

マイコンカーラリー用  
トレーニングボード  
実習マニュアル  
kit06 版

第 1.10 版  
2007.03.05  
ジャパンマイコンカーラリー実行委員会

# 注意事項 (rev.1.1)

## 著作権

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

## 禁止事項

ユーザーは以下の内容を行うことはできません。

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

## 転載、複製

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

## 責任の制限

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

## その他

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

## 連絡先

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

# 目 次

1. 概要 .....	1
2. トレーニングボードの仕様 .....	2
2.1 外観 .....	2
2.1.1 トレーニングボード本体 .....	2
2.1.2 LCD .....	2
2.2 ブロック図 .....	3
2.3 コネクタ .....	4
2.4 部品 .....	5
2.5 回路図 .....	6
2.6 部品実装 .....	7
2.7 CPU ボードのコネクタ追加 .....	8
2.8 CPU ボードとトレーニングボードの接続 .....	8
3. サンプルプログラム .....	10
3.1 ルネサス統合開発環境 .....	10
3.2 サンプルプログラムのインストール .....	10
3.3 ワークスペース「training06」を開く .....	11
3.4 プロジェクト .....	12
4. プロジェクト「tr_01」 LCD の使い方 .....	14
4.1 内容 .....	14
4.2 プロジェクトの構成 .....	14
4.3 プログラム「tr_01.c」 .....	14
4.4 プログラムの解説 .....	16
4.4.1 ヘッダファイルの取り込み .....	16
4.4.2 「lcd2.c」と「lcd2.h」ファイルについて .....	16
4.4.3 lcd2.c ファイルで使用できる関数 .....	17
4.4.4 初期化 .....	18
4.4.5 表示データを作る .....	18
4.4.6 ポートの入出力設定 .....	19
4.4.7 LCD 表示処理関数 .....	19
4.4.8 LCD の表示スピードについて .....	19
4.4.9 ワークスペース「kit06」のプロジェクト「kit06」に lcd2.c を追加する例 .....	20
5. プロジェクト「tr_02」 プッシュスイッチの使い方 .....	21
5.1 内容 .....	21
5.2 プロジェクトの構成 .....	21
5.3 プログラム「tr_02.c」 .....	21
5.4 プログラムの解説 .....	22
5.4.1 ヘッダファイルの取り込み .....	22
5.4.2 switch.c ファイルで使用できる関数 .....	23
5.4.3 初期化 .....	24
5.4.4 プログラムでの使用 .....	25
5.4.5 スイッチ処理 .....	25

6. プロジェクト「tr_03」 ブザーの使い方 1 .....	26
6.1 内容 .....	26
6.2 プロジェクトの構成 .....	26
6.3 プログラム「tr_03.c」 .....	26
6.4 プログラム「tr_03start.src」 .....	27
6.5 プログラムの解説 .....	29
6.5.1 ヘッドファイルの取り込み .....	29
6.5.2 beep.c ファイルで使用できる関数 .....	29
6.5.3 初期化 .....	30
6.5.4 プログラムでの使用 .....	30
6.5.5 ブザー処理 .....	31
6.5.6 スタートアップルーチン「tr_03start.src」の追加、変更 .....	31
7. プロジェクト「tr_04」 ブザーの使い方 2 .....	33
7.1 内容 .....	33
7.2 プロジェクトの構成 .....	33
7.3 プログラム「tr_04.c」 .....	34
7.4 プログラムの解説 .....	35
7.4.1 プログラムでの使用 .....	35
8. プロジェクト「tr_05」 EEPROM の使い方 .....	36
8.1 内容 .....	36
8.2 プロジェクトの構成 .....	36
8.3 プログラム「tr_05.c」 .....	36
8.4 プログラムの解説 .....	37
8.4.1 ヘッドファイルの取り込み .....	37
8.4.2 eeprom.c ファイルで使用できる関数 .....	38
8.4.3 初期化 .....	39
8.4.4 EEPROM のチェック .....	39
8.4.5 プログラムでの使用 .....	40
8.4.6 EEPROM を使うポイント .....	41
9. プロジェクト「tr_11」 走行プログラムの高速化対応 .....	42
9.1 内容 .....	42
9.2 プロジェクトの構成 .....	42
9.3 プログラム「tr_11.c」 .....	42
9.4 プログラムの解説 .....	45
9.4.1 モータ、サーボの PWM 周期の変更 .....	45
9.4.2 ハンドルの切れ角 .....	45
9.4.3 クロスライン検出後、2 本目を読み飛ばすまでの時間 .....	46
9.4.4 クランク検出後の待ち時間 .....	46
9.4.5 右ハーフライン検出後の待ち時間 .....	47
9.4.6 左ハーフライン検出後の待ち時間 .....	47
10. プロジェクト「tr_12」 サーボのセンタ調整 .....	48
10.1 内容 .....	48
10.2 プロジェクトの構成 .....	48
10.3 プログラム「tr_12.c」 .....	48
10.4 プログラム「tr_12start.src」 .....	52

10.5	プログラムの解説	53
10.5.1	ヘッダファイルの追加	53
10.5.2	EEP-ROM エリアの確保	53
10.5.3	初期化	54
10.5.4	EEP-ROM から読み込み	54
10.5.5	スイッチ入力待ち	55
10.5.6	ポートの入出力設定	57
10.5.7	割り込み処理の追加	58
10.5.8	handle 関数の変更	58
10.5.9	スタートアップルーチン「tr_12start.src」の追加、変更	59
11.	プロジェクト「tr_13」 スピードの調整	60
11.1	内容	60
11.2	プロジェクトの構成	60
11.3	プログラム「tr_13.c」	60
11.4	プログラムの解説	63
11.4.1	EEP-ROM エリアの追加	63
11.4.2	EEP-ROM から読み込み	63
11.4.3	スイッチ入力待ち	64
11.4.4	speed 関数の変更	65
12.	プロジェクト「tr_14」 カーブでの左右回転差の計算	66
12.1	内容	66
12.2	プロジェクトの構成	66
12.3	プログラム「tr_14.c」	66
12.4	プログラムの解説	70
12.4.1	プロトタイプの追加	70
12.4.2	大域変数の追加、handle 関数内の追加	70
12.4.3	配列の追加	71
12.4.4	diff 関数の追加	72
12.4.5	プログラムでの使い方	73
12.4.6	配列データの作り方	73
13.	プロジェクト「tr_15」 大カーブでのセンサ状態の追加	76
13.1	内容	76
13.2	プロジェクトの構成	76
13.3	プログラム「tr_15.c」	76
13.4	プログラムの解説	78
13.4.1	パターン 12 右大曲げのパターン	78
13.4.2	パターン 13 左大曲げのパターン	79
14.	プロジェクト「tr_16」 大カーブでの PWM 値の調整	80
14.1	内容	80
14.2	プロジェクトの構成	80
14.3	プログラム「tr_16.c」	80
14.4	プログラムの解説	83
14.4.1	EEP-ROM エリアの追加	83
14.4.2	EEP-ROM から読み込み	84
14.4.3	スイッチ入力待ち	84
14.4.4	パターン 12 右大曲げ	86

14.4.5	パターン 13 左大曲げ .....	87
15.	プロジェクト「tr_17」 クロスライン検出後の PWM 値の調整 .....	88
15.1	内容 .....	88
15.2	プロジェクトの構成 .....	88
15.3	プログラム「tr_17.c」 .....	88
15.4	プログラムの解説 .....	91
15.4.1	EEP-ROM エリアの追加 .....	91
15.4.2	EEP-ROM から読み込み .....	92
15.4.3	スイッチ入力待ち .....	92
15.4.4	パターン 23 クロスライン後のトレース、クランク検出 .....	94
15.4.5	パターン 53 右ハーフライン後のトレース、レーンチェンジ .....	95
15.4.6	パターン 63 左ハーフライン後のトレース、レーンチェンジ .....	96
16.	プロジェクト「tr_21」 クロスライン検出後、10cm 直進させる(エンコーダ使用) .....	97
16.1	内容 .....	97
16.2	プロジェクトの構成 .....	97
16.3	プログラム「tr_21.c」 .....	97
16.4	エンコーダの回転数とパルス数の関係 .....	100
16.5	プログラムの解説 .....	101
16.5.1	エンコーダ関連の変数の宣言 .....	101
16.5.2	パターン1 スタート前の初期化 .....	101
16.5.3	パターン 21 クロスライン検出時の積算値を取得 .....	102
16.5.4	パターン 22 2本目を読み飛ばす .....	103
16.5.5	パターン 51 右ハーフライン検出時の積算値を取得 .....	104
16.5.6	パターン 52 2本目を読み飛ばす .....	105
16.5.7	パターン 61~62 左ハーフライン部分の処理 .....	107
16.5.8	入出力設定、ITU 設定の変更 .....	108
16.5.9	割り込み処理プログラムの改造 .....	109
16.6	エンコーダの回転数が違う場合の変更点 .....	110
17.	プロジェクト「tr_22」 大カーブで速いとブレーキ(エンコーダ使用) .....	111
17.1	内容 .....	111
17.2	プロジェクトの構成 .....	111
17.3	プログラム「tr_22.c」 .....	111
17.4	プログラムの解説 .....	116
17.4.1	EEP-ROM エリアの変更 .....	116
17.4.2	EEP-ROM から読み込み .....	116
17.4.3	スイッチ入力待ち .....	117
17.4.4	パターン 12 右へ大曲げの終わりのチェック .....	119
17.4.5	speed2 関数 .....	121
17.4.6	パターン 13 左へ大曲げの終わりのチェック .....	122
17.5	エンコーダの回転数が違う場合の変更点 .....	123
18.	プロジェクト「tr_23」 クロスライン等検出後の速度設定(エンコーダ使用) .....	124
18.1	内容 .....	124
18.2	プロジェクトの構成 .....	124
18.3	プログラム「tr_23.c」 .....	124
18.4	プログラムの解説 .....	129
18.4.1	EEP-ROM エリアの変更 .....	129

18.4.2	EEP-ROM から読み込み .....	129
18.4.3	スイッチ入力待ち .....	130
18.4.4	パターン 23 エンコーダでスピード制御するように変更 .....	131
18.4.5	パターン 53 右レーンチェンジの徐行部分をエンコーダでスピード制御するように変更 .....	132
18.4.6	パターン 63 左レーンチェンジの徐行部分をエンコーダでスピード制御するように変更 .....	132
18.5	エンコーダの回転数が違う場合の変更点 .....	132
19.	プロジェクト「tr_24」 クロスライン検出後の速度設定その2 (エンコーダ使用) .....	133
19.1	内容 .....	133
19.2	プロジェクトの構成 .....	133
19.3	プログラム「tr_24.c」 .....	133
19.4	プログラムの解説 .....	137
19.4.1	EEP-ROM エリアの追加 .....	137
19.4.2	EEP-ROM から読み込み .....	137
19.4.3	スイッチ入力待ち .....	138
19.4.4	パターン 23 エンコーダ値により2段階でスピード制御するように変更 .....	139
19.4.5	パターン 53 右レーンチェンジの徐行部分をエンコーダ値により2段階でスピード制御するように変更 .....	139
19.4.6	パターン 63 左レーンチェンジの徐行部分をエンコーダ値により2段階でスピード制御するように変更 .....	139
19.5	エンコーダの回転数が違う場合の変更点 .....	140
20.	プロジェクト「tr_25」 設定距離でマイコンカーを止める (エンコーダ使用) .....	141
20.1	内容 .....	141
20.2	プロジェクトの構成 .....	141
20.3	プログラム「tr_25.c」 .....	141
20.4	プログラムの解説 .....	144
20.4.1	EEP-ROM エリアの追加 .....	144
20.4.2	EEP-ROM から読み込み .....	144
20.4.3	スイッチ入力待ち .....	145
20.4.4	パターン 11 走行距離以上走ったかチェック .....	147
20.5	エンコーダの回転数が違う場合の変更点 .....	147
21.	プロジェクト「tr_31」 走行データの記録(内蔵 RAM 使用) .....	148
21.1	内容 .....	148
21.2	プロジェクトの構成 .....	148
21.3	プログラム「tr_31.c」 .....	148
21.4	プログラムの解説 .....	153
21.4.1	保存するデータ .....	153
21.4.2	定数の追加 .....	153
21.4.3	変数の追加 .....	153
21.4.4	パソコンとの通信するための初期設定 .....	154
21.4.5	パターン 1 スタート .....	154
21.4.6	パターン 11 の追加、変更 ログ転送処理へ .....	155
21.4.7	パターン 71 停止 .....	155
21.4.8	パターン 72 1秒待ち .....	155
21.4.9	パターン 73 スイッチが離されたかチェック .....	156
21.4.10	パターン 74 スイッチが押されたかチェック .....	156
21.4.11	パターン 75 タイトルの転送 .....	156
21.4.12	パターン 76 データの転送 .....	157
21.4.13	パターン 77 終了 .....	158
21.4.14	データの保存 .....	158

21.5 エンコーダの回転数が違う場合の変更点 .....	159
21.6 転送の仕方 .....	159
22. プロジェクト「tr_32」 走行データの記録(外付け EEPROM 使用).....	164
22.1 内容 .....	164
22.2 プロジェクトの構成 .....	164
22.3 マイコンカーの構成 .....	165
22.4 プログラム「tr_32.c」 .....	165
22.5 プログラムの解説 .....	169
22.5.1 保存するデータ .....	169
22.5.2 外付け EEPROM を使用するための初期設定 .....	169
22.5.3 転送データ .....	170
22.6 エンコーダの回転数が違う場合の変更点 .....	170
23. カーブでのタイヤの左右回転差の計算方法.....	171
23.1 2WD、後輪駆動、センターピボット方式 .....	171
23.2 センターピボット方式 4 輪の回転数計算 .....	174
23.3 アッカーマン方式 4 輪の回転数計算 .....	175
24. 参考文献 .....	178



## 1. 概要

通常、マイコンを使った機器(組み込み系と呼ばれています)のデバッグ(誤りの修正)は、デバッガと呼ばれる装置でパソコンと機器を接続して行います。パソコンの画面上で、現在実行しているプログラムの位置やレジスタの値等を確認しながら正常に動作しているかデバッグします。

マイコンカーの場合も、本当はデバッガを使用してプログラムの動作確認をすると良いのですが、動き回るためデバッガと接続することができません。そのため、マイコンカーの CPU ボードにプログラムを書き込み、コースを走らせ、修正箇所を見つけてまた書き込む、という作業を繰り返します。修正箇所が確実に分かっているのいいのですが、どこが問題なのか分かることは少なく、何度もプログラムを書き換えて試行錯誤しながら問題部分を直すのがほとんどだと思います。

そこで、2002 年以降支給している CPU ボード「RY3048F-ONE ボード」に搭載するトレーニングボードを作りました。

主な機能は、

### LCD(液晶)表示

プログラムにて、文字を表示して情報を確認します。32 文字表示できますので、LED で状態を表示するより格段に情報量が違います。

例) センサの情報表示、マイコンカーのパラメータの表示

### プッシュスイッチ

LCD 表示の情報の切り替え、パラメータの調整を行います。スイッチは5つあります。本プログラムでは、値の保存、メニューの切り替え(2スイッチ)、パラメータの増減(2スイッチ)で5つ使います。

例) LCD のメニューの切り替え、パラメータの増減用

### EEP-ROM(書き替え可能なメモリ)

ちょっとしたパラメータの変更をするのにプログラムの変更、書き込みを行うのは時間的に、そして H8 のフラッシュメモリの寿命から好ましくありません。そこで、安価な EEPROM を外部に取り付け、EEPROM 内にパラメータを保存します。LCD に数値を表示、スイッチで値を増減させ、EEPROM に保存させるとパソコンが無くともパラメータの変更ができるようになります。今回のトレーニングボードでは、-32,768 ~ 32,767 の値のデータを128 個保存することができます。

### ブザー

マイコンカーを走らせているとき、LED や LCD での確認は困難なため、音による確認ができるようにします。ブザーは、音階を自由にならすことができます。

例) クランクで音を鳴らす、急カーブで音を鳴らす

### LED

LCD の下に LED が8個付いています。LCD を表示させるか、LCD を取って LED を点灯させるかを基板上のジャンパで選びます。LCD を取り付けるまでもないが、情報を表示したい場合は LED にします。

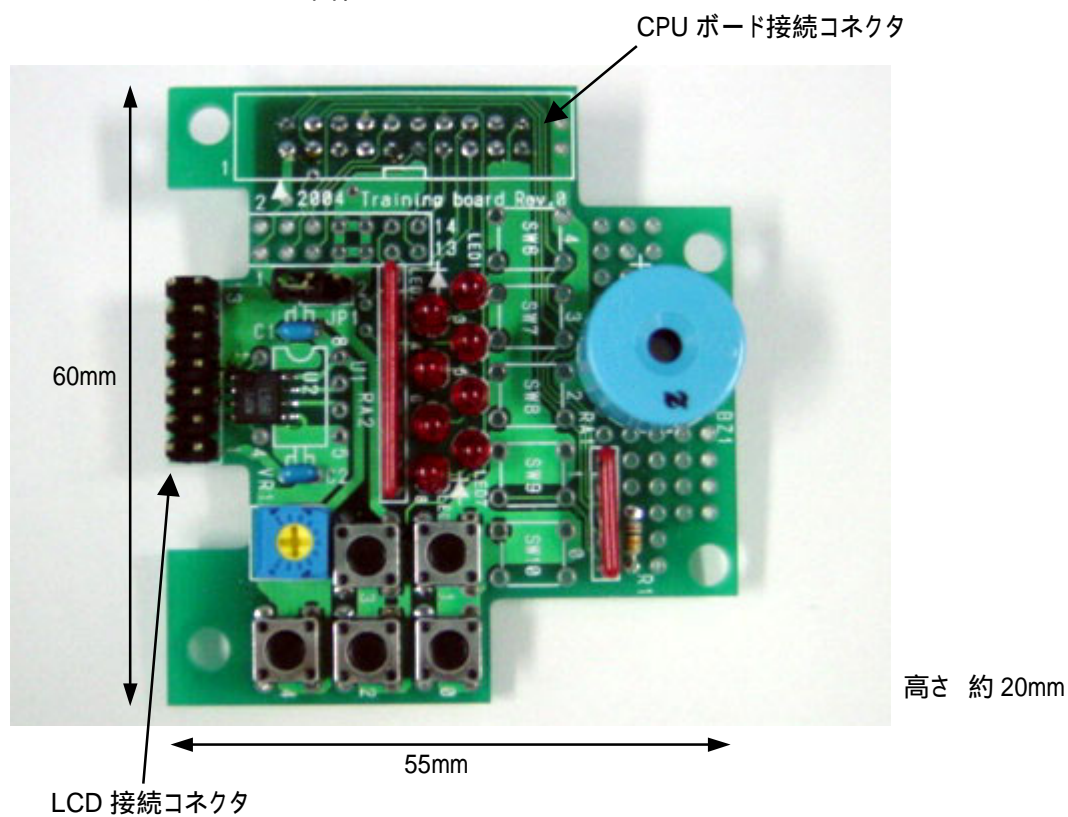
LCD を使うか、LED を使うか、どちらかしか選べません。

本書では、トレーニングボードの仕様について、各機能の使い方、マイコンカーに組み込んでパラメータの調整する方法を説明していきます。

## 2. トレーニングボードの仕様

### 2.1 外観

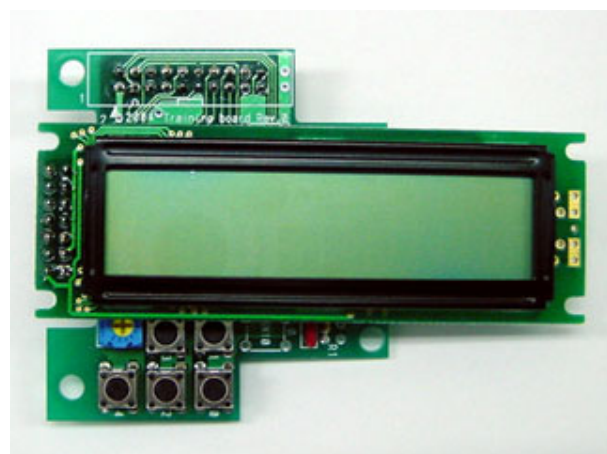
#### 2.1.1 トレーニングボード本体



#### 2.1.2 LCD

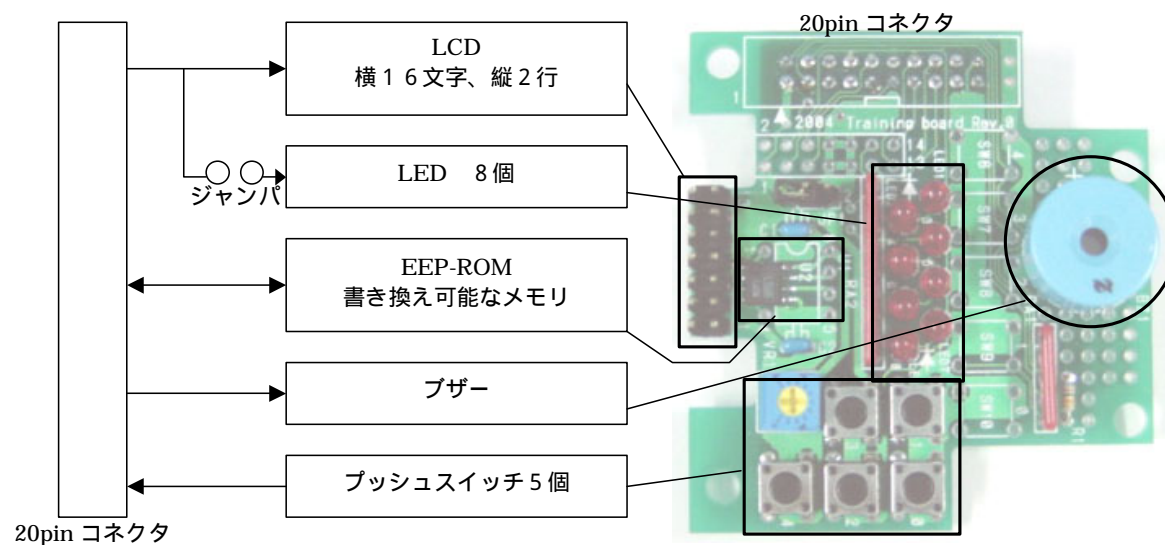


トレーニングボード本体の LCD コネクタ部に重ねて取り付けます。逆差し防止対策はしてないので、逆差しに気をつけてください。



トレーニングボードとLCDを重ねたところです。スイッチ、ボリュームは、この状態で操作できます。

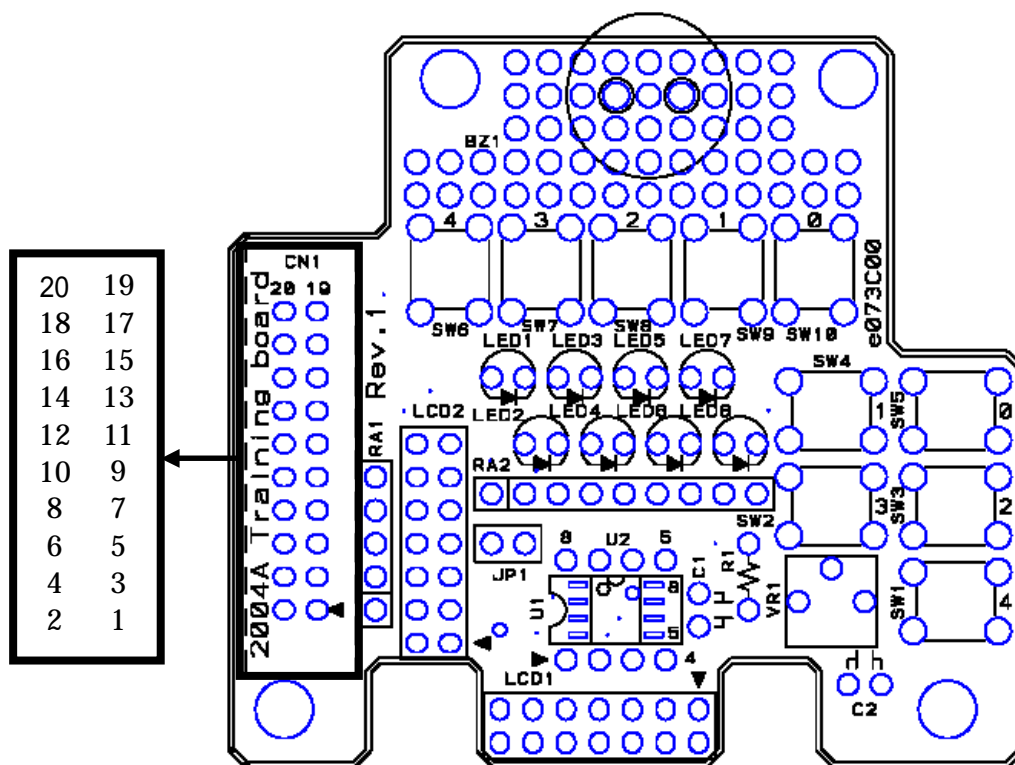
## 2.2 ブロック図



LCD	<p>プログラムにて、文字を表示して情報を確認します。32文字表示できますので、LEDで状態を表示するより格段に情報量が違います。</p> <p>例) センサの情報表示、マイコンカーのパラメータの表示</p>
LED	<p>LCDの下にLEDが8個付いています。LCDを表示させるか、LCDを取ってLEDを点灯させるかを基板上のジャンパで選びます。LCDを取り付けるまでもないが、情報を表示したい場合はLEDにします。</p> <p>LCDを使うか、LEDを使うかは、どちらかしか選べません。</p>
EEP-ROM	<p>ちょっとしたパラメータの変更をするのにプログラムの変更、書き込みを行うのは時間的に、そしてH8のフラッシュメモリの寿命から好ましくありません。そこで、安価なEEP-ROMを外部に取り付け、EEP-ROM内にパラメータを保存します。LCDに数値を表示、スイッチで値を増減させ、EEP-ROMに保存させるとパソコンが無くともパラメータの変更ができるようになります。今回のトレーニングボードでは、128ワードのデータを保存することができます。1ワードは16ビットです。</p>
ブザー	<p>マイコンカーを走らせているとき、LEDやLCDでの確認は困難なため、音による確認ができるようにします。ブザーは、音階を自由にならすことができます。</p> <p>例) クランクで音を鳴らす、急カーブで音を鳴らす</p>
プッシュスイッチ	<p>LCD表示の情報の切り替え、パラメータの調整を行います。スイッチは5つあります。本プログラムでは、値の保存、メニューの切り替え(2スイッチ)、パラメータの増減(2スイッチ)で5つ使います。</p> <p>例) LCDのメニューの切り替え、パラメータの増減用</p>

### 2.3 コネクタ

番号	信号名	CPU - ボード	DDRの設定	接続名	信号
1	Vcc	-		Vcc	
2	P36		0	SW_4	ON で"0"、OFF で"1"
3	P35		0	SW_3	ON で"0"、OFF で"1"
4	P34		0	SW_2	ON で"0"、OFF で"1"
5	P33		1	EED-ROM CS	
6	P32		1	EED-ROM SK	
7	P31		1	EED-ROM DI	
8	P30		0	EED-ROM DO	
9	P47		1	無接続 または LED	
10	P46		1	LCD E または LED	
11	P45		1	LCD RW または LED	
12	P44		1	LCD RS または LED	
13	P43		1	LCD D7 または LED	
14	P42		1	LCD D6 または LED	
15	P41		1	LCD D5 または LED	
16	P40		1	LCD D4 または LED	
17	P94		0	SW_1	ON で"0"、OFF で"1"
18	P92		0	SW_0	ON で"0"、OFF で"1"
19	P90		1	ブザー	
20	GND	-		GND	



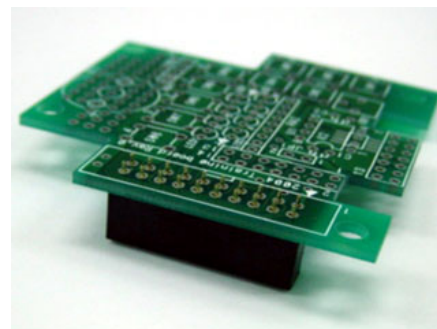
## 2.4 部品

番号	部品番号	部品名	型式・仕様	メーカー	数量	備考
1		基板	60.0 × 55.0mm × 1.6t		1	
2	U1	IC (DIP)	93LC56B-1/P	Microchip	1	入手しやすいどちらかを使用
3	U2	IC (SOP)	93LC56B-1/SN	Microchip		
4	CN1	20P オスコネクタ ストレート	HIF3FC-20PA2.54DSA	ヒロセ電機(株)	1	コネクタを上から取り付けフラットケーブルで接続する場合
5	CN1'	ピンソケット	20P(2*10) メス C-00083	(株)秋月電子通商		コネクタを下から取り付け、基板に重ねる場合
6	LCD1	LCDキャラクタディスプレイモジュール	16 × 2 行バックライト無し SC1602BS-B	(株)秋月電子通商	1	
7	C1-2	積層セラミックコンデンサ	0.1uF		2	
8	RA1	集合抵抗	M5-1 4 素子 1 コモン 10k	ピーアイ・テクノロジー ジャパン(株)	1	
9	RA2	集合抵抗	M9-1 8 素子 1 コモン 1k	ピーアイ・テクノロジー ジャパン(株)	1	
10	R1	抵抗	10k 1/8W 5mm ピッチ		1	
11	LED1-8	発光ダイオード	SEL2110R	サンケン電気(株)	8	
12	SW1-5	タクトスイッチ	B3F-1050	オムロン(株)	5	
13	VR1	半固定抵抗	CT-6P 10k	日本電産コパル 電子(株)	1	LCD 表示感度用
14	JP1	短絡コネクタ	XG8S-0231	オムロン(株)	1	
15	JP1 ソケット	短絡ソケット	XJ8A-0211	オムロン(株)	1	
16	BZ1	ブザー	KBS-13DB-4P-2	京セラ(株)	1	

U1,U2 は、同じ IC で、DIP タイプか SOP タイプかの違いです。どちらか入手しやすい方を選びます。  
CN1,CN1'は、20 ピンコネクタです。部品面に取り付け 20 ピンのフラットケーブルで CPU ボードと接続する場合は CN1 のコネクタ、半田面に取り付け CPU ボード上に重ねる場合は CN1' のコネクタを選びます。

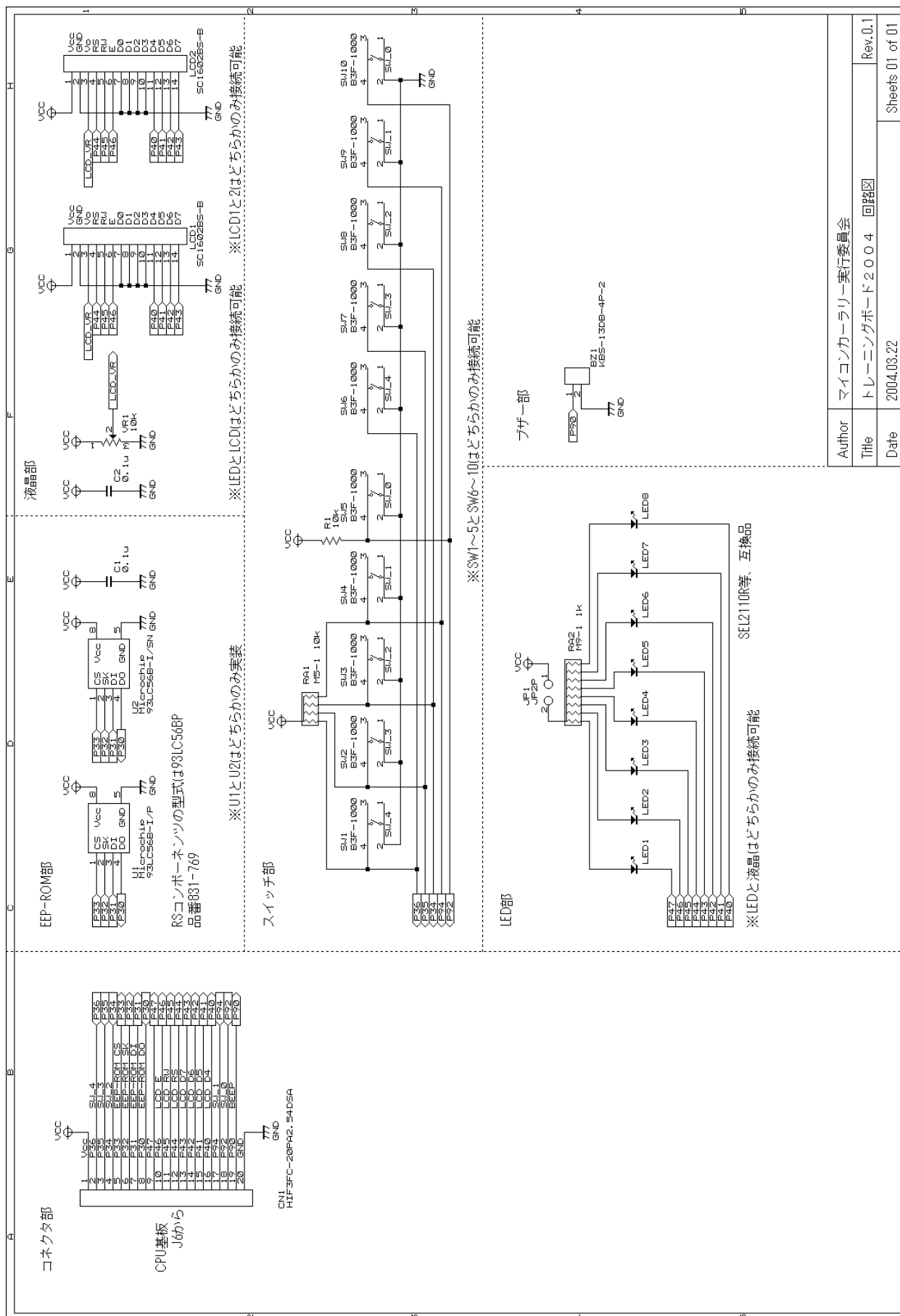


CN1 コネクタ実装例



CN1'コネクタ実装例

2.5 回路図



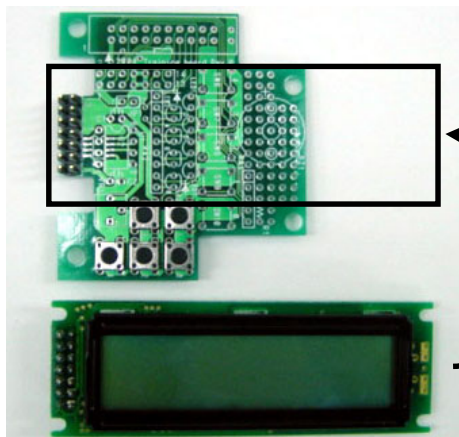
Author	マイコンカーラー実行委員会
Title	トレーニングボード2004 回路図
Date	2004.03.22
Rev.0.1	
Sheets 01 of 01	



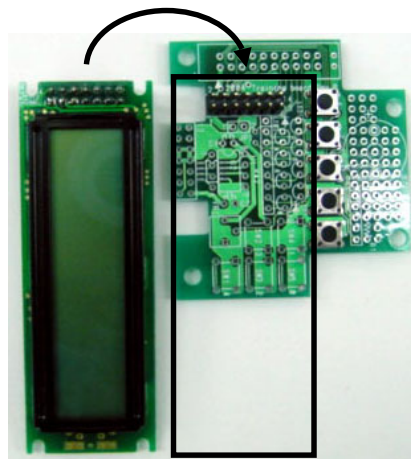
## 2.6 部品実装

LCD、スイッチの配置は2パターン選べます。

パターン1



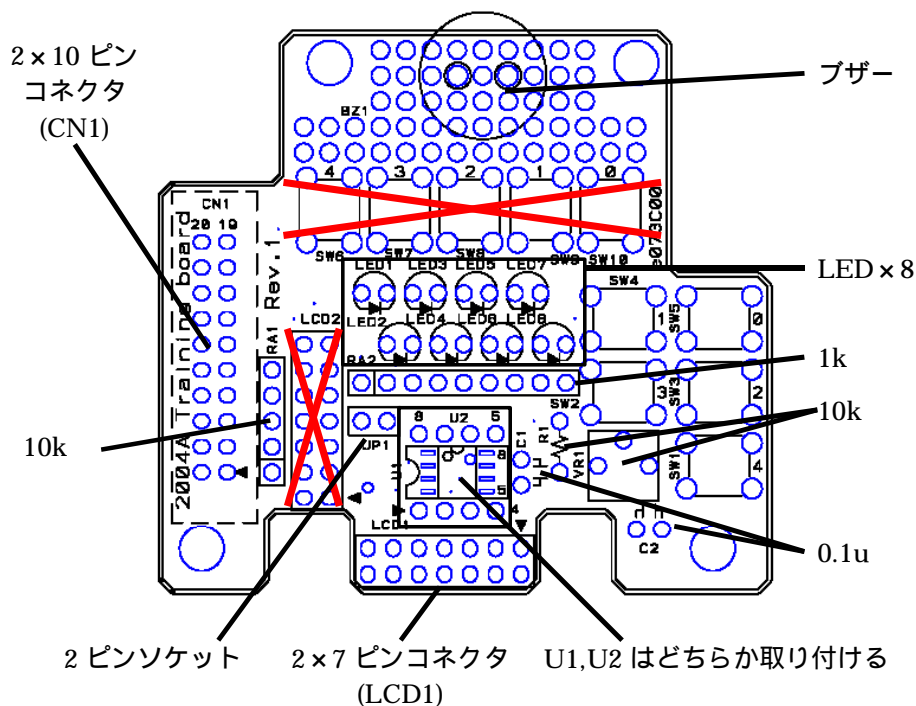
パターン2



LCD を横に配置します。スイッチはその下に取り付けます。

LCD を縦に配置します。スイッチはその右に取り付けます。

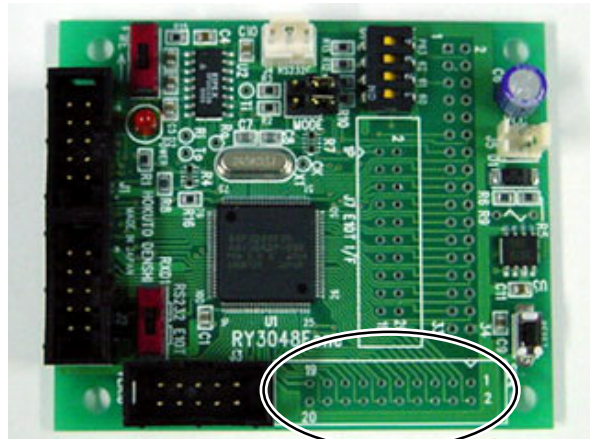
基板に部品を取り付けるとき、パターン1にするか2にするか決めます。両方一度にはできません。下図は、パターン1の時の実装図です。パターン2にする場合は、×印部分にスイッチとLCD用コネクタを取り付けます。



LEDを使わない場合は、LED1~8、1k の集合抵抗(RA2)、JP1 は実装する必要はありません。

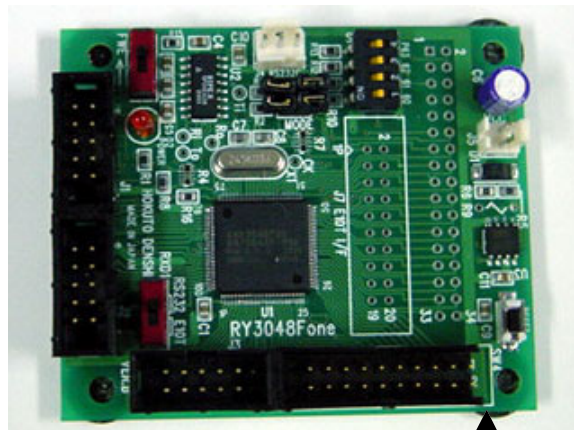
## 2.7 CPU ボードのコネクタ追加

CPU ボード「RY3048F-ONE」の J6 コネクタ部分は最初、何も実装されていません。トレーニングボードを接続するために、20 ピンコネクタオスを実装します。コネクタはトレーニングボード部品表の CN1 と同じヒロセ電機(株)「HIF3FC-20PA2.54DSA」、または同等のコネクタを実装します。



実装前

J6

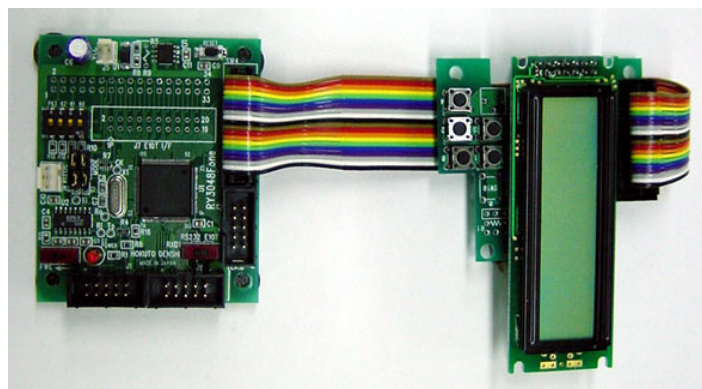


実装後

コネクタの1ピン側を少し削るとCPU  
ボードのネジ止めし易い

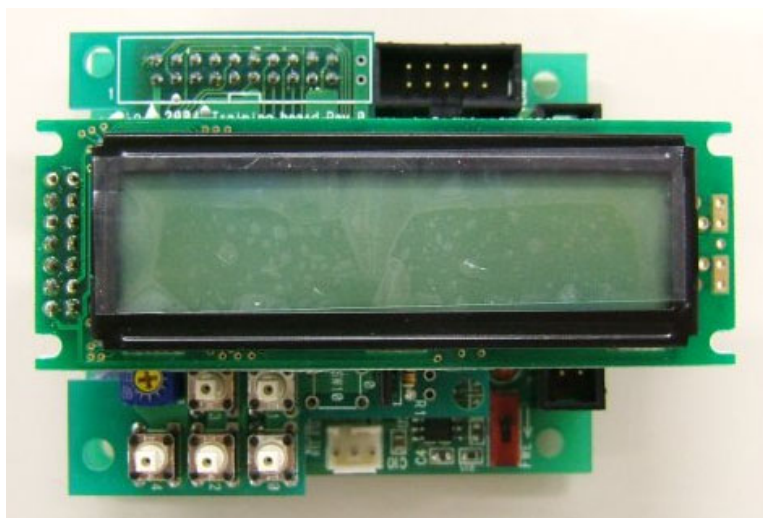
## 2.8 CPU ボードとトレーニングボードの接続

CPU ボードとトレーニングボードを別々に配置する場合、20 ピンのフラットケーブルでそれぞれを接続します。





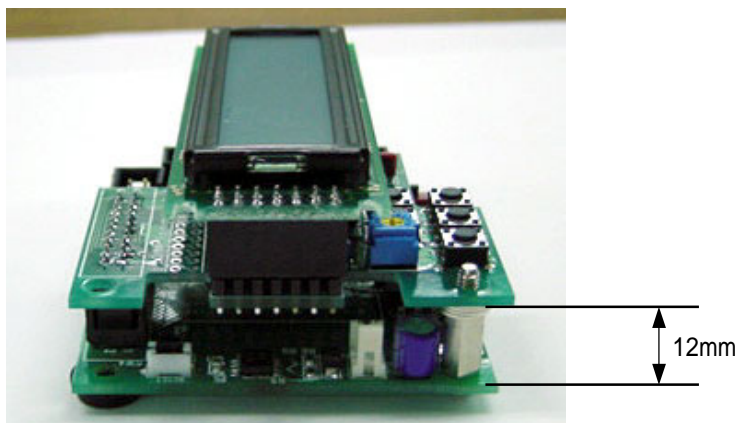
CPU ボードとトレーニングボードを重ねて配置する場合、写真のようにCPUボードの20ピンコネクタ部分に重ねます。



20ピンコネクタ左のネジ止め固定用穴を上から見ると、CPUボードの穴と同位置にあり、下が見えます。**見えない場合はコネクタの取り付け位置がずれていますので、直してください。**



固定は写真のように、高さが12mmになるようスタットなどを取り付け、ネジ止めします。



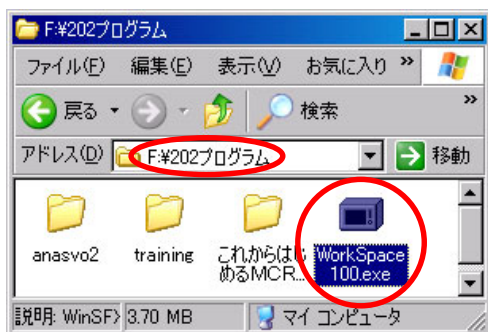
## 3. サンプルプログラム

### 3.1 ルネサス統合開発環境

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

### 3.2 サンプルプログラムのインストール

サンプルプログラムをインストールします。



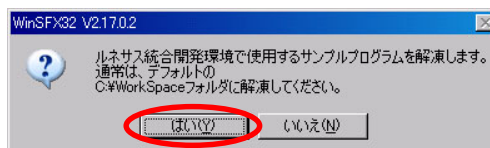
1. 講習会 CD の「CD ドライブ 202 プログラム」フォルダにある、Workspace100.exe を実行します。数字の100 は、バージョン 1.00 のことです。数字は変わることがあります。
2. または、マイコンカーラリーサイト「<http://www.mcr.gr.jp/>」の技術情報 ダウンロード内のページへ行きます。

#### マイコンカーキットプログラム、開発環境の資料

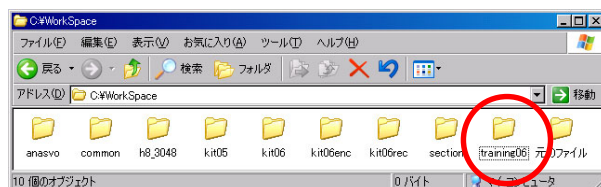
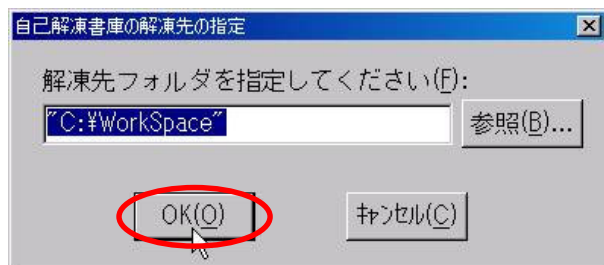
●ルネサス統合開発環境 操作マニュアル 第1.02版 2006.07.13 **NEW!**  
ルネサス統合開発環境のダウンロード方法、インストール方法、操作方法、及び設定方法が載っています。ルネサス統合開発環境は、ルネサステクノロジサイトよりダウンロードして下さい(マニュアル内に方法が記載されています)。  
→ [DOWNLOAD](#) (PDF 約19.3MB)

●ルネサス統合開発環境用その他ソフト Ver1.00 2006.07.06 **NEW!**  
ルネサス統合開発環境以外で使用するソフトをインストールします。  
→ [DOWNLOAD](#) (EXE 約0.3MB)

●ルネサス統合開発環境用マイコンカー関連プログラム Ver1.00 2006.07.06 **NEW!**  
ルネサス統合開発環境用のマイコンカー関係プログラムです。自己解凍方式で、実行すると自動でプログラムがインストールされます。  
→ [DOWNLOAD](#) (EXE 約3.68MB)



3. 「ルネサス統合開発環境用マイコンカー関連プログラム」をダウンロードします。
4. CD またはダウンロードした「Workspace100.exe」を実行します。「はい」をクリックします(数字は違うことがあります)。

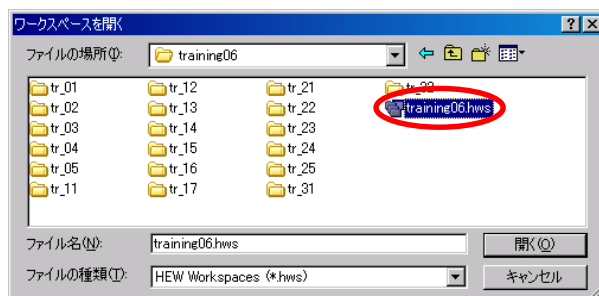
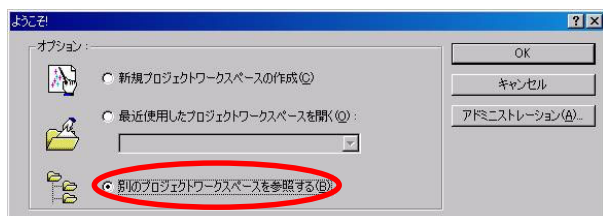


5. ファイルの解凍先を選択します。そのまま「OK」をクリックします。フォルダの変更はできません。
6. 解凍が終わったら、エクスプローラで「C ドライブ Workspace」フォルダを開いてみてください。「h8\_3048」フォルダ等がインストールされているはずですが、今回使用するのは、「training06」です。

### 3.3 ワークスペース「training06」を開く

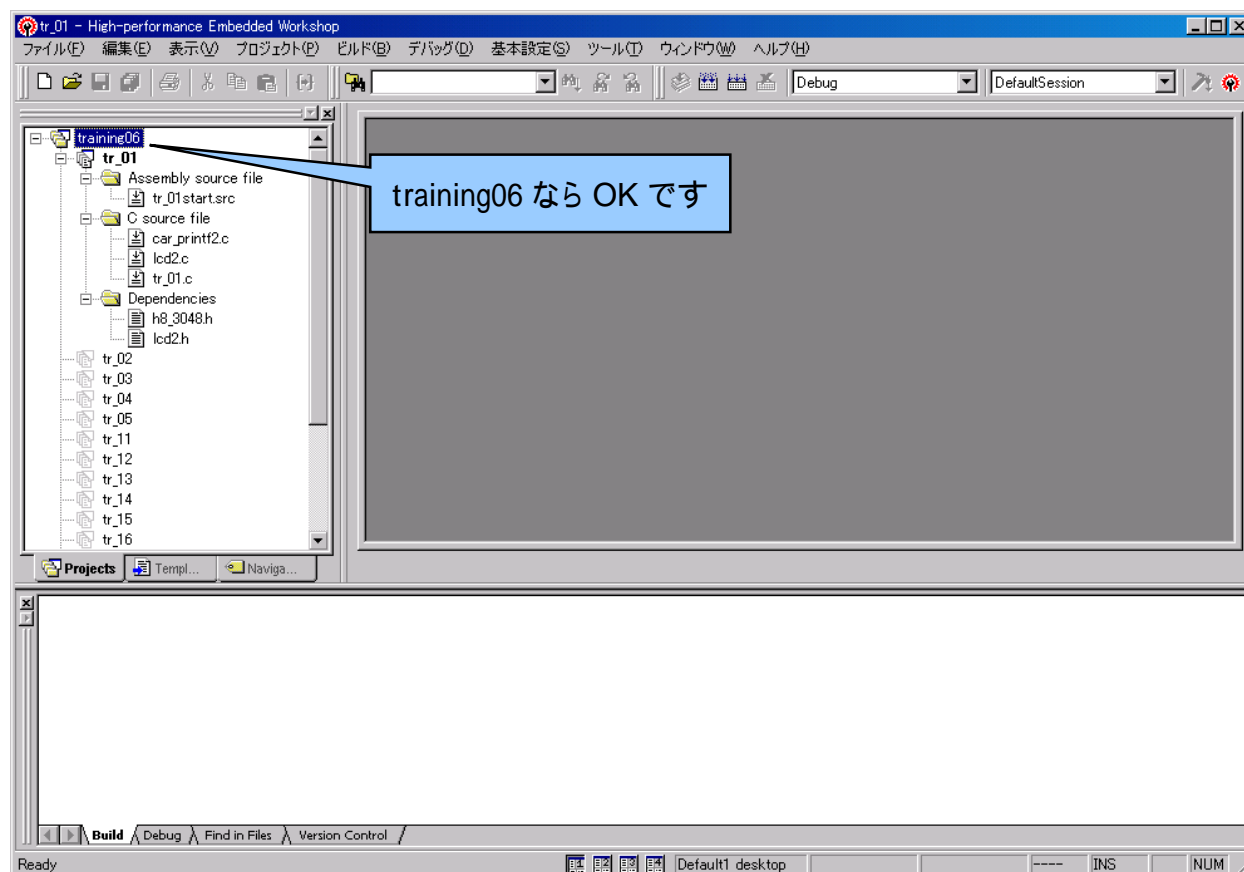


1. ルネサス統合開発環境を実行します。



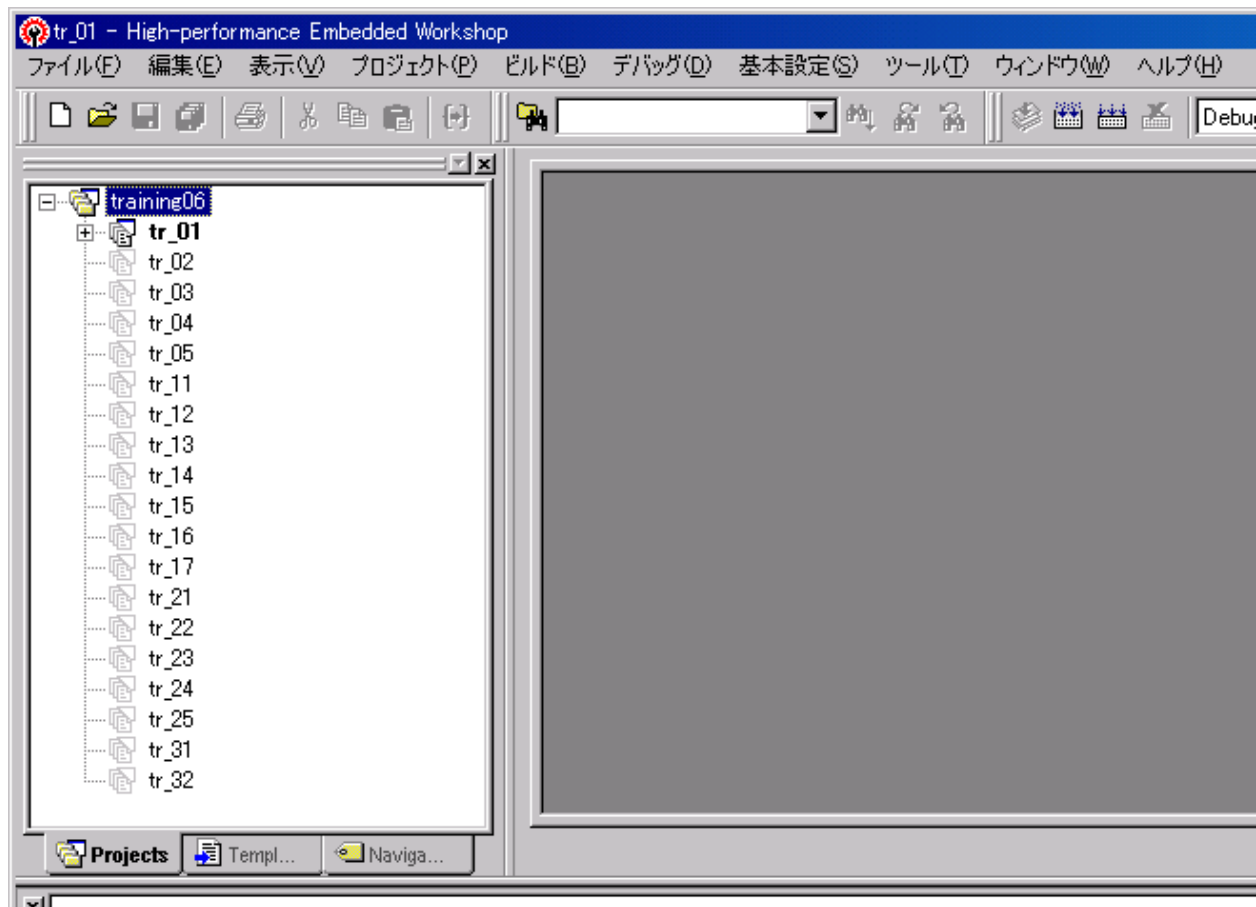
2. 「別のプロジェクトワークスペースを参照する」を選択します。

3. Cドライブ Workspace training06 の「training06.hws」を選択します。



4. 「training06」というワークスペースが開かれます。

### 3.4 プロジェクト



ワークスペース「training06」には、19つのプロジェクトが登録されています。元のプログラムは、ワークスペース「kit06」のプロジェクト「kit06」のプログラムです。このプロジェクトの複製を作り「tr\_01」とします。少しずつ機能を増やしてプロジェクト名の最後の数字も増やしていきます。

下記に、プロジェクト名とその内容を示します。

プロジェクト名	内容
tr_01	LCD の使い方 lcd2.c を使った、LCD の制御についてのサンプルプログラムです。
tr_02	プッシュスイッチの使い方 switch.c を使った、プッシュスイッチで値を操作するサンプルプログラムです。
tr_03	ブザーの使い方 beep.c を使った、ブザーで音を鳴らすサンプルプログラムです。
tr_04	ブザーの使い方2 beep.c を使った、先ほどとは違うブザーの鳴らし方をするサンプルプログラムです。
tr_05	EEP-ROM の使い方 eeprom.c を使った、EEP-ROM からデータ読み込み、データ保存を行うサンプルプログラムです。
tr_11	トレーニングボードを使用したプログラムを作る前に、マイコンカー走行プログラム「kit06.c」を、高速化対応させたプログラムです(トレーニングボードは使用していません)。
tr_12	サーボのセンタ調整をプッシュスイッチで行います。
tr_13	今まで、CPU ボード上のディップスイッチでPWMの設定を行っていましたが、プッシュスイッチで0~100%まで自由に設定します。
tr_14	カーブでは、内輪と外輪で回転差が出ます。それをテーブルデータとしてプログラムに追加します。ハンドル角度を変えても内外輪差を自動で計算するようにします。
tr_15	カーブのセンサ状態は、微曲げ、小曲げ、中曲げ、大曲げでしたが、大曲げ時のセンサ判別状態を増やしてカーブでの対応パターンを増やします。
tr_16	大曲げ時の PWM 値の設定を、プッシュスイッチで行います。
tr_17	クロスライン検出後の PWM 値の設定を、プッシュスイッチで行います。
tr_21	ロータリエンコーダを使用して、クロスライン、右ハーフライン、左ハーフライン検出後 10cm はセンサを見ないようにします。2 本目の誤検出防止用です。
tr_22	ロータリエンコーダを使用して、設定値以上のスピードで大カーブを検出するとブレーキするようにします。
tr_23	ロータリエンコーダを使用して、クロスライン、右ハーフライン、左ハーフライン検出後の速度を設定値一定にします。
tr_24	tr_23 の処理を 2 段階にして、高速なときでも対応できるようにします。
tr_25	ロータリエンコーダを使用して、一定距離でマイコンカーを止めるようにします。例えば、1 周 20m のコースなら 22m くらいに設定しておくくと便利です。
tr_31	内蔵 RAM を利用して、走行データを保存、走行後にパソコンへ転送してデータ解析できるようにします。
tr_32	外付け EEP-ROM(24C256)を利用して、走行データを保存、走行後にパソコンへ転送してデータ解析できるようにします。

## 4. プロジェクト「tr\_01」 LCD の使い方

### 4.1 内容

まずは、基本である LCD (液晶) に文字を表示してみます。文字だけでなく、cnt1 変数の値を表示します。

### 4.2 プロジェクトの構成

	<ul style="list-style-type: none"> <li>・tr_01start.src ベクタアドレスの設定、スタートアップルーチンが記述されています。</li> <li>・car_printf2.c セクションの初期化、printf 文、scanf 文を実行します。</li> <li>・lcd2.c <b>LCD 制御を行うために追加します。</b></li> <li>・tr_01.c メインプログラムです。</li> </ul>
--	---

### 4.3 プログラム「tr\_01.c」

プログラムのゴシック体部分が、「kit06.c」から追加した部分です。

```

1 : /*****
2 : /* トレーニングボードを使用したマイコンカートレースプログラム(kit06版) */
3 : /* 2006.08 ジャパンマイコンカーラリー実行委員会 */
4 : /*****
5 :
6 : /*=====*/
7 : /* インクルード */
8 : /*=====*/
9 : #include <no_float.h> /* stdioの簡略化 最初に置く*/
10: #include <stdio.h>
11: #include <machine.h>
12: #include "h8_3048.h"
13: #include "lcd2.h" /* LCD表示用追加 */
14:
15: /*=====*/
16: /* シンボル定義 */
17: /*=====*/
18:
19: /* 定数設定 */
20: #define TIMER_CYCLE 3071 /* タイマのサイクル 1ms */
21: /* /8で使用する場合、 */
22: /* /8 = 325.5[ns] */
23: /* TIMER_CYCLE = */
24: /* 1[ms] / 325.5[ns] */
25: /* = 3072 */
26: #define PWM_CYCLE 49151 /* PWMのサイクル 16ms */
27: /* PWM_CYCLE = */
28: /* 16[ms] / 325.5[ns] */
29: /* = 49152 */
30: #define SERVO_CENTER 5000 /* サーボのセンタ値 */
31: #define HANDLE_STEP 26 /* 1 分の値 */
32:
33: /* マスク値設定 x : マスクあり(無効) : マスク無し(有効) */
34: #define MASK2_2 0x66 /* x x x x */
35: #define MASK2_0 0x60 /* x x x x x x */
36: #define MASK0_2 0x06 /* x x x x x x */
37: #define MASK3_3 0xe7 /* x x x x x x */
38: #define MASK0_3 0x07 /* x x x x x x */
39: #define MASK3_0 0xe0 /* x x x x x x */
40: #define MASK4_0 0xf0 /* x x x x x x */
41: #define MASK0_4 0x0f /* x x x x x x */
42: #define MASK4_4 0xff /* x x x x x x */
43:
44: /*=====*/
45: /* プロトタイプ宣言 */

