

基板マイコンカー 製作キット Ver.2 プログラム解説マニュアル

- ・公開しているプログラムは、レーンチェンジには対応していません。
- ・公開しているプログラムは、基本的な考え方のみ記述しています。実際にコースを完走させるには、各自プログラムを改造して対応してください。

本マニュアルでプログラムの説明をしている基板	基板マイコンカー製作キット Ver.2
本基板の 対象マイコンボード	ミニマイコンカー製作キット Ver.2 のマイコンボード部分、 もしくは RMC-R8C35A マイコンボード

第 1.13 版

2015.04.20

株式会社日立ドキュメントソリューションズ

注意事項 (rev.6.0H)

著作権

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

禁止事項

ユーザーは以下の内容を行うことはできません。

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

転載、複製

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

責任の制限

本マニュアルに記載した情報は、正確を期すため、慎重に制作したのですが万一本マニュアルの記述誤りに起因する損害が生じた場合でも、株式会社日立ドキュメントソリューションズはその責任を負いません。

その他

- ・本マニュアルに記載の情報は本マニュアル発行時点のものであり、株式会社日立ドキュメントソリューションズは、予告なしに、本マニュアルに記載した情報または仕様を変更することがあります。製作に当たりましては、最新の内容を確認いただきますようお願いいたします。
- ・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. 基板マイコンカーVer.2 の仕様.....	3
1.1 概要.....	3
1.2 仕様.....	4
1.3 ブロック図.....	5
1.4 外観と寸法.....	6
2. RMC-R8C35A マイコンボード.....	7
2.1 外観.....	7
2.2 R8C/35A マイコンのポート表.....	8
2.3 R8C/35A マイコンで使用する内蔵周辺機能.....	9
3. 基板マイコンカーVer.2 の回路.....	10
3.1 モータドライブ回路.....	10
3.1.1 モータドライブ基板 TypeS Ver.4 との違い.....	10
3.1.2 N チャネル FET のみを使う利点.....	11
3.1.3 N チャネル FET のみを使うときの問題点.....	12
3.1.4 N チャネル FET のみを使うときの解決策.....	13
3.1.5 IR2302 の動作原理.....	13
3.1.6 実際の回路.....	15
3.2 ロータリエンコーダ信号入力回路.....	17
3.3 ステアリング角度検出用ポテンショメータ信号入力回路.....	17
4. ワークスペース「rmc_frame_ver2」.....	18
4.1 ワークスペースのインストール.....	18
4.2 プロジェクト.....	20
4.3 プロジェクトの構成.....	21
5. プログラムの解説.....	22
5.1 プログラムリスト「rmc_frame_ver2.c」.....	22
5.2 プログラムの解説.....	33
5.2.1 シンボル定義.....	33
5.2.2 変数の定義.....	34
5.2.3 内輪差値計算用の配列追加.....	36
5.2.4 init 関数のクロックの選択.....	37
5.2.5 ポートの入出力設定.....	38
5.2.6 A/D コンバータの設定.....	40
5.2.7 タイマ RB の設定.....	42
5.2.8 タイマ RC の設定.....	44
5.2.9 タイマ RD0 の設定.....	54
5.2.10 タイマ RD1 の設定.....	65
5.2.11 タイマ RB の 1ms ごとの割り込みプログラム.....	68
5.2.12 アナログセンサ基板 TypeS Ver.2 のデジタルセンサ値読み込み.....	70
5.2.13 アナログセンサ基板 TypeS Ver.2 の中心デジタルセンサ読み込み.....	71
5.2.14 アナログセンサ基板 TypeS Ver.2 のスタートバー検出センサ読み込み.....	72
5.2.15 RMC-R8C35A マイコンボード上のディップスイッチ値読み込み.....	73
5.2.16 基板マイコンカーVer.2 上のプッシュスイッチ値読み込み.....	74

5.2.17 RMC-R8C35A マイコンボード上の LED 制御	75
5.2.18 後輪の速度制御(ディップスイッチによる PWM 減速あり)	76
5.2.19 後輪の速度制御 2(ディップスイッチには関係しない motor 関数)	78
5.2.20 前輪の速度制御(ディップスイッチによる PWM 減速あり)	80
5.2.21 前輪の速度制御 2(ディップスイッチには関係しない motor 関数)	81
5.2.22 後モータ停止動作(ブレーキ、フリー)設定	82
5.2.23 前モータ停止動作(ブレーキ、フリー)設定	83
5.2.24 ステアリングモータの PWM 設定	83
5.2.25 クロスラインの検出処理	85
5.2.26 ステアリングモータ角度の取得	86
5.2.27 アナログセンサ値の取得	87
5.2.28 ステアリングモータ制御	89
5.2.29 内輪 PWM 値計算	92
5.2.30 main 関数-初期化	93
5.2.31 パターン処理	94
5.2.32 パターン 0:スタート待ち	94
5.2.33 パターン 1:スタートバー開待ち	95
5.2.34 パターン 11:通常トレース	96
5.2.35 パターン 21:クロスライン検出処理	97
5.2.36 パターン 22:クロスライン後のトレース、直角検出処理	98
5.2.37 パターン 31:右クランク処理	100
5.2.38 パターン 32:右クランク処理後、少し時間がたつまで待つ	101
5.2.39 パターン 41:左クランク処理	101
5.2.40 パターン 42:左クランク処理後、少し時間がたつまで待つ	101
6. 4 輪の回転数計算	102
6.1 センターピボット方式 4 輪の回転数計算	102
6.2 アッカーマン方式 4 輪の回転数計算	104
7. ステアリングモータの角度指定	108
7.1 PD 制御	108
7.2 プログラム	108
7.2.1 グローバル変数の追加	108
7.2.2 関数の追加	109
7.2.3 割り込みプログラムの追加	109
7.2.4 使い方	110
8. プログラムの調整ポイント	111
9. 参考文献	112

1. 基板マイコンカーVer.2 の仕様

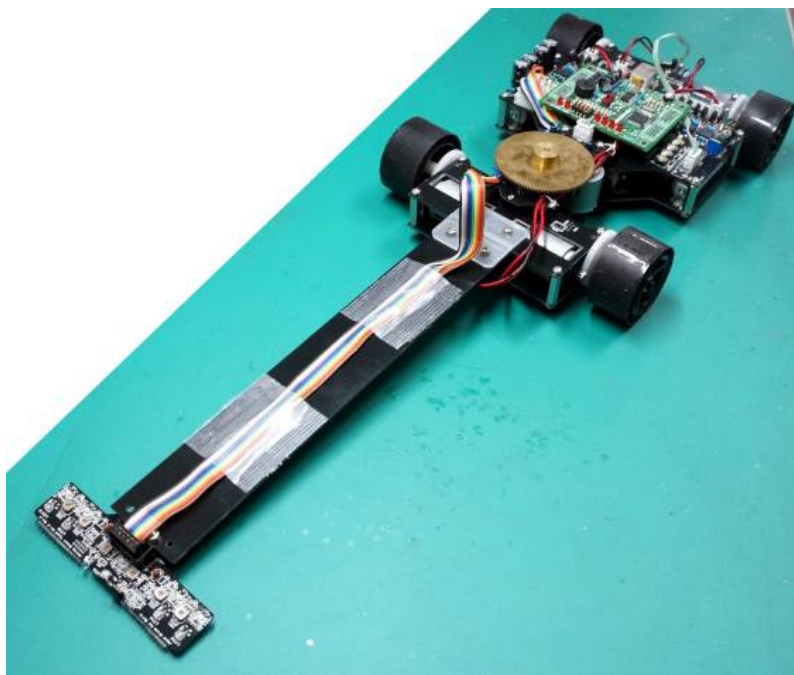
※本マニュアルでは、「基板マイコンカー製作キット Ver.2」を、「基板マイコンカーVer.2」と略します。

※モータドライブ基板 TypeS Ver.4 のプログラムと互換にするために、「ステアリングモータ」と「サーボモータ」の用語が混在しています。本マニュアルでは、「ステアリングモータ=サーボモータ」です。

1.1 概要

基板マイコンカー製作キット Ver.2 は、ミニマイコンカー製作キット Ver.2 (RMC-R8C35A) を使用し、ジャパンマイコンカーラリーの Advanced Class に出場できる車体にするためのキットです。

本マニュアルは、基板マイコンカー製作キット Ver.2 のプログラムについて解説します。



それぞれの基板、機器の詳しい説明は下表のマニュアルを参照してください。

基板、機器名	キット、製作についてのマニュアル	プログラムについてのマニュアル
基板マイコンカー製作キット Ver.2	基板マイコンカー製作キット Ver.2 製作マニュアル	本マニュアル
ミニマイコンカー製作キット Ver.2 のマイコンボード部分、もしくは RMC-R8C35A マイコンボード	ミニマイコンカー製作キット Ver.2 組み立てマニュアル https://www2.himdx.net/mcr/product/mini_micom_car_ver2.html	ミニマイコンカー製作キット Ver.2 マイコン実習マニュアル R8C/35A 版 (URL は左と同じ)
アナログセンサ基板 TypeS Ver.2	アナログセンサ基板 TypeS Ver.2 製作マニュアル(R8C/38A 版)	モータドライブ基板 TypeS Ver.4 アナログセンサ基板 TypeS Ver.2 プログラム解説マニュアル
ロータリエンコーダ	ロータリエンコーダ Ver.2 製作マニュアル(R8C/38A 版)	ロータリエンコーダ kit12.38a プログラム解説マニュアル (R8C/38A 版)

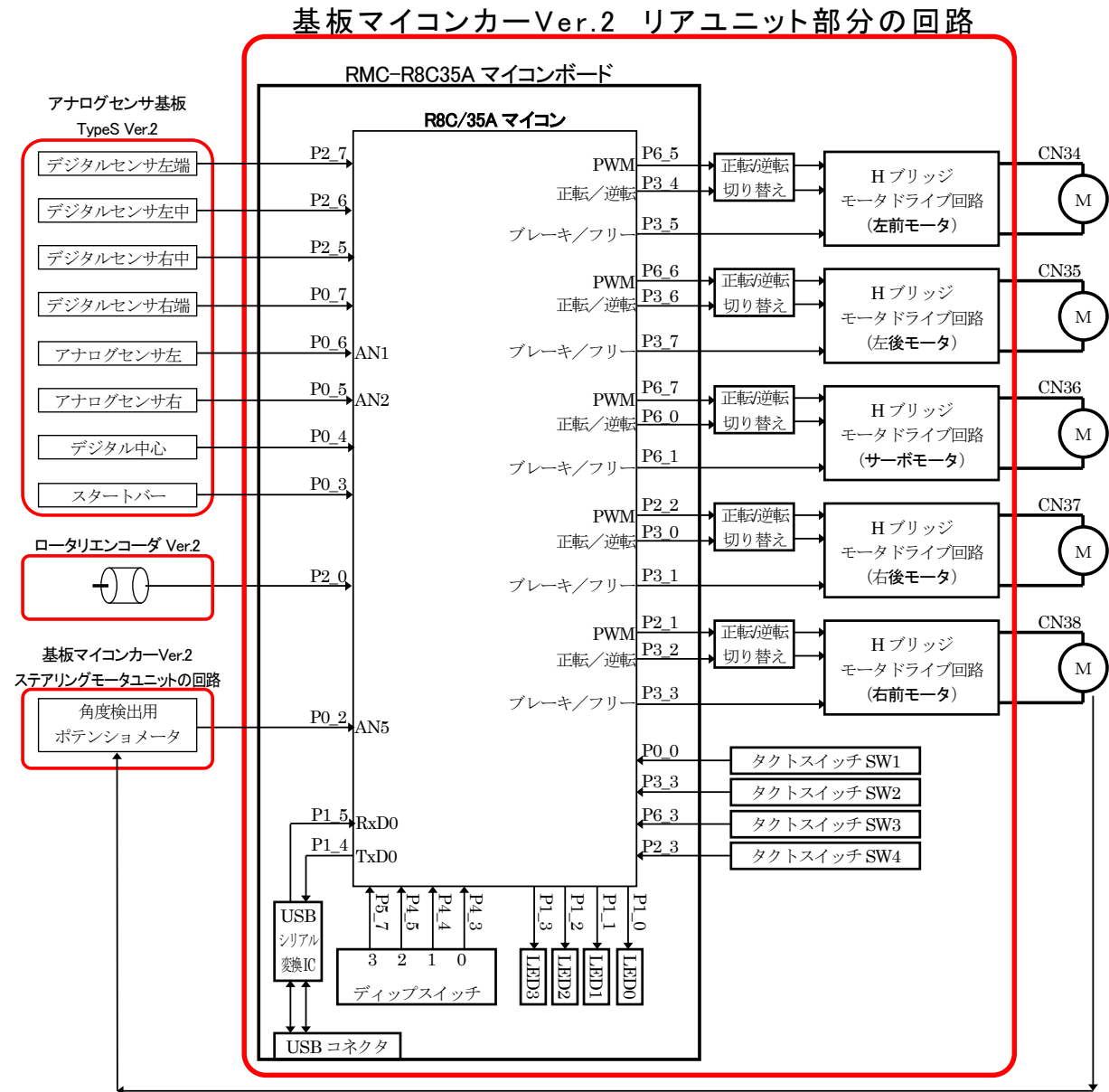
※URL が記載していないマニュアルは、「<http://www.mcr.gr.jp/tech/download/main11.html>」からダウンロードできます。

1.2 仕様

内容	詳細
全長	510mm
全幅	170mm
全高	70mm
重量	960g ※単三 2 次電池(2000mAh)8 本含んだときの参考重量
ホイールベース	180mm
トレッド F/R	145mm/145mm ※左右のタイヤの中心
ギア比	駆動 8:1 (8T/64T) ステアリング 58.7:1 (8T/110T、15T/64T)
電源	単 3 型充電電池×8 本
マイコンボード	マイコンカーラリー販売サイト「ミニマイコンカー製作キット Ver.2」のマイコンボード部分、もしくは同「RMC-R8C35A マイコンボード」を使用(別売り)
モータドライバ	ローサイド、ハイサイド Nch MOS FET Hブリッジ×5 個 タイマ RC とタイマ RD0 の PWM により、モータを 5 個駆動可能 PWM 周期: 1kHz モータ動作: 正転、逆転、短絡ブレーキ、フリー ※IR2302 の仕様で、100%回転はできません。今回は 99%を上限にしています
速度制御	マイコンカーラリー販売サイト「M-S145:ロータリエンコーダセット Ver.2(2台1セット)」を使用(別売り) パルス数: 72 パルス/回転
コース検出センサ	マイコンカーラリー販売サイト「M-S199:アナログセンサ基板 TypeS Ver.2」を使用(別売り) アナログセンサ×2 個、デジタルセンサ×5 個、スタートバー検出センサ×1 個 を搭載
ステアリング角度	約±40 度(実測)
ステアリング角度検出用ポテンショメータの分解能	実測で A/D 値±127 1 度当たりの A/D 値 $127 / 40 = \text{約 } 3.18$
その他	タクトスイッチ×4 個 ※マイコンボード上の SW2 はリセットスイッチです。

1.3 ブロック図

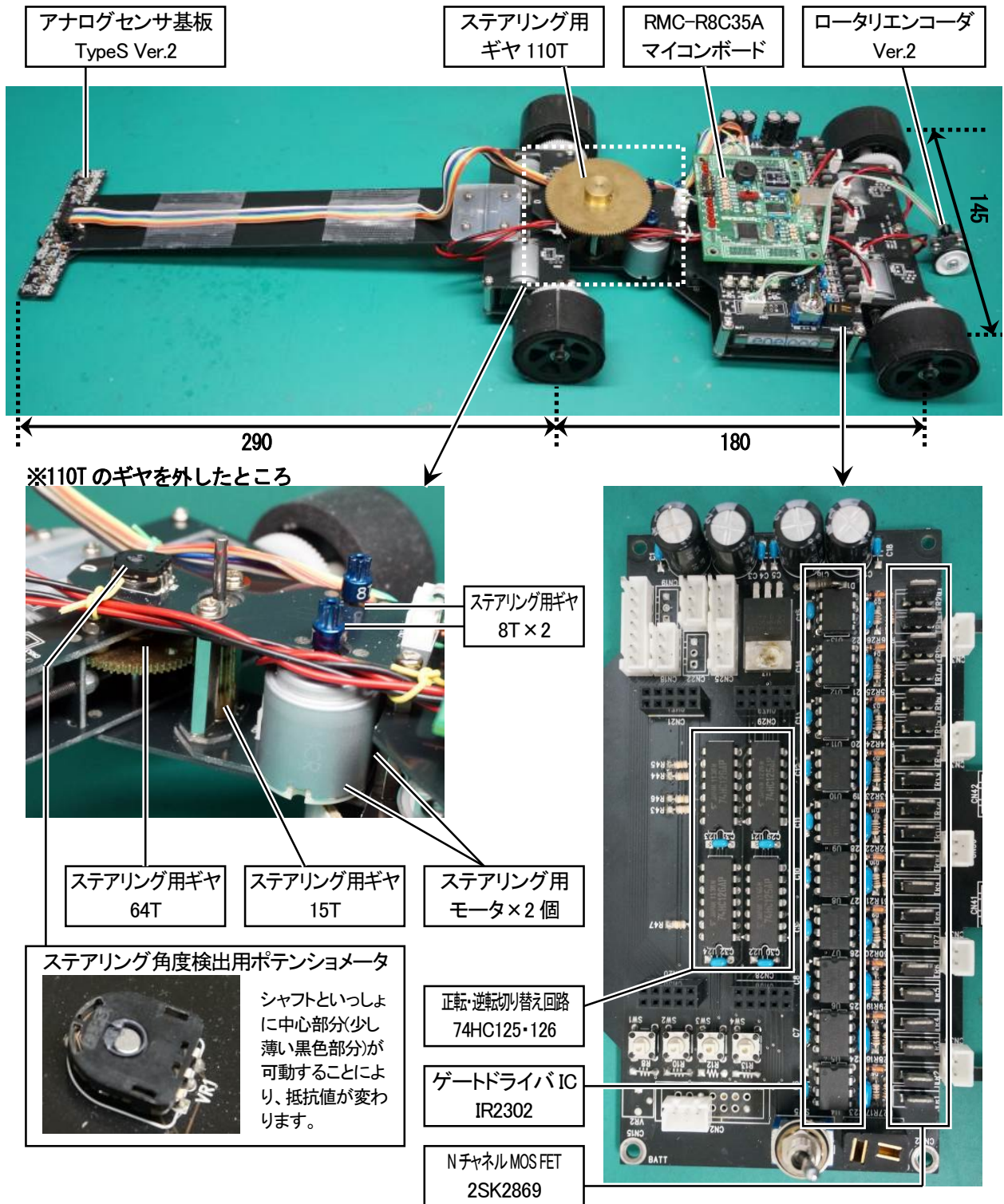
基板マイコンカーVer.2 のブロック図を下記に示します。



1. 基板マイコンカーVer.2 の仕様

1.4 外観と寸法

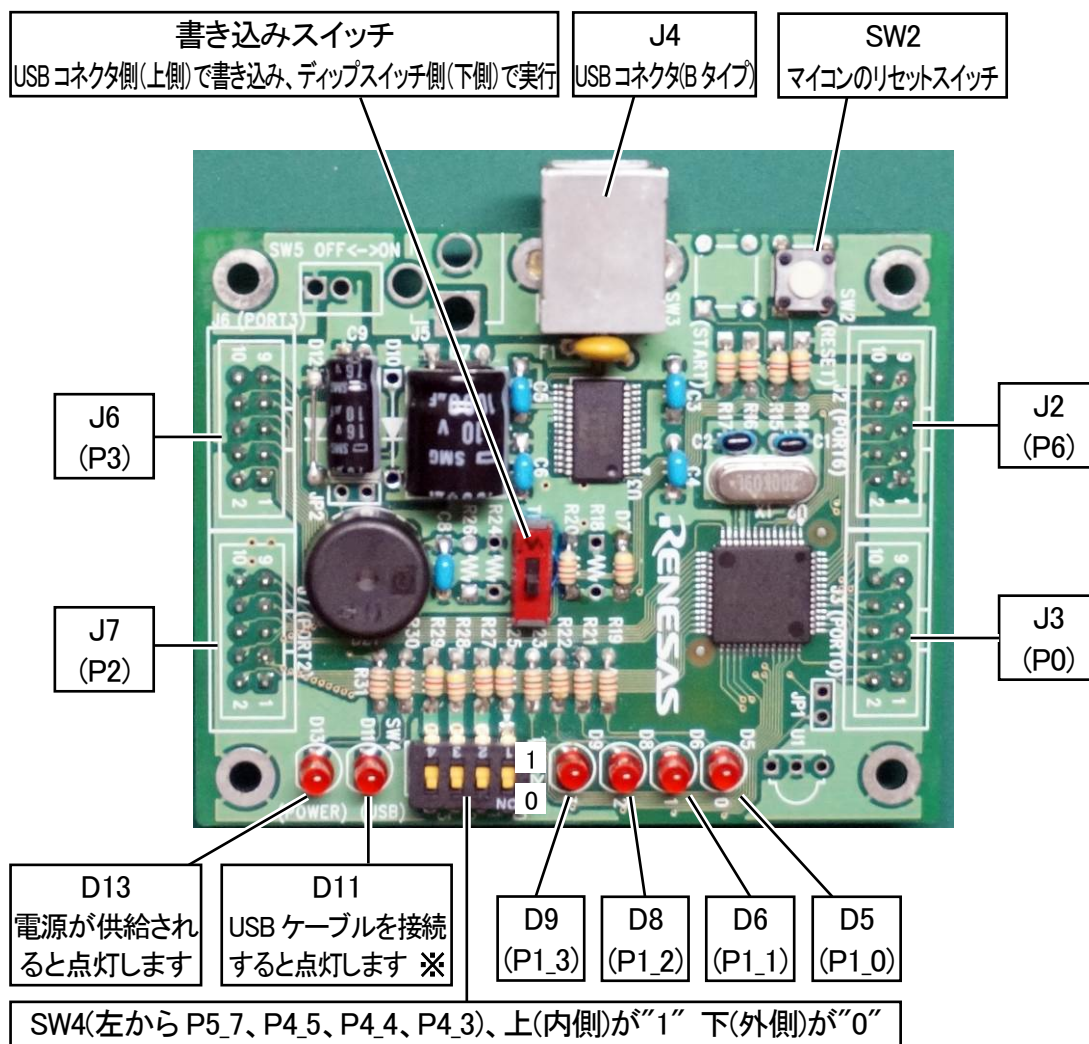
基板マイコンカーVer.2 の外観と寸法を、下写真に示します。



2. RMC-R8C35A マイコンボード

2.1 外観

今回の RMC-R8C35A マイコンボードは、基板マイコンカー Ver.2 用として部品の付け替えをしています。付け替えの内容は、「基板マイコンカー製作キット Ver.2 製作マニュアル」を参照してください。



※D11のLEDは、USBケーブルを接続すると点灯しますが、基板マイコンカー Ver.2 や RMC-R8C35A マイコンボードには電源は供給されません。書き込みや Tera Term で通信をするときは、基板マイコンカー Ver.2 の電源を入れてください(D13のLEDを点灯させてください)。

2.2 R8C/35A マイコンのポート表

コネクタ	pin	Port	bit	端子名	接続先
J3	2	P0	7	P0_7/AN0/DA1 (/TRCIOC)	アナログセンサー基板TypeS Ver. 2 デジタルセンサ右端
	3	P0	6	P0_6/AN1/DA0 (/TRCIOD)	アナログセンサー基板TypeS Ver. 2 アナログセンサ左
	4	P0	5	P0_5/AN2 (/TRCIOB)	アナログセンサー基板TypeS Ver. 2 アナログセンサ右
	5	P0	4	P0_4/AN3/TRE0 (/TRCIOB)	アナログセンサー基板TypeS Ver. 2 デジタルセンサ中心
	6	P0	3	P0_3/AN4 (/CLK1/TRCIOB)	アナログセンサー基板TypeS Ver. 2 スタートバー検出センサ
	7	P0	2	P0_2/AN5 (/RXD1/TRCIOA/TRCTR)	ポテンシオメータ (ステアリング角度検出用)
	8	P0	1	P0_1/AN6 (/TXD1/TRCIOA/TRCTR)	
	9	P0	0	P0_0/AN7 (/TRCIOA/TRCTR)	SW2
			P1	7	P1_7/IVCMP1/INT1 (/TRAIO)
		P1	6	P1_6/LVCOUT2/IVREF1 (/CLK0)	(マイコンボードからは出力されていない端子のため使用できません)
		P1	5	P1_5 (/INT1/RXD0/TRAIO)	RxD0
		P1	4	P1_4 (/TXD0/TRCCLK)	TxD0
		P1	3	P1_3/AN11/LVCOUT1/K13/TRB0 (/TRCIOC)	LED3
		P1	2	P1_2/AN10/LVREF/K12 (/TRCIOB)	LED2
		P1	1	P1_1/AN9/LVCMP2/KI1 (/TRCIOA/TRCTR)	LED1
		P1	0	P1_0/AN8/LVCMP1/KI0 (/TRCIOD)	LED0
J7	2	P2	7	P2_7 (/TRDIOD1)	アナログセンサー基板TypeS Ver. 2 デジタルセンサ左端
	3	P2	6	P2_6 (/TRDIOC1)	アナログセンサー基板TypeS Ver. 2 デジタルセンサ左中
	4	P2	5	P2_5 (/TRDIOB1)	アナログセンサー基板TypeS Ver. 2 デジタルセンサ右中
	5	P2	4	P2_4 (/TRDIOA1)	
	6	P2	3	P2_3 (/TRDIOD0)	SW4
	7	P2	2	P2_2 (/TRCIOD/TRDIOB0)	右後モータPWM
	8	P2	1	P2_1 (/TRCIOC/TRDIOC0)	右前モータPWM
	9	P2	0	P2_0 (/INT1/TRCIOB/TRDIOA0/TRDCLK)	ロータリエンコーダパルスカウント
	J6	2	P3	7	P3_7/SDA/SS0/TRA0 (/RXD2/SCL2/TXD2/SDA2)
3		P3	6	P3_6 (/INT1)	左後モータ 方向 1:正転 0:逆転
4		P3	5	P3_5/SCL/SSCK (/CLK2/TRCIOD)	左前モータ 0:フリー 1:ブレーキ
5		P3	4	P3_4/IVREF3/SSI (/RXD2/SCL2/TXD2/SDA2/TRCIOC)	左前モータ 方向 1:正転 0:逆転
6		P3	3	P3_3/IVCMP3/INT3/SCS (/CTS2/RTS2/TRCCLK)	右前モータ 0:フリー 1:ブレーキ
7		P3	2	P3_2 (/INT1/INT2/TRAIO)	右前モータ 方向 1:正転 0:逆転
8		P3	1	P3_1 (/TRB0)	右後モータ 0:フリー 1:ブレーキ
9		P3	0	P3_0 (/TRA0)	右後モータ 方向 1:正転 0:逆転
			P4	7	P4_7/XOUT
		P4	6	P4_6/XIN	クリスタル(XIN) (20MHz)
		P4	5	P4_5/ADTRG/INT0 (/RXD2/SCL2)	マイコンボード上のディップスイッチ2(SW4)
		P4	4	P4_4 (/XCOUT)	マイコンボード上のディップスイッチ1(SW4)
		P4	3	P4_3 (/XCIN)	マイコンボード上のディップスイッチ0(SW4)
		P4	2	P4_2/VREF	Vccに接続(A/D変換の基準電圧)
		P5	7	P5_7	マイコンボード上のディップスイッチ3(SW4)
J2		P5	6	P5_6 (/TRA0)	(マイコンボードからは出力されていない端子のため使用できません)
	2	P6	7	P6_7 (/INT3/TRCIOD)	ステアリングモータPWM
	3	P6	6	P6_6/INT2 (/TXD2/SDA2/TRCIOC)	左後モータPWM
	4	P6	5	P6_5/INT4 (/CLK1/CLK2/TRCIOB)	左前モータPWM
	5	P6	4	P6_4 (/RXD1)	SW1
	6	P6	3	P6_3 (/TXD1)	SW3
	7	P6	2	P6_2 (/CLK1)	
	8	P6	1	P6_1	ステアリングモータ 0:フリー 1:ブレーキ
	9	P6	0	P6_0 (/TRE0)	ステアリングモータ 方向 1:正転 0:逆転

※コネクタは、RMC-R8C35A マイコンボードの番号です。

2.3 R8C/35A マイコンで使用する内蔵周辺機能

機能	詳細
A/D コンバータ	A/Dコンバータを、繰り返し掃引モードで使います。繰り返し掃引モードは、複数の端子を繰り返し A/D 変換するモードです。今回、下記の 3 つの端子の電圧を読み込みます。 ※今回のプログラムでは A/D 変換を P0_0~P0_7 の 8 端子分、実行していますが、A/D 変換値を読み込んでいるのが、AD1、AD2、AD5 の 3 つの端子です。
	P0_6(AD1) アナログセンサ左の電圧を読み込みます。
	P0_5(AD2) アナログセンサ右の電圧を読み込みます。
	P0_2(AD5) ステアリング角度検出用ポテンショメータの電圧を読み込みます。
タイマ RA	未使用です。
タイマ RB	インターバルタイマとして使用して、1ms ごとに割り込みを発生させます。
タイマ RC	PWM モードとして使います。タイマ RC は PWM 信号を 3 本出力することができます。今回、タイマ RC でスピード制御しているモータを、下記に示します。 ①左後モータ ②左前モータ ③ステアリングモータ
タイマ RD0	PWM モードとして使います。タイマ RD0 は PWM 信号を 3 本出力することができます。今回、タイマ RD0 でスピード制御しているモータを、下記に示します。 ①右後モータ ②右前モータ
タイマ RD1	ロータリエンコーダ Ver.2 のパルス入力として使います。
タイマ RE	未使用です。

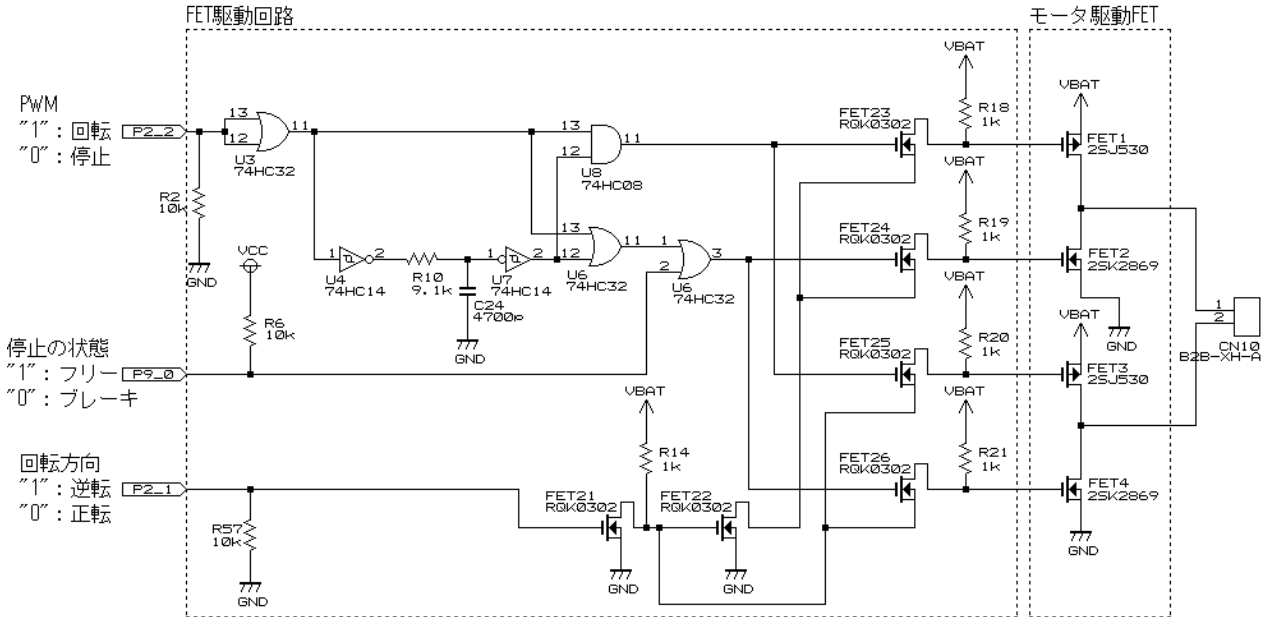
3. 基板マイコンカーVer.2 の回路

3.1 モータドライブ回路

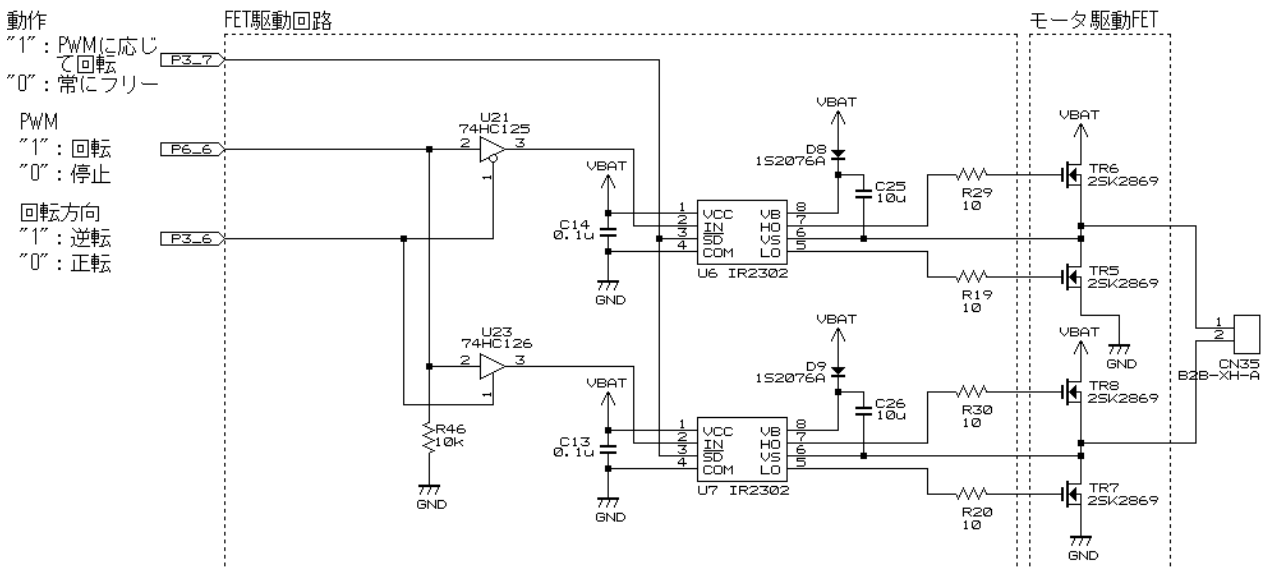
3.1.1 モータドライブ基板 TypeS Ver.4 との違い

モータドライブ基板 TypeS Ver.4 と基板マイコンカーVer.2 の左後モータを駆動する回路図を下記に示します。

●モータドライブ基板TypeS Ver.4の左後モータ



●基板マイコンカーVer.2の左後モータ



※モータドライブ基板 TypeS Ver.4 の回路の動作原理は、「モータドライブ基板 TypeS Ver.4 アナログセンサ基板 TypeS Ver.2 プログラム解説マニュアル(R8C/38A 版)」を参照してください。

マニュアルは、「ダウンロードページ(<http://mcr.gr.jp/tech/download/main01.html>)」

「R8C/38A マイコン(RY_R8C38 ボード)に関する資料」からダウンロードできます。

それぞれの回路の特徴を下記に示します(ゴシック体太字が良い方)。

内容	モータドライブ基板 TypeS Ver.4	基板マイコンカーVer.2
FET 駆動回路	ロジック IC (OR 回路、AND 回路、シュミットトリガの NOT 回路) と遅延回路、FET(RQK0302)による回路	ロジック IC (74HC125、126)とダイオード、コンデンサ(10 μ F)、FET を駆動する専用 IC(IR2302)による回路
モータ駆動 FET	P チャネル FET(2SJ530)を 2 個使用、N チャネル FET(2SK2869)を 2 個使用	N チャネル FET(2SK2869)を 4 個使用
部品点数	多い	少ない(専用 IC 内に機能が含まれている)
部品が壊れたとき	汎用の部品を使うので、故障したときに同等の部品が手元にある場合がある、またはすぐに購入できる	FET を駆動する専用 IC(IR2302)を使うので、手に入りづらい
フリー動作	「動作<->ブレーキ」か「動作<->フリー」を選択可能	「動作<->ブレーキ」か「フリー(のみ)」を選択可能

今回の回路の最大の特徴は、N チャネル FET のみを使用して、モータを駆動していることです。

3.1.2 N チャネル FET のみを使う利点

FET は P チャネル型と N チャネル型があります。これらをペアで使いたいとき、「**コンプリメンタリ・ペア**」の FET を使います。「**コンプリメンタリ(相補的)・ペア**」とは、それぞれで極性が反転している他は特性の似た P チャネル型と N チャネル型の FET の組のことです。

※出典: ウィキペディア <http://ja.wikipedia.org/wiki/バイポーラトランジスタ>

モータドライブ基板 TypeS Ver.4 で使用している 2SJ530 と 2SK2869 はコンプリメンタリです。コンプリメンタリかどうかは、部品メーカーのホームページやデータシートなどに記載されています。

基板マイコンカーVer.2 は、N チャネル FET だけでモータを駆動します。なぜでしょうか？ 2SJ530 と 2SK2869 のデータの一部を下記に示します(ゴシック体太字の方が特性が良い)。

内容	2SJ530 (P チャネル)	2SK2869 (N チャネル)
ドレイン・ソースオン抵抗 条件: $I_D=10V$ 、 $V_{GS}=10V$	最大値: 0.10 Ω	最大値: 0.045Ω
ドレイン電流 I_D	-15A	20A
ターン・オン遅延時間 + 上昇時間 ※FET が OFF の状態から ON になるまでの時間	標準値: 87ns	標準値: 120ns
ターン・オフ遅延時間 + 下降時間 ※FET が ON の状態から OFF になるまでの時間	標準値: 200ns	標準値: 225ns
値段	コンプリメンタリの場合、一般的に P チャネルが高い	コンプリメンタリの場合、一般的に N チャネルが安い

N チャネル型 FET の 2SK2869 のみで駆動した方が、ON 抵抗が低く、多くの電流を流すことができます。また、部品点数を少なくすることもできます。2 種類 2 個ずつ揃えるより、1 種類 4 個を揃えた方が効率的で、数も多くなるので単価が安くなるかもしれません。

しかし、モータドライブ基板 TypeS Ver.4 ではそうしていません。なぜでしょうか。

3. 基板マイコンカーVer.2 の回路

3.1.3 N チャンネル FET のみを使うときの問題点

まず、モータドライブ基板 TypeS Ver.4 の P チャンネル FET(2SJ530)、N チャンネル FET(2SK2869)を使った回路の動作原理を説明します(VBAT=10V として説明します)。

<p>FET はソース(S)を 0V としたとき、ゲート(G)に加える電圧によって、ソース(S)-ドレイン(D)間が ON するか OFF するか決まります。</p>	<p>0V を出力したいとき、それぞれの FET のゲート(G)には下記の電圧を入力します。</p> <ul style="list-style-type: none"> ●P チャンネル FET(2SJ530)のゲート(G)に 10V を入力すると $V_{GS}=0V$ となりソース(S)-ドレイン(D)間は OFF になります。 ●N チャンネル FET(2SK2869)のゲート(G)に 10V を入力すると $V_{GS}=10V$ となりソース(S)-ドレイン(D)間は ON になります。 	<p>10V を出力したいとき、それぞれの FET のゲート(G)には下記の電圧を入力します。</p> <ul style="list-style-type: none"> ●P チャンネル FET(2SJ530)のゲート(G)に 0V を入力すると $V_{GS}=-10V$ となりソース(S)-ドレイン(D)間は ON になります。 ●N チャンネル FET(2SK2869)のゲート(G)に 0V を入力すると $V_{GS}=0V$ となりソース(S)-ドレイン(D)間は OFF になります。

