アナログセンサ基板 TypeS
モータドライブ基板 TypeS
プログラム解説マニュアル

H8/3048F-ONE 版

第 1.13 版

2010.09.02

ジャパンマイコンカーラリー実行委員会

注 意 事 項 (rev.3.0J)

著作権

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

禁止事項

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

転載、複製

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

責任の制限

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

その他

本マニュアルに記載の情報は本マニュアル発行時点のものであり、ジャパンマイコンカーラリー実行委員会は、予告なしに、本マニュアルに記載した情報または仕様を変更することがあります。製作に当たりましては、最新の内容を確認いただきますようお願いします。

連絡先

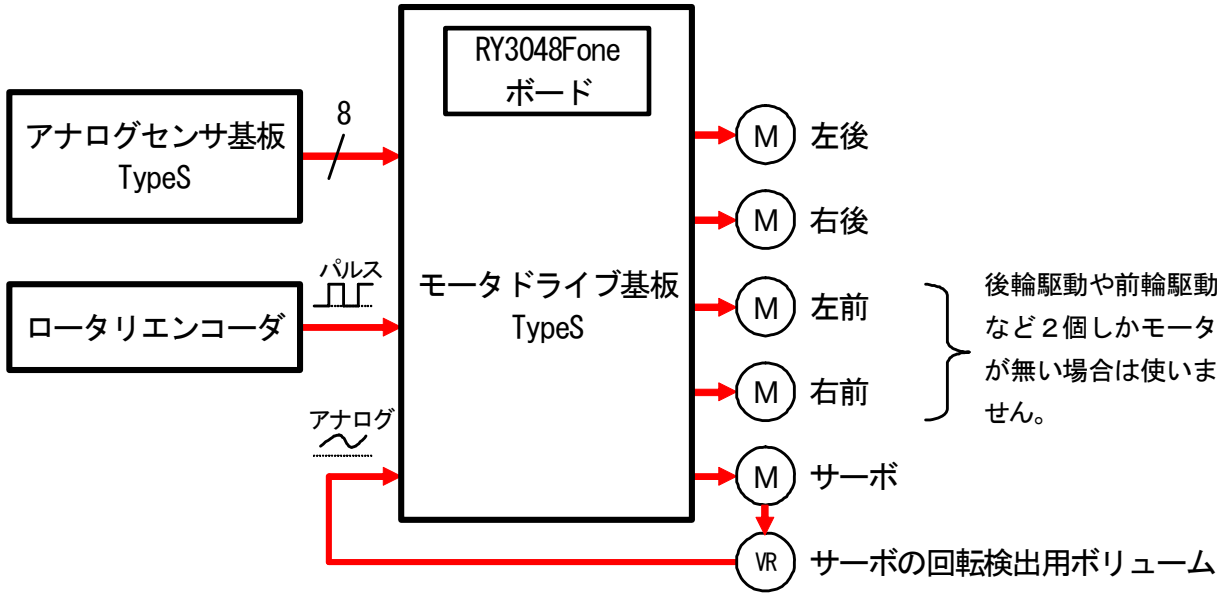
(株)ルネサスソリューションズ　ルネサスマイコンカーラリー事務局
〒162-0824　東京都新宿区揚場町 2-1　軽子坂MNビル
TEL　(03)-3266-8510
E-mail:official@mcr.gr.jp

目　次

1. 仕様

1.1 構成

本マニュアルは、下記構成のマイコンカーを対象に説明しています。



- モータドライブ基板 TypeS 使用
- アナログセンサ基板 TypeS 使用
- サーボモータ(ステアリング機構)の回転検出ボリューム搭載済み
- ロータリエンコーダ搭載済み(1 回転あたりのパルス数と距離はプログラム内で設定します)

1.2 参照

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。

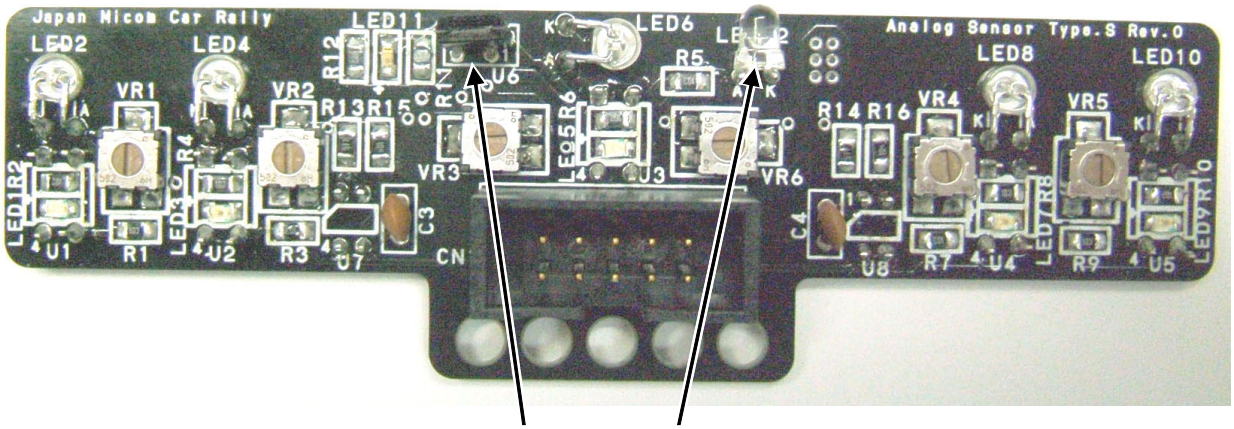
基板、機器名	キット、製作についてのマニュアル	プログラムについてのマニュアル
モータドライブ基板 TypeS	モータドライブ基板 TypeS 製作マニュアル	本マニュアル
アナログセンサ基板 TypeS	アナログセンサ基板 TypeS 製作マニュアル	本マニュアル
ロータリエンコーダ	ロータリエンコーダ Ver.2 製作マニュアル	ロータリエンコーダ実習マニュアル kit07 版
モータドライブ基板 TypeS に実装されている 24C256 について	モータドライブ基板 TypeS 製作マニュアル	データ解析実習マニュアル kit07 版

2. アナログセンサ基板TypeS

2.1 概要

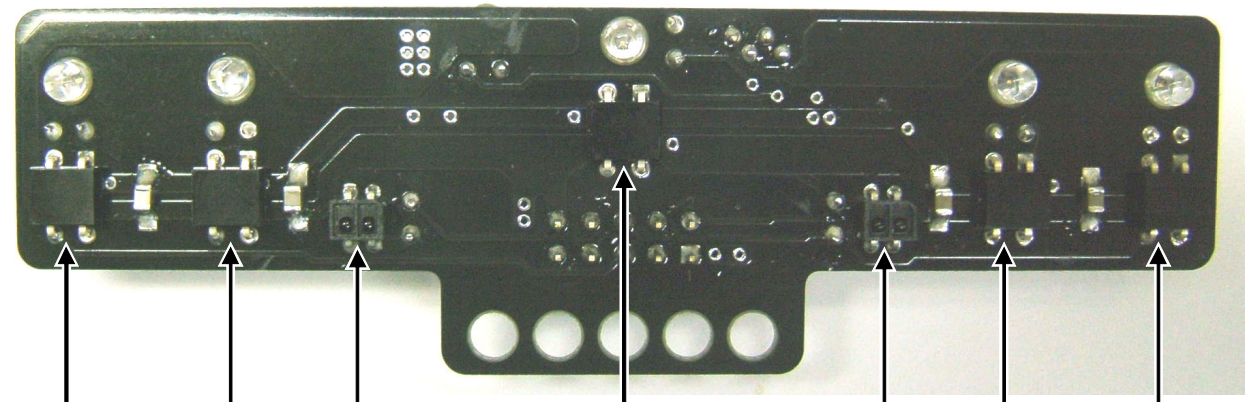
アナログセンサ基板 TypeS は、コースを見るアナログセンサ 2 個、コースを見るデジタルセンサ 5 個、スタートバーを見るデジタルセンサ 1 個を搭載した基板です。

部 品 面



スタートバー検出センサ(左が受光側、右が発光側)

半 田 面



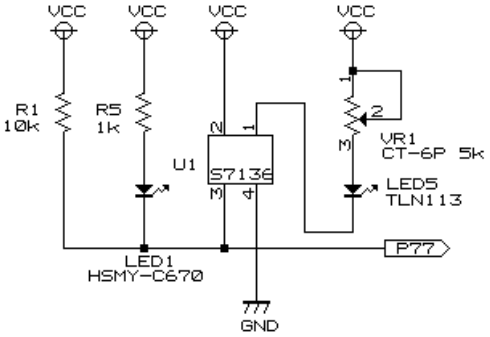
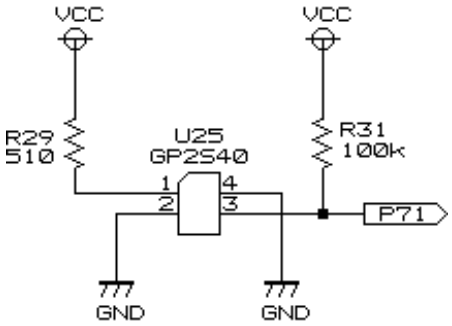
右端デジタル　右中デジタル　右アナログ　中心デジタル　左アナログ　左中デジタル　左端デジタル

※半田面は、裏から見ているため左右が逆になります。

詳しくは、アナログセンサ基板 TypeS 製作マニュアルを参照してください。

2.2 デジタルセンサとアナログセンサ

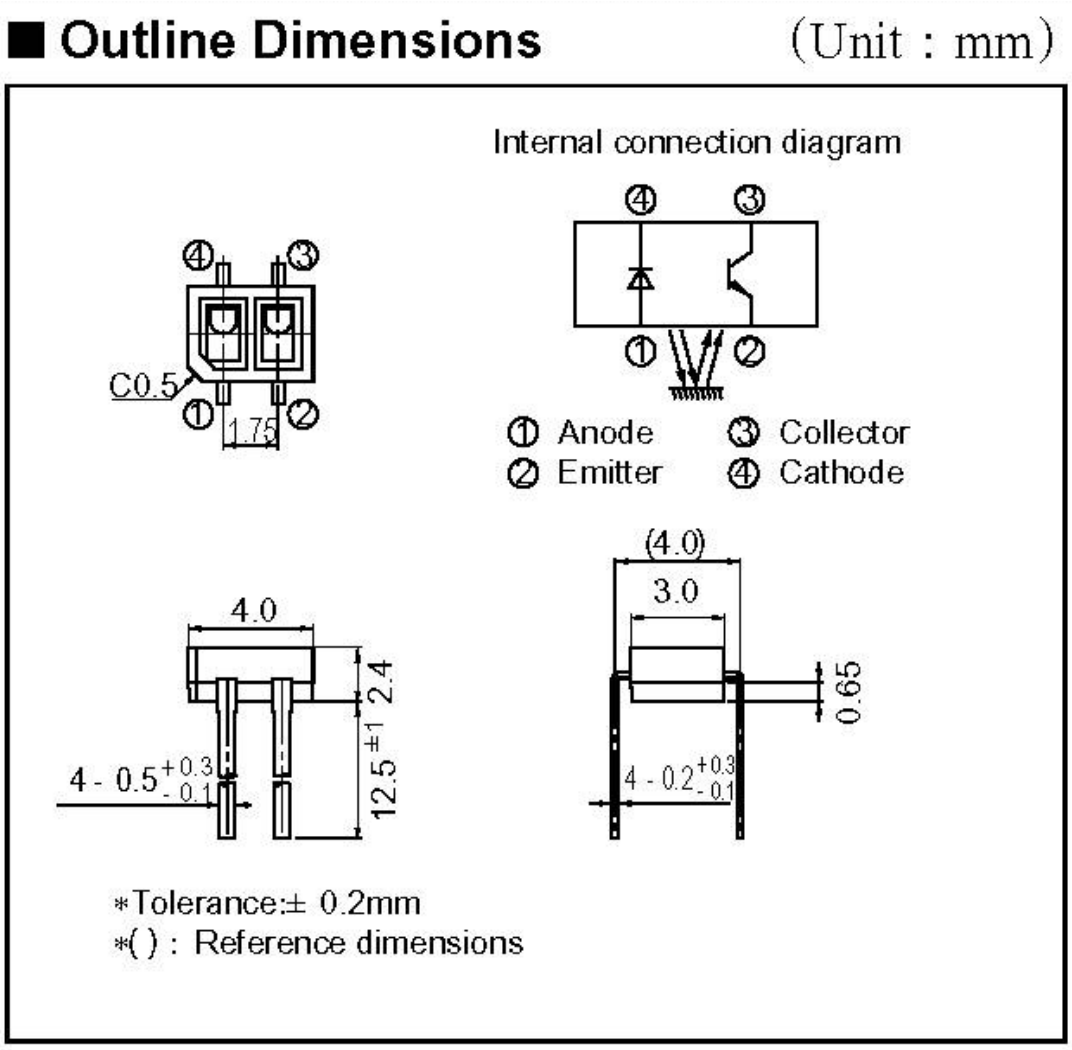
マイコンカーで使用する場合の、デジタルセンサとアナログセンサの特徴を下記に示します。

項目	デジタルセンサ	アナログセンサ
回路例	 センサのピン振り 1:赤外 LED のカソード　2:＋電源 3:出力　4:GND	 センサのピン振り 1:アノード　2:エミッタ 3:コレクタ　4:カソード
センサ出力	センサ下部が白色(灰色)か黒色かの判断	センサ下部が、白色か灰色か黒色か、さらに黒に近い灰色か、白に近い灰色かなど、細かく検出可能
外乱	強い、S7136 は特に強い	非常に弱い
コースとの間隔	2mm～10mm、幅が広い	約 2～4mm　常に一定にする必要がある
ポート	センサからの信号はデジタル出力なので、H8 マイコンのどのポートでも入力可能	センサからの信号はアナログ出力なので、H8 の AN 端子(ポート7)のみで入力可能

アナログセンサはデジタルセンサに比べ、外乱に弱いですがコースの状態を細かく知ることができます。この情報をうまく使えばステアリング制御を非常に細かくできるため、ライントレースを非常に滑らかに行うことができます。

2.3 アナログセンサで使用する素子

アナログセンサ基板 TypeS では、アナログセンサとしてシャープ(株)製の「**GP2S40**」という**フォトインタラプタ**を使用しています。外形は下記のとおりです。



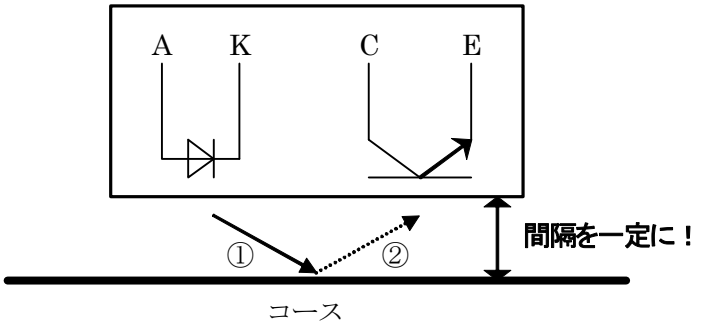
※GP2S40 のデータシートより抜粋

4.0×3.0mm 角の中に発光部である赤外 LED と、受光部であるフォトトランジスタが内蔵されており、非常にコンパクトです。この素子をアナログセンサ基板 TypeS の半田面に取り付けます。

詳しくは、
http://www.sharp.co.jp/products/device/lineup/data/pdf/datasheet/gp2s40_j.pdf
にデータシートがあります。

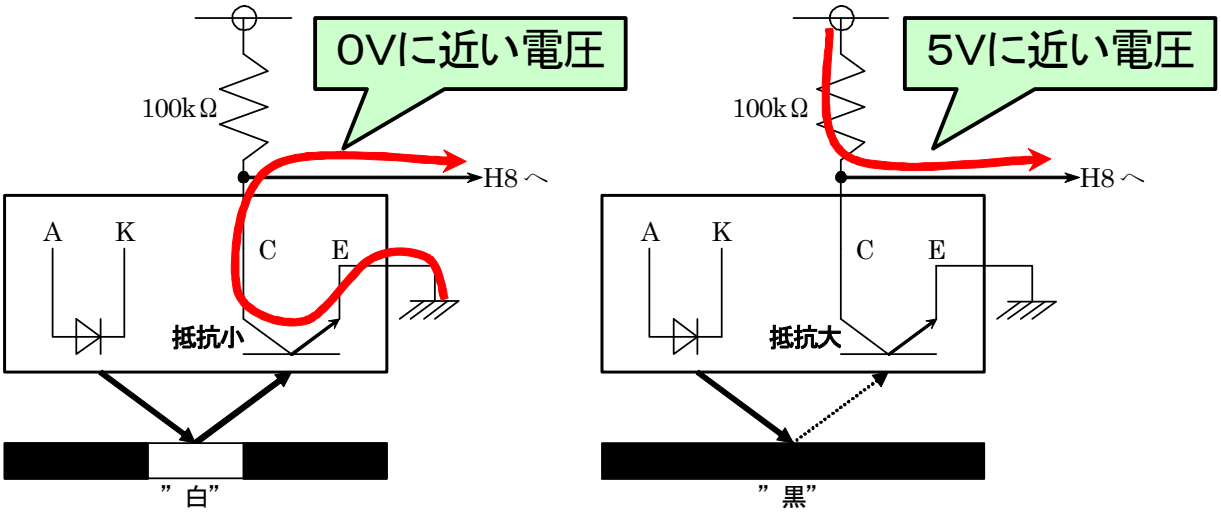
2.4 アナログセンサの動作原理

アナログセンサが、どのように白色、灰色、黒色を認識するかを説明します。



① 赤外 LED 側から赤外線を出します。
② コースに反射した光をフォトトランジスタで受けます。

光は、白色は反射、黒色は吸収することを利用します。センサ下部が白色なら赤外 LED から出た光は多く反射して、フォトトランジスタに届きます。電圧出力は、エミッターコレクタ間の抵抗が少なるため低くなります。センサ下部が黒色なら赤外 LED から出た光は吸収されてしまい、フォトトランジスタにあまり届きません。電圧出力はエミッターコレクタ間の抵抗が大きくなり、コレクタに接続されているプルアップ抵抗を通して高くなります。



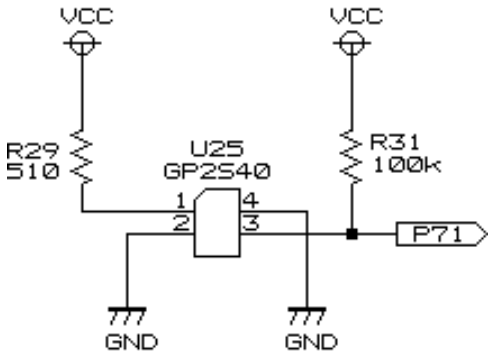
2.5 H8 マイコンで電圧を取り込む

H8/3048F-ONE には 10bit 精度の A/D 変換器が内蔵されています。これは、0～5V(CPU ボードの電源電圧)の電圧を 0～1023(2^{10}-1)の値に変換することができます。
アナログ入力端子のポイントは、下記のとおりです。

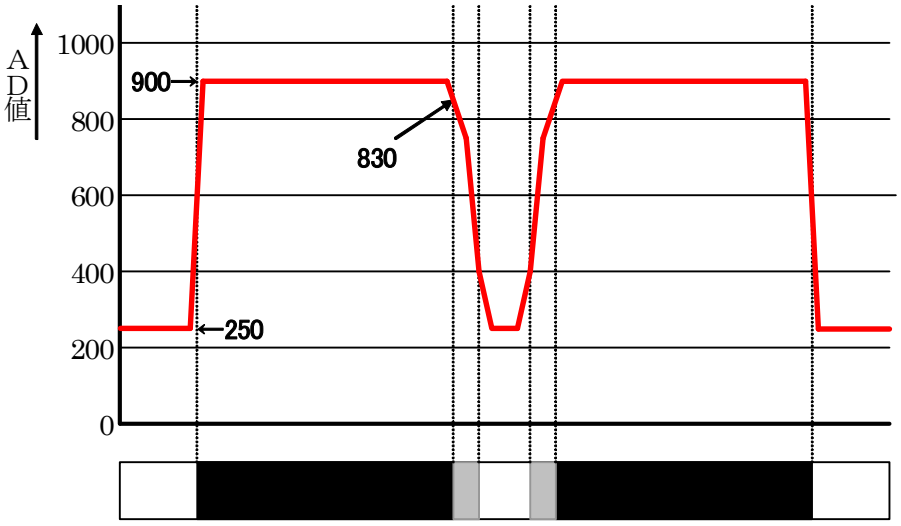
- **アナログ入力端子は、ポート 7 の bit0～7 の 8 端子ある**
- 0～5V(CPU ボードの電源電圧)の電圧を、0～1023(2^{10}-1)の値に変換する
- 1 度に A/D 変換できるのは 1 端子のみ、どの端子電圧を A/D 変換するかはプログラムで設定する

A/D 変換について詳しくは、「H8/3048F-ONE 実習マニュアル」のプロジェクト「ad」を参照してください。

2.6 コースの状態を取り込む



上記のような回路を組み、コースとセンサの間隔を約 3mm 一定にしてコースの端から端までずらしていき、そのときの A/D 取得値を簡単なグラフにしてみました。



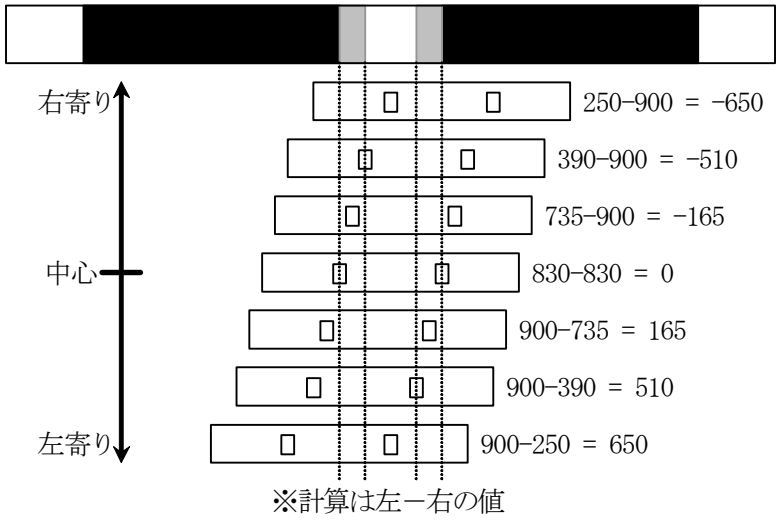
白色が 250、黒色が 900、中心の黒、灰、白部分は 900 から 250 へ変化していきます。正確には比例していませんが、ほぼ比例していると考えて差し支えありません。

マイコンカーはコース中心の白色と灰色をトレースするので、この部分を詳しく見てみます。コース中心部分は 250、ずれるにしたがって値が大きくなり、黒色部分では約 900 になります。A/D 値をチェックすることにより、中心部分のずれを細かく知ることができます。
ただし、1 つ問題があります。例えば黒色と灰色のちょうど境目は 830 くらいですが、数字を見ただけでは右側

か左側か分かりません。アナログセンサ 1 個では右にずれているのか左にずれているのか分からないのです。
そこで、アナログセンサを 2 個、40mm の間隔で取り付け、次の計算を行います。

センサの値 ＝ 左センサの値 － 右センサの値

この計算を行うことによりセンサの値は、左側にずれているなら正の数、右側なら負の数となり、左右のどちら側に寄っているのか分かります。



今回の例では、センサの値が−650 から 650 まで変化します。センサの状態が正の数なら左に寄っていますのでハンドルを右へ、負の数なら右へ寄っていますのでハンドルを左へ曲げます。また値の大きさで、どのくらいの強さで曲げるかを調整することができます。例えば 50 なら弱く曲げる、500 なら強く曲げる、というように制御します。

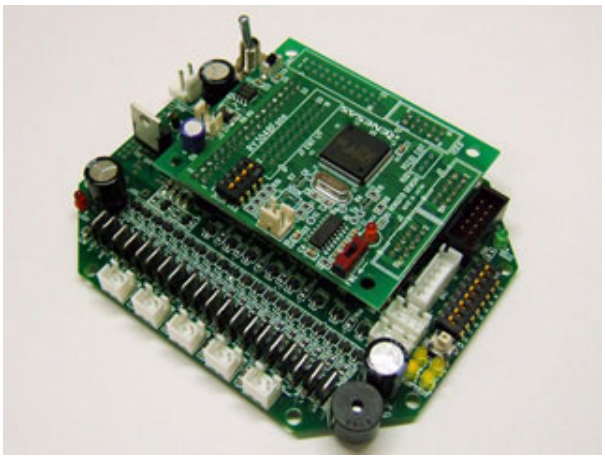
3. モータドライブ基板TypeS

3.1 概要

モータドライブ基板 TypeS は、モータを 5 個制御することのできるモータドライブ基板です。仕様を下表にまとめます。

	モータドライブ基板 TypeS	モータドライブ基板 Vol.3 (参考)
対象	既にもの作りを経験されている方が対象	すべての方が対象
部品数	リード線のある部品:約 54 個、 表面実装部品:約 121 個、 部品のピンの間隔は最小 1.27mm	リード線のある部品:約 52 個 表面実装部品:0 個 部品のピンの間隔は 2.54mm 以上
RY3048Fone ボードの改造	既存 10 ピンコネクタの取り外し、10 ピンメスコネクタの取り付け	なし
RY3048Fone ボードとの接続方法	ドライブ基板 S の上に重ねる	10 芯フラットケーブルにより接続
制御できるモータ数	**5 個** **自作サーボ、左前、右前、左後、右後の各モータ**	2 個 左モータ、右モータ
制御できる ラジコンサーボ	なし	1 個
入力電圧	7.2V 以上(単三電池 6 本～8 本)	5V 以上(単三電池 4 本～8 本) ただし 5 本以上の電圧を加える場合、 LM350 追加セットの追加が必要です
プッシュスイッチ	1 個	1 個
ディップスイッチ	8bit	なし
プログラムで 点灯、消灯できる LED	4 個	2 個
リミットスイッチなどの 接点入力回路	4 個分(CN8)	なし
エンコーダ入力回路	あり(CN15)	なし
ボリューム入力回路	あり(CN9)	なし
ブザー	あり (周波数は固定です、圧電ブザーではありません)	なし
EEP-ROM	あり(32KB)	なし
センサ基板の 信号入力コネクタ	あり アナログセンサ基板 TypeS、またはセンサ基板 Ver.4	なし ※ドライブ基板 3 の場合は CPU ボードにセンサ基板を接続します
基板外形	110×90×厚さ 1.6mm	80×75×厚さ 1.6mm
重量(基板のみ)	約 30g	約 15g
完成時の寸法(実寸)	幅 110×奥行き 90×高さ 35mm ※スイッチを除くと高さ 22mm	幅 80×奥行き 65×高さ 20mm
重量(完成品の実測)	約 65g (CPU ボードは除く) ※リード線の長さや半田の量で変わります ※参考:CPU ボード込みで実測 89g	約 33g ※リード線の長さや半田の量で変わります

詳しくは、アナログセンサ基板 TypeS 製作マニュアルを参照してください。



▲モータドライブ基板 TypeS

3.2 ラジコンサーボと自作サーボ

マイコンカーで使用する場合の、ラジコンサーボと自作サーボの特徴を下記に示します。

項目	ラジコンサーボ	自作サーボ
制御周期	サーボに加える PWM 周期が制御周期 標準的なサーボで 16ms、 デジタルサーボで 5ms 秒速 4m/s なら 16ms で 64mm 進んでしまう	プログラムで可変可能 サンプルプログラムは 1ms 秒速 4m/s なら 1ms で 4mm しか進まない！
モータドライブ回路	サーボに内蔵のため不要	必要
プログラム	PWM のデューティ比を変えるだけ、簡単	現在の角度と目標の角度から、加える PWM のデューティ比を計算、調整が難しい
現在の角度検出	できない ただし別途ボリューム、またはロータリエンコーダを付けることにより可能	ボリューム、またはロータリエンコーダ必要
モータ	サーボに内蔵のため不要 逆に言えば選べない	自分で選定する
ギヤ比	サーボ固有のギヤ比	自分で組む必要があるが、 組み方により自由に設定できる

アナログセンサと自作サーボは基本的にペアで使用します。それは、「**制御周期**」に関わりがあります。

せっかくアナログセンサで中心からのずれが細かく分かっても、サーボを制御する間隔が長ければ意味がありません。そこで自由に制御周期を設定できるように、サーボ機構を自作します。

自作サーボはギヤ比、モータ、制御周期を自分で選定できるため、高価なラジコンサーボ以上の性能を出すことも可能です。

※サーボについて

サーボは、「物体の位置、方位、姿勢など(機械量)を制御量とし、目標値の任意の変化に追従するように構成された制御系。」(出典:Wikipedia)という意味です。

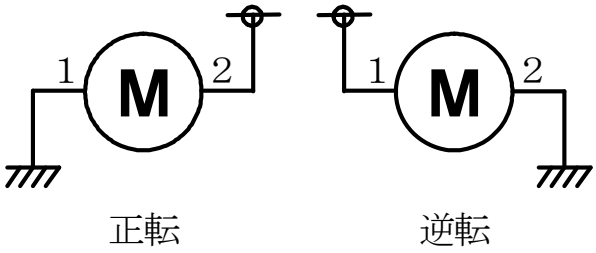
本書では、ラジコン屋さんで販売されているPWMを加えると自動で動くサーボを**ラジコンサーボ**、マイコンで直接モータを制御するサーボを**自作サーボ**と使い分けています。どちらもサーボではありますが、マイコンでの制御方法が大きく異なります。

3.3 モータドライブ回路

3.3.1 モータの回し方(電圧と動作の関係)

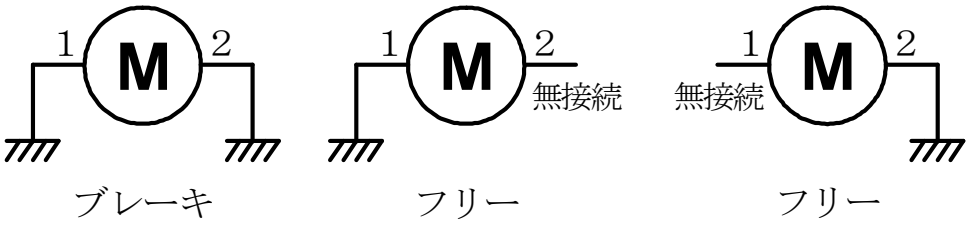
マイコンカーを制御するには、モータを「正転、逆転、停止」させる必要があります。これらの状態は、モータの端子 1、端子 2 に加える電圧を変えることにより制御します。

動作	端子 1	端子 2
正転	GND 接続	+接続
逆転	+接続	GND 接続
停止	後述	

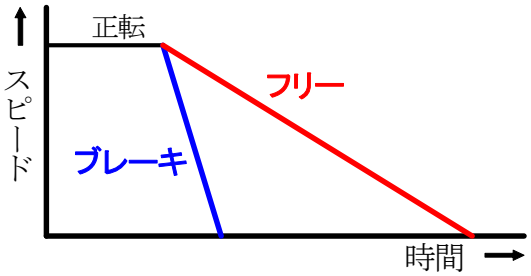


停止には、ブレーキとフリーの2種類あります。
ブレーキは、端子間をショートさせモータの発電作用(逆起電圧)を利用しモータを素早く止める方法です。
フリーは、モータの端子 1、または端子 2 のどちらか(または両方)を無接続にすることにより、惰性でモータの回転が遅くなる動作をいいます。

動作	端子 1	端子 2
ブレーキ	GND 接続	GND 接続
フリー	＋接続または GND 接続	無接続
フリー	無接続	＋接続または GND 接続



正転状態からブレーキ動作にしたとき、フリー動作にしたときのスピードの落ち方の違いを下図に示します。

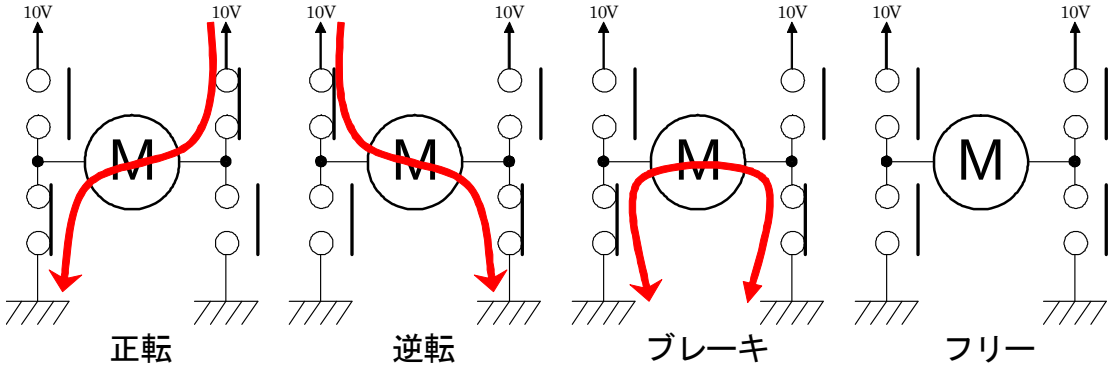


フリーはブレーキと比べ、スピードの減速が緩やかです。フリーは、スピードをゆっくり落としたい場合などに使用します。

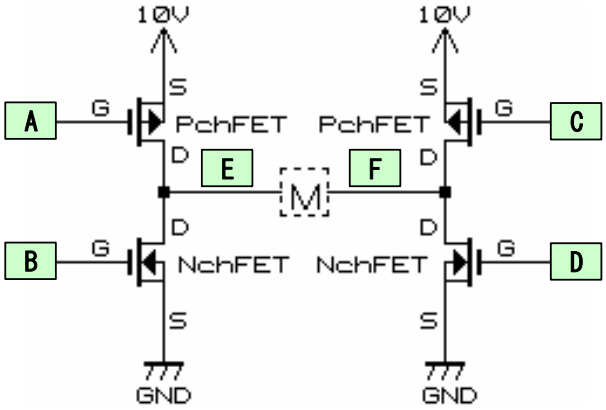
3.3.2 Hブリッジ回路

モータを正転、逆転、ブレーキ、フリーにするにはどうすればよいのでしょうか。下図のように、モータを中心としてH型に4つのスイッチを付けます。その形から「**Hブリッジ回路**」と呼ばれています。

この4つのスイッチをそれぞれON／OFFすることにより、正転、逆転、ブレーキ、フリー制御を行います。



3.3.3 スイッチをFETにする



実際の回路では、前記のスイッチを FET で行います。電源のプラス側に P チャネル FET(2SJ タイプ)、マイナス側に N チャネル FET(2SK タイプ)を使用します。

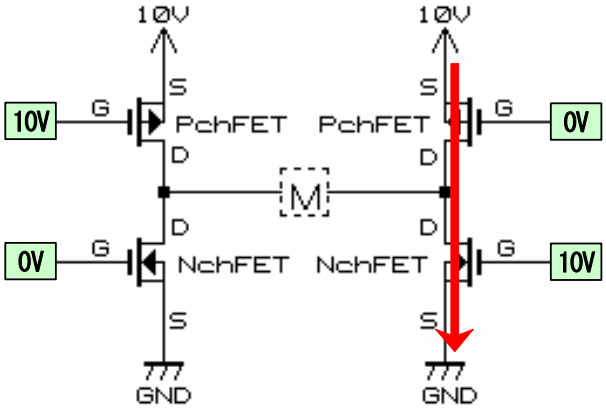
P チャネル FET は、V_G(ゲート電圧)＜V_S(ソース電圧)のとき、D-S(ドレイン－ソース)間に電流が流れます。

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

これら4つの FET のゲートに加える電圧を変えることにより、正転、逆転、ブレーキ、フリーの動作を行います。下表に FET A～D のゲートに 0V または 10V を加えたときの動作を示します。

A	B	C	D	FET A の動作	FET B の動作	FET C の動作	FET D の動作	E の電圧	F の電圧	モータ動作
0V	0V	0V	0V	ON	OFF	ON	OFF	10V	10V	ブレーキ
0V	0V	0V	10V	ON	OFF	ON	ON	10V	**ショート!**	設定不可
0V	0V	10V	0V	ON	OFF	OFF	OFF	10V	フリー	フリー
0V	0V	10V	10V	ON	OFF	OFF	ON	10V	0V	**逆転**
0V	10V	0V	0V	ON	ON	ON	OFF	**ショート!**	10V	設定不可
0V	10V	0V	10V	ON	ON	ON	ON	**ショート!**	**ショート!**	設定不可
0V	10V	10V	0V	ON	ON	OFF	OFF	**ショート!**	フリー	設定不可
0V	10V	10V	10V	ON	ON	OFF	ON	**ショート!**	0V	設定不可
10V	0V	0V	0V	OFF	OFF	ON	OFF	フリー	10V	フリー
10V	0V	0V	10V	OFF	OFF	ON	ON	フリー	**ショート!**	設定不可
10V	0V	10V	0V	OFF	OFF	OFF	OFF	フリー	フリー	フリー
10V	0V	10V	10V	OFF	OFF	OFF	ON	フリー	0V	フリー
10V	10V	0V	0V	OFF	ON	ON	OFF	0V	10V	**正転**
10V	10V	0V	10V	OFF	ON	ON	ON	0V	**ショート!**	設定不可
10V	10V	10V	0V	OFF	ON	OFF	OFF	0V	フリー	フリー
10V	10V	10V	10V	OFF	ON	OFF	ON	0V	0V	ブレーキ