```

液晶が表示されない場合、  
液晶の近くにあるボリューム  
を調整してみてください。

## トレーニングボード 実習マニュアル(kit06 版)

```

46 : /*=====*/
47 : void init( void );
48 : void timer( unsigned long timer_set );
49 : int check_crossline( void );
50 : int check_rightline( void );
51 : int check_leftline( void );
52 : unsigned char sensor_inp( unsigned char mask );
53 : unsigned char dipsw_get( void );
54 : unsigned char pushsw_get( void );
55 : unsigned char startbar_get( void );
56 : void led_out( unsigned char led );
57 : void speed( int accele_l, int accele_r );
58 : void handle( int angle );
59 : char unsigned bit_change( char unsigned in );
60 :
61 : /*=====*/
62 : /* グローバル変数の宣言 */
63 : /*=====*/
64 : unsigned long cnt0; /* timer関数用 */
65 : unsigned long cnt1; /* main内で使用 */
66 : int pattern; /* パターン番号 */
67 :
68 : /*=====*/
69 : /* メインプログラム */
70 : /*=====*/
71 : void main( void )
72 : {
73 :     int i;
74 :
75 :     /* マイコン機能の初期化 */
76 :     init(); /* 初期化 */
77 :     set_ccr( 0x00 ); /* 全体割り込み許可 */
78 :     initLcd(); /* LCD初期化 */
79 :
80 :     /* マイコンカーの状態初期化 */
81 :     handle( 0 );
82 :     speed( 0, 0 );
83 :
84 :     while( 1 ) {
85 :         lcdPosition( 0, 0 );
86 :         lcdPrintf( "TrainingBoard'06" );
87 :         lcdPosition( 0, 1 );
88 :         lcdPrintf( "time = %08ld ", cnt1 );
89 :     }
90 :
91 :     while( 1 ) {
92 :         switch( pattern ) {

```

中略

```

494 : /*=====*/
495 : /* H8/3048F-ONE 内蔵周辺機能 初期化 */
496 : /*=====*/
497 : void init( void )
498 : {
499 :     /* I/Oポートの入出力設定 */
500 :     P1DDR = 0xff;
501 :     P2DDR = 0xff;
502 :     P3DDR = 0x8e; /* スイッチ、EEP-ROM */
503 :     P4DDR = 0xff; /* LCD接続 */
504 :     P5DDR = 0xff;
505 :     P6DDR = 0xf0; /* CPU基板上のDIP SW */
506 :     P8DDR = 0xff;
507 :     P9DDR = 0xe3; /* bit4,2:sw bit3:232c */
508 :     PADDR = 0xf7; /* スタートバー検出センサ */
509 :     PBDR = 0xc0;
510 :     PBDDR = 0xfe; /* モータドライブ基板Vol.3 */
511 :     /* センサ基板のP7は、入力専用なので入出力設定はありません */
512 :
513 :     /* ITU0 1ms毎の割り込み */
514 :     ITU0_TCR = 0x23;
515 :     ITU0_GRA = TIMER_CYCLE;
516 :     ITU0_IER = 0x01;
517 :
518 :     /* ITU3,4 リセット同期PWMモード 左右モータ、サーボ用 */
519 :     ITU3_TCR = 0x23;
520 :     ITU_FCR = 0x3e;
521 :     ITU3_GRA = PWM_CYCLE; /* 周期の設定 */
522 :     ITU3_GRB = ITU3_BRB = 0; /* 左モータのPWM設定 */
523 :     ITU4_GRA = ITU4_BRA = 0; /* 右モータのPWM設定 */
524 :     ITU4_GRB = ITU4_BRB = SERVO_CENTER; /* サーボのPWM設定 */
525 :     ITU_TOER = 0x38;
526 :
527 :     /* ITUのカウンタスタート */
528 :     ITU_STR = 0x09;
529 : }
530 :
531 : /*=====*/
532 : /* ITU0 割り込み処理 */
533 : /*=====*/
534 : #pragma interrupt( interrupt_timer0 )

```



```

535 : void interrupt_timer0( void )
536 : {
537 :     ITU0_TSR &= 0xfe;           /* フラグクリア          */
538 :     cnt0++;
539 :     cnt1++;
540 :
541 :     /* LCD表示処理用関数です。1ms毎に実行します。          */
542 :     lcdShowProcess();
543 : }

```

以下、略

## 4.4 プログラムの解説

**液晶が表示されない場合、液晶の近くにあるボリュームを調整してみてください。**

### 4.4.1 ヘッドファイルの取り込み

```

9 : #include <no_float.h>           /* stdio の簡略化 最初に置く */
10 : #include <stdio.h>
11 : #include <machine.h>
12 : #include "h8_3048.h"
13 : #include "lcd2.h"               /* LCD 表示用追加          */

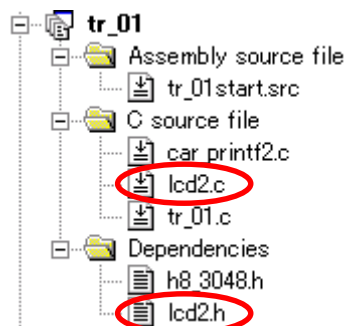
```

10 行目の「stdio.h」は、標準ライブラリと呼ばれるファイルで、ルネサス統合開発環境(コンパイラ側)で用意されているヘッドファイルです。これは、「printf」関数を使用するのに必要です。後述する「lcdPrintf」関数内で printf が使用されています。

9 行目の「no\_float.h」は、ルネサス統合開発環境専用のヘッドファイルです。「stdio.h」をインクルードすると、MOT ファイルがかなり大きくなってしまいます。もし、浮動小数点演算をしない場合、このファイルをインクルードすると MOT ファイルのサイズを抑えることができます。今回は、浮動小数点演算を行っていないのでこのファイルを追加しています。

13 行目の「lcd2.h」は LCD を制御するためのプログラム「lcd2.c」ファイルを使うために、インクルードしなければいけないヘッドファイルです。

### 4.4.2 「lcd2.c」と「lcd2.h」ファイルについて



- ・lcd2.c...LCD 制御プログラムです。  
追加方法は後述します。
- ・lcd2.h...「tr\_01.c」内で、「lcd2.c」内の関数を使うために  
#include "lcd2.h"  
という記述を追加します。  
**lcd2.c を追加すれば、Dependencies 欄に自動的に追加されます。**



#### 4.4.3 lcd2.c ファイルで使用できる関数

「lcd2.c」ファイルを追加したことにより使用できる関数は、下記の表のとおりです。

関数名	内容
initLcd	int initLcd( void ) LCD を初期化します。必ず割り込みを許可した後に実行します。 例) initLcd();
lcdPosition	void lcdPosition(char x ,char y) LCD に表示する位置を決めます。 引数:横(0~15)、縦(0~1) 例) lcdPosition( 5, 1 );
lcdPrintf	int lcdPrintf(char *format, ...) LCD に表示します。書式は、printf と同じです。表示位置は前回に表示された続きか、lcdPosition 関数で指定された位置です。 例) lcdPrintf( "time = %08ld ", cnt1 );
lcdShowProcess	void lcdShowProcess( void ) LCD の制御を行います。1ms ごとにこの関数を実行するようにします。 例) lcdShowProcess();
lcd_position	void lcd_position(char tx ,char ty) LCD に表示する位置を決めます。 引数:横(0~15)、縦(0~1) 例) lcd_position( 5, 1 ); lcdPosition 関数と同機能です。電波新聞社「これからはじめるマイコンカーラリー」で紹介されているサンプルプログラムと互換性を持たすために定義しています。
lcd_put_hex	void lcd_put_hex(int cnt) 引数の値を 16 進数 2 桁に変換して、LCD に表示します。表示位置は前回に表示された続きか、lcdPosition 関数で指定された位置です。 例) lcd_put_hex( 31 ); LCD に"1F"と表示される
lcd_put_num	void lcd_put_num(long cnt, int keta) 引数の値を 10 進数で LCD に表示します。表示する桁数を指定します。表示位置は前回に表示された続きか、lcdPosition 関数で指定された位置です。 例) lcd_put_num( 123, 4 ); LCD に"0123"と表示される
lcd_put_str	void lcd_put_str(char *str) 引数のアドレスの文字列を表示します。'¥0'のコードが有るまで続けます。表示位置は前回に表示された続きか、lcdPosition 関数で指定された位置です。 例) lcd_put_str( "abcdefg" ); LCD に"abcdefg"と表示されます(文字列の最後には自動で'¥0'がつきます)。
setLcdBuff	void setLcdBuff( char *rp ) LCD に表示したい配列の先頭アドレスを渡します。 引 数:表示したいアドレス 戻り値:なし  lcd.h との互換性を持たすためにあります。通常は使用しないでください。

#### 4.4.4 初期化

```

75 :      /* マイコン機能の初期化 */
76 :      init();                          /* 初期化                */
77 :      set_ccr( 0x00 );                  /* 全体割り込み許可    */
78 :      initLcd();                        /* LCD 初期化          */
    
```

割り込みが許可された状態 (「set\_ccr(0x00)」以降) で、LCD の初期化関数を実行します。

#### 4.4.5 表示データを作る

```

84 :      while( 1 ) {
85 :          lcdPosition( 0, 0 );          LCD の 0 列目 0 行目にセット
86 :          lcdPrintf( "TrainingBoard'06" );
87 :          lcdPosition( 0, 1 );          LCD の 0 列目 1 行目にセット
88 :          lcdPrintf( "time = %08ld ", cnt1 );
89 :      }
    
```

lcdPosition 関数は、lcdPrintf 関数等で LCD へ表示する時の位置を指定しています。86 行で LCD の 0 行目に表示する文字列を、88 行で LCD の 1 行目に表示する文字列を作っています。「"time = %08ld ", cnt1」となっているので%の後は、

- 0...空白桁を 0 で埋めます。
- 8...幅を指定します。8 桁です。
- l...引数を long 型のサイズとして扱います。
- d...10 進数に変換して表示します。

結果、cnt1 の値を、8 桁で空白桁は 0 で埋めて 10 進数に変換します。例えば、cnt1 に 12345 が代入されていたとすると、

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	列
0	行	T	r	a	i	n	i	n	g	B	o	a	r	d	'	0	6	
1		t	i	m	e	=		0	0	0	1	2	3	4	5			

となります。

マイコンカープログラム kit06.c では、cnt1 変数を時間計測用に使用しました。LCD で cnt1 の値が変化することを見ることができます。

#### 4.4.6 ポートの入出力設定

```

494 : /*****/
495 : /* H8/3048F-ONE 内蔵周辺機能 初期化 */
496 : /*****/
497 : void init( void )
498 : {
499 :     /* I/O ポートの入出力設定 */
500 :     P1DDR = 0xff;
501 :     P2DDR = 0xff;
502 :     P3DDR = 0x8e;          /* スイッチ、EEP-ROM */
503 :     P4DDR = 0xff;        /* LCD 接続 */
504 :     P5DDR = 0xff;
505 :     P6DDR = 0xf0;       /* CPU 基板上的 DIP SW */
506 :     P8DDR = 0xff;
507 :     P9DDR = 0xe3;       /* bit4,2:sw bit3:232c */
508 :     PADDR = 0xf7;      /* スタートバー検出センサ */
509 :     PBDR = 0xc0;
510 :     PBDDR = 0xfe;      /* モータドライブ基板 Vol.3 */
511 :     /* センサ基板の P7 は、入力専用なので入出力設定はありません */

```

トレーニングボードのコネクタに合わせて、出力する機器は出力、入力する機器は入力にポートを設定します。EEP-ROM 等、まだ使っていない機器も、繋がっているなら必ず入出力設定を行います。

#### 4.4.7 LCD 表示処理関数

```

531 : /*****/
532 : /* ITU0 割り込み処理 */
533 : /*****/
534 : #pragma interrupt( interrupt_timer0 )
535 : void interrupt_timer0( void )
536 : {
537 :     ITU0_TSR &= 0xfe;    /* フラグクリア */
538 :     cnt0++;
539 :     cnt1++;
540 :
541 :     /* LCD 表示処理用関数です。1ms 毎に実行します。 */
542 :     lcdShowProcess();
543 : }

```

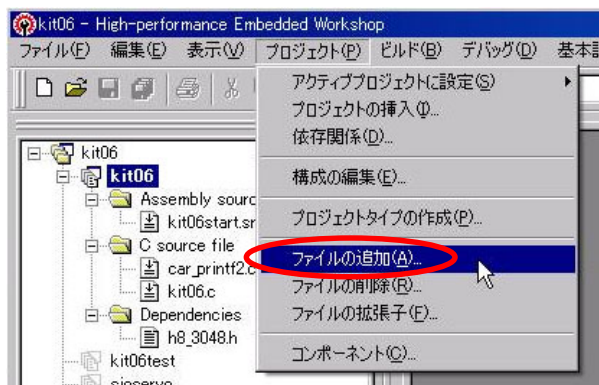
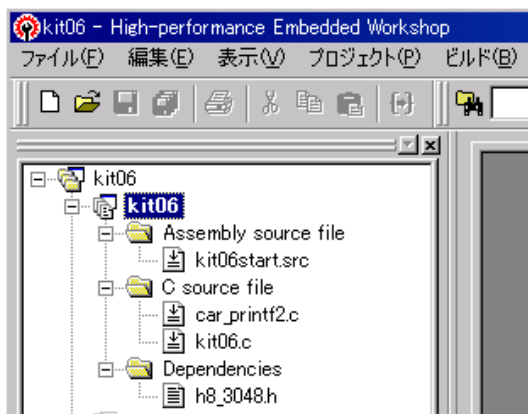
実は、lcdPosition 関数や lcdPrintf 関数は、表示をする準備をしているだけです。実際に表示させているのが lcdShowProcess 関数です。1ms ごとにこの関数を実行します。「kit06.c」ではちょうど interrupt\_timer0 関数が 1ms ごとに呼ばれているので、この割り込み関数内に lcdShowProcess 関数を記述しています。

#### 4.4.8 LCD の表示スピードについて

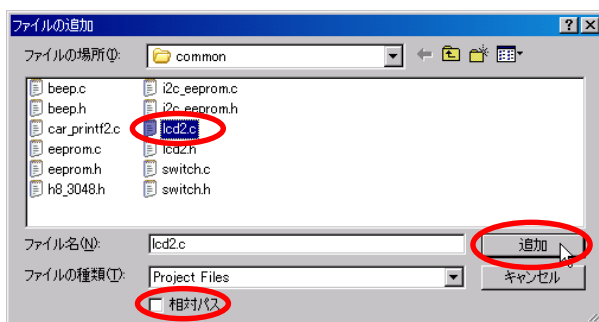
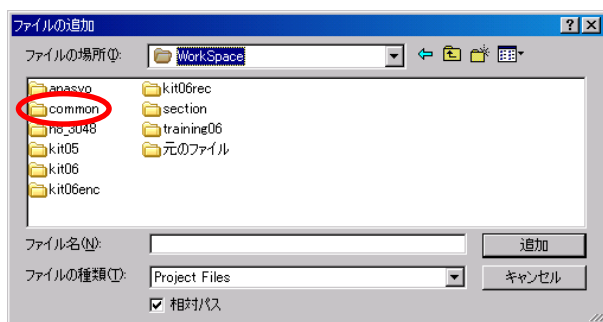
LCD の表示は遅いです。データシートによると、1文字表示に最大で 10ms の時間がかかります。32文字表示しようとする、最大で 320ms かかってしまいます。1回の表示に 320ms も時間を取られては他の処理が何もできなくなり大問題です。

lcdShowProcess 関数では、LCD の状態をチェックして、表示待ちの時はすぐに処理を中止して次の処理を行うようにします。そのため、LCD 表示だけに時間を取られてマイコンカーの制御ができないということはありません。

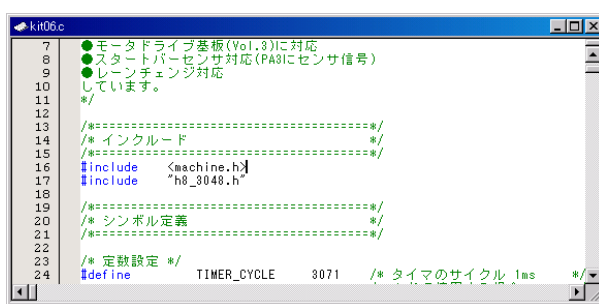
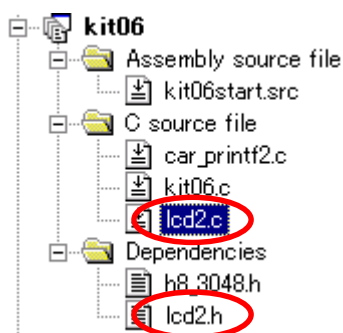
4.4.9 ワークスペース「kit06」のプロジェクト「kit06」に lcd2.c を追加する例



1. ルネサス統合開発環境でワークスペース「kit06」を開きます。プロジェクト「kit06」を有効なプロジェクトにします (太字であれば OK です)。
2. 「プロジェクト ファイルの追加」を選択します。



3. 「c:\workspace」フォルダにある、「common」フォルダを選択します。
4. 最初に相対パス欄のチェックを**はず**します。「lcd2.c」を選択、**追加**をクリックします。「lcd2.h」は**追加**しません。



5. リストに、「lcd2.c」が追加されました。lcd2.h は Dependencies 欄に自動で追加されます。
6. 後は、「kit06.c」ファイル内に、LCD 制御に関するプログラムを追加してください。

## 5. プロジェクト「tr\_02」 プッシュスイッチの使い方

### 5.1 内容

5つのスイッチを押すことにより、LCD に表示される数字が+1、-1、+10、-10、クリアされます。

### 5.2 プロジェクトの構成

	<ul style="list-style-type: none"> <li>·tr_02start.src ベクタアドレスの設定、スタートアップルーチンが記述されています。</li> <li>·car_printf2.c セクションの初期化、printf 文、scanf 文を実行します。</li> <li>·lcd2.c LCD 制御を行います。</li> <li>·switch.c <b>スイッチ制御を行います。</b></li> <li>·tr_02.c メインプログラムです。</li> </ul>
--	---

### 5.3 プログラム「tr\_02.c」

プログラムのゴシック体部分が追加した部分です。

```

1 :  /*****
2 :  /* トレーニングボードを使用したマイコンカートレースプログラム(kit06版) */
3 :  /*                               2006.08 ジャパンマイコンカーラリー実行委員会 */
4 :  /*****
5 :
6 :  /*=====*/
7 :  /* インクルード */
8 :  /*=====*/
9 :  #include <no_float.h> /* stdioの簡略化 最初に置く*/
10: #include <stdio.h>
11: #include <machine.h>
12: #include "h8_3048.h"
13: #include "lcd2.h" /* LCD表示用追加 */
14: #include "switch.h" /* スイッチ追加 */

```

中略

```

69:  /*****
70:  /* メインプログラム */
71:  /*****
72:  void main( void )
73:  {
74:      int    i, j;
75:
76:      /* マイコン機能の初期化 */
77:      init(); /* 初期化 */
78:      set_ccr( 0x00 ); /* 全体割り込み許可 */
79:      initLcd(); /* LCD初期化 */
80:      initSwitch(); /* スイッチ初期化 */
81:
82:      /* マイコンカーの状態初期化 */
83:      handle( 0 );
84:      speed( 0, 0 );
85:
86:      j = 0;
87:      while( 1 ) {
88:          /* スイッチ処理 */
89:          if( getSwFlag(SW_0) ) {
90:              i--;
91:              if( j < -10000 ) j = -10000;
92:          }
93:          if( getSwFlag(SW_1) ) {
94:              j++;

```

```

95 :         if( j > 10000 ) j = 10000;
96 :     }
97 :     if( getSwFlag(SW_2) ) {
98 :         j -= 10;
99 :         if( j < -10000 ) j = -10000;
100 :    }
101 :    if( getSwFlag(SW_3) ) {
102 :        j += 10;
103 :        if( j > 10000 ) j = 10000;
104 :    }
105 :    if( getSwFlag(SW_4) ) {
106 :        j = 0;
107 :    }
108 :    /* LCD処理 */
109 :    lcdPosition( 0, 0 );
110 :    lcdPrintf( "getSwNow() = %02x", getSwNow() );
111 :    lcdPosition( 0, 1 );
112 :    lcdPrintf( "data = %+06d", j );
113 : }
114 :
115 : while( 1 ) {
116 :     switch( pattern ) {

```

中略

```

555 : /******
556 : /* ITU0 割り込み処理
557 : /******
558 : #pragma interrupt( interrupt_timer0 )
559 : void interrupt_timer0( void )
560 : {
561 :     ITU0_TSR &= 0xfe;          /* フラグクリア          */
562 :     cnt0++;
563 :     cnt1++;
564 :
565 :     /* LCD表示処理用関数です。1ms毎に実行します。          */
566 :     lcdShowProcess();
567 :     /* 拡張スイッチ用関数です。1ms毎に実行します。          */
568 :     switchProcess();
569 : }

```

以下、略

## 5.4 プログラムの解説

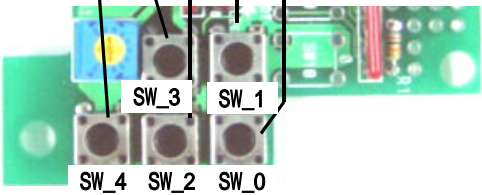
### 5.4.1 ヘッドファイルの取り込み

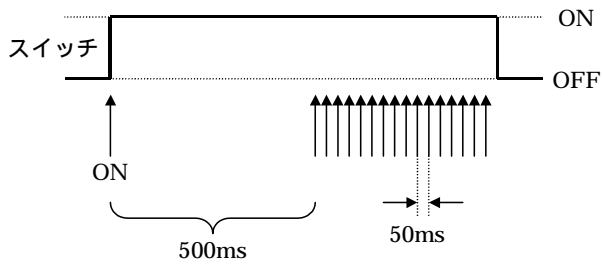
9	: #include	<no_float.h>	/* stdio の簡略化 最初に置く */
10	: #include	<stdio.h>	
11	: #include	<machine.h>	
12	: #include	"h8_3048.h"	
13	: #include	"lcd2.h"	/* LCD 表示用追加          */
14	: #include	"switch.h"	/* スイッチ追加          */

「switch.h」はスイッチを制御するためのプログラム「switch.c」ファイルを使うために、インクルードしなければならないヘッダファイルです。

### 5.4.2 switch.c ファイルで使用できる関数

「switch.c」ファイルを追加したことにより使用できる関数は、下記の表のとおりです。

関数名	内容																		
initSwitch	void initSwitch( void ) スイッチを初期化します。必ず最初に実行します。 例) initSwitch();																		
switchProcess	void switchProcess( void ) スイッチの制御を行います。1ms ごとにこの関数を実行するようにします。 例) switchProcess();																		
getSwNow	<p>unsigned char getSwNow( void ) スイッチの現在値を取得します。 引 数:なし 戻り値:現在押されている状態</p> <p>getSwNow の戻り値は、符号無し 8bit 幅の値(unsigned char)です。</p> <div style="text-align: center;"> <table style="border-collapse: collapse; margin: 0 auto;"> <tr> <td style="padding-right: 5px;">7</td><td style="padding-right: 5px;">6</td><td style="padding-right: 5px;">5</td><td style="padding-right: 5px;">4</td><td style="padding-right: 5px;">3</td><td style="padding-right: 5px;">2</td><td style="padding-right: 5px;">1</td><td style="padding-right: 5px;">0</td><td style="padding-right: 5px;">bit</td> </tr> <tr> <td style="border: 1px solid black; width: 20px; text-align: center;">0</td><td style="border: 1px solid black; width: 20px; text-align: center;">0</td><td style="border: 1px solid black; width: 20px; text-align: center;">0</td><td style="border: 1px solid black; width: 20px; text-align: center;"></td><td style="border: 1px solid black; width: 20px; text-align: center;"></td><td style="border: 1px solid black; width: 20px; text-align: center;"></td><td style="border: 1px solid black; width: 20px; text-align: center;"></td><td style="border: 1px solid black; width: 20px; text-align: center;"></td><td style="border: 1px solid black; width: 20px; text-align: center;"></td> </tr> </table> <p style="margin-top: 5px;">↑ ↑ ↑ ↑ ↑</p>  </div> <p>SW_0 ~ SW_4 が bit0 ~ 4 に対応しています。 SW_1 と SW_3 が押されている場合、 getSwNow() = 0x0a となります。 switch.h 内で、 #define SW_0 0x01 #define SW_1 0x02 #define SW_2 0x04 #define SW_3 0x08 #define SW_4 0x10 と宣言しており、getSwNow 関数と SW_x の AND を取ることにより、そのスイッチが押されているかどうか判断できます。</p> <p>例) <pre>if( getSwNow() &amp; SW_0 ) {     SW_0 が"1"なら } else {     SW_0 が"0"なら }</pre></p>	7	6	5	4	3	2	1	0	bit	0	0	0						
7	6	5	4	3	2	1	0	bit											
0	0	0																	

getSwFlag	<p>unsigned char getSwFlag( unsigned char flag )                  スイッチのキーリピート処理後の値を取得します。                  引 数:取得するキー-SW_0 ~ SW_4                  戻り値:キーリピート処理後の値 0=OFF、0 以外=ON</p> <p>時計などの時刻を設定するとき、押した瞬間 + 1 だけして、そのまま押し続けると連続して値がプラスされていく仕組みがあります。この関数はその機能を実現します。                  getSwFlag 関数では、次のような動作になります。</p>  <p>スイッチが押された瞬間、ON になります。その後 0.5 秒押し続けるとリピート機能が働いて、50ms ごとに ON 信号が送られてきます。1 秒で 20 カウントします。</p> <pre>                 例) if( getSwFlag(SW_1) ) {                     SW_1 が ON なら                 } else {                     SW_1 が OFF なら                 }             </pre>
-----------	--

### 5.4.3 初期化

76 :	/* マイコン機能の初期化 */		
77 :	init();	/* 初期化	*/
78 :	set_ccr( 0x00 );	/* 全体割り込み許可	*/
79 :	initLcd();	/* LCD 初期化	*/
80 :	initSwitch();	/* スイッチ初期化	*/

割り込みが許可されている状態('set\_ccr(0x00)以降)で、スイッチの初期化関数を呼び出します。



#### 5.4.4 プログラムでの使用

```

86 :     j = 0;
87 :     while( 1 ) {
88 :         /* スイッチ処理 */
89 :         if( getSwFlag(SW_0) ) {
90 :             j--;
91 :             if( j < -10000 ) j = -10000;
92 :         }
93 :         if( getSwFlag(SW_1) ) {
94 :             j++;
95 :             if( j > 10000 ) j = 10000;
96 :         }
97 :         if( getSwFlag(SW_2) ) {
98 :             j -= 10;
99 :             if( j < -10000 ) j = -10000;
100 :        }
101 :        if( getSwFlag(SW_3) ) {
102 :            j += 10;
103 :            if( j > 10000 ) j = 10000;
104 :        }
105 :        if( getSwFlag(SW_4) ) {
106 :            j = 0;
107 :        }
108 :        /* LCD 処理 */
109 :        lcdPosition( 0, 0 );
110 :        lcdPrintf( "getSwNow() = %02x", getSwNow() );
111 :        lcdPosition( 0, 1 );
112 :        lcdPrintf( "data = %+06d", j );
113 :    }

```

89行から107行でSW\_0~4を押したとき、変数jを操作します。キーリピートを使っています。

スイッチ	内容
SW_0	jを-1します。
SW_1	jを+1します。
SW_2	jを-10します。
SW_3	jを+10します。
SW_4	jを0にします。

109~112行で表示文字列を作っています。ここではgetSwNow関数で現在のスイッチ値と、j変数の値を表示しています。

#### 5.4.5 スイッチ処理

```

567 :     /* 拡張スイッチ用関数です。1ms 毎に実行します。 */
568 :     switchProcess();

```

1msごとにスイッチ処理関数を呼び出します。この関数で、キーリピート処理を行っています。

## 6. プロジェクト「tr\_03」 ブザーの使い方 1

### 6.1 内容

スイッチを押して圧電ブザーを鳴らします。5つのスイッチがあるので、「ド・レ・ミ・ファ・ソ」の5つの音を鳴らすことができます。音階自体は、プログラムでド～シまで自由に鳴らすことができます。

### 6.2 プロジェクトの構成

	<ul style="list-style-type: none"> <li>・tr_03start.src ベクタアドレスの設定、スタートアップルーチンが記述されています。</li> <li>・car_printf2.c セクションの初期化、printf 文、scanf 文を実行します。</li> <li>・lcd2.c LCD 制御を行います。</li> <li>・switch.c スイッチ制御を行います。</li> <li>・beep.c <b>ブザー制御を行うために追加します。</b></li> <li>・tr_03.c メインプログラムです。</li> </ul>
--	--

### 6.3 プログラム「tr\_03.c」

プログラムのゴシック体部分が追加した部分です。

```

1 : /*-----*/
2 : /* トレーニングボードを使用したマイコンカートレースプログラム(kit06版) */
3 : /* 2006.08 ジャパンマイコンカーラリー実行委員会 */
4 : /*-----*/
5 :
6 : /*=====*/
7 : /* インクルード */
8 : /*=====*/
9 : #include <no_float.h> /* stdioの簡略化 最初に置く*/
10 : #include <stdio.h>
11 : #include <machine.h>
12 : #include "h8_3048.h"
13 : #include "lcd2.h" /* LCD表示用追加 */
14 : #include "switch.h" /* スイッチ追加 */
15 : #include "beep.h" /* ブザー追加 */

```

中略

```

70 : /*-----*/
71 : /* メインプログラム */
72 : /*-----*/
73 : void main( void )
74 : {
75 :     int i;
76 :
77 :     /* マイコン機能の初期化 */
78 :     init(); /* 初期化 */
79 :     set_ccr( 0x00 ); /* 全体割り込み許可 */
80 :     initLcd(); /* LCD初期化 */
81 :     initSwitch(); /* スイッチ初期化 */
82 :     initBeep(); /* ブザー初期化 */
83 :
84 :     /* マイコンカーの状態初期化 */
85 :     handle( 0 );
86 :     speed( 0, 0 );
87 :
88 :     while( 1 ) {
89 :         /* LCD処理 */

```

```

90 :         lcdPosition( 0, 0 );
91 :         lcdPrintf( "beep sample" );
92 :
93 :         /* スイッチ処理 */
94 :         if( getSwNow() & SW_4 ) {
95 :             beepOut( M_DO );
96 :             lcdPosition( 0, 1 );
97 :             lcdPrintf( "beep out = do " );
98 :         } else if( getSwNow() & SW_3 ) {
99 :             beepOut( M_RE );
100 :            lcdPosition( 0, 1 );
101 :            lcdPrintf( "beep out = re " );
102 :        } else if( getSwNow() & SW_2 ) {
103 :            beepOut( M_MI );
104 :            lcdPosition( 0, 1 );
105 :            lcdPrintf( "beep out = mi " );
106 :        } else if( getSwNow() & SW_1 ) {
107 :            beepOut( M_FA );
108 :            lcdPosition( 0, 1 );
109 :            lcdPrintf( "beep out = fa " );
110 :        } else if( getSwNow() & SW_0 ) {
111 :            beepOut( M_SO );
112 :            lcdPosition( 0, 1 );
113 :            lcdPrintf( "beep out = so " );
114 :        } else {
115 :            beepOut( 0 );
116 :            lcdPosition( 0, 1 );
117 :            lcdPrintf( "beep out = none" );
118 :        }
119 :    }

```

中略

```

561 : /****** */
562 : /* ITU0 割り込み処理 */
563 : /****** */
564 : #pragma interrupt( interrupt_timer0 )
565 : void interrupt_timer0( void )
566 : {
567 :     ITU0_TSR &= 0xfe;          /* フラグクリア */
568 :     cnt0++;
569 :     cnt1++;
570 :
571 :     /* LCD表示処理用関数です。1ms毎に実行します。 */
572 :     lcdShowProcess();
573 :     /* 拡張スイッチ用関数です。1ms毎に実行します。 */
574 :     switchProcess();
575 :     /* ブザー処理用関数です。1ms毎に実行します。 */
576 :     beepProcess();
577 : }

```

以下、略

## 6.4 プログラム「tr\_03start.src」

**ブザー処理は、「tr05\_03start.src」ファイルにも追記、変更が必要です。忘れがちなので気をつけてください。**

```

1 : ;=====
2 : ; 定義
3 : ;=====
4 : RESERVE: .EQU    H'FFFFFFF      ; 未使用領域のアドレス
5 :
6 : ;=====
7 : ; 外部参照
8 : ;=====
9 :     .IMPORT _main
10 :     .IMPORT _INIT$CT
11 :     .IMPORT _interrupt_timer0
12 :     .IMPORT _int$TXIO
13 :
14 : ;=====
15 : ; ヘクタセクション
16 : ;=====
17 :     .SECTION V
18 :     .DATA.L RESET_START      ; 0 H'000000   リセット
19 :     .DATA.L RESERVE          ; 1 H'000004   システム予約
20 :     .DATA.L RESERVE          ; 2 H'000008   システム予約
21 :     .DATA.L RESERVE          ; 3 H'00000c   システム予約
22 :     .DATA.L RESERVE          ; 4 H'000010   システム予約
23 :     .DATA.L RESERVE          ; 5 H'000014   システム予約
24 :     .DATA.L RESERVE          ; 6 H'000018   システム予約
25 :     .DATA.L RESERVE          ; 7 H'00001c   外部割り込み NMI
26 :     .DATA.L RESERVE          ; 8 H'000020   トラップ 命令
27 :     .DATA.L RESERVE          ; 9 H'000024   トラップ 命令
28 :     .DATA.L RESERVE          ; 10 H'000028  トラップ 命令

```

トレーニングボード 実習マニュアル(kit06 版)

```

29 : .DATA.L RESERVE ; 11 H'00002c トラップ 命令
30 : .DATA.L RESERVE ; 12 H'000030 外部割り込み IRQ0
31 : .DATA.L RESERVE ; 13 H'000034 外部割り込み IRQ1
32 : .DATA.L RESERVE ; 14 H'000038 外部割り込み IRQ2
33 : .DATA.L RESERVE ; 15 H'00003c 外部割り込み IRQ3
34 : .DATA.L RESERVE ; 16 H'000040 外部割り込み IRQ4
35 : .DATA.L RESERVE ; 17 H'000044 外部割り込み IRQ5
36 : .DATA.L RESERVE ; 18 H'000048 システム予約
37 : .DATA.L RESERVE ; 19 H'00004c システム予約
38 : .DATA.L RESERVE ; 20 H'000050 WDT MOVI
39 : .DATA.L RESERVE ; 21 H'000054 REF CMI
40 : .DATA.L RESERVE ; 22 H'000058 システム予約
41 : .DATA.L RESERVE ; 23 H'00005c システム予約
42 : .DATA.L interrupt_timer0 ; 24 h'000060 ITU0 IMIA0
43 : .DATA.L RESERVE ; 25 H'000064 ITU0 IMIB0
44 : .DATA.L RESERVE ; 26 H'000068 ITU0 OV10
45 : .DATA.L RESERVE ; 27 H'00006c システム予約
46 : .DATA.L RESERVE ; 28 H'000070 ITU1 IMIA1
47 : .DATA.L RESERVE ; 29 H'000074 ITU1 IMIB1
48 : .DATA.L RESERVE ; 30 H'000078 ITU1 OV11
49 : .DATA.L RESERVE ; 31 H'00007c システム予約
50 : .DATA.L RESERVE ; 32 H'000080 ITU2 IMIA2
51 : .DATA.L RESERVE ; 33 H'000084 ITU2 IMIB2
52 : .DATA.L RESERVE ; 34 H'000088 ITU2 OV12
53 : .DATA.L RESERVE ; 35 H'00008c システム予約
54 : .DATA.L RESERVE ; 36 H'000090 ITU3 IMIA3
55 : .DATA.L RESERVE ; 37 H'000094 ITU3 IMIB3
56 : .DATA.L RESERVE ; 38 H'000098 ITU3 OV13
57 : .DATA.L RESERVE ; 39 H'00009c システム予約
58 : .DATA.L RESERVE ; 40 H'0000a0 ITU4 IMIA4
59 : .DATA.L RESERVE ; 41 H'0000a4 ITU4 IMIB4
60 : .DATA.L RESERVE ; 42 H'0000a8 ITU4 OV14
61 : .DATA.L RESERVE ; 43 H'0000ac システム予約
62 : .DATA.L RESERVE ; 44 H'0000b0 DMAC DEND0A
63 : .DATA.L RESERVE ; 45 H'0000b4 DMAC DEND0B
64 : .DATA.L RESERVE ; 46 H'0000b8 DMAC DEND1A
65 : .DATA.L RESERVE ; 47 H'0000bc DMCA DEND1B
66 : .DATA.L RESERVE ; 48 H'0000c0 システム予約
67 : .DATA.L RESERVE ; 49 H'0000c4 システム予約
68 : .DATA.L RESERVE ; 50 H'0000c8 システム予約
69 : .DATA.L RESERVE ; 51 H'0000cc システム予約
70 : .DATA.L RESERVE ; 52 H'0000d0 SC10 ER10
71 : .DATA.L RESERVE ; 53 H'0000d4 SC10 RX10
72 : .DATA.L intTXIO ; 54 h'0000d8 SC10 TX10
73 : .DATA.L RESERVE ; 55 H'0000dc SC10 TE10
74 : .DATA.L RESERVE ; 56 H'0000e0 SC11 ER11
75 : .DATA.L RESERVE ; 57 H'0000e4 SC11 RX11
76 : .DATA.L RESERVE ; 58 H'0000e8 SC11 TX11
77 : .DATA.L RESERVE ; 59 H'0000ec SC11 TE11
78 : .DATA.L RESERVE ; 60 H'0000f0 A/D ADI
79 :
80 : ;=====
81 : ; スタートアッププログラム
82 : ;=====
83 : .SECTION P
84 : RESET_START:
85 : MOV.L #H'FFF10,ER7 ; スタックの設定
86 : JSR @_INITSCT ; RAMエリアの初期化
87 : JSR @_main ; C言語のmain()関数へジャンプ
88 : OWARI:
89 : BRA OWARI
90 :
91 : .END

```

## 6.5 プログラムの解説

### 6.5.1 ヘッダファイルの取り込み

```

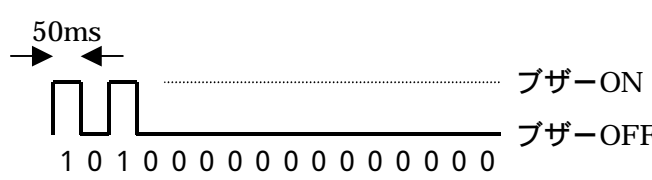
 9 : #include <no_float.h>          /* stdio の簡略化 最初に置く */
10 : #include <stdio.h>
11 : #include <machine.h>
12 : #include "h8_3048.h"
13 : #include "lcd2.h"              /* LCD 表示用追加          */
14 : #include "switch.h"           /* スイッチ追加            */
15 : #include "beep.h"              /* ブザー追加             */

```

「beep.h」はブザーを制御するためのプログラム「beep.c」ファイルを使うために、インクルードしなければならないヘッダファイルです。

### 6.5.2 beep.c ファイルで使用できる関数

「beep.c」ファイルを追加したことにより使用できる関数は、下記の表のとおりです。

関数名	内容
initBeep	void initBeep( void ) ブザーを初期化します。必ず最初に実行します。 例) initBeep();
beepOut	void beepOut( int f ) 引数に設定した周波数を出力します。引数に 0 を設定すると音は鳴りやみます。 引 数:周波数 戻り値:なし 例) beepOut( 1000 ); 1000Hz の方形波をブザーへ出力します。
setBeepPattern	void setBeepPattern( unsigned int data ) ブザーを鳴らすパターンをセットします。 引 数:鳴らすパターン 戻り値:なし  setBeepPattern 関数でブザーを鳴らすパターンをセットします。16ビット分指定します。1ビットあたり 50 ミリ秒の長さの音を鳴らします。例えば 16 進数で「0xa000」を設定したとします。2進数に直すと「1010 0000 0000 0000」となり、これは 50ms ブザー-ON、50ms ブザー-OFF、50ms ブザー-ON、残りブザー-OFF という設定です。耳には、「ピッピッ」と聞こえます。   もし、「0x8000」にすると、50ms ブザーが ON、後は全て OFF、「0xffff」なら、50ms × 16 = 800ms 間ブザーが鳴り続けます。  例) setBeepPattern( 0xa800 ); ブザーがピッピッピッピと鳴る。

beepProcess	void beepProcess( void ) ブザー処理を行います。1ms ごとにこの関数を実行するようにします。 例) beepProcess();
-------------	---

### 6.5.3 初期化

77 :	/* マイコン機能の初期化 */		
78 :	init();	/* 初期化	*/
79 :	set_ccr( 0x00 );	/* 全体割り込み許可	*/
80 :	initLcd();	/* LCD 初期化	*/
81 :	initSwitch();	/* スイッチ初期化	*/
82 :	initBeep();	/* <b>ブザー初期化</b>	*/

割り込みが許可されている状態(「set\_ccr(0x00)」以降)で、ブザーの初期化関数を呼び出します。

### 6.5.4 プログラムでの使用

88 :	while( 1 ) {
89 :	/* LCD 処理 */
90 :	lcdPosition( 0, 0 );
91 :	lcdPrintf( "beep sample" );
92 :	
93 :	/* スイッチ処理 */
94 :	if( getSwNow() & SW_4 ) {
95 :	beepOut( M_DO );
96 :	lcdPosition( 0, 1 );
97 :	lcdPrintf( "beep out = do " );
98 :	} else if( getSwNow() & SW_3 ) {
99 :	beepOut( M_RE );
100 :	lcdPosition( 0, 1 );
101 :	lcdPrintf( "beep out = re " );
102 :	} else if( getSwNow() & SW_2 ) {
103 :	beepOut( M_MI );
104 :	lcdPosition( 0, 1 );
105 :	lcdPrintf( "beep out = mi " );
106 :	} else if( getSwNow() & SW_1 ) {
107 :	beepOut( M_FA );
108 :	lcdPosition( 0, 1 );
109 :	lcdPrintf( "beep out = fa " );
110 :	} else if( getSwNow() & SW_0 ) {
111 :	beepOut( M_SO );
112 :	lcdPosition( 0, 1 );
113 :	lcdPrintf( "beep out = so " );
114 :	} else {
115 :	beepOut( 0 );
116 :	lcdPosition( 0, 1 );
117 :	lcdPrintf( "beep out = none" );
118 :	}
119 :	}

現在押されているスイッチを取得する getSwNow 関数を呼び出し、どのスイッチが押されているか if 文で判別します。押されているキーによって、「ド、レ、ミ、ファ、ソ」の音階を鳴らします。何も押されていない場合は、音を消します。同時に、鳴らしている音階を LCD に表示するようにしています。

スイッチ	音階
SW_4	ド
SW_3	レ
SW_2	ミ
SW_1	ファ
SW_0	ソ
何も押されていない場合	音を停止

### 6.5.5 ブザー処理

```
575 :      /* ブザー処理用関数です。1ms 毎に実行します。          */
576 :      beepProcess();
```

1ms ごとにブザー処理関数を呼び出します。

### 6.5.6 スタートアップルーチン「tr\_03start.src」の追加、変更

ブザー処理では、ベクタアドレス 54 番の割り込みを使用します。割り込み関数は intTXIO 関数です。そのため、スタートアップルーチンに intTXIO 関数を呼び出すよう追加する必要があります。

```
6 : ;=====
7 : ; 外部参照
8 : ;=====
9 :      .IMPORT _main
10 :      .IMPORT _INITSTC
11 :      .IMPORT _interrupt_timer0
12 :      .IMPORT _intTXIO
```

「.IMPORT」宣言部分に「\_intTXIO」を追加します。「intTXIO 関数が tr\_03start.src ファイルとは別のファイルにあるので、外部を参照してください」ということを宣言しています。ちなみに、intTXIO 関数は、beep.c ファイル内にあります。

ベクタ番号 54 の SCIO TXIO 割り込みの宣言を「RESERVE」から「\_intTXIO」に変更します。

```
72 :      .DATA.L RESERVE          ; 54 h'0000d8  SCIO TXIO
```

```
72 :      .DATA.L _intTXIO        ; 54 h'0000d8  SCIO TXIO
```

12 行を追加、72 行を変更すれば、ブザーを使用することができます。

## 音階について

音階とは、「ドレミファソラシド」のことです。この音階の周波数が分かれば、周期が分かるので、その周期のデューティ比 50%の PWM を圧電スピーカに出力すれば、「ドレミファソラシド」と音を鳴らすことができます。

音階は、「ド」から次の高い「ド」まで「ド、ド#、レ、レ#、ミ、ファ、ファ#、ソ、ソ#、ラ、ラ#、シ」の 12 段階あります。最初のドの周波数は、261.6[Hz]です。次に高いドの周波数は、2 倍の 523.2[Hz]となります。この間の周波数は、

$261.6 \times 2$  の  $(x / 12)$  乗

で求められます。x は、ドが 0、ド# が 1、レ が 2、レ# が 3、ミ が 4、ファ が 5、ファ# が 6、ソ が 7、ソ# が 8、ラ が 9、ラ# が 10、シ が 11 というように一つずつ増えていきます。

音	x	計算	周波数[Hz]
ド	0	$261.6 \times 2^{(0/12)}$	261.6
ド#	1	$261.6 \times 2^{(1/12)}$	277.2
レ	2	$261.6 \times 2^{(2/12)}$	293.6
レ#	3	$261.6 \times 2^{(3/12)}$	311.1
ミ	4	$261.6 \times 2^{(4/12)}$	329.6
ファ	5	$261.6 \times 2^{(5/12)}$	349.2
ファ#	6	$261.6 \times 2^{(6/12)}$	370.0
ソ	7	$261.6 \times 2^{(7/12)}$	392.0
ソ#	8	$261.6 \times 2^{(8/12)}$	415.3
ラ	9	$261.6 \times 2^{(9/12)}$	440.0
ラ#	10	$261.6 \times 2^{(10/12)}$	466.1
シ	11	$261.6 \times 2^{(11/12)}$	493.8
ド	12	$261.6 \times 2^{(12/12)}$	523.2

beep.h に周波数を四捨五入して、下記のように宣言しています。

```

4 : #define M_DO 262 /* ド */
5 : #define M_DO# 277 /* ド# */
6 : #define M_RE 294 /* レ */
7 : #define M_RE# 311 /* レ# */
8 : #define M_MI 330 /* ミ */
9 : #define M_FA 349 /* ファ */
10 : #define M_FA# 370 /* ファ# */
11 : #define M_SO 392 /* ソ */
12 : #define M_SO# 415 /* ソ# */
13 : #define M_RA 440 /* ラ */
14 : #define M_RA# 466 /* ラ# */
15 : #define M_SI 494 /* シ */
16 : #define H1_DO 523 /* ド
    
```

先頭に付いている「M」は、「Middle(真ん中の)」という意味です。

```
beepOut( M_DO );
```

で、ドの音階がブザーから出力されます。

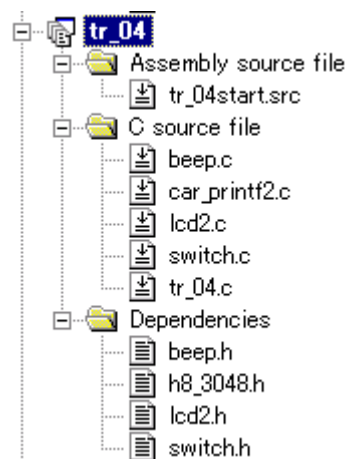


## 7. プロジェクト「tr\_04」 ブザーの使い方2

### 7.1 内容

スイッチを押して圧電ブザーを鳴らします。押すスイッチによってブザーの鳴り方が変わります。

### 7.2 プロジェクトの構成



- tr\_04start.src  
ベクタアドレスの設定、スタートアップルーチンが記述されています。
- car\_printf2.c  
セクションの初期化、printf 文、scanf 文を実行します。
- lcd2.c  
LCD 制御を行います。
- switch.c  
スイッチ制御を行います。
- beep.c  
ブザー制御を行います。
- tr\_04.c  
メインプログラムです。

### 7.3 プログラム「tr\_04.c」

プログラムのゴシック体部分が追加した部分です。

前略

```

70 :  /****** */
71 :  /* メインプログラム */
72 :  /****** */
73 :  void main( void )
74 :  {
75 :      int    i;
76 :
77 :      /* マイコン機能の初期化 */
78 :      init();                /* 初期化 */
79 :      set_ccr( 0x00 );      /* 全体割り込み許可 */
80 :      initLcd();           /* LCD初期化 */
81 :      initSwitch();        /* スイッチ初期化 */
82 :      initBeep();          /* ブザー初期化 */
83 :
84 :      /* マイコンカーの状態初期化 */
85 :      handle( 0 );
86 :      speed( 0, 0 );
87 :
88 :      while( 1 ) {
89 :          /* LCD処理 */
90 :          lcdPosition( 0, 0 );
91 :          lcdPrintf( "beep sample 2" );
92 :
93 :          if( getSwFlag(SW_4) ) {
94 :              setBeepPattern( 0x8000 );
95 :              lcdPosition( 0, 1 );
96 :              lcdPrintf( "SW4 ON" );
97 :          }
98 :          if( getSwFlag(SW_3) ) {
99 :              setBeepPattern( 0xc000 );
100 :              lcdPosition( 0, 1 );
101 :              lcdPrintf( "SW3 ON" );
102 :          }
103 :          if( getSwFlag(SW_2) ) {
104 :              setBeepPattern( 0xf000 );
105 :              lcdPosition( 0, 1 );
106 :              lcdPrintf( "SW2 ON" );
107 :          }
108 :          if( getSwFlag(SW_1) ) {
109 :              setBeepPattern( 0xa000 );
110 :              lcdPosition( 0, 1 );
111 :              lcdPrintf( "SW1 ON" );
112 :          }
113 :          if( getSwFlag(SW_0) ) {
114 :              setBeepPattern( 0xaaaa );
115 :              lcdPosition( 0, 1 );
116 :              lcdPrintf( "SW0 ON" );
117 :          }
118 :      }

```

以下、略

## 7.4 プログラムの解説

プロジェクト「tr\_03」の「tr\_03.c」では、beepOut 関数を使用しました。ここでは、もう一つの音の鳴らし方として setBeepPattern 関数を使用してみます。この関数は、設定したパターンの音を鳴らします。

### 7.4.1 プログラムでの使用

```

88 :     while( 1 ) {
89 :         /* LCD 処理 */
90 :         lcdPosition( 0, 0 );
91 :         lcdPrintf( "beep sample 2" );
92 :
93 :         if( getSwFlag(SW_4) ) {
94 :             setBeepPattern( 0x8000 );
95 :             lcdPosition( 0, 1 );
96 :             lcdPrintf( "SW4 ON" );
97 :         }
98 :         if( getSwFlag(SW_3) ) {
99 :             setBeepPattern( 0xc000 );
100 :            lcdPosition( 0, 1 );
101 :            lcdPrintf( "SW3 ON" );
102 :        }
103 :        if( getSwFlag(SW_2) ) {
104 :            setBeepPattern( 0xf000 );
105 :            lcdPosition( 0, 1 );
106 :            lcdPrintf( "SW2 ON" );
107 :        }
108 :        if( getSwFlag(SW_1) ) {
109 :            setBeepPattern( 0xa000 );
110 :            lcdPosition( 0, 1 );
111 :            lcdPrintf( "SW1 ON" );
112 :        }
113 :        if( getSwFlag(SW_0) ) {
114 :            setBeepPattern( 0xaaaa );
115 :            lcdPosition( 0, 1 );
116 :            lcdPrintf( "SW0 ON" );
117 :        }
118 :    }

```

押されたスイッチにより、音を鳴らすパターンを変えます。音を鳴らしている最中は、再度 setBeepPattern 関数が呼ばれても無視されます。

スイッチ	パターン	パターン 2進数	鳴り方
SW_4	0x8000	1000 0000 0000 0000	ピッ
SW_3	0xc000	1100 0000 0000 0000	ピーツ
SW_2	0xf000	1111 0000 0000 0000	ピーーツ
SW_1	0xa000	1010 0000 0000 0000	ピッピッ
SW_0	0xaaaa	1010 1010 1010 1010	ピッピッピッピッピッピッピッピッ(8回)

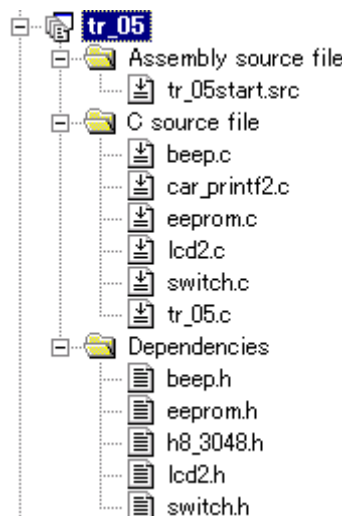
音階を鳴らすか、音のパターンを鳴らすか、使用する関数で選ぶことができます。使用する状況に応じて選んでください。

## 8. プロジェクト「tr\_05」 EEP-ROM の使い方

### 8.1 内容

5つのスイッチを押すことにより、LCD に表示している数字を+1、-1、+10、-10 させることができます。プロジェクト「tr\_02」の「tr\_02.c」と似ていますが、押すと数値がクリアされたボタンが「データの保存」ボタンに変わっています。データをEEP-ROM に保存することにより電源を切っても、電源を入れ直すと保存ボタンを押したときの数値が表示されます。

### 8.2 プロジェクトの構成

	<ul style="list-style-type: none"> <li>・tr_05start.src ベクタアドレスの設定、スタートアップルーチンが記述されています。</li> <li>・car_printf2.c セクションの初期化、printf 文、scanf 文を実行します。</li> <li>・lcd2.c LCD 制御を行います。</li> <li>・switch.c スイッチ制御を行います。</li> <li>・beep.c ブザー制御を行います。</li> <li>・eeprom.c <b>EEP-ROM 制御を行うために追加します。</b></li> <li>・tr_05.c メインプログラムです。</li> </ul>
--	--

### 8.3 プログラム「tr\_05.c」

プログラムのゴシック体部分が追加した部分です。

```

1 : /*-----*/
2 : /* トレーニングボードを使用したマイコンカートレースプログラム(kit06版) */
3 : /* 2006.08 ジャパンマイコンカーラリー実行委員会 */
4 : /*-----*/
5 :
6 : /*-----*/
7 : /* インクルード */
8 : /*-----*/
9 : #include <no_float.h> /* stdioの簡略化 最初に置く*/
10 : #include <stdio.h>
11 : #include <machine.h>
12 : #include "h8_3048.h"
13 : #include "lcd2.h" /* LCD表示用追加 */
14 : #include "switch.h" /* スイッチ追加 */
15 : #include "beep.h" /* ブザー追加 */
16 : #include "eeprom.h" /* EEP-ROM追加 */

```

中略

```

71 : /*-----*/
72 : /* メインプログラム */
73 : /*-----*/
74 : void main( void )
75 : {
76 :     int i, j;
77 :
78 :     /* マイコン機能の初期化 */
79 :     init(); /* 初期化 */
80 :     set_ccr( 0x00 ); /* 全体割り込み許可 */

```

```

81 :   initLcd();           /* LCD初期化          */
82 :   initSwitch();      /* スイッチ初期化    */
83 :   initBeep();        /* ブザー初期化      */
84 :   initEeprom();      /* EEPROM初期化      */
85 :
86 :   /*EEP-ROMのチェック */
87 :   if( readEeprom( 0x00 ) != 0x1234 ) {
88 :     /*
89 :     IDのチェック EEPROMを初めて使うかどうか
90 :     0x00番地にIDが書かれていなければ初めて使うと判断して初期化する
91 :     */
92 :     writeEeprom( 0x00, 0x1234 );
93 :     while( !checkEeprom() ); /* 書き込み終了チェック */
94 :     writeEeprom( 0x01, 0 );
95 :     while( !checkEeprom() ); /* 書き込み終了チェック */
96 :     j = 0;
97 :   } else {
98 :     /* 2回目以降の使用の場合 */
99 :     j = readEeprom( 0x01 );
100 :   }
101 :
102 :   /* マイコンカーの状態初期化 */
103 :   handle( 0 );
104 :   speed( 0, 0 );
105 :
106 :   while( 1 ) {
107 :     /* スイッチ処理 */
108 :     if( getSwFlag(SW_0) ) {
109 :       j--;
110 :       if( j < -10000 ) j = -10000;
111 :     }
112 :     if( getSwFlag(SW_1) ) {
113 :       j++;
114 :       if( j > 10000 ) j = 10000;
115 :     }
116 :     if( getSwFlag(SW_2) ) {
117 :       j -= 10;
118 :       if( j < -10000 ) j = -10000;
119 :     }
120 :     if( getSwFlag(SW_3) ) {
121 :       j += 10;
122 :       if( j > 10000 ) j = 10000;
123 :     }
124 :     if( getSwFlag(SW_4) ) {
125 :       writeEeprom( 0x01, j ); /* EEPROMにデータ書き込み */
126 :       while( !checkEeprom() ); /* 書き込み終了チェック */
127 :       setBeepPattern( 0xa000 );
128 :     }
129 :     /* LCD処理 */
130 :     lcdPosition( 0, 0 );
131 :     lcdPrintf( "getSwNow() = %02x ", getSwNow() );
132 :     lcdPosition( 0, 1 );
133 :     lcdPrintf( "data = %+06d", j );
134 :   }

```

以下、略

## 8.4 プログラムの解説

### 8.4.1 ヘッドファイルの取り込み

```

9 : #include <no_float.h>           /* stdio の簡略化 最初に置く */
10 : #include <stdio.h>
11 : #include <machine.h>
12 : #include "h8_3048.h"
13 : #include "lcd2.h"                /* LCD 表示用追加          */
14 : #include "switch.h"             /* スイッチ追加            */
15 : #include "beep.h"               /* ブザー追加              */
16 : #include "eeprom.h"             /* EEPROM追加               */

```

「eeprom.h」はEEP-ROMを制御するためのプログラム「eeprom.c」ファイルを使うために、インクルードしなければならないヘッダファイルです。

## 8.4.2 eeprom.c ファイルで使用できる関数

「eeprom.c」ファイルを追加したことにより使用できる関数は、下記の表のとおりです。

関数名	内容
initEeprom	void initEeprom( void ) EEP-ROM を初期化します。必ず最初に実行します。 例) initEeprom();
readEeprom	int readEeprom( unsigned char address ) EEP-ROM からデータを読み込みます。 引 数: アドレス 0 ~ 127 戻り値: 値 例) i = readEeprom( 0x05 ); 0x05 番地のデータを読み込み、変数 i に代入します。
writeEeprom	void writeEeprom( unsigned char address, int write ) EEP-ROM にデータを書き込みます。 引 数: アドレス 0 ~ 127、値 -32,768 ~ 32,767 戻り値: なし 例) writeEeprom( 0x0e, 12345 ); EEP-ROM の 0x0e 番地に 12345 を書き込みます。  書き込んだ後、checkEeprom 関数で次に読み書きできるか調べます。 最大で約 10ms の時間、読み書きできません。10ms はマイコンの時間ではかなり長い時間です。
checkEeprom	int checkEeprom( void ) EEP-ROM 書き込み後、次に読み書きできるかチェックします。 引 数: なし 戻り値: 1 = 読み書き OK 、 0 = 読み書き不可 例) while( !checkEeprom() ); EEP-ROM に書き込み後は、最高で 10ms 間は読み書きができません。 この関数は、次に読み書きができるまで待ちます。書き込みだけでなく、読み込みもできません。

### 8.4.3 初期化

```

71 : /****** */
72 : /* メインプログラム */
73 : /****** */
74 : void main( void )
75 : {
76 :     int    i, j;
77 :
78 :     /* マイコン機能の初期化 */
79 :     init();           /* 初期化 */
80 :     set_ccr( 0x00 ); /* 全体割り込み許可 */
81 :     initLcd();       /* LCD 初期化 */
82 :     initSwitch();    /* スイッチ初期化 */
83 :     initBeep();      /* ブザー初期化 */
84 :     initEeprom();    /* EEP-ROM 初期化 */

```

EEP-ROM の初期化を行います。

### 8.4.4 EEP-ROM のチェック

```

86 : /*EEP-ROM のチェック */
87 : if( readEeprom( 0x00 ) != 0x1234 ) {
88 :     /*
89 :     ID のチェック EEP-ROM を初めて使うかどうか
90 :     0x00 番地に ID が書かれていなければ初めて使うと判断して初期化する
91 :     */
92 :     writeEeprom( 0x00, 0x1234 );
93 :     while( !checkEeprom() ); /* 書き込み終了チェック */
94 :     writeEeprom( 0x01, 0 );
95 :     while( !checkEeprom() ); /* 書き込み終了チェック */
96 :     j = 0;
97 : } else {
98 :     /* 2 回目以降の使用の場合 */
99 :     j = readEeprom( 0x01 );
100 : }

```

EEP-ROM に書き込む内容を決めておきます。

EEP-ROM 番地	内容
0x00	EEP-ROM チェック用 0x1234 で無ければ、0x01 番地の値は 0 とする
0x01	j の値 保存用

0x00 番地を EEP-ROM チェック用として、特定の値を書き込む領域にします。今回は 0x1234 を書き込みます。もし、0x1234 が書かれていなければ初めて EEP-ROM を使用すると判断して、0x00 番地に 0x1234 を、0x01 番地に 0 を書き込みます。次に j 変数の値を 0 にします。

チェック用の値は、0x0000 と 0xffff 以外の数字にしてください。EEP-ROM は最初これらの値になっていることがあり、その場合、初めて使うか判断できません。

