マイコンカーラリー用

液晶・microSD 基板 (Ver.2)
kit12_38a プログラム
解説マニュアル
液晶編
（R8C/38A 版）

2013 年度から、RY_R8C38 ボードに搭載されているマイコンが R8C/38A から R8C/38C に変更されました。R8C/38A マイコンと R8C/38C マイコンは、機能的にほぼ互換で、マイコンカーで使う範囲においてはプログラムの変更はほとんどありません。
よって、本マニュアルではマイコンの名称を『R8C/38A』で統一します。

本マニュアルで説明 しているセット内容	液晶・microSD 基板、及び液晶・microSD 基板 Ver.2 ※本マニュアルの「液晶・microSD 基板」は、「液晶・microSD 基板、及び液晶・microSD 基板 Ver.2」と読み替えてください（別途、明記されている部分は除く）。 ※どちらの基板も同じプログラムで動作します
対象マイコンボード	RY_R8C38 ボード(R8C/38A マイコン)

第 1.03A 版
2015.04.27
ジャパンマイコンカーラリー実行委員会
株式会社日立ドキュメントソリューションズ

注 意 事 項 (rev.6.0J)

著作権

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

禁止事項

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

転載、複製

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

責任の制限

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

その他

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

連絡先

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

目　次

1. 概要

※本マニュアルで使用している基板は「液晶・microSD 基板」と「および液晶・microSD 基板 Ver.2」です。どちらの基板も回路は互換なので、プログラムの変更はありません。
※本マニュアルの「液晶・microSD 基板」は、「液晶・microSD 基板、及び液晶・microSD 基板 Ver.2」と読み替えてください（別途、明記されている部分は除く）。

通常、マイコンを使った機器(組み込みシステム)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など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ができれば良いのですが、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、マイコンカーのマイコン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れば良いのですが、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、プログラムの書き替えをしなくても手元でパラメータの調整ができるように、RY_R8C38 ボードに搭載する「液晶・microSD 基板」、及び「「液晶・microSD 基板 Ver.2」」を開発しました。

液晶・microSD 基板、及び液晶・micorSD 基板 Ver.2 の主な機能を下記に示します。

●液晶表示

液晶は、文字を表示して情報を確認することができます。32 文字表示できるので、たくさんの情報を表示することができます。
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

プッシュスイッチは、液晶表示の情報の切り替え、パラメータの調整を行います。プッシュスイッチは5個あります。本プログラムでは、値の保存（1スイッチ）、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個使います。
例）液晶のメニューの切り替え、パラメータの増減用

●microSD（書き替え可能なメモリ）

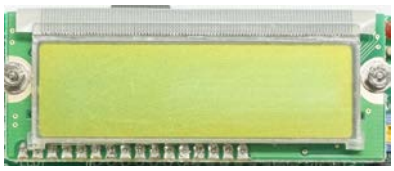
microSD は、マイコンカーの走行データなどを保存し、パソコンで解析することができます。
例）マイコンカーの走行データの保存

本書では液晶・microSD 基板の、液晶、プッシュスイッチの仕様や使い方、マイコンカーに組み込んでパラメータの調整をする方法を説明していきます。microSD の仕様、使い方については、「液晶・microSD 基板 kit12_38a プログラム解説マニュアルデータ解析（microSD）編（R8C/38A 版）」を参照してください。

2. 液晶・microSD 基板の仕様

2.1 仕様

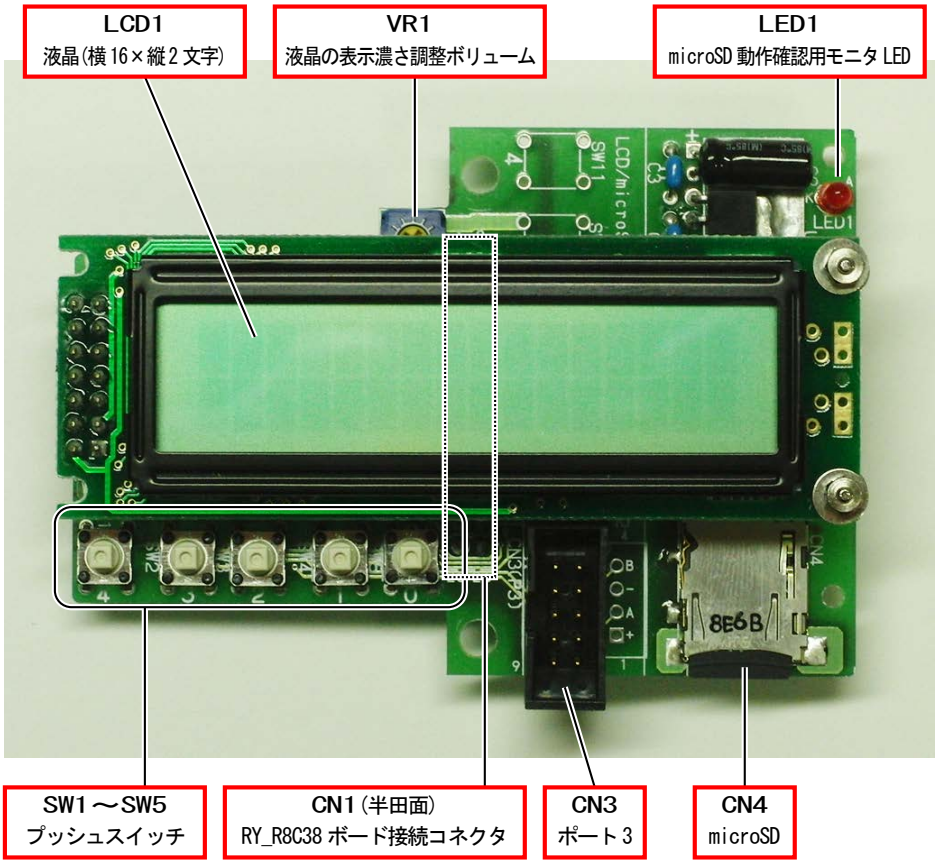
液晶・microSD 基板、及び液晶・microSD 基板 Ver.2 の仕様を、下表に示します。
2 種類の基板の違いは、液晶・microSD 基板 Ver.2 の液晶の方が小型、軽量になり、それに伴い基板外形も変更されています。回路的な違いはほとんど無く、どちらの基板も同じプログラムで動作します。

	液晶・microSD 基板　Ver.2	**液晶・microSD 基板**
部品数	リード線のある部品:約 33 個 面実装部品:1 個（microSD コネクタ）	リード線のある部品:約 36 個 面実装部品:1 個（microSD コネクタ）
RY_R8C38 ボードとの接続方法	RY_R8C38 ボードの CN5(26 ピンコネクタ)と液晶部の CN1 を接続	←
液晶	1 個 横 16 文字×縦 2 文字の表示が可能 外形:66×26×高さ約 8mm 重量:約 16g 	1 個 横 16 文字×縦 2 文字の表示が可能 外形:85×30×高さ約 13mm 重量:約 24g 
プッシュスイッチ	5 個	←
microSD コネクタ	1 個（※microSD は別売りです）	←
LED	1 個 microSD の動作確認用として使用	←
電圧	DC5.0V±10%	←
拡張コネクタ	ポート 3 に接続できる 10 ピンコネクタを 1 個搭載	←
基板外形	幅 76.2×奥行き 48.26×厚さ 1.2mm	幅 85×奥行き 60×厚さ 1.6mm
完成時の寸法(実寸)	幅 76.2×奥行き 48.26×高さ約 28mm (半田面コネクタを除くと高さ約 20mm)	幅 85×奥行き 60×高さ約 35mm (半田面コネクタを除くと高さ約 25mm)
重量	約 41g ※実測です　リード線の長さや半田の量で変わります	約 53g ※実測です　リード線の長さや半田の量で変わります
開発環境	ルネサス統合開発環境	←
サンプルプログラム (サンプルワークスペース)	液晶とプッシュスイッチ　:kit12lcd_38a microSD　:kit12msd_38a	←

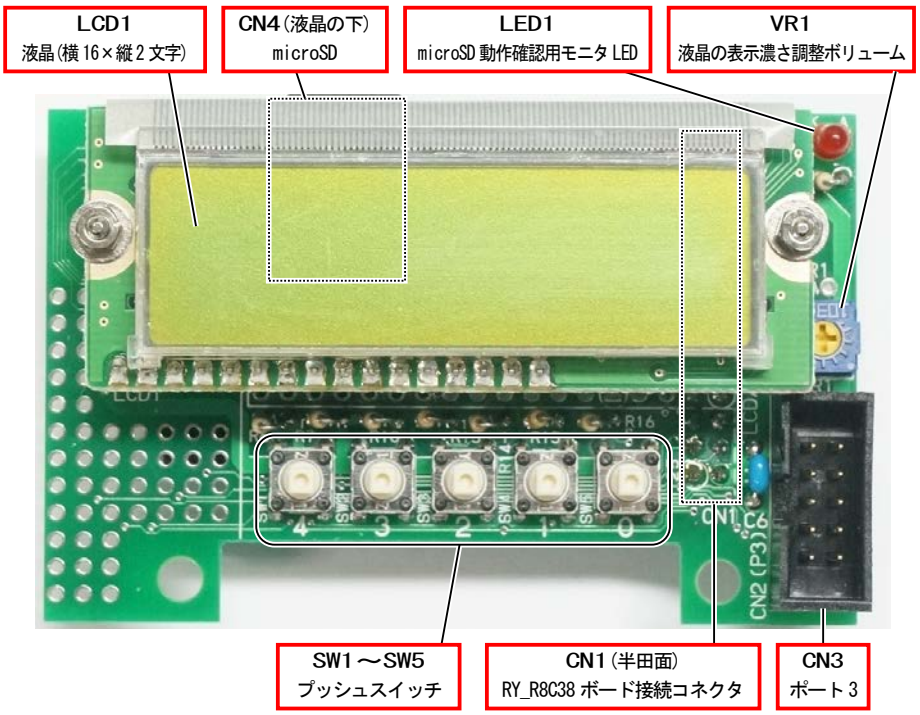
2.2　外観

※液晶が表示されない場合は、ボリューム(VR1)を調整してください。

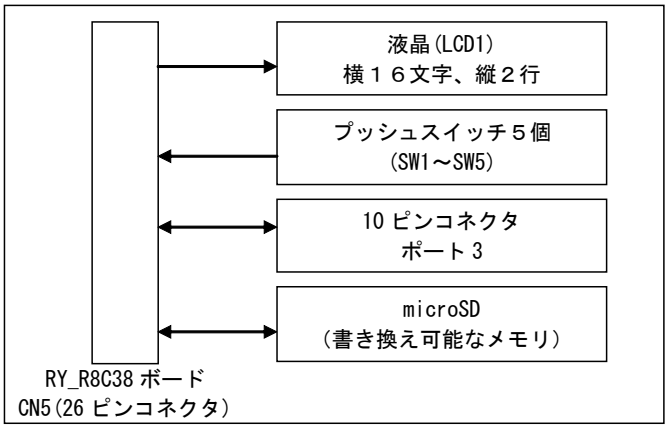
●液晶・microSD 基板の外観



●液晶・microSD 基板 Ver.2 の外観



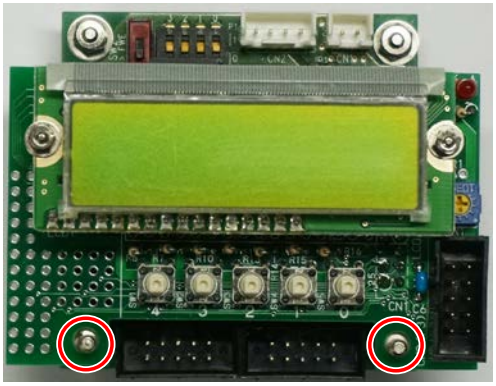
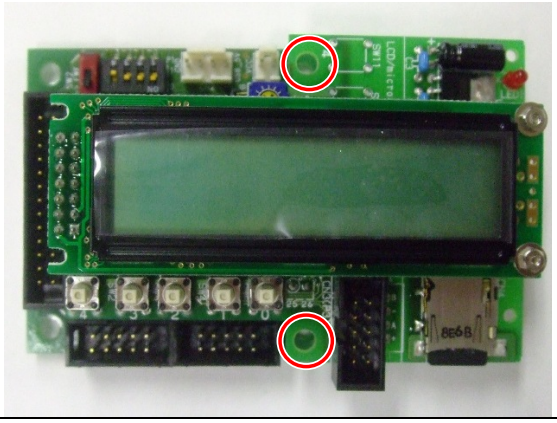
2.3 ブロック図



液晶	液晶は、文字を表示して情報を確認することができます。32 文字表示できるので、たくさんの情報を表示することができます。 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	プッシュスイッチは、液晶表示の情報の切り替え、パラメータの調整を行います。プッシュスイッチは5個あります。本プログラムでは、値の保存(1スイッチ)、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個使います。 例)液晶のメニューの切り替え、パラメータの増減用
10 ピンコネクタ	このコネクタはポート 3 に直結されています。通常の I/O ポートとして使用可能です。また、ロータリエンコーダを使用する場合は、この 10 ピンコネクタに接続します。 例)ロータリエンコーダのパルス入力用
microSD	microSD は、マイコンカーの走行データなどを保存し、パソコンで解析することができます。 例)マイコンカーの走行データの保存

2.4 RY_R8C38 ボードと液晶・microSD 基板の接続

液晶・microSD 基板は、RY_R8C38 ボードの 26 ピンコネクタ部分（CN5）に重ねて使用します。重ねたところを下写真に示します。

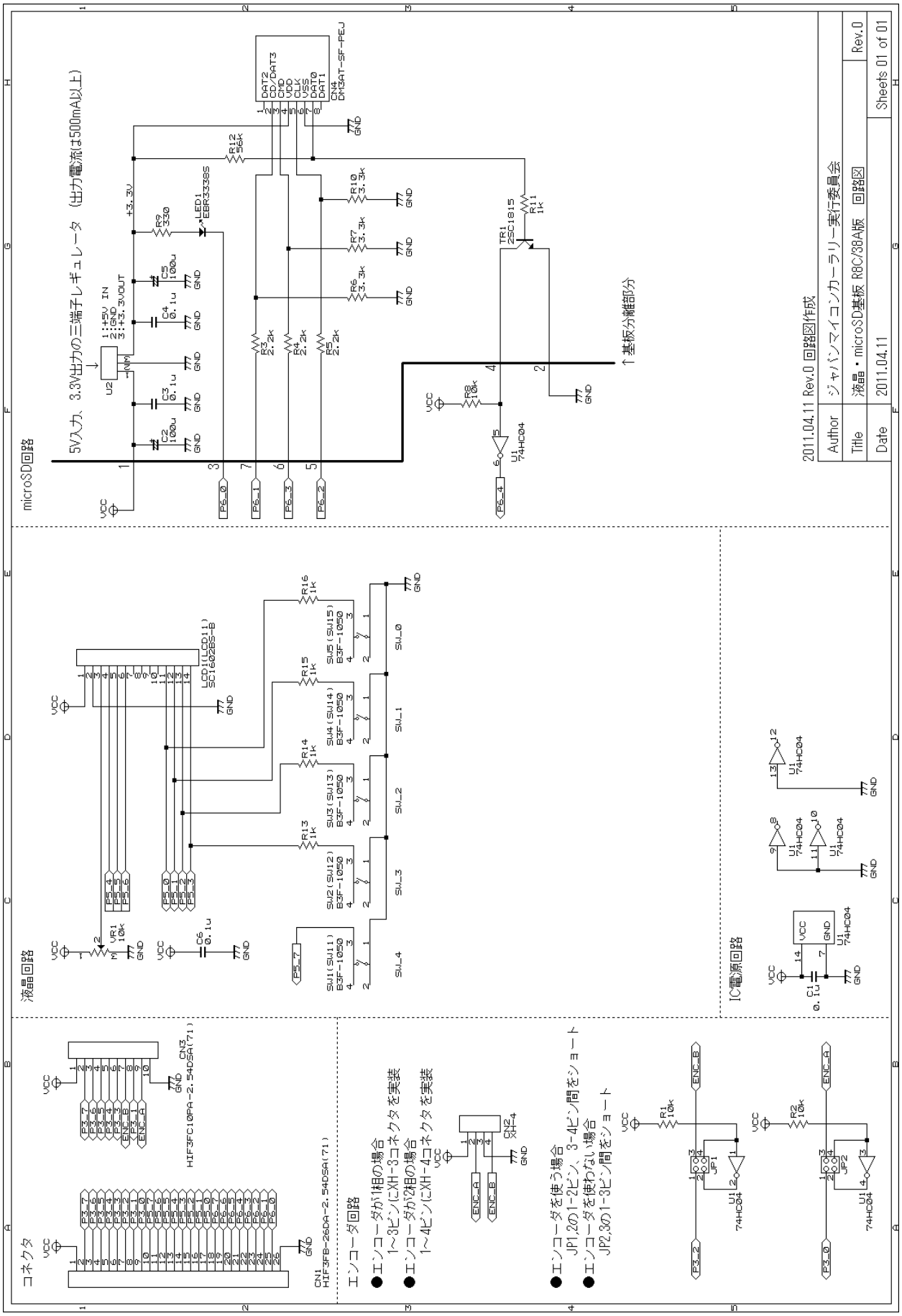
液晶・microSD 基板 Ver.2	液晶・microSD 基板
	

2.5 コネクタ

RY_R8C38 ボードの CN5(ポート 3、ポート 5、ポート 6)のポート表を、下表に示します。

ピン番号	信号名	マイコン ー 液晶・microSD 基板間	接続先
1	Vcc	―	Vcc
2	P3_7	⟷	10 ピンコネクタ 2 番ピン
3	P3_6	⟷	10 ピンコネクタ 3 番ピン
4	P3_5	⟷	10 ピンコネクタ 4 番ピン
5	P3_4	⟷	10 ピンコネクタ 5 番ピン
6	P3_3	⟷	10 ピンコネクタ 6 番ピン
7	P3_2	⟷	10 ピンコネクタ 7 番ピン
8	P3_1	⟷	10 ピンコネクタ 8 番ピン
9	P3_0	⟷	10 ピンコネクタ 9 番ピン
10	P5_7	←	SW_4
11	P5_6	→	液晶 E
12	P5_5	→	液晶 RW
13	P5_4	→	液晶 RS
14	P5_3	液晶: → スイッチ: ←	液晶 D3 and SW_3
15	P5_2	液晶: → スイッチ: ←	液晶 D2 and SW_2
16	P5_1	液晶: → スイッチ: ←	液晶 D1 and SW_1
17	P5_0	液晶: → スイッチ: ←	液晶 D0 and SW_0
18	P6_7	→	
19	P6_6	→	
20	P6_5	→	
21	P6_4	←	microSD DAT0(RXD1)
22	P6_3	→	microSD CMD(TXD1)
23	P6_2	→	microSD CLK(CLK1)
24	P6_1	→	microSD CD
25	P6_0	→	LED
26	GND	―	GND

