マイコン
実習マニュアル
(R8C/38A 版)

2013 年度から、RY_R8C38 ボードに搭載されているマイコンが R8C/38A から R8C/38C に変更されました。R8C/38A マイコンと R8C/38C マイコンは、機能的にほぼ互換で、マイコンカーで使う範囲においてはプログラムの変更はほとんどありません。
本マニュアルでは『R8C/38A』のマイコン名称が残っていますが、すべて『R8C/38A および R8C/38C』の意味になります。

第 1.25 版
2015.09.08
ジャパンマイコンカーラリー実行委員会
株式会社日立ドキュメントソリューションズ

注 意 事 項 (rev.6.0J)

著作権

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

禁止事項

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

転載、複製

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

責任の制限

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

その他

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

連絡先

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

目　次

1. R8C/38C マイコンの仕様 1
1.1 シリーズ展開 1
1.2 R8C/38C の特徴 2
1.3 ブロック図 3
1.4 ピン配置 4
1.5 メモリマップ 5
1.6 割り込みベクタテーブル領域 6
1.7 固定割り込みベクタテーブル領域 7
1.8 可変ベクタテーブル領域 8
1.9 CPU レジスタ 9
1.10 タイマの概要 11
1.11 タイマ RA～RG の使用用途 12
2. RY_R8C38 ボードの仕様 13
2.1 RY_R8C38 ボードの仕様 13
2.2 外観 14
2.3 寸法 16
2.4 ポート表 17
2.5 コネクタ信号表 18
2.6 回路図 19
3. 書き込み基板の仕様 20
3.1 書き込み方法 20
3.2 USB-TTL 変換基板(RY-WRITER 基板) 22
3.2.1 外観 22
3.2.2 接続図 23
3.2.3 電源の流れ 23
3.2.4 回路図 24
3.3 RS232C-TTL レベル変換基板 25
3.3.1 外観 25
3.3.2 接続図 26
3.3.3 電源の流れ 26
3.3.4 回路図 27
4. RY_R8C38 ボードコネクタ変換基板 28
4.1 外観 28
4.2 回路図 28
4.3 ポートの変換表 29
4.4 RY_R8C38 ボードへ実装 29
5. 実習基板 Ver.2 30
5.1 仕様 30
5.2 外観 31
5.3 スライドスイッチ部 32
5.3.1 コネクタ 32
5.3.2 回路図 33

5.3.3 動作原理 33
5.4 LED 部 34
5.4.1 コネクタ 34
5.4.2 回路図 35
5.4.3 動作原理 35
5.5 7 セグメント LED 部 36
5.5.1 コネクタ 36
5.5.2 回路図 37
5.5.3 動作原理 39
5.6 ブザー部 40
5.6.1 コネクタ 40
5.6.2 回路図 41
5.6.3 動作原理 41
5.7 トグルスイッチ部 42
5.7.1 コネクタ 42
5.7.2 回路図 43
5.7.3 動作原理 43
5.8 ボリューム部 46
5.8.1 コネクタ 46
5.8.2 回路図 47
5.8.3 動作原理 47
5.9 SW10～SW13 を切り替えるときの注意点 48

6. 開発環境のダウンロード、インストール 49

6.1 MY Renesas に登録 49
6.2 ルネサス統合開発環境のダウンロード 50
6.3 ルネサス統合開発環境のインストール 52
6.4 ショートカットの作成 54
6.5 R8C Writer のダウンロード 55
6.6 R8C Writer のインストール 56
6.7 R8C Writer をルネサス統合開発環境に登録する 57
6.8 ドライバーのインストール 60
6.9 拡張子の表示 63
6.9.1 WindowsXP の場合 63
6.9.2 Windows Vista、Windows 7 の場合 64

7. サンプルプログラムのダウンロード、インストール 66

7.1 サンプルプログラムのダウンロード 66
7.2 サンプルプログラムのインストール 67

8. 演習手順 68

8.1 ワークスペースを開く 68
8.2 プロジェクトを開く 69
8.3 ファイル編集 71
8.4 ビルド（MOT ファイルの作成） 74
8.5 プログラムの書き込み 77

9. プロジェクトの内容 81

10. 電源が入ってからのマイコンの動作 83

10.1 概要 83
10.2 ファイルの構成 83
10.3 動作手順 84
10.3.1 リセットの種類 84
10.3.2 電源投入してからプログラム実行までの動作 85
10.4 プログラム「startup.c」 86
10.5 セクション 88
10.5.1 概要 88
10.5.2 セクションの種類 89
10.5.3 プログラムをセクションに分類 91
10.5.4 アドレスに配置 93
10.5.5 セクションの設定 94
10.6 プログラムの解説「startup.c」 95
10.6.1 スタックサイズの設定 95
10.6.2 CPU レジスタの宣言 96
10.6.3 SB の値をコンパイラに設定 97
10.6.4 オプション機能選択レジスタの設定 97
10.6.5 ID コードの設定 99
10.6.6 RAM の初期化 102
10.6.7 セクションの先頭アドレスの型定義 104
10.6.8 ヒープ領域の設定 105
10.6.9 固定割り込みベクタアドレスの設定 106
10.6.10 可変割り込みベクタの設定 111
10.6.11 スタートアッププログラム（start 関数） 111
10.6.12 SB 相対アドレッシング 113
10.6.13 switch_table セクション 114

11. I/O ポートの入出力（プロジェクト：io） 117

11.1 概要 117
11.2 接続 117
11.3 プロジェクトの構成 118
11.4 プログラム「io.c」 118
11.5 プログラムの解説 119
11.5.1 ヘッダファイルの取り込み 119
11.5.2 init 関数(クロックの選択) 120
11.5.3 init 関数(I/O ポートの入出力設定) 129
11.5.4 init 関数(端子のプルアップ) 138
11.5.5 main 関数 140
11.6 ビット操作 141
11.6.1 特定のビットを"0"にする（マスク処理） 141
11.6.2 特定のビットを"1"にする 143
11.6.3 特定のビットを反転する 144
11.6.4 全ビット反転する（NOT 演算） 144
11.6.5 ビットシフトする 145
11.7 演習 146

12. I/O ポートの入出力 2（プロジェクト：io2） 147

12.1 概要 147
12.2 接続 147
12.3 プロジェクトの構成 148

12.4 プログラム「io2.c」.....148
12.5 プログラムの解説.....150
12.5.1 dipsw_get 関数.....150
12.5.2 led_out_m 関数.....151
12.5.3 main 関数.....152
12.6 ビット単位でデータを入力、出力する.....153
12.7 演習.....153

13. ソフトウェアによるタイマ（プロジェクト：timer1）.....154

13.1 概要.....154
13.2 接続.....154
13.3 プロジェクトの構成.....155
13.4 プログラム「timer1.c」.....155
13.5 プログラムの解説.....157
13.5.1 timer 関数（時間稼ぎ）.....157
13.5.2 main 関数.....158
13.6 演習.....159

14. 割り込みによるタイマ（プロジェクト：timer2）.....160

14.1 概要.....160
14.2 接続.....160
14.3 プロジェクトの構成.....161
14.4 プログラム「timer2.c」.....161
14.5 プログラムの解説.....163
14.5.1 割り込みとは.....163
14.5.2 init 関数(タイマ RB の設定).....166
14.5.3 intTRB 関数（1ms ごとに実行される関数）.....175
14.5.4 timer 関数（割り込みを使った時間稼ぎ）.....177
14.5.5 main 関数.....179
14.5.6 割り込みの発生タイミング.....180
14.6 演習.....182

15. 7 セグメント LED の表示（プロジェクト：seven_seg）.....183

15.1 概要.....183
15.2 接続.....183
15.3 プロジェクトの構成.....184
15.4 プログラム「seven_seg.c」.....184
15.5 プログラムの解説.....186
15.5.1 変数領域.....186
15.5.2 intTRB 関数(1ms ごとの割り込み).....187
15.5.3 main 関数.....189
15.6 演習.....189

16. 外部割り込み（プロジェクト：int_interrupt）.....190

16.1 概要.....190
16.2 接続.....190
16.3 プロジェクトの構成.....191
16.4 プログラム「int_interrupt.c」.....191
16.5 プログラムの解説.....193
16.5.1 init 関数($\overline{\text{INT3}}$割り込みの設定).....193

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) 201
16.5.3 main 関数.....202
16.6 演習.....202

17. A/D コンバータ(単発モード)(プロジェクト:ad) 203

17.1 概要.....203
17.2 接続.....203
17.3 プロジェクトの構成 204
17.4 プログラム「ad.c」.....204
17.5 プログラムの解説 205
17.5.1 init 関数(ポートの入出力設定) 205
17.5.2 get_ad7 関数(A/D コンバータの設定、A/D 値取得).....206
17.5.3 main 関数.....215
17.6 演習.....215

18. A/D コンバータ(繰り返しモード 0)(プロジェクト:ad_kurikaeshi) 216

18.1 概要.....216
18.2 接続.....216
18.3 プロジェクトの構成 216
18.4 プログラム「ad_kurikaeshi.c」.....217
18.5 プログラムの解説 218
18.5.1 init 関数(I/O ポートの入出力設定) 218
18.5.2 init 関数(A/D コンバータの設定).....219
18.5.3 get_ad7 関数.....224
18.5.4 main 関数.....225
18.6 演習.....225

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) 226

19.1 概要.....226
19.2 接続.....226
19.3 プロジェクトの構成 228
19.4 プログラム「ad_kurikaeshi_souin.c」.....228
19.5 プログラムの解説 230
19.5.1 init 関数(I/O ポートの入出力設定) 230
19.5.2 init 関数(A/D コンバータの設定).....231
19.5.3 main 関数.....236
19.6 演習.....237

20. パルスカウント(プロジェクト:timer_rg_counter) 238

20.1 概要.....238
20.2 接続.....238
20.3 プロジェクトの構成 239
20.4 プログラム「timer_rg_counter.c」.....240
20.5 ロータリエンコーダ(1相).....241
20.5.1 概要 241
20.5.2 原理 241
20.5.3 市販されているロータリエンコーダ(1相出力).....242
20.5.4 ロータリエンコーダの自作 242
20.6 プログラムの解説 243
20.6.1 init 関数(ポートの入出力設定) 243

20.6.2 init 関数(タイマ RG の設定) 243
20.6.3 main 関数 249
20.7 演習 249

21. 2 相エンコーダのパルスカウント（プロジェクト：timer_rg_isoukeisu） 250

21.1 概要 250
21.2 接続 250
21.3 プロジェクトの構成 251
21.4 プログラム「timer_rg_isoukeisu.c」 252
21.5 ロータリエンコーダ（2相） 253
21.5.1 概要 253
21.5.2 2 相出力のロータリエンコーダ 253
21.5.3 市販されているロータリエンコーダ(2 相出力) 254
21.6 プログラムの解説 254
21.6.1 init 関数（ポートの入出力設定） 254
21.6.2 init 関数(タイマ RG の設定) 255
21.6.3 main 関数 259

22. 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） 260

22.1 概要 260
22.2 接続 260
22.3 プロジェクトの構成 261
22.4 プログラム「timer_rd_doukipwm.c」 261
22.5 PWM 263
22.5.1 PWM とは？ 263
22.5.2 PWM 波形を電圧換算する 264
22.6 プログラムの解説 265
22.6.1 init 関数(タイマ RD の設定) 265
22.6.2 main 関数 287
22.7 演習 289

23. 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） 290

23.1 概要 290
23.2 接続 290
23.3 プロジェクトの構成 291
23.4 プログラム「timer_rd_pwm6.c」 291
23.5 プログラムの解説 293
23.5.1 init 関数(タイマ RD の設定) 293
23.5.2 main 関数 311
23.6 演習 315

24. モータの制御（プロジェクト：motor） 316

24.1 概要 316
24.2 接続 316
24.3 プロジェクトの構成 317
24.4 プログラム「motor.c」 317
24.5 モータの制御 320
24.5.1 モータドライブ基板の役割 320
24.5.2 正転、逆転、ブレーキ、フリーの原理 320
24.5.3 H ブリッジ回路 321

24.5.4 H ブリッジ回路のスイッチをFETにする.....321
24.5.5 P チャネル FET と N チャネル FET の短絡防止回路.....324
24.5.6 フリー回路.....327
24.5.7 実際の回路.....328
24.5.8 左モータの動作.....330
24.5.9 右モータの動作.....330
24.5.10 RY_R8C38 ボードとモータドライブ基板 Ver.5 の接続.....331
24.6 プログラムの解説.....332
24.6.1 init 関数(タイマ RD の設定).....332
24.6.2 motor 関数.....344
24.6.3 main 関数.....350

25. サーボの制御（プロジェクト：servo）.....351

25.1 概要.....351
25.2 接続.....351
25.3 プロジェクトの構成.....352
25.4 プログラム「servo.c」.....352
25.5 サーボ.....354
25.5.1 サーボの接続.....354
25.5.2 RY_R8C38 ボードとモータドライブ基板 Ver.5 の接続.....355
25.6 プログラムの解説.....356
25.6.1 init 関数(タイマ RD の設定).....356
25.6.2 handle 関数.....358
25.6.3 main 関数.....360

26. 通信（プロジェクト：uart0）.....361

26.1 概要.....361
26.2 接続.....361
26.3 プロジェクトの構成.....362
26.4 プログラム「uart0.c」.....362
26.5 パソコンと RY_R8C38 ボードの通信.....365
26.5.1 RY_R8C38 ボードの通信方法.....365
26.5.2 COM ポートの確認.....365
26.5.3 シリアル通信の設定.....366
26.6 プログラムの解説.....368
26.6.1 init 関数(UART0 の設定).....368
26.6.2 get_uart0 関数.....375
26.6.3 put_uart0 関数.....378
26.6.4 main 関数.....380
26.7 実習手順.....381
26.7.1 Tera Term のインストール.....381
26.7.2 Tera Term を使って、マイコンと通信しよう.....386
26.7.3 Tera Term に表示される文字について.....388
26.8 演習.....389

27. printf、scanf 文を使った通信（プロジェクト：uart0_printf）.....390

27.1 概要.....390
27.2 接続.....390
27.3 プロジェクトの構成.....391
27.4 プログラム「uart0_printf.c」.....391

27.5 printf 文、scanf 文を使おう！ 393
27.6 プログラムの解説「printf_lib.c」....394
27.6.1 「printf_lib.c」の追加 394
27.6.2 関数一覧....396
27.7 プログラムの解説「uart0_printf.c」....397
27.7.1 インクルード部分 397
27.7.2 main 関数（初期化部分）....397
27.7.3 main 関数（通信部分）....398
27.7.4 受信バッファのクリア....398

28. データフラッシュ（プロジェクト：data_flash）....400

28.1 概要....400
28.2 接続....400
28.3 プロジェクトの構成 401
28.4 プログラム「data_flash.c」....401
28.5 データフラッシュを使おう！....403
28.5.1 概要 403
28.5.2 データフラッシュのメモリマップ....404
28.5.3 データフラッシュを効率的に使う....405
28.6 プログラムの解説「data_flash_lib.c」....406
28.6.1 「data_flash_lib.c」の追加....406
28.6.2 関数一覧....407
28.7 プログラムの解説「data_flash.c」....409
28.7.1 インクルード部分 409
28.7.2 変数領域....409
28.7.3 main 関数（データフラッシュから読み込み）....410
28.7.4 main 関数（データフラッシュのイレーズ、書き込み）....411
28.7.5 main 関数（無限ループ部分）....411
28.8 実習手順....412
28.9 演習....413

29. 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）....414

29.1 概要....414
29.2 接続....414
29.3 プロジェクトの構成 415
29.4 プログラム「timer_rc_pwm.c」....416
29.5 プログラムの解説 420
29.5.1 init 関数(タイマ RC の設定)....420
29.5.2 beep 関数....436
29.5.3 intTRB 関数（タイマ RB 割り込み）....439
29.5.4 main 関数....440
29.6 音階....440
29.6.1 音階の周波数 440
29.6.2 音階名の定義 442
29.6.3 音階の時間の長さ 443
29.6.4 音データ 444

30. モータのスピードをボリュームで制御（プロジェクト：ad_pwm）....445

30.1 概要....445
30.2 接続....445

30.3 プロジェクトの構成 447
30.4 プログラム「ad_pwm.c」 447
30.5 プログラムの解説 449
30.5.1 data_cnv 関数（データ変換関数） 449
30.5.2 main 関数 450

31. タイマ RE によるリアルタイムクロック（プロジェクト：timer_re_rtc） 452

31.1 概要 452
31.2 接続 452
31.3 プロジェクトの構成 453
31.4 プログラム「timer_re_rtc.c」 453
31.5 プログラムの解説 455
31.5.1 init 関数(XCIN クロック(32.768kHz)を有効にする) 455
31.5.2 init 関数(ポートの入出力設定) 460
31.5.3 init 関数(タイマ RE の設定) 461
31.5.4 intTRE 関数(1 秒ごとに実行される関数) 474
31.5.5 main 関数 475
31.5.6 ビジーフラグと値の読み込み方 475

32. パルスカウント（プロジェクト：timer_ra_counter） 476

32.1 概要 476
32.2 接続 476
32.3 プロジェクトの構成 477
32.4 プログラム「timer_ra_counter.c」 478
32.5 プログラムの解説 479
32.5.1 init 関数（ポートの入出力設定） 479
32.5.2 init 関数(タイマ RA の設定) 479
32.5.3 main 関数 486

33. パルスカウント（プロジェクト：timer_rd_counter） 487

33.1 概要 487
33.2 接続 487
33.3 プロジェクトの構成 488
33.4 プログラム「timer_rd_counter.c」 488
33.5 プログラムの解説 489
33.5.1 init 関数（ポートの入出力設定） 489
33.5.2 init 関数(タイマ RD の設定) 490
33.5.3 main 関数 498

34. タイマ RE による割り込み（プロジェクト：timer_re_timerwarikomi） 499

34.1 概要 499
34.2 接続 499
34.3 プロジェクトの構成 500
34.4 プログラム「timer_re_timerwarikomi.c」 500
34.5 プログラムの解説 502
34.5.1 init 関数(タイマ RE の設定) 502
34.5.2 intTRE 関数（1ms ごとに実行される関数） 510
34.5.3 timer_re 関数（割り込みを使った時間稼ぎ） 512
34.5.4 main 関数 512

35. 高速オンチップオシレータの使用（プロジェクト：hoco_io）.....513
35.1 概要.....513
35.2 接続.....513
35.3 プロジェクトの構成.....514
35.4 プログラム「hoco_io.c」.....514
35.5 プログラムの解説.....515
35.5.1 ヘッダファイルの取り込み.....515
35.5.2 init 関数(クロックの選択).....516
35.5.3 その他のプログラム.....521
36. 付録.....522
36.1 R8C マイコンの変数のサイズ.....522
36.2 演算子.....523
36.3 優先順位.....524
36.4 型が混合したときの演算.....525
36.5 printf 関数の使い方.....526
36.6 scanf 関数の使い方.....528
36.7 プログラムの容量が 48KB 以上になったとき.....529
36.7.1 概要.....529
36.7.2 データフラッシュを使う.....530
36.7.3 走りに関係しない関数を 0x10000 番地から配置する.....531
36.8 switch 文が長い場合（L2330 (E) Relocation size overflow エラー）.....533
36.9 コンパイラオプションの最適化について.....534
36.9.1 概要.....534
36.9.2 最適化を変更する.....534
36.9.3 最適化すると動作しなくなるプログラム例と対策.....536
37. 参考文献.....537

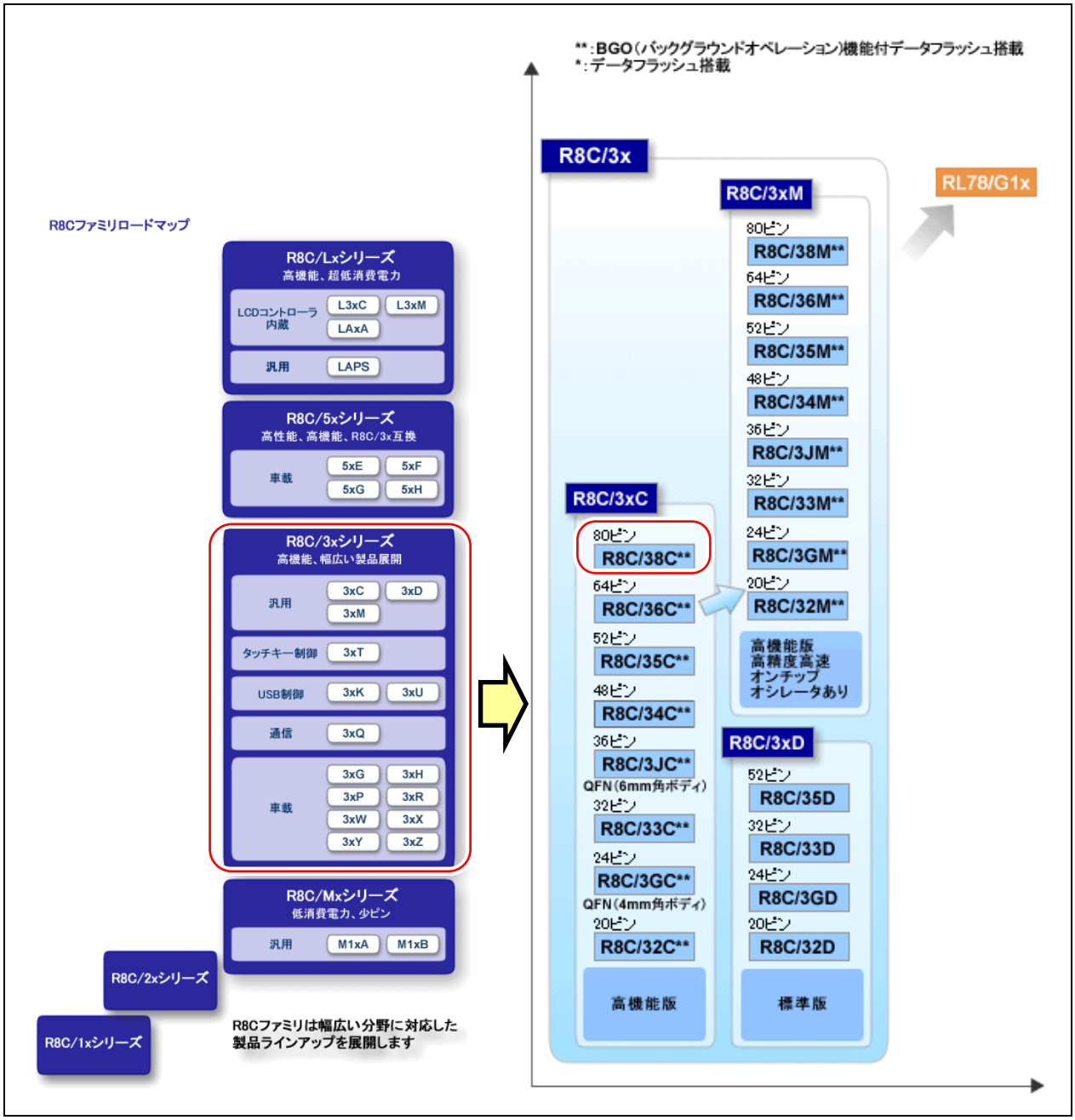
1. R8C/38C マイコンの仕様

2013 年度から、RY_R8C38 ボードに搭載されているマイコンが R8C/38A から R8C/38C に変更されました。R8C/38A マイコンと R8C/38C マイコンは、機能的にほぼ互換で、マイコンカーで使う範囲においてはプログラムの変更はほとんどありません。

本マニュアルでは『R8C/38A』のマイコン名称が残っていますが、すべて『R8C/38A および R8C/38C』の意味になります。

1.1 シリーズ展開

R8C/38C マイコンは、ルネサス エレクトロニクス製の R8C ファミリに属したマイコンです。R8C ファミリのシリーズ展開を下記に示します。



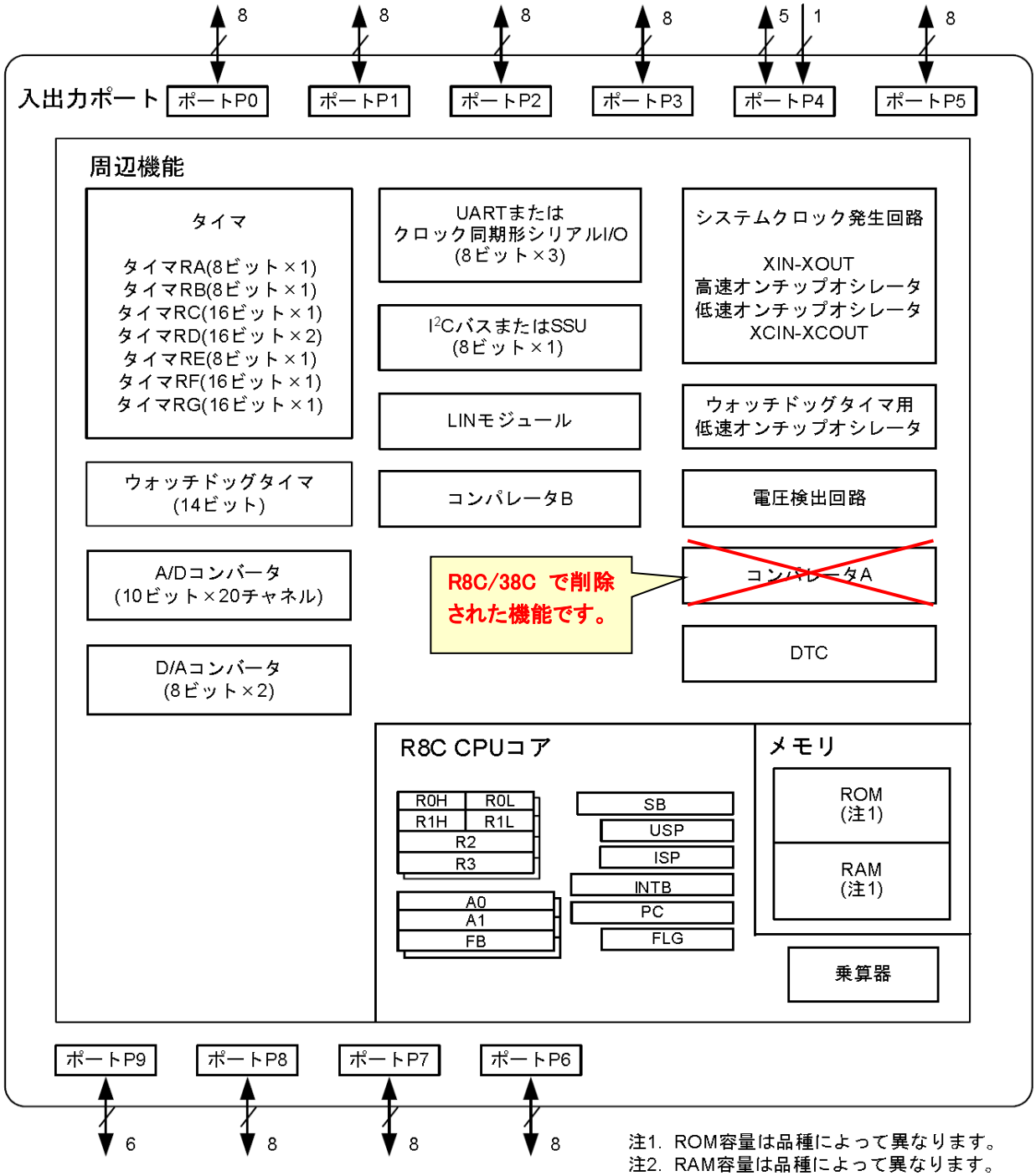
▲ルネサス エレクトロニクスのホームページより（2013 年 5 月現在）

1.2　R8C/38C の特徴

R8C/38C 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 ビットプリスケーラ付 8 ビット多機能タイマ（タイマ RA、RB）：2 チャネル
- 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×2、RF、RG）：5 チャネル
- 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 チャネル
- UART/クロック同期形シリアルインタフェース：3 チャネル
- I^2C-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 チャネル
- LIN モジュール：1 チャネル（タイマ RA、UART0 を使用）
- 10 ビット A/D コンバータ：20 チャネル
- 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　**R8C/38C で削除された機能です**
- DTC（データトランスファコントローラ）：1 チャネル
- ウォッチドッグタイマ　**R8C/38C で追加された機能です**
- クロック発生回路：XIN クロック発振回路、オンチップオシレータ（**高速**/低速）、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：75 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

1.3　ブロック図



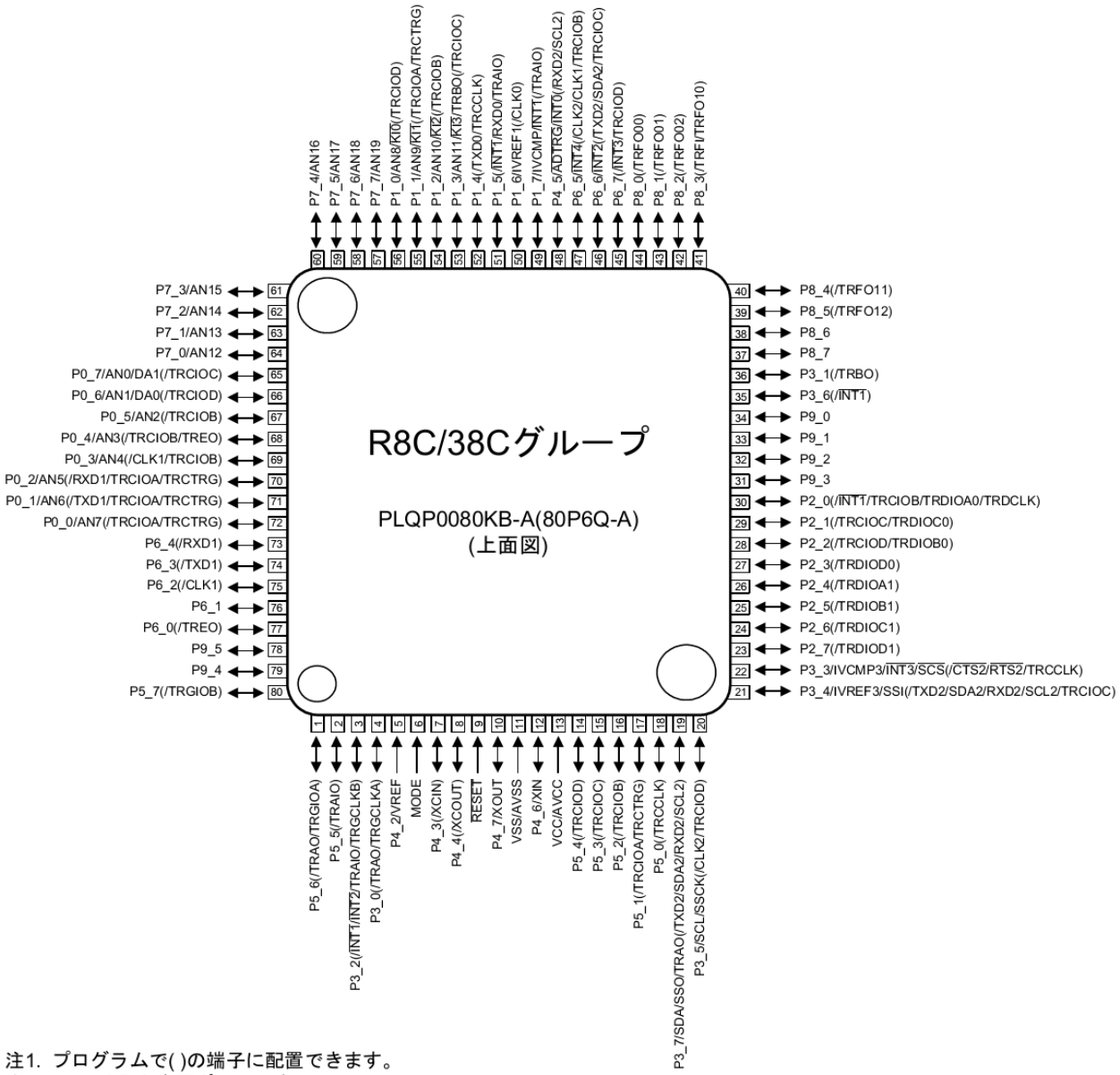
注1. ROM容量は品種によって異なります。
注2. RAM容量は品種によって異なります。