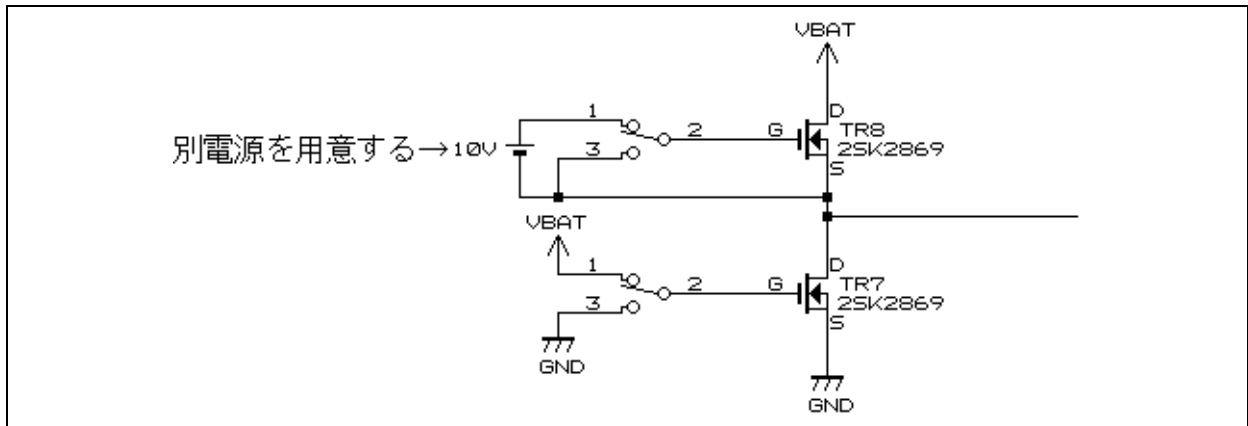
次に、基板マイコンカーVer.2 の N チャンネル FET(2SK2869)を使った回路の動作原理を説明します。

<p>FET はソース(S)を 0V としたとき、ゲート(G)に加える電圧によって、ソース(S)-ドレイン(D)間が ON するか OFF するか決まります。</p>	<p>0V を出力したいとき、それぞれの FET のゲート(G)には下記の電圧を入力します。</p> <ul style="list-style-type: none"> ●上側の N チャンネル FET(2SK2869)のゲート(G)に 0V を入力すると $V_{GS}=0V$ となりソース(S)-ドレイン(D)間は OFF になります(ただし、右欄の説明を参照)。 ●下側の N チャンネル FET(2SK2869)のゲート(G)に 10V を入力すると $V_{GS}=10V$ となりソース(S)-ドレイン(D)間は ON になります。 	<p>10V を出力したいとき、それぞれの FET のゲート(G)には下記の電圧を入力します。</p> <ul style="list-style-type: none"> ●上側の N チャンネル FET のソース(S)には電圧が入力されていないため、ゲート(G)に電圧を加えても V_{GS} に電圧は加えられません。よって、この回路では動作しません。左図の 0V を出力したいときも、下側 FET が ON になる前は同様なので、0V 出力もこの回路では動作しません。

このように、N チャンネル FET(2SK2869)だけを使った回路では、うまく動作しません。

3.1.4 Nチャネル FET のみを使うときの解決策

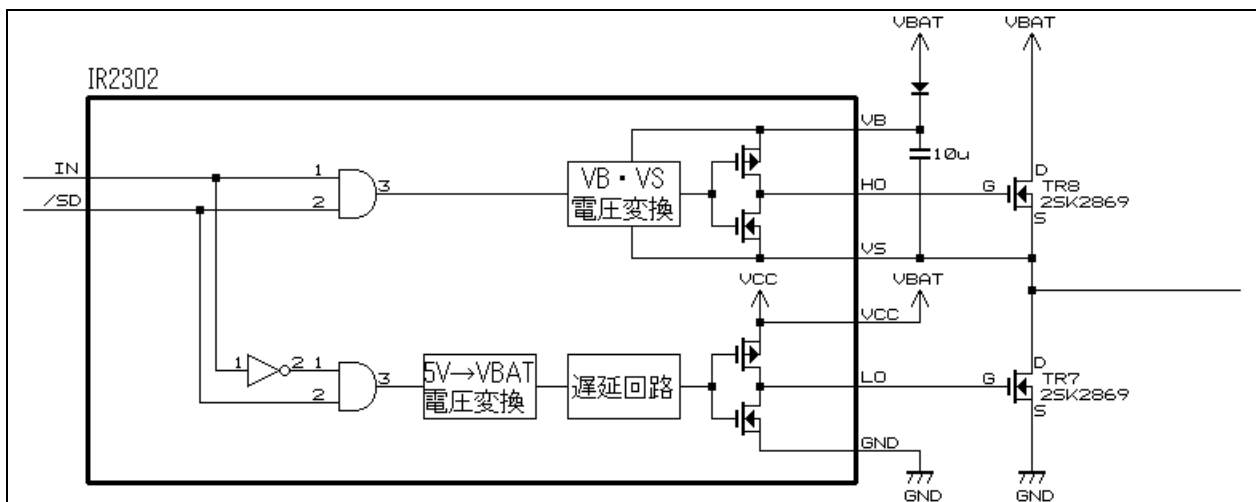
そこで、上側 FET を動作させる、専用の別電源を用意します。下記に、上側 FET 専用の 10V 電源を用意した様子を示します。



当然、別電源を用意するのは大変ですし、マイコンカーのルールにも違反しています。別電源を用意せずに 1 つの電源のみで実現するのが「IR2302」という、FET を駆動する専用 IC です。

3.1.5 IR2302 の動作原理

IR2302 の動作原理を下記に示します。上側 FET のゲートに加える電圧は、IC 内部の「VB・VS 電圧変換回路」で 5V の信号 (IN や /SD 信号) を VB、VS の電圧に変換します。

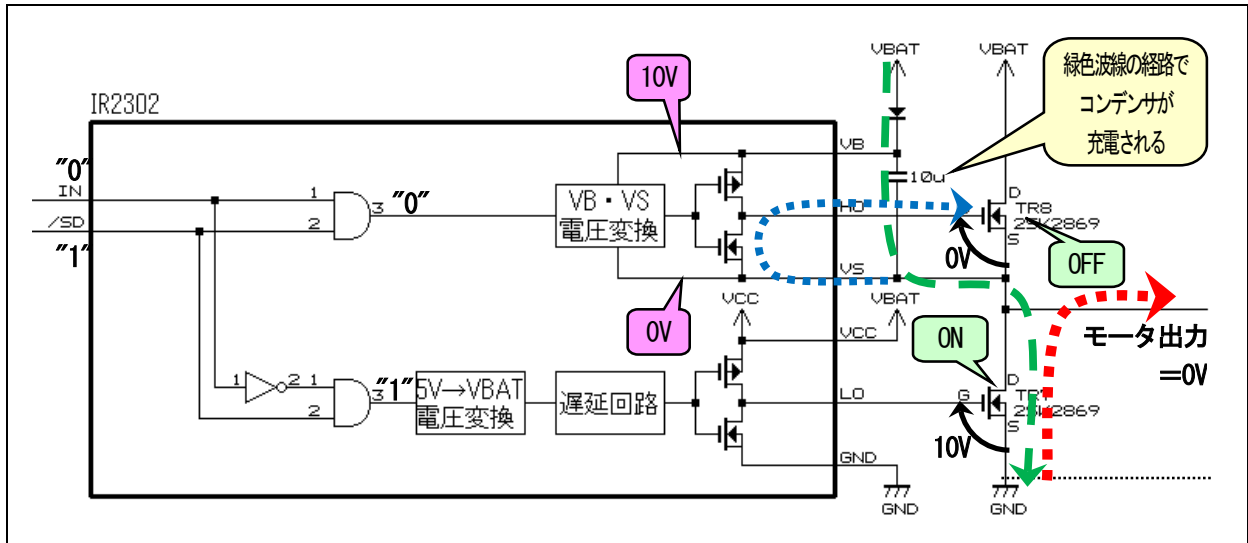


VCC	内部の電源回路と下側 FET の電源です。今回の IR2302 は 25V まで加えることができます。
LO	下側 FET のゲート(G)に接続します。
GND	内部の電源回路と下側 FET の GND です。
VB	上側 FET の電源です。
HO	上側 FET のゲート(G)に接続します。
VS	下側 FET の GND です。
IN	"1":HO=High、LO=Low "0":HO=Low、LO=High ※3V 以上で"1"とみなします。
/SD	"1":IN 信号に従って動作 "0":フリー動作(HO=Low、LO=Low) ※3V 以上で"1"とみなします。

3. 基板マイコンカーVer.2 の回路

IN="0"にしたときの様子を下記に示します。

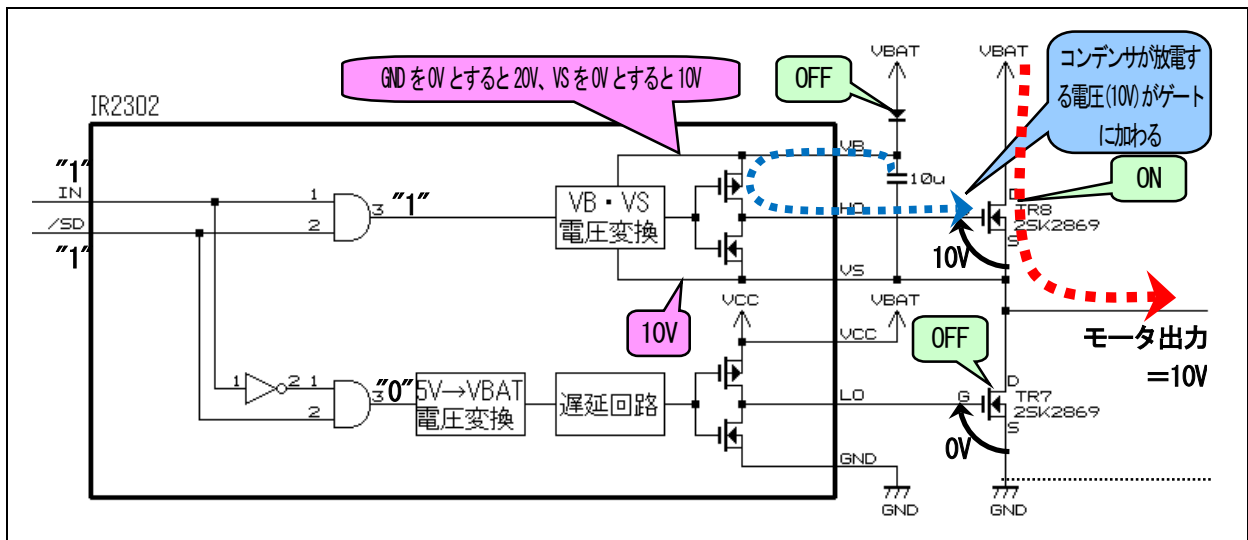
下側 FET が ON になり、モータ出力は 0V になります。このとき、VS=0V、VB=VBAT の電圧となり、10 μ F のコンデンサが充電されます(下記)。



IN="1"にしたときの様子を下記に示します。

GND を 0V とすると VS は 10V、VB は 20V になります。このとき、ダイオードのアノード側(VBAT)は 10V なので、ダイオードは OFF になります。

GND を 0V とすると VB は 20V ですが、VS=0V としたとき VB は 10V になります。上側 FET のゲートには 10V が入力され ON になり、モータへの出力は 10V になります。



※100%出力について

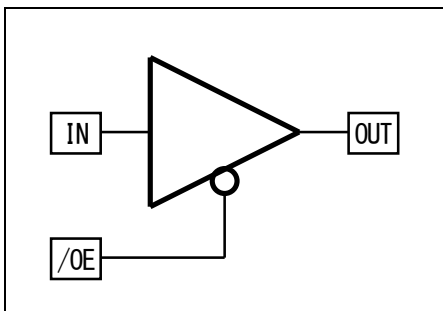
常に IN を "1" にすると (PWM=100% にすると)、コンデンサを充電する時間がありません。コンデンサの電圧が落ちると、上側 FET が OFF してしまい、モータ出力が 0V になってしまいます。よって、PWM を 100% にすることはできません。

今回のプログラムでは、99%以上は99%になるよう作られています。1%はOFFになるので、このときコンデンサが充電されます。

3.1.6 実際の回路

実際の回路では、74HC125 と 74HC126 を使用しています。この IC の動作を下記に示します。

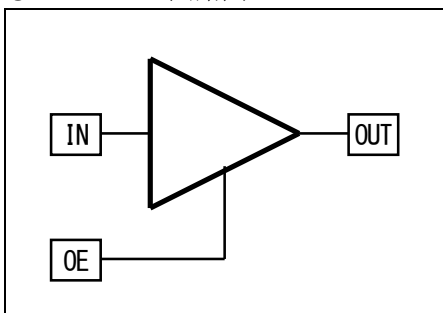
●74HC125 の回路図



●入力と出力の関係

IN	/OE	OUT
0	0	0 (/OE が"0"のとき OUT=IN)
1	0	1 (/OE が"0"のとき OUT=IN)
0or1	1	ハイインピーダンス (/OE が"1"のとき IN に関わらず無接続状態)

●74HC126 の回路図



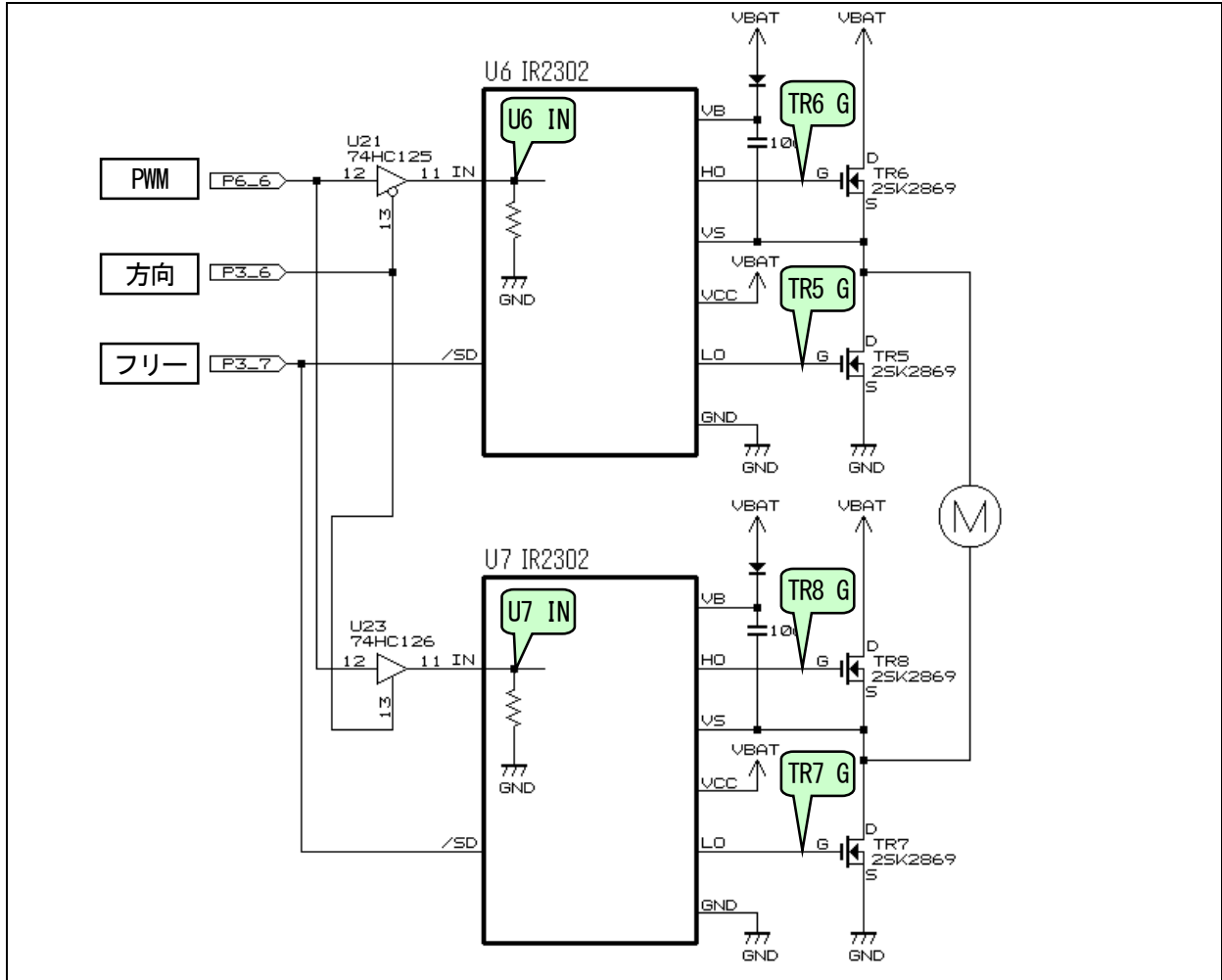
●入力と出力の関係

IN	OE	OUT
0	1	0 (/OE が"1"のとき OUT=IN)
1	1	1 (/OE が"1"のとき OUT=IN)
0or1	0	ハイインピーダンス (OE が"0"のとき IN に関わらず無接続状態)

74HC125 と 74HC126 の違いは、「OE」の論理です。74HC125 は「/OE」が"**0**"のときに「OUT=IN」の動作、74HC126 は「OE」が"**1**"のときに「OUT=IN」の動作になります。

3. 基板マイコンカーVer.2 の回路

左後モータを駆動する実際の回路を下記に示します。IR2302 は、2 組で 1 個のモータを制御します。



IN	方向	フリー	U6 IN	U7 IN	TR6 G	TR5 G	TR8 G	TR7 G	モータ動作
0 (0V)	0 (0V)	1 (5V)	0 (0V)	0 (0V)	0V (OFF)	10V (ON)	0V (OFF)	10V (ON)	ブレーキ 両端子 GND
1 (5V)	1 (5V)		1 (5V)	0 (0V)	10V (ON)	0V (OFF)	0V (OFF)	10V (ON)	正転 上側=10V 下側=0V
0 (0V)	1 (5V)	1 (5V)	0 (0V)	0 (0V)	0V (OFF)	10V (ON)	0V (OFF)	10V (ON)	ブレーキ 両端子 GND
1 (5V)			0 (0V)	1 (5V)	0V (OFF)	10V (ON)	10V (ON)	0V (OFF)	逆転 上側=0V 下側=10V
x	x	0 (0V)	x	x	0V (OFF)	0V (OFF)	0V (OFF)	0V (OFF)	フリー 無接続状態

x = "0"でも"1"でも動作に関係ない

例えば、方向が"0"のとき、U23 の出力は無接続状態になりますが、U7 の IN 端子にはプルダウン抵抗が内蔵されているため、"0"になります。よって、U7 の出力は常に 0V になります。このときのモータの動作は、IN 端子が"0"なら U6 は 0V (モータはブレーキ状態)、"1"なら 10V (モータは正転) となります。

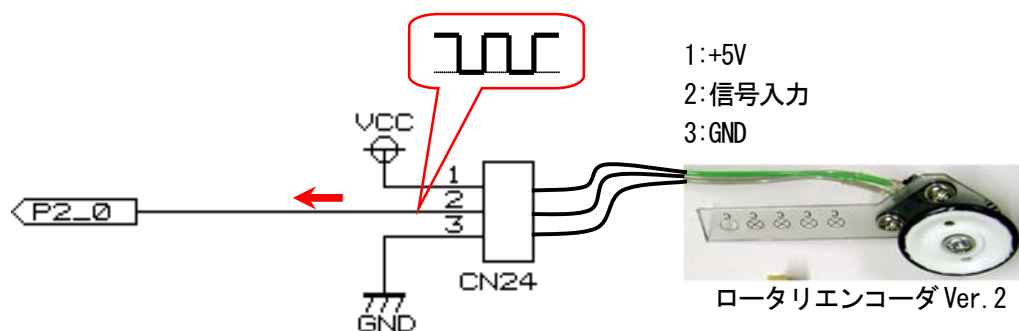
フリー端子を"0"にすると、U6、U7 は FET への出力電圧を常に 0V にしてモータを無接続状態、要はフリー動作にします。

3.2 ロータリエンコーダ信号入力回路

基板マイコンカーVer.2 の CN24 は、ロータリエンコーダの信号を入力するコネクタです。基板マイコンカーVer.2 のロータリエンコーダ回路、プログラムの特徴を、下記に示します。

- ロータリエンコーダの接続は、CN24
- ロータリエンコーダのパルスを、タイマ RD1 の外部クロック入力端子(P2_0)から入力し、立ち上がり、立ち下がりの回数をカウント

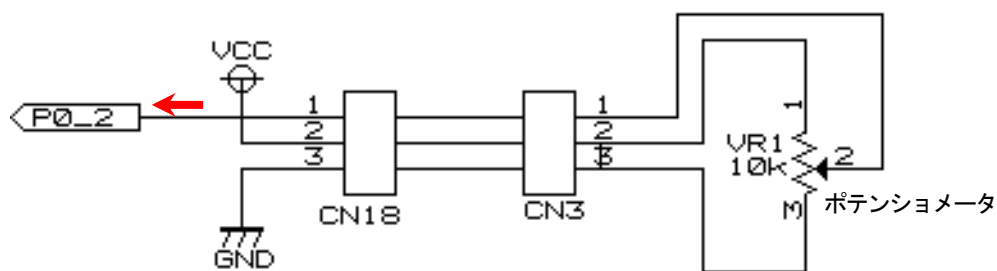
ロータリエンコーダ Ver.2 について詳しくは、「ロータリエンコーダ kit12_38a プログラム解説マニュアル(R8C/38A 版)」を参照してください。



3.3 ステアリング角度検出用ポテンシオメータ信号入力回路

基板マイコンカーVer.2 には、ステアリング角度検出用のポテンシオメータの入力コネクタが実装されています。基板マイコンカーVer.2 の回路、プログラムの特徴を下記に示します。

- 3ピン(抵抗の両端と可変部分がある)のポテンシオメータ(ボリューム)を取り付け可能
- 入力された電圧 0~5V を、R8C/35A マイコンの P0_2 端子で A/D 変換して 0~1023($2^{10}-1$)に変換




4. ワークスペース「rmc_frame_ver2」

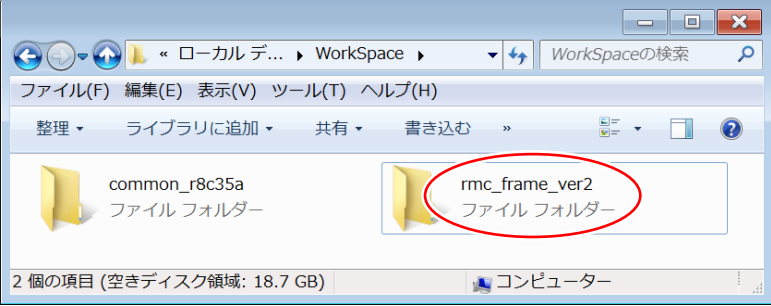
4.1 ワークスペースのインストール

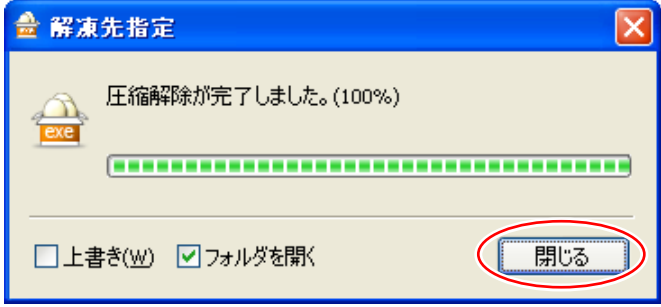
1	 <p>マイコンカーラリーとは? 今から始めるマイコンカーラリー 技術情報 会記録 MCRファン倶楽部 お問い合わせ</p> <p>ダウンロード</p> <p>上位入賞マイコンカー 参加者レポート お問い合わせ (FAQ)</p> <p>ダウンロード</p> <p>1/15 遅くなりましたが、2011年度の地区大会日程を掲載しました。下記の「2012年大会日程」をご覧ください。 JMCR2012大会へのホームページリニューアルは7月を予定しています。もうしばらくお待ちください。 /04 「お知らせ」ページにJMCR2012についてを掲載しました。 /18 「ダウンロード」ページにマイコンカーキットVer.5に関する「マイコンカーキットVer.5 本体組み立て製作マニュアル 第</p>	<p>マイコンカーラリーホームページ http://www.mcr.gr.jp/index2.html にアクセスします。 「技術情報→ダウンロード」をクリックします。</p>
---	---	--

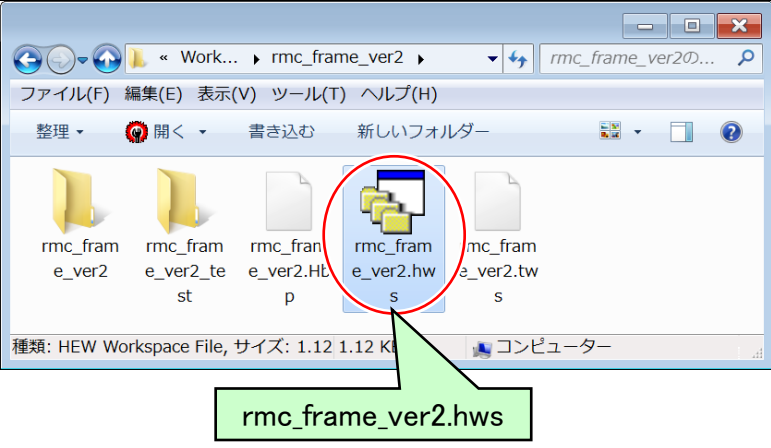
2	<p>免責事項</p> <p>「マニュアル」、「ソフトウェア」は万全な体制で制作されており、通常の使用環境においては正常に動作するように作成されていますが、万が一「マニュアル」、「ソフトウェア」による損失・損害が発生した時には、『ジャパンマイコンカーラリー実行委員会』はいかなる場合も責任を負いません。個人の免責が取れる範囲内であらかじめ了承した上でご使用くださるようお願いいたします。</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 15%;">対象マイコン</th> <th style="width: 55%;">内容</th> <th style="width: 30%;">更新日</th> </tr> </thead> <tbody> <tr> <td>R8C/38A</td> <td>R8C/38Aマイコン(RY_R8C38ボード)に関する資料</td> <td>2014.06.17</td> </tr> <tr> <td>R8C/35A</td> <td>R8C/35Aマイコンに関する資料</td> <td>2014.07.22</td> </tr> <tr> <td>R8C/</td> <td>R8C/M12Aマイコンに関する資料</td> <td>2014.07.22</td> </tr> </tbody> </table>	対象マイコン	内容	更新日	R8C/38A	R8C/38Aマイコン(RY_R8C38ボード)に関する資料	2014.06.17	R8C/35A	R8C/35Aマイコンに関する資料	2014.07.22	R8C/	R8C/M12Aマイコンに関する資料	2014.07.22	<p>「R8C/35A マイコンに関する資料」をクリックします。</p>
対象マイコン	内容	更新日												
R8C/38A	R8C/38Aマイコン(RY_R8C38ボード)に関する資料	2014.06.17												
R8C/35A	R8C/35Aマイコンに関する資料	2014.07.22												
R8C/	R8C/M12Aマイコンに関する資料	2014.07.22												

3	<p>■基板マイコンカーに関する資料</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 25%;">基板</th> <th style="width: 25%;">製作マニュアル</th> <th style="width: 25%;">プログラム解説マニュアル</th> <th style="width: 25%;">プログラム</th> </tr> </thead> <tbody> <tr> <td>基板マイコンカーVer.2 アナログセンサー基板TypeS Ver.2、ロータリエンコーダVer.2を使用し、基板を使ったフレームのAdvancedClass対応マイコンカーです。</td> <td>基板マイコンカーVer.2 組み立てマニュアル 第1.00版 2014.07.22</td> <td>基板マイコンカー製作キットVer.2 プログラム解説マニュアル 第1.00版 2014.07.22 内輪差、外輪差計算エクセルシート 2012.10.18</td> <td>rmc_frame_ver2.zip 2014.07.22</td> </tr> </tbody> </table>	基板	製作マニュアル	プログラム解説マニュアル	プログラム	基板マイコンカーVer.2 アナログセンサー基板TypeS Ver.2、ロータリエンコーダVer.2を使用し、基板を使ったフレームのAdvancedClass対応マイコンカーです。	基板マイコンカーVer.2 組み立てマニュアル 第1.00版 2014.07.22	基板マイコンカー製作キットVer.2 プログラム解説マニュアル 第1.00版 2014.07.22 内輪差、外輪差計算エクセルシート 2012.10.18	rmc_frame_ver2.zip 2014.07.22	<p>「rmc_frame_ver2.zip」をダウンロード、解凍します。</p>
基板	製作マニュアル	プログラム解説マニュアル	プログラム							
基板マイコンカーVer.2 アナログセンサー基板TypeS Ver.2、ロータリエンコーダVer.2を使用し、基板を使ったフレームのAdvancedClass対応マイコンカーです。	基板マイコンカーVer.2 組み立てマニュアル 第1.00版 2014.07.22	基板マイコンカー製作キットVer.2 プログラム解説マニュアル 第1.00版 2014.07.22 内輪差、外輪差計算エクセルシート 2012.10.18	rmc_frame_ver2.zip 2014.07.22							

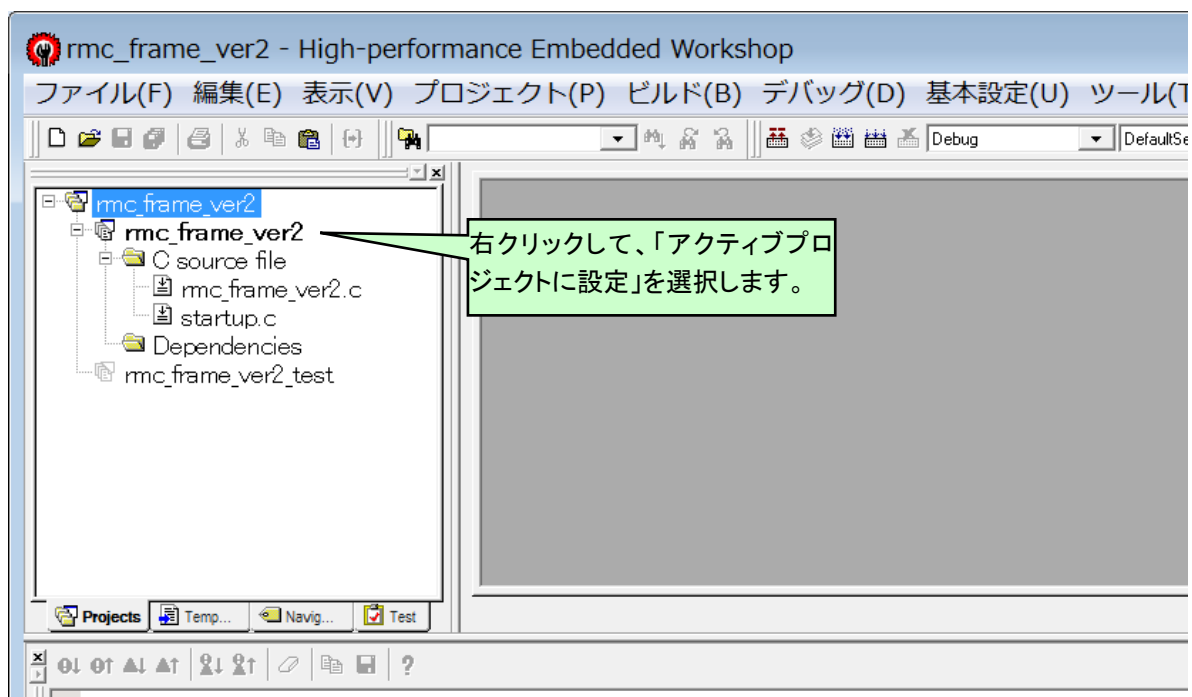
4		<p>圧縮解除をクリックします。</p> <p>※フォルダは変更できません。変更した場合は、ルネサス統合開発環境の設定を変更する場合がございます。</p>
---	---	---

5		<p>解凍が終わったら、自動的に「Cドライブ → Workspace」フォルダが開かれます。今回使用するの、「rmc_frame_ver2」です。</p>
---	--	---

6		<p>閉じるをクリックして終了です。</p>
---	--	------------------------

7		<p>「Cドライブ→ Workspace→ rmc_frame_ver2→ rmc_frame_ver2.hws」 をダブルクリックすると、ルネサス 統合開発環境が立ち上がります。</p>
---	--	--

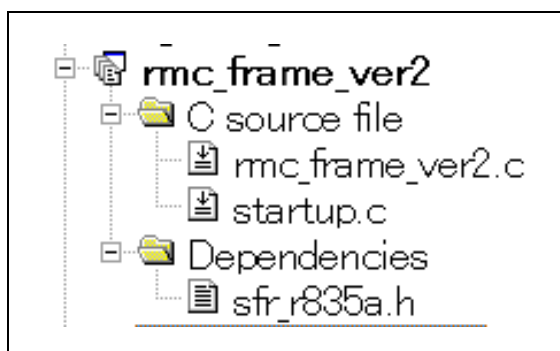
4.2 プロジェクト



ワークスペース「rnc_frame_ver2」には、2つのプロジェクトが登録されています。

プロジェクト名	内容
rnc_frame_ver2	RMC-R8C35A マイコンボード、アナログセンサ基板 TypeS Ver.2、ロータリエンコーダ Ver.2 を使った、基板マイコンカーVer.2 制御プログラムです。本プログラムは基本的な考え方のみ記述しています。実際にコースを完走させるには、各自プログラムを改造して対応してください。 今回は、このプロジェクトを使います。「rnc_frame_ver2_test」プロジェクトをアクティブ(操作対象)にしてください。
rnc_frame_ver2_test	基板マイコンカーVer.2 の動作テスト用プログラムです。

4.3 プロジェクトの構成



	ファイル名	内容
1	rmc_frame_ver2.c	実際に制御するプログラムが書かれています。R8C/35A の内蔵周辺機能(SFR)の初期化も行います。 ファイルの位置→C:\¥Workspace¥rmc_frame_ver2¥rmc_frame_ver2¥rmc_frame_ver2.c
2	startup.c	固定割り込みベクタアドレスの設定、スタートアッププログラム、RAMの初期化(初期値のないグローバル変数、初期値のあるグローバル変数の設定)などを行います。 ファイルの位置→C:\¥Workspace¥rmc_frame_ver2¥rmc_frame_ver2¥startup.c
3	sfr_r835a.h	R8C/35A マイコンの内蔵周辺機能を制御するためのレジスタ(Special Function Registers)を定義したファイルです。 ファイルの位置→C:\¥Workspace¥common_r8c35a¥sfr_r838a.h

5. プログラムの解説

5.1 プログラムリスト「rnc_frame_ver2.c」

```

1 : /******
2 : /* 対象マイコン R8C/35A or R8C/35C */
3 : /* ファイル内容 ミニマイコンカーVer.2のマイコンボード(RMC-R8C35A)を使った 基板マイコンカーVer.2制御プログラム */
4 : /*
5 : /* バージョン Ver. 1.00 */
6 : /* Date 2014.06.23 */
7 : /* Copyright ルネサスマイコンカーラリー事務局 */
8 : /* 株式会社日立ドキュメントソリューションズ */
9 : /******
10 :
11 : /*
12 : 本プログラムは、
13 : ●基板マイコンカーVer.2
14 : ●アナログセンサ基板TypeS Ver.2
15 : を使用した基板マイコンカーVer.2を動作させるプログラムです。
16 : */
17 :
18 : /*=====*/
19 : /* インクルード */
20 : /*=====*/
21 : #include <stdio.h>
22 : #include "sfr_r835a.h" /* R8C/35A SFRの定義ファイル */
23 :
24 : /*=====*/
25 : /* シンボル定義 */
26 : /*=====*/
27 : /* 定数設定 */
28 : #define TRC_MOTOR_CYCLE 20000 /* 左前,右前モータPWMの周期 */
29 : /* 50[ns] * 20000 = 1.00[ms] */
30 : #define TRD_MOTOR_CYCLE 20000 /* 左後,右後,サホモータPWMの周期 */
31 : /* 50[ns] * 20000 = 1.00[ms] */
32 : #define FREE 1 /* モータモード フリー */
33 : #define BRAKE 0 /* モータモード ブレーキ */
34 :
35 : /*=====*/
36 : /* プロトタイプ宣言 */
37 : /*=====*/
38 : void init( void );
39 : unsigned char sensor_inp( void );
40 : unsigned char center_inp( void );
41 : unsigned char startbar_get( void );
42 : unsigned char dipsw_get( void );
43 : unsigned char pushsw_get( void );
44 : void led_out( unsigned char led );
45 : void motor_r( int accele_l, int accele_r );
46 : void motor2_r( int accele_l, int accele_r );
47 : void motor_f( int accele_l, int accele_r );
48 : void motor2_f( int accele_l, int accele_r );
49 : void motor_mode_r( int mode_l, int mode_r );
50 : void motor_mode_f( int mode_l, int mode_r );
51 : void servoPwmOut( int pwm );
52 : int check_crossline( void );
53 : int getServoAngle( void );
54 : int getAnalogSensor( void );
55 : void servoControl( void );
56 : int diff( int pwm );
57 :
58 : /*=====*/
59 : /* グローバル変数の宣言 */
60 : /*=====*/
61 : int pattern; /* マイコンカー動作パターン */
62 : int crank_mode; /* 1:クランクモード 0:通常 */
63 : unsigned long cnt1; /* タイマ用 */
64 :
65 : /* エンコーダ関連 */
66 : int iTimer10; /* 10msカウント用 */
67 : long lEncoderTotal; /* 積算値保存用 */
68 : int iEncoder; /* 10ms毎の最新値 */
69 : unsigned int uEncoderBuff; /* 計算用 割り込み内で使用 */
70 :
71 : /* サーボ関連 */
72 : int iSensorBefore; /* 前回のセンサ値保存 */
73 : int iServoPwm; /* サーボPWM値 */
74 : int iAngle0; /* 中心時のA/D値保存 */
75 :
76 : /* センサ関連 */
77 : int iSensorPattern; /* センサ状態保持用 */
78 :

```



```

79 : /* TRCレジスタのバッファ */
80 : unsigned int   trcgrb_buff;           /* TRCGRBのバッファ */
81 : unsigned int   trcgrc_buff;           /* TRCGRCのバッファ */
82 : unsigned int   trcgrd_buff;           /* TRCGRDのバッファ */
83 :
84 : /* TRD0レジスタのバッファ */
85 : unsigned int   trdgrb0_buff;          /* TRDGRB0のバッファ */
86 : unsigned int   trdgrc0_buff;          /* TRDGRC0のバッファ */
87 :
88 : /* 内輪差値計算用 各マイコンカーに合わせて再計算して下さい */
89 : const revolution_difference[] = {     /* 角度から内輪、外輪回転差計算 */
90 :     100, 99, 97, 96, 95,
91 :     93, 92, 91, 89, 88,
92 :     87, 85, 84, 83, 82,
93 :     81, 79, 78, 77, 76,
94 :     74, 73, 72, 71, 70,
95 :     68, 67, 66, 65, 64,
96 :     62, 61, 60, 59, 57,
97 :     56, 55, 53, 52, 51,
98 :     50, 48, 47, 45, 44,
99 :     43 };
100 :
101 : /******
102 : /* メインプログラム */
103 : /******
104 : void main( void )
105 : {
106 :     int i;
107 :     unsigned char b;
108 :
109 :     /* マイコン機能の初期化 */
110 :     init();                               /* 初期化 */
111 :     asm(" fset I ");                       /* 全体の割り込み許可 */
112 :
113 :     /* マイコンカーの状態初期化 */
114 :     motor_mode_f( BRAKE, BRAKE );         /* 基板マイコンカーのFREEは、 */
115 :     motor_mode_r( BRAKE, BRAKE );         /* PWM値に関係なく必ずフリーになる */
116 :     motor_f( 0, 0 );
117 :     motor_r( 0, 0 );
118 :     servoPwmOut( 0 );
119 :
120 :     while( 1 ) {
121 :
122 :         switch( pattern ) {
123 :         case 0:
124 :             /* プッシュスイッチ(SW4)押下待ち */
125 :             servoPwmOut( 0 );
126 :             if( pushsw_get() == 0x01 ) {
127 :                 cnt1 = 0;
128 :                 pattern = 1;
129 :                 break;
130 :             }
131 :             i = (cnt1/200) % 2 + 1;
132 :             if( startbar_get() ) {
133 :                 i += ((cnt1/100) % 2 + 1) << 2;
134 :             }
135 :             led_out( i );                   /* LED点滅処理 */
136 :             break;
137 :
138 :         case 1:
139 :             /* スタートバー開待ち */
140 :             servoPwmOut( iServoPwm / 2 );
141 :             if( !startbar_get() ) {
142 :                 iAngle0 = getServoAngle(); /* 0度の位置記憶 */
143 :                 led_out( 0x0 );
144 :                 cnt1 = 0;
145 :                 pattern = 11;
146 :                 break;
147 :             }
148 :             led_out( 1 << (cnt1/50) % 4 );
149 :             break;
150 :