2.6 液晶・microSD 基板 回路図



3. サンプルプログラム

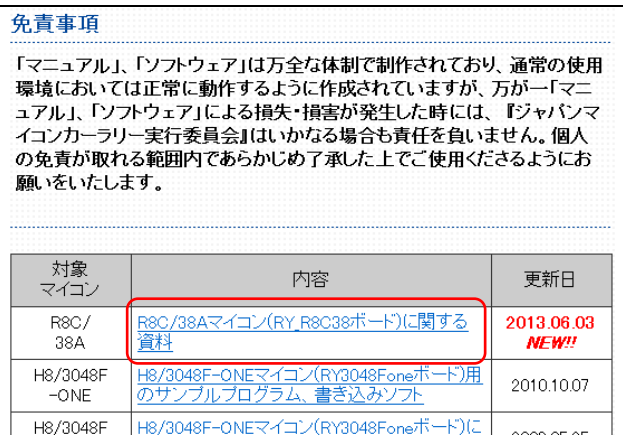
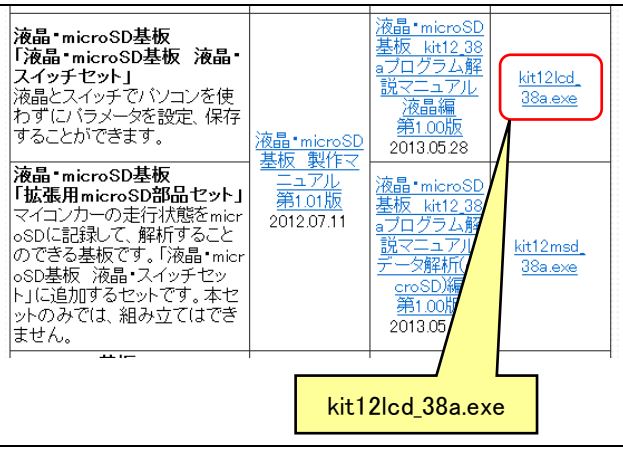
プログラムについて、液晶・microSD 基板と、液晶 microSD 基板 Ver.2 は互換です。どちらの基板も同じプログラムで動作します。

3.1 プログラムの開発環境

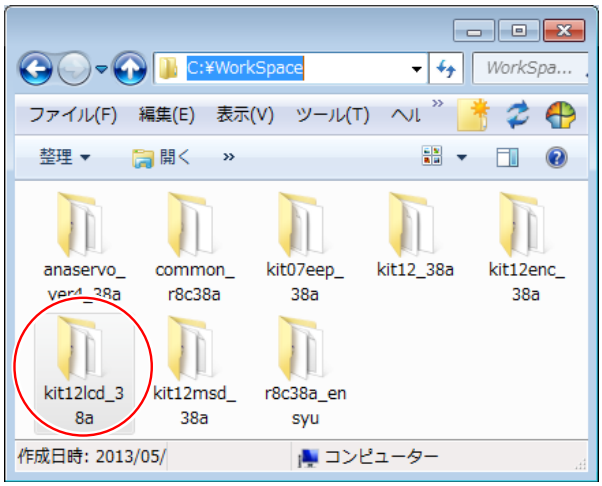
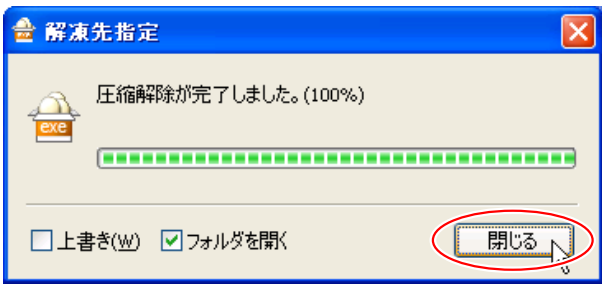
プログラムの開発は、ルネサス統合開発環境（High-performance Embedded Workshop）を使います。ルネサス統合開発環境についてのインストール、開発方法は、「ルネサス統合開発環境　操作マニュアル（R8C/38A 版）」を参照してください。

3.2 サンプルプログラムのインストール

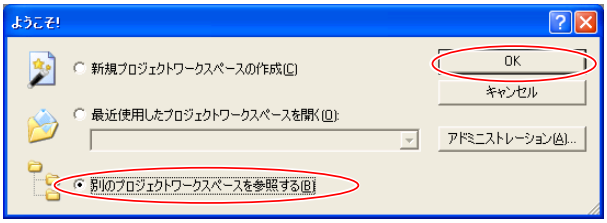
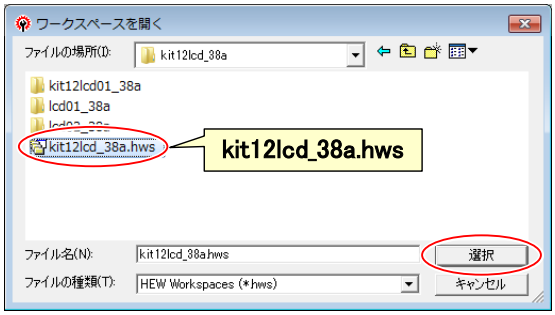
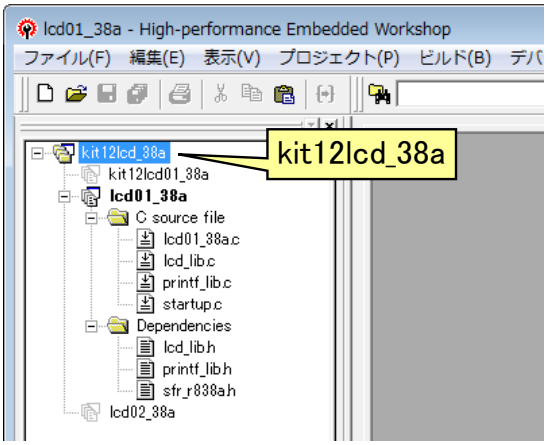
3.2.1 プログラムのダウンロード

1		マイコンカーラリーサイト 「http://www.mcr.gr.jp/index2.html」 の「技術情報→ダウンロード」をアクセスします。
2		「R8C/38A マイコン(RY_R8C38 ボード)に関する資料」をクリックします。
3		「kit12lcd_38a.exe」をクリック、ダウンロードします。

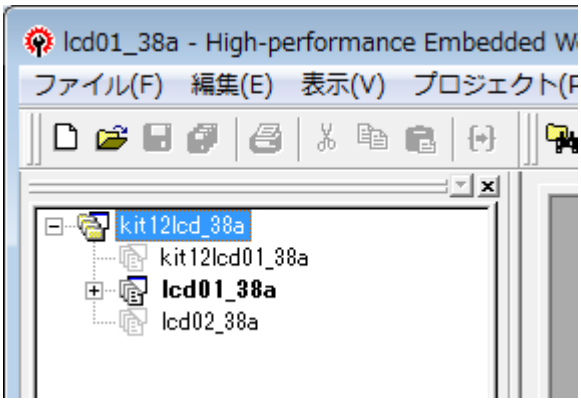
3.2.3 インストール

2		「kit12lcd_38a.exe」を実行します。 圧縮解除をクリックします。 **※解凍先フォルダは変更しないでください。**
3		解凍が終わったら、「Cドライブ→Workspace」フォルダが開かれます。複数のフォルダがあります。今回使用するのは、「**kit12lcd_38a**」です。
4		閉じるをクリックします。

3.3 ワーススペース「kit12lcd_38a」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→kit12lcd_38a の「kit12lcd_38a.hws」を選択し、選択をクリックします。
4		「kit12lcd_38a」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「kit12lcd_38a」には、3 つのプロジェクトが登録されています。

プロジェクト名	内容
lcd01_38a	液晶・microSD 基板の液晶制御についてのサンプルプログラムです。
lcd02_38a	液晶・microSD 基板のプッシュスイッチ制御についてのサンプルプログラムです。
kit12lcd01_38a	標準走行プログラム「kit12_38a.c」を改造して、液晶・microSD 基板の液晶、プッシュスイッチを使えるようにしたプログラムです。本プログラムでは、サーボセンタ、全体の PWM 値の割合、大曲げ PWM 値、クランク PWM 値を調整することができます。

4. 液晶制御ライブラリ

「lcd_lib.c」は、液晶に文字を表示する専用の関数が用意されているファイルです。液晶・microSD 基板の液晶を使用するときは、プロジェクトに「lcd_lib.c」を追加して使用します。
「lcd_lib.c」は、「C:¥WorkSpace¥common_r8c38a」フォルダにあります。

4.1 lcd_lib.c ファイルで使用できる関数

■initLcd 関数

書式	int initLcd(void)
内容	液晶を初期化します。
引数	なし
戻り値	0:異常　1:正常
使用例	asm(" fset I ");　/* 全体の割り込み許可　　*/ initLcd();　　/* LCD 初期化　　*/ **※initLcd 関数は、全体割り込み許可をした後に実行してください。**

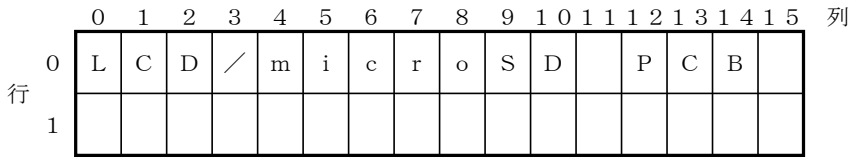
■lcdPrintf 関数

書式	int lcdPrintf(char *format, ...)
内容	液晶に文字を表示します。書式は、printf 関数と同じです。表示位置は前回表示された続きか、lcdPosition 関数で指定された位置です。
引数	char *format　書式文字列 ...　可変個引数
戻り値	正常時:出力した文字列　　異常時:負の数

使用例	lcdPrintf("0123456789abcdef");
	lcdPrintf("0123456789ABCDEF");
	
	実行すると下図のようになります。

行＼列	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

■lcdPosition 関数

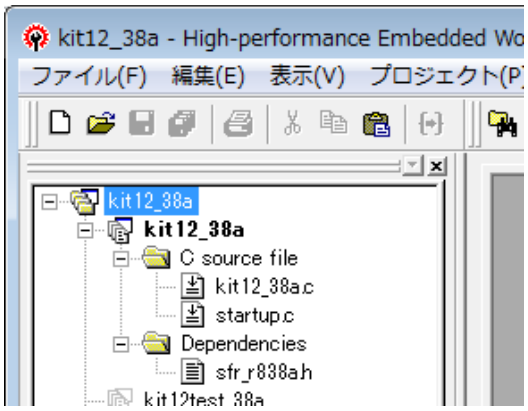
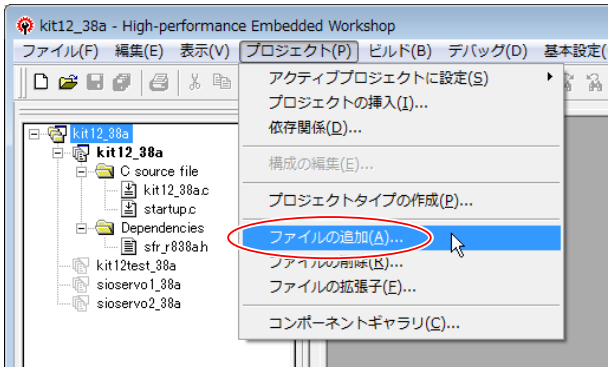
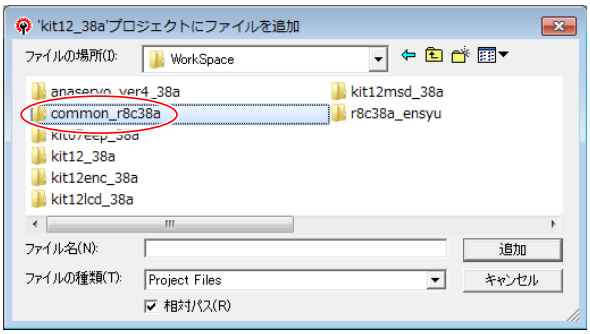
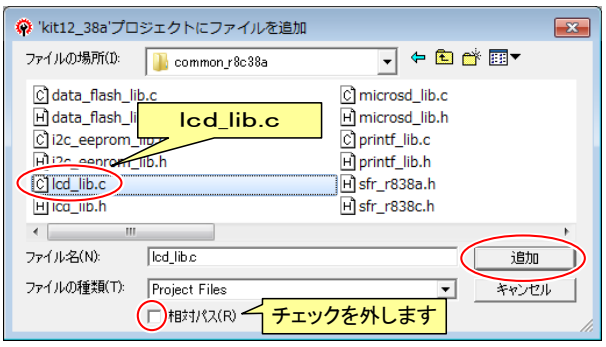
書式	void lcdPosition(char x ,char y)
内容	液晶に表示する位置を指定します。
引数	char x　列 (0 ～ 15) char y　行 (0 ～　1)
戻り値	なし
使用例	lcdPosition(0, 0);　　/* 表示する位置を指定 */ lcdPrintf("LCD/microSD PCB "); /* 表示する文字列　　*/ 実行すると下図のようになります。 

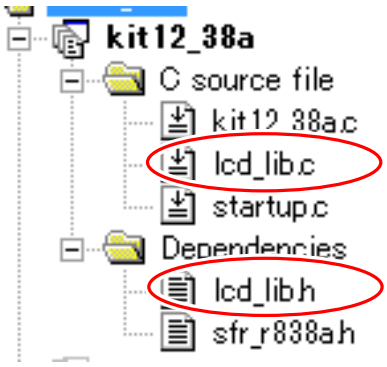
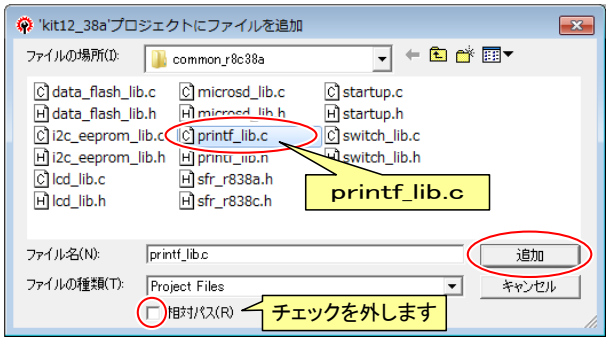
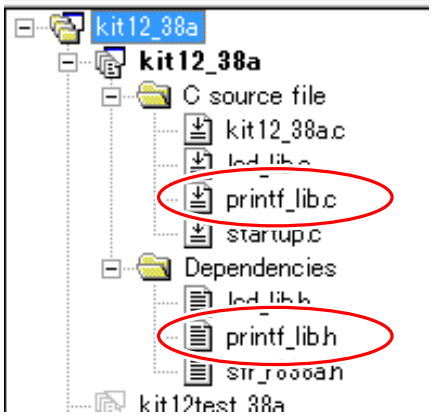
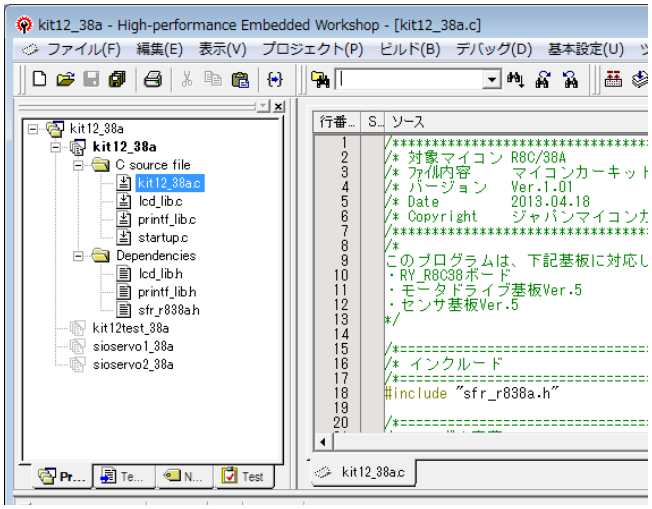
■lcdShowProcess 関数

書式	void lcdShowProcess(void)
内容	液晶の制御を行います。この関数は割り込み処理などで 1ms ごとに実行してください。
引数	なし
戻り値	なし
使用例	/* LCD 表示処理用関数(1ms ごとに実行)　　*/ lcdShowProcess(); 液晶の表示は、1 文字表示に最大で 10ms の時間がかかります(データシート参照)。32 文字表示しようとすると、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 関数は 1ms ごとに実行して、1ms ごとに少しずつ液晶表示処理をする関数です。この関数は 1ms ごとに実行してください。今回は、タイマ RB の 1ms ごとの割り込み関数内で実行します。 **【この関数の仕組み】** 本来は、1 文字分の表示データを液晶に送った後、10ms 待ってから次の表示データを送らなければいけません。 この関数は 1 文字分の表示データを液晶に送った後、すぐに関数を終了します。次にこの関数が実行されても、まだ 1ms しか経っていないので何もせずに関数を終了させます。10 回目で、次の 1 文字分のデータを送り、またすぐに関数を終了します。これを繰り返します。

4.2 プロジェクトに lcd_lib.c を追加する

ワークスペース「kit12_38a」のプロジェクト「kit12_38a」に、lcd_lib.c を追加する方法を紹介します。

1		ルネサス統合開発環境でワークスペース「kit12_38a」を開きます。 プロジェクト「kit12_38a」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_r8c38a」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**。 「lcd_lib.c」を選択、追加をクリックします。 **※「lcd_lib.h」は追加しません。**

5		リストに、「lcd_lib.c」が追加されました。 **※lcd_lib.h は Dependencies 欄に自動で追加されます。**
6		再度、「プロジェクト→ファイルの追加」を選択します。 最初に相対パス欄のチェックを**はずします**。 「printf_lib.c」を選択、追加をクリックします。 **※「printf_lib.h」は追加しません。**
7		リストに、「printf_lib.c」が追加されました。 **※printf_lib.h は Dependencies 欄に自動で追加されます。** **※printf 関数は使いませんが、液晶に表示させるために printf_lib.c 内の関数を使います。**
8		後は、「kit12_38a.c」ファイル内に、液晶制御に関するプログラムを追加してください。

5. プッシュスイッチ制御ライブラリ

「switch_lib.c」は、プッシュスイッチの状態を検出する専用の関数が用意されているファイルです。液晶・microSD 基板のプッシュスイッチを使用するときは、プロジェクトに「switch_lib.c」を追加して使用します。
「switch_lib.c」は、「C:¥WorkSpace¥common_r8c38a」フォルダにあります。
このファイルを追加すると、次の関数を実行することができます。

※プロジェクトに switch_lib.c を追加する方法については、「4.2　プロジェクトに lcd_lib.c を追加する」を参考にして追加してください。

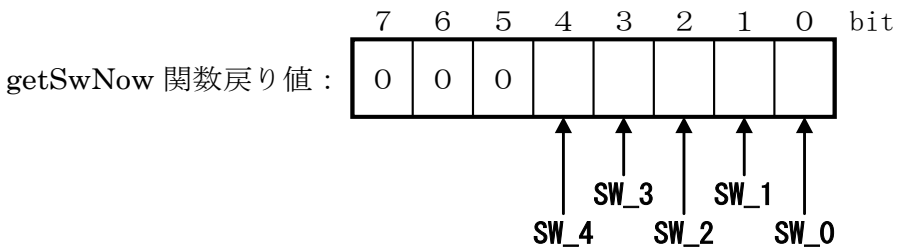
■initSwitch 関数

書式	void initSwitch(void)
内容	プッシュスイッチを初期化します。
引数	なし
戻り値	なし
使用例	asm(" fset I "); /* 全体の割り込み許可　　*/ initSwitch();　/* スイッチ初期化　　*/ **※initSwitch 関数は、全体割り込み許可をした後に実行してください。**