※ポイント

・ポートは 0～9 まで、10 ポートあります。

・ポート 4、ポート 9 以外は、入出力可能な端子が 8 端子あるポートです。

1.4 ピン配置

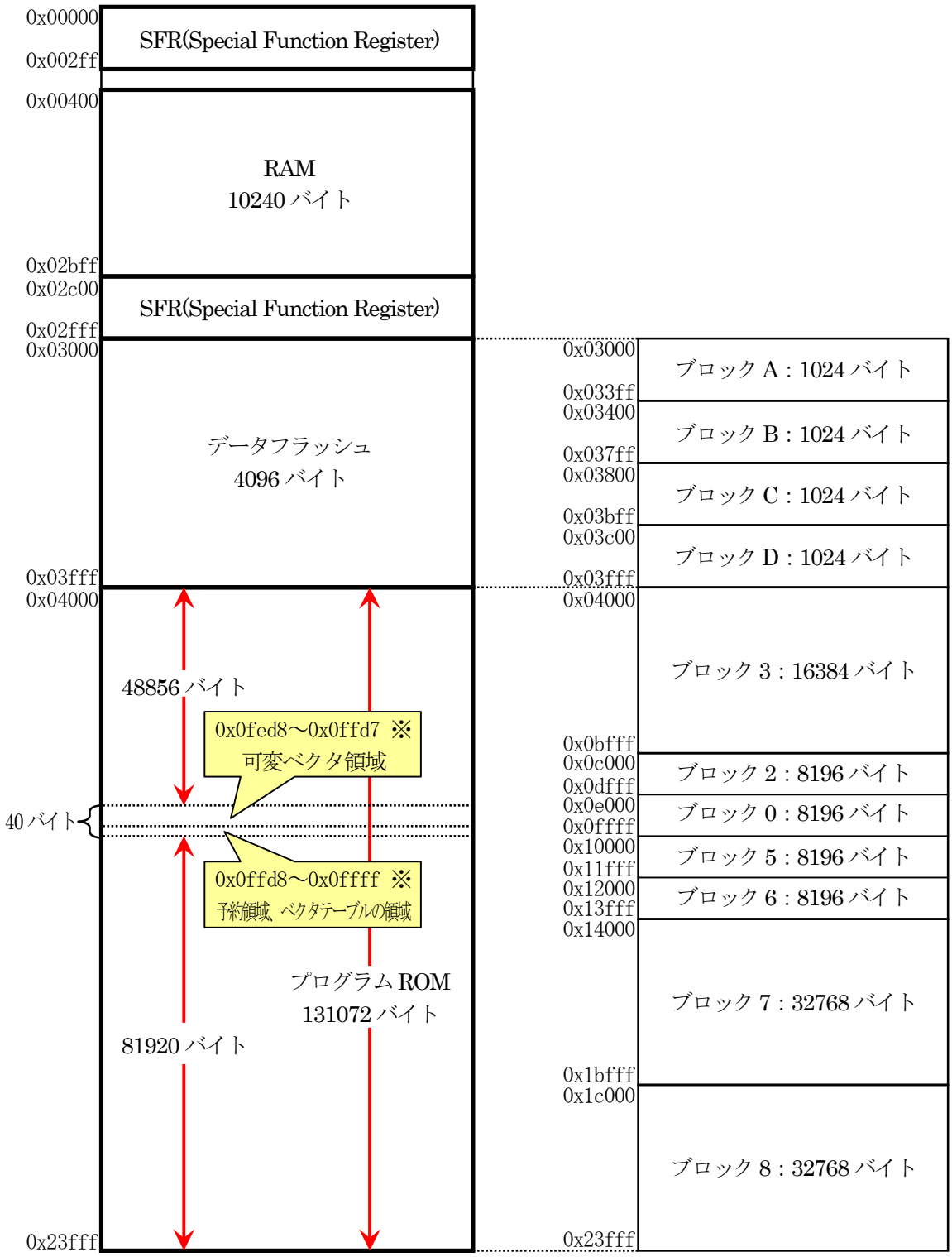


注1. プログラムで( )の端子に配置できます。
注2. P4_2は入力専用ポートです。
注3. パッケージの1ピンの位置は「外形寸法図」で確認してください。

※Pxy は、ポートx の bity という意味です。
例）P1_0→ポート 1 の bit0、P9_5→ポート 9 の bit5

1.5　メモリマップ

今回使用する R8C/38C(R5F2138CCNFP)は、プログラム ROM：128KB、データフラッシュ：4KB、RAM：10KB が内蔵されています。



※0x0fed8～0x0ffd7 番地は、今回のプログラムでは可変ベクタテーブルを置いているため、プログラムは置けません。

※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。
※プログラムは、0x04000～0x0fed7 番地(約 48KB)と 0x10000～0x23fff 番地(約 80KB)に分ける必要があります。連続させることはできません。また、0x10000 番地以降は、プログラムの命令によっては実行スピードが遅くなることがあります(アセンブリ言語の MOV 命令（一般命令）が LDE(特殊命令)に変わるなど、します)。プログラムはできるだけ、0x04000～0x0ffd7 番地に置いてください。
※プログラムが 48KB を超えて、エラーになった場合の対応は、「**34. 付録**」を参照してください。

1.6 割り込みベクタテーブル領域

R8C/38C マイコンには、2 つのベクタテーブル領域があります。

ベクタテーブル領域	内容
固定割り込みベクタテーブル	0x0ffdc～0x0ffff 番地には、下記の割り込みが発生したときのジャンプ先アドレスを記載しておく領域です。アドレスは決められています。 ・未定義命令割り込み発生時のジャンプ先アドレス ・オーバフロー割り込み発生時のジャンプ先アドレス ・BRK 命令割り込み発生時のジャンプ先アドレス ・アドレス一致割り込み発生時のジャンプ先アドレス ・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 ・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 ・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 ・リセット時のジャンプ先アドレス 詳しい内容は、「**1.7 固定割り込みベクタテーブル領域**」を参照してください。
可変ベクタテーブル	割り込みテーブルレジスタ(INTB)の内容で示された値を先頭アドレスとする 256 バイトの割り込みベクタテーブルです。 ベクタテーブルは、SFR 領域以外のメモリ空間に配置することができます。各ベクタは 4 バイトで構成され、0～63 までのソフトウェア割り込み番号が割り付けられています。 今回のツールチェインの設定では、可変ベクタテーブルのセクション名を vector、アドレスは 0xfed8 番地に割り当てています。 詳しい内容は、「**1.8 可変ベクタテーブル領域**」を参照してください。

1.7 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。

1.8　可変ベクタテーブル領域

可変ベクタテーブルの割り込み要因とソフトウェア割り込み番号の関係は、下記のとおりです。

割り込み要因	ベクタ番号(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
BRK 命令(注 3)	+0～　+3 (0000h～0003h)	0	
フラッシュメモリレディ	+4～　+7 (0004h～0007h)		FMRDYIC
—(予約)		2～5	
$\overline{\text{INT4}}$	+24～ +27 (0018h～001Bh)	6	INT4IC
タイマ RC	+28～ +31 (001Ch～001Fh)	7	TRCIC
タイマ RD0	+32～ +35 (0020h～0023h)	8	TRD0IC
タイマ RD1	+36～ +39 (0024h～0027h)	9	TRD1IC
タイマ RE	+40～ +43 (0028h～002Bh)	10	TREIC
UART2 送信/NACK2	+44～ +47 (002Ch～002Fh)	11	S2TIC
UART2 受信/ACK2	+48～ +51 (0030h～0033h)	12	S2RIC
キー入力	+52～ +55 (0034h～0037h)	13	KUPIC
A/D 変換	+56～ +59 (0038h～003Bh)	14	ADIC
シンクロナスシリアルコミュニケーション ユニット/I2C バスインタフェース(注 2)	+60～ +63 (003Ch～003Fh)	15	SSUIC/ IICIC
タイマ RF コンペア 1	+64～ +67 (0040h～0043h)	16	CMP1IC
UART0 送信	+68～ +71 (0044h～0047h)	17	S0TIC
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
$\overline{\text{INT2}}$	+84～ +87 (0054h～0057h)	21	INT2IC
タイマ RA	+88～ +91 (0058h～005Bh)	22	TRAIC
— (予約)		23	
タイマ RB	+96～ +99 (0060h～0063h)	24	TRBIC
$\overline{\text{INT1}}$	+100～+103 (0064h～0067h)	25	INT1IC
$\overline{\text{INT3}}$	+104～+107 (0068h～006Bh)	26	INT3IC
タイマ RF	+108～+111 (006Ch～006Fh)	27	TRFIC
タイマ RF コンペア 0	+112～+115 (0070h～0073h)	28	CMP0IC
$\overline{\text{INT0}}$	+116～+119 (0074h～0077h)	29	INT0IC
UART2 バス衝突検出	+120～+123 (0078h～007Bh)	30	U2BCNIC
タイマ RF キャプチャ	+124～+127 (007Ch～007Fh)	31	CAPIC
ソフトウェア(注 3)	+128～+131 (0080h～0083h)～ +164～+167 (00A4h～00A7h)	32～41	
— (予約)		42	
タイマ RG	+172～+175 (00ACh～00AFh)	43	TRGIC
— (予約)		44～49	
電圧監視 1~~/コンパレータ A1~~	+200～+203 (00C8h～00CBh)	50	VCMP1IC
電圧監視 2~~/コンパレータ A~~	+204～+207 (00CCh～00CFh)	51	VCMP2IC
— (予約)		52～55	
ソフトウェア(注 3)	+224～+227 (00E0h～00E3h)～ +252～+255 (00FCh～00FFh)	56～63	

R8C/38C で削除された機能です。

注 1. INTB レジスタが示す番地からの相対番地です。
注 2. SSUIICSR 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。

1.9 CPU レジスタ

R8C/38C 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	0x0fffc～0xfffe 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、0x0fffc～0xfffe 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

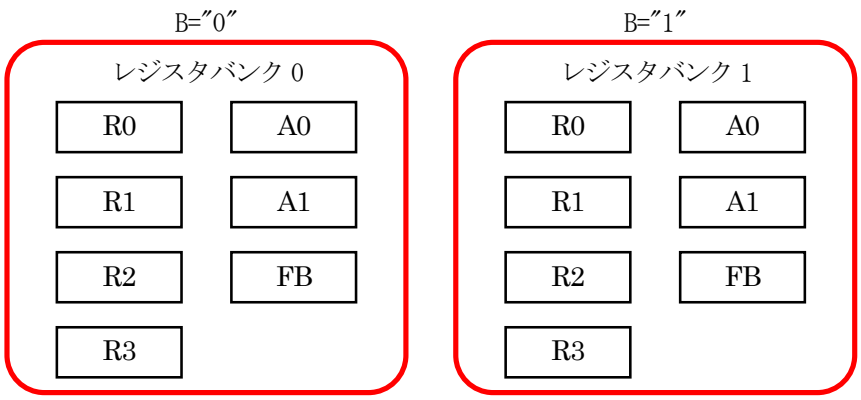
フラグレジスタのビット構成を下記に示します。

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。



1.10 タイマの概要

R8C/38C には、タイマ RA～タイマ RG という名称の 7 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 ビットプリスケーラ付)×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 ビットプリスケーラ付)×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
タイマ RF	16 ビット×1 インプットキャプチャモード(インプットキャプチャ機能)、 アウトプットコンペアモード(アウトプットコンペア機能)
タイマ RG	16 ビット×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 1 本)、位相計数モード(2 相エンコーダのカウント数の自動計測が可能)

1.11 タイマ RA～RG の使用用途

タイマ RA～RG を、どの用途で使用できるかまとめた表を、下記に示します。「◎」は、マイコンカー（オプション含む）で使用している用途です。

	タイマ RA (8bit)	タイマ RB (8bit)	タイマ RC (16bit)	タイマ RD (16bit)	タイマ RE (8bit)	タイマ RF (16bit)	タイマ RG (16bit)
インターバル割り込み	○	◎	○	○	○	○	○
PWM バッファあり		○ 1ch	○ 1ch	◎ 3ch			
PWM バッファなし		○ 1ch	◎ 3ch	○ 6ch	△ デューティ 50%固定	○ 1ch	○ 1ch
パルス幅測定	○ 1ch		○ 1ch	○ 4ch		○ 1ch	○ 1ch
パルス周期測定	○ 1ch						
パルスカウント	○ 立ち上がり、または立ち下がりのみ			○ 両エッジ可能			◎ 両エッジ可能
位相計数モード							○
時計					○		

マイコンカーでは、次のように使用しています。

タイマ名	詳細
タイマ RB	1ms ごとの割り込みで使用しています。
タイマ RD	PWM 信号を 3 チャネル出力して、左モータ、右モータ、サーボを制御します。
タイマ RG	ロータリエンコーダのパルスカウントをして、スピード制御、距離制御をします。
タイマ RC	モータドライブ基板 TypeS で左前モータ、右前モータを制御します。

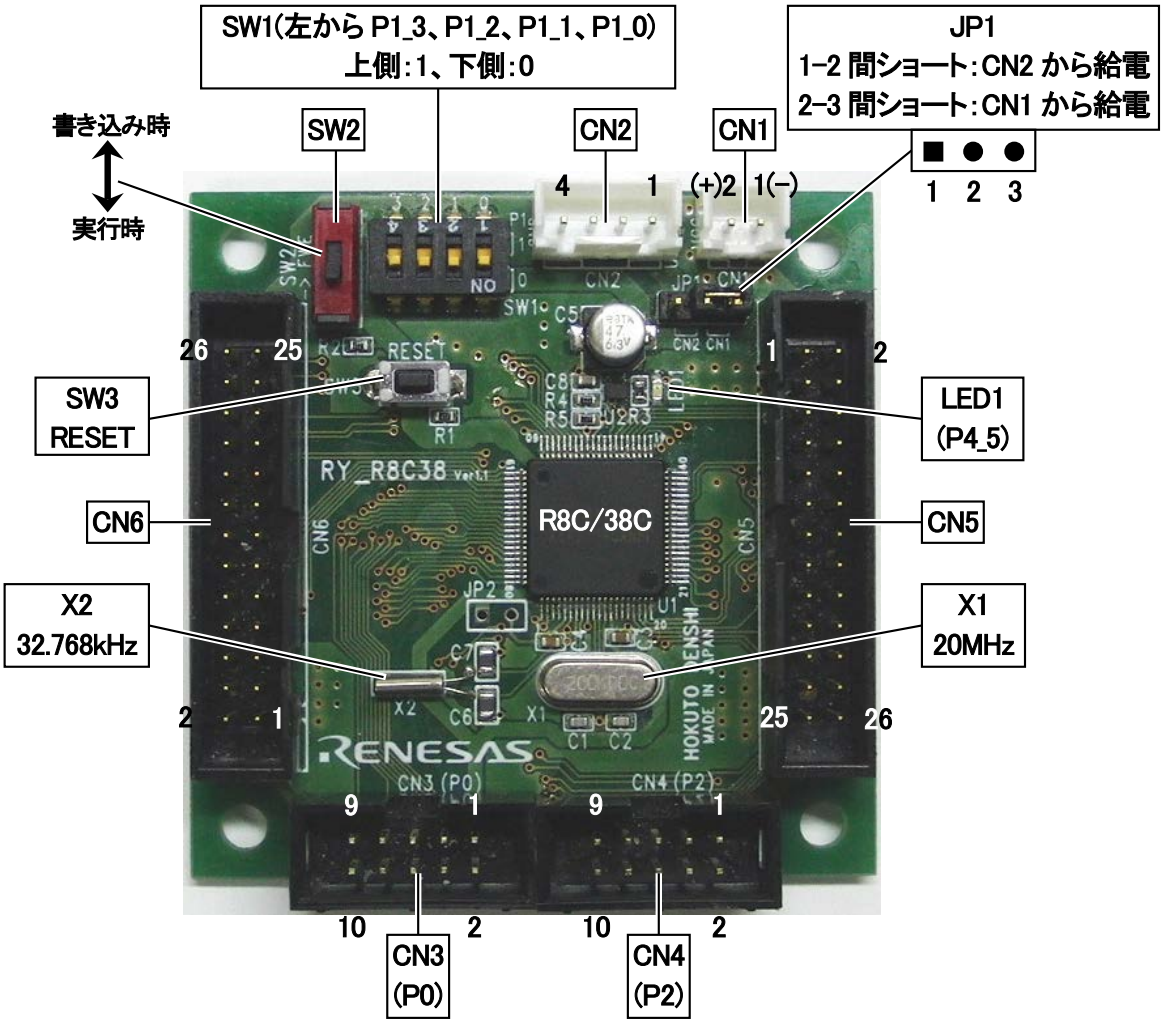
2. RY_R8C38 ボードの仕様

2.1 RY_R8C38 ボードの仕様

RY_R8C38 ボードの仕様を下表に示します。参考に RY3048Fone ボードの仕様と比較します。

<table>
<tr><th colspan="2"></th><th>RY_R8C38ボード（本ボード）</th><th>RY3048Foneボード</th></tr>
<tr><td colspan="2">マイコン</td><td>ルネサス エレクトロニクス製
R8C/38C （R5F2138CCNFP）</td><td>ルネサス エレクトロニクス製
H8/3048F-ONE</td></tr>
<tr><td colspan="2">パッケージ</td><td>80ピンLQFP</td><td>100ピンQFP</td></tr>
<tr><td colspan="2">ボードのクリスタル値</td><td>20.0MHz</td><td>24.576MHz</td></tr>
<tr><td colspan="2">動作電圧</td><td>動作周波数が5.0～20.0MHzの場合：2.7～5.5V
動作周波数が5.0MHz以下の場合：1.8～5.5V</td><td>4.5～5.5V</td></tr>
<tr><td colspan="2">基板外形</td><td>W60×D60×H13mm(実寸)</td><td>W70×D59×H13mm(実寸)</td></tr>
<tr><td rowspan="3">内蔵メモリ</td><td>ROM</td><td>128KB：0x4000～0x23fff番地
プログラム、イレーズ(消去)回数：保証回数1000回</td><td>128KB：0x00000～0x1ffff番地
プログラム、イレーズ(消去)回数：保証回数100回</td></tr>
<tr><td>RAM</td><td>10KB：0x0400～0x2bff番地</td><td>4KB：0xfef10～0xfff0f番地</td></tr>
<tr><td>データフラッシュ</td><td>4KB：0x3000～0x3fff番地
プログラム、イレーズ(消去)回数：保証回数10000回</td><td>なし</td></tr>
<tr><td colspan="2">アドレス空間</td><td>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。</td><td>外部メモリの接続は可能</td></tr>
<tr><td colspan="2">マイコンの
動作周波数</td><td>次の~~3~~ 4つから選べます。
●高速オンチップオシレータ(内蔵クリスタル)
40MHz±5%　R8C/38Cで追加
※マイコンの動作周波数として使用する場合は
1/2して使用します。よって20MHzとなります。
●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）※オプションです
※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。</td><td>外付けクリスタルにて
2.0～25.0MHz</td></tr>
<tr><td colspan="2">コネクタ</td><td><table><tr><th>コネクタ(ポート)</th><th>I/O 数</th></tr><tr><td>CN3(ポート 0)</td><td>8</td></tr><tr><td>CN4(ポート 2)</td><td>8</td></tr><tr><td>CN5(ポート 3, 5, 6)</td><td>24</td></tr><tr><td>CN6(ポート 7, 8 など)</td><td>24</td></tr><tr><td>合計</td><td>64</td></tr></table></td><td><table><tr><th>コネクタ(ポート)</th><th>I/O 数</th></tr><tr><td>J1(ポート 7)</td><td>8(入力専用)</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></table></td></tr>
<tr><td colspan="2">プログラム書き込み
信号レベル</td><td>TTLレベル信号</td><td>RS-232C信号
(基板内に、RS-232C→TTLレベル変換ICを搭載)</td></tr>
<tr><td colspan="2">ディップスイッチ</td><td>P1_3～P1_0に接続(4ビット分)
※P1_3…ポート1のbit3　という意味です。</td><td>P63～P60に接続(4ビット分)
※P63…ポート6のbit3　という意味です。</td></tr>
</table>

2.2　外観

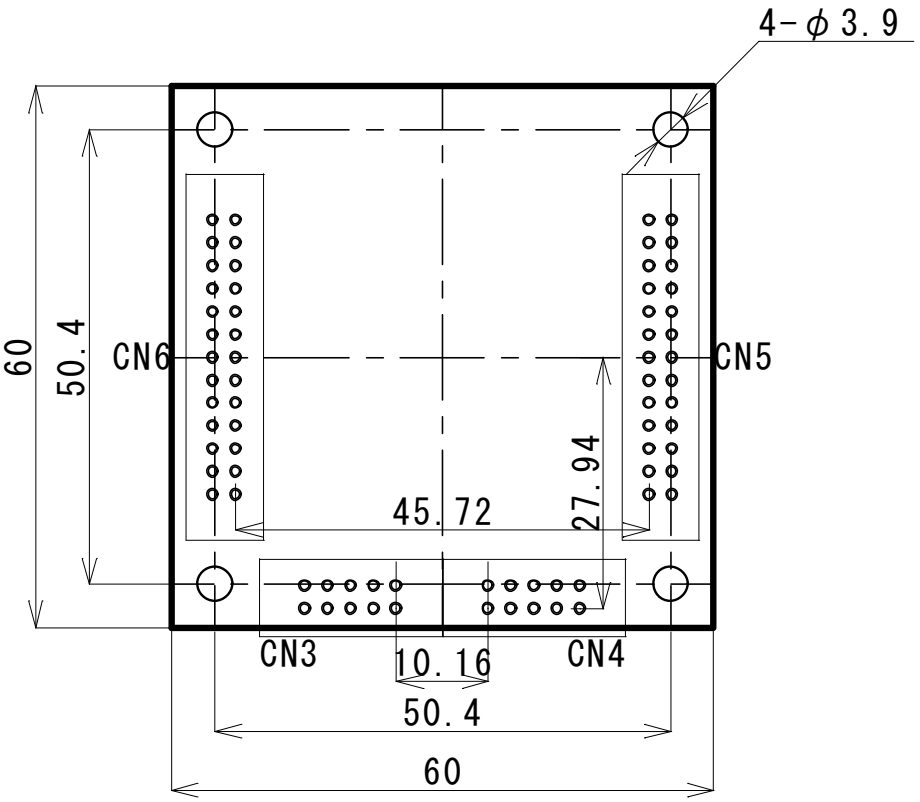


※CN3、CN4 に実装するコネクタは同封されているので、各自実装してください。
※CN5、CN6、X2、C6、C7 はオプションです。

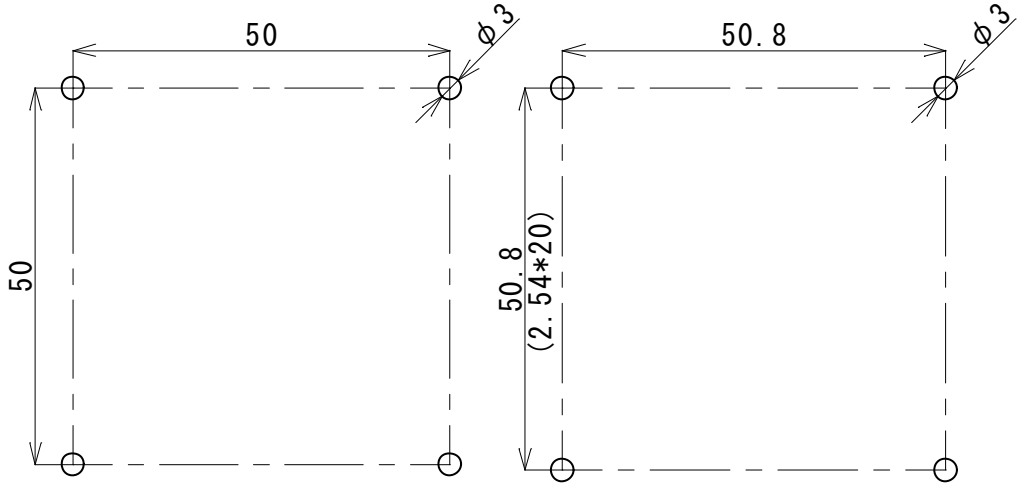
部品番号	説明
CN1	電源コネクタです。1 ピン:GND、2 ピン:＋電源を供給します。**動作保証電圧は 2.7～5.5V(5MHz 以上で動作の場合)なので、5.5V 以上の電圧を加えないようにしてください(絶対最大定格は 6.5V です)**。 電池 4 本の電圧を直接供給する場合、 充電電池の場合 …… 4 本×1.2V＝4.8V → OK アルカリ電池の場合 … 4 本×1.5V＝6.0V → **NG** と、**アルカリ電池 4 本では、5.5V を超えるので、直接アルカリ電池 4 本を RY_R8C38 ボードに加えないようにしてください。**
CN2	TTL レベルシリアル信号コネクタです。パソコンからの TTL レベルシリアル信号をこのコネクタに接続して、プログラムの書き込みやシリアル通信を行います。 **※RY3048Fone ボードで使っていた通信ケーブル(3P)は、RS-232C レベルです。信号の電圧が違うため、接続することはできません。接続すると、RY_R8C38 ボードが壊れますので絶対に RY3048Fone ボードで使っていた通信ケーブル(3P)を接続しないでください。**

CN3	10 ピンコネクタです。マイコンのポート 0 に接続されています。 RY_R8C38 ボード開封時、このコネクタは実装されていません。このポートを使う場合はコネクタを実装してください。10 ピンオスコネクタを使う場合は、同封のコネクタを実装してください。同封のコネクタ以外を使いたい場合は、各自で用意して実装してください。
CN4	10 ピンコネクタです。マイコンのポート 2 に接続されています。 RY_R8C38 ボード開封時、このコネクタは実装されていません。このポートを使う場合はコネクタを実装してください。10 ピンオスコネクタを使う場合は、同封のコネクタを実装してください。同封のコネクタ以外を使いたい場合は、各自で用意して実装してください。
CN5	26 ピンコネクタです。マイコンのポート 3、ポート 5、ポート 6 に接続されています。 このコネクタは同封、実装されていません。このポートを使う場合は、各自で用意したコネクタを実装してください。
CN6	26 ピンコネクタです。マイコンのポート 1 の 2bit 分、ポート 7、ポート 8、ポート 9 の 6bit 分に接続されています。 このコネクタは同封、実装されていません。このポートを使う場合は、各自で用意したコネクタを実装してください。
SW1	4bit のディップスイッチです。マイコンとの接続は、左からマイコンの P1_3、P1_2、P1_1、P1_0 です。 スイッチを上側にすると"1"、下側にすると"0"の信号がそれぞれの端子から読み込まれます。 **※プルアップ抵抗が接続されていません。プルアップ制御レジスタでこれらの端子のプルアップを ON にして使ってください。**
SW2	プログラム書き込みスイッチです。 このスイッチを上側（FWE と書いてある側）にしてリセット（電源 ON）すると、マイコンは書き込みモードで立ち上がります。パソコンからプログラムを転送してプログラムを書き込んでください。 このスイッチを下側（FWE と書いてない側）にしてリセット（電源 ON）すると、マイコンは書き込んだプログラムを実行します。
SW3	リセットスイッチです。 マイコンをリセットして、いちばん最初のプログラムから実行します。
LED1	この LED は、P4_5 に接続されています。LED の点灯、消灯条件を下記に示します。 • 点灯条件：P4_5 が入力端子になっているか、"1"が出力された場合 • 消灯条件：P4_5 から"0"が出力された場合 プログラム書き込み時、またはリセット後にポートの入出力設定をしていない状態では、P4_5 は入力端子になっているため点灯します。
X1	XINクロック入力端子に接続されているクリスタルです。 20.0MHz のクリスタルが実装されています。この周波数でマイコンは動作します。
X2	XCINクロック入力端子に接続されているクリスタルです。 X2、C6、C7は実装されていません。時計機能を使う場合は、次の部品を各自実装してください。 X1：32.768kHz のクリスタル(実装できるサイズ) C6、C7：15pF のチップコンデンサ(1608 サイズ)
JP1	電源供給切り替え用ジャンパです。 • 1-2 ピン間をショート RY_R8C38 ボードへの電源供給は、CN2 コネクタを経由して行います。CN2 は、パソコンの USB 電源です。パソコンの電源で RY_R8C38 ボードを動作させることができるので便利ですが、USB 電源は 500mA 以上（または接続先の規定以上）の電流を流せません。それ以上の電流が流れないよう、コネクタの接続など気をつけてください。 • 2-3 ピン間をショート CN1 コネクタから電源を供給します。

2.3 寸法



基板の四隅の穴は、50.4mm 間隔でφ3.9mm で開いています。これは、50mm 間隔（下図左）または、50.8(2.54×20)mm 間隔（下図右）の固定用の穴の両方に取り付けられる寸法です。