```

5. プログラムの解説

```
151 :     case 11:
152 :         /* 通常トレース */
153 :         servoPwmOut( iServoPwm );
154 :         i = getServoAngle();
155 :         if( i > 110 ) {
156 :             motor_f( 0, 0 );
157 :             motor_r( 0, 0 );
158 :         } else if( i > 15 ) {
159 :             motor_f( diff(80), 80 );
160 :             motor_r( diff(80), 80 );
161 :         } else if( i < -110 ) {
162 :             motor_f( 0, 0 );
163 :             motor_r( 0, 0 );
164 :         } else if( i < -15 ) {
165 :             motor_f( 80, diff(80) );
166 :             motor_r( 80, diff(80) );
167 :         } else {
168 :             motor_f( 100, 100 );
169 :             motor_r( 100, 100 );
170 :         }
171 :         if( check_crossline() ) { /* クロスラインチェック */
172 :             cnt1 = 0;
173 :             crank_mode = 1;
174 :             pattern = 21;
175 :         }
176 :         break;
177 :
178 :     case 21:
179 :         /* クロスライン通過処理 */
180 :         servoPwmOut( iServoPwm );
181 :         led_out( 0x3 );
182 :         motor_f( 0, 0 );
183 :         motor_r( 0, 0 );
184 :         if( cnt1 >= 100 ) {
185 :             cnt1 = 0;
186 :             pattern = 22;
187 :         }
188 :         break;
189 :
190 :     case 22:
191 :         /* クロスライン後のトレース、直角検出処理 */
192 :         servoPwmOut( iServoPwm );
193 :         if( iEncoder >= 11 ) { /* エンコーダによりスピード制御 */
194 :             motor_f( 0, 0 );
195 :             motor_r( 0, 0 );
196 :         } else {
197 :             motor2_f( 70, 70 );
198 :             motor2_r( 70, 70 );
199 :         }
200 :
201 :         if( (sensor_inp() & 0x01) == 0x01 ) { /* 右クランク? */
202 :             led_out( 0x1 );
203 :             cnt1 = 0;
204 :             pattern = 31;
205 :             break;
206 :         }
207 :         if( (sensor_inp() & 0x08) == 0x08 ) { /* 左クランク? */
208 :             led_out( 0x2 );
209 :             cnt1 = 0;
210 :             pattern = 41;
211 :             break;
212 :         }
213 :         break;
214 :
215 :     case 31:
216 :         /* 右クランク処理 */
217 :         servoPwmOut( 50 ); /* 振りが弱いときは大きくする */
218 :         motor_f( 60, 35 ); /* この部分は「角度計算(4WD時).xls」 */
219 :         motor_r( 48, 24 ); /* で計算 */
220 :         if( sensor_inp() == 0x04 ) { /* 曲げ終わりチェック */
221 :             cnt1 = 0;
222 :             iSensorPattern = 0;
223 :             crank_mode = 0;
224 :             pattern = 32;
225 :         }
226 :         break;
227 :
228 :     case 32:
229 :         /* 少し時間が経つまで待つ */
230 :         servoPwmOut( iServoPwm );
231 :         motor2_r( 80, 80 );
232 :         motor2_f( 80, 80 );
233 :         if( cnt1 >= 100 ) {
234 :             led_out( 0x0 );
235 :             pattern = 11;
236 :         }
237 :         break;
238 :
```

```

239 :     case 41:
240 :         /* 左クランク処理 */
241 :         servoPwmOut( -50 );           /* 振りが弱いときは大きくする */
242 :         motor_f( 35, 60 );           /* この部分は「角度計算(4WD時).xls」 */
243 :         motor_r( 24, 48 );           /* で計算 */
244 :         if( sensor_inp() == 0x02 ) { /* 曲げ終わりチェック */
245 :             cnt1 = 0;
246 :             iSensorPattern = 0;
247 :             crank_mode = 0;
248 :             pattern = 42;
249 :         }
250 :         break;
251 :
252 :     case 42:
253 :         /* 少し時間が経つまで待つ */
254 :         servoPwmOut( iServoPwm );
255 :         motor2_f( 80, 80 );
256 :         motor2_r( 80, 80 );
257 :         if( cnt1 >= 100 ) {
258 :             led_out( 0x0 );
259 :             pattern = 11;
260 :         }
261 :         break;
262 :
263 :     default:
264 :         break;
265 :     }
266 : }
267 : }
268 :
269 : /*****
270 : /* R8C/35A スペシャルファンクションレジスタ(SFR)の初期化 */
271 : /*****
272 : void init( void )
273 : {
274 :     int    i;
275 :
276 :     /* クロックをXINクロック(20MHz)に変更 */
277 :     prc0 = 1;           /* プロテクト解除 */
278 :     cml3 = 1;           /* P4_6, P4_7をXIN-XOUT端子にする */
279 :     cm05 = 0;           /* XINクロック発振 */
280 :     for(i=0; i<50; i++ ); /* 安定するまで少し待つ(約10ms) */
281 :     ocd2 = 0;           /* システムクロックをXINにする */
282 :     prc0 = 0;           /* プロテクトON */
283 :
284 :     /* ポートの入出力設定 */
285 :     /* bit7          bit6          bit5          bit4
286 :        bit3          bit2          bit1          bit0          */
287 :
288 :     /* センサデジタル右端   センサ左アナログ   センサ右アナログ   センサデジタル中心
289 :        センサスタートバー   角度ボリューム   未接続             フッシュスイッチ(SW2) */
290 :     p0 = 0x00;
291 :     prc2 = 1;           /* PD0のプロテクト解除 */
292 :     pd0 = 0x00;
293 :     pu00 = 1;           /* p0_0~p0_3プルアップON */
294 :
295 :     /* 未接続             未接続             RxD0             TxD0
296 :        マイコンポートのLED3   マイコンポートのLED2   マイコンポートのLED1   マイコンポートのLED0 */
297 :     p1 = 0x0f;
298 :     pd1 = 0xdf;
299 :
300 :     /* センサデジタル左端   センサデジタル左中   センサデジタル右中   未接続
301 :        フッシュスイッチ(SW4)   右後PWM(TRDGRB0)   右前PWM(TRDGRCO)   エンコーダパルス */
302 :     p2 = 0x00;
303 :     pd2 = 0x06;
304 :     pu04 = 1;           /* p2_0~p2_3プルアップON */
305 :
306 :     /* 左後BREAKorFREE   左後方向             左前BREAKorFREE   左前方向
307 :        右前BREAKorFREE   右前方向             右後BREAKorFREE   右後方向 */
308 :     p3 = 0xaa;
309 :     pd3 = 0xff;
310 :
311 :     /* XOUT             XIN             マイコンポートDIPSW2   マイコンポートDIPSW1
312 :        マイコンポートDIPSW0   VREF             端子無し             端子無し */
313 :     p4 = 0x00;
314 :     pd4 = 0x80;
315 :
316 :     /* マイコンポートDIPSW3   未接続             端子無し             端子無し
317 :        端子無し             端子無し             端子無し             端子無し */
318 :     p5 = 0x00;
319 :     pd5 = 0x40;
320 :
321 :     /* ステアPWM(TRCGRD)   左後PWM(TRCGRC)   左前PWM(TRCGRB)   フッシュスイッチ(SW1)
322 :        フッシュスイッチ(SW3)   未接続             ステアBREAKorFREE   ステア方向 */
323 :     p6 = 0x02;
324 :     pd6 = 0xe3;
325 :     pu14 = 1;           /* p6_0~p6_3プルアップON */
326 :     pu15 = 1;           /* p6_4~p6_7プルアップON */
327 :
328 :     pinsr = 0x08;       /* 入出力に関係なく端子レベルを読む */
329 :

```

5. プログラムの解説

```

330 :      /* A/Dコンバータの設定 */
331 :      admod = 0x33;          /* 繰り返し掃引モードに設定 */
332 :      adinsel = 0x30;       /* 入力端子P0の8端子を選択 */
333 :      adcon1 = 0x30;       /* A/D動作可能 */
334 :      asm(" nop ");        /* φADの1サイクルウエイト入れる*/
335 :      adcon0 = 0x01;       /* A/D変換スタート */
336 :
337 :      /* タイマRBの設定 */
338 :      /* 割り込み周期 = 1 / 20[MHz] * (TRBPRE+1) * (TRBPR+1)
339 :                  = 1 / (20*106) * 200 * 100
340 :                  = 0.001[s] = 1[ms]
341 :      */
342 :      trbmr = 0x00;        /* 動作モード、分周比設定 */
343 :      trbpre = 200-1;     /* プリスケアラレジスタ */
344 :      trbpr = 100-1;     /* プライマリレジスタ */
345 :      trbic = 0x05;      /* 割り込み優先レベル設定 */
346 :      trbcr = 0x01;     /* カウント開始 */
347 :
348 :      /* タイマRC PWMモード設定(左前モータ、右前モータ、ステアリングモータ) */
349 :      trcpsr0 = 0x60;     /* TRCIOA端子=なし, B端子=P6_5 */
350 :      trcpsr1 = 0x55;     /* TRCIOC端子=P6_6, D端子=P6_7 */
351 :      trcmr = 0x0f;      /* PWMモード選択ビット設定 */
352 :      trcrl = 0x8e;      /* ソースカウト:f1, 初期出力の設定 */
353 :      trcrr = 0x00;     /* 出力レベルの設定 */
354 :      trcgra = TRC_MOTOR_CYCLE - 1; /* 周期設定 */
355 :      trcgrb = trcgrb_buff = trcgra; /* P6_5端子のON幅(左前モータ) */
356 :      trcgrc = trcgrc_buff = trcgra; /* P6_6端子のON幅(左後モータ) */
357 :      trcgrd = trcgrd_buff = trcgra; /* P6_7端子のON幅(ステアリングモータ) */
358 :      trcic = 0x07;     /* 割り込み優先レベル設定 */
359 :      trcier = 0x01;    /* IMIAを許可 */
360 :      trcoer = 0x01;   /* 出力端子の選択 */
361 :      trcmr |= 0x80;   /* TRCカウント開始 */
362 :
363 :      /* タイマRD0 リセット同期PWMモード設定(左後モータ、右後モータ) */
364 :      trdpsr0 = 0x28;   /* TRDIOB0=P2_2, C0=P2_1, D0=なし */
365 :      trdpmr = 0x00;   /* レジスタは独立動作 */
366 :      trdpmr = 0x07;   /* PWM端子設定 B0, C0をPMW端子に */
367 :      trdfcr = 0x80;   /* アップダウン機能に設定 */
368 :      trdcr0 = 0x20;   /* ソースカウトの選択:f1 */
369 :      trdgra0 = TRD_MOTOR_CYCLE - 1; /* 周期設定 */
370 :      trdgrb0 = trdgrb0_buff = trdgra0; /* P2_2端子のON幅(右後モータ) */
371 :      trdgrc0 = trdgrc0_buff = trdgra0; /* P2_1端子のON幅(右前モータ) */
372 :      imiea_trdier0 = 1; /* IMFAビットによる割り込み許可 */
373 :      trd0ic = 0x06;   /* 割り込み優先レベル設定 */
374 :      polb_trdpocr0 = 0; /* TRDIOB0端子 出力レベルは"L"アクティブ */
375 :      polc_trdpocr0 = 0; /* TRDIOC0端子 出力レベルは"L"アクティブ */
376 :      tob0_trdocr = 1; /* TRDIOB0端子 初期はアクティブレベル */
377 :      toc0_trdocr = 1; /* TRDIOC0端子 初期はアクティブレベル */
378 :      trdoer1 = 0xf9; /* 出力端子の選択 */
379 :      tstart0_trdstr= 1; /* TRD0カウント開始 */
380 :
381 :      /* タイマRD1 外部入力 (ロータリエンコーダのパルスカウント) */
382 :      trdioa0sel0 = 1; /* TRDCLK端子:P2_0に設定 */
383 :      stclk_trdfcr = 1; /* 外部クロック入力有効に設定 */
384 :      trdcr1 = 0x15; /* TRDCLK端子 両エッジカウント */
385 :      tstart1_trdstr= 1; /* TRD1カウント開始 */
386 : }
387 :
388 : /*****
389 : /* タイマRB 割り込み処理
390 : *****/
391 : #pragma interrupt intTRB(vect=24)
392 : void intTRB( void )
393 : {
394 :     unsigned int i;
395 :
396 :     asm(" fset I "); /* タイマRB以上の割り込み許可 */
397 :
398 :     cnt1++;
399 :
400 :     /* サーボモータ制御 */
401 :     servoControl();
402 :
403 :     /* 10回中1回実行する処理 */
404 :     iTimer10++;
405 :     switch( iTimer10 ) {
406 :     case 1:
407 :         /* エンコーダ制御 */
408 :         i = trd1;
409 :         iEncoder = i - uEncoderBuff;
410 :         lEncoderTotal += iEncoder;
411 :         uEncoderBuff = i;
412 :         break;
413 :
414 :     case 2:
415 :         break;
416 :
417 :     case 3:
418 :         break;
419 :

```

```

420 :     case 4:
421 :         break;
422 :
423 :     case 5:
424 :         break;
425 :
426 :     case 6:
427 :         break;
428 :
429 :     case 7:
430 :         break;
431 :
432 :     case 8:
433 :         break;
434 :
435 :     case 9:
436 :         break;
437 :
438 :     case 10:
439 :         /* iTimer10変数の処理 */
440 :         iTimer10 = 0;
441 :         break;
442 :     }
443 : }
444 :
445 : /*****
446 : /* タイマRC 割り込み処理
447 : /*****
448 : #pragma interrupt /B intTRC(vect=7)
449 : void intTRC( void )
450 : {
451 :     /* タイマRC デューティ比の設定 */
452 :     trcgrd = trcgrd_buff;          /* ステアリングモータ PWMセット */
453 :     trcgrb = trcgrb_buff;          /* 右前モータ PWMセット */
454 :     trcgrc = trcgrc_buff;          /* 右後モータ PWMセット */
455 :
456 :     imfa_trcsr = 0;
457 : }
458 :
459 : /*****
460 : /* タイマR0D0 割り込み処理
461 : /*****
462 : #pragma interrupt intTRD0(vect=8)
463 : void intTRD0( void )
464 : {
465 :     asm(" fset I ");              /* タイマR0D0以上の割り込み許可 */
466 :
467 :     /* タイマR0D0 デューティ比の設定 */
468 :     trdgrb0 = trdgrb0_buff;        /* 左前モータ PWMセット */
469 :     trdgrc0 = trdgrc0_buff;        /* 左後モータ PWMセット */
470 :
471 :     imfa_trdsr0 = 0;
472 : }
473 :
474 : /*****
475 : /* アナログセンサ基板TypeS Ver. 2のデジタルセンサ値読み込み
476 : /* 引数 なし
477 : /* 戻り値 左端、左中、右中、右端のデジタルセンサ 0:黒 1:白
478 : /*****
479 : unsigned char sensor_inp( void )
480 : {
481 :     unsigned char sensor;
482 :
483 :     sensor = (p2_7<<3) | (p2_6<<2) | (p2_5<<1) | p0_7;
484 :     sensor = ~sensor;
485 :     sensor &= 0x0f;
486 :
487 :     return sensor;
488 : }
489 :
490 : /*****
491 : /* アナログセンサ基板TypeS Ver. 2の中心デジタルセンサ読み込み
492 : /* 引数 なし
493 : /* 戻り値 中心デジタルセンサ 0:黒 1:白
494 : /*****
495 : unsigned char center_inp( void )
496 : {
497 :     unsigned char sensor;
498 :
499 :     sensor = ~p0_4 & 0x01;
500 :
501 :     return sensor;
502 : }
503 :

```

5. プログラムの解説

```
504 : /******  
505 : /* アナログセンサ基板TypeS Ver.2のスタートバー検出センサ読み込み */  
506 : /* 引数 なし */  
507 : /* 戻り値 0:スタートバーなし 1:スタートバーあり */  
508 : /******  
509 : unsigned char startbar_get( void )  
510 : {  
511 :     unsigned char sensor;  
512 :  
513 :     sensor = ~p0_3 & 0x01;  
514 :  
515 :     return sensor;  
516 : }  
517 :  
518 : /******  
519 : /* マイコンボード上のディップスイッチ値読み込み */  
520 : /* 引数 なし */  
521 : /* 戻り値 スイッチ値 0~15 */  
522 : /******  
523 : unsigned char dipsw_get( void )  
524 : {  
525 :     unsigned char sw;  
526 :  
527 :     sw = (p5_7<<3) | (p4_5<<2) | (p4_4<<1) | p4_3;  
528 :  
529 :     return sw;  
530 : }  
531 :  
532 : /******  
533 : /* 基板マイコンカー上のプッシュスイッチ値読み込み(SW1~4) */  
534 : /* 引数 なし */  
535 : /* 戻り値 スイッチ値 bit3:SW1 bit2:SW2 bit1:SW3 bit0:SW4 0:OFF 1:ON */  
536 : /******  
537 : unsigned char pushsw_get( void )  
538 : {  
539 :     unsigned char sw;  
540 :  
541 :     sw = (p6_4<<3) | (p0_0<<2) | (p6_3<<1) | p2_3;  
542 :     sw = ~sw;  
543 :     sw &= 0x0f;  
544 :  
545 :     return sw;  
546 : }  
547 :  
548 : /******  
549 : /* マイコンボード上のLED制御 */  
550 : /* 引数 4個のLED制御 0:OFF 1:ON */  
551 : /* 戻り値 なし */  
552 : /******  
553 : void led_out( unsigned char led )  
554 : {  
555 :     unsigned char d;  
556 :  
557 :     d = p1 & 0xf0;  
558 :     p1 = d | (~led & 0x0f);  
559 : }  
560 :  
561 : /******  
562 : /* 後輪の速度制御 */  
563 : /* 引数 左モータ:-100~100, 右モータ:-100~100 */  
564 : /* 0で停止、100で正転100%、-100で逆転100% */  
565 : /* 戻り値 なし */  
566 : /******  
567 : void motor_r( int accele_l, int accele_r )  
568 : {  
569 :     int sw_data;  
570 :  
571 :     sw_data = dipsw_get() + 5; /* ディップスイッチ読み込み */  
572 :     accele_l = accele_l * sw_data / 20;  
573 :     accele_r = accele_r * sw_data / 20;  
574 :  
575 :     motor2_r( accele_l, accele_r );  
576 : }  
577 :
```

```

578 : /******  

579 : /* 後輪の速度制御2 ディップスイッチには関係しない\motor関数 */  

580 : /* 引数 左モータ:-100~100 , 右モータ:-100~100 */  

581 : /* 0で停止、100で正転100%、-100で逆転100% */  

582 : /* 戻り値 なし */  

583 : /* メモ 1~4%は、5%になります */  

584 : /******  

585 : void motor2_r( int accele_l, int accele_r )  

586 : {  

587 :     /* 左後モータ */  

588 :     if( accele_l >= 0 ) {  

589 :         p3_6 = 0;  

590 :     } else {  

591 :         p3_6 = 1;  

592 :         accele_l = -accele_l;  

593 :     }  

594 :     if( accele_l == 0 ) {  

595 :         // 0%のとき  

596 :         tregrc = tregrc_buff = trcgra;  

597 :     } else if( accele_l <= 5 ) {  

598 :         // 1~5%のときは、プログラムの仕様で5%とする  

599 :         tregrc_buff = (unsigned long)(TRC_MOTOR_CYCLE-1) * 5 / 100;  

600 :     } else if( accele_l <= 99 ) {  

601 :         // 6~99%のとき  

602 :         tregrc_buff = (unsigned long)(TRC_MOTOR_CYCLE-1) * accele_l / 100;  

603 :     } else {  

604 :         // 100%のとき(実際は99%を設定)  

605 :         tregrc_buff = (unsigned long)(TRC_MOTOR_CYCLE-1) * 99 / 100;  

606 :     }  

607 :  

608 :     /* 右後モータ */  

609 :     if( accele_r >= 0 ) {  

610 :         p3_0 = 0;  

611 :     } else {  

612 :         p3_0 = 1;  

613 :         accele_r = -accele_r;  

614 :     }  

615 :     if( accele_r == 0 ) {  

616 :         // 0%のとき  

617 :         trdgrb0 = trdgrb0_buff = trdgra0;  

618 :     } else if( accele_r <= 5 ) {  

619 :         // 1~5%のときは、プログラムの仕様で5%とする  

620 :         trdgrb0_buff = (unsigned long)(TRD_MOTOR_CYCLE-1) * 5 / 100;  

621 :     } else if( accele_r <= 99 ) {  

622 :         // 6~99%のとき  

623 :         trdgrb0_buff = (unsigned long)(TRD_MOTOR_CYCLE-1) * accele_r / 100;  

624 :     } else {  

625 :         // 100%のとき(実際は99%を設定)  

626 :         trdgrb0_buff = (unsigned long)(TRD_MOTOR_CYCLE-1) * 99 / 100;  

627 :     }  

628 : }  

629 :  

630 : /******  

631 : /* 前輪の速度制御 */  

632 : /* 引数 左モータ:-100~100 , 右モータ:-100~100 */  

633 : /* 0で停止、100で正転100%、-100で逆転100% */  

634 : /* 戻り値 なし */  

635 : /******  

636 : void motor_f( int accele_l, int accele_r )  

637 : {  

638 :     int sw_data;  

639 :  

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

641 :     accele_l = accele_l * sw_data / 20;  

642 :     accele_r = accele_r * sw_data / 20;  

643 :  

644 :     motor2_f( accele_l, accele_r );  

645 : }  

646 :  

647 : /******  

648 : /* 前輪の速度制御2 ディップスイッチには関係しない\motor関数 */  

649 : /* 引数 左モータ:-100~100 , 右モータ:-100~100 */  

650 : /* 0で停止、100で正転100%、-100で逆転100% */  

651 : /* 戻り値 なし */  

652 : /* メモ 1~4%は、5%になります */  

653 : /******  

654 : void motor2_f( int accele_l, int accele_r )  

655 : {  

656 :     /* 左前モータ */  

657 :     if( accele_l >= 0 ) {  

658 :         p3_4 = 0;  

659 :     } else {  

660 :         p3_4 = 1;  

661 :         accele_l = -accele_l;  

662 :     }

```

5. プログラムの解説

```

663 :     if( accele_l == 0 ) {
664 :         // 0%のとき
665 :         trcgrb = trcgrb_buff = trcgra;
666 :     } else if( accele_l <= 5 ) {
667 :         // 1~5%のときは、プログラムの仕様で5%とする
668 :         trcgrb_buff = (unsigned long)(TRC_MOTOR_CYCLE-1) * 5 / 100;
669 :     } else if( accele_l <= 99 ) {
670 :         // 6~99%のとき
671 :         trcgrb_buff = (unsigned long)(TRC_MOTOR_CYCLE-1) * accele_l / 100;
672 :     } else {
673 :         // 100%のとき(実際は99%を設定)
674 :         trcgrb_buff = (unsigned long)(TRC_MOTOR_CYCLE-1) * 99 / 100;
675 :     }
676 :
677 :     /* 右前モータ */
678 :     if( accele_r >= 0 ) {
679 :         p3_2 = 0;
680 :     } else {
681 :         p3_2 = 1;
682 :         accele_r = -accele_r;
683 :     }
684 :     if( accele_r == 0 ) {
685 :         // 0%のとき
686 :         trdgrc0 = trdgrc0_buff = trdgra0;
687 :     } else if( accele_r <= 5 ) {
688 :         // 1~5%のときは、プログラムの仕様で5%とする
689 :         trdgrc0_buff = (unsigned long)(TRD_MOTOR_CYCLE-1) * 5 / 100;
690 :     } else if( accele_r <= 99 ) {
691 :         // 6~99%のとき
692 :         trdgrc0_buff = (unsigned long)(TRD_MOTOR_CYCLE-1) * accele_r / 100;
693 :     } else {
694 :         // 100%のとき(実際は99%を設定)
695 :         trdgrc0_buff = (unsigned long)(TRD_MOTOR_CYCLE-1) * 99 / 100;
696 :     }
697 : }
698 :
699 : /******
700 : /* 後モータ動作 (BRAKE=動作とブレーキの繰り返し、FREE=0%のフリー) */
701 : /* 引数 左モータ:FREE or BRAKE , 右モータ:FREE or BRAKE */
702 : /* 戻り値 なし */
703 : /* メモ フリーにすると、PWM値に関わらず0%のフリーになります */
704 : /* モータタイプ基板TypeSとは動作が違いますので気を付けてください */
705 : /******
706 : void motor_mode_r( int mode_l, int mode_r )
707 : {
708 :     if( mode_l ) {
709 :         p3_7 = 0;
710 :     } else {
711 :         p3_7 = 1;
712 :     }
713 :     if( mode_r ) {
714 :         p3_1 = 0;
715 :     } else {
716 :         p3_1 = 1;
717 :     }
718 : }
719 :
720 : /******
721 : /* 前モータ動作 (BRAKE=動作とブレーキの繰り返し、FREE=0%のフリー) */
722 : /* 引数 左モータ:FREE or BRAKE , 右モータ:FREE or BRAKE */
723 : /* 戻り値 なし */
724 : /* メモ フリーにすると、PWM値に関わらず0%のフリーになります */
725 : /* モータタイプ基板TypeSとは動作が違いますので気を付けてください */
726 : /******
727 : void motor_mode_f( int mode_l, int mode_r )
728 : {
729 :     if( mode_l ) {
730 :         p3_5 = 0;
731 :     } else {
732 :         p3_5 = 1;
733 :     }
734 :     if( mode_r ) {
735 :         p3_3 = 0;
736 :     } else {
737 :         p3_3 = 1;
738 :     }
739 : }
740 :

```



```

741 : /******
742 : /* サーボモータ制御 */
743 : /* 引数 サーボモータPWM : -100~100 */
744 : /* 0で停止、100で右に100%、-100で左に100% */
745 : /* 戻り値 なし */
746 : /******
747 : void servoPwmOut( int pwm )
748 : {
749 :     if( pwm >= 0 ) {
750 :         p6_0 = 0;
751 :     } else {
752 :         p6_0 = 1;
753 :         pwm = -pwm;
754 :     }
755 :
756 :     if( pwm <= 2 ) {
757 :         trcgrd = trcgrd_buff = trcgra;
758 :     } if( pwm <= 99 ) {
759 :         trcgrd_buff = (unsigned long)(TRC_MOTOR_CYCLE-1) * pwm / 100;
760 :     } else {
761 :         trcgrd_buff = (unsigned long)(TRC_MOTOR_CYCLE-1) * 99 / 100;
762 :     }
763 : }
764 :
765 : /******
766 : /* クロスライン検出処理 */
767 : /* 引数 なし */
768 : /* 戻り値 0:クロスラインなし 1:あり */
769 : /******
770 : int check_crossline( void )
771 : {
772 :     unsigned char b;
773 :     int ret = 0;
774 :
775 :     b = sensor_inp();
776 :     if( b==0x0f || b==0x0e || b==0x0d || b==0x0b || b==0x07 ) {
777 :         ret = 1;
778 :     }
779 :     return ret;
780 : }
781 :
782 : /******
783 : /* サーボ角度取得 */
784 : /* 引数 なし */
785 : /* 戻り値 入れ替え後の値 */
786 : /******
787 : int getServoAngle( void )
788 : {
789 :     return( ad5 - iAngle0 );
790 : }
791 :
792 : /******
793 : /* アナログセンサ値取得 */
794 : /* 引数 なし */
795 : /* 戻り値 センサ値 */
796 : /******
797 : int getAnalogSensor( void )
798 : {
799 :     int ret;
800 :
801 :     ret = ad1 - ad2; /* アナログセンサ情報取得 */
802 :
803 :     if( !crank_mode ) {
804 :         /* クランクモードでなければ補正処理 */
805 :         switch( iSensorPattern ) {
806 :             case 0:
807 :                 if( sensor_inp() == 0x04 ) {
808 :                     ret = -650;
809 :                     break;
810 :                 }
811 :                 if( sensor_inp() == 0x02 ) {
812 :                     ret = 650;
813 :                     break;
814 :                 }
815 :                 if( sensor_inp() == 0x0c ) {
816 :                     ret = -700;
817 :                     iSensorPattern = 1;
818 :                     break;
819 :                 }
820 :                 if( sensor_inp() == 0x03 ) {
821 :                     ret = 700;
822 :                     iSensorPattern = 2;
823 :                     break;
824 :                 }
825 :                 break;
826 :

```

5. プログラムの解説

```
827 :         case 1:
828 :             /* センサ右寄り */
829 :             ret = -700;
830 :             if( sensor_inp() == 0x04 ) {
831 :                 iSensorPattern = 0;
832 :             }
833 :             break;
834 :
835 :         case 2:
836 :             /* センサ左寄り */
837 :             ret = 700;
838 :             if( sensor_inp() == 0x02 ) {
839 :                 iSensorPattern = 0;
840 :             }
841 :             break;
842 :     }
843 : }
844 :
845 : return ret;
846 : }
847 :
848 : /*****
849 : /* サーボモータ制御
850 : /* 引数 なし
851 : /* 戻り値 グローバル変数 iServoPwm に代入
852 : /*****
853 : void servoControl( void )
854 : {
855 :     int i, iRet, iP, iD;
856 :     int kp, kd;
857 :
858 :     i = getAnalogSensor();          /* センサ値取得
859 :     kp = 1;                          /* kpを増やしていき、ブルブルが
860 :     kd = 0;                          /* 大きくなったらkdを増やしてく
861 :                                         /* ださい
862 :     /* サーボモータ用PWM値計算 */
863 :     iP = kp * i;                      /* 比例
864 :     iD = kd * (iSensorBefore - i);    /* 微分(目安はPの5~10倍)
865 :     iRet = iP - iD;
866 :     iRet /= 64;
867 :
868 :     /* PWMの上限の設定 */
869 :     if( iRet > 50 ) iRet = 50;        /* マイコンカーが安定したら
870 :     if( iRet < -50 ) iRet = -50;     /* 上限を95くらいにしてください
871 :     iServoPwm = iRet;
872 :
873 :     iSensorBefore = i;                /* 今回はこの値が1ms前の値となる*/
874 : }
875 :
876 : /*****
877 : /* 外輪のPWMから、内輪のPWMを割り出す ハンドル角度は現在の値を使用
878 : /* 引数 外輪PWM
879 : /* 戻り値 内輪PWM
880 : /*****
881 : int diff( int pwm )
882 : {
883 :     int i, ret;
884 :
885 :     i = getServoAngle() / 3;         /* 1度あたりの増分で割る
886 :     if( i < 0 ) i = -i;
887 :     if( i > 45 ) i = 45;
888 :     ret = revolution_difference[i] * pwm / 100;
889 :
890 :     return ret;
891 : }
892 :
893 : /*****
894 : /* end of file
895 : /*****
896 :
897 : /*
898 : 改訂経歴
899 :
900 : 2014.06.23 Ver. 1.00 作成
901 : */
```

5.2 プログラムの解説

5.2.1 シンボル定義

```

24 : /*=====*/
25 : /* シンボル定義 */
26 : /*=====*/
27 : /* 定数設定 */
28 : #define TRC_MOTOR_CYCLE 20000 /* 左前,右前モータPWMの周期 */
29 : /* 50[ns] * 20000 = 1.00[ms] */
30 : #define TRD_MOTOR_CYCLE 20000 /* 左後,右後,サホモータPWMの周期 */
31 : /* 50[ns] * 20000 = 1.00[ms] */
32 : #define FREE 1 /* モータモード フリー */
33 : #define BRAKE 0 /* モータモード ブレーキ */
    
```

変数名	内容
TRC_MOTOR_CYCLE	<p>タイマ RC の PWM 波形の周期を決める値です。タイマ RC では、左前モータ、右前モータ、ステアリングモータを制御しています。</p> <p>タイマ RC カウントソースは 20MHz の水晶振動子を使います。周期は次のようになります。</p> $1 / (20 \times 10^6) = 50.00[\text{ns}]$ <p>今回、PWM 周期は 1ms にします。TRC_MOTOR_CYCLE は、次のようになります。</p> $\text{PWM 周期} / \text{タイマ RC カウントソース} = (1 \times 10^{-3}) / (50 \times 10^{-9}) = 20,000$
TRD_MOTOR_CYCLE	<p>タイマ RD0 の PWM 波形の周期を決める値です。タイマ RD0 では、左後モータ、右後モータを制御しています。</p> <p>タイマ RD0 カウントソースは 20MHz の水晶振動子を使います。周期は次のようになります。</p> $1 / (20 \times 10^6) = 50.00[\text{ns}]$ <p>今回、PWM 周期は 1ms にします。TRD_MOTOR_CYCLE は、次のようになります。</p> $\text{PWM 周期} / \text{タイマ RD0 カウントソース} = (1 \times 10^{-3}) / (50 \times 10^{-9}) = 20,000$
FREE BRAKE	<p>motor_mode_r 関数、motor_mode_f 関数で使用する定数です。</p> <p>モータをフリー動作にしたい場合は「FREE」、PWM 出力時の「0」の信号をブレーキにしたい場合は「BRAKE」を引数にセットします。</p> <p>例) motor_mode_f(BRAKE , FREE); // 左前モータはブレーキ、右前モータはフリー motor_mode_r(FREE, BRAKE); // 左後モータはフリー、右後モータはブレーキ</p> <p>※基板マイコンカーVer.2 のフリーは、motor_r 関数、motor_f 関数の値に関係なく、PWM が 0%のフリー動作になります。モータドライブ基板 TypeS Ver.4 のフリーとは、動作が異なります。</p>

5. プログラムの解説

5.2.2 変数の定義

```

58 : /*=====*/
59 : /* グローバル変数の宣言 */
60 : /*=====*/
61 : int          pattern;          /* マイコンカー動作パターン */
62 : int          crank_mode;      /* 1:クランクモード 0:通常 */
63 : unsigned long cnt1;          /* タイマ用 */
64 :
65 : /* エンコーダ関連 */
66 : int          iTimer10;        /* 10msカウント用 */
67 : long         lEncoderTotal;   /* 積算値保存用 */
68 : int          iEncoder;        /* 10ms毎の最新値 */
69 : unsigned int uEncoderBuff;    /* 計算用 割り込み内で使用 */
70 :
71 : /* サーボ関連 */
72 : int          iSensorBefore;   /* 前回のセンサ値保存 */
73 : int          iServoPwm;       /* サーボPWM値 */
74 : int          iAngle0;         /* 中心時のA/D値保存 */
75 :
76 : /* センサ関連 */
77 : int          iSensorPattern;  /* センサ状態保持用 */
78 :
79 : /* TRCレジスタのバッファ */
80 : unsigned int trcgrb_buff;     /* TRCGRBのバッファ */
81 : unsigned int trcgrc_buff;     /* TRCGRCのバッファ */
82 : unsigned int trcgrd_buff;     /* TRCGRDのバッファ */
83 :
84 : /* TRD0レジスタのバッファ */
85 : unsigned int trdgrb0_buff;    /* TRDGRB0のバッファ */
86 : unsigned int trdgrc0_buff;    /* TRDGRC0のバッファ

```

変数名	内容
pattern	マイコンカーの現在の動作パターンを設定します。
crank_mode	アナログセンサ値を、デジタルセンサを使って補正するかしないか設定します。 0:補正 ON(通常トレース状態) 1:補正 OFF(クロスラインを検出後とクランクトレースモード時など)
cnt1	タイマです。1msごとに増加していきます。この変数を使って100ms待つなど、時間のカウントをします。
iTimer10	ロータリエンコーダ処理は、タイマRBの割り込み関数内で行います。割り込みは1msごとに発生しますが、ロータリエンコーダ処理は10msごとです。そのため、この変数をタイマRB割り込み処理で足していき、10になったら処理するようにすれば10msごとに処理するのと同じこととなります。
lEncoderTotal	ロータリエンコーダの積算値が保存されています。この値で、基板マイコンカーVer.2がスタートしてから距離が分かります。10msごとに更新されます。
iEncoder	10ms間のロータリエンコーダのパルス値が保存されます。この値で、基板マイコンカーVer.2の速度が分かります。10msごとに更新されます。
uEncoderBuff	ロータリエンコーダ変数の計算用です。通常のプログラムでは使用しません。
iSensorBefore	前回のアナログセンサ値を保存します。通常のプログラムでは使用しません。

iServoPwm	タイマ RB の割り込み関数内で計算したサーボモータ用の PWM 値保存用です。
iAngle0	ステアリング角度 0 度のときのボリューム A/D 値を保存します。
iSensorPattern	アナログセンサの A/D 値を取得する getAnalogSensor 関数で使います。アナログセンサが中央ラインをはずれたときの対処用です。通常のプログラムでは使用しません。
trcgrb_buff	TRCGRB のバッファとして使います。TRCGRB の値は、P6_5 端子から出力する PWM 波形の ON 幅(左前モータ)を設定するレジスタです。 TRCGRB の値を変えるとき、直接このレジスタの値を変えのではなく、trcgrb_buff 変数の値を変更します。タイマ RC の割り込み関数で、trcgrb_buff 変数の値を、TRCGRB に設定します。 これは、直接 TRCGRB の値を変更すると、PWM 波形が乱れることがあるためです。詳しくは後述します。
trcgrc_buff	TRCGRC のバッファとして使います。TRCGRC の値は、P6_6 端子から出力する PWM 波形の ON 幅(左後モータ)を設定するレジスタです。 TRCGRC の値を変えるとき、直接このレジスタの値を変えのではなく、trcgrc_buff 変数の値を変更します。タイマ RC の割り込み関数で、trcgrc_buff 変数の値を、TRCGRC に設定します。 これは、直接 TRCGRC の値を変更すると、PWM 波形が乱れることがあるためです。詳しくは後述します。
trcgrd_buff	TRCGRD のバッファとして使います。TRCGRD の値は、P6_7 端子から出力する PWM 波形の ON 幅(ステアリングモータ)を設定するレジスタです。 TRCGRD の値を変えるとき、直接このレジスタの値を変えのではなく、trcgrd_buff 変数の値を変更します。タイマ RC の割り込み関数で、trcgrd_buff 変数の値を、TRCGRD に設定します。 これは、直接 TRCGRD の値を変更すると、PWM 波形が乱れることがあるためです。詳しくは後述します。
trdgrb0_buff	TRDGRB0 のバッファとして使います。TRDGRB0 の値は、P2_2 端子から出力する PWM 波形の ON 幅(右後モータ)を設定するレジスタです。 TRDGRB0 の値を変えるとき、直接このレジスタの値を変えのではなく、trdgrb0_buff 変数の値を変更します。タイマ RD0 の割り込み関数で、trdgrb0_buff 変数の値を、TRDGRB0 に設定します。 これは、直接 TRDGRB0 の値を変更すると、PWM 波形が乱れることがあるためです。詳しくは後述します。
trdgrc0_buff	TRDGRC0 のバッファとして使います。TRDGRC0 の値は、P2_1 端子から出力する PWM 波形の ON 幅(右前モータ)を設定するレジスタです。 TRDGRC0 の値を変えるとき、直接このレジスタの値を変えのではなく、trdgrc0_buff 変数の値を変更します。タイマ RD0 の割り込み関数で、trdgrc0_buff 変数の値を、TRDGRC0 に設定します。 これは、直接 TRDGRC0 の値を変更すると、PWM 波形が乱れることがあるためです。詳しくは後述します。