「設定不可」の部分は、ショート状態となるため、設定してはいけません。例えば、A=10V、B=0V、C=0V、D=10V のとき、下図のように左側の P チャネル FET と N チャネル FET が 10V から 0V まで直接つながり、ショートと同じ状態になってしまいます。



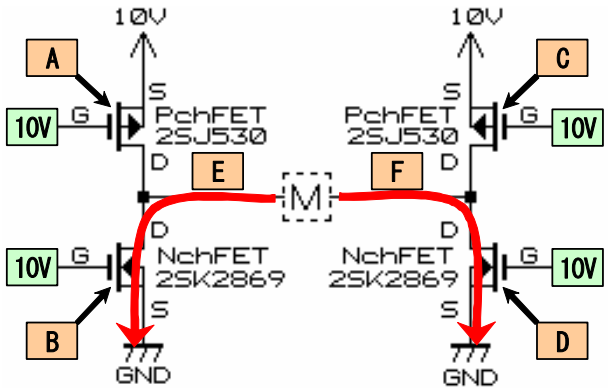
3.3.4 スピード制御

正転するスピードを変えたい場合、「正転→ブレーキ→正転→ブレーキ…」を高速に繰り返します。今回のサンプルプログラムは、「正転→ブレーキ」を 1ms 間で行い、正転とブレーキの割合を変えることでスピードを変えます。

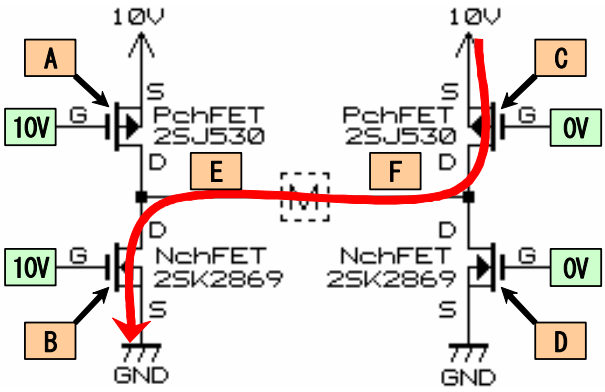
正転とブレーキを繰り返すときの 4 つの FET のゲート電圧は、下表のようになります。

A	B	C	D	FET A の動作	FET B の動作	FET C の動作	FET D の動作	E の電圧	F の電圧	モータ動作
10V	10V	10V	10V	OFF	ON	OFF	ON	0V	0V	ブレーキ
10V	10V	0V	0V	OFF	ON	ON	OFF	0V	10V	正転

●ブレーキ動作



●正転動作



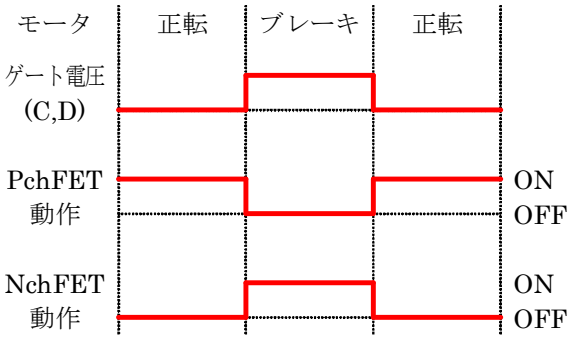
表より FET C と FET D のゲート電圧を、0V と 10V に変えればよいことが分かります。

3.3.5 正転とブレーキの切り替え時にショートしてしまう

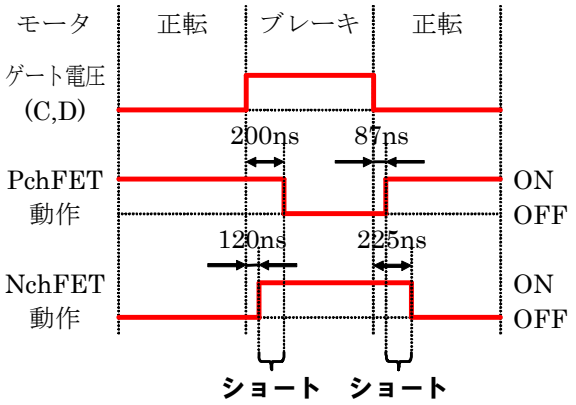
前記の回路を実際に組んで動作させると、FET が非常に熱くなりました。どうしてでしょうか。

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

理想的な波形



実際の波形



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

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

参考までにルネサス エレクトロニクス(株)製の FET 2SJ530 と 2SK2869 の電気的特性を下記に示します。

2SJ530(P チャネル)

電気的特性

(Ta=25℃)

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

注) 4. パルス測定

OFF→ON は 87ns 遅れる

ON→OFF は 200ns 遅れる

2SK2869(Nチャネル)

電気的特性

(Ta=25℃)

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

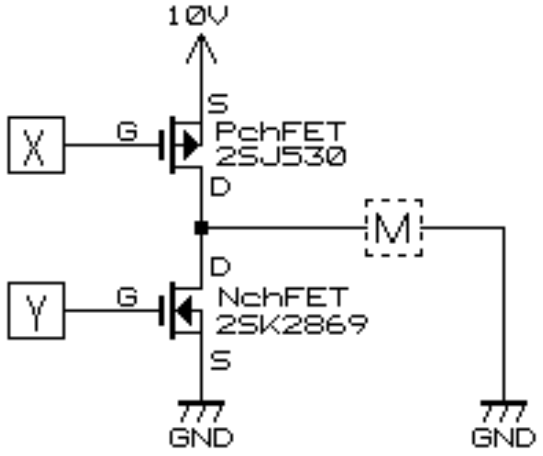
注) 1. パルス測定

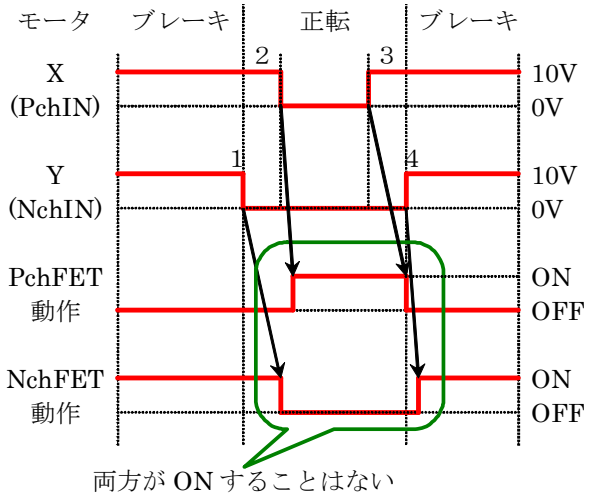
OFF→ON は 120ns 遅れる

ON→OFF は 225ns 遅れる

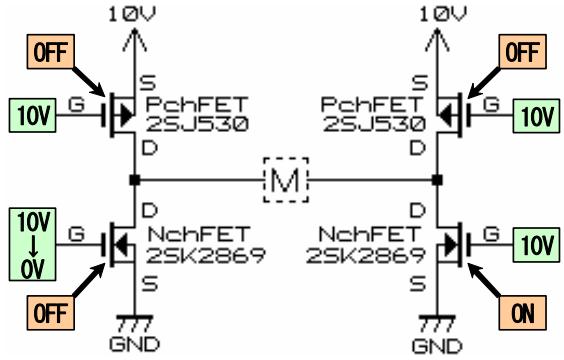
3.3.6 PチャネルとNチャネルの短絡防止回路

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



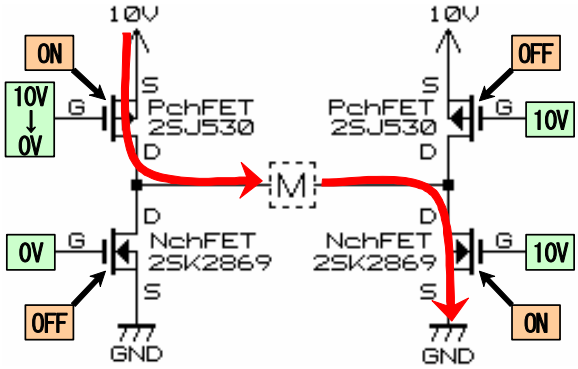


(1) NチャネルFETをOFFにする(フリー状態)



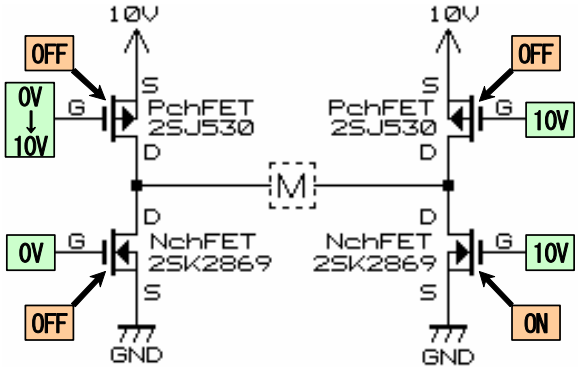
P チャネル FET を ON する前より先に、N チャネル FET のゲート電圧を 10V→0V にします。225ns 後に OFF になります。フリー状態です。

(2) PチャネルFETをONにする(正転状態)



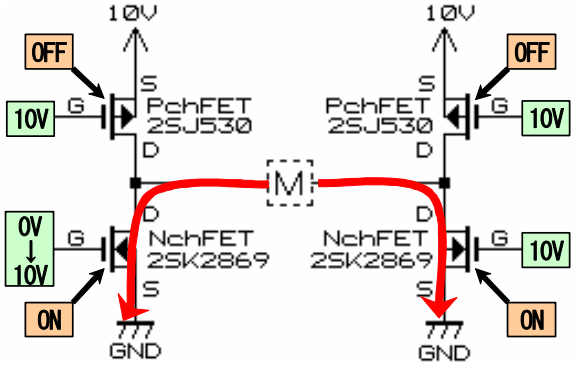
次に、P チャネル FET のゲート電圧を 10V→0V にします。87ns 後に ON します。正転状態です。

(3) PチャネルFETをOFFにする(フリー状態)



次に、P チャネル FET のゲート電圧を 0V→10V にします。200ns 後に OFF します。フリー状態です。

(4) NチャネルFETをONにする(ブレーキ状態)



次に、N チャネル FET のゲート電圧を 0V→10V にします。120ns 後に ON します。ブレーキ状態です。

3.3.7 H8/3048F-ONEマイコンでのPチャネルとNチャネルの短絡防止

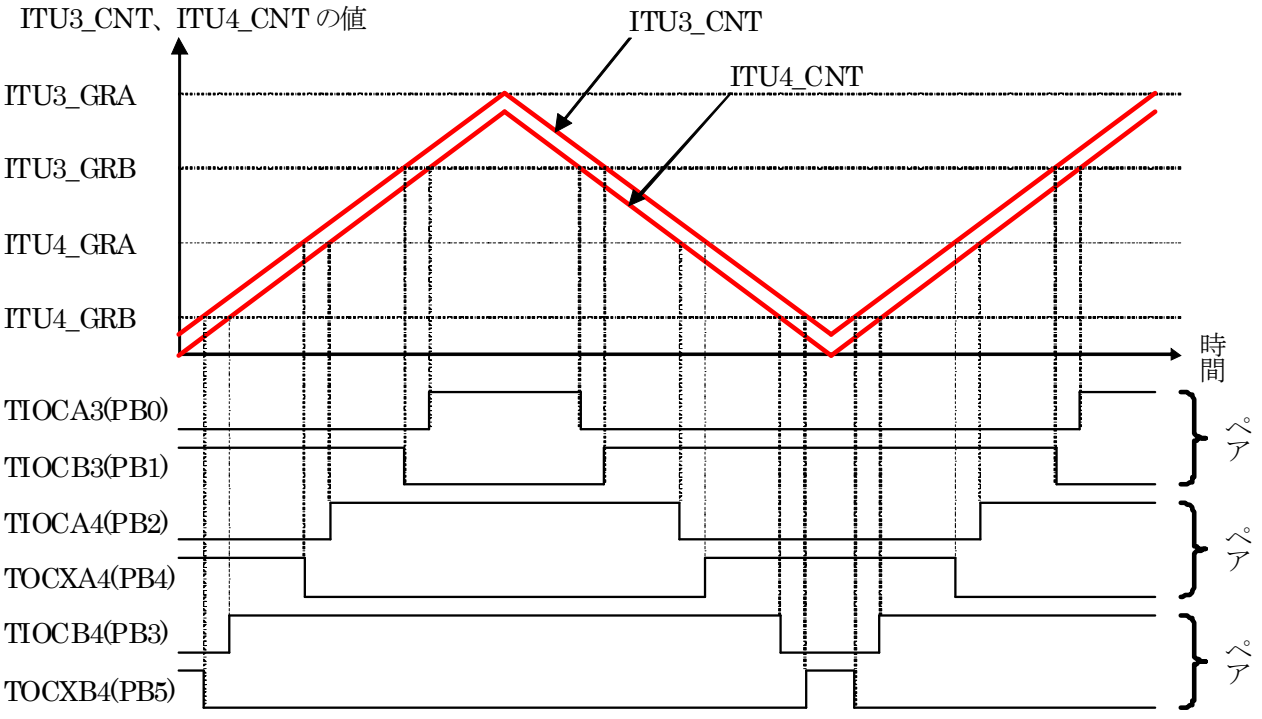
モータドライブ基板 TypeS は、短絡防止回路(プログラム)が 2 種類あります。

モータ	CN	短絡防止方法
左後モータ	CN10	H8/3048F-ONE の相補 PWM モードを使った対策(ITU3,4 を使用)
右後モータ	CN11	H8/3048F-ONE の相補 PWM モードを使った対策(ITU3,4 を使用)
サーボモータ	CN12	H8/3048F-ONE の相補 PWM モードを使った対策(ITU3,4 を使用)
左前モータ	CN13	遅延回路を使った対策(コンデンサ、抵抗を使用)
右前モータ	CN14	遅延回路を使った対策(コンデンサ、抵抗を使用)

5 個あるモータのうち、3 個は H8/3048F-ONE の相補 PWM モードという機能を使い短絡防止を行います。左前モータ、右前モータは、遅延回路を使い短絡防止をしています。遅延回路は、モータドライブ基板 Vol.3 でも使用しています。詳しくは、プログラム解説マニュアル kit07 版、またはそれに関わるマニュアルを参照してください。

3.3.8 相補PWMモード

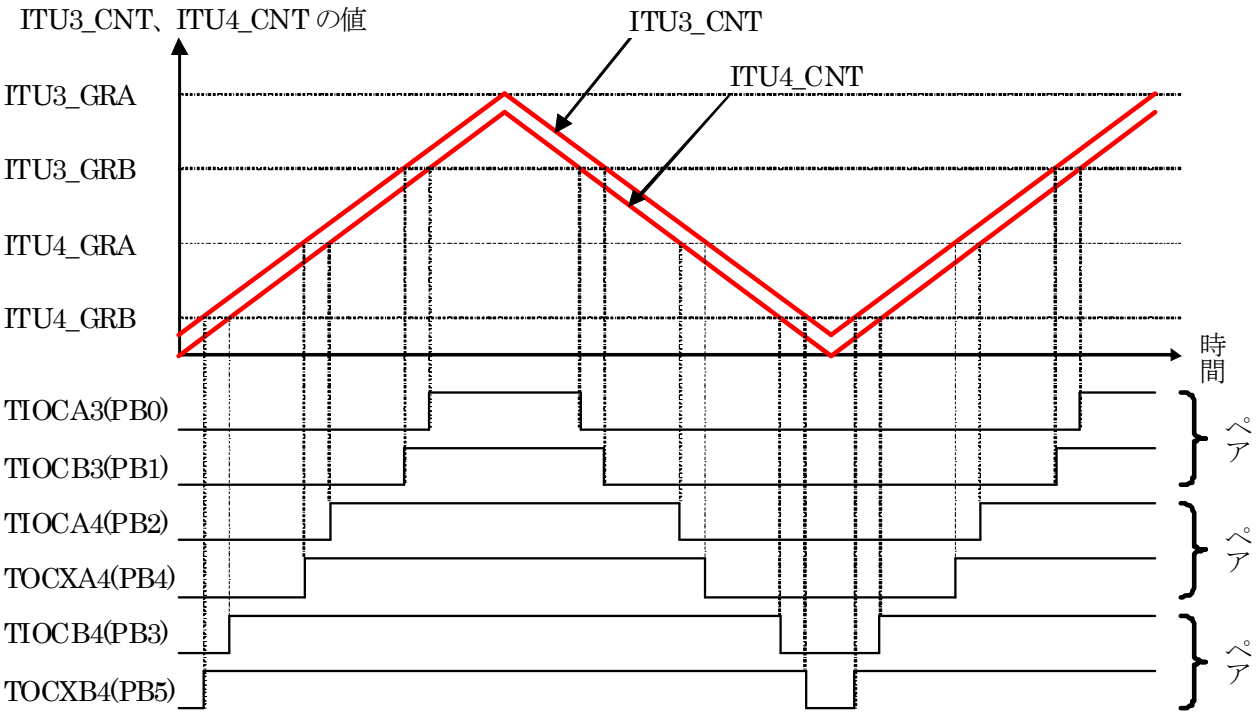
H8/3048F-ONE の ITU には、相補 PWM モードというモードがあります。リセット同期式 PWM モードと同じように ITU3 と ITU4 をペアにして使い、下記のように通常の波形と少しずれた波形を 1 セットとして、3 セットの PWM 波形を出力することができます。下記に、標準の相補 PWM モードの波形を示します。



ITU3_GRA で周期を決めます。ITU3_GRB、ITU4_GRA、ITU4_GRB のレジスタで各波形のデューティ比を決め

ます。

PB1、PB4、PB5 から出力される波形が必要としている波形とは逆になっています。これはプログラムで変えることができ、PB1、PB4、PB5 を反転させます。下記に、反転後の相補 PWM モードの波形を示します。反転方法は後述します。



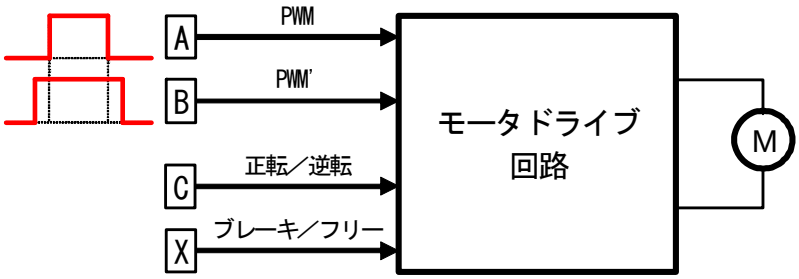
▲反転後の波形

モータドライブ基板 TypeS では、下記のように PWM 出力端子を割り振っています。

端子名	詳細
TIOCA3(PB0)	右後モータ用(CN11)の P チャネル FET
TIOCB3(PB1)	右後モータ用(CN11)の N チャネル FET
TIOCA4(PB2)	左後モータ用(CN10)の P チャネル FET
TOCXA4(PB4)	左後モータ用(CN10)の N チャネル FET
TIOCB4(PB3)	サーボモータ用(CN12)の P チャネル FET
TOCXB4(PB5)	サーボモータ用(CN12)の N チャネル FET

このように、P チャネル FET と N チャネル FET のゲートに加える波形を遅らせる方法として、H8/3048F-ONE の相補 PWM モードを使用します。

3.3.9 マイコンのポート割り振り



※ PWM'=PWM より立ち上がりが早く、立ち下がりの早いパルス

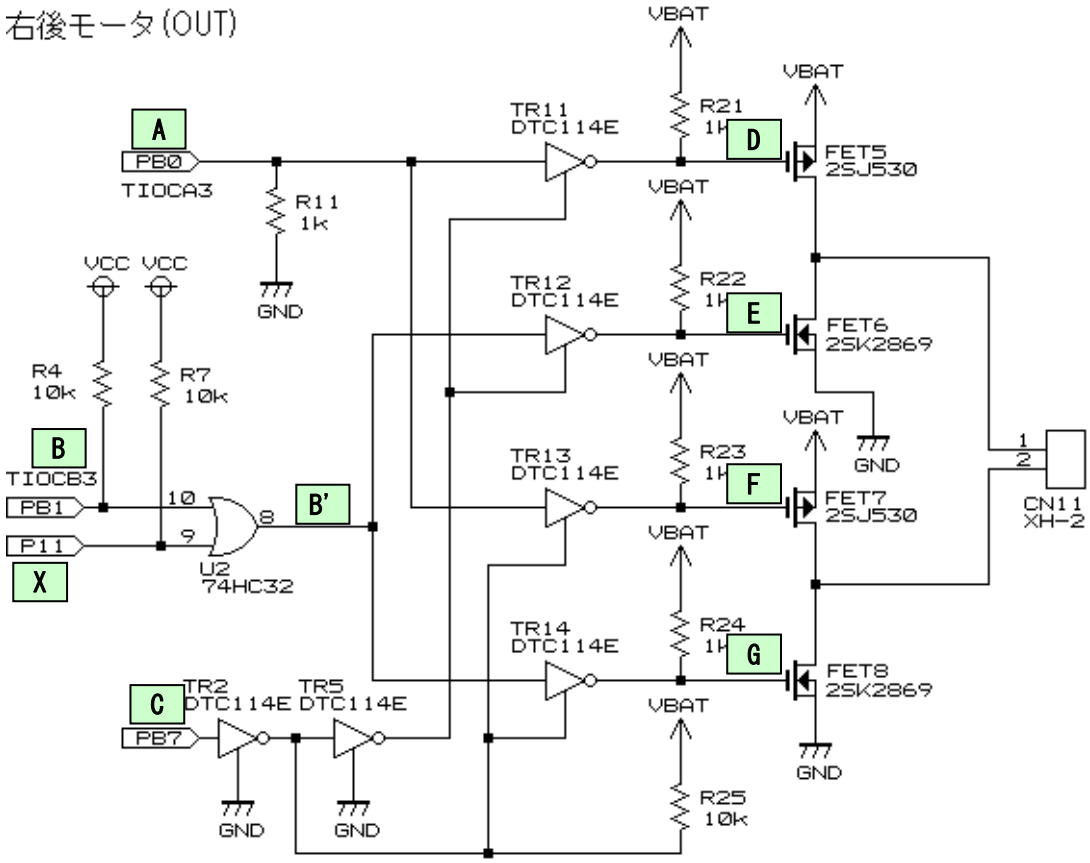
各モータを制御するマイコンのポートを下表に示します。

モータ	コネクタ	PWM 端子	PWM'端子	正転／逆転 切り替え端子	フリー／ブレーキ 切り替え端子
		A	**B**	**C**	**X**
左後	CN10	PB2	PB4	PB6	P10
右後	CN11	PB0	PB1	PB7	P11
サーボ	CN12	PB3	PB5	PA7	P37
左前	CN13	PA2	CR 回路で作成	PA1	P12
右前	CN14	PA4	CR 回路で作成	PA6	P13

左後モータ、右後モータ、サーボモータは各 4bit 分、左前モータ、右前モータは各 3bit 分の端子を使っています。すべて出力端子にします。

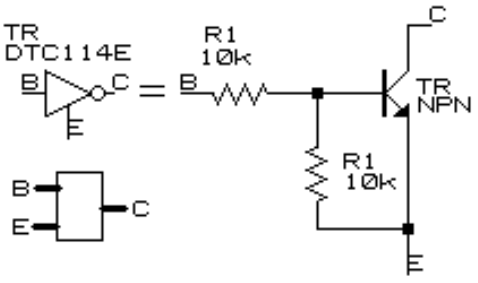
3.3.10 実際の回路

右後モータ回路を例に説明します。PB0(A)、PB1(B)が相補 PWM モードの PWM 出力端子、PB7(C)が正転／逆転切り替え端子、P11(X)がブレーキ／フリー切り替え端子です。



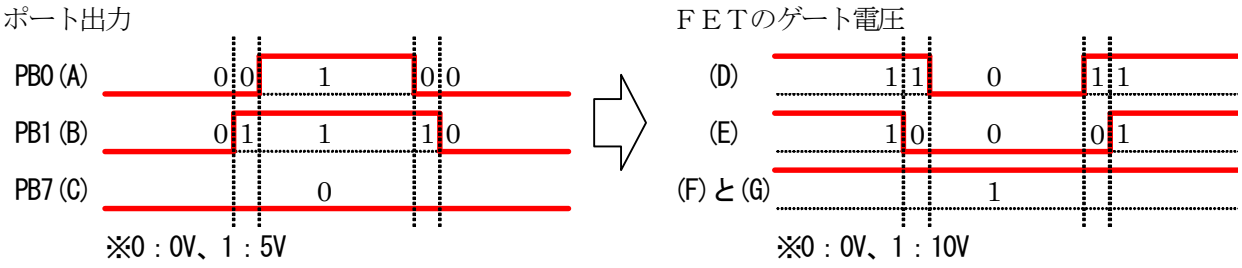
※ VBAT = 10V とする
※ B' = B or X

DTC114E はデジタルトランジスタと呼び、下記のようにトランジスタ 1 個と抵抗 2 個が内蔵された回路です。



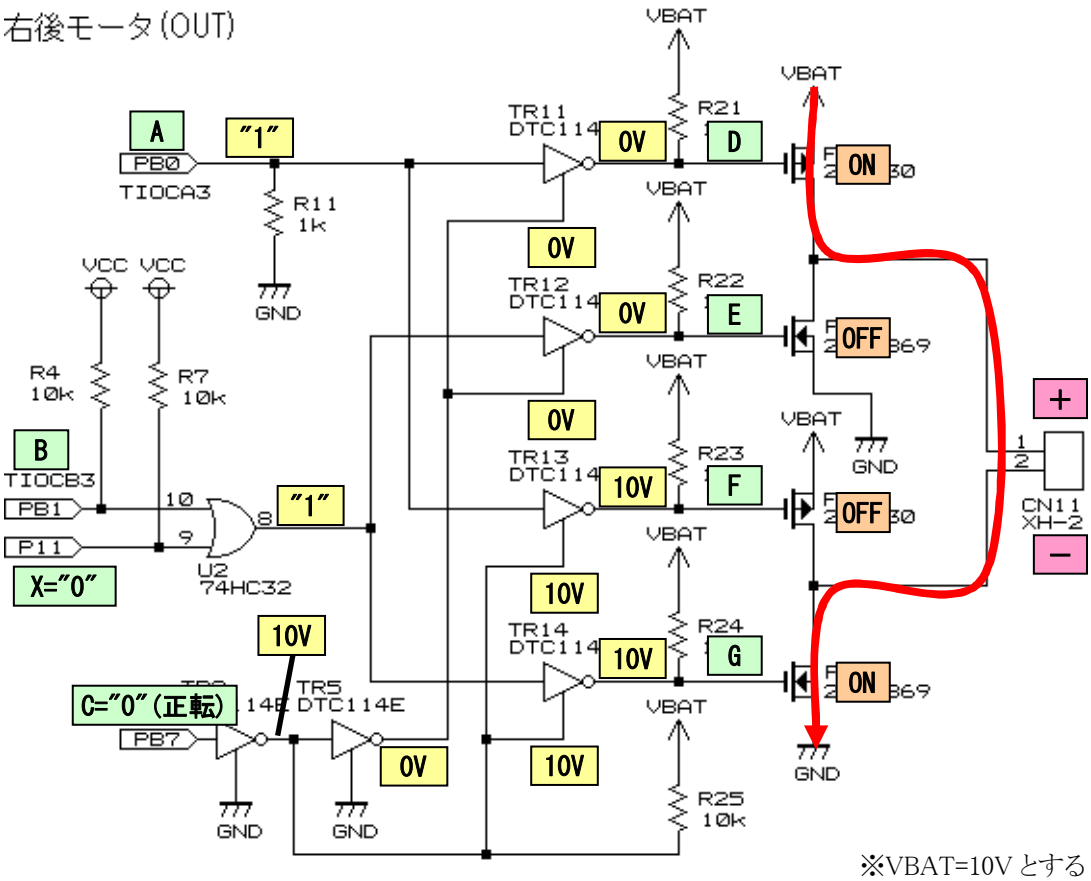
3.3.11 正転、ブレーキ時の動作

正転時(PB7="0")、PB0、PB1 端子に 1 パルス入ったときの波形と各端子の状態は下記のようになります。



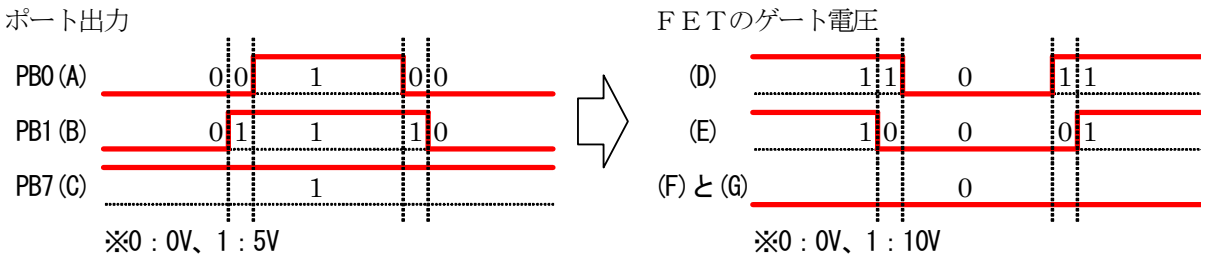
A	B	C	D	E	F	G	モータ動作
0 (0V)	0 (0V)	0 (0V)	10V (OFF)	10V (ON)	10V (OFF)	10V (ON)	**ブレーキ** 両端子 GND
0 (0V)	1 (5V)		10V (OFF)	0V (OFF)	10V (OFF)	10V (ON)	**フリー** 無接続状態
1 (5V)	1 (5V)		0V (ON)	0V (OFF)	10V (OFF)	10V (ON)	**正転** CN11.1=10V CN11.2=0V
0 (0V)	1 (5V)		10V (OFF)	0V (OFF)	10V (OFF)	10V (ON)	**フリー** 無接続状態
0 (0V)	0 (0V)		10V (OFF)	10V (ON)	10V (OFF)	10V (ON)	**ブレーキ** 両端子 GND

A="1"、B="1"、C="0"のときの動作を下記に示します。



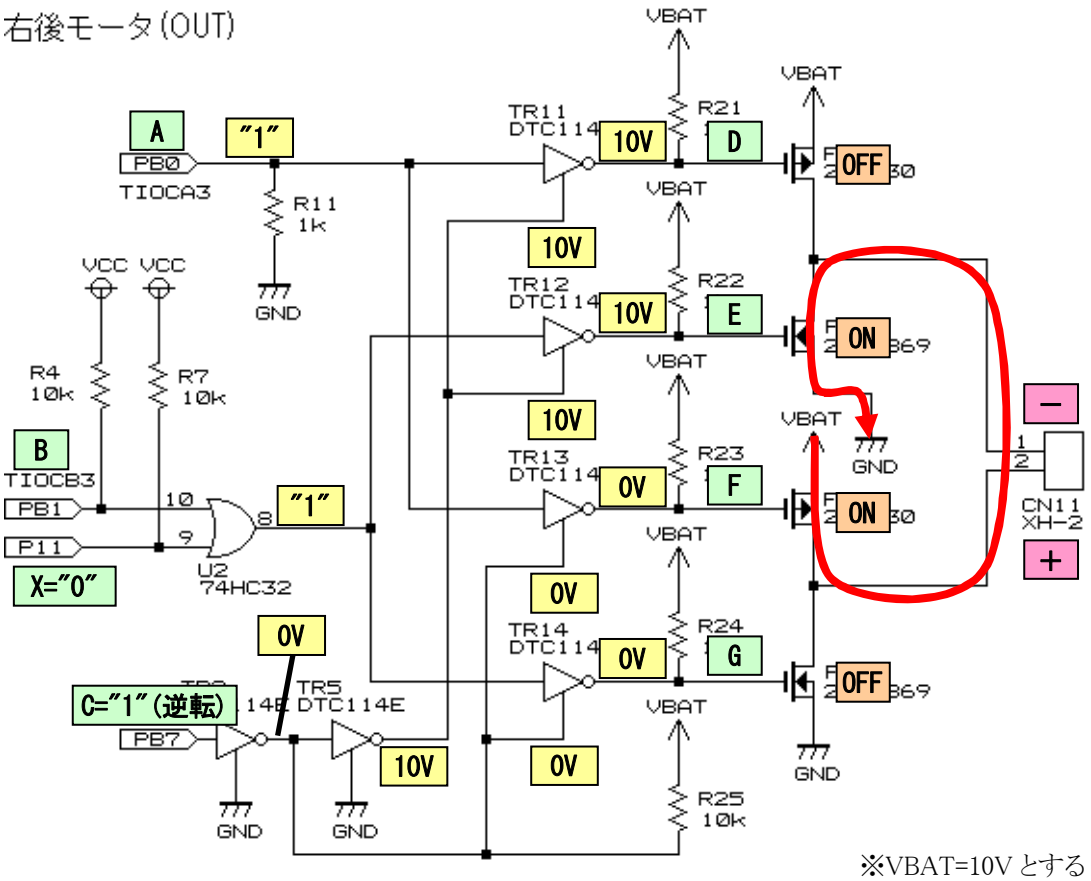
3.3.12 逆転、ブレーキ時の動作

逆転時(PB7="1")、PB0、PB1 端子に 1 パルス入ったときの波形と各端子の状態は下記のようになります。



A	B	C	D	E	F	G	モータ動作
0 (0V)	0 (0V)		10V (OFF)	10V (ON)	10V (OFF)	10V (ON)	**ブレーキ** 両端子　GND
0 (0V)	1 (5V)		10V (OFF)	10V (ON)	10V (OFF)	0V (OFF)	**フリー** 無接続状態
1 (5V)	1 (5V)	1 (5V)	10V (OFF)	10V (ON)	0V (ON)	0V (OFF)	**逆転** CN11.1=0V CN11.2=10V
0 (0V)	1 (5V)		10V (OFF)	10V (ON)	10V (OFF)	0V (OFF)	**フリー** 無接続状態
0 (0V)	0 (0V)		10V (OFF)	10V (ON)	10V (OFF)	10V (ON)	**ブレーキ** 両端子　GND

