P4_5:double                  */
747 :
748 :      // Module stop 33(MTU2) canceling
749 :      CPGSTBCR3  &= 0xf7;
750 :
751 :      // MTU2_3 and MTU2_4 (Motor PWM)
752 :      MTU2TCR_3   = 0x20;                /* TCNT Clear(TGRA), P0φ/1  */
753 :      MTU2TOCR1   = 0x04;                /*                          */
754 :      MTU2TOCR2   = 0x40;                /* N L>H  P H>L             */
755 :      MTU2TMDR_3  = 0x38;                /* Buff:ON Reset-Synchronized PWM mode */
756 :      MTU2TMDR_4  = 0x30;                /* Buff:ON                  */
757 :      MTU2TOER    = 0xc6;                /* TIOC3B,4A,4B enabled output */
758 :      MTU2TCNT_3  = MTU2TCNT_4 = 0;      /* TCNT3,TCNT4 Set 0        */
759 :      MTU2TGRA_3  = MTU2TGRC_3 = MOTOR_PWM_CYCLE;
760 :                                         /* PWM-Cycle(1ms)           */
761 :      MTU2TGRA_4  = MTU2TGRC_4 = 0;      /* Left-motor(P4_4)         */
762 :      MTU2TGRB_4  = MTU2TGRD_4 = 0;      /* Right-motor(P4_5)        */
763 :      MTU2TSTR   |= 0x40;                /* TCNT_4 Start             */
764 :  }
```

6.2.7 init_MTU2_PWM_Servo 関数

init_MTU2_PWM_Servo 関数は、サーボ用の PWM と I/O を設定する関数です。

```
766 :  //-----------------------------------------------------------------//
767 :  // Initialize MTU2 PWM Functions
768 :  //-----------------------------------------------------------------//
769 :  // MTU2_0
770 :  // PWM mode 1
771 :  // TIOC0A(P4_0) :Servo-motor
772 :  //-----------------------------------------------------------------//
773 :  void init_MTU2_PWM_Servo( void )
774 :  {
775 :      // Port setting for S/W I/O Control
776 :      // alternative mode
777 :
778 :      // MTU2_0 (P4_0)
779 :      GPIOPBDC4   = 0x0000;              /* Bidirection mode disabled*/
780 :      GPIOPFCAE4 &= 0xfffe;              /* The alternative Function of a pin */
781 :      GPIOPFCE4  &= 0xfffe;              /* The alternative Function of a pin */
782 :      GPIOPFC4   |= 0x0001;              /* The alternative Function of a pin */
783 :                                         /* 2nd alternative Function/output   */
784 :      GPIOP4     &= 0xfffe;              /*                                   */
785 :      GPIOPM4    &= 0xfffe;              /* p4_0:output                       */
786 :      GPIOPMC4   |= 0x0001;              /* P4_0:double                       */
787 :
788 :      // Module stop 33(MTU2) canceling
789 :      CPGSTBCR3  &= 0xf7;
790 :
791 :      // MTU2_0 (Motor PWM)
792 :      MTU2TCR_0   = 0x22;                /* TCNT Clear(TGRA), P0φ/16 */
793 :      MTU2TIORH_0 = 0x52;                /* TGRA L>H, TGRB H>L       */
794 :      MTU2TMDR_0  = 0x32;                /* TGRC and TGRD = Buff-mode*/
795 :                                         /* PWM-mode1                */
796 :      MTU2TCNT_0  = 0;                   /* TCNT0 Set 0              */
797 :      MTU2TGRA_0  = MTU2TGRC_0 = SERVO_PWM_CYCLE;
798 :                                         /* PWM-Cycle(16ms)          */
799 :      MTU2TGRB_0  = MTU2TGRD_0 = 0;      /* Servo-motor(P4_0)        */
800 :      MTU2TSTR   |= 0x01;                /* TCNT_0 Start             */
801 :  }
```

6.2.8 Start_Video_Camera 関数

Start_Video_Camera 関数は、カメラの初期化する関数です。

```
803 :  //-----------------------------------------------------------------//
804 :  // Initialize Camera Function
805 :  //-----------------------------------------------------------------//
806 :  static void Start_Video_Camera( void )
807 :  {
808 :      // Initialize the background to black
809 :      for (uint32_t i = 0; i < sizeof(FrameBuffer_Video_A); i += 2) {
810 :          FrameBuffer_Video_A[i + 0] = 0x10;
811 :          FrameBuffer_Video_A[i + 1] = 0x80;
812 :      }
813 :      // Interrupt callback Function setting
                (Field end signal for recording Function in scaler 0)
814 :      Display.Graphics_Irq_Handler_Set(VIDEO_INT_TYPE, 0, IntCallbackFunc_Vfield);
815 :      // Video capture setting (progressive form fixed)
816 :      Display.Video_Write_Setting(
817 :          VIDEO_INPUT_CH,
818 :          DisplayBase::COL_SYS_NTSC_358,
819 :          write_buff_addr,
820 :          VIDEO_BUFFER_STRIDE,
821 :          #if(VIDEO_FORMAT == VIDEO_YCBCR422)
822 :          DisplayBase::VIDEO_FORMAT_YCBCR422,
823 :          DisplayBase::WR_RD_WRSWA_32_16BIT,
824 :          #endif
825 :          #if(VIDEO_FORMAT == VIDEO_RGB888)
826 :          DisplayBase::VIDEO_FORMAT_RGB888,
827 :          DisplayBase::WR_RD_WRSWA_32BIT,
828 :          #endif
829 :          PIXEL_VW,
830 :          PIXEL_HW
831 :      );
832 :      EasyAttach_CameraStart(Display, VIDEO_INPUT_CH);
833 :  }
```

6.2.1 IntCallbackFunc_Vfield 関数

IntCallbackFunc_Vfield 関数は、カメラモジュールから画像データの取得が完了すると実行される割り込み関数です。

```
835 :  //-----------------------------------------------------------------------//
836 :  // @brief       Interrupt callback Function
837 :  // @param[in]   int_type    : VDC5 interrupt type
838 :  // @retval      None
839 :  //-----------------------------------------------------------------------//
840 :  static void IntCallbackFunc_Vfield(DisplayBase::int_type_t int_type)
841 :  {
842 :      // Top or Bottom
843 :      if( field == BOTTOM ) field = TOP;
844 :      else                  field = BOTTOM;
845 :      // Led
846 :      led_m_user( field );
847 :  }
```

6.2.2 intTimer関数（1msごとの割り込み）

intTimer 関数は、1ms ごとに実行されます。

```
849 :  //---------------------------------------------------------------------//
850 :  // Interrupt Function( intTimer )
851 :  //---------------------------------------------------------------------//
852 :  void intTimer( void )
853 :  {
854 :      cnt0++;
855 :      cnt1++;
856 :
857 :      if( !DebugMode ) {
858 :          if( field_buff != field ) {
859 :              field_buff  = field;
860 :              PNumber = 0;
861 :          }
862 :          intImagePro();
863 :      }
864 :
865 :      led_m_pro();
866 :  }
```

852 行	割り込みにより実行する関数です。割り込み関数は、引数、戻り値ともに指定することはできません。すなわち、「void 関数名(void)」である必要があります。
854 行	cnt0 変数を+1 します。この関数は 1ms ごとに実行されるので、cnt0 変数は 1ms ごとに+1 されることになります。
855 行	cnt1 変数を+1 します。この関数は 1ms ごとに実行されるので、cnt1 変数は 1ms ごとに+1 されることになります。
858 行 ～ 861 行	読み込む画像のフィールドを切り替えます。奇数フィールドの場合は、偶数フィールドに、偶数フィールドの場合は、奇数フィールドに切り替えます。
862 行	画像処理関数です。画像からセンサ情報を取得します。
865 行	GR-PEACH ボード上の LED を制御します。

6.2.3 intImagePro 関数

intImagePro 関数は、カメラから画像データを取得し、画像データのリサイズ(縮小)、しきい値の自動計算、二値化処理、センサ情報の取得をする関数です。

```
868 :  //---------------------------------------------------------------------//
869 :  // Interrupt Function( intImagePro )
870 :  //---------------------------------------------------------------------//
871 :  void intImagePro( void )
872 :  {
873 :      switch( PNumber++ ) {
874 :
875 :      #if(VIDEO_FORMAT == VIDEO_YCBCR422)
876 :      case 0:
877 :          // Size( 160 * 120 ) -> ( 80 * 60 ) Top field or Bottom field
878 :          intImageCopy( write_buff_addr, PIXEL_HW, PIXEL_VW, VIDEO_FORMAT,
                                              field, &ImgInp0, ICS, INTERMITTENT_PRO4 );
879 :          break;
880 :      case 1:
881 :          intImageCopy( write_buff_addr, PIXEL_HW, PIXEL_VW, VIDEO_FORMAT,
                                              field, &ImgInp0, ICS, INTERMITTENT_PRO4 );
882 :          break;
883 :      case 2:
884 :          intImageCopy( write_buff_addr, PIXEL_HW, PIXEL_VW, VIDEO_FORMAT,
                                              field, &ImgInp0, ICS, INTERMITTENT_PRO4 );
885 :          break;
886 :      case 3:
887 :          intImageCopy( write_buff_addr, PIXEL_HW, PIXEL_VW, VIDEO_FORMAT,
                                              field, &ImgInp0, ICS, INTERMITTENT_PRO4 );
888 :          break;
889 :      case 4:
890 :          // Size( 80 * 60 ) -> ( 40 * 30 )
891 :          intImageReduction( &ImgInp0, &ImgInp1, VIDEO_FORMAT, Rate,
                                                                  INTERMITTENT_PRO2 );
892 :          break;
893 :      case 5:
894 :          intImageReduction( &ImgInp0, &ImgInp1, VIDEO_FORMAT, Rate,
                                                                  INTERMITTENT_PRO2 );
895 :          break;
896 :      case 6:
897 :          // Automatic calculation of threshold value ( YCBCR422 Only function )
898 :          ThresholdBuff = DiscriminantAnalysisMethod( &ImgInp1 );
899 :          // Threshold Max
900 :          ThresholdMax  = ThresholdBuff + 255;
901 :          if( ThresholdMax > 255 ) ThresholdMax = 255;
902 :          // Threshold Min
903 :          ThresholdMin  = ThresholdBuff + 0;
904 :          if( ThresholdMin  <=  50 ) ThresholdMin =  50;
905 :          break;
906 :      case 7:
907 :          // Binary process
908 :          ImageBinarySetY( ThresholdMin, ThresholdMax );
909 :          intImageBinary( &ImgInp1, &ImgInp1, BINARY_Y, INTERMITTENT_PRO1 );
910 :          SensorPro( &ImgInp1, &sensor1 );
911 :          SensorPro( &ImgInp1, &sensor0 );
```

```
912 :          break;
913 :      #endif
914 :
915 :      #if(VIDEO_FORMAT == VIDEO_RGB888)
916 :      case 0:
917 :          // Size( 160 * 120 ) -> ( 80 * 60 ) Top field or Bottom field
918 :          intImageCopy( write_buff_addr, PIXEL_HW, PIXEL_VW, VIDEO_FORMAT,
                                              field, &ImgInp0, ICS, INTERMITTENT_PRO4 );
919 :          break;
920 :      case 1:
921 :          intImageCopy( write_buff_addr, PIXEL_HW, PIXEL_VW, VIDEO_FORMAT,
                                              field, &ImgInp0, ICS, INTERMITTENT_PRO4 );
922 :          break;
923 :      case 2:
924 :          intImageCopy( write_buff_addr, PIXEL_HW, PIXEL_VW, VIDEO_FORMAT,
                                              field, &ImgInp0, ICS, INTERMITTENT_PRO4 );
925 :          break;
926 :      case 3:
927 :          intImageCopy( write_buff_addr, PIXEL_HW, PIXEL_VW, VIDEO_FORMAT,
                                              field, &ImgInp0, ICS, INTERMITTENT_PRO4 );
928 :          break;
929 :      case 4:
930 :          // Size( 80 * 60 ) -> ( 40 * 30 )
931 :          intImageReduction( &ImgInp0, &ImgInp1, VIDEO_FORMAT, Rate,
                                                                  INTERMITTENT_PRO2 );
932 :          break;
933 :      case 5:
934 :          intImageReduction( &ImgInp0, &ImgInp1, VIDEO_FORMAT, Rate,
                                                                  INTERMITTENT_PRO2 );
935 :          break;
936 :      case 6:
937 :          // RGB -> YCBCR422
938 :          intRGB_YCBCR_Converter( &ImgInp1, INTERMITTENT_PRO1 );
939 :          break;
940 :      case 7:
941 :          // Automatic calculation of threshold value ( YCBCR422 Only function )
942 :          ThresholdBuff = DiscriminantAnalysisMethod( &ImgInp1 );
943 :          // Threshold Max
944 :          ThresholdMax  = ThresholdBuff + 255;
945 :          if( ThresholdMax > 255 ) ThresholdMax = 255;
946 :          // Threshold Min
947 :          ThresholdMin  = ThresholdBuff + 0;
948 :          if( ThresholdMin  <=  50 ) ThresholdMin =  50;
949 :          break;
950 :      case 8:
951 :          // Binary process
952 :          ImageBinarySetR( ThresholdMin, ThresholdMax );
953 :          ImageBinarySetG( ThresholdMin, ThresholdMax );
954 :          ImageBinarySetB( ThresholdMin, ThresholdMax );
955 :          intImageBinary( &ImgInp1, &ImgInp1, BINARY_RGB, INTERMITTENT_PRO1 );
956 :          SensorPro( &ImgInp1, &sensor1 );
957 :          SensorPro( &ImgInp1, &sensor0 );
958 :          break;
959 :      #endif
960 :
```

```
961 :       default:
962 :           PNumber = 16;
963 :           break;
964 :       }
965 :   }
```

6.2.4 sensor_inp 関数(センサ状態の読み込み)

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

```
1253 :  //---------------------------------------------------------------------//
1254 :  // Sensor Input Function
1255 :  // S          : SENSOR
1256 :  // Return     : Sensor Information( 8bit )
1257 :  //---------------------------------------------------------------------//
1258 :  unsigned char sensor_inp( SENSOR *S, unsigned char mask )
1259 :  {
1260 :      return ( S->v & 0xff ) & mask;
1261 :  }
```

1258 行	sensor 変数を unsigned char 型で宣言します。この変数は、sensor_inp 関数内でセンサの状態を加工する変数です。センサ情報は、8bit 幅である char 型とし、符号無しで宣言しています。
1260 行	センサ情報を読み込みます。 センサ情報 sensor 変数と sensor_inp 関数の引数であるマスク値を AND 演算し、sensor 変数に代入します。不要なビットを強制的に"0"にしています。 白＝"1"、黒＝"0"を戻します。

①sensor_inp 関数の構造

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

```
1253 :  //---------------------------------------------------------------------//
1254 :  // Sensor Input Function
1255 :  // S          : SENSOR
1256 :  // Return     : Sensor Information( 8bit )
1257 :  //---------------------------------------------------------------------//
1258 :  unsigned char sensor_inp( SENSOR *S, unsigned char mask )
1259 :  {
1260 :      return ( S->v & 0xff ) & mask;
1261 :  }
```

センサ値を加工したいちばん最後でマスク処理する

②マスク値の定義

kit20_gr-peach.cpp では、よく使うマスク値を define 定義しています。

```
#define MASKA_B          マスク値
```

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

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

今回定義した内容を、一覧表で示します。

定義したマスク文字列	マスク値	2 進数	意味
MASK2_2	0x66	0110 0110	左の真ん中 2 個、右の真ん中 2 個を有効に、他をマスクする。
MASK2_0	0x60	0110 0000	左の真ん中 2 個を有効に、他をマスクする。
MASK0_2	0x06	0000 0110	右の真ん中 2 個を有効に、他をマスクする。
MASK3_3	0xe7	1110 0111	左 3 個、右 3 個を有効に、他をマスクする。
MASK0_3	0x07	0000 0111	右 3 個を有効に、他をマスクする。
MASK3_0	0xe0	1110 0000	左 3 個を有効に、他をマスクする。
MASK4_0	0xf0	1111 0000	左 4 個を有効に、他をマスクする。
MASK0_4	0x0f	0000 1111	右 4 個を有効に、他をマスクする。
MASK4_4	0xff	1111 1111	右 4 個を有効に、左 4 個を有効に、他をマスクする。 ※結果、MASK4_4 はマスクせずにすべてのセンサを有効にしています。これは、**sensor_inp 関数のカッコの中には必ず値を入れなければいけないので、マスクする必要がなくても「0xff」として全ビット有効にする値を入れています**。

sensor_inp 関数のカッコの中に設定するマスク値は、上記のマスク文字列を設定します。
マスクしたいマスク文字列が無ければ、自分で定義して増やしてください。または、マスク文字列を使わずに、直接数値をいれても構いません。

(1) sensor_inp関数の使い方

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

マスク値には、センサの値をマスクする値、センサチェック値はマスク後にチェックしたい値を入れます。
例えば、センサ値は 0x1f、マスク値は MASK0_2、センサチェック値は 0x04 のときの動作を下記に示します。

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

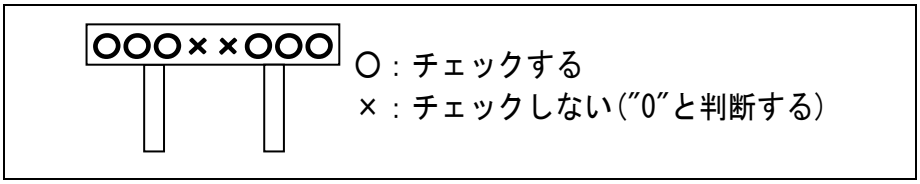
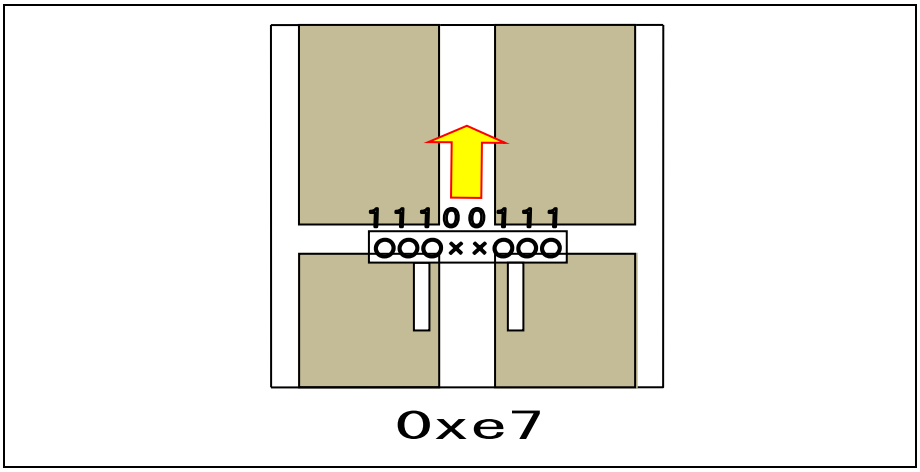
①	センサの値は「0001 1111」、sensor_inp 関数のマスク値は「0000 0110」です。AND 演算を行うと結果は下記のようになります。 センサの値 0001 1111 マスク値 0000 0110 (AND 結果 0000 0110 → 16 進数で 0x06
②	結果の 0x06 とセンサチェック値の 0x04 をチェックします。一致しないので、「式が成り立たないなら、この中を実行」部分を実行します。

6.2.5 check_crossline関数（クロスラインチェック）

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

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

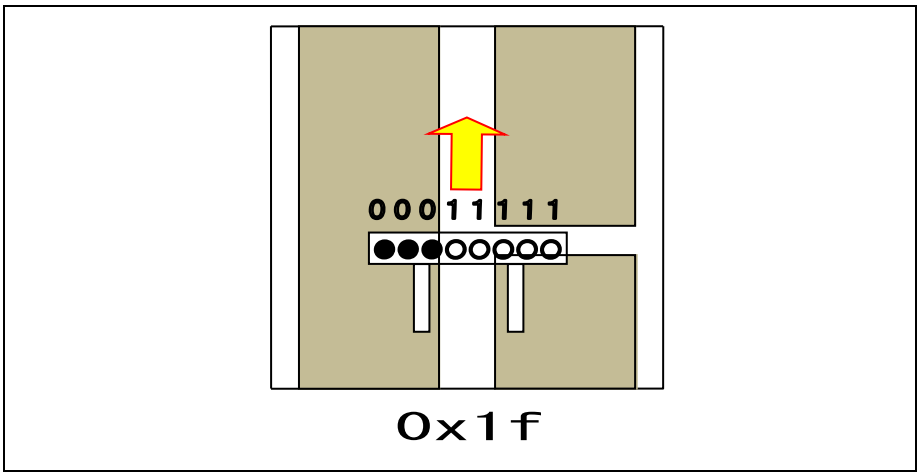
```
1282 :  //----------------------------------------------------------------//
1283 :  // Check cross line Function
1284 :  // Return values: 0: no cross line, 1: cross line
1285 :  //----------------------------------------------------------------//
1286 :  int check_crossline( void )
1287 :  {
1288 :      unsigned char b;
1289 :      int ret;
1290 :
1291 :      ret = 0;
1292 :      b = sensor_inp( &sensor0, MASK3_3);
1293 :      if( b==0xe7 ) {
1294 :          ret = 1;
1295 :      }
1296 :      return ret;
1297 :  }
```

1289 行	戻り値を保存する ret 変数を初期化しています。ret 変数には、クロスラインなら 1、違うなら 0 を代入します。今のところどちらか分からないので、とりあえず違うと判断して 0 を入れておきます。
1292 行	センサを読み込み、変数 b へ保存します。センサのマスク値は、「MASK3_3(0xe7)」なので、左 3 個、右 3 個の合計 6 個のセンサを読み込みます。 読み込むセンサを下図に示します。 
1293 行	センサの状態が 0xe7 かどうかチェックします。0xe7 は、クロスラインを検出したと判断します。下図にその様子を示します。  センサが、0xe7 なら if の条件が成り立つので ret 変数は 1、違うなら ret 変数は変化せず 0 のままとなります。 ret 変数が戻り値なので、"1"＝クロスラインあり、"0"＝クロスラインなし、ということになります。

6.2.6 check_rightling関数（右ハーフライン検出処理）

右レーンチェンジの 300mm～1000mm 手前に右ハーフラインがあります。check_rightline 関数は、右ハーフラインの検出を行う関数です。
戻り値は、右ハーフラインと判断すると 1、右ハーフラインでなければ 0 とします。

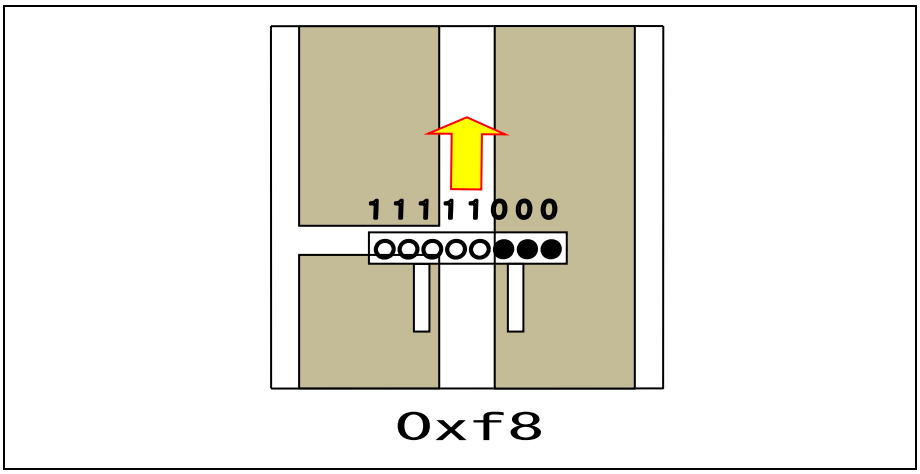
```
1299 :  //-----------------------------------------------------------------//
1300 :  // Check right line Function
1301 :  // Return values: 0: not detected, 1: detected
1302 :  //-----------------------------------------------------------------//
1303 :  int check_rightline( void )
1304 :  {
1305 :      unsigned char b;
1306 :      int ret;
1307 :
1308 :      ret = 0;
1309 :      b = sensor_inp( &sensor0, MASK4_4);
1310 :      if( b==0x1f ) {
1311 :          ret = 1;
1312 :      }
1313 :      return ret;
1314 :  }
```

1306 行	戻り値を保存する ret 変数を初期化しています。ret 変数には、右ハーフラインなら 1、違うなら 0 を代入します。今のところどちらか分からないので、とりあえず違うと判断して 0 を入れておきます。
1308 行 ～ 1313 行	センサの状態をチェックします。マスク値は、MASK4_4 なのでセンサの状態を全て読み込みます。このとき、0x1f なら右ハーフラインを検出したと判断します。下図にその様子を示します。  センサが、0x1f なら if の条件が成り立つので ret 変数は 1、違うなら ret 変数は変化せず 0 のままとなります。 ret 変数が戻り値なので、"1"＝右ハーフラインあり、"0"＝右ハーフラインなし、ということになります。

6.2.7 check_leftline関数（左ハーフライン検出処理）

左レーンチェンジの300mm～1000mm手前に左ハーフラインがあります。check_leftline関数は、左ハーフラインの検出を行う関数です。
戻り値は、左ハーフラインと判断すると1、左ハーフラインでなければ0とします。