2.4 ポート表

※~~取消線赤色~~は、R8C/38C で削除された端子です。

Port	bit	端子名	接続先	マイコンカー接続先
P0	7	P0_7/AN0/DA1(/TRCIOC)	CN3.2	センサ基板
P0	6	P0_6/AN1/DA0(/TRCIOD)	CN3.3	センサ基板
P0	5	P0_5/AN2(/TRCIOB)	CN3.4	センサ基板
P0	4	P0_4/AN3(/TRCIOB/TREO)	CN3.5	センサ基板
P0	3	P0_3/AN4(/CLK1/TRCIOB)	CN3.6	センサ基板
P0	2	P0_2/AN5(/RXD1/TRCIOA/TRCTRG)	CN3.7	センサ基板
P0	1	P0_1/AN6(/TXD1/TRCIOA/TRCTRG)	CN3.8	センサ基板
P0	0	P0_0/AN7(/TRCIOA/TRCTRG)	CN3.9	センサ基板
P1	7	P1_7/IVCMP/INT1(/TRAIO)	CN6.18	
P1	6	P1_6/~~LVCOUT2~~/IVREF1(/CLK0)	CN6.19	
P1	5	P1_5(/INT1/RXD0/TRAIO)	CN2.3(シリアルデータ受信)	
P1	4	P1_4(/TXD0/TRCCLK)	CN2.2(シリアルデータ送信)	
P1	3	P1_3/AN11/~~LVCOUT1~~/KI3/TRBO(/TRCIOC)	DIP SW3	
P1	2	P1_2/AN10/LVREF/KI2(/TRCIOB)	DIP SW2	
P1	1	P1_1/AN9/~~LVCMP2~~/KI1(/TRCIOA/TRCTRG)	DIP SW1	
P1	0	P1_0/AN8/~~LVCMP1~~/KI0(/TRCIOD)	DIP SW0	
P2	7	P2_7(/TRDIOD1)	CN4.2	モータドライブ基板
P2	6	P2_6(/TRDIOC1)	CN4.3	モータドライブ基板
P2	5	P2_5(/TRDIOB1)	CN4.4	モータドライブ基板(PWM)
P2	4	P2_4(/TRDIOA1)	CN4.5	モータドライブ基板(PWM)
P2	3	P2_3(/TRDIOD0)	CN4.6	モータドライブ基板
P2	2	P2_2(/TRCIOD/TRDIOB0)	CN4.7	モータドライブ基板(PWM)
P2	1	P2_1(/TRCIOC/TRDIOC0)	CN4.8	モータドライブ基板
P2	0	P2_0(/INT1/TRCIOB/TRDIOA0/TRDCLK)	CN4.9	モータドライブ基板
P3	7	P3_7/SDA/SSO/TRAO(/TXD2/SDA2/RXD2/SCL2)	CN5.2	EEP-ROM(SDA)
P3	6	P3_6(/INT1)	CN5.3	
P3	5	P3_5/SCL/SSCK(/CLK2/TRCIOD)	CN5.4	EEP-ROM(SCL)
P3	4	P3_4/IVREF3/SSI(/TXD2/SDA2/RXD2/SCL2/TRCIOC)	CN5.5	
P3	3	P3_3/IVCMP3/INT3/SCS(/CTS2/RTS2/TRCCLK)	CN5.6	
P3	2	P3_2(/INT1/INT2/TRAIO/TRGCLKB)	CN5.7	エンコーダ(TRGCLKB)
P3	1	P3_1(/TRBO)	CN5.8	
P3	0	P3_0(/TRAO/TRGCLKA)	CN5.9	エンコーダ(TRGCLKA)
P4	7	P4_7/XOUT	Xout	
P4	6	P4_6/XIN	Xin	
P4	5	P4_5/ADTRG/INT0(/RXD2/SCL2)	LED	
P4	4	P4_4(/XCOUT)	Xcout(未接続)	
P4	3	P4_3(/XCIN)	Xcin(未接続)	
P4	2	P4_2/VREF	Vcc	
P5	7	P5_7(/TRGIOB)	CN5.10	
P5	6	P5_6(/TRAO/TRGIOA)	CN5.11	液晶 E
P5	5	P5_5(/TRAIO)	CN5.12	液晶 RW
P5	4	P5_4(/TRCIOD)	CN5.13	液晶 RS
P5	3	P5_3(/TRCIOC)	CN5.14	液晶 D3
P5	2	P5_2(/TRCIOB)	CN5.15	液晶 D2
P5	1	P5_1(/TRCIOA/TRCTRG)	CN5.16	液晶 D1
P5	0	P5_0(/TRCCLK)	CN5.17	液晶 D0
P6	7	P6_7(/INT3/TRCIOD)	CN5.18	
P6	6	P6_6/INT2(/TXD2/SDA2/TRCIOC)	CN5.19	
P6	5	P6_5/INT4(/CLK2/CLK1/TRCIOB)	CN5.20	
P6	4	P6_4(/RXD1)	CN5.21	microSD(/RXD1)
P6	3	P6_3(/TXD1)	CN5.22	microSD(/TXD1)
P6	2	P6_2(/CLK1)	CN5.23	microSD(/CLK1)
P6	1	P6_1	CN5.24	microSD
P6	0	P6_0(/TREO)	CN5.25	

Port	bit	端子名	接続先	マイコンカーキット
P7	7	P7_7/AN19	CN6.2	
P7	6	P7_6/AN18	CN6.3	
P7	5	P7_5/AN17	CN6.4	
P7	4	P7_4/AN16	CN6.5	
P7	3	P7_3/AN15	CN6.6	
P7	2	P7_2/AN14	CN6.7	
P7	1	P7_1/AN13	CN6.8	
P7	0	P7_0/AN12	CN6.9	
P8	7	P8_7	CN6.10	
P8	6	P8_6	CN6.11	
P8	5	P8_5(/TRFO12)	CN6.12	
P8	4	P8_4(/TRFO11)	CN6.13	
P8	3	P8_3(/TRFI/TRFO10)	CN6.14	
P8	2	P8_2(/TRFO02)	CN6.15	
P8	1	P8_1(/TRFO01)	CN6.16	
P8	0	P8_0(/TRFO00)	CN6.17	
P9	5	P9_5	CN6.20	
P9	4	P9_4	CN6.21	
P9	3	P9_3	CN6.22	
P9	2	P9_2	CN6.23	
P9	1	P9_1	CN6.24	
P9	0	P9_0	CN6.25	

2.5 コネクタ信号表

CN1(CLP2502-0101F)

1	GND
2	+5V

CN2(CLP2504-0101F)

1	+5V
2	P1_4(TxD0)
3	P1_5(RxD0)
4	GND

CN3(10ピンボックスコネクタ)

1	+5V
2	P0_7
3	P0_6
4	P0_5
5	P0_4
6	P0_3
7	P0_2
8	P0_1
9	P0_0
10	GND

CN4(10ピンボックスコネクタ)

1	+5V
2	P2_7
3	P2_6
4	P2_5
5	P2_4
6	P2_3
7	P2_2
8	P2_1
9	P2_0
10	GND

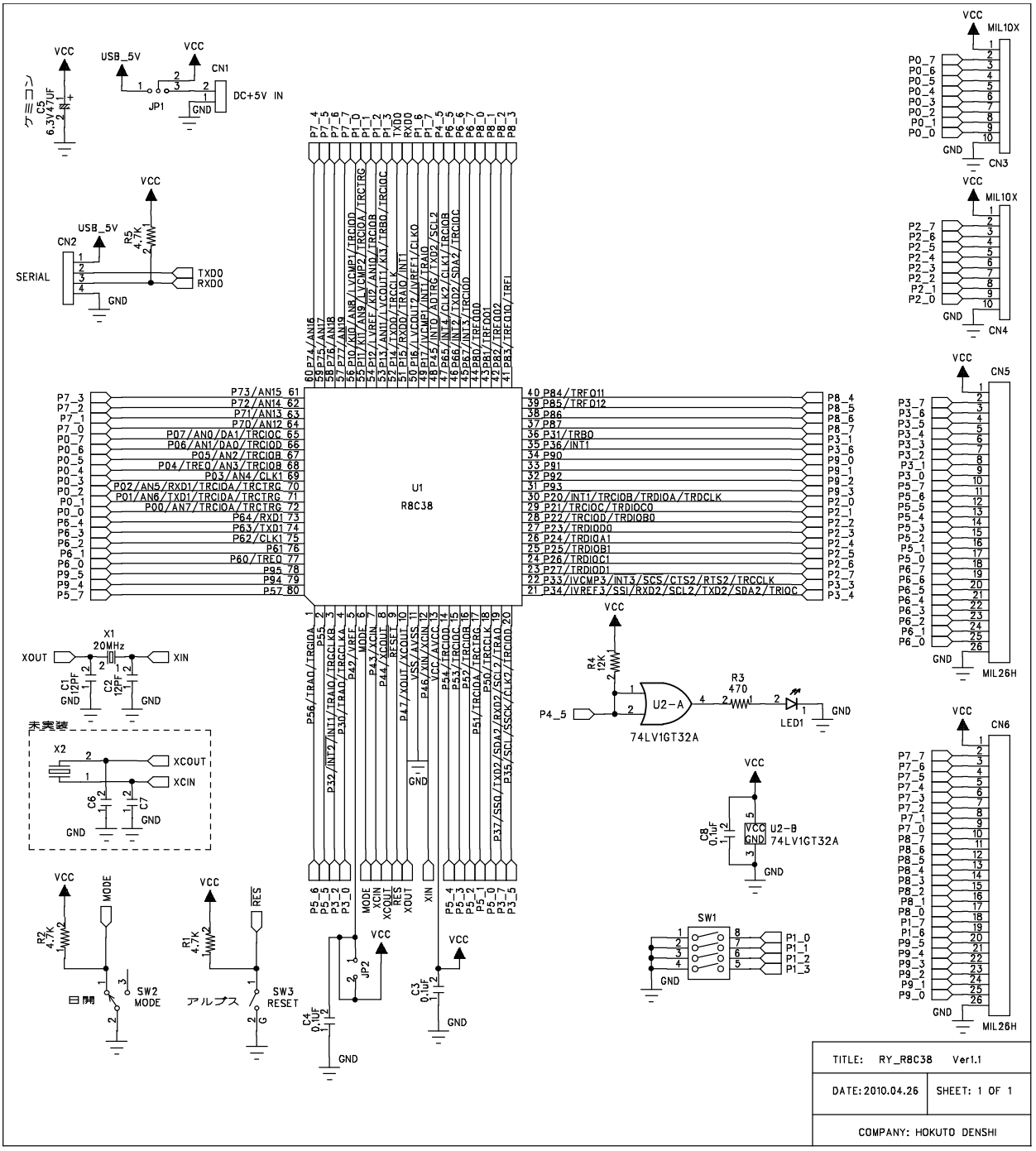
CN5(26ピンボックスコネクタ)

1	+5V
2	P3_7
3	P3_6
4	P3_5
5	P3_4
6	P3_3
7	P3_2
8	P3_1
9	P3_0
10	P5_7
11	P5_6
12	P5_5
13	P5_4
14	P5_3
15	P5_2
16	P5_1
17	P5_0
18	P6_7
19	P6_6
20	P6_5
21	P6_4
22	P6_3
23	P6_2
24	P6_1
25	P6_0
26	GND

CN6(26ピンボックスコネクタ)

1	+5V
2	P7_7
3	P7_6
4	P7_5
5	P7_4
6	P7_3
7	P7_2
8	P7_1
9	P7_0
10	P8_7
11	P8_6
12	P8_5
13	P8_4
14	P8_3
15	P8_2
16	P8_1
17	P8_0
18	P1_7
19	P1_6
20	P9_5
21	P9_4
22	P9_3
23	P9_2
24	P9_1
25	P9_0
26	GND

2.6 回路図



※LED1 はプログラム書き込み時(SW2 が ON のとき)、点灯します。これは、P4_5 端子が入力になっているため R4 のプルアップ抵抗で LED1 のアノード側が 5V になるためです。プログラム実行時は、P4_5 端子の状態によって、LED1 は点灯・消灯します。

※ディップスイッチ(SW1)のプルアップは、マイコン内蔵のプルアップ抵抗を使っています。

※P4_2 端子は、A/D コンバータの基準電圧を入力する端子で+5V に繋がっています。入力ポートとしては使えません。JP2 間のパターンをカットすれば入力端子として使用することができます（ただし、A/D コンバータは使用できなくなります）。

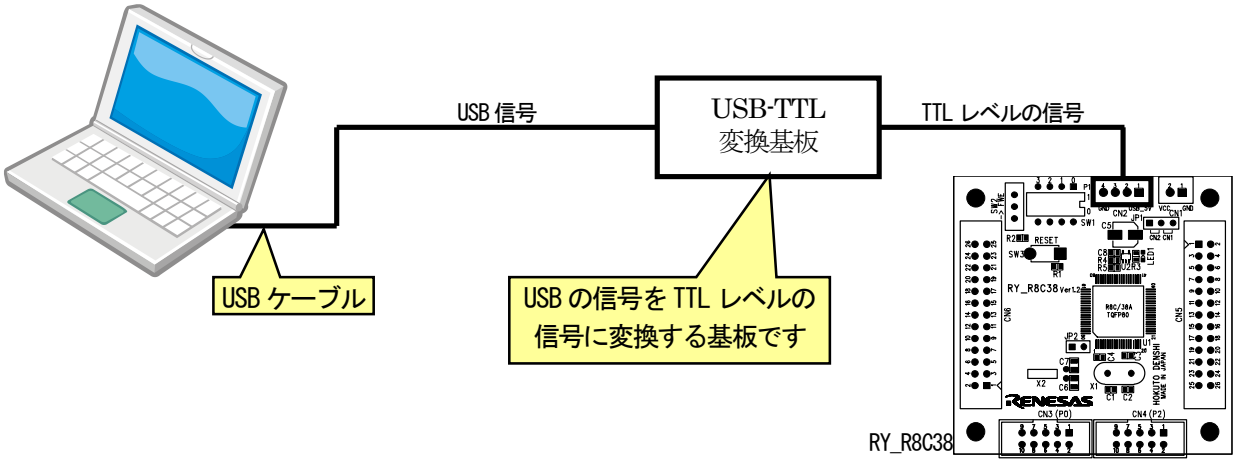
3. 書き込み基板の仕様

3.1 書き込み方法

RY_R8C38 ボードにプログラムを書き込むとき、パソコンと RY_R8C38 ボードの CN2（TTL レベルシリアル信号コネクタ）を接続して信号のやり取りを行います。CN2 の信号は、TTL レベルです。**パソコンからは TTL レベルの信号が出力されていないので、必ず信号変換をする必要があります。**
主な変換方法を下記に示します。

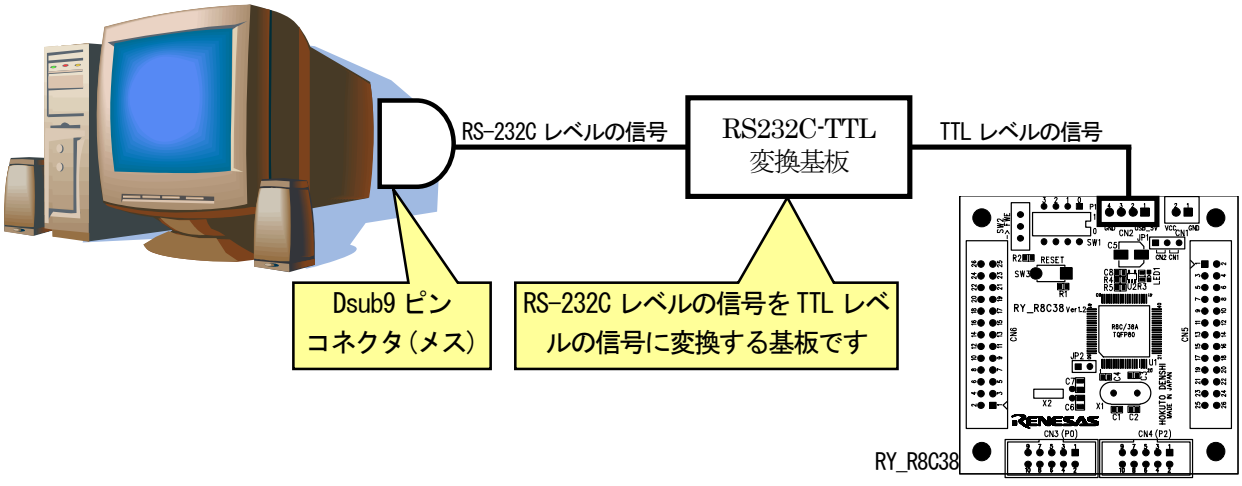
(1) パソコンに USB がある場合

パソコンからの USB 信号を TTL レベルのシリアル信号に変換する基板を介して、RY_R8C38 ボードにプログラムを書き込みます。パソコンからは COM ポートと認識されるので、Tera Term などの通信ソフトを使ったパソコン-マイコン間通信も可能です。USB-TTL 変換基板は、RY-WRITER 基板などが市販されてます。また、USB-TTL 変換 IC 単体でも売られていますので自作も可能です。



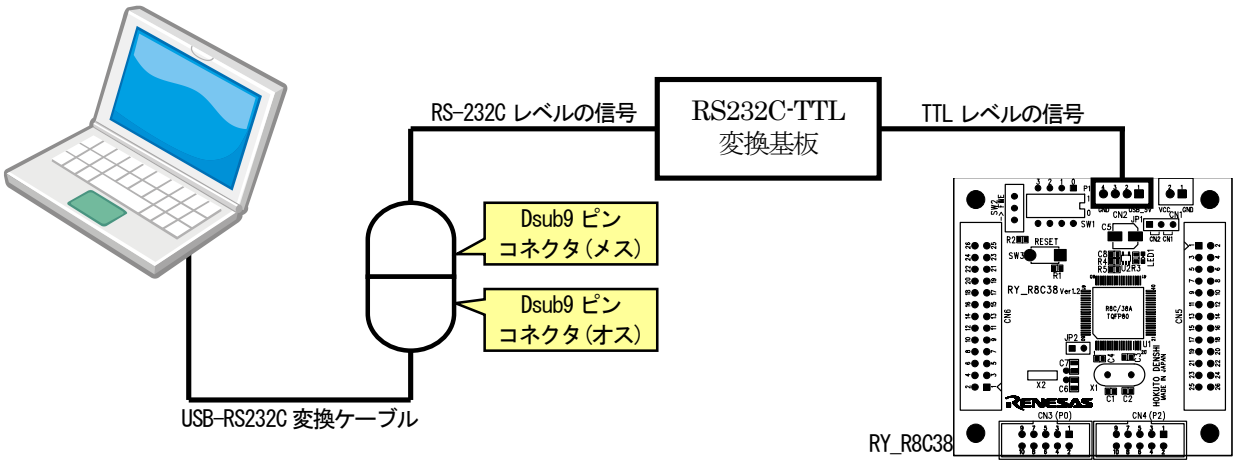
(2) パソコンに RS-232C コネクタがある場合

パソコンからの RS-232C 信号を TTL レベルのシリアル信号に変換する基板を介して、RY_R8C38 ボードにプログラムを書き込みます。RS232C-TTL 変換基板は、市販されています。また、RS-232C－TTL 変換 IC 単体でも売られていますので自作も可能です。



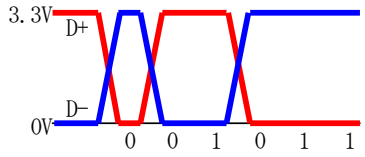
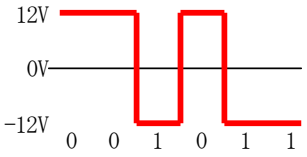
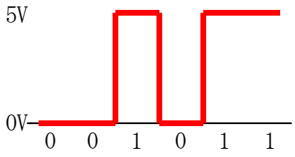
(3) USB-RS232C 変換ケーブルがある場合

RY3048Fone ボードは、パソコンとのやり取りを RS-232C 信号レベルで行います。パソコンに RS-232C 規格の Dsub9 ピンコネクタが無い場合は、USB-RS232C 変換ケーブルを用意して信号変換を行います。この変換ケーブルがある場合は、RS232C-TTL 変換基板を使って RY_R8C38 ボードにプログラムを書き込むことができます。



※各信号のレベル

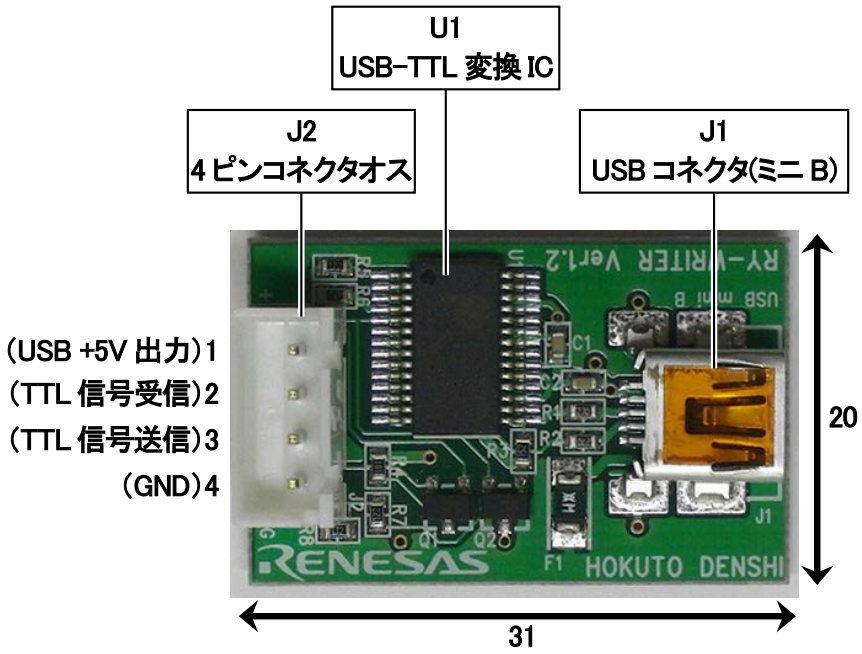
各信号のレベルを下記に示します。各信号には互換がないので、それぞれの信号を直結することはできません。これらの信号同士を変換する回路を介して信号のやり取りを行う必要があります。

USB	RS-232C	TTL
USB 信号は、3.3V の差動信号を半二重で送受信して通信を行います。差動とは、2 本の通信線を使い（USB は D+、D-という通信線）、信号の電圧の差を検出して"1"か"0"かを判断する方式です。 通信方式は、NRZI 方式を採用しており、"0"は信号を反転、"1"は信号を保持します。	RS-232C は、"1"は-3V～-15V、"0"は+3V～+15V の電圧で信号のやり取りを行う方式です。	TTL は、"1"は 5V、"0"は 0V の電圧で信号のやり取りを行う方式です。 ※電源電圧は、5V とします。
		

3.2 USB-TTL 変換基板(RY-WRITER 基板)

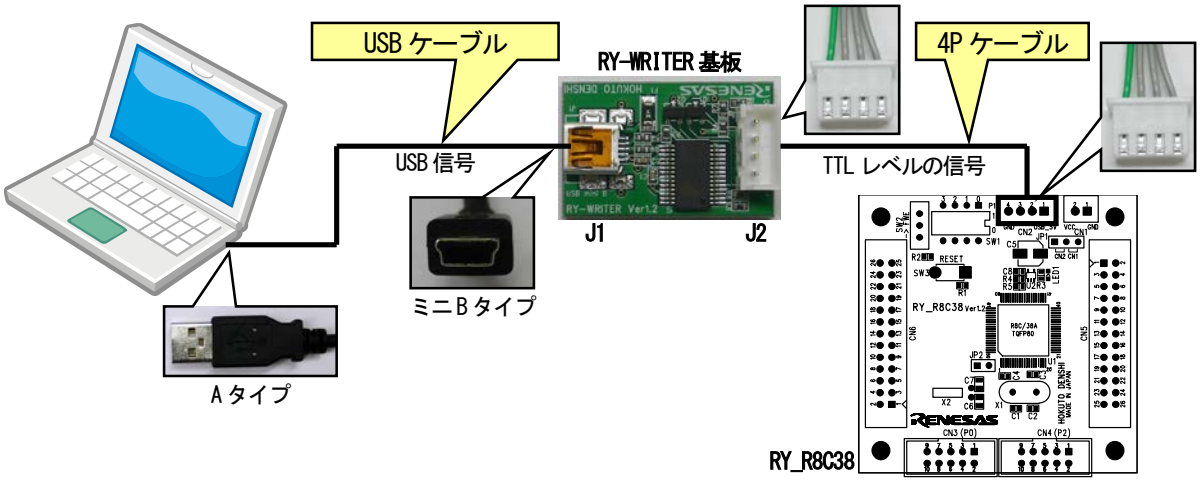
RY-WRITER 基板は、USB 信号を TTL レベルの信号、またはその逆に変換することのできる基板です。この基板を初めてパソコン側に接続すると新しいデバイスとして認識されます。ドライバーをインストールして、COM ポートとして認識させます。ドライバーのインストール方法は後述します。

3.2.1 外観



部品番号	説明
J1	USB コネクタ(ミニ B)です。このコネクタは、パソコンと USB ケーブル（A:ミニ B タイプ)を使って接続します。
J2	4 ピンコネクタです。このコネクタと RY_R8C38 ボードの CN2 を接続します。
U1	プロリフィックテクノロジー(Prolific Technology Inc)社の PL2303 という IC で、USB 信号を TTL レベルの信号に変換します。違うメーカからも同様の IC が販売されています。

3.2.2　接続図

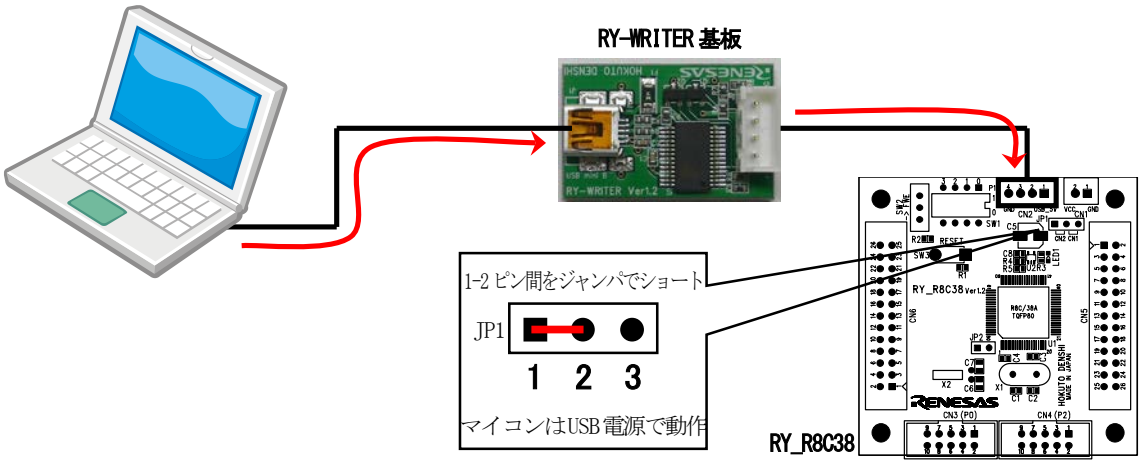


ケーブル	説明
USB ケーブル	コネクタの形状が、「A⇔ミニ B」タイプの USB ケーブルを用意します。
4P ケーブル	4 線のケーブルです。RY-WRITER 基板には未完成のケーブルが付属しています。完成させて接続します。

3.2.3　電源の流れ

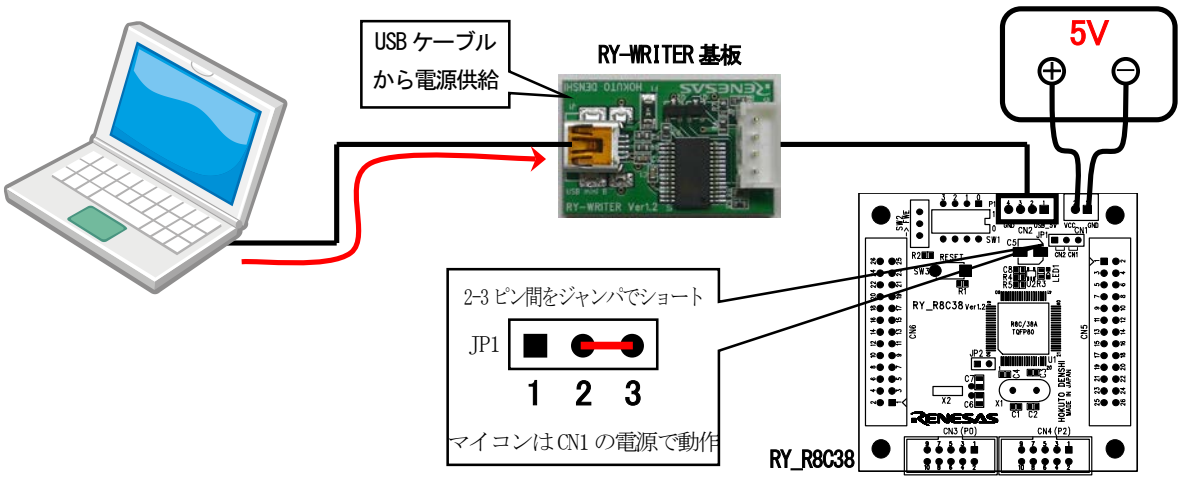
(1) USB から電源供給する場合

RY_R8C38 ボードの電源は、RY_R8C38 ボードの CN2 コネクタから供給します。この電源は、パソコンの USB ケーブルを通して供給されます。RY_R8C38 ボードを動作させるために、電源を用意する必要がないので便利ですが、USB 電源は 500mA 以上（または接続先の規定以上）の電流は流せません。それ以上の電流が流れないよう、コネクタの接続など気をつけてください。

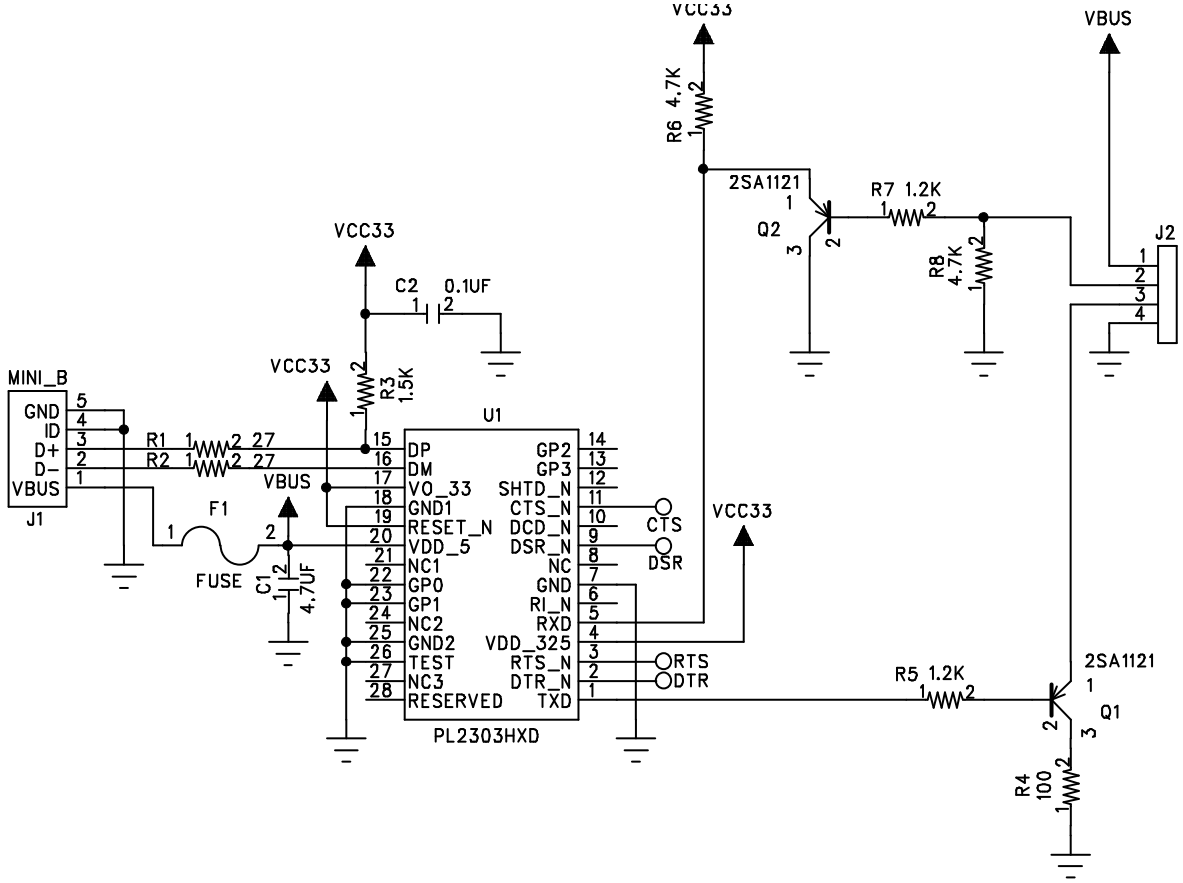


(2) CN1 から電源供給する場合

RY_R8C38 ボードの電源は、RY_R8C38 ボードの CN1 コネクタから供給されます。RY-WRITER 基板の電源は、USB ケーブルから供給します。