■switchProcess 関数

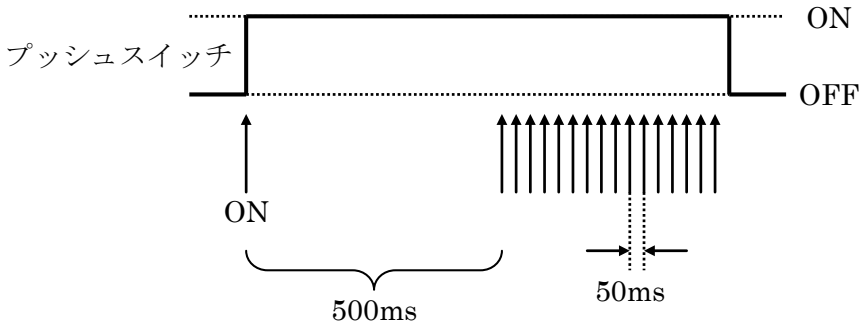
書式	void switchProcess(void)
内容	プッシュスイッチの制御を行います。この関数は割り込み処理などで 1ms ごとに実行してください。
引数	なし
戻り値	なし
使用例	/* 拡張スイッチ用関数(1ms ごとに実行)　*/ switchProcess(); 実際に、プッシュスイッチ処理をしているのは、swichProcess 関数です。この関数は、1ms ごとに実行してください。今回は、タイマ RB の 1ms ごとの割り込み関数内で実行します。

■getSwNow 関数

書式	unsigned char getSwNow(void)
内容	プッシュスイッチの現在値を取得します。
引数	なし
戻り値	符号なし 8bit 幅の値 (unsigned char) 

使用例	if(getSwNow() & SW_0) {
	　SW_0 が"1"なら
	} else {
	　SW_0 が"0"なら
	}
	
	SW_0～SW_4 は、switch_lib.h 内で、
	#define SW_0　(0x01 << 0)
	#define SW_1　(0x01 << 1)
	#define SW_2　(0x01 << 2)
	#define SW_3　(0x01 << 3)
	#define SW_4　(0x01 << 4)
	と宣言しています。getSwNow 関数と SW_x の論理積(AND)を取ることにより、そのプッシュスイッチが押されているか判断することができます。

■getSwFlag 関数

書式	unsigned char getSwFlag(unsigned char flag)
内容	プッシュスイッチのキーリピート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 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。  プッシュ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。

使用例	if(getSwFlag(SW_1)) {
	　SW_1 が ON なら
	} else {
	　SW_1 が OFF なら
	}

6. データフラッシュ制御ライブラリ

「data_flash_lib.c」は、R8C/38A マイコンのデータフラッシュにデータを保存したり、読み込んだりする専用の関数が用意されているファイルです。R8C/38A マイコンのデータフラッシュを使用するときは、プロジェクトに「data_flash_lib.c」を追加して使用します。
「data_flash_lib.c」は、「C:¥WorkSpace¥common_r8c38a」フォルダにあります。
このファイルを追加すると、次の関数を実行することができます。

※プロジェクトに data_flash_lib.c を追加する方法については、「4.2 プロジェクトに lcd_lib.c を追加する」を参考にして追加してください。

6.1 データフラッシュとは

データフラッシュとは、通常プログラムの格納用として使用されるフラッシュメモリとは別に、主にデータの格納用として使用するフラッシュメモリのことです。
RAM はデータの読み書きを自由に行えますが、電源を切ると内容が消えてしまいます。ROM は一度書き込んだ内容は消去できませんが、電源を切っても内容は消えません。この 2 種類の要素を兼ね備えたメモリが、フラッシュメモリです。
フラッシュメモリ(別名:フラッシュ EEPROM)は、データの消去・書き込みを自由に行うことができ、電源を切っても内容が消えない半導体メモリで、従来の EEPROM を発展させたものです。EEPROM は、1 バイト単位でデータの書き換えができたのに対し、データフラッシュはブロック単位で読み書きを行います。データフラッシュにデータを書き込む場合は、あらかじめデータを消去(イレーズ)してから書き込みを行います。ブロック単位で書き込みを行うため、速度が EEPROM よりも速く、大量のデータを格納する用途に優れています。

6.2 data_flash_lib.c ファイルで使用できる関数

■readDataFlash 関数

書式	void readDataFlash(unsigned int r_address, signed char *w_address, int count)
内容	データフラッシュから、データを読み込みます。
引数	unsigned int　　読み込み元アドレス 0x3000～0x3fff signed char*　　読み込み先アドレス int　　読み込むデータ数
戻り値	なし
使用例	signed char　　data_buff[32];　　/* データフラッシュ用保管エリア */ readDataFlash(0x3000, data_buff, 32); /* 0x3000 番地から、data_buff 配列に、32 個のデータを読み込みます　　*/

■writeDataFlash 関数

書式	int writeDataFlash(unsigned int w_address, signed char *r_address, int count)
内容	データフラッシュに、データを書き込みます。
引数	unsigned int　書き込み先アドレス 0x3000～0x3fff signed char*　書き込み元アドレス int　書き込みデータ数
戻り値	1:エラーなし　0:エラーあり
使用例	signed char　data_buff[32];　/* データフラッシュ用保管エリア */ writeDataFlash(0x3000, data_buff, 32); /* データフラッシュの 0x3000 番地に data_buff 配列のデータを 32 個分、書き込みます */
注意点	●書き込み先アドレスについて データフラッシュは、ブロックA(0x3000～0x33ff 番地）、ブロックB(0x3400～0x37ff 番地）、ブロックC(0x3800～0x3bff 番地）、ブロックD(0x3c00～0x3fff 番地）に分かれています。これらのブロックをまたいだ書き込みはできません。例えば、0x33f0 番地から32 個のデータを書き込むとき、0x33ff 番地の次は、0x3400 番地になります。ブロックは、A からB に変わります。これはできません。 例）writeDataFlash(0x33f0, data_buff, 32); /* これはできない */ ●書き込む前のイレーズについて 書き込み先アドレスのデータは、0xff（イレーズ状態）でなければ書き込みできません。イレーズされていないときは、blockEraseDataFlash 関数でイレーズしてください。ただし、イレーズはブロック全体に対して行われます。例えば、0x3000 番地に 1 バイトデータを書き込むとき、ブロック A(0x3000～0x33ff 番地）のデータがすべてイレーズされます。データを残しておきたい場合は、いったんすべてのデータを RAM に読み込んで、今回書き込みたいデータと合わせて書き込むようにしてください。

■blockEraseDataFlash 関数

書式	int blockEraseDataFlash(unsigned int address)
内容	ブロック単位でイレーズします。
引数	unsigned int　イレーズ先ブロックのアドレス ブロック A 0x3000～0x33ff ブロック B 0x3400～0x37ff ブロック C 0x3800～0x3bff ブロック D 0x3c00～0x3fff
戻り値	1:エラーなし　0:エラーあり
使用例	/* ブロック A イレーズします */ blockEraseDataFlash(0x3000); 0x3000～0x33ff のどの値を入れてもブロック A 全体がイレーズされます。 **※イレーズとは、0xff で埋めることです。**

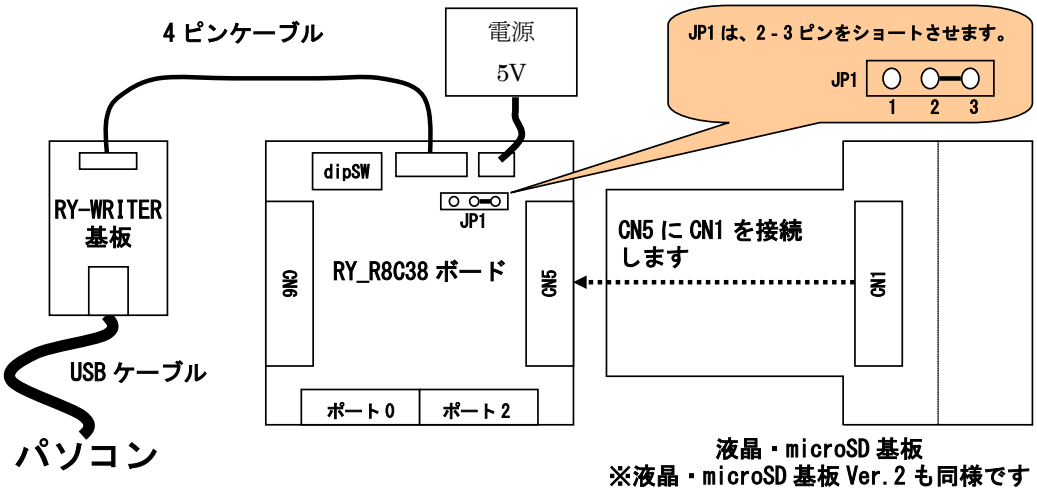
7. プロジェクト「lcd01_38a」 液晶の使い方

7.1 概要

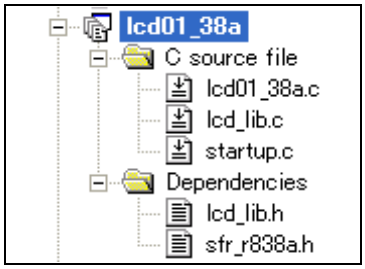
液晶に文字を表示します。文字だけでなく、cnt1 変数の値を表示します。

7.2 接続

・RY_R8C38 ボードの CN5(ポート 3、ポート 5、ポート 6)と液晶・microSD 基板の CN1 のコネクタを重ねて接続します。
・RY_R8C38 ボードとパソコン間を RY-WRITER 基板、USB ケーブル、4 ピンケーブルで接続します。



7.3 プロジェクトの構成



	ファイル名	内容
1	lcd01_38a.c	実際に制御するプログラムが書かれています。R8C/38A マイコンの内蔵周辺機能(SFR)の初期化も行います。 ファイルの位置→C:¥WorkSpace¥kit12lcd_38a¥lcd01_38a¥lcd01_38a.c
2	lcd_lib.c	液晶制御ライブラリです。液晶を使用する場合は、このファイルを追加します。 ファイルの位置→C:¥WorkSpace¥common_r8c38a¥lcd_lib.c
3	startup.c	固定割り込みベクタアドレスの設定、スタートアッププログラム、RAM の初期化(初期値のないグローバル変数、初期値のあるグローバル変数の設定)などを行います。 ファイルの位置→C:¥WorkSpace¥kit12lcd_38a¥lcd01_38a¥startup.c
4	lcd_lib.h	液晶制御ライブラリのヘッダファイルです。 ファイルの位置→C:¥WorkSpace¥common_r8c38a¥lcd_lib.h
5	sfr_r838a.h	R8C/38A マイコンの内蔵周辺機能を制御するためのレジスタ(Special Function Register)を定義したファイルです。 ファイルの位置→C:¥WorkSpace¥common_r8c38a¥sfr_r838a.h

7.4 プログラム「lcd01_38a.c」

プログラムのゴシック体部分が、液晶制御で追加した部分です。

```
 1 :  /***********************************************************************/
 2 :  /* 対象マイコン R8C/38A                                                */
 3 :  /* ﾌｧｲﾙ内容     液晶・microSD基板　制御プログラム　その1               */
 4 :  /* バージョン   Ver.1.00                                               */
 5 :  /* Date         2011.04.01                                             */
 6 :  /* Copyright    ジャパンマイコンカーラリー実行委員会                   */
 7 :  /***********************************************************************/
 8 :  /*
 9 :  本ワークペースは、RY_R8C38ボードで液晶・microSD基板を
10 :  制御するプログラムです。
11 :  本プログラムでは、液晶を制御します。
12 :
13 :  */
14 :
15 :  /*=====================================*/
16 :  /* インクルード                        */
17 :  /*=====================================*/
18 :  #include <stdio.h>
19 :  #include "sfr_r838a.h"                 /* R8C/38A SFRの定義ファイル    */
20 :  #include "lcd_lib.h"                   /* LCD表示用追加                */
21 :
22 :  /*=====================================*/
23 :  /* シンボル定義                        */
24 :  /*=====================================*/
25 :
26 :  /*=====================================*/
27 :  /* プロトタイプ宣言                    */
28 :  /*=====================================*/
29 :  void init( void );
30 :
```

```
 31 :  /*====================================*/
 32 :  /* グローバル変数の宣言               */
 33 :  /*====================================*/
 34 :  unsigned long   cnt1;                  /* 1msごとに+1                */
 35 :
 36 :  /************************************************************************/
 37 :  /* メインプログラム                                                     */
 38 :  /************************************************************************/
 39 :  void main( void )
 40 :  {
 41 :      /* マイコン機能の初期化 */
 42 :      init();                           /* 初期化                       */
 43 :      asm(" fset I ");                  /* 全体の割り込み許可           */
 44 :      initLcd();                        /* LCD初期化                    */
 45 :
 46 :      lcdPosition( 0, 0 );
 47 :      lcdPrintf( "LCD/microSD PCB " );
 48 :      while( 1 ) {
 49 :          lcdPosition( 0, 1 );
 50 :          lcdPrintf( "time = %08ld ", cnt1 );
 51 :      }
 52 :  }
 53 :
 54 :  /************************************************************************/
 55 :  /* R8C/38A スペシャルファンクションレジスタ(SFR)の初期化                */
 56 :  /************************************************************************/
 57 :  void init( void )
 58 :  {
 59 :      int i;
 60 :
 61 :      /* クロックをXINクロック(20MHz)に変更 */
 62 :      prc0  = 1;                        /* プロテクト解除               */
 63 :      cm13  = 1;                        /* P4_6,P4_7をXIN-XOUT端子にする*/
 64 :      cm05  = 0;                        /* XINクロック発振              */
 65 :      for(i=0; i<50; i++ );             /* 安定するまで少し待つ(約10ms) */
 66 :      ocd2  = 0;                        /* システムクロックをXINにする  */
 67 :      prc0  = 0;                        /* プロテクトON                 */
 68 :
 69 :      /* ポートの入出力設定 */
 70 :      prc2 = 1;                         /* PD0のプロテクト解除          */
 71 :      pd0 = 0x00;                       /* 7-0:センサ基板Ver.5          */
 72 :      pd1 = 0xd0;                       /* 5:RXD0 4:TXD0 3-0:DIP SW     */
 73 :      p2  = 0xc0;
 74 :      pd2 = 0xfe;                       /* 7-0:モータドライブ基板Ver.5  */
 75 :      pd3 = 0xff;                       /*                              */
 76 :      p4  = 0x20;                       /* P4_5のLED:初期は点灯         */
 77 :      pd4 = 0xb8;                       /* 7:XOUT 6:XIN 5:LED 2:VREF    */
 78 :      pd5 = 0x7f;                       /* 7-0:LCD/microSD基板          */
 79 :      pd6 = 0xef;                       /* 4-0:LCD/microSD基板          */
 80 :      pd7 = 0xff;                       /*                              */
 81 :      pd8 = 0xff;                       /*                              */
 82 :      pd9 = 0x3f;                       /*                              */
 83 :      pur0 = 0x04;                      /* P1_3～P1_0のプルアップON     */
 84 :
 85 :      /* タイマRBの設定 */
 86 :      /* 割り込み周期 = 1 / 20[MHz]   * (TRBPRE+1) * (TRBPR+1)
 87 :                      = 1 / (20*10^6) * 200        * 100
 88 :                      = 0.001[s] = 1[ms]
 89 :      */
 90 :      trbmr  = 0x00;                    /* 動作モード、分周比設定       */
 91 :      trbpre = 200-1;                   /* プリスケーラレジスタ         */
 92 :      trbpr  = 100-1;                   /* プライマリレジスタ           */
 93 :      trbic  = 0x07;                    /* 割り込み優先レベル設定       */
 94 :      trbcr  = 0x01;                    /* カウント開始                 */
 95 :  }
 96 :
 97 :  /************************************************************************/
 98 :  /* タイマRB 割り込み処理                                                */
 99 :  /************************************************************************/
100 :  #pragma interrupt intTRB(vect=24)
101 :  void intTRB( void )
102 :  {
103 :      cnt1++;
104 :
105 :      /* LCD表示処理用関数(1msごとに実行)     */
106 :      lcdShowProcess();
107 :  }
108 :
109 :  /************************************************************************/
110 :  /* end of file                                                          */
111 :  /************************************************************************/
```

7.5 プログラムの解説

7.5.1 ヘッダファイルの取り込み

```
15 :  /*====================================*/
16 :  /* インクルード                       */
17 :  /*====================================*/
18 :  #include <stdio.h>
19 :  #include "sfr_r838a.h"                /* R8C/38A SFRの定義ファイル    */
20 :  #include "lcd_lib.h"                  /* LCD表示用追加                */
```

「lcd_lib.h」は、液晶を制御するためのプログラム「lcd_lib.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ダファイルです。

7.5.2 ポートの入出力設定

```
69 :      /* ポートの入出力設定 */
70 :      prc2 = 1;                          /* PD0のプロテクト解除          */
71 :      pd0 = 0x00;                        /* 7-0:センサ基板Ver.5          */
72 :      pd1 = 0xd0;                        /* 5:RXD0 4:TXD0 3-0:DIP SW     */
73 :      p2  = 0xc0;
74 :      pd2 = 0xfe;                        /* 7-0:モータドライブ基板Ver.5  */
75 :      pd3 = 0xff;                        /*                              */
76 :      p4  = 0x20;                        /* P4_5のLED:初期は点灯         */
77 :      pd4 = 0xb8;                        /* 7:XOUT 6:XIN 5:LED 2:VREF    */
78 :      pd5 = 0x7f;                        /* 7-0:LCD/microSD基板          */
79 :      pd6 = 0xef;                        /* 4-0:LCD/microSD基板          */
80 :      pd7 = 0xff;                        /*                              */
81 :      pd8 = 0xff;                        /*                              */
82 :      pd9 = 0x3f;                        /*                              */
83 :      pur0 = 0x04;                       /* P1_3～P1_0のプルアップON     */
```

※79 行目の PD6 については、「液晶・microSD 基板 kit12_38a プログラム解説マニュアルデータ解析(microSD)編(R8C/38A 版)」を参照してください。

液晶・microSD 基板の液晶とプッシュスイッチは、RY_R8C38 ボードのポート 5 を使用しています。ポート 5 のポート表を下表に示します。

番号	ポート	液晶を制御する場合		プッシュスイッチを制御する場合	
		信号名	マイコンから見た方向	信号名	マイコンから見た方向
1	+5V	-	-	-	-
2	P5_7	-	入力	SW_4	入力
3	P5_6	E	出力	-	出力
4	P5_5	RW	出力	-	出力
5	P5_4	RS	出力	-	出力
6	P5_3	D3	入出力	SW_3	入力
7	P5_2	D2	入出力	SW_2	入力
8	P5_1	D1	入出力	SW_1	入力
9	P5_0	D0	入出力	SW_0	入力
10	GND	-	-	-	-

P5_3～P5_0 は、液晶とプッシュスイッチの信号が重なっています。RY_R8C38 ボードは、液晶にデータを出力する場合は出力設定に、プッシュスイッチが押されたかどうかをチェックする場合は入力設定に切り替えて制御をしています。

液晶部を使用する場合の PD5 の入出力設定を下表に示します。また、この設定を初期設定値にします。

ビット	7	6	5	4	3	2	1	0
ポート 5 の 入出力設定	入力	出力	出力	出力	出力	出力	出力	出力

入力は"0"、出力は"1"を設定します。初期設定値を 16 進数に直すと、0111 1111 → 0x7f となります。

プッシュスイッチ部を使用する場合の PD5 の入出力設定を下表に示します。

ビット	7	6	5	4	3	2	1	0
ポート 5 の 入出力設定	入力	出力	出力	出力	入力	入力	入力	入力

switchProcess 関数を実行したときに、この入出力設定にしてプッシュスイッチの状態を読み込みます。読み込んだ後、液晶の入出力設定に戻します。

7.5.3 初期化

```
41 :    /* マイコン機能の初期化 */
42 :    init();                          /* 初期化                  */
43 :    asm(" fset I ");                 /* 全体の割り込み許可      */
44 :    initLcd();                       /* LCD初期化               */
```

initLcd（液晶の初期化）関数は、割り込みが許可された状態（「asm(" fset I ")」以降）で実行します。

7.5.4 表示データを作る

```
 46 :      lcdPosition( 0, 0 );                ←液晶の0列目0行目にセット
 47 :      lcdPrintf( "LCD/microSD PCB " );
 48 :      while( 1 ) {
 49 :          lcdPosition( 0, 1 );            ←液晶の0列目1行目にセット
 50 :          lcdPrintf( "time = %08ld ", cnt1 );
 51 :      }
```