```
1316 :  //---------------------------------------------------------------------//
1317 :  // Check left line Function
1318 :  // Return values: 0: not detected, 1: detected
1319 :  //---------------------------------------------------------------------//
1320 :  int check_leftline( void )
1321 :  {
1322 :      unsigned char b;
1323 :      int ret;
1324 :
1325 :      ret = 0;
1326 :      b = sensor_inp( &sensor0, MASK4_4);
1327 :      if( b==0xf8 ) {
1328 :          ret = 1;
1329 :      }
1330 :      return ret;
1331 :  }
```

1323行	戻り値を保存するret変数を初期化しています。ret変数には、左ハーフラインなら1、違うなら0を代入します。今のところどちらか分からないので、とりあえず違うと判断して0を入れておきます。
1325行 ～ 1330行	センサの状態をチェックします。マスク値は、MASK4_4なのでセンサの状態を全て読み込みます。このとき、0xf8なら左ハーフラインを検出したと判断します。下図にその様子を示します。  センサが、0xf8ならifの条件が成り立つのでret変数は1、違うならret変数は変化せず0のままとなります。 ret変数が戻り値なので、"1"＝左ハーフラインあり、"0"＝左ハーフラインなし、ということになります。

6.2.8 dipsw_get関数（ディップスイッチ値読み込み）

dipsw_get 関数は、GR-MCR 基板 Rev.1.0 のディップスイッチの状態を読み込む関数です。
戻り値は、ディップスイッチの値によって 0～15 の値が返ってきます。

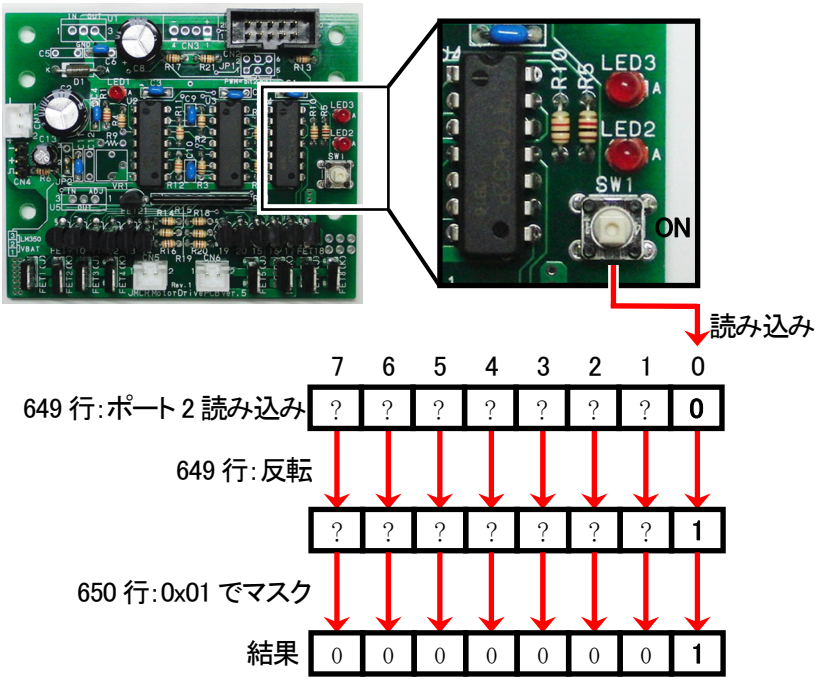
```
1106 :  //-----------------------------------------------------------------//
1107 :  // Dipsw get Function
1108 :  // Return     : SW value 0 ~ 15
1109 :  //-----------------------------------------------------------------//
1110 :  unsigned char dipsw_get( void )
1111 :  {
1112 :      return( dipsw.read() & 0x0f );
1113 :  }
```

6.2.9 pushsw_get関数(プッシュスイッチ値読み込み)

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

```
1049 :  //------------------------------------------------------------------//
1050 :  // Push SW Get Function
1051 :  // Return      : 0:OFF, 1:ON
1052 :  //------------------------------------------------------------------//
1053 :  unsigned char pushsw_get( void )
1054 :  {
1055 :      return (~push_sw) & 0x1;            /* Read ports with switches */
1056 :  }
```

プッシュスイッチは、ポート 2 の bit0 に接続されています。ポート 2 の bit7～1 は関係ないので、ビット演算を使ってプッシュスイッチの bit だけ取り込みます。
プッシュスイッチを押したときの動きを下図に示します。



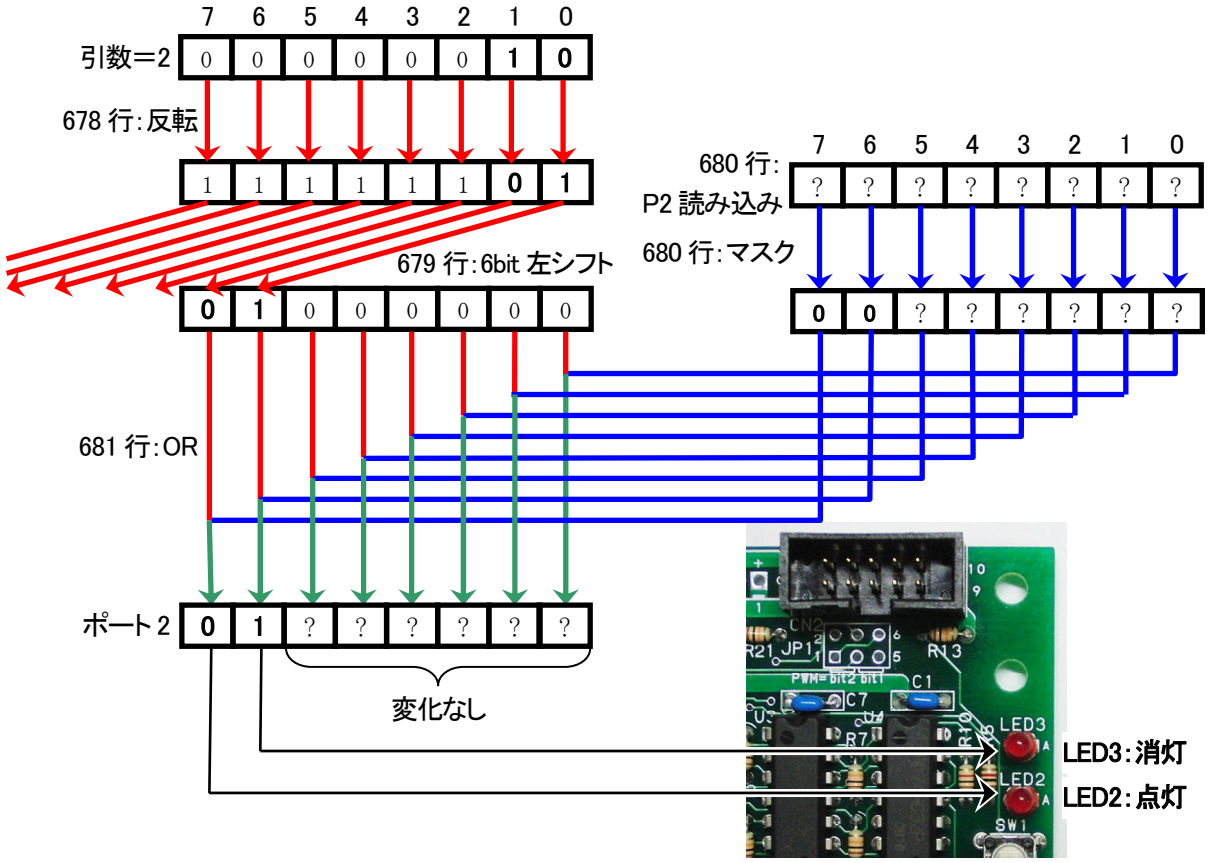
6.2.10 led_out関数(LED制御)

led_out 関数は、モータドライブ基板の LED2、LED3 を消灯／点灯させる関数です。
引数と LED2、LED3 の関係を下記に示します。

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

```
1037 :  //--------------------------------------------------------------------//
1038 :  // Led Out Function
1039 :  // led       : LED2:bit1 LED3:bit0  "0":OFF "1":ON
1040 :  // For example ) 0x3->LED2:ON LED3:ON  0x2->LED2:ON LED3:OFF
1041 :  //--------------------------------------------------------------------//
1042 :  void led_out(int led)
1043 :  {
1044 :      led = ~led;
1045 :      LED_3 = led & 0x1;
1046 :      LED_2 = ( led >> 1 ) & 0x1;
1047 :  }
```

引数が 2 のときの動きを下図に示します。



6.2.11 motor関数(モータ速度制御)

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

```
1058 :  //-----------------------------------------------------------------//
1059 :  // motor speed control(PWM)
1060 :  // Arguments: motor:-100 to 100
1061 :  // Here, 0 is stop, 100 is forward, -100 is reverse
1062 :  //-----------------------------------------------------------------//
1063 :  void motor( int accele_l, int accele_r )
1064 :  {
1065 :      int    sw_data;
1066 :
1067 :      sw_data = dipsw_get() + 5;
1068 :      accele_l = ( accele_l * sw_data ) / 20;
1069 :      accele_r = ( accele_r * sw_data ) / 20;
1070 :
1071 :      // Left Motor Control
1072 :      if( accele_l >= 0 ) {
1073 :          // forward
1074 :          Left_motor_signal = 0;
1075 :          MTU2TGRC_4 = (long)( MOTOR_PWM_CYCLE - 1 ) * accele_l / 100;
1076 :      } else {
1077 :          // reverse
1078 :          Left_motor_signal = 1;
1079 :          MTU2TGRC_4 = (long)( MOTOR_PWM_CYCLE - 1 ) * ( -accele_l ) / 100;
1080 :      }
1081 :
1082 :      // Right Motor Control
1083 :      if( accele_r >= 0 ) {
1084 :          // forward
1085 :          Right_motor_signal = 0;
1086 :          MTU2TGRD_4 = (long)( MOTOR_PWM_CYCLE - 1 ) * accele_r / 100;
1087 :      } else {
1088 :          // reverse
1089 :          Right_motor_signal = 1;
1090 :          MTU2TGRD_4 = (long)( MOTOR_PWM_CYCLE - 1 ) * ( -accele_r ) / 100;
1091 :      }
1092 :  }
```

(1) motor関数の使い方

motor 関数の使い方を下記に示します。

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

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

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

実際にモータに出力される割合を、下記に示します。

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

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

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

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

実際のモータに出力される PWM 値は、下記のようになります。

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

```
1067 :      sw_data = dipsw_get() + 5;        dipsw_get() = ディップスイッチの値0～15
1068 :      accele_l = ( accele_l * sw_data ) / 20;
1069 :      accele_r = ( accele_r * sw_data ) / 20;
```

(3) ディップスイッチの値とモータ出力

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

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

6.2.12 handle関数(サーボハンドル操作)

```
1094 :  //---------------------------------------------------------------------//
1095 :  // handle Function
1096 :  //---------------------------------------------------------------------//
1097 :  void handle( int angle )
1098 :  {
1099 :      // When the servo move from left to right in reverse, replace "-" with "+"
1100 :      MTU2TGRD_0 = SERVO_CENTER - angle * HANDLE_STEP;
1101 :  }
```

(1) handle関数の使い方

handle 関数の使い方を下記に示します。

```
handle( サーボの角度 );
```

引数は、サーボの角度を設定します。値とサーボの角度の関係を下記に示します。

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

プログラム例を、下記に示します。

```
handle( 0 );        0 度

handle( 30 );       右 30 度

handle( -45 );      左 45 度
```

6.2.13 スタート

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

```
233 :  //*****************************************************************//
234 :  // Main Function
235 :  //*****************************************************************/
236 :  int main( void )
237 :  {
238 :      volatile int    SL;                 /* Sensor Line Pixel        */
239 :
240 :      // Initialize MCU Functions
241 :      init_MTU2_PWM_Motor();
242 :      init_MTU2_PWM_Servo();
243 :      pc.baud(230400);
244 :
245 :      // Interrupt Function( intTimer )( 1ms )
246 :      interrput.attach(&intTimer, 0.001);
247 :      DebugMode = 0;
248 :
249 :      // Initialize Micon Car state
250 :      handle( 0 );
251 :      motor( 0, 0 );
252 :      led_out( 0x0 );
253 :      led_m_set( STOP );
254 :
255 :      // Initialize threshold
256 :      ThresholdBuff = THRESHOLD;
257 :
258 :      // Camera Start
259 :      EasyAttach_Init(Display, PIXEL_HW, PIXEL_VW);
260 :      Start_Video_Camera();
261 :
262 :      // wait to stabilize NTSC signal (about 200ms)
263 :      wait(0.2);
264 :
265 :      // Screen clear
266 :      pc.printf( "¥033[2J" );
267 :      pc.printf( "¥033[50F" );
268 :
269 :      // Initialize Sensor
270 :      // start bar sensor
271 :      SL = ( ( PIXEL_VW >> ICS ) * Rate ) * (double)0.7;
272 :      senError  = initSensor( &ImgInp1, &sensor1, SL );
273 :      // line trace sensor
274 :      SL = ( ( PIXEL_VW >> ICS ) * Rate ) * (double)0.8;
275 :      senError += initSensor( &ImgInp1, &sensor0, SL );
276 :      if( senError !=0 ) {
277 :          pc.printf( "initSensor Function Error ¥n¥r" );
278 :          pc.printf( "¥n¥r" );
279 :          led_m_set( ERROR );
280 :          cnt1 = 0;
281 :          while( cnt1 < 3000 ){
282 :              if( cnt1 % 200 < 100 ) {
283 :                  led_out( 0x3 );
```

