

ミニマイコンカー製作キット Ver.2

マイコン 実習マニュアル R8C/35A 版

2013年度から、ミニマイコンカーVer.2(RMC-R8C35A ボード)に搭載されているマイコンが R8C/35A から R8C/35C に変更されました。R8C/35A マイコンと R8C/35C 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。

本マニュアルでは『R8C/35A』のマイコン名称が残っていますが、すべて『R8C/35A および R8C/35C』の意味になります。

第 1.09 版

2015.07.15

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

注意事項 (rev.6.0J)

著作権

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

禁止事項

ユーザーは以下の内容を行うことはできません。

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

転載、複製

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

責任の制限

本マニュアルに記載した情報は、正確を期すため、慎重に制作した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いたします。
- ・すべての商標および登録商標は、それぞれの所有者に帰属します。

連絡先

株式会社 日立ドキュメントソリューションズ

〒135-0016 東京都江東区東陽六丁目 3 番 2 号 イースト 21 タワー

E-mail:himdx.m-carrally.dd@hitachi.com

目次

1. ミニマイコンカーVer.2 の仕様	1
1.1 概要.....	1
1.2 仕様.....	2
1.3 外観.....	3
1.4 基板の分離.....	4
1.5 接続図.....	4
1.6 マイコンのポート表.....	7
1.7 センサ部のコネクタピン配置.....	8
1.7.1 外観.....	8
1.7.2 J1 の内容.....	8
1.7.3 D1～D4 の点灯方法.....	9
1.7.4 分離時の接続方法.....	11
1.8 マイコン部のコネクタピン配置.....	12
1.8.1 外観.....	12
1.8.2 J2 の内容.....	13
1.8.3 J3 の内容.....	13
1.8.4 J6 の内容.....	14
1.8.5 J7 の内容.....	14
1.8.6 オプション部品.....	15
1.9 モータドライブ部のコネクタピン配置.....	16
1.9.1 外観.....	16
1.9.2 J9 の詳細.....	16
1.9.3 分離時の接続方法.....	18
2. R8C/35A マイコンの仕様	21
2.1 シリーズ展開.....	21
2.2 R8C/35A の特徴.....	22
2.3 機能.....	23
2.4 ブロック図.....	24
2.5 ピン配置.....	25
2.6 メモリマップ.....	26
2.7 固定割り込みベクタテーブル領域.....	27
2.8 CPU レジスタ.....	28
2.9 タイマの概要.....	30
2.10 ミニマイコンカーVer.2 で使用する内蔵周辺機能.....	30
3. 開発環境のダウンロード、インストール	31
3.1 MY Renesas に登録.....	31
3.2 ルネサス統合開発環境のダウンロード.....	32
3.3 ルネサス統合開発環境のインストール.....	34
3.4 ショートカットの作成.....	36
3.5 R8C Writer のダウンロード.....	37
3.6 R8C Writer のインストール.....	38
3.7 R8C Writer をルネサス統合開発環境に登録する.....	39
3.8 ドライバのインストール.....	42
3.9 拡張子の表示.....	47

3.9.1 WindowsXP の場合	47
3.9.2 WindowsVista の場合	48
4. サンプルプログラムのダウンロード、インストール	50
4.1 サンプルプログラムのダウンロード	50
4.2 サンプルプログラムのインストール	51
5. 演習手順	52
5.1 ワークスペースを開く	52
5.2 プロジェクトを開く	53
5.3 ファイル編集	55
5.4 ビルド(MOT ファイルの作成)	58
5.5 プログラムの書き込み	61
6. 実習基板 Ver.2	65
6.1 仕様	65
6.2 外観	66
6.3 スライドスイッチ部	67
6.3.1 コネクタ	67
6.3.2 回路図	68
6.3.3 動作原理	68
6.4 LED 部	69
6.4.1 コネクタ	69
6.4.2 回路図	70
6.4.3 動作原理	70
6.5 7セグメントLED 部	71
6.5.1 コネクタ	71
6.5.2 回路図	72
6.5.3 動作原理	74
6.6 ブザー部	75
6.6.1 コネクタ	75
6.6.2 回路図	76
6.6.3 動作原理	76
6.7 トグルスイッチ部	77
6.7.1 コネクタ	77
6.7.2 回路図	78
6.7.3 動作原理	78
6.8 ボリューム部	81
6.8.1 コネクタ	81
6.8.2 回路図	82
6.8.3 動作原理	82
6.9 SW10～SW13 を切り替えるときの注意点	83
7. プロジェクトの内容	84
8. 電源が入ってからのマイコンの動作	86
8.1 概要	86
8.2 ファイルの構成	86
8.3 動作手順	87
8.3.1 リセットの種類	87

8.3.2	電源投入してからプログラム実行までの動作.....	88
8.4	プログラム「startup.c」.....	89
8.5	セクション.....	91
8.5.1	概要.....	91
8.5.2	セクションの種類.....	92
8.5.3	プログラムをセクションに分類.....	94
8.5.4	アドレスに配置.....	96
8.5.5	セクションの設定.....	97
8.6	プログラムの解説「startup.c」.....	98
8.6.1	スタックサイズの設定.....	98
8.6.2	CPUレジスタの宣言.....	99
8.6.3	SB の値をコンパイラに設定.....	100
8.6.4	オプション機能選択レジスタの設定.....	100
8.6.5	ID コードの設定.....	102
8.6.6	RAM の初期化.....	105
8.6.7	セクションの先頭アドレスの型定義.....	107
8.6.8	ヒープ領域の設定.....	108
8.6.9	固定割り込みベクタアドレスの設定.....	109
8.6.10	可変割り込みベクタの設定.....	114
8.6.11	スタートアッププログラム(start 関数).....	114
8.6.12	SB 相対アドレッシング.....	116
8.6.13	switch_table セクション.....	117
9.	I/O ポートの入出力(プロジェクト:io).....	120
9.1	概要.....	120
9.2	接続.....	120
9.3	プロジェクトの構成.....	121
9.4	プログラム「io.c」.....	121
9.5	プログラムの解説.....	122
9.5.1	ヘッダファイルの取り込み.....	122
9.5.2	init 関数(クロックの選択).....	123
9.5.3	init 関数(I/O ポートの入出力設定).....	132
9.5.4	main 関数.....	140
9.6	ビット操作.....	141
9.6.1	特定のビットを"0"にする(マスク処理).....	141
9.6.2	特定のビットを"1"にする.....	143
9.6.3	特定のビットを反転する.....	144
9.6.4	全ビット反転する(NOT 演算).....	144
9.6.5	ビットシフトする.....	145
9.7	ビット単位でデータを入力、出力する.....	146
9.8	演習.....	147
10.	I/O ポートの入出力 2(プロジェクト:io2).....	148
10.1	概要.....	148
10.2	接続.....	148
10.3	プロジェクトの構成.....	149
10.4	プログラム「io2.c」.....	149
10.5	プログラムの解説.....	151
10.5.1	dipsw_get 関数.....	151
10.5.2	led_out 関数.....	153

10.5.3 main 関数	155
10.6 演習	155
11. プッシュスイッチの情報入力(プロジェクト:pushsw)	156
11.1 概要	156
11.2 接続	156
11.3 プロジェクトの構成	157
11.4 プログラム「pushsw.c」	157
11.5 プログラムの解説	159
11.5.1 pushsw_get 関数	159
11.5.2 main 関数	160
11.6 演習	161
12. マイクロスイッチの情報入力(プロジェクト:microsw)	162
12.1 概要	162
12.2 接続	162
12.3 プロジェクトの構成	163
12.4 プログラム「microsw.c」	163
12.5 プログラムの解説	165
12.5.1 microsw_get 関数	165
12.5.2 main 関数	166
12.6 演習	167
13. ソフトウェアによるタイマ(プロジェクト:timer1)	168
13.1 概要	168
13.2 接続	168
13.3 プロジェクトの構成	169
13.4 プログラム「timer1.c」	169
13.5 プログラムの解説	171
13.5.1 timer 関数(時間稼ぎ)	171
13.5.2 main 関数	172
13.6 演習	173
14. 割り込みによるタイマ(プロジェクト:timer2)	174
14.1 概要	174
14.2 接続	174
14.3 プロジェクトの構成	175
14.4 プログラム「timer2.c」	175
14.5 プログラムの解説	177
14.5.1 割り込みとは	177
14.5.2 init 関数(タイマ RB の設定)	180
14.5.3 intTRB 関数(1ms ごとに実行される関数)	189
14.5.4 timer 関数(割り込みを使った時間稼ぎ)	191
14.5.5 main 関数	192
14.5.6 割り込みの発生タイミング	193
15. センサ部の LED へ出力(プロジェクト:sensor_led)	196
15.1 概要	196
15.2 接続	196
15.3 プロジェクトの構成	197

15.4	プログラム「sensor_led.c」.....	197
15.5	センサ部の LED の回路	199
15.5.1	マイコンボードとセンサ部の LED の結線.....	199
15.5.2	J1 と J3 の詳細.....	200
15.5.3	ポートの信号レベルと LED の関係.....	201
15.6	プログラムの解説.....	201
15.6.1	グローバル変数の宣言	201
15.6.2	sensorled_out 関数	202
15.6.3	割り込み関数.....	202
15.6.4	main 関数	204
15.7	演習.....	204
16.	外部割り込み(プロジェクト:int_interrupt)	205
16.1	概要.....	205
16.2	接続.....	205
16.3	プロジェクトの構成.....	206
16.4	プログラム「int_interrupt.c」.....	206
16.5	プログラムの解説.....	208
16.5.1	init 関数(INT3割り込みの設定)	208
16.5.2	intINT3 関数(INT3端子に立ち下がりエッジの信号が入力されたときに実行される関数)	216
16.5.3	main 関数	218
16.6	演習.....	218
17.	A/D コンバータ(単発モード)(プロジェクト:ad)	219
17.1	概要.....	219
17.2	接続.....	219
17.3	プロジェクトの構成.....	221
17.4	プログラム「ad.c」	221
17.5	プログラムの解説.....	222
17.5.1	init 関数	222
17.5.2	get_ad7 関数(A/D コンバータの設定、A/D 値取得).....	223
17.5.3	main 関数	232
17.6	演習.....	232
18.	A/D コンバータ(繰り返しモード 0)(プロジェクト:ad_kurikaeshi)	233
18.1	概要.....	233
18.2	接続.....	233
18.3	プロジェクトの構成.....	233
18.4	プログラム「ad_kurikaeshi.c」.....	234
18.5	プログラムの解説.....	235
18.5.1	init 関数(I/O ポートの入出力設定)	235
18.5.2	init 関数(A/D コンバータの設定).....	236
18.5.3	get_ad7 関数.....	241
18.5.4	main 関数	242
18.6	演習.....	242
19.	A/D コンバータ(繰り返し掃引モード)(プロジェクト:ad_kurikaeshi_souin)	243
19.1	概要.....	243
19.2	接続.....	243
19.3	プロジェクトの構成.....	245

19.4	プログラム「ad_kurikaeshi_souin.c」	245
19.5	プログラムの解説	247
19.5.1	init 関数(I/O ポートの入出力設定)	247
19.5.2	init 関数(A/D コンバータの設定)	248
19.5.3	main 関数	253
19.6	演習	254
20.	パルスカウント(プロジェクト:timer_ra_counter)	255
20.1	概要	255
20.2	接続	255
20.3	プロジェクトの構成	256
20.4	プログラム「timer_ra_counter.c」	257
20.5	プログラムの解説	258
20.5.1	init 関数(タイマ RA の設定)	258
20.5.2	main 関数	265
21.	タイマ RD による PWM 波形出力(リセット同期 PWM モード)(プロジェクト:timer_rd_doukipwm)	266
21.1	概要	266
21.2	接続	266
21.3	プロジェクトの構成	267
21.4	プログラム「timer_rd_doukipwm.c」	267
21.5	PWM とは?	269
21.6	プログラムの解説	271
21.6.1	init 関数(タイマ RD の設定)	271
21.6.2	main 関数	293
21.7	演習	295
22.	タイマ RD による PWM 波形出力(PWM モード)(プロジェクト:timer_rd_pwm6)	296
22.1	概要	296
22.2	接続	296
22.3	プロジェクトの構成	297
22.4	プログラム「timer_rd_pwm6.c」	297
22.5	プログラムの解説	299
22.5.1	init 関数(タイマ RD の設定)	299
22.5.2	main 関数	317
22.6	演習	321
23.	モータの制御(プロジェクト:motor)	322
23.1	概要	322
23.2	接続	322
23.3	プロジェクトの構成	323
23.4	プログラム「motor.c」	323
23.5	モータドライブ部	326
23.5.1	マイコン部とモータドライブ部の接続	326
23.5.2	J7 と J9 の詳細	327
23.5.3	左モータの動作	328
23.5.4	右モータの動作	328
23.6	プログラムの解説	329
23.6.1	init 関数(タイマ RD の設定)	329
23.6.2	motor 関数	341

23.6.3 main 関数	347
24. サーボの制御(プロジェクト:servo)	348
24.1 概要	348
24.2 接続	348
24.3 プロジェクトの構成	349
24.4 プログラム「servo.c」	349
24.5 サーボ	351
24.5.1 サーボの接続	351
24.5.2 J7 と J9 の詳細	352
24.6 プログラムの解説	353
24.6.1 init 関数(タイマ RD の設定)	353
24.6.2 handle 関数	355
24.6.3 main 関数	357
25. 通信(プロジェクト:uart0)	358
25.1 概要	358
25.2 接続	358
25.3 プロジェクトの構成	359
25.4 プログラム「uart0.c」	359
25.5 パソコンとミニマイコンカーVer.2 の通信	362
25.5.1 ミニマイコンカーVer.2 の通信方法	362
25.5.2 COM ポートの確認	362
25.5.3 シリアル通信の設定	363
25.6 プログラムの解説	365
25.6.1 init 関数(UART0 の設定)	365
25.6.2 get_uart0 関数	372
25.6.3 put_uart0 関数	375
25.6.4 main 関数	377
25.7 実習手順	378
25.7.1 パソコン設定する前準備	378
25.7.2 TeraTerm のインストール	378
25.7.3 TeraTerm を使って、マイコンと通信しよう	383
25.7.4 TeraTerm に表示される文字について	385
25.8 演習	386
26. printf、scanf 文を使った通信(プロジェクト:uart0_printf)	387
26.1 概要	387
26.2 接続	387
26.3 プロジェクトの構成	388
26.4 プログラム「uart0_printf.c」	388
26.5 printf 文、scanf 文を使おう！	390
26.6 プログラムの解説「printf_lib.c」	391
26.6.1 「printf_lib.c」の追加	391
26.6.2 関数一覧	393
26.7 プログラムの解説「uart0_printf.c」	394
26.7.1 インクルード部分	394
26.7.2 main 関数－初期化部分	394
26.7.3 main 関数－通信部分	395
26.7.4 受信バッファのクリア	395

27. データフラッシュ(プロジェクト: data_flash)	397
27.1 概要	397
27.2 接続	397
27.3 プロジェクトの構成	398
27.4 プログラム「data_flash.c」	398
27.5 データフラッシュを使おう！	401
27.5.1 概要	401
27.5.2 メモリマップ	402
27.5.3 データフラッシュを効率的に使う	403
27.6 プログラムの解説「data_flash_lib.c」	404
27.6.1 「data_flash_lib.c」の追加	404
27.6.2 関数一覧	406
27.7 プログラムの解説「data_flash.c」	407
27.7.1 インクルード部分	407
27.7.2 変数領域	408
27.7.3 main 関数(データフラッシュから読み込み)	408
27.7.4 main 関数(データフラッシュに書き込み)	409
27.7.5 main 関数(無限ループ部分)	410
27.8 実習手順	411
27.9 演習	412
28. タイマ RC によるブザー制御(PWM 波形出力)(プロジェクト: timer_rc_pwm)	413
28.1 概要	413
28.2 接続	413
28.3 プロジェクトの構成	414
28.4 プログラム「timer_rc_pwm.c」	415
28.5 プログラムの解説	419
28.5.1 init 関数(タイマ RC の設定)	419
28.5.2 beep 関数	435
28.5.3 intTRB 関数(タイマ RB 割り込み)	438
28.5.4 main 関数	439
28.6 音階	439
28.6.1 音階の周波数	439
28.6.2 音階名の定義	441
28.6.3 音階の時間の長さ	442
28.6.4 音データ	443
29. 付録	444
29.1 R8C マイコンの変数のサイズ	444
29.2 演算子	445
29.3 優先順位	446
29.4 型が混合したときの演算	447
29.5 printf 関数の使い方	448
29.6 scanf 関数の使い方	450
30. 参考文献	451

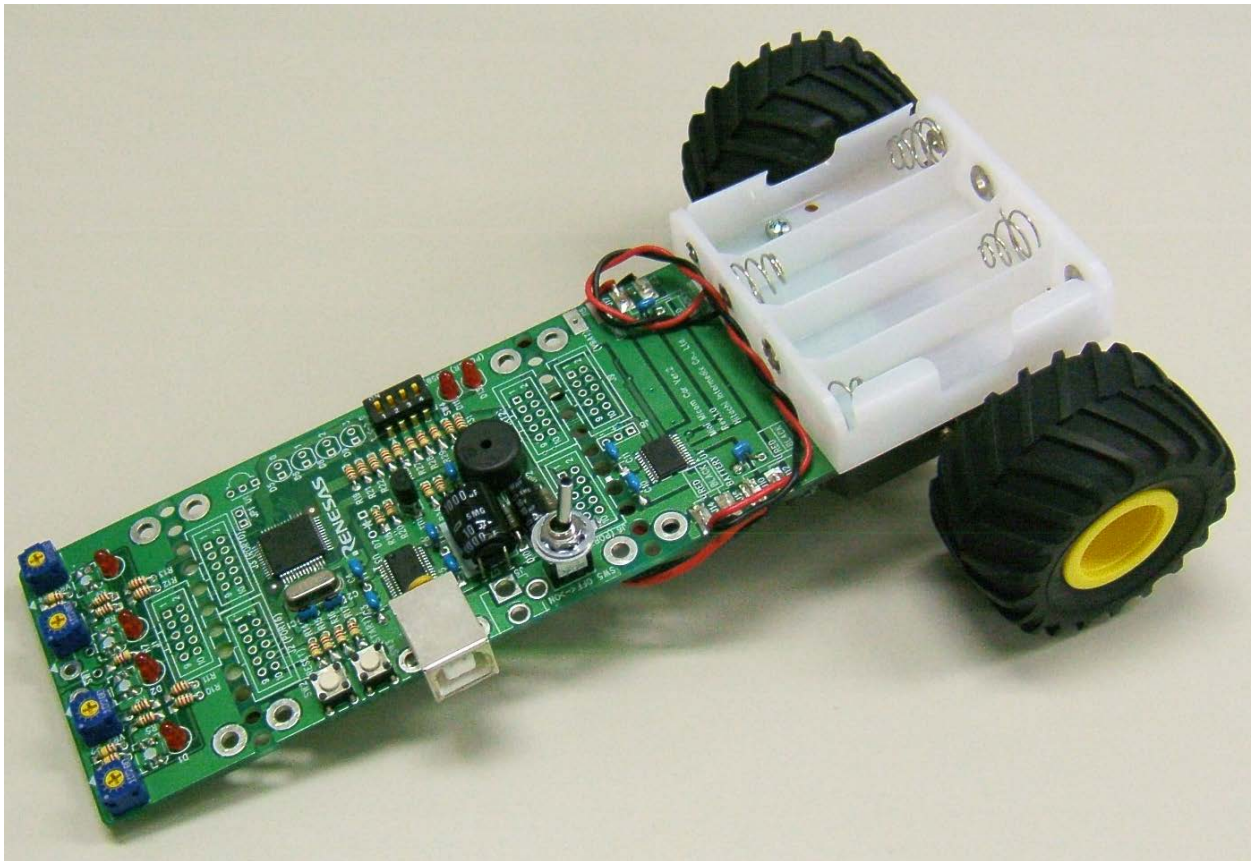
1. ミニマイコンカーVer.2 の仕様

1.1 概要

ミニマイコンカーVer.2 は、ルネサス エレクトロニクス製の R8C/35A マイコンを搭載した自走式のライトレースカーです。

特徴を下記に示します。

- R8C/35A マイコンを搭載しています。
- パソコンからミニマイコンカーへのプログラム書き込みは、USB で行います。
- センサ 4 個で白・黒のラインを検出し走行します。
- ブロックソフト(ブロック・コマンダー)、または C 言語(ルネサス統合開発環境)でプログラムすることができます。
- マイコン学習に必要なデバッグスイッチ、LED、圧電サウンダ(以下、ブザーといいます)などを搭載しています。
- センサ部分、モータドライブ部分を分離して、マイコンボードとして使用可能です。



▲ミニマイコンカーVer.2(オプションを取り付けていない状態)

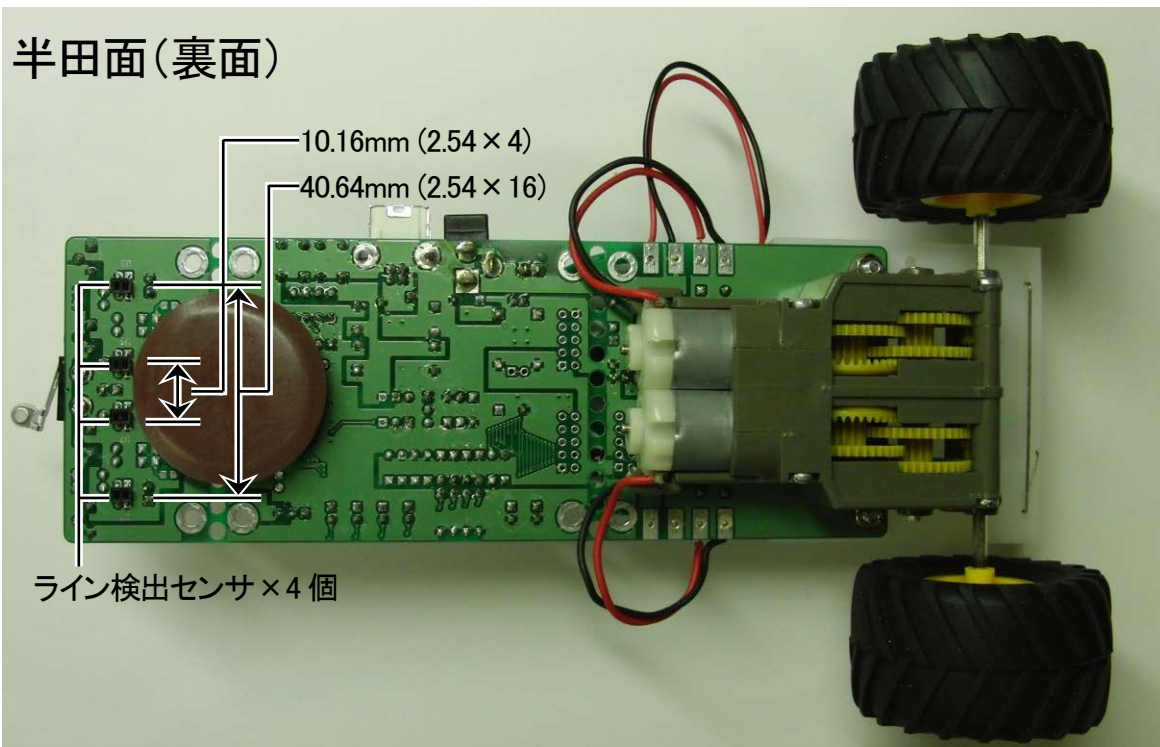
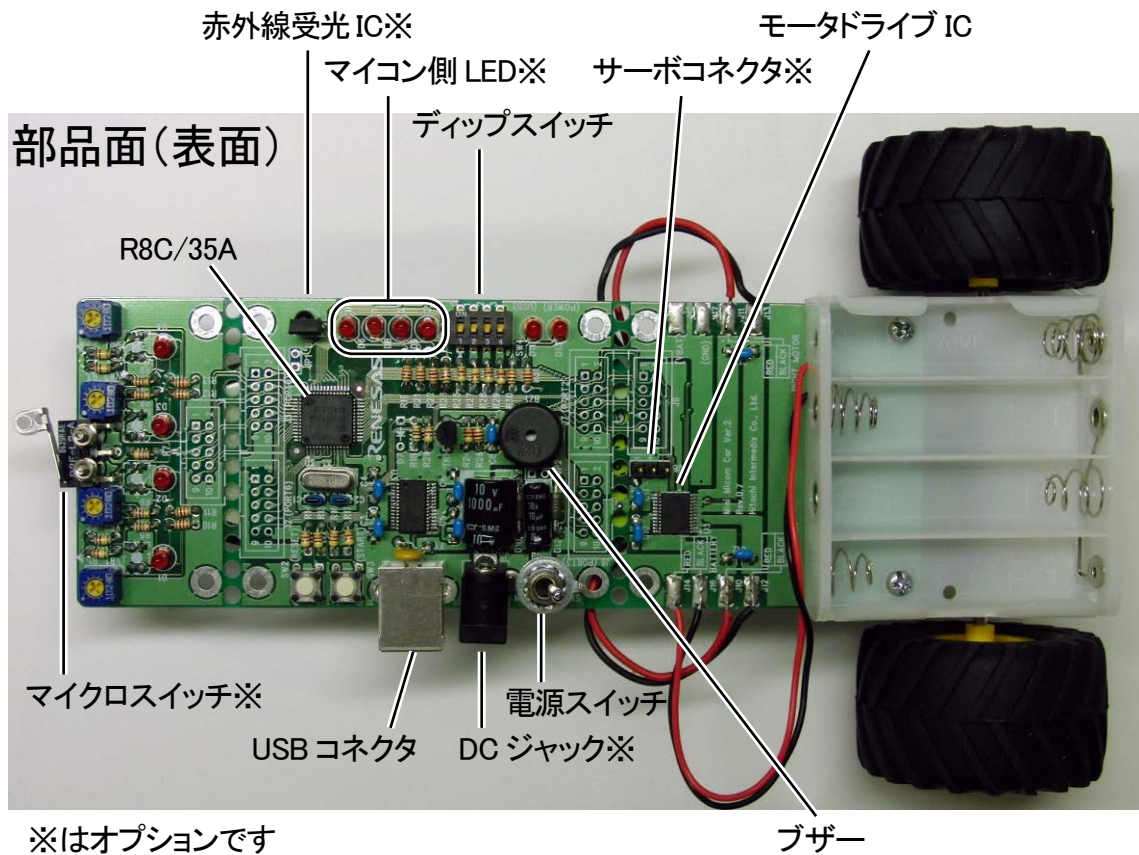
1.2 仕様

内容	詳細
マイコン	ルネサス エレクトロニクス製 R8C/35A
電源	単三電池 4 本(アルカリ電池または充電電池) 別売り DC ジャックコネクタと AC アダプタを使用することにより、商用電源(AC100V)で動作することもできます。
開発環境	ブロック・コマンダーを使ったブロックの組み合わせによるプログラム開発、またはルネサス統合開発環境を使った C 言語によるプログラム開発 ※各ソフトは、web サイトよりダウンロード可能です。
プログラム書き込み	パソコンの USB コネクタ経由にて書き込み ※USB ケーブルは、AB タイプが接続可能です。 ※書き込みソフトは、ジャパンマイコンカーラリー実行委員会が提供している R8C Writer を使用します。
組み立て内容	電子部品の半田付け(面実装部品は実装済み)、ギヤーボックス、タイヤの組み立て
ギヤーボックス	タミヤのツインモータギヤーボックス
モータ	ツインモータギヤーボックス付属の FA130 モータを 2 個
タイヤ	タミヤのトラックタイヤセット
I/O	<ul style="list-style-type: none"> ● ライン検出センサ×4 個 ● ライン検出センサ用 LED×4 個 ● マイコン側 LED×4 個(オプション) ● 4bit ディップスイッチ×1 個 ● タクトスイッチ×1 個 ● ブザー×1 個 ● モータ制御×2 個 ● 障害物検出用マイクロスイッチ×1 個(オプション) ● 赤外線受光 IC×1 個(オプション) ● サーボコネクタ×1 個(オプション) ● 拡張 I/O コネクタ×4 個(オプション) <p>他に、下記がありますが、マイコンの I/O ポートとは接続されていません。</p> <ul style="list-style-type: none"> ● マイコンリセット用のタクトスイッチ ● 電源モニタ用(POWER と表示)の LED ● USB 電源モニタ用(USB と表示)の LED
その他	センサ部分、モータドライブ部分を分離して、マイコン部をマイコンボードとして使用可能です。

1. ミニマイコンカーVer.2 の仕様

1.3 外観

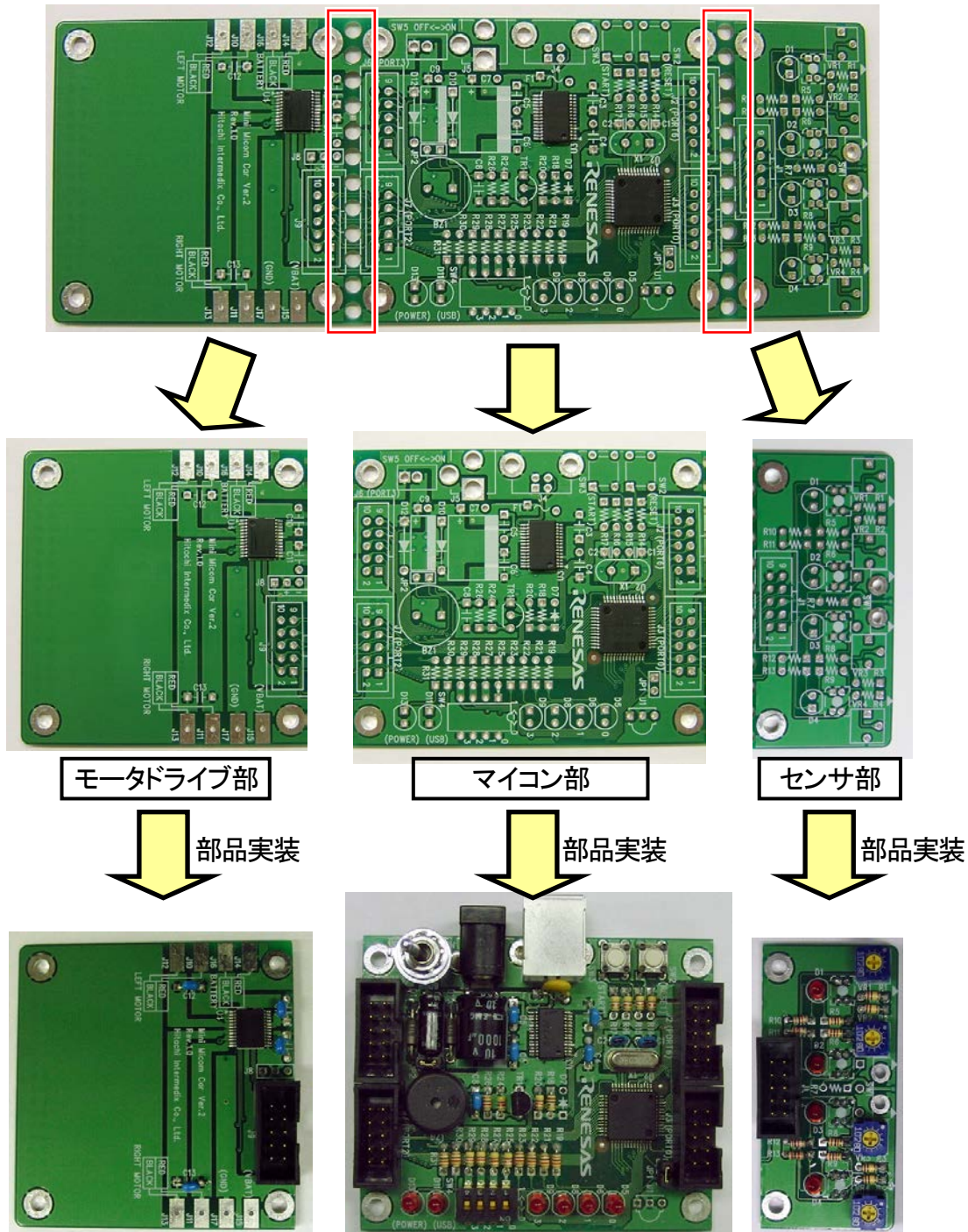
ミニマイコンカーVer.2 の外観を、下記に示します。



1. ミニマイコンカーVer.2 の仕様

1.4 基板の分離

基板に開いている穴部分(写真の四角で囲った部分)をニップなどで切断、ヤスリがけすると、モータドライブ部、マイコン部、センサ部の3つの基板に分離することができます。分離後は電氣的に切れますが、フラットケーブルなどでそれぞれの基板間を接続すると、電氣的に繋ぐことができます。

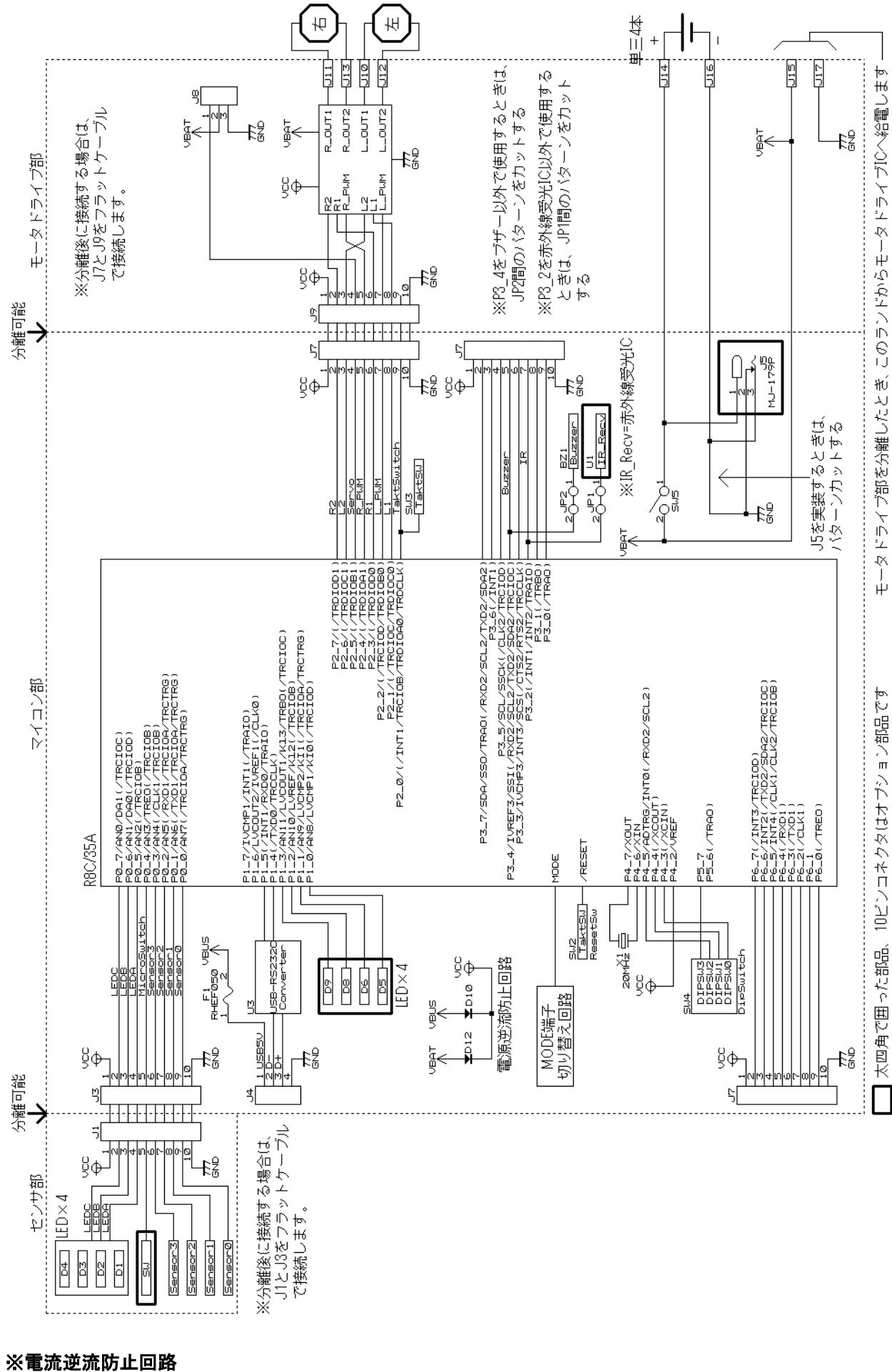


※写真で実装している部品は、標準品以外の部品も含まれます。

1.5 接続図

ミニマイコンカーVer.2の簡単な接続図を示します。センサ部、モータドライブ部を分離すると電氣的に切れます。再度、電氣的に接続したい場合は10ピンコネクタ同士をフラットケーブルで接続します。

1. ミニマイコンカーVer.2の仕様



※電流逆流防止回路

1. ミニマイコンカーVer.2 の仕様

ミニマイコンカーVer.2 は、USB ケーブル、電池の 2 系統から給電されます。接続図の VBAT、VBUS、VCC がプラス側の電源です。意味を下表に示します。

電源の名称	意味
VBAT	電池の電圧です。
VBUS	USB ケーブルの電圧です。
VCC	VBAT と VBUS の電圧が高い方の電圧です。

VCC は、2 系統の高い方の電圧が供給されます。その流れを下図に示します。

	VBAT>VBUS の場合(VBUS=0V も含む)	VBUS>VBAT の場合(VBAT=0V も含む)
電流の流れ		
VCC の電圧	VCC には、ダイオードを通した VBAT の電圧が給電されるので、 VCC = VBAT - 0.6V となります。	VCC には、ダイオードを通した VBUS の電圧が給電されるので、 VCC = VBUS - 0.6V となります。

※MODE 端子

MODE 端子は、0V の状態で電源を入れる(またはリセット解除する)とマイコンが書き込みモードで起動し、5V の状態で電源を入れる(またはリセット解除する)と書き込んだプログラムを実行します。

VBAT と VBUS の状態と MODE 端子の電圧の関係を下表に示します。

VBAT=5V , VBUS=給電無し	VBAT=給電無し , VBUS=5V	VBAT=5V , VBUS=5V
MODE = 5V	MODE = 0V	MODE=4.7K/(4.7K+100)*5V=4.89V

VBAT(電池)から給電されている場合は、VBUS の電圧に関わらずプログラムを実行します。

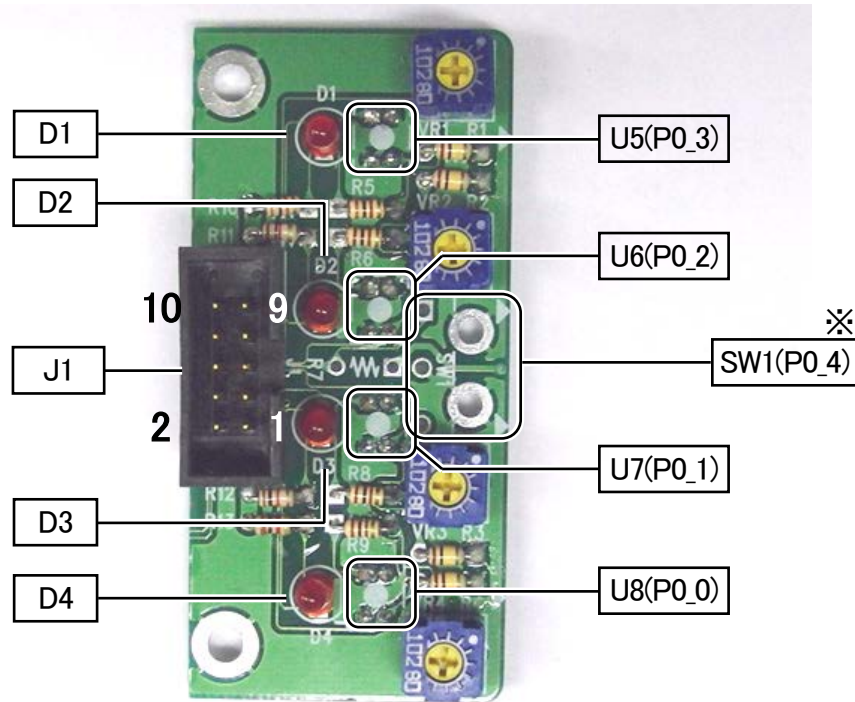
1. ミニマイコンカーVer.2 の仕様

1.6 マイコンのポート表

コネクタ	pin	Port	bit	端子名	接続先	備考
J3	2	P0	7	P0_7/AN0/DA1 (/TRCIOC)	LEDC	センサ部を分離すれば自由に使用できます。
	3	P0	6	P0_6/AN1/DA0 (/TRCIOD)	LEDB	
	4	P0	5	P0_5/AN2 (/TRCIOB)	LEDA	
	5	P0	4	P0_4/AN3/TREO (/TRCIOB)	リミットスイッチ	
	6	P0	3	P0_3/AN4 (/CLK1/TRCIOB)	センサ3	
	7	P0	2	P0_2/AN5 (/RXD1/TRCIOA/TRCTRG)	センサ2	
	8	P0	1	P0_1/AN6 (/TXD1/TRCIOA/TRCTRG)	センサ1	
	9	P0	0	P0_0/AN7 (/TRCIOA/TRCTRG)	センサ0	
		P1	7	P1_7/IVCMP1/INT1 (/TRAIO)		
		P1	6	P1_6/LVCOUT2/IVREF1 (/CLK0)		
		P1	5	P1_5 (/INT1/RXD0/TRAIO)	RxD0	
		P1	4	P1_4 (/TXD0/TRCCLK)	TxD0	
		P1	3	P1_3/AN11/LVCOUT1/K13/TRB0 (/TRCIOC)	LED3	
		P1	2	P1_2/AN10/LVREF/K12 (/TRCIOB)	LED2	
		P1	1	P1_1/AN9/LVCMP2/K11 (/TRCIOA/TRCTRG)	LED1	
		P1	0	P1_0/AN8/LVCMP1/K10 (/TRCIOD)	LED0	
J7	2	P2	7	P2_7 (/TRDIOD1)	モータ右2	モータドライブ部を分離すれば自由に使用できます。
	3	P2	6	P2_6 (/TRDIOC1)	モータ左2	
	4	P2	5	P2_5 (/TRDIOB1)	サーボ (/TRDIOB1)	
	5	P2	4	P2_4 (/TRDIOA1)	モータ右PWM (/TRDIOA1)	
	6	P2	3	P2_3 (/TRDIOD0)	モータ右1	
	7	P2	2	P2_2 (/TRCIOD/TRDIOB0)	モータ左PWM (/TRDIOB0)	
	8	P2	1	P2_1 (/TRCIOC/TRDIOC0)	モータ左1	
	9	P2	0	P2_0 (/INT1/TRCIOB/TRDIOA0/TRDCLK)	プッシュスイッチ	
J6	2	P3	7	P3_7/SDA/SS0/TRA0 (/RXD2/SCL2/TXD2/SDA2)		
	3	P3	6	P3_6 (/INT1)		
	4	P3	5	P3_5/SCL/SSCK (/CLK2/TRCIOD)		
	5	P3	4	P3_4/IVREF3/SSI (/RXD2/SCL2/TXD2/SDA2/TRCIOC)	ブザー (TRCIOC)	JP2間の半田面パターンを切断すると自由に使用可能
	6	P3	3	P3_3/IVCMP3/INT3/SCS (/CTS2/RTS2/TRCCLK)		
	7	P3	2	P3_2 (/INT1/INT2/TRAIO)	赤外線受信 (TRAIO)	JP1間の半田面パターンを切断すると自由に使用可能
		P3	1	P3_1 (/TRB0)		
		P3	0	P3_0 (/TRA0)		
		P4	7	P4_7/XOUT	クリスタル (XOUT)	
		P4	6	P4_6/XIN	クリスタル (XIN)	
		P4	5	P4_5/ADTRG/INT0 (/RXD2/SCL2)	ディップスイッチ2	
		P4	4	P4_4 (/XCOUT)	ディップスイッチ1	
		P4	3	P4_3 (/XCIN)	ディップスイッチ0	
		P4	2	P4_2/VREF	Vcc	
		P5	7	P5_7	ディップスイッチ3	
		P5	6	P5_6 (/TRA0)		
J2	2	P6	7	P6_7 (/INT3/TRCIOD)		
	3	P6	6	P6_6/INT2 (/TXD2/SDA2/TRCIOC)		
	4	P6	5	P6_5/INT4 (/CLK1/CLK2/TRCIOB)		
	5	P6	4	P6_4 (/RXD1)		
	6	P6	3	P6_3 (/TXD1)		
	7	P6	2	P6_2 (/CLK1)		
	8	P6	1	P6_1		
	9	P6	0	P6_0 (/TREO)		

1.7 センサ部のコネクタピン配置

1.7.1 外観



※ポートは、J1 とマイコンボードの J3(ポート 0)をフラットケーブルで接続した場合です。

※SW1 のマイクロスイッチはオプションです。

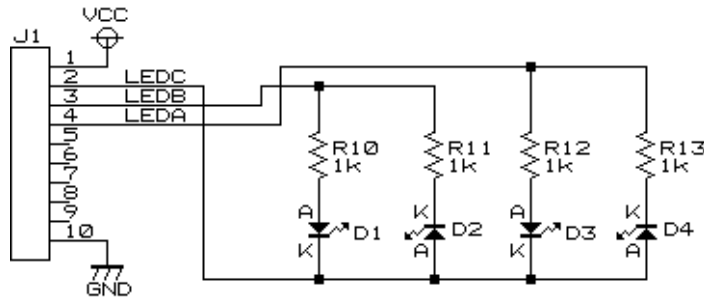
1.7.2 J1 の内容

ピン番号	詳細	"0"	"1"	マイコンの 接続先	備考	
1	VCC(+5V)					
2	LED_C	LEDA、LEDB、LEDC の信号レベルにより、D1～D4 の LED をどう点灯させるか選択します。		P0_7	赤外線フォトインタラプタからは、0～5V の信号が出力されます。マイコンは、2.5V 以下なら白、以上なら黒と判断します。	
3	LED_B			P0_6		
4	LED_A			P0_5		
5	マイクロスイッチ (SW1)	ON	OFF	P0_4		マイクロスイッチはオプションです
6	赤外線フォトインタラプタ (U5)	白	黒(未反応)	P0_3		
7	赤外線フォトインタラプタ (U6)	白	黒(未反応)	P0_2		
8	赤外線フォトインタラプタ (U7)	白	黒(未反応)	P0_1		
9	赤外線フォトインタラプタ (U8)	白	黒(未反応)	P0_0		
10	GND					

※マイコンの接続先は、センサ部とマイコン部を分離していない場合、パターンで接続されています。分離した場合は、J1 と J3 をフラットケーブルなどで接続してください。

1.7.3 D1～D4 の点灯方法

J1 の残っているピン数の関係から、D1～D4 の 4 個の LED を 3 つの信号の組み合わせで制御しています。



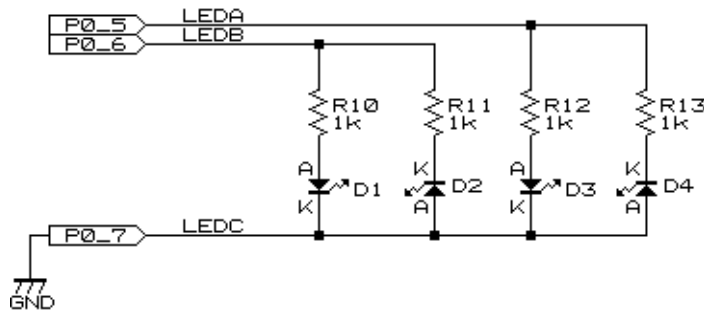
LED の電流制限抵抗は次の計算で求めることができます。

$$R = \frac{V_{cc} - \text{LEDの順電圧}}{I}$$

今回の抵抗は 1kΩ です。式を変形して、電流を求めます。

$$I = \frac{V_{cc} - \text{LEDの順電圧}}{R} = \frac{5 - 1.7}{1000} = 3.3\text{mA}$$

(1) P0_7=0V("0")のとき



P0_5 信号で、D3 の LED の点灯の仕方を変えます。P0_7 信号線が 0V("0")のとき、D4 は常に消灯です。D3 を点灯するときは P0_5 を 5V("1")に、D3 を消灯するときは P0_5 を 0V("0")にします。

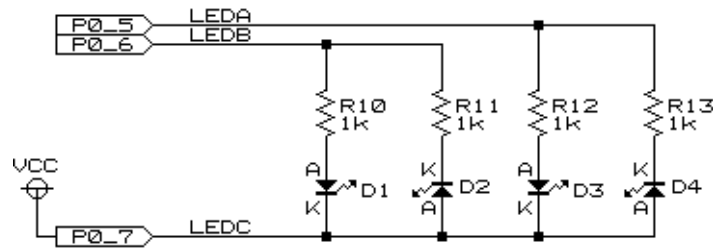
P0_5 (LEDA)	D3	D4
0V("0")	消灯	消灯
5V("1")	点灯	消灯

P0_6 信号は、D1 の LED の点灯の仕方を変えます。P0_7 信号線が 0V("0")のとき、D2 は常に消灯です。D1 を点灯するときは P0_6 を 5V("1")に、D1 を消灯するときは P0_6 を 0V("0")にします。

P0_6 (LEDB)	D1	D2
0V("0")	消灯	消灯
5V("1")	点灯	消灯

1. ミニマイコンカーVer.2 の仕様

(2) P0_7=5V("1")のとき



P0_5 信号で、D4 の LED の点灯の仕方を変えます。P0_7 信号線が 5V("1")のとき、D3 は常に消灯です。D4 を点灯するときは P0_5 を 0V("0")に、D4 を消灯するときは P0_5 を 5V("1")にします。

P0_5 (LEDA)	D3	D4
0V("0")	消灯	点灯
5V("1")	消灯	消灯

P0_6 信号で、D2 の LED の点灯の仕方を変えます。P0_7 信号線が 5V("1")のとき、D1 は常に消灯です。D2 を点灯するときは P0_6 を 0V("0")に、D2 を消灯するときは P0_6 を 5V("1")にします。

P0_6 (LEDB)	D1	D2
0V("0")	消灯	点灯
5V("1")	消灯	消灯

(3) まとめ

P0_7 LEDC	P0_6 LEDB	P0_5 LEDA	D1	D2	D3	D4
0V("0")	0V("0")	0V("0")	消灯	消灯	消灯	消灯
0V("0")	0V("0")	5V("1")	消灯	消灯	点灯	消灯
0V("0")	5V("1")	0V("0")	点灯	消灯	消灯	消灯
0V("0")	5V("1")	5V("1")	点灯	消灯	点灯	消灯
5V("1")	0V("0")	0V("0")	消灯	点灯	消灯	点灯
5V("1")	0V("0")	5V("1")	消灯	点灯	消灯	消灯
5V("1")	5V("1")	0V("0")	消灯	消灯	消灯	点灯
5V("1")	5V("1")	5V("1")	消灯	消灯	消灯	消灯

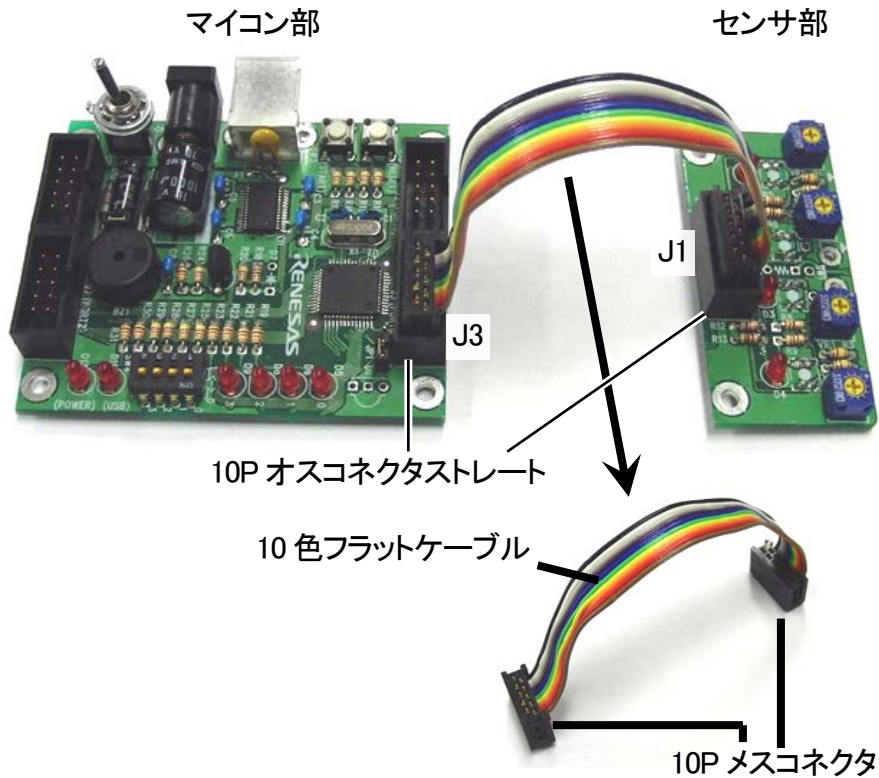
プログラムでは、次のように時間を分けて LED を制御します。

- 1ms の間、P0_7 を 0V にして、D1、D3 を点灯、または消灯させる。
- 1ms の間、P0_7 を 5V にして、D2、D4 を点灯、または消灯させる。

1. ミニマイコンカーVer.2 の仕様

1.7.4 分離時の接続方法

マイコン部、センサ部を分離したときに、これらの基板を接続する場合は、J1、J3 に 10 ピンストレートコネクタオスを実装し、フラットケーブルでこれらのコネクタ間を繋ぎます。

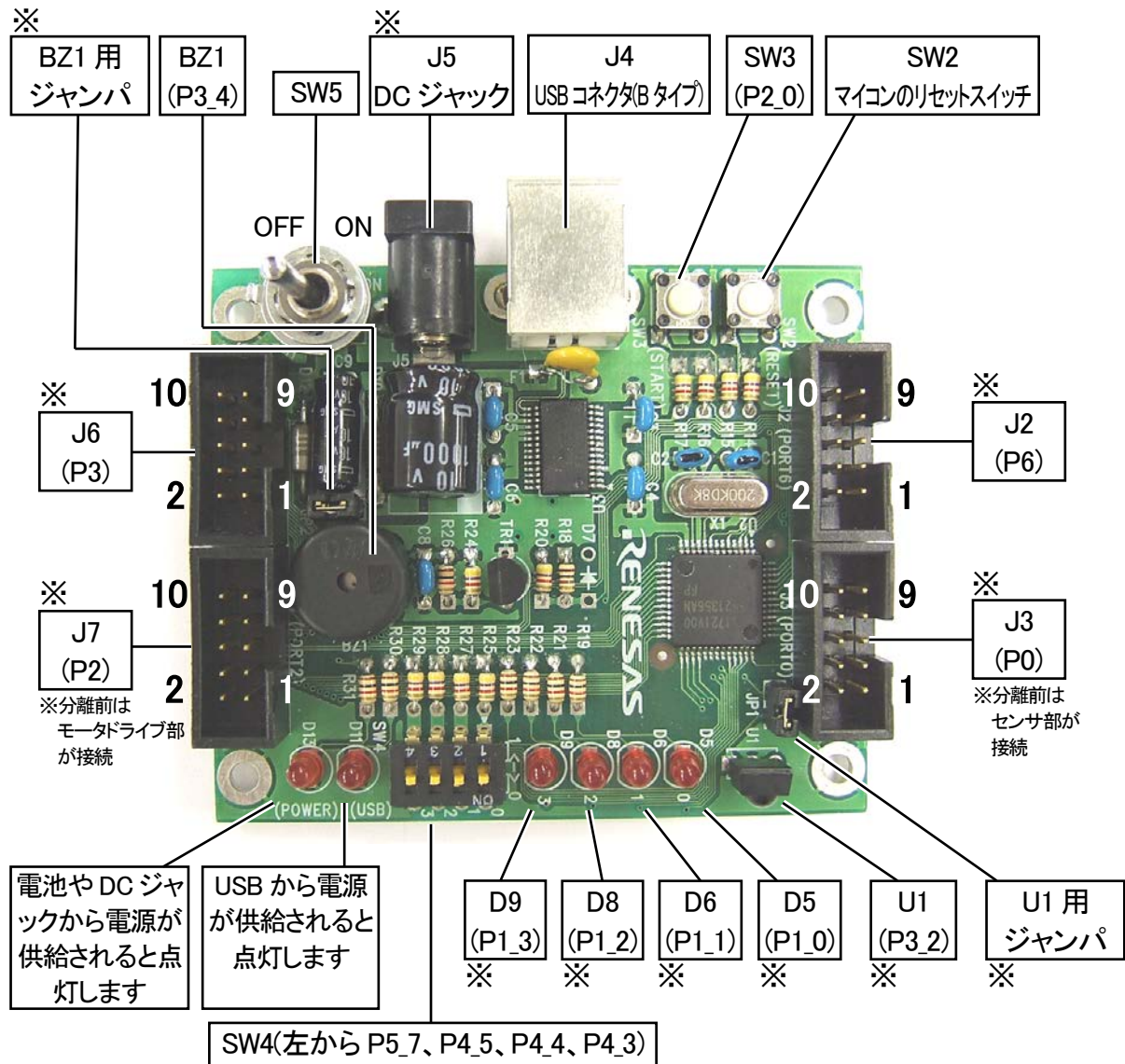


コネクタ、フラットケーブルは、ミニマイコンカーVer.2 には付属していません。これらの部品は、マイコンカーラリー販売サイト(<http://www2.himdx.net/mcr/>)などで販売しています。

部品名	型式	写真	マイコンカーラリー販売サイト 型式
10P オスコネクタ ストレート	ヒロセ電機(株) HIF3FC-10PA 2.54DSA		M-S42
10P メスコネクタ	オムロン(株) XG4M-1030		M-S44
10 色フラット ケーブル	1.27mm ピッチ		M-S45

1.8 マイコン部のコネクタピン配置

1.8.1 外観



※U1 (赤外線受光 IC)、J5 (DC ジャック)、JP1 (U1 切り離し用ジャンパ)、JP2 (BZ1 切り離し用ジャンパ)、D5、D6、D8、D9 (LED)、R19、R21、R22、R23 (LED の電流制限抵抗)、J2、J3、J6、J7 (10P オスコネクタストレート) はオプションです。

1. ミニマイコンカーVer.2 の仕様

1.8.2 J2 の内容

ピン番号	詳細	"0"	"1"	備考
1	VCC(+5V)			
2	P6_7			
3	P6_6			
4	P6_5			
5	P6_4			
6	P6_3			
7	P6_2			
8	P6_1			
9	P6_0			
10	GND			

1.8.3 J3 の内容

ピン番号	詳細	"0"	"1"	備考
1	VCC(+5V)			
2	P0_7			センサ部の J1.2 と接続
3	P0_6			センサ部の J1.3 と接続
4	P0_5			センサ部の J1.4 と接続
5	P0_4			センサ部の J1.5 と接続
6	P0_3			センサ部の J1.6 と接続
7	P0_2			センサ部の J1.7 と接続
8	P0_1			センサ部の J1.8 と接続
9	P0_0			センサ部の J1.9 と接続
10	GND			

1. ミニマイコンカーVer.2 の仕様

1.8.4 J6 の内容

ピン番号	詳細	"0"	"1"	備考
1	VCC(+5V)			
2	P3_7			
3	P3_6			
4	P3_5			
5	P3_4／ブザー(BZ1)			JP2 のランド間にある、半田面のパターンを切断することにより、BZ1 を切り離せます。その後、つなげたい場合は、JP2 のランド間をジャンパなどでショートさせます
6	P3_3			
7	P3_2／赤外線受光 IC(U1)			JP1 のランド間にある、半田面のパターンを切断することにより、U1 を切り離せます。その後、つなげたい場合は、JP1 のランド間をジャンパなどでショートさせます
8	P3_1			
9	P3_0			
10	GND			

1.8.5 J7 の内容

ピン番号	詳細	"0"	"1"	備考
1	VCC(+5V)			
2	P2_7			モータドライブ部の J9.2 と接続
3	P2_6			モータドライブ部の J9.3 と接続
4	P2_5			モータドライブ部の J9.4 と接続
5	P2_4			モータドライブ部の J9.5 と接続
6	P2_3			モータドライブ部の J9.6 と接続
7	P2_2			モータドライブ部の J9.7 と接続
8	P2_1			モータドライブ部の J9.8 と接続
9	P2_0／タクトスイッチ	押す	離す	モータドライブ部の J9.9 と接続
10	GND			

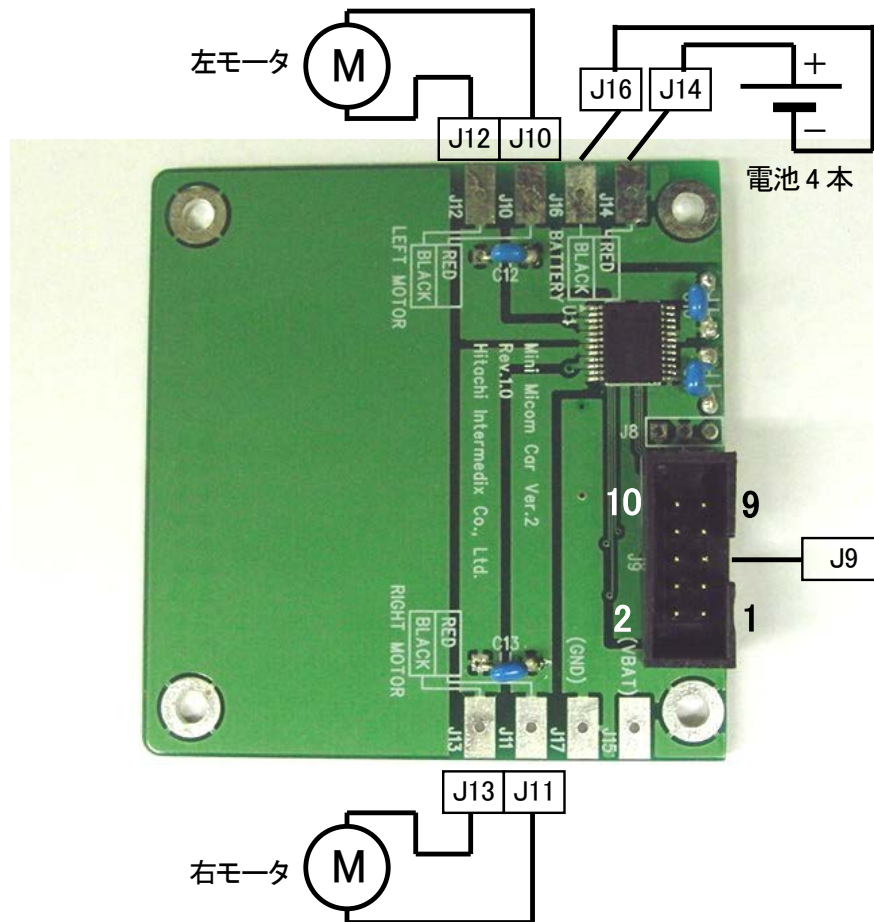
1.8.6 オプション部品

オプション部品の部品表を下記に示します。これらの部品は、マイコンカーラリー販売サイト (<http://www2.himdx.net/mcr/>)などで販売しています。

基板 番号	部品名	型式	写真	マイコンカーラリー 販売サイト 型式
J2,J3, J6,J7	10P オス コネクタ ストレート	ヒロセ電機(株) HIF3FC-10PA 2.54DSA		M-S42
JP1, JP2	2P 短絡 コネクタ	オムロン(株) XG8S-0231 ※JP1,JP2 を取り付けるとき は、JP1,JP2 の 1ピンと2ピ ンのパターン(半田面)をカ ットしてください。		
JP1, JP2 のソケ ット	短絡 ソケット	オムロン(株) XJ8A-0211		
D5, D6, D8, D9	LED	スタンレー電気(株) EBR3338S		M-S116
R19, R21, R22, R23	抵抗	コア(株) 1KΩ CFS1/4C		M-S91
U1	赤外線 受光 IC	シャープ(株) GP1UX511QS または、 (株)秋月電子通商 PL-IRM2121		
J5	DC ジャック	2.1mm 標準 DC ジャック (基板取付用) MJ-179P 内径 2.1mm 外径 5.5mm ※J5 を取り付けるときは、J5 の丸ランド間を繋いでいる パターン(半田面の銀色の 部分)をカットしてください。		M-S207

1.9 モータードライブ部のコネクタピン配置

1.9.1 外観



1.9.2 J9の詳細

ピン番号	詳細	"0"	"1"	マイコンの 接続先	備考
1	VCC(+5V)				
2	モータ右 2			P2_7	右モータの動作参照
3	モータ左 2			P2_6	左モータの動作参照
4	サーボ(オプション)			P2_5(PWM 出力)	
5	モータ右 PWM			P2_4(PWM 出力)	右モータの動作参照
6	モータ右 1			P2_3	右モータの動作参照
7	モータ左 PWM			P2_2(PWM 出力)	左モータの動作参照
8	モータ左 1			P2_1	左モータの動作参照
9	未接続				
10	GND				

※マイコンの接続先は、モータードライブ部とマイコン部を分離していない場合、パターンで接続されています。分離した場合は、J9とJ7(ポート2)をフラットケーブルなどで接続してください。

1. ミニマイコンカーVer.2 の仕様

(1) 左モータの動作

モータ左 1 P2_1	モータ左 2 P2_6	モータ左 PWM P2_2	モータ動作
1	1	x	ブレーキ
0	0	x	フリー
0	1	PWM	PWM="1"なら正転、"0"ならブレーキ
1	0	PWM	PWM="1"なら逆転、"0"ならブレーキ

P2_2 端子は、タイマ RD のリセット同期 PWM モードを使って PWM 波形を出力することができます。ミニマイコンカーVer.2 では、PWM 波形を使ってモータのスピード制御を行っています。詳しくはプロジェクト「motor」部分の説明を参照してください。

(2) 右モータの動作

モータ右 1 P2_3	モータ右 2 P2_7	モータ右 PWM P2_4	モータ動作
1	1	x	ブレーキ
0	0	x	フリー
0	1	PWM	PWM="1"なら正転、"0"ならブレーキ
1	0	PWM	PWM="1"なら逆転、"0"ならブレーキ

P2_4 端子は、タイマ RD のリセット同期 PWM モードを使って PWM 波形を出力することができます。ミニマイコンカーVer.2 では、PWM 波形を使ってモータのスピード制御を行っています。詳しくはプロジェクト「motor」部分の説明を参照してください。

1. ミニマイコンカーVer.2 の仕様

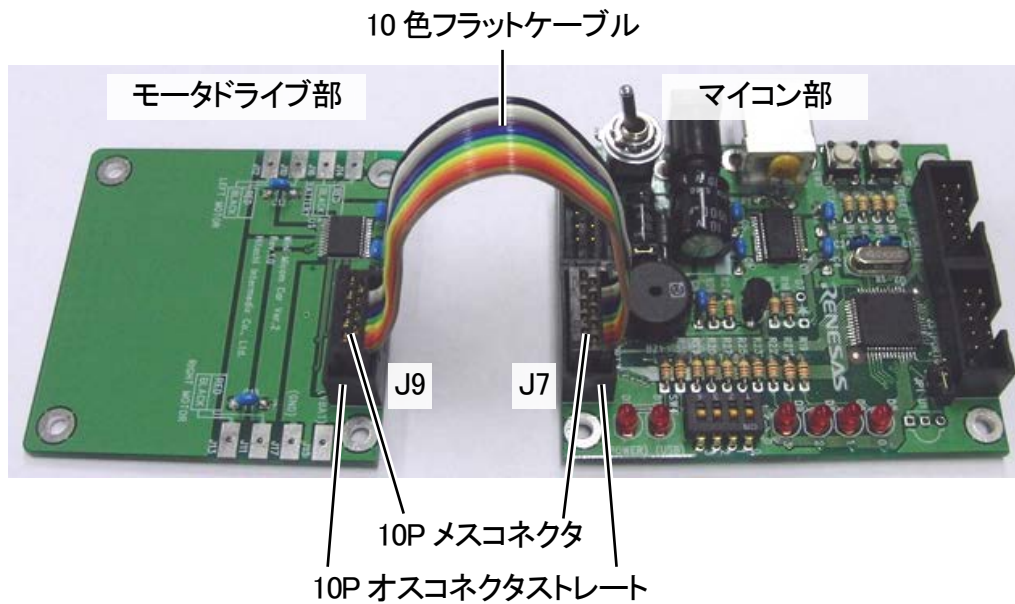
1.9.3 分離時の接続方法

マイコン部、モータドライブ部を分離したときに、これらの基板を接続する場合は、次の 2 系統の信号を接続する必要があります。

- ・マイコンの信号の接続
- ・電源系の接続

(1) マイコンの信号の接続

J7、J9 に 10 ピンストレートコネクタオスを実装し、フラットケーブルでこれらのコネクタ間を繋ぎます。



コネクタ、フラットケーブルは、ミニマイコンカーVer.2 には付属していません。これらの部品は、マイコンカーラリー販売サイト(<http://www2.himdx.net/mcr/>)などで販売しています。

部品名	型式	写真	マイコンカーラリー販売サイト 型式
10P オスコネクタ ストレート	ヒロセ電機(株) HIF3FC-10PA 2.54DSA		M-S42
10P メスコネクタ	オムロン(株) XG4M-1030		M-S44
10 色フラット ケーブル	1.27mm ピッチ		M-S45

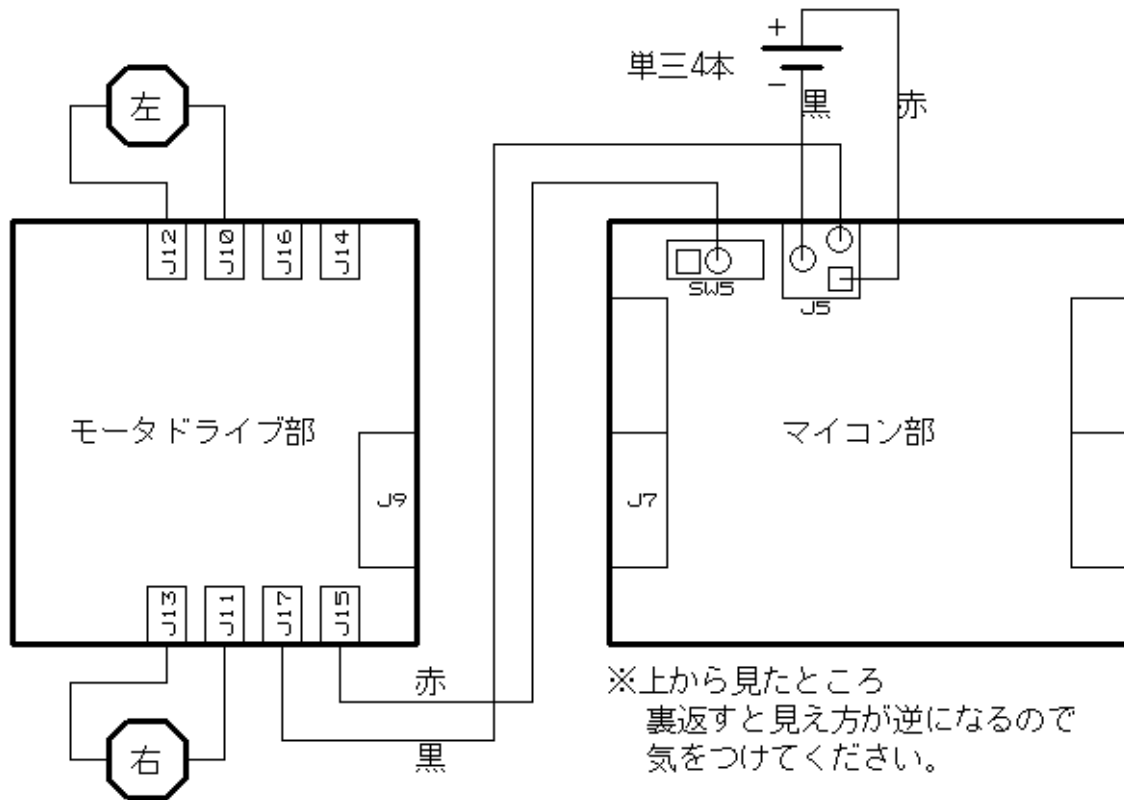
1. ミニマイコンカーVer.2 の仕様

(2) 電源系の接続



電源系は、電池 4 本 1 組で動作させる方法と、電池 4 本 2 組で動作させる方法があります。それぞれ線のつなぎ方が異なります。

①電池 4 本 1 組で動作させる接続方法

電池ボックスの線を半田面(裏面)から下図のように半田付けします。また同様にマイコン部とモータドライブ部を結ぶ線を半田面から半田付けします。



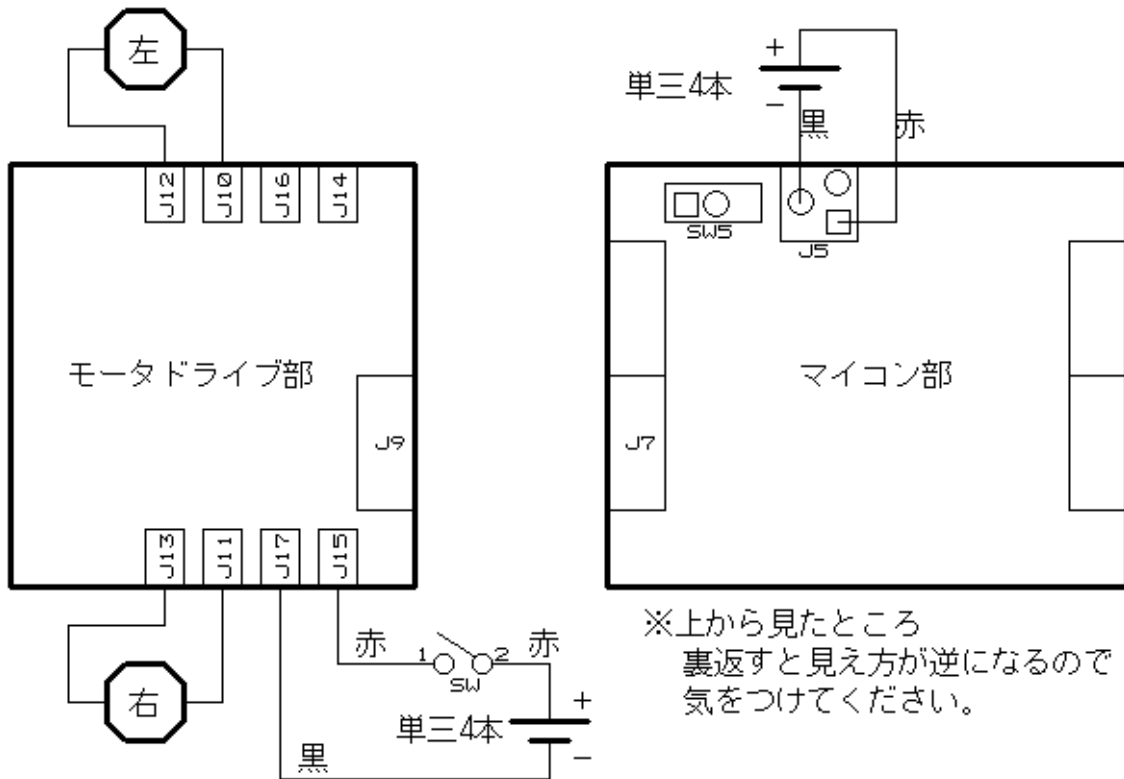
モータドライブ部とマイコン部を結線するケーブルは、ミニマイコンカーVer.2 には付属していません。これらの部品は、マイコンカーラー販売サイト(<http://www2.himdx.net/mcr/>)などで販売しています。

部品名	型式	写真	マイコンカーラー販売 サイト 型式
ビニール線(赤)	縊り線 赤 0.3KV 12 芯 15m		M-S119
ビニール線(黒)	縊り線 黒 0.3KV 12 芯 15m		M-S120





1. ミニマイコンカーVer.2 の仕様

②電池 4 本 2 組で動作させる接続方法

電池ボックスの線を半田面(裏面)から下図のように半田付けします。モータドライブ部の J15、J17 には別途用意したスイッチと電池ボックスを結線してください。



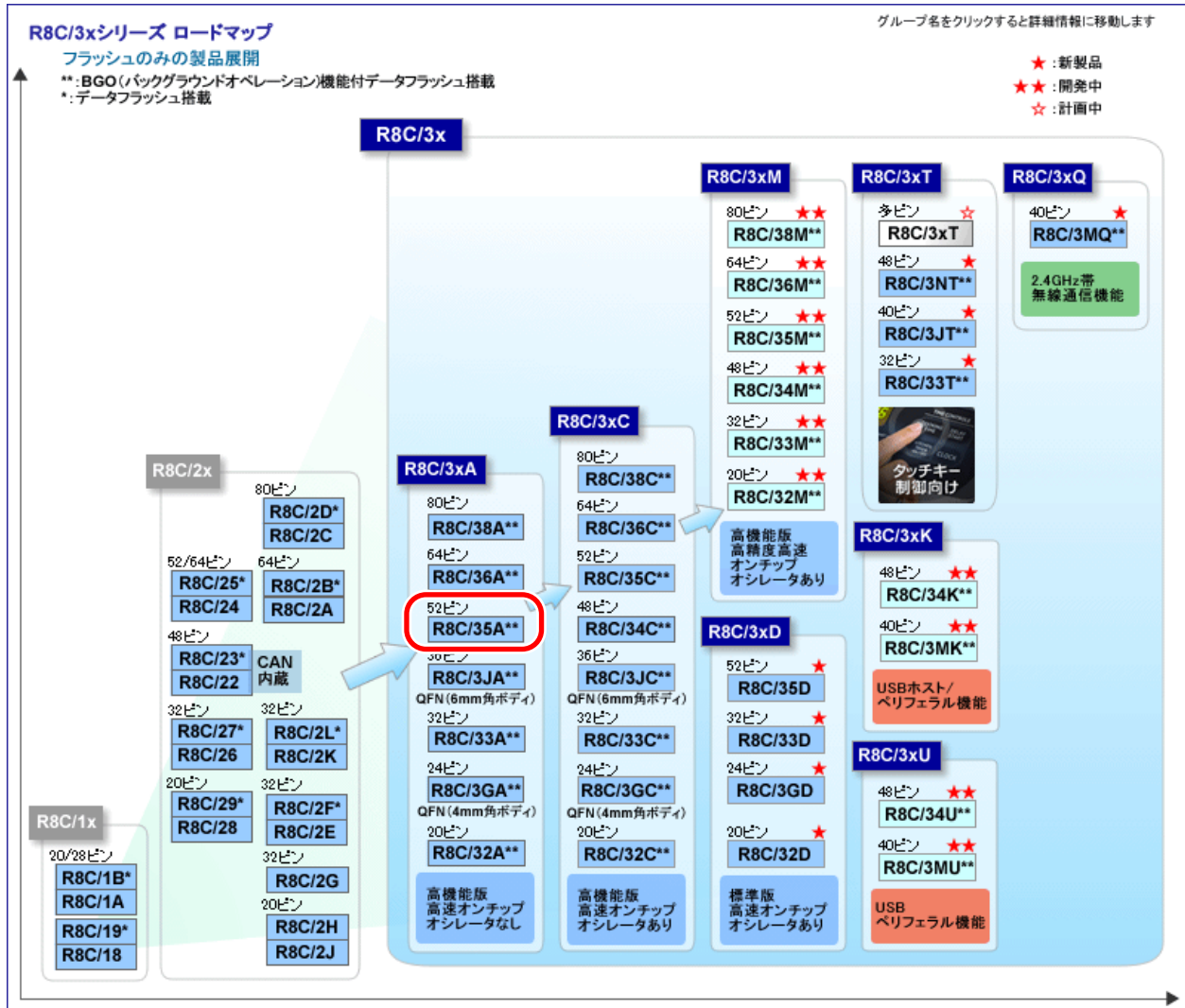
ケーブル、電池ボックス(追加分)、スイッチ(追加分)は、ミニマイコンカーVer.2 には付属していません。これらの部品は、マイコンカーラー販売サイト(<http://www2.himdx.net/mcr/>)などで販売しています。

部品名	型式	写真	マイコンカーラー販売サイト 型式
ビニール線(赤)	縊り線 赤 0.3KV 12 芯 15m		M-S119
ビニール線(黒)	縊り線 黒 0.3KV 12 芯 15m		M-S120
電池ボックス	(株)石川製作所 A-311 型 単 3×4 リード式		M-S104
トグルスイッチ	ミヤマ電器(株) MS243		M-S107

2. R8C/35A マイコンの仕様

2.1 シリーズ展開

ミニマイコンカーVer.2 で使用しているマイコンは、ルネサス エレクトロニクス製の「R8C/35A」です。このマイコンは、R8C ファミリーに属しています。R8C ファミリーのシリーズ展開を下記に示します。



▲ルネサス エレクトロニクスのホームページより

※2011年4月現在

2.2 R8C/35A の特徴

R8C/35A は R8C CPU コアを搭載しています。最大動作周波数は 20MHz です。フラッシュメモリ版を用意しており、内蔵フラッシュメモリは単一電源で書き換え可能です。

特徴を下記に示します。

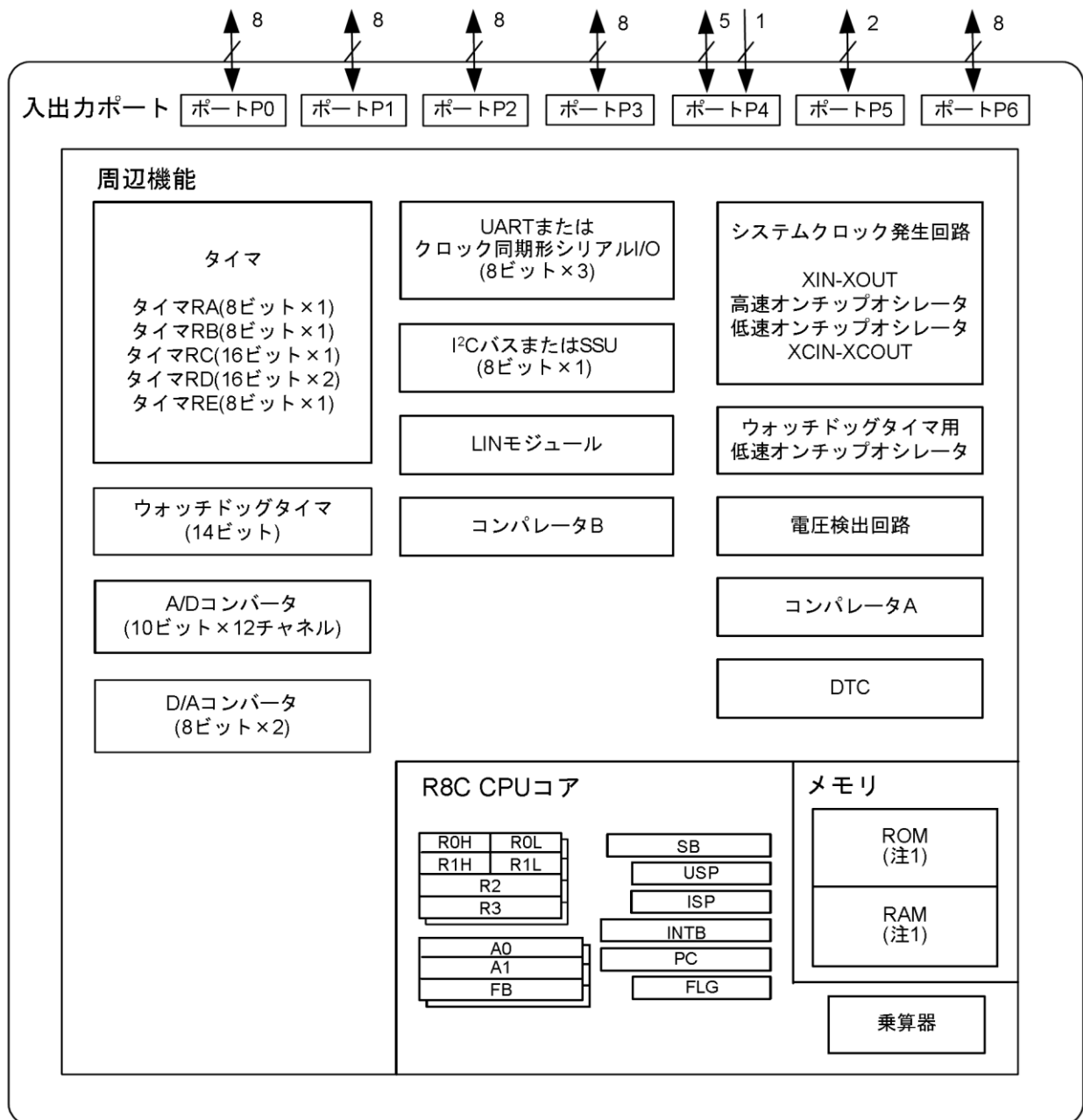
- 8ビットプリスケラ付 8ビット多機能タイマ(タイマ RA、RB) :2 チャンネル
- 16ビットインプットキャプチャ/アウトプットコンペアタイマ(タイマ RC、RD) :3 チャンネル
- 8ビットコンペアマッチ機能付リアルタイムクロックタイマ(タイマ RE) :1 チャンネル
- UART/クロック同期形シリアルインタフェース:3 チャンネル
- I²C-bus インタフェース(IIC)/シンクロナスシリアルコミュニケーションユニット:1 チャンネル
- LIN モジュール:1 チャンネル(タイマ RA、UART0 を使用)
- 10ビット A/D コンバータ:12 チャンネル
- 8ビット D/A コンバータ:2 回路
- コンパレータ:2 回路(電圧監視 1、電圧監視 2 と兼用)+2 回路
- DTC(データトランスファコントローラ) :1 チャンネル
- ウォッチドッグタイマ
- クロック発生回路:低速オンチップオシレータ、XIN クロック発振回路、XCIN クロック発振回路
- 発振停止検出機能
- 電圧検出回路
- パワーオンリセット回路
- 入出力ポート:47 本 (LED 駆動用ポート含む)
- 外部割り込み入力:9 本
- BGO(バックグラウンドオペレーション)機能付データフラッシュ:4KB

2.3 機能

R8C/35AとH8/3048F-ONEの機能を下表に示します。今までRY3048Foneボード(H8/3048F-ONE)を使用していた場合、どう違うか比べてみてください。

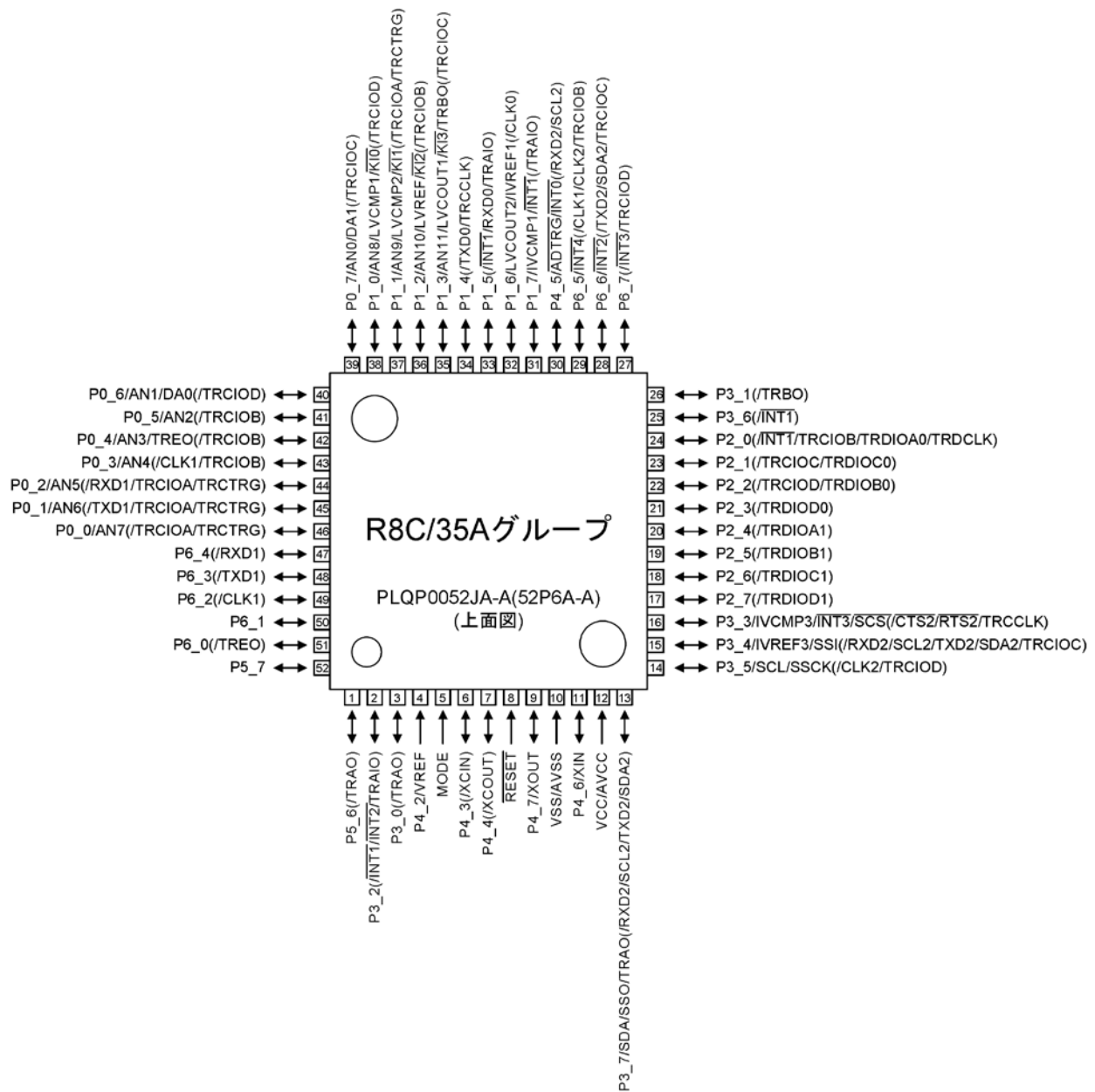
		本ボード	RY3048Foneボード																										
マイコン		ルネサス エレクトロニクス製 R8C/35A (R5F21356ANFP)	ルネサス エレクトロニクス製 H8/3048F-ONE																										
パッケージ		52ピンLQFP	100ピンQFP																										
ボードのクリスタル値		20.0MHz	24.576MHz																										
動作電圧		動作周波数が5.0～20.0MHzの場合:2.7～5.5V 動作周波数が2.0～5.0MHzの場合:1.8～5.5V	4.5～5.5V																										
基板外形		マイコン部のみでW72×D60×H12mm(実寸) ※トグルスイッチ含むと高さは26mm	W70×D59×H13mm(実寸)																										
内蔵メモリ	ROM	32KB: 0x8000～0xffff番地 プログラム、イレーズ(消去)回数:保証回数1000回	128KB: 0x00000～0x1ffff番地 プログラム、イレーズ(消去)回数:保証回数100回																										
	RAM	2.5KB: 0x0400～0x0dff番地	4KB: 0xfef10～0xffff0f番地																										
	データフラッシュ	4KB: 0x3000～0x3fff番地 プログラム、イレーズ(消去)回数:保証回数10000回	なし																										
アドレス空間		外部メモリの接続は不可 ※アドレスバスと、データバスのあるメモリです。	外部メモリの接続は可能																										
マイコンの動作周波数		次の3つから選べます。 ●低速オンチップオシレータ(内蔵クリスタル) 約125kHz ●XINクロック入力(外付けクリスタル) 2.0～20.0MHz ●XCINクロック入力(外付けクリスタル) 32.768kHz(時計用) ※マイコン起動時は、低速オンチップオシレータで動作します。動作後、プログラムでどのクロックを使用するか選択します。	外付けクリスタルにて 2.0～25.0MHz																										
コネクタ		<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>コネクタ(ポート)</th> <th>I/O数</th> </tr> </thead> <tbody> <tr> <td>J2(ポート6)</td> <td>8</td> </tr> <tr> <td>J3(ポート0)※</td> <td>8</td> </tr> <tr> <td>J6(ポート3)</td> <td>8</td> </tr> <tr> <td>J7(ポート2)※</td> <td>8</td> </tr> <tr> <td>合計</td> <td>32</td> </tr> </tbody> </table> <p>※J3はセンサ部分、J7はモータドライブ部分に接続されており、使用できません。それぞれを分割すれば、通常のI/Oポートとして使用できます。</p>	コネクタ(ポート)	I/O数	J2(ポート6)	8	J3(ポート0)※	8	J6(ポート3)	8	J7(ポート2)※	8	合計	32	<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>コネクタ(ポート)</th> <th>I/O数</th> </tr> </thead> <tbody> <tr> <td>J1(ポート7)※</td> <td>8</td> </tr> <tr> <td>J2(ポートA)</td> <td>8</td> </tr> <tr> <td>J3(ポートB)</td> <td>8</td> </tr> <tr> <td>J6(ポート3, 4など)</td> <td>18</td> </tr> <tr> <td>J8(ポート1, 2, 5など)</td> <td>29</td> </tr> <tr> <td>合計</td> <td>71</td> </tr> </tbody> </table> <p>※J1は入力専用です。</p>	コネクタ(ポート)	I/O数	J1(ポート7)※	8	J2(ポートA)	8	J3(ポートB)	8	J6(ポート3, 4など)	18	J8(ポート1, 2, 5など)	29	合計	71
コネクタ(ポート)	I/O数																												
J2(ポート6)	8																												
J3(ポート0)※	8																												
J6(ポート3)	8																												
J7(ポート2)※	8																												
合計	32																												
コネクタ(ポート)	I/O数																												
J1(ポート7)※	8																												
J2(ポートA)	8																												
J3(ポートB)	8																												
J6(ポート3, 4など)	18																												
J8(ポート1, 2, 5など)	29																												
合計	71																												
ディップスイッチ		P5.7, P4.5～P4.3に接続(4ビット分) ※P5.7…ポート5のbit7という意味です。	P63～P60に接続(4ビット分) ※P63…ポート6のbit3 という意味です。																										

2.4 ブロック図



2. R8C/35A マイコンの仕様

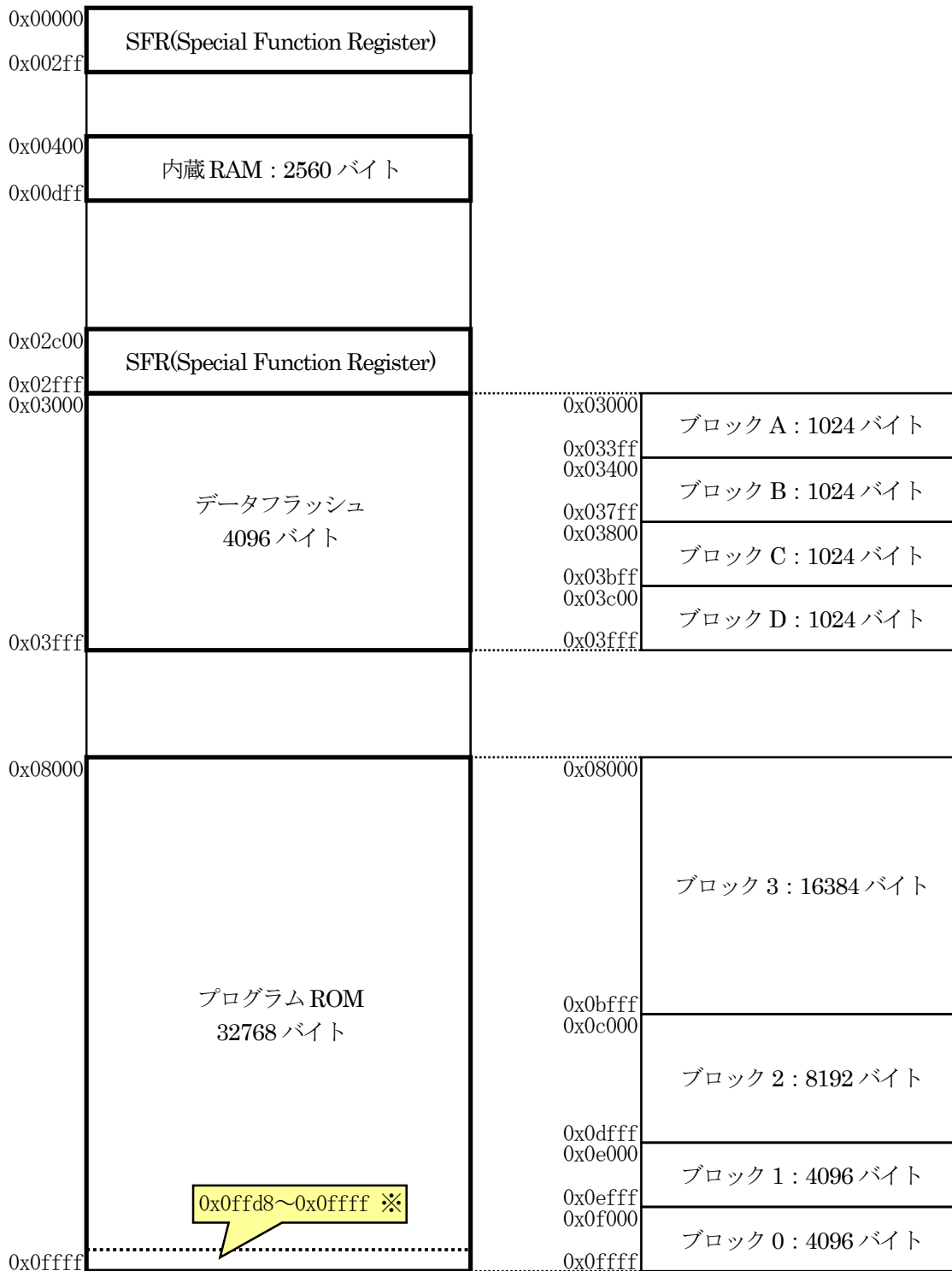
2.5 ピン配置



- 注1. プログラムで()の端子に配置できます。
注2. パッケージの1ピンの位置は「外形寸法図」で確認してください。

2.6 メモリマップ

ミニマイコンカーVer.2 で使用している R8C/35A(R5F21356ANFP)には、プログラム ROM:32KB、データフラッシュ:4KB、内蔵 RAM:2.5KB が内蔵されています。



※0x0fffd8~0x0ffdb 番地は予約領域、0x0ffdc~0x0ffff 番地は固定割り込みベクタテーブルの領域のため、プログラムは置けません。

2.7 固定割り込みベクタテーブル領域

0x0ffd8～0x0ffdb 番地は予約領域、0x0ffdc～0x0ffff 番地は固定割り込みベクタテーブルの領域のため、プログラムは置けません。固定割り込みベクタテーブル領域の詳細を下表に示します。

アドレス	内容
0x0ffd8	予約領域
0x0ffd9	予約領域
0x0ffda	予約領域
0x0ffdb	オプション機能選択レジスタ 2 (OFS2)
0x0ffdc	未定義命令割り込み発生時のジャンプ先アドレス 下位
0x0ffdd	未定義命令割り込み発生時のジャンプ先アドレス 中位
0x0ffde	未定義命令割り込み発生時のジャンプ先アドレス 上位
0x0ffdf	ID1
0x0ffe0	オーバフロー割り込み発生時のジャンプ先アドレス 下位
0x0ffe1	オーバフロー割り込み発生時のジャンプ先アドレス 中位
0x0ffe2	オーバフロー割り込み発生時のジャンプ先アドレス 上位
0x0ffe3	ID2
0x0ffe4	BRK 命令割り込み発生時のジャンプ先アドレス 下位
0x0ffe5	BRK 命令割り込み発生時のジャンプ先アドレス 中位
0x0ffe6	BRK 命令割り込み発生時のジャンプ先アドレス 上位
0x0ffe7	
0x0ffe8	アドレス一致割り込み発生時のジャンプ先アドレス 下位
0x0ffe9	アドレス一致割り込み発生時のジャンプ先アドレス 中位
0x0ffea	アドレス一致割り込み発生時のジャンプ先アドレス 上位
0x0ffeb	ID3
0x0ffec	シングルステップ割り込み発生時のジャンプ先アドレス 下位
0x0ffed	シングルステップ割り込み発生時のジャンプ先アドレス 中位
0x0ffee	シングルステップ割り込み発生時のジャンプ先アドレス 上位
0x0ffef	ID4
0x0fff0	ウォッチドッグタイマ、発振停止検出、電圧検出割り込み発生時のジャンプ先アドレス 下位
0x0fff1	ウォッチドッグタイマ、発振停止検出、電圧検出割り込み発生時のジャンプ先アドレス 中位
0x0fff2	ウォッチドッグタイマ、発振停止検出、電圧検出割り込み発生時のジャンプ先アドレス 上位
0x0fff3	ID5
0x0fff4	アドレスブレイク割り込み発生時のジャンプ先アドレス 下位
0x0fff5	アドレスブレイク割り込み発生時のジャンプ先アドレス 中位
0x0fff6	アドレスブレイク割り込み発生時のジャンプ先アドレス 上位
0x0fff7	ID6
0x0fff8	(予約)
0x0fff9	(予約)
0x0fffa	(予約)
0x0fffb	ID7
0x0fffc	リセット時のジャンプ先アドレス 下位
0x0fffd	リセット時のジャンプ先アドレス 中位
0x0fffe	リセット時のジャンプ先アドレス 上位
0x0ffff	オプション機能選択レジスタ (OFS)

※固定割り込みベクタテーブルは、ROM 領域なので、プログラム書き込み時に設定します。プログラム実行時に書き換えはできません。

※ID1～ID7:ID コード領域です。

2.8 CPU レジスタ

R8C/35A の CPU レジスタを下表に示します。

レジスタ名	名称	ビット幅	リセット後の初期値	説明
R0	データレジスタ R0	16bit	0x0000	主に転送や算術、論理演算に使用します。 R0 はさらに上位と下位に分かれており、上位は R0H、下位は R0L です。R0=R0H+R0L の関係です。
R1	データレジスタ R1	16bit	0x0000	主に転送や算術、論理演算に使用します。 R1 はさらに上位と下位に分かれており、上位は R1H、下位は R1L です。R1=R1H+R1L の関係です
R2	データレジスタ R2	16bit	0x0000	主に転送や算術、論理演算に使用します。
R3	データレジスタ R3	16bit	0x0000	主に転送や算術、論理演算に使用します。
A0	アドレスレジスタ A0	16bit	0x0000	間接/相対アドレッシング時のベースアドレス格納用レジスタとして使用します。C 言語では、ポインタや配列を使うときに使用します。
A1	アドレスレジスタ A1	16bit	0x0000	間接/相対アドレッシング時のベースアドレス格納用レジスタとして使用します。C 言語では、ポインタや配列を使うときに使用します。
FB	フレームベースレジスタ	16bit	0x0000	auto 変数と引数を相対アドレッシングでアクセスする際のベースアドレス格納用として使用します。
PC	プログラムカウンタ	20bit	0x0fff _c ~0x0fff _e 番地のアドレスにある値	次に実行する命令のアドレスを示すレジスタです。リセット後、0x0fff _c ~0x0fff _e 番地のデータを PC に転送して、そのアドレスに書かれているプログラムから実行します。
INTB	割り込みテーブルレジスタ	20bit	0x00000	可変割り込みベクタの先頭アドレスを格納するレジスタです。startup.c の start 関数内で、セクション「vector」の先頭アドレス 0x0fed8 番地を設定します。INTB は、上位と下位に分かれており、上位(bit19~16)は INTBH、下位(bit15~bit0)は INTBL です。INTB=INTBH+INTBL の関係です。
USP	ユーザスタックポインタ	16bit	0x0000	関数を呼び出すとき、元に戻る番地や引数を保存する領域(スタック領域)を示すレジスタです。要は、このレジスタの値が示すアドレスを作業用領域として使います。
ISP	割り込みスタックポインタ	16bit	0x0000	USP と同じです。 通常プログラムでは U="1" で USP を使用、割り込み発生時には U="0" になり ISP を使用します。 USP と ISP は同時に使用することができません。プログラムで SP(スタックポインタ)と指定すると、USP か ISP になります。選択は、FLG レジスタの U ビットで行います。 FLG の U ビット="0" のとき、SP=ISP になる FLG の U ビット="1" のとき、SP=USP になる
SB	スタティックベースレジスタ	16bit	0x0000	変数のアクセス効率を向上させるために用意されているレジスタです。変数の相対アドレッシング時にベースアドレスを格納します。
FLG	フラグレジスタ	16bit	0x0000	1 ビット単位のフラグと 3 ビットのプロセッサ割り込み優先レベル(IPL)で構成されたレジスタです。

※フラグレジスタの詳細

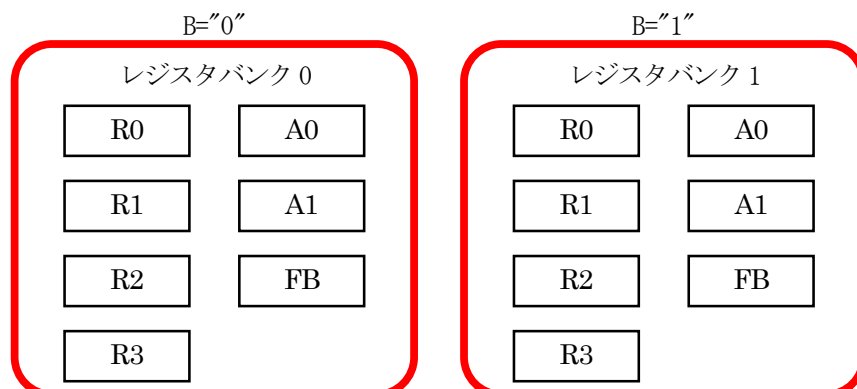
フラグレジスタのビット構成を下記に示します。

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
内容		IPL							U	I	O	B	S	Z	D	C

bit	名称	詳細
14~12	IPL	プロセッサ割り込み優先レベル(Interrupt Priority Level)です。レベル0~レベル7を指定可能です。どの優先レベルの割り込みまで受け付けるかを指定します。 IPL<発生した割り込みの割り込み優先レベル の条件で割り込みが発生します。 IPL はリセット後の初期値は 0 なので、リセット後は割り込み優先レベル 1 以上の割り込みを受け付けます。
7	U	スタックポインタ指定フラグです。プログラムで「SP」を使うとき、U="0"なら ISP、U="1"なら USP を使用します。
6	I	割り込み許可フラグです。I="0"で割り込み禁止、I="1"で割り込み許可です。 割り込みを使用するときは I="1"にします。
5	O	オーバフローフラグです。演算結果がオーバフローしたとき(8ビット時:-128~+127、16ビット時:-32768~+32767)に O="1"になります。
4	B	レジスタバンク指定フラグです。B="0"のときレジスタバンク 0 を使用、B="1"のときレジスタバンク 1 を使用します。
3	S	サインフラグです。演算結果が負のとき S="1"になります。
2	Z	ゼロフラグです。演算結果が 0 のとき Z="1"になります。
1	D	デバッグフラグです。シングルステップ割り込みを許可するフラグです。D="1"で命令実行後にシングルステップ割り込みが発生します。割り込み要求受け付け後、D="0"になります。
0	C	キャリーフラグです。算術演算ユニットで発生したキャリー(桁上がり時 C="1")、ボロー(桁借り時 C="0")、シフトアウトしたビットを保持します。
その他		予約ビットです。値を書く場合は、"0"を書いてください。読み込んだ場合の値は不定です。

※レジスタバンクについて

R0~R3、A0~A1、FB は、レジスタバンク 0 とレジスタバンク 1 の 2 組あります。FLG のレジスタバンク指定フラグ(B)で 0 と 1 のどちらを使うか設定します。例えば、レジスタバンク 0 の R0 とレジスタバンク 1 の R0 は、どちらもプログラムからは R0 としか指定できませんが、レジスタバンク指定フラグ(B)により参照するレジスタが変わりますので、プログラムでは間違えないように気をつけてください。C 言語でプログラムするときは、C コンパイラが指定するので、意識する必要はありません。



2.9 タイマの概要

R8C/35Aには、タイマ RA～タイマ RE という名称の 5 つのタイマを内蔵しています。それぞれのタイマの機能を下記に示します。

タイマ名	機能
タイマ RA	8ビット(8ビットプリスケアラ付)×1 タイマモード(周期タイマ)、パルス出力モード(周期ごとのレベル反転出力)、イベントカウンタモード、パルス幅測定モード、パルス周期測定モード
タイマ RB	8ビット(8ビットプリスケアラ付)×1 タイマモード(周期タイマ)、プログラマブル波形発生モード(PWM 出力)、プログラマブルワンショット発生モード、プログラマブルウェイトワンショット発生モード
タイマ RC	16ビット(キャプチャ/コンペアレジスタ 4 本付)×1 タイマモード(インプットキャプチャ機能、アウトプットコンペア機能)、PWM モード(出力 3 本)、PWM2 モード(PWM 出力 1 本)
タイマ RD	16ビット(キャプチャ/コンペアレジスタ 4 本付)×2 タイマモード(インプットキャプチャ機能、アウトプットコンペア機能)、PWM モード(出力 6 本)、リセット同期 PWM モード(三相波形出力(6 本)鋸波変調)、相補 PWM モード(三相波形出力(6 本)三角波変調)、PWM3 モード(同一周期の PWM 出力 2 本)
タイマ RE	8ビット×1 リアルタイムクロックモード(秒、分、時、曜日カウント)、アウトプットコンペアモード

2.10 ミニマイコンカーVer.2 で使用する内蔵周辺機能

R8C/35A でミニマイコンカーVer.2を制御するときの内蔵周辺モジュールの一覧を下記に示します。参考として、H8/3048F-ONE でマイコンカーを制御するときの内蔵周辺モジュールと比較します。

項目	R8C/35A でミニマイコンカーVer.2を制御するときの内蔵周辺モジュール	H8/3048F-ONE でマイコンカーを制御するときの内蔵周辺モジュール(参考)
1ms ごとの割り込み	タイマ RB	ITU0
左モータ、右モータ、サーボの制御	タイマ RD によるリセット同期 PWM モード ※サーボ、サーボ用 3P コネクタはオプションです	ITU3,4 によるリセット同期 PWM モード
ブザー	タイマ RC	SCI1(通信機能)をブザー制御に使用
赤外線を受光制御	タイマ RA	
ロータリエンコーダ (パルスカウント)	タイマ RA ※ミニマイコンカーVer.2 ではロータリエンコーダは使用しませんが、使用したい場合は、タイマ RA を使用することによりパルスカウントができます。	ITU2

3. 開発環境のダウンロード、インストール

3.1 MY Renesas に登録

R8C/35A でプログラム開発、書き込みするためのソフトウェアは、ルネサス エレクトロニクスのホームページから無償評価版をダウンロードすることができます。ダウンロードするには、「MY Renesas」という Web ユーザ登録(無料)をする必要があります。すでに「MY Renesas」に登録している場合は、ここで登録する必要はありません。

1		<p>ルネサス エレクトロニクスのサイト http://japan.renesas.com/ にアクセスします。</p> <p>○部の 「MY Rnesas」をクリックします。</p>
---	---	---

2		<p>新規登録をクリックして、登録手続きをしてください。メールアドレスとパスワードは、ファイルをダウンロードするときに必要ですので控えておいてください。</p>
---	--	---

3. 開発環境のダウンロード、インストール

3.2 ルネサス統合開発環境のダウンロード

1		<p>ルネサス エレクトロニクスのホームページ (http://japan.renesas.com/) を開き、「開発環境→統合開発環境(IDE)」をクリックします。</p>
---	--	---

2		<p>右側にある「評価版ソフトウェアツール」をクリックします。</p>
---	--	-------------------------------------

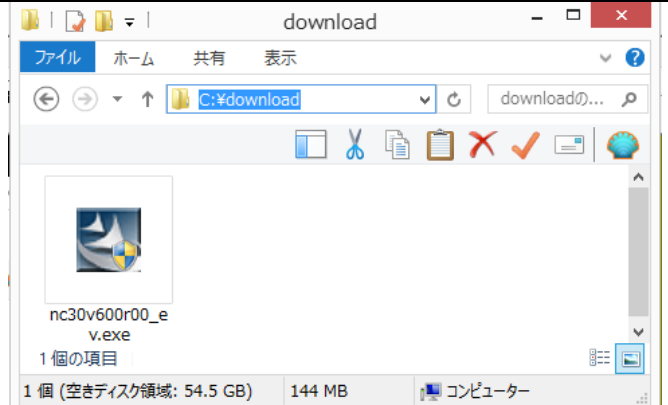
3	<table border="1" style="width: 100%;"> <tr> <td style="width: 30%;"> <p>M16Cシリーズ、R8Cファミリ用 C/C++コンパイラパッケージ (M3T-NC30WA)</p> <p>評価版ダウンロード</p> <p>製品版をインストールの際は、R8C、M16Cファミリ用C/C++コンパイラパッケージをご注文ください。詳細は製品ページを参照ください。</p> </td> <td style="width: 40%;"> <p>V.6.00 Release 00</p> <ul style="list-style-type: none"> ■ 試用期限内は製品版と同じ。 ■ 試用期限を過ぎるとリンクサイズが64KB以内制限されます。 ■ High-Performance Embedded Workshopおよびシミュレータデバッグを同様。 </td> <td style="width: 30%;"> <p>60日</p> <p>初めて評価版ソフトウェアツールをインストールした後、最初ビルドを行った日から60日。</p> <p>*一度でもビルドを行った場合は評価版ソフトウェアツールをインストールしなおしても試用期限の延長はできません。</p> </td> </tr> </table> <p style="text-align: center; margin-top: 10px;">評価版ダウンロード</p>	<p>M16Cシリーズ、R8Cファミリ用 C/C++コンパイラパッケージ (M3T-NC30WA)</p> <p>評価版ダウンロード</p> <p>製品版をインストールの際は、R8C、M16Cファミリ用C/C++コンパイラパッケージをご注文ください。詳細は製品ページを参照ください。</p>	<p>V.6.00 Release 00</p> <ul style="list-style-type: none"> ■ 試用期限内は製品版と同じ。 ■ 試用期限を過ぎるとリンクサイズが64KB以内制限されます。 ■ High-Performance Embedded Workshopおよびシミュレータデバッグを同様。 	<p>60日</p> <p>初めて評価版ソフトウェアツールをインストールした後、最初ビルドを行った日から60日。</p> <p>*一度でもビルドを行った場合は評価版ソフトウェアツールをインストールしなおしても試用期限の延長はできません。</p>	<p>M16C シリーズ、R8C ファミリ用 C コンパイラパッケージ (M3T-NC30WA) の欄の中にある、「評価版ダウンロード」をクリックします。「V.6.00 Release 00」部分はバージョンにより異なります。</p>
<p>M16Cシリーズ、R8Cファミリ用 C/C++コンパイラパッケージ (M3T-NC30WA)</p> <p>評価版ダウンロード</p> <p>製品版をインストールの際は、R8C、M16Cファミリ用C/C++コンパイラパッケージをご注文ください。詳細は製品ページを参照ください。</p>	<p>V.6.00 Release 00</p> <ul style="list-style-type: none"> ■ 試用期限内は製品版と同じ。 ■ 試用期限を過ぎるとリンクサイズが64KB以内制限されます。 ■ High-Performance Embedded Workshopおよびシミュレータデバッグを同様。 	<p>60日</p> <p>初めて評価版ソフトウェアツールをインストールした後、最初ビルドを行った日から60日。</p> <p>*一度でもビルドを行った場合は評価版ソフトウェアツールをインストールしなおしても試用期限の延長はできません。</p>			

4	<table border="1" style="width: 100%;"> <tr> <td style="width: 20%;">M3T-NC30WA</td> <td style="width: 30%; text-align: center;"> <p>【無償評価版】M16Cシリーズ、R8Cファミリ用 C/C++コンパイラパッケージ M3T-NC30WA V.6.00 Release 00</p> </td> <td style="width: 10%; text-align: center;">Apr.05.11</td> <td style="width: 40%;"> <p>無償評価版です。Windows® 7、Windows Vista®、Windows® XPにのみインストールできます。High-Performance Embedded Workshopおよびシミュレータデバッグを同様。</p> </td> </tr> </table>	M3T-NC30WA	<p>【無償評価版】M16Cシリーズ、R8Cファミリ用 C/C++コンパイラパッケージ M3T-NC30WA V.6.00 Release 00</p>	Apr.05.11	<p>無償評価版です。Windows® 7、Windows Vista®、Windows® XPにのみインストールできます。High-Performance Embedded Workshopおよびシミュレータデバッグを同様。</p>	<p>「【無償評価版】M16Cシリーズ、R8Cファミリ用 C/C++コンパイラパッケージ M3T-NC30WA V.6.00 Release 00」をクリックします。「V.6.00 Release 00」部分はバージョンにより異なります。※無い場合は、HPの2ページ以降を参照してください。</p>
M3T-NC30WA	<p>【無償評価版】M16Cシリーズ、R8Cファミリ用 C/C++コンパイラパッケージ M3T-NC30WA V.6.00 Release 00</p>	Apr.05.11	<p>無償評価版です。Windows® 7、Windows Vista®、Windows® XPにのみインストールできます。High-Performance Embedded Workshopおよびシミュレータデバッグを同様。</p>			

3. 開発環境のダウンロード、インストール

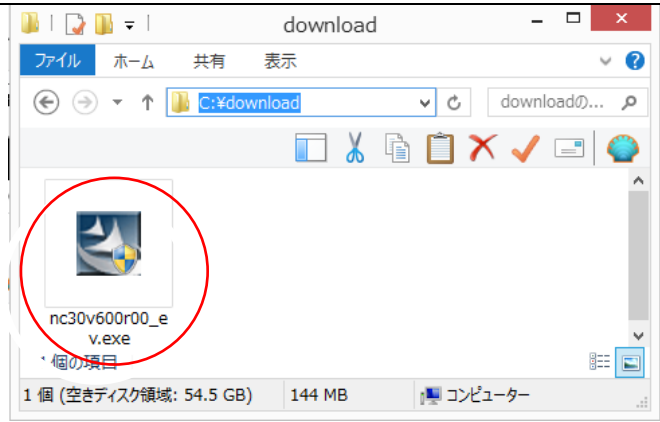
5	<p>本書による事前の承諾なしに転載または複製することを固くお断り致します。</p> <p>お問い合わせ、その他お気付きの点等ございましたら弊社営業窓口までご連絡ください。</p> <p>上記事項に</p> <p style="text-align: center;"> <input type="button" value="同意する"/> <input type="button" value="同意しない"/> </p>	<p>注意事項が表示されます。同意する場合は、同意するをクリックします。</p>
---	---	---

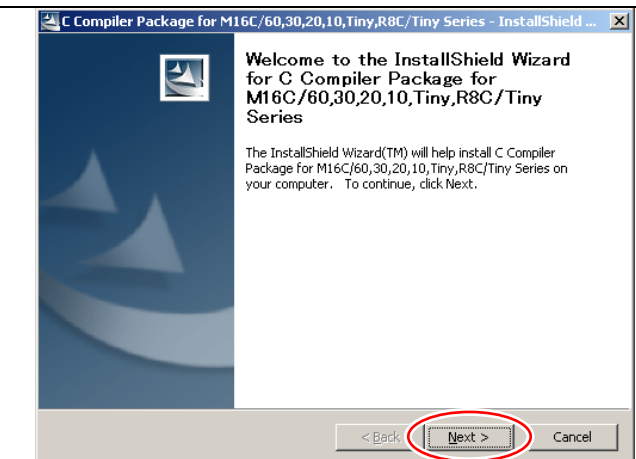
6	<p>注意！！</p> <p>ここからダウンロードできるソフトウェアは評価版です。評価版コンパイラに対する技術サポートは行っておりません。</p> <p>ダウンロード</p> <table border="1"> <thead> <tr> <th>ダウンロード製品名</th> <th>ファイル名</th> <th>ファイルサイズ</th> <th>リンク</th> </tr> </thead> <tbody> <tr> <td>【無償評価版】M16Cシリーズ, R8Cファミリ用C/C++コンパイラパッケージ M3T-NC30WA V.6.00 Release 00</td> <td>nc30v600r00_ev.exe</td> <td>151,029,104 bytes (144.0 Mbytes)</td> <td style="text-align: center;"> <input type="button" value="ダウンロード"/> </td> </tr> </tbody> </table>	ダウンロード製品名	ファイル名	ファイルサイズ	リンク	【無償評価版】M16Cシリーズ, R8Cファミリ用C/C++コンパイラパッケージ M3T-NC30WA V.6.00 Release 00	nc30v600r00_ev.exe	151,029,104 bytes (144.0 Mbytes)	<input type="button" value="ダウンロード"/>	<p>ダウンロードをクリックし、ファイルをダウンロードします。</p>
ダウンロード製品名	ファイル名	ファイルサイズ	リンク							
【無償評価版】M16Cシリーズ, R8Cファミリ用C/C++コンパイラパッケージ M3T-NC30WA V.6.00 Release 00	nc30v600r00_ev.exe	151,029,104 bytes (144.0 Mbytes)	<input type="button" value="ダウンロード"/>							

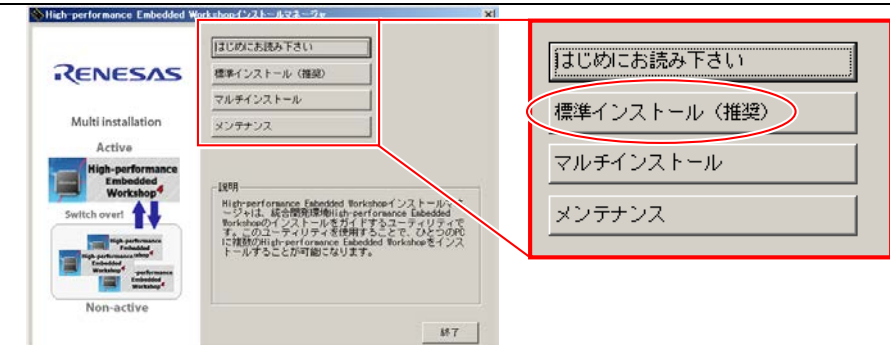
7		<p>ダウンロードできました。</p>
---	--	---------------------

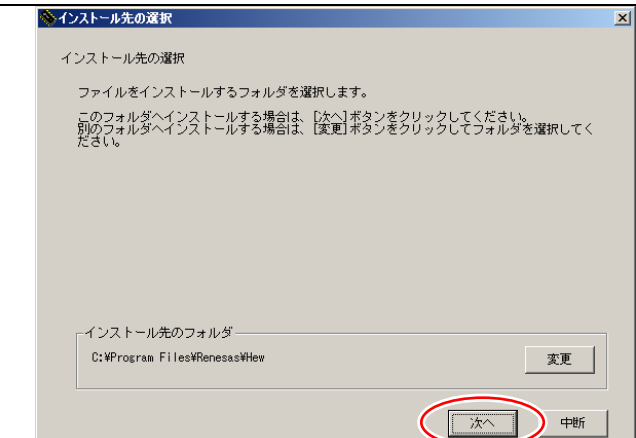
3. 開発環境のダウンロード、インストール

3.3 ルネサス統合開発環境のインストール

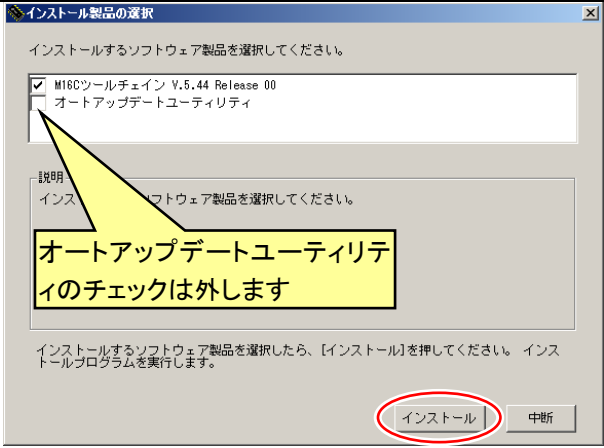
1		<p>ダウンロードした「nc30v600r00_ev.exe」を実行します。 「v600r00」部分はバージョンです。異なることがあります。</p> <p>※既にルネサス統合開発環境が入っていてもアンインストールせず、そのまま今回のバージョンのルネサス統合開発環境をインストールしてください。</p>
---	---	--

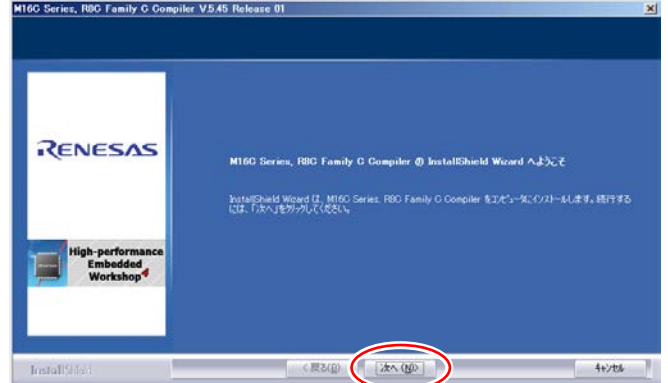
2		<p>Next > をクリックします。</p>
---	--	----------------------------

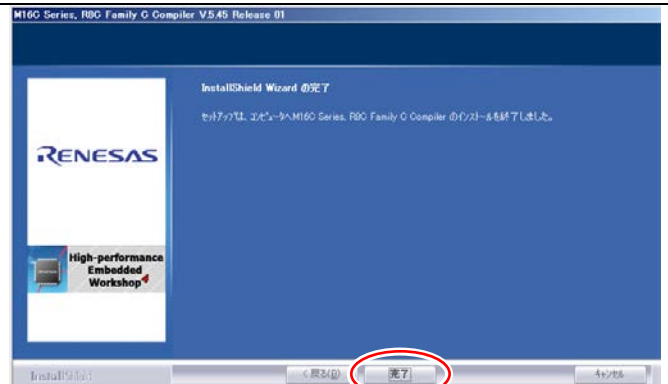
3		<p>標準インストール (推奨) をクリックします。</p>
---	--	--------------------------------

4		<p>次へ をクリックします。</p>
---	---	---------------------

3. 開発環境のダウンロード、インストール

5		<p>インストールをクリックします。</p>
---	---	------------------------

6		<p>次へをクリックします。</p>
---	--	--------------------

7		<p>完了をクリックして、インストール完了です。</p>
---	---	------------------------------

※ルネサス統合開発環境の無償評価版の制限について

ルネサス統合開発環境の無償評価版は製品版と比べ、次の制限があります。

- ・インストール後 60 日以上たつとリンクサイズが 64KB 以内に制限されます。
- ・無償評価版のサポートは一切行いませんので、ご了解の上ご使用ください。
- ・お問い合わせ窓口へのご質問につきましても、サポート対象外となります。

3.4 ショートカットの作成

ルネサス統合開発環境(High-performance Embedded Workshop)がすぐに実行できるように、デスクトップにショートカットを作成します。

<p>1</p>		<p>「スタート」 ↓ 「すべてのプログラム」 ↓ 「Renesas」 ↓ 「High-performance Embedded Workshop」 ↓ までたどっていき、「High-performance Embedded Workshop」で、右クリックして、「コピー」をクリックします。</p>
<p>2</p>		<p>デスクトップで右クリックして、「貼り付け」をクリックします。デスクトップにショートカットが作成されます。</p>

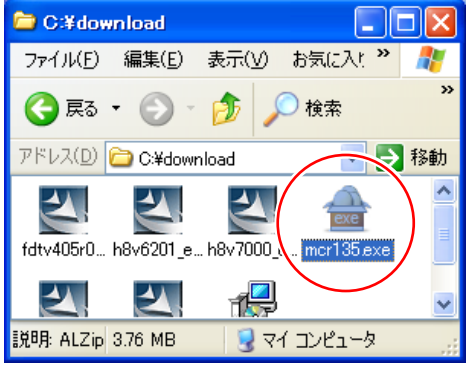
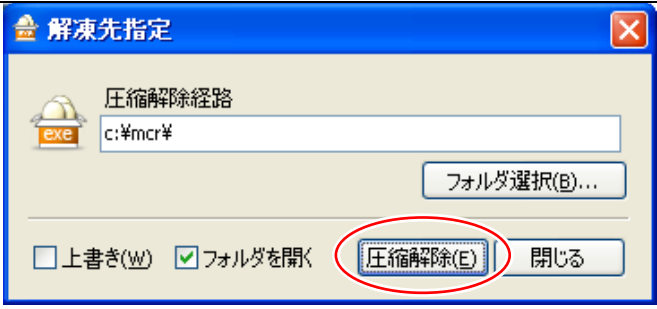
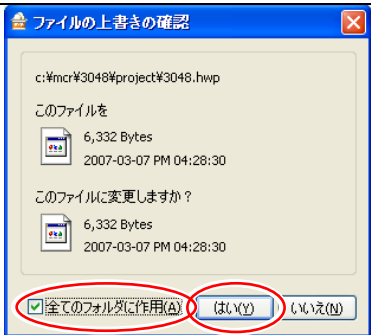
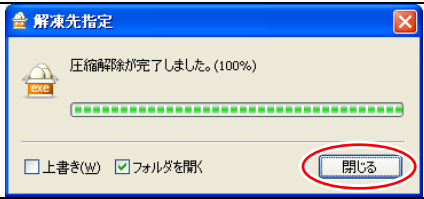
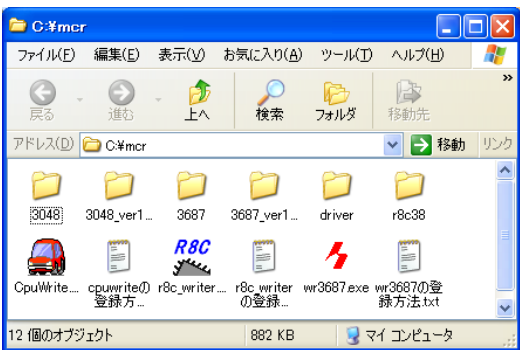
3.5 R8C Writer のダウンロード

R8C Writer とは、R8C/35A マイコンにプログラムを書き込むソフトウェアです。ルネサス統合開発環境に組み込んで使用します。

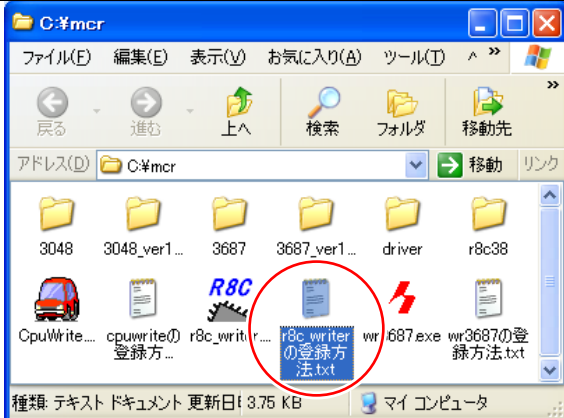
1	 <p>日立のサイト内検索 <input type="text"/> 検索 Powered by Google → 株式会社日立システムソリューションズ JAPAN</p> <p style="text-align: center;">マイコンカーラリー販売 HITACHI Inspire the Next</p> <p>商品一覧 注文・支払・発送方法について 商品のご注文 サイト未掲載商品の注文について ダウンロード お問い合わせ</p> <p>マイコンカーラリー販売</p> <p>日立システムソリューションズは、マイコンカー製作キットをはじめとして、ものづくりから基本的なマイコン制御を試行錯誤しながら学習ができるマイコン学習教材の開発・販売を行っています。</p> <p>関連リンク</p> <ul style="list-style-type: none"> → 日立システムソリューションズ ⇨ マイコンカーラリー公式サイト ⇨ ルネサス エレクトロニクス <p>お問い合わせ</p>	<p>マイコンカーラリー販売サイト 「https://www2.hitachi.com/mcr/」のダウンロードのページへ行きます。</p>
---	--	--

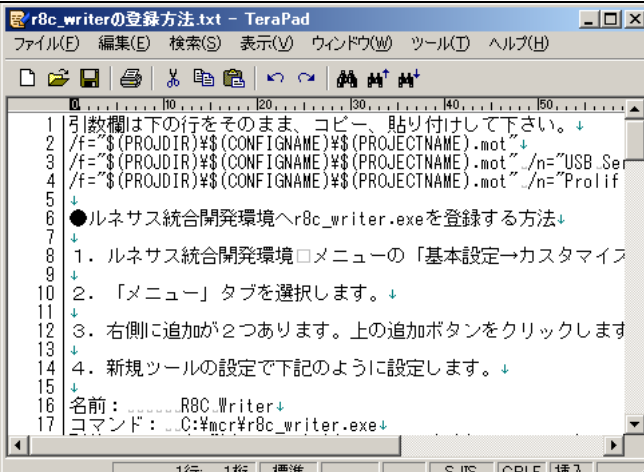
2	<p>開発環境に関する資料</p> <table border="1" style="width: 100%;"> <thead> <tr> <th style="width: 33%;">資料</th> <th style="width: 33%;">マニュアル</th> <th style="width: 33%;">ソフト、プログラム</th> </tr> </thead> <tbody> <tr> <td> <p>ルネサス統合開発環境 操作マニュアル</p> <p>ルネサス統合開発環境のダウンロード方法、インストール方法、操作方法、及びプログラムの書き込み方法を説明しています。ルネサス統合開発環境は、ルネサス エレクトロニクスのサイトよりダウンロードしてください(マニュアル内に方法を記載しています)。</p> </td> <td> <p>ルネサス統合開発環境 操作マニュアル(R8C/38A版) 第1.33版 2015.04.20</p> <p>※R8C/38Aマイコン以外のR8Cマイコンも本マニュアルで開発環境を構築してください。メモリ配置などは変わりますので、各マイコンの仕様に合わせて設定してください。</p> </td> <td> <p>ルネサス統合開発環境用 その他ソフト Ver1.47 mcr_sonota_soft.zip 2015.01.06</p> </td> </tr> <tr> <td>R8C/38Aマイコン、R8C/35Aマイコン</td> <td></td> <td></td> </tr> </tbody> </table>	資料	マニュアル	ソフト、プログラム	<p>ルネサス統合開発環境 操作マニュアル</p> <p>ルネサス統合開発環境のダウンロード方法、インストール方法、操作方法、及びプログラムの書き込み方法を説明しています。ルネサス統合開発環境は、ルネサス エレクトロニクスのサイトよりダウンロードしてください(マニュアル内に方法を記載しています)。</p>	<p>ルネサス統合開発環境 操作マニュアル(R8C/38A版) 第1.33版 2015.04.20</p> <p>※R8C/38Aマイコン以外のR8Cマイコンも本マニュアルで開発環境を構築してください。メモリ配置などは変わりますので、各マイコンの仕様に合わせて設定してください。</p>	<p>ルネサス統合開発環境用 その他ソフト Ver1.47 mcr_sonota_soft.zip 2015.01.06</p>	R8C/38Aマイコン、R8C/35Aマイコン			<p>「ルネサス統合開発環境用その他ソフト」をダウンロードします。このファイルに圧縮されている「mcr_sonota_soft.zip」ファイルを解凍ソフトで解凍し、実行します。</p>
資料	マニュアル	ソフト、プログラム									
<p>ルネサス統合開発環境 操作マニュアル</p> <p>ルネサス統合開発環境のダウンロード方法、インストール方法、操作方法、及びプログラムの書き込み方法を説明しています。ルネサス統合開発環境は、ルネサス エレクトロニクスのサイトよりダウンロードしてください(マニュアル内に方法を記載しています)。</p>	<p>ルネサス統合開発環境 操作マニュアル(R8C/38A版) 第1.33版 2015.04.20</p> <p>※R8C/38Aマイコン以外のR8Cマイコンも本マニュアルで開発環境を構築してください。メモリ配置などは変わりますので、各マイコンの仕様に合わせて設定してください。</p>	<p>ルネサス統合開発環境用 その他ソフト Ver1.47 mcr_sonota_soft.zip 2015.01.06</p>									
R8C/38Aマイコン、R8C/35Aマイコン											


3.6 R8C Writer のインストール

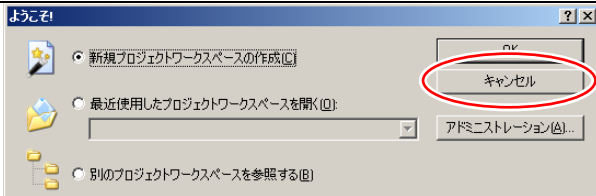
1		<p>解凍した「mcr135.exe」を実行します。</p> <p>※「135」はバージョンです。ダウンロードした時期により異なることがあります。</p>
2		<p>圧縮解除をクリックします。</p> <p>※フォルダ(圧縮解除経路)は替えないでください。替えた場合は、次以降で行う作業が、変わる部分があります。</p>
3		<p>もし、左画面がでてきた場合、「全てのファイルに作用」のチェックを付けて、「はい」をクリックして上書きコピーします。</p> <p>※上書きしたくない場合は、一度終了して、元あるファイルを移動してから、再度実行してください。</p>
4		<p>閉じるをクリックします。</p>
5		<p>インストール先のフォルダが開かれます。</p>

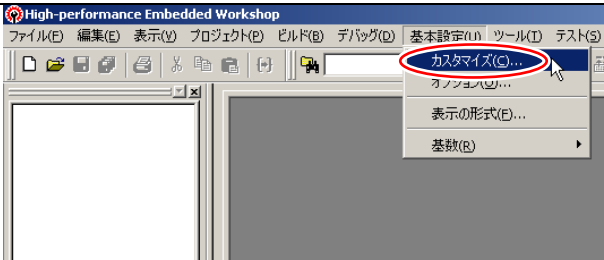
3.7 R8C Writer をルネサス統合開発環境に登録する

1		<p>エクスプローラなどで、「c:\mcr」フォルダを開きます。「r8c_writer の登録方法.txt」を開きます。</p>
---	---	--

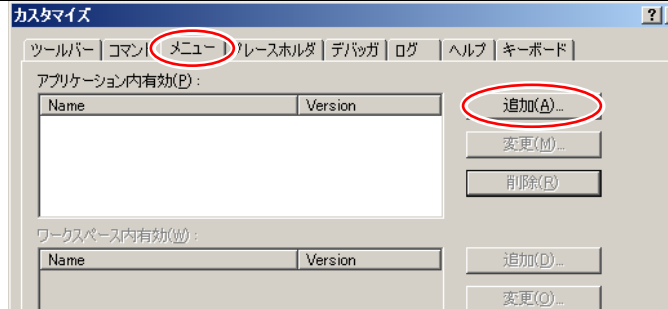
2		<p>この内容は後で使用しますので、最小化しておきます。</p>
---	--	----------------------------------

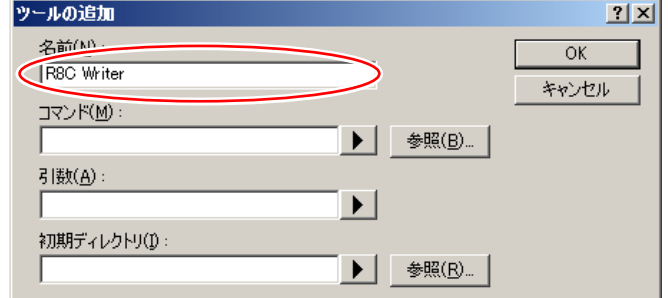
3		<p>ルネサス統合開発環境を実行します。</p>
---	---	--------------------------

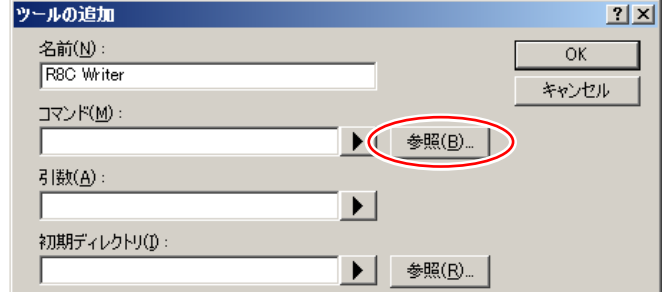
4		<p>キャンセルをクリックします。</p>
---	---	-----------------------

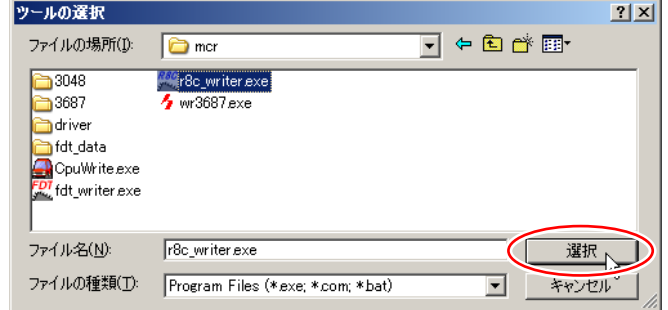
5		<p>「基本設定→カスタマイズ」をクリックします。</p>
---	---	-------------------------------

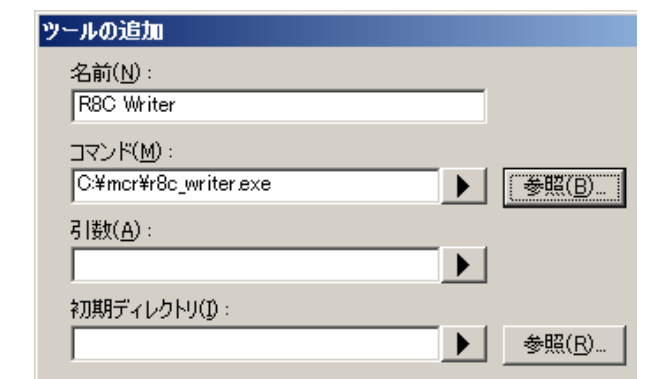
3. 開発環境のダウンロード、インストール

6		<p>「メニュー」タブをクリックし、追加をクリックします。</p> <p>※アプリケーション内有効に既に「R8C Writer」がある場合、登録済みですので、この操作は必要ありません。</p>
---	---	---

7		<p>名前欄に次のように入力します。</p> <p>R8C Writer</p>
---	---	--

8		<p>コマンドを入力します。コマンドとは、書き込みソフトのある場所のことです。参照をクリックします。</p>
---	--	---

9		<p>ファイルを選ぶ画面が出てきます。「C ドライブ→mcr→r8c_writer.exe」を選択します。選択をクリックします。</p>
---	---	---

10		<p>コマンドが入力されました。</p>
----	---	----------------------

3. 開発環境のダウンロード、インストール

<p>11</p>	<p>先ほど開いた「r8c_writer の登録方法.txt」ファイルを開きます。2～4 行目のどれが 1 行だけを選択してコピーしてください。</p> <ul style="list-style-type: none"> ・3 行目を選択すると、ミニマイコンカーVer.2 で使用している FTDI 社製の USB シリアル変換 IC が接続されている COM ポートを自動選択します。ミニマイコンカーVer.2 を使うときは、3 行目を選択してください。 ・2 行目を選択すると、いちばん番号の若い COM 番号を選択します。 ・4 行目を選択すると、RY-WRITER 基板などで使用している Prolific 社製の USB シリアル変換 IC が接続されている COM ポートを自動選択します。RY-WRITER 基板を使うときは、4 行目を選択してください。 <p>コピー後は、「r8c_writer の登録方法.txt」ファイルを閉じて構いません。</p>
-----------	--

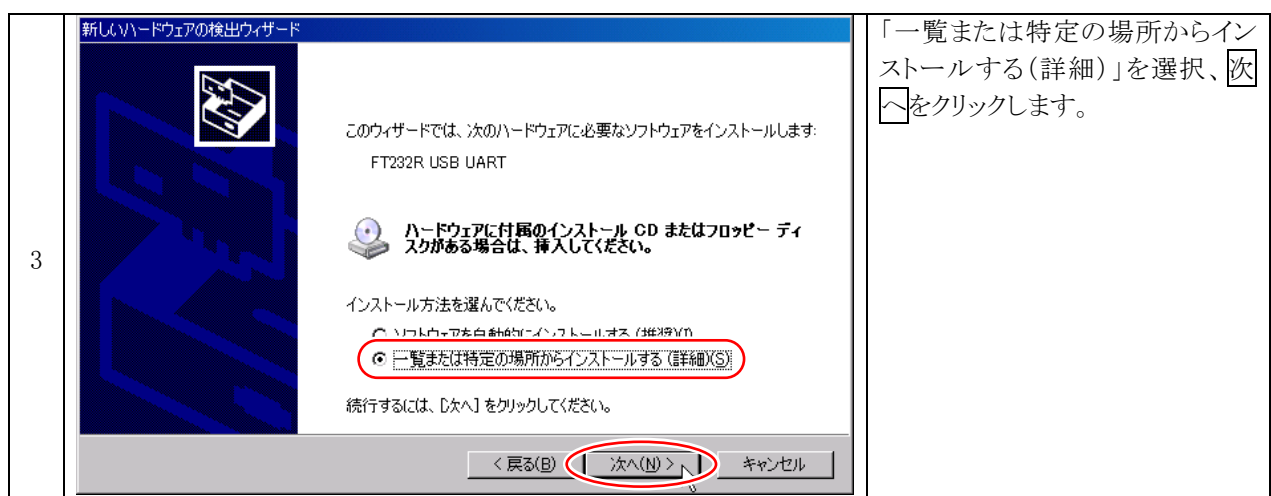
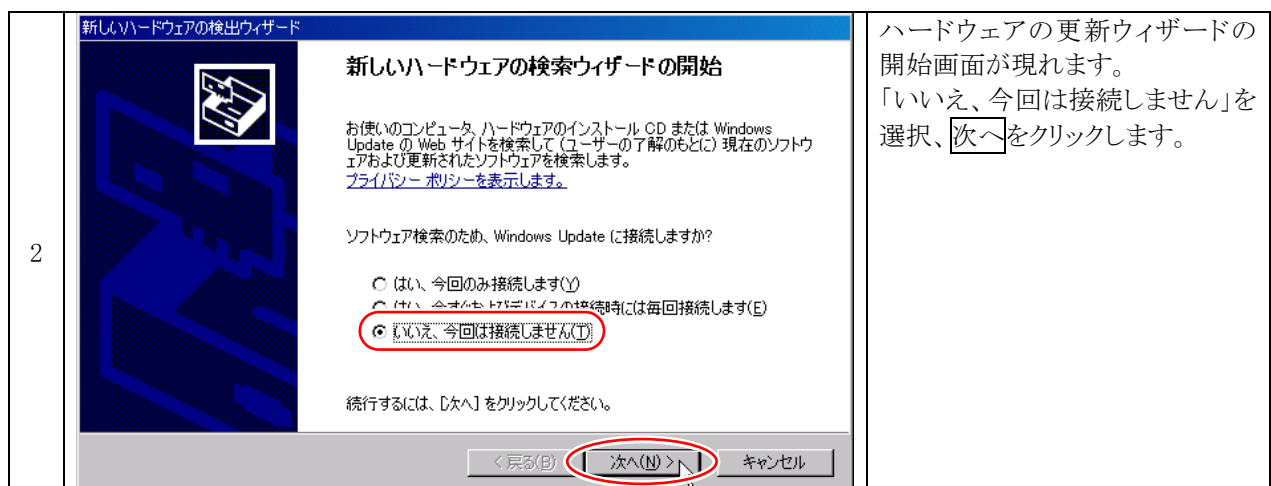
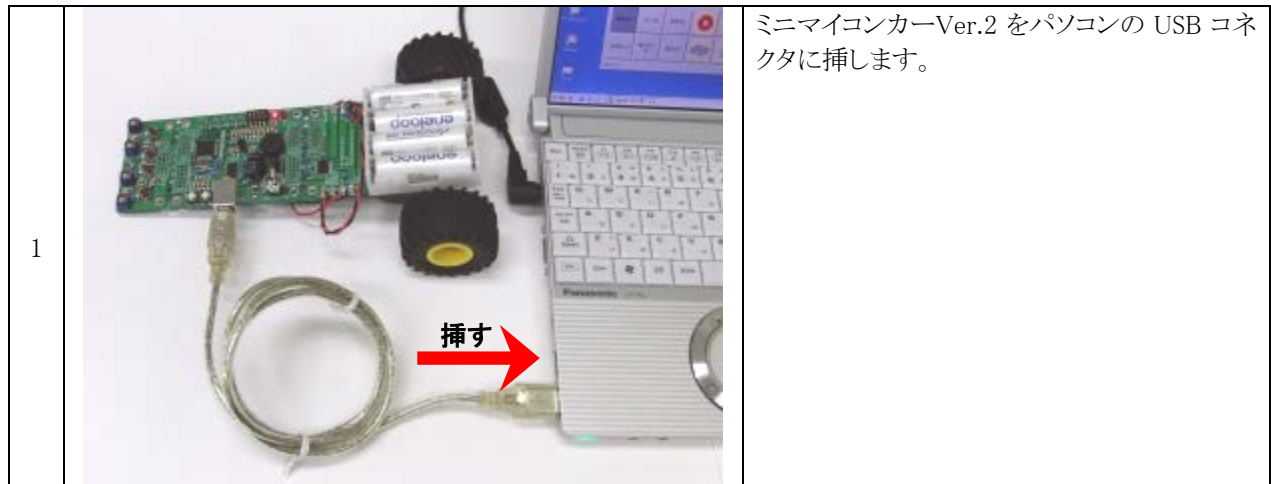
<p>12</p>	<p>ツールの追加画面に戻り、引数欄で右クリックして「貼り付け」をクリックします。</p>
-----------	---