46 行目	lcdPosition 関数は、lcdPrintf 関数などで液晶へ表示するときの位置を指定しています。
47 行目	液晶の 0 行目に表示する文字列を作ります。
50 行目	液晶の 1 行目に表示する文字列を作ります。 lcdPrintf 関数の引数部分は、「"time = %08ld ", cnt1」となっています。 %08ld（出力変換指定子）の%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。 1…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 が代入されているとすると、 列 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0行: L C D ／ m i c r o S D　P C B 1行: 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 kit12_38a.c では、cnt1 変数を時間計測用に使用しました。液晶で cnt1 の値が変化するのを見ることができます。

7.5.5 液晶表示処理関数

```
 97 : /************************************************************************/
 98 : /* タイマRB 割り込み処理                                                */
 99 : /************************************************************************/
100 : #pragma interrupt intTRB(vect=24)
101 : void intTRB( void )
102 : {
103 :     cnt1++;
104 :
105 :     /* LCD表示処理用関数(1msごとに実行)     */
106 :     lcdShowProcess();
107 : }
```

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。今回は、タイマ RB 割り込み関数が 1ms ごとに実行されているので、この関数内に lcdShowProcess 関数を記述しています。

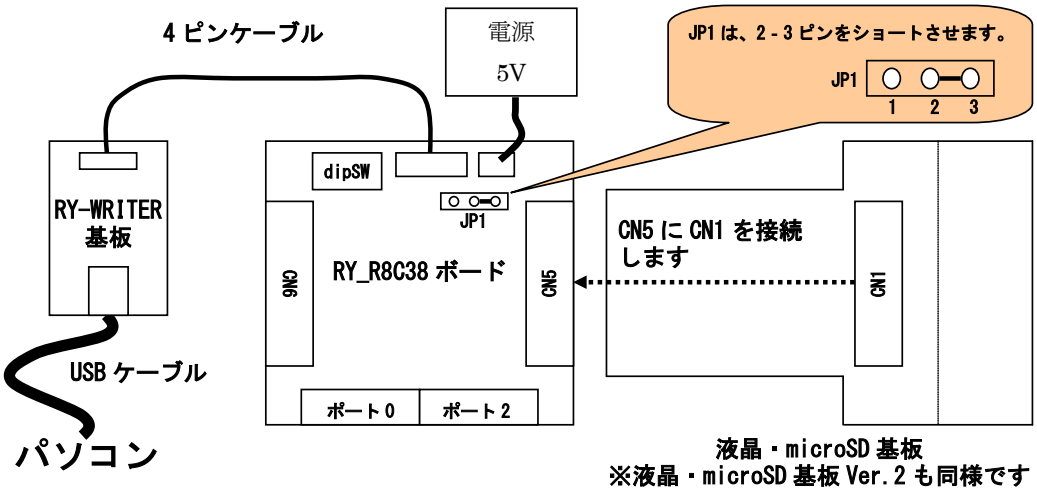
8. プロジェクト「lcd02_38a」 プッシュスイッチの使い方

8.1 概要

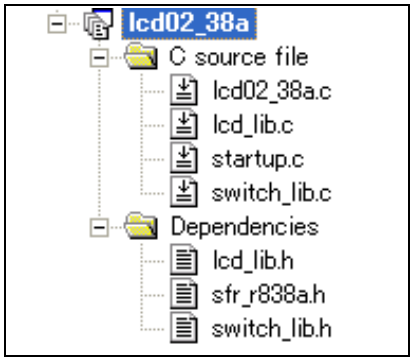
5 個のプッシュスイッチを操作することにより、液晶に表示される数字が+1、-1、+10、-10、クリアされます。

8.2 接続

・RY_R8C38 ボードの CN5(ポート 3、ポート 5、ポート 6)と液晶・microSD 基板の CN1 のコネクタを重ねて接続します。
・RY_R8C38 ボードとパソコン間を RY-WRITER 基板、USB ケーブル、4 ピンケーブルで接続します。



8.3 プロジェクトの構成



	ファイル名	内容
1	lcd02_38a.c	実際に制御するプログラムが書かれています。R8C/38A マイコンの内蔵周辺機能(SFR)の初期化も行います。 ファイルの位置→C:¥WorkSpace¥kit12lcd_38a¥lcd02_38a¥lcd02_38a.c
2	lcd_lib.c	液晶制御ライブラリです。液晶を使用する場合は、このファイルを追加します。 ファイルの位置→C:¥WorkSpace¥common_r8c38a¥lcd_lib.c
3	startup.c	固定割り込みベクタアドレスの設定、スタートアッププログラム、RAM の初期化(初期値のないグローバル変数、初期値のあるグローバル変数の設定)などを行います。 ファイルの位置→C:¥WorkSpace¥kit12lcd_38a¥lcd02_38a¥startup.c
4	switch_lib.c	プッシュスイッチ制御ライブラリです。プッシュスイッチを使用する場合は、このファイルを追加します。 ファイルの位置→C:¥WorkSpace¥common_r8c38a¥switch_lib.c
5	lcd_lib.h	液晶制御ライブラリのヘッダファイルです。 ファイルの位置→C:¥WorkSpace¥common_r8c38a¥lcd_lib.h
6	sfr_r838a.h	R8C/38A マイコンの内蔵周辺機能を制御するためのレジスタ(Special Function Register)を定義したファイルです。 ファイルの位置→C:¥WorkSpace¥common_r8c38a¥sfr_r838a.h
7	switch_lib.h	プッシュスイッチ制御ライブラリのヘッダファイルです。 ファイルの位置→C:¥WorkSpace¥common_r8c38a¥switch_lib.h

8.4 プログラム「lcd02_38a.c」

プログラムのゴシック体部分が、「lcd01_38a.c」からプッシュスイッチ制御を追加、変更した部分です。

```
 1 :  /***********************************************************************/
 2 :  /* 対象マイコン  R8C/38A                                               */
 3 :  /* ﾌｧｲﾙ内容      液晶・microSD基板　制御プログラム　その2              */
 4 :  /* バージョン    Ver.1.00                                              */
 5 :  /* Date          2011.04.01                                            */
 6 :  /* Copyright     ジャパンマイコンカーラリー実行委員会                  */
 7 :  /***********************************************************************/
 8 :  /*
 9 :  本ワークペースは、RY_R8C38ボードで液晶・microSD基板を
10 :  制御するプログラムです。
11 :  本プログラムでは、液晶とスイッチを制御します。
12 :
13 :  */
14 :
```

```
 15 :  /*====================================*/
 16 :  /* インクルード                       */
 17 :  /*====================================*/
 18 :  #include <stdio.h>
 19 :  #include "sfr_r838a.h"                /* R8C/38A SFRの定義ファイル    */
 20 :  #include "lcd_lib.h"                  /* LCD表示用追加                */
 21 :  #include "switch_lib.h"               /* スイッチ追加                 */
```

中略

```
 37 :  /************************************************************************/
 38 :  /* メインプログラム                                                     */
 39 :  /************************************************************************/
 40 :  void main( void )
 41 :  {
 42 :      int     i;
 43 :
 44 :      /* マイコン機能の初期化 */
 45 :      init();                           /* 初期化                       */
 46 :      asm(" fset I ");                  /* 全体の割り込み許可           */
 47 :      initLcd();                        /* LCD初期化                    */
 48 :      initSwitch();                     /* スイッチ初期化               */
 49 :
 50 :      i = 0;
 51 :      while( 1 ) {
 52 :          /* スイッチ処理 */
 53 :          if( getSwFlag(SW_0) ) {
 54 :              i--;
 55 :              if( i < -10000 ) i = -10000;
 56 :          }
 57 :          if( getSwFlag(SW_1) ) {
 58 :              i++;
 59 :              if( i > 10000 ) i = 10000;
 60 :          }
 61 :          if( getSwFlag(SW_2) ) {
 62 :              i -= 10;
 63 :              if( i < -10000 ) i = -10000;
 64 :          }
 65 :          if( getSwFlag(SW_3) ) {
 66 :              i += 10;
 67 :              if( i > 10000 ) i = 10000;
 68 :          }
 69 :          if( getSwFlag(SW_4) ) {
 70 :              i = 0;
 71 :          }
 72 :
 73 :          /* LCD処理 */
 74 :          lcdPosition( 0, 0 );
 75 :          lcdPrintf( "getSwNow() = %02x", getSwNow() );
 76 :          lcdPosition( 0, 1 );
 77 :          lcdPrintf( "data = %+06d", i );
 78 :      }
 79 :  }
```

中略

```
124 :  /************************************************************************/
125 :  /* タイマRB 割り込み処理                                                */
126 :  /************************************************************************/
127 :  #pragma interrupt intTRB(vect=24)
128 :  void intTRB( void )
129 :  {
130 :      cnt1++;
131 :
132 :      /* 拡張スイッチ用関数(1msごとに実行)    */
133 :      switchProcess();
134 :
135 :      /* LCD表示処理用関数(1msごとに実行)     */
136 :      lcdShowProcess();
137 :  }
```

以下、略

8.5 プログラムの解説

8.5.1 ヘッダファイルの取り込み

```
18 :  #include <stdio.h>
19 :  #include "sfr_r838a.h"              /* R8C/38A SFRの定義ファイル    */
20 :  #include "lcd_lib.h"                /* LCD表示用追加                */
21 :  #include "switch_lib.h"             /* スイッチ追加                 */
```

「switch_lib.h」は、プッシュスイッチを制御するためのプログラム「switch_lib.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ダファイルです。

8.5.2 初期化

```
44 :      /* マイコン機能の初期化 */
45 :      init();                          /* 初期化                       */
46 :      asm(" fset I ");                 /* 全体の割り込み許可           */
47 :      initLcd();                       /* LCD初期化                    */
48 :      initSwitch();                    /* スイッチ初期化               */
```

initSwitch(プッシュスイッチの初期化)関数は、割り込みが許可されている状態(「asm(" fset I ")」以降)で実行します。

8.5.3 プログラムでの使用

```
50 :      i = 0;
51 :      while( 1 ) {
52 :          /* スイッチ処理 */
53 :          if( getSwFlag(SW_0) ) {
54 :              i--;
55 :              if( i < -10000 ) i = -10000;
56 :          }
57 :          if( getSwFlag(SW_1) ) {
58 :              i++;
59 :              if( i > 10000 ) i = 10000;
60 :          }
61 :          if( getSwFlag(SW_2) ) {
62 :              i -= 10;
63 :              if( i < -10000 ) i = -10000;
64 :          }
65 :          if( getSwFlag(SW_3) ) {
66 :              i += 10;
67 :              if( i > 10000 ) i = 10000;
68 :          }
69 :          if( getSwFlag(SW_4) ) {
70 :              i = 0;
71 :          }
72 :
```

```
 73 :          /* LCD処理 */
 74 :          lcdPosition( 0, 0 );
 75 :          lcdPrintf( "getSwNow() = %02x", getSwNow() );
 76 :          lcdPosition( 0, 1 );
 77 :          lcdPrintf( "data = %+06d", i );
 78 :      }
```

53 行目 ～ 71 行目

SW_0～SW_4 を押したとき、i 変数を操作します。キーリピートを使っています。

プッシュスイッチ	内容
SW_0	i 変数を-1 します。
SW_1	i 変数を+1 します。
SW_2	i 変数を-10 します。
SW_3	i 変数を+10 します。
SW_4	i 変数を 0 にします。

74 行目 ～ 77 行目

表示文字列を作っています。ここでは getSwNow 関数で現在のプッシュスイッチ値と、i 変数の値を表示しています。

SW_0 が押されているとき、getSwNow 関数の戻り値は"0x01"です。このときの表示例を下図に示します。(SW_0 を押すと data の値が 1 ずつカウントダウンします。)

行＼列	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	g	e	t	S	w	N	o	w	(	)		=		0	1	
1	d	a	t	a		=		－	0	0	0	0	1			

8.5.4 プッシュスイッチ処理

```
124 : /************************************************************************/
125 : /* タイマRB 割り込み処理                                                */
126 : /************************************************************************/
127 : #pragma interrupt intTRB(vect=24)
128 : void intTRB( void )
129 : {
130 :     cnt1++;
131 :
132 :     /* 拡張スイッチ用関数(1msごとに実行)     */
133 :     switchProcess();
134 :
135 :     /* LCD表示処理用関数(1msごとに実行)      */
136 :     lcdShowProcess();
137 : }
```

133 行目で、タイマ RB 割り込み関数内で 1ms ごとにプッシュ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。

※lcdShowProcess 関数がある場合は、switchProcess 関数を先に実行します。

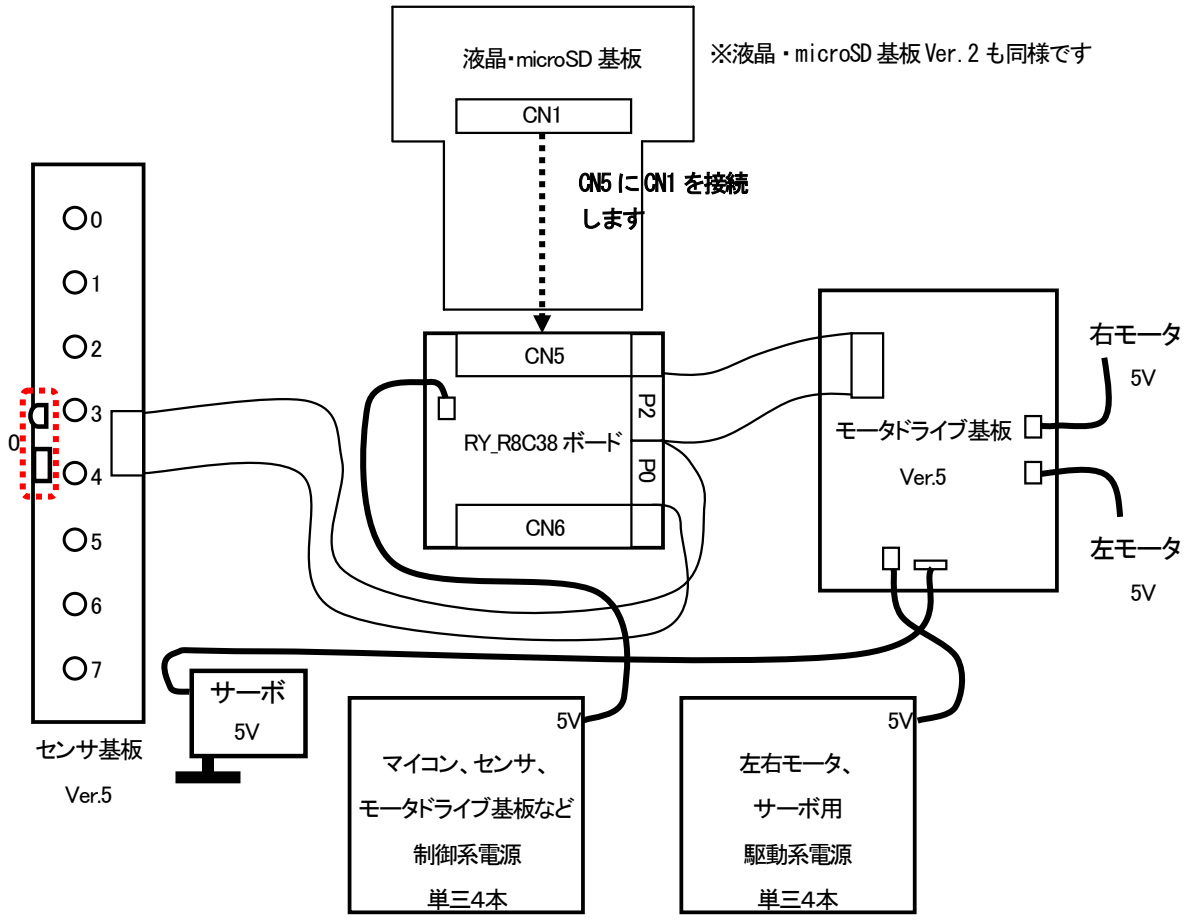
9. プロジェクト「kit12lcd01_38a」 パラメータを液晶で設定する

9.1 概要

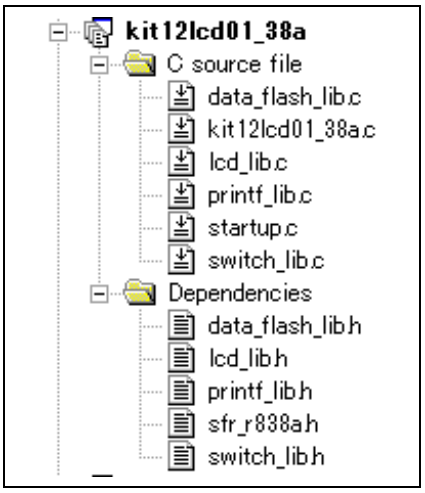
液晶・microSD 基板を使ってパラメータ(PWM 値)を調整できるようにします。今回は、サーボセンタ値、PWM 値(走行 PWM 全体の割合)、大曲げ PWM 値、クランク PWM 値を液晶で調整します。

9.2 接続

マイコンカーキット Ver.5.1 の構成です。LM350 追加セットで電池 8 本を直列に繋いでいる構成でも OK です。
・RY_R8C38 ボードのポート 0 と、マイコンカーキット Ver.5 のセンサ基板 Ver.5 を接続します。
・RY_R8C38 ボードのポート 2 と、マイコンカーキット Ver.5 のモータドライブ基板 Ver.5 を接続します。
・RY_R8C38 ボードの CN5 と液晶・microSD 基板の CN1 を接続します。



9.3 プロジェクトの構成



	ファイル名	内容
1	data_flash_lib.c	データフラッシュ制御ライブラリです。データフラッシュを使用する場合は、このファイルを追加します。 ファイルの位置→C:¥WorkSpace¥common_r8c38a¥data_flash_lib.c
2	kit12lcd01_38a.c	実際に制御するプログラムが書かれています。R8C/38A マイコンの内蔵周辺機能(SFR)の初期化も行います。 ファイルの位置→C:¥WorkSpace¥kit12lcd_38a¥kit12lcd01_38a¥kit12lcd01_38a.c
3	lcd_lib.c	液晶制御ライブラリです。液晶を使用する場合は、このファイルを追加します。 ファイルの位置→C:¥WorkSpace¥common_r8c38a¥lcd_lib.c
4	startup.c	固定割り込みベクタアドレスの設定、スタートアッププログラム、RAM の初期化(初期値のないグローバル変数、初期値のあるグローバル変数の設定)などを行います。 ファイルの位置→C:¥WorkSpace¥kit12lcd_38a¥kit12lcd01_38a¥startup.c
5	switch_lib.c	プッシュスイッチ制御ライブラリです。プッシュスイッチを使用する場合は、このファイルを追加します。 ファイルの位置→C:¥WorkSpace¥common_r8c38a¥switch_lib.c
6	data_flash_lib.h	データフラッシュ制御ライブラリのヘッダファイルです。 ファイルの位置→C:¥WorkSpace¥common_r8c38a¥data_flash_lib.h
7	lcd_lib.h	液晶制御ライブラリのヘッダファイルです。 ファイルの位置→C:¥WorkSpace¥common_r8c38a¥lcd_lib.h
8	sfr_r838a.h	R8C/38A マイコンの内蔵周辺機能を制御するためのレジスタ（Special Function Register）を定義したファイルです。 ファイルの位置→C:¥WorkSpace¥common_r8c38a¥sfr_r838a.h
9	switch_lib.h	プッシュスイッチ制御ライブラリのヘッダファイルです。 ファイルの位置→C:¥WorkSpace¥common_r8c38a¥switch_lib.h