```
284 :               } else {
285 :                   led_out( 0x0 );
286 :               }
287 :           }
288 :           led_m_set( STOP );
289 :       }
290 :
291 :       // Debug Program
292 :       if( user_button_get() ) {
293 :           led_m_set( DEBUG );
294 :           wait(0.1);  // ON  Time
295 :           while( user_button_get() );
296 :           wait(0.5);  // OFF Time
297 :           DebugPro();
298 :       }
```

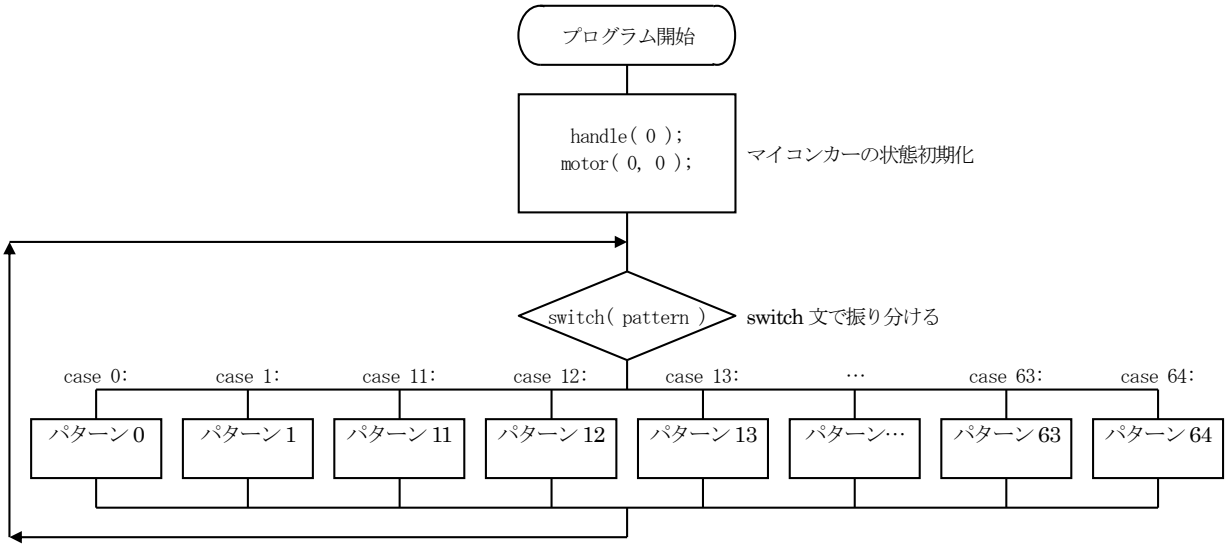
240 行 ～ 243 行	RZ/A1H マイコンの内蔵周辺機能を初期化する関数です。
246 行	割り込み関数を設定する関数です。
249 行 ～ 253 行	マイコンカーの状態を初期化します。 handle 関数でサーボの角度を 0 度にします。 motor 関数で左モータの回転を 0%、右モータの回転を 0%にします。
256 行	二値化する際のしきい値の設定します。
258 行 ～ 263 行	カメラの設定を初期化する関数です。
269 行 ～ 289 行	センサ情報を取得する関数を初期化する関数です。センサ関数の初期化に失敗した場合は、モータドライブ基板の LED が 3 秒間点滅します。LED が点滅した場合は、マイコンボードのリセットを押してください。
291 行 ～ 297 行	デバックプログラムを実行するモードです。

6.2.14 パターン方式について

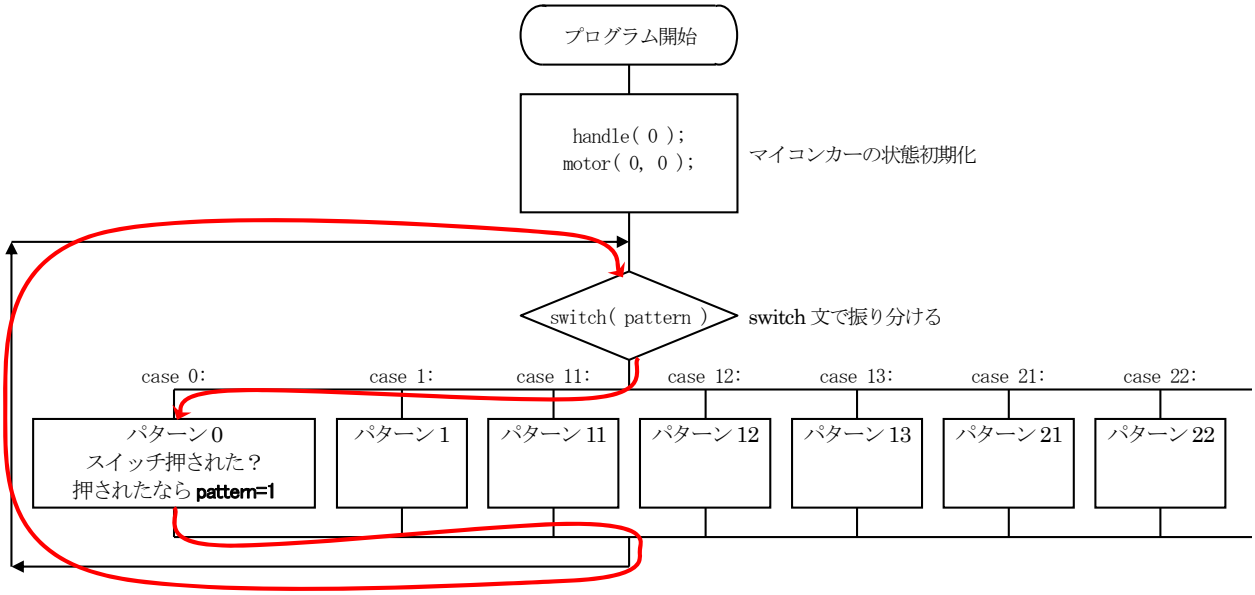
Kit20_gr-peach.cpp では、パターン方式という方法でプログラムを実行します。
仕組みは、あらかじめプログラムを細かく分けておきます。例えば、「スイッチ入力待ちの処理を行うプログラム」、「スタートバーが開いたかチェックする処理を行うプログラム」、などです。
次に、pattern という変数を作ります。この変数に設定した値により、どのプログラムを実行するか選択します。例えば、パターン変数が 0 のときはスイッチ入力待ちの処理、パターン変数が 1 のときはスタートバーが開いたかチェックする処理. . . などです。
この方式を使うと、パターンごとに処理を分けられるため、プログラムが見やすくなります。パターン方式は、「**プログラムのブロック化**」と言うこともできます。

6.2.15 プログラムの作り方

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

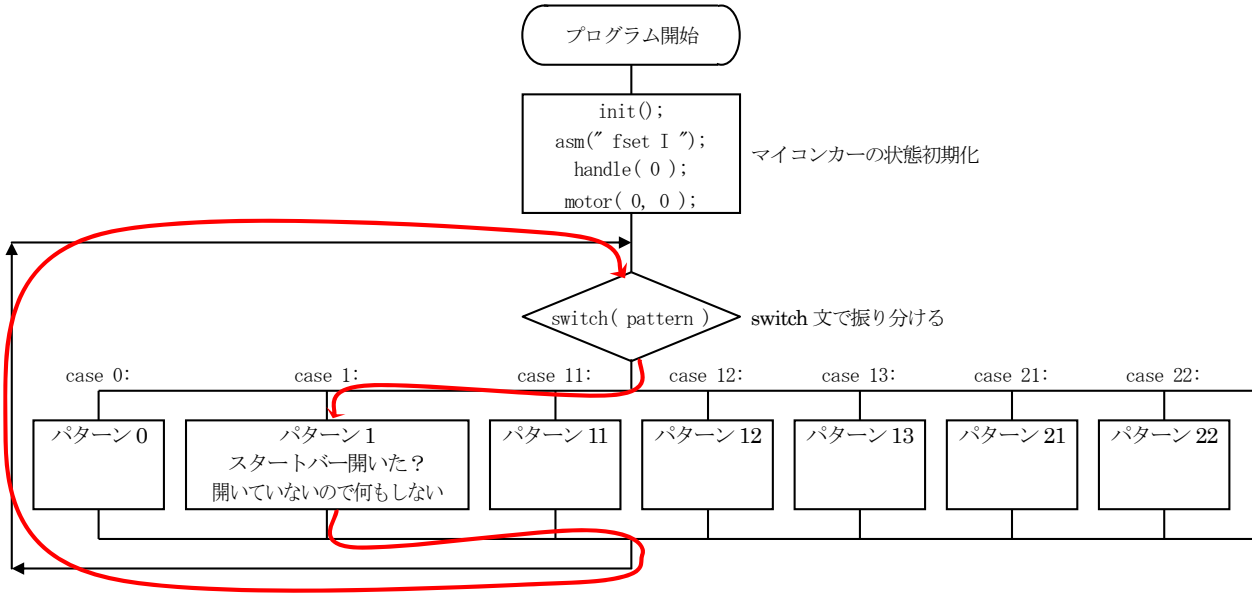


起動時、pattern 変数は 0 です。switch 文により case 0 部分のプログラム「パターン 0 プログラム」を実行し続けます。後ほど説明しますが、パターン 0 はスイッチ入力待ちです。スイッチが押されると、「pattern=1」が実行されます。フローチャートを下図に示します。



次に switch 文を実行したとき、pattern 変数の値が 1 になっているので、case 1 部分のプログラムが実行されます。本プログラムでは、switch(pattern)の case 1 のプログラムを、「パターン 1 のプログラム」と言うことにします。

パターン 1 は、スタートバーが開いたかどうかチェックする部分です。フローチャートを下図に示します。



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

プログラムを下記に示します。通常の switch～case 文です。

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

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

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

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

6.2.16 パターンの内容

kit20_gr-peach.c のパターン番号と、プログラムの処理内容、パターンが変わる条件を下記に示します。

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

6.2.17 パターン方式の最初while、switch部分

```
301 :      while( 1 ) {
312 :      switch( pattern ) {
336 :      case 0:
               パターン0時の処理
362 :          break;
363 :
364 :      case 1:
               パターン1時の処理
380 :          break;
                                                          対      対
           それぞれのパターン処理

716 :      default:
717 :          break;
718 :      }
719 :      }
```

301 行の「while(1){」と 719 行の「}」が対、312 行の「switch(pattern){」と 718 行の「}」が対となります。

通常、中カッコ「{」があるとそれと対になる中カッコ閉じ「}」が来るまで、くくられた中は 4 文字右にずらして分かりやすくします。このプログラムも 4 文字右にずらしています。しかし、while 部分と switch 部分は同列に書いています。これは、プログラムが複雑になった場合に画面の右端を超えて 2 行になり、見づらくなるのを防ぐためです。元々、人間に分かりやすくするために列をずらしているわけですから、コンパイル上はまったく問題ありません。どうしても気になってしまう場合は、301～719 行を 4 文字分、右にずらすと良いでしょう。

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

```
while( 式が真なら ) {
    …
    …
    …
}
```

```
while( 式が偽なら ) {
    …
    …
    …
}
…
```

「真」、「偽」とは、

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

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

6.2.18 パターン 0:スイッチ入力待ち

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

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

```
336 :      case 0:
337 :          /* wait for switch input */
338 :          if( pushsw_get() ) {
339 :              led_out( 0x0 );
340 :              pattern = 1;
341 :              while( pushsw_get() );
342 :              cnt1 = 0;
343 :              break;
344 :          }
345 :          if( !startbar_get() ) {
346 :              if( cnt1 < 100 ) { /* LED flashing processing */
347 :                  led_out( 0x1 );
348 :              } else if( cnt1 < 200 ) {
349 :                  led_out( 0x2 );
350 :              } else {
351 :                  cnt1 = 0;
352 :              }
353 :          } else {
354 :              if( cnt1 < 50 ) { /* LED flashing processing */
355 :                  led_out( 0x1 );
356 :              } else if( cnt1 < 100 ) {
357 :                  led_out( 0x2 );
358 :              } else {
359 :                  cnt1 = 0;
360 :              }
361 :          }
362 :          break;
```

6.2.19 パターン 1:スタートバーが開いたかチェック

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

```
364 :      case 1:
365 :          /* Check if start bar is open */
366 :          if( !startbar_get() ) {
367 :              /* Start!! */
368 :              led_out( 0x0 );
369 :              pattern = 11;
370 :              cnt1 = 0;
371 :              break;
372 :          }
373 :          if( cnt1 < 50 ) { /* LED flashing processing */
374 :              led_out( 0x1 );
375 :          } else if( cnt1 < 100 ) {
376 :              led_out( 0x2 );
377 :          } else {
378 :              cnt1 = 0;
379 :          }
380 :          break;
```

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

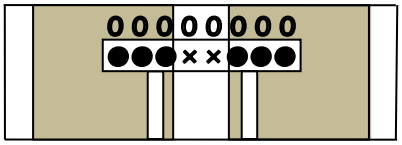
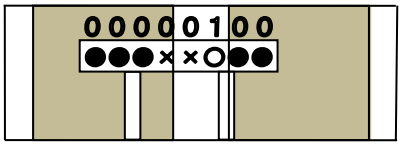
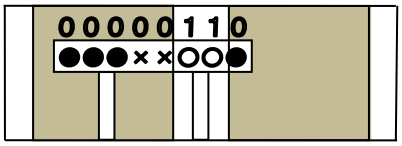
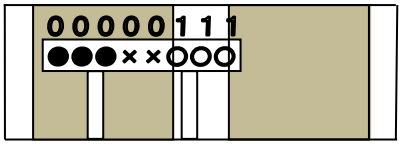
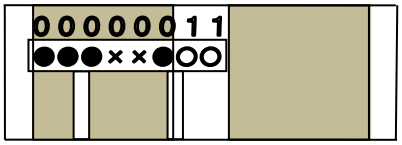
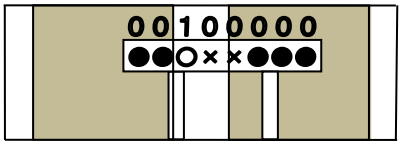
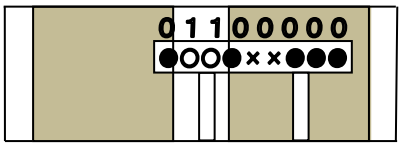
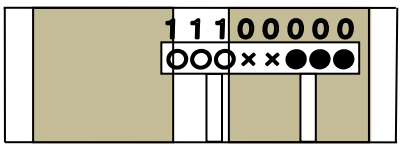
6.2.20 パターン 11:通常トレース

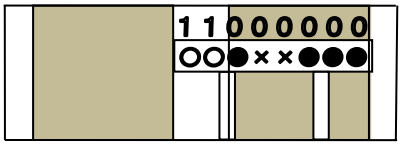
パターン 11 は、センサを読み込んで、左モータ、右モータ、サーボを制御する状態です。

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

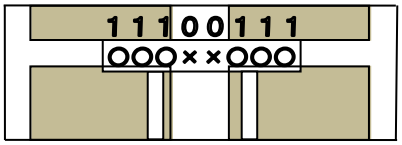
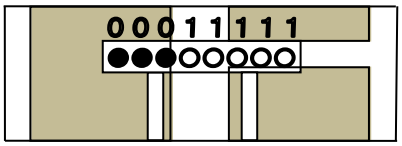
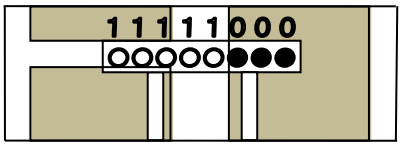
次に、そのときの左モータの PWM 値、右モータの PWM 値、ハンドル切れ角を考えます。考え方は、センサが中心のときはハンドルをまっすぐにしてスピードを上げます。センサのずれが大きくなればなるほど、ハンドルの曲げ角を大きくして、左モータ、右モータのスピードを落とします。

kit12_38a.c では、下記のようにしました。

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

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

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

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

この表に基づいて、プログラムを書いていきます。

(1) センサ読み込み

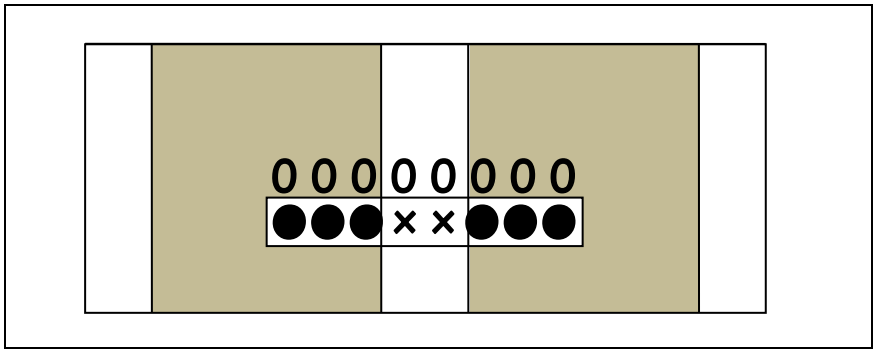
```
396 :          switch( sensor_inp( &sensor0, MASK3_3 ) ) {
```

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

(2) 直進

```
397 :              case 0x00:
398 :                  /* Center -> straight */
399 :                  handle( 0 );
400 :                  motor( 100, 100 );
401 :                  break;
```

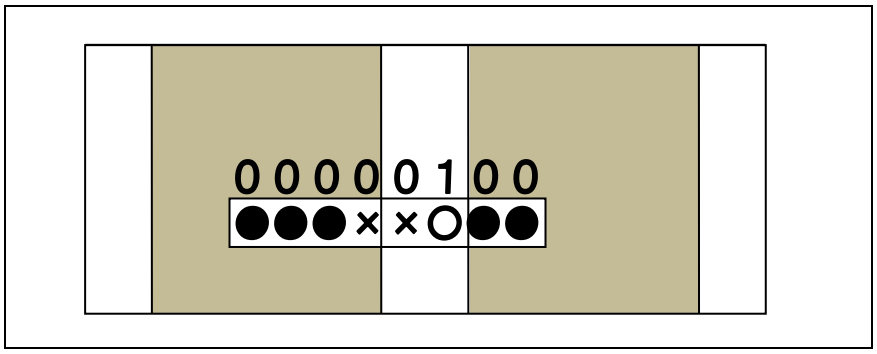
センサが「0x00」の状態です。この状態は下図のように、まっすぐ進んでいる状態です。サーボ角度 0 度、左モータ 100%、右モータ 100%で進みます。



(3) 左寄り

```
403 :              case 0x04:
404 :                  /* Slight amount left of center -> slight turn to right */
405 :                  handle( 5 );
406 :                  motor( 100, 100 );
407 :                  break;
```

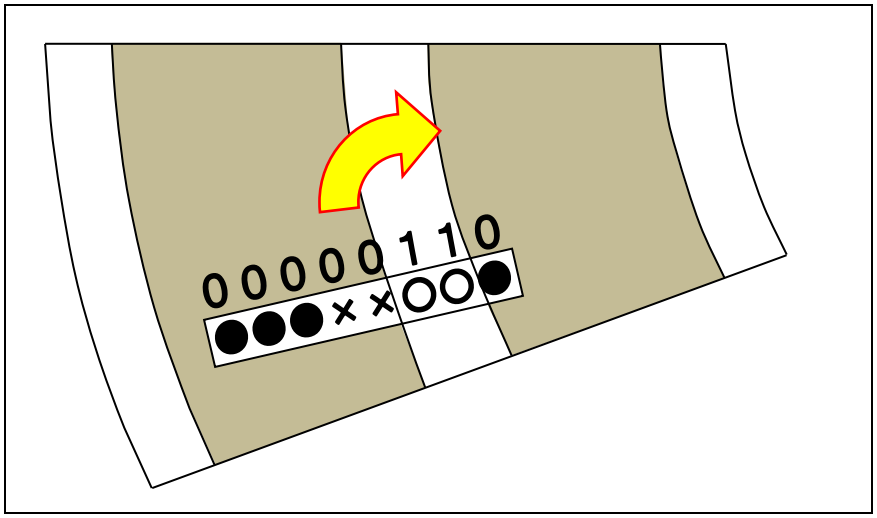
センサが「0x04」の状態です。この状態は下図のように、マイコンカーが微妙に左に寄っている状態です。サーボを右に 5 度、左モータ 100%、右モータ 100%で進み、中心に寄るようにします。



(4) 少し左寄り

```
409 :               case 0x06:
410 :                   /* Small amount left of center -> small turn to right */
411 :                   handle( 10 );
412 :                   motor( 80, 67 );
413 :                   break;
```

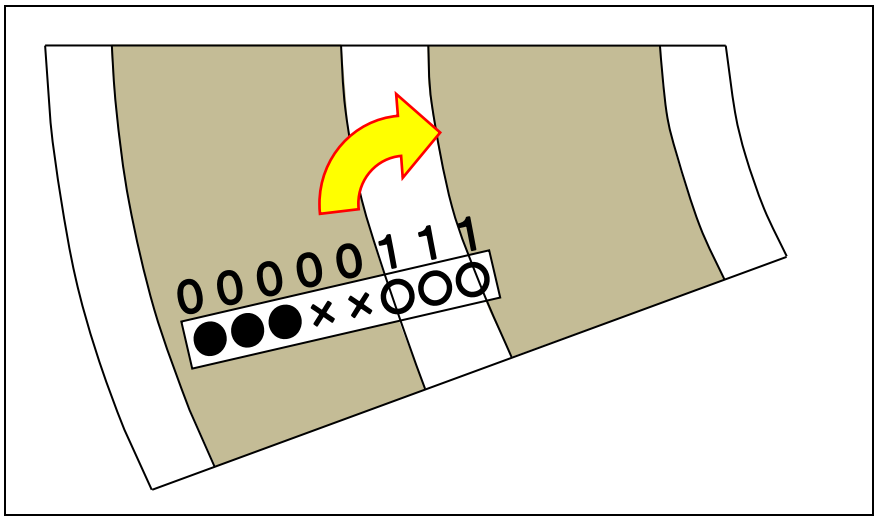
センサが「0x06」の状態です。この状態は下図のように、マイコンカーが少し左に寄っている状態です。サーボを右に 10 度、左モータ 80%、右モータ 67%で進み、減速しながら中心に寄るようにします。



(5) 中くらい左寄り

```
415 :               case 0x07:
416 :                   /* Medium amount left of center -> medium turn to right */
417 :                   handle( 15 );
418 :                   motor( 50, 38 );
419 :                   break;
```

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

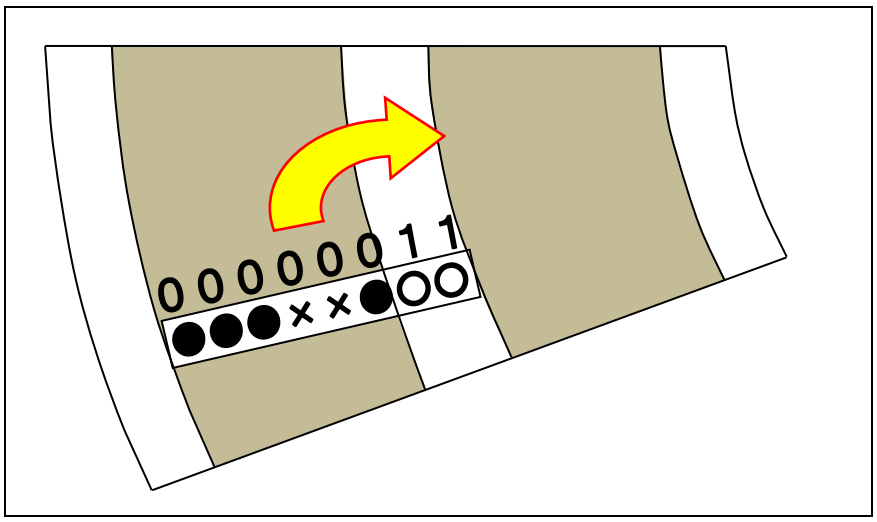


(6) 大きく左寄り

```
421 :               case 0x03:
422 :                   /* Large amount left of center -> large turn to right */
423 :                   handle( 25 );
424 :                   motor( 30, 19 );
426 :                   break;
```

※本当は 425 行がありますが、ここでは省略して説明します。後述します。

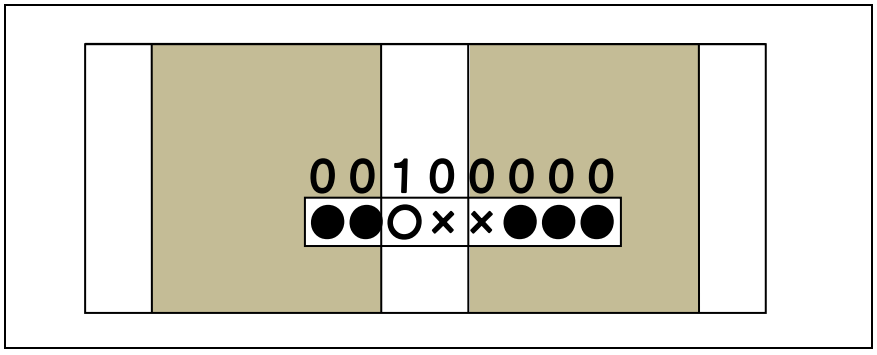
センサが「0x03」の状態です。この状態は下図のように、マイコンカーが大きく左に寄っている状態です。サーボを右に 25 度、左モータ 30%、右モータ 19%で進み、かなり減速しながら中心に寄るようにします。



(7) 微妙に右寄り