5.2.3 内輪差値計算用の配列追加

```

88 : /* 内輪差値計算用 各マイコンカーに合わせて再計算して下さい */
89 : const revolution_difference[] = { /* 角度から内輪、外輪回転差計算 */
90 :     100, 99, 97, 96, 95,
91 :     93, 92, 91, 89, 88,
92 :     87, 85, 84, 83, 82,
93 :     81, 79, 78, 77, 76,
94 :     74, 73, 72, 71, 70,
95 :     68, 67, 66, 65, 64,
96 :     62, 61, 60, 59, 57,
97 :     56, 55, 53, 52, 51,
98 :     50, 48, 47, 45, 44,
99 :     43 };

```

revolution_difference という回転の差を計算した配列を追加します。この配列は const を先頭に付けています。値の変更しない変数や配列は RAM 上に配置する必要はありません。const 型修飾子を指定するとマイコンの ROM エリアに配置されます。今回の R8C/35A マイコンはプログラム ROM が 32KB、内蔵 RAM が 2.5KB と RAM が少ないので、RAM の有効活用を考えて const を付けました。const を取ると RAM エリアに配置されます。

revolution_difference 配列の [] 内に数字を入れると、入れた数字番目の数値が返ってきます。[] の中に入る数字を添字といいます。添字を入れたときの値を下記に示します。

```

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

```

添字に 46 以上の値を設定してもエラーにはなりません、不定な値が返ってきます。添字は 46 以上にしないようにしてください。

値の意味は、外輪の回転を 100 としたとき、添字に現在のステアリング角度(単位:度)を入れると内輪の回転数が返ってきます。

例えば添字が 2 のとき、97 が返ってきます。これは外輪 100、ステアリング角度が 2 度のとき、内輪の回転数は 97 ということです。ステアリング角度が 0 度～45 度のとき、内輪の値はあらかじめ計算しておきます。今回は、基板マイコンカー Ver.2 のホイールベース=0.18、トレッド=0.145 で計算しました。

本プログラムでは、後輪の内外輪差のみ計算し、前輪は後輪の値を使っています。4 輪の内外輪差を計算したい場合は、「7. 4 輪の回転数計算」を参考にして、4 輪の配列を作ると良いでしょう。

5.2.4 init 関数のクロックの選択

R8C/35A マイコンは起動時、マイコン内蔵の低速オンチップオシレータ(約 125kHz)で動作します。これを P4_6 端子、P4_7 端子に接続されている 20MHz の水晶振動子に切り替えます。

```
269 : /*****  
270 : /* R8C/35A スペシャルファンクションレジスタ(SFR)の初期化 */  
271 : /*****  
272 : void init( void )  
273 : {  
274 :     int    i;  
275 :  
276 :     /* クロックをXINクロック(20MHz)に変更 */  
277 :     prc0 = 1;          /* プロテクト解除          */  
278 :     cm13 = 1;         /* P4_6, P4_7をXIN-XOUT端子にする*/  
279 :     cm05 = 0;         /* XINクロック発振          */  
280 :     for(i=0; i<50; i++ ); /* 安定するまで少し待つ(約10ms) */  
281 :     ocd2 = 0;         /* システムクロックをXINにする */  
282 :     prc0 = 0;         /* プロテクトON          */
```

277~282 行で、低速オンチップオシレータ(約 125kHz)から、外付けの水晶振動子(20MHz)に切り替えています。詳しくは、「マイコン実習マニュアル(R8C/35A 版)」を参照してください。

5. プログラムの解説

5.2.5 ポートの入出力設定

```

284 : /* ポートの入出力設定 */
285 : /* bit7          bit6          bit5          bit4
286 :    bit3          bit2          bit1          bit0          */
287 :
288 : /* センサデジタル右端 センサ左アナログ センサ右アナログ センサデジタル中心
289 :    センサスタートパル 角度ボリューム 未接続          プッシュスイッチ(SW2) */
290 : p0  = 0x00;
291 : prc2 = 1; /* PD0のプロテクト解除 */
292 : pd0  = 0x00;
293 : pu00 = 1; /* p0_0~p0_3プルアップON */
294 :
295 : /* 未接続          未接続          RxD0          TxD0
296 :    マイコンポートのLED3 マイコンポートのLED2 マイコンポートのLED1 マイコンポートのLED0 */
297 : p1  = 0x0f;
298 : pd1 = 0xdf;
299 :
300 : /* センサデジタル左端 センサデジタル左中 センサデジタル右中 未接続
301 :    プッシュスイッチ(SW4) 右後PWM (TRDGRBO) 右前PWM (TRDGRCO) エンコーダパルス*/
302 : p2  = 0x00;
303 : pd2 = 0x06;
304 : pu04 = 1; /* p2_0~p2_3プルアップON */
305 :
306 : /* 左後BREAKorFREE 左後方向          左前BREAKorFREE 左前方向
307 :    右前BREAKorFREE 右前方向          右後BREAKorFREE 右後方向 */
308 : p3  = 0xaa;
309 : pd3 = 0xff;
310 :
311 : /* XOUT          XIN          マイコンポートDIPSW2 マイコンポートDIPSW1
312 :    マイコンポートDIPSW0 VREF          端子無し          端子無し */
313 : p4  = 0x00;
314 : pd4 = 0x80;
315 :
316 : /* マイコンポートDIPSW3 未接続          端子無し          端子無し
317 :    端子無し          端子無し          端子無し          端子無し */
318 : p5  = 0x00;
319 : pd5 = 0x40;
320 :
321 : /* ステアPWM (TRCGRD) 左後PWM (TRCGRC) 左前PWM (TRCGRB) プッシュスイッチ(SW1)
322 :    プッシュスイッチ(SW3) 未接続          ステアBREAKorFREE ステア方向 */
323 : p6  = 0x02;
324 : pd6 = 0xe3;
325 : pu14 = 1; /* p6_0~p6_3プルアップON */
326 : pu15 = 1; /* p6_4~p6_7プルアップON */
327 :
328 : pinsr = 0x08; /* 入出力に関係なく端子レベルを読む*/

```

PD0~PD6 で、ポート0~ポート6の入出力設定を行います。PD0~PD6の該当ビットを"1"にするとそのビットが出力、"0"にすると入力になります。

(1) ポートの入出力

基板マイコンカーVer.2 の接続機器に合わせてポートの入出力設定を行います。ポートの接続状態を、下表に示します。

ポート	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
0	デジタル センサ右端 入力	アナログ センサ左 アナログ入力	アナログ センサ右 アナログ入力	デジタル センサ中心 入力	スタートバー 検出センサ 入力	ポテンショメータ(ステアリング 角度検出) アナログ入力	未接続	タクトスイッチ 2 入力
	内蔵プルアップ OFF				内蔵プルアップ ON			
1	未接続	未接続	マイコンボード RxD 入力	マイコンボード TxD 出力	マイコンボード LED3 出力	マイコンボード LED2 出力	マイコンボード LED1 出力	マイコンボード LED0 出力
	内蔵プルアップ OFF				内蔵プルアップ OFF			
2	デジタル センサ左端 入力	デジタル センサ左中 入力	デジタル センサ右中 入力	未接続	タクトスイッチ 4 入力	右後モータ PWM 出力	右前モータ PWM 出力	ロータリ エンコーダ パルス入力
	内蔵プルアップ OFF				内蔵プルアップ ON			
3	左後モータ フリー/ブレーキ 出力	左後モータ 正転/逆転 出力	左前モータ フリー/ブレーキ 出力	左前モータ 正転/逆転 出力	右前モータ フリー/ブレーキ 出力	右前モータ 正転/逆転 出力	右後モータ フリー/ブレーキ 出力	右後モータ 正転/逆転 出力
	内蔵プルアップ OFF				内蔵プルアップ OFF			
4	水晶振動子 (20MHz) 出力	水晶振動子 (20MHz) 入力	マイコンボード ディップスイッチ 2 入力	マイコンボード ディップスイッチ 1 入力	マイコンボード ディップスイッチ 0 入力	Vcc 入力	/	/
	内蔵プルアップ OFF				内蔵プルアップ OFF			
5	マイコンボード ディップスイッチ 3 入力	未接続	/	/	/	/	/	/
	内蔵プルアップ OFF							
6	ステアリングモータ PWM 出力	左後モータ PWM 出力	左前モータ PWM 出力	タクトスイッチ 1 入力	タクトスイッチ 3 入力	未接続	ステアリングモータ フリー/ブレーキ 出力	ステアリングモータ 正転/逆転 出力
	内蔵プルアップ ON				内蔵プルアップ ON			

※表の斜線の bit は、端子がない bit です。

※未接続ポートは出力設定にします。

※リセット後は、全て入力ポートです。

(2) 端子のプルアップ

タクトスイッチ 1~4 には、プルアップ抵抗が接続されていないので、マイコン内蔵のプルアップ抵抗を使用します。

マイコン内蔵のプルアップ抵抗は、例えば P0_3~P0_0 など、一部を除いて複数端子を ON するか OFF するか選択できません。タクトスイッチ以外の端子も内蔵プルアップ抵抗を ON にしてしまいがちですが、内蔵プルアップ抵抗は、標準値 50kΩ(最小値:25kΩ、最大値:100kΩ)と、抵抗値が大きいため、今回の回路には影響ありません。

5. プログラムの解説

5.2.6 A/D コンバータの設定

A/D コンバータを使い、アナログセンサ左、アナログセンサ右、ポテンショメータの各電圧を、デジタル値に変換します。

```

330 :      /* A/D コンバータの設定 */
331 :      admod   = 0x33;          /* 繰り返し掃引モードに設定 */
332 :      adinsel = 0x30;          /* 入力端子 P0 の 8 端子を選択 */
333 :      adcon1  = 0x30;          /* A/D 動作可能 */
334 :      asm(" nop ");           /* φAD の 1 サイクルウェイト入れる*/
335 :      adcon0  = 0x01;          /* A/D 変換スタート */
    
```

A/D コンバータに関するレジスタ設定を下記に示します。

(1) A/D モードレジスタ(ADM0D:A-D mode register)の設定

ビット	シンボル	説明	設定値
7	adcap1	"0"を設定	0x33
6	adcap0	"0"を設定	
5	md2	A/D 動作モード選択を設定します。 000:単発モード 001:設定しないでください	
4	md1	010:繰り返しモード 0 011:繰り返しモード 1 100:単掃引モード 101:設定しないでください	
3	md0	110:繰り返し掃引モード 111:設定しないでください	
2	cks2	クロック源選択ビットを設定します。 0:f1 (20MHz)を選択 1:fOCO-F(高速オンチップオシレータ)を選択	
1	cks1	分周選択ビットを設定します。 00:fAD の 8 分周 (8/20MHz=400ns) 01:fAD の 4 分周 (4/20MHz=200ns) 10:fAD の 2 分周 (2/20MHz=100ns)	
0	cks0	11:fAD の 1 分周 (1/20MHz=50ns) fAD とは、bit2 で設定したクロック源のことです。このクロックを何分周で使用するか選択します。	

(2) A/D 入力選択レジスタ (ADINSEL: A-D input select register)

ビット	シンボル	説明	設定値	
7	adgsel1	どのアナログ入力端子を A/D 変換するか設定します。 0000: AN0(P0_7)~AN1(P0_6)の 2 端子 0001: AN0(P0_7)~AN3(P0_4)の 4 端子 0010: AN0(P0_7)~AN5(P0_2)の 6 端子 0011: AN0(P0_7)~AN7(P0_0)の 8 端子 0100: AN8(P1_0)~AN9(P1_1)の 2 端子 0101: AN8(P1_0)~AN11(P1_3)の 4 端子	0x30	
6	adgsel0			
5	scan1			
4	scan0			
3	-			"0"を設定
2	ch2			"0"を設定
1	ch1			"0"を設定
0	ch0			"0"を設定

今回は、P0_6 端子にアナログセンサ左、P0_5 端子にアナログセンサ右、P0_2 端子にステアリング角度検出用ポテンショメータが接続されています。これらの端子を A/D 入力端子にします。それ以外の端子を A/D 変換にしても問題は無いので、ポート 0 の 8 端子を A/D 変換する設定にします。

(3) A/D 制御レジスタ 1 (ADCON1: A-D control register1)

ビット	シンボル	説明	設定値
7	adddaen	"0"を設定	0x30
6	adddaen	"0"を設定	
5	adstby	A/D スタンバイビットを設定します。 0: A/D 動作停止(スタンバイ) 1: A/D 動作可能 この bit を"0"から"1"にしたときは、 ϕ A/D の 1 サイクル以上経過した後に A/D 変換を開始します。	
4	bits	8/10 ビットモード選択ビットを設定します。 0: 8 ビットモード 1: 10 ビットモード	
3	-	"0"を設定	
2	-	"0"を設定	
1	-	"0"を設定	
0	adex0	"0"を設定	

(4) ϕ AD の 1 サイクル以上ウェイトを入れる

adstby ビットを設定した後、 ϕ A/D の 1 サイクル以上経過した後に A/D 変換を開始しなければいけません。そのウェイトを入れるため、アセンブリ言語の nop 命令を実行します。C 言語ソースプログラム内では、アセンブリ言語は実行できないため、asm 命令というアセンブリ言語を実行できる C 言語の命令を使って nop 命令を実行します。ちなみに、nop は「no operation (何もしない)」命令で、この命令を実行するのに 1 サイクル分(50ns)の時間がかかります。

```
asm( " nop " );
```

5. プログラムの解説

(5) A/D 制御レジスタ 0(ADCON0:A-D control register0)

ビット	シンボル	説明	設定値
7	-	"0"を設定	0x01
6	-	"0"を設定	
5	-	"0"を設定	
4	-	"0"を設定	
3	-	"0"を設定	
2	-	"0"を設定	
1	-	"0"を設定	
0	adst	A/D 変換開始フラグを設定します。 0:A/D 変換停止 1:A/D 変換開始	

5.2.7 タイマ RB の設定

タイマ RB を使い、1ms ごとに割り込みを発生させます。

```

337 :      /* タイマ RB の設定 */
338 :      /* 割り込み周期 = 1 / 20[MHz] * (TRBPRE+1) * (TRBPR+1)
339 :                        = 1 / (20*10^6) * 200 * 100
340 :                        = 0.001[s] = 1[ms]
341 :      */
342 :      trbmr  = 0x00;          /* 動作モード、分周比設定 */
343 :      trbpre = 200-1;        /* プリスケアラレジスタ */
344 :      trbpr  = 100-1;        /* プライマリレジスタ */
345 :      trbic  = 0x05;          /* 割り込み優先レベル設定 */
346 :      trbcr  = 0x01;          /* カウント開始 */
    
```

タイマ RB に関するレジスタ設定を下記に示します。

(1) タイマ RB モードレジスタ(TRBMR: Timer RB mode register)の設定

ビット	シンボル	説明	設定値
7	tckcut_trbmr	"0"を設定	0x00
6	-	"0"を設定	
5	tck1_trbmr	タイマ RB カウントソース選択ビットを設定します。 00:f1 (1/20MHz=50ns) 01:f8 (8/20MHz=400ns)	
4	tck0_trbmr	10:タイマ RA のアンダフロー 11:f2 (2/20MHz=100ns)	
3	-	"0"を設定	
2	twrc_trbmr	"0"を設定	
1	tmod1_trbmr	タイマ RB 動作モード選択ビットを設定します。 00:タイマモード 01:プログラマブル波形発生モード	
0	tmod0_trbmr	10:プログラマブルワンショット発生モード 11:プログラマブルウェイトワンショット発生モード	

(2) タイマ RB プリスケアラレジスタ(TRBPRES: Timer RB prescaler register)の設定

(3) タイマ RB プライマリレジスタ(TRBPR: Timer RB Primary Register)の設定

ビット	シンボル	説明	設定値
TRBPRES 7~0	-	<p>割り込み周期を設定します。計算式を下記に示します。</p> $(TRBPRES+1) \times (TRBPR+1) = \text{タイマ RB 割り込み要求周期} / \text{タイマ RB カウントソース}$ <p>今回、タイマ RB 割り込み要求周期は 1ms です。タイマ RB カウントソースは、TRBMR で設定した 50ns です。よって、</p> $(TRBPRES+1) \times (TRBPR+1) = 1 \times 10^{-3} / 50 \times 10^{-9}$ $= 20,000$ <p>これを満たす TRBPRES、TRBPR を探します。今回は、 $TRBPRES+1=200$、$TRBPR=200-1$ $TRBPR+1=100$、$TRBPRES=100-1$ にします。 ただし、下記を満たすようにしてください。 ・TRBPRES ≤ 255 ・TRBPR ≤ 255</p>	200-1
TRBPR 7~0	-	上記の計算により、100-1 にします。	100-1

(4) タイマ RB 割り込み制御レジスタ(TRBIC: Timer RB interrupt control register)の設定

ビット	シンボル	説明	設定値
7	-	"0"を設定	0x05
6	-	"0"を設定	
5	-	"0"を設定	
4	-	"0"を設定	
3	ir_trbic	"0"を設定	
2	ilvl2_trbic	他の割り込みが同時に発生した場合、どの割り込みを優先させるか設定します。レベルの高い割り込みが優先されます。割り込みを 2 つ以上使う場合、どれを優先させるかここで決めます。今回はタイマ RC 割り込みを 7、タイマ RD0 割り込みを 6、タイマ RB 割り込みを 5 に設定します。	
1	ilvl1_trbic	000:レベル 0 (割り込み禁止) 001:レベル 1 010:レベル 2 011:レベル 3	
0	ilvl0_trbic	100:レベル 4 101:レベル 5 110:レベル 6 111:レベル 7	

5. プログラムの解説

(5) タイマ RB 制御レジスタ (TRBCR: Timer RB Control Register) の設定

ビット	シンボル	説明	設定値
7	-	"0"を設定	0x01
6	-	"0"を設定	
5	-	"0"を設定	
4	-	"0"を設定	
3	-	"0"を設定	
2	tstop_trbcr	"0"を設定	
1	tcstf_trbcr	"0"を設定	
0	tstart_trbcr	タイマ RB カウント開始ビットを設定します。 0:カウント停止 1:カウント開始	

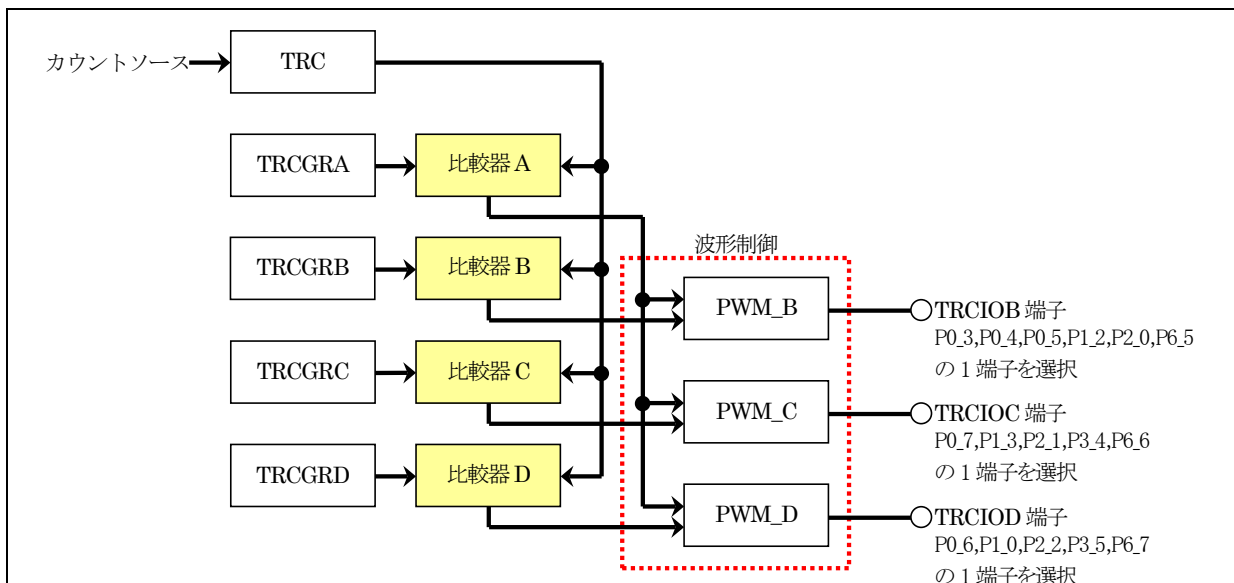
5.2.8 タイマ RC の設定

```

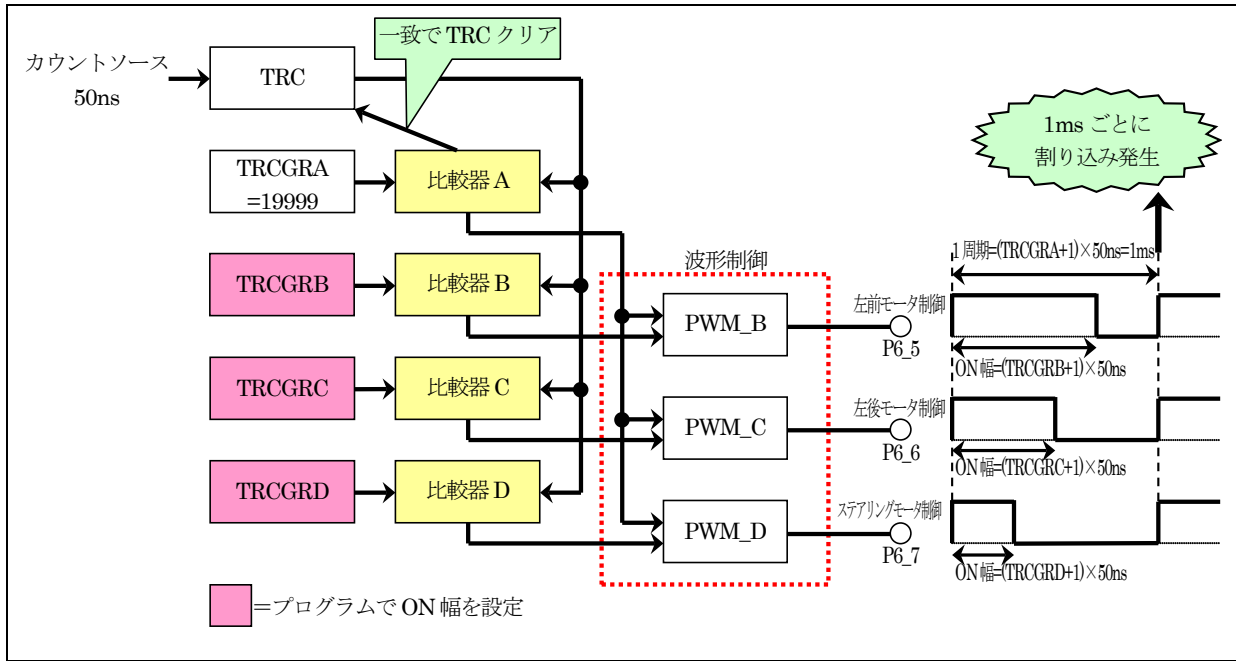
348 :      /* タイマRC PWMモード設定 (左前モータ、右前モータ、ステアリングモータ) */
349 :      trcpsr0 = 0x60;          /* TRCIOA端子=なし, B端子=P6_5 */
350 :      trcpsr1 = 0x55;          /* TRCIOC端子=P6_6, D端子=P6_7 */
351 :      trcmr = 0x0f;           /* PWMモード選択ビット設定 */
352 :      trccr1 = 0x8e;          /* ツーサウト:f1, 初期出力の設定 */
353 :      trccr2 = 0x00;          /* 出力レベルの設定 */
354 :      trecgra = TRC_MOTOR_CYCLE - 1; /* 周期設定 */
355 :      trecgrb = trecrb_buff = trecgra; /* P6_5端子のON幅 (右前モータ) */
356 :      trecgrc = trecrc_buff = trecgra; /* P6_6端子のON幅 (右後モータ) */
357 :      trecgrd = trecrd_buff = trecgra; /* P6_7端子のON幅 (ステアリングモータ) */
358 :      trcic = 0x07;           /* 割り込み優先レベル設定 */
359 :      trcier = 0x01;          /* IMIAを許可 */
360 :      trcoer = 0x01;          /* 出力端子の選択 */
361 :      trcmr |= 0x80;           /* TRCカウント開始 */
    
```

タイマ RC を使うと、同一周期の PWM 波形を 3 本出力することができます。ただし、ON 幅を設定するタイミングによっては、波形が乱れる場合がありますので、プログラムで対処が必要です。

タイマ RC を使った PWM 波形出力の代表的な様子を、下図に示します。



今回の設定をした PWM 波形が出力される様子を、下図に示します。



(1) タイマ RC 端子選択レジスタ 0 (TRCPSR0: Timer RC function select register 0) の設定

ビット	シンボル	説明	設定値
7	-	"0"を設定	0x60
6	trciobsel2	TRCIOB 端子選択ビットを設定します。 000: TRCIOB 端子は使用しない	
5	trciobsel1	010: P0_3 に割り当てる 011: P0_4 に割り当てる 100: P0_5 に割り当てる 101: P2_0 に割り当てる	
4	trciobsel0	110: P6_5 に割り当てる 上記以外: 設定しないでください	
3	-	"0"を設定	
2	trcioasel2	TRCIOA/TRCTRГ 端子選択ビットを設定します。 000: TRCIOA/TRCTRГ 端子は使用しない	
1	trcioasel1	001: P1_1 に割り当てる 010: P0_0 に割り当てる 011: P0_1 に割り当てる	
0	trcioasel0	100: P0_2 に割り当てる 上記以外: 設定しないでください	

5. プログラムの解説

(2) タイマ RC 端子選択レジスタ 1(TRCPSR1:Timer RC function select register 1)の設定

ビット	シンボル	説明	設定値
7	-	"0"を設定	0x55
6	trciodsel2	TRCIOD 端子選択ビットを設定します。 000:TRCIOD 端子は使用しない	
5	trciodsel1	001:P1_0 に割り当てる 010:P3_5 に割り当てる 011:P0_6 に割り当てる	
4	trciodsel0	100:P2_2 に割り当てる 101:P6_7 に割り当てる	
3	-	"0"を設定	
2	trciocsel2	TRCIOC 端子選択ビットを設定します。 000:TRCIOC 端子は使用しない	
1	trciocsel1	001:P1_3 に割り当てる 010:P3_4 に割り当てる 011:P0_7 に割り当てる	
0	trciocsel0	100:P2_1 に割り当てる 101:P6_6 に割り当てる	

(3) タイマ RC モードレジスタ(TRCMR:Timer RC mode register)の設定

ビット	シンボル	説明	設定値
7	tstart_trcmr	TRC カウント開始ビットを設定します。 0:カウント停止 1:カウント開始 カウント開始は最後にします。今回は"0"を設定します。	0x0f
6	-	"0"を設定	
5	bfd_trcmr	TRCGRD レジスタ機能選択ビットを設定します。 0:ジェネラルレジスタ 1:TRCGRB レジスタのバッファレジスタ	
4	bfc_trcmr	TRCGRC レジスタ機能選択ビットを設定します。 0:ジェネラルレジスタ 1:TRCGRA レジスタのバッファレジスタ	
3	pwm2_trcmr	PWM2 モード選択ビットを設定します。 0:PWM2 モード 1:タイマモードまたは PWM モード	
2	pwm_d_trcmr	TRCIOD PWM モード選択ビットを設定します。 0:タイマモード 1:PWM モード	
1	pwm_c_trcmr	TRCIOC PWM モード選択ビットを設定します。 0:タイマモード 1:PWM モード	
0	pwm_b_trcmr	TRCIOB PWM モード選択ビットを設定します。 0:タイマモード 1:PWM モード	

(4) タイマ RC 制御レジスタ 1(TRCCR1: Timer RC control register 1)の設定

ビット	シンボル	説明	設定値
7	cclr_trccr1	"1"を設定	0x8e
6	tck2_trccr1	カウントソース選択ビットを設定します。 000:f1 (1/20MHz=50ns)	
5	tck1_trccr1	001:f2 (2/20MHz=100ns) 010:f4 (4/20MHz=200ns) 011:f8 (8/20MHz=400ns) 100:f32 (32/20MHz=1600ns) 101:TRCCLK 入力の立ち上がりエッジ	
4	tck0_trccr1	110:fOCO40M (高速オンチップオシレータ 40MHz=今回は未接続) 111:fOCO-F (高速オンチップオシレータをFRA2 で分周したクロック=今回は未接続)	
3	tod_trccr1	TRCIOD 出力レベル選択ビットを設定します。 0:初期出力はアクティブでないレベル 1:初期出力はアクティブレベル	
2	toc_trccr1	TRCIOC 出力レベル選択ビットを設定します。 0:初期出力はアクティブでないレベル 1:初期出力はアクティブレベル	
1	tob_trccr1	TRCIOB 出力レベル選択ビットを設定します。 0:初期出力はアクティブでないレベル 1:初期出力はアクティブレベル	
0	toa_trccr1	"0"を設定	

(5) タイマ RC 制御レジスタ 2(TRCCR2: Timer RC control register 2)の設定

ビット	シンボル	説明	設定値
7	tceg1_trccr2	"0"を設定	0x00
6	tceg0_trccr2	"0"を設定	
5	cstp_trccr2	"0"を設定	
4	-	"0"を設定	
3	-	"0"を設定	
2	pold_trccr2	PWM モードアウトプットレベル制御ビット D を設定します。 0:TRCIOD の出力レベルは“L”アクティブ 1:TRCIOD の出力レベルは“H”アクティブ	
1	pole_trccr2	PWM モードアウトプットレベル制御ビット C を設定します。 0:TRCIOC の出力レベルは“L”アクティブ 1:TRCIOC の出力レベルは“H”アクティブ	
0	polb_trccr2	PWM モードアウトプットレベル制御ビット B を設定します。 0:TRCIOB の出力レベルは“L”アクティブ 1:TRCIOB の出力レベルは“H”アクティブ	

5. プログラムの解説

(6) タイマ RC ジェネラルレジスタ A(TRCGRA:Timer RC General register A)の設定

ビット	シンボル	説明	設定値
15~0	-	<p>TRCGRA は、PWM 波形の周期を設定します。計算式を下記に示します。</p> $\text{TRCGRA} = \text{PWM 波形の周期} / \text{タイマ RC カウンタのカウントソース} - 1$ <p>今回、タイマ RC の PWM 波形の周期は 1ms に設定します。タイマ RC カウンタのカウントソースは、TRCCR1 で設定した 50ns です。よって、</p> $\begin{aligned} \text{TRCGRA} &= (1 \times 10^{-3}) / (50 \times 10^{-9}) - 1 \\ &= 20000 - 1 \end{aligned}$ <p>となります。</p> <p>プログラムでは、define 文を使って</p> <pre> 28 : #define TRC_MOTOR_CYCLE 20000 /* 左前,右前モータ PWM の周期 */ 29 : /* 50[ns] * 20000 = 1.00[ms] */ とタイマ RC の PWM 波形の周期を記号定数で設定しています。よってプログラムでは、 354 : trcgra = TRC_MOTOR_CYCLE - 1; /* 周期設定 */ </pre> <p>としています。</p>	19999

(7) タイマ RC ジェネラルレジスタ B(TRCGRB:Timer RC General register B)の設定

ビット	シンボル	説明	設定値
15~0	-	<p>TRCGRB は、P6.5 端子の ON 幅を設定します。今回は、左前モータの PWM 波形の ON 幅を設定します。計算式を下記に示します。</p> $\text{TRCGRB} = \text{PWM 波形の ON 幅} / \text{タイマ RC カウンタのカウントソース} - 1$ <p>ただし、設定によっては下記のようになります。</p> <ul style="list-style-type: none"> ①TRCGRB の値を、TRCGRA と同じ値にすると ON 幅は 0%になる ②TRCGRB の値を、(TRCGRA+1)以上にすると ON 幅は 100%になる ③TRCGRB を設定するタイミングによっては、ON 幅 100%の波形が 1 周期出力されることがある <p>③は 1 周期ですがモータの回転が 100%になり問題です。設定するタイミングについては後述します。</p> <p>ここではまだ ON 幅は 0%にするので、①の TRCGRA と同じ値にします。</p>	19999

(8) タイマ RC ジェネラルレジスタ C(TRCGRC:Timer RC General register C)の設定

ビット	シンボル	説明	設定値
15~0	-	TRCGRC は、P6_6 端子の ON 幅を設定します。今回は、左後モータの PWM 波形の ON 幅を設定します。 設定内容は、TRCGRB と同じです。ここではまだ ON 幅は 0%にするので、TRCGRA と同じ値にします。	19999

(9) タイマ RC ジェネラルレジスタ D(TRCGRD:Timer RC General register D)の設定

ビット	シンボル	説明	設定値
15~0	-	TRCGRD は、P6_7 端子の ON 幅を設定します。今回は、ステアリングモータの PWM 波形の ON 幅を設定します。 設定内容は、TRCGRB と同じです。ここではまだ ON 幅は 0%にするので、TRCGRA と同じ値にします。	19999

(10) タイマ RC 割り込み制御レジスタ(TRCIC:Timer RC interrupt control register)の設定

ビット	シンボル	説明	設定値
7	-	"0"を設定	0x07
6	-	"0"を設定	
5	-	"0"を設定	
4	-	"0"を設定	
3	ir_trcic	"0"を設定	
2	ilvl2_trcic	他の割り込みが同時に発生した場合、どの割り込みを優先させるか設定します。レベルの高い割り込みが優先されます。割り込みを 2 つ以上使う場合、どれを優先させるかここで決めます。今回はタイマ RC 割り込みを 7、タイマ RD0 割り込みを 6、タイマ RB 割り込みを 5 に設定します。	
1	ilvl1_trcic	000:レベル 0 (割り込み禁止) 001:レベル 1 010:レベル 2 011:レベル 3	
0	ilvl0_trcic	100:レベル 4 101:レベル 5 110:レベル 6 111:レベル 7	

5. プログラムの解説

(11) タイマ RC 割り込み許可レジスタ(TRCIER: Timer RC Interrupt Enable Register)の設定

ビット	シンボル	説明	設定値
7	ovie_trcier	オーバフロー割り込み許可ビットを設定します。 0:OVF ビットによる割り込み(OVI)禁止 1:OVF ビットによる割り込み(OVI)許可	0x01
6	-	"0"を設定	
5	-	"0"を設定	
4	-	"0"を設定	
3	imied_trcier	インプットキャプチャ/コンペアー一致割り込み許可ビット D を設定します。 0:IMFD ビットによる割り込み(IMID)禁止 1:IMFD ビットによる割り込み(IMID)許可	
2	imiec_trcier	インプットキャプチャ/コンペアー一致割り込み許可ビット C を設定します。 0:IMFC ビットによる割り込み(IMIC)禁止 1:IMFC ビットによる割り込み(IMIC)許可	
1	imieb_trcier	インプットキャプチャ/コンペアー一致割り込み許可ビット B を設定します。 0:IMFB ビットによる割り込み(IMIB)禁止 1:IMFB ビットによる割り込み(IMIB)許可	
0	imiea_trcier	インプットキャプチャ/コンペアー一致割り込み許可ビット A を設定します。 0:IMFA ビットによる割り込み(IMIB)禁止 1:IMFA ビットによる割り込み(IMIB)許可 TRC と TRCGRA の値が一致したら、割り込みを発生させます。TRCGRA は PMW 波形の周期を設定しているため、今回は 1ms ごとに割り込みが発生することになります。 タイマ RC の割り込みプログラムでの処理内容は後述します。	

(12) タイマ RC アウトプットマスタ許可レジスタ(TRCOER: Timer RC output master enable register)の設定

ビット	シンボル	説明	設定値
7	pto_trcoer	"0"を設定	0x01
6	-	"0"を設定	
5	-	"0"を設定	
4	-	"0"を設定	
3	ed_trcoer	TRCIOD 出力禁止ビットを設定します。 0:出力許可(P6_7 端子を PWM 出力にする) 1:出力禁止(TRCIOD 端子はプログラマブル入出力ポート)	
2	ec_trcoer	TRCIOC 出力禁止ビットを設定します。 0:出力許可(P6_6 端子を PWM 出力にする) 1:出力禁止(TRCIOC 端子はプログラマブル入出力ポート)	
1	eb_trcoer	TRCIOB 出力禁止ビットを設定します。 0:出力許可(P6_5 端子を PWM 出力にする) 1:出力禁止(TRCIOB 端子はプログラマブル入出力ポート)	
0	ea_trcoer	TRCIOA 出力禁止ビットを設定します。 0:出力許可 1:出力禁止(TRCIOA 端子はプログラマブル入出力ポート)	

(13) タイマ RC モードレジスタ(TRCMR: Timer RC mode register)の設定

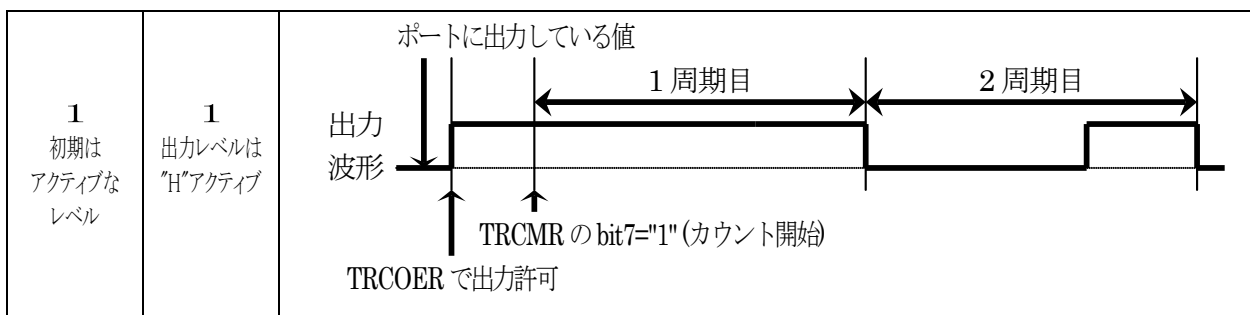
ビット	シンボル	説明	設定値
7	tstart_trcmr	TRC カウント開始ビットを設定します。 0:カウント停止 1:カウント開始	0x80 で OR
6	-	変更せず	
5	bfd_trcmr	変更せず	
4	bfc_trcmr	変更せず	
3	pwm2_trcmr	変更せず	
2	pwm_d_trcmr	変更せず	
1	pwm_c_trcmr	変更せず	
0	pwm_b_trcmr	変更せず	

(14) 出力されるタイミングと初期出力、アクティブレベルについて

タイマ RC 制御レジスタ 1(TRCCR1)の初期出力を設定するビット(bit3~1)と、タイマ RC 制御レジスタ 2(TRCCR2)のアクティブレベルを設定するビット(bit2~0)の関係を、下図に示します。

TRCCR1 bit3~1	TRCCR2 bit2~0	波形
0 初期は アクティブで ないレベル	0 出力レベルは "L"アクティブ	<p>ポートに出力している値</p> <p>出力波形</p> <p>1 周期目</p> <p>2 周期目</p> <p>TRCMR の bit7="1" (カウント開始)</p> <p>TRCOER で出力許可</p>
0 初期は アクティブで ないレベル	1 出力レベルは "H"アクティブ	<p>ポートに出力している値</p> <p>出力波形</p> <p>1 周期目</p> <p>2 周期目</p> <p>TRCMR の bit7="1" (カウント開始)</p> <p>TRCOER で出力許可</p>
1 初期は アクティブな レベル	0 出力レベルは "L"アクティブ	<p>ポートに出力している値</p> <p>出力波形</p> <p>1 周期目</p> <p>2 周期目</p> <p>TRCMR の bit7="1" (カウント開始)</p> <p>TRCOER で出力許可</p>