### 8.4.5 プログラムでの使用

```

106 :     while( 1 ) {
107 :         /* スイッチ処理 */
108 :         if( getSwFlag(SW_0) ) {
109 :             j--;
110 :             if( j < -10000 ) j = -10000;
111 :         }
112 :         if( getSwFlag(SW_1) ) {
113 :             j++;
114 :             if( j > 10000 ) j = 10000;
115 :         }
116 :         if( getSwFlag(SW_2) ) {
117 :             j -= 10;
118 :             if( j < -10000 ) j = -10000;
119 :         }
120 :         if( getSwFlag(SW_3) ) {
121 :             j += 10;
122 :             if( j > 10000 ) j = 10000;
123 :         }
124 :         if( getSwFlag(SW_4) ) {
125 :             writeEeprom( 0x01, j );    /* EEPROM にデータ書き込み */
126 :             while( !checkEeprom() );  /* 書き込み終了チェック */
127 :             setBeepPattern( 0xa000 );
128 :         }
129 :         /* LCD 処理 */
130 :         lcdPosition( 0, 0 );
131 :         lcdPrintf( "getSwNow() = %02x ", getSwNow() );
132 :         lcdPosition( 0, 1 );
133 :         lcdPrintf( "data = %+06d", j );
134 :     }

```

SW\_0～4 を押したとき、j を操作します。キーリピートを使っています。

スイッチ	内容
SW_0	j を-1 します。
SW_1	j を+1 します。
SW_2	j を-10 します。
SW_3	j を+10 します。
SW_4	j の値を EEPROM に保存します。 保存後、ブザーを鳴らします。

電源を入れたとき、前回 j の値を保存しているなら、その保存した値が表示されます。トレーニングボードの EEPROM は、128 ワードのデータが保存できるのでかなり情報を保存することができます。128 ワードとは、16 ビット幅(-32,768～32,767)のデータを 128 個ということです。



#### 8.4.6 EEP-ROM を使うポイント

EEP-ROMからデータを読み込んだり書き込んだりするには時間がかかります。特に、書き込みは最大で10msの時間がかかります。そのため、writeEeprom関数やreadEeprom関数はプログラムの中ではあまり使用せず、最初に変数(H8のRAM)へ読み出し、最後に書き込むようにしてください。

##### 速度の遅いプログラム例

```
while( 1 ) {  
    lcdPrintf ( "data = %+06d  ", readEeprom( 0x01 ) );  
}
```

##### 推奨プログラム例

```
int j;  
  
j = readEeprom( 0x01 );  
while( 1 ) {  
    lcdPrintf ( "data = %+06d  ", j );  
}
```

## 9. プロジェクト「tr\_11」 走行プログラムの高速化対応

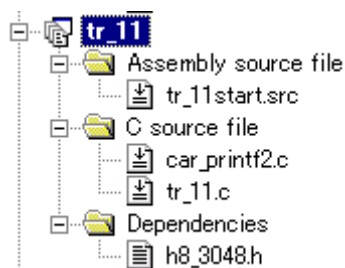
### 9.1 内容

ワークスペース「kit06」のプロジェクト「kit06」の「kit06.c」は、標準キット(モータ電圧 6V)に合わせたプログラムです。速いマイコンカーのほとんどは、モータに電池 6~8 本分の電圧を加えて高速化しています。

そこで、高速化したときに変更するポイントを解説します。ここに書いた数値は、マイコンカーによって違います。変更する場所を確認して、数値自体は各自のマイコンカーに合わせて調整してください。

今回は、トレーニングボードを追加する前準備をします。このプロジェクトではトレーニングボードは使用していません。

### 9.2 プロジェクトの構成



- ・tr\_11start.src  
ベクタアドレスの設定、スタートアップルーチンが記述されています。
- ・car\_printf2.c  
セクションの初期化、printf 文、scanf 文を実行します。
- ・tr\_11.c  
メインプログラムです。

### 9.3 プログラム「tr\_11.c」

プログラムのゴシック体部分が追加、変更した部分です。

```

1 : /******
2 : /* トレーニングボードを使用したマイコンカートレースプログラム(kit06版) */
3 : /* 2006.08 ジャパンマイコンカーラリー実行委員会 */
4 : /******
5 :
6 : /*=====*/
7 : /* インクルード */
8 : /*=====*/
9 : #include <machine.h>
10: #include "h8_3048.h"
11:
12: /*=====*/
13: /* シンボル定義 */
14: /*=====*/
15:
16: /* 定数設定 */
17: #define TIMER_CYCLE 3071 /* タイマのサイクル 1ms */
18: /* /8で使用する場合、 */
19: /* /8 = 325.5[ns] */
20: /* TIMER_CYCLE = */
21: /* 1[ms] / 325.5[ns] */
22: /* = 3072 */
23: #define PWM_CYCLE 24575 /* PWMのサイクル 8msに変更 */
24: /* PWM_CYCLE = */
25: /* 8[ms] / 325.5[ns] */
26: /* = 24576 */
27: #define SERVO_CENTER 5000 /* サーボのセンタ値 */
28: #define HANDLE_STEP 26 /* 1分の値 */

```

中略

```

141: case 11:
142: /* 通常トレース */
143: if( check_crossline() ) { /* クロスラインチェック */
144: pattern = 21;
145: break;

```

トレーニングボード 実習マニュアル(kit06 版)

```

146 :     }
147 :     if( check_rightline() ) {          /* 右ハーフラインチェック */
148 :         pattern = 51;
149 :         break;
150 :     }
151 :     if( check_leftline() ) {          /* 左ハーフラインチェック */
152 :         pattern = 61;
153 :         break;
154 :     }
155 :     switch( sensor_inp(MASK3_3) ) {
156 :     case 0x00:
157 :         /* センタ まっすぐ */
158 :         handle( 0 );
159 :         speed( 100 ,100 );
160 :         break;
161 :
162 :     case 0x04:
163 :         /* 微妙に左寄り 右へ微曲げ */
164 :         handle( 3 );
165 :         speed( 100 ,100 );
166 :         break;
167 :
168 :     case 0x06:
169 :         /* 少し左寄り 右へ小曲げ */
170 :         handle( 10 );
171 :         speed( 80 ,69 );
172 :         break;
173 :
174 :     case 0x07:
175 :         /* 中くらい左寄り 右へ中曲げ */
176 :         handle( 15 );
177 :         speed( 50 ,40 );
178 :         break;
179 :
180 :     case 0x03:
181 :         /* 大きく左寄り 右へ大曲げ */
182 :         handle( 25 );
183 :         speed( 30 ,21 );
184 :         pattern = 12;
185 :         break;
186 :
187 :     case 0x20:
188 :         /* 微妙に右寄り 左へ微曲げ */
189 :         handle( -3 );
190 :         speed( 100 ,100 );
191 :         break;
192 :
193 :     case 0x60:
194 :         /* 少し右寄り 左へ小曲げ */
195 :         handle( -10 );
196 :         speed( 69 ,80 );
197 :         break;
198 :
199 :     case 0xe0:
200 :         /* 中くらい右寄り 左へ中曲げ */
201 :         handle( -15 );
202 :         speed( 40 ,50 );
203 :         break;
204 :
205 :     case 0xc0:
206 :         /* 大きく右寄り 左へ大曲げ */
207 :         handle( -25 );
208 :         speed( 21 ,30 );
209 :         pattern = 13;
210 :         break;
211 :
212 :     default:
213 :         break;
214 :     }
215 :     break;

```

中略

```

264 :     case 22:
265 :         /* 2本目を読み飛ばす */
266 :         if( cnt1 > 50 ) {
267 :             pattern = 23;
268 :             cnt1 = 0;
269 :         }
270 :         break;

```

中略

```

317 :     case 31:
318 :         /* 左クランククリア処理 安定するまで少し待つ */
319 :         if( cnt1 > 50 ) {
320 :             pattern = 32;
321 :             cnt1 = 0;
322 :         }
323 :         break;
324 :

```

## トレーニングボード 実習マニュアル(kit06 版)

```
325 :     case 32:
326 :         /* 左クランククリア処理 曲げ終わりのチェック */
327 :         if( sensor_inp(MASK3_3) == 0x60 ) {
328 :             led_out( 0x0 );
329 :             pattern = 11;
330 :             cnt1 = 0;
331 :         }
332 :         break;
333 :
334 :     case 41:
335 :         /* 右クランククリア処理 安定するまで少し待つ */
336 :         if( cnt1 > 50 ) {
337 :             pattern = 42;
338 :             cnt1 = 0;
339 :         }
340 :         break;
341 :
342 :     case 42:
343 :         /* 右クランククリア処理 曲げ終わりのチェック */
344 :         if( sensor_inp(MASK3_3) == 0x06 ) {
345 :             led_out( 0x0 );
346 :             pattern = 11;
347 :             cnt1 = 0;
348 :         }
349 :         break;
350 :
351 :     case 51:
352 :         /* 1本目の右ハーフライン検出時の処理 */
353 :         led_out( 0x2 );
354 :         handle( 0 );
355 :         speed( 0 ,0 );
356 :         pattern = 52;
357 :         cnt1 = 0;
358 :         break;
359 :
360 :     case 52:
361 :         /* 2本目を読み飛ばす */
362 :         if( cnt1 > 50 ) {
363 :             pattern = 53;
364 :             cnt1 = 0;
365 :         }
366 :         break;
367 :
```

中略

```
413 :     case 61:
414 :         /* 1本目の左ハーフライン検出時の処理 */
415 :         led_out( 0x1 );
416 :         handle( 0 );
417 :         speed( 0 ,0 );
418 :         pattern = 62;
419 :         cnt1 = 0;
420 :         break;
421 :
422 :     case 62:
423 :         /* 2本目を読み飛ばす */
424 :         if( cnt1 > 50 ) {
425 :             pattern = 63;
426 :             cnt1 = 0;
427 :         }
428 :         break;
429 :
```

以下、略

## 9.4 プログラムの解説

### 9.4.1 モータ、サーボの PWM 周期の変更

```

23 : #define          PWM_CYCLE      24575  /* PWM のサイクル 8ms に変更 */
24 :                  /*          PWM_CYCLE =          */
25 :                  /*          8[ms] / 325.5[ns] */
26 :                  /*          = 24576      */

```

モータ、サーボの PWM 周期は、サーボの規格である 16ms に設定します。サーボは、ON 幅を検出して切れ角を決めます。要は、16ms ごとに切れ角を設定しているということになります。

秒速 4m/s で走行している場合、1 秒で 4メートル走行します。16ms 間では 64mm 進みます。64mm ごとに切れ角を設定しているということです。直線だけなら問題ありませんがカーブがある場合、64mm 進めばセンサの位置がかなりずれてしまいます。

そこで、周期を短くします。サーボの周期は規格で決まっていますが、周期を短くしても動作するサーボがあります(現在市販されているほとんどのサーボは大丈夫なようです)。今回は、半分の 8ms の周期にしてみます。

周期 8ms ÷ ITU3\_CNT がカウントする時間 325.52ns = 24576

PWM\_CYCLE には1小さい値をセットするので、24575 をセットします。

秒速 4m/s で走行している場合、8ms 間に 32mm 進みます。32mm ごとに切れ角を設定しているということになり、先ほどと比べて1 / 2の距離ごとに切れ角を設定していることになります。高性能サーボによっては、5 ~ 6ms でも動作するサーボがあるようですので、いろいろ試してみると良いでしょう。

同時にモータの周期も変わりますが、モータの周期は 1ms くらいまで小さくしても全く問題ありません。

### 9.4.2 ハンドルの切れ角

```

141 :     case 11:
142 :         /* 通常トレース */
中略
155 :         switch( sensor_inp(MASK3_3) ) {
156 :             case 0x00:
157 :                 /* センタ まっすぐ */
158 :                 handle( 0 );
159 :                 speed( 100 ,100 );
160 :                 break;
161 :
162 :             case 0x04:
163 :                 /* 微妙に左寄り 右へ微曲げ */
164 :                 handle( 3 );
165 :                 speed( 100 ,100 );
166 :                 break;
167 :
中略
187 :             case 0x20:
188 :                 /* 微妙に右寄り 左へ微曲げ */
189 :                 handle( -3 );
190 :                 speed( 100 ,100 );
191 :                 break;

```

スピードが上がると、ちょっとハンドルを曲げただけでもぶれ続ける原因となります。そこで微曲げを 5 度から 3 度へ変更します。今回は小曲げ、中曲げはそのままですが、小曲げ、中曲げも切れ角を小さくした方が良いでしょう。

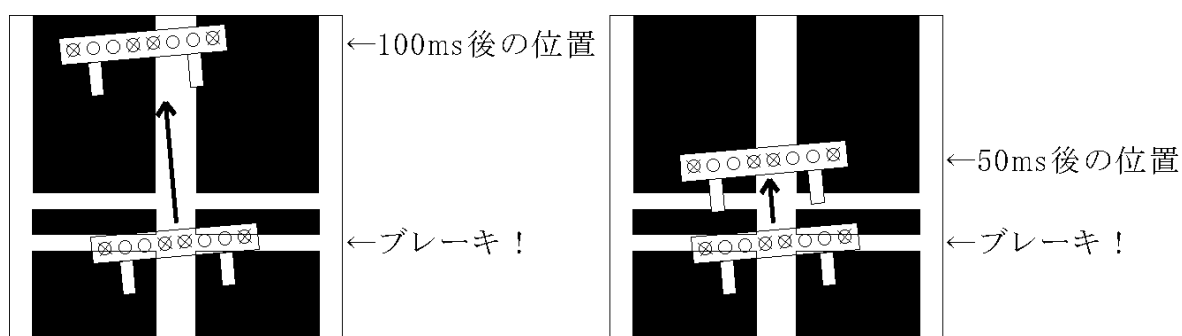
### 9.4.3 クロスライン検出後、2本目を読み飛ばすまでの時間

```

264 :     case 22:
265 :         /* 2本目を読み飛ばす */
266 :         if( cnt1 > 50 ) {
267 :             pattern = 23;
268 :             cnt1 = 0;
269 :         }
270 :         break;

```

クロスライン検出後、2本目を読み飛ばすまでの時間は 100ms でしたが 50ms に変更しています。今までは、100ms 間、センサを見ずに時間が過ぎるのを待っていました。高速化したときに進む距離が長くなり中心からずれてしまうことが多くなるためです。そこで読み飛ばす時間を減らして、中心からずれにくくしています。



### 9.4.4 クランク検出後の待ち時間

```

317 :     case 31:
318 :         /* 左クランククリア処理 安定するまで少し待つ */
319 :         if( cnt1 > 50 ) {
320 :             pattern = 32;
321 :             cnt1 = 0;
322 :         }
323 :         break;

```

中略

```

334 :     case 41:
335 :         /* 右クランククリア処理 安定するまで少し待つ */
336 :         if( cnt1 > 50 ) {
337 :             pattern = 42;
338 :             cnt1 = 0;
339 :         }
340 :         break;

```

クランクを検出すると、ハンドル、モータの回転を直角部分が曲がる状態にして少し待ちます。その後、設定したセンサ状態になるまで曲げ続けます。この少し待つ時間を 200ms から 50ms に短くしました。

これは、200ms 以内に、直角を曲がり終え中心線を超えてしまった場合、内側に落ちてしまうためです。逆に時間が短すぎても、本当は曲がり終えていない状態で、曲がり終えたセンサ状態を検出してしまい脱輪してしまいます。

#### 9.4.5 右ハーフライン検出後の待ち時間

```
360 :     case 52:
361 :         /* 2本目を読み飛ばす */
362 :         if( cnt1 > 50 ) {
363 :             pattern = 53;
364 :             cnt1 = 0;
365 :         }
366 :         break;
```

右ハーフライン検出後、2本目を読み飛ばすまでの時間は100msでしたが50msに変更しています。今までは100ms間、センサを見ずに時間が過ぎるのを待っていましたが、高速化したときに進む距離が長くなり中心からずれてしまうことが多くなるためです。そこで読み飛ばす時間を減らして、中心からずれにくくしています。

#### 9.4.6 左ハーフライン検出後の待ち時間

```
422 :     case 62:
423 :         /* 2本目を読み飛ばす */
424 :         if( cnt1 > 50 ) {
425 :             pattern = 63;
426 :             cnt1 = 0;
427 :         }
428 :         break;
```

左ハーフライン検出後、2本目を読み飛ばすまでの時間は100msでしたが50msに変更しています。今までは100ms間、センサを見ずに時間が過ぎるのを待っていましたが、高速化したときに進む距離が長くなり中心からずれてしまうことが多くなるためです。そこで読み飛ばす時間を減らして、中心からずれにくくしています。

## 10. プロジェクト「tr\_12」 サーボのセンタ調整

### 10.1 内容

ここから、実際にトレーニングボードを使ってマイコンカーのチューニングを行っていきます。

最初は、サーボのセンタ調整です。サーボのセンタの調整は微妙で、何度もプログラムのセンタ値を直して書き込むという、時間のかかる作業をしてきたと思います。これをパソコンを使わずに手元のスイッチで調整することができれば非常に便利です。そこで、トレーニングボードのスイッチでセンタ値を調整するプログラムを作ってみます。

### 10.2 プロジェクトの構成

	<ul style="list-style-type: none"> <li>・tr_12start.src ベクタアドレスの設定、スタートアップルーチンが記述されています。</li> <li>・car_printf2.c セクションの初期化、printf 文、scanf 文を実行します。</li> <li>・lcd2.c <b>LCD 制御を行うために追加します。</b></li> <li>・switch.c <b>スイッチ制御を行うために追加します。</b></li> <li>・beep.c <b>ブザー制御を行うために追加します。</b></li> <li>・eeprom.c <b>EEP-ROM 制御を行うために追加します。</b></li> <li>・tr_12.c メインプログラムです。</li> </ul>
--	---

### 10.3 プログラム「tr\_12.c」

プログラムのゴシック体部分が「tr\_11.c」から追加、変更した部分です。

```

1 : /******
2 : /* トレーニングボードを使用したマイコンカートレースプログラム(kit06版) */
3 : /*                               2006.08 ジャパンマイコンカーラリー実行委員会 */
4 : /******
5 :
6 : /*=====*/
7 : /* インクルード */
8 : /*=====*/
9 : #include <no_float.h> /* stdioの簡略化 最初に置く*/
10: #include <stdio.h>
11: #include <machine.h>
12: #include "h8_3048.h"
13: #include "lcd2.h" /* LCD表示用追加 */
14: #include "switch.h" /* スイッチ追加 */
15: #include "beep.h" /* ブザー追加 */
16: #include "eeprom.h" /* EEP-ROM追加 */
17:
18: /*=====*/
19: /* シンボル定義 */
20: /*=====*/
21:
22: /* 定数設定 */
23: #define TIMER_CYCLE 3071 /* タイマのサイクル 1ms */
24: /* /8で使用する場合、 */
25: /* /8 = 325.5[ns] */
26: /* TIMER_CYCLE = */
27: /* 1[ms] / 325.5[ns] */
28: /* = 3072 */
29: #define PWM_CYCLE 24575 /* PWMのサイクル 8msに変更 */

```



トレーニングボード 実習マニュアル(kit06 版)

```

30 :                                     /* PWM_CYCLE =          */
31 :                                     /*      8[ms] / 325.5[ns] */
32 :                                     /*                        = 24576 */
33 : #define SERVO_CENTER 5000          /* サーボのセンタ値      */
34 : #define HANDLE_STEP 26             /* 1° 分の値              */
35 :
36 : /* マスク値設定 × : マスクあり(無効) : マスク無し(有効) */
37 : #define MASK2_2 0x66               /* × × × × ×            */
38 : #define MASK2_0 0x60               /* × × × × × × ×       */
39 : #define MASK0_2 0x06               /* × × × × × × ×       */
40 : #define MASK3_3 0xe7               /* × × × × × × ×       */
41 : #define MASK0_3 0x07               /* × × × × × × ×       */
42 : #define MASK3_0 0xe0               /* × × × × × × ×       */
43 : #define MASK4_0 0xf0               /* × × × × × × ×       */
44 : #define MASK0_4 0x0f               /* × × × × × × ×       */
45 : #define MASK4_4 0xff               /* × × × × × × ×       */
46 :
47 : /* EEP-ROM関連 */
48 : #define EEP_ROM_SIZE 16            /* EEP-ROM使用サイズ     */
49 :
50 : #define EEPROM_CHECK 0x00          /* EEP-ROMチェック       */
51 : #define EEPROM_SERVO 0x01          /* サーボセンタ値        */
52 :
53 : /*=====*/
54 : /* プロトタイプ宣言 */
55 : /*=====*/
56 : void init( void );
57 : void timer( unsigned long timer_set );
58 : int check_crossline( void );
59 : int check_rightline( void );
60 : int check_leftline( void );
61 : unsigned char sensor_inp( unsigned char mask );
62 : unsigned char dipsw_get( void );
63 : unsigned char pushsw_get( void );
64 : unsigned char startbar_get( void );
65 : void led_out( unsigned char led );
66 : void speed( int accele_l, int accele_r );
67 : void handle( int angle );
68 : char unsigned bit_change( char unsigned in );
69 :
70 : /*=====*/
71 : /* グローバル変数の宣言 */
72 : /*=====*/
73 : unsigned long cnt0;                /* timer関数用           */
74 : unsigned long cnt1;                /* main内で使用          */
75 : int pattern;                       /* パターン番号          */
76 :
77 : /* EEP-ROM設定 */
78 : int eep_buff[ EEP_ROM_SIZE ];
79 :
80 : /*=====*/
81 : /* メインプログラム */
82 : /*=====*/
83 : void main( void )
84 : {
85 :     int i, j;
86 :     int lcd_pattern = 1;
87 :
88 :     /* マイコン機能の初期化 */
89 :     init();                          /* 初期化                 */
90 :     set_ccr( 0x00 );                  /* 全体割り込み許可     */
91 :     initLcd();                        /* LCD初期化             */
92 :     initSwitch();                    /* スイッチ初期化       */
93 :     initBeep();                      /* ブザー初期化         */
94 :     initEeprom();                    /* EEP-ROM初期化        */
95 :
96 :     /*EEP-ROMのチェック */
97 :     if( readEeprom( EEPROM_CHECK ) != 0x2006 ) {
98 :         /*
99 :          IDのチェック EEP-ROMを初めて使うかどうか
100 :          0x00番地にIDが書かれていなければ初めて使うと判断して初期化する
101 :          */
102 :         eep_buff[EEPROM_CHECK] = 0x2006;
103 :         eep_buff[EEPROM_SERVO] = SERVO_CENTER;
104 :     } else {
105 :         /* 2回目以降の使用の場合、データ読み込み */
106 :         for( i=0; i<EEP_ROM_SIZE; i++ ) {
107 :             eep_buff[ i ] = readEeprom( i );
108 :         }
109 :     }
110 :
111 :     /* マイコンカーの状態初期化 */
112 :     handle( 0 );
113 :     speed( 0, 0 );
114 :
115 :     while( 1 ) {
116 :         switch( pattern ) {
117 :
118 :             /*=====*/
119 :             パターンについて
120 :             0: スイッチ入力待ち

```

トレーニングボード 実習マニュアル(kit06 版)

```

121 :      1: スタートバーが開いたかチェック
122 :      11: 通常トレース
123 :      12: 右へ大曲げの終わりのチェック
124 :      13: 左へ大曲げの終わりのチェック
125 :      21: 1本目のクロスライン検出時の処理
126 :      22: 2本目を読み飛ばす
127 :      23: クロスライン後のトレース、クランク検出
128 :      31: 左クランククリア処理 安定するまで少し待つ
129 :      32: 左クランククリア処理 曲げ終わりのチェック
130 :      41: 右クランククリア処理 安定するまで少し待つ
131 :      42: 右クランククリア処理 曲げ終わりのチェック
132 :      51: 1本目の右ハーフライン検出時の処理
133 :      52: 2本目を読み飛ばす
134 :      53: 右ハーフライン後のトレース
135 :      54: 右レーンチェンジ終了のチェック
136 :      61: 1本目の左ハーフライン検出時の処理
137 :      62: 2本目を読み飛ばす
138 :      63: 左ハーフライン後のトレース
139 :      64: 左レーンチェンジ終了のチェック
140 :      *****/
141 :
142 : case 0:
143 :     /* スイッチ入力待ち */
144 :     if( pushsw_get() ) {
145 :         setBeepPattern( 0xc000 );
146 :         /* 保存 */
147 :         for( i=0; i<EEP_ROM_SIZE; i++ ) {
148 :             writeEeprom( i, eep_buff[ i ] );
149 :             while( !checkEeprom() ); /* 書き込み終了チェック */
150 :         }
151 :         pattern = 1;
152 :         cnt1 = 0;
153 :         break;
154 :     }
155 :
156 :     /* スイッチ4 設定値保存 */
157 :     if( getSwFlag(SW_4) ) {
158 :         setBeepPattern( 0x8000 );
159 :         /* 保存 */
160 :         for( i=0; i<EEP_ROM_SIZE; i++ ) {
161 :             writeEeprom( i, eep_buff[ i ] );
162 :             while( !checkEeprom() ); /* 書き込み終了チェック */
163 :         }
164 :         break;
165 :     }
166 :     /* スイッチ3 メニュー+ 1 */
167 :     if( getSwFlag(SW_3) ) {
168 :         lcd_pattern++;
169 :         if( lcd_pattern == 2 ) lcd_pattern = 1;
170 :     }
171 :     /* スイッチ2 メニュー- 1 */
172 :     if( getSwFlag(SW_2) ) {
173 :         lcd_pattern--;
174 :         if( lcd_pattern == 0 ) lcd_pattern = 1;
175 :     }
176 :
177 :     /* スイッチ、LCD処理 */
178 :     switch( lcd_pattern ) {
179 :     case 1:
180 :         /* サーボセンタ値調整 */
181 :         i = eep_buff[EEPROM_SERVO];
182 :         if( getSwFlag(SW_1) ) {
183 :             i++;
184 :             if( i > 10000 ) i = 10000;
185 :         }
186 :         if( getSwFlag(SW_0) ) {
187 :             i--;
188 :             if( i < 1000 ) i = 1000;
189 :         }
190 :         eep_buff[EEPROM_SERVO] = i;
191 :         handle( 0 );
192 :
193 :         /* LCD処理 */
194 :         lcdPosition( 0, 0 );
195 :         /* 0123456789a..ef 1行16文字 */
196 :         lcdPrintf( "1 servo = %05d ", i );
197 :         /* 01234567..89abcde.f 1行16文字 */
198 :         lcdPrintf( "sensor=%02x bar=%d ",
199 :             sensor_inp( 0xff ), startbar_get() );
200 :         break;
201 :     }
202 :
203 :     if( cnt1 < 100 ) { /* LED点滅処理 */
204 :         led_out( 0x1 );
205 :     } else if( cnt1 < 200 ) {
206 :         led_out( 0x2 );
207 :     } else {
208 :         cnt1 = 0;
209 :     }
210 :     break;
211 :

```

## トレーニングボード 実習マニュアル(kit06 版)

```

212 :     case 1:
213 :         /* スタートバーが開いたかチェック */
214 :         if( !startbar_get() ){
215 :             /* スタート!! */
216 :             led_out( 0x0 );
217 :             pattern = 11;
218 :             cnt1 = 0;
219 :             break;
220 :         }
221 :         if( cnt1 < 50 ) {           /* LED点滅処理          */
222 :             led_out( 0x1 );
223 :         } else if( cnt1 < 100 ) {
224 :             led_out( 0x2 );
225 :         } else {
226 :             cnt1 = 0;
227 :         }
228 :         break;

```

中略

```

572 : /*****
573 : /* H8/3048F-ONE 内蔵周辺機能 初期化          */
574 : *****/
575 : void init( void )
576 : {
577 :     /* I/Oポートの入出力設定 */
578 :     P1DDR = 0xff;
579 :     P2DDR = 0xff;
580 :     P3DDR = 0x8e;           /* スイッチ、EEP-ROM      */
581 :     P4DDR = 0xff;         /* LCD接続                */
582 :     P5DDR = 0xff;
583 :     P6DDR = 0xf0;         /* CPU基板上のDIP SW     */
584 :     P8DDR = 0xff;
585 :     P9DDR = 0xe3;         /* bit4,2:sw bit3:232c   */
586 :     PADDR = 0xf7;         /* スタートバー検出センサ */
587 :     PBDR = 0xc0;
588 :     PBDDR = 0xfe;         /* モータドライブ基板Vol.3 */
589 :     /* センサ基板のP7は、入力専用なので入出力設定はありません */
590 :
591 :     /* ITU0 1ms毎の割り込み */
592 :     ITU0_TCR = 0x23;
593 :     ITU0_GRA = TIMER_CYCLE;
594 :     ITU0_IER = 0x01;
595 :
596 :     /* ITU3,4 リセット同期PWMモード 左右モータ、サーボ用 */
597 :     ITU3_TCR = 0x23;
598 :     ITU_FCR = 0x3e;
599 :     ITU3_GRA = PWM_CYCLE;           /* 周期の設定          */
600 :     ITU3_GRB = ITU3_BRB = 0;       /* 左モータのPWM設定   */
601 :     ITU4_GRA = ITU4_BRA = 0;       /* 右モータのPWM設定   */
602 :     ITU4_GRB = ITU4_BRB = SERVO_CENTER; /* サーボのPWM設定     */
603 :     ITU_TOER = 0x38;
604 :
605 :     /* ITUのカウンタスタート */
606 :     ITU_STR = 0x09;
607 : }
608 :
609 : /*****
610 : /* ITU0 割り込み処理          */
611 : *****/
612 : #pragma interrupt( interrupt_timer0 )
613 : void interrupt_timer0( void )
614 : {
615 :     ITU0_TSR &= 0xfe;           /* フラグクリア        */
616 :     cnt0++;
617 :     cnt1++;
618 :
619 :     /* LCD表示処理用関数です。1ms毎に実行します。          */
620 :     lcdShowProcess();
621 :     /* 拡張スイッチ用関数です。1ms毎に実行します。          */
622 :     switchProcess();
623 :     /* ブザー処理用関数です。1ms毎に実行します。          */
624 :     beepProcess();
625 : }

```

中略

```

799 : /*****
800 : /* サーボハンドルの操作          */
801 : /* 引数   サーボ操作角度：-90~90          */
802 : /*       -90で左へ90度、0でまっすぐ、90で右へ90度回転          */
803 : *****/
804 : void handle( int angle )
805 : {
806 :     ITU4_BRB = eep_buff[EEPROM_SERVO] - angle * HANDLE_STEP;
807 : }

```

以下、略

## 10.4 プログラム「tr\_12start.src」

「kit06start.src」と比べて、太字部分が追加、変更されています。ブザー処理を行うためです。忘れがちですので気をつけてください。

```

1 : ;=====
2 : ; 定義
3 : ;=====
4 : RESERVE: .EQU    H'FFFFFFF      ; 未使用領域のアドレス
5 :
6 : ;=====
7 : ; 外部参照
8 : ;=====
9 :     .IMPORT _main
10 :     .IMPORT _INITSC
11 :     .IMPORT _interrupt_timer0
12 :     .IMPORT _intTXIO
13 :
14 : ;=====
15 : ; ベクタセクション
16 : ;=====
17 :     .SECTION V
18 :     .DATA.L RESET_START          ; 0 H'000000 リセット
19 :     .DATA.L RESERVE              ; 1 H'000004 システム予約
20 :     .DATA.L RESERVE              ; 2 H'000008 システム予約
21 :     .DATA.L RESERVE              ; 3 H'00000c システム予約
22 :     .DATA.L RESERVE              ; 4 H'000010 システム予約
23 :     .DATA.L RESERVE              ; 5 H'000014 システム予約
24 :     .DATA.L RESERVE              ; 6 H'000018 システム予約
25 :     .DATA.L RESERVE              ; 7 H'00001c 外部割り込み NMI
26 :     .DATA.L RESERVE              ; 8 H'000020 トラップ 命令
27 :     .DATA.L RESERVE              ; 9 H'000024 トラップ 命令
28 :     .DATA.L RESERVE              ; 10 H'000028 トラップ 命令
29 :     .DATA.L RESERVE              ; 11 H'00002c トラップ 命令
30 :     .DATA.L RESERVE              ; 12 H'000030 外部割り込み IRQ0
31 :     .DATA.L RESERVE              ; 13 H'000034 外部割り込み IRQ1
32 :     .DATA.L RESERVE              ; 14 H'000038 外部割り込み IRQ2
33 :     .DATA.L RESERVE              ; 15 H'00003c 外部割り込み IRQ3
34 :     .DATA.L RESERVE              ; 16 H'000040 外部割り込み IRQ4
35 :     .DATA.L RESERVE              ; 17 H'000044 外部割り込み IRQ5
36 :     .DATA.L RESERVE              ; 18 H'000048 システム予約
37 :     .DATA.L RESERVE              ; 19 H'00004c システム予約
38 :     .DATA.L RESERVE              ; 20 H'000050 WDT MOV1
39 :     .DATA.L RESERVE              ; 21 H'000054 REF CMI
40 :     .DATA.L RESERVE              ; 22 H'000058 システム予約
41 :     .DATA.L RESERVE              ; 23 H'00005c システム予約
42 :     .DATA.L _interrupt_timer0 ; 24 h'000060 ITU0 IMIA0
43 :     .DATA.L RESERVE              ; 25 H'000064 ITU0 IMIB0
44 :     .DATA.L RESERVE              ; 26 H'000068 ITU0 OV10
45 :     .DATA.L RESERVE              ; 27 H'00006c システム予約
46 :     .DATA.L RESERVE              ; 28 H'000070 ITU1 IMIA1
47 :     .DATA.L RESERVE              ; 29 H'000074 ITU1 IMIB1
48 :     .DATA.L RESERVE              ; 30 H'000078 ITU1 OV11
49 :     .DATA.L RESERVE              ; 31 H'00007c システム予約
50 :     .DATA.L RESERVE              ; 32 H'000080 ITU2 IMIA2
51 :     .DATA.L RESERVE              ; 33 H'000084 ITU2 IMIB2
52 :     .DATA.L RESERVE              ; 34 H'000088 ITU2 OV12
53 :     .DATA.L RESERVE              ; 35 H'00008c システム予約
54 :     .DATA.L RESERVE              ; 36 H'000090 ITU3 IMIA3
55 :     .DATA.L RESERVE              ; 37 H'000094 ITU3 IMIB3
56 :     .DATA.L RESERVE              ; 38 H'000098 ITU3 OV13
57 :     .DATA.L RESERVE              ; 39 H'00009c システム予約
58 :     .DATA.L RESERVE              ; 40 H'0000a0 ITU4 IMIA4
59 :     .DATA.L RESERVE              ; 41 H'0000a4 ITU4 IMIB4
60 :     .DATA.L RESERVE              ; 42 H'0000a8 ITU4 OV14
61 :     .DATA.L RESERVE              ; 43 H'0000ac システム予約
62 :     .DATA.L RESERVE              ; 44 H'0000b0 DMAC DEND0A
63 :     .DATA.L RESERVE              ; 45 H'0000b4 DMAC DEND0B
64 :     .DATA.L RESERVE              ; 46 H'0000b8 DMAC DEND1A
65 :     .DATA.L RESERVE              ; 47 H'0000bc DMCA DEND1B
66 :     .DATA.L RESERVE              ; 48 H'0000c0 システム予約
67 :     .DATA.L RESERVE              ; 49 H'0000c4 システム予約
68 :     .DATA.L RESERVE              ; 50 H'0000c8 システム予約
69 :     .DATA.L RESERVE              ; 51 H'0000cc システム予約
70 :     .DATA.L RESERVE              ; 52 H'0000d0 SC10 ER10
71 :     .DATA.L RESERVE              ; 53 H'0000d4 SC10 RX10
72 :     .DATA.L _intTXIO          ; 54 h'0000d8 SC10 TXIO
73 :     .DATA.L RESERVE              ; 55 H'0000dc SC10 TE10
74 :     .DATA.L RESERVE              ; 56 H'0000e0 SC11 ER11
75 :     .DATA.L RESERVE              ; 57 H'0000e4 SC11 RX11
76 :     .DATA.L RESERVE              ; 58 H'0000e8 SC11 TX11
77 :     .DATA.L RESERVE              ; 59 H'0000ec SC11 TE11
78 :     .DATA.L RESERVE              ; 60 H'0000f0 A/D ADI
79 :
80 : ;=====

```

## トレーニングボード 実習マニュアル(kit06 版)

```

81 : ; スタートアッププログラム
82 : ;=====
83 : .SECTION P
84 : RESET_START:
85 :     MOV.L   #H'FFF10,ER7      ; スタックの設定
86 :     JSR     @_INITSCCT        ; RAMエリアの初期化
87 :     JSR     @_main            ; C言語のmain()関数へジャンプ
88 : OWARI :
89 :     BRA     OWARI
90 :
91 :     .END

```

## 10.5 プログラムの解説

### 10.5.1 ヘッドファイルの追加

```

6 : /*=====*/
7 : /* インクルード */
8 : /*=====*/
9 : #include <no_float.h> /* stdio の簡略化 最初に置く*/
10 : #include <stdio.h>
11 : #include <machine.h>
12 : #include "h8_3048.h"
13 : #include "lcd2.h" /* LCD 表示用追加 */
14 : #include "switch.h" /* スイッチ追加 */
15 : #include "beep.h" /* ブザー追加 */
16 : #include "eeprom.h" /* EEPROM追加 */

```

トレーニングボードを使用するに当たって、ヘッドファイルを追加します。

### 10.5.2 EEPROM エリアの確保

```

47 : /* EEPROM 関連 */
48 : #define EEPROM_ROM_SIZE 16 /* EEPROM 使用サイズ */
49 :
50 : #define EEPROM_CHECK 0x00 /* EEPROM チェック */
51 : #define EEPROM_SERVO 0x01 /* サーボセンタ値 */
中略
77 : /* EEPROM 設定 */
78 : int eep_buff[ EEPROM_ROM_SIZE ];

```

EEP-ROM のデータの読み書きは時間がかかるため、最初に全データを読み込み、スタート前に全データを書き込みます。読み込むための領域を RAM 上に確保します。

番地	番地名	内容	初期値
0x00	EEPROM_CHECK	EEP-ROM チェック用 0x2006 で無ければ、初期値を読み込む	0x2006
0x01	EEPROM_SERVO	<b>サーボセンタ値</b>	<b>5000</b>
0x02 ~ 0x0f	未定義		

### 10.5.3 初期化

```

83 : void main( void )
84 : {
85 :     int    i, j;
86 :     int    lcd_pattern = 1;
87 :
88 :     /* マイコン機能の初期化 */
89 :     init();                               /* 初期化                */
90 :     set_ccr( 0x00 );                       /* 全体割り込み許可    */
91 :     initLcd();                             /* LCD 初期化          */
92 :     initSwitch();                         /* スイッチ初期化      */
93 :     initBeep();                          /* ブザー初期化        */
94 :     initEeprom();                         /* EEPROM 初期化       */

```

新しく使用する変数の宣言と、トレーニングボードで使用する機能の初期化をおこないます。

### 10.5.4 EEPROM-ROM から読み込み

```

96 :     /*EEP-ROM のチェック */
97 :     if( readEeprom( EEPROM_CHECK ) != 0x2006 ) {
98 :         /*
99 :         ID のチェック EEPROM-ROM を初めて使うかどうか
100 :         0x00 番地に ID が書かれていなければ初めて使うと判断して初期化する
101 :         */
102 :         eep_buff[EEPROM_CHECK] = 0x2006;
103 :         eep_buff[EEPROM_SERVO] = SERVO_CENTER;
104 :     } else {
105 :         /* 2 回目以降の使用の場合、データ読み込み */
106 :         for( i=0; i<EEP_ROM_SIZE; i++ ) {
107 :             eep_buff[ i ] = readEeprom( i );
108 :         }
109 :     }

```

97 行で EEPROM\_CHECK(0x00)番地が 0x2006 かどうかチェックします。違うなら初めて使うと判断して EEPROM-ROM から読み込むデータ全てを初期化します。

106 ~ 107 行で、チェック値が 0x2006 なら 2 回目以降と判断して EEPROM-ROM からデータ読み込みます。読み込むのは 48 行目で定義した EEPROM\_SIZE 個数分の、16 個になります。ただし、今回は 0x02 番地以降のデータは読み込んでも意味がありません。

### 10.5.5 スイッチ入力待ち

```

142 :     case 0:
143 :         /* スイッチ入力待ち */
144 :         if( pushsw_get() ) {
145 :             setBeepPattern( 0xc000 );
146 :             /* 保存 */
147 :             for( i=0; i<EEP_ROM_SIZE; i++ ) {
148 :                 writeEeprom( i, eep_buff[ i ] );
149 :                 while( !checkEeprom() ); /* 書き込み終了チェック */
150 :             }
151 :             pattern = 1;
152 :             cnt1 = 0;
153 :             break;
154 :         }

```

「kit06.c」同様 pattern が 0 の時、スイッチ入力待ちとなります。モータドライブ基板のプッシュスイッチが押されると、

- ・145 行 ... ブザーを鳴らす
  - ・147～150 行 ... EEPROM にデータを保存する
  - ・151 行 ... パターン 1 へ移行する
- という動作をします。

```

156 :         /* スイッチ 4 設定値保存 */
157 :         if( getSwFlag(SW_4) ) {
158 :             setBeepPattern( 0x8000 );
159 :             /* 保存 */
160 :             for( i=0; i<EEP_ROM_SIZE; i++ ) {
161 :                 writeEeprom( i, eep_buff[ i ] );
162 :                 while( !checkEeprom() ); /* 書き込み終了チェック */
163 :             }
164 :             break;
165 :         }

```

157 行で、トレーニングボードの SW\_4 が押されたかどうかチェックします。SW\_4 が押されると、データを保存します。モータドライブ基板のプッシュスイッチと違うのは、データの保存のみを行いパターン 1 へは移りません。

```

166 :         /* スイッチ 3 メニュー + 1 */
167 :         if( getSwFlag(SW_3) ) {
168 :             lcd_pattern++;
169 :             if( lcd_pattern == 2 ) lcd_pattern = 1;
170 :         }
171 :         /* スイッチ 2 メニュー - 1 */
172 :         if( getSwFlag(SW_2) ) {
173 :             lcd_pattern--;
174 :             if( lcd_pattern == 0 ) lcd_pattern = 1;
175 :         }

```

lcd\_pattern という変数は、LCD に表示する内容を選ぶ変数です。今回、表示項目はサーボセンタ値のみですが、今後、lcd\_pattern の内容を増やしていき、SW\_3、SW\_2 で表示する内容を選択します。

169 行で上限のチェックです。lcd\_pattern の 2 はありませんので 1 にします。

174 行で下限のチェックです。lcd\_pattern の 0 はありませんので 1 にします(結局ここでは、SW\_3、SW\_2 を押しても何も変わりません)。

lcd_pattern	内容
1	サーボセンタ値の調整

```

177 :      /* スイッチ、LCD処理 */
178 :      switch( lcd_pattern ) {
179 :      case 1:
180 :          /* サーボセンタ値調整 */
181 :          i = eep_buff[EEPROM_SERVO];
182 :          if( getSwFlag(SW_1) ) {
183 :              i++;
184 :              if( i > 10000 ) i = 10000;
185 :          }
186 :          if( getSwFlag(SW_0) ) {
187 :              i--;
188 :              if( i < 1000 ) i = 1000;
189 :          }
190 :          eep_buff[EEPROM_SERVO] = i;
191 :          handle( 0 );
192 :
193 :          /* LCD 処理 */
194 :          lcdPosition( 0, 0 );
195 :          /* 0123456789a..ef 1行16文字 */
196 :          lcdPrintf( "1 servo = %05d ", i );
197 :          /* 01234567..89abcde.f 1行16文字 */
198 :          lcdPrintf( "sensor=%02x bar=%d ",
199 :                  sensor_inp( 0xff ), startbar_get() );
200 :          break;
201 :      }

```

178 行で、lcd\_pattern の番号によりプログラムをジャンプします。今回は、1 しか無いので必ず 179 行へ飛びます。

181 行で、eep\_buff[EEPROM\_SERVO]変数の値をいったん i 変数に代入します。

182 行で、SW\_1 が押されているかチェックします。押されていれば、i 変数を1つ増加させます。

186 行で、SW\_0 が押されているかチェックします。押されていれば、i 変数を1つ減少させます。

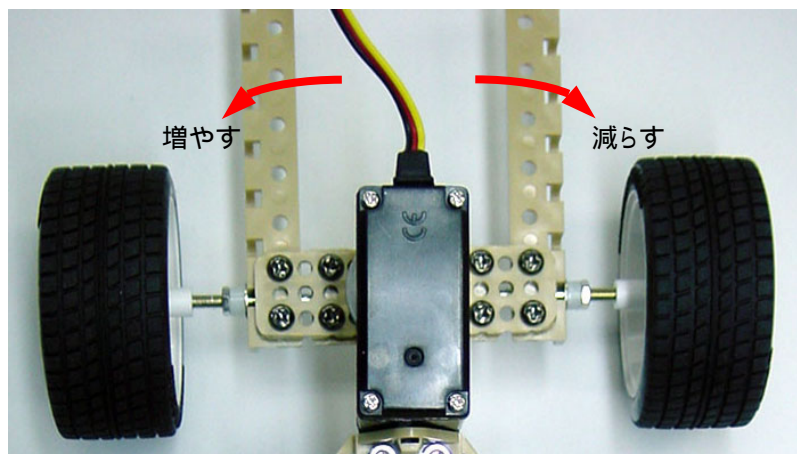
190 行で、i 変数の値を eep\_buff[EEPROM\_SERVO]変数へ代入します。SW\_1、SW\_2 で操作された値が代入されます。

191 行でハンドル角度を 0 度に設定します。0 度と言っても eep\_buff[EEPROM\_SERVO]変数の値を変えていまずので、その値に応じて動きます。

194 行から LCD に現在のサーボセンタ値を表示します。2 行目が余っているのでセンサの状態を表示させています。



キットのサーボは、eep\_buff[EEPROM\_SERVO]変数の値を増やせば向かって左側、減らせば右側に向きます。



### 10.5.6 ポートの入出力設定

```

572 : /****** /
573 : /* H8/3048F-ONE 内蔵周辺機能 初期化 */
574 : /****** /
575 : void init( void )
576 : {
577 :     /* I/O ポートの入出力設定 */
578 :     P1DDR = 0xff;
579 :     P2DDR = 0xff;
580 :     P3DDR = 0x8e; /* スイッチ、EEP-ROM */
581 :     P4DDR = 0xff; /* LCD 接続 */
582 :     P5DDR = 0xff;
583 :     P6DDR = 0xf0; /* CPU 基板上的 DIP SW */
584 :     P8DDR = 0xff;
585 :     P9DDR = 0xe3; /* bit4,2:sw bit3:232c */
586 :     PADDR = 0xf7; /* スタートバー検出センサ */
587 :     PBDR = 0xc0;
588 :     PBDDR = 0xfe; /* モータドライブ基板 Vol.3 */
589 :     /* センサ基板の P7 は、入力専用なので入出力設定はありません */

```

トレーニングボードのコネクタに合わせて、出力する機器は出力、入力する機器は入力にポートを設定します。

### 10.5.7 割り込み処理の追加

```

609 : /*****/
610 : /* ITU0 割り込み処理 */
611 : /*****/
612 : #pragma interrupt( interrupt_timer0 )
613 : void interrupt_timer0( void )
614 : {
615 :     ITU0_TSR &= 0xfe;          /* フラグクリア */
616 :     cnt0++;
617 :     cnt1++;
618 :
619 :     /* LCD 表示処理用関数です。1ms 毎に実行します。 */
620 :     lcdShowProcess();
621 :     /* 拡張スイッチ用関数です。1ms 毎に実行します。 */
622 :     switchProcess();
623 :     /* ブザー処理用関数です。1ms 毎に実行します。 */
624 :     beepProcess();
625 : }
    
```

interrupt\_timer0 関数内に、LCD 処理、スイッチ処理、ブザー処理を行う関数を追加します。

### 10.5.8 handle 関数の変更

```

804 : void handle( int angle )
805 : {
806 :     ITU4_BRB = eep_buff[EEPROM_SERVO] - angle * HANDLE_STEP;
807 : }
    
```

センタの値を SERVO\_CENTER の固定値から、eep\_buff[EEPROM\_SERVO]変数に変えます。SW\_1、SW\_0 でこの変数を増減すると、それに応じてサーボも少しずつ動きます。

もし、直接 EEP-ROM からデータを読み込むとどうなるでしょう。

```

804 : void handle( int angle )
805 : {
806 :     ITU4_BRB = readEeprom( EEPROM_SERVO ) - angle * HANDLE_STEP;
807 : }
    
```

**読み込みに時間がかかり動作に影響するので×**

eep\_buff[EEPROM\_SERVO]は、RAM にある変数なので短い時間で読み込むことができますが、EEP-ROM から直接データを読み込む場合は時間がかかってしまい、マイコンカー全体の動作に影響を与えてしまう場合があります。必ず変数にします。

### 10.5.9 スタートアップルーチン「tr\_12start.src」の追加、変更

ブザー処理では、ベクタアドレス 54 番の割り込みを使用します。割り込み関数は intTXIO 関数です。そのため、スタートアップルーチンに intTXIO 関数を追加する必要があります。

```

6 : ;=====
7 : ; 外部参照
8 : ;=====
9 :     .IMPORT _main
10 :     .IMPORT _INITSC
11 :     .IMPORT _interrupt_timer0
12 :     .IMPORT _intTXIO
    
```

「.IMPORT」宣言部分に「\_intTXIO」を追加します。「intTXIO 関数が tr\_12start.src ファイルとは別のファイルにあり、外部を参照してください」ということを宣言しています。ちなみに、intTXIO 関数は、beep.c ファイル内にあります。

ベクタ番号 54 の SCIO TXIO 割り込みの宣言を「RESERVE」から「\_intTXIO」に変更します。

```
72 :     .DATA.L RESERVE                ; 54 h'0000d8    SCIO TXIO
```

```
72 :     .DATA.L _intTXIO                ; 54 h'0000d8    SCIO TXIO
```


12 行を追加、72 行を変更すれば、ブザーを使用することができます。

## 11. プロジェクト「tr\_13」 スピードの調整

### 11.1 内容

マイコンカーのスピード調整は、CPU ボードのディップスイッチで PWM の値を変えることにより行っていました。せっかく LCD とスイッチがありますので、PWM 値を LCD に表示してスイッチで増減させるようにします。PWM 値は 0 ~ 100% まで 1% ごとに設定でき、細かい調整が可能です。

### 11.2 プロジェクトの構成

 tr_13	·tr_13start.src
Assembly source file	ベクタアドレスの設定、スタートアップルーチンが記述されています。
tr_13start.src	·car_printf2.c
C source file	セクションの初期化、printf 文、scanf 文を実行します。
beep.c	·lcd2.c
car_printf2.c	LCD 制御を行います。
eeprom.c	·switch.c
lcd2.c	スイッチ制御を行います。
switch.c	·beep.c
tr_13.c	ブザー制御を行います。
Dependencies	·eeprom.c
beep.h	EEP-ROM 制御を行います。
eeprom.h	·tr_13.c
h8_3048.h	メインプログラムです。
lcd2.h	
switch.h	

### 11.3 プログラム「tr\_13.c」

プログラムのゴシック体部分が追加した部分です。

前略

```

47 : /* EEP-ROM関連 */
48 : #define EEP_ROM_SIZE 16 /* EEP-ROM使用サイズ */
49 :
50 : #define EEPROM_CHECK 0x00 /* EEP-ROMチェック */
51 : #define EEPROM_SERVO 0x01 /* サーボセンタ値 */
52 : #define EEPROM_PWM 0x02 /* PWM値 */
53 :
54 : /*=====*/
55 : /* プロトタイプ宣言 */
56 : /*=====*/
57 : void init( void );
58 : void timer( unsigned long timer_set );
59 : int check_crossline( void );
60 : int check_rightline( void );
61 : int check_leftline( void );
62 : unsigned char sensor_inp( unsigned char mask );
63 : unsigned char dipsw_get( void );
64 : unsigned char pushsw_get( void );
65 : unsigned char startbar_get( void );
66 : void led_out( unsigned char led );
67 : void speed( int accele_l, int accele_r );
68 : void handle( int angle );
69 : char unsigned bit_change( char unsigned in );
70 :
71 : /*=====*/
72 : /* グローバル変数の宣言 */
73 : /*=====*/
74 : unsigned long cnt0; /* timer関数用 */
75 : unsigned long cnt1; /* main内で使用 */
76 : int pattern; /* パターン番号 */

```

トレーニングボード 実習マニュアル(kit06 版)

```

77 :
78 : /* EEPROM設定 */
79 : int      eep_buff[ EEPROM_SIZE ];
80 :
81 : /*****
82 : /* メインプログラム */
83 : /*****
84 : void main( void )
85 : {
86 :     int      i, j;
87 :     int      lcd_pattern = 2;
88 :
89 :     /* マイコン機能の初期化 */
90 :     init(); /* 初期化 */
91 :     set_ccr( 0x00 ); /* 全体割り込み許可 */
92 :     initLcd(); /* LCD初期化 */
93 :     initSwitch(); /* スイッチ初期化 */
94 :     initBeep(); /* ブザー初期化 */
95 :     initEeprom(); /* EEPROM初期化 */
96 :
97 :     /*EEP-ROMのチェック */
98 :     if( readEeprom( EEPROM_CHECK ) != 0x2006 ) {
99 :         /*
100 :         IDのチェック EEPROMを初めて使うかどうか
101 :         0x00番地にIDが書かれていなければ初めて使うと判断して初期化する
102 :         */
103 :         eep_buff[EEPROM_CHECK] = 0x2006;
104 :         eep_buff[EEPROM_SERVO] = SERVO_CENTER;
105 :         eep_buff[EEPROM_PWM] = 50;
106 :     } else {
107 :         /* 2回目以降の使用の場合、データ読み込み */
108 :         for( i=0; i<EEP_ROM_SIZE; i++ ) {
109 :             eep_buff[ i ] = readEeprom( i );
110 :         }
111 :     }
112 :
113 :     /* マイコンカーの状態初期化 */
114 :     handle( 0 );
115 :     speed( 0, 0 );
116 :
117 :     while( 1 ) {
118 :         switch( pattern ) {
119 :
120 :         case 0:
121 :             /* スイッチ入力待ち */
122 :             if( pushsw_get() ) {
123 :                 setBeepPattern( 0xc000 );
124 :                 /* 保存 */
125 :                 for( i=0; i<EEP_ROM_SIZE; i++ ) {
126 :                     writeEeprom( i, eep_buff[ i ] );
127 :                     while( !checkEeprom() ); /* 書き込み終了チェック */
128 :                 }
129 :                 pattern = 1;
130 :                 cnt1 = 0;
131 :                 break;
132 :             }
133 :
134 :             /* スイッチ4 設定値保存 */
135 :             if( getSwFlag(SW_4) ) {
136 :                 setBeepPattern( 0x8000 );
137 :                 /* 保存 */
138 :                 for( i=0; i<EEP_ROM_SIZE; i++ ) {
139 :                     writeEeprom( i, eep_buff[ i ] );
140 :                     while( !checkEeprom() ); /* 書き込み終了チェック */
141 :                 }
142 :                 break;
143 :             }
144 :
145 :             /* スイッチ3 メニュー+ 1 */
146 :             if( getSwFlag(SW_3) ) {
147 :                 lcd_pattern++;
148 :                 if( lcd_pattern == 3 ) lcd_pattern = 1;
149 :             }
150 :
151 :             /* スイッチ2 メニュー- 1 */
152 :             if( getSwFlag(SW_2) ) {
153 :                 lcd_pattern--;
154 :                 if( lcd_pattern == 0 ) lcd_pattern = 2;
155 :             }
156 :
157 :             /* スイッチ、LCD処理 */
158 :             switch( lcd_pattern ) {
159 :             case 1:
160 :                 /* サーボセンタ値調整 */
161 :                 i = eep_buff[EEPROM_SERVO];
162 :                 if( getSwFlag(SW_1) ) {
163 :                     i++;
164 :                     if( i > 10000 ) i = 10000;
165 :                 }
166 :                 if( getSwFlag(SW_0) ) {
167 :                     i--;
168 :                     if( i < 1000 ) i = 1000;
169 :                 }
170 :             }
171 :             }
172 :
173 :             /* サーボセンタ値調整 */
174 :             i = eep_buff[EEPROM_SERVO];
175 :             if( getSwFlag(SW_1) ) {
176 :                 i++;
177 :                 if( i > 10000 ) i = 10000;
178 :             }
179 :             if( getSwFlag(SW_0) ) {
180 :                 i--;
181 :                 if( i < 1000 ) i = 1000;
182 :             }
183 :         }
184 :     }
185 : }
186 :
187 :
188 :
189 :
190 :
191 :

```

## トレーニングボード 実習マニュアル(kit06 版)

```

192 :         eep_buff[EEPROM_SERVO] = i;
193 :         handle( 0 );
194 :
195 :         /* LCD処理 */
196 :         lcdPosition( 0, 0 );
197 :         /* 0123456789a..ef 1行16文字 */
198 :         lcdPrintf( "1 servo = %05d ", i );
199 :         /* 01234567..89abcde.f 1行16文字 */
200 :         lcdPrintf( "sensor=%02x bar=%d ",
201 :                   sensor_inp( 0xff ), startbar_get() );
202 :         break;
203 :
204 :     case 2:
205 :         /* PWM調整 */
206 :         i = eep_buff[EEPROM_PWM];
207 :         if( getSwFlag(SW_1) ) {
208 :             i++;
209 :             if( i > 100 ) i = 100;
210 :         }
211 :         if( getSwFlag(SW_0) ) {
212 :             i--;
213 :             if( i < 0 ) i = 0;
214 :         }
215 :         eep_buff[EEPROM_PWM] = i;
216 :
217 :         /* LCD処理 */
218 :         lcdPosition( 0, 0 );
219 :         /* 012345678..abcdef 1行16文字 */
220 :         lcdPrintf( "2 pwm = %03d ", i );
221 :         /* 01234567..89abcde.f 1行16文字 */
222 :         lcdPrintf( "sensor=%02x bar=%d ",
223 :                   sensor_inp( 0xff ), startbar_get() );
224 :         break;
225 :     }
226 :
227 :     if( cnt1 < 100 ) { /* LED点滅処理 */
228 :         led_out( 0x1 );
229 :     } else if( cnt1 < 200 ) {
230 :         led_out( 0x2 );
231 :     } else {
232 :         cnt1 = 0;
233 :     }
234 :     break;

```

中略

```

789 : /****** */
790 : /* 速度制御 */
791 : /* 引数 左モータ:-100~100 , 右モータ:-100~100 */
792 : /*      0で停止、100で正転100%、-100で逆転100% */
793 : /****** */
794 : void speed( int accele_l, int accele_r )
795 : {
796 :     unsigned long speed_max;
797 :
798 :     speed_max = (unsigned long)(PWM_CYCLE-1) * eep_buff[EEPROM_PWM] / 100;
799 :
800 :     /* 左モータ */
801 :     if( accele_l >= 0 ) {
802 :         PBDR &= 0xfb;
803 :         ITU3_BRB = speed_max * accele_l / 100;
804 :     } else {
805 :         PBDR |= 0x04;
806 :         accele_l = -accele_l;
807 :         ITU3_BRB = speed_max * accele_l / 100;
808 :     }
809 :
810 :     /* 右モータ */
811 :     if( accele_r >= 0 ) {
812 :         PBDR &= 0xf7;
813 :         ITU4_BRA = speed_max * accele_r / 100;
814 :     } else {
815 :         PBDR |= 0x08;
816 :         accele_r = -accele_r;
817 :         ITU4_BRA = speed_max * accele_r / 100;
818 :     }
819 : }

```

以下、略

## 11.4 プログラムの解説

### 11.4.1 EEP-ROM エリアの追加

```

47 : /* EEP-ROM 関連 */
48 : #define    EEP_ROM_SIZE    16    /* EEP-ROM 使用サイズ    */
49 :
50 : #define    EEPROM_CHECK    0x00    /* EEP-ROM チェック    */
51 : #define    EEPROM_SERVO    0x01    /* サーボセンタ値    */
52 : #define    EEPROM_PWM    0x02    /* PWM 値    */
    
```

PWM を設定する EEP-ROM の番地を決めます。0x01 番地まで使っていたので、0x02 番地を EEPROM\_PWM として定義します。

番地	番地名	内容	初期値
0x00	EEPROM_CHECK	EEP-ROM チェック用 0x2006 で無ければ、初期値を読み込む	0x2006
0x01	EEPROM_SERVO	サーボセンタ値	5000
<b>0x02</b>	<b>EEPROM_PWM</b>	<b>PWM 値</b>	<b>50</b>
0x03 ~ 0x0f	未定義		

### 11.4.2 EEP-ROM から読み込み

```

97 : /*EEP-ROM のチェック */
98 : if( readEeprom( EEPROM_CHECK ) != 0x2006 ) {
99 :     /*
100 :     ID のチェック EEP-ROM を初めて使うかどうか
101 :     0x00 番地に ID が書かれていなければ初めて使うと判断して初期化する
102 :     */
103 :     eep_buff[EEPROM_CHECK] = 0x2006;
104 :     eep_buff[EEPROM_SERVO] = SERVO_CENTER;
105 :     eep_buff[EEPROM_PWM] = 50;
106 : } else {
107 :     /* 2 回目以降の使用の場合、データ読み込み */
108 :     for( i=0; i<EEP_ROM_SIZE; i++ ) {
109 :         eep_buff[ i ] = readEeprom( i );
110 :     }
111 : }
    
```

105 行に eep\_buff[EEPROM\_PWM]変数の初期化を追加しています。

### 11.4.3 スイッチ入力待ち

```

168 :      /* スイッチ 3 メニュー + 1 */
169 :      if( getSwFlag(SW_3) ) {
170 :          lcd_pattern++;
171 :          if( lcd_pattern == 3 ) lcd_pattern = 1;
172 :      }
173 :      /* スイッチ 2 メニュー - 1 */
174 :      if( getSwFlag(SW_2) ) {
175 :          lcd_pattern--;
176 :          if( lcd_pattern == 0 ) lcd_pattern = 2;
177 :      }