<p>13</p>	<p>OK をクリックして、ツールの追加を完了します。</p>
-----------	---------------------------------

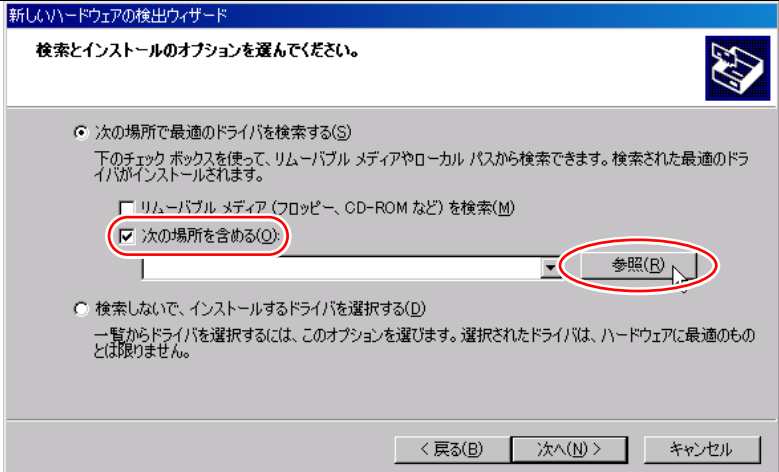
<p>14</p>	<p>アプリケーション内有効に「R8C Writer」があることを確認して、OK をクリックします。無い場合は登録が正しくできていませんので手順を再確認してもう一度登録してください。アプリケーション内有効に他の内容があっても問題ありません。</p>
-----------	--

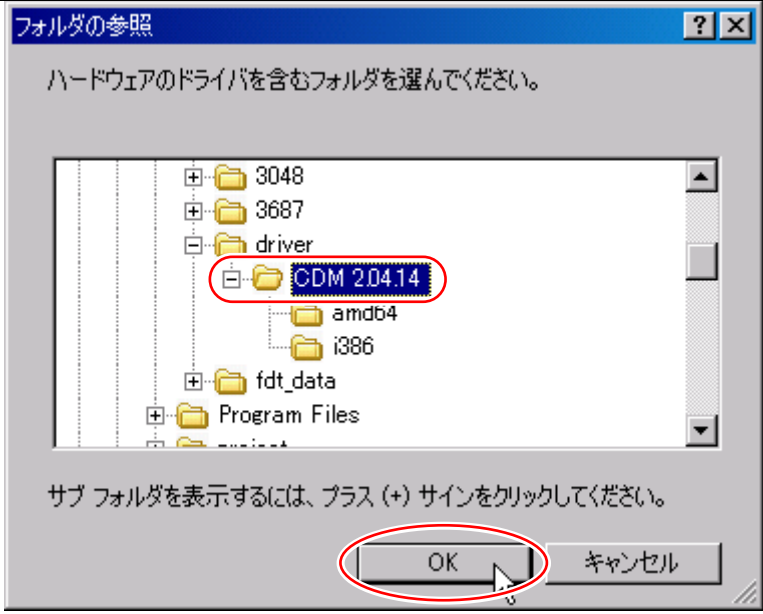
3.8 ドライバーのインストール

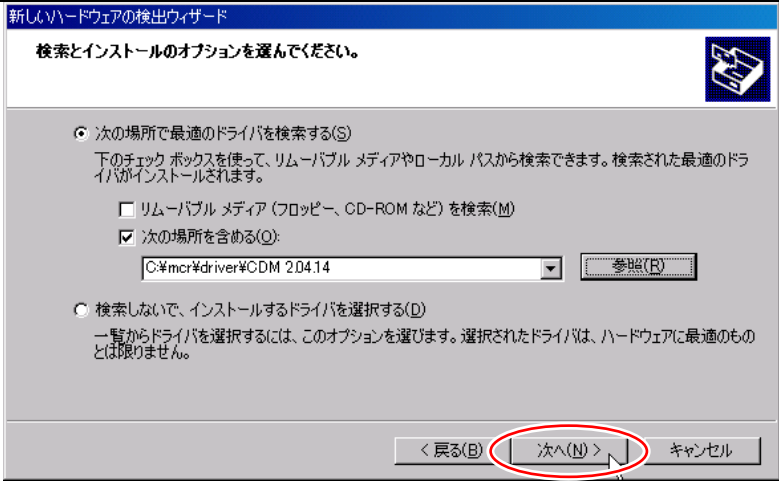
ミニマイコンカーVer.2にはUSB-シリアル変換ICが実装されており、ミニマイコンカーVer.2をパソコンのUSBポートに挿すと、COMポート(シリアルポート)として認識されます。はじめに挿し込んだときは、シリアルポートと認識させるためにドライバーのインストール作業が必要です。インストール方法について説明します。



3. 開発環境のダウンロード、インストール

4	 <p>新しいハードウェアの検出ウィザード</p> <p>検索とインストールのオプションを選んでください。</p> <p>○ 次の場所で最適なドライバを検索する(S) 下のチェック ボックスを使って、リムーバブル メディアやローカル バスから検索できます。検索された最適なドライバがインストールされます。</p> <p><input type="checkbox"/> リムーバブル メディア (フロッピー、CD-ROM など) を検索(M)</p> <p><input checked="" type="checkbox"/> 次の場所を含める(O):</p> <p>_____</p> <p><input type="checkbox"/> 検索しないで、インストールするドライバを選択する(D) 一覧からドライバを選択するには、このオプションを選びます。選択されたドライバは、ハードウェアに最適なものと限りません。</p> <p>< 戻る(B) 次へ(N) > キャンセル</p>	<p>検索とインストールのオプションを選びます。</p> <p>「次の場所を含める」のチェックを入れて、参照をクリックします。</p>
---	---	--

5	 <p>フォルダの参照</p> <p>ハードウェアのドライバを含むフォルダを選んでください。</p> <p>3048 3687 driver CDM 2.01.14 amd64 i386 fdt_data Program Files</p> <p>サブ フォルダを表示するには、プラス (+) サインをクリックしてください。</p> <p>OK キャンセル</p>	<p>C ドライブ ↓ mcr ↓ driver ↓ CDM 2.01.14</p> <p>を選択します。 <input type="checkbox"/> OK をクリックします。</p> <p>※64bit 版の Windows の場合は、 「CDM 2.01.14 (64bit)」など、 「64bit」の名前の付いたフォルダ を選択してください。</p>
---	---	--

6	 <p>新しいハードウェアの検出ウィザード</p> <p>検索とインストールのオプションを選んでください。</p> <p>○ 次の場所で最適なドライバを検索する(S) 下のチェック ボックスを使って、リムーバブル メディアやローカル バスから検索できます。検索された最適なドライバがインストールされます。</p> <p><input type="checkbox"/> リムーバブル メディア (フロッピー、CD-ROM など) を検索(M)</p> <p><input checked="" type="checkbox"/> 次の場所を含める(O):</p> <p>C:\mcr\driver\CDM 2.01.14</p> <p><input type="checkbox"/> 検索しないで、インストールするドライバを選択する(D) 一覧からドライバを選択するには、このオプションを選びます。選択されたドライバは、ハードウェアに最適なものと限りません。</p> <p>< 戻る(B) 次へ(N) > キャンセル</p>	<p>次へをクリックします。 ※検索は、数分程度時間がかかる場合があります。</p>
---	---	---

3. 開発環境のダウンロード、インストール

7		<p>Windows ログテストに合格していないというメッセージが出ますが、特に問題ありません。 続行をクリックします。</p>
---	--	--

8		<p>USB Serial Converter のインストールが完了しました。 完了をクリックします。</p>
---	--	---

9		<p>2回目のハードウェアの検索ウィザードがでできます。 「いいえ、今回は接続しません」を選択、次へをクリックします。</p>
---	--	---

10		<p>USB Serial Port のドライバーをインストールします。 「一覧または特定の場所からインストールする(詳細)」を選択、次へをクリックします。</p>
----	--	--

3. 開発環境のダウンロード、インストール

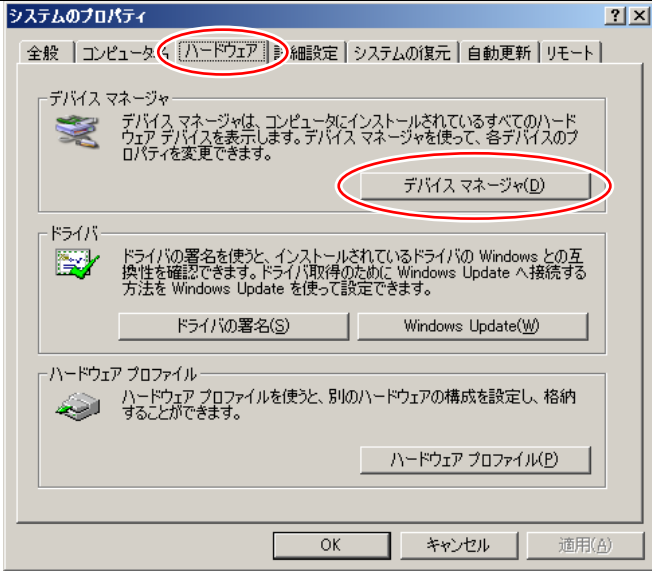
11		<p>検索とインストールのオプションを選びます。</p> <p>「次の場所を含める」のチェックを入れます。</p> <p>場所は「C:\mcr\driver\CDM2.04.14」(先ほどと同じフォルダ)になっていれば「次へ」をクリックします。違う場合は、「参照」をクリックして、先ほどと同じフォルダを選択、「次へ」をクリックします。</p>
----	--	---

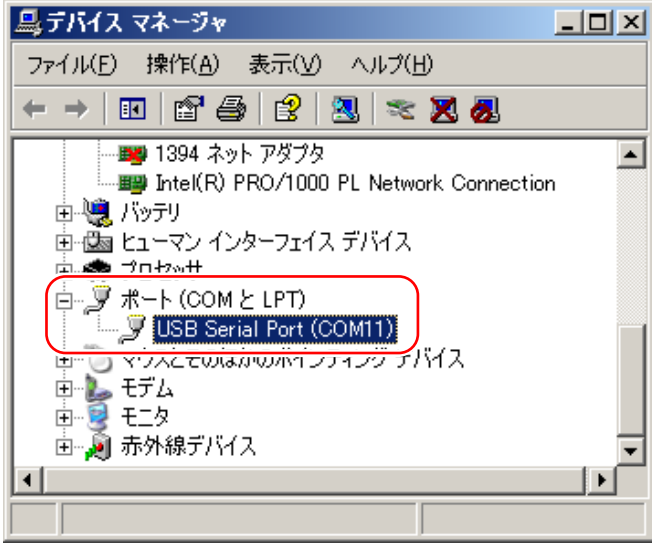
12		<p>Windows ログテストに合格していないというメッセージが出ますが、特に問題ありません。「続行」をクリックします。</p>
----	--	---

13		<p>USB Serial Port のインストールが完了しました。「完了」をクリックします。</p>
----	--	---

14		<p>デスクトップにある「マイコンコンピュータ」で右クリック、プロパティを選択します。デスクトップにマイコンコンピュータが無い場合は、コントロールパネル(クラシック表示)のシステムを選択してください。</p>
----	--	--

3. 開発環境のダウンロード、インストール

15		<p>「ハードウェア」タブを選択します。 デバイスマネージャをクリックします。</p>
----	---	--


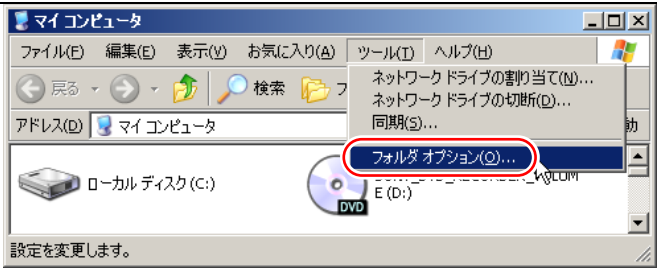
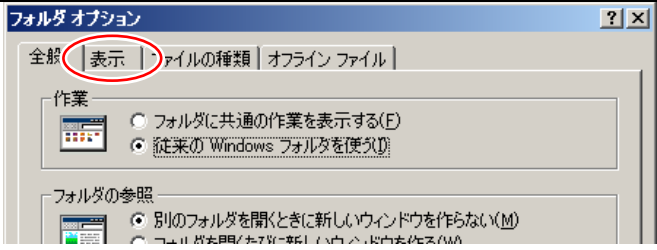
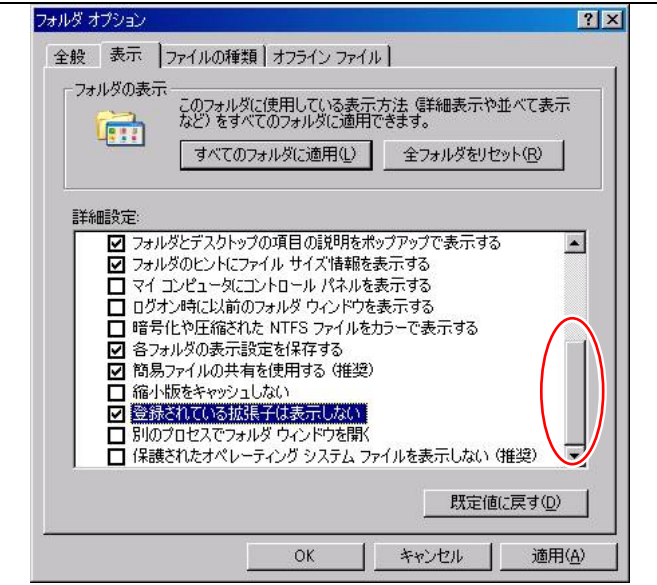
16		<p>「ポート(COMとLPT)」をクリックすると、 USB Serial Port(COM●) があります。この内容が今回インストールした項目です。 USB Serial Port 自体が無い場合、ドライバーのインストールがうまくいっていません。接続しているミニマイコンカーVer.2のUSBケーブルをいったん抜き挿しして、再度ドライバーのインストールをしてください。 ※●部分は、数字です。接続するタイミングによって異なります。</p>
----	--	---

3.9 拡張子の表示

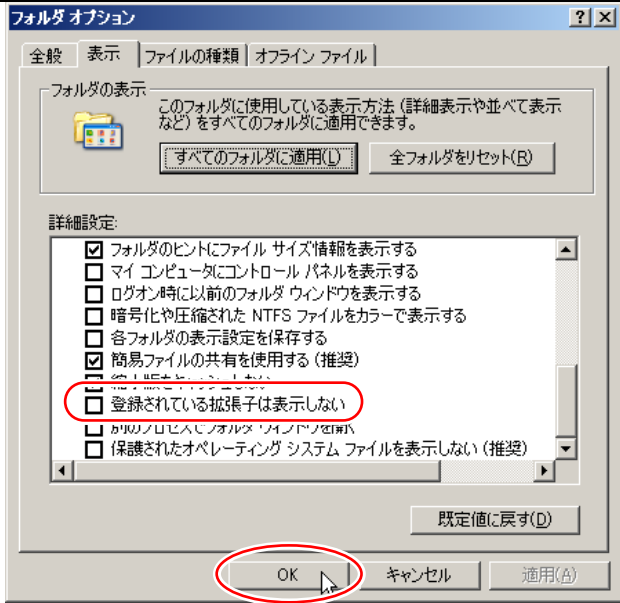
ルネサス統合開発環境で扱うファイルには、「io.c」、「io.obj」、「io.mot」など、拡張子だけ違うファイルが多数あります。そのため、Windows 標準設定の「登録されている拡張子は表示しない」にしておくと、どれがどの種類のファイルか分からなくなります。ここで拡張子を表示する設定にしておきます。

Windows XP の場合は「3.9.1 WindowsXP の場合」を、Windows Vista の場合は「3.9.2 WindowsVista の場合」を参照してください。


3.9.1 WindowsXP の場合

1		<p>マイコンコンピュータのアイコンをダブルクリックします。</p>
2		<p>「ツール→フォルダオプション」をクリックします。</p>
3		<p>「表示」タブをクリックします。</p>
4		<p>スクロールバーを下に移動します。</p>

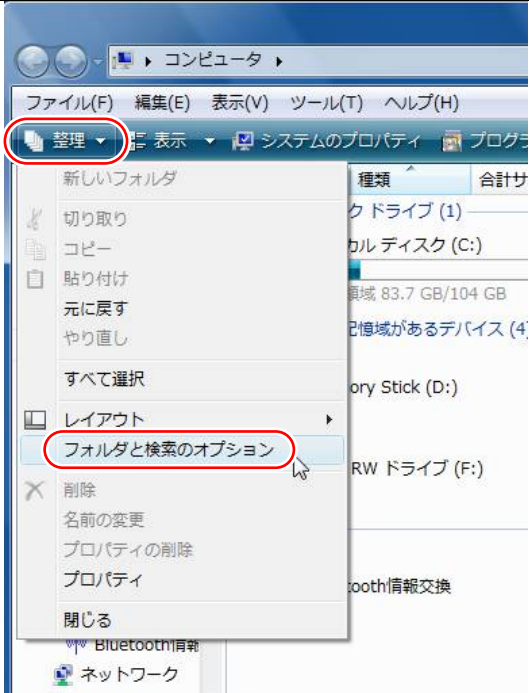
3. 開発環境のダウンロード、インストール

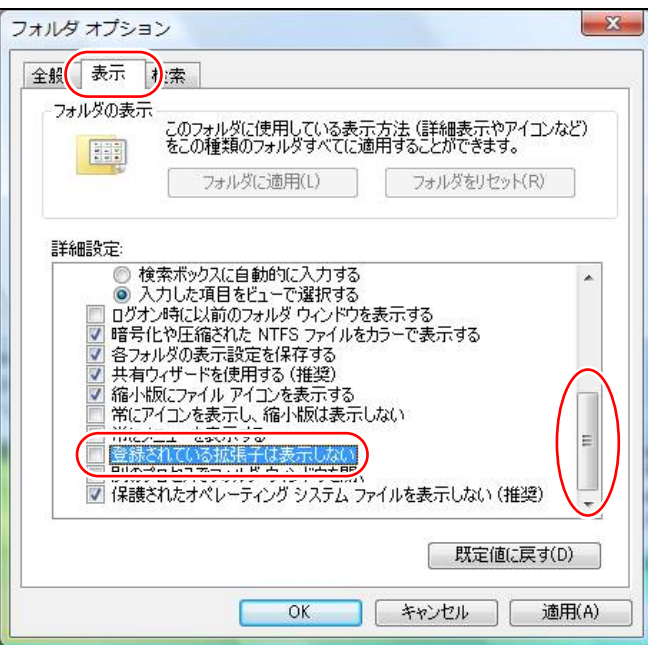
5		<p>「登録されている拡張子は表示しない」のチェックを外します。 OKをクリックして、設定完了です。</p>
---	---	---

3.9.2 WindowsVista の場合

1		<p>「スタート(左下のボタン)→コンピュータ」をクリックします。</p>
---	--	---------------------------------------

3. 開発環境のダウンロード、インストール

2		<p>「整理→フォルダと検索のオプション」をクリックします。</p>
---	---	------------------------------------

3		<p>「表示」タブをクリックします。 詳細設定のスクロールバーを下に移動します。 「登録されている拡張子は表示しない」のチェックを外します。</p>
---	--	---

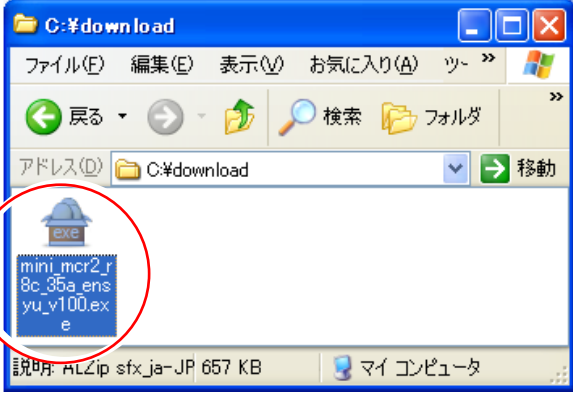

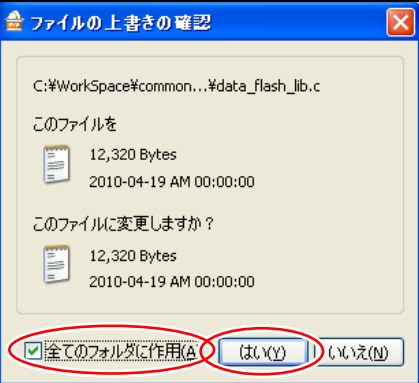
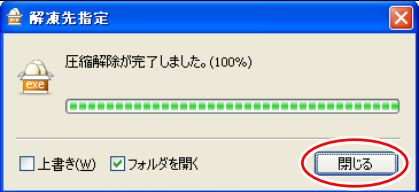
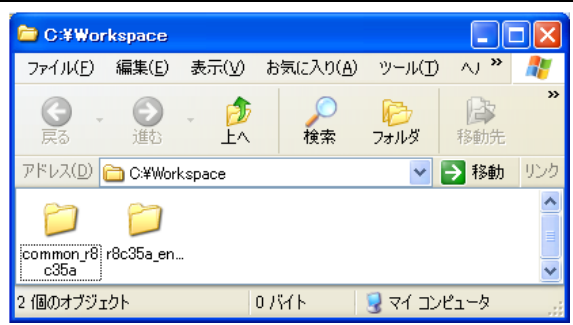
4. サンプルプログラムのダウンロード、インストール

4.1 サンプルプログラムのダウンロード

1	 <p>日立のサイト内検索 <input type="text"/> <input type="button" value="検索"/> Powered by Google → 株式会社日立ドキュメントソリューションズ JAPAN</p> <p style="text-align: center;">マイコンカーラリー販売</p> <p style="text-align: right;">HITACHI Inspire the Next</p> <p>商品一覧 注文・支払・発送方法について 商品のご注文 サイト未掲載商品の注文について ダウンロード → お問い合わせ</p> <p>マイコンカーラリー販売</p> <p>日立ドキュメントソリューションズは、マイコンカー製作キットをはじめとして、ものづくりから基本的なマイコン制御を試行錯誤しながら学習ができるマイコン学習教材の開発・販売を行っています。</p>	<p>マイコンカーラリー販売サイト 「https://www2.himdx.net/mcr/」のダウンロードのページへ行きます。</p>
---	--	--

2	<p style="text-align: center;">ミニマイコンカーVer.2Iに関する資料</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 30%;">基板・資料</th> <th style="width: 20%;">製作マニュアル</th> <th style="width: 20%;">プログラム解説マニュアル</th> <th style="width: 30%;">プログラム</th> </tr> </thead> <tbody> <tr> <td> <p>ミニマイコンカー製作キット Ver.2 ワンボードで総合的にマイコン学習ができる教材です。ミニマイコンカーの製作、モーターの制御やスイッチを使用したLEDの点灯制御、プザーを使用した電子オルゴールの制御などをとおして、マイコン制御に興味をもって学習ができます。拡張用のオプションなどを組み合わせることによって、メカトロニクスの基礎から応用まで幅広く学習することが可能です。</p> <p>●ブロック・コマンダー ミニマイコンカー製作キット Ver.2専用のプログラムソフトです。難しいプログラム言語が分からなくても、ブロックを組み合わせることによりプログラムを作成することができます。</p> <p>●ブロック・シミュレーター ブロック・コマンダーで作成した走行プログラムを、PCでシミュレーションすることができます。</p> </td> <td style="text-align: center;"> <p>ミニマイコンカー製作キット Ver.2 組み立てマニュアル 第2.21A版 2015.04.27</p> </td> <td style="text-align: center;"> <p>ブロック・コマンダー 操作マニュアル 第1.51版 2015.04.20</p> </td> <td> <p>●ブロック・コマンダー bcv106_r00.zip 2015.01.05</p> <p>●ブロック・シミュレーター bsv104_r03.zip 2015.01.05</p> </td> </tr> <tr> <td> <p>ミニマイコンカー製作キット Ver.2 C言語走行プログラム ブロック・コマンダーを使わず、C言語の走行プログラムでライントレースをすることができます。</p> </td> <td style="text-align: center;">-</td> <td style="text-align: center;"> <p>ミニマイコンカー製作キット Ver.2 C言語走行プログラム解説マニュアル 第1.05版 2015.07.15</p> </td> <td style="text-align: center;"> <p>mini_mcr2_v103.zip 2010.06.28</p> </td> </tr> </tbody> </table>	基板・資料	製作マニュアル	プログラム解説マニュアル	プログラム	<p>ミニマイコンカー製作キット Ver.2 ワンボードで総合的にマイコン学習ができる教材です。ミニマイコンカーの製作、モーターの制御やスイッチを使用したLEDの点灯制御、プザーを使用した電子オルゴールの制御などをとおして、マイコン制御に興味をもって学習ができます。拡張用のオプションなどを組み合わせることによって、メカトロニクスの基礎から応用まで幅広く学習することが可能です。</p> <p>●ブロック・コマンダー ミニマイコンカー製作キット Ver.2専用のプログラムソフトです。難しいプログラム言語が分からなくても、ブロックを組み合わせることによりプログラムを作成することができます。</p> <p>●ブロック・シミュレーター ブロック・コマンダーで作成した走行プログラムを、PCでシミュレーションすることができます。</p>	<p>ミニマイコンカー製作キット Ver.2 組み立てマニュアル 第2.21A版 2015.04.27</p>	<p>ブロック・コマンダー 操作マニュアル 第1.51版 2015.04.20</p>	<p>●ブロック・コマンダー bcv106_r00.zip 2015.01.05</p> <p>●ブロック・シミュレーター bsv104_r03.zip 2015.01.05</p>	<p>ミニマイコンカー製作キット Ver.2 C言語走行プログラム ブロック・コマンダーを使わず、C言語の走行プログラムでライントレースをすることができます。</p>	-	<p>ミニマイコンカー製作キット Ver.2 C言語走行プログラム解説マニュアル 第1.05版 2015.07.15</p>	<p>mini_mcr2_v103.zip 2010.06.28</p>	<p>「mini_mcr2_v103.zip」をダウンロード、解凍ソフトで解凍します。</p> <p>※「103」はバージョンです。ダウンロードした時期により異なることがあります。</p>
基板・資料	製作マニュアル	プログラム解説マニュアル	プログラム											
<p>ミニマイコンカー製作キット Ver.2 ワンボードで総合的にマイコン学習ができる教材です。ミニマイコンカーの製作、モーターの制御やスイッチを使用したLEDの点灯制御、プザーを使用した電子オルゴールの制御などをとおして、マイコン制御に興味をもって学習ができます。拡張用のオプションなどを組み合わせることによって、メカトロニクスの基礎から応用まで幅広く学習することが可能です。</p> <p>●ブロック・コマンダー ミニマイコンカー製作キット Ver.2専用のプログラムソフトです。難しいプログラム言語が分からなくても、ブロックを組み合わせることによりプログラムを作成することができます。</p> <p>●ブロック・シミュレーター ブロック・コマンダーで作成した走行プログラムを、PCでシミュレーションすることができます。</p>	<p>ミニマイコンカー製作キット Ver.2 組み立てマニュアル 第2.21A版 2015.04.27</p>	<p>ブロック・コマンダー 操作マニュアル 第1.51版 2015.04.20</p>	<p>●ブロック・コマンダー bcv106_r00.zip 2015.01.05</p> <p>●ブロック・シミュレーター bsv104_r03.zip 2015.01.05</p>											
<p>ミニマイコンカー製作キット Ver.2 C言語走行プログラム ブロック・コマンダーを使わず、C言語の走行プログラムでライントレースをすることができます。</p>	-	<p>ミニマイコンカー製作キット Ver.2 C言語走行プログラム解説マニュアル 第1.05版 2015.07.15</p>	<p>mini_mcr2_v103.zip 2010.06.28</p>											


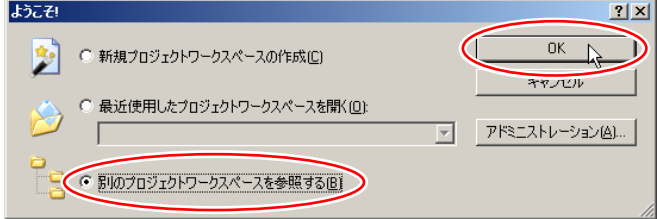
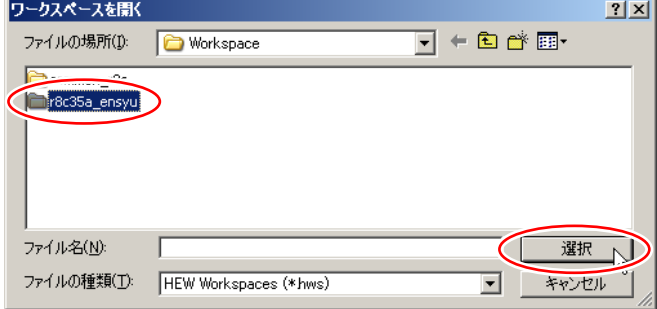
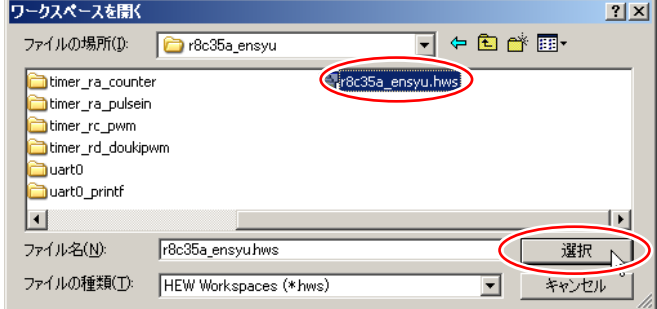
4.2 サンプルプログラムのインストール

1		<p>解凍した、「mini_mcr2_r8c_35a_ensyu_v100.exe」を実行します。 「100」部分はバージョンです。100 以上のバージョンのファイルでインストールしてください。</p>
2		<p>圧縮解除をクリックします。</p> <p>※圧縮解除経路(フォルダ)は替えないでください。替えた場合は、ルネサス統合開発環境のツールチェーンの設定変更が必要です。</p>
3		<p>ファイルの上書き確認が画面が出てきた場合、「全てのフォルダに作用」のチェックを付けて、「はい」をクリックします。</p>
4		<p>閉じるをクリックします。</p>
5		<p>インストールが完了すると、「C:\workspace」フォルダが自動で開きます。</p>

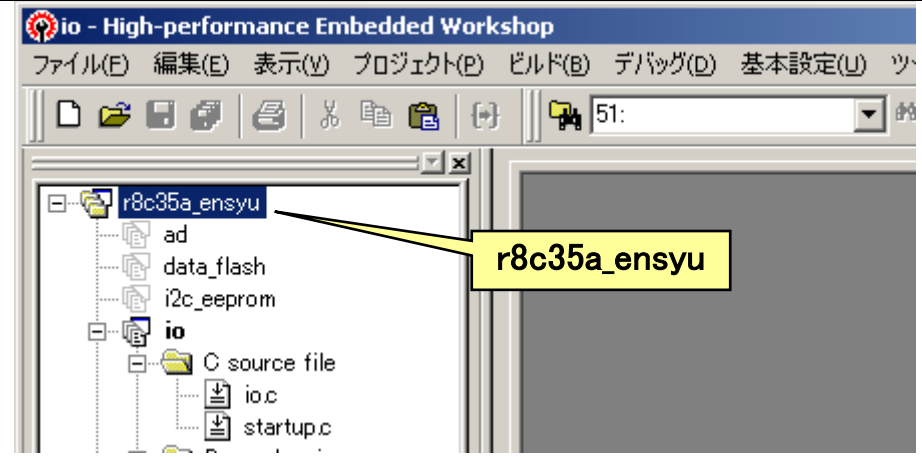
5. 演習手順

これから、サンプルプログラムを使って、演習をしていきます。その前に、操作手順を説明していきます。詳しくは、マイコンカーラー販売サイトのダウンロードページにある「ルネサス統合開発環境 操作マニュアル」を参照してください。

5.1 ワークスペースを開く

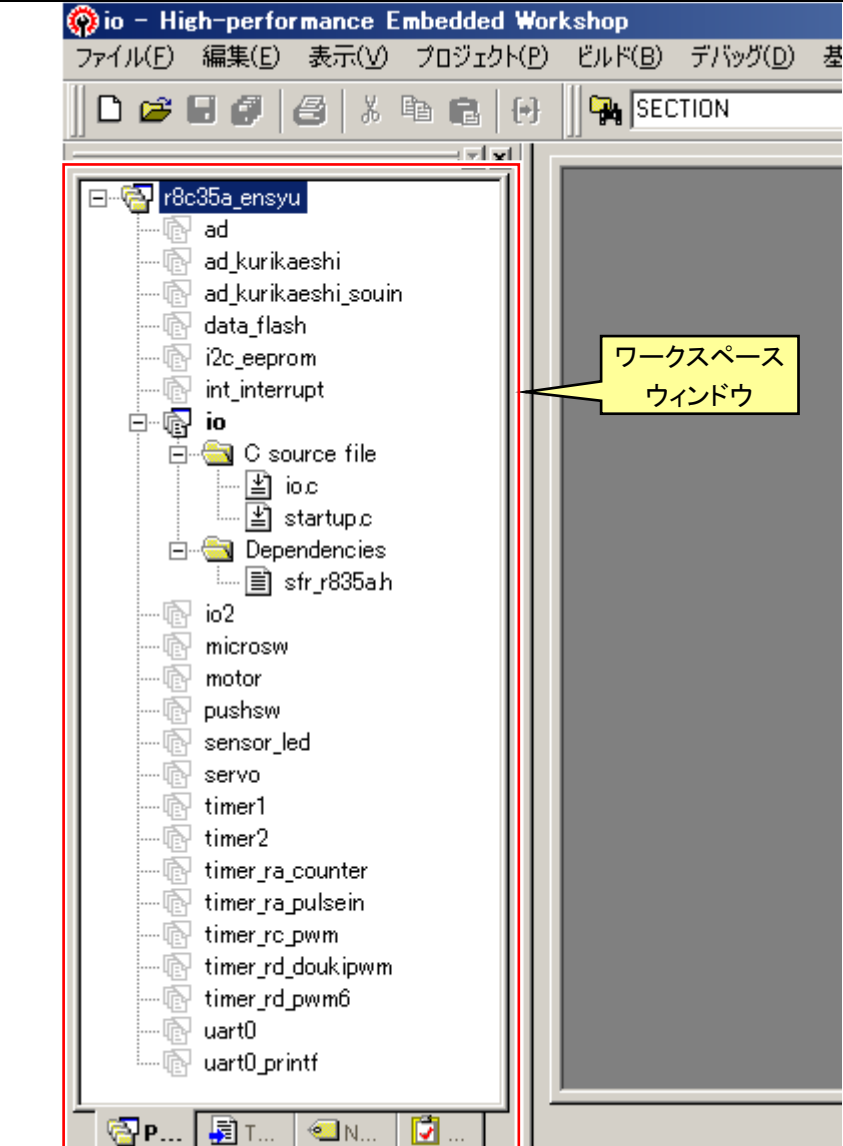
1		ルネサス統合開発環境を実行します。
2		「別のプロジェクトワークスペースを参照する」を選択、 OK をクリックします。
3		「r8c35a_ensyu」を選択、 選択 をクリックします。
4		「r8c35a_ensyu.hws」を選択、 選択 をクリックします。

5. 演習手順

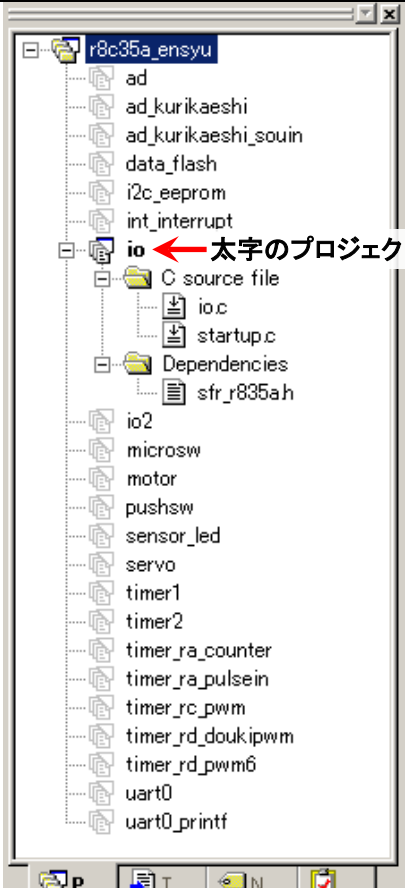
5		<p>「r8c35a_ensyu」というワークスペースが開かれました。本実習マニュアルでは、このワークスペースを使用します。</p>
---	--	--

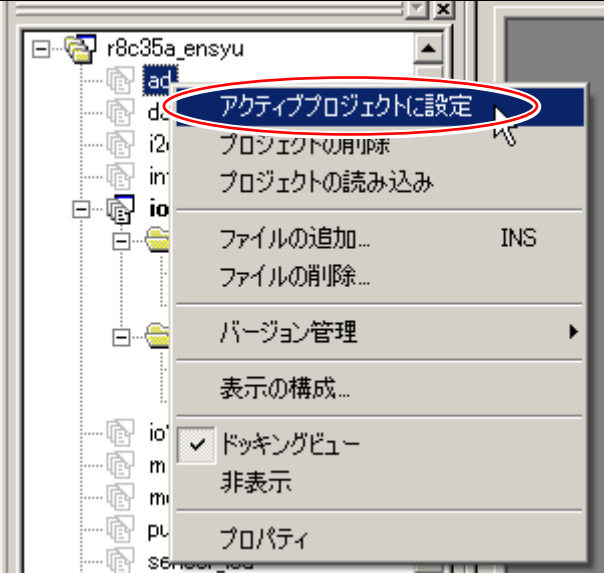
5.2 プロジェクトを開く

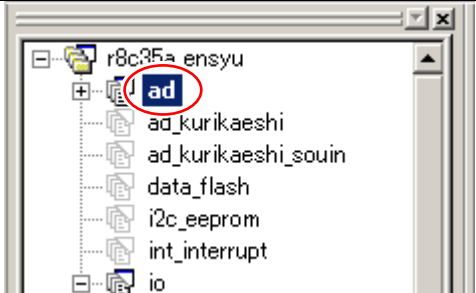
例えば、これからプロジェクト「ad」の演習をしますとします。

1		<p>ワークスペースウィンドウと呼ばれるスペースにたくさんのプロジェクトが登録されています。</p>
---	---	--

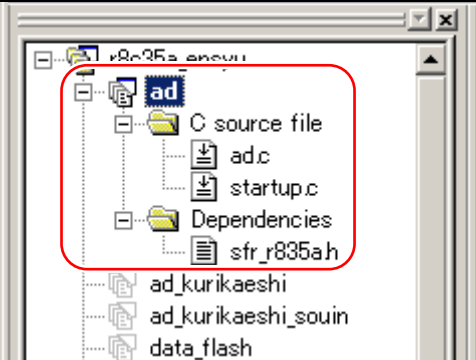
5. 演習手順

2		<p>上から、 ad, ad_kurikaeshi, ad_kurikaeshi_souin, data_flash, i2c_eeprom, int_interrupt, io, io2, motor, pushsw, sensor_led, servo, timer1, timer2, timer_ra_counter, timer_ra_pulsein, timer_rc_pwm, timer_rd_doukipwm, timer_rd_pwm6, uart0, uart0_printf というプロジェクトがあります。 プロジェクト＝演習の単位となります。</p> <p>左画面では、プロジェクト「io」が太字になっています。このプロジェクトが現在有効なプロジェクト(アクティブプロジェクト)です。有効なプロジェクトは、</p> <ul style="list-style-type: none"> ・ビルドの対象 ・ツールチェーン(各種設定)の対象 ・書き込みの対象 <p>となります。「ad」の演習を行うので、「ad」を有効なプロジェクトにしなければいけません。</p>
---	--	---

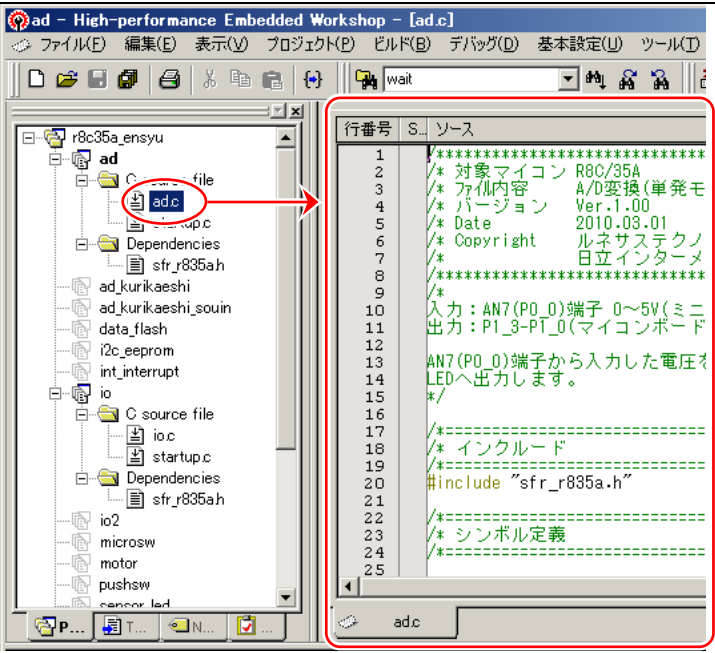
3		<p>プロジェクト「ad」上で右クリック、「アクティブプロジェクトに設定」をクリックします。</p>
---	---	--

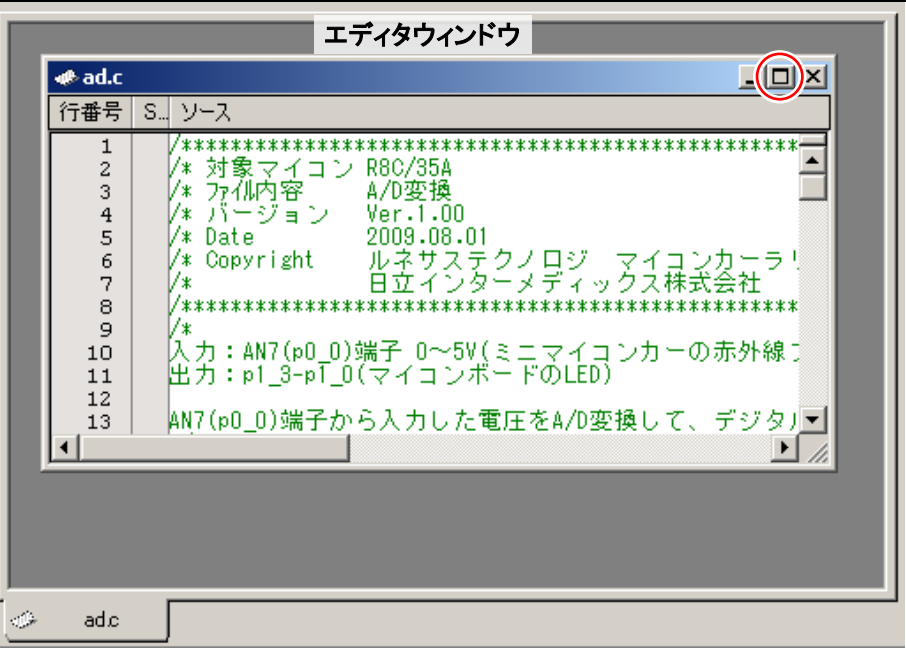
4		<p>「ad」が太字になりました。これでアクティブプロジェクトの設定完了です。</p>
---	---	---

5. 演習手順

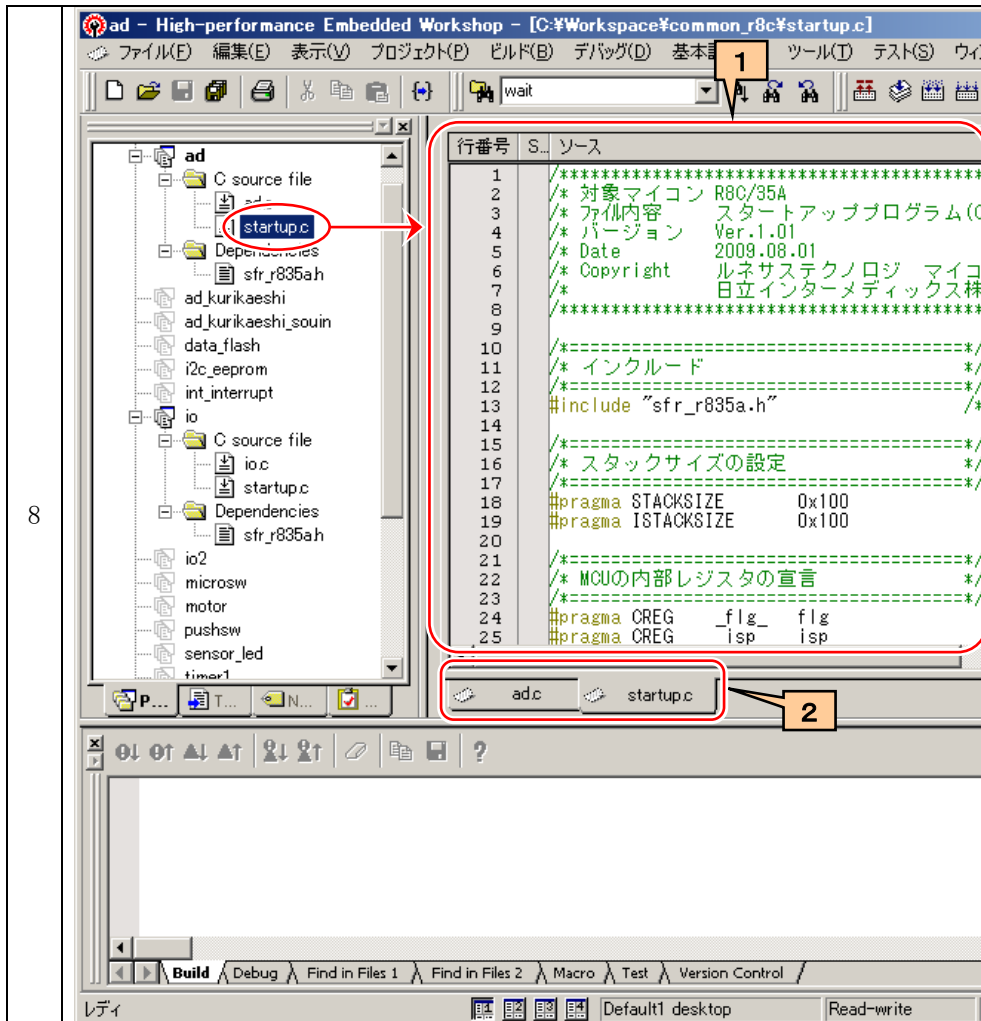
5		<p>「ad」をダブルクリックします。 これがプロジェクト「ad」に登録されているファイルです。</p> <ul style="list-style-type: none"> ● ad.c ● startup.c <p>という 2 ファイルが登録されています。 Dependencies に登録されているファイルを、「依存ファイル」といい、主にヘッダファイルが登録されています。今回は、「sfr_r835a.h」が登録されています。</p>
---	---	--

5.3 ファイル編集

6		<p>「ad.c」をダブルクリックすると、エディタウィンドウが開きます。ここでファイルを編集します。</p>
---	---	--

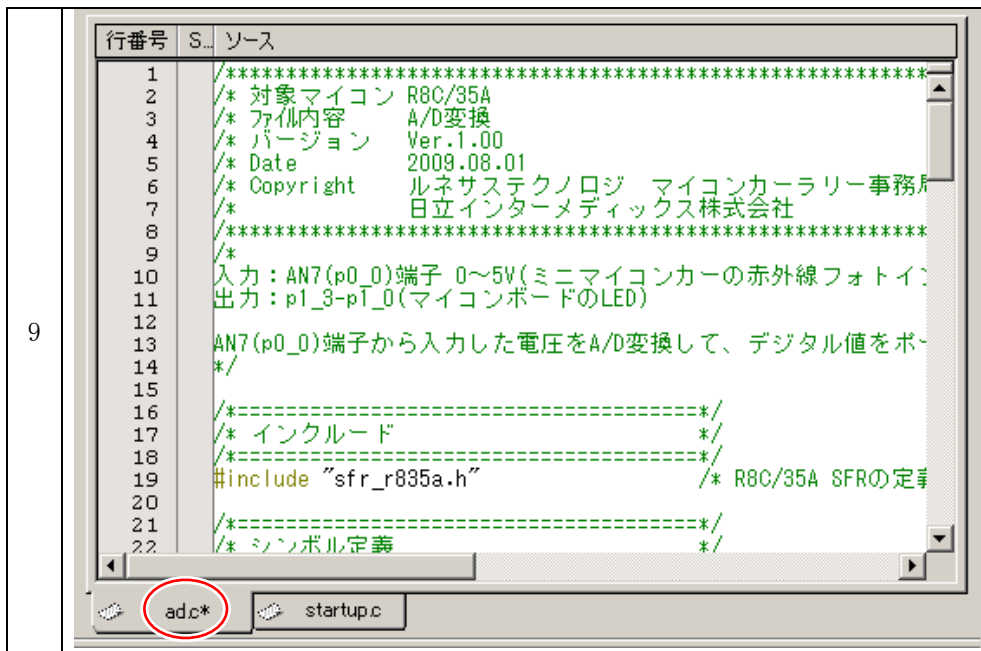
7		<p>エディタウィンドウが小さく開いたとき、枠全体に広げるなら○部分の最大化ボタンをクリックします。</p>
---	--	--

5. 演習手順



①「startup.c」をダブルクリックすると、startup.cのエディタウィンドウが開きます。

②2つのファイルを開きました。ファイルを切り替えるには、タブで編集したいファイル名を選びます。



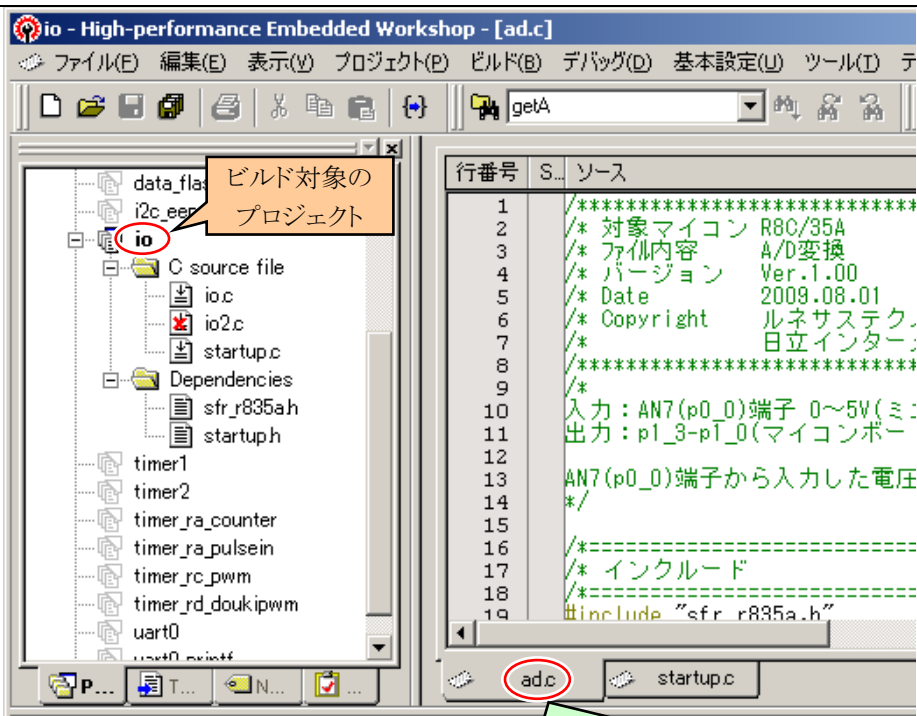
ファイルを編集して内容が変更されると、エディタウィンドウのタブのファイル名の右に「*」が表示されます。

5. 演習手順

10		<p>プログラムを変更した場合、保存しなければいけません。保存ボタンは次の2つあります。</p> <p>①…現在編集集中のファイルを保存するボタンです。</p> <p>②…変更したすべてのファイルを保存するボタンです。</p> <p>②のアイコンを使用すれば変更したすべてのファイルが保存されますので、適宜②のアイコンで保存してください。</p>
----	--	---

11		<p>編集が終わったら、をクリックしてファイルを閉じます。編集が終わって表示する必要のないファイルは、間違って編集してしまわないよう適宜閉じるようにしましょう。</p>
----	--	--

5.4 ビルド(MOT ファイルの作成)

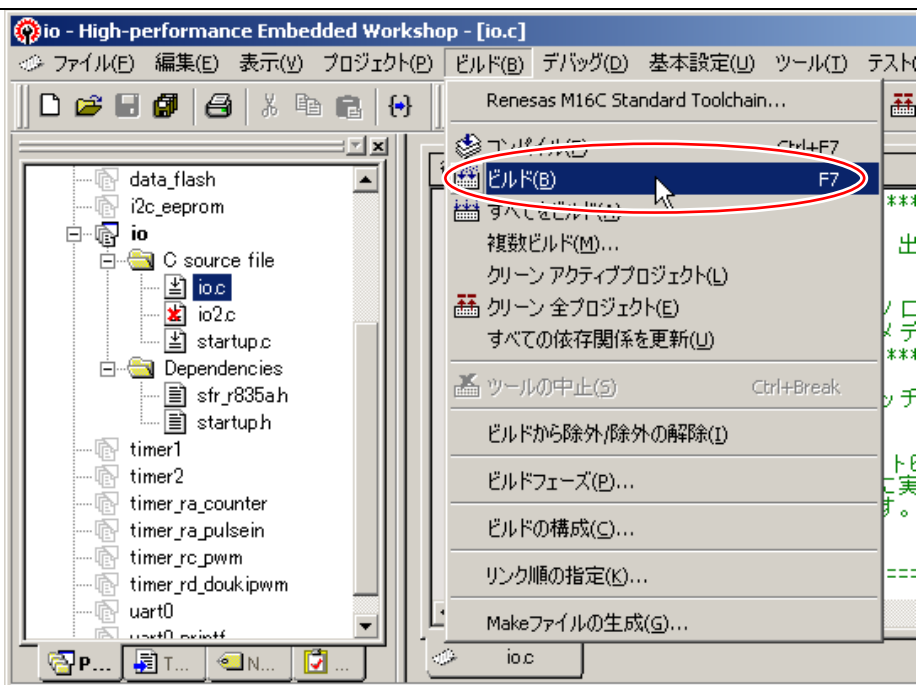


1

ad.c が表示されていても、ビルドされるのは、io プロジェクト内のファイルです。プロジェクトを変えるときは開いているファイルを閉じるという癖を付けておくとうい良いでしょう。今回の場合は、「ad.c」は閉じておきましょう。

ビルドとは、プロジェクトに登録されているC言語ソースファイルなどを翻訳して、マイコンに書き込む最終ファイルである MOT ファイルを作成することです。

ビルドは、現在有効なプロジェクトが対象となります。必ず有効なプロジェクトを確認してください。エディタウインドウに表示されているファイルとは、全く無関係です。例えば、エディタウインドウに ad.c が表示されていても、有効なプロジェクトが「io」なら io プロジェクトに関するファイルである「io.c」と「startup.c」がビルドされるファイルです。



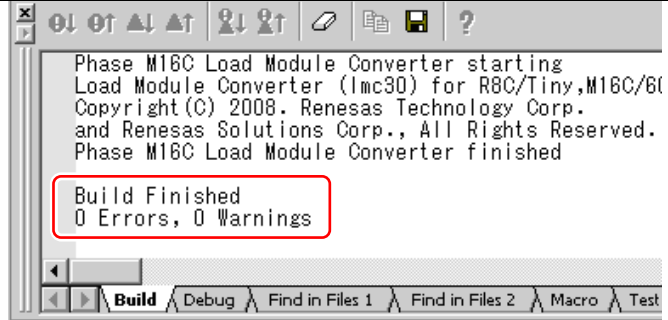
2

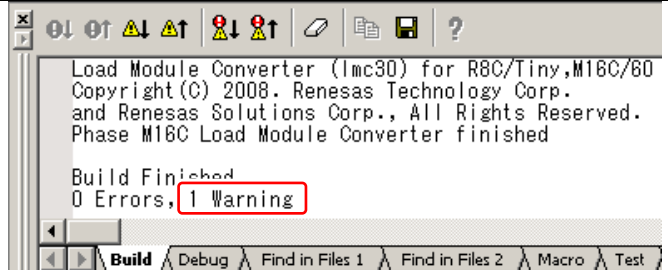
ビルドは、次の2種類があります。

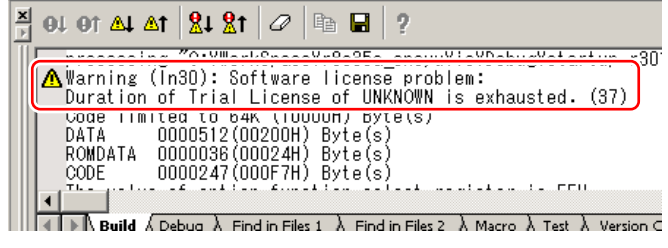
- 「ビルド→ビルド」
更新したファイルを自動で検出して、必要なファイルだけビルドします。
- 「ビルド→すべてをビルド」
ファイルリストに登録しているファイルのすべてをビルドします。

通常は、「ビルド」で問題ありません。

5. 演習手順

3	 <p>Phase M16C Load Module Converter starting Load Module Converter (lmc30) for R8C/Tiny,M16C/61 Copyright (C) 2008. Renesas Technology Corp. and Renesas Solutions Corp., All Rights Reserved. Phase M16C Load Module Converter finished</p> <p>Build Finished 0 Errors, 0 Warnings</p> <p>Build Debug Find in Files 1 Find in Files 2 Macro Test</p>	<p>ビルドを実行すると、自動的にアセンブル、コンパイル、リンク作業に入り、結果が左画面のように表示されます。</p> <p>●Error 誤りのことです。これが出た場合は必ずプログラムやツールチェーンの設定を直します。</p> <p>●Warning 警告です。必ずしも誤っているとは言い切れないけども、間違っている可能性があるので確認してくださいというメッセージです。こちらも必ず直します。</p> <p>Errors や Warnings が "0" なら、プログラムに誤りはないということで MOT ファイルが作成されます。もし、Errors や Warnings が 1 つでもあれば、正常にビルドができていないので MOT ファイルができていないか、もしくはできていても不完全な状態である可能性があります。プログラムの問題箇所を訂正して、エラーが無くなるまで再度ビルドしてください。</p>
---	---	---

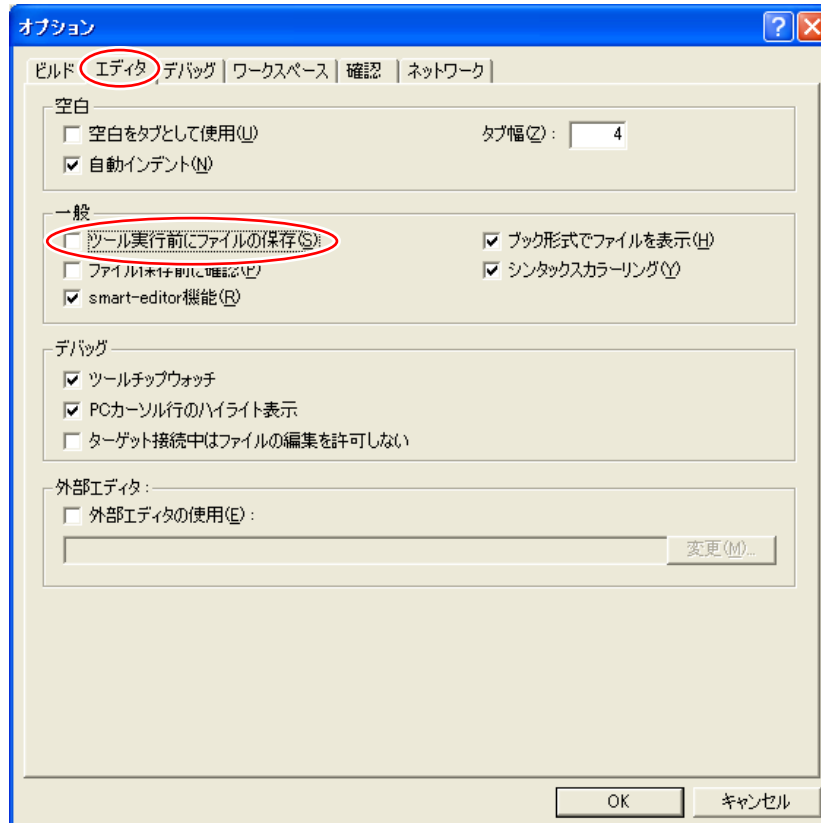
4	 <p>Load Module Converter (lmc30) for R8C/Tiny,M16C/60 Copyright (C) 2008. Renesas Technology Corp. and Renesas Solutions Corp., All Rights Reserved. Phase M16C Load Module Converter finished</p> <p>Build Finished 0 Errors, 1 Warning</p> <p>Build Debug Find in Files 1 Find in Files 2 Macro Test</p>	<p>ただし、Warning が必ず出ることがあります。それは、ルネサス統合開発環境の無償評価版の場合、インストールしてから 60 日以上経つと、64KB までしかビルドすることができなくなります。60 日以上経つと、このメッセージが Warning としてでるようになります。</p>
---	---	---

5	 <p>Warning (In30): Software license problem: Duration of Trial License of UNKNOWN is exhausted. (37)</p> <p>Code limited to 64K (10000H) byte(s) DATA 0000512(00200H) Byte(s) ROMDATA 0000036(00024H) Byte(s) CODE 0000247(000F7H) Byte(s) The value of section function select register is FFH</p> <p>Build Debug Find in Files 1 Find in Files 2 Macro Test Version C</p>	<p>ワーニングメッセージを確認しておきます。</p> <p>Warning (In30): Software license problem:</p> <p>という、ライセンスに問題があるというメッセージです。このメッセージ(ワーニング)は気にしないで構いません。</p>
---	---	--

※ファイルの保存について

ビルドを実行すると、自動でファイルの保存が行われます。**すぐにビルドを行う場合は、ファイルを保存する必要はありません。**保存ボタンは、ファイルの編集のみを行いビルドしないとき実行してください。

もし、自動保存をしたくない場合は、「基本設定→オプション」でオプション画面を開きます。「エディタ」タブを選び、「一般:ツール実行前にファイルの保存」のチェックを外します。



この機能は、複数のファイルを編集集中、誤ってビルドボタンを押して、勝手に保存されないようにするときにチェックを外しておく便利です。

5. 演習手順

5.5 プログラムの書き込み

1

マイコンボードの準備をします。

①パソコンとマイコンボードを USB ケーブルで接続します (接続されていればそのまま構いません)。

②電源スイッチを OFF にします。写真でいうとスイッチを左側に倒します。

③リセットボタンを押します。

このとき、マイコンボード左下にある2個のLED が次の状態になっているか確認してください。

(POWER): 消灯
(USB): 点灯

2

ルネサス統合開発環境の「ツール→R8C Writer」をクリックします。

5. 演習手順

3		<p>書き込みソフトが起動します。</p> <p>①「通信ポート」の番号をミニマイコンカー Ver.2 のポート番号に設定します。通信ポートの番号が分からない場合は、「※通信ポートの番号が分からない場合」を参照してください。</p> <p>②「書き込み開始」をクリックすると、書き込みを開始します。</p> <p>※書き込み後、ベリファイチェックする 「書き込み後、ベリファイチェックする」のチェックを付けると、書き込み後、書き込みデータが正しいか確認します。チェック ON がお勧めです。</p> <p>※書き込み完了時、自動終了する 「書き込み完了時、自動終了する」のチェックを付けると、書き込み完了時に R8C Writer が自動終了します。チェック ON がお勧めです。</p>
---	--	--

4		<p>R8C Writer で書き込み中です。書き込みが完了したら、終了をクリックして、R8C Writer を終了させてください。「書き込み完了時、自動終了する」のチェックが付いていると、自動で終了します。</p>
---	--	--

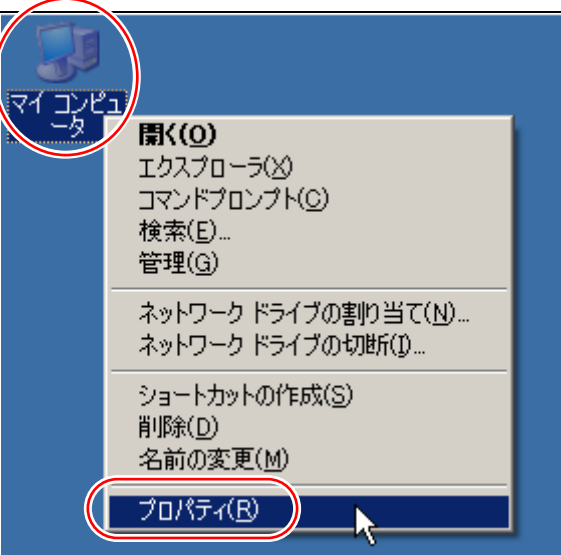
5		<p>もし、書き込みができなければ、左画面のようなエラー画面が出てきます。USB ケーブルやマイコンの状態を確認して、OK をクリック、再度書き込みを実行してください。</p>
---	--	--

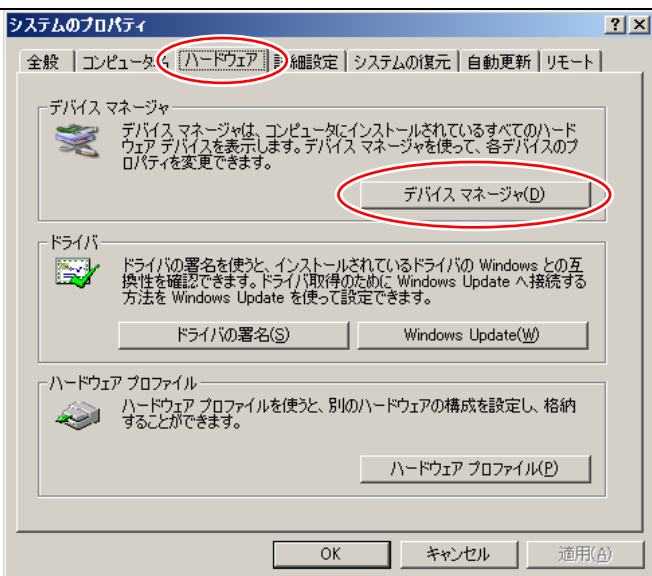
5. 演習手順


6		<p>書き込みが終わったら、次の手順で書き込んだプログラムを実行させます。</p> <p>①USB ケーブルは接続していても抜いても、どちらでも構いません(ミニマイコンカーを走らせる場合はもちろん抜きます)。</p> <p>②電源スイッチを ON にします。写真の場合はスイッチを右側に倒します。</p> <p>③リセットボタンを押します。</p> <p>このとき、マイコンボード左下にある2個のLED が次の状態になってるか確認してください。</p> <p>(POWER): 点灯</p> <p>(USB): 消灯でも点灯でもどちらでもよい</p>
---	--	--

5. 演習手順

※通信ポートが分からない場合

1		<p>デスクトップにある「マイコンコンピュータ」で右クリック、プロパティを選択します。 デスクトップにマイコンコンピュータが無い場合は、コントロールパネル(クラシック表示)のシステムを選択してください。</p>
---	---	--

2		<p>「ハードウェア」タブを選択します。 デバイス マネージャをクリックします。</p>
---	--	--

3		<p>「ポート (COM と LPT)」をクリックすると、USB Serial Port (COM●)があります。●部分が通信ポートの番号です。</p>
---	---	---

6. 実習基板 Ver.2

本書では、R8C/35A マイコンの実習を行うために、実習基板 Ver.2 を使用します。この章では、実習基板 Ver.2 について説明します。

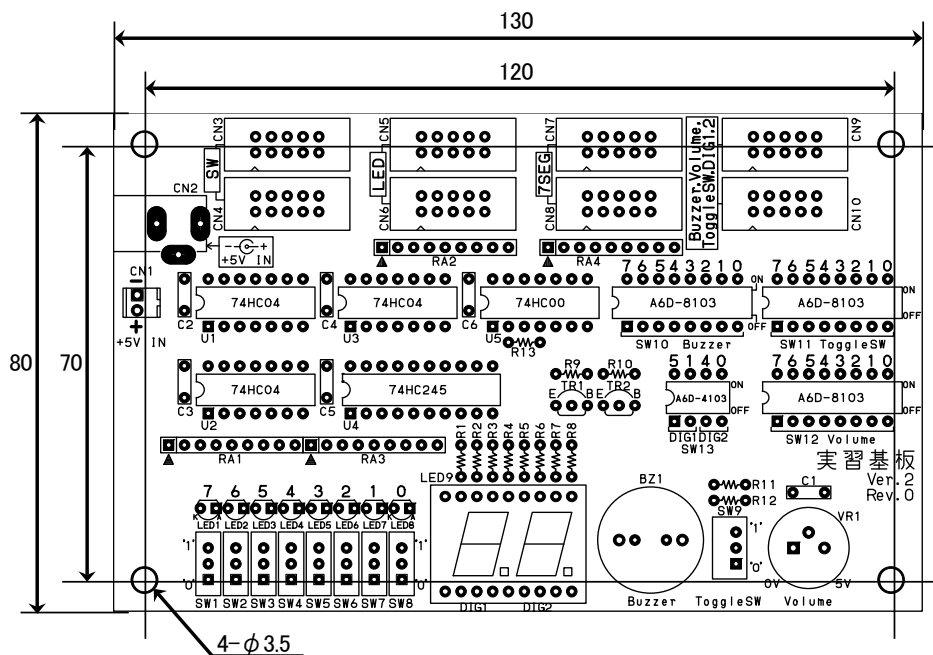
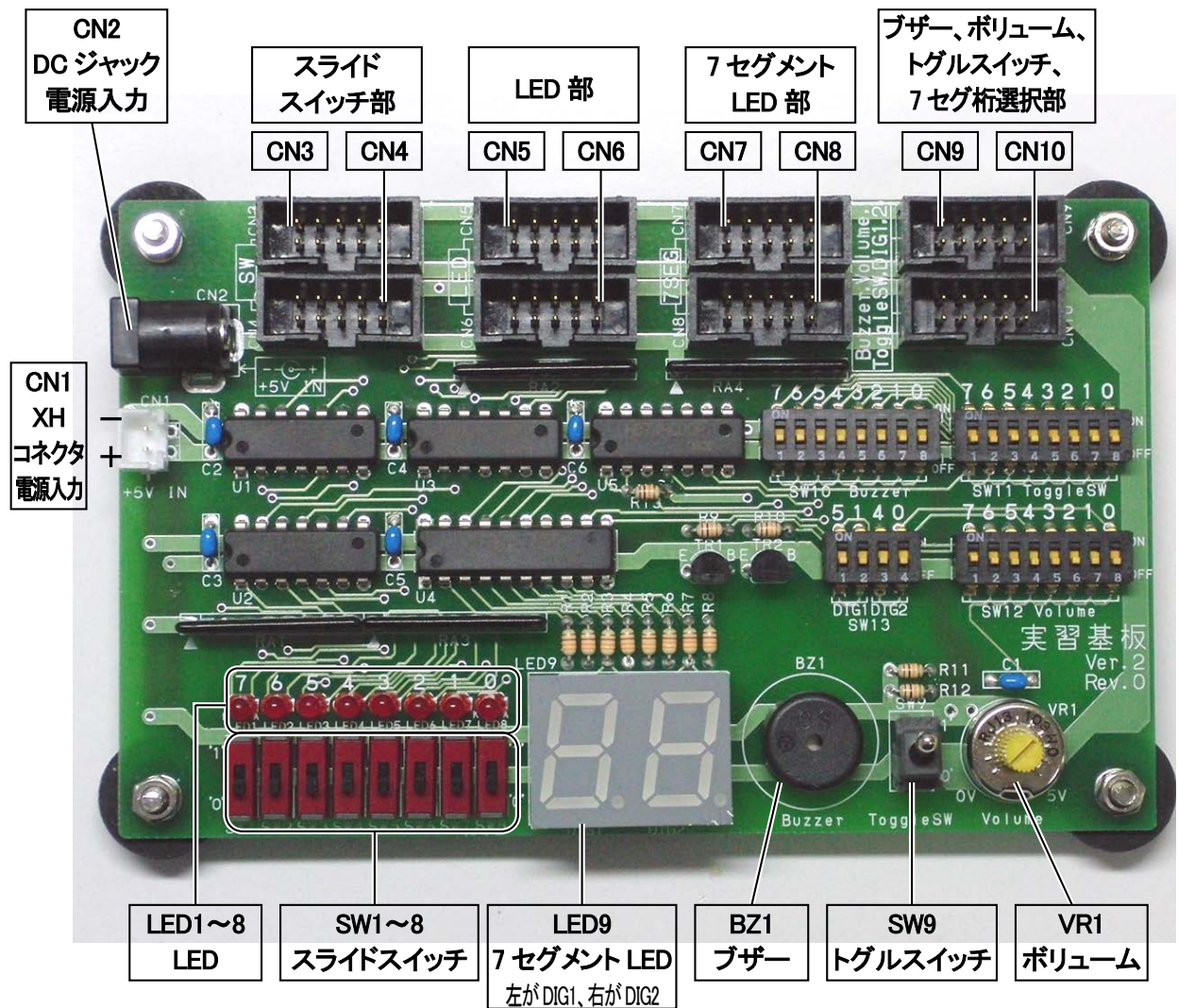
6.1 仕様

実習基板 Ver.2 の仕様を下表に示します。

内容	詳細
スライドスイッチ (SW1～8)	スライドスイッチを 8 個搭載しています。 スイッチの状態を替えることによって、CN3 と CN4 に出力する信号レベルが変わります。 スライドスイッチ上側: "1"(5V) を出力 スライドスイッチ下側: "0"(0V) を出力
LED (LED1～8)	LED を 8 個搭載しています。 CN5、または CN6 から入力する信号によって、LED が点灯／消灯します。 "0" の信号が入力された場合: LED は消灯 "1" の信号が入力された場合: LED は点灯
7 セグメント LED (LED9)	7 セグメント LED を 2 桁搭載しています。点灯方式は、ダイナミック点灯です。 CN7、または CN8 から入力する信号で表示内容、CN9、または CN10 から入力する信号で表示する桁を選びます。表示する桁を選ぶビットは、SW13 で選択します。
ブザー (BZ1)	圧電サウンダ(以下、ブザーといいます)を 1 個搭載しています。 CN9、または CN10 から入力する信号で音を鳴らします。接続するビットは、SW10 で選択します。 ブザーに加える周波数を変えることで、音階が変わります。
トグルスイッチ (SW9)	トグルスイッチを 1 個搭載しています。 トグルスイッチの状態を変えることによって CN9、または CN10 に出力する信号レベルが変わります。接続するビットは、SW11 で選択します。 トグルスイッチ上側: "1"(5V) を出力 トグルスイッチ下側: "0"(0V) を出力
ボリューム (VR1)	ボリュームを 1 個搭載しています。 ボリュームを回すことによって、CN9、または CN10 に出力する電圧が変わります。接続するビットは、SW12 で選択します。 ボリュームをいちばん反時計回りにしたときの電圧: 0V ボリュームをいちばん時計回りにしたときの電圧: 5V

6. 実習基板 Ver.2

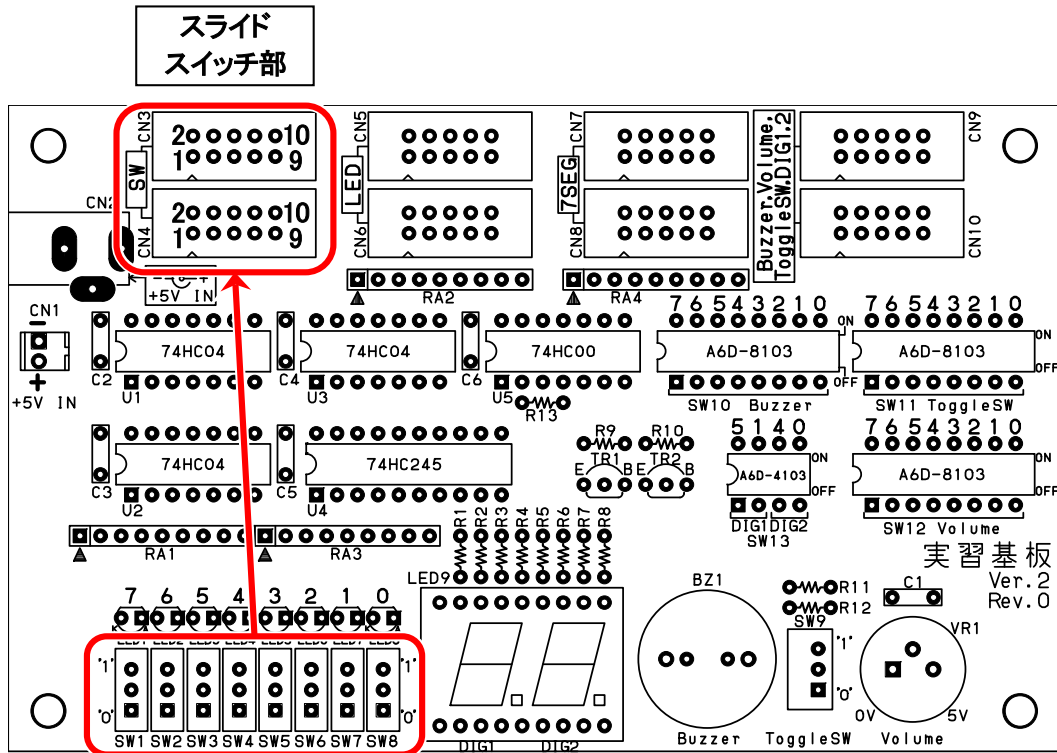
6.2 外観



6.3 スライドスイッチ部

6.3.1 コネクタ

スライドスイッチ SW1～SW8 の 8bit 分の信号を CN3(CN4)のコネクタから外部に出力します。CN3 と CN4 は並列に接続されています。

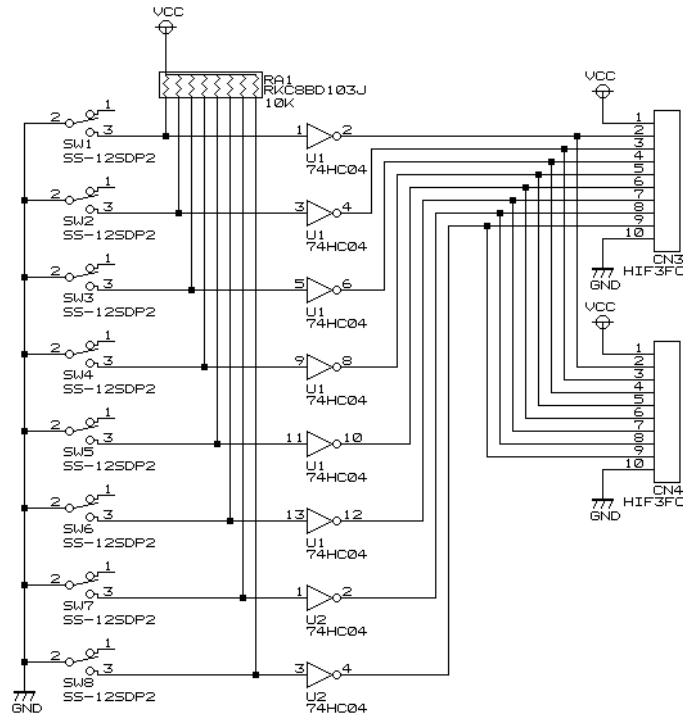


スライドスイッチと CN3(CN4)の端子番号との関係を、下表に示します。

CN3,CN4 端子番号	信号	RY_R8C38 ボード コネクタ接続時のビット	“0”が出力され るときのスイッ チの位置	“1”が出力され るときのスイッ チの位置
1	+5V		—	—
2	スライドスイッチ SW1	7	下側	上側
3	スライドスイッチ SW2	6	下側	上側
4	スライドスイッチ SW3	5	下側	上側
5	スライドスイッチ SW4	4	下側	上側
6	スライドスイッチ SW5	3	下側	上側
7	スライドスイッチ SW6	2	下側	上側
8	スライドスイッチ SW7	1	下側	上側
9	スライドスイッチ SW8	0	下側	上側
10	GND		—	—

6. 実習基板 Ver.2

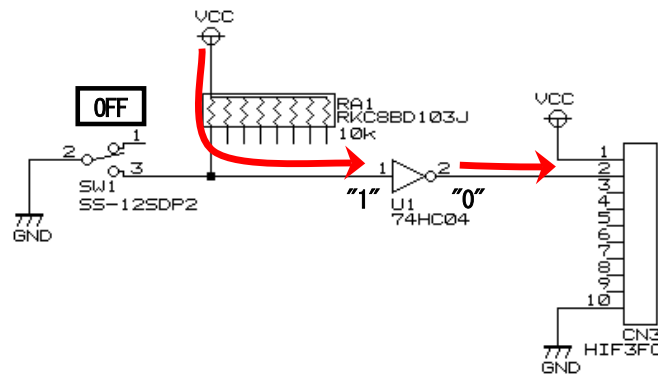
6.3.2 回路図



6.3.3 動作原理

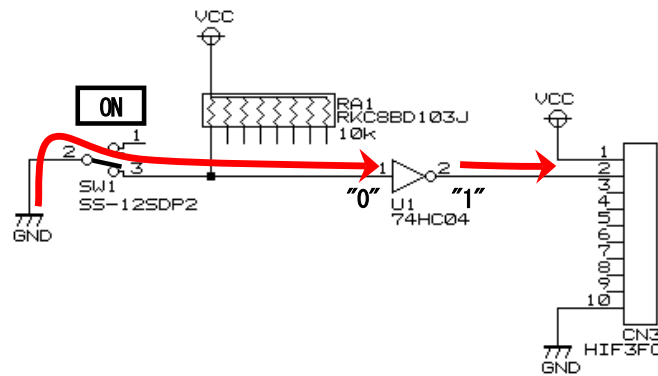
■スライドスイッチが"0"側(下側)

スイッチが下側の場合は OFF になり、プルアップ抵抗、74HC04(NOT)回路を通して、端子から"0"が出力されます。



■スライドスイッチが"1"側(上側)

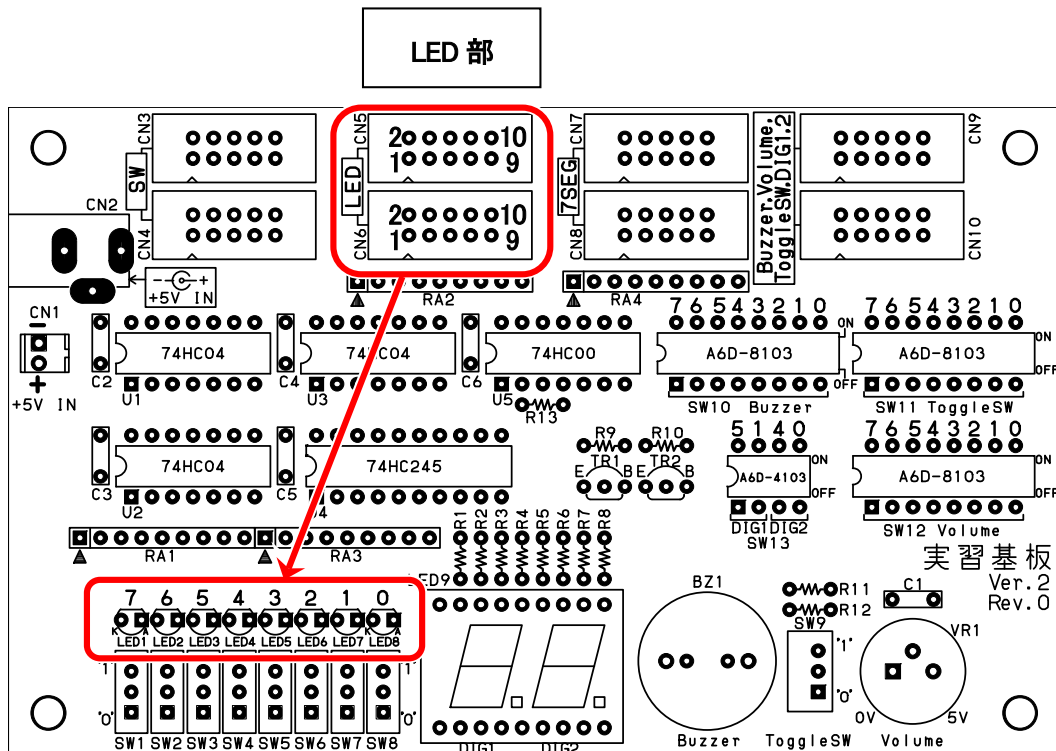
スイッチが上側の場合は ON になり、スライドスイッチ、74HC04(NOT)回路を通して、端子から"1"が出力されます。



6.4 LED 部

6.4.1 コネクタ

CN5(CN6)のコネクタから入力した信号を、LED1～LED8 へ出力します。CN5 と CN6 は並列に接続されています。

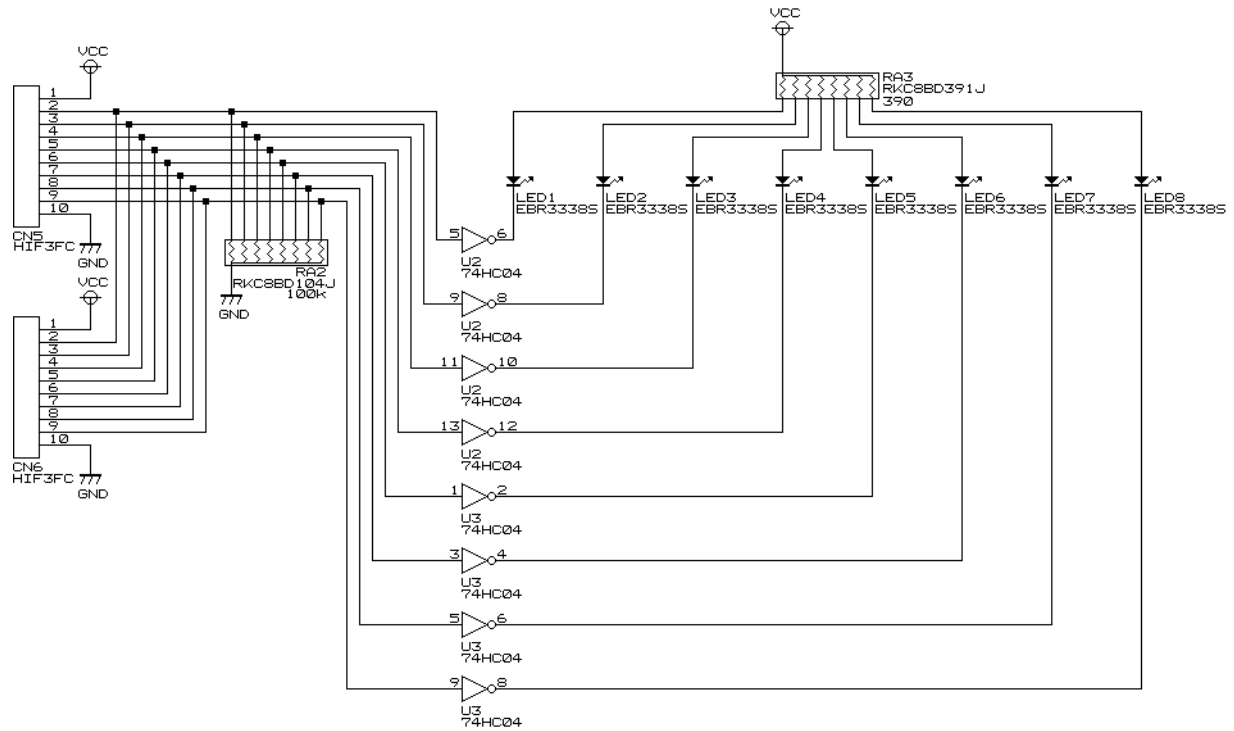


LED と CN5(CN6)の端子番号との関係を、下表に示します。

CN5,CN6 端子番号	信号	RY_R8C38 ボード コネクタ接続時のビット	“0”が入力され たときの LED	“1”が入力され たときの LED
1	+5V		—	—
2	LED1	7	消灯	点灯
3	LED2	6	消灯	点灯
4	LED3	5	消灯	点灯
5	LED4	4	消灯	点灯
6	LED5	3	消灯	点灯
7	LED6	2	消灯	点灯
8	LED7	1	消灯	点灯
9	LED8	0	消灯	点灯
10	GND		—	—

6. 実習基板 Ver.2

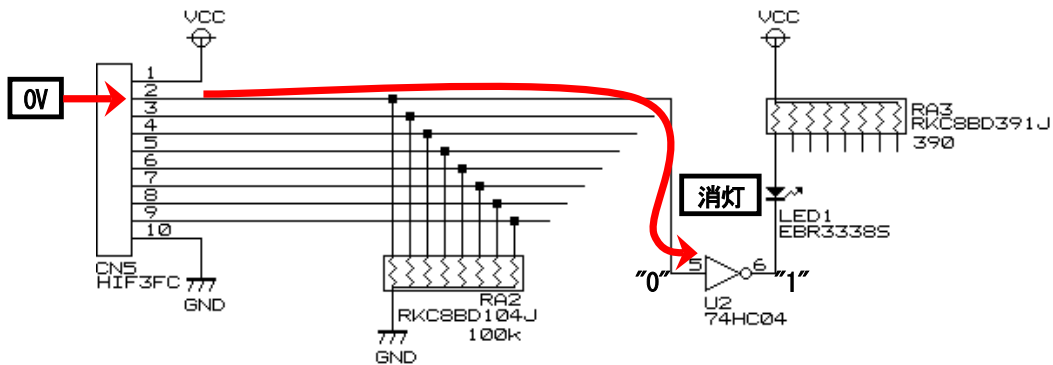
6.4.2 回路図



6.4.3 動作原理

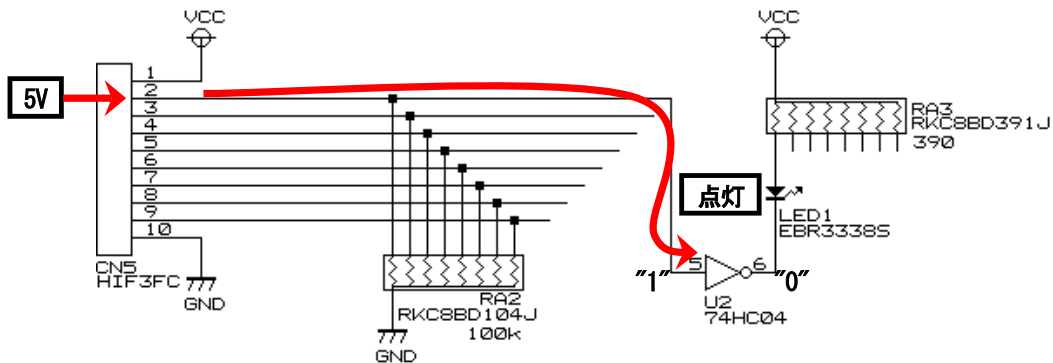
■入力信号が 0V("0")の場合

入力信号が 0V("0")の場合、74HC04(NOT)回路を介して LED のカソード側が 5V("1")になり LED は消灯します。



■入力信号が 5V("1")の場合

入力信号が 5V("1")の場合、74HC04(NOT)回路を介して LED のカソード側が 0V("0")になり LED は点灯します。

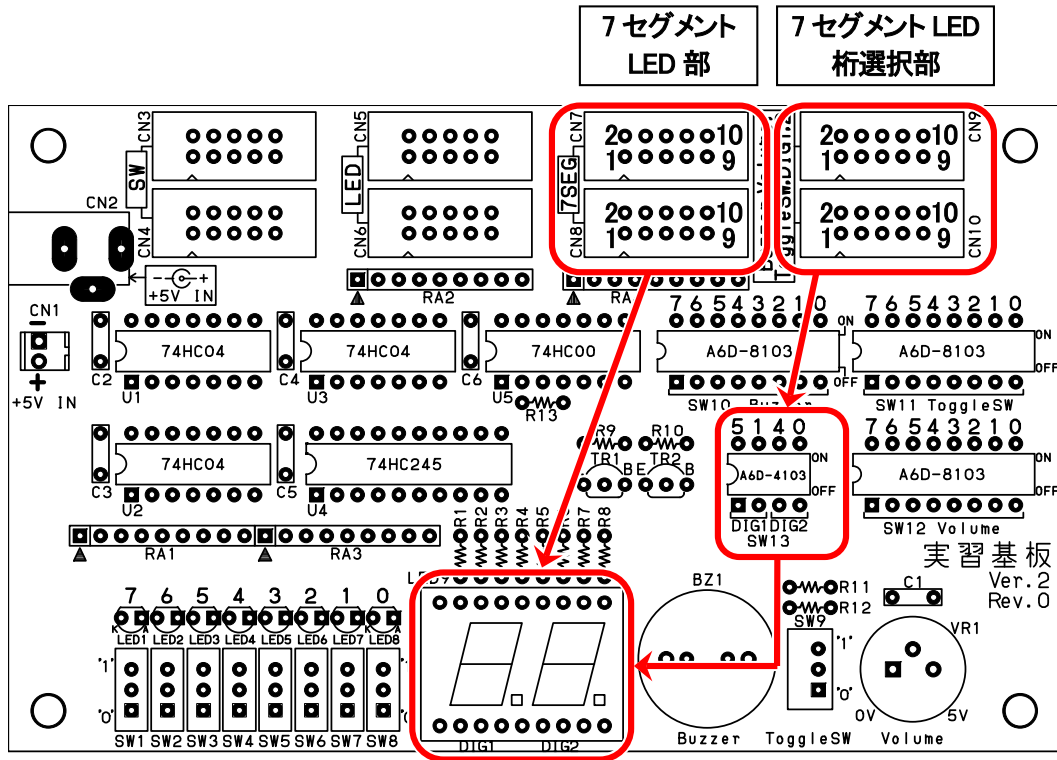


6.5 7セグメント LED 部

6.5.1 コネクタ

CN9(CN10)のコネクタから入力した信号で、表示する桁を選択します。CN9 と CN10 は並列に接続されています。

CN7(CN8)のコネクタから入力した信号を、7セグメントLEDへ出力します。表示する桁は、CN9で選択している桁になります。CN7 と CN8 は並列に接続されています。



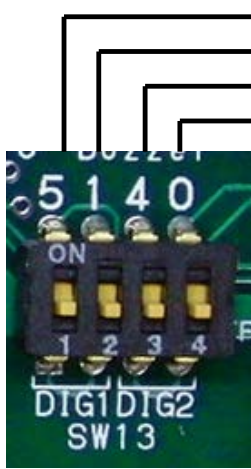
7セグメントLEDとCN7(CN8)の端子番号との関係を、下表に示します。

CN7,CN8 端子番号	信号	RY_R8C38 ボード コネクタ接続時のビット	“0”が入力され たときの7セグ メントLED	“1”が入力され たときの7セグ メントLED
1	+5V		—	—
2	7セグメントLED DP	7	消灯	点灯
3	7セグメントLED G	6	消灯	点灯
4	7セグメントLED F	5	消灯	点灯
5	7セグメントLED E	4	消灯	点灯
6	7セグメントLED D	3	消灯	点灯
7	7セグメントLED C	2	消灯	点灯
8	7セグメントLED B	1	消灯	点灯
9	7セグメントLED A	0	消灯	点灯
10	GND		—	—

※ただし、桁選択 ON の桁のみ表示されます。

6. 実習基板 Ver.2

左桁を点灯させるか、右桁を点灯させるかは、CN9(CN10)からの入力信号で決めます。CN9(CN10)と、左桁、右桁を接続する bit を、SW13 で決めます。CN9(CN10)と SW13 の関係を下図に示します。

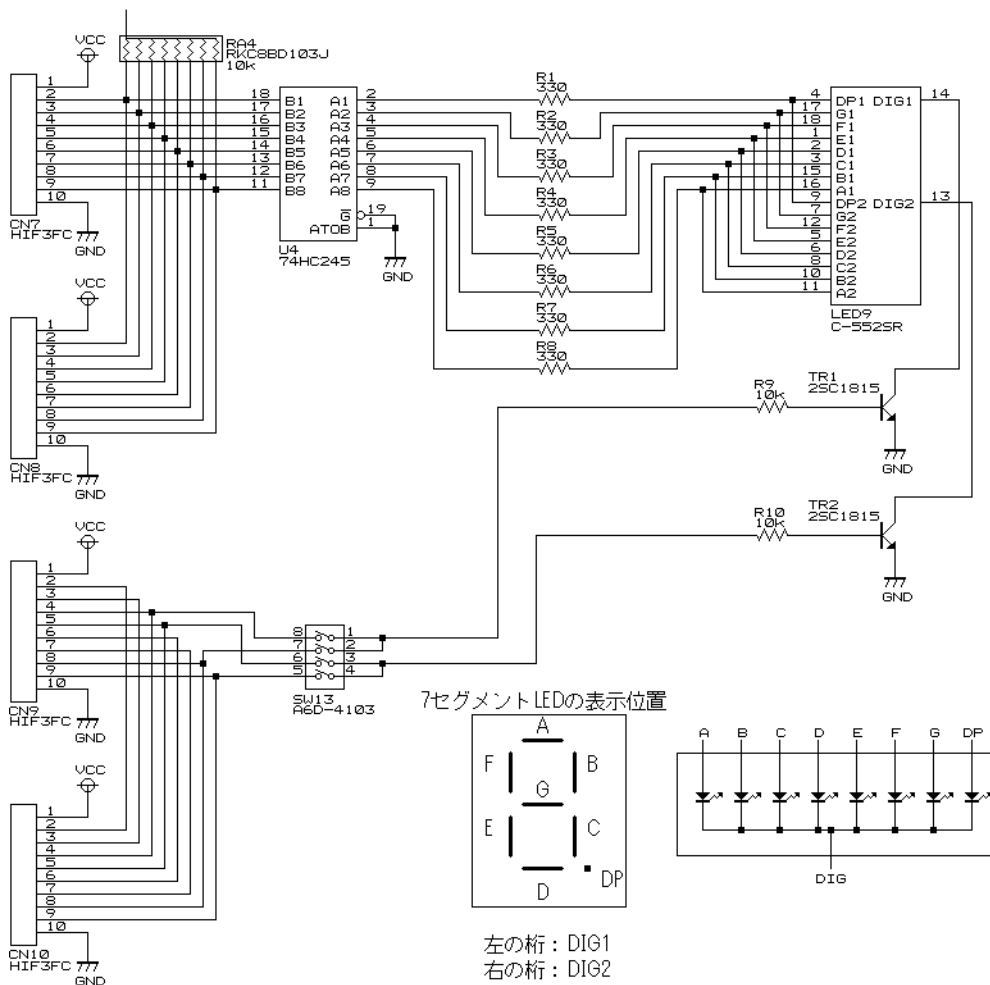


- ON すると DIG1 端子(左の桁)と CN9 の bit5 が接続されます。
- ON すると DIG1 端子(左の桁)と CN9 の bit1 が接続されます。
- ON すると DIG2 端子(右の桁)と CN9 の bit4 が接続されます。
- ON すると DIG2 端子(右の桁)と CN9 の bit0 が接続されます。

- ※5 と 1 は両方 OFF か、どちらか片方だけ ON にしてください。
両方 ON にすると、CN9 の bit5 と bit1 が接続状態となり
それぞれの信号出力レベルによっては、ショート状態となります。
- ※4 と 0 は両方 OFF か、どちらか片方だけ ON にしてください。
両方 ON にすると、CN9 の bit4 と bit0 が接続状態となり
それぞれの信号出力レベルによっては、ショート状態となります。

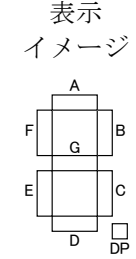
















DIG1 端子、または DIG2 端子は“0”で表示選択 OFF、“1”で表示選択 ON になります。

6.5.2 回路図



6. 実習基板 Ver.2

7 セグメント LED に表示する値と、7 セグメント LED に送るデータを下表に示します。

表示する値	DP	G	F	E	D	C	B	A	表示イメージ 	16進数
0	0	0	1	1	1	1	1	1		0x3f
1	0	0	0	0	0	1	1	0		0x06
2	0	1	0	1	1	0	1	1		0x5b
3	0	1	0	0	1	1	1	1		0x4f
4	0	1	1	0	0	1	1	0		0x66
5	0	1	1	0	1	1	0	1		0x6d
6	0	1	1	1	1	1	0	1		0x7d
7	0	0	0	0	0	1	1	1		0x07
8	0	1	1	1	1	1	1	1		0x7f
9	0	1	1	0	1	1	1	1		0x6f
a	0	1	1	1	0	1	1	1		0x77
b	0	1	1	1	1	1	0	0		0x7c
c	0	1	0	1	1	0	0	0		0x58
d	0	1	0	1	1	1	1	0		0x5e
e	0	1	1	1	1	0	0	1		0x79
f	0	1	1	1	0	0	0	1		0x71

上表の16進数の値を、CN7(CN8)に接続されているポートから出力すると、7セグメントLEDに値が表示されます。

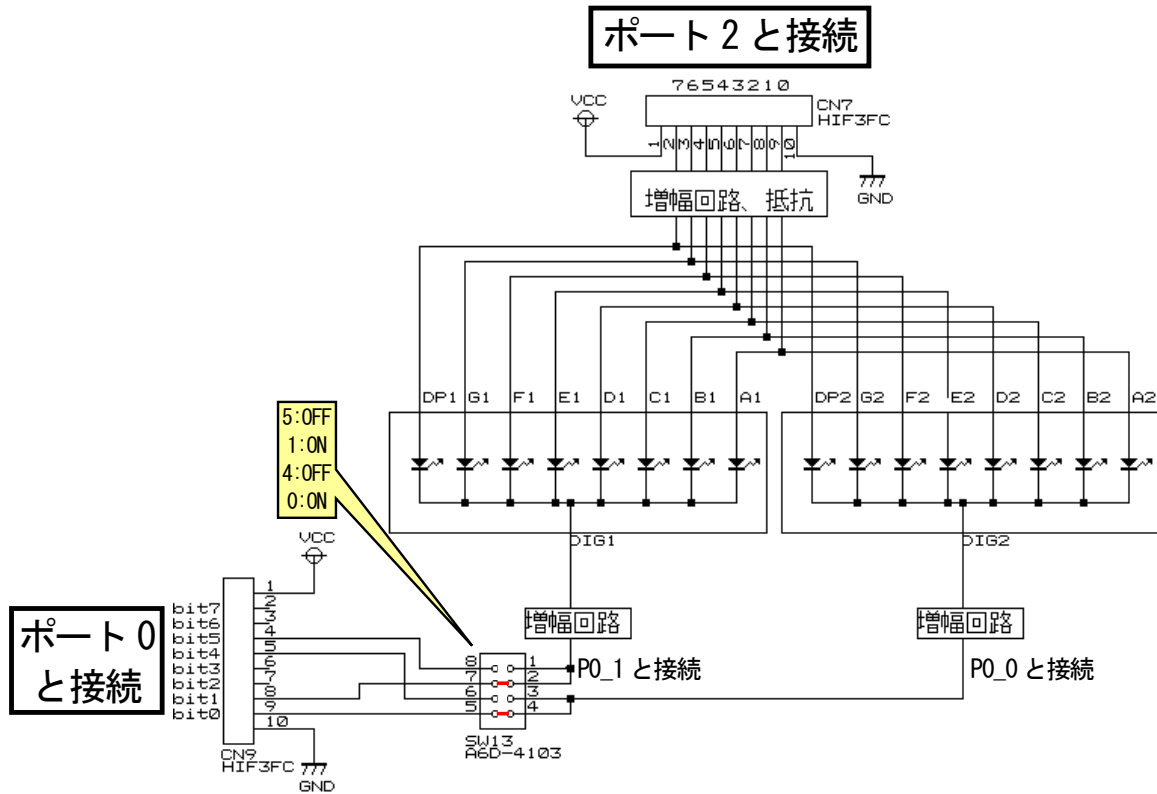
ただし、7セグメントLEDの左桁と右桁のどちらを表示させるかで、CN9(CN10)側からもDIG1端子、DIG2端子を制御する必要があります。次で説明します。

6. 実習基板 Ver.2

6.5.3 動作原理

下記の状態として、説明します。

- CN7 はポート 2 と接続
- CN9 はポート 0 と接続
- SW13 は左から、5:OFF、1:ON、4:OFF、0:ON
この設定で、P0_1 と DIG1 端子(左の桁)、P0_0 と DIG2 端子(右の桁)が接続されます。
- 左の桁には「8」、右の桁には「1」を表示



表示する手順を下記に示します。

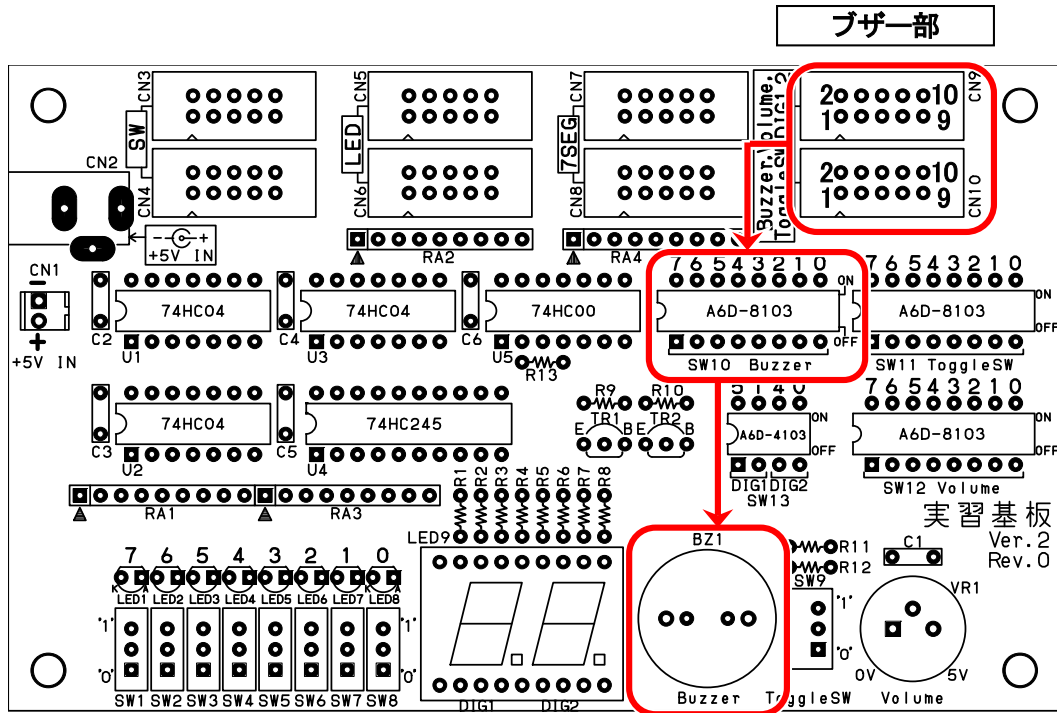
①	左桁、右桁は表示させません。 ポート 0 の bit1 と bit0 を「0」にします。
②	左桁に表示したい値を設定します。今回は「8」です。 ポート 2 に 0x7f を設定します。
③	左桁を表示します。 DIG1 端子に接続されているポート 0 の bit1 を「1」にします。A1～DP1 の LED のカソード側 DIG1 が GND につながり、アノード側が 5V の LED は点灯します。DIG2 は開放状態なので、A2～DP2 は点灯しません。
④	少し待ちます。
⑤	左桁、右桁は表示させません。 ポート 0 の bit1 と bit0 を「0」にします。
⑥	右桁に表示したい値を設定します。今回は「1」です。 ポート 2 に 0x06 を設定します。
⑦	右桁を表示します。 DIG2 端子に接続されているポート 0 の bit0 を「1」にします。A2～DP2 の LED のカソード側 DIG2 が GND につながり、アノード側が 5V の LED は点灯します。DIG1 は開放状態なので、A1～DP1 は点灯しません。
⑧	少し待ちます。

表示は、左桁、右桁を 1 桁ずつ交互に点灯させていますが、①～⑧を高速で繰り返すと人の目には両方点灯しているように見えます。少し待つ部分を本書の演習では、1ms にしています。

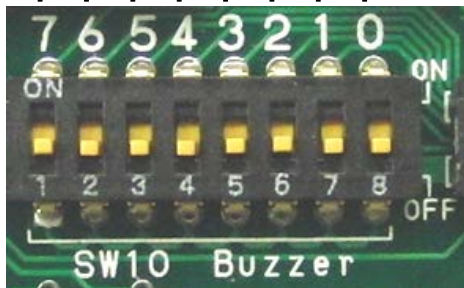
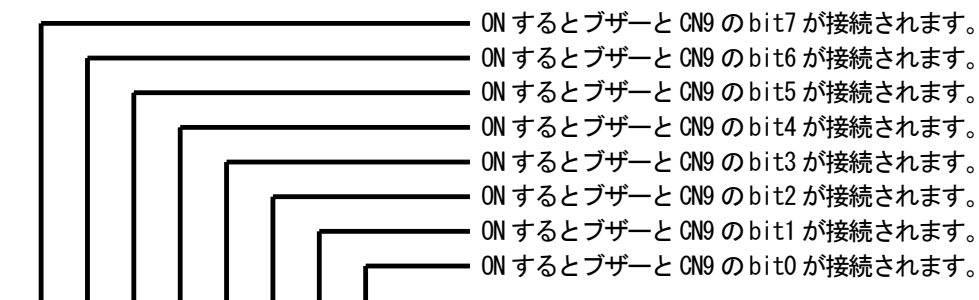
6.6 ブザー部

6.6.1 コネクタ

CN9(CN10)のコネクタから入力された信号でブザーを鳴らします。



CN9(CN10)とブザーを、どの bit に接続するかを SW10 で決めます。CN9(CN10)と SW10 の関係を下図に示します。

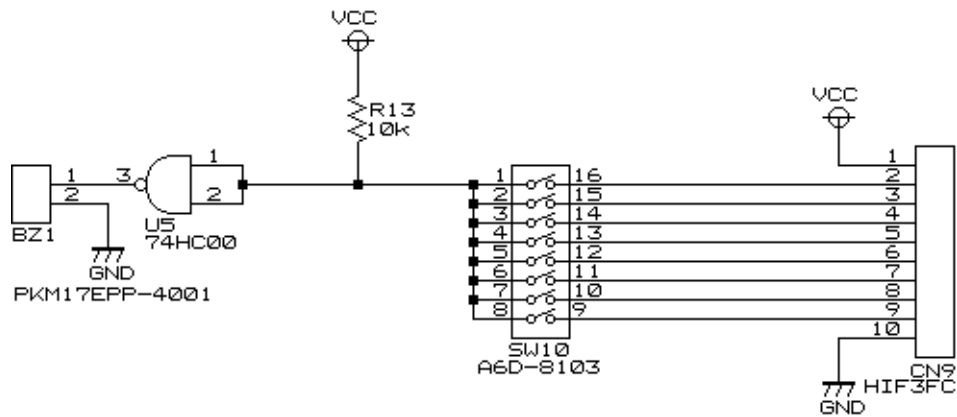


※SW10 は1つだけ ON にしてください。

複数 ON にすると、CN9 の ON にした端子同士が接続状態となりポートの入出力設定によっては、ショート状態となります。

6. 実習基板 Ver.2

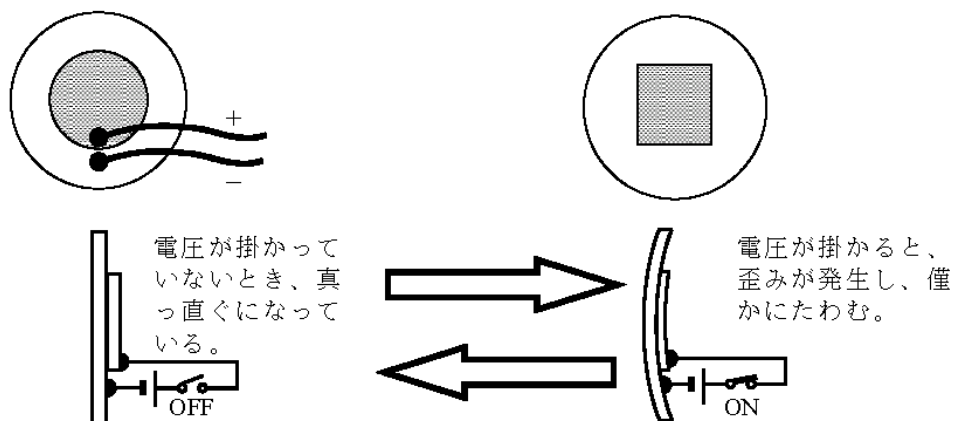
6.6.2 回路図



6.6.3 動作原理

ブザーは、下図のように 2 枚の電極を貼り合わせたような形をしています。2 枚の電極は絶縁されていますが、既定値以上の電圧を加えると壊れてしまいます。外側は薄い鉄板になっていて、内側には特殊な電極が張り付いています。両方とも半田付けができるようになっています。

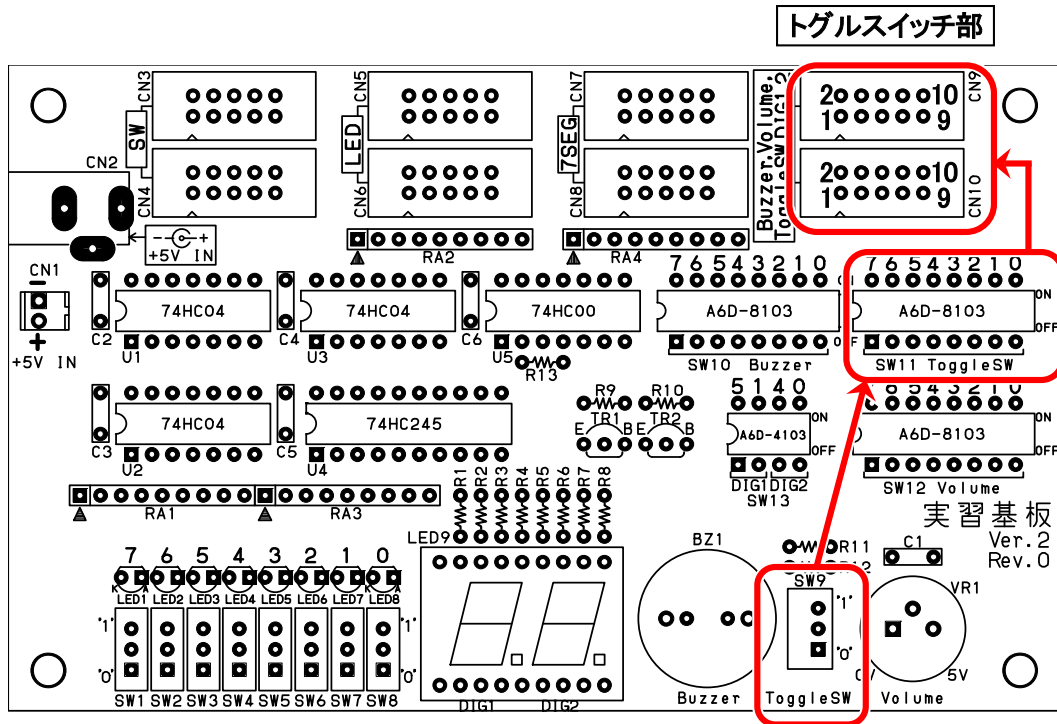
この 2 枚の電極に電圧をかけると、歪みが発生して僅かにたわみます。電圧がかかっていなければ、まっすぐの状態です。これを高速に繰り返すことにより、音波が発生して音が鳴ります。



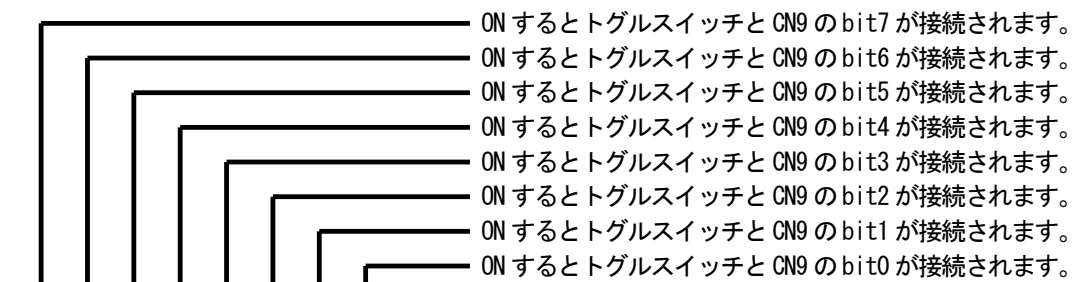
6.7 トグルスイッチ部

6.7.1 コネクタ

トグルスイッチの"0"か"1"の状態を CN9(CN10)のコネクタから外部に出力します。



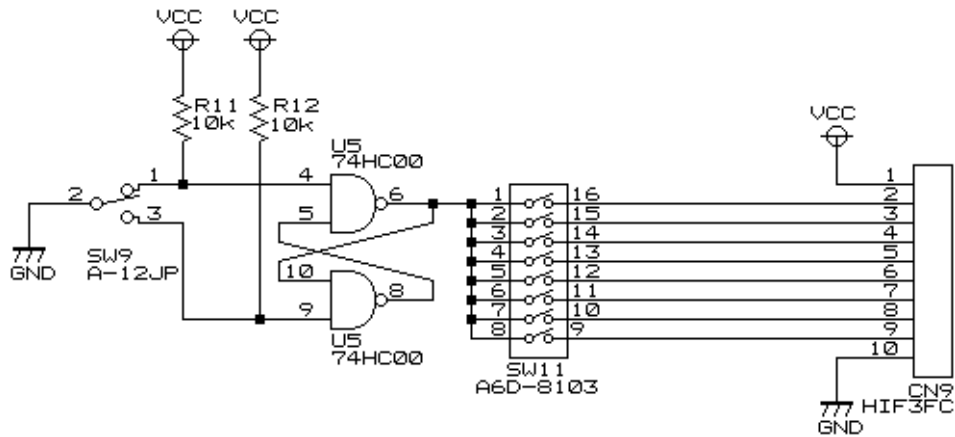
CN9(CN10)とトグルスイッチを、どの bit に接続するかを SW11 で決めます。CN9(CN10)と SW11 の関係を下図に示します。



※SW11 は1つだけ ON にしてください。

複数 ON にすると、CN9 の ON にした端子同士が接続状態となりポートの入出力設定によっては、ショート状態となります。

6.7.2 回路図

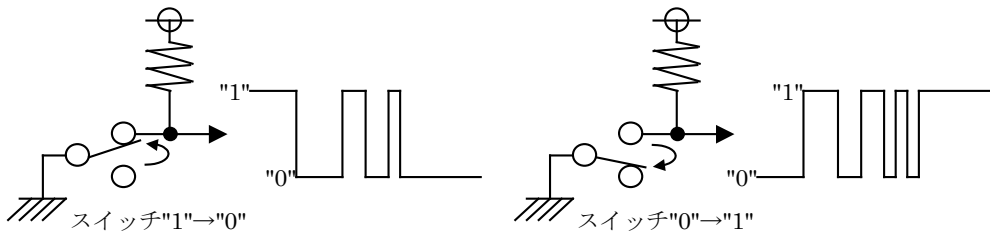


6.7.3 動作原理

下図のように、通常のプルアップしたスイッチを“1”から“0”、もしくは“0”から“1”に変更した場合、何度か接点が付いたり離れたりして、最終的に“1”や“0”になります。これを「チャタリング」といいます。

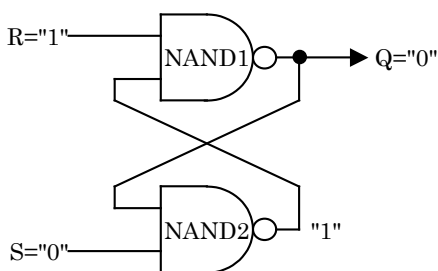
チャタリングの時間は、数ミリ秒の出来事で人間にはあまり関係ありませんが、高速で動くマイコンの場合、それぞれの状態を検出して、1回しか変化させていないつもりでも何度も“0”、“1”を繰り返したと判断してしまいます。“1”か“0”か判断するだけならあまり問題にならないことが多いのですが、パルスの回数を数えるプログラムの場合、すべてカウントしてしまい、誤動作となってしまいます。

下図の“1”→“0”の例では、一度しか“1”→“0”にしていないつもりでも 3 回もスイッチを上げ下げしたと判断されてしまいます。更にやっかいなことは、発生するパルス数や収束するまでの時間が、毎回違うということです。



チャタリングを解消する回路の一つに「リセット・セット・フリップフロップ (RS-FF)」という回路があります。NAND 回路をループさせた形の回路です。R(Reset)端子に“1”が入力されると出力 Q は“0”になり、S(Set)端子に“1”が入力されると出力 Q は“1”になることからそう呼ばれています。動作原理を下記に示します。

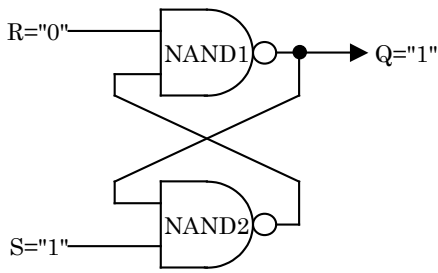
(1)入力(R,S)=(“1”,”0”)のとき



- NAND2 は S="0"のため、もう一方の入力が何であろうと無条件で出力"1"
 - NAND1 は R="1"、もう一方の入力は NAND2 の出力"1"なので出力"0"
 - NAND2 の S 側でない入力が"0"で確定するが S="0"のため、出力は変わらず"1"
- 結果、リセット信号だけが"1"のため、出力 Q="0"となったと考えることができます。

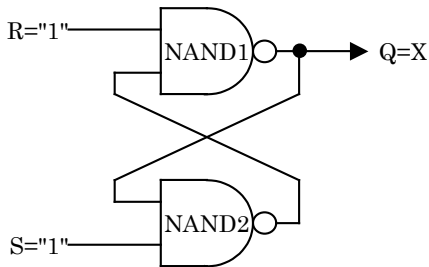
6. 実習基板 Ver.2

(2)入力(R,S)=(“0”,“1”)のとき



- NAND1 は R="0"のため、もう一方の入力が何であろうと無条件で出力"1"
 - NAND2 は S="1"、もう一方の入力は NAND1 の出力"1"なので出力"0"
 - NAND1 の R 側でない入力が"0"で確定するが R="0"のため、出力は変わらず"1"
- 結果、セット信号だけが"1"のため、出力 Q="1"となったと考えることができます。

(3)入力(R,S)=(“1”,“1”)のとき



- NAND1、NAND2、共に入力が決まらなければ出力は決定されないため、NAND1 の出力 Q=X と仮定する
- NAND2 の入力は S="1"と X なので、出力は \overline{X} となる
- NAND1 の入力は R="1"と \overline{X} なので

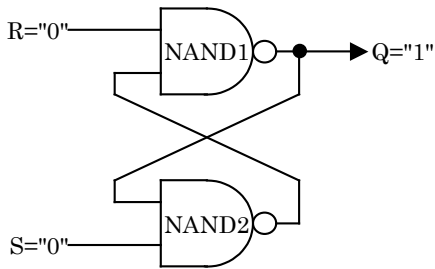
$$Q = \overline{1 \cdot \overline{X}} = \overline{\overline{X}} = X$$

となり、最初に仮定した X と同じ値になる

これは、

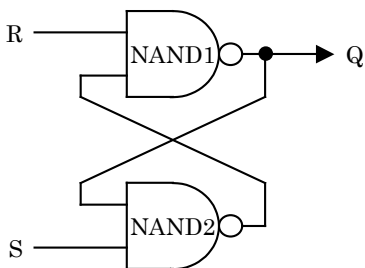
- Q="0"のとき、(R,S)=(“1”,“1”)としても前の出力を保持
- Q="1"のとき、(R,S)=(“1”,“1”)としても前の出力を保持するということです。

(4)入力(R,S)=(“0”,“0”)のとき



- NAND1 は R="0"、もう一方の入力が何であろうと無条件で"1"
 - NAND2 は S="0"、もう一方の入力が何であろうと無条件で"1"
- 結果、Q="1"となります。
- ただし、(R,S)=(“0”,“0”)はリセットともセットとも呼べない状態です。Q="1"となりセット状態ではありますが、リセット、セット、どちらかに"1"信号があるものとする論理に反する入力状態のため、通常は「禁止」とされています。ショートなどして回路が壊れるために禁止とするのではなく、論理が合わなくなるため禁止としています。

(5)まとめ

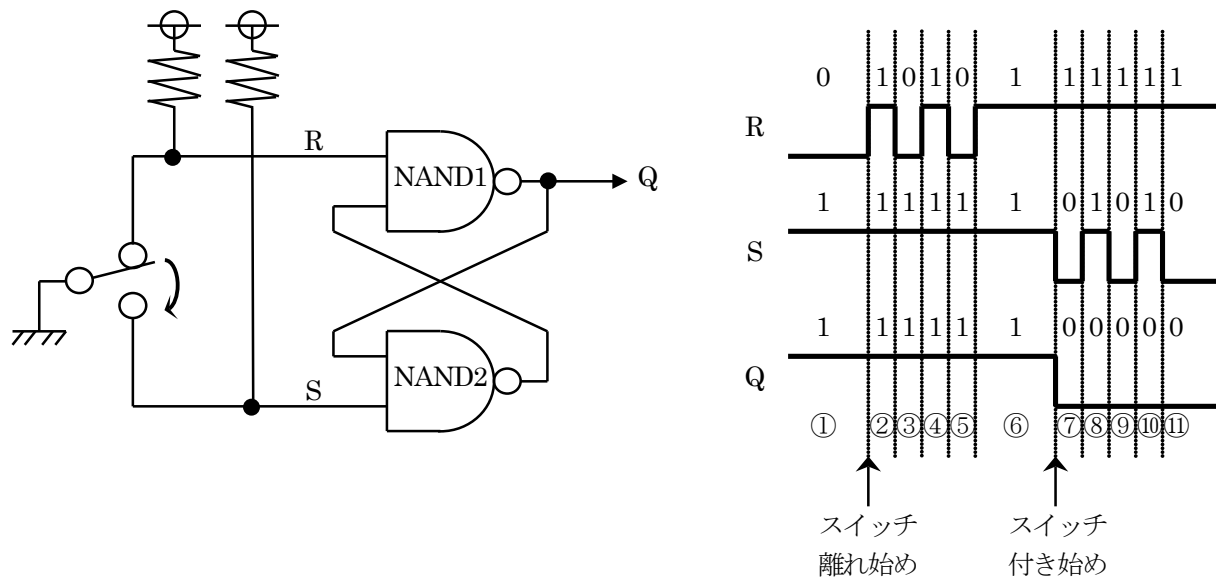


まとめると下表のようになります。

入力 R	入力 S	出力 Q
0	1	1(セット状態)
1	0	0(リセット状態)
1	1	前出力保持
0	0	禁止

6. 実習基板 Ver.2

実際に RS-FF をチャタリング防止回路に使った動作を説明します。

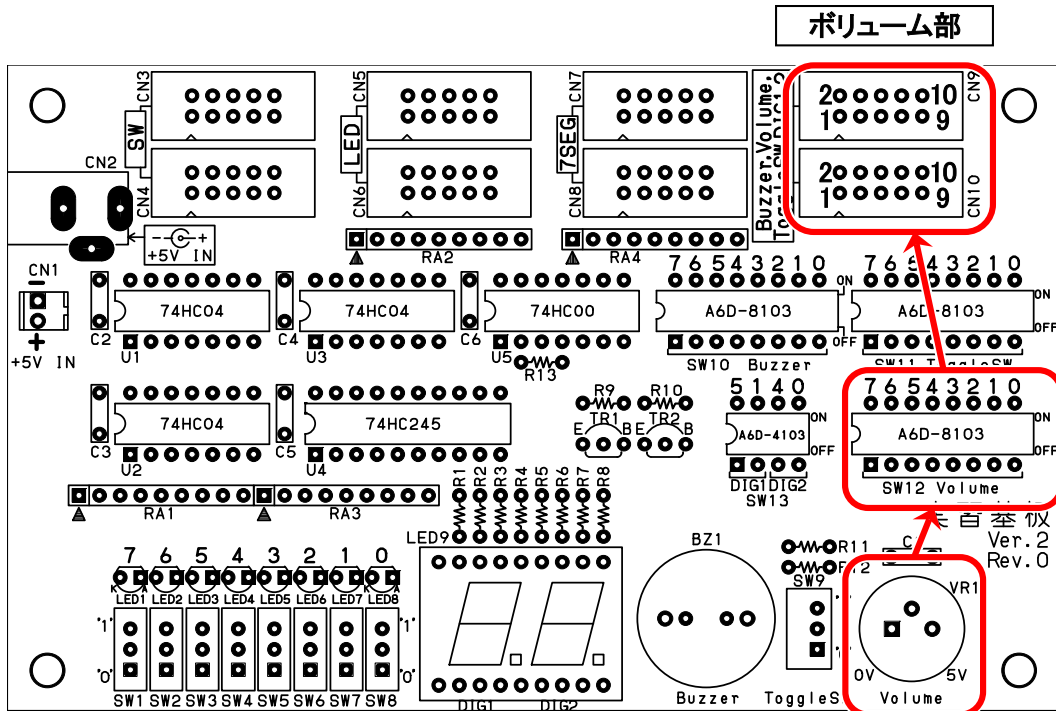


①	最初、(R,S)=(“0”,“1”)のため、Q=“1”となります。
②	スイッチを下側に切り替え、R 側から接点が離れ始めた時です。(R,S)=(“1”,“1”)のため、Q は前の値を保持します。
③	(R,S)=(“0”,“1”)のため、Q=“1”となります。
④	(R,S)=(“1”,“1”)のため、Q は前の値を保持します。
⑤	(R,S)=(“0”,“1”)のため、Q=“1”となります。
⑥	(R,S)=(“1”,“1”)のため、Q は前の値を保持します。接点が中間になりました。R 端子、S 端子共にスイッチの接点には繋がっていない状態ですが、プルアップ抵抗があるので“1”になっています。
⑦	スイッチの接点が S 側の端子に付き始めた状態です。(R,S)=(“1”,“0”)のため、Q=“0”となります。
⑧	(R,S)=(“1”,“1”)のため、Q は前の値を保持します。
⑨	(R,S)=(“1”,“0”)のため、Q=“0”となります。
⑩	(R,S)=(“1”,“1”)のため、Q は前の値を保持します。
⑪	(R,S)=(“1”,“0”)のため、Q=“0”となります。チャタリングが収まりました。この状態が次にスイッチを切り替えるまで続きます。

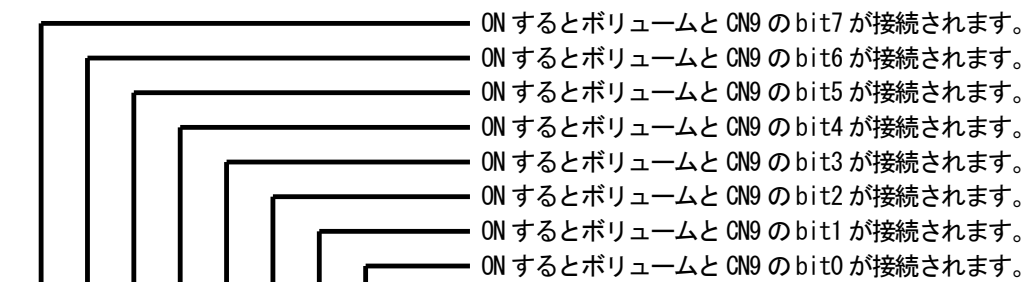
6.8 ボリューム部

6.8.1 コネクタ

ボリューム(VR1)の電圧を CN9(CN10)のコネクタから外部に出力します。



CN9(CN10)とボリュームを、どの bit に接続するかを SW12 で決めます。CN9(CN10)と SW12 の関係を下図に示します。

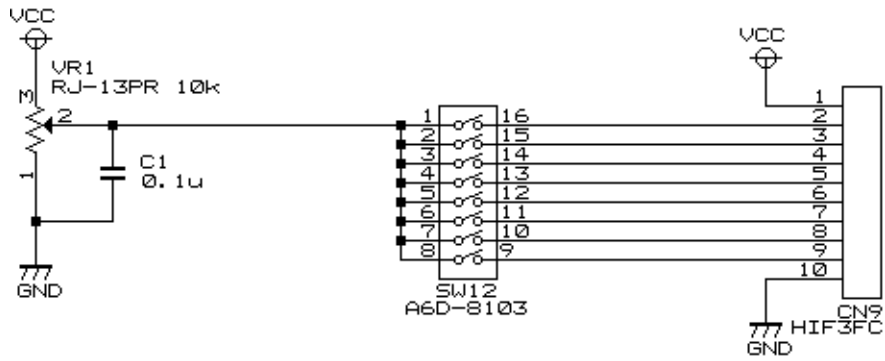


※SW12 は1つだけ ON にしてください。

複数 ON にすると、CN9 の ON にした端子同士が接続状態となりポートの入出力設定によっては、ショート状態となります。

6. 実習基板 Ver.2

6.8.2 回路図

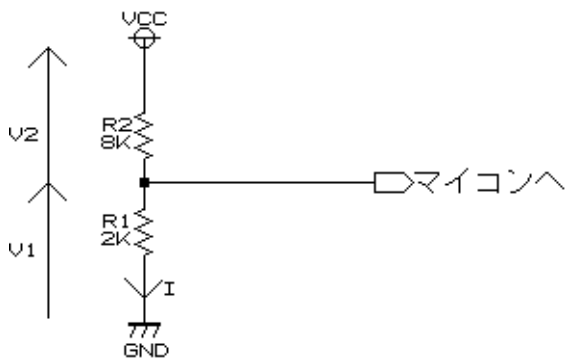


6.8.3 動作原理

10kΩの抵抗の両端(1ピン、3ピン)がGNDと+5Vに接続されており、中心端子(2ピン)の抵抗値がつまみに合わせて変わります。2ピンの電圧が変わる様子を下記に示します。

つまみがいちばん左のとき	つまみがいちばん右のとき	つまみが 2/10 のとき
2ピンは、1ピンのGNDに繋がるので0Vが出力されます。	2ピンは、3ピンの+5Vに繋がるので5Vが出力されます。	2ピンは、2kΩと8kΩで分圧された1.0Vが出力されます。

つまみが 2/10 のときの計算を下記に示します。

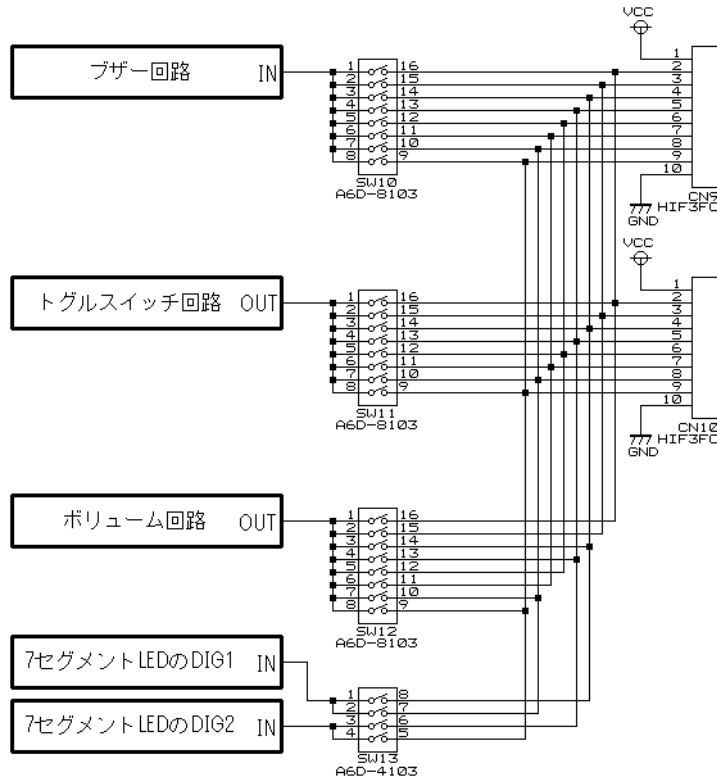


$$\begin{aligned}
 I &= V_{cc} / (R1 + R2) \\
 &= 5 / (2K + 8K) \\
 &= 5 / 10K \\
 &= 0.0005A \\
 &= 0.5mA \\
 V1 &= I \cdot R1 \\
 &= 0.0005 \times 2K \\
 &= 1.0V
 \end{aligned}$$

このように、ボリュームのつまみの位置によって、電圧が出力されます。

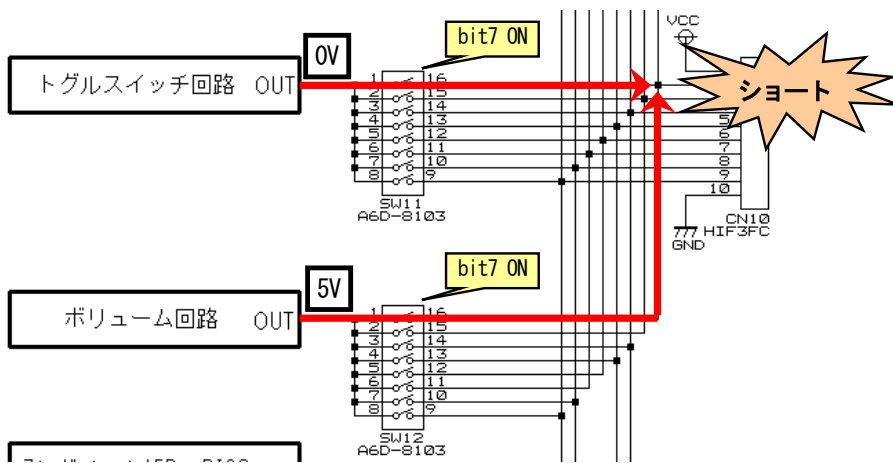
6.9 SW10～SW13 を切り替えるときの注意点

CN9(CN10)は、ブザー、ボリューム、トグルスイッチ、7セグメントLED桁選択のそれぞれの信号を共用しています。それぞれの信号は、CN9(CN10)に接続したポートの bit を変更できるよう SW10～SW13 で切り替えることができます。CN9(CN10)に接続されている回路を下図に示します。



例えば、下記の状態とします。

- ボリュームから 5V が出力されていて、SW12 の bit7 が ON の状態
 - トグルスイッチから 0V("0")が出力されていて、SW11 の bit7 が ON の状態
- この場合、出力同士が接続し、ショート状態になります。その様子を下図に示します。



そのため、SW10、SW11、SW12、SW13 の同じ番号(bit)を同時に ON にしないでください。SW10、SW11、SW12、SW13 を切り替えるときは、すべて OFF にしてから、該当の bit を ON にしてください。

7. プロジェクトの内容

本実習マニュアルで使用するワークスペース「r8c35a_ensyu」に登録されているプロジェクトを、下表に示します。

	プロジェクト名	使用するマイコンの内蔵周辺機能	基板分離せず実習可能？	内容
1	ad	A/D コンバータ (単発モード)	○	マイコン内蔵の A/D コンバータ(単発モード)を使用して、0~5V の電圧を 0~1023 の値に変換します。
2	ad_kurikaeshi	A/D コンバータ (繰り返しモード 0)	○	マイコン内蔵の A/D コンバータ(繰り返しモード 0)を使用して、0~5V の電圧を 0~1023 の値に変換します。
3	ad_kurikaeshi_souin	A/D コンバータ (繰り返し掃引モード)	○	マイコン内蔵の A/D コンバータ(繰り返し掃引モード)を使用して、0~5V の電圧を 0~1023 の値に変換します。
4	data_flash	データフラッシュ	○	マイコン内蔵のデータフラッシュを使用して、データを保存したりデータを読み出します。
5	i2c_eeprom	I ² C	○※	外付けの 24C256(32KB の EEPROM)を I ² C で使用して、データを保存したりデータを読み出します。 ※外付けの EEPROM 基板が必要です。
6	int_interrupt	INT3 割り込み	×	外部からのエッジ信号により割り込みを発生させ、割り込みプログラムを実行します。
7	io		×	ポートの入出力を行います。制御の基本中の基本です。
8	io2		○※	マイコンボード上のディップスイッチと LED を使ってポートの入出力を行います。ビット操作をするため、プロジェクト「io」より若干複雑になります。 ※オプションの LED、抵抗各 4 個の取り付けが必要です。
9	microsw		○※	センサ部のマイクロスイッチの状態を、マイコンボード上の LED へ出力します。 ※オプションのマイクロスイッチの取り付けが必要です。
10	motor	タイマ RB、タイマ RD	○	タイマ RD のリセット同期 PWM モードを使用して、左モータ、右モータを制御します。
11	pushsw		○	マイコンボード上のプッシュスイッチ(SW3)の状態を、マイコンボード上の LED へ出力します。
12	sensor_led	タイマ RB	○	マイコンボード上のディップスイッチの状態を、センサ部の LED 4 個へ出力します。

7. プロジェクトの内容

13	servo	タイマ RB、タイマ RD	○※	タイマ RD のリセット同期 PWM モードを使用して、サーボを制御します。 ※オプションのサーボコネクタ、サーボの取り付けが必要です。
14	timer1		○	ソフトウェアタイマを使用して、LED を点滅させます。
15	timer2	タイマ RB	○	タイマ RB を使用して、1ms ごとに割り込みを発生させます。この割り込み実行回数で時間を計り、LED を点滅させます。
16	timer_ra_counter	タイマ RA	×	タイマ RA を使用して、外部から入力されたパルス数を数えます。
17	timer_ra_pulsein	タイマ RA	×	タイマ RA を使用して、外部から入力されたパルスの幅の時間を測定します。
18	timer_rc_pwm	タイマ RC	×	タイマ RC を使用して、PWM 波形を出力します。今回は、ブザーに PWM 波形を出力して音を鳴らします。
19	timer_rd_doukipwm	タイマ RD	×	タイマ RD のリセット同期 PWM モードを使用して、PWM 波形を出力します。
20	timer_rd_pwm6	タイマ RD	×	タイマ RD を使用して、PWM 波形 6 本を出力します。
21	uart0	UART0	○	UART0 を使用して、パソコンの RS232C 経由で通信します。
22	uart0_printf	UART0	○	UART0 を使用して、パソコンの RS232C 経由で通信します。printf 関数と scanf 関数を使います。