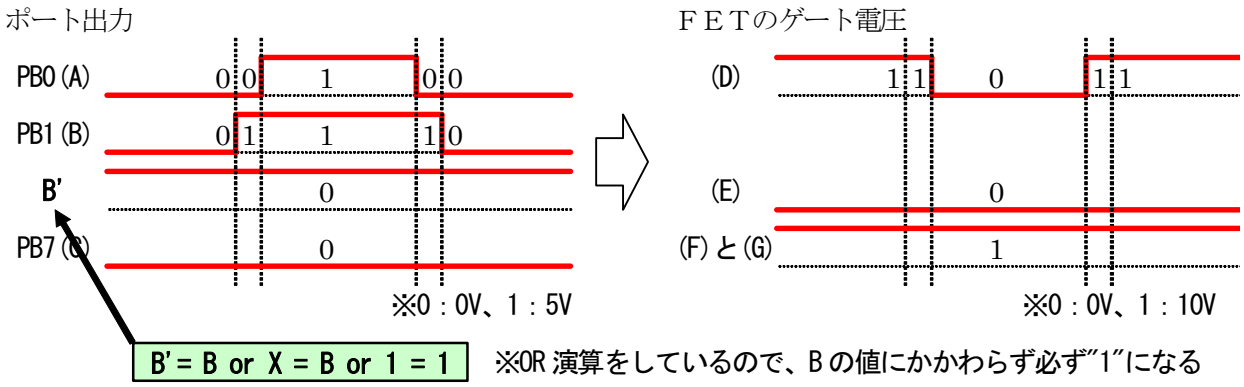
A="1"、B="1"、C="1"のときの動作を下記に示します。



※VBAT=10V とする

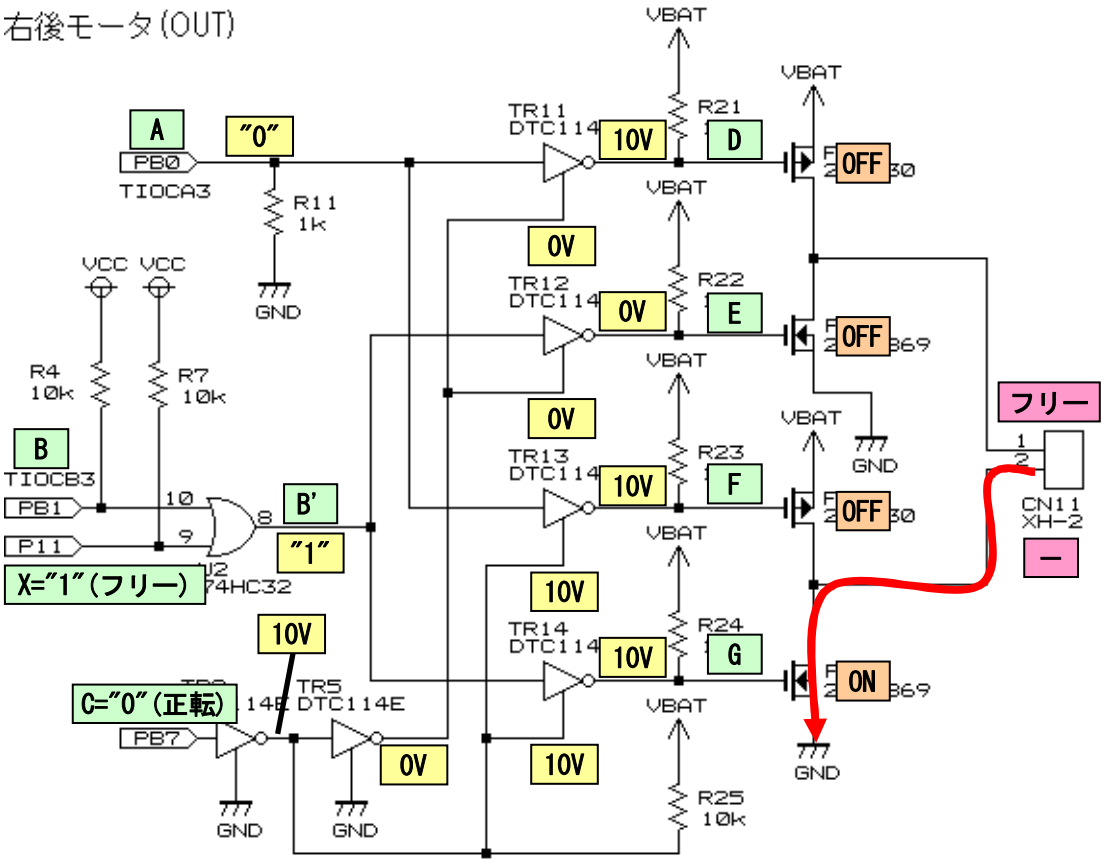
3.3.13 正転、フリー時の動作

正転時(PB7="0")、PB0、PB1 端子に 1 パルス入ったときの波形と各端子の状態は下記のようになります。



A	B'	C	D	E	F	G	モータ動作
0 (0V)	1 (5V)	0 (0V)	10V (OFF)	0V (OFF)	10V (OFF)	10V (ON)	**フリー** 無接続状態
1 (5V)	1 (5V)		0V (ON)	0V (OFF)	10V (OFF)	10V (ON)	**正転** CN11.1=10V CN11.2=0V
0 (0V)	1 (5V)		10V (OFF)	0V (OFF)	10V (OFF)	10V (ON)	**フリー** 無接続状態

A="0"、B="1"、C="0"、X="1"のときの動作を下記に示します。



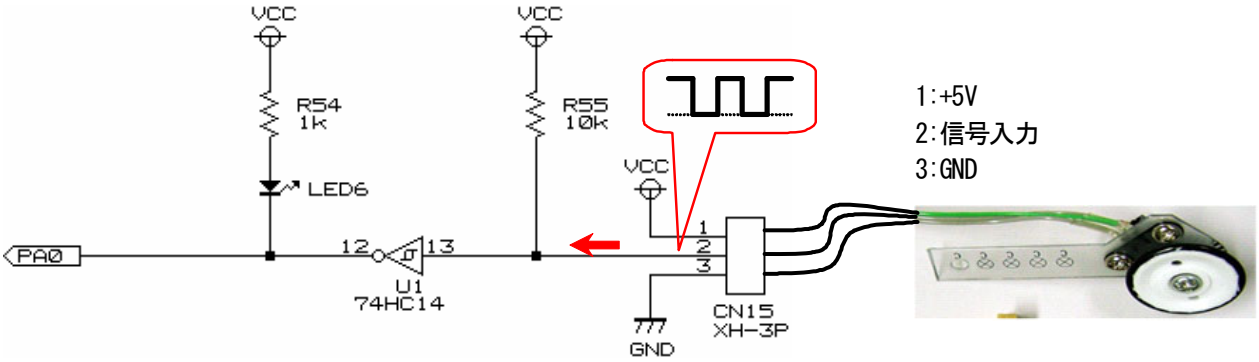
※VBAT=10V とする

3.4 ロータリエンコーダ信号入力回路

モータドライブ基板 TypeS には、ロータリエンコーダの信号を入力するコネクタがあります。ロータリエンコーダは別売りです。本基板のロータリエンコーダ回路の特徴は、下記の内容です。

- ロータリエンコーダを接続する専用コネクタ CN15 が実装済み
- 入力信号 10kΩでプルアップ済み、オープンコレクタ信号でも外付け抵抗の取り付け必要なし
- ロータリエンコーダから入力された信号をシュミット・トリガ NOT 回路(74HC14)にて波形整形
- ロータリエンコーダの信号を LED6 にて確認可能
- ロータリエンコーダのパルス入力端子はマイコンの PA0 端子

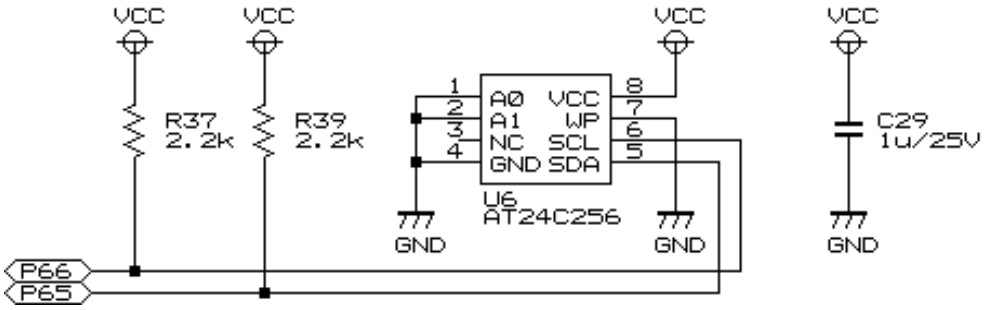
詳しくは、ロータリエンコーダ実習マニュアル kit07 版を参照してください。



3.5 EEP-ROM制御回路

モータドライブ基板 TypeS には、24C256 という EEP-ROM が実装されています。本基板の回路、サンプルプログラムの特徴は、下記の内容です。

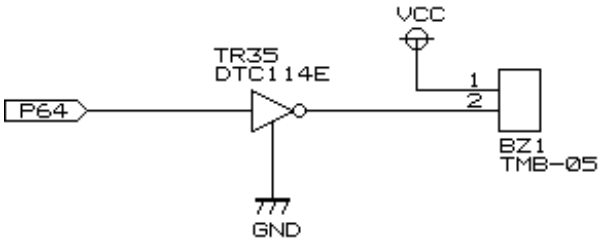
- EEP-ROM と H8 マイコンは、P66 と P65 で接続
- メモリ容量は、32KB(2^{15}=32768 バイト)
- 10ms ごとに 16 バイトのデータを保存(サンプルプログラムは 12 バイト使用、4 バイトは予備)
- 保存内容は、パターン番号、デジタルセンサ値、アナログセンサ値、角度、モータの回転数、エンコーダ値(詳しくは、後述のプログラムを参照してください)
- 保存時間は、メモリ容量 32768bytes÷1 回に保存する容量 16bytes×保存間隔 10ms＝20.48 秒



3.6 ブザー回路

モータドライブ基板 TypeS には、TMB-05 というブザーが実装されています。本基板の回路、サンプルプログラムの特徴は、下記の内容です。

- P64 端子を"1"にするとデジタルトランジスタを介して、ブザーに 5V がかかる(ブザーON)
- ブザーに 5V を加えると音が鳴る(回路内蔵型ブザー)
- 音は約 2,300Hz 固定

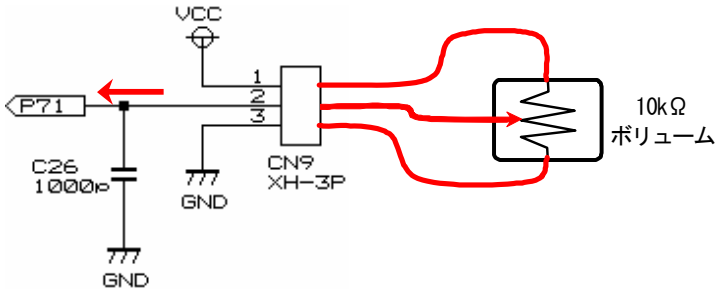


ちなみに、トレーニングボードに実装されているブザーは「圧電ブザー」といい、電圧を加えるだけでは音は鳴りません。圧電ブザーは、鳴らしたい音の周波数のパルスを加えます。例えば、「ド」の音を鳴らしたい場合は約 260Hz のパルスを加えます。

3.7 ボリューム信号入力回路

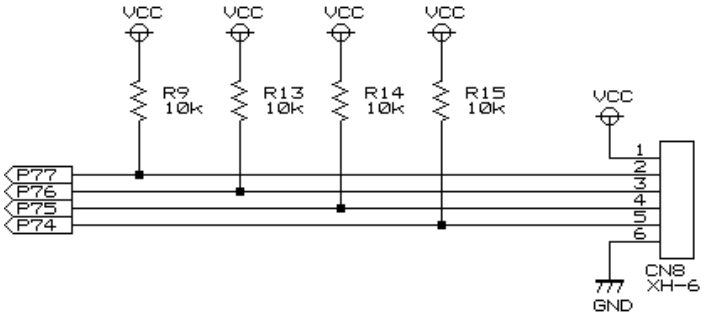
モータドライブ基板 TypeS には、ステアリング角度検出用のボリュームの入力コネクタが実装されています。本基板の回路、サンプルプログラムの特徴は、下記の内容です。

- 3 ピン(抵抗の両端と可変部分がある)のボリュームを取り付け可能
- 1000pF のセラミックコンデンサをノイズ低減用に実装
- 可変部分の電圧 0～5V を、H8 マイコンで A/D 変換して 0～1023(2^{10}-1)に変換



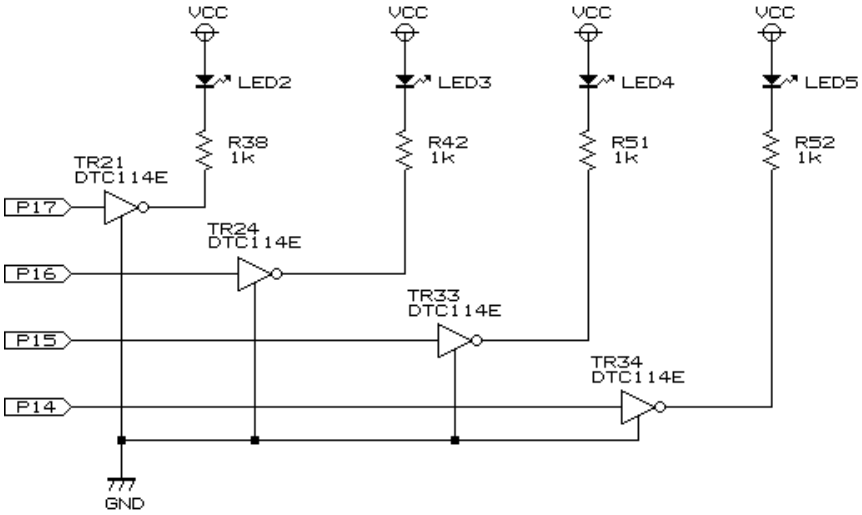
3.8 信号入力回路

モータドライブ基板 TypeS には、リミットスイッチなどの 4 個分の信号を入力するコネクタが実装されています。基板側でプルアップ済みです。



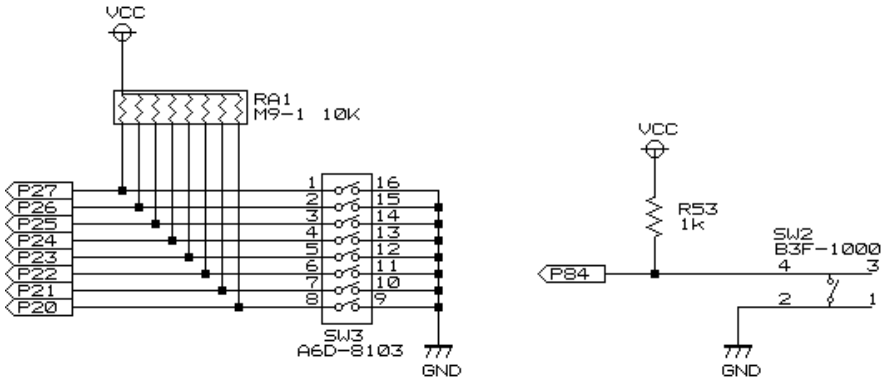
3.9 LED回路

モータドライブ基板 TypeS には、プログラムで点灯／消灯を制御することのできる LED が 4 個あります。



3.10 ディップスイッチ、プッシュスイッチ回路

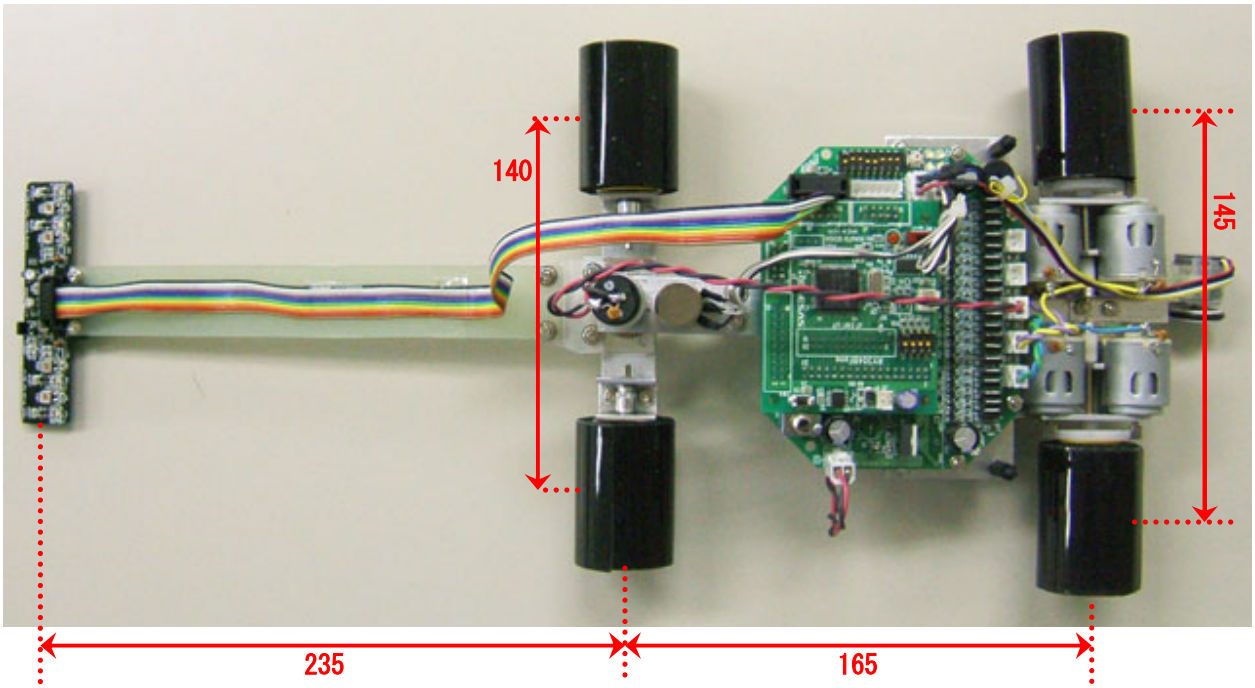
モータドライブ基板 TypeS には、8bit ディップスイッチが 1 個、プッシュスイッチが 1 個あります。



4. 説明用マイコンカーの仕様

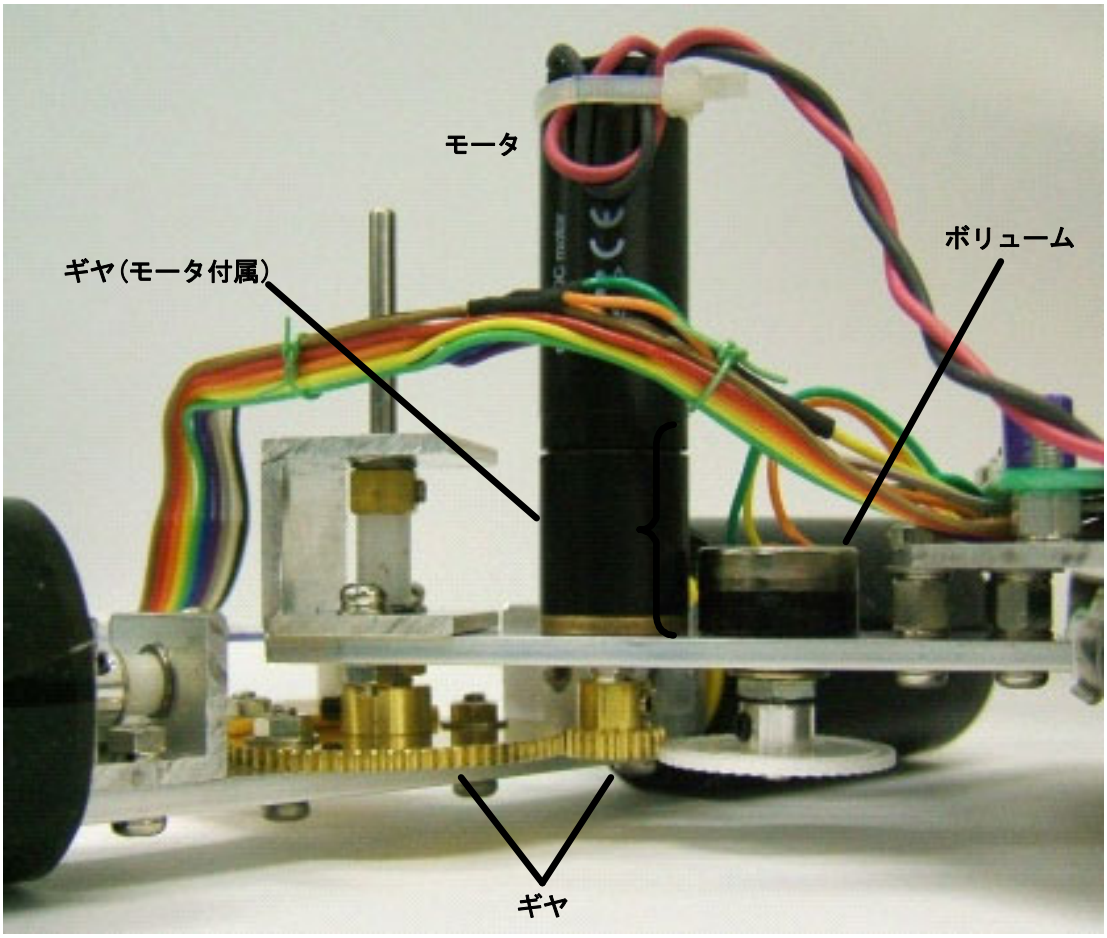
4.1 寸法

本マニュアルで説明しているマイコンカーは、下記のような寸法です。



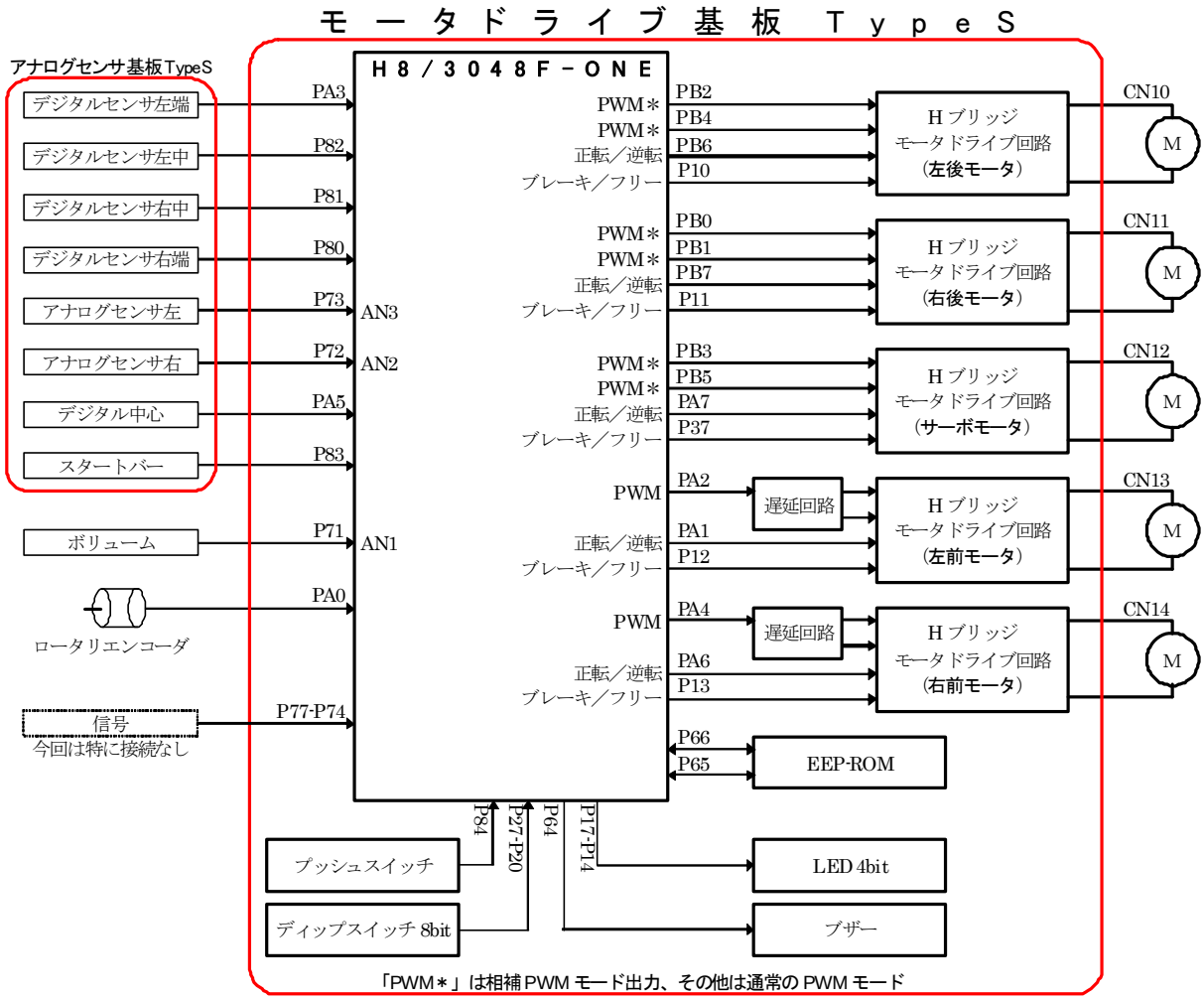
4.2 サーボ機構の自作

ラジコンサーボ＝モータドライブ回路＋モータ＋ギヤ＋ボリューム(＋制御プログラム)
となります。それらをマイコンカーに組み込めば自作サーボの完成です。例として下写真に説明用マイコンカーのサーボ機構の製作例を示します。



項目	詳細
モータ	説明用マイコンカーは、マクソンモータ 118682 (RE16 3.2W)を使用しています。ギヤは 110322 (GP16A 19:1)を使用しています。 最初は高性能モータで実験し、調整のコツをつかんでから安価なモータを使用すると良いでしょう。 ちなみに、指定モータ RC-260RA18130 を 2 個並列に接続すれば、市販されているラジコンサーボにも負けない性能にすることもできます。
ギヤ	ギヤ比はモータのトルクによりますが、今回のモータの場合、40～80 くらいが良いでしょう。写真の例では、 モータ付属のギヤ部分 1／19×自作部分 20／80＝1:76.0 です。 ギヤの組み合わせが多すぎるとギヤの遊び(バックラッシュ)が大きくなり微妙な制御ができません。ステアリング用のギヤは、遊びを極力少なくしてください。
ボリューム	ハンドルの切れ角を検出するためのボリュームです。ハンドルをまっすぐにしたときをボリュームの中心(約 2.5V)に合わせます。左右に目一杯切ったときにボリュームの電圧が 0.5V～4.5V くらいになると良好です。それ以下の電圧でも検出範囲が狭くなるだけで問題はありません。

4.3 ブロック図



4.4 H8/3048F-ONEで使用する内蔵周辺機能

機能	詳細
A/D 変換器	P73～P70 をアナログ電圧入力用として使用します。P73～P70 端子の電圧を A/D 変換器でデジタル値に変換します。 P73…左アナログセンサ電圧入力 P72…右アナログセンサ電圧入力 P71…ボリューム電圧入力 P70…未接続(プルアップ抵抗を接続)
ITU0	PWM モードとして使用しています。左前モータを制御します。
ITU1	PWM モードとして使用しています。右前モータを制御します。
ITU2	ロータリエンコーダパルス入力として使用します。
ITU3,4	ITU3 と 4 を組み合わせて相補 PWM モードとして使用します。また、1周期ごとに割り込みを発生させています。1周期は 1ms なので、1ms ごとに割り込みが発生することになります。下記の 4 つの役割をしています。 1. 左後モータ 2. 右後モータ 3. サーボモータ 4. 1ms ごとの割り込み

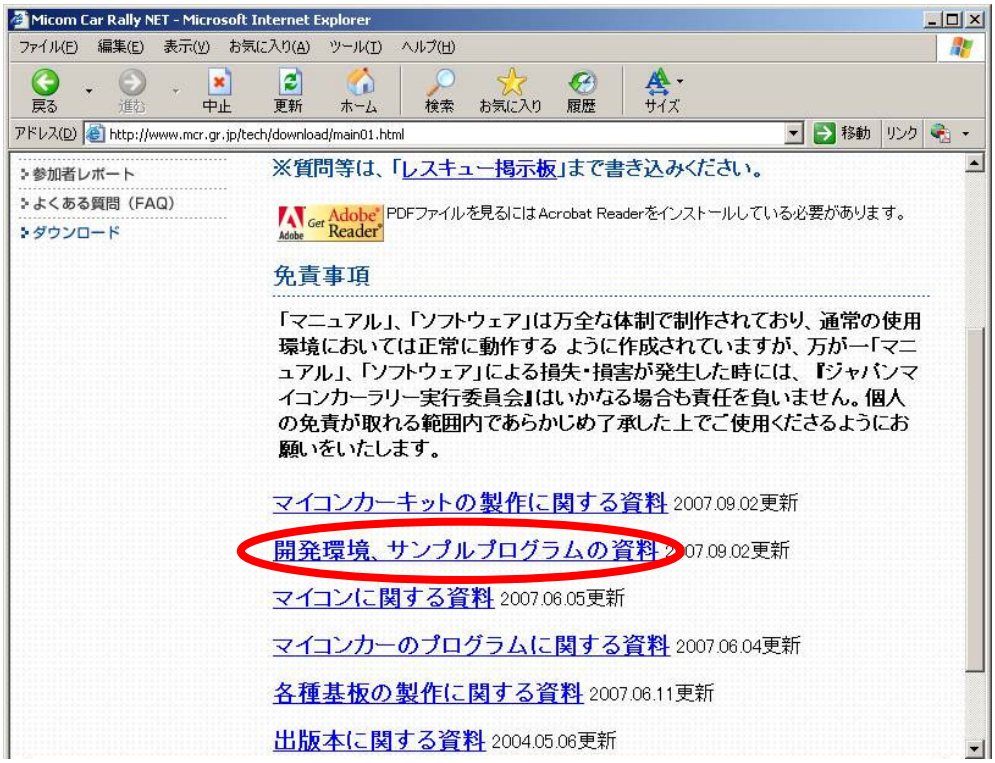
5. ワークスペース「anaservo2」

5.1 インストール

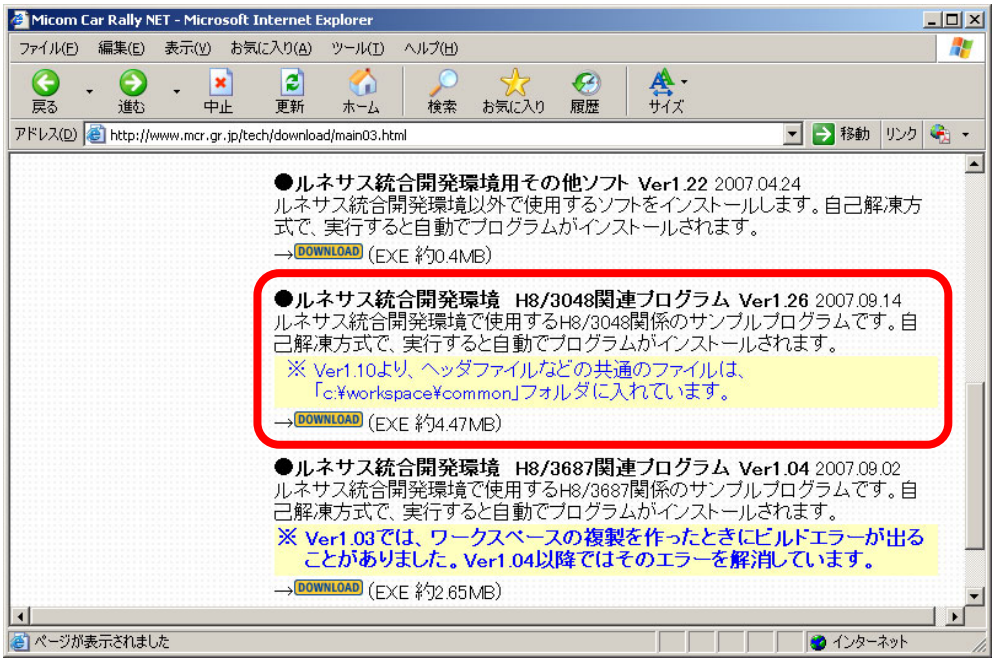
1.マイコンカーラリーホームページ

http://www.mcr.gr.jp

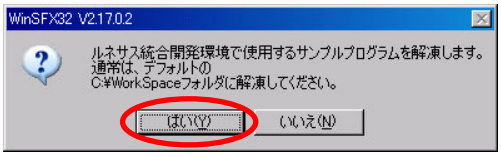
を開き、「技術情報→ダウンロード」をクリックします。



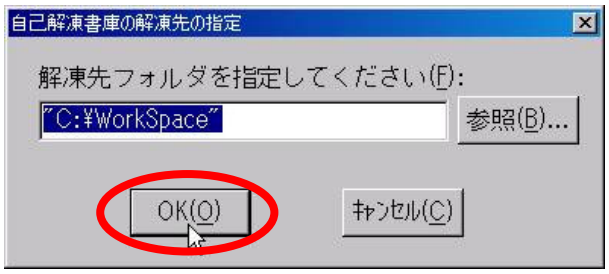
2.「開発環境、サンプルプログラムの資料」をクリックします。



3.「ルネサス統合開発環境 H8/3048 関連プログラム Ver1.26」(Ver の数字はバージョンにより異なります)をクリックして、ファイルをダウンロードします。



4.ダウンロードした「Workspace126.exe」を実行します。はいをクリックします。

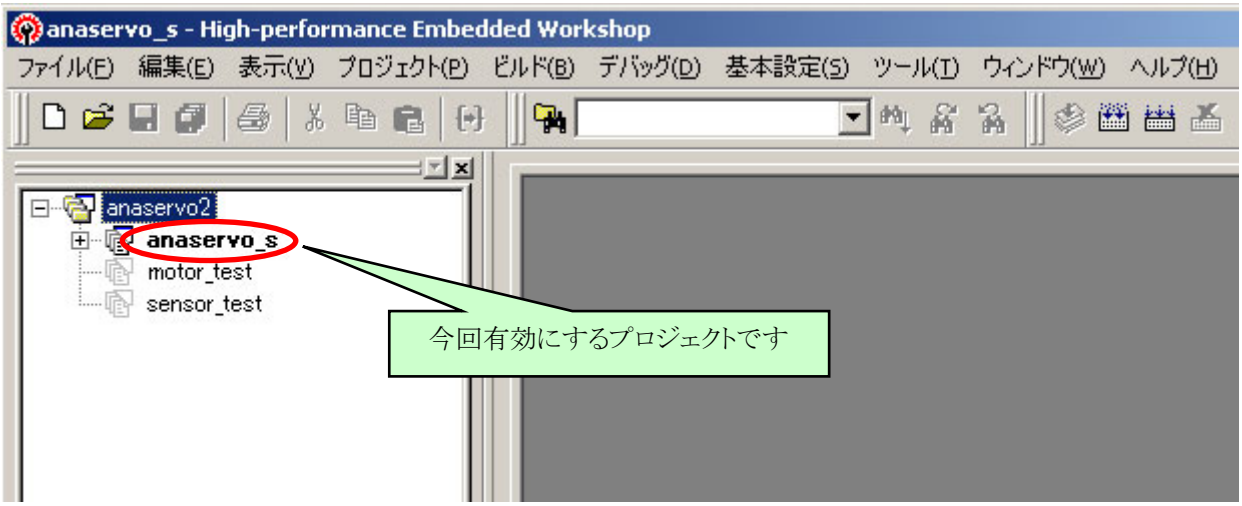


5.ファイルの解凍先を選択します。このフォルダは変更できません。OK をクリックします。

6. 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。複数のフォルダがあります。今回使用するのは、「anaservo2」です。

7.「anaservo2」フォルダを開くと、「anaservo2.hws」ファイルがあります。このファイルがルネサス統合開発環境で開くファイルです。ダブルクリックして開きます。

5.2 プロジェクト

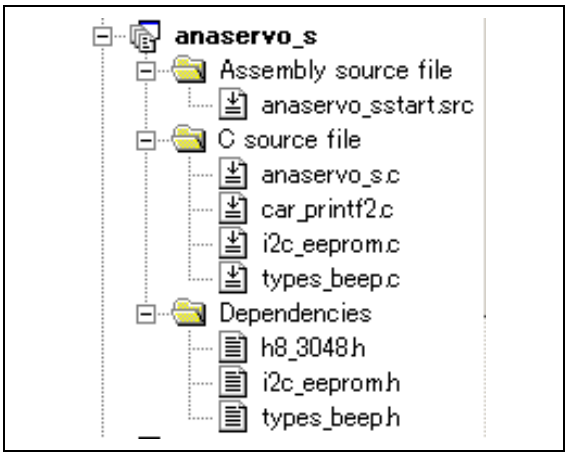


ワークスペース「anaservo2」には、3 つのプロジェクトが登録されています。

プロジェクト名	内容
anaservo_s	アナログセンサ基板 TypeS とモータドライブ基板 TypeS を使ったマイコンカーの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。**コースを完走させるには、プログラムの改造が必要です**。
motor_test	モータドライブ基板 TypeS の動作テスト用プログラムです。
sensor_test	アナログセンサ基板 TypeS の動作テスト用プログラムです。

「anaservo_s」プロジェクトが有効(太字)であることを確認してください。太字になっていない場合、「anaservo_s」で右クリックし「アクティブプロジェクトに設定」し、「anaservo_s」プロジェクトを有効(操作対象)にしてください。

5.3 プロジェクトの構成



	ファイル名	内容
1	anaservo_sstart.src	アセンブリ言語で記述されたアセンブリソースファイルです。このファイルの構造は下記のようになっています。 ベクタアドレス ＋ スタートアップルーチン
2	anaservo_s.c	main 関数がある、マイコンカーを制御するプログラムが書かれています。
3	car_printf2.c	初期値のないグローバル変数(セクション B 領域)、初期値のあるグローバル変数(セクション R 領域)の初期化用、及び printf 関数、scanf 関数を使用するために追加しています。
4	i2c_eeprom.c	モータドライブ基板 TypeS にある EEP-ROM(24C256)を制御するプログラムです。
5	types_beep.c	モータドライブ基板 TypeS にあるブザーを制御するプログラムです。モータドライブ基板 TypeS に実装されているブザーは、電圧を加えると自動的に音が鳴ります。音の高低(周波数)は選べません。 ちなみに、トレーニングボードのブザーはパルスを送り、その周波数の音を鳴らすことができます。制御方式が違います。 ファイル名は、モータドライブ基板 TypeS のブザー制御という意味になっています。
6	h8_3048.h	H8/3048F-ONE マイコンの内蔵周辺機能の I/O レジスタを定義したファイルです。
7	i2c_eeprom.h	i2c_eeprom.c のヘッダファイルです。
8	types_beep.h	types_beep.c のヘッダファイルです。