```

SW\_3 で LCD に表示する内容を次に進めます。SW\_2 で戻します。lcd\_pattern=2 を PWM 値の調整として追加します。

171 行で上限のチェックです。lcd\_pattern の 3 はありませんので 1 にします。

176 行で下限のチェックです。lcd\_pattern の 0 はありませんので 2 にします。

lcd_pattern	内容
1	サーボセンタ値の調整
2	PWM 値の調整

```

179 :      /* スイッチ、LCD 処理 */
180 :      switch( lcd_pattern ) {
181 :      case 1:
182 :          /* サーボセンタ値調整 */
183 :          中略
202 :          break;
203 :
204 :      case 2:
205 :          /* PWM 調整 */
206 :          i = eep_buff[EEPROM_PWM];
207 :          if( getSwFlag(SW_1) ) {
208 :              i++;
209 :              if( i > 100 ) i = 100;
210 :          }
211 :          if( getSwFlag(SW_0) ) {
212 :              i--;
213 :              if( i < 0 ) i = 0;
214 :          }
215 :          eep_buff[EEPROM_PWM] = i;
216 :
217 :          /* LCD 処理 */
218 :          lcdPosition( 0, 0 );
219 :          /* 012345678..abcdef 1 行 16 文字 */
220 :          lcdPrintf( "2 pwm = %03d", i );
221 :          /* 01234567..89abcde.f 1 行 16 文字 */
222 :          lcdPrintf( "sensor=%02x bar=%d ",
223 :                  sensor_inp( 0xff ), startbar_get() );
224 :          break;
225 :      }

```



180 行で、lcd\_pattern の番号によりプログラムをジャンプします。2 なら 204 行へ飛びます。  
206 行で、eep\_buff[EEPROM\_PWM]変数の値をいったん i 変数に代入します。  
207 行で、SW\_1 が押されているかチェックします。押されていれば、i 変数を1つ増加させます。  
211 行で、SW\_0 が押されているかチェックします。押されていれば、i 変数を1つ減少させます。  
215 行で、i 変数の値を eep\_buff[EEPROM\_PWM]変数へ代入します。SW\_1、SW\_2 で操作された値が代入されます。  
218 行から LCD に走行 PWM 値を表示します。2 行目が余っているのでセンサの状態を表示させています。

#### 11.4.4 speed 関数の変更

```
794 : void speed( int accele_l, int accele_r )
795 : {
796 :     unsigned long  speed_max;
797 :
798 :     speed_max = (unsigned long)(PWM_CYCLE-1) * eep_buff[EEPROM_PWM] / 100;
以下、略
```

「kit06.c」ではディップスイッチの状態に応じて、speed\_max の値を決めていましたが、今回は eep\_buff[EEPROM\_PWM]変数の値に応じて割合を決めます。この変数は 0～100 の値ですので、100 で割ります。

$$\text{speed\_max} = (\text{PWM\_CYCLE}-1) \times \frac{\text{eep\_buff}[\text{EEPROM\_PWM}]}{100}$$

例えば、eep\_buff[EEPROM\_PWM]が 75 の場合、

$$\begin{aligned} \text{speed\_max} &= (\text{PWM\_CYCLE}-1) * \text{eep\_buff}[\text{EEPROM\_PWM}] / 100 \\ &= (24575 - 1) * 75 / 100 \\ &= 24574 * 75 / 100 \\ &= 18430 \end{aligned}$$

となります。

## 12. プロジェクト「tr\_14」 カーブでの左右回転差の計算

### 12.1 内容

ハンドルを切る角度によってタイヤの左右回転差を計算すると、回転差を無くすることができます。しかし、走行テスト中にハンドルの切れ角を変えるたびに計算するのはちょっと骨の折れる作業です。そこで、外輪(多く回るタイヤ側)のPWM値を設定すると、現在のハンドルの切れ角を考慮して、内輪のPWM値が返ってくる関数を作ります。

### 12.2 プロジェクトの構成

	<ul style="list-style-type: none"> <li>·tr_14start.src ベクタアドレスの設定、スタートアップルーチンが記述されています。</li> <li>·car_printf2.c セクションの初期化、printf 文、scanf 文を実行します。</li> <li>·lcd2.c LCD 制御を行います。</li> <li>·switch.c スイッチ制御を行います。</li> <li>·beep.c ブザー制御を行います。</li> <li>·eeprom.c EEP-ROM 制御を行います。</li> <li>·tr_14.c メインプログラムです。</li> </ul>
--	---

### 12.3 プログラム「tr\_14.c」

プログラムのゴシック体部分が追加した部分です。

前略

```

54 : /*=====*/
55 : /* プロトタイプ宣言 */
56 : /*=====*/
57 : void init( void );
58 : void timer( unsigned long timer_set );
59 : int check_crossline( void );
60 : int check_rightline( void );
61 : int check_leftline( void );
62 : unsigned char sensor_inp( unsigned char mask );
63 : unsigned char dipsw_get( void );
64 : unsigned char pushsw_get( void );
65 : unsigned char startbar_get( void );
66 : void led_out( unsigned char led );
67 : void speed( int accele_l, int accele_r );
68 : void handle( int angle );
69 : char unsigned bit_change( char unsigned in );
70 : int diff( int pwm );
71 :
72 : /*=====*/
73 : /* グローバル変数の宣言 */
74 : /*=====*/
75 : unsigned long cnt0; /* timer関数用 */
76 : unsigned long cnt1; /* main内で使用 */
77 : int pattern; /* パターン番号 */
78 :
79 : /* EEPROM設定 */
80 : int eep_buff[ EEPROM_SIZE ];
81 :
82 : /* 角度関連 */

```

トレーニングボード 実習マニュアル(kit06 版)

```

83 : int          angle_buff;          /* 現在ハンドル角度保持用 */
84 :
85 : const revolution_difference[] = { /* 角度から内輪、外輪回転差計算 */
86 :     100, 99, 97, 96, 95,
87 :     93, 92, 91, 89, 88,
88 :     87, 86, 84, 83, 82,
89 :     81, 79, 78, 77, 76,
90 :     75, 73, 72, 71, 70,
91 :     69, 67, 66, 65, 64,
92 :     62, 61, 60, 59, 58,
93 :     56, 55, 54, 52, 51,
94 :     50, 48, 47, 46, 44,
95 :     43 };

```

中略

```

270 : case 11:
271 :     /* 通常トレース */
272 :     if( check_crossline() ) { /* クロスラインチェック */
273 :         pattern = 21;
274 :         break;
275 :     }
276 :     if( check_rightline() ) { /* 右ハーフラインチェック */
277 :         pattern = 51;
278 :         break;
279 :     }
280 :     if( check_leftline() ) { /* 左ハーフラインチェック */
281 :         pattern = 61;
282 :         break;
283 :     }
284 :     switch( sensor_inp(MASK3_3) ) {
285 :         case 0x00:
286 :             /* センタ まっすぐ */
287 :             handle( 0 );
288 :             speed( 100 ,100 );
289 :             break;
290 :
291 :         case 0x04:
292 :             /* 微妙に左寄り 右へ微曲げ */
293 :             handle( 3 );
294 :             speed( 100 ,100 );
295 :             break;
296 :
297 :         case 0x06:
298 :             /* 少し左寄り 右へ小曲げ */
299 :             handle( 10 );
300 :             speed( 80 ,diff(80) );
301 :             break;
302 :
303 :         case 0x07:
304 :             /* 中くらい左寄り 右へ中曲げ */
305 :             handle( 15 );
306 :             speed( 50 ,diff(50) );
307 :             break;
308 :
309 :         case 0x03:
310 :             /* 大きく左寄り 右へ大曲げ */
311 :             handle( 25 );
312 :             speed( 30 ,diff(30) );
313 :             pattern = 12;
314 :             break;
315 :
316 :         case 0x20:
317 :             /* 微妙に右寄り 左へ微曲げ */
318 :             handle( -3 );
319 :             speed( 100 ,100 );
320 :             break;
321 :
322 :         case 0x60:
323 :             /* 少し右寄り 左へ小曲げ */
324 :             handle( -10 );
325 :             speed( diff(80) ,80 );
326 :             break;
327 :
328 :         case 0xe0:
329 :             /* 中くらい右寄り 左へ中曲げ */
330 :             handle( -15 );
331 :             speed( diff(50) ,50 );
332 :             break;
333 :
334 :         case 0xc0:
335 :             /* 大きく右寄り 左へ大曲げ */
336 :             handle( -25 );
337 :             speed( diff(30) ,30 );
338 :             pattern = 13;
339 :             break;
340 :
341 :         default:
342 :             break;
343 :     }
344 :     break;

```

345 :

中略

```

401 :     case 23:
402 :         /* クロスライン後のトレース、クランク検出 */
403 :         if( sensor_inp(MASK4_4)==0xf8 ) {
404 :             /* 左クランクと判断 左クランククリア処理へ */
405 :             led_out( 0x1 );
406 :             handle( -38 );
407 :             speed( 10 ,50 );
408 :             pattern = 31;
409 :             cnt1 = 0;
410 :             break;
411 :         }
412 :         if( sensor_inp(MASK4_4)==0x1f ) {
413 :             /* 右クランクと判断 右クランククリア処理へ */
414 :             led_out( 0x2 );
415 :             handle( 38 );
416 :             speed( 50 ,10 );
417 :             pattern = 41;
418 :             cnt1 = 0;
419 :             break;
420 :         }
421 :         switch( sensor_inp(MASK3_3) ) {
422 :             case 0x00:
423 :                 /* センタ まっすぐ */
424 :                 handle( 0 );
425 :                 speed( 40 ,40 );
426 :                 break;
427 :             case 0x04:
428 :             case 0x06:
429 :             case 0x07:
430 :             case 0x03:
431 :                 /* 左寄り 右曲げ */
432 :                 handle( 8 );
433 :                 speed( 40 ,diff(40) );
434 :                 break;
435 :             case 0x20:
436 :             case 0x60:
437 :             case 0xe0:
438 :             case 0xc0:
439 :                 /* 右寄り 左曲げ */
440 :                 handle( -8 );
441 :                 speed( diff(40) ,40 );
442 :                 break;
443 :         }
444 :         break;

```

中略

```

497 :     case 53:
498 :         /* 右ハーフライン後のトレース、レーンチェンジ */
499 :         if( sensor_inp(MASK4_4) == 0x00 ) {
500 :             handle( 15 );
501 :             speed( 40 ,diff(40) );
502 :             pattern = 54;
503 :             cnt1 = 0;
504 :             break;
505 :         }
506 :         switch( sensor_inp(MASK3_3) ) {
507 :             case 0x00:
508 :                 /* センタ まっすぐ */
509 :                 handle( 0 );
510 :                 speed( 40 ,40 );
511 :                 break;
512 :             case 0x04:
513 :             case 0x06:
514 :             case 0x07:
515 :             case 0x03:
516 :                 /* 左寄り 右曲げ */
517 :                 handle( 8 );
518 :                 speed( 40 ,diff(40) );
519 :                 break;
520 :             case 0x20:
521 :             case 0x60:
522 :             case 0xe0:
523 :             case 0xc0:
524 :                 /* 右寄り 左曲げ */
525 :                 handle( -8 );
526 :                 speed( diff(40) ,40 );
527 :                 break;
528 :             default:
529 :                 break;
530 :         }
531 :         break;

```

中略

```

559 :     case 63:
560 :         /* 左ハーフライン後のトレース、レーンチェンジ */

```

## トレーニングボード 実習マニュアル(kit06 版)

```

561 :         if( sensor_inp(MASK4_4) == 0x00 ) {
562 :             handle( -15 );
563 :             speed( diff(40) ,40 );
564 :             pattern = 64;
565 :             cnt1 = 0;
566 :             break;
567 :         }
568 :         switch( sensor_inp(MASK3_3) ) {
569 :             case 0x00:
570 :                 /* センタ まっすぐ */
571 :                 handle( 0 );
572 :                 speed( 40 ,40 );
573 :                 break;
574 :             case 0x04:
575 :             case 0x06:
576 :             case 0x07:
577 :             case 0x03:
578 :                 /* 左寄り 右曲げ */
579 :                 handle( 8 );
580 :                 speed( 40 ,diff(40) );
581 :                 break;
582 :             case 0x20:
583 :             case 0x60:
584 :             case 0xe0:
585 :             case 0xc0:
586 :                 /* 右寄り 左曲げ */
587 :                 handle( -8 );
588 :                 speed( diff(40) ,40 );
589 :                 break;
590 :             default:
591 :                 break;
592 :         }
593 :         break;

```

中略

```

837 : /*****/
838 : /* サーボハンドル操作 */
839 : /* 引数 サーボ操作角度: -90~90 */
840 : /* -90で左へ90度、0でまっすぐ、90で右へ90度回転 */
841 : /*****/
842 : void handle( int angle )
843 : {
844 :     angle_buff = angle; /* 現在の角度保存 */
845 :     ITU4_BRB = eep_buff[EEPROM_SERVO] - angle * HANDLE_STEP;
846 : }

```

中略

```

866 : /*****/
867 : /* 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 */
868 : /* 引数 外輪PWM */
869 : /* 戻り値 内輪PWM */
870 : /*****/
871 : int diff( int pwm )
872 : {
873 :     int ret;
874 :
875 :     if( pwm >= 0 ) {
876 :         /* PWM値が正の数なら */
877 :         if( angle_buff < 0 ) {
878 :             angle_buff = -angle_buff;
879 :         }
880 :         ret = revolution_difference[angle_buff] * pwm / 100;
881 :     } else {
882 :         /* PWM値が負の数なら */
883 :         ret = pwm; /* そのまま返す */
884 :     }
885 :     return ret;
886 : }

```

以下、略

## 12.4 プログラムの解説

### 12.4.1 プロトタイプの追加

```
70 : int diff( int pwm );
```

diff という関数を追加するので、プロトタイプ宣言します。ちなみに diff とは、「difference」の略で「差」という意味です。

### 12.4.2 大域変数の追加、handle 関数内の追加

```
82 : /* 角度関連 */
83 : int          angle_buff;          /* 現在ハンドル角度保持用 */
```

angle\_buff 変数を追加します。これは、後ほど説明する左右回転差を計算する関数に、現在のハンドル角度を知らせるための変数です。

```
837 : /******/
838 : /* サーボハンドル操作 */
839 : /* 引数   サーボ操作角度：-90～90 */
840 : /*      -90で左へ90度、0でまっすぐ、90で右へ90度回転 */
841 : /******/
842 : void handle( int angle )
843 : {
844 :     angle_buff = angle;          /* 現在の角度保存 */
845 :     ITU4_BRB = eep_buff[EEPROM_SERVO] - angle * HANDLE_STEP;
846 : }
```

844 行で、角度を angle\_buff 変数に保存しています。diff 関数で使用するための準備です。

### 12.4.3 配列の追加

```

85 :  const revolution_difference[] = {          /* 角度から内輪、外輪回転差計算 */
86 :      100, 99, 97, 96, 95,
87 :      93, 92, 91, 89, 88,
88 :      87, 86, 84, 83, 82,
89 :      81, 79, 78, 77, 76,
90 :      75, 73, 72, 71, 70,
91 :      69, 67, 66, 65, 64,
92 :      62, 61, 60, 59, 58,
93 :      56, 55, 54, 52, 51,
94 :      50, 48, 47, 46, 44,
95 :      43 };
    
```

revolution\_difference という回転の差を計算した配列を追加します。この配列には、const を先頭に付けています。値を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ROM エリアに配置されます。H8/3048F-ONE はフラッシュ ROM が 128KB、RAM が 4KB と、RAM 容量が少ないので RAM の有効活用を考えて const を付けました。

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] = 99
revolution_difference[ 2] = 97
      ⋮
revolution_difference[45] = 43
    
```

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プログラマが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 の時、97 が返ってきます。これは外輪 100、ハンドル角度が 2 度するとき、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 度の時の内輪の値を、あらかじめエクセルなどで計算しておきます。計算方法は後述します。

#### 12.4.4 diff 関数の追加

```

866 : /*****/
867 : /* 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 */
868 : /* 引数 外輪PWM */
869 : /* 戻り値 内輪PWM */
870 : /*****/
871 : int diff( int pwm )
872 : {
873 :     int ret;
874 :
875 :     if( pwm >= 0 ) {
876 :         /* PWM値が正の数なら */
877 :         if( angle_buff < 0 ) {
878 :             angle_buff = -angle_buff;
879 :         }
880 :         ret = revolution_difference[angle_buff] * pwm / 100;
881 :     } else {
882 :         /* PWM値が負の数なら */
883 :         ret = pwm; /*そのまま返す */
884 :     }
885 :     return ret;
886 : }

```

diff 関数は、引数に外輪(多く回るタイヤ側)の PWM 値を入れて呼び出すと、内輪(少なく回るタイヤ側)の PWM 値が返って来るとい関数です。現在のハンドル角度は、angle\_buff 変数から読み込みます。

例えば、今のハンドル角度が-25 度、外輪の PWM 値を 60 とするとします。

まず、875 行で角度がマイナスかどうかチェックします。-25 度なのでマイナスです。878 行でプラスに直します。

次に回転差を計算します。外輪の回転数を 100 と考えて、内輪の回転数を計算します。

```

ret = revolution_difference[angle_buff]
    = revolution_difference[ 25 ]
    = 69

```

で内輪の回転数が出ます。

次に、外輪が 100 では無い場合、内輪は外輪の回転に比例しますので、

```

ret = 69 * 外輪の PWM 値 / 100
    = 69 * 60 / 100
    = 41

```

となります。結果、ハンドル角度が-25 度、外輪(右)の PWM 値を 60 とすると内輪(左)の PWM 値は 41 となります。



### 12.4.5 プログラムでの使い方

まず、ハンドル角度を設定します。

```
handle( 15 );
```

handle 関数は正の数で右へ、負の数で左へハンドルを切る関数ですので、15 度というと右側へハンドルを切っています。そのため、外輪が左タイヤ、内輪が右タイヤとなります。

次に speed 関数です。PWM 値を 50%にしたい場合、外輪の左タイヤを 50 に、内輪の右タイヤを diff(50)とします。どちらが外輪で、どちらが内輪かは、プログラムが判断して diff 関数を使います。

```
speed( 50 ,diff(50) );

    外輪   内輪
```

diff(50)の戻り値は、

$$\begin{aligned} & \text{revolution\_difference}[\text{angle\_buff}] * \text{pwm} / 100 \\ = & \text{revolution\_difference}[15] * 50 / 100 \\ = & 81 * 50 / 100 \\ = & 40 \end{aligned}$$

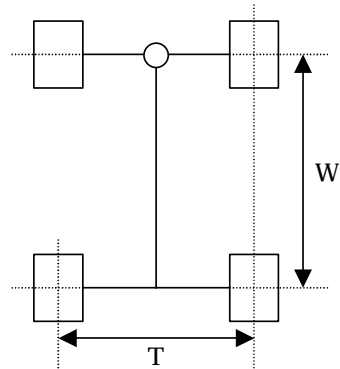
となります。

ハンドル角度を変えると、自動で内輪側の PWM 値を計算してくれるので便利です。

### 12.4.6 配列データの作り方

数値の計算方法は、「23. カーブでのタイヤの左右回転差の計算方法」を参照してください。  
「角度計算.xls」ファイルを開きます。

	度	rad	r2	r1	r3	r1/r3*100
0	0	0				100
1	1	0.017	10.031	9.961	10.101	99
2	2	0.035	5.014	4.944	5.084	97
3	3	0.052	3.341	3.271	3.411	96
4	4	0.070	2.504	2.434	2.574	95
5	5	0.087	2.001	1.931	2.071	93
6	6	0.105	1.666	1.596	1.736	92
7	7	0.122	1.426	1.356	1.496	91
8	8	0.140	1.246	1.176	1.316	89
9	9	0.157	1.105	1.035	1.175	88
10	10	0.174	0.993	0.923	1.063	87



T = トレッド  
左右輪の中心線の距離

W = ホイールベース  
前輪と後輪の間隔

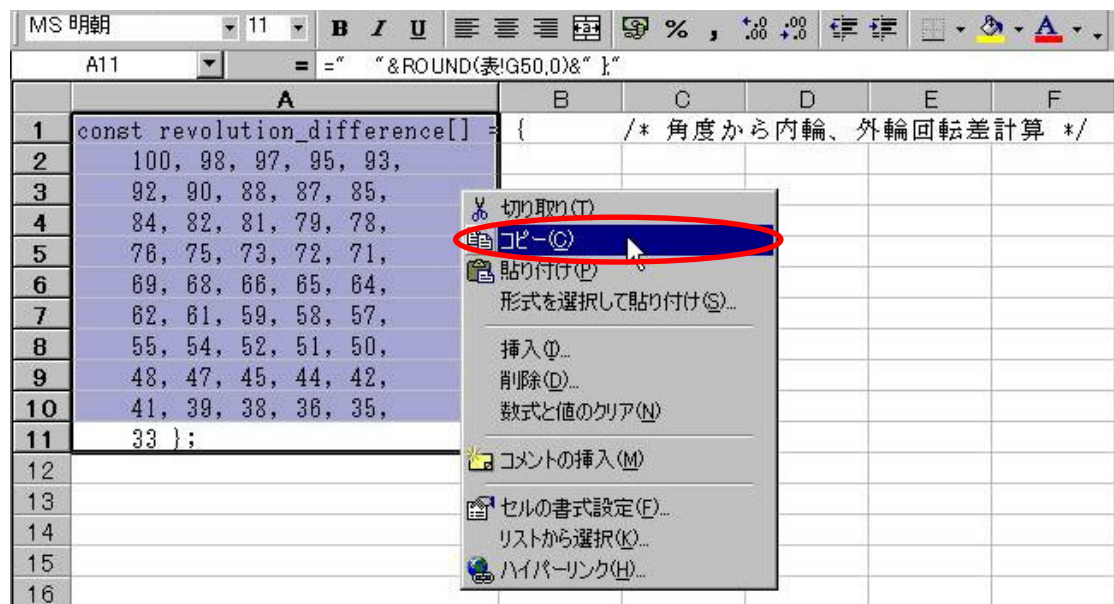
W 欄にはホイールベース、T 欄にはトレッドを入れます。自分のマイコンカーのそれぞれの長さを測って、メートル単位で入力します。

あるマイコンカーの長さを測ると、 $W=0.15\text{m}$ 、 $T=0.15\text{m}$ でした。入力すると、自動で0～45度にハンドルを切ったときの内輪回転数が計算されます。

	A	B	C	D	E	F	G
1	W		0.15	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	8.598	8.523	8.673	98
7		2	0.035	4.298	4.223	4.373	97
8		3	0.052	2.864	2.789	2.939	95
9		4	0.070	2.146	2.071	2.221	93
10		5	0.087	1.715	1.640	1.790	92
11		6	0.105	1.428	1.353	1.503	90
12		7	0.122	1.222	1.147	1.297	88
13		8	0.140	1.068	0.993	1.143	87
14		9	0.157	0.948	0.873	1.023	85
15		10	0.174	0.851	0.776	0.926	84
16		11	0.192	0.772	0.697	0.847	82
17		12	0.209	0.706	0.631	0.781	81
18		13	0.227	0.650	0.575	0.725	79
19		14	0.244	0.602	0.527	0.677	78
20		15	0.262	0.560	0.485	0.635	76
21		16	0.279	0.523	0.448	0.598	75
22		17	0.297	0.491	0.416	0.566	73
23		18	0.314	0.462	0.387	0.537	72
24		19	0.331	0.436	0.361	0.511	71
25		20	0.349	0.412	0.337	0.487	69
26		21	0.366	0.391	0.316	0.466	68
27		22	0.384	0.371	0.296	0.446	66
28		23	0.401	0.354	0.279	0.429	65
29		24	0.419	0.337	0.262	0.412	64
30		25	0.436	0.322	0.247	0.397	62

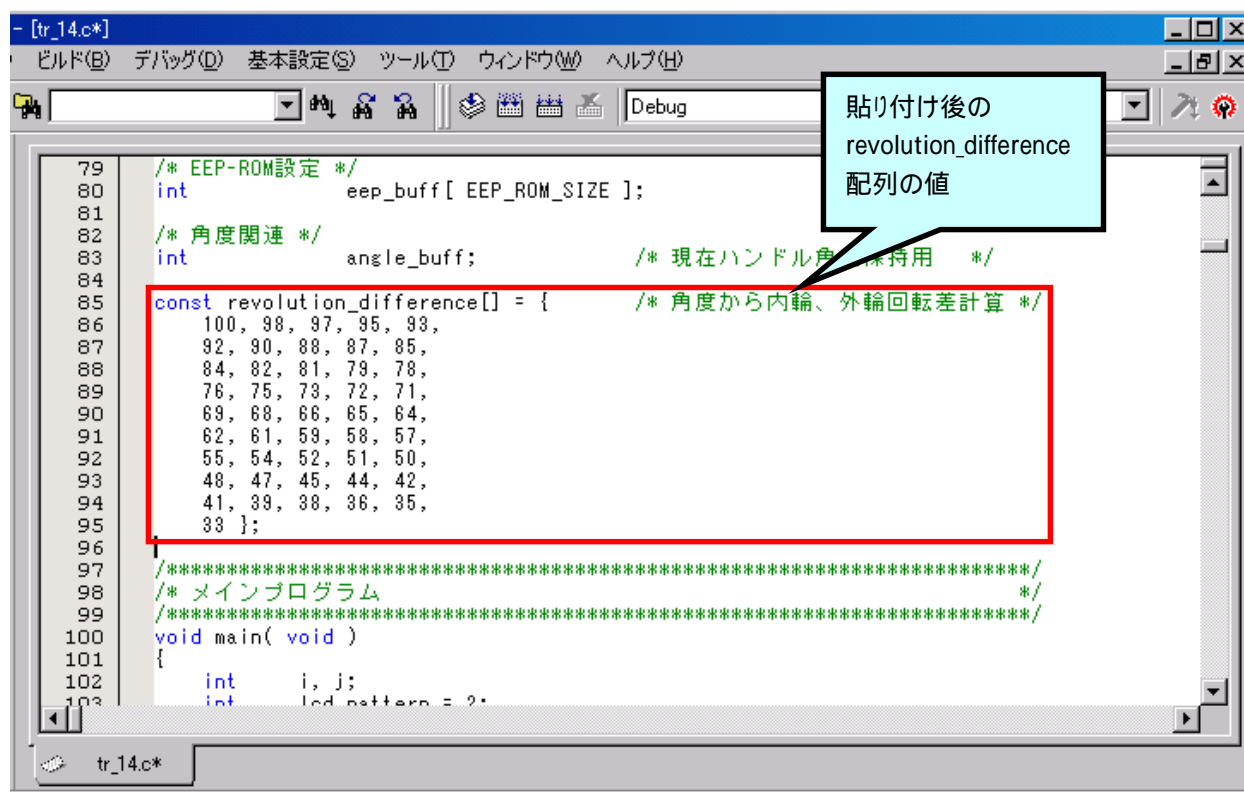
自動で再計算される

上記の「コピーして貼り付け用」タブをクリックします。



A1～A11 を選択して、「右クリック コピー」で内容をコピーします。

「tr\_14.c」を開きます。元の revolution\_difference 配列を消して、CTRL+Vで新しいデータを貼り付けます。



```
79  /* EEPROM設定 */
80  int    eep_buff[ EEPROM_SIZE ];
81
82  /* 角度関連 */
83  int    angle_buff;          /* 現在ハンドル角保持用 */
84
85  const revolution_difference[] = { /* 角度から内輪、外輪回転差計算 */
86      100, 98, 97, 95, 93,
87      92, 90, 88, 87, 85,
88      84, 82, 81, 79, 78,
89      76, 75, 73, 72, 71,
90      69, 68, 66, 65, 64,
91      62, 61, 59, 58, 57,
92      55, 54, 52, 51, 50,
93      48, 47, 45, 44, 42,
94      41, 39, 38, 36, 35,
95      33 };
96
97  /*-----*/
98  /* メインプログラム */
99  /*-----*/
100 void main( void )
101 {
102     int    i, j;
103     int    led_pattern = 2;
```

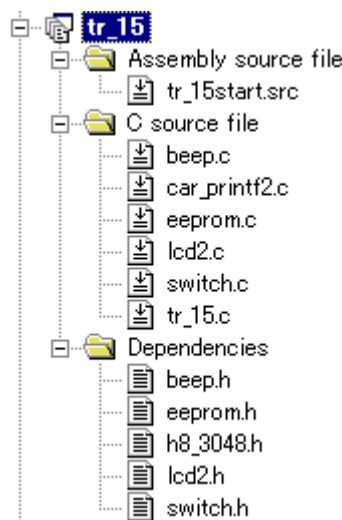
これで、自分のマイコンカーに合った左右回転差が計算されます。

## 13. プロジェクト「tr\_15」 大カーブでのセンサ状態の追加

### 13.1 内容

大カーブでのセンサ状態は右へ大曲げがハンドル 25 度、PWM30%、左へ大曲げがハンドル-25 度、PWM30% だけでした。そこで、大曲げのセンサ状態を増やして、ハンドルの切れ角、スピードの減速の段階を増やします。

### 13.2 プロジェクトの構成

	<ul style="list-style-type: none"> <li>·tr_15start.src ベクタアドレスの設定、スタートアップルーチンが記述されています。</li> <li>·car_printf2.c セクションの初期化、printf 文、scanf 文を実行します。</li> <li>·lcd2.c LCD 制御を行います。</li> <li>·switch.c スイッチ制御を行います。</li> <li>·beep.c ブザー制御を行います。</li> <li>·eeprom.c EEP-ROM 制御を行います。</li> <li>·tr_15.c メインプログラムです。</li> </ul>
--	---

### 13.3 プログラム「tr\_15.c」

プログラムのゴシック体部分が追加した部分です。

前略

```

346 :     case 12:
347 :         /* 右へ大曲げの終わりのチェック */
348 :         if( check_crossline() ) { /* 大曲げ中もクロスラインチェック */
349 :             pattern = 21;
350 :             break;
351 :         }
352 :         if( check_rightline() ) { /* 右ハーフラインチェック */
353 :             pattern = 51;
354 :             break;
355 :         }
356 :         if( check_leftline() ) { /* 左ハーフラインチェック */
357 :             pattern = 61;
358 :             break;
359 :         }
360 :         switch( sensor_inp(MASK3_3) ) {
361 :             case 0x06:
362 :                 pattern = 11;
363 :                 break;
364 :             case 0x03:
365 :                 handle( 20 );
366 :                 speed( 60, diff(60) );
367 :                 break;
368 :             case 0x81:
369 :                 handle( 25 );
370 :                 speed( 50, diff(50) );
371 :                 break;
372 :             case 0xc1:
373 :             case 0xc0:
374 :                 handle( 30 );
375 :                 speed( 40, diff(40) );
376 :                 break;
377 :             case 0x60:

```

トレーニングボード 実習マニュアル(kit06 版)

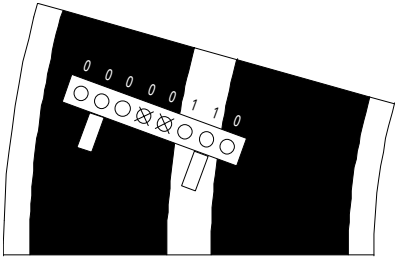
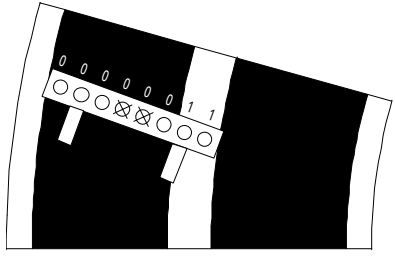
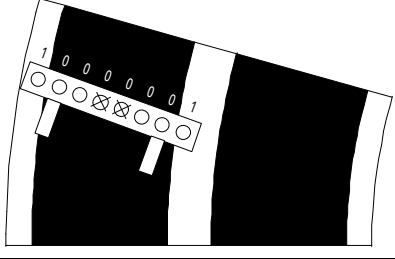
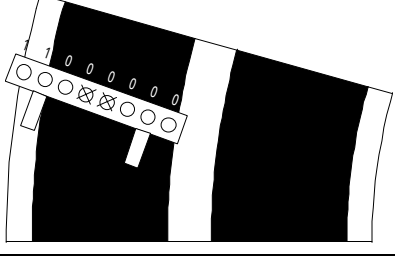
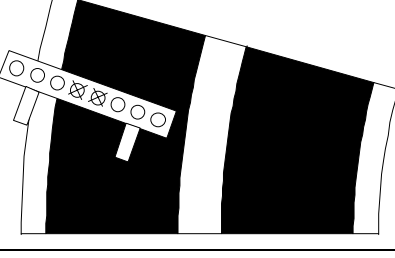
```
378 :         handle( 35 );
379 :         speed( 0, 0 );
380 :         break;
381 :     }
382 :     break;
383 :
384 : case 13:
385 :     /* 左へ大曲げの終わりのチェック */
386 :     if( check_crossline() ) { /* 大曲げ中もクロスラインチェック */
387 :         pattern = 21;
388 :         break;
389 :     }
390 :     if( check_rightline() ) { /* 右ハーフラインチェック */
391 :         pattern = 51;
392 :         break;
393 :     }
394 :     if( check_leftline() ) { /* 左ハーフラインチェック */
395 :         pattern = 61;
396 :         break;
397 :     }
398 :     switch( sensor_inp(MASK3_3) ) {
399 :     case 0x60:
400 :         pattern = 11;
401 :         break;
402 :     case 0xc0:
403 :         handle( -20 );
404 :         speed( diff(60), 60 );
405 :         break;
406 :     case 0x81:
407 :         handle( -25 );
408 :         speed( diff(50), 50 );
409 :         break;
410 :     case 0x83:
411 :     case 0x03:
412 :         handle( -30 );
413 :         speed( diff(40), 40 );
414 :         break;
415 :     case 0x06:
416 :         handle( -35 );
417 :         speed( 0, 0 );
418 :         break;
419 :     }
420 :     break;
```

以下、略

## 13.4 プログラムの解説

### 13.4.1 パターン 12 右大曲げのパターン

右大曲げ時のセンサとハンドル、PWM 値の関係です。

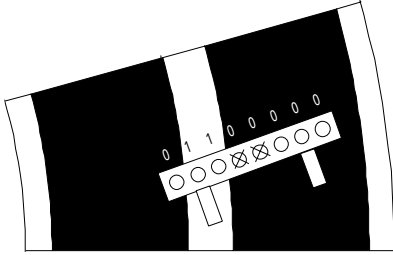
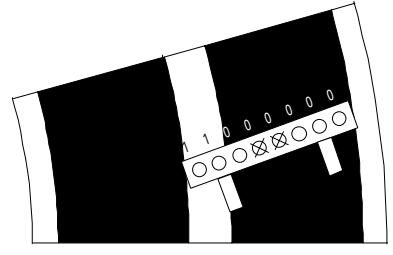
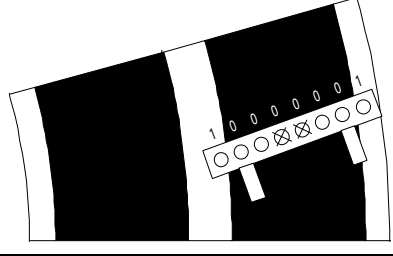
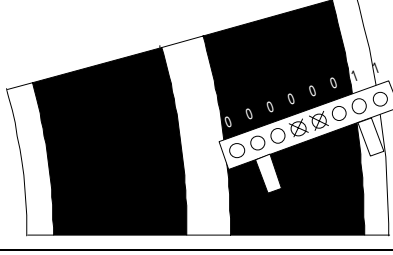
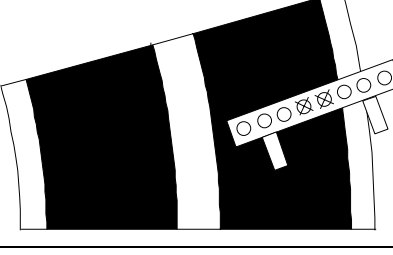
センサ状態	図	ハンドル角	PWM 値
0x06		パターン 11 に戻る	
0x03		20 度	60%
0x81		25 度	50%
0xc1 0xc0		30 度	40%
0x60		35 度	0%

今までは 0x03 の 1 種類だけでしたが、4 種類を増やしてカーブで滑らかに走行するようにします。

0x60 は一番大きい最後のセンサ状態なので、PWM は 0% にします。ただし、センサ状態 0x60 が続くと PWM が 0% のままなので、0x60 が一定時間以上続いたら、モータを回すようにした方が良くもかもしれません。

13.4.2 パターン 13 左大曲げのパターン

左大曲げ時のセンサとハンドル、PWM 値の関係です。

センサ状態	図	ハンドル角	PWM 値
0x60		パターン 11 に戻る	
0xc0		-20 度	60%
0x81		-25 度	50%
0x83 0x03		-30 度	40%
0x06		-35 度	0%

今までは 0xc0 の 1 種類だけでしたが、4 種類を増やしてカーブで滑らかに走行するようにします。

0x06 は一番大きい最後のセンサ状態なので、PWM は 0% にします。ただし、センサ状態 0x06 が続くと PWM が 0% のままなので、0x06 が一定時間以上続いたら、モータを回すようにした方が良くもかもしれません。

## 14. プロジェクト「tr\_16」 大カーブでの PWM 値の調整

### 14.1 内容

大カーブでは脱輪しないように減速しますが、減速しすぎるとタイムロスにつながります。そのため、大カーブでの PWM 値を調整できるようにして、最適な減速値を見つけます。

### 14.2 プロジェクトの構成

	<ul style="list-style-type: none"> <li>・tr_16start.src ベクタアドレスの設定、スタートアップルーチンが記述されています。</li> <li>・car_printf2.c セクションの初期化、printf 文、scanf 文を実行します。</li> <li>・lcd2.c LCD 制御を行います。</li> <li>・switch.c スイッチ制御を行います。</li> <li>・beep.c ブザー制御を行います。</li> <li>・eeprom.c EEP-ROM 制御を行います。</li> <li>・tr_16.c メインプログラムです。</li> </ul>
--	---

### 14.3 プログラム「tr\_16.c」

プログラムのゴシック体部分が追加した部分です。

前略

```

47 : /* EEPROM関連 */
48 : #define      EEPROM_SIZE      16      /* EEPROM使用サイズ      */
49 :
50 : #define      EEPROM_CHECK      0x00   /* EEPROMチェック      */
51 : #define      EEPROM_SERVO      0x01   /* サーボセンタ値      */
52 : #define      EEPROM_PWM        0x02   /* PWM値      */
53 : #define      EEPROM_CURVE_PWM  0x03   /* 大カーブのPWM      */

```

中略

```

98 : /****** */
99 : /* メインプログラム */
100 : /****** */
101 : void main( void )
102 : {
103 :     int    i, j;
104 :     int    lcd_pattern = 2;
105 :
106 :     /* マイコン機能の初期化 */
107 :     init();                /* 初期化      */
108 :     set_ccr( 0x00 );       /* 全体割り込み許可      */
109 :     initLcd();            /* LCD初期化      */
110 :     initSwitch();         /* スイッチ初期化      */
111 :     initBeep();           /* ブザー初期化      */
112 :     initEeprom();         /* EEPROM初期化      */
113 :
114 :     /*EEP-ROMのチェック */
115 :     if( readEeprom( EEPROM_CHECK ) != 0x2006 ) {
116 :         /*
117 :         IDのチェック EEPROMを初めて使うかどうか
118 :         0x00番地にIDが書かれていなければ初めて使うと判断して初期化する
119 :         */

```



トレーニングボード 実習マニュアル(kit06 版)

```

120 :         eep_buff[EEPROM_CHECK]      = 0x2006;
121 :         eep_buff[EEPROM_SERVO]      = SERVO_CENTER;
122 :         eep_buff[EEPROM_PWM]        = 50;
123 :         eep_buff[EEPROM_CURVE_PWM]  = 50;
124 :     } else {
125 :         /* 2回目以降の使用の場合、データ読み込み */
126 :         for( i=0; i<EEP_ROM_SIZE; i++ ) {
127 :             eep_buff[ i ] = readEeprom( i );
128 :         }
129 :     }
130 :
131 :     /* マイコンカーの状態初期化 */
132 :     handle( 0 );
133 :     speed( 0, 0 );
134 :
135 :     while( 1 ) {
136 :         switch( pattern ) {
137 :
138 :             /******
139 :             パターンについて
140 :             0: スイッチ入力待ち
141 :             1: スタートバーが開いたかチェック
142 :             11: 通常トレース
143 :             12: 右へ大曲げの終わりのチェック
144 :             13: 左へ大曲げの終わりのチェック
145 :             21: 1本目のクロスライン検出時の処理
146 :             22: 2本目を読み飛ばす
147 :             23: クロスライン後のトレース、クランク検出
148 :             31: 左クランククリア処理 安定するまで少し待つ
149 :             32: 左クランククリア処理 曲げ終わりのチェック
150 :             41: 右クランククリア処理 安定するまで少し待つ
151 :             42: 右クランククリア処理 曲げ終わりのチェック
152 :             51: 1本目の右ハーフライン検出時の処理
153 :             52: 2本目を読み飛ばす
154 :             53: 右ハーフライン後のトレース
155 :             54: 右レーンチェンジ終了のチェック
156 :             61: 1本目の左ハーフライン検出時の処理
157 :             62: 2本目を読み飛ばす
158 :             63: 左ハーフライン後のトレース
159 :             64: 左レーンチェンジ終了のチェック
160 :             *****/
161 :
162 :         case 0:
163 :             /* スイッチ入力待ち */
164 :             if( pushsw_get() ) {
165 :                 setBeepPattern( 0xc000 );
166 :                 /* 保存 */
167 :                 for( i=0; i<EEP_ROM_SIZE; i++ ) {
168 :                     writeEeprom( i, eep_buff[ i ] );
169 :                     while( !checkEeprom() ); /* 書き込み終了チェック */
170 :                 }
171 :                 pattern = 1;
172 :                 cnt1 = 0;
173 :                 break;
174 :             }
175 :
176 :             /* スイッチ4 設定値保存 */
177 :             if( getSwFlag(SW_4) ) {
178 :                 setBeepPattern( 0x8000 );
179 :                 /* 保存 */
180 :                 for( i=0; i<EEP_ROM_SIZE; i++ ) {
181 :                     writeEeprom( i, eep_buff[ i ] );
182 :                     while( !checkEeprom() ); /* 書き込み終了チェック */
183 :                 }
184 :                 break;
185 :             }
186 :             /* スイッチ3 メニュー + 1 */
187 :             if( getSwFlag(SW_3) ) {
188 :                 lcd_pattern++;
189 :                 if( lcd_pattern == 4 ) lcd_pattern = 1;
190 :             }
191 :             /* スイッチ2 メニュー - 1 */
192 :             if( getSwFlag(SW_2) ) {
193 :                 lcd_pattern--;
194 :                 if( lcd_pattern == 0 ) lcd_pattern = 3;
195 :             }
196 :
197 :             /* スイッチ、LCD処理 */
198 :             switch( lcd_pattern ) {
199 :             case 1:
200 :                 /* サーボセンタ値調整 */
201 :                 中略
220 :                 break;
221 :
222 :             case 2:
223 :                 /* PWM調整 */
224 :                 中略
242 :                 break;
243 :

```

```

244 :     case 3:
245 :         /* 大カーブPWM調整 */
246 :         i = eep_buff[EEPROM_CURVE_PWM];
247 :         if( getSwFlag(SW_1) ) {
248 :             i++;
249 :             if( i > 100 ) i = 100;
250 :         }
251 :         if( getSwFlag(SW_0) ) {
252 :             i--;
253 :             if( i < 0 ) i = 0;
254 :         }
255 :         eep_buff[EEPROM_CURVE_PWM] = i;
256 :
257 :         /* LCD処理 */
258 :         lcdPosition( 0, 0 );
259 :         /* 0123456789ab..def 1行16文字 */
260 :         lcdPrintf( "3 curve = %03d ", i );
261 :         /* 01234567..89abcde.f 1行16文字 */
262 :         lcdPrintf( "sensor=%02x bar=%d ",
263 :             sensor_inp( 0xff ), startbar_get() );
264 :         break;
265 :     }
266 :
267 :     if( cnt1 < 100 ) { /* LED点滅処理 */
268 :         led_out( 0x1 );
269 :     } else if( cnt1 < 200 ) {
270 :         led_out( 0x2 );
271 :     } else {
272 :         cnt1 = 0;
273 :     }
274 :     break;

```

中略

```

370 :     case 12:
371 :         /* 右へ大曲げの終わりのチェック */
372 :         if( check_crossline() ) { /* 大曲げ中もクロスラインチェック */
373 :             pattern = 21;
374 :             break;
375 :         }
376 :         if( check_rightline() ) { /* 右ハーフラインチェック */
377 :             pattern = 51;
378 :             break;
379 :         }
380 :         if( check_leftline() ) { /* 左ハーフラインチェック */
381 :             pattern = 61;
382 :             break;
383 :         }
384 :         switch( sensor_inp(MASK3_3) ) {
385 :             case 0x06:
386 :                 pattern = 11;
387 :                 break;
388 :             case 0x03:
389 :                 handle( 20 );
390 :                 i = eep_buff[EEPROM_CURVE_PWM];
391 :                 speed( i, diff(i) );
392 :                 break;
393 :             case 0x81:
394 :                 handle( 25 );
395 :                 i = eep_buff[EEPROM_CURVE_PWM];
396 :                 speed( i, diff(i) );
397 :                 break;
398 :             case 0xc1:
399 :             case 0xc0:
400 :                 handle( 30 );
401 :                 i = eep_buff[EEPROM_CURVE_PWM];
402 :                 speed( i, diff(i) );
403 :                 break;
404 :             case 0x60:
405 :                 handle( 35 );
406 :                 speed( 0, 0 );
407 :                 break;
408 :         }
409 :         break;
410 :
411 :     case 13:
412 :         /* 左へ大曲げの終わりのチェック */
413 :         if( check_crossline() ) { /* 大曲げ中もクロスラインチェック */
414 :             pattern = 21;
415 :             break;
416 :         }
417 :         if( check_rightline() ) { /* 右ハーフラインチェック */
418 :             pattern = 51;
419 :             break;
420 :         }
421 :         if( check_leftline() ) { /* 左ハーフラインチェック */
422 :             pattern = 61;
423 :             break;
424 :         }
425 :         switch( sensor_inp(MASK3_3) ) {
426 :             case 0x60:

```

```

427 :         pattern = 11;
428 :         break;
429 :     case 0xc0:
430 :         handle( -20 );
431 :         i = eep_buff[EEPROM_CURVE_PWM];
432 :         speed( diff(i), i );
433 :         break;
434 :     case 0x81:
435 :         handle( -25 );
436 :         i = eep_buff[EEPROM_CURVE_PWM];
437 :         speed( diff(i), i );
438 :         break;
439 :     case 0x83:
440 :     case 0x03:
441 :         handle( -30 );
442 :         i = eep_buff[EEPROM_CURVE_PWM];
443 :         speed( diff(i), i );
444 :         break;
445 :     case 0x06:
446 :         handle( -35 );
447 :         speed( 0, 0 );
448 :         break;
449 :     }
450 :     break;

```

以下、略

## 14.4 プログラムの解説

### 14.4.1 EEP-ROM エリアの追加

```

47 : /* EEP-ROM 関連 */
48 : #define    EEP_ROM_SIZE    16    /* EEP-ROM 使用サイズ    */
49 :
50 : #define    EEPROM_CHECK    0x00    /* EEP-ROM チェック    */
51 : #define    EEPROM_SERVO    0x01    /* サーボセンタ値    */
52 : #define    EEPROM_PWM    0x02    /* PWM 値    */
53 : #define    EEPROM_CURVE_PWM    0x03    /* 大カーブの PWM    */

```

PWM を設定する EEP-ROM の番地を決めます。0x02 番地まで使っていたので、0x03 番地を EEPROM\_CURVE\_PWM として定義します。

番地	番地名	内容	初期値
0x00	EEPROM_CHECK	EEP-ROM チェック用 0x2006 で無ければ、初期値を読み込む	0x2006
0x01	EEPROM_SERVO	サーボセンタ値	5000
0x02	EEPROM_PWM	PWM 値	50
<b>0x03</b>	<b>EEPROM_CURVE_PWM</b>	<b>大カーブの PWM 値</b>	<b>50</b>
0x04 ~ 0x0f	未定義		

#### 14.4.2 EEPROM から読み込み

```

114 :      /*EEP-ROM のチェック */
115 :      if( readEeprom( EEPROM_CHECK ) != 0x2006 ) {
116 :          /*
117 :           ID のチェック EEPROM を初めて使うかどうか
118 :           0x00 番地に ID が書かれていなければ初めて使うと判断して初期化する
119 :          */
120 :          eep_buff[EEPROM_CHECK]      = 0x2006;
121 :          eep_buff[EEPROM_SERVO]      = SERVO_CENTER;
122 :          eep_buff[EEPROM_PWM]        = 50;
123 :          eep_buff[EEPROM_CURVE_PWM] = 50;
124 :      } else {
125 :          /* 2 回目以降の使用の場合、データ読み込み */
126 :          for( i=0; i<EEP_ROM_SIZE; i++ ) {
127 :              eep_buff[ i ] = readEeprom( i );
128 :          }
129 :      }

```

123 行に eep\_buff[EEPROM\_CURVE\_PWM]変数の初期化を追加しています。

#### 14.4.3 スイッチ入力待ち

```

186 :      /* スイッチ 3 メニュー + 1 */
187 :      if( getSwFlag(SW_3) ) {
188 :          lcd_pattern++;
189 :          if( lcd_pattern == 4 ) lcd_pattern = 1;
190 :      }
191 :      /* スイッチ 2 メニュー - 1 */
192 :      if( getSwFlag(SW_2) ) {
193 :          lcd_pattern--;
194 :          if( lcd_pattern == 0 ) lcd_pattern = 3;
195 :      }

```

SW\_3 で LCD に表示する内容を次に進めます。SW\_2 で戻します。lcd\_pattern=3 を大カーブの PWM 値の調整として追加します。

189 行で上限のチェックです。lcd\_pattern の 4 はありませんので 1 にします。

194 行で下限のチェックです。lcd\_pattern の 0 はありませんので 3 にします。

lcd_pattern	内容
1	サーボセンタ値の調整
2	PWM 値の調整
3	<b>大カーブの PWM 値の調整</b>

```

197 :      /* スイッチ、LCD処理 */
198 :      switch( lcd_pattern ) {
中略
244 :      case 3:
245 :          /* 大カーブPWM調整 */
246 :          i = eep_buff[EEPROM_CURVE_PWM];
247 :          if( getSwFlag(SW_1) ) {
248 :              i++;
249 :              if( i > 100 ) i = 100;
250 :          }
251 :          if( getSwFlag(SW_0) ) {
252 :              i--;
253 :              if( i < 0 ) i = 0;
254 :          }
255 :          eep_buff[EEPROM_CURVE_PWM] = i;
256 :
257 :          /* LCD処理 */
258 :          lcdPosition( 0, 0 );
259 :          /* 0123456789ab..def 1行16文字 */
260 :          lcdPrintf( "3 curve = %03d ", i );
261 :          /* 01234567..89abcde.f 1行16文字 */
262 :          lcdPrintf( "sensor=%02x bar=%d ",
263 :                  sensor_inp( 0xff ), startbar_get() );
264 :          break;
265 :      }

```

198行で、lcd\_patternの番号によりプログラムをジャンプします。3なら244行へ飛びます。

246行で、eep\_buff[EEPROM\_CURVE\_PWM]変数の値をいったんi変数に代入します。

247行で、SW\_1が押されているかチェックします。押されていれば、i変数を1つ増加させます。

251行で、SW\_0が押されているかチェックします。押されていれば、i変数を1つ減少させます。

255行で、i変数の値をeep\_buff[EEPROM\_CURVE\_PWM]変数へ代入します。SW\_1、SW\_2で操作された値が代入されます。

258行からLCDに大カーブでのPWM値を表示します。2行目が余っているのでセンサの状態を表示させています。

14.4.4 パターン 12 右大曲げ

```

370 :     case 12:
371 :         /* 右へ大曲げの終わりのチェック */
372 :         if( check_crossline() ) {          /* 大曲げ中もクロスラインチェック */
373 :             pattern = 21;
374 :             break;
375 :         }
376 :         if( check_rightline() ) {          /* 右ハーフラインチェック */
377 :             pattern = 51;
378 :             break;
379 :         }
380 :         if( check_leftline() ) {          /* 左ハーフラインチェック */
381 :             pattern = 61;
382 :             break;
383 :         }
384 :         switch( sensor_inp(MASK3_3) ) {
385 :         case 0x06:
386 :             pattern = 11;
387 :             break;
388 :         case 0x03:
389 :             handle( 20 );
390 :             i = eep_buff[EEPROM_CURVE_PWM];
391 :             speed( i, diff(i) );
392 :             break;
393 :         case 0x81:
394 :             handle( 25 );
395 :             i = eep_buff[EEPROM_CURVE_PWM];
396 :             speed( i, diff(i) );
397 :             break;
398 :         case 0xc1:
399 :         case 0xc0:
400 :             handle( 30 );
401 :             i = eep_buff[EEPROM_CURVE_PWM];
402 :             speed( i, diff(i) );
403 :             break;
404 :         case 0x60:
405 :             handle( 35 );
406 :             speed( 0, 0 );
407 :             break;
408 :         }
409 :         break;

```

右大曲げ時の PWM の設定です。eep\_buff[EEPROM\_CURVE\_PWM]変数から PWM 値を読み込みます。まず、変数 i に代入します。外輪が左タイヤ、内輪が右タイヤなので、speed 関数に代入する値は、左モータは「i」、右モータは「diff(i)」とします。

14.4.5 パターン 13 左大曲げ

```

411 :     case 13:
412 :         /* 左へ大曲げの終わりのチェック */
413 :         if( check_crossline() ) {          /* 大曲げ中もクロスラインチェック */
414 :             pattern = 21;
415 :             break;
416 :         }
417 :         if( check_rightline() ) {          /* 右ハーフラインチェック */
418 :             pattern = 51;
419 :             break;
420 :         }
421 :         if( check_leftline() ) {          /* 左ハーフラインチェック */
422 :             pattern = 61;
423 :             break;
424 :         }
425 :         switch( sensor_inp(MASK3_3) ) {
426 :         case 0x60:
427 :             pattern = 11;
428 :             break;
429 :         case 0xc0:
430 :             handle( -20 );
431 :             i = eep_buff[EEPROM_CURVE_PWM];
432 :             speed( diff(i), i );
433 :             break;
434 :         case 0x81:
435 :             handle( -25 );
436 :             i = eep_buff[EEPROM_CURVE_PWM];
437 :             speed( diff(i), i );
438 :             break;
439 :         case 0x83:
440 :         case 0x03:
441 :             handle( -30 );
442 :             i = eep_buff[EEPROM_CURVE_PWM];
443 :             speed( diff(i), i );
444 :             break;
445 :         case 0x06:
446 :             handle( -35 );
447 :             speed( 0, 0 );
448 :             break;
449 :         }
450 :         break;

```

















左大曲げ時の PWM の設定です。eep\_buff[EEPROM\_CURVE\_PWM]変数から PWM 値を読み込みます。まず、変数 i に代入します。外輪が右タイヤ、内輪が左タイヤなので、speed 関数に代入する値は、左モータは「diff(i)」、右モータは「i」とします。

## 15. プロジェクト「tr\_17」 クロスライン検出後の PWM 値の調整

### 15.1 内容

クロスライン検出後、徐行して進みます。その PWM 値を調整できるようにします。右レーンチェンジ、左レーンチェンジ後のスピードも同様に調整します。

### 15.2 プロジェクトの構成

 <b>tr_17</b>	
 Assembly source file	·tr_17start.src
 tr_17start.src	ベクタアドレスの設定、スタートアップルーチンが記述されています。
 C source file	·car_printf2.c
 beep.c	セクションの初期化、printf 文、scanf 文を実行します。
 car_printf2.c	·lcd2.c
 eeprom.c	LCD 制御を行います。
 lcd2.c	·switch.c
 switch.c	スイッチ制御を行います。
 tr_17.c	·beep.c
 Dependencies	ブザー制御を行います。
 beep.h	·eeprom.c
 eeprom.h	EEP-ROM 制御を行います。
 h8_3048.h	·tr_17.c
 lcd2.h	メインプログラムです。
 switch.h	

### 15.3 プログラム「tr\_17.c」

プログラムのゴシック体部分が追加した部分です。

前略

```

47 : /* EEP-ROM関連 */
48 : #define      EEP_ROM_SIZE      16      /* EEP-ROM使用サイズ      */
49 :
50 : #define      EEPROM_CHECK      0x00    /* EEP-ROMチェック      */
51 : #define      EEPROM_SERVO      0x01    /* サーボセンタ値      */
52 : #define      EEPROM_PWM        0x02    /* PWM値                  */
53 : #define      EEPROM_CURVE_PWM  0x03    /* 大カーブのPWM         */
54 : #define      EEPROM_CRANK_PWM  0x04    /* クランク部分のPWM     */

```

中略

```

115 : /*EEP-ROMのチェック */
116 : if( readEeprom( EEPROM_CHECK ) != 0x2006 ) {
117 : /*
118 : IDのチェック EEP-ROMを初めて使うかどうか
119 : 0x00番地にIDが書かれていなければ初めて使うと判断して初期化する
120 : */
121 : eep_buff[EEPROM_CHECK]      = 0x2006;
122 : eep_buff[EEPROM_SERVO]      = SERVO_CENTER;
123 : eep_buff[EEPROM_PWM]        = 50;
124 : eep_buff[EEPROM_CURVE_PWM]  = 50;
125 : eep_buff[EEPROM_CRANK_PWM]  = 40;
126 : } else {
127 : /* 2回目以降の使用の場合、データ読み込み */
128 : for( i=0; i<EEP_ROM_SIZE; i++ ) {
129 : eep_buff[ i ] = readEeprom( i );
130 : }
131 : }
132 :
133 : /* マイコンカーの状態初期化 */
134 : handle( 0 );
135 : speed( 0, 0 );

```



トレーニングボード 実習マニュアル(kit06 版)