3.2.4　回路図

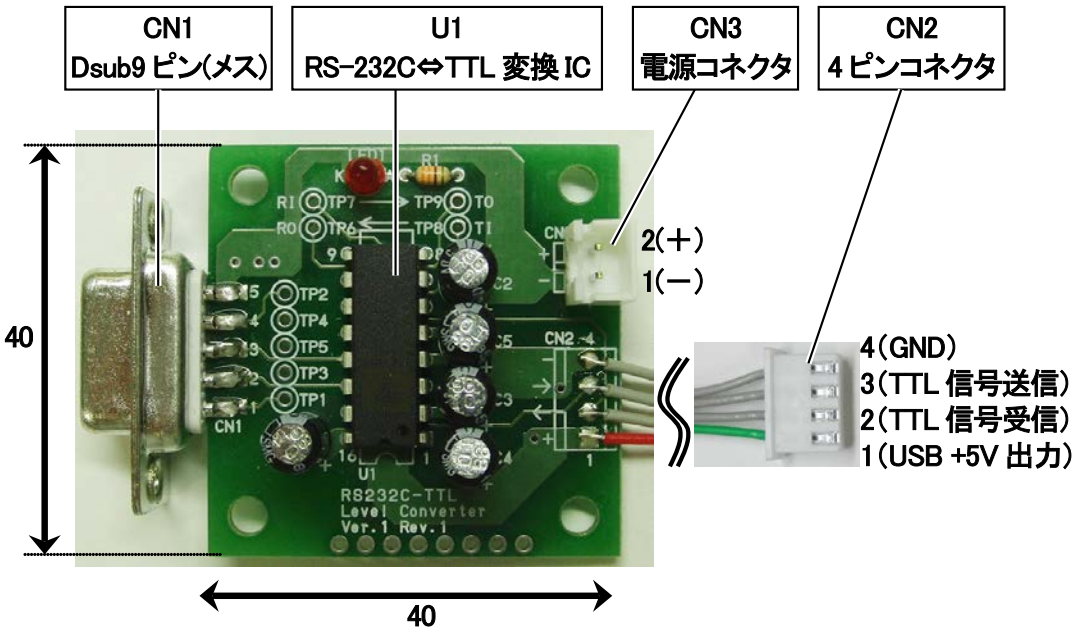


※F1 はリセッタブルヒューズで、USB 電源の過電流を遮断する素子です。過電流が流れなくなると復帰します。
※Ver.1.2 基板は、U1 の 2 ピン、3 ピン、9 ピン、11 ピンにつながっているランドが半田面にあります。

3.3 RS232C-TTL レベル変換基板

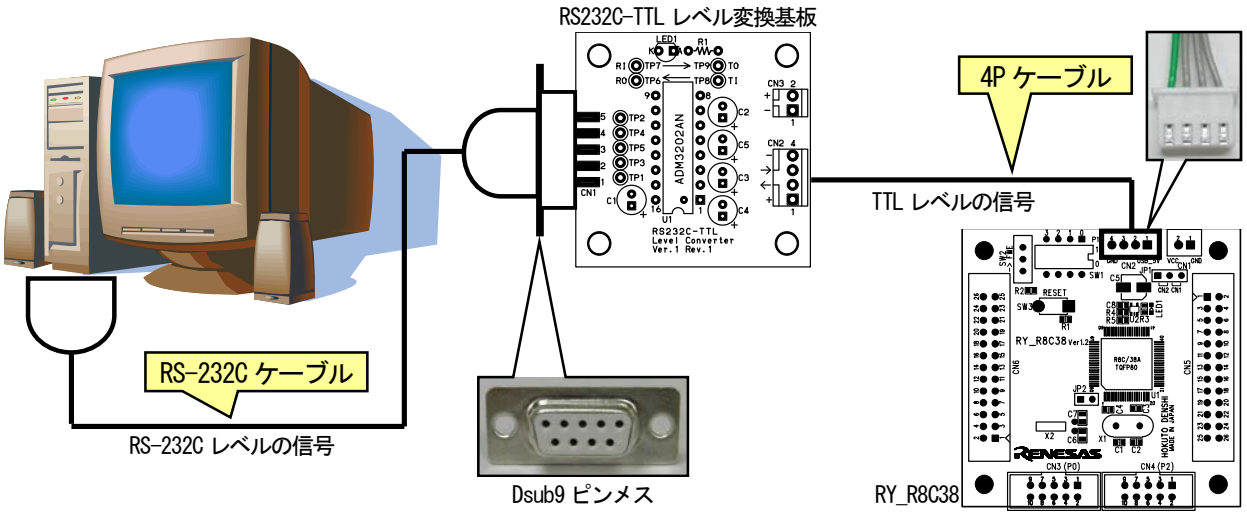
RS232C-TTL レベル変換基板は、RS-232C レベルの信号を TTL レベルの信号に、またはその逆に変換することのできる基板です。

3.3.1 外観



部品番号	説明
CN1	Dsub9 ピンコネクタ(メス)です。パソコン側の Dsub9 ピンコネクタ(オス)と接続します。パソコンには直接接続するか、ストレートケーブルを介して接続します。
CN2	4 ピンコネクタです。このコネクタと RY_R8C38 ボードの CN2 を接続します。
CN3	2 ピンコネクタです。RS232C-TTL 変換基板に電源を供給します。
U1	ADM3202AN という IC で、RS-232C 信号を TTL レベルの信号に変換しています。MAX232 と互換の IC です。

3.3.2　接続図

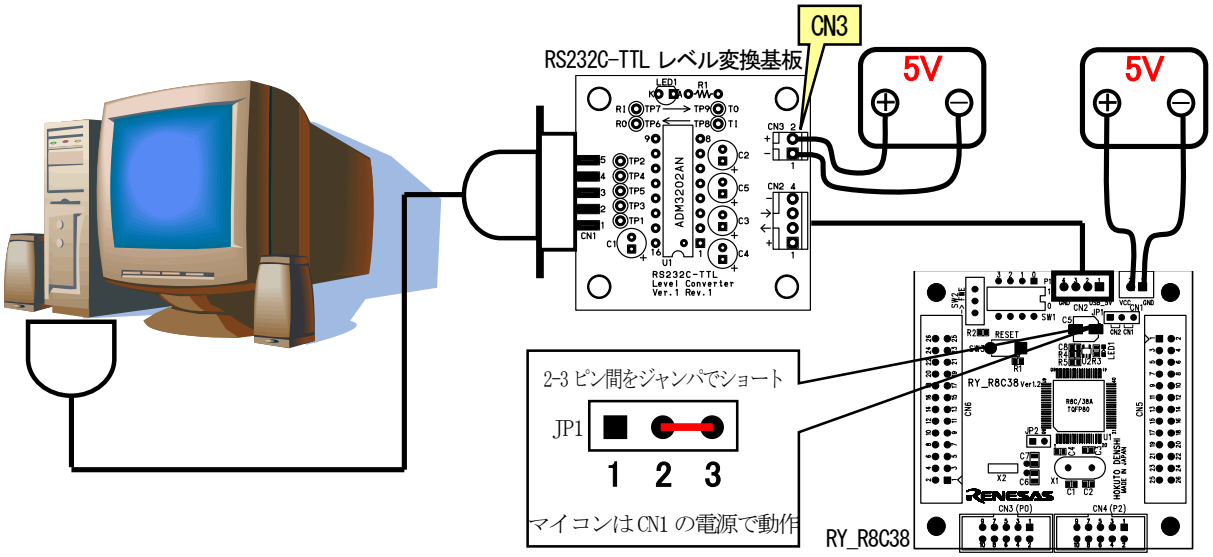


ケーブル	説明
RS-232C ケーブル	コネクタの形状が、「Dsub9 ピンオス⇔Dsub9 ピンメス」タイプのストレートケーブルを用意します。RS232C-TTL 変換基板の Dsub9 ピンメス(CN1)を、パソコンの Dsub9 ピンコネクタ(オス)に直接接続しても構いません。
4P ケーブル	4 線のケーブルです。RS232C-TTL 変換基板に未完成のケーブルが付属しています。完成させて接続します。

3.3.3　電源の流れ

(1) RS232C-TTL レベル変換基板の電源供給を別に用意する場合

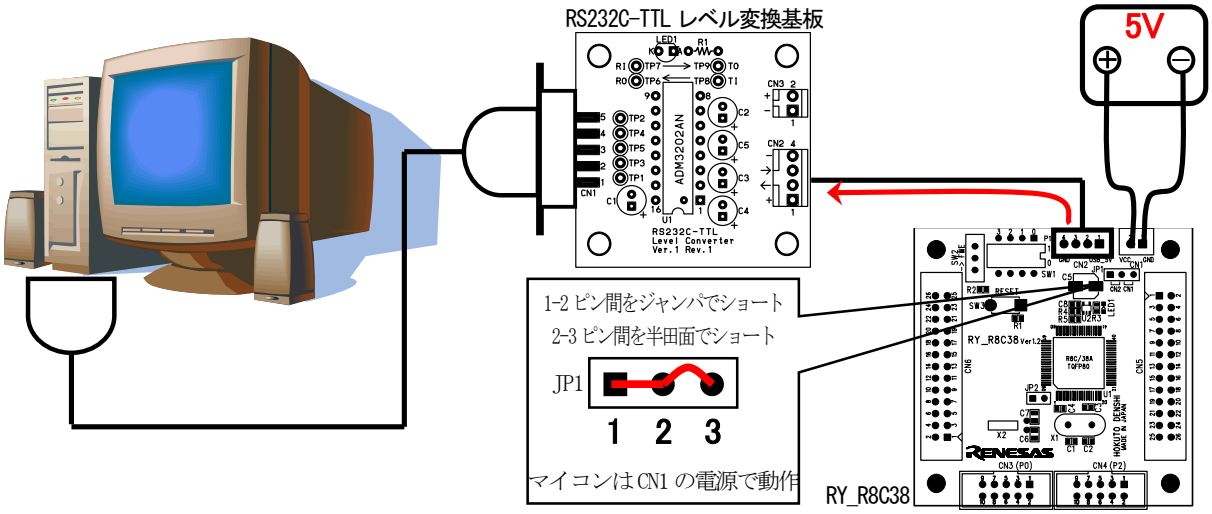
RY_R8C38 ボードの電源は、RY_R8C38 ボードの CN1 コネクタから供給します。RS232C-TTL レベル変換基板の電源は、RS232C-TTL レベル変換基板の CN3 から供給してください。RY-WRITER 基板のように、パソコン側から電源の供給はされません。



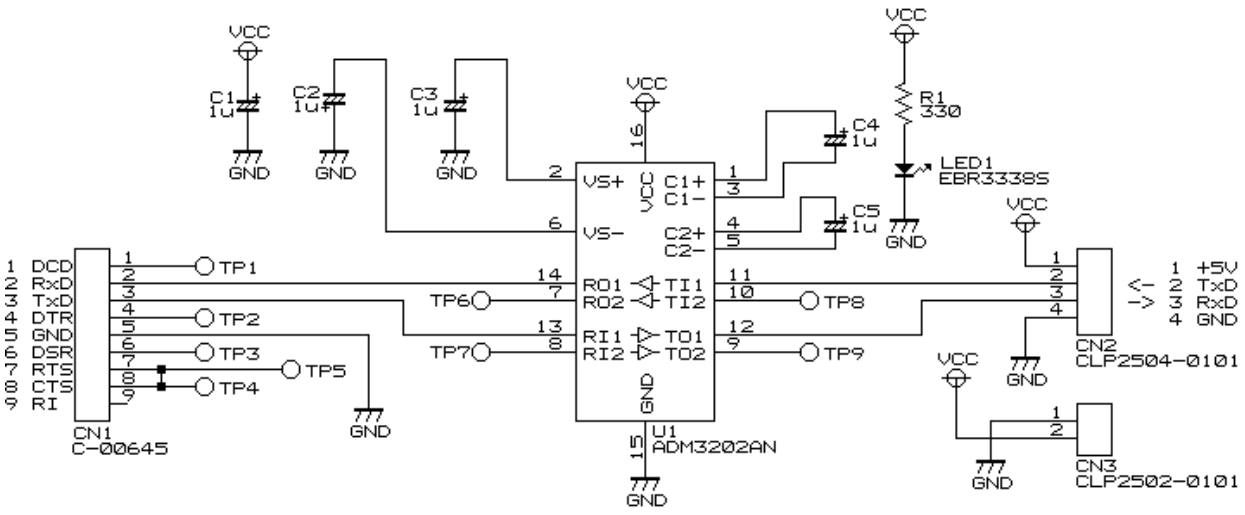
(2) RS232C-TTL レベル変換基板の電源供給を RY_R8C38 ボードから行う場合

RY_R8C38 ボードの JP1 は、1-2 ピン間をジャンパでショートさせ、2-3 ピン間は半田面でショートするように半田付けしておきます。

RY_R8C38 ボードの電源は、RY_R8C38 ボードの CN1 コネクタから供給します。RS232C-TTL レベル変換基板の電源は、RY_R8C38 ボードの CN2 から供給されます。供給したくない場合は、JP1 のジャンパを抜いてください。



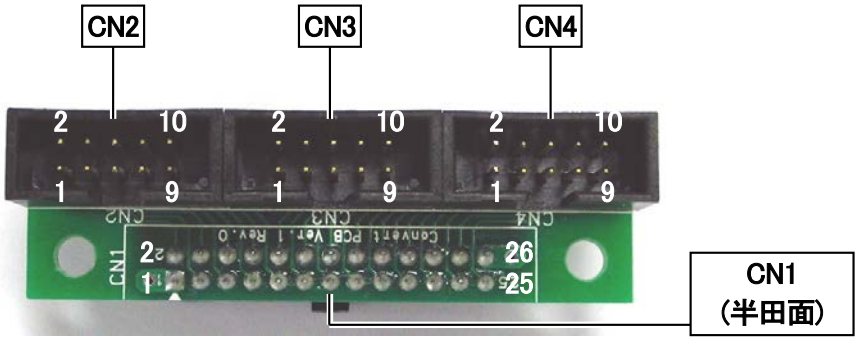
3.3.4　回路図



4. RY_R8C38 ボードコネクタ変換基板

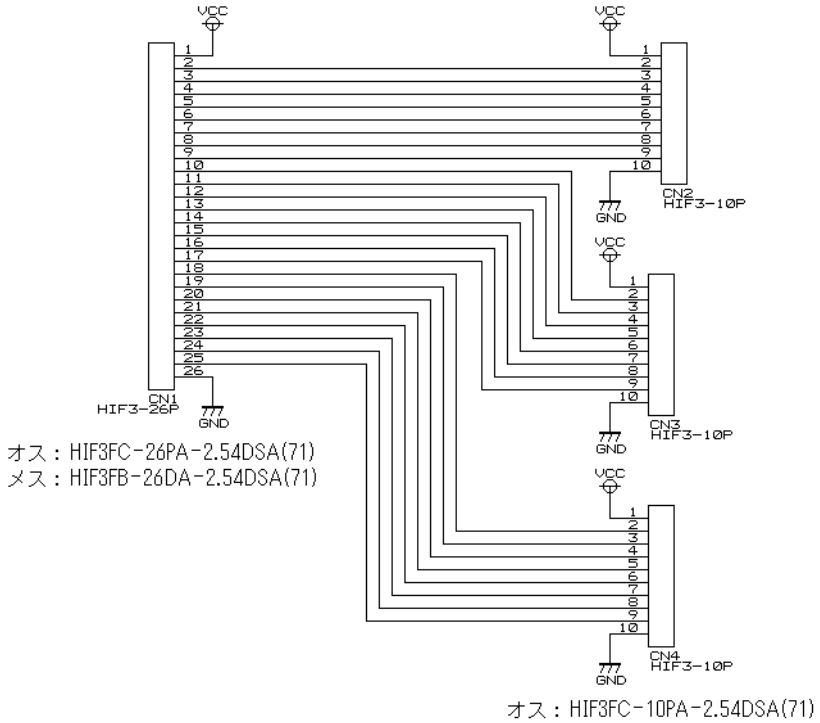
RY_R8C38 ボードの 26 ピンコネクタは、既存の 10 ピンコネクタと互換性がありません。そこで 26 ピンコネクタを 10 ピンコネクタに変換する基板があります。この基板について説明します。

4.1 外観



部品番号	説明
CN1	26 ピンコネクタ（メス）です。半田面から実装しています。RY_R8C38 ボードの CN5、CN6 に接続します。
CN2	10 ピンコネクタ（オス）です。CN1 と繋がっています。
CN3	10 ピンコネクタ（オス）です。CN1 と繋がっています。
CN4	10 ピンコネクタ（オス）です。CN1 と繋がっています。

4.2 回路図



4.3 ポートの変換表

26 ピンコネクタ 1 個を 10 ピンコネクタ 3 個に変換します。
既にある 10 ピンコネクタの基板と接続することができます。

RY_R8C38の CN5(26ピンボックスコネクタ)

1	+5V
2	P3_7
3	P3_6
4	P3_5
5	P3_4
6	P3_3
7	P3_2
8	P3_1
9	P3_0
10	P5_7
11	P5_6
12	P5_5
13	P5_4
14	P5_3
15	P5_2
16	P5_1
17	P5_0
18	P6_7
19	P6_6
20	P6_5
21	P6_4
22	P6_3
23	P6_2
24	P6_1
25	P6_0
26	GND

変換基板のCN2

1	+5V
2	P3_7
3	P3_6
4	P3_5
5	P3_4
6	P3_3
7	P3_2
8	P3_1
9	P3_0
10	GND

変換基板のCN3

1	+5V
2	P5_7
3	P5_6
4	P5_5
5	P5_4
6	P5_3
7	P5_2
8	P5_1
9	P5_0
10	GND

変換基板のCN4

1	+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

RY_R8C38の CN6(26ピンボックスコネクタ)

1	+5V
2	P7_7
3	P7_6
4	P7_5
5	P7_4
6	P7_3
7	P7_2
8	P7_1
9	P7_0
10	P8_7
11	P8_6
12	P8_5
13	P8_4
14	P8_3
15	P8_2
16	P8_1
17	P8_0
18	P1_7
19	P1_6
20	P9_5
21	P9_4
22	P9_3
23	P9_2
24	P9_1
25	P9_0
26	GND

変換基板のCN2

1	+5V
2	P7_7
3	P7_6
4	P7_5
5	P7_4
6	P7_3
7	P7_2
8	P7_1
9	P7_0
10	GND

変換基板のCN3

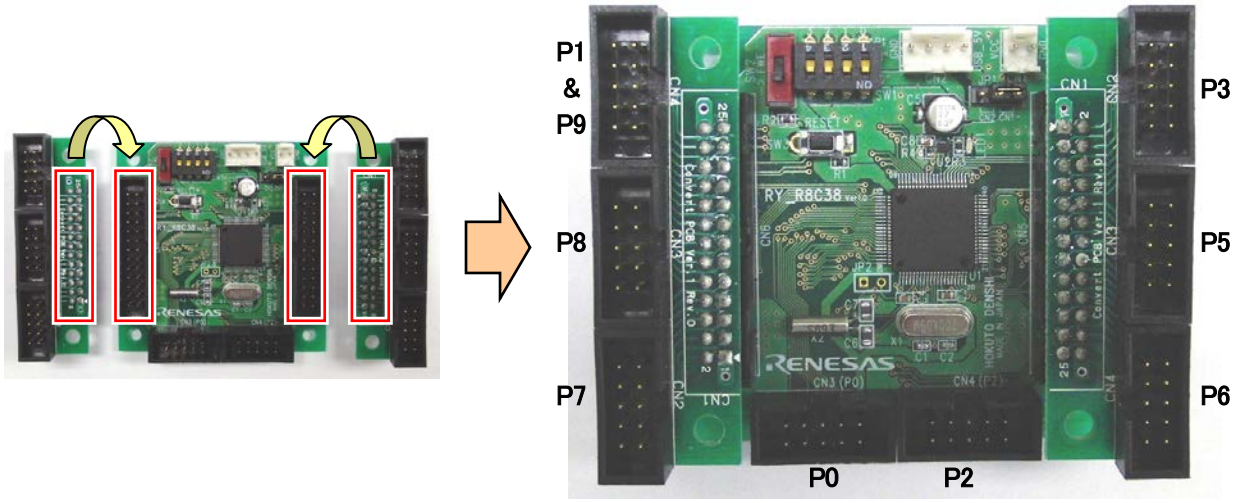
1	+5V
2	P8_7
3	P8_6
4	P8_5
5	P8_4
6	P8_3
7	P8_2
8	P8_1
9	P8_0
10	GND

変換基板のCN4

1	+5V
2	P1_7
3	P1_6
4	P9_5
5	P9_4
6	P9_3
7	P9_2
8	P9_1
9	P9_0
10	GND

4.4 RY_R8C38 ボードへ実装

RY_R8C38 ボードにコネクタ変換基板を実装したときの 10 ピンコネクタのポート番号を下図に示します。



5. 実習基板 Ver.2

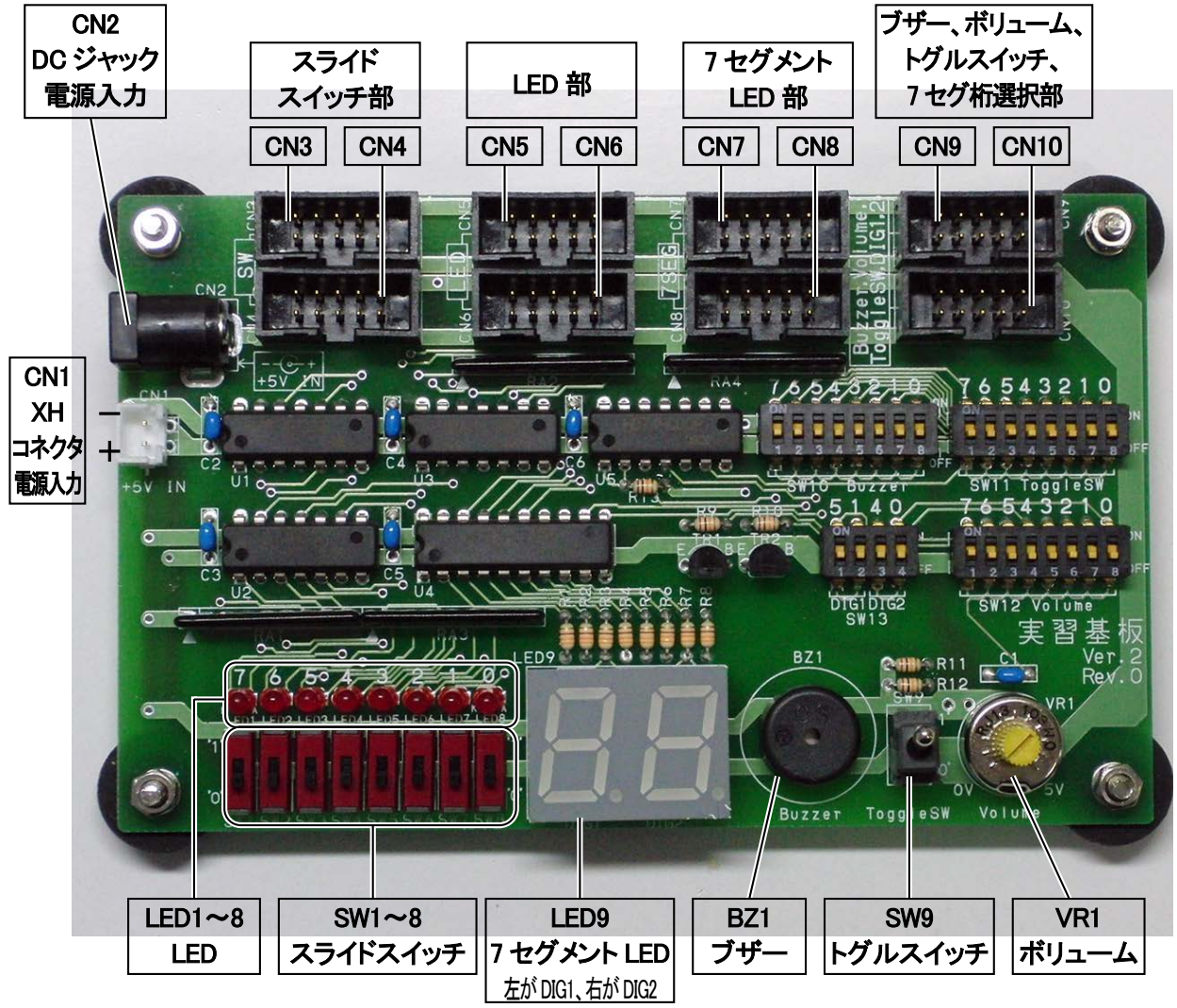
本書では、R8C/38A マイコンの実習を行うために、実習基板 Ver.2 を使用します。実習基板 Ver.2 について説明します。

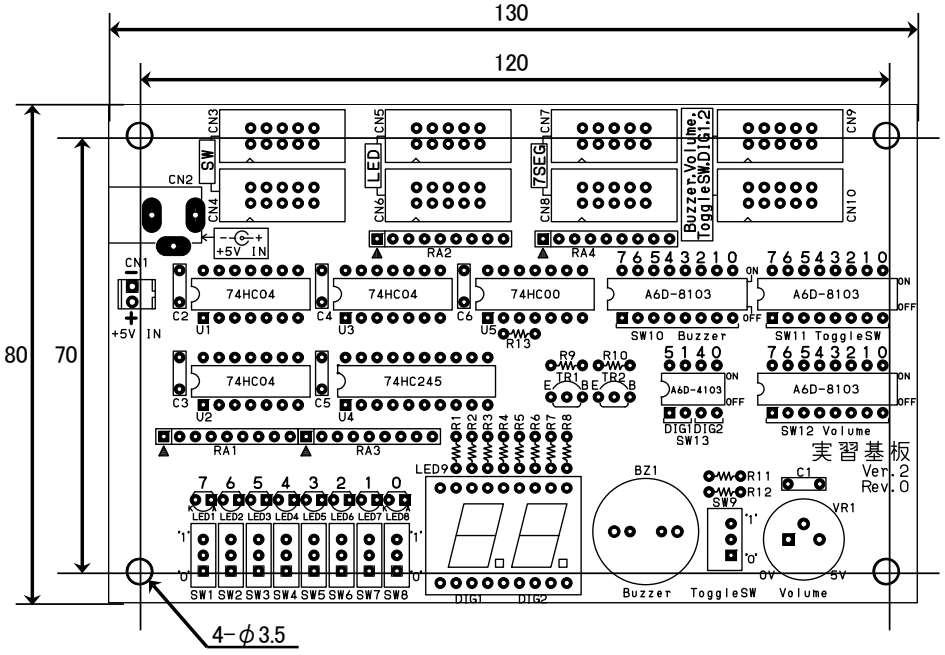
5.1 仕様

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

5.2　外観

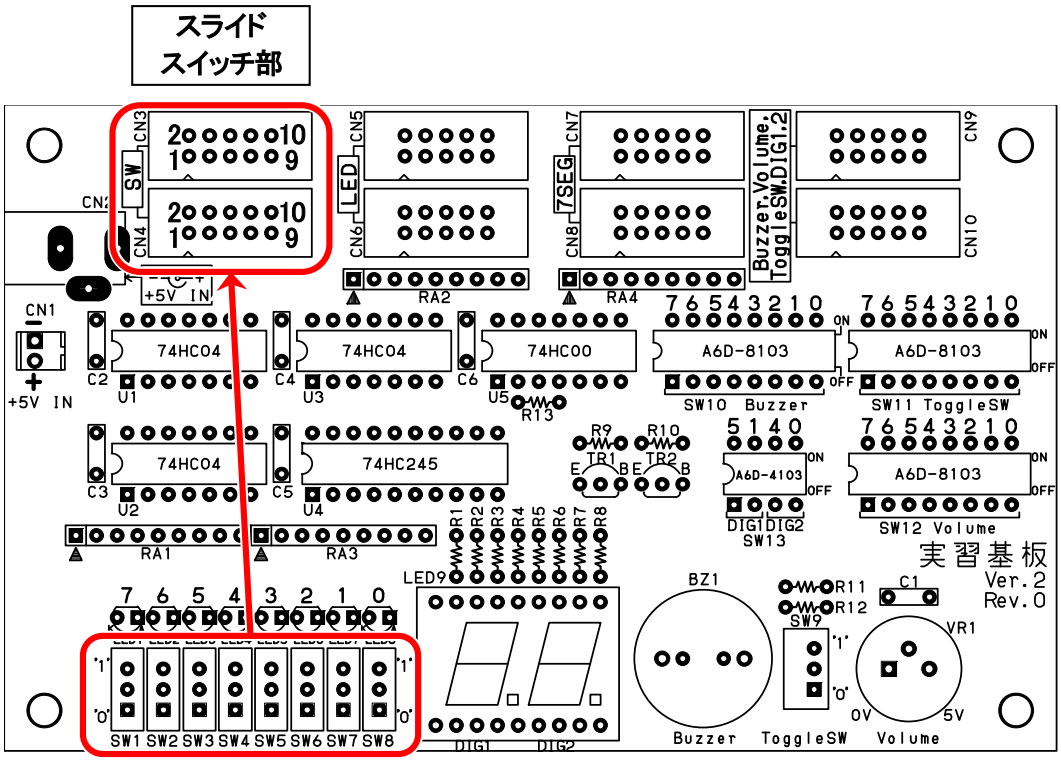

CN2
DC ジャック
電源入力
スライド
スイッチ部
LED 部
7 セグメント
LED 部
ブザー、ボリューム、
トグルスイッチ、
7 セグ桁選択部
CN3
CN4
CN5
CN6
CN7
CN8
CN9
CN10
CN1
XH
コネクタ
電源入力
－
＋
LED1～8
LED
SW1～8
スライドスイッチ
LED9
7 セグメント LED
左が DIG1、右が DIG2
BZ1
ブザー
SW9
トグルスイッチ
VR1
ボリューム
実習基板
Ver.2
Rev.0