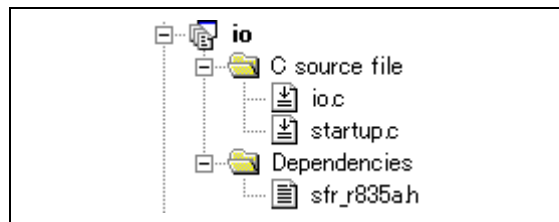
8. 電源が入ってからのマイコンの動作

8.1 概要

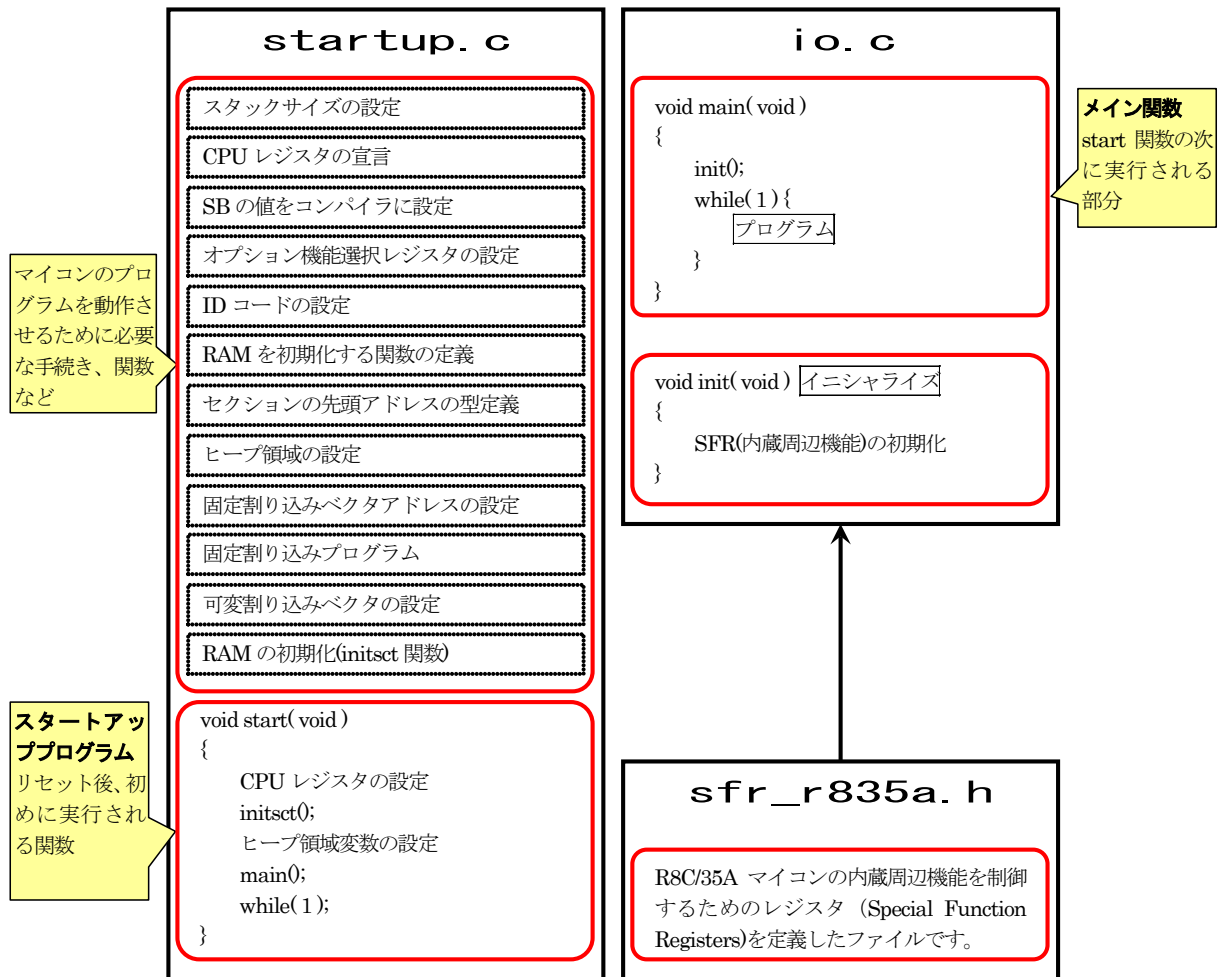
マイコンボードの電源が入ってから、どのようにマイコンが動作するのかを説明します。また、これから実習するプロジェクトのファイル構成、実行順について説明します。

8.2 ファイルの構成

ここでは例として、プロジェクト「io」のファイル構成で説明します。プロジェクト「io」には、3 つのファイル「io.c」、「startup.c」、「sfr_r835a.h」が登録されています。



ファイルの内容を、下記に示します。



8.3 動作手順

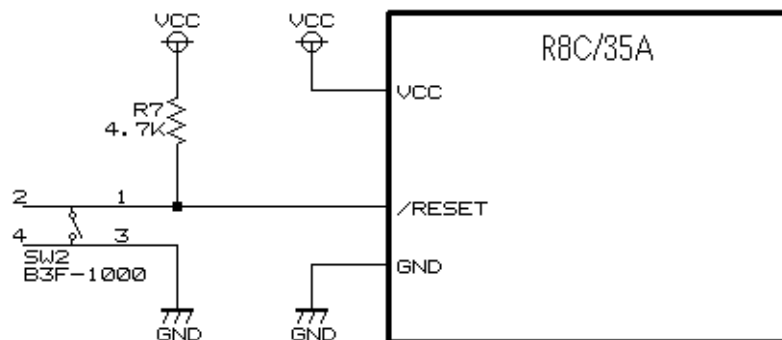
8.3.1 リセットの種類

R8C/35A のリセットには、下記の種類があります。

リセットの名称	要因
ハードウェアリセット	$\overline{\text{RESET}}$ 端子の入力電圧が“0”
パワーオンリセット	VCCの上昇
電圧監視0リセット	VCCの下降(監視電圧:Vdet0)
ウォッチドッグタイマリセット	ウォッチドッグタイマのアンダフロー
ソフトウェアリセット	PM0レジスタのPM03ビットに“1”を書く

$\overline{\text{RESET}}$ 端子に抵抗を介してVCCに接続すると、パワーオンリセット機能が有効になります。本マイコンボードは、4.7kΩの抵抗を介してVCCに接続されており、電源を投入するとパワーオンリセットで立ち上がります。リセット端子の回路を下記に示します。

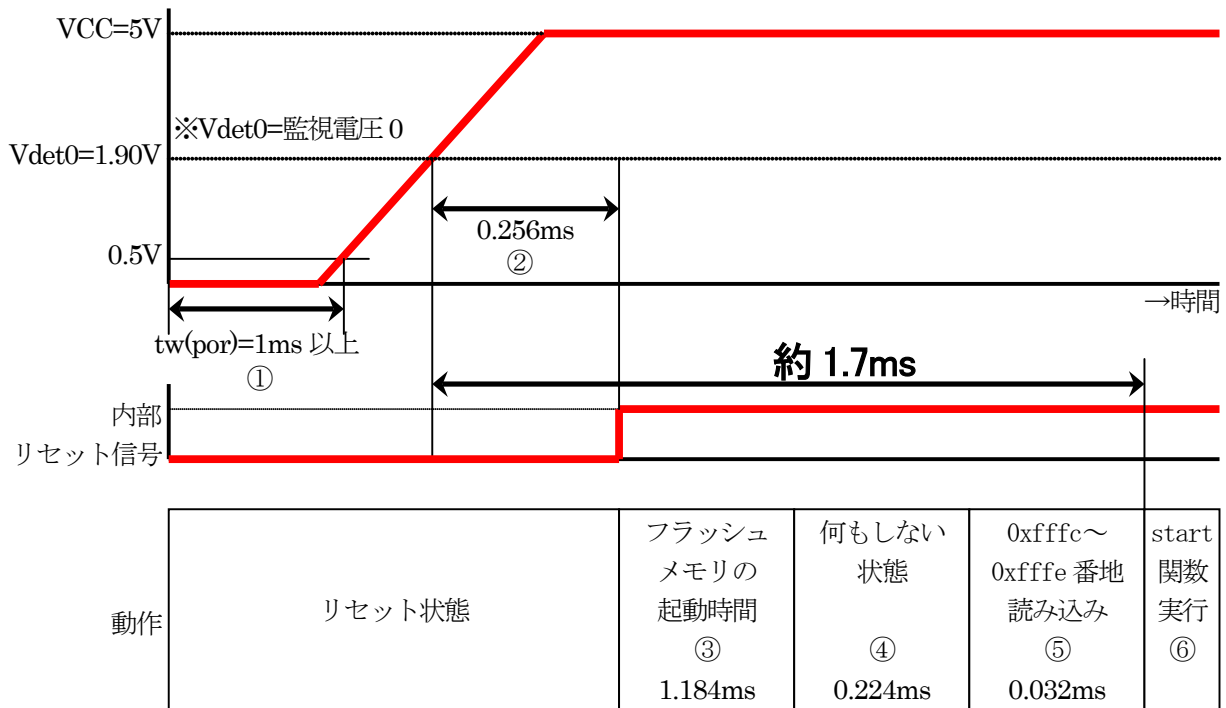
タクトスイッチ SW2 を押すと、強制的に $\overline{\text{RESET}}$ 端子を“0”にしてハードウェアリセットでリセットさせます。SW2 を離すと、パワーオンリセットで立ち上がります。



8. 電源が入ってからのマイコンの動作

8.3.2 電源投入してからプログラム実行までの動作

マイコンボードの電源電圧 (VCC) が立ち上がったときに、マイコンのリセットが解除され、プログラムが実行される動作を下記に示します。



※fOCO-S=低速オンチップオシレータ=約 125kHz

①	0.5V 以下を 1ms 以上続けると、パワーオンリセット機能が有効になります。
②	VCC が 1.90V になってから、リセットが解除される時間は、下記の時間です。 $1/fOCO-S \times 32 = 1/(125 \times 10^3) \times 32 = 0.256ms$
③	フラッシュメモリの起動時間は、下記の時間です。 $1/fOCO-S \times 148 = 1/(125 \times 10^3) \times 148 = 1.184ms$
④	何もしない状態は、下記の時間です。 $1/fOCO-S \times 28 = 1/(125 \times 10^3) \times 28 = 0.224ms$
⑤	0x0fffc~0x0fffe 番地の読み込みは、下記の時間です。 $1/fOCO-S \times 4 = 1/(125 \times 10^3) \times 4 = 0.032ms$ 0x0fffc~0x0fffe 番地にはあらかじめ、初めに実行するアドレスを書き込んでおきます。例えば、0x80dc 番地から実行したければ、 0x0fffc 番地=0xdc 0x0fffd 番地=0x80 0x0fffe 番地=0x00 を書き込んでおきます。リセット後、start 関数から始める設定は startup.c で行います。start 関数が何番地になるかは、ルネサス統合開発環境でビルド実行後に決まります。
⑥	電圧が 1.90V 以上になってから、約 1.7ms 後に 0xfffc~0xfffe 番地に書かれているアドレスのプログラム、すなわち startup.c の start 関数が実行されます。

8. 電源が入ってからのマイコンの動作

```

80 : /*=====*/
81 : /* ヒープ領域の設定 */
82 : /*=====*/
83 : #pragma sectaddress heap_NE, DATA
84 : #define __HEAPSIZE__ 0x100
85 :
86 : unsigned char heap_area[ __HEAPSIZE__ ]; /* ヒープ領域確保 */
87 : extern unsigned char _far *_mnext; /* 次に使用できるメモリの先頭アドレス */
88 : extern unsigned long _msize; /* 残りのバイト数 */
89 :
90 : /*=====*/
91 : /* 固定割り込みベクタアドレスの設定 */
92 : /*=====*/
93 : #pragma sectaddress fvector, ROMDATA 0xffd8
94 :
95 : _asm(" .addr 0FFFFFFFH"); /* 予約 */
96 : _asm(" .byte 0FFH"); /* 0FS2 */
97 : #pragma interrupt/v _dummy_int /* 未定義命令割り込みベクタ */
98 : #pragma interrupt/v _dummy_int /* オーバフロー割り込みベクタ */
99 : #pragma interrupt/v _dummy_int /* BRK命令割り込みベクタ */
100 : #pragma interrupt/v _dummy_int /* アドレス一致割り込みベクタ */
101 : #pragma interrupt/v _dummy_int /* シングルステップ割り込みベクタ */
102 : #pragma interrupt/v _dummy_int /* ウォッチドックタイマなどの割り込みベクタ */
103 : #pragma interrupt/v _dummy_int /* アドレスレイク割り込みベクタ */
104 : #pragma interrupt/v _dummy_int /* 予約 */
105 : #pragma interrupt/v start /* リセットベクタ */
106 :
107 : /*=====*/
108 : /* 固定割り込みプログラム */
109 : /*=====*/
110 : #pragma sectaddress interrupt, CODE
111 : #pragma interrupt _dummy_int()
112 : void _dummy_int(void)
113 : {
114 : /* ダミー関数 */
115 : }
116 :
117 : /*=====*/
118 : /* 可変割り込みベクタの設定 */
119 : /*=====*/
120 : #pragma sectaddress vector, ROMDATA
121 :
122 : /* ここではセクション名の設定のみ行う */
123 :
124 : /*=====*/
125 : /* RAMの初期化 */
126 : /*=====*/
127 : #pragma sectaddress program, CODE
128 : void initset(void)
129 : {
130 : sclear("bss_SE", "data", "align");
131 : sclear("bss_SO", "data", "noalign");
132 : sclear("bss_NE", "data", "align");
133 : sclear("bss_NO", "data", "noalign");
134 :
135 : scopy("data_SE", "data", "align");
136 : scopy("data_SO", "data", "noalign");
137 : scopy("data_NE", "data", "align");
138 : scopy("data_NO", "data", "noalign");
139 : }
140 :
141 : /*=====*/
142 : /* スタートアッププログラム */
143 : /*=====*/
144 : #pragma entry start
145 : void start( void )
146 : {
147 : _isp_ = &_istack_top; /* ISPに割り込みスタックのアドレス設定 */
148 : _flg_ = 0x0080; /* FLGのU=1 */
149 : _sp_ = &_stack_top; /* USPにユーザースタックのアドレス設定 */
150 : _sb_ = 0x0400U; /* SB相対アドレッシングの設定 */
151 : _intbh_ = (unsigned int*)0x00; /* INTBH = vector(上位)に設定 */
152 : _intbl_ = &_vector_top; /* INTBL = vector(下位)に設定 */
153 : initset(); /* RAMの初期化 */
154 : _mnext = &heap_area[0]; /* ヒープ領域変数の設定 */
155 : _msize = (unsigned long)__HEAPSIZE__;
156 : _fb_ = 0U; /* FB = 0 */
157 :
158 : main(); /* main関数実行 */
159 :
160 : while(1);
161 : }
162 :
163 : /*=====*/
164 : /* end of file */
165 : /*=====*/

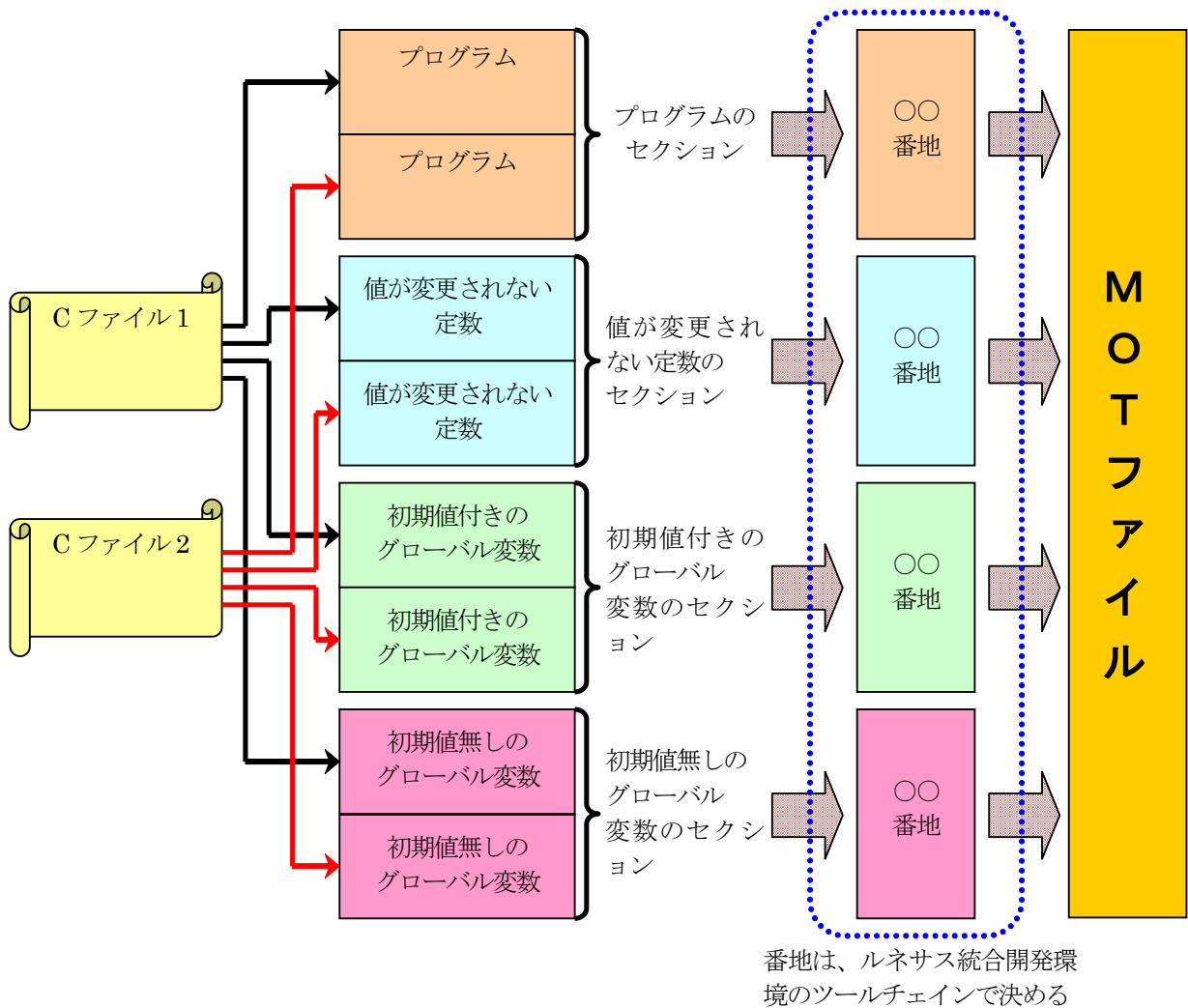
```

8.5 セクション

8.5.1 概要

コンパイラがプログラムをオブジェクトファイル(機械語)に翻訳するとき、プログラム、初期値付きの変数、初期値無しの変数など、内容に応じて分類しながら翻訳します。**分類されたまとまりをセクションと言います**。セクションは「部分、区切り」と翻訳できますが、あまりピンと来ません。ここでは、セクションをグループと言い替えると分かりやすいと思います。

ルネサス統合開発環境のツールチェーンで、どのセクションを何番地に配置するかあらかじめ設定しておきます。ビルドを行うと、ルネサス統合開発環境はその設定に従って何番地に割り当てるか決めて、MOT ファイルを作成します。



8. 電源が入ってからのマイコンの動作

8.5.2 セクションの種類

コンパイラは、プログラムやデータをオブジェクトファイル(機械語)に変換し、セクションごとに分類します。分類したセクション名と内容を下表に示します。

セクション名	内容	H8/3048F-ONE のセクション名 (参考)	配置先の メモリ
data	初期値を持つグローバル変数の領域 ※data セクションは、初期値を ROM に保存し、startup.c の start 関数実行時に RAM にコピーします。	R と D	RAM と ROM
bss	初期値を持たないグローバル変数の領域 ※bss は、「Block Started by Symbol(シンボル名で示された番地から始まるメモリブロック)」の略です。	B	RAM
rom	文字列/定数(const 修飾された変数の領域)	C	ROM
heap	ヒープ領域(メモリ管理関数により動的に確保される領域)	B	RAM
stack	スタック領域(ユーザスタックポインタ(USP)の領域)	B	RAM
istack	割り込みスタック領域(割り込みスタックポインタ(ISP)の領域)	B	RAM
switch_table	switch-case 文のテーブルデータ領域	C	ROM
program	プログラム領域	P	ROM
interrupt	割り込みプログラム領域	P	ROM
vector	可変ベクタ領域	V	ROM

data、bss、rom、heap の各セクションはデータの内容により、下記のようにさらに細かく分かれます。

$\text{data または bss または rom または heap} + \square + \textcircled{1} + \textcircled{2} + \textcircled{3}$
--

番号	属性	意味
①	S	#pragma SBDATA で指定された変数を格納しているデータです。 ※変数に SBDATA を指定すると、プログラムサイズが少なくなります。
	N	near(ニア)データを格納します。near データとは、0x0000～0xffff 番地のアドレスのデータです。
	F	far(ファー)データを格納します。far データとは、0x00000～0xfffff 番地のアドレスのデータです。 ※ミニマイコンカーVer.2 に搭載している R8C/35A マイコンのメモリは、0x0000～0xffff 番地なので、far データはありません。
②	E	偶数(even)サイズのデータを格納します。int 型や long 型などが該当します。
	O	奇数(odd)サイズのデータを格納します。char 型などが該当します。
③	I	初期値(initialize)を持つ変数の初期値を格納します。
	(なし)	初期値を持たない変数です。

8. 電源が入ってからのマイコンの動作

組み合わせを下記に示します。

セクションベース名	①	②	③	セクション名	配置先
data 初期値を持つ グローバル変数の 領域 (RAMとROM)	S SBDATA 指定変数	E 偶数	なし(初期値無し)	data_SE	RAM
			I (初期値付き)	data_SEI	ROM
		O 奇数	なし(初期値無し)	data_SO	RAM
			I (初期値付き)	data_SOI	ROM
	N near データ	E 偶数	なし(初期値無し)	data_NE	RAM
			I (初期値付き)	data_NEI	ROM
		O 奇数	なし(初期値無し)	data_NO	RAM
			I (初期値付き)	data_NOI	ROM
bss 初期値を持たない グローバル変数の 領域 (RAM)	S SBDATA 指定変数	E 偶数		bss_SE	RAM
			O 奇数		bss_SO
	N near データ	E 偶数			bss_NE
			O 奇数		bss_NO
rom 文字列/定数/const 修 飾された変数の領域 (ROM)	N near データ	E 偶数			rom_NE
			O 奇数		rom_NO
heap ヒープ領域 (RAM)	N near データ	E 偶数			heap_NE

アドレスが 0x10000 番地以上の場合、far データになりますが、ミニマイコンカー Ver.2 搭載の R8C/35A マイコンのメモリは 0x0000～0xffff 番地までなので、far データはありません。偶数、奇数については、M16C マイコンの場合(R8C の上位マイコン)、偶数データは2バイトアクセスを行いスピードやメモリ効率を上げることができますが、R8C は偶数でも奇数でもスピードやメモリ効率に変化はありません。

「I」は、data セクションにのみ付きます。詳しくは「8.6.6 RAM の初期化」を参照してください。

SBDATA 指定変数は、グローバル変数に対して指定するので、rom セクション、heap セクションには付きません。

8.5.3 プログラムをセクションに分類

下記プログラムをビルドします。

```
#include "sfr_r835a.h"          /* R8C/35A SFRの定義ファイル */

#pragma sdata c4                /* SBDATA指定 */
#pragma sdata i4                /* SBDATA指定 */

①const char    c1 = 0x00;
②const int     i1 = 0x1234;
③unsigned char c2 = 0xff;
④unsigned int  i2 = 0x2345;
⑤unsigned char c3;
⑥unsigned int  i3;
⑦unsigned char c4;
⑧unsigned int  i4 = 1;

⑨void main( void )
{
    prc2 = i4;
    pd0  = c1;
    pd6  = c2;

    while( 1 ) {
        c4 = p0;
        p6 = c4;
    }
}
```

プログラムやデータは、下記のようなセクションに分類されます。

プログラム	配置させるセクション	説明
①const char c1 = 0x00;	rom_NO	const は値を変更できない変数なので、ROM に配置します。char 型は 1 バイト幅なので奇数アドレスで格納します。
②const int i1 = 0x1234;	rom_NE	const は値を変更できない変数なので、ROM に配置します。int 型は 2 バイト幅なので偶数アドレスで格納します。
③unsigned char c2=0xff;	data_NOI , data_NO	初期値付き変数なので、初期値を ROM に保存し、実行後 RAM にコピーします。ROM 側のセクションが data_NOI、RAM 側のセクションが data_NO です。char 型は 1 バイト幅なので奇数アドレスで格納します。
④unsigned int i2 = 0x2345;	data_NEI , data_NE	初期値付き変数なので、初期値を ROM に保存し、実行後 RAM にコピーします。ROM 側のセクションが data_NEI、RAM 側のセクションが data_NE です。int 型は 2 バイト幅なので偶数アドレスで格納します。

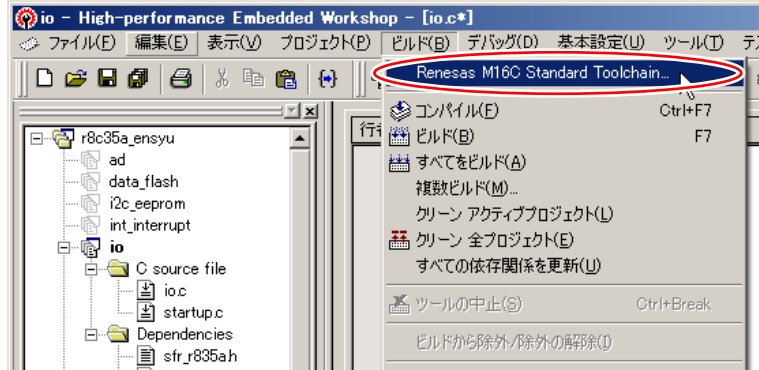
8. 電源が入ってからのマイコンの動作

<p>⑤unsigned char c3;</p>	<p>bss_N0</p>	<p>初期値無し変数なので、RAM に配置します。char 型は 1 バイト幅なので奇数アドレスで格納します。</p>
<p>⑥unsigned int i3;</p>	<p>bss_NE</p>	<p>初期値無し変数なので、RAM に配置します。int 型は 2 バイト幅なので偶数アドレスで格納します。</p>
<p>⑦unsigned char c4;</p>	<p>bss_S0</p>	<p>初期値無し変数なので、RAM に配置します。SBDATA 指定された変数なので、「S」が付きます。char 型は 1 バイト幅なので奇数アドレスで格納します。</p>
<p>⑧unsigned int i4 = 1;</p>	<p>data_SEI , data_SE</p>	<p>初期値付き変数なので、初期値を ROM に保存し、実行後 RAM にコピーします。SBDATA 指定された変数なので「S」が付きます。ROM 側のセクションが data_SEI、RAM 側のセクションが data_SE です。int 型は 2 バイト幅なので偶数アドレスで格納します。</p>
<pre> ⑨void main(void) { prc2 = i4; pd0 = c1; pd6 = c2; while(1) { c4 = p0; p6 = c4; } } </pre>	<p>program</p>	<p>プログラムです。</p>

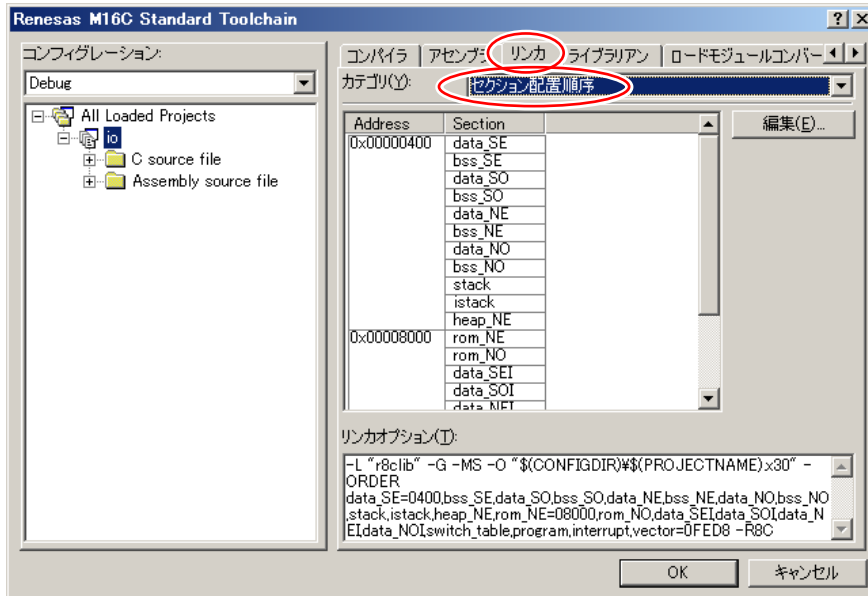
8. 電源が入ってからのマイコンの動作

8.5.4 アドレスに配置

分類したセクションを何番地に配置するかは、ルネサス統合開発環境のツールチェーンで設定します。ルネサス統合開発環境の「ビルド→Renesas M16C Standard Toolchain」(ツールチェーン)を選択します。



「リンカ」タブの「カテゴリ:セクション配置順序」を選択します。



今回のサンプルプログラムで設定されているツールチェーンのアドレスと、セクションの関係を下記に示します。アドレスとセクションの関係を変更したいとき、またはセクションを新たに追加したいときは、ツールチェーンの設定で行ってください。

アドレス	セクション名	アドレス	セクション名
0x0400 (RAM)	data_SE	0x8000 (ROM)	rom_NE
	bss_SE		rom_NO
	data_SO		data_SEI
	bss_SO		data_SOI
	data_NE		data_NEI
	bss_NE		data_NOI
	data_NO		switch_table
	bss_NO		program
	stack		interrupt
	istack		vector
	heap_NE		fvector
		0xfed8 (ROM)	
		0xffd8 (ROM)	

※fvector は、startup.c の「固定割り込みベクタアドレスの設定」で直接アドレス指定をしています。

8. 電源が入ってからのマイコンの動作

8.5.5 セクションの設定

プログラムをどのセクションにするか、設定しなければいけません。「#pragma sectaddress」は、セクションの定義、アドレス指定をする疑似命令です。この命令の次の行のプログラムから、セクション名で定義したセクションになります。ただし、data、bss、rom の 3 つのセクションは、ルネサス統合開発環境のコンパイラが自動で判別するので、セクション名を設定する必要はありません。

```
#pragma sectaddress セクション名, セクションのタイプ アドレス
↑ スペースです
```

セクション名	R8C マイコンは、何番地にどのプログラムやデータを配置するかを「セクション」として管理しています。このセクションの名前を設定します。
セクションのタイプ	定義するセクションがどのタイプか指定します。次の 3 種類あります。 <ul style="list-style-type: none"> • CODE…………プログラム領域として定義します。ルネサス統合開発環境のツールチェーンの設定では、ROM 領域に配置されるようにアドレス指定してください。 • DATA…………可変データ領域として定義します。ルネサス統合開発環境のツールチェーンの設定では、RAM 領域に配置されるようにアドレス指定してください。 • ROMDATA…プログラム以外の固定データ領域として定義します。ルネサス統合開発環境のツールチェーンの設定では、ROM 領域に配置されるようにアドレス指定してください。
アドレス	セクションを何番地に配置するか指定します。ルネサス統合開発環境のツールチェーンで設定するので、普通はここで設定しません。セクションのアドレスを強制的に設定しておきたい場合は、ここで設定しておきます。アドレスを設定しない場合は、何も書きません。

設定例を下記に示します。

```
// fvectorというセクション名を固定データ領域として定義する。アドレスは0xffd8番地から配置する
#pragma sectaddress fvector,ROMDATA 0xffd8
プログラム ←fvectorセクションとしてffd8番地から配置する

// interruptというセクション名をプログラム領域として定義する。
#pragma sectaddress interrupt, CODE
プログラム ←interruptセクションとしてツールチェーンで設定する番地に配置する

// vectorというセクション名を固定データ領域として定義する。
#pragma sectaddress vector,ROMDATA
プログラム ←vectorセクションとしてツールチェーンで設定する番地に配置する

// programというセクション名をプログラム領域として定義する。
#pragma sectaddress program, CODE
プログラム ←programセクションとしてツールチェーンで設定する番地に配置する
```

このように、プログラムならプログラム領域としてセクションを定義など、プログラムやデータの内容に応じてセクションのタイプを定義します。

8.6 プログラムの解説「startup.c」

8.6.1 スタックサイズの設定

```

15 : /*=====*/
16 : /* スタックサイズの設定          */
17 : /*=====*/
18 : #pragma STACKSIZE      0x100
19 : #pragma ISTACKSIZE     0x100
    
```

18 行	<p>「#pragma STACKSIZE」は、ユーザスタックポインタ(USP)で使用するスタックサイズを定義する命令です。今回は、0x100 個(256 個)確保しています。例えば、関数を呼び出すときの戻り先のアドレスや引数領域の確保など、このエリアを使います。</p> <p>この命令を実行すると、下記のアセンブルプログラムが生成されます。</p> <pre> .section stack,DATA stack セクションを生成します .blkb 00000100h blank bytes の略で、指定したバイト数の RAM 領域を確保します __stack_top: セクションの先頭アドレスを「__stack_top」というラベルで 定義します </pre>
19 行	<p>「#pragma ISTACKSIZE」は、割り込みスタックポインタ(ISP)で使用するスタックサイズを定義する命令です。今回は、0x100 個(256 個)確保しています。割り込みが発生したときの割り込みが終わったときの戻り先やフラグレジスタ(FLG)の値、割り込みプログラム内で使う変数領域の確保など、このエリアを使います。</p> <p>この命令を実行すると、下記のアセンブルプログラムが生成されます。</p> <pre> .section istack,DATA istack セクションを生成します .blkb 00000100h blank bytes の略で、指定したバイト数の RAM 領域を確保します __istack_top: セクションの先頭アドレスを「__istack_top」というラベルで 定義します </pre>

8. 電源が入ってからのマイコンの動作

8.6.2 CPUレジスタの宣言

```

21 : /*=====*/
22 : /* CPUレジスタの宣言 */
23 : /*=====*/
24 : #pragma CREG _flg_ flg
25 : #pragma CREG _isp_ isp
26 : #pragma CREG _sp_ sp
27 : #pragma CREG _sb_ sb
28 : #pragma CREG _fb_ fb
29 : #pragma CREG _intbh_ intbh
30 : #pragma CREG _intbl_ intbl
31 :
32 : unsigned int _flg_;
33 : unsigned int _sb_;
34 : unsigned int _fb_;
35 : unsigned int *_sp_;
36 : unsigned int *_isp_;
37 : unsigned int *_intbh_;
38 : unsigned int *_intbl_;
    
```

24行 ～ 30行	<p>「#pragma CREG」命令を使って、CPUレジスタをC言語プログラムで使えるように定義します。start関数内で、CPUレジスタに値を設定しています。そこで使えるように定義しています。</p> <p>「#pragma CREG」命令の使い方を、下記に示します。</p> <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <pre>#pragma CREG C言語プログラムでの変数名 CPUレジスタ名</pre> </div>
32行～ 38行	<p>定義した変数(CPUレジスタ)の型を宣言します(※)。</p>

※CPUレジスタの型の定義

CPUレジスタ名	CPUレジスタの意味	C言語プログラムでの変数名	型
FLG	フラグレジスタ	_flg_	unsigned int
ISP	割り込みスタックポインタ	_isp_	unsigned int
SP	ユーザスタックポインタ	_sp_	unsigned int
SB	スタティックベースレジスタ	_sb_	unsigned int *
FB	フレームベースレジスタ	_fb_	unsigned int *
INTBH	割り込みテーブルレジスタ上位	_intbh_	unsigned int *
INTBL	割り込みテーブルレジスタ下位	_intbl_	unsigned int *

8. 電源が入ってからのマイコンの動作

8.6.3 SB の値をコンパイラに設定

```

40 : /*=====*/
41 : /* SBの値をコンパイラに設定 */
42 : /*=====*/
43 :     _asm("     .glb     __SB__\n"
44 :         "     __SB__ .equ     0400H     ");

```

「.glb」は、SB(スタティックベースレジスタ)の値をアセンブラに知らせるアセンブラ指示命令です。この命令を C 言語で記述することはできないので、C 言語プログラムの中でアセンブリ言語の記述をすることのできるインラインアセンブル機能(asm 関数)を使って記述します。_asm 関数は、「_asm(" ");」と記述し、ダブルクォーテーションの中にアセンブリ言語の命令を記述します。

今回のプログラムは、下記のようにアセンブリ言語に変換されます。

```

     .glb     __SB__     シンボルが外部シンボルであることを宣言します
__SB__ .equ     0400H     __SB__に0400H(16進数)を設定します

```

「0400H」は、RAM の先頭番地の 0x400 番地を示しています。SB(スタティックベースレジスタ)を使って、0x0400 ~0x04ff 番地にある変数(もしくは、SB に設定したアドレス~+ff 番地のアドレスまで)を参照するプログラムの命令コードのサイズを小さくすることができます。方法は後述します。

8.6.4 オプション機能選択レジスタの設定

```

46 : /*=====*/
47 : /* オプション機能選択レジスタの設定 */
48 : /*=====*/
49 :     _asm("     .ofsreg 0BFH     ");           /* OFS = 0xbf (ハワリオンセット使用) */

```

オプション機能選択レジスタ(OFS)を設定します。

オプション機能選択レジスタ(OFS)は 0xffff 番地の ROM 上にあるので、プログラム中で設定するのではなく、ROM に書き込むときにあらかじめ設定しておきます。

オプション機能選択レジスタ(OFS)に値を設定する専用のアセンブラ指示命令があり、下記のように記述します。

```

     .ofsreg OFSに設定する値

```

C 言語では、アセンブラ指示命令は使えないため、インラインアセンブル機能(C 言語プログラムの中に、アセンブラ命令を記述できる命令:asm 関数)を使って、記述します。

```

     _asm("     .ofsreg OFSに設定する値");

```

8. 電源が入ってからのマイコンの動作

オプション機能選択レジスタ(OFS)の各ビットの意味を、下記に示します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7	リセット後カウントソース保護 モード選択ビット CSPROINI	0:リセット後、カウントソース保護モード有効 1:リセット後、カウントソース保護モード無効 リセット後、ウォッチドッグタイマのカウントソース保護モードをどうするか設定します。今回は無効にします。	1
bit6	電圧検出 0 回路起動ビット (注 3) LVDAS	0:リセット後、電圧監視 0 リセット有効 1:リセット後、電圧監視 0 リセット無効 パワーオンリセットを使用するときは有効にします。	0
bit5,4	電圧検出 0 レベル選択ビット (注 2) VDSEL1,0	00:3.80V を選択(Vdet0_3) 01:2.85V を選択(Vdet0_2) 10:2.35V を選択(Vdet0_1) 11:1.90V を選択(Vdet0_0) bit6="0"のとき、電圧を選択します。今回はパワーオンリセットの電圧を選択します。"11"を設定します。	11
bit3	ROM コードプロテクトビット ROMCP1	0:ROM コードプロテクト有効 1:ROM コードプロテクト解除 パラレル入出力モードというモードを使ってマイコン内蔵メモリへ読み書きするとき、ROM を読めるようにするか読めないようにするか設定します。パソコンからの書き込みはシリアル入出力モードで今回はこのモードは使いませんが、読めるようにします。	1
bit2	ROM コードプロテクト解除ビット ROMCR	0:ROM コードプロテクト解除 1:ROMCP1 ビット有効 bit3 を有効にするか、無効にするかの設定です。有効にします。	1
bit1		"1"を設定	1
bit0	ウォッチドッグタイマ起動選択 ビット WDTON	0:リセット後、ウォッチドッグタイマは自動的に起動 1:リセット後、ウォッチドッグタイマは停止状態 今回は、リセット後にウォッチドッグタイマをしません。	1

注 1. OFS を含むブロックを消去すると、OFS は 0xff になります。

注 2. VDSEL0～VDSEL1 ビットで選択した電圧検出 0 レベルは、電圧監視 0 リセットおよびパワーオンリセットの両機能に、同じレベルで設定されます。

注 3. パワーオンリセット、電圧監視 0 リセットを使用する場合、LVDAS ビットを"0"(リセット後、電圧監視 0 リセット有効)にしてください。

8. 電源が入ってからのマイコンの動作

オプション機能選択レジスタ(OFS)の設定値を、下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	1	0	1	1	1	1	1	1
16進数	b				f			

よって、今回は下記のようにプログラムします。

```
_asm(" .ofsreg 0BFH"); // アセンブラ命令で16進数を記述するときは、最後にHを付けます
```

8.6.5 IDコードの設定

IDコードを設定します。

```
51 : /*=====*/
52 : /* IDコードの設定 */
53 : /*=====*/
54 : _asm(" .id ""¥"#FFFFFFFFFFFFFF¥" ");
```

IDコードは、シリアル入出力モード時(パソコンから書き込むとき、このモードです)に不正書き込みを防止するセキュリティ機能です。

パソコンから送られてくるチェック用のIDコードと、フラッシュROM内に書かれている7バイトのIDコードを判定して、一致しない場合はパソコンから送られてくるコマンドを受け付けず、書き込みをできなくする機能です。

IDコードは、固定割り込みベクタテーブル領域のそれぞれのジャンプ先アドレスの合間に7個あります。

アドレス	内容
0x0ffd8	予約領域
0x0ffd9	予約領域
0x0ffda	予約領域
0x0ffdb	オプション機能選択レジスタ2(OFS2)
0x0ffdc	未定義命令割り込み発生時のジャンプ先アドレス 下位
0x0ffdd	未定義命令割り込み発生時のジャンプ先アドレス 中位
0x0ffde	未定義命令割り込み発生時のジャンプ先アドレス 上位
0x0fdf	ID1
0x0ffe0	オーバフロー割り込み発生時のジャンプ先アドレス 下位
0x0ffe1	オーバフロー割り込み発生時のジャンプ先アドレス 中位
0x0ffe2	オーバフロー割り込み発生時のジャンプ先アドレス 上位
0x0fe3	ID2
0x0ffe4	BRK 命令割り込み発生時のジャンプ先アドレス 下位
0x0ffe5	BRK 命令割り込み発生時のジャンプ先アドレス 中位
0x0ffe6	BRK 命令割り込み発生時のジャンプ先アドレス 上位
0x0ffe7	
0x0ffe8	アドレス一致割り込み発生時のジャンプ先アドレス 下位
0x0ffe9	アドレス一致割り込み発生時のジャンプ先アドレス 中位
0x0ffea	アドレス一致割り込み発生時のジャンプ先アドレス 上位
0x0feb	ID3

8. 電源が入ってからのマイコンの動作

0x0ffec	シングルステップ割り込み発生時のジャンプ先アドレス 下位
0x0ffed	シングルステップ割り込み発生時のジャンプ先アドレス 中位
0x0ffee	シングルステップ割り込み発生時のジャンプ先アドレス 上位
0x0ffef	ID4
0x0fff0	ウォッチドッグタイマ、発振停止検出、電圧検出割り込み発生時のジャンプ先アドレス 下位
0x0fff1	ウォッチドッグタイマ、発振停止検出、電圧検出割り込み発生時のジャンプ先アドレス 中位
0x0fff2	ウォッチドッグタイマ、発振停止検出、電圧検出割り込み発生時のジャンプ先アドレス 上位
0x0fff3	ID5
0x0fff4	アドレスブレイク割り込み発生時のジャンプ先アドレス 下位
0x0fff5	アドレスブレイク割り込み発生時のジャンプ先アドレス 中位
0x0fff6	アドレスブレイク割り込み発生時のジャンプ先アドレス 上位
0x0fff7	ID6
0x0fff8	(予約)
0x0fff9	(予約)
0x0fffa	(予約)
0x0fffb	ID7
0x0fffc	リセット時のジャンプ先アドレス 下位
0x0fffd	リセット時のジャンプ先アドレス 中位
0x0fffe	リセット時のジャンプ先アドレス 上位
0x0ffff	オプション機能選択レジスタ(OFS)

※固定割り込みベクタテーブルは、ROM 領域なので、プログラム書き込み時に設定します。プログラム実行後に書き換えはできません。

※ID1～ID7:ID コード領域です。

プログラムで ID1～ID7 に値を設定します。ID1(0x0ffdf 番地)に〇〇を書き込んで、ID2(0xffe3 番地)に〇〇を書き込んで・・・というように、7 個の値を個別に設定するのは大変です。そのため、ID コードをまとめて設定する専用のアセンブラ指示命令があり、下記のように記述します。


```
.id "#ID1～ID7"
```

C 言語では、アセンブラ指示命令は使えないため、インラインアセンブル機能(C 言語プログラムの中に、アセンブラ命令を記述できる命令:asm 関数)を使って、記述します。

```
_asm(" .id ""¥"#ID1～ID7¥" ");
```

分解すると、下記ようになります。

```
_asm( " .id " " ¥" # ID1～ID7 ¥" );
```

「」が 1 つの区切りです。2 組に分けて、ダブルクォーテーションで区切っています。「¥」は、文字列の開始と終了を定義する「"」と区別するために「¥」として文字列の中のダブルクォーテーションであることを示しています(エスケープ文字といいます)。

8. 電源が入ってからのマイコンの動作

本当は下記のようにダブルクォーテーション1組だけでよいのですが、過去のコンパイラの仕様で2組に区切っています。今回インストールしたコンパイラは下記のように記述して問題ありませんが、過去の名残のままにしています。

```
_asm( " .id ¥" # ID1~ID7 ¥" );
```

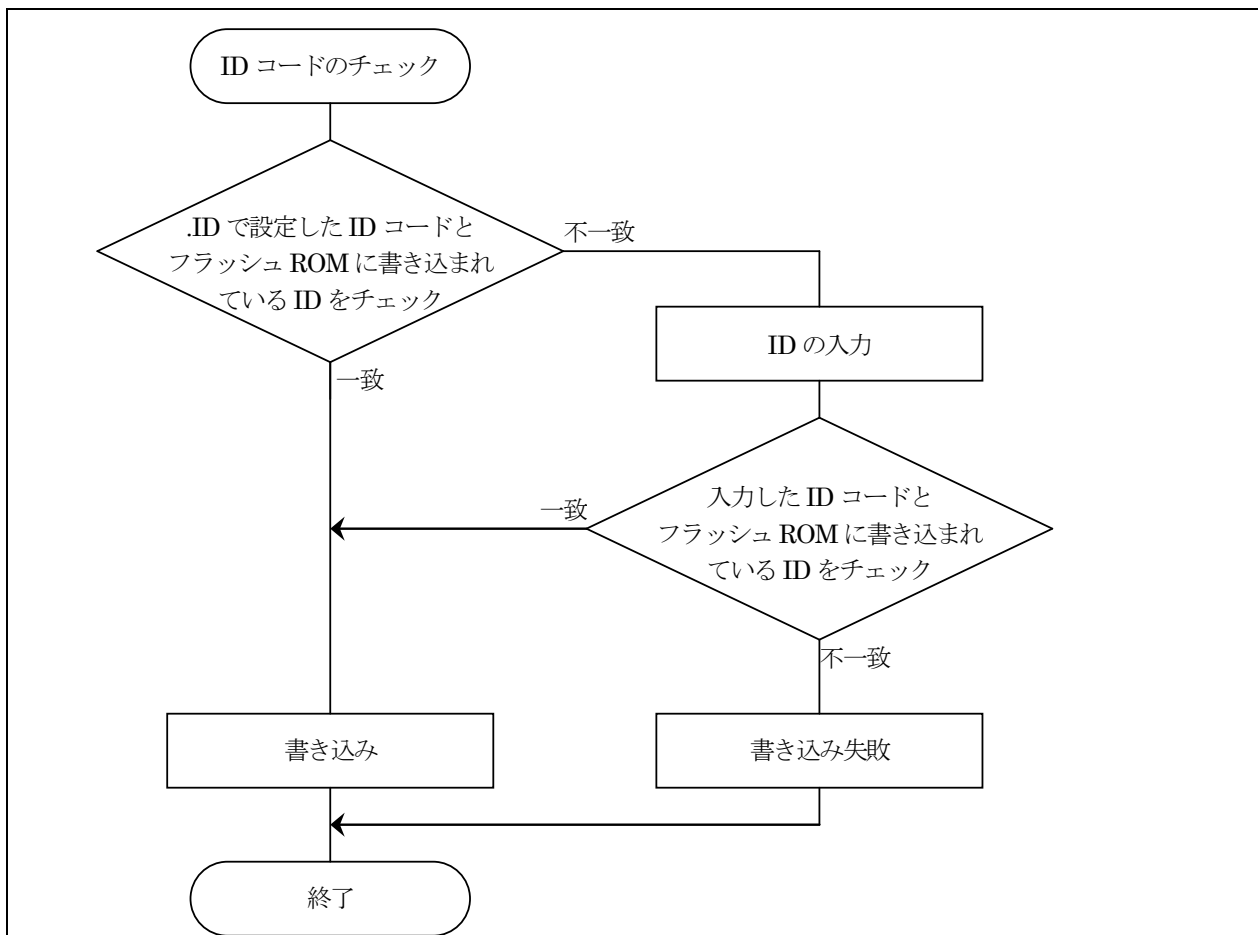
例えば、ID1=0x12、ID2=0x34、ID3=0x56、ID4=0x78、ID5=0x9a、ID6=0xbc、ID7=0xde にしたいとき、下記のように記述します。

```
_asm( " .id "¥"#123456789abcde¥" );
```

※IDコードは、不用意に書き換えないでください。不用意に書き換えると、書き込みができなくなります。

※ID コードチェックについて

プログラム書き込み時、次のように ID をチェックして書き込みを行います。



そのため、前回フラッシュROMに書き込んだIDコードを忘れると、プログラムの書き換えができなくなります。ID は不用意に換えないようにしてください。

ID コードを忘れた場合は、ID コードをアスキーコードで「ALeRASE」にするとフラッシュ ROM を強制イレーズ (0xff で埋めます) します。R8C Writer は ID コードが 0xff 以外の場合、ROM 強制イレーズしてから書き込みます。

8. 電源が入ってからのマイコンの動作

8.6.6 RAM の初期化

RAM のグローバル変数を初期化します。

```

56 : /*=====*/
57 : /* RAMを初期化する関数の定義          */
58 : /*=====*/
59 : #define scopy(X,Y,Z)  _asm(" .initsct "X","Y","Z"\n"¥
60 :      " .initsct "X"l,rom"Y",noalign¥n"¥
61 :      " mov.w  #(topof "X"l)&0ffffH,A0¥n"¥
62 :      " mov.b  #(topof "X"l)>>16,R1H¥n"¥
63 :      " mov.w  #(topof "X")&0ffffH,A1¥n"¥
64 :      " mov.w  #sizeof "X",R3¥n"¥
65 :      " smovf.b")
66 :
67 : #define sclear(X,Y,Z)  _asm(" .initsct "X","Y","Z"\n"¥
68 :      " mov.b  #00H,R0L¥n"¥
69 :      " mov.w  #(topof "X") ,A1¥n"¥
70 :      " mov.w  #sizeof "X",R3¥n"¥
71 :      " sstr.b")

```

中略

```

129 : /*=====*/
130 : /* RAMの初期化          */
131 : /*=====*/
132 : #pragma sectaddress program, CODE セクション名はprogramでプログラム領域として定義
133 : void initsct(void)
134 : {
135 :     sclear("bss_SE","data","align");
136 :     sclear("bss_SO","data","noalign");
137 :     sclear("bss_NE","data","align");
138 :     sclear("bss_NO","data","noalign");
139 :
140 :     scopy("data_SE","data","align");
141 :     scopy("data_SO","data","noalign");
142 :     scopy("data_NE","data","align");
143 :     scopy("data_NO","data","noalign");
144 : }

```

グローバル変数は、RAM エリアを使います。電源を入れたとき、RAM の内容は不定(どのような値になっているか分からない)の状態です。C 言語では、下記のように決まっています。

変数	C 言語での決まり	実際(RAM の値)
初期値付きのグローバル変数	グローバル変数には、初期値が入っています。	不定です。
初期値無し of グローバル変数	グローバル変数には、0が入っています。	不定です。

そのため、start 関数内で、C 言語の決まりに合うようにします。これが 59～71 行で定義している「scopy」と「sclear」関数です。これらの関数を startup.c の 135～143 行で実行します。

8. 電源が入ってからのマイコンの動作

■scopy 関数

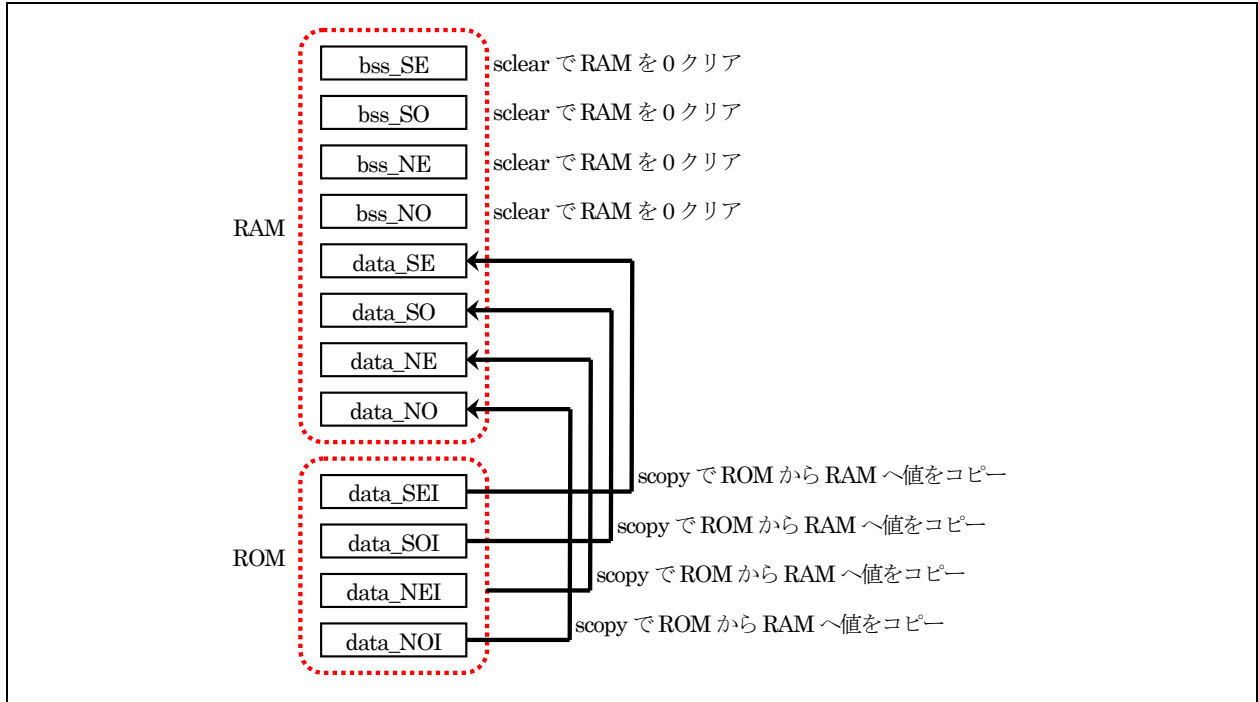
書式	偶数セクションの場合 : scopy("セクション名", "data", "align"); 奇数セクションの場合 : scopy("セクション名", "data", "noalign");								
内容	ROM にある初期値付きのグローバル変数の値を、RAM にコピーします。								
引数	セクション名… ROM から RAM へコピーしたい初期値付きグローバル変数のセクションを指定します。セクション名は「I」を抜いて指定します。								
戻り値	無し								
プログラム内容	<p>initsct 関数中で下記のようにプログラムを実行して、セクションの内容をコピーします。</p> <pre>scopy("data_SE", "data", "align"); scopy("data_SO", "data", "noalign"); scopy("data_NE", "data", "align"); scopy("data_NO", "data", "noalign");</pre> <table border="1" style="display: inline-table; vertical-align: top;"> <tr><td>data_SEI</td><td>→ data_SE へコピー</td></tr> <tr><td>data_SOI</td><td>→ data_SO へコピー</td></tr> <tr><td>data_NEI</td><td>→ data_NE へコピー</td></tr> <tr><td>data_NOI</td><td>→ data_NO へコピー</td></tr> </table>	data_SEI	→ data_SE へコピー	data_SOI	→ data_SO へコピー	data_NEI	→ data_NE へコピー	data_NOI	→ data_NO へコピー
data_SEI	→ data_SE へコピー								
data_SOI	→ data_SO へコピー								
data_NEI	→ data_NE へコピー								
data_NOI	→ data_NO へコピー								

■sclear 関数

書式	偶数セクションの場合 : sclear("セクション名_NE", "data", "align"); 奇数セクションの場合 : sclear("セクション名_NO", "data", "noalign");								
内容	初期値無し of グローバル変数の値を、0 クリアします。								
引数	セクション名… 0 クリアしたいセクションを指定します。								
戻り値	無し								
プログラム内容	<p>initsct 関数中で下記のようにプログラムを実行して、セクションを 0 クリアします。</p> <pre>sclear("bss_SE", "data", "align"); sclear("bss_SO", "data", "noalign"); sclear("bss_NE", "data", "align"); sclear("bss_NO", "data", "noalign");</pre> <table border="1" style="display: inline-table; vertical-align: top;"> <tr><td>bss_SE</td><td>を 0 クリア</td></tr> <tr><td>bss_SO</td><td>を 0 クリア</td></tr> <tr><td>bss_NE</td><td>を 0 クリア</td></tr> <tr><td>bss_NO</td><td>を 0 クリア</td></tr> </table>	bss_SE	を 0 クリア	bss_SO	を 0 クリア	bss_NE	を 0 クリア	bss_NO	を 0 クリア
bss_SE	を 0 クリア								
bss_SO	を 0 クリア								
bss_NE	を 0 クリア								
bss_NO	を 0 クリア								

8. 電源が入ってからのマイコンの動作

グローバル変数の初期化の様子を下記に示します。



8.6.7 セクションの先頭アドレスの型定義

セクションの先頭アドレスの型定義を行います。

```

73 : /*=====*/
74 : /* セクションの先頭アドレスの型定義 */
75 : /*=====*/
76 : extern unsigned int _stack_top;
77 : extern unsigned int _istack_top;
78 : extern unsigned int _vector_top;
    
```

コンパイラは、セクションの先頭アドレスに変数名を設定します。変数名は、下記のようなルールで生成します。

$$\boxed{\text{[]}} + \boxed{\text{セクション名}} + \boxed{\text{[_top]}}$$

ここでは、設定されたセクションの先頭アドレスにある変数の型を定義します。さらに extern (外部変数) 宣言もしています。「extern」は、定義されている変数を、別な C 言語ソースファイルで参照できるようにする宣言です。今回の定義内容を下表に示します。

セクション名	セクションの先頭アドレスにある変数	変数の型
stack	_stack_top	unsigned int
istack	_istack_top	unsigned int
vector	_vector_top	unsigned int

startup.c では、これらの 3 つのセクションの先頭アドレスを使うので、型を定義しています。これらの変数は、startup.c ではない場所で定義されているので、extern (外部変数) 宣言しておきます。

8. 電源が入ってからのマイコンの動作

8.6.8 ヒープ領域の設定

ヒープ領域とは、動的に確保可能なメモリの領域のことです。

```

80 : /*=====*/
81 : /* ヒープ領域の設定 */
82 : /*=====*/
83 : #pragma sectaddress heap_NE, DATA
84 : #define __HEAPSIZE__ 0x100
85 :
86 : unsigned char heap_area[ __HEAPSIZE__ ]; /* ヒープ領域確保 */
87 : extern unsigned char _far *_mnext; /* 次に使用できるメモリの先頭アドレス */
88 : extern unsigned long _msize; /* 残りのバイト数 */
    
```

83 行	ヒープ領域は、RAM にデータとして確保するので、可変データ領域(DATA)として定義します。セクション名は、heap_NE とします。
84 行	「__HEAPSIZE__」という定数に、ヒープ領域のサイズを定義します。今回は、0x100 個分(256 個)ヒープ領域を確保します。 もし、ヒープ領域のサイズを換えたい場合はこの数字を変更してください。
86 行	unsigned char 型で heap_area という配列を宣言しています。サイズは、「__HEAPSIZE__」個分の領域です。
87 行	unsigned char*型で _mnext という変数を宣言しています。この変数は、ヒープ領域を使用する先頭アドレスを示しています。ヒープ領域を使うと、ヒープ領域を使った関数が変数の値を変更します。ユーザ側で変数の値を変更することはできません。「_far」は UNIX 仕様に合わせるために指定しています。
88 行	unsigned long 型で _msize という変数を宣言しています。この変数は、ヒープ領域を使うことのできる残メモリ容量を示します。ユーザ側で変更することはできません。

※ヒープ領域を使う理由

下記の左プログラムのように、str 配列を宣言した場合、一時的にしか使わないのにずっとメモリが使われたままになります。下記の右プログラムのように、malloc 関数などで必要に応じてメモリを確保して、必要なくなったら解放すれば、メモリを有効活用することができます。

<pre> #include <stdio.h> int main(void) { char str[80]; gets(str); /* 文字列を入力 */ puts(str); /* 文字列の表示 */ } </pre> <p>続きのプログラム ※str 配列をもう使わなくともメモリは占有されるので str [80]は確保されたまま</p>	<pre> #include <stdio.h> #include <stdlib.h> int main(void) { char *str; /* 文字列のためのメモリを確保 */ str = (char *)malloc(80); 80 バイト確保 if(str == NULL) { printf("メモリが確保できません\n"); exit(EXIT_FAILURE); } gets(str); /* 文字列を入力 */ puts(str); /* 文字列の表示 */ /* メモリの解放 */ free(str); 80 バイト解除 } </pre> <p>続きのプログラム</p>
--	---

8. 電源が入ってからのマイコンの動作

8.6.9 固定割り込みベクタアドレスの設定

```

90 : /*=====*/
91 : /* 固定割り込みベクタアドレスの設定 */
92 : /*=====*/
93 : #pragma sectaddress fvector,ROMDATA 0xffd8
94 :
95 :     _asm(" .addr 0FFFFFFH"); /* 予約 */
96 :     _asm(" .byte 0FFH"); /* OFS2 */
97 : #pragma interrupt/v _dummy_int /* 未定義命令割り込みベクタ */
98 : #pragma interrupt/v _dummy_int /* オーバフロー割り込みベクタ */
99 : #pragma interrupt/v _dummy_int /* BRK命令割り込みベクタ */
100 : #pragma interrupt/v _dummy_int /* アドレス一致割り込みベクタ */
101 : #pragma interrupt/v _dummy_int /* シングルステップ割り込みベクタ */
102 : #pragma interrupt/v _dummy_int /* ウォッチドッグタイマなどの割り込みベクタ */
103 : #pragma interrupt/v _dummy_int /* アドレスブレイク割り込みベクタ */
104 : #pragma interrupt/v _dummy_int /* 予約 */
105 : #pragma interrupt/v start /* リセットベクタ */
106 :
107 : /*=====*/
108 : /* 固定割り込みプログラム */
109 : /*=====*/
110 : #pragma sectaddress interrupt, CODE
111 : #pragma interrupt _dummy_int()
112 : void _dummy_int(void)
113 : {
114 :     /* ダミー関数 */
115 : }

```

アドレス	
ffdc	ffde, fddf
ffdb	
ffe0	ffe1, ffe2, ffe3
ffe4	ffe5, ffe6, ffe7
ffe8	ffe9, ffea, ffeb
ffec	ffed, ffee, ffef
fff0	fff1, fff2, fff3
fff4	fff5, fff6, fff7
fff8	fff9, fffa, fffb
fffc	fffd, fffe, ffff

93 行	固定割り込みベクタアドレスは、ROM に配置します。ベクタアドレスは、アドレスを羅列したデータなので固定データ領域(ROMDATA)として定義します。セクション名は「fvector」とし、0xffd8 番地から配置します。アドレスは変更できません。
95 行	「.addr」は、3 バイト長のデータを ROM 領域に格納する、アセンブラ指示命令です。C 言語プログラムでアセンブラ指示命令は記述できないので、_asm 関数で記述します。予約領域なので、0xff で埋めておきます。
96 行	「.byte」は、1 バイト長のデータを ROM 領域に格納する、アセンブラ指示命令です。C 言語プログラムでアセンブラ指示命令は記述できないので、_asm 関数で記述します。0xffdb 番地は、オプション機能選択レジスタ 2(OFS2)です。0xff を設定します。
97 行	「#pragma interrupt/v」で、割り込み処理関数のアドレスを設定します。0xffdc～0xffdf 番地に「_dummy_int」関数があるアドレスを書きこみます。この番地は、未定義命令割り込み発生時のジャンプ先アドレスを記述しておきます。例えば、「_dummy_int」関数が、0x8070 番地にある場合、0xffdc 番地=0x70, 0xffdd 番地=0x80, 0xffde 番地=0x00, 0xffdf 番地=0x00 が設定されます。ただし、0xffdf 番地は ID1 の値が上書きされます。
98 行	0xffe0～0xffe3 番地に「_dummy_int」関数があるアドレスを書きこみます。この番地は、オーバフロー割り込み発生時のジャンプ先アドレスを記述しておきます。ただし、0xffe3 番地は ID2 の値が上書きされます。

8. 電源が入ってからのマイコンの動作

99 行	0xffe4～0xffe7 番地に「_dummy_int」関数があるアドレスを書きこみます。この番地は、BRK 命令割り込み発生のジャンプ先アドレスを記述しておきます。
100 行	0xffe8～0xffeb 番地に「_dummy_int」関数があるアドレスを書きこみます。この番地は、アドレス一致割り込み発生時のジャンプ先アドレスを記述しておきます。 ただし、0xffeb 番地は ID3 の値が上書きされます。
101 行	0xffec～0xffef 番地に「_dummy_int」関数があるアドレスを書きこみます。この番地は、シングルステップ割り込み発生時のジャンプ先アドレスを記述しておきます。 ただし、0xffef 番地は ID4 の値が上書きされます。
102 行	0xffff0～0xffff3 番地に「_dummy_int」関数があるアドレスを書きこみます。この番地は、ウォッチドッグタイマ、発振停止検出、電圧検出割り込み発生時のジャンプ先アドレスを記述しておきます。 ただし、0xffff3 番地は ID5 の値が上書きされます。
103 行	0xffff4～0xffff7 番地に「_dummy_int」関数があるアドレスを書きこみます。この番地は、アドレスブレイク割り込み発生時のジャンプ先アドレスを記述しておきます。 ただし、0xffff7 番地は ID6 の値が上書きされます。
104 行	0xffff8～0xffffb 番地に「_dummy_int」関数があるアドレスを書きこみます。この番地は、現在ほどの割り込み発生時のジャンプ先アドレスか決まっていますが、将来使われる可能性があります。一応、「_dummy_int」関数を設定しておきます。 ただし、0xffffb 番地は ID7 の値が上書きされます。
105 行	0xffffc～0xfffff 番地に「start」関数があるアドレスを書きこみます。この番地は、リセット時のジャンプ先アドレスを記述しておきます。 ただし、0xfffff 番地はオプション機能選択レジスタ(OFS)の値が上書きされます。
110 行	固定割り込みのプログラムは、ROM に配置します。またプログラムなので、プログラム領域(CODE)として定義します。セクション名は「interrupt」とします。
111 行	_dummy_int 関数が、固定割り込みで呼ばれる関数であることを指定します。
112 行 ～ 115 行	割り込みプログラムである_dummy_int 関数の内容です。今回は、何もしていません。

※オプション機能選択レジスタ 2(OFS2)

オプション機能選択レジスタ 2(OFS2)の各ビットの意味を下記に示します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7~4		"1111"を設定	1111
bit3,2	ウォッチドッグタイマリフレッシュ 受付周期設定ビット bit3:WDTRCS1 bit2:WDTRCS0	00:25% 01:50% 10:75% 11:100% 特に設定しませんのでデフォルトの"11"を設定します。	11
bit1,0	ウォッチドッグタイマアンダフ ロー周期設定ビット bit1:WDTUFS1 bit0:WDTUFS0	00:03FFh 01:0FFFh 10:1FFFh 11:3FFFh 特に設定しませんのでデフォルトの"11"を設定します。	11

注 1. OFS2 を含むブロックを消去すると、OFS2 は 0xff になります。

オプション機能選択レジスタ 2(OFS2)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	1	1	1	1	1	1	1	1
16 進数	f				f			

※「#pragma interrupt」命令について

「#pragma interrupt」命令は、下記の種類があります。

■固定割り込み関数名の先頭アドレス設定

書式	#pragma interrupt/v 固定割り込み処理関数名
内容	固定割り込み処理関数名 の先頭アドレスを設定します。
使用例	#pragma interrupt/v start → start 関数が 0x8020 番地の場合、 0x20, 0x80, 0x00, 0x00 の 4 バイトに変換される

■固定割り込み処理関数の設定

書式	#pragma interrupt 固定割り込み処理関数名
内容	固定割り込み処理関数名 の関数が、固定割り込みで呼ばれる関数であることを指定します。「/v」とペアで使います。指定すると、次のことが行われます。 <ul style="list-style-type: none"> 関数の最初に全レジスタのスタックへの待避、最後に全レジスタのスタックへの復帰 関数の最後は、reit 命令によるリターン
使用例	<pre>#pragma interrupt _dummy_int() void _dummy_int(void) { 割り込みプログラム } </pre> → <pre>void _dummy_int(void) { pushm r0, r1, r2, r3, a0, a1, fb 割り込みプログラム popm r0, r1, r2, r3, a0, a1, fb reit 割り込みからの復帰 } </pre>

■可変割り込み処理関数の設定

書式	#pragma interrupt 割り込み処理関数名 (vect= ソフトウェア割り込み番号)
内容	割り込み処理関数名 の関数が、 ソフトウェア割り込み番号 で指定した割り込みであることを設定します。設定すると、次のことが行われます。 <ul style="list-style-type: none"> 可変ベクタテーブルを自動的に生成 指定したソフトウェア割り込み番号の領域に関数の先頭アドレスを設定 関数の最初に全レジスタのスタックへの待避、最後に全レジスタのスタックへの復帰 関数の最後は、reit 命令によるリターン
使用例	<pre>#pragma interrupt intTRB(vect=24) void intTRB(void) { 割り込みプログラム } </pre> → <pre>.rvecgtor 24, _intTRB 先頭アドレス設定 void intTRB(void) { pushm r0, r1, r2, r3, a0, a1, fb 割り込みプログラム popm r0, r1, r2, r3, a0, a1, fb reit } </pre>

8. 電源が入ってからのマイコンの動作

■レジスタバンク切り替えによるレジスタの待避、復帰指定

書式	<ul style="list-style-type: none"> ●固定割り込み処理関数の場合 #pragma interrupt /B 固定割り込み処理関数名 ●可変割り込み処理関数の設定 #pragma interrupt /B 割り込み処理関数名 (vect=ソフトウェア割り込み番号)
内容	<p>関数の最初に全レジスタのスタックへの待避、最後に全レジスタのスタックへの復帰を、レジスタバンクの切り替えで行い、処理時間を短くします。</p> <p>ただし、メインプログラム内でレジスタバンク 0 と 1 の両方を使っている場合、および多重割り込みを許可している場合はレジスタバンク切り替えによるレジスタの待避、復帰はできません。</p>
使用例	<pre> #pragma interrupt /B intTRB(vect=24) void intTRB(void) { 割り込みプログラム } </pre> <p style="text-align: center;">→</p> <pre> .rvecgtor 24, _intTRB 先頭アドレス設定 void intTRB(void) { fset B レジスタバンク 0→1 割り込みプログラム reit レジスタバンクを戻す } </pre>

■多重割り込みを許可する割り込み関数の指定

書式	<ul style="list-style-type: none"> ●固定割り込み処理関数の場合 #pragma interrupt /E 固定割り込み処理関数名 ●可変割り込み処理関数の設定 #pragma interrupt /E 割り込み処理関数名 (vect=ソフトウェア割り込み番号)
内容	<p>関数の入り口で割り込み許可フラグ(I フラグ)をセットし、多重割り込みを許可します。「/B」と併用はできません。</p>
使用例	<pre> #pragma interrupt /E intTRB(vect=24) void intTRB(void) { 割り込みプログラム } </pre> <p style="text-align: center;">→</p> <pre> .rvecgtor 24, _intTRB 先頭アドレス設定 void intTRB(void) { fset I 割り込み許可フラグのセット pushm r0, r1, r2, r3, a0, a1, fb 割り込みプログラム popm r0, r1, r2, r3, a0, a1, fb reit } </pre>

8. 電源が入ってからのマイコンの動作

8.6.10 可変割り込みベクタの設定

```

117 : /*=====*/
118 : /* 可変割り込みベクタの設定 */
119 : /*=====*/
120 : #pragma sectaddress vector,ROMDATA
121 :
122 : /* ここではセクション名の設定のみ行う */
    
```

120 行	可変割り込み処理関数の設定(「#pragma interrupt <u>割り込み処理関数名</u> (vect= <u>ベクタ番号</u>)」)を指定すると、セクション vector に可変ベクタテーブルを生成します。可変割り込みベクタは、ROMに配置します。プログラム以外の固定データなので、固定データ領域 (ROMDATA) として定義します。セクション vector の設定をしないと、生成した可変ベクタテーブルの配置先が分かりません。ここでセクション定義しておきます。
-------	---

8.6.11 スタートアッププログラム (start 関数)

```

141 : /*=====*/
142 : /* スタートアッププログラム */
143 : /*=====*/
144 : #pragma entry start
145 : void start( void )
146 : {
147 :     _isp_ = &_istack_top; /* ISPに割り込みスタックのアドレス設定 */
148 :     _flg_ = 0x0080; /* FLGのU="1" */
149 :     _sp_ = &_stack_top; /* USPにユーザスタックのアドレス設定 */
150 :     _sb_ = 0x0400U; /* SB相対アドレッシングの設定 */
151 :     _intbh_ = (unsigned int*)0x00; /* INTBH = vector (上位) に設定 */
152 :     _intbl_ = &_vector_top; /* INTBL = vector (下位) に設定 */
153 :     initsct(); /* RAMの初期化 */
154 :     _mnext = &heap_area[0]; /* ヒープ領域変数の設定 */
155 :     _msize = (unsigned long) __HEAPSIZE__;
156 :     _fb_ = 0U; /* FB = 0 */
157 :
158 :     main(); /* main関数実行 */
159 :
160 :     while(1);
161 : }
    
```

144 行	#pragma entry <u>関数名</u> で、リセット後、最初に実行する関数を定義します。今回は、start 関数なので、start を定義します。
147 行	ISP(割り込みスタックポインタ)に istack セクションの先頭アドレスである「_istack_top」を設定します。
148 行	FLG(フラグレジスタ)に 0x0080(=0000 0000 1000 0000)を設定します。bit7 は U(スタックポインタ指定フラグ)で、U="0"で ISP(割り込みスタックポインタ)を使用、U="1"で USP(ユーザスタックポインタ)を使用します。ここでは、U="1"にして、以後、USP を使用する設定にしています。

8. 電源が入ってからのマイコンの動作

149 行	SP(スタックポインタ)に stack セクションの先頭アドレスである「_stack_top」を設定します。 SP=USP(ユーザスタックポインタ)のことです。
150 行	SB(スタティックベースレジスタ)に 0x0400 を設定します。
151 行、 152 行	INTBH(割り込みテーブルレジスタ H)に 0x00 を設定します。 INTBL(割り込みテーブルレジスタ L)に vector セクションの先頭アドレスである「_vector_top」を設定 します。 例えば、vector セクションの先頭アドレスが 0xABCDEF なら、INTBH には 0xAB(上位アドレス)を、INTBL には 0xCDEF を設定します。今回の可変割り込みベクタの設定は 0xfed8 番地に設定しているので、 0xAB の部分は必ず 0x00 と分かっています。そのため、INTBH には直接 0x00 を設定しています。
153 行	初期値付きのグローバル変数、初期値無し of グローバル変数を設定する <code>initset</code> 関数を呼び出し、こ れらの変数を初期化します。
154 行	ヒープ領域を管理する <code>*_mnext</code> 変数に、ヒープ領域の先頭アドレスを設定します。
155 行	ヒープ領域を管理する <code>_msize</code> 変数に、ヒープ領域のサイズを設定します。 84行で、 84 : #define __HEAPSIZE__ 0x100 と設定していますので、0x100 が代入されます。
156 行	FB(フレームベースアドレス)に 0 を設定します。
158 行	<code>main</code> 関数を実行します。
160 行	<code>main</code> 関数が終わって処理が戻ってきた場合、このまま終わると <code>start</code> 関数が呼ばれた部分へ戻りま す。といっても、 <code>start</code> 関数から始まっていますので、戻り先が無くプログラムは暴走します。そのため、 無限ループでここで何もせず終わるようにします。

8. 電源が入ってからのマイコンの動作

8.6.12 SB 相対アドレッシング

R8C/35A は、変数の値を参照するとき、SB 相対アドレッシングを使用するとマシン語レベルで命令コードを 1 バイト減らすことができます。

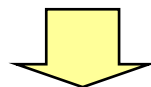
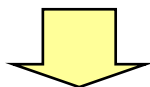
ここでは、プロジェクト「timer2」の timer2.c の timer 関数で比較します。

よく使う変数を、SB 相対アドレッシング使用変数宣言すると、命令コードを 1 バイト減らすことができます。指定方法を、下記に示します。

```
#pragma SBDATA 変数名
```

SB 相対アドレッシング無しとありの場合のメモリ量の比較を、下記に示します。

SB 相対アドレッシング無し(C 言語)	SB 相対アドレッシングあり(C言語)
<pre>unsigned long cnt_rb; void timer(unsigned long timer_set) { cnt_rb = 0; while(cnt_rb < timer_set); }</pre>	<pre>#pragma SBDATA cnt_rb unsigned long cnt_rb; void timer(unsigned long timer_set) { cnt_rb = 0; while(cnt_rb < timer_set); }</pre>



SB 相対アドレッシング無し(アセンブリ言語)	SB 相対アドレッシングあり(アセンブリ言語)
<p>←マシン語→ ←アセンブリ言語→</p> <pre> _timer: 7CF200 enter #00H D90F0000 mov.w #0000H, _cnt_rb D90F0000 mov.w #0000H, _cnt_rb+2 L19: C1BF070000 cmp.w 5+2[FB], _cnt_rb+2 6CFA jltu L19 6908 jgtu M3 C1BF050000 cmp.w 5[FB], _cnt_rb 6CF1 jltu L19 M3: 7DF2 exitd ※0000部分は、配置先アドレスにより変わります</pre>	<p>←マシン語→ ←アセンブリ言語→</p> <pre> _timer: 7CF200 enter #00H D90A00 mov.w #0000H, _cnt_rb D90A00 mov.w #0000H, _cnt_rb+2 L19: C1BA0700 cmp.w 5+2[FB], _cnt_rb+2 6CFB jltu L19 6907 jgtu M3 C1BA0500 cmp.w 5[FB], _cnt_rb 6CF3 jltu L19 M3: 7DF2 exitd ※00部分は、配置先アドレスにより変わります</pre>
29 バイト	25 バイト

※注意点

- SB 相対アドレッシングできる変数は、合計 256 バイトまでです。
- SB 相対アドレッシングで指定した変数のセクションは、「S」が付きます。
例) bss_S0 , data_SEI , data_SE など

※SB 相対アドレッシングのアドレスを変える場合

SB 相対アドレッシングを行うために、次の 3 つの設定を行っています。

- コンパイラに SB レジスタの値を知らせる (startup.c の 43 行、44 行)
- SB レジスタに、番地を設定 (startup.c の 150 行)
- ツールチェーンで、data_SE, bss_SE, data_SO, bss_SO の 4 つのセクションを SB レジスタで設定した番地にする

今回は 400 番地にしており、上記 3 つの設定を全て 400 番地に設定しています。例えば、500 番地にしたければ、上記の項目すべてを 0x500 に変更します。

8.6.13 switch_table セクション

switch-case 文を使うとき、分岐の方法は次の 2 つの方法があります。

- 値の比較を 1 回 1 回行う
- ルネサス統合開発環境がジャンプテーブルを作って行う

どちらにするかは、プログラムサイズや実行スピードを判定して、ルネサス統合開発環境が自動的に決めます。

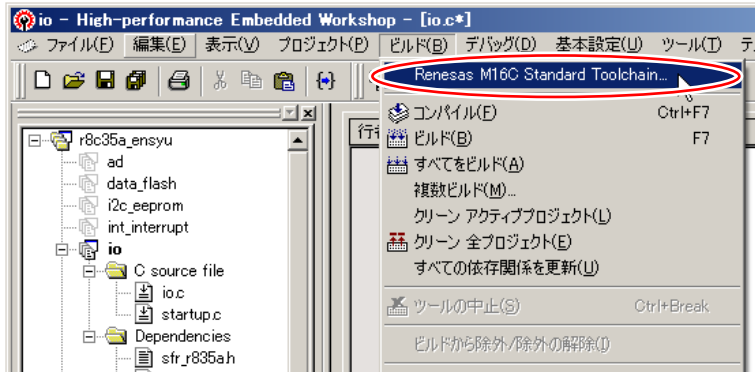
ジャンプテーブルを作るとき、このジャンプテーブルを「switch_table」セクションに割り当てるか、「program」セクションにプログラムと一緒に割り当てるかを設定することができます。

例えば、プログラムサイズが ROM 領域(0x8000~0xffff 番地)の 32KB を超えた場合、ジャンプテーブルをデータフラッシュ領域(0x3000~0x3fff 番地)に置くなどすることができます。

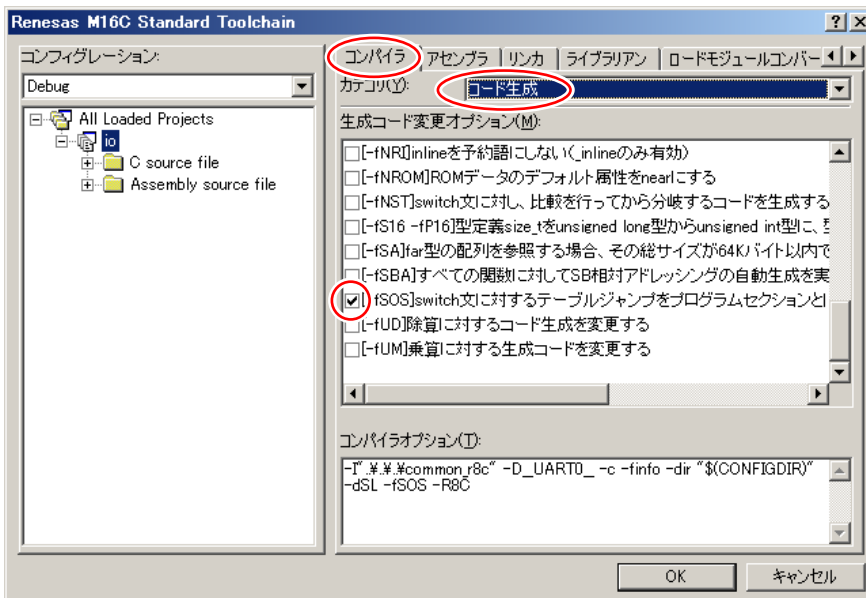
8. 電源が入ってからのマイコンの動作

■ジャンプテーブルを「switch_talbe」セクションに割り当てる設定方法

ルネサス統合開発環境の「ビルド→Renesas M16C Standard Toolchain」(ツールチェーン)を選択します。



「コンパイラ」タブの「カテゴリ:コード生成」を選択します。「生成コード変更オプション」の「[-fSOS]switch 文に対するテーブルジャンプをプログラムセクションとは別セクションに出力する」のチェックを付けます。



例えば、io.c を下記のプログラムに改造して、ビルドしました。C 言語ソースファイルとアセンブリソースファイルを下記に示します。

C 言語ソースファイル	アセンブリソースファイル
<pre>void main(void) { unsigned char d; init(); while(1) { d = p0; switch(d) { case 0: p6 = 0x01; break; } } } (次のページに続く)</pre>	<pre>_main: enter #01H jsr _init L1: mov.w #0001H, R0 jeq L3 mov.b _p0_addr, -1[FB] ; d mov.b -1[FB], ROL ; d mov.b ROL, A0 cmp.b #11H, A0 ; case width check jnc M1 (次のページに続く)</pre>

8. 電源が入ってからのマイコンの動作

<pre> case 1: p6 = 0x02; break; case 2: p6 = 0x04; break; case 3: p6 = 0x08; break; case 4: p6 = 0x10; break; case 5: p6 = 0x20; break; case 6: p6 = 0x40; break; case 7: p6 = 0x80; break; case 8: p6 = ~0x01; break; case 9: p6 = ~0x02; break; case 10: p6 = ~0x04; break; case 11: p6 = ~0x08; break; case 12: p6 = ~0x10; break; case 13: p6 = ~0x20; break; case 14: p6 = ~0x40; break; case 15: p6 = ~0x80; break; default: p6 = 0x00; break; } } } </pre>	<pre> mov.w #10H, A0 M1: sha.w #1, A0 S1: impi.w L41[A0] L41: .word L7-S1&0ffffH ; case 0 のジャンプ先 .word L9-S1&0ffffH ; case 1 のジャンプ先 .word L11-S1&0ffffH ; case 2 のジャンプ先 .word L13-S1&0ffffH ; case 3 のジャンプ先 .word L15-S1&0ffffH ; case 4 のジャンプ先 .word L17-S1&0ffffH ; case 5 のジャンプ先 .word L19-S1&0ffffH ; case 6 のジャンプ先 .word L21-S1&0ffffH ; case 7 のジャンプ先 .word L23-S1&0ffffH ; case 8 のジャンプ先 .word L25-S1&0ffffH ; case 9 のジャンプ先 .word L27-S1&0ffffH ; case 10 のジャンプ先 .word L29-S1&0ffffH ; case 11 のジャンプ先 .word L31-S1&0ffffH ; case 12 のジャンプ先 .word L33-S1&0ffffH ; case 13 のジャンプ先 .word L35-S1&0ffffH ; case 14 のジャンプ先 .word L37-S1&0ffffH ; case 15 のジャンプ先 .word L39-S1&0ffffH ; default のジャンプ先 L7: mov.b #01H, _p6_addr jmp L1 L9: mov.b #02H, _p6_addr jmp L1 L11: mov.b #04H, _p6_addr jmp L1 L13: mov.b #08H, _p6_addr jmp L1 L15: mov.b #10H, _p6_addr jmp L1 L17: mov.b #20H, _p6_addr jmp L1 L19: mov.b #40H, _p6_addr jmp L1 L21: mov.b #80H, _p6_addr jmp L1 L23: mov.b #0feH, _p6_addr jmp L1 L25: mov.b #0fdH, _p6_addr jmp L1 L27: mov.b #0fbH, _p6_addr jmp L1 L29: mov.b #0f7H, _p6_addr jmp L1 L31: mov.b #0efH, _p6_addr jmp L1 L33: mov.b #0dfH, _p6_addr jmp L1 L35: mov.b #0bfH, _p6_addr jmp L1 L37: mov.b #7fH, _p6_addr jmp L1 L39: mov.b #00H, _p6_addr jmp L1 L3: exitd E1: </pre>
---	---

アセンブリソースファイルの□で囲った部分を program セクションにするか switch_table セクションにするかが変わります。

9. I/O ポートの入出力(プロジェクト:io)

9.1 概要

本章では、マイコンのクロックの切り替え、I/O ポートの入出力設定、データの出力、データの入力方法を説明します。

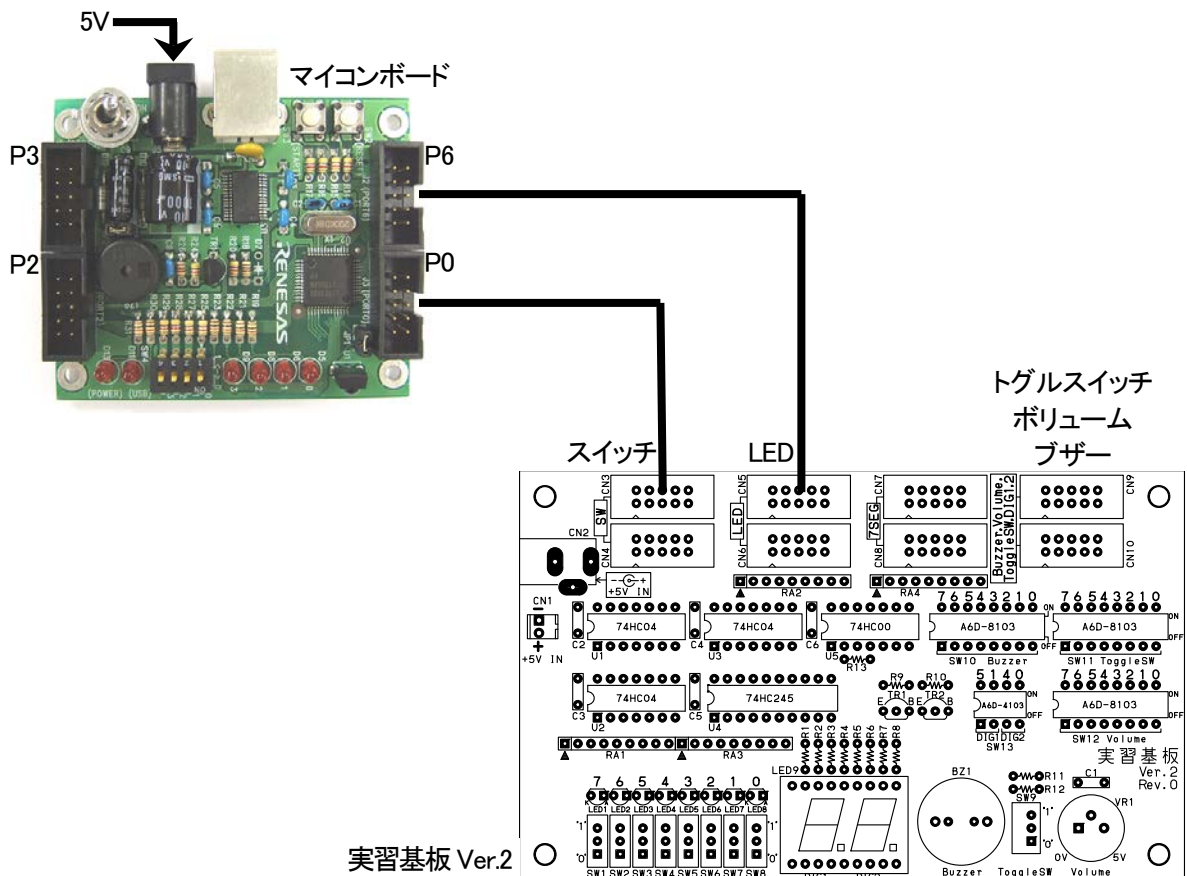
9.2 接続

■使用ポート

マイコンのポート	接続内容
P0 (J3)	実習基板 Ver.2 のディップスイッチ部など、入力機器を接続します。マイコンカーラリーのセンサ基板 Ver.4 を接続すると、チェックすることができます。
P6 (J2)	実習基板 Ver.2 の LED 部など、出力機器を接続します。

■接続例

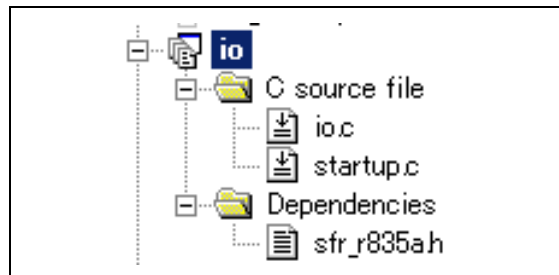
実習基板 Ver.2 を使ったときの接続例を下記に示します。



■操作方法

実習基板 Ver.2 のディップスイッチ(SW3)を ON/OFF すると、それに合わせて LED(LED1~8)が ON/OFF します。入力したデータを出力する、制御の基本中の基本です。

9.3 プロジェクトの構成



	ファイル名	内容
1	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。このファイルは共通で、どのプロジェクトもこのファイルから実行されます。
2	io.c	実際に制御するプログラムが書かれています。R8C/35A の内蔵周辺機能(SFR)の初期化も行います。
3	sfr_r835a.h	R8C/35A マイコンの内蔵周辺機能を制御するためのレジスタ(Special Function Registers)を定義したファイルです。

9.4 プログラム「io.c」

```

1 :  /*****
2 :  /* 対象マイコン R8C/35A
3 :  /* ファイル内容 データの入力、出力
4 :  /* バージョン Ver. 1.20
5 :  /* Date 2010.04.19
6 :  /* Copyright ルネサスマイコンカーラーリ事務局
7 :  /* 日立インターメディアックス株式会社
8 :  /*****
9 :  /*
10 :  入力 : P0_7-P0_0(ディップスイッチなど)
11 :  出力 : P6_7-P6_0(LEDなど)
12 :
13 :  ポート0に入力した状態を、ポート6に出力します。
14 :  ポート0にセンサ基板、ポート6に実習基板のLED部分を接続すれば
15 :  センサ基板のチェックになります。
16 :  */
17 :
18 :  /*=====*/
19 :  /* インクルード
20 :  /*=====*/
21 :  #include "sfr_r835a.h" /* R8C/35A SFRの定義ファイル */
22 :
23 :  /*=====*/
24 :  /* シンボル定義
25 :  /*=====*/
26 :
27 :  /*=====*/
28 :  /* プロトタイプ宣言
29 :  /*=====*/
30 :  void init( void );
31 :

```

9. I/O ポートの入出力(プロジェクト:io)

```

32 : /*****
33 : /* メインプログラム
34 : /*****
35 : void main( void )
36 : {
37 :     unsigned char d;
38 :
39 :     init();                /* 初期化
40 :
41 :     while( 1 ) {
42 :         d = p0;
43 :         p6 = d;
44 :     }
45 : }
46 :
47 : /*****
48 : /* R8C/35A スペシャルファンクションレジスタ(SFR)の初期化
49 : /*****
50 : void init( void )
51 : {
52 :     int i;
53 :
54 :     /* クロックをXINクロック(20MHz)に変更 */
55 :     prc0 = 1;              /* プロテクト解除
56 :     cm13 = 1;              /* P4_6, P4_7をXIN-XOUT端子にする*/
57 :     cm05 = 0;              /* XINクロック発振
58 :     for(i=0; i<50; i++ ); /* 安定するまで少し待つ(約10ms)
59 :     ocd2 = 0;              /* システムクロックをXINにする
60 :     prc0 = 0;              /* プロテクトON
61 :
62 :     /* ポートの入出力設定 */
63 :     prc2 = 1;              /* PD0のプロテクト解除
64 :     pd0 = 0x00;           /* スイッチなど入力
65 :     p1 = 0x0f;            /* 3-0:LEDは消灯
66 :     pd1 = 0xdf;           /* 5:RXD0 4:TXD0 3-0:LED
67 :     pd2 = 0xfe;           /* 0:PushSW
68 :     pd3 = 0xfb;           /* 4: buzzer 2: IR
69 :     pd4 = 0x80;           /* 7: XOUT 6: XIN 5-3: DIP SW 2: VREF
70 :     pd5 = 0x40;           /* 7: DIP SW
71 :     pd6 = 0xff;           /* LEDなど出力
72 : }
73 :
74 : /*****
75 : /* end of file
76 : /*****

```

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

```

21 : #include "sfr_r835a.h"          /* R8C/35A SFR の定義ファイル

```

「#include」はインクルードと読み、外部のファイルを取り込む命令です。取り込むファイルは「< >」、または「 ” ”」で囲い、その中に記述します。違いを下表に示します。

< >	通常の include フォルダから取り込むファイルを探します。 標準ライブラリとしてルネサス統合開発環境が用意しているファイルは、こちらを使います。 例えば、stdio.h , stdlib.h , math.h などです。ちなみに、これらのファイルは、 C:\Program Files\Renesas\HewlettTools\Renesas\nc30wa\545r00\inc30 フォルダにあります。 <u>波線部分</u> は、ルネサス統合開発環境のバージョンにより異なります。
“ ”	まず、カレント・フォルダ(Cプログラムファイルがあるフォルダ)を探して、そこに無ければ通常の include フォルダから取り込むファイルを探します。自分で作ったファイルは、こちらを使います。

ここでは、「sfr_r835a.h」を取り込んでいます。このヘッダファイルは、R8C/35A マイコンの内蔵周辺機能を制御するためのレジスタ(Special Function Registers)を定義したファイルです。ちなみにこのファイルは、次のフォルダにあります。

C:\Workspace\common_r8c35a

9.5.2 init 関数(クロックの選択)

init 関数で、R8C/35A マイコンに内蔵されている機能の初期化を行います。「init」とは、「initialize(イニシャライズ)」の略で、初期化の意味です。io.c では、init 関数内でクロックの選択とI/O ポートの入出力設定を行います。ここでは、クロックの選択について説明します。

```

50 : void init( void )
51 : {
52 :     int i;
53 :
54 :     /* クロックをXINクロック (20MHz)に変更 */
55 :     prc0 = 1;          /* プロテクト解除          */
56 :     cm13 = 1;         /* P4_6, P4_7をXIN-XOUT端子にする*/
57 :     cm05 = 0;        /* XINクロック発振          */
58 :     for(i=0; i<50; i++ ); /* 安定するまで少し待つ(約10ms) */
59 :     ocd2 = 0;        /* システムクロックをXINにする */
60 :     prc0 = 0;        /* プロテクトON            */

```

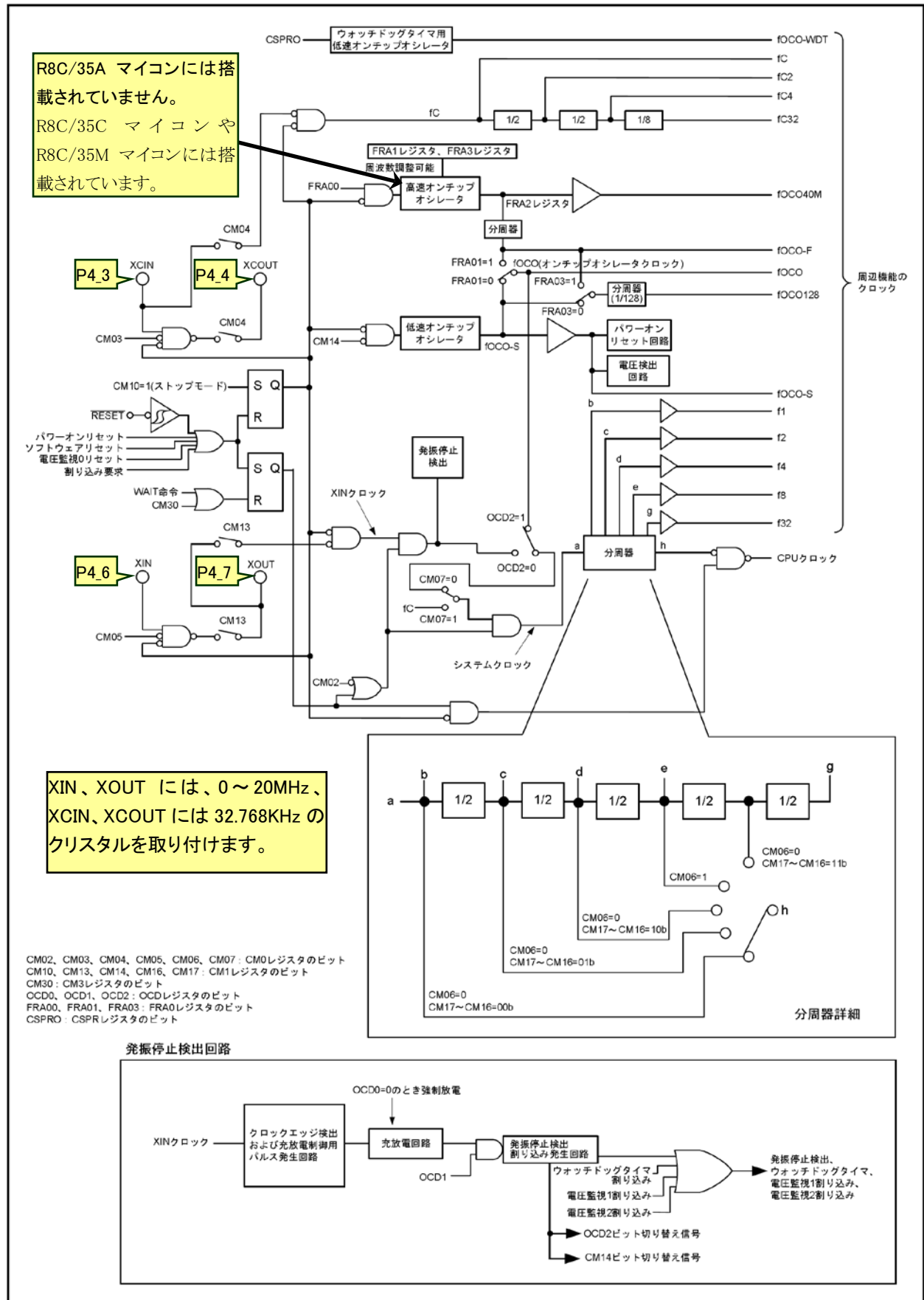
(1) R8C/35A のクロック

R8C/35A は、2 種類のオシレータ(発振器)が内蔵されています。また、2 種類のクリスタルを外付けすることができます。どのクロックを使うかは、プログラムで設定します。下表にまとめます。

	XIN クロック 発信回路	XCIN クロック 発信回路	高速オンチップ オシレータ (内蔵クロック)	低速オンチップ オシレータ (内蔵クロック)	ウォッチドッグタイマ用 低速オンチップオシレータ (内蔵クロック)
接続端子	XIN(P4_6)、 XOUT(P4_7) に接続	XCIN(P4_3)、 XCOUT(P4_4) に接続	マイコン内蔵	マイコン内蔵	マイコン内蔵
クロック周 波数	0~20MHz	32.768kHz	約 40MHz	約 125kHz	約 125kHz
リセット後 の状態	停止	停止	停止	発振	停止、発振はROMの 特定の番地に書き込 む値で決めます
備考	ミニマイコンカー Ver.2 では 20MHz のクリスタルを実 装しています。リ セット後、プログラ ムでこのクロック に切り替えます。	取り付けて いません。	R8C/35A には搭 載されていませ ん(R8C/35C に は搭載されてい ます)。	リセット直後のみ 使用します。すぐ に、XIN に切り替 えます。	使用しません。

(2) ブロック図

R8C/35A のクロック発生回路を下記に示します。図の状態は、リセット直後の状態です。



(3) クロックの設定

R8C/35A マイコンは起動時、低速オンチップオシレータ(約 125kHz)で動作しています。init 関数内のプログラムで XIN クロック発振回路に切り替えて、20MHz で動作させます。レジスタの設定手順を下記に示します。



①プロテクトレジスタ(PRCR:Protect register)の設定(書き込み許可)

R8C/35A マイコンには、プログラムが暴走したときに備え重要なレジスタは簡単に書き替えられないように保護するレジスタがあります。プロテクトレジスタ(PRCR)は、プロテクトがかかっているレジスタの書き込みを禁止、または許可するレジスタです。

設定 bit	上:ビット名 下:シンボル	内容	今回の内容
bit7~4		"0000"を設定	変更なし
bit3	プロテクトビット 3 prc3	OCVREFCR、VCA2、VD1LS、VW0C、VW1C、VW2C レジスタへの書き込み許可 0:書き込み禁止 1:書き込み許可	変更なし
bit2	プロテクトビット 2 prc2	PD0 レジスタへの書き込み許可 0:書き込み禁止 1:書き込み許可(注 1)	変更なし
bit1	プロテクトビット 1 prc1	PM0、PM1 レジスタへの書き込み許可 0:書き込み禁止 1:書き込み許可	変更なし
bit0	プロテクトビット 0 prc0	CM0、CM1、CM3、OCD、FRA0、FRA1、FRA2、FRA3 レジスタへの書き込み許可 0:書き込み禁止 1:書き込み許可	1

注 1. PRC2 ビットは“1”を書いた後、任意の番地に書き込みを実行すると、“0”になります。他のビットは“0”になりませんので、プログラムで“0”にしてください。

これから CM1、CM0、OCD の設定をします。これらのレジスタはプロテクトレジスタ(PRCR)で保護されており、値を変更するには保護を解除します。今回、これらのレジスタを書き込み許可するために bit0 を“1”にします。bit0 のみ OR 演算で“1”にします。下記にプログラムを示します。

```
prcr |= 0x01;
```

もう1つ、設定方法があります。上表の左から2列目に「下:シンボル」とありますが、各レジスタのビットに名前を付けています。プロテクトレジスタ(PRCR)の bit0 は、表より「**prc0**」という名前が付いています。よって下記のようにプログラムすることもできます。今回のプログラムでは、「prc0」として bit0 を“1”にしています。

```
55 :      prc0 = 1;                /* プロテクト解除          */
```


②システムクロック制御レジスタ 1(CM1: System clock control register1)の設定

XIN 端子、XOUT 端子に切り替えます。CM1 レジスタは、PRCR レジスタの PRC0 ビットを“1”(書き込み許可)にした後で書き換えてください。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7,6	CPUクロック分周比選択ビット 1(注 1) bit7:cm17 bit6:cm16	00:分周なしモード 01:2 分周モード 10:4 分周モード 11:16 分周モード	変更 なし
bit5		“1”を設定	変更 なし
bit4	低速オンチップオシレータ発振停止ビット(注 3、4) cm14	0:低速オンチップオシレータ発振 1:低速オンチップオシレータ停止	変更 なし
bit3	ポート XIN-XOUT 切り替えビット(注 5) cm13	0:入出力ポート P4_6、P4_7 1:XIN-XOUT 端子	1
bit2	XCIN-XCOUT 内蔵帰還抵抗選択ビット cm12	0:内蔵帰還抵抗有効 1:内蔵帰還抵抗無効	変更 なし
bit1	XIN-XOUT 内蔵帰還抵抗選択ビット cm11	0:内蔵帰還抵抗有効 1:内蔵帰還抵抗無効	変更 なし
bit0	全クロック停止制御ビット(注 2) cm10	0:クロック発振 1:全クロック停止(ストップモード)	変更 なし

注 1. CM06 ビットが“0”(CM16、CM17 ビット有効)の場合、CM16～CM17 ビットは有効となります。

注 2. CM10 ビットが“1”(ストップモード)の場合、内蔵している帰還抵抗は無効となります。

注 3. CM14 ビットは OCD2 ビットが“0”(XIN クロック選択)のとき、“1”(低速オンチップオシレータ停止)にできます。OCD2 ビットを“1”(オンチップオシレータクロック選択)にすると、CM14 ビットは“0”(低速オンチップオシレータ発振)になります。“1”を書いても変化しません。

注 4. 電圧監視 1 割り込み、電圧監視 2 割り込みを使用する場合(デジタルフィルタを使用する場合)、CM14 ビットを“0”(低速オンチップオシレータ発振)にしてください。

注 5. CM13 ビットはプログラムで一度“1”にすると、“0”にはできません。

ミニマイコンカー Ver.2 は、P4_6、P4_7 の端子に外付けクリスタル(20MHz)が接続されています。この端子を XIN 端子、XOUT 端子に切り替えます。CM1 の bit3 を“1”にします。今回は、ビット指定で CM13 を“1”にします。

56 :	cm13 = 1;	/* P4_6,P4_7 を XIN-XOUT 端子にする*/
------	-----------	---------------------------------

③システムクロック制御レジスタ 0(CM0: System clock control register0)の設定

XIN クロックを発振させます。CM0 レジスタは、PRCR レジスタの PRC0 ビットを“1”(書き込み許可)にした後で書き換えてください。

設定 bit	上:ビット名 下:シンボル	内容	今回の内容
bit7	XIN、XCIN クロック選択ビット (注 7) cm07	0:XIN クロック 1:XCIN クロック	変更なし
bit6	CPUクロック分周比選択ビット 0(注 4) cm06	0:CM1 レジスタの CM16、CM17 ビット有効 1:8 分周モード	変更なし
bit5	XIN クロック(XIN-XOUT)停止 ビット(注 1、3) cm05	0:発振 1:停止(注 2)	0
bit4	ポート、XCIN-XCOUT 切り替 えビット(注 5) cm04	0:入出力ポート P4_3、P4_4 1:XCIN、XCOUT 端子(注 6)	変更なし
bit3	XCIN クロック停止ビット cm03	0:発振 1:停止	変更なし
bit2	ウェイトモード時周辺機能クロ ック停止ビット cm02	0:ウェイトモード時、周辺機能クロックを停止しない 1:ウェイトモード時、周辺機能クロックを停止する	変更なし
bit1,0		“00”を設定	変更なし

注 1. CM05 ビットは高速オンチップオシレータモード、低速オンチップオシレータモードにすると XIN クロックを停止させるビットです。XIN クロックが停止したかどうかの検出には使えません。XIN クロックを停止させる場合、次のようにしてください。

- (1) OCD レジスタの OCD1~OCD0 ビットを“00b”にする。
- (2) OCD2 ビットを“1”(オンチップオシレータクロック選択)にする。

注 2. 外部クロック入力時には、クロック発振バッファだけ停止し、クロック入力は受け付けられます。

注 3. CM05 ビットが“1”(XIN クロック停止) かつ CM1 レジスタの CM13 ビットが“0”(P4_6、P4_7) の場合のみ、P4_6、P4_7 は入出力ポートとして使用できます。

注 4. ストップモードへの移行時、CM06 ビットは“1”(8 分周モード)になります。

注 5. CM04 ビットはプログラムで“1”にできますが、“0”にできません。

注 6. XCIN クロックを使用する場合、CM04 ビットを“1”、PINSR レジスタの XCSEL ビットを“1”にしてください。また、ポート P4_3、P4_4 は入力ポートで、プルアップなしにしてください。

注 7. CM04 ビットを“1”(XCIN-XCOUT 端子)にし、XCIN クロックの発振が安定した後に、CM07 ビットを“0”から“1”(XCIN クロック)にしてください。

XIN クロック(XIN-XOUT)停止ビット(CM05)を“0”にして、XIN クロックを発振させます。CM0 の bit5 を“0”にします。今回は、ビット指定で CM05 を“0”にします。

57 :	cm05 = 0;	/* XIN クロック発振	*/
------	-----------	---------------	----

④ウェイト

③で XIN クロックを発振させました。発振が安定するまで 10ms 程度、待ちます。
 プログラムでは、for 文を使って、「50 回分何もせずに繰り返す」命令を実行します。実測で約 10ms です。

```
58 :      for(i=0; i<50; i++ );          /* 安定するまで少し待つ(約 10ms) */
```

プログラムの実行時間は、机上での計算はできないので、実測する必要があります。
 実測の方法は、プログラム実行前に空いているポートの端子を"1"にして、プログラム実行後にその端子を"0"にします。端子の"1"の時間を、オシロスコープなどで測定すると実行時間が分かります。
 ポート 6 を使って実行時間を計測するプログラム例を、下記に示します。ポート 6 はすべて空き端子とします。

```

p6 = 0x00;          最初に実行 ポート 6 の値をすべて"0"にする
pd6 = 0xff;        最初に実行 ポート 6 の入出力設定 すべて出力

~~~~~

p6 = 0xff;          // ポート 6 の値をすべて"1"にする
for(i=0; i<50; i++ );  /* 安定するまで少し待つ(約 10ms) */
p6 = 0x00;          // ポート 6 の値をすべて"0"にする
    
```

⑤発振停止検出レジスタ(OCD:Oscillation stop detection register)の設定

システムクロックを低速オンチップオシレータ(約 125kHz)から、XIN クロックに切り替えます。OCD レジスタは、PRCR レジスタの PRC0 ビットを“1”(書き込み許可)にした後、書き換えてください。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7~4		”0000”を設定	変更 なし
bit3	クロックモニタビット(注 4、5) ocd3	0:XIN クロック発振 1:XIN クロック停止	変更 なし
bit2	システムクロック選択ビット(注 3) ocd2	0:XIN クロック選択(注 6) 1:オンチップオシレータクロック選択(注 2)	0
bit1	発振停止検出割り込み許可 ビット ocd1	0:禁止(注 1) 1:許可	変更 なし
bit0	発振停止検出有効ビット(注 6) ocd0	0:発振停止検出機能無効(注 1) 1:発振停止検出機能有効	変更 なし

注 1. ストップモード、高速オンチップオシレータモード、低速オンチップオシレータモード(XIN クロック停止) に移行する前に OCD1~OCD0 ビットを“00b”に設定してください。

注 2. OCD2 ビットを“1”(オンチップオシレータクロック選択)にすると、CM14 ビットは“0”(低速オンチップオシレータ発振)になります。

注 3. OCD2 ビットは、OCD1~OCD0 ビットが“11b”のときに XIN クロック発振停止を検出すると、自動的に“1”(オンチップオシレータクロック選択)に切り替わります。また、OCD3 ビットが“1”(XIN クロック停止)のとき、OCD2 ビットに“0”(XIN クロック選択)を書いても変化しません。

注 4. OCD3 ビットは OCD0 ビットが“1”(発振停止検出機能有効)のとき有効です。

注 5. OCD1~OCD0 ビットが“00b”のとき OCD3 ビットは“0”(XIN クロック発振)になり、変化しません。

注 6. 発振停止検出後、XIN クロックが再発振した場合の切り替え手順は、R8C/35A ハードウェアマニュアルの「図 9.10 低速オンチップオシレータから XIN クロックへの切り替え手順」を参照してください。

先の設定で、XIN クロック(XIN-XOUT)停止ビット(CM05)を“0”にして、XIN クロックを発振させました。システムクロックを低速オンチップオシレータ(約 125kHz)から、XIN クロックに切り替えます。ここから、動作が 20MHz になります。OCD の bit2 を“0”にします。今回は、OCD2 を“0”にします。

59 :	ocd2 = 0;	/* システムクロックを XIN にする */
------	-----------	-------------------------

⑥プロテクトレジスタ(PRCR:Protect register)の設定(書き込み禁止)

①で書き込みを許可しました。それを、元に戻します(書き込み禁止にします)。

60 :	prc0 = 0;	/* プロテクト ON */
------	-----------	----------------

9.5.3 init 関数(I/O ポートの入出力設定)

init 関数内でクロックの選択と、I/O ポートの入出力設定を行っています。ここでは I/O ポートの入出力について説明します。プログラムを下記に示します。

```

62 :      /* ポートの入出力設定 */
63 :      prc2 = 1;                /* PD0のプロテクト解除      */
64 :      pd0 = 0x00;             /* スイッチなど入力        */
65 :      p1 = 0x0f;              /* 3-0:LEDは消灯          */
66 :      pd1 = 0xdf;             /* 5:RXD0 4:TXD0 3-0:LED   */
67 :      pd2 = 0xfe;             /* 0:PushSW                */
68 :      pd3 = 0xfb;             /* 4:Buzzer 2:IR           */
69 :      pd4 = 0x80;             /* 7:XOUT 6:XIN 5-3:DIP SW 2:VREF*/
70 :      pd5 = 0x40;             /* 7:DIP SW                 */
71 :      pd6 = 0xff;             /* LEDなど出力             */
72 :  }
```

(1) I/O ポートとは

I/O ポートは、Input/Output Port の略で、直訳すると「**入力や出力を行う港**」という意味です。今回は「入力、出力をまとめて行う場所」というような意味合いです。I/O ポートの入出力設定は、プログラムの初めに行います。

(2) ポートの入出力設定

R8C/35A には、ポート 0～ポート 6 まで 7 本のポートがあります。基本的には 1 ポート 8 ビットですが、8 ビットないポートもあります。それぞれの端子を入力用として使用するか、出力用として使用するか、表にまとめてみます。

考え方を、下記に示します。

- ディップスイッチなど、外部からの信号をマイコンへ入力する場合、端子を「入力」にします。
- LED など、マイコンから外部へ信号を出力する場合、端子を「出力」にします。

9. I/O ポートの入出力(プロジェクト:io)

今回の実習でのポートの一覧を下記に示します。ゴシック体部分は、実習基板 Ver.2 を接続した場合の内容です。

ポート	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
0	ディップ スイッチ 入力	ディップ スイッチ 入力	ディップ スイッチ 入力	ディップ スイッチ 入力	ディップ スイッチ 入力	ディップ スイッチ 入力	ディップ スイッチ 入力	ディップ スイッチ 入力
1	未接続	未接続	RxD0 入力	TxD0 出力	マイコンボード 上の LED3 出力	マイコンボード 上の LED2 出力	マイコンボード 上の LED1 出力	マイコンボード 上の LED0 出力
2	未接続	未接続	未接続	未接続	未接続	未接続	未接続	マイコンボード 上のタクトスイッ チ入力
3	未接続	未接続	未接続	マイコンボ ード上のブザー 出力	未接続	マイコンボ ード上の赤外線 受光 IC 入力	未接続	未接続
4	クリスタル 出力	クリスタル 入力	マイコンボ ード上の SW2 入力	マイコンボ ード上の SW1 入力	マイコンボ ード上の SW0 入力	Vcc 入力	/	/
5	マイコンボ ード 上の SW3 入力	未接続	/	/	/	/	/	/
6	LED 出力	LED 出力	LED 出力	LED 出力	LED 出力	LED 出力	LED 出力	LED 出力

※表の斜線の bit は、端子がない bit です。

※リセット後は、全て入力ポートです。

ポートの入出力方向の設定は、「**ポート Pi 方向レジスタ**」(i は 0~6 の数字が入ります)で行い、レジスタ名は「**pd0~pd6**」の 7 個あります。最後の数字がポート番号です。例えばポート 0 は「PD0」となります。

ポート Pi 方向レジスタの設定方法を下記に示します。

- ①外部へ信号を出力する端子は、“1”を設定する
- ②外部から信号を入力する端子は、“0”を設定する
- ③未接続の端子は、プルアップ抵抗、またはプルダウン抵抗を接続して入力(“0”)にするか、何も接続せず出力(“1”)にする。今回は、後者で設定する
- ④端子が無い場合(表の斜線部分)は、“0”にする

9. I/O ポートの入出力(プロジェクト:io)

①～④の考え方を基に、“1”、“0”で書き換えた表を下記に示します。

ポート	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	16進数
0	0	0	0	0	0	0	0	0	0x00
1	1	1	0	1	1	1	1	1	0xdf
2	1	1	1	1	1	1	1	0	0xfe
3	1	1	1	1	1	0	1	1	0xfb
4	1	0	0	0	0	0	0	0	0x80
5	0	1	0	0	0	0	0	0	0x40
6	1	1	1	1	1	1	1	1	0xff

C 言語は 2 進数表記はできないので、10 進数か 16 進数に変換します。通常は、2 進数を 16 進数に変換する方が簡単なため、16 進数に変換します。

表より、ポート P0 方向レジスタ(PD0)～ポート P6 方向レジスタ(PD6)に設定する値を、下記に示します。

ポート	方向レジスタ名	設定値
0	PD0	0x00
1	PD1	0xdf
2	PD2	0xfe
3	PD3	0xfb
4	PD4	0x80
5	PD5	0x40
6	PD6	0xff

プログラムを下記に示します。

63 :	prc2 = 1;	/* PD0のプロテクト解除	*/
64 :	pd0 = 0x00;	/* スイッチなど入力	*/
65 :	p1 = 0x0f;	/* 3-0:LEDは消灯	*/
66 :	pd1 = 0xdf;	/* 5:RXD0 4:TXD0 3-0:LED	*/
67 :	pd2 = 0xfe;	/* 0:PushSW	*/
68 :	pd3 = 0xfb;	/* 4:Buzzer 2:IR	*/
69 :	pd4 = 0x80;	/* 7:XOUT 6:XIN 5-3:DIP SW 2:VREF*/	
70 :	pd5 = 0x40;	/* 7:DIP SW	*/
71 :	pd6 = 0xff;	/* LEDなど出力	*/

63 行は、次のページで説明します。

※PD0 を設定するときの注意点

R8C/35A マイコンには、プログラムが暴走したときに備え重要なレジスタは簡単に書き換えられないように保護するレジスタがあります。

PD0～PD6 の中で、PD0 だけは保護(プロテクト)されており、保護を解除しないと書き換えることができません。PD0 を書き換える前に、プロテクトレジスタ(PCR)の bit2 を"1"にします。このビットは「sfr_r835a.h(R8C/35A のレジスタ定義ファイル)」で「PRC2」と定義されており、「PRC2」と記述すると PCR の bit2 の意味になります。プログラムを下記に示します。

```
prc2 = 1;    ←PD0の書き換えを許可(PRC2=PCRのbit2の意味です)
pd0 = 0x00; ←その後、書き換える
```

PRC2 を"1"にした後、次に何かの命令を実行すると PRC2 は自動的に"0"(書き換え不可)になります。そのため、**必ず PRC2 を"1"にした次の行で PD0 を設定してください。**プログラムで PRC2 を"0"に戻す必要はありません。

(3) データの出力

出力に設定した端子からデータを出力するときの方法を説明します。

データを出力するときは、「**ポート Pi レジスタ**」(i は 0～6 の数字が入ります)で行い、**レジスタ名は「P0～P6」の 7 個あります。**最後の数字がポート番号です。例えばポート 0 は「P0」となります。

ポート Pi レジスタの設定値は、次のようになります。

- ①端子から"1"(5V)を出力したい場合は、"1"を設定する
- ②端子から"0"(0V)を出力したい場合は、"0"を設定する
- ③入力端子は、"0"、"1"のどちらを設定しても構わないが、"1"だと意味があるように思われるため"0"を設定しておく
- ④端子が無い場合は、"0"にする

例えば、ポート 3 の出力端子から次のように電圧を出力したいとします。

ポート	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
3	5V	0V	0V	0V	5V	入力	0V	5V

5V 部分を"1"に、0V 部分を"0"に、入力部分を"0"にすれば、ポート 3 に設定する値になります。

ポート	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
3	1	0	0	0	1	0	0	1

2 進数で「1000 1001」、16 進数に直すと「0x89」になります。プログラムは、次のように記述します。

```
p3 = 0x89;
```

(4) データの入力

入力に設定した端子からデータを入力するときの方法を説明します。

データを入力するときは、「**ポート Pi レジスタ**」(i は 0~6 の数字が入ります)で行い、**レジスタ名は「P0~P6」の 7 個あります**。最後の数字がポート番号です。例えばポート 0 は「P0」となります(レジスタ名は出力と同じです)。

ポートPiレジスタの値を読み込めば、端子の状態が分かります。端子の状態とポートPiレジスタ関係は、次のようになります。

- ①端子に 5V が入力されている場合は、“1”が読み込まれる
- ②端子に 0V が入力されている場合は、“0”が読み込まれる
- ③出力端子を読み込むと、現在出力している状態が読み込まれる
- ④端子が無い場合は、不定(“0”か“1”か分からない状態)が読み込まれる

例えば、ポート 0 には次の電圧が入力されているとします(例なので今回の演習とは関係ありません)。

ポート	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
0	出力端子 (0V 出力)	出力端子 (0V 出力)	出力端子 (5V 出力)	出力端子 (5V 出力)	5V	5V	0V	0V

ポート0を読み込んだときに入力される値は、5V 部分は“1”が、0V 部分は“0”が読み込まれます。また、出力端子は現在出力している状態が読み込まれます。よって次のようになります。

ポート	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
0	0	0	1	1	1	1	0	0

2 進数で「0011 1100」、16 進数に直すと「0x3c」になります。次のようにプログラムすると、変数 c には、0x3c が代入されます。

```
c = p0; // 変数cには0x3cが代入される
```

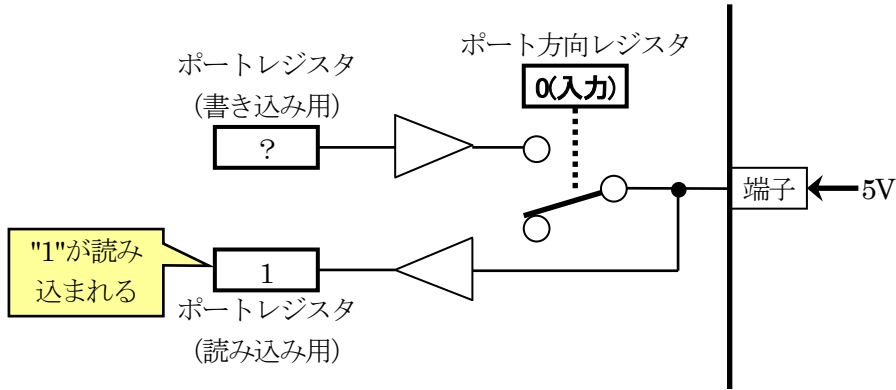
(5) 出力設定にするときのポート方向レジスタレジスタ(PD0~PD6)とポートレジスタ(P0~P6)の設定手順

ポート方向レジスタ(PD0~PD6)とポートレジスタ(P0~P6)を設定するとき、どちらを先に設定すればよいのか説明します。マイコンリセット後の初期値を下記に示します。

レジスタ	リセット後の初期値
ポート方向レジスタ(PD0~PD6)	0x00(入力設定)
ポートレジスタ(P0~P6)	不定(どのような値か分からない)

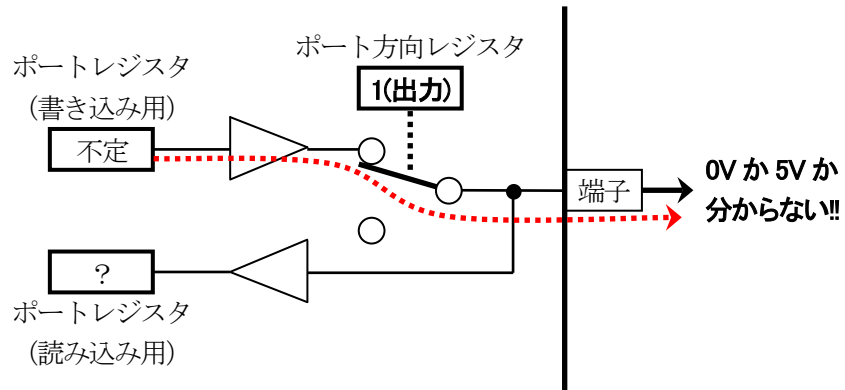
●入力設定の場合

リセット後は入力設定なので、ポートレジスタの値を読み込むと端子に入力されている状態が読み込まれます。

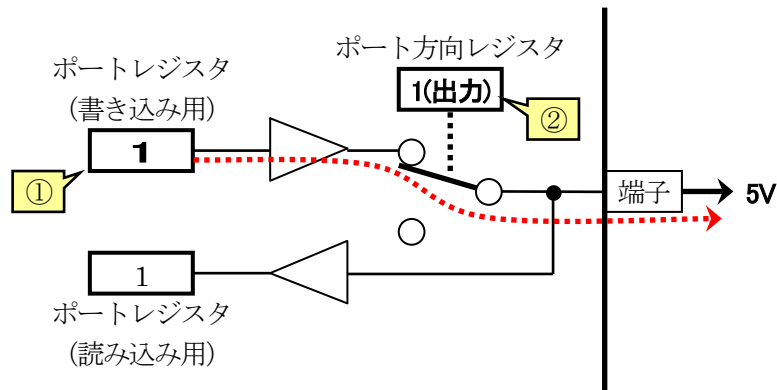


●出力設定の場合

リセット後、ポート方向レジスタを"1"にすると、出力設定になり、ポートレジスタの値が出力されます。ただし、ポートレジスタの値はリセット後、不定です。そのため、出力設定にすると"0"が出力されるか"1"が出力されるか分かりません。



そのため、出力設定するときには、①ポートレジスタに出力したい値を設定してから、②ポート方向レジスタで出力に設定します。



9. I/O ポートの入出力(プロジェクト:io)

例えば、マイコンボード上の LED(ポート 1 の bit3~0)は、“0”で点灯、“1”で消灯します。ポート方向レジスタを設定する前に、ポートレジスタに“1”を設定してからポート方向レジスタを設定します。

p1 = 0x0f;	/* 3-0:LEDは消灯	*/
pd1 = 0xdf;	/* 5:RXD0 4:TXD0 3-0:LED	*/

(6) 入力時の電圧について

入力ポートの場合、“1”と判断する電圧(下表の V_{IH})と“0”と判断する電圧(下表の V_{IL})を、下表に示します。

記号	項目		測定条件	規格値			単位		
				最小	標準	最大			
Vcc/AVcc	電源電圧			1.8	—	5.5	V		
Vss/AVss	電源電圧			—	0	—	V		
V _{IH}	“H” 入力電圧	CMOS入力以外			0.8Vcc	—	Vcc	V	
		CMOS入力	入力レベル切り替え機能(I/Oポート)	入力レベル選択: 0.35Vcc	4.0V ≤ Vcc ≤ 5.5V	0.5Vcc	—	Vcc	V
					2.7V ≤ Vcc < 4.0V	0.55Vcc	—	Vcc	V
					1.8V ≤ Vcc < 2.7V	0.65Vcc	—	Vcc	V
				入力レベル選択: 0.5Vcc	4.0V ≤ Vcc ≤ 5.5V	0.65Vcc	—	Vcc	V
					2.7V ≤ Vcc < 4.0V	0.7Vcc	—	Vcc	V
					1.8V ≤ Vcc < 2.7V	0.8Vcc	—	Vcc	V
			入力レベル選択: 0.7Vcc	4.0V ≤ Vcc ≤ 5.5V	0.85Vcc	—	Vcc	V	
				2.7V ≤ Vcc < 4.0V	0.85Vcc	—	Vcc	V	
				1.8V ≤ Vcc < 2.7V	0.85Vcc	—	Vcc	V	
V _{IL}	“L” 入力電圧	CMOS入力以外			0	—	0.2Vcc	V	
		CMOS入力	入力レベル切り替え機能(I/Oポート)	入力レベル選択: 0.35Vcc	4.0V ≤ Vcc ≤ 5.5V	0	—	0.2Vcc	V
					2.7V ≤ Vcc < 4.0V	0	—	0.2Vcc	V
					1.8V ≤ Vcc < 2.7V	0	—	0.2Vcc	V
				入力レベル選択: 0.5Vcc	4.0V ≤ Vcc ≤ 5.5V	0	—	0.4Vcc	V
					2.7V ≤ Vcc < 4.0V	0	—	0.3Vcc	V
					1.8V ≤ Vcc < 2.7V	0	—	0.2Vcc	V
			入力レベル選択: 0.7Vcc	4.0V ≤ Vcc ≤ 5.5V	0	—	0.55Vcc	V	
				2.7V ≤ Vcc < 4.0V	0	—	0.45Vcc	V	
				1.8V ≤ Vcc < 2.7V	0	—	0.35Vcc	V	

電源電圧(Vcc)は 5.0V、入力レベル選択は 0.5Vcc です。よって、下記のように判断します。

“1”と判断する電圧は表より

$$0.65V_{cc} = 0.65 \times 5.0 = 3.25V \quad \therefore 3.25V \sim 5.0V \text{ の電圧なら“1”と判断}$$

“0”と判断する電圧は表より

$$0.4V_{cc} = 0.4 \times 5.0 = 2.0V \quad \therefore 0 \sim 2.0V \text{ の電圧なら“0”と判断}$$

では、上記の範囲に入っていない 2.0~3.25V の電圧が入力された場合はどうなるのでしょうか。この場合は、“0”と判断するか“1”と判断するか分かりません。ある時は“0”と判断し、ある時は“1”と判断するかもしれません。そのため、この範囲の電圧は加えないようにします。

(7) 出力時の電圧、電流について

出力ポートの場合、流せる電流を下表に示します。

I _{OH(sum)}	“H” 尖頭総出力電流	全端子のI _{OH(peak)} の総和	—	—	— 160	mA
I _{OH(sum)}	“H” 平均総出力電流	全端子のI _{OH(avg)} の総和	—	—	— 80	mA
I _{OH(peak)}	“H” 尖頭出力電流	駆動能力Low時	—	—	— 10	mA
		駆動能力High時	—	—	— 40	mA
I _{OH(avg)}	“H” 平均出力電流	駆動能力Low時	—	—	— 5	mA
		駆動能力High時	—	—	— 20	mA
I _{OL(sum)}	“L” 尖頭総出力電流	全端子のI _{OL(peak)} の総和	—	—	160	mA
I _{OL(sum)}	“L” 平均総出力電流	全端子のI _{OL(avg)} の総和	—	—	80	mA
I _{OL(peak)}	“L” 尖頭出力電流	駆動能力Low時	—	—	10	mA
		駆動能力High時	—	—	40	mA
I _{OL(avg)}	“L” 平均出力電流	駆動能力Low時	—	—	5	mA
		駆動能力High時	—	—	20	mA
f _(XIN)	XINクロック入力発振周波数	2.7V ≤ V _{cc} ≤ 5.5V	—	—	20	MHz
		1.8V ≤ V _{cc} < 2.7V	—	—	5	MHz
f _(XCIN)	XCINクロック入力発振周波数	1.8V ≤ V _{cc} ≤ 5.5V	—	32.768	50	kHz
f _{OCO40M}	タイマRC、タイマRDのカウントソース(注3)	2.7V ≤ V _{cc} ≤ 5.5V	32	—	40	MHz
f _{OCO-F}	f _{OCO-F} 周波数	2.7V ≤ V _{cc} ≤ 5.5V	—	—	20	MHz
		1.8V ≤ V _{cc} < 2.7V	—	—	5	MHz
—	システムクロック周波数	2.7V ≤ V _{cc} ≤ 5.5V	—	—	20	MHz
		1.8V ≤ V _{cc} < 2.7V	—	—	5	MHz
f _(BCLK)	CPUクロック周波数	2.7V ≤ V _{cc} ≤ 5.5V	—	—	20	MHz
		1.8V ≤ V _{cc} < 2.7V	—	—	5	MHz

注1. 指定のない場合は、V_{cc} = 1.8V ~ 5.5V、Topr = -20°C ~ 85°C(Nバージョン)/-40°C ~ 85°C(Dバージョン)です。

注2. 平均出力電流は100 msの期間内での平均値です。

注3. f_{OCO40M}はV_{cc} = 2.7V ~ 5.5Vの範囲で、タイマRC、タイマRDのカウントソースとして使用することができます。

“1”(5V)にした端子から流すことのできる電流は、5mA です(I_{OH(avg)})。また、マイコン全体では、“1”(5V)にした端子から流すことのできる電流は 80mA です(I_{OH(sum)})。

“0”(0V)にした端子から流すことのできる電流は、5mA です(I_{OL(avg)})。また、マイコン全体では、“0”(0V)にした端子から流すことのできる電流は 80mA です(I_{OL(sum)})。

9. I/O ポートの入出力(プロジェクト:io)

出力ポートの場合、“1”と“0”の実際の出力電圧を下表に示します。

記号	項目	測定条件	規格値			単位
			最小	標準	最大	
VOH	“H” 出力電圧	駆動能力High Vcc = 5V IOH = -20mA	Vcc - 2.0	—	Vcc	V
		駆動能力Low Vcc = 5V IOH = -5mA	Vcc - 2.0	—	Vcc	V
VOL	“L” 出力電圧	駆動能力High Vcc = 5V IOl = 20mA	—	—	2.0	V
		駆動能力Low Vcc = 5V IOl = 5mA	—	—	2.0	V
VT+-VT-	ヒステリシス	INT0、INT1、INT2、INT3、INT4、KI0、KI1、KI2、KI3、TRAIO、TRBO、TRCIOA、TRCIOB、TRCIOC、TRCIOD、TRDIOA0、TRDIOB0、TRDIOC0、TRDIOD0、TRDIOA1、TRDIOB1、TRDIOC1、TRDIOD1、TRCTRG、TRCLK、ADTRG、RXD0、RXD1、RXD2、CLK0、CLK1、CLK2、SSI、SCL、SDA、SSO	0.1	1.2	—	V
		RESET	0.1	1.2	—	V
IiH	“H” 入力電流	VI = 5V	—	—	5.0	μA
IiL	“L” 入力電流	VI = 0V	—	—	-5.0	μA
RPULLUP	プルアップ抵抗	VI = 0V	25	50	100	kΩ
RfXIN	帰還抵抗	XIN	—	0.3	—	MΩ
RfXCIN	帰還抵抗	XCIN	—	8	—	MΩ
VRAM	RAM保持電圧	ストップモード時	1.8	—	—	V

注1. 指定のない場合は、 $4.2V \leq V_{cc} \leq 5.5V$ 、 $T_{opr} = -20^{\circ}C \sim 85^{\circ}C$ (Nバージョン)/ $-40^{\circ}C \sim 85^{\circ}C$ (Dバージョン)、 $f(XIN) = 20MHz$ です。

“1”出力は、 $(V_{cc}-2.0) \sim V_{cc}$ の電圧が出力されます。 V_{cc} は 5.0V なので、3.0~5.0V が出力されます。マイコンに接続されている回路は、この電圧で動作するように回路設計する必要があります。

“0”出力は、0~2V が出力されます。マイコンに接続されている回路は、この電圧で動作するように回路設計する必要があります。

9.5.4 main 関数

```

35 : void main( void )
36 : {
37 :     unsigned char d;
38 :
39 :     init();                /* 初期化                */
40 :
41 :     while( 1 ) {
42 :         d = p0;
43 :         p6 = d;
44 :     }
45 : }
    
```

9. I/O ポートの入出力(プロジェクト:io)

「io.c」は、main 関数から実行されます。

39 行	init 関数を実行します。
41 行	while()は、カッコの中が成り立ったなら(真なら)中カッコ(波カッコ)の中を実行、成り立たないなら(偽なら)中カッコを抜ける命令です。今回は()の中が「1」です。これは、C 言語では成り立つという意味になり、常に成り立つ状態です。41 行目の最後の「{」から、44 行目の「}」の間のプログラムが無限に繰り返されます。要は、42 行、43 行が無限に繰り返されます。
42 行	変数 d にポート 0 の状態を読み込み、保存します。
43 行	ポート d の値を、ポート 6 に出力します。

9.6 ビット操作

9.6.1 特定のビットを"0"にする(マスク処理)

特定のビットを強制的に"0"にするには、AND 演算を行います。

例えば、ポート 0 の bit7~4 はそのままにして、bit3~0 を強制的に"0"にしたいとします。表にまとめます。

bit	7	6	5	4	3	2	1	0
状態	そのまま	そのまま	そのまま	そのまま	"0"にする	"0"にする	"0"にする	"0"にする

そのままの部分に「1」に、「0」にする部分を「0」に置き換えます。

bit	7	6	5	4	3	2	1	0
変換後	1	1	1	1	0	0	0	0
16 進数	f				0			

ポート 0 と変換後の値を AND 演算することにより、bit3~0 の値を強制的に"0"にすることができます。AND 演算の記述は C 言語では"&"(アンパサンド)を用います。C 言語で記述すると次のようになります。

```
d = p0 & 0xf0;
p6 = d;          ←結果をポート6へ出力
```

例えば、ポート 0 の値が 0x55 なら、結果(変数 d の値)は次のようになります。

bit	7	6	5	4	3	2	1	0
P0 の値	0	1	0	1	0	1	0	1
AND 値	1	1	1	1	0	0	0	0
結果	0	1	0	1	0	0	0	0

AND 値が"1"なら、結果は P0 の値になる

AND 値が"0"なら、結果は"0"になる

■マスクについて

ポートの特定のビットが"1"か"0"かチェックしたいとします。**1 ポートの単位は 8bit のため、1bit だけチェックすることはできません**(ビットフィールドという方法を使えばできますが、ここでは無しにします)。**必ず 8bit まとめてのチェックとなります。**

例えば、ポート 0 にディップスイッチ基板が繋がっていて、bit7 が"1"かどうかチェックしたい場合、次のようにすればいいように思えます。

```
if( p0 == 0x80 ) {
    /* bit 7 が "1" ならこの中を実行 */
}
```

しかし、ポート 0 の bit6~0 がどのような値になっているか分かりません。例えば、bit7 が"1"、bit0 も"1"ならポート 0 の値は 2 進数で"1000 0001"、16 進数に直すと 0x81 となります。そのため、次のようになります。

```
if( p0 == 0x80 ) {
    /* bit 7 は "1" だが、この中を実行しない!! */
}
```

bit7 は"1"ですが、bit0 も"1"なので、式は成り立たずカッコの中を実行しません。このようなときは、**チェック不要なビットを AND 演算で強制的に"0"にします。これを「マスク」(覆い隠す)といいます。マスクは、AND 処理でチェックに不要なビットを強制的に"0"にすることです。AND 処理する値をマスク値といいます。**

今回、チェックしたい bit は 7、その他はチェックしません。表にまとめます。

bit	7	6	5	4	3	2	1	0
チェックする?	する	しない	しない	しない	しない	しない	しない	しない

マスク値は、チェックする bit を"1"に、チェックしない bit を"0"にするだけです。

bit	7	6	5	4	3	2	1	0
マスク値	1	0	0	0	0	0	0	0

プログラムでは、チェックする値(今回はポート 0)とマスク値を AND 演算した結果をチェックします。結果は、bit6~0 が"0"であることが分かっているので、bit7 だけ"1"か"0"かをチェックすればいいことになります。

bit7 が"0"かどうか調べるなら、プログラムは次のようになります。

```
if( (p0 & 0x80) == 0x00 ) {
    /* bit 7 が "0" ならこの中を実行 */
}
```

P0 が 0x71 なら、次のような計算になります。

$$\begin{array}{r}
 P0 \quad 0111\ 0001 \\
 \underline{\text{マスク値 } 1000\ 0000 \text{ (AND)}} \\
 \text{結果} \quad 0000\ 0000
 \end{array}$$

bit7 は"0"なので式は成り立ち、カッコの中が実行されます。

9. I/O ポートの入出力(プロジェクト:io)

bit7 が"1"かどうか調べるなら、プログラムは次のようになります。

```
if( (p0 & 0x80) == 0x80 ) {
    /* bit 7 が "1" ならこの中を実行 */
}
```

P0 が 0xaf なら、次のような計算になります。

P0	1010 1111	
マスク値	1000 0000	(AND)
結果	1000 0000	

bit7 は"1"なので式は成り立ち、カッコの中が実行されます。

9.6.2 特定のビットを"1"にする

特定のビットを強制的に"1"にするには、OR 演算を行います。

例えば、ポート0 の bit7~4 はそのままにして、bit3~0 を強制的に"1"にしたいとします。表にまとめます。

bit	7	6	5	4	3	2	1	0
状態	そのまま	そのまま	そのまま	そのまま	"1"にする	"1"にする	"1"にする	"1"にする

そのままの部分を「0」に、「1」にする部分を「1」に置き換えます。

bit	7	6	5	4	3	2	1	0
変換後	0	0	0	0	1	1	1	1
16 進数	0				f			

ポート0 と変換後の値を OR 演算することにより、bit3~0 の値を強制的に"1"にすることができます。OR 演算の記述は C 言語では"|" (縦線) を用います。C 言語で記述すると次のようになります。

```
d = p0 | 0x0f;
p6 = d;          ←結果をポート6へ出力
```

例えば、ポート0 の値が 0x55 なら、結果(変数 d の値)は次のようになります。

bit	7	6	5	4	3	2	1	0
P0 の値	0	1	0	1	0	1	0	1
OR 値	0	0	0	0	1	1	1	1
結果	0	1	0	1	1	1	1	1

OR 値が"0"なら、結果は P0 の値になる

OR 値が"1"なら、結果は"1"になる

9.6.3 特定のビットを反転する

特定のビットを反転する("0"なら"1"に、"1"なら"0"にする)には、XOR(exclusive or、排他的論理和)演算を行います。

例えば、ポート0のbit7~4はそのままにして、bit3~0を反転したいとします。表にまとめます。

bit	7	6	5	4	3	2	1	0
状態	そのまま	そのまま	そのまま	そのまま	反転する	反転する	反転する	反転する

そのままの部分で「0」に、反転する部分を「1」に置き換えます。

bit	7	6	5	4	3	2	1	0
変換後	0	0	0	0	1	1	1	1
16進数	0				f			

ポート0と変換後の値をXOR演算することにより、bit3~0の値を反転することができます。XOR演算の記述はC言語では"^(アクサンシルコンプレックス)を用います。C言語で記述すると次のようになります。

```

d = p0 ^ 0x0f;
p6 = d;           ←結果をポート6へ出力
    
```

例えば、ポート0の値が0x55なら、結果(変数dの値)は次のようになります。

bit	7	6	5	4	3	2	1	0
P0の値	0	1	0	1	0	1	0	1
XOR値	0	0	0	0	1	1	1	1
結果	0	1	0	1	1	0	1	0

XOR値が"0"なら、結果はP0の値になる

XOR値が"1"なら、結果は反転する

9.6.4 全ビット反転する(NOT演算)

全ビット反転するなら、C言語では"~(チルダ)を用います。C言語で記述すると次のようになります。

```

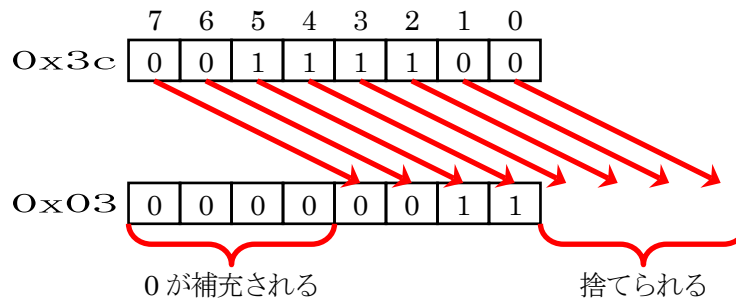
d = ~p0;
p6 = d;           ←結果をポート6へ出力
    
```

反転したい値の先頭に"~"を付けます。

9.6.5 ビットシフトする

ビットシフトとは、各ビットを右や左にずらすことです。

例えば、ポート0を4ビット右にシフトするとします。ポート0が0x3c なら、結果は0x03 になります。

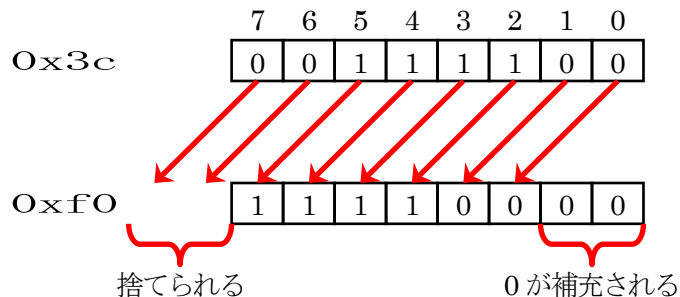


右シフトの記述はC言語では" \gg "を用います。C言語で記述すると次のようになります。

```

d = p0 >> 4;
    └─シフトする数
p6 = d;      ←結果をポート6へ出力
    
```

例えば、ポート0を2ビット左にシフトするとします。ポート0が0x3c なら、結果は0xf0 になります。



左シフトの記述はC言語では" \ll "を用います。C言語で記述すると次のようになります。

```

d = p0 << 2;
    └─シフトする数
p6 = d;      ←結果をポート6へ出力
    
```

9.7 ビット単位でデータを入力、出力する

R8C/35A マイコンの内蔵周辺機能を制御するためのレジスタ(Special Function Registers)を定義したファイル「sfr_r835a.h」は、ポート Pi レジスタ(P0～P6)をビット単位で設定できるように定義しています。

シンボル名は、下記のように定義されています。

`p x _ y` : ポート x の bit y

■ポートにデータを出力する

例えば、ポート 2 の bit0 に"1"を設定したい場合のプログラムを下記に示します。