```
428 :               case 0x20:
429 :                   /* Slight amount right of center -> slight turn to left */
430 :                   handle( -5 );
431 :                   motor( 100, 100 );
432 :                   break;
```

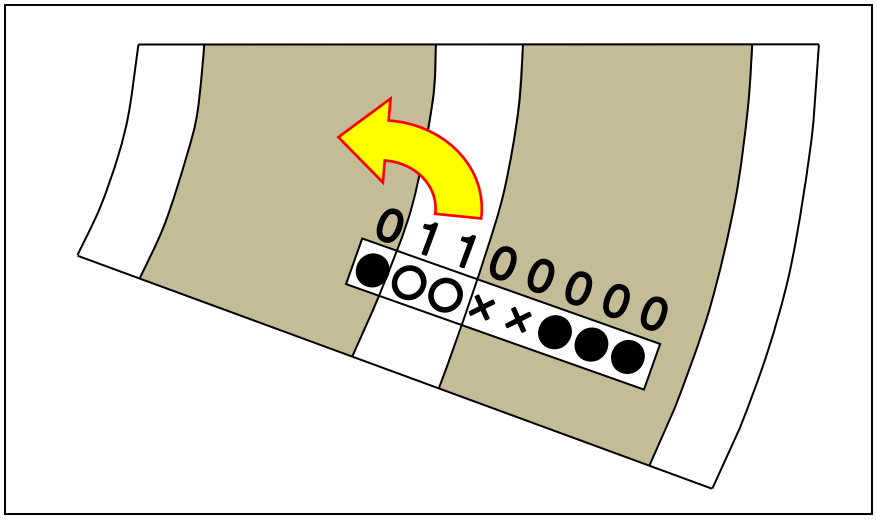
センサが「0x20」の状態です。この状態は下図のように、マイコンカーが微妙に右に寄っている状態です。サーボを左に 5 度、左モータ 100%、右モータ 100%で進み、中心に寄るようにします。



(8) 少し右寄り

```
434 :               case 0x60:
435 :                   /* Small amount right of center -> small turn to left */
436 :                   handle( -10 );
437 :                   motor( 67, 80 );
438 :                   break;
```

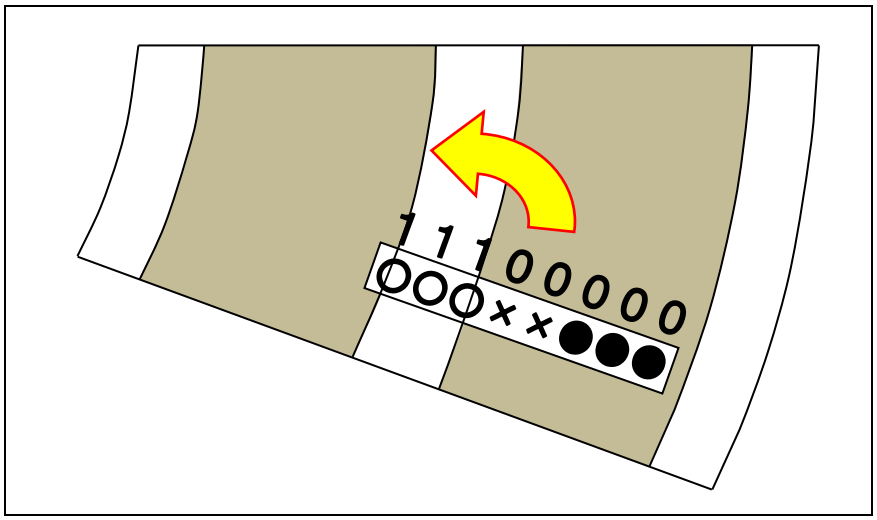
センサが「0x60」の状態です。この状態は下図のように、マイコンカーが少し右に寄っている状態です。サーボを左に 10 度、左モータ 67%、右モータ 80%で進み、減速しながら中心に寄るようにします。



(9) 中くらい右寄り

```
440 :               case 0xe0:
441 :                   /* Medium amount right of center -> medium turn to left */
442 :                   handle( -15 );
443 :                   motor( 38, 50 );
444 :                   break;
```

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

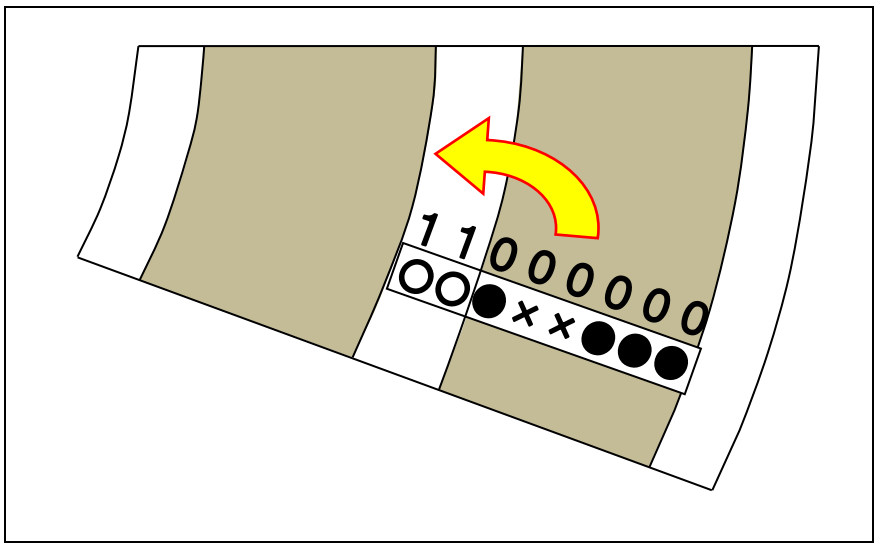


(10) 大きく右寄り

```
446 :               case 0xc0:
447 :                   /* Large amount right of center -> large turn to left */
448 :                   handle( -25 );
449 :                   motor( 19, 30 );
451 :                   break;
```

※本当は 450 行がありますが、ここでは省略して説明します。後述します。

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



(11) クロスラインチェック

```
384 :          if( check_crossline() ) { /* Cross line check */
385 :              pattern = 21;
386 :              break;
387 :          }
```

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

(12) 右ハーフライン

```
388 :          if( check_rightline() ) { /* Right half line detection check */
389 :              pattern = 51;
390 :              break;
391 :          }
```

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

(13) 左ハーフライン

```
392 :          if( check_leftline() ) { /* Left half line detection check */
393 :              pattern = 61;
394 :              break;
395 :          }
```

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

(14) それ以外

```
453 :              default:
454 :                  break;
```

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

(15) break文で抜ける位置

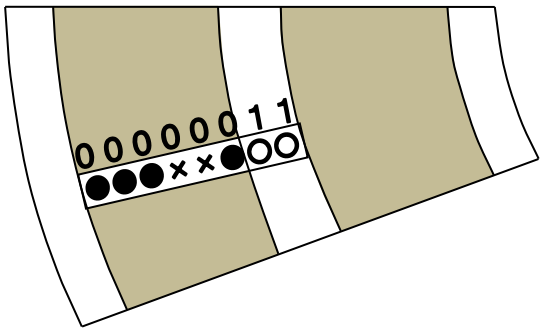
break 文は、switch 文または for、while、do～while のループから脱出させるための文です。**多重ループの中で用いられると、break 文が存在するループをひとつだけ打ち切り、すぐ外側のループに処理を移します。**「**ひとつだけ打ち切り**」が重要です。

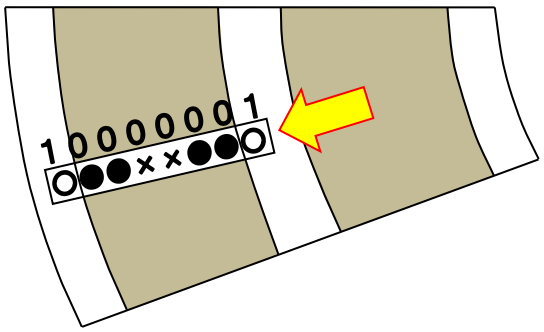
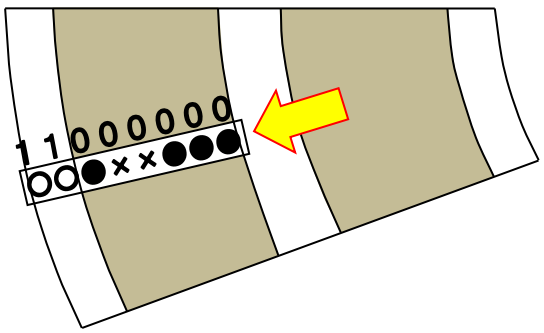
パターン 11 の break で抜ける位置は下記のようになります。抜ける位置が違いますので、どのループの中で break 文が使われているか見極めて判断してください。

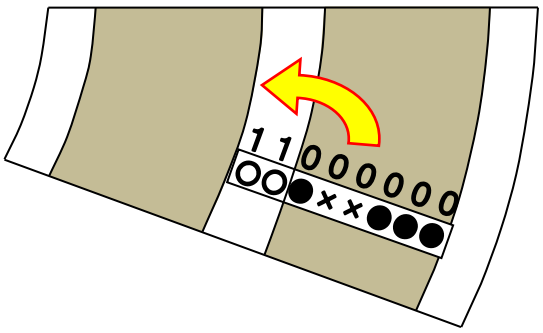
```
while( 1 ) {
switch( pattern ) {

    中略

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

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

        中略

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

    中略

} 1
}
```

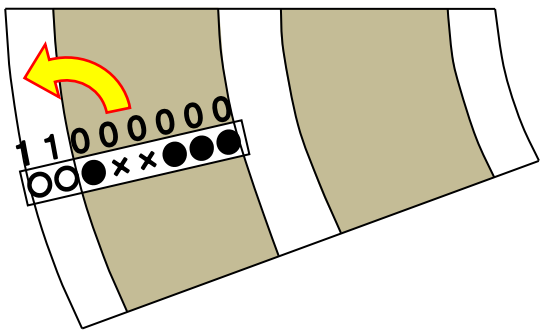
6.2.21 パターン 12:右へ大曲げの終わりのチェック

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

1		大きく左に寄ったときのセンサの状態です。
		センサは、「0000 0011」です。
		この状態になったなら、下記プログラムが実行されます。
		handle(25);
		motor(30 ,19);
		よって、右に 25 度ハンドルを切って左モータ 30%、右モータ 19%で回します。

2		さらに左に寄りました。 センサは、「1000 0001」です。 この状態のとき、実行するプログラムの記述はありません。この場合、前の状態を保持します。 前の状態は「0000 0011」なので、このセンサ値のモータ回転数、ハンドル角度になります。
3		さらに左に寄りました。 センサは、「1100 0000」です。

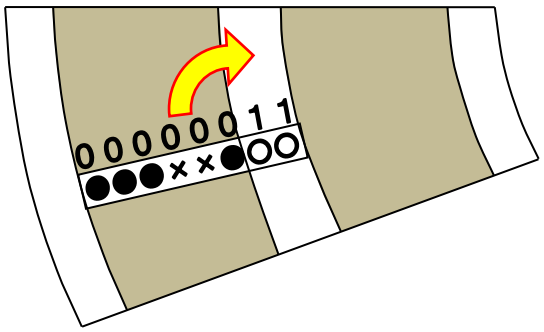
4		センサ状態「1100 0000」は、本プログラムでは、左図のように、マイコンカーが右に寄っているので、左にハンドルを曲げる状態を想定しています。この状態になったなら、下記プログラムが実行されます。
		handle(-25);
		motor(19, 30);
		よって、左に 25 度ハンドルを切って左モータ 19%、右モータ 30%で回します。

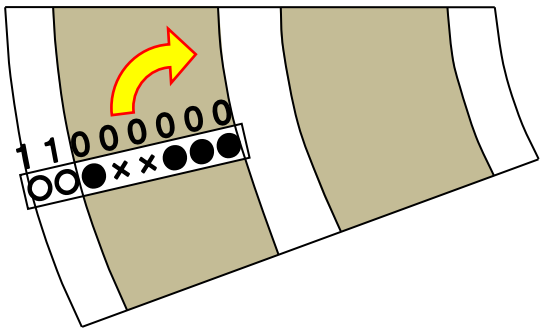
5		実際は、左に大きく寄っているので、この状態で左にハンドルを曲げると、脱輪してしまいます。

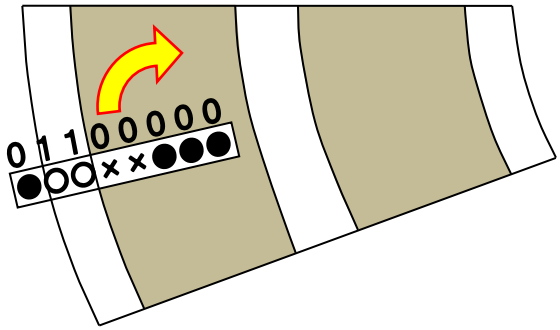
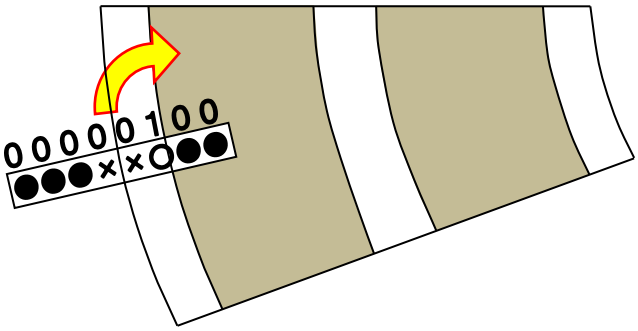
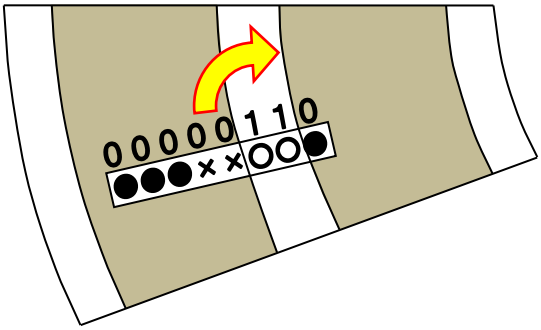
そこで、右に大曲げしたら、あるセンサ状態に戻るまで右に大曲げし続けるようにます。この“**あるセンサ状態**”を判定する部分を、パターン 12 に作ります。

パターン 11 の case 0x03 部分

```
421 :              case 0x03:
422 :                  /* Large amount left of center -> large turn to right */
423 :                  handle( 25 );
424 :                  motor( 30, 19 );
425 :                  pattern = 12;  ←追加　パターン１２へ移る
426 :                  break;
```

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

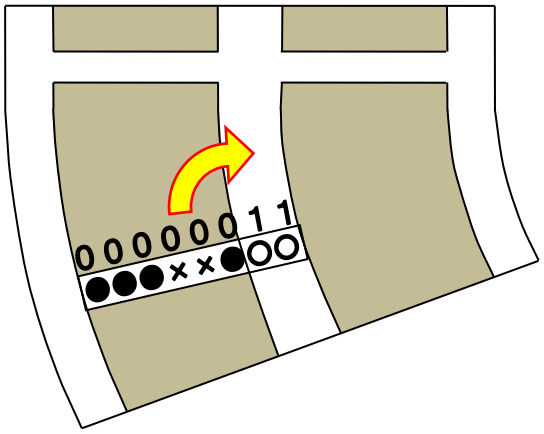
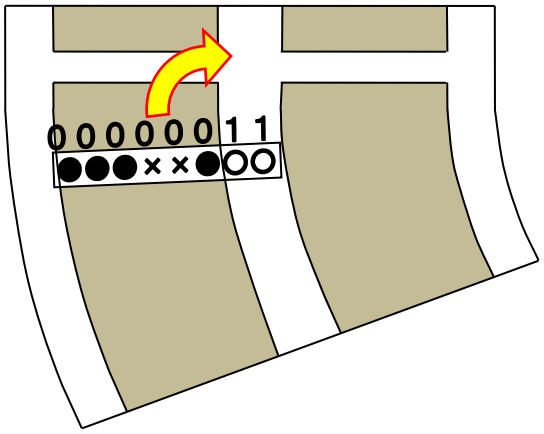
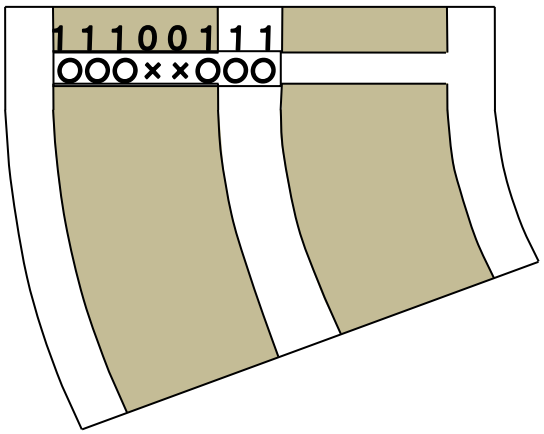
7		センサが「1100 0000」になりました。「0000 0110」ではないので、まだ右に曲げ続けます。 先ほどは右に寄っていると勘違いしましたが、今回はパターン 12 なので勘違いしません。

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

この考え方でプログラムします。

```
        case 12:
            /* Check end of large turn to right */
            if( sensor_inp( &sensor0, MASK3_3 ) == 0x06 ) {
                pattern = 11;
            }
            break;
```

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

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

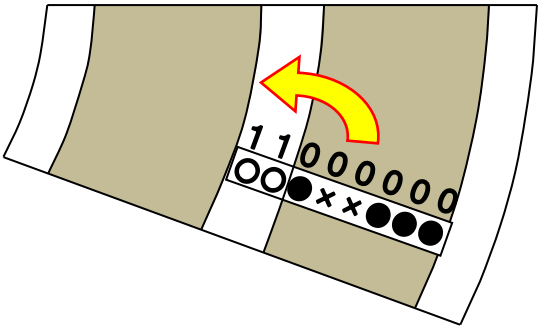
最終的なプログラムを下記に示します。

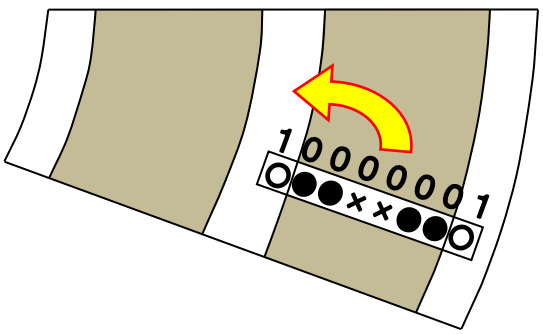
```
458 :       case 12:
459 :           /* Check end of large turn to right */
460 :           if( sensor_inp( &sensor0, MASK3_3 ) == 0x06 ) {
461 :               pattern = 11;
462 :           }
463 :           if( check_crossline() ) { /* Cross line check */
464 :               pattern = 21;
465 :               break;
466 :           }
467 :           if( check_rightline() ) { /* Right half line detection check */
468 :               pattern = 51;
469 :               break;
470 :           }
471 :           if( check_leftline() ) { /* Left half line detection check */
472 :               pattern = 61;
473 :               break;
474 :           }
475 :           break;
```

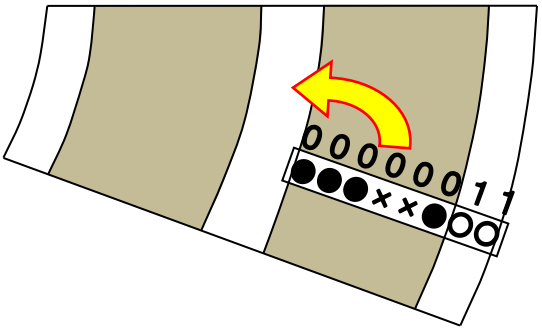
パターン 12 のプログラムはこれで完成です。

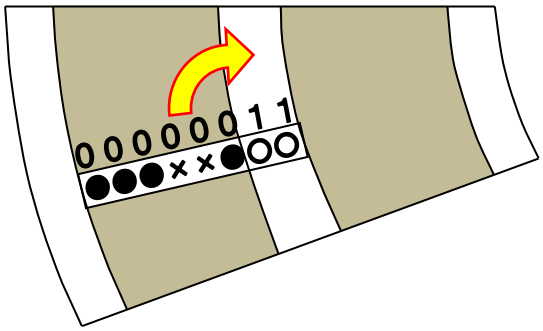
6.2.22 パターン 13:左へ大曲げの終わりのチェック

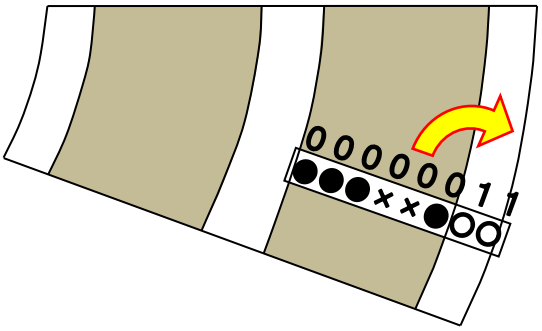
センサ状態 0xc0 は、一番大きく右に寄ったときのセンサ状態です。そのため、これ以上カーブで脹らんだ場合、下図のようになる可能性があります。

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

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

3		さらに右に寄りました。 センサは、「0000 0011」です。

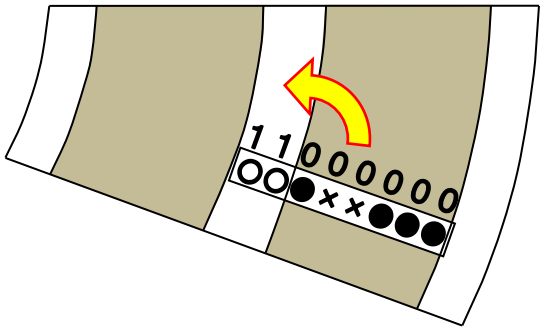
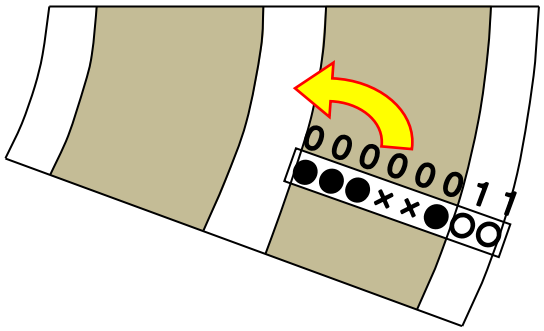
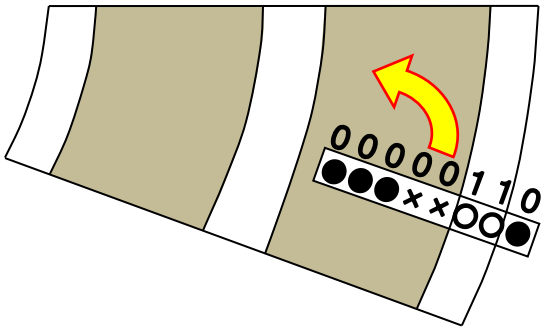
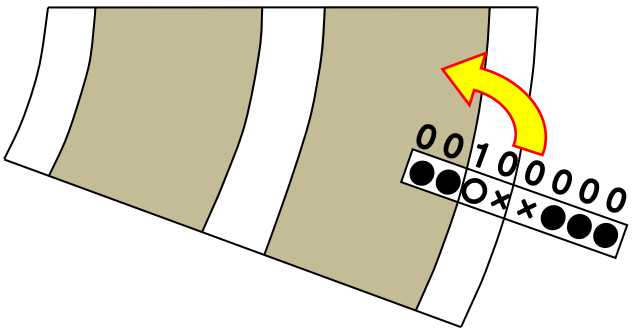
4		センサ状態「0000 0011」は、本プログラムでは、左図のように、マイコンカーが左に寄っているので、右にハンドルを曲げる状態を想定しています。この状態になったなら、下記プログラムが実行されます。
		handle(25);
		motor(30, 19);
		よって、右に 25 度ハンドルを切って左モータ 30%、右モータ 19%で回します。

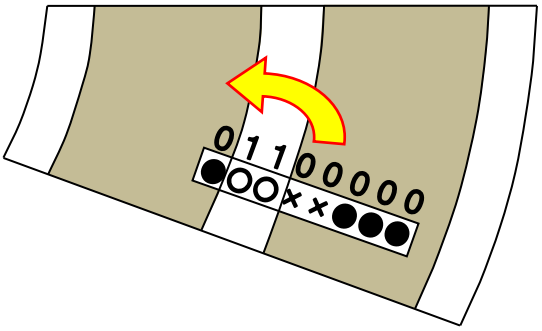
5		実際は、右に大きく寄っているので、この状態で右にハンドルを曲げると、脱輪してしまいます。

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

パターン 11 の case 0xc0 部分