```

136 :
137 :     while( 1 ) {
138 :     switch( pattern ) {
139 :
140 :     case 0:
141 :         /* スイッチ入力待ち */
142 :         if( pushsw_get() ) {
143 :             setBeepPattern( 0xc000 );
144 :             /* 保存 */
145 :             for( i=0; i<EEP_ROM_SIZE; i++ ) {
146 :                 writeEeprom( i, eep_buff[ i ] );
147 :                 while( !checkEeprom() ); /* 書き込み終了チェック */
148 :             }
149 :             pattern = 1;
150 :             cnt1 = 0;
151 :             break;
152 :         }
153 :
154 :         /* スイッチ4 設定値保存 */
155 :         if( getSwFlag(SW_4) ) {
156 :             setBeepPattern( 0x8000 );
157 :             /* 保存 */
158 :             for( i=0; i<EEP_ROM_SIZE; i++ ) {
159 :                 writeEeprom( i, eep_buff[ i ] );
160 :                 while( !checkEeprom() ); /* 書き込み終了チェック */
161 :             }
162 :             break;
163 :         }
164 :
165 :         /* スイッチ3 メニュー+ 1 */
166 :         if( getSwFlag(SW_3) ) {
167 :             lcd_pattern++;
168 :             if( lcd_pattern == 5 ) lcd_pattern = 1;
169 :         }
170 :
171 :         /* スイッチ2 メニュー- 1 */
172 :         if( getSwFlag(SW_2) ) {
173 :             lcd_pattern--;
174 :             if( lcd_pattern == 0 ) lcd_pattern = 4;
175 :         }
176 :
177 :         /* スイッチ、LCD処理 */
178 :         switch( lcd_pattern ) {
179 :         case 1:
180 :             /* サーボセンタ値調整 */
181 :             break;
182 :
183 :         case 2:
184 :             /* PWM調整 */
185 :             break;
186 :
187 :         case 3:
188 :             /* 大カーブPWM調整 */
189 :             break;
190 :
191 :         case 4:
192 :             /* クロスライン検出後のPWM調整 */
193 :             i = eep_buff[EEPROM_CRANK_PWM];
194 :             if( getSwFlag(SW_1) ) {
195 :                 i++;
196 :                 if( i > 100 ) i = 100;
197 :             }
198 :             if( getSwFlag(SW_0) ) {
199 :                 i--;
200 :                 if( i < 0 ) i = 0;
201 :             }
202 :             eep_buff[EEPROM_CRANK_PWM] = i;
203 :
204 :             /* LCD処理 */
205 :             lcdPosition( 0, 0 );
206 :             /* 0123456789abcd..f 1行16文字 */
207 :             lcdPrintf( "4 crankpwm = %03d", i );
208 :             /* 01234567..89abcde.f 1行16文字 */
209 :             lcdPrintf( "sensor=%02x bar=%d ",
210 :                 sensor_inp( 0xff ), startbar_get() );
211 :             break;
212 :         }
213 :
214 :         if( cnt1 < 100 ) { /* LED点滅処理 */
215 :             led_out( 0x1 );
216 :         } else if( cnt1 < 200 ) {
217 :             led_out( 0x2 );
218 :         } else {
219 :             cnt1 = 0;
220 :         }
221 :         break;
222 :     }
223 :
224 :     }
225 :
226 :     }
227 :
228 :     }
229 :
230 :     }
231 :
232 :     }
233 :
234 :     }
235 :
236 :     }
237 :
238 :     }
239 :
240 :     }
241 :
242 :     }
243 :
244 :     }
245 :
246 :     }
247 :
248 :     }
249 :
250 :     }
251 :
252 :     }
253 :
254 :     }
255 :
256 :     }
257 :
258 :     }
259 :
260 :     }
261 :
262 :     }
263 :
264 :     }
265 :
266 :     }
267 :
268 :     }
269 :
270 :     }
271 :
272 :     }
273 :
274 :     }
275 :
276 :     }
277 :
278 :     }
279 :
280 :     }
281 :
282 :     }
283 :
284 :     }
285 :
286 :     }
287 :
288 :     }
289 :
290 :     }
291 :
292 :     }
293 :
294 :     }
295 :
296 :     }
297 :
298 :     }
299 :
300 :     }
301 :
302 :     }
303 :
304 :     }
305 :
306 :     }
307 :
308 :     }
309 :
310 :     }
311 :
312 :     }
313 :
314 :     }
315 :
316 :     }
317 :
318 :     }
319 :
320 :     }
321 :
322 :     }
323 :
324 :     }
325 :
326 :     }
327 :
328 :     }
329 :
330 :     }
331 :
332 :     }
333 :
334 :     }
335 :
336 :     }
337 :
338 :     }
339 :
340 :     }
341 :
342 :     }
343 :
344 :     }
345 :
346 :     }
347 :
348 :     }
349 :
350 :     }
351 :
352 :     }
353 :
354 :     }
355 :
356 :     }
357 :
358 :     }
359 :
360 :     }
361 :
362 :     }
363 :
364 :     }
365 :
366 :     }
367 :
368 :     }
369 :
370 :     }
371 :
372 :     }
373 :
374 :     }
375 :
376 :     }
377 :
378 :     }
379 :
380 :     }
381 :
382 :     }
383 :
384 :     }
385 :
386 :     }
387 :
388 :     }
389 :
390 :     }
391 :
392 :     }
393 :
394 :     }
395 :
396 :     }
397 :
398 :     }
399 :
400 :     }
401 :
402 :     }
403 :
404 :     }
405 :
406 :     }
407 :
408 :     }
409 :
410 :     }
411 :
412 :     }
413 :
414 :     }
415 :
416 :     }
417 :
418 :     }
419 :
420 :     }
421 :
422 :     }
423 :
424 :     }
425 :
426 :     }
427 :
428 :     }
429 :
430 :     }
431 :
432 :     }
433 :
434 :     }
435 :
436 :     }
437 :
438 :     }
439 :
440 :     }
441 :
442 :     }
443 :
444 :     }
445 :
446 :     }
447 :
448 :     }
449 :
450 :     }
451 :
452 :     }
453 :
454 :     }
455 :
456 :     }
457 :
458 :     }
459 :
460 :     }
461 :
462 :     }
463 :
464 :     }
465 :
466 :     }
467 :
468 :     }
469 :
470 :     }
471 :
472 :     }
473 :
474 :     }
475 :
476 :     }
477 :
478 :     }
479 :
480 :     }
481 :
482 :     }
483 :
484 :     }
485 :
486 :     }
487 :
488 :     }
489 :
490 :     }
491 :
492 :     }
493 :
494 :     }
495 :
496 :     }
497 :
498 :     }
499 :
500 :     }
501 :
502 :     }
503 :
504 :     }
505 :
506 :     }
507 :
508 :     }
509 :
510 :     }
511 :
512 :     }
513 :
514 :     }
515 :
516 :     }
517 :
518 :     }
519 :
520 :     }
521 :
522 :     }
523 :
524 :     }
525 :
526 :     }
527 :
528 :     }
529 :
530 :     }
531 :
532 :     }
533 :
534 :     }
535 :
536 :     }
537 :
538 :     }
539 :
540 :     }
541 :
542 :     }
543 :
544 :     }
545 :
546 :     }
547 :
548 :     }
549 :
550 :     }
551 :
552 :     }
553 :
554 :     }
555 :
556 :     }
557 :
558 :     }
559 :
560 :     }
561 :
562 :     }
563 :
564 :     }
565 :
566 :     }
567 :
568 :     }
569 :
570 :     }
571 :
572 :     }
573 :
574 :     }
575 :
576 :     }
577 :
578 :     }
579 :
580 :     }
581 :
582 :     }
583 :
584 :     }
585 :
586 :     }
587 :
588 :     }
589 :
590 :     }
591 :
592 :     }
593 :
594 :     }
595 :
596 :     }
597 :
598 :     }
599 :
600 :     }
601 :
602 :     }
603 :
604 :     }
605 :
606 :     }
607 :
608 :     }
609 :
610 :     }
611 :
612 :     }
613 :
614 :     }
615 :
616 :     }
617 :
618 :     }
619 :
620 :     }
621 :
622 :     }
623 :
624 :     }
625 :
626 :     }
627 :
628 :     }
629 :
630 :     }
631 :
632 :     }
633 :
634 :     }
635 :
636 :     }
637 :
638 :     }
639 :
640 :     }
641 :
642 :     }
643 :
644 :     }
645 :
646 :     }
647 :
648 :     }
649 :
650 :     }
651 :
652 :     }
653 :
654 :     }
655 :
656 :     }
657 :
658 :     }
659 :
660 :     }
661 :
662 :     }
663 :
664 :     }
665 :
666 :     }
667 :
668 :     }
669 :
670 :     }
671 :
672 :     }
673 :
674 :     }
675 :
676 :     }
677 :
678 :     }
679 :
680 :     }
681 :
682 :     }
683 :
684 :     }
685 :
686 :     }
687 :
688 :     }
689 :
690 :     }
691 :
692 :     }
693 :
694 :     }
695 :
696 :     }
697 :
698 :     }
699 :
700 :     }
701 :
702 :     }
703 :
704 :     }
705 :
706 :     }
707 :
708 :     }
709 :
710 :     }
711 :
712 :     }
713 :
714 :     }
715 :
716 :     }
717 :
718 :     }
719 :
720 :     }
721 :
722 :     }
723 :
724 :     }
725 :
726 :     }
727 :
728 :     }
729 :
730 :     }
731 :
732 :     }
733 :
734 :     }
735 :
736 :     }
737 :
738 :     }
739 :
740 :     }
741 :
742 :     }
743 :
744 :     }
745 :
746 :     }
747 :
748 :     }
749 :
750 :     }
751 :
752 :     }
753 :
754 :     }
755 :
756 :     }
757 :
758 :     }
759 :
760 :     }
761 :
762 :     }
763 :
764 :     }
765 :
766 :     }
767 :
768 :     }
769 :
770 :     }
771 :
772 :     }
773 :
774 :     }
775 :
776 :     }
777 :
778 :     }
779 :
780 :     }
781 :
782 :     }
783 :
784 :     }
785 :
786 :     }
787 :
788 :     }
789 :
790 :     }
791 :
792 :     }
793 :
794 :     }
795 :
796 :     }
797 :
798 :     }
799 :
800 :     }
801 :
802 :     }
803 :
804 :     }
805 :
806 :     }
807 :
808 :     }
809 :
810 :     }
811 :
812 :     }
813 :
814 :     }
815 :
816 :     }
817 :
818 :     }
819 :
820 :     }
821 :
822 :     }
823 :
824 :     }
825 :
826 :     }
827 :
828 :     }
829 :
830 :     }
831 :
832 :     }
833 :
834 :     }
835 :
836 :     }
837 :
838 :     }
839 :
840 :     }
841 :
842 :     }
843 :
844 :     }
845 :
846 :     }
847 :
848 :     }
849 :
850 :     }
851 :
852 :     }
853 :
854 :     }
855 :
856 :     }
857 :
858 :     }
859 :
860 :     }
861 :
862 :     }
863 :
864 :     }
865 :
866 :     }
867 :
868 :     }
869 :
870 :     }
871 :
872 :     }
873 :
874 :     }
875 :
876 :     }
877 :
878 :     }
879 :
880 :     }
881 :
882 :     }
883 :
884 :     }
885 :
886 :     }
887 :
888 :     }
889 :
890 :     }
891 :
892 :     }
893 :
894 :     }
895 :
896 :     }
897 :
898 :     }
899 :
900 :     }
901 :
902 :     }
903 :
904 :     }
905 :
906 :     }
907 :
908 :     }
909 :
910 :     }
911 :
912 :     }
913 :
914 :     }
915 :
916 :     }
917 :
918 :     }
919 :
920 :     }
921 :
922 :     }
923 :
924 :     }
925 :
926 :     }
927 :
928 :     }
929 :
930 :     }
931 :
932 :     }
933 :
934 :     }
935 :
936 :     }
937 :
938 :     }
939 :
940 :     }
941 :
942 :     }
943 :
944 :     }
945 :
946 :     }
947 :
948 :     }
949 :
950 :     }
951 :
952 :     }
953 :
954 :     }
955 :
956 :     }
957 :
958 :     }
959 :
960 :     }
961 :
962 :     }
963 :
964 :     }
965 :
966 :     }
967 :
968 :     }
969 :
970 :     }
971 :
972 :     }
973 :
974 :     }
975 :
976 :     }
977 :
978 :     }
979 :
980 :     }
981 :
982 :     }
983 :
984 :     }
985 :
986 :     }
987 :
988 :     }
989 :
990 :     }
991 :
992 :     }
993 :
994 :     }
995 :
996 :     }
997 :
998 :     }
999 :
1000 :    }

```

中略

トレーニングボード 実習マニュアル(kit06 版)

```

493 :     case 23:
494 :         /* クロスライン後のトレース、クランク検出 */
495 :         i = eep_buff[EEPROM_CRANK_PWM];
496 :         if( sensor_inp(MASK4_4)==0xf8 ) {
497 :             /* 左クランクと判断 左クランククリア処理へ */
498 :             led_out( 0x1 );
499 :             handle( -38 );
500 :             speed( 10 ,50 );
501 :             pattern = 31;
502 :             cnt1 = 0;
503 :             break;
504 :         }
505 :         if( sensor_inp(MASK4_4)==0x1f ) {
506 :             /* 右クランクと判断 右クランククリア処理へ */
507 :             led_out( 0x2 );
508 :             handle( 38 );
509 :             speed( 50 ,10 );
510 :             pattern = 41;
511 :             cnt1 = 0;
512 :             break;
513 :         }
514 :         switch( sensor_inp(MASK3_3) ) {
515 :             case 0x00:
516 :                 /* センタ まっすぐ */
517 :                 handle( 0 );
518 :                 speed( i ,i );
519 :                 break;
520 :             case 0x04:
521 :             case 0x06:
522 :             case 0x07:
523 :             case 0x03:
524 :                 /* 左寄り 右曲げ */
525 :                 handle( 8 );
526 :                 speed( i ,diff(i) );
527 :                 break;
528 :             case 0x20:
529 :             case 0x60:
530 :             case 0xe0:
531 :             case 0xc0:
532 :                 /* 右寄り 左曲げ */
533 :                 handle( -8 );
534 :                 speed( diff(i) ,i );
535 :                 break;
536 :         }
537 :         break;

```

中略

```

590 :     case 53:
591 :         /* 右ハーフライン後のトレース、レーンチェンジ */
592 :         if( sensor_inp(MASK4_4) == 0x00 ) {
593 :             handle( 15 );
594 :             speed( 40 ,diff(40) );
595 :             pattern = 54;
596 :             cnt1 = 0;
597 :             break;
598 :         }
599 :         i = eep_buff[EEPROM_CRANK_PWM];
600 :         switch( sensor_inp(MASK3_3) ) {
601 :             case 0x00:
602 :                 /* センタ まっすぐ */
603 :                 handle( 0 );
604 :                 speed( i ,i );
605 :                 break;
606 :             case 0x04:
607 :             case 0x06:
608 :             case 0x07:
609 :             case 0x03:
610 :                 /* 左寄り 右曲げ */
611 :                 handle( 8 );
612 :                 speed( i ,diff(i) );
613 :                 break;
614 :             case 0x20:
615 :             case 0x60:
616 :             case 0xe0:
617 :             case 0xc0:
618 :                 /* 右寄り 左曲げ */
619 :                 handle( -8 );
620 :                 speed( diff(i) ,i );
621 :                 break;
622 :             default:
623 :                 break;
624 :         }
625 :         break;
626 :

```

中略

```

653 :     case 63:
654 :         /* 左ハーフライン後のトレース、レーンチェンジ */
655 :         if( sensor_inp(MASK4_4) == 0x00 ) {

```

```

656 :         handle( -15 );
657 :         speed( diff(40) ,40 );
658 :         pattern = 64;
659 :         cnt1 = 0;
660 :         break;
661 :     }
662 :     i = eep_buff[EEPROM_CRANK_PWM];
663 :     switch( sensor_inp(MASK3_3) ) {
664 :     case 0x00:
665 :         /* センタ まっすぐ */
666 :         handle( 0 );
667 :         speed( i ,i );
668 :         break;
669 :     case 0x04:
670 :     case 0x06:
671 :     case 0x07:
672 :     case 0x03:
673 :         /* 左寄り 右曲げ */
674 :         handle( 8 );
675 :         speed( i ,diff(i) );
676 :         break;
677 :     case 0x20:
678 :     case 0x60:
679 :     case 0xe0:
680 :     case 0xc0:
681 :         /* 右寄り 左曲げ */
682 :         handle( -8 );
683 :         speed( diff(i) ,i );
684 :         break;
685 :     default:
686 :         break;
687 :     }
688 :     break;

```

以下、略

## 15.4 プログラムの解説

### 15.4.1 EEP-ROM エリアの追加

47 :	/* EEP-ROM 関連 */		
48 :	#define	EEP_ROM_SIZE	16 /* EEP-ROM 使用サイズ */
49 :			
50 :	#define	EEPROM_CHECK	0x00 /* EEP-ROM チェック */
51 :	#define	EEPROM_SERVO	0x01 /* サーボセンタ値 */
52 :	#define	EEPROM_PWM	0x02 /* PWM 値 */
53 :	#define	EEPROM_CURVE_PWM	0x03 /* 大カーブの PWM */
54 :	#define	EEPROM_CRANK_PWM	0x04 /* クランク部分の PWM */

クロスライン検出後の PWM を設定する EEP-ROM の番地を決めます。0x03 まで使っていたので、0x04 番地を EEPROM\_CRANK\_PWM として定義します。

番地	番地名	内容	初期値
0x00	EEPROM_CHECK	EEP-ROM チェック用 0x2006 で無ければ、初期値を読み込む	0x2006
0x01	EEPROM_SERVO	サーボセンタ値	5000
0x02	EEPROM_PWM	PWM 値	50
0x03	EEPROM_CURVE_PWM	大カーブでの PWM 値	50
<b>0x04</b>	<b>EEPROM_CRANK_PWM</b>	<b>クロスライン検出後の PWM 値</b>	<b>40</b>
0x05 ~ 0x0f	未定義		

### 15.4.2 EEPROM から読み込み

```

115 :      /*EEP-ROM のチェック */
116 :      if( readEeprom( EEPROM_CHECK ) != 0x2006 ) {
117 :          /*
118 :           ID のチェック EEPROM を初めて使うかどうか
119 :           0x00 番地に ID が書かれていなければ初めて使うと判断して初期化する
120 :          */
121 :          eep_buff[EEPROM_CHECK]      = 0x2006;
122 :          eep_buff[EEPROM_SERVO]      = SERVO_CENTER;
123 :          eep_buff[EEPROM_PWM]        = 50;
124 :          eep_buff[EEPROM_CURVE_PWM]  = 50;
125 :          eep_buff[EEPROM_CRANK_PWM] = 40;
126 :      } else {
127 :          /* 2 回目以降の使用の場合、データ読み込み */
128 :          for( i=0; i<EEP_ROM_SIZE; i++ ) {
129 :              eep_buff[ i ] = readEeprom( i );
130 :          }
131 :      }

```

125 行に eep\_buff[EEPROM\_CRANK\_PWM]変数の初期化を追加しています。

### 15.4.3 スイッチ入力待ち

```

188 :          /* スイッチ 3 メニュー + 1 */
189 :          if( getSwFlag(SW_3) ) {
190 :              lcd_pattern++;
191 :              if( lcd_pattern == 5 ) lcd_pattern = 1;
192 :          }
193 :          /* スイッチ 2 メニュー - 1 */
194 :          if( getSwFlag(SW_2) ) {
195 :              lcd_pattern--;
196 :              if( lcd_pattern == 0 ) lcd_pattern = 4;
197 :          }

```

SW\_3 で LCD に表示する内容を次へ進めます。SW\_2 で戻します。lcd\_pattern=4 をクロスライン後の PWM 値の調整として追加します。

191 行で上限のチェックです。lcd\_pattern の 5 はありませんので 1 にします。

196 行で下限のチェックです。lcd\_pattern の 0 はありませんので 4 にします。

lcd_pattern	内容
1	サーボセンタ値の調整
2	PWM 値の調整
3	大カーブの PWM 値の調整
4	<b>クロスライン後の PWM 値の調整</b>

```

199 :      /* スイッチ、LCD処理 */
200 :      switch( lcd_pattern ) {
中略
268 :          case 4:
269 :              /* クロスライン検出後のPWM調整 */
270 :              i = eep_buff[EEPROM_CRANK_PWM];
271 :              if( getSwFlag(SW_1) ) {
272 :                  i++;
273 :                  if( i > 100 ) i = 100;
274 :              }
275 :              if( getSwFlag(SW_0) ) {
276 :                  i--;
277 :                  if( i < 0 ) i = 0;
278 :              }
279 :              eep_buff[EEPROM_CRANK_PWM] = i;
280 :
281 :              /* LCD処理 */
282 :              lcdPosition( 0, 0 );
283 :              /* 0123456789abcd..f 1行16文字 */
284 :              lcdPrintf( "4 crankpwm = %03d", i );
285 :              /* 01234567..89abcde.f 1行16文字 */
286 :              lcdPrintf( "sensor=%02x bar=%d ",
287 :                          sensor_inp( 0xff ), startbar_get() );
288 :              break;
289 :          }

```

200行で、lcd\_patternの番号によりプログラムをジャンプします。4なら268行へ飛びます。  
 270行で、eep\_buff[EEPROM\_CRANK\_PWM]変数の値をいったんi変数に代入します。  
 271行で、SW\_1が押されているかチェックします。押されていれば、i変数を1つ増加させます。  
 275行で、SW\_0が押されているかチェックします。押されていれば、i変数を1つ減少させます。  
 279行で、i変数の値をeep\_buff[EEPROM\_CRANK\_PWM]変数へ代入します。SW\_1、SW\_2で操作された値が代入されます。  
 282行からLCDにクロスライン後のPWM値を表示します。2行目が余っているのでセンサの状態を表示させています。

15.4.4 パターン 23 クロスライン後のトレース、クランク検出

```

493 :     case 23:
494 :         /* クロスライン後のトレース、クランク検出 */
495 :         i = eep_buff[EEPROM_CRANK_PWM];
496 :         if( sensor_inp(MASK4_4)==0xf8 ) {
497 :             /* 左クランクと判断 左クランククリア処理へ */
498 :             led_out( 0x1 );
499 :             handle( -38 );
500 :             speed( 10 ,50 );
501 :             pattern = 31;
502 :             cnt1 = 0;
503 :             break;
504 :         }
505 :         if( sensor_inp(MASK4_4)==0x1f ) {
506 :             /* 右クランクと判断 右クランククリア処理へ */
507 :             led_out( 0x2 );
508 :             handle( 38 );
509 :             speed( 50 ,10 );
510 :             pattern = 41;
511 :             cnt1 = 0;
512 :             break;
513 :         }
514 :         switch( sensor_inp(MASK3_3) ) {
515 :             case 0x00:
516 :                 /* センタ まっすぐ */
517 :                 handle( 0 );
518 :                 speed( i ,i );
519 :                 break;
520 :             case 0x04:
521 :             case 0x06:
522 :             case 0x07:
523 :             case 0x03:
524 :                 /* 左寄り 右曲げ */
525 :                 handle( 8 );
526 :                 speed( i ,diff(i) );
527 :                 break;
528 :             case 0x20:
529 :             case 0x60:
530 :             case 0xe0:
531 :             case 0xc0:
532 :                 /* 右寄り 左曲げ */
533 :                 handle( -8 );
534 :                 speed( diff(i) ,i );
535 :                 break;
536 :         }
537 :         break;

```

クロスライン後の PWM 値の設定です。eep\_buff[EEPROM\_CRANK\_PWM]変数から PWM 値を読み込みます。変数 i に代入します。speed 関数に代入する PWM 値は、変数 i の値となります。

今までは speed 関数に 40 を入れていました。今回は eep\_buff[EEPROM\_CRANK\_PWM]を入れて、クロスライン後のスピードを調整できるようにしています。

15.4.5 パターン 53 右ハーフライン後のトレース、レーンチェンジ

```

590 :     case 53:
591 :         /* 右ハーフライン後のトレース、レーンチェンジ */
592 :         if( sensor_inp(MASK4_4) == 0x00 ) {
593 :             handle( 15 );
594 :             speed( 40 ,diff(40) );
595 :             pattern = 54;
596 :             cnt1 = 0;
597 :             break;
598 :         }
599 :         i = eep_buff[EEPROM_CRANK_PWM];
600 :         switch( sensor_inp(MASK3_3) ) {
601 :             case 0x00:
602 :                 /* センタ まっすぐ */
603 :                 handle( 0 );
604 :                 speed( i ,i );
605 :                 break;
606 :             case 0x04:
607 :             case 0x06:
608 :             case 0x07:
609 :             case 0x03:
610 :                 /* 左寄り 右曲げ */
611 :                 handle( 8 );
612 :                 speed( i ,diff(i) );
613 :                 break;
614 :             case 0x20:
615 :             case 0x60:
616 :             case 0xe0:
617 :             case 0xc0:
618 :                 /* 右寄り 左曲げ */
619 :                 handle( -8 );
620 :                 speed( diff(i) ,i );
621 :                 break;
622 :             default:
623 :                 break;
624 :         }
625 :         break;

```

右ハーフライン後のPWM値の設定です。eep\_buff[EEPROM\_CRANK\_PWM]変数からPWM値を読み込みます。変数iに代入します。speed関数に代入するPWM値は、変数iの値となります。

今まではspeed関数に40を入れていました。今回はeep\_buff[EEPROM\_CRANK\_PWM]を入れて、右ハーフライン後のスピードを調整できるようにしています。

15.4.6 パターン 63 左ハーフライン後のトレース、レーンチェンジ

```

653 :     case 63:
654 :         /* 左ハーフライン後のトレース、レーンチェンジ */
655 :         if( sensor_inp(MASK4_4) == 0x00 ) {
656 :             handle( -15 );
657 :             speed( diff(40) ,40 );
658 :             pattern = 64;
659 :             cnt1 = 0;
660 :             break;
661 :         }
662 :         i = eep_buff[EEPROM_CRANK_PWM];
663 :         switch( sensor_inp(MASK3_3) ) {
664 :             case 0x00:
665 :                 /* センタ まっすぐ */
666 :                 handle( 0 );
667 :                 speed( i ,i );
668 :                 break;
669 :             case 0x04:
670 :             case 0x06:
671 :             case 0x07:
672 :             case 0x03:
673 :                 /* 左寄り 右曲げ */
674 :                 handle( 8 );
675 :                 speed( i ,diff(i) );
676 :                 break;
677 :             case 0x20:
678 :             case 0x60:
679 :             case 0xe0:
680 :             case 0xc0:
681 :                 /* 右寄り 左曲げ */
682 :                 handle( -8 );
683 :                 speed( diff(i) ,i );
684 :                 break;
685 :             default:
686 :                 break;
687 :         }
688 :         break;

```

左ハーフライン後のPWM値の設定です。eep\_buff[EEPROM\_CRANK\_PWM]変数からPWM値を読み込みます。変数iに代入します。speed関数に代入するPWM値は、変数iの値となります。

今まではspeed関数に40を入れていました。今回はeep\_buff[EEPROM\_CRANK\_PWM]を入れて、左ハーフライン後のスピードを調整できるようにしています。



















## 16. プロジェクト「tr\_21」 クロスライン検出後、10cm 直進させる(エンコーダ使用)

### 16.1 内容

今までのマイコンカーは、現在の走行スピードが分かりません。実際の車で例えると、一般道路を速度メータを見ないで走っているようなものです。普通の運転では、速度メータを見ながら走ります。この速度メータに当たる部分が、**ロータリエンコーダ**という装置です。ロータリエンコーダをマイコンカーに取り付けると、現在のスピードと走行距離が分かります。速ければPWM値を落とす、遅ければPWM値を上げる、とプログラムすれば現在のスピードに合わせてマイコンカーを走らせることができます。ロータリエンコーダについての詳細は、「ロータリエンコーダ実習マニュアル」をご覧ください。

今までのプログラムはクロスラインを検出した直後の 50ms 間は、2 本目を読み飛ばす為にセンサを見ません。たまたまマイコンカーの速度が遅く、50ms 後でもまだクロスラインがあれば誤動作するでしょう。ロータリエンコーダは速度と距離を測ることができます。今回は、クロスライン検出後、10cm は何もしない、10cm 後からトレース処理(パターン23)を実行するようにします。距離にすれば、マイコンカーのスピードが遅かるうが速かるうが 10cm は 10cm ですので、常に同じ位置(クロスラインの 10cm 先)からパターン 23 が実行されるようになります。

### 16.2 プロジェクトの構成

 <b>tr_21</b>	
 Assembly source file	· tr_21start.src
 tr_21start.src	ベクタアドレスの設定、スタートアップルーチンが記述されています。
 C source file	· car_printf2.c
 beep.c	セクションの初期化、printf 文、scanf 文を実行します。
 car_printf2.c	· lcd2.c
 eeprom.c	LCD 制御を行います。
 lcd2.c	· switch.c
 switch.c	スイッチ制御を行います。
 tr_21.c	· beep.c
 Dependencies	· beep.c
 beep.h	ブザー制御を行います。
 eeprom.h	· eeprom.c
 h8_3048.h	EEP-ROM 制御を行います。
 lcd2.h	· tr_21.c
 switch.h	メインプログラムです。この中にロータリエンコーダに関わるプログラムを追加します。

### 16.3 プログラム「tr\_21.c」

プログラムのゴシック体部分が追加した部分です。

前略

```

99 :  /* エンコーダ関連 */
100 :  int      iTimer10;          /* エンコーダ取得間隔 */
101 :  long     iEncoderTotal;     /* 積算値 */
102 :  int      iEncoderMax;      /* 現在最大値 */
103 :  int      iEncoder;         /* 現在値 */
104 :  unsigned int uEncoderBuff; /* 前回値保存 */
105 :  long     iEncoderLine;     /* ライン検出時の積算値 */

```

中略

```

308 :      case 1:
309 :          /* スタートバーが開いたかチェック */
310 :          if( !startbar_get() ){
311 :              /* スタート!! */

```

トレーニングボード 実習マニュアル(kit06 版)

```

312 :         iTimer10      = 0;
313 :         IEncoderTotal  = 0;
314 :         iEncoderMax    = 0;
315 :         iEncoder       = 0;
316 :         led_out( 0x0 );
317 :         pattern = 11;
318 :         cnt1 = 0;
319 :         break;
320 :     }
321 :     if( cnt1 < 50 ) {          /* LED点滅処理          */
322 :         led_out( 0x1 );
323 :     } else if( cnt1 < 100 ) {
324 :         led_out( 0x2 );
325 :     } else {
326 :         cnt1 = 0;
327 :     }
328 :     break;

```

中略

```

488 :     case 21:
489 :         /* 1本目のクロスライン検出時の処理 */
490 :         IEncoderLine = IEncoderTotal;
491 :         led_out( 0x3 );
492 :         handle( 0 );
493 :         speed( 0 ,0 );
494 :         pattern = 22;
495 :         cnt1 = 0;
496 :         break;
497 :
498 :     case 22:
499 :         /* 2本目を読み飛ばす */
500 :         if( IEncoderTotal-IEncoderLine >= 97 ) { /* 約10cm */
501 :             pattern = 23;
502 :             cnt1 = 0;
503 :         }
504 :         break;

```

中略

```

586 :     case 51:
587 :         /* 1本目の右ハーフライン検出時の処理 */
588 :         IEncoderLine = IEncoderTotal;
589 :         led_out( 0x2 );
590 :         handle( 0 );
591 :         speed( 0 ,0 );
592 :         pattern = 52;
593 :         cnt1 = 0;
594 :         break;
595 :
596 :     case 52:
597 :         /* 2本目を読み飛ばす */
598 :         if( IEncoderTotal-IEncoderLine >= 97 ) { /* 約10cm */
599 :             pattern = 53;
600 :             cnt1 = 0;
601 :         }
602 :         break;

```

中略

```

650 :     case 61:
651 :         /* 1本目の左ハーフライン検出時の処理 */
652 :         IEncoderLine = IEncoderTotal;
653 :         led_out( 0x1 );
654 :         handle( 0 );
655 :         speed( 0 ,0 );
656 :         pattern = 62;
657 :         cnt1 = 0;
658 :         break;
659 :
660 :     case 62:
661 :         /* 2本目を読み飛ばす */
662 :         if( IEncoderTotal-IEncoderLine >= 97 ) { /* 約10cm */
663 :             pattern = 63;
664 :             cnt1 = 0;
665 :         }
666 :         break;

```

中略

```

722 : /******
723 : /* H8/3048F-ONE 内蔵周辺機能 初期化          */
724 : /******
725 : void init( void )
726 : {
727 :     /* I/Oポートの入出力設定 */
728 :     P1DDR = 0xff;
729 :     P2DDR = 0xff;
730 :     P3DDR = 0x8e;          /* スイッチ、EEP-ROM      */
731 :     P4DDR = 0xff;          /* LCD接続                */
732 :     P5DDR = 0xff;

```

## トレーニングボード 実習マニュアル(kit06 版)

```

733 : P6DDR = 0xf0; /* CPU基板上的DIP SW */
734 : P8DDR = 0xff;
735 : P9DDR = 0xe3; /* bit4,2:sw bit3:232c */
736 : PADDR = 0xf6; /* 3:barセンサ 0:エンコーダ */
737 : PBDR = 0xc0;
738 : PBDDR = 0xfe; /* モータドライブ基板Vol.3 */
739 : /* センサ基板のP7は、入力専用なので入出力設定はありません */
740 :
741 : /* ITU0 1ms毎の割り込み */
742 : ITU0_TCR = 0x23;
743 : ITU0_GRA = TIMER_CYCLE;
744 : ITU0_IER = 0x01;
745 :
746 : /* ITU2 パルス入力の設定 */
747 : ITU2_TCR = 0x04; /* PA0端子のパルスでカウント*/
748 :
749 : /* ITU3,4 リセット同期PWMモード 左右モータ、サーボ用 */
750 : ITU3_TCR = 0x23;
751 : ITU_FCR = 0x3e;
752 : ITU3_GRA = PWM_CYCLE; /* 周期の設定 */
753 : ITU3_GRB = ITU3_BRB = 0; /* 左モータのPWM設定 */
754 : ITU4_GRA = ITU4_BRA = 0; /* 右モータのPWM設定 */
755 : ITU4_GRB = ITU4_BRB = SERVO_CENTER; /* サーボのPWM設定 */
756 : ITU_TOER = 0x38;
757 :
758 : /* ITUのカウンタスタート */
759 : ITU_STR = 0x0d;
760 : }
761 :
762 : /*****
763 : /* ITU0 割り込み処理 */
764 : *****/
765 : #pragma interrupt( interrupt_timer0 )
766 : void interrupt_timer0( void )
767 : {
768 :     unsigned int i;
769 :
770 :     ITU0_TSR &= 0xfe; /* フラグクリア */
771 :     cnt0++;
772 :     cnt1++;
773 :
774 :     /* LCD表示処理用関数です。1ms毎に実行します。 */
775 :     lcdShowProcess();
776 :     /* 拡張スイッチ用関数です。1ms毎に実行します。 */
777 :     switchProcess();
778 :     /* ブザー処理用関数です。1ms毎に実行します。 */
779 :     beepProcess();
780 :
781 :     /* エンコーダ関連 */
782 :     iTimer10++;
783 :     if( iTimer10 >= 10 ) {
784 :         iTimer10 = 0;
785 :         i = ITU2_CNT;
786 :         iEncoder = i - uEncoderBuff;
787 :         iEncoderTotal += iEncoder;
788 :         if( iEncoder > iEncoderMax )
789 :             iEncoderMax = iEncoder;
790 :         uEncoderBuff = i;
791 :     }
792 : }

```

以下、略

## 16.4 エンコーダの回転数とパルス数の関係

エンコーダについての詳しい説明は、「ロータリエンコーダ実習マニュアル」を参照してください。ここでは、エンコーダの回転数とパルス数の関係を計算しておきます。これが分からないとプログラムできません。

下記条件のエンコーダ、タイヤとします。

項目	内容
エンコーダの1回転のパルス数	100 パルス / 回転
タイヤ直径(実寸)	33mm

タイヤの直径が分かるので、円周が分かります。

$$\text{円周} = 2 \times r = 33 \times 3.14 = 103.62\text{mm}$$

エンコーダは 100 パルス / 回転なので、

$$103.62\text{mm} \text{ 進むと } 100 \text{ パルス}$$

ということになります。

1m 進んだときのパルス数は、

$$100 \text{ パルス} : 103.62\text{mm} = x \text{ パルス} : 1000\text{mm}$$

$$x = 965 \text{ パルス}$$

となります。1m/s で進んだとき、1 秒間のパルス数は、965 パルスということになります。

1m/s で進んだとき、10ms 間のパルス数は、

$$1 \text{ 秒} : 965 \text{ パルス} = 0.01 \text{ 秒} : x \text{ パルス}$$

$$x = 9.65 \text{ パルス}$$

となります。まとめると下表のようになります。

項目	内容
1m 進んだときのパルス数	965 パルス
1m/s で進んだときの 10ms 間に カウントするパルス数	9.65 パルス

例えば、速度が 1m/s 以下かどうか調べたいとき、10ms に検出したパルス数が 10 以下かどうか調べればいいこととなります。2m/s なら 19 以下かどうか、3m/s なら 29 以下かどうか...というように比例します。小数点は使えないのでパルス値は四捨五入します。

10ms 間ごとに最新のエンコーダのパルス値を更新する作業は、プログラムで行います。時間が短ければ短いほど最新のスピードが分かりますが、パルス数が少なくなるため精度が悪くなります。時間が長ければ精度が良くなりますが、更新の間隔が長くなり最新の速度が分かりません。10ms ごとにカウントするのが、経験上良いかと思えます。

## 16.5 プログラムの解説

### 16.5.1 エンコーダ関連の変数の宣言



ロータリエンコーダを使用するに当たって、新たに変数を宣言しています。

変数名	意味	内容
iTimer10	10ms タイマ	エンコーダ値の更新は、interrupt_timer0 関数内で行います。interrupt_timer0 関数は 1ms ごとに実行されますが、エンコーダ処理は 10ms ごとです。そこで、この変数を 1ms ごとに +1 して 10 になったかどうかチェックしています。
lEncoderTotal	エンコーダ積算値	スタートしてからのエンコーダパルスの積算値を保存しています。long 型変数ですので、21 億回までカウントできます。
iEncoderMax	10ms ごとの最大値	10ms ごとに更新されるエンコーダ値の最大値を保存しています。走行後、この値をチェックすれば最速値が分かります。
iEncoder	10ms ごとの現在値	10ms ごとに更新されるエンコーダ値の現在値を保存しています。この値をチェックすれば、現在のスピードが分かります。
uEncoderBuff	前回値保存用バッファ	ITU2_CNT の前回の値を保存しています。main 関数では使用しません。
lEncoderLine	横線検出時の積算値保存用	横線検出時の積算値を保存するための変数です。

### 16.5.2 パターン1 スタート前の初期化



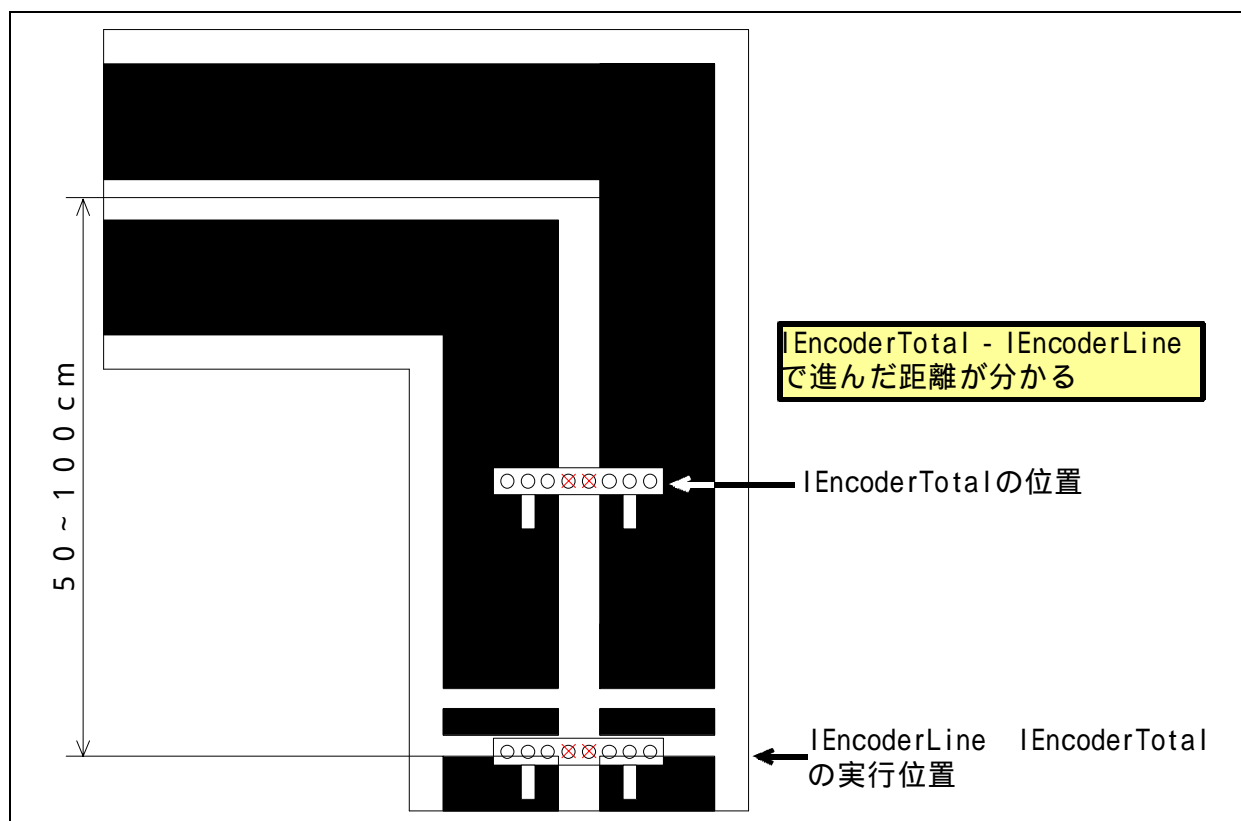
スタート直前に、もう一度 312~315 行でロータリエンコーダに関する変数を初期化します。待機中に回ってしまったカウント値をクリアします。

16.5.3 パターン 21 クロスライン検出時の積算値を取得

```

488 :     case 21:
489 :         /* 1本目のクロスライン検出時の処理 */
490 :         IEncoderLine = IEncoderTotal;
491 :         led_out( 0x3 );
492 :         handle( 0 );
493 :         speed( 0 ,0 );
494 :         pattern = 22;
495 :         cnt1 = 0;
496 :         break;
    
```

クロスラインを検出した瞬間の積算値 IEncoderTotal の値を、IEncoderLine にコピーしています。「IEncoderTotal - IEncoderLine」で、クロスラインを検出してからのパルス数が分かります。要は、クロスラインから進んだ距離が分かります。



16.5.4 パターン 22 2本目を読み飛ばす

```

498 :     case 22:
499 :         /* 2本目を読み飛ばす */
500 :         if( |EncoderTotal-|EncoderLine >= 97 ) { /* 約10cm */
501 :             pattern = 23;
502 :             cnt1 = 0;
503 :         }
504 :         break;

```

500 行で、10cm 進んだかチェックしています。距離は、1 本目の白線 2cm + 黒部分 3cm + 2 本目の白線 2cm で、合計 7cm です。余裕を見て 10cm としています。次のような意味です。

$$|EncoderTotal - |EncoderLine \geq 10cm$$

現在の積算値 - クロスラインを検出したときの積算値  $\geq 10cm$

クランク内で進んだパルス数(距離)  $\geq 10cm$

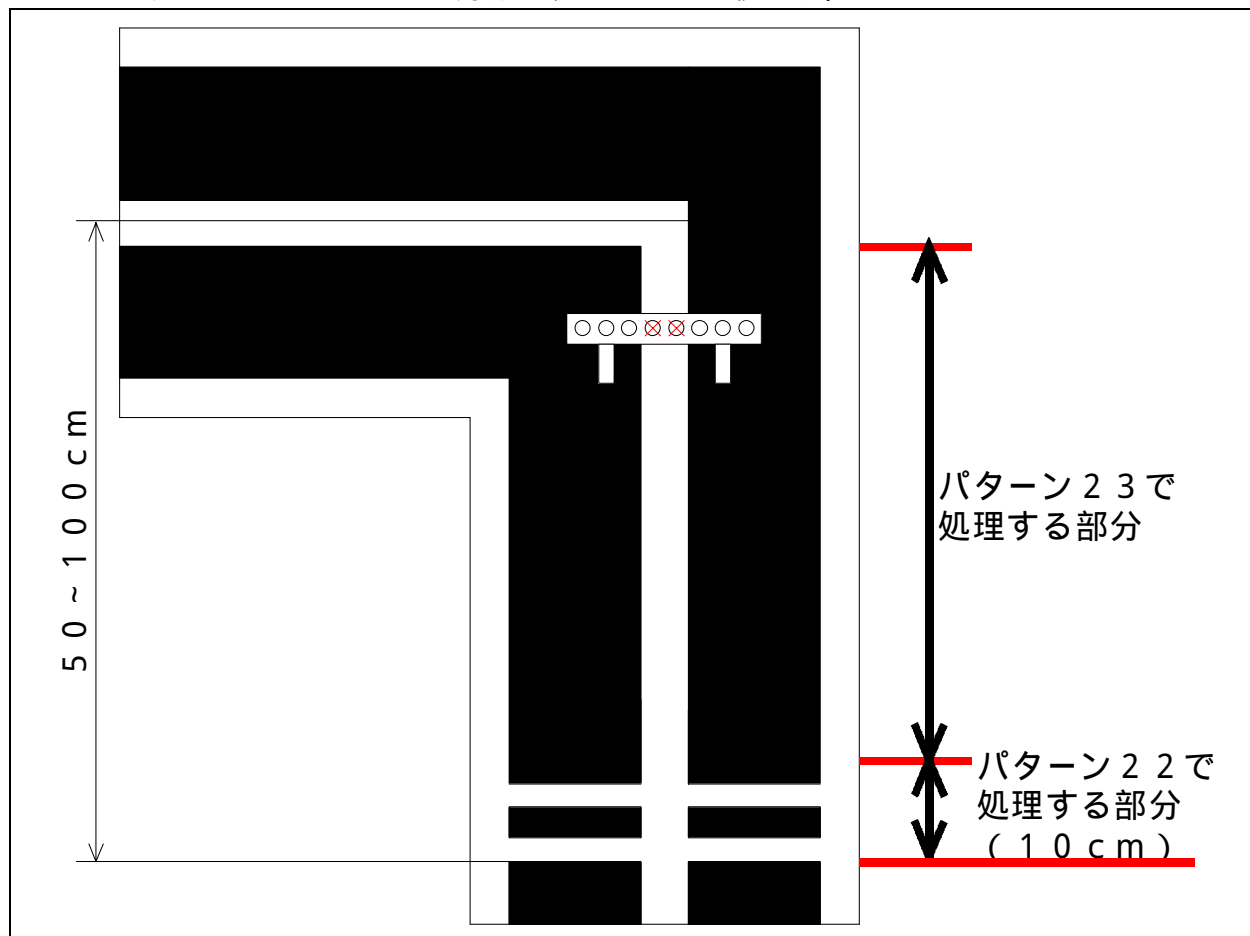
今回のエンコーダは 1m で 965 パルスのエンコーダなので、10cm 進んだかどうかチェックするには、

$$1m : 965 \text{ パルス} = 0.1m : x \text{ パルス}$$

$$x = 96.5 \text{ パルス}$$

と、クロスラインを検出した瞬間から 97 パルス以上(四捨五入)になったかプログラムで見れば良いことになります。

97 パルス以上になると 10cm 進んだと判断して、パターン 23 へ移ります。



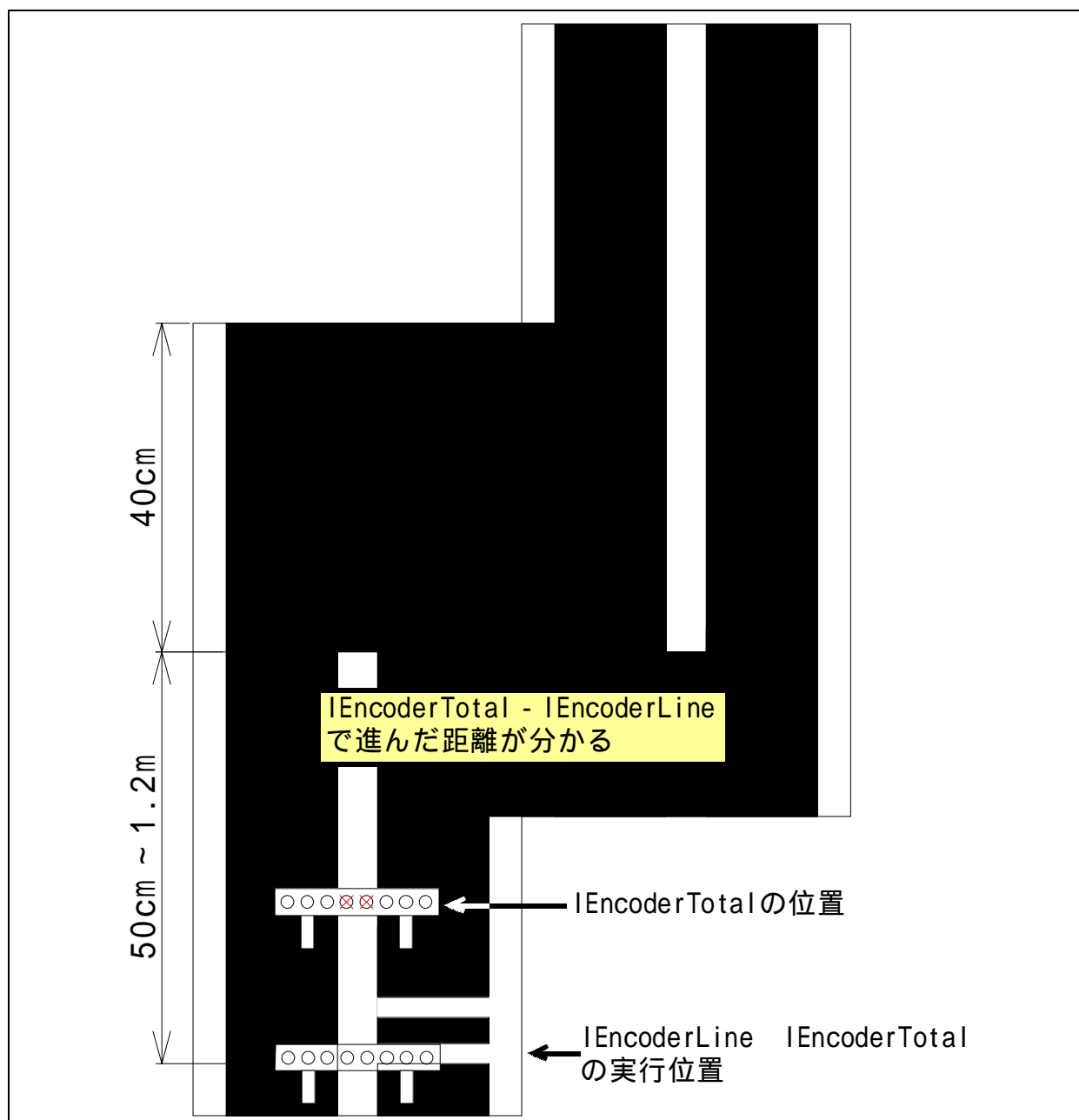
16.5.5 パターン 51 右ハーフライン検出時の積算値を取得

```

586 :     case 51:
587 :         /* 1本目の右ハーフライン検出時の処理 */
588 :         IEncoderLine = IEncoderTotal;
589 :         led_out( 0x2 );
590 :         handle( 0 );
591 :         speed( 0 ,0 );
592 :         pattern = 52;
593 :         cnt1 = 0;
594 :         break;

```

右ハーフラインを検出した瞬間の積算値 IEncoderTotal の値を、IEncoderLine にコピーしています。「IEncoderTotal - IEncoderLine」で、右ハーフラインを検出してからのパルス数が分かります。要は、**右ハーフラインから進んだ距離**が分かります。





### 16.5.6 パターン 52 2本目を読み飛ばす

```
596 :     case 52:
597 :         /* 2本目を読み飛ばす */
598 :         if( IEncoderTotal-IEncoderLine >= 97 ) { /* 約10cm */
599 :             pattern = 53;
600 :             cnt1 = 0;
601 :         }
602 :         break;
```

598 行で、10cm 進んだかチェックしています。距離は、1 本目の白線 2cm + 黒部分 3cm + 2 本目の白線 2cm で、合計 7cm です。余裕を見て 10cm としています。次のような意味です。

$IEncoderTotal - IEncoderLine \geq 10cm$

現在の積算値 - 右ハーフラインを検出したときの積算値  $\geq 10cm$

右ハーフライン検出後に進んだパルス数(距離)  $\geq 10cm$

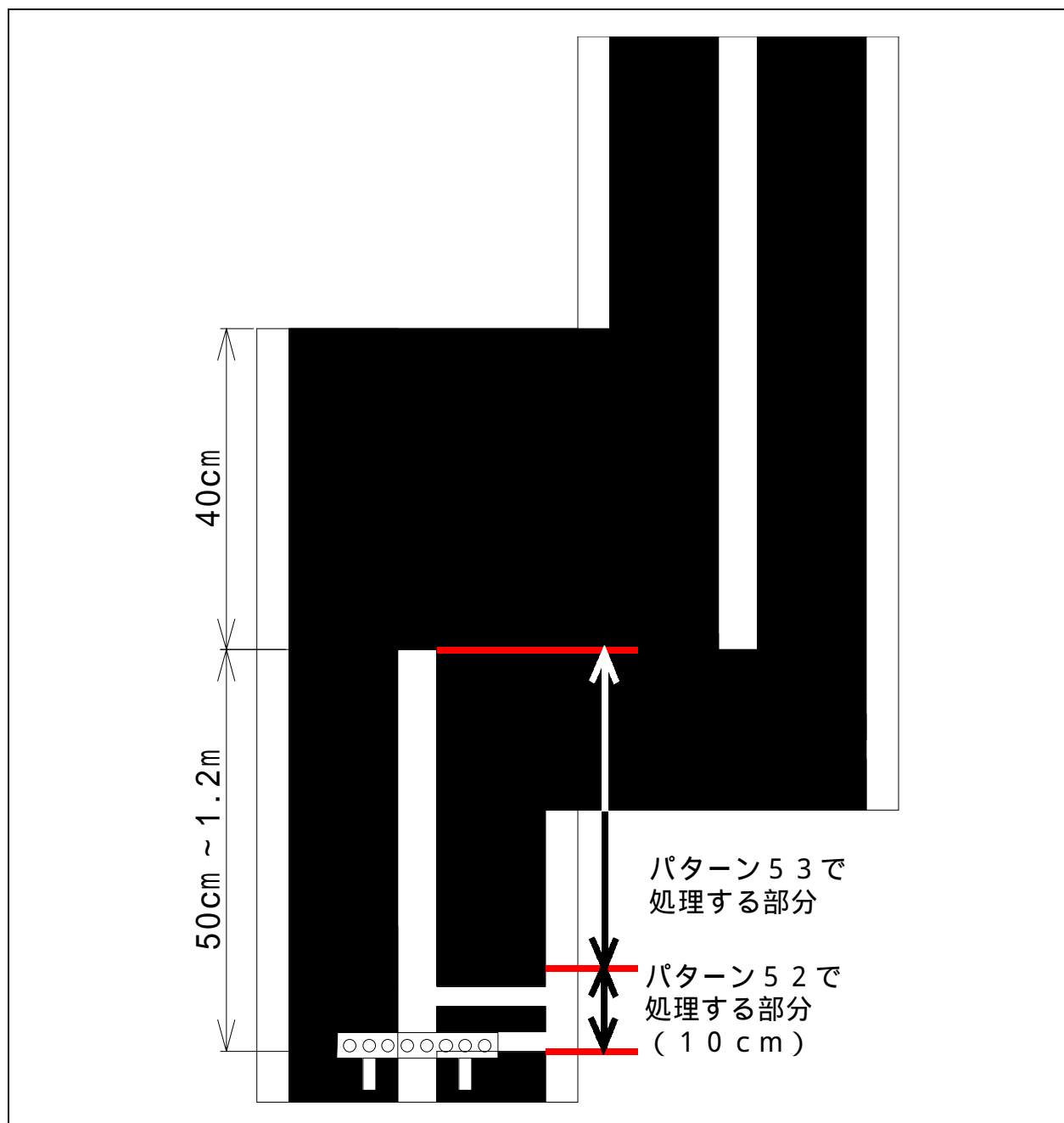
今回のエンコーダは 1m で 965 パルスのエンコーダなので、10cm 進んだかどうかチェックするには、

$1m : 965 \text{ パルス} = 0.1m : x \text{ パルス}$

$x = 96.5 \text{ パルス}$

と、右ハーフラインを検出した瞬間から 97 パルス以上になったかプログラムで見れば良いことになります。

97 パルス以上になると 10cm 進んだと判断して、パターン 53 へ移ります。



16.5.7 パターン 61～62 左ハーフライン部分の処理

```
650 :     case 61:
651 :         /* 1本目の左ハーフライン検出時の処理 */
652 :         IEncoderLine = IEncoderTotal;
653 :         led_out( 0x1 );
654 :         handle( 0 );
655 :         speed( 0 ,0 );
656 :         pattern = 62;
657 :         cnt1 = 0;
658 :         break;
659 :
660 :     case 62:
661 :         /* 2本目を読み飛ばす */
662 :         if( IEncoderTotal-IEncoderLine >= 97 ) { /* 約10cm */
663 :             pattern = 63;
664 :             cnt1 = 0;
665 :         }
666 :         break;
```

パターン 61、62 は、パターン 51、52 部分と比べて、右ハーフラインが左ハーフラインに変わるだけです。652 行で左ハーフラインを検出したときの距離を記憶します。すぐにパターン 62 へ移ります。662 行で、10cm 進んだかチェックします。進んだならば、パターン 63 へ移ります。

16.5.8 入出力設定、ITU 設定の変更

```

722 : /****** */
723 : /* H8/3048F-ONE 内蔵周辺機能 初期化 */
724 : /****** */
725 : void init( void )
726 : {
727 :     /* I/O ポートの入出力設定 */
728 :     P1DDR = 0xff;
729 :     P2DDR = 0xff;
730 :     P3DDR = 0x8e;          /* スイッチ、EEP-ROM */
731 :     P4DDR = 0xff;          /* LCD 接続 */
732 :     P5DDR = 0xff;
733 :     P6DDR = 0xf0;          /* CPU 基板上的 DIP SW */
734 :     P8DDR = 0xff;
735 :     P9DDR = 0xe3;          /* bit4,2:sw bit3:232c */
736 :     PADDR = 0xf6;          /* 3:bar センサ 0:エンコーダ */
737 :     PBDR = 0xc0;
738 :     PBDDR = 0xfe;          /* モータドライブ基板 Vol.3 */
739 :     /* センサ基板の P7 は、入力専用なので入出力設定はありません */
740 :
741 :     /* ITU0 1ms 毎の割り込み */
742 :     ITU0_TCR = 0x23;
743 :     ITU0_GRA = TIMER_CYCLE;
744 :     ITU0_IER = 0x01;
745 :
746 :     /* ITU2 パルス入力の設定 */
747 :     ITU2_TCR = 0x04;          /* PA0 端子のパルスでカウント*/
748 :
749 :     /* ITU3,4 リセット同期 PWM モード 左右モータ、サーボ用 */
750 :     ITU3_TCR = 0x23;
751 :     ITU_FCR = 0x3e;
752 :     ITU3_GRA = PWM_CYCLE;    /* 周期の設定 */
753 :     ITU3_GRB = ITU3_BRB = 0; /* 左モータの PWM 設定 */
754 :     ITU4_GRA = ITU4_BRA = 0; /* 右モータの PWM 設定 */
755 :     ITU4_GRB = ITU4_BRB = SERVO_CENTER; /* サーボの PWM 設定 */
756 :     ITU_TOER = 0x38;
757 :
758 :     /* ITU のカウントスタート */
759 :     ITU_STR = 0x0d;
760 : }

```

736 行のポート A の入出力設定を変更します。ポート A の bit0 は、今まで未接続だったので出力用に設定してあります。今回からエンコーダのパルス入力に変わったので、bit0 を入力設定の"0"にします。

ポート A の入出力設定は、

ビット	7	6	5	4	3	2	1	0
ポート A の入出力設定	出力	出力	出力	出力	スタートパ ー検出セン サ基板入力	出力	出力	ロータリ エンコー ダ入力

PADDR への設定値は、出力"1"、入力"0"にすれば良いだけです。

ビット	7	6	5	4	3	2	1	0
ポート A の 入出力設定	1	1	1	1	0	1	1	<b>0</b>

16進数に直すと、1111 0110 0xf6 となります。

747 行で ITU2 を外部パルス入力用とするために、ITU2\_TCR を設定しています。これは、

- ・ITU2\_CNT のクリアはしない
- ・外部エッジの立ち上がりエッジでカウント
- ・外部クロックの入力端子は TCLKA 端子(PA0)を選択

の設定です。この設定により、PA0 端子がパルス入力端子となります。入力されたパルスの数は自動的に、ITU2\_CNT に入ります。ちなみに、ITU2\_TCR の値を 0x14 にすると、立ち上がり、立ち下がり両エッジでカウントする設定となり、2 倍のカウントになります。エンコーダを自作して、パルス数の少ない場合は有効です(ただし、穴の空いている距離と空いていない距離を等間隔にする必要があります)。

759 行で、ITU のカウントをスタートさせます。ITU2 を追加したので ITU0,2,3 をスタートさせます。

### 16.5.9 割り込み処理プログラムの改造

```

762 : /*****/
763 : /* ITU0 割り込み処理 */
764 : /*****/
765 : #pragma interrupt( interrupt_timer0 )
766 : void interrupt_timer0( void )
767 : {
768 :     unsigned int i;
769 :
770 :     ITU0_TSR &= 0xfe;          /* フラグクリア */
771 :     cnt0++;
772 :     cnt1++;
773 :
774 :     /* LCD 表示処理用関数です。1ms 毎に実行します。 */
775 :     lcdShowProcess();
776 :     /* 拡張スイッチ用関数です。1ms 毎に実行します。 */
777 :     switchProcess();
778 :     /* ブザー処理用関数です。1ms 毎に実行します。 */
779 :     beepProcess();
780 :
781 :     /* エンコーダ関連 */
782 :     iTimer10++;
783 :     if( iTimer10 >= 10 ) {
784 :         iTimer10 = 0;
785 :         i = ITU2_CNT;
786 :         iEncoder      = i - uEncoderBuff;
787 :         lEncoderTotal += iEncoder;
788 :         if( iEncoder > iEncoderMax )
789 :             iEncoderMax = iEncoder;
790 :         uEncoderBuff = i;
791 :     }
792 : }

```

太字の部分が、ロータリエンコーダ処理として追加した部分です。詳しくは、「ロータリエンコーダ実習マニュアル」を参照してください。

## 16.6 エンコーダの回転数が違う場合の変更点

このサンプルプログラムは、100パルス/回転、エンコーダのタイヤ直径33mmのエンコーダを使用した場合です。条件が違うとき、プログラムを変更しなければいけない部分を下記に示します。


行番号	元の数値	変更後の数値
500	97	10cm 進んだときのパルス数を入れます。 例)16 パルス/回転、直径 21mm なら <b>24</b>
598	97	10cm 進んだときのパルス数を入れます。 例)16 パルス/回転、直径 21mm なら <b>24</b>
662	97	10cm 進んだときのパルス数を入れます。 例)16 パルス/回転、直径 21mm なら <b>24</b>

## 17. プロジェクト「tr\_22」 大カーブで速いとブレーキ(エンコーダ使用)

### 17.1 内容

前回のプロジェクトは、距離を検出しました。今回は、ロータリエンコーダを使用してスピードを検出します。大カーブで設定値以上のスピードになるとブレーキをかけるように、プログラムを追加します。

### 17.2 プロジェクトの構成

 <b>tr_22</b>	·tr_22tart.src
Assembly source file	ベクタアドレスの設定、スタートアップルーチンが記述されています。
tr_22start.src	·car_printf2.c
C source file	セクションの初期化、printf 文、scanf 文を実行します。
beep.c	·lcd2.c
car_printf2.c	LCD 制御を行います。
eeprom.c	·switch.c
lcd2.c	スイッチ制御を行います。
switch.c	·beep.c
tr_22.c	ブザー制御を行います。
Dependencies	·eeprom.c
beep.h	EEP-ROM 制御を行います。
eeprom.h	·tr_22.c
h8_3048.h	メインプログラムです。
lcd2.h	
switch.h	

### 17.3 プログラム「tr\_22.c」

プログラムのゴシック体部分が追加、変更した部分です。

前略

```

47 : /* EEP-ROM関連 */
48 : #define      EEP_ROM_SIZE      16      /* EEP-ROM使用サイズ      */
49 :
50 : #define      EEPROM_CHECK      0x00    /* EEP-ROMチェック      */
51 : #define      EEPROM_SERVO      0x01    /* サーボセンタ値      */
52 : #define      EEPROM_PWM        0x02    /* PWM値      */
53 : #define      EEPROM_CURVE_ENC  0x03    /* 大カーブのエンコーダ値      */
54 : #define      EEPROM_CRANK_PWM  0x04    /* クランク部分のPWM      */
55 :
56 : /*=====*/
57 : /* プロトタイプ宣言      */
58 : /*=====*/
59 : void init( void );
60 : void timer( unsigned long timer_set );
61 : int check_crossline( void );
62 : int check_rightline( void );
63 : int check_leftline( void );
64 : unsigned char sensor_inp( unsigned char mask );
65 : unsigned char dipsw_get( void );
66 : unsigned char pushsw_get( void );
67 : unsigned char startbar_get( void );
68 : void led_out( unsigned char led );
69 : void speed( int accele_l, int accele_r );
70 : void speed2( int accele_l, int accele_r );
71 : void handle( int angle );
72 : char unsigned bit_change( char unsigned in );
73 : int diff( int pwm );

```

中略

## トレーニングボード 実習マニュアル(kit06 版)