130
120
80
70
4-φ3.5

5.3　スライドスイッチ部

5.3.1　コネクタ

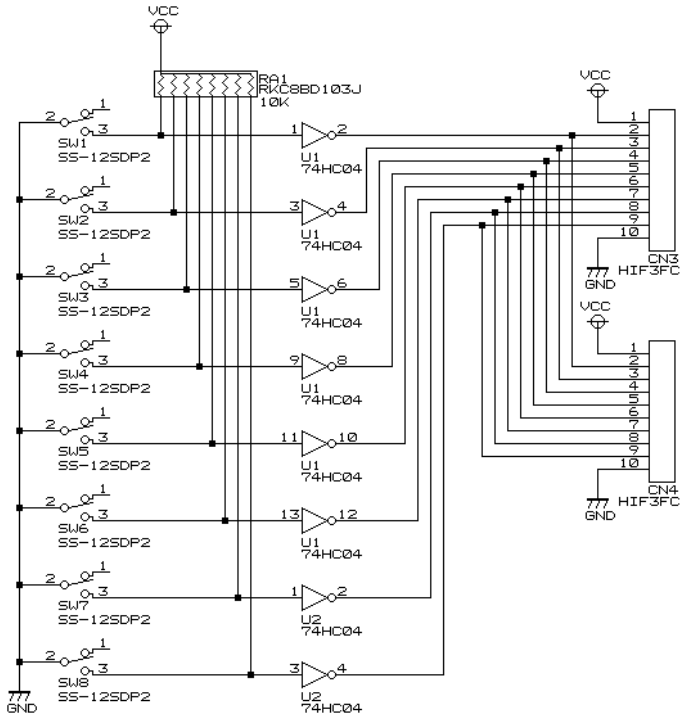
スライドスイッチ SW1～SW8 の 8bit 分の信号を CN3(CN4)のコネクタから外部に出力します。CN3 と CN4 は並列に接続されています。



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		―	―

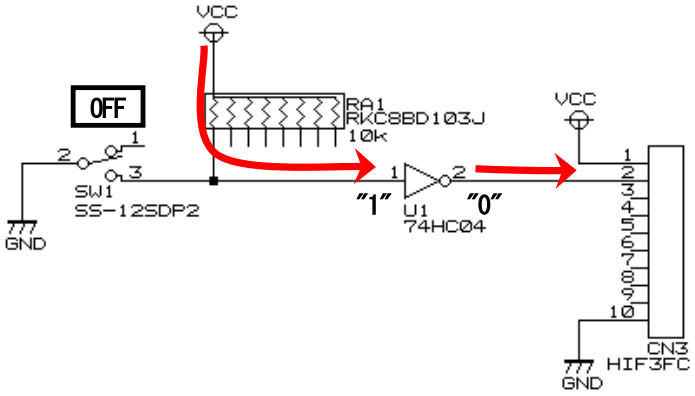
5.3.2　回路図



5.3.3　動作原理

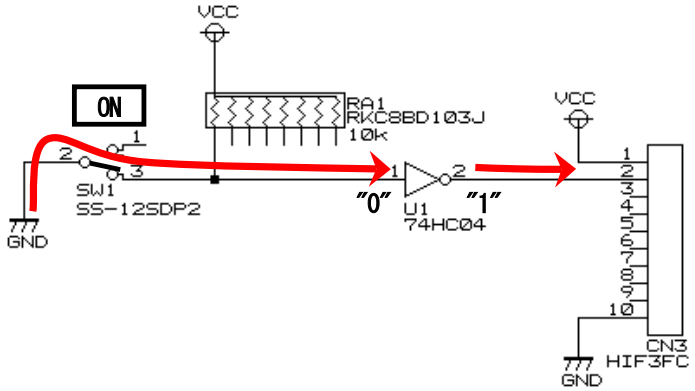
■スライドスイッチが"0"側（下側）

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）

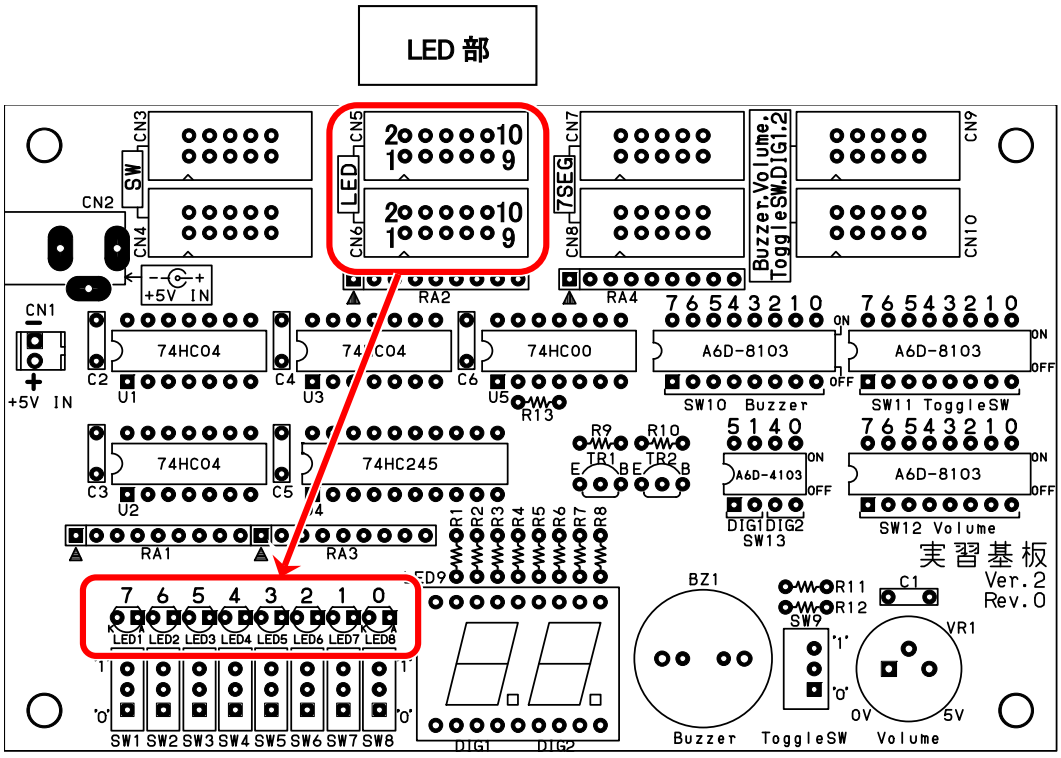
スイッチが上側の場合は ON になり、スライドスイッチ、74HC04(NOT)回路を通して、端子から"1"が出力されます。



5.4 LED 部

5.4.1 コネクタ

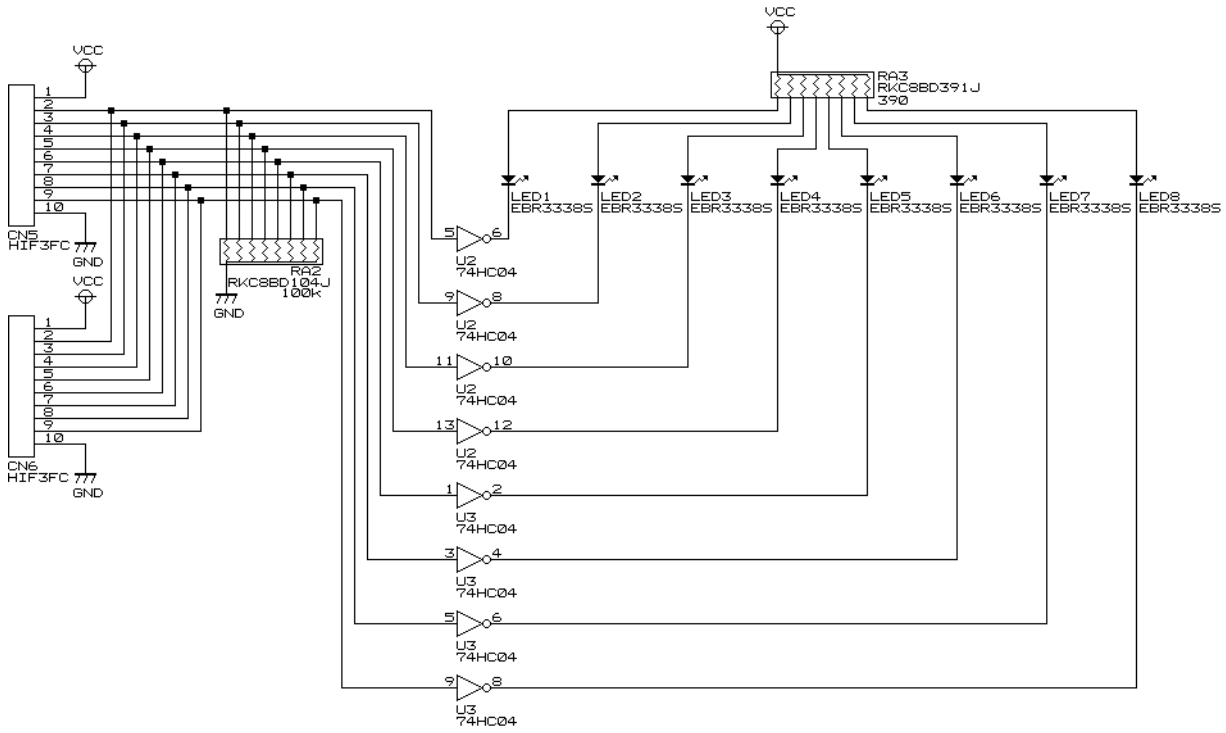
CN5(CN6)のコネクタから入力した信号を、LED1～LED8 へ出力します。CN5 と CN6 は並列に接続されています。



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		―	―

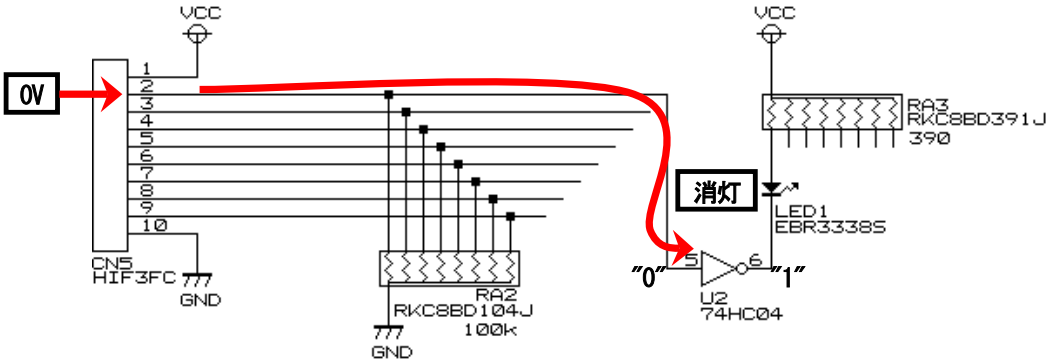
5.4.2　回路図



5.4.3　動作原理

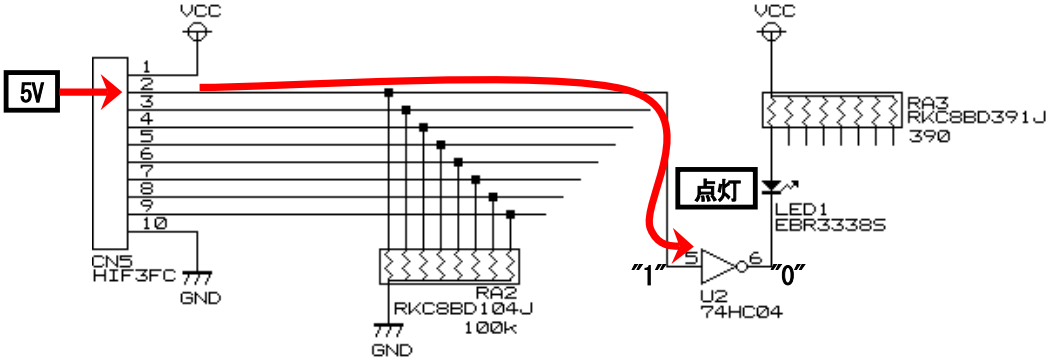
■入力信号が 0V(″0″)の場合

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 は消灯します。

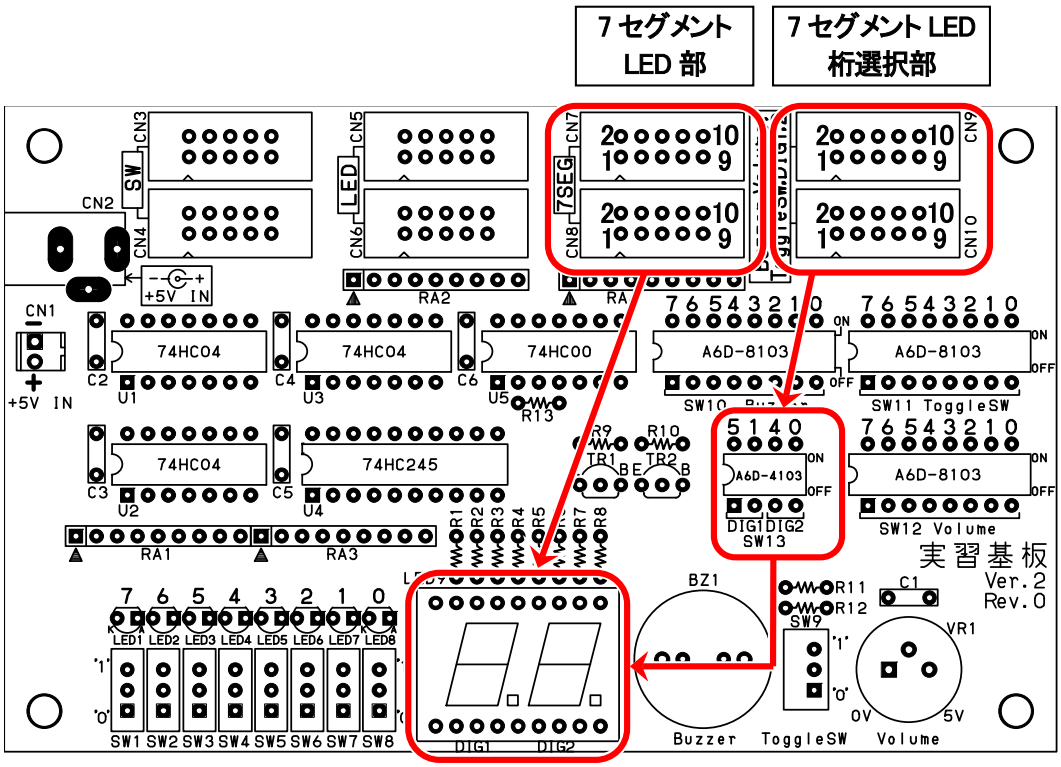


5.5　7 セグメント LED 部

5.5.1　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。

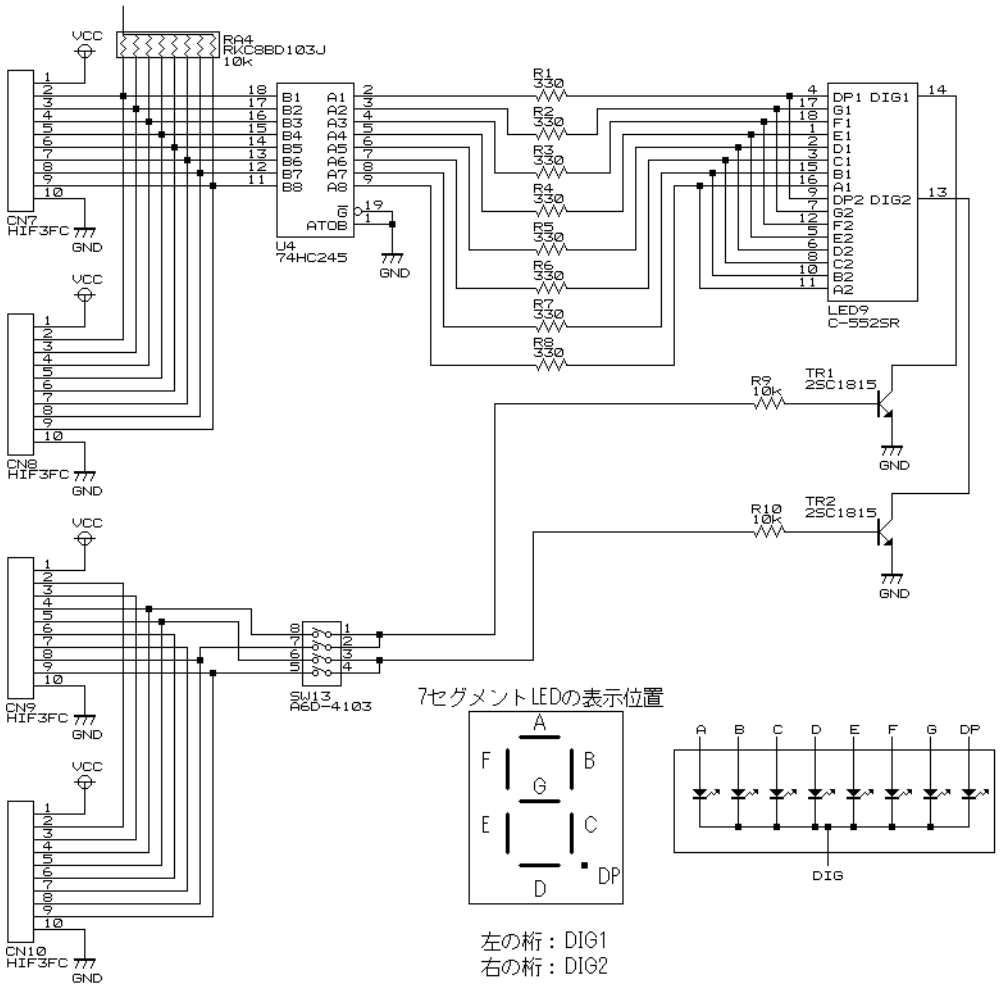
左桁を点灯させるか、右桁を点灯させるかは、CN9(CN10)からの入力信号で決めます。どの bit にするかは、SW13 で決めます。CN9(CN10)と SW13 の関係を下図に示します。

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。

5.5.2 回路図



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

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。

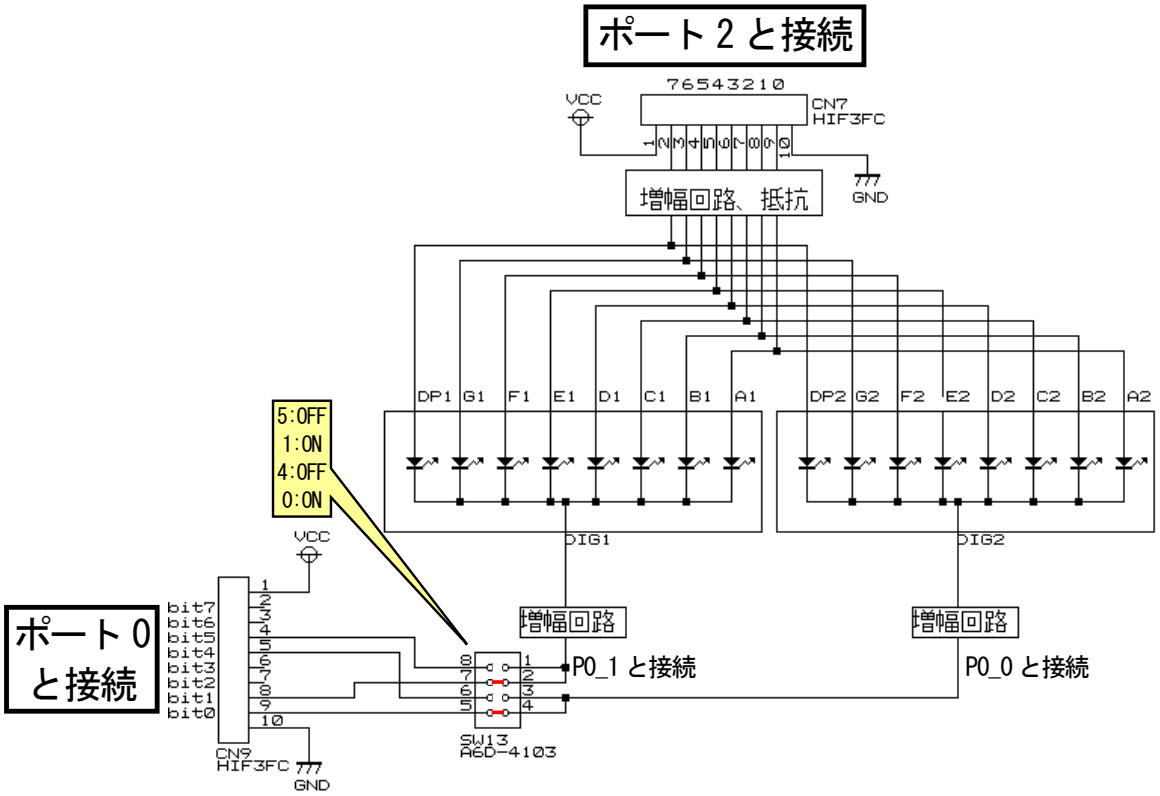
5.5.3 動作原理

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。

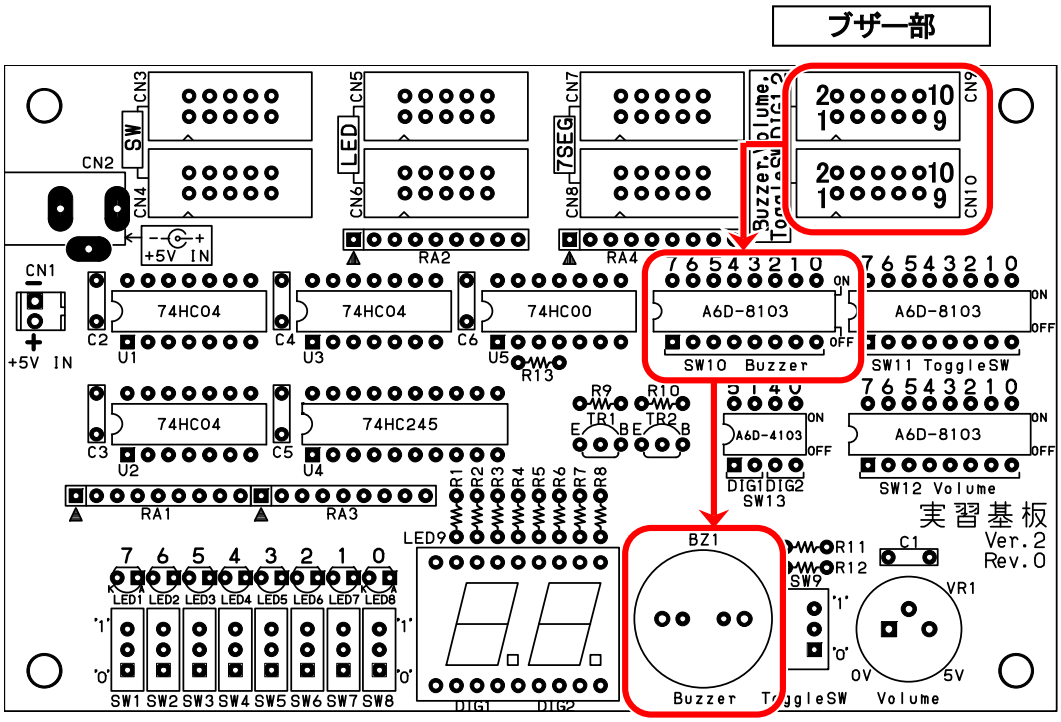
①	左桁、右桁は表示させません。 ポート 0 の bit1 と bit0 を"0"にします。
②	左桁に表示したい値を設定します。今回は「8」です。 ポート 2 に 0x7f を設定します。
③	左桁を表示します。 DIG1 端子に接続されているポート 0 の bit1 を"1"にします。A1～DP1 の LED のカソード側 DIG1 が GND につながり、アノード側が 5V の LED は点灯します。DIG2 は開放状態なので、A2～DP2 は点灯しません。
④	少し待ちます。
⑤	左桁、右桁は表示させません。 ポート 0 の bit1 と bit0 を"0"にします。
⑥	右桁に表示し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

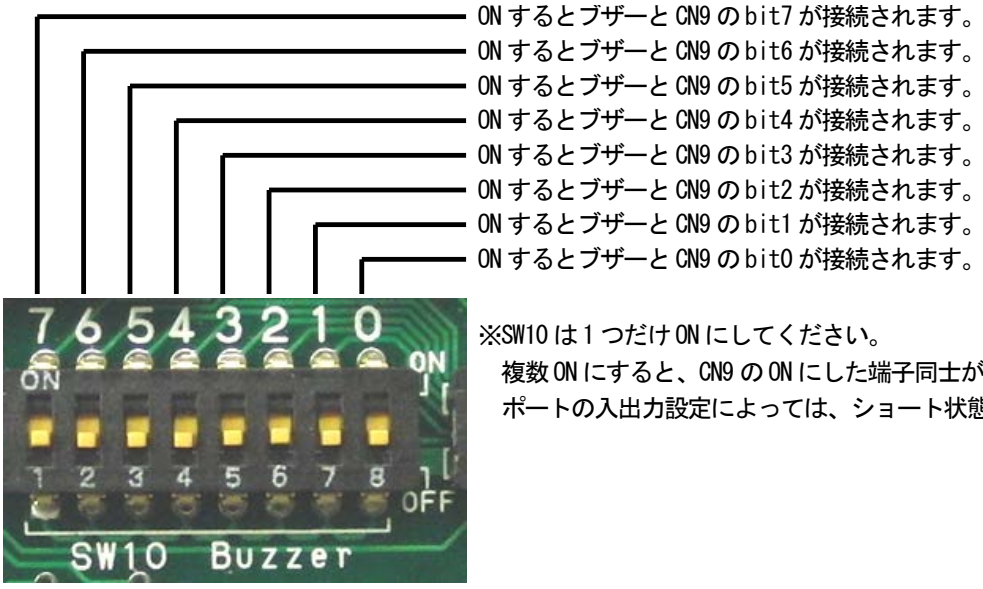
5.6 ブザー部

5.6.1 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。

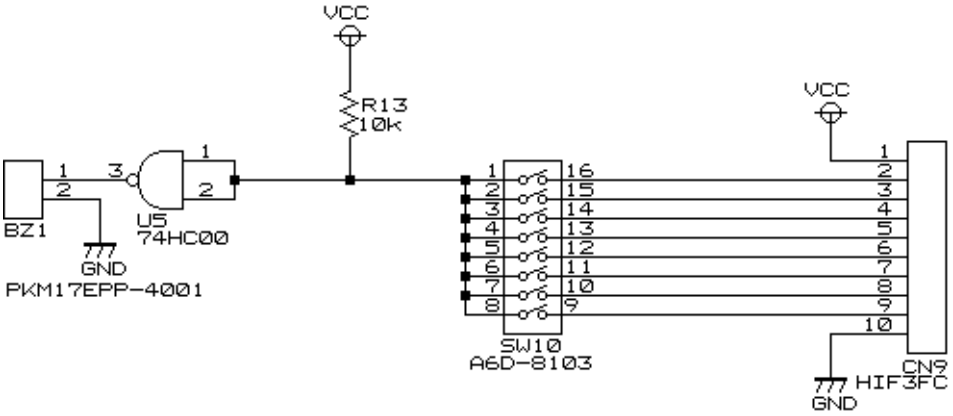


CN9(CN10)とブザーを、どの bit に接続するかは SW10 で決めます。CN9(CN10)と SW10 の関係を下図に示します。



※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。

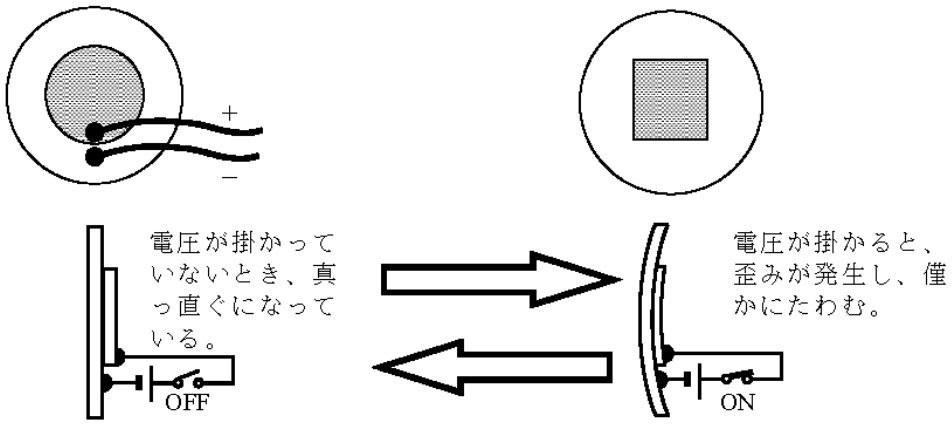
5.6.2 回路図



5.6.3 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。

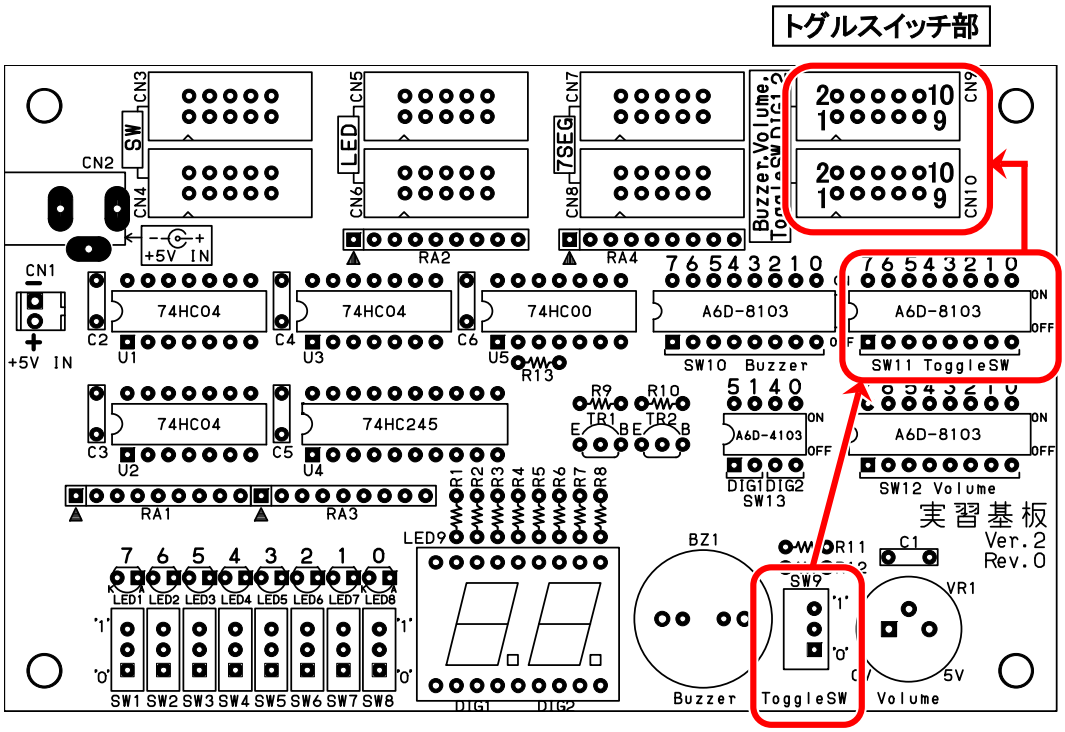
この 2 枚の電極に電圧をかけると、歪みが発生して僅かにたわみます。電圧がかかっていなければ、まっすぐの状態です。これを高速に繰り返すことにより、音波が発生して音が鳴ります。