9.4 プログラム「kit12lcd01_38a.c」

プログラムのゴシック体部分が、「kit12_38a.c」から液晶制御、プッシュスイッチ制御、パラメータ調整をできるように追加した部分です。

```
/************************************************************************/
/* 対象マイコン R8C/38A                                                 */
/* ﾌｧｲﾙ内容     液晶・microSD基板　液晶を使用したトレースプログラム     */
/* バージョン   Ver.1.00                                                */
/* Date         2013.05.22                                              */
/* Copyright    ジャパンマイコンカーラリー実行委員会                    */
/************************************************************************/
/*
このプログラムは、下記基板に対応しています。
・RY_R8C38ボード
・モータドライブ基板Ver.5
・センサ基板Ver.5
*/

/*====================================*/
/* インクルード                       */
/*====================================*/
#include <stdio.h>
#include "sfr_r838a.h"                  /* R8C/38A SFRの定義ファイル    */
#include "lcd_lib.h"                    /* LCD表示用追加                */
#include "switch_lib.h"                 /* スイッチ追加                 */
#include "data_flash_lib.h"             /* データフラッシュライブラリ   */

/*====================================*/
/* シンボル定義                       */
/*====================================*/

/* 定数設定 */
#define PWM_CYCLE       39999           /* モータPWMの周期              */
#define SERVO_CENTER    3750            /* サーボのセンタ値             */
#define HANDLE_STEP     22              /* 1°分の値                    */

/* マスク値設定 ×：マスクあり(無効)　○：マスク無し(有効) */
#define MASK2_2         0x66            /* ××○○××○×                  */
#define MASK2_0         0x60            /* ××○○××××                  */
#define MASK0_2         0x06            /* ××××××○×                   */
#define MASK3_3         0xe7            /* ○○○××○○○                   */
#define MASK0_3         0x07            /* ××××××○○                    */
#define MASK3_0         0xe0            /* ○○○××××××                   */
#define MASK4_0         0xf0            /* ○○○○××××                    */
#define MASK0_4         0x0f            /* ××××○○○○                    */
#define MASK4_4         0xff            /* ○○○○○○○○                    */

/* DataFlash関連 */
#define DF_ADDR_START   0x3000          /* 書き込み開始アドレス         */
#define DF_ADDR_END     0x33ff          /* 書き込み終了アドレス         */

#define DF_PARA_SIZE    32              /* DataFlashパラメータ数        */

#define DF_CHECK        0x00            /* DataFlashチェック            */
#define DF_SERVO1       0x01            /* サーボセンタ値               */
#define DF_SERVO2       0x02            /* サーボセンタ値               */
#define DF_PWM          0x03            /* PWM値                        */
#define DF_CURVE_PWM    0x04            /* 大曲げPWM値                  */
#define DF_CRANK_PWM    0x05            /* クランクPWM値                */

/*====================================*/
/* プロトタイプ宣言                   */
/*====================================*/
void init( void );
void timer( unsigned long timer_set );
int check_crossline( void );
int check_rightline( void );
int check_leftline( void );
unsigned char sensor_inp( unsigned char mask );
unsigned char dipsw_get( void );
unsigned char pushsw_get( void );
unsigned char startbar_get( void );
void led_out( unsigned char led );
void motor( int accele_l, int accele_r );
void handle( int angle );
int diff( int pwm );
void readDataFlashParameter( void );
void writeDataFlashParameter( void );
int lcdProcess( void );

```

```
 77 :  /*====================================*/
 78 :  /* グローバル変数の宣言               */
 79 :  /*====================================*/
 80 :  unsigned long   cnt0;                  /* timer関数用                */
 81 :  unsigned long   cnt1;                  /* main内で使用               */
 82 :  unsigned long   cnt_lcd;               /* LCD処理で使用              */
 83 :  int             pattern;               /* パターン番号               */
 84 :  int             servo_center;          /* サーボセンタ値             */
 85 :
 86 :  /* DataFlash関係 */
 87 :  signed char     data_buff[ DF_PARA_SIZE ];
 88 :
 89 :  /* LCD関連 */
 90 :  int             lcd_pattern = 2;
 91 :
 92 :  /* 角度関連 */
 93 :  int             angle_buff;            /* 現在ハンドル角度保持用    */
 94 :
 95 :  const int revolution_difference[] = {   /* 角度から内輪、外輪回転差計算 */
 96 :      100, 99, 97, 96, 95,
 97 :      93, 92, 91, 89, 88,
 98 :      87, 86, 84, 83, 82,
 99 :      81, 79, 78, 77, 76,
100 :      75, 73, 72, 71, 70,
101 :      69, 67, 66, 65, 64,
102 :      62, 61, 60, 59, 58,
103 :      56, 55, 54, 52, 51,
104 :      50, 48, 47, 46, 44,
105 :      43 };
106 :
107 :  /************************************************************************/
108 :  /* メインプログラム                                                     */
109 :  /************************************************************************/
110 :  void main( void )
111 :  {
112 :      int     i;
113 :
114 :      /* マイコン機能の初期化 */
115 :      init();                             /* 初期化                     */
116 :      asm(" fset I ");                    /* 全体の割り込み許可         */
117 :      initLcd();                          /* LCD初期化                  */
118 :      initSwitch();                       /* スイッチ初期化             */
119 :
120 :      readDataFlashParameter();           /* DataFlashパラメータ読み込み  */
121 :      servo_center  = (unsigned char)data_buff[DF_SERV01] * 0x100;
122 :      servo_center |= (unsigned char)data_buff[DF_SERV02];
123 :
124 :      /* マイコンカーの状態初期化 */
125 :      handle( 0 );
126 :      motor( 0, 0 );
127 :
128 :      while( 1 ) {
129 :
130 :      // LCD表示、パラメータ設定処理
131 :      lcdProcess();
132 :
133 :      switch( pattern ) {
134 :
135 :      /*****************************************************************
136 :      パターンについて
137 :       0：スイッチ入力待ち
138 :       1：スタートバーが開いたかチェック
139 :      11：通常トレース
140 :      12：右へ大曲げの終わりのチェック
141 :      13：左へ大曲げの終わりのチェック
142 :      21：クロスライン検出時の処理
143 :      22：クロスラインを読み飛ばす
144 :      23：クロスライン後のトレース、クランク検出
145 :      31：左クランククリア処理　安定するまで少し待つ
146 :      32：左クランククリア処理　曲げ終わりのチェック
147 :      41：右クランククリア処理　安定するまで少し待つ
148 :      42：右クランククリア処理　曲げ終わりのチェック
149 :      51：右ハーフライン検出時の処理
150 :      52：右ハーフラインを読み飛ばす
151 :      53：右ハーフライン後のトレース、レーンチェンジ
152 :      54：右レーンチェンジ終了のチェック
153 :      61：左ハーフライン検出時の処理
154 :      62：左ハーフラインを読み飛ばす
155 :      63：左ハーフライン後のトレース、レーンチェンジ
156 :      64：左レーンチェンジ終了のチェック
157 :      *****************************************************************/
158 :
159 :      case 0:
160 :          /* スイッチ入力待ち */
161 :          if( pushsw_get() ) {
162 :              // パラメータ保存
163 :              writeDataFlashParameter();
164 :              pattern = 1;
165 :              cnt1 = 0;
166 :              break;
```

```
167 :          }
168 :          if( cnt1 < 100 ) {              /* LED点滅処理                  */
169 :              led_out( 0x1 );
170 :          } else if( cnt1 < 200 ) {
171 :              led_out( 0x2 );
172 :          } else {
173 :              cnt1 = 0;
174 :          }
175 :          break;
```

中略

```
271 :      case 12:
272 :          /* 右へ大曲げの終わりのチェック */
273 :          if( check_crossline() ) {       /* 大曲げ中もクロスラインチェック */
274 :              pattern = 21;
275 :              break;
276 :          }
277 :          if( check_rightline() ) {       /* 右ハーフラインチェック       */
278 :              pattern = 51;
279 :              break;
280 :          }
281 :          if( check_leftline() ) {        /* 左ハーフラインチェック       */
282 :              pattern = 61;
283 :              break;
284 :          }
285 :          i = data_buff[DF_CURVE_PWM];
286 :          switch( sensor_inp(MASK3_3) ) {
287 :          case 0x06:
288 :              pattern = 11;
289 :              break;
290 :          case 0x03:
291 :              handle( 20 );
292 :              motor( i, diff(i) );
293 :              break;
294 :          case 0x81:
295 :              handle( 25 );
296 :              motor( i, diff(i) );
297 :              break;
298 :          case 0xc1:
299 :          case 0xc0:
300 :              handle( 30 );
301 :              motor( i, diff(i) );
302 :              break;
303 :          case 0x60:
304 :              handle( 35 );
305 :              motor( 0, 0 );
306 :              break;
307 :          }
308 :          break;
309 :
310 :      case 13:
311 :          /* 左へ大曲げの終わりのチェック */
312 :          if( check_crossline() ) {       /* 大曲げ中もクロスラインチェック */
313 :              pattern = 21;
314 :              break;
315 :          }
316 :          if( check_rightline() ) {       /* 右ハーフラインチェック       */
317 :              pattern = 51;
318 :              break;
319 :          }
320 :          if( check_leftline() ) {        /* 左ハーフラインチェック       */
321 :              pattern = 61;
322 :              break;
323 :          }
324 :          i = data_buff[DF_CURVE_PWM];
325 :          switch( sensor_inp(MASK3_3) ) {
326 :          case 0x60:
327 :              pattern = 11;
328 :              break;
329 :          case 0xc0:
330 :              handle( -20 );
331 :              motor( diff(i), i );
332 :              break;
333 :          case 0x81:
334 :              handle( -25 );
335 :              motor( diff(i), i );
336 :              break;
337 :          case 0x83:
338 :          case 0x03:
339 :              handle( -30 );
340 :              motor( diff(i), i );
341 :              break;
342 :          case 0x06:
343 :              handle( -35 );
344 :              motor( 0, 0 );
345 :              break;
346 :          }
347 :          break;
348 :
```

```
349 :      case 21:
350 :          /* クロスライン検出時の処理 */
351 :          led_out( 0x3 );
352 :          handle( 0 );
353 :          motor( 0 ,0 );
354 :          pattern = 22;
355 :          cnt1 = 0;
356 :          break;
357 :
358 :      case 22:
359 :          /* クロスラインを読み飛ばす */
360 :          if( cnt1 > 100 ) {
361 :              pattern = 23;
362 :              cnt1 = 0;
363 :          }
364 :          break;
365 :
366 :      case 23:
367 :          /* クロスライン後のトレース、クランク検出 */
368 :          if( sensor_inp(MASK4_4)==0xf8 ) {
369 :              /* 左クランクと判断→左クランククリア処理へ */
370 :              led_out( 0x1 );
371 :              handle( -38 );
372 :              motor( 10 ,50 );
373 :              pattern = 31;
374 :              cnt1 = 0;
375 :              break;
376 :          }
377 :          if( sensor_inp(MASK4_4)==0x1f ) {
378 :              /* 右クランクと判断→右クランククリア処理へ */
379 :              led_out( 0x2 );
380 :              handle( 38 );
381 :              motor( 50 ,10 );
382 :              pattern = 41;
383 :              cnt1 = 0;
384 :              break;
385 :          }
386 :          i = data_buff[DF_CRANK_PWM];
387 :          switch( sensor_inp(MASK3_3) ) {
388 :              case 0x00:
389 :                  /* センター→まっすぐ */
390 :                  handle( 0 );
391 :                  motor( i ,i );
392 :                  break;
393 :              case 0x04:
394 :              case 0x06:
395 :              case 0x07:
396 :              case 0x03:
397 :                  /* 左寄り→右曲げ */
398 :                  handle( 8 );
399 :                  motor( i ,diff(i) );
400 :                  break;
401 :              case 0x20:
402 :              case 0x60:
403 :              case 0xe0:
404 :              case 0xc0:
405 :                  /* 右寄り→左曲げ */
406 :                  handle( -8 );
407 :                  motor( diff(i) ,i );
408 :                  break;
409 :          }
410 :          break;
411 :
412 :      case 31:
413 :          /* 左クランククリア処理　安定するまで少し待つ */
414 :          if( cnt1 > 200 ) {
415 :              pattern = 32;
416 :              cnt1 = 0;
417 :          }
418 :          break;
419 :
420 :      case 32:
421 :          /* 左クランククリア処理　曲げ終わりのチェック */
422 :          if( sensor_inp(MASK3_3) == 0x60 ) {
423 :              led_out( 0x0 );
424 :              pattern = 11;
425 :              cnt1 = 0;
426 :          }
427 :          break;
428 :
429 :      case 41:
430 :          /* 右クランククリア処理　安定するまで少し待つ */
431 :          if( cnt1 > 200 ) {
432 :              pattern = 42;
433 :              cnt1 = 0;
434 :          }
435 :          break;
436 :
```

```
437 :      case 42:
438 :          /* 右クランククリア処理　曲げ終わりのチェック */
439 :          if( sensor_inp(MASK3_3) == 0x06 ) {
440 :              led_out( 0x0 );
441 :              pattern = 11;
442 :              cnt1 = 0;
443 :          }
444 :          break;
445 :
446 :      case 51:
447 :          /* 右ハーフライン検出時の処理 */
448 :          led_out( 0x2 );
449 :          handle( 0 );
450 :          motor( 0 ,0 );
451 :          pattern = 52;
452 :          cnt1 = 0;
453 :          break;
454 :
455 :      case 52:
456 :          /* 右ハーフラインを読み飛ばす */
457 :          if( cnt1 > 100 ) {
458 :              pattern = 53;
459 :              cnt1 = 0;
460 :          }
461 :          break;
462 :
463 :      case 53:
464 :          /* 右ハーフライン後のトレース、レーンチェンジ */
465 :          if( sensor_inp(MASK4_4) == 0x00 ) {
466 :              handle( 15 );
467 :              motor( 40 ,diff(40) );
468 :              pattern = 54;
469 :              cnt1 = 0;
470 :              break;
471 :          }
472 :          i = data_buff[DF_CRANK_PWM];
473 :          switch( sensor_inp(MASK3_3) ) {
474 :              case 0x00:
475 :                  /* センター→まっすぐ */
476 :                  handle( 0 );
477 :                  motor( i ,i );
478 :                  break;
479 :              case 0x04:
480 :              case 0x06:
481 :              case 0x07:
482 :              case 0x03:
483 :                  /* 左寄り→右曲げ */
484 :                  handle( 8 );
485 :                  motor( i ,diff(i) );
486 :                  break;
487 :              case 0x20:
488 :              case 0x60:
489 :              case 0xe0:
490 :              case 0xc0:
491 :                  /* 右寄り→左曲げ */
492 :                  handle( -8 );
493 :                  motor( diff(i) ,i );
494 :                  break;
495 :              default:
496 :                  break;
497 :          }
498 :          break;
499 :
500 :      case 54:
501 :          /* 右レーンチェンジ終了のチェック */
502 :          if( sensor_inp( MASK4_4 ) == 0x3c ) {
503 :              led_out( 0x0 );
504 :              pattern = 11;
505 :              cnt1 = 0;
506 :          }
507 :          break;
508 :
509 :      case 61:
510 :          /* 左ハーフライン検出時の処理 */
511 :          led_out( 0x1 );
512 :          handle( 0 );
513 :          motor( 0 ,0 );
514 :          pattern = 62;
515 :          cnt1 = 0;
516 :          break;
517 :
518 :      case 62:
519 :          /* 左ハーフラインを読み飛ばす */
520 :          if( cnt1 > 100 ) {
521 :              pattern = 63;
522 :              cnt1 = 0;
523 :          }
524 :          break;
525 :
```

```
526 :      case 63:
527 :          /* 左ハーフライン後のトレース、レーンチェンジ */
528 :          if( sensor_inp(MASK4_4) == 0x00 ) {
529 :              handle( -15 );
530 :              motor( diff(40) ,40 );
531 :              pattern = 64;
532 :              cnt1 = 0;
533 :              break;
534 :          }
535 :          i = data_buff[DF_CRANK_PWM];
536 :          switch( sensor_inp(MASK3_3) ) {
537 :              case 0x00:
538 :                  /* センター→まっすぐ */
539 :                  handle( 0 );
540 :                  motor( i ,i );
541 :                  break;
542 :              case 0x04:
543 :              case 0x06:
544 :              case 0x07:
545 :              case 0x03:
546 :                  /* 左寄り→右曲げ */
547 :                  handle( 8 );
548 :                  motor( i ,diff(i) );
549 :                  break;
550 :              case 0x20:
551 :              case 0x60:
552 :              case 0xe0:
553 :              case 0xc0:
554 :                  /* 右寄り→左曲げ */
555 :                  handle( -8 );
556 :                  motor( diff(i) ,i );
557 :                  break;
558 :              default:
559 :                  break;
560 :          }
561 :          break;
562 :
563 :      case 64:
564 :          /* 左レーンチェンジ終了のチェック */
565 :          if( sensor_inp( MASK4_4 ) == 0x3c ) {
566 :              led_out( 0x0 );
567 :              pattern = 11;
568 :              cnt1 = 0;
569 :          }
570 :          break;
571 :
572 :      default:
573 :          /* どれでもない場合は待機状態に戻す */
574 :          pattern = 0;
575 :          break;
576 :      }
577 :      }
578 :  }
```

中略

```
640 :  /************************************************************************/
641 :  /* タイマRB 割り込み処理                                                */
642 :  /************************************************************************/
643 :  #pragma interrupt intTRB(vect=24)
644 :  void intTRB( void )
645 :  {
646 :      cnt0++;
647 :      cnt1++;
648 :      cnt_lcd++;
649 :
650 :      /* 拡張スイッチ用関数(1msごとに実行)    */
651 :      switchProcess();
652 :
653 :      /* LCD表示処理用関数(1msごとに実行)     */
654 :      lcdShowProcess();
655 :  }
```

中略

```
793 :  /************************************************************************/
794 :  /* モータ速度制御                                                       */
795 :  /* 引数    左モータ:-100～100、右モータ:-100～100                       */
796 :  /*         0で停止、100で正転100%、-100で逆転100%                       */
797 :  /* 戻り値 なし                                                          */
798 :  /************************************************************************/
799 :  void motor( int accele_l, int accele_r )
800 :  {
801 :      int    sw_data;
802 :
803 :      accele_l = accele_l * data_buff[DF_PWM] / 100;
804 :      accele_r = accele_r * data_buff[DF_PWM] / 100;
805 :
```

```
806 :      /* 左モータ制御 */
807 :      if( accele_l >= 0 ) {
808 :          p2 &= 0xfd;
809 :          trdgrd0 = (long)( PWM_CYCLE - 1 ) * accele_l / 100;
810 :      } else {
811 :          p2 |= 0x02;
812 :          trdgrd0 = (long)( PWM_CYCLE - 1 ) * ( -accele_l ) / 100;
813 :      }
814 :
815 :      /* 右モータ制御 */
816 :      if( accele_r >= 0 ) {
817 :          p2 &= 0xf7;
818 :          trdgrc1 = (long)( PWM_CYCLE - 1 ) * accele_r / 100;
819 :      } else {
820 :          p2 |= 0x08;
821 :          trdgrc1 = (long)( PWM_CYCLE - 1 ) * ( -accele_r ) / 100;
822 :      }
823 :  }
```