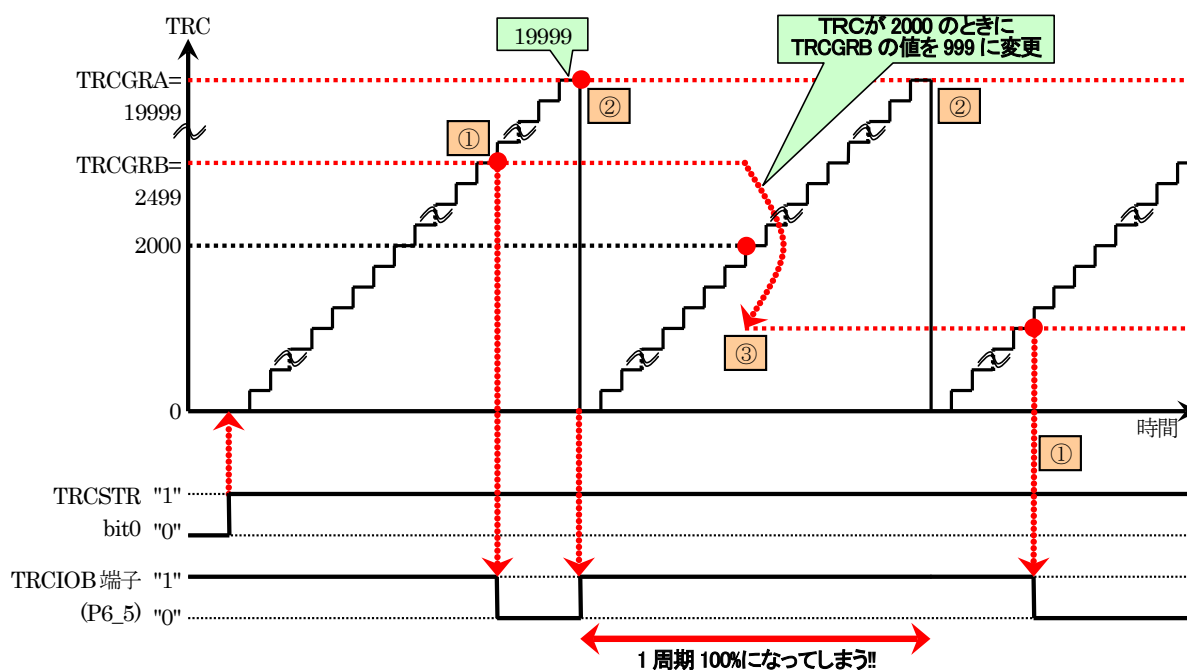
5. プログラムの解説



今回の設定は、「初期はアクティブなレベル、出力レベルは"L"アクティブ」にしています。

(15) PWM 波形

P6_5 端子の PWM 波形の様子を例に、各レジスタの値について下図に示します。



①	①TRC = TRCGRB+1 になった瞬間、出力波形は"0"になります。
②	②TRC = TRCGRA+1 になった瞬間、"1"になります。TRC はこのタイミングで 0 になります。PWM 波形は、①と②を繰り返すことにより、PWM 波形が出力されます。
③	ON 幅を変えるときは、プログラムで TRCGRB の値を変えます。例えば、③のように TRC の値が 2000 のとき、TRCGRB の値を 999 に変えた場合、①の波形が"0"になるタイミングがなく、1 周期 100%の波形が出力されてしまいます。

そこで、trcgrb_buffという変数を作り、プログラムではtrcgrb_buff変数の値を変更して、TRCGRBの値は直接変更しません。②のタイミングで割り込みが発生するようにして、割り込みプログラム内で trcgrb_buff 変数の値を、TRCGRB へ代入するようにします。

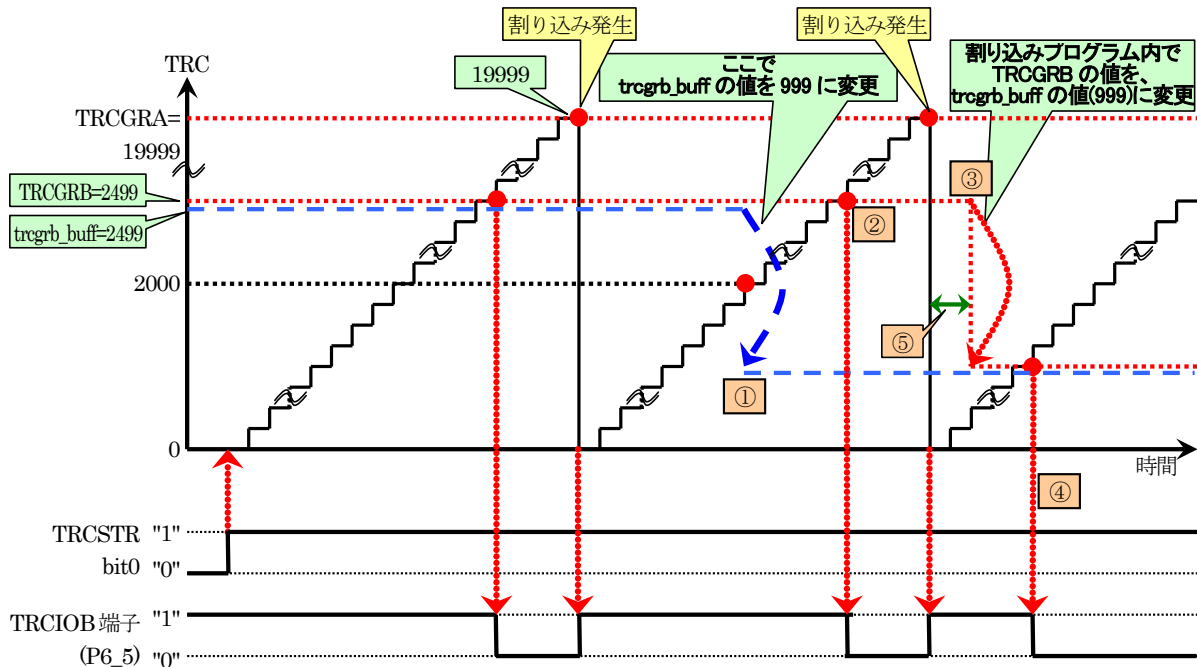
TRCGRC、TRCGRD も同様です。

割り込みプログラムを、下記に示します。

```

447 : /*****
448 : /* タイマRC 割り込み処理
449 : /*****
450 : #pragma interrupt /B intTRC(vect=7)
451 : void intTRC( void )
452 : {
453 :     /* タイマRC デューティ比の設定 */
454 :     trcgrd = trcgrd_buff;          /* ステアリングモータ PWMセット */
455 :     trcgrb = trcgrb_buff;          /* 右前モータ PWMセット */
456 :     trcgrc = trcgrc_buff;          /* 右後モータ PWMセット */
457 :
458 :     imfa_trcsr = 0;
459 : }
    
```

割り込みプログラムの処理を含めた P6_5 端子の PWM 波形の様子を、下図に示します。



①	プログラムでは TRCGRB の値ではなく、trcgrb_buff 変数の値を変えます。
②	TRCGRB の値は変わっていないため「TRC=TRCGRB+1」になり、波形は「0」になります。
③	タイマ RC の割り込みプログラム内で trcgrb_buff 変数を TRCGRB に代入します。
④	「TRC=TRCGRB+1」になり、波形は「0」になります。
⑤	割り込みは「TRC=TRCGRB+1」になった瞬間にかかりますが下記の理由により代入は若干遅れます。 ・現在実行している命令(アセンブリ言語レベル)が終わるまでに数百 ns~数 μs の時間がかかります。 ・割り込み関数を実行するまでに数 μ 秒の時間がかかります。 ・プログラムを実行するのに数百 ns~数 μs の時間がかかります。 例えば 452 行を実行する時点で TRC が 500、trcgrb_buff が 200 の場合、TRCGRB に 200 を代入しても、「TRC=TRCGRB+1」が起こらず 1 周期 100% の PWM 波形が出力されてしまいます。 そのため、trcgrb_buff 変数には 1000 以上(5%以上)の値を代入してください。0%にするときは、TRCGRB に TRCGRB の値を代入してください。ただし、ステアリングモータは、マイコンカー制御の要なので、400 以上(2%以上)として、ぎりぎりまで細かい値を設定できるようにしています。

5. プログラムの解説

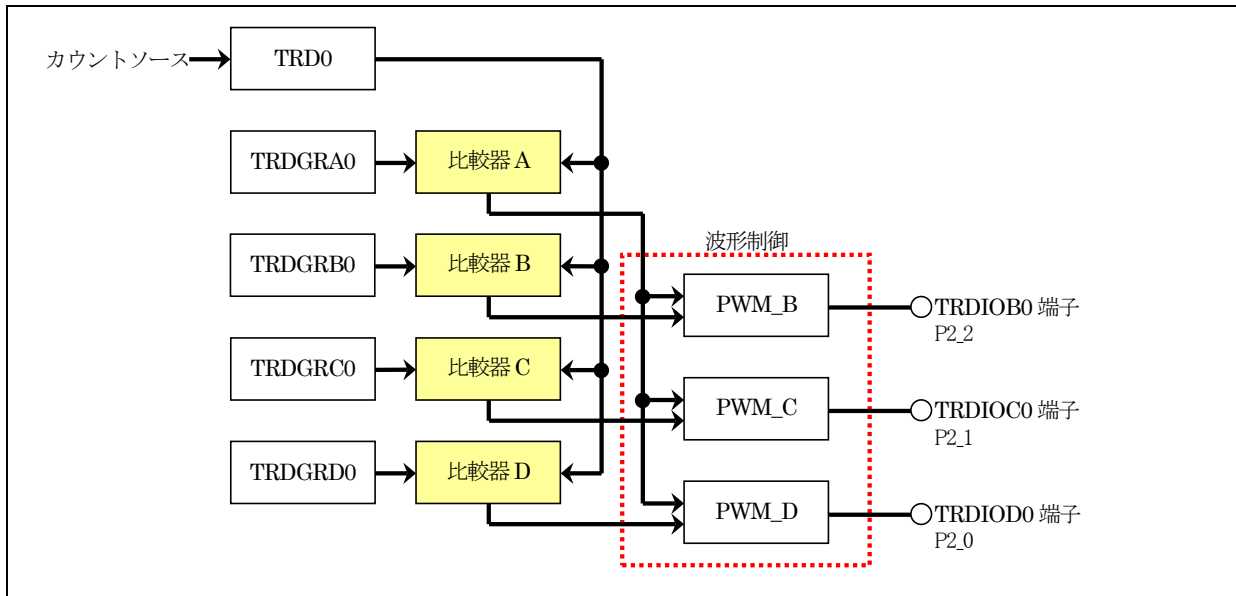
5.2.9 タイマ RD0 の設定

```

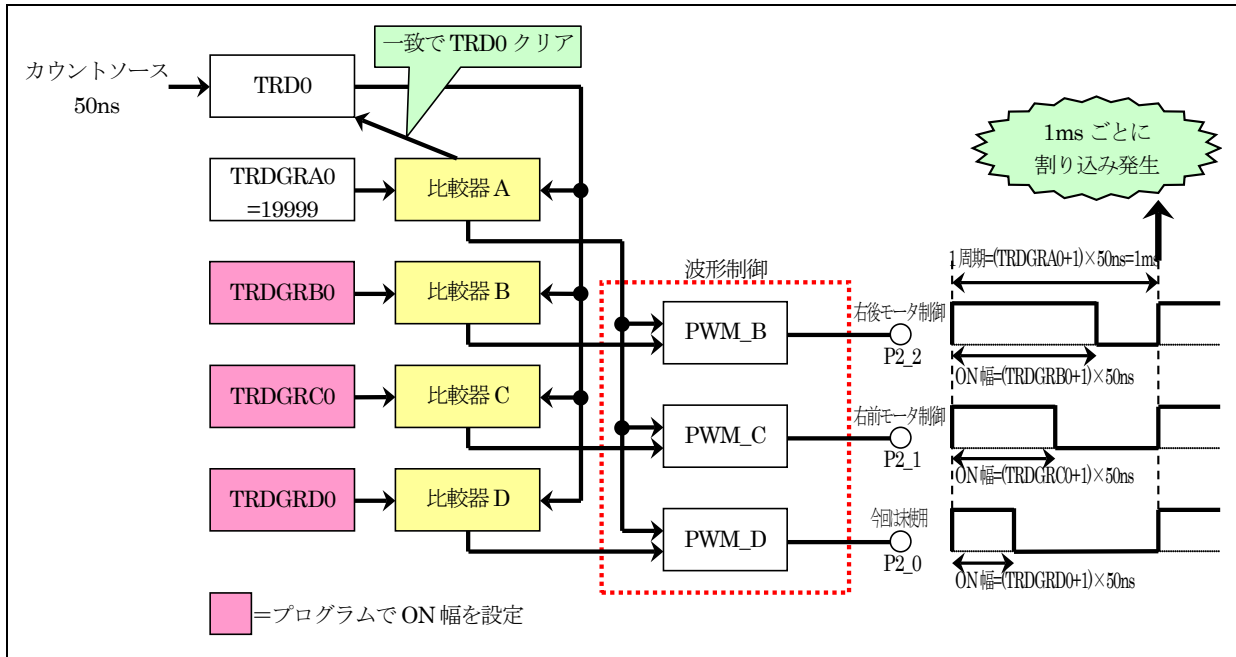
363 :      /* タイマRD0 リセット同期PWMモード設定(左後モータ、右後モータ) */
364 :      trdpsr0 = 0x28;          /* TRDIOB0=P2_2, C0=P2_1, D0=なし */
365 :      trdmr   = 0x00;          /* レジスタは独立動作 */
366 :      trdpmr   = 0x07;          /* PWM端子設定 B0, C0をPWM端子に */
367 :      trdfcr   = 0x80;          /* アウトプットコンパ機能に設定 */
368 :      trdcr0   = 0x20;          /* ソースカウントの選択:f1 */
369 :      trdgra0  = TRD_MOTOR_CYCLE - 1; /* 周期設定 */
370 :      trdgrb0  = trdgrb0_buff = trdgra0; /* P2_2端子のON幅(右後モータ) */
371 :      trdgrc0  = trdgrc0_buff = trdgra0; /* P2_1端子のON幅(右前モータ) */
372 :      imiea_trdier0 = 1;        /* IMFAビットによる割り込み許可 */
373 :      trd0ic   = 0x06;          /* 割り込み優先レベル設定 */
374 :      polb_trdpocr0 = 0;        /* TRDIOB0端子 出力レベルは"L"アクティブ */
375 :      polc_trdpocr0 = 0;        /* TRDIOC0端子 出力レベルは"L"アクティブ */
376 :      tob0_trdoocr = 1;        /* TRDIOB0端子 初期はアクティブレベル */
377 :      toc0_trdoocr = 1;        /* TRDIOC0端子 初期はアクティブレベル */
378 :      trdoerl   = 0xf9;        /* 出力端子の選択 */
379 :      tstart0_trdstr= 1;        /* TRD0カウント開始 */
    
```

タイマ RD0 を使うと、同一周期の PWM 波形を 3 本出力することができます。ただし、ON 幅を設定するタイミングによっては、波形が乱れる場合がありますので、プログラムで対処が必要です。

タイマ RD0 を使った PWM 波形出力の代表的な様子を、下図に示します。



今回の設定をした PWM 波形が出力される様子を、下図に示します。



(1) タイマ RD 端子選択レジスタ 0 (TRDPSR0: Timer RD function select register 0) の設定

ビット	シンボル	説明	設定値
7	-	"0"を設定	0x28
6	trdiob0sel0	TRDIOB0 端子選択ビットを設定します。 0: TRDIOB0 端子は使用しない 1: P2_3 に割り当てる	
5	trdioc0sel1	TRDIOC0 端子選択ビットを設定します。 00: TRDIOC0 端子は使用しない 01: 設定しないでください	
4	trdioc0sel0	10: P2_1 に割り当てる 11: 設定しないでください	
3	trdiob0sel1	TRDIOB0 端子選択ビットを設定します。 00: TRDIOB0 端子は使用しない 01: 設定しないでください	
2	trdiob0sel0	10: P2_2 に割り当てる 11: 設定しないでください	
1	-	"0"を設定	
0	trdiao0sel0	TRDIOA0/TRDCLK 端子選択ビットを設定します。 0: TRDIOA0/TRDCLK 端子は使用しない 1: P2_0 に割り当てる	

5. プログラムの解説

(2) タイマ RD モードレジスタ (TRDMR: Timer RD mode register) の設定

ビット	シンボル	説明	設定値
7	bfd1_trdmr	TRDGRD1 レジスタ機能選択ビットを設定します。 0: ジェネラルレジスタ 1: TRDGRB1 レジスタのバッファレジスタ	0x00
6	bfc1_trdmr	TRDGRC1 レジスタ機能選択ビットを設定します。 0: ジェネラルレジスタ 1: TRDGRA1 レジスタのバッファレジスタ	
5	bfd0_trdmr	TRDGRD0 レジスタ機能選択ビットを設定します。 0: ジェネラルレジスタ 1: TRDGRB0 レジスタのバッファレジスタ	
4	bfc0_trdmr	TRDGRC0 レジスタ機能選択ビットを設定します。 0: ジェネラルレジスタ 1: TRDGRA0 レジスタのバッファレジスタ	
3	-	"0"を設定	
2	-	"0"を設定	
1	-	"0"を設定	
0	sync_trdmr	タイマ RD 同期ビットを設定します。 0: TRD0 と TRD1 は独立動作 1: TRD0 と TRD1 は同期動作	

(3) タイマ RD PWM モードレジスタ (TRDPMR: Timer RD PWM mode register) の設定

ビット	シンボル	説明	設定値
7	-	"0"を設定	0x07
6	pwmd1_trdpmr	TRDIOD1 PWM モード選択ビットを設定します。 0: タイマモード 1: PWM モード	
5	pwmc1_trdpmr	TRDIOC1 PWM モード選択ビットを設定します。 0: タイマモード 1: PWM モード	
4	pwmb1_trdpmr	TRDIOB1 PWM モード選択ビットを設定します。 0: タイマモード 1: PWM モード	
3	-	"0"を設定	
2	pwmd0_trdpmr	TRDIOD0 PWM モード選択ビットを設定します。 0: タイマモード 1: PWM モード	
1	pwmc0_trdpmr	TRDIOC0 PWM モード選択ビットを設定します。 0: タイマモード 1: PWM モード	
0	pwmb0_trdpmr	TRDIOB0 PWM モード選択ビットを設定します。 0: タイマモード 1: PWM モード	

(4) タイマ RD 機能制御レジスタ(TRDFCR:Timer RD function control register)の設定

ビット	シンボル	説明	設定値
7	pwm3_trdfer	PWM3 モード選択ビットを設定します。 "1"を設定 ※"1"は PWM3 モードが無効になる設定です。	0x80
6	stclk_trdfer	外部クロック入力選択ビットを設定します。 0:外部クロック入力無効 1:外部クロック入力有効	
5	-	"0"を設定	
4	-	"0"を設定	
3	-	"0"を設定	
2	-	"0"を設定	
1	cmd1_trdfer	コンビネーションモード選択ビットを設定します。 PWM モードでは"00"(タイマモード、PWM モード、PWM3 モード)にしてください	
0	cmd0_trdfer		

(5) タイマ RD 制御レジスタ 0(TRDCR0:Timer RD control register 0)の設定

ビット	シンボル	説明	設定値
7	cclr2_trdcr0	TRD0 カウンタクリア選択ビットを設定します。 PWM モードの場合は、"001"(TRDGRA0 とのコンペアー一致で TRD0 レジスタクリア)に設定	0x20
6	cclr1_trdcr0		
5	cclr0_trdcr0		
4	ckeg1_trdcr0	外部クロックエッジ選択ビット(注 3)を設定します。 00:立ち上がりエッジでカウント 01:立ち下がりエッジでカウント 10:両エッジでカウント 11:設定しないでください ※今回は外部クロックを使わないので、"11"以外のどれかを設定します	
3	ckeg0_trdcr0		
2	tck2_trdcr0	カウントソース選択ビットを設定します。 000:f1 (1/20MHz=50ns) 001:f2 (2/20MHz=100ns) 010:f4 (4/20MHz=200ns) 011:f8 (8/20MHz=400ns) 100:f32 (32/20MHz=1600ns) 101:TRDCLK 入力(注 1)または fC2 (注 2) fC2 = 2/XCIN クロック=今回は未接続	
1	tck1_trdcr0		
0	tck0_trdcr0	110:fOCO40M (高速オンチップオシレータ 40MHz=今回は未接続) 111:fOCO-F(注 4) (高速オンチップオシレータを FRA2 で分周したクロック=今回は未接続)	

注 1. TRDECR レジスタの ITCLK0 ビットが"0"(TRDCLK 入力)かつ TRDFCR レジスタの STCLK ビットが"1"(外部クロック入力有効)のとき、有効です。

注 2. タイマモードで、TRDECR レジスタの ITCLK0 ビットが"1"(fC2)のとき有効です。

注 3. TCK2~TCK0 ビットが"101"(TRDCLK 入力または fC2)、TRDECR レジスタの ITCLK0 ビットが"0"(TRDCLK 入力)、かつ TRDFCR レジスタの STCLK ビットが"1"(外部クロック入力有効)のとき、有効です。

注 4. fOCO-F を選択するとき、CPU クロックより速いクロック周波数に fOCO-F を設定してください。

5. プログラムの解説

(6) タイマ RD ジェネラルレジスタ A0(TRDGRA0: Timer RD General register A0)の設定

ビット	シンボル	説明	設定値
15~0	-	<p>TRDGRA0 は、PWM 波形の周期を設定します。計算式を下記に示します。</p> $\text{TRDGRA0} = \text{PWM 波形の周期} / \text{タイマ RD0 カウンタのカウントソース} - 1$ <p>今回、タイマ RD0 の PWM 波形の周期は 1ms に設定します。タイマ RD0 カウンタのカウントソースは、TRDCR0 で設定した 50ns です。よって、</p> $\begin{aligned} \text{TRDGRA0} &= (1 \times 10^{-3}) / (50 \times 10^{-9}) - 1 \\ &= 20000 - 1 \end{aligned}$ <p>となります。</p> <p>プログラムでは、define 文を使って</p> <pre>30 : #define TRD_MOTOR_CYCLE 20000 /* 左後, 右後, サホモータPWMの周期 */ 31 :</pre> <p>とタイマ RD0 の PWM 波形の周期を記号定数で設定しています。よってプログラムでは、</p> <pre>369 : trdgra0 = TRD_MOTOR_CYCLE - 1; /* 周期設定 */</pre> <p>としています。</p>	19999

(7) タイマ RD ジェネラルレジスタ B0(TRDGRB0: Timer RD General register B0)の設定

ビット	シンボル	説明	設定値
15~0	-	<p>TRDGRB0 は、P2_2 端子の ON 幅を設定します。今回は、右後モータの PWM 波形の ON 幅を設定します。計算式を下記に示します。</p> $\text{TRDGRB0} = \text{PWM 波形の ON 幅} / \text{タイマ RC カウンタのカウントソース} - 1$ <p>ただし、設定によっては下記のようになります。</p> <ul style="list-style-type: none"> ①TRDGRB0 の値を、TRDGRA0 と同じ値にすると ON 幅は 0%になる ②TRDGRB0 の値を、(TRDGRA0+1)以上にすると ON 幅は 100%になる ③TRDGRB0 を設定するタイミングによっては、ON 幅 100%の波形が 1 周期出力されることがある <p>③は 1 周期ですがモータの回転が 100%になり問題です。設定するタイミングについては後述します。</p> <p>ここではまだ ON 幅は 0%にするので、①の TRDGRA0 と同じ値にします。</p>	19999

(8) タイマ RD ジェネラルレジスタ C0(TRDGRC0:Timer RD General register C0)の設定

ビット	シンボル	説明	設定値
15~0	-	TRDGRC0 は、P2_1 端子の ON 幅を設定します。今回は、右前モータの PWM 波形の ON 幅を設定します。 設定内容は、TRDGRB0と同じです。ここではまだ ON 幅は0%にするので、TRDGRA0と同じ値にします。	19999

(9) タイマ RD 割り込み許可レジスタ 0(TRDIER0:Timer RD interrupt enable register 0)の設定

ビット	シンボル	説明	設定値
7	-	"0"を設定	bit0="1"
6	-	"0"を設定	
5	-	"0"を設定	
4	ovie_trdier0	オーバフロー/アンダフロー割り込み許可ビットを設定します。 0:OVF ビットによる割り込み(OVI)禁止 1:OVF ビットによる割り込み(OVI)許可	
3	imied_trdier0	インプットキャプチャ/コンペアー一致割り込み許可ビット D を設定します。 0:IMFD ビットによる割り込み(IMID)禁止 1:IMFD ビットによる割り込み(IMID)許可	
2	imiec_trdier0	インプットキャプチャ/コンペアー一致割り込み許可ビット C を設定します。 0:IMFC ビットによる割り込み(IMIC)禁止 1:IMFC ビットによる割り込み(IMIC)許可	
1	imieb_trdier0	インプットキャプチャ/コンペアー一致割り込み許可ビット B を設定します。 0:IMFB ビットによる割り込み(IMIB)禁止 1:IMFB ビットによる割り込み(IMIB)許可	
0	imiea_trdier0	インプットキャプチャ/コンペアー一致割り込み許可ビット A を設定します。 0:IMFA ビットによる割り込み(IMIA)禁止 1:IMFA ビットによる割り込み(IMIA)許可	

(10) タイマ RD0 割り込み制御レジスタ(TRD0IC:Timer RD0 interrupt control register)の設定

ビット	シンボル	説明	設定値
7	-	"0"を設定	0x06
6	-	"0"を設定	
5	-	"0"を設定	
4	-	"0"を設定	
3	ir_trd0ic	"0"を設定	
2	ilvl2_trd0ic	他の割り込みが同時に発生した場合、どの割り込みを優先させるか設定します。レベルの高い割り込みが優先されます。割り込みを 2 つ以上使う場合、どれを優先させるかここで決めます。今回はタイマ RC 割り込みを 7、タイマ RD0 割り込みを 6、タイマ RB 割り込みを 5 に設定します。	
1	ilvl1_trd0ic	000:レベル 0 (割り込み禁止) 001:レベル 1 010:レベル 2 011:レベル 3	
0	iilvl0_trd0ic	100:レベル 4 101:レベル 5 110:レベル 6 111:レベル 7	

5. プログラムの解説

(11) タイマ RD PWM モードアウトプットレベル制御レジスタ 0(TRDPOCR0:Timer RD PWM mode output level control register 0)の設定

ビット	シンボル	説明	設定値
7	-	"0"を設定	bit1="0" bit0="0"
6	-	"0"を設定	
5	-	"0"を設定	
4	-	"0"を設定	
3	-	"0"を設定	
2	pold_trdpocr0	PWM モードアウトプットレベル制御ビット D を設定します。 0:TRDIOD0 の出力レベルは“L”アクティブ 1:TRDIOD0 の出力レベルは“H”アクティブ	
1	polc_trdpocr0	PWM モードアウトプットレベル制御ビット C を設定します。 0:TRDIOC0 の出力レベルは“L”アクティブ 1:TRDIOC0 の出力レベルは“H”アクティブ	
0	polb_trdpocr0	PWM モードアウトプットレベル制御ビット B を設定します。 0:TRDIOB0 の出力レベルは“L”アクティブ 1:TRDIOB0 の出力レベルは“H”アクティブ	

(12) タイマ RD アウトプット制御レジスタ(TRDOCR:Timer RD output control register)の設定

ビット	シンボル	説明	設定値
7	tod1_trdocr	TRDIOD1 初期出力レベル選択ビットを設定します。 0:初期出力はアクティブでないレベル 1:初期出力はアクティブレベル	bit2="1" bit1="1"
6	toc1_trdocr	TRDIOC1 初期出力レベル選択ビットを設定します。 0:初期出力はアクティブでないレベル 1:初期出力はアクティブレベル	
5	tob1_trdocr	TRDIOB1 初期出力レベル選択ビットを設定します。 0:初期出力はアクティブでないレベル 1:初期出力はアクティブレベル	
4	-	"0"を設定	
3	tod0_trdocr	TRDIOD0 初期出力レベル選択ビットを設定します。 0:初期出力はアクティブでないレベル 1:初期出力はアクティブレベル	
2	toc0_trdocr	TRDIOC0 初期出力レベル選択ビットを設定します。 0:初期出力はアクティブでないレベル 1:初期出力はアクティブレベル	
1	tob0_trdocr	TRDIOB0 初期出力レベル選択ビットを設定します。 0:初期出力はアクティブでないレベル 1:初期出力はアクティブレベル	
0	-	"0"を設定	

(13) タイマ RD アウトプットマスタ許可レジスタ 1(TRDOER1:Timer RD output master enable register 1)の設定

ビット	シンボル	説明	設定値
7	ed1_trdoer1	TRDIOD1 出力禁止ビットを設定します。 0:出力許可 1:出力禁止 (TRDIOD1 端子はプログラマブル入出力ポート)	0xf9
6	ec1_trdoer1	TRDIOC1 出力禁止ビットを設定します。 0:出力許可 1:出力禁止 (TRDIOC1 端子はプログラマブル入出力ポート)	
5	eb1_trdoer1	TRDIOB1 出力禁止ビットを設定します。 0:出力許可 1:出力禁止 (TRDIOB1 端子はプログラマブル入出力ポート)	
4	-	"1"を設定	
3	ed0_trdoer1	TRDIOD0 出力禁止ビットを設定します。 0:出力許可 1:出力禁止 (TRDIOD0 端子はプログラマブル入出力ポート)	
2	ec0_trdoer1	TRDIOC0 出力禁止ビットを設定します。 0:出力許可 1:出力禁止(TRDIOC0 端子はプログラマブル入出力ポート) TRDIOC0 を出力許可に設定すると、P2_1 端子から PWM 信号が出力されます。	
1	eb0_trdoer1	TRDIOB0 出力禁止ビットを設定します。 0:出力許可 1:出力禁止(TRDIOB0 端子はプログラマブル入出力ポート) TRDIOB0 を出力許可に設定すると、P2_2 端子から PWM 信号が出力されます。	
0	-	"1"を設定	

(14) タイマ RD スタートレジスタ(TRDSTR:Timer RD start register)の設定

ビット	シンボル	説明	設定値
7	-	"0"を設定	bit0="1"
6	-	"0"を設定	
5	-	"0"を設定	
4	-	"0"を設定	
3	csel1_trdstr	TRD1 カウント動作選択ビットを設定します。 0:TRDGRA1 レジスタとのコンペアー一致でカウント停止 1:TRDGRA1 レジスタとのコンペアー一致後もカウント継続	
2	csel0_trdstr	TRD0 カウント動作選択ビットを設定します。 0:TRDGRA0 レジスタとのコンペアー一致でカウント停止 1:TRDGRA0 レジスタとのコンペアー一致後もカウント継続	
1	tstart1_trdstr	TRD1 カウント開始フラグを設定します。 0:カウント停止 1:カウント開始	
0	tstart0_trdstr	TRD0 カウント開始フラグを設定します。 0:カウント停止 1:カウント開始	

5. プログラムの解説

(15) 出力されるタイミングと初期出力、アクティブレベルについて

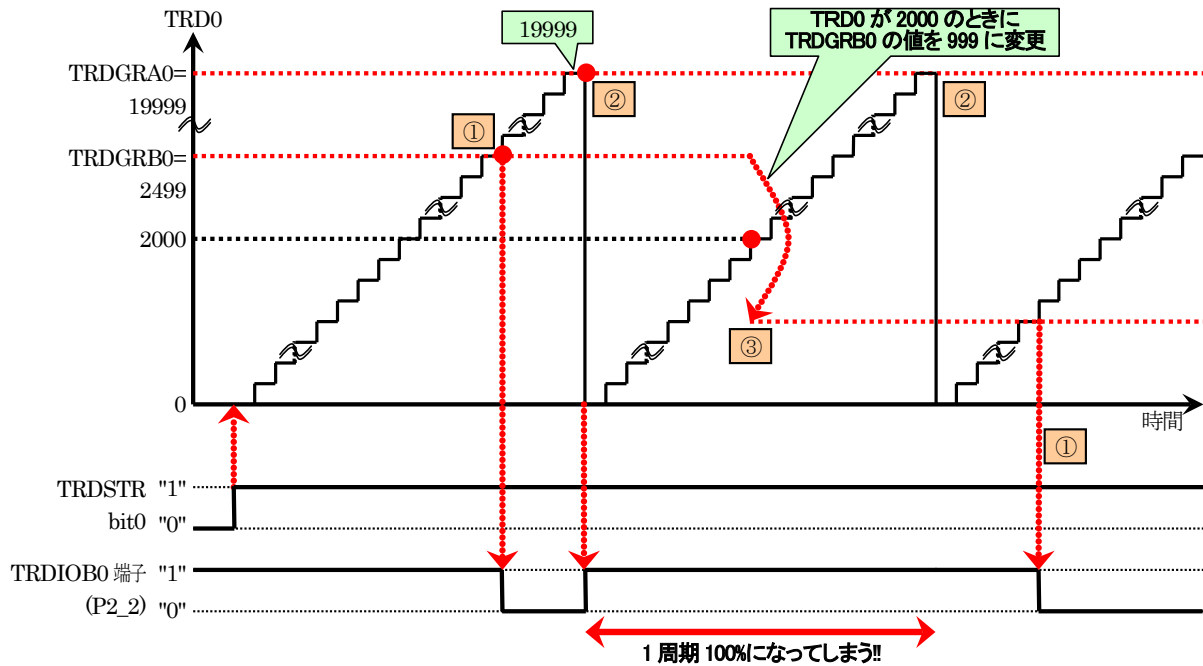
タイマ RD アウトプット制御レジスタ(TRDOCR)の初期出力を設定するビット(bit3~1)と、タイマ RD PWM モードアウトプットレベル制御レジスタ 0(TRDPOCR0)のアクティブレベルを設定するビット(bit2~0)の関係を、下図に示します。

TRDOCR bit3~1	TRDPOCR0 bit2~0	波形
0 初期は アクティブで ないレベル	0 出力レベルは "L"アクティブ	<p>ポートに出力している値</p> <p>出力波形</p> <p>1 周期目</p> <p>2 周期目</p> <p>TRDSTR の bit0="1" (カウント開始)</p> <p>TRDOER1 で出力許可</p>
0 初期は アクティブで ないレベル	1 出力レベルは "H"アクティブ	<p>ポートに出力している値</p> <p>出力波形</p> <p>1 周期目</p> <p>2 周期目</p> <p>TRDSTR の bit0="1" (カウント開始)</p> <p>TRDOER1 で出力許可</p>
1 初期は アクティブな レベル	0 出力レベルは "L"アクティブ	<p>ポートに出力している値</p> <p>出力波形</p> <p>1 周期目</p> <p>2 周期目</p> <p>TRDSTR の bit0="1" (カウント開始)</p> <p>TRDOER1 で出力許可</p>
1 初期は アクティブな レベル	1 出力レベルは "H"アクティブ	<p>ポートに出力している値</p> <p>出力波形</p> <p>1 周期目</p> <p>2 周期目</p> <p>TRDSTR の bit0="1" (カウント開始)</p> <p>TRDOER1 で出力許可</p>

今回の設定は、「初期はアクティブなレベル、出力レベルは"L"アクティブ」にしています。

(16) PWM 波形

P2_2 端子の PWM 波形の様子を例に、各レジスタの値について下図に示します。



①	①TRD0=TRDGRB0+1 になった瞬間、出力波形は"0"になります。
②	②TRD0=TRDGRA0+1 になった瞬間、"1"になります。TRD0 はこのタイミングで 0 になります。PWM 波形は、下記の①と②を繰り返すことにより、PWM 波形が出力されます。
③	ON 幅を変えるときは、プログラムで TRDGRB0 の値を変えます。このとき、図③のように TRDGRB0 の値を変えてしまった場合、①の波形が"0"になるタイミングがなく、1 周期 100%の波形が出力されてしまいます。

そこで、trdgrb0_buff という変数を作り、プログラムでは trdgrb0_buff 変数の値を変更して、TRDGRB0 の値は直接変更しません。②のタイミングで割り込みが発生するようにして、割り込みプログラム内で trdgrb0_buff 変数の値を、TRDGRB0 へ代入するようにします。

TRDGRC0 も同様です。

5. プログラムの解説

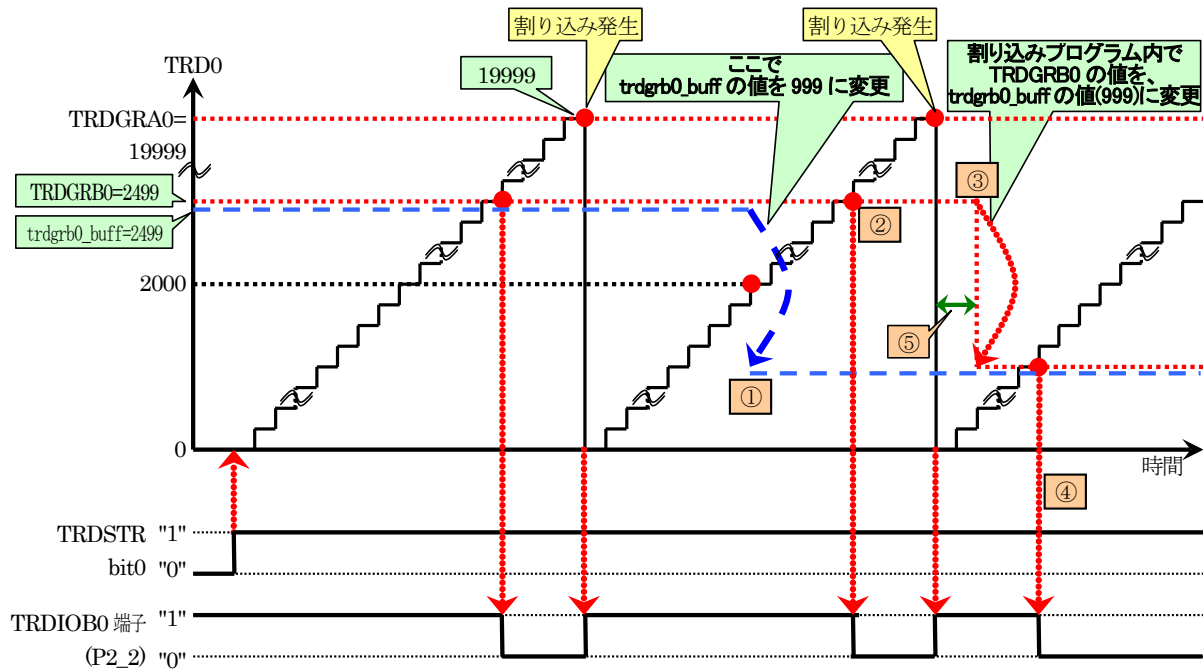
割り込みプログラムを、下記に示します。

```

459 : /*****
460 : /* タイマRD0 割り込み処理
461 : /*****
462 : #pragma interrupt intTRD0(vect=8)
463 : void intTRD0( void )
464 : {
465 :     asm(" fset I "); /* タイマRD0以上の割り込み許可 */
466 :
467 :     /* タイマRD0 デューティ比の設定 */
468 :     trdgrb0 = trdgrb0_buff; /* 左前モータ PWMセット */
469 :     trdgrc0 = trdgrc0_buff; /* 左後モータ PWMセット */
470 :
471 :     imfa_trdsr0 = 0;
472 : }
    
```