6. マイコンカー走行プログラムの解説

6.1 プログラムリスト「anasvo2.c」

```
/************************************************************************/
/* TypeS基板使用マイコンカー　サンプルプログラム Ver1.12                */
/*                          2008.03 ジャパンマイコンカーラリー実行委員会  */
/************************************************************************/
/*
Ver1.00 2008.03.29 作成
Ver1.01 2008.06.12 一部修正
Ver1.10 2009.02.01 相補PWMモード部分、大幅に修正
Ver1.11 2009.04.28 speed_r関数の一部修正
Ver1.12 2010.09.02 servoPwmOut関数のsaveData[5]→saveData[6]に変更
*/

/*====================================*/
/* インクルード                       */
/*====================================*/
#include    <no_float.h>                /* stdioの簡略化 最初に置く     */
#include    <stdio.h>                   /* printfなど用                 */
#include    <machine.h>                 /* 組み込み関数用               */
#include    "h8_3048.h"                 /* H8/3048F-ONEレジスタ定義     */
#include    "i2c_eeprom.h"              /* EEP-ROM追加(データ記録)      */
#include    "types_beep.h"              /* TypeS基板のブザー            */

/*====================================*/
/* シンボル定義                       */
/*====================================*/
/* 定数設定 */
#define     MOTOR_CYCLE         3072    /* モータPWMのサイクル兼、      */
                                        /*          タイマ値 1ms        */
#define     MOTOR_OFFSET        37      /* パルスをずらす時間           */
                                        /* 1=325.52[ns]                 */

#define     FREE                1       /* モータモード　フリー         */
#define     BRAKE               0       /* モータモード　ブレーキ       */

/*====================================*/
/* プロトタイプ宣言                   */
/*====================================*/
void init( void );
unsigned char sensor_inp( void );
unsigned char center_inp( void );
unsigned char startbar_get( void );
unsigned char dipsw_get( void );
unsigned char dipsw_get2( void );
unsigned char pushsw_get( void );
unsigned char cn8_get( void );
void led_out( unsigned char led );
void speed_r( int accele_l, int accele_r );
void speed2_r( int accele_l, int accele_r );
void speed_f( int accele_l, int accele_r );
void speed2_f( int accele_l, int accele_r );
void motor_mode_r( int mode_l, int mode_r );
void motor_mode_f( int mode_l, int mode_r );
void servoPwmOut( int pwm );
void motor_mode_s( int mode );
void beep_out( int flag );

int check_crossline( void );
int getServoAngle( void );
int getAnalogSensor( void );
void servoControl( void );
int diff( int pwm );

/*====================================*/
/* グローバル変数の宣言               */
/*====================================*/
int             pattern;                /* マイコンカー動作パターン     */
int             crank_mode;             /* 1:クランクモード 0:通常      */
unsigned long   cnt1;                   /* タイマ用                     */

/* エンコーダ関連 */
int             iTimer10;               /* 10msカウント用               */
long            lEncoderTotal;          /* 積算値保存用                 */
int             iEncoderMax;            /* 10ms毎の値の最大値保存用     */
int             iEncoder;               /* 10ms毎の最新値               */
unsigned int    uEncoderBuff;           /* 計算用　割り込み内で使用     */

/*  サーボ関連 */
int             iSensorBefore;          /* 前回のセンサ値保存           */
int             iServoPwm;              /* サーボＰＷＭ値               */
int             iAngle0;                /* 中心時のA/D値保存            */
```

```
 81 :
 82 :  /* センサ関連 */
 83 :  int             iSensorPattern;         /* センサ状態保持用             */
 84 :
 85 :  /* データ保存関連 */
 86 :  int             saveIndex;              /* 保存インデックス             */
 87 :  int             saveSendIndex;          /* 送信インデックス             */
 88 :  int             saveFlag;               /* 保存フラグ                   */
 89 :  char            saveData[16];           /* 一時保存エリア               */
 90 :  /*
 91 :  保存内容
 92 :  0:pattern       1:Sensor        2:現在の角度1   3:現在の角度2
 93 :  4:アナログ値1   5:アナログ値2   6:サーボのPWM   7:motor_f_l
 94 :  8:motor_f_r     9:motor_r_l     10:motor_r_r    11:iEncoder
 95 :  12:             13:             14:             15:
 96 :  */
 97 :
 98 :  /* 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 */
 99 :  const revolution_difference[] = {       /* 角度から内輪、外輪回転差計算 */
100 :      100, 98, 97, 95, 94,
101 :      92, 91, 89, 88, 87,
102 :      85, 84, 82, 81, 80,
103 :      78, 77, 76, 74, 73,
104 :      72, 70, 69, 68, 66,
105 :      65, 64, 62, 61, 60,
106 :      58, 57, 56, 54, 53,
107 :      52, 50, 49, 48, 46,
108 :      45, 43, 42, 40, 39,
109 :      38 };
110 :
111 :  /************************************************************************/
112 :  /* メインプログラム                                                     */
113 :  /************************************************************************/
114 :  void main( void )
115 :  {
116 :      int             i, j;
117 :      unsigned int    u;
118 :      char            c;
119 :
120 :      /* マイコン機能の初期化 */
121 :      init();                                 /* 初期化                       */
122 :      initI2CEeprom( &P6DDR, &P6DR, 0x90, 6, 5);  /* EEP-ROM初期設定      */
123 :      initBeepS();                            /* ブザー関連処理               */
124 :      init_sci1( 0x00, 79 );                  /* SCI1初期化                   */
125 :      set_ccr( 0x00 );                        /* 全体割り込み許可             */
126 :
127 :      /* マイコンカーの状態初期化 */
128 :      motor_mode_f( BRAKE, BRAKE );
129 :      motor_mode_r( BRAKE, BRAKE );
130 :      motor_mode_s( BRAKE );
131 :      speed_f( 0, 0 );
132 :      speed_r( 0, 0 );
133 :      servoPwmOut( 0 );
134 :      setBeepPatternS( 0x8000 );
135 :
136 :      /* スタート時、スイッチが押されていればデータ転送モード */
137 :      if( pushsw_get() ) {
138 :          pattern = 101;
139 :          cnt1 = 0;
140 :      }
141 :
142 :      while( 1 ) {
143 :
144 :      I2CEepromProcess();                     /* I2C EEP-ROM保存処理          */
145 :
146 :      switch( pattern ) {
147 :      case 0:
148 :          /* プッシュスイッチ押下待ち */
149 :          servoPwmOut( 0 );
150 :          if( pushsw_get() ) {
151 :              setBeepPatternS( 0x8000 );
152 :              clearI2CEeprom();           /* 数秒かかる                   */
153 :              setBeepPatternS( 0xcc00 );
154 :              cnt1 = 0;
155 :              pattern = 1;
156 :              break;
157 :          }
158 :          i =  (cnt1/200) % 2 + 1;
159 :          if( startbar_get() ) {
160 :              i += ((cnt1/100 ) % 2 + 1) << 2;
161 :          }
162 :          led_out( i );                   /* LED点滅処理                  */
163 :          break;
164 :
165 :      case 1:
166 :          /* スタートバー開待ち */
167 :          servoPwmOut( iServoPwm / 2 );
168 :          if( !startbar_get() ) {
169 :              iAngle0 = getServoAngle();  /* 0度の位置記憶                */
170 :              led_out( 0x0 );
171 :              cnt1 = 0;
```

```
172 :              pattern = 11;
173 :              saveFlag = 1;               /* データ保存開始               */
174 :              break;
175 :          }
176 :          led_out( 1 << (cnt1/50) % 4 );
177 :          break;
178 :
179 :      case 11:
180 :          /* 通常トレース */
181 :          servoPwmOut( iServoPwm );
182 :          i = getServoAngle();
183 :          if( i > 170 ) {
184 :              speed_f( 0, 0 );
185 :              speed_r( 0, 0 );
186 :          } else if( i > 25 ) {
187 :              speed_f( diff(80), 80 );
188 :              speed_r( diff(80), 80 );
189 :          } else if( i < -170 ) {
190 :              speed_f( 0, 0 );
191 :              speed_r( 0, 0 );
192 :          } else if( i < -25 ) {
193 :              speed_f( 80, diff(80) );
194 :              speed_r( 80, diff(80) );
195 :          } else {
196 :              speed_f( 100, 100 );
197 :              speed_r( 100, 100 );
198 :          }
199 :          if( check_crossline() ) {       /* クロスラインチェック         */
200 :              cnt1 = 0;
201 :              crank_mode = 1;
202 :              pattern = 21;
203 :          }
204 :          break;
205 :
206 :      case 21:
207 :          /* クロスライン通過処理 */
208 :          servoPwmOut( iServoPwm );
209 :          led_out( 0x3 );
210 :          speed_f( 0, 0 );
211 :          speed_r( 0, 0 );
212 :          if( cnt1 >= 100 ) {
213 :              cnt1 = 0;
214 :              pattern = 22;
215 :          }
216 :          break;
217 :
218 :      case 22:
219 :          /* クロスライン後のトレース、直角検出処理 */
220 :          servoPwmOut( iServoPwm );
221 :          if( iEncoder >= 11 ) {          /* エンコーダによりスピード制御 */
222 :              speed_f( 0, 0 );
223 :              speed_r( 0, 0 );
224 :          } else {
225 :              speed2_f( 70, 70 );
226 :              speed2_r( 70, 70 );
227 :          }
228 :
229 :          if( (sensor_inp()&0x01) == 0x01 ) { /* 右クランク？             */
230 :              led_out( 0x1 );
231 :              cnt1 = 0;
232 :              pattern = 31;
233 :              break;
234 :          }
235 :          if( (sensor_inp()&0x08) == 0x08 ) {  /* 左クランク？             */
236 :              led_out( 0x2 );
237 :              cnt1 = 0;
238 :              pattern = 41;
239 :              break;
240 :          }
241 :          break;
242 :
243 :      case 31:
244 :          /* 右クランク処理 */
245 :          servoPwmOut( 50 );          /* 振りが弱いときは大きくする       */
246 :          speed_f( 60, 33 );          /* この部分は「角度計算(4WD時).xls」*/
247 :          speed_r( 49, 22 );          /* で計算                           */
248 :          if( sensor_inp() == 0x04 ) {    /* 曲げ終わりチェック           */
249 :              cnt1 = 0;
250 :              iSensorPattern = 0;
251 :              crank_mode = 0;
252 :              pattern = 32;
253 :          }
254 :          break;
255 :
256 :      case 32:
257 :          /* 少し時間が経つまで待つ */
258 :          servoPwmOut( iServoPwm );
259 :          speed2_r( 80, 80 );
260 :          speed2_f( 80, 80 );
261 :          if( cnt1 >= 100 ) {
262 :              led_out( 0x0 );
```

```
263 :              pattern = 11;
264 :          }
265 :          break;
266 :
267 :      case 41:
268 :          /* 左クランク処理 */
269 :          servoPwmOut( -50 );          /* 振りが弱いときは大きくする       */
270 :          speed_f( 33, 60 );           /* この部分は「角度計算(4WD時).xls」*/
271 :          speed_r( 22, 49 );           /* で計算                           */
272 :          if( sensor_inp() == 0x02 ) {    /* 曲げ終わりチェック            */
273 :              cnt1 = 0;
274 :              iSensorPattern = 0;
275 :              crank_mode = 0;
276 :              pattern = 42;
277 :          }
278 :          break;
279 :
280 :      case 42:
281 :          /* 少し時間が経つまで待つ */
282 :          servoPwmOut( iServoPwm );
283 :          speed2_f( 80, 80 );
284 :          speed2_r( 80, 80 );
285 :          if( cnt1 >= 100 ) {
286 :              led_out( 0x0 );
287 :              pattern = 11;
288 :          }
289 :          break;
290 :
291 :      case 101:
292 :          /* 停止 */
293 :          servoPwmOut( 0 );
294 :          speed_f( 0, 0 );
295 :          speed_r( 0, 0 );
296 :          setBeepPatternS( 0xc000 );
297 :          saveFlag = 0;
298 :          saveSendIndex = 0;
299 :          pattern = 102;
300 :          cnt1 = 0;
301 :          break;
302 :
303 :      case 102:
304 :          /* プッシュスイッチが離されたかチェック */
305 :          if( !pushsw_get() ) {
306 :              pattern = 103;
307 :              cnt1 = 0;
308 :          }
309 :          break;
310 :
311 :      case 103:
312 :          /* 0.5s待ち */
313 :          if( cnt1 >= 500 ) {
314 :              pattern = 104;
315 :              cnt1 = 0;
316 :          }
317 :          break;
318 :
319 :      case 104:
320 :          /* プッシュスイッチが押されたかチェック */
321 :          led_out( cnt1 / 200 % 2 ? 0x6 : 0x9  );
322 :          if( pushsw_get() ) {
323 :              pattern = 105;
324 :              cnt1 = 0;
325 :          }
326 :          break;
327 :
328 :      case 105:
329 :          /* タイトル転送、転送準備 */
330 :          printf( "¥n" );
331 :          printf( "CarName Data Out¥n" ); /* 自分のカーネームを入れてください */
332 :          printf( "Pattern, Sensor, 角度, アナログ値, サーボPWM, " );
333 :          printf( "左前PWM, 右前PWM, 左後PWM, 右後PWM, エンコーダ¥n");
334 :          pattern = 106;
335 :          break;
336 :
337 :      case 106:
338 :          /* データ転送 */
339 :          led_out( 1 << (cnt1/100) % 4 );
340 :
341 :          /* 終わりのチェック */
342 :          if( (readI2CEeprom( saveSendIndex )==0) ||
343 :                                  (saveSendIndex >= 0x8000) ) {
344 :              pattern = 107;
345 :              setBeepPatternS( 0xff00 );
346 :              cnt1 = 0;
347 :              break;
348 :          }
349 :
350 :          /* データの転送 */
351 :          printf( "%d,0x%02x,%d,%d,%d,%d,%d,%d,%d,%d,%d¥n",
352 :              /* パターン */
353 :              (int)readI2CEeprom( saveSendIndex+0 ),
```

```
354 :              /* センサ */
355 :              (unsigned char)readI2CEeprom( saveSendIndex+1 ),
356 :              /* 角度 */
357 :              (int)((unsigned char)readI2CEeprom(saveSendIndex+2)*0x100 +
358 :                              (unsigned char)readI2CEeprom(saveSendIndex+3) ),
359 :              /* アナログセンサ値 */
360 :              (int)((unsigned char)readI2CEeprom(saveSendIndex+4)*0x100 +
361 :                              (unsigned char)readI2CEeprom(saveSendIndex+5) ),
362 :              /* サーボPWM */
363 :              readI2CEeprom( saveSendIndex+6 ),
364 :              /* 左前PWM */
365 :              readI2CEeprom( saveSendIndex+7 ),
366 :              /* 右前PWM */
367 :              readI2CEeprom( saveSendIndex+8 ),
368 :              /* 左後PWM */
369 :              readI2CEeprom( saveSendIndex+9 ),
370 :              /* 右後PWM */
371 :              readI2CEeprom( saveSendIndex+10 ),
372 :              /* エンコーダ */
373 :              readI2CEeprom( saveSendIndex+11 )
374 :          );
375 :
376 :          saveSendIndex += 16;            /* 次の送信準備                 */
377 :          break;
378 :
379 :      case 107:
380 :          /* 転送終了 */
381 :          led_out( 0xf );
382 :          break;
383 :
384 :          default:
385 :              break;
386 :          }
387 :      }
388 :  }
389 :
390 :  /************************************************************************/
391 :  /* H8/3048F-ONE 内蔵周辺機能の初期化                                    */
392 :  /************************************************************************/
393 :  void init( void )
394 :  {
395 :      /* ポートの入出力設定 */
396 :      P1DDR = 0xff;
397 :      P2DDR = 0x00;
398 :      P3DDR = 0xff;
399 :      P4DDR = 0xff;
400 :      P5DDR = 0xff;
401 :      P6DDR = 0x90;                       /* CPU基板上のDIP SW            */
402 :      P8DDR = 0xe0;
403 :      P9DDR = 0xf7;
404 :      PADDR = 0xd6;
405 :      PBDDR = 0xff;
406 :
407 :      /* A/Dの初期設定 */
408 :      AD_CSR = 0x1b;                      /* スキャンモード使用AN0-AN3    */
409 :      AD_CSR |= 0x20;                     /* ADスタート                   */
410 :
411 :      /* ITU0 左前モータ用PWM */
412 :      ITU0_TCR = 0x23;                    /* カウンタ、クリアの設定       */
413 :      ITU0_GRA = MOTOR_CYCLE;             /* PWM周期                      */
414 :      ITU0_GRB = 0;                       /* デューティ比設定             */
415 :
416 :      /* ITU1 右前モータ用PWM */
417 :      ITU1_TCR = 0x23;                    /* カウンタ、クリアの設定       */
418 :      ITU1_GRA = MOTOR_CYCLE;             /* PWM周期                      */
419 :      ITU1_GRB = 0;                       /* デューティ比設定             */
420 :
421 :      /* ITU2 エンコーダ */
422 :      ITU2_TCR = 0x14;                    /* PA0パルス入力端子            */
423 :
424 :      /* ITU3,4 右後、左後、サーボモータ用PWM兼、割り込み */
425 :      ITU3_TCR = 0x03;
426 :      ITU4_TCR = 0x03;
427 :      ITU_FCR  = 0x2e;                    /* ITU3,4で相補PWMモード        */
428 :      ITU_TOCR = 0x12;                    /* TIOCB3,TOCXA4,TOCXB4は反転   */
429 :      ITU3_IER = 0x01;                    /* TCNT = GRAによる割込み制御   */
430 :      ITU3_CNT = MOTOR_OFFSET;
431 :      ITU4_CNT = 0;
432 :      ITU3_GRA = MOTOR_CYCLE / 2 + MOTOR_OFFSET - 2;  /* PWM周期設定      */
433 :      ITU3_BRB = ITU3_GRB = MOTOR_CYCLE / 2 - 2;
434 :      ITU4_BRA = ITU4_GRA = MOTOR_CYCLE / 2 - 2;
435 :      ITU4_BRB = ITU4_GRB = MOTOR_CYCLE / 2 - 2;
436 :
437 :      ITU_MDR = 0x03;                     /* PWMモード設定                */
438 :      ITU_STR = 0x1f;                     /* ITUのカウントスタート        */
439 :  }
440 :
441 :  /************************************************************************/
442 :  /* ITU3 割り込み処理                                                    */
443 :  /************************************************************************/
444 :  #pragma interrupt( interrupt_timer3 )
```

```
void interrupt_timer3( void )
{
    unsigned int    i;

    ITU3_TSR &= 0xfe;                   /* フラグクリア                 */
    cnt1++;

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

    /* ブザー処理 */
    beepProcessS();

    /* エンコーダ制御 */
    iTimer10++;
    if( iTimer10 >= 10 ) {
        iTimer10 = 0;
        i = ITU2_CNT;
        iEncoder       = i - uEncoderBuff;
        lEncoderTotal += iEncoder;
        if( iEncoder > iEncoderMax )
                    iEncoderMax = iEncoder;
        uEncoderBuff = i;

        /* データ保存関連 */
        if( saveFlag ) {
            saveData[0] = pattern;      /* パターン                     */
            saveData[1] = (center_inp()<<4) + sensor_inp(); /* センサ   */
            i = getServoAngle();        /* 角度                         */
            saveData[2] = i >> 8;
            saveData[3] = i & 0xff;
            i = getAnalogSensor();      /* アナログセンサ値             */
            saveData[4] = i >> 8;
            saveData[5] = i & 0xff;
            /*  6はハンドル関数内でサーボPWM保存        */
            /*  7はモータ関数内で左前モータPWM値保存    */
            /*  8はモータ関数内で右前モータPWM値保存    */
            /*  9はモータ関数内で左後モータPWM値保存    */
            /* 10はモータ関数内で右後モータPWM値保存    */
            saveData[11] = iEncoder;    /* エンコーダ                   */
            saveData[12] = 0;
            saveData[13] = 0;
            saveData[14] = 0;
            saveData[15] = 0;
            setPageWriteI2CEeprom( saveIndex, 16, saveData );
            saveIndex += 16;
            if( saveIndex >= 0x8000 ) saveFlag = 0;
        }
    }
}

/*************************************************************************/
/* アナログセンサ基板TypeSのデジタルセンサ値読み込み                    */
/* 引数   なし                                                           */
/* 戻り値 左端、左中、右中、右端のデジタルセンサ 0:黒 1:白               */
/*************************************************************************/
unsigned char sensor_inp( void )
{
    unsigned char sensor;

    sensor  = (~PADR & 0x08) | (~P8DR & 0x07);

    return sensor;
}

/*************************************************************************/
/* アナログセンサ基板TypeSの中心デジタルセンサ読み込み                  */
/* 引数   なし                                                           */
/* 戻り値 中心デジタルセンサ 0:黒 1:白                                   */
/*************************************************************************/
unsigned char center_inp( void )
{
    unsigned char sensor;

    sensor  = ~PADR & 0x20;             /* アナログセンサ基板TypeSの    */
    sensor  = !!sensor;                 /* 中心デジタルセンサ読み込み   */

    return sensor;
}

/*************************************************************************/
/* アナログセンサ基板TypeSのスタートバー検出センサ読み込み              */
/* 引数   なし                                                           */
/* 戻り値 0:スタートバーなし 1:スタートバーあり                          */
/*************************************************************************/
unsigned char startbar_get( void )
{
    unsigned char sensor;

    sensor  = ~P8DR & 0x08;             /* アナログセンサ基板TypeSの    */
    sensor  = !!sensor;                 /* スタートバーセンサ読み込み   */
```

```
536 :
537 :      return sensor;
538 :  }
539 :
540 :  /************************************************************************/
541 :  /* CPUボード上のディップスイッチ値読み込み                              */
542 :  /* 戻り値 スイッチ値 0～15                                               */
543 :  /************************************************************************/
544 :  unsigned char dipsw_get( void )
545 :  {
546 :      unsigned char sw;
547 :
548 :      sw  = ~P6DR;                        /* ディップスイッチ読み込み     */
549 :      sw &= 0x0f;
550 :
551 :      return  sw;
552 :  }
553 :
554 :  /************************************************************************/
555 :  /* モータドライブ基板TypeS上のディップスイッチ値読み込み                */
556 :  /* 戻り値 スイッチ値 0～15                                               */
557 :  /************************************************************************/
558 :  unsigned char dipsw_get2( void )
559 :  {
560 :      unsigned char sw;
561 :
562 :      sw  = ~P2DR;                        /* ドライブ基板TypeSのSW読み込み*/
563 :
564 :      return  sw;
565 :  }
566 :
567 :  /************************************************************************/
568 :  /* モータドライブ基板TypeS上のプッシュスイッチ値読み込み                */
569 :  /* 戻り値 スイッチ値 0:OFF 1:ON                                          */
570 :  /************************************************************************/
571 :  unsigned char pushsw_get( void )
572 :  {
573 :      unsigned char sw;
574 :
575 :      sw = ~P8DR & 0x10;                  /* プッシュスイッチ読み込み     */
576 :      sw = !!sw;
577 :
578 :      return  sw;
579 :  }
580 :
581 :  /************************************************************************/
582 :  /* モータドライブ基板TypeSのCN8の状態読み込み                           */
583 :  /* 戻り値 0～15                                                         */
584 :  /************************************************************************/
585 :  unsigned char cn8_get( void )
586 :  {
587 :      unsigned char data;
588 :
589 :      data = P7DR >> 4;
590 :
591 :      return  data;
592 :  }
593 :
594 :  /************************************************************************/
595 :  /* モータドライブ基板TypeSのLED制御                                     */
596 :  /* 引数  4個のLED制御 0:OFF 1:ON                                        */
597 :  /************************************************************************/
598 :  void led_out( unsigned char led )
599 :  {
600 :      unsigned char data;
601 :
602 :      led <<= 4;
603 :      data = P1DR & 0x0f;
604 :      P1DR = data | led;
605 :  }
606 :
607 :  /************************************************************************/
608 :  /* 後輪の速度制御                                                       */
609 :  /* 引数   左モータ:-100～100 , 右モータ:-100～100                       */
610 :  /*        0で停止、100で正転100%、-100で逆転100%                        */
611 :  /************************************************************************/
612 :  void speed_r( int accele_l, int accele_r )
613 :  {
614 :      unsigned int    sw_data;
615 :      unsigned int    work;
616 :
617 :      sw_data  = dipsw_get() + 5;         /* ディップスイッチ読み込み     */
618 :      sw_data *= 5;                       /* 5～20 → 25～100に変換        */
619 :
620 :      saveData[ 9] = accele_l * sw_data / 100;    /* ログ保存             */
621 :      saveData[10] = accele_r * sw_data / 100;    /* ログ保存             */
622 :
623 :      /* 左モータ */
624 :      if( accele_l > 0 ) {
625 :          PBDR &= 0xbf;
626 :      } else if( accele_l < 0 ) {
```

```
627 :          PBDR |= 0x40;
628 :          accele_l = -accele_l;
629 :      }
630 :      work   = MOTOR_CYCLE / 2 - MOTOR_OFFSET - 2;
631 :      work   = (long)work * accele_l * sw_data / 10000;
632 :      ITU4_BRA = MOTOR_CYCLE / 2 - 2 - work;
633 :
634 :      /* 右モータ */
635 :      if( accele_r > 0 ) {
636 :          PBDR &= 0x7f;
637 :      } else if( accele_r < 0 ) {
638 :          PBDR |= 0x80;
639 :          accele_r = -accele_r;
640 :      }
641 :      work   = MOTOR_CYCLE / 2 - MOTOR_OFFSET - 2;
642 :      work   = (long)work * accele_r * sw_data / 10000;
643 :      ITU3_BRB = MOTOR_CYCLE / 2 - 2 - work;
644 :  }
645 :
646 :  /*********************************************************************/
647 :  /* 後輪の速度制御2 ディップスイッチには関係しないspeed関数             */
648 :  /* 引数   左モータ:-100～100 , 右モータ:-100～100                     */
649 :  /*        0で停止、100で正転100%、-100で逆転100%                      */
650 :  /*********************************************************************/
651 :  void speed2_r( int accele_l, int accele_r )
652 :  {
653 :      unsigned int    sw_data;
654 :      unsigned int    work;
655 :
656 :      saveData[ 9] = accele_l;            /* ログ保存                  */
657 :      saveData[10] = accele_r;            /* ログ保存                  */
658 :
659 :      /* 左モータ */
660 :      if( accele_l > 0 ) {
661 :          PBDR &= 0xbf;
662 :      } else if( accele_l < 0 ) {
663 :          PBDR |= 0x40;
664 :          accele_l = -accele_l;
665 :      }
666 :      work   = MOTOR_CYCLE / 2 - MOTOR_OFFSET - 2;
667 :      work   = (long)work * accele_l / 100;
668 :      ITU4_BRA = MOTOR_CYCLE / 2 - 2 - work;
669 :
670 :      /* 右モータ */
671 :      if( accele_r > 0 ) {
672 :          PBDR &= 0x7f;
673 :      } else if( accele_r < 0 ) {
674 :          PBDR |= 0x80;
675 :          accele_r = -accele_r;
676 :      }
677 :      work   = MOTOR_CYCLE / 2 - MOTOR_OFFSET - 2;
678 :      work   = (long)work * accele_r / 100;
679 :      ITU3_BRB = MOTOR_CYCLE / 2 - 2 - work;
680 :  }
681 :
682 :  /*********************************************************************/
683 :  /* 前輪の速度制御                                                    */
684 :  /* 引数   左モータ:-100～100 , 右モータ:-100～100                    */
685 :  /*        0で停止、100で正転100%、-100で逆転100%                     */
686 :  /*********************************************************************/
687 :  void speed_f( int accele_l, int accele_r )
688 :  {
689 :      unsigned char   sw_data;
690 :      unsigned long   speed_max;
691 :
692 :      sw_data  = dipsw_get() + 5;         /* ディップスイッチ読み込み    */
693 :      speed_max = (unsigned long)(MOTOR_CYCLE-1) * sw_data / 20;
694 :
695 :      saveData[7] = accele_l * sw_data / 20;  /* ログ保存               */
696 :      saveData[8] = accele_r * sw_data / 20;  /* ログ保存               */
697 :
698 :      /* 左前モータ */
699 :      if( accele_l > 0 ) {
700 :          PADR &= 0xfd;
701 :      } else if( accele_l < 0 ) {
702 :          PADR |= 0x02;
703 :          accele_l = -accele_l;
704 :      }
705 :      /* GRBがCNTより20以上小さい値かどうかのチェック */
706 :      if( ITU0_GRB > 20 ) {
707 :          /* GRBが20以上なら  単純に比較 */
708 :          while( (ITU0_CNT >= ITU0_GRB-20) && (ITU0_CNT <= ITU0_GRB) );
709 :      } else {
710 :          /* GRBが20以下なら  上限値からの値も参照する */
711 :          while( (ITU0_CNT >= ITU0_GRA-20) || (ITU0_CNT <= ITU0_GRB) );
712 :      }
713 :      ITU0_GRB = speed_max * accele_l / 100;
714 :
715 :      /* 右前モータ */
716 :      if( accele_r > 0 ) {
717 :          PADR &= 0xbf;
```

```
718 :      } else if( accele_r < 0 ) {
719 :          PADR |= 0x40;
720 :          accele_r = -accele_r;
721 :      }
722 :      /* GRBがCNTより20以上小さい値かどうかのチェック */
723 :      if( ITU1_GRB > 20 ) {
724 :          /* GRBが20以上なら  単純に比較 */
725 :          while( (ITU1_CNT >= ITU1_GRB-20) && (ITU1_CNT <= ITU1_GRB) );
726 :      } else {
727 :          /* GRBが20以下なら  上限値からの値も参照する */
728 :          while( (ITU1_CNT >= ITU1_GRA-20) || (ITU1_CNT <= ITU1_GRB) );
729 :      }
730 :      ITU1_GRB = speed_max * accele_r / 100;
731 :  }
732 :
733 :  /*********************************************************************/
734 :  /* 前輪の速度制御2 ディップスイッチには関係しないspeed関数             */
735 :  /* 引数   左モータ:-100～100 , 右モータ:-100～100                     */
736 :  /*        0で停止、100で正転100%、-100で逆転100%                      */
737 :  /*********************************************************************/
738 :  void speed2_f( int accele_l, int accele_r )
739 :  {
740 :      unsigned long   speed_max;
741 :
742 :      saveData[7] = accele_l;             /* ログ保存                  */
743 :      saveData[8] = accele_r;             /* ログ保存                  */
744 :
745 :      speed_max = MOTOR_CYCLE - 1;
746 :
747 :      /* 左前モータ */
748 :      if( accele_l > 0 ) {
749 :          PADR &= 0xfd;
750 :      } else if( accele_l < 0 ) {
751 :          PADR |= 0x02;
752 :          accele_l = -accele_l;
753 :      }
754 :      /* GRBがCNTより20以上小さい値かどうかのチェック */
755 :      if( ITU0_GRB > 20 ) {
756 :          /* GRBが20以上なら  単純に比較 */
757 :          while( (ITU0_CNT >= ITU0_GRB-20) && (ITU0_CNT <= ITU0_GRB) );
758 :      } else {
759 :          /* GRBが20以下なら  上限値からの値も参照する */
760 :          while( (ITU0_CNT >= ITU0_GRA-20) || (ITU0_CNT <= ITU0_GRB) );
761 :      }
762 :      ITU0_GRB = speed_max * accele_l / 100;
763 :
764 :      /* 右前モータ */
765 :      if( accele_r > 0 ) {
766 :          PADR &= 0xbf;
767 :      } else if( accele_r < 0 ) {
768 :          PADR |= 0x40;
769 :          accele_r = -accele_r;
770 :      }
771 :      /* GRBがCNTより20以上小さい値かどうかのチェック */
772 :      if( ITU1_GRB > 20 ) {
773 :          /* GRBが20以上なら  単純に比較 */
774 :          while( (ITU1_CNT >= ITU1_GRB-20) && (ITU1_CNT <= ITU1_GRB) );
775 :      } else {
776 :          /* GRBが20以下なら  上限値からの値も参照する */
777 :          while( (ITU1_CNT >= ITU1_GRA-20) || (ITU1_CNT <= ITU1_GRB) );
778 :      }
779 :      ITU1_GRB = speed_max * accele_r / 100;
780 :  }
781 :
782 :  /*********************************************************************/
783 :  /* 後モータ停止動作 (ブレーキ、フリー)                                */
784 :  /* 引数   左モータ:FREE or BRAKE , 右モータ:FREE or BRAKE             */
785 :  /*********************************************************************/
786 :  void motor_mode_r( int mode_l, int mode_r )
787 :  {
788 :      if( mode_l ) {
789 :          P1DR |= 0x01;
790 :      } else {
791 :          P1DR &= 0xfe;
792 :      }
793 :      if( mode_r ) {
794 :          P1DR |= 0x02;
795 :      } else {
796 :          P1DR &= 0xfd;
797 :      }
798 :  }
799 :
800 :  /*********************************************************************/
801 :  /* 前モータ停止動作 (ブレーキ、フリー)                                */
802 :  /* 引数   左モータ:FREE or BRAKE , 右モータ:FREE or BRAKE             */
803 :  /*********************************************************************/
804 :  void motor_mode_f( int mode_l, int mode_r )
805 :  {
806 :      if( mode_l ) {
807 :          P1DR |= 0x04;
808 :      } else {
```

```
809 :          P1DR &= 0xfb;
810 :      }
811 :      if( mode_r ) {
812 :          P1DR |= 0x08;
813 :      } else {
814 :          P1DR &= 0xf7;
815 :      }
816 :  }
817 :
818 :  /*************************************************************************/
819 :  /* サーボモータ制御                                                      */
820 :  /* 引数   サーボモータPWM：-100～100                                     */
821 :  /*************************************************************************/
822 :  void servoPwmOut( int pwm )
823 :  {
824 :      unsigned int    work;
825 :
826 :      saveData[6] = pwm;                  /* ログ保存                   */
827 :
828 :      if( pwm > 0 ) {
829 :          PADR &= 0x7f;
830 :      } else if( pwm < 0 ) {
831 :          PADR |= 0x80;
832 :          pwm = -pwm;
833 :      }
834 :      work   = MOTOR_CYCLE / 2 - MOTOR_OFFSET - 2;
835 :      work   = (long)work * pwm / 100;
836 :      ITU4_BRB = MOTOR_CYCLE / 2 - 2 - work;
837 :  }
838 :
839 :  /*************************************************************************/
840 :  /* サーボモータ停止動作（ブレーキ、フリー）                              */
841 :  /* 引数   FREE or BRAKE                                                  */
842 :  /*************************************************************************/
843 :  void motor_mode_s( int mode )
844 :  {
845 :      if( mode ) {
846 :          P3DR |= 0x80;
847 :      } else {
848 :          P3DR &= 0x7f;
849 :      }
850 :  }
851 :
852 :  /*************************************************************************/
853 :  /* ブザー制御                                                            */
854 :  /* 引数  0:ブザーOFF 1:ブザーON                                          */
855 :  /*************************************************************************/
856 :  void beep_out( int flag )
857 :  {
858 :      if( flag ) {
859 :          P6DR |= 0x10;
860 :      } else {
861 :          P6DR &= 0xef;
862 :      }
863 :  }
864 :
865 :  /*=======================================================================*/
866 :  /* 応用関数                                                              */
867 :  /*=======================================================================*/
868 :
869 :  /*************************************************************************/
870 :  /* クロスライン検出処理                                                  */
871 :  /* 戻り値 0:クロスラインなし 1:あり                                      */
872 :  /*************************************************************************/
873 :  int check_crossline( void )
874 :  {
875 :      unsigned char b;
876 :      int ret = 0;
877 :
878 :      b = sensor_inp();
879 :      if( b==0x0f || b==0x0e || b==0x0d || b==0x0b || b==0x07 ) {
880 :          ret = 1;
881 :      }
882 :      return ret;
883 :  }
884 :
885 :  /*************************************************************************/
886 :  /* サーボ角度取得                                                        */
887 :  /* 引数   なし                                                           */
888 :  /* 戻り値 入れ替え後の値                                                 */
889 :  /*************************************************************************/
890 :  int getServoAngle( void )
891 :  {
892 :      return( (AD_DRB>>6) - iAngle0 );
893 :  }
894 :
895 :  /*************************************************************************/
896 :  /* アナログセンサ値取得                                                  */
897 :  /* 引数   なし                                                           */
898 :  /* 戻り値 センサ値                                                       */
899 :  /*************************************************************************/
```