```

108 : /*****
109 : /* メインプログラム */
110 : /*****
111 : void main( void )
112 : {
113 :     int    i, j;
114 :     int    lcd_pattern = 2;
115 :
116 :     /* マイコン機能の初期化 */
117 :     init(); /* 初期化 */
118 :     set_ccr( 0x00 ); /* 全体割り込み許可 */
119 :     initLcd(); /* LCD初期化 */
120 :     initSwitch(); /* スイッチ初期化 */
121 :     initBeep(); /* ブザー初期化 */
122 :     initEeprom(); /* EEPROM初期化 */
123 :
124 :     /*EEP-ROMのチェック */
125 :     if( readEeprom( EEPROM_CHECK ) != 0x2006 ) {
126 :         /*
127 :         IDのチェック EEPROMを初めて使うかどうか
128 :         0x00番地にIDが書かれていなければ初めて使うと判断して初期化する
129 :         */
130 :         eep_buff[EEPROM_CHECK] = 0x2006;
131 :         eep_buff[EEPROM_SERVO] = SERVO_CENTER;
132 :         eep_buff[EEPROM_PWM] = 50;
133 :         eep_buff[EEPROM_CURVE_ENC] = 10;
134 :         eep_buff[EEPROM_CRANK_PWM] = 40;
135 :     } else {
136 :         /* 2回目以降の使用の場合、データ読み込み */
137 :         for( i=0; i<EEP_ROM_SIZE; i++ ) {
138 :             eep_buff[ i ] = readEeprom( i );
139 :         }
140 :     }
141 :
142 :     /* マイコンカーの状態初期化 */
143 :     handle( 0 );
144 :     speed( 0, 0 );
145 :
146 :     while( 1 ) {
147 :         switch( pattern ) {
148 :
149 :             /*****
150 :             パターンについて
151 :             0: スイッチ入力待ち
152 :             1: スタートバーが開いたかチェック
153 :             11: 通常トレース
154 :             12: 右へ大曲げの終わりのチェック
155 :             13: 左へ大曲げの終わりのチェック
156 :             21: 1本目のクロスライン検出時の処理
157 :             22: 2本目を読み飛ばす
158 :             23: クロスライン後のトレース、クランク検出
159 :             31: 左クランククリア処理 安定するまで少し待つ
160 :             32: 左クランククリア処理 曲げ終わりのチェック
161 :             41: 右クランククリア処理 安定するまで少し待つ
162 :             42: 右クランククリア処理 曲げ終わりのチェック
163 :             51: 1本目の右ハーフライン検出時の処理
164 :             52: 2本目を読み飛ばす
165 :             53: 右ハーフライン後のトレース
166 :             54: 右レーンチェンジ終了のチェック
167 :             61: 1本目の左ハーフライン検出時の処理
168 :             62: 2本目を読み飛ばす
169 :             63: 左ハーフライン後のトレース
170 :             64: 左レーンチェンジ終了のチェック
171 :             *****/
172 :
173 :             case 0:
174 :                 /* スイッチ入力待ち */
175 :                 if( pushsw_get() ) {
176 :                     setBeepPattern( 0xc000 );
177 :                     /* 保存 */
178 :                     for( i=0; i<EEP_ROM_SIZE; i++ ) {
179 :                         writeEeprom( i, eep_buff[ i ] );
180 :                         while( !checkEeprom() ); /* 書き込み終了チェック */
181 :                     }
182 :                     pattern = 1;
183 :                     cnt1 = 0;
184 :                     break;
185 :                 }
186 :
187 :                 /* スイッチ4 設定値保存 */
188 :                 if( getSwFlag(SW_4) ) {
189 :                     setBeepPattern( 0x8000 );
190 :                     /* 保存 */
191 :                     for( i=0; i<EEP_ROM_SIZE; i++ ) {
192 :                         writeEeprom( i, eep_buff[ i ] );
193 :                         while( !checkEeprom() ); /* 書き込み終了チェック */
194 :                     }
195 :                     break;
196 :                 }
197 :                 /* スイッチ3 メニュー + 1 */
198 :                 if( getSwFlag(SW_3) ) {

```



トレーニングボード 実習マニュアル(kit06 版)

```

199 :         lcd_pattern++;
200 :         if( !lcd_pattern == 5 ) lcd_pattern = 1;
201 :     }
202 :     /* スイッチ2 メニュー - 1 */
203 :     if( getSwFlag(SW_2) ) {
204 :         lcd_pattern--;
205 :         if( !lcd_pattern == 0 ) lcd_pattern = 4;
206 :     }
207 :
208 :     /* スイッチ、LCD処理 */
209 :     switch( lcd_pattern ) {
210 :     case 1:
211 :         /* サーボセンタ値調整 */
212 :         i = eep_buff[EEPROM_SERVO];
213 :         if( getSwFlag(SW_1) ) {
214 :             i++;
215 :             if( i > 10000 ) i = 10000;
216 :         }
217 :         if( getSwFlag(SW_0) ) {
218 :             i--;
219 :             if( i < 1000 ) i = 1000;
220 :         }
221 :         eep_buff[EEPROM_SERVO] = i;
222 :         handle( 0 );
223 :
224 :         /* LCD処理 */
225 :         lcdPosition( 0, 0 );
226 :         /* 0123456789a..ef 1行16文字 */
227 :         lcdPrintf( "1 servo = %05d ", i );
228 :         /* 01234567..89abcde.f 1行16文字 */
229 :         lcdPrintf( "sensor=%02x bar=%d ",
230 :             sensor_inp( 0xff ), startbar_get() );
231 :         break;
232 :
233 :     case 2:
234 :         /* PWM調整 */
235 :         i = eep_buff[EEPROM_PWM];
236 :         if( getSwFlag(SW_1) ) {
237 :             i++;
238 :             if( i > 100 ) i = 100;
239 :         }
240 :         if( getSwFlag(SW_0) ) {
241 :             i--;
242 :             if( i < 0 ) i = 0;
243 :         }
244 :         eep_buff[EEPROM_PWM] = i;
245 :
246 :         /* LCD処理 */
247 :         lcdPosition( 0, 0 );
248 :         /* 012345678..abcdef 1行16文字 */
249 :         lcdPrintf( "2 pwm = %03d ", i );
250 :         /* 01234567..89abcde.f 1行16文字 */
251 :         lcdPrintf( "sensor=%02x bar=%d ",
252 :             sensor_inp( 0xff ), startbar_get() );
253 :         break;
254 :
255 :     case 3:
256 :         /* 大カーブのエンコーダ値調整 */
257 :         i = eep_buff[EEPROM_CURVE_ENC];
258 :         if( getSwFlag(SW_1) ) {
259 :             i++;
260 :             if( i > 100 ) i = 100;
261 :         }
262 :         if( getSwFlag(SW_0) ) {
263 :             i--;
264 :             if( i < 0 ) i = 0;
265 :         }
266 :         eep_buff[EEPROM_CURVE_ENC] = i;
267 :
268 :         /* LCD処理 */
269 :         lcdPosition( 0, 0 );
270 :         /* 0123456789abcd..f 1行16文字 */
271 :         lcdPrintf( "3 curve enc =%03d", i );
272 :         /* 0..234..678....f 1行16文字 */
273 :         lcdPrintf( "%03d %03d %08ld",
274 :             iEncoder, iEncoderMax, IEncoderTotal );
275 :         break;
276 :
277 :     case 4:
278 :         /* クロスライン検出後のPWM調整 */
279 :         i = eep_buff[EEPROM_CRANK_PWM];
280 :         if( getSwFlag(SW_1) ) {
281 :             i++;
282 :             if( i > 100 ) i = 100;
283 :         }
284 :         if( getSwFlag(SW_0) ) {
285 :             i--;
286 :             if( i < 0 ) i = 0;
287 :         }
288 :         eep_buff[EEPROM_CRANK_PWM] = i;
289 :

```

## トレーニングボード 実習マニュアル(kit06 版)

```

290 :         /* LCD処理 */
291 :         lcdPosition( 0, 0 );
292 :         /* 0123456789abcd..f 1行16文字 */
293 :         lcdPrintf( "4 crankpwm = %03d", i );
294 :         /* 01234567..89abcde.f 1行16文字 */
295 :         lcdPrintf( "sensor=%02x bar=%d ",
296 :                 sensor_inp( 0xff ), startbar_get() );
297 :         break;
298 :     }
299 :
300 :     if( cnt1 < 100 ) {           /* LED点滅処理          */
301 :         led_out( 0x1 );
302 :     } else if( cnt1 < 200 ) {
303 :         led_out( 0x2 );
304 :     } else {
305 :         cnt1 = 0;
306 :     }
307 :     break;

```

中略

```

407 :     case 12:
408 :         /* 右へ大曲げの終わりのチェック */
409 :         if( check_crossline() ) { /* 大曲げ中もクロスラインチェック */
410 :             pattern = 21;
411 :             break;
412 :         }
413 :         if( check_rightline() ) { /* 右ハーフラインチェック */
414 :             pattern = 51;
415 :             break;
416 :         }
417 :         if( check_leftline() ) { /* 左ハーフラインチェック */
418 :             pattern = 61;
419 :             break;
420 :         }
421 :         if( iEncoder >= eep_buff[EEPROM_CURVE_ENC] ) {
422 :             /* エンコーダ値によりPWM値設定 */
423 :             setBeepPattern( 0xc000 );
424 :             i = 0;
425 :         } else {
426 :             i = 60;
427 :         }
428 :         switch( sensor_inp(MASK3_3) ) {
429 :         case 0x06:
430 :             pattern = 11;
431 :             break;
432 :         case 0x03:
433 :             handle( 20 );
434 :             speed2( i, diff(i) );
435 :             break;
436 :         case 0x81:
437 :             handle( 25 );
438 :             speed2( i, diff(i) );
439 :             break;
440 :         case 0xc1:
441 :         case 0xc0:
442 :             handle( 30 );
443 :             speed2( i, diff(i) );
444 :             break;
445 :         case 0x60:
446 :             handle( 35 );
447 :             speed2( 0, 0 );
448 :             break;
449 :         }
450 :         break;
451 :
452 :     case 13:
453 :         /* 左へ大曲げの終わりのチェック */
454 :         if( check_crossline() ) { /* 大曲げ中もクロスラインチェック */
455 :             pattern = 21;
456 :             break;
457 :         }
458 :         if( check_rightline() ) { /* 右ハーフラインチェック */
459 :             pattern = 51;
460 :             break;
461 :         }
462 :         if( check_leftline() ) { /* 左ハーフラインチェック */
463 :             pattern = 61;
464 :             break;
465 :         }
466 :         if( iEncoder >= eep_buff[EEPROM_CURVE_ENC] ) {
467 :             /* エンコーダ値によりPWM値設定 */
468 :             setBeepPattern( 0xc000 );
469 :             i = 0;
470 :         } else {
471 :             i = 60;
472 :         }
473 :         switch( sensor_inp(MASK3_3) ) {
474 :         case 0x60:
475 :             pattern = 11;
476 :             break;

```

トレーニングボード 実習マニュアル(kit06 版)

```

477 :         case 0xc0:
478 :             handle( -20 );
479 :             speed2( diff(i), i );
480 :             break;
481 :         case 0x81:
482 :             handle( -25 );
483 :             speed2( diff(i), i );
484 :             break;
485 :         case 0x83:
486 :         case 0x03:
487 :             handle( -30 );
488 :             speed2( diff(i), i );
489 :             break;
490 :         case 0x06:
491 :             handle( -35 );
492 :             speed2( 0, 0 );
493 :             break;
494 :     }
495 :     break;

```

中略

```

973 :  /******
974 :  /* 速度制御2
975 :  /* 引数 左モータ:-100~100 , 右モータ:-100~100
976 :  /*      0で停止、100で正転100%、-100で逆転100%
977 :  /*      ティップスイッチは関係なし
978 :  /******
979 :  void speed2( int accele_l, int accele_r )
980 :  {
981 :     unsigned long  speed_max;
982 :
983 :     speed_max = PWM_CYCLE - 1;
984 :
985 :     /* 左モータ */
986 :     if( accele_l >= 0 ) {
987 :         PBDR &= 0xfb;
988 :         ITU3_BRB = speed_max * accele_l / 100;
989 :     } else {
990 :         PBDR |= 0x04;
991 :         accele_l = -accele_l;
992 :         ITU3_BRB = speed_max * accele_l / 100;
993 :     }
994 :
995 :     /* 右モータ */
996 :     if( accele_r >= 0 ) {
997 :         PBDR &= 0xf7;
998 :         ITU4_BRA = speed_max * accele_r / 100;
999 :     } else {
1000 :         PBDR |= 0x08;
1001 :         accele_r = -accele_r;
1002 :         ITU4_BRA = speed_max * accele_r / 100;
1003 :     }
1004 : }

```

以下、略

## 17.4 プログラムの解説

### 17.4.1 EEP-ROM エリアの変更

```

47 : /* EEP-ROM 関連 */
48 : #define EEP_ROM_SIZE 16 /* EEP-ROM 使用サイズ */
49 :
50 : #define EEPROM_CHECK 0x00 /* EEP-ROM チェック */
51 : #define EEPROM_SERVO 0x01 /* サーボセンタ値 */
52 : #define EEPROM_PWM 0x02 /* PWM 値 */
53 : #define EEPROM_CURVE_ENC 0x03 /* 大カーブのエンコーダ値 */
54 : #define EEPROM_CRANK_PWM 0x04 /* クランク部分の PWM */
    
```

PWMを設定するEEP-ROMの番地を決めます。今まで0x03番地は、大カーブのPWM値でした。ここを大カーブのエンコーダ値に変更して、EEPROM\_CURVE\_PWMとして定義します。

番地	番地名	内容	初期値
0x00	EEPROM_CHECK	EEP-ROM チェック用 0x2006 で無ければ、初期値を読み込む	0x2006
0x01	EEPROM_SERVO	サーボセンタ値	5000
0x02	EEPROM_PWM	PWM 値	50
<b>0x03</b>	<b>EEPROM_CURVE_ENC</b>	<b>大カーブでのエンコーダ値(変更)</b>	<b>10</b>
0x04	EEPROM_CRANK_PWM	クロスライン検出後の PWM 値	40
0x05 ~ 0x0f	未定義		

### 17.4.2 EEP-ROM から読み込み

```

124 : /*EEP-ROM のチェック */
125 : if( readEeprom( EEPROM_CHECK ) != 0x2006 ) {
126 :     /*
127 :     ID のチェック EEP-ROM を初めて使うかどうか
128 :     0x00 番地に ID が書かれていなければ初めて使うと判断して初期化する
129 :     */
130 :     eep_buff[EEPROM_CHECK] = 0x2006;
131 :     eep_buff[EEPROM_SERVO] = SERVO_CENTER;
132 :     eep_buff[EEPROM_PWM] = 50;
133 :     eep_buff[EEPROM_CURVE_ENC] = 10;
134 :     eep_buff[EEPROM_CRANK_PWM] = 40;
135 : } else {
136 :     /* 2 回目以降の使用の場合、データ読み込み */
137 :     for( i=0; i<EEP_ROM_SIZE; i++ ) {
138 :         eep_buff[ i ] = readEeprom( i );
139 :     }
140 : }
    
```

133 行で、eep\_buff[EEPROM\_CURVE\_ENC]変数の初期化を追加しています。

### 17.4.3 スイッチ入力待ち

```

197 :      /* スイッチ 3 メニュー + 1 */
198 :      if( getSwFlag(SW_3) ) {
199 :          lcd_pattern++;
200 :          if( lcd_pattern == 5 ) lcd_pattern = 1;
201 :      }
202 :      /* スイッチ 2 メニュー - 1 */
203 :      if( getSwFlag(SW_2) ) {
204 :          lcd_pattern--;
205 :          if( lcd_pattern == 0 ) lcd_pattern = 4;
206 :      }

```

SW\_3 で LCD に表示する内容を次へ進めます。SW\_2 で戻します。今回は lcd\_pattern=3 を、大カーブの PWM 値の設定から大カーブのエンコーダ値調整へ変更しただけなので表示画面の追加はありません。

lcd_pattern	内容
1	サーボセンタ値の調整
2	PWM 値の調整
3	<b>大カーブのエンコーダ値調整(変更)</b>
4	クロスライン後の PWM 値の調整

```

208 :      /* スイッチ、LCD 処理 */
209 :      switch( lcd_pattern ) {
210 :      case 1:
211 :          /* サーボセンタ値調整 */
中略
231 :          break;
232 :
233 :      case 2:
234 :          /* PWM 調整 */
中略
253 :          break;
254 :
255 :      case 3:
256 :          /* 大カーブのエンコーダ値調整 */
257 :          i = eep_buff[EEPROM_CURVE_ENC];
258 :          if( getSwFlag(SW_1) ) {
259 :              i++;
260 :              if( i > 100 ) i = 100;
261 :          }
262 :          if( getSwFlag(SW_0) ) {
263 :              i--;
264 :              if( i < 0 ) i = 0;
265 :          }
266 :          eep_buff[EEPROM_CURVE_ENC] = i;
267 :
268 :          /* LCD 処理 */
269 :          lcdPosition( 0, 0 );
270 :          /* 0123456789abcd..f 1 行 16 文字 */
271 :          lcdPrintf( "3 curve enc =%03d", i );

```

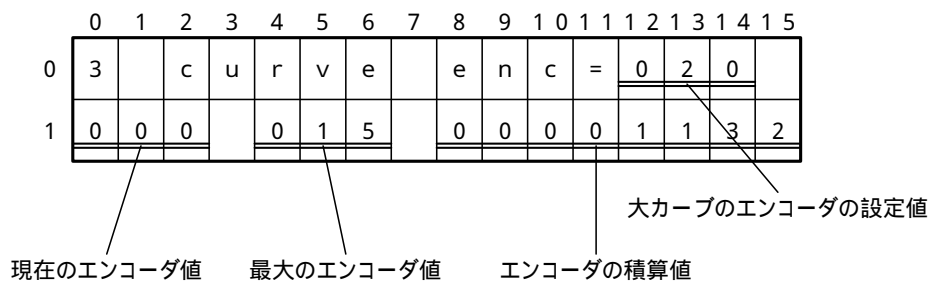
```

272 :          /* 0..234..678...f 1行16文字 */
273 :          lcdPrintf( "%03d %03d %08ld",
274 :                    iEncoder, iEncoderMax, lEncoderTotal );
275 :          break;
276 :
277 :          case 4:
中略
297 :          break;
298 :      }

```

209 行で、lcd\_pattern の番号によりプログラムをジャンプします。3 なら 255 行へ飛びます。  
 257 行で、eep\_buff[EEPROM\_CURVE\_ENC]変数の値をいったん i 変数に代入します。  
 258 行で、SW\_1 が押されているかチェックします。押されていれば、i 変数を1つ増加させます。  
 262 行で、SW\_0 が押されているかチェックします。押されていれば、i 変数を1つ減少させます。  
 266 行で、i 変数の値を eep\_buff[EEPROM\_CURVE\_ENC]変数へ代入します。SW\_1、SW\_2 で操作された値が代入されます。  
 269 行から LCD に大カーブのエンコーダ値を表示します。2 行目が余っているのが現在のエンコーダ状態を表示させています。

lcd\_pattern が 3 のとき、下記のように表示されます。



SW\_1 と SW\_0 で大カーブのエンコーダ値を増減させます。

17.4.4 パターン 12 右へ大曲げの終わりのチェック

```

407 :     case 12:
408 :         /* 右へ大曲げの終わりのチェック */
409 :         if( check_crossline() ) {          /* 大曲げ中もクロスラインチェック */
410 :             pattern = 21;
411 :             break;
412 :         }
413 :         if( check_rightline() ) {          /* 右ハーフラインチェック */
414 :             pattern = 51;
415 :             break;
416 :         }
417 :         if( check_leftline() ) {          /* 左ハーフラインチェック */
418 :             pattern = 61;
419 :             break;
420 :         }
421 :         if( iEncoder >= eep_buff[EEPROM_CURVE_ENC] ) {
422 :             /* エンコーダ値により PWM 値設定 */
423 :             setBeepPattern( 0xc000 );
424 :             i = 0;
425 :         } else {
426 :             i = 60;
427 :         }
428 :         switch( sensor_inp(MASK3_3) ) {
429 :         case 0x06:
430 :             pattern = 11;
431 :             break;
432 :         case 0x03:
433 :             handle( 20 );
434 :             speed2( i, diff(i) );
435 :             break;
436 :         case 0x81:
437 :             handle( 25 );
438 :             speed2( i, diff(i) );
439 :             break;
440 :         case 0xc1:
441 :         case 0xc0:
442 :             handle( 30 );
443 :             speed2( i, diff(i) );
444 :             break;
445 :         case 0x60:
446 :             handle( 35 );
447 :             speed2( 0, 0 );
448 :             break;
449 :         }
450 :         break;

```

421 行で、現在のエンコーダ値「iEncoder」と、設定値「eep\_buff[EEPROM\_CURVE\_ENC]」を比較します。

現在のエンコーダ値の方が大きければ、設定値よりスピードが速いと判断して、PWM 値を 0% にしています(424 行)。現在のエンコーダ値が小さければ、PWM 値を 60% とします(426 行)。それぞれ、変数 i に代入して、speed 関数に設定するスピード値を i としています。

eep\_buff[EEPROM\_CURVE\_PWM]で設定した数値は何を意味しているのでしょうか。「1m/s で進んだときの10ms 間にカウントするパルス数は 9.65 パルス」の場合を例として考えます。すなわち、eep\_buff[EEPROM\_CURVE\_PWM]に 10(9.65 を四捨五入)を設定すれば、1m/s 以上なら、PWM は0%、以下なら60%ということになります。もし、カーブでのブレーキを 2m/s に設定したければ、

$$1\text{m/s} : 9.65 = 2\text{m/s} : x$$

$$x = 19.30 \quad 19$$

ということになり、スタート前の大カーブでのエンコーダ値調整画面で、19 を設定すればOKです。「1m/s で進んだときの 10ms 間にカウントするパルス数」を常に頭に入れておきながら設定します。

一般的に、下記のような関係になります。

	特徴	長所	短所
設定値が小さい場合	ブレーキを多くかける	カーブで脱輪しづらい	タイムロスが多くなる
設定値が大きい場合	ブレーキを余りかけない	タイムロスは少ない	カーブで脱輪しやすい

各自のマイコンカーに合わせて、一番きついカーブで脱輪ないように調整します。



### 17.4.5 speed2 関数

```

973 : /*****/
974 : /* 速度制御2 */
975 : /* 引数 左モータ:-100~100 , 右モータ:-100~100 */
976 : /*      0で停止、100で正転100%、-100で逆転100% */
977 : /*      ディップスイッチは関係なし */
978 : /*****/
979 : void speed2( int accele_l, int accele_r )
980 : {
981 :     unsigned long  speed_max;
982 :
983 :     speed_max = PWM_CYCLE - 1;
984 :
985 :     /* 左モータ */
986 :     if( accele_l >= 0 ) {
987 :         PBDR &= 0xfb;
988 :         ITU3_BRB = speed_max * accele_l / 100;
989 :     } else {
990 :         PBDR |= 0x04;
991 :         accele_l = -accele_l;
992 :         ITU3_BRB = speed_max * accele_l / 100;
993 :     }
994 :
995 :     /* 右モータ */
996 :     if( accele_r >= 0 ) {
997 :         PBDR &= 0xf7;
998 :         ITU4_BRA = speed_max * accele_r / 100;
999 :     } else {
1000 :         PBDR |= 0x08;
1001 :         accele_r = -accele_r;
1002 :         ITU4_BRA = speed_max * accele_r / 100;
1003 :     }
1004 : }

```

speed 関数を良く見ると... speed2 関数？ 2 が付いています。  
speed 関数は、

speed 関数の引数の割合 × eep\_buff[EEPROM\_PWM] ÷ 100 = **実際にモータに出力される PWM 値**

でした。エンコーダを使えば、パルス数によってスピードを制御するので eep\_buff[EEPROM\_PWM] でスピードを落とす必要がありません。そこで eep\_buff[EEPROM\_PWM] には関係なく、**speed 関数の引数のみでモータに出力される PWM 値を決める speed2 関数を作りました**。983 行のように、最大値は PWM 周期 - 1 となり、eep\_buff[EEPROM\_PWM] の値は関係ありません。

**speed2 関数の引数の割合 = 実際にモータに出力される PWM 値**

となります。**エンコーダ値を比較してスピード制御する部分には、speed2 関数を使用します**。

関数を追加したときは 70 行目のように、プロトタイプ宣言も追加してください。

17.4.6 パターン 13 左へ大曲げの終わりのチェック

```

452 :     case 13:
453 :         /* 左へ大曲げの終わりのチェック */
454 :         if( check_crossline() ) {          /* 大曲げ中もクロスラインチェック */
455 :             pattern = 21;
456 :             break;
457 :         }
458 :         if( check_rightline() ) {          /* 右ハーフラインチェック */
459 :             pattern = 51;
460 :             break;
461 :         }
462 :         if( check_leftline() ) {          /* 左ハーフラインチェック */
463 :             pattern = 61;
464 :             break;
465 :         }
466 :         if( iEncoder >= eep_buff[EEPROM_CURVE_ENC] ) {
467 :             /* エンコーダ値により PWM 値設定 */
468 :             setBeepPattern( 0xc000 );
469 :             i = 0;
470 :         } else {
471 :             i = 60;
472 :         }
473 :         switch( sensor_inp(MASK3_3) ) {
474 :             case 0x60:
475 :                 pattern = 11;
476 :                 break;
477 :             case 0xc0:
478 :                 handle( -20 );
479 :                 speed2( diff(i), i );
480 :                 break;
481 :             case 0x81:
482 :                 handle( -25 );
483 :                 speed2( diff(i), i );
484 :                 break;
485 :             case 0x83:
486 :             case 0x03:
487 :                 handle( -30 );
488 :                 speed2( diff(i), i );
489 :                 break;
490 :             case 0x06:
491 :                 handle( -35 );
492 :                 speed2( 0, 0 );
493 :                 break;
494 :         }
495 :         break;

```

左へ大曲げの終わりのチェックも、右のとときと考え方は同じです。

466 行で、現在のエンコーダ値「iEncoder」と、設定値「eep\_buff[EEPROM\_CURVE\_ENC]」を比較します。

現在のエンコーダ値の方が大きければ、設定値よりスピードが速いと判断して、PWM 値を 0% にしています(469 行)。現在のエンコーダ値が小さければ、PWM 値を 60% とします(471 行)。それぞれ、変数 i に代入して、speed2 関数に設定するスピード値を i としています。

## 17.5 エンコーダの回転数が違う場合の変更点

このサンプルプログラムは、100 パルス / 回転、エンコーダのタイヤ直径 33mm のエンコーダを使用した場合です。条件が違うとき、プログラムを変更しなければいけない部分を下記に示します。

行番号	元の数値	変更後の数値
509	97	10cm 進んだときのパルス数を入れます。 例)16 パルス / 回転、直径 21mm なら <b>24</b>
607	97	10cm 進んだときのパルス数を入れます。 例)16 パルス / 回転、直径 21mm なら <b>24</b>
671	97	10cm 進んだときのパルス数を入れます。 例)16 パルス / 回転、直径 21mm なら <b>24</b>

## 18. プロジェクト「tr\_23」 クロスライン等検出後の速度設定(エンコーダ使用)

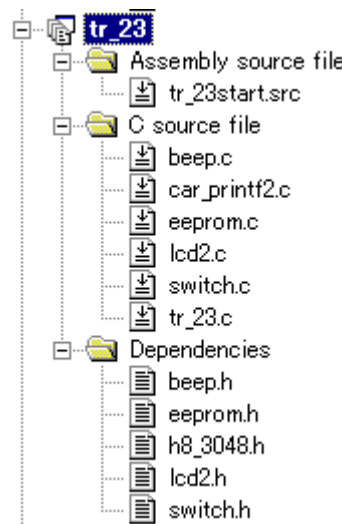
### 18.1 内容

マイコンカーがクロスラインを通過する速度は、クロスライン直前のコースレイアウトにより変わります。コースが長い直線の後では速い速度で、カーブ等の後では遅い速度で進入します。そのため、クロスライン検出後のブレーキ時間が同じなら、速いスピードで進入した場合は減速しきれずに脱輪、遅く進入した場合は減速しすぎてタイムロスにつながります。

そこで、クロスライン検出後のスピードをエンコーダで検出して、進入速度が違ってても一定のスピードで走行するようにします。

同様に、右ハーフライン、左ハーフライン進入時も速度制御します。

### 18.2 プロジェクトの構成

	<ul style="list-style-type: none"> <li>· tr_23tart.src ベクタアドレスの設定、スタートアップルーチンが記述されています。</li> <li>· car_printf2.c セクションの初期化、printf 文、scanf 文を実行します。</li> <li>· lcd2.c LCD 制御を行います。</li> <li>· switch.c スイッチ制御を行います。</li> <li>· beep.c ブザー制御を行います。</li> <li>· eeprom.c EEP-ROM 制御を行います。</li> <li>· tr_23.c メインプログラムです。</li> </ul>
--	---

### 18.3 プログラム「tr\_23.c」

プログラムのゴシック体部分が追加、変更した部分です。

前略

```

47 : /* EEP-ROM関連 */
48 : #define EEP_ROM_SIZE 16 /* EEP-ROM使用サイズ */
49 :
50 : #define EEPROM_CHECK 0x00 /* EEP-ROMチェック */
51 : #define EEPROM_SERVO 0x01 /* サーボセンタ値 */
52 : #define EEPROM_PWM 0x02 /* PWM値 */
53 : #define EEPROM_CURVE_ENC 0x03 /* 大カーブのエンコーダ値 */
54 : #define EEPROM_CRANK_ENC 0x04 /* クランク部分のエンコーダ値*/
    
```

中略

```

108 : /*******/
109 : /* メインプログラム */
110 : /*******/
111 : void main( void )
112 : {
113 :     int i, j;
114 :     int lcd_pattern = 2;
115 :
116 :     /* マイコン機能の初期化 */
117 :     init(); /* 初期化 */
118 :     set_ccr( 0x00 ); /* 全体割り込み許可 */
119 :     initLcd(); /* LCD初期化 */
    
```

## トレーニングボード 実習マニュアル(kit06 版)

```

120 :   initSwitch();           /* スイッチ初期化          */
121 :   initBeep();            /* ブザー初期化          */
122 :   initEeprom();         /* EEPROM初期化          */
123 :
124 :   /*EEP-ROMのチェック */
125 :   if( readEeprom( EEPROM_CHECK ) != 0x2006 ) {
126 :     /*
127 :     IDのチェック EEPROMを初めて使うかどうか
128 :     0x00番地にIDが書かれていなければ初めて使うと判断して初期化する
129 :     */
130 :     eep_buff[EEPROM_CHECK] = 0x2006;
131 :     eep_buff[EEPROM_SERVO] = SERVO_CENTER;
132 :     eep_buff[EEPROM_PWM] = 50;
133 :     eep_buff[EEPROM_CURVE_ENC] = 10;
134 :     eep_buff[EEPROM_CRANK_ENC] = 10;
135 :   } else {
136 :     /* 2回目以降の使用の場合、データ読み込み */
137 :     for( i=0; i<EEP_ROM_SIZE; i++ ) {
138 :       eep_buff[ i ] = readEeprom( i );
139 :     }
140 :   }
141 :
142 :   /* マイコンカーの状態初期化 */
143 :   handle( 0 );
144 :   speed( 0, 0 );
145 :
146 :   while( 1 ) {
147 :     switch( pattern ) {
148 :
149 :     /******
150 :     パターンについて
151 :     0: スイッチ入力待ち
152 :     1: スタートバーが開いたかチェック
153 :     11: 通常トレース
154 :     12: 右へ大曲げの終わりのチェック
155 :     13: 左へ大曲げの終わりのチェック
156 :     21: 1本目のクロスライン検出時の処理
157 :     22: 2本目を読み飛ばす
158 :     23: クロスライン後のトレース、クランク検出
159 :     31: 左クランククリア処理 安定するまで少し待つ
160 :     32: 右クランククリア処理 曲げ終わりのチェック
161 :     41: 右クランククリア処理 安定するまで少し待つ
162 :     42: 左クランククリア処理 曲げ終わりのチェック
163 :     51: 1本目の右ハーフライン検出時の処理
164 :     52: 2本目を読み飛ばす
165 :     53: 右ハーフライン後のトレース
166 :     54: 右レーンチェンジ終了のチェック
167 :     61: 1本目の左ハーフライン検出時の処理
168 :     62: 2本目を読み飛ばす
169 :     63: 左ハーフライン後のトレース
170 :     64: 左レーンチェンジ終了のチェック
171 :     *****/
172 :
173 :     case 0:
174 :       /* スイッチ入力待ち */
175 :       if( pushsw_get() ) {
176 :         setBeepPattern( 0xc000 );
177 :         /* 保存 */
178 :         for( i=0; i<EEP_ROM_SIZE; i++ ) {
179 :           writeEeprom( i, eep_buff[ i ] );
180 :           while( !checkEeprom() ); /* 書き込み終了チェック */
181 :         }
182 :         pattern = 1;
183 :         cnt1 = 0;
184 :         break;
185 :       }
186 :
187 :       /* スイッチ4 設定値保存 */
188 :       if( getSwFlag(SW_4) ) {
189 :         setBeepPattern( 0x8000 );
190 :         /* 保存 */
191 :         for( i=0; i<EEP_ROM_SIZE; i++ ) {
192 :           writeEeprom( i, eep_buff[ i ] );
193 :           while( !checkEeprom() ); /* 書き込み終了チェック */
194 :         }
195 :         break;
196 :       }
197 :       /* スイッチ3 メニュー + 1 */
198 :       if( getSwFlag(SW_3) ) {
199 :         lcd_pattern++;
200 :         if( lcd_pattern == 5 ) lcd_pattern = 1;
201 :       }
202 :       /* スイッチ2 メニュー - 1 */
203 :       if( getSwFlag(SW_2) ) {
204 :         lcd_pattern--;
205 :         if( lcd_pattern == 0 ) lcd_pattern = 4;
206 :       }
207 :
208 :       /* スイッチ、LCD処理 */
209 :       switch( lcd_pattern ) {
210 :       case 1:

```

トレーニングボード 実習マニュアル(kit06 版)

```

211 :         /* サーボセンタ値調整 */
212 :         i = eep_buff[EEPROM_SERVO];
213 :         if( getSwFlag(SW_1) ) {
214 :             i++;
215 :             if( i > 10000 ) i = 10000;
216 :         }
217 :         if( getSwFlag(SW_0) ) {
218 :             i--;
219 :             if( i < 1000 ) i = 1000;
220 :         }
221 :         eep_buff[EEPROM_SERVO] = i;
222 :         handle( 0 );
223 :
224 :         /* LCD処理 */
225 :         lcdPosition( 0, 0 );
226 :         /* 0123456789a..ef 1行16文字 */
227 :         lcdPrintf( "1 servo = %05d", i );
228 :         /* 01234567..89abcde.f 1行16文字 */
229 :         lcdPrintf( "sensor=%02x bar=%d",
230 :                 sensor_inp( 0xff ), startbar_get() );
231 :         break;
232 :
233 :     case 2:
234 :         /* PWM調整 */
235 :         i = eep_buff[EEPROM_PWM];
236 :         if( getSwFlag(SW_1) ) {
237 :             i++;
238 :             if( i > 100 ) i = 100;
239 :         }
240 :         if( getSwFlag(SW_0) ) {
241 :             i--;
242 :             if( i < 0 ) i = 0;
243 :         }
244 :         eep_buff[EEPROM_PWM] = i;
245 :
246 :         /* LCD処理 */
247 :         lcdPosition( 0, 0 );
248 :         /* 012345678..abcdef 1行16文字 */
249 :         lcdPrintf( "2 pwm = %03d", i );
250 :         /* 01234567..89abcde.f 1行16文字 */
251 :         lcdPrintf( "sensor=%02x bar=%d",
252 :                 sensor_inp( 0xff ), startbar_get() );
253 :         break;
254 :
255 :     case 3:
256 :         /* 大カーブのエンコーダ値調整 */
257 :         i = eep_buff[EEPROM_CURVE_ENC];
258 :         if( getSwFlag(SW_1) ) {
259 :             i++;
260 :             if( i > 100 ) i = 100;
261 :         }
262 :         if( getSwFlag(SW_0) ) {
263 :             i--;
264 :             if( i < 0 ) i = 0;
265 :         }
266 :         eep_buff[EEPROM_CURVE_ENC] = i;
267 :
268 :         /* LCD処理 */
269 :         lcdPosition( 0, 0 );
270 :         /* 0123456789abcd..f 1行16文字 */
271 :         lcdPrintf( "3 curve enc = %03d", i );
272 :         /* 0..234..678...f 1行16文字 */
273 :         lcdPrintf( "%03d %03d %08ld",
274 :                 iEncoder, iEncoderMax, lEncoderTotal );
275 :         break;
276 :
277 :     case 4:
278 :         /* クロスライン検出後のエンコーダ値調整 */
279 :         i = eep_buff[EEPROM_CRANK_ENC];
280 :         if( getSwFlag(SW_1) ) {
281 :             i++;
282 :             if( i > 100 ) i = 100;
283 :         }
284 :         if( getSwFlag(SW_0) ) {
285 :             i--;
286 :             if( i < 0 ) i = 0;
287 :         }
288 :         eep_buff[EEPROM_CRANK_ENC] = i;
289 :
290 :         /* LCD処理 */
291 :         lcdPosition( 0, 0 );
292 :         /* 0123456789abcd..f 1行16文字 */
293 :         lcdPrintf( "4 crank enc = %03d", i );
294 :         /* 0..234..678...f 1行16文字 */
295 :         lcdPrintf( "%03d %03d %08ld",
296 :                 iEncoder, iEncoderMax, lEncoderTotal );
297 :         break;
298 :     }
299 :
300 :     if( cnt1 < 100 ) { /* LED点滅処理 */
301 :         led_out( 0x1 );

```

トレーニングボード 実習マニュアル(kit06 版)

```

302 :         } else if( cnt1 < 200 ) {
303 :             led_out( 0x2 );
304 :         } else {
305 :             cnt1 = 0;
306 :         }
307 :         break;

```

中略

```

515 :     case 23:
516 :         /* クロスライン後のトレース、クランク検出 */
517 :         if( iEncoder >= eep_buff[EEPROM_CRANK_ENC] ) {
518 :             i = 0;
519 :         } else {
520 :             i = 70;
521 :         }
522 :         if( sensor_inp(MASK4_4)==0xf8 ) {
523 :             /* 左クランクと判断 左クランククリア処理へ */
524 :             led_out( 0x1 );
525 :             handle( -38 );
526 :             speed( 10 ,50 );
527 :             pattern = 31;
528 :             cnt1 = 0;
529 :             break;
530 :         }
531 :         if( sensor_inp(MASK4_4)==0x1f ) {
532 :             /* 右クランクと判断 右クランククリア処理へ */
533 :             led_out( 0x2 );
534 :             handle( 38 );
535 :             speed( 50 ,10 );
536 :             pattern = 41;
537 :             cnt1 = 0;
538 :             break;
539 :         }
540 :         switch( sensor_inp(MASK3_3) ) {
541 :             case 0x00:
542 :                 /* センタ まっすぐ */
543 :                 handle( 0 );
544 :                 speed2( i ,i );
545 :                 break;
546 :             case 0x04:
547 :             case 0x06:
548 :             case 0x07:
549 :             case 0x03:
550 :                 /* 左寄り 右曲げ */
551 :                 handle( 8 );
552 :                 speed2( i ,diff(i) );
553 :                 break;
554 :             case 0x20:
555 :             case 0x60:
556 :             case 0xe0:
557 :             case 0xc0:
558 :                 /* 右寄り 左曲げ */
559 :                 handle( -8 );
560 :                 speed2( diff(i) ,i );
561 :                 break;
562 :         }
563 :         break;

```

中略

```

617 :     case 53:
618 :         /* 右ハーフライン後のトレース、レーンチェンジ */
619 :         if( sensor_inp(MASK4_4) == 0x00 ) {
620 :             handle( 15 );
621 :             speed( 40 ,diff(40) );
622 :             pattern = 54;
623 :             cnt1 = 0;
624 :             break;
625 :         }
626 :         if( iEncoder >= eep_buff[EEPROM_CRANK_ENC] ) {
627 :             i = 0;
628 :         } else {
629 :             i = 70;
630 :         }
631 :         switch( sensor_inp(MASK3_3) ) {
632 :             case 0x00:
633 :                 /* センタ まっすぐ */
634 :                 handle( 0 );
635 :                 speed2( i ,i );
636 :                 break;
637 :             case 0x04:
638 :             case 0x06:
639 :             case 0x07:
640 :             case 0x03:
641 :                 /* 左寄り 右曲げ */
642 :                 handle( 8 );
643 :                 speed2( i ,diff(i) );
644 :                 break;
645 :             case 0x20:
646 :             case 0x60:

```

## トレーニングボード 実習マニュアル(kit06 版)

```
647 :         case 0xe0:
648 :         case 0xc0:
649 :             /* 右寄り 左曲げ */
650 :             handle( -8 );
651 :             speed2( diff(i) , i );
652 :             break;
653 :         default:
654 :             break;
655 :     }
656 :     break;
```

中略

```
685 :     case 63:
686 :         /* 左ハーフライン後のトレース、レーンチェンジ */
687 :         if( sensor_inp(MASK4_4) == 0x00 ) {
688 :             handle( -15 );
689 :             speed( diff(40) , 40 );
690 :             pattern = 64;
691 :             cnt1 = 0;
692 :             break;
693 :         }
694 :         if( iEncoder >= eep_buff[EEPROM_CRANK_ENC] ) {
695 :             i = 0;
696 :         } else {
697 :             i = 70;
698 :         }
699 :         switch( sensor_inp(MASK3_3) ) {
700 :             case 0x00:
701 :                 /* センタ まっすぐ */
702 :                 handle( 0 );
703 :                 speed2( i , i );
704 :                 break;
705 :             case 0x04:
706 :             case 0x06:
707 :             case 0x07:
708 :             case 0x03:
709 :                 /* 左寄り 右曲げ */
710 :                 handle( 8 );
711 :                 speed2( i , diff(i) );
712 :                 break;
713 :             case 0x20:
714 :             case 0x60:
715 :             case 0xe0:
716 :             case 0xc0:
717 :                 /* 右寄り 左曲げ */
718 :                 handle( -8 );
719 :                 speed2( diff(i) , i );
720 :                 break;
721 :             default:
722 :                 break;
723 :         }
724 :     break;
```

以下、略



## 18.4 プログラムの解説

### 18.4.1 EEP-ROM エリアの変更

```

47 : /* EEP-ROM 関連 */
48 : #define EEP_ROM_SIZE 16 /* EEP-ROM 使用サイズ */
49 :
50 : #define EEPROM_CHECK 0x00 /* EEP-ROM チェック */
51 : #define EEPROM_SERVO 0x01 /* サーボセンタ値 */
52 : #define EEPROM_PWM 0x02 /* PWM 値 */
53 : #define EEPROM_CURVE_ENC 0x03 /* 大カーブのエンコーダ値 */
54 : #define EEPROM_CRANK_ENC 0x04 /* クランク部分のエンコーダ値*

```

今まで 0x04 番地は、「EEPROM\_CRANK\_PWM」としてクランク部分の PWM 値を直接設定していました。今回はクロスライン、右ハーフライン、左ハーフライン検出後の速度をエンコーダで制御します。そこで、0x04 番地を EEPROM\_CRANK\_ENC と変更して定義します。

番地	番地名	内容	初期値
0x00	EEPROM_CHECK	EEP-ROM チェック用 0x2006 で無ければ、初期値を読み込む	0x2006
0x01	EEPROM_SERVO	サーボセンタ値	5000
0x02	EEPROM_PWM	PWM 値	50
0x03	EEPROM_CURVE_ENC	大カーブでのエンコーダ値	10
<b>0x04</b>	<b>EEPROM_CRANK_ENC</b>	<b>クロスライン検出後のエンコーダ値(変更)</b>	<b>10</b>
0x05 ~ 0x0f	未定義		

### 18.4.2 EEP-ROM から読み込み

```

124 : /*EEP-ROM のチェック */
125 : if( readEeprom( EEPROM_CHECK ) != 0x2006 ) {
126 :     /*
127 :     ID のチェック EEP-ROM を初めて使うかどうか
128 :     0x00 番地に ID が書かれていなければ初めて使うと判断して初期化する
129 :     */
130 :     eep_buff[EEPROM_CHECK] = 0x2006;
131 :     eep_buff[EEPROM_SERVO] = SERVO_CENTER;
132 :     eep_buff[EEPROM_PWM] = 50;
133 :     eep_buff[EEPROM_CURVE_ENC] = 10;
134 :     eep_buff[EEPROM_CRANK_ENC] = 10;
135 : } else {
136 :     /* 2 回目以降の使用の場合、データ読み込み */
137 :     for( i=0; i<EEP_ROM_SIZE; i++ ) {
138 :         eep_buff[ i ] = readEeprom( i );
139 :     }
140 : }

```

134 行を eep\_buff[EEPROM\_CRANK\_PWM]から eep\_buff[EEPROM\_CRANK\_ENC]変数の初期化に変更します。

### 18.4.3 スイッチ入力待ち

lcd\_pattern が 4 のときの表示内容を変えます。今までは、クロスライン検出後の PWM 値でしたが、エンコーダ値へ変更します。

lcd_pattern	内容
1	サーボセンタ値の調整
2	PWM 値の調整
3	大カーブのエンコーダ値調整
4	クロスライン後のエンコーダ値の調整 (今までは PWM 値)

```

208 :      /* スイッチ、LCD処理 */
209 :      switch( lcd_pattern ) {
中略
277 :          case 4:
278 :              /* クロスライン検出後のエンコーダ値調整 */
279 :              i = eep_buff[EEPROM_CRANK_ENC];
280 :              if( getSwFlag(SW_1) ) {
281 :                  i++;
282 :                  if( i > 100 ) i = 100;
283 :              }
284 :              if( getSwFlag(SW_0) ) {
285 :                  i--;
286 :                  if( i < 0 ) i = 0;
287 :              }
288 :              eep_buff[EEPROM_CRANK_ENC] = i;
289 :
290 :              /* LCD 処理 */
291 :              lcdPosition( 0, 0 );
292 :              /* 0123456789abcd..f 1行16文字 */
293 :              lcdPrintf( "4 crank enc =%03d", i );
294 :              /* 0..234..678...f 1行16文字 */
295 :              lcdPrintf( "%03d %03d %08ld",
296 :                          iEncoder, iEncoderMax, lEncoderTotal );
297 :              break;
298 :          }

```

209 行で、lcd\_pattern の番号によりプログラムをジャンプします。4 なら 277 行へ飛びます。

279 行で、eep\_buff[EEPROM\_CRANK\_ENC]変数の値をいったん i 変数に代入します。

280 行で、SW\_1 が押されているかチェックします。押されていれば、i 変数を1つ増加させます。

284 行で、SW\_0 が押されているかチェックします。押されていれば、i 変数を1つ減少させます。

288 行で、i 変数の値を eep\_buff[EEPROM\_CRANK\_ENC]変数へ代入します。SW\_1、SW\_2 で操作された値が代入されます。

291 行から LCD にクロスライン後のエンコーダ値を表示します。2 行目が余っているので現在のエンコーダ状態を表示させています。

## 18.4.4 パターン 23 エンコーダでスピード制御するように変更

```

515 :     case 23:
516 :         /* クロスライン後のトレース、クランク検出 */
517 :         if( iEncoder >= eep_buff[EEPROM_CRANK_ENC] ) {
518 :             i = 0;
519 :         } else {
520 :             i = 70;
521 :         }
522 :         if( sensor_inp(MASK4_4)==0xf8 ) {
523 :             /* 左クランクと判断 左クランククリア処理へ */
524 :             led_out( 0x1 );
525 :             handle( -38 );
526 :             speed( 10 ,50 );
527 :             pattern = 31;
528 :             cnt1 = 0;
529 :             break;
530 :         }
531 :         if( sensor_inp(MASK4_4)==0x1f ) {
532 :             /* 右クランクと判断 右クランククリア処理へ */
533 :             led_out( 0x2 );
534 :             handle( 38 );
535 :             speed( 50 ,10 );
536 :             pattern = 41;
537 :             cnt1 = 0;
538 :             break;
539 :         }
540 :         switch( sensor_inp(MASK3_3) ) {
541 :             case 0x00:
542 :                 /* センタ まっすぐ */
543 :                 handle( 0 );
544 :                 speed2( i ,i );
545 :                 break;
546 :             case 0x04:
547 :             case 0x06:
548 :             case 0x07:
549 :             case 0x03:
550 :                 /* 左寄り 右曲げ */
551 :                 handle( 8 );
552 :                 speed2( i ,diff(i) );
553 :                 break;
554 :             case 0x20:
555 :             case 0x60:
556 :             case 0xe0:
557 :             case 0xc0:
558 :                 /* 右寄り 左曲げ */
559 :                 handle( -8 );
560 :                 speed2( diff(i) ,i );
561 :                 break;
562 :         }
563 :         break;

```

517 行で現在のエンコーダ値 (iEncoder 変数) とエンコーダの設定値 (eep\_buff[EEPROM\_CRANK\_ENC]) を比較します。現在のエンコーダ値の方が大きければ 518 行で i を 0 に、小さければ 520 行で i を 70 にしています。それ以降のプログラムで i の値を speed 関数の引数としていますので、動作としては「設定値以上なら PWM 0%、設定

値以下なら PWM 70%」にしています。

今回使用しているエンコーダは、「1m/s で進んだときの 10ms 間にカウントするパルス数は 9.75 パルス」でした。すなわち 10 を設定すれば、1m/s 以上なら PWM 値は 0%、1m/s 以下なら PWM 値は 70%ということになります。

speed 関数は、エンコーダ値でスピードを制御しますので、設定した PWM 値に関係されない speed2 関数を使用します。

#### 18.4.5 パターン 53 右レーンチェンジの徐行部分をエンコーダでスピード制御するように変更

パターン 23 と同様に変更します。右ハーフライン後、中心線が無くなるまでのスピードをエンコーダ値に応じて制御します。

#### 18.4.6 パターン 63 左レーンチェンジの徐行部分をエンコーダでスピード制御するように変更

パターン 23 と同様に変更します。左ハーフライン後、中心線が無くなるまでのスピードをエンコーダ値に応じて制御します。

### 18.5 エンコーダの回転数が違う場合の変更点

このサンプルプログラムは、100 パルス / 回転、エンコーダのタイヤ直径 33mm のエンコーダを使用した場合です。条件が違うとき、プログラムを変更しなければいけない部分を下記に示します。

















行番号	元の数値	変更後の数値
509	97	10cm 進んだときのパルス数を入れます。 例)16 パルス / 回転、直径 21mm なら 24
611	97	10cm 進んだときのパルス数を入れます。 例)16 パルス / 回転、直径 21mm なら 24
679	97	10cm 進んだときのパルス数を入れます。 例)16 パルス / 回転、直径 21mm なら 24

## 19. プロジェクト「tr\_24」 クロスライン検出後の速度設定その2 (エンコーダ使用)

### 19.1 内容

マイコンカーが高速の場合、「tr\_23.c」の減速処理ではクランクを曲がることのできるスピードまで落ちきらない場合があります。そこでクロスライン後の速度制御を2段階にします。ある速度以上なら逆転ブレーキ、それよりも遅ければブレーキ、それ以下ならモータを回すようにします。

### 19.2 プロジェクトの構成

 <b>tr_24</b>	· tr_24tart.src
 Assembly source file	ベクタアドレスの設定、スタートアップルーチンが記述されています。
 tr_24start.src	· car_printf2.c
 C source file	セクションの初期化、printf 文、scanf 文を実行します。
 beep.c	· lcd2.c
 car_printf2.c	LCD 制御を行います。
 eeprom.c	· switch.c
 lcd2.c	スイッチ制御を行います。
 switch.c	· beep.c
 tr_24.c	ブザー制御を行います。
 Dependencies	· eeprom.c
 beep.h	EEP-ROM 制御を行います。
 eeprom.h	· tr_24.c
 h8_3048.h	メインプログラムです。
 lcd2.h	
 switch.h	

### 19.3 プログラム「tr\_24.c」

プログラムのゴシック体部分が追加、変更した部分です。

前略

```

47 : /* EEP-ROM関連 */
48 : #define EEP_ROM_SIZE 16 /* EEP-ROM使用サイズ */
49 :
50 : #define EEPROM_CHECK 0x00 /* EEP-ROMチェック */
51 : #define EEPROM_SERVO 0x01 /* サーボセンタ値 */
52 : #define EEPROM_PWM 0x02 /* PWM値 */
53 : #define EEPROM_CURVE_ENC 0x03 /* 大カーブのエンコーダ値 */
54 : #define EEPROM_CRANK_ENC 0x04 /* クランク部分のエンコーダ値*/
55 : #define EEPROM_CRANK2_ENC 0x05 /* クランク部分のエンコーダ値2*/

```

中略

```

109 : /*****
110 : /* メインプログラム */
111 : *****/
112 : void main( void )
113 : {
114 :     int i, j;
115 :     int lcd_pattern = 2;
116 :
117 :     /* マイコン機能の初期化 */
118 :     init(); /* 初期化 */
119 :     set_ccr( 0x00 ); /* 全体割り込み許可 */
120 :     initLcd(); /* LCD初期化 */
121 :     initSwitch(); /* スイッチ初期化 */
122 :     initBeep(); /* ブザー初期化 */
123 :     initEeprom(); /* EEP-ROM初期化 */
124 :
125 :     /*EEP-ROMのチェック */
126 :     if( readEeprom( EEPROM_CHECK ) != 0x2006 ) {

```

トレーニングボード 実習マニュアル(kit06 版)

```

127 :      /*
128 :      IDのチェック EEP-ROMを初めて使うかどうか
129 :      0x00番地にIDが書かれていなければ初めて使うと判断して初期化する
130 :      */
131 :      eep_buff[EEPROM_CHECK] = 0x2006;
132 :      eep_buff[EEPROM_SERVO] = SERVO_CENTER;
133 :      eep_buff[EEPROM_PWM] = 50;
134 :      eep_buff[EEPROM_CURVE_ENC] = 10;
135 :      eep_buff[EEPROM_CRANK_ENC] = 10;
136 :      eep_buff[EEPROM_CRANK2_ENC] = 15;
137 :    } else {
138 :      /* 2回目以降の使用の場合、データ読み込み */
139 :      for( i=0; i<EEP_ROM_SIZE; i++ ) {
140 :        eep_buff[ i ] = readEeprom( i );
141 :      }
142 :    }
143 :
144 :    /* マイコンカーの状態初期化 */
145 :    handle( 0 );
146 :    speed( 0, 0 );
147 :
148 :    while( 1 ) {
149 :      switch( pattern ) {
150 :
151 :      /******
152 :      パターンについて
153 :      0 : スイッチ入力待ち
154 :      1 : スタートバーが開いたかチェック
155 :      11 : 通常トレース
156 :      12 : 右へ大曲げの終わりのチェック
157 :      13 : 左へ大曲げの終わりのチェック
158 :      21 : 1本目のクロスライン検出時の処理
159 :      22 : 2本目を読み飛ばす
160 :      23 : クロスライン後のトレース、クランク検出
161 :      31 : 左クランククリア処理 安定するまで少し待つ
162 :      32 : 左クランククリア処理 曲げ終わりのチェック
163 :      41 : 右クランククリア処理 安定するまで少し待つ
164 :      42 : 右クランククリア処理 曲げ終わりのチェック
165 :      51 : 1本目の右ハーフライン検出時の処理
166 :      52 : 2本目を読み飛ばす
167 :      53 : 右ハーフライン後のトレース
168 :      54 : 右レーンチェンジ終了のチェック
169 :      61 : 1本目の左ハーフライン検出時の処理
170 :      62 : 2本目を読み飛ばす
171 :      63 : 左ハーフライン後のトレース
172 :      64 : 左レーンチェンジ終了のチェック
173 :      *****/
174 :
175 :      case 0:
176 :        /* スイッチ入力待ち */
177 :        if( pushsw_get() ) {
178 :          setBeepPattern( 0xc000 );
179 :          /* 保存 */
180 :          for( i=0; i<EEP_ROM_SIZE; i++ ) {
181 :            writeEeprom( i, eep_buff[ i ] );
182 :            while( !checkEeprom() ); /* 書き込み終了チェック */
183 :          }
184 :          pattern = 1;
185 :          cnt1 = 0;
186 :          break;
187 :        }
188 :
189 :        /* スイッチ4 設定値保存 */
190 :        if( getSwFlag(SW_4) ) {
191 :          setBeepPattern( 0x8000 );
192 :          /* 保存 */
193 :          for( i=0; i<EEP_ROM_SIZE; i++ ) {
194 :            writeEeprom( i, eep_buff[ i ] );
195 :            while( !checkEeprom() ); /* 書き込み終了チェック */
196 :          }
197 :          break;
198 :        }
199 :        /* スイッチ3 メニュー + 1 */
200 :        if( getSwFlag(SW_3) ) {
201 :          lcd_pattern++;
202 :          if( lcd_pattern == 6 ) lcd_pattern = 1;
203 :        }
204 :        /* スイッチ2 メニュー - 1 */
205 :        if( getSwFlag(SW_2) ) {
206 :          lcd_pattern--;
207 :          if( lcd_pattern == 0 ) lcd_pattern = 5;
208 :        }
209 :
210 :        /* スイッチ、LCD処理 */
211 :        switch( lcd_pattern ) {
中略
301 :      case 5:
302 :        /* クロスライン検出後のエンコーダ値調整 その2 */
303 :        i = eep_buff[EEPROM_CRANK2_ENC];
304 :        if( getSwFlag(SW_1) ) {
305 :          i++;

```

トレーニングボード 実習マニュアル(kit06 版)

```

306 :         if( i > 100 ) i = 100;
307 :     }
308 :     if( getSwFlag(SW_0) ) {
309 :         i--;
310 :         if( i < 0 ) i = 0;
311 :     }
312 :     eep_buff[EEPROM_CRANK2_ENC] = i;
313 :
314 :     /* LCD処理 */
315 :     lcdPosition( 0, 0 );
316 :     /* 0123456789abcd..f 1行16文字 */
317 :     lcdPrintf( "5 crank enc2=%03d", i );
318 :     /* 0..234..678...f 1行16文字 */
319 :     lcdPrintf( "%03d %03d %08id",
320 :               iEncoder, iEncoderMax, IEncoderTotal );
321 :     break;
322 : }
323 :
324 : if( cnt1 < 100 ) {           /* LED点滅処理          */
325 :     led_out( 0x1 );
326 : } else if( cnt1 < 200 ) {
327 :     led_out( 0x2 );
328 : } else {
329 :     cnt1 = 0;
330 : }
331 : break;

```

中略

```

539 : case 23:
540 :     /* クロスライン後のトレース、クランク検出 */
541 :     if( iEncoder >= eep_buff[EEPROM_CRANK2_ENC] ) {
542 :         i = -70;
543 :     } else if( iEncoder >= eep_buff[EEPROM_CRANK_ENC] ) {
544 :         i = 0;
545 :     } else {
546 :         i = 70;
547 :     }
548 :     if( sensor_inp(MASK4_4)==0xf8 ) {
549 :         /* 左クランクと判断 左クランククリア処理へ */
550 :         led_out( 0x1 );
551 :         handle( -38 );
552 :         speed( 10 ,50 );
553 :         pattern = 31;
554 :         cnt1 = 0;
555 :         break;
556 :     }
557 :     if( sensor_inp(MASK4_4)==0x1f ) {
558 :         /* 右クランクと判断 右クランククリア処理へ */
559 :         led_out( 0x2 );
560 :         handle( 38 );
561 :         speed( 50 ,10 );
562 :         pattern = 41;
563 :         cnt1 = 0;
564 :         break;
565 :     }
566 :     switch( sensor_inp(MASK3_3) ) {
567 :     case 0x00:
568 :         /* センタ まっすぐ */
569 :         handle( 0 );
570 :         speed2( i ,i );
571 :         break;
572 :     case 0x04:
573 :     case 0x06:
574 :     case 0x07:
575 :     case 0x03:
576 :         /* 左寄り 右曲げ */
577 :         handle( 8 );
578 :         speed2( i ,diff(i) );
579 :         break;
580 :     case 0x20:
581 :     case 0x60:
582 :     case 0xe0:
583 :     case 0xc0:
584 :         /* 右寄り 左曲げ */
585 :         handle( -8 );
586 :         speed2( diff(i) ,i );
587 :         break;
588 :     }
589 :     break;

```

中略

```

643 : case 53:
644 :     /* 右ハーフライン後のトレース、レーンチェンジ */
645 :     if( sensor_inp(MASK4_4) == 0x00 ) {
646 :         handle( 15 );
647 :         speed( 40 ,diff(40) );
648 :         pattern = 54;
649 :         cnt1 = 0;
650 :         break;

```