中略

```
838 :  /************************************************************************/
839 :  /* 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     */
840 :  /* 引数　 外輪PWM                                                       */
841 :  /* 戻り値 内輪PWM                                                       */
842 :  /************************************************************************/
843 :  int diff( int pwm )
844 :  {
845 :      int ret;
846 :
847 :      if( pwm >= 0 ) {
848 :          /* PWM値が正の数なら */
849 :          if( angle_buff < 0 ) {
850 :              angle_buff = -angle_buff;
851 :          }
852 :          ret = revolution_difference[angle_buff] * pwm / 100;
853 :      } else {
854 :          /* PWM値が負の数なら */
855 :          ret = pwm;                      /* そのまま返す             */
856 :      }
857 :      return ret;
858 :  }
859 :
860 :  /************************************************************************/
861 :  /* DataFlashのパラメータ読み込み                                        */
862 :  /* 引数         なし                                                    */
863 :  /* 戻り値       なし                                                    */
864 :  /************************************************************************/
865 :  void readDataFlashParameter( void )
866 :  {
867 :      int             i;
868 :      unsigned int    st = DF_ADDR_END + 1 - DF_PARA_SIZE;
869 :      signed char     c;
870 :
871 :      while( 1 ) {
872 :          // 読み込む番地を探す
873 :          readDataFlash( st, &c, 1 );
874 :          if( c == 0x11 ) {
875 :              readDataFlash( st, data_buff, DF_PARA_SIZE );
876 :              break;
877 :          }
878 :
879 :          st -= DF_PARA_SIZE;
880 :
881 :          if( st < DF_ADDR_START ) {
882 :              // 該当無し　初めて使用
883 :              for( i=0; i<DF_PARA_SIZE; i++ ) data_buff[ i ] = 0;
884 :              data_buff[DF_CHECK]     = 0x11;
885 :              data_buff[DF_SERVO1]    = SERVO_CENTER >> 8;
886 :              data_buff[DF_SERVO2]    = SERVO_CENTER & 0xff;
887 :              data_buff[DF_PWM]       = 50;
888 :              data_buff[DF_CURVE_PWM] = 50;
889 :              data_buff[DF_CRANK_PWM] = 40;
890 :
891 :              blockEraseDataFlash( DF_ADDR_START );
892 :              writeDataFlash( DF_ADDR_START, data_buff, DF_PARA_SIZE );
893 :              break;
894 :          }
895 :      }
896 :  }
897 :
```

```
898 :  /************************************************************************/
899 :  /* DataFlashへパラメータ書き込み                                        */
900 :  /* 引数         なし                                                    */
901 :  /* 戻り値       なし                                                    */
902 :  /************************************************************************/
903 :  void writeDataFlashParameter( void )
904 :  {
905 :      unsigned int    st = DF_ADDR_START;
906 :      signed char     c;
907 :
908 :      while( 1 ) {
909 :          // 書き込む番地を探す
910 :          readDataFlash( st, &c, 1 );
911 :          if( c == -1 ) {
912 :              writeDataFlash( st, data_buff, DF_PARA_SIZE );
913 :              break;
914 :          }
915 :
916 :          st += DF_PARA_SIZE;
917 :
918 :          if( st > DF_ADDR_END ) {
919 :              // すべて使用したら、イレーズして先頭に書き込み
920 :              blockEraseDataFlash( DF_ADDR_START );
921 :              writeDataFlash( DF_ADDR_START, data_buff, DF_PARA_SIZE );
922 :              break;
923 :          }
924 :      }
925 :  }
926 :
927 :  /************************************************************************/
928 :  /* LCDとスイッチを使ったパラメータセット処理                            */
929 :  /* 引数         なし                                                    */
930 :  /* 戻り値       なし                                                    */
931 :  /************************************************************************/
932 :  int lcdProcess( void )
933 :  {
934 :      int i;
935 :
936 :      if( pattern != 0 ) {
937 :          if( cnt_lcd >= 250 ) {
938 :              cnt_lcd = 0;
939 :              lcdPosition( 0, 0 );
940 :                      /* 0123456789abcbef 1行16文字 */
941 :              lcdPrintf( "pattern = %3d   ", pattern );
942 :                      /* 01234567..89abcde.f 1行16文字 */
943 :              lcdPrintf( "sensor=%02x bar=%d ",
944 :                          sensor_inp( 0xff ), startbar_get() );
945 :          }
946 :          return;
947 :      }
948 :
949 :      /* スイッチ4  設定値保存 */
950 :      if( getSwFlag(SW_4) ) {
951 :          // パラメータ保存
952 :          writeDataFlashParameter();
953 :      }
954 :
955 :      /* スイッチ3  メニュー＋１ */
956 :      if( getSwFlag(SW_3) ) {
957 :          lcd_pattern++;
958 :          if( lcd_pattern == 5 ) lcd_pattern = 1;
959 :      }
960 :
961 :      /* スイッチ2  メニューー１ */
962 :      if( getSwFlag(SW_2) ) {
963 :          lcd_pattern--;
964 :          if( lcd_pattern == 0 ) lcd_pattern = 4;
965 :      }
966 :
967 :      /* LCD、スイッチ処理 */
968 :      switch( lcd_pattern ) {
969 :      case 1:
970 :          /* サーボセンタ値調整 */
971 :          if( getSwFlag(SW_1) ) {
972 :              servo_center++;
973 :              if( servo_center > 10000 ) servo_center = 10000;
974 :          }
975 :          if( getSwFlag(SW_0) ) {
976 :              servo_center--;
977 :              if( servo_center < 1000 ) servo_center = 1000;
978 :          }
979 :          data_buff[DF_SERVO1] = servo_center >> 8;
980 :          data_buff[DF_SERVO2] = servo_center & 0xff;
981 :          handle( 0 );
982 :
983 :          /* LCD処理 */
984 :          lcdPosition( 0, 0 );
985 :                  /* 0123456789ab..f 1行16文字 */
986 :          lcdPrintf( "01 servo = %05d", servo_center );
987 :                  /* 01234567..89abcde.f 1行16文字 */
```

```
 988 :          lcdPrintf( "sensor=%02x bar=%d ",
 989 :                      sensor_inp( 0xff ), startbar_get() );
 990 :          break;
 991 :
 992 :      case 2:
 993 :          /* PWM値調整 */
 994 :          i = data_buff[DF_PWM];
 995 :          if( getSwFlag(SW_1) ) {
 996 :              i++;
 997 :              if( i > 100 ) i = 100;
 998 :          }
 999 :          if( getSwFlag(SW_0) ) {
1000 :              i--;
1001 :              if( i < 0 ) i = 0;
1002 :          }
1003 :          data_buff[DF_PWM] = i;
1004 :
1005 :          /* LCD処理 */
1006 :          lcdPosition( 0, 0 );
1007 :                  /* 0123456789..bcdef 1行16文字 */
1008 :          lcdPrintf( "02 pwm = %03d    ", i );
1009 :                  /* 01234567..89abcde.f 1行16文字 */
1010 :          lcdPrintf( "sensor=%02x bar=%d ",
1011 :                      sensor_inp( 0xff ), startbar_get() );
1012 :          break;
1013 :
1014 :      case 3:
1015 :          /* 大曲げPWM値調整 */
1016 :          i = data_buff[DF_CURVE_PWM];
1017 :          if( getSwFlag(SW_1) ) {
1018 :              i++;
1019 :              if( i > 100 ) i = 100;
1020 :          }
1021 :          if( getSwFlag(SW_0) ) {
1022 :              i--;
1023 :              if( i < 0 ) i = 0;
1024 :          }
1025 :          data_buff[DF_CURVE_PWM] = i;
1026 :
1027 :          /* LCD処理 */
1028 :          lcdPosition( 0, 0 );
1029 :                  /* 0123456789abc..ef 1行16文字 */
1030 :          lcdPrintf( "03 curve =  %03d ", i );
1031 :                  /* 01234567..89abcde.f 1行16文字 */
1032 :          lcdPrintf( "sensor=%02x bar=%d ",
1033 :                      sensor_inp( 0xff ), startbar_get() );
1034 :          break;
1035 :
1036 :      case 4:
1037 :          /* クランクPWM値調整 */
1038 :          i = data_buff[DF_CRANK_PWM];
1039 :          if( getSwFlag(SW_1) ) {
1040 :              i++;
1041 :              if( i > 100 ) i = 100;
1042 :          }
1043 :          if( getSwFlag(SW_0) ) {
1044 :              i--;
1045 :              if( i < 0 ) i = 0;
1046 :          }
1047 :          data_buff[DF_CRANK_PWM] = i;
1048 :
1049 :          /* LCD処理 */
1050 :          lcdPosition( 0, 0 );
1051 :                  /* 0123456789abcd..f 1行16文字 */
1052 :          lcdPrintf( "04crankpwm = %03d", i );
1053 :                  /* 01234567..89abcde.f 1行16文字 */
1054 :          lcdPrintf( "sensor=%02x bar=%d ",
1055 :                      sensor_inp( 0xff ), startbar_get() );
1056 :          break;
1057 :      }
1058 :  }
1059 :
1060 :  /************************************************************************/
1061 :  /* end of file                                                          */
1062 :  /************************************************************************/
```

以下、略

9.5 プログラムの解説

9.5.1 ヘッダファイルの取り込み

```
15 :  /*====================================*/
16 :  /* インクルード                       */
17 :  /*====================================*/
18 :  #include <stdio.h>
19 :  #include "sfr_r838a.h"              /* R8C/38A SFRの定義ファイル    */
20 :  #include "lcd_lib.h"                /* LCD表示用追加                */
21 :  #include "switch_lib.h"             /* スイッチ追加                 */
22 :  #include "data_flash_lib.h"         /* データフラッシュライブラリ   */
```

22 行目の「data_flash_lib.h」は、データフラッシュを制御するためのプログラム「data_flash_lib.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ダファイルです。

9.5.2 シンボル定義(データフラッシュ部分)

```
44 :  /* DataFlash関連 */
45 :  #define DF_ADDR_START   0x3000        /* 書き込み開始アドレス         */
46 :  #define DF_ADDR_END     0x33ff        /* 書き込み終了アドレス         */
47 :
48 :  #define DF_PARA_SIZE    32            /* DataFlashパラメータ数        */
49 :
50 :  #define DF_CHECK        0x00          /* DataFlashチェック            */
51 :  #define DF_SERVO1       0x01          /* サーボセンタ値               */
52 :  #define DF_SERVO2       0x02          /* サーボセンタ値               */
53 :  #define DF_PWM          0x03          /* PWM値                        */
54 :  #define DF_CURVE_PWM    0x04          /* 大曲げPWM値                  */
55 :  #define DF_CRANK_PWM    0x05          /* クランクPWM値                */
```

今回は保存領域として、データフラッシュ ブロック A を使用します。シンボル定義の内容を下表に示します。

シンボル定義	値	内容
DF_ADDR_START	0x3000	データフラッシュ ブロック A の開始アドレス
DF_ADDR_END	0x33ff	データフラッシュ ブロック A の終了アドレス
DF_PARA_SIZE	32	データフラッシュに書き込むデータの数

「DF_xxx」は、data_buff配列変数の何番目にデータを格納するかを決める番地です。シンボル定義の内容を下表に示します。

シンボル定義	番地	内容
DF_CHECK	0x00	データフラッシュに保存している値が、正しい値かどうかチェックします。具体的には、この番地の値が 0x11 でないなら、正しくないと判断します。
DF_SERVO1	0x01	サーボセンタ値を保存します。1 バイトは、-128～127 までしか保存できないので、2 バイト確保して、-32768～32767 まで保存できるようにします。
DF_SERVO2	0x02	
DF_PWM	0x03	PWM 値を保存します。motor 関数の設定値×この値が、実際のモータに出力される PWM 値になります。
DF_CURVE_PWM	0x04	大曲げ時の PWM 値です。
DF_CRANK_PWM	0x05	クロスライン検出後、徐行して進むときの PWM 値です。

9.5.3 プロトタイプ宣言

```
 57 :  /*====================================*/
 58 :  /* プロトタイプ宣言                   */
 59 :  /*====================================*/
 60 :  void init( void );
 61 :  void timer( unsigned long timer_set );

中略

 72 :  int diff( int pwm );
 73 :  void readDataFlashParameter( void );
 74 :  void writeDataFlashParameter( void );
 75 :  int lcdProcess( void );
```

72 行目	外輪 PWM 値とハンドル角度から内輪 PWM 値を返す関数です。
73 行目	パラメータをデータフラッシュに書き込むための関数です。
74 行目	パラメータをデータフラッシュから読み込むための関数です。
75 行目	パラメータの調整をするための関数です。

※73 行目～75 行目の関数の詳細については、後述します。

9.5.4 グローバル変数の宣言

```
 77 :  /*====================================*/
 78 :  /* グローバル変数の宣言               */
 79 :  /*====================================*/
 80 :  unsigned long   cnt0;                /* timer関数用                */
 81 :  unsigned long   cnt1;                /* main内で使用               */
 82 :  unsigned long   cnt_lcd;             /* LCD処理で使用              */
 83 :  int             pattern;             /* パターン番号               */
 84 :  int             servo_center;        /* サーボセンタ値             */
 85 :
 86 :  /* DataFlash関係 */
 87 :  signed char     data_buff[ DF_PARA_SIZE ];
 88 :
 89 :  /* LCD関連 */
 90 :  int             lcd_pattern = 2;
```

パラメータの調整をするに当たって、新たにグローバル変数を追加しています。各変数の役割を下表に示します。

変数名	内容
cnt_lcd	lcdProcess 関数内で使用するタイマ用の変数です。タイマ RB 割り込み関数内で 1ms ごとにインクリメントしています。
servo_center	サーボセンタ値を格納します。
data_buff	データフラッシュに書き込むデータ、または、データフラッシュから読み込んだデータを格納する配列変数です。
lcd_pattern	lcdProcess 関数内で使用するパターン変数です。main 関数内で使用しているパターン変数と同じ役割をしています。

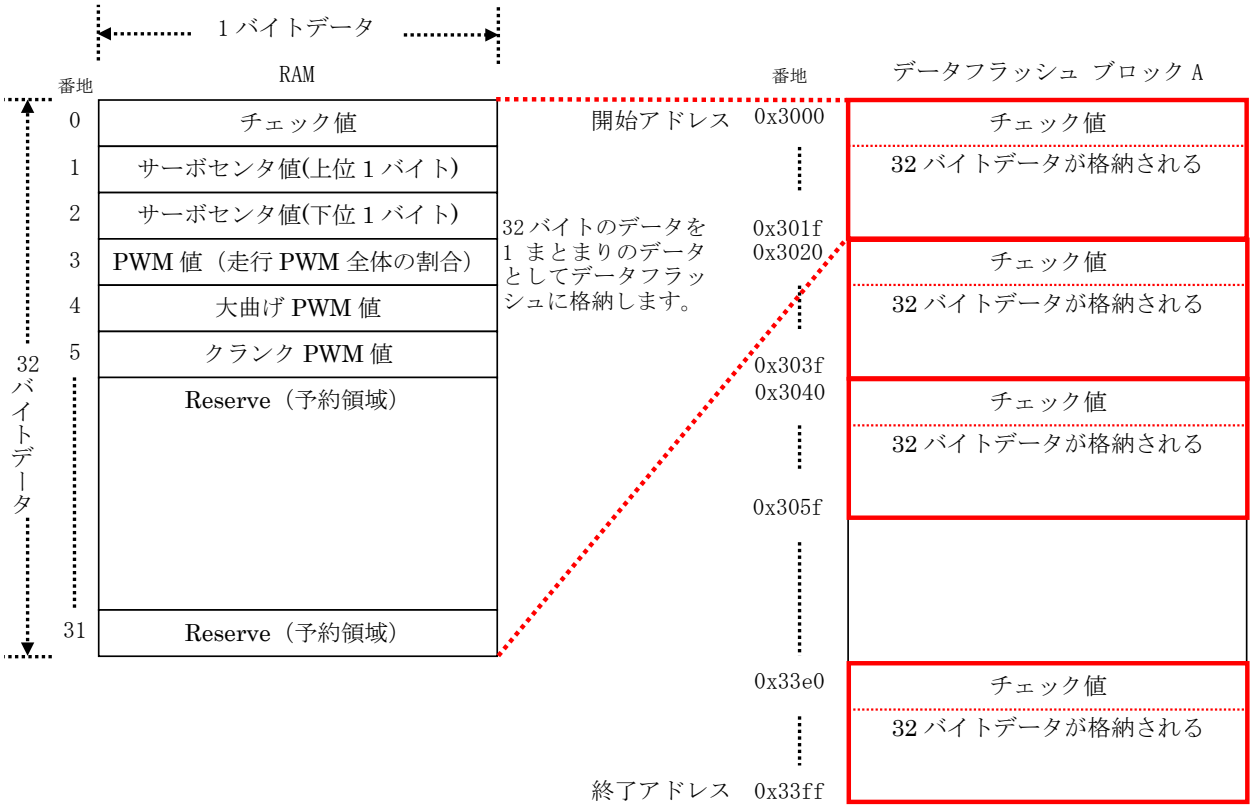
9.5.5 writeDataFlashParameter 関数(データフラッシュにパラメータの書き込み)

writeDataFlashParameter 関数は、データフラッシュにパラメータを書き込みます。

```
898 :  /************************************************************************/
899 :  /* DataFlashへパラメータ書き込み                                        */
900 :  /* 引数         なし                                                    */
901 :  /* 戻り値       なし                                                    */
902 :  /************************************************************************/
903 :  void writeDataFlashParameter( void )
904 :  {
905 :      unsigned int    st = DF_ADDR_START;
906 :      signed char     c;
907 :
908 :      while( 1 ) {
909 :          // 書き込む番地を探す
910 :          readDataFlash( st, &c, 1 );
911 :          if( c == -1 ) {
912 :              writeDataFlash( st, data_buff, DF_PARA_SIZE );
913 :              break;
914 :          }
915 :
916 :          st += DF_PARA_SIZE;
917 :
918 :          if( st > DF_ADDR_END ) {
919 :              // すべて使用していたら、イレーズして先頭に書き込み
920 :              blockEraseDataFlash( DF_ADDR_START );
921 :              writeDataFlash( DF_ADDR_START, data_buff, DF_PARA_SIZE );
922 :              break;
923 :          }
924 :      }
925 :  }
```