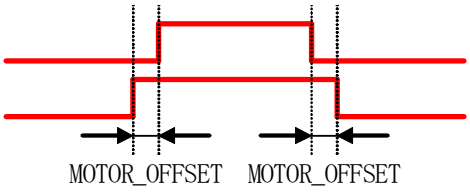
```
int getAnalogSensor( void )
{
    int     ret;

    ret = (AD_DRD>>6) - (AD_DRC>>6);    /* アナログセンサ情報取得       */

    if( !crank_mode ) {
        /* クランクモードでなければ補正処理 */
        switch( iSensorPattern ) {
        case 0:
            if( sensor_inp() == 0x04 ) {
                ret = -650;
                break;
            }
            if( sensor_inp() == 0x02 ) {
                ret = 650;
                break;
            }
            if( sensor_inp() == 0x0c ) {
                ret = -700;
                iSensorPattern = 1;
                break;
            }
            if( sensor_inp() == 0x03 ) {
                ret = 700;
                iSensorPattern = 2;
                break;
            }
            break;

        case 1:
            /* センサ右寄り */
            ret = -700;
            if( sensor_inp() == 0x04 ) {
                iSensorPattern = 0;
            }
            break;

        case 2:
            /* センサ左寄り */
            ret = 700;
            if( sensor_inp() == 0x02 ) {
                iSensorPattern = 0;
            }
            break;
        }
    }

    return ret;
}

/************************************************************************/
/* モジュール名 servoControl                                            */
/* 処理概要     サーボモータ制御                                        */
/* 引数         なし                                                    */
/* 戻り値       グローバル変数 iServoPwm に代入                         */
/************************************************************************/
void servoControl( void )
{
    int     i, iRet, iP, iD;
    int     kp, kd;

    i = getAnalogSensor();              /* センサ値取得                 */
    kp = dipsw_get2() & 0x0f;           /* 調整できたらP,D値は固定値に  */
    kd = (dipsw_get2() >> 4) * 5;       /* してください                 */

    /* サーボモータ用PWM値計算 */
    iP = kp * i;                        /* 比例                         */
    iD = kd * (iSensorBefore - i );     /* 微分(目安はPの5～10倍)       */
    iRet = iP - iD;
    iRet /= 64;

    /* PWMの上限の設定 */
    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90くらいにしてください */
    iServoPwm = iRet;

    iSensorBefore = i;                  /* 次回はこの値が1ms前の値となる*/
}

/************************************************************************/
/* 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     */
/* 引数　 外輪PWM                                                       */
/* 戻り値 内輪PWM                                                       */
/************************************************************************/
int diff( int pwm )
{
    int i, ret;

    i  = getServoAngle() / 5;           /* 1度あたりの増分で割る        */
    if( i <  0 ) i = -i;
```

```
991 :       if( i > 45 ) i = 45;
992 :       ret = revolution_difference[i] * pwm / 100;
993 :
994 :       return ret;
995 :   }
996 :
997 :   /************************************************************************/
998 :   /* end of file                                                          */
999 :   /************************************************************************/
```

6.2 解説

6.2.1 シンボル定義

```
23 :   /*======================================*/
24 :   /* シンボル定義                         */
25 :   /*======================================*/
26 :   /* 定数設定 */
27 :   #define     MOTOR_CYCLE         3072    /* モータPWMのサイクル兼、      */
28 :                                           /*          タイマ値 1ms        */
29 :   #define     MOTOR_OFFSET        37      /* パルスをずらす時間           */
30 :                                           /* 1=325.52[ns]                 */
31 :
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
MOTOR_CYCLE	モータ PWM 周期と割り込み周期の値を決める値です。今回は 1ms とします。 RY3048Fone ボードのクリスタルは、24.576[MHz]ですので、1クロック分の周期は、 $1／(24.576×10^{6})＝40.69[ns]$ となります。ITU は、そのクロックの 8 分周で使うこととします。そのため、1[ms]分の時間を作るのにカウントする回数は、 $(1×10^{-3})÷(40.69×10^{-9})÷8＝3,072$ となります。
MOTOR_OFFSET	 P チャネル FET と N チャネル FET のゲートに加える波形のずらす時間を設定します。FET のターンオン、ターンオフ時間は 1[μs]以下ですが、ゲート駆動用のデジタルトランジスタの時間も含めて考えます。 デジタルトランジスタの 最大ターンオン時間…5.2[μs] 最大ターンオフ時間…11[μs] FET のターンオン、ターンオフを 1[μs]とすると、最大 12[μs]となります。よって、 $(12×10^{-6})／(325.52×10^{-9})≒37$ となります。
FREE BRAKE	motor_mode_r 関数、motor_mode_f 関数、motor_mode_s 関数で使用する定数です。 モータの停止をフリーにしたい場合は「FREE」、ブレーキにしたい場合は「BRAKE」を引数にセットします。 例) motor_mode_r(FREE, BRAKE); 左後モータの停止はフリー、右後モータの停止はブレーキ

6.2.2 変数の定義

```
63 :  /*====================================*/
64 :  /* グローバル変数の宣言                 */
65 :  /*====================================*/
66 :  int             pattern;              /* マイコンカー動作パターン     */
67 :  int             crank_mode;           /* 1:クランクモード 0:通常      */
68 :  unsigned long   cnt1;                 /* タイマ用                     */
69 :
70 :  /* エンコーダ関連 */
71 :  int             iTimer10;             /* 10msカウント用               */
72 :  long            lEncoderTotal;        /* 積算値保存用                 */
73 :  int             iEncoderMax;          /* 10ms毎の値の最大値保存用     */
74 :  int             iEncoder;             /* 10ms毎の最新値               */
75 :  unsigned int    uEncoderBuff;         /* 計算用　割り込み内で使用     */
76 :
77 :  /*  サーボ関連 */
78 :  int             iSensorBefore;        /* 前回のセンサ値保存           */
79 :  int             iServoPwm;            /* サーボＰＷＭ値               */
80 :  int             iAngle0;              /* 中心時のA/D値保存            */
81 :
82 :  /* センサ関連 */
83 :  int             iSensorPattern;       /* センサ状態保持用             */
84 :
85 :  /* データ保存関連 */
86 :  int             saveIndex;            /* 保存インデックス             */
87 :  int             saveSendIndex;        /* 送信インデックス             */
88 :  int             saveFlag;             /* 保存フラグ                   */
89 :  char            saveData[16];         /* 一時保存エリア               */
90 :  /*
91 :  保存内容
92 :  0:pattern       1:Sensor        2:現在の角度1   3:現在の角度2
93 :  4:アナログ値1   5:アナログ値2   6:サーボのPWM   7:motor_f_l
94 :  8:motor_f_r     9:motor_r_l     10:motor_r_r    11:iEncoder
95 :  12:            13:             14:             15:
96 :  */
```

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を設定します。 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割り込み内で行います。割り込みは 1ms ごとですが、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 1 回の割り込みごとに足していき、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になります。

lEncoderTotal	ロータリエンコーダの積算値が保存されています。スタートしてからの距離が分かります。
iEncoderMax	10ms ごとに計測したロータリエンコーダ値の最大値が保存されます。
iEncoder	10ms ごとに計測したロータリエンコーダ値の最新値が保存されます。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	割り込み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 値を取得する関数内で使用します。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。
saveIndex	EEP-ROM へ走行データを保存するときの、保存インデックス(保存アドレス)です。
saveSendIndex	EEP-ROM からデータをパソコンへ送るときの、送信インデックス(送信アドレス)です。
saveFlag	EEP-ROM へ走行データを保存するかしないかのフラグです。 0:保存しない 1:保存する
saveData[16]	走行データを保存するときの、一時保管エリアです。

6.2.3 内輪差値計算用の配列追加

```
 98 :  /* 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 */
 99 :  const revolution_difference[] = {        /* 角度から内輪、外輪回転差計算 */
100 :      100, 98, 97, 95, 94,
101 :      92, 91, 89, 88, 87,
102 :      85, 84, 82, 81, 80,
103 :      78, 77, 76, 74, 73,
104 :      72, 70, 69, 68, 66,
105 :      65, 64, 62, 61, 60,
106 :      58, 57, 56, 54, 53,
107 :      52, 50, 49, 48, 46,
108 :      45, 43, 42, 40, 39,
109 :      38 };
```

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セクション C(ROM)に配置されます。H8 はフラッシュ ROM が 128KB、RAM が 4KB 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 を取ると初期値付き大域変数となり、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と同じ意味になります。[ ]の中に入れる数字を添字といいます。添字は 0 から数えます。

```
revolution_difference[ 0] = 100
revolution_difference[ 1] = 98
revolution_difference[ 2] = 97
          ⋮              ⋮      ⋮
revolution_difference[45] = 38
```

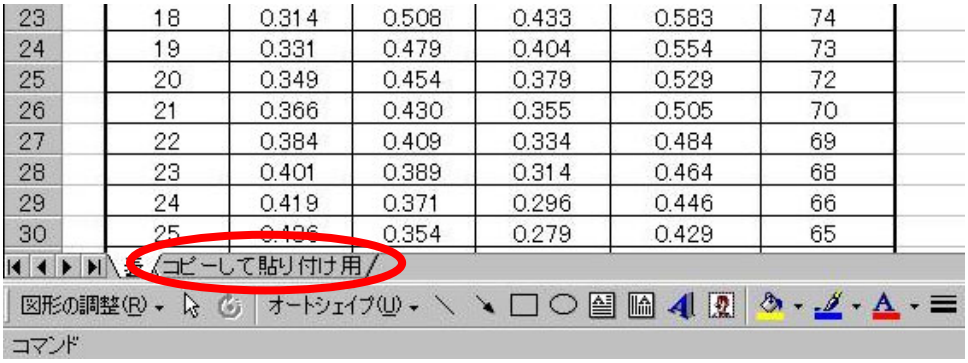
というように、順番に値が返ってきます。ちなみに 46 以上の値を設定してもエラーにはなりませんが、不定な値が返ってきます。46 以上にしないように注意する必要があります。

値の意味は、外輪の回転を100としたとき、添字に現在のハンドル角度を入れると内輪の回転数が返ってくるようにしています。添字が 2 のとき、97 が返ってきます。これは外輪 100、ハンドル角度が 2 度のとき、内輪の回転数は 97 ということです。ハンドル角度が 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、マイコンカーのトレッド、ホイールベース、ハンドル角度を入力すると、内輪の PWM 値が出力されるエクセル表「**角度計算.xls**」を用意しました。そこに、ホイールベース＝0.165、トレッド＝0.15 と入力します。

	A	B	C	D	E	F	G
1		W	0.165	m ←ホイールベースを入力してください			
2		T	0.15	m ←トレッドを入力してください			
3							
4		度	rad	r2	r1	r3	r1/r3*100
5		0	0				100
6		1	0.017	9.458	9.383	9.533	98
7		2	0.035	4.727	4.652	4.802	97
8		3	0.052	3.150	3.075	3.225	95
9		4	0.070	2.361	2.286	2.436	94
10		5	0.087	1.887	1.812	1.962	92
11		6	0.105	1.571	1.496	1.646	91
12		7	0.122	1.345	1.270	1.420	89
13		8	0.140	1.175	1.100	1.250	88
14		9	0.157	1.042	0.967	1.117	87

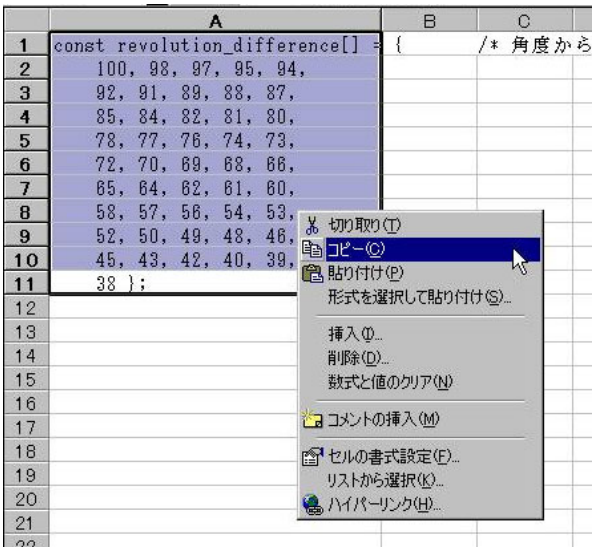
一覧表ができましたので、下の「コピーして貼り付け用」タグを選択、内容をコピーしてプログラムに貼り付けます。



23	18	0.314	0.508	0.433	0.583	74
24	19	0.331	0.479	0.404	0.554	73
25	20	0.349	0.454	0.379	0.529	72
26	21	0.366	0.430	0.355	0.505	70
27	22	0.384	0.409	0.334	0.484	69
28	23	0.401	0.389	0.314	0.464	68
29	24	0.419	0.371	0.296	0.446	66
30	25	0.436	0.354	0.279	0.429	65

コピーして貼り付け用 / 図形の調整(R) オートシェイプ(U) コマンド

A1～A11まで選択し、右クリックで「コピー」を選択します。



プログラムの、「revolution_difference」部分を更新します。これで、自分のマイコンカーにあった内輪差が計算されました。

6.2.4 ポートの入出力設定

```
/************************************************************************/
/* H8/3048F-ONE 内蔵周辺機能の初期化                                    */
/************************************************************************/
void init( void )
{
    /* ポートの入出力設定 */
    P1DDR = 0xff;
    P2DDR = 0x00;
    P3DDR = 0xff;
    P4DDR = 0xff;
    P5DDR = 0xff;
    P6DDR = 0x90;                   /* CPU 基板上の DIP SW          */
    P8DDR = 0xe0;
    P9DDR = 0xf7;
    PADDR = 0xd6;
    PBDDR = 0xff;
```

モータドライブ基板 TypeS、アナログセンサ基板 TypeS の接続機器に合わせてポートの入出力設定を行います。未接続ポートは出力設定にします。下表にポートの接続状態を示します。

ポート	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
1	LED 出力	LED 出力	LED 出力	LED 出力	右前モータ フリー/ブレーキ 出力	左前モータ フリー/ブレーキ 出力	右後モータ フリー/ブレーキ 出力	左後モータ フリー/ブレーキ 出力
2	ディップ SW (TypeS) 入力	ディップ SW (TypeS) 入力	ディップ SW (TypeS) 入力	ディップ SW (TypeS) 入力	ディップ SW (TypeS) 入力	ディップ SW (TypeS) 入力	ディップ SW (TypeS) 入力	ディップ SW (TypeS) 入力
3	サーボモータ フリー/ブレーキ 出力							
4								
5	−	−	−	−				
6	−	EEP-ROM SCL 入力	EEP-ROM SDA 入力	ブザー 出力	ディップ SW (CPU) 入力	ディップ SW (CPU) 入力	ディップ SW (CPU) 入力	ディップ SW (CPU) 入力
7	信号 入力	信号 入力	信号 入力	信号 入力	アナログ センサ左 入力	アナログ センサ右 入力	ボリューム 入力	
8	−	−	−	プッシュ SW 入力	スタートバー センサ 入力	デジタル センサ左中 入力	デジタル センサ右中 入力	デジタル センサ右端 入力
9					通信(RxD) 入力		通信(TxD) 出力	
A	サーボモータ 正転／逆転 出力	右前モータ 正転／逆転 出力	デジタル センサ中心 入力	右前モータ PWM 出力	デジタル センサ左端 入力	左前モータ PWM 出力	左前モータ 正転／逆転 出力	ロータリ エンコーダ 入力
B	右後モータ 正転／逆転 出力	左後モータ 正転／逆転 出力	サーボモータ PWM 出力	左後モータ PWM 出力	サーボモータ PWM 出力	左後モータ PWM 出力	右後モータ PWM 出力	右後モータ PWM 出力
ポート	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0

※未記入部分:未接続(出力に設定、ポート 7 は必ず入力)
※ー:端子なし

6.2.5 A/Dの設定

```
407 :       /* A/Dの初期設定 */
408 :       AD_CSR = 0x1b;                  /* スキャンモード使用AN0-AN3     */
409 :       AD_CSR |= 0x20;                 /* ADスタート                    */
```

P73～P70 の端子をアナログ入力に設定します。AD_CSR を設定します。ちなみにポート 7 の端子をアナログ電圧入力として使うとき、アナログ AN 端子とも呼びます。例えば AN0 端子は、P70 端子と同じです。

■AD_CSR(A/D コントロール／ステータスレジスタ)の設定内容

ビット:	7	6	5	4	3	2	1	0
AD_CSR:	ADF	ADIE	ADST	SCAN	CKS	CH2	CH1	CH0
設定値:	0	0	0	1	1	0	1	1
16 進数:	1				b			

・ビット 7:A/D エンドフラグ(ADF)

A/D 変換の終了を示すステータスフラグです。

ADF	説明
0	〔クリア条件〕 ADF＝1 の状態で、ADF フラグをリードした後、ADF フラグに 0 をライトしたとき
1	〔セット条件〕 (1)単一モード:A/D 変換が終了したとき (2)スキャンモード:設定されたすべてのチャネルの A/D 変換が終了したとき

関係ないので、0 のままにしておきます。

・ビット 6:A/D インタラプトイネーブル(ADIE)

A/D 変換の終了による割り込み(ADI)要求の許可／禁止を選択します。

ADIE	説明
0	A/D 変換終了による割り込み(ADI)要求を禁止
1	A/D 変換終了による割り込み(ADI)要求を許可

関係ないので、0 のままにしておきます。

・ビット 5:A/D スタート(ADST)

A/D 変換の開始／停止を選択します。A/D 変換中は 1 を保持します。また、ADST ビットは A/D 外部トリガ入力端子(ADTRG)により 1 にセットすることもできます。

ADST	説明
0	A/D 変換を停止
1	(1)単一モード:A/D 変換を開始し、変換が終了すると自動的に 0 にクリア (2)スキャンモード:A/D 変換を開始し、ソフトウェア、リセット、またはスタンバイモードによって 0 にクリアされるまで選択されたチャネルを順次連続変換

初期化するときは"0"にしておきます。すべて設定後、"1"にします。

・ビット 4:スキャンモード(SCAN)

A/D 変換のモードを、単一モード／スキャンモードから選択します。**モードの切り替えは、ADST＝0 の状態で行ってください**。

SCAN	説明
0	単一モード
1	スキャンモード

AN3～0(P73～P70)を使用します。"1"としてスキャンモードにします。ADST=0 の状態(bit5 が 0 の状態)で変更します。

・ビット 3:クロックセレクト(CKS)

A/D 変換時間の設定を行います。

変換時間の切り替えは、ADST＝0 の状態で行ってください。

CKS	説明
0	変換時間＝266 ステート(max)
1	変換時間＝134 ステート(max)

変換時間は精度と全く関係ありません。ただ速いか遅いかの違いです。速い方が良いので"1"にします。

・ビット 2-0:チャネルセレクト(CH2-0)

SCAN ビットと共にアナログ入力チャネルを選択します。

チャネル選択と切り替えは、ADST＝0 の状態で行ってください。

CH2	CH1	CH0	単一モード	**スキャンモード**
0	0	0	AN0	AN0
0	0	1	AN1	AN0、AN1
0	1	0	AN2	AN0～AN2
0	**1**	**1**	AN3	**AN0～AN3**
1	0	0	AN4	AN4
1	0	1	AN5	AN4、AN5
1	1	0	AN6	AN4～AN6
1	1	1	AN7	AN4～AN7

今回はスキャンモードで A/D 変換します。

スキャンモードは、複数チャネル(1 チャネルを含む)のアナログ入力を常にモニタするような応用に適しています。今回は AN0～AN3 を使用するので、"011"を設定します。A/D 変換は ADST ビットが 1 にセットされると、AN0～AN3 が1チャネルごとに繰り返し A/D 変換され続けます。

■AD_DRA、AD_DRB、AD_DRC、AD_DRD(A/D データレジスタ A～D)の設定内容

ビット:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AD_DRx:	AD9	AD8	AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0	－	－	－	－	－	－

AD_DR は A から D まで 4 つあり、AN0～AN7 の端子に入力して A/D 変換された電圧値が右にシフトされた状態で格納されます。**プログラムの中で使うには 6 ビット右にシフトして使います**。AN0～7 と AD_DRA～D との対応は決まっていて下表のようになっています。

電圧を入力する AN 端子	読み込むレジスタ
AN0 と AN 4	AD_DRA
AN 1 と AN 5	AD_DRB
AN 2 と AN 6	AD_DRC
AN 3 と AN 7	AD_DRD

今回のプログラムでは、下記のように使用しています。

接続されている AN 端子	接続名	読み込むレジスタ
AN0	未接続	
AN1	ボリューム	AD_DRB
AN2	右アナログセンサ	AD_DRC
AN3	左アナログセンサ	AD_DRD

6.2.6 ITU2 パルスカウントの設定

```
421 :       /* ITU2 エンコーダ */
422 :       ITU2_TCR = 0x14;                    /* PA0パルス入力端子          */
```

ITU2 を外部パルス入力用に設定することにより、ロータリエンコーダのパルスをカウントします。

■ITU2_TCR(タイマコントロールレジスタ)の設定内容

ビット:	7	6	5	4	3	2	1	0
ITU2_TCR:	−	CCLR1	CCLR0	CKEG1	CKEG0	TPSC2	TPSC1	TPSC0
設定値:	0	0	0	1	0	1	0	0
16 進数:	1				4			

・ビット 6,5:カウンタクリア 1,0

CNT のカウンタクリア要因を選択します。

CCLR1	CCLR0	説明
0	0	CNT のクリア禁止
0	1	GRA のコンペアマッチ／インプットキャプチャで CNT をクリア
1	0	GRB のコンペアマッチ／インプットキャプチャで CNT をクリア
1	1	同期クリア

今回は ITU2_CNT をクリアする必要はないので、クリアしません。

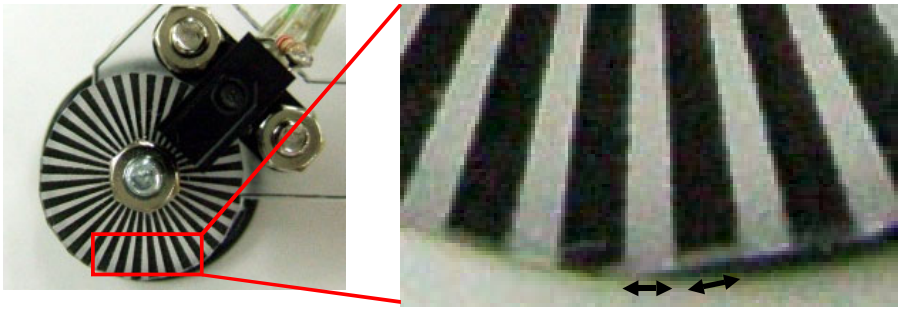
・ビット 4,3:クロックエッジ 1,0

外部クロック選択時に、外部クロックの入力エッジを選択します。

CKEG1	CKEG0	説明
0	0	立ち上がりエッジでカウント
0	1	立ち下がりエッジでカウント
1	0	立ち上がり／立ち下がりの両エッジでカウント
1	1	立ち上がり／立ち下がりの両エッジでカウント

外部パルスの立ち上がり、立ち下がりで ITU2_CNT が＋1します。例えば、ロータリエンコーダが 1 周 36 組の透明・黒部分があれば、倍の 72 カウントとなります。

ただし、立ち上がり／立ち下がりの両エッジでカウントの設定にする場合は、円盤の黒い部分と透明部分(または穴の空いている部分と空いていない部分)の間隔が同じである必要があります(下写真)。



同じ間隔なら 0x14 にできます

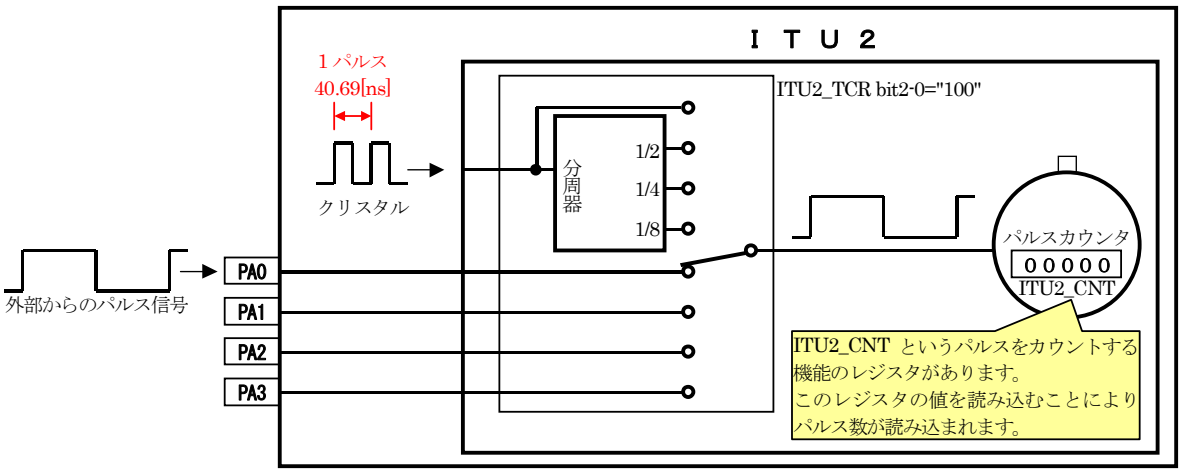
間隔が違う場合は、「ITU2_TCR = 0x04;」として、立ち上がりのみにします。

・ビット 2～0:タイマプリスケーラ 2～0

CNT のカウントクロックを選択します。

TPSC2	TPSC1	TPSC0	説明
0	0	0	内部クロック:φでカウント
0	0	1	内部クロック:φ／2でカウント
0	1	0	内部クロック:φ／4でカウント
0	1	1	内部クロック:φ／8でカウント
1	0	0	外部クロックA:TCLKA 端子(PA0)でカウント
1	0	1	外部クロックB:TCLKB 端子(PA1)でカウント
1	1	0	外部クロックC:TCLKC 端子(PA2)でカウント
1	1	1	外部クロックD:TCLKD 端子(PA3)でカウント

イメージとしては下図のようになります。



割り込みやPWMでは、CPUボード上のクリスタルのクロックによってカウンタの値を＋1していましたが、今回はロータリエンコーダによるパルスで増やしていきます。ポートAのbit3～bit0のどれかを選ぶことができます。今回は、PA0に接続します。

外部パルスのカウントは、ポートAのbit3～bit0を必ず使用します。それ以外の端子では外部パルスはカウントできません。

6.2.7 ITU0、ITU1 左前モータ、右前モータの設定

ITU0、ITU1 を PWM モードで使用します。ITU0 で左前モータ、ITU1 で右前モータを制御します。

```
411 :      /* ITU0 左前モータ用PWM */
412 :      ITU0_TCR = 0x23;                  /* カウンタ、クリアの設定      */
413 :      ITU0_GRA = MOTOR_CYCLE;           /* PWM周期                     */
414 :      ITU0_GRB = 0;                     /* デューティ比設定            */
415 :
416 :      /* ITU1 右前モータ用PWM */
417 :      ITU1_TCR = 0x23;                  /* カウンタ、クリアの設定      */
418 :      ITU1_GRA = MOTOR_CYCLE;           /* PWM周期                     */
419 :      ITU1_GRB = 0;                     /* デューティ比設定            */

中略

437 :      ITU_MDR = 0x03;                   /* PWMモード設定               */
438 :      ITU_STR = 0x1f;                   /* ITUのカウントスタート       */
```

各レジスタの意味は、下記のとおりです。

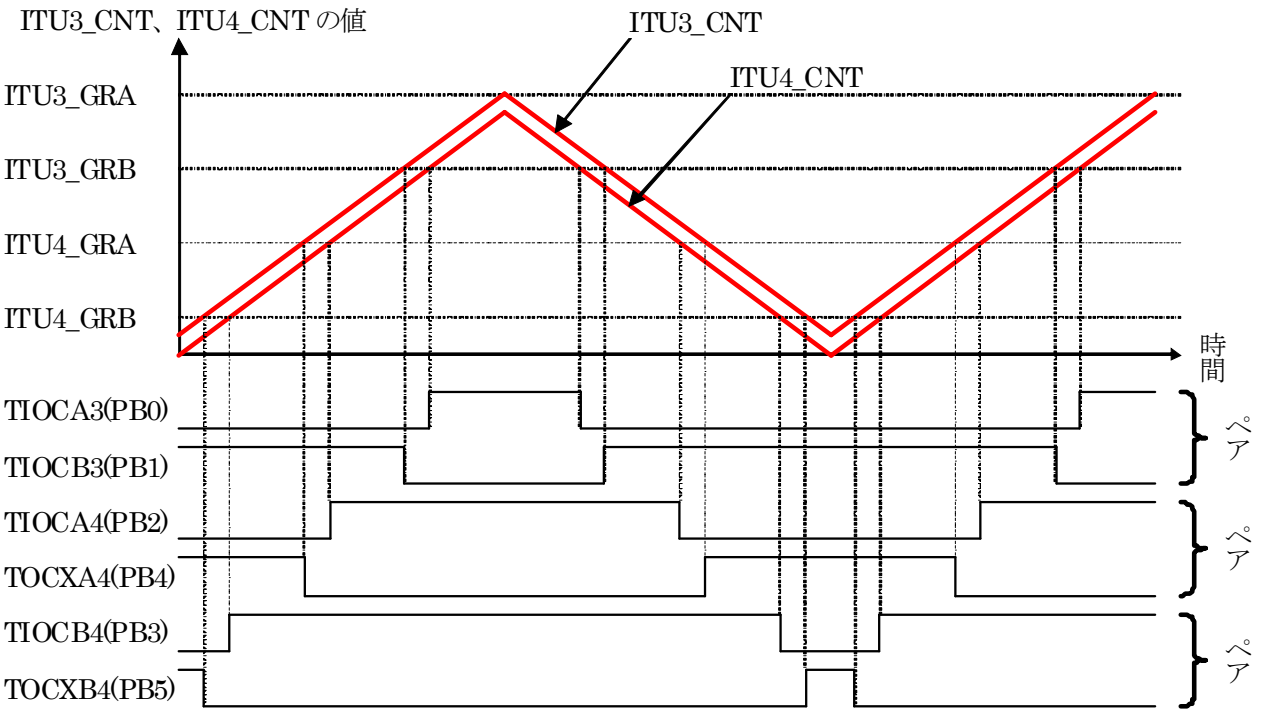
<table>
<tr><th>設定するレジスタ</th><th>詳細</th></tr>
<tr><td>ITU0_TCR
(ITU1_TCR も同様です)</td><td>ITU0_CNT の＋1する時間、クリア要因の設定をします。
0x20…40.69[ns]でカウント　0x21…81.38[ns]でカウント
0x22…162.76[ns]でカウント　0x23…325.52[ns]でカウント
今回は、0x23 を設定します。</td></tr>
<tr><td>ITU0_GRA</td><td>周期を設定します。計算は、「設定したい周期÷ITU0_CNT のカウント時間－1」です。周期 1ms、カウント時間 325.52ns なら
$(1\times10^{-3})\div(325.52\times10^{-9})-1=3071$</td></tr>
<tr><td>ITU0_GRB</td><td>出力したい PWM 波形の ON 幅を設定します。
計算は、「設定したい幅÷ITU3_CNT のカウント時間－1」です。
最初は 0%にするため、0 を設定します。</td></tr>
<tr><td>ITU_MDR</td><td>ITU を PWM モードで使用するか選択します。
<table>
<tr><td>bit</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>意味</td><td>0固定</td><td>0固定</td><td>0固定</td><td>ITU4
PWM モード
0:通常動作
1:PWM モード</td><td>ITU3
PWM モード
0:通常動作
1:PWM モード</td><td>ITU2
PWM モード
0:通常動作
1:PWM モード</td><td>ITU1
PWM モード
0:通常動作
1:PWM モード</td><td>ITU0
PWM モード
0:通常動作
1:PWM モード</td></tr>
<tr><td>設定値</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr>
</table></td></tr>
<tr><td>ITU_STR</td><td>ITU のカウンタ(CNT)を動作させる設定です。
ITU0:左前モータ用、ITU1:右前モータ用、ITU2:ロータリエンコーダ用
ITU3,4:相補 PWM モードとして使用
よって、ITU0～4 まですべて使用します。
<table>
<tr><td>bit</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>意味</td><td>0固定</td><td>0固定</td><td>0固定</td><td>ITU4
カウント動作
0:しない
1:する</td><td>ITU3
カウント動作
0:しない
1:する</td><td>ITU2
カウント動作
0:しない
1:する</td><td>ITU1
カウント動作
0:しない
1:する</td><td>ITU0
カウント動作
0:しない
1:する</td></tr>
<tr><td>設定値</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr>
</table></td></tr>
</table>

6.2.8 相補PWMモードの設定