```
p2_0 = 1;
```

■ポートからデータを入力する

例えば、ポート 0 の bit2 の値を変数 c に代入するプログラムを書きに示します。

```
c = p0_2;
```

例えば、ポート 0 の bit5 の状態を確認する場合のプログラムを書きに示します。

```
if( p0_5 == 1 ) {  
    p0_5 が"1"ならこの部分を実行  
} else {  
    p0_5 が"0"なら、この部分を実行  
}
```

9.8 演習

- (1) ポート 0 に LED 部、ポート 6 にディップスイッチ部を接続して、ディップスイッチの状態を LED へ出力しなさい。
- (2) (1)のとき、入力した値の bit7~4 を強制的に"0"にして、LED へ出力しなさい。
- (3) (1)のとき、入力した値の bit3~0 を強制的に"1"にして、LED へ出力しなさい。
- (4) (1)のとき、ディップスイッチの値をすべて反転して LED へ出力しなさい。
- (5) (1)のとき、ディップスイッチの bit4 が"1"なら LED に(1111 0000)を、bit4 が"0"なら LED に(0000 1111)を出力しなさい。ただし、ディップスイッチの bit4 以外は"1"か"0"か分からないものとする。

10. I/O ポートの入出力 2(プロジェクト:io2)

10.1 概要

本章では、マイコンボード上にあるディップスイッチ(4bit)と LED(4bit)を使って、I/O ポートの入出力を行う方法を説明します。ポートが分かれているため、ビット操作を行って入出力を行います。

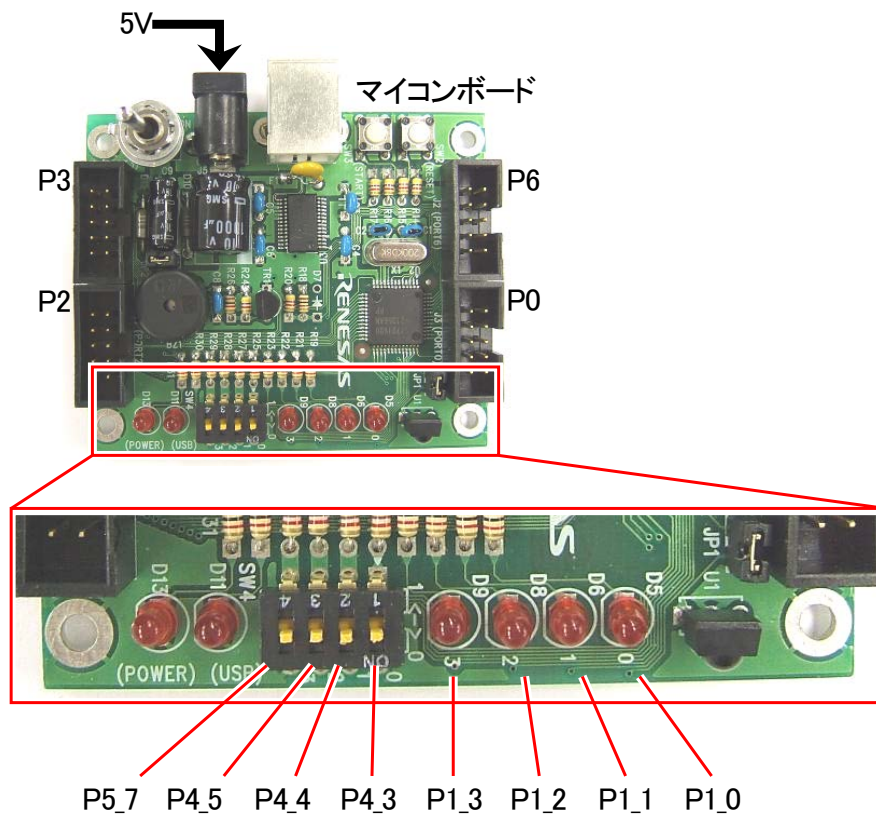
10.2 接続

■使用ポート

マイコンのポート	接続内容
P5_7、P4_5、P4_4、P4_3	マイコンボード上のディップスイッチです。
P1_3、P1_2、P1_1、P1_0	マイコンボード上の LED です。

■接続例

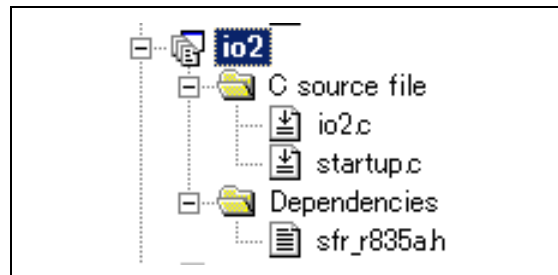
マイコンボードだけで実習できます。



■操作方法

マイコンボードのディップスイッチ(SW4)を ON/OFF すると、それに合わせて LED(D9,D8,D6,D5)が点灯／消灯します。

10.3 プロジェクトの構成



	ファイル名	内容
1	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。このファイルは共通で、どのプロジェクトもこのファイルから実行されます。
2	io2.c	実際に制御するプログラムが書かれています。R8C/35A の内蔵周辺機能(SFR)の初期化も行います。
3	sfr_r835a.h	R8C/35A マイコンの内蔵周辺機能を制御するためのレジスタ (Special Function Registers)を定義したファイルです。

10.4 プログラム「io2.c」

```

1 : /*****
2 : /* 対象マイコン R8C/35A */
3 : /* ファイル内容 データの入力、出力 */
4 : /* バージョン Ver. 1.20 */
5 : /* Date 2010.04.19 */
6 : /* Copyright ルネサスマイコンカーラリー事務局 */
7 : /* 日立インターメディアックス株式会社 */
8 : *****/
9 : /*
10 : 入力：マイコンボードのディップスイッチ(4bit)
11 : 出力：マイコンボードのLED(4bit)
12 :
13 : マイコンボードのディップスイッチ(4bit)から入力した状態を、
14 : マイコンボードのLED(4bit)に出力します。
15 : */
16 :
17 : /*=====*/
18 : /* インクルード */
19 : /*=====*/
20 : #include "sfr_r835a.h" /* R8C/35A SFRの定義ファイル */
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 : unsigned char dipsw_get( void );
31 : void led_out( unsigned char led );
32 :

```

10. I/O ポートの入出力 2(プロジェクト:io2)

```

33 : /*****/
34 : /* メインプログラム */
35 : /*****/
36 : void main( void )
37 : {
38 :     unsigned char d;
39 :
40 :     init();                /* 初期化 */
41 :
42 :     while( 1 ) {
43 :         d = dipsw_get();
44 :         led_out( d );
45 :     }
46 : }
47 :
48 : /*****/
49 : /* R8C/35A スペシャルファンクションレジスタ(SFR)の初期化 */
50 : /*****/
51 : void init( void )
52 : {
53 :     int i;
54 :
55 :     /* クロックをXINクロック(20MHz)に変更 */
56 :     prc0 = 1;                /* プロテクト解除 */
57 :     cm13 = 1;                /* P4_6, P4_7をXIN-XOUT端子にする*/
58 :     cm05 = 0;                /* XINクロック発振 */
59 :     for(i=0; i<50; i++ );    /* 安定するまで少し待つ(約10ms) */
60 :     ocd2 = 0;                /* システムクロックをXINにする */
61 :     prc0 = 0;                /* プロテクトON */
62 :
63 :     /* ポートの入出力設定 */
64 :     prc2 = 1;                /* PD0のプロテクト解除 */
65 :     pd0 = 0xe0;              /* 7-5:LED 4:MicroSW 3-0:Sensor */
66 :     p1 = 0x0f;                /* 3-0:LEDは消灯 */
67 :     pd1 = 0xdf;              /* 5:RXD0 4:TXD0 3-0:LED */
68 :     pd2 = 0xfe;              /* 0:PushSW */
69 :     pd3 = 0xfb;              /* 4: buzzer 2: IR */
70 :     pd4 = 0x83;              /* 7:XOUT 6:XIN 5-3:DIP SW 2:VREF*/
71 :     pd5 = 0x40;              /* 7:DIP SW */
72 :     pd6 = 0xff;
73 : }
74 :
75 : /*****/
76 : /* ディップスイッチ値読み込み */
77 : /* 戻り値 スイッチ値 0~15 */
78 : /*****/
79 : unsigned char dipsw_get( void )
80 : {
81 :     unsigned char sw, sw1, sw2;
82 :
83 :     sw1 = (p5>>4) & 0x08;    /* ディップスイッチ読み込み3 */
84 :     sw2 = (p4>>3) & 0x07;    /* ディップスイッチ読み込み2,1,0*/
85 :     sw = sw1 | sw2;          /* P5とP4の値を合わせる */
86 :
87 :     return sw;
88 : }
89 :
90 : /*****/
91 : /* マイコン部のLED出力 */
92 : /* 引数 スイッチ値 0~15 */
93 : /*****/
94 : void led_out( unsigned char led )
95 : {
96 :     unsigned char data;
97 :
98 :     led = ~led;
99 :     led &= 0x0f;
100 :     data = p1 & 0xf0;
101 :     p1 = data | led;
102 : }
103 :
104 : /*****/
105 : /* end of file */
106 : /*****/

```


10.5 プログラムの解説

10.5.1 dipsw_get 関数

dipsw_get 関数は、マイコンボードの 4bit のディップスイッチの値を読み込む関数です。

```

75 :  /*****/
76 :  /* ディップスイッチ値読み込み */
77 :  /* 戻り値 スイッチ値 0~15 */
78 :  /*****/
79 :  unsigned char dipsw_get( void )
80 :  {
81 :      unsigned char sw, sw1, sw2;
82 :
83 :      sw1 = (p5>>4) & 0x08;          /* ディップスイッチ読み込み3 */
84 :      sw2 = (p4>>3) & 0x07;          /* ディップスイッチ読み込み2, 1, 0*/
85 :      sw = sw1 | sw2;                /* P5とP4の値を合わせる */
86 :
87 :      return sw;
88 :  }
```

マイコンボードには 4bit のディップスイッチが搭載されています。左から順にマイコンの P5_7、P4_5、P4_4、P4_3 の各 bit に接続されています。

ポートがばらばらなので、ビット演算を使って、次の図のようにビットを移動させて bit3~0 になるようにします。これを行うのが、dipsw_get 関数です。dipsw_get 関数を呼ぶと、0~15 の値が返ってきます。ディップスイッチを上にする"1"、下(ON と書いてある側)にすると"0"です。

まず、変数 sw1 にポート 5(P5)の値を読み込みます。

```

83 :      sw1 = (p5>>4) & 0x08;          /* ディップスイッチ読み込み3 */
                ①      ②
```

① ポート 5(P5)の値を 4bit 右シフトします。

② ①の値を 0x08 でマスクします。0x08 は、「0000 1000」なので bit3 のみ有効に、他は強制的に"0"にします。

次に、変数 sw2 にポート 4(P4)の値を読み込みます。

```

84 :      sw2 = (p4>>3) & 0x07;          /* ディップスイッチ読み込み2, 1, 0*/
                ①      ②
```

① ポート 4(P4)の値を 3bit 右シフトします。

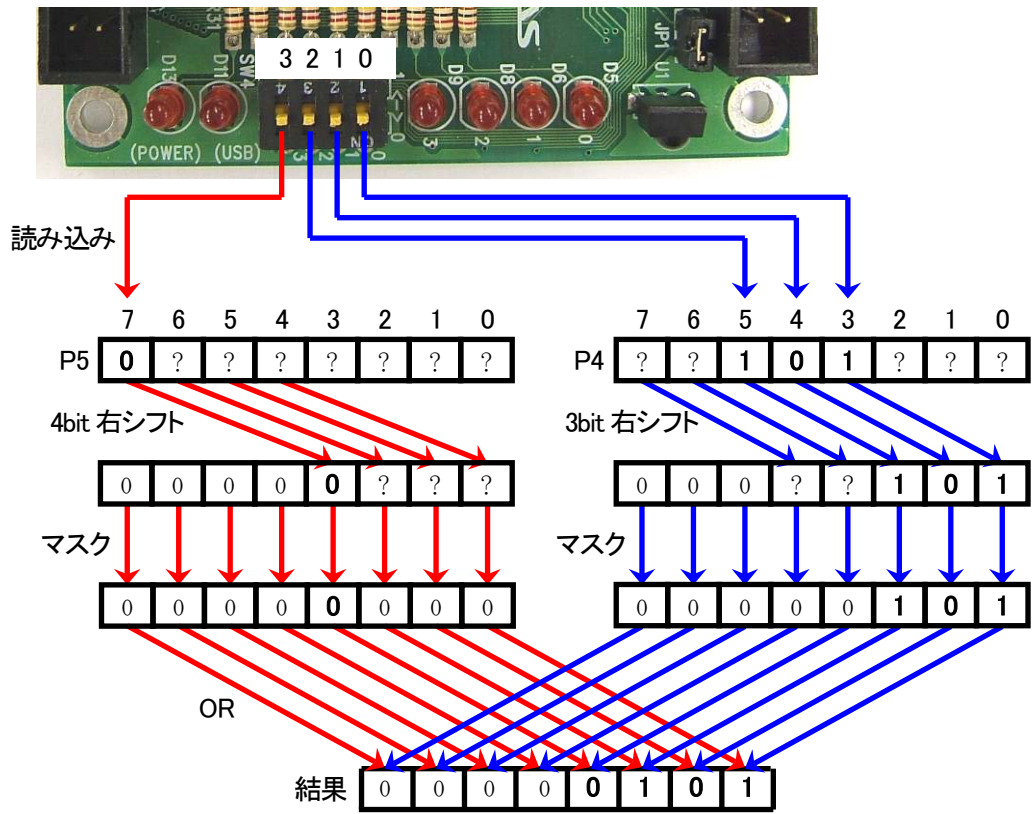
② ①の値を 0x07 でマスクします。0x07 は、「0000 0111」なので bit2~0 のみ有効に、他は強制的に"0"にします。

10. I/O ポートの入出力 2(プロジェクト:io2)

最後に、sw1 と sw2 の値を OR 演算で合わせます。

```
85 :      sw = sw1 | sw2;                /* P5とP4の値を合わせる */
```

ディップスイッチが"0101"(下上下上)のときの、dipsw_get 関数の動きを下図に示します。



10.5.2 led_out 関数

led_out 関数は、マイコンボードの 4 個の LED を点灯／消灯させる関数です。

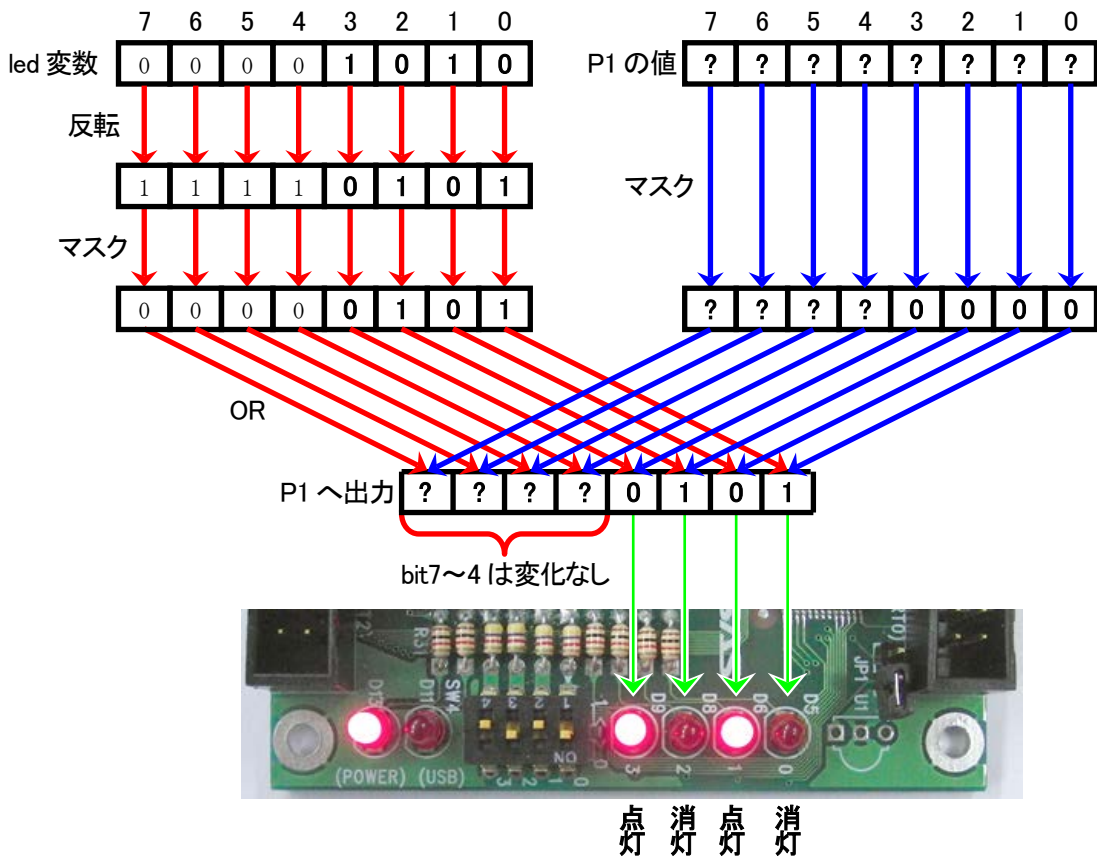
```

90 : /*****/
91 : /* マイコン部のLED出力 */
92 : /* 引数 スイッチ値 0~15 */
93 : /*****/
94 : void led_out( unsigned char led )
95 : {
96 :     unsigned char data;
97 :
98 :     led = ~led;
99 :     led &= 0x0f;
100 :     data = p1 & 0xf0;
101 :     p1 = data | led;
102 : }
    
```

マイコンボードには 4 個の LED が搭載されています。左から順にマイコンの P1_3、P1_2、P1_1、P1_0 の各ビットに接続されています。

LED はポート 1 の bit3~0 に接続されていますので、ポート 1 の bit7~4 には影響がないようにします。また、LED は"1"で消灯、"0"で点灯なので、反転させます。これを行うのが、led_out 関数です。

led_out 関数に引数 10 を代入したときの動きを、下図に示します。



10. I/O ポートの入出力 2(プロジェクト:io2)

led 変数に入力されている値は、“0”で消灯、“1”で点灯です。実際の LED は“1”で消灯、“0”で点灯なので、引数の led 変数を反転させます。

```
98 :    led = ~led;
```

LED は bit3～0 に接続されています。関係ない bit7～4 を“0”にしておきます。

```
99 :    led &= 0x0f;
```

次に、LED が繋がっているポート 1(P1)の値を読み込みます。このとき、LED のある bit3～0 は“0”にしておきます。bit7～4 の値は現在のポート 1(P1)の値のままにしておきます。この値を変数 data に代入します。

```
100 :    data = p1 & 0xf0;
```

最後に、data と led の値を OR 演算で合わせ、ポート 1(P1)へ出力します。

```
101 :    p1 = data | led;
```

10.5.3 main 関数

```

36 : void main( void )
37 : {
38 :     unsigned char d;
39 :
40 :     init();                /* 初期化                */
41 :
42 :     while( 1 ) {
43 :         d = dipsw_get();
44 :         led_out( d );
45 :     }
46 : }

```

main 関数は次のような動作をします。

43 行	変数 d にマイコンボード上のディップスイッチの値を読み込みます。
44 行	マイコンボード上の LED に変数 d の値を出力します。

結果、マイコンボード上のディップスイッチの値を、マイコンボード上の LED へ出力します。

10.6 演習

本演習では、
ディップスイッチ…マイコンボード上のディップスイッチ
LED…マイコンボード上の LED
とする。

- (1) ディップスイッチの値を反転させて、LED へ出力しなさい。
- (2) ディップスイッチの bit0 が "1"なら LED へ "1010"、"0"なら LED へ "0101"を出力するようにしなさい。ただし、bit0 以外は "0"か "1"かは分からないものとする。

11. プッシュスイッチの情報入力(プロジェクト:pushsw)

11.1 概要

本章では、マイコンボードのプッシュスイッチ(SW3)の値を読みこむ方法を説明します。

※製作マニュアルでは、SW3 をタクトスイッチと説明していますが、マイコンカー関連のマニュアルでは、本スイッチを「プッシュスイッチ」と説明しています。そのため、本マニュアルでも「プッシュスイッチ」という名称で説明します。

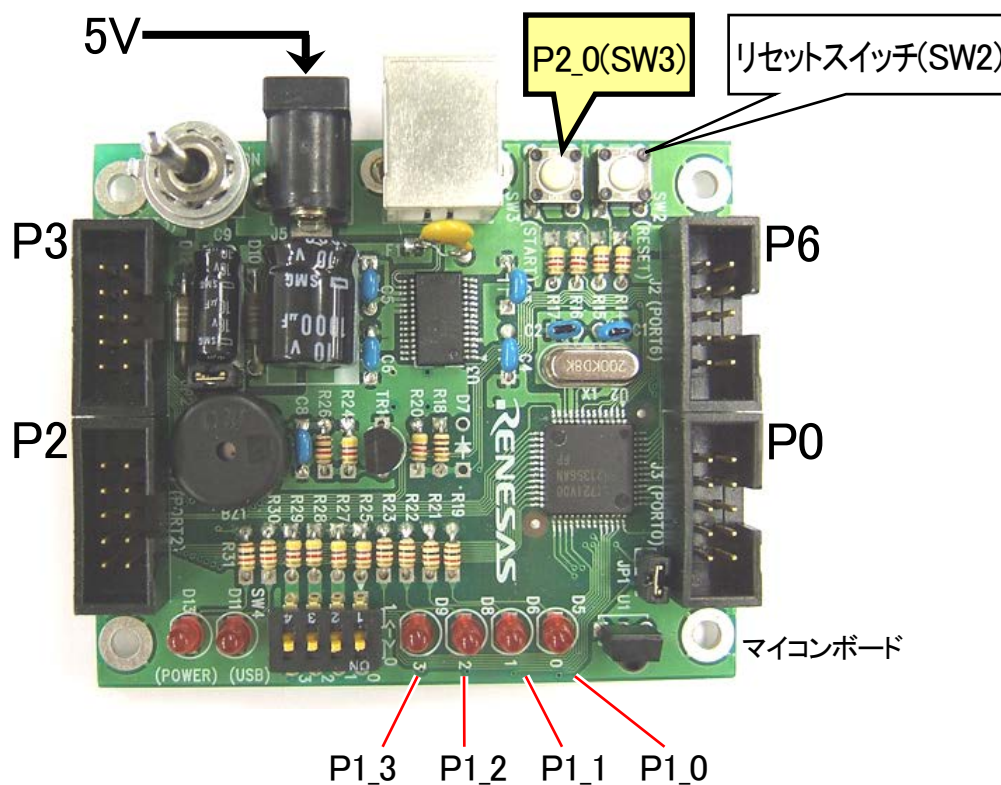
11.2 接続

■使用ポート

マイコンのポート	接続内容
P2_0	マイコンボード上のプッシュスイッチ(SW3)です。 ※SW2 はリセットスイッチで、マイコンのポートには接続されていません。
P1_3、P1_2、P1_1、P1_0	マイコンボード上の LED です。

■接続例

マイコンボードだけで実習できます。

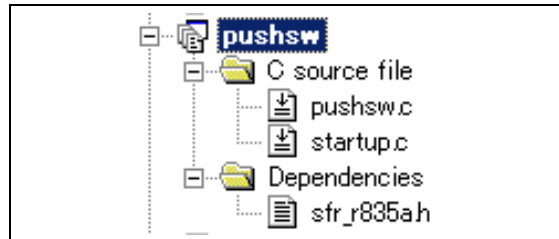


11. プッシュスイッチの情報入力(プロジェクト:pushsw)

■操作方法

マイコンボードのプッシュスイッチ(SW3)を ON/OFF すると、それに合わせて LED(D5)が点灯/消灯します。

11.3 プロジェクトの構成



	ファイル名	内容
1	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。このファイルは共通で、どのプロジェクトもこのファイルから実行されます。
2	pushsw.c	実際に制御するプログラムが書かれています。R8C/35A の内蔵周辺機能(SFR)の初期化も行います。
3	sfr_r835a.h	R8C/35A マイコンの内蔵周辺機能を制御するためのレジスタ(Special Function Registers)を定義したファイルです。

11.4 プログラム「pushsw.c」

```

1 : /******
2 : /* 対象マイコン R8C/35A */
3 : /* ファイル内容 プッシュスイッチの読み込み */
4 : /* バージョン Ver. 1.20 */
5 : /* Date 2010. 04. 19 */
6 : /* Copyright ルネサスマイコンカーラーイ事務局 */
7 : /* 日立インターメディックス株式会社 */
8 : /******
9 : /*
10 : 入力 : マイコンボードのプッシュスイッチSW3 (P2_0)
11 : 出力 : マイコンボードのLED(4bit)
12 :
13 : マイコンボードのプッシュスイッチSW3 (P2_0)から入力した状態を、
14 : マイコンボードのLED(4bit)に出力します。
15 : */
16 :
17 : /*=====*/
18 : /* インクルード */
19 : /*=====*/
20 : #include "sfr_r835a.h" /* R8C/35A SFRの定義ファイル */
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 : unsigned char pushsw_get( void );
31 : void led_out( unsigned char led );
32 :

```

11. プッシュスイッチの情報入力(プロジェクト:pushsw)

```

33 : /*****/
34 : /* メインプログラム */
35 : /*****/
36 : void main( void )
37 : {
38 :     unsigned char d;
39 :
40 :     init();                /* 初期化 */
41 :
42 :     while( 1 ) {
43 :         d = pushsw_get();
44 :         led_out( d );
45 :     }
46 : }
47 :
48 : /*****/
49 : /* R8C/35A スペシャルファンクションレジスタ(SFR)の初期化 */
50 : /*****/
51 : void init( void )
52 : {
53 :     int i;
54 :
55 :     /* クロックをXINクロック(20MHz)に変更 */
56 :     prc0 = 1;              /* プロテクト解除 */
57 :     cm13 = 1;              /* P4_6, P4_7をXIN-XOUT端子にする */
58 :     cm05 = 0;              /* XINクロック発振 */
59 :     for(i=0; i<50; i++ ); /* 安定するまで少し待つ(約10ms) */
60 :     ocd2 = 0;              /* システムクロックをXINにする */
61 :     prc0 = 0;              /* プロテクトON */
62 :
63 :     /* ポートの入出力設定 */
64 :     prc2 = 1;              /* PD0のプロテクト解除 */
65 :     pd0 = 0xe0;            /* 7-5:LED 4:MicroSW 3-0:Sensor */
66 :     p1 = 0x0f;             /* 3-0:LEDは消灯 */
67 :     pd1 = 0xdf;            /* 5:RXD0 4:TXD0 3-0:LED */
68 :     pd2 = 0xfe;            /* 0:PushSW */
69 :     pd3 = 0xfb;            /* 4:Buzzer 2:IR */
70 :     pd4 = 0x83;            /* 7:XOUT 6:XIN 5-3:DIP SW 2:VREF */
71 :     pd5 = 0x40;            /* 7:DIP SW */
72 :     pd6 = 0xff;
73 : }
74 :
75 : /*****/
76 : /* プッシュスイッチ値読み込み */
77 : /* 戻り値 プッシュスイッチの値 0:OFF 1:ON */
78 : /*****/
79 : unsigned char pushsw_get( void )
80 : {
81 :     unsigned char sw;
82 :
83 :     sw = ~p2;              /* プッシュスイッチ読み込み */
84 :     sw &= 0x01;            /* 不要ビットを"0"にする */
85 :
86 :     return sw;
87 : }
88 :
89 : /*****/
90 : /* マイコン部のLED出力 */
91 : /* 引数 スイッチ値 0~15 */
92 : /*****/
93 : void led_out( unsigned char led )
94 : {
95 :     unsigned char data;
96 :
97 :     led = ~led;
98 :     led &= 0x0f;
99 :     data = p1 & 0xf0;
100 :     p1 = data | led;
101 : }
102 :
103 : /*****/
104 : /* end of file */
105 : /*****/

```


11.5 プログラムの解説

11.5.1 pushsw_get 関数

マイコンボードにはプッシュスイッチが 2 個あります。機能を下記に示します。

- SW2:マイコンのリセットスイッチです。マイコンのポートには接続されていません。
- SW3:マイコンの P2_0 に接続されています。

pushsw_get 関数は、プッシュスイッチ SW3 の値を読み込む関数です。

```

75 : /*****/
76 : /* プッシュスイッチ値読み込み */
77 : /* 戻り値 プッシュスイッチの値 0:OFF 1:ON */
78 : /*****/
79 : unsigned char pushsw_get( void )
80 : {
81 :     unsigned char sw;
82 :
83 :     sw = ~p2;          /* プッシュスイッチ読み込み */
84 :     sw &= 0x01;      /* 不要ビットを"0"にする */
85 :
86 :     return sw;
87 : }
```

まず、変数 sw にポート 2(P2)の値を読み込みます。

```

83 :     sw = ~ p2;          /* プッシュスイッチ読み込み */
           ② ①
```

① ポート 2(P2)の値を読み込みます。

② このとき、反転させて sw 変数へ代入します。「~ (チルダ)」は C 言語で反転という意味です。

次に、変数 sw の値をマスクします。

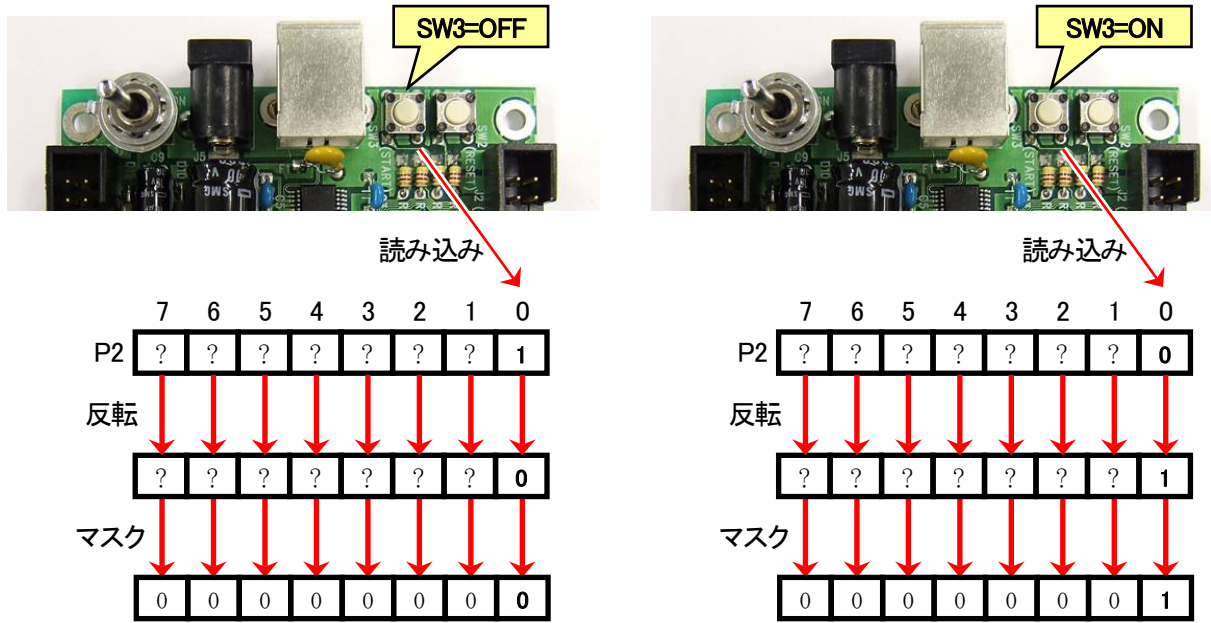
```

84 :     sw &= 0x01;      /* 不要ビットを"0"にする */
```

読み込んだ値を 0x01 でマスクします。0x01 は、「0000 0001」なので bit0 のみ有効に、他は強制的に"0"にします。

11. プッシュスイッチの情報入力(プロジェクト:pushsw)

プッシュスイッチが OFF のとき、ON のときの、pushsw_get 関数の動きを下図に示します。



11.5.2 main 関数

```

33 : /*****
34 : /* メインプログラム
35 : *****/
36 : void main( void )
37 : {
38 :     unsigned char d;
39 :
40 :     init();                /* 初期化
41 :
42 :     while( 1 ) {
43 :         d = pushsw_get();
44 :         led_out( d );
45 :     }
46 : }
    
```

main 関数は次のような動作をします。

43 行	変数 d にマイコンボード上のプッシュスイッチの値を読み込みます。
44 行	マイコンボード上の LED に変数 d の値を出力します。

結果、マイコンボード上のプッシュスイッチの値を、マイコンボード上の LED に出力します。

11.6 演習

本演習では、LED=マイコンボード上の D9,D8,D6,D5 とする。LED="1100"とは、左から D9="1"、D8="1"、D6="0"、D5="0"という意味とする。

- (1) プッシュスイッチが OFF で D6,D5 が点灯(その他は消灯)、ON で D9,D8 が点灯(その他は消灯)するようにしなさい。
- (2) ディップスイッチが押されるたびに、LED の値が"0000"(10 進数で 0)→"0001"(10 進数で 1、以下同じ)→"0010"→...→"1111"→"0000"と 1 つずつ増えていくようにしなさい。

12. マイクロスイッチの情報入力(プロジェクト:microsw)

12.1 概要

本章では、ミニマイコンカーVer.2 のセンサ部にあるのマイクロスイッチ(SW1)の値を読みこむ方法を説明します。

※マイクロスイッチはオプションです。

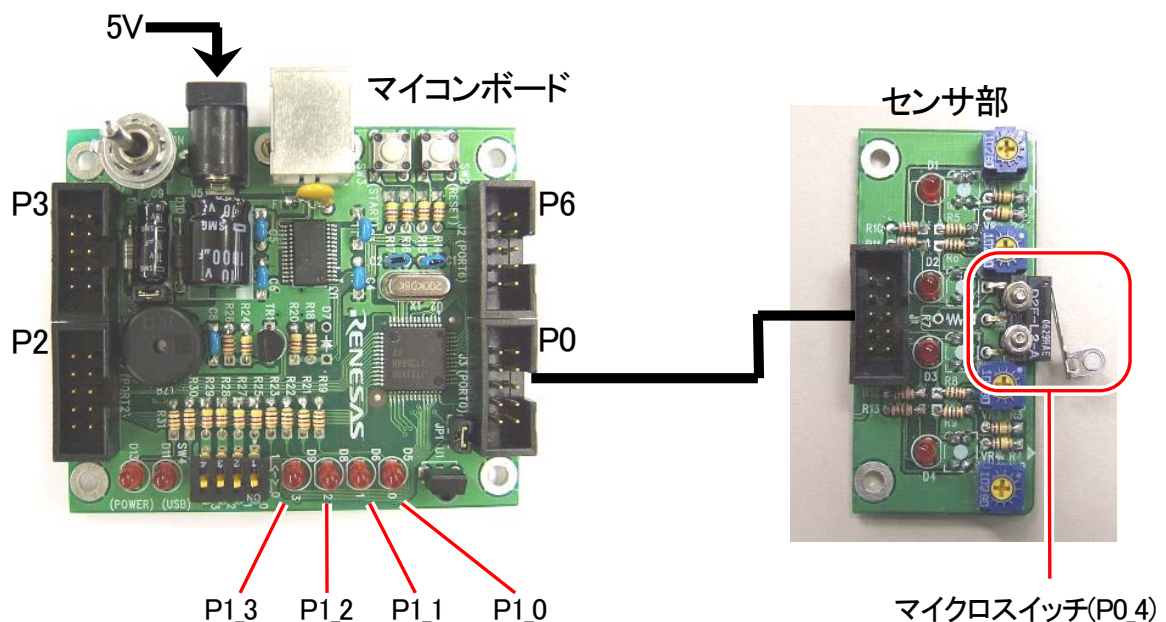
12.2 接続

■使用ポート

マイコンのポート	接続内容
P0_4	ミニマイコンカーVer.2 のマイクロスイッチ(SW1)。
P1_3、P1_2、P1_1、P1_0	マイコンボード上の LED です。

■接続例

マイコンボードの P0 とセンサ部をフラットケーブルで接続します。分離していない場合は、フラットケーブルで接続する必要はありません。

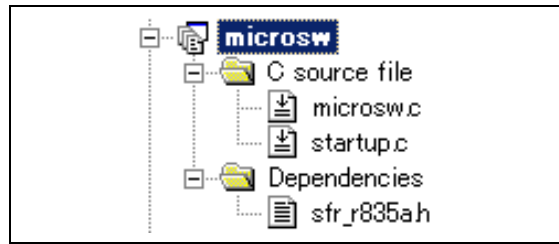


※マイクロスイッチはオプションです

■操作方法

マイクロスイッチ(SW1)を ON/OFF すると、それに合わせて LED(D5)が点灯/消灯します。

12.3 プロジェクトの構成



	ファイル名	内容
1	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。このファイルは共通で、どのプロジェクトもこのファイルから実行されます。
2	microsw.c	実際に制御するプログラムが書かれています。R8C/35A の内蔵周辺機能(SFR)の初期化も行います。
3	sfr_r835a.h	R8C/35A マイコンの内蔵周辺機能を制御するためのレジスタ(Special Function Registers)を定義したファイルです。

12.4 プログラム「microsw.c」

```

1 : /******
2 : /* 対象マイコン R8C/35A */
3 : /* ファイル内容 マイクロスイッチの読み込み */
4 : /* バージョン Ver. 1.20 */
5 : /* Date 2010. 04. 19 */
6 : /* Copyright ルネサスマイコンカーラーリ事務局 */
7 : /* 日立インターメディックス株式会社 */
8 : /******
9 : /*
10 : 入力：マイクロスイッチ(P0_4)
11 : 出力：マイコンボードのLED(4bit)
12 :
13 : センサ部のマイクロスイッチ(P0_4)から入力した状態を、
14 : マイコンボードのLED(4bit)に出力します。
15 : */
16 :
17 : /*=====*/
18 : /* インクルード */
19 : /*=====*/
20 : #include "sfr_r835a.h" /* R8C/35A SFRの定義ファイル */
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 : unsigned char microsw_get( void );
31 : void led_out( unsigned char led );
32 :
33 : /******
34 : /* メインプログラム */
35 : /******
36 : void main( void )
37 : {
38 :     unsigned char d;
39 :
40 :     init(); /* 初期化 */
41 :
42 :     while( 1 ) {
43 :         d = microsw_get();
44 :         led_out( d );
45 :     }
46 : }
47 :

```

12. マイクロスイッチの情報入力(プロジェクト:microsw)

```

48 : /*****
49 : /* R8C/35A スペシャルファンクションレジスタ(SFR)の初期化 */
50 : /*****
51 : void init( void )
52 : {
53 :     int i;
54 :
55 :     /* クロックをXINクロック(20MHz)に変更 */
56 :     prc0 = 1; /* プロテクト解除 */
57 :     cm13 = 1; /* P4_6, P4_7をXIN-XOUT端子にする*/
58 :     cm05 = 0; /* XINクロック発振 */
59 :     for(i=0; i<50; i++ ); /* 安定するまで少し待つ(約10ms) */
60 :     ocd2 = 0; /* システムクロックをXINにする */
61 :     prc0 = 0; /* プロテクトON */
62 :
63 :     /* ポートの入出力設定 */
64 :     prc2 = 1; /* PD0のプロテクト解除 */
65 :     pd0 = 0xe0; /* 7-5:LED 4:MicroSW 3-0:Sensor */
66 :     p1 = 0x0f; /* 3-0:LEDは消灯 */
67 :     pd1 = 0xdf; /* 5:RXD0 4:TXD0 3-0:LED */
68 :     pd2 = 0xfe; /* 0:PushSW */
69 :     pd3 = 0xfb; /* 4: buzzer 2: IR */
70 :     pd4 = 0x83; /* 7:XOUT 6:XIN 5-3:DIP SW 2:VREF*/
71 :     pd5 = 0x40; /* 7:DIP SW */
72 :     pd6 = 0xff;
73 : }
74 :
75 : /*****
76 : /* マイクロスイッチ値読み込み */
77 : /* 戻り値 マイクロスイッチの値 0:何も無し 1:押された状態 */
78 : /*****
79 : unsigned char microsw_get( void )
80 : {
81 :     unsigned char sw;
82 :
83 :     sw = ~p0 >> 4; /* マイクロスイッチ読み込み */
84 :     sw &= 0x01; /* 不要ビットを"0"にする */
85 :
86 :     return sw;
87 : }
88 :
89 : /*****
90 : /* マイコン部のLED出力 */
91 : /* 引数 スイッチ値 0~15 */
92 : /*****
93 : void led_out( unsigned char led )
94 : {
95 :     unsigned char data;
96 :
97 :     led = ~led;
98 :     led &= 0x0f;
99 :     data = p1 & 0xf0;
100 :     p1 = data | led;
101 : }
102 :
103 : /*****
104 : /* end of file */
105 : /*****

```

12.5 プログラムの解説

12.5.1 microsw_get 関数

microsw_get 関数は、ミニマイコンカーVer.2 のセンサ部のマイクロスイッチ SW1 の値を読み込む関数です。

```

75 : /*****/
76 : /* マイクロスイッチ値読み込み */
77 : /* 戻り値 マイクロスイッチの値 0:何も無し 1:押された状態 */
78 : /*****/
79 : unsigned char microsw_get( void )
80 : {
81 :     unsigned char sw;
82 :
83 :     sw = ~p0 >> 4;          /* マイクロスイッチ読み込み */
84 :     sw &= 0x01;           /* 不要ビットを"0"にする */
85 :
86 :     return sw;
87 : }
```

まず、変数 sw にポート 0(P0)の値を読み込みます。

```

83 :     sw = ~ p0 >> 4;          /* マイクロスイッチ読み込み */
           ② ① ③
```

- ① ポート 0(P0)の値を読み込みます。
- ② このとき、反転させて sw 変数へ代入します。「~ (チルダ)」は C 言語で反転という意味です。
- ③ 右に 4bit シフトします。

次に、変数 sw の値をマスクします。

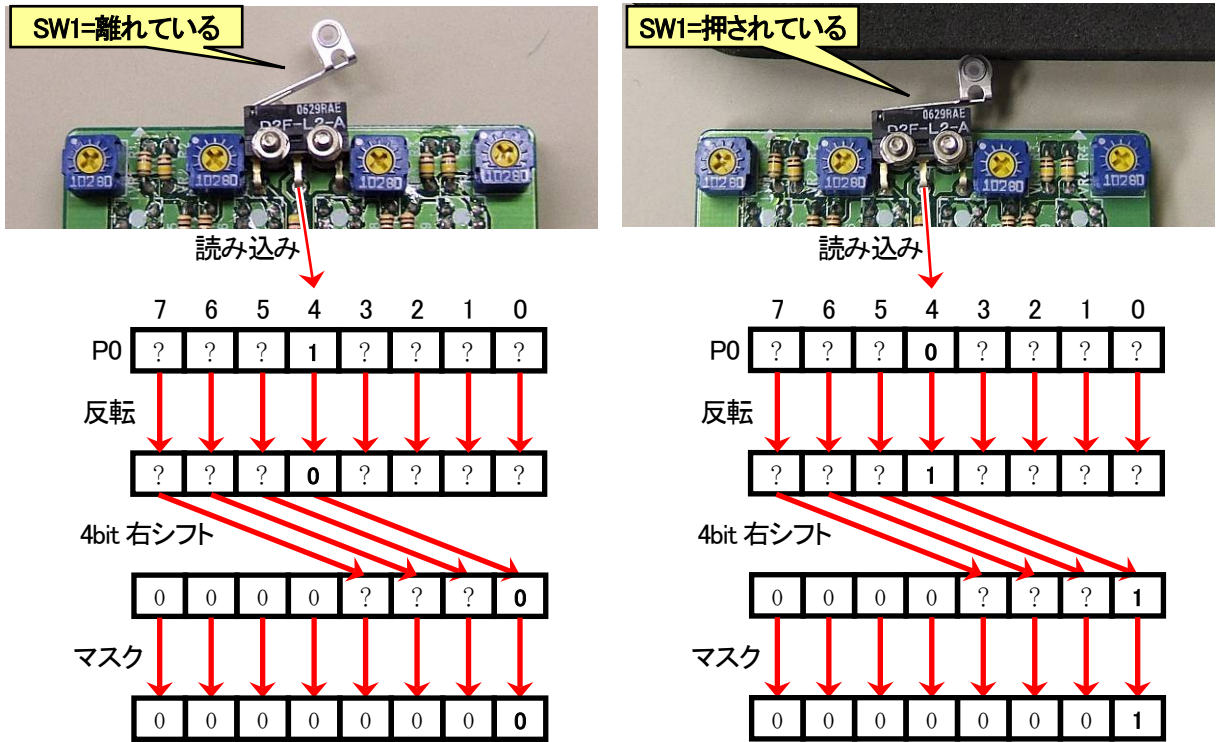
```

84 :     sw &= 0x01;           /* 不要ビットを"0"にする */
```

読み込んだ値を 0x01 でマスクします。0x01 は、「0000 0001」なので bit0 のみ有効に、他は強制的に"0"にします。

12. マイクロスイッチの情報入力(プロジェクト:microsw)

マイクロスイッチが離れているとき、押されているときの、microsw_get 関数の動きを下図に示します。



12.5.2 main 関数

```

33 : /*****/
34 : /* メインプログラム */
35 : /*****/
36 : void main( void )
37 : {
38 :     unsigned char d;
39 :
40 :     init();           /* 初期化 */
41 :
42 :     while( 1 ) {
43 :         d = microsw_get();
44 :         led_out( d );
45 :     }
46 : }
    
```

main 関数は次のような動作をします。

43 行	変数 d にミニマイコンカーVer.2 のマイクロスイッチの値を読み込みます。
44 行	マイコンボード上の LED に変数 d の値を出力します。

結果、ミニマイコンカーVer.2 のマイクロスイッチの値を、マイコンボード上の LED へ出力します。

12.6 演習

本演習では、LED=マイコンボード上の D9,D8,D6,D5 とする。LED="1100"とは、左から D9="1"(点灯)、D8="1"(点灯)、D6="0"(消灯)、D5="0"(消灯)という意味とする。

- (1) マイクロスイッチが OFF で D9,D8 が点灯(その他は消灯)、ON で D6,D5 が点灯(その他は消灯)するようにしなさい。
- (2) マイクロスイッチが押されるたびに、LED の値が"1111"(10 進数で 15)→"1110"(10 進数で 14、以下同じ)→"1101"→・・・→"0000"→"1111"と 1 つずつ減っていくようにしなさい。

13. ソフトウェアによるタイマ(プロジェクト:timer1)

13.1 概要

本章では、時間稼ぎをする関数(timer 関数)を使い、LED の光り方を 1 秒ごとに替える方法を説明します。LED をたくさん制御すれば、電飾などに応用可能です。ただし、時間の測り方は簡易的な方法を使っているため正確ではありません。正確なタイマが必要な場合は、プロジェクト「timer2」を参照してください。

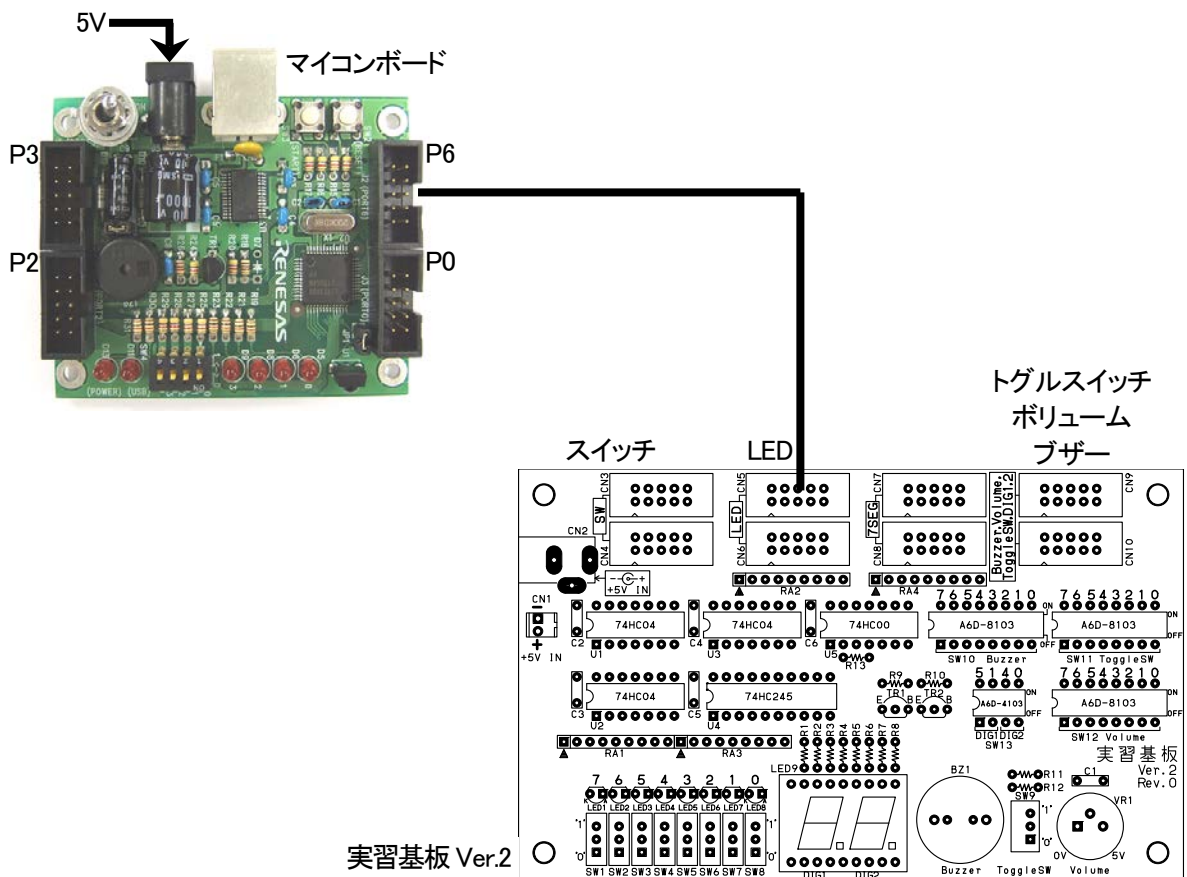
13.2 接続

■使用ポート

マイコンのポート	接続内容
P6 (J2)	実習基板 Ver.2 の LED 部など、出力機器を接続します。

■接続例

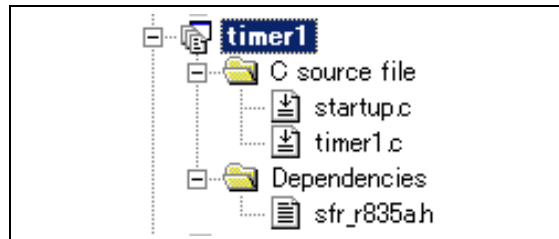
実習基板 Ver.2 を使ったときの接続例を下記に示します。



■操作方法

操作は特にありません。電源を入れるとLED が点滅します。LED の点滅の仕方をよく観察してください。

13.3 プロジェクトの構成



	ファイル名	内容
1	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。このファイルは共通で、どのプロジェクトもこのファイルから実行されます。
2	timer1.c	実際に制御するプログラムが書かれています。R8C/35A の内蔵周辺機能(SFR)の初期化も行います。
3	sfr_r835a.h	R8C/35A マイコンの内蔵周辺機能を制御するためのレジスタ(Special Function Registers)を定義したファイルです。

13.4 プログラム「timer1.c」

```

1 : /*****
2 : /* 対象マイコン R8C/35A */
3 : /* ファイル内容 ソフトウェアタイマ */
4 : /* バージョン Ver. 1.20 */
5 : /* Date 2010.04.19 */
6 : /* Copyright ルネサスマイコンカーラーリ事務局 */
7 : /* 日立インターメディアックス株式会社 */
8 : /*****
9 : /*
10 : 出力 : P6_7-P6_0(LEDなど)
11 :
12 : ポート6に繋いだLEDを1秒間隔で点滅させます。
13 : タイマはループによるソフトウェアタイマを使用します。
14 : */
15 :
16 : /*=====*/
17 : /* インクルード */
18 : /*=====*/
19 : #include "sfr_r835a.h" /* R8C/35A SFRの定義ファイル */
20 :
21 : /*=====*/
22 : /* シンボル定義 */
23 : /*=====*/
24 :
25 : /*=====*/
26 : /* プロトタイプ宣言 */
27 : /*=====*/
28 : void init( void );
29 : void timer( unsigned long timer_set );
30 :

```

13. ソフトウェアによるタイマ(プロジェクト:timer1)

```

31 : /*****/
32 : /* メインプログラム */
33 : /*****/
34 : void main( void )
35 : {
36 :     unsigned char d;
37 :
38 :     init();                /* 初期化 */
39 :
40 :     while( 1 ) {
41 :         p6 = 0x55;
42 :         timer( 1000 );
43 :         p6 = 0xaa;
44 :         timer( 1000 );
45 :         p6 = 0x00;
46 :         timer( 1000 );
47 :     }
48 : }
49 :
50 : /*****/
51 : /* R8C/35A スペシャルファンクションレジスタ(SFR)の初期化 */
52 : /*****/
53 : void init( void )
54 : {
55 :     int i;
56 :
57 :     /* クロックをXINクロック(20MHz)に変更 */
58 :     prc0 = 1;                /* プロテクト解除 */
59 :     cm13 = 1;                /* P4_6, P4_7をXIN-XOUT端子にする*/
60 :     cm05 = 0;                /* XINクロック発振 */
61 :     for(i=0; i<50; i++ );    /* 安定するまで少し待つ(約10ms) */
62 :     ocd2 = 0;                /* システムクロックをXINにする */
63 :     prc0 = 0;                /* プロテクトON */
64 :
65 :     /* ポートの入出力設定 */
66 :     prc2 = 1;                /* PDOのプロテクト解除 */
67 :     pd0 = 0xe0;              /* 7-5:LED 4:MicroSW 3-0:Sensor */
68 :     p1 = 0x0f;                /* 3-0:LEDは消灯 */
69 :     pd1 = 0xdf;              /* 5:RXD0 4:TXD0 3-0:LED */
70 :     pd2 = 0xfe;              /* 0:PushSW */
71 :     pd3 = 0xfb;              /* 4:Buzzer 2:IR */
72 :     pd4 = 0x83;              /* 7:XOUT 6:XIN 5-3:DIP SW 2:VREF*/
73 :     pd5 = 0x40;              /* 7:DIP SW */
74 :     pd6 = 0xff;              /* LEDなど出力 */
75 : }
76 :
77 : /*****/
78 : /* タイマ本体 */
79 : /* 引数 タイマ値 1=1ms */
80 : /*****/
81 : void timer( unsigned long timer_set )
82 : {
83 :     int i;
84 :
85 :     do {
86 :         for( i=0; i<1240; i++ );
87 :     } while( timer_set-- );
88 : }
89 :
90 : /*****/
91 : /* end of file */
92 : /*****/

```

13.5 プログラムの解説

13.5.1 timer 関数(時間稼ぎ)

timer 関数は、実行した行で時間稼ぎをする関数です。

```

81 : void timer( unsigned long timer_set )
82 : {
83 :     int i;
84 :
85 :     do {
86 :         for( i=0; i<1240; i++ ); この行で 1ms の時間稼ぎ
87 :     } while( timer_set-- );
88 : }

```

86 行	<p>この行で、1ms の時間稼ぎをします。i を 1 足して 1240 以下なら for の次の命令を実行します。今回は、命令がないので、何もせずに終わります。また i を 1 足して・・・ を繰り返し、i が 1240 になったら次の行へ移ります。この繰り返しは 1ms になります。1240 という数字は、実測です。</p> <p>※1240 について この数値は、</p> <ul style="list-style-type: none"> ・ルネサス統合開発環境のバージョン(コンパイラのバージョン) ・ツールチェーンの設定 ・クリスタルの値 <p>によって違います。今回の条件固有の数値と覚えておくと良いでしょう。</p>
85 行、 87 行	<pre> do { 命令 } while(条件); </pre> <p>として、条件 が成り立つ間、命令 を実行し続けます。今回の条件は、timer_set 変数を-1 ずつして、0 になったら終了です。timer_set 変数は、関数の引数です。例えば、 timer(1000); と実行したなら、timer_set 変数には 1000 が代入され、do~while 文が 1000 回実行されることになります。</p>

使い方を下記に示します。

```
timer( 時間稼ぎする時間[ms] );
```

カッコの中には、時間稼ぎをする時間を ms 単位で代入します。1 秒にしたいなら、1 秒=1000ms なので、1000 を代入します。

13.5.2 main 関数

```

34 : void main( void )
35 : {
36 :     unsigned char d;
37 :
38 :     init();                /* 初期化                */
39 :
40 :     while( 1 ) {
41 :         p6 = 0x55;
42 :         timer( 1000 );
43 :         p6 = 0xaa;
44 :         timer( 1000 );
45 :         p6 = 0x00;
46 :         timer( 1000 );
47 :     }
48 : }

```

41 行	ポート 6 に 0x55(0101 0101)を出力します。
42 行	timer 関数で 1000ms(=1 秒)の時間稼ぎをします。
43 行	ポート 6 に 0xaa(1010 1010)を出力します。
44 行	timer 関数で 1000ms(=1 秒)の時間稼ぎをします。
45 行	ポート 6 に 0x00(0000 0000)を出力します。
46 行	timer 関数で 1000ms(=1 秒)の時間稼ぎをします。

※命令の実行時間について

プログラムの1命令は、数百 ns(ナノ秒)から数 μ s(マイクロ秒)という非常に短い時間で終わります。逆に言うと、短くても時間がかかるということで、何十万回も繰り返すと秒単位の時間となります。timer 関数は、何もしないことを何千回も繰り返すことによって、長い時間、時間稼ぎをしています。

main 関数のそれぞれの行の実行時間を、下記に示します。

```

34 : void main( void )
35 : {
36 :     unsigned char d;
37 :
38 :     init();                ←init関数内の命令を実行する時間かかる(数百  $\mu$ s程度)
39 :
40 :     while( 1 ) {
41 :         p6 = 0x55;        ←数  $\mu$ s
42 :         timer( 1000 );   ←約1000ms
43 :         p6 = 0xaa;        ←数  $\mu$ s
44 :         timer( 1000 );   ←約1000ms
45 :         p6 = 0x00;        ←数  $\mu$ s
46 :         timer( 1000 );   ←約1000ms
47 :     }
48 : }
```

13.6 演習

(1) 次の状態をポート6のLEDに出力するようにしなさい。

- ① 1111 0000 を 0.5 秒間
- ② 0000 1111 を 0.5 秒間
- ③ 0000 0000 を 0.25 秒間

(2) 次の状態をマイコンボードのLEDに出力するようにしなさい。

- ① 0101 を 0.2 秒間
- ② 1010 を 0.2 秒間

14. 割り込みによるタイマ(プロジェクト:timer2)

14.1 概要

本章は、動作はプロジェクト「timer1」と同じですが、時間の測り方を R8C/35A 内蔵のタイマ RB を使い、正確に時間を計ります。具体的には、タイマ RB で 1ms ごとに割り込みを発生させ、その回数で時間を計ります。

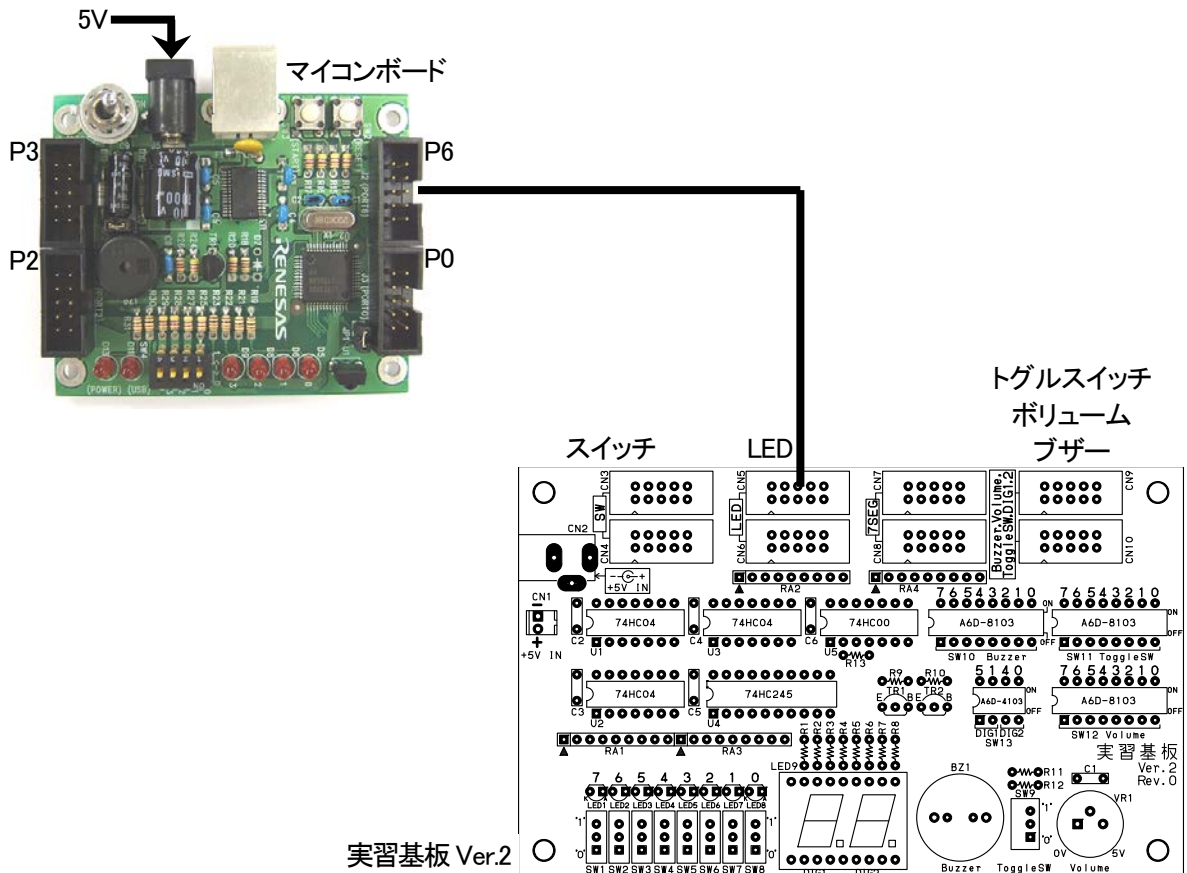
14.2 接続

■使用ポート

マイコンのポート	接続内容
P6 (J2)	実習基板 Ver.2 の LED 部など、出力機器を接続します。

■接続例

実習基板 Ver.2 を使ったときの接続例を下記に示します。

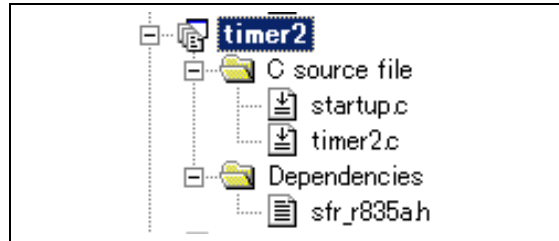


14. 割り込みによるタイマ(プロジェクト:timer2)

■操作方法

操作は特にありません。電源を入れるとLED が点滅します。LED の点滅の仕方をよく観察してください。

14.3 プロジェクトの構成



	ファイル名	内容
1	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。このファイルは共通で、どのプロジェクトもこのファイルから実行されます。
2	timer2.c	実際に制御するプログラムが書かれています。R8C/35A の内蔵周辺機能(SFR)の初期化も行います。
3	sfr_r835a.h	R8C/35A マイコンの内蔵周辺機能を制御するためのレジスタ(Special Function Registers)を定義したファイルです。

14.4 プログラム「timer2.c」

```

1 : /*****
2 : /* 対象マイコン R8C/35A */
3 : /* ファイル内容 タイマRB割り込みによるタイマ */
4 : /* バージョン Ver. 1.20 */
5 : /* Date 2010.04.19 */
6 : /* Copyright ルネサスマイコンカーラーリ事務局 */
7 : /* 日立インターメディアックス株式会社 */
8 : /*****
9 : /*
10 : 出力 : P6_7-P6_0(LEDなど)
11 :
12 : ポート6に繋いだLEDを1秒間隔で点滅させます。
13 : タイマはタイマRB割り込みによる正確なタイマを使用します。
14 : */
15 :
16 : /*=====*/
17 : /* インクルード */
18 : /*=====*/
19 : #include "sfr_r835a.h" /* R8C/35A SFRの定義ファイル */
20 :
21 : /*=====*/
22 : /* シンボル定義 */
23 : /*=====*/
24 :
25 : /*=====*/
26 : /* プロトタイプ宣言 */
27 : /*=====*/
28 : void init( void );
29 : void timer( unsigned long timer_set );
30 :
31 : /*=====*/
32 : /* グローバル変数の宣言 */
33 : /*=====*/
34 : unsigned long cnt_rb; /* タイマRB用 */
35 :

```

14. 割り込みによるタイマ(プロジェクト:timer2)

```

36 : /*****/
37 : /* メインプログラム */
38 : /*****/
39 : void main( void )
40 : {
41 :     init(); /* 初期化 */
42 :     asm(" fset I "); /* 全体の割り込み許可 */
43 :
44 :     while( 1 ) {
45 :         p6 = 0x55;
46 :         timer( 1000 );
47 :         p6 = 0xaa;
48 :         timer( 1000 );
49 :         p6 = 0x00;
50 :         timer( 1000 );
51 :     }
52 : }
53 :
54 : /*****/
55 : /* R8C/35A 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 = 0xe0; /* 7-5:LED 4:MicroSW 3-0:Sensor */
72 :     p1 = 0x0f; /* 3-0:LEDは消灯 */
73 :     pd1 = 0xdf; /* 5:RXD0 4:TXD0 3-0:LED */
74 :     pd2 = 0xfe; /* 0:PushSW */
75 :     pd3 = 0xfb; /* 4:Buzzer 2:IR */
76 :     pd4 = 0x83; /* 7:XOUT 6:XIN 5-3:DIP SW 2:VREF*/
77 :     pd5 = 0x40; /* 7:DIP SW */
78 :     pd6 = 0xff; /* LEDなど出力 */
79 :
80 :     /* タイマRBの設定 */
81 :     /* 割り込み周期 = 1 / 20[MHz] * (TRBPRE+1) * (TRBPR+1)
82 :     = 1 / (20*10-6) * 200 * 100
83 :     = 0.001[s] = 1[ms]
84 :
85 :     */
86 :     trbmr = 0x00; /* 動作モード、分周比設定 */
87 :     trbpre = 200-1; /* プリスケールレジスタ */
88 :     trbpr = 100-1; /* プライマリレジスタ */
89 :     trbic = 0x07; /* 割り込み優先レベル設定 */
90 :     trbcr = 0x01; /* カウント開始 */
91 : }
92 : /*****/
93 : /* タイマ本体 */
94 : /* 引数 タイマ値 1=1ms */
95 : /*****/
96 : void timer( unsigned long timer_set )
97 : {
98 :     cnt_rb = 0;
99 :     while( cnt_rb < timer_set );
100 : }
101 :
102 : /*****/
103 : /* タイマRB 割り込み処理 */
104 : /*****/
105 : #pragma interrupt intTRB(vect=24)
106 : void intTRB( void )
107 : {
108 :     cnt_rb++;
109 : }
110 :
111 : /*****/
112 : /* end of file */
113 : /*****/

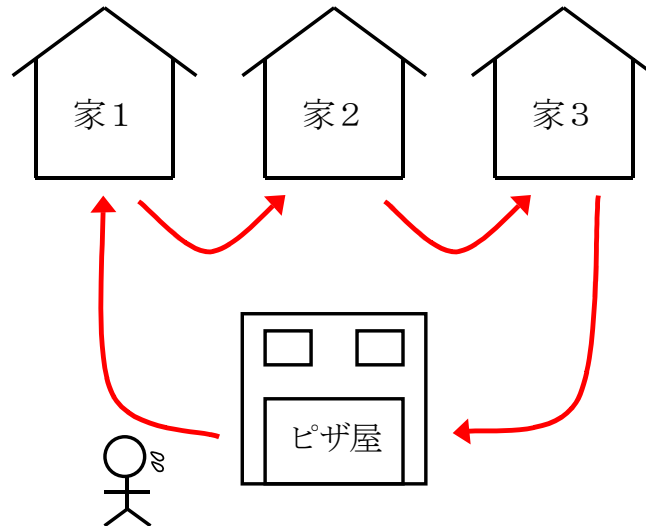
```

14.5 プログラムの解説

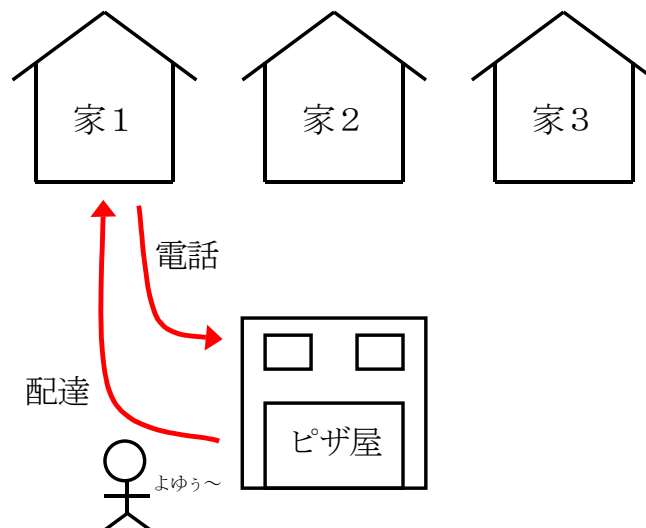
14.5.1 割り込みとは

(1) 割り込みとは

例えば、ピザ屋さんが家 1～3 に注文がないか回るとします。バイト君は、定期的に家を回らなければいけません。定期的に聞きに行くことを制御の用語で**ポーリング**といいます。回る間隔が長いと、待たせることになります。また、注文がなければ無駄足になってしまいます(下図)。



そこで、電話で注文を受けることにします。バイト君は、それぞれの家を回る必要がありません。電話の注文が来ればその家に届けばよいので作業効率が良いです(下図)。



ただし、電話を用意する必要があります。プログラムに当てはめると、割り込み設定に当たります。さらに、電話の受け答えをする必要があります。割り込みプログラムに当たります。

14. 割り込みによるタイマ(プロジェクト:timer2)

まとめると下記のようになります。

・注文がないか聞きに回る

制御の用語で「ポーリング」といいます。定期的に監視しなければいけないので、監視する部分が多いと、監視が遅れたり、監視もれが起きます。

・電話で注文をうける

制御の用語(でもないですが)で「割り込み」といいます。電話のように、きっかけがあったときにだけ対処すれば良いので効率が良いです。ただし、電話の用意、電話の受け答えをする必要があります。

人で例えましたが、マイコンの場合は下記のようになります。

人間の場

電話を用意する

→

ベルが鳴る

→

電話対応する

→

マイコンの場

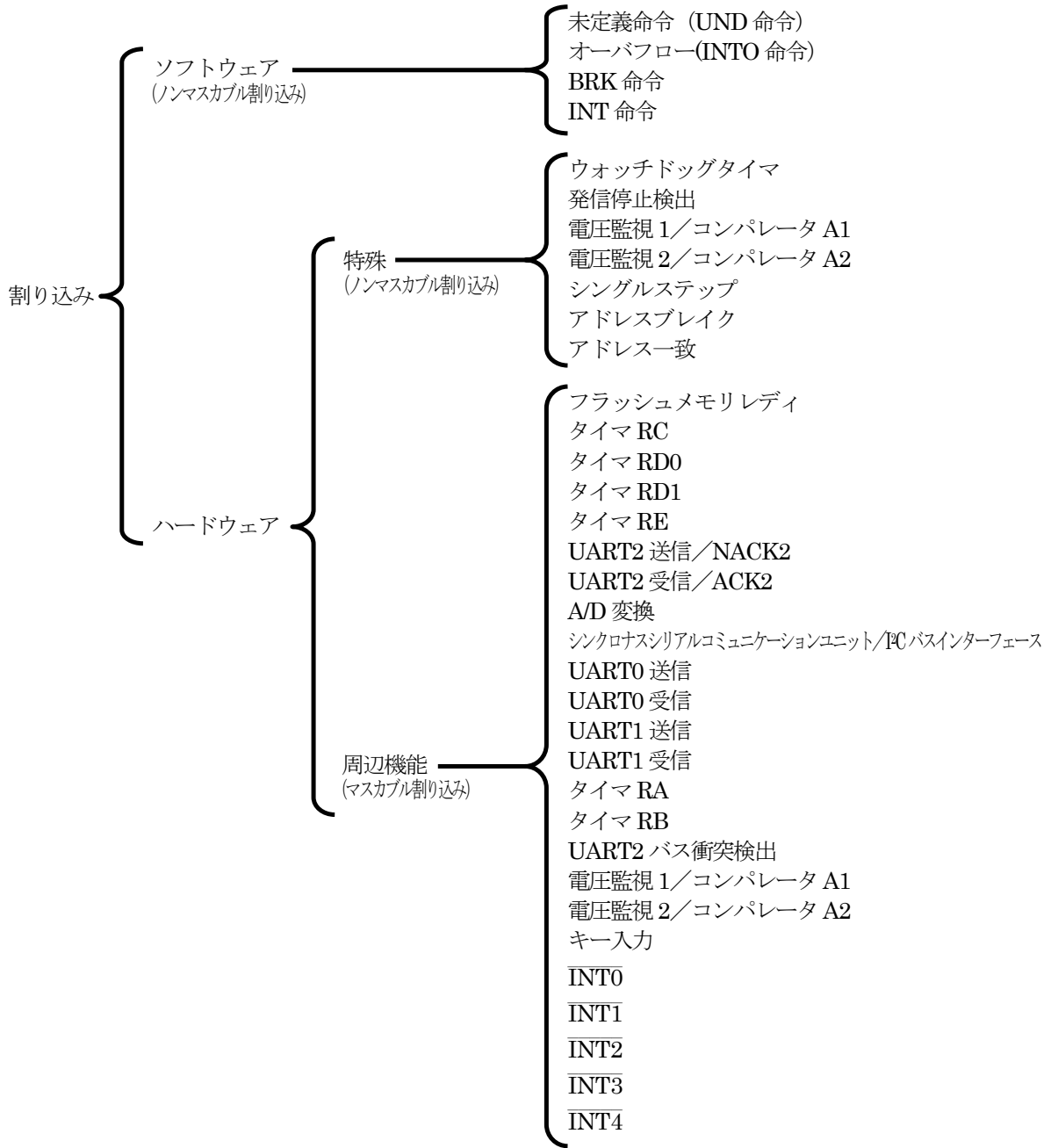
割り込みプログラムを設定する

割り込みが発生する

割り込みプログラムを実行する

(2) 割り込みの種類

R8C/35A の割り込みの種類を、下表に示します。



マスクブル割り込み	フラグレジスタ(FLG)の割り込み許可フラグ(I フラグ)による割り込みの許可(禁止)や割り込み優先レベルによる割り込み優先順位の変更が 可能
ノンマスクブル割り込み	フラグレジスタ(FLG)の割り込み許可フラグ(I フラグ)による割り込みの許可(禁止)や割り込み優先レベルによる割り込み優先順位の変更が 不可能

14.5.2 init 関数(タイマ RB の設定)

タイマ RB を使って、1ms ごとに割り込みを発生させます。

80 :	/* タイマRBの設定 */	
81 :	/* 割り込み周期 = 1 / 20[MHz] * (TRBPRE+1) * (TRBPR+1)	
82 :	= 1 / (20*10 ⁻⁶) * 200 * 100	
83 :	= 0.001[s] = 1[ms]	
84 :	*/	
85 :	trbmr = 0x00;	/* 動作モード、分周比設定 */
86 :	trbpre = 200-1;	/* プリスケーラレジスタ */
87 :	trbpr = 100-1;	/* プライマリレジスタ */
88 :	trbic = 0x07;	/* 割り込み優先レベル設定 */
89 :	trbcr = 0x01;	/* カウント開始 */

(1) タイマ RB とは

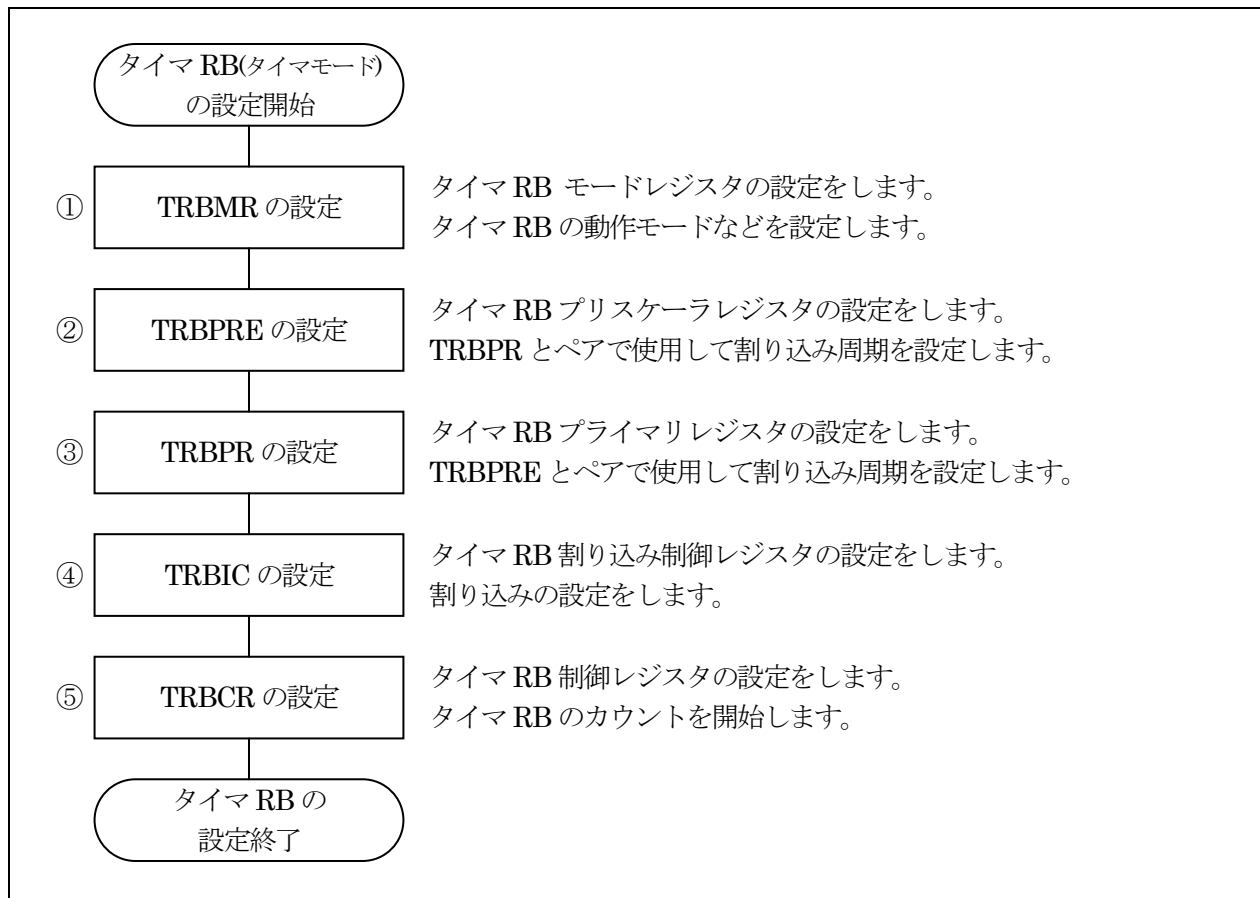
R8C/35A には、タイマ RB というタイマが 1 個内蔵されています。タイマ RB には、次の 4 種類のモードがあります。

モード	詳細
タイマモード	内部カウントソース(周辺機能クロックまたはタイマ RA のアンダフロー)をカウントするモードです。
プログラマブル 波形発生モード	任意のパルス幅を連続して出力するモードです。
プログラマブル ワンショット発生モード	ワンショットパルスを出力するモードです。
プログラマブルウェイト ワンショット発生モード	ディレイドワンショットパルスを出力するモードです。

本プロジェクトでは、タイマモードを使い、1ms ごとに割り込みを発生させます。

(3) タイマ RB の設定(タイマモード)

今回は、タイマ RB をタイマモードで使用して、1ms ごとに割り込みを発生させるように設定にします。レジスタの設定手順を下記に示します。



14. 割り込みによるタイマ(プロジェクト:timer2)

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

タイマ RB のモードを設定します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7	タイマ RB カウントソース遮断 ビット(注 1) tckcut_trbmr	0:カウントソース供給 1:カウントソース遮断 供給するので"0"を設定します。	0
bit6		"0"を設定	0
bit5,4	タイマ RB カウントソース選択 ビット(注 1) bit5:tck1_trbmr bit4:tck0_trbmr	00:f1 (1/20MHz=50ns) 01:f8 (8/20MHz=400ns) 10:タイマ RA のアンダフロー 11:f2 (2/20MHz=100ns) f1 を選択します。	00
bit3	タイマ RB 書き込み制御ビット (注 2) twrc_trbmr	0:リロードレジスタとカウンタへの書き込み 1:リロードレジスタのみ書き込み "0"を設定します。	0
bit2		"0"を設定	0
bit1,0	タイマ RB 動作モード選択ビッ ト(注 1) bit1:tmod1_trbmr bit0:tmod0_trbmr	00:タイマモード 01:プログラマブル波形発生モード 10:プログラマブルワンショット発生モード 11:プログラマブルウェイトワンショット発生モード タイマモードで動作させるので"00"を設定します。	00

注 1. TMOD1~TMOD0 ビット、TCK1~TCK0 ビット、TCKCUT ビットは、TRBCR レジスタの TSTART ビットと TCSTF ビットが共に"0"(カウント停止)のときに変更してください。

注 2. TWRC ビットは、タイマモードのとき"0"または"1"が選択できます。プログラマブル波形発生モード、プログラマブルワンショット発生モード、プログラマブルウェイトワンショット発生モードでは"1"(リロードレジスタのみ書き込み)にしてください。

タイマ RB モードレジスタ(TRBMR)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	0	0	0	0
16 進数	0				0			

※タイマ RB モードレジスタ(TRBMR)のタイマ RB カウントソース選択ビットの設定方法

タイマ RB モードレジスタ(TRBMR)のタイマ RB カウントソース選択ビット(bit5,4)で、タイマ RB プリスケールレジスタ(TRBPRE)、タイマ RB プライマリレジスタ(TRBPR)がどのくらいの間隔で+1 するか設定します。

タイマ RB モードレジスタ(TRBMR)のタイマ RB カウントソース選択ビットの値と、割り込み間隔の関係を下記に示します。

TRBMR bit5,4	内容
00	<p>タイマ RB プリスケールレジスタ(TRBPRE)がカウントアップする時間を、f1 に設定します。時間は、 $f1/20\text{MHz}=1/20\text{MHz}=50\text{ns}$ 設定できる割り込み間隔の最大は、 $50\text{ns} \times 65,536 = \mathbf{3.2768\text{ms}}$ よって、この時間以内の割り込み間隔を設定する場合は"00"を設定、これ以上の割り込み間隔を設定したい場合は次以降の値を検討します。</p>
11	<p>タイマ RB プリスケールレジスタ(TRBPRE)がカウントアップする時間を、f2 に設定します。時間は、 $f2/20\text{MHz}=2/20\text{MHz}=100\text{ns}$ 設定できる割り込み間隔の最大は、 $100\text{ns} \times 65,536 = \mathbf{6.5536\text{ms}}$ よって、この時間以内の割り込み間隔を設定する場合は"11"を設定、これ以上の割り込み間隔を設定したい場合は次以降の値を検討します。</p>
01	<p>タイマ RB プリスケールレジスタ(TRBPRE)がカウントアップする時間を、f8 に設定します。時間は、 $f8/20\text{MHz}=8/20\text{MHz}=400\text{ns}$ 設定できる割り込み間隔の最大は、 $400\text{ns} \times 65,536 = \mathbf{26.2144\text{ms}}$ よって、この時間以内の割り込み間隔を設定する場合は"01"を設定します。これ以上の割り込み間隔を設定することはできません。これ以上の割り込み間隔を設定しなくても良いように、プログラム側で工夫してください。</p>

今回は、割り込み間隔を 1ms にします。

"00"の設定…最大の割り込み間隔は 3.2768ms、今回設定したい 1ms の割り込み間隔を設定できるので OK

よって、"00"を設定します。

14. 割り込みによるタイマ(プロジェクト:timer2)

②タイマ RB プリスケールレジスタ(TRBPRES:Timer RB prescaler register)の設定

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

タイマ RB プリスケールレジスタ(TRBPRES)とタイマ RB プライマリレジスタ(TRBPR)はペアで使い、割り込み周期を設定します。

TRBPRES と TRBPR の値を計算する式を、下記に示します。

$$\text{タイマ RB 割り込み要求周期} = \text{タイマ RB カウントソース} \times (\text{TRBPRES} + 1) \times (\text{TRBPR} + 1)$$

TRBPRES と TRBPR を左辺に移動します。

$$(\text{TRBPRES} + 1) \times (\text{TRBPR} + 1) = \text{タイマ RB 割り込み要求周期} / \text{タイマ RB カウントソース}$$

今回、設定する割り込み周期は 1ms です。タイマ RB カウントソースとは、タイマ RB モードレジスタ(TRBMR)の bit5,4 に設定している内容で、今回は f1 (50ns) です。よって、

$$(\text{TRBPRES} + 1) \times (\text{TRBPR} + 1) = (1 \times 10^{-3}) / (50 \times 10^{-9})$$

$$\frac{(\text{TRBPRES} + 1) \times (\text{TRBPR} + 1)}{B \quad C \quad A} = \underline{20,000}$$

次の条件になるよう、A、B、C 部分を設定してください。

A…65,536 以下にする必要があります。65,537 以上の場合、カウントソースを長い時間に設定し直してください。

B…1~256 以下になるよう、値を設定してください。値は整数です。

C…1~256 以下になるよう、値を設定してください。値は整数です。

今回、A は 20,000 なので、A の条件は満たしています。

B、C を決める公式はありません。B×C が、20,000 になるような数字を見つけてください。

例えば、B=200 とすると、

$$200 \times C = 20,000$$

$$\therefore C = 100$$

となります。

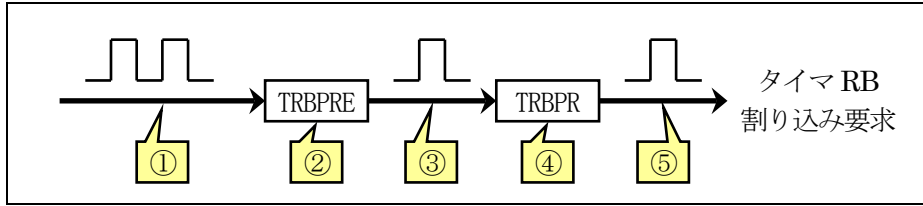
$$B = \text{TRBPRES} + 1 \quad \therefore \text{TRBPRES} = 200 - 1 = \mathbf{199}$$

$$C = \text{TRBPR} + 1 \quad \therefore \text{TRBPR} = 100 - 1 = \mathbf{99}$$

を設定します。

14. 割り込みによるタイマ(プロジェクト:timer2)

このときの動作を下記に示します。



①	タイマ RB モードレジスタ(TRBMR)の RB カウントソース選択ビットで設定したパルスが入力されます。今回は f1 を選択しているので、次のパルスが入力されます。 f1=1/20MHz=50ns
②	タイマ RB プリスケアラレジスタ(TRBPRES)はダウンカウントです。設定した値からスタートし、1 つずつ値が減っていき 0 の次は設定値になります。例えば 9 を設定したなら 9→8→7→6→5→4→3→2→1→0→9→8・・・となります。このように 0 も含めてカウントするため、10 回カウントしたければ、1 小さい値の 9 を設定します。 今回は、199 を設定します。そのため、199→198→・・・→2→1→0→199→198・・・、とカウントされます。
③	タイマ RB プリスケアラレジスタ(TRBPRES)が 0→199 になった瞬間、1 パルス出力されます。これは TRBPRES に 200 パルス入ると 1 パルス出力されるということです。パルスが出力される間隔は、次のようになります。 50ns(入力されるパルスの間隔)×200=10,000ns=10 μs
④	タイマ RB プライマリレジスタ(TRBPR)はダウンカウントです。設定した値からスタート、0 の次は設定値になります。 今回は 99 を設定します。そのため、99→98→・・・→2→1→0→99→98・・・、とカウントされます。TRBPR には 10 μs ごとにパルスが入力されます。要は、10 μs ごとに、TRBPR が-1 されます。
⑤	タイマ RB プライマリレジスタ(TRBPR)が 0→99 になった瞬間、1 パルス出力されます。これは TRBPR に 100 パルス入ると 1 パルス出力されるということです。パルスが出力される間隔は、次のようになります。 10 μs(入力されるパルスの間隔)×100=1,000 μs = 1ms

このように、TRBPR から 1ms ごとにパルスが出力されます。**このパルスが割り込みを発生させるきっかけになります。**要は、タイマ RB によって 1ms ごとに割り込みを発生させます。

タイマ RB プリスケアラレジスタ(TRBPRES)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	1 9 9							

タイマ RB プライマリレジスタ(TRBPR)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	9 9							

14. 割り込みによるタイマ(プロジェクト:timer2)

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

タイマ RB の割り込み関係の設定をします。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7~4		"0"を設定	0000
bit3	割り込み要求ビット ir_trbic	0:割り込み要求なし 1:割り込み要求あり 割り込みが発生すると自動で"1"になります。割り込みプログラムプログラムを実行すると自動的に"0"になります。設定は、"0"にします。	0
bit2~0	割り込み優先レベル選択ビット bit2: ilvl2_trbic bit1: ilvl1_trbic bit0: ilvl0_trbic	000:レベル 0 (割り込み禁止) 001:レベル 1 010:レベル 2 011:レベル 3 100:レベル 4 101:レベル 5 110:レベル 6 111:レベル 7 他の割り込みが同時に発生した場合、どの割り込みを優先させるか設定します。レベルの高い割り込みが優先されます。割り込みを2つ以上使う場合は、どれを優先させるかここで決めます。今回の割り込みは、タイマ RB だけなのでレベル 1~7 のどれを設定しても構いません。一応、レベルのいちばん高い"111"を設定します。	111

タイマ RB 割り込み制御レジスタ(TRBIC)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	0	1	1	1
16進数	0				7			

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

タイマ RB のカウント動作を開始するよう設定します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7~3		"00000"を設定	00000
bit2	タイマ RB カウント強制停止ビット(注 1、2) tstop_trbcr	"1"を書くとカウントが強制停止します。 読んだ場合、その値は"0"になります。	0
bit1	タイマ RB カウントステータスフラグ(注 1) tcstf_trbcr	0:カウント停止 1:カウント中(注 3) カウント中かどうかチェックするフラグです。書き込みは無効です。書き込むときは"0"を設定します。	0
bit0	タイマ RB カウント開始ビット(注 1) tstart_trbcr	0:カウント停止 1:カウント開始 タイマ RB のカウントを開始するので"1"を設定します。 設定した瞬間から、カウントが開始されます。	1

注 1. TSTART、TCSTF、TSTOP ビットの使用上の注意事項については、ハードウェアマニュアルの「18.7 タイマ RB 使用上の注意」を参照してください。

注 2. TSTOPビットに"1"を書くと、TRBPRES レジスタ、TRBSC レジスタ、TRBPR レジスタ、TSTART ビット、TCSTF ビット、TRBOCR レジスタの TOSSTF ビットがリセット後の値になります。

注 3. タイマモード、プログラマブル波形発生モードでは、カウント中を示します。プログラマブルワンショット発生モード、プログラマブルウェイトワンショット発生モードでは、ワンショットパルスのトリガを受け付けられることを示します。

タイマ RB 制御レジスタ(TRBCR)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	0	0	0	1
16 進数	0				1			

14.5.3 intTRB 関数(1ms ごとに実行される関数)

先の設定で、タイマ RB を 1ms ごとに割り込みを発生させる設定にしました。intTRB 関数は、この割り込みが発生したときに実行される関数です。

```

105 : #pragma interrupt intTRB(vect=24)
106 : void intTRB( void )
107 : {
108 :     cnt_rb++;
109 : }
    
```

105 行	#pragma interrupt 割り込み処理関数名 (vect= ソフトウェア割り込み番号) とすることで、 ソフトウェア割り込み番号 の割り込みが発生したとき、 割り込み処理関数名 を実行します。 ソフトウェア割り込み番号の表を次ページに示します。タイマ RB 割り込みは表より、24 番です。 よって、24 番の割り込みが発生したときに intTRB 関数を実行するよう、「#pragma interrupt」で設定します。
106 行	タイマ RB 割り込みにより実行する関数です。割り込み関数は、引数、戻り値ともに指定することはできません。すなわち、「void 関数名(void)」である必要があります。
108 行	cnt_rb 変数を+1 します。この関数は 1ms ごとに実行されるので、cnt_rb は 1ms ごとに+1 されることとなります。

14. 割り込みによるタイマ(プロジェクト:timer2)

※ソフトウェア割り込み番号

割り込み要因とソフトウェア割り込み番号の関係を、下記に示します。

割り込み要因	ベクタ番地(注1) 番地(L)～番地(H)	ソフトウェア 割り込み番号	割り込み制御 レジスタ	参照先
BRK 命令(注3)	+0～+3(0000h～0003h)	0	—	R8C/Tinyシリーズ ソフトウェアマニュアル
フラッシュメモリレディ	+4～+7(0004h～0007h)	1	FMRDYIC	32. フラッシュメモリ
—(予約)		2～5	—	—
INT4	+24～+27(0018h～001Bh)	6	INT4IC	11.4 INT割り込み
タイマRC	+28～+31(001Ch～001Fh)	7	TRCIC	19. タイマRC
タイマRD0	+32～+35(0020h～0023h)	8	TRD0IC	20. タイマRD
タイマRD1	+36～+39(0024h～0027h)	9	TRD1IC	
タイマRE	+40～+43(0028h～002Bh)	10	TREIC	21. タイマRE
UART2送信/NACK2	+44～+47(002Ch～002Fh)	11	S2TIC	23. シリアルインタフェース (UART2)
UART2受信/ACK2	+48～+51(0030h～0033h)	12	S2RIC	
キー入力	+52～+55(0034h～0037h)	13	KUPIC	11.5 キー入力割り込み
A/D変換	+56～+59(0038h～003Bh)	14	ADIC	28. A/Dコンバータ
シンクロナスシリアルコミュニ ケーションユニット/I ² Cバ スインタフェース(注2)	+60～+63(003Ch～003Fh)	15	SSUIC/ IICIC	25. シンクロナスシリアルコミュニ ケーションユニット(SSU)、 26. I ² Cバスインタフェース
—(予約)		16	—	—
UART0送信	+68～+71(0044h～0047h)	17	S0TIC	22. シリアルインタフェース (UART _i (i=0～1))
UART0受信	+72～+75(0048h～004Bh)	18	S0RIC	
UART1送信	+76～+79(004Ch～004Fh)	19	S1TIC	
UART1受信	+80～+83(0050h～0053h)	20	S1RIC	
INT2	+84～+87(0054h～0057h)	21	INT2IC	11.4 INT割り込み
タイマRA	+88～+91(0058h～005Bh)	22	TRAIC	17. タイマRA
—(予約)		23	—	—
タイマRB	+96～+99(0060h～0063h)	24	TRBIC	18. タイマRB
INT1	+100～+103(0064h～0067h)	25	INT1IC	11.4 INT割り込み
INT3	+104～+107(0068h～006Bh)	26	INT3IC	
—(予約)		27	—	—
—(予約)		28	—	—
INT0	+116～+119(0074h～0077h)	29	INT0IC	11.4 INT割り込み
UART2バス衝突検出	+120～+123(0078h～007Bh)	30	U2BCNIC	23. シリアルインタフェース (UART2)
—(予約)		31	—	—
ソフトウェア(注3)	+128～+131(0080h～0083h)～ +164～+167(00A4h～00A7h)	32～41	—	R8C/Tinyシリーズ ソフトウェアマニュアル
—(予約)		42～49	—	—
電圧監視1/コンパレータA1	+200～+203(00C8h～00CBh)	50	VCMP1IC	6. 電圧検出回路
電圧監視2/コンパレータA2	+204～+207(00CCh～00CFh)	51	VCMP2IC	30. コンパレータA
—(予約)		52～55	—	—
ソフトウェア(注3)	+224～+227(00E0h～00E3h)～ +252～+255(00FCh～00FFh)	56～63		R8C/Tinyシリーズ ソフトウェアマニュアル

注1. INTBレジスタが示す番地からの相対番地です。

注2. SSUICSRレジスタのIICSELビットで選択できます。

注3. Iフラグによる禁止はできません。

今回は、タイマRBを使用して割り込みを発生させるので、表より番号は24番となります。

14. 割り込みによるタイマ(プロジェクト:timer2)

次のように、#pragma interrupt 命令を記述して、「ソフトウェア割り込み番号 24 番が発生したときに、実行する関数は割り込み処理関数名ですよ」ということを、宣言します。

```
#pragma interrupt 割り込み処理関数名(vect=24)
```

関数名のプログラムを記述して、割り込みが発生したときに実行するプログラムを作成します。

```
void 割り込み処理関数名( void )
{
    プログラム
}
```

14.5.4 timer 関数(割り込みを使った時間稼ぎ)

timer 関数は、実行した行で時間稼ぎをする関数です。プロジェクト「timer1」はソフトウェアによるタイマでした。そのため、for 文で使用した 1240 という数値を見つけるのは大変です。また、コンパイラのバージョンの違いやツールチェーンの設定により、時間が変わる可能性があります。今回の timer 関数は、クリスタルの値を基準としているため、正確な計測が可能です。

```
92 : /*****/
93 : /* タイマ本体 */
94 : /* 引数 タイマ値 1=1ms */
95 : /*****/
96 : void timer( unsigned long timer_set )
97 : {
98 :     cnt_rb = 0;
99 :     while( cnt_rb < timer_set );
100 : }
```

98 行	cnt_rb を 0 にクリアします。
99 行	<p>cnt_rb が timer_set より小さいなら、99 行を繰り返し続けます。cnt_rb は割り込みプログラムで 1ms ごとに+1 されます。timer_set は、timer 関数を実行したときに引数でセットした値です。</p> <p>例えば、</p> <pre>timer(500);</pre> <p>と実行したなら、timer_set には 500 が入ります。timer 関数を実行したいちばん最初は、</p> <pre>while(0 < 500);</pre> <p>となり、成り立つので 99 行を繰り返します。1ms 後は、割り込みプログラムで cnt_rb が+1 されるので、</p> <pre>while(1 < 500);</pre> <p>となります。まだ成り立つので、99 行目を繰り返します。timer 関数を実行してから 500ms たったなら、割り込みプログラムも 500 回実行され、cnt_rb は 500 となります。</p> <pre>while(500 < 500);</pre> <p>成り立たなくなり、次の行へ進みます。よって、99 行を実行し終わるまで 500ms かかることとなります。</p>

14.5.5 main 関数

```
39 : void main( void )
40 : {
41 :     init();                /* 初期化                */
42 :     asm( " fset I ");      /* 全体の割り込み許可    */
43 :
44 :     while( 1 ) {
45 :         p6 = 0x55;
46 :         timer( 1000 );
47 :         p6 = 0xaa;
48 :         timer( 1000 );
49 :         p6 = 0x00;
50 :         timer( 1000 );
51 :     }
52 : }
```

42 行

全体の割り込みを許可する命令です。

init 関数内でタイマ RB の割り込みを許可していますが、全体の割り込みを許可しなければ割り込みは発生しません。全体の割り込みを許可する命令は、C 言語で記述することができないため、asm 命令を使ってアセンブリ言語で割り込みを許可する命令を記述しています。

14.5.6 割り込みの発生タイミング

main 関数の while(1)のカッコ内の C 言語をアセンブリ言語に変換すると、下記のようになります。

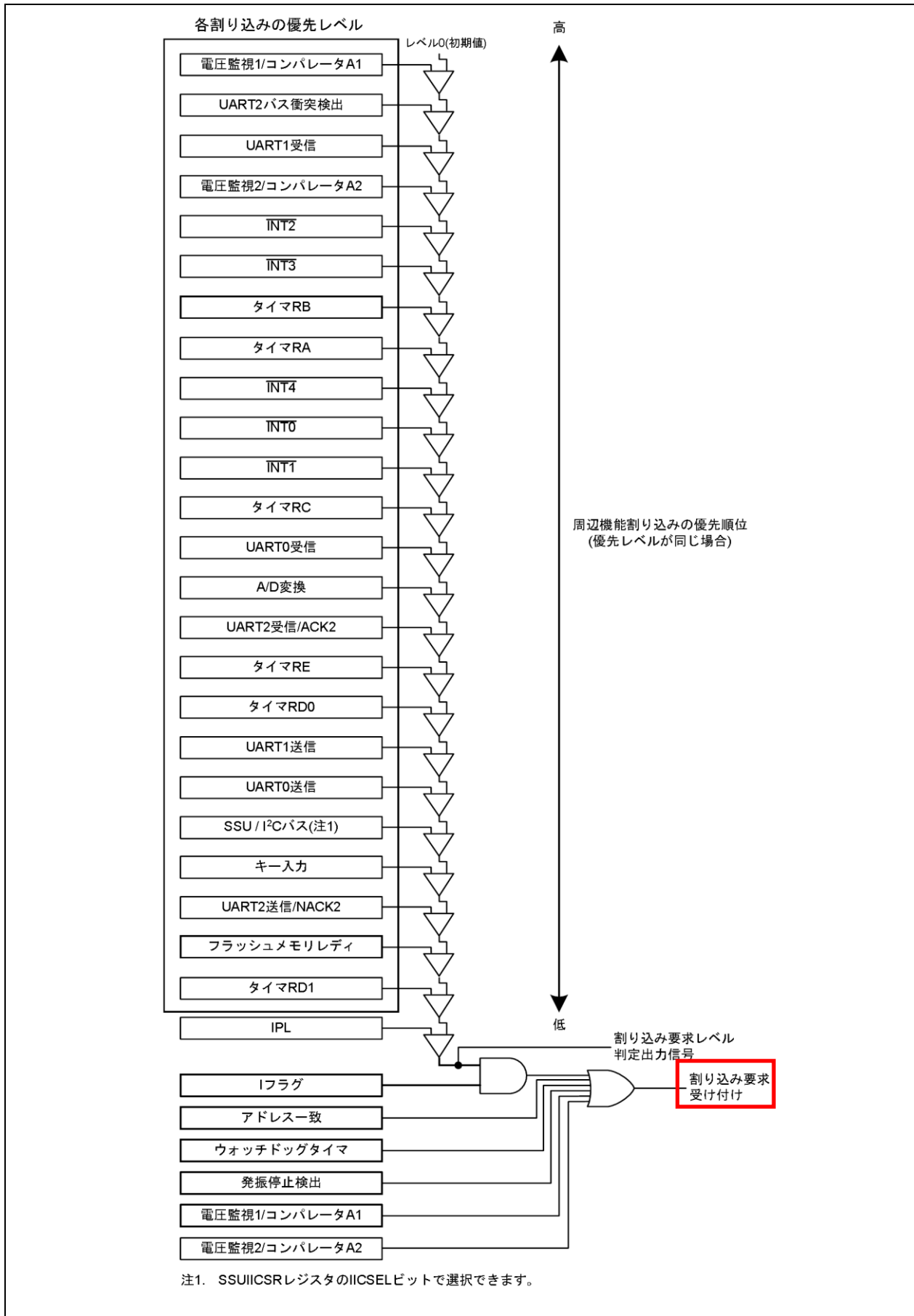
<pre> ; while(1)のカッコ内をアセンブリ言語に ; 変換した内容 L1: mov.w #0001H, R0 jeq L3 mov.b #55H, _p6_addr push.w #0000H push.w #03e8H jsr _timer add.b #04H, SP mov.b #0aaH, _p6_addr (右上へ続く) </pre>	<pre> (左下から続き) push.w #0000H push.w #03e8H jsr _timer add.b #04H, SP mov.b #00H, _p6_addr push.w #0000H push.w #03e8H jsr _timer add.b #04H, SP jmp L1 L3: </pre>
--	---

割り込みがあるかどうかは、アセンブリ言語の命令を 1 行終了後、毎回チェックが行われます。イメージを下記に示します。

<pre> ; while(1)のカッコ内をアセンブリ言語に ; 変換した内容 L1: mov.w #0001H, R0 割り込み要求があるかチェック jeq L3 割り込み要求があるかチェック mov.b #55H, _p6_addr 割り込み要求があるかチェック push.w #0000H 割り込み要求があるかチェック push.w #03e8H 割り込み要求があるかチェック jsr _timer 割り込み要求があるかチェック add.b #04H, SP 割り込み要求があるかチェック mov.b #0aaH, _p6_addr 割り込み要求があるかチェック (右上へ続く) </pre>	<pre> (左下から続き) push.w #0000H 割り込み要求があるかチェック push.w #03e8H 割り込み要求があるかチェック jsr _timer 割り込み要求があるかチェック add.b #04H, SP 割り込み要求があるかチェック mov.b #00H, _p6_addr 割り込み要求があるかチェック push.w #0000H 割り込み要求があるかチェック push.w #03e8H 割り込み要求があるかチェック jsr _timer 割り込み要求があるかチェック add.b #04H, SP 割り込み要求があるかチェック jmp L1 L3: </pre>
--	---

14. 割り込みによるタイマ(プロジェクト:timer2)

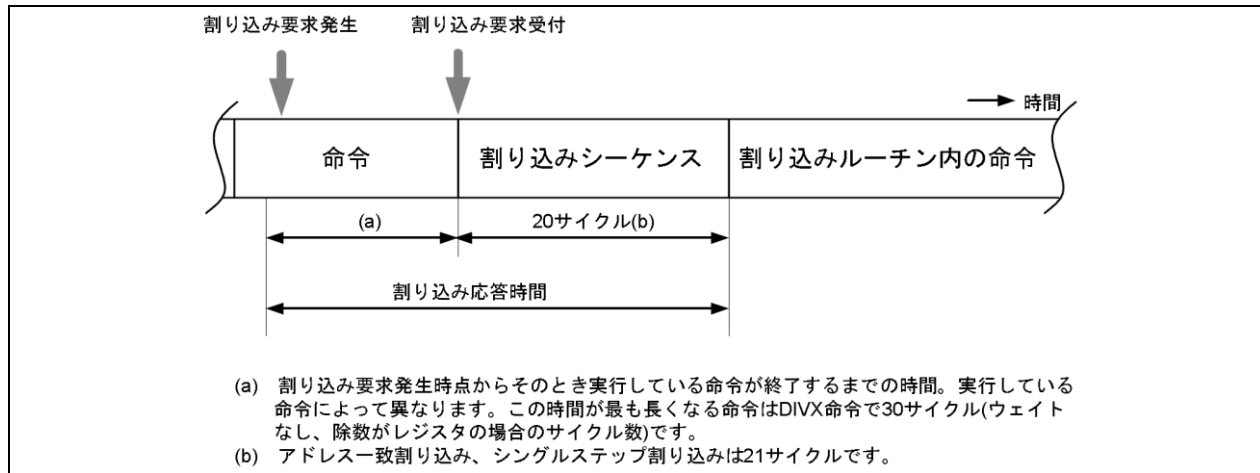
「割り込み要求があるかチェック」の部分は、下記の「割り込み要求受け付け」信号をチェック、「1」なら割り込みありと判断します。



14. 割り込みによるタイマ(プロジェクト:timer2)

「割り込み要求受け付け」が"1"なら、割り込みが発生した割り込みプログラムを実行します。

割り込み応答時間を下記に示します。割り込み応答時間は、割り込み要求が発生してから割り込みルーチン内の最初の命令を実行するまでの時間です。この時間は、割り込み要求発生時点から、そのとき実行している命令が終了するまでの時間(a)と割り込みシーケンスを実行する時間(20 サイクル(b))で構成されます。



本マイコンボードは、20MHz のクリスタルで動作しています。1 サイクルは、

$$1/20\text{MHz}=50\text{ns}$$

です。割り込みシーケンスの時間は、20 サイクルですので、

$$\text{割り込みシーケンスの時間} = 1 \text{ サイクル} \times 20 = 50\text{ns} \times 20 = 1000\text{ns} = 1 \mu\text{s}$$

となります。

割り込みルーチンの処理が終わったら、割り込みシーケンス発生前に実行していた命令の、次の命令から実行を再開します。

15. センサ部の LED へ出力(プロジェクト:sensor_led)

15.1 概要

本章では、ミニマイコンカーVer.2 のセンサ部にある LED(D1~D4) 4 個を点灯／消灯させる方法を説明します。センサ部にある LED は 4 個ですが、マイコンのポートは 3 ビット分しか LED に接続されていません。3 線で 4 個の LED をどのように制御しているか解説します。

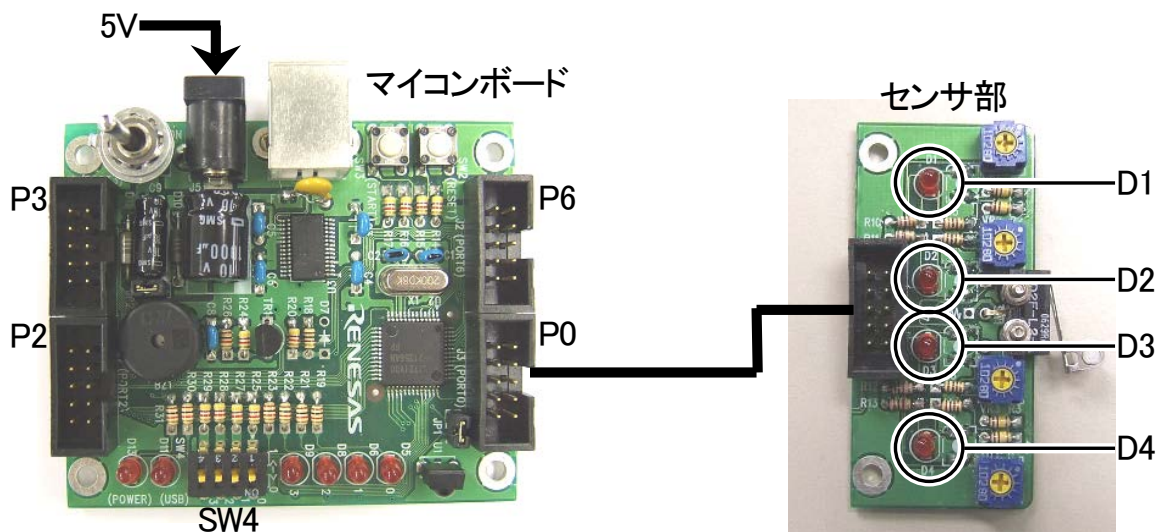
15.2 接続

■使用ポート

マイコンのポート	接続内容
P5_7、P4_5、P4_4、P4_3	マイコンボード上のディップスイッチです。
P1_7、P1_6、P1_5	センサ部の LED が 4 個接続されています。

■接続例

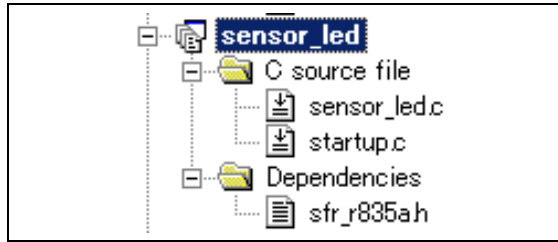
マイコンボードのポート 0(P0)とセンサ部をフラットケーブルで接続します。分離していない場合は、フラットケーブルで接続する必要はありません。



■操作方法

マイコンボードのディップスイッチ(SW4)を ON/OFF すると、それに合わせて LED(D4,D3,D2,D1)が点灯／消灯します。

15.3 プロジェクトの構成



	ファイル名	内容
1	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。このファイルは共通で、どのプロジェクトもこのファイルから実行されます。
2	sensor_led.c	実際に制御するプログラムが書かれています。R8C/35A の内蔵周辺機能(SFR)の初期化も行います。
3	sfr_r835a.h	R8C/35A マイコンの内蔵周辺機能を制御するためのレジスタ(Special Function Registers)を定義したファイルです。

15.4 プログラム「sensor_led.c」

```

1 : /******
2 : /* 対象マイコン R8C/35A */
3 : /* ファイル内容 センサ部のLEDへ値を出力 */
4 : /* バージョン Ver. 1.20 */
5 : /* Date 2010. 04. 19 */
6 : /* Copyright ルネサスマイコンカーラー事務所 */
7 : /* 日立インターメディックス株式会社 */
8 : /******
9 : /*
10 : 入力：マイコンボードのディップスイッチ(4bit)
11 : 出力：センサ部のLED D1～D4(P0_7～P0_5)
12 :
13 : マイコンボードのディップスイッチ(4bit)から入力した状態を、
14 : センサ部のLED(P0_7～P0_5)に出力します。
15 : P0_7～P0_5の3bitの組み合わせで、センサ部の4個のLED(D1～D4)を点灯させます。
16 : 実際の点灯作業は、割り込み内で行います。
17 : */
18 :
19 : /*=====*/
20 : /* インクルード */
21 : /*=====*/
22 : #include "sfr_r835a.h" /* R8C/35A SFRの定義ファイル */
23 :
24 : /*=====*/
25 : /* シンボル定義 */
26 : /*=====*/
27 :
28 : /*=====*/
29 : /* プロトタイプ宣言 */
30 : /*=====*/
31 : void init( void );
32 : unsigned char dipsw_get( void );
33 : void sensorled_out( unsigned char led );
34 :
35 : /*=====*/
36 : /* グローバル変数の宣言 */
37 : /*=====*/
38 : unsigned char sensorled_data; /* sensorledへ出力する値 */
39 :

```

15. センサ部の LED へ出力(プロジェクト:sensor_led)

```

40 : /*****/
41 : /* メインプログラム */
42 : /*****/
43 : void main( void )
44 : {
45 :     unsigned char d;
46 :
47 :     init();                /* 初期化 */
48 :     asm(" fset I ");      /* 全体の割り込み許可 */
49 :
50 :     while( 1 ) {
51 :         d = dipsw_get();
52 :         sensorled_out( d );
53 :     }
54 : }
55 :
56 : /*****/
57 : /* R8C/35A スペシャルファンクションレジスタ(SFR)の初期化 */
58 : /*****/
59 : void init( void )
60 : {
61 :     int i;
62 :
63 :     /* クロックをXINクロック(20MHz)に変更 */
64 :     prc0 = 1;             /* プロテクト解除 */
65 :     cml3 = 1;            /* P4_6, P4_7をXIN-XOUT端子にする */
66 :     cm05 = 0;           /* XINクロック発振 */
67 :     for(i=0; i<50; i++ ); /* 安定するまで少し待つ(約10ms) */
68 :     ocd2 = 0;           /* システムクロックをXINにする */
69 :     prc0 = 0;           /* プロテクトON */
70 :
71 :     /* ポートの入出力設定 */
72 :     prc2 = 1;            /* PDOのプロテクト解除 */
73 :     pd0 = 0xe0;          /* 7-5:LED 4:MicroSW 3-0:Sensor */
74 :     p1 = 0x0f;           /* 3-0:LEDは消灯 */
75 :     pd1 = 0xdf;          /* 5:RXD0 4:TXD0 3-0:LED */
76 :     pd2 = 0xfe;          /* 0:PushSW */
77 :     pd3 = 0xfb;          /* 4: buzzer 2:IR */
78 :     pd4 = 0x83;          /* 7:XOUT 6:XIN 5-3:DIP SW 2:VREF */
79 :     pd5 = 0x40;          /* 7:DIP SW */
80 :     pd6 = 0xff;
81 :
82 :     /* タイマRBの設定 */
83 :     /* 割り込み周期 = 1 / 20[MHz] * (TRBPRE+1) * (TRBPR+1)
84 :                = 1 / (20*10^-6) * 200 * 100
85 :                = 0.001[s] = 1[ms]
86 :     */
87 :     trbmr = 0x00;        /* 動作モード、分周比設定 */
88 :     trbpre = 200-1;      /* プリスケアラレジスタ */
89 :     trbpr = 100-1;       /* プライマリレジスタ */
90 :     trbic = 0x07;        /* 割り込み優先レベル設定 */
91 :     trbcr = 0x01;        /* カウント開始 */
92 : }
93 :
94 : /*****/
95 : /* ディップスイッチ値読み込み */
96 : /* 戻り値 スイッチ値 0~15 */
97 : /*****/
98 : unsigned char dipsw_get( void )
99 : {
100 :     unsigned char sw, sw1, sw2;
101 :
102 :     sw1 = (p5>>4) & 0x08; /* ディップスイッチ読み込み3 */
103 :     sw2 = (p4>>3) & 0x07; /* ディップスイッチ読み込み2,1,0 */
104 :     sw = sw1 | sw2;       /* P5とP4の値を合わせる */
105 :
106 :     return sw;
107 : }
108 :
109 : /*****/
110 : /* センサ部のLED出力 */
111 : /* 引数 LEDへの出力値 0~15 */
112 : /*****/
113 : void sensorled_out( unsigned char led )
114 : {
115 :     /* この関数では、LED出力値を保存するだけ */
116 :     sensorled_data = led;
117 : }
118 :

```


15. センサ部の LED へ出力(プロジェクト:sensor_led)

```

119 : /*****/
120 : /* タイマRB 割り込み処理 */
121 : /*****/
122 : #pragma interrupt intTRB(vect=24)
123 : void intTRB( void )
124 : {
125 :     /* 実際にセンサ部のLEDへ出力する */
126 :     if( p0 & 0x80 ) {
127 :         /* P0_7が"1"なら */
128 :         p0 &= 0x1f; /* P0_7を"0"にして全消灯 */
129 :         if( sensorled_data & 0x8 ) p0 |= 0x40; /* D1点灯 */
130 :         if( sensorled_data & 0x2 ) p0 |= 0x20; /* D3点灯 */
131 :     } else {
132 :         /* P0_7が"0"なら */
133 :         p0 |= 0xe0; /* P0_7を"1"にして全消灯 */
134 :         if( sensorled_data & 0x4 ) p0 &= 0xbf; /* D2点灯 */
135 :         if( sensorled_data & 0x1 ) p0 &= 0xdf; /* D4点灯 */
136 :     }
137 : }
138 :
139 : /*****/
140 : /* end of file */
141 : /*****/

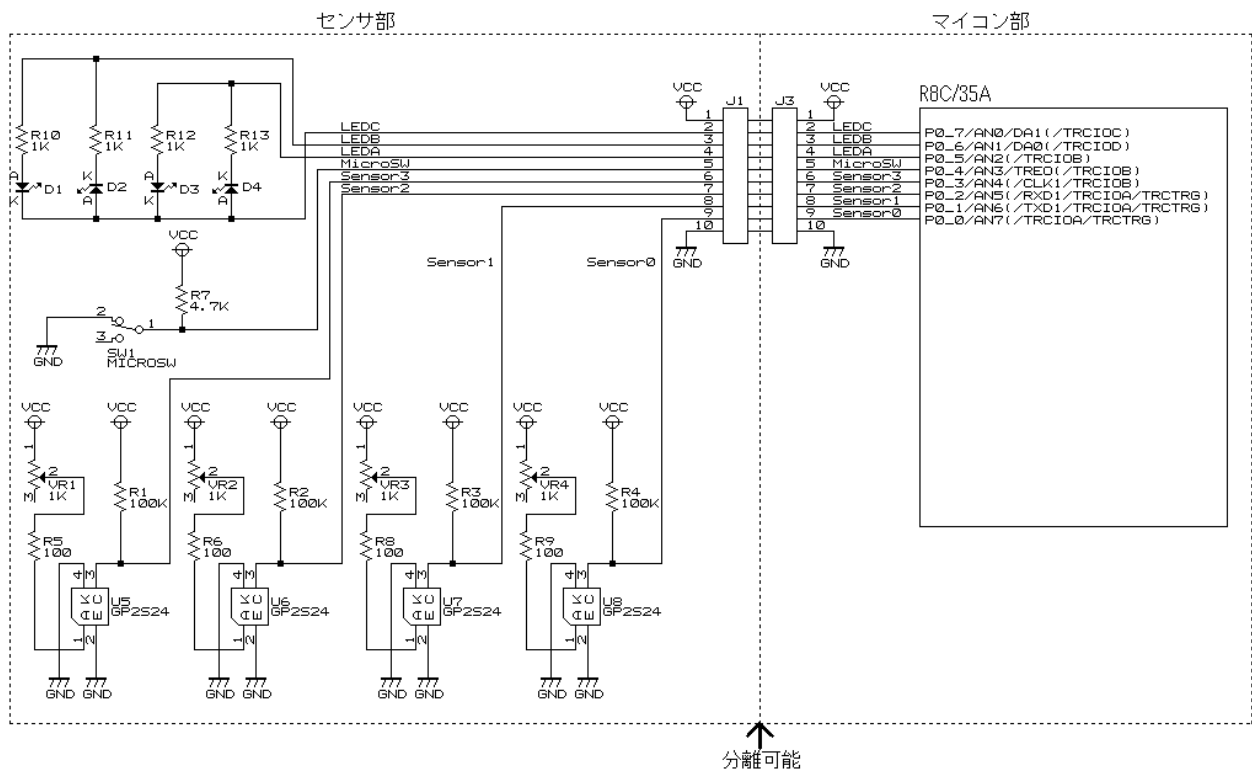
```

15.5 センサ部の LED の回路

15.5.1 マイコンボードとセンサ部の LED の結線

マイコンボードとセンサ部の LED は、下記のように結線されています。

D1、D2、D3、D4の4個のLEDを、P0_7(LED C)、P0_6(LED B)、P0_5(LED A)の3本の信号線で制御しています。



15.5.2 J1 と J3 の詳細

J1 と J3 のピン番号に対する信号名を下表に示します。J1 と J3 はあらかじめパターンで接続されています。マイコン部とセンサ部を分離した場合は、フラットケーブルなどで J1 と J3 を接続してください。

J1		J3 (マイコン部)			
ピン番号	信号名	ピン番号	信号名	入出力設定	説明
1	VCC(+5V)	1	VCC(+5V)		
2	LEDC	2	P0_7	出力	P0_7 は通常の I/O ポートです。
3	LEDB	3	P0_6	出力	P0_6 は通常の I/O ポートです。
4	LEDA	4	P0_5	出力	P0_5 は通常の I/O ポートです。
5	マイクロスイッチ	5	P0_4	入力	P0_4 は通常の I/O ポートです。
6	ラインセンサ 左	6	P0_3	入力	P0_3 は通常の I/O ポートです。
7	ラインセンサ 左中	7	P0_2	入力	P0_2 は通常の I/O ポートです。
8	ラインセンサ 右中	8	P0_1	入力	P0_1 は通常の I/O ポートです。
9	ラインセンサ 右	9	P0_0	入力	P0_0 は通常の I/O ポートです。
10	GND	10	GND		

↑パターンで接続されています。

15.5.3 ポートの信号レベルと LED の関係

P0_7(LED C)、P0_6(LED B)、P0_5(LED A)の信号レベルと D1~D4 の関係を、下表に示します。

P0_7 LEDC	P0_6 LEDB	P0_5 LEDA	D1	D2	D3	D4
0V("0")	0V("0")	0V("0")	消灯	消灯	消灯	消灯
0V("0")	0V("0")	5V("1")	消灯	消灯	点灯	消灯
0V("0")	5V("1")	0V("0")	点灯	消灯	消灯	消灯
0V("0")	5V("1")	5V("1")	点灯	消灯	点灯	消灯
5V("1")	0V("0")	0V("0")	消灯	点灯	消灯	点灯
5V("1")	0V("0")	5V("1")	消灯	点灯	消灯	消灯
5V("1")	5V("1")	0V("0")	消灯	消灯	消灯	点灯
5V("1")	5V("1")	5V("1")	消灯	消灯	消灯	消灯

P0_7 が"0"のときは、D1 と D3 を点灯させることができます。P0_7 が"1"のときは、D2 と D4 を点灯させることができます。プログラムでは、下記のように制御して 4 個の LED を点灯させます。

①	P0_7、P0_6、P0_5 を"0"にして、すべての LED を消灯します。
②	D1 を点灯させたいければ P0_6 を"1"にします。D3 を点灯させたいければ P0_5 を"1"にします。
③	1ms 間待ちます。
④	P0_7、P0_6、P0_5 を"1"にして、すべての LED を消灯します。
⑤	D2 を点灯させたいければ P0_6 を"0"にします。D4 を点灯させたいければ P0_5 を"0"にします。
⑥	1ms 間待ちます。

これを、割り込みプログラムで実行、繰り返し続けて D1~D4 を点灯/消灯処理を行います。

15.6 プログラムの解説

15.6.1 グローバル変数の宣言

グローバル変数を宣言しています。

35	: /*=====*/
36	: /* グローバル変数の宣言 */
37	: /*=====*/
38	: unsigned char sensorled_data; /* sensorledへ出力する値 */

変数 sensorled_data は、センサ部の LED へ出力する値を保存する変数です。LED 出力は、割り込みプログラムで行っています。割り込みプログラムから sensorled_data の値を読み込んで、ポートを制御しています。

15.6.2 sensorled_out 関数

sensorled_out 関数は、センサ部の LED へ値を出力する関数です。

```

109 : /*****/
110 : /* センサ部のLED出力 */
111 : /* 引数 LEDへの出力値 0~15 */
112 : /*****/
113 : void sensorled_out( unsigned char led )
114 : {
115 :     /* この関数では、LED出力値を保存するだけ */
116 :     sensorled_data = led;
117 : }
```

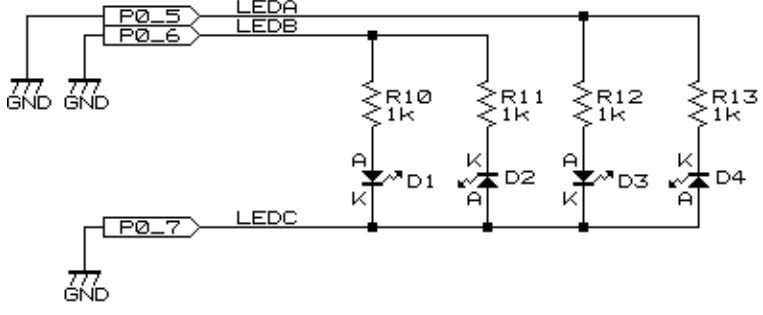
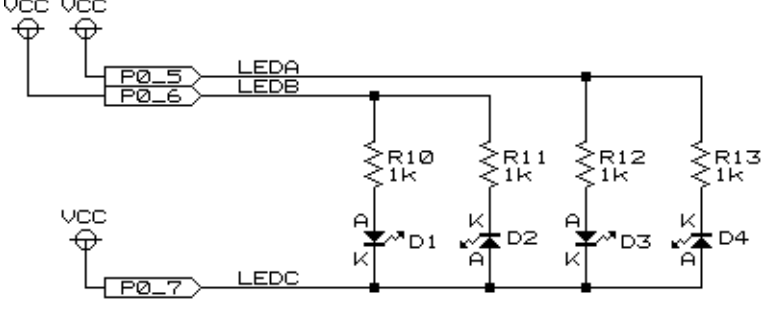
この関数では LED の制御をしていません。実際に LED の制御をしているのは割り込み関数の中です。ここでは、グローバル変数 sensorled_data に LED への出力値 0~15 を保存しています。

15.6.3 割り込み関数

割り込み関数は、1ms ごとに実行されます。この関数内で、グローバル変数 sensorled_data の値を読み込んで、センサ部の LED へ値を出力しています。

```

119 : /*****/
120 : /* タイマRB 割り込み処理 */
121 : /*****/
122 : #pragma interrupt intTRB(vect=24)
123 : void intTRB( void )
124 : {
125 :     /* 実際にセンサ部のLEDへ出力する */
126 :     if( p0 & 0x80 ) {
127 :         /* P0_7が"1"なら */
128 :         p0 &= 0x1f;          /* P0_7を"0"にして全消灯 */
129 :         if( sensorled_data & 0x8 ) p0 |= 0x40;    /* D1点灯 */
130 :         if( sensorled_data & 0x2 ) p0 |= 0x20;    /* D3点灯 */
131 :     } else {
132 :         /* P0_7が"0"なら */
133 :         p0 |= 0xe0;          /* P0_7を"1"にして全消灯 */
134 :         if( sensorled_data & 0x4 ) p0 &= 0xbf;    /* D2点灯 */
135 :         if( sensorled_data & 0x1 ) p0 &= 0xdf;    /* D4点灯 */
136 :     }
137 : }
```

126 行	<p>現在、P0_7 に出力している信号レベルが"1"なら 128～130 行を、"0"なら 133～135 行を実行します。</p>
128 行 ～ 130 行	<p>128 行で P0_7、P0_6、P0_5 を"0"にして、すべての LED を消灯させます(下図)。</p>  <p>D1 を点灯させたければ P0_6 を"1"に、D3 を点灯させたければ P0_5 を"1"にします。消灯のままであれば、何もしません。 P0_7="0"なので、D2 と D4 は P0_6、P0_5 が"0"、"1"のどちらでも消灯のままです。 この状態を次の割り込みがかかるまで、すなわち 1ms 後まで保持します。</p>
133 行 ～ 135 行	<p>133 行で P0_7、P0_6、P0_5 を"1"にして、すべての LED を消灯させます(下図)。</p>  <p>D2 を点灯させたければ P0_6 を"0"に、D4 を点灯させたければ P0_5 を"0"にします。消灯のままであれば、何もしません。 P0_7="1"なので、D1 と D3 は P0_6、P0_5 が"0"、"1"のどちらでも消灯のままです。 この状態を次の割り込みがかかるまで、すなわち 1ms 後まで保持します。</p>

15.6.4 main 関数

```

40 : /*****
41 : /* メインプログラム
42 : *****/
43 : void main( void )
44 : {
45 :     unsigned char d;
46 :
47 :     init();                /* 初期化
48 :     asm( " fset I ");      /* 全体の割り込み許可
49 :
50 :     while( 1 ) {
51 :         d = dipsw_get();
52 :         sensorled_out( d );
53 :     }
54 : }
```

51 行	変数 d にマイコンボードのディップスイッチの値(0~15)を読み込みます。
52 行	センサ部の LED に変数 d の値を出力します。

結果、マイコンボードのディップスイッチの値を、センサ部の LED へ出力します。

15.7 演習

本演習では、LED=センサ部のD1~D4とする。ディップスイッチ=マイコンボード上のディップスイッチとする。

- (1) マイコンボードの J2(ポート6)にセンサ部をつなぎ替えて、ディップスイッチの値を LED へ出力するようにしなさい。
- (2) (1)の状態、LED が 1 秒ごとに"0000"(10 進数で 0)→"0001"(10 進数で 1、以下同じ)→・・・→"1111"→"0000"→"0001"と 1 つずつ増えていくようにしなさい。

16. 外部割り込み (プロジェクト:int_interrupt)

16.1 概要

本章では、外部からの信号で割り込みをかけて(外部割り込み)、割り込みプログラムを実行する方法を説明します。今回は、INT3端子を P3_3 に割り当てて、割り込みを発生させます。

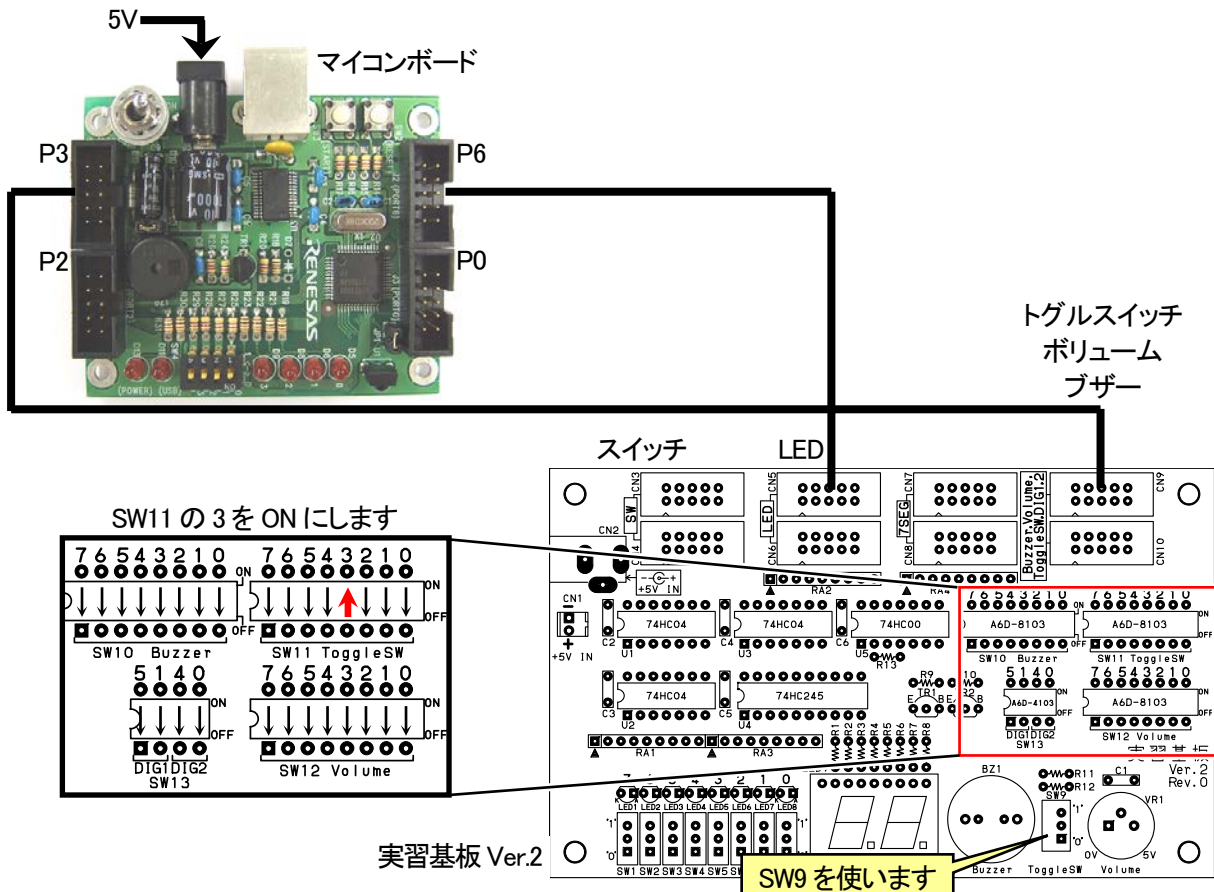
16.2 接続

■使用ポート

マイコンのポート	接続内容
P6 (J2)	実習基板 Ver.2 の LED 部など、出力機器を接続します。
P3_3 (J6)	実習基板 Ver.2 のトグルスイッチ部など、入力機器を接続します。
P1_3、P1_2、P1_1、P1_0	マイコンボード上の LED です。

■接続例

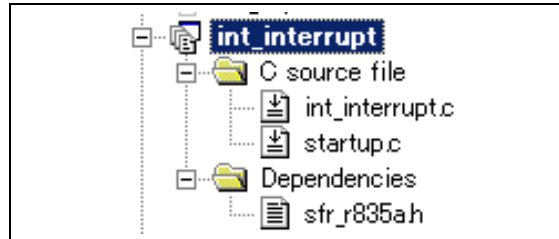
実習基板 Ver.2 を使ったときの接続例を下記に示します。



■操作方法

実習基板 Ver.2 のトグルスイッチ SW2 を上下させると、マイコンボードの LED(D9,D8,D6,D5)の値が「"0000"→"0001"→ … 」と、増えていきます("0"は消灯,"1"は点灯)。その動作と関係なく、ポート 6 に接続している LED が 1 秒ごとに点灯します。ポート 6 に接続している LED の点灯方法は、プロジェクト「timer1」と同じです。

16.3 プロジェクトの構成



	ファイル名	内容
1	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。このファイルは共通で、どのプロジェクトもこのファイルから実行されます。
2	int_interrupt.c	実際に制御するプログラムが書かれています。R8C/35A の内蔵周辺機能(SFR)の初期化も行います。
3	sfr_r835a.h	R8C/35A マイコンの内蔵周辺機能を制御するためのレジスタ(Special Function Registers)を定義したファイルです。

16.4 プログラム「int_interrupt.c」

```

1 :  /******
2 :  /* 対象マイコン  R8C/35A
3 :  /* ファイル内容  INT割り込み
4 :  /* バージョン   Ver. 1.20
5 :  /* Date        2010. 04. 19
6 :  /* Copyright   ルネサスマイコンカーラー事務局
7 :  /*             日立インターメディックス株式会社
8 :  /******
9 :  /*
10:  入力：INT3(P3_3)(実習基板のトグルスイッチ部などチャタリングのない信号)
11:  出力：マイコンボードのLED(4bit)
12:
13:  INT3(P3_3)端子から入力された信号により割り込みプログラムを実行します。
14:  割り込みは、立ち下がり(1→0の瞬間)でかかるように設定します。
15:  INT3割り込みの発生ごとに、マイコンボードのLEDが+1していきます。
16:  メイン関数では、プロジェクト「timer1」のmain関数の内容を実行しています。
17:  */
18:
19:  /*=====*/
20:  /* インクルード
21:  /*=====*/
22:  #include "sfr_r835a.h" /* R8C/35A SFRの定義ファイル */
23:
24:  /*=====*/
25:  /* シンボル定義
26:  /*=====*/
27:
28:  /*=====*/
29:  /* プロトタイプ宣言
30:  /*=====*/
31:  void init( void );
32:  void led_out( unsigned char led );
33:  void timer( unsigned long timer_set );
34:

```


16. 外部割り込み(プロジェクト:int_interrupt)

```

35 : /*=====*/
36 : /* グローバル変数の宣言 */
37 : /*=====*/
38 : unsigned char cnt_int3; /* INT3割り込みごとに+1 */
39 :
40 : /*=====*/
41 : /* メインプログラム */
42 : /*=====*/
43 : void main( void )
44 : {
45 :     unsigned char d;
46 :
47 :     init(); /* 初期化 */
48 :     asm(" fset I "); /* 全体の割り込み許可 */
49 :
50 :     while( 1 ) {
51 :         p6 = 0x55;
52 :         timer( 1000 );
53 :         p6 = 0xaa;
54 :         timer( 1000 );
55 :         p6 = 0x00;
56 :         timer( 1000 );
57 :     }
58 : }
59 :
60 : /*=====*/
61 : /* R8C/35A スペシャルファンクションレジスタ(SFR)の初期化 */
62 : /*=====*/
63 : void init( void )
64 : {
65 :     int i;
66 :
67 :     /* クロックをXINクロック(20MHz)に変更 */
68 :     prc0 = 1; /* プロテクト解除 */
69 :     cm13 = 1; /* P4_6,P4_7をXIN-XOUT端子にする*/
70 :     cm05 = 0; /* XINクロック発振 */
71 :     for(i=0; i<50; i++ ); /* 安定するまで少し待つ(約10ms) */
72 :     ocd2 = 0; /* システムクロックをXINにする */
73 :     prc0 = 0; /* プロテクトON */
74 :
75 :     /* ポートの入出力設定 */
76 :     prc2 = 1; /* PD0のプロテクト解除 */
77 :     pd0 = 0xe0; /* 7-5:LED 4:MicroSW 3-0:Sensor */
78 :     p1 = 0x0f; /* 3-0:LEDは消灯 */
79 :     pd1 = 0xdf; /* 5:RXD0 4:TXD0 3-0:LED */
80 :     pd2 = 0xfe; /* 0:PushSW */
81 :     pd3 = 0xf3; /* 4:Buzzer 3:INT3 2:IR */
82 :     pd4 = 0x80; /* 7:XOUT 6:XIN 5-3:DIP SW 2:VREF*/
83 :     pd5 = 0x40; /* 7:DIP SW */
84 :     pd6 = 0xff; /* LEDなど出力 */
85 :
86 :     /* INT0~4割り込み設定(今回はINT3を設定) */
87 :     intsr = 0x00; /* INT1~3の入力端子設定 */
88 :     inten = 0x40; /* INT0~3の外部入力許可設定 */
89 :     inten1 = 0x00; /* INT4の外部入力許可設定 */
90 :     intf = 0xc0; /* INT0~3の入力フィルタ選択 */
91 :     intf1 = 0x00; /* INT4の入力フィルタ選択 */
92 :
93 :     int0ic = 0x00; /* INT0割り込み優先レベル設定 */
94 :     int1ic = 0x00; /* INT1割り込み優先レベル設定 */
95 :     int2ic = 0x00; /* INT2割り込み優先レベル設定 */
96 :     int3ic = 0x07; /* INT3割り込み優先レベル設定 */
97 :     int4ic = 0x00; /* INT4割り込み優先レベル設定 */
98 : }
99 :
100 : /*=====*/
101 : /* マイコン部のLED出力 */
102 : /* 引数 スイッチ値 0~15 */
103 : /*=====*/
104 : void led_out( unsigned char led )
105 : {
106 :     unsigned char data;
107 :
108 :     led = ~led;
109 :     led &= 0x0f;
110 :     data = p1 & 0xf0;
111 :     p1 = data | led;
112 : }
113 :
114 : /*=====*/
115 : /* タイマ本体 */
116 : /* 引数 タイマ値 1=1ms */
117 : /*=====*/
118 : void timer( unsigned long timer_set )
119 : {
120 :     int i;
121 :
122 :     do {
123 :         for( i=0; i<1240; i++ );
124 :     } while( timer_set-- );
125 : }

```

16. 外部割り込み(プロジェクト:int_interrupt)

```

126 :
127 : /*****/
128 : /* INT3 割り込み処理 */
129 : /*****/
130 : #pragma interrupt intINT3(vect=26)
131 : void intINT3( void )
132 : {
133 :     cnt_int3++;
134 :     led_out( cnt_int3 );
135 : }
136 :
137 : /*****/
138 : /* end of file */
139 : /*****/
    