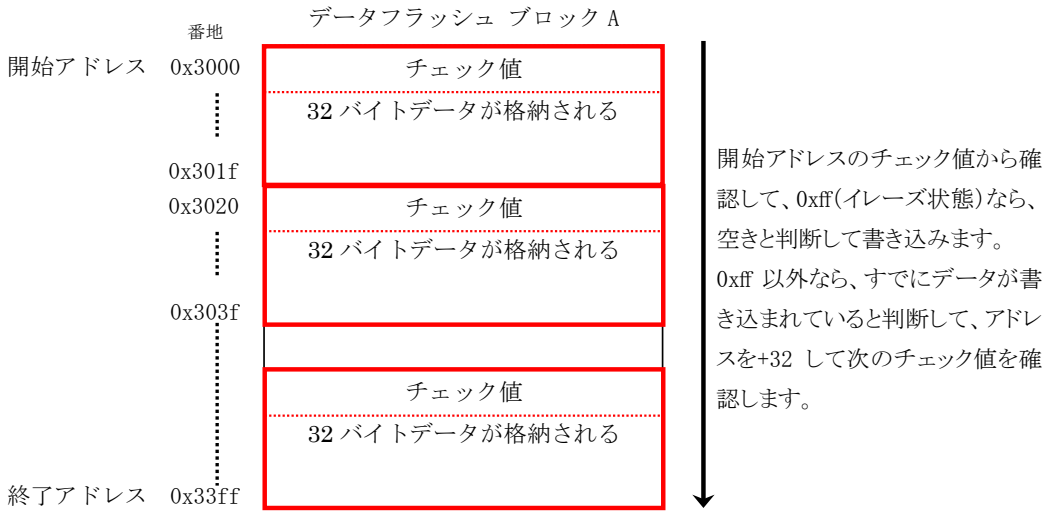
905 行目	データフラッシュに書き込むブロックの開始アドレスを st 変数に設定します。 今回は、ブロックＡを使用します。ブロックＡの番地は 0x3000～0x33ff 番地(1KB)です。開始アドレスを設定しますので 0x3000 を設定します。0x3000 は、シンボル定義で「DF_ADDR_START」と定義していますので、DF_ADDR_START と記述します。
910 行目 ～ 914 行目	910 行目でチェック値を読み込み、-1(0xff)かどうかチェックします。0xff(イレーズ状態)であれば 912 行目でデータを書き込み、関数処理を終了します。
916 行目	すでにデータが書き込まれていたと判断した場合は、次のデータ保存領域の番地をst変数に設定します。DF_PARA_SIZE は、シンボル定義で 32 と定義していますので、次の番地はst変数に 32 を足した番地となります。
918 行目 ～ 923 行目	st変数がブロックの終了アドレスを超えた場合は、ブロック A で保存できる空き領域がなかったと判断します。その場合は、ブロック A をイレーズ(データの消去)し、ブロックの開始アドレスに書き込みます。

データフラッシュに書き込むデータは、チェック値、サーボセンタ値、PWM 値(走行 PWM 全体の割合)、大曲げ PWM 値、クランク PWM 値の 5 つのデータを書き込みます。データフラッシュは、1 番地(アドレス)あたり1バイトのデータを書き込むことができます。書き込み先の番地を下図に示します。



Reserve(予約領域)は、後々プログラムを改良してパラメータで設定できる項目を増やし、パラメータなどのデータを保存することができる領域として確保しています。データフラッシュ ブロック A は 1024 バイト(1KB)ありますので、32 バイトのデータを 32 回分保存することができます。

1 回目の書き込みをするときは、データフラッシュ ブロック A の開始番地からデータを書き込みます。2 回目以降の書き込み先は、ブロックの開始アドレスから終了アドレスに向かって空き領域を探し、空き領域にデータを書き込みます。32 バイトのデータを 32 回書き込むとデータフラッシュ ブロック A のメモリがいっぱいになります。メモリがいっぱいになった場合は、イレーズ(データの消去)を行い、再度データフラッシュ ブロック A の先頭番地から順に書き込みます。



9.5.6 readDataFlashParameter 関数(データフラッシュからパラメータの読み込み)

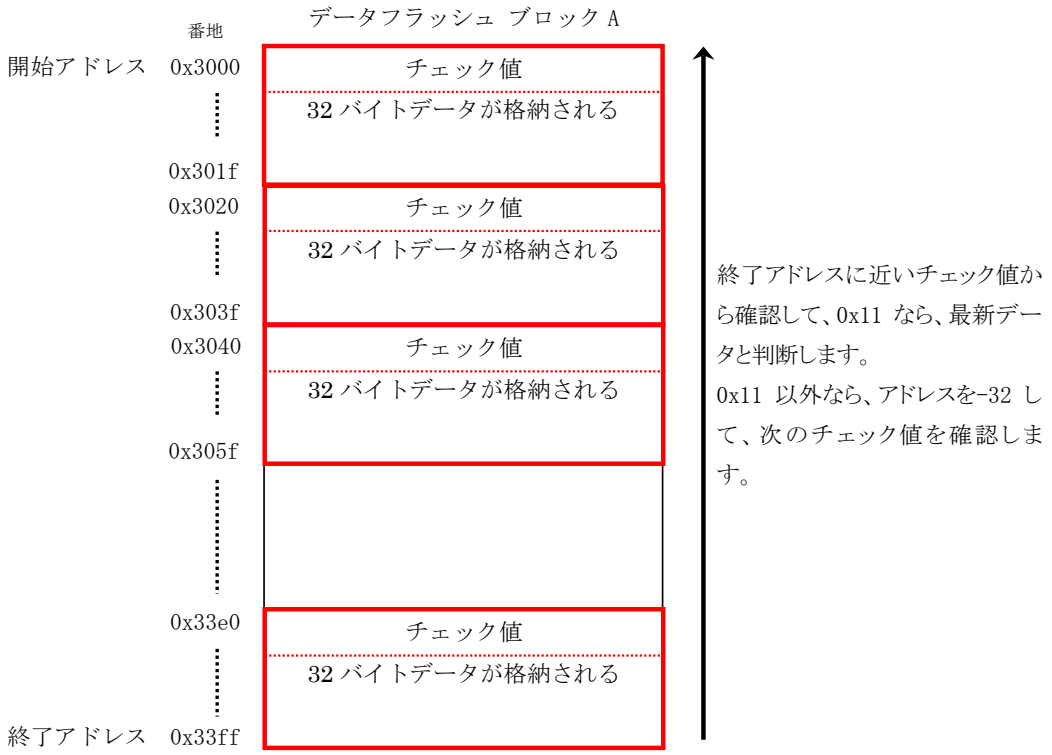
readDataFlashParameter 関数は、データフラッシュから最新のパラメータを読み込みます。

```
860 :  /*****************************************************************************/
861 :  /* DataFlashのパラメータ読み込み                                             */
862 :  /* 引数         なし                                                         */
863 :  /* 戻り値       なし                                                         */
864 :  /*****************************************************************************/
865 :  void readDataFlashParameter( void )
866 :  {
867 :      int             i;
868 :      unsigned int    st = DF_ADDR_END + 1 - DF_PARA_SIZE;
869 :      signed char     c;
870 :
871 :      while( 1 ) {
872 :          // 読み込む番地を探す
873 :          readDataFlash( st, &c, 1 );
874 :          if( c == 0x11 ) {
875 :              readDataFlash( st, data_buff, DF_PARA_SIZE );
876 :              break;
877 :          }
878 :
879 :          st -= DF_PARA_SIZE;
880 :
881 :          if( st < DF_ADDR_START ) {
882 :              // 該当無し  初めて使用
883 :              for( i=0; i<DF_PARA_SIZE; i++ ) data_buff[ i ] = 0;
884 :              data_buff[DF_CHECK]     = 0x11;
885 :              data_buff[DF_SERVO1]    = SERVO_CENTER >> 8;
886 :              data_buff[DF_SERVO2]    = SERVO_CENTER & 0xff;
887 :              data_buff[DF_PWM]       = 50;
888 :              data_buff[DF_CURVE_PWM] = 50;
889 :              data_buff[DF_CRANK_PWM] = 40;
890 :
891 :              blockEraseDataFlash( DF_ADDR_START );
892 :              writeDataFlash( DF_ADDR_START, data_buff, DF_PARA_SIZE );
893 :              break;
894 :          }
895 :      }
896 :  }
```

868 行目	データフラッシュからデータを読み込む最初のチェック値が書き込まれている番地をst変数に設定します。st変数に設定する番地は、DF_ADDR_END + 1 - DF_PARA_SIZE＝0x33e0 です。 **(0x33e0 番地にブロック A 最後のチェック値が保存されています。)**
873 行目 ～ 877 行目	873 行目でチェック値を読み込み、0x11 かどうかをチェックします。0x11 であれば最後に書き込んだデータと判断し、875 行目で 32 バイトのデータを読み込み、関数処理を終了します。
879 行目	チェック値が 0x11 でない場合は、次のデータ保存領域の番地をst変数に設定します。DF_PARA_SIZE は、シンボル定義で 32 と定義していますので、次の番地はst変数に 32 を引いた番地となります。
881 行目 ～ 894 行目	st変数がブロックの開始アドレスより小さくなった場合は、今回のプログラムでデータフラッシュを初めて使用したと判断し、イレーズ後初期値を書き込みます。

データフラッシュからデータを読み込む場合は、最新のデータを読み込まなければいけません。

最新データは、最後に書き込んだデータなのでデータフラッシュの後ろにあります。よって、ブロックの終了アドレスから開始アドレスに向かって保存されているデータの位置をチェックし、最新のデータを読み込みます。その様子を下図に示します。



9.5.7 lcdProcess 関数(パラメータの調整)

(1)pattern 変数が"0"以外のときの処理

pattern 変数値が"0"(パターン 0:スイッチ入力待ち)以外のときはマイコンカーが動いている状態です。動いている状態ではパラメータの調整はしませんので、液晶に pattern 変数値、センサ値、スタートバー検出センサ値を表示するようにします。値は 250ms ごとに更新します。

```
927 :  /*****************************************************************************/
928 :  /* LCDとスイッチを使ったパラメータセット処理                                  */
929 :  /* 引数          なし                                                        */
930 :  /* 戻り値        なし                                                        */
931 :  /*****************************************************************************/
932 :  int lcdProcess( void )
933 :  {
934 :      int i;
935 :
936 :      if( pattern != 0 ) {
937 :          if( cnt_lcd >= 250 ) {
938 :              cnt_lcd = 0;
939 :              lcdPosition( 0, 0 );
940 :                       /* 0123456789abcbef 1行16文字 */
941 :              lcdPrintf( "pattern = %3d   ", pattern );
942 :                       /* 01234567..89abcde.f 1行16文字 */
943 :              lcdPrintf( "sensor=%02x bar=%d ",
944 :                          sensor_inp( 0xff ), startbar_get() );
945 :          }
946 :          return;
947 :      }
948 :
949 :      /* スイッチ4  設定値保存 */
950 :      if( getSwFlag(SW_4) ) {
951 :          // パラメータ保存
952 :          writeDataFlashParameter();
953 :      }
```

936 行目	メイン関数で使用している pattern 変数が"0"以外のとき、937 行目～946 行目のプログラムを実行します。
937 行目 ～ 945 行目	938 行目の cnt_lcd 変数は、タイマ RB 割り込み関数内で 1ms ごとにカウントアップします。cnt_lcd 変数が 250 以上になったら、if 文の中を実行し、cnt_lcd 変数を"0"にします。 939 行目～944 行目は、pattern 変数、センサ、スタートバー検出センサの値を液晶に表示します。 結果、938 行目～945 行目は、250ms ごとに液晶の表示を更新しているということになります。
946 行目	return 文で lcdProcess 関数を抜け、メイン関数に戻ります。

（2）pattern 変数が"0"のときの処理（パラメータの調整）

pattern 変数値が"0"（パターン 0:スイッチ入力待ち）のときは、パラメータを調整します。

①メニュー画面の切り替えと保存

```
949 :       /* スイッチ4　設定値保存 */
950 :       if( getSwFlag(SW_4) ) {
951 :           // パラメータ保存
952 :           writeDataFlashParameter();
953 :       }
954 :
955 :       /* スイッチ3　メニュー＋１ */
956 :       if( getSwFlag(SW_3) ) {
957 :           lcd_pattern++;
958 :           if( lcd_pattern == 5 ) lcd_pattern = 1;
959 :       }
960 :
961 :       /* スイッチ2　メニューー１ */
962 :       if( getSwFlag(SW_2) ) {
963 :           lcd_pattern--;
964 :           if( lcd_pattern == 0 ) lcd_pattern = 4;
965 :       }
```

SW_4～SW_2 を押したとき、lcd_pattern 変数を操作します。キーリピートを使用しています。

プッシュスイッチ	内容
SW_4	data_buff 配列変数に格納された各パラメータの値をデータフラッシュへ保存します。
SW_3	lcd_pattern 変数をインクリメントします。lcd_pattern 変数が"5"になったら、"1"にします。
SW_2	lcd_pattern 変数をデクリメントします。lcd_pattern 変数が"0"になったら、"4"にします。

②サーボセンタ値調整

lcd_pattern 変数が"1"であれば、サーボセンタ値調整モードです。
サーボセンサ値調整モードでは、サーボのセンタ値を調整します。

```
967 :       /* LCD、スイッチ処理 */
968 :       switch( lcd_pattern ) {
969 :       case 1:
970 :           /* サーボセンタ値調整 */
971 :           if( getSwFlag(SW_1) ) {
972 :               servo_center++;
973 :               if( servo_center > 10000 ) servo_center = 10000;
974 :           }
975 :           if( getSwFlag(SW_0) ) {
976 :               servo_center--;
977 :               if( servo_center < 1000 ) servo_center = 1000;
978 :           }
979 :           data_buff[DF_SERVO1] = servo_center >> 8;
980 :           data_buff[DF_SERVO2] = servo_center & 0xff;
981 :           handle( 0 );
982 :
983 :           /* LCD処理 */
984 :           lcdPosition( 0, 0 );
985 :                   /* 0123456789ab..f 1行16文字 */
986 :           lcdPrintf( "01 servo = %05d", servo_center );
987 :                   /* 01234567..89abcde.f 1行16文字 */
988 :           lcdPrintf( "sensor=%02x bar=%d ",
989 :                       sensor_inp( 0xff ), startbar_get() );
990 :           break;
```

971 行目 ～ 978 行目	SW_1 で servo_center 変数の値をインクリメントします。 SW_0 で servo_center 変数の値をデクリメントします。 調整できる値の範囲は、1000～10000 までです。
979 行目	data_buff 配列変数の DF_SERVO1(0x01)個目に、servo_center の上位 1 バイトデータを格納します。servo_center 変数は 2 バイトデータなので、8 ビット右にシフトした値を data_buff 配列変数の DF_SERVO1(0x01)個目に格納します。 例)servo_center = 0000 1110 1010 0110 (0x0EA6)　>>　8 = 0000 0000 0000 1110 (0x000E) となります。
980 行目	data_buff 配列変数の DF_SERVO2(0x02)個目に、servo_center の下位 1 バイトデータを格納します。servo_center 変数は 2 バイトデータなので、論理積(AND)を使って下位 1 バイトデータを取り出した data_buff 配列変数の DF_SERVO2(0x02)個目に格納します。 例)servo_center = 0000 1110 1010 0110 (0x0EA6)　&　0xff = 0000 0000 1010 0110 (0x00A6) となります。
981 行目	handle 関数を使って、servo_center に設定した値をサーボモータへ出力します。
984 行目 ～ 989 行目	液晶の 1 行目に servo_center の値を表示し、2 行目にセンサとスタートバー検出センサの値を表示します。

③PWM 値調整(走行 PWM 全体の割合)

lcd_pattern 変数が"2"であれば、PWM 値調整モードです。
PWM 値調整モードでは、実際に、モータに出力される割合を調整します。

```
 992 :      case 2:
 993 :          /* PWM値調整 */
 994 :          i = data_buff[DF_PWM];
 995 :          if( getSwFlag(SW_1) ) {
 996 :              i++;
 997 :              if( i > 100 ) i = 100;
 998 :          }
 999 :          if( getSwFlag(SW_0) ) {
1000 :              i--;
1001 :              if( i < 0 ) i = 0;
1002 :          }
1003 :          data_buff[DF_PWM] = i;
1004 :
1005 :          /* LCD処理 */
1006 :          lcdPosition( 0, 0 );
1007 :                   /* 0123456789..bcdef 1行16文字 */
1008 :          lcdPrintf( "02 pwm = %03d    ", i );
1009 :                   /* 01234567..89abcde.f 1行16文字 */
1010 :          lcdPrintf( "sensor=%02x bar=%d ",
1011 :                      sensor_inp( 0xff ), startbar_get() );
1012 :          break;
```

994 行目	PWM 値が格納されている data_buff 配列変数の DF_PWM(0x03)個目から値を i 変数に読み込みます。
995 行目 ～ 1002 行目	SW_1 で i 変数の値をインクリメントします。 SW_0 で i 変数の値をデクリメントします。 調整できる値の範囲は、0～100 までです。
1003 行目	i 変数で調整をした値を data_buff 配列変数の DF_PWM(0x03)個目に格納します。
1006 行目 ～ 1011 行目	液晶の 1 行目に PWM(i 変数)値を表示し、2 行目にセンサとスタートバー検出センサの値を表示します。

④大曲げ PWM 値調整

lcd_pattern 変数が"3"であれば、大曲げ PWM 値調整モードです。
大曲げ PWM 値調整モードでは、右大曲げ(case 12)、左大曲げ(case 13)時の PWM 値を調整します。

```
1014 :      case 3:
1015 :          /* 大曲げPWM値調整 */
1016 :          i = data_buff[DF_CURVE_PWM];
1017 :          if( getSwFlag(SW_1) ) {
1018 :              i++;
1019 :              if( i > 100 ) i = 100;
1020 :          }
1021 :          if( getSwFlag(SW_0) ) {
1022 :              i--;
1023 :              if( i < 0 ) i = 0;
1024 :          }
1025 :          data_buff[DF_CURVE_PWM] = i;
1026 :
1027 :          /* LCD処理 */
1028 :          lcdPosition( 0, 0 );
1029 :                   /* 0123456789abc..ef 1行16文字 */
1030 :          lcdPrintf( "03 curve =  %03d ", i );
1031 :                   /* 01234567..89abcde.f 1行16文字 */
1032 :          lcdPrintf( "sensor=%02x bar=%d ",
1033 :                      sensor_inp( 0xff ), startbar_get() );
1034 :          break;
```

1016 行目	大曲げ PWM 値が格納されている data_buff 配列変数の DF_CURVE_PWM(0x04)個目から値を i 変数に読み込みます。
1017 行目 ～ 1024 行目	SW_1 で i 変数の値をインクリメントします。 SW_0 で i 変数の値をデクリメントします。 調整できる値の範囲は、0～100 までです。
1025 行目	i 変数で調整をした値を data_buff 配列変数の DF_CURVE_PWM(0x04)個目に格納します。
1028 行目 ～ 1033 行目	液晶の 1 行目に大曲げ PWM(i 変数)値を表示し、2 行目にセンサとスタートバー検出センサの値を表示します。

⑤クランク PWM 値調整

lcd_pattern 変数が"4"であれば、クランク PWM 値調整モードです。
クランク PWM 値調整モードでは、クロスライン検出後の PWM 値を調整します。今回のサンプルプログラムでは、クランク PWM 値で調整した値が左レーンチェンジ、右レーンチェジの PWM 値にも反映されるようになっています。

```
1036 :      case 4:
1037 :          /* クランクPWM値調整 */
1038 :          i = data_buff[DF_CRANK_PWM];
1039 :          if( getSwFlag(SW_1) ) {
1040 :              i++;
1041 :              if( i > 100 ) i = 100;
1042 :          }
1043 :          if( getSwFlag(SW_0) ) {
1044 :              i--;
1045 :              if( i < 0 ) i = 0;
1046 :          }
1047 :          data_buff[DF_CRANK_PWM] = i;
1048 :
1049 :          /* LCD処理 */
1050 :          lcdPosition( 0, 0 );
1051 :                   /* 0123456789abcd..f 1行16文字 */
1052 :          lcdPrintf( "04crankpwm = %03d", i );
1053 :                   /* 01234567..89abcde.f 1行16文字 */
1054 :          lcdPrintf( "sensor=%02x bar=%d ",
1055 :                      sensor_inp( 0xff ), startbar_get() );
1056 :          break;
1057 :      }
1058 :  }
```