```
424 :       /* ITU3,4 右後、左後、サーボモータ用PWM兼、割り込み */
425 :       ITU3_TCR = 0x03;
426 :       ITU4_TCR = 0x03;
427 :       ITU_FCR  = 0x2e;                    /* ITU3,4で相補PWMモード          */
428 :       ITU_TOCR = 0x12;                    /* TIOCB3,TOCXA4,TOCXB4は反転     */
429 :       ITU3_IER = 0x01;                    /* TCNT = GRAによる割込み制御     */
430 :       ITU3_CNT = MOTOR_OFFSET;
431 :       ITU4_CNT = 0;
432 :       ITU3_GRA = MOTOR_CYCLE / 2 + MOTOR_OFFSET - 2;  /* PWM周期設定        */
433 :       ITU3_BRB = ITU3_GRB = MOTOR_CYCLE / 2 - 2;
434 :       ITU4_BRA = ITU4_GRA = MOTOR_CYCLE / 2 - 2;
435 :       ITU4_BRB = ITU4_GRB = MOTOR_CYCLE / 2 - 2;
436 :
437 :       ITU_MDR = 0x03;                     /* PWMモード設定                  */
438 :       ITU_STR = 0x1f;                     /* ITUのカウントスタート          */
```

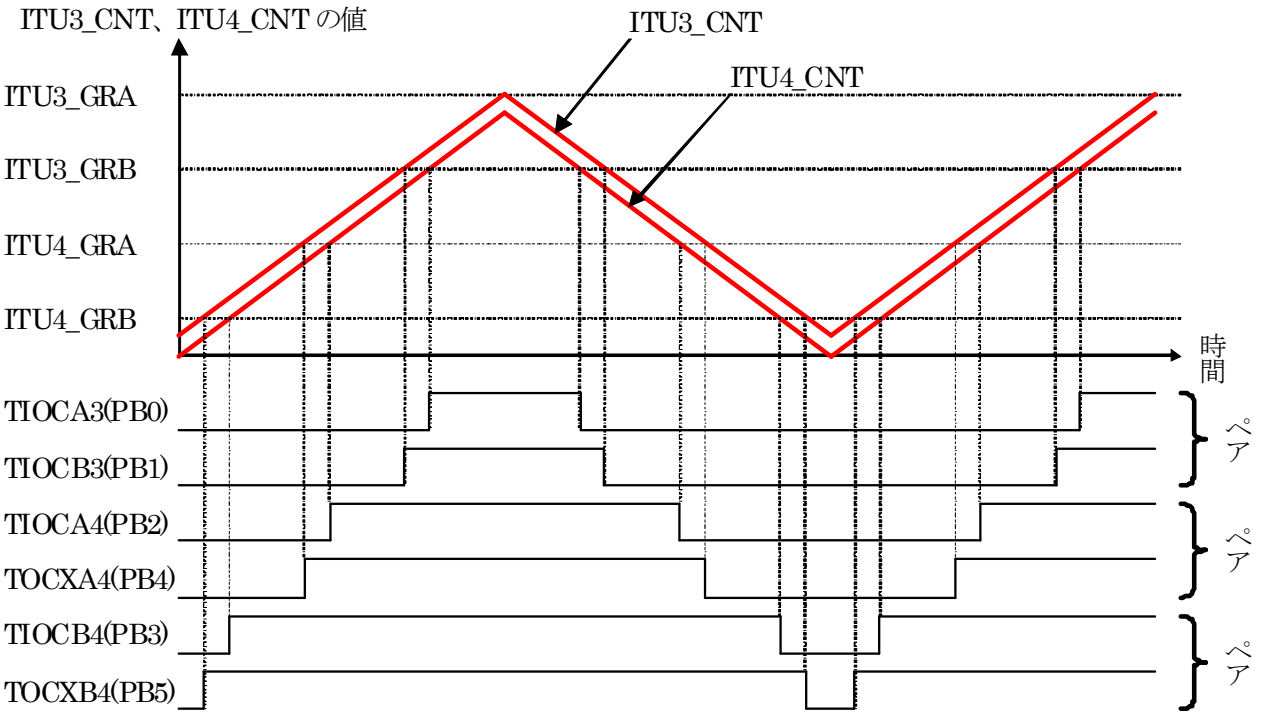
H8/3048F-ONE には、ITU3、ITU4 を1組として使用し、レジスタの設定により下記のような 1 組 2 つの波形、合計 3 組の波形を出力することができます。これを**相補 PWM モード**といいます。

標準では、下記のような波形が出力されます。



レジスタ名	機能
ITU3_GRA	周期を設定
ITU3_GRB	PB0、PB1 端子の PWM 出力のデューティ比を設定
ITU4_GRA	PB2、PB4 端子の PWM 出力のデューティ比を設定
ITU4_GRB	PB3、PB5 端子の PWM 出力のデューティ比を設定

ITU_TOCR レジスタにて、それぞれの波形をプログラムで反転させることができます。ここでは PB1、PB4、PB5 の波形を反転させます。ITU_TOCR に 0x12 を書き込みます。下記のような波形となります。

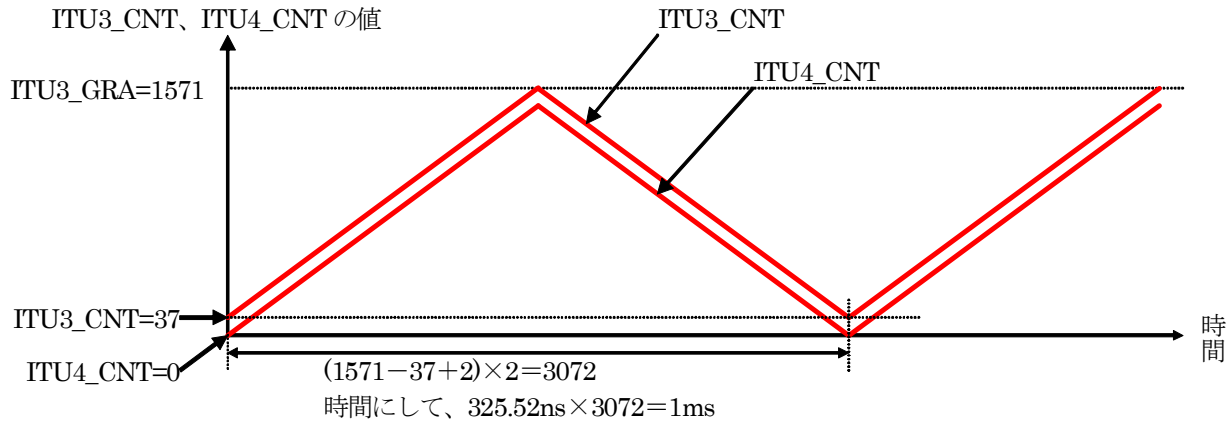


(1) 周期の設定

周期の設定は、ITU3_GRA に値を設定することになります。下図のように、上って下がって1周期なので目的の周期の 1/2 を設定します。さらにオフセット分の 37 を加えます。

ITU3_GRA＝MOTOR_CYCLE／2＋オフセット分
＝3072／2＋37
＝1573

となります。さらに、オーバーフロー時、アンダーフロー時のカウント分を考慮し-2 します。最終的には **1571** を設定します。これで周期が 1ms になります。



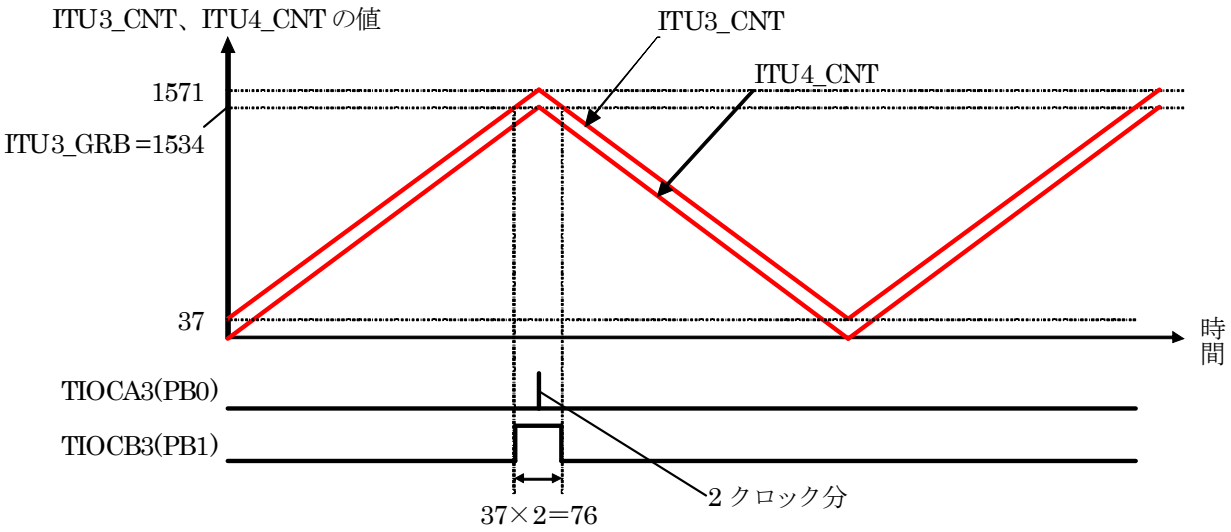
PWM のデューティ比を設定するには ITU3_GRB、ITU4_GRA、ITU4_GRB を設定します。

(2) デューティ比 0%の設定

例として、ITU3_GRB に値を設定することとします。
デューティ比を少なくするには、ITU3_GRB の値を多くすれば、ON する幅が少なくなります。0%にするには ITU3_GRB に次の値を設定します。

周期 0%のときの ITU3_GRB　＝ITU4_GRA－オフセット分
＝1571－37
＝1534

下図に、ITU3_GRB に 1534 を設定したときの波形を示します。



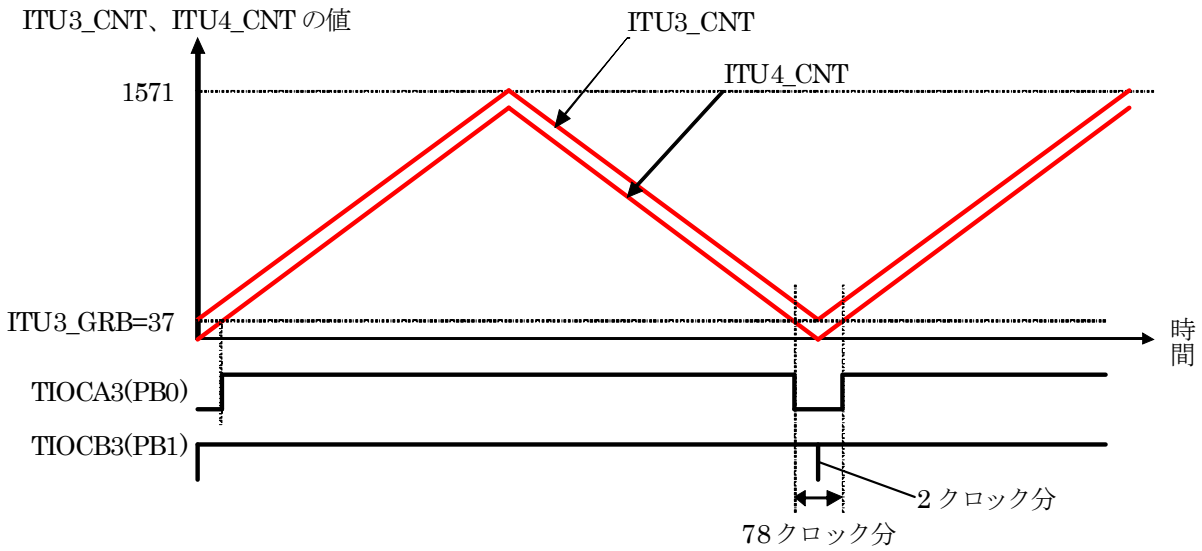
ITU3_GRB に 1534 を設定にすると、
・PB0 は、2 クロック分の約 650ns だけ ON
・PB1 は、76 クロック分の約 2.47μs だけ ON
します。便宜上、この状態を 0%とします。相補 PWM モードの仕様により、これ以上短くできません。
PB0=″0″、PB1=″1″のとき、モータの動作はフリーです。PB0=″1″、PB1=″1″のとき、モータの動作は正転です。正転状態は 650ns しかなくデジタルトランジスタや FET に遅延があるので、実際は正転はせずにフリー動作になります。

(3) デューティ比 100%の設定

デューティ比を多くするには、ITU3_GRB の値を小さくすれば、ON する幅が多くなります。100%にするには ITU3_GRB に次の値を設定します。

周期 100%のときの ITU3_GRB ＝オフセット分
＝37

下図に、ITU3_GRB に 37 を設定したときの波形を示します。



ITU3_GRB に 37 を設定にすると、
・PB0 は、78 クロック分の約 2.54μs だけ ON
・PB1 は、2 クロック分の約 650ns だけ ON
します。便宜上、この状態を 100%とします。相補 PWM モードの仕様により、これ以上長くできません。

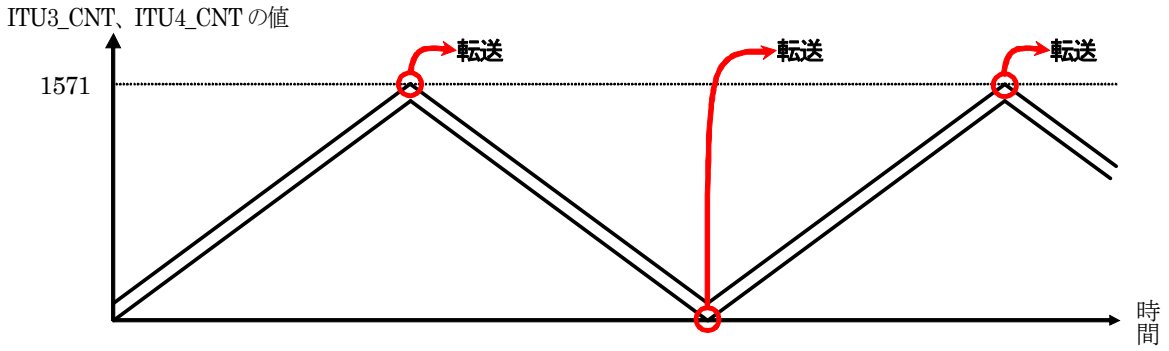
0%から 100%の間を 100 等分すれば、1～99%のデューティ比を設定することができます。

(4) バッファレジスタの使用

相補 PWM モードを使用した場合、デューティ比を決めるのは、IUT3_GRB、ITU4_GRA、ITU4_GRB です。これらのレジスタの値を直接操作すると、ITU3_CNT や ITU4_CNT の値によっては、一致するタイミングがずれて、予期せぬ波形が出力されることがあります。

そこで、プログラムで変更するのは、バッファレジスタというレジスタに書き込み、下記○部分のタイミングになったときに、バッファレジスタからそれぞれのレジスタに転送を行います。転送は設定により自動で行います。転送元と転送先の関係を、下表に示します。

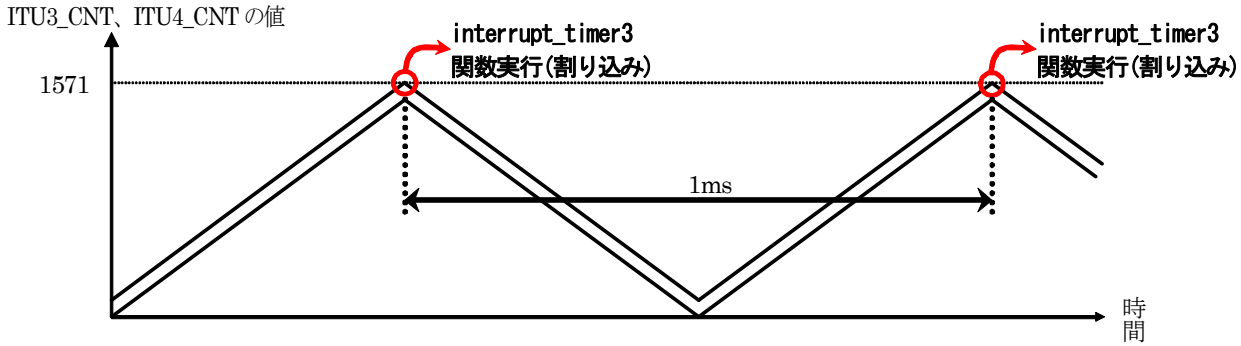
バッファレジスタ(転送元)	転送先のレジスタ
ITU3_BRB	ITU3_GRB
ITU4_BRA	ITU4_GRA
ITU4_BRB	ITU4_GRB



(5) 割り込みの使用

マイコンカーキット Ver.4 などのサンプルプログラムは、ITU0 を割り込み専用で使用しています。今回も同じようにできれば良いのですが、5 個のモータ制御、ロータリエンコーダのパルス入力で ITU をすべて使っています。そこで、ITU3、4 を**相補 PWM モード兼 1ms ごとの割り込み**として使用するように設定します。

ITU3_IER に 0x01 を設定することにより、ITU3_GRA と ITU3_CNT の値が一致すると割り込みが発生します。src ファイル内で、ITU3 の IMIA3 割り込みが発生したときのジャンプ先を interrupt_timer3 にしています。interrupt_timer3 関数内で、タイマ用変数のカウントアップなど、1ms ごとに処理を行っています。下図にその様子を示します。



(6) レジスタ設定のまとめ

<table>
<tr><th>設定するレジスタ</th><th>詳細</th></tr>
<tr><td>ITU3_TCR
ITU4_TCR</td><td>ITU3_CNT の＋1する時間を設定します。今回、ITU3_CNT は 1 カウント 325.52ns で進みます。
<table>
<tr><td>bit</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>意味</td><td>0固定</td><td>0固定</td><td>0固定</td><td>0固定</td><td>0固定</td><td colspan="3">000…40.69[ns]でカウント
001…81.38[ns]でカウント
010…162.76[ns]でカウント
011…325.52[ns]でカウント</td></tr>
<tr><td>設定値</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr>
</table></td></tr>
<tr><td>ITU_FCR</td><td>相補 PWM モードの設定と、バッファレジスタを使用するかの設定します。
<table>
<tr><td>bit</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>意味</td><td>0固定</td><td>0固定</td><td colspan="2">00…ITU3,4 通常使用
01…ITU3,4 通常使用
10…ITU3,4 で
相補 PWM モード
11…ITU3,4 でリセット
同期 PWM モード</td><td>ITU4_GRB と ITU4_BRB をバッファ動作にするか
0…しない
1…する</td><td>ITU4_GRA と ITU4_BRA をバッファ動作にするか
0…しない
1…する</td><td>ITU3_GRB と ITU3_BRB をバッファ動作にするか
0…しない
1…する</td><td>ITU3_GRA と ITU3_BRA をバッファ動作にするか
0…しない
1…する</td></tr>
<tr><td>設定値</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td></tr>
</table>
相補 PWM モードにすることで、PB0～PB5 端子からは PWM 波形がされます。</td></tr>
<tr><td>ITU3_GRA</td><td>周期を設定します。
設定したい周期の値　＝設定したい周期÷ITU3_CNT のカウント時間
＝$(1\times10^{-3})\div(325.52\times10^{-9})$
＝3072
ITU3_GRA に設定する値　＝設定したい周期の値／2＋オフセット－2
＝3072／2＋37－2
＝1571</td></tr>
<tr><td>ITU3_BRB
(ITU3_GRB)</td><td>PB0、PB1 の幅を設定します。
計算は、0%は ITU3_GRA-オフセット値である 1534、100%はオフセット値である 37 です。1534～37 の間で 100 等分して、0～100%まで設定できるように speed 関数で計算します。</td></tr>
<tr><td>ITU4_BRA
(ITU4_GRA)</td><td>PB2、PB4 の幅を設定します。
計算は、ITU3_BRB と同様です。</td></tr>
<tr><td>ITU4_BRB
(ITU4_GRB)</td><td>PB3、PB5 の幅を設定します。
計算は、ITU3_BRB と同様です。</td></tr>
<tr><td>ITU3_IER</td><td>割り込みを許可するかしないかを設定します。
今回は、(ITU3_GRA+1)=ITU3_CNT で割り込みが発生するようにします。この式が成り立つのは 1ms ごとなので、1ms ごとに割り込みが発生します。</td></tr>
<tr><td>ITU_STR</td><td>ITU のカウンタ(CNT)を動作させる設定です。
<table>
<tr><td>bit</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>意味</td><td>0固定</td><td>0固定</td><td>0固定</td><td>ITU4_CNT カウント動作
0:しない
1:する</td><td>ITU3_CNT カウント動作
0:しない
1:する</td><td>ITU2_CNT カウント動作
0:しない
1:する</td><td>ITU1_CNT カウント動作
0:しない
1:する</td><td>ITU0_CNT カウント動作
0:しない
1:する</td></tr>
<tr><td>設定値</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr>
</table>
相補 PWM モードで使用する ITU3 と ITU4 の他にも ITU0、1、2 も使用するので、ITU0～4 をカウント動作する設定にします。</td></tr>
</table>

6.2.9 割り込みプログラム

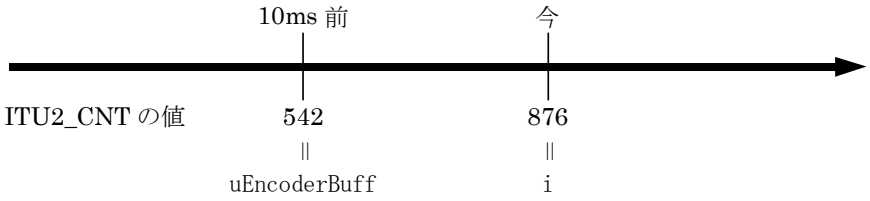
```
441 :  /*****************************************************************************/
442 :  /* ITU3 割り込み処理                                                         */
443 :  /*****************************************************************************/
444 :  #pragma interrupt( interrupt_timer3 )
445 :  void interrupt_timer3( void )
446 :  {
447 :      unsigned int    i;
448 :
449 :      ITU3_TSR &= 0xfe;                   /* フラグクリア                  */
450 :      cnt1++;
451 :
452 :      /* サーボモータ制御 */
453 :      servoControl();
454 :
455 :      /* ブザー処理 */
456 :      beepProcessS();
```

449 行…割り込み時に"1"になったフラグをクリアします。
450 行…タイマ変数 cnt1 を1つ増加させます。cnt1 は 1ms ごとに増加していきます。
453 行…サーボモータの PWM 値を計算します。
456 行…ブザーを鳴らす処理を行います。

```
458 :      /* エンコーダ制御 */
459 :      iTimer10++;
460 :      if( iTimer10 >= 10 ) {
461 :          iTimer10 = 0;
462 :          i = ITU2_CNT;
463 :          iEncoder       = i - uEncoderBuff;
464 :          lEncoderTotal += iEncoder;
465 :          if( iEncoder > iEncoderMax )
466 :                      iEncoderMax = iEncoder;
467 :          uEncoderBuff = i;
```

459 行…iTimer10 変数を増加させます。
460 行…iTimer10 変数が 10 以上なら次の行を実行します。ITU0 割り込みは、1ms ごとに実行されますが、エンコーダ関連処理は 10ms ごとに処理します。そのため、実行回数を数えて 10 回目なら次の行に移りエンコーダ処理を行います。それ以下なら 492 行へ移りエンコーダ処理をしません。
461 行…iTimer10 変数を 0 にして、実行回数を数え直します。
462 行…現在のカウント値 ITU2_CNT を変数 i に代入します。なぜ、ITU2_CNT の値を直接使わないのでしょうか。ITU2_CNT の値は、エンコーダからのパルスが入力されるたびに増加していきます。プログラムが 1 行進むと違う値になっているかもしれません。そのため、いったん別な変数に代入して、この値をプログラムでは最新値として使います。
463 行…最新の 10ms 間のエンコーダのカウント数を計算しています。計算は、
10ms 間のエンコーダのカウント数 = i - uEncoderBuff
としています。i は現在のカウンタ値、uEncoderBuff のカウンタ値です。言い換えれば、
10ms 間のエンコーダのカウント数 = 現在のカウンタ値 - 1回前のカウンタ値
となります。ITU2_CNT は 16 ビット幅の符号無し int 型の大きさなので、0～65,535 までカウントされます。65,535 の次は 0 に戻ってカウントを続けます。そのため、前の値を覚えておき、現在の値を引くことにより前回と今回の差分がでます。これが 10ms 間のパルス数です。

図解すると下記のようなイメージです。



i- uEncoderBuff の値が、最新の 10ms 間のエンコーダパルス値となる

464 行…エンコーダの積算値を計算しています。計算は、

積算値 = 積算値 + 最新の 10ms 間のエンコーダ値

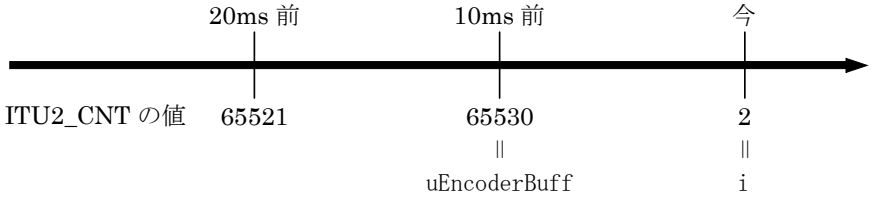
です。積算値は、long 型ですので、21 億までカウントできます。1m で 1000 カウントとすると、約 2,100,000m(=2,100km)まで計算できます。マイコンカーでは十分です。

465 行…iEncoder と iEncoderMax 変数を比較しています。iEncoder 変数の方の値が大きければ次の行へ進みます。

466 行…iEncoderMax 変数に、iEncoder 変数の値を代入します。iEncoderMax 変数には 10ms 間に計測したパルス数の最大値が代入されます。走行後、この変数をチェックすればマイコンカーの瞬間最大速度が分かります。

467 行…i には現在の ITU2_CNT の値が入っています。最後に uEncoderBuff 変数に i の値を代入します。今は uEncoderBuff の値は最新値を代入したことになりますが、次にエンコーダ関連処理をするのは 10ms 後なので、そのときの uEncoderBuff は 10ms 前の値となります。

※ITU2_CNT が 65535 から 0 になったとき



i- uEncoderBuff の値が、最新の 10ms 間のエンコーダパルス値となる

ITU2_CNT は、符号無しの 16 ビット幅です。上限は 65535 で次が 0 に戻ります。

10ms 間のパルス値を計算するのには、

(現在の ITU2_CNT)－(10ms 前の ITU2_CNT)

です。図のように、10ms 前の ITU2_CNT の値が 65530、現在の値が 0 に戻って 2 になった場合、どのようになるのでしょうか。

普通に考えると、

(現在の ITU2_CNT)－(10ms 前の ITU2_CNT)＝ 2 - 65530 ＝ -65528

となり、とんでもない値になります。

16 進数に直すと、

0x0002－0xfffa＝0xffff0008

ただし、**計算結果も符号無し 16 ビット幅**なので、

0x0002－0xfffa＝0x0008

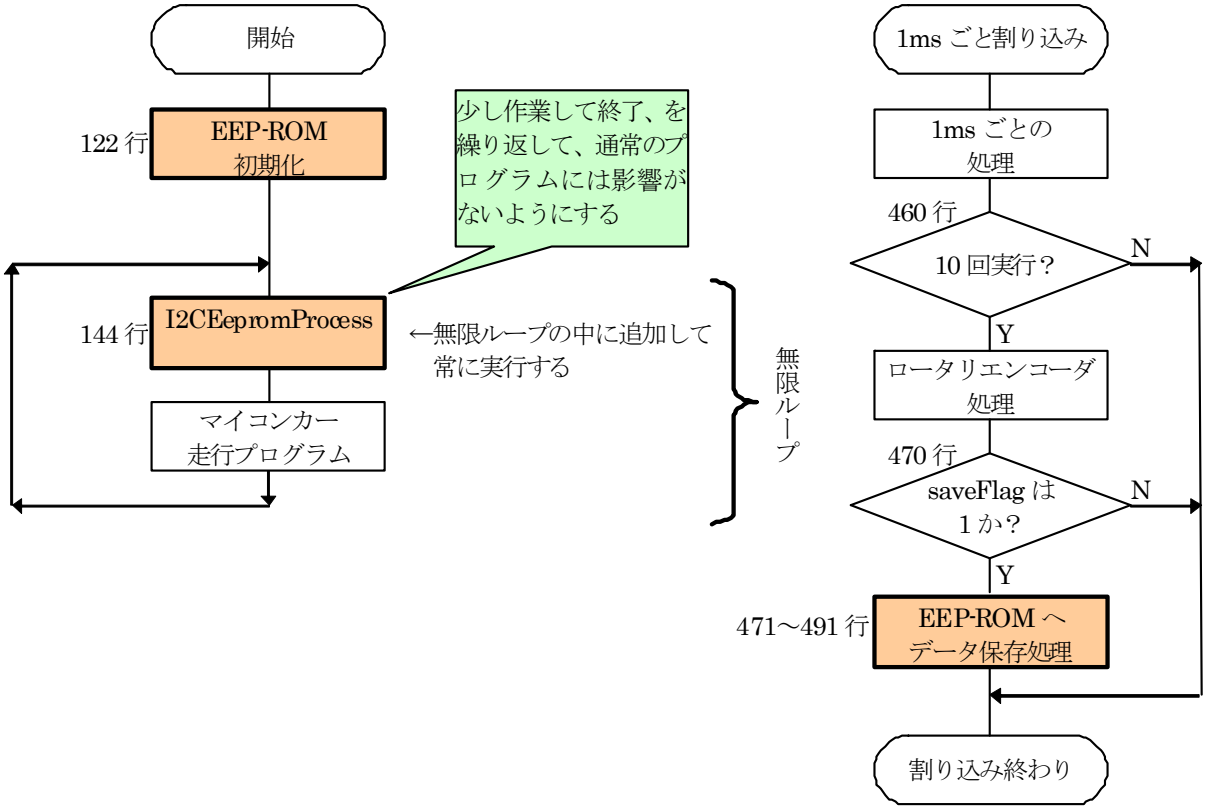
となり、結果は 8 になります。また、カウント分を計算すると、65531,65532,65533,65534,65535,0,1,2 と 8 カウント分になり計算は合います。

このように、符号無し 16 ビット幅で計算しているので、いったん 0 に戻ってもきちんと計算されます。

```
469 :          /* データ保存関連 */
470 :          if( saveFlag ) {
471 :              saveData[0] = pattern;      /* パターン                        */
472 :              saveData[1] = (center_inp()<<4) + sensor_inp(); /* センサ   */
473 :              i = getServoAngle();        /* 角度                            */
474 :              saveData[2] = i >> 8;
475 :              saveData[3] = i & 0xff;
476 :              i = getAnalogSensor();      /* アナログセンサ値                */
477 :              saveData[4] = i >> 8;
478 :              saveData[5] = i & 0xff;
479 :              /*  6はハンドル関数内でサーボPWM保存        */
480 :              /*  7はモータ関数内で左前モータPWM値保存    */
481 :              /*  8はモータ関数内で右前モータPWM値保存    */
482 :              /*  9はモータ関数内で左後モータPWM値保存    */
483 :              /* 10はモータ関数内で右後モータPWM値保存    */
484 :              saveData[11] = iEncoder;    /* エンコーダ                      */
485 :              saveData[12] = 0;
486 :              saveData[13] = 0;
487 :              saveData[14] = 0;
488 :              saveData[15] = 0;
489 :              setPageWriteI2CEeprom( saveIndex, 16, saveData );
490 :              saveIndex += 16;
491 :              if( saveIndex >= 0x8000 ) saveFlag = 0;
492 :          }
493 :      }
494 :  }
```

470 行で、saveFlag 変数をチェック、0 以外なら 471 行から 491 行を実行して走行データを EEP-ROM に保存する処理を行います。この処理は、10ms ごとに行われます。

保存は下記の手順で行います。



saveData という配列を用意し、パソコンに転送したいデータを保存します。EEP-ROM は 1 データ 1byte なので、-128～127(または 0～255)の範囲しか保存できません。int 配列の値は、2bytes なので、2 つの配列に分けて保存します。保存数は、2 の n 乗である 1 個、2 個、4 個、8 個、16 個のどれかにします。12 個だけ保存と言うことはできません。余りの配列 12～15 には特に何も設定しません。

下記に、サンプルプログラムで EEP-ROM に保存する内容を示します。

配列	保存変数名	内容
saveData[0]	pattern	パターン番号
saveData[1]	(center_inp()<<4) + sensor_inp()	センタデジタルセンサとデジタルセンサ
saveData[2]	getServoAngle()の上位 8bit	ハンドル角度の上位 8bit
saveData[3]	getServoAngle()の下位 8bit	ハンドル角度の下位 8bit
saveData[4]	getAnalogSensor()の上位 8bit	アナログセンサ値の上位 8bit
saveData[5]	getAnalogSensor()の下位 8bit	アナログセンサ値の下位 8bit
saveData[6]		ハンドル関数内でサーボ PWM 保存
saveData[7]		モータ関数内で左前モータ PWM 値保存
saveData[8]		モータ関数内で右前モータ PWM 値保存
saveData[9]		モータ関数内で左後前モータ PWM 値保存
saveData[10]		モータ関数内で右後前モータ PWM 値保存
saveData[11]	iEncoder	ロータリエンコーダ値
saveData[12]		なし
saveData[13]		なし
saveData[14]		なし
saveData[15]		なし

次に setPageWriteI2CEeprom 関数を実行し、EEP-ROM に書き込む**準備を行います。**この関数の引数は下記のように設定します。実際に EEP-ROM へ書き込む処理は、I2CEepromProcess 関数で行っています。

```
setPageWriteI2CEeprom( saveIndex, 16, saveData );
                              ↑     ↑     ↑
                              1     2     3
```

1…EEP-ROM の何番地に保存するか指定します。
2…保存するデータ数を指定します。2 の n 乗個にします(1、2、4、8、16 個など)。
3…保存データの格納されている配列のアドレスを指定します。

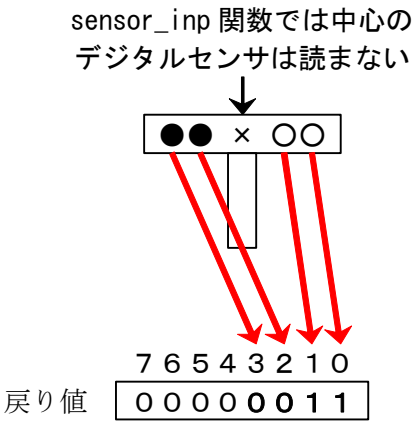
```
490 :               saveIndex += 16;
491 :               if( saveIndex >= 0x8000 ) saveFlag = 0;
```

490 行で、EEP-ROM へ保存するアドレスを+16 して、次に保存するアドレスをセットしています。
491 行で、アドレスの上限である 0x8000(2^{15})を超えていないかチェック、超えていたらデータ保存を終了します。

6.2.10 アナログセンサ基板TypeSのデジタルセンサ値読み込み

```
496 :  /*************************************************************************/
497 :  /* アナログセンサ基板TypeSのデジタルセンサ値読み込み                      */
498 :  /* 引数    なし                                                          */
499 :  /* 戻り値 左端、左中、右中、右端のデジタルセンサ 0:黒 1:白                */
500 :  /*************************************************************************/
501 :  unsigned char sensor_inp( void )
502 :  {
503 :      unsigned char sensor;
504 :
505 :      sensor  = (~PADR & 0x08) | (~P8DR & 0x07);
506 :
507 :      return sensor;
508 :  }
```

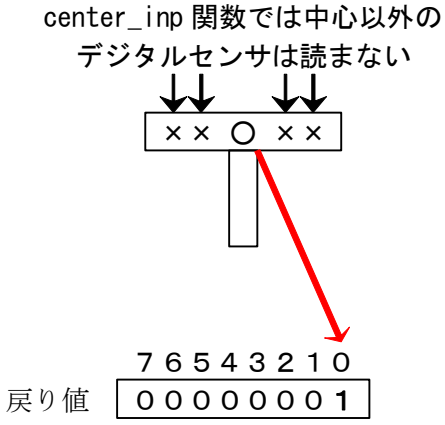
アナログセンサ基板 TypeS のデジタルセンサ 4 個を読み込む関数です。デジタルセンサは黒で"1"、白で"0"なのでポートから読み込むときに"～"(チルダ)をつけて反転させます。中心のデジタルセンサ値を読み込む関数は、center_inp 関数です。



6.2.11 アナログセンサ基板TypeSの中心デジタルセンサ読み込み

```
510 :  /*****************************************************************************/
511 :  /* アナログセンサ基板TypeSの中心デジタルセンサ読み込み                       */
512 :  /* 引数    なし                                                              */
513 :  /* 戻り値 中心デジタルセンサ 0:黒 1:白                                       */
514 :  /*****************************************************************************/
515 :  unsigned char center_inp( void )
516 :  {
517 :      unsigned char sensor;
518 :
519 :      sensor  = ~PADR & 0x20;             /* アナログセンサ基板TypeSの     */
520 :      sensor  = !!sensor;                 /* 中心デジタルセンサ読み込み    */
521 :
522 :      return sensor;
523 :  }
```

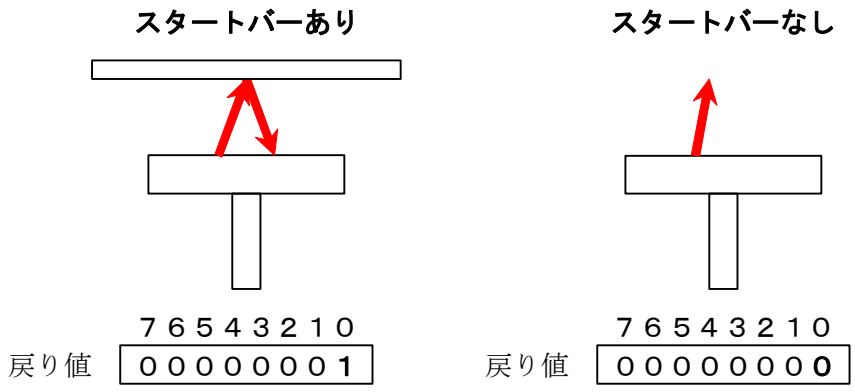
アナログセンサ基板 TypeS の中心デジタルセンサ 1 個を読み込む関数です。



6.2.12 アナログセンサ基板TypeSのスタートバー検出センサ読み込み

```
525 :  /************************************************************************/
526 :  /* アナログセンサ基板TypeSのスタートバー検出センサ読み込み              */
527 :  /* 引数    なし                                                         */
528 :  /* 戻り値 0:スタートバーなし 1:スタートバーあり                          */
529 :  /************************************************************************/
530 :  unsigned char startbar_get( void )
531 :  {
532 :      unsigned char sensor;
533 :
534 :      sensor  = ~P8DR & 0x08;             /* アナログセンサ基板TypeSの    */
535 :      sensor  = !!sensor;                 /* スタートバーセンサ読み込み   */
536 :
537 :      return sensor;
538 :  }
```

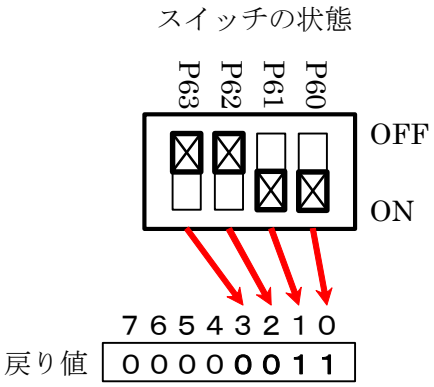
アナログセンサ基板 TypeS のスタートバー検出センサの状態を読み込む関数です。



6.2.13 CPUボード上のディップスイッチ値読み込み

```
540 :  /*****************************************************************************/
541 :  /* CPUボード上のディップスイッチ値読み込み                                   */
542 :  /* 戻り値 スイッチ値 0～15                                                    */
543 :  /*****************************************************************************/
544 :  unsigned char dipsw_get( void )
545 :  {
546 :      unsigned char sw;
547 :
548 :      sw  = ~P6DR;                        /* ディップスイッチ読み込み      */
549 :      sw &= 0x0f;
550 :
551 :      return  sw;
552 :  }
```

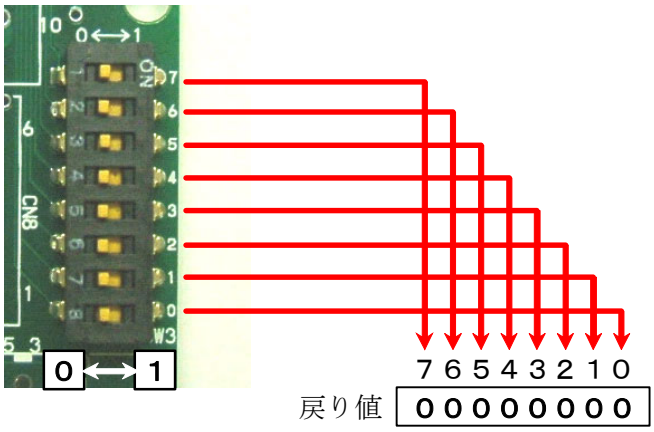
CPU ボードのディップスイッチの値を読み込む関数です。ON 側にすると"1"、OFF 側にすると"0"として読み込みます。戻り値は、0～15 (2^4-1)の範囲です。



6.2.14 モータドライブ基板TypeS上のディップスイッチ値読み込み

```
554 :  /*********************************************************************/
555 :  /* モータドライブ基板TypeS上のディップスイッチ値読み込み             */
556 :  /* 戻り値 スイッチ値 0～15                                            */
557 :  /*********************************************************************/
558 :  unsigned char dipsw_get2( void )
559 :  {
560 :      unsigned char sw;
561 :
562 :      sw  = ~P2DR;                        /* ドライブ基板TypeSのSW読み込み*/
563 :
564 :      return  sw;
565 :  }
```

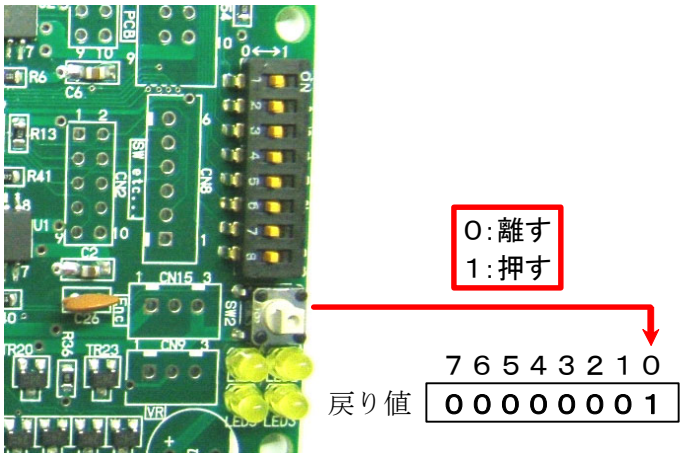
モータドライブ基板 TypeS 上のディップスイッチの値を読み込む関数です。



6.2.15 モータドライブ基板TypeS上のプッシュスイッチ値読み込み