```

16.5 プログラムの解説

今回は、 $\overline{\text{INT3}}$ 端子を P3_3 に割り当てて使用します。エッジは、立ち下がりエッジとします。

16.5.1 init 関数($\overline{\text{INT3}}$ 割り込みの設定)

(1) $\overline{\text{INT0}}\sim\overline{\text{INT4}}$ 割り込み

$\overline{\text{INT0}}\sim\overline{\text{INT4}}$ 割り込みとは、 $\overline{\text{INT0}}\sim\overline{\text{INT4}}$ 端子から入力されるエッジによって発生する割り込みのことです。エッジの種類を、下表に示します。

立ち上がりエッジ	"0"→"1"になる瞬間を検出することです。
立ち下がりエッジ	"1"→"0"になる瞬間を検出することです。
両エッジ	立ち上がりエッジと立ち下がりエッジの両方を検出することです。

設定したエッジの信号があると、割り込みが発生します。

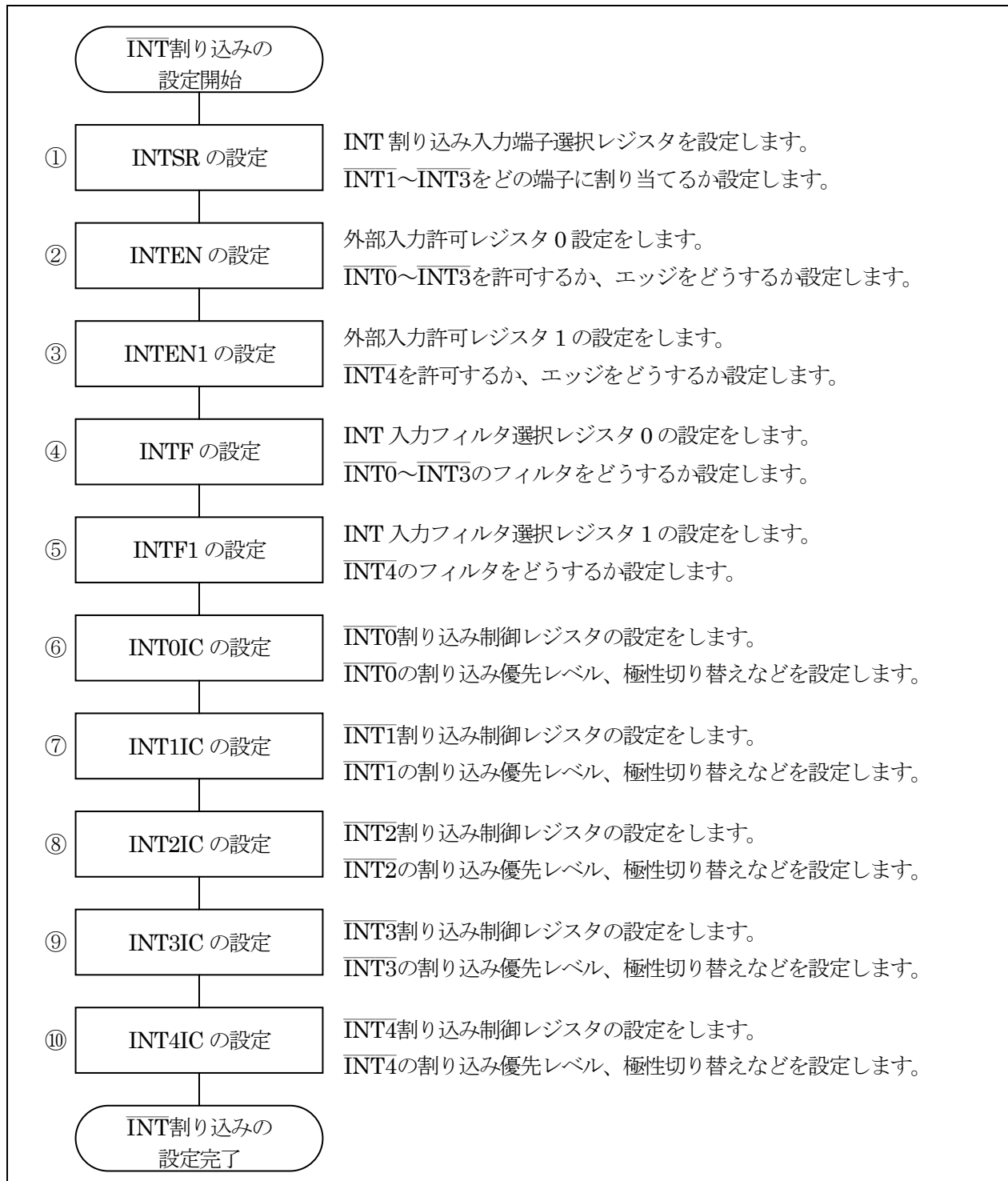
(2) $\overline{\text{INT0}}\sim\overline{\text{INT4}}$ 割り込み端子

$\overline{\text{INT0}}\sim\overline{\text{INT4}}$ 割り込み端子に割り当てることができる端子を下表に示します。

端子名	割り当てることができる端子
$\overline{\text{INT0}}$ 端子	P4_5 に設定可能です。
$\overline{\text{INT1}}$ 端子	P1_5、P1_7、P2_0、P3_2、P3_6 のどれかに設定可能です。 設定は、INT 割り込み入力端子選択レジスタ(INTSR)で行います。
$\overline{\text{INT2}}$ 端子	P3_2、P6_6 のどれかに設定可能です。 設定は、INT 割り込み入力端子選択レジスタ(INTSR)で行います。
$\overline{\text{INT3}}$ 端子	P3_3、P6_7 のどれかに設定可能です。 設定は、INT 割り込み入力端子選択レジスタ(INTSR)で行います。
$\overline{\text{INT4}}$ 端子	P6_5 に設定可能です。

(3) INT3割り込みの設定

レジスタの設定手順を下記に示します。



16. 外部割り込み(プロジェクト:int_interrupt)

①INT 割り込み入力端子選択レジスタ(INTSR:INT function select register)の設定

INT 割り込み入力端子選択レジスタを設定します。 $\overline{\text{INT1}}$ ～ $\overline{\text{INT3}}$ をどの端子に割り当てるか設定します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7,6	$\overline{\text{INT3}}$ 端子選択ビット bit7: int3sel1 bit6: int3sel0	00:P3_3 に割り当てる 01:設定しないでください 10:P6_7 に割り当てる 11:設定しないでください 今回は、P3_3 に割り当てるので"00"に設定します。	00
bit5		"0"を設定	0
bit4	$\overline{\text{INT2}}$ 端子選択ビット int2sel0	0:P6_6 に割り当てる 1:P3_2 に割り当てる 今回、 $\overline{\text{INT2}}$ は使用しませんが"0"を設定しておきます。	0
bit3,2,1	$\overline{\text{INT1}}$ 端子選択ビット bit3: int1sel2 bit2: int1sel1 bit1: int1sel0	000:P1_7 に割り当てる 001:P1_5 に割り当てる 010:P2_0 に割り当てる 011:P3_6 に割り当てる 100:P3_2 に割り当てる 上記以外:設定しないでください 今回、 $\overline{\text{INT1}}$ は使用しませんが"000"を設定しておきます。	000
bit0		"0"を設定	0

INT 割り込み入力端子選択レジスタ(INTSR)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	0	0	0	0
16 進数	0				0			

②外部入力許可レジスタ 0(INTEN: External interrupt enable register)の設定

外部入力許可レジスタ 0 設定をします。 $\overline{\text{INT0}} \sim \overline{\text{INT3}}$ を許可するか、エッジをどうするか設定します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7	$\overline{\text{INT3}}$ 入力極性選択ビット(注 1、2) int3pl	0:片エッジ 1:両エッジ 今回は、片エッジを選択します。立ち上がりか立ち下 がりかは、INT3IC で設定します。	0
bit6	$\overline{\text{INT3}}$ 入力許可ビット int3en	0:禁止 1:許可 今回は $\overline{\text{INT3}}$ を使いますので、許可します。	1
bit5	$\overline{\text{INT2}}$ 入力極性選択ビット(注 1、2) int2pl	0:片エッジ 1:両エッジ	0
bit4	$\overline{\text{INT2}}$ 入力許可ビット int2en	0:禁止 1:許可 今回は $\overline{\text{INT2}}$ を使いませんので禁止にしておきます。	0
bit3	$\overline{\text{INT1}}$ 入力極性選択ビット(注 1、2) int1pl	0:片エッジ 1:両エッジ	0
bit2	$\overline{\text{INT1}}$ 入力許可ビット int1en	0:禁止 1:許可 今回は $\overline{\text{INT1}}$ を使いませんので禁止にしておきます。	0
bit1	$\overline{\text{INT0}}$ 入力極性選択ビット(注 1、2) int0pl	0:片エッジ 1:両エッジ	0
bit0	$\overline{\text{INT0}}$ 入力許可ビット int0en	0:禁止 1:許可 今回は $\overline{\text{INT0}}$ を使いませんので禁止にしておきます。	0

注 1. INTiPL ビット(i=0~3)を“1”(両エッジ)にする場合、INTiIC レジスタの POL ビットを“0”(立ち下がりエッジを
選択)にしてください。

注 2. INTiPL ビットを変更すると、INTiIC レジスタの IR ビットが“1”(割り込み要求あり)になることがあります。詳し
くはハードウェアマニュアルの「11.8.4 割り込み要因の変更」を参照してください。

外部入力許可レジスタ 0(INTEN)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	1	0	0	0	0	0	0
16 進数	4				0			

③外部入力許可レジスタ 1(INTEN1:external input enable register 1)の設定

外部入力許可レジスタ 1 設定をします。 $\overline{\text{INT4}}$ を許可するか、エッジをどうするか設定します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7~2		"000000"を設定	0000 00
bit1	$\overline{\text{INT4}}$ 入力極性選択ビット(注 1、2) int4pl	0:片エッジ 1:両エッジ	0
bit0	$\overline{\text{INT4}}$ 入力許可ビット int4en	0:禁止 1:許可 今回は $\overline{\text{INT4}}$ を使いませんので禁止にしておきます。	0

注 1. INT4PL ビットを“1”(両エッジ)にする場合、INT4IC レジスタの POL ビットを“0”(立ち下がりエッジを選択)にしてください。

注 2. INT4PL ビットを変更すると、INT4IC レジスタの IR ビットが“1”(割り込み要求あり)になることがあります。詳しくはハードウェアマニュアルの「11.8.4 割り込み要因の変更」を参照してください。

外部入力許可レジスタ 1(INTEN1)の設定値を下記に示します。

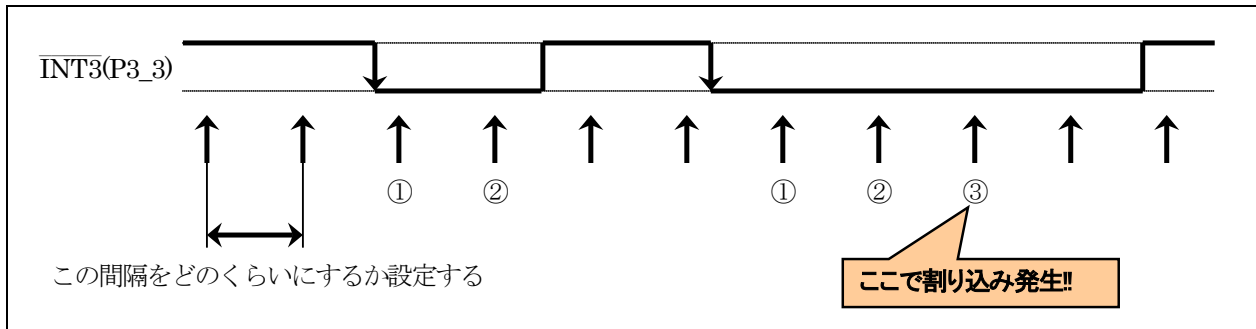
bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	0	0	0	0
16 進数	0				0			

④INT 入力フィルタ選択レジスタ 0 (INTF:INT0 input filter select register)の設定

INT 入力フィルタ選択レジスタ 0 の設定をします。 $\overline{\text{INT0}} \sim \overline{\text{INT3}}$ のフィルタをどうするか設定します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7,6	$\overline{\text{INT3}}$ 入力フィルタ選択ビット bit7:int3f1 bit6:int3f0	00:フィルタなし 01:フィルタあり、f1(1/20MHz=50ns)でサンプリング 10:フィルタあり、f8(8/20MHz=400ns)でサンプリング 11:フィルタあり、f32(32/20MHz=1600ns)でサンプリング 今回は、f32 でサンプリングします。	11
bit5,4	$\overline{\text{INT2}}$ 入力フィルタ選択ビット bit5:int2f1 bit4:int2f0	00:フィルタなし 01:フィルタあり、f1(1/20MHz=50ns)でサンプリング 10:フィルタあり、f8(8/20MHz=400ns)でサンプリング 11:フィルタあり、f32(32/20MHz=1600ns)でサンプリング	00
bit3,2	$\overline{\text{INT1}}$ 入力フィルタ選択ビット bit3:int1f1 bit2:int1f0	00:フィルタなし 01:フィルタあり、f1(1/20MHz=50ns)でサンプリング 10:フィルタあり、f8(8/20MHz=400ns)でサンプリング 11:フィルタあり、f32(32/20MHz=1600ns)でサンプリング	00
bit1,0	$\overline{\text{INT0}}$ 入力フィルタ選択ビット bit1:int0f1 bit0:int0f0	00:フィルタなし 01:フィルタあり、f1(1/20MHz=50ns)でサンプリング 10:フィルタあり、f8(8/20MHz=400ns)でサンプリング 11:フィルタあり、f32(32/20MHz=1600ns)でサンプリング	00

入力フィルタとは、設定した間隔で信号を読み込んで、3 回一致した時点で割り込みを発生させる機能です。



今回、「f32でサンプリング」にしました。これは、1600nsごとに読み込んで3回連続して“0”なら、信号が“0”になったと判断して割り込みを発生させます。よって、立ち下がりエッジから約 4800ns(1600ns×3)後に割り込みがかかります。

INT 入力フィルタ選択レジスタ 0 (INTF)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	1	1	0	0	0	0	0	0
16 進数	c				0			

⑤INT 入力フィルタ選択レジスタ 1 (INTF1:INT input filter select register 1)の設定

INT 入力フィルタ選択レジスタ 1 の設定をします。 $\overline{\text{INT4}}$ のフィルタをどうするか設定します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7~2		"000000"を設定	000000
bit1,0	$\overline{\text{INT4}}$ 入力フィルタ選択ビット bit1: int4f1 bit0: int4f0	00:フィルタなし 01:フィルタあり、f1(1/20MHz=50ns)でサンプリング 10:フィルタあり、f8(8/20MHz=400ns)でサンプリング 11:フィルタあり、f32(32/20MHz=1600ns)でサンプリング	00

INT 入力フィルタ選択レジスタ 1 (INTF1)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	0	0	0	0
16 進数	0				0			

⑨INT3割り込み制御レジスタ(INT3IC:INT3 interrupt control register)の設定

INT3割り込み制御レジスタの設定をします。INT3の割り込み優先レベル、極性切り替えなどを設定します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7~5		"000"を設定	000
bit4	極性切り替えビット(注 3) pol_intlic	0:立ち下がりエッジを選択 1:立ち上がりエッジを選択(注 2) 立ち下がりエッジを選択します。	0
bit3	割り込み要求ビット(注 1) ir_intlic	0:割り込み要求なし 1:割り込み要求あり 書き込みは"0"のみです。	0
bit2~0	割り込み優先レベル選択ビット bit2:ilvl2_intlic bit1:ilvl1_intlic bit0:ilvl0_intlic	000:レベル 0 (割り込み禁止) 001:レベル 1 010:レベル 2 011:レベル 3 100:レベル 4 101:レベル 5 110:レベル 6 111:レベル 7 他の割り込みが同時に発生した場合、レベルの高い割り込みが優先されます。割り込みを 2 つ以上使う場合は、どちらを優先させるか決めて設定します。今回は INT3割り込みだけなのでレベル 1~7 のどれを設定しても構いません。一応、レベルのいちばん高い"111"を設定します。	111

注 1. IR ビットは"0"のみ書けます("1"を書かないでください)。

注 2. INTEN レジスタの INTiPL ビットが"1"(両エッジの場合、POL ビットを"0"(立ち下がりエッジを選択)にしてください。

注 3. POL ビットを変更すると、IR ビットが"1"(割り込み要求あり)になることがあります。詳しくは、ハードウェアマニュアルの「11.8.4 割り込み要因の変更」を参照してください。

割り込み制御レジスタの変更は、そのレジスタに対応する割り込み要求が発生しない箇所で行ってください。詳しくは、ハードウェアマニュアルの「11.8.5 割り込み制御レジスタの変更」を参照してください。

INT3割り込み制御レジスタ(INT3IC)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	0	1	1	1
16 進数	0				7			

16. 外部割り込み (プロジェクト:int_interrupt)

- ⑥ $\overline{\text{INT0}}$ 割り込み制御レジスタ(INT0IC)の設定
- ⑦ $\overline{\text{INT1}}$ 割り込み制御レジスタ(INT1IC)の設定
- ⑧ $\overline{\text{INT2}}$ 割り込み制御レジスタ(INT2IC)の設定
- ⑩ $\overline{\text{INT4}}$ 割り込み制御レジスタ(INT4IC)の設定

$\overline{\text{INT0}}$ 、 $\overline{\text{INT1}}$ 、 $\overline{\text{INT2}}$ 、 $\overline{\text{INT4}}$ 割り込み制御レジスタの設定をします。設定内容は INT3IC と同じです。今回は $\overline{\text{INT0}}$ 、 $\overline{\text{INT1}}$ 、 $\overline{\text{INT2}}$ 、 $\overline{\text{INT4}}$ は使いませんので、レベルは 0(割り込み禁止)に設定します。

$\overline{\text{INT0}}$ 割り込み制御レジスタ(INT0IC)、 $\overline{\text{INT1}}$ 割り込み制御レジスタ(INT1IC)、 $\overline{\text{INT2}}$ 割り込み制御レジスタ(INT2IC)、 $\overline{\text{INT4}}$ 割り込み制御レジスタ(INT4IC)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	0	0	0	0
16 進数	0				0			

16.5.2 intINT3 関数 (INT3端子に立ち下がりエッジの信号が入力されたときに実行される関数)

先の設定で、 $\overline{\text{INT3}}$ 端子(P3_3)に立ち下がりエッジの信号が入力されると、割り込みを発生させる設定にしました。intINT3 関数は、この割り込みが発生したときに実行される関数です。

```

127 : /*****/
128 : /* INT3 割り込み処理 */
129 : /*****/
130 : #pragma interrupt intINT3(vect=26)
131 : void intINT3( void )
132 : {
133 :     cnt_int3++;
134 :     led_out( cnt_int3 );
135 : }
    
```

130 行	#pragma interrupt 割り込み処理関数名 (vect= ソフトウェア割り込み番号) とすることで、 ソフトウェア割り込み番号 の割り込みが発生したとき、 割り込み処理関数名 を実行します。 ソフトウェア割り込み番号の表より、 $\overline{\text{INT3}}$ 割り込みは、26 番です。 よって、26 番の割り込みが発生したときに intINT3 関数を実行するよう、「#pragma interrupt」で設定します。
131 行	$\overline{\text{INT3}}$ 割り込みにより実行する関数です。今回の設定では、P3_3 の立ち下がりエッジ信号ごとに実行されます。割り込み関数は、引数、戻り値ともに指定することはできません。
133 行	変数 cnt_int3 を+1 します。
134 行	マイコンボード上の LED へ変数 cnt_int3 の値を出力します。

※ソフトウェア割り込み番号

割り込み要因とソフトウェア割り込み番号の関係を、下記に示します。

割り込み要因	ベクタ番地(注1) 番地(L)～番地(H)	ソフトウェア 割り込み番号	割り込み制御 レジスタ	参照先
BRK 命令(注3)	+0～+3(0000h～0003h)	0	—	R8C/Tinyシリーズ ソフトウェアマニュアル
フラッシュメモリレディ	+4～+7(0004h～0007h)	1	FMRDYIC	32. フラッシュメモリ
—(予約)		2～5	—	—
INT4	+24～+27(0018h～001Bh)	6	INT4IC	11.4 INT割り込み
タイマRC	+28～+31(001Ch～001Fh)	7	TRCIC	19. タイマRC
タイマRD0	+32～+35(0020h～0023h)	8	TRD0IC	20. タイマRD
タイマRD1	+36～+39(0024h～0027h)	9	TRD1IC	
タイマRE	+40～+43(0028h～002Bh)	10	TREIC	21. タイマRE
UART2送信/NACK2	+44～+47(002Ch～002Fh)	11	S2TIC	23. シリアルインタフェース (UART2)
UART2受信/ACK2	+48～+51(0030h～0033h)	12	S2RIC	
キー入力	+52～+55(0034h～0037h)	13	KUPIC	11.5 キー入力割り込み
A/D変換	+56～+59(0038h～003Bh)	14	ADIC	28. A/Dコンバータ
シンクロナスシリアルコミュニ ケーションユニット/I ² Cバ スインタフェース(注2)	+60～+63(003Ch～003Fh)	15	SSUIC/ IICIC	25. シンクロナスシリアルコミュニ ケーションユニット(SSU)、 26. I ² Cバスインタフェース
—(予約)		16	—	—
UART0送信	+68～+71(0044h～0047h)	17	S0TIC	22. シリアルインタフェース (UART _i (i=0～1))
UART0受信	+72～+75(0048h～004Bh)	18	S0RIC	
UART1送信	+76～+79(004Ch～004Fh)	19	S1TIC	
UART1受信	+80～+83(0050h～0053h)	20	S1RIC	
INT2	+84～+87(0054h～0057h)	21	INT2IC	11.4 INT割り込み
タイマRA	+88～+91(0058h～005Bh)	22	TRAIC	17. タイマRA
—(予約)		23	—	—
タイマRB	+96～+99(0060h～0063h)	24	TRBIC	18. タイマRB
INT1	+100～+103(0064h～0067h)	25	INT1IC	11.4 INT割り込み
INT3	+104～+107(0068h～006Bh)	26	INT3IC	
—(予約)		27	—	—
—(予約)		28	—	—
INT0	+116～+119(0074h～0077h)	29	INT0IC	11.4 INT割り込み
UART2バス衝突検出	+120～+123(0078h～007Bh)	30	U2BCNIC	23. シリアルインタフェース (UART2)
—(予約)		31	—	—
ソフトウェア(注3)	+128～+131(0080h～0083h)～ +164～+167(00A4h～00A7h)	32～41	—	R8C/Tinyシリーズ ソフトウェアマニュアル
—(予約)		42～49	—	—
電圧監視1/コンパレータA1	+200～+203(00C8h～00CBh)	50	VCMP1IC	6. 電圧検出回路
電圧監視2/コンパレータA2	+204～+207(00CCh～00CFh)	51	VCMP2IC	30. コンパレータA
—(予約)		52～55	—	—
ソフトウェア(注3)	+224～+227(00E0h～00E3h)～ +252～+255(00FCh～00FFh)	56～63		R8C/Tinyシリーズ ソフトウェアマニュアル

注1. INTBレジスタが示す番地からの相対番地です。

注2. SSUICSRレジスタのIICSELビットで選択できます。

注3. Iフラグによる禁止はできません。

今回は、INT3を使用して割り込みを発生させるので、表より番号は26番となります。

16.5.3 main 関数

```

40 : /*****
41 : /* メインプログラム */
42 : *****/
43 : void main( void )
44 : {
45 :     unsigned char d;
46 :
47 :     init();                /* 初期化 */
48 :     asm( " fset I ");     /* 全体の割り込み許可 */
49 :
50 :     while( 1 ) {
51 :         p6 = 0x55;
52 :         timer( 1000 );
53 :         p6 = 0xaa;
54 :         timer( 1000 );
55 :         p6 = 0x00;
56 :         timer( 1000 );
57 :     }
58 : }

```

48 行	<p>全体の割り込みを許可する命令です。</p> <p>init 関数内で INT3 割り込みを許可していますが、全体の割り込みを許可しなければ割り込みは発生しません。全体の割り込みを許可する命令は、C 言語で記述することができないため、asm 命令を使ってアセンブリ言語で割り込みを許可する命令を記述しています。</p>
50 行 ～ 57 行	<p>プロジェクト「timer1」と同じプログラムです。プロジェクト「timer1」の解説を参照してください。</p>

16.6 演習

- (1) INT3 端子に立ち上がりエッジの信号が入力されると、割り込みが発生するようにしなさい。
- (2) INT1 割り込みを P3.2 端子に設定して、立ち下がりエッジで割り込みがかかるようにしなさい。割り込みプログラムはサンプルプログラムと同じとする。
(実習基板 Ver.2 の SW11 は、2 のみ上にしてください)

17. A/D コンバータ(単発モード)(プロジェクト:ad)

17.1 概要

本章では、0~5V の電圧をマイコンの A/D コンバータで読み込む方法を説明します。A/D 変換した結果は、マイコンボードの LED に出します。今回の A/D 変換は、単発モードを使います。

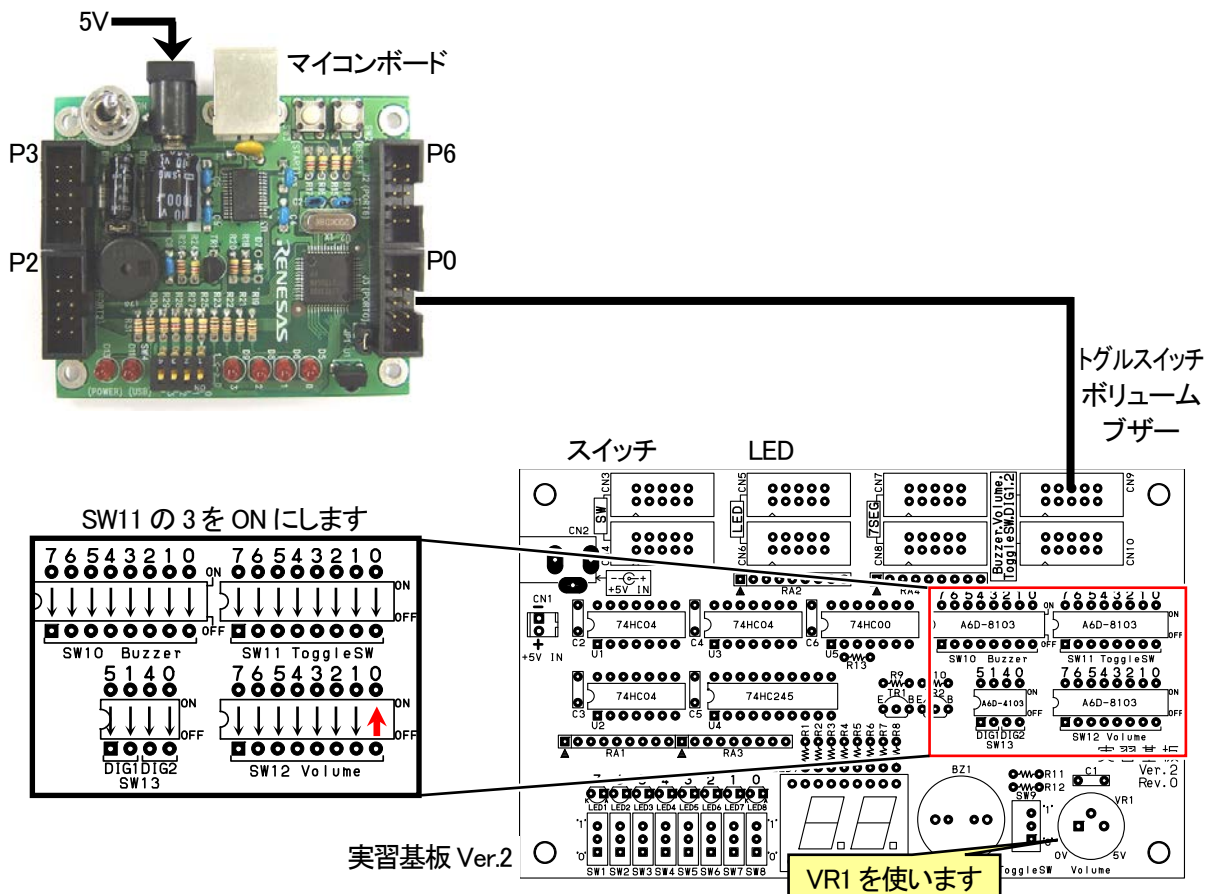
17.2 接続

■使用ポート

マイコンのポート	接続内容
P0_0 (J3)	ボリュームやアナログセンサなど 0~5V を出力する機器を接続します。実習基板 Ver.2 またはセンサ部も接続可能です。
P1_3、P1_2、P1_1、P1_0	マイコンボード上の LED です。

■接続例 1

実習基板 Ver.2 を使ったときの接続例を下記に示します。

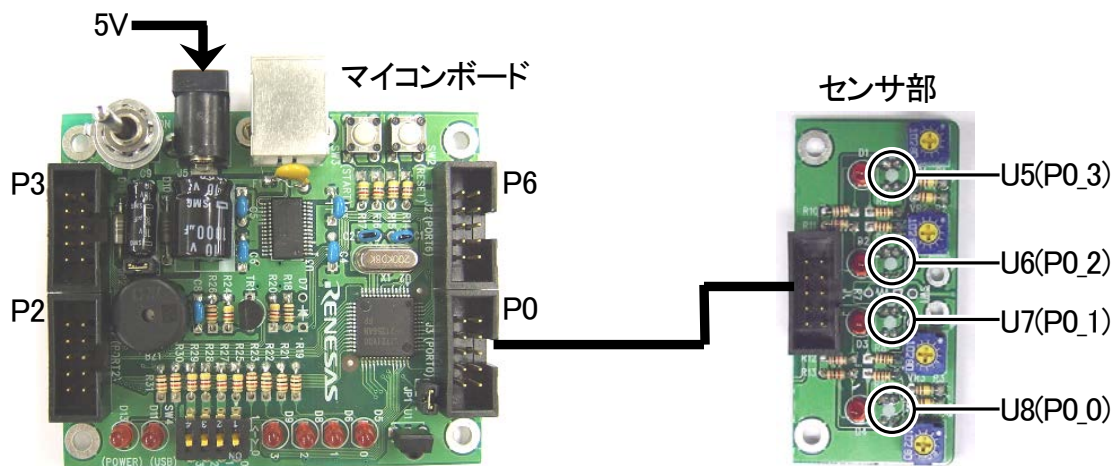


■操作方法(接続例 1 のとき)

実習基板 Ver.2 のボリューム VR1 のつまみを左右に回します。ボリュームから出力された 0~5V の電圧をマイコンの P0_0 から読み込み、A/D 変換した値をマイコンボードの LED に出力します。

■接続例 2

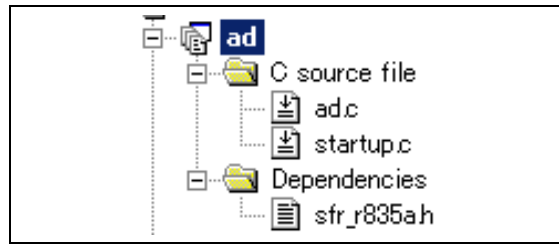
マイコンボードのポート 0(P0)とセンサ部をフラットケーブルで接続します。分離していない場合は、フラットケーブルで接続する必要はありません。センサは 4 個ありますが、今回電圧を読み込むのは、U8(P0_0)のフォトインタラプタです。



■操作方法(接続例 2 のとき)

センサ部の U8 の下部を白色や灰色や黒色に近づけます。フォトインタラプタ U8 から出力された 0~5V の電圧をマイコンの P0_0 から読み込み、A/D 変換した値をマイコンボードの LED に出力します。

17.3 プロジェクトの構成



	ファイル名	内容
1	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。このファイルは共通で、どのプロジェクトもこのファイルから実行されます。
2	adc	実際に制御するプログラムが書かれています。R8C/35A の内蔵周辺機能(SFR)の初期化も行います。
3	sfr_r835a.h	R8C/35A マイコンの内蔵周辺機能を制御するためのレジスタ(Special Function Registers)を定義したファイルです。

17.4 プログラム「adc.c」

```

1 : /******
2 : /* 対象マイコン R8C/35A */
3 : /* ファイル内容 A/D変換(単発モード) */
4 : /* バージョン Ver. 1.20 */
5 : /* Date 2010.04.19 */
6 : /* Copyright ルネサスマイコンカーラーリ事務局 */
7 : /* 日立インターメディックス株式会社 */
8 : /******
9 : /*
10 : 入力 : AN7(P0_0)端子 0~5V(ミニマイコンカーの赤外線フォトインタラプタU8)
11 : 出力 : P1_3-P1_0(マイコンボードのLED)
12 :
13 : AN7(P0_0)端子から入力した電圧をA/D変換して、デジタル値をマイコンボードの
14 : LEDへ出力します。
15 : */
16 :
17 : /*=====*/
18 : /* インクルード */
19 : /*=====*/
20 : #include "sfr_r835a.h" /* R8C/35A SFRの定義ファイル */
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 : int get_ad7( void );
31 : void led_out( unsigned char led );
32 :
33 : /******
34 : /* メインプログラム */
35 : /******
36 : void main( void )
37 : {
38 :     int ad;
39 :
40 :     init(); /* 初期化 */
41 :
42 :     while( 1 ) {
43 :         ad = get_ad7();
44 :         ad = ad >> 6;
45 :         led_out( ad );
46 :     }
47 : }
48 :

```

17. A/D コンバータ(単発モード)(プロジェクト:ad)

```

49 : /******
50 : /* R8C/35A スペシャルファンクションレジスタ(SFR)の初期化 */
51 : /******
52 : void init( void )
53 : {
54 :     int i;
55 :
56 :     /* クロックをXINクロック(20MHz)に変更 */
57 :     prc0 = 1; /* プロテクト解除 */
58 :     cm13 = 1; /* P4_6,P4_7をXIN-XOUT端子にする*/
59 :     cm05 = 0; /* XINクロック発振 */
60 :     for(i=0; i<50; i++ ); /* 安定するまで少し待つ(約10ms) */
61 :     ocd2 = 0; /* システムクロックをXINにする */
62 :     prc0 = 0; /* プロテクトON */
63 :
64 :     /* ポートの入出力設定 */
65 :     prc2 = 1; /* PD0のプロテクト解除 */
66 :     pd0 = 0xe0; /* 7-5:LED 4:SW 3-0:アナログ電圧*/
67 :     p1 = 0x0f; /* 3-0:LEDは消灯 */
68 :     pd1 = 0xdf; /* 5:RXD0 4:TXD0 3-0:LED */
69 :     pd2 = 0xfe; /* 0:PushSW */
70 :     pd3 = 0xfb; /* 4: buzzer 2: IR */
71 :     pd4 = 0x83; /* 7:XOUT 6:XIN 5-3:DIP SW 2:VREF*/
72 :     pd5 = 0x40; /* 7:DIP SW */
73 :     pd6 = 0xff;
74 : }
75 :
76 : /******
77 : /* A/D値読み込み(AN7) */
78 : /* 引数 なし */
79 : /* 戻り値 A/D値 0~1023 */
80 : /******
81 : int get_ad7( void )
82 : {
83 :     int i;
84 :
85 :     /* A/Dコンバータの設定 */
86 :     admod = 0x03; /* 単発モードに設定 */
87 :     adinsel = 0x07; /* 入力端子AN7(P0_0)を選択 */
88 :     adcon1 = 0x30; /* A/D動作可能 */
89 :     asm( " nop "); /* φADの1サイクルウェイト入れる*/
90 :     adcon0 = 0x01; /* A/D変換スタート */
91 :
92 :     while( adcon0 & 0x01 ); /* A/D変換終了待ち */
93 :
94 :     i = ad7;
95 :
96 :     return i;
97 : }
98 :
99 : /******
100 : /* マイコン部のLED出力 */
101 : /* 引数 スイッチ値 0~15 */
102 : /******
103 : void led_out( unsigned char led )
104 : {
105 :     unsigned char data;
106 :
107 :     led = ~led;
108 :     led &= 0x0f;
109 :     data = p1 & 0xf0;
110 :     p1 = data | led;
111 : }
112 :
113 : /******
114 : /* end of file */
115 : /******

```

17.5 プログラムの解説

17.5.1 init 関数

P0_0 はアナログ電圧入力端子なので、ポートの入出力設定は入力にします。忘れやすいので、気をつけてください。


```

64 :      /* ポートの入出力設定 */
65 :      prc2 = 1;                          /* PD0のプロテクト解除      */
66 :      pd0 = 0xe0;                        /* 7-5:LED 4:SW 3-0:アナログ電圧*/
67 :      p1  = 0x0f;                          /* 3-0:LEDは消灯          */
68 :      pd1 = 0xdf;                          /* 5:RXD0 4:TXD0 3-0:LED  */
69 :      pd2 = 0xfe;                          /* 0:PushSW              */
70 :      pd3 = 0xfb;                          /* 4:Buzzer 2:IR         */
71 :      pd4 = 0x83;                          /* 7:XOUT 6:XIN 5-3:DIP SW 2:VREF*/
72 :      pd5 = 0x40;                          /* 7:DIP SW              */
73 :      pd6 = 0xff;

```

ポート0 にセンサ部を接続している場合は、P0_4 はマイクロスイッチ、P0_3～P0_1 はセンサが繋がっているので入力にします。実習基板 Ver.2 などを使ってこれらの端子が未接続の場合は、出力にしてください。

17.5.2 get_ad7 関数(A/D コンバータの設定、A/D 値取得)

今回は、AD7 から A/D 変換値を読み込むので、関数名を「get_ad7」にしました。アナログ入力端子を替える場合は、A/D 変換する端子名に合わせて関数名を付けてください。

```

76 :  /*****
77 :  /* A/D値読み込み (AN7)
78 :  /* 引数   なし
79 :  /* 戻り値 A/D値 0～1023
80 :  *****/
81 :  int get_ad7( void )
82 :  {
83 :      int i;
84 :
85 :      /* A/Dコンバータの設定 */
86 :      admod  = 0x03;                /* 単発モードに設定      */
87 :      adinsel = 0x07;                /* 入力端子AN7 (P0_0) を選択 */
88 :      adcon1  = 0x30;                /* A/D動作可能          */
89 :      asm( " nop " );                /* φADの1サイクルウェイト入れる*/
90 :      adcon0  = 0x01;                /* A/D変換スタート      */
91 :
92 :      while( adcon0 & 0x01 );        /* A/D変換終了待ち      */
93 :
94 :      i = ad7;
95 :
96 :      return i;
97 :  }

```

(1) A/D コンバータとは

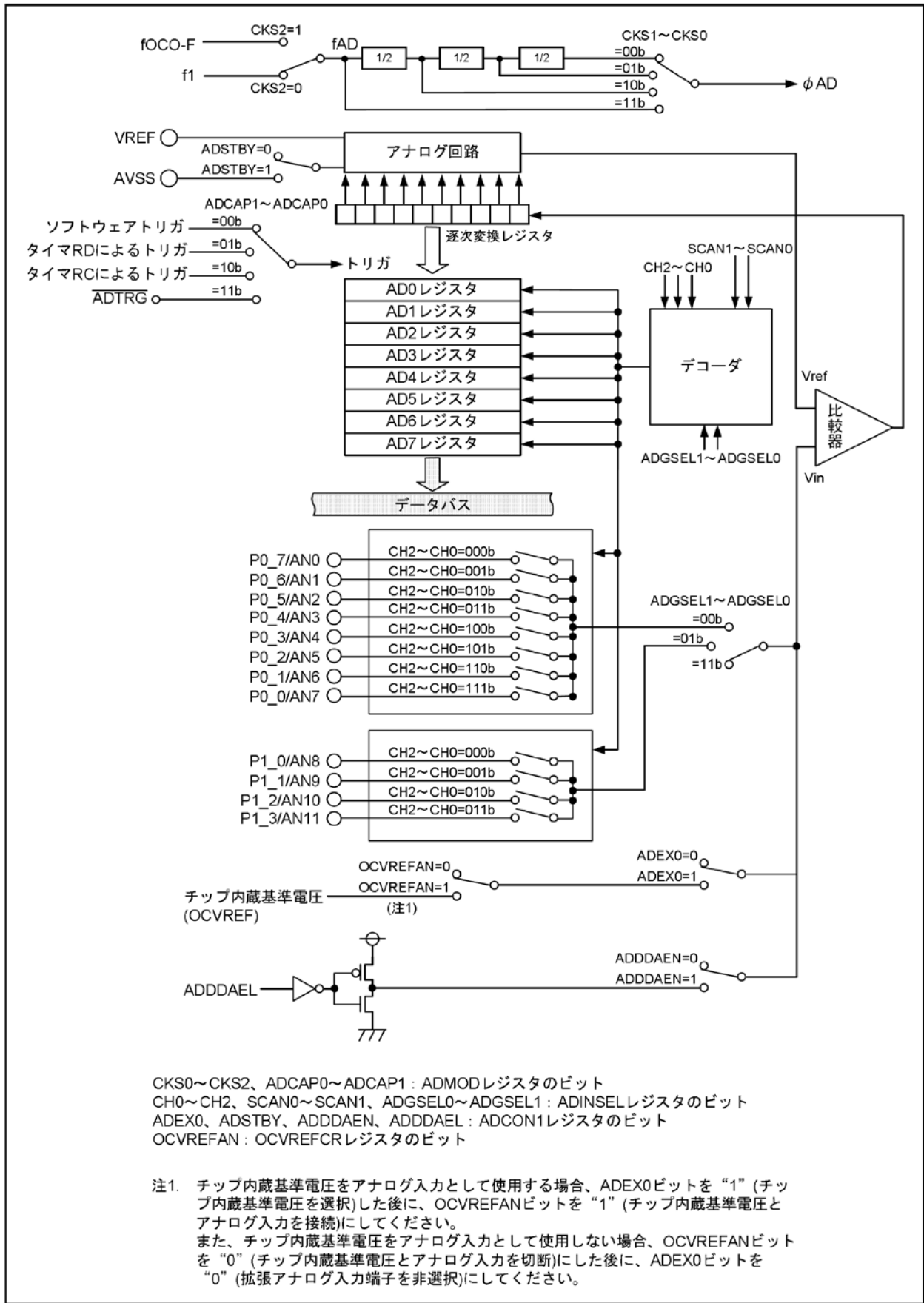
A/D コンバータは、アナログ/デジタルコンバータの略称です。マイコンは“0”か“1”か(“なし”か“あり”か)のデジタルで判断するので、外部からの入力電圧も 0V(“0”)か 5V(“1”)の 2 通りしか判断することができません。これ以外の電圧が入力されても、ある電圧を境にして“0”または“1”と判断します。

A/D コンバータを使うと、外部から入力された電圧が何 V か知ることができます。例えば、A/D コンバータを使わないと 4.0V の電圧が入力されてもマイコンは“1”としか判断できませんが、A/D コンバータを使うと 4.0V という電圧が入力されている、と判断することができます。

R8C/35A マイコンの A/D コンバータの特徴を下記に示します。

内容	詳細	
A/D 変換方式	容量結合増幅器で構成された、10ビットの逐次比較変換方式	
アナログ入力端子	P0_0～P0_7、P1_0～P1_3 の 12 本をアナログ入力端子として使用可能	
A/D 変換器の数	1 個 複数の A/D 変換を同時に行うことはできません。複数のアナログ入力端子の電圧を A/D 変換したい場合は、1 端子ずつ順番に行います。どの端子の電圧を A/D 変換するかは、プログラムで設定します。	
分解能	10bit 10bit とは 2 進数で 1 が 10 個ということです。(11 1111 1111) ₂ =(1023) ₁₀ となります。 0～5V(電源電圧)を 0～1023 の値に変換することができます。 電圧は次の式で求めることができます。 電圧=A/D 変換値/1023×5.000(電源電圧) [V] A/D 変換値が 0 なら 0.000V、1023 なら 5.000V、1 なら約 0.004886V が入力されていることとなります。A/D 変換値 1 あたり、約 0.004886V(1/1023)です。 ※「(数値) ₂ 」は 2 進数、「(数値) ₁₀ 」は 10 進数を表します。	
変換時間	最小 43 φ fAD fAD は A/D 変換するクロックです。クロックは内蔵クロック、外部クロックなど選択することができます。今回は外部クロックを選択します。φ は動作クロックです。ミニマイコンカー Ver.2 は、外部クロックとして 20MHz のクリスタルを使っています。よって、変換時間は、次のようになります。 $43 \times (1/\text{外部クロック}) = 43 \times (1/20 \times 10^6) = 43 \times 0.05 \times 10^{-6} = 2.15 \times 10^{-6} = \mathbf{2.15 \mu s}$ 実際は A/D 変換する端子を設定したり、A/D 変換値を変数に保存したり計算したりするので、すべてを含めると 1 端子あたり数十 μs 程度かかります(計算によって変わります)。	
動作モード	単発モード	AN0～AN11 から選択した 1 本の端子の入力電圧を、1 回 A/D 変換するモードです。
	繰り返しモード 0	AN0～AN11 から選択した 1 本の端子の入力電圧を、繰り返し A/D 変換するモードです。
	繰り返しモード 1	AN0～AN11 から選択した 1 本の端子の入力電圧を、繰り返し A/D 変換するモードです。A/D 変換値は、8 個のレジスタに順番に代入します。過去 8 個分の A/D 変換値を保存することができます。
	単掃引モード	AN0～AN11 から選択した 2 本、4 本、6 本、または 8 本の端子の入力電圧を、1 回ずつ A/D 変換するモードです。
	繰り返し掃引モード	AN0～AN11 から選択した 2 本、4 本、6 本、または 8 本の端子の入力電圧を、繰り返し A/D 変換するモードです。
アナログ入力端子	AN0(P0_7)、AN1(P0_6)、AN2(P0_5)、AN3(P0_4)、AN4(P0_3)、AN5(P0_2)、AN6(P0_1)、AN7(P0_0)、AN8(P1_0)、AN9(P1_1)、AN10(P1_2)、AN11(P1_3) の合計 12 本	

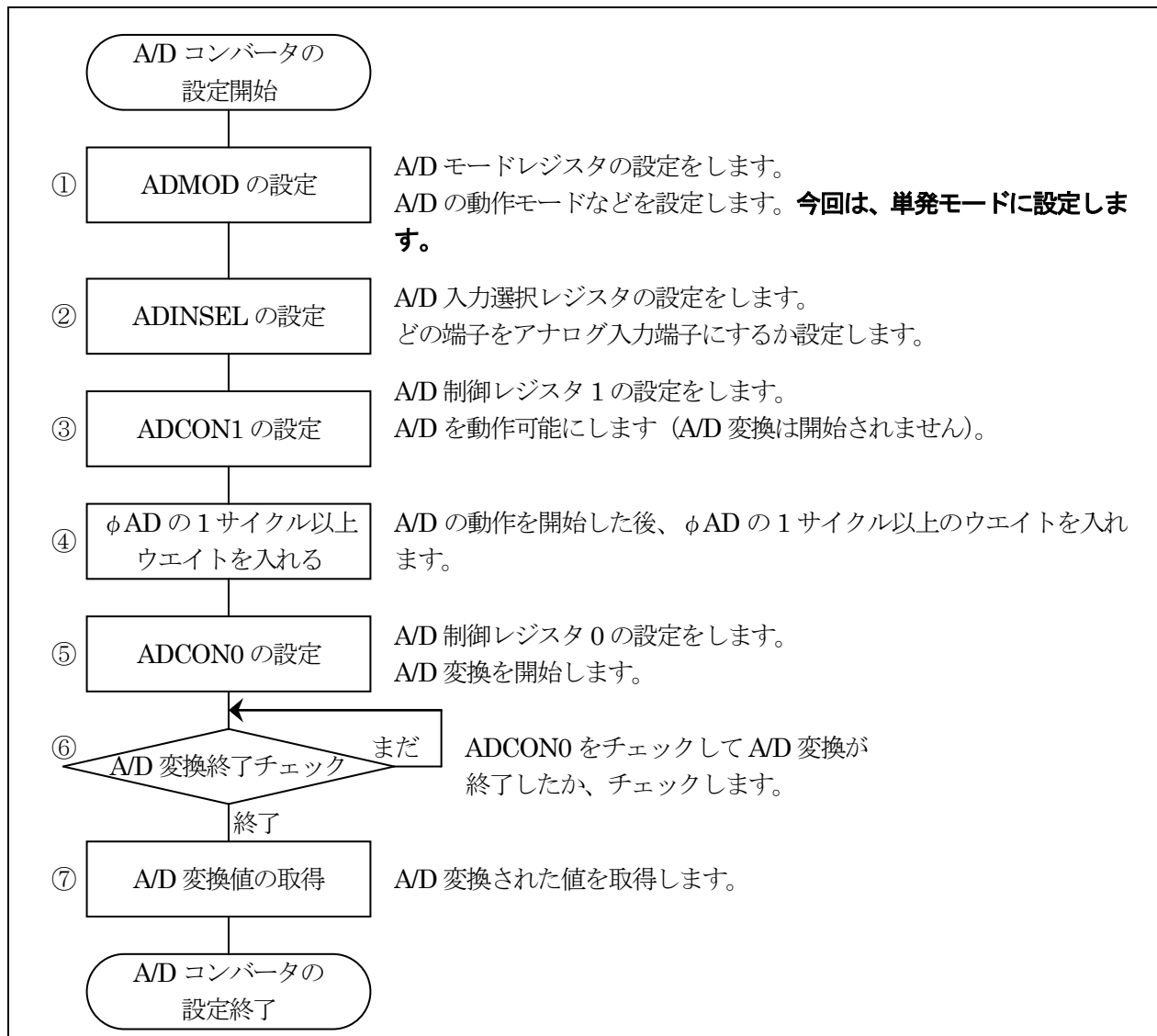
(2) A/D コンバータのブロック図



(3) A/D コンバータの設定

今回は、1本の端子からアナログ電圧を読み込み、1回だけA/D変換をする設定(単発モード)にします。フォトインタラプタが接続されているP0_0(AN7)の電圧を読み込みます。

レジスタの設定手順を下記に示します。



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

A/D の動作モードを設定します。今回は、単発モードに設定します。

設定 bit	設定内容	内容	今回の内容
bit7,6	A/D 変換トリガ選択ビット bit7: adcap1 bit6: adcap0	00:ソフトウェアトリガ(ADCON0 レジスタの ADST ビット)による A/D 変換開始 01:タイマ RD からの変換トリガによる A/D 変換開始 10:タイマ RC からの変換トリガによる A/D 変換開始 11:外部トリガ(ADTRG)による A/D 変換開始 A/D 変換を開始するきっかけをどれにするか設定します。ソフト的に開始するので、“00”を選択します。	00
bit5~3	A/D 動作モード選択 bit5: md2 bit4: md1 bit3: md0	000:単発モード 001:設定しないでください 010:繰り返しモード 0 011:繰り返しモード 1 100:単掃引モード 101:設定しないでください 110:繰り返し掃引モード 111:設定しないでください 今回は、単発モードを選択します。	000
bit2	クロック源選択ビット cks2	0:f1 (20MHz)を選択 1:fOCO-F (高速オンチップオシレータ)を選択 f1 を選択します。	0
bit1,0	分周選択ビット bit1: cks1 bit0: cks0	00:fAD の 8 分周 (8/20MHz=400ns) 01:fAD の 4 分周 (4/20MHz=200ns) 10:fAD の 2 分周 (2/20MHz=100ns) 11:fAD の 1 分周 (1/20MHz=50ns) fAD とは、bit2 で設定したクロック源のことです。このクロックを何分周で使用するか選択します。遅くする必要はないので、いちばん速い 1 分周で使用します。	11

A/D モードレジスタ(ADMOD)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	0	0	1	1
16 進数	0				3			

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

どのアナログ入力端子を A/D 変換するか、設定します。

A/D 入力グループ 選択ビット		bit5	bit4	bit3	アナログ入力端子 選択ビット			アナログ入力端子
bit7	bit6				bit2	bit1	bit0	
0	0	0	0	0	0	0	0	AN0(P0_7)
0	0	0	0	0	0	0	1	AN1(P0_6)
0	0	0	0	0	0	1	0	AN2(P0_5)
0	0	0	0	0	0	1	1	AN3(P0_4)
0	0	0	0	0	1	0	0	AN4(P0_3)
0	0	0	0	0	1	0	1	AN5(P0_2)
0	0	0	0	0	1	1	0	AN6(P0_1)
0	0	0	0	0	1	1	1	AN7(P0_0)
0	1	0	0	0	0	0	0	AN8(P1_0)
0	1	0	0	0	0	0	1	AN9(P1_1)
0	1	0	0	0	0	1	0	AN10(P1_2)
0	1	0	0	0	0	1	1	AN11(P1_3)

今回は、AN7(P0_0)を選択します。A/D 入力選択レジスタ (ADINSEL)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	0	1	1	1
16 進数	0				7			

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

A/D を動作可能にします。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7	A/D 断線検出アシスト方式選択ビット(注 4)	0:変換前ディスチャージ 1:変換前プリチャージ A/D 断線検出アシストしませんのでどちらでも構いませんが、今回は“0”にしておきます。	0
bit6	A/D 断線検出アシスト機能許可ビット(注 4)	0:禁止 1:許可 A/D 断線検出アシストは使いません。	0
bit5	A/D スタンバイビット(注 3) adstby	0:A/D 動作停止(スタンバイ) 1:A/D 動作可能 A/D 動作可能にして A/D 変換できるようにします。この bit を“0”から“1”にしたときは、φ A/D の 1 サイクル以上経過した後に A/D 変換を開始します。	1
bit4	8/10 ビットモード選択ビット bits	0:8 ビットモード 1:10 ビットモード A/D 変換を 10bit(0~1023)にするか、8bit(0~255)にするか選択します。今回は、10bit にします。	1
bit3~1		“000”を設定	000
bit0	拡張アナログ入力端子選択ビット(注 1) adex0	0:拡張アナログ入力端子を非選択 1:チップ内蔵基準電圧を選択(注 2) 拡張アナログ入力端子は使いません。	0

注 1. チップ内蔵基準電圧をアナログ入力として使用する場合、ADEX0 ビットを“1”(チップ内蔵基準電圧を選択)にした後に、OCVREFCR レジスタの OCVREFAN ビットを“1”(チップ内蔵基準電圧とアナログ入力を接続)にしてください。また、チップ内蔵基準電圧をアナログ入力として使用しない場合、OCVREFAN ビットを“0”(チップ内蔵基準電圧とアナログ入力を切断)にした後に、ADEX0 ビットを“0”(拡張アナログ入力端子を非選択)にしてください。

注 2. 単掃引モード、繰り返し掃引モードでは設定しないでください。

注 3. ADSTBY ビットを“0”(A/D 動作停止) から“1”(A/D 動作可能) にしたときは、φ AD の 1 サイクル以上経過した後に A/D 変換を開始してください。

注 4. A/D 断線検出アシスト機能を許可するためには、ADDDAEN ビットを“1”(許可)にした後、ADDDAEL ビットで変換開始状態を選択してください。断線時の変換結果は、外付け回路によって変化します。本機能はシステムに合わせた評価を十分に行った上で、使用してください。

A/D 制御レジスタ 1 (ADCON1)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	1	1	0	0	0	0
16 進数	3				0			

④ φAD の 1 サイクル以上ウエイトを入れる

③の bit5 の A/D スタンバイビットを"1"にした場合、φ A/D の 1 サイクル以上経過した後に A/D 変換を開始しなければいけません。

このウエイトを入れるため、アセンブリ言語の nop 命令を実行します。C 言語ソースファイル内では、アセンブリ言語は実行できないため、asm 命令というアセンブリ言語を実行できる命令を使って nop 命令を実行します。ちなみに、nop は「No Operation (何もしない)」命令で、この命令を実行するのに 1 サイクル分の時間がかかります。プログラムを下記に示します。

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

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

A/D 変換を開始します。

設定 bit	上:ビット名 下:シンボル	内容	今回の内容
bit7~1		"0000000"を設定	0000 000
bit0	A/D 変換開始フラグ adst	0:A/D 変換停止 1:A/D 変換開始 A/D 変換を開始させるので"1"を設定します。A/D 変換が終了すると自動で"0"になります。	1

A/D 制御レジスタ 0 (ADCON0)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	0	0	0	1
16 進数	0				1			

⑥A/D 変換の終了チェック

A/D 変換が終了すると、A/D 制御レジスタ 0 (ADCON0)の bit0 が"0"になります。プログラムでは、ADCON0 の bit0 が"0"かどうかチェック、"0"なら終了、"1"ならまだ変換中と判断できます。

```
while( adcon0 & 0x01 ); /* A/D 変換終了待ち */
```

A/D 変換中、ADCON0 の bit0 が"1"なので、「adcon0 & 0x01=0x01」となります。while 文はカッコの中が 0 以外なら繰り返しますので、この行を繰り返し続けます。

A/D 変換が完了すると、ADCON0 の bit0 が"0"になります。「adcon0 & 0x01=0x00」となります。while 文はカッコの中が 0 なので、次の行へ進みます。

⑦A/D 変換値の取得

A/D 変換された結果は、A/D レジスタ 0~7(AD0~AD7)に格納されます。AD0~AD7 のどのレジスタに格納されるかは、アナログ入力端子によって変わります。アナログ入力端子と A/D レジスタの関係を下記に示します。

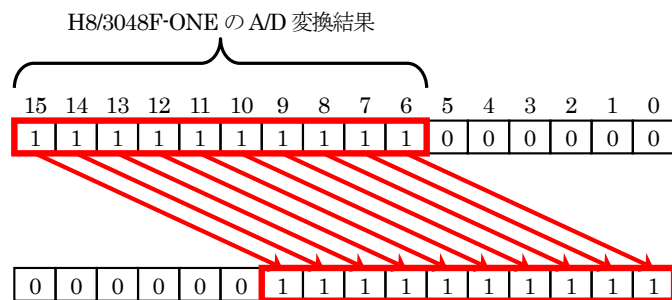
アナログ入力端子	読み込むレジスタ
AN0(P0_7)	AD0
AN1(P0_6)	AD1
AN2(P0_5)	AD2
AN3(P0_4)	AD3
AN4(P0_3)	AD4
AN5(P0_2)	AD5
AN6(P0_1)	AD6
AN7(P0_0)	AD7
AN8(P1_0)	AD0
AN9(P1_1)	AD1
AN10(P1_2)	AD2
AN11(P1_3)	AD3

今回は、AN7(P0_0 端子)を使用しているので、表より AD7 レジスタを読み込みます。

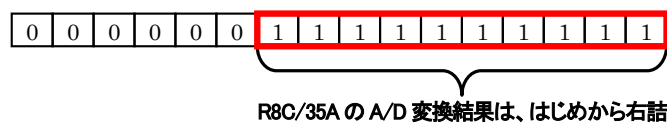
```
i = ad7;
```

※H8/3048F-ONE と R8C/35A の格納方法の違い

H8/3048F-ONE の A/D 変換結果は、左詰で格納されるためプログラムで 6bit 右シフトしなければいけません。



R8C/35A の A/D 変換結果は、右詰で格納されるためプログラムでシフト処理は必要ありません。



17.5.3 main 関数

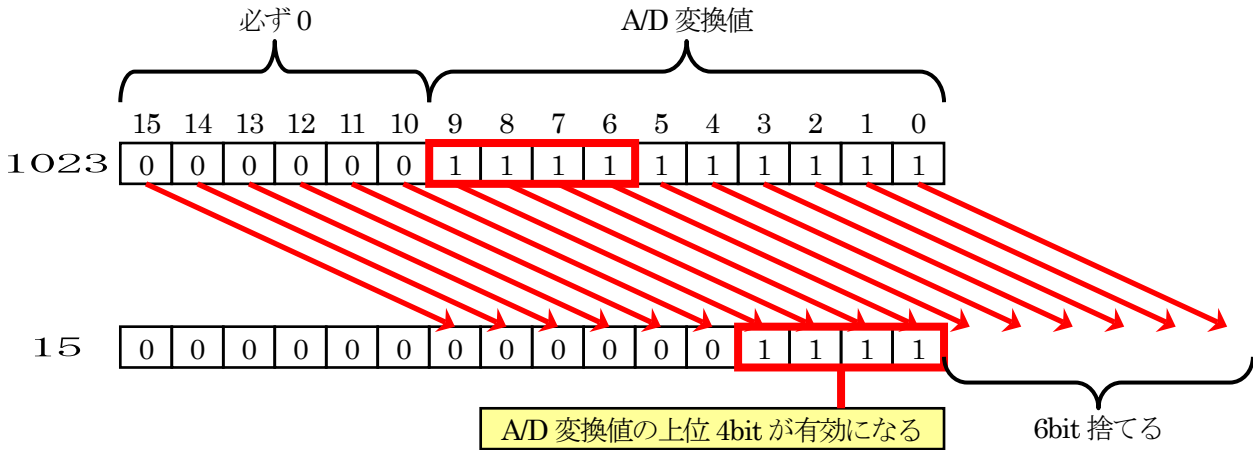
A/D 変換値を取得、その値をマイコンボード上の LED へ出力します。

```

36 : void main( void )
37 : {
38 :     int ad;
39 :
40 :     init();           /* 初期化           */
41 :
42 :     while( 1 ) {
43 :         ad = get_ad7();
44 :         ad = ad >> 6;
45 :         led_out( ad );
46 :     }
47 : }
    
```

43 行	get_ad7 関数で A/D 変換値を取得し、ad 変数に格納します。
44 行	A/D 変換値は、0~1023(2 進数で 11 1111 1111)の値です。LED は 4 個しかありません。そのため今回は、2 進数で 10 桁の A/D 値を 4 桁に変換します。プログラムは、右シフトを 6 ビット分行い、下位の 6 桁を捨てます。その結果、A/D 値は 0~15 の値になり、ad 変数に代入します。
45 行	0~15 に変換した A/D 値をマイコンボード上の LED に出力します。

変数 ad の値が 1023 のとき、44 行のビットシフトの様子を下記に示します。



17.6 演習

- (1) ポート 6 に実習基板 Ver.2 の LED 部を接続して、A/D 変換値の上位 8bit をその LED へ出力しなさい。
- (2) (1)の状態、アナログ入力端子を P0_1 端子に変更して、LED へ出力しなさい。

18. A/D コンバータ(繰り返しモード 0)(プロジェクト:ad_kurikaeshi)

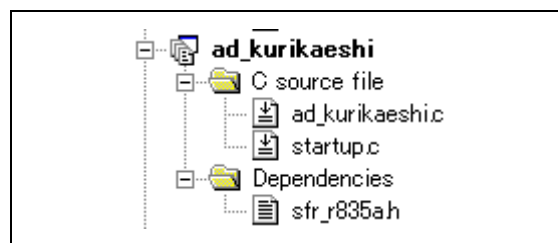
18.1 概要

本章では、0～5V の電圧をマイコンの A/D コンバータで読み込む方法を説明します。A/D 変換した結果は、マイコンボードの LED に出力します。今回の A/D 変換は、繰り返しモード 0 を使います。

18.2 接続

「17. A/D コンバータ(単発モード)(プロジェクト:ad)」と同じです。

18.3 プロジェクトの構成



	ファイル名	内容
1	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。このファイルは共通で、どのプロジェクトもこのファイルから実行されます。
2	ad_kurikaeshi.c	実際に制御するプログラムが書かれています。R8C/35A の内蔵周辺機能(SFR)の初期化も行います。
3	sfr_r835a.h	R8C/35A マイコンの内蔵周辺機能を制御するためのレジスタ(Special Function Registers)を定義したファイルです。

18.4 プログラム「ad_kurikaeshi.c」

```

1 : /*****
2 : /* 対象マイコン R8C/35A */
3 : /* ファイル内容 A/D変換(繰り返しモード0) */
4 : /* バージョン Ver. 1.20 */
5 : /* Date 2010.04.19 */
6 : /* Copyright ルネサスマイコンカーラリー事務局 */
7 : /* 日立インターメディックス株式会社 */
8 : *****/
9 :
10 : 入力: AN7(P0_0)端子 0~5V(ミニマイコンカーの赤外線フォトインタラプタU8)
11 : 出力: P1_3-P1_0(マイコンボードのLED)
12 :
13 : AN7(P0_0)端子から入力した電圧をA/D変換して、デジタル値をマイコンボードの
14 : LEDへ出力します。
15 : */
16 :
17 : /*=====*/
18 : /* インクルード */
19 : /*=====*/
20 : #include "sfr_r835a.h" /* R8C/35A SFRの定義ファイル */
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 : int get_ad7( void );
31 : void led_out( unsigned char led );
32 :
33 : *****/
34 : /* メインプログラム */
35 : *****/
36 : void main( void )
37 : {
38 :     int ad;
39 :
40 :     init(); /* 初期化 */
41 :
42 :     while( 1 ) {
43 :         ad = get_ad7();
44 :         ad = ad >> 6;
45 :         led_out( ad );
46 :     }
47 : }
48 :
49 : *****/
50 : /* R8C/35A スペシャルファンクションレジスタ(SFR)の初期化 */
51 : *****/
52 : void init( void )
53 : {
54 :     int i;
55 :
56 :     /* クロックをXINクロック(20MHz)に変更 */
57 :     prc0 = 1; /* プロテクト解除 */
58 :     cm13 = 1; /* P4_6, P4_7をXIN-XOUT端子にする*/
59 :     cm05 = 0; /* XINクロック発振 */
60 :     for(i=0; i<50; i++ ); /* 安定するまで少し待つ(約10ms) */
61 :     ocd2 = 0; /* システムクロックをXINにする */
62 :     prc0 = 0; /* プロテクトON */
63 :
64 :     /* ポートの入出力設定 */
65 :     prc2 = 1; /* PD0のプロテクト解除 */
66 :     pd0 = 0xe0; /* 7-5:LED 4:SW 3-0:アナログ電圧*/
67 :     p1 = 0x0f; /* 3-0:LEDは消灯 */
68 :     pd1 = 0xdf; /* 5:RXD0 4:TXD0 3-0:LED */
69 :     pd2 = 0xfe; /* 0:PushSW */
70 :     pd3 = 0xfb; /* 4:Buzzer 2:IR */
71 :     pd4 = 0x83; /* 7:XOUT 6:XIN 5-3:DIP SW 2:VREF*/
72 :     pd5 = 0x40; /* 7:DIP SW */
73 :     pd6 = 0xff;
74 :
75 :     /* A/Dコンバータの設定 */
76 :     admod = 0x13; /* 繰り返しモード0に設定 */
77 :     adinsel = 0x07; /* 入力端子AN7(P0_0)を選択 */
78 :     adcon1 = 0x30; /* A/D動作可能 */
79 :     asm( " nop " ); /* φADの1サイクルウェイト入れる*/
80 :     adcon0 = 0x01; /* A/D変換スタート */
81 : }
82 :

```

18. A/D コンバータ(繰り返しモード 0)(プロジェクト:ad_kurikaeshi)

```

83 : /******-/
84 : /* A/D値読み込み(AN7) */
85 : /* 引数 なし */
86 : /* 戻り値 A/D値 0~1023 */
87 : /******-/
88 : int get_ad7( void )
89 : {
90 :     int i;
91 :
92 :     /* 繰り返しモード0は、自動的に繰り返すので、結果を読み込むだけ */
93 :     i = ad7;
94 :
95 :     return i;
96 : }
97 :
98 : /******-/
99 : /* マイコン部のLED出力 */
100 : /* 引数 スイッチ値 0~15 */
101 : /******-/
102 : void led_out( unsigned char led )
103 : {
104 :     unsigned char data;
105 :
106 :     led = ~led;
107 :     led &= 0x0f;
108 :     data = p1 & 0xf0;
109 :     p1 = data | led;
110 : }
111 :
112 : /******-/
113 : /* end of file */
114 : /******-/

```

18.5 プログラムの解説

18.5.1 init 関数(I/O ポートの入出力設定)

P0_0 はアナログ電圧入力端子なので、ポートの入出力設定は入力にします。忘れやすいので、気をつけてください。

```

64 :     /* ポートの入出力設定 */
65 :     prc2 = 1;                /* PD0のプロテクト解除 */
66 :     pd0 = 0xe0;            /* 7-5:LED 4:SW 3-0:アナログ電圧*/
67 :     p1 = 0x0f;              /* 3-0:LEDは消灯 */
68 :     pd1 = 0xdf;            /* 5:RXD0 4:TXD0 3-0:LED */
69 :     pd2 = 0xfe;            /* 0:PushSW */
70 :     pd3 = 0xfb;            /* 4:Buzzer 2:IR */
71 :     pd4 = 0x83;            /* 7:XOUT 6:XIN 5-3:DIP SW 2:VREF*/
72 :     pd5 = 0x40;            /* 7:DIP SW */
73 :     pd6 = 0xff;

```

ポート0 にセンサ部を接続している場合は、P0_4 はマイクロスイッチ、P0_3~P0_1 はセンサが繋がっているので入力にします。実習基板 Ver.2 などを使ってこれらの端子が未接続の場合は、出力にしてください。

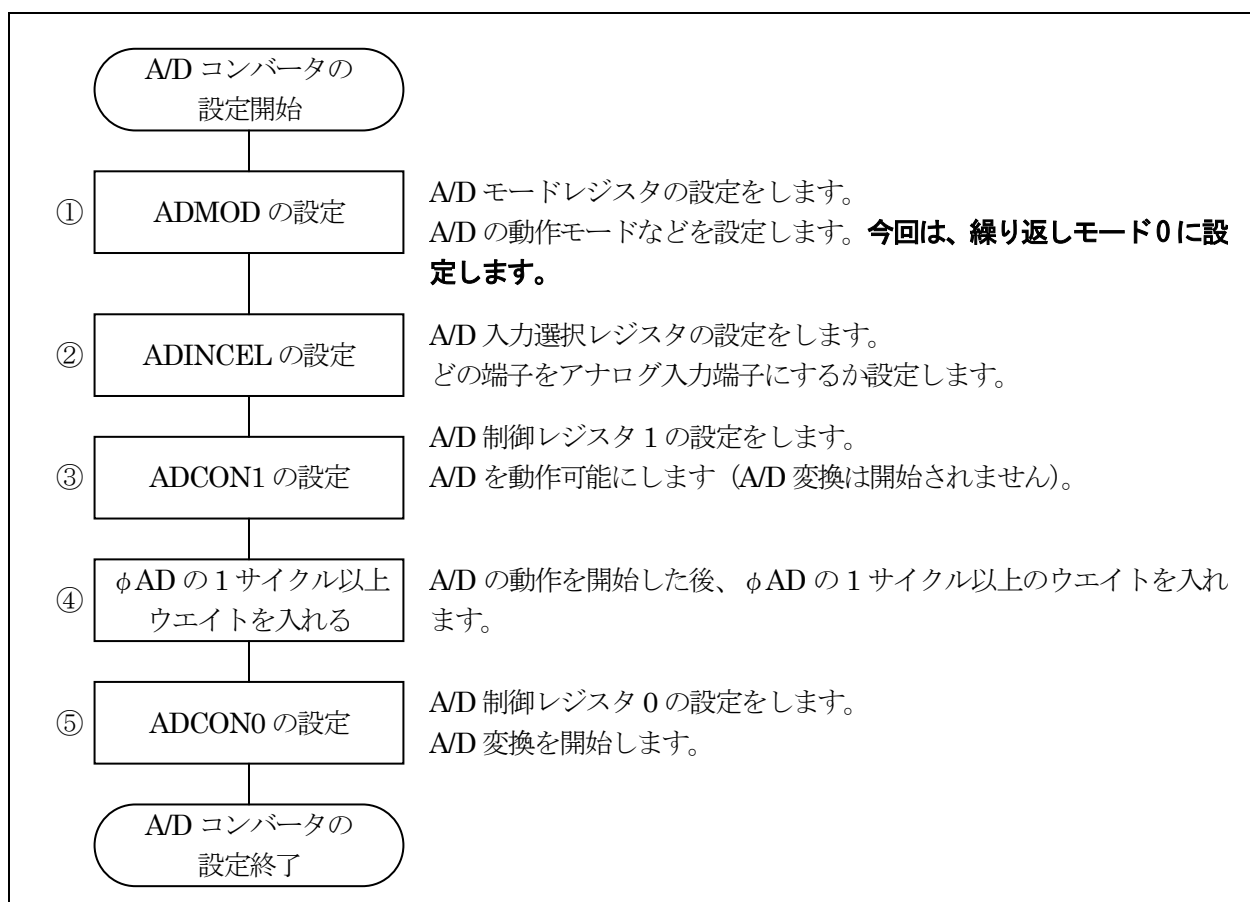
18.5.2 init 関数(A/D コンバータの設定)

A/D コンバータを設定するプログラムは、次のようになります。

75 :	/* A/Dコンバータの設定 */		
76 :	admod = 0x13;	/* 繰り返しモード0に設定	*/
77 :	adinsel = 0x07;	/* 入力端子AN7 (p0_0)を選択	*/
78 :	adcon1 = 0x30;	/* A/D動作可能	*/
79 :	asm(" nop ");	/* φADの1サイクルウェイト入れる*/	
80 :	adcon0 = 0x01;	/* A/D変換スタート	*/

今回は、1本の端子からアナログ電圧を読み込み、繰り返しA/D変換する設定(繰り返しモード0)にします。フォトインタラプタが接続されているP0_0(AN7)の電圧を読み込みます。

レジスタの設定手順を下記に示します。



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

A/D の動作モードを設定します。今回は繰り返しモード 0 に設定します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7,6	A/D 変換トリガ選択ビット bit7: adcap1 bit6: adcap0	00:ソフトウェアトリガ(ADCON0 レジスタの ADST ビット)による A/D 変換開始 01:タイマ RD からの変換トリガによる A/D 変換開始 10:タイマ RC からの変換トリガによる A/D 変換開始 11:外部トリガ(ADTRG)による A/D 変換開始 A/D 変換を開始するきっかけをどれにするか設定します。ソフト的に開始するので、“00”を選択します。	00
bit5~3	A/D 動作モード選択 bit5: md2 bit4: md1 bit3: md0	000:単発モード 001:設定しないでください 010:繰り返しモード 0 011:繰り返しモード 1 100:単掃引モード 101:設定しないでください 110:繰り返し掃引モード 111:設定しないでください 今回は、繰り返しモード 0 を選択します。	010
bit2	クロック源選択ビット cks2	0:f1 (20MHz)を選択 1:fOCO-F(高速オンチップオシレータ)を選択 f1 を選択します。	0
bit1,0	分周選択ビット bit1: cks1 bit0: cks0	00:fAD の 8 分周 (8/20MHz=400ns) 01:fAD の 4 分周 (4/20MHz=200ns) 10:fAD の 2 分周 (2/20MHz=100ns) 11:fAD の 1 分周 (1/20MHz=50ns) fAD とは、bit2 で設定したクロック源のことです。このクロックを何分周で使用するか選択します。遅くする必要はないので、いちばん速い 1 分周で使用します。	11

A/D モードレジスタ(ADMOD)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	1	0	0	1	1
16 進数	1				3			

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

どのアナログ入力端子を A/D 変換するか、設定します。

A/D 入力グループ 選択ビット		bit5	bit4	bit3	アナログ入力端子 選択ビット			アナログ入力端子
bit7	bit6				bit2	bit1	bit0	
0	0	0	0	0	0	0	0	AN0(P0_7)
0	0	0	0	0	0	0	1	AN1(P0_6)
0	0	0	0	0	0	1	0	AN2(P0_5)
0	0	0	0	0	0	1	1	AN3(P0_4)
0	0	0	0	0	1	0	0	AN4(P0_3)
0	0	0	0	0	1	0	1	AN5(P0_2)
0	0	0	0	0	1	1	0	AN6(P0_1)
0	0	0	0	0	1	1	1	AN7(P0_0)
0	1	0	0	0	0	0	0	AN8(P1_0)
0	1	0	0	0	0	0	1	AN9(P1_1)
0	1	0	0	0	0	1	0	AN10(P1_2)
0	1	0	0	0	0	1	1	AN11(P1_3)

今回は、AN7(P0_0)を選択します。A/D 入力選択レジスタ (ADINSEL)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	0	1	1	1
16 進数	0				7			

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

A/D を動作可能にします。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7	A/D 断線検出アシスト方式選択ビット(注 4)	0:変換前デイスチャージ 1:変換前プリチャージ A/D 断線検出アシストしませんのでどちらでも構いませんが、今回は"0"にしておきます。	0
bit6	A/D 断線検出アシスト機能許可ビット(注 4)	0:禁止 1:許可 A/D 断線検出アシストは使いません。	0
bit5	A/D スタンバイビット(注 3) adstby	0:A/D 動作停止(スタンバイ) 1:A/D 動作可能 A/D 動作可能にして A/D 変換できるようにします。この bit を"0"から"1"にしたときは、 ϕ A/D の 1 サイクル以上経過した後に A/D 変換を開始します。	1
bit4	8/10 ビットモード選択ビット bits	0:8 ビットモード 1:10 ビットモード A/D 変換を 10bit(0~1023)にするか、8bit(0~255)にするか選択します。今回は、10bit にします。	1
bit3~1		"000"を設定	000
bit0	拡張アナログ入力端子選択ビット(注 1) adex0	0:拡張アナログ入力端子を非選択 1:チップ内蔵基準電圧を選択(注 2) 拡張アナログ入力端子は使いません。	0

注 1. チップ内蔵基準電圧をアナログ入力として使用する場合、ADEX0 ビットを"1"(チップ内蔵基準電圧を選択)にした後に、OCVREFCR レジスタの OCVREFAN ビットを"1"(チップ内蔵基準電圧とアナログ入力を接続)にしてください。また、チップ内蔵基準電圧をアナログ入力として使用しない場合、OCVREFAN ビットを"0"(チップ内蔵基準電圧とアナログ入力を切断)にした後に、ADEX0 ビットを"0"(拡張アナログ入力端子を非選択)にしてください。

注 2. 単掃引モード、繰り返し掃引モードでは設定しないでください。

注 3. ADSTBY ビットを"0"(A/D 動作停止) から"1"(A/D 動作可能) にしたときは、 ϕ AD の 1 サイクル以上経過した後に A/D 変換を開始してください。

注 4. A/D 断線検出アシスト機能を許可にするためには、ADDDAEN ビットを"1"(許可)にした後、ADDDAEL ビットで変換開始状態を選択してください。断線時の変換結果は、外付け回路によって変化します。本機能はシステムに合わせた評価を十分に行った上で、使用してください。

A/D 制御レジスタ 1 (ADCON1)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	1	1	0	0	0	0
16 進数	3				0			

④ φAD の 1 サイクル以上ウェイトを入れる

③の bit5 の A/D スタンバイビットを"1"にした場合、φ A/D の 1 サイクル以上経過した後に A/D 変換を開始しなければいけません。

そのウェイトを入れるため、アセンブリ言語の nop 命令を実行します。C 言語ソースファイル内では、アセンブリ言語は実行できないため、asm 命令というアセンブリ言語を実行できる命令を使って nop 命令を実行します。ちなみに、nop は「No Operation (何もしない)」命令で、この命令を実行するのに 1 サイクル分の時間がかかります。プログラムを下記に示します。

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

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

A/D 変換を開始します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7~1		"0000000"を設定	000 0000
bit0	A/D 変換開始フラグ adst	0:A/D 変換停止 1:A/D 変換開始 A/D 変換を開始させるので"1"を設定します。	1

A/D 制御レジスタ 0(ADCON0)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	0	0	0	1
16 進数	0				1			

18.5.3 get_ad7 関数

get_ad7 関数は、A/D 変換した結果を取得する関数です。

```

83 : /*****/
84 : /* A/D値読み込み (AN7) */
85 : /* 引数 なし */
86 : /* 戻り値 A/D値 0~1023 */
87 : /*****/
88 : int get_ad7( void )
89 : {
90 :     int i;
91 :
92 :     /* 繰り返しモード0は、自動的に繰り返すので、結果を読み込むだけ */
93 :     i = ad7;
94 :
95 :     return i;
96 : }
    
```

93 行	ad 変換した結果が格納されている ad7 レジスタの値を、変数 i に代入します。
------	--

A/D 変換された結果は、A/D レジスタ 0~7(AD0~AD7)に格納されます。AD0~AD7 のどのレジスタに格納されるかは、アナログ入力端子によって変わります。アナログ入力端子と A/D レジスタの関係を下記に示します。

アナログ入力端子	読み込むレジスタ
AN0(P0_7)	AD0
AN1(P0_6)	AD1
AN2(P0_5)	AD2
AN3(P0_4)	AD3
AN4(P0_3)	AD4
AN5(P0_2)	AD5
AN6(P0_1)	AD6
AN7(P0_0)	AD7
AN8(P1_0)	AD0
AN9(P1_1)	AD1
AN10(P1_2)	AD2
AN11(P1_3)	AD3

今回は、AN7(P0_0 端子)を使用しているので、表より AD7 レジスタを読み込みます。

19. A/D コンバータ(繰り返し掃引モード)(プロジェクト:ad_kurikaeshi_souin)

19.1 概要

本章では、2本の0～5Vの電圧信号をマイコンのA/Dコンバータで読み込む方法を説明します。A/D変換した結果は、マイコンボードのLEDに出力します。今回のA/D変換は、繰り返し掃引モードを使います。今回は2本分ですが、プログラムを替えることにより8本の電圧信号まで読み込むことができます。

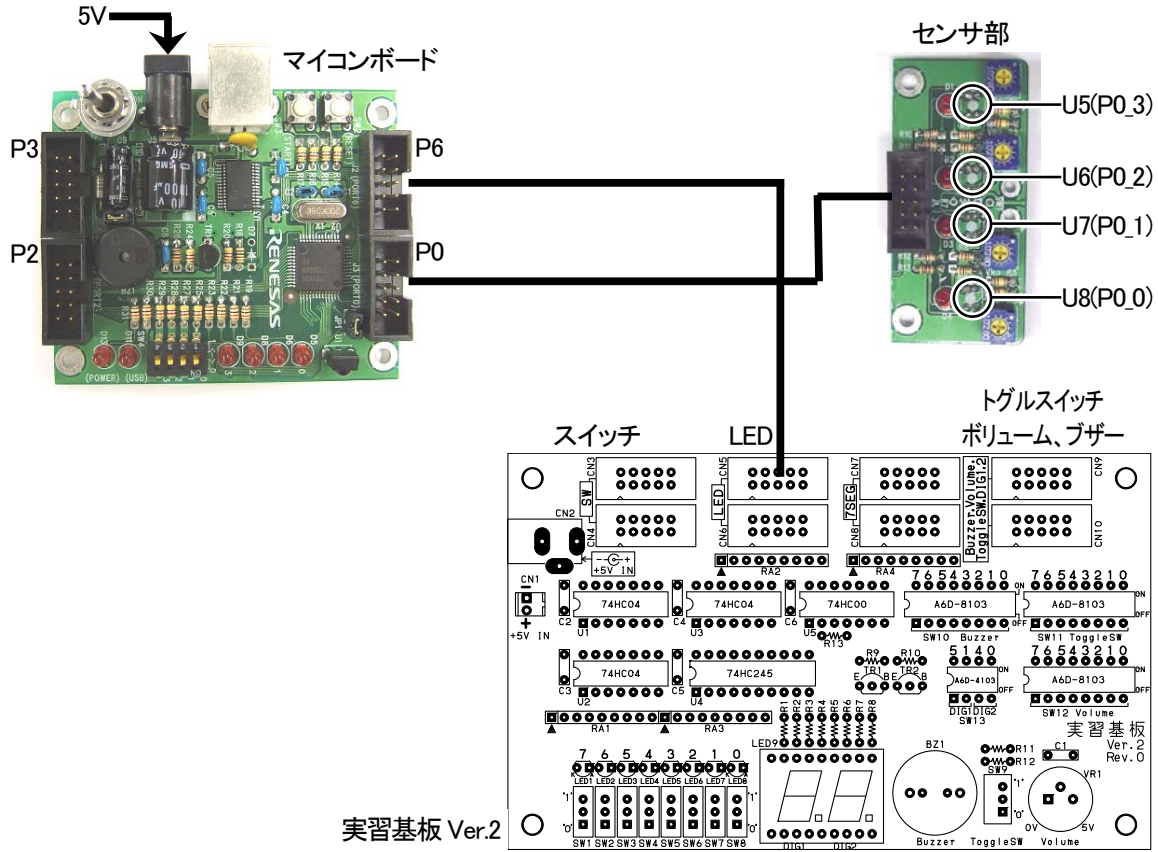
19.2 接続

■使用ポート

マイコンのポート	接続内容
P0_0、P0_1 (J3)	センサ部を接続します。U8、U7のA/D値を読み込みます。
P1_3、P1_2、 P1_1、P1_0	マイコンボード上のLEDです。
P6 (J2)	実習基板 Ver.2のLED部など、出力機器を接続します。

■接続

マイコンボードのポート0とセンサ部をフラットケーブルで接続します。センサは4個ありますが、今回 LED に出力するのは、U8(P0_0)と U7(P0_1)のセンサです。また、マイコンボードのポート 6 と実習基板 Ver.2 の LED 部を接続します。

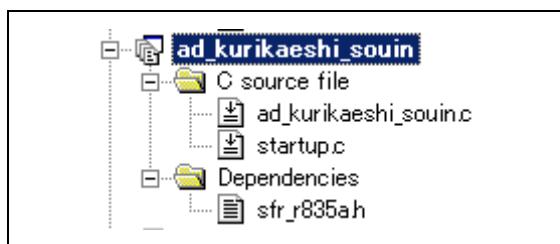


■操作方法

センサ部の U8 の下部を白色や灰色や黒色に近づけます。フォトインタラプタ U8 から出力された 0~5V の電圧をマイコンの P0_0 から読み込み、A/D 変換した値をマイコンボードの LED に出力します。

センサ部の U7 の下部を白色や灰色や黒色に近づけます。フォトインタラプタ U7 から出力された 0~5V の電圧をマイコンの P0_1 から読み込み、A/D 変換した値を実習基板 Ver.2 の LED 部に出力します。

19.3 プロジェクトの構成



	ファイル名	内容
1	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。このファイルは共通で、どのプロジェクトもこのファイルから実行されます。
2	ad_kurikaeshi_souin.c	実際に制御するプログラムが書かれています。R8C/35A の内蔵周辺機能(SFR)の初期化も行います。
3	sfr_r835a.h	R8C/35A マイコンの内蔵周辺機能を制御するためのレジスタ (Special Function Registers)を定義したファイルです。

19.4 プログラム「ad_kurikaeshi_souin.c」

```

1 : /*****
2 : /* 対象マイコン R8C/35A */
3 : /* ファイル内容 A/D変換(繰り返し掃引モード) */
4 : /* バージョン Ver. 1.20 */
5 : /* Date 2010.04.19 */
6 : /* Copyright ルネサスマイコンカーラー事務局 */
7 : /* 日立インターメディックス株式会社 */
8 : /*****
9 : /*
10 : 入力 : AN0(P0_7)~AN7(P0_0) 端子
11 : 0~5V(ミニマイコンカーの赤外線フォトインタラプタ(U5,U6,U7,U8)
12 : 出力 : P1_3-P1_0(マイコンボードのLED)
13 : P6_7-P6_0(実習基板のLED部など)
14 :
15 : AN0(P0_7)~AN7(P0_0) 端子の8端子から入力した電圧をA/D変換して、
16 : デジタル値をマイコンボードのLEDと実習ボードのLED部へ出力します。
17 : 8端子分のA/D変換しますが、入出力設定を出力しているAD端子の値は不定です。
18 : */
19 :
20 : /*=====*/
21 : /* インクルード */
22 : /*=====*/
23 : #include "sfr_r835a.h" /* R8C/35A SFRの定義ファイル */
24 :
25 : /*=====*/
26 : /* シンボル定義 */
27 : /*=====*/
28 :
29 : /*=====*/
30 : /* プロトタイプ宣言 */
31 : /*=====*/
32 : void init( void );
33 : void led_out( unsigned char led );
34 :
    
```

```

35 : /*****/
36 : /* メインプログラム */
37 : /*****/
38 : void main( void )
39 : {
40 :     int ad7data, ad6data;
41 :
42 :     init();                /* 初期化 */
43 :
44 :     while( 1 ) {
45 :         // AD7を出力
46 :         ad7data = ad7;    /* AD7取得 */
47 :         ad7data = ad7data >> 6;
48 :         led_out( ad7data ); /* マイコンボードのLEDへ出力 */
49 :
50 :         // AD6を出力
51 :         ad6data = ad6;    /* AD6取得 */
52 :         ad6data = ad6data >> 6;
53 :         p6 = ad6data;     /* P6のbit3~0のLEDへ出力 */
54 :     }
55 : }
56 :
57 : /*****/
58 : /* R8C/35A スペシャルファンクションレジスタ(SFR)の初期化 */
59 : /*****/
60 : void init( void )
61 : {
62 :     int i;
63 :
64 :     /* クロックをXINクロック(20MHz)に変更 */
65 :     prc0 = 1;             /* プロテクト解除 */
66 :     cm13 = 1;            /* P4_6, P4_7をXIN-XOUT端子にする */
67 :     cm05 = 0;            /* XINクロック発振 */
68 :     for(i=0; i<50; i++ ); /* 安定するまで少し待つ(約10ms) */
69 :     ocd2 = 0;            /* システムクロックをXINにする */
70 :     prc0 = 0;            /* プロテクトON */
71 :
72 :     /* ポートの入出力設定 */
73 :     prc2 = 1;             /* PD0のプロテクト解除 */
74 :     pd0 = 0xe0;          /* 7-5:LED 4:SW 3-0:アナログ電圧 */
75 :     p1 = 0x0f;           /* 3-0:LEDは消灯 */
76 :     pd1 = 0xdf;          /* 5:RXD0 4:TXD0 3-0:LED */
77 :     pd2 = 0xfe;          /* 0:PushSW */
78 :     pd3 = 0xfb;          /* 4: buzzer 2: IR */
79 :     pd4 = 0x83;          /* 7:XOUT 6:XIN 5-3:DIP SW 2:VREF */
80 :     pd5 = 0x40;          /* 7:DIP SW */
81 :     pd6 = 0xff;          /* LEDなど出力 */
82 :
83 :     /* A/Dコンバータの設定 */
84 :     admod = 0x33;         /* 繰り返し掃引モードに設定 */
85 :     adinsel = 0x30;       /* 入力端子P0の8端子を選択 */
86 :     adcon1 = 0x30;        /* A/D動作可能 */
87 :     asm( " nop ");        /* φADの1サイクルウェイト入れる */
88 :     adcon0 = 0x01;        /* A/D変換スタート */
89 : }
90 :
91 : /*****/
92 : /* マイコン部のLED出力 */
93 : /* 引数 スイッチ値 0~15 */
94 : /*****/
95 : void led_out( unsigned char led )
96 : {
97 :     unsigned char data;
98 :
99 :     led = ~led;
100 :     led &= 0x0f;
101 :     data = p1 & 0xf0;
102 :     p1 = data | led;
103 : }
104 :
105 : /*****/
106 : /* end of file */
107 : /*****/

```


19.5 プログラムの解説

19.5.1 init 関数(I/O ポートの入出力設定)

ポート 0 にはセンサ部が接続されています。bit7~5 は LED で出力、bit4 はマイクロスイッチで入力、bit3~0 はフォトインタラプタで入力に設定します。

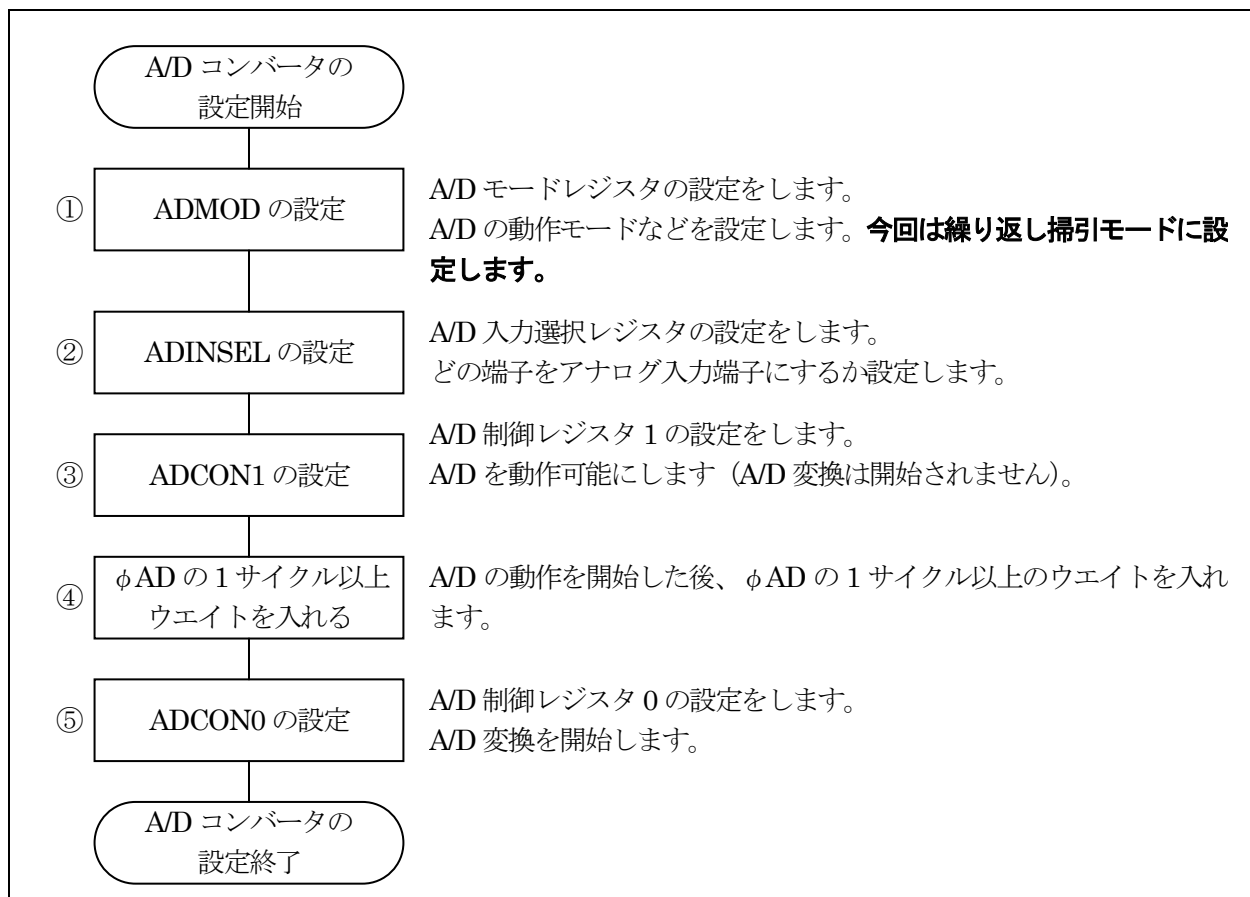
72 :	/* ポートの入出力設定 */	
73 :	prc2 = 1;	/* PD0のプロテクト解除 */
74 :	pd0 = 0xe0;	/* 7-5:LED 4:SW 3-0:アナログ電圧*/
75 :	p1 = 0x0f;	/* 3-0:LEDは消灯 */
76 :	pd1 = 0xdf;	/* 5:RXDO 4:TXDO 3-0:LED */
77 :	pd2 = 0xfe;	/* 0:PushSW */
78 :	pd3 = 0xfb;	/* 4:Buzzer 2:IR */
79 :	pd4 = 0x83;	/* 7:XOUT 6:XIN 5-3:DIP SW 2:VREF*/
80 :	pd5 = 0x40;	/* 7:DIP SW */
81 :	pd6 = 0xff;	/* LEDなど出力 */

19.5.2 init 関数(A/D コンバータの設定)

A/D コンバータを設定するプログラムは、次のようになります。

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

今回は、ポート0の8本の端子からアナログ電圧を読み込み、繰り返しA/D変換する設定(繰り返し掃引モード)にします。ポート0からAD値を読み込めるのは、フォトインタラプタが接続されているP0_0(AN7)~P0_3(AN4)の4端子です。残り4端子は読み込んでも値は不定です。
レジスタの設定手順を下記に示します。



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

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7,6	A/D 変換トリガ選択ビット bit7: adcap1 bit6: adcap0	00:ソフトウェアトリガ(ADCON0 レジスタの ADST ビット)による A/D 変換開始 01:タイマ RD からの変換トリガによる A/D 変換開始 10:タイマ RC からの変換トリガによる A/D 変換開始 11:外部トリガ(ADTRG)による A/D 変換開始 A/D 変換を開始するきっかけをどれにするか設定します。ソフト的に開始するので、“00”を選択します。	00
bit5~3	A/D 動作モード選択 bit5: md2 bit4: md1 bit3: md0	000:単発モード 001:設定しないでください 010:繰り返しモード 0 011:繰り返しモード 1 100:単掃引モード 101:設定しないでください 110:繰り返し掃引モード 111:設定しないでください 今回は、繰り返し掃引モードを選択します。	110
bit2	クロック源選択ビット cks2	0:f1 (20MHz)を選択 1:fOCO-F(高速オンチップオシレータ)を選択 f1 を選択します。	0
bit1,0	分周選択ビット bit1: cks1 bit0: cks0	00:fAD の 8 分周 (8/20MHz=400ns) 01:fAD の 4 分周 (4/20MHz=200ns) 10:fAD の 2 分周 (2/20MHz=100ns) 11:fAD の 1 分周 (1/20MHz=50ns) fAD とは、bit2 で設定したクロック源のことです。このクロックを何分周で使用するか選択します。遅くする必要はないので、いちばん速い 1 分周で使用します。	11

A/D モードレジスタ(ADMOD)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	1	1	0	0	1	1
16 進数	3				3			

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

どのアナログ入力端子を A/D 変換するか、設定します。

A/D 入力グループ 選択ビット		A/D 掃引端子数 選択ビット		bit3	bit2	bit1	bit0	アナログ入力端子
bit7	bit6	bit5	bit4					
0	0	0	0	0	0	0	0	AN0(P0_7)~AN1(P0_6)の 2 端子
0	0	0	1	0	0	0	0	AN0(P0_7)~AN3(P0_4)の 4 端子
0	0	1	0	0	0	0	0	AN0(P0_7)~AN5(P0_2)の 6 端子
0	0	1	1	0	0	0	0	AN0(P0_7)~AN7(P0_0)の 8 端子
0	1	0	0	0	0	0	0	AN8(P1_0)~AN9(P1_1)の 2 端子
0	1	0	1	0	0	0	0	AN8(P1_0)~AN11(P1_3)の 4 端子

※それ以外は設定禁止

今回は、AN0(P0_7)~AN7(P0_0)の 8 端子を選択します。A/D 入力選択レジスタ (ADINSEL)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	1	1	0	0	0	0
16 進数	3				0			

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

A/D を動作可能にします。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7	A/D 断線検出アシスト方式選択ビット(注 4)	0:変換前デイスチャージ 1:変換前プリチャージ A/D 断線検出アシストしませんのでどちらでも構いませんが、今回は"0"にしておきます。	0
bit6	A/D 断線検出アシスト機能許可ビット(注 4)	0:禁止 1:許可 A/D 断線検出アシストは使いません。	0
bit5	A/D スタンバイビット(注 3) adstby	0:A/D 動作停止(スタンバイ) 1:A/D 動作可能 A/D 動作可能にして A/D 変換できるようにします。この bit を"0"から"1"にしたときは、 ϕ A/D の 1 サイクル以上経過した後に A/D 変換を開始します。	1
bit4	8/10 ビットモード選択ビット bits	0:8 ビットモード 1:10 ビットモード A/D 変換を 10bit(0~1023)にするか、8bit(0~255)にするか選択します。今回は、10bit にします。	1
bit3~1		"000"を設定	000
bit0	拡張アナログ入力端子選択ビット(注 1) adex0	0:拡張アナログ入力端子を非選択 1:チップ内蔵基準電圧を選択(注 2) 拡張アナログ入力端子は使いません。	0

注 1. チップ内蔵基準電圧をアナログ入力として使用する場合、ADEX0 ビットを"1"(チップ内蔵基準電圧を選択)にした後に、OCVREFCR レジスタの OCVREFAN ビットを"1"(チップ内蔵基準電圧とアナログ入力を接続)にしてください。また、チップ内蔵基準電圧をアナログ入力として使用しない場合、OCVREFAN ビットを"0"(チップ内蔵基準電圧とアナログ入力を切断)にした後に、ADEX0 ビットを"0"(拡張アナログ入力端子を非選択)にしてください。

注 2. 単掃引モード、繰り返し掃引モードでは設定しないでください。

注 3. ADSTBY ビットを"0"(A/D 動作停止) から"1"(A/D 動作可能) にしたときは、 ϕ AD の 1 サイクル以上経過した後に A/D 変換を開始してください。

注 4. A/D 断線検出アシスト機能を許可するためには、ADDDAEN ビットを"1"(許可)にした後、ADDDAEL ビットで変換開始状態を選択してください。断線時の変換結果は、外付け回路によって変化します。本機能はシステムに合わせた評価を十分に行った上で、使用してください。

A/D 制御レジスタ 1 (ADCON1)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	1	1	0	0	0	0
16 進数	3				0			

④ φAD の 1 サイクル以上ウェイトを入れる

③の bit5 の A/D スタンバイビットを"1"にした場合、φ A/D の 1 サイクル以上経過した後に A/D 変換を開始しなければいけません。

そのウェイトを入れるため、アセンブリ言語の nop 命令を実行します。「ad.c」内では、アセンブリ言語は実行できないため、asm 命令というアセンブリ言語を実行できる命令を使って nop 命令を実行します。ちなみに、nop は「No Operation (何もしない)」命令で、この命令を実行するのに 1 サイクル分の時間がかかります。

プログラムを下記に示します。

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

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

A/D 変換を開始します。

設定 bit	上:ビット名 下:シンボル	内容	今回の内容
bit7~1		"0000000"を設定	0000 000
bit0	A/D 変換開始フラグ adst	0:A/D 変換停止 1:A/D 変換開始 A/D 変換を開始させるので"1"を設定します。	1

A/D 制御レジスタ 0(ADCON0)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	0	0	0	1
16 進数	0				1			

19.5.3 main 関数

AN7 端子、AN6 端子から A/D 値を取得、マイコンボード上の LED と実習基板 Ver.2 の LED へ値を出力します。

```

38 : void main( void )
39 : {
40 :     int ad7data, ad6data;
41 :
42 :     init();                /* 初期化                */
43 :
44 :     while( 1 ) {
45 :         // AD7を出力
46 :         ad7data = ad7;    /* AD7取得                */
47 :         ad7data = ad7data >> 6;
48 :         led_out( ad7data ); /* マイコンボードのLEDへ出力 */
49 :
50 :         // AD6を出力
51 :         ad6data = ad6;    /* AD6取得                */
52 :         ad6data = ad6data >> 6;
53 :         p6 = ad6data;    /* P6のbit3~0のLEDへ出力 */
54 :     }
55 : }
    
```

46 行	AD7 端子(P0_0)の A/D 変換値を取得し、ad7data 変数に格納します。
47 行	A/D 変換値は、0~1023(2進数で 11 1111 1111)の値です。右シフトを6ビット分行い、下位の6桁を捨てます。その結果、A/D 値は 0~15 の値になり、ad7data 変数に代入します。
48 行	0~15 に変換した A/D 値をマイコンボード上の LED に出力します。
50 行	AD6 端子(P0_1)の A/D 変換値を取得し、ad6data 変数に格納します。
51 行	A/D 変換値は、0~1023(2進数で 11 1111 1111)の値です。右シフトを6ビット分行い、下位の6桁を捨てます。その結果、A/D 値は 0~15 の値になり、ad6data 変数に代入します。
52 行	0~15 に変換した A/D 値を実習基板 Ver.2 の LED に出力します。

※A/D 変換値を取得するレジスタ

A/D 変換された結果は、A/D レジスタ 0~7(AD0~AD7)に格納されます。AD0~AD7 のどのレジスタに格納されるかは、アナログ入力端子によって変わります。アナログ入力端子と A/D レジスタの関係を下記に示します。

アナログ入力端子	読み込むレジスタ
AN0(P0_7)	AD0
AN1(P0_6)	AD1
AN2(P0_5)	AD2
AN3(P0_4)	AD3
AN4(P0_3)	AD4
AN5(P0_2)	AD5
AN6(P0_1)	AD6
AN7(P0_0)	AD7
AN8(P1_0)	AD0
AN9(P1_1)	AD1
AN10(P1_2)	AD2
AN11(P1_3)	AD3

19.6 演習

- (1) AN5 の A/D 変換値をマイコンボードの LED へ、AN4 の A/D 変換値を実習基板 Ver.2 の LED 部へ出力しなさい。
- (2) マイコンボードのディップスイッチの値 0~7 によって、実習基板 Ver.2 の LED 部へ AN0~AN7 の A/D 変換値を出力するようにしなさい。

20. パルスカウント(プロジェクト:timer_ra_counter)

20.1 概要

本章では、外部から入力したパルスの数をタイマ RA で数える方法を説明します。タイマ RA を使うことにより、プログラムで端子の状態をチェックしなくても、パルス数を数えることができます(オーバーフローは除く)。

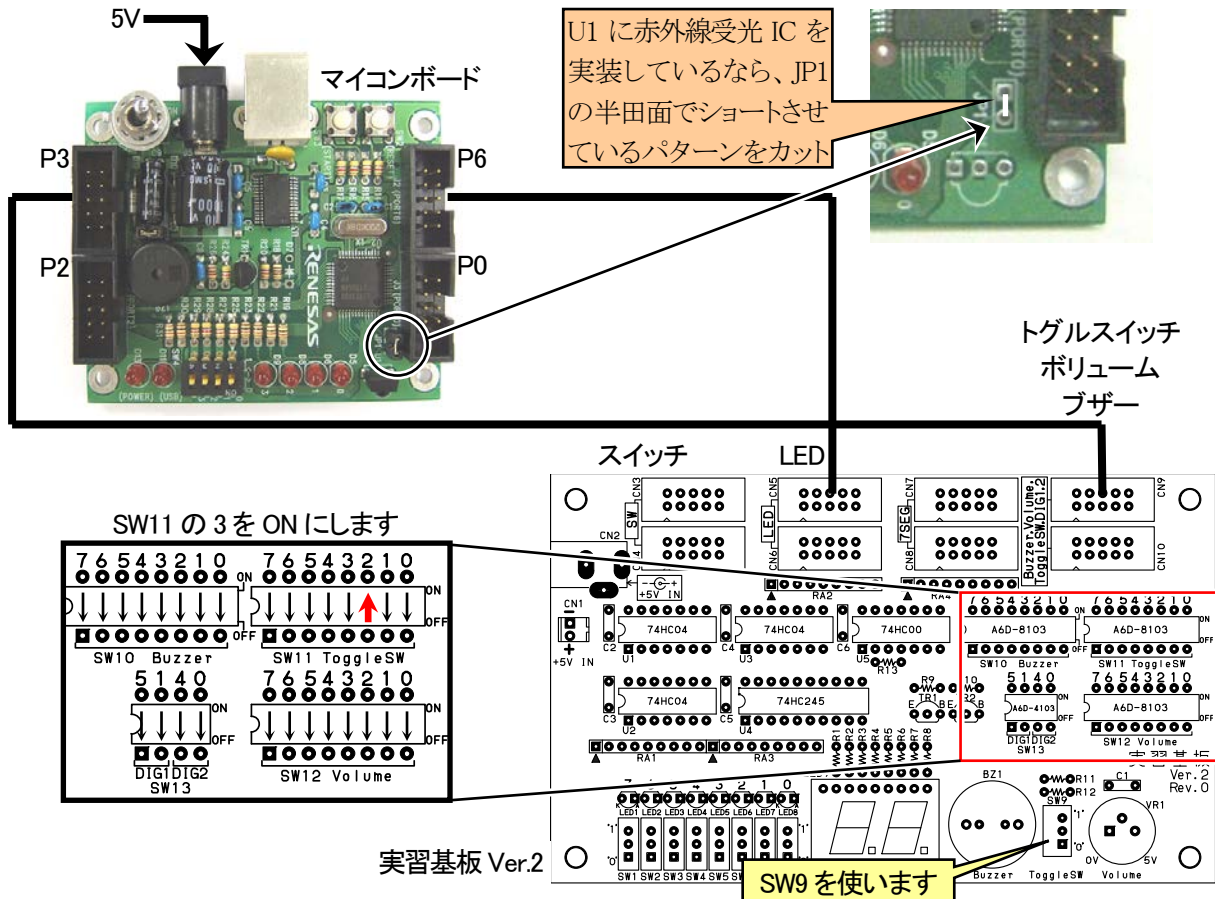
20.2 接続

■使用ポート

マイコンのポート	接続内容
P3.2 (J6)	実習基板 Ver.2 のトグルスイッチ部、またはロータリエンコーダなどを接続します。マイコンボードに赤外線受光 IC(U1)が実装されている場合は、J1 の 1 ピンと 2 ピンをショートしている半田面のパターンをカットしてください。
P6 (J2)	実習基板 Ver.2 の LED 部など、出力機器を接続します。

■接続例

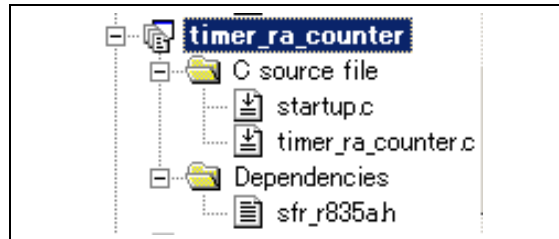
実習基板 Ver.2 を使ったときの接続例を下記に示します。



■操作方法

実習基板 Ver.2 のトグルスイッチ SW9 を上下させると、実習基板 Ver.2 の LED の値が「"0000 0000"→"0000 0001"→ … 」と、増えていきます("0"は消灯、"1"は点灯)。

20.3 プロジェクトの構成



	ファイル名	内容
1	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。このファイルは共通で、どのプロジェクトもこのファイルから実行されます。
2	timer_ra_counter.c	実際に制御するプログラムが書かれています。R8C/35A の内蔵周辺機能(SFR)の初期化も行います。
3	sfr_r835a.h	R8C/35A マイコンの内蔵周辺機能を制御するためのレジスタ (Special Function Registers)を定義したファイルです。

20.4 プログラム「timer_ra_counter.c」

```

1 : /******
2 : /* 対象マイコン R8C/35A */
3 : /* ファイル内容 タイマRAによるパルス数計測 */
4 : /* バージョン Ver. 1.20 */
5 : /* Date 2010.04.19 */
6 : /* Copyright ルネサスマイコンカーラリー事務局 */
7 : /* 日立インターメディックス株式会社 */
8 : /******
9 : /*
10 : 入力 : TRAI0端子(P3_2) (エンコーダなど)
11 : 出力 : P6_7-P6_0(LEDなど)
12 :
13 : P3_2に入力したパルスの数を、LEDへ出力します。
14 : */
15 :
16 : /*=====*/
17 : /* インクルード */
18 : /*=====*/
19 : #include "sfr_r835a.h" /* R8C/35A SFRの定義ファイル */
20 :
21 : /*=====*/
22 : /* シンボル定義 */
23 : /*=====*/
24 :
25 : /*=====*/
26 : /* プロトタイプ宣言 */
27 : /*=====*/
28 : void init( void );
29 :
30 : /******
31 : /* メインプログラム */
32 : /******
33 : void main( void )
34 : {
35 :     unsigned char d;
36 :
37 :     init(); /* 初期化 */
38 :
39 :     while( 1 ) {
40 :         p6 = 255 - trapre; /* パルスカウント下位8bit */
41 :         //p6 = 255 - tra; /* パルスカウント上位8bit */
42 :     }
43 : }
44 :
45 : /******
46 : /* R8C/35A スペシャルファンクションレジスタ(SFR)の初期化 */
47 : /******
48 : void init( void )
49 : {
50 :     int i;
51 :
52 :     /* クロックをXINクロック(20MHz)に変更 */
53 :     prc0 = 1; /* プロテクト解除 */
54 :     cml3 = 1; /* P4_6, P4_7をXIN-XOUT端子にする */
55 :     cm05 = 0; /* XINクロック発振 */
56 :     for(i=0; i<50; i++ ); /* 安定するまで少し待つ(約10ms) */
57 :     ocd2 = 0; /* システムクロックをXINにする */
58 :     prc0 = 0; /* プロテクトON */
59 :
60 :     /* ポートの入出力設定 */
61 :     prc2 = 1; /* PD0のプロテクト解除 */
62 :     pd0 = 0xe0; /* 7-5:LED 4:MicroSW 3-0:Sensor */
63 :     p1 = 0x0f; /* 3-0:LEDは消灯 */
64 :     pd1 = 0xdf; /* 5:RXD0 4:TXD0 3-0:LED */
65 :     pd2 = 0xfe; /* 0:PushSW */
66 :     pd3 = 0xfb; /* 4:Buzzer 2:パルス入力 */
67 :     pd4 = 0x83; /* 7:XOUT 6:XIN 5-3:DIP SW 2:VREF */
68 :     pd5 = 0x40; /* 7:DIP SW */
69 :     pd6 = 0xff; /* LEDなど出力 */
70 :
71 :     /* タイマRA イベントカウンタモード(パルスカウント)の設定 */
72 :     tramr = 0x02; /* イベントカウンタモードに設定 */
73 :     trasr = 0x03; /* TRAI0端子:P3_2に設定 */
74 :     traioc = 0x00; /* TRAI0端子の立ち上がりでカウント */
75 :     tracr = 0x01; /* TRAのカウント開始 */
76 : }
77 :
78 : /******
79 : /* end of file */
80 : /******

```

20.5 プログラムの解説

20.5.1 init 関数(タイマ RA の設定)

タイマ RA を設定するプログラムは、次のようになります。

```

71 :      /* タイマRA イベントカウンタモード(パルスカウント)の設定 */
72 :      tramr = 0x02;          /* イベントカウンタモードに設定 */
73 :      trasr = 0x03;          /* TRAI0端子:P3_2に設定 */
74 :      traioc = 0x00;         /* TRAI0端子の立ち上がりでカウント*/
75 :      tracr = 0x01;          /* TRAのカウンタ開始 */

```

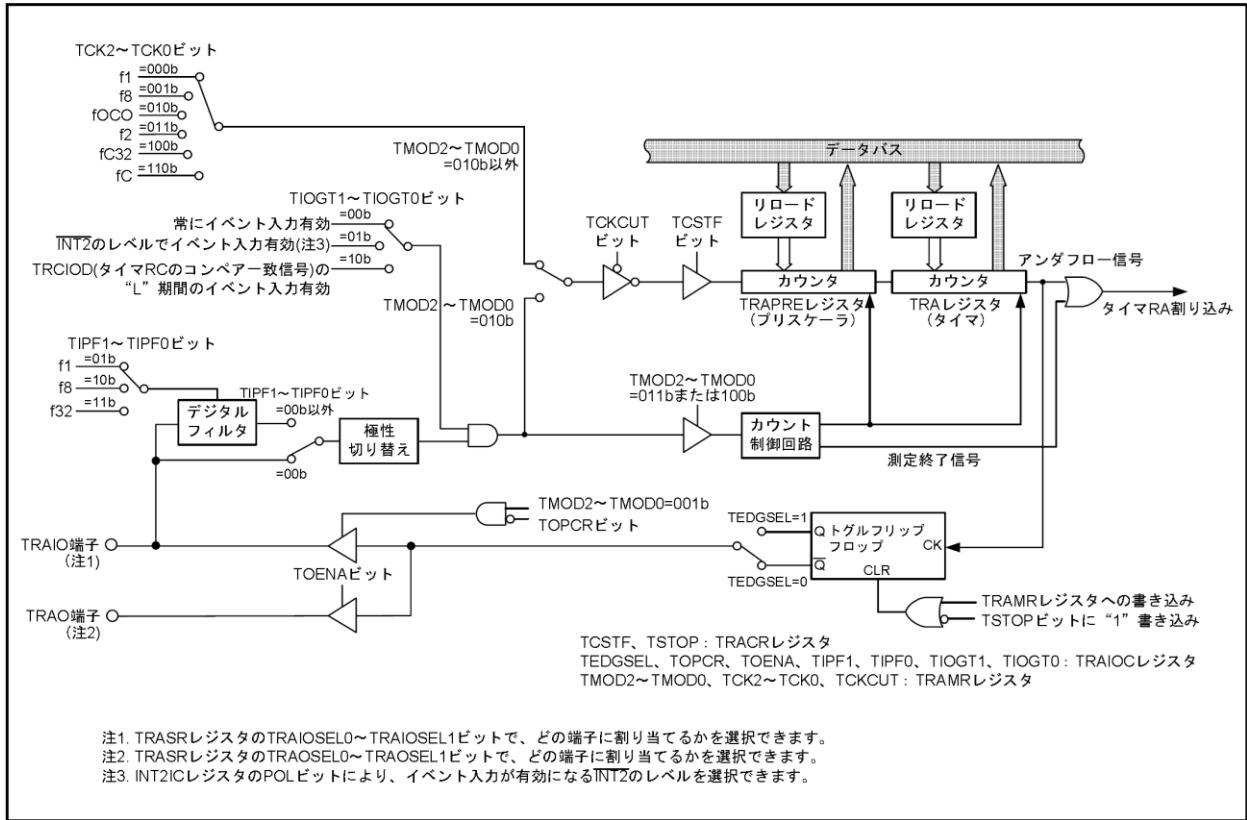
(1) タイマ RA とは

R8C/35A には、タイマ RA というタイマが 1 個内蔵されています。タイマ RA には、5 つのモードがあります。

モード	詳細
タイマモード	内部カウントソースをカウントするモードです。
パルス出力モード	内部カウントソースをカウントし、タイマのアンダフローで極性を反転したパルスを出力するモードです。
イベントカウンタモード	外部パルスをカウントするモードです。
パルス幅測定モード	外部パルスのパルス幅を測定するモードです。
パルス周期測定モード	外部パルスのパルス周期を測定するモード

本プロジェクトでは、イベントカウンタモードを使い、パルスの数を数えます。

(2) タイマ RA のブロック図



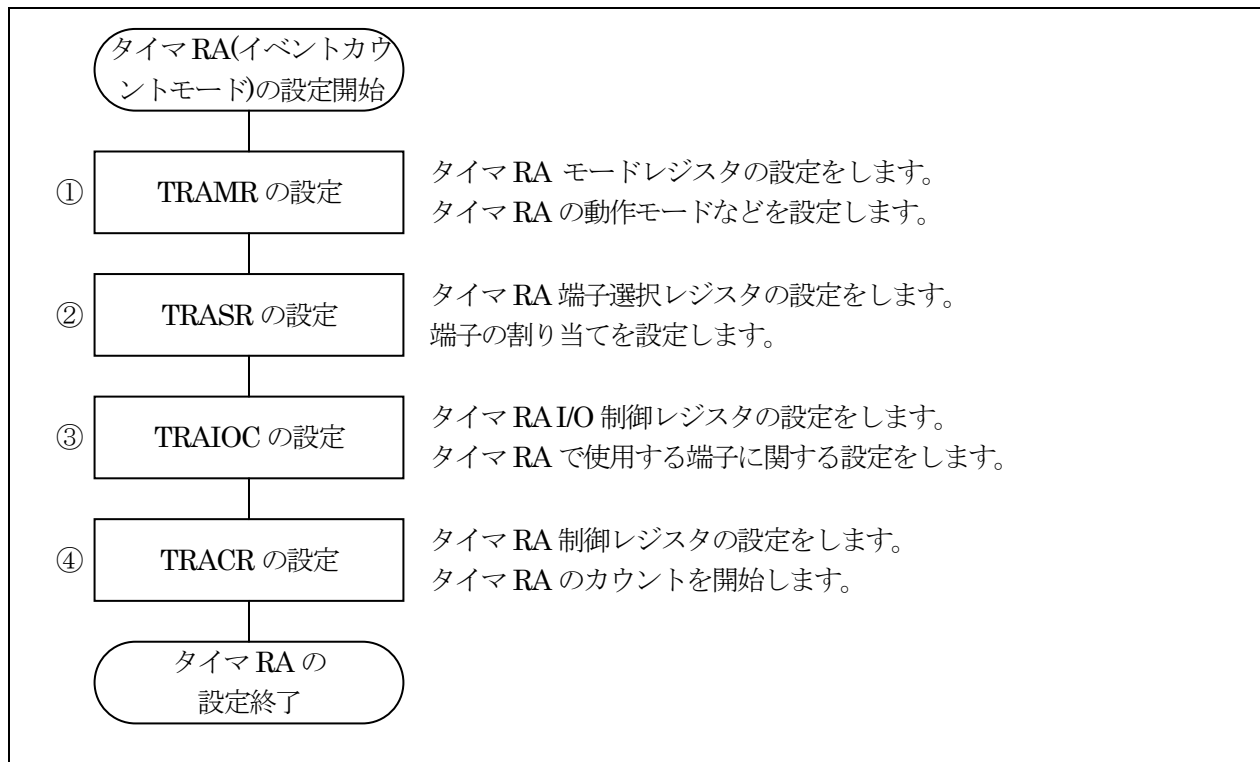
※タイマ RA の端子構成

端子名	割り当てる端子	入出力	機能
TRAI0	P1_5、P1_7 または P3_2	入出力	モードによって機能が異なります。 詳細は各モードを参照してください。
TRAO	P3_0、P3_7 または P5_6	出力	

今回のプログラムでは、P3_2 端子をパルス入力端子として使用します。プログラムで、P1_5 端子、P1_7 端子に変更することができます。他の端子をパルス入力端子として使用することはできません。

(3) タイマ RA の設定 (イベントカウンタモード)

今回は、タイマ RA を**イベントカウンタモード**に設定して、外部からのパルスを数えるように設定にします。レジスタの設定手順を下記に示します。



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

タイマ RA の動作モードなどを設定します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7	タイマ RA カウントソース遮断 ビット tckcut_tramr	0:カウントソース供給 1:カウントソース遮断 供給するので"0"を設定します。	0
bit6~4	タイマ RA カウントソース選択 ビット bit6:tck2_tramr bit5:tck1_tramr bit4:tck0_tramr	000:f1 (1/20MHz=50ns) 001:f8 (8/20MHz=400ns) 010:fOCO (オンチップオシレータクロック) 011:f2 (2/20MHz=100ns) 100:fC32 (32/XCIN クロック=今回は未接続) 101:設定しないでください 110:fC (1/XCIN クロック=今回は未接続) 111:設定しないでください イベントカウンタモードにすると、外部からのパルス入力になり、この部分の設定は無効です。何を設定しても構いませんが"000"にしておきます。	000
bit3		"0"を設定	0
bit2~0	タイマ RA 動作モード選択ビット bit2:tmod2_tramr bit1:tmod1_tramr bit0:tmod0_tramr	000:タイマモード 001:パルス出力モード 010:イベントカウンタモード 011:パルス幅測定モード 100:パルス周期測定モード 101:設定しないでください 110:設定しないでください 111:設定しないでください 今回は、イベントカウンタモードを使用するので、"010"を設定します。	010

※TRACR レジスタの TSTART ビットと TCSTF ビットがともに"0"(カウント停止)のときに、TRAMR レジスタを変更してください。

タイマ RA モードレジスタ(TRAMR)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	0	0	1	0
16 進数	0				2			

②タイマ RA 端子選択レジスタ(TRASR:Timer RA function select register)の設定

端子の割り当てを設定します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7~5		"000"を設定	000
bit4,3	TRAO 端子選択ビット bit4:traosel1 bit3:traosel0	00:P3_7 に割り当てる 01:P3_0 に割り当てる 10:P5_6 に割り当てる 11:設定しないでください 今回は、TRAO 端子をいませんのでどれを設定しても構いません。一応"00"を設定します。	00
bit2		"0"を設定	0
bit1,0	TRAI0 端子選択ビット bit1:traiosel1 bit0:traiosel0	00:TRAI0 端子は使用しない 01:P1_7 に割り当てる 10:P1_5 に割り当てる 11:P3_2 に割り当てる 今回は、TRAI0 端子をP3_2に割り当てます。"11"を設定します。	11

タイマ RA 端子選択レジスタ(TRASR)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	0	0	1	1
16 進数	0				3			

③タイマ RA I/O 制御レジスタ(TRAIOC:Timer RA I/O control register)の設定 [イベントカウンタモード時]

タイマ RA で使用する端子に関する設定をします。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7,6	TRAI0 イベント入力制御ビット bit7:tiogt1_traioc bit6:tiogt0_traioc	00:常にイベント入力有効 01: $\overline{\text{INT2}}$ のレベルでイベント入力有効(注 2) 10:TRCIOD(タイマ RC のコンペアー一致信号)の “L”期間のイベント入力有効 11:設定しないでください イベント入力有効にします。イベントとは外部からのパ ルスのことです。“00”を設定します。	00
bit5,4	TRAI0 入力フィルタ選択ビッ ト(注 1) bit4:tipf1_traioc bit3:tipf0_traioc	00:フィルタなし 01:フィルタあり、f1 でサンプリング 10:フィルタあり、f8 でサンプリング 11:フィルタあり、f32 でサンプリング フィルタは使用しません。“00”を設定します。	00
bit3		“0”を設定	0
bit2	TRAO 出力許可ビット toena_traioc	0:ポート P3_0、P3_7 または P5_6 1:TRAO 出力 TRAO 出力はしません。“0”を設定します。	0
bit1		“0”を設定	0
bit0	TRAI0 極性切り替えビット tedgsel_traioc	0:TRAI0 入力の立ち上がりエッジでカウント また、“L”から TRAO 出力開始 1:TRAI0 入力の立ち下がりエッジでカウント また、“H”から TRAO 出力開始 今回は TRAI0 入力は立ち上がりエッジでカウントする ようにしますので、“0”を設定します。立ち下がりにする なら“1”を設定してください。	0

注 1. TRAI0 端子から同じ値を 3 回連続してサンプリングした時点で入力が増定します。

注 2. INTEN レジスタの INT2PL ビットを“0”(片エッジ)にしてください。INT2IC レジスタの POL ビットを“0”(立ち
下がりエッジを選択)にすると、 $\overline{\text{INT2}}$ の“H”期間のイベント入力が有効になります。POL ビットを“1”(立ち
上がりエッジを選択)にすると、INT2 の“L”期間のイベント入力が有効になります。

タイマ RA I/O 制御レジスタ(TRAIOC)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	0	0	0	0
16 進数	0				0			

④タイマ RA 制御レジスタ(TRACR:Timer RA control register)の設定

タイマ RA のカウントを開始します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7,6		"00"を設定	00
bit5	タイマ RA アンダフローフラグ (注 3、4) tundf_tracr	0:アンダフローなし 1:アンダフローあり 注 4 より、"0"を設定します。	0
bit4	有効エッジ判定フラグ(注 3、4) tedgf_tracr	0:有効エッジなし 1:有効エッジあり(測定期間終了) 注 4 より、"0"を設定します。	0
bit3		0 を設定	0
bit2	タイマ RA カウント強制停止ビット(注 2) tstop_tracr	"1"を書くとカウントが強制停止します。読んだ場合、その値は"0"。 停止はしませんので"0"を設定します。	0
bit1	タイマ RA カウントステータスフラグ(注 1) tcstf_tracr	0:カウント停止 1:カウント中 読み込みのみ有効です。書き込むときは"0"にしておきます。	0
bit0	タイマ RA カウント開始ビット(注 1) tstart_tracr	0:カウント停止 1:カウント開始 タイマ RA のカウントを開始するので"1"を設定します。設定した瞬間から、カウントが開始されます。	1

注 1. TSTART、TCSTF ビットの使用上の注意事項については、ハードウェアマニュアルを参照してください。

注 2. TSTOP ビットに"1"を書くと、TSTART ビット、TCSTF ビット、TRAPRE レジスタ、TRA レジスタがリセット後の値になります。

注 3. プログラムで"0"を書くと、"0"になります("1"を書いても変化しません)。

注 4. タイマモード、パルス出力モード、イベントカウンタモードでは"0"にしてください。

タイマ RA 制御レジスタ(TRACR)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	0	0	0	1
16 進数	0				1			

20.5.2 main 関数

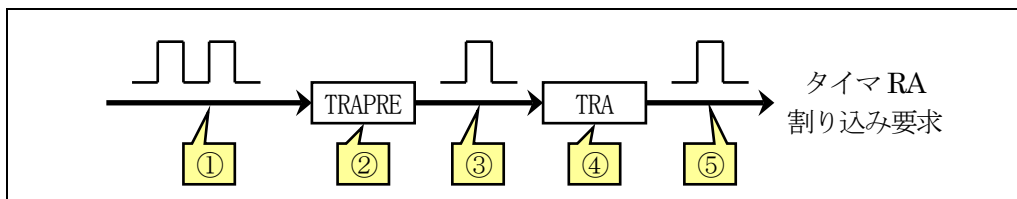
タイマ RA でパルスカウントを行い、カウントした値を LED へ出力します。

```

33 : void main( void )
34 : {
35 :     unsigned char d;
36 :
37 :     init();                /* 初期化                */
38 :
39 :     while( 1 ) {
40 :         p6 = 255 - trapre; /* パルスカウント下位8bit */
41 :         //p6 = 255 - tra;  /* パルスカウント上位8bit */
42 :     }
43 : }
    
```

40 行	<p>タイマ RA プリスケアラレジスタ(TRAPRE)は P3_2 端子からパルスが入力されるたびに値が減っていきます(ダウンカウント)。LED への出力は、パルスが入力されるたびに 0→1→2 と値を増やしたいので、TRAPRE を 255 で引いた値を LED へ出力します。TRAPRE の値と LED 出力値の例を下記に示します。</p> <p>TRAPRE が 255 なら $255 - \text{TRAPRE} = 255 - 255 = 0$ よって 0 が LED に出力される TRAPRE が 254 なら $255 - \text{TRAPRE} = 255 - 254 = 1$ よって 1 が LED に出力される TRAPRE が 253 なら $255 - \text{TRAPRE} = 255 - 253 = 2$ よって 2 が LED に出力される</p>
41 行	<p>41 行は現在、単一行コメント(行の先頭に「//」を付けるとプログラムは実行されない)になっています。タイマ RA レジスタ(TRA)の値を LED に出力したい場合は、40 行目をコメントにして、41 行目の「//」を取って、プログラムを実行するようにします。</p> <pre> 40 : //p6 = 255 - trapre; 41 : p6 = 255 - tra; </pre> <p>コメントにする!! //をとる!!</p>

タイマ RA プリスケアラレジスタ(TRAPRE)とタイマ RA レジスタ(TRA)の値は、P3_2 端子に入力されるパルスにより値が変わります。



①	タイマ RA 端子選択レジスタ(TRASR)の TRATIO 端子選択ビットで設定したパルスが入力されます。今回は、P3_2 に割り当てたので、P3_2 端子に入力されたパルスになります。
②	TRAPRE はダウンカウントです。初期値は 255 です。パルスが入力されると 255→254→253・・・と値が減っていきます。0 の次は 255 に戻ります。
③	TRAPRE が 0→255 になった瞬間、1 パルス出力されます。これは TRAPRE に 256 パルス入力されると 1 パルス出力されるということです。
④	TRA はダウンカウントです。初期値は 255 です。③のパルスが入力されると 255→254→253・・・と値が減っていきます。0 の次は 255 に戻ります。
⑤	TRA が 0→255 になった瞬間、1 パルス出力されます。このとき、割り込みを発生させることができます。今回は発生させる設定にしています。

21. タイマ RD による PWM 波形出力(リセット同期 PWM モード) (プロジェクト:timer_rd_doukipwm)

21.1 概要

本章では、PWM 波形を出力する方法を紹介します。今回は、タイマ RD をリセット同期 PWM モードで使用して、同じ周期の PWM 波形を 3 本、その波形を反転した波形を 3 本、1 周期ごとに反転する波形を 1 本、合計 7 本の波形を出力します。タイマ RD の初期設定後は、プログラムが関与しなくても PWM 波形を出力し続けます。プログラムでは、PWM 波形出力処理以外の処理をすることができます。

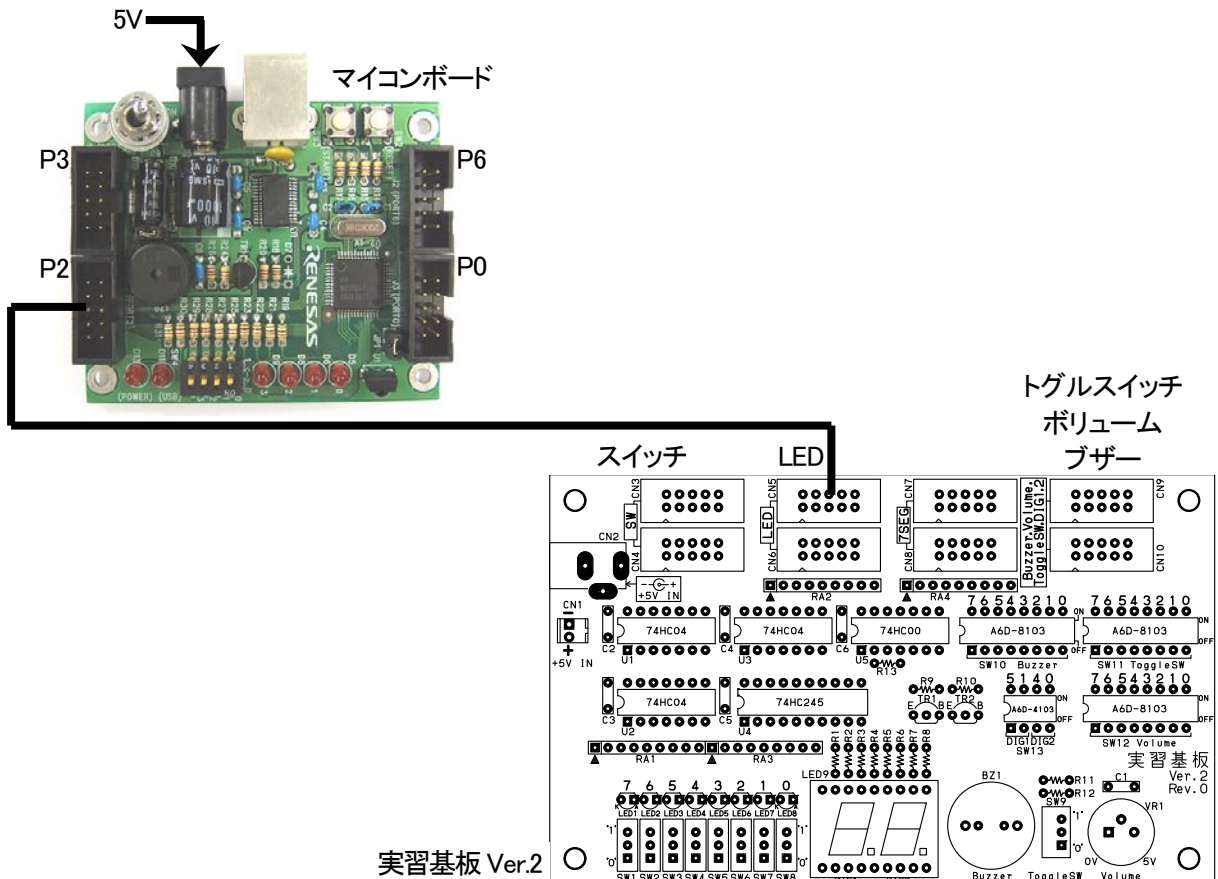
21.2 接続

■使用ポート

マイコンのポート	接続内容
P5_7、P4_5、P4_4、P4_3	マイコンボード上のディップスイッチです。
P2 (J7)	実習基板 Ver.2 の LED 部など、出力機器を接続します。

■接続例

実習基板 Ver.2 を使ったときの接続例を下記に示します。



■操作方法

マイコンボードのディップスイッチ(SW4)の値 0~15 によって、LED の点灯する明るさが変わります。このとき、どの LED が明るくなって、どの LED が暗くなるか観察してください。

21.3 プロジェクトの構成



	ファイル名	内容
1	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。このファイルは共通で、どのプロジェクトもこのファイルから実行されます。
2	timer_rd_doukipwm.c	実際に制御するプログラムが書かれています。R8C/35A の内蔵周辺機能(SFR)の初期化も行います。
3	sfr_r835a.h	R8C/35A マイコンの内蔵周辺機能を制御するためのレジスタ(Special Function Registers)を定義したファイルです。

21.4 プログラム「timer_rd_doukipwm.c」

```

1 : /******
2 : /* 対象マイコン R8C/35A */
3 : /* ファイル内容 タイマRDによるリセット同期PWM */
4 : /* バージョン Ver. 1.20 */
5 : /* Date 2010.04.19 */
6 : /* Copyright ルネサスマイコンカーラー事務局 */
7 : /* 日立インターメディアックス株式会社 */
8 : /******
9 : /*
10 : 入力：マイコンボードのディップスイッチ
11 : 出力：P2_1端子～P2_7端子からPWM出力
12 :
13 : ディップスイッチでデューティ比(ONの割合)を設定し、LEDに出力します。
14 : */
15 :
16 : /*=====*/
17 : /* インクルード */
18 : /*=====*/
19 : #include "sfr_r835a.h" /* R8C/35A SFRの定義ファイル */
20 :
21 : /*=====*/
22 : /* シンボル定義 */
23 : /*=====*/
24 :
25 : /*=====*/
26 : /* プロトタイプ宣言 */
27 : /*=====*/
28 : void init( void );
29 : unsigned char dipsw_get( void );
30 :

```

```

31 : /*****/
32 : /* メインプログラム */
33 : /*****/
34 : void main( void )
35 : {
36 :     init(); /* 初期化 */
37 :
38 :     while( 1 ) {
39 :         trdgrd0 = 39998 * dipsw_get() / 15;
40 :     }
41 : }
42 :
43 : /*****/
44 : /* R8C/35A スペシャルファンクションレジスタ(SFR)の初期化 */
45 : /*****/
46 : void init( void )
47 : {
48 :     int i;
49 :
50 :     /* クロックをXINクロック(20MHz)に変更 */
51 :     prc0 = 1; /* プロテクト解除 */
52 :     cm13 = 1; /* P4_6, P4_7をXIN-XOUT端子にする */
53 :     cm05 = 0; /* XINクロック発振 */
54 :     for(i=0; i<50; i++ ); /* 安定するまで少し待つ(約10ms) */
55 :     ocd2 = 0; /* システムクロックをXINにする */
56 :     prc0 = 0; /* プロテクトON */
57 :
58 :     /* ポートの入出力設定 */
59 :     prc2 = 1; /* PD0のプロテクト解除 */
60 :     pd0 = 0xe0; /* 7-5:LED 4:MicroSW 3-0:Sensor */
61 :     pd1 = 0x0f; /* 3-0:LEDは消灯 */
62 :     pd1 = 0xdf; /* 5:RXD0 4:TXD0 3-0:LED */
63 :     pd2 = 0xfe; /* 0:PushSW */
64 :     pd3 = 0xff; /* 4:Buzzer 2:IR */
65 :     pd4 = 0x83; /* 7:XOUT 6:XIN 5-3:DIP SW 2:VREF */
66 :     pd5 = 0x40; /* 7:DIP SW */
67 :     pd6 = 0xff;
68 :
69 :     /* タイマRD リセット同期PWMモードの設定 */
70 :     trdfcr = 0x01; /* リセット同期PWMモードに設定 */
71 :     trdmr = 0xf0; /* バッファレジスタ設定 */
72 :     trdoer1 = 0x01; /* 出力端子の選択 */
73 :     trdpsr0 = 0x68; /* TRDIOB0, C0, D0端子設定 */
74 :     trdpsr1 = 0x55; /* TRDIOA1, B1, C1, D1端子設定 */
75 :     trdcr0 = 0x23; /* ソースカウンタの選択:f8 */
76 :     trdgra0 = trdgrc0 = 39999; /* 周期 */
77 :     trdgrb0 = trdgrd0 = 0; /* P2_2端子のON幅設定 */
78 :     trdgral = trdgrcl = 0; /* P2_4端子のON幅設定 */
79 :     trdgrbl = trdgrdl = 0; /* P2_5端子のON幅設定 */
80 :     trdstr = 0x0d; /* TRD0カウンタ開始 */
81 : }
82 :
83 : /*****/
84 : /* ディップスイッチ値読み込み */
85 : /* 戻り値 スイッチ値 0~15 */
86 : /*****/
87 : unsigned char dipsw_get( void )
88 : {
89 :     unsigned char sw, sw1, sw2;
90 :
91 :     sw1 = (p5>>4) & 0x08; /* ディップスイッチ読み込み3 */
92 :     sw2 = (p4>>3) & 0x07; /* ディップスイッチ読み込み2, 1, 0 */
93 :     sw = sw1 | sw2; /* P5とP4の値を合わせる */
94 :
95 :     return sw;
96 : }
97 :
98 : /*****/
99 : /* end of file */
100 : /*****/

```

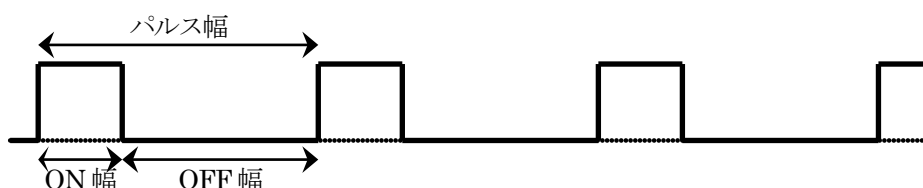
21.5 PWM とは？

モータのスピード制御を考えてみます。

モータを回したければ、電圧を加えます。止めたければ、電圧を加えなければいけません。では、その中間のスピードや10%、20%…など、細かくスピード調整したいときはどうすればよいのでしょうか。

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

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



デューティ比は下記で表すことができます。

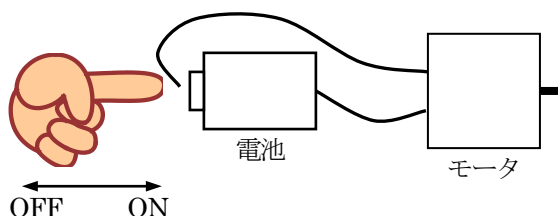
$$\text{デューティ比} = \text{ON 幅} / \text{パルス幅 (ON 幅 + OFF 幅)}$$

例えば、100ms のパルスに対して、ON 幅が 60ms なら、

$$\text{デューティ比} = 60\text{ms} / 100\text{ms} = 0.6 = 60\%$$

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

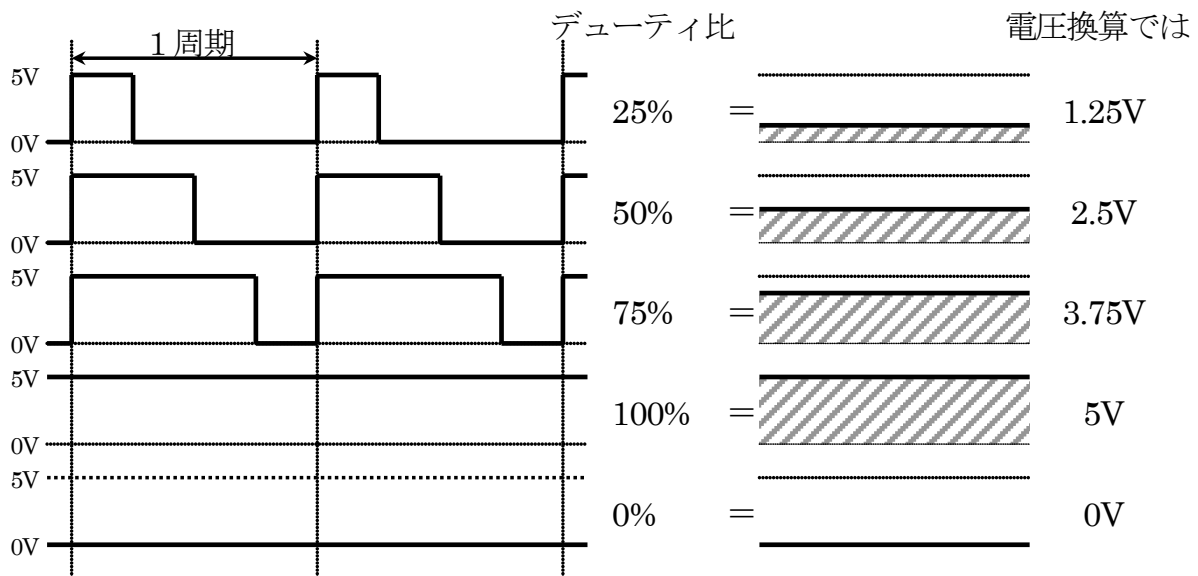
「PWM」と聞くと、何か難しく感じてしまいがちですが、下記のように手でモータと電池の線を「繋ぐ」、「離す」の繰り返し、それも PWM と言えます。繋いでいる時間が長いとモータは速く回ります。離している時間が長いとモータは少ししか回りません。人なら「繋ぐ」、「離す」の動作をコンマ数秒でしか行えませんが、マイコンならマイクロ数、またはミリ秒単位で行うことができます。



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

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

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



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

21.6 プログラムの解説

21.6.1 init 関数(タイマ RD の設定)

タイマ RD を使い、リセット同期 PWM モードの設定を行います。