1038 行目	クランク PWM 値が格納されている data_buff 配列変数の DF_CRANK_PWM(0x05)個目から値を i 変数に読み込みます。
1039 行目 ～ 1046 行目	SW_1 で i 変数の値をインクリメントします。 SW_0 で i 変数の値をデクリメントします。 調整できる値の範囲は、0～100 までです。
1047 行目	i 変数で調整をした値を data_buff 配列変数の DF_CRANK_PWM(0x05)個目に格納します。
1050 行目 ～ 1055 行目	液晶の 1 行目に、クランク PWM(i 変数)値を表示し、2 行目にセンサとスタートバー検出センサの値を表示します。

9.5.8 サーボセンタ値を読み込む

```
114 :      /* マイコン機能の初期化 */
115 :      init();                              /* 初期化                          */
116 :      asm(" fset I ");                     /* 全体の割り込み許可              */
117 :      initLcd();                           /* LCD初期化                       */
118 :      initSwitch();                        /* スイッチ初期化                  */
119 :
120 :      readDataFlashParameter();            /* DataFlashパラメータ読み込み  */
121 :      servo_center  = (unsigned char)data_buff[DF_SERV01] * 0x100;
122 :      servo_center |= (unsigned char)data_buff[DF_SERV02];
```

マイコンカーの電源を入れた瞬間、servo_center 変数の値は"0"です。そのため、サーボが正しく動きません。120 行目～122 行目で、データフラッシュからサーボセンタ値を読み込みます。

120 行目	データフラッシュに保存されているパラメータの値を 32 個、読み込みます。
121 行目	サーボセンタ値は、2 バイトデータのためデータフラッシュにデータを保存するとき、上位 1 バイト、下位 1 バイトに分けて保存しています。そのため、これらのデータを 1 つに結合します。 121 行目では、上位 1 バイトのデータを読み込み、0x100 倍しています。これは、下記の例のように左へ 8bit シフトしているのと同じになります。 例) 0x0E(上位 1 バイト) ＊ 0x100 = 0x0E00　/* 0x100 倍した場合 */ (0x0E(上位 1 バイト) & 0xff) << 8 = 0x0E00　/* 左へ 8bit シフトした場合 */
122 行目	サーボセンタ値の下位 1 バイトデータを読み込み、121 行目で読み込んだ上位 1 バイトデータと論理和(OR)で合わせます。 例)0x0E00(上位 1 バイト)｜0x00A6(下位 1 バイト) = 0x0EA6 → 3750 となります。

9.5.9 パラメータの調整

```
124 :      /* マイコンカーの状態初期化 */
125 :      handle( 0 );
126 :      motor( 0, 0 );
127 :
128 :      while( 1 ) {
129 :
130 :      // LCD表示、パラメータ設定処理
131 :      lcdProcess();
```

lcdProcess 関数は、サーボセンタ値、PWM 値(走行 PWM 全体の割合)、大曲げ PWM 値、クランク PWM 値のパラメータを調整するための関数です。マイコンカーの電源投入(または、リセット)後調整が行えます。pattern 変数が"0"以外のとき液晶には、pattern 変数の状態、センサの状態、スタートバー検出センサの状態が表示されます。

lcdProcess 関数は、128 行目の while 文の中を 1 周するたびに実行されます。

9.5.10 パターン 0:スイッチ入力待ち(パラメータの保存)

```
159 :       case 0:
160 :           /* スイッチ入力待ち */
161 :           if( pushsw_get() ) {
162 :               // パラメータ保存
163 :               writeDataFlashParameter();
164 :               pattern = 1;
165 :               cnt1 = 0;
166 :               break;
167 :           }
168 :           if( cnt1 < 100 ) {              /* LED点滅処理                  */
169 :               led_out( 0x1 );
170 :           } else if( cnt1 < 200 ) {
171 :               led_out( 0x2 );
172 :           } else {
173 :               cnt1 = 0;
174 :           }
175 :           break;
```

パラメータの調整などが終わり走行させるとき、モータドライブ基板 Ver.5 のプッシュスイッチを押します。調整したパラメータの値を 163 行目の writeDataFlashParameter 関数で保存し、パターン 1 へ移動します。

9.5.11 パターン 12、13:右大曲げ、左大曲げ(パラメータの反映)

```
271 :      case 12:
272 :          /* 右へ大曲げの終わりのチェック */
273 :          if( check_crossline() ) {       /* 大曲げ中もクロスラインチェック */
274 :              pattern = 21;
275 :              break;
276 :          }
277 :          if( check_rightline() ) {       /* 右ハーフラインチェック       */
278 :              pattern = 51;
279 :              break;
280 :          }
281 :          if( check_leftline() ) {        /* 左ハーフラインチェック       */
282 :              pattern = 61;
283 :              break;
284 :          }
285 :          i = data_buff[DF_CURVE_PWM];
286 :          switch( sensor_inp(MASK3_3) ) {
287 :          case 0x06:
288 :              pattern = 11;
289 :              break;
290 :          case 0x03:
291 :              handle( 20 );
292 :              motor( i, diff(i) );
293 :              break;
294 :          case 0x81:
295 :              handle( 25 );
296 :              motor( i, diff(i) );
297 :              break;
298 :          case 0xc1:
299 :          case 0xc0:
300 :              handle( 30 );
301 :              motor( i, diff(i) );
302 :              break;
303 :          case 0x60:
304 :              handle( 35 );
305 :              motor( 0, 0 );
306 :              break;
307 :          }
308 :          break;
```

285 行目で data_buff 配列変数の DF_CURVE_PWM(0x04)個目から i 変数にパラメータを代入します。代入された i 変数を motor 関数の引数に設定します。内輪側は、diff 関数を使用します。

パターン 13 は、パターン 12 の値を左右反対にしたものです。

9.5.12 パターン 23:クロスライン後のトレース、クランク検出(パラメータの反映)

```
366 :       case 23:
367 :           /* クロスライン後のトレース、クランク検出 */
368 :           if( sensor_inp(MASK4_4)==0xf8 ) {
369 :               /* 左クランクと判断→左クランククリア処理へ */
370 :               led_out( 0x1 );
371 :               handle( -38 );
372 :               motor( 10 ,50 );
373 :               pattern = 31;
374 :               cnt1 = 0;
375 :               break;
376 :           }
377 :           if( sensor_inp(MASK4_4)==0x1f ) {
378 :               /* 右クランクと判断→右クランククリア処理へ */
379 :               led_out( 0x2 );
380 :               handle( 38 );
381 :               motor( 50 ,10 );
382 :               pattern = 41;
383 :               cnt1 = 0;
384 :               break;
385 :           }
386 :           i = data_buff[DF_CRANK_PWM];
387 :           switch( sensor_inp(MASK3_3) ) {
388 :               case 0x00:
389 :                   /* センター→まっすぐ */
390 :                   handle( 0 );
391 :                   motor( i ,i );
392 :                   break;
393 :               case 0x04:
394 :               case 0x06:
395 :               case 0x07:
396 :               case 0x03:
397 :                   /* 左寄り→右曲げ */
398 :                   handle( 8 );
399 :                   motor( i ,diff(i) );
400 :                   break;
401 :               case 0x20:
402 :               case 0x60:
403 :               case 0xe0:
404 :               case 0xc0:
405 :                   /* 右寄り→左曲げ */
406 :                   handle( -8 );
407 :                   motor( diff(i) ,i );
408 :                   break;
409 :           }
410 :           break;
```

386 行目で data_buff 配列変数の DF_CRANK_PWM(0x05)個目から i 変数にパラメータを代入します。代入された i 変数を motor 関数の引数に設定します。内輪側は、diff 関数を使用します。

9.5.13 パターン 53、63:右ハーフライン、左ハーフライン後のトレース、レーンチェンジ

```
463 :      case 53:
464 :          /* 右ハーフライン後のトレース、レーンチェンジ */
465 :          if( sensor_inp(MASK4_4) == 0x00 ) {
466 :              handle( 15 );
467 :              motor( 40 ,diff(40) );
468 :              pattern = 54;
469 :              cnt1 = 0;
470 :              break;
471 :          }
472 :          i = data_buff[DF_CRANK_PWM];
473 :          switch( sensor_inp(MASK3_3) ) {
474 :              case 0x00:
475 :                  /* センター→まっすぐ */
476 :                  handle( 0 );
477 :                  motor( i ,i );
478 :                  break;
479 :              case 0x04:
480 :              case 0x06:
481 :              case 0x07:
482 :              case 0x03:
483 :                  /* 左寄り→右曲げ */
484 :                  handle( 8 );
485 :                  motor( i ,diff(i) );
486 :                  break;
487 :              case 0x20:
488 :              case 0x60:
489 :              case 0xe0:
490 :              case 0xc0:
491 :                  /* 右寄り→左曲げ */
492 :                  handle( -8 );
493 :                  motor( diff(i) ,i );
494 :                  break;
495 :              default:
496 :                  break;
497 :          }
498 :          break;
```

レーンチェンジは、472 行目でクランクと同じ data_buff 配列変数の DF_CRANK_PWM(0x05)個目から i 変数にパラメータを代入します。代入された i 変数を motor 関数の引数に設定します。内輪側は、diff 関数を使用します。

パターン 63 は、パターン 53 の値を左右反対にしたものです。

9.5.14 diff 関数

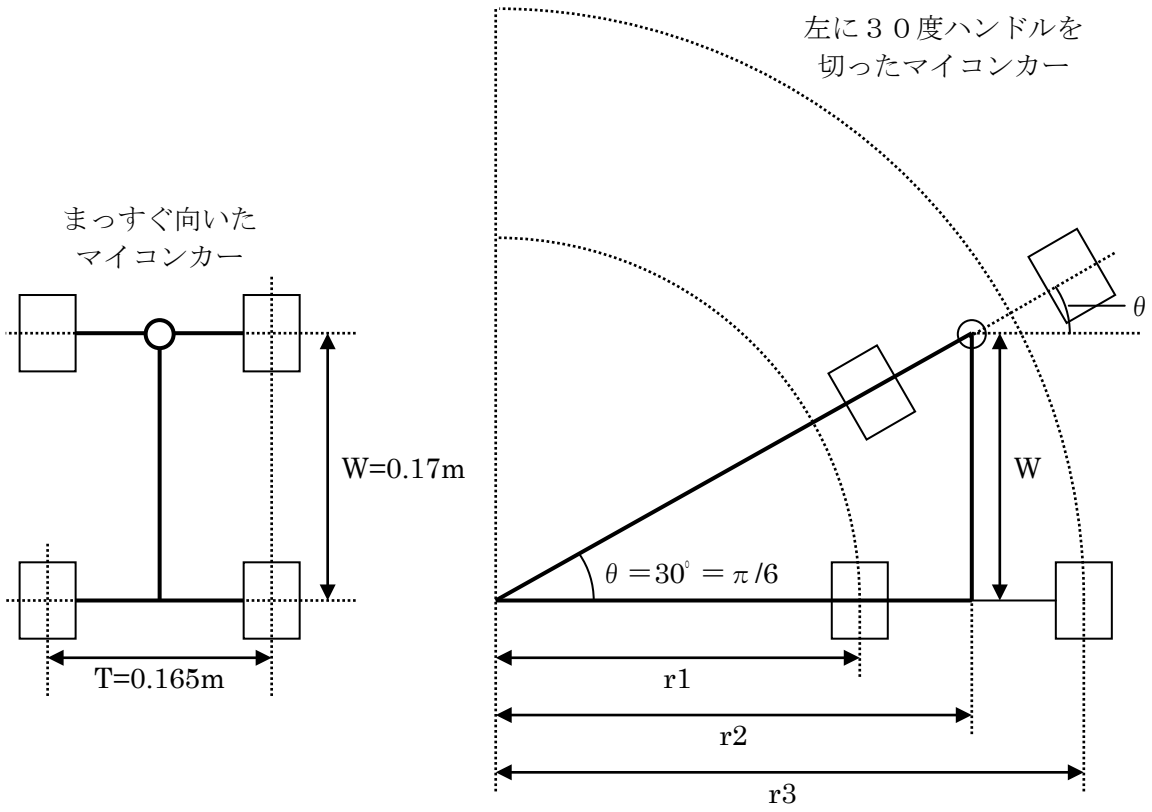
diff 関数は、現在の外輪 PWM 値とハンドル角度から内輪 PWM 値を求める関数です。

```
 95 :  const int revolution_difference[] = {   /* 角度から内輪、外輪回転差計算 */
 96 :      100, 99, 97, 96, 95,
 97 :      93, 92, 91, 89, 88,
 98 :      87, 86, 84, 83, 82,
 99 :      81, 79, 78, 77, 76,
100 :      75, 73, 72, 71, 70,
101 :      69, 67, 66, 65, 64,
102 :      62, 61, 60, 59, 58,
103 :      56, 55, 54, 52, 51,
104 :      50, 48, 47, 46, 44,
105 :      43 };

中略

838 :  /************************************************************************/
839 :  /* 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      */
840 :  /* 引数　 外輪PWM                                                       */
841 :  /* 戻り値 内輪PWM                                                       */
842 :  /************************************************************************/
843 :  int diff( int pwm )
844 :  {
845 :      int ret;
846 :
847 :      if( pwm >= 0 ) {
848 :          /* PWM値が正の数なら */
849 :          if( angle_buff < 0 ) {
850 :              angle_buff = -angle_buff;
851 :          }
852 :          ret = revolution_difference[angle_buff] * pwm / 100;
853 :      } else {
854 :          /* PWM値が負の数なら */
855 :          ret = pwm;                          /* そのまま返す             */
856 :      }
857 :      return ret;
858 :  }
```

95 行目 ～ 105 行目	revolution_difference 配列変数は、外輪 PWM の値が 100 のとき、ハンドル角度が 0～45 度までの 45 通りの内輪 PWM 値をあらかじめ計算した値を代入するための配列変数です。 **※内輪 PWM 値の計算方法については、「10. カーブ時のタイヤの左右回転差の計算」を参照してください。**
847 行目	外輪 PWM 値が 0 以上であるかをチェックします。0 以下であれば、855 行目で外輪 PWM 値と同じ値を ret 変数に代入して、外輪 PWM 値と同じ値を返します。
849 行目 ～ 851 行目	ハンドル角度が負の数であれば、正の数に直します。
852 行目	revolution_difference 配列変数の angle_buff(ハンドル角度)個目の値を取り出し、内輪 PWM 値を求めます。結果を ret 変数に代入して内輪 PWM 値を返します。
855 行目	PWM 値が負の数の場合は、正確な計算ができないので、そのまま負の数の PWM 値を返します。

10. カーブ時のタイヤの左右回転差の計算

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。



T＝トレッド…左右輪の中心線の距離　キットでは 0.165[m]です。
W＝ホイールベース…前輪と後輪の間隔　キットでは 0.17[m]です。

図のように、底辺 r2、高さ W、角度 θ の三角形の関係は次のようです。

$\tan\theta = W / r2$

角度 θ 、W が分かっていますので、r2 が分かります。

$r2 = W / \tan\theta = 0.170 / \tan(\pi/6) = 0.295$[m]

内輪の半径は、

$r1 = r2 - T/2 = 0.295 - 0.0825 = 0.212$[m]

外輪の半径は、

$r3 = r2 + T/2 = 0.295 + 0.0825 = 0.377$[m]

よって、外輪を 100 とすると内輪の回転数は、

$r1 / r3 \times 100 = 0.212 / 0.377 \times 100 = 56$

となります。

計算結果から、次のことが分かりました。

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 56 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( 56, 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の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	0.17	m ←ホイールベースを入力してください			
2		T	0.165	m ←トレッドを入力してください			
3							
4		度	rad	r2	r1	r3	r1/r3*100
5		0	0				100
6		1	0.017	9.744	9.662	9.827	98
7		2	0.035	4.871	4.788	4.953	97
8		3	0.052	3.245	3.163	3.328	95
9		4	0.070	2.432	2.350	2.515	93
10		5	0.087	1.944	1.862	2.027	92
11		6	0.105	1.618	1.536	1.701	90
12		7	0.122	1.385	1.303	1.468	89
13		8	0.140	1.210	1.128	1.293	87
14		9	0.157	1.074	0.991	1.156	86
15		10	0.174	0.965	0.882	1.047	84
16		11	0.192	0.875	0.793	0.958	83
17		12	0.209	0.800	0.718	0.883	81
18		13	0.227	0.737	0.654	0.819	80
19		14	0.244	0.682	0.600	0.765	78
20		15	0.262	0.635	0.552	0.717	77
21		16	0.279	0.593	0.511	0.676	76

セル	内容	値、式の例
C1	ホイールベースを入力	キットなら 0.17
C2	トレッドを入力	キットなら 0.165
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	9.744	9.662	9.827	98
2	0.035	4.871	4.788	4.953	97
3	0.052	3.245	3.163	3.328	95
4	0.070	2.432	2.350	2.515	93
5	0.087	1.944	1.862	2.027	92
6	0.105	1.618	1.536	1.701	90
7	0.122	1.385	1.303	1.468	89
8	0.140	1.210	1.128	1.293	87
9	0.157	1.074	0.991	1.156	86
10	0.174	0.965	0.882	1.047	84
11	0.192	0.875	0.793	0.958	83
12	0.209	0.800	0.718	0.883	81
13	0.227	0.737	0.654	0.819	80
14	0.244	0.682	0.600	0.765	78
15	0.262	0.635	0.552	0.717	77
16	0.279	0.593	0.511	0.676	76
17	0.297	0.556	0.474	0.639	74
18	0.314	0.523	0.441	0.606	73
19	0.331	0.494	0.411	0.576	71
20	0.349	0.467	0.385	0.550	70
21	0.366	0.443	0.361	0.526	69
22	0.384	0.421	0.339	0.504	67
23	0.401	0.401	0.318	0.483	66
24	0.419	0.382	0.300	0.465	64
25	0.436	0.365	0.282	0.447	63
26	0.454	0.349	0.266	0.431	62
27	0.471	0.334	0.251	0.416	60
28	0.488	0.320	0.237	0.402	59
29	0.506	0.307	0.224	0.389	58
30	0.523	0.295	0.212	0.377	56
31	0.541	0.283	0.201	0.366	55
32	0.558	0.272	0.190	0.355	53
33	0.576	0.262	0.179	0.344	52
34	0.593	0.252	0.170	0.335	51
35	0.611	0.243	0.160	0.325	49
36	0.628	0.234	0.152	0.317	48
37	0.645	0.226	0.143	0.308	46
38	0.663	0.218	0.135	0.300	45
39	0.680	0.210	0.128	0.293	44
40	0.698	0.203	0.120	0.285	42
41	0.715	0.196	0.113	0.278	41
42	0.733	0.189	0.106	0.271	39
43	0.750	0.182	0.100	0.265	38
44	0.768	0.176	0.094	0.259	36
45	0.785	0.170	0.088	0.253	35

※W を 0.17[m]、T を 0.165[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。

11. 参考文献

・ルネサス エレクトロニクス(株)
R8C/38C グループ　ユーザーズマニュアル　ハードウェア編 Rev.1.10
・ルネサス エレクトロニクス(株)
M16C シリーズ,R8C ファミリ用 C/C++コンパイラパッケージ V.6.00
C/C++コンパイラユーザーズマニュアル Rev.1.00
・ルネサス エレクトロニクス(株)
High-performance Embedded Workshop V.4.09　ユーザーズマニュアル Rev.1.00
・ルネサス半導体トレーニングセンター　C言語入門コーステキスト　第 1 版
・電波新聞社　マイコン入門講座　大須賀威彦著　第 1 版
・ソフトバンク(株)　新C言語入門シニア編　林晴比古著　初版
・共立出版(株)　プログラマのための ANSI C 全書　L.Ammeraal 著
吉田敬一・竹内淑子・吉田恵美子訳　初版

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

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

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

http://japan.renesas.com/

の製品情報にある「マイコン」→「R8C」でご覧頂けます