```
446 :               case 0xc0:
447 :                   /* Large amount right of center -> large turn to left */
448 :                   handle( -25 );
449 :                   motor( 19, 30 );
450 :                   pattern = 13;     ←追加　パターン 13 へ移る
451 :                   break;
```

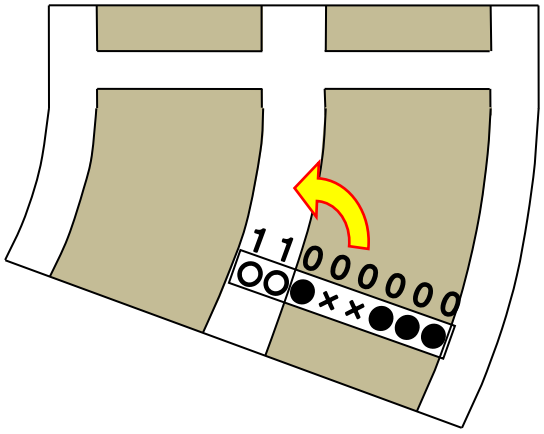
6		センサが「1100 0000」になったなら、パターン 13 に移ります。 今回パターン 13 では、この 1 つ内側のセンサ状態である「0110 0000」になったなら、パターン 11 に戻るプログラムを考えました。 この考え方で良いか検証してみます。
7		センサが「0000 0011」になりました。「0110 0000」ではないので、まだ左に曲げ続けます。 先ほどは左に寄っていると勘違いしましたが、今回はパターン 13 なので勘違いしません。
8		センサが「0000 0110」になりました。「0110 0000」ではないので、まだ左に曲げ続けます。
9		センサが「0010 0000」になりました。落ちそうですが、プログラムとしては「0110 0000」ではないので、まだ左に曲げ続けます。 ただ、車体は落ちているかもしれません。

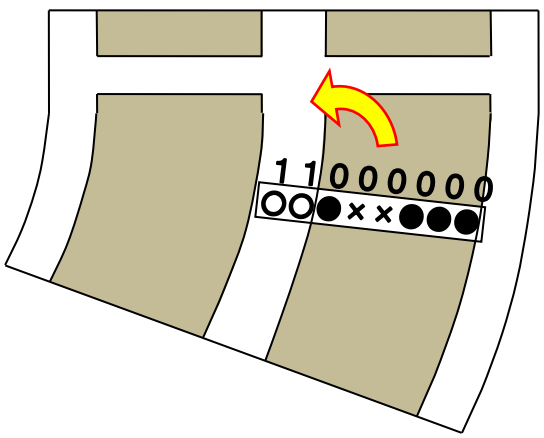
10		右に寄っていたのが左に戻り始め、センサが「0110 0000」になりました。 この状態でパターン 11 に戻ります。

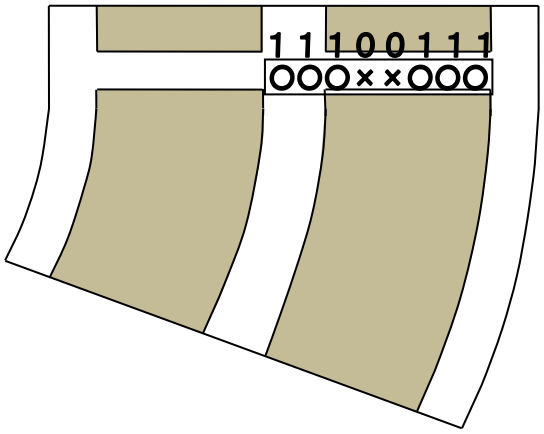
この考え方でプログラムします。

```
        case 13:
            /* Check end of large turn to left */
            if( sensor_inp( &sensor0, MASK3_3 ) == 0x60 ) {
                pattern = 11;
            }
            break;
```

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

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

12		センサは「0110 0000」ではないので、左に曲げ続けます。

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

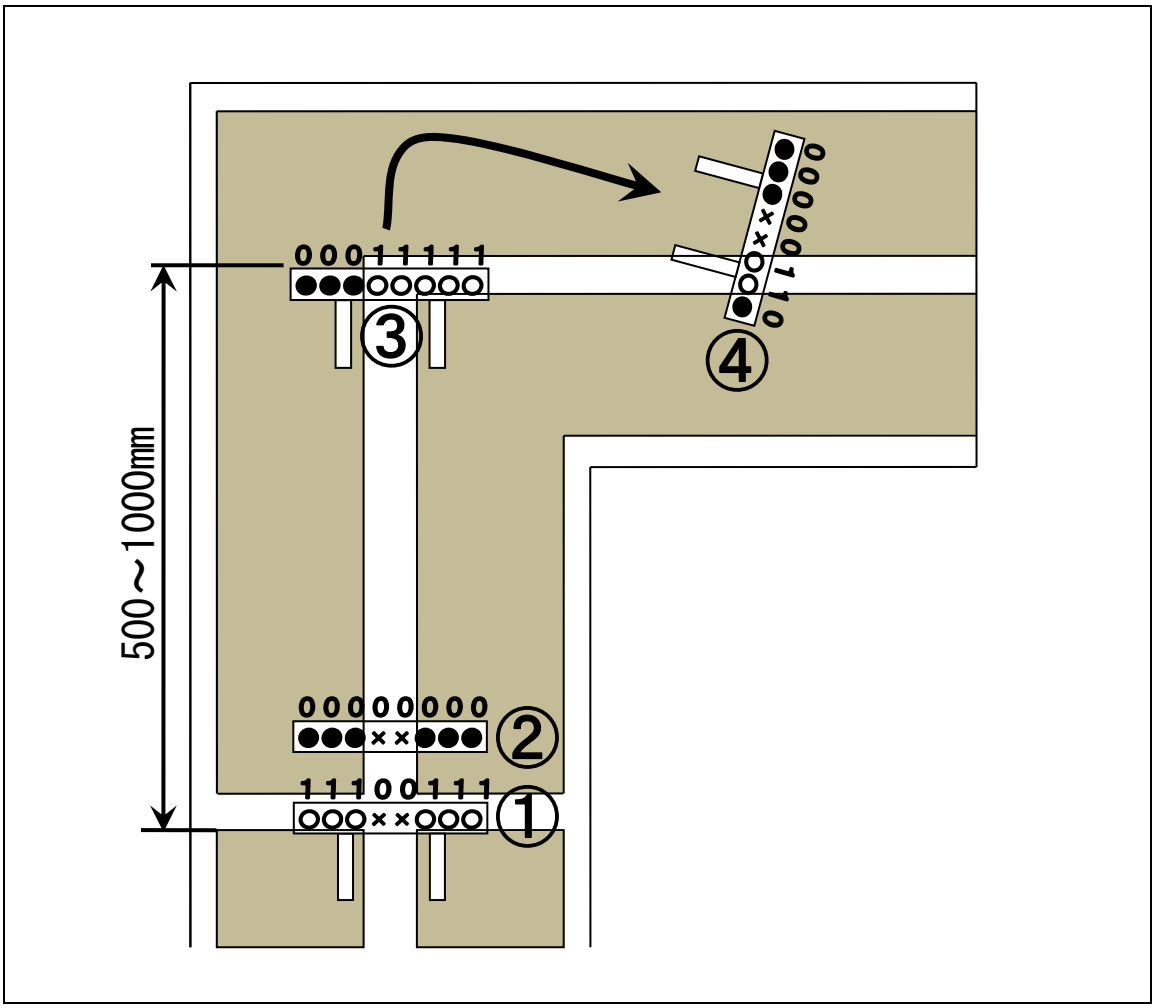
最終的なプログラムを下記に示します。

```
477 :      case 13:
478 :          /* Check end of large turn to left */
479 :          if( sensor_inp( &sensor0, MASK3_3 ) == 0x60 ) {
480 :              pattern = 11;
481 :          }
482 :          if( check_crossline() ) { /* Cross line check */
483 :              pattern = 21;
484 :              break;
485 :          }
486 :          if( check_rightline() ) { /* Right half line detection check */
487 :              pattern = 51;
488 :              break;
489 :          }
490 :          if( check_leftline() ) { /* Left half line detection check */
491 :              pattern = 61;
492 :              break;
493 :          }
494 :          break;
```

パターン 13 のプログラムはこれで完成です。

6.2.23 クランク概要

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

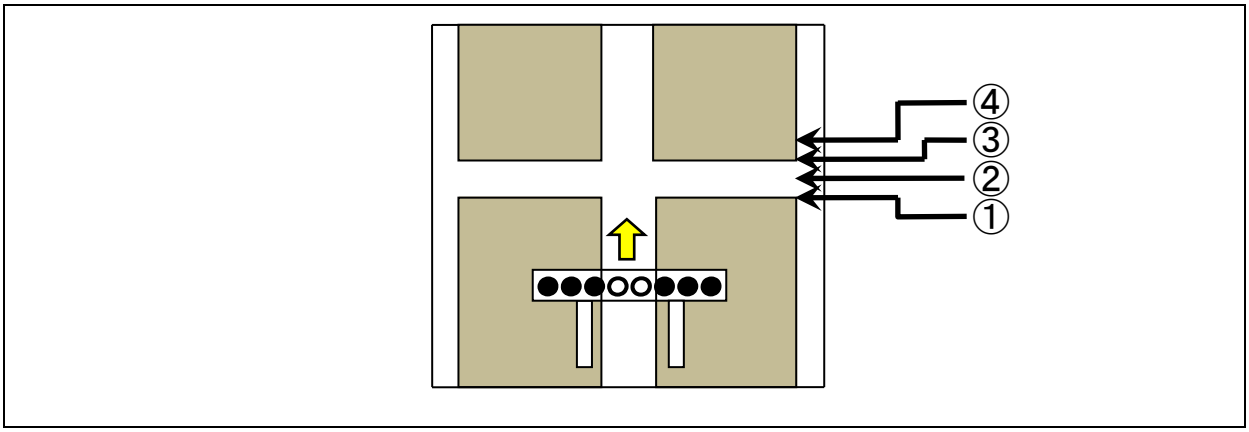


①	check_crossline 関数でクロスラインを検出します。500～1000mm 先で、右クランクか左クランクがあるので、クリアできるスピードにするためブレーキをかけます。また、クロスライン通過中にセンサが誤検出しないよう②の位置までセンサは見ません。
②	この位置から徐行開始します。中心線をトレースしながら進んでいきます。
③	クランクを検出すると、クランクがある方向にハンドルを切ります。
④	中心線を検出すると、パターン 11 に戻りライントレースを再開します。

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

6.2.24 パターン 21:クロスライン検出時の処理

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



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

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

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

これをパターン 21 でプログラム化します。

```
    case 21:
        /* Processing at 1st cross line */
        led_out( 0x3 );
        handle( 0 );
        motor( 0 ,0 );
        if( cnt1 > 100 ) {
            pattern = 22;
        }
        break;
```

完成しました。本当にこれでよいか見直してみます。cnt1 が 100 以上になったら(100 ミリ秒たったら)、パターン

22 へ移るようにしています。これはパターン 21 を開始したときに、cnt1 が 0 になっている必要があります。例えばパターン 21 にプログラムが移ってきた時点で cnt1 が 1000 なら、1 回目の比較で cnt1 は 100 以上と判断して、すぐにパターン 22 に移ってしまいます。0.1 秒どころか、1 回しかパターン 21 が実行しません(約数 10μs)。そこで、パターンをもう一つ増やします。パターン 21 はブレーキをかけ cnt1 をクリア、パターン 22 で 0.1 秒たったかチェックするようにします。

再度まとめると、下記のようになります。

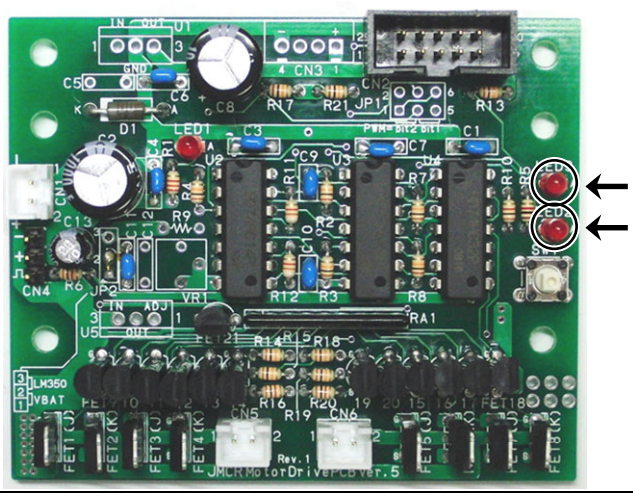
パターン 21 で行うこと	• LED2,3 を点灯 • ハンドルを 0 度に • 左右モータ PWM を 0%にしてブレーキをかける • パターンを次へ移す • cnt1 をクリア
パターン 22 で行うこと	• cnt1 が 100 以上になったなら、次のパターンへ移す

上記にしたがって再度プログラムを作ってみます。

```
496 :       case 21:
497 :           /* Processing at 1st cross line */
498 :           led_out( 0x3 );
499 :           handle( 0 );
500 :           motor( 0 ,0 );
501 :           pattern = 22;
502 :           cnt1 = 0;
503 :           break;
504 :
505 :       case 22:
506 :           /* Read but ignore 2nd line */
507 :           if( cnt1 > 100 ) {
508 :               pattern = 23;
509 :               cnt1 = 0;
510 :           }
511 :           break;
```

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

ポイント



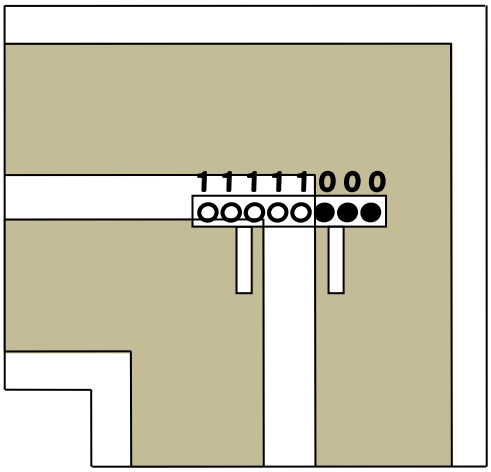
クロスラインを検出すると、モータドライブ基板の LED が 2 個点きます。点いていなければ、クロスラインを検出していません。
クロスラインの検出がうまくできているか分からない場合は、モータのコネクタを抜いて、マイコンカーを手で押しながら進めて確かめてください。

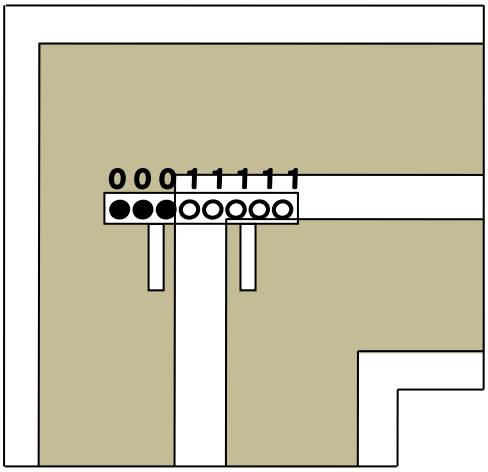
6.2.25 パターン 23:クロスライン後のトレース、クランク検出

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

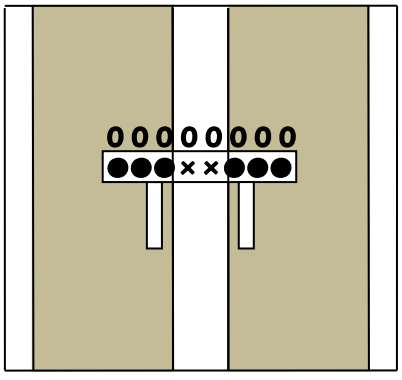
クロスラインを過ぎたので、後はクランク(直角)の検出です。クランクを見つけたらすぐに曲げなければいけませんので徐行して進んでいきます。またクランクまでの間、中心線をトレースしながら進んでいく処理も必要です。

今回、下図のように考えました。

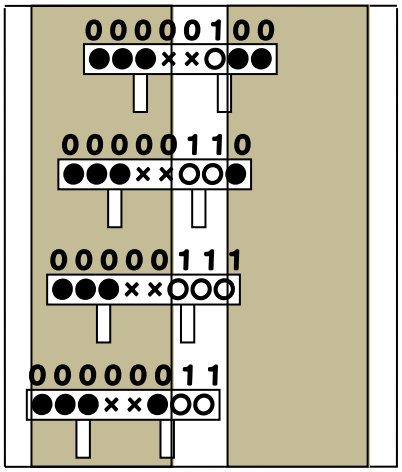
<table>
<tr>
<td> 

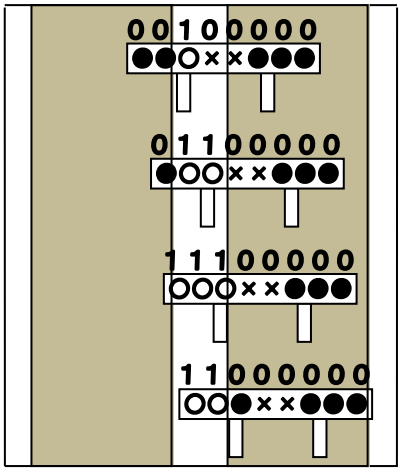
0x f8(8個すべてチェック)</td>
<td>左クランク部分では、8 個のセンサ状態が左図のように「0xf8」になりました。そこで、「0xf8」の状態を左クランクと判断するようにします。
このとき、ハンドルを左いっぱいまで曲げなければ外側へ脹らんで脱輪してしまいます。何度曲げるか… それはマイコンカーの作りによって違いますので、実際のマイコンカーを見て何処までハンドルが切れるか確かめる必要があります。プロジェクト「sioservo2_38a」でハンドルの最大切れ角を調べるとキットは大体 40 度くらいです。2 度くらい余裕を見て、プログラムでは 38 度にします。
左モータ、右モータの回転数は、左に大きく曲げるので、左モータを少なく、右モータを多めにします。実際に何%にするかは、走らせて確かめるとして、ここでは、左モータ 10%、右モータ 50%にしておきます。まとめると下記のようになります。
ハンドル:-38 度
左モータ:10% 右モータ:50%
その後、パターン 31 へ移ります。</td>
</tr>
<tr>
<td> 

0x1f(8個すべてチェック)</td>
<td>右クランク部分です。考え方は左クランクと同様です。
まとめると下記のようになります。
ハンドル:38 度
左モータ:50% 右モータ:10%

その後、パターン 41 へ移ります。</td>
</tr>
<tr>
<td> 

0x00</td>
<td>直進時、センサの状態は「0x00」です。マイコンカーは中心にあると判断します。ハンドルは 0 度です。問題はモータの PWM 値です。モータの PWM 値は、クランクを見つけたときに直角を曲がれるスピードにしなければいけません。ここでは 40%にしておき、実際に走らせて微調整することにします。まとめると下記のようになります。
ハンドル:0 度
左モータ:40% 右モータ:40%</td>
</tr>
</table>

<table>
<tr>
<td> 

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

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

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

プログラム化すると下記のようになります。