```

651 :     }
652 :     if( iEncoder >= eep_buff[EEPROM_CRANK2_ENC] ) {
653 :         i = -70;
654 :     } else if( iEncoder >= eep_buff[EEPROM_CRANK_ENC] ) {
655 :         i = 0;
656 :     } else {
657 :         i = 70;
658 :     }
659 :     switch( sensor_inp(MASK3_3) ) {
660 :         case 0x00:
661 :             /* センタ まっすぐ */
662 :             handle( 0 );
663 :             speed2( i , i );
664 :             break;
665 :         case 0x04:
666 :         case 0x06:
667 :         case 0x07:
668 :         case 0x03:
669 :             /* 左寄り 右曲げ */
670 :             handle( 8 );
671 :             speed2( i ,diff(i) );
672 :             break;
673 :         case 0x20:
674 :         case 0x60:
675 :         case 0xe0:
676 :         case 0xc0:
677 :             /* 右寄り 左曲げ */
678 :             handle( -8 );
679 :             speed2( diff(i) , i );
680 :             break;
681 :         default:
682 :             break;
683 :     }
684 :     break;

```

中略

```

713 :     case 63:
714 :         /* 左ハーフライン後のトレース、レーンチェンジ */
715 :         if( sensor_inp(MASK4_4) == 0x00 ) {
716 :             handle( -15 );
717 :             speed( diff(40) ,40 );
718 :             pattern = 64;
719 :             cnt1 = 0;
720 :             break;
721 :         }
722 :         if( iEncoder >= eep_buff[EEPROM_CRANK2_ENC] ) {
723 :             i = -70;
724 :         } else if( iEncoder >= eep_buff[EEPROM_CRANK_ENC] ) {
725 :             i = 0;
726 :         } else {
727 :             i = 70;
728 :         }
729 :         switch( sensor_inp(MASK3_3) ) {
730 :             case 0x00:
731 :                 /* センタ まっすぐ */
732 :                 handle( 0 );
733 :                 speed2( i , i );
734 :                 break;
735 :             case 0x04:
736 :             case 0x06:
737 :             case 0x07:
738 :             case 0x03:
739 :                 /* 左寄り 右曲げ */
740 :                 handle( 8 );
741 :                 speed2( i ,diff(i) );
742 :                 break;
743 :             case 0x20:
744 :             case 0x60:
745 :             case 0xe0:
746 :             case 0xc0:
747 :                 /* 右寄り 左曲げ */
748 :                 handle( -8 );
749 :                 speed2( diff(i) , i );
750 :                 break;
751 :             default:
752 :                 break;
753 :         }
754 :         break;

```

以下、略



## 19.4 プログラムの解説

### 19.4.1 EEP-ROM エリアの追加

```

47 : /* EEP-ROM 関連 */
48 : #define EEP_ROM_SIZE 16 /* EEP-ROM 使用サイズ */
49 :
50 : #define EEPROM_CHECK 0x00 /* EEP-ROM チェック */
51 : #define EEPROM_SERVO 0x01 /* サーボセンタ値 */
52 : #define EEPROM_PWM 0x02 /* PWM 値 */
53 : #define EEPROM_CURVE_ENC 0x03 /* 大カーブのエンコーダ値 */
54 : #define EEPROM_CRANK_ENC 0x04 /* クランク部分のエンコーダ値 */
55 : #define EEPROM_CRANK2_ENC 0x05 /* クランク部分のエンコーダ値 2 */
    
```

クロスライン検出後のエンコーダ値2を設定する EEP-ROM の番地を決めます。0x04 まで使っていたので、0x05 番地を EEPROM\_CRANK2\_ENC として定義します。

番地	番地名	内容	初期値
0x00	EEPROM_CHECK	EEP-ROM チェック用 0x2006 で無ければ、初期値を読み込む	0x2006
0x01	EEPROM_SERVO	サーボセンタ値	5000
0x02	EEPROM_PWM	PWM 値	50
0x03	EEPROM_CURVE_ENC	大カーブでのエンコーダ値	10
0x04	EEPROM_CRANK_ENC	クロスライン検出後のエンコーダ値	10
<b>0x05</b>	<b>EEPROM_CRANK2_ENC</b>	<b>クロスライン検出後のエンコーダ値 2</b>	<b>15</b>
0x06 ~ 0x0f	未定義		

### 19.4.2 EEP-ROM から読み込み

```

125 : /*EEP-ROM のチェック */
126 : if( readEeprom( EEPROM_CHECK ) != 0x2006 ) {
127 :     /*
128 :     ID のチェック EEP-ROM を初めて使うかどうか
129 :     0x00 番地に ID が書かれていなければ初めて使うと判断して初期化する
130 :     */
131 :     eep_buff[EEPROM_CHECK] = 0x2006;
132 :     eep_buff[EEPROM_SERVO] = SERVO_CENTER;
133 :     eep_buff[EEPROM_PWM] = 50;
134 :     eep_buff[EEPROM_CURVE_ENC] = 10;
135 :     eep_buff[EEPROM_CRANK_ENC] = 10;
136 :     eep_buff[EEPROM_CRANK2_ENC] = 15;
137 : } else {
138 :     /* 2 回目以降の使用の場合、データ読み込み */
139 :     for( i=0; i<EEP_ROM_SIZE; i++ ) {
140 :         eep_buff[ i ] = readEeprom( i );
141 :     }
142 : }
    
```

136 行に eep\_buff[EEPROM\_CRANK2\_ENC]変数の初期化を追加しています。

### 19.4.3 スイッチ入力待ち

```

199 :      /* スイッチ 3 メニュー + 1 */
200 :      if( getSwFlag(SW_3) ) {
201 :          lcd_pattern++;
202 :          if( lcd_pattern == 6 ) lcd_pattern = 1;
203 :      }
204 :      /* スイッチ 2 メニュー - 1 */
205 :      if( getSwFlag(SW_2) ) {
206 :          lcd_pattern--;
207 :          if( lcd_pattern == 0 ) lcd_pattern = 5;
208 :      }
    
```

SW\_3 で LCD に表示する内容を次へ進めます。SW\_2 で戻します。lcd\_pattern=5 をクロスライン後のエンコーダ値2の調整として追加します。

202 行で上限のチェックです。lcd\_pattern の 6 はありませんので 1 にします。

207 行で下限のチェックです。lcd\_pattern の 0 はありませんので 5 にします。

lcd_pattern	内容
1	サーボセンタ値の調整
2	PWM 値の調整
3	大カーブのエンコーダ値調整
4	クロスライン後のエンコーダ値の調整
5	<b>クロスライン後のエンコーダ値の調整 2</b>

```

210 :      /* スイッチ、LCD 処理 */
211 :      switch( lcd_pattern ) {
中略
301 :          case 5:
302 :              /* クロスライン検出後のエンコーダ値調整 その 2 */
303 :              i = eep_buff[EEPROM_CRANK2_ENC];
304 :              if( getSwFlag(SW_1) ) {
305 :                  i++;
306 :                  if( i > 100 ) i = 100;
307 :              }
308 :              if( getSwFlag(SW_0) ) {
309 :                  i--;
310 :                  if( i < 0 ) i = 0;
311 :              }
312 :              eep_buff[EEPROM_CRANK2_ENC] = i;
313 :
314 :              /* LCD 処理 */
315 :              lcdPosition( 0, 0 );
316 :              /* 0123456789abcd..f 1 行 16 文字 */
317 :              lcdPrintf( "5 crank enc2=%03d", i );
318 :              /* 0..234..678...f 1 行 16 文字 */
319 :              lcdPrintf( "%03d %03d %08ld",
320 :                          iEncoder, iEncoderMax, lEncoderTotal );
321 :              break;
322 :          }
    
```

211 行で、lcd\_pattern の番号によりプログラムをジャンプします。5 なら 301 行へ飛びます。

303 行で、eep\_buff[EEPROM\_CRANK2\_ENC]変数の値をいったん i 変数に代入します。  
304 行で、SW\_1 が押されているかチェックします。押されていれば、i 変数を1つ増加させます。  
308 行で、SW\_0 が押されているかチェックします。押されていれば、i 変数を1つ減少させます。  
312 行で、i 変数の値を eep\_buff[EEPROM\_CRANK2\_ENC]変数へ代入します。SW\_1、SW\_2 で操作された値が代入されます。  
315 行から LCD にクロスライン検出後のエンコーダ値 2 を表示します。2 行目が余っているので現在のエンコーダ状態を表示させています。

**調整の注意点としては必ず、**

```
eep_buff[EEPROM_CRANK2_ENC] > eep_buff[EEPROM_CRANK_ENC]
```

とします。

#### 19.4.4 パターン 23 エンコーダ値により 2 段階でスピード制御するように変更

```
539 :      case 23:
540 :          /* クロスライン後のトレース、クランク検出 */
541 :          if( iEncoder >= eep_buff[EEPROM_CRANK2_ENC] ) {
542 :              i = -70;
543 :          } else if( iEncoder >= eep_buff[EEPROM_CRANK_ENC] ) {
544 :              i = 0;
545 :          } else {
546 :              i = 70;
547 :          }
以下略
```

541 ~ 547 行でスピードをチェックして、PWM値を決めています。

・iEncoder >= eep_buff[EEPROM_CRANK2_ENC]なら、PWM-70%	高速ならバックブレーキ
・iEncoder >= eep_buff[EEPROM_CRANK_ENC]なら、PWM0%	中速ならブレーキ
・それ以外なら、PWM70%	低速なら進める

とします。

#### 19.4.5 パターン 53 右レーンチェンジの徐行部分をエンコーダ値により 2 段階でスピード制御するように変更

パターン 23 と同様に変更します。右ハーフライン後、中心線が無くなるまでのスピードをエンコーダ値に応じて制御します。

#### 19.4.6 パターン 63 左レーンチェンジの徐行部分をエンコーダ値により 2 段階でスピード制御するように変更

パターン 23 と同様に変更します。左ハーフライン後、中心線が無くなるまでのスピードをエンコーダ値に応じて制御します。

## 19.5 エンコーダの回転数が違う場合の変更点

このサンプルプログラムは、100 パルス / 回転、エンコーダのタイヤ直径 33mm のエンコーダを使用した場合です。条件が違うとき、プログラムを変更しなければいけない部分を下記に示します。


行番号	元の数値	変更後の数値
533	97	10cm 進んだときのパルス数を入れます。 例)16 パルス / 回転、直径 21mm なら <b>24</b>
637	97	10cm 進んだときのパルス数を入れます。 例)16 パルス / 回転、直径 21mm なら <b>24</b>
707	97	10cm 進んだときのパルス数を入れます。 例)16 パルス / 回転、直径 21mm なら <b>24</b>

## 20. プロジェクト「tr\_25」 設定距離でマイコンカーを止める(エンコーダ使用)

### 20.1 内容

マイコンカーのスピードが速いと、コースを走るマイコンカーを止めるのは至難の業です。そこで、停止距離を設定できるようにして、設定した距離を走ったら止まるようにします。

### 20.2 プロジェクトの構成

 tr_25	·tr_25start.src	ベクタアドレスの設定、スタートアップルーチンが記述されています。
Assembly source file	·car_printf2.c	セクションの初期化、printf 文、scanf 文を実行します。
tr_25start.src	·lcd2.c	LCD 制御を行います。
C source file	·switch.c	スイッチ制御を行います。
beep.c	·beep.c	ブザー制御を行います。
car_printf2.c	·eeprom.c	EEP-ROM 制御を行います。
eeprom.c	·tr_25.c	メインプログラムです。
lcd2.c		
switch.c		
tr_25.c		
Dependencies		
beep.h		
eeprom.h		
h8_3048.h		
lcd2.h		
switch.h		

### 20.3 プログラム「tr\_25.c」

プログラムのゴシック体部分が追加した部分です。

前略

```

47 : /* EEP-ROM関連 */
48 : #define      EEP_ROM_SIZE      16      /* EEP-ROM使用サイズ      */
49 :
50 : #define      EEPROM_CHECK      0x00    /* EEP-ROMチェック      */
51 : #define      EEPROM_SERVO      0x01    /* サーボセンタ値      */
52 : #define      EEPROM_PWM        0x02    /* PWM値      */
53 : #define      EEPROM_CURVE_ENC  0x03    /* 大カーブのエンコーダ値 */
54 : #define      EEPROM_CRANK_ENC  0x04    /* クランク部分のエンコーダ値 */
55 : #define      EEPROM_CRANK2_ENC 0x05    /* クランク部分のエンコーダ値2 */
56 : #define      EEPROM_STOP_DISTANCE 0x06 /* 止まる距離      */

```

中略

```

110 : /****** */
111 : /* メインプログラム */
112 : /****** */
113 : void main( void )
114 : {
115 :     int    i, j;
116 :     int    lcd_pattern = 2;
117 :
118 :     /* マイコン機能の初期化 */
119 :     init(); /* 初期化 */
120 :     set_ccr( 0x00 ); /* 全体割り込み許可 */
121 :     initLcd(); /* LCD初期化 */
122 :     initSwitch(); /* スイッチ初期化 */
123 :     initBeep(); /* ブザー初期化 */
124 :     initEeprom(); /* EEP-ROM初期化 */
125 :
126 :     /*EEP-ROMのチェック */
127 :     if( readEeprom( EEPROM_CHECK ) != 0x2006 ) {
128 :         /*

```

## トレーニングボード 実習マニュアル(kit06 版)

```

129 :         IDのチェック EEPROMを初めて使うかどうか
130 :         0x00番地にIDが書かれていなければ初めて使うと判断して初期化する
131 :         */
132 :         eep_buff[EEPROM_CHECK]           = 0x2006;
133 :         eep_buff[EEPROM_SERVO]           = SERVO_CENTER;
134 :         eep_buff[EEPROM_PWM]             = 50;
135 :         eep_buff[EEPROM_CURVE_ENC]       = 10;
136 :         eep_buff[EEPROM_CRANK_ENC]       = 10;
137 :         eep_buff[EEPROM_CRANK2_ENC]      = 15;
138 :         eep_buff[EEPROM_STOP_DISTANCE]   = 100;
139 :     } else {
140 :         /* 2回目以降の使用の場合、データ読み込み */
141 :         for( i=0; i<EEP_ROM_SIZE; i++ ) {
142 :             eep_buff[ i ] = readEeprom( i );
143 :         }
144 :     }
145 :
146 :     /* マイコンカーの状態初期化 */
147 :     handle( 0 );
148 :     speed( 0, 0 );
149 :
150 :     while( 1 ) {
151 :         switch( pattern ) {
152 :
153 :             /******
154 :             パターンについて
155 :             0 : スイッチ入力待ち
156 :             1 : スタートバーが開いたかチェック
157 :             11 : 通常トレース
158 :             12 : 右へ大曲げの終わりのチェック
159 :             13 : 左へ大曲げの終わりのチェック
160 :             21 : 1本目のクロスライン検出時の処理
161 :             22 : 2本目を読み飛ばす
162 :             23 : クロスライン後のトレース、クランク検出
163 :             31 : 左クランククリア処理 安定するまで少し待つ
164 :             32 : 左クランククリア処理 曲げ終わりのチェック
165 :             41 : 右クランククリア処理 安定するまで少し待つ
166 :             42 : 右クランククリア処理 曲げ終わりのチェック
167 :             51 : 1本目の右ハーフライン検出時の処理
168 :             52 : 2本目を読み飛ばす
169 :             53 : 右ハーフライン後のトレース
170 :             54 : 右レーンチェンジ終了のチェック
171 :             61 : 1本目の左ハーフライン検出時の処理
172 :             62 : 2本目を読み飛ばす
173 :             63 : 左ハーフライン後のトレース
174 :             64 : 左レーンチェンジ終了のチェック
175 :             *****/
176 :
177 :         case 0:
178 :             /* スイッチ入力待ち */
179 :             if( pushsw_get() ) {
180 :                 setBeepPattern( 0xc000 );
181 :                 /* 保存 */
182 :                 for( i=0; i<EEP_ROM_SIZE; i++ ) {
183 :                     writeEeprom( i, eep_buff[ i ] );
184 :                     while( !checkEeprom() ); /* 書き込み終了チェック */
185 :                 }
186 :                 pattern = 1;
187 :                 cnt1 = 0;
188 :                 break;
189 :             }
190 :
191 :             /* スイッチ4 設定値保存 */
192 :             if( getSwFlag(SW_4) ) {
193 :                 setBeepPattern( 0x8000 );
194 :                 /* 保存 */
195 :                 for( i=0; i<EEP_ROM_SIZE; i++ ) {
196 :                     writeEeprom( i, eep_buff[ i ] );
197 :                     while( !checkEeprom() ); /* 書き込み終了チェック */
198 :                 }
199 :                 break;
200 :             }
201 :             /* スイッチ3 メニュー + 1 */
202 :             if( getSwFlag(SW_3) ) {
203 :                 lcd_pattern++;
204 :                 if( lcd_pattern == 7 ) lcd_pattern = 1;
205 :             }
206 :             /* スイッチ2 メニュー - 1 */
207 :             if( getSwFlag(SW_2) ) {
208 :                 lcd_pattern--;
209 :                 if( lcd_pattern == 0 ) lcd_pattern = 6;
210 :             }
211 :
212 :             /* スイッチ、LCD処理 */
213 :             switch( lcd_pattern ) {

```

中略

```

325 :         case 6:
326 :             /* 止める距離 */
327 :             i = eep_buff[EEPROM_STOP_DISTANCE];
328 :             if( getSwFlag(SW_1) ) {
329 :                 i++;
330 :                 if( i > 100 ) i = 100;
331 :             }
332 :             if( getSwFlag(SW_0) ) {
333 :                 i--;
334 :                 if( i < 0 ) i = 0;
335 :             }
336 :             eep_buff[EEPROM_STOP_DISTANCE] = i;
337 :
338 :             /* LCD処理 */
339 :             lcdPosition( 0, 0 );
340 :             /* 0123456789abc..ef 1行16文字 */
341 :             lcdPrintf( "6 stop dis =%03dm", i );
342 :             /* 0..234..678...f 1行16文字 */
343 :             lcdPrintf( "%03d %03d %08ld",
344 :                 iEncoder, iEncoderMax, lEncoderTotal );
345 :             break;
346 :         }
347 :
348 :         if( cnt1 < 100 ) { /* LED点滅処理 */
349 :             led_out( 0x1 );
350 :         } else if( cnt1 < 200 ) {
351 :             led_out( 0x2 );
352 :         } else {
353 :             cnt1 = 0;
354 :         }
355 :         break;

```

中略

```

379 :         case 11:
380 :             /* 通常トレース */
381 :
382 :             /* 走行距離の計算 設定値以上なら停止 */
383 :             if( lEncoderTotal >=
384 :                 (long)eep_buff[EEPROM_STOP_DISTANCE] * 965 ) {
385 :                 speed( 0, 0 );
386 :                 pattern = 0;
387 :                 setBeepPattern( 0xcc0 );
388 :                 break;
389 :             }

```

以下、略

## 20.4 プログラムの解説

### 20.4.1 EEPROM-ROM エリアの追加

```

47 : /* EEPROM-ROM 関連 */
48 : #define    EEPROM_ROM_SIZE    16    /* EEPROM-ROM 使用サイズ */
49 :
50 : #define    EEPROM_CHECK    0x00    /* EEPROM-ROM チェック */
51 : #define    EEPROM_SERVO    0x01    /* サーボセンタ値 */
52 : #define    EEPROM_PWM    0x02    /* PWM 値 */
53 : #define    EEPROM_CURVE_ENC    0x03    /* 大カーブのエンコーダ値 */
54 : #define    EEPROM_CRANK_ENC    0x04    /* クランク部分のエンコーダ値 */
55 : #define    EEPROM_CRANK2_ENC    0x05    /* クランク部分のエンコーダ値 2 */
56 : #define    EEPROM_STOP_DISTANCE    0x06    /* 停止距離 */

```

停止距離を設定する EEPROM-ROM の番地を決めます。0x05 番地まで使っていたので、0x06 番地を EEPROM\_STOP\_DISTANCE として定義します。

番地	番地名	内容	初期値
0x00	EEPROM_CHECK	EEP-ROM チェック用 0x2006 で無ければ、初期値を読み込む	0x2006
0x01	EEPROM_SERVO	サーボセンタ値	5000
0x02	EEPROM_PWM	PWM 値	50
0x03	EEPROM_CURVE_ENC	大カーブでのエンコーダ値	10
0x04	EEPROM_CRANK_ENC	クロスライン検出後のエンコーダ値	10
0x05	EEPROM_CRANK2_ENC	クロスライン検出後のエンコーダ値 2	15
<b>0x06</b>	<b>EEPROM_STOP_DISTANCE</b>	<b>停止距離</b>	<b>100</b>
0x07 ~ 0x0f	未定義		

### 20.4.2 EEPROM-ROM から読み込み

```

126 : /*EEP-ROM のチェック */
127 : if( readEeprom( EEPROM_CHECK ) != 0x2006 ) {
128 :     /*
129 :     ID のチェック EEPROM-ROM を初めて使うかどうか
130 :     0x00 番地に ID が書かれていなければ初めて使うと判断して初期化する
131 :     */
132 :     eep_buff[EEPROM_CHECK]    = 0x2006;
133 :     eep_buff[EEPROM_SERVO]    = SERVO_CENTER;
134 :     eep_buff[EEPROM_PWM]      = 50;
135 :     eep_buff[EEPROM_CURVE_ENC] = 10;
136 :     eep_buff[EEPROM_CRANK_ENC] = 10;
137 :     eep_buff[EEPROM_CRANK2_ENC] = 15;
138 :     eep_buff[EEPROM_STOP_DISTANCE] = 100;
139 : } else {
140 :     /* 2 回目以降の使用の場合、データ読み込み */
141 :     for( i=0; i<EEP_ROM_SIZE; i++ ) {
142 :         eep_buff[ i ] = readEeprom( i );
143 :     }
144 : }

```



138 行に eep\_buff[EEPROM\_STOP\_DISTANCE]変数の初期化を追加しています。

### 20.4.3 スイッチ入力待ち

```

201 :      /* スイッチ 3 メニュー + 1 */
202 :      if( getSwFlag(SW_3) ) {
203 :          lcd_pattern++;
204 :          if( lcd_pattern == 7 ) lcd_pattern = 1;
205 :      }
206 :      /* スイッチ 2 メニュー - 1 */
207 :      if( getSwFlag(SW_2) ) {
208 :          lcd_pattern--;
209 :          if( lcd_pattern == 0 ) lcd_pattern = 6;
210 :      }

```

SW\_3 で LCD に表示する内容を次に進めます。SW\_2 で戻します。lcd\_pattern=6 を停止距離の設定として追加します。

204 行で上限のチェックです。lcd\_pattern の 7 はありませんので 1 にします。

209 行で下限のチェックです。lcd\_pattern の 0 はありませんので 6 にします。

lcd_pattern	内容
1	サーボセンタ値の調整
2	PWM 値の調整
3	大カーブの PWM 値の調整
4	クロスライン後のエンコーダ値の調整
5	クロスライン後のエンコーダ値の調整 2
6	<b>停止距離の設定</b>

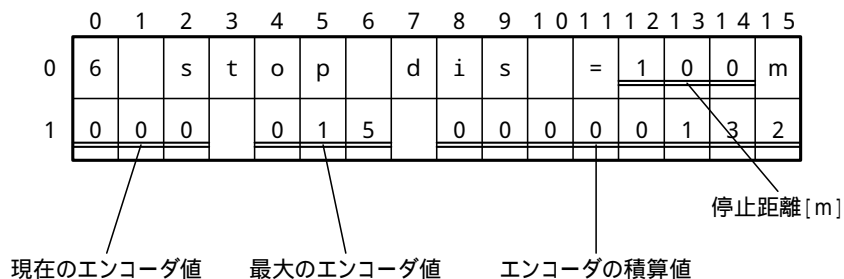
```

212 :      /* スイッチ、LCD処理 */
213 :      switch( lcd_pattern ) {
中略
325 :      case 6:
326 :          /* 停止距離 */
327 :          i = eep_buff[EEPROM_STOP_DISTANCE];
328 :          if( getSwFlag(SW_1) ) {
329 :              i++;
330 :              if( i > 100 ) i = 100;
331 :          }
332 :          if( getSwFlag(SW_0) ) {
333 :              i--;
334 :              if( i < 0 ) i = 0;
335 :          }
336 :          eep_buff[EEPROM_STOP_DISTANCE] = i;
337 :
338 :          /* LCD処理 */
339 :          lcdPosition( 0, 0 );
340 :          /* 0123456789abc..ef 1行16文字 */
341 :          lcdPrintf( "6 stop dis =%03dm", i );
342 :          /* 0..234..678...f 1行16文字 */
343 :          lcdPrintf( "%03d %03d %08ld",
344 :                    iEncoder, iEncoderMax, lEncoderTotal );
345 :          break;
346 :      }

```

213行で、lcd\_patternの番号によりプログラムをジャンプします。6なら325行へ飛びます。  
 327行で、eep\_buff[EEPROM\_STOP\_DISTANCE]変数の値をいったんi変数に代入します。  
 328行で、SW\_1が押されているかチェックします。押されていれば、i変数を1つ増加させます。  
 332行で、SW\_0が押されているかチェックします。押されていれば、i変数を1つ減少させます。  
 336行で、i変数の値をeep\_buff[EEPROM\_STOP\_DISTANCE]変数へ代入します。SW\_1、SW\_2で操作された値が代入されます。  
 339行からLCDに停止距離を表示します。2行目が余っているので現在のエンコーダ状態を表示させています。

lcd\_patternが6のとき、下記のように表示されます。



SW\_1とSW\_0で走行距離(単位:メートル)を増減させます。最初は100mになっています。

## 20.4.4 パターン 11 走行距離以上走ったかチェック

```

379 :     case 11:
380 :         /* 通常トレース */
381 :
382 :         /* 走行距離の計算 設定値以上なら停止 */
383 :         if( lEncoderTotal >=
384 :             (long)eep_buff[EEPROM_STOP_DISTANCE] * 965 ) {
385 :             speed( 0 ,0 );
386 :             pattern = 0;
387 :             setBeepPattern( 0xccc0 );
388 :             break;
389 :         }

```

以下、略

383、384 行で、走行距離をチェックしています。次のような意味です。

$$lEncoderTotal \geq eep\_buff[EEPROM\_STOP\_DISTANCE] * 965$$

現在の積算値  $\geq$  止まるまでの設定距離[m]                    \* 1m 進んだときのパルス値

現在の積算パルス数(距離)  $\geq$  止まるまでの設定距離[m]                    \* 1m 進んだときのパルス値

となり、eep\_buff[EEPROM\_STOP\_DISTANCE]メートル以上走ったかどうかのチェックです。  
eep\_buff[EEPROM\_STOP\_DISTANCE]メートル以上走ると、386 行でパターン 0 へ移行し、停止します。  
ちなみに、**計算結果が int 型を越える恐れがあるのでいったん long 型に変換して計算しています。**  
例えば、eep\_buff[EEPROM\_STOP\_DISTANCE]変数の値が 34 なら  
 $34 \times 965 = 32,810$   
で、int 型を超えてしまい値が不定となり、きちんと動作しません。

## 20.5 エンコーダの回転数が違う場合の変更点

このサンプルプログラムは、100パルス/回転、エンコーダのタイヤ直径33mmのエンコーダを使用した場合です。条件が違うとき、プログラムを変更しなければいけない部分を下記に示します。

行番号	元の数値	変更後の数値
384	965	100cm 進んだときのパルス数を入れます。 例)16 パルス/回転、直径 21mm なら <b>242</b>
567	97	10cm 進んだときのパルス数を入れます。 例)16 パルス/回転、直径 21mm なら <b>24</b>
671	97	10cm 進んだときのパルス数を入れます。 例)16 パルス/回転、直径 21mm なら <b>24</b>
741	97	10cm 進んだときのパルス数を入れます。 例)16 パルス/回転、直径 21mm なら <b>24</b>

















## 21. プロジェクト「tr\_31」 走行データの記録(内蔵 RAM 使用)

### 21.1 内容

ほとんどの場合、マイコンカーの走っている状態を目で見て、ここはもっと速くできる、ここはぎりぎりだな、などと判断していると思います。

そこで、走行データを RAM に保存、パソコンへ転送して、走行状態を数値で確認してみましょう。さらに、エクセルなどのソフトに取り込んでグラフ化してみましょう。取得したデータを元にプログラムを解析すれば、理論的にプログラムの解析、変更をすることができます。

### 21.2 プロジェクトの構成

 tr_31	· tr_31tart.src
 Assembly source file	ベクタアドレスの設定、スタートアップルーチンが記述されています。
 tr_31start.src	· car_printf2.c
 C source file	セクションの初期化、printf 文、scanf 文を実行します。
 beep.c	· lcd2.c
 car_printf2.c	LCD 制御を行います。
 eeprom.c	· switch.c
 lcd2.c	スイッチ制御を行います。
 switch.c	· beep.c
 tr_31.c	ブザー制御を行います。
 Dependencies	· eeprom.c
 beep.h	EEP-ROM 制御を行います。
 eeprom.h	· tr_31.c
 h8_3048.h	メインプログラムです。
 lcd2.h	
 switch.h	

### 21.3 プログラム「tr\_31.c」

プログラムのゴシック体部分が追加した部分です。

```

1 :  /*-----*/
2 :  /* トレーニングボードを使用したマイコンカートレースプログラム(kit06版) */
3 :  /*                               2006.08 ジャパンマイコンカーラリー実行委員会 */
4 :  /*-----*/
5 :
6 :  /*-----*/
7 :  /* インクルード */
8 :  /*-----*/
9 :  #include <no_float.h> /* stdioの簡略化 最初に置く*/
10 : #include <stdio.h>
11 : #include <machine.h>
12 : #include "h8_3048.h"
13 : #include "lcd2.h" /* LCD表示用追加 */
14 : #include "switch.h" /* スイッチ追加 */
15 : #include "beep.h" /* ブザー追加 */
16 : #include "eeprom.h" /* EEP-ROM追加 */
17 :
18 : /*-----*/
19 : /* シンボル定義 */
20 : /*-----*/
21 :
22 : /* 定数設定 */
23 : #define TIMER_CYCLE 3071 /* タイマのサイクル 1ms */
24 : /* /8で使用する場合、 */
25 : /* /8 = 325.5[ns] */
26 : /* TIMER_CYCLE = */
27 : /* 1[ms] / 325.5[ns] */
28 : /* = 3072 */
29 : #define PWM_CYCLE 24575 /* PWMのサイクル 8msに変更 */

```

トレーニングボード 実習マニュアル(kit06 版)

```

30 :                                     /* PWM_CYCLE =          */
31 :                                     /*      8[ms] / 325.5[ns] */
32 :                                     /*                      = 24576 */
33 : #define SERVO_CENTER 5000          /* サーボのセンタ値      */
34 : #define HANDLE_STEP 26            /* 1° 分の値              */
35 :
36 : /* マスク値設定 × : マスクあり(無効) : マスク無し(有効) */
37 : #define MASK2_2 0x66              /* × × × × × × × × × × */
38 : #define MASK2_0 0x60              /* × × × × × × × × × × */
39 : #define MASK0_2 0x06              /* × × × × × × × × × × */
40 : #define MASK3_3 0xe7              /* × × × × × × × × × × */
41 : #define MASK0_3 0x07              /* × × × × × × × × × × */
42 : #define MASK3_0 0xe0              /* × × × × × × × × × × */
43 : #define MASK4_0 0xf0              /* × × × × × × × × × × */
44 : #define MASK0_4 0x0f              /* × × × × × × × × × × */
45 : #define MASK4_4 0xff              /* × × × × × × × × × × */
46 :
47 : /* EEP-ROM関連 */
48 : #define EEP_ROM_SIZE 16          /* EEP-ROM使用サイズ    */
49 :
50 : #define EEPROM_CHECK 0x00        /* EEP-ROMチェック      */
51 : #define EEPROM_SERVO 0x01        /* サーボセンタ値      */
52 : #define EEPROM_PWM 0x02          /* PWM値                 */
53 : #define EEPROM_CURVE_ENC 0x03    /* 大カーブのエンコーダ値 */
54 : #define EEPROM_CRANK_ENC 0x04    /* クランク部分のエンコーダ値 */
55 : #define EEPROM_CRANK2_ENC 0x05   /* クランク部分のエンコーダ値2 */
56 : #define EEPROM_STOP_DISTANCE 0x06 /* 停止距離              */
57 :
58 : /* 保存データ関連 */
59 : #define SAVE_SIZE 800            /* データ保存数          */
60 :
61 : /*=====*/
62 : /* プロトタイプ宣言 */
63 : /*=====*/
64 : void init( void );
65 : void timer( unsigned long timer_set );
66 : int check_crossline( void );
67 : int check_rightline( void );
68 : int check_leftline( void );
69 : unsigned char sensor_inp( unsigned char mask );
70 : unsigned char dipsw_get( void );
71 : unsigned char pushsw_get( void );
72 : unsigned char startbar_get( void );
73 : void led_out( unsigned char led );
74 : void speed( int accele_l, int accele_r );
75 : void speed2( int accele_l, int accele_r );
76 : void handle( int angle );
77 : char unsigned bit_change( char unsigned in );
78 : int diff( int pwm );
79 :
80 : /*=====*/
81 : /* グローバル変数の宣言 */
82 : /*=====*/
83 : unsigned long cnt0;              /* timer関数用          */
84 : unsigned long cnt1;              /* main内で使用         */
85 : int pattern;                    /* パターン番号          */
86 :
87 : /* EEP-ROM設定 */
88 : int eep_buff[ EEP_ROM_SIZE ];
89 :
90 : /* 角度関連 */
91 : int angle_buff;                 /* 現在ハンドル角度保持用 */
92 :
93 : const revolution_difference[] = { /* 角度から内輪、外輪回転差計算 */
94 :     100, 99, 97, 96, 95,
95 :     93, 92, 91, 89, 88,
96 :     87, 86, 84, 83, 82,
97 :     81, 79, 78, 77, 76,
98 :     75, 73, 72, 71, 70,
99 :     69, 67, 66, 65, 64,
100 :     62, 61, 60, 59, 58,
101 :     56, 55, 54, 52, 51,
102 :     50, 48, 47, 46, 44,
103 :     43 };
104 :
105 : /* 割り込み内関連 */
106 : int iTimer10;                   /* 取得間隔計算用      */
107 :
108 : /* エンコーダ関連 */
109 : long lEncoderTotal;             /* 積算値                */
110 : int iEncoderMax;                /* 現在最大値            */
111 : int iEncoder;                   /* 現在値                */
112 : unsigned int uEncoderBuff;      /* 前回値保存            */
113 : long lEncoderLine;              /* ライン検出時の積算値 */
114 :
115 : /* データ保存関連 データは2.5KB以内にしてください */
116 : unsigned char saveData[SAVE_SIZE][3];
117 : int saveIndex;                  /* 保存インデックス      */
118 : int saveSendIndex;              /* 送信インデックス      */
119 : int saveFlag;                   /* 保存フラグ            */
120 :

```

## トレーニングボード 実習マニュアル(kit06 版)

```

121 : /*****
122 : /* メインプログラム */
123 : /*****
124 : void main( void )
125 : {
126 :     int    i, j;
127 :     int    lcd_pattern = 2;
128 :
129 :     /* マイコン機能の初期化 */
130 :     init();
131 :     init_sci1( 0x00, 79 );
132 :     set_ccr( 0x00 );
133 :     initLcd();
134 :     initSwitch();
135 :     initBeep();
136 :     initEeprom();
137 :
138 :     /*EEP-ROMのチェック */
139 :     if( readEeprom( EEPROM_CHECK ) != 0x2006 ) {
140 :         /*
141 :         IDのチェック EEPROMを初めて使うかどうか
142 :         0x00番地にIDが書かれていなければ初めて使うと判断して初期化する
143 :         */
144 :         eep_buff[EEPROM_CHECK]          = 0x2006;
145 :         eep_buff[EEPROM_SERVO]          = SERVO_CENTER;
146 :         eep_buff[EEPROM_PWM]            = 50;
147 :         eep_buff[EEPROM_CURVE_ENC]      = 10;
148 :         eep_buff[EEPROM_CRANK_ENC]      = 10;
149 :         eep_buff[EEPROM_CRANK2_ENC]     = 15;
150 :         eep_buff[EEPROM_STOP_DISTANCE]  = 100;
151 :     } else {
152 :         /* 2回目以降の使用の場合、データ読み込み */
153 :         for( i=0; i<EEP_ROM_SIZE; i++ ) {
154 :             eep_buff[ i ] = readEeprom( i );
155 :         }
156 :     }
157 :
158 :     /* マイコンカーの状態初期化 */
159 :     handle( 0 );
160 :     speed( 0, 0 );
161 :
162 :     while( 1 ) {
163 :         switch( pattern ) {
164 :
165 :         /*****
166 :         パターンについて
167 :         0: スイッチ入力待ち
168 :         1: スタートバーが開いたかチェック
169 :         11: 通常トレース
170 :         12: 右へ大曲げの終わりのチェック
171 :         13: 左へ大曲げの終わりのチェック
172 :         21: 1本目のクロスライン検出時の処理
173 :         22: 2本目を読み飛ばす
174 :         23: クロスライン後のトレース、クランク検出
175 :         31: 左クランククリア処理 安定するまで少し待つ
176 :         32: 左クランククリア処理 曲げ終わりのチェック
177 :         41: 右クランククリア処理 安定するまで少し待つ
178 :         42: 右クランククリア処理 曲げ終わりのチェック
179 :         51: 1本目の右ハーフライン検出時の処理
180 :         52: 2本目を読み飛ばす
181 :         53: 右ハーフライン後のトレース
182 :         54: 右レーンチェンジ終了のチェック
183 :         61: 1本目の左ハーフライン検出時の処理
184 :         62: 2本目を読み飛ばす
185 :         63: 左ハーフライン後のトレース
186 :         64: 左レーンチェンジ終了のチェック
187 :         *****/
188 :
189 :         case 0:
190 :             /* スイッチ入力待ち */
191 :             if( pushsw_get() ) {
192 :                 setBeepPattern( 0xc000 );
193 :                 /* 保存 */
194 :                 for( i=0; i<EEP_ROM_SIZE; i++ ) {
195 :                     writeEeprom( i, eep_buff[ i ] );
196 :                     while( !checkEeprom() ); /* 書き込み終了チェック */
197 :                 }
198 :                 pattern = 1;
199 :                 cnt1 = 0;
200 :                 break;
201 :             }

```

中略

```

369 :         case 1:
370 :             /* スタートバーが開いたかチェック */
371 :             if( !startbar_get() ) {
372 :                 /* スタート!! */
373 :                 iTimer10 = 0;
374 :                 lEncoderTotal = 0;
375 :                 iEncoderMax = 0;

```

トレーニングボード 実習マニュアル(kit06 版)

```

376 :         iEncoder          = 0;
377 :         led_out( 0x0 );
378 :         pattern = 11;
379 :         cnt1 = 0;
380 :         saveIndex = 0;
381 :         saveFlag = 1;
382 :         break;
383 :     }
384 :     if( cnt1 < 50 ) {                /* LED点滅処理          */
385 :         led_out( 0x1 );
386 :     } else if( cnt1 < 100 ) {
387 :         led_out( 0x2 );
388 :     } else {
389 :         cnt1 = 0;
390 :     }
391 :     break;
392 :
393 : case 11:
394 :     /* 通常トレース */
395 :
396 :     if( pushsw_get() ) {
397 :         pattern = 71;                /* データ転送処理へ      */
398 :         break;
399 :     }
400 :
401 :     /* 走行距離の計算 設定値以上なら停止 */
402 :     if( lEncoderTotal >=
403 :         (long)eep_buff[EEPROM_STOP_DISTANCE] * 965 ) {
404 :         speed( 0, 0 );
405 :         pattern = 71;
406 :         setBeepPattern( 0xcc0 );
407 :         break;
408 :     }

```

中略

```

818 : case 71:
819 :     /* 停止 */
820 :     handle( 0 );
821 :     speed( 0, 0 );
822 :     saveSendIndex = 0;
823 :     saveFlag = 0;
824 :     pattern = 72;
825 :     cnt1 = 0;
826 :     break;
827 :
828 : case 72:
829 :     /* 1s待ち */
830 :     if( cnt1 > 1000 ) {
831 :         pattern = 73;
832 :         cnt1 = 0;
833 :     }
834 :     break;
835 :
836 : case 73:
837 :     /* スイッチが離されたかチェック */
838 :     if( !pushsw_get() ) {
839 :         lcdPosition( 0, 0 );
840 :         /* 0123456789abcdef 1行16文字 */
841 :         lcdPrintf( "waitting..." );
842 :         /* 0123456789abcdef 1行16文字 */
843 :         lcdPrintf( " " );
844 :
845 :         pattern = 74;
846 :         cnt1 = 0;
847 :     }
848 :     break;
849 :
850 : case 74:
851 :     /* スイッチが押されたかチェック */
852 :     led_out( (cnt1/500) % 2 + 1 );
853 :     if( pushsw_get() ) {
854 :         pattern = 75;
855 :         cnt1 = 0;
856 :     }
857 :     break;
858 :
859 : case 75:
860 :     /* 転送 */
861 :     printf( "%n" );
862 :     printf( "tr_31 Data Out%n" );
863 :     printf( "Encoder Max = %d%n", iEncoderMax );
864 :     printf( "Pattern, Sensor, Encoder%n" );
865 :     pattern = 76;
866 :     break;
867 :
868 : case 76:
869 :     /* データ転送 */
870 :
871 :     /* 終わりのチェック */
872 :     if( saveIndex <= saveSendIndex ) {

```

```

873 :         pattern = 77;
874 :         cnt1 = 0;
875 :     }
876 :
877 :     /* データの転送 */
878 :     printf( "%d,%02x,%d\n",
879 :         saveData[saveSendIndex][0], /* パターン */
880 :         saveData[saveSendIndex][1], /* センサ */
881 :         saveData[saveSendIndex][2] ); /* エンコーダ */
882 :
883 :     /* 状態表示 */
884 :     lcdPosition( 0, 0 );
885 :     /* 012345..9ab..f 1行16文字 */
886 :     lcdPrintf( "data %05d/%05d", saveSendIndex, saveIndex );
887 :
888 :     led_out( (saveSendIndex/8) % 2 + 1 ); /* LED点滅処理 */
889 :
890 :     saveSendIndex++; /* 次の準備 */
891 :     break;
892 :
893 : case 77:
894 :     /* 転送終了 */
895 :     led_out( 0x3 );
896 :     break;
897 :
898 : default:
899 :     /* どれでもない場合は待機状態に戻す */
900 :     pattern = 0;
901 :     break;
902 : }
903 : }
904 : }

```

中略

```

946 : /******
947 : /* ITU0 割り込み処理 */
948 : /******
949 : #pragma interrupt( interrupt_timer0 )
950 : void interrupt_timer0( void )
951 : {
952 :     unsigned int i;
953 :
954 :     ITU0_TSR &= 0xfe; /* フラグクリア */
955 :     cnt0++;
956 :     cnt1++;
957 :
958 :     /* LCD表示処理用関数です。1ms毎に実行します。 */
959 :     lcdShowProcess();
960 :     /* 拡張スイッチ用関数です。1ms毎に実行します。 */
961 :     switchProcess();
962 :     /* ブザー処理用関数です。1ms毎に実行します。 */
963 :     beepProcess();
964 :
965 :     /* エンコーダ関連 */
966 :     iTimer10++;
967 :     if( iTimer10 >= 10 ) {
968 :         iTimer10 = 0;
969 :         i = ITU2_CNT;
970 :         iEncoder = i - uEncoderBuff;
971 :         lEncoderTotal += iEncoder;
972 :         if( iEncoder > iEncoderMax )
973 :             iEncoderMax = iEncoder;
974 :         uEncoderBuff = i;
975 :
976 :         /* データ保存関連 */
977 :         if( saveFlag ) {
978 :             saveData[saveIndex][0] = pattern;
979 :             saveData[saveIndex][1] = sensor_inp(0xff);
980 :             saveData[saveIndex][2] = iEncoder;
981 :             saveIndex++;
982 :             if( saveIndex >= SAVE_SIZE ) saveFlag = 0;
983 :         }
984 :     }
985 : }

```

以下、略



## 21.4 プログラムの解説

### 21.4.1 保存するデータ

このプログラムでは、

- ・パターン番号
- ・センサ状態
- ・エンコーダ値

の3つの値を保存します。  
保存間隔は 10ms ごとです。

### 21.4.2 定数の追加

```
59 : #define          SAVE_SIZE      800    /* データ保存数          */
```

データを保存する個数です。今回は、800 回分保存します。保存間隔は 10ms なので、  
10ms × 800 = 8.00 秒  
8 秒間、データを保存することができます。

### 21.4.3 変数の追加

```
115 : /* データ保存関連 データは 2.5KB 以内にしてください */
116 : unsigned char  saveData[SAVE_SIZE][3];
117 : int             saveIndex;          /* 保存インデックス      */
118 : int             saveSendIndex;     /* 送信インデックス     */
119 : int             saveFlag;          /* 保存フラグ           */
```

変数名	意味	内容
saveData	データ保存用	saveData は 2 次元配列で、SAVE_SIZE 個分の値を保存します。saveData は unsigned char 型なので、メモリは 1 データ 1 バイト使用します。「saveData[SAVE_SIZE][3]」なので、1 回の保存で 3 バイト使用します。H8/3048F-ONE の RAM 領域は 4KB ありますが、すべて使えるわけではありません。関数の戻り先アドレスの保存、レジスタの値保存など、マイコン側でも使用します。そのため、最大で約 2.5KB の確保が限界です。今回は1回で 3 バイト分のメモリを使用するので、 $2500 \div 3 = 833 \quad 850$ 850 個くらいが限界です。
saveIndex	保存インデックス	配列の何番目に保存するかを指定する変数です。
saveSendIndex	保存送信インデックス	配列の何番目のデータを送信するかを指定する変数です。
saveFlag	保存フラグ	1 なら割り込み内プログラム内で 10ms ごとにデータを保存します。0 なら保存しません。

#### 21.4.4 パソコンとの通信するための初期設定

```

129 :      /* マイコン機能の初期化 */
130 :      init();                          /* 初期化 */
131 :      init_sci1( 0x00, 79 );           /* SCI1 初期化 */
132 :      set_ccr( 0x00 );                 /* 全体割り込み許可 */
133 :      initLcd();                       /* LCD 初期化 */
134 :      initSwitch();                   /* スイッチ初期化 */
135 :      initBeep();                     /* ブザー初期化 */
136 :      initEeprom();                   /* EEP-ROM 初期化 */

```

パソコンと通信するために、H8/3048F-ONE の内蔵周辺機能である SCI1 の初期化を行います。car\_printf2.c 内に、簡単に SCI1 を初期化することができる init\_sci1 関数がありますので、この関数を使って初期化します。意味合いについては、「H8/3048F-ONE 実習マニュアル」のプロジェクト「sio」の解説を参照してください。

#### 21.4.5 パターン 1 スタート

```

369 :      case 1:
370 :          /* スタートバーが開いたかチェック */
371 :          if( !startbar_get() ) {
372 :              /* スタート!! */
373 :              iTimer10      = 0;
374 :              lEncoderTotal = 0;
375 :              iEncoderMax   = 0;
376 :              iEncoder      = 0;
377 :              led_out( 0x0 );
378 :              pattern = 11;
379 :              cnt1 = 0;
380 :              saveIndex = 0;
381 :              saveFlag = 1;
382 :              break;
383 :          }
384 :          if( cnt1 < 50 ) {                /* LED 点滅処理 */
385 :              led_out( 0x1 );
386 :          } else if( cnt1 < 100 ) {
387 :              led_out( 0x2 );
388 :          } else {
389 :              cnt1 = 0;
390 :          }
391 :          break;

```

スタート時に、

- ・saveIndex を 0 にして、saveData へ保存する配列の番号を初期化しています。
- ・saveFlag を 1 にして、保存を開始します。

#### 21.4.6 パターン 11 の追加、変更 ログ転送処理へ

```

393 :     case 11:
394 :         /* 通常トレース */
395 :
396 :         if( pushsw_get() ) {
397 :             pattern = 71;           /* データ転送処理へ          */
398 :             break;
399 :         }
400 :
401 :         /* 走行距離の計算 設定値以上なら停止 */
402 :         if( !EncoderTotal >=
403 :             (long)eep_buff[EEPROM_STOP_DISTANCE] * 965 ) {
404 :             speed( 0 ,0 );
405 :             pattern = 71;
406 :             setBeepPattern( 0xcc0 );
407 :             break;
408 :         }

```

パターン 11 の通常トレース時に、スイッチが押されるとパターン 71 へ移ります (396 ~ 399 行)。また、設定距離で止まったときもパターン 71 へ移るようにします (405 行)。

#### 21.4.7 パターン 71 停止

```

818 :     case 71:
819 :         /* 停止 */
820 :         handle( 0 );
821 :         speed( 0, 0 );
822 :         saveSendIndex = 0;
823 :         saveFlag = 0;
824 :         pattern = 72;
825 :         cnt1 = 0;
826 :         break;

```

マイコンカーを停止して、saveFlag を 0 にします。走行データの保存が終わりになります。saveSendIndex を 0 にして、送信データを配列の 0 番目からにします。

#### 21.4.8 パターン 72 1秒待ち

```

828 :     case 72:
829 :         /* 1s 待ち */
830 :         if( cnt1 > 1000 ) {
831 :             pattern = 73;
832 :             cnt1 = 0;
833 :         }
834 :         break;

```

単純に1秒待ちます。走行しているマイコンカーを止めるためにスイッチを押します。このときの、2度押し防止用です。

#### 21.4.9 パターン 73 スイッチが離されたかチェック

```

836 :     case 73:
837 :         /* スイッチが離されたかチェック */
838 :         if( !pushsw_get() ) {
839 :             lcdPosition( 0, 0 );
840 :             /* 0123456789abcdef 1行 16文字 */
841 :             lcdPrintf( "waitting...      " );
842 :             /* 0123456789abcdef 1行 16文字 */
843 :             lcdPrintf( "                " );
844 :
845 :             pattern = 74;
846 :             cnt1 = 0;
847 :         }
848 :         break;

```

1秒待った後、スイッチが離されているかチェックします。マイコンカーを止めるためにスイッチを押しっぱなしにしている場合は、ここで処理を止めます。

#### 21.4.10 パターン 74 スイッチが押されたかチェック

```

850 :     case 74:
851 :         /* スイッチが押されたかチェック */
852 :         led_out( (cnt1/500) % 2 + 1 );
853 :         if( pushsw_get() ) {
854 :             pattern = 75;
855 :             cnt1 = 0;
856 :         }
857 :         break;

```

スイッチが押されると、データ転送を開始します。ここでは、通信の準備をします。ケーブルを接続して、ハイパーターミナルや Tera Term Pro などの通信ソフトの準備をします。準備ができれば、マイコンカーのスイッチを押します。

#### 21.4.11 パターン 75 タイトルの転送

```

859 :     case 75:
860 :         /* 転送 */
861 :         printf( "%n" );
862 :         printf( "tr_31 Data Out%n" );
863 :         printf( "Encoder Max = %d%n", iEncoderMax );
864 :         printf( "Pattern, Sensor, Encoder%n" );
865 :         pattern = 76;
866 :         break;

```

データを転送する前に、タイトルや、エンコーダの最大値などをパソコンへ転送します。

例えば、下記のようにパソコンへ転送されます。

tr\_31 Data Out  
Encoder Max = 26  
Pattern, Sensor, Encoder

#### 21.4.12 パターン 76 データの転送

```

868 :     case 76:
869 :         /* データ転送 */
870 :
871 :         /* 終わりのチェック */
872 :         if( saveIndex <= saveSendIndex ) {
873 :             pattern = 77;
874 :             cnt1 = 0;
875 :         }
876 :
877 :         /* データの転送 */
878 :         printf( "%d,%02x,%d\n",
879 :             saveData[saveSendIndex][0], /* パターン */
880 :             saveData[saveSendIndex][1], /* センサ */
881 :             saveData[saveSendIndex][2] ); /* エンコーダ */
882 :
883 :         /* 状態表示 */
884 :         lcdPosition( 0, 0 );
885 :             /* 012345..9ab..f 1行16文字 */
886 :         lcdPrintf( "data %05d/%05d", saveSendIndex, saveIndex );
887 :
888 :         led_out( (saveSendIndex/8) % 2 + 1 ); /* LED点滅処理 */
889 :
890 :         saveSendIndex++; /* 次の準備 */
891 :         break;

```

データを転送します。

転送後、890行で送信する配列番号を+1します。872行で、保存した番号より送信する番号が大きいかチェックして、大きいなら転送を終了します。下記は、データ送信例です。

13,c0,12	11,1c,12
13,e0,11	11,1c,11
11,60,11	11,0c,13
11,60,10	11,0c,13
11,30,11	11,0c,12
11,30,11	11,06,13
11,38,10	11,06,13
11,18,12	11,07,14
11,18,13	11,07,13
11,18,12	12,03,14
11,18,13	12,03,13

### 21.4.13 パターン 77 終了

```

893 :     case 77:
894 :         /* 転送終了 */
895 :         led_out( 0x3 );
896 :         break;

```

モータドライブ基板の LED を 2 つとも点灯させ、何もしません。終了です。

### 21.4.14 データの保存