```

69 :      /* タイマRD リセット同期PWMモードの設定*/
70 :      trdfcr = 0x01;          /* リセット同期PWMモードに設定 */
71 :      trdmr  = 0xf0;          /* バッファレジスタ設定 */
72 :      trdoerl = 0x01;         /* 出力端子の選択 */
73 :      trdpsr0 = 0x68;         /* TRDIOB0, C0, D0端子設定 */
74 :      trdpsr1 = 0x55;         /* TRDIOA1, B1, C1, D1端子設定 */
75 :      trdcr0  = 0x23;         /* ソースカウントの選択:f8 */
76 :      trdgra0 = trdgrc0 = 39999; /* 周期 */
77 :      trdgrb0 = trdgrd0 = 0;   /* P2_2端子のON幅設定 */
78 :      trdgral = trdgrc1 = 0;   /* P2_4端子のON幅設定 */
79 :      trdgrbl = trdgrd1 = 0;   /* P2_5端子のON幅設定 */
80 :      trdstr  = 0x0d;          /* TRD0カウント開始 */

```

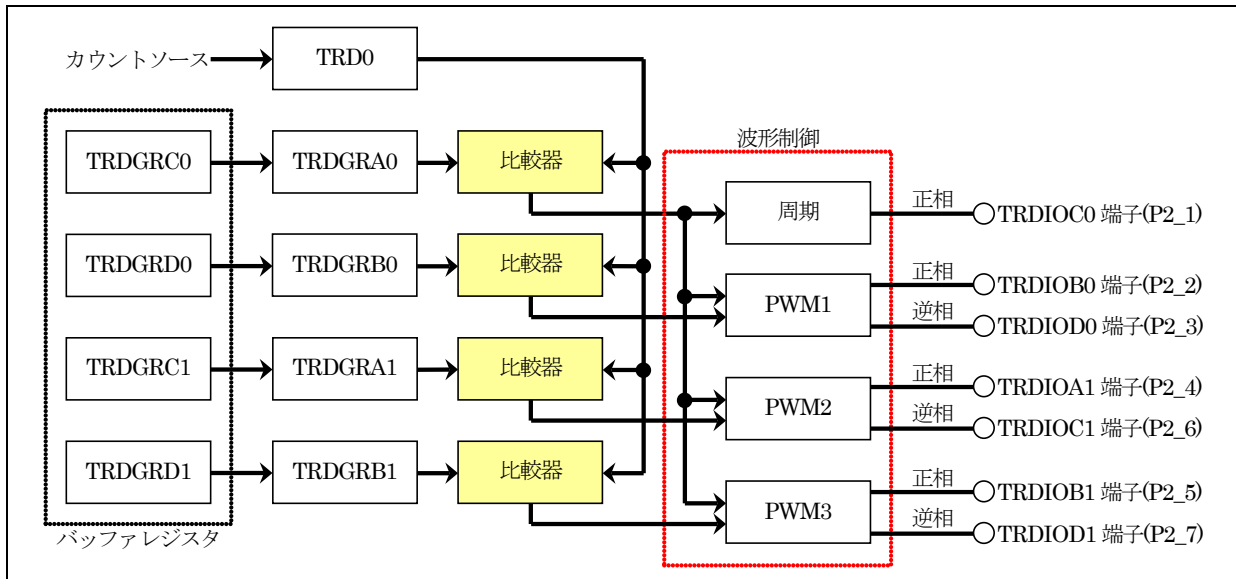
(1) タイマ RD とは

R8C/35A には、タイマ RD というタイマが 2 個内蔵されています。タイマ RD には、次の 5 種類のモードがあります。今回は、「**リセット同期 PWM モード**」を使います。

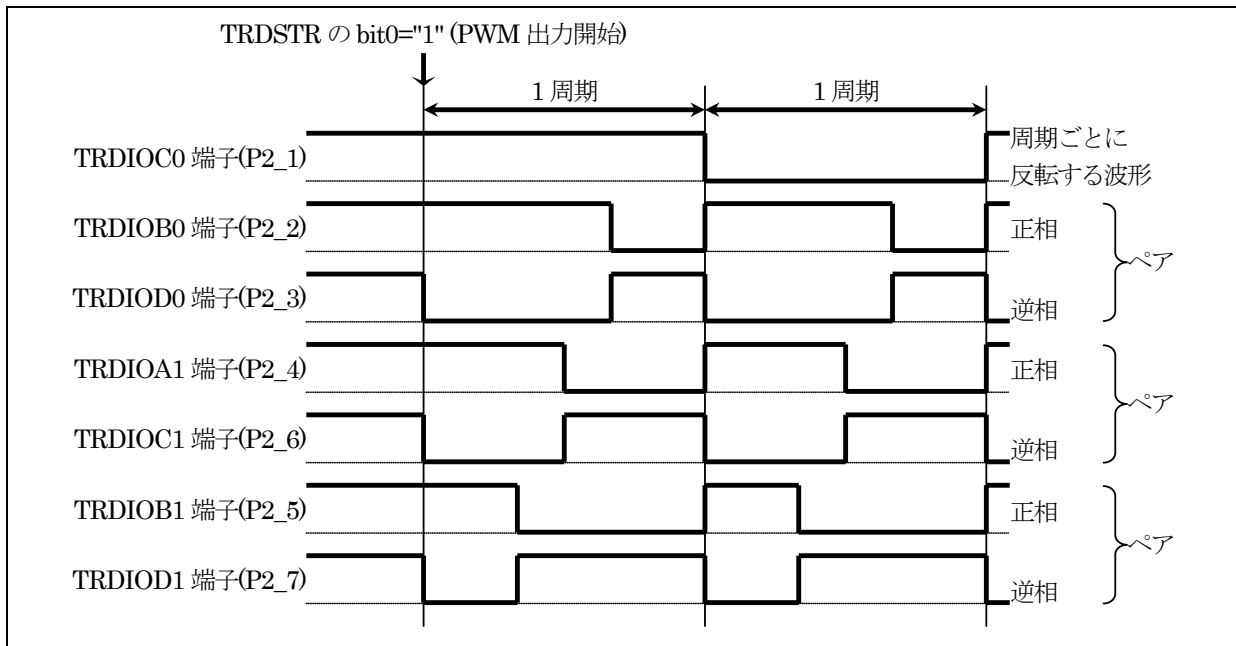
モード	詳細
タイマモード	タイマモードには、次の 2 つの機能があります。 <ul style="list-style-type: none"> ・入力キャプチャ機能 外部信号をトリガにしてカウンタの値をレジスタに取り込む機能 ・アウトプットコンペア機能 カウンタとレジスタの値の一致を検出する機能(検出時に端子出力変更可能)
PWM モード	任意の幅のパルスを連続して出力するモード
リセット同期 PWM モード	鋸波変調、短絡防止時間なしの三相波形(6 本)を出力するモード
相補 PWM モード	三角波変調、短絡防止時間ありの三相波形(6 本)を出力するモード
PWM3 モード	同一周期の PWM 波形(2 本)を出力するモード

(2) タイマ RD のブロック図

リセット同期 PWM モードのブロック図を下記に示します。リセット同期 PWM モードは、タイマ RD のチャンネル 0 とチャンネル 1 を組み合わせて使います。



PWM 波形の出力例を下記に示します。



(3) タイマ RD の設定 (リセット同期 PWM モード)

今回は、タイマ RD をリセット同期式 PWM モードで使用して、PWM 波形を正相 3 本、逆相 3 本、周期ごとに反転する波形の合計 7 本出力します。レジスタの設定手順を下記に示します。



①タイマ RD 機能制御レジスタ(TRDFCR:Timer RD function control register)の設定

タイマ RD の機能を設定します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7		"0"を設定	0
bit6	外部クロック入力選択ビット stclk_trdfer	0:外部クロック入力無効 1:外部クロック入力有効 今回は、内部のクロックを使用しますので、無効を選択 します。	0
bit5,4		"00"を設定	00
bit3	逆相出力レベル選択ビット (リセット同期 PWM モードまた は相補 PWM モード時) ols1_trdfer	0:初期出力"H"、アクティブレベル"L" 1:初期出力"L"、アクティブレベル"H" 今回は、"0"を設定します。	0
bit2	正相出力レベル選択ビット (リセット同期 PWM モードまた は相補 PWM モード時) ols0_trdfer	0:初期出力"H"、アクティブレベル"L" 1:初期出力"L"、アクティブレベル"H" 今回は、"0"を設定します。	0
bit1,0	コンビネーションモード選択 ビット bit1:cmd1_trdfer bit0:cmd0_trdfer	リセット同期 PWM モードでは"01"(リセット同期 PWM モード)にしてください	01

タイマ RD 機能制御レジスタ(TRDFCR)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	0	0	0	1
16 進数	0				1			

※正相出力レベル選択ビットと出力波形の関係

正相出力とは、P2_2 端子、P2_4 端子、P2_5 端子から出力される波形のことです。この端子から出力される波形のレベルをどうするか設定します。

bit2	波形	説明
0 (標準)	<p>出力波形</p> <p>TRDSTR の bit0="1" (PWM 出力開始)</p>	"1" からスタートする設定です。
1	<p>出力波形</p> <p>TRDSTR の bit0="1" (PWM 出力開始)</p>	"0" からスタートする設定です。

※逆相出力レベル選択ビットと出力波形の関係

逆相出力とは、P2_3 端子、P2_6 端子、P2_7 端子から出力される波形のことです。この端子から出力される波形のレベルをどうするか設定します。

bit3	波形	説明
0 (標準)	<p>出力波形</p> <p>TRDSTR の bit0="1" (PWM 出力開始)</p>	"0" からスタートする設定です。
1	<p>出力波形</p> <p>TRDSTR の bit0="1" (PWM 出力開始)</p>	"1" からスタートする設定です。

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

タイマ RD のジェネラルレジスタをバッファレジスタとして使用するかどうか設定します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7	TRDGRD1 レジスタ機能選択 ビット bfd1_trdmr	0:ジェネラルレジスタ 1:TRDGRB1 レジスタのバッファレジスタ バッファレジスタとして使用します。1 を設定します。	1
bit6	TRDGRC1 レジスタ機能選択 ビット bfc1_trdmr	0:ジェネラルレジスタ 1:TRDGRA1 レジスタのバッファレジスタ バッファレジスタとして使用します。1 を設定します。	1
bit5	TRDGRD0 レジスタ機能選択 ビット bfd0_trdmr	0:ジェネラルレジスタ 1:TRDGRB0 レジスタのバッファレジスタ バッファレジスタとして使用します。1 を設定します。	1
bit4	TRDGRC0 レジスタ機能選択 ビット bfc0_trdmr	0:ジェネラルレジスタ 1:TRDGRA0 レジスタのバッファレジスタ バッファレジスタとして使用します。1 を設定します。	1
bit3~0		"0000"を設定	0000

バッファレジスタについては、後述します。

タイマ RD モードレジスタ(TRDMR)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	1	1	1	1	0	0	0	0
16 進数	f				0			

③タイマ RD アウトプットマスタ許可レジスタ 1(TRDOER1:Timer RD output master enable register 1)の設定

リセット同期 PWM モードは 7 端子から PWM 波形を出力することができますが、出力するか、通常の I/O ポートとして使用するかを選択できます。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7	TRDIOD1(P2_7)出力禁止ビット ed1_trdoer1	0:出力許可 1:出力禁止(TRDIOD1 端子はプログラマブル入出力ポート) 今回は、出力を許可します。	0
bit6	TRDIOC1(P2_6)出力禁止ビット ec1_trdoer1	0:出力許可 1:出力禁止(TRDIOC1 端子はプログラマブル入出力ポート) 今回は、出力を許可します。	0
bit5	TRDIOB1(P2_5)出力禁止ビット eb1_trdoer1	0:出力許可 1:出力禁止(TRDIOB1 端子はプログラマブル入出力ポート) 今回は、出力を許可します。	0
bit4	TRDIOA1(P2_4)出力禁止ビット ea1_trdoer1	0:出力許可 1:出力禁止(TRDIOA1 端子はプログラマブル入出力ポート) 今回は、出力を許可します。	0
bit3	TRDIOD0(P2_3)出力禁止ビット ed0_trdoer1	0:出力許可 1:出力禁止(TRDIOD0 端子はプログラマブル入出力ポート) 今回は、出力を許可します。	0
bit2	TRDIOC0(P2_1)出力禁止ビット ec0_trdoer1	0:出力許可 1:出力禁止(TRDIOC0 端子はプログラマブル入出力ポート) 今回は、出力を許可します。	0
bit1	TRDIOB0(P2_2)出力禁止ビット eb0_trdoer1	0:出力許可 1:出力禁止(TRDIOB0 端子はプログラマブル入出力ポート) 今回は、出力を許可します。	0
bit0		"1"を設定	1

タイマ RD アウトプットマスタ許可レジスタ 1(TRDOER1)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	0	0	0	1
16 進数	0				1			

④タイマ RD 端子選択レジスタ 0(TRDPSR0:Timer RD function select register 0)の設定

PWM 波形の出力端子をどのポートに割り当てるか設定します。タイマ RD 端子選択レジスタ 0(TRDPSR0)では、TRDIOD0 端子、TRDIOC0 端子、TRDIOB0 端子の割り当てを設定します。

R8C/35A では、割り当てる端子は決まっております端子を変更することができません。PWM 端子として割り当てるか、割り当てないか(通常の I/O ポートとして使用するか)を設定します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7		"0"を設定	0
bit6	TRDIOD0 端子選択ビット trdioid0sel0	0:TRDIOD0 端子は使用しない 1:P2.3 に割り当てる 今回は、P2.3 に割り当てて、この端子から PWM 波形を出力します。	1
bit5,4	TRDIOC0 端子選択ビット bit5:trdioc0sel1 bit4:trdioc0sel0	00:TRDIOC0 端子は使用しない 01:設定しないでください 10:P2.1 に割り当てる 11:設定しないでください 今回は、P2.1 に割り当てて、この端子から PWM 波形(周期の波形)を出力します。	10
bit3,2	TRDIOB0 端子選択ビット bit3:trdiob0sel1 bit2:trdiob0sel0	00:TRDIOB0 端子は使用しない 01:設定しないでください 10:P2.2 に割り当てる 11:設定しないでください 今回は、P2.2 に割り当てて、この端子から PWM 波形を出力します。	10
bit1		"0"を設定	0
bit0	TRDIOA0/TRDCLK 端子選 択ビット trdioa0sel0	0:TRDIOA0/TRDCLK 端子は使用しない 1:P2.0 に割り当てる リセット同期 PWM モードを使用するときは"0"を設定します。	0

タイマ RD 端子選択レジスタ 0(TRDPSR0)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	1	1	0	1	0	0	0
16 進数	6				8			

⑤タイマ RD 端子選択レジスタ 1(TRDPSR1:Timer RD function select register 1)の設定

PWM 波形の出力端子をどのポートに割り当てるか設定します。タイマ RD 端子選択レジスタ 1(TRDPSR1)では、TRDIOD1 端子、TRDIOC1 端子、TRDIOB1 端子、TRDIOA1 端子の割り当てを設定します。

R8C/35A では、割り当てる端子は決まっております端子を変更することができません。PWM 端子として割り当てるか、割り当てないか(通常の I/O ポートとして使用するか)を設定します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7		"0"を設定	0
bit6	TRDIOD1 端子選択ビット trdioid1sel0	0:TRDIOD1 端子は使用しない 1:P2_7 に割り当てる 今回は、P2_7 に割り当てて、この端子から PWM 波形を出力します。	1
bit5		"0"を設定	0
bit4	TRDIOC1 端子選択ビット trdioc1sel0	0:TRDIOC1 端子は使用しない 1:P2_6 に割り当てる 今回は、P2_6 に割り当てて、この端子から PWM 波形を出力します。	1
bit3		"0"を設定	0
bit2	TRDIOB1 端子選択ビット trdiob1sel0	0:TRDIOB1 端子は使用しない 1:P2_5 に割り当てる 今回は、P2_5 に割り当てて、この端子から PWM 波形を出力します。	1
bit1		"0"を設定	0
bit0	TRDIOA1 端子選択ビット trdioa1sel0	0:TRDIOA1 端子は使用しない 1:P2_4 に割り当てる 今回は、P2_4 に割り当てて、この端子から PWM 波形を出力します。	1

タイマ RD 端子選択レジスタ 1(TRDPSR1)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	1	0	1	0	1	0	1
16 進数	5				5			

⑥タイマ RD 制御レジスタ 0(TRDCR0:Timer RD control register 0)の設定

タイマ RD カウンタ 0(TRD0)がカウントアップする時間などを選択します。

設定 bit	上:ビット名 下:シンボル	内容	今回の内容
bit7~5	TRD0 カウンタクリア選択ビット bit7:cclr2_trdcr0 bit6:cclr1_trdcr0 bit5:cclr0_trdcr0	リセット同期 PWM モードの場合は、“001”(TRDGRA0とのコンペアー一致で TRD0 レジスタクリア)に設定	001
bit4,3	外部クロックエッジ選択ビット (注 3) bit4:ckeg1_trdcr0 bit3:ckeg0_trdcr0	00:立ち上がりエッジでカウント 01:立ち下がりエッジでカウント 10:両エッジでカウント 11:設定しないでください 外部クロックは使いませんので何を設定しても変化ありません。今回は“00”を設定します。	00
bit2~0	カウントソース選択ビット bit2:tck2_trdcr0 bit1:tck1_trdcr0 bit0:tck0_trdcr0	000:f1 (1/20MHz=50ns) 001:f2 (2/20MHz=100ns) 010:f4 (4/20MHz=200ns) 011:f8 (8/20MHz=400ns) 100:f32 (32/20MHz=1600ns) 101:TRDCLK 入力(注 1)または fC2 (注 2) fC2 = 2/XCIN クロック=今回は未接続 110:fOCO40M (高速オンチップオシレータ 40MHz=今回は未接続) 111:fOCO-F(注 4) (高速オンチップオシレータを FRA2 で分周したクロック=今回は未接続) タイマ RD カウンタ 0(TRD0)がカウントアップする時間を設定します。今回は“011”を設定します。TRD0 は、400ns ごとに+1 していきます。	011

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

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

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

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

タイマ RD 制御レジスタ 0(TRDCR0)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	1	0	0	0	1	1
16 進数	2				3			

※タイマ RD 制御レジスタ 0(TRDCR0)カウントソース選択ビットの設定方法

タイマ RD 制御レジスタ 0(TRDCR0)のカウントソース選択ビット(bit2~0)で、タイマ RD カウンタ 0(TRD0)がどのくらいの間隔で+1 するか設定します。TRD0 は、0 からスタートして最大 65,535 までカウントアップします。65,535 の次は 0 に戻ります。PWM の周期や ON 幅は TRD0 の値を基準にするので、カウントアップする時間×65,536 以上の時間を設定することができません。

タイマ RD 制御レジスタ 0(TRDCR0)のカウントソース選択ビットの値と、周期の関係を下記に示します。

bit2~0	内容
000	タイマ RD カウンタ 0(TRD0)がカウントアップする時間を、f1 に設定します。時間は、 $f1/20\text{MHz}=1/20\text{MHz}=50\text{ns}$ 設定できる PWM 周期の最大は、 $50\text{ns} \times 65,536 = \mathbf{3.2768\text{ms}}$ よって、この時間以内の PWM 周期を設定する場合は"000"を設定、これ以上の PWM 周期を設定したい場合は次以降の値を検討します。
001	タイマ RD カウンタ 0(TRD0)がカウントアップする時間を、f2 に設定します。時間は、 $f2/20\text{MHz}=2/20\text{MHz}=100\text{ns}$ 設定できる PWM 周期の最大は、 $100\text{ns} \times 65,536 = \mathbf{6.5536\text{ms}}$ よって、この時間以内の PWM 周期を設定する場合は"001"を設定、これ以上の PWM 周期を設定したい場合は次以降の値を検討します。
010	タイマ RD カウンタ 0(TRD0)がカウントアップする時間を、f4 に設定します。時間は、 $f4/20\text{MHz}=4/20\text{MHz}=200\text{ns}$ 設定できる PWM 周期の最大は、 $200\text{ns} \times 65,536 = \mathbf{13.1072\text{ms}}$ よって、この時間以内の PWM 周期を設定する場合は"010"を設定、これ以上の PWM 周期を設定したい場合は次以降の値を検討します。
011	タイマ RD カウンタ 0(TRD0)がカウントアップする時間を、f8 に設定します。時間は、 $f8/20\text{MHz}=8/20\text{MHz}=400\text{ns}$ 設定できる PWM 周期の最大は、 $400\text{ns} \times 65,536 = \mathbf{26.2144\text{ms}}$ よって、この時間以内の PWM 周期を設定する場合は"011"を設定、これ以上の PWM 周期を設定したい場合は次以降の値を検討します。
100	タイマ RD カウンタ 0(TRD0)がカウントアップする時間を、f32 に設定します。時間は、 $f32/20\text{MHz}=32/20\text{MHz}=1600\text{ns}$ 設定できる PWM 周期の最大は、 $1600\text{ns} \times 65,536 = \mathbf{104.8576\text{ms}}$ よって、この時間以内の PWM 周期を設定する場合は"100"を設定します。 これ以上の PWM 周期を設定することはできません。 これ以上の PWM 周期を設定しなくても良いように、回路側を工夫してください。

今回は、PWM 波形の周期を 16ms にするので、

- ①"000"の設定…最大の PWM 周期は 3.2768ms、今回設定したい 16ms の周期を設定できないので不可
- ②"001"の設定…最大の PWM 周期は 6.5536ms、今回設定したい 16ms の周期を設定できないので不可
- ③"010"の設定…最大の PWM 周期は 13.1072ms、今回設定したい 16ms の周期を設定できないので不可
- ④"011"の設定…最大の PWM 周期は 26.2144ms、今回設定したい 16ms の周期を設定できるので OK

よって、"011"を設定します。

⑦タイマ RD ジェネラルレジスタ A0(TRDGRA0:Timer RD General register A0)、
タイマ RD ジェネラルレジスタ C0(TRDGRC0:Timer RD General register C0)の設定

タイマ RD ジェネラルレジスタ A0(TRDGRA0)に値を設定することによって、出力する PWM 波形の周期を設定します。PWM 周期は、下記の式で決まります。

$$\text{PWM 周期} = \text{タイマ RD カウンタ 0 のカウントソース} \times (\text{TRDGRA0} + 1)$$

TRDGRA0 を左辺に移動して、TRDGRA0 を求める式に変形します。

$$\text{TRDGRA0} = \text{PWM 周期} / \text{タイマ RD カウンタ 0 のカウントソース} - 1$$

今回、PWM 波形の周期を 16ms にします。タイマ RD カウンタ 0 のカウントソースとは、タイマ RD 制御レジスタ 0(TRDCR0)の bit2~0 で設定した時間のことで、今回は 400ns に設定しています。よって、タイマ RD ジェネラルレジスタ A0(TRDGRA0)は次のようになります。

$$\begin{aligned} \text{TRDGRA0} &= \text{周期} / \text{カウントソース} - 1 \\ \text{TRDGRA0} &= (16 \times 10^{-3}) / (400 \times 10^{-9}) - 1 \\ \text{TRDGRA0} &= 40000 - 1 = 39999 \end{aligned}$$

TRDGRA0 の値は、65,535 以下にする必要があります。今回の結果は、65,535 以下なので、400ns の設定で大丈夫です。65,536 以上になった場合、タイマ RD 制御レジスタ 0(TRDCR0)のカウントソース選択ビット(bit2~0)の設定を大きい時間にしてください。

■設定のポイント

※1…タイマ RD ジェネラルレジスタ A0(TRDGRA0)を使うときは、タイマ RD ジェネラルレジスタ C0(TRDGRC0)をバッファレジスタに設定して、ペアで使用してください。

※2…タイマ RD ジェネラルレジスタ A0(TRDGRA0)の設定は、イニシャライズ時に 1 回だけ行ってください。2 回目以降、周期を変更したい場合は、タイマ RD ジェネラルレジスタ C0(TRDGRC0)に値を設定してください。

※3…タイマ RD ジェネラルレジスタ C0(TRDGRC0)のイニシャライズ時に設定する値は、タイマ RD ジェネラルレジスタ A0(TRDGRA0)と同じ値にしてください。

プログラム例を下記に示します。

```
void main( void )
{
    // メインプログラム
    init();                               /* レジスタの初期化 */
    ~~~~~
    trdgrc0 = 19999;                       /* 2 回目以降は必ず TRDGRC0 へ行う */
    ~~~~~
    trdgrc0 = 9999;                         /* 2 回目以降は必ず TRDGRC0 へ行う */
}
void init( void )
{
    // レジスタの初期化
    ~~~~~
    trdgra0 = trdgrc0 = 39999;             /* 1 回だけ TRDGRA0 に値を設定 */
                                           /* TRDGRC0 にも同じ値を設定する */
    ~~~~~
}
```

詳しくは、後述するバッファ動作で説明します。

⑧タイマ RD ジェネラルレジスタ B0(TRDGRB0:Timer RD General register B0)、
タイマ RD ジェネラルレジスタ D0(TRDGRD0:Timer RD General register D0)の設定

タイマ RD ジェネラルレジスタ B0(TRDGRB0)に値を設定することによって、P2_2 端子から出力される PWM 波形の ON 幅を設定します。P2_3 端子からは、その反転した波形が出力されます。

P2_2 端子から出力される PWM 波形の ON 幅は、下記の式で決まります。

$$\text{P2}_2 \text{ 端子から出力される PWM 波形の ON 幅} = \text{タイマ RD カウンタ 0 のカウントソース} \times (\text{TRDGRB0} + 1)$$

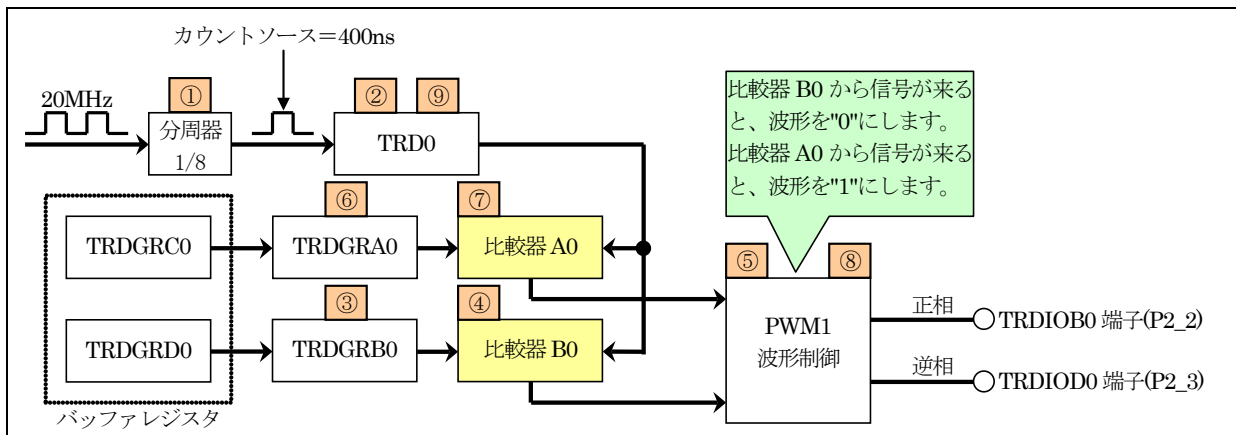
TRDGRB0 を左辺に移動して、TRDGRB0 を求める式に変形します。

$$\text{TRDGRB0} = \text{P2}_2 \text{ 端子から出力される PWM 波形の ON 幅} / \text{タイマ RD カウンタ 0 のカウントソース} - 1$$

今回、タイマ RD カウンタ 0 のカウントソースは 400ns です。例えば ON 幅を 1ms にするなら、TRDGRB0 の値は次のようになります。よって、タイマ RD ジェネラルレジスタ B0(TRDGRB0)は次のようになります。

$$\begin{aligned} \text{TRDGRB0} &= \text{ON 幅} / \text{カウントソース} - 1 \\ \text{TRDGRB0} &= (1 \times 10^{-3}) / (400 \times 10^{-9}) - 1 \\ \text{TRDGRB0} &= 2500 - 1 = 2499 \end{aligned}$$

■仕組み



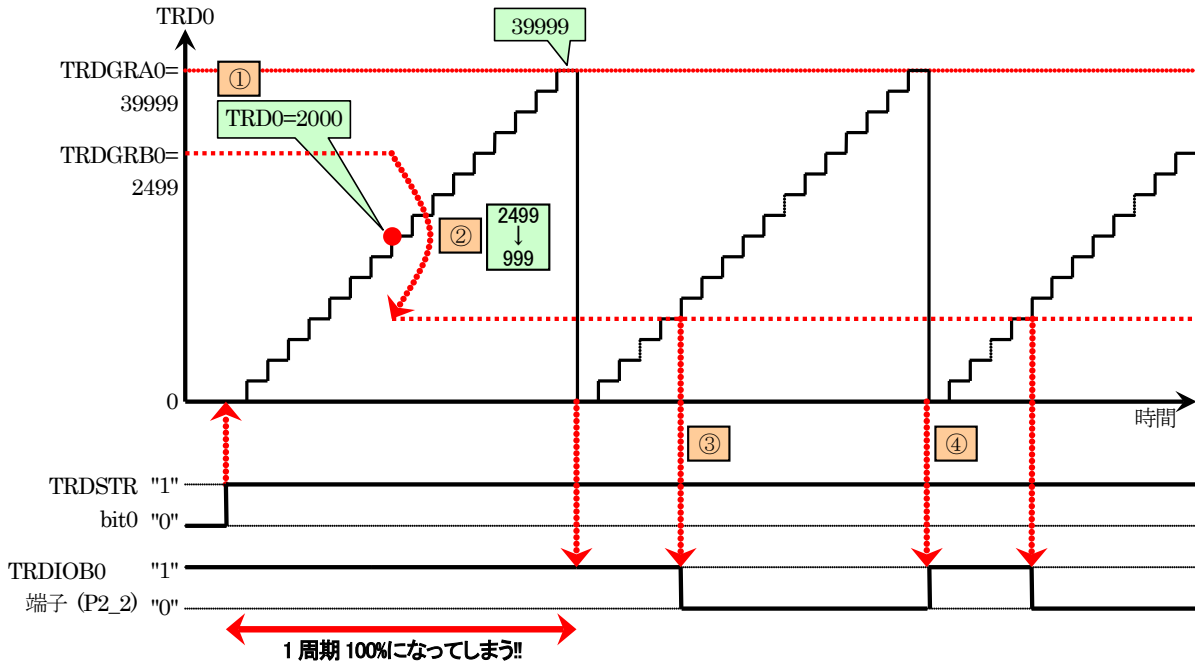
①	分周器には、クリスタルの 20MHz のパルスが入力されます。分周器は、タイマ RD 制御レジスタ 0 (TRDCR0) のカウントソース選択ビットで何分周にするか設定します。今回は f8 を選択しています。これは、8 分周することです。よって分周器からの出力は、 $8 / 20\text{MHz} = 2.5\text{MHz}$ のパルスが出力されます。時間は、 $1 / 2.5\text{M} = 400\text{ns}$ となります。
②	TRD0 は、400ns ごとにカウントアップ (+1 ずつ) します。TRD0 の最大値は 65,535 で、その次の値は 0 に戻りカウントアップし続けます。

<p>③ ④</p>	<p>TRDGRB0 には、波形の ON 幅を設定します。 TRDGRB0 と TRD0 の値は、常に比較器 B0 によって比較されています。次の式が成り立つと、PWM1 波形制御回路へ信号を送ります。</p> $\text{TRDGRB0} + 1 = \text{TRD0}$ <p>TRD0 は 400ns ごとに +1 します。TRDGRB0 には 2499 が設定されているとします。マイコンが起動したとき、TRD0 は 0 なので</p> $2499 (\text{TRDGRB0}) + 1 \neq 0 (\text{TRD0})$ <p>となり、成り立ちません。400ns 後、TRD0 は +1 されて 1 になります。</p> $2499 (\text{TRDGRB0}) + 1 \neq 1 (\text{TRD0})$ <p>まだ成り立ちません。式が成り立つ TRD0 が 2500 になるには $2500 \times 400\text{ns} = 1\text{ms}$ です。よって、1ms 経つと</p> $2499 (\text{TRDGRB0}) + 1 = 2500 (\text{TRD0})$ <p>が成り立ち、比較器 B0 から信号が PWM1 波形制御部分へ出力されます。</p>
<p>⑤</p>	<p>PWM 波形制御部分は、比較器 B0 から信号が送られてくると、波形を“0”にします。</p>
<p>⑥ ⑦</p>	<p>TRDGRA0 には、波形の周期を設定します。 TRDGRA0 と TRD0 の値は、常に比較器 A0 によって比較されています。次の式が成り立つと、PWM1 波形制御回路へ信号を送ります。</p> $\text{TRDGRA0} + 1 = \text{TRD0}$ <p>TRD0 は 400ns ごとに +1 します。TRDGRA0 には 39999 が設定されているとします。マイコンが起動したとき、TRD0 は 0 なので</p> $39999 (\text{TRDGRA0}) + 1 \neq 0 (\text{TRD0})$ <p>となり、成り立ちません。400ns 後、TRD0 は +1 されて 1 になります。</p> $39999 (\text{TRDGRA0}) + 1 \neq 1 (\text{TRDGRA0})$ <p>まだ成り立ちません。式が成り立つ TRD0 が 40000 になるには $40000 \times 400\text{ns} = 16\text{ms}$ です。よって、16ms 経つと</p> $39999 (\text{TRDGRA0}) + 1 = 40000 (\text{TRD0})$ <p>が成り立ち、比較器 A0 から信号が PWM1 波形制御部分へ出力されます。</p>
<p>⑧ ⑨</p>	<p>PWM 波形制御部分は、比較器 A0 から信号が送られてくると、波形を“1”にします。 これと同時に、比較器 A0 は TRD0 の値を 0 にクリアします。TRD0 は、0 からカウントし直します。</p>

■バッファレジスタはなぜ必要か

リセット同期 PWM モードを使用する場合、バッファレジスタというレジスタを使います。なぜバッファレジスタを使わなければいけないのでしょうか。

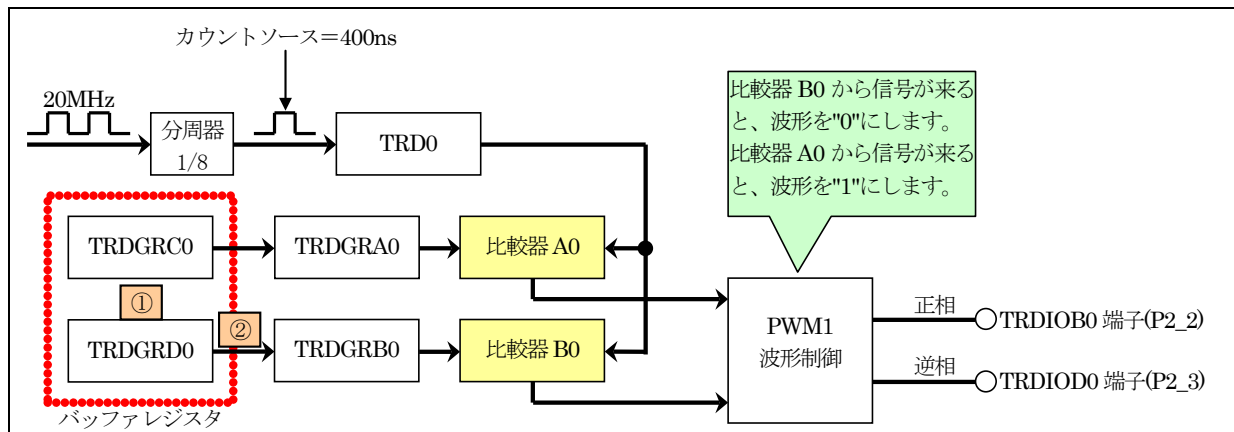
例えば、TRDGRB0=2499、TRD0=2000 のときに、プログラムで TRDGRB0 の値を 999 に変更したとします。



<p>① ②</p>	<p>TRD0 が 2000 のときに(①)、プログラムで TRDGRB0 の値を 2499 から 999 に変更します(②)。TRD0 の値が 400ns ごとに増えていき 40000 になりました。40000 になると「TRD0=(TRDGRA0+1)」が成り立ち、P2_2 端子が「1」になります。同時に TRD0 が 0 にクリアされます。この間、「TRD0=(TRDGRB0+1)」が成り立つ条件がありませんでした。そのため、今回の 1 周期分すべてが「1」となり 100%出力になってしまいます。</p> <p>今回のように、「TRD0=(TRDGRB0+1)」が成り立つ前に TRDGRB0 の値を TRD0 より小さい値に変更すると、PWM100%という想定外の値になります。</p>
<p>③</p>	<p>次の周期は、TRD0=(TRDGRB0+1)になるため、P2_2 端子が「0」になります。</p>
<p>④</p>	<p>TRD0=(TRDGRA0+1)になるため、P2_2 端子が「1」になります。同時に TRD0 が 0 になります。</p>

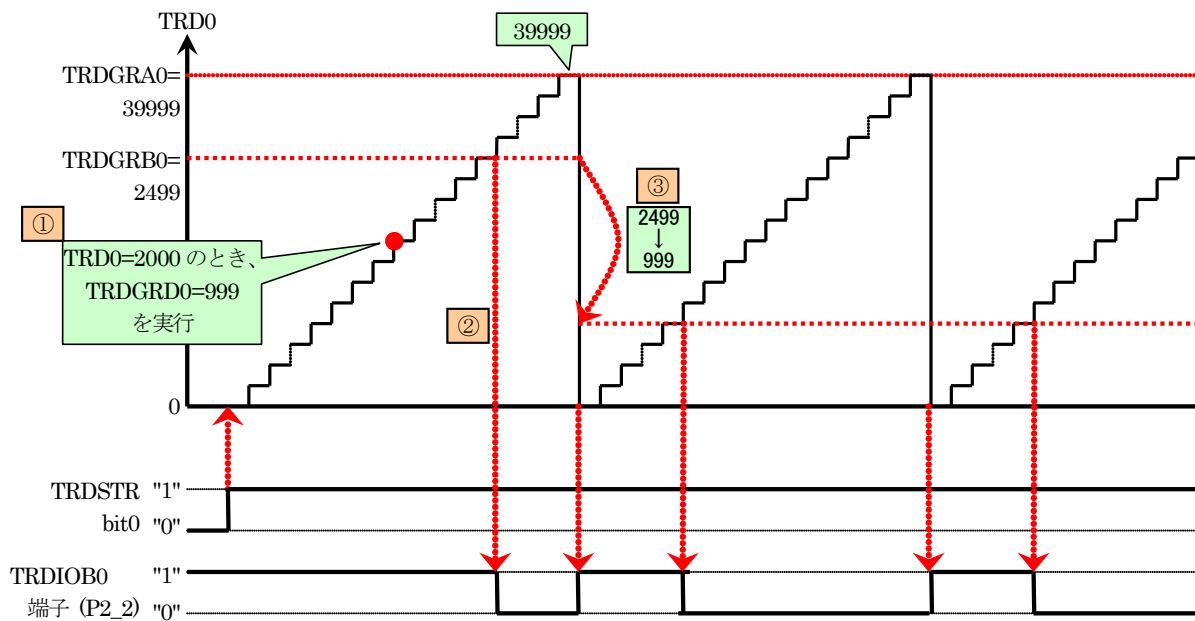
このように、TRDGRB0 の値を変えるタイミングによっては PWM100%という想定外の波形になってしまいます。モータを制御していたら、一瞬ですが 100%の回転となってしまう、暴走してしまうかもしれません。

そこで、バッファレジスタというのを使います(点線で囲った部分)。



<p>①</p>	<p>TRDGRB0 のバッファレジスタは、TRDGRD0 です。 P2.2 端子の ON 幅の設定は、TRDGRB0 ではなく TRDGRD0 に値を設定することにより行います。 プログラム例) TRDGRD0 = 2499; プログラムで TRDGRB0 の値を、直接変更しません。直接変更すると、前記のとおり 100% の波形が出力されることがあります。ただし、最初は TRDGRB0 に値が設定されていけませんので、イニシャライズ時に 1 回だけ値を設定してください。このとき、TRDGRB0 と TRDGRD0 には同じ値を設定します。</p>
<p>②</p>	<p>「TRD0=(TRDGRA0+1)」が成り立つと、下記が実行されます。 ・P2.2 端子が "1" になります。 ・TRD0 の値が 0 にクリアされます。 ・TRDGRD0 の値が、TRDGRB0 に転送されます。</p> <p>TRDGRB0 の値は、TRD0 の値が 0 になると同時に更新されます。これ以外のタイミングで TRDGRB0 の値が変わることはありません。そのため、前記の不具合が発生しないこととなります。これがバッファレジスタを使う目的です。</p>

TRD0 の値と波形の関係を図解すると下記のようになります。



①	TRD0 が 2000 のときに、プログラムで TRDGRD0 の値を 2499 から 999 に変更します。
②	TRDGRB0 の値は変わっていません。そのため、「TRD0=(TRDGRB0+1)」が成り立ち、P2_2 端子が"0"になります。
③	<p>「TRD0=(TRDGRA0+1)」が成り立つと、下記が実行されます。</p> <ul style="list-style-type: none"> ・P2_2 端子が"1"になります。 ・TRD0 の値が 0 になります。 <p>TRDGRD0 の値が、TRDGRB0 に転送されます。</p> <p>このタイミングで、TRDGRB0 の値が 999 に変更されます。次に「TRD0=(TRDGRB0+1)」が成り立つタイミングで、P2_2 端子が"0"に成ります。</p>

このように、バッファレジスタを使うと PWM 波形が 100%になるという不具合が発生しなくなります。

■0%出力

P2_2 端子が"0"になる時間、すなわち P2_2 端子が"1"の時間は、下記のようにした。

$$\text{P2}_2 \text{ 端子が"1"の時間} = (\text{TRDGRB0} + 1) \times \text{TRD0 が+1 する間隔}$$

0%出力にするには、"1"の時間を 0 にすれば良いことになります。すなわち TRDGRB0 の値を 0 にします。

$$\begin{aligned} \text{P2}_2 \text{ 端子が"1"の時間} &= (0 + 1) \times 400\text{ns} \\ &= 1 \times 400\text{ns} = 400\text{ns} \end{aligned}$$

このように、TRDGRB0 に 0 を代入しても完全な"0"になりません。400ns の時間だけ"1"になります。

リセット同期 PWM モードでは特殊な方法で 0%にします。それは、TRDGRB0 の値を TRDGRA0 と同じ値にします。そうすると、次のことが同時に起こります。

- ・「TRD0=(TRDGRB0 + 1)」が成り立ち、P2_2 端子を"0"にします。
- ・「TRD0=(TRDGRA0 + 1)」が成り立ち、P2_2 端子を"1"にします。

相反することが同時に起こります。**この場合 R8C/35A は、P2_2 端子を"0"にすることを優先させます。**

よって、TRDGRB0=TRDGRA0 にすると端子を"0"にすることが優先され、端子を"1"にする動作を行いません。

PWM 出力を 0%にしたい場合、次のようにプログラムします。設定は、TRDGRB0 ではなく、バッファレジスタである TRDGRD0 に設定します。

```
trdgrd0 = trdgra0; // バッファレジスタに設定する
```

■100%出力

「TRD0=(TRDGRB0 + 1)」が成り立つと、P2_2 端子が"0"になります。100%出力にするには、「TRD0=(TRDGRB0 + 1)」が成り立たない値を TRDGRB0 に設定すれば良いことになります。TRD0 は、(TRDGRA0 + 1)と一致すると 39999→0 になります。すなわち、TRDGRA0 より大きい値を設定します。今回は、TRDGRA0 より 1 つ大きい値を設定することになります。

PWM 出力を 100%にしたい場合、次のようにプログラムします。

```
trdgrd0 = 40000; // バッファレジスタに設定する
```

ただし、上記プログラムは、TRDGRA0 が 39999(周期が 16ms)のときだけです。TRDGRA0 がどのような値でも 100%にするために、次のプログラムの方が良いでしょう。

```
trdgrd0 = trdgra0 + 1; // バッファレジスタに設定する
```

このとき、TRDGRA0 が 65535 のときは、TRDGRD0 に 65536 を設定することはできないので、100%出力はできません。

■設定のポイント

- ※1…タイマ RD ジェネラルレジスタ B0(TRDGRB0)を使うときは、タイマ RD ジェネラルレジスタ D0(TRDGRD0)をバッファレジスタに設定して、ペアで使用してください。
- ※2…タイマ RD ジェネラルレジスタ B0(TRDGRB0)の設定は、イニシャライズ時に 1 回だけ行ってください。2 回目以降、ON 幅を変更したい場合は、タイマ RD ジェネラルレジスタ D0(TRDGRD0)に値を設定してください。
- ※3…タイマ RD ジェネラルレジスタ D0(TRDGRD0)のイニシャライズ時に設定する値は、タイマ RD ジェネラルレジスタ B0(TRDGRB0)と同じ値にしてください。

- ⑨タイマ RD ジェネラルレジスタ A1(TRDGRA1:Timer RD General register A1)、
タイマ RD ジェネラルレジスタ C1(TRDGRC1:Timer RD General register C1)の設定

タイマ RD ジェネラルレジスタ A1(TRDGRA1)に値を設定することによって、P2_4 端子から出力される PWM 波形の ON 幅を設定します。P2_6 端子からは、その反転した波形が出力されます。

P2_4 端子から出力される PWM 波形の ON 幅は、下記の式で決まります。

$$\text{P2}_4 \text{ 端子から出力される PWM 波形の ON 幅} = \text{タイマ RD カウンタ 0 のカウントソース} \times (\text{TRDGRA1} + 1)$$

TRDGRA1 を左辺に移動して、TRDGRA1 を求める式に変形します。

$$\text{TRDGRA1} = \text{P2}_4 \text{ 端子から出力される PWM 波形の ON 幅} / \text{タイマ RD カウンタ 0 のカウントソース} - 1$$

今回、タイマ RD カウンタ 0 のカウントソースは 400ns です。ON 幅を 10ms にするなら、タイマ RD ジェネラルレジスタ A1(TRDGRA1)は次のようになります。

$$\begin{aligned} \text{TRDGRA1} &= \text{ON 幅} / \text{カウントソース} - 1 \\ \text{TRDGRA1} &= (10 \times 10^{-3}) / (400 \times 10^{-9}) - 1 \\ \text{TRDGRA1} &= 25000 - 1 = 24999 \end{aligned}$$

■設定のポイント

- ※1…タイマ RD ジェネラルレジスタ A1(TRDGRA1)を使うときは、タイマ RD ジェネラルレジスタ C1(TRDGRC1)をバッファレジスタに設定して、ペアで使用してください。
- ※2…タイマ RD ジェネラルレジスタ A1(TRDGRA1)の設定は、イニシャライズ時に 1 回だけ行ってください。2 回目以降、ON 幅を変更したい場合は、タイマ RD ジェネラルレジスタ C1(TRDGRC1)に値を設定してください。
- ※3…タイマ RD ジェネラルレジスタ C1(TRDGRC1)のイニシャライズ時に設定する値は、タイマ RD ジェネラルレジスタ A1(TRDGRA1)と同じ値にしてください。

- ⑩タイマ RD ジェネラルレジスタ B1(TRDGRB1:Timer RD General register B1)、
タイマ RD ジェネラルレジスタ D1(TRDGRD1:Timer RD General register D1)の設定

タイマ RD ジェネラルレジスタ B1(TRDGRB1)に値を設定することによって、P2_5 端子から出力される PWM 波形の ON 幅を設定します。P2_7 端子からは、その反転した波形が出力されます。

P2_5 端子から出力される PWM 波形の ON 幅は、下記の式で決まります。

$$\text{P2}_5 \text{ 端子から出力される PWM 波形の ON 幅} = \text{タイマ RD カウンタ 0 のカウントソース} \times (\text{TRDGRB1} + 1)$$

TRDGRB1 を左辺に移動して、TRDGRB1 を求める式に変形します。

$$\text{TRDGRB1} = \text{P2}_5 \text{ 端子から出力される PWM 波形の ON 幅} / \text{タイマ RD カウンタ 0 のカウントソース} - 1$$

今回、タイマ RD カウンタ 0 のカウントソースは 400ns です。ON 幅を 15ms にするなら、タイマ RD ジェネラルレジスタ B1(TRDGRB1)は次のようになります。

$$\begin{aligned} \text{TRDGRB1} &= \text{ON 幅} / \text{カウントソース} - 1 \\ \text{TRDGRB1} &= (15 \times 10^{-3}) / (400 \times 10^{-9}) - 1 \\ \text{TRDGRB1} &= 37500 - 1 = 37499 \end{aligned}$$

■設定のポイント

- ※1…タイマ RD ジェネラルレジスタ B1(TRDGRB1)を使うときは、タイマ RD ジェネラルレジスタ D1(TRDGRD1)をバッファレジスタに設定して、ペアで使用してください。
- ※2…タイマ RD ジェネラルレジスタ B1(TRDGRB1)の設定は、イニシャライズ時に 1 回だけ行ってください。2 回目以降、ON 幅を変更したい場合は、タイマ RD ジェネラルレジスタ D1(TRDGRD1)に値を設定してください。
- ※3…タイマ RD ジェネラルレジスタ D1(TRDGRD1)のイニシャライズ時に設定する値は、タイマ RD ジェネラルレジスタ B1(TRDGRB1)と同じ値にしてください。

⑩タイマ RD スタートレジスタ(TRDSTR:Timer RD start register)の設定

TRD0 をカウントさせるか、停止させるか設定します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7~4		"0000"を設定	0000
bit3	TRD1 カウント動作選択ビット csel1_trdstr	0:TRDGRA1 レジスタとのコンペア一致でカウント停止 1:TRDGRA1 レジスタとのコンペア一致後もカウント継続 今回は"1"を設定します。	1
bit2	TRD0 カウント動作選択ビット csel0_trdstr	0:TRDGRA0 レジスタとのコンペア一致でカウント停止 1:TRDGRA0 レジスタとのコンペア一致後もカウント継続 今回は"1"を設定します。	1
bit1	TRD1 カウント開始フラグ(注 4) tstart1_trdstr	0:カウント停止(注 2) 1:カウント開始 設定した瞬間から、TRD1 のカウントが開始されます。リ セット同期 PWM モードでは TRD1 は使いませんので "0"を設定します。	0
bit0	TRD0 カウント開始フラグ(注 3) tstart0_trdstr	0:カウント停止(注 1) 1:カウント開始 設定した瞬間から、TRD0 のカウントが開始されます。 "1"を設定します。	1

注 1. bit2 が"1"に設定されているとき、bit0 へ"0"を書いてください。

注 2. bit3 が"1"に設定されているとき、bit1 へ"0"を書いてください。

注 3. bit2 が"0"でコンペア一致信号(TRDIOA0)が発生したとき、"0"(カウント停止)になります。

注 4. bit3 が"0"でコンペア一致信号(TRDIOA1)が発生したとき、"0"(カウント停止)になります。

タイマ RD スタートレジスタ(TRDSTR)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	1	1	0	1
16 進数	0				d			

21.6.2 main 関数

P2_2 端子に繋がっている LED などへ PWM 信号を出力します。PWM のデューティ比は、マイコンボードのディップスイッチで設定します。

```

34 : void main( void )
35 : {
36 :     init();                /* 初期化                */
37 :
38 :     while( 1 ) {
39 :         trdgrd0 = 39998 * dipsw_get() / 15;
40 :     }
41 : }

```

マイコンボードのディップスイッチの値に応じて、タイマ RD ジェネラルレジスタ D0(TRDGRD0)に値を設定しています。ディップスイッチとデューティ比の関係を、下表に示します。

ディップスイッチの値 (dipsw_get 関数)	TRDGRD0 の値 (A)	TRD0 と一致する値 (A)+1	デューティ比 ((A)+1) / 40000
0	$39998 * 0 / 15 = 0$	1	0.0025%
1	$39998 * 1 / 15 = 2666$	2667	6.6675%
2	$39998 * 2 / 15 = 5333$	5334	13.3350%
3	$39998 * 3 / 15 = 7999$	8000	20.0000%
4	$39998 * 4 / 15 = 10666$	10667	26.6675%
5	$39998 * 5 / 15 = 13332$	13333	33.3325%
6	$39998 * 6 / 15 = 15999$	16000	40.0000%
7	$39998 * 7 / 15 = 18665$	18666	46.6650%
8	$39998 * 8 / 15 = 21332$	21333	53.3325%
9	$39998 * 9 / 15 = 23998$	23999	59.9975%
10	$39998 * 10 / 15 = 26665$	26666	66.6650%
11	$39998 * 11 / 15 = 29331$	29332	73.3300%
12	$39998 * 12 / 15 = 31998$	31999	79.9975%
13	$39998 * 13 / 15 = 34664$	34665	86.6625%
14	$39998 * 14 / 15 = 37331$	37332	93.3300%
15	$39998 * 15 / 15 = 39998$	39999	99.9975%

39 行目だけで 0%~100%まで対応させる、簡単な計算式です。ただし、0%は完全な 0%ではなく、400ns だけ"1"になります。100%も完全な 100%ではなく、400ns だけ"0"になります。

完全な 0%、100%にも対応するようには、下記プログラムのように 0%、100%、それ以外で場合分けします。

```
void main( void )
{
    init();                /* 初期化                */

    while( 1 ) {
        if( dipsw_get() == 0 ) {
            // 0%
            trdgrd0 = trdgra0;
        } else if( dipsw_get() == 15 ) {
            // 100%
            trdgrd0 = trdgra0 + 1;
        } else {
            // 0%、100%以外
            trdgrd0 = 39999 * dipsw_get() / 15;
        }
    }
}
```

ディップスイッチとデューティ比の関係を、下表に示します。

ディップスイッチの値 (dipsw_get 関数)	TRDGRD0 の値 (A)	TRD0 と一致する値 (A)+1	デューティ比 ((A)+1)/40000
0	39999 を直接指定	40000 (“0”と“1”が同時に起こり、 “0”が優先される)	0%
1	$39999 * 1 / 15 = 2666$	2667	6.6675%
2	$39999 * 2 / 15 = 5333$	5334	13.3350%
3	$39999 * 3 / 15 = 7999$	8000	20.0000%
4	$39999 * 4 / 15 = 10666$	10667	26.6675%
5	$39999 * 5 / 15 = 13332$	13333	33.3325%
6	$39999 * 6 / 15 = 15999$	16000	40.0000%
7	$39999 * 7 / 15 = 18665$	18666	46.6650%
8	$39999 * 8 / 15 = 21332$	21333	53.3325%
9	$39999 * 9 / 15 = 23998$	23999	59.9975%
10	$39999 * 10 / 15 = 26665$	26666	66.6650%
11	$39999 * 11 / 15 = 29331$	29332	73.3300%
12	$39999 * 12 / 15 = 31998$	31999	79.9975%
13	$39999 * 13 / 15 = 34664$	34665	86.6625%
14	$39999 * 14 / 15 = 37331$	37332	93.3300%
15	40000 を直接指定	一致せず	100%

21.7 演習

- (1) 周期が出力される端子(P2_1)の PWM 出力を禁止して、通常の I/O ポートにしてください。このとき、この端子は出力端子として"0"を出力してください。
- (2) 逆相の端子(P2_3、P2_6、P2_7 の 3 つとも)の PWM 出力を禁止して、通常の I/O ポートにしてください。このとき、この端子は出力端子として"0"を出力してください。
※プログラムは、(1)の状態から改造するものとする。
- (3) P2_4 端子も P2_2 端子と同様に、ディップスイッチに合わせて PWM 信号が出力されるようにしてください。出力の仕方は、サンプルプログラム 39 行目の「39998 * dipsw_get() / 15」を使用することとする。
※プログラムは、(2)の状態から改造するものとする。
- (4) P2_5 端子も P2_2 端子と同様に、ディップスイッチに合わせて PWM 信号が出力されるようにしてください。出力の仕方は、サンプルプログラム 39 行目の「39998 * dipsw_get() / 15」を使用することとする。
※プログラムは、(3)の状態から改造するものとする。
- (5) P2_4 端子、P2_5 端子の PWM 出力を禁止して、通常の I/O ポートにしてください。このとき、この端子は出力端子として"0"を出力してください。
※プログラムは、(4)の状態から改造するものとする。

22. タイマ RD による PWM 波形出力(PWM モード) (プロジェクト:timer_rd_pwm6)

22.1 概要

本章では、PWM 波形を出力する方法を紹介します。今回は、タイマ RD を PWM モードで使用して、同じ周期の PWM 波形を 6 本出力します。タイマ RD の初期設定後は、プログラムが関与しなくても PWM 波形を出力し続けます。プログラムでは、PWM 波形出力処理以外の処理をすることができます。ただし、リセット同期 PWM モードで設定していたバッファレジスタは PWM モードでは使えないため、PWM の ON 幅を変更するときはプログラムでタイミングを計ります。この方法も解説します。

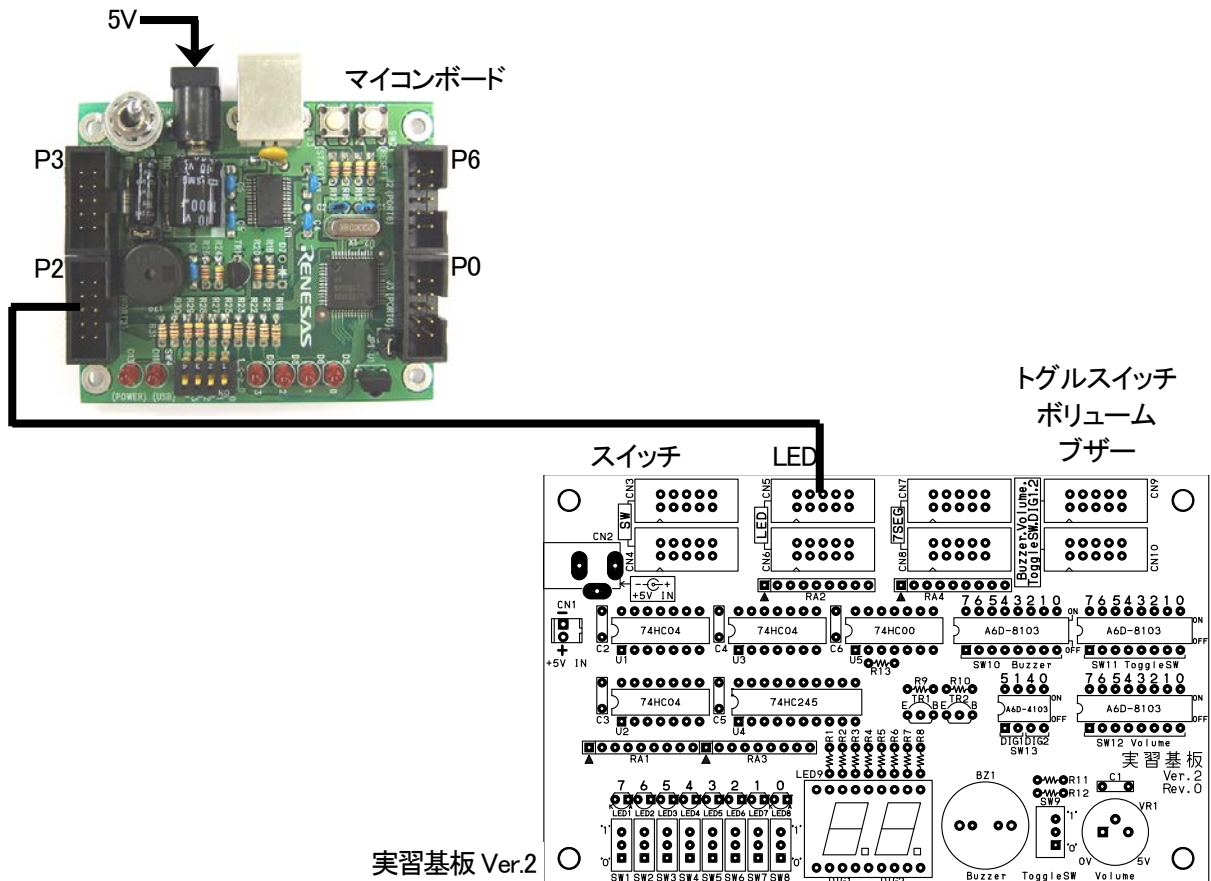
22.2 接続

■使用ポート

マイコンのポート	接続内容
P5_7、P4_5、P4_4、P4_3	マイコンボード上のディップスイッチです。
P2 (J7)	実習基板 Ver.2 の LED 部など、出力機器を接続します。

■接続例

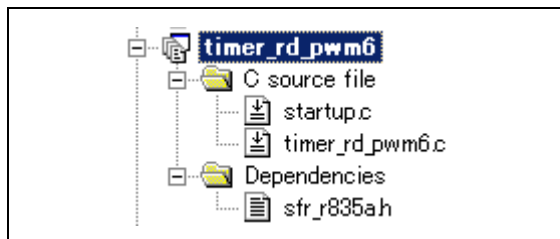
実習基板 Ver.2 を使ったときの接続例を下記に示します。



■操作方法

マイコンボードのディップスイッチ(SW4)の値 0~15 によって、LED の点灯する明るさが変わります。このとき、どの LED が明るくなるか観察してください。

22.3 プロジェクトの構成



	ファイル名	内容
1	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。このファイルは共通で、どのプロジェクトもこのファイルから実行されます。
2	timer_rd_pwm6.c	実際に制御するプログラムが書かれています。R8C/35A の内蔵周辺機能(SFR)の初期化も行います。
3	sfr_r835a.h	R8C/35A マイコンの内蔵周辺機能を制御するためのレジスタ(Special Function Registers)を定義したファイルです。

22.4 プログラム「timer_rd_pwm6.c」

```

1 : /******
2 : /* 対象マイコン R8C/35A */
3 : /* ファイル内容 タイマRDのPWMモードを使ってPWM6本出力 */
4 : /* バージョン Ver. 1. 20 */
5 : /* Date 2010. 04. 19 */
6 : /* Copyright ルネサスマイコンカーラー事務局 */
7 : /* 日立インターメディックス株式会社 */
8 : /******
9 : */
10 : 入力 : マイコンボードのディップスイッチ(4bit)
11 : 出力 : P2_1, P2_2, P2_3, P2_5, P2_6, P2_7端子からPWM出力
12 :
13 : マイコンボードのディップスイッチ(4bit)から入力した割合により、
14 : ポート2の6端子からPWM波形を出力します。
15 : */
16 :
17 : /*=====*/
18 : /* インクルード */
19 : /*=====*/
20 : #include "sfr_r835a.h" /* R8C/35A SFRの定義ファイル */
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 : unsigned char dipsw_get( void );
31 :
32 : /******
33 : /* メインプログラム */
34 : /******
35 : void main( void )
36 : {
37 :     unsigned int pwm;
38 :
    
```

22. タイマ RD による PWM 波形出力(PWM モード) (プロジェクト:timer_rd_pwm6)

```

39 :     init();                               /* 初期化                               */
40 :
41 :     while( 1 ) {
42 :         if( dipsw_get() == 0 ) {
43 :             // trdgra0, trdgralと同じにすると0%出力
44 :             pwm = 39999;
45 :         } else if( dipsw_get() == 15 ) {
46 :             // trdgra0, trdgralより大きくすると100%出力
47 :             pwm = 40000;
48 :         } else {
49 :             // 1~14なら割合に応じてPWM出力する
50 :             pwm = (long)39999 * dipsw_get() / 15;
51 :         }
52 :
53 :         trdgrb0 = pwm;                       /* P2_2のON幅設定                       */
54 :         trdgrc0 = pwm;                       /* P2_1のON幅設定                       */
55 :         trdgrd0 = pwm;                       /* P2_3のON幅設定                       */
56 :         trdgrb1 = pwm;                       /* P2_5のON幅設定                       */
57 :         trdgrc1 = pwm;                       /* P2_6のON幅設定                       */
58 :         trdgrd1 = pwm;                       /* P2_7のON幅設定                       */
59 :     }
60 : }
61 :
62 : /*****
63 : /* R8C/35A スペシャルファンクションレジスタ(SFR)の初期化
64 : /*****
65 : void init( void )
66 : {
67 :     int i;
68 :
69 :     /* クロックをXINクロック(20MHz)に変更 */
70 :     prc0 = 1;                               /* プロテクト解除                       */
71 :     cm13 = 1;                               /* P4_6, P4_7をXIN-XOUT端子にする*/
72 :     cm05 = 0;                               /* XINクロック発振                       */
73 :     for(i=0; i<50; i++ );                  /* 安定するまで少し待つ(約10ms) */
74 :     ocd2 = 0;                               /* システムクロックをXINにする */
75 :     prc0 = 0;                               /* プロテクトON                         */
76 :
77 :     /* ポートの入出力設定 */
78 :     prc2 = 1;                               /* PD0のプロテクト解除                 */
79 :     pd0 = 0x00;                             /* スイッチなど入力                     */
80 :     p1 = 0x0f;                              /* 3-0:LEDは消灯                       */
81 :     pd1 = 0xdf;                             /* 5:RXD0 4:TXD0 3-0:LED               */
82 :     pd2 = 0xfe;                             /* 0:PushSW                             */
83 :     pd3 = 0xfb;                             /* 4:Buzzer 2:IR                       */
84 :     pd4 = 0x80;                             /* 7:XOUT 6:XIN 5-3:DIP SW 2:VREF*/
85 :     pd5 = 0x40;                             /* 7:DIP SW                             */
86 :     pd6 = 0xff;                             /* LEDなど出力                         */
87 :
88 :     /* タイマRD PWMモードの設定(6本のPWM出力) */
89 :     trdpmr = 0x77;                          /* 各端子PWMモードにするかしないか*/
90 :     trdfcr = 0x80;                          /* PWMモードに設定                     */
91 :     trdoer1 = 0x11;                         /* 各端子出力許可するかしないか */
92 :     trdpsr0 = 0x68;                         /* TRDIOB0, C0, D0端子設定             */
93 :     trdpsr1 = 0x54;                         /* TRDIOB1, C1, D1端子設定             */
94 :     trdocr = 0x00;                          /* 各端子初期出力設定                 */
95 :     trdcr0 = 0x23;                          /* TRD0ソースカウンタの選択:f8        */
96 :     trdcr1 = 0x23;                          /* TRD1ソースカウンタの選択:f8        */
97 :     trdpocr0 = 0x00;                        /* B0~D0のアクティブレベル設定        */
98 :     trdpocr1 = 0x00;                        /* B1~D1のアクティブレベル設定        */
99 :     trdgra0 = 39999;                        /* TRD0の周期設定                     */
100 :    trdgral = 39999;                         /* TRD1の周期設定                     */
101 :    trdgrb0 = 0;                             /* P2_2のON幅設定                     */
102 :    trdgrc0 = 0;                             /* P2_1のON幅設定                     */
103 :    trdgrd0 = 0;                             /* P2_3のON幅設定                     */
104 :    trdgrb1 = 0;                             /* P2_5のON幅設定                     */
105 :    trdgrc1 = 0;                             /* P2_6のON幅設定                     */
106 :    trdgrd1 = 0;                             /* P2_7のON幅設定                     */
107 :    trdsta = 0x0f;                           /* TRD0, TRD1カウンタ開始             */
108 : }
109 :
110 : /*****
111 : /* デイプスイッチ値読み込み
112 : /* 戻り値 スイッチ値 0~15
113 : /*****
114 : unsigned char dipsw_get( void )
115 : {
116 :     unsigned char sw, sw1, sw2;
117 :
118 :     sw1 = (p5>>4) & 0x08;                  /* デイプスイッチ読み込み3           */
119 :     sw2 = (p4>>3) & 0x07;                  /* デイプスイッチ読み込み2,1,0*/
120 :     sw = sw1 | sw2;                         /* P5とP4の値を合わせる             */
121 :
122 :     return sw;
123 : }
124 :
125 : /*****
126 : /* end of file
127 : /*****

```

22.5 プログラムの解説

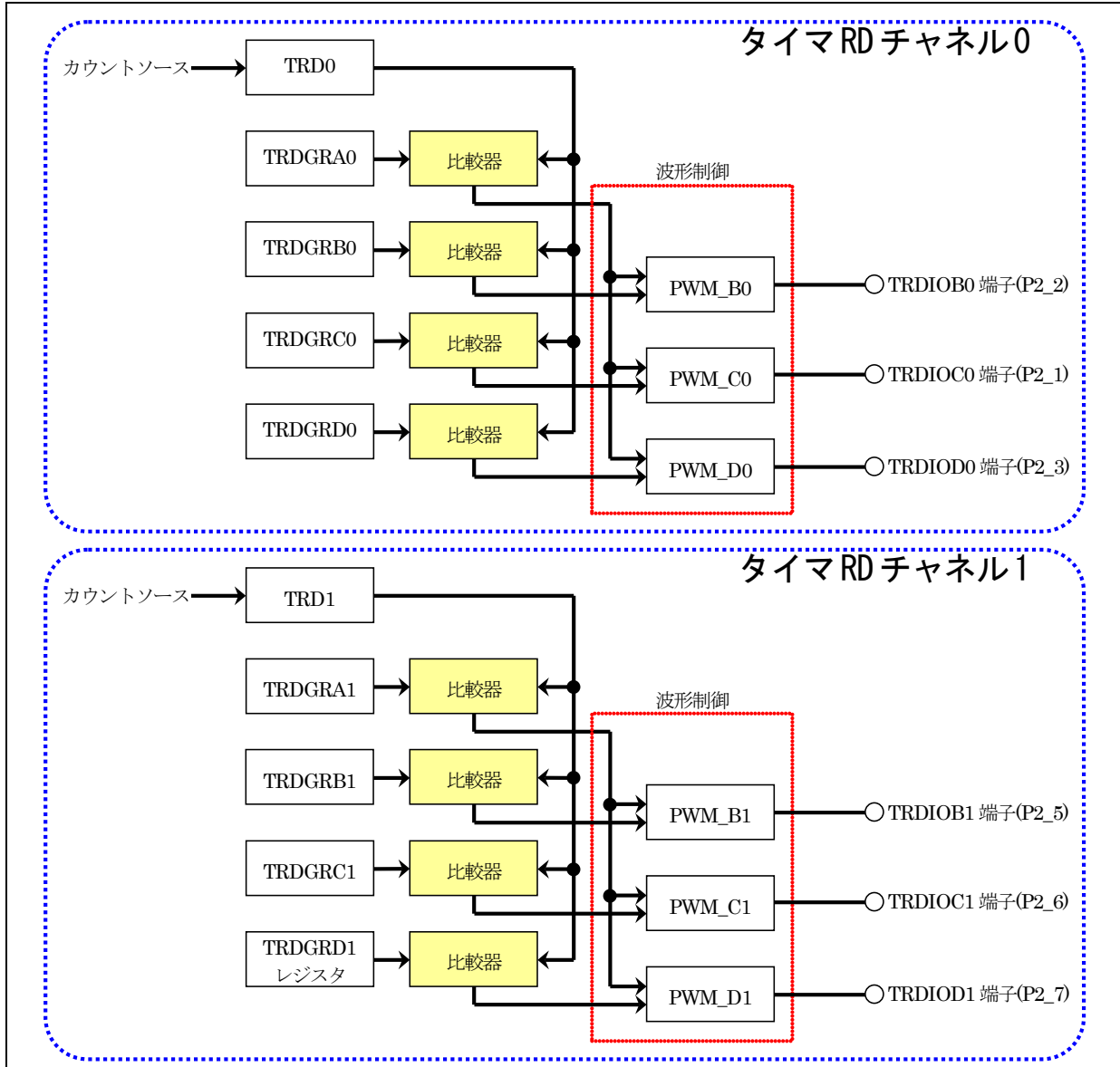
22.5.1 init 関数(タイマ RD の設定)

タイマ RD を使った PWM モードの設定を行います。

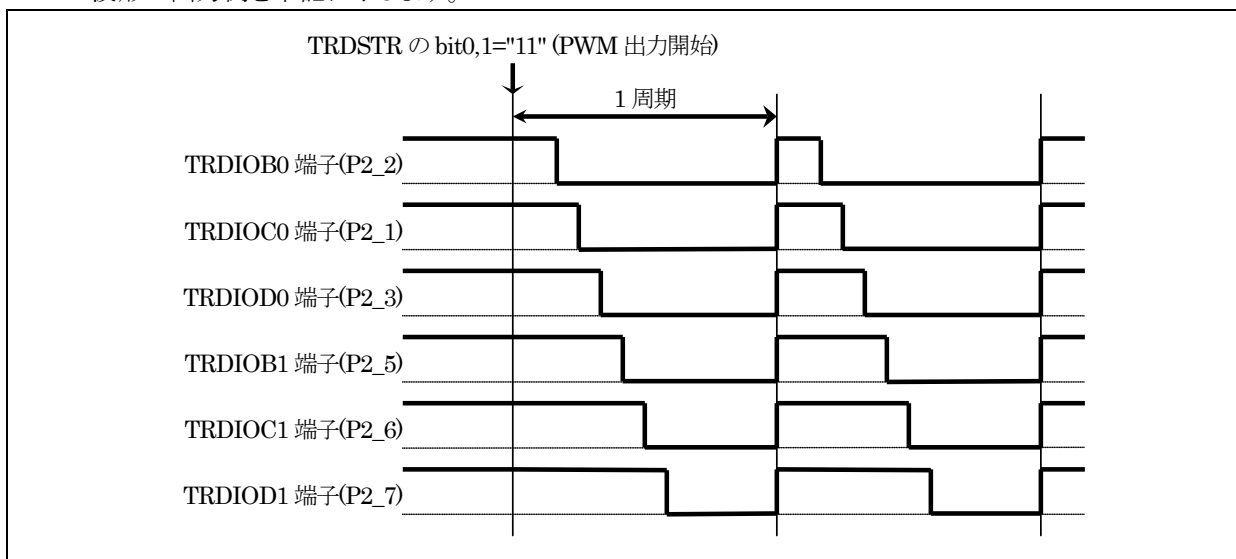
```
88 :      /* タイマRD PWMモードの設定(6本のPWM出力) */
89 :      trdpmr   = 0x77;          /* 各端子PWMモードにするかしないか*/
90 :      trdfcr   = 0x80;          /* PWMモードに設定 */
91 :      trdoerl  = 0x11;          /* 各端子出力許可するかしないか */
92 :      trdpsr0  = 0x68;          /* TRDIOB0, C0, D0端子設定 */
93 :      trdpsr1  = 0x54;          /* TRDIOB1, C1, D1端子設定 */
94 :      trdocr   = 0x00;          /* 各端子初期出力設定 */
95 :      trdcr0   = 0x23;          /* TRD0ソースカウンタの選択:f8 */
96 :      trdcr1   = 0x23;          /* TRD1ソースカウンタの選択:f8 */
97 :      trdpocr0 = 0x00;          /* B0~D0のアクティブレベル設定 */
98 :      trdpocr1 = 0x00;          /* B1~D1のアクティブレベル設定 */
99 :      trdgra0  = 39999;         /* TRD0の周期設定 */
100 :      trdgra1  = 39999;         /* TRD1の周期設定 */
101 :      trdgrb0  = 0;            /* P2_2のON幅設定 */
102 :      trdgrc0  = 0;            /* P2_1のON幅設定 */
103 :      trdgrd0  = 0;            /* P2_3のON幅設定 */
104 :      trdgrb1  = 0;            /* P2_5のON幅設定 */
105 :      trdgrc1  = 0;            /* P2_6のON幅設定 */
106 :      trdgrd1  = 0;            /* P2_7のON幅設定 */
107 :      trdstr   = 0x0f;          /* TRD0, TRD1カウンタ開始 */
```

(1) タイマ RD のブロック図

PWM モードのブロック図を下記に示します。

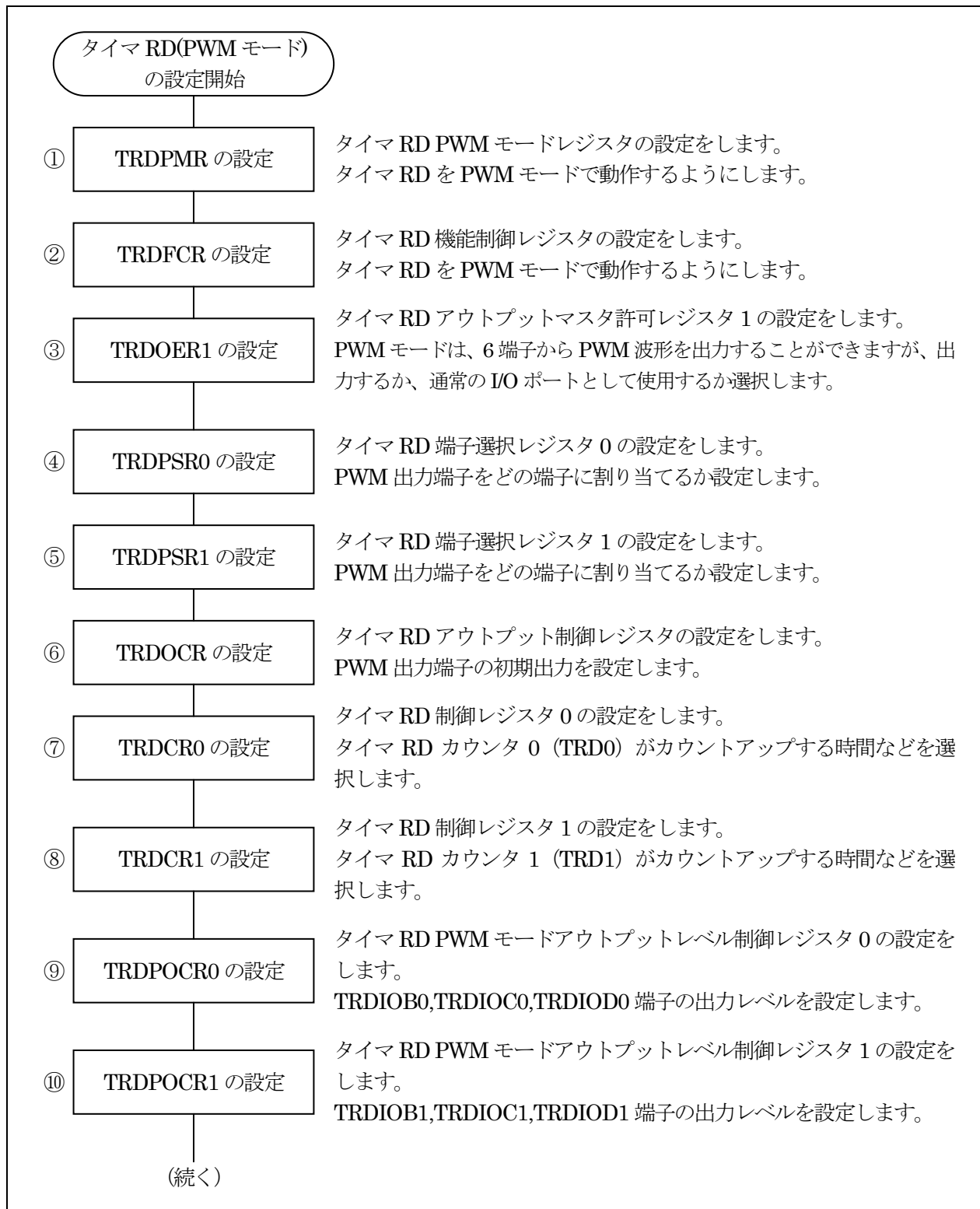


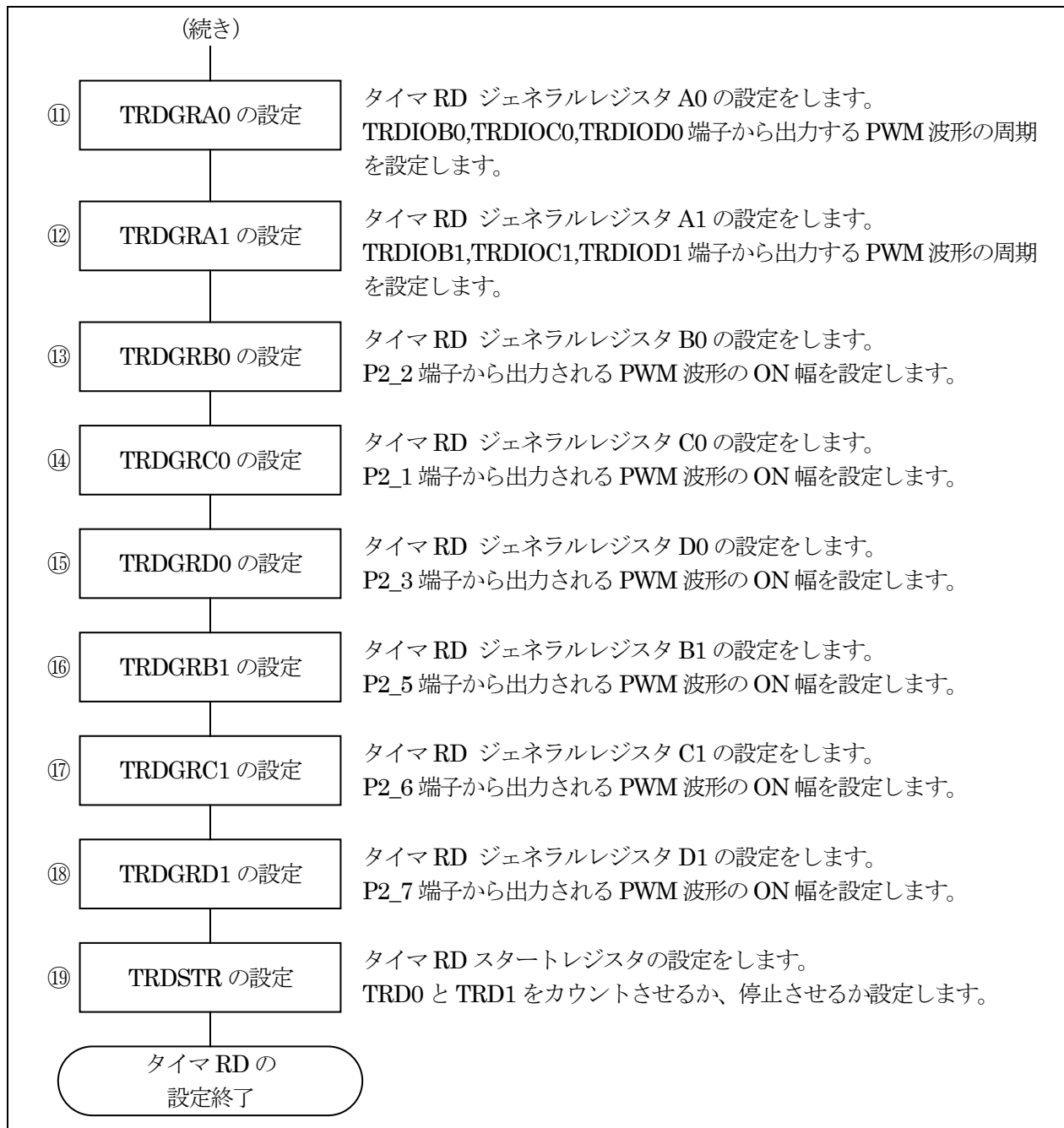
PWM 波形の出力例を下記に示します。



(2) タイマ RD の設定 (PWM モード)

今回は、タイマ RD を PWM モードで使用して、PWM 波形を 6 本出力します。レジスタの設定手順を下記に示します。





①タイマ RD PWM モードレジスタ(TRDPMR:Timer RD PWM mode register)の設定

タイマ RD を PWM モードで動作するようにします。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7		"0"を設定	0
bit6	TRDIOD1 PWM モード選択 ビット pwmd1_trdpmr	0:タイマモード 1:PWM モード TRDIOD1 を PWM として使うか、タイマとして使うかを 設定します。PWM モードを設定します。	1
bit5	TRDIOC1 PWM モード選択 ビット pwmc1_trdpmr	0:タイマモード 1:PWM モード TRDIOC1 を PWM として使うか、タイマとして使うかを 設定します。PWM モードを設定します。	1
bit4	TRDIOB1 PWMモード選択ビ ット pwmb1_trdpmr	0:タイマモード 1:PWM モード TRDIOB1をPWMとして使うか、タイマとして使うかを設 定します。PWM モードを設定します。	1
bit3		"0"を設定	0
bit2	TRDIOD0 PWM モード選択 ビット pwmd0_trdpmr	0:タイマモード 1:PWM モード TRDIOD0 を PWM として使うか、タイマとして使うかを 設定します。PWM モードを設定します。	1
bit1	TRDIOC0 PWM モード選択 ビット pwmc0_trdpmr	0:タイマモード 1:PWM モード TRDIOC0 を PWM として使うか、タイマとして使うかを 設定します。PWM モードを設定します。	1
bit0	TRDIOB0 PWMモード選択ビ ット pwmb0_trdpmr	0:タイマモード 1:PWM モード TRDIOB0をPWMとして使うか、タイマとして使うかを設 定します。PWM モードを設定します。	1

タイマ RD PWM モードレジスタ (TRDPMR)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	1	1	1	0	1	1	1
16 進数	7				7			

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

タイマ RD を PWM モードで動作するようにします。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7	PWM3 モード選択ビット pwm3_trdfer	"1"を設定 ※"1"は PWM3 モードが無効になる設定です。	1
bit6	外部クロック入力選択ビット stclk_trdfer	0:外部クロック入力無効 1:外部クロック入力有効 今回は、内部のクロックを使用しますので、無効を選択します。	0
bit5~2		"0000"を設定	0000
bit1,0	コンビネーションモード選択 ビット bit1:cmd1_trdfer bit0:cmd0_trdfer	PWM モードでは"00"(タイマモード、PWM モード、 PWM3 モード)にしてください	00

タイマ RD 機能制御レジスタ (TRDFCR)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	1	0	0	0	0	0	0	0
16 進数	8				0			

③タイマ RD アウトプットマスタ許可レジスタ 1(TRDOER1:Timer RD output master enable register 1)の設定

PWM モードは 6 端子から PWM 波形を出力することができますが、出力するか、通常の I/O ポートとして使用するかを選択します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7	TRDIOD1(P2_7)出力禁止ビット ed1_trdoer1	0:出力許可 1:出力禁止(TRDIOD1 端子はプログラマブル入出力ポート) 今回は、出力を許可します。	0
bit6	TRDIOC1(P2_6)出力禁止ビット ec1_trdoer1	0:出力許可 1:出力禁止(TRDIOC1 端子はプログラマブル入出力ポート) 今回は、出力を許可します。	0
bit5	TRDIOB1(P2_5)出力禁止ビット eb1_trdoer1	0:出力許可 1:出力禁止(TRDIOB1 端子はプログラマブル入出力ポート) 今回は、出力を許可します。	0
bit4	TRDIOA1 出力禁止ビット ea1_trdoer1	PWM モード時、“1”に設定	1
bit3	TRDIOD0(P2_3)出力禁止ビット ed0_trdoer1	0:出力許可 1:出力禁止(TRDIOD0 端子はプログラマブル入出力ポート) 今回は、出力を許可します。	0
bit2	TRDIOC0(P2_1)出力禁止ビット ec0_trdoer1	0:出力許可 1:出力禁止(TRDIOC0 端子はプログラマブル入出力ポート) 今回は、出力を許可します。	0
bit1	TRDIOB0(P2_2)出力禁止ビット eb0_trdoer1	0:出力許可 1:出力禁止(TRDIOB0 端子はプログラマブル入出力ポート) 今回は、出力を許可します。	0
bit0	TRDIOA0 出力禁止ビット ea0_trdoer1	PWM モード時、“1”に設定	1

タイマ RD アウトプットマスタ許可レジスタ 1(TRDOER1)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	1	0	0	0	1
16 進数	1				1			

④タイマ RD 端子選択レジスタ 0(TRDPSR0:Timer RD function select register 0)の設定

PWM 波形の出力端子をどのポートに割り当てるか設定します。タイマ RD 端子選択レジスタ 0(TRDPSR0)では、TRDIOD0 端子、TRDIOC0 端子、TRDIOB0 端子の割り当てを設定します。

R8C/35A では、割り当てる端子は決まっております端子を変更することができません。PWM 端子として割り当てるか、割り当てないか(通常の I/O ポートとして使用するか)を設定します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7		"0"を設定	0
bit6	TRDIOD0 端子選択ビット trdiiod0sel0	0:TRDIOD0 端子は使用しない 1:P2.3 に割り当てる 今回は、P2.3 に割り当てて、この端子から PWM 波形を出力します。	1
bit5,4	TRDIOC0 端子選択ビット bit5:trdioc0sel1 bit4:trdioc0sel0	00:TRDIOC0 端子は使用しない 01:設定しないでください 10:P2.1 に割り当てる 11:設定しないでください 今回は、P2.1 に割り当てて、この端子から PWM 波形を出力します。	10
bit3,2	TRDIOB0 端子選択ビット bit3:trdiob0sel1 bit2:trdiob0sel0	00:TRDIOB0 端子は使用しない 01:設定しないでください 10:P2.2 に割り当てる 11:設定しないでください 今回は、P2.2 に割り当てて、この端子から PWM 波形を出力します。	10
bit1,0		"00"を設定	00

タイマ RD 端子選択レジスタ 0(TRDPSR0)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	1	1	0	1	0	0	0
16 進数	6				8			

⑤タイマ RD 端子選択レジスタ 1(TRDPSR1:Timer RD function select register 1)の設定

PWM 波形の出力端子をどのポートに割り当てるか設定します。タイマ RD 端子選択レジスタ 1(TRDPSR1)では、TRDIOD1 端子、TRDIOC1 端子、TRDIOB1 端子、TRDIOA1 端子の割り当てを設定します。

R8C/35A では、割り当てる端子は決まっております端子を変更することができません。PWM 端子として割り当てるか、割り当てないか(通常の I/O ポートとして使用するか)を設定します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7		"0"を設定	0
bit6	TRDIOD1 端子選択ビット trdioid1sel0	0:TRDIOD1 端子は使用しない 1:P2_7 に割り当てる 今回は、P2_7 に割り当てて、この端子から PWM 波形を出力します。	1
bit5		0 を設定	0
bit4	TRDIOC1 端子選択ビット trdioc1sel0	0:TRDIOC1 端子は使用しない 1:P2_6 に割り当てる 今回は、P2_6 に割り当てて、この端子から PWM 波形を出力します。	1
bit3		"0"を設定	0
bit2	TRDIOB1 端子選択ビット trdiob1sel0	0:TRDIOB1 端子は使用しない 1:P2_5 に割り当てる 今回は、P2_5 に割り当てて、この端子から PWM 波形を出力します。	1
bit1		"0"を設定	0
bit0	TRDIOA1 端子選択ビット trdioa1sel0	PWM モードでは"0"を設定	0

タイマ RD 端子選択レジスタ 1(TRDPSR1)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	1	0	1	0	1	0	0
16 進数	5				4			

⑥タイマ RD アウトプット制御レジスタ 0(TRDOCR:Timer RD output control register)の設定

PWM 出力端子の初期出力を設定します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7	TRDIOD1 初期出力レベル選 択ビット(注 1) tod1_trdoctr	0:アクティブでないレベル 1:アクティブレベル 初期出力はアクティブでないレベルにします。	0
bit6	TRDIOC1 初期出力レベル選 択ビット(注 1) toc1_trdoctr	0:アクティブでないレベル 1:アクティブレベル 初期出力はアクティブでないレベルにします。	0
bit5	TRDIOB1 初期出力レベル選 択ビット(注 1) tob1_trdoctr	0:アクティブでないレベル 1:アクティブレベル 初期出力はアクティブでないレベルにします。	0
bit4	TRDIOA1 初期出力レベル選 択ビット(注 1) toa1_trdoctr	PWM モードでは"0"を設定	0
bit3	TRDIOD0 初期出力レベル選 択ビット(注 1) tod0_trdoctr	0:アクティブでないレベル 1:アクティブレベル 初期出力はアクティブでないレベルにします。	0
bit2	TRDIOC0 初期出力レベル選 択ビット(注 1) toc0_trdoctr	0:アクティブでないレベル 1:アクティブレベル 初期出力はアクティブでないレベルにします。	0
bit1	TRDIOB0 初期出力レベル選 択ビット(注 1) tob0_trdoctr	0:アクティブでないレベル 1:アクティブレベル 初期出力はアクティブでないレベルにします。	0
bit0	TRDIOA0 初期出力レベル選 択ビット(注 1) toa0_trdoctr	PWM モードでは"0"を設定	0

注 1. 端子の機能が波形出力の場合(「ハードウェアマニュアル 7.5 ポートの設定」参照)、TRDOCR レジスタを設定したとき、初期出力レベルが出力されます。

タイマ RD 制御レジスタ 0(TRDOCR)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	0	0	0	0
16 進数	0				0			

⑦タイマ RD 制御レジスタ 0(TRDCR0:Timer RD control register 0)の設定

タイマ RD カウンタ 0(TRD0)がカウントアップする時間などを選択します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7~5	TRD0 カウンタクリア選択ビット bit7:cclr2_trdcr0 bit6:cclr1_trdcr0 bit5:cclr0_trdcr0	PWM モードの場合は、“001”(TRDGRA0 とのコンペア一致で TRD0 レジスタクリア)に設定	001
bit4,3	外部クロックエッジ選択ビット (注 3) bit4:ckeg1_trdcr0 bit3:ckeg0_trdcr0	00:立ち上がりエッジでカウント 01:立ち下がりエッジでカウント 10:両エッジでカウント 11:設定しないでください 外部クロックは使いませんので何を設定しても変化ありません。今回は“00”を設定します。	00
bit2~0	カウントソース選択ビット bit2:tck2_trdcr0 bit1:tck1_trdcr0 bit0:tck0_trdcr0	000:f1 (1/20MHz=50ns) 001:f2 (2/20MHz=100ns) 010:f4 (4/20MHz=200ns) 011:f8 (8/20MHz=400ns) 100:f32 (32/20MHz=1600ns) 101:TRDCLK 入力(注 1)または fC2 (注 2) fC2 = 2/XCIN クロック=今回は未接続 110:fOCO40M (高速オンチップオシレータ 40MHz= 今回は未接続) 111:fOCO-F(注 4) (高速オンチップオシレータを FRA2 で分周したクロック=今回は未接続) タイマ RD カウンタ 0(TRD0)がカウントアップする時間を 設定します。今回は“011”を設定します。TRD0 は、 400ns ごとに+1 していきます。	011

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

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

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

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

タイマ RD 制御レジスタ 0(TRDCR0)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	1	0	0	0	1	1
16 進数	2				3			

⑧タイマ RD 制御レジスタ 1(TRDCR1:Timer RD control register 1)の設定

タイマ RD カウンタ 1(TRD1)がカウントアップする時間などを選択します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7~5	TRD1 カウンタクリア選択ビット bit7:cclr2_trdcr1 bit6:cclr1_trdcr1 bit5:cclr0_trdcr1	PWM モードの場合は、“001”(TRDGRA1 とのコンペア一致で TRD1 レジスタクリア)に設定	001
bit4,3	外部クロックエッジ選択ビット (注 3) bit4:ckeg1_trdcr1 bit3:ckeg0_trdcr1	00:立ち上がりエッジでカウント 01:立ち下がりエッジでカウント 10:両エッジでカウント 11:設定しないでください 外部クロックは使いませんので何を設定しても変化ありません。今回は“00”を設定します。	00
bit2~0	カウントソース選択ビット bit2:tck2_trdcr1 bit1:tck1_trdcr1 bit0:tck0_trdcr1	000:f1 (1/20MHz=50ns) 001:f2 (2/20MHz=100ns) 010:f4 (4/20MHz=200ns) 011:f8 (8/20MHz=400ns) 100:f32 (32/20MHz=1600ns) 101:TRDCLK 入力(注 1)または fC2 (注 2) fC2 = 2/XCIN クロック=今回は未接続 110:fOCO40M (高速オンチップオシレータ 40MHz= 今回は未接続) 111:fOCO-F(注 4) (高速オンチップオシレータを FRA2 で分周したクロック=今回は未接続) タイマ RD カウンタ 1(TRD1)がカウントアップする時間を 設定します。今回は“011”を設定します。TRD1 は、 400ns ごとに+1 していきます。	011

注 1. TRDECR レジスタの ITCLK1 ビットが“0”(TRDCLK 入力)かつ TRDFCR レジスタの STCLK ビットが“1”(外部クロック入力有効)のとき、有効です。

注 2. タイマモードで、TRDECR レジスタの ITCLK1 ビットが“1”(fC2)のとき有効です。

注 3. TCK2~TCK0 ビットが“101”(TRDCLK 入力または fC2)、TRDECR レジスタの ITCLK1 ビットが“0”(TRDCLK 入力)、かつ TRDFCR レジスタの STCLK ビットが“1”(外部クロック入力有効)のとき、有効です。

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

タイマ RD 制御レジスタ 1(TRDCR1)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	1	0	0	0	1	1
16 進数	2				3			

⑨タイマ RD PWM モードアウトプットレベル制御レジスタ 0

(TRDPOCR0: Timer RD PWM mode output level control register 0)の設定

タイマ RD PWM モードアウトプットレベル制御レジスタ 0(TRDPOCR0)の設定をします。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7~3		"00000"を設定	00000
bit2	PWM モードアウトプットレベル制御ビット D pold_trdpocr0	0:TRDIOD0 の出力レベルは“L”アクティブ 1:TRDIOD0 の出力レベルは“H”アクティブ 出力レベルは“L”アクティブに設定	0
bit1	PWM モードアウトプットレベル制御ビット C polc_trdpocr0	0:TRDIOC0 の出力レベルは“L”アクティブ 1:TRDIOC0 の出力レベルは“H”アクティブ 出力レベルは“L”アクティブに設定	0
bit0	PWM モードアウトプットレベル制御ビット B polb_trdpocr0	0:TRDIOB0 の出力レベルは“L”アクティブ 1:TRDIOB0 の出力レベルは“H”アクティブ 出力レベルは“L”アクティブに設定	0

※出力レベルと出力波形の関係

出力レベルによって、下記のように出力波形のレベルが変わります。

出力 レベル	波形	説明
“H”アク ティブ		“0”からスタートする設定です。
“L”アク ティブ		“1”からスタートする設定です。

タイマ RD PWM モードアウトプットレベル制御レジスタ 0(TRDPOCR0)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	0	0	0	0
16 進数	0				0			

⑩タイマ RD PWM モードアウトプットレベル制御レジスタ 1

(TRDPOCR1: Timer RD PWM mode output level control register 1)の設定

タイマ RD PWM モードアウトプットレベル制御レジスタ 1(TRDPOCR1)の設定をします。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7~3		"00000"を設定	00000
bit2	PWM モードアウトプットレベル制御ビット D pold_trdpocr1	0:TRDIOD1 の出力レベルは“L”アクティブ 1:TRDIOD1 の出力レベルは“H”アクティブ 出力レベルは“L”アクティブに設定	0
bit1	PWM モードアウトプットレベル制御ビット C polc_trdpocr1	0:TRDIOD1 の出力レベルは“L”アクティブ 1:TRDIOD1 の出力レベルは“H”アクティブ 出力レベルは“L”アクティブに設定	0
bit0	PWM モードアウトプットレベル制御ビット B polb_trdpocr1	0:TRDIOD1 の出力レベルは“L”アクティブ 1:TRDIOD1 の出力レベルは“H”アクティブ 出力レベルは“L”アクティブに設定	0

※出力レベルと出力波形の関係

出力レベルによって、下記のように出力波形のレベルが変わります。

出力 レベル	波形	説明
“H”アク ティブ		“0”からスタートする設定です。
“L”アク ティブ		“1”からスタートする設定です。

タイマ RD PWM モードアウトプットレベル制御レジスタ 1(TRDPOCR1)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	0	0	0	0
16 進数	0				0			

⑪タイマ RD ジェネラルレジスタ A0(TRDGRA0:Timer RD General register A0)の設定

タイマ RD ジェネラルレジスタ A0(TRDGRA0)に値を設定することによって、TRDIOB0(P2_2) 端子,TRDIOC0(P2_1)端子,TRDIOD0(P2_3)端子から出力する PWM 波形の周期を設定します。

PWM 周期は、下記の式で決まります。

$$\text{PWM 周期} = \text{タイマ RD カウンタ 0 のカウントソース} \times (\text{TRDGRA0} + 1)$$

TRDGRA0 を左辺に移動して、TRDGRA0 を求める式に変形します。

$$\text{TRDGRA0} = \text{PWM 周期} / \text{タイマ RD カウンタ 0 のカウントソース} - 1$$

今回、PWM 波形の周期を 16ms にします。タイマ RD カウンタ 0 のカウントソースとは、タイマ RD 制御レジスタ 0(TRDCR0)のカウントソース選択ビット(bit2~0)で設定した時間のことで、今回は 400ns に設定しています。よって、タイマ RD ジェネラルレジスタ A0(TRDGRA0)は次のようになります。

$$\begin{aligned} \text{TRDGRA0} &= \text{周期} / \text{カウントソース} - 1 \\ \text{TRDGRA0} &= (16 \times 10^{-3}) / (400 \times 10^{-9}) - 1 \\ \text{TRDGRA0} &= 40000 - 1 = 39999 \end{aligned}$$

TRDGRA0 の値は、65,535 以下にする必要があります。今回の結果は、65,535 以下なので、400ns の設定で大丈夫です。65,536 以上になった場合、タイマ RD 制御レジスタ 0(TRDCR0)のカウントソース選択ビット(bit2~0)の設定を大きい時間にしてください。

⑫タイマ RD ジェネラルレジスタ A1(TRDGRA1:Timer RD General register A1)の設定

タイマ RD ジェネラルレジスタ A1(TRDGRA1)に値を設定することによって、TRDIOB1(P2_5) 端子,TRDIOC1(P2_6)端子,TRDIOD1(P2_7)端子から出力する PWM 波形の周期を設定します。

PWM 周期は、下記の式で決まります。

$$\text{PWM 周期} = \text{タイマ RD カウンタ 1 のカウントソース} \times (\text{TRDGRA1} + 1)$$

TRDGRA1 を左辺に移動して、TRDGRA1 を求める式に変形します。

$$\text{TRDGRA1} = \text{PWM 周期} / \text{タイマ RD カウンタ 1 のカウントソース} - 1$$

今回、PWM 波形の周期を 16ms にします。タイマ RD カウンタ 1 のカウントソースとは、タイマ RD 制御レジスタ 1(TRDCR1)のカウントソース選択ビット(bit2~0)で設定した時間のことで、今回は 400ns に設定しています。よって、タイマ RD ジェネラルレジスタ A1(TRDGRA1)は次のようになります。

$$\begin{aligned} \text{TRDGRA1} &= \text{PWM 周期} / \text{カウントソース} - 1 \\ \text{TRDGRA1} &= (16 \times 10^{-3}) / (400 \times 10^{-9}) - 1 \\ \text{TRDGRA1} &= 40000 - 1 = 39999 \end{aligned}$$

TRDGRA1 の値は、65,535 以下にする必要があります。今回の結果は、65,535 以下なので、400ns の設定で大丈夫です。65,536 以上になった場合、タイマ RD 制御レジスタ 1(TRDCR1)のカウントソース選択ビット(bit2~0)の設定を大きい時間にしてください。

- ⑬タイマ RD ジェネラルレジスタ B0(TRDGRB0:Timer RD General register B0)の設定
- ⑭タイマ RD ジェネラルレジスタ C0(TRDGRC0:Timer RD General register C0)の設定
- ⑮タイマ RD ジェネラルレジスタ D0(TRDGRD0:Timer RD General register D0)の設定
- ⑯タイマ RD ジェネラルレジスタ B1(TRDGRB1:Timer RD General register B1)の設定
- ⑰タイマ RD ジェネラルレジスタ C1(TRDGRC1:Timer RD General register C1)の設定
- ⑱タイマ RD ジェネラルレジスタ D1(TRDGRD1:Timer RD General register D1)の設定

各端子から PWM 波形を出力するときの、出力波形の ON 幅を設定します。

番号	レジスタ名	レジスタ名 略称	出力 端子	ON 幅の計算
⑬	タイマ RD ジェネラル レジスタ B0	TRDGRB0	P2_2	$TRDGRB0 = ON \text{ 幅} \times \text{タイマ RD カウンタ 0 のカウントソース} - 1$
⑭	タイマ RD ジェネラル レジスタ C0	TRDGRC0	P2_1	$TRDGRC0 = ON \text{ 幅} \times \text{タイマ RD カウンタ 0 のカウントソース} - 1$
⑮	タイマ RD ジェネラル レジスタ D0	TRDGRD0	P2_3	$TRDGRD0 = ON \text{ 幅} \times \text{タイマ RD カウンタ 0 のカウントソース} - 1$
⑯	タイマ RD ジェネラル レジスタ B1	TRDGRB1	P2_5	$TRDGRB1 = ON \text{ 幅} \times \text{タイマ RD カウンタ 1 のカウントソース} - 1$
⑰	タイマ RD ジェネラル レジスタ C1	TRDGRC1	P2_6	$TRDGRC1 = ON \text{ 幅} \times \text{タイマ RD カウンタ 1 のカウントソース} - 1$
⑱	タイマ RD ジェネラル レジスタ D1	TRDGRD1	P2_7	$TRDGRD1 = ON \text{ 幅} \times \text{タイマ RD カウンタ 1 のカウントソース} - 1$

例として、タイマ RD ジェネラルレジスタ B0(TRDGRB0)を計算してみます。TRDGRB0 を設定することにより、P2_2 端子から出力される PWM 波形の ON 幅を設定します。

P2_2 端子から出力される PWM 波形の ON 幅は、下記の式で決まります。

$$TRDGRB0 = P2_2 \text{ 端子から出力される PWM 波形の ON 幅} / \text{タイマ RD カウンタ 0 のカウントソース} - 1$$

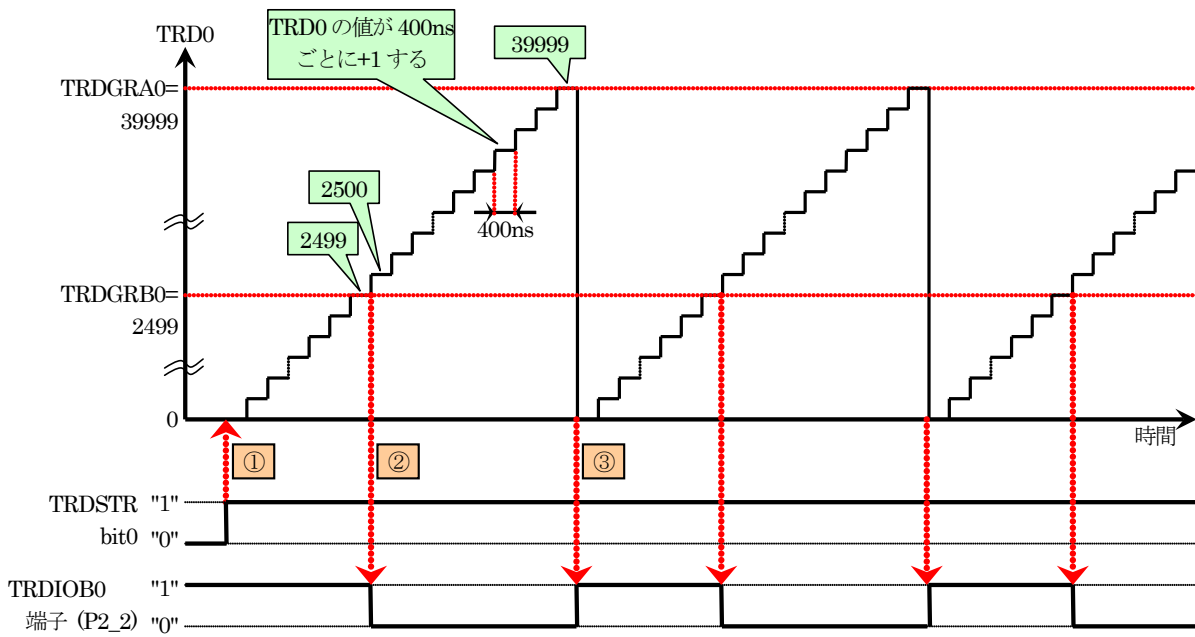
今回、タイマ RD カウンタ 0 のカウントソースは 400ns です。例えば ON 幅を 1ms にするなら、タイマ RD ジェネラルレジスタ B0(TRDGRB0)の値は次のようになります。

$$TRDGRB0 = ON \text{ 幅} / \text{カウントソース} - 1$$

$$TRDGRB0 = (1 \times 10^{-3}) / (400 \times 10^{-9}) - 1$$

$$TRDGRB0 = 2500 - 1 = 2499$$

TRD0、TRDGRA0、TRDGRB0 の値と波形の関係を図解すると下記のようになります。



①	TRDSTR の bit0 を"1"にすると、TRD0 のカウントが開始されます。
②	<p>TRD0=(TRDGRB0+1)になると、P2_2 端子が"0"になります。 TRDGRB0 の値は 2499、TRD0 が+1 する間隔は 400ns です。 よって、P2_2 端子が"1"の時間は、 P2_2 端子が"1"の時間 = (TRDGRB0+1)×TRD0 が+1 する間隔 = (2499+1)×400ns = 1,000,000 [ns] = 1[ms]</p>
③	<p>TRD0=(TRDGRA0+1)になると、P2_2 端子が"1"になります。同時に TRD0 が 0 にクリアされ、また 0 から値が増えていきます。 TRDGRA0 の値は 39999、TRD0 が+1 する間隔は 400ns です。 よって、P2_2 端子の PWM 周期は、 P2_2 端子の PWM 周期 = (TRDGRA0+1)×TRD0 が+1 する間隔 = (39999+1)×400ns = 16,000,000 [ns] = 16[ms]</p> <p>ちなみに、 「PWM 周期」=「ON("1")の時間」+「OFF("0")の時間」 ですので、「0」の時間は、 「OFF("0")の時間」=「PWM 周期」-「ON("1")の時間」 = 16[ms] - 1[ms] = 15[ms] となります。</p>

⑨タイマ RD スタートレジスタ (TRDSTR:Timer RD start register)の設定

TRD0 をカウントさせるか、停止させるか設定します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7~4		"0000"を設定	0000
bit3	TRD1 カウント動作選択ビット csel1_trdstr	0:TRDGRA1 レジスタとのコンペア一致でカウント停止 1:TRDGRA1 レジスタとのコンペア一致後もカウント継続 "1"を設定します。	1
bit2	TRD0 カウント動作選択ビット csel0_trdstr	0:TRDGRA0 レジスタとのコンペア一致でカウント停止 1:TRDGRA0 レジスタとのコンペア一致後もカウント継続 "1"を設定します。	1
bit1	TRD1 カウント開始フラグ(注 4) tstart1_trdstr	0:カウント停止(注 2) 1:カウント開始 設定した瞬間から、TRD1 のカウントが開始されます。 "1"を設定します。	1
bit0	TRD0 カウント開始フラグ(注 3) tstart0_trdstr	0:カウント停止(注 1) 1:カウント開始 設定した瞬間から、TRD0 のカウントが開始されます。 "1"を設定します。	1

注 1. bit2 が“1”に設定されているとき、bit0 へ“0”を書いてください。

注 2. bit3 が“1”に設定されているとき、bit1 へ“0”を書いてください。

注 3. bit2 が“0”でコンペア一致信号(TRDIOA0)が発生したとき、“0”(カウント停止)になります。

注 4. bit3 が“0”でコンペア一致信号(TRDIOA1)が発生したとき、“0”(カウント停止)になります。

タイマ RD スタートレジスタ (TRDSTR)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	1	1	1	1
16 進数	0				f			

22.5.2 main 関数

マイコンボードのディップスイッチの値 0～15 に応じて、PWM 端子から 0%～100%の PWM 波形を出力します。

```

35 : void main( void )
36 : {
37 :     unsigned int    pwm;
38 :
39 :     init();          /* 初期化                */
40 :
41 :     while( 1 ) {
42 :         if( dipsw_get() == 0 ) {
43 :             // trdgra0, trdgralと同じにすると0%出力
44 :             pwm = 39999;
45 :         } else if( dipsw_get() == 15 ) {
46 :             // trdgra0, trdgralより大きくすると100%出力
47 :             pwm = 40000;
48 :         } else {
49 :             // 1～14なら割合に応じてPWM出力する
50 :             pwm = (long)39999 * dipsw_get() / 15;
51 :         }
52 :
53 :         trdgrb0 = pwm;      /* P2_2のON幅設定      */
54 :         trdgrc0 = pwm;      /* P2_1のON幅設定      */
55 :         trdgrd0 = pwm;      /* P2_3のON幅設定      */
56 :         trdgrb1 = pwm;      /* P2_5のON幅設定      */
57 :         trdgrc1 = pwm;      /* P2_6のON幅設定      */
58 :         trdgrd1 = pwm;      /* P2_7のON幅設定      */
59 :     }
60 : }

```

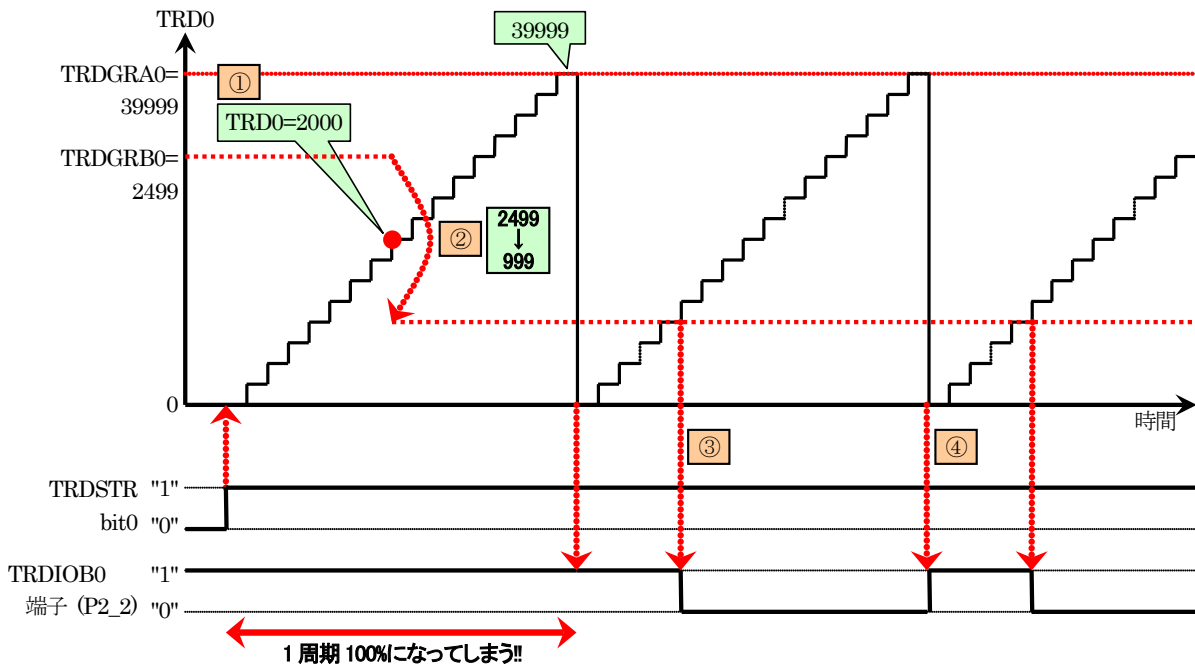
42行	ディップスイッチの値が0かどうかチェックします。0なら、44行を実行します。
44行	PWMを0%出力するために、pwm変数にTRDGRA0(TRDGRA1)と同じ値の39999を設定します。周期を替えた場合は、TRDGRA0(TRDGRA1)の値は39999では無くなりますので、TRDGRA0(TRDGRA1)と同じ値を設定してください。 詳しくは、「21. タイマ RD による PWM 波形出力(リセット同期 PWM モード) (プロジェクト: timer_rd_doukipwm)」を参照してください。
45行	ディップスイッチの値が15かどうかチェックします。15なら、47行を実行します。
47行	PWMを100%出力するために、pwm変数にTRDGRA0(TRDGRA1)より1つ大きい値の40000を設定します。周期を替えた場合は、TRDGRA0(TRDGRA1)の値は40000では無くなりますので、TRDGRA0+1(TRDGRA1+1)の値を設定してください。 詳しくは、「21. タイマ RD による PWM 波形出力(リセット同期 PWM モード) (プロジェクト: timer_rd_doukipwm)」を参照してください。
50行	ディップスイッチの値が1～14の場合、ディップスイッチの割合に応じてPWM値を計算します。
53行～ 58行	各ジェネラルレジスタにpwm変数の値を設定して、PWM波形のON幅を設定します。

マイコンボードのディップスイッチとデューティ比の関係を、下表に示します。

ディップスイッチの値 (dipsw_get 関数)	TRDGRD0 の値 (A)	TRD0 と一致する値 (A)+1	デューティ比 ((A)+1)/40000
0	39999 を直接指定	40000 (2つの一致が 同時に起こり、 "0"が優先される)	0%
1	$39999 * 1 / 15 = 2666$	2667	6.6675%
2	$39999 * 2 / 15 = 5333$	5334	13.3350%
3	$39999 * 3 / 15 = 7999$	8000	20.0000%
4	$39999 * 4 / 15 = 10666$	10667	26.6675%
5	$39999 * 5 / 15 = 13332$	13333	33.3325%
6	$39999 * 6 / 15 = 15999$	16000	40.0000%
7	$39999 * 7 / 15 = 18665$	18666	46.6650%
8	$39999 * 8 / 15 = 21332$	21333	53.3325%
9	$39999 * 9 / 15 = 23998$	23999	59.9975%
10	$39999 * 10 / 15 = 26665$	26666	66.6650%
11	$39999 * 11 / 15 = 29331$	29332	73.3300%
12	$39999 * 12 / 15 = 31998$	31999	79.9975%
13	$39999 * 13 / 15 = 34664$	34665	86.6625%
14	$39999 * 14 / 15 = 37331$	37332	93.3300%
15	40000 を直接指定	一致せず	100%

※問題点

例えば、TRDGRB0=2499、TRD0=2000 のときに、プログラムで TRDGRB0 の値を 999 に変更したとします。



① ②	<p>TRD0 が 2000 のときに(①)、プログラムで TRDGRB0 の値を 2499 から 999 に変更します(②)。 TRD0 の値が 400ns ごとに増えていき 40000 になりました。40000 になると「TRD0=(TRDGRA0+1)」が成り立ち、P2_2 端子が"1"になります。同時に TRD0 が 0 にクリアされます。この間、「TRD0=(TRDGRB0+1)」が成り立つ条件がありませんでした。そのため、今回の 1 周期分すべてが"1"となり 100%出力になってしまいます。 今回のように、「TRD0=(TRDGRB0+1)」が成り立つ前に TRDGRB0 の値を TRD0 より小さい値に変更すると、PWM100%という想定外の値になります。</p>
③	TRD0=(TRDGRB0+1)になるため、P2_2 端子が"0"になります。
④	TRD0=(TRDGRA0+1)になるため、P2_2 端子が"1"になります。

このように、TRDGRB0 の値を変えるタイミングによっては PWM100%という想定外の波形になってしまいます。モータを制御していたら、一瞬ですが 100%の回転となってしまう、暴走してしまうかもしれません。

リセット同期 PWM モードの場合、バッファレジスタがありました。PWM モードにはバッファレジスタの機能がありません。そのため、プログラムでこのような不具合が起こらないようにします。

各ジェネラルレジスタを設定するとき、下記のようにプログラムします。

```
// trdgrb0を設定するとき
while( trd0 <= trdgrb0 && trdgra0 > trdgrb0 );
trdgrb0 = pwm;

// trdgrc0を設定するとき
while( trd0 <= trdgrc0 && trdgra0 > trdgrc0 );
trdgrc0 = pwm;

// trdgrd0を設定するとき
while( trd0 <= trdgrd0 && trdgra0 > trdgrd0 );
trdgrd0 = pwm;

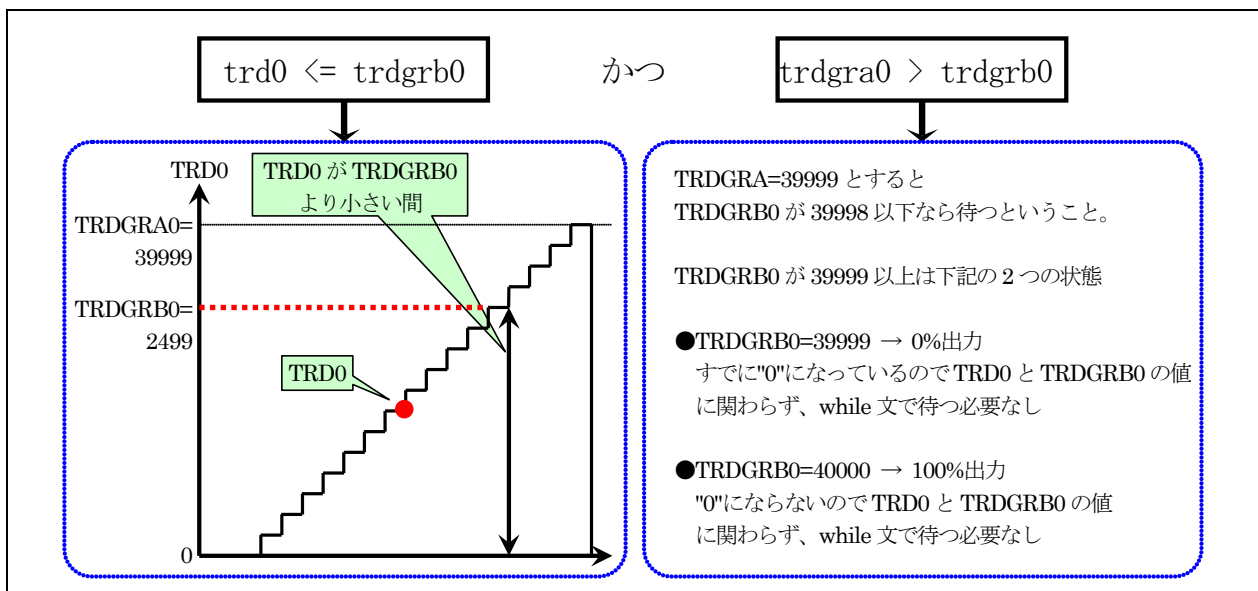
// trdgrb1を設定するとき
while( trd1 <= trdgrb1 && trdgra1 > trdgrb1 );
trdgrb1 = pwm;

// trdgrc1を設定するとき
while( trd1 <= trdgrc1 && trdgra1 > trdgrc1 );
trdgrc1 = pwm;

// trdgrd1を設定するとき
while( trd1 <= trdgrd1 && trdgra1 > trdgrd1 );
trdgrd1 = pwm;
```

不具合が発生するタイミングかどうか、while 文でチェックします。不具合が発生するタイミングの場合、while 文の行で不具合が発生しなくなるタイミングまで待ちます。そのため、待つ時間が長くなるとバッファレジスタを使うよりプログラム実行時間が長くなります。

while 文のカッコの中の意味を下記に示します。



22.6 演習

- (1) 問題点を解決したプログラムに改造しなさい。
- (2) TRDGRC0(P2_1)=0%、TRDGRB0(P2_2)=20%、TRDGRD0(P2_3)=40%、TRDGRB1=60%、TRDGRC1(P2_6)=80%、TRDGRD1(P2_7)=100%を初期 PWM 値として、1 秒ごとに 20%ずつ PWM 値を増やすプログラムを作成しなさい。ただし、100%の次は 0%とする。
- (3) (2)の初期 PWM 値から初めて、1 秒ごとに 10%ずつ PWM 値を減らすプログラムを作成しなさい。ただし、0%の次は 100%とする。

23. モータの制御(プロジェクト: motor)

23.1 概要

本章では、ミニマイコンカーVer.2 の左モータ、右モータの回転を制御する方法を紹介します。プログラムで、左モータ、右モータそれぞれを正転、停止、逆転させることができます。また、正転、逆転は、ゆっくり正転、速めに逆転など、プログラムでスピードを制御することができます。スピード制御は、タイマRDによるリセット同期PWMモードを使用します。

なお本章では、タイマ RB による割り込みを使っていますがここでは説明していません。タイマ RB による割り込みについては、「14. 割り込みによるタイマ(プロジェクト: timer2)」を参照してください。

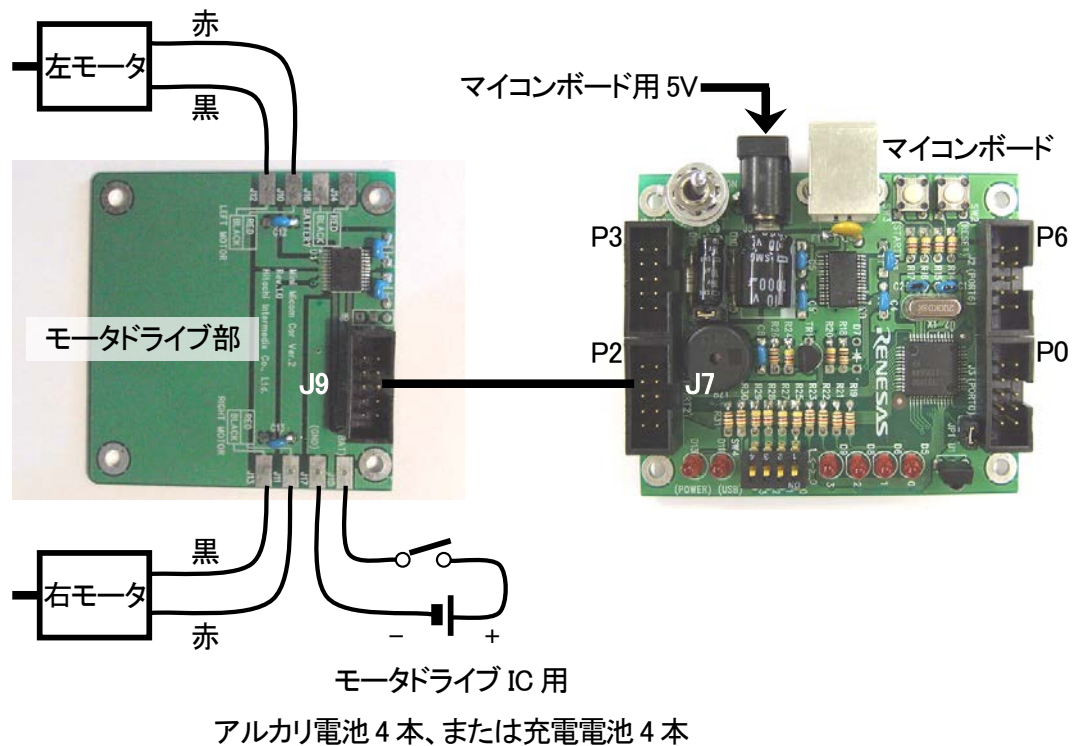
23.2 接続

■使用ポート

マイコンのポート	接続内容
P2 (J7)	ミニマイコンカーVer.2 のモータドライブ部を使用

■接続例

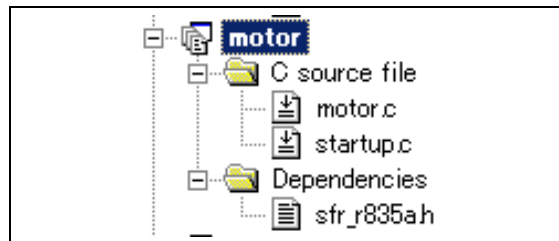
ミニマイコンカーVer.2 のマイコンボードとモータドライブ部を分離している場合の接続図を下記に示します。分離していない(購入時のまま)場合は、特に結線はありません。そのままの状態、本実習の演習ができます。



■操作方法

操作は特にありません。電源を入れると左右のモータが動き出します。右モータ、左モータの動きをよく観察してください。モータ、タイヤが回りますのでミニマイコンカーVer.2 は浮かした状態で実験してください。

23.3 プロジェクトの構成



	ファイル名	内容
1	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。このファイルは共通で、どのプロジェクトもこのファイルから実行されます。
2	motor.c	実際に制御するプログラムが書かれています。R8C/35Aの内蔵周辺機能(SFR)の初期化も行います。
3	sfr_r835a.h	R8C/35Aマイコンの内蔵周辺機能を制御するためのレジスタ(Special Function Registers)を定義したファイルです。

23.4 プログラム「motor.c」

```

1 : /******
2 : /* 対象マイコン R8C/35A */
3 : /* ファイル内容 ミニマイコンカーVer.2のモータ制御 */
4 : /* バージョン Ver. 1.20 */
5 : /* Date 2010.04.19 */
6 : /* Copyright ルネサスマイコンカーラリー事務局 */
7 : /* 日立インターメディアックス株式会社 */
8 : /******
9 : /*
10 : 入力：マイコンボード上のディップスイッチ
11 : 出力：ミニマイコンカーVer.2の右モータ、左モータ
12 :
13 : ミニマイコンカーVer.2の右モータ、左モータを制御します。
14 : マイコンボード上のディップスイッチで、スピード調整することができます。
15 : */
16 :
17 : /*=====*/
18 : /* インクルード */
19 : /*=====*/
20 : #include "sfr_r835a.h" /* R8C/35A SFRの定義ファイル */
21 :
22 : /*=====*/
23 : /* シンボル定義 */
24 : /*=====*/
25 : #define PWM_CYCLE 39999 /* モータPWMの周期 */
26 :
27 : /*=====*/
28 : /* プロトタイプ宣言 */
29 : /*=====*/
30 : void init( void );
31 : void timer( unsigned long timer_set );
32 : void motor( int data1, int data2 );
33 : unsigned char dipsw_get( void );
34 :

```

23. モータの制御(プロジェクト: motor)

```

35 : /*=====*/
36 : /* グローバル変数の宣言 */
37 : /*=====*/
38 : unsigned long cnt_rb; /* タイマRB用 */
39 :
40 : /*=====*/
41 : /* メインプログラム */
42 : /*=====*/
43 : void main( void )
44 : {
45 :     init(); /* 初期化 */
46 :     asm(" fset I "); /* 全体の割り込み許可 */
47 :
48 :     while( 1 ) {
49 :         motor( 100 , 0);
50 :         timer( 1000 );
51 :         motor( 0, 80 );
52 :         timer( 1000 );
53 :         motor( -60, 0 );
54 :         timer( 1000 );
55 :         motor( 0, -40 );
56 :         timer( 1000 );
57 :         motor( 0, 0 );
58 :         timer( 1000 );
59 :     }
60 : }
61 :
62 : /*=====*/
63 : /* R8C/35A スペシャルファンクションレジスタ(SFR)の初期化 */
64 : /*=====*/
65 : void init( void )
66 : {
67 :     int i;
68 :
69 :     /* クロックをXINクロック(20MHz)に変更 */
70 :     prc0 = 1; /* プロテクト解除 */
71 :     cm13 = 1; /* P4_6, P4_7をXIN-XOUT端子にする */
72 :     cm05 = 0; /* XINクロック発振 */
73 :     for(i=0; i<50; i++ ); /* 安定するまで少し待つ(約10ms) */
74 :     ocd2 = 0; /* システムクロックをXINにする */
75 :     prc0 = 0; /* プロテクトON */
76 :
77 :     /* ポートの入出力設定 */
78 :     prc2 = 1; /* PD0のプロテクト解除 */
79 :     pd0 = 0xe0; /* 7-5:LED 4:MicroSW 3-0:Sensor */
80 :     p1 = 0x0f; /* 3-0:LEDは消灯 */
81 :     pd1 = 0xdf; /* 5:RXD0 4:TXD0 3-0:LED */
82 :     pd2 = 0xfe; /* 7-1:モータドライブ部 0:PushSW */
83 :     pd3 = 0xfb; /* 4:Buzzer 2:IR */
84 :     pd4 = 0x80; /* 7:XOUT 6:XIN 5-3:DIP SW 2:VREF */
85 :     pd5 = 0x40; /* 7:DIP SW */
86 :     pd6 = 0xff;
87 :
88 :     /* タイマRBの設定 */
89 :     /* 割り込み周期 = 1 / 20[MHz] * (TRBPRE+1) * (TRBPR+1)
90 :     = 1 / (20*10-6) * 200 * 100
91 :     = 0.001[s] = 1[ms]
92 :
93 :     /*
94 :     trbmr = 0x00; /* 動作モード、分周比設定 */
95 :     trbpre = 200-1; /* プリスケアラレジスタ */
96 :     trbpr = 100-1; /* プライマリレジスタ */
97 :     trbic = 0x07; /* 割り込み優先レベル設定 */
98 :     trbcr = 0x01; /* カウント開始 */
99 :
100 :     /* タイマRD リセット同期PWMモードの設定 */
101 :     /* PWM周期 = 1 / 20[MHz] * カウントソース * (TRDGRA0+1)
102 :     = 1 / (20*10-6) * 8 * 40000
103 :     = 0.016[s] = 16[ms]
104 :
105 :     /*
106 :     trdfcr = 0x01; /* リセット同期PWMモードに設定 */
107 :     trdmr = 0xf0; /* バッファレジスタ設定 */
108 :     trdoer1 = 0xcd; /* 出力端子の選択 */
109 :     trdpsr0 = 0x08; /* TRDIOB0, C0, D0端子設定 */
110 :     trdpsr1 = 0x05; /* TRDIOA1, B1, C1, D1端子設定 */
111 :     trdcr0 = 0x23; /* ソースカウンタの選択:f8 */
112 :     trdgra0 = trdgrc0 = PWM_CYCLE; /* 周期 */
113 :     trdgrb0 = trdgrd0 = 0; /* P2_2端子のON幅設定 */
114 :     trdgra1 = trdgrc1 = 0; /* P2_4端子のON幅設定 */
115 :     trdgrb1 = trdgrd1 = 0; /* P2_5端子のON幅設定 */
116 :     trdstr = 0x0d; /* TRD0カウント開始 */

```

23. モータの制御(プロジェクト: motor)

```

117 : /*****/
118 : /* ディップスイッチ値読み込み */
119 : /* 戻り値 スイッチ値 0~15 */
120 : /*****/
121 : unsigned char dipsw_get( void )
122 : {
123 :     unsigned char sw, sw1, sw2;
124 :
125 :     sw1 = (p5>>4) & 0x08;          /* ディップスイッチ読み込み3 */
126 :     sw2 = (p4>>3) & 0x07;          /* ディップスイッチ読み込み2,1,0*/
127 :     sw = sw1 | sw2;               /* P5とP4の値を合わせる */
128 :
129 :     return sw;
130 : }
131 :
132 : /*****/
133 : /* タイマ本体 */
134 : /* 引数 タイマ値 1=1ms */
135 : /*****/
136 : void timer( unsigned long timer_set )
137 : {
138 :     cnt_rb = 0;
139 :     while( cnt_rb < timer_set );
140 : }
141 :
142 : /*****/
143 : /* タイマRB 割り込み処理 */
144 : /*****/
145 : #pragma interrupt intTRB(vect=24)
146 : void intTRB( void )
147 : {
148 :     cnt_rb++;
149 : }
150 :
151 : /*****/
152 : /* モータ速度制御 */
153 : /* 引数 左モータ:-100~100、右モータ:-100~100 */
154 : /* 0で停止、100で正転100%、-100で逆転100% */
155 : /* 戻り値 なし */
156 : /*****/
157 : void motor( int data1, int data2 )
158 : {
159 :     int motor_r, motor_l, sw_data;
160 :
161 :     sw_data = dipsw_get() + 5;
162 :     motor_l = data1 * sw_data / 20;
163 :     motor_r = data2 * sw_data / 20;
164 :
165 :     /* 左モータ制御 */
166 :     if( motor_l >= 0 ) {
167 :         p2 &= 0xfd;
168 :         p2 |= 0x40;
169 :         trdgrd0 = (long)( PWM_CYCLE - 1 ) * motor_l / 100;
170 :     } else {
171 :         p2 |= 0x02;
172 :         p2 &= 0xbf;
173 :         trdgrd0 = (long)( PWM_CYCLE - 1 ) * ( -motor_l ) / 100;
174 :     }
175 :
176 :     /* 右モータ制御 */
177 :     if( motor_r >= 0 ) {
178 :         p2 &= 0xf7;
179 :         p2 |= 0x80;
180 :         trdgrd1 = (long)( PWM_CYCLE - 1 ) * motor_r / 100;
181 :     } else {
182 :         p2 |= 0x08;
183 :         p2 &= 0x7f;
184 :         trdgrd1 = (long)( PWM_CYCLE - 1 ) * ( -motor_r ) / 100;
185 :     }
186 : }
187 :
188 : /*****/
189 : /* end of file */
190 : /*****/

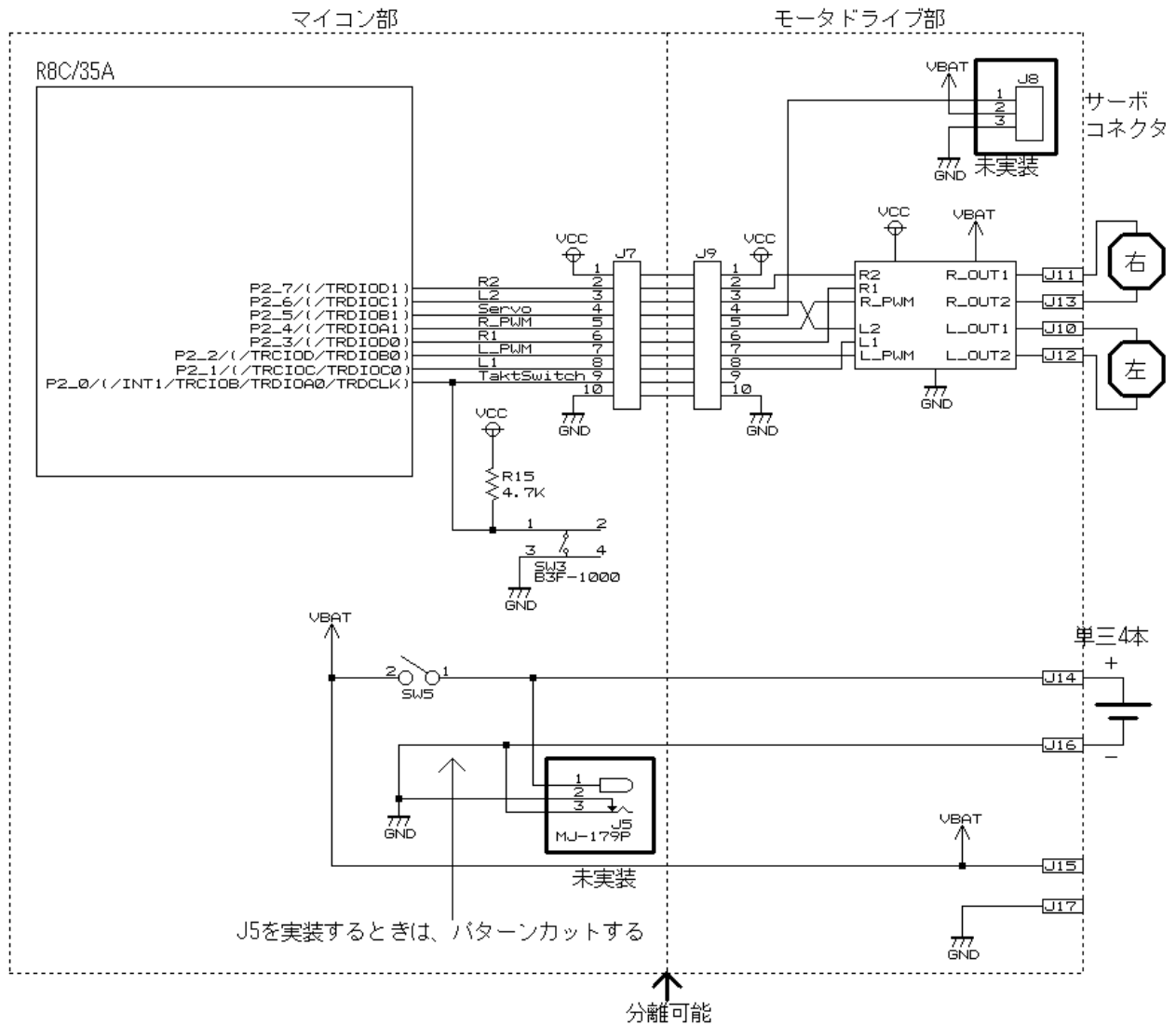
```

23.5 モータドライブ部

23.5.1 マイコン部とモータドライブ部の接続

モータドライブ部の J9 は、マイコン部の J7 に接続します。基板を分離していない場合は、あらかじめパターンで接続されているので何もする必要はありません。分離した場合は、フラットケーブルなどで J7 と J9 を接続しモータ用電源 (VBAT) の配線も行ってください。

マイコン部とモータドライブ部の結線を下記に示します。



※サーボコネクタ(J8)、DC ジャック(J5)はオプションです。

23.5.2 J7 と J9 の詳細

J7 と J9 のピン番号に対する信号名を下表に示します。J7 と J9 はあらかじめパターンで接続されています。マイコン部とモータドライブ部を分離した場合は、フラットケーブルなどで J7 と J9 を接続してください。

プログラムで P2_2 端子、P2_4 端子を PWM 端子にします。オプションでサーボを取り付けることができます。今回はサーボがありませんが、今後のことも考えて P2_5 端子も PWM 端子にしておきます。他は I/O 端子に設定します。PWM の周期は 16ms にします。

J9		J7 (マイコン部)			
ピン番号	信号名	ピン番号	信号名	入出力設定	説明
1	VCC(+5V)	1	VCC(+5V)		
2	モータ右 2	2	P2_7	出力	P2_7 は通常の I/O ポートです。
3	モータ左 2	3	P2_6	出力	P2_6 は通常の I/O ポートです。
4	サーボ (オプション)	4	P2_5	出力 (PWM 波形出力)	この端子は PWM 出力許可にします。TRDGRD1 で ON 幅を設定します。
5	モータ右 PWM	5	P2_4	出力 (PWM 波形出力)	この端子は PWM 出力許可にします。TRDGRC1 で ON 幅を設定します。
6	モータ右 1	6	P2_3	出力	P2_3 は通常の I/O ポートです。
7	モータ左 PWM	7	P2_2	出力 (PWM 波形出力)	この端子は PWM 出力許可にします。TRDGRD0 で ON 幅を設定します。
8	モータ左 1	8	P2_1	出力	P2_1 は通常の I/O ポートです。
9	未接続	9	P2_0	入力	P2_0 は通常の I/O ポートです。マイコンボード上のタクトスイッチに繋がっています。
10	GND	10	GND		

↑パターン上で接続されています。

23.5.3 左モータの動作

左モータは、P2_1、P2_2、P2_6 の 3 端子で制御します。

モータ左 1 P2_1	モータ左 2 P2_6	モータ左 PWM P2_2	モータ動作
1	1	0 または 1	ブレーキ
0	0	0 または 1	フリー
0	1	PWM	PWM="1"なら正転、"0"ならブレーキ
1	0	PWM	PWM="1"なら逆転、"0"ならブレーキ

左モータを正転させたい場合、P2_1="0"、P2_6="1"にして、P2_2 から PWM 波形を出力すると、左モータが PWM の割合に応じて正転します。例えば、PWM が 0%ならモータの回転は停止、PWM が 50%ならモータの回転は正転 50%、PWM100%ならモータの回転は正転 100%になります。

左モータを逆転させたい場合、P2_1="1"、P2_6="0"にして、P2_2 から PWM 波形を出力すると、左モータが PWM の割合に応じて逆転します。例えば、PWM が 0%ならモータの回転は停止、PWM が 50%ならモータの回転は逆転 50%、PWM100%ならモータの回転は逆転 100%になります。

23.5.4 右モータの動作

右モータは、P2_3、P2_4、P2_7 の 3 端子で制御します。

モータ右 1 P2_3	モータ右 2 P2_7	モータ右 PWM P2_4	モータ動作
1	1	0 または 1	ブレーキ
0	0	0 または 1	フリー
0	1	PWM	PWM="1"なら正転、"0"ならブレーキ
1	0	PWM	PWM="1"なら逆転、"0"ならブレーキ

右モータを正転させたい場合、P2_3="0"、P2_7="1"にして、P2_4 から PWM 波形を出力すると、右モータが PWM の割合に応じて正転します。例えば、PWM が 0%ならモータの回転は停止、PWM が 50%ならモータの回転は正転 50%、PWM100%ならモータの回転は正転 100%になります。

右モータを逆転させたい場合、P2_3="1"、P2_7="0"にして、P2_4 から PWM 波形を出力すると、右モータが PWM の割合に応じて逆転します。例えば、PWM が 0%ならモータの回転は停止、PWM が 50%ならモータの回転は逆転 50%、PWM100%ならモータの回転は逆転 100%になります。

23.6 プログラムの解説

23.6.1 init 関数(タイマ RD の設定)

タイマ RD を使いリセット同期 PWM モードの設定を行います。

```

22 : /*=====*/
23 : /* シンボル定義 */
24 : /*=====*/
25 : #define PWM_CYCLE 39999 /* モータPWMの周期 */