```
513 :      case 23:
514 :          /* Trace, crank detection after cross line */
515 :          if( sensor_inp( &sensor0, MASK4_4 ) == 0xf8 ) {
516 :              /* Left crank determined -> to left crank clearing processing */
517 :              led_out( 0x1 );
518 :              handle( -38 );
519 :              motor( 10 ,50 );
520 :              pattern = 31;
521 :              cnt1 = 0;
522 :              break;
523 :          }
524 :          if( sensor_inp( &sensor0 , MASK4_4 ) == 0x1f ) {
525 :              /* Right crank determined -> to right crank clearing processing */
526 :              led_out( 0x2 );
527 :              handle( 38 );
528 :              motor( 50 ,10 );
529 :              pattern = 41;
530 :              cnt1 = 0;
531 :              break;
532 :          }
533 :          switch( sensor_inp( &sensor0, MASK3_3 ) ) {
534 :              case 0x00:
535 :                  /* Center -> straight */
536 :                  handle( 0 );
537 :                  motor( 40 ,40 );
538 :                  break;
539 :              case 0x04:
540 :              case 0x06:
541 :              case 0x07:
542 :              case 0x03:
543 :                  /* Left of center -> turn to right */
544 :                  handle( 8 );
545 :                  motor( 40 ,35 );
546 :                  break;
547 :              case 0x20:
548 :              case 0x60:
549 :              case 0xe0:
550 :              case 0xc0:
551 :                  /* Right of center -> turn to left */
552 :                  handle( -8 );
553 :                  motor( 35 ,40 );
554 :                  break;
555 :          }
556 :          break;
```

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

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

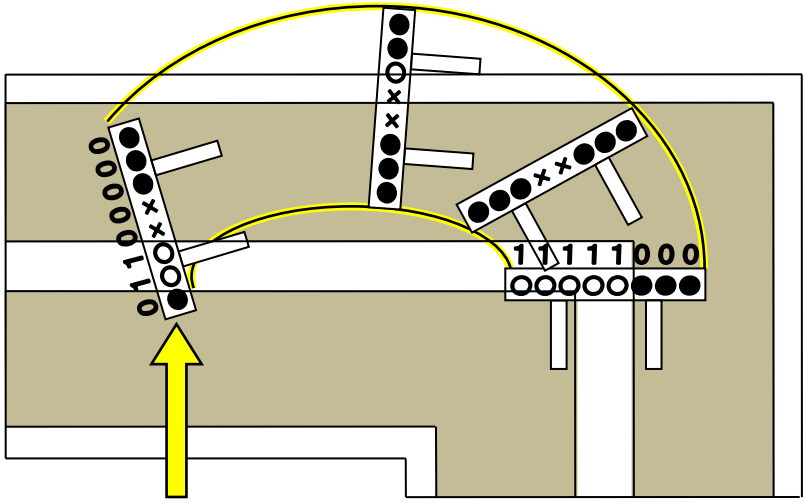
左クランク、右クランクは、if 文で判定しています。その他は、switch 文を使って「sensor_inp(MASK3_3)」の値で case へ分岐するようにしました。

6.2.26 パターン 31、32:左クランククリア処理

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

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

下図のように考えました。



センサの状態が「0x60」になったなら曲げ終わり、
パターン 11 へ戻る

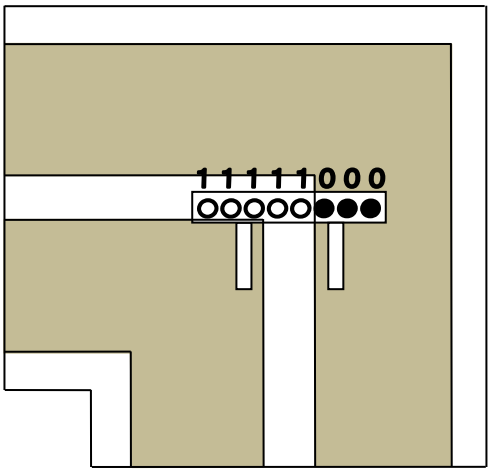
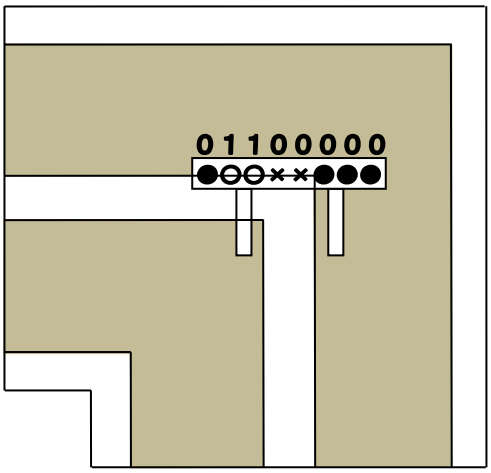
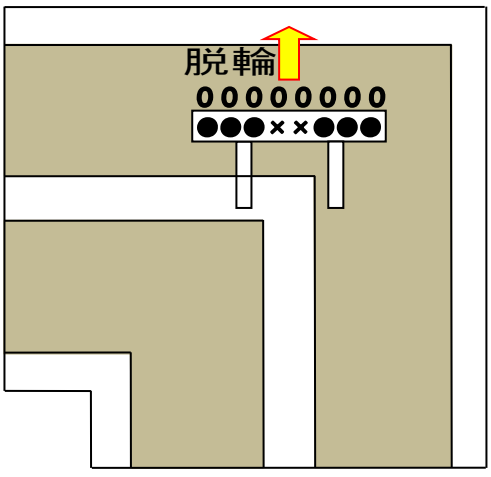
「0xf8」と判断すると左へ大曲げしますが、スピードがついているので膨らみ気味に曲がっていきます。センサが中心線付近に来て「0x60」になったときを曲げ終わりと判断してパターン 11 へ戻ります。

これをプログラム化してみます。

```
case 31:
if( sensor_inp( &sensor0, MASK3_3 ) == 0x60 ) {
    pattern = 11;
}
break;
```

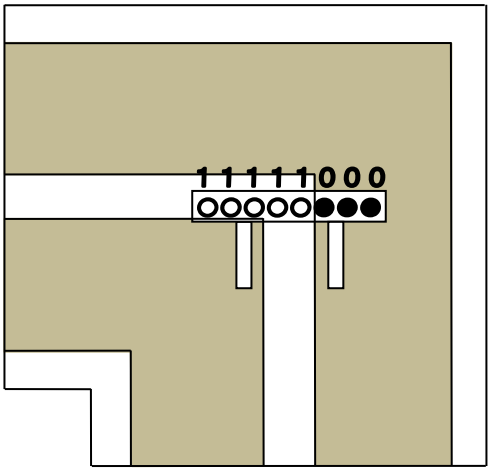
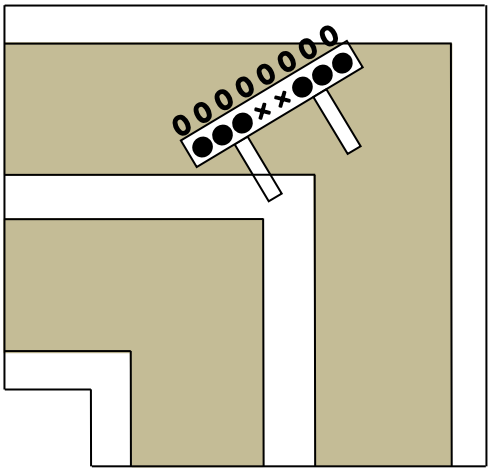
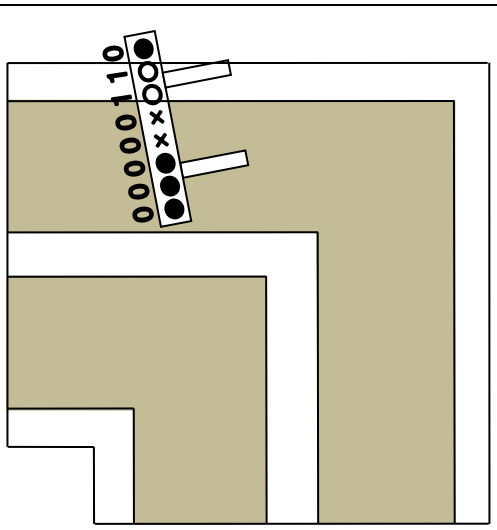
上記プログラムで実際に走らせてみました。左クランクでセンサの状態が「0xf8」になった瞬間、ハンドルが左へ曲がり始めました。想定では、センサが「0x60」になるまで曲げ続けるはずでしたが、すぐにハンドルがまっすぐ向いてしまい、クランク部分を直進し脱輪してしまいました。動作が早すぎてよく分からないので、モータとサーボのコネクタを抜いて左クランク部分を手で押しながらゆっくりと進ませていきます。センサの状態をじっくりと観察すると次の図のようになっていました。

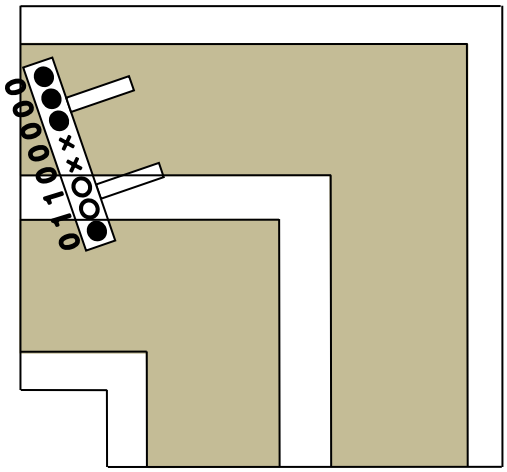
参考
マイコンカーの動きがよく分からない場合は、モータのコネクタを抜いて、
手で押しながらセンサ状態を確認することをお勧めします。

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

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

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

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

7		センサの状態が「0x60」になりました。パターン 11 へ移り通常トレースを行います。

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

```
558 :       case 31:
559 :           /* Left crank clearing processing ? wait until stable */
560 :           if( cnt1 > 200 ) {
561 :               pattern = 32;
562 :               cnt1 = 0;
563 :           }
564 :           break;
565 :
566 :       case 32:
567 :           /* Left crank clearing processing ? check end of turn */
568 :           if( sensor_inp( &sensor0, MASK3_3 ) == 0x60 ) {
569 :               led_out( 0x0 );
570 :               pattern = 11;
571 :               cnt1 = 0;
572 :           }
573 :           break;
```

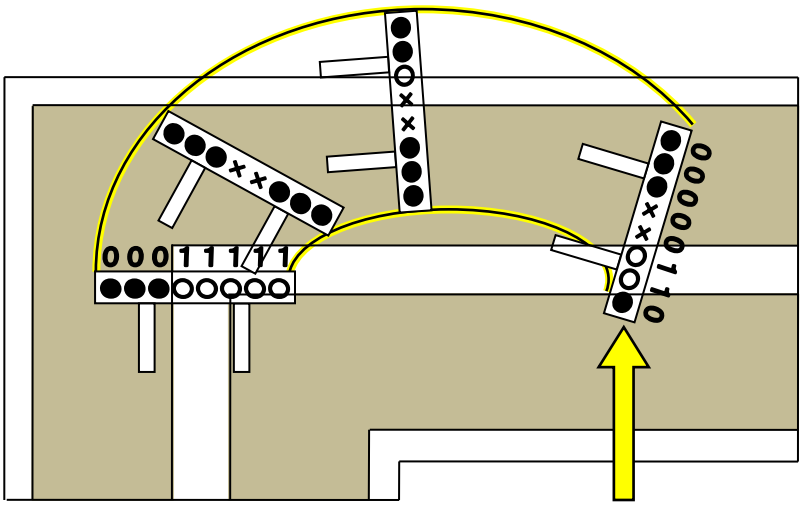
317 行で cnt1 が 200 以上かチェックしています。200 以上なら、すなわち 0.2 秒たったならパターン 32 へ移ります。ちなみに cnt1 のクリアは、パターン 31 へ移る前の 278 行で行っています。

6.2.27 パターン 41、42:右クランククリア処理

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

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

下図のように考えました。



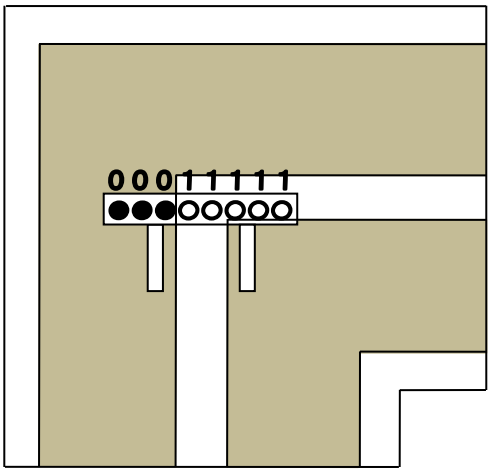
「0x1f」と判断すると右へ大曲げしますが、スピードがついているので膨らみ気味に曲がっていきます。センサが中心線付近に来て「0x06」になったときを曲げ終わりと判断してパターン 11 へ戻ります。

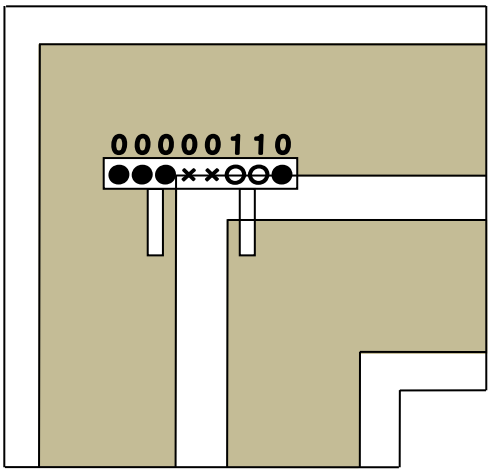
これをプログラム化してみます。

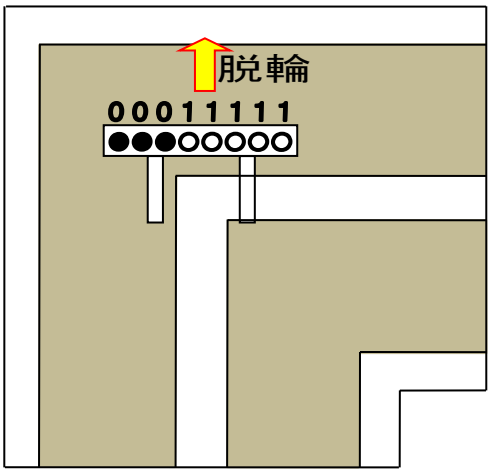
```
        case 41:
            /* Right crank clearing processing ? check end of turn */
            if( sensor_inp( &sensor0, MASK3_3 ) == 0x06 ) {
                pattern = 11;
            }
            break;
```

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

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

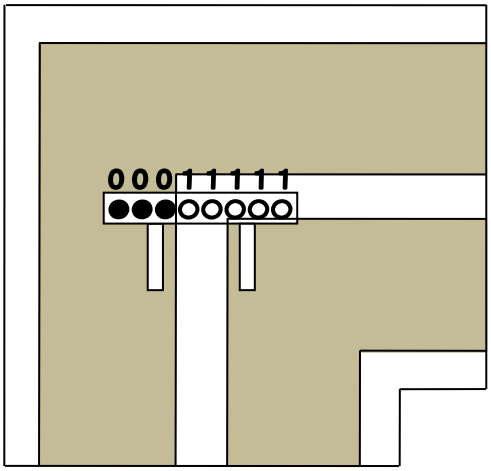
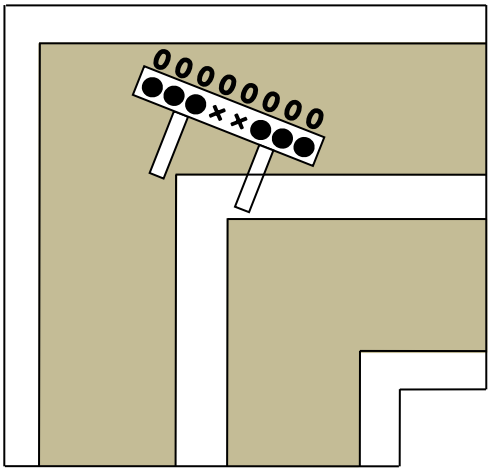
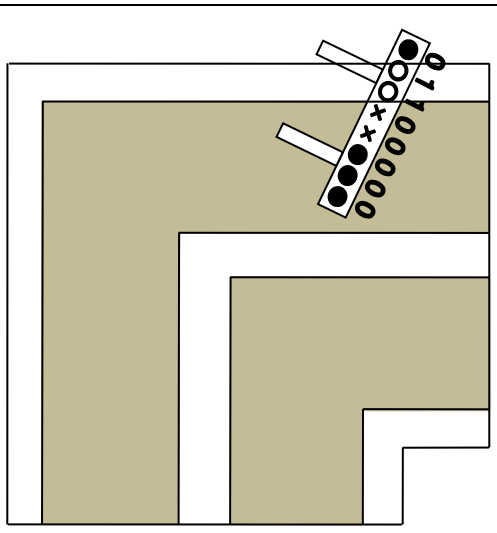
1		センサの状態が「0x1f」になったので、センサの状態が「0x06」になるまで右に 38 度曲げます。

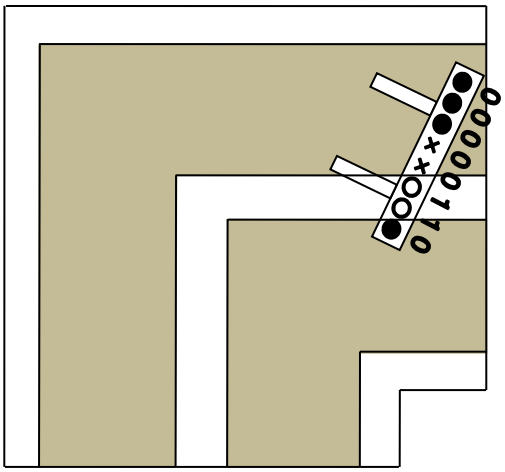
2		ハンドルを曲げ始めた瞬間、白と黒の境目で(本当は白と灰と黒色ですが、灰色は白と見なします)センサの状態が「0x06」になりました。 プログラムでは、「0x06」になったので、この状態でパターン 11 に移ります。

3		パターン 11 では、センサ状態「0x00」はセンサが中心にあると判断して、ハンドル 0 度、モータは 100%で走行します。 まっすぐ進んでしまい脱輪します。

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

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

4		センサの状態が「0x1f」になったので、ハンドルを右に 38 度曲げます。その後、センサを見ずに 0.2 秒進ませます。
5		0.2 秒後です。ここからセンサの状態が「0x06」かどうかチェックします。
6		まだセンサの状態が「0x06」ではないので曲げ続けます。

7		センサの状態が「0x06」になりました。パターン 11 へ移り通常トレースを行います。

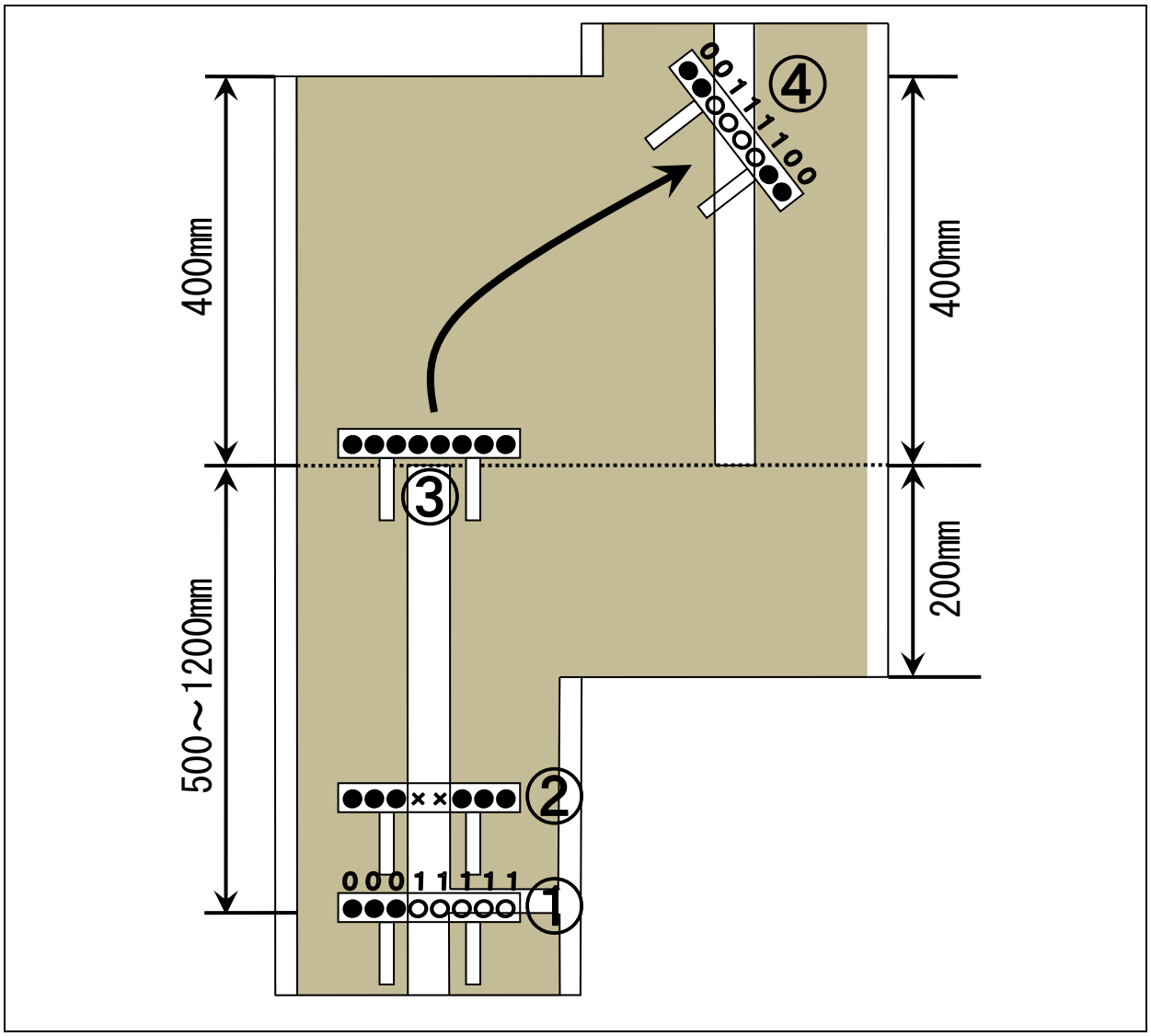
クランク検出を検出した 0.2 秒後、図 5 のように白と黒の境目を越えた位置にあります。その後は「0x06」になるまで安心して曲げ続けるだけです。これなら良さそうです。プログラム化します。