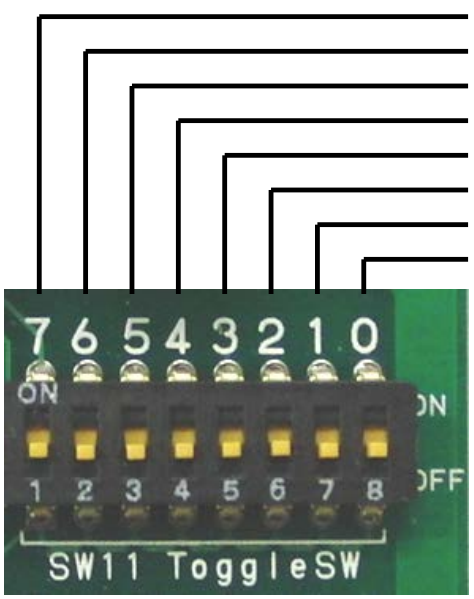
5.7 トグルスイッチ部

5.7.1 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。



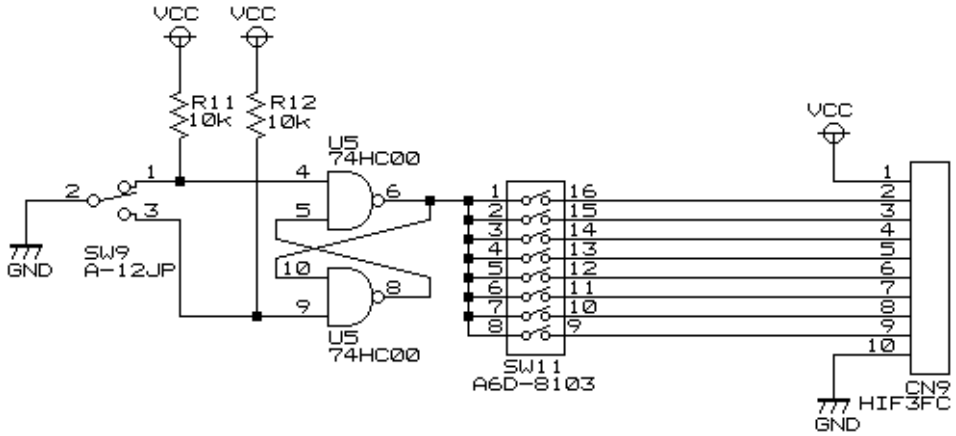
CN9(CN10)とトグルスイッチを、どの bit に接続するかは SW11 で決めます。CN9(CN10)と SW11 の関係を下図に示します。



ON するとトグルスイッチと CN9 の bit7 が接続されます。
ON するとトグルスイッチと CN9 の bit6 が接続されます。
ON するとトグルスイッチと CN9 の bit5 が接続されます。
ON するとトグルスイッチと CN9 の bit4 が接続されます。
ON するとトグルスイッチと CN9 の bit3 が接続されます。
ON するとトグルスイッチと CN9 の bit2 が接続されます。
ON するとトグルスイッチと CN9 の bit1 が接続されます。
ON するとトグルスイッチと CN9 の bit0 が接続されます。

※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。

5.7.2 回路図

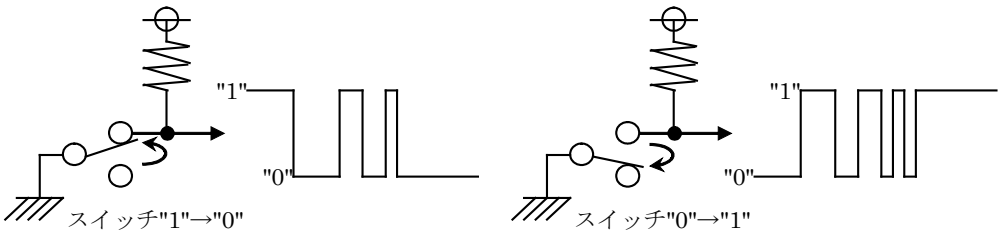


5.7.3 動作原理

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。

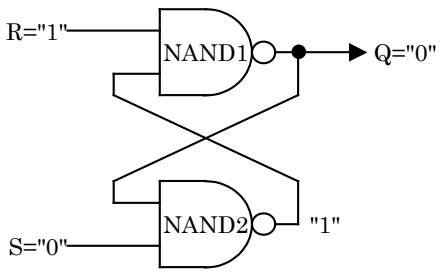
チャタリングの時間は、数ミリ秒の出来事で人間にはあまり関係ありませんが、高速で動くマイコンの場合、それぞれの状態を検出して、1 回しか変化させていないつもりでも何度も"0"、"1"を繰り返したと判断してしまいます。"1"か"0"か判断するだけならあまり問題にならないことが多いのですが、パルスの回数を数えるプログラムの場合、すべてカウントしてしまい、誤動作となってしまいます。

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。動作原理は次のとおりです。

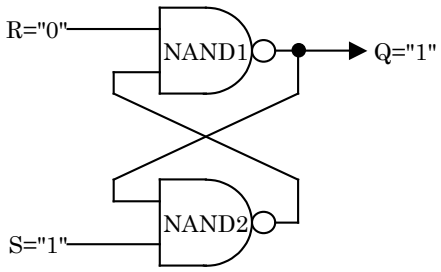
(1)入力(R,S)=("1","0")のとき



・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"

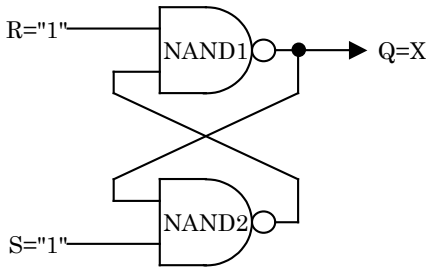
結果、リセット信号だけが"1"のため、出力 Q="0"となったと考えることができます。

(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

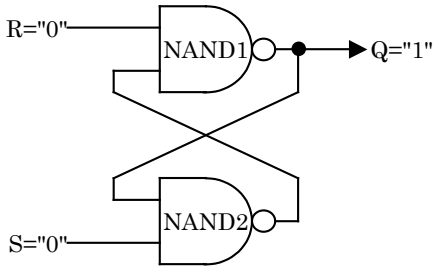


・NAND1、NAND2、共に入力が決まらなければ出力は決定されないので、NAND1 の出力 Q=X と仮定する
・NAND2 の入力は S=″1″と X なので、出力は $\overline{X}$となる
・NAND1 の入力は R=″1″と $\overline{X}$なので

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

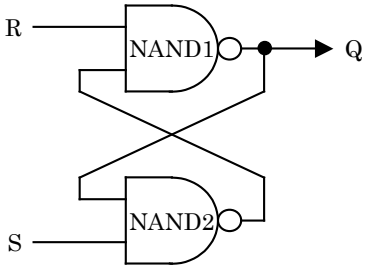
となり、最初に仮定した X と同じ値になる
これは、
・Q=″0″のとき、(R,S)=(″1″,″1″)としても前の出力を保持
・Q=″1″のとき、(R,S)=(″1″,″1″)としても前の出力を保持
するということです。

(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。

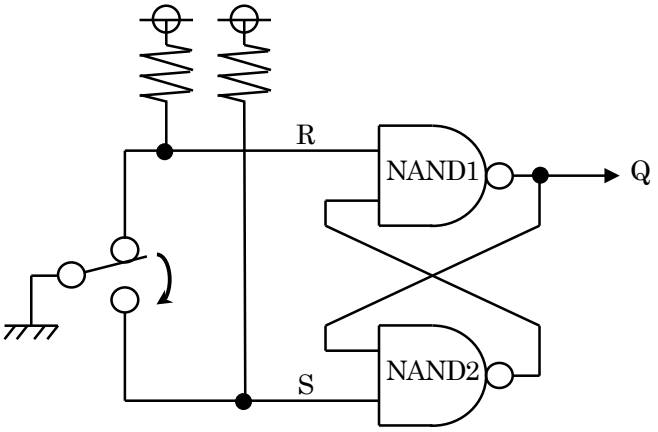
(5)まとめ

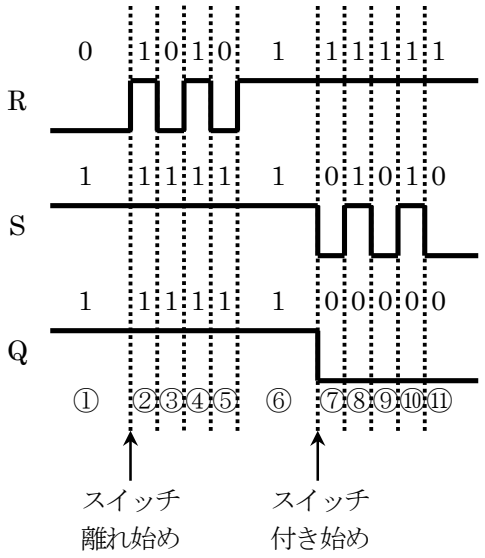


まとめると下表のようになります。

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

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。



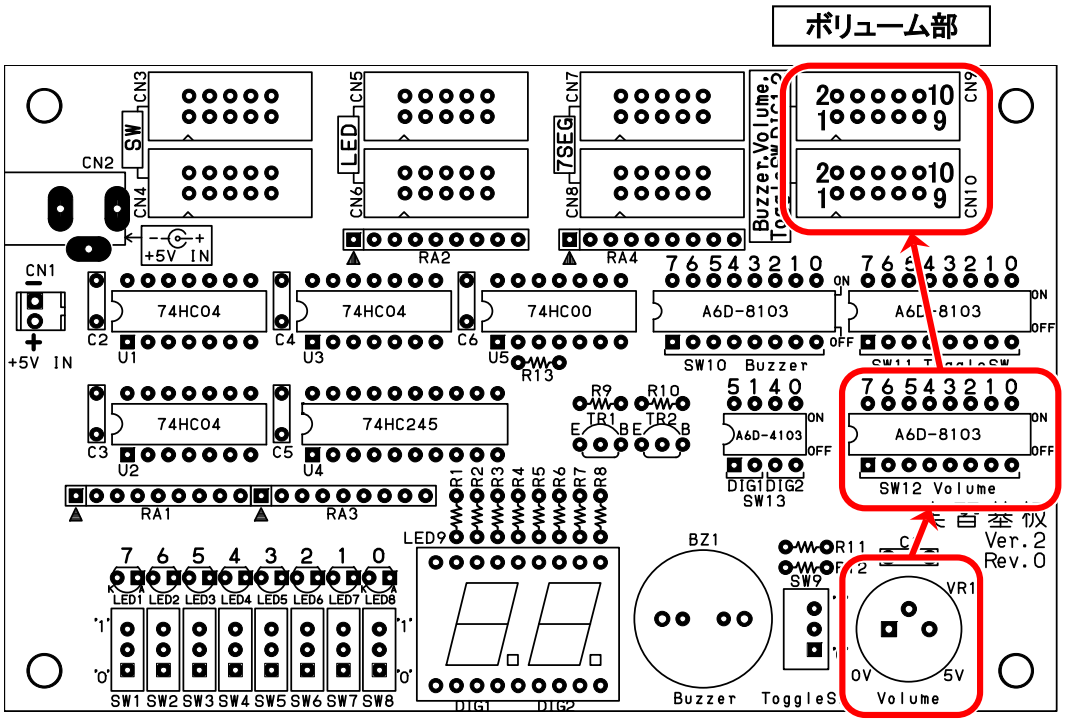


①	最初、(R,S)=(″0″,″1″)のため、Q=″1″となります。
②	スイッチを下側に切り替え、R 側から接点が離れ始めた時です。(R,S)=(″1″,″1″)のため、Q は前の値を保持します。
③	(R,S)=(″0″,″1″)のため、Q=″1″となります。
④	(R,S)=(″1″,″1″)のため、Q は前の値を保持します。
⑤	(R,S)=(″0″,″1″)のため、Q=″1″となります。
⑥	(R,S)=(″1″,″1″)のため、Q は前の値を保持します。接点が中間になりました。R 端子、S 端子共にスイッチの接点には繋がっていない状態ですが、プルアップ抵抗があるので″1″になっています。
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。

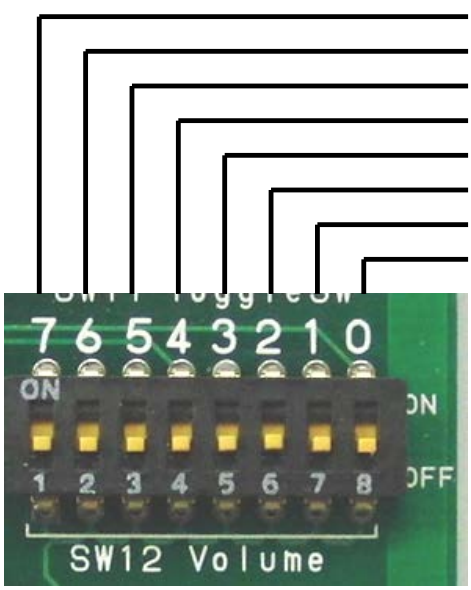
5.8 ボリューム部

5.8.1 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。



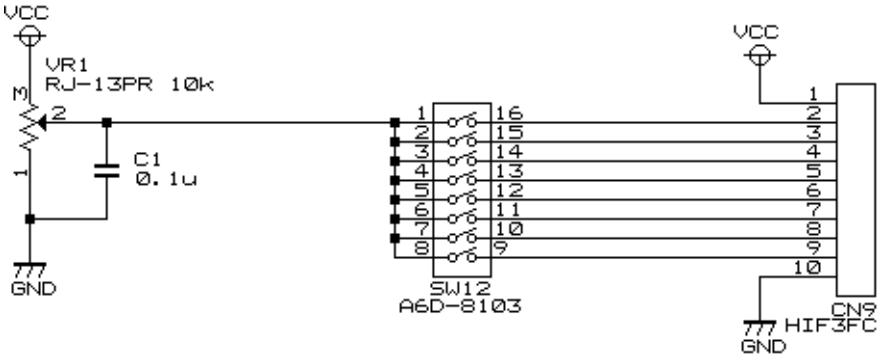
CN9(CN10)とボリュームを、どの bit に接続するかは SW12 で決めます。CN9(CN10)と SW12 の関係を下図に示します。



ON するとボリュームと CN9 の bit7 が接続されます。
ON するとボリュームと CN9 の bit6 が接続されます。
ON するとボリュームと CN9 の bit5 が接続されます。
ON するとボリュームと CN9 の bit4 が接続されます。
ON するとボリュームと CN9 の bit3 が接続されます。
ON するとボリュームと CN9 の bit2 が接続されます。
ON するとボリュームと CN9 の bit1 が接続されます。
ON するとボリュームと CN9 の bit0 が接続されます。

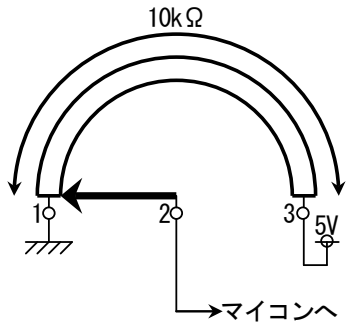
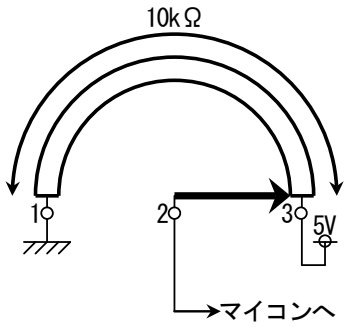
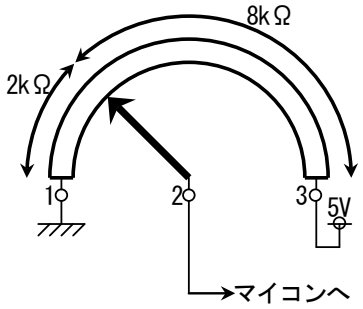
※SW12 は 1 つだけ ON にしてください。
複数 ON にすると、CN9 の ON にした端子同士が接続状態となり
ポートの入出力設定によっては、ショート状態となります。

5.8.2 回路図

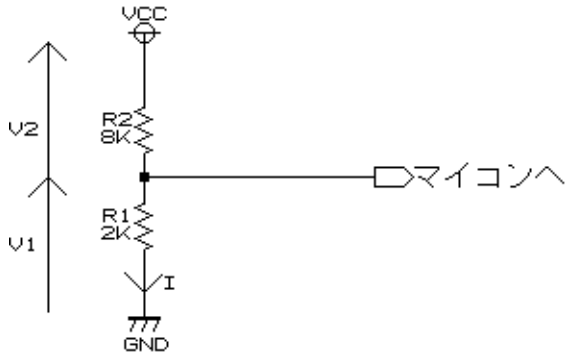


5.8.3 動作原理

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。

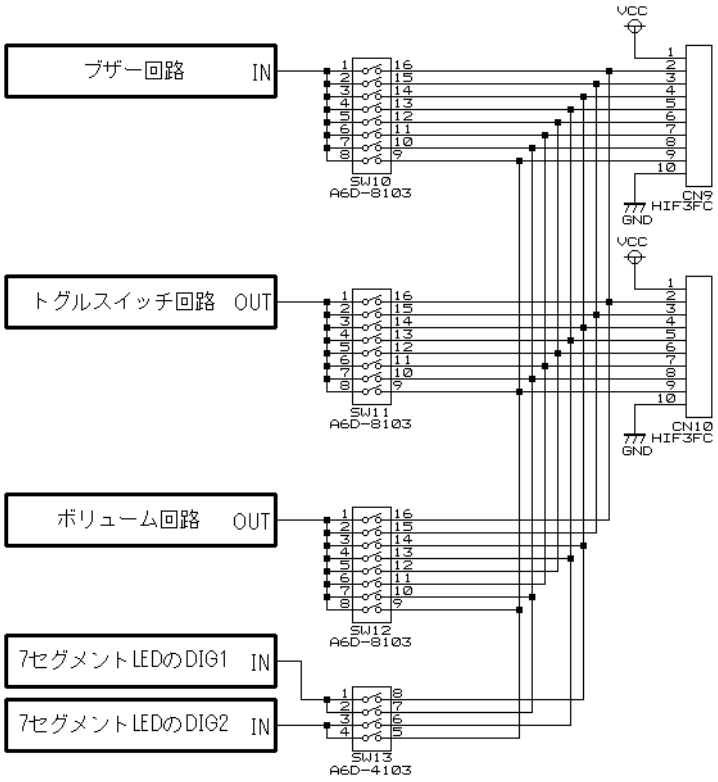


I ＝ Vcc／(R1+R2)
＝ 5／(2K＋8K)
＝ 5／10K
＝ 0.0005A
＝ 0.5mA
V1＝ I･R1
＝ 0.0005×2K
＝ 1.0V

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。

5.9　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。

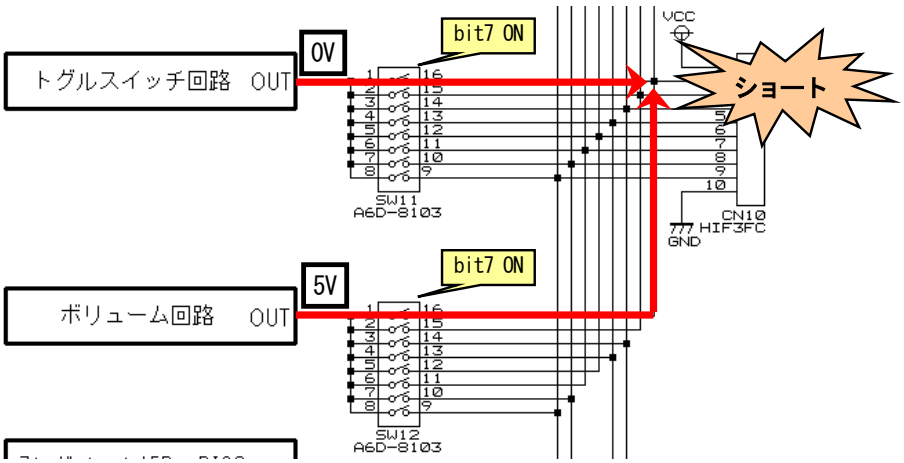


例えば、下記の状態とします。

・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。

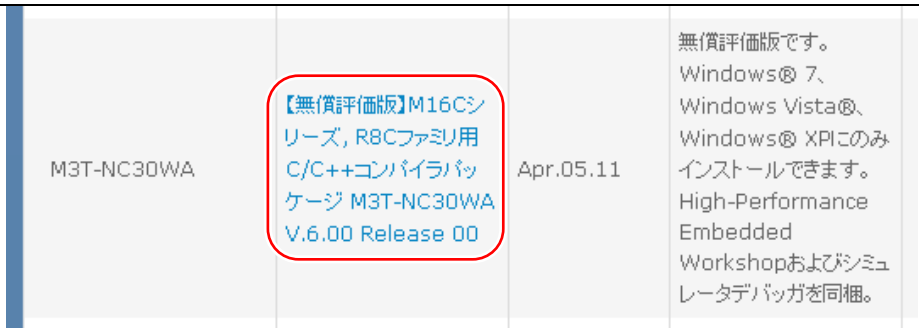
6. 開発環境のダウンロード、インストール

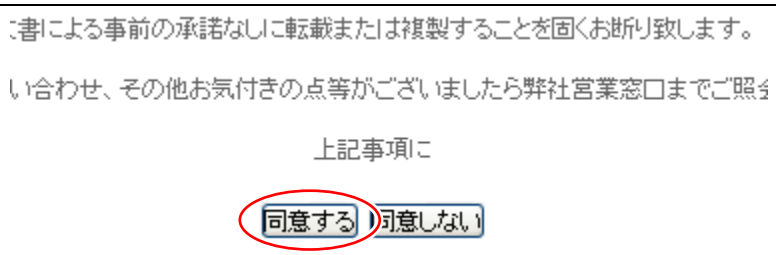
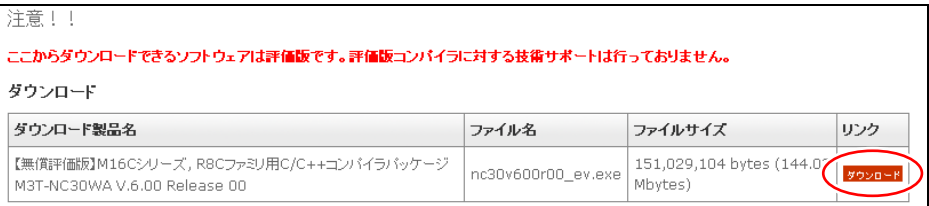
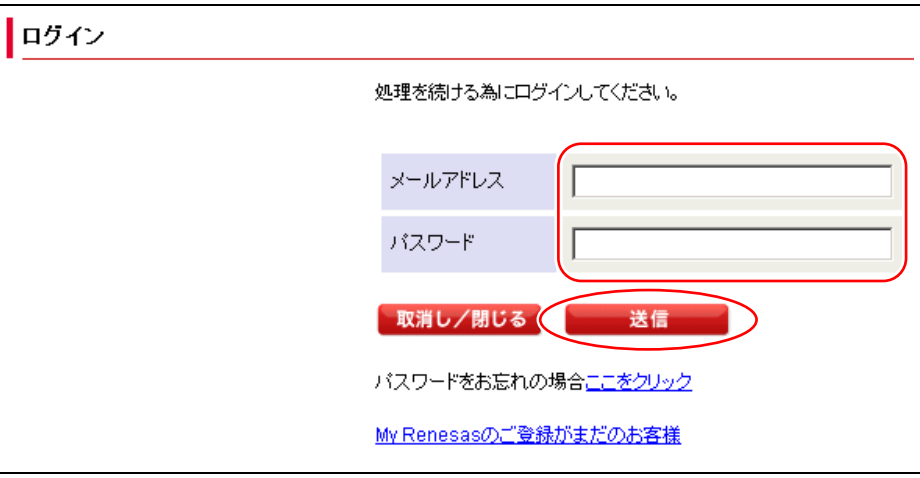
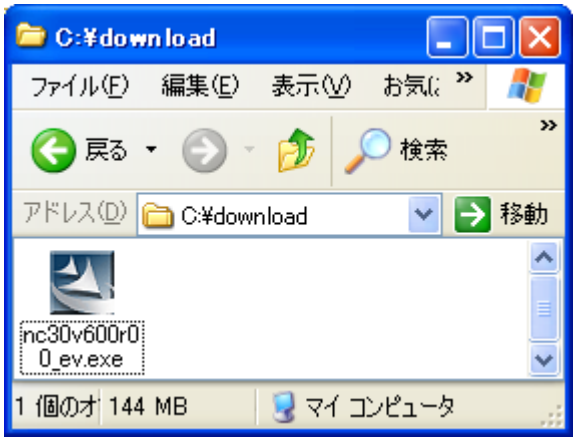
6.1 MY Renesas に登録

R8C/38A 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。

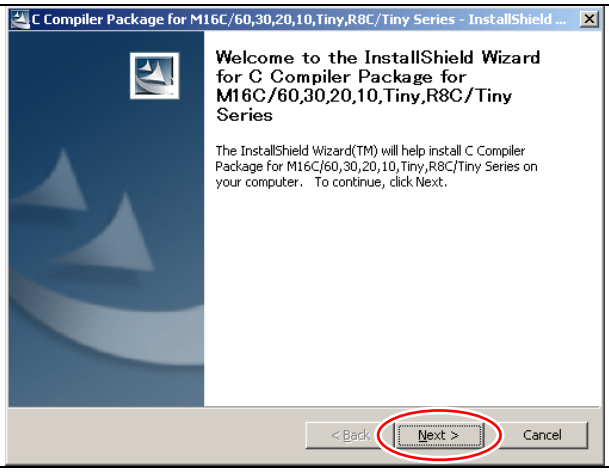
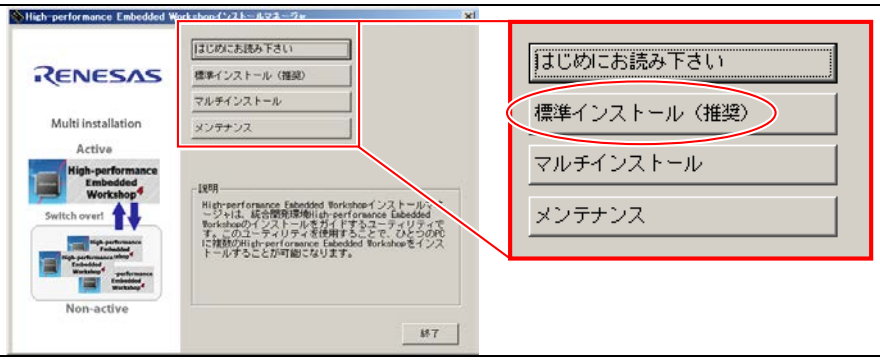
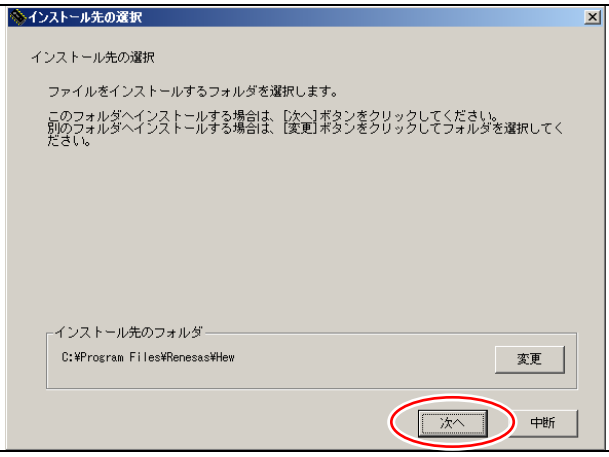
1		ルネサス エレクトロニクスのサイト http://japan.renesas.com/ にアクセスします。 ○部の「MY Rnenesas」をクリックします。
2		新規登録をクリックして、登録手続きをしてください。メールアドレスとパスワードは、ファイルをダウンロードするときに必要ですので控えておいてください。

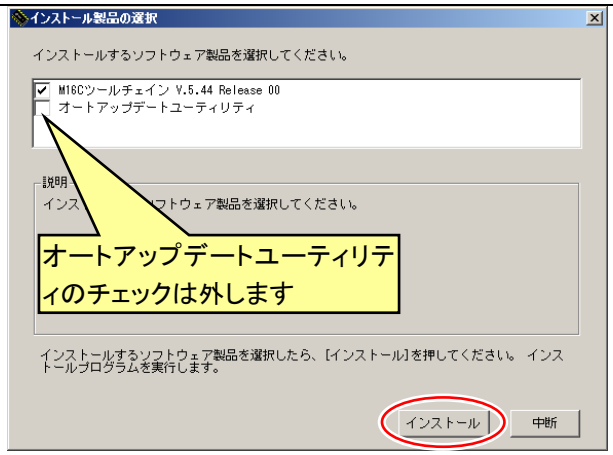
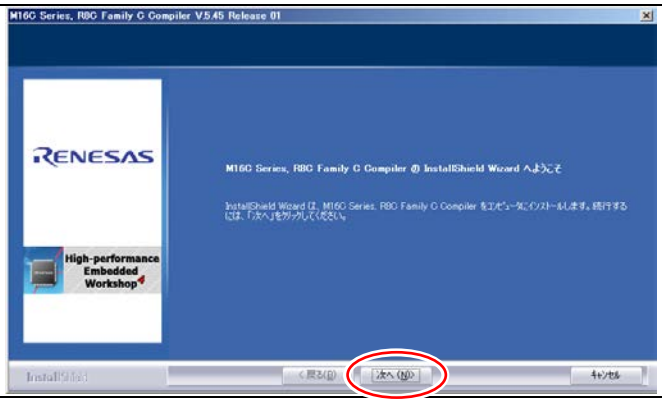
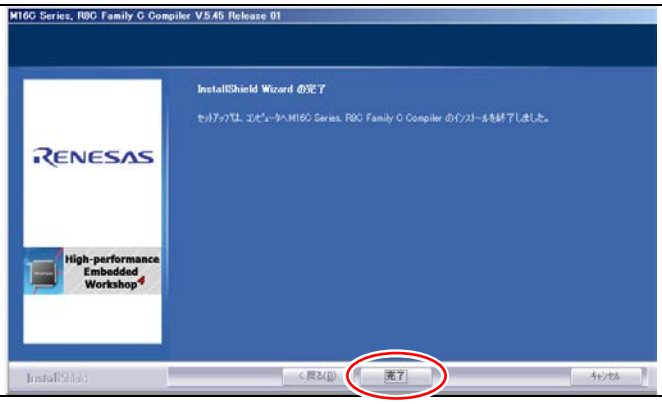
6.2　ルネサス統合開発環境のダウンロード

1		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→開発環境」をクリックします。
2		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。
3		**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。
4		**「【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、ホームページの 2 ページ以降を参照してください。

5		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。
6		ダウンロードをクリックします。
7		My Renesas で登録したメールアドレス、パスワード入力して、送信をクリックします。
8		ダウンロードできました。

6.3　ルネサス統合開発環境のインストール

1		ダウンロードしたルネサス統合開発環境インストールファイル「nc30v600r00_ ev.exe」を実行します(バージョンにより v600r00 部分は異なります)。 **※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。**
2		Next >をクリックします。
3		標準インストール（推奨）をクリックします。
4		次へをクリックします。