タイマ RD0 割り込み処理実行中に、タイマ RD0 割り込み処理より優先順位の高い割り込みが発生した場合、その割り込み処理に移ります。今回は、タイマ RC 割り込みが発生した場合、タイマ RC 割り込みに移ります。

割り込みプログラムの処理を含めた P2_2 端子の PWM 波形の様子を、下図に示します。



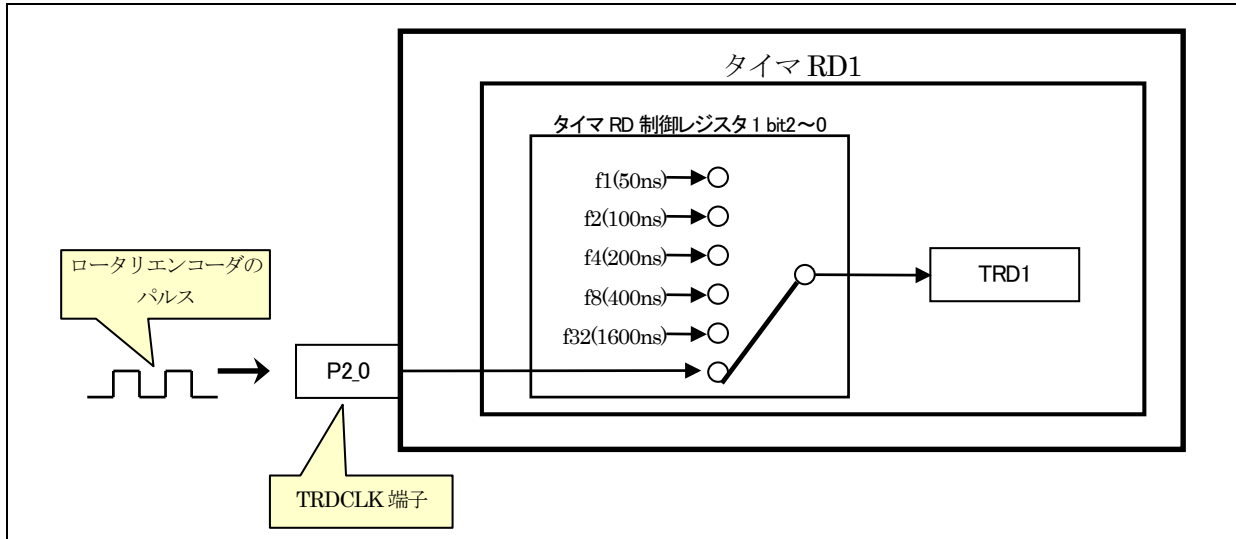
①	プログラムでは TRDGRB0 の値ではなく、trdgrb0_buff 変数の値を変えます。
②	TRDGRB0 の値は変わっていないため「TRD0=TRDCGRB0+1」になり、波形は「0」になります。
③	タイマ RD0 の割り込みプログラム内で trdgrb0_buff 変数を TRDGRB0 に代入します。
④	「TRD0=TRDGRB0+1」になり、波形は「0」になります。
⑤	<p>割り込みは「TRD0=TRDGRA0+1」になった瞬間にかかりますが下記の理由により代入は若干遅れます。</p> <ul style="list-style-type: none"> ・現在実行している命令(アセンブリ言語レベル)が終わるまでに数百 ns~数 μs の時間がかかります。 ・割り込み関数を実行するまでに数 μ 秒の時間がかかります。 ・プログラムを実行するのに数百 ns~数 μs の時間がかかります。 <p>例えば 468 行を実行する時点で TRD0 が 500、trdgrb0_buff が 200 の場合、TRDGRB0 に 200 を代入しても、「TRD0=TRDGRB0+1」が起こらず 1 周期 100% の PWM 波形が出力されてしまいます。そのため、trdgrb0_buff 変数には 1000 以上(5%以上)の値を代入してください。0%にするときは、TRDGRB0 に TRDGRA0 の値を代入してください。</p>

5.2.10 タイマ RD1 の設定

```

381 :      /* タイマRD1 外部入力 (ロータリエンコーダのパルスカウント) */
382 :      trdia0sel0 = 1;          /* TRDCLK端子:P2_0に設定 */
383 :      stclk_trdfcr = 1;       /* 外部クロック入力有効に設定 */
384 :      trdcr1 = 0x15;         /* TRDCLK端子 両エッジカウント */
385 :      tstart1_trdstr= 1;     /* TRD1カウント開始 */
    
```

タイマ RD1 を外部クロック入力にして、ロータリエンコーダのパルスを入力します。今回のタイマ RD1 の設定を、下図に示します。



(1) タイマ RD 端子選択レジスタ 0 (TRDPSR0: Timer RD function select register 0) の設定

ビット	シンボル	説明	設定値
7	-	"0"を設定	bit0="1"
6	trdioid0sel0	TRDIOD0 端子選択ビットを設定します。 0: TRDIOD0 端子は使用しない 1: P2_3 に割り当てる	
5	trdioc0sel1	TRDIOC0 端子選択ビットを設定します。 00: TRDIOC0 端子は使用しない 01: 設定しないでください	
4	trdioc0sel0	10: P2_1 に割り当てる 11: 設定しないでください	
3	trdiob0sel1	TRDIOB0 端子選択ビットを設定します。 00: TRDIOB0 端子は使用しない 01: 設定しないでください	
2	trdiob0sel0	10: P2_2 に割り当てる 11: 設定しないでください	
1	-	"0"を設定	
0	trdia0sel0	TRDIOA0/TRDCLK 端子選択ビットを設定します。 0: TRDIOA0/TRDCLK 端子は使用しない 1: P2_0 に割り当てる ここでは、P2_0 端子を TRDIOA0 出力端子 / TRDCLK 入力端子します。どちらの機能にするかは、この後で設定します。	

5. プログラムの解説

(2) タイマ RD 機能制御レジスタ(TRDFCR:Timer RD function control register)の設定

ビット	シンボル	説明	設定値
7	pwm3_trdfcr	PWM3 モード選択ビットを設定します。 "1"を設定 ※"1"は PWM3 モードが無効になる設定です。	bit6="1"
6	stclk_trdfcr	外部クロック入力選択ビットを設定します。 0:外部クロック入力無効 1:外部クロック入力有効 この設定で、P2_0 端子をタイマ RD の外部クロック入力端子にします。	
5	-	"0"を設定	
4	-	"0"を設定	
3	-	"0"を設定	
2	-	"0"を設定	
1	cmd1_trdfcr	コンビネーションモード選択ビットを設定します。	
0	cmd0_trdfcr	PWM モードでは"00"(タイマモード、PWM モード、PWM3 モード)にしてください	

(3) タイマ RD 制御レジスタ 1(TRDCR1:Timer RD control register 1)の設定

ビット	シンボル	説明	設定値
7	cclr2_trdcr1	TRD1 カウンタクリア選択ビットを設定します。 000:クリア禁止(フリーランニング動作)	0x15
6	cclr1_trdcr1	001: TRDGRAi のコンペア一致でクリア 010: TRDGRBi のコンペア一致でクリア 011: 同期クリア(他のタイマ RDi のカウンタと同時にクリア)(注 4) 100: 設定しないでください	
		101: TRDGRCi のコンペア一致でクリア 110: TRDGRDi のコンペア一致でクリア 111: 設定しないでください	
4	ckeg1_trdcr0	外部クロックエッジ選択ビット(注 3)を設定します。 00: 立ち上がりエッジでカウント 01: 立ち下がりエッジでカウント 10: 両エッジでカウント 11: 設定しないでください	
3	ckeg0_trdcr0	今回は、ロータリエンコーダのパルスを立ち上がり、立ち下がりの両方でカウントするようにします。	
2	tck2_trdcr0	カウントソース選択ビットを設定します。 000: f1 (1/20MHz=50ns) 001: f2 (2/20MHz=100ns) 010: f4 (4/20MHz=200ns)	
1	tck1_trdcr0	011: f8 (8/20MHz=400ns) 100: f32 (32/20MHz=1600ns) 101: TRDCLK 入力(注 1)または fC2 (注 2) fC2 = 2/XCIN クロック=今回は未接続	
0	tck0_trdcr0	110: fOCO40M (高速オンチップオシレータ 40MHz=今回は未接続) 111: fOCO-F(注 4) (高速オンチップオシレータを FRA2 で分周したクロック=今回は未接続)	

- 注 1. TRDECR レジスタの ITCLK0 ビットが“0”(TRDCLK 入力)かつ TRDFCR レジスタの STCLK ビットが“1”(外部クロック入力有効)のとき、有効です。
- 注 2. タイマモードで、TRDECR レジスタの ITCLK0 ビットが“1”(fC2)のとき有効です。
- 注 3. TCK2～TCK0 ビットが“101”(TRDCLK 入力または fC2)、TRDECR レジスタの ITCLK0 ビットが“0”(TRDCLK 入力)、かつ TRDFCR レジスタの STCLK ビットが“1”(外部クロック入力有効)のとき、有効です。
- 注 4. fOCO-F を選択するとき、CPU クロックより速いクロック周波数に fOCO-F を設定してください。

(4) タイマ RD スタートレジスタ(TRDSTR: Timer RD start register)の設定

ビット	シンボル	説明	設定値
7	-	"0"を設定	bit1="1"
6	-	"0"を設定	
5	-	"0"を設定	
4	-	"0"を設定	
3	cse11_trdstr	TRD1 カウント動作選択ビットを設定します。 0:TRDGRA1 レジスタとのコンペア一致でカウント停止 1:TRDGRA1 レジスタとのコンペア一致後もカウント継続	
2	cse10_trdstr	TRD0 カウント動作選択ビットを設定します。 0:TRDGRA0 レジスタとのコンペア一致でカウント停止 1:TRDGRA0 レジスタとのコンペア一致後もカウント継続	
1	tstart1_trdstr	TRD1 カウント開始フラグを設定します。 0:カウント停止 1:カウント開始	
0	tstart0_trdstr	TRD0 カウント開始フラグを設定します。 0:カウント停止 1:カウント開始	

5. プログラムの解説

5.2.11 タイマ RB の 1ms ごとの割り込みプログラム

```

388 : /*****
389 : /* タイマRB 割り込み処理 */
390 : *****/
391 : #pragma interrupt intTRB(vect=24)
392 : void intTRB( void )
393 : {
394 :     unsigned int i;
395 :
396 :     asm(" fset I ");          /* タイマRB以上の割り込み許可 */
397 :
398 :     cnt1++;
399 :
400 :     /* サーボモータ制御 */
401 :     servoControl();

```

396 行	<p>タイマ RB 割り込み以上のレベルの割り込みがあったときに、割り込み処理に移るよう割り込みを許可しておきます。割り込みプログラムが実行された時点で NMI 割り込み以外の割り込みが禁止されるので、割り込みプログラム内で割り込みを許可する場合は、改めて割り込みを許可しておかなければ行けません。</p> <p>今回は、タイマ RD0 割り込み、タイマ RC 割り込みが発生した場合、それぞれの割り込み処理に移ります。</p>
398 行	cnt1 変数の値を 1 つ増やします。よって、cnt1 は 1ms ごとに増加していきます。
401 行	ステアリングモータの PWM 値を計算します。

```

403 :     /* 10回中1回実行する処理 */
404 :     iTimer10++;
405 :     switch( iTimer10 ) {
406 :     case 1:
407 :         /* エンコーダ制御 */
408 :         i = trd1;
409 :         iEncoder      = i - uEncoderBuff;
410 :         lEncoderTotal += iEncoder;
411 :         uEncoderBuff  = i;
412 :         break;
中略
438 :     case 10:
439 :         /* iTimer10変数の処理 */
440 :         iTimer10 = 0;
441 :         break;
442 :     }

```

404 行	iTimer10 変数の値を 1 つ増やします。
405～ 442 行	iTimer10 変数の値に応じて分岐させます。case 文は 1～10 まであり iTimer10 変数は 1ms ごとに増えていくので、10ms ごとに 1 回 case 文が順番に実行されることになります。
440 行	iTimer10 変数を 0 にして、次の 1ms 後の割り込みで case 1 の部分が実行されるようにします。

ロータリエンコーダに関する変数処理です。

タイマ RD1 カウンタ(TRD1)の値が、ロータリエンコーダの積算パルス数です。範囲は 0~65535 で、65535 の次は 0 になります。

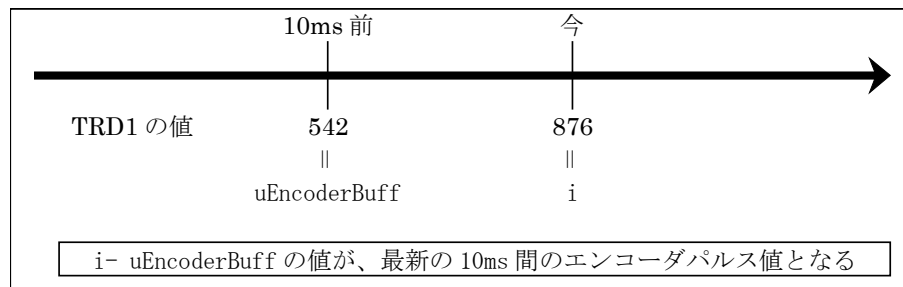
iEncoder..... 10ms 間のロータリエンコーダのパルス数を入力します。

lEncoderTotal.. ロータリエンコーダの積算値を入力します。long 型なので約 21 億パルスまでカウントできます。

iEncoder 値は、次の計算で求めます。

$$iEncoder = \text{現在の TRD1 の値}(i \text{ 変数}) - 10\text{ms 前の TRD1 の値}(uEncoderBuff \text{ 変数})$$

計算の様子を、下図に示します。



lEncoderTotal 変数は、10ms ごとに iEncoder の値を加えていきます。

408~
411 行

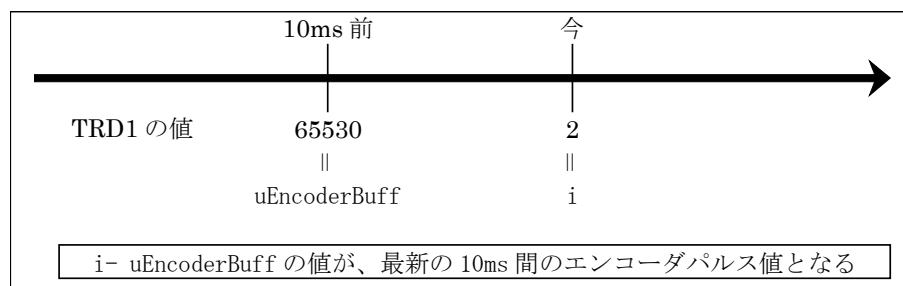
408 行で、現在の TRD1 の値(i 変数)を、uEncoderBuff 変数に代入します。10ms 後は、uEncoderBuff 変数の値が、10ms 前の値になっています。

今回、i 変数に TRD1 の値を一度コピーしてから 10ms 前の値の差分を計算して、iEncoder に代入しています。下記のプログラムのように、TRD1 の値を直接使った方が分かりやすいですが、そうしていません。

```
1 : iEncoder = trd1;
2 : trd1 = 0;
3 : lEncoderTotal += iEncoder;
```

これは、1 行目が終わってから 2 行目の TRD1 の値をクリアするまでのわずかな間に、パルスがカウントされてしまった場合、カウント洩れが起きてしまうためです。それを防ぐためにちょっと複雑ですが、今回のようなプログラムにしています。

TRD1 の上限である、65535 を超えた場合はどうなるのでしょうか。



$$iEncoder = \text{現在の TRG の値}(i \text{ 変数}) - 10\text{ms 前の TRG の値}(uEncoderBuff \text{ 変数})$$

$$iEncoder = 2 - 65530 = -62228$$

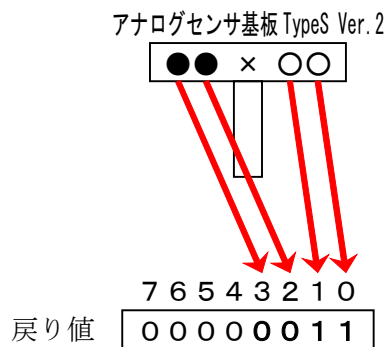
この値を、16 進数で表すと 0xffff0008 となります。変数の型は、符号無し 16bit 幅なので、下位の 16bit のみ有効となり 0x0008=8 となります。カウント回数としては、「①65531、②65532、③65533、④65534、⑤65535、⑥0、⑦1、⑧2」の 8 カウント分となり、計算結果と一致します。

5.2.12 アナログセンサ基板 TypeS Ver.2 のデジタルセンサ値読み込み

```

474 : /*****/
475 : /* アナログセンサ基板TypeS Ver.2のデジタルセンサ値読み込み */
476 : /* 引数 なし */
477 : /* 戻り値 左端、左中、右中、右端のデジタルセンサ 0:黒 1:白 */
478 : /*****/
479 : unsigned char sensor_inp( void )
480 : {
481 :     unsigned char sensor;
482 :
483 :     sensor = (p2_7<<3) | (p2_6<<2) | (p2_5<<1) | p0_7;
484 :     sensor = ~sensor;
485 :     sensor &= 0x0f;
486 :
487 :     return sensor;
488 : }
    
```

アナログセンサ基板 TypeS Ver.2 のデジタルセンサ 4 個を読み込む関数です。デジタルセンサは黒で"1"、白で"0"なのでポートから読み込むときに"~"(チルダ)をつけて反転させます。



※中心のデジタルセンサ値を読み込む関数は、center_inp 関数です。

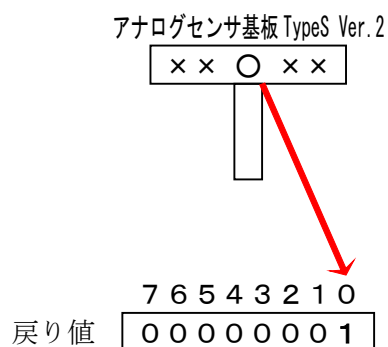
5.2.13 アナログセンサ基板 TypeS Ver.2 の中心デジタルセンサ読み込み

```

490 : /*****/
491 : /* アナログセンサ基板TypeS Ver.2の中心デジタルセンサ読み込み */
492 : /* 引数 なし */
493 : /* 戻り値 中心デジタルセンサ 0:黒 1:白 */
494 : /*****/
495 : unsigned char center_inp( void )
496 : {
497 :     unsigned char sensor;
498 :
499 :     sensor = ~p0_4 & 0x01;
500 :
501 :     return sensor;
502 : }

```

アナログセンサ基板 TypeS Ver.2 の中心デジタルセンサ 1 個を読み込む関数です。



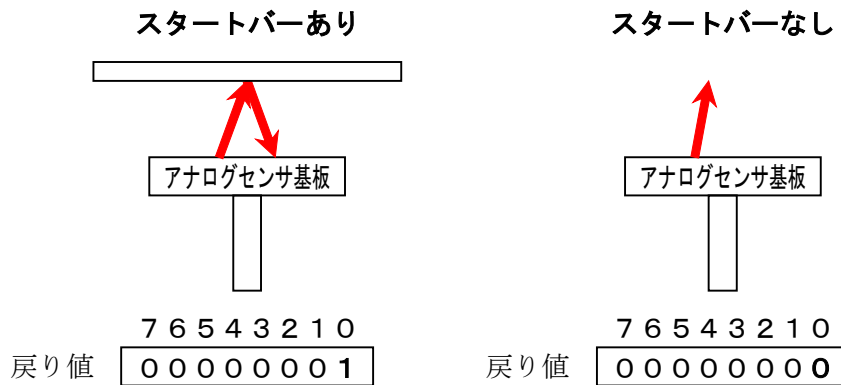
※中心以外のデジタルセンサ値を読み込む関数は、sensor_inp 関数です。

5.2.14 アナログセンサ基板 TypeS Ver.2 のスタートバー検出センサ読み込み

```

504 : /*****/
505 : /* アナログセンサ基板TypeS Ver.2のスタートバー検出センサ読み込み */
506 : /* 引数 なし */
507 : /* 戻り値 0:スタートバーなし 1:スタートバーあり */
508 : /*****/
509 : unsigned char startbar_get( void )
510 : {
511 :     unsigned char sensor;
512 :
513 :     sensor = ~p0_3 & 0x01;
514 :
515 :     return sensor;
516 : }
    
```

アナログセンサ基板 TypeS Ver.2 のスタートバー検出センサの状態を読み込む関数です。



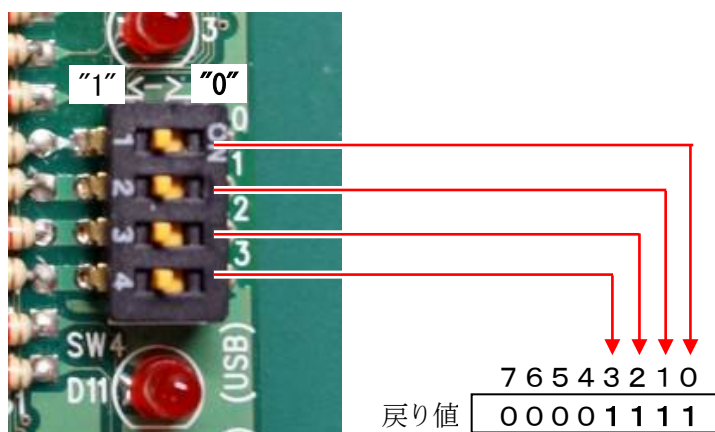
5.2.15 RMC-R8C35A マイコンボード上のディップスイッチ値読み込み

```

518 : /*****/
519 : /* マイコンボード上のディップスイッチ値読み込み */
520 : /* 引数 なし */
521 : /* 戻り値 スイッチ値 0~15 */
522 : /*****/
523 : unsigned char dipsw_get( void )
524 : {
525 :     unsigned char sw;
526 :
527 :     sw = (p5_7<<3) | (p4_5<<2) | (p4_4<<1) | p4_3;
528 :
529 :     return sw;
530 : }

```

RMC-R8C35A マイコンボードのディップスイッチの値を読み込む関数です。戻り値は、0~15 (2^4-1)です。

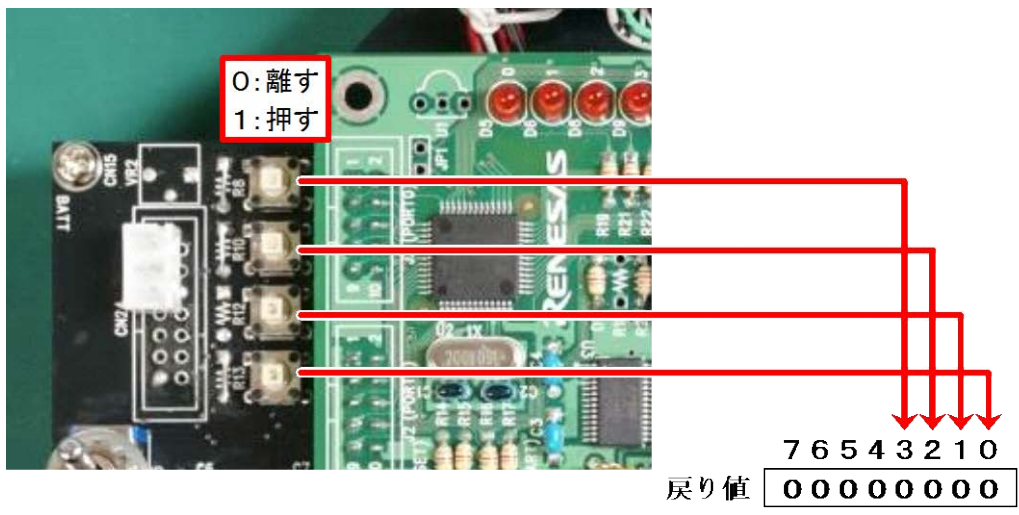


5.2.16 基板マイコンカーVer.2 上のプッシュスイッチ値読み込み

```

532 : /*****/
533 : /* 基板マイコンカー上のプッシュスイッチ値読み込み (SW1~4) */
534 : /* 引数 なし */
535 : /* 戻り値 スイッチ値 bit3:SW1 bit2:SW2 bit1:SW3 bit0:SW4 0:OFF 1:ON */
536 : /*****/
537 : unsigned char pushsw_get( void )
538 : {
539 :     unsigned char sw;
540 :
541 :     sw = (p6_4<<3) | (p0_0<<2) | (p6_3<<1) | p2_3;
542 :     sw = ~sw;
543 :     sw &= 0x0f;
544 :
545 :     return sw;
546 : }
    
```

基板マイコンカーVer.2 上のプッシュスイッチの値を読み込む関数です。

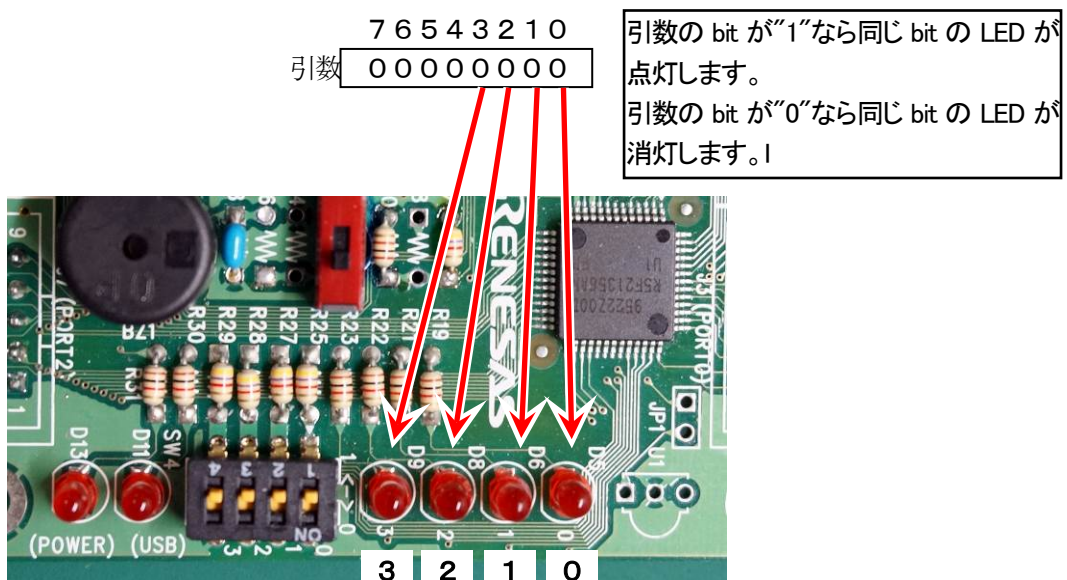


5.2.17 RMC-R8C35A マイコンボード上の LED 制御

```

548 : /*****/
549 : /* マイコンボード上のLED制御 */
550 : /* 引数 4個のLED制御 0:OFF 1:ON */
551 : /* 戻り値 なし */
552 : /*****/
553 : void led_out( unsigned char led )
554 : {
555 :     unsigned char d;
556 :
557 :     d = p1 & 0xf0;
558 :     p1 = d | (~led & 0x0f);
559 : }
    
```

RMC-R8C35A マイコンボード上の LED を 4 個、制御する関数です。



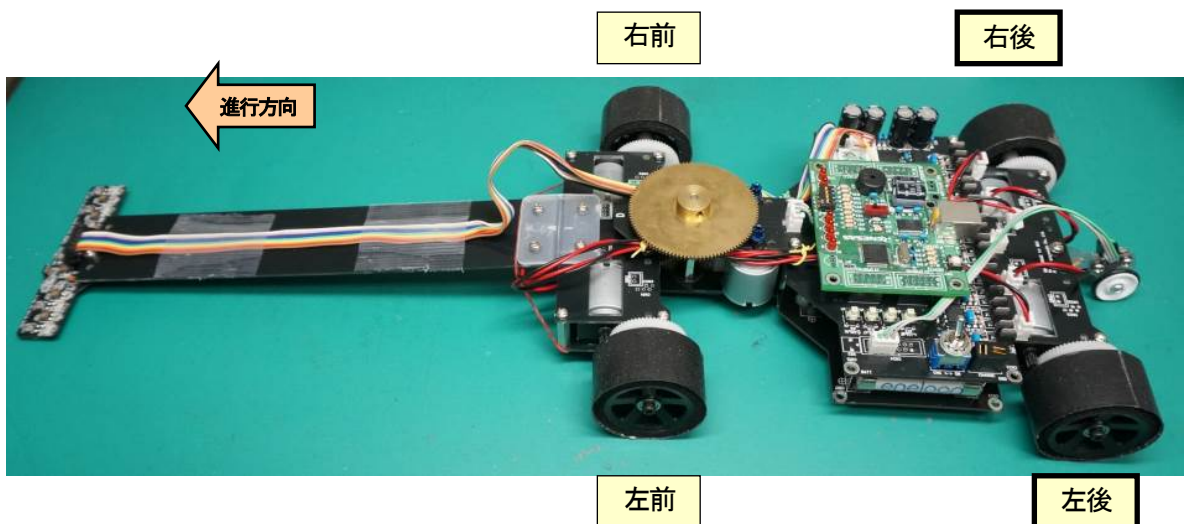
5. プログラムの解説

5.2.18 後輪の速度制御(ディップスイッチによる PWM 減速あり)

```

561 : /*****/
562 : /* 後輪の速度制御 */
563 : /* 引数 左モータ:-100~100 , 右モータ:-100~100 */
564 : /*      0で停止、100で正転100%、-100で逆転100% */
565 : /* 戻り値 なし */
566 : /*****/
567 : void motor_r( int accele_l, int accele_r )
568 : {
569 :     int sw_data;
570 :
571 :     sw_data = dipsw_get() + 5;          /* ディップスイッチ読み込み */
572 :     accele_l = accele_l * sw_data / 20;
573 :     accele_r = accele_r * sw_data / 20;
574 :
575 :     motor2_r( accele_l, accele_r ); motor2_r関数については次の説明を参照してください
576 : }
    
```

基板マイコンカーVer.2 の後輪モータ 2 個を制御する関数です。



使い方を下記に示します。

```

motor_r ( 左後モータの PWM, 右後モータの PWM );
    
```

左後モータの PWM、右後モータの PWM の値は、下記を設定することができます。

- 0… 停止
- 1～ 100… 正転の割合 100 が一番速い
- 1～-100… 逆転の割合 100 が一番速い

モータへの出力は、motor_r 関数で設定した割合がそのままモータに出力されるのではなく、下記の計算結果がモータに出力されます。

実際のモータに出力される割合 = 引数 × (マイコンボードのディップスイッチの値 + 5) ÷ 20

例えば、ディップスイッチの値が 10 で下記プログラムを実行したとします。

```
motor_r( 50, 100 );
```

左後モータに出力される割合 = プログラムの割合 × (ディップスイッチの値 + 5) ÷ 20
 = 50 × (10 + 5) ÷ 20
 = 37.5
 ≒ **37** (小数点以下は切り捨てです)

右後モータに出力される割合 = プログラムの割合 × (ディップスイッチの値 + 5) ÷ 20
 = 100 × (10 + 5) ÷ 20
 = **75**

				10 進数	計算	実際に出力される PWM の割合
bit3	bit2	bit1	bit0			
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%

5.2.19 後輪の速度制御 2(ディップスイッチには関係しない motor 関数)

```
578 : /*****/
579 : /* 後輪の速度制御2 ディップスイッチには関係しないmotor関数 */
580 : /* 引数 左モータ:-100~100 , 右モータ:-100~100 */
581 : /*      0で停止、100で正転100%、-100で逆転100% */
582 : /* 戻り値 なし */
583 : /* メモ 1~4%は、5%になります */
584 : /*****/
585 : void motor2_r( int accele_l, int accele_r )
586 : {
587 :     /* 左後モータ */
588 :     if( accele_l >= 0 ) {
589 :         p3_6 = 0;
590 :     } else {
591 :         p3_6 = 1;
592 :         accele_l = -accele_l;
593 :     }
594 :     if( accele_l == 0 ) {
595 :         // 0%のとき
596 :         trcgrc = trcgrc_buff = trcgra;
597 :     } else if( accele_l <= 5 ) {
598 :         // 1~5%のときは、プログラムの仕様で5%とする
599 :         trcgrc_buff = (unsigned long)(TRC_MOTOR_CYCLE-1) * 5 / 100;
600 :     } else if( accele_l <= 99 ) {
601 :         // 6~99%のとき
602 :         trcgrc_buff = (unsigned long)(TRC_MOTOR_CYCLE-1) * accele_l / 100;
603 :     } else {
604 :         // 100%のとき(実際は99%を設定)
605 :         trcgrc_buff = (unsigned long)(TRC_MOTOR_CYCLE-1) * 99 / 100;
606 :     }
607 :
608 :     /* 右後モータ */
609 :     if( accele_r >= 0 ) {
610 :         p3_0 = 0;
611 :     } else {
612 :         p3_0 = 1;
613 :         accele_r = -accele_r;
614 :     }
615 :     if( accele_r == 0 ) {
616 :         // 0%のとき
617 :         trdgrb0 = trdgrb0_buff = trdgra0;
618 :     } else if( accele_r <= 5 ) {
619 :         // 1~5%のときは、プログラムの仕様で5%とする
620 :         trdgrb0_buff = (unsigned long)(TRD_MOTOR_CYCLE-1) * 5 / 100;
621 :     } else if( accele_r <= 99 ) {
622 :         // 6~99%のとき
623 :         trdgrb0_buff = (unsigned long)(TRD_MOTOR_CYCLE-1) * accele_r / 100;
624 :     } else {
625 :         // 100%のとき(実際は99%を設定)
626 :         trdgrb0_buff = (unsigned long)(TRD_MOTOR_CYCLE-1) * 99 / 100;
627 :     }
628 : }
```

motor_r 関数は、ディップスイッチの設定でモータに出力する割合をさらに落としましたが、motor2_r 関数は、プロ

グラムの引数どおりの割合をモータに出力します。右後モータのプログラムを解説します。

609行～614行	<p>右後モータに設定するPWM値がプラスなら、P3_0端子を"0"にして右後モータの回転を正転、マイナスなら、P3_0端子を"1"にして右後モータの回転を逆転させます。</p>
615行～617行	<p>右後モータに設定するPWMが0%なら、TRDGRB0にTRDGRA0(周期を設定するレジスタ)と同じ値を設定しPWMを0%にします。</p> <p>下図のように、波形を"1"にするタイミングと"0"にするタイミングが同時に起こります。この場合、"0"にするタイミングが優先されるため、波形が"0"になり続け、モータに出力するPWMが0%になります。</p>
618行～620行	<p>右後モータに設定するPWMが1～5%の場合、PWMの設定はinterrupt intTRD0 割り込みで設定しているため、PWM値が1～5%だと割り込みプログラムで設定しようとしている最中に、波形が"0"になってしまいます。よって最小値は5%にすることとして、PWMを1～5%にするときは5%の値を設定します。</p> <p>※最小値5%は、割り込みが発生してから468行、469行が実行されるまでの時間を実測して決めました。</p>
621行～623行	<p>右後モータに設定するPWMが6～99%なら、trdgrb0_buffにTRDGRB0に設定する値を計算して設定します。実際にTRDGRB0に設定されるのは割り込みプログラム内で行われ、下図のように、PWM波形が出力されます。</p>

5. プログラムの解説

624行～ 626行	設定する PWM が 100%なら、波形を常に“1”にしますが、今回、FET ドライバに IR2302 という IC を使っています。この IC は、PWM を加えないと動作しなくなるため、PWM100%にすることはできません。今回は 99%を上限に設定します。
---------------	---

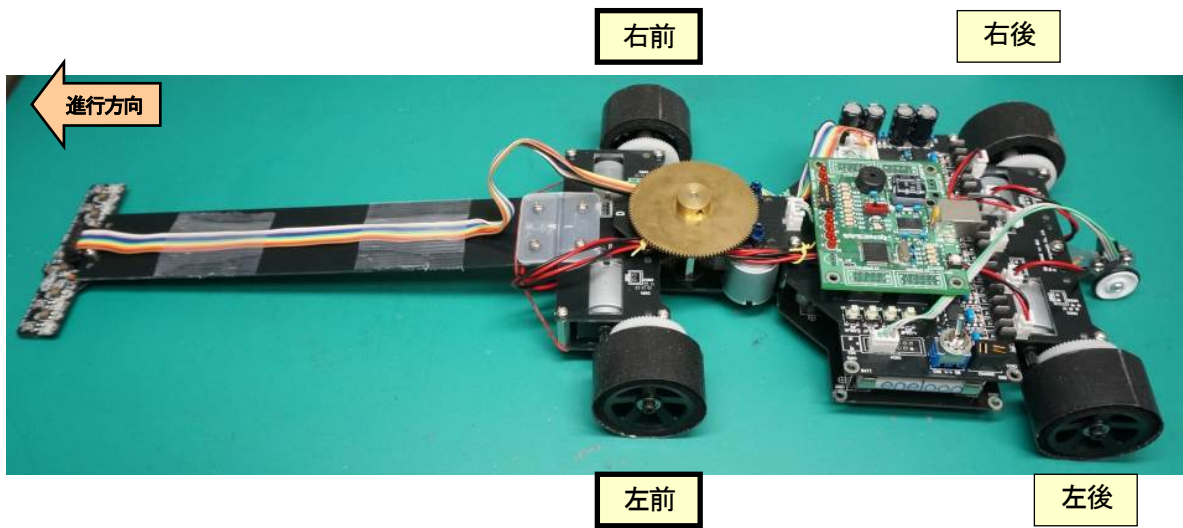
左後モータについても同様の考え方なので、説明は省略します。

5.2.20 前輪の速度制御(ディップスイッチによる PWM 減速あり)

```

630 : /*****/
631 : /* 前輪の速度制御 */
632 : /* 引数 左モータ:-100~100 , 右モータ:-100~100 */
633 : /*      0で停止、100で正転100%、-100で逆転100% */
634 : /* 戻り値 なし */
635 : /*****/
636 : void motor_f( int accele_l, int accele_r )
637 : {
638 :     int sw_data;
639 :
640 :     sw_data = dipsw_get() + 5;      /* ディップスイッチ読み込み */
641 :     accele_l = accele_l * sw_data / 20;
642 :     accele_r = accele_r * sw_data / 20;
643 :
644 :     motor2_f( accele_l, accele_r );
645 : }
    
```

基板マイコンカーVer.2 の前輪モータ 2 個を制御する関数です。



使い方は、motor_r 関数と同様です。motor_r 関数部分を参照してください。

5.2.21 前輪の速度制御 2(ディップスイッチには関係しない motor 関数)

```

647 : /*****/
648 : /* 前輪の速度制御2 ディップスイッチには関係しないmotor関数 */
649 : /* 引数 左モータ:-100~100 , 右モータ:-100~100 */
650 : /*      0で停止、100で正転100%、-100で逆転100% */
651 : /* 戻り値 なし */
652 : /* メモ 1~4%は、5%になります */
653 : /*****/
654 : void motor2_f( int accele_l, int accele_r )
655 : {
656 :     /* 左前モータ */
657 :     if( accele_l >= 0 ) {
658 :         p3_4 = 0;
659 :     } else {
660 :         p3_4 = 1;
661 :         accele_l = -accele_l;
662 :     }
663 :     if( accele_l == 0 ) {
664 :         // 0%のとき
665 :         trcgrb = trcgrb_buff = trcgra;
666 :     } else if( accele_l <= 5 ) {
667 :         // 1~5%のときは、プログラムの仕様で5%とする
668 :         trcgrb_buff = (unsigned long)(TRC_MOTOR_CYCLE-1) * 5 / 100;
669 :     } else if( accele_l <= 99 ) {
670 :         // 6~99%のとき
671 :         trcgrb_buff = (unsigned long)(TRC_MOTOR_CYCLE-1) * accele_l / 100;
672 :     } else {
673 :         // 100%のとき(実際は99%を設定)
674 :         trcgrb_buff = (unsigned long)(TRC_MOTOR_CYCLE-1) * 99 / 100;
675 :     }
676 :
677 :     /* 右前モータ */
678 :     if( accele_r >= 0 ) {
679 :         p3_2 = 0;
680 :     } else {
681 :         p3_2 = 1;
682 :         accele_r = -accele_r;
683 :     }
684 :     if( accele_r == 0 ) {
685 :         // 0%のとき
686 :         trdgrc0 = trdgrc0_buff = trdgra0;
687 :     } else if( accele_r <= 5 ) {
688 :         // 1~5%のときは、プログラムの仕様で5%とする
689 :         trdgrc0_buff = (unsigned long)(TRD_MOTOR_CYCLE-1) * 5 / 100;
690 :     } else if( accele_r <= 99 ) {
691 :         // 6~99%のとき
692 :         trdgrc0_buff = (unsigned long)(TRD_MOTOR_CYCLE-1) * accele_r / 100;
693 :     } else {
694 :         // 100%のとき(実際は99%を設定)
695 :         trdgrc0_buff = (unsigned long)(TRD_MOTOR_CYCLE-1) * 99 / 100;
696 :     }
697 : }