```
575 :      case 41:
576 :          /* Right crank clearing processing ? wait until stable */
577 :          if( cnt1 > 200 ) {
578 :              pattern = 42;
579 :              cnt1 = 0;
580 :          }
581 :          break;
582 :
583 :      case 42:
584 :          /* Right crank clearing processing ? check end of turn */
585 :          if( sensor_inp( &sensor0, MASK3_3 ) == 0x06 ) {
586 :              led_out( 0x0 );
587 :              pattern = 11;
588 :              cnt1 = 0;
589 :          }
590 :          break;
```

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

6.2.28 右レーンチェンジ概要

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

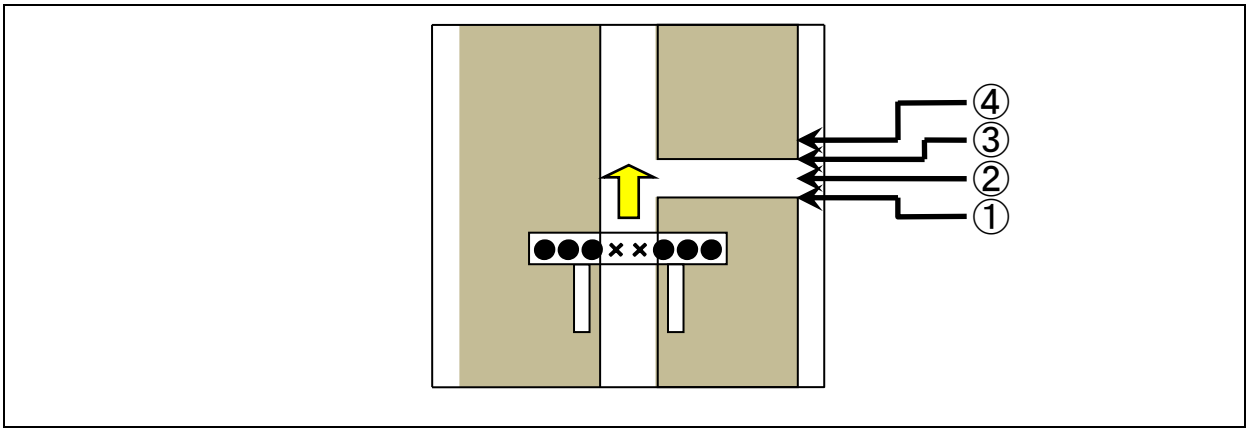


①	check_rightline 関数で右ハーフラインを検出します。500～1200mm 先で、右レーンに移動するために右に曲がらなければいけないのでブレーキをかけます。また、右ハーフライン通過時にセンサが誤検出しないよう②の位置までセンサは見ません。
②	この位置から徐行開始します。中心線をトレースしながら進んでいきます。
③	中心線が無くなると、右へハンドルを切ります。
④	新しい中心線を検出すると、今度はこの中心線でライントレースを再開します。

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

6.2.29　パターン 51：右ハーフライン検出時の処理

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



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

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

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

これをパターン 51 でプログラム化します。

```
        case 51:
            /* Processing at 1st right half line detection */
            led_out( 0x2 );
            handle( 0 );
            motor( 0 ,0 );
            if( cnt1 > 100 ) {
                pattern = 52; /* 0.1 秒後パターン 52 へ*/
            }
            break;
```

完成しました。本当にこれでよいか見直してみます。cnt1 が 100 以上になったら(100 ミリ秒たったら)、パターン

52 へ移るようにしています。これはパターン 51 を開始したときに、cnt1 が 0 になっている必要があります。例えばパターン 51 にプログラムが移ってきた時点で cnt1 が 1000 であったら、1 回目の比較で cnt1 は 100 以上と判断して、すぐにパターン 52 に移ってしまいます。0.1 秒どころか、1 回しかパターン 51 を実行しません(約数 10μs)。そこで、パターンをもう一つ増やします。パターン 51 でブレーキをかけ cnt1 をクリア、パターン 52 で 0.1 秒たったかチェックするようにします。

再度まとめると、下記のようになります。

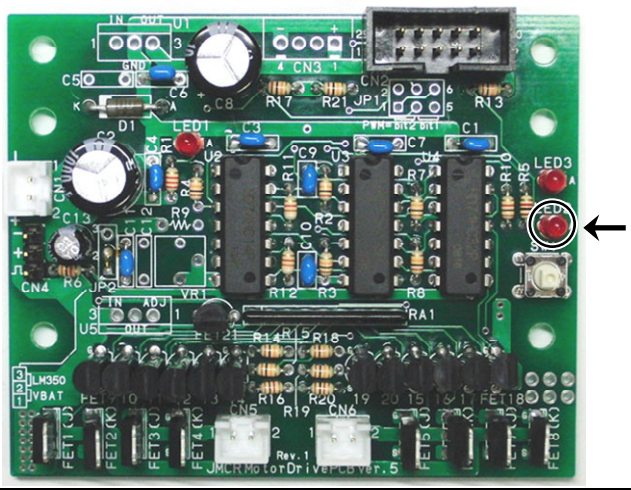
パターン 51 で行うこと	• LED2 を点灯 • ハンドルを 0 度に • 左右モータ PWM を 0%にしてブレーキをかける • パターンを次へ移す • cnt1 をクリア
パターン 52 で行うこと	• cnt1 が 100 以上になったなら、次のパターンへ移す

上記にしたがって再度プログラムを作ってみます。

```
592 :       case 51:
593 :           /* Processing at 1st right half line detection */
594 :           led_out( 0x2 );
595 :           handle( 0 );
596 :           motor( 0 ,0 );
597 :           pattern = 52;
598 :           cnt1 = 0;
599 :           break;
600 :
601 :       case 52:
602 :           /* Read but ignore 2nd time */
603 :           if( cnt1 > 100 ){
604 :               pattern = 53;
605 :               cnt1 = 0;
606 :           }
607 :           break;
```

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

ポイント



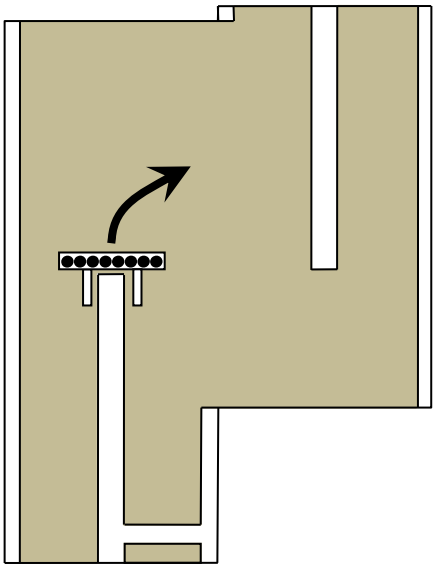
右ハーフラインを検出すると、モータドライブ基板の LED が 1 個点きます(下側)。
点いていなければ、右ハーフラインを検出していません。
右ハーフラインの検出がうまくできているか分からない場合は、モータのコネクタを抜いて、マイコンカーを手で押しながら進めて確かめてください。

6.2.30 パターン 53:右ハーフライン後のトレース

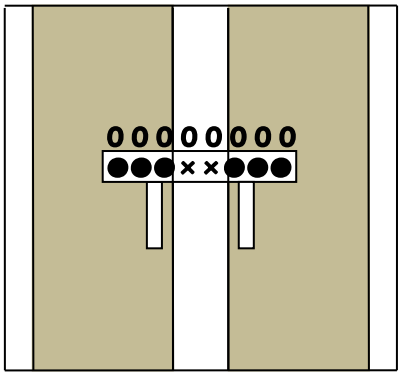
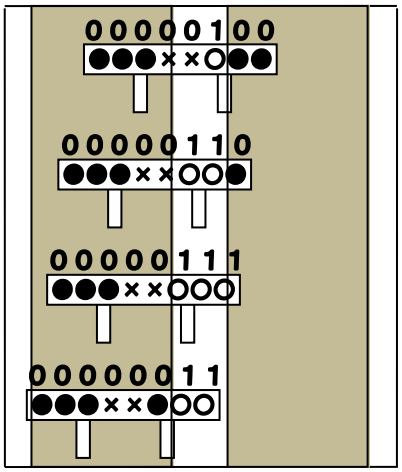
パターン 51、52 では、右ハーフラインを検出後、ブレーキを 0.1 秒かけ右ハーフラインを通過させました。パターン 53 では、その後の処理を行います。

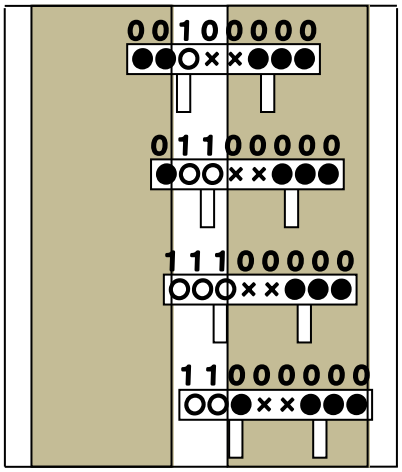
右ハーフラインを過ぎたので、後は中心線が無くなったかどうかチェックしながら進んでいきます。また中心線が無くなるまでの間、直線をトレースしなければいけませんので通常トレースも必要です。

今回は、下図のように考えました。

<table>
<tr>
<td> 
O×OO(8 個すべてチェック)</td>
<td>右ハーフライン検出後、トレースしていき中心線が無くなると、8 個のセンサ状態が左図のように「0x00」になりました。この状態を検出すると、右曲げを開始します。
このとき、右に曲がるので、右モータの回転数を少なく、左モータの回転数を多めにすることは予想できます。実際に何%にすればよいかは、マイコンカーのスピードやタイヤの滑り具合、サーボの反応速度によって変わるので、実際のマイコンカーを見て決めます。ここでは下記のように決め、後は実際に走らせて決めたいと思います。

ハンドル:15 度
左モータ:40%　右モータ:31%

その後、パターン 54 へ移ります。</td>
</tr>
<tr>
<td> 
0x00</td>
<td>直進時、センサの状態は「0x00」です。これを直進状態と判断します。ハンドルをまっすぐにしなければいけないのは、疑いの余地がありません。問題はモータの PWM 値です。こちらは実際に走らせなければどのくらいのスピードになるのか何とも言えません。中心線が無くなったら曲がれるスピードにしなければいけません。とりあえず 40%にしておき、実際に走らせて微調整することにします。まとめると下記のようになります。
ハンドル:0 度
左モータ:40%　右モータ:40%</td>
</tr>
<tr>
<td> 
0x04
0x06
0x07
0x03</td>
<td>マイコンカーが左に寄ったときを考えています。中心から少しずつ左へずらしていくと左図のように 4 つの状態になりました。センサの状態はもっと左へ寄ることも考えられますが、右ハーフラインの後は直線しかないと分かっているので、これ以上センサ状態は増やさないでおきます。
動作は、左へ寄っているので右へハンドルを切ります。切り角が小さすぎるとずれが大きいときに戻りきれなくなり、大きすぎるとセンサが左右にばたばた振れてしまいます。ちょうど良い角度調整は難しいです。今回は、とりあえずどの状態も 8 度にします。まとめると下記のようになります。
ハンドル:8 度
左モータ:40%　右モータ:35%</td>
</tr>
</table>



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

ポイントは、中心線があるかどうかのチェックは 8 個のセンサすべてを使用することです。他は「MASK3_3」でマスクして中心の 2 個のセンサは使用しません。
プログラム化すると下記のようになります。

```
609 :       case 53:
610 :           /* Trace, lane change after right half line detection */
611 :           if( sensor_inp( &sensor0, MASK4_4 ) == 0x00 ) {
612 :               handle( 15 );
613 :               motor( 40 ,31 );
614 :               pattern = 54;
615 :               cnt1 = 0;
616 :               break;
617 :           }
618 :           switch( sensor_inp( &sensor0, MASK3_3) ) {
619 :               case 0x00:
620 :                   /* Center -> straight */
621 :                   handle( 0 );
622 :                   motor( 40 ,40 );
623 :                   break;
624 :               case 0x04:
625 :               case 0x06:
626 :               case 0x07:
627 :               case 0x03:
628 :                   /* Left of center -> turn to right */
629 :                   handle( 8 );
630 :                   motor( 40 ,35 );
631 :                   break;
632 :               case 0x20:
633 :               case 0x60:
634 :               case 0xe0:
635 :               case 0xc0:
636 :                   /* Right of center -> turn to left */
637 :                   handle( -8 );
638 :                   motor( 35 ,40 );
639 :                   break;
640 :               default:
641 :                   break;
642 :           }
643 :           break;
```

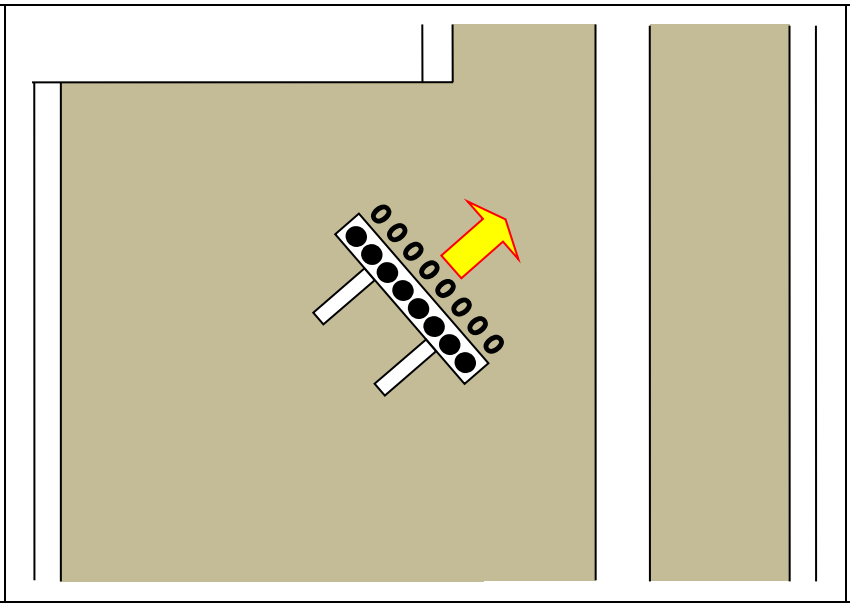
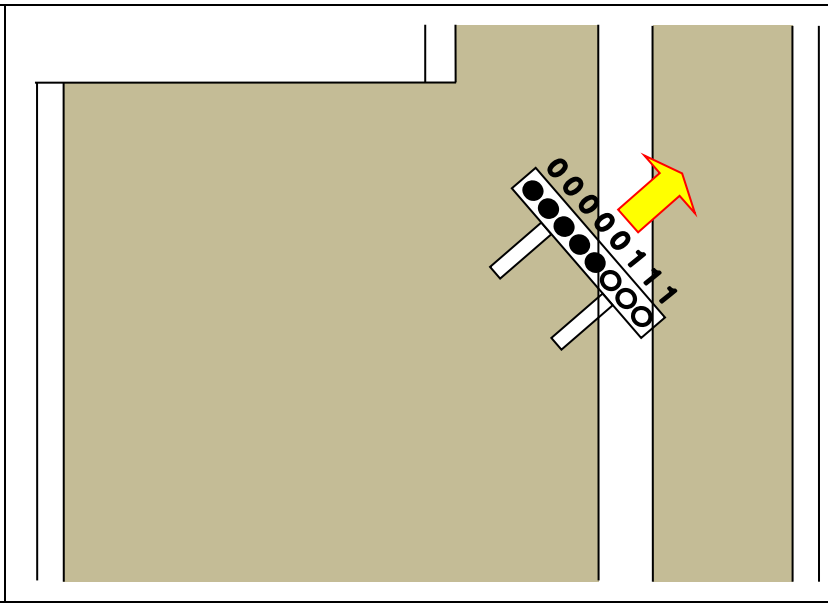
case を続けて書くと
0x04 または 0x06 または 0x07 または 0x03 のとき
という意味になります。

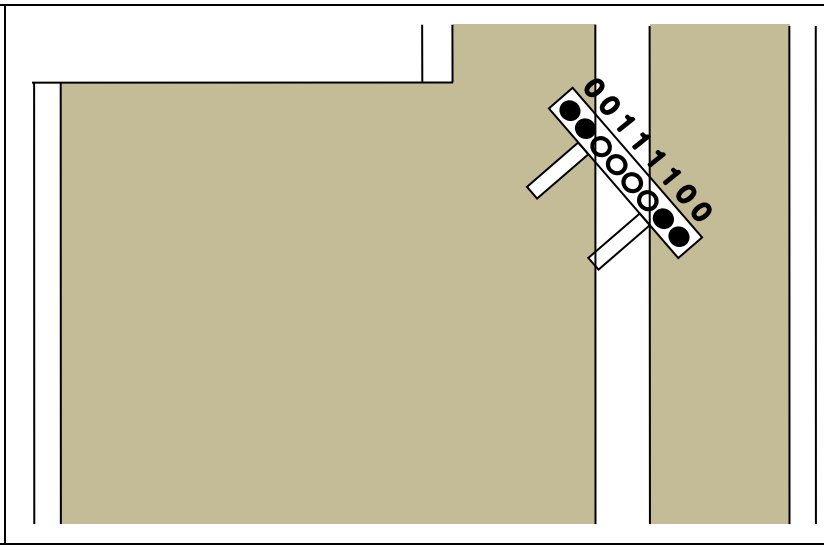
case を続けて書くと
0x20 または 0x60 または 0xe0 または 0xc0 のとき
という意味になります。

6.2.31 パターン 54:右レーンチェンジ終了のチェック

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

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

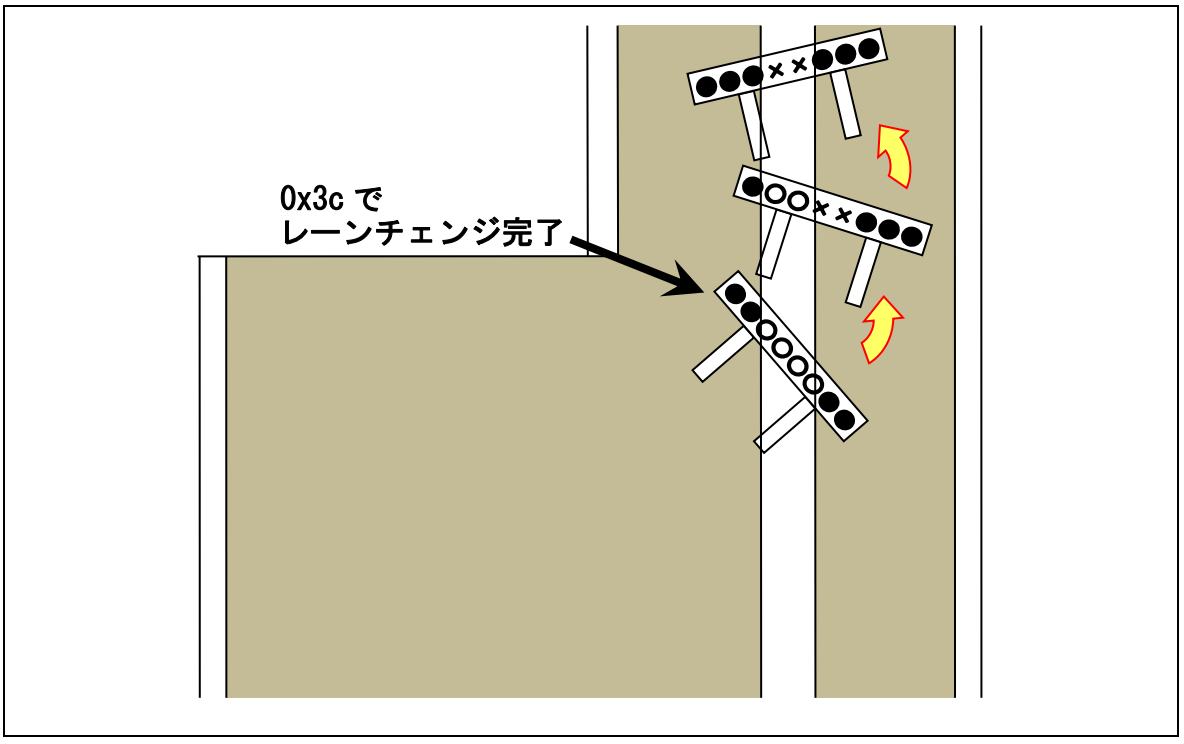
1		曲げながら進んできました。新しい中心線を見つける直前です。
2		中心線をセンサの右端で検出しました。 センサの状態は、 「0000 0111」です。

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

上記の考え方で、プログラムします。

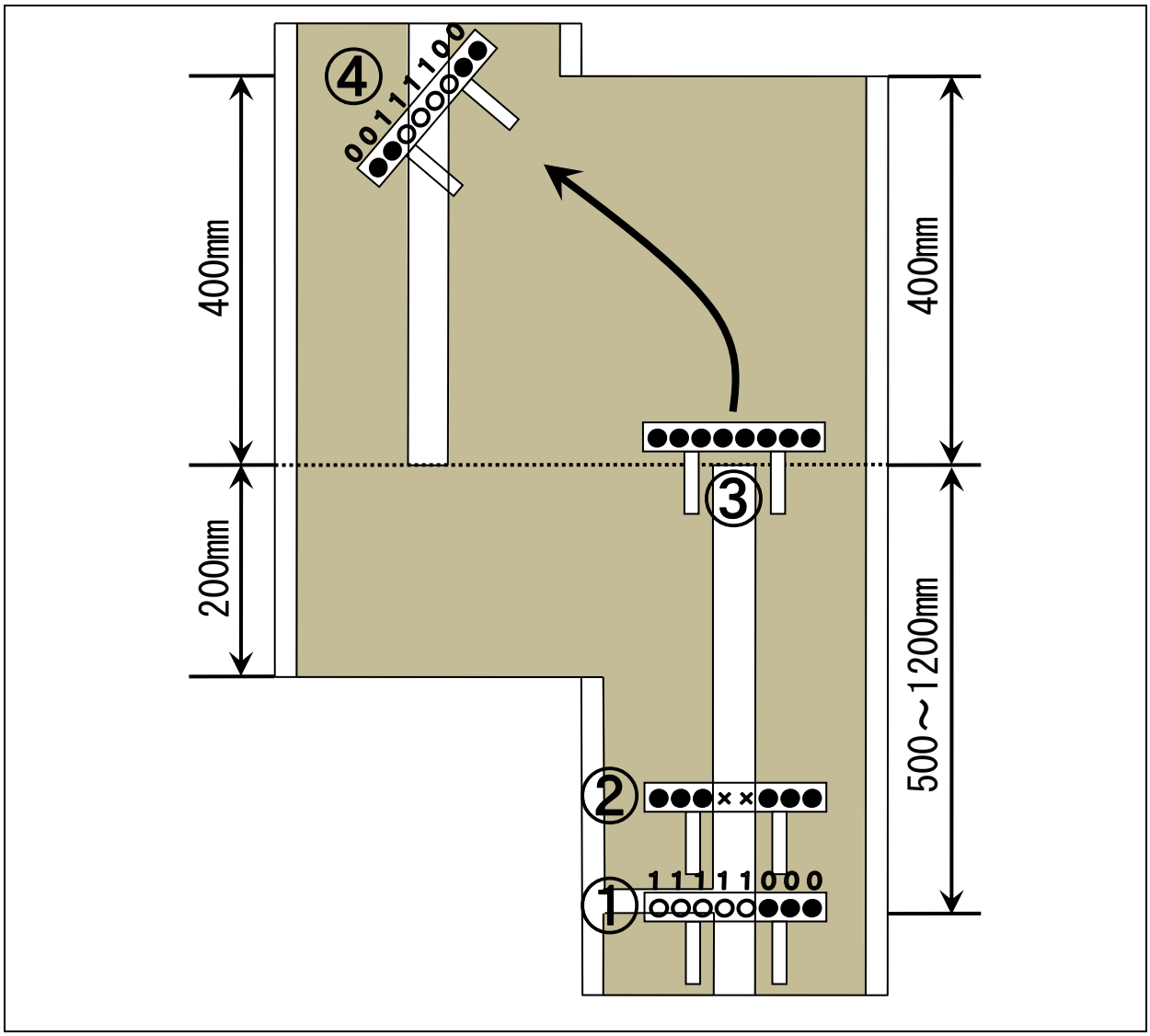
```
645 :       case 54:
646 :           /* Right lane change end check */
647 :           if( sensor_inp( &sensor0, MASK4_4 ) == 0x3c ) {
648 :               led_out( 0x0 );
649 :               pattern = 11;
650 :               cnt1 = 0;
651 :           }
652 :           break;
```

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



6.2.32 左レーンチェンジ概要

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

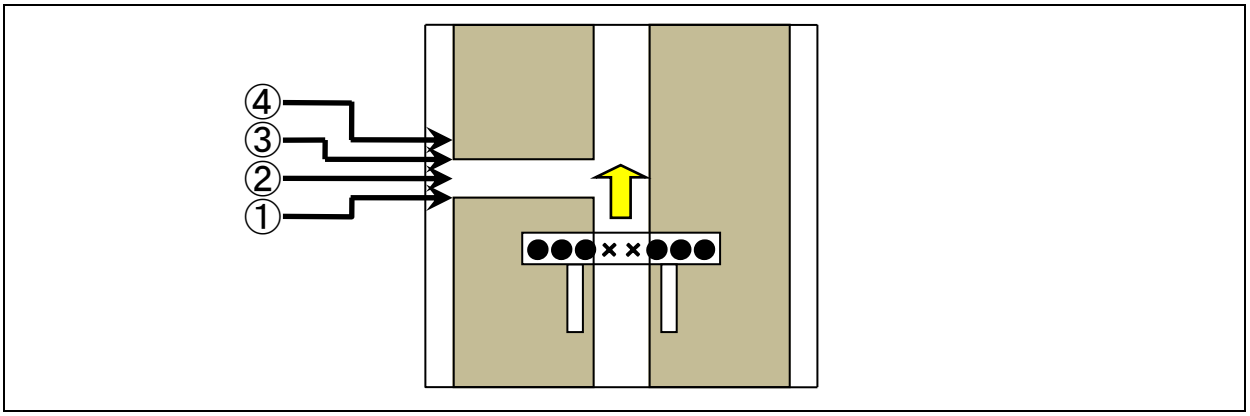


①	check_leftline 関数で左ハーフラインを検出します。500～1200mm 先で、左レーンに移動するために左に曲がらなければいけないのでブレーキをかけます。また、左ハーフライン通過時にセンサが誤検出しないよう②の位置までセンサは見ません。
②	この位置から徐行開始します。中心線をトレースしながら進んでいきます。
③	中心線が無くなると、左へハンドルを切ります。
④	新しい中心線を検出すると、今度はこの中心線でライントレースを再開します。

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

6.2.33 パターン 61:左ハーフライン検出時の処理

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



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

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

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

まとめると、

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

これをパターン 61 でプログラム化します。