中略

104 : trdfcr = 0x01; /* リセット同期PWMモードに設定 */
105 : trdmr = 0xf0; /* バッファレジスタ設定 */
106 : trdoerl = 0xcd; /* 出力端子の選択 */
107 : trdpsr0 = 0x08; /* TRDIOB0, C0, D0端子設定 */
108 : trdpsr1 = 0x05; /* TRDIOA1, B1, C1, D1端子設定 */
109 : trdcr0 = 0x23; /* ソースカウントの選択:f8 */
110 : trdgra0 = trdgrc0 = PWM_CYCLE; /* 周期 */
111 : trdgrb0 = trdgrd0 = 0; /* P2_2端子のON幅設定 */
112 : trdgral = trdgrcl = 0; /* P2_4端子のON幅設定 */
113 : trdgrbl = trdgrdl = 0; /* P2_5端子のON幅設定 */
114 : trdstr = 0x0d; /* TRD0カウント開始 */

```

PWM の周期は 16ms です。110 行で、タイマ RD ジェネラルレジスタ A0(TRDGRA0)、タイマ RD ジェネラルレジスタ C0(TRDGRC0)に計算結果である 39999 を設定します。今回は 25 行目で「PWM_CYCLE」という PWM の周期を設定する定数を定義して、「PWM_CYCLE」に 39999 を割り当てます。周期を変更する場合は、25 行にある「39999」を変更してください。

①タイマ RD 機能制御レジスタ(TRDFCR: Timer RD function control register)の設定

タイマ RD の機能を設定します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7		"0"を設定	0
bit6	外部クロック入力選択ビット stclk_trdfer	0:外部クロック入力無効 1:外部クロック入力有効 今回は、内部のクロックを使用しますので、無効を選択 します。	0
bit5,4		"00"を設定	00
bit3	逆相出力レベル選択ビット (リセット同期 PWM モードまた は相補 PWM モード時) ols1_trdfer	0:初期出力"H"、アクティブレベル"L" 1:初期出力"L"、アクティブレベル"H" 今回は、"0"を設定します。	0
bit2	正相出力レベル選択ビット (リセット同期 PWM モードまた は相補 PWM モード時) ols0_trdfer	0:初期出力"H"、アクティブレベル"L" 1:初期出力"L"、アクティブレベル"H" 今回は、"0"を設定します。	0
bit1,0	コンビネーションモード選択 ビット bit1:cmd1_trdfer bit0:cmd0_trdfer	リセット同期 PWM モードでは"01"(リセット同期 PWM モード)にしてください	01

タイマ RD 機能制御レジスタ(TRDFCR)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	0	0	0	1
16 進数	0				1			

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

タイマ RD のジェネラルレジスタをバッファレジスタとして使用するかどうか設定します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7	TRDGRD1 レジスタ機能選択 ビット bfd1_trdmr	0:ジェネラルレジスタ 1:TRDGRB1 レジスタのバッファレジスタ バッファレジスタとして使用します。	1
bit6	TRDGRC1 レジスタ機能選択 ビット bfc1_trdmr	0:ジェネラルレジスタ 1:TRDGRA1 レジスタのバッファレジスタ バッファレジスタとして使用します。	1
bit5	TRDGRD0 レジスタ機能選択 ビット bfd0_trdmr	0:ジェネラルレジスタ 1:TRDGRB0 レジスタのバッファレジスタ バッファレジスタとして使用します。	1
bit4	TRDGRC0 レジスタ機能選択 ビット bfc0_trdmr	0:ジェネラルレジスタ 1:TRDGRA0 レジスタのバッファレジスタ バッファレジスタとして使用します。	1
bit3~0		"0000"を設定	0000

タイマ RD モードレジスタ(TRDMR)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	1	1	1	1	0	0	0	0
16 進数	f				0			

③タイマ RD アウトプットマスタ許可レジスタ 1(TRDOER1: Timer RD output master enable register 1)の設定

リセット同期 PWM モードは 7 端子から PWM 波形を出力することができますが、出力するか、通常の I/O ポートとして使用するかを選択できます。

今回は、P2_2 端子、P2_4 端子、P2_5 端子を PWM 波形出力、その他の端子は I/O ポートとして使用します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7	TRDIOD1(P2_7)出力禁止ビット ed1_trdoer1	0:出力許可 1:出力禁止(TRDIOD1 端子はプログラマブル入出力ポート) 今回は、出力を禁止します。	1
bit6	TRDIOC1(P2_6)出力禁止ビット ec1_trdoer1	0:出力許可 1:出力禁止(TRDIOC1 端子はプログラマブル入出力ポート) 今回は、出力を禁止します。	1
bit5	TRDIOB1(P2_5)出力禁止ビット eb1_trdoer1	0:出力許可 1:出力禁止(TRDIOB1 端子はプログラマブル入出力ポート) サーボ PWM 用に、出力を許可します。	0
bit4	TRDIOA1(P2_4)出力禁止ビット ea1_trdoer1	0:出力許可 1:出力禁止(TRDIOA1 端子はプログラマブル入出力ポート) 右モータ PWM 用に、出力を許可します。	0
bit3	TRDIOD0(P2_3)出力禁止ビット ed0_trdoer1	0:出力許可 1:出力禁止(TRDIOD0 端子はプログラマブル入出力ポート) 今回は、出力を禁止します。	1
bit2	TRDIOC0(P2_1)出力禁止ビット ec0_trdoer1	0:出力許可 1:出力禁止(TRDIOC0 端子はプログラマブル入出力ポート) 今回は、出力を禁止します。	1
bit1	TRDIOB0(P2_2)出力禁止ビット eb0_trdoer1	0:出力許可 1:出力禁止(TRDIOB0 端子はプログラマブル入出力ポート) 左モータ PWM 用に、出力を許可します。	0
bit0		"1"を設定	1

タイマ RD アウトプットマスタ許可レジスタ 1(trdoer1)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	1	1	0	0	1	1	0	1
16 進数	c				d			

④タイマ RD 端子選択レジスタ 0(TRDPSR0:Timer RD function select register 0)の設定

PWM 波形の出力端子をどのポートに割り当てるか設定します。タイマ RD 端子選択レジスタ 0(TRDPSR0)では、TRDIOD0 端子、TRDIOC0 端子、TRDIOB0 端子の割り当てを設定します。

R8C/35A では、割り当てる端子は決まっております端子を変更することができません。PWM 端子として割り当てるか、割り当てないか(通常の I/O ポートとして使用するか)を設定します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7		"0"を設定	0
bit6	TRDIOD0 端子選択ビット trdiiod0sel0	0:TRDIOD0 端子は使用しない 1:P2_3 に割り当てる 今回は、使用しません。	0
bit5,4	TRDIOC0 端子選択ビット bit5:trdioc0sel1 bit4:trdioc0sel0	00:TRDIOC0 端子は使用しない 01:設定しないでください 10:P2_1 に割り当てる 11:設定しないでください 今回は、使用しません。	00
bit3,2	TRDIOB0 端子選択ビット bit3:trdiob0sel1 bit2:trdiob0sel0	00:TRDIOB0 端子は使用しない 01:設定しないでください 10:P2_2 に割り当てる 11:設定しないでください 今回は、P2_2 に割り当てて、この端子からPWM 波形を出力します。左モータ PWM 用です。	10
bit1,0		"0"を設定	00

タイマ RD 端子選択レジスタ 0(TRDPSR0)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	1	0	0	0
16 進数	0				8			

⑤タイマ RD 端子選択レジスタ 1(TRDPSR1:Timer RD function select register 1)の設定

PWM 波形の出力端子をどのポートに割り当てるか設定します。タイマ RD 端子選択レジスタ 1(TRDPSR1)では、TRDIOD1 端子、TRDIOC1 端子、TRDIOB1 端子、TRDIOA1 端子の割り当てを設定します。

R8C/35A では、割り当てる端子は決まっております端子を変更することができません。PWM 端子として割り当てるか、割り当てないか(通常の I/O ポートとして使用するか)を設定します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7		"0"を設定	0
bit6	TRDIOD1 端子選択ビット trdioid1sel0	0:TRDIOD1 端子は使用しない 1:P2_7 に割り当てる 今回は、使用しません。	0
bit5		"0"を設定	0
bit4	TRDIOC1 端子選択ビット trdioc1sel0	0:TRDIOC1 端子は使用しない 1:P2_6 に割り当てる 今回は、使用しません。	0
bit3		"0"を設定	0
bit2	TRDIOB1 端子選択ビット trdiob1sel0	0:TRDIOB1 端子は使用しない 1:P2_5 に割り当てる 今回は、P2_5 に割り当てて、この端子から PWM 波形を出力します。サーボ PWM 用です。	1
bit1		"0"を設定	0
bit0	TRDIOA1 端子選択ビット trdioa1sel0	0:TRDIOA1 端子は使用しない 1:P2_4 に割り当てる 今回は、P2_4 に割り当てて、この端子から PWM 波形を出力します。右モータ PWM 用です。	1

タイマ RD 端子選択レジスタ 1(TRDPSR1)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	0	1	0	1
16 進数	0				5			

⑥タイマ RD 制御レジスタ 0(TRDCR0: Timer RD control register 0)の設定

タイマ RD カウンタ 0(TRD0)がカウントアップする時間などを選択します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7~5	TRD0 カウンタクリア選択ビット bit7:cclr2_trdcr0 bit6:cclr1_trdcr0 bit5:cclr0_trdcr0	リセット同期 PWM モードの場合は、“001”を設定してください (TRDGRA0 とのコンペア一致で TRD0 レジスタクリアにしてください)	001
bit4,3	外部クロックエッジ選択ビット (注 3) bit4:ckeg1_trdcr0 bit3:ckeg0_trdcr0	00: 立ち上がりエッジでカウント 01: 立ち下がりエッジでカウント 10: 両エッジでカウント 11: 設定しないでください 外部クロックは使いませんので何を設定しても変化ありません。今回は 00 を設定します。	00
bit2~0	カウントソース選択ビット bit2:tck2_trdcr0 bit1:tck1_trdcr0 bit0:tck0_trdcr0	000: f1 (1/20MHz=50ns) 001: f2 (2/20MHz=100ns) 010: f4 (4/20MHz=200ns) 011: f8 (8/20MHz=400ns) 100: f32 (32/20MHz=1600ns) 101: TRDCLK 入力(注 1)または fC2 (注 2) fC2 = 2/XCIN クロック=今回は未接続 110: fOCO40M (高速オンチップオシレータ 40MHz= 今回は未接続) 111: fOCO-F(注 4) (高速オンチップオシレータを FRA2 で分周したクロック=今回は未接続) タイマ RD カウンタ 0(TRD0)がカウントアップする時間を 設定します。今回は“011”を設定します。TRD0 は、 400ns ごとに+1 していきます。	011

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

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

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

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

タイマ RD 制御レジスタ 0(TRDCR0)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	1	0	0	0	1	1
16 進数	2				3			

※タイマ RD 制御レジスタ 0(TRDCR0)カウンタソース選択ビットの設定方法

タイマ RD 制御レジスタ 0(TRDCR0)のカウンタソース選択ビット(bit2~0)で、タイマ RD カウンタ 0(TRD0)がどのくらいの間隔で+1 するか設定します。TRD0 は、0 からスタートして最大 65,535 までカウントアップします。65,535 の次は 0 に戻ります。PWM の周期や ON 幅は TRD0 の値を基準にするので、カウントアップする時間×65,536 以上の時間を設定することができません。

タイマ RD 制御レジスタ 0(TRDCR0)のカウンタソース選択ビットの値と、周期の関係を下記に示します。

bit2~0	内容
000	タイマ RD カウンタ 0(TRD0)がカウントアップする時間を、f1 に設定します。時間は、 $f1/20\text{MHz}=1/20\text{MHz}=50\text{ns}$ 設定できる PWM 周期の最大は、 $50\text{ns} \times 65,536 = \mathbf{3.2768\text{ms}}$ よって、この時間以内の PWM 周期を設定する場合は"000"を設定、これ以上の PWM 周期を設定したい場合は次以降の値を検討します。
001	タイマ RD カウンタ 0(TRD0)がカウントアップする時間を、f2 に設定します。時間は、 $f2/20\text{MHz}=2/20\text{MHz}=100\text{ns}$ 設定できる PWM 周期の最大は、 $100\text{ns} \times 65,536 = \mathbf{6.5536\text{ms}}$ よって、この時間以内の PWM 周期を設定する場合は"001"を設定、これ以上の PWM 周期を設定したい場合は次以降の値を検討します。
010	タイマ RD カウンタ 0(TRD0)がカウントアップする時間を、f4 に設定します。時間は、 $f4/20\text{MHz}=4/20\text{MHz}=200\text{ns}$ 設定できる PWM 周期の最大は、 $200\text{ns} \times 65,536 = \mathbf{13.1072\text{ms}}$ よって、この時間以内の PWM 周期を設定する場合は"010"を設定、これ以上の PWM 周期を設定したい場合は次以降の値を検討します。
011	タイマ RD カウンタ 0(TRD0)がカウントアップする時間を、f8 に設定します。時間は、 $f8/20\text{MHz}=8/20\text{MHz}=400\text{ns}$ 設定できる PWM 周期の最大は、 $400\text{ns} \times 65,536 = \mathbf{26.2144\text{ms}}$ よって、この時間以内の PWM 周期を設定する場合は"011"を設定、これ以上の PWM 周期を設定したい場合は次以降の値を検討します。
100	タイマ RD カウンタ 0(TRD0)がカウントアップする時間を、f32 に設定します。時間は、 $f32/20\text{MHz}=32/20\text{MHz}=1600\text{ns}$ 設定できる PWM 周期の最大は、 $1600\text{ns} \times 65,536 = \mathbf{104.8576\text{ms}}$ よって、この時間以内の PWM 周期を設定する場合は"100"を設定します。 これ以上の PWM 周期を設定することはできません。 これ以上の PWM 周期を設定しなくても良いように、回路側を工夫してください。

今回は、PWM 波形の周期を 16ms にするので、

- ①"000"の設定…最大の PWM 周期は 3.2768ms、今回設定したい 16ms の周期を設定できないので不可
- ②"001"の設定…最大の PWM 周期は 6.5536ms、今回設定したい 16ms の周期を設定できないので不可
- ③"010"の設定…最大の PWM 周期は 13.1072ms、今回設定したい 16ms の周期を設定できないので不可
- ④"011"の設定…最大の PWM 周期は 26.2144ms、今回設定したい 16ms の周期を設定できるので OK

よって、"011"を設定します。

⑦タイマ RD ジェネラルレジスタ A0(TRDGRA0: Timer RD General register A0)、
タイマ RD ジェネラルレジスタ C0(TRDGRC0: Timer RD General register C0)の設定

タイマ RD ジェネラルレジスタ A0(TRDGRA0)に値を設定することによって、出力するPWM 波形の周期を設定します。

PWM 周期は、下記の式で決まります。

$$\text{PWM 周期} = \text{タイマ RD カウンタ 0 のカウントソース} \times (\text{TRDGRA0} + 1)$$

TRDGRA0 を左辺に移動して、TRDGRA0 を求める式に変形します。

$$\text{TRDGRA0} = \text{PWM 周期} / \text{タイマ RD カウンタ 0 のカウントソース} - 1$$

今回、PWM 波形の周期を 16ms にします。タイマ RD カウンタ 0 のカウントソースとは、タイマ RD 制御レジスタ 0(TRDCR0)の bit2~0 で設定した時間のことで、今回は 400ns に設定しています。よって、タイマ RD ジェネラルレジスタ A0(TRDGRA0)は次のようになります。

$$\begin{aligned} \text{TRDGRA0} &= \text{周期} / \text{カウントソース} - 1 \\ \text{TRDGRA0} &= (16 \times 10^{-3}) / (400 \times 10^{-9}) - 1 \\ \text{TRDGRA0} &= 40000 - 1 = 39999 \end{aligned}$$

TRDGRA0 の値は、65,535 以下にする必要があります。今回の結果は、65,535 以下なので、400ns の設定で大丈夫です。65,536 以上になった場合、タイマ RD 制御レジスタ 0(TRDCR0)の bit2~0 の設定を大きい時間にしてください。

■設定のポイント

- ※1…タイマ RD ジェネラルレジスタ A0(TRDGRA0)を使うときは、タイマ RD ジェネラルレジスタ C0(TRDGRC0)をバッファレジスタに設定して、ペアで使用してください。
- ※2…タイマ RD ジェネラルレジスタ A0(TRDGRA0)の設定は、イニシャライズ時に 1 回だけ行ってください。2 回目以降、周期を変更したい場合は、タイマ RD ジェネラルレジスタ C0(TRDGRC0)に値を設定してください。
- ※3…タイマ RD ジェネラルレジスタ C0(TRDGRC0)のイニシャライズ時に設定する値は、タイマ RD ジェネラルレジスタ A0(TRDGRA0)と同じ値にしてください。

プログラム例を下記に示します。

```
void main( void )
{
    // メインプログラム
    init();                               /* レジスタの初期化 */
    ~~~~~
    trdgrc0 = 19999;                     /* 2 回目以降は必ず trdgrc0 に設定する */
    ~~~~~
    trdgrc0 = 9999;                       /* 2 回目以降は必ず trdgrc0 に設定する */
}
void init( void )
{
    // レジスタの初期化
    ~~~~~
    trdgra0 = trdgrc0 = 39999;           /* 1 回だけ trdgra0 に値を設定 */
                                           /* trdgrc0 にも同じ値を設定する */
    ~~~~~
}
```

- ⑧タイマ RD ジェネラルレジスタ B0(TRDGRB0: Timer RD General register B0)、
タイマ RD ジェネラルレジスタ D0(TRDGRD0: Timer RD General register D0)の設定

タイマ RD ジェネラルレジスタ B0(TRDGRB0)に値を設定することによって、P2_2 端子から出力される PWM 波形の ON 幅を設定します。P2_3 端子からは、その反転した波形が出力されます。

P2_2 端子から出力される PWM 波形の ON 幅は、下記の式で決まります。

$$\text{P2}_2 \text{ 端子から出力される PWM 波形の ON 幅} = \text{タイマ RD カウンタ 0 のカウントソース} \times (\text{TRDGRB0} + 1)$$

TRDGRB0 を左辺に移動して、TRDGRB0 を求める式に変形します。

$$\text{TRDGRB0} + 1 = \text{P2}_2 \text{ 端子から出力される PWM 波形の ON 幅} / \text{タイマ RD カウンタ 0 のカウントソース}$$

今回、タイマ RD カウンタ 0 のカウントソースは 400ns です。例えば ON 幅を 1ms にするなら、タイマ RD ジェネラルレジスタ B0(TRDGRB0)の値は次のようになります。

$$\begin{aligned} \text{TRDGRB0} + 1 &= \text{ON 幅} / \text{カウントソース} \\ \text{TRDGRB0} + 1 &= (1 \times 10^{-3}) / (400 \times 10^{-9}) \\ \text{TRDGRB0} + 1 &= 2500 \\ \text{TRDGRB0} &= 2499 \end{aligned}$$

■設定のポイント

- ※1…タイマ RD ジェネラルレジスタ B0(TRDGRB0)を使うときは、タイマ RD ジェネラルレジスタ D0(TRDGRD0)をバッファレジスタに設定して、ペアで使用してください。
- ※2…タイマ RD ジェネラルレジスタ B0(TRDGRB0)の設定は、イニシャライズ時に 1 回だけ行ってください。2 回目以降、ON 幅を変更したい場合は、タイマ RD ジェネラルレジスタ D0(TRDGRD0)に値を設定してください。
- ※3…タイマ RD ジェネラルレジスタ D0(TRDGRD0)のイニシャライズ時に設定する値は、タイマ RD ジェネラルレジスタ B0(TRDGRB0)と同じ値にしてください。

- ⑨タイマ RD ジェネラルレジスタ A1(TRDGRA1: Timer RD General register A1)、
タイマ RD ジェネラルレジスタ C1(TRDGRC1: Timer RD General register C1)の設定

タイマ RD ジェネラルレジスタ A1(TRDGRA1)に値を設定することによって、P2_4 端子から出力される PWM 波形の ON 幅を設定します。P2_6 端子からは、その反転した波形が出力されます。

P2_4 端子から出力される PWM 波形の ON 幅は、下記の式で決まります。

$$\text{P2}_4 \text{ 端子から出力される PWM 波形の ON 幅} = \text{タイマ RD カウンタ 0 のカウントソース} \times (\text{TRDGRA1} + 1)$$

TRDGRA1 を左辺に移動して、TRDGRA1 を求める式に変形します。

$$\text{TRDGRA1} = \text{P2}_4 \text{ 端子から出力される PWM 波形の ON 幅} / \text{タイマ RD カウンタ 0 のカウントソース} - 1$$

今回、タイマ RD カウンタ 0 のカウントソースは 400ns です。ON 幅を 10ms にするなら、タイマ RD ジェネラルレジスタ A1(TRDGRA1)は次のようになります。

$$\begin{aligned} \text{TRDGRA1} &= \text{ON 幅} / \text{カウントソース} - 1 \\ \text{TRDGRA1} &= (10 \times 10^{-3}) / (400 \times 10^{-9}) - 1 \\ \text{TRDGRA1} &= 25000 - 1 = 24999 \end{aligned}$$

■設定のポイント

- ※1…タイマ RD ジェネラルレジスタ A1(TRDGRA1)を使うときは、タイマ RD ジェネラルレジスタ C1(TRDGRC1)をバッファレジスタに設定して、ペアで使用してください。
- ※2…タイマ RD ジェネラルレジスタ A1(TRDGRA1)の設定は、イニシャライズ時に 1 回だけ行ってください。2 回目以降、ON 幅を変更したい場合は、タイマ RD ジェネラルレジスタ C1(TRDGRC1)に値を設定してください。
- ※3…タイマ RD ジェネラルレジスタ C1(TRDGRC1)のイニシャライズ時に設定する値は、タイマ RD ジェネラルレジスタ A1(TRDGRA1)と同じ値にしてください。

⑩タイマ RD ジェネラルレジスタ B1(TRDGRB1: Timer RD General register B1)、
タイマ RD ジェネラルレジスタ D1(TRDGRD1: Timer RD General register D1)の設定

タイマ RD ジェネラルレジスタ B1(TRDGRB1)に値を設定することによって、P2_5 端子から出力される PWM 波形の ON 幅を設定します。P2_7 端子からは、その反転した波形が出力されます。

P2_5 端子から出力される PWM 波形の ON 幅は、下記の式で決まります。

$$\text{P2}_5 \text{ 端子から出力される PWM 波形の ON 幅} = \text{タイマ RD カウンタ 0 のカウントソース} \times (\text{TRDGRB1} + 1)$$

TRDGRB1 を左辺に移動して、TRDGRB1 を求める式に変形します。

$$\text{TRDGRB1} = \text{P2}_5 \text{ 端子から出力される PWM 波形の ON 幅} / \text{タイマ RD カウンタ 0 のカウントソース} - 1$$

今回、タイマ RD カウンタ 0 のカウントソースは 400ns です。ON 幅を 15ms にするなら、タイマ RD ジェネラルレジスタ B1(TRDGRB1)は次のようになります。

$$\begin{aligned} \text{TRDGRB1} &= \text{ON 幅} / \text{カウントソース} - 1 \\ \text{TRDGRB1} &= (15 \times 10^{-3}) / (400 \times 10^{-9}) - 1 \\ \text{TRDGRB1} &= 37500 - 1 = 37499 \end{aligned}$$

■設定のポイント

- ※1…タイマ RD ジェネラルレジスタ B1(TRDGRB1)を使うときは、タイマ RD ジェネラルレジスタ D1(TRDGRD1)をバッファレジスタに設定して、ペアで使用してください。
- ※2…タイマ RD ジェネラルレジスタ B1(TRDGRB1)の設定は、イニシャライズ時に 1 回だけ行ってください。2 回目以降、ON 幅を変更したい場合は、タイマ RD ジェネラルレジスタ D1(TRDGRD1)に値を設定してください。
- ※3…タイマ RD ジェネラルレジスタ D1(TRDGRD1)のイニシャライズ時に設定する値は、タイマ RD ジェネラルレジスタ B1(TRDGRB1)と同じ値にしてください。

⑩タイマ RD スタートレジスタ (TRDSTR: Timer RD start register) の設定

TRD0 をカウントさせるか、停止させるか設定します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7~4		"0000"を設定	0000
bit3	TRD1 カウント動作選択ビット csel1_trdstr	0: TRDGRA1 レジスタとのコンペア一致でカウント停止 1: TRDGRA1 レジスタとのコンペア一致後もカウント継続 今回は"1"を設定します。	1
bit2	TRD0 カウント動作選択ビット csel0_trdstr	0: TRDGRA0 レジスタとのコンペア一致でカウント停止 1: TRDGRA0 レジスタとのコンペア一致後もカウント継続 今回は"1"を設定します。	1
bit1	TRD1 カウント開始フラグ(注 4) tstart1_trdstr	0: カウント停止(注 2) 1: カウント開始 設定した瞬間から、TRD1 のカウントが開始されます。リ セット同期 PWM モードでは TRD1 は使いませんので "0"を設定します。	0
bit0	TRD0 カウント開始フラグ(注 3) tstart0_trdstr	0: カウント停止(注 1) 1: カウント開始 設定した瞬間から、TRD0 のカウントが開始されます。 "1"を設定します。	1

注 1. bit2 が"1"に設定されているとき、bit0 へ"0"を書いてください。

注 2. bit3 が"1"に設定されているとき、bit1 へ"0"を書いてください。

注 3. bit2 が"0"でコンペア一致信号(TRDIOA0)が発生したとき、"0"(カウント停止)になります。

注 4. bit3 が"0"でコンペア一致信号(TRDIOA1)が発生したとき、"0"(カウント停止)になります。

タイマ RD スタートレジスタ (TRDSTR) の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	1	1	0	1
16 進数	0				d			

23.6.2 motor 関数

左モータ、右モータへ PWM を出力する関数です。

```

157 : void motor( int data1, int data2 )
158 : {
159 :     int    motor_r, motor_l, sw_data;
160 :
161 :     sw_data = dipsw_get() + 5;
162 :     motor_l = data1 * sw_data / 20;
163 :     motor_r = data2 * sw_data / 20;
164 :
165 :     /* 左モータ制御 */
166 :     if( motor_l >= 0 ) {
167 :         p2 &= 0xfd;
168 :         p2 |= 0x40;
169 :         trdgrd0 = (long)( PWM_CYCLE - 1 ) * motor_l / 100;
170 :     } else {
171 :         p2 |= 0x02;
172 :         p2 &= 0xbf;
173 :         trdgrd0 = (long)( PWM_CYCLE - 1 ) * ( -motor_l ) / 100;
174 :     }
175 :
176 :     /* 右モータ制御 */
177 :     if( motor_r >= 0 ) {
178 :         p2 &= 0xf7;
179 :         p2 |= 0x80;
180 :         trdgrc1 = (long)( PWM_CYCLE - 1 ) * motor_r / 100;
181 :     } else {
182 :         p2 |= 0x08;
183 :         p2 &= 0x7f;
184 :         trdgrc1 = (long)( PWM_CYCLE - 1 ) * ( -motor_r ) / 100;
185 :     }
186 : }

```

(1) motor 関数の使い方

motor 関数の使い方を下記に示します。

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

引数は、左モータの PWM 値と右モータの PWM 値をカンマで区切って入れます。値とモータの回転の関係を下記に示します。

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

実際にモータに出力される割合を、下記に示します。

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

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

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

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

実際のモータに出力される PWM 値は、下記のようになります。

$$\text{左モータに出力される PWM} = -70 \times (12 + 5) \div 20 = -70 \times 0.85 = -59.5 = -59\%$$

$$\text{右モータに出力される PWM} = 100 \times (12 + 5) \div 20 = 100 \times 0.85 = 85\%$$

左モータの計算結果は-59.5%ですが、小数点は計算できないので切り捨てられ整数になります。よって左モータに出力される PWM 値は逆転 59%、右モータに出力される PWM 値は正転 85%となります。

これから、上記の内容がどのように実行されるのか説明します。

(2) ディップスイッチの割合に応じて、PWM 値を変更

```
161 :      sw_data = dipsw_get() + 5;      dipsw_get() = ディップスイッチの値0~15
162 :      motor_l = data1 * sw_data / 20;
163 :      motor_r = data2 * sw_data / 20;
```

161 行	sw_data 変数にディップスイッチの値+5の値を代入します。ディップスイッチの値は0~15なので、sw_data 変数の値は、5~20 になります。
162 行	motor_l 変数は、左モータに加える PWM 値の割合を代入する変数です。data1 が motor 関数に設定した左モータの PWM 値です。 よって、次の計算を行って motor_l 変数に、左モータに加える PWM 値を設定します。 motor_l = data1(motor 関数で設定した左モータの PWM) × sw_data / 20 motor_l 変数の範囲は、-100~100 の値です。
163 行	motor_r 変数は、右モータに加える PWM 値の割合を代入する変数です。data2 が motor 関数に設定した右モータの PWM 値です。 よって、次の計算を行って motor_r 変数に、右モータに加える PWM 値を設定します。 motor_r = data2(motor 関数で設定した右モータの PWM) × sw_data / 20 motor_r 変数の範囲は、-100~100 の値です。

(3) 左モータ制御

左モータを制御する部分です。左モータの PWM は P2_2 端子から出力します。P2_2 端子から出力する PWM の設定は、タイマ RD ジェネラルレジスタ B0(TRDGRB0)に PWM 値を設定します。ただし、直接設定するのではなく、バッファレジスタであるタイマ RD ジェネラルレジスタ D0(TRDGRD0)に設定します。

```

165 :      /* 左モータ制御 */
166 :      if( motor_l >= 0 ) {
167 :          p2 &= 0xfd;          p2 = p2 AND 0xfdという意味です
168 :          p2 |= 0x40;         p2 = p2 OR 0x40という意味です
169 :          trdgrd0 = (long)( PWM_CYCLE - 1 ) * motor_l / 100;
170 :      } else {
171 :          p2 |= 0x02;         p2 = p2 OR 0x02という意味です
172 :          p2 &= 0xbf;         p2 = p2 AND 0xbfという意味です
173 :          trdgrd0 = (long)( PWM_CYCLE - 1 ) * ( -motor_l ) / 100;
174 :      }
    
```

166 行	<p>左モータの PWM 値の割合が正の数か負の数かをチェックします。 正の数なら 167~169 行、負の数なら 171~173 行を実行します。</p>																																																																								
167 行 ~ 169 行	<p>正の数なら 167~169 行を実行します。 P2_1="0"、P2_6="1"を設定し、P2_2 端子から PWM 出力すると、PWM の割合に応じてモータが正転します。 167 行目で下表の計算を行い、P2_1 端子を"0"にします。</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr><th>bit</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr> <tr><td>元の値 (ポート2)</td><td>P2_7</td><td>P2_6</td><td>P2_5</td><td>P2_4</td><td>P2_3</td><td>P2_2</td><td>P2_1</td><td>P2_0</td></tr> <tr><td>AND 値</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td></tr> <tr><td>結果</td><td>P2_7</td><td>P2_6</td><td>P2_5</td><td>P2_4</td><td>P2_3</td><td>P2_2</td><td>0</td><td>P2_0</td></tr> </table> <p>168 行目で下表の計算を行い、P2_6 端子を"1"にします。</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr><th>bit</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr> <tr><td>元の値 (ポート2)</td><td>P2_7</td><td>P2_6</td><td>P2_5</td><td>P2_4</td><td>P2_3</td><td>P2_2</td><td>P2_1</td><td>P2_0</td></tr> <tr><td>OR 値</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>結果</td><td>P2_7</td><td>1</td><td>P2_5</td><td>P2_4</td><td>P2_3</td><td>P2_2</td><td>P2_1</td><td>P2_0</td></tr> </table> <p>169 行目で次の計算を行って、タイマ RD ジェネラルレジスタ D0(TRDGRD0)へ PWM 値を設定しています。小数点が出た場合は、切り捨てられます。</p> $ \begin{aligned} \text{TRDGRD0} &= (\text{PWM_CYCLE} - 1) \times \frac{\text{motor_l} (0\sim 100)}{100} \\ &= 39998 \times \frac{\text{motor_l} (0\sim 100)}{100} \end{aligned} $ <p>例えば motor_l=80 なら、TRDGRD0 は次のように計算されます。</p> $ \text{TRDGRD0} = 39998 \times 80 / 100 = 31998.4 = 31998 $	bit	7	6	5	4	3	2	1	0	元の値 (ポート2)	P2_7	P2_6	P2_5	P2_4	P2_3	P2_2	P2_1	P2_0	AND 値	1	1	1	1	1	1	0	1	結果	P2_7	P2_6	P2_5	P2_4	P2_3	P2_2	0	P2_0	bit	7	6	5	4	3	2	1	0	元の値 (ポート2)	P2_7	P2_6	P2_5	P2_4	P2_3	P2_2	P2_1	P2_0	OR 値	0	1	0	0	0	0	0	0	結果	P2_7	1	P2_5	P2_4	P2_3	P2_2	P2_1	P2_0
bit	7	6	5	4	3	2	1	0																																																																	
元の値 (ポート2)	P2_7	P2_6	P2_5	P2_4	P2_3	P2_2	P2_1	P2_0																																																																	
AND 値	1	1	1	1	1	1	0	1																																																																	
結果	P2_7	P2_6	P2_5	P2_4	P2_3	P2_2	0	P2_0																																																																	
bit	7	6	5	4	3	2	1	0																																																																	
元の値 (ポート2)	P2_7	P2_6	P2_5	P2_4	P2_3	P2_2	P2_1	P2_0																																																																	
OR 値	0	1	0	0	0	0	0	0																																																																	
結果	P2_7	1	P2_5	P2_4	P2_3	P2_2	P2_1	P2_0																																																																	

負の数なら 171～173 行を実行します。

P2_1="1"、P2_6="0"を設定し、P2_2 端子から PWM 出力すると、PWM の割合に応じてモータが逆転します。

171 行目で下表の計算を行い、P2_1 端子を"1"にします。

bit	7	6	5	4	3	2	1	0
元の値 (ポート2)	P2_7	P2_6	P2_5	P2_4	P2_3	P2_2	P2_1	P2_0
OR 値	0	0	0	0	0	0	1	0
結果	P2_7	P2_6	P2_5	P2_4	P2_3	P2_2	1	P2_0

172 行目で下表の計算を行い、P2_6 端子を"0"にします。

bit	7	6	5	4	3	2	1	0
元の値 (ポート2)	P2_7	P2_6	P2_5	P2_4	P2_3	P2_2	P2_1	P2_0
AND 値	1	0	1	1	1	1	1	1
結果	P2_7	0	P2_5	P2_4	P2_3	P2_2	P2_1	P2_0

171 行
～
173 行

173 行目で次の計算を行って、タイマ RD ジェネラルレジスタ D0(TRDGRD0)へ PWM 値を設定しています。小数点が出た場合は、切り捨てられます。

$$\begin{aligned} \text{TRDGRD0} &= (\text{PWM_CYCLE} - 1) \times \frac{-\text{motor_1} \ (-1 \sim -100)}{100} \\ &= 39998 \times \frac{-\text{motor_1} \ (-1 \sim -100)}{100} \end{aligned}$$

ポイントは、motor_1 変数が負の数ということです。回路的には P2_1="1"、P2_6="0"で逆転の設定になっています。そのため motor_1 は、正の数に直して計算します。直し方は、計算式の中で「-motor_1」としています。例えば motor_1=-50 なら、TRDGRD0 は次のように計算されます。

$$\text{TRDGRD0} = 39998 \times \{-(-50)\} / 100 = 39998 \times 50 / 100 = 19999$$

(4) 右モータ制御

右モータを制御する部分です。右モータの PWM は P2_4 端子から出力します。P2_4 端子から出力する PWM の設定は、タイマ RD ジェネラルレジスタ A1(TRDGRA1)に PWM 値を設定します。ただし、直接設定するのではなく、バッファレジスタであるタイマ RD ジェネラルレジスタ C1(TRDGRC1)に設定します。

```

176 :      /* 右モータ制御 */
177 :      if( motor_r >= 0 ) {
178 :          p2 &= 0xf7;          p2 = p2 AND 0xf7という意味です
179 :          p2 |= 0x80;          p2 = p2 OR 0x80という意味です
180 :          trdgrc1 = (long)( PWM_CYCLE - 1 ) * motor_r / 100;
181 :      } else {
182 :          p2 |= 0x08;          p2 = p2 OR 0x08という意味です
183 :          p2 &= 0x7f;          p2 = p2 AND 0x7fという意味です
184 :          trdgrc1 = (long)( PWM_CYCLE - 1 ) * ( -motor_r ) / 100;
185 :      }
    
```

177 行	<p>右モータの PWM 値の割合が正の数か負の数かをチェックします。 正の数なら 178～180 行、負の数なら 182～184 行を実行します。</p>																																																																								
178 行 ～ 180 行	<p>正の数なら 178～180 行を実行します。 P2_3="0"、P2_7="1"を設定し、P2_4 端子から PWM 出力すると、PWM の割合に応じてモータが正転します。 178 行目で下表の計算を行い、P2_3 端子を"0"にします。</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr><th>bit</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr> <tr><td>元の値 (ポート2)</td><td>P2_7</td><td>P2_6</td><td>P2_5</td><td>P2_4</td><td>P2_3</td><td>P2_2</td><td>P2_1</td><td>P2_0</td></tr> <tr><td>AND 値</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td></tr> <tr><td>結果</td><td>P2_7</td><td>P2_6</td><td>P2_5</td><td>P2_4</td><td>0</td><td>P2_2</td><td>P2_1</td><td>P2_0</td></tr> </table> <p>179 行目で下表の計算を行い、P2_7 端子を"1"にします。</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr><th>bit</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr> <tr><td>元の値 (ポート2)</td><td>P2_7</td><td>P2_6</td><td>P2_5</td><td>P2_4</td><td>P2_3</td><td>P2_2</td><td>P2_1</td><td>P2_0</td></tr> <tr><td>OR 値</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>結果</td><td>1</td><td>P2_6</td><td>P2_5</td><td>P2_4</td><td>P2_3</td><td>P2_2</td><td>P2_1</td><td>P2_0</td></tr> </table> <p>180 行目で次の計算を行って、タイマ RD ジェネラルレジスタ C1(TRDGRC1)へ PWM 値を設定しています。小数点が出た場合は、切り捨てられます。</p> $ \begin{aligned} \text{TRDGRC1} &= (\text{PWM_CYCLE} - 1) \times \frac{\text{motor_r} (0\sim 100)}{100} \\ &= 39998 \times \frac{\text{motor_r} (0\sim 100)}{100} \end{aligned} $ <p>例えば例えば motor_r=20 なら、TRDGRC1 は次のように計算されます。</p> $\text{TRDGRC1} = 39998 \times 20 / 100 = 7999.6 = 7999$	bit	7	6	5	4	3	2	1	0	元の値 (ポート2)	P2_7	P2_6	P2_5	P2_4	P2_3	P2_2	P2_1	P2_0	AND 値	1	1	1	1	0	1	1	1	結果	P2_7	P2_6	P2_5	P2_4	0	P2_2	P2_1	P2_0	bit	7	6	5	4	3	2	1	0	元の値 (ポート2)	P2_7	P2_6	P2_5	P2_4	P2_3	P2_2	P2_1	P2_0	OR 値	1	0	0	0	0	0	0	0	結果	1	P2_6	P2_5	P2_4	P2_3	P2_2	P2_1	P2_0
bit	7	6	5	4	3	2	1	0																																																																	
元の値 (ポート2)	P2_7	P2_6	P2_5	P2_4	P2_3	P2_2	P2_1	P2_0																																																																	
AND 値	1	1	1	1	0	1	1	1																																																																	
結果	P2_7	P2_6	P2_5	P2_4	0	P2_2	P2_1	P2_0																																																																	
bit	7	6	5	4	3	2	1	0																																																																	
元の値 (ポート2)	P2_7	P2_6	P2_5	P2_4	P2_3	P2_2	P2_1	P2_0																																																																	
OR 値	1	0	0	0	0	0	0	0																																																																	
結果	1	P2_6	P2_5	P2_4	P2_3	P2_2	P2_1	P2_0																																																																	

負の数なら 182~184 行を実行します。

P2_3="1"、P2_7="0"を設定し、P2_4 端子から PWM 出力すると、PWM の割合に応じてモータが逆転します。

182 行目で下表の計算を行い、P2_3 端子を"1"にします。

bit	7	6	5	4	3	2	1	0
元の値 (ポート 2)	P2_7	P2_6	P2_5	P2_4	P2_3	P2_2	P2_1	P2_0
OR 値	0	0	0	0	1	0	0	0
結果	P2_7	P2_6	P2_5	P2_4	1	P2_2	P2_1	P2_0

183 行目で下表の計算を行い、P2_7 端子を"0"にします。

bit	7	6	5	4	3	2	1	0
元の値 (ポート 2)	P2_7	P2_6	P2_5	P2_4	P2_3	P2_2	P2_1	P2_0
AND 値	0	1	1	1	1	1	1	1
結果	0	P2_6	P2_5	P2_4	P2_3	P2_2	P2_1	P2_0

182 行
~
184 行

184 行目で次の計算を行って、タイマ RD ジェネラルレジスタ C1(TRDGRC1)へ PWM 値を設定しています。小数点が出た場合は、切り捨てられます。

$$\begin{aligned}
 \text{TRDGRC1} &= (\text{PWM_CYCLE} - 1) \times \frac{-\text{motor_r} (-1 \sim -100)}{100} \\
 &= 39998 \times \frac{-\text{motor_r} (-1 \sim -100)}{100}
 \end{aligned}$$

ポイントは、motor_r 変数が負の数だということです。回路的には P2_3="1"、P2_7="0"で逆転の設定になっています。そのため motor_r は、正の数に直して計算します。直し方は、計算式の中で「-motor_r」としています。例えば motor_r=-90 なら、TRDGRC1 は次のように計算されます。

$$\text{TRDGRC1} = 39998 \times \{-(-90)\} / 100 = 39998 \times 90 / 100 = 35998.2 = 35998$$

23.6.3 main 関数

```

43 : void main( void )
44 : {
45 :     init();                /* 初期化                */
46 :     asm(" fset I ");      /* 全体の割り込み許可    */
47 :
48 :     while( 1 ) {
49 :         motor( 100 , 0);
50 :         timer( 1000 );
51 :         motor( 0, 80 );
52 :         timer( 1000 );
53 :         motor( -60, 0 );
54 :         timer( 1000 );
55 :         motor( 0, -40 );
56 :         timer( 1000 );
57 :         motor( 0, 0 );
58 :         timer( 1000 );
59 :     }
60 : }

```

45 行	init 関数を実行します。 init 関数では、ポートの入出力設定、タイマ RB による 1ms ごとの割り込み設定、タイマ RD によるリセット同期 PWM モードの設定を行っています。
46 行	全体の割り込みを許可する命令です。 今回は、タイマ RB による 1ms ごとの割り込みを許可するために実行しています。
48 行	while 文で無限ループを作っています。カッコでくくられた 49 行～58 行が実行され続けます。
49 行	motor 関数を使って、左モータ 100%、右モータ 0%で回転させます。ただし、実際のモータに出力される PWM 波形は、ディップスイッチの割合も加わります。
50 行	1000ms の時間、この行で待ちます。
51 行～ 58 行	同様にモータを制御します。

24. サーボの制御(プロジェクト:servo)

24.1 概要

本章では、ミニマイコンカーVer.2 でサーボ 1 個を制御する方法を紹介します。サーボの角度を±90 度、自由に変わることができます。サーボ制御は、タイマ RD によるリセット同期 PWM モードを使用します。

なお本章では、タイマ RB による割り込みを使っていますがここでは説明していません。タイマ RB による割り込みについては、「14. 割り込みによるタイマ(プロジェクト:timer2)」を参照してください。

※サーボ取り付けコネクタ、サーボはオプションです。

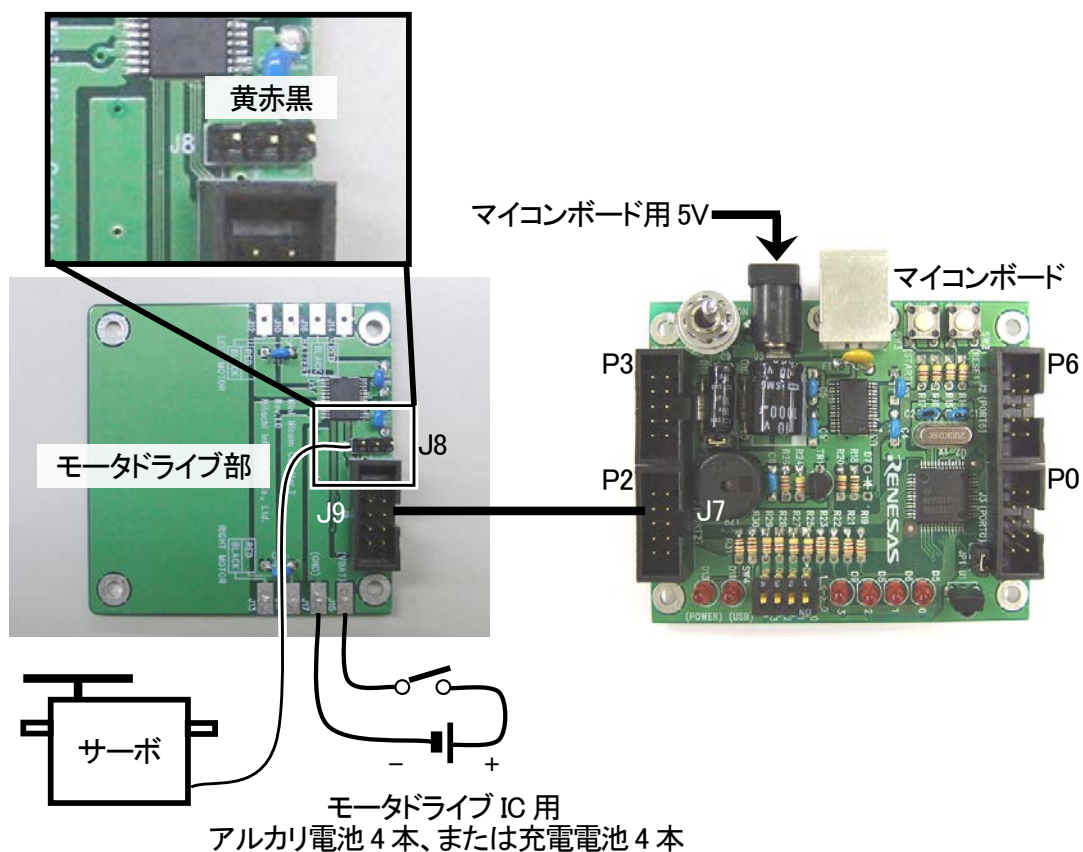
24.2 接続

■使用ポート

マイコンのポート	接続内容
P2 (J7)	ミニマイコンカーVer.2 のモータドライブ部を接続 モータドライブ部の J8 にはサーボを接続

■接続例

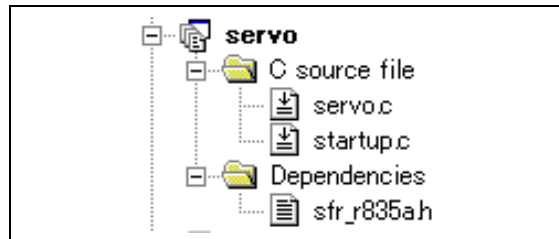
ミニマイコンカーVer.2 のマイコンボードとモータドライブ部を分離している場合の接続図を下記に示します。分離していない(購入時のまま)場合は、特に結線はありません。そのままの状態、本実習の演習ができます。モータドライブ部の J8 にサーボを接続します。J8 の 3 ピンコネクタ、サーボはオプションです。



■操作方法

操作は特にありません。電源を入れるとサーボが動き出します。サーボの動きをよく観察してください。

24.3 プロジェクトの構成



	ファイル名	内容
1	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。このファイルは共通で、どのプロジェクトもこのファイルから実行されます。
2	servo.c	実際に制御するプログラムが書かれています。R8C/35Aの内蔵周辺機能(SFR)の初期化も行います。
3	sfr_r835a.h	R8C/35Aマイコンの内蔵周辺機能を制御するためのレジスタ(Special Function Registers)を定義したファイルです。

24.4 プログラム「servo.c」

```

1 : /*****
2 : /* 対象マイコン R8C/35A */
3 : /* ファイル内容 ミニマイコンカーVer.2のサーボ制御 */
4 : /* バージョン Ver. 1.20 */
5 : /* Date 2010.04.19 */
6 : /* Copyright ルネサスマイコンカーラー事務局 */
7 : /* 日立インターメディアックス株式会社 */
8 : *****/
9 : /*
10 : 入力:マイコンボード上のディップスイッチ
11 : 出力:ミニマイコンカーVer.2のサーボ(オプション)
12 :
13 : ミニマイコンカーVer.2のサーボ(オプション)を制御します。
14 : */
15 :
16 : /*=====*/
17 : /* インクルード */
18 : /*=====*/
19 : #include "sfr_r835a.h" /* R8C/35A SFRの定義ファイル */
20 :
21 : /*=====*/
22 : /* シンボル定義 */
23 : /*=====*/
24 : #define PWM_CYCLE 39999 /* モータPWMの周期 */
25 : #define SERVO_CENTER 3750 /* サーボのセンタ値 */
26 : #define HANDLE_STEP 26 /* 1°分の値 */
27 :
28 : /*=====*/
29 : /* プロトタイプ宣言 */
30 : /*=====*/
31 : void init( void );
32 : void timer( unsigned long timer_set );
33 : void handle( int angle );
34 :
35 : /*=====*/
36 : /* グローバル変数の宣言 */
37 : /*=====*/
38 : unsigned long cnt_rb; /* タイマRB用 */
39 :

```

24. サーボの制御(プロジェクト:servo)

```

40 : /*****/
41 : /* メインプログラム */
42 : /*****/
43 : void main( void )
44 : {
45 :     init(); /* 初期化 */
46 :     asm(" fset I "); /* 全体の割り込み許可 */
47 :
48 :     while( 1 ) {
49 :         handle( 0 );
50 :         timer( 1000 );
51 :         handle( 30 );
52 :         timer( 1000 );
53 :         handle( 0 );
54 :         timer( 1000 );
55 :         handle( -30 );
56 :         timer( 1000 );
57 :     }
58 : }
59 :
60 : /*****/
61 : /* R8C/35A スペシャルファンクションレジスタ(SFR)の初期化 */
62 : /*****/
63 : void init( void )
64 : {
65 :     int i;
66 :
67 :     /* クロックをXINクロック(20MHz)に変更 */
68 :     prc0 = 1; /* プロテクト解除 */
69 :     cm13 = 1; /* P4_6, P4_7をXIN-XOUT端子にする */
70 :     cm05 = 0; /* XINクロック発振 */
71 :     for(i=0; i<50; i++ ); /* 安定するまで少し待つ(約10ms) */
72 :     ocd2 = 0; /* システムクロックをXINにする */
73 :     prc0 = 0; /* プロテクトON */
74 :
75 :     /* ポートの入出力設定 */
76 :     prc2 = 1; /* PD0のプロテクト解除 */
77 :     pd0 = 0xe0; /* 7-5:LED 4:MicroSW 3-0:Sensor */
78 :     p1 = 0xf; /* 3-0:LEDは消灯 */
79 :     pd1 = 0xdf; /* 5:RXD0 4:TXD0 3-0:LED */
80 :     pd2 = 0xfe; /* 7-1:モータドライブ部 0:PushSW */
81 :     pd3 = 0xfb; /* 4: buzzer 2: IR */
82 :     pd4 = 0x80; /* 7: XOUT 6: XIN 5-3: DIP SW 2: VREF */
83 :     pd5 = 0x40; /* 7: DIP SW */
84 :     pd6 = 0xff;
85 :
86 :     /* タイマRBの設定 */
87 :     /* 割り込み周期 = 1 / 20[MHz] * (TRBPRE+1) * (TRBPR+1)
88 :     = 1 / (20*10^6) * 200 * 100
89 :     = 0.001[s] = 1[ms]
90 :
91 :     /*
92 :     trbmr = 0x00; /* 動作モード、分周比設定 */
93 :     trbpre = 200-1; /* プリスケールレジスタ */
94 :     trbpr = 100-1; /* プライマリレジスタ */
95 :     trbic = 0x07; /* 割り込み優先レベル設定 */
96 :     trbcr = 0x01; /* カウント開始 */
97 :
98 :     /* タイマRD リセット同期PWMモードの設定 */
99 :     /* PWM周期 = 1 / 20[MHz] * カウントソース * (TRDGRA0+1)
100 :     = 1 / (20*10^6) * 8 * 40000
101 :     = 0.016[s] = 16[ms]
102 :
103 :     /*
104 :     trdfcr = 0x01; /* リセット同期PWMモードに設定 */
105 :     trdmr = 0xf0; /* バッファレジスタ設定 */
106 :     trdoer1 = 0xcd; /* 出力端子の選択 */
107 :     trdpsr0 = 0x08; /* TRDIOB0, C0, D0端子設定 */
108 :     trdpsr1 = 0x05; /* TRDIOA1, B1, C1, D1端子設定 */
109 :     trdcr0 = 0x23; /* ソースカウントの選択:f8 */
110 :     trdgra0 = trdgrc0 = PWM_CYCLE; /* 周期 */
111 :     trdgrb0 = trdgrd0 = 0; /* P2_2端子のON幅設定 */
112 :     trdgra1 = trdgrc1 = 0; /* P2_4端子のON幅設定 */
113 :     trdgrb1 = trdgrd1 = SERVO_CENTER; /* P2_5端子のON幅設定 */
114 :     trdstr = 0x0d; /* TRD0カウント開始 */
115 : }
116 :
117 : /*****/
118 : /* ディップスイッチ値読み込み */
119 : /* 戻り値 スイッチ値 0~15 */
120 : /*****/
121 : unsigned char dipsw_get( void )
122 : {
123 :     unsigned char sw, sw1, sw2;
124 :
125 :     sw1 = (p5>>4) & 0x08; /* ディップスイッチ読み込み3 */
126 :     sw2 = (p4>>3) & 0x07; /* ディップスイッチ読み込み2, 1, 0 */
127 :     sw = sw1 | sw2; /* P5とP4の値を合わせる */
128 :
129 :     return sw;

```


24. サーボの制御(プロジェクト:servo)

```

130 : /*****/
131 : /* タイマ本体 */
132 : /* 引数 タイマ値 1=1ms */
133 : /*****/
134 : void timer( unsigned long timer_set )
135 : {
136 :     cnt_rb = 0;
137 :     while( cnt_rb < timer_set );
138 : }
139 :
140 : /*****/
141 : /* タイマRB 割り込み処理 */
142 : /*****/
143 : #pragma interrupt intTRB(vect=24)
144 : void intTRB( void )
145 : {
146 :     cnt_rb++;
147 : }
148 :
149 : /*****/
150 : /* サーボハンドル操作 */
151 : /* 引数 サーボ操作角度：-90~90 */
152 : /* -90で左へ90度、0でまっすぐ、90で右へ90度回転 */
153 : /* 戻り値 なし */
154 : /*****/
155 : void handle( int angle )
156 : {
157 :     /* サーボが左右逆に動く場合は、「-」を「+」に替えてください */
158 :     trdgrd1 = SERVO_CENTER - angle * HANDLE_STEP;
159 : }
160 :
161 : /*****/
162 : /* end of file */
163 : /*****/

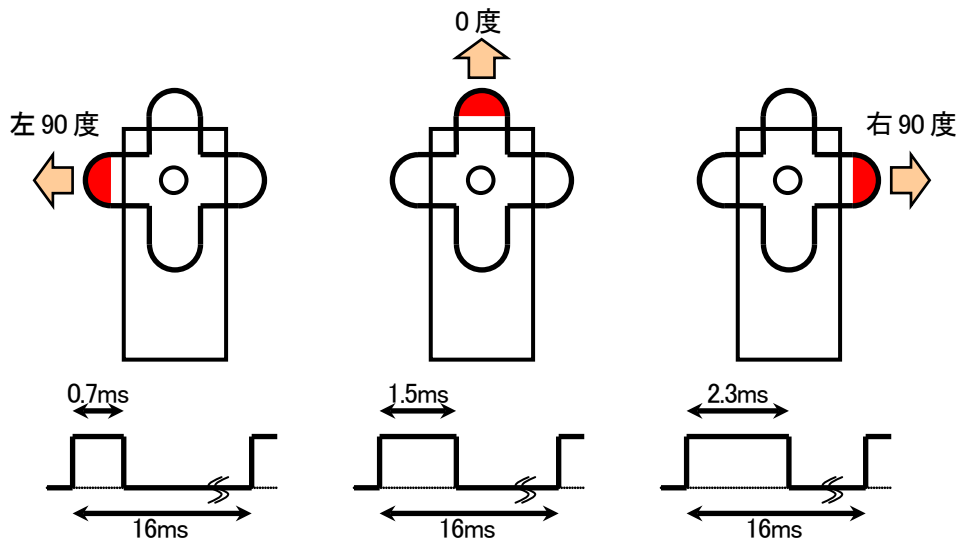
```

24.5 サーボ

24.5.1 サーボの接続

サーボは周期 16[ms]のパルスを加え、そのパルスの ON 幅でサーボの角度が決まります。

サーボの回転角度とパルスの ON 幅の関係は、サーボのメーカーや個体差によって多少の違いがありますが、ほとんどが下図のような関係です。



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

リセット同期 PWM モードで下記のような PWM 信号を出力して、サーボを制御します。

- ・周期 16[ms]
- ・ON 幅 0.7~2.3[ms]

24.5.2 J7 と J9 の詳細

J7 と J9 のピン番号に対する信号名を下表に示します。J7 と J9 はあらかじめパターンで接続されています。マイコン部とモータドライブ部を分離した場合は、フラットケーブルなどで J7 と J9 を接続してください。

サーボを制御する PWM 波形は、P2.5 端子から出力します。プログラムで P2.5 端子を PWM 端子にします。モータ制御用の PWM 端子は、P2.2 端子、P2.4 端子です。その他は I/O 端子に設定します。PWM の周期は 16ms にします。

J9		J7 (マイコン部)			
ピン番号	信号名	ピン番号	信号名	入出力設定	説明
1	VCC(+5V)	1	VCC(+5V)		
2	モータ右 2	2	P2_7	出力	P2_7 は通常の I/O ポートです。
3	モータ左 2	3	P2_6	出力	P2_6 は通常の I/O ポートです。
4	サーボ (オプション)	4	P2_5	出力 (PWM 波形出力)	この端子は PWM 出力許可にします。TRDGRD1 で ON 幅を設定します。
5	モータ右 PWM	5	P2_4	出力 (PWM 波形出力)	この端子は PWM 出力許可にします。TRDGRC1 で ON 幅を設定します。
6	モータ右 1	6	P2_3	出力	P2_3 は通常の I/O ポートです。
7	モータ左 PWM	7	P2_2	出力 (PWM 波形出力)	この端子は PWM 出力許可にします。TRDGRD0 で ON 幅を設定します。
8	モータ左 1	8	P2_1	出力	P2_1 は通常の I/O ポートです。
9	未接続	9	P2_0	入力	P2_0 は通常の I/O ポートです。マイコンボード上のタクトスイッチに繋がっています。
10	GND	10	GND		

↑パターン上で接続されています。

24.6 プログラムの解説

24.6.1 init 関数(タイマ RD の設定)

```

21 : /*=====*/
22 : /* シンボル定義 */
23 : /*=====*/
24 : #define PWM_CYCLE      39999          /* モータPWMの周期 */
25 : #define SERVO_CENTER   3749          /* サーボのセンタ値 */
26 : #define HANDLE_STEP    22           /* 1° 分の値 */

中略

102 :      trdfcr = 0x01;                  /* リセット同期PWMモードに設定 */
103 :      trdmr = 0xf0;                   /* バッファレジスタ設定 */
104 :      trdoer1 = 0xcd;                 /* 出力端子の選択 */
105 :      trdpsr0 = 0x08;                 /* TRDIOB0, C0, D0端子設定 */
106 :      trdpsr1 = 0x05;                 /* TRDIOA1, B1, C1, D1端子設定 */
107 :      trdcr0 = 0x23;                 /* ソースカウントの選択:f8 */
108 :      trdgra0 = trdgrc0 = PWM_CYCLE;  /* 周期 */
109 :      trdgrb0 = trdgrd0 = 0;          /* P2_2端子のON幅設定 */
110 :      trdgra1 = trdgrc1 = 0;          /* P2_4端子のON幅設定 */
111 :      trdgrb1 = trdgrd1 = SERVO_CENTER; /* P2_5端子のON幅設定 */
112 :      trdstr = 0x0d;                 /* TRD0カウント開始 */
113 : }

```

24 行	<p>PWM 周期を「PWM_CYCLE」という名前で定義します。今回、PWM 周期は 16ms にします。タイマ RD 制御レジスタ 0(TRDCR0)のカウントソース選択ビットで、タイマ RD カウンタ 0(TRD0)がカウントアップする時間を 400ns に設定しています。</p> <p>よって、PWM 周期を設定するタイマ RD ジェネラルレジスタ A0(TRDGRA0)に設定する値は下記ようになります。</p> $\begin{aligned} \text{TRDGRA0} &= \text{PWM 周期} / \text{タイマ RD カウンタ 0 のカウントソース} - 1 \\ &= (16 \times 10^{-3}) / (400 \times 10^{-9}) - 1 \\ &= 40000 - 1 = 39999 \end{aligned}$ <p>よって、PWM_CYCLE を 39999 として定義します。PWM 周期を変更する場合は、この値を変更してください。</p>
25 行	<p>サーボが 0 度(まっすぐ向く角度)のときの値を「SERVO_CENTER」という名前で定義します。今回、PWM の ON 幅を 1.5ms にします。</p> <p>P2.5 端子の PWM の ON 幅を設定するタイマ RD ジェネラルレジスタ B1(TRDGRB1)に設定する値は下記ようになります。</p> $\begin{aligned} \text{TRDGRB1} &= \text{P2.5 端子から出力される PWM 波形の ON 幅} / \text{タイマ RD カウンタ 0 のカウントソース} - 1 \\ &= (1.5 \times 10^{-3}) / (400 \times 10^{-9}) - 1 \\ &= 3750 - 1 = 3749 \end{aligned}$ <p>よって、SERVO_CENTER を 3749 として定義します。実際はサーボに個体差があり、3749 がまっすぐ向く場合はほとんどありません。サーボのセンタ値を調整する場合は、この値を変更してください。</p>

24. サーボの制御(プロジェクト:servo)

<p>26 行</p>	<p>サーボが 1 度分動く値を「HANDLE_STEP」という名前で定義します。 左 90 度の PWM の ON 幅は、0.7ms です。右 90 度の PWM の ON 幅は、2.3ms です。この差分を 180 で割ると、1 度当たりの値が計算できます。</p> <ul style="list-style-type: none"> • 左 90 度の PWM の ON 幅 $\begin{aligned} \text{TRDGRB1} &= \text{P2.5 端子から出力される PWM 波形の ON 幅} / \text{タイマ RD カウンタ 0 のカウントソース} - 1 \\ &= (0.7 \times 10^{-3}) / (400 \times 10^{-9}) - 1 \\ &= 1750 - 1 = 1749 \end{aligned}$ • 右 90 度の PWM の ON 幅 $\begin{aligned} \text{TRDGRB1} &= \text{P2.5 端子から出力される PWM 波形の ON 幅} / \text{タイマ RD カウンタ 0 のカウントソース} - 1 \\ &= (2.3 \times 10^{-3}) / (400 \times 10^{-9}) - 1 \\ &= 5750 - 1 = 5749 \end{aligned}$ • 1 度当たりの値 $(\text{右} - \text{左}) / 180 = (5749 - 1749) / 180 = 22.22 \approx 22$ <p>よって、HANDLE_STEP を 22 として定義します。1 度当たりの値を変更する場合は、この値を変更してください。</p>
<p>102～ 112 行</p>	<p>タイマ RD をリセット同期 PWM モードで動作するように設定しています。 詳しくは、プロジェクト「timer_rd_doukipwm」、プロジェクト「motor」の説明を参照してください。</p> <ul style="list-style-type: none"> • 右モータのスピード制御 右モータは、P2_4 端子から PWM 波形を出力してスピード制御します。今回は、右モータは回しませんので、PWM は 0%にします。TRDGRA1 には、0 を設定します。 • 左モータのスピード制御 左モータは、P2_2 端子から PWM 波形を出力してスピード制御します。今回は、左モータは回しませんので、PWM は 0%にします。TRDGRB0 には、0 を設定します。 • サーボの角度制御 サーボは、P2_5 端子から PWM 波形を出力して角度制御します。初期値は 0 度になるように設定します。TRDGRB1 には、「SERVO_CENTER」の値を設定します。

24.6.2 handle 関数

サーボへPWMを出力し、サーボの角度を制御する関数です。

```

149 : /*****/
150 : /* サーボハンドル操作 */
151 : /* 引数   サーボ操作角度：-90～90 */
152 : /*      -90で左へ90度、0でまっすぐ、90で右へ90度回転 */
153 : /* 戻り値 なし */
154 : /*****/
155 : void handle( int angle )
156 : {
157 :     /* サーボが左右逆に動く場合は、「-」を「+」に替えてください */
158 :     trdgrd1 = SERVO_CENTER - angle * HANDLE_STEP;
159 : }
    
```

(1) handle 関数の使い方

handle 関数の使い方を下記に示します。

```
handle( サーボの角度 );
```

引数は、サーボの角度を設定します。値とサーボの角度の関係を下記に示します。

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

プログラム例を、下記に示します。

```

handle( 0 );           0 度
handle( 30 );         右 30 度
handle( -45 );        左 45 度
    
```

(2) プログラムの内容

```
158 :      trdgrd1 = SERVO_CENTER - angle * HANDLE_STEP;
           ①           ②           ③           ④
```

①	サーボに接続されている P2.5 端子の PWM の ON 幅を設定するのは、TRDGRB1 です。ただ、あるタイミングで TRDGRB1 の値を書き換えると PWM 波形が 100%出力になることがあるので、バッファレジスタを使います。TRDGRB1 のバッファレジスタは TRDGRD1 になります。今回は、このレジスタに PWM 値を設定します。
②	0 度のときの値です。
③	handle 関数で指定した角度が代入されている変数です。
④	1 度当たりの増分です。

TRDGRD1 に代入される値の計算例を下記に示します。

※SERVO_CENTER=3749、HANDLE_STEP=22 とします。

• 0 度のとき

$$\begin{aligned} \text{TRDGRD1} &= \text{SERVO_CENTER} - \text{angle} * \text{HANDLE_STEP} \\ &= 3749 - \boxed{0} * 22 \\ &= 3749 \end{aligned}$$

• 30 度のとき

$$\begin{aligned} \text{TRDGRD1} &= \text{SERVO_CENTER} - \text{angle} * \text{HANDLE_STEP} \\ &= 3749 - \boxed{30} * 22 \\ &= 3749 - 660 \\ &= 3089 \end{aligned}$$

• -45 度のとき

$$\begin{aligned} \text{TRDGRD1} &= \text{SERVO_CENTER} - \text{angle} * \text{HANDLE_STEP} \\ &= 3749 - \boxed{(-45)} * 22 \\ &= 3749 - (-990) \\ &= 4739 \end{aligned}$$

24.6.3 main 関数

```

43 : void main( void )
44 : {
45 :     init();                /* 初期化                */
46 :     asm(" fset I ");      /* 全体の割り込み許可   */
47 :
48 :     while( 1 ) {
49 :         handle( 0 );
50 :         timer( 1000 );
51 :         handle( 30 );
52 :         timer( 1000 );
53 :         handle( 0 );
54 :         timer( 1000 );
55 :         handle( -30 );
56 :         timer( 1000 );
57 :     }
58 : }

```

45 行	init 関数を実行します。 init 関数では、ポートの入出力設定、タイマ RB による 1ms ごとの割り込み設定、タイマ RD によるリセット同期 PWM モードの設定を行っています。
46 行	全体の割り込みを許可する命令です。 今回は、タイマ RB による 1ms ごとの割り込みを許可するために実行しています。
48 行	while 文で無限ループを作っています。カッコでくくられた 49 行～56 行が実行され続けます。
49 行	handle 関数を使って、サーボを 0 度にします。
50 行	1000ms の時間、この行で待ちます。
51 行～ 56 行	同様にサーボを制御します。

25. 通信(プロジェクト:uart0)

25.1 概要

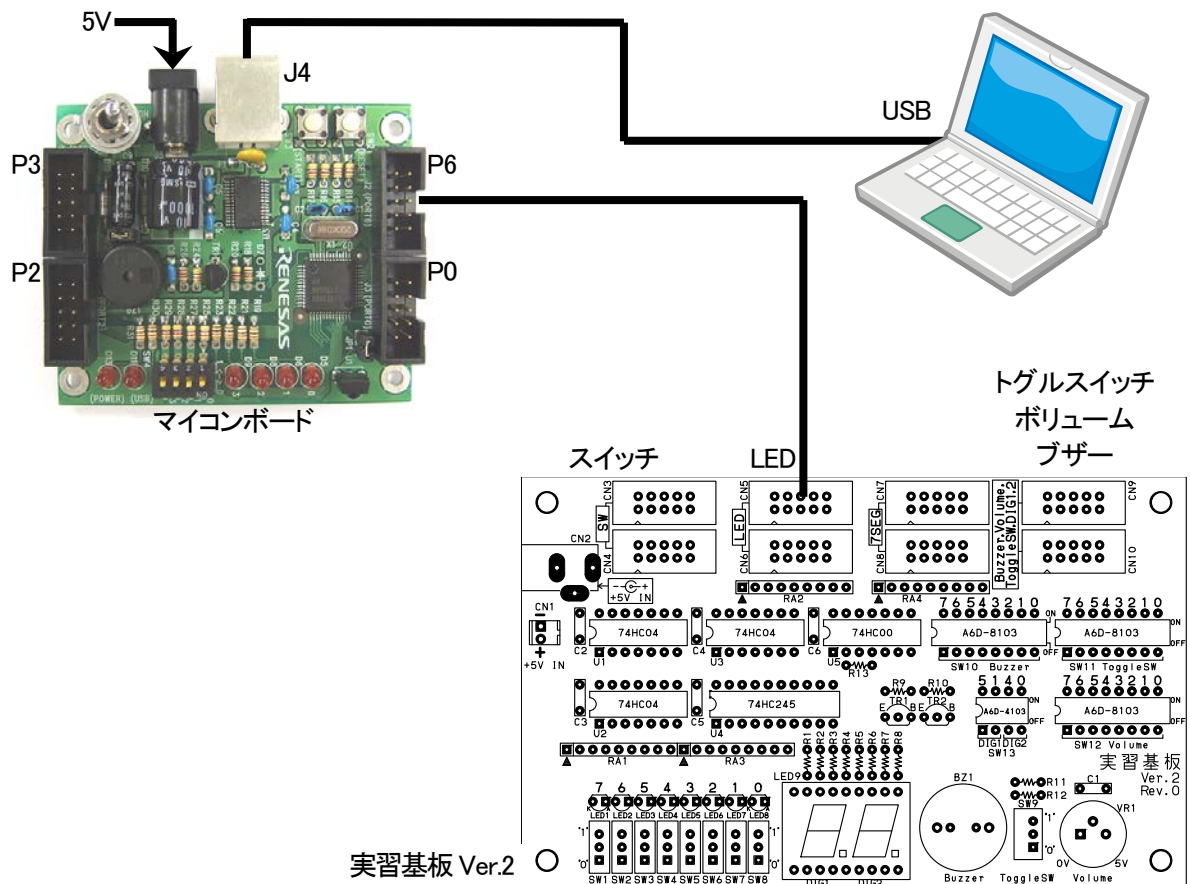
本章では、マイコンとパソコンで RS-232C 通信を行う方法を紹介します。通信は、マイコン内蔵機能の UART0 を使います。シリアル通信についても解説しています。

25.2 接続

■使用ポート

マイコンのポート	接続内容
J4 (USB コネクタ)	USB コネクタを通して、パソコン(通信ソフト)のキーボードから打ち込んだ文字コードを LED へ出力します。
P6 (J2)	実習基板 Ver.2 の LED 部など、出力機器を接続します。

■接続例

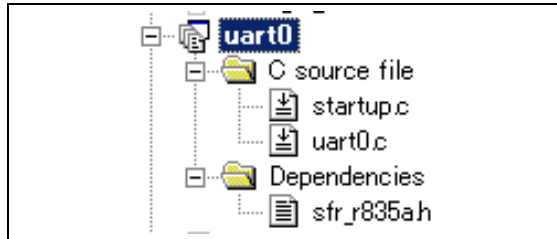


25. 通信(プロジェクト:uart0)

■操作方法

パソコン側は TeraTerm やハイパーターミナルなどの通信ソフトを使います。キーボードから打ち込んだ文字コードが、ポート6に接続されているLEDに出力されます。また、マイコンで受信した文字コードをそのままパソコンに送信し、通信ソフトの画面上に表示されます。パソコン側の設定など実習方法は、「25.7 実習手順」を参照してください。

25.3 プロジェクトの構成



	ファイル名	内容
1	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。このファイルは共通で、どのプロジェクトもこのファイルから実行されます。
2	uart0.c	実際に制御するプログラムが書かれています。R8C/35A の内蔵周辺機能(SFR)の初期化も行います。
3	sfr_r835a.h	R8C/35A マイコンの内蔵周辺機能を制御するためのレジスタ(Special Function Registers)を定義したファイルです。

25.4 プログラム「uart0.c」

```

1 : /******
2 : /* 対象マイコン R8C/35A */
3 : /* ファイル内容 UART0の使用例 */
4 : /* バージョン Ver. 1.20 */
5 : /* Date 2010.04.19 */
6 : /* Copyright ルネサスマイコンカーラー事務局 */
7 : /* 日立インターメディックス株式会社 */
8 : /******
9 : /*
10 : 入力 : UART0(パソコンのキーボードで入力したデータ)
11 : ※パソコンはTeraTermProなどの通信ソフトを使用します
12 : 出力 : UART0(通信ソフトの画面)
13 :
14 : RS232Cから出力されたデータをUART0で受信、そのままUART0から出力します。
15 : TeraTermProなどの通信ソフトを通して、キーボードから入力した文字が
16 : 通信ソフトの画面上にそのまま表示されます。
17 : */
18 :
19 : /*=====*/
20 : /* インクルード */
21 : /*=====*/
22 : #include "sfr_r835a.h" /* R8C/35A SFRの定義ファイル */
23 : #include <stdio.h>
24 :
25 : /*=====*/
26 : /* シンボル定義 */
27 : /*=====*/
28 :
29 : /*=====*/
30 : /* プロトタイプ宣言 */
31 : /*=====*/
32 : void init( void );
33 : int get_uart0( unsigned char *s );
34 : int put_uart0( unsigned char r );
35 :

```

25. 通信(プロジェクト:uart0)

```

36 : /*****/
37 : /* メインプログラム */
38 : /*****/
39 : void main( void )
40 : {
41 :     unsigned char  d;
42 :     int            ret;
43 :
44 :     init();          /* SFRの初期化 */
45 :
46 :     while( 1 ) {
47 :         ret = get_uart0( &d );
48 :         p6 = d;
49 :         if( ret == 1 ) put_uart0( d );
50 :     }
51 : }
52 :
53 : /*****/
54 : /* R8C/35A スペシャルファンクションレジスタ(SFR)の初期化 */
55 : /*****/
56 : void init( void )
57 : {
58 :     int i;
59 :
60 :     /* クロックをXINクロック(20MHz)に変更 */
61 :     prc0 = 1;          /* プロテクト解除 */
62 :     cm13 = 1;         /* P4_6, P4_7をXIN-XOUT端子にする */
63 :     cm05 = 0;         /* XINクロック発振 */
64 :     for(i=0; i<50; i++ ); /* 安定するまで少し待つ(約10ms) */
65 :     ocd2 = 0;         /* システムクロックをXINにする */
66 :     prc0 = 0;         /* プロテクトON */
67 :
68 :     /* ポートの入出力設定 */
69 :     prc2 = 1;          /* PDOのプロテクト解除 */
70 :     pd0 = 0xe0;       /* 7-5:LED 4:MicroSW 3-0:Sensor */
71 :     p1 = 0x0f;        /* 3-0:LEDは消灯 */
72 :     pd1 = 0xdf;       /* 5:RXD0 4:TXD0 3-0:LED */
73 :     pd2 = 0xfe;       /* 0:PushSW */
74 :     pd3 = 0xfb;       /* 4:Buzzer 2:IR */
75 :     pd4 = 0x83;       /* 7:XOUT 6:XIN 5-3:DIP SW 2:VREF */
76 :     pd5 = 0x40;       /* 7:DIP SW */
77 :     pd6 = 0xff;       /* LEDなど出力 */
78 :
79 :     /* UART0の設定 */
80 :     /* U0BRG = カウントソースの周波数/(設定したいbps*16)-1
81 :                = 20MHz / ( 9600 *16)-1
82 :                = 129.208 = 129
83 :     */
84 :     u0sr = 0x05;      /* P14=TXD0, P15=RXD0に設定 */
85 :     u0c0 = 0x00;      /* カウントソースなどの設定 */
86 :     u0c1 = 0x05;      /* 送信,受信許可 */
87 :     u0brg = 129;      /* 通信速度=9600pbs */
88 :     u0mr = 0x05;      /* UART0 データ長8bit 1ストップビット */
89 : }
90 :
91 : /*****/
92 : /* 1文字受信 */
93 : /* 引数 受信文字格納アドレス */
94 : /* 戻り値 -1:受信エラー 0:受信なし 1:受信あり 文字は*sに格納 */
95 : /*****/
96 : int get_uart0( unsigned char *s )
97 : {
98 :     int ret = 0, i;
99 :     unsigned int data;
100 :
101 :     if (ri_u0c1 == 1){ /* 受信データあり? */
102 :         data = u0rb;
103 :         *s = (unsigned char)data;
104 :         ret = 1;
105 :         if( data & 0xf000 ) { /* エラーあり? */
106 :             /* エラー時は再設定 */
107 :             re_u0c1 = 0;
108 :             for( i=0; i<50; i++ );
109 :             re_u0c1 = 1;
110 :
111 :             ret = -1;
112 :         }
113 :     }
114 :     return ret;
115 : }
116 :

```

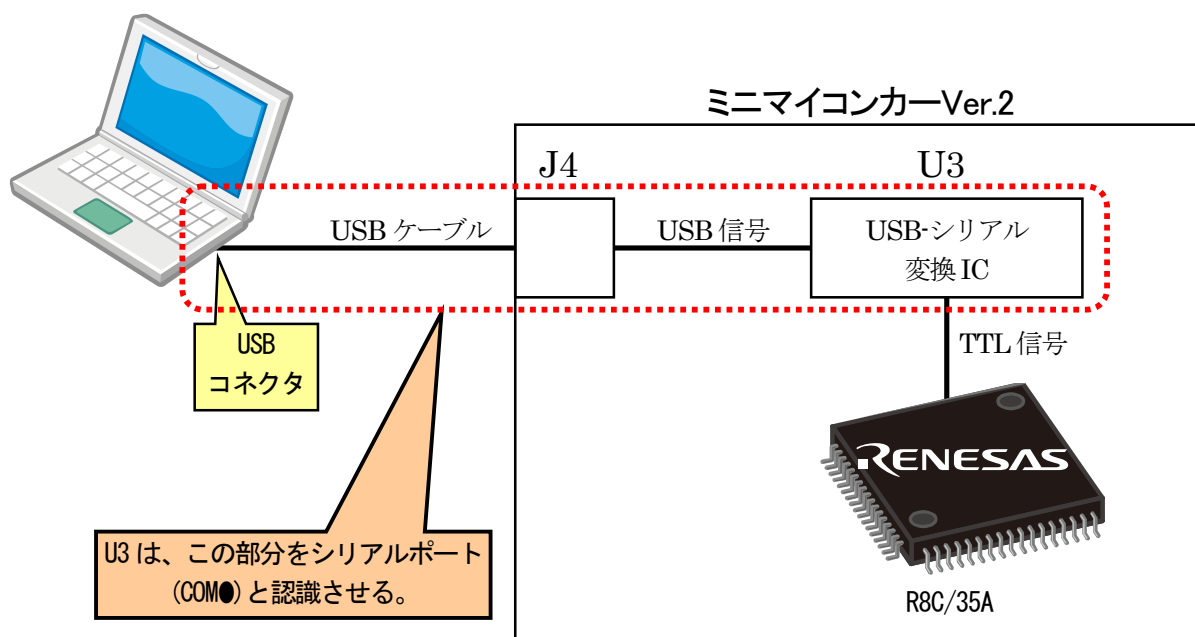
25. 通信(プロジェクト:uart0)

```
117 : /******  
118 : /* 1文字出力 */  
119 : /* 引数 送信データ */  
120 : /* 戻り値 0:送信中のため、送信できず 1:送信セット完了 */  
121 : /******  
122 : int put_uart0( unsigned char r )  
123 : {  
124 :     if(ti_u0c1 == 1) { /* 送信データなし? */  
125 :         u0tbl = r;  
126 :         return 1;  
127 :     } else {  
128 :         /* 先に送信中(今回のデータは送信せずに終了) */  
129 :         return 0;  
130 :     }  
131 : }  
132 :  
133 : /******  
134 : /* end of file */  
135 : /******
```

25.5 パソコンとミニマイコンカーVer.2 の通信

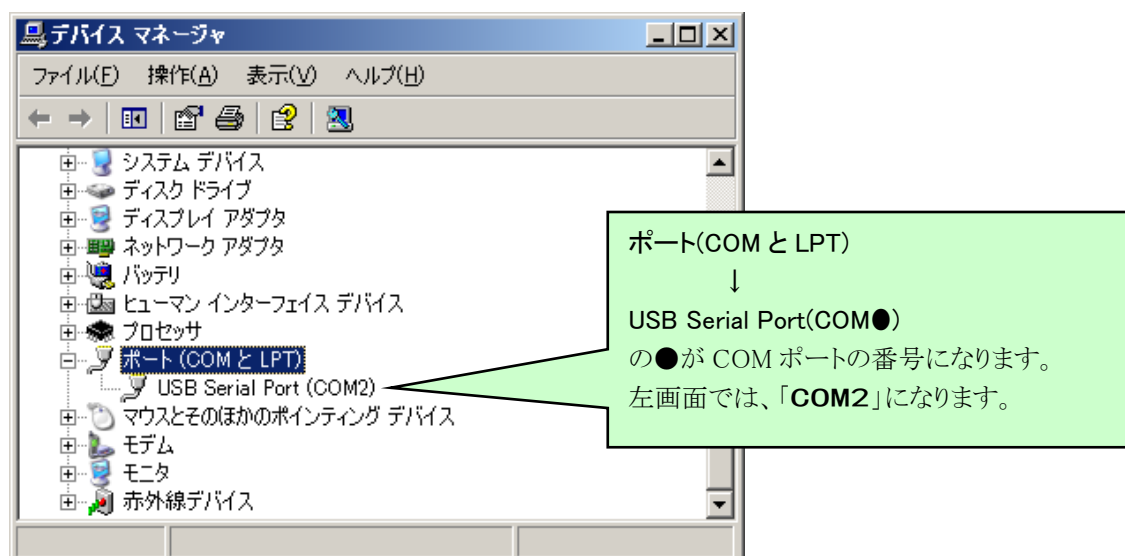
25.5.1 ミニマイコンカーVer.2 の通信方法

ミニマイコンカーVer.2 のマイコンボードは、パソコンとの通信を USB コネクタを通して行います。ミニマイコンカーVer.2 には、USB-シリアル変換 IC(U3)が搭載されており、パソコン(TeraTerm やハイパーターミナルなどの通信ソフト)はミニマイコンカーVer.2 を COM ポート(シリアルポート)として認識します。



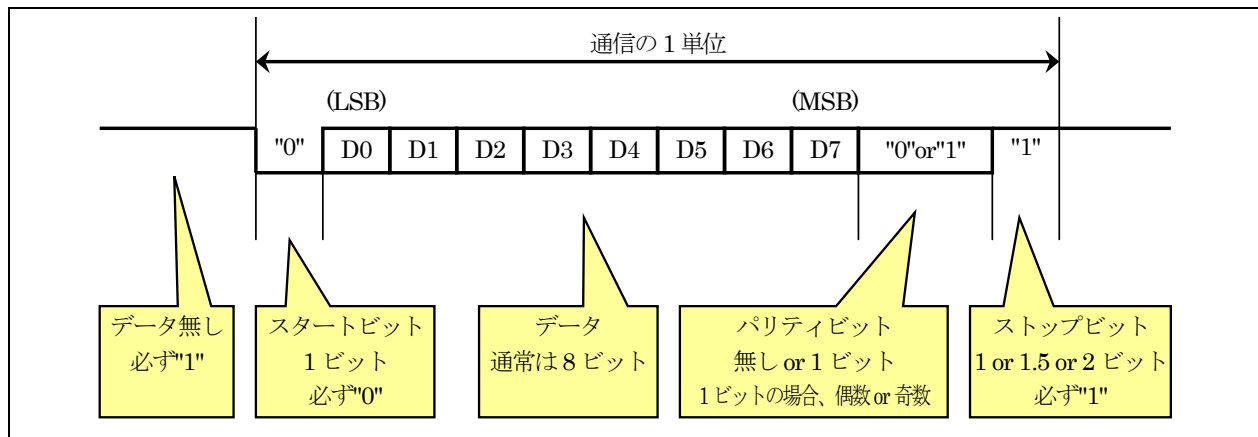
25.5.2 COM ポートの確認

ミニマイコンカーVer.2 をパソコンの USB に接続した状態で「コントロールパネル→システム」を選択、システムのプロパティを開きます。「ハードウェア」タブを選択し、「デバイスマネージャ」をクリックします。



25.5.3 シリアル通信の設定

1 文字のデータを送る場合、文字データだけ送るわけではありません。受信側で正しく受け取るために送信側は、「スタートビット、データ、パリティビット、ストップビット」という形で送ります(下図)。



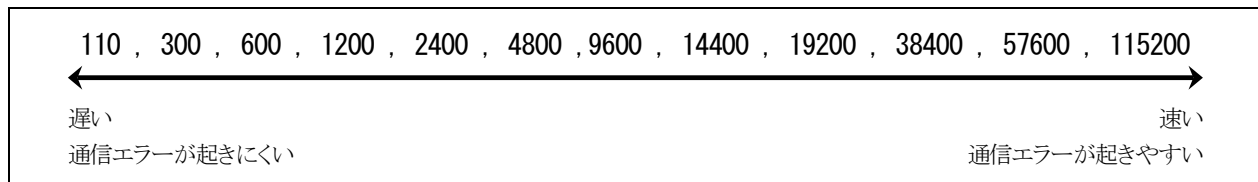
※LSB…Least Significant Bit(最下位ビット)のことです。

※MSB…Most Significant Bit(最上位ビット)のことです。

(1) 通信速度(ボーレート)

1 秒間に何ビット分のデータを送るか設定します。単位は、「Bits Per Second」で略して「bps(ビーピーエス)」です。

例えば 9600bps は、1 秒間に 9600 個のビットを送ることができます。TeraTerm で設定できる通信速度と特徴を下記に示します。



(2) ストップビット

ストップビットは必ず"0"です。ちなみに、データが無いときは通信線は"1"になっています。"1"→"0"になると通信スタートと判断して、設定されている通信速度でデータを送信、または受信します。

(3) データ長

データのビット数を設定します。7ビット、8ビット、9ビットなどがあります。1文字=8ビットなので、8ビットにすることがほとんどです。

(4) パリティビット

パリティビットは、データが正しいかどうか誤り検出をするためのビットです。パリティ(parity)とは、「等しいこと」です。パリティビットは、無し/偶数パリティ/奇数パリティがあり、無しのときは、このビット自体がありません。

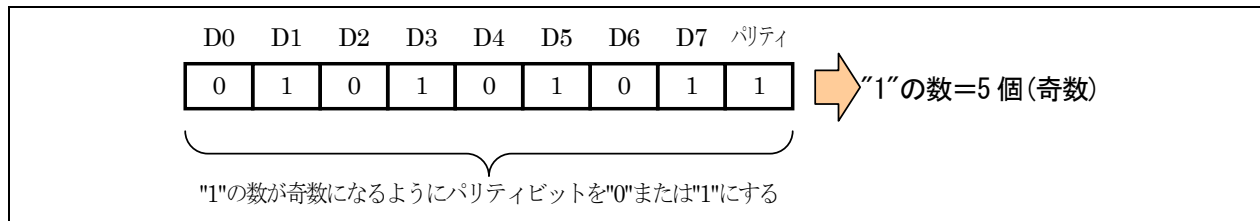
●偶数パリティに設定した場合

データ8ビット分+パリティビットの各ビットの"1"の数が偶数になるようにパリティビットを設定して、送信します。受信側にも、"1"の数が偶数と設定しておけば、偶数ならエラー無し、奇数ならエラーと判断できます。エラーなら、パリティエラーとなります。



●奇数パリティに設定した場合

データ8ビット分+パリティビットの各ビットの"1"の数が奇数になるようにパリティビットを設定して、送信します。受信側にも、"1"の数が奇数と設定しておけば、奇数ならエラー無し、偶数ならエラーと判断できます。エラーなら、パリティエラーとなります。



(5) ストップビット

ストップビットは必ず"1"です。1文字の通信が終わったことを示します。ビット幅は1ビット、1.5ビット、2ビットなどがあります。通常は1ビットです。受信側でストップビットを検出できない場合は、フレーミングエラーとなります。

(6) フロー制御

フロー制御とは、受信側で受信処理が間に合わない、または受信準備が整っていないときに、送信側に送信を待つように知らせ、受信準備ができたなら送信を行うように知らせる機能です。

フロー制御を行うには、送信側、受信側にフロー制御を行うプログラムが入っていないとけません。今回の演習ではフロー制御は行いません。

(7) 今回の演習の設定

今回の演習の通信設定を下記に示します。送信側、受信側の両方で設定します。

項目	設定内容
通信速度(ボーレート)	9600bps
データ長	8ビット
パリティビット	無し
ストップビット	1ビット
フロー制御	無し

25.6 プログラムの解説

25.6.1 init 関数(UART0 の設定)

シリアルインターフェース 0(UART0)の設定プログラムは、次のようになります。

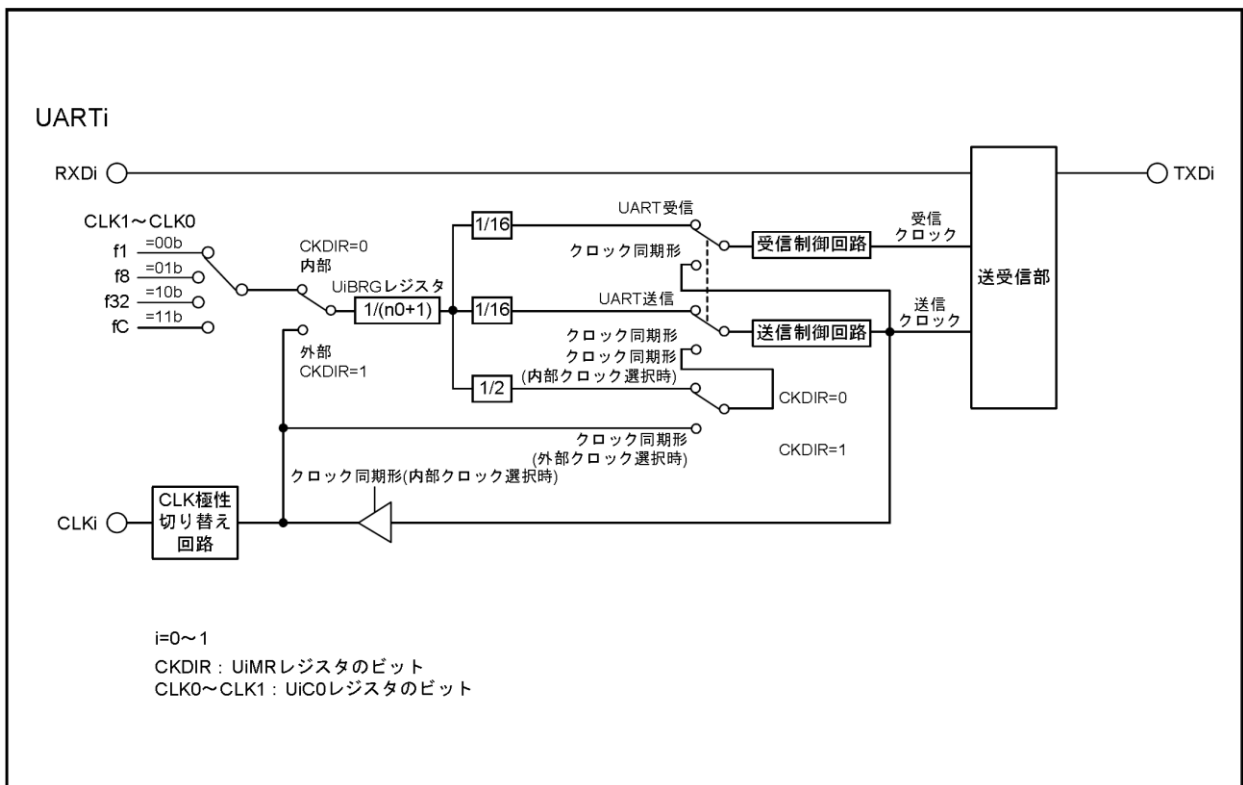
```

79 :      /* UART0の設定 */
80 :      /* U0BRG = カウントソースの周波数/(設定したいbps*16)-1
81 :          = 20MHz          /(    9600    *16)-1
82 :          = 129.208 = 129
83 :      */
84 :      u0sr = 0x05;          /* P14=TXD0, P15=RXD0に設定      */
85 :      u0c0 = 0x00;          /* カウントソースなどの設定      */
86 :      u0c1 = 0x05;          /* 送信、受信許可                  */
87 :      u0brg = 129;          /* 通信速度=9600pbs                */
88 :      u0mr = 0x05;          /* UART0 データ長8bit 1ストップビット */
    
```

(1) シリアルインターフェース

R8C/35Aには、シリアルインタフェース(UART)がUART0～UART2の3チャンネルあります。本プロジェクトでは、シリアルインタフェース 0(UART0)を使って実習します。ちなみに UART とは、「Universal Asynchronous Receiver Transmitter」の略称で、訳は特に決まっていますが「調歩同期方式によるシリアル信号を行うための集積回路」というような訳し方をすることが多いです。

シリアルインタフェースとは、1本の送信線、1本の受信線でデータのやり取りを行う通信方式です。RS-232Cは、シリアルインタフェースのひとつです。下記に、UARTi (i=0～1)のブロック図を示します。



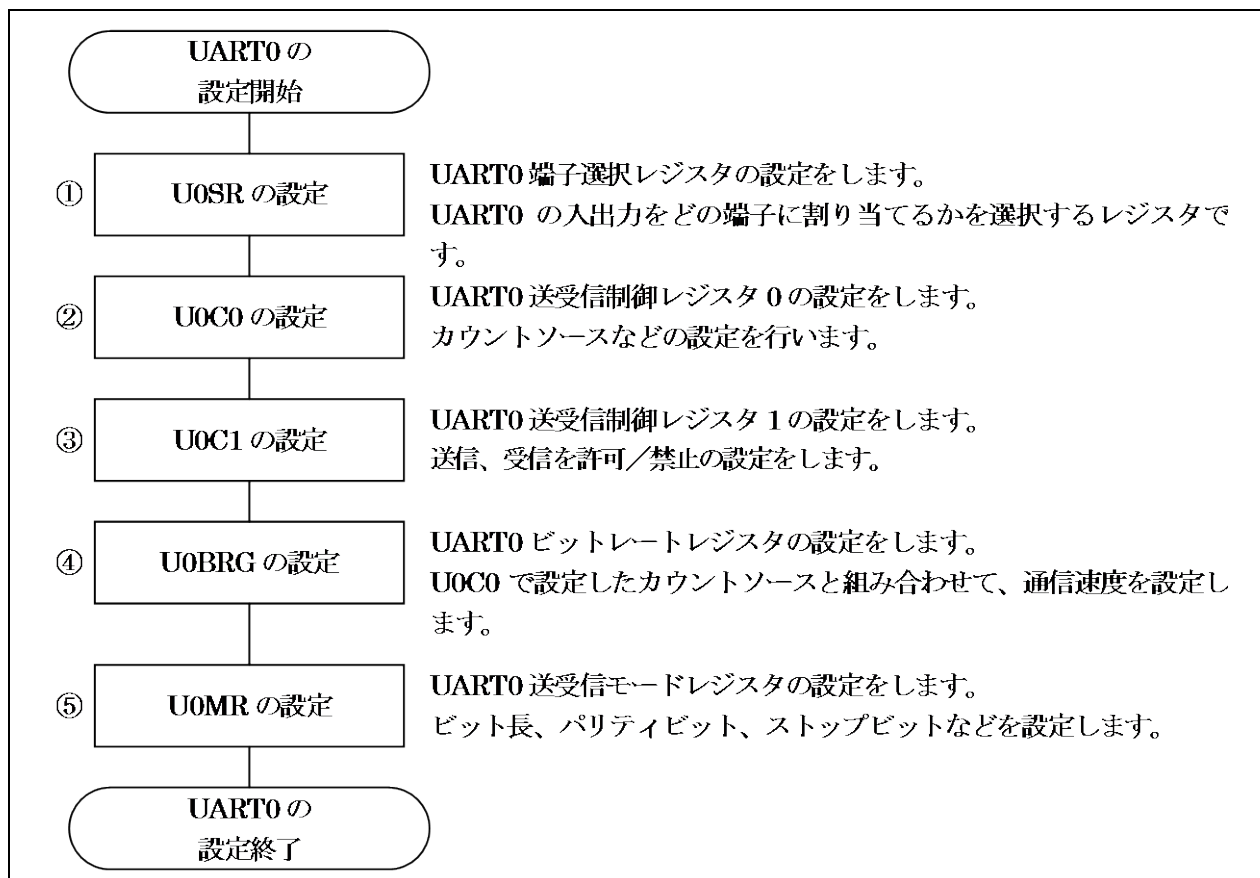
25. 通信(プロジェクト:uart0)

端子構成を下表に示します。

端子名	割り当てる端子	入出力	機能
TXD0	P1_4	出力	シリアルデータ出力
RXD0	P1_5	入力	シリアルデータ入力
CLK0	P1_6	入出力	転送クロック入出力
TXD1	P0_1またはP6_3	出力	シリアルデータ出力
RXD1	P0_2またはP6_4	入力	シリアルデータ入力
CLK1	P0_3、P6_2またはP6_5	入出力	転送クロック入出力

(2) シリアルインタフェース 0(UART0)の設定

シリアルインターフェース 0(UART0)を使用して、パソコンと通信を行います。レジスタの設定手順を下記に示します。



①UART0 端子選択レジスタ(U0SR:UART0 function select register)の設定

UART0 の入出力をどの端子に割り当てるかを選択するレジスタです。UART0 は、端子が決まっています。ここでは、使うか使わないかを選択します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7~5		"000"を設定	000
bit4	CLK0 端子選択ビット clk0sel0	0:CLK0 端子は使用しない 1:P1_6 に割り当てる CLK0 端子は、同期通信で使用します。シリアル通信は非同期通信なので使用しません。	0
bit3		"0"を設定	0
bit2	RXD0 端子選択ビット rxd0sel0	0:RXD0 端子は使用しない 1:P1_5 に割り当てる RXD0 端子は、外部(パソコン)から通信信号を受信する端子です。使用します。	1
bit1		"0"を設定	0
bit0	TXD0 端子選択ビット txd0sel0	0:TXD0 端子は使用しない 1:P1_4 に割り当てる TXD0 端子は、外部(パソコン)へ通信信号を送信する端子です。使用します。	1

UART0 端子選択レジスタ(U0SR)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	0	1	0	1
16 進数	0				5			

②UART0 送受信制御レジスタ 0 (U0C0:UARTi transmit/receive control register0)の設定

カウントソースなどの設定を行います。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7	転送フォーマット選択ビット uform_u0c0	0:LSB ファースト 1:MSB ファースト LSB ファーストを選択します。通常のシリアル通信は、LSB ファーストです。 LSB とは、Least Significant Bit (最下位ビット) のことで、LSB ファーストとはデータをいちばん下のビットから送信することです。MSB とは、Most Significant Bit (最上位ビット) のことで、MSB ファーストとはデータをいちばん上のビットから送信することです。	0
bit6	CLK 極性選択ビット ckpol_u0c0	0:転送クロックの立ち下がりで送信データ出力、立ち上がりで受信データ入力 1:転送クロックの立ち上がりで送信データ出力、立ち下がりで受信データ入力 CLK 端子は使いませんのでどちらでも構いません。今回は"0"にしておきます。	0
bit5	データ出力選択ビット nch_u0c0	0:TXD0 端子は CMOS 出力 1:TXD0 端子は N チャネルオープンドレイン出力 通常は CMOS 出力です。	0
bit4		"0"を設定	0
bit3	送信レジスタ空フラグ txept_u0c0	0:送信レジスタにデータあり(送信中) 1:送信レジスタにデータなし(送信完了) このビットは、読み込みしかできません。設定するときは 0 にしておきます。	0
bit2		"0"を設定	0
bit1,0	BRG カウントソース選択ビット (注 1) bit1:clk1_u0c0 bit0:clk0_u0c0	00:f1 を選択 (f1 =20MHz÷1=20MHz) 01:f8 選択 (f8 =20MHz÷8=2.5MHz) 10:f32 を選択 (f32 =20MHz÷32=0.625MHz) 11:fC を選択 (fC =本ボードでは無し) 今回は"00"を設定します。	00

注 1. BRG カウントソースを変更した場合は、U0BRG レジスタを再設定してください。

UART0 送受信制御レジスタ 0 (U0C0)の設定値を下記に示します。BRG カウントソース選択ビット(bit1,0)の設定方法は、UART0 ビットレートレジスタ(U0BRG)で説明します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	0	0	0	0
16 進数	0				0			

③UART0 送受信制御レジスタ 1 (U0C1:UART0 transmit/receive control register1)の設定

送信、受信を許可/禁止の設定をします。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7,6		"00"を設定	00
bit5	UART0 連続受信モード許可 ビット(注 2) u0rrm_u0c1	0:連続受信モード禁止 1:連続受信モード許可	0
bit4	UART0 送信割り込み要因選 択ビット u0irs_u0c1	0:送信バッファ空(TI=1) 1:送信完了(TXEPT=1)	0
bit3	受信完了フラグ(注 1) ri_u0c1	0:U0RB にデータなし 1:U0RB にデータあり このビットは、読み込みしかできません。設定は"0"に しておきます。	0
bit2	受信許可ビット re_u0c1	0:受信禁止 1:受信許可 データの受信をするので、受信許可にします。	1
bit1	送信バッファ空フラグ ti_u0c1	0:U0TB にデータあり 1:U0TB にデータなし このビットは、読み込みしかできません。設定は"0"に しておきます。	0
bit0	送信許可ビット te_u0c1	0:送信禁止 1:送信許可 データの送信をするので、送信許可にします。	1

注 1. RI ビット(bit3)は U0RB レジスタの上位バイトを読み出したとき、“0”になります。

注 2. UART モード時、U0RRM ビットは“0”(連続受信モード禁止)にしてください。

UART0 送受信制御レジスタ 1 (U0C1)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	0	1	0	1
16 進数	0				5			

④UART0 ビットレートレジスタ (U0BRG:UART0 bit rate register)の設定

UART0 ビットレートレジスタ(U0BRG)は、UART0 送受信制御レジスタ 0(U0C0)の BRG カウントソース選択ビット (bit1,0)と組み合わせて、通信速度を設定します。

U0C0 の bit1,0 の値と U0BRG の計算方法の関係を、下表に示します。

	U0C0 の bit1,0	U0BRG の計算
①	00	$U0BRG = 1,250,000 / \text{設定したいビットレート} - 1$
②	01	$U0BRG = 156,250 / \text{設定したいビットレート} - 1$
③	10	$U0BRG = 39,062.5 / \text{設定したいビットレート} - 1$
④	11	本ボードでは設定できません

設定したいビットレートは、下記のように計算します。結果は、四捨五入して整数にします。

- ・①を計算し、結果が 1～255 の値なら確定(U0C0 の bit1,0 は"00"にする)、それ以外なら ②で再計算
- ・②を計算し、結果が 1～255 の値なら確定(U0C0 の bit1,0 は"01"にする)、それ以外なら ③で再計算
- ・③を計算し、結果が 1～255 の値なら確定(U0C0 の bit1,0 は"10"にする)、
それ以外なら設定できないので、ビットレートを再検討

今回は、通信速度を 9600bps にします。

- ・①を計算、 $U0BRG = 1,250,000 / 9600 - 1 = 129.208 \approx 129$

結果は、1～255 の値なのでこれで OK です。よって、U0BRG には 129 を設定します。

```
u0brg = 129;    // 9600bpsを設定
```

⑤UART0 送受信モードレジスタ(U0MR:UART0 transmit/receive mode register)の設定

ビット長、パリティビット、ストップビットなどを設定します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7		"0"を設定	0
bit6	パリティ許可ビット prye_u0mr	0:パリティ禁止 1:パリティ許可 今回はパリティを設定しません。	0
bit5	パリティ奇/偶選択ビット pry_u0mr	0:奇数パリティ 1:偶数パリティ 今回は、パリティを使用しないのでどちらでも構いません。"0"にしておきます。	0
bit4	ストップビット長選択ビット stps_u0mr	0:1 ストップビット 1:2 ストップビット 1 ストップビットを設定します。	0
bit3	内/外部クロック選択ビット ckdir_u0mr	0:内部クロック 1:外部クロック(CLK0 端子) 内蔵クロックを選択します。	0
bit2~0	シリアル I/O モード選択ビット bit2:smd2_u0mr bit1:smd1_u0mr bit0:smd0_u0mr	000:シリアルインタフェースは無効 001:クロック同期形シリアル I/O モード 100:UART モード転送データ長 7ビット 101:UART モード転送データ長 8ビット 110:UART モード転送データ長 9ビット 上記以外:設定しないでください モードは、UART モードの転送データ長 8ビットに設定 します。	101

UART0 送受信モードレジスタ(U0MR)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	0	1	0	1
16 進数	0				5			

25.6.2 get_uart0 関数

get_uart0 関数は、シリアルインターフェース 0(UART0)から 1 文字受信する関数です。

```

91 : /*****/
92 : /* 1文字受信 */
93 : /* 引数 受信文字格納アドレス */
94 : /* 戻り値 -1:受信エラー 0:受信なし 1:受信あり 文字は*sに格納 */
95 : /*****/
96 : int get_uart0( unsigned char *s )
97 : {
98 :     int ret = 0, i;
99 :     unsigned int data;
100 :
101 :     if (ri_u0c1 == 1){ /* 受信データあり? */
102 :         data = u0rb;
103 :         *s = (unsigned char)data;
104 :         ret = 1;
105 :         if( data & 0xf000 ) { /* エラーあり? */
106 :             /* エラー時は再設定 */
107 :             re_u0c1 = 0;
108 :             for( i=0; i<50; i++ );
109 :             re_u0c1 = 1;
110 :
111 :             ret = -1;
112 :         }
113 :     }
114 :     return ret;
115 : }

```

(1) get_uart0 関数の使い方

get_uart0 関数の使い方を下記に示します。

```

unsigned char d; // 受信データ
int ret; // 受信結果

ret = get_uart0( &d ); // 受信データを保存する変数は、アドレス参照にする(&をつける)

```

引数は、受信したデータを保存する変数をアドレス参照で設定します(変数に「&」を付けます)。戻り値と引数(上記の場合は変数 d)の関係は、下表のようになります。

戻り値	説明
1	受信データありです。変数 d には、受信した 1 文字が格納されます。
0	受信データなしです。変数 d には、何も代入されません。
-1	受信エラーです。何かの問題で、うまく受信できませんでした。変数 d には、受信した 1 文字が格納されていますが、エラーで受信した不完全なデータが格納されます。

25. 通信(プロジェクト:uart0)

パソコンから「a」を送信し、下記プログラムを実行したとします。

```

unsigned char  d;           // 受信した値
int           ret;        // 受信結果

ret = get_uart0( &d );    // ret=1 , d=0x61 ('a')
    
```

変数 ret には 1、変数 d には 0x61 が代入されます。0x61 はアスキーコードの'a'です。これから、上記の内容がどのように実行されるのか説明します。

(2) 変数

```

98 :   int ret = 0, i;
99 :   unsigned int data;
    
```

98 行	変数 ret には、戻り値を設定します。まずは 0 を代入して、仮で受信データなしにしておきます。変数 i は、この関数内での作業用です。
99 行	変数 data には、受信したデータとエラー情報を格納します。

(3) データが受信されたときの処理

受信データがあるかどうかチェック、受信データがあったら格納処理などを行います。

```

101 :   if (ri_u0c1 == 1) {           /* 受信データあり?          */
102 :       data = u0rb;
103 :       *s = (unsigned char) data;
104 :       ret = 1;
    
```

101 行	「RI_U0C1」とは、UART0 送受信制御レジスタ 1 (U0C1)の受信完了フラグ(bit3)のことです。受信完了フラグ(bit3)は、 0: UART0 受信バッファレジスタ(U0RB)にデータなし 1: UART0 受信バッファレジスタ(U0RB)にデータあり なので、このビットが"1"なら受信ありと判断できます。																																		
102 行	<p>UART0 受信バッファレジスタ(U0RB)は、受信したデータとエラーの有無を保存しているレジスタです。このデータを変数 data に代入します。U0RB の内容を下記に示します。</p> <table border="1" style="margin-left: 20px;"> <tr> <th>bit</th> <th>15</th> <th>14</th> <th>13</th> <th>12</th> <th>11</th> <th>10</th> <th>9</th> <th>8</th> <th>7</th> <th>6</th> <th>5</th> <th>4</th> <th>3</th> <th>2</th> <th>1</th> <th>0</th> </tr> <tr> <td>内容</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>受信データ 0x00~0xff</td> </tr> </table> <p style="margin-left: 20px;"> オーバランエラーフラグ フレーミングエラーフラグ パリティエラーフラグ エラーサムフラグ </p> <p>U0RB の値を読み出した時点で、RI_U0C1 は自動的に"0"になります。</p>	bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	内容																受信データ 0x00~0xff
bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																			
内容																受信データ 0x00~0xff																			

103 行	UART0 受信バッファレジスタ(U0RB)は、受信したデータとエラーの有無を保存しているレジスタです。このデータをポインタ演算子 s が示すアドレスに代入します。「(unsigned char)」は、U0BR の下位 8 ビット(bit7～bit0)のみ代入するためのキャスト演算子です。
104 行	変数 ret を 1 にして、戻り値を受信ありにします。

(3) データにエラーがないかチェック

受信データにエラーが無いかどうかチェックします。

```

105 :         if( data & 0xf000 ) {           /* エラーあり?           */
106 :             /* エラー時は再設定 */
107 :                 re_u0c1 = 0;
108 :                 for( i=0; i<50; i++ );
109 :                 re_u0c1 = 1;
110 :
111 :                 ret = -1;
112 :         }
    
```

105 行	変数 data の bit15～bit12 をチェックします。0 以外なら受信エラーがあったと判断して、106 行以降を実行します。
107 行 ～109 行	エラーがあった場合、R8C/35A ハードウェアマニュアルには、次のことを行うように書かれています。 RE_U0C1 を"0"にしてから、RE_U0C1 を"1"にしてください そのため、107 行で RE_U0C1 を"0"、108 行でウェイトを入れ、109 行で RE_U0C1 を"1"にします。
111 行	戻り値をエラー(-1)にします。

(4) 終了

データありか、受信ありか、エラーかの情報を保存している変数 ret の値を戻り値にして終わります。

```

114 :         return ret;
115 :     }
    
```


25.6.3 put_uart0 関数

put_uart0 関数は、シリアルインターフェース 0(UART0)へ、1 文字送信する関数です。

```

117 : /*****
118 : /* 1 文字出力                                     */
119 : /* 引数   送信データ                               */
120 : /* 戻り値 0:送信中のため、送信できず 1:送信セット完了 */
121 : *****/
122 : int put_uart0( unsigned char r )
123 : {
124 :     if(ti_u0c1 == 1) {                               /* 送信データなし? */
125 :         u0tbl = r;
126 :         return 1;
127 :     } else {
128 :         /* 先に送信中(今回のデータは送信せずに終了) */
129 :         return 0;
130 :     }
131 : }

```

(1) put_uart0 関数の使い方

put_uart0 関数の使い方を下記に示します。

```

unsigned char  d;           // 送信するデータ
int           ret;        // 送信結果

ret = put_uart0( d );     // データ送信

```

引数は、送信するデータを設定している変数を設定します。戻り値は、下表のようになります。

戻り値	説明
1	送信セット完了です。シリアルコミュニケーション 0(UART0)は、送信を開始します。
0	前回、送信したデータを送信中のため、今回のデータは送信できません。

マイコンから、「a」を送信するために、下記プログラムを実行したとします。

```

unsigned char  d = 'a';   // 送信したいデータ
int           ret;       // 受信結果

ret = put_uart0( d );    // 'a' を送信, ret=1(送信セット完了)

```

送信セットが完了すれば変数 ret には 1 が代入されます。ret=0 の場合は送信セットできませんでしたので、再度送りたいときは戻り値が 1 になるまで put_uart0 関数を実行します。

25. 通信(プロジェクト:uart0)

送信セットされるまで繰り返したい場合は、下記のようにします。

```
while( !put_uart0( d ) );    // 送信セット完了するまで繰り返し続ける
```

while(値)は、値が 0 以外なら繰り返す、という命令です。put_uart0 関数の戻り値は、0 なら送信しなかったということなので、「!」(以外)を付けて、戻り値を「0(送信せず)以外」にしています。

これから、上記の作業がどのようにプログラムで実行されるのか説明します。

(2) 送信処理

```
124 :     if(ti_u0c1 == 1) {                               /* 送信データなし?          */
125 :         u0tbl = r;
126 :         return 1;
```

124 行	「TLU0C1」とは、UART0 送受信制御レジスタ 1(U0C1)の送信バッファ空フラグ(bit1)のことです。送信バッファ空フラグ(bit1)は、 0:UART0 送信バッファレジスタ(U0TB)にデータあり(現在送信中) 1:UART0 送信バッファレジスタ(U0TB)にデータなし なので、このビットが"1"なら今現在、送信しているデータはなしと判断できます。
125 行	UART0 送信バッファレジスタ(U0TB)に送信したいデータを代入して、送信を開始します。U0TB は 16bit 幅あり、送信データは下位ビットに設定します。「U0TBL」は U0TB の下位 8bit を示す名称です。そのため、U0TBL に値を代入します。U0TBL に値を代入すると、TLU0C1 が自動で"0"(データあり)になります。送信が完了すると、TLU0C1 が自動で"1"(データなし)になります。
126 行	送信データをセットできたので、戻り値 1 で終わります。

(3) 送信中のデータがあった場合

```
127 :     } else {
128 :         /* 先に送信中(今回のデータは送信せずに終了) */
129 :         return 0;
130 :     }
```

129 行	現在、送信中のデータがある場合は、何もせずに戻り値 0 でこの関数を終わります。
-------	--

25.6.4 main 関数

```

39 : void main( void )
40 : {
41 :     unsigned char  d;
42 :     int             ret;
43 :
44 :     init();                /* SFRの初期化                */
45 :
46 :     while( 1 ) {
47 :         ret = get_uart0( &d );
48 :         p6 = d;
49 :         if( ret == 1 ) put_uart0( d );
50 :     }
51 : }

```

44 行	init 関数を実行します。 init 関数では、ポートの入出力設定、シリアルコミュニケーション 0(UART0)の設定を行っています。
47 行	get_uart0 関数で、パソコンから送られてきたデータを受信します。 受信結果は変数 ret に、受信文字は変数 d に代入します。
48 行	受信したデータをポート 6 に出力します。
49 行	変数 ret が 1 なら、すなわち受信データがあるなら、put_uart0 関数で受信したデータと同じデータをパソコンに送信します。

25.7 実習手順

25.7.1 パソコン設定する前準備

これからパソコンの設定をします。その前に下記作業をあらかじめ済ませておきます。

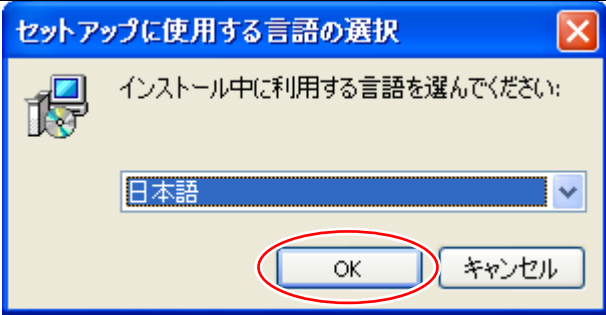
- ・マイコンボードに「uart0.mot」ファイルを書き込みます。
- ・USB ケーブルは接続したままにしておきます。


25.7.2 TeraTerm のインストール

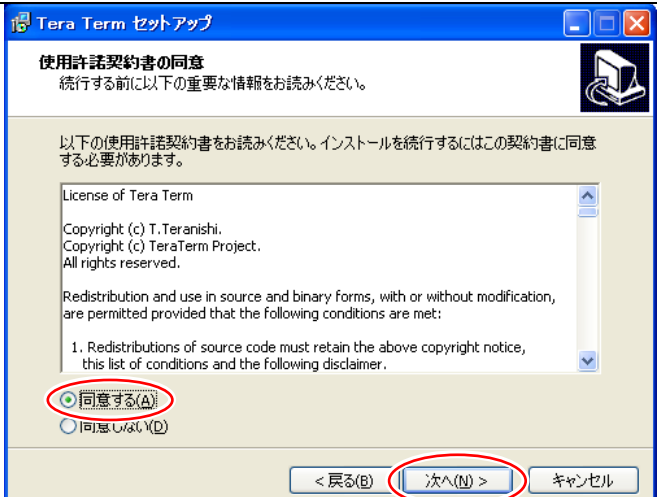
フリーソフトで通信のできる「Tera Term」というソフトを使用して、マイコンと通信をおこないます。ここではインストール手順を説明します。既に Tera Term をインストールしている場合は、インストールする必要はありません。

1	 <p>Telnetとシリアル接続に対応したターミナルエミュレーター「Tera Term Pro」を、多くの開発者の手で拡張したバージョン。原作者の許可を得てオープンソースで開発されている。「Tera Term Pro」からの一番の変更点は文字コードUTF-8の表示と、SSH/SSH2プロトコルによる接続への対応だが、</p>	<p>まず、ソフトをダウンロードします。インターネットブラウザで http://www.forest.impress.co.jp/lib/inet/servernt/remote/utf8teraterm.html にアクセスします。</p> <p>※窓の杜という Windows 用オンラインソフト紹介サイト、または同等のサイトでダウンロードします。</p>
2		<p>「DOWNLOAD」をクリックし、ファイルをダウンロードします。</p>
3		<p>ダウンロードした「teraterm-4.67.exe」を実行します。 ※バージョンにより、「4.67」部分は異なります。</p>

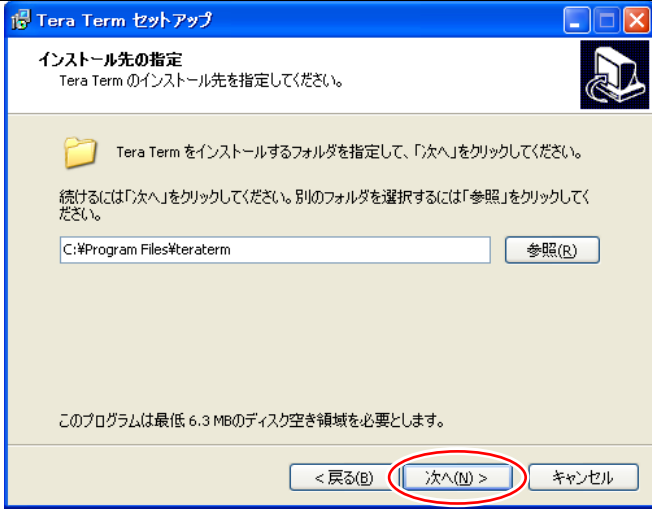
25. 通信(プロジェクト:uart0)

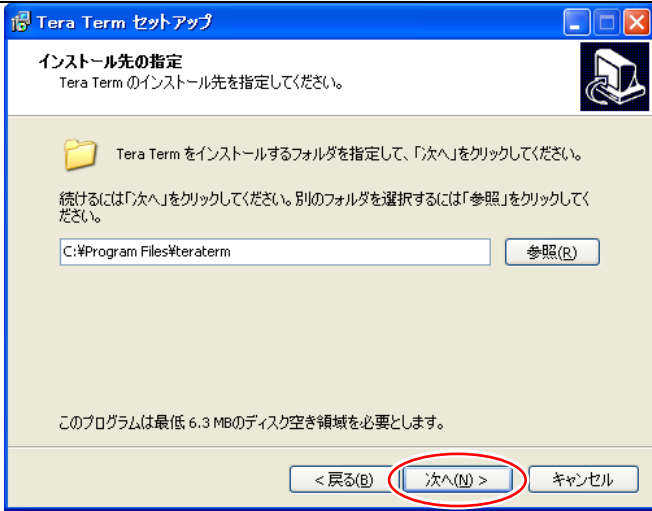
4		言語の選択です。 OK をクリックします。
---	---	------------------------------


5		次へをクリックします。
---	--	-------------

6		同意する場合は、「同意する」をクリックして、次へをクリックします。
---	---	-----------------------------------

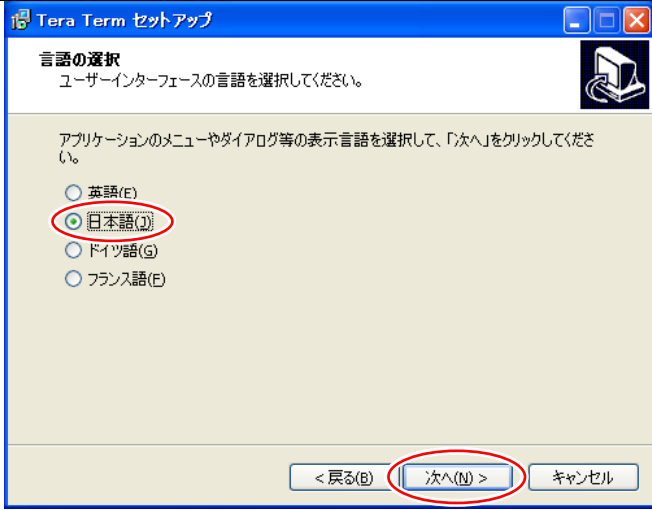
25. 通信(プロジェクト:uart0)

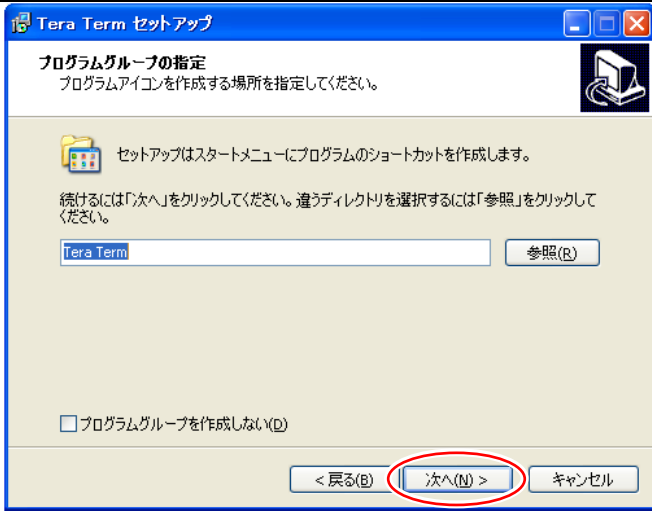
7	 <p>Tera Term セットアップ インストール先の指定 Tera Term のインストール先を指定してください。</p> <p>Tera Term をインストールするフォルダを指定して、「次へ」をクリックしてください。</p> <p>続けるには「次へ」をクリックしてください。別のフォルダを選択するには「参照」をクリックしてください。</p> <p>C:\Program Files\teraterm 参照(R)</p> <p>このプログラムは最低 6.3 MBのディスク空き領域を必要とします。</p> <p>< 戻る(B) 次へ(N) > キャンセル</p>	<p>次へをクリックします。</p>
---	---	--------------------

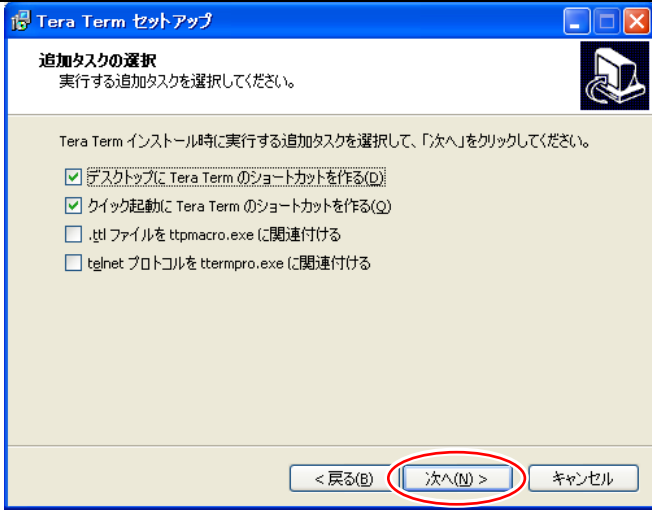
8	 <p>Tera Term セットアップ インストール先の指定 Tera Term のインストール先を指定してください。</p> <p>Tera Term をインストールするフォルダを指定して、「次へ」をクリックしてください。</p> <p>続けるには「次へ」をクリックしてください。別のフォルダを選択するには「参照」をクリックしてください。</p> <p>C:\Program Files\teraterm 参照(R)</p> <p>このプログラムは最低 6.3 MBのディスク空き領域を必要とします。</p> <p>< 戻る(B) 次へ(N) > キャンセル</p>	<p>次へをクリックします。</p>
---	--	--------------------

9	 <p>Tera Term セットアップ コンポーネントの選択 インストールコンポーネントを選択してください。</p> <p>インストールするコンポーネントを選択してください。インストールする必要のないコンポーネントはチェックを外してください。続行するには「次へ」をクリックしてください。</p> <p>カスタムインストール</p> <table border="1"> <tr><td><input checked="" type="checkbox"/></td><td>Tera Term & Macro</td><td>5.6 MB</td></tr> <tr><td><input type="checkbox"/></td><td>TTSSH</td><td>1.3 MB</td></tr> <tr><td><input type="checkbox"/></td><td>CygTerm+</td><td>0.1 MB</td></tr> <tr><td><input type="checkbox"/></td><td>LogMeTT</td><td>2.2 MB</td></tr> <tr><td><input type="checkbox"/></td><td>TTLEdit</td><td>1.6 MB</td></tr> <tr><td><input type="checkbox"/></td><td>TeraTerm Menu</td><td>0.2 MB</td></tr> <tr><td><input type="checkbox"/></td><td>TTProxy</td><td>0.3 MB</td></tr> <tr><td><input type="checkbox"/></td><td>Collector</td><td>1.6 MB</td></tr> <tr><td><input type="checkbox"/></td><td>追加プラグイン</td><td></td></tr> </table> <p>現在の選択は最低 6.3 MBのディスク空き領域を必要とします。</p> <p>< 戻る(B) 次へ(N) > キャンセル</p>	<input checked="" type="checkbox"/>	Tera Term & Macro	5.6 MB	<input type="checkbox"/>	TTSSH	1.3 MB	<input type="checkbox"/>	CygTerm+	0.1 MB	<input type="checkbox"/>	LogMeTT	2.2 MB	<input type="checkbox"/>	TTLEdit	1.6 MB	<input type="checkbox"/>	TeraTerm Menu	0.2 MB	<input type="checkbox"/>	TTProxy	0.3 MB	<input type="checkbox"/>	Collector	1.6 MB	<input type="checkbox"/>	追加プラグイン		<p>コンポーネントの選択です。どのコンポーネントも使いませんので、すべてのチェックを外します。次へをクリックします。</p>
<input checked="" type="checkbox"/>	Tera Term & Macro	5.6 MB																											
<input type="checkbox"/>	TTSSH	1.3 MB																											
<input type="checkbox"/>	CygTerm+	0.1 MB																											
<input type="checkbox"/>	LogMeTT	2.2 MB																											
<input type="checkbox"/>	TTLEdit	1.6 MB																											
<input type="checkbox"/>	TeraTerm Menu	0.2 MB																											
<input type="checkbox"/>	TTProxy	0.3 MB																											
<input type="checkbox"/>	Collector	1.6 MB																											
<input type="checkbox"/>	追加プラグイン																												

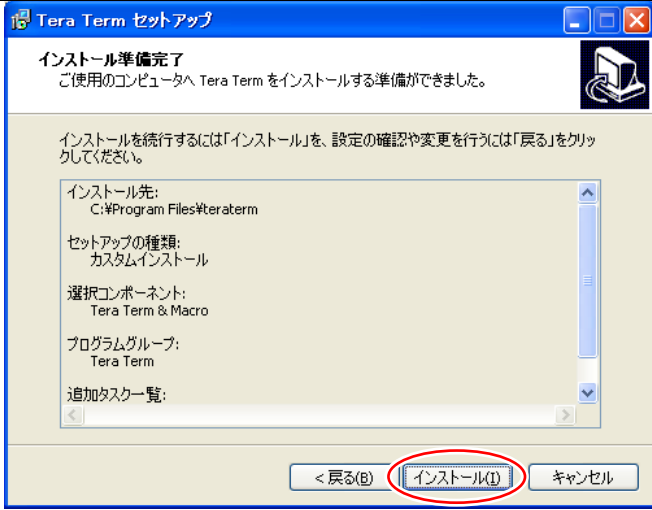
25. 通信(プロジェクト:uart0)

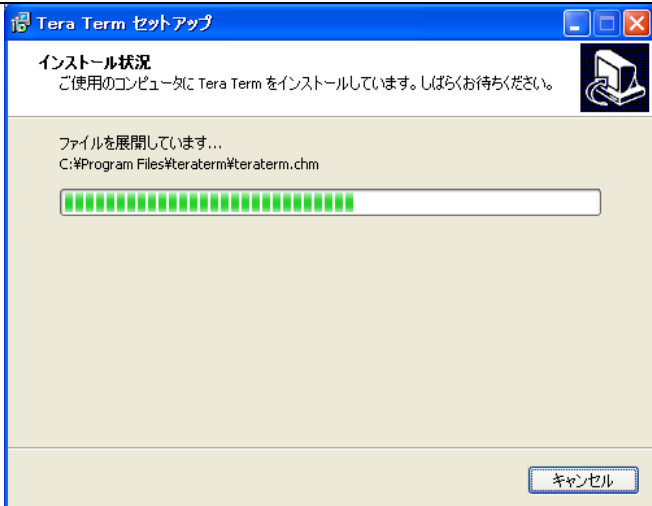
10		<p>「日本語」を選択して、「次へ」をクリックします。</p>
----	---	---------------------------------


11		<p>次へをクリックします。</p>
----	--	--------------------

12		<p>追加するタスクを選択して(通常は変更なしで大丈夫です)、「次へ」をクリックします。</p>
----	---	--

25. 通信(プロジェクト:uart0)

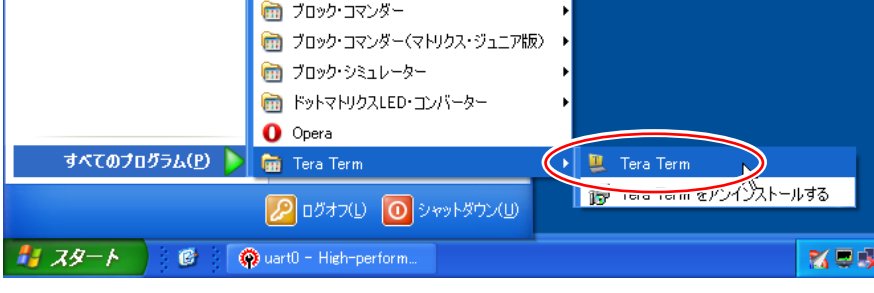
13		インストールをクリックします。
----	---	-----------------

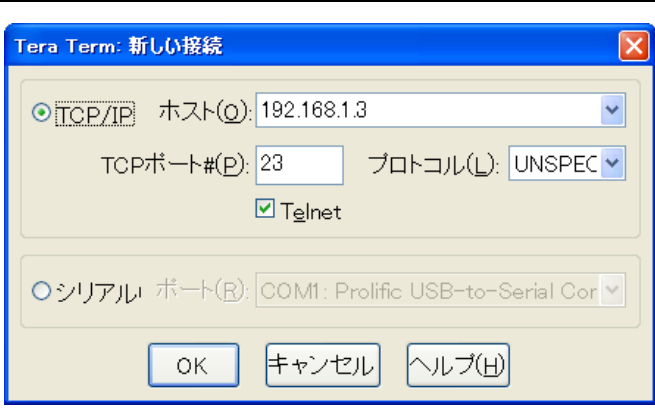
14		インストール中です。
----	--	------------


15		完了をクリックして、インストール終了です。
----	---	-----------------------


25.7.3 TeraTerm を使って、マイコンと通信しよう

※R8C Writer を使い、「uart0.mot」をマイコンに書き込んでおきます。

1		<p>「スタート」→「すべてのプログラム」→「Tera Term」→「Tera Term」で Tera Term を立ち上げます。</p>
---	--	---

2		<p>最初に、接続先を確認する画面が出てきます。</p>
---	--	------------------------------

3		<p>「Serial」を選んで、ポート番号を選びます。ポート番号は、R8C Writer で選択している番号と同じ番号にします。ミニマイコンカーVer.2のマイコン部の場合は、「USB Serial Port」と表示されている番号です。選択後、OKをクリックして次へ進みます。</p>
---	---	---

4		<p>立ち上がりました。</p>
---	---	------------------

25. 通信(プロジェクト:uart0)

7		<p>①電源スイッチを ON にします。左写真の場合はスイッチを右側に倒します。</p> <p>②リセットボタンを押します。</p> <p>このとき、マイコンボード左下にある2個のLEDが次の状態になっているか確認してください。</p> <p>(POWER): 点灯 (U S B): 点灯</p>
---	--	---

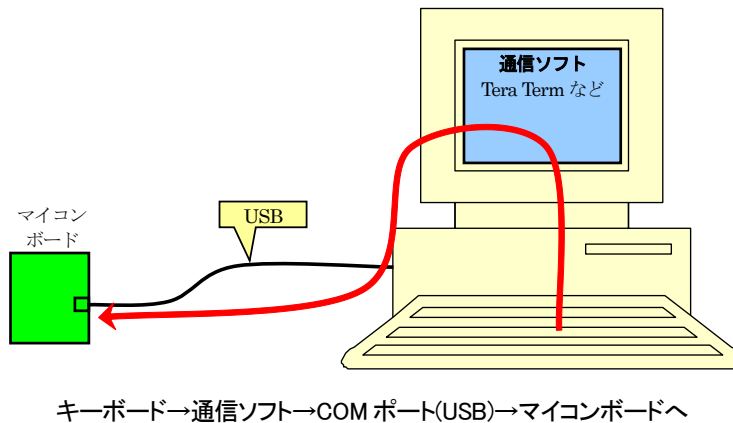
6		<p>これで準備が整いました。</p> <p>[a]キーを押してみてください。</p> <p>TeraTerm のウィンドウ内に「a」が表示されます。</p> <p>もし、表示されない場合は、</p> <ul style="list-style-type: none"> ・マイコンボードと USB 接続できていない ・COM ポートの番号が合っていない ・TeraTerm の設定が合っていない <p>などの理由が考えられます。もう一度、手順を確認してください。</p>
---	--	--

25.7.4 TeraTerm に表示される文字について

ワープロソフト(Microsoft の「Word」など)は、キーボードから入力された文字をソフト上に表示します。TeraTerm は、一見同じように動作しますが、実はキーボードから打ち込んだ文字が表示されているわけではありません。仕組みを説明します。

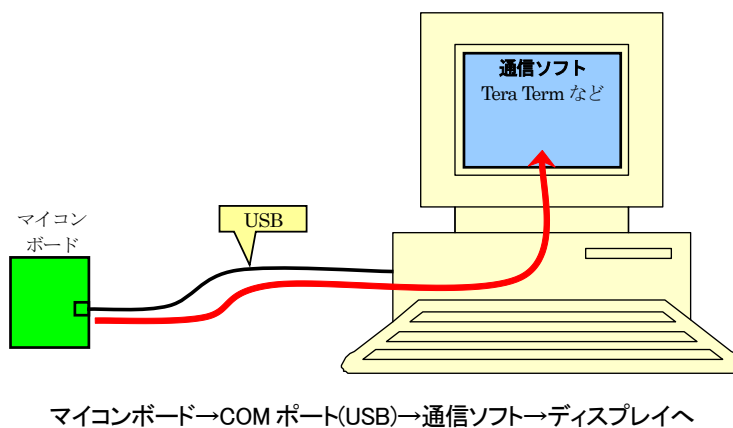
(1) キーボードに入力されたデータをマイコンボードで受信

通信ソフト(TeraTerm など)は、キーボードに入力された文字を、COMポートを通じてマイコンボードへ送ります。その様子を下図に示します。



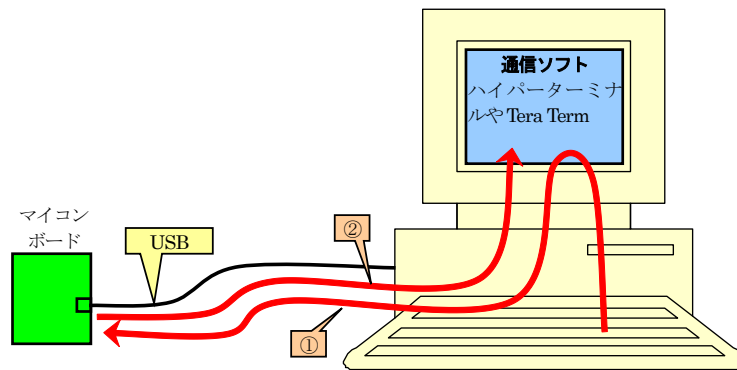
(2) マイコンボードから送られたデータを、パソコンに表示

通信ソフト(TeraTerm など)は、マイコンボードから送られてきた文字データを、COMポートを通じて画面に表示します。その様子を下図に示します。



(3) 今回の実習の通信経路

今回の実習の通信経路を下図に示します。



- ①キーボード→通信ソフト→COMポート(USB)→マイコンボード
- ②マイコンボード→COMポート(USB)→通信ソフト→ディスプレイ

今回、マイコンに書き込んだプログラム(uart0.c)は、受信したデータをそのまま送信するだけなので、キーボードから入力された文字が通信ソフト上に表示されるように見えます。しかし、通信ソフトに表示された文字は、マイコンボードから送られてきたデータを表示させているだけなのです。たまたま、キーボードに打ち込んだ文字と表示された文字が同じだっただけのことです。

試しに、マイコンボードの電源を切って、キーボードに文字を入力してみてください。通信ソフト上に打ち込んだ文字が表示されるでしょうか？

25.8 演習

(1) uart0.c の main 関数を下記のように書き換えてプログラムを書き込み、TeraTerm を立ち上げ、キーボードから文字を入力して動作を確かめなさい。

```
void main( void )
{
    unsigned char    d;
    int              ret;

    init();          /* SFR の初期化          */

    while( 1 ) {
        ret = get_uart0( &d );
        p6 = d;
        if( ret == 1 ) put_uart0( d + 1 );    // + 1 を追加
    }
}
```

26. printf、scanf 文を使った通信(プロジェクト:uart0_printf)

26.1 概要

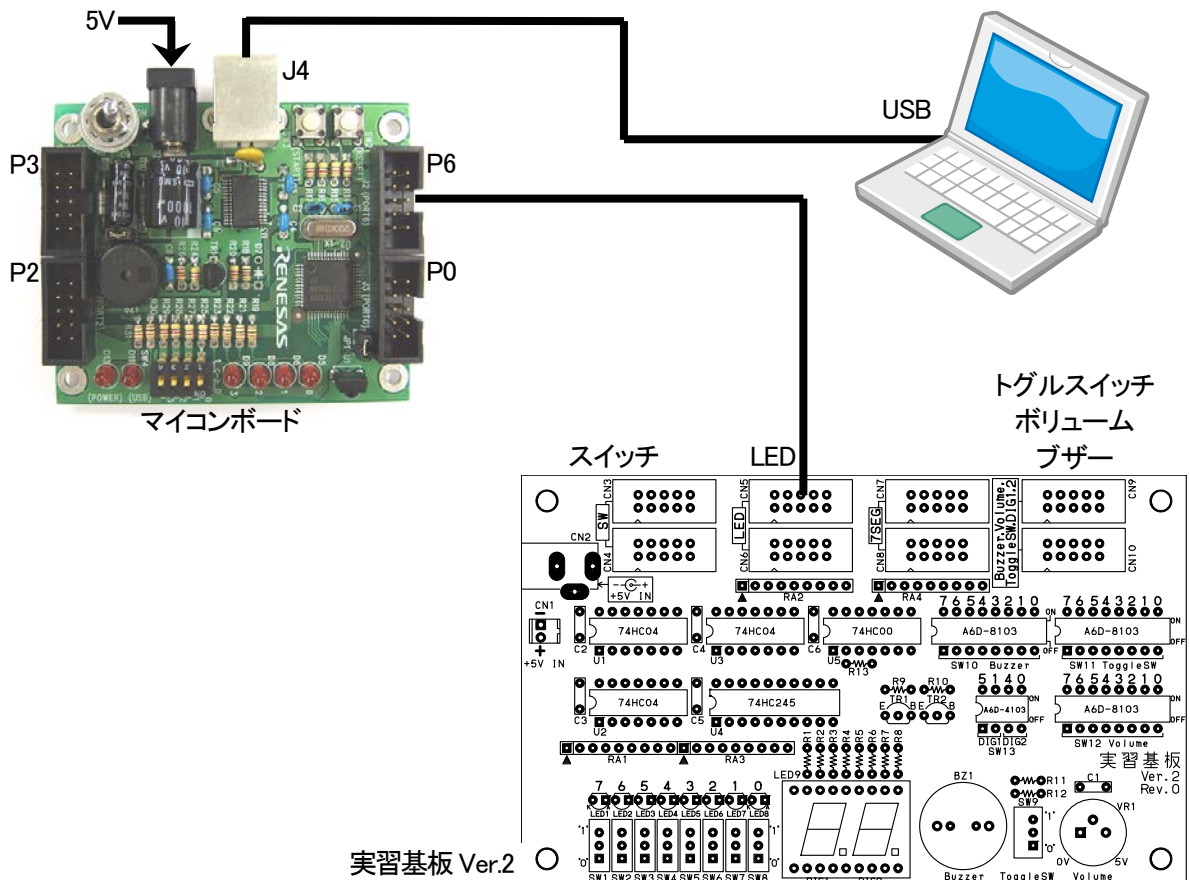
通信は、プロジェクト「uart0」と同じですが、本章ではパソコンの C 言語でよく使われる printf 文、scanf 文をマイコンで使う方法を説明します。

26.2 接続

■使用ポート

マイコンのポート	接続内容
J4 (USB コネクタ)	USB コネクタを通して、パソコン(通信ソフト)のキーボードから打ち込んだ文字コードを LED へ出力します。
P6 (J2)	実習基板 Ver.2 の LED 部など、出力機器を接続します。

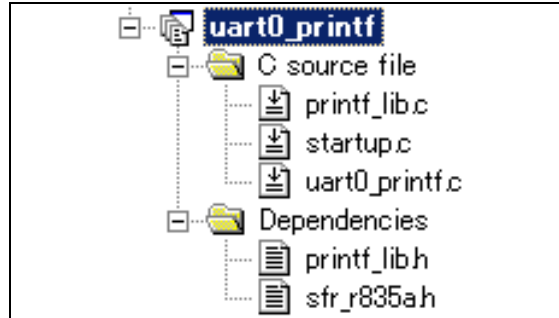
■接続例



■操作方法

パソコン側はTeraTerm やハイパーターミナルなどの通信ソフトを使います。キーボードから0~255までの数値を入力して、エンターキーを押すと、LED へ入力した値が出力されます。256 以上の値や、数値以外の文字も入力して、どうなるか試してみてください。

26.3 プロジェクトの構成



	ファイル名	内容
1	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。このファイルは共通で、どのプロジェクトもこのファイルから実行されます。
2	uart0_printf.c	実際に制御するプログラムが書かれています。R8C/35A の内蔵周辺機能(SFR)の初期化も行います。
3	printf_lib.c	printf 文、scanf 文を使用するためのライブラリです。 このファイルは変更しません。
4	sfr_r835a.h	R8C/35A マイコンの内蔵周辺機能を制御するためのレジスタ(Special Function Registers)を定義したファイルです。

26.4 プログラム「uart0_printf.c」

```

1 : /******
2 : /* 対象マイコン R8C/35A */
3 : /* ファイル内容 printf文、scanf文の使用例 */
4 : /* バージョン Ver. 1.20 */
5 : /* Date 2010.04.19 */
6 : /* Copyright ルネサスマイコンカーラリー事務局 */
7 : /* 日立インターメディアックス株式会社 */
8 : /******
9 : /*
10 : 入力 : UART0(パソコンのキーボードで入力したデータ)
11 : ※パソコンはTeraTermProなどの通信ソフトを使用します
12 : 出力 : P6_7-P6_0(LEDなど)
13 :
14 : TeraTermProなどの通信ソフトを通して、キーボードから入力した数値を
15 : ポート6に接続したLEDなどへ出力します。
16 : C言語でおなじみのprintf文、scanf文を使用した演習です。
17 : */
18 :
19 : /*=====*/
20 : /* インクルード */
21 : /*=====*/
22 : #include <stdio.h>
23 : #include "sfr_r835a.h" /* R8C/35A SFRの定義ファイル */
24 : #include "printf_lib.h" /* printf使用ライブラリ */
25 :
26 : /*=====*/
27 : /* シンボル定義 */
28 : /*=====*/
29 :

```

26. printf、scanf 文を使った通信(プロジェクト:uart0_printf)

```

30 : /*=====*/
31 : /* プロトタイプ宣言 */
32 : /*=====*/
33 : void init( void );
34 :
35 : /*=====*/
36 : /* メインプログラム */
37 : /*=====*/
38 : void main( void )
39 : {
40 :     int    i, ret;
41 :     char   c;
42 :
43 :     init();                /* SFRの初期化 */
44 :     init_uart0_printf( SPEED_9600 ); /* UART0とprintf関連の初期化 */
45 :     asm(" fset I ");      /* 全体の割り込み許可 */
46 :
47 :     printf( "Hello World!\n" );
48 :
49 :     while( 1 ) {
50 :         printf( "Input data : " );
51 :         ret = scanf( "%d", &i );
52 :         if( ret == 1 ) {
53 :             printf( "Get data : %d\n", i );
54 :             p6 = i;
55 :         } else {
56 :             printf( "Data Error!!\n" );
57 :             scanf( "%*[\n]" );
58 :         }
59 :     }
60 : }
61 :
62 : /*=====*/
63 : /* R8C/35A スペシャルファンクションレジスタ(SFR)の初期化 */
64 : /*=====*/
65 : void init( void )
66 : {
67 :     int i;
68 :
69 :     /* クロックをXINクロック(20MHz)に変更 */
70 :     prc0 = 1;                /* プロテクト解除 */
71 :     cm13 = 1;                /* P4_6, P4_7をXIN-XOUT端子にする */
72 :     cm05 = 0;                /* XINクロック発振 */
73 :     for(i=0; i<50; i++ );   /* 安定するまで少し待つ(約10ms) */
74 :     ocd2 = 0;                /* システムクロックをXINにする */
75 :     prc0 = 0;                /* プロテクトON */
76 :
77 :     /* ポートの入出力設定 */
78 :     prc2 = 1;                /* PD0のプロテクト解除 */
79 :     pd0 = 0xe0;              /* 7-5:LED 4:MicroSW 3-0:Sensor */
80 :     p1 = 0x0f;               /* 3-0:LEDは消灯 */
81 :     pd1 = 0xdf;              /* 5:RXD0 4:TXD0 3-0:LED */
82 :     pd2 = 0xfe;              /* 0:PushSW */
83 :     pd3 = 0xfb;              /* 4:Buzzer 2:IR */
84 :     pd4 = 0x83;              /* 7:XOUT 6:XIN 5-3:DIP SW 2:VREF */
85 :     pd5 = 0x40;              /* 7:DIP SW */
86 :     pd6 = 0xff;              /* LEDなど出力 */
87 : }
88 :
89 : /*=====*/
90 : /* end of file */
91 : /*=====*/

```

26.5 printf 文、scanf 文を使おう！

C 言語を習い始めると、パソコン上で動作するプログラムの場合は決まって以下のようなプログラムを作成します。

```
#include <stdio.h>
void main( void )
{
    printf("Hello World!\n") ;
}
```

コンパイルして実行してみるとパソコンの画面に、下記のように表示されます。

```
Hello World!
```

printf 関数が画面に表示する作業を行ってくれているのです。printf 関数は標準入出力ライブラリの「stdio.h」ファイルをインクルードすると使用できます。パソコンの場合は、表示したい内容をディスプレイに出力したり、キーボードから文字を入力したりすることができます。

しかしマイコンが組み込まれている装置は、特定の機器に組み込んで使用することが主な目的のため、ディスプレイやキーボードが付いていることはほとんどありません。ルネサス統合開発環境のCコンパイラは、ANSI C 規格に準じているため printf や scanf などの関数が用意されています。しかし、出力先や入力元が無い(分からない)ため、実行しても何も起こらないのです。**今回は、その printf 関数と scanf 関数を実行できるようにしてみました！！**

といっても、マイコンボードにディスプレイやキーボードを簡単に接続することはできません。接続する回路や制御プログラムが大変になります。

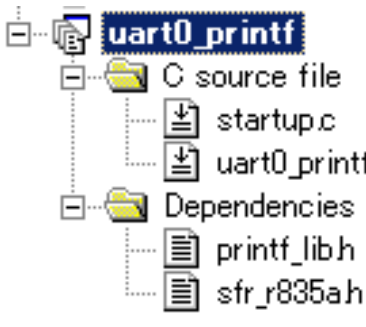
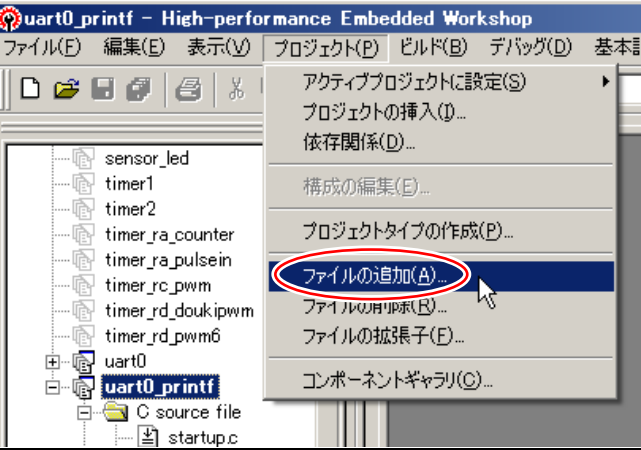
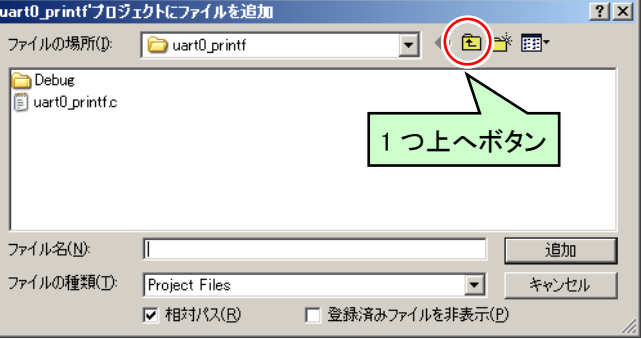
マイコンボードにプログラムを書き込むとき、マイコンボードとパソコンは USB で接続します。パソコンはミニマイコンカーVer.2 を COM ポートとして認識します。この COM ポートを通して、パソコンの画面やキーボードをあたかもマイコンボードの装置であるかのごとく使用することができます。ただ、パソコンの画面やキーボードはマイコンボードとパソコンを USB ケーブルで接続するだけでは使用できず、「ハイパーターミナル」や「Tera Term」などの通信ソフトを経由して使用します。

※printf 関数は多くのメモリを消費するため、R8C マイコンでは「%e, %E, %f, %g, %G」の変換指定文字は使用できません。

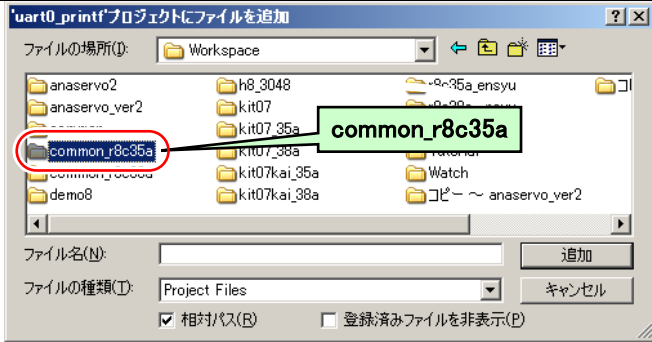
26.6 プログラムの解説「printf_lib.c」

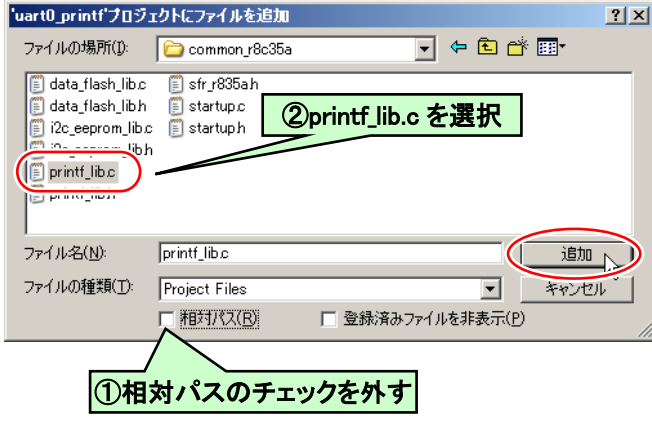
「printf_lib.c」は、R8C/35A で printf 文、scanf 文を COM ポートを通して使えるようにするためのファイルです。printf 文、scanf 文を使うときは、プロジェクトにこのファイルを追加します。このファイルは全プロジェクト共通のファイルです。このファイルを変更すると、このファイルを使っている別のプロジェクトにも影響しますので変更するときは気をつけてください。基本的には変更する必要がありません。

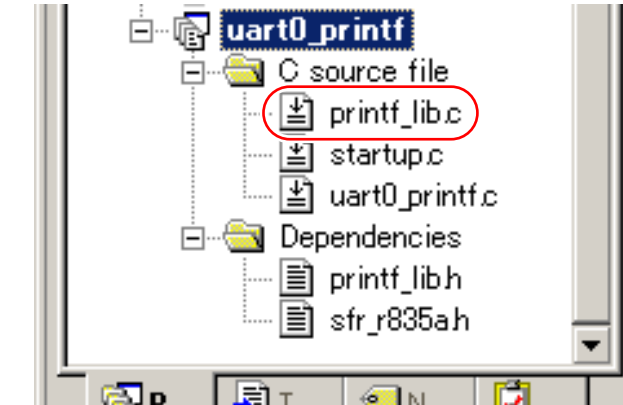
26.6.1 「printf_lib.c」の追加

1		<p>「printf_lib.c」がまだ追加されていないとします (サンプルプログラムには最初から追加されています)。</p>
2		<p>「プロジェクト→ファイルの追加」を選択します。</p>
3		<p>一つ上へボタンを何度かクリックして、Cドライブへ移動します。</p>

26. printf, scanf 文を使った通信(プロジェクト:uart0_printf)

4		<p>Cドライブ ↓ Workspace ↓ common_r8c35a</p> <p>フォルダを開きます。</p>
---	---	--

5		<p>①相対パスのチェックを外します。 ②「printf_lib.c」を選択します。 ③追加をクリックします。</p>
---	---	---

6		<p>「printf_lib.c」が追加されました。</p>
---	---	--------------------------------

26.6.2 関数一覧

■init_uart0_printf 関数

書式	<code>void init_uart0_printf(int sp)</code>
内容	printf 文の出力先や scanf 文の入力先を、UART0 にするように設定します。
引数	int 通信速度 通信速度を設定します。設定できる内容は下記のとおりです。 SPEED_4800 …通信速度を 4800bps に設定します。 SPEED_9600 …通信速度を 9600bps に設定します。 SPEED_19200 …通信速度を 19200bps に設定します。 SPEED_38400 …通信速度を 38400bps に設定します。 その他の設定は、データ長:8bit、パリティビット:無し、ストップビット:1bit、の設定で変更できません。 なお、init_uart0_printf 関数実行後、割り込みを許可してください。
戻り値	無し
使用例	<pre>init_uart0_printf(SPEED_9600); // 9600bps で通信 asm(" fset I "); // init_uart0_printf 関数実行後、割り込み許可する</pre>

■printf 関数

書式	<code>int printf(const char *format, ...);</code>
内容	UART0 に format で指定する書式に従い、データを出力します。
引数	付録を参照してください。
戻り値	正常時:出力した文字 異常時:EOF
使用例	<pre>printf("i は、%d です。", i); // i の値を表示</pre>
その他	プログラムの最初に stdio.h をインクルードしてください。 <code>#include <stdio.h></code>

■scanf 関数

書式	<code>int scanf(const char *format, ...);</code>
内容	UART0 から format で指定する書式に従い、データを読み込みます。
引数	付録を参照してください。
戻り値	正常時:読み込んで変換されたデータの数 入力終了または異常時:EOF
使用例	<pre>scanf("%d", &i); // i に 10 進数を読み込む</pre>
その他	プログラムの最初に stdio.h をインクルードしてください。 <code>#include <stdio.h></code>

26.7 プログラムの解説「uart0_printf.c」

26.7.1 インクルード部分

printf 文、scanf 文を使うために、専用のヘッダファイルをインクルードします。