```

motor_f 関数は、ディップスイッチの設定でモータに出力する割合をさらに落としましたが、motor2_f 関数は、プロ

5. プログラムの解説

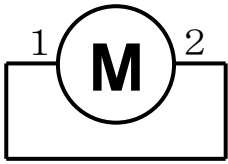
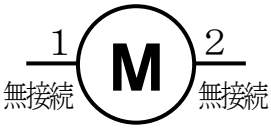
プログラムの引数どおりの割合をモータに出力します。動作原理は、motor2_r 関数と同様です。motor2_r 関数の説明を参照してください。

5.2.22 後モータ停止動作(ブレーキ、フリー)設定

```

699 : /*****/
700 : /* 後モータ動作 (BRAKE=動作とブレーキの繰り返し、FREE=0%のフリー) */
701 : /* 引数 左モータ:FREE or BRAKE , 右モータ:FREE or BRAKE */
702 : /* 戻り値 なし */
703 : /* メモ フリーにすると、PWM値に関わらず0%のフリーになります */
704 : /* モータドライブ基板TypeSとは動作が違いますので気を付けてください */
705 : /*****/
706 : void motor_mode_r( int mode_l, int mode_r )
707 : {
708 :     if( mode_l ) {
709 :         p3_7 = 0;
710 :     } else {
711 :         p3_7 = 1;
712 :     }
713 :     if( mode_r ) {
714 :         p3_1 = 0;
715 :     } else {
716 :         p3_1 = 1;
717 :     }
718 : }
    
```

後モータの停止状態を、ブレーキにするかフリーにするかを設定する関数です。フリーとブレーキの違いを下図に示します。

	ブレーキ	フリー
回路		
動作	<p>モータの端子間に電圧を加えると、軸が回転します。逆に、軸を回転させると、端子間に電圧が発生します。軸が回転している状態では端子間に電圧が発生しますが、端子間をショートさせると、端子間に大電流が流れます。この電流によって、モータの軸には回転を止める方向に回転力が発生します。これがブレーキのような動作になります。</p>	<p>モータの軸が回っている状態で端子を無接続にすると、モータの軸の回転は惰性で遅くなっていきます。これをフリーといいます。</p>

モータを PWM60%で正転させたときの、ブレーキとフリーの違いを下表に示します。

	基板マイコンカーVer.2	モータドライブ基板 TypeS Ver.4
BRAKE にしたときの動作	60%正転、40%ブレーキを繰り返します。	60%正転、40%ブレーキを繰り返します。
FREE にしたときの動作	PWMの割合にかかわらず、100%フリー動作になります。	60%正転、40%フリーを繰り返します。

基板マイコンカーVer.2 のフリーは、PWM 値に関わらずフリー100%になります。モータドライブ基板 TypeS Ver.4 のプログラムとは動作が違うので、移植する場合は気をつけてください。

5.2.23 前モータ停止動作(ブレーキ、フリー)設定

```

720 : /*****/
721 : /* 前モータ動作 (BRAKE=動作とブレーキの繰り返し、FREE=0%のフリー) */
722 : /* 引数 左モータ:FREE or BRAKE , 右モータ:FREE or BRAKE */
723 : /* 戻り値 なし */
724 : /* メモ フリーにすると、PWM値に関わらず0%のフリーになります */
725 : /* モータドライブ基板TypeSとは動作が違いますので気を付けてください */
726 : /*****/
727 : void motor_mode_f( int mode_l, int mode_r )
728 : {
729 :     if( mode_l ) {
730 :         p3_5 = 0;
731 :     } else {
732 :         p3_5 = 1;
733 :     }
734 :     if( mode_r ) {
735 :         p3_3 = 0;
736 :     } else {
737 :         p3_3 = 1;
738 :     }
739 : }

```

前モータを回すとき、停止の状態をブレーキにするかフリーにするか選択します。他は motor_mode_r 関数と同じです。

5.2.24 ステアリングモータの PWM 設定

```

741 : /*****/
742 : /* サーボモータ制御 */
743 : /* 引数 サーボモータPWM : -100~100 */
744 : /* 0で停止、100で右に100%、-100で左に100% */
745 : /* 戻り値 なし */
746 : /*****/
747 : void servoPwmOut( int pwm )
748 : {
749 :     if( pwm >= 0 ) {
750 :         p6_0 = 0;
751 :     } else {
752 :         p6_0 = 1;
753 :         pwm = -pwm;
754 :     }
755 :
756 :     if( pwm <= 2 ) {
757 :         trcgrd = trcgrd_buff = trcgra;
758 :     } if( pwm <= 99 ) {
759 :         trcgrd_buff = (unsigned long)(TRC_MOTOR_CYCLE-1) * pwm / 100;
760 :     } else {
761 :         trcgrd_buff = (unsigned long)(TRC_MOTOR_CYCLE-1) * 99 / 100;
762 :     }
763 : }

```

基板マイコンカーVer.2 のステアリングモータに PWM を設定する関数です。

5. プログラムの解説

749行～ 754行	ステアリングモータに設定する PWM 値が、プラスなら P6_0 端子を"0"にしてステアリングモータを正転、マイナスなら P6_0 端子を"1"にしてステアリングモータを逆転させます。
756行～ 757行	設定する PWM が 0～2%の場合、PWM を 0%として設定します。設定内容について詳しくは「5.2.19 後輪の速度制御 2(ディップスイッチには関係しない motor 関数)」を参照してください。
758行～ 759行	設定する PWM が 3～100%の場合、PWM 値を計算して、trcgrd_buff 変数に値を設定します。この変数は、1ms ごとの割り込み関数「intTRC 関数」で、TRCGRD へ代入されます。 100%を設定したときは、 $\begin{aligned} \text{trcgrd_buff} &= (\text{unsigned long})(\text{TRC_MOTOR_CYCLE}-2) * \text{pwm} / 100 \\ &= (20000 - 2) * 100 / 100 \\ &= 19998 \end{aligned}$ を設定します。これは、TRC の 1 カウント分の 50ns だけ"0"になりますが、「50ns/1ms=0.005%」なので、ほぼ 100%と見なして問題ありません。 設定内容について詳しくは「5.2.19 後輪の速度制御 2(ディップスイッチには関係しない motor 関数)」を参照してください。
760行～ 761行	設定する PWM が 100%なら、波形を常に"1"にしますが、今回、FET ドライバに IR2302 という IC を使っています。この IC は、PWM を加えないと動作しなくなるため、PWM100%にすることはできません。今回は 99%を上限に設定します。

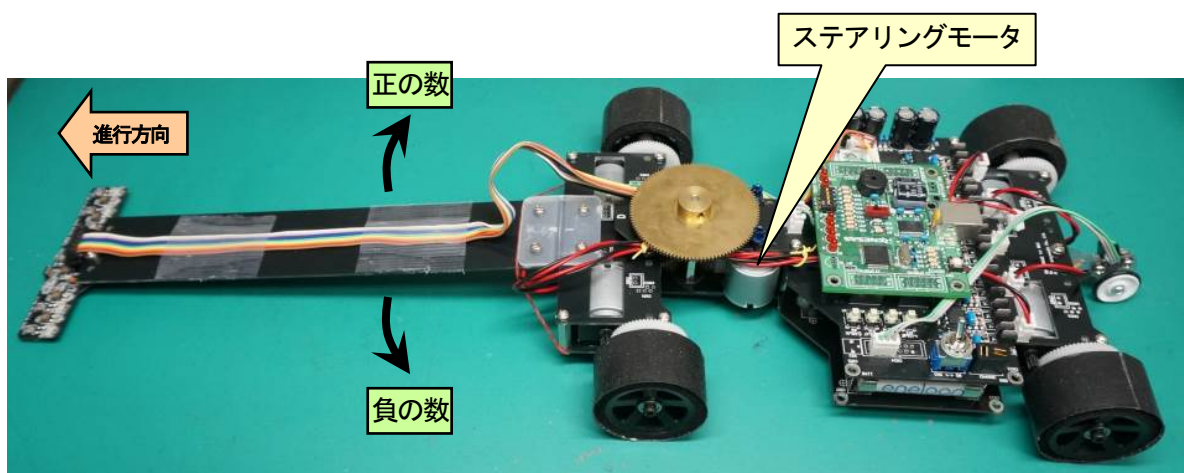
使い方を、下記に示します。

```
servoPwmOut ( ステアリングモータの PWM );
```

ステアリングモータの PWM の値は、下記を設定することができます。

- 0… 停止
- 1～100… 右回転の割合、100 が一番速い
- 1～-100… 左回転の割合、100 が一番速い

今回のプログラムは、引数を正の数にすると右へ、負の数にすると左へステアリングが回ります。



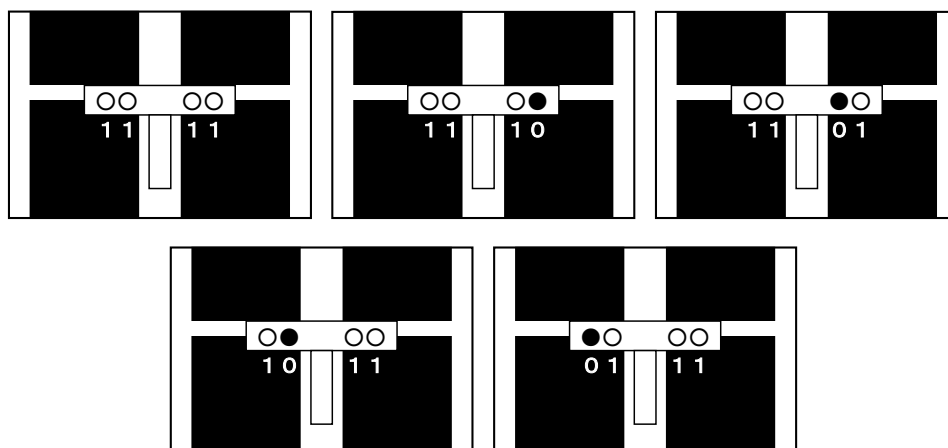
5.2.25 クロスラインの検出処理

```

765 : /*****
766 : /* クロスライン検出処理
767 : /* 引数 なし
768 : /* 戻り値 0:クロスラインなし 1:あり
769 : /*****
770 : int check_crossline( void )
771 : {
772 :     unsigned char b;
773 :     int ret = 0;
774 :
775 :     b = sensor_inp();
776 :     if( b==0x0f || b==0x0e || b==0x0d || b==0x0b || b==0x07 ) {
777 :         ret = 1;
778 :     }
779 :     return ret;
780 : }

```

センサの状態をチェックして、クロスラインかどうか判断する関数です。アナログセンサ基板 TypeS Ver.2 の中心を除くデジタルセンサ 4 つの内、3 つ以上が白を検出するとクロスラインと判断します。戻り値は、クロスラインを検出したら"1"、クロスラインなしは"0"が返ってきます。



5. プログラムの解説

5.2.26 ステアリングモータ角度の取得

```

782 : /*****
783 : /* サーボ角度取得
784 : /* 引数 なし
785 : /* 戻り値 入れ替え後の値
786 : /*****
787 : int getServoAngle( void )
788 : {
789 :     return( ad5 - iAngle0 );
790 : }
    
```

ステアリングモータの角度は、AN5(P0_2)端子に接続されているポテンシオメータ(ボリューム)の値で分かります。iAngle0 は、0 度のときの A/D 変換値を入れておきます。例えば、角度が 0 度のとき A/D 変換値が 456 なら、iAngle0 変数に 456 を代入すると、戻り値は下記のようにになります。

$$\begin{aligned}
 \text{戻り値} &= \text{A/D 変換値} - 0 \text{ 度のときの A/D 変換値} \\
 &= 456 - 456 \\
 &= 0
 \end{aligned}$$

基板マイコンカーVer.2 は、実測で左右 40 度ずつハンドルが切れました。中心と左右最大にハンドルを切ったときの電圧を測ります。テストでポテンシオメータの電圧(CN3 の 1 ピンと 3 ピン間)を計った結果、

左いっぱい…2.85V 中心…2.23V 右いっぱい…1.61V

となりました(下左図)。5.00V(電源電圧)が 1023 なので、それぞれの電圧を A/D 値に変換すると、

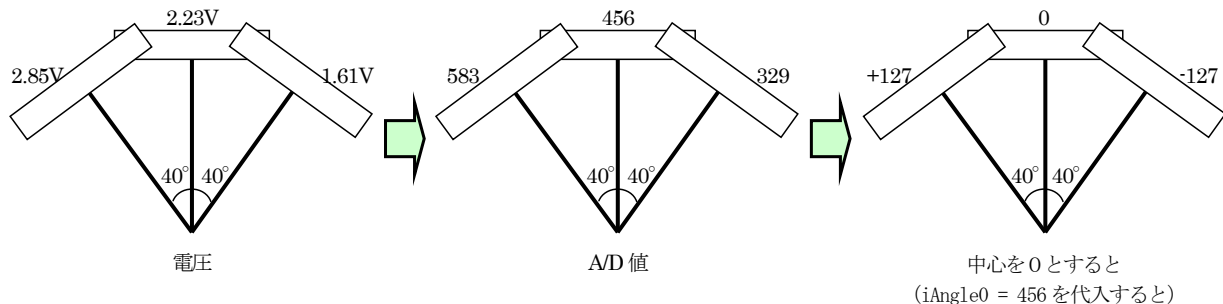
左いっぱい…2.85/5*1023=683 中心…2.23/5*1023=456 右いっぱい…1.61/5*1023=329

となります(下中図)。

787 行の iAngle0 変数には、0 度のときの A/D 値を入れておきます。iAngle0 変数に 456 の値を入れると、

左いっぱい…+127 中心…0 右いっぱい…-127

となります(下右図)。



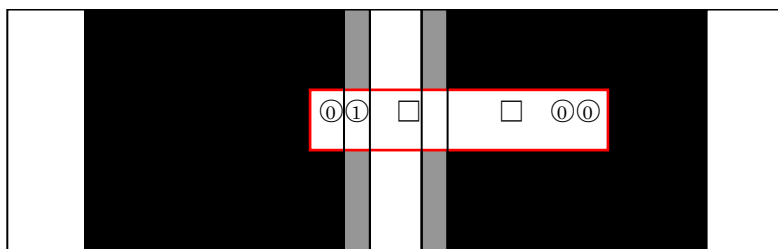
5.2.27 アナログセンサ値の取得

```
792 : /*****  
793 : /* アナログセンサ値取得 */  
794 : /* 引数 なし */  
795 : /* 戻り値 センサ値 */  
796 : *****/  
797 : int getAnalogSensor( void )  
798 : {  
799 :     int ret;  
800 :  
801 :     ret = ad1 - ad2; /* アナログセンサ情報取得 */  
802 :  
803 :     if( !crank_mode ) {  
804 :         /* クランクモードでなければ補正処理 */  
805 :         switch( iSensorPattern ) {  
806 :             case 0:  
807 :                 if( sensor_inp() == 0x04 ) {  
808 :                     ret = -650;  
809 :                     break;  
810 :                 }  
811 :                 if( sensor_inp() == 0x02 ) {  
812 :                     ret = 650;  
813 :                     break;  
814 :                 }  
815 :                 if( sensor_inp() == 0x0c ) {  
816 :                     ret = -700;  
817 :                     iSensorPattern = 1;  
818 :                     break;  
819 :                 }  
820 :                 if( sensor_inp() == 0x03 ) {  
821 :                     ret = 700;  
822 :                     iSensorPattern = 2;  
823 :                     break;  
824 :                 }  
825 :                 break;  
826 :             case 1:  
827 :                 /* センサ右寄り */  
828 :                 ret = -700;  
829 :                 if( sensor_inp() == 0x04 ) {  
830 :                     iSensorPattern = 0;  
831 :                 }  
832 :                 break;  
833 :             case 2:  
834 :                 /* センサ左寄り */  
835 :                 ret = 700;  
836 :                 if( sensor_inp() == 0x02 ) {  
837 :                     iSensorPattern = 0;  
838 :                 }  
839 :                 break;  
840 :             }  
841 :         }  
842 :     }  
843 : }  
844 :  
845 :     return ret;  
846 : }
```

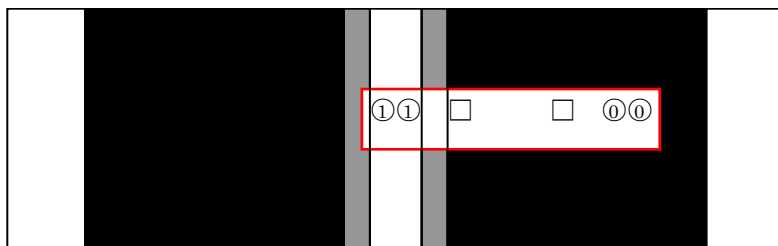
アナログセンサ左(P0_6 端子に接続)とアナログセンサ右(P0_5 端子に接続)の差分を計算して、コース中心からのずれを検出します。801 行で「アナログセンサ左－アナログセンサ右」の計算をしています。それ以降の行は、急

5. プログラムの解説

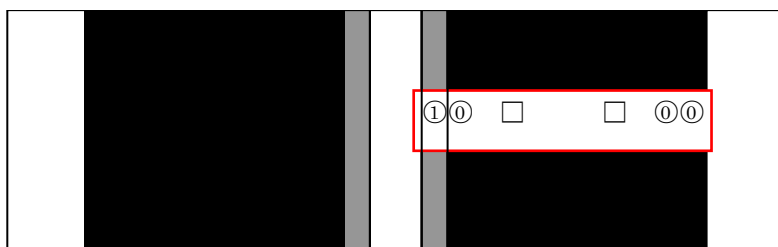
カーブのときにステアリングが反応しきれずに中心線を大きくはずれてしまった場合、デジタルセンサを使用して補正させる処理を行っています。その処理を 805 行から 842 行まで行っています。



センサが“0100”になると、センサ値を強制的に-650 とします。



デジタルセンサが“1100”になると、センサ値を強制的に-700 とします。**この状態をデジタルセンサが“0100”になるまで保持します。**



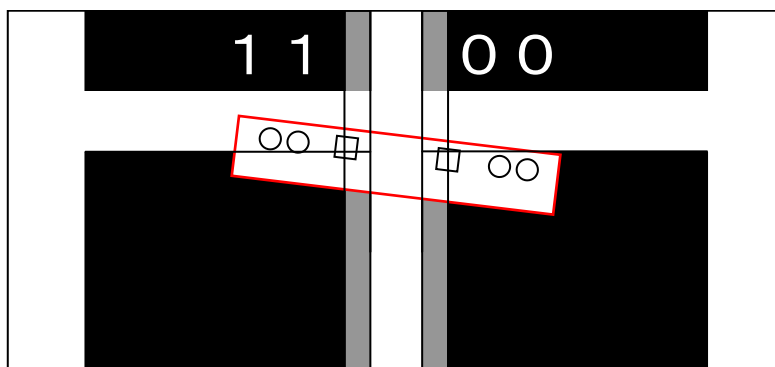
更にはずれてもデジタルセンサが“0100”になるまで-700 の値を保持し続けます。この状態になると、アナログセンサは両方もも黒色ですので差分をとっても 0 となります。補正がなければ、この状態を中心と認識してしまいます。

このように、アナログセンサだけでは追従しきれない場合を想定して、デジタルセンサを使いアナログセンサの値を補正しています。逆のずれも、考え方は同じです。

803 行目で crank_mode 変数の値をチェックしています。これは、クロスラインを検出したときや直角を検出するときにデジタルセンサの補正を行わないよう、補正機能を OFF にするための変数です。crank_mode が 0 なら、if の { } 内は実行しません。

例えば、下図のようなときはクロスライン検出状態です。センサの反応は“1100”となり、補正するとセンサ値は -700 となってしまいます。もう少し進み、“1110”や“1111”になったらクロスラインと判断して、すぐに crank_mode を 1 にしてデジタルセンサの補正を停止します。補正を停止しないと、急カーブと判断してしまい、ハンドルを切って脱輪します。

ちなみに、“1100”から“1111”に変化するまでの間は急カーブと判断してしまいますが、非常に短い時間なのでほとんど影響はありません。



5.2.28 ステアリングモータ制御

```

848 : /*****/
849 : /* サーボモータ制御 */
850 : /* 引数 なし */
851 : /* 戻り値 グローバル変数 iServoPwm に代入 */
852 : /*****/
853 : void servoControl( void )
854 : {
855 :     int i, iRet, iP, iD;
856 :     int kp, kd;
857 :
858 :     i = getAnalogSensor(); /* センサ値取得 */
859 :     kp = 1; /* kp を増やしていき、ブルブルが */
860 :     kd = 0; /* 大きくなったら kd を増やしてく */
861 : /* ださい */
862 : /* サーボモータ用 PWM 値計算 */
863 :     iP = kp * i; /* 比例 */
864 :     iD = kd * (iSensorBefore - i); /* 微分(目安は P の 5~10 倍) */
865 :     iRet = iP - iD;
866 :     iRet /= 64;
867 :
868 :     /* PWM の上限の設定 */
869 :     if( iRet > 50 ) iRet = 50; /* マイコンカーが安定したら */
870 :     if( iRet < -50 ) iRet = -50; /* 上限を 95 くらいにしてください */
871 :     iServoPwm = iRet;
872 :
873 :     iSensorBefore = i; /* 次回はこの値が 1ms 前の値となる*/
874 : }

```

この関数で、コースの中心線からセンサがどれだけずれているかを検出して、ステアリングモータの PWM 値を計算します。ステアリングモータ制御の要(かなめ)の部分です。

(1) PID 制御とは？

自動制御方式の中でもっとも良く使われる制御方式に PID 制御という方式があります。この PID とは

P : Proportional (比例)

I : Integral (積分)

D : Differential (微分)

の 3 つの組み合わせで制御する方式で、きめ細かくステアリングモータの PWM を調整してスムーズな制御を行うことができます。

PID 制御についての詳細は、ホームページや書籍が多数出ていますのでそちらを参照してください。

今回、ステアリングモータの制御は比例制御と微分制御を行います。PD 制御と呼びます。

(2) P(比例)制御

比例制御とは、目標値からのずれに対して比例した制御量 P を与えます。 P を計算する式を下記に示します。

$$\text{制御量 } P = k_p \times p$$

k_p = 定数

p = 現在の値 - 目標の値

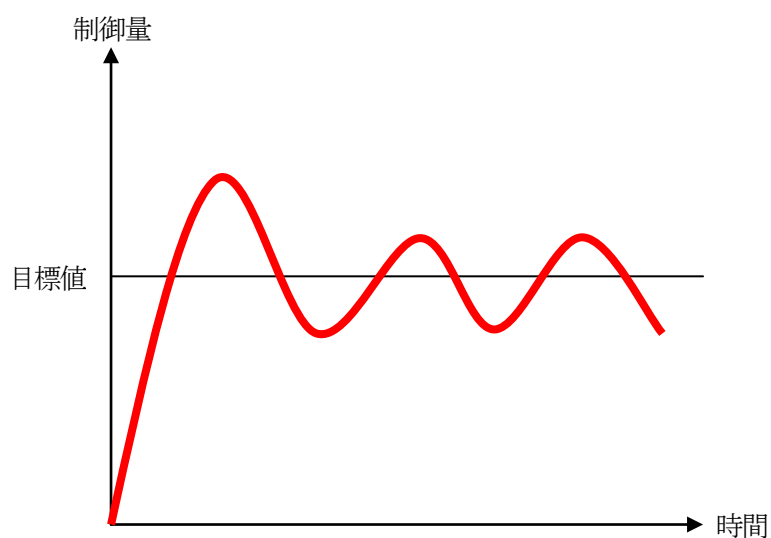
= 現在のアナログセンサ値 - 目標のアナログセンサ値

= `getAnalogSensor()` - 0

= `getAnalogSensor()`

目標のアナログセンサ値は、ちょうどコースの中心の値である 0 になります。

制御としては早く目標値に近づけたいので、ずれが大きいほどステアリングモータの PWM を多くします。そのため、目標値に到達しても速度を落とさず、目標値をいったりきたりと振動してしまいます。



(3) D(微分)制御を加える

微分制御とは、瞬間的な変化量を計算して比例制御を押さえるような働きをします。微分制御量 D を計算する式を下記に示します。

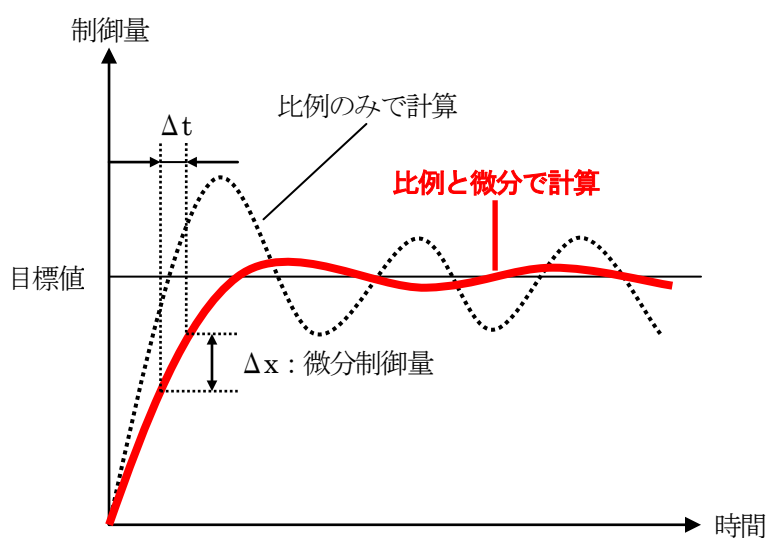
制御量 $D = kd \times d$

kd = 定数

d = 過去のアナログセンサ値 - 現在のアナログセンサ値
= `iSensorBefore` - `getAnalogSensor()`

過去のアナログセンサ値を `iSensorBefore` というグローバル変数に保存しておきます。

比例制御のみで振動していても、微分量を加えると振動を抑えることができます。ただし、比例制御を押さえる働きをしますので、目標値に近づく時間は長くなります。時間は数 ms～数十 ms のレベルです。それが、実際の走りに対して、どう影響するかは検証する必要があります。



(4) 最終制御量

ステアリングモータに加える制御量を計算する式を下記に示します。

最終制御量 = P値 - D値

プログラムでは、最終制御量に定数をかけて PWM 値に調整します。

最後に、ステアリングモータに大きい PWM を加えるとステアリング部のギヤが壊れてしまうので、PWM の上限を設けます。サンプルプログラムは、50%以上にならないようにしています。この数値を小さくしすぎると、せつかくの PD 制御も上制限されてしまうので反応が遅くなります。大きすぎると万が一大きい PWM をかけてしまった場合、ギヤが壊れます。50%の設定は最初だけとして、コーストレースが安定したら 90%程度にしてください。

今回のプログラムでは、P 値、D 値は、下記のように設定しています。

P制御の定数 = **1** …859 行

D制御の定数 = **0** …860 行

P 値、D 値をそれぞれ調整して、ステアリング部分がばたつかず滑らかに、かつ力強く、中心線をトレースするように調整してください。※力が弱いと S 字カーブでステアリングを左右どちらかに曲げた直後、逆に曲げるようとしたときに中心を追従しません。

5. プログラムの解説

5.2.29 内輪 PWM 値計算

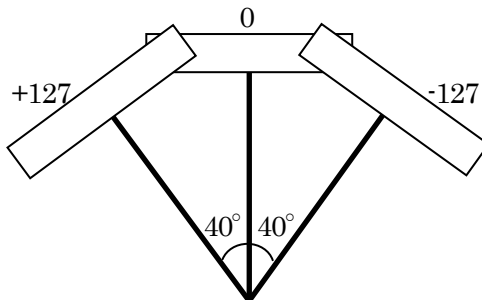
```

876 : /*****
877 : /* 外輪のPWMから、内輪のPWMを割り出す ハンドル角度は現在の値を使用 */
878 : /* 引数 外輪PWM */
879 : /* 戻り値 内輪PWM */
880 : /*****
881 : int diff( int pwm )
882 : {
883 :     int i, ret;
884 :
885 :     i = getServoAngle() / 3;          /* 1度あたりの増分で割る */
886 :     if( i < 0 ) i = -i;
887 :     if( i > 45 ) i = 45;
888 :     ret = revolution_difference[i] * pwm / 100;
889 :
890 :     return ret;
891 : }
    
```

※小数点は使わないでください。
 小数点にしたい場合は、かけ算と割り算を組み合わせてください 例) / 3.5 → * 7 / 2

diff関数は、引数に外輪(多く回るタイヤ側)のPWM値を入れて呼び出すと、内輪(少なく回るタイヤ側)のPWM値が返って来るとい関数です。

885行で、現在の角度を取得します。getServoAngle関数の戻り値はA/D値なので、「度」に直す必要があります。ハンドルを±40度動かしたときのA/D値を、下記に示します。



左右に40度ステアリングを動かしたときのA/D値の変化量は±127でした。左40度するとき、A/D値は127なので、1度あたりのA/D値は下記ようになります。

$$\begin{aligned}
 \text{1度あたりのA/D値} &= \text{左40度ときのA/D値} \div \text{左40度} \\
 &= 127 \div 40 \\
 &= 3.175 \approx 3
 \end{aligned}$$

値は四捨五入して、整数にします。よって、A/D値3が約1度となります。

886行は、負の数を正の数に変換しています。

887行は、45度以上の角度のときは、45度にしておきます。これは次に説明する配列の設定が、45度までしか無いためです。

888行は、左タイヤと右タイヤの回転差を計算します。まず、外輪の回転数を100と考えて、内輪の回転数を計算します。例えば、現在の角度が25度するとき、添字部分に25が入り、内輪の回転数が戻り値となります。

```

ret = revolution_difference[i]
    = revolution_difference[ 25 ]
    = 68
    
```

外輪 100%のとき、内輪は戻り値である 68%であることが分かります。

次に、外輪が 100%で無い場合を計算します。内輪は外輪の回転に比例しますので、割合をかければ内輪の PWM 値が分かります。例えば、外輪が 60%なら内輪は次の計算で求めることができます。

```
ret = 68 * 外輪の PWM 値 / 100
    = 68 * 60 / 100
    = 40.8 ≒ 40      ※小数点は扱えないので、切り捨てられます
```

ステアリング角度が 25 度、外輪が 60%のとき、下記プログラムを実行すると kakudo 変数には 40 が代入されます。

```
kakudo = diff( 60 );
```

5.2.30 main 関数—初期化

```
101 : /*****/
102 : /* メインプログラム */
103 : /*****/
104 : void main( void )
105 : {
106 :     int i;
107 :     unsigned char b;
108 :
109 :     /* マイコン機能の初期化 */
110 :     init();                /* 初期化 */
111 :     asm( " fset I " );    /* 全体の割り込み許可 */
112 :
113 :     /* マイコンカーの状態初期化 */
114 :     motor_mode_f( BRAKE, BRAKE ); /* 基板マイコンカーのFREEは、 */
115 :     motor_mode_r( BRAKE, BRAKE ); /* PWM値に関係なく必ずフリーになる */
116 :     motor_f( 0, 0 );
117 :     motor_r( 0, 0 );
118 :     servoPwmOut( 0 );
```

main 関数では最初に、R8C/35A マイコンの内蔵周辺機能の初期化、割り込みの許可、モータを停止状態にします。

5. プログラムの解説

5.2.31 パターン処理

マイコンカーの状態は pattern 変数で管理しています。通称、パターン処理と呼ぶことにします。pattern が 0 でスタート待ち、11 で通常トレースなど、それぞれの状態に応じてパターンを変えて処理内容を変えていきます。

現在のモード (pattern)	状態	pattern 変数が変わる条件
0	プッシュスイッチ押下待ち	・プッシュスイッチを押したら 1 へ
1	スタートバー開待ち	・スタートバーが開いたら 11 へ
11	通常トレース	・クロスラインを検出したら 21 へ
21	クロスライン通過処理	・200ms たったら 22 へ
22	クロスライン後のトレース、直角検出処理	・右クランクを見つけたら 31 へ ・左クランクを見つけたら 41 へ
31	右クランク処理	・曲げ終わりを検出すると 32 へ
32	少し時間がたつまで待つ	・100ms たったら 11 へ
41	左クランク処理	・曲げ終わりを検出すると 42 へ
42	少し時間がたつまで待つ	・100ms たったら 11 へ
その他	—	・0 へ

5.2.32 パターン 0:スタート待ち

```

120 :   while( 1 ) {
121 :
122 :     switch( pattern ) {
123 :     case 0:
124 :       /* プッシュスイッチ(SW4)押下待ち */
125 :       servoPwmOut( 0 );
126 :       if( pushsw_get() == 0x01 ) {
127 :         cnt1 = 0;
128 :         pattern = 1;
129 :         break;
130 :       }
131 :       i = (cnt1/200) % 2 + 1;
132 :       if( startbar_get() ) {
133 :         i += ((cnt1/100) % 2 + 1) << 2;
134 :       }
135 :       led_out( i );           /* LED 点滅処理          */
136 :       break;

```

プッシュスイッチ(SW4)押下待ちです。プッシュスイッチを押すまでの間、LED 4 個中 2 個を点滅させプッシュスイッチが押されるまで待ちます。また、スタートバー検出センサが反応すると、さらに 2 個(合計 4 個)の LED を点滅させスタートバー閉を検出していることを、選手に分かりやすく知らせます。

プッシュスイッチ(SW4)を押すと、パターン 1 へ移ります。

5.2.33 パターン 1:スタートバー開待ち

```
138 :     case 1:
139 :         /* スタートバー開待ち */
140 :         servoPwmOut( iServoPwm / 2 );
141 :         if( !startbar_get() ) {
142 :             iAngle0 = getServoAngle(); /* 0度の位置記憶          */
143 :             led_out( 0x0 );
144 :             cnt1 = 0;
145 :             pattern = 11;
146 :             break;
147 :         }
148 :         led_out( 1 << (cnt1/50) % 4 );
149 :         break;
```

パターン 1 は、スタートバーが開かれるのを待っている状態です。

140 行でステアリングモータ制御を行っています。iServoPwm 変数がステアリングモータに加える PWM 値です。割り込みプログラム内で 1ms ごとに自動で更新されていきます。スタート時、ステアリング部分がブルブル震えないように、PWM 値を半分にしています。

スタートバーが開かれると、現在の角度を getServoAngle 関数で読み込み、その値を iAngle0 変数にセットして **この状態を 0 度とします**。その後パターン 11 に移行します。

5.2.34 パターン 11:通常トレース

```

151 :     case 11:
152 :         /* 通常トレース */
153 :         servoPwmOut( iServoPwm );
154 :         i = getServoAngle();
155 :         if( i > 110 ) {
156 :             motor_f( 0, 0 );
157 :             motor_r( 0, 0 );
158 :         } else if( i > 15 ) {
159 :             motor_f( diff(80), 80 );
160 :             motor_r( diff(80), 80 );
161 :         } else if( i < -110 ) {
162 :             motor_f( 0, 0 );
163 :             motor_r( 0, 0 );
164 :         } else if( i < -15 ) {
165 :             motor_f( 80, diff(80) );
166 :             motor_r( 80, diff(80) );
167 :         } else {
168 :             motor_f( 100, 100 );
169 :             motor_r( 100, 100 );
170 :         }
171 :         if( check_crossline() ) {           /* クロスラインチェック           */
172 :             cnt1 = 0;
173 :             crank_mode = 1;
174 :             pattern = 21;
175 :         }
176 :         break;

```

153 行でステアリングモータ制御を行っています。次に 154 行でステアリング角度を取得します。角度に応じて左右回転数の設定をしています。サンプルプログラムは、ハンドル角度と駆動モータの関係を下記のようにします。

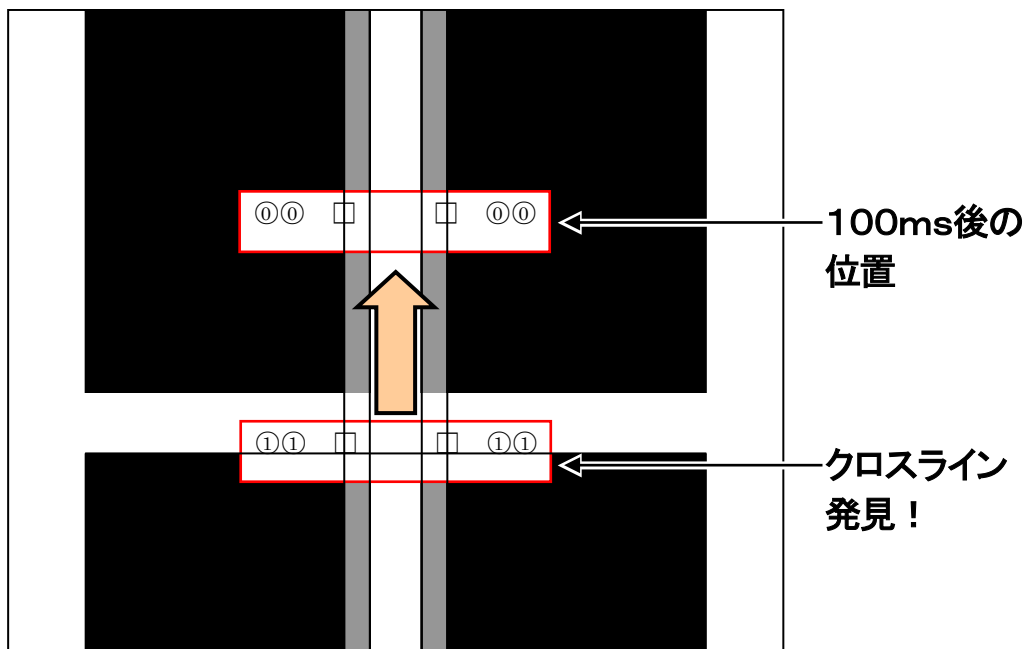
A/D 値	角度に変換 A/D 値 ÷ 3.18	左モータ PWM	右モータ PWM
111 以上	35 以上	0	0
16~110	5~35	diff(80)	80
-111 以下	-35 以下	0	0
-16~-110	-5~-35	80	diff(80)
それ以外 (-15~15)	-5~5	100	100

171 行でクロスラインチェックを行います。クロスラインを検出すると crank_mode に 1 を代入して、アナログセンサ値を取得する getAnalogSensor 関数内でデジタルセンサ補正を行わないようにします。パターンは 21 へ移ります。

5.2.35 パターン 21:クロスライン検出処理

```
178 :     case 21:
179 :         /* クロスライン通過処理 */
180 :         servoPwmOut( iServoPwm );
181 :         led_out( 0x3 );
182 :         motor_f( 0, 0 );
183 :         motor_r( 0, 0 );
184 :         if( cnt1 >= 100 ) {
185 :             cnt1 = 0;
186 :             pattern = 22;
187 :         }
188 :         break;
```

クロスラインを見つけるとブレーキをかけながら、ステアリングモータ制御を行います。100ms の間にクロスラインを通過させ、100ms 後にはパターン 22 へ移行します。クロスラインを通過しきる前にパターン 22 に移ってしまうと、クロスラインを直角と見間違ってしまうことがあります。