5		インストールをクリックします。
6		次へをクリックします。
7		完了をクリックして、インストール完了です。

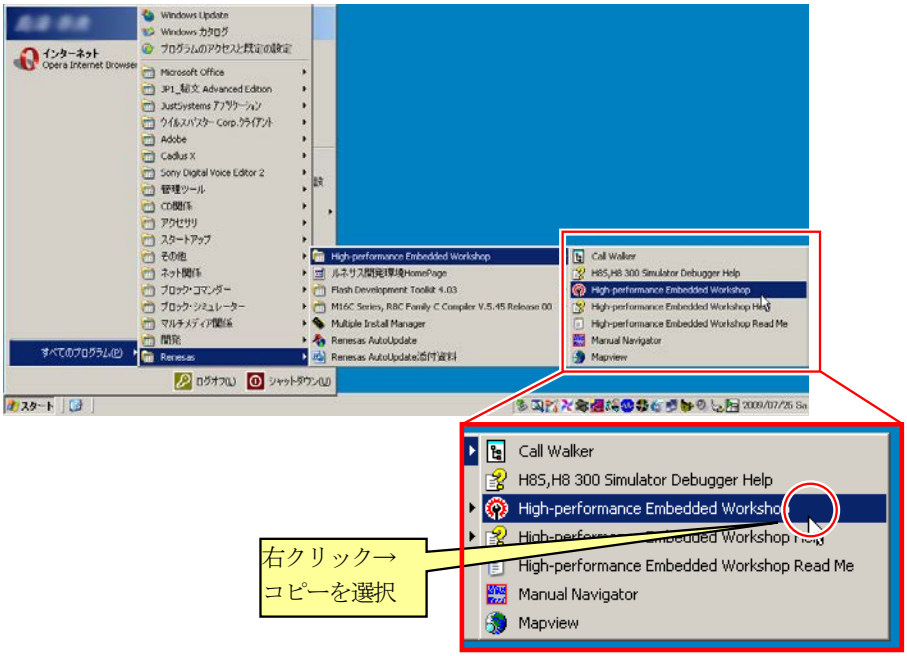
※ルネサス統合開発環境の無償評価版の制限について

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。

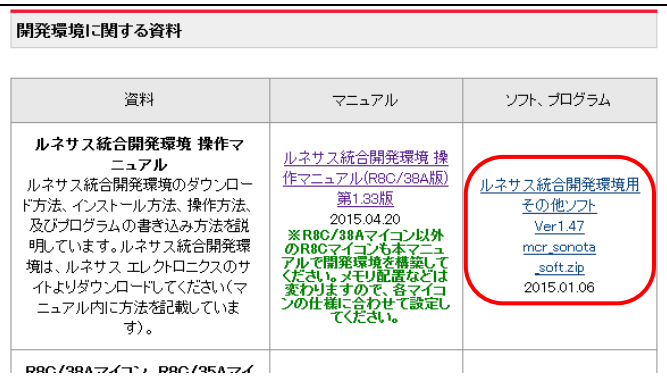
6.4 ショートカットの作成

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。

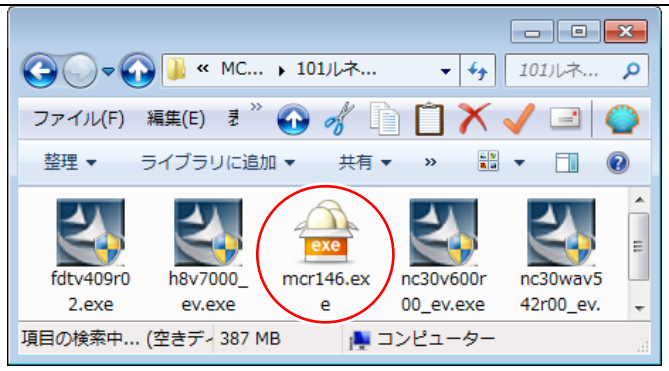
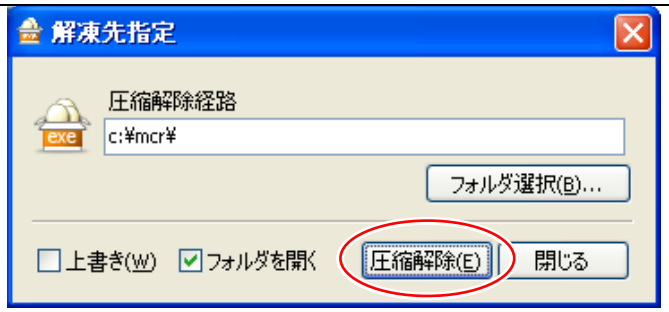
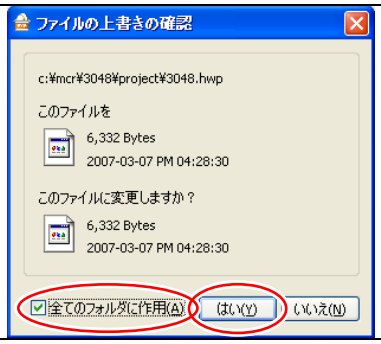
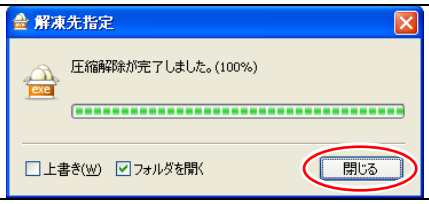
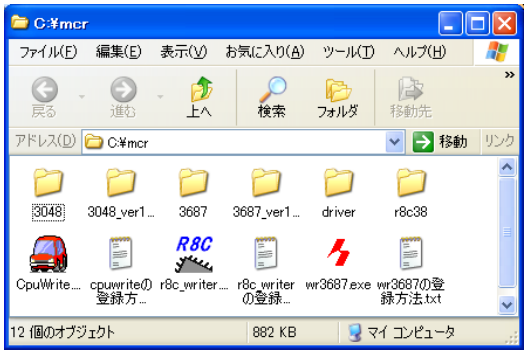
1		「スタート」 ↓ 「すべてのプログラム」 ↓ 「Renesas」 ↓ 「High-performance Embedded Workshop」 ↓ までたどっていき、 「High-performance Embedded Workshop」 で、右クリックして、「コピー」をクリックします。
2		デスクトップで右クリックして、「貼り付け」をクリックします。デスクトップにショートカットが作成されます。

6.5　R8C　Writer のダウンロード

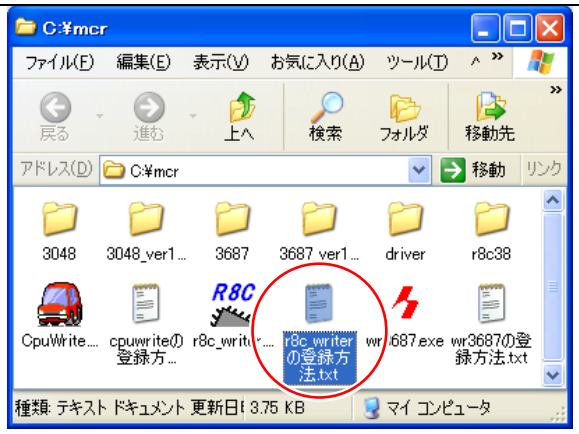
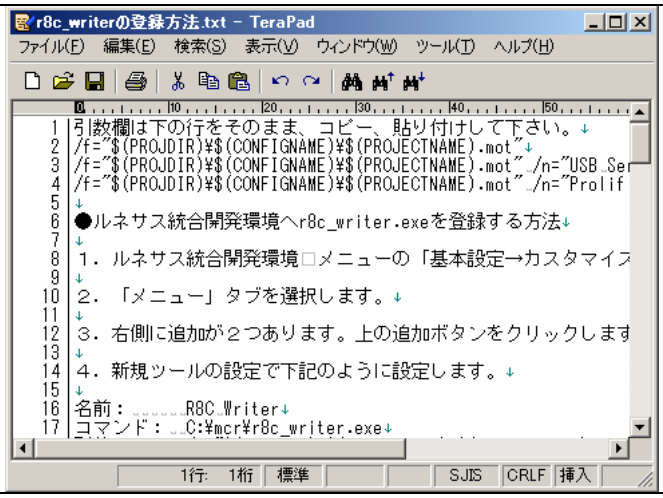
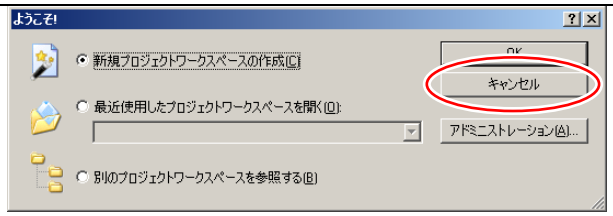
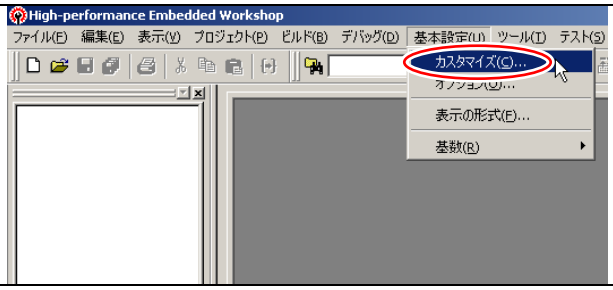
R8C　Writer とは、R8C/38A マイコンにプログラムを書き込むソフトウェアです。ルネサス統合開発環境に組み込んで使用します。

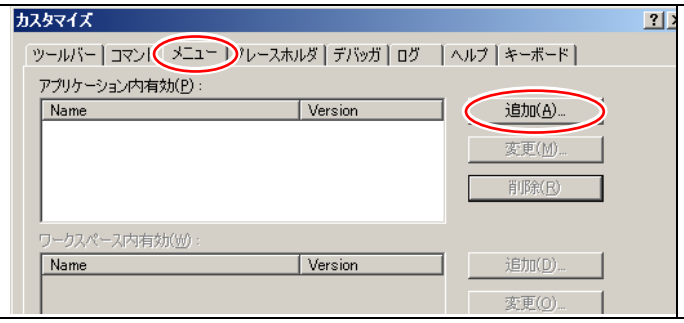
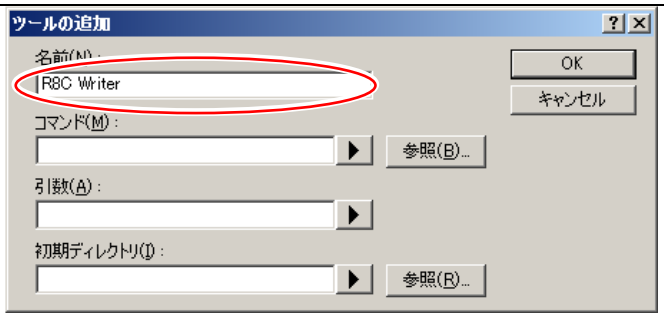
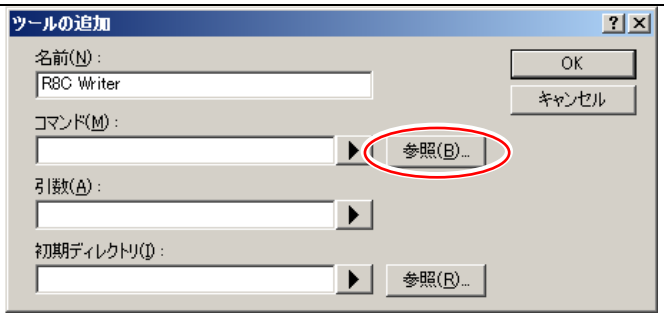
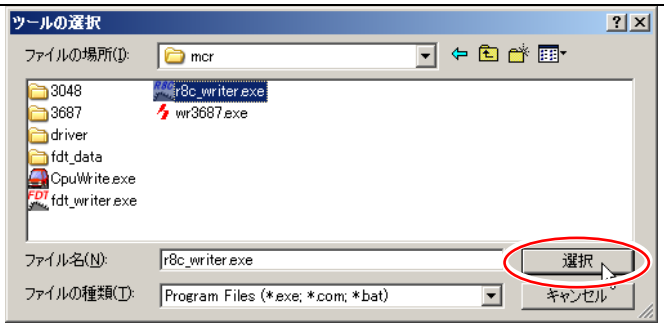
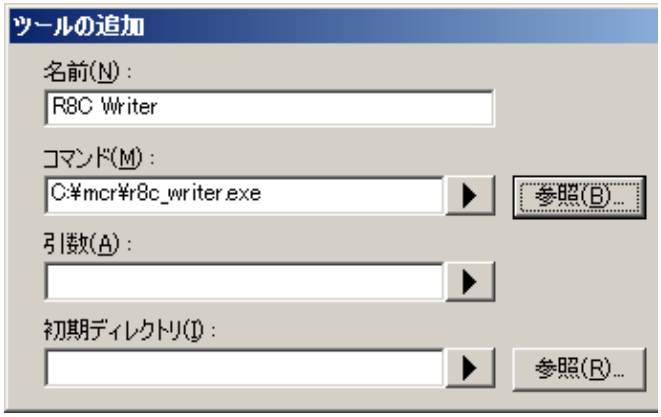
1		マイコンカーラリー販売サイト「https://www2.himdx.net/mcr/」のダウンロードのページへ行きます。
2		「ルネサス統合開発環境用その他ソフト」をダウンロードします。 このファイルに圧縮されている「mcr146.exe」ファイルを、解凍ソフトで解凍します。

6.6　R8C　Writer のインストール

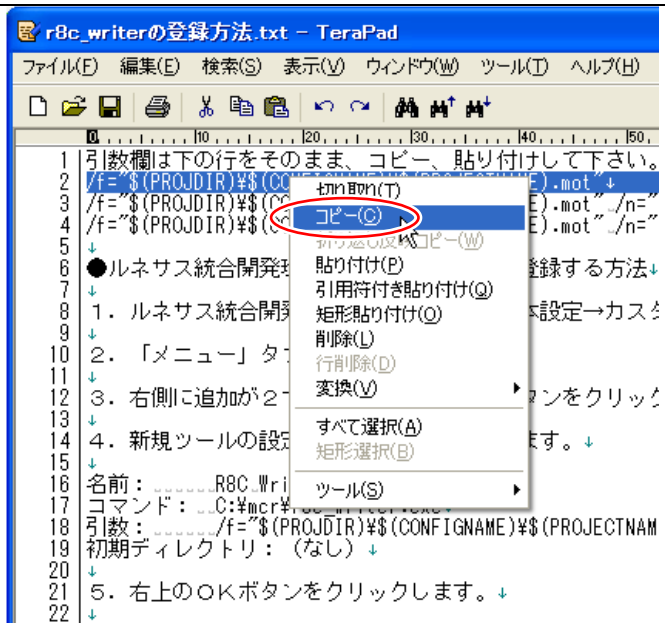
1		「mcr146.exe」を実行します。 ※「146」はバージョンです。ダウンロードした時期により異なることがあります。
2		圧縮解除をクリックします。 ※フォルダ（圧縮解除経路）は替えないでください。替えた場合は、次で行う R8C Writer の登録するフォルダが替わります。
3		もし、左画面がでてきた場合、「全てのファイルに作用」のチェックを付けて、はいをクリックして上書きコピーします。 ※上書きしたくない場合は、一度終了して、元あるファイルを移動してから、再度実行してください。
4		閉じるをクリックします。
5		インストール先のフォルダが開かれます。

6.7　R8C　Writer をルネサス統合開発環境に登録する

1		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。
2		この内容は、後ほど使用しますので最小化しておきます。
3		ルネサス統合開発環境を実行します。
4		キャンセルをクリックします。
5		「基本設定→カスタマイズ」をクリックします。

6		「メニュー」タブをクリックし、追加をクリックします。 ※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。
7		名前欄に次のように入力します。 R8C Writer
8		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。
9		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。
10		コマンドが入力されました。

11



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。

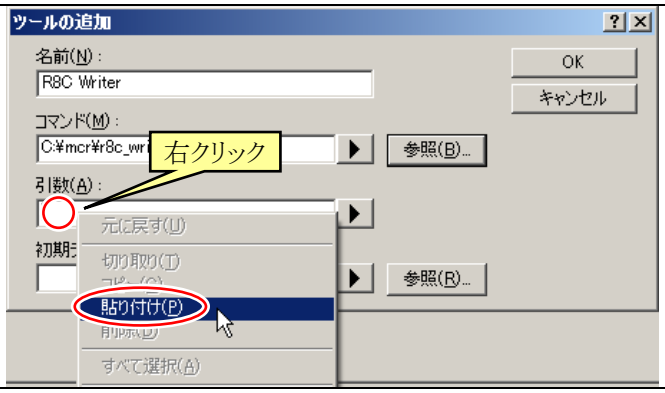
・4 行目を選択すると、RY-WRITER 基板で使用している Prolific 社製の USB シリアル変換 IC が接続されている COM ポートを自動選択します。RY-WRITER 基板を使うときは、4 行目を選択してください。

・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。

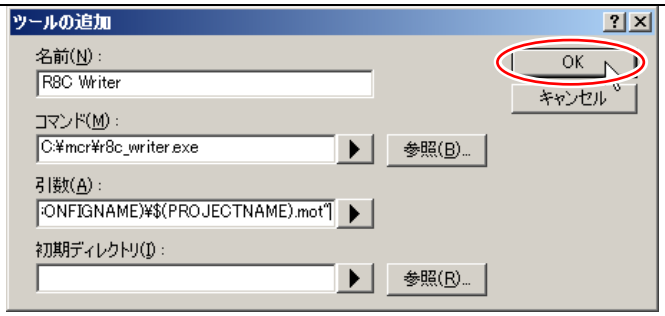
コピー後は、「r8c_writer の登録方法.txt」ファイルを閉じて構いません。

12



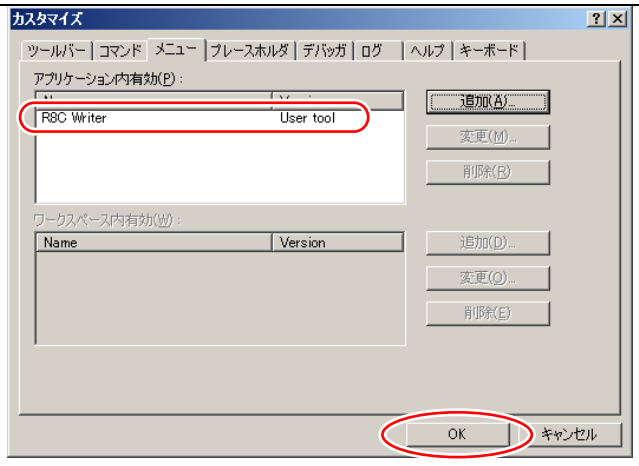
ツールの追加画面に戻り、引数欄で右クリックして「貼り付け」をクリックします。

13



OK をクリックして、ツールの追加を完了します。

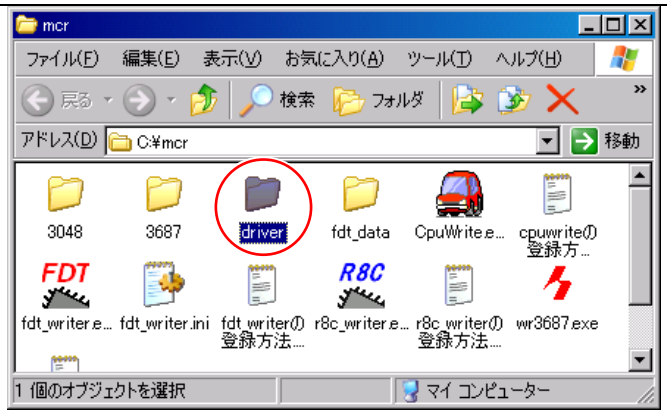
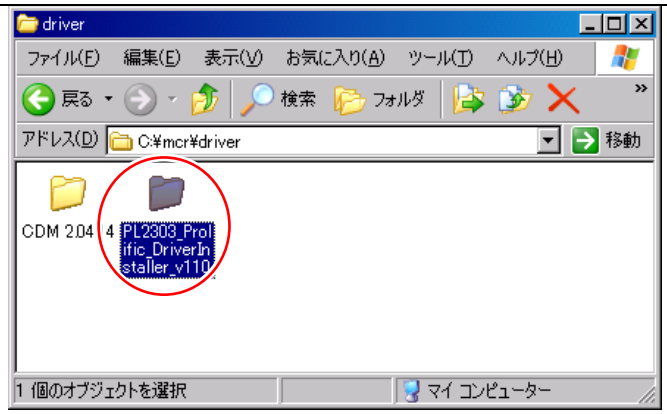
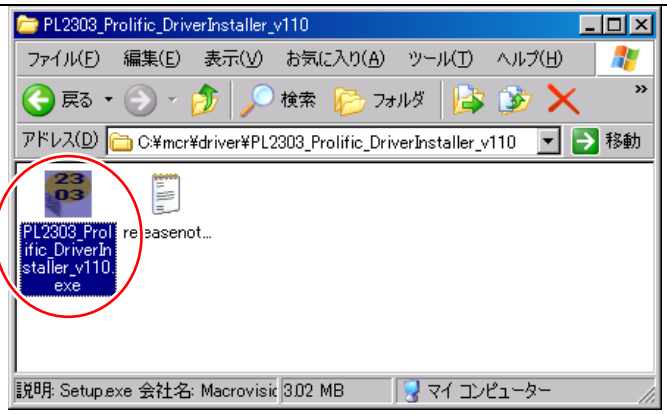
14

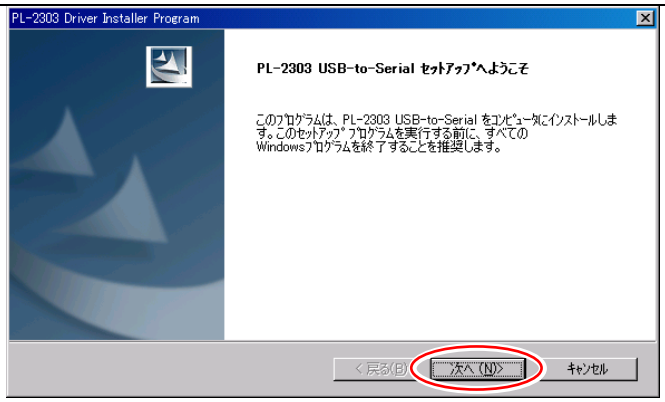
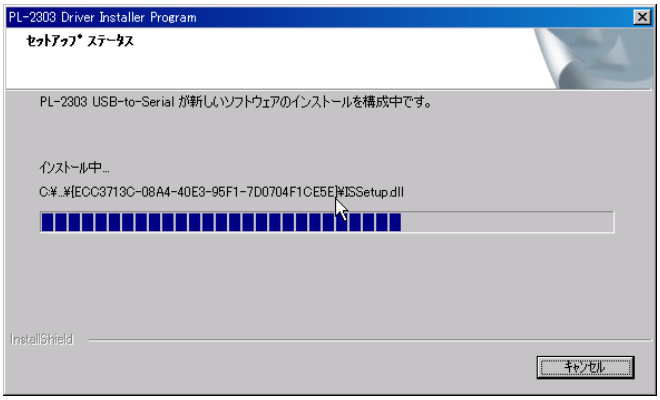
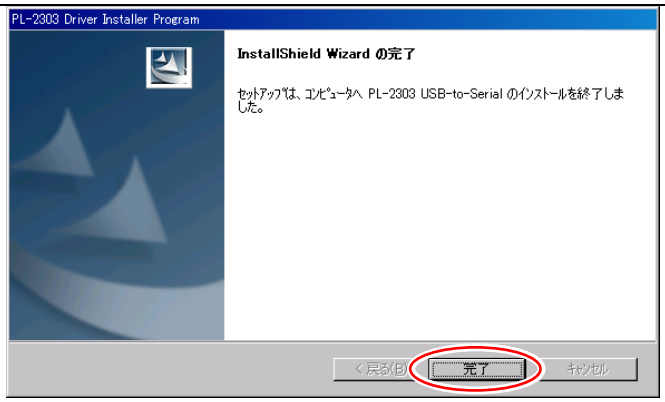
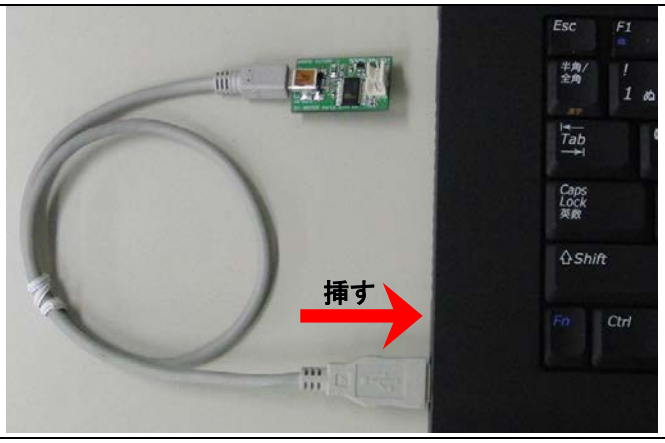


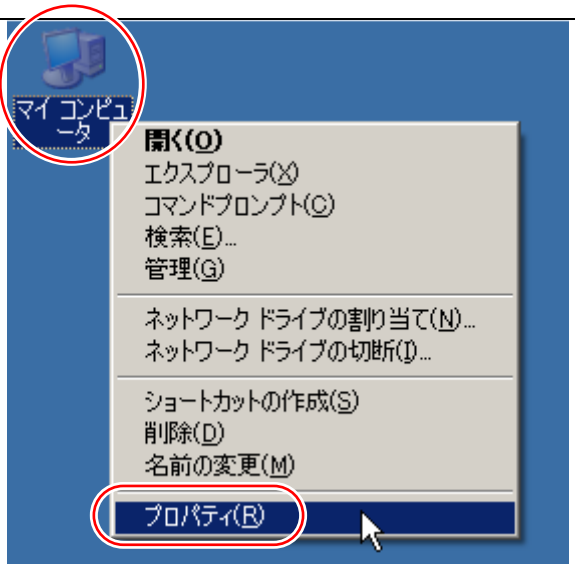
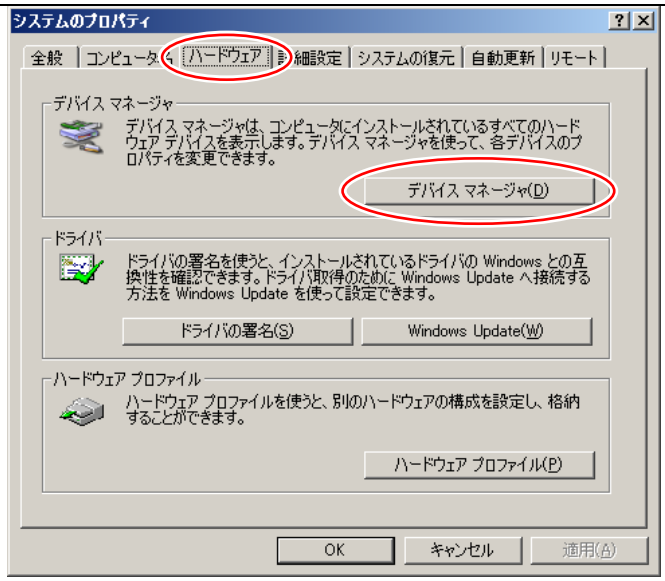
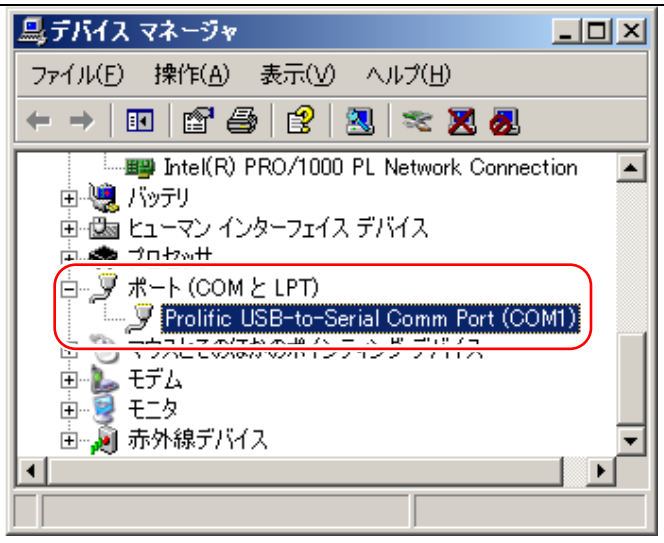
アプリケーション内有効に「R8C Writer」があることを確認して、OK をクリックします。無い場合は登録が正しくできていませんので手順を再確認してもう一度登録してください。アプリケーション内有効に他の内容があっても問題ありません。

6.8 ドライバーのインストール

※RY-WRITER 基板を使って RY_R8C38 ボードへプログラムを書き込む場合、USB-シリアル変換 IC のドライバーをインストールしてください。

1		「C ドライブ→mcr フォルダ→driver」フォルダを開きます。
2		「PL2303_Prolific_DriverInstaller_v110」フォルダを開きます。 ※「110」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。
3		「PL2303_Prolific_DriverInstaller_v110.exe」を実行します。 ※「110」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。

4		次へをクリックします。 ※「**選択したアプリケーション、およびすべての機能を完全に削除しますか？**」というメッセージが出た場合、既にドライバーがインストールされていますのでいいえをクリックして、インストールを中止します。ただし、最新版のドライバーが入っているか分からない場合、はいをクリックしてドライバーをアンインストールしてから、最新版のドライバー(今回のファイル)を改めてインストールしてください。
5		インストール中です。
6		完了をクリックして、完了します。
7		RY-WRITER 基板をパソコンの USB コネクタに挿します。