```

19 : /*=====*/
20 : /* インクルード */
21 : /*=====*/
22 : #include <stdio.h>
23 : #include "sfr_r835a.h" /* R8C/35A SFRの定義ファイル */
24 : #include "printf_lib.h" /* printf使用ライブラリ */
    
```

stdio.h	ファイルがあるフォルダの位置は、 「C:\Program Files\Renesas\Hew\Tools\Renesas\nc30wa\v545r00\inc30」です。波線部分は、ルネサス統合開発環境のバージョンによって異なります。 このファイルを取り込む(インクルード)ことにより、printf 文や scanf 文を使えるようにします。 このファイルだけでは、printf 文や scanf 文が使えるようになるだけで、UART0 を使った通信は行えません。
printf_lib.h	ファイルがあるフォルダの位置は、「C:\Workspace\common_r8c35a」です。 このファイルを取り込む(インクルード)ことにより、printf 文の出力先や scanf 文の入力先を、UART0 にするように設定します。

26.7.2 main 関数—初期化部分

```

35 : /*=====*/
36 : /* メインプログラム */
37 : /*=====*/
38 : void main( void )
39 : {
40 :     int    i, ret;
41 :     char   c;
42 :
43 :     init(); /* SFRの初期化 */
44 :     init_uart0_printf( SPEED_9600 ); /* UART0とprintf関連の初期化 */
45 :     asm( "fset I " ); /* 全体の割り込み許可 */
    
```

43 行	init 関数を実行します。 init 関数では、ポートの入出力設定を行います。 シリアルコミュニケーション 0(UART0)の設定は、init 関数では行いません。プロジェクト「uart0」とは違いますので、気をつけてください。
44 行	init_uart0_printf 関数で、UART0 の初期化と printf 文、scanf 文を使えるようにしています。UART0 の通信速度は、9600bps に設定します。
45 行	printf 文でデータを送信する際、送信割り込みを使うので、UART0 の初期化後に全体の割り込みを許可します。

26.7.3 main 関数—通信部分

```

47 :     printf( "Hello World!¥n" );
48 :
49 :     while( 1 ) {
50 :         printf( "Input data : " );
51 :         ret = scanf( "%d", &i );
52 :         if( ret == 1 ) {
53 :             printf( "Get data : %d¥n", i );
54 :             p6 = i;
55 :         } else {
56 :             printf( "Data Error!!¥n" );
57 :             scanf( "%*[^¥n]" );
58 :         }
59 :     }
60 : }

```

47 行	最初に「Hello World!」と表示します。
50 行	「Input data : 」と表示して、データ入力待ちメッセージを表示します。
51 行	データの入力を待ちます。エンタキー入力で次の行へ進みます。正しくデータが入力されたなら変数 i に入力値が入ります。
52 行	scanf 関数の戻り値をチェックします。
53,54 行	データが正常に入力されていたら受信データをパソコンに送り返して、ポート 6 へ出力します。
56,57 行	データが異常ならエラーメッセージを表示して、受信バッファをクリアします。

26.7.4 受信バッファのクリア

プログラムの 57 行目で、バッファのクリアをするために代入抑止文字を使っています。

```
scanf( "%*[^¥n]" );
```

scanf 内の文字を分解すると下記ようになります。

%	*	[^	¥n]
①	②	③	④	⑤	

- ① 変換指定文字の開始です。
- ② 読み捨てる意味です。以後の書式を読み捨てます。
- ③ ⑤と対で、その中の文字のみを読み飛ばします。
- ④ 読み飛ばす文字は¥n、すなわち改行コードです。

総合すると、バッファを読み捨てますが、「¥n」のみ読み捨てるのを飛ばします。すなわち「¥n」のみが残ります。「¥n」により scanf 関数はバッファの終了と判断して次へ進みます。

もう一度、最初の入力部分の scanf 関数を見ると、

```
51 :          ret = scanf( "%d", &i );
```

ここで、「aaa(改行)」を入力したとします(改行=エンタ)。入力バッファには、

```
aaa¥n
```

と文字が保存されます。改行が入力されると変換が開始されます。変換指定文字「%d」は、10 進数入力です。それ以外のアルファベットなどの文字が入力されてしまうと「10 進数入力にもかかわらず数字以外のものが入力されていた場合は、その文字を読み込まずに作業は終了する」に当てはまってしまいます。この場合、エラー終了してしまいます。バッファはクリアされません。ここで再度、数値入力のため、

```
ret = scanf( "%d", &i );
```

という命令を記述するとどうなるでしょう。

バッファはクリアされておらず、親切にも(!!) 改行コードも有りますので、scanf 関数はすでにキー入力されたと勘違いして変換を開始してしまいます。もちろんエラーですのでバッファはクリアしないまま次へ移ります。これを繰り返すと無限ループになり暴走してしまいます。そこで、入力値が正しいかどうか scanf 関数の戻り値を判定してエラーがあればバッファをクリアします。scanf 関数の戻り値は先に説明したとおり正常に終了すると、読み込んで変換されたデータの数が返ってきます。ここでは1です。1以外ならエラーと判断できます。

```
52 :          if( ret == 1 ) {
53 :              printf( "Get data : %d¥n", i );
54 :              p6 = i;
```

戻り値を格納した ret 変数が 1 なら、printf 関数で値を表示して、ポート6へその値を出力します。

```
55 :          } else {
56 :              printf( "Data Error!!¥n" );
57 :              scanf( "%*[^¥n]" );
58 :          }
```

else 文、すなわち戻り値が1でなくエラーであれば、入力エラーと表示して、バッファをクリアします。書籍によっては scanf 関数は、変換指定文字以外のデータが入力されたとき暴走するので使わない方が良いと書かれていますが、うまくバッファをクリアすればこれほど便利な関数はありません。

27. データフラッシュ(プロジェクト:data_flash)

27.1 概要

本章では、マイコン内蔵のデータフラッシュをプログラムで書き換える方法を説明します。パラメータなど、何か値を保存しておきたいときデータフラッシュに保存し、次回電源が ON になったときに読み込むことができます。データフラッシュの容量は 4KB あります。それ以下の容量を保存するなら、外付けのメモリ(EEP-ROM など)は必要ありません。

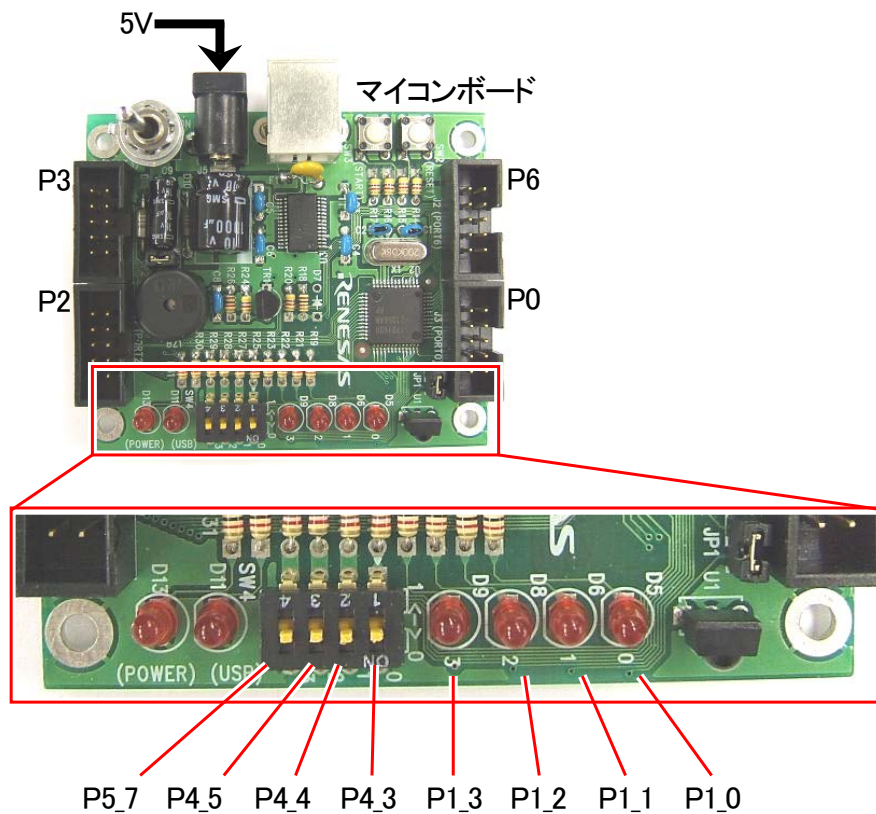
27.2 接続

■使用ポート

マイコンのポート	接続内容
P5_7、P4_5、P4_4、P4_3	マイコンボード上のディップスイッチです。
P1_3、P1_2、P1_1、P1_0	マイコンボード上の LED です。

■接続例

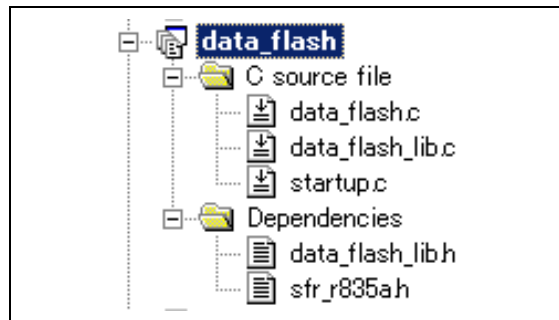
マイコンボードだけで実習できます。



■操作方法

「27.8 実習手順」を参照してください。

27.3 プロジェクトの構成



	ファイル名	内容
1	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。このファイルは共通で、どのプロジェクトもこのファイルから実行されます。
2	data_flash.c	実際に制御するプログラムが書かれています。R8C/35A の内蔵周辺機能(SFR)の初期化も行います。
3	data_flash_lib.c	printf 文、scanf 文を使用するためのライブラリです。 このファイルは変更しません。
4	sfr_r835a.h	R8C/35A マイコンの内蔵周辺機能を制御するためのレジスタ (Special Function Registers)を定義したファイルです。

27.4 プログラム「data_flash.c」

```

1 : /******
2 : /* 対象マイコン R8C/35A */
3 : /* ファイル内容 データフラッシュの書き換え */
4 : /* バージョン Ver. 1. 21 */
5 : /* Date 2012. 03. 12 */
6 : /* Copyright ルネサスマイコンカーラーイ事務局 */
7 : /* 日立インターメディックス株式会社 */
8 : /******
9 : /*
10 : 入力：マイコンボードのディップスイッチ
11 : 出力：マイコンボードのLED
12 :
13 : データフラッシュに書き込んでいる値をマイコンボードのLEDに出力します。
14 : マイコンボード上のプッシュスイッチを押すと、
15 : ディップスイッチの値をデータフラッシュに保存します。
16 : */
17 :
18 : /*=====*/
19 : /* インクルード */
20 : /*=====*/
21 : #include "sfr_r835a.h" /* R8C/35A SFRの定義ファイル */
22 : #include "data_flash_lib.h" /* データフラッシュライブラリ */
23 :
24 : /*=====*/
25 : /* シンボル定義 */
26 : /*=====*/
27 : /* データフラッシュ関連 */
28 : #define DF_SIZE 64 /* 読み書きサイズ */
29 :
30 : #define DF_CHECK 0 /* データフラッシュチェック */
31 : #define DF_DATA 1 /* データ */
32 :

```


27. データフラッシュ(プロジェクト:data_flash)

```

33 : /*=====*/
34 : /* プロトタイプ宣言 */
35 : /*=====*/
36 : void init( void );
37 : unsigned char dipsw_get( void );
38 : void led_out( unsigned char led );
39 : unsigned char pushsw_get( void );
40 : void timer( unsigned long timer_set );
41 :
42 : /*=====*/
43 : /* グローバル変数の宣言 */
44 : /*=====*/
45 : /* データフラッシュ関連 */
46 : signed char data_buff[ DF_SIZE ]; /* 一時保存エリア */
47 :
48 : /*=====*/
49 : /* メインプログラム */
50 : /*=====*/
51 : void main( void )
52 : {
53 :     int ret;
54 :
55 :     init(); /* 初期化 */
56 :
57 :     readDataFlash( 0x3000, data_buff, DF_SIZE );
58 :
59 :     if( data_buff[DF_CHECK] != 0x35 ) {
60 :         data_buff[DF_CHECK] = 0x35;
61 :         data_buff[DF_DATA] = 0;
62 :     }
63 :
64 :     led_out( data_buff[DF_DATA] );
65 :
66 :     while( !pushsw_get() );
67 :
68 :     data_buff[DF_DATA] = dipsw_get();
69 :     blockEraseDataFlash( 0x3000 );
70 :     ret = writeDataFlash( 0x3000, data_buff, DF_SIZE );
71 :
72 :     if( ret == 0 ) {
73 :         /* 書き込みエラーなら */
74 :         while( 1 ) {
75 :             led_out( 0xc );
76 :             timer( 100 );
77 :             led_out( 0x3 );
78 :             timer( 100 );
79 :         }
80 :     }
81 :
82 :     /* 書き込みが正常にできたなら */
83 :     while( 1 ) {
84 :         led_out( 0x5 );
85 :         timer( 200 );
86 :         led_out( 0xa );
87 :         timer( 200 );
88 :     }
89 : }
90 :
91 : /*=====*/
92 : /* R8C/35A スペシャルファンクションレジスタ(SFR)の初期化 */
93 : /*=====*/
94 : void init( void )
95 : {
96 :     int i;
97 :
98 :     /* クロックをXINクロック(20MHz)に変更 */
99 :     prc0 = 1; /* プロテクト解除 */
100 :     cm13 = 1; /* P4_6, P4_7をXIN-XOUT端子にする */
101 :     cm05 = 0; /* XINクロック発振 */
102 :     for(i=0; i<50; i++ ); /* 安定するまで少し待つ(約10ms) */
103 :     ocd2 = 0; /* システムクロックをXINにする */
104 :     prc0 = 0; /* プロテクトON */
105 :
106 :     /* ポートの入出力設定 */
107 :     prc2 = 1; /* PD0のプロテクト解除 */
108 :     pd0 = 0xe0; /* 7-5:LED 4:MicroSW 3-0:Sensor */
109 :     p1 = 0x0f; /* 3-0:LEDは消灯 */
110 :     pd1 = 0xdf; /* 5:RXD0 4:TXD0 3-0:LED */
111 :     pd2 = 0xfe; /* 0:PushSW */
112 :     pd3 = 0xfb; /* 4:Buzzer 2:IR */
113 :     pd4 = 0x83; /* 7:XOUT 6:XIN 5-3:DIP SW 2:VREF */
114 :     pd5 = 0x40; /* 7:DIP SW */
115 :     pd6 = 0xff;
116 : }
117 :

```

27. データフラッシュ(プロジェクト:data_flash)

```

118 : /*****/
119 : /* ディップスイッチ値読み込み */
120 : /* 戻り値 スイッチ値 0~15 */
121 : /*****/
122 : unsigned char dipsw_get( void )
123 : {
124 :     unsigned char sw, sw1, sw2;
125 :
126 :     sw1 = (p5>>4) & 0x08;          /* ディップスイッチ読み込み3 */
127 :     sw2 = (p4>>3) & 0x07;          /* ディップスイッチ読み込み2,1,0*/
128 :     sw = sw1 | sw2;                /* P5とP4の値を合わせる */
129 :
130 :     return sw;
131 : }
132 :
133 : /*****/
134 : /* プッシュスイッチ値読み込み */
135 : /* 戻り値 スイッチ値 ON:1 OFF:0 */
136 : /*****/
137 : unsigned char pushsw_get( void )
138 : {
139 :     unsigned char sw;
140 :
141 :     sw = ~p2;                       /* スイッチのあるポート読み込み */
142 :     sw &= 0x01;
143 :
144 :     return sw;
145 : }
146 :
147 : /*****/
148 : /* マイコン部のLED出力 */
149 : /* 引数 スイッチ値 0~15 */
150 : /*****/
151 : void led_out( unsigned char led )
152 : {
153 :     unsigned char data;
154 :
155 :     led = ~led;
156 :     led &= 0x0f;
157 :     data = p1 & 0xf0;
158 :     p1 = data | led;
159 : }
160 :
161 : /*****/
162 : /* タイマ本体 */
163 : /* 引数 タイマ値 l=1ms */
164 : /*****/
165 : void timer( unsigned long timer_set )
166 : {
167 :     int i;
168 :
169 :     do {
170 :         for( i=0; i<1240; i++ );
171 :     } while( timer_set-- );
172 : }
173 :
174 : /*****/
175 : /* end of file */
176 : /*****/

```

27.5 データフラッシュを使おう！

27.5.1 概要

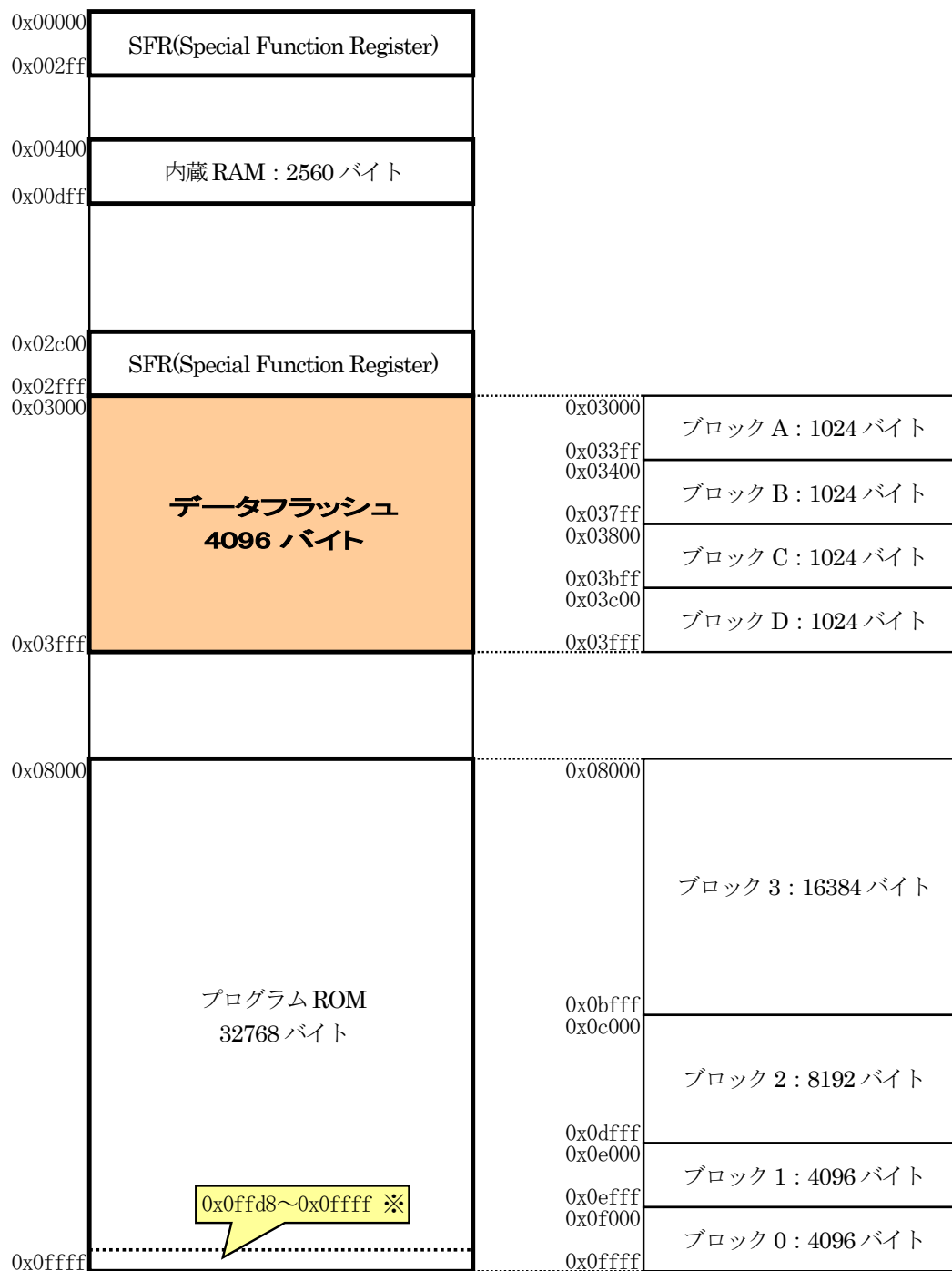
R8C/35A には、電气的に書き換え可能なフラッシュメモリが内蔵されています。フラッシュメモリは 2 種類あり、次のような特徴があります。

フラッシュメモリ	プログラム ROM	データフラッシュ
内容	主にプログラムを格納するためのフラッシュメモリです。	主に書き換えが必要なデータを格納するためのフラッシュメモリです。ただ、プログラムを格納しても問題ありません。
プログラム、イレーズ回数	1,000 回(メーカー保証回数)	10,000 回(メーカー保証回数)
容量	32KB(32,768 バイト)	4KB(4,096 バイト)
ブロック	メモリはブロックに分かれており、イレーズはブロックごとに行います。 ブロック 3:0x08000~0x0bfff 番地の 16KB ブロック 2:0x0c000~0x0dfff 番地の 8KB ブロック 1:0x0e000~0x0efff 番地の 4KB ブロック 0:0x0f000~0x0ffff 番地の 4KB	メモリはブロックに分かれており、イレーズはブロックごとに行います。 ブロック A:0x03000~0x033ff 番地の 1KB ブロック B:0x03400~0x037ff 番地の 1KB ブロック C:0x03800~0x03bff 番地の 1KB ブロック D:0x03c00~0x03fff 番地の 1KB

※イレーズとは、メモリを 0xff にすることです。

27.5.2 メモリマップ

データフラッシュは、0x3000~0x3fff 番地にあります。

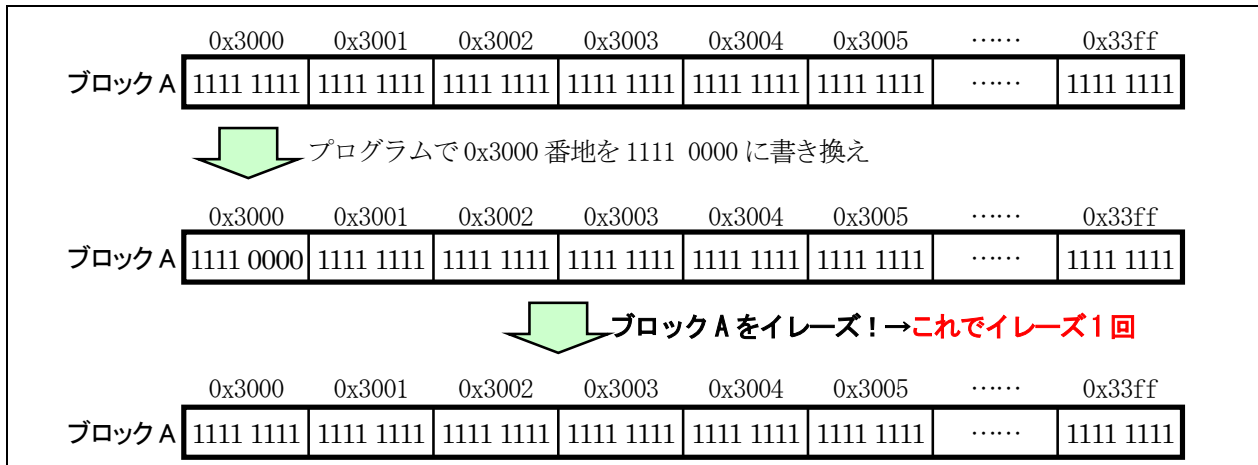


※0x0ffd8~0x0ffdb 番地は予約領域、0x0ffdc~0x0fff 番地は固定割り込みベクタテーブルの領域のため、プログラムは置けません。

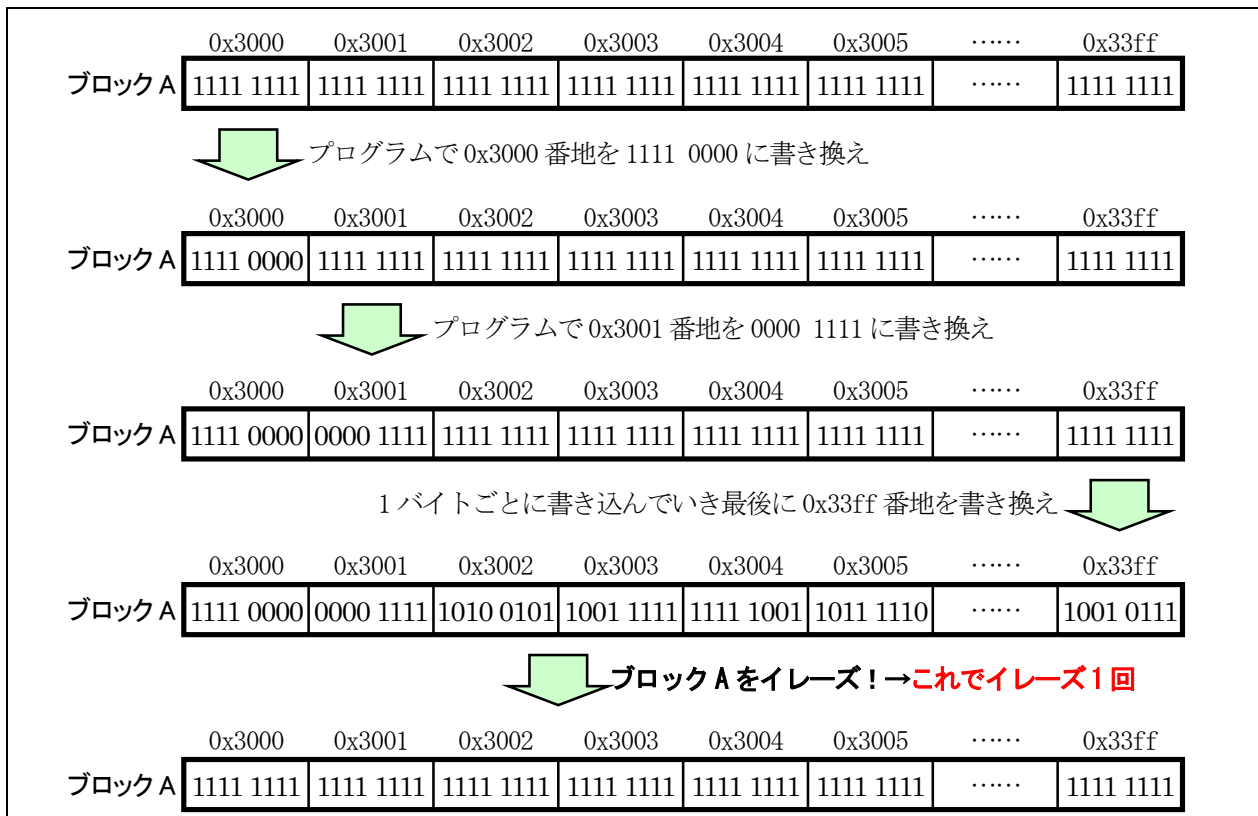
27.5.3 データフラッシュを効率的に使う

データフラッシュの初期値は 0xff(1111 1111₂)です。書き込みは"1"のビットを"0"にすることを言います。"0"のビットを"1"にすることはできません。"0"のビットを"1"にする作業をイレーズといい、先ほどの書き換え回数はイレーズ回数のことです。書き込み回数は制限がありません。イレーズは、ブロック単位で行います。例えば、0x3000番地をイレーズしたくても 0x3000番地だけイレーズすることはできず、0x3000番地があるブロック A(0x3000～0x33ff番地)がイレーズされます。

例えば、0x3000番地に 0xf0 を書き込んだとします。0x3000番地だけをイレーズ 0xff にすることになります。イレーズ回数は 1 回したことになります(下図)。



次に、0x3000番地に 0xf0 を書き込んだとします。さらに、0x3001番地に 0xf0 を書き込みます。どんどん書き込んでいき 0x33ff まで 1024 バイト分書き込んだ後にイレーズすると、イレーズ回数は 1 回になります(下図)。

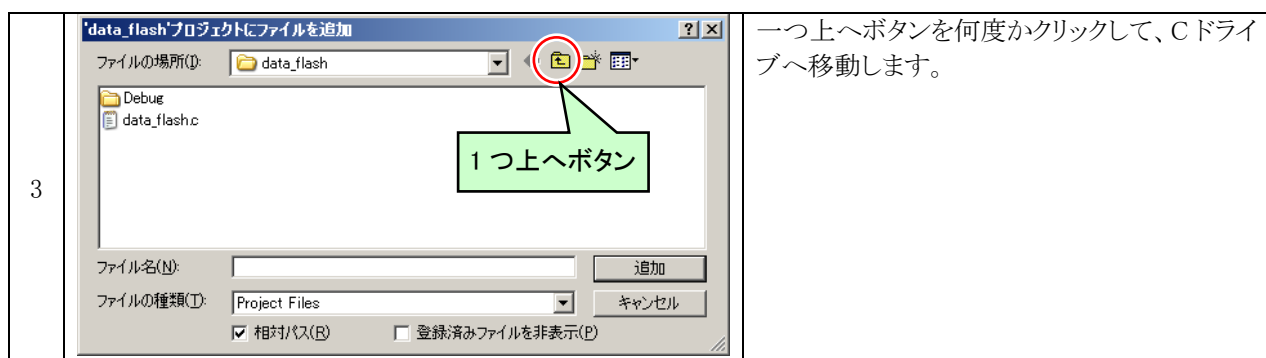
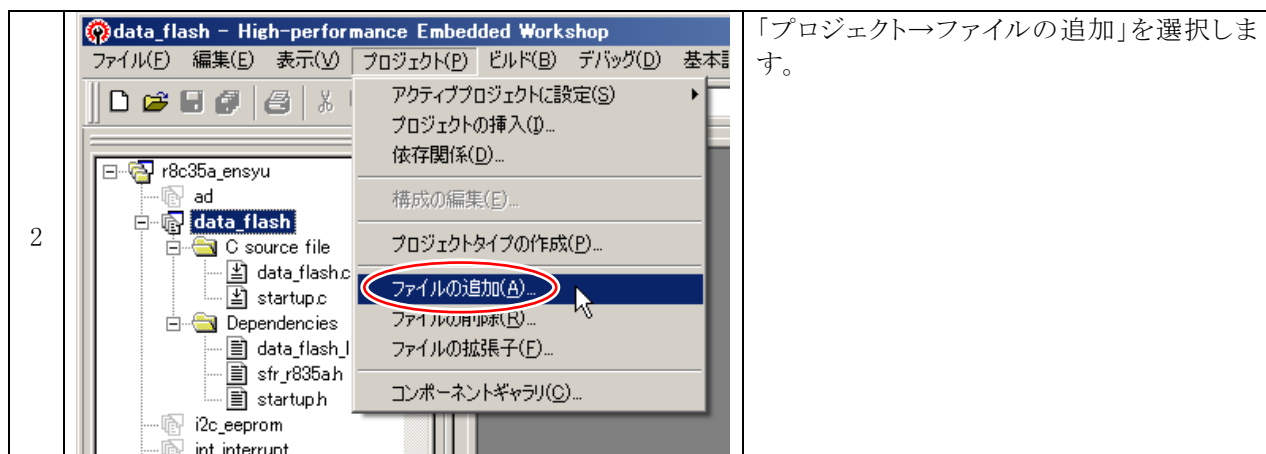
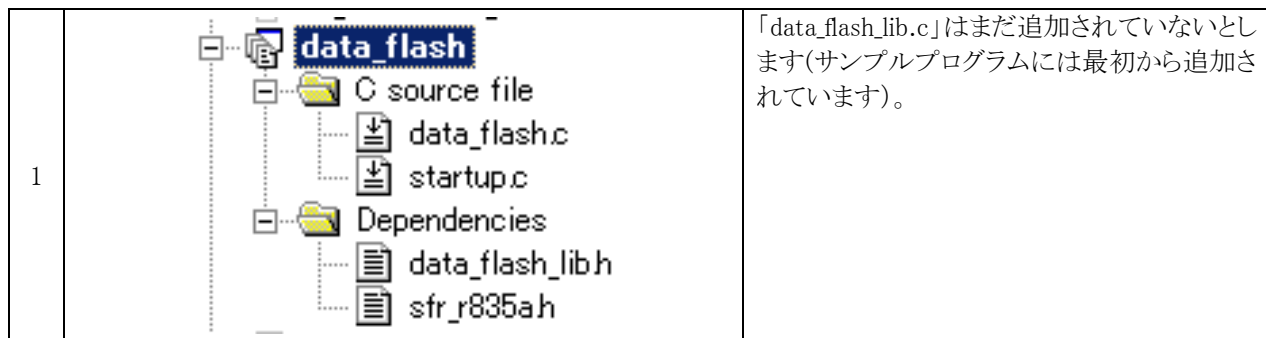


このように、イレーズ回数を減らすことによってデータフラッシュを効率的(長く)に使うことができます。プログラムでは、1 ブロック使い切るまで書き込んでいき、使い切った時点でイレーズするようにします。

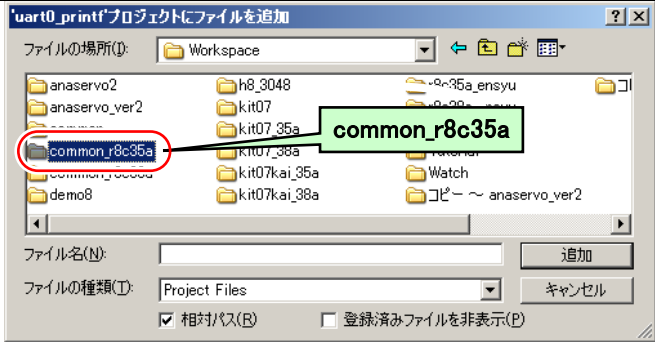
27.6 プログラムの解説「data_flash_lib.c」

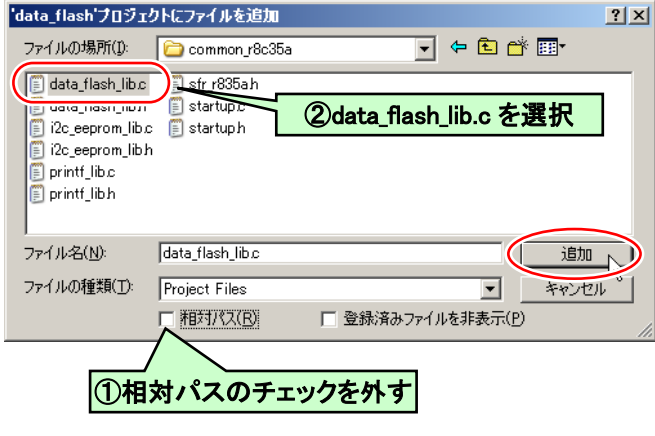
「data_flash_lib.c」は、R8C/35Aに内蔵しているデータフラッシュを簡単に使えるようにするためのファイルです。データフラッシュを使うときは、プロジェクトにこのファイルを追加します。このファイルは全プロジェクト共通のファイルです。このファイルを変更すると、このファイルを使っている別のプロジェクトにも影響しますので変更するときは気をつけてください。基本的には変更する必要がありません。

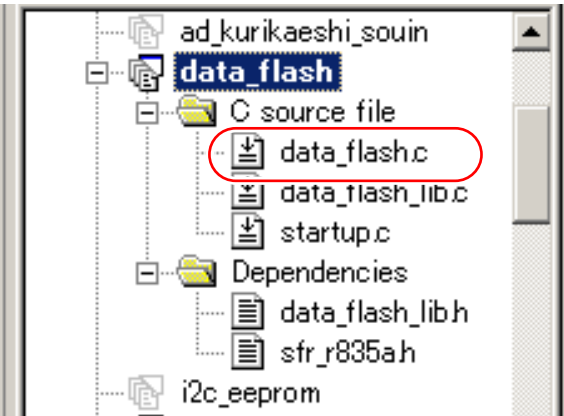
27.6.1 「data_flash_lib.c」の追加



27. データフラッシュ(プロジェクト:data_flash)

4		<p>Cドライブ ↓ Workspace ↓ common_r8c35a</p> <p>フォルダを開きます。</p>
---	---	--

5		<p>①相対パスのチェックを外します。 ②「data_flash_lib.c」を選択します。 ③追加をクリックします。</p>
---	---	---

6		<p>「data_flash_lib.c」が追加されました。</p>
---	---	------------------------------------

27.6.2 関数一覧

■readDataFlash 関数

書式	void readDataFlash(unsigned int r_address, 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 char* 読み込み先アドレス int 読み込むデータ数
戻り値	無し
使用例	char data_buff[64]; /* データフラッシュ用保管エリア */ readDataFlash(0x3000, data_buff, 64); /* 0x3000 番地から、data_buff 配列に、64 個のデータを読み込みます。 */

■writeDataFlash 関数

書式	int writeDataFlash(unsigned int w_address, 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 char* 書き込むデータのあるアドレス int 書き込むデータ数
戻り値	1:書き込み完了 0:異常終了
使用例	char data_buff[64]; /* データフラッシュ用保管エリア */ writeDataFlash(0x3000, data_buff, 64); /* 0x3000 番地から、data_buff 配列のデータを、64 個書き込みます。 */
注意点	<p>●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 64 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。</p> <p>例) writeDataFlash(0x33f0, data_buff, 64); /* これはできない */</p> <p>●書き込む前のイレーズについて 書き込む前に、書き込む該当ブロックをイレーズします。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。</p>

■blockEraseDataFlash 関数

書式	int blockEraseDataFlash(unsigned int address)
内容	データフラッシュをイレーズ(0xff で埋めること)します。
引数	unsigned int イレーズしたいアドレス 0x3000~0x3fff
戻り値	1:イレーズ完了 0:異常終了
使用例	writeDataFlash(0x3010); /* 0x3010 番地のあるブロック(ブロック A)の内容をイレーズします */
イレーズする範囲について	イレーズする範囲は、ブロック単位になります。例えば、引数に 0x3010 番地を指定したとき、イレーズされるのは 0x3010 番地だけではなく、該当するブロック A である 0x3000~0x33ff 番地のすべてがイレーズされます。0x3010 番地以外の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 に読み込んでイレーズし、改めて書き込むようにしてください。

27.7 プログラムの解説「data_flash.c」

27.7.1 インクルード部分

「data_flash_lib.c」ファイル内の関数を使うために、専用のヘッダファイルをインクルードします。

```

18 : /*=====*/
19 : /* インクルード */
20 : /*=====*/
21 : #include "sfr_r835a.h" /* R8C/35A SFRの定義ファイル */
22 : #include "data_flash_lib.h" /* データフラッシュライブラリ */
    
```

data_flash_lib.h	ファイルがあるフォルダの位置は、「C:¥Workspace¥common_r8c35a」です。 このファイルを取り込む(インクルード)することにより、readDataFlash 関数、writeDataFlash 関数が使えるようになります。
------------------	---

27.7.2 変数領域

```

24 : /*=====*/
25 : /* シンボル定義 */
26 : /*=====*/
27 : /* データフラッシュ関連 */
28 : #define DF_SIZE 64 /* 読み書きサイズ */
29 :
30 : #define DF_CHECK 0 /* データフラッシュチェック */
31 : #define DF_DATA 1 /* データ */
中略
42 : /*=====*/
43 : /* グローバル変数の宣言 */
44 : /*=====*/
45 : /* データフラッシュ関連 */
46 : signed char data_buff[ DF_SIZE ]; /* 一時保存エリア */

```

24 行～31 行でシンボル定義をしています。

DF_SIZE	データフラッシュから読み込んだり、書き込んだりするサイズです。今回は 64 バイトのデータを読み書きします。
DF_CHECK	データフラッシュから読み込むとき、正しいデータを保存しているかチェックするための領域です。64 バイトのデータのうち、0 個目をチェック用にします。
DF_DATA	データフラッシュからデータを読み込むときの場所です。今回は 1 個目です (0 から始まるので 2 番目となります)。

42 行～46 行で、グローバル変数の宣言をしています。

data_buff	データフラッシュから読み込んだデータを格納したり、書き込むデータを格納しておく配列です。サイズは、DF_SIZE 個、すなわち 64 個になります。
-----------	--

27.7.3 main 関数(データフラッシュから読み込み)

```

51 : void main( void )
52 : {
53 :     int ret;
54 :
55 :     init(); /* 初期化 */
56 :
57 :     readDataFlash( 0x3000, data_buff, DF_SIZE );
58 :
59 :     if( data_buff[DF_CHECK] != 0x35 ) {
60 :         data_buff[DF_CHECK] = 0x35;
61 :         data_buff[DF_DATA] = 0;
62 :     }
63 :
64 :     led_out( data_buff[DF_DATA] );
65 :
66 :     while( !pushsw_get() );

```

55 行	init 関数を実行します。 init 関数では、ポートの入出力設定を行います。
57 行	readDataFlash 関数で、 ・0x3000 番地から ・data_buff 配列に ・DF_SIZE 個(64 個)のデータを 読み込みます。
59 行	data_buff 配列の DF_CHECK 番目(0 番目)のデータが 0x35 かチェックします。
60,61 行	59 行のチェックで、0x35 でなければ、初めて 0x3000 番地のデータフラッシュからデータを読み込んだと判断して、 ・DF_CHECK 番目(0 番目)に 0x35 を ・DF_DATA 番目(1 番目)に 0 を 設定します。 ちなみに 0x35 なら、DF_DATA には前回保存したデータが読み込まれます。
64 行	マイコンボードの LED に、前回保存したデータを表示します。もし、今回初めて 0x3000 番地を読み込んだ場合は、61 行目に設定している 0 を表示します(LED は点灯しません)。
66 行	マイコンボードのプッシュスイッチが押されるまで、66 行を繰り返し続けます。

27.7.4 main 関数(データフラッシュに書き込み)

68 :	<code>data_buff[DF_DATA] = dipsw_get();</code>
69 :	<code>blockEraseDataFlash(0x3000);</code>
70 :	<code>ret = writeDataFlash(0x3000, data_buff, DF_SIZE);</code>

68 行	マイコンボードのディップスイッチの値を、data_buff 配列の DF_DATA 番目(1 番目)に保存します。
69 行	blockEraseDataFlash 関数で、0x3000 番地を含むブロックをイレーズします。 注意点は、0x3000 番地のブロックであるブロック A のデータをすべてイレーズすることです。 ブロック A は 0x3000~0x33ff 番地なので、このアドレスのデータがイレーズされます。この範囲のアドレスで消したくないデータは、内蔵 RAM に保存しておきます。ただし、内蔵 RAM は 4KB しかありませんので、RAM の容量がオーバーしないよう気をつけてください。
70 行	writeDataFlash 関数で、 ・データフラッシュの 0x3000 番地に ・data_buff 配列の内容を ・DF_SIZE 個(64 個) 書き込みます。 書き込みが正常にできれば変数 ret には 1 が、書き込みできなければ 0 が入ります。

27.7.5 main 関数(無限ループ部分)

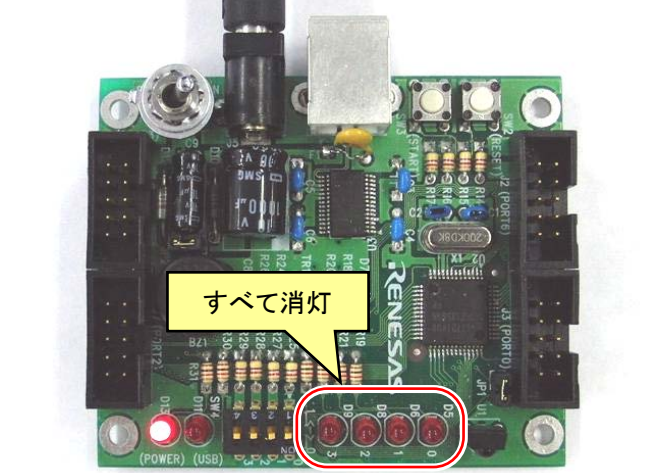
```

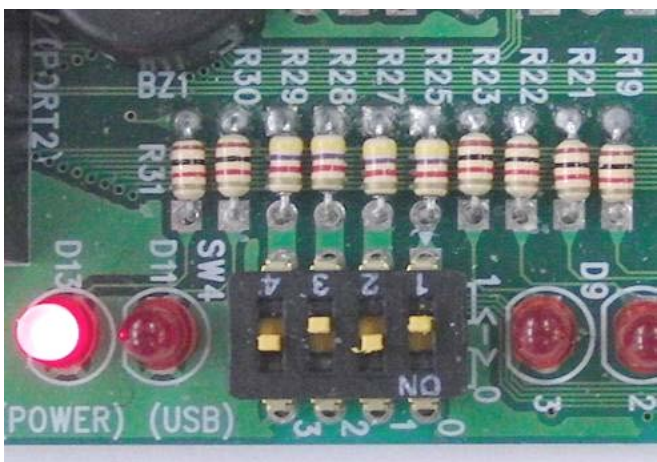
72 :     if( ret == 0 ) {
73 :         /* 書き込みエラーなら */
74 :         while( 1 ) {
75 :             led_out( 0xc );
76 :             timer( 100 );
77 :             led_out( 0x3 );
78 :             timer( 100 );
79 :         }
80 :     }
81 :
82 :     /* 書き込みが正常にできたなら */
83 :     while( 1 ) {
84 :         led_out( 0x5 );
85 :         timer( 200 );
86 :         led_out( 0xa );
87 :         timer( 200 );
88 :     }
89 : }

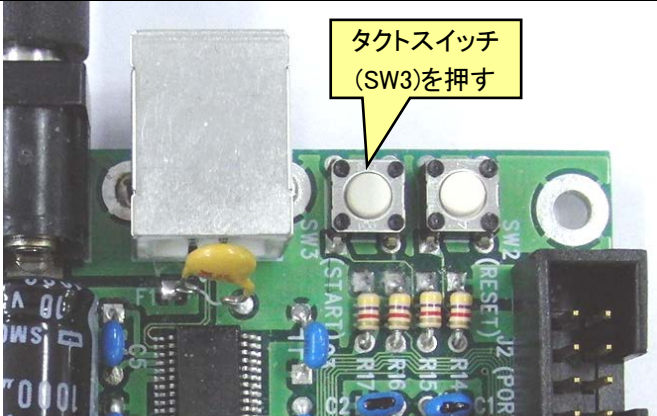
```

72 行	正常に書き込みができたかチェックします。ret 変数が 0 なら、エラーです。
73～ 80 行	書き込みがエラーなら、マイコンボードの LED に 0xc と 0x3 を 0.1 秒ごとに表示させます。
83～ 88 行	正常に書き込みができれば、マイコンボードの LED に 0x5 と 0xa を 0.2 秒ごとに表示させます。

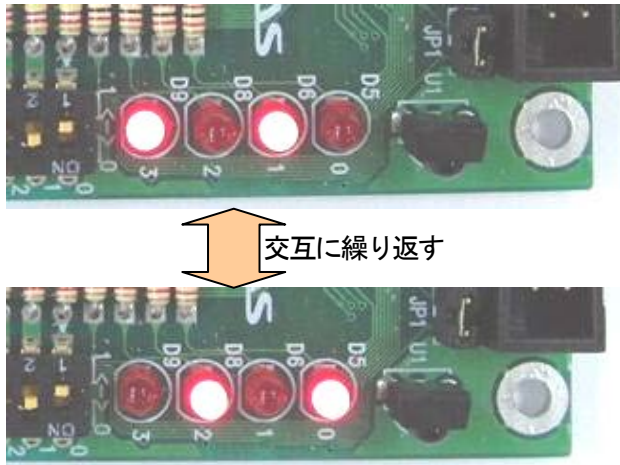
27.8 実習手順

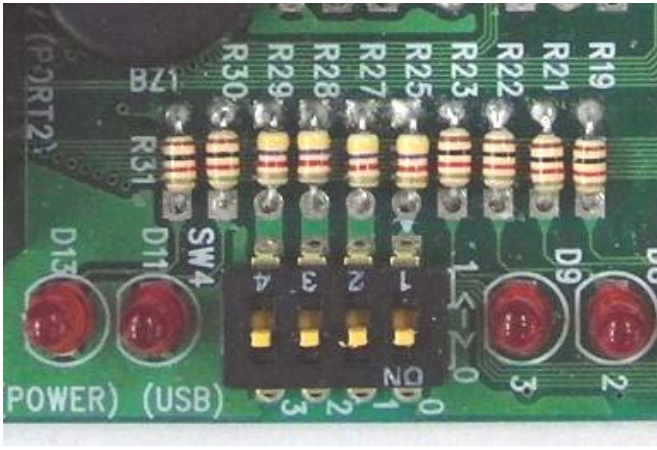
1	 <p>すべて消灯</p>	<ul style="list-style-type: none">• 「data_flash.mot」を書き込みます。• マイコンボードの電源を入れます。• リセットスイッチを押します。 <p>マイコンボードのLED (D5～D9)は消灯しています。</p> <p>※前回、本実習をした場合は、前回に保存した値が表示されます。</p>
---	--	--

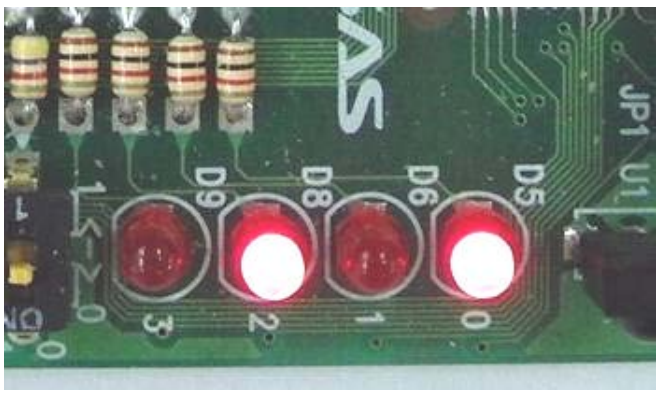
2		<p>マイコンボードのディップスイッチを、好きな状態に変えます。</p> <p>写真は、左から、「下上下上」です。</p> <p>下が「0」、上が「1」なので、左から「0101」になります。ディップスイッチの状態を覚えておいてください。</p>
---	--	--

3	 <p>タクトスイッチ (SW3)を押す</p>	<p>タクトスイッチ(SW3)を押します。</p> <p>押した瞬間、ディップスイッチの値をデータフラッシュに書き込みます。</p>
---	---	--

27. データフラッシュ(プロジェクト:data_flash)

4		<p>LED が"1010"と"0101"の点灯を交互に繰り返し、データフラッシュに書き込んだことを知らせます。</p>
---	---	--

5		<p>マイコンボードの電源を切って、ディップスイッチを先ほどとは違う値にします。</p>
---	--	--

6		<p>再度、電源を入れると、先ほどタクトスイッチを押したときのディップスイッチ値をデータフラッシュから呼び出して、LED に表示します。</p>
---	---	--

27.9 演習

- (1) マイコンボードのポート6(J2)に実習基板 Ver.2 の LED 部を、ポート0(J3)に実習基板 Ver.2 のディップスイッチ部を接続し(結線図は、プロジェクト「io」を参照)、実習基板 Ver.2 のディップスイッチ部の値を保存、電源を入れると実習基板 Ver.2 の LED 部に出力するようにしなさい。

28. タイマ RC によるブザー制御(PWM 波形出力)(プロジェクト:timer_rc_pwm)

28.1 概要

本章では、タイマ RC を使って任意の周期、ON 幅の波形を出力する方法を説明します。タイマ RC の初期設定後は、プログラムが関与しなくても PWM 波形を出力し続けます。プログラムでは、PWM 波形出力処理以外の処理をすることができます。今回はこの PWM 波形を使って、ブザーを鳴らします。

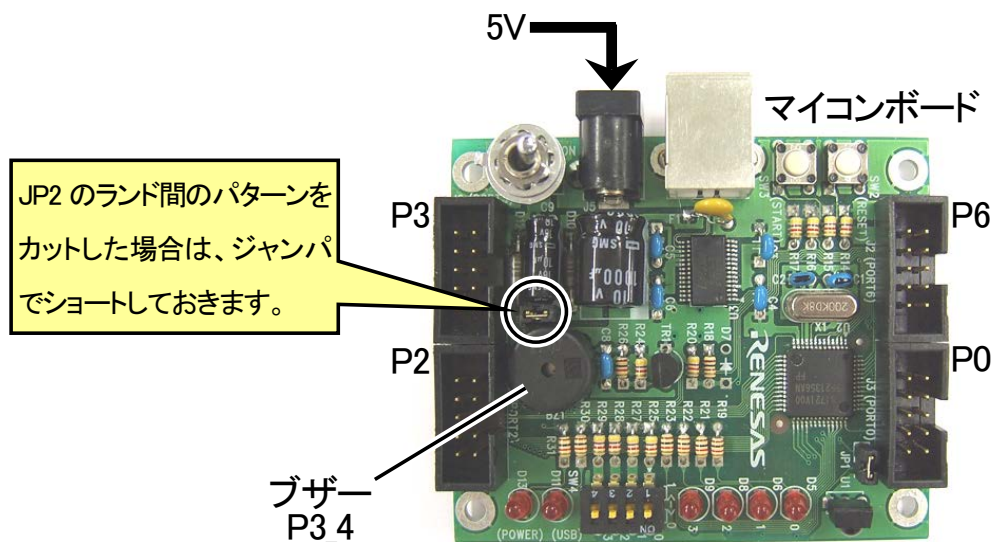
28.2 接続

■使用ポート

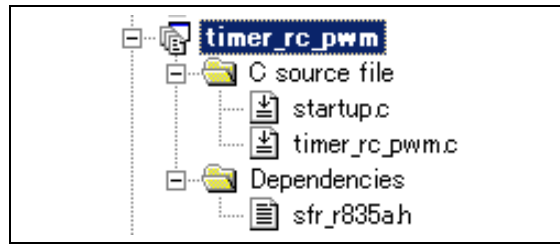
マイコンのポート	接続内容
P3_4 (J6)	マイコンボードだけで実習できます。JP2 のランド間のパターンをカットした場合は、ジャンパでショートしておきます。

■接続

マイコンボードだけで実習できます。



28.3 プロジェクトの構成



	ファイル名	内容
1	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。このファイルは共通で、どのプロジェクトもこのファイルから実行されます。
2	timer_rc_pwm.c	実際に制御するプログラムが書かれています。R8C/35A の内蔵周辺機能(SFR)の初期化も行います。
3	sfr_r835a.h	R8C/35A マイコンの内蔵周辺機能を制御するためのレジスタ(Special Function Registers)を定義したファイルです。

28.4 プログラム「timer_rc_pwm.c」

```

1 : /******
2 : /* 対象マイコン R8C/35A */
3 : /* ファイル内容 タイマRCによるPWM出力 */
4 : /* バージョン Ver. 1.20 */
5 : /* Date 2010.04.19 */
6 : /* Copyright ルネサスマイコンカーラリー事務局 */
7 : /* 日立インターメディアックス株式会社 */
8 : /******
9 : /*
10 : 入力：なし
11 : 出力：TRCIO端子 (P3_4) にブザーを接続
12 :
13 : TRCIO端子 (P3_4) に接続したブザーから、音を鳴らします。
14 : */
15 :
16 : /*=====*/
17 : /* インクルード */
18 : /*=====*/
19 : #include "sfr_r835a.h" /* R8C/35A SFRの定義ファイル */
20 :
21 : /*=====*/
22 : /* シンボル定義 */
23 : /*=====*/
24 : /* 3オクターブ目の音階 */
25 : #define DO_3 38226 /* ド */
26 : #define DOU_3 36081 /* ド# */
27 : #define RE_3 34056 /* レ */
28 : #define REU_3 32144 /* レ# */
29 : #define MI_3 30340 /* ミ */
30 : #define FA_3 28637 /* ファ */
31 : #define FAU_3 27030 /* ファ# */
32 : #define SO_3 25513 /* ソ */
33 : #define SOU_3 24081 /* ソ# */
34 : #define RA_3 22729 /* ラ */
35 : #define RAU_3 21454 /* ラ# */
36 : #define SI_3 20250 /* シ */
37 :
38 : /* 4オクターブ目の音階 */
39 : #define DO_4 19113 /* ド */
40 : #define DOU_4 18040 /* ド# */
41 : #define RE_4 17028 /* レ */
42 : #define REU_4 16072 /* レ# */
43 : #define MI_4 15170 /* ミ */
44 : #define FA_4 14319 /* ファ */
45 : #define FAU_4 13515 /* ファ# */
46 : #define SO_4 12756 /* ソ */
47 : #define SOU_4 12041 /* ソ# */
48 : #define RA_4 11365 /* ラ */
49 : #define RAU_4 10727 /* ラ# */
50 : #define SI_4 10125 /* シ */
51 :
52 : /* 5オクターブ目の音階 */
53 : #define DO_5 9557 /* ド */
54 : #define DOU_5 9020 /* ド# */
55 : #define RE_5 8514 /* レ */
56 : #define REU_5 8036 /* レ# */
57 : #define MI_5 7585 /* ミ */
58 : #define FA_5 7159 /* ファ */
59 : #define FAU_5 6758 /* ファ# */
60 : #define SO_5 6378 /* ソ */
61 : #define SOU_5 6020 /* ソ# */
62 : #define RA_5 5682 /* ラ */
63 : #define RAU_5 5363 /* ラ# */
64 : #define SI_5 5062 /* シ */
65 :
66 : #define TEMPO 60 /* テンポ */
67 :
68 : /*=====*/
69 : /* プロトタイプ宣言 */
70 : /*=====*/
71 : void init( void );
72 : void beep( unsigned int tone );
73 :
74 : /*=====*/
75 : /* グローバル変数の宣言 */
76 : /*=====*/
77 : unsigned long cnt_rb; /* タイマRB用 */
78 : int music_dim; /* 音楽データ配列の位置 */
79 : int music_flag; /* 音楽データならすかどうか */
80 :

```

28. タイマ RC によるブザー制御(PWM 波形出力)(プロジェクト:timer_rc_pwm)

```

81 : const int music_data[][2] = {          /* 大きな古時計 音楽データ */
82 : /*
83 : 長さは、四分音符を4として、二分音符は8、八分音符は2となります。
84 : 休符も同様に、四分休符を4として、二部休符は8、八分休符は2となります。
85 : */
86 : /* 音階, 長さ */
87 : 0, 0, /* スタート*/
88 :
89 : RE_4, 4, /* お */
90 : SO_4, 4, /* お */
91 : FAU_4, 2, /* き */
92 : SO_4, 2, /* な */
93 : RA_4, 4, /* のつ */
94 : SO_4, 2, /* ぼ */
95 : RA_4, 2, /* の */
96 : SI_4, 2, /* ふ */
97 : SI_4, 2, /* る */
98 : DO_5, 2, /* ど */
99 : SI_4, 2, /* け */
100 : MI_4, 4, /* い */
101 : RA_4, 2, /* お */
102 : RA_4, 2, /* じ */
103 : SO_4, 4, /* い */
104 : SO_4, 2, /* さ */
105 : SO_4, 2, /* ん */
106 : FAU_4, 4, /* の */
107 : MI_4, 2, /* と */
108 : FAU_4, 2, /* け */
109 : SO_4, 10, /* い */
110 : 0, 2,
111 :
112 : RE_4, 2, /* ひや */
113 : RE_4, 2, /* く */
114 : SO_4, 4, /* ねん */
115 : FAU_4, 2, /* い */
116 : SO_4, 2, /* つ */
117 : RA_4, 4, /* も */
118 : SO_4, 2, /* う */
119 : RA_4, 2, /* ご */
120 : SI_4, 4, /* い */
121 : DO_5, 2, /* て */
122 : SI_4, 2, /* いた */
123 : MI_4, 4, /* た */
124 : RA_4, 2, /* ご */
125 : RA_4, 2, /* じ */
126 : SO_4, 4, /* ま */
127 : SO_4, 2, /* ん */
128 : SO_4, 2, /* の */
129 : FAU_4, 4, /* と */
130 : MI_4, 2, /* け */
131 : FAU_4, 2, /* い */
132 : SO_4, 10, /* さ */
133 : 0, 2,
134 :
135 : SO_4, 2, /* お */
136 : SI_4, 2, /* じ */
137 : RE_5, 4, /* い */
138 : SI_4, 2, /* さ */
139 : RA_4, 2, /* ん */
140 : SO_4, 4, /* の */
141 : FAU_4, 2, /* う */
142 : SO_4, 2, /* ま */
143 : RA_4, 2, /* れ */
144 : SO_4, 2, /* た */
145 : FAU_4, 2, /* あ */
146 : MI_4, 2, /* さ */
147 : RE_4, 4, /* に */
148 : SO_4, 2, /* か */
149 : SI_4, 2, /* っ */
150 : RE_5, 4, /* て */
151 : SI_4, 2, /* き */
152 : RA_4, 2, /* た */
153 : SO_4, 4, /* と */
154 : FAU_4, 2, /* け */
155 : SO_4, 2, /* い */
156 : RA_4, 10, /* さ */
157 : 0, 2,
158 :
159 : RE_4, 2, /* い */
160 : SO_4, 2, /* ま */
161 : SO_4, 2, /* は */
162 : 0, 4,
163 : RA_4, 2, /* もう */
164 : 0, 2,
165 : 0, 4,
166 : SI_4, 2, /* う */
167 : SI_4, 2, /* ご */
168 : DO_5, 2, /* か */
169 : SI_4, 2, /* ない */
170 : MI_4, 4, /* い */
171 : RA_4, 2, /* そ */

```

```

172 :      RA_4,  2, /* の */
173 :      SO_4,  8, /* と */
174 :      FAU_4, 8, /* け */
175 :      SO_4,  8, /* い */
176 :      0,     2,
177 :
178 :      RE_4,  2, /* ひや */
179 :      RE_4,  2, /* く */
180 :      SO_4,  4, /* ねん */
181 :      RE_4,  2, /* や */
182 :      RE_4,  2, /* す */
183 :      MI_4,  2, /* ま */
184 :      RE_4,  2, /* ず */
185 :      RE_4,  4, /* に */
186 :      SI_3,  2, /* ちつく */
187 :      0,     2,
188 :      RE_4,  2, /* たつく */
189 :      0,     2,
190 :      SI_3,  2, /* ちつく */
191 :      0,     2,
192 :      RE_4,  2, /* たつく */
193 :      RE_4,  2, /* お */
194 :      SO_4,  4, /* じー */
195 :      RE_4,  2, /* さん */
196 :      RE_4,  2, /* と */
197 :      MI_4,  4, /* いつ */
198 :      RE_4,  2, /* しよ */
199 :      RE_4,  2, /* に */
200 :      SI_3,  2, /* ちつく */
201 :      0,     2,
202 :      RE_4,  2, /* たつく */
203 :      0,     2,
204 :      SI_3,  2, /* ちつく */
205 :      0,     2,
206 :      RE_4,  2, /* たつく */
207 :
208 :      RE_4,  2, /* い */
209 :      SO_4,  2, /* ま */
210 :      SO_4,  2, /* は */
211 :      0,     4,
212 :      RA_4,  2, /* もう */
213 :      0,     2,
214 :      0,     4,
215 :      SI_4,  2, /* う */
216 :      SI_4,  2, /* ご */
217 :      DO_5,  2, /* か */
218 :      SI_4,  2, /* な */
219 :      MI_4,  4, /* い */
220 :      RA_4,  2, /* そ */
221 :      RA_4,  2, /* の */
222 :      SO_4,  8, /* と */
223 :      FAU_4, 8, /* け */
224 :      SO_4, 12, /* い */
225 :      0,     4,
226 :
227 :      -1,    0 /* 終了 */
228 : };
229 :
230 : /*****
231 : /* メインプログラム */
232 : *****/
233 : void main( void )
234 : {
235 :     unsigned char d;
236 :
237 :     init(); /* 初期化 */
238 :     asm(" fset I "); /* 全体の割り込み許可 */
239 :
240 :     music_flag = 1; /* 音楽スタート */
241 :
242 :     while( 1 ) {
243 :     }
244 : }
245 :
246 : /*****
247 : /* ブザーを鳴らす */
248 : /* 引数 音階のPWM値 */
249 : *****/
250 : void beep( unsigned int tone )
251 : {
252 :     if( tone ) {
253 :         trcmr  &= 0x7f; /* TRCのカウント停止 */
254 :         trc    = tone - 1;
255 :         trecra = tone - 1; /* 周期設定 */
256 :         trecrc = tone / 2 - 1; /* ON幅設定 */
257 :         trcmr |= 0x80; /* TRCのカウント開始 */
258 :     } else {

```

28. タイマ RC によるブザー制御(PWM 波形出力)(プロジェクト:timer_rc_pwm)

```

259 :         trcmr  &= 0x7f;          /* TRCのカウンタ停止          */
260 :         trc    = 0;
261 :         trcgra = 0;              /* 周期設定                    */
262 :         tregrc = 0;              /* ON幅設定                    */
263 :         trcmr  |= 0x80;          /* TRCのカウンタ開始          */
264 :     }
265 : }
266 :
267 : /*****
268 : /* R8C/35A スペシャルファンクションレジスタ(SFR)の初期化
269 : /*****
270 : void init( void )
271 : {
272 :     int i;
273 :
274 :     /* クロックをXINクロック(20MHz)に変更 */
275 :     prc0 = 1;                    /* プロテクト解除              */
276 :     cm13 = 1;                    /* P4_6, P4_7をXIN-XOUT端子にする */
277 :     cm05 = 0;                    /* XINクロック発振              */
278 :     for(i=0; i<50; i++ );        /* 安定するまで少し待つ(約10ms) */
279 :     ocd2 = 0;                    /* システムクロックをXINにする */
280 :     prc0 = 0;                    /* プロテクトON                 */
281 :
282 :     /* ポートの入出力設定 */
283 :     prc2 = 1;                    /* PDOのプロテクト解除          */
284 :     pd0 = 0xe0;                  /* 7-5:LED 4:MicroSW 3-0:Sensor */
285 :     p1  = 0x0f;                  /* 3-0:LEDは消灯                */
286 :     pd1 = 0xdf;                  /* 5:RXD0 4:TXD0 3-0:LED        */
287 :     pd2 = 0xfe;                  /* 0:PushSW                      */
288 :     pd3 = 0xfb;                  /* 4:Buzzer 2:IR                 */
289 :     pd4 = 0x83;                  /* 7:XOUT 6:XIN 5-3:DIP SW 2:VREF */
290 :     pd5 = 0x40;                  /* 7:DIP SW                       */
291 :     pd6 = 0xff;
292 :
293 :     /* タイマRBの設定 */
294 :     /* 割り込み周期 = 1 / 20[MHz] * (TRBPRE+1) * (TRBPR+1)
295 :     = 1 / (20*10-6) * 200 * 100
296 :     = 0.001[s] = 1[ms]
297 :     */
298 :     trbpre = 200-1;              /* プリスケアラレジスタ          */
299 :     trbpr  = 100-1;              /* プライマリレジスタ            */
300 :     trbmr  = 0x00;              /* 動作モード、分周比設定        */
301 :     trbic  = 0x07;              /* 割り込み優先レベル設定        */
302 :     trbcr  = 0x01;              /* カウンタ開始                  */
303 :
304 :     /* タイマRC PWMモード設定 */
305 :     trcmr  = 0x0a;              /* TRCIO端子はPWM出力            */
306 :     trccr1 = 0xa0;              /* カウンタソース、初期出力の設定 */
307 :     trccr2 = 0x00;              /* 出力レベルの設定              */
308 :     trcoer = 0x0b;              /* TROIOC出力許可                */
309 :     trcpsr0 = 0x00;             /* TRCIOA, B端子の設定           */
310 :     trcpsr1 = 0x02;             /* TRCIO, D端子の設定            */
311 :     tregra = 0;                 /* 周期設定                      */
312 :     tregrc = 0;                 /* ON幅設定                      */
313 : }
314 :
315 : /*****
316 : /* タイマRB 割り込み処理
317 : /*****
318 : #pragma interrupt intTRB(vect=24)
319 : void intTRB( void )
320 : {
321 :     cnt_rb++;
322 :
323 :     if( music_flag ) {
324 :         /* ブザー処理 */
325 :         if( cnt_rb >= 15000L * music_data[music_dim][1] / TEMPO ) {
326 :             /* 次の音階をならす */
327 :             cnt_rb = 0;
328 :             music_dim++;
329 :             if( music_data[music_dim][0] == -1 ) {
330 :                 /* -1なら終了 */
331 :                 beep( 0 );
332 :                 music_flag = 0;
333 :             } else {
334 :                 /* -1でないなら次の音階セット */
335 :                 beep( music_data[music_dim][0] );
336 :             }
337 :         }
338 :     }
339 : }
340 :
341 : /*****
342 : /* end of file
343 : /*****

```

28.5 プログラムの解説

28.5.1 init 関数(タイマ RC の設定)

304 :	/* タイマRC PWMモード設定 */	
305 :	trcmr = 0x0a;	/* TRCIOC端子はPWM出力 */
306 :	trccr1 = 0xa0;	/* カウントソース, 初期出力の設定*/
307 :	trccr2 = 0x00;	/* 出力レベルの設定 */
308 :	trcoer = 0x0b;	/* TRCIOC出力許可 */
309 :	trcpsr0 = 0x00;	/* TRCIOC, B端子の設定 */
310 :	trcpsr1 = 0x02;	/* TRCIOC, D端子の設定 */
311 :	trcgra = 0;	/* 周期設定 */
312 :	trcgrc = 0;	/* ON幅設定 */

タイマ RC カウンタ(TRC)のカウント開始は、beep 関数内で行っています。

(1) タイマ RC とは

R8C/35A には、タイマ RC というタイマが 1 個内蔵されています。タイマ RC には、次の 3 種類のモードがあります。今回は、「**PWM モード**」を使います。

モード	内容
タイマモード	タイマモードには、次の 2 つの機能があります。 ・インプットキャプチャ機能 外部信号をトリガにしてカウンタの値をレジスタに取り込む機能 ・アウトプットコンペア機能 カウンタとレジスタの値の一致を検出する機能(検出時に端子出力変更可能)
PWM モード	任意の幅のパルスを連続して出力するモード
PWM2 モード	トリガからウェイト時間をおいて、ワンショット波形または PWM 波形を出力するモード

※インプットキャプチャ機能、アウトプットコンペア機能、PWM モードは、1 端子ごとに機能とモードを選択できます。

※PWM2 モードは、カウンタやレジスタを組み合わせることで波形を出力します。端子の機能はモードによって決まります。

タイマ RC の端子構成を下表に示します。

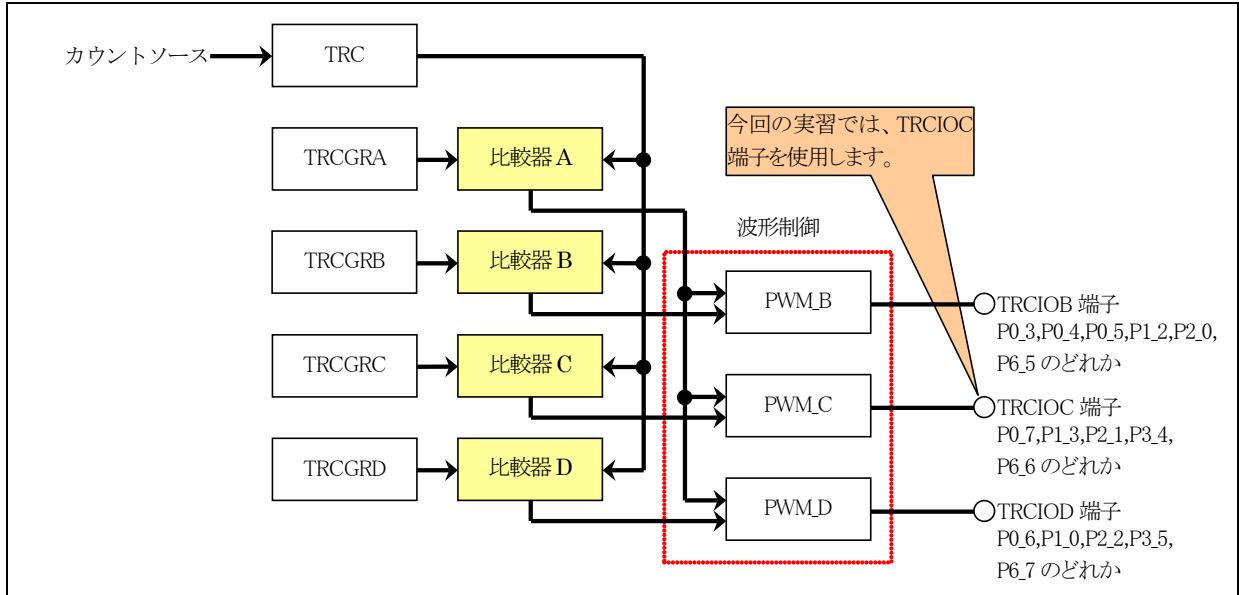
端子名	割り当てる端子	入出力	機能
TRCIOA	P0_0, P0_1, P0_2 または P1_1	入出力	モードによって機能が異なります。詳細は書くモードを参照してください。
TRCIOB	P0_3, P0_4, P0_5, P1_2, P2_0 または P6_5		
TRCIOC	P0_7, P1_3, P2_1, P3_4 または P6_6		
TRCIOD	P0_6, P1_0, P2_2, P3_5 または P6_7		
TRCCLK	P1_4 または P3_3	入力	外部クロック入力
TRCTRG	P0_0, P0_1, P0_2 または P1_1	入力	PWM2 モードの外部トリガ入力

(2) タイマ RC のブロック図

PWM モードのブロック図を下記に示します。

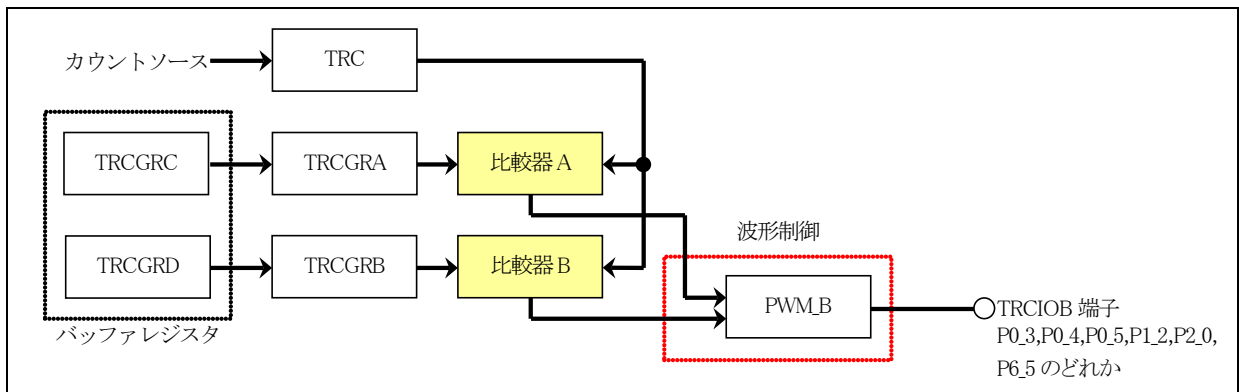
●バッファレジスタを使わない場合

同一周期の PWM 波形を 3 本出力できます。ただし、周期や PWM の ON 幅を設定するタイミングによっては、波形が乱れる場合がありますので、プログラムで対処が必要です。今回の実習は、この内容です。



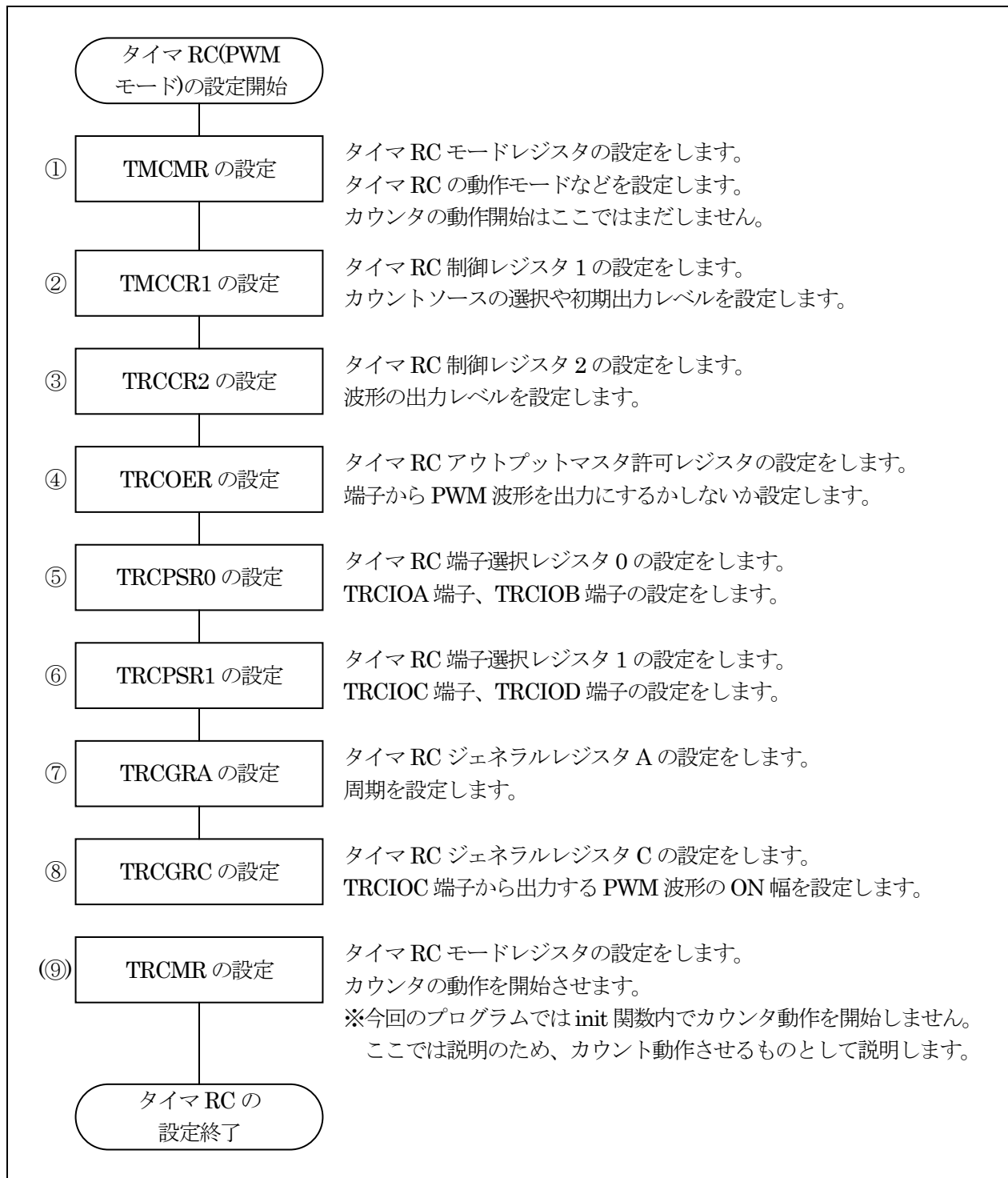
●バッファレジスタを使う場合

PWM 波形を 1 本出力できます。周期や PWM の ON 幅を設定するタイミングはバッファレジスタに設定するので、プログラムでの対処は入りません。



(3) タイマ RC の設定 (PWM モード)

今回は、タイマ RC を PWM モードで使用して PWM 波形を出力します。バッファレジスタは使いません。レジスタの設定手順を下記に示します。



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

タイマ RC のモードを設定します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7	TRC カウント開始ビット tstart_trcmr	0:カウント停止 1:カウント開始 カウント開始は最後にします。今は"0"を設定します。	0
bit6		"0"を設定	0
bit5	TRCGRD レジスタ機能選択ビ ット bfd_trcmr	0:ジェネラルレジスタ 1:TRCGRB レジスタのバッファレジスタ TRCGRD をジェネラルレジスタとして使用します。	0
bit4	TRCGRC レジスタ機能選択ビ ット(注 2) bfc_trcmr	0:ジェネラルレジスタ 1:TRCGRA レジスタのバッファレジスタ TRCGRC をジェネラルレジスタとして使用します。	0
bit3	PWM2 モード選択ビット pwm2_trcmr	0:PWM2 モード 1:タイマモードまたは PWM モード 今回は PWM モードで使用します。	1
bit2	TRCIOD PWM モード選択ビ ット(注 1) pwm_d_trcmr	0:タイマモード 1:PWM モード TRCIOD は使用しませんので"0"を設定しておきます。	0
bit1	TRCIOC PWM モード選択ビ ット(注 1) pwm_c_trcmr	0:タイマモード 1:PWM モード TRCIOC を PWM モードとして使用します。	1
bit0	TRCIOB PWM モード選択ビ ット(注 1) pwm_b_trcmr	0:タイマモード 1:PWM モード TRCIOB は使用しませんので"0"を設定しておきます。	0

注 1. これらのビットは PWM2 ビットが"1"(タイマモードまたは PWM モード)のとき有効です。

注 2. PWM2 モードでは BFC ビットを"0"(ジェネラルレジスタ)にしてください。

TRCMR レジスタの PWM2 モード時の注意事項は「19.9.6 PWM2 モード時の TRCMR レジスタ」を参照してください。

タイマ RC モードレジスタ(TRCMR)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	1	0	1	0
16 進数	0				a			

②タイマ RC 制御レジスタ 1(TRCCR1:Timer RC control register 1)の設定をします。

カウントソースの選択や初期出力レベルを設定します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7	TRC カウンタクリア選択ビット cclr_trccr1	0:クリア禁止(フリーランニング動作) 1:インプットキャプチャまたは TRCGRA のコンペア一致で TRC カウンタをクリア PWM モードで使用するときは"1"を設定します。	1
bit6~4	カウントソース選択ビット(注 1) bit6:tck2_trccr1 bit5:tck1_trccr1 bit4:tck0_trccr1	000:f1 (1/20MHz=50ns) 001:f2 (2/20MHz=100ns) 010:f4 (4/20MHz=200ns) 011:f8 (8/20MHz=400ns) 100:f32 (32/20MHz=1600ns) 101:TRCCLK 入力の立ち上がりエッジ 110:fOCO40M (高速オンチップオシレータ 40MHz=今回は未接続) 111:fOCO-F(注 2) (高速オンチップオシレータを FRA2 で分周したクロック=今回は未接続) タイマ RC カウンタ(TRC)がカウントアップする時間を設定します。今回は"010"を設定します。TRC は、200ns ごとに+1していきます。	010
bit3	TRCIOD 出力レベル選択ビット(注 1、2) tod_trccr1	0:初期出力はアクティブでないレベル 1:初期出力はアクティブレベル 今回は、TRCIOD 端子は使用しませんので"0"を設定します。	0
bit2	TRCIOC 出力レベル選択ビット(注 1、2) toc_trccr1	0:初期出力はアクティブでないレベル 1:初期出力はアクティブレベル TRCIOC 端子からの初期出力はアクティブでないレベルに設定します。	0
bit1	TRCIOB 出力レベル選択ビット(注 1、2) tob_trccr1	0:初期出力はアクティブでないレベル 1:初期出力はアクティブレベル 今回は、TRCIOB 端子は使用しませんので"0"を設定します。	0
bit0	TRCIOA 出力レベル選択ビット(注 1、2) toa_trccr1	PWM モードでは無効、"0"を設定	0

注 1. TRCMR レジスタの TSTART ビットが"0"(カウント停止)のとき、書いてください。

注 2. 端子の機能が波形出力の場合(ハードウェアマニュアル「7.5 ポートの設定」参照)、TRCCR1 レジスタを設定したとき、初期出力レベルが出力されます。

注 3. fOCO-F を選択するときは、CPU クロックより速いクロック周波数に fOCO-F を設定してください。

タイマ RC 制御レジスタ 1(TRCCR1)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	1	0	1	0	0	0	0	0
16 進数	a				0			

※タイマ RC 制御レジスタ 1(TRCCR1)のカウンソース選択ビットの設定方法

タイマ RC 制御レジスタ 1(TRCCR1)のカウンソース選択ビット(bit6~4)で、タイマ RC カウンタ(TRC)がどのくらいの間隔でカウントアップするか設定します。TRC は、0 からスタートして最大 65,535 までカウントアップします。65,535 の次は 0 に戻ります。PWM の周期や ON 幅は TRC の値を基準にするので、カウントアップする時間×65,536 以上の時間を設定することができません。

タイマ RC 制御レジスタ 1(TRCCR1)のカウンソース選択ビットの値と、周期の関係を下記に示します。

bit6~4	内容
000	タイマ RC カウンタ(TRC)がカウントアップする時間を、f1 に設定します。時間は、 $f1/20\text{MHz}=1/20\text{MHz}=50\text{ns}$ 設定できる PWM 周期の最大は、 $50\text{ns} \times 65,536 = \mathbf{3.2768\text{ms}}$ よって、この時間以内の PWM 周期を設定する場合は"000"を設定、これ以上の PWM 周期を設定したい場合は次以降の値を検討します。
001	タイマ RC カウンタ(TRC)がカウントアップする時間を、f2 に設定します。時間は、 $f2/20\text{MHz}=2/20\text{MHz}=100\text{ns}$ 設定できる PWM 周期の最大は、 $100\text{ns} \times 65,536 = \mathbf{6.5536\text{ms}}$ よって、この時間以内の PWM 周期を設定する場合は"001"を設定、これ以上の PWM 周期を設定したい場合は次以降の値を検討します。
010	タイマ RC カウンタ(TRC)がカウントアップする時間を、f4 に設定します。時間は、 $f4/20\text{MHz}=4/20\text{MHz}=200\text{ns}$ 設定できる PWM 周期の最大は、 $200\text{ns} \times 65,536 = \mathbf{13.1072\text{ms}}$ よって、この時間以内の PWM 周期を設定する場合は"010"を設定、これ以上の PWM 周期を設定したい場合は次以降の値を検討します。
011	タイマ RC カウンタ(TRC)がカウントアップする時間を、f8 に設定します。時間は、 $f8/20\text{MHz}=8/20\text{MHz}=400\text{ns}$ 設定できる PWM 周期の最大は、 $400\text{ns} \times 65,536 = \mathbf{26.2144\text{ms}}$ よって、この時間以内の PWM 周期を設定する場合は"011"を設定、これ以上の PWM 周期を設定したい場合は次以降の値を検討します。
100	タイマ RC カウンタ(TRC)がカウントアップする時間を、f32 に設定します。時間は、 $f32/20\text{MHz}=32/20\text{MHz}=1600\text{ns}$ 設定できる PWM 周期の最大は、 $1600\text{ns} \times 65,536 = \mathbf{104.8576\text{ms}}$ よって、この時間以内の PWM 周期を設定する場合は"100"を設定します。 これ以上の PWM 周期を設定することはできません。 これ以上の PWM 周期を設定しなくても良いように、回路側を工夫してください。

今回、いちばん低い「ド」の音階は、130.8Hz です(詳しくは後述)。周波数は、

$$1/130.8=7.65[\text{ms}]$$

です。この周波数を設定できる PWM 周期に設定します。

- ①"000"の設定…最大の PWM 周期は 3.2768ms、今回設定したい 7.65ms の周期を設定できないので不可
- ②"001"の設定…最大の PWM 周期は 6.5536ms、今回設定したい 7.65ms の周期を設定できないので不可
- ③"010"の設定…最大の PWM 周期は 13.1072ms、今回設定したい 7.65ms の周期を設定できるので OK

よって、"010"を設定します。

③タイマ RC 制御レジスタ 2(TRCCR2:Timer RC control register 2)の設定をします。

波形の出力レベルを設定します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7,6	TRCTRГ 入力エッジ選択ビット(注 3) bit7:tceg1_trccr2 bit6:tceg0_trccr2	00:TRCTRГ からのトリガ入力を禁止 01:立ち上がりエッジを選択 10:立ち下がりエッジを選択 11:立ち上がり/立ち下がり両エッジを選択 このビットは、PWM2 モード時に使用します。今回は使用しません。"00"を設定します。	00
bit5	TRC カウント動作選択ビット(注 2) cstp_trccr2	0:TRCGRAレジスタとのコンペア一致後もカウント継続 1:TRCGRAレジスタとのコンペア一致でカウント停止 PWM 波形を出力し続けるので"0"を設定	0
bit4,3		"00"を設定	00
bit2	PWM モードアウトプットレベル制御ビット D(注 1) pold_trccr2	0:TRCIOD の出力レベルは"L"アクティブ 1:TRCIOD の出力レベルは"H"アクティブ TRCIOD 端子は使用しません。今回は"0"を設定します。	0
bit1	PWM モードアウトプットレベル制御ビット C(注 1) polc_trccr2	0:TRCIOC の出力レベルは"L"アクティブ 1:TRCIOC の出力レベルは"H"アクティブ TRCIOC 端子の出力レベルを"L"アクティブにします。	0
bit0	PWM モードアウトプットレベル制御ビット B(注 1) polb_trccr2	0:TRCIOB の出力レベルは"L"アクティブ 1:TRCIOB の出力レベルは"H"アクティブ TRCIOB 端子は使用しません。今回は"0"を設定します。	0

注 1. PWM モードのとき有効です。

注 2. アウトプットコンペア機能、PWM モード、PWM2 モードのとき有効です。PWM2 モード時の注意事項は「ハードウェアマニュアル 19.9.6 PWM2 モード時の TRCMR レジスタ」を参照してください。

注 3. PWM2 モードのとき有効です。

タイマ RC 制御レジスタ 2(TRCCR2)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	0	0	0	0
16 進数	0				0			

※初期出力とアクティブレベルについて

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

TRCCR1 bit3~1	TRCCR2 bit2~0	波形
0 初期は アクティブで ないレベル	0 出力レベルは "L"アクティブ	<p>出力 波形</p> <p>1 周期目 2 周期目</p> <p>↑ TRCMR の bit7="1" (カウント開始)</p>
0 初期は アクティブで ないレベル	1 出力レベルは "H"アクティブ	<p>出力 波形</p> <p>1 周期目 2 周期目</p> <p>↑ TRCMR の bit7="1" (カウント開始)</p>
1 初期は アクティブな レベル	0 出力レベルは "L"アクティブ	<p>出力 波形</p> <p>1 周期目 2 周期目</p> <p>↑ TRCMR の bit7="1" (カウント開始)</p>
1 初期は アクティブな レベル	1 出力レベルは "H"アクティブ	<p>出力 波形</p> <p>1 周期目 2 周期目</p> <p>↑ TRCMR の bit7="1" (カウント開始)</p>

今回、TRCIOC 端子は、「TRCCR1 bit2="0"、TRCCR2 bit1="0"」を設定しています。よって、「初期はアクティブでないレベル、出力レベルは"L"アクティブ」となります。

④タイマ RC アウトプットマスタ許可レジスタ(TRCOER:Timer RC output master enable register)の設定

端子から PWM 波形を出力にするかしないか設定します。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7	パルス出力強制遮断信号 入力 $\overline{\text{INT0}}$ 有効ビット pto_trcoer	0:パルス出力強制遮断入力無効 1:パルス出力強制遮断入力有効($\overline{\text{INT0}}$ 端子に“L”を 入力すると、EA、EB、EC、ED ビットが“1”(出力禁 止)になる) 強制遮断入力無効は使用しないの“0”を設定	0
bit6~4		“000”を設定	000
bit3	TRCIOD 出力禁止ビット(注1) ed_trcoer	0:出力許可 1:出力禁止(TRCIOD 端子はプログラマブル入出力ポート) 今回は使用しないので、“1”を設定	1
bit2	TRCIOC 出力禁止ビット(注1) ec_trcoer	0:出力許可 1:出力禁止(TRCIOC 端子はプログラマブル入出力ポート) 今回は使用するのので、“0”を設定	0
bit1	TRCIOB 出力禁止ビット(注1) eb_trcoer	0:出力許可 1:出力禁止(TRCIOB 端子はプログラマブル入出力ポート) 今回は使用しないので、“1”を設定	1
bit0	TRCIOA 出力禁止ビット(注1) ea_trcoer	0:出力許可 1:出力禁止(TRCIOA 端子はプログラマブル入出力ポート) 今回は使用しないので、“1”を設定	1

注 1. 端子をインプットキャプチャ入力として使用するときは無効です。

タイマ RC アウトプットマスタ許可レジスタ(TRCOER)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	1	0	1	1
16 進数	0				b			

⑤タイマ RC 端子選択レジスタ 0(TRCPSR0:Timer RC function select register 0)の設定

TRCIOA 端子、TRCIOB 端子の設定をします。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7		"0"を設定	0
bit6~4	TRCIOB 端子選択ビット bit6:trciobsel2 bit5:trciobsel1 bit4:trciobsel0	000:TRCIOB 端子は使用しない 001:P1_2 に割り当てる 010:P0_3 に割り当てる 011:P0_4 に割り当てる 100:P0_5 に割り当てる 101:P2_0 に割り当てる 110:P6_5 に割り当てる 上記以外:設定しないでください 今回は使用しません。	000
bit3		"0"を設定	0
bit2~0	TRCIOA/TRCTRG 端子選択 ビット bit2:trcioasel2 bit1:trcioasel1 bit0:trcioasel0	000:TRCIOA/TRCTRG 端子は使用しない 001:P1_1 に割り当てる 010:P0_0 に割り当てる 011:P0_1 に割り当てる 100:P0_2 に割り当てる 上記以外:設定しないでください 今回は使用しません。	000

タイマ RC 端子選択レジスタ 0(TRCPSR0)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	0	0	0	0
16 進数	0				0			

⑥タイマ RC 端子選択レジスタ 1(TRCPSR1:Timer RC function select register 1)の設定

TRCIOD 端子、TRCIOD 端子の設定をします。

設定 bit	上:ビット名 下:シンボル	内容	今回の 内容
bit7		"0"を設定	0
bit6~4	TRCIOD 端子選択ビット bit6:trciodsel2 bit5:trciodsel1 bit4:trciodsel0	0 0 0:TRCIOD 端子は使用しない 0 0 1:P1_0 に割り当てる 0 1 0:P3_5 に割り当てる 0 1 1:P0_6 に割り当てる 1 0 0:P2_2 に割り当てる 1 0 1:P6_7 に割り当てる 上記以外:設定しないでください 今回は使用しません。	000
bit3		"0"を設定	0
bit2~0	TRCIOD 端子選択ビット bit2:trciocsel2 bit1:trciocsel1 bit0:trciocsel0	000:TRCIOD 端子は使用しない 001:P1_3 に割り当てる 010:P3_4 に割り当てる 011:P0_7 に割り当てる 100:P2_1 に割り当てる 101:P6_6 に割り当てる 上記以外:設定しないでください 今回は P3_4 端子に割り当てるので"010"を設定します。	010

タイマ RC 端子選択レジスタ 1(TRCPSR1)の設定値を下記に示します。

bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	0	0	1	0
16 進数	0				2			

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

タイマ RC ジェネラルレジスタ A(TRCGRA)に値を設定することによって、出力する PWM 波形の周期を設定します。

PWM 周期は、下記の式で決まります。

$$\text{PWM 周期} = \text{タイマ RC カウンタのカウントソース} \times (\text{TRCGRA} + 1)$$

TRCGRA を左辺に移動して、TRCGRA を求める式に変形します。

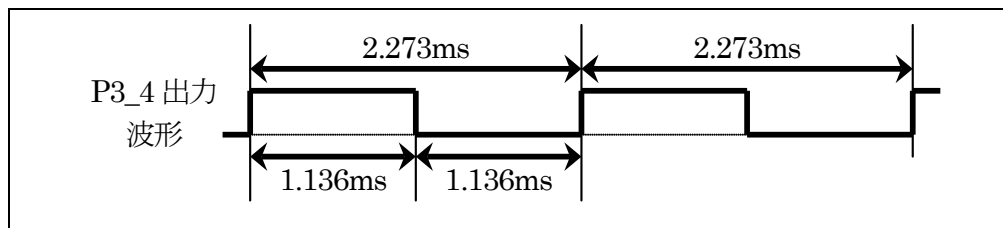
$$\text{TRCGRA} = \text{PWM 周期} / \text{タイマ RC カウンタのカウントソース} - 1$$

タイマ RC カウンタのカウントソースとは、タイマ RC 制御レジスタ 1(TRDCR1)のカウントソース選択ビット(bit6~4)で設定した時間のことで、今回は 200ns に設定しています。

PWM 周期は、ブザーの音階によって変わります。ここでは例として、「4 オクターブ目のラ」の音を鳴らすこととします。「4 オクターブ目のラ」は、440.0Hz の周波数です。時間に直すと下記のようになります。

$$1/440.0 = 2.273\text{ms}$$

波形を下図に示します。



この時間が周期になります。よって、タイマ RC ジェネラルレジスタ A(TRCGRA)は次のようになります。

$$\text{TRCGRA} = \text{周期} / \text{カウントソース} - 1$$

$$\text{TRCGRA} = (2.273 \times 10^{-3}) / (200 \times 10^{-9}) - 1$$

$$\text{TRCGRA} = 11364.7 - 1 \approx 11365 - 1 = 11364$$

init 関数実行時は、PWM 波形を出力しませんので、0 を代入しておきます。

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

タイマ RC ジェネラルレジスタ C(TRCGRC)に値を設定することによって、TRCIOC 端子から出力する PWM 波形の ON 幅を設定します。TRCIOC 端子をどのポートにするかは、タイマ RC 端子選択レジスタ 1(TRCPSR1)で設定し、今回は P3_4 にしています。

ON 幅は、下記の式で決まります。

$$\text{TRCIOC 端子(P3_4)の ON 幅} = \text{タイマ RC カウンタのカウンソース} \times (\text{TRCGRC} + 1)$$

TRCGRC を左辺に移動して、TRCGRC を求める式に変形します。

$$\text{TRCGRC} = \text{TRCIOC 端子(P3_4)の ON 幅} / \text{タイマ RC カウンタのカウンソース} - 1$$

タイマ RC カウンタのカウンソースとは、タイマ RC 制御レジスタ 1(TRDCR1)のカウンソース選択ビット(bit6~4)で設定した時間のことで、今回は 200ns に設定しています。

PWM 周期はブザーの音階によって変わります。ここでは例として、「4 オクターブ目のラ」を鳴らすことにします。「4 オクターブ目のラ」は、440.0Hz の周波数です。時間に直すと下記のようになります。

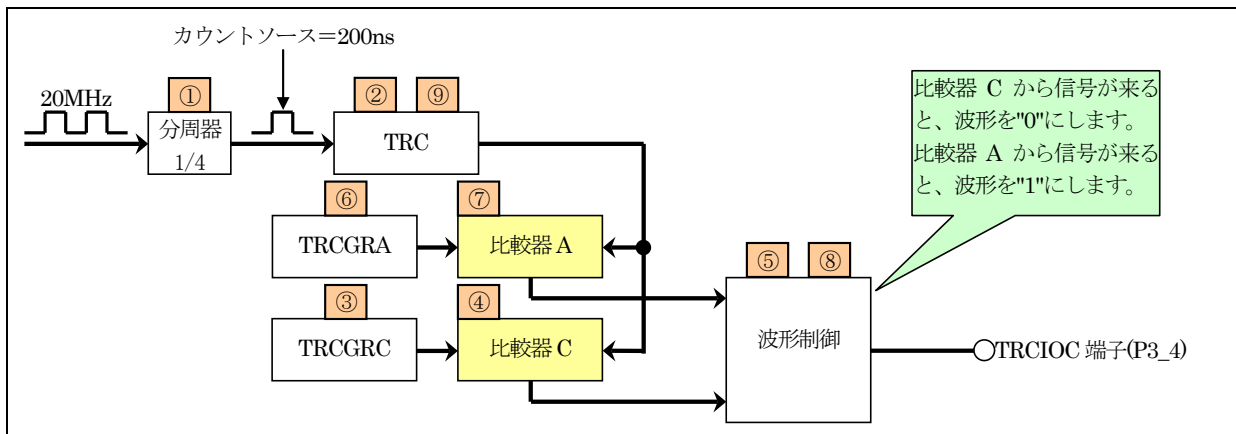
$$1 / 440.0 = 2.273\text{ms}$$

この時間が周期になります。ON 幅はその半分なので 1.136ms となります。よって、タイマ RC ジェネラルレジスタ C(TRCGRC)は次のようになります。

$$\begin{aligned} \text{TRCGRC} &= \text{ON 幅} / \text{カウンソース} - 1 \\ \text{TRCGRC} &= (1.136 \times 10^{-3}) / (200 \times 10^{-9}) - 1 \\ \text{TRCGRC} &= 5682.4 - 1 \approx 5683 - 1 = 5682 \end{aligned}$$

init 関数実行時は、PWM 波形を出力しませんので、0 を代入しておきます。

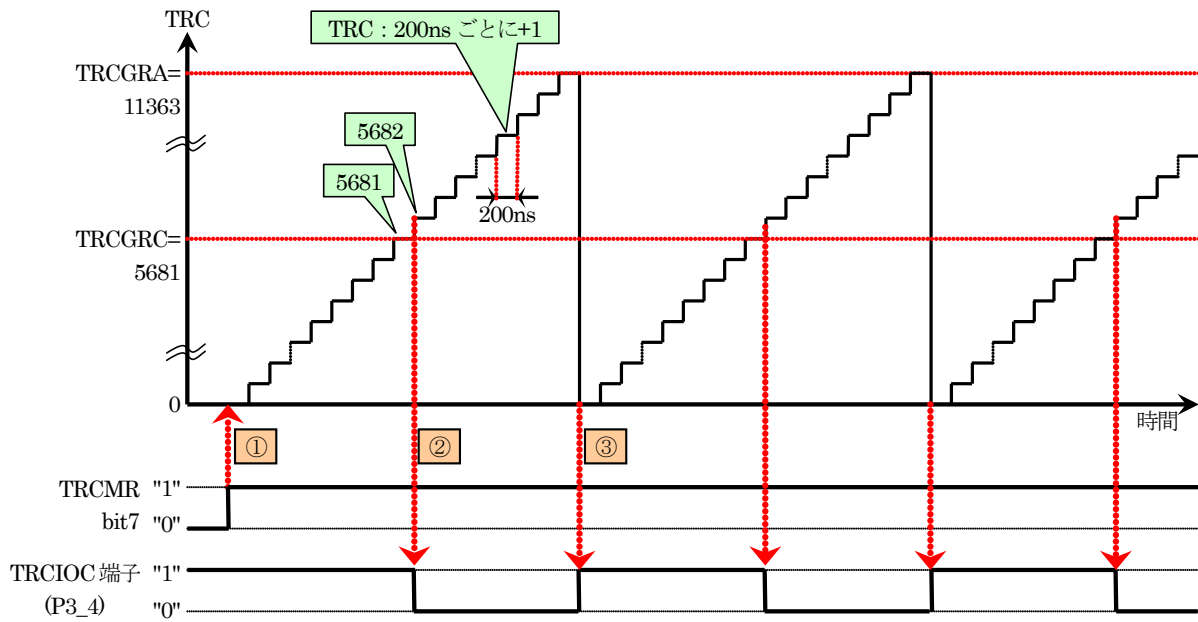
■仕組み



①	分周器には、水晶の 20MHz のパルスが入力されます。分周器は、タイマ RC 制御レジスタ 1(TRDCR1)のカウンソース選択ビットで何分周にするか設定します。今回は f4 を選択しています。これは、4 分周するという事です。よって分周器からの出力は、 20MHz/4=5MHz のパルスが出力されます。時間は、 1/5M=200ns となります。
---	---

②	<p>TRC は、200ns ごとにカウントアップ(+1)します。TRC の最大値は 65,535 で、その次の値は 0 に戻りカウントアップし続けます。</p>
③ ④	<p>TRCGRC には、波形の ON 幅を設定します。 TRCGRC と TRC の値は、常に比較器 C によって比較されています。次の式が成り立つと、波形制御回路へ信号を送ります。</p> $\text{TRCGRC} + 1 = \text{TRC}$ <p>TRC は 200ns ごとに +1 します。TRCGRC には 5681 が設定されているとします。マイコンが起動したとき、TRC は 0 なので</p> $5681 (\text{TRCGRC}) + 1 \neq 0 (\text{TRC})$ <p>となり、成り立ちません。200ns 後、TRC は +1 されて 1 になります。</p> $5681 (\text{TRCGRC}) + 1 \neq 1 (\text{TRC})$ <p>まだ成り立ちません。式が成り立つ TRC が 5682 になるには $5682 \times 200\text{ns} = 1.1364\text{ms}$ です。よって、1.1364ms 経つと</p> $5681 (\text{TRCGRC}) + 1 = 5682 (\text{TRC})$ <p>が成り立ち、比較器 C から信号が波形制御部分へ出力されます。</p>
⑤	<p>PWM 波形制御部分は、比較器 C から信号が送られてくると、波形を“0”にします。</p>
⑥ ⑦	<p>TRCGRA には、波形の周期を設定します。 TRCGRA と TRC の値は、常に比較器 A によって比較されています。次の式が成り立つと、波形制御回路へ信号を送ります。</p> $\text{TRCGRA} + 1 = \text{TRC}$ <p>TRC は 200ns ごとに +1 します。TRCGRA には 11363 が設定されているとします。マイコンが起動したとき、TRD0 は 0 なので</p> $11363 (\text{TRCGRA}) + 1 \neq 0 (\text{TRD0})$ <p>となり、成り立ちません。200ns 後、TRC は +1 されて 1 になります。</p> $11363 (\text{TRCGRA}) + 1 \neq 1 (\text{TRD0})$ <p>まだ成り立ちません。式が成り立つ TRC が 11364 になるには $11364 \times 200\text{ns} = 2.273\text{ms}$ です。よって、2.273ms 経つと</p> $11363 (\text{TRCGRA}) + 1 = 11364 (\text{TRC})$ <p>が成り立ち、比較器 A から信号が波形制御部分へ出力されます。</p>
⑧ ⑨	<p>波形制御部分は、比較器 A から信号が送られてくると、波形を“1”にします。 これと同時に、比較器 A は TRC の値を 0 にクリアします。TRC は、0 からカウントし直します。</p>

TRC の値と波形の関係を図解すると下記のようになります。



①	TRCMR の bit7 を"1"にすると、TRC のカウントが開始されます。
②	TRC=(TRCGRC + 1)になると、P3_4 端子が"0"になります。 TRCGRC の値は 5681、TRC が+1 する間隔は 200ns です。 よって、P3_4 端子が"1"の時間は、 P3_4 端子が"1"の時間 = (TRCGRC + 1) × TRC が+1 する間隔 = (5681 + 1) × 200ns = 1,136,400 [ns] = 約 1.14[ms]
③	TRC=(TRCGRA + 1)になると、P3_4 端子が"1"になります。同時に TRC が 0 にクリアされ、また 0 から値が増えていきます。 TRCGRA の値は 11363、TRC が+1 する間隔は 200ns です。 よって、P3_4 端子の PWM 周期は、 P3_4 端子の PWM 周期 = (TRCGRA + 1) × TRC が+1 する間隔 = (11363 + 1) × 200ns = 2,272,800 = 2.27[ms] ちなみに、 「PWM 周期」=「ON("1")の時間」+「OFF("0")の時間」 ですので、「0」の時間は、 「OFF("0")の時間」=「PWM 周期」-「ON("1")の時間」 = 2.27[ms] - 1.14[ms] = 1.14[ms] となります。

⑨タイマ RC モードレジスタ(TRCMR:Timer RC mode register)の設定

今回のプログラムでは、init 関数内でカウントを開始させていません。もし、init 関数内でカウントを開始(PWM 波形出力開始)する場合の設定を説明します。実際のカウント開始は、beep 関数で行っています。

設定 bit	上:ビット名 下:シンボル	内容	今回の内容
bit7	TRC カウント開始ビット tstart_trcmr	0:カウント停止 1:カウント開始 カウントを開始します。	1
bit6		"0"を設定	変更せず
bit5	TRCGRD レジスタ機能選択ビット bfd_trcmr	0:ジェネラルレジスタ 1:TRCGRB レジスタのバッファレジスタ 値は変更しません。	変更せず
bit4	TRCGRC レジスタ機能選択ビット(注 2) bfc_trcmr	0:ジェネラルレジスタ 1:TRCGRA レジスタのバッファレジスタ 値は変更しません。	変更せず
bit3	PWM2 モード選択ビット pwm2_trcmr	0:PWM2 モード 1:タイマモードまたは PWM モード 値は変更しません。	変更せず
bit2	TRCIOD PWM モード選択ビット(注 1) pwnd_trcmr	0:タイマモード 1:PWM モード 値は変更しません。	変更せず
bit1	TRCIOC PWM モード選択ビット(注 1) pwmc_trcmr	0:タイマモード 1:PWM モード 値は変更しません。	x
bit0	TRCIOB PWM モード選択ビット(注 1) pwmb_trcmr	0:タイマモード 1:PWM モード 値は変更しません。	x

注 1. これらのビットは PWM2 ビットが“1”(タイマモードまたは PWM モード)のとき有効です。

注 2. PWM2 モードでは BFC ビットを“0”(ジェネラルレジスタ)にしてください。

カウントを開始させるため bit7 のみ“1”に、その他のビットは値を変えません。ここでは OR 演算を行って、bit7 のみ“1”にします。

bit	7	6	5	4	3	2	1	0
TRCMR の値	0	0	0	0	1	0	1	0
OR 値	1	0	0	0	0	0	0	0
結果	1	0	0	0	1	0	1	0

強制的に“1”

変化なし

28.5.2 beep 関数

タイマ RC の PWM モードを使って P3_4 端子から PWM 波形を出力し、ブザーを鳴らす関数です。

```

246 : /*****/
247 : /* ブザーを鳴らす */
248 : /* 引数 音階のPWM値 */
249 : /*****/
250 : void beep( unsigned int tone )
251 : {
252 :     if( tone ) {
253 :         trcmr  &= 0x7f;          /* TRCのカウンタ停止 */
254 :         trc    = tone - 1;
255 :         trcgra = tone - 1;      /* 周期設定 */
256 :         trcgrc = tone / 2 - 1; /* ON幅設定 */
257 :         trcmr |= 0x80;          /* TRCのカウンタ開始 */
258 :     } else {
259 :         trcmr  &= 0x7f;          /* TRCのカウンタ停止 */
260 :         trc    = 0;
261 :         trcgra = 0;             /* 周期設定 */
262 :         trcgrc = 0;             /* ON幅設定 */
263 :         trcmr |= 0x80;          /* TRCのカウンタ開始 */
264 :     }
265 : }

```

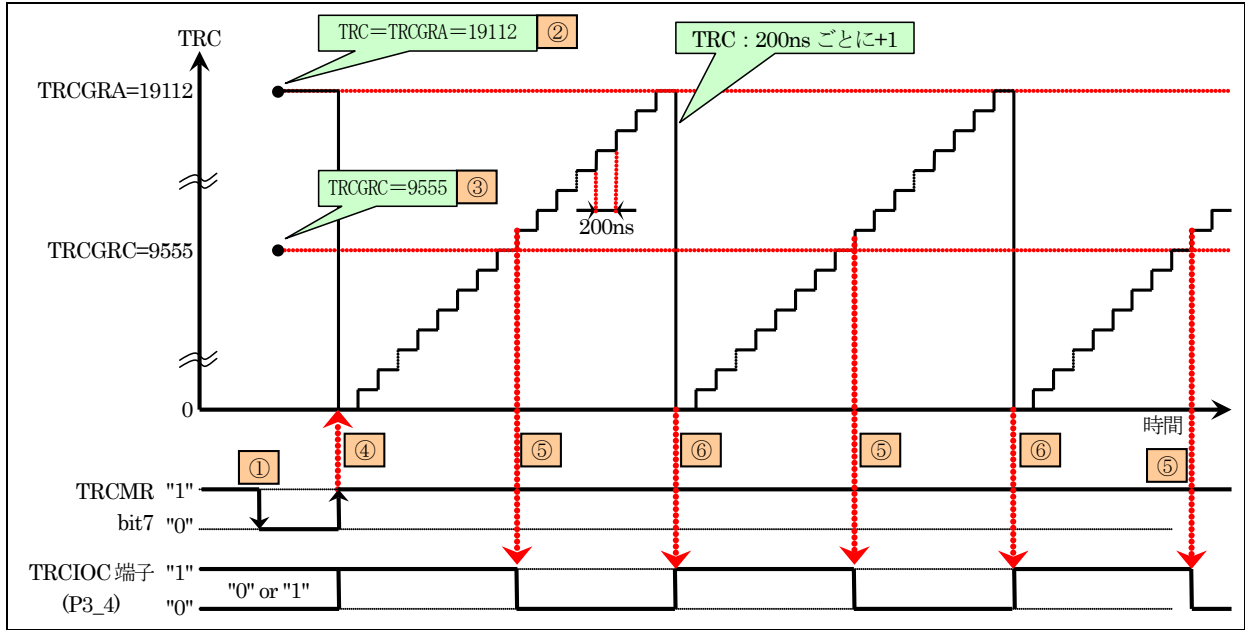
252 行	引数は、タイマ RC ジェネラルレジスタ A(TRCGRA)に設定する周期のデータです。このデータが 0 でないならブザーを鳴らすために 253 行へ、ブザーを止めるなら 259 行へジャンプします。
253 行	タイマ RC モードレジスタ(TRCMR)の bit7 を"0"にして、TRC のカウンタ動作を停止します。PWM 波形出力が止まるので、ブザーが鳴りやみます。
254 行～ 256 行	タイマ RC カウンタ(TRC)にカウンタの初期値、タイマ RC ジェネラルレジスタ A(TRCGRA)に周期、タイマ RC ジェネラルレジスタ C(TRCGRC)に ON 幅を設定します。
257 行	タイマ RC モードレジスタ(TRCMR)の bit7 を"1"にして、TRC のカウンタ動作を開始します。P3_4 端子から PWM 波形が出力され、ブザーが鳴ります。
259 行	タイマ RC モードレジスタ(TRCMR)の bit7 を"0"にして、TRC のカウンタ動作を停止します。PWM 波形出力が止まるので、ブザーが鳴りやみます。ただし、PWM 端子は"0"で止まったか、"1"で止まったかは分かりません。
260 行～ 262 行	タイマ RC カウンタ(TRC)、タイマ RC ジェネラルレジスタ A(TRCGRA)、タイマ RC ジェネラルレジスタ C(TRCGRC)のそれぞれのレジスタに 0 を設定します。
263 行	タイマ RC モードレジスタ(TRCMR)の bit7 を"1"にして、TRC のカウンタ動作を開始します。P3_4 端子は"0"になり、ブザーは鳴りやみます。

■音階の再設定について

例えば、4 オクターブ目のドを鳴らしたい場合、beep 関数の引数は 19113 になり、beep 関数の下記プログラムが実行されます。19113部分が tone 変数の部分です。

```

253 :      tremr  &= 0x7f;          /* TRCのカウンタ停止          */
254 :      trc    = 19113 - 1;
255 :      trcgra  = 19113 - 1;      /* 周期設定                    */
256 :      trcgrc  = 19113 / 2 - 1; /* ON幅設定                    */
257 :      tremr  |= 0x80;          /* TRCのカウンタ開始          */
    
```



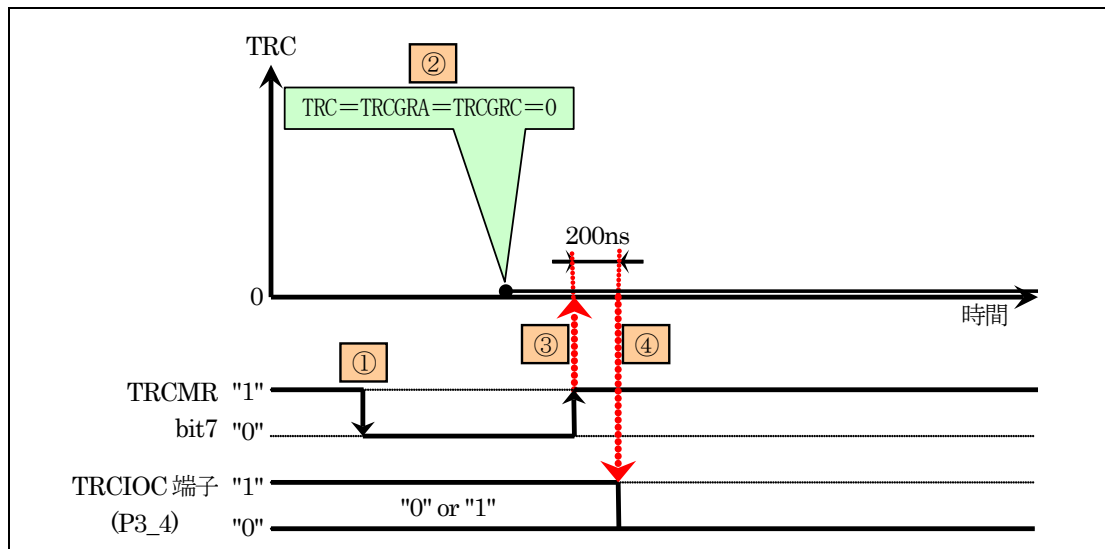
①	253 行	タイマ RC カウンタ(TRC)を停止させます。
②	254 行 255 行	タイマ RC ジェネラルレジスタ A(TRCGRA)に周期を設定します。タイマ RC カウンタ(TRC)に、TRCGRA と同じ値を設定します。
③	256 行	タイマ RC ジェネラルレジスタ C(TRCGRC)に ON 幅を設定します。周期の半分の時間です。
④	257 行	タイマ RC カウンタ(TRC)を動作させます。 動作させてから、200ns 経つと TRCGRA+1=TRC が成り立ち、P3_4 が"1"に、TRC が 0 になります。
⑤		TRCGRC+1=TRC が成り立つと、P3_4 が"0"になります。
⑥		TRCGRA+1=TRC が成り立ち、P3_4 が"1"に、TRC が 0 になります。

■ブザー停止について

ブザーの鳴動を停止するとき、beep 関数の下記プログラムが実行されます。

```

259 :      tcmr  &= 0x7f;          /* TRCのカウンタ停止      */
260 :      trc   = 0;
261 :      trcgra = 1;              /* 周期設定                */
262 :      trcgrc = 1;              /* ON幅設定                 */
263 :      tcmr  |= 0x80;          /* TRCのカウンタ開始      */
    
```



①	259 行	タイマ RC カウンタ(TRC)を停止させます。
②	260～ 262 行	タイマ RC ジェネラルレジスタ A(TRCGRA)、タイマ RC ジェネラルレジスタ C(TRCGRC)、タイマ RC カウンタ(TRC)に 0 を設定します。
③	263 行	タイマ RC カウンタ(TRC)を動作させます。
④		200ns 後、TRC が 0 から 1 になります。 TRCGRA+1=TRC TRCGRC+1=TRC が同時に成り立ちます。TRC は 0 になります。 同時に成り立った場合、波形を"0"にする方が優先されます。 よって、波形は常に"0"になります。

28.5.3 intTRB 関数(タイマ RB 割り込み)

タイマ RB 割り込みで、intTRB 関数を 1ms ごとに実行します。タイマ RB 割り込みについては、プロジェクト:timer2 を参照してください。

```

315 : /*****/
316 : /* タイマRB 割り込み処理 */
317 : /*****/
318 : #pragma interrupt intTRB(vect=24)
319 : void intTRB( void )
320 : {
321 :     cnt_rb++;
322 :
323 :     if( music_flag ) {
324 :         /* ブザー処理 */
325 :         if( cnt_rb >= 15000L * music_data[music_dim][1] / TEMPO ) {
326 :             /* 次の音階をならす */
327 :             cnt_rb = 0;
328 :             music_dim++;
329 :             if( music_data[music_dim][0] == -1 ) {
330 :                 /* -1なら終了 */
331 :                 beep( 0 );
332 :                 music_flag = 0;
333 :             } else {
334 :                 /* -1でないなら次の音階セット */
335 :                 beep( music_data[music_dim][0] );
336 :             }
337 :         }
338 :     }
339 : }

```

321 行	cnt_rb 変数を+1 します。ブザーのタイミング制御用として使用します。
323 行	music_flag をチェックします。0 ならブザー処理をしません。0 以外ならブザー処理を行います。
325 行	<p>次の音階を鳴らすかどうかチェックしています。 比較は、「cnt_rb」と「15000L * music_data[music_dim][1] / TEMPO」を比較しています。 music_data は音データを格納している配列です。music_data 配列に格納されている内容は下記のとおりです。</p> <p>music_data[添字 1][添字 2] 添字 1: 音データの番号 添字 2: 0:音階データ 1:音階を鳴らす長さ</p> <p>添字 1 の部分は、music_dim 変数を使って参照しており、音データの番号を参照しています。音を鳴らすごとに+1 されていき、次の音データを参照します。 添字 2 は、0 なら音階データ、1 なら音階を鳴らす長さを指定します。今回は、どのくらいの時間を鳴らせばよいかチェックしているので「1」を代入して、時間を引き出しています。 音階を鳴らす時間以上の時間が経ったなら、if 文が成り立って次の行に進みます。</p>
327 行	cnt_rb 変数を 0 クリアして、次の音階データを鳴らす時間チェック用にします。
328 行	music_dim 変数を+1 して、次の音階データの参照先にします。

329～ 322 行	音階データが「-1」なら、音階データが終わりだと判断して、beep 関数でブザーへの出力を止め、music_flag を 0 にします。
334 行	音階データが「-1」でないなら、beep 関数で次の音階を鳴らします。

28.5.4 main 関数

main 関数は、初期の設定を行った後、音を鳴らす処理の全てを割り込み内で行っているため、何も行っていません。

```

233 : void main( void )
234 : {
235 :     unsigned char d;
236 :
237 :     init();                /* 初期化                */
238 :     asm( " fset I ");      /* 全体の割り込み許可    */
239 :
240 :     music_flag = 1;        /* 音楽スタート          */
241 :
242 :     while( 1 ) {
243 :     }
244 : }
```

237 行	ポートの入出力設定、タイマ RC による PWM 出力設定、タイマ RB による 1ms ごとの割り込み設定を行います。
238 行	割り込みを許可します。
240 行	music_flag を 1 にします。この行以降の割り込みプログラムから、音を鳴らします。
242～ 243 行	無限ループです。今回は、何もしていません。音を鳴らすのは、割り込みで行っています。

28.6 音階

28.6.1 音階の周波数

音階とは、「ドレミファソラシド」のことです。この音階の周波数が分かれば周期が分かりますので、デューティ比 50% の PWM をブザーに出力すれば、「ドレミファソラシド」と音を鳴らすことができます。

音階は、「ド」から次の高い「ド」まで「ド、ド#、レ、レ#、ミ、ファ、ファ#、ソ、ソ#、ラ、ラ#、シ」の 12 段階あります。12 段階を 1 オクターブといいます。

4 オクターブ目のドの音の周波数は 261.6Hz です。12 段階の周波数は、次の式で求めることができます。

$$\text{周波数} = 261.6 \times 2^{(x/12)} \text{ [Hz]}$$

x は、ドが 0、ド#が 1・・・、シが 11 というように一つずつ増えていきます。

1つ高い 5 オクターブ目のドの周波数は、2 倍の 523.2Hz となります。x に当たる部分は、12 になります。

1つ低い 3 オクターブ目のドの周波数は、1/2 の 130.8Hz となります。x に当たる部分は、-12 になります。

4 オクターブ目の音階の周波数、またそれぞれのオクターブのドの音の計算表を、下記に示します。

オクターブ	音	x	計算	周波数[Hz]	周期[ms]	TRCGRA の値
0	ド	-48	$261.6 \times 2^{(-48/12)}$	16.4	61.16	305810※
1	ド	-36	$261.6 \times 2^{(-36/12)}$	32.7	30.58	152905※
2	ド	-24	$261.6 \times 2^{(-24/12)}$	65.4	15.29	76453※
3	ド	-12	$261.6 \times 2^{(-12/12)}$	130.8	7.65	38226
4	ド	0	$261.6 \times 2^{(0/12)}$	261.6	3.82	19113
	ド#	1	$261.6 \times 2^{(1/12)}$	277.2	3.61	18040
	レ	2	$261.6 \times 2^{(2/12)}$	293.6	3.41	17028
	レ#	3	$261.6 \times 2^{(3/12)}$	311.1	3.21	16072
	ミ	4	$261.6 \times 2^{(4/12)}$	329.6	3.03	15170
	ファ	5	$261.6 \times 2^{(5/12)}$	349.2	2.86	14319
	ファ#	6	$261.6 \times 2^{(6/12)}$	370.0	2.70	13515
	ソ	7	$261.6 \times 2^{(7/12)}$	392.0	2.55	12756
	ソ#	8	$261.6 \times 2^{(8/12)}$	415.3	2.41	12041
	ラ	9	$261.6 \times 2^{(9/12)}$	440.0	2.27	11365
	ラ#	10	$261.6 \times 2^{(10/12)}$	466.1	2.15	10727
シ	11	$261.6 \times 2^{(11/12)}$	493.8	2.02	10125	
5	ド	12	$261.6 \times 2^{(12/12)}$	523.2	1.91	9557
6	ド	24	$261.6 \times 2^{(24/12)}$	1046.4	0.96	4778
7	ド	36	$261.6 \times 2^{(36/12)}$	2092.8	0.48	2389
8	ド	48	$261.6 \times 2^{(48/12)}$	4185.6	0.24	1195

※TRCGRA は、65536 以上の値は設定できません。設定したい場合は、タイマ RC 制御レジスタ 1(TRCCR1)のカウントソース選択ビットを切り替えて、タイマ RC カウンタ(TRC)がカウントアップする時間を変更、65535 以下になるように調整してください。

例として、4 オクターブ目の「ド」を鳴らすときの、タイマ RC ジェネラルレジスタ A(TRCGRA)の計算を下記に示します。

$$\begin{aligned} & \text{ドの周期} / \text{タイマ RC カウンタ(TRC)のカウントアップする間隔} \\ & = (3.82 \times 10^{-3}) / (200 \times 10^{-9}) = 19113 \end{aligned}$$

上表で抜けている音階をエクセルなどで計算して、TRCGRA に代入する値の一覧を作ってください。

28.6.2 音階名の定義

プログラムでは、下記のルールに従って、音階名を定義します。

音階 + **□** + **オクターブ**

音階名は、下記のように定義します。

音階	プログラム中の標記
ド	DO
ド#	DOU
レ	RE
レ#	REU
ミ	MI
ファ	FA
ファ#	FAU
ソ	SO
ソ#	SOU
ラ	RA
ラ#	RAU
シ	SI







プログラムでは、#define で音階名と TRCGRA の値を定義します。
4 オクターブ目の定義を下記に示します。

```
38 : /* 4 オクターブ目の音階 */
      名称      TRCGRA の設定値
39 : #define    DO_4      19113    /* ド          */
40 : #define    DOU_4     18040    /* ド#         */
41 : #define    RE_4      17028    /* レ          */
42 : #define    REU_4     16072    /* レ#         */
43 : #define    MI_4      15170    /* ミ          */
44 : #define    FA_4      14319    /* ファ        */
45 : #define    FAU_4     13515    /* ファ#       */
46 : #define    SO_4      12756    /* ソ          */
47 : #define    SOU_4     12041    /* ソ#         */
48 : #define    RA_4      11365    /* ラ          */
49 : #define    RAU_4     10727    /* ラ#         */
50 : #define    SI_4      10125    /* シ          */
```







28.6.3 音階の時間の長さ

楽譜に「♩=100」というような記号があります。これは 1 分間あたりの拍数を表しており、1 分間に 4 分音符(♩)を 100 回演奏する早さです。60 であれば、4 分音符 1 個で 1 秒になります。プログラムでは「TEMPO」として定義しています。

音符の種類、拍数を下表に示します。

音符	音符名	説明	拍	四分音符を4としたときの長さ	テンポが60のときの時間[秒]
	全音符	4 分音符の 4 倍に当たる長さの音を表現する音符。	4	16	4
	2 分音符	4 分音符の 2 倍に当たる長さの音を表現する音符。	2	8	2
	4 分音符	基準となる長さの音。便宜上、この音の長さを 1 拍とする。	1	4 (1あたり0.25秒)	1
	8 分音符	4 分音符の 1/2 に当たる長さの音を表現する音符。	1/2	2	0.5
	16 分音符	4 分音符の 1/4 に当たる長さの音を表現する音符。	1/4	1	0.25
	32 分音符	4 分音符の 1/8 に当たる長さの音を表現する音符。	1/8	なし	

休符の種類、拍数を下表に示します。休符は、何も演奏せずに休むことです。

休符	休符名	説明	拍	四分休符を4としたときの値	テンポが60のときの時間[秒]
	全休符	4 分休符の 4 倍に当たる長さの休み(無音)を表現する休符。	4	16	4
	2 分休符	4 分休符の 2 倍に当たる長さの休み(無音)を表現する休符。	2	8	2
	4 分休符	4 分音符と同じ長さに当たる休み(無音)を表現する休符。	1	4 (1あたり0.25秒)	1
	8 分休符	4 分音符の 1/2 に当たる長さの休み(無音)を表現する休符。	1/2	2	0.5
	16 分休符	4 分音符の 1/4 に当たる長さの休み(無音)を表現する休符。	1/4	1	0.25
	32 分休符	4 分音符の 1/8 に当たる長さの休み(無音)を表現する休符。	1/8	なし	

プログラムでは、4 分音符(休符)の長さを「4」とします。テンポが 60 のとき、4 で 1 秒の時間になります。このよって、時間の計算は下記のようになります。なお、長さは ms 単位とします。

$$\begin{aligned}
 \text{時間} &= (60/\text{テンポ}) \times (\text{長さ}/4) \times 1000\text{ms} \\
 &= 15/\text{テンポ} \times \text{長さ} \times 1000 \\
 &= 15000 \times \text{長さ}/\text{テンポ} \quad [\text{ms}]
 \end{aligned}$$

計算の順番は、割り算を最後にしたほうが誤差が少なくなるため割り算を最後にしています。

プログラムでは、テンポを設定するシンボルがあります。

```
66 : #define      TEMPO      60      /* テンポ      */
```

時間の計測は、cnt_rb 変数で行っています。よって、時間のチェックは下記のようになります。

```
325 :      if( cnt_rb >= 15000L * 長さ / TEMPO ) {
```

「15000L」の「L」は long 型にする意味です。「L」を付けないと、15000 は int 型になります。「15000×長さ」の結果が 32768 以上になると int 型の範囲を超えて値が不定になるので、強制的に long 型にして計算します。

28.6.4 音データ

音データは、配列を使ってプログラムします。今回は、「music_data」という配列に音データを入れます。書式を下記に示します。

```
const int music_data[][2] = {
    0      , 0      ,      スタートは必ず 0, 0 から始める
    音階名 , 長さ ,
    DO_4   , 4      ,      4 オクターブ目のドを 4 の長さ演奏(4 分音符)
    FAU_5   , 16     ,      5 オクターブ目のファ # を 16 の長さ演奏(全音符)
    0      , 8      ,      休符は 0 にして、長さを入れる
    -1     , 0      ,      終了は必ず音階名部分を -1 にする
};
```

小数点は指定できません。16 分音符より短い時間の音を鳴らしたい場合は、テンポを 2 倍、長さも 2 倍にするなどして、小数点にならないようにしてください。

29. 付録

29.1 R8C マイコンの変数のサイズ

C言語は”手続き型言語”と呼ばれ、変数(データの入れ物)は初めに用意し、しかも大きさも決めておかななくてはなりません。

```
void main( void )
{
    int a,b,c ;
    a = 100 ; b = 2000 ;
    c = a * b ;
    printf("Yn c = %d ",c) ;
}
```

画面にはどのように表示されるでしょうか。200000 と正確に表示されるものもあれば 3392 と、とんでもない表示をするものもあります。実はint型の大きさは”処理系に依存する”という言葉で表現されており何ビットであるかは決まっていないのです。Cコンパイラを作成した開発者に任されている部分なのです。

R8C マイコンのコンパイラで結果をだすと 3392 になります($65,536 \times 3 + 3,392 = 200,000$)。

コンパイラは、桁あふれしたかどうかの確認を取るようなことはしません。データの取り扱いはプログラマに任されています。

R8C のコンパイラは以下のようになっています。

●整数型(R8C マイコンのコンパイラの場合)

型	値の範囲	データサイズ
char (unsigned char 型として扱われる) ※H8 は、signed char 型として扱われます。 R8C と H8 では異なります。	0 ~ 255	1 バイト
signed char	-128 ~ 127	1 バイト
unsigned char	0 ~ 255	1 バイト
short	-32768 ~ 32767	2 バイト
unsigned short	0 ~ 65535	2 バイト
int	-32768 ~ 32767	2 バイト
unsigned int	0 ~ 65535	2 バイト
long	-2147483648 ~ 2147483647	4 バイト
unsigned long	0 ~ 4294967295	4 バイト

●実数型(R8C マイコンのコンパイラの場合)

型	データサイズ	限界値	
		最大値	正の最小値
float	4 バイト	3. 4028235677973364e+38f (0x7F7FFFFFFF)	7. 0064923216240862e-46f (0x00000001)
double long double	8 バイト	1. 7976931348623158e+308 (0x7FEFFFFFFFFFFFFFFF)	4. 9406564584124655e-324 (0x0000000000000001)

29.2 演算子

演算子を用いると、値をいろいろと加工することができます。演算というのはちょっと硬い表現ですが、簡単に言えば、足す、引く、掛ける、割るなどの計算です。

演算子の機能を下記に示します。

演算子	機能	備考	例	
単項演算子	-	負	-a	
	+	正	+b	
	~	ビットごとの反転	チルダと読む ~a	
	--	デクリメント	a--	
	++	インクリメント	b++	
	&	変数のアドレス	&a はaの変数が格納されている アドレス	&a
	*	ポインタ変数の指す内容	*p はpの指す内容	*p
2項演算子	-	減算	a = b - c;	
	+	加算	a = b + c;	
	*	積	a = b * c;	
	/	商	a = b / c;	
	%	整数除算の余り	a = b % c;	
	&	ビットごとの論理積	a = b & 0x7f;	
		ビットごとの論理和	a = b 0x80;	
	^	ビット毎の排他的論理和	a = b ^ 0x55;	
	&&	論理積	答えは真か偽	if(a==b && c==d)
		論理和	答えは真か偽	if(a==b c==d)
	>>	右シフト	(変数名) >> (シフトするビット数)	a = a >> 2;
	<<	左シフト	(変数名) << (シフトするビット数)	a = a << 2;
	代入演算子	=	代入	a = b;
+=		加算して代入	a += b;	
-=		減算して代入	a -= b;	
/=		除算して代入	a /= b;	
%=		剰余演算して代入	a %= b;	
<<=		左シフト演算して代入	a <<= 5;	
>>=		右シフト演算して代入	a >>= 2;	
&=		論理積して代入	a &= 0x55;	
=		論理和して代入	a = b;	
^=		排他的論理和して代入	a ^= b;	
比較演算子	==	等しい	if(a == b)	
	!=	等しくない	if(a != b)	
	>	より大きい	if(a > b)	
	<	より小さい	if(a < b)	
	>=	より大きいか等しい	if(a >= b)	
	<=	より小さいか等しい	if(a <= b)	

29.3 優先順位

式は優先順位の高い順に評価され、同順位なら結合規則にしたがって、左→右または右→左に評価されます。優先順位を変えるには()を用います。

優先順位 演算子	1 高	2	3	4	5	6	7	8	9	10	11	12	13	14	15 低
関数、カッコ	()														
配列	[]														
構造体	・ ->														
型		(型) sizeof													
ポインタ		* &													
インクリメント /デクリメント		++ --													
算術		+ -	* / %	+ -	※優先順位 2 は、単項演算子 例) -a 優先順位 4 は、二項演算子 例) a-b										
関係						< <= > >=	== !=								
ビット		~			<< >>			&	^						
論理		!									&&				
条件													?:		
代入														= += *= など	
コンマ															,
結合規則	左→右	左→右	左 → 右										左←右	左→右	
演算子 優先順位	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

29.4 型が混合したときの演算

違う型が混合した計算の場合、下記のような決まりで演算されます。

- char と unsigned char と short は int
 - unsigned short は unsigned int
 - float は double
- } に変換され、

long double > double > unsigned long > long > unsigned int > int

の優先度で型の高い方に変換されて演算されます。

(ただし、char < short = int とする)

29.5 printf 関数の使い方

printf 関数の呼び出しは次の形式で記述します。

```
ret = printf( fmt , arg1 , arg2 , ... ) ;
```

ただし ret :int 型。出力した文字数(エラー時は -1)。
 fmt :char 型へのポインタ型。フォーマット変換を指定する文字列。
 arg1, arg2, ... :定数または表示データの格納された変数や式(書式に依存)。

(A) printf 関数は、fmt で示される文字列をそのまま表示します。

ただし

(B) 文字列の中に「%」があると、それに続く文字により、2番目以降(arg1)の引数の示す値を変換し、変換結果を文字列に埋め込んで表示します。

(C) 「%」の後に続くものはオプションと変換指定文字です。

%[オプション]変換指定文字 []は省略可

(1) 変換指定文字

変換指定文字	引数の型	表示のされ方
d	int	10 進数
o	int	8進数
x	int	16 進数
u	int	符号なし 10 進数
e	double	[-]m.nnnnnne[±]xx の形式の 10 進浮動小数点数 (n の桁数はオプションで可変だが標準では6桁)
f	double	[-]m.nnnnnn の形式の 10 進浮動小数点数 (n の桁数はオプションで可変だが標準では6桁)
c	int	単一文字
s	char *	文字列
p	void *	ポインタ
%		%

※printf 関数は多くのメモリを消費するため R8C マイコンでは、「%e, %E, %f, %g, %G」の変換指定文字は使用できません。

(2) オプション

l : 引数を long 型のサイズとして扱う。なお、l は整数型に適用可能。

数値 : 出力幅の指定。変換結果の文字数が指定値より少ないときは右詰めとなる。
 また、「数値」の左に「-」を付けると出力欄に左詰めされる。

「.」を含む数字列 : 浮動小数点形式に変換する場合の出力欄の幅と精度(小数点以下の桁数)を「.」で区切って示す。指定がなければ小数点以下 6 桁表示となる。

29. 付録

プログラム

```

2: #include <stdio.h>
3:
4: void main( void )
5: {
6:     int    a = 64 , b = -64;
7:     double c = 6.4;
8:     char   data[20] = { "I Love You !" };
9:
10:    printf("decimal    :%20d%20d\n", a, b);
11:    printf("unsigned   :%20u%20u\n", a, b);
12:    printf("octal      :%20o%20o\n", a, b);
13:    printf("hexadecimal :%20x%20x\n", a, b);
14:    printf("character  :%20c\n", a);
15:    printf("double     :%20f\n", c);
16:    printf("double - e  :%20e\n", c);
17:    printf("string     :%20s\n", data);
18: }

```

実行結果

```

decimal    :                64                -64
unsigned   :                64                65472
octal      :                100               177700
hexadecimal :                40                ffc0
character  :                @
double     :                6.400000
double - e :                6.400000e+00
string     :                I Love You !

```

解説

2行	printf 関数を使用するため「stdio.h」をインクルードします。
6～8行	必要となる変数を初期値付きで宣言します。
10行	変換指定 %d により 64 と -64 が表示されます。
11行	変換指定 %u により -64 を符号なし 2 進数とみなして 10 進数に変換し、65472 が表示されます。
12行	変換指定 %o により 64 と -64 の 8 進表現が表示されます。
13行	変換指定 %x により 64 と -64 の 16 進表現が表示されます。
14行	変換指定 %c により 64 を文字に変換し @ が表示されます。
15行	変換指定 %f により 6.4 が小数点のみの形式で表示されます。
16行	変換指定 %e により 6.4 が指数形式で表示されます。
17行	変換指定 %s により char 型配列の内容が文字列として表示されます。

29.6 scanf 関数の使い方

scanf 関数の呼び出しは次の形式で記述します。

```
ret = scanf( fmt , arg1 , arg2 , ... ) ;
```

ただし	ret	:int 型。読み込んで変換されたデータの数(エラー時は -1)。
	fmt	:char 型へのポインタ型。フォーマット変換を指定する文字列。
	arg1, arg2, ...	:変換したデータの格納先を示すアドレスや式(書式に依存)。

- (A) キーボードから改行キーが入力されるまで文字をバッファに入力する。改行が入力されてはじめて変換作業が始まり、バッファから読み込まれて処理される。バッファの管理はライブラリ関数側で自動的に行われる。
- (B) fmt が示す文字列中の「%」に続く「オプション」(省略可)と「変換指定文字」に従って変換を行い、結果を2番目以降(arg1)の引数が示す格納先に格納する。
- (C) fmt が示す文字列は、原則として「%」と「変換指定文字」の列で構成する。ただし、例外的に「 % 」の付かない文字を挿入することがある。(F)参照
- (D) 変換指定文字が「c」以外の場合、改行やスペースなどの非印字文字は区切り記号とみなされる。また、変換データよりも前にある場合は読み飛ばされ、後の場合はバッファ内に読み残される。したがって、文字列の読み込みの際にスペースも含めて入力したい場合、変換指定文字の「s」は使えない。
変換結果が正常でない場合、例えば 10 進入力にもかかわらず数字以外のものが入力されていた場合は、その文字を読み込まずに作業は終了する。
- (E) 変換指定文字が「c」の場合は、ASCII コードはすべて(改行やスペースも含め)処理の対象となる。そのため、それ以前の scanf関数の処理によってバッファに読み残された非印字文字があれば、それを読んでもしまうことになる。これを避けるため `" %c"` とスペースを挿入して記述する方法が用いられる。
- (F) オプション「l」は long 型および double 型のデータを入力する場合に使用する。

(1) 変換指定文字

変換指定文字	引数の型	入力データ
d	int *	10 進数
o	int *	8 進数
x	int *	16 進数
f	float *	浮動小数点数
e	float *	浮動小数点数
c	char *	単一文字
s	char *	文字列(最後にヌルコードが付加される)
p	void *	ポインタ
[文字]	char *	入力データの中から[]内に出てくる文字のみを入力する。[]にない最初の文字で入力終了し、この文字は入力されない
[^文字]	char *	入力データの中から[]内の文字のみを読み飛ばす

また、「%」と変換指定文字の間に「*」(代入抑止文字)を付けると、変換指定文字を「読み捨てる」という意味になります。

30. 参考文献

- ・ルネサス エレクトロニクス(株)
R8C/35A グループ ハードウェアマニュアル Rev.0.40
- ・ルネサス エレクトロニクス(株)
High-performance Embedded Workshop V.4.00 ユーザーズマニュアル Rev.3.00
- ・ルネサス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 エレクトロニクスのホームページをご覧ください。

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

の「製品情報」欄→「マイコン」→「R8C」でご覧頂けます