5.2.36 パターン 22:クロスライン後のトレース、直角検出処理

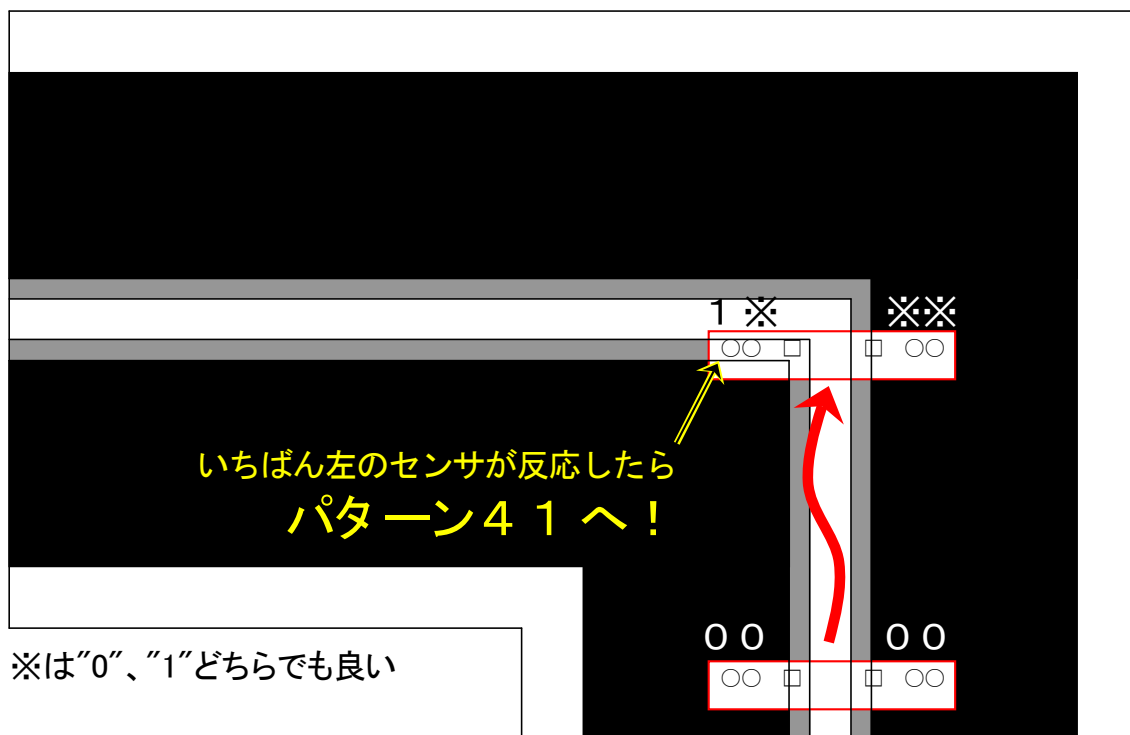
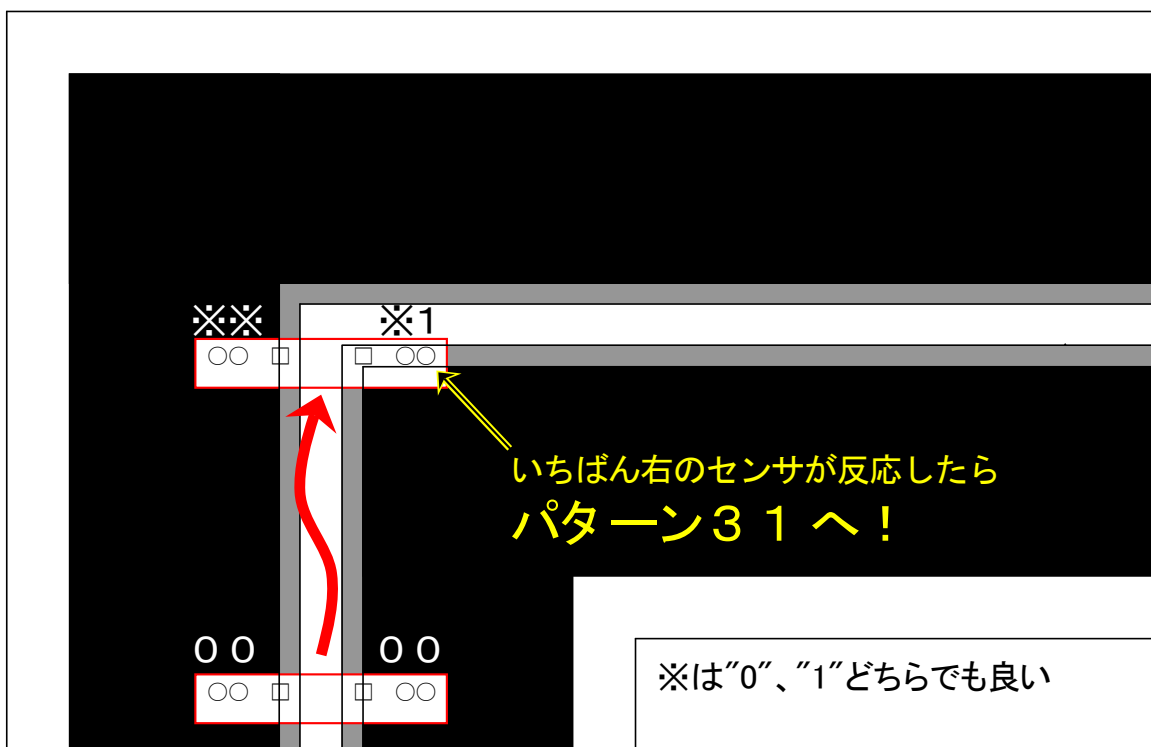
```
190 :     case 22:
191 :         /* クロスライン後のトレース、直角検出処理 */
192 :         servoPwmOut( iServoPwm );
193 :         if( iEncoder >= 11 ) {           /* エンコーダによりスピード制御 */
194 :             motor_f( 0, 0 );
195 :             motor_r( 0, 0 );
196 :         } else {
197 :             motor2_f( 70, 70 );
198 :             motor2_r( 70, 70 );
199 :         }
200 :
201 :         if( (sensor_inp()&0x01) == 0x01 ) { /* 右クランク?           */
202 :             led_out( 0x1 );
203 :             cnt1 = 0;
204 :             pattern = 31;
205 :             break;
206 :         }
207 :         if( (sensor_inp()&0x08) == 0x08 ) { /* 左クランク?           */
208 :             led_out( 0x2 );
209 :             cnt1 = 0;
210 :             pattern = 41;
211 :             break;
212 :         }
213 :         break;
```

クロスライン通過後の処理を行います。

193～199行でロータリエンコーダによる速度制御を行っています。サンプルプログラムは、1m/s以上ならPWM0%、以下ならPWM70%で走行します。

201行目で、いちばん右のデジタルセンサのみをチェック、反応すれば右クランクと判断しパターン 31へ移ります。

207行目で、いちばん左のデジタルセンサのみをチェック、反応すれば左クランクと判断しパターン 41へ移ります。



5. プログラムの解説

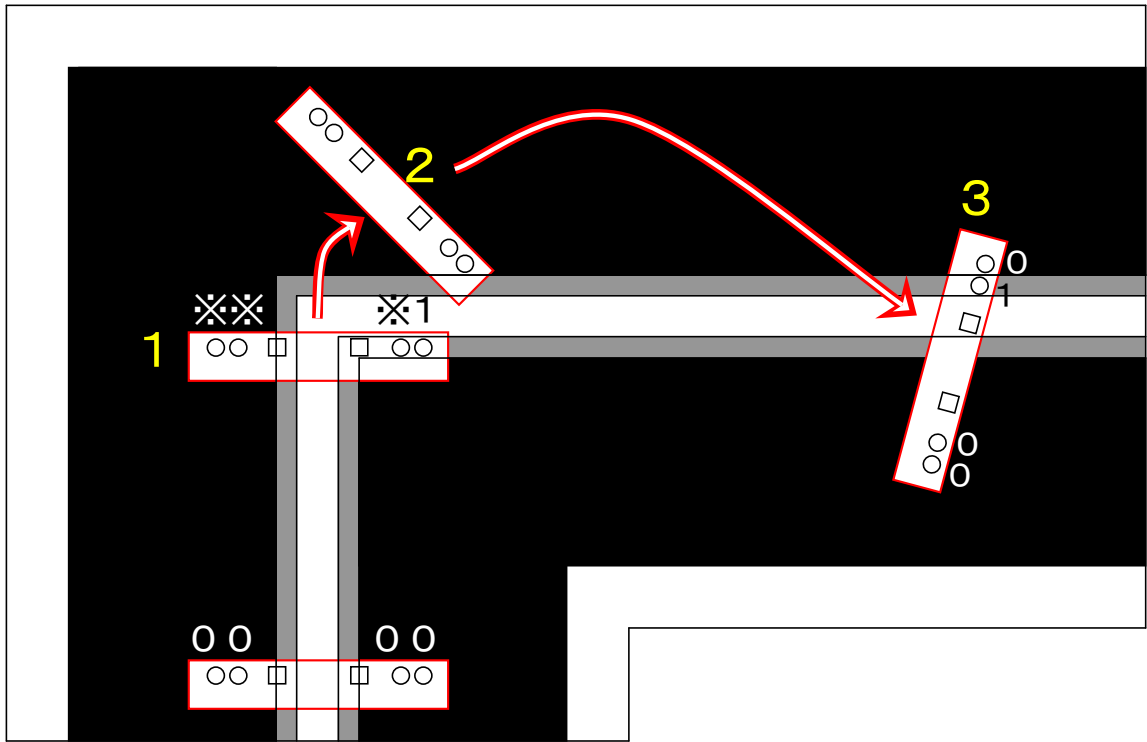
5.2.37 パターン 31: 右クランク処理

```

215 :     case 31:
216 :         /* 右クランク処理 */
217 :         servoPwmOut( 50 );           /* 振りが弱いときは大きくする */
218 :         motor_f( 60, 35 );          /* この部分は「角度計算(4WD時).xls」 */
219 :         motor_r( 48, 24 );          /* で計算 */
220 :         if( sensor_inp() == 0x04 ) { /* 曲げ終わりチェック */
221 :             cnt1 = 0;
222 :             iSensorPattern = 0;
223 :             crank_mode = 0;
224 :             pattern = 32;
225 :         }
226 :         break;
    
```

パターン 31 は、右にハンドルを曲げて、曲げ終わりかどうかチェックしている状態です。ステアリングモータの PWM 値はセンサ状態に関係なく右に 50%回転させています。ステアリングモータの動きが遅い場合は PWM 値を大きくします。ただし、ステアリング部分が約 40 度になり車体にぶつかると、ステアリングモータがロックして大電流が流れ、モータの性能が劣化したり、ギヤがかけたりすることがあります。40 度弱の角度になったら、PWM 値を小さくするプログラムを追加すると良いでしょう。

いつまで続けるかというのが 220 行です。中心以外のデジタルセンサをチェックして、“0100”ならパターン 32 に移ります。移る前に、クランクが終わったので crank_mode 変数を 0 に戻します。



1. いちばん右のデジタルセンサが“1”になったので、右クランクと判断しステアリングモータを 50%、左前モータを 60%、右前モータを 35%、左後モータを 48%、右後モータを 24%で回します。左右回転差は、ステアリング角度 40 度で計算しています。
2. デジタルセンサが“0100”になるまで待ちます。まだです。
3. デジタルセンサが“0100”になりました。パターン 32 へ移ります。

5.2.38 パターン 32: 右クランク処理後、少し時間がたつまで待つ

```

228 :     case 32:
229 :         /* 少し時間が経つまで待つ */
230 :         servoPwmOut( iServoPwm );
231 :         motor2_r( 80, 80 );
232 :         motor2_f( 80, 80 );
233 :         if( cnt1 >= 100 ) {
234 :             led_out( 0x0 );
235 :             pattern = 11;
236 :         }
237 :         break;

```

センサが中心線に到達してもすぐにパターン 11 に戻らずに、100ms 間は駆動モータを 80%にします。これはパターン 32 に移ってきたときは、ステアリングをかなり曲げています。この状態でパターン 11 に戻ると、ポテンショメータの A/D 値が-111 以下なので駆動モータの PWM 値が 0%になってしまいます。これを防ぐために 100ms 間、ステアリング角度に関係なく駆動モータの PWM を 80%にして、少し進ませます。この間に角度が浅くなりパターン 11 に戻ったときの角度でさらに進んでいきます。

5.2.39 パターン 41: 左クランク処理

```

239 :     case 41:
240 :         /* 左クランク処理 */
241 :         servoPwmOut( -50 );           /* 振りが弱いときは大きくする */
242 :         motor_f( 35, 60 );           /* この部分は「角度計算(4WD時).xls」 */
243 :         motor_r( 24, 48 );           /* で計算 */
244 :         if( sensor_inp() == 0x02 ) { /* 曲げ終わりチェック */
245 :             cnt1 = 0;
246 :             iSensorPattern = 0;
247 :             crank_mode = 0;
248 :             pattern = 42;
249 :         }
250 :         break;

```

パターン 31 と同様の考え方です。デジタルセンサが"0010"になると、パターン 42 へ移ります。

5.2.40 パターン 42: 左クランク処理後、少し時間がたつまで待つ

```

252 :     case 42:
253 :         /* 少し時間が経つまで待つ */
254 :         servoPwmOut( iServoPwm );
255 :         motor2_f( 80, 80 );
256 :         motor2_r( 80, 80 );
257 :         if( cnt1 >= 100 ) {
258 :             led_out( 0x0 );
259 :             pattern = 11;
260 :         }
261 :         break;

```

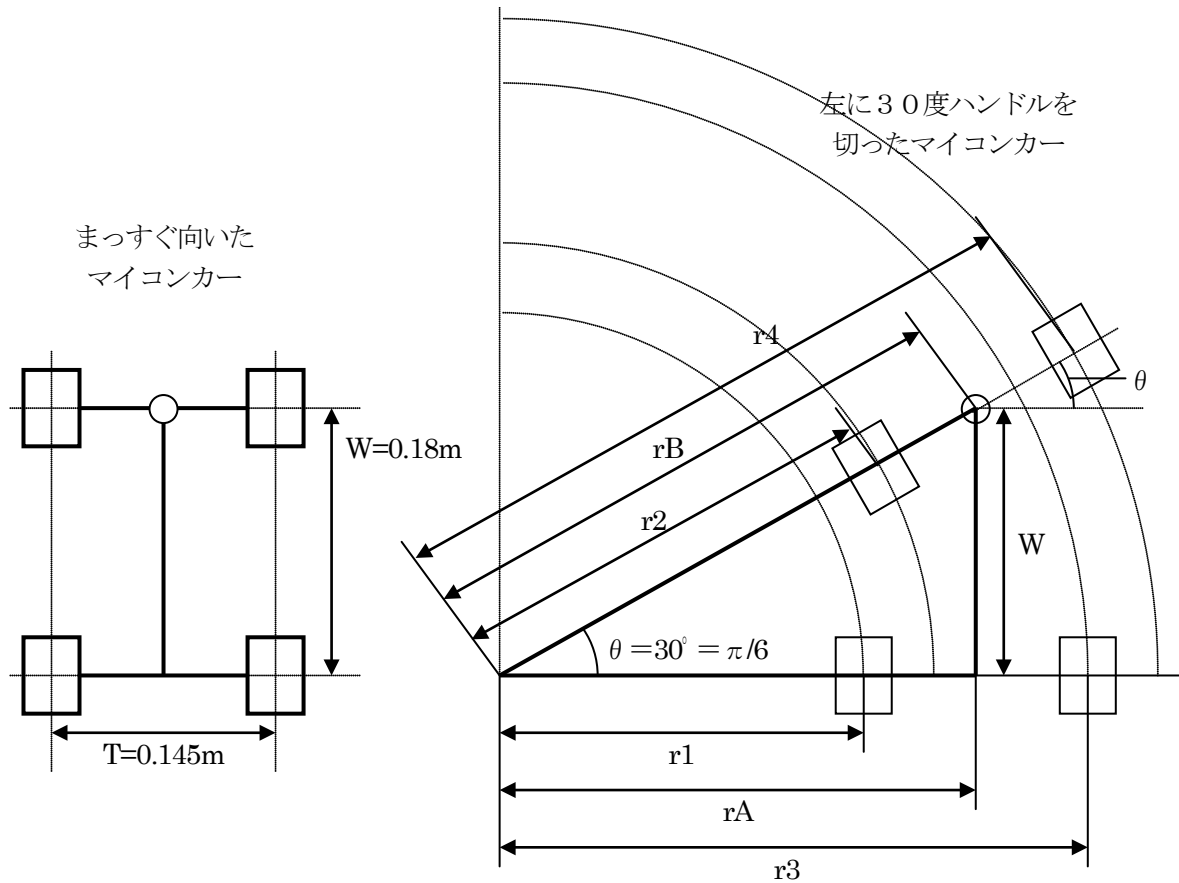
パターン 32 と同様の考え方です。

6. 4 輪の回転数計算

「角度計算(4WD 時).xls」で、下記計算ができます。

6.1 センターピボット方式 4 輪の回転数計算

センターピボット方式の 4 輪の回転数の計算方法を、説明します(下図)。



T=トレッド…左右輪の中心線の距離 (基板マイコンカーVer.2 では 0.145[m]です)

W=ホイールベース…前輪と後輪の間隔 (基板マイコンカーVer.2 では 0.18[m]です)

図のように、後輪部の底辺 r_A 、高さ W 、角度 θ の三角形の関係は次のようです。

$$\tan \theta = W / r_A$$

角度 θ 、 W が分かっていますので、 r_A が分かります。

$$r_A = W / \tan \theta = 0.18 / \tan(\pi / 6) = 0.312[m]$$

後輪内輪の半径は、

$$r_1 = r_A - T/2 = 0.312 - 0.0725 = 0.2395 [m]$$

後輪外輪の半径は、

$$r_3 = r_A + T/2 = 0.312 + 0.0725 = 0.3845 \text{ [m]}$$

また、前輪部の底辺 r_B 、高さ W 、角度 θ の三角形の関係は次のようです。

$$\sin \theta = W / r_B$$

角度 θ 、 W が分かっていますので、 r_B が分かります。

$$r_B = W / \sin \theta = 0.18 / \sin(\pi / 6) = 0.360 \text{ [m]}$$

前輪内輪の半径は、

$$r_2 = r_B - T/2 = 0.360 - 0.0725 = 0.2875 \text{ [m]}$$

前輪外輪の半径は、

$$r_4 = r_B + T/2 = 0.360 + 0.0725 = 0.4325 \text{ [m]}$$

一番回転する r_4 を 100 としたときのそれぞれの回転数は、

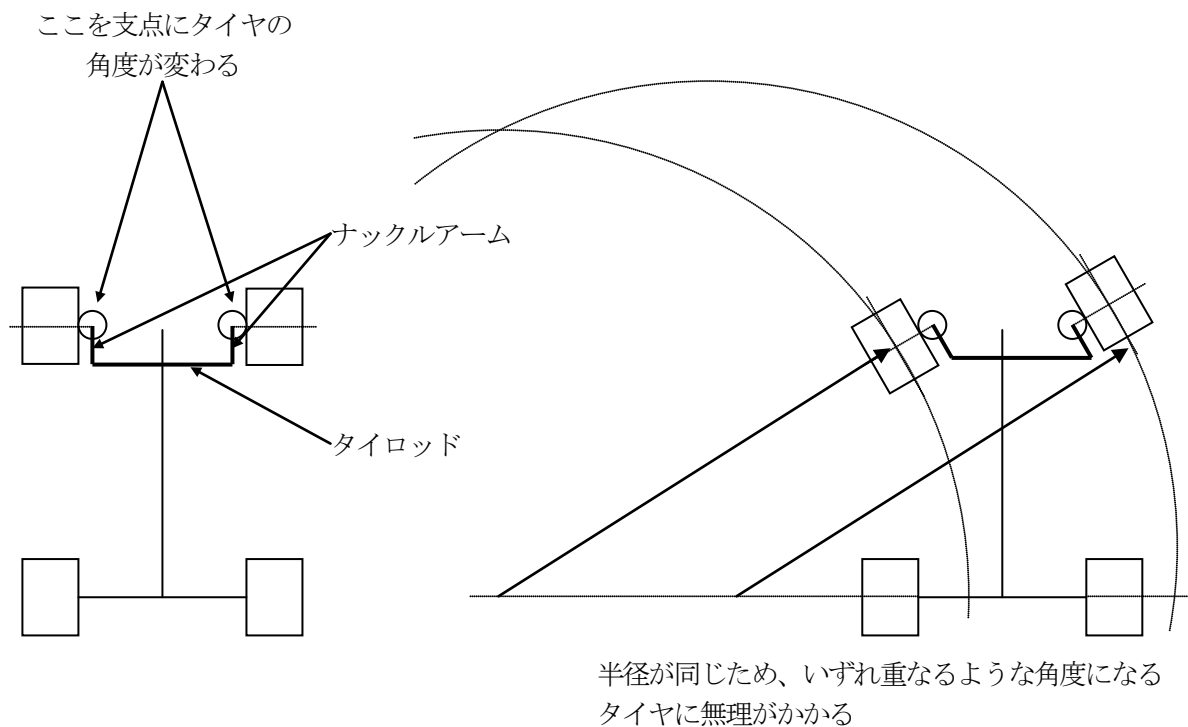
$$\begin{aligned} r_1 : r_2 : r_3 : r_4 \\ &= 0.2395 : 0.2875 : 0.3845 : 0.4325 \\ &= 0.2395 \times 100 / 0.4325 : 0.2875 \times 100 / 0.4325 : 0.3845 \times 100 / 0.4325 : 0.4325 \times 100 / 0.4325 \\ &= 55 : 66 : 89 : 100 \end{aligned}$$

ハンドル角度 30 度、前輪外輪が 100 回転するとき、後輪外輪は 89 回転、前輪内輪は 66 回転、後輪内輪は 55 回転することになります。

6.2 アッカーマン方式 4 輪の回転数計算

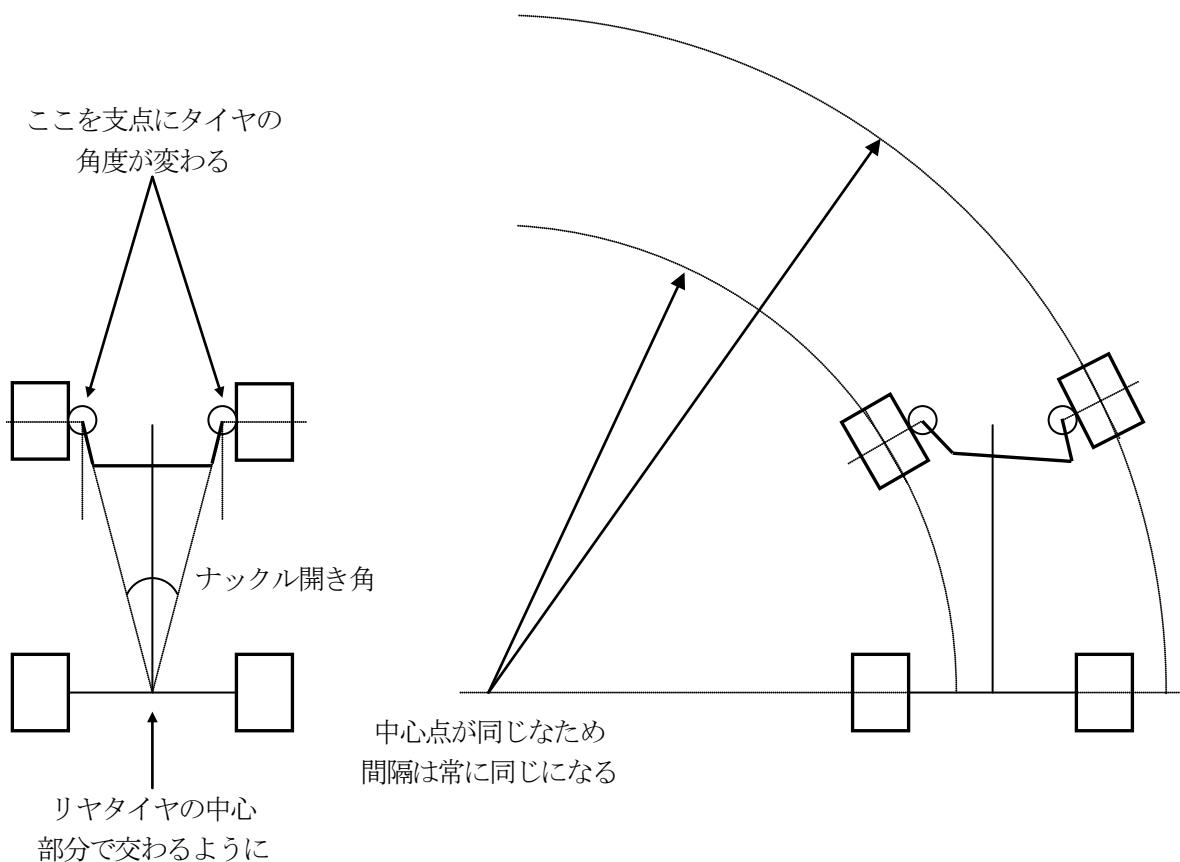
※基板マイコンカーVer.2 は、センターピポット方式なので、アッカーマン方式は必要ありませんが、参考までに掲載しておきます。

アッカーマン方式とは、通常の車のようにハンドルを切る方法です。左タイヤ、右タイヤの切れ角は実は同じではありません。もし同じ切れ角ならどうなるのでしょうか(下図)。



ナックルアームと呼ばれる部分をタイヤと平行に取り付けると、ハンドルを切ったとき、内輪と外輪の切れ角が同じになり、軌跡を見ると交差してしまいます。タイヤの幅は常に一定のため、タイヤに無理がかかります。

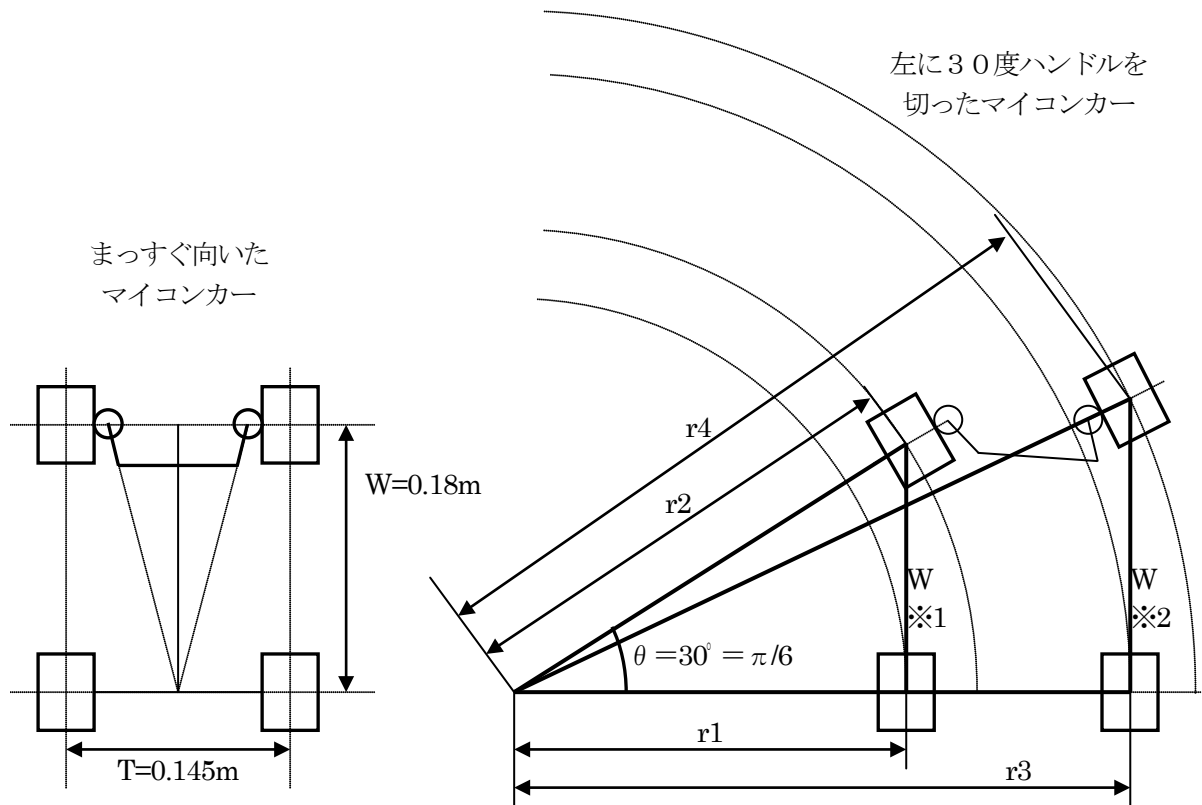
この問題を解決したのが、ドイツ人のアッカーマン、及びフランス人のジャントで、この機構をアッカーマン・ジャント方式、または単にアッカーマン方式と呼びます(下図)。



タイヤに角度を与える左右のナックルアームに開き角を付けていれば、サーボによりタイロッドが左右に動くときナックルアームの動きに差が出て、コーナ内側のタイヤが大きな角度になります。

6. 4 輪の回転数計算

ナックルアームの開き角度は、リアタイヤの中心部分で交わるようにします。ホイールベース、トレッドにより変わってくるので、マイコンカーに合わせて角度を決める必要があります。



T=トレッド…左右輪の中心線の距離（基板マイコンカーVer.2 では 0.145[m]です）

W=ホイールベース…前輪と後輪の間隔（基板マイコンカーVer.2 では 0.18[m]です）

角度 θ は、前輪内側タイヤの切れ角です。

※1…実際はホイールベースより短いですが、ほとんど変わらないので W とします。

※2…実際はホイールベースより長いですが、ほとんど変わらないので W とします。

図のように、後輪部の底辺 r_1 、高さ W 、角度 θ の三角形の関係は次のようです。

$$\tan \theta = W / r_1$$

角度 θ 、 W が分かっていますので、後輪内輪 r_1 が分かります。

$$r_1 = W / \tan \theta = 0.18 / \tan(\pi / 6) = 0.312[m]$$

後輪外輪の半径は、

$$r_3 = r_1 + T = 0.312 + 0.145 = 0.457[m]$$

また、前輪内径 r_2 、高さ W 、角度 θ の三角形の関係は次のようです。

$$\sin \theta = W / r_2$$

角度 θ 、 W が分かっていますので、前輪内輪 r_2 が分かります。

$$r_2 = W / \sin \theta = 0.18 / \sin(\pi / 6) = 0.360[\text{m}]$$

前輪外輪の半径 r_4 は、底辺と高さが分かっているので、ピタゴラスの定理より、

$$r_4 = \sqrt{(r_3^2 + W^2)} = \sqrt{(0.457^2 + 0.18^2)} = 0.491[\text{m}]$$

一番回転する r_4 を 100 としたときのそれぞれの回転数は、

$$\begin{aligned} r_1 : r_2 : r_3 : r_4 \\ &= 0.312 : 0.360 : 0.457 : 0.491 \\ &= 0.312 \times 100 / 0.491 : 0.360 \times 100 / 0.491 : 0.457 \times 100 / 0.491 : 0.491 \times 100 / 0.491 \\ &= 64 : 73 : 93 : 100 \end{aligned}$$

ハンドル角度 30 度、前輪外輪が 100 回転するとき、後輪外輪は 93 回転、前輪内輪は 74 回転、後輪内輪は 64 回転することになります。

7. ステアリングモータの角度指定

今までのステアリング制御は、センサ基板が常にコースの中心にくるような制御をしていました。ラジコンサーボのように、右に何度曲げたい、左に何度曲げたいというように制御したい場合、どうすればよいのでしょうか。

7.1 PD 制御

アナログセンサをPD制御したとき、アナログセンサの値が0になるようにステアリングモータを制御していました。これを、角度(ポテンシオメータのA/D値)にすれば良いだけです。

	アナログセンサの値にするとき	角度の値にするとき
比例制御	制御量 $P = k_p \times p$ k_p = 定数 p = 現在のアナログセンサ値 - 目標のアナログセンサ値 = <code>getAnalogSensor () - 0</code> = <code>getAnalogSensor ()</code>	制御量 $P = k_p \times p$ k_p = 定数 p = 現在の角度 - 目標の角度 = <code>getServoAngle () - iSetAngle</code> ※目標の角度を <code>iSetAngle</code> 変数に入れます
微分制御	制御量 $D = k_d \times d$ k_d = 定数 d = 過去のアナログセンサ値 - 現在のアナログセンサ値 = <code>iSensorBefore - getAnalogSensor ()</code>	制御量 $D = k_d \times d$ k_d = 定数 d = 過去の角度 - 現在の角度 = <code>iAngleBefore2 - getServoAngle ()</code>

7.2 プログラム

7.2.1 グローバル変数の追加

グローバル変数を追加します。

```

/* サーボ関係2 */
int      iSetAngle;           /* 設定したい角度(AD値)      */
int      iAngleBefore2;     /* 前回の角度保存           */
int      iServoPwm2;        /* サーボPWM値              */
    
```

7.2.2 関数の追加

ステアリングモータの角度指定用の servoControl2 関数を追加します。関数を追加したので、プロトタイプ宣言もしておきましょう。

今回は、比例定数 10、微分定数 50 にしています。この値は、ギヤのガタなどで違ってきますので各自調整してください。また、PWM 上限の 50%も、ステアリング動作が安定したら 95%程度に増やしてください。

```

/*****
/* モジュール名 servoControl2 */
/* 処理概要 サーボモータ制御 角度指定用 */
/* 引数 なし */
/* 戻り値 グローバル変数 iOutPwm2 に代入 */
*****/
void servoControl2( void )
{
    int i, j, iRet, iP, iD;

    i = iSetAngle; /* 設定したい角度 */
    j = getServoAngle(); /* 現在の角度 */

    /* サーボモータ用PWM値計算 */
    iP = 10 * (j - i); /* 比例 */
    iD = 50 * (iAngleBefore2 - j); /* 微分 */
    iRet = iP - iD;
    iRet /= 2;

    if( iRet > 50 ) iRet = 50; /* マイコンカーが安定したら */
    if( iRet < -50 ) iRet = -50; /* 上限を95くらいにしてください */
    iServoPwm2 = iRet;

    iAngleBefore2 = j;
}

```

7.2.3 割り込みプログラムの追加

割り込みプログラムに servoControl2 関数を追加して、1ms ごとに実行するようにします。

```

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

    asm(" fset I "); /* タイマ RB 以上の割り込み許可 */

    cnt1++;

    /* サーボモータ制御 */
    servoControl();
    servoControl2(); 追加
}

```

以下略

7. ステアリングモータの角度指定

7.2.4 使い方

main 関数内で使用するときは、

- iSetAngle 変数に、ステアリングモータで角度を指定したい A/D 値を代入します。
- プログラムは、「servoPwmOut(iServoPwm²);」を実行します。「servoPwmOut(iServoPwm);」とすると、センサ基板がコースの中心になるようなステアリング制御になります。2 が付くか付かないかの違いです。

下記に、main プログラムの最初で角度指定した例を示します。このプログラムできちんと角度指定できているか iSetAngle 変数の値を変えて実験してみましょう。

なお、元々のプログラムの iAngle0 変数の設定は走行開始直前なので、iAngle0 の設定をいちばん最初に行います。

```

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

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

    /* マイコンカーの状態初期化 */
    motor_mode_f( BRAKE, BRAKE ); /* 基板マイコンカーの FREE は、 */
    motor_mode_r( BRAKE, BRAKE ); /* PWM 値に関係なく必ずフリーになる */
    motor_f( 0, 0 );
    motor_r( 0, 0 );
    servoPwmOut( 0 );

    cnt1 = 0;
    while( cnt1 <= 10 );
    iAngle0 = getServoAngle(); /* 0 度の位置記憶 */

    iSetAngle = 32; /* 1 度当たり 3.18 なので、32 で約 10 度 */
    while( 1 ) {
        servoPwmOut( iServoPwm2 );
    }
}

```

以下略

角度指定が正しくできたことを確認できたら、実際の走行プログラムに組み込んでみましょう。下記プログラムは新しくパターン 52 を作り、A/D 値が-50 になるような位置にステアリングモータを移動させる例です。

```

case 52:
    iSetAngle = -50;
    servoPwm( iServoPwm2 );
    break;

```


8. プログラムの調整ポイント

本プログラムは、ゆっくり(1m/s程度)で走行できるようになっています(レーンチェンジ部分は除く)。速度を上げたときの調整ポイントを解説します。

行	内容	説明
89~99	内輪の PWM 値	今回のプログラムは後輪駆動として、内輪差を計算しています。基板マイコンカーVer.2は4輪駆動なので、「角度計算(4WD時).xls」を使用して、4輪の PWM 値を計算して motor 関数で設定すると、4輪が角度に合った回転数になります。
155~170	ステアリング角度と駆動モータの PWM 値	今回のプログラムは、角度が①35 以上 ②5~35 ③-35 以下 ④-5~-35 ⑤それ以外(-5~5) で駆動モータの PWM を調整しています。もっと細かく調整してみましょう。 また、速度が速いときの 5~10 度、遅いときの 5~10 度、というように速度も使って PWM 値を調整してみましょう。
193	クロスライン検出後の速度	今回のプログラムでは 1m/s にしています。それ以上速くても直角を曲がれるか実験して、曲がれるならその速度にしてみましょう。 また、1m/s 以上の速度なら PWM を 0% にしていますが、クロスラインから直角までの距離が 50cm だと、減速しきれません。PWM 値をマイナスにして逆転ブレーキをかけてみましょう。
217 241	右クランク検出時のステアリングを曲げる PWM 値、 左クランク検出時のステアリングを曲げる PWM 値	右クランク検出時、右にステアリングを曲げる PWM 値を設定します。今回は 50% です。弱いと曲げるスピードが遅くなり、外側にふくらんで脱輪してしまいます。また 40 度になるとステアリング部分が車体にぶつかって、ステアリングモータがロックしてしまいます。ぶつかる直前にステアリングモータの PWM を弱めましょう。 左クランク検出時も同様です。
218,219 242,243	右クランククリア時の PWM 値、 左クランククリア時の PWM 値	計算値では、左前:60%、右前:35%、左後:48%、右後:24% ですが、実際は違う値の方が右クランクが曲がりやすいかもしれません。いろいろな PWM を試してみましょう。例えば、内輪の後輪の PWM を小さくした方が曲がりやすいかもしれません。 左クランククリア時も同様です。
869,870	ステアリングモータに加える PWM の上限設定	ステアリングモータに加える PWM の上限を設定しています。サンプルプログラムは、50% 以上にならないようにしています。この数値が小さいと、せっかくの PD 制御もこの部分で制限されてしまうのでステアリングモータの反応が遅くなってしまいます。大きすぎると万が一、ステアリング部分が本体にぶつかってロックしてしまったときにステアリングモータに大電流が流れてしまいます。最初は 50% として、ライントレースが安定したら 90% 程度にしてください。 例) 869 : if(iRet > 90) iRet = 90; 870 : if(iRet < -90) iRet = -90;

9. 参考文献

- ルネサス エレクトロニクス(株)
R8C/35C グループ ハードウェアマニュアル Rev.1.00
- ルネサス エレクトロニクス(株)
M16C シリーズ,R8C ファミリー用 C/C++コンパイラパッケージ V.6.00
C/C++コンパイラユーザーズマニュアル Rev.1.00
- ルネサス エレクトロニクス(株)
High-performance Embedded Workshop V.4.09 ユーザーズマニュアル Rev.1.00
- ルネサス半導体トレーニングセンター C言語入門コーステキスト 第1版
- 電波新聞社 マイコン入門講座 大須賀威彦著 第1版
- ソフトバンク(株) 新C言語入門シニア編 林晴比古著 初版
- 共立出版(株) プログラマのための ANSI C 全書 L.Ammeraal 著
吉田敬一・竹内淑子・吉田恵美子訳 初版

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

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

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

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

の製品情報にある「マイコン」→「R8C」でご覧頂けます