```
567 :  /************************************************************************/
568 :  /* モータドライブ基板TypeS上のプッシュスイッチ値読み込み                 */
569 :  /* 戻り値 スイッチ値 0:OFF 1:ON                                          */
570 :  /************************************************************************/
571 :  unsigned char pushsw_get( void )
572 :  {
573 :      unsigned char sw;
574 :
575 :      sw = ~P8DR & 0x10;                  /* プッシュスイッチ読み込み      */
576 :      sw = !!sw;
577 :
578 :      return  sw;
579 :  }
```

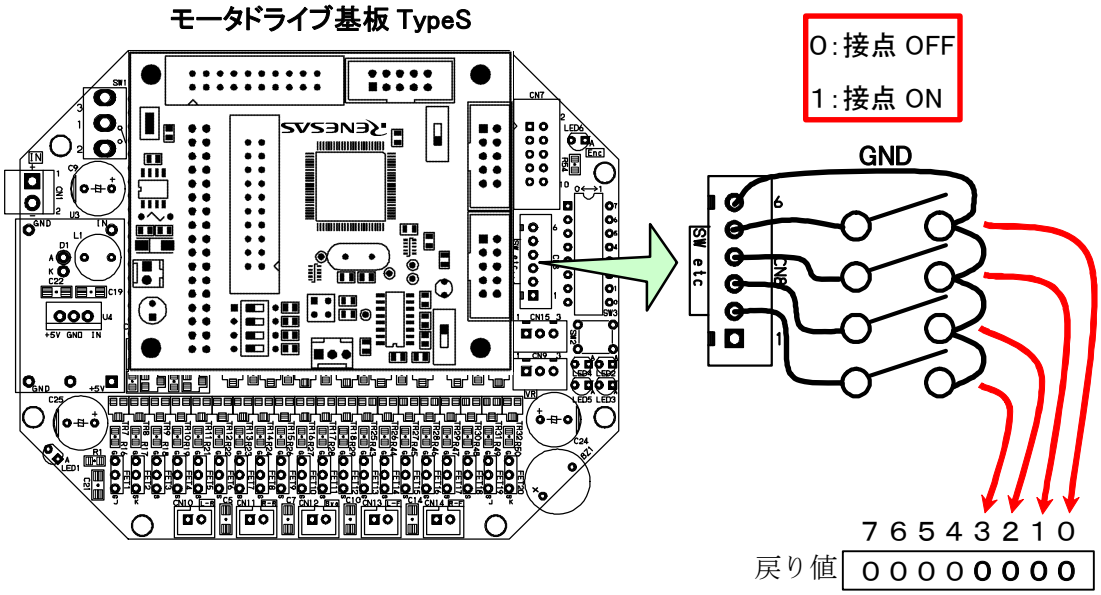
モータドライブ基板 TypeS 上のプッシュスイッチの値を読み込む関数です。



6.2.16 モータドライブ基板TypeSのCN8 の状態読み込み

```
581 :  /*****************************************************************************/
582 :  /* モータドライブ基板TypeSのCN8の状態読み込み                                */
583 :  /* 戻り値 0～15                                                              */
584 :  /*****************************************************************************/
585 :  unsigned char cn8_get( void )
586 :  {
587 :      unsigned char data;
588 :
589 :      data = P7DR >> 4;
590 :
591 :      return  data;
592 :  }
```

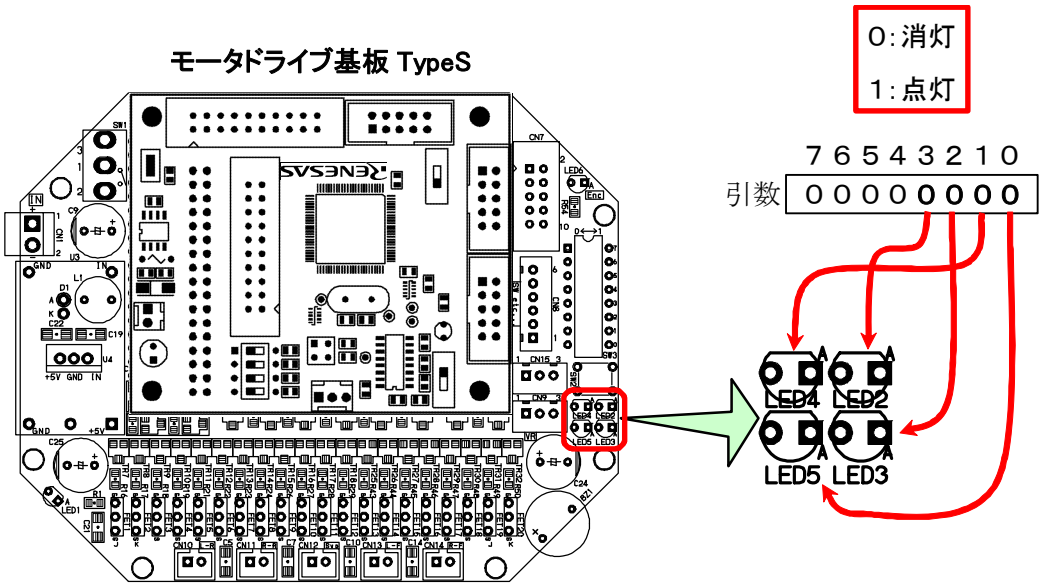
モータドライブ基板 TypeS の CN8 の状態を読み込む関数です。



6.2.17 モータドライブ基板TypeSのLED制御

```
594 :  /************************************************************************/
595 :  /* モータドライブ基板TypeSのLED制御                                     */
596 :  /* 引数  4個のLED制御 0:OFF 1:ON                                        */
597 :  /************************************************************************/
598 :  void led_out( unsigned char led )
599 :  {
600 :      unsigned char data;
601 :
602 :      led <<= 4;
603 :      data = P1DR & 0x0f;
604 :      P1DR = data | led;
605 :  }
```

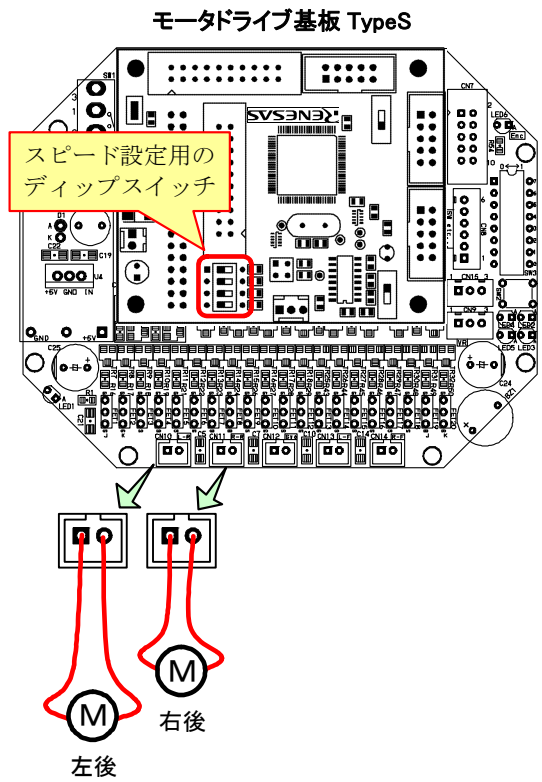
モータドライブ基板 TypeS の LED を制御する関数です。



6.2.18 後輪の速度制御

```
607 :  /************************************************************************/
608 :  /* 後輪の速度制御                                                       */
609 :  /* 引数    左モータ:-100～100 , 右モータ:-100～100                      */
610 :  /*         0で停止、100で正転100%、-100で逆転100%                       */
611 :  /************************************************************************/
612 :  void speed_r( int accele_l, int accele_r )
613 :  {
614 :      unsigned int    sw_data;
615 :      unsigned int    work;
616 :
617 :      sw_data  = dipsw_get() + 5;         /* ディップスイッチ読み込み     */
618 :      sw_data *= 5;                       /* 5～20 → 25～100に変換        */
619 :
620 :      saveData[ 9] = accele_l * sw_data / 100;    /* ログ保存             */
621 :      saveData[10] = accele_r * sw_data / 100;    /* ログ保存             */
622 :
623 :      /* 左モータ */
624 :      if( accele_l > 0 ) {
625 :          PBDR &= 0xbf;
626 :      } else if( accele_l < 0 ) {
627 :          PBDR |= 0x40;
628 :          accele_l = -accele_l;
629 :      }
630 :      work   = MOTOR_CYCLE / 2 - MOTOR_OFFSET - 2;
631 :      work   = (long)work * accele_l * sw_data / 10000;
632 :      ITU4_BRA = MOTOR_CYCLE / 2 - 2 - work;
633 :
634 :      /* 右モータ */
635 :      if( accele_r > 0 ) {
636 :          PBDR &= 0x7f;
637 :      } else if( accele_r < 0 ) {
638 :          PBDR |= 0x80;
639 :          accele_r = -accele_r;
640 :      }
641 :      work   = MOTOR_CYCLE / 2 - MOTOR_OFFSET - 2;
642 :      work   = (long)work * accele_r * sw_data / 10000;
643 :      ITU3_BRB = MOTOR_CYCLE / 2 - 2 - work;
644 :  }
```

モータドライブ基板 TypeS の後輪モータ 2 個を制御する関数です。



使い方は、

speed_r (左後モータのPWM, 右後モータのPWM) ;

です。PWM 値は、

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

を設定することができます。

モータへの出力は、speed_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×(CPU 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。

```
speed_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**

6.2.19 後輪の速度制御 2 ディップスイッチには関係しないspeed関数

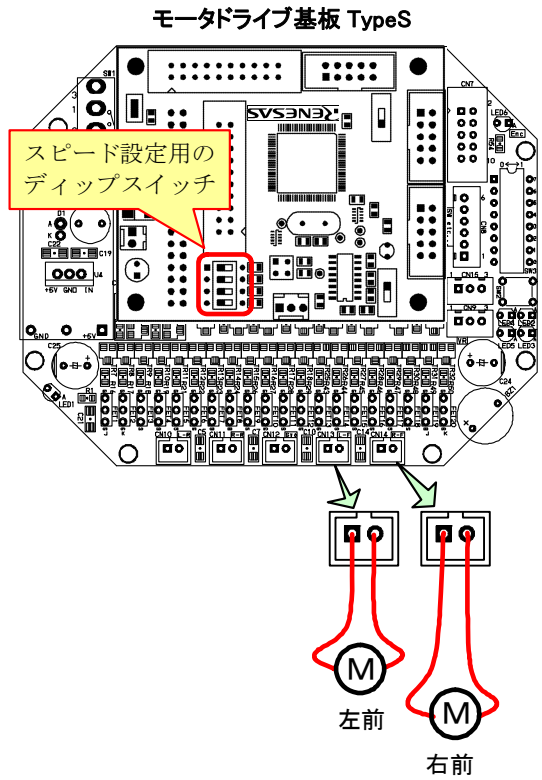
```
646 :  /*********************************************************************/
647 :  /* 後輪の速度制御2 ディップスイッチには関係しないspeed関数            */
648 :  /* 引数    左モータ:-100～100 , 右モータ:-100～100                    */
649 :  /*         0で停止、100で正転100%、-100で逆転100%                     */
650 :  /*********************************************************************/
651 :  void speed2_r( int accele_l, int accele_r )
652 :  {
653 :      unsigned int    sw_data;
654 :      unsigned int    work;
655 :
656 :      saveData[ 9] = accele_l;            /* ログ保存                  */
657 :      saveData[10] = accele_r;            /* ログ保存                  */
658 :
659 :      /* 左モータ */
660 :      if( accele_l > 0 ) {
661 :          PBDR &= 0xbf;
662 :      } else if( accele_l < 0 ) {
663 :          PBDR |= 0x40;
664 :          accele_l = -accele_l;
665 :      }
666 :      work   = MOTOR_CYCLE / 2 - MOTOR_OFFSET - 2;
667 :      work   = (long)work * accele_l / 100;
668 :      ITU4_BRA = MOTOR_CYCLE / 2 - 2 - work;
669 :
670 :      /* 右モータ */
671 :      if( accele_r > 0 ) {
672 :          PBDR &= 0x7f;
673 :      } else if( accele_r < 0 ) {
674 :          PBDR |= 0x80;
675 :          accele_r = -accele_r;
676 :      }
677 :      work   = MOTOR_CYCLE / 2 - MOTOR_OFFSET - 2;
678 :      work   = (long)work * accele_r / 100;
679 :      ITU3_BRB = MOTOR_CYCLE / 2 - 2 - work;
680 :  }
```

speed_r 関数は、ディップスイッチの割合でモータに出力する割合をさらに落としましたが、speed2_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。

6.2.20 前輪の速度制御

```
682 :  /*********************************************************************/
683 :  /* 前輪の速度制御                                                    */
684 :  /* 引数    左モータ:-100～100 , 右モータ:-100～100                   */
685 :  /*         0で停止、100で正転100%、-100で逆転100%                    */
686 :  /*********************************************************************/
687 :  void speed_f( int accele_l, int accele_r )
688 :  {
689 :      unsigned char   sw_data;
690 :      unsigned long   speed_max;
691 :
692 :      sw_data  = dipsw_get() + 5;          /* ディップスイッチ読み込み     */
693 :      speed_max = (unsigned long)(MOTOR_CYCLE-1) * sw_data / 20;
694 :
695 :      saveData[7] = accele_l * sw_data / 20;  /* ログ保存                  */
696 :      saveData[8] = accele_r * sw_data / 20;  /* ログ保存                  */
697 :
698 :      /* 左前モータ */
699 :      if( accele_l > 0 ) {
700 :          PADR &= 0xfd;
701 :      } else if( accele_l < 0 ) {
702 :          PADR |= 0x02;
703 :          accele_l = -accele_l;
704 :      }
705 :      /* GRBがCNTより20以上小さい値かどうかのチェック */
706 :      if( ITU0_GRB > 20 ) {
707 :          /* GRBが20以上なら  単純に比較 */
708 :          while( (ITU0_CNT >= ITU0_GRB-20) && (ITU0_CNT <= ITU0_GRB) );
709 :      } else {
710 :          /* GRBが20以下なら  上限値からの値も参照する */
711 :          while( (ITU0_CNT >= ITU0_GRA-20) || (ITU0_CNT <= ITU0_GRB) );
712 :      }
713 :      ITU0_GRB = speed_max * accele_l / 100;
714 :
715 :      /* 右前モータ */
716 :      if( accele_r > 0 ) {
717 :          PADR &= 0xbf;
718 :      } else if( accele_r < 0 ) {
719 :          PADR |= 0x40;
720 :          accele_r = -accele_r;
721 :      }
722 :      /* GRBがCNTより20以上小さい値かどうかのチェック */
723 :      if( ITU1_GRB > 20 ) {
724 :          /* GRBが20以上なら  単純に比較 */
725 :          while( (ITU1_CNT >= ITU1_GRB-20) && (ITU1_CNT <= ITU1_GRB) );
726 :      } else {
727 :          /* GRBが20以下なら  上限値からの値も参照する */
728 :          while( (ITU1_CNT >= ITU1_GRA-20) || (ITU1_CNT <= ITU1_GRB) );
729 :      }
730 :      ITU1_GRB = speed_max * accele_r / 100;
731 :  }
```

モータドライブ基板 TypeS の前輪モータ 2 個を制御する関数です。



使い方は、

speed_f（ 左前モータのPWM, 右前モータのPWM ）;

です。PWM 値は、

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

を設定することができます。

モータへの出力は、speed_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×(CPU 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 12 で下記プログラムを実行したとします。

```
speed_f( 50, 100 );
```

左前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×(12＋5)÷20
＝42.5
≒**42** （小数点以下は切り捨てです）

右前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×(12＋5)÷20
＝**85**

6.2.21 前輪の速度制御 2 ディップスイッチには関係しないspeed関数

```
733 :  /*********************************************************************/
734 :  /* 前輪の速度制御2 ディップスイッチには関係しないspeed関数            */
735 :  /* 引数    左モータ:-100～100 , 右モータ:-100～100                    */
736 :  /*         0で停止、100で正転100%、-100で逆転100%                     */
737 :  /*********************************************************************/
738 :  void speed2_f( int accele_l, int accele_r )
739 :  {
740 :      unsigned long   speed_max;
741 :
742 :      saveData[7] = accele_l;             /* ログ保存                  */
743 :      saveData[8] = accele_r;             /* ログ保存                  */
744 :
745 :      speed_max = MOTOR_CYCLE - 1;
746 :
747 :      /* 左前モータ */
748 :      if( accele_l > 0 ) {
749 :          PADR &= 0xfd;
750 :      } else if( accele_l < 0 ) {
751 :          PADR |= 0x02;
752 :          accele_l = -accele_l;
753 :      }
754 :      /* GRBがCNTより20以上小さい値かどうかのチェック */
755 :      if( ITU0_GRB > 20 ) {
756 :          /* GRBが20以上なら  単純に比較 */
757 :          while( (ITU0_CNT >= ITU0_GRB-20) && (ITU0_CNT <= ITU0_GRB) );
758 :      } else {
759 :          /* GRBが20以下なら  上限値からの値も参照する */
760 :          while( (ITU0_CNT >= ITU0_GRA-20) || (ITU0_CNT <= ITU0_GRB) );
761 :      }
762 :      ITU0_GRB = speed_max * accele_l / 100;
763 :
764 :      /* 右前モータ */
765 :      if( accele_r > 0 ) {
766 :          PADR &= 0xbf;
767 :      } else if( accele_r < 0 ) {
768 :          PADR |= 0x40;
769 :          accele_r = -accele_r;
770 :      }
771 :      /* GRBがCNTより20以上小さい値かどうかのチェック */
772 :      if( ITU1_GRB > 20 ) {
773 :          /* GRBが20以上なら  単純に比較 */
774 :          while( (ITU1_CNT >= ITU1_GRB-20) && (ITU1_CNT <= ITU1_GRB) );
775 :      } else {
776 :          /* GRBが20以下なら  上限値からの値も参照する */
777 :          while( (ITU1_CNT >= ITU1_GRA-20) || (ITU1_CNT <= ITU1_GRB) );
778 :      }
779 :      ITU1_GRB = speed_max * accele_r / 100;
780 :  }
```

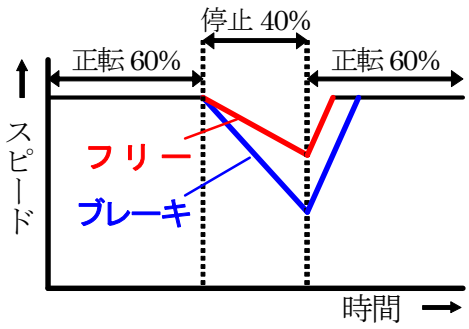
speed_f 関数は、ディップスイッチの割合でモータに出力する割合をさらに落としましたが、speed2_f 関数は、プログラムの引数どおりの割合をモータに出力します。

6.2.22 後モータ停止動作(ブレーキ、フリー)設定

```
782 :  /*********************************************************************/
783 :  /* 後モータ停止動作 (ブレーキ、フリー)                                */
784 :  /* 引数    左モータ:FREE or BRAKE , 右モータ:FREE or BRAKE            */
785 :  /*********************************************************************/
786 :  void motor_mode_r( int mode_l, int mode_r )
787 :  {
788 :      if( mode_l ) {
789 :          P1DR |= 0x01;
790 :      } else {
791 :          P1DR &= 0xfe;
792 :      }
793 :      if( mode_r ) {
794 :          P1DR |= 0x02;
795 :      } else {
796 :          P1DR &= 0xfd;
797 :      }
798 :  }
```

後モータを回すとき、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。例えば、60%でモータを正転させるとき、60%が正転、残りの 40%が停止です。この 40%の停止状態をブレーキにするかフリーにするかを選択します。100%で回すときは、停止状態が無いため、motor_mode_r 関数の設定は無効になります。

下図に停止時をブレーキにしたときとフリーにしたときのスピードイメージを示します。あくまでイメージです。実際はタイヤの抵抗など条件により変わりますので、各自検証してください。



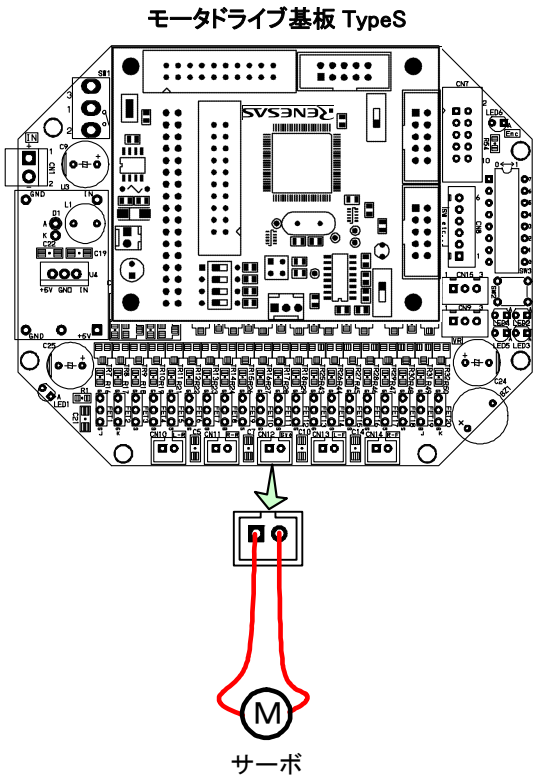
6.2.23 前モータ停止動作(ブレーキ、フリー)設定

```
800 :  /*********************************************************************/
801 :  /* 前モータ停止動作 (ブレーキ、フリー)                                */
802 :  /* 引数    左モータ:FREE or BRAKE , 右モータ:FREE or BRAKE            */
803 :  /*********************************************************************/
804 :  void motor_mode_f( int mode_l, int mode_r )
805 :  {
806 :      if( mode_l ) {
807 :          P1DR |= 0x04;
808 :      } else {
809 :          P1DR &= 0xfb;
810 :      }
811 :      if( mode_r ) {
812 :          P1DR |= 0x08;
813 :      } else {
814 :          P1DR &= 0xf7;
815 :      }
816 :  }
```

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。

6.2.24 サーボモータの制御

```
818 :  /*****************************************************************************/
819 :  /* サーボモータ制御                                                          */
820 :  /* 引数    サーボモータPWM：-100～100                                        */
821 :  /*****************************************************************************/
822 :  void servoPwmOut( int pwm )
823 :  {
824 :      unsigned int    work;
825 :
826 :      saveData[6] = pwm;                  /* ログ保存                       */
827 :
828 :      if( pwm > 0 ) {
829 :          PADR &= 0x7f;
830 :      } else if( pwm < 0 ) {
831 :          PADR |= 0x80;
832 :          pwm = -pwm;
833 :      }
834 :      work    = MOTOR_CYCLE / 2 - MOTOR_OFFSET - 2;
835 :      work    = (long)work * pwm / 100;
836 :      ITU4_BRB = MOTOR_CYCLE / 2 - 2 - work;
837 :  }
```



モータドライブ基板 TypeS のサーボモータを制御する関数です。

使い方は、

servoPwmOut（ サーボモータのPWM ）;

です。PWM 値は、

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

を設定することができます。

今回のサンプルプログラムは、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るよう配線します。逆の場合は、サーボモータの線を入れ変えてください。

6.2.25 サーボモータ停止動作(ブレーキ、フリー)設定

```
839 :  /*************************************************************************/
840 :  /* サーボモータ停止動作 (ブレーキ、フリー)                                 */
841 :  /* 引数   FREE or BRAKE                                                  */
842 :  /*************************************************************************/
843 :  void motor_mode_s( int mode )
844 :  {
845 :      if( mode ) {
846 :          P3DR |= 0x80;
847 :      } else {
848 :          P3DR &= 0x7f;
849 :      }
850 :  }
```

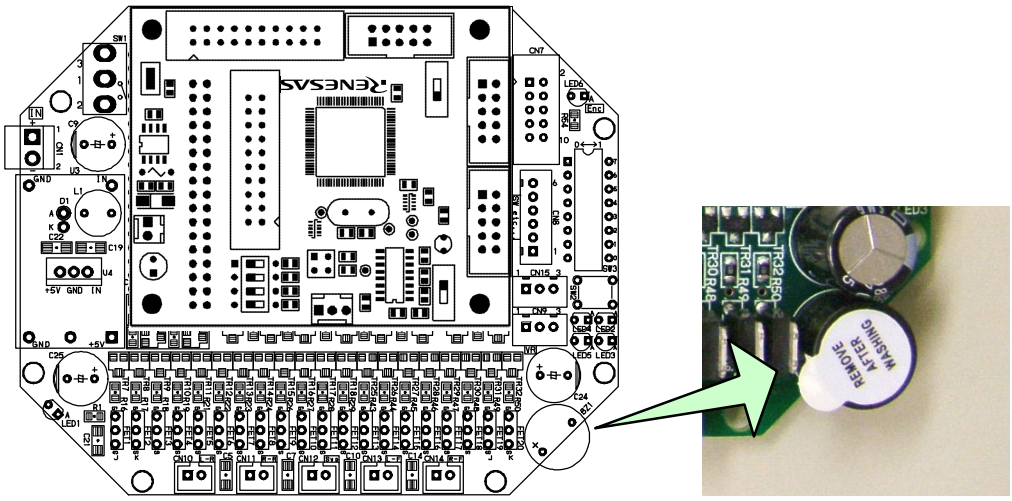
サーボモータを回すとき、停止の状態をブレーキにするかフリーにするか選択します。他は motor_mode_l 関数や motor_mode_r 関数と同じです。

6.2.26 ブザーの制御

```
852 :  /*************************************************************************/
853 :  /* ブザー制御                                                            */
854 :  /* 引数  0:ブザーOFF 1:ブザーON                                          */
855 :  /*************************************************************************/
856 :  void beep_out( int flag )
857 :  {
858 :      if( flag ) {
859 :          P6DR |= 0x10;
860 :      } else {
861 :          P6DR &= 0xef;
862 :      }
863 :  }
```

モータドライブ基板 TypeS のブザーを制御する関数です。

モータドライブ基板 TypeS



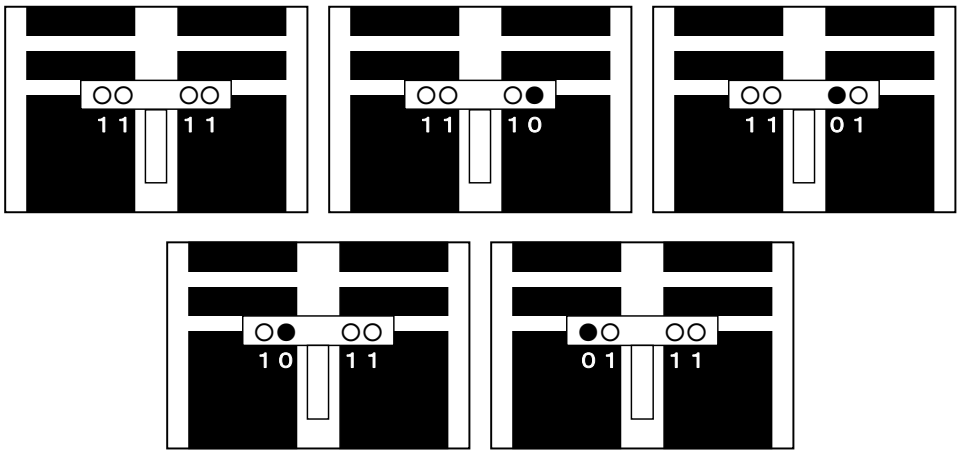
プログラムでの使用例を、下記に示します。

```
beep_out( 0 );  // ブザーOFF
beep_out( 1 );  // ブザーON
```

6.2.27 クロスラインの検出処理

```
869 :  /************************************************************************/
870 :  /* クロスライン検出処理                                                 */
871 :  /* 戻り値 0:クロスラインなし 1:あり                                     */
872 :  /************************************************************************/
873 :  int check_crossline( void )
874 :  {
875 :      unsigned char b;
876 :      int ret = 0;
877 :
878 :      b = sensor_inp();
879 :      if( b==0x0f || b==0x0e || b==0x0d || b==0x0b || b==0x07 ) {
880 :          ret = 1;
881 :      }
882 :      return ret;
883 :  }
```

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 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。



6.2.28 サーボ角度の取得

```
885 :  /************************************************************************/
886 :  /* サーボ角度取得                                                       */
887 :  /* 引数    なし                                                         */
888 :  /* 戻り値 入れ替え後の値                                                */
889 :  /************************************************************************/
890 :  int getServoAngle( void )
891 :  {
892 :      return( (AD_DRB>>6) - iAngle0 );
893 :  }
```

サーボの角度は、AN1(P71)端子に接続されているボリュームの値で分かります。iAngle0 は、0 度のときの A/D 変換値です。例えば、iAngle0 変数が 456、角度が 0 度で A/D 変換値が 456 なら戻り値は下記のようになります。

戻り値　＝A/D 変換値－0 度のときの A/D 変換値
　　＝456－456
　　＝0

今回のマイコンカーは左右 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。テスタでボリューム値を計った結果、

左いっぱい…3.26V　中心…2.23V　右いっぱい…1.21V

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

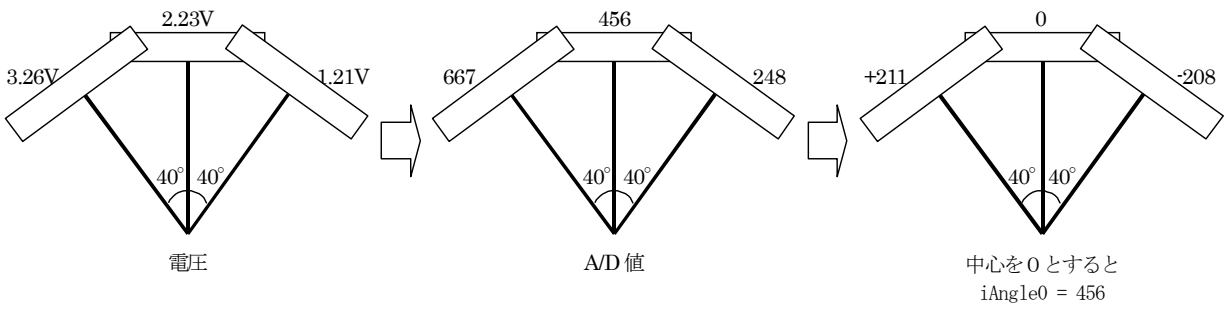
左いっぱい…3.26/5*1023=667　中心…2.23/5*1023=456　右いっぱい…1.21/5*1023=248

となります(下中図)。

892 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…+211　中心…0　右いっぱい…-208

となります(下右図)。

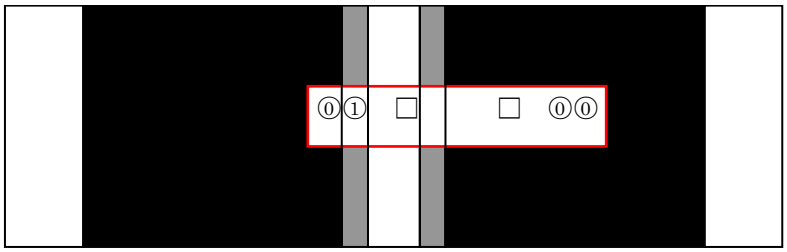


6.2.29 アナログセンサ値の取得

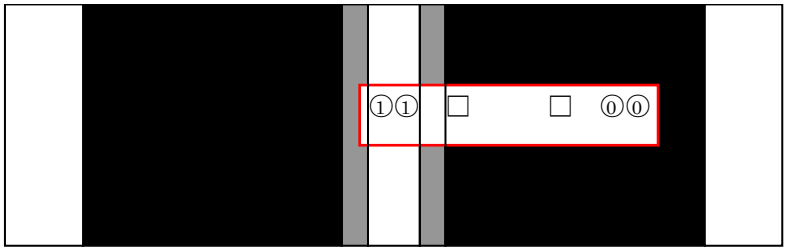
```
895 :  /*********************************************************************/
896 :  /* アナログセンサ値取得                                              */
897 :  /* 引数    なし                                                      */
898 :  /* 戻り値 センサ値                                                   */
899 :  /*********************************************************************/
900 :  int getAnalogSensor( void )
901 :  {
902 :      int     ret;
903 :
904 :      ret = (AD_DRD>>6) - (AD_DRC>>6);    /* アナログセンサ情報取得       */
905 :
906 :      if( !crank_mode ) {
907 :          /* クランクモードでなければ補正処理 */
908 :          switch( iSensorPattern ) {
909 :          case 0:
910 :              if( sensor_inp() == 0x04 ) {
911 :                  ret = -650;
912 :                  break;
913 :              }
914 :              if( sensor_inp() == 0x02 ) {
915 :                  ret = 650;
916 :                  break;
917 :              }
918 :              if( sensor_inp() == 0x0c ) {
919 :                  ret = -700;
920 :                  iSensorPattern = 1;
921 :                  break;
922 :              }
923 :              if( sensor_inp() == 0x03 ) {
924 :                  ret = 700;
925 :                  iSensorPattern = 2;
926 :                  break;
927 :              }
928 :              break;
929 :
930 :          case 1:
931 :              /* センサ右寄り */
932 :              ret = -700;
933 :              if( sensor_inp() == 0x04 ) {
934 :                  iSensorPattern = 0;
935 :              }
936 :              break;
937 :
938 :          case 2:
939 :              /* センサ左寄り */
940 :              ret = 700;
941 :              if( sensor_inp() == 0x02 ) {
942 :                  iSensorPattern = 0;
943 :              }
944 :              break;
945 :          }
946 :      }
947 :
948 :      return ret;
949 :  }
```

2 個あるアナログセンサの値の差分を計算して、コース中心からのずれを検出します。904 行で「左アナログセンサー右アナログセンサ」の計算をしています。それ以降の行は、急カーブのときにステアリングが反応しきれず

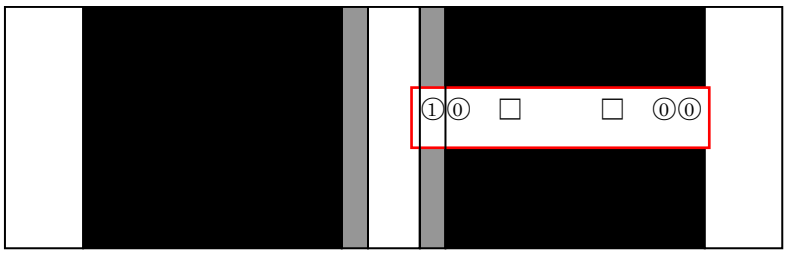
にセンタラインを大きくはずれてしまった場合、デジタルセンサを使用して補正させる処理を行っています。その処理を 908 行から 945 行まで行っています。



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と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。

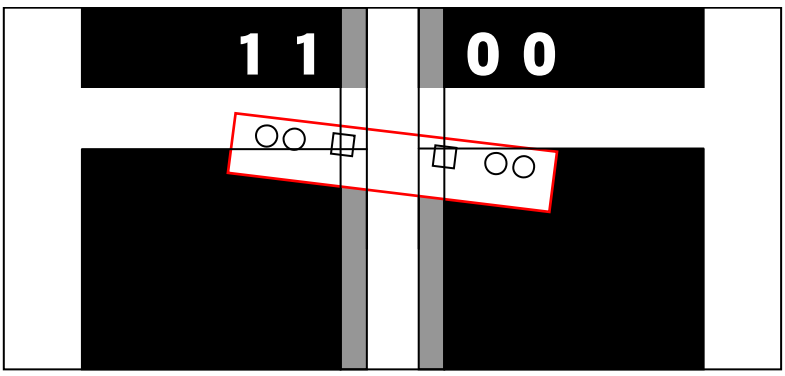
このように、アナログセンサだけでは追従しきれない場合を想定して、デジタルセンサを使いアナログセンサの値を補正しています。

逆のずれも、デジタルセンサの状態と値が変わるだけで考え方は同じです。

906 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 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"になったらすぐに crank_mode を 1 として、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。