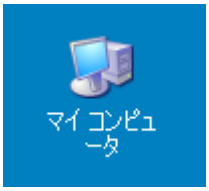
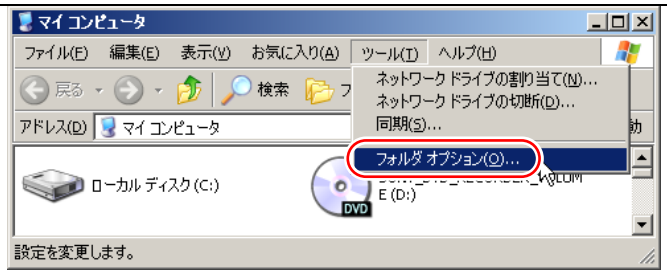
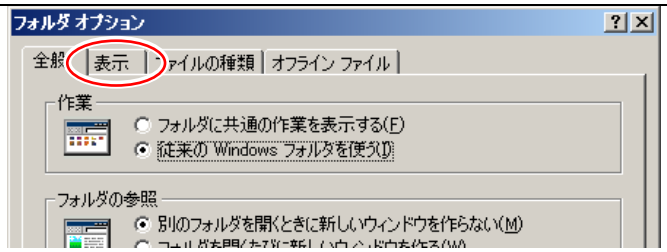
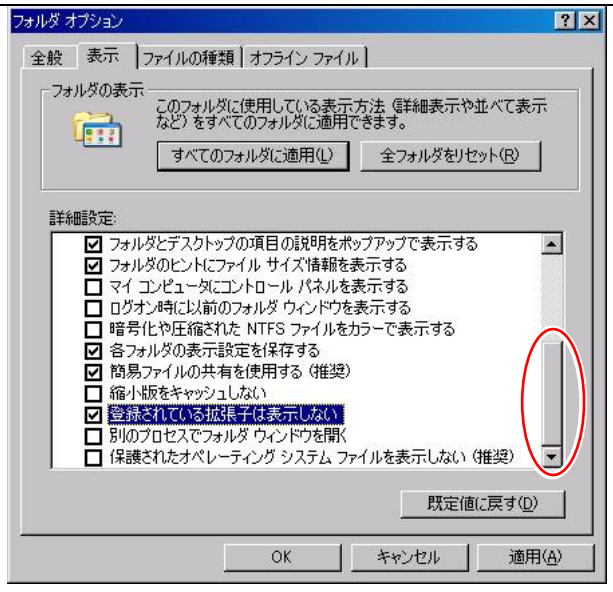
8		デスクトップにある「マイコンコンピュータ」で右クリック、プロパティを選択します。 デスクトップにマイコンピュータが無い場合は、コントロールパネルのシステムを選択してください。 ※Windows＋Pause キーでもシステム画面を開くことができます。
9		「ハードウェア」タブを選択します。 デバイスマネージャをクリックします。
10		「ポート(COM と LPT)」をクリックすると、 Prolific USB-to-Serial Comm Port(COM●) があります。この内容が今回インストールした項目です。 Prolific USB-to-Serial Comm Port が無い場合、USB ケーブルの接続などを確認してください。 ※●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。

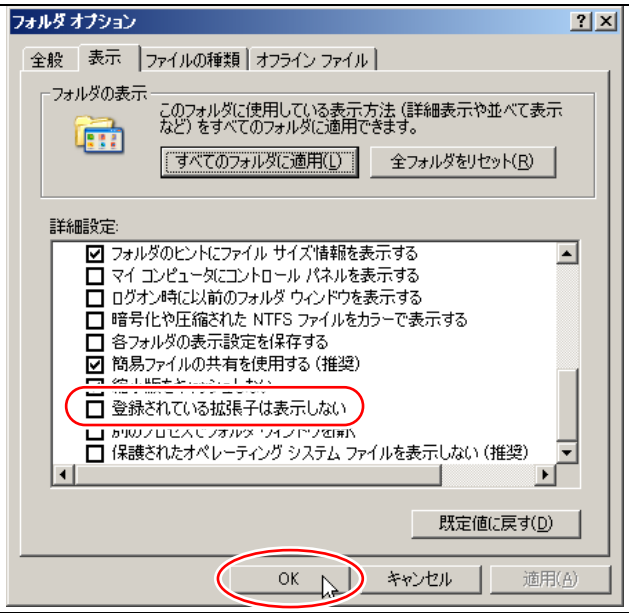
6.9　拡張子の表示

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。

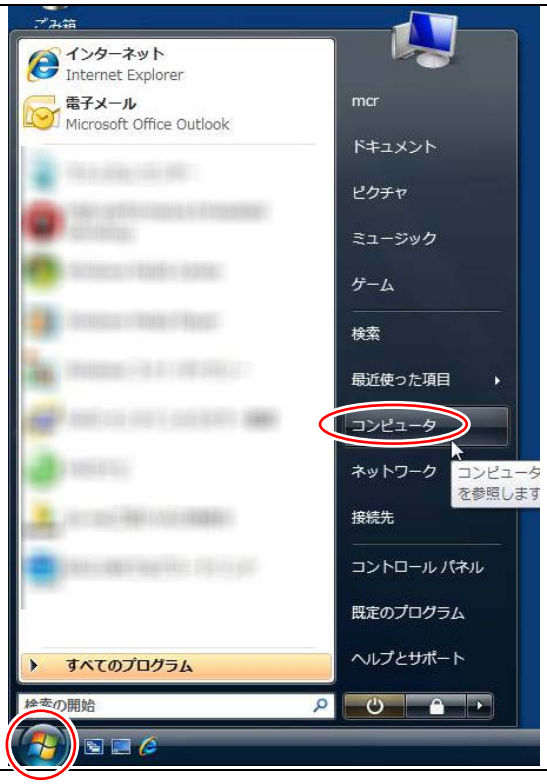
Windows XPの場合は「**6.9.1　WindowsXP の場合**」を、WindowsVistaの場合は「**6.9.2　Windows Vista、Windows 7 の場合**」を参照してください。

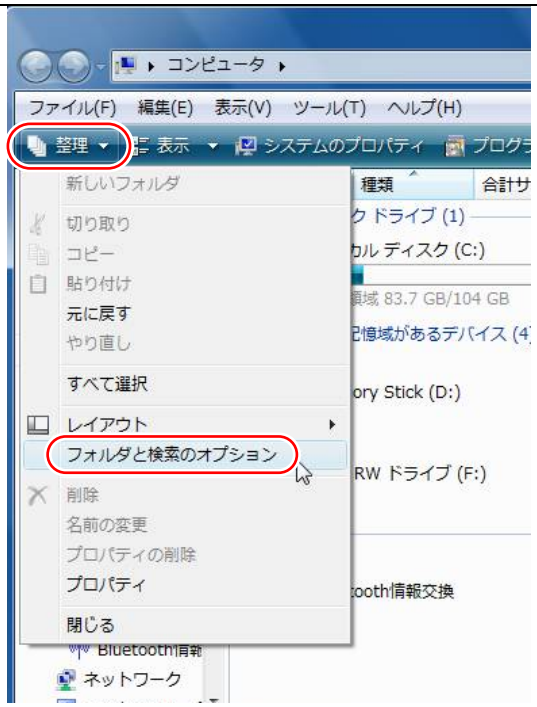
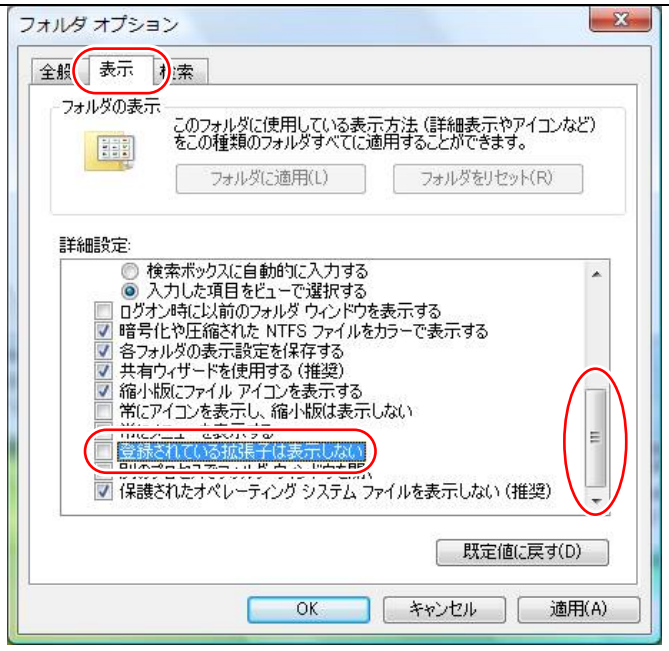
6.9.1　WindowsXP の場合

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

5		「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。

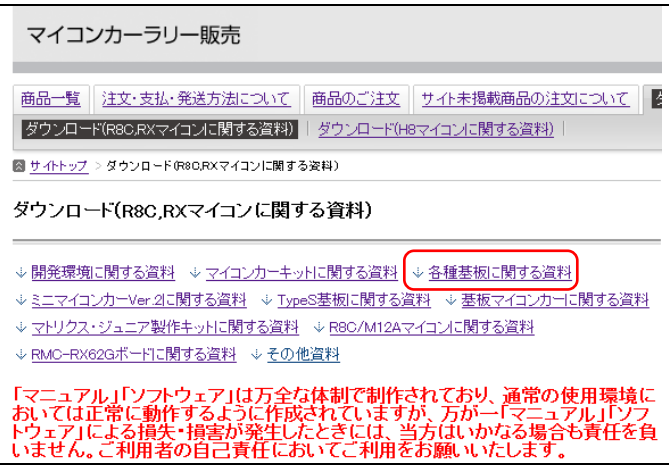
6.9.2　Windows Vista、Windows 7 の場合

1		「スタート（左下のボタン）→コンピュータ」をクリックします。

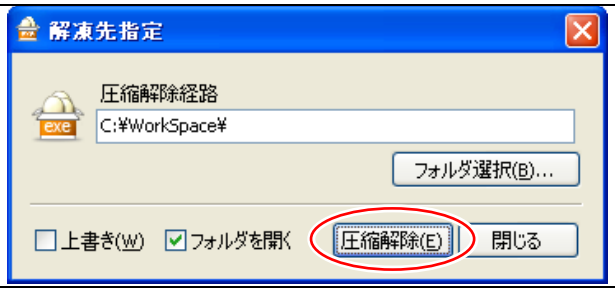
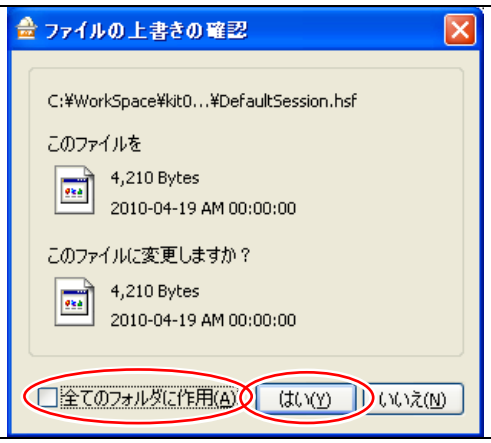
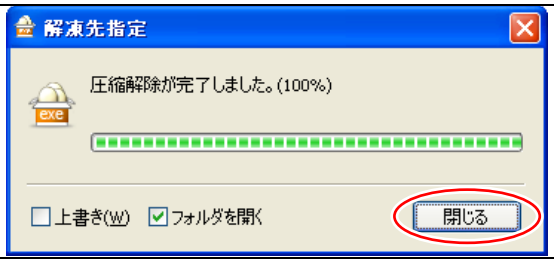
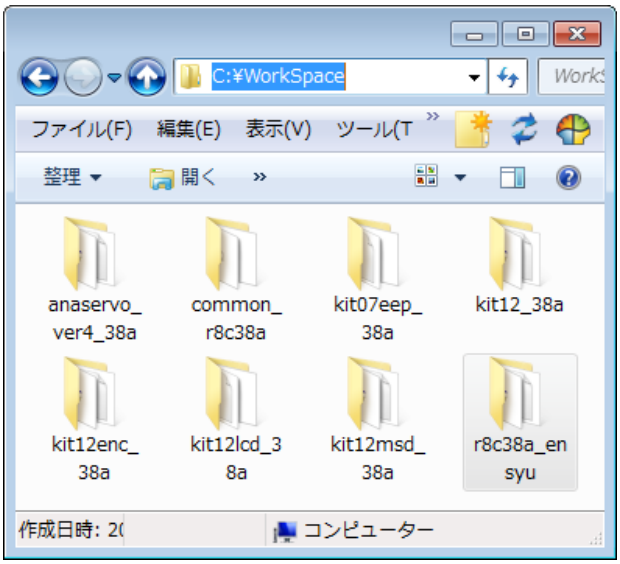
2		「整理→フォルダと検索のオプション」をクリックします。
3		「表示」タブをクリックします。 詳細設定のスクロールバーを下に移動します。 「登録されている拡張子は表示しない」の**チェックを外します**。

7.　サンプルプログラムのダウンロード、インストール

7.1　サンプルプログラムのダウンロード

1		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。
2		「各種基板に関する資料」をクリックします。
3		「r8c38a_ensyu.zip」をダウンロードします。 このファイルに圧縮されている「r8c38a_ensyu.exe」ファイルを、解凍ソフトで解凍します。

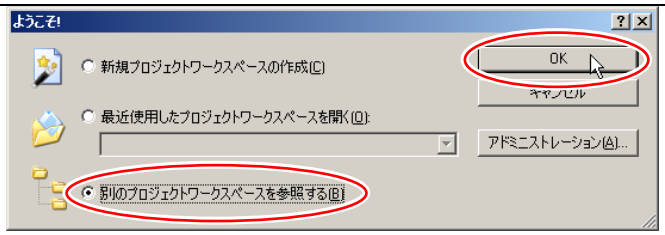
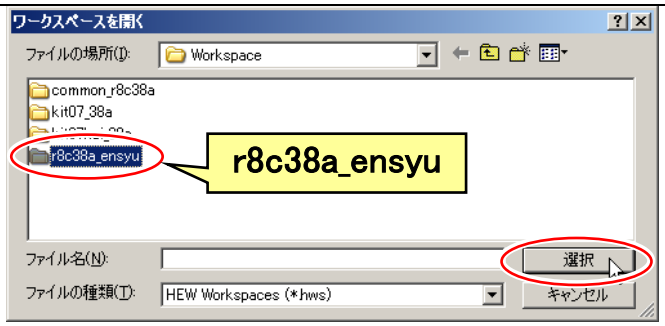
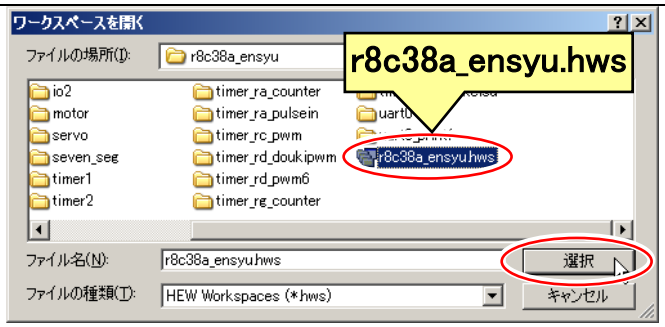
7.2　サンプルプログラムのインストール

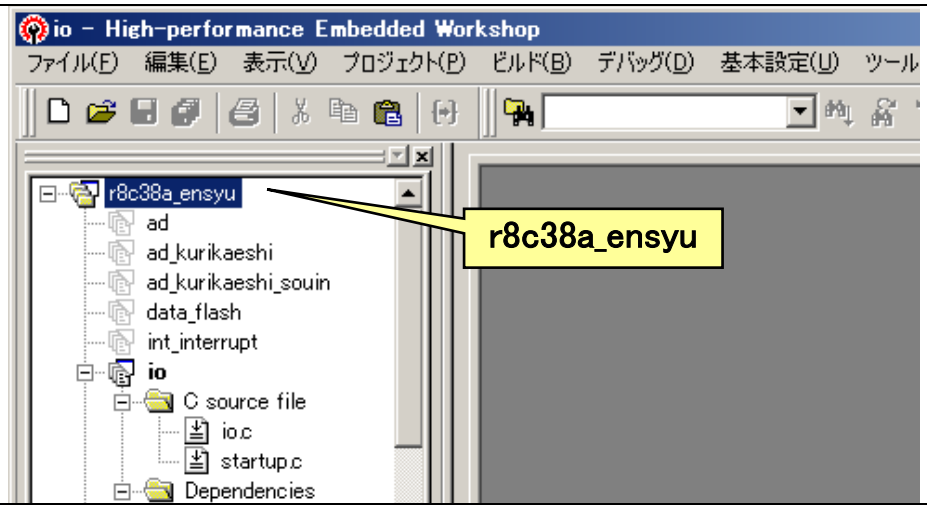
1		「r8c38a_ensyu.exe」を実行します。 圧縮解除をクリックします。
2		ファイルの上書き確認の画面が出てきた場合、「全てのファイルに作用」のチェックを付けて、はいをクリックします。 ※上書きしたくない場合は、元々あるファイルを保存してから実行してください。
3		インストールが終わりました。 OK をクリックします。
4		「C:¥Workspace」のフォルダが自動で開きます。

8. 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、マイコンカーラリー販売サイト(https://www2.himdx.net/mcr/)の「ダウンロード→開発環境に関する資料」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。

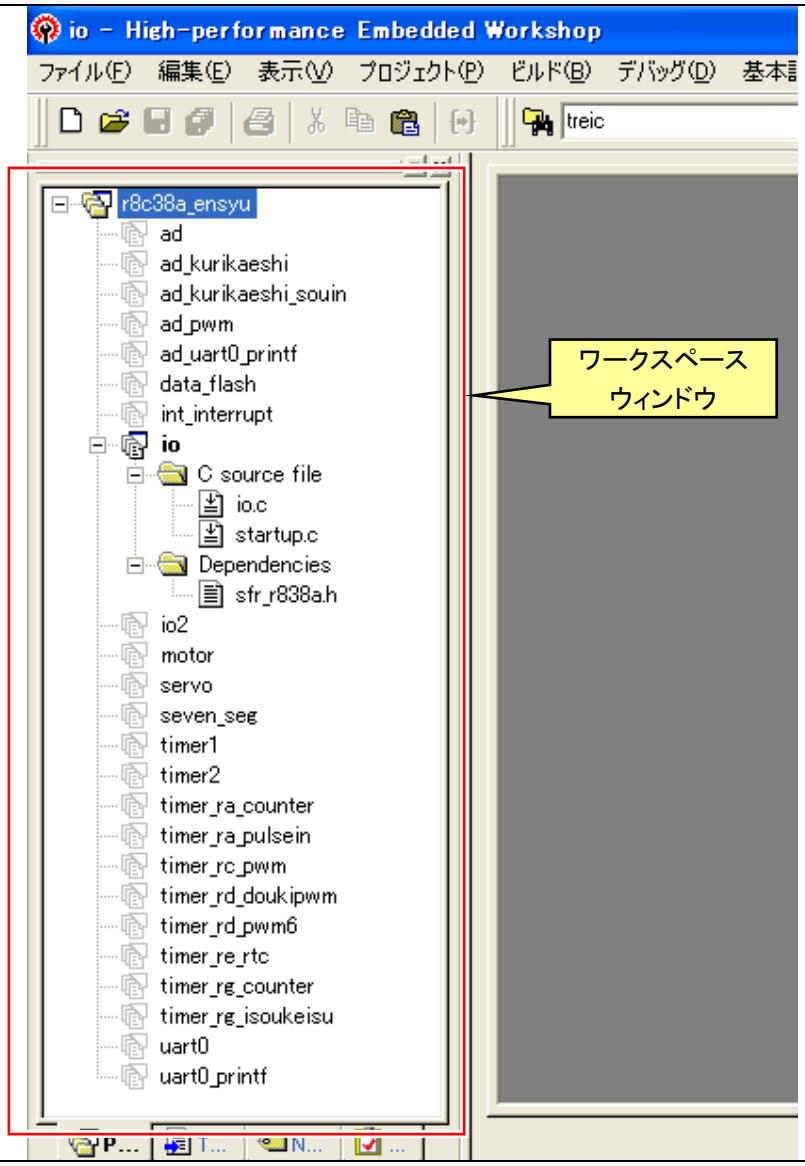
8.1 ワークスペースを開く

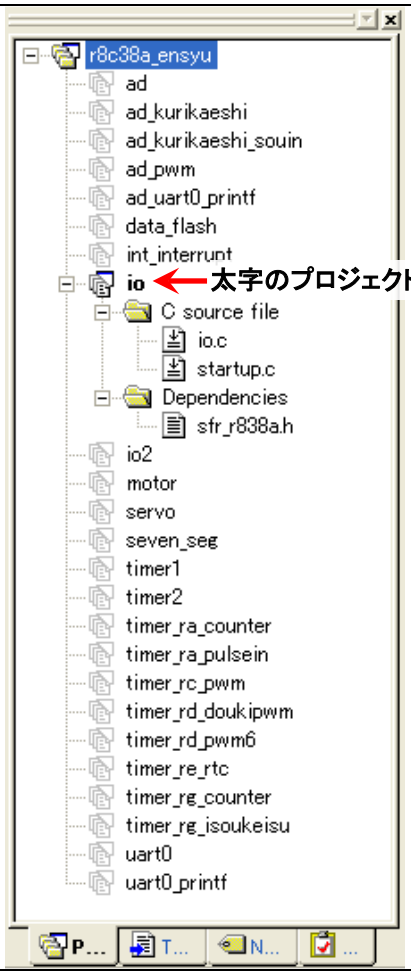
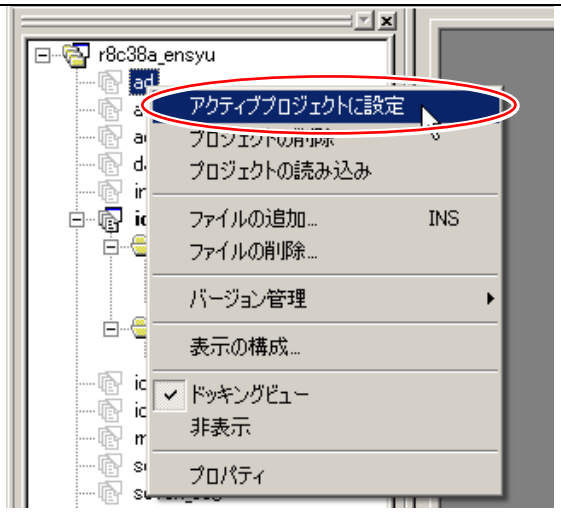
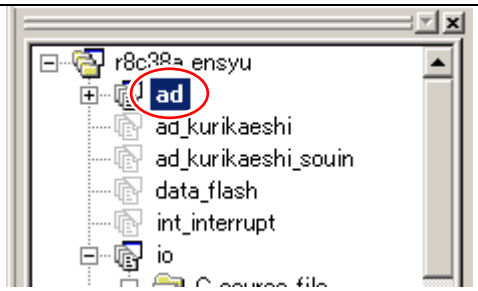
1		ルネサス統合開発環境を実行します。
2		「別のプロジェクトワークスペースを参照する」を選択、OK をクリックします。
3		「r8c38a_ensyu」を選択、選択をクリックします。
4		「r8c38a_ensyu.hws」を選択、選択をクリックします。 **※拡張子 hws ファイルがルネサス統合開発環境で開くファイルです。** ※hws は、「**H**EW **W**ork**S**pace」の略です。

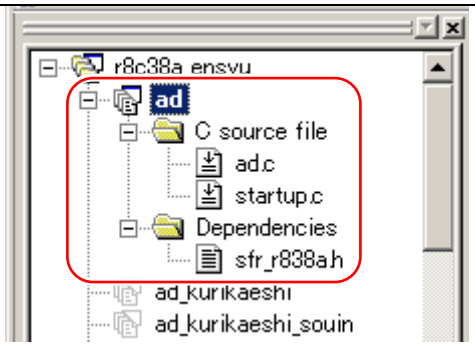
5		「r8c38a_ensyu」というワークスペースが開かれました。本実習マニュアルでは、このワークスペースを使用します。

8.2　プロジェクトを開く

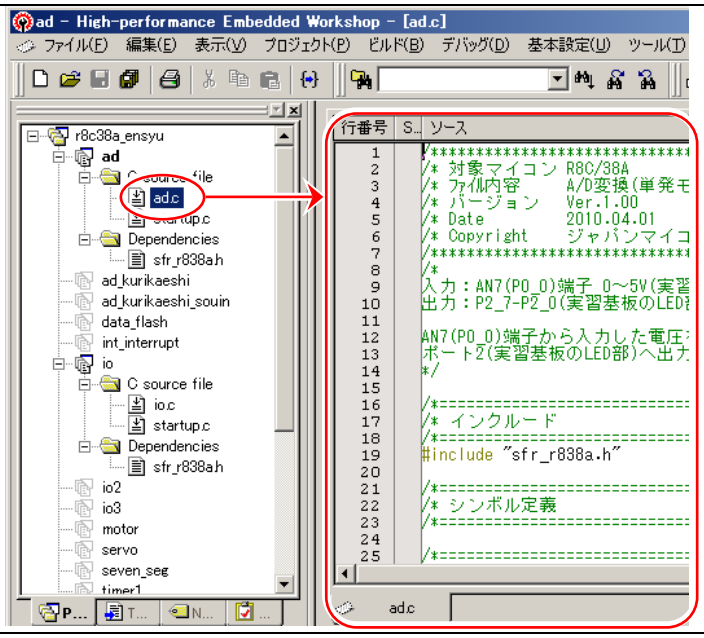
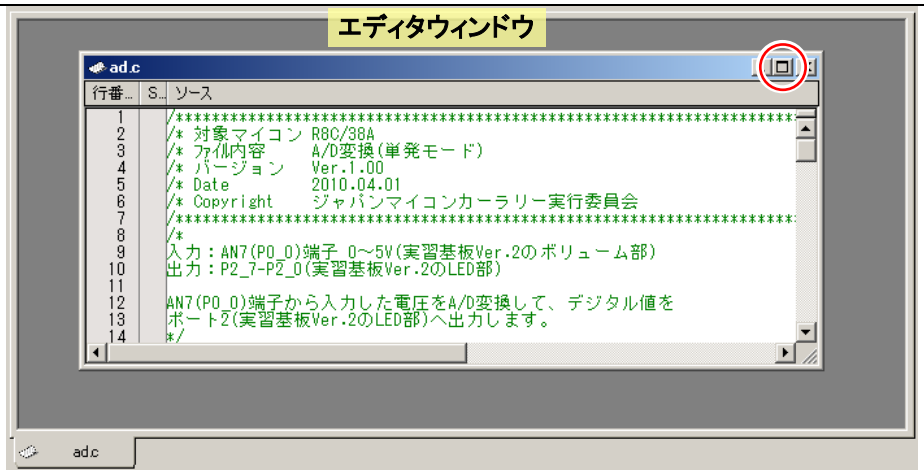
例えば、これからプロジェクト「ad」の演習をするとします。

1		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。

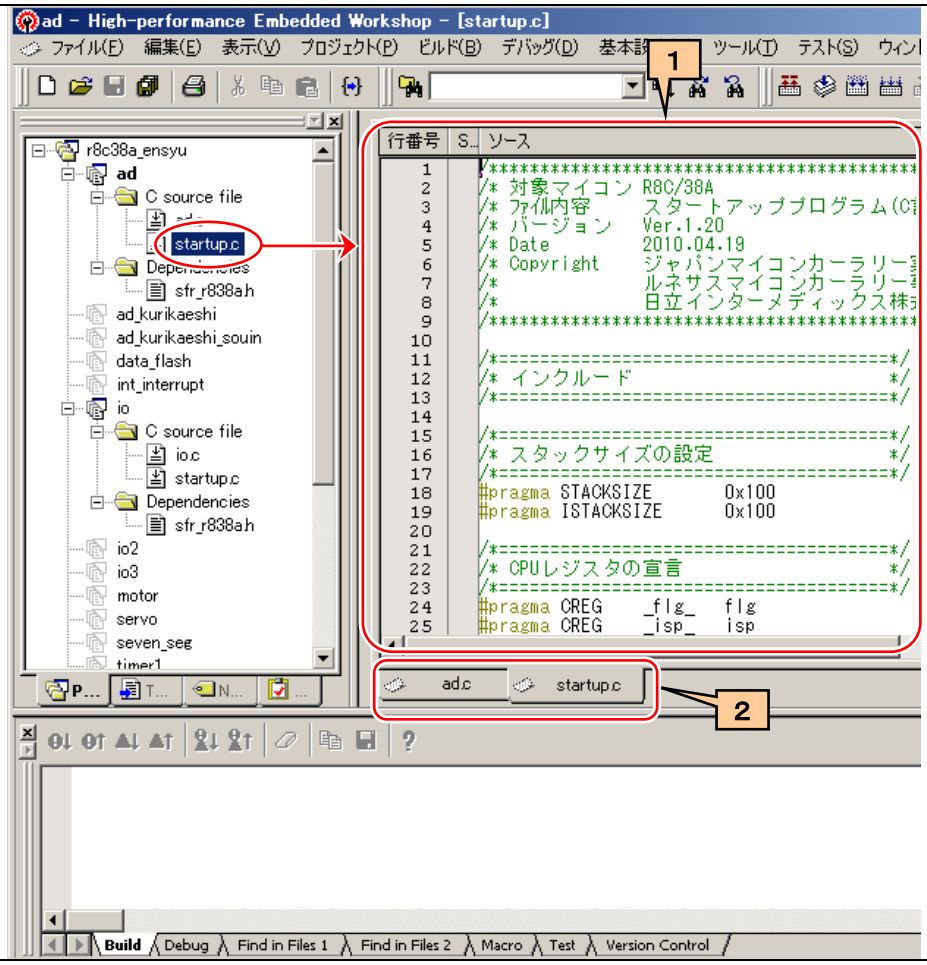
2		上から、 ad，ad_kurikaeshi，ad_kurikaeshi_souin， data_flash, int_interupt，io，io2， motor，servo，seven_seg， timer1，timer2，timer_ra_counter, timer_ra_pulsein，timer_rc_pwm， timer_rd_doukipwm, timer_rd_pwm6， timer_rg_counter，timer_rg_isoukeisu， uart0，uart0_printf というプロジェクトがあります。 **プロジェクト＝演習の単位となります**。 太く表示されているプロジェクトが、有効なプロジェクトで**アクティブプロジェクト**といいます。左画面は「io」がアクティブプロジェクトの状態です。有効なプロジェクトは、 ・ビルドの対象 ・ツールチェイン（各種設定）の対象 ・書き込みの対象 となります。**要は、すべての操作対象となります**。
3		これから、プロジェクト「ad」の実習をするので、このプロジェクトを有効にします。 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。
4		「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。

5		「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。 ●ad.c ●startup.c 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_r838a.h」が登録されています。

8.3 ファイル編集

1		「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。
2		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。

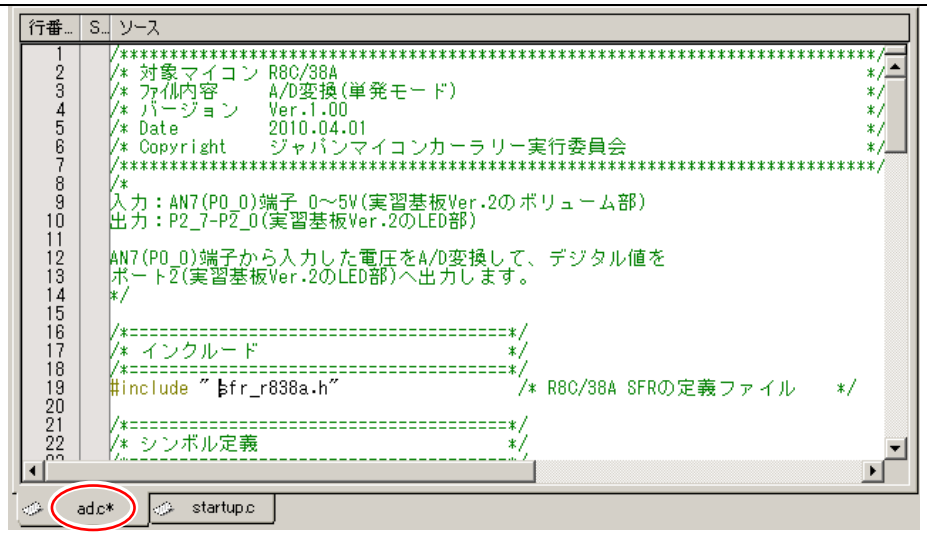
3