```

946 :  /******
947 :  /* ITU0 割り込み処理
948 :  /******
949 :  #pragma interrupt( interrupt_timer0 )
950 :  void interrupt_timer0( void )
951 :  {
952 :      unsigned int i;
953 :
954 :      ITU0_TSR &= 0xfe;          /* フラグクリア          */
955 :      cnt0++;
956 :      cnt1++;
957 :
958 :      /* LCD 表示処理用関数です。1ms 毎に実行します。          */
959 :      lcdShowProcess();
960 :      /* 拡張スイッチ用関数です。1ms 毎に実行します。          */
961 :      switchProcess();
962 :      /* ブザー処理用関数です。1ms 毎に実行します。          */
963 :      beepProcess();
964 :
965 :      /* エンコーダ関連 */
966 :      iTimer10++;
967 :      if( iTimer10 >= 10 ) {
968 :          iTimer10 = 0;
969 :          i = ITU2_CNT;
970 :          iEncoder      = i - uEncoderBuff;
971 :          lEncoderTotal += iEncoder;
972 :          if( iEncoder > iEncoderMax )
973 :              iEncoderMax = iEncoder;
974 :          uEncoderBuff = i;
975 :
976 :          /* データ保存関連 */
977 :          if( saveFlag ) {
978 :              saveData[saveIndex][0] = pattern;
979 :              saveData[saveIndex][1] = sensor_inp(0xff);
980 :              saveData[saveIndex][2] = iEncoder;
981 :              saveIndex++;
982 :              if( saveIndex >= SAVE_SIZE ) saveFlag = 0;
983 :          }
984 :      }
985 :  }

```

保存は、10ms 間隔です。ロータリエンコーダ処理も 10ms 間隔なので、ロータリエンコーダ処理の中に入れて

10ms ごとに保存します。

977 行で、saveFlag をチェックして 0 以外ならデータを保存、0 なら保存しません。

978 行で、パターン変数を保存します。

979 行で、センサの状態を保存します。マスクは 0xff としてすべての情報を保存します。

980 行で、現在のエンコーダ値を保存します。スピードが分かります。

981 行で、配列へ保存する番号を + 1 します。

982 行で、保存する上限を超えているかチェックします。上限なら、saveFlag 変数を 0 にして保存を終了します。

978 ~ 980 行の保存する変数や関数を変えれば、保存する内容を変えることができます。

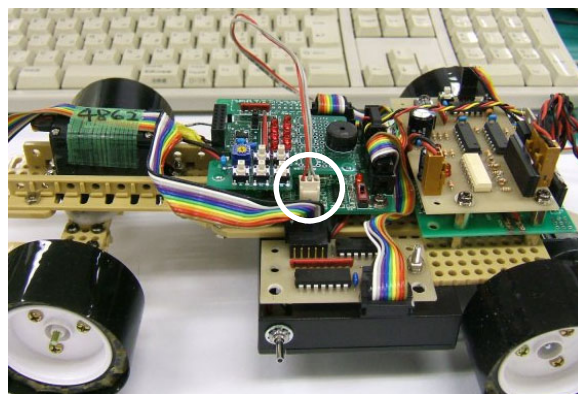
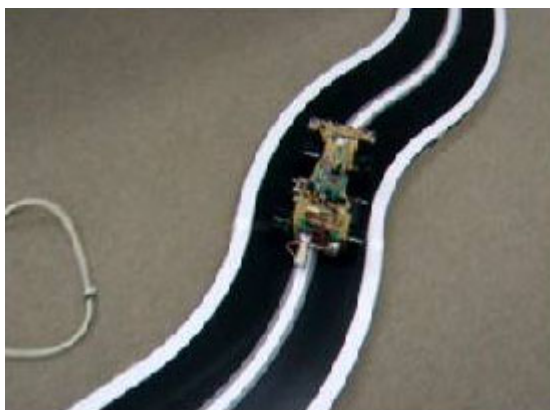
## 21.5 エンコーダの回転数が違う場合の変更点

このサンプルプログラムは、100 パルス / 回転、エンコーダのタイヤ直径 33mm のエンコーダを使用した場合です。条件が違うとき、プログラムを変更しなければいけない部分を下記に示します。

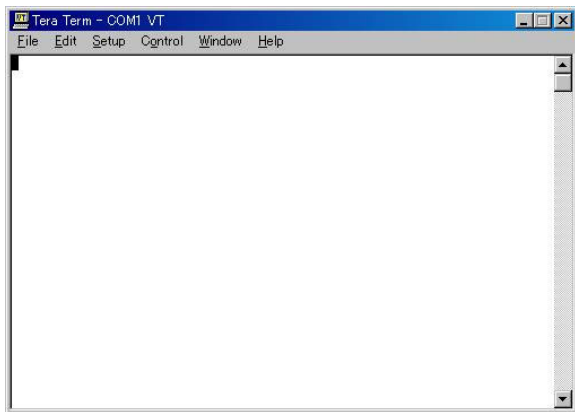
行番号	元の数値	変更後の数値
403	965	100cm 進んだときのパルス数を入れます。 例)16 パルス / 回転、直径 21mm なら 242
586	97	10cm 進んだときのパルス数を入れます。 例)16 パルス / 回転、直径 21mm なら 24
690	97	10cm 進んだときのパルス数を入れます。 例)16 パルス / 回転、直径 21mm なら 24
760	97	10cm 進んだときのパルス数を入れます。 例)16 パルス / 回転、直径 21mm なら 24

## 21.6 転送の仕方

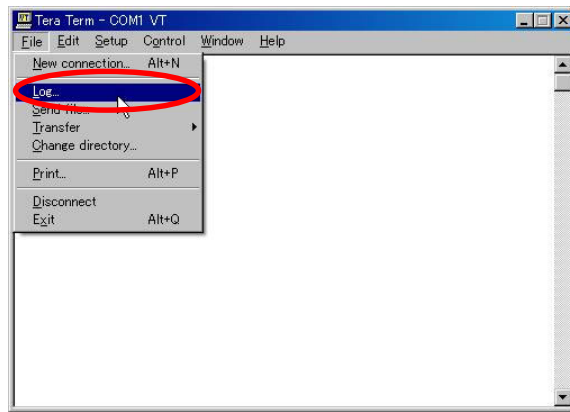
実際に転送してみましょう。



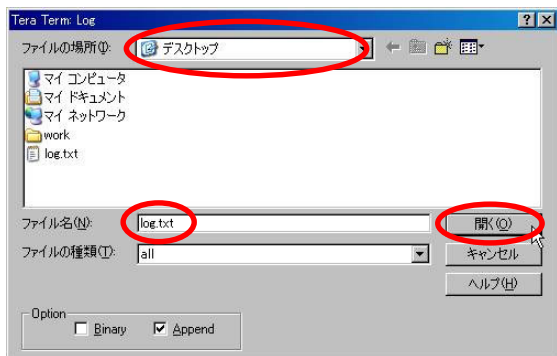
1. マイコンカーを走らせます。記録時間以上走らせた後、モータドライブ基板のプッシュスイッチを押して止めます。電源を切ると、せっかく記録したデータが消えてしまいます。
2. 通信ケーブルをマイコンカーに接続します。マイコンカーの電源は切りません。



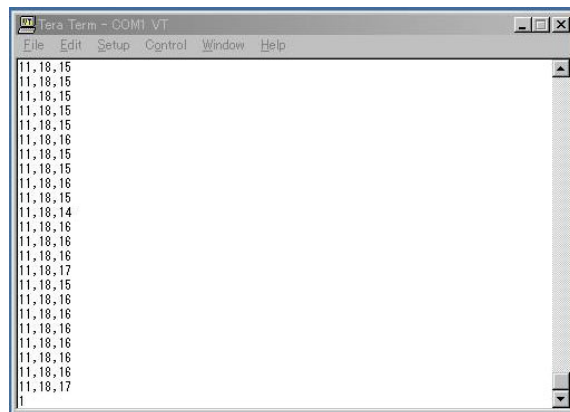
3. Tera Term Pro を立ち上げます。



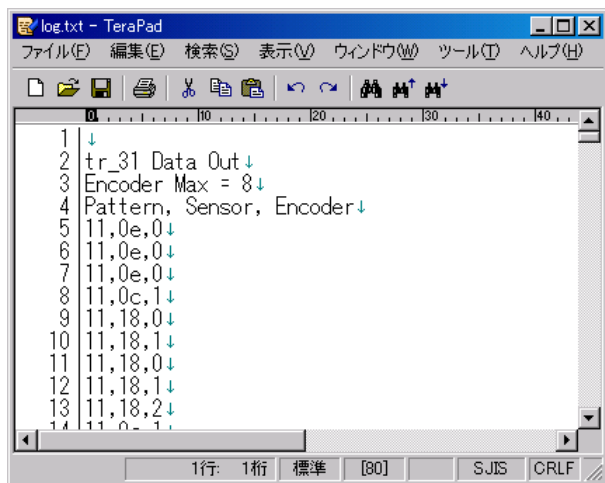
4. 「File Log」でログを保存するファイル名を選択します。



5. ここでは、「ファイルの場所: デスクトップへ保存」、「ファイル名: log.txt」とします。開くをクリックします。パソコンで受信したデータを、保存する準備ができました。

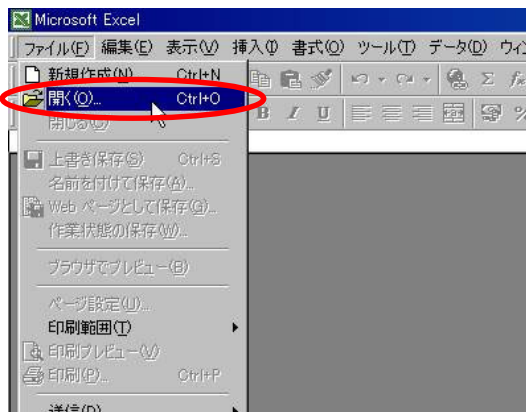


6. マイコンカーのプッシュスイッチを押します。走行データの送信が開始されます。転送が終わったら、マイコンカーの電源を切り、TeraTermProも閉じます。

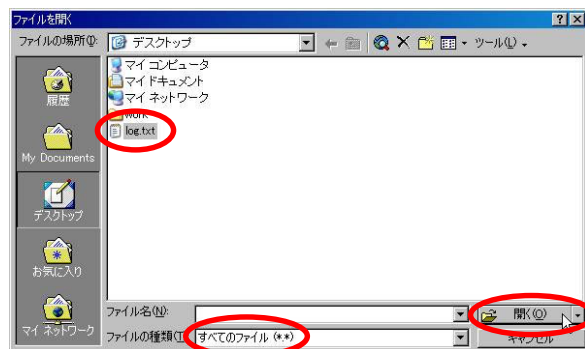


7. TeraPad 等のエディタで開くと、保存データを見ることができます。

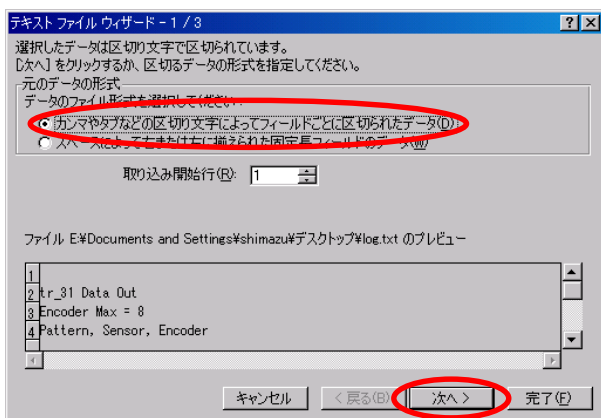




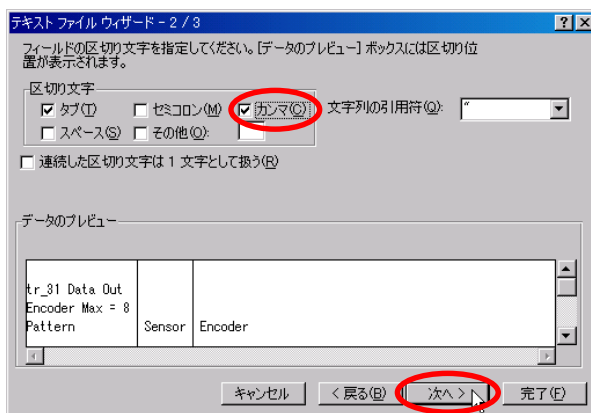
8. エクセルで開いてみます。「ファイル 開く」を選択します。



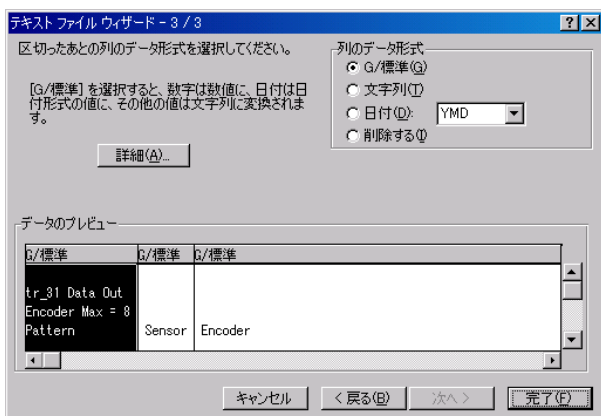
9. 「ファイルの場所: デスクトップ(保存したフォルダ)」、「ファイルの種類: すべてのファイル(\*.\*)」にします。デスクトップの log.txt を選択して開きます。



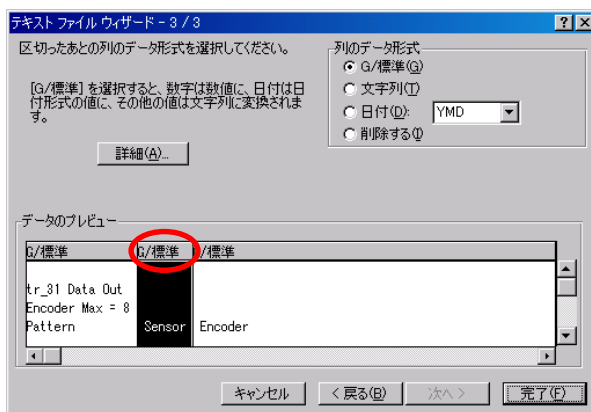
10. テキストファイルをエクセルのセルに分ける画面が出てきます。  
「カンマやタブなどの区切り文字によってフィールドごとに区切られたデータ」を選択して、次へ進みます。



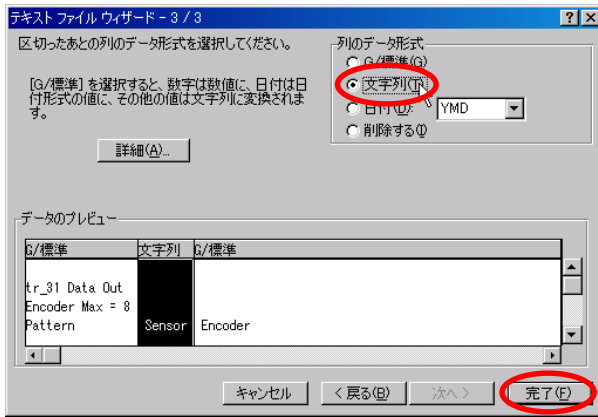
11. 「カンマ」を選択して、次へ進みます。



12. このような画面になります。



13. 左から2つ目の「G / 標準」を選択します。

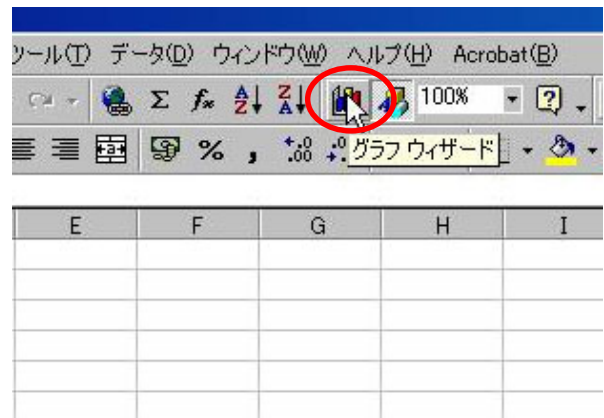


	A	B	C	D
1				
2	tr_31 Data Out			
3	Encoder Max = 8			
4	Pattern	Sensor	Encoder	
5		11 0e	0	
6		11 0e	0	
7		11 0e	0	
8		11 0c	1	
9		11 18	0	
10		11 18	1	
11		11 18	0	
12		11 18	1	
13		11 18	2	
14		11 0c	1	
15		11 0c	1	

14. 2列目は、センサ状態が16進数で記録されているので、「文字列」とします。完了を選択します。

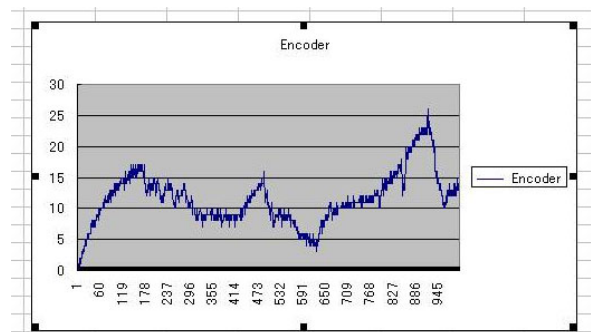
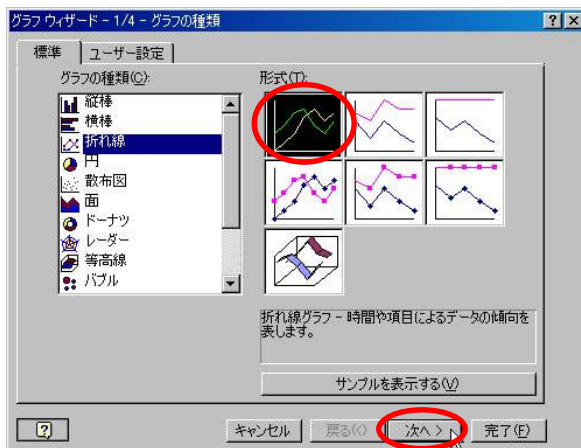
15. データがセルに取り込まれました。

	A	B	C	D
1				
2	tr_31 Data Out			
3	Encoder Max = 8			
4	Pattern	Sensor	Encoder	
5		11 0e	0	
6		11 0e	0	
7		11 0e	0	
8		11 0c	1	
9		11 18	0	
10		11 18	1	
11		11 18	0	
12		11 18	1	
13		11 18	2	
14		11 0c	1	
15		11 0c	1	



16. エンコーダ値をグラフ化してみましょう。エンコーダの数値部分を選択します。

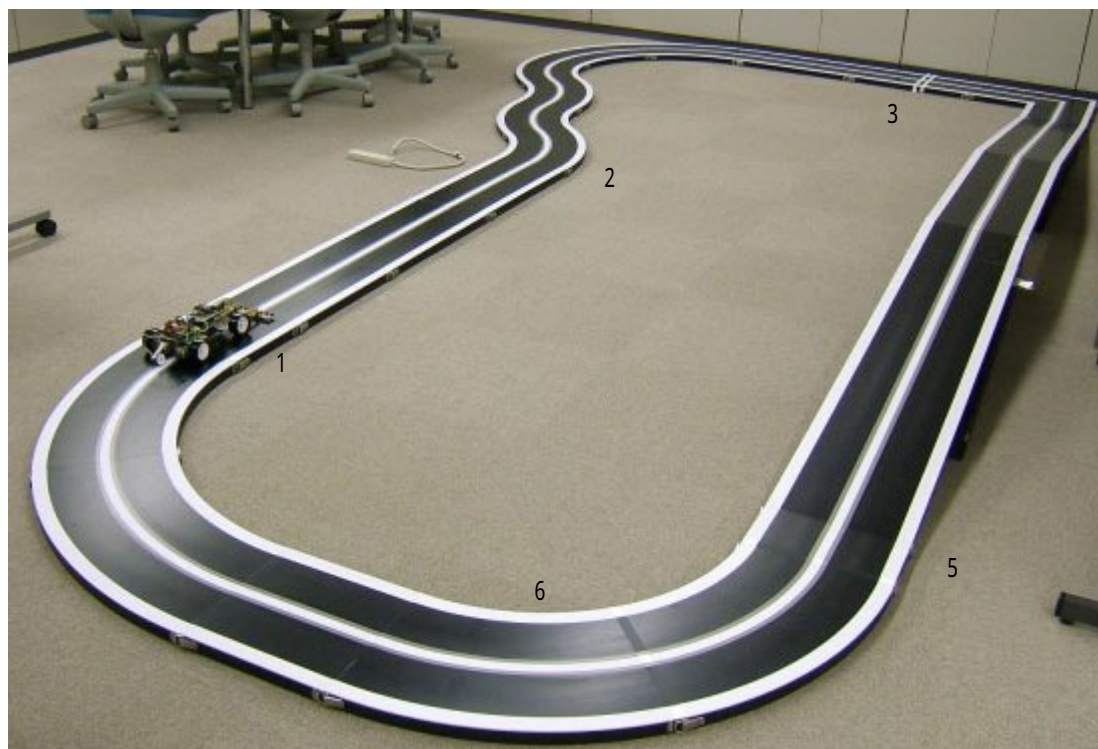
17. グラフ ウィザードを選択します。



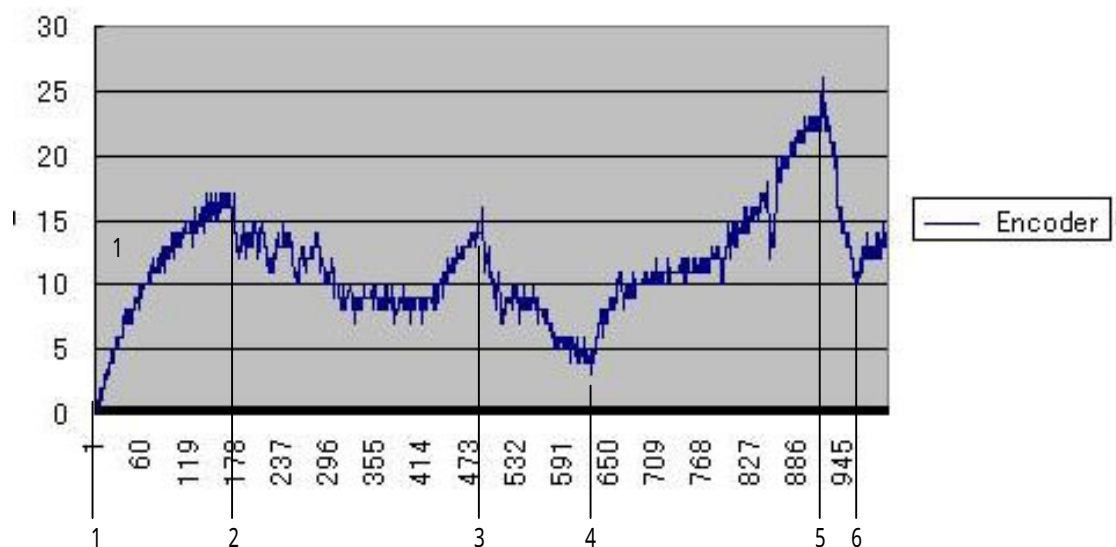
18. 折れ線グラフを選択します。

19. 後は、それぞれの項目を設定して、完了ボタンを押します。グラフ化されました。

今回のテストコースとグラフを見比べてみます。



Encoder



- |           |                   |
|-----------|-------------------|
| 1. スタート位置 | 2. S字             |
| 3. クロスライン | 4. クランク(曲げ終わりの瞬間) |
| 5. 下り坂終了  | 6. カーブ            |

このように、コースとグラフを見比べると、どこでどのくらいのスピードが出ていたかはっきりと分かります。ちなみに、4番部分のエンコーダ値は5で、約 0.5m/s のスピードでした。5番部分のスピードは下り坂の終わりで、最速の26、約 2.6m/s です。

今回は、パターン、センサ値、エンコーダ値を保存しました。いろいろな変数を保存して、自分なりに解析すると新たな発見があるかもしれません。

## 22. プロジェクト「tr\_32」 走行データの記録(外付け EEP-ROM 使用)

### 22.1 内容

プロジェクト「tr\_31」は、内蔵 RAM に走行データを保存しています。プログラムの変更だけで対応可能なため、簡単にできて良いのですが、

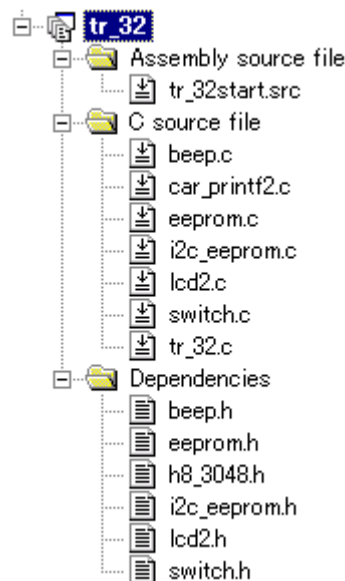
- ・保存時間が短い
- ・電源を切ると消えてしまう(RAM なので当然なのですが)

という問題があります。そこで、EEP-ROM という、電气的に書き換え可能なメモリを外付けして、そのメモリに走行データを保存するようにします。

詳しくは、「データ解析実習マニュアル」を参照してください。

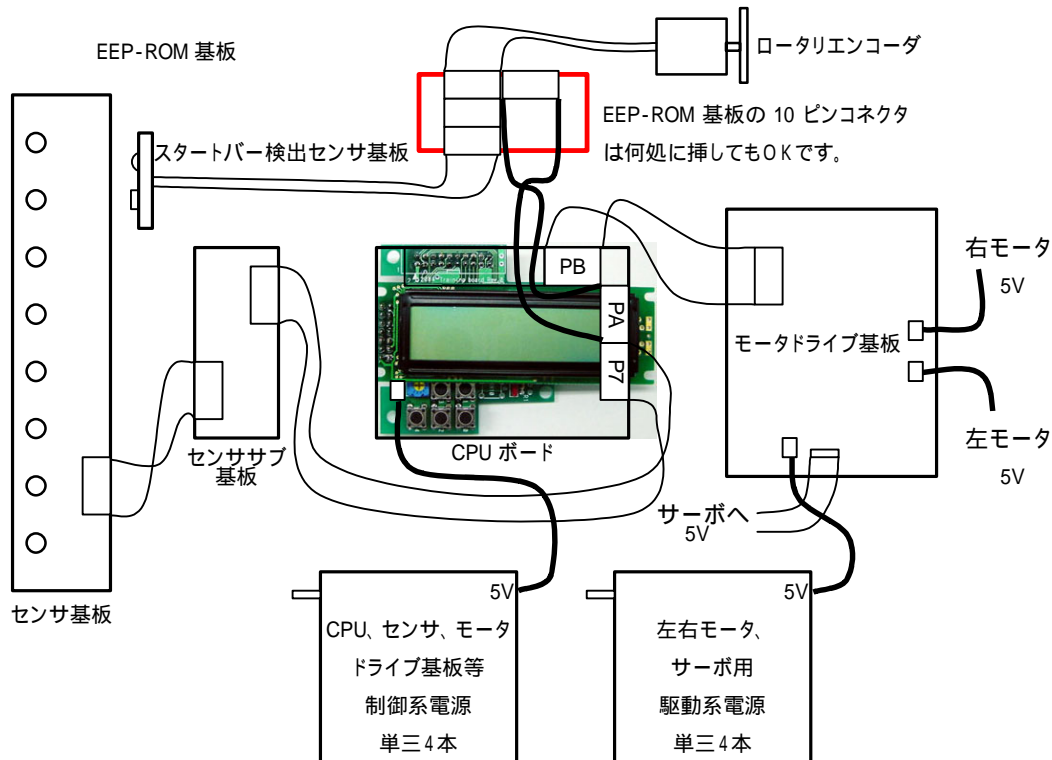
本プログラムは、「tr\_31.c」の RAM 部分を EEPROM に変更した内容となります。

### 22.2 プロジェクトの構成

	
Assembly source file	· tr_32tart.src ベクタアドレスの設定、スタートアップルーチンが記述されています。
tr_32start.src	
C source file	· car_printf2.c セクションの初期化、printf 文、scanf 文を実行します。
beep.c	· lcd2.c LCD 制御を行います。
car_printf2.c	· switch.c スイッチ制御を行います。
eprom.c	· beep.c ブザー制御を行います。
i2c_eprom.c	· eeprom.c EEP-ROM 制御を行います。トレーニングボード上に載っている 93C56 という EEPROM 制御用です。設定保存用として使用しています。
lcd2.c	· tr_32.c メインプログラムです。
switch.c	· i2c_eprom.c EEP-ROM 制御するプログラムを追加しています。こちらは、今回外付けした EEPROM で、24C256 という EEPROM 制御用です。データ保存用として使用しています。
tr_32.c	
Dependencies	
beep.h	
eeprom.h	
h8_3048.h	
i2c_eprom.h	
lcd2.h	
switch.h	

## 22.3 マイコンカーの構成

ポートAにEEP-ROM基板を接続します。スタートパー検出センサもエンコーダもポートAなので、線で分岐するなどして上手く接続するようにしてください。市販されているEEP-ROM基板を使用する場合は、10ピンコネクタに挿すだけで大丈夫です。



## 22.4 プログラム「tr\_32.c」

プログラムのゴシック体部分が、24C256を制御するために変更、追加した部分です。

```

1 : /*-----*/
2 : /* トレーニングボードを使用したマイコンカートレースプログラム(kit06版) */
3 : /* 2006.08 ジャパンマイコンカーラリー実行委員会 */
4 : /*-----*/
5 :
6 : /*-----*/
7 : /* インクルード */
8 : /*-----*/
9 : #include <no_float.h> /* stdioの簡略化 最初に置く*/
10 : #include <stdio.h>
11 : #include <machine.h>
12 : #include "h8_3048.h"
13 : #include "lcd2.h" /* LCD表示用追加 */
14 : #include "switch.h" /* スイッチ追加 */
15 : #include "beep.h" /* ブザー追加 */
16 : #include "eprom.h" /* EEPROM追加(設定保存) */
17 : #include "i2c_eprom.h" /* EEPROM追加(データ記録) */
18 :
19 : /*-----*/
20 : /* シンボル定義 */
21 : /*-----*/
22 :
23 : /* 定数設定 */
24 : #define TIMER_CYCLE 3071 /* タイマのサイクル 1ms */
25 : /* /8で使用する場合、 */
26 : /* /8 = 325.5[ns] */
27 : /* TIMER_CYCLE = */
28 : /* 1[ms] / 325.5[ns] */
29 : /* = 3072 */

```

トレーニングボード 実習マニュアル(kit06 版)

```

30 : #define      PWM_CYCLE      24575 /* PWMのサイクル 8msに変更 */
31 :                                     /* PWM_CYCLE = */
32 :                                     /*      8[ms] / 325.5[ns] */
33 :                                     /*      = 24576 */
34 : #define      SERVO_CENTER    5000 /* サーボのセンタ値 */
35 : #define      HANDLE_STEP     26   /* 1°分の値 */
36 :
37 : /* マスク値設定  x : マスクあり(無効)      : マスク無し(有効) */
38 : #define      MASK2_2          0x66 /* x   x x   x */
39 : #define      MASK2_0          0x60 /* x   x x x x */
40 : #define      MASK0_2          0x06 /* x x x x x   x */
41 : #define      MASK3_3          0xe7 /*   x   x   */
42 : #define      MASK0_3          0x07 /* x x x x x   */
43 : #define      MASK3_0          0xe0 /*   x   x x x x */
44 : #define      MASK4_0          0xf0 /*   x   x x x x */
45 : #define      MASK0_4          0x0f /* x x x x   */
46 : #define      MASK4_4          0xff /* */
47 :
48 : /* EEP-ROM関連 */
49 : #define      EEP_ROM_SIZE     16   /* EEP-ROM使用サイズ */
50 :
51 : #define      EEPROM_CHECK     0x00 /* EEP-ROMチェック */
52 : #define      EEPROM_SERVO     0x01 /* サーボセンタ値 */
53 : #define      EEPROM_PWM       0x02 /* PWM値 */
54 : #define      EEPROM_CURVE_ENC 0x03 /* 大カーブのエンコーダ値 */
55 : #define      EEPROM_CRANK_ENC 0x04 /* クランク部分のエンコーダ値 */
56 : #define      EEPROM_CRANK2_ENC 0x05 /* クランク部分のエンコーダ値2 */
57 : #define      EEPROM_STOP_DISTANCE 0x06 /* 停止距離 */
58 :
59 : /* ===== */
60 : /* プロトタイプ宣言 */
61 : /* ===== */
62 : void init( void );
63 : void timer( unsigned long timer_set );
64 : int check_crossline( void );
65 : int check_rightline( void );
66 : int check_leftline( void );
67 : unsigned char sensor_inp( unsigned char mask );
68 : unsigned char dipsw_get( void );
69 : unsigned char pushsw_get( void );
70 : unsigned char startbar_get( void );
71 : void led_out( unsigned char led );
72 : void speed( int accele_l, int accele_r );
73 : void speed2( int accele_l, int accele_r );
74 : void handle( int angle );
75 : char unsigned bit_change( char unsigned in );
76 : int diff( int pwm );
77 :
78 : /* ===== */
79 : /* グローバル変数の宣言 */
80 : /* ===== */
81 : unsigned long cnt0; /* timer関数用 */
82 : unsigned long cnt1; /* main内で使用 */
83 : int pattern; /* パターン番号 */
84 :
85 : /* EEP-ROM設定 */
86 : int eep_buff[ EEP_ROM_SIZE ];
87 :
88 : /* 角度関連 */
89 : int angle_buff; /* 現在ハンドル角度保持用 */
90 :
91 : const revolution_difference[] = { /* 角度から内輪、外輪回転差計算 */
92 :     100, 99, 97, 96, 95,
93 :     93, 92, 91, 89, 88,
94 :     87, 86, 84, 83, 82,
95 :     81, 79, 78, 77, 76,
96 :     75, 73, 72, 71, 70,
97 :     69, 67, 66, 65, 64,
98 :     62, 61, 60, 59, 58,
99 :     56, 55, 54, 52, 51,
100 :     50, 48, 47, 46, 44,
101 :     43 };
102 :
103 : /* 割り込み内関連 */
104 : int iTimer10; /* 取得間隔計算用 */
105 :
106 : /* エンコーダ関連 */
107 : long iEncoderTotal; /* 積算値 */
108 : int iEncoderMax; /* 現在最大値 */
109 : int iEncoder; /* 現在値 */
110 : unsigned int uEncoderBuff; /* 前回値保存 */
111 : long iEncoderLine; /* ライン検出時の積算値 */
112 :
113 : /* データ保存関連 */
114 : int saveIndex; /* 保存インデックス */
115 : int saveSendIndex; /* 送信インデックス */
116 : int saveFlag; /* 保存フラグ */
117 : char saveData[8]; /* 一時保存エリア */
118 : /*
119 : 保存内容
120 : 0:pattern 1:Sensor 2:handle 3:motor_l

```

トレーニングボード 実習マニュアル(kit06 版)

```

121 : 4:motor_r 5:encoder 6:          7:
122 : */
123 :
124 : /*****/
125 : /* メインプログラム */
126 : /*****/
127 : void main( void )
128 : {
129 :     int    i, j;
130 :     int    lcd_pattern = 2;
131 :
132 :     /* マイコン機能の初期化 */
133 :     init(); /* 初期化 */
134 :     init_sci1( 0x00, 79 ); /* SC11初期化 */
135 :     set_ccr( 0x00 ); /* 全体割り込み許可 */
136 :     initLcd(); /* LCD初期化 */
137 :     initSwitch(); /* スイッチ初期化 */
138 :     initBeep(); /* ブザー初期化 */
139 :     initEeprom(); /* EEP-ROM初期化(設定保存) */
140 :     initI2CEeprom( &PADDR, &PADR, 0x56, 7, 5); /* EEP-ROM初期設定 */
141 :
142 :     /*EEP-ROMのチェック */
143 :     if( readEeprom( EEPROM_CHECK ) != 0x2006 ) {
144 :         /*
145 :         IDのチェック EEP-ROMを初めて使うかどうか
146 :         0x00番地にIDが書かれていなければ初めて使うと判断して初期化する
147 :         */
148 :         eep_buff[EEPROM_CHECK] = 0x2006;
149 :         eep_buff[EEPROM_SERVO] = SERVO_CENTER;
150 :         eep_buff[EEPROM_PWM] = 50;
151 :         eep_buff[EEPROM_CURVE_ENC] = 10;
152 :         eep_buff[EEPROM_CRANK_ENC] = 10;
153 :         eep_buff[EEPROM_CRANK2_ENC] = 15;
154 :         eep_buff[EEPROM_STOP_DISTANCE] = 100;
155 :     } else {
156 :         /* 2回目以降の使用の場合、データ読み込み */
157 :         for( i=0; i<EEP_ROM_SIZE; i++ ) {
158 :             eep_buff[ i ] = readEeprom( i );
159 :         }
160 :     }
161 :
162 :     /* マイコンカーの状態初期化 */
163 :     handle( 0 );
164 :     speed( 0, 0 );
165 :
166 :     /* 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 */
167 :     if( pushsw_get() ) {
168 :         pattern = 71;
169 :         cnt1 = 0;
170 :     }
171 :
172 :     while( 1 ) {
173 :
174 :         I2CEepromProcess(); /* I2C EEP-ROM保存処理 */
175 :
176 :         switch( pattern ) {
177 :
178 :         case 0:
179 :             /* スイッチ入力待ち */
180 :             if( pushsw_get() ) {
181 :                 setBeepPattern( 0xc000 );
182 :                 /* 保存 */
183 :                 for( i=0; i<EEP_ROM_SIZE; i++ ) {
184 :                     writeEeprom( i, eep_buff[ i ] );
185 :                     while( !checkEeprom() ); /* 書き込み終了チェック */
186 :                 }
187 :                 clearI2CEeprom(); /* 数秒かかる */
188 :                 pattern = 1;
189 :                 cnt1 = 0;
190 :                 break;
191 :             }
192 :
193 :             case 75:
194 :                 /* タイトル転送、準備 */
195 :                 printf( "\n" );
196 :                 printf( "tr_32 Data Out\n" );
197 :                 printf( "Encoder Max = %d\n", iEncoderMax );
198 :                 printf( "Pattern, Sensor, ハンドル, " );
199 :                 printf( "左モータ, 右モータ, Encoder\n" );
200 :                 pattern = 76;
201 :                 break;
202 :
203 :             case 76:
204 :                 /* データ転送 */
205 :
206 :                 /* 終わりのチェック */
207 :                 if( (readI2CEeprom( saveSendIndex )==0) ||
208 :                     (saveSendIndex >= 0x8000) ) {
209 :                     pattern = 77;

```

中略



トレーニングボード 実習マニュアル(kit06 版)

```

889 :         cnt1 = 0;
890 :         break;
891 :     }
892 :
893 :     /* データの転送 */
894 :     printf( "%d,%02x,%d,%d,%d,%d\n",
895 :         (char)readI2CEeprom( saveSendIndex+0 ), /* バターン */
896 :         (unsigned char)readI2CEeprom( saveSendIndex+1 ), /* センサ */
897 :         (char)readI2CEeprom( saveSendIndex+2 ), /* ハンドル */
898 :         (char)readI2CEeprom( saveSendIndex+3 ), /* 左モータ */
899 :         (char)readI2CEeprom( saveSendIndex+4 ), /* 右モータ */
900 :         (char)readI2CEeprom( saveSendIndex+5 ) /* エンコーダ */
901 :     );
902 :
903 :     /* 状態表示 */
904 :     lcdPosition( 0, 0 );
905 :     /* 012345..9ab..f 1行16文字 */
906 :     lcdPrintf( "data %05d/%05d", saveSendIndex, saveIndex );
907 :
908 :     led_out( (saveSendIndex/32) % 2 + 1 ); /* LED点滅処理 */
909 :
910 :     saveSendIndex += 8; /* 次の準備 */
911 :     break;
912 :
913 : case 77:
914 :     /* 転送終了 */
915 :     led_out( 0x3 );
916 :     break;
917 :
918 : default:
919 :     /* どれでもない場合は待機状態に戻す */
920 :     pattern = 0;
921 :     break;
922 : }
923 : }
924 : }

```

中略

```

966 : /*******/
967 : /* ITU0 割り込み処理 */
968 : /*******/
969 : #pragma interrupt( interrupt_timer0 )
970 : void interrupt_timer0( void )
971 : {
972 :     unsigned int i;
973 :
974 :     ITU0_TSR &= 0xfe; /* フラグクリア */
975 :     cnt0++;
976 :     cnt1++;
977 :
978 :     /* LCD表示処理用関数です。1ms毎に実行します。 */
979 :     lcdShowProcess();
980 :     /* 拡張スイッチ用関数です。1ms毎に実行します。 */
981 :     switchProcess();
982 :     /* ブザー処理用関数です。1ms毎に実行します。 */
983 :     beepProcess();
984 :
985 :     /* エンコーダ関連 */
986 :     iTimer10++;
987 :     if( iTimer10 >= 10 ) {
988 :         iTimer10 = 0;
989 :         i = ITU2_CNT;
990 :         iEncoder = i - uEncoderBuff;
991 :         lEncoderTotal += iEncoder;
992 :         if( iEncoder > iEncoderMax )
993 :             iEncoderMax = iEncoder;
994 :         uEncoderBuff = i;
995 :
996 :     /* ログ保存関連 */
997 :     if( saveFlag ) {
998 :         saveData[0] = pattern;
999 :         saveData[1] = sensor_inp( 0xff );
1000 :         /* 2はハンドル関数内で保存 */
1001 :         /* 3はモータ関数内で左モータPWM値保存 */
1002 :         /* 4はモータ関数内で右モータPWM値保存 */
1003 :         saveData[5] = iEncoder;
1004 :         saveData[6] = 0; /* 予備 */
1005 :         saveData[7] = 0; /* 予備 */
1006 :         setPageWriteI2CEeprom( saveIndex, 8, saveData );
1007 :         saveIndex += 8;
1008 :         if( saveIndex >= 0x8000 ) saveFlag = 0;
1009 :     }
1010 : }
1011 : }

```



## 22.5 プログラムの解説

EER-ROM に関する関数は、データ解析実習マニュアルを参照してください。

### 22.5.1 保存するデータ

このプログラムでは、

- ・パターン番号
- ・センサ状態
- ・ハンドル角度
- ・左モータ PWM
- ・右モータ PWM
- ・エンコーダ値
- ・予備
- ・予備

の 8 つを保存します。保存は 2 の n 乗個という決まりがあるため 6 個の保存はできません。そのため、予備を 2 個分用意して、8 個分のデータを保存しています。保存間隔は 10ms ごとです。

### 22.5.2 外付け EEP-ROM を使用するための初期設定

```

132 :      /* マイコン機能の初期化 */
133 :      init();                          /* 初期化                */
134 :      init_sci1( 0x00, 79 );            /* SCI1 初期化          */
135 :      set_ccr( 0x00 );                  /* 全体割り込み許可    */
136 :      initLcd();                         /* LCD 初期化           */
137 :      initSwitch();                     /* スイッチ初期化      */
138 :      initBeep();                        /* ブザー初期化        */
139 :      initEeprom();                      /* EEP-ROM 初期化(設定保存) */
140 :      initI2CEeprom( &PADDR, &PADR, 0x56, 7, 5); /* EEP-ROM 初期設定 */
    
```

140 行で、24C256 という外付けの EEP-ROM を使用するための初期化を行います。EEP-ROM が 2 つあります。まとめると下表のようになります。勘違いしないようにしてください。

関数名	定義されているファイル名	内容
initEeprom	eeeprom.c	トレーニングボード上の 93C56 という EEP-ROM(容量 128 ワード)を初期化するための関数です。 <b>マイコンカーの設定データを保存するためのメモリです。</b> 1 ワード=16bit 幅
initI2CEeprom	i2c_eeeprom.c	ポート A に接続している 24C256 という EEP-ROM(容量 32KB)を初期化するための関数です。 <b>走行データを保存するためのメモリです。</b>

### 22.5.3 転送データ

転送方法は、tr\_31 と同じです。転送データ例を下記に示します。

tr_32 Data Out	52,1c,0,0,0,11
Encoder Max = 0	52,1f,0,0,0,11
Pattern, Sensor, ハンドル, 左モータ, 右モータ, Encoder	52,1f,0,0,0,10
11,18,0,50,50,0	53,1f,8,0,0,11
11,18,0,50,50,1	53,1c,8,0,0,10
11,18,0,50,50,0	53,1c,8,0,0,10
11,18,0,50,50,1	53,1c,8,0,0,9
11,18,0,50,50,1	53,1c,8,0,0,9
11,18,0,50,50,2	53,18,0,70,70,9
11,18,0,50,50,1	53,18,0,0,0,9
11,18,0,50,50,2	53,18,0,0,0,10
11,18,0,50,50,3	53,18,0,0,0,9
11,18,0,50,50,3	53,18,0,0,0,9
11,18,0,50,50,2	53,1c,8,70,62,9
11,18,0,50,50,2	53,1c,8,0,0,10
11,18,0,50,50,4	53,1c,8,0,0,10
11,18,0,50,50,6	53,1c,8,0,0,9
11,18,0,50,50,6	53,1c,8,70,62,10
11,18,0,50,50,7	53,1c,8,0,0,10
11,18,0,50,50,8	53,1c,8,0,0,10
11,18,0,50,50,8	53,1c,8,0,0,9
11,18,0,50,50,9	53,1c,8,70,62,9
11,18,0,50,50,10	53,1c,8,70,62,9
11,18,0,50,50,9	53,18,0,70,70,9
11,18,0,50,50,10	53,18,0,70,70,9
11,18,0,50,50,10	53,18,0,70,70,9
52,1f,0,0,0,11	53,18,0,70,70,9
52,1f,0,0,0,11	53,18,0,70,70,9
52,1f,0,0,0,11	53,18,0,70,70,9
52,1c,0,0,0,11	53,1c,8,70,62,10
52,1c,0,0,0,10	53,1c,8,0,0,9

### 22.6 エンコーダの回転数が違う場合の変更点

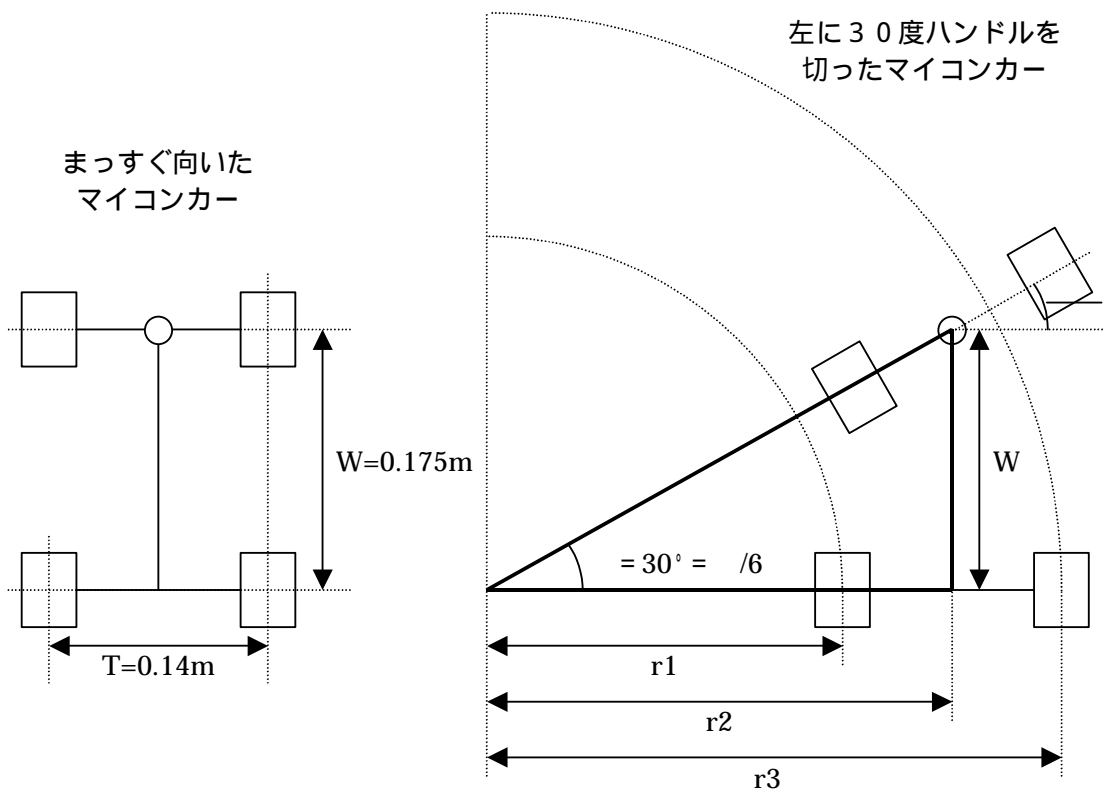
このサンプルプログラムは、100 パルス / 回転、エンコーダのタイヤ直径 33mm のエンコーダを使用した場合です。条件が違うとき、プログラムを変更しなければいけない部分を下記に示します。

行番号	元の数値	変更後の数値
417	965	10cm 進んだときのパルス数を入れます。 例)16 パルス / 回転、直径 21mm なら <b>242</b>
600	97	10cm 進んだときのパルス数を入れます。 例)16 パルス / 回転、直径 21mm なら <b>24</b>
704	97	10cm 進んだときのパルス数を入れます。 例)16 パルス / 回転、直径 21mm なら <b>24</b>
774	97	10cm 進んだときのパルス数を入れます。 例)16 パルス / 回転、直径 21mm なら <b>24</b>

## 23. カーブでのタイヤの左右回転差の計算方法

ハンドルを切ったとき、内輪と外輪ではタイヤの回転数が違います。その計算方法を下記に示します。

### 23.1 2WD、後輪駆動、センターピボット方式



T = トレッド...左右輪の中心線の距離 キットでは 0.14[m] です。

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

図のように、底辺  $r_2$ 、高さ  $W$ 、角度  $\theta$  の三角形の関係は次のようです。

$$\tan \theta = W / r_2$$

角度  $\theta$ 、 $W$  が分かっていますので、 $r_2$  が分かります。

$$r_2 = W / \tan \theta = 0.175 / \tan(\pi/6) = 0.303[m]$$

内輪の半径は、

$$r_1 = r_2 - T/2 = 0.303 - 0.07 = 0.233[m]$$

外輪の半径は、

$$r_3 = r_2 + T/2 = 0.303 + 0.07 = 0.373[m]$$

よって、外輪を 100 とすると内輪の回転数は、

$$r_1 / r_3 \times 100 = 0.233 / 0.373 \times 100 = 62$$

となります。

左に 30° ハンドルを切ったとき、右タイヤ 100 に対して、左タイヤ 62 の回転となる。

プログラムでは次のようにすると、内輪と外輪のロスのない回転ができます。

```
handle( -30 );
speed( 62, 100 );
```

エクセルで、表を作って 0~45 度くらいまでの角度と左右タイヤの回転比の表を作っておくと便利です。

	A	B	C	D	E	F	G
1		W	0.175	m ←ホイールベースを入力してください			
2		T	0.14	m ←トレッドを入力してください			
3							
4		度	rad	r2	r1	r3	r1/r3*100
5		0	0				100
6		1	0.017	10.031	9.961	10.101	99
7		2	0.035	5.014	4.944	5.084	97
8		3	0.052	3.341	3.271	3.411	96
9		4	0.070	2.504	2.434	2.574	95
10		5	0.087	2.001	1.931	2.071	93
11		6	0.105	1.666	1.596	1.736	92
12		7	0.122	1.426	1.356	1.496	91
13		8	0.140	1.246	1.176	1.316	89
14		9	0.157	1.105	1.035	1.175	88
15		10	0.174	0.993	0.923	1.063	87
16		11	0.192	0.901	0.831	0.971	86
17		12	0.209	0.824	0.754	0.894	84
18		13	0.227	0.758	0.688	0.828	83
19		14	0.244	0.702	0.632	0.772	82
20		15	0.262	0.653	0.583	0.723	81
21		16	0.279	0.611	0.541	0.681	79

セル	内容	値、式の例
C1	ホイールベースを入力	キットなら 0.175
C2	トレッドを入力	キットなら 0.14
B 列	角度を入力 0 から 45 まで	直接入力
C 列	角度 ° を rad に変換	C6 セル = B6*3.14/180
D 列	図の r2 の計算	D6 セル = \$C\$1/TAN(C6)
E 列	図の r1 の計算	E6 セル = D6-\$C\$2/2
F 列	図の r3 の計算	F6 セル = D6+\$C\$2/2
G 列	比率の計算	G6 セル = E6/F6*100

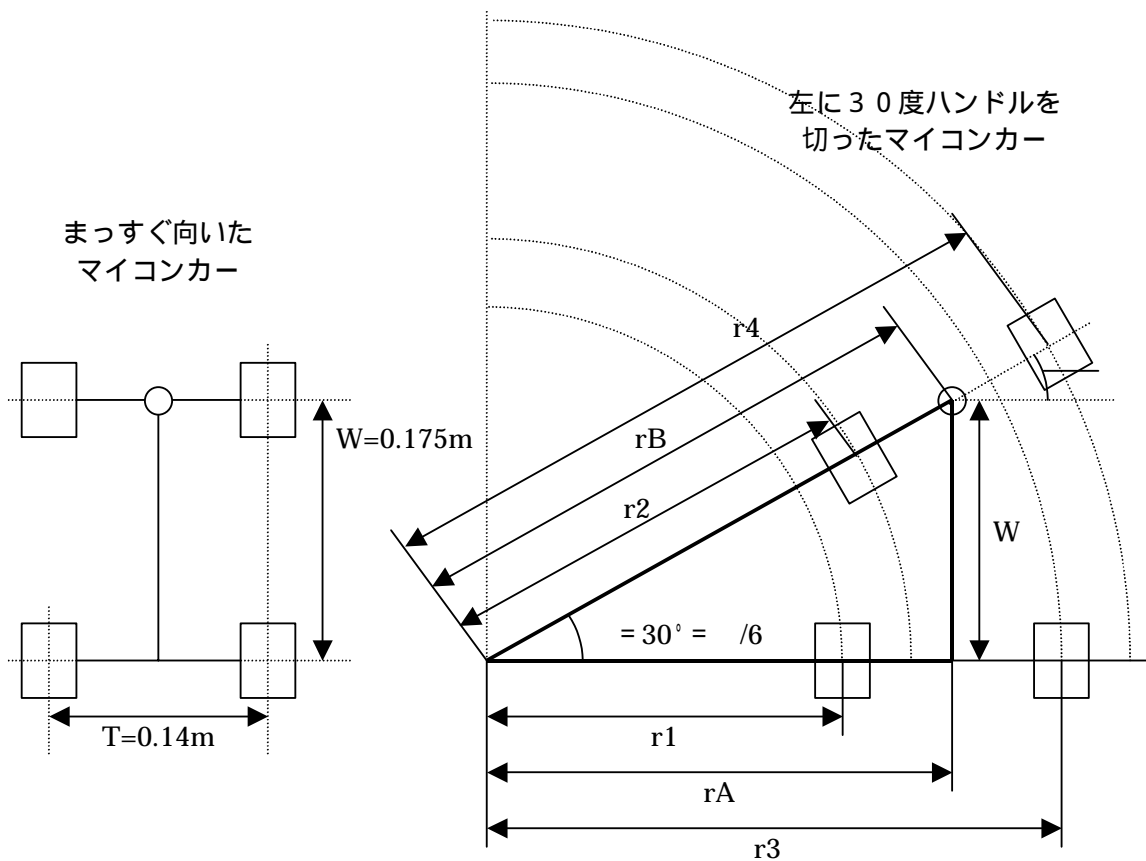
表 内輪側の関係

度	rad	r2	r1	r3	r1/r3*100
0	0				100
1	0.017	10.031	9.961	10.101	99
2	0.035	5.014	4.944	5.084	97
3	0.052	3.341	3.271	3.411	96
4	0.070	2.504	2.434	2.574	95
5	0.087	2.001	1.931	2.071	93
6	0.105	1.666	1.596	1.736	92
7	0.122	1.426	1.356	1.496	91
8	0.140	1.246	1.176	1.316	89
9	0.157	1.105	1.035	1.175	88
10	0.174	0.993	0.923	1.063	87
11	0.192	0.901	0.831	0.971	86
12	0.209	0.824	0.754	0.894	84
13	0.227	0.758	0.688	0.828	83
14	0.244	0.702	0.632	0.772	82
15	0.262	0.653	0.583	0.723	81
16	0.279	0.611	0.541	0.681	79
17	0.297	0.573	0.503	0.643	78
18	0.314	0.539	0.469	0.609	77
19	0.331	0.509	0.439	0.579	76
20	0.349	0.481	0.411	0.551	75
21	0.366	0.456	0.386	0.526	73
22	0.384	0.433	0.363	0.503	72
23	0.401	0.413	0.343	0.483	71
24	0.419	0.393	0.323	0.463	70
25	0.436	0.376	0.306	0.446	69
26	0.454	0.359	0.289	0.429	67
27	0.471	0.344	0.274	0.414	66
28	0.488	0.329	0.259	0.399	65
29	0.506	0.316	0.246	0.386	64
30	0.523	0.303	0.233	0.373	62
31	0.541	0.291	0.221	0.361	61
32	0.558	0.280	0.210	0.350	60
33	0.576	0.270	0.200	0.340	59
34	0.593	0.260	0.190	0.330	58
35	0.611	0.250	0.180	0.320	56
36	0.628	0.241	0.171	0.311	55
37	0.645	0.232	0.162	0.302	54
38	0.663	0.224	0.154	0.294	52
39	0.680	0.216	0.146	0.286	51
40	0.698	0.209	0.139	0.279	50
41	0.715	0.201	0.131	0.271	48
42	0.733	0.195	0.125	0.265	47
43	0.750	0.188	0.118	0.258	46
44	0.768	0.181	0.111	0.251	44
45	0.785	0.175	0.105	0.245	43

W を 0.175[m]、T を 0.14[m]としたときの場合

W と T の値を自分のマイコンカーの長さに変えると、左右のタイヤの回転比率が分かります。

## 23.2 センターピボット方式 4 輪の回転数計算



T = トレッド...左右輪の中心線の距離 キットでは 0.14[m] です。

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

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

$$\tan \theta = W / rA$$

角度  $\theta$ 、W が分かっていますので、rA が分かります。

$$rA = W / \tan \theta = 0.175 / \tan(\theta / 6) = 0.303[m]$$

後輪内輪の半径は、

$$r1 = rA - T/2 = 0.303 - 0.07 = 0.233[m]$$

後輪外輪の半径は、

$$r3 = rA + T/2 = 0.303 + 0.07 = 0.373[m]$$

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

$$\sin \theta = W / rB$$

角度  $\theta$ 、W が分かっていますので、rB が分かります。

$$rB = W / \sin \theta = 0.175 / \sin(\theta / 6) = 0.350[m]$$

前輪内輪の半径は、

$$r2 = rB - T/2 = 0.350 - 0.07 = 0.280[m]$$

前輪外輪の半径は、  
 $r4 = rB + T/2 = 0.350 + 0.07 = 0.420[m]$

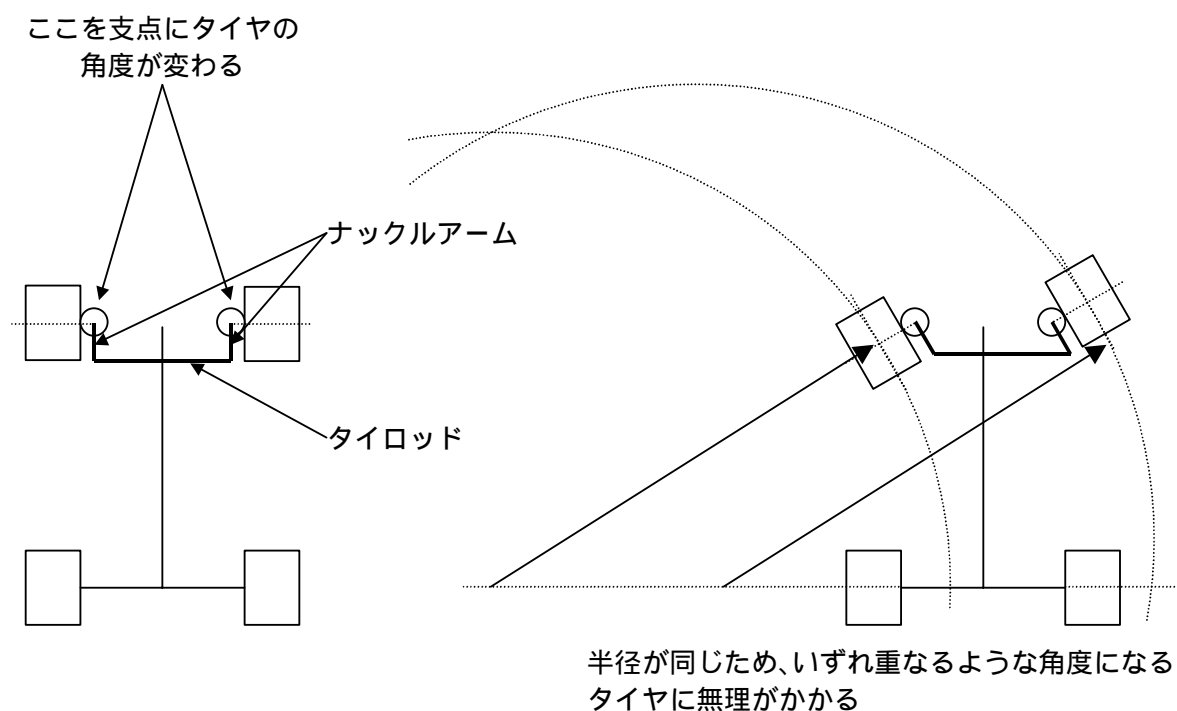
となります。

一番回転する  $r4$  を 100 としたときのそれぞれの回転数は、  
 $r1 : r2 : r3 : r4$   
 $= 0.233 : 0.280 : 0.373 : 0.420$   
 $= 0.233 \times 100/0.420 : 0.280 \times 100/0.420 : 0.373 \times 100/0.420 : 0.420 \times 100/0.420$   
 $= 55 : 67 : 89 : 100$

前輪外輪が 100 回転するとき、後輪外輪は 89 回転、前輪内輪は 67 回転、後輪内輪は 55 回転することになります。

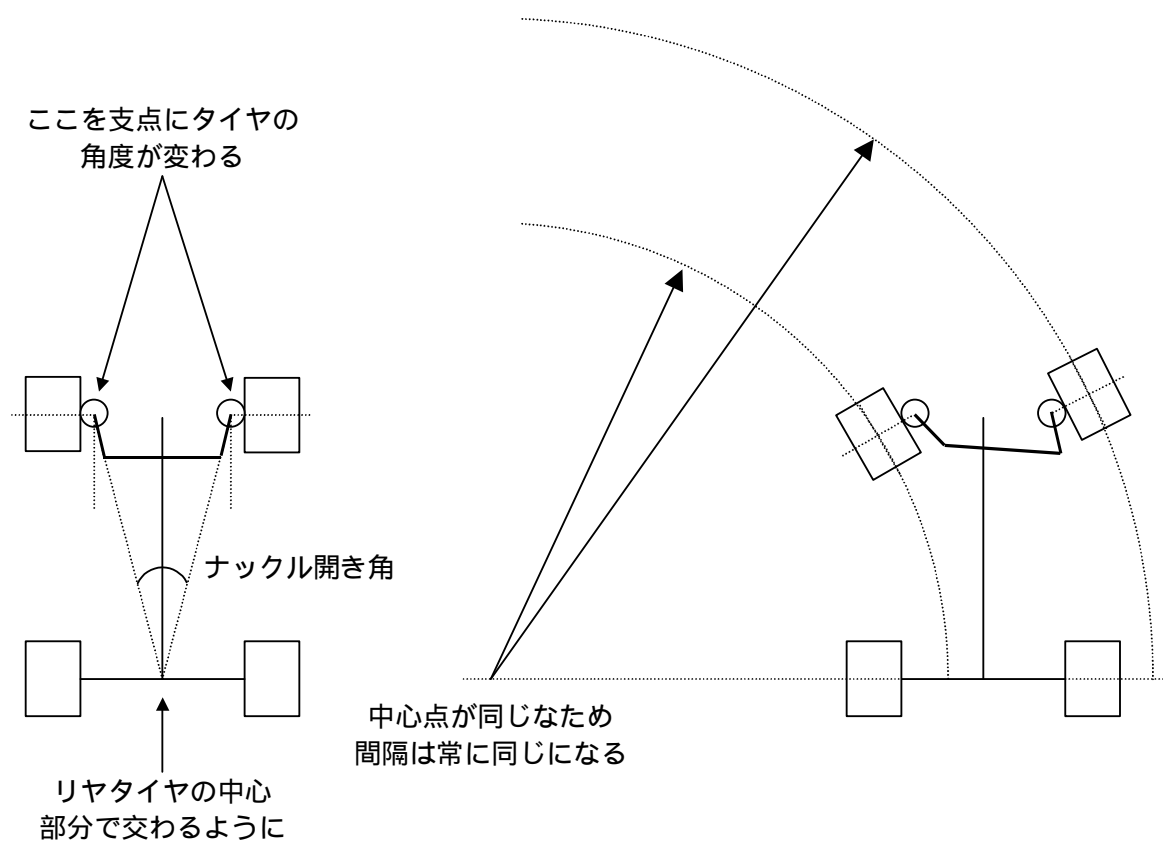
### 23.3 アッカーマン方式 4 輪の回転数計算

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



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

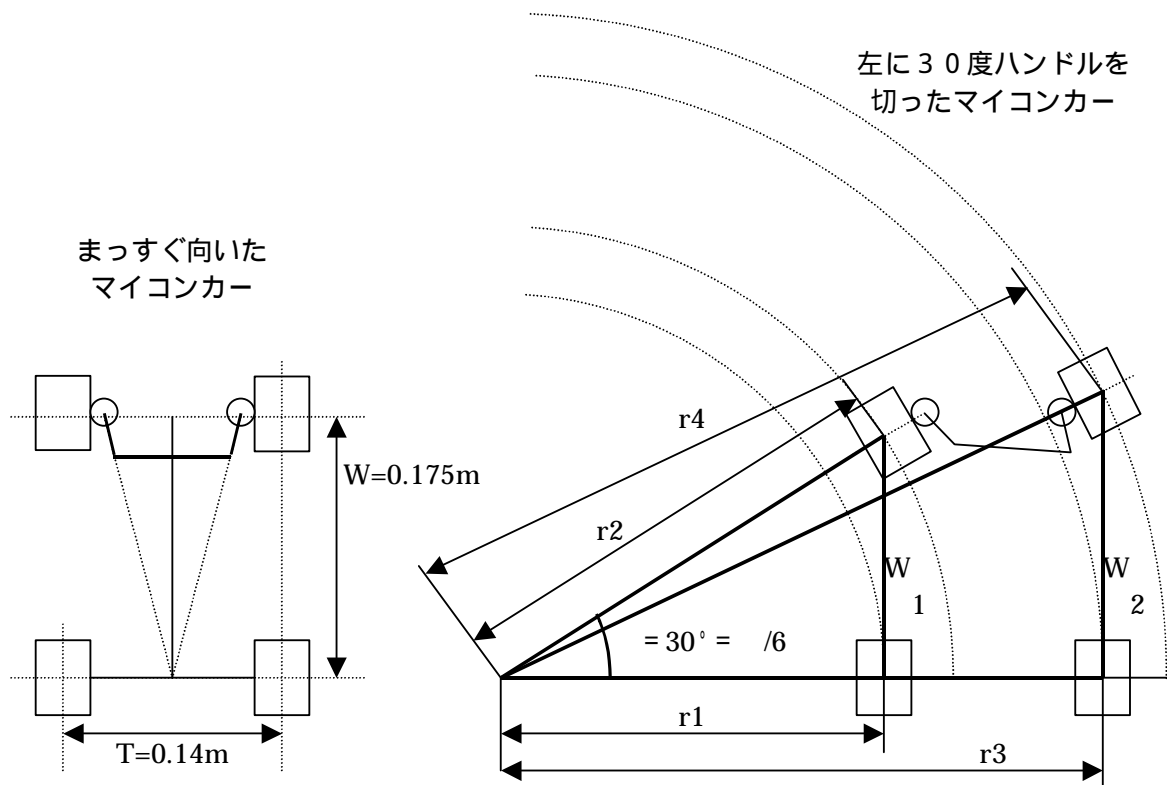
この問題を解決したのが、ドイツ人のアッカーマン、及びフランス人のジャントで、この機構をアッカーマン・ジャント方式、または単にアッカーマン方式と呼びます。



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

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





T=トレッド...左右輪の中心線の距離 キットでは0.14[m]です。

W=ホイールベース...前輪と後輪の間隔 キットでは0.175[m]です。

角度  $\theta$  は、前輪内側タイヤの切れ角です。

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

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

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

角度  $\theta$ 、W が分かっていますので、後輪内輪  $r_1$  が分かります。

$$r_1 = W / \tan \theta = 0.175 / \tan(\theta / 6) = 0.303[\text{m}]$$

後輪外輪の半径は、

$$r_3 = r_1 + T = 0.303 + 0.14 = 0.443[\text{m}]$$

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

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

角度  $\theta$ 、W が分かっていますので、前輪内輪  $r_2$  が分かります。

$$r_2 = W / \sin \theta = 0.175 / \sin(\theta / 6) = 0.350[\text{m}]$$

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

$$r_4 = \sqrt{(r_3^2 + W^2)} = \sqrt{(0.443^2 + 0.175^2)} = 0.476[\text{m}]$$

となります。

一番回転する r4 を 100 としたときのそれぞれの回転数は、  
r1 : r2 : r3 : r4  
= 0.303 : 0.350 : 0.443 : 0.476  
= 0.303 × 100/0.476 : 0.350 × 100/0.476 : 0.443 × 100/0.476 : 0.476 × 100/0.476  
= 64 : 74 : 93 : 100

前輪外輪が 100 回転するとき、後輪外輪は 93 回転、前輪内輪は 74 回転、後輪内輪は 64 回転することになります。

## 24. 参考文献

1. (株)ルネサス テクノロジ  
H8/3048 シリーズ、H8/3048F-ZTAT™ (H8/3048F、H8/3048F-ONE)ハードウェアマニュアル 第7版
2. (株)ルネサス テクノロジ 半導体トレーニングセンター C言語入門コーステキスト 第1版
3. (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
4. ソフトバンク(株) 新C言語入門シニア編 林晴比古著 初版
5. 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
6. 電波新聞社 マイコン入門講座 大須賀威彦著 第1版
7. (株)グランプリ出版 細川武志著 クルマのメカ&仕組み図鑑 ISBN4-87687-241-4

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

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

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

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

の「マイコン」 「H8 ファミリ」 「H8/3048B グループ」でご覧頂けます

リンクは、2007年3月現在の情報です。