6.2.30 サーボモータ制御

```
951 :  /************************************************************************/
952 :  /* モジュール名 servoControl                                            */
953 :  /* 処理概要     サーボモータ制御                                        */
954 :  /* 引数         なし                                                    */
955 :  /* 戻り値       グローバル変数 iServoPwm に代入                         */
956 :  /************************************************************************/
957 :  void servoControl( void )
958 :  {
959 :      int     i, iRet, iP, iD;
960 :      int     kp, kd;
961 :
962 :      i = getAnalogSensor();              /* センサ値取得                 */
963 :      kp = dipsw_get2() & 0x0f;           /* 調整できたら P,D 値は固定値に  */
964 :      kd = (dipsw_get2() >> 4) * 5;       /* してください                 */
965 :
966 :      /* サーボモータ用 PWM 値計算 */
967 :      iP = kp * i;                        /* 比例                         */
968 :      iD = kd * (iSensorBefore - i );     /* 微分(目安は P の 5～10 倍)      */
969 :      iRet = iP - iD;
970 :      iRet /= 64;
971 :
972 :      /* PWM の上限の設定 */
973 :      if( iRet >  50 ) iRet =  50;        /* マイコンカーが安定したら     */
974 :      if( iRet < -50 ) iRet = -50;        /* 上限を 90 くらいにしてください */
975 :      iServoPwm = iRet;
976 :
977 :      iSensorBefore = i;                  /* 次回はこの値が 1ms 前の値となる*/
978 :  }
```

この関数で、現在のセンサのずれを検出して、サーボモータの PWM 値を計算する、サーボ制御の要(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：Proportinal（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、きめ細かくサーボモータの 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。

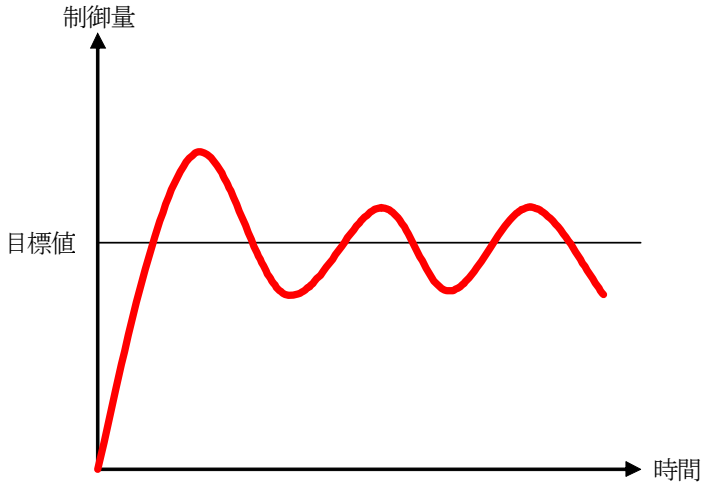
今回、サーボモータの制御は比例制御と微分制御を行います。PD 制御と呼びます。

(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 を計算する式は下記のようになります。

制御量 P＝kp×p
kp ＝ 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、ずれが大きいほどサーボモータの 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しきれず、目標値をいったりきたりと振動してしまいます。

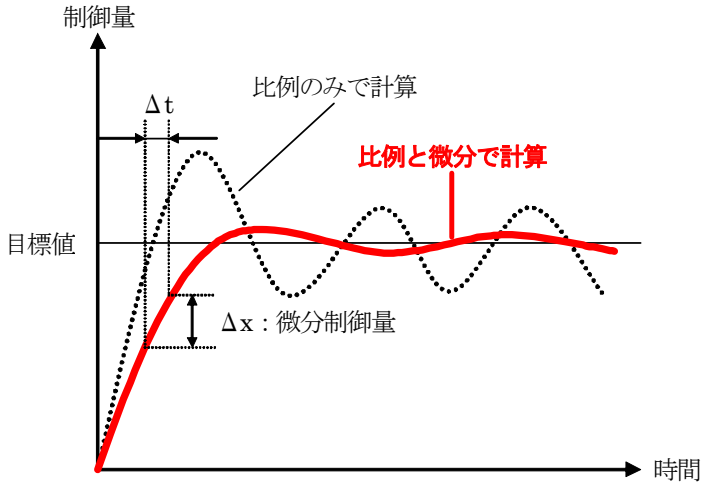


(3) D(微分)制御を加える

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 を計算する式は下記のようになります。

制御量 D＝kd×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

サーボに加える制御量を計算する式は下記のようになります。

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、サーボモータに大きい 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、せっかくの PD 制御も上限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、
P制御の定数＝ モータドライブ基板 TypeS のディップスイッチ下位 4 ビット
D制御の定数＝ モータドライブ基板 TypeS のディップスイッチ上位 4 ビット
最終定数 ＝ 1/64
としました。最初はディップスイッチの値をすべて 0 にしておきます。P 制御の定数、D 制御の定数は、モータ、ギヤ、電圧により違ってきますので個々のマイコンカーに合わせてカット&トライで調整する必要があります。調整するときは、1 つずつ値を増やしサーボがコースを滑らかにトレースするように調整してください。

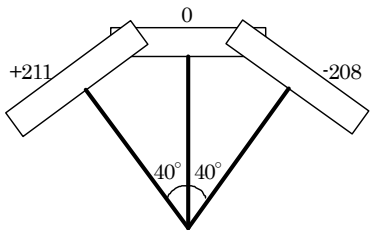
6.2.31 内輪PWM値計算

```
980 :  /*********************************************************************/
981 :  /* 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      */
982 :  /* 引数　　外輪PWM                                                    */
983 :  /* 戻り値 内輪PWM                                                     */
984 :  /*********************************************************************/
985 :  int diff( int pwm )
986 :  {
987 :      int i, ret;
988 :
989 :      i  = getServoAngle() / 5;          /* 1度あたりの増分で割る          */
990 :      if( i <  0 ) i = -i;
991 :      if( i > 45 ) i = 45;
992 :      ret = revolution_difference[i] * pwm / 100;
993 :
994 :      return ret;
995 :  }
```

各自のマイコンカーに
合わせて調整してください

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う関数です。

989 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 40 度のとき A/D 値は 211、右 40 度のとき A/D 値は-208 でした。A/D 値や角度の測り方で若干の誤差がありますので、ここではそれぞれ±210 とします。A/D 値 210 のとき、左 40 度なので、1 度あたりの A/D 値は下記のようになります。

1 度あたりの A/D 値＝左 40 度のときの A/D 値÷左 40 度
＝210÷40
＝5.25 ≒ 5

値は四捨五入して、整数にします。よって、A/D 値 5 が約 1 度となります。

990 行は、負の数を正の数に変換してます。
991 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。
992 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 度のとき、添え字部分に 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 ]
    = 65
```

外輪 100%のとき、内輪は戻り値である 65%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 = 65 * 外輪の PWM 値 / 100
    = 65 * 60 / 100
    = 39
```

サーボ角度が 25 度のとき、下記プログラムを実行すると kakudo 変数には 39 が代入されます。

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

6.2.32 main関数－初期化

```
111 :  /************************************************************************/
112 :  /* メインプログラム                                                     */
113 :  /************************************************************************/
114 :  void main( void )
115 :  {
116 :      int             i, j;
117 :      unsigned int    u;
118 :      char            c;
119 :
120 :      /* マイコン機能の初期化 */
121 :      init();                             /* 初期化                       */
122 :      initI2CEeprom( &P6DDR, &P6DR, 0x90, 6, 5);  /* EEP-ROM初期設定      */
123 :      initBeepS();                        /* ブザー関連処理               */
124 :      init_sci1( 0x00, 79 );              /* SCI1初期化                   */
125 :      set_ccr( 0x00 );                    /* 全体割り込み許可             */
126 :
127 :      /* マイコンカーの状態初期化 */
128 :      motor_mode_f( BRAKE, BRAKE );
129 :      motor_mode_r( BRAKE, BRAKE );
130 :      motor_mode_s( BRAKE );
131 :      speed_f( 0, 0 );
132 :      speed_r( 0, 0 );
133 :      servoPwmOut( 0 );
134 :      setBeepPatternS( 0x8000 );
135 :
136 :      /* スタート時、スイッチが押されていればデータ転送モード */
137 :      if( pushsw_get() ) {
138 :          pattern = 101;
139 :          cnt1 = 0;
140 :      }
```

main 関数では最初に H8/3048F-ONE の内蔵周辺機能の初期化、変数の初期化、割り込みの許可、モータを停止状態にします。
また、電源投入時、プッシュスイッチが押されていれば、データ転送モードであるパターン 101 へ移ります。

6.2.33 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、1 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
なし	電源投入時	・プッシュスイッチが押されていれば 101 へ
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	・1秒たったら 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
101	停止	・マイコンカーを停止状態にして 102 へ
102	プッシュスイッチが離されたかチェック	・プッシュスイッチが離されたら 103 へ
103	0.5s 待ち	・0.5s たったら 104 へ
104	プッシュスイッチが押されたかチェック	・プッシュスイッチが押されたら 105 へ
105	タイトル転送、転送準備	・パソコンへタイトル転送、転送準備が整ったら 106 へ
106	データ転送	・パソコンへデータ転送が完了したら 107 へ
107	転送終了	特になし(電源を切るまで待つ)
その他	―	・0 へ

6.2.34 パターン 0:スタート待ち

```
146 :       switch( pattern ) {
147 :       case 0:
148 :           /* プッシュスイッチ押下待ち */
149 :           servoPwmOut( 0 );
150 :           if( pushsw_get() ) {
151 :               setBeepPatternS( 0x8000 );
152 :               clearI2CEeprom();           /* 数秒かかる                    */
153 :               setBeepPatternS( 0xcc00 );
154 :               cnt1 = 0;
155 :               pattern = 1;
156 :               break;
157 :           }
158 :           i =  (cnt1/200) % 2 + 1;
159 :           if( startbar_get() ) {
160 :               i += ((cnt1/100 ) % 2 + 1) << 2;
161 :           }
162 :           led_out( i );                   /* LED 点滅処理                  */
163 :           break;
```

プッシュスイッチ押下待ちです。プッシュスイッチを押すまでの間、LED4 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)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。

プッシュスイッチを押すと、下記処理を実行します。

・ブザーを鳴らす(ボタンを押した確認)
・EEP-ROM の内容をクリア(約 5 秒かかります)
・ブザーを鳴らす(EEP-ROM をクリアした確認)
・パターン 1 へ移ります。

6.2.35 パターン 1:スタートバー開待ち

```
165 :       case 1:
166 :           /* スタートバー開待ち */
167 :           servoPwmOut( iServoPwm / 2 );
168 :           if( !startbar_get() ) {
169 :               iAngle0 = getServoAngle();  /* 0 度の位置記憶               */
170 :               led_out( 0x0 );
171 :               cnt1 = 0;
172 :               pattern = 11;
173 :               saveFlag = 1;               /* データ保存開始               */
174 :               break;
175 :           }
176 :           led_out( 1 << (cnt1/50) % 4 );
177 :           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。

161 行で早速サーボ制御を行っています。iServoPwm 変数がサーボモータに加える 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、センサ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。

6.2.36 パターン 11:通常トレース

```
179 :       case 11:
180 :           /* 通常トレース */
181 :           servoPwmOut( iServoPwm );
182 :           i = getServoAngle();
183 :           if( i > 170 ) {
184 :               speed_f( 0, 0 );
185 :               speed_r( 0, 0 );
186 :           } else if( i > 25 ) {
187 :               speed_f( diff(80), 80 );
188 :               speed_r( diff(80), 80 );
189 :           } else if( i < -170 ) {
190 :               speed_f( 0, 0 );
191 :               speed_r( 0, 0 );
192 :           } else if( i < -25 ) {
193 :               speed_f( 80, diff(80) );
194 :               speed_r( 80, diff(80) );
195 :           } else {
196 :               speed_f( 100, 100 );
197 :               speed_r( 100, 100 );
198 :           }
199 :           if( check_crossline() ) {       /* クロスラインチェック          */
200 :               cnt1 = 0;
201 :               crank_mode = 1;
202 :               pattern = 21;
203 :           }
204 :           break;
```

181 行でサーボ制御を行っています。 次に 182 行でハンドル角度を取得します。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÷5	左モータ PWM	右モータ PWM
171 以上	34 以上	0	0
26～170	5～34	diff(80)	80
-171 以下	-34 以下	0	0
-26～-170	-5～-34	80	diff(80)
それ以外 (-25～25)	-5～5	100	100

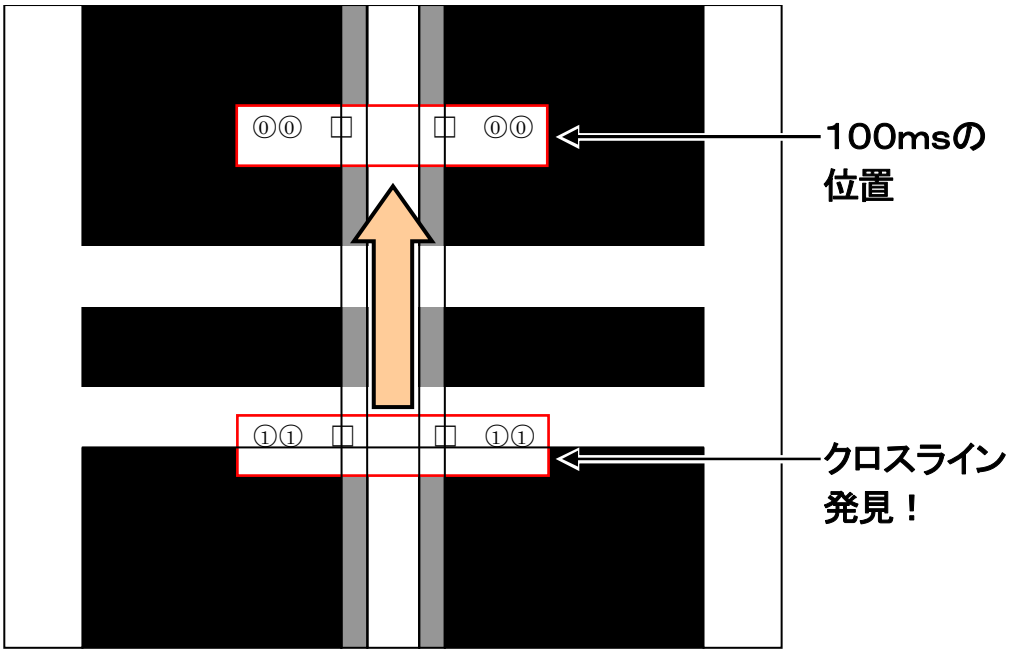
A/D 値が、±(26～170)なら、外輪を 80%として、内輪をステアリングの切れ角に応じて PWM 値を可変します。

最後に、199 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 へ移行します。

6.2.37 パターン 21:クロスライン検出処理

```
206 :       case 21:
207 :           /* クロスライン通過処理 */
208 :           servoPwmOut( iServoPwm );
209 :           led_out( 0x3 );
210 :           speed_f( 0, 0 );
211 :           speed_r( 0, 0 );
212 :           if( cnt1 >= 100 ) {
213 :               cnt1 = 0;
214 :               pattern = 22;
215 :           }
216 :           break;
```

ここではブレーキをかけて、サーボ制御を行います。100ms の間に 2 本目の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脱輪してしまいます。



6.2.38 パターン 22:クロスライン後のトレース、直角検出処理

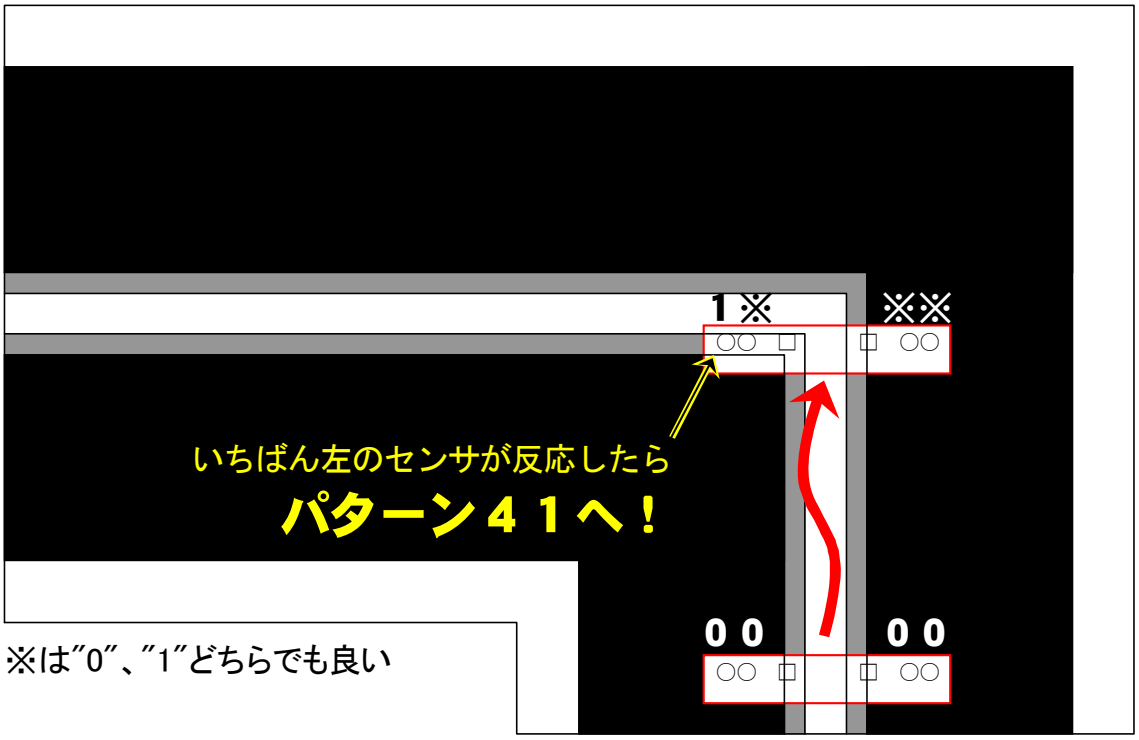
```
218 :      case 22:
219 :          /* クロスライン後のトレース、直角検出処理 */
220 :          servoPwmOut( iServoPwm );
221 :          if( iEncoder >= 11 ) {          /* エンコーダによりスピード制御 */
222 :              speed_f( 0, 0 );
223 :              speed_r( 0, 0 );
224 :          } else {
225 :              speed2_f( 70, 70 );
226 :              speed2_r( 70, 70 );
227 :          }
228 :
229 :          if( (sensor_inp()&0x01) == 0x01 ) { /* 右クランク？             */
230 :              led_out( 0x1 );
231 :              cnt1 = 0;
232 :              pattern = 31;
233 :              break;
234 :          }
235 :          if( (sensor_inp()&0x08) == 0x08 ) {  /* 左クランク？            */
236 :              led_out( 0x2 );
237 :              cnt1 = 0;
238 :              pattern = 41;
239 :              break;
240 :          }
241 :          break;
```

クロスライン通過後の処理を行います。

221～227 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。

229 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。同様に 235 行目で、いちばん左の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 へ移動します。


※※
※1
いちばん右のセンサが反応したら
パターン３１へ！
0 0
0 0
※は"0"、"1"どちらでも良い


1※
※※
いちばん左のセンサが反応したら
パターン４１へ！
0 0
0 0
※は"0"、"1"どちらでも良い

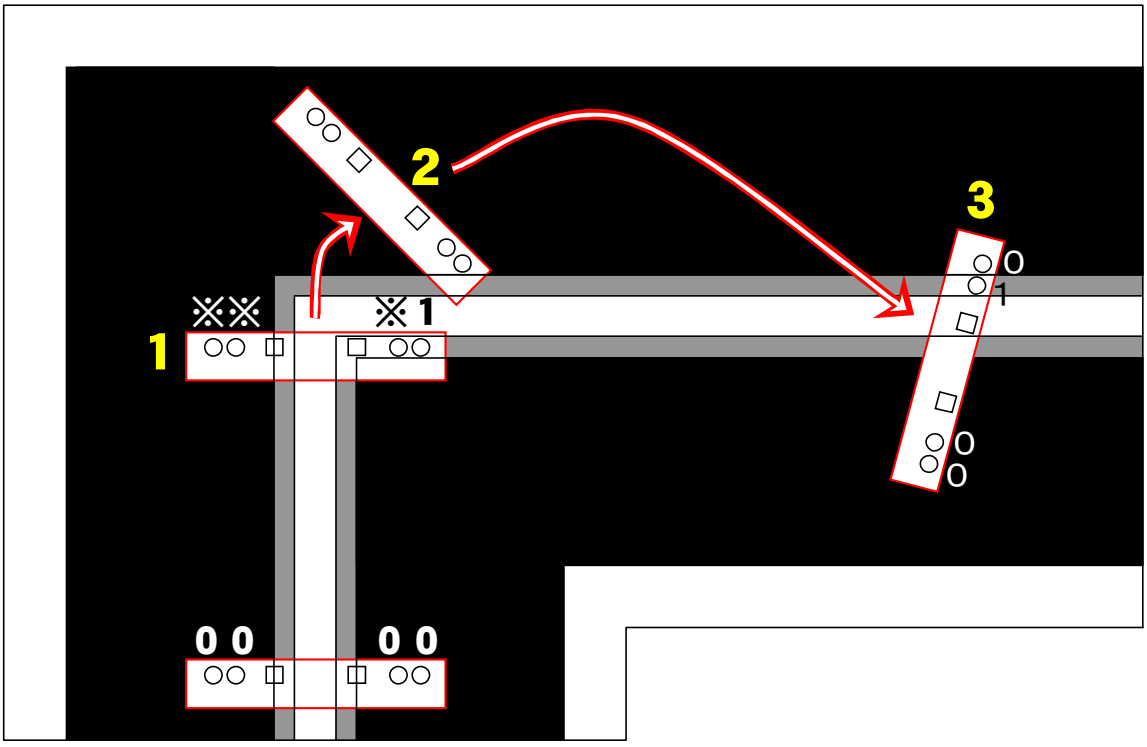
6.2.39 パターン 31:右クランク処理

```
243 :       case 31:
244 :           /* 右クランク処理 */
245 :           servoPwmOut( 50 );          /* 振りが弱いときは大きくする        */
246 :           speed_f( 60, 33 );          /* この部分は「角度計算(4WD 時).xls」*/
247 :           speed_r( 49, 22 );          /* で計算                            */
248 :           if( sensor_inp() == 0x04 ) {    /* 曲げ終わりチェック            */
249 :               cnt1 = 0;
250 :               iSensorPattern = 0;
251 :               crank_mode = 0;
252 :               pattern = 32;
253 :           }
254 :           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。

サーボモータの PWM はセンサ状態に関係なく右に 50%回転させています。サーボの動きが遅い場合はこの値を大きくしますが、曲げすぎて車体にステアリング部分がぶつかりロックしないようにしてください。サーボモータのトルクが大きすぎたりギヤが弱い場合、ギヤがかけたり車体が曲がったりすることがありますので気をつけます。

何処まで回し続けるかというのが 242 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サーボモータを 50%、左前モータを 60%、右前モータを 33%、左後モータを 49%、右後モータを 22%で回します。左右回転差は、ハンドル 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。

6.2.40 パターン 32:右クランク処理後、少し時間がたつまで待つ

```
256 :       case 32:
257 :           /* 少し時間が経つまで待つ */
258 :           servoPwmOut( iServoPwm );
259 :           speed2_r( 80, 80 );
260 :           speed2_f( 80, 80 );
261 :           if( cnt1 >= 100 ) {
262 :               led_out( 0x0 );
263 :               pattern = 11;
264 :           }
265 :           break;
```

右クランク処理終了後、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、ハンドルをかなり曲げています。この状態でパターン 11 に戻ると、ボリューム値が-170 以下なのでモータスピードが左右共に 0%になってしまいます。これを防ぐために 100ms 間、ハンドルの角度に関係なく PWM を 80%にして少し進ませます。

6.2.41 パターン 41:左クランク処理

```
267 :       case 41:
268 :           /* 左クランク処理 */
269 :           servoPwmOut( -50 );          /* 振りが弱いときは大きくする        */
270 :           speed_f( 33, 60 );           /* この部分は「角度計算(4WD 時).xls」*/
271 :           speed_r( 22, 49 );           /* で計算                            */
272 :           if( sensor_inp() == 0x02 ) {    /* 曲げ終わりチェック             */
273 :               cnt1 = 0;
274 :               iSensorPattern = 0;
275 :               crank_mode = 0;
276 :               pattern = 42;
277 :           }
278 :           break;
```

左クランクも同様です。サーボモータは左へ 50%で回転させ、駆動モータを 40 度ハンドルを切ったと仮定して PWM を設定します。この状態をデジタルセンサの状態が"0010"になるまで繰り返します。"0010"になるとパターン 42 へ移ります。

6.2.42 パターン 42:左クランク処理後、少し時間がたつまで待つ

```
280 :       case 42:
281 :           /* 少し時間が経つまで待つ */
282 :           servoPwmOut( iServoPwm );
283 :           speed2_f( 80, 80 );
284 :           speed2_r( 80, 80 );
285 :           if( cnt1 >= 100 ) {
286 :               led_out( 0x0 );
287 :               pattern = 11;
288 :           }
289 :           break;
```

左クランク処理終了後、100ms 間は駆動モータを 80%にします。これはパターン 42 に移ってきたときは、ハンド

ルをかなり曲げています。この状態でパターン11に戻ると、ボリューム値が170以下なのでモータスピードが左右共に0%になってしまいます。これを防ぐために100ms間、ハンドルの角度に関係なくPWMを80%にして少し進ませます。

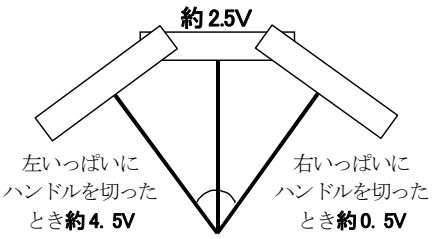
6.3 ハードウェアの調整のポイント

6.3.1 サーボ用モータの回転方向

サーボ用モータを接続するコネクタの2ピン側のモータ端子に＋、1ピン側のモータ端子にーの電圧を加えたとき、進行方向向かって右側にステアリングが回転するようにします。逆の場合、モータの線を左右入れ替えます。

6.3.2 ボリュームの調整

中心はほぼ2.5Vになるようにボリュームの向きとステアリングの角度を合わせます。
左右それぞれいっぱいまでハンドルを切ったとき、2.5±2Vになるようにするのが理想です。A/D値をいっぱいまで使用した方が、ちょっとのハンドルの曲げでも数値が変化するので精度が良くなります。一応下限の0Vと上限の5Vまで0.5Vの余裕を持たせて、切りすぎたときに変化しないということがないようにしています。下図に、その様子を示します。



左にハンドルを切ったときに電圧が高くなるように、右は電圧が低くなるように配線します。逆の場合は、ボリュームの1ピンと3ピンの線を逆にします。
ギヤの関係で、電圧の可変範囲が小さくなっても構いませんが精度が悪くなります。今回の説明用マイコンカーはギヤ比の関係で約±1.0Vしか電圧が変化しません。これは悪い例です。

6.3.3 角度を測っておく

最大まで曲げたときの、角度を測っておきます。また、いっしょにそのときのA/D値も計算しておきます。
説明用マイコンカーの場合、テスタでボリュームの電圧を計った結果、下記のようになりました。

左いっぱい…3.26V　中心…2.23V　右いっぱい…1.21V

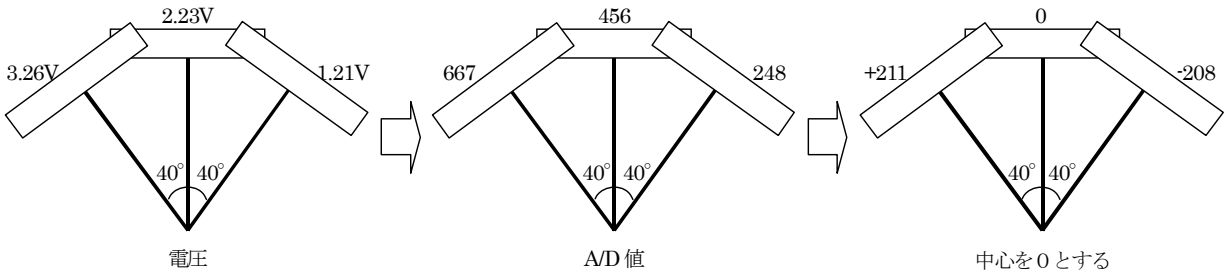
5.00Vが1023なので、それぞれの電圧をA/D値に変換すると、下記のようになります。

左いっぱい…3.26/5*1023=667　中心…2.23/5*1023=456　右いっぱい…1.21/5*1023=248

中心を0とすると、下記のようになります。

左いっぱい…先ほどの計算結果－中心値＝667-456＝ 211 右いっぱい…先ほどの計算結果－中心値＝248-456＝-208

分度器で最大曲げ角を測ると、左右に 40 度ずつ曲げることができました。これらをまとめると下図のようになります。



ちなみに、1度あたりの A/D 値は、211/40≒5 です。
同様に、自分のマイコンカーの値を計算しておきましょう。

6.4 プログラムの調整のポイント

このサンプルプログラムは、説明用マイコンカーの車体に合わせて作成しています。もし、このサンプルプログラムを使用する場合は、自分のマイコンカーに合わせる必要があります。そのポイントを解説します。

行	内容	説明
99～109	内輪の PWM 値	エクセルの「角度計算.xls」を使用します。ホイールベースとトレッドを自分のマイコンカーの長さに合わせて入力し、配列の値を更新してください。
183	左へハンドルを切ったとき PWM を 0 にするときの A/D 値	最大まで左へハンドルを切ったときの約 8 割の値を使用します。説明用マイコンカーは 211 だったので 211×0.8＝168.8≒170 を設定します。
186	左へ 5 度ほどハンドルを切ったときの A/D 値	説明用マイコンカーは 1 度あたり A/D 値は 5 なので、5 度のときの A/D 値は、 5×5=25 を設定します。
187,188	左へ 5 度以上ハンドルを切ったときの PWM 値	説明用マイコンカーは 80%ですが、それぞれのマイコンカーに合わせてください。
189	右へハンドルを切ったとき PWM を 0 にするときの A/D 値	最大まで右へハンドルを切ったときの約 8 割の値を使用します。説明用マイコンカーは-208 だったので -208×0.8＝-166.4≒-170 を設定します。
192	右へ 5 度ほどハンドルを切ったときの A/D 値	説明用マイコンカーは 1 度あたり A/D 値は 5 なので、5 度のときの A/D 値は、 5×-5=-25 を設定します。
193,194	右へ 5 度以上ハンドルを切ったときの PWM 値	説明用マイコンカーは 80%ですが、それぞれのマイコンカーに合わせてください。

221	クロスライン検出後の スピード	あらかじめ 1m/s で進んでいるときの 10ms ごとのパルス値を計算しておきます。説明用マイコンカーはロータリエンコーダ製作キット Ver.2 を使用しているので 10.92 です。 サンプルプログラムは約 1m/s で走行させています。整数しか使えないので数値は四捨五入します。今回は 11 になります。 それぞれのマイコンカーのクランクを曲がれるスピードに設定します。 もし自分のマイコンカーが 2m/s でクランクを曲がれるなら、 10.92×2＝21.84≒約 22 となります。
245	右クランク検出時の ハンドルを曲げる PWM 値	右クランク検出時、右にハンドルを曲げる PWM 値を設定します。今回は 50%です。右に曲げ続けると車体にステアリングがぶつかりロックしてしまいます。かといって小さすぎると、曲げるスピードが遅くなり脱輪の原因となります。カット&トライで調整してください。
246,247	右クランククリア時の PWM 値	右クランククリア時の PWM 値を設定します。説明用マイコンカーでは外輪を 60%、ハンドル角度を 40 度と仮定して、「角度計算(4WD 時).xls」で計算した値にします。右に曲がるので、右側のモータが外輪になります。
269	左クランク検出時の ハンドルを曲げる PWM 値	左クランク検出時、左にハンドルを曲げる PWM 値を設定します。今回は-50%です。左に曲げ続けると車体にステアリングがぶつかりロックしてしまいます。かといって小さすぎると、曲げるスピードが遅くなり脱輪の原因となります。カット&トライで調整してください。
270,271	左クランククリア時の PWM 値	左クランククリア時の PWM 値を設定します。サンプルプログラムでは外輪を 60%、ハンドル角度を 40 度と仮定して、「角度計算(4WD 時).xls」で計算した値にします。左に曲がるので、左側のモータが外輪になります。
963	サーボモータ PD 制御の比例定数	中心線からのずれに応じてサーボモータを回す強さです。この値は、モータドライブ基板 TypeS のディップスイッチ下位 4 ビットで 0～15 まで設定することができます。カット&トライで調整し、値が分かったら固定値にしてください。
964	サーボモータ PD 制御の微分定数	比例定数が小さすぎると中心線からずれたときの戻りが遅くなります。かといって大きすぎると、ハンドルが左右にブルブル震えて発振してしまいます。この微分定数を加えることにより、ブルブルを押さえることができます。 この値はモータドライブ基板 TypeS のディップスイッチ上位 4 ビットで 0～15 まで設定することができます。カット&トライで調整し、値が分かったら固定値にしてください。
973,974	サーボモータに加える PWM の上限設定	サーボモータに加える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。この数値を小さくしすぎると、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。最初は 50%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。 例) 970 :　　if(iRet > 90) iRet = 90; 971 :　　if(iRet < -90) iRet = -90;

6.5 自作サーボの角度指定

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。

6.5.1 PD制御

アナログセンサを PD 制御をしたとき、アナログセンサの値が 0 になるようにサーボモータを制御していました。これを、角度(ボリュームの 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＝kp×p kp　＝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 ＝getAnalogSensor()－0 ＝getAnalogSensor()	制御量 P＝kp×p kp　＝定数 p　＝**現在の角度　－目標の角度** ＝**getServoAngle ()－iSetAngle** ※目標の角度を iSetAngle 変数に入れます
微分制御	制御量 D＝kd×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()	制御量 D＝kd×d kd　＝定数 d　＝**過去の角度　－現在の角度** ＝**iAngleBefore2－getServoAngle()**

6.5.2 プログラム

(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           */
```

(2) 関数の追加

サーボ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。

サンプルプログラムは比例定数 20、微分定数 100、計算後の調整値は 1/2 にしています。この値は、サーボ機構の作り方により違ってきますので各自調整してください。

```
/*************************************************************************/
/* 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 = 20 * (j - i);              /* 比例 */
    iD = 100 * (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90くらいにしてください */
    iServoPwm2 = iRet;

    iAngleBefore2 = j;
}
```

(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。

```
void interrupt_timer3( void )
{
    unsigned int    i;

    ITU3_TSR &= 0xfe;               /* フラグクリア             */
    ITU3_BRB = iPWM_r;              /* PWM 値の更新             */
    ITU4_BRA = iPWM_l;              /* PWM 値の更新             */
    ITU4_BRB = iPWM_s;              /* PWM 値の更新             */
    cnt1++;

    /* サーボモータ制御 */
    servoControl();
    servoControl2();        追加
以下略
```

(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 プログラムの一番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しています。

```
void main( void )
{
    int             i, j;
    unsigned int    u;
    char            c;

    /* マイコン機能の初期化 */
    init();                             /* 初期化                          */
    initI2CEeprom( &P6DDR, &P6DR, 0x90, 6, 5);  /* EEP-ROM 初期設定       */
    initBeepS();                        /* ブザー関連処理                  */
    init_sci1( 0x00, 79 );              /* SCI1 初期化                      */
    set_ccr( 0x00 );                    /* 全体割り込み許可                */

    /* マイコンカーの状態初期化 */
    motor_mode_f( BRAKE, BRAKE );
    motor_mode_r( BRAKE, BRAKE );
    motor_mode_s( BRAKE );
    speed_f( 0, 0 );
    speed_r( 0, 0 );
    servoPwmOut( 0 );
    setBeepPatternS( 0x8000 );

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

    iSetAngle = 100;
    while( 1 ) {
        servoPwmOut( iServoPwm2 );
    }

以下略
```

角度指定が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マイコンカー制御に使用します。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自作サーボモータを移動させる例です。

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

7. 参考文献

・ルネサス エレクトロニクス(株)
H8/3048 シリーズ、H8/3048F-ZTAT™ (H8/3048F、H8/3048F-ONE)ハードウェアマニュアル 第 7 版
・ルネサス エレクトロニクス(株)
High-performance Embedded Workshop V.4.00 ユーザーズマニュアル Rev.3.00
・ルネサス エレクトロニクス(株) 半導体トレーニングセンター C言語入門コーステキスト 第 1 版
・(株)オーム社 H8 マイコン完全マニュアル 藤澤幸穂著 第1版
・電波新聞社 マイコン入門講座 大須賀威彦著 第 1 版
・(株)オーム社 C言語でH8マイコンを使いこなす 鹿取祐二著 第 1 版
・ソフトバンク(株) 新C言語入門シニア編 林晴比古著 初版
・共立出版(株) プログラマのための ANSI C 全書 L.Ammeraal 著
吉田敬一・竹内淑子・吉田恵美子訳 初版

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

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

H8 マイコンについての詳しい情報は、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」→「H8」でご覧頂けます

※リンクは、2010 年 9 月現在の情報です。