```
        case 61:
            /* Processing at 1st left half line detection */
            led_out( 0x1 );
            handle( 0 );
            motor( 0 ,0 );
            if( cnt1 > 100 ) {
                pattern = 62; /* 0.1 秒後パターン 62 へ*/
            }
            break;
```

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

再度まとめると、下記のようになります。

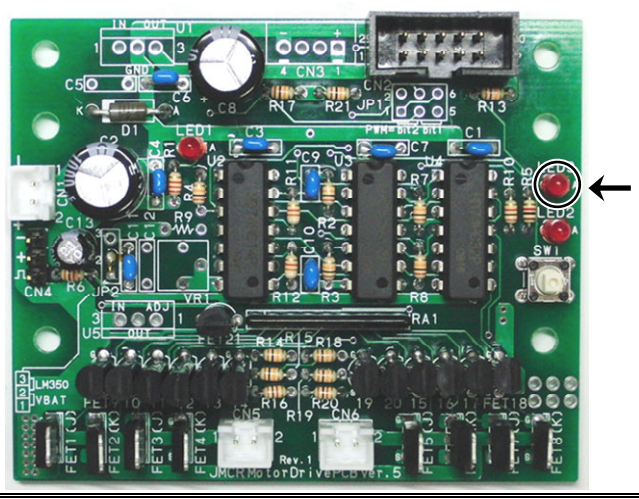
パターン 61 で行うこと	• LED3 を点灯 • ハンドルを 0 度に • 左右モータ PWM を 0%にしてブレーキをかける • パターンを次へ移す • cnt1 をクリア
パターン 62 で行うこと	• cnt1 が 100 以上になったなら、次のパターンへ移す

上記にしたがって再度プログラムを作ってみます。

```
654 :       case 61:
655 :           /* Processing at 1st left half line detection */
656 :           led_out( 0x1 );
657 :           handle( 0 );
658 :           motor( 0 ,0 );
659 :           pattern = 62;
660 :           cnt1 = 0;
661 :           break;
662 :
663 :       case 62:
664 :           /* Read but ignore 2nd time */
665 :           if( cnt1 > 100 ){
666 :               pattern = 63;
667 :               cnt1 = 0;
668 :           }
669 :           break;
```

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

ポイント



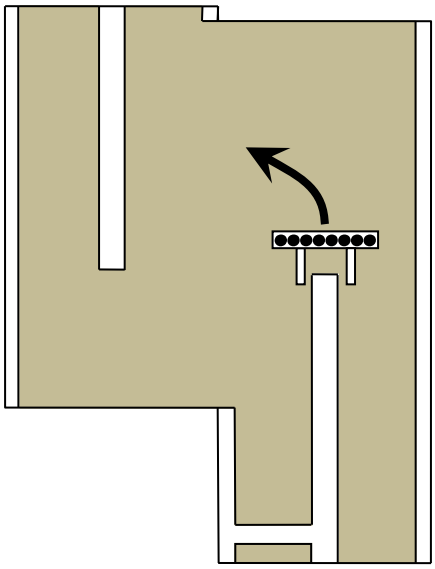
左ハーフラインを検出すると、モータドライブ基板の LED が 1 個点きます(上側)。
点いていなければ、左ハーフラインを検出していません。
左ハーフラインの検出がうまくできているか分からない場合は、モータのコネクタを抜いて、マイコンカーを手で押しながら進めて確かめてください。

6.2.34 パターン 63:左ハーフライン後のトレース

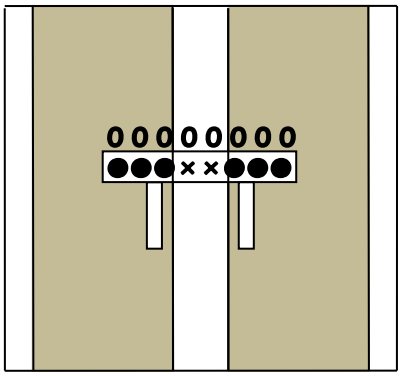
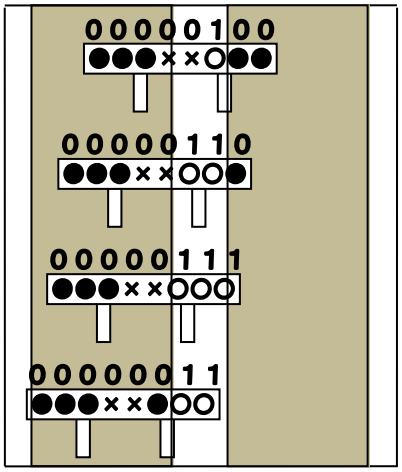
パターン 61、62 では、左ハーフラインを検出後、ブレーキを 0.1 秒かけ左ハーフラインを通過させました。パターン 63 では、その後の処理を行います。

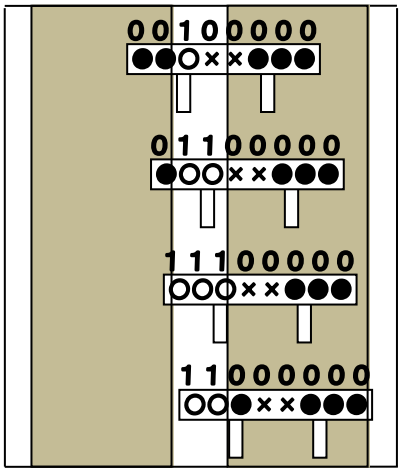
左ハーフラインを過ぎたので、後は中心線が無くなったかどうかチェックしながら進んでいきます。また中心線が無くなるまでの間、直線をトレースしなければいけませんので通常トレースも必要です。

今回は、下図のように考えました。

<table>
<tr>
<td> 
OxOO(8 個すべてチェック)</td>
<td>左ハーフライン検出後、トレースしていき中心線が無くなると、8 個のセンサ状態が左図のように「0x00」になりました。この状態を検出すると、左曲げを開始します。
このとき、左に曲がるので、左モータの回転数を少なく、右モータの回転数を多めにすることは予想できます。実際に何%にすればよいかは、マイコンカーのスピードやタイヤの滑り具合、サーボの反応速度によって変わるので、実際のマイコンカーを見て決めます。ここでは下記のように決め、後は実際に走らせて決めたいと思います。

ハンドル:-15 度
左モータ:31%　右モータ:40%

その後、パターン 64 へ移ります。</td>
</tr>
<tr>
<td> 
0x00</td>
<td>直進時、センサの状態は「0x00」です。これを直進状態と判断します。ハンドルをまっすぐにしなければいけないのは、疑いの余地がありません。問題はモータの PWM 値です。こちらは実際に走らせなければどのくらいのスピードになるのか何とも言えません。中心線が無くなったら曲がれるスピードにしなければいけません。とりあえず 40%にしておき、実際に走らせて微調整することにします。まとめると下記のようになります。
ハンドル:0 度
左モータ:40%　右モータ:40%</td>
</tr>
<tr>
<td> 
0x04
0x06
0x07
0x03</td>
<td>マイコンカーが左に寄ったときを考えています。中心から少しずつ左へずらしていくと左図のように 4 つの状態になりました。センサの状態はもっと左へ寄ることも考えられますが、右ハーフラインの後は直線しかないと分かっているので、これ以上センサ状態は増やさないでおきます。
動作は、左へ寄っているので右へハンドルを切ります。切り角が小さすぎるとずれが大きいときに戻りきれなくなり、大きすぎるとセンサが左右にばたばた振れてしまいます。ちょうど良い角度調整は難しいです。今回は、とりあえずどの状態も 8 度にします。まとめると下記のようになります。
ハンドル:8 度
左モータ:40%　右モータ:35%</td>
</tr>
</table>



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

ポイントは、中心線があるかどうかのチェックは 8 個のセンサすべてを使用することです。他は「MASK3_3」でマスクして中心の 2 個のセンサは使用しません。
プログラム化すると下記のようになります。

```
671 :       case 63:
672 :           /* Trace, lane change after left half line detection */
673 :           if( sensor_inp( &sensor0, MASK4_4) == 0x00 ) {
674 :               handle( -15 );
675 :               motor( 31 ,40 );
676 :               pattern = 64;
677 :               cnt1 = 0;
678 :               break;
679 :           }
680 :           switch( sensor_inp( &sensor0, MASK3_3) ) {
681 :               case 0x00:
682 :                   /* Center -> straight */
683 :                   handle( 0 );
684 :                   motor( 40 ,40 );
685 :                   break;
686 :               case 0x04:
687 :               case 0x06:
688 :               case 0x07:
689 :               case 0x03:
690 :                   /* Left of center -> turn to right */
691 :                   handle( 8 );
692 :                   motor( 40 ,35 );
693 :                   break;
694 :               case 0x20:
695 :               case 0x60:
696 :               case 0xe0:
697 :               case 0xc0:
698 :                   /* Right of center -> turn to left */
699 :                   handle( -8 );
700 :                   motor( 35 ,40 );
701 :                   break;
702 :               default:
703 :                   break;
704 :           }
705 :           break;
```

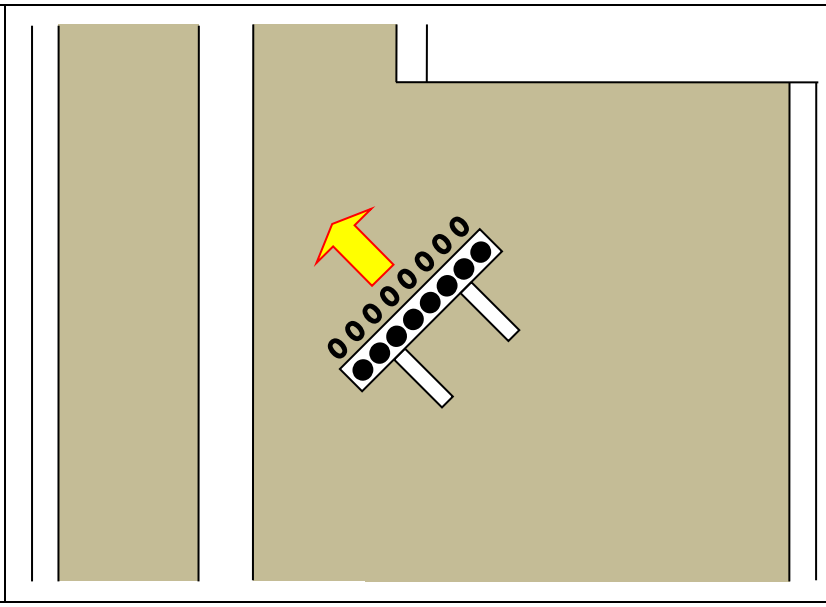
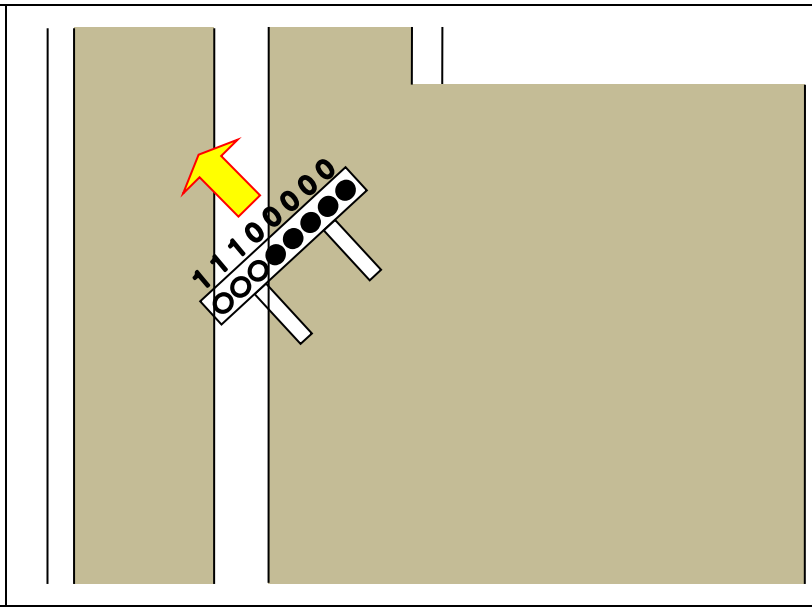
case を続けて書くと
0x04 または 0x06 または 0x07 または 0x03 のとき
という意味になります。

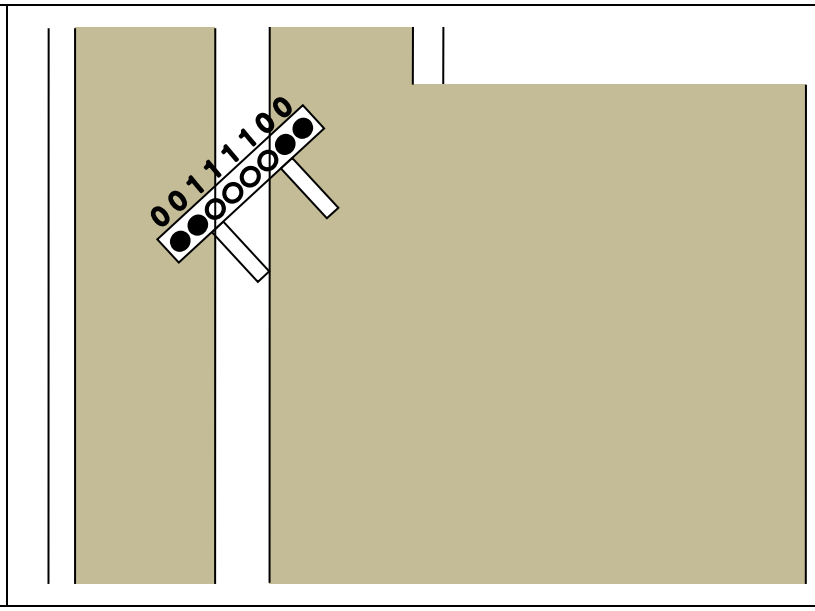
case を続けて書くと
0x20 または 0x60 または 0xe0 または 0xc0 のとき
という意味になります。

6.2.35 パターン 64:左レーンチェンジ終了のチェック

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

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

1		曲げながら進んできました。新しい中心線を見つける直前です。
2		中心線をセンサの左端で検出しました。 センサの状態は、 「1110 0000」です。

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

上記の考え方で、プログラムします。

```
707 :       case 64:
708 :           /* Left lane change end check */
709 :           if( sensor_inp( &sensor0, MASK4_4 ) == 0x3c ) {
710 :               led_out( 0x0 );
711 :               pattern = 11;
712 :               cnt1 = 0;
713 :           }
714 :           break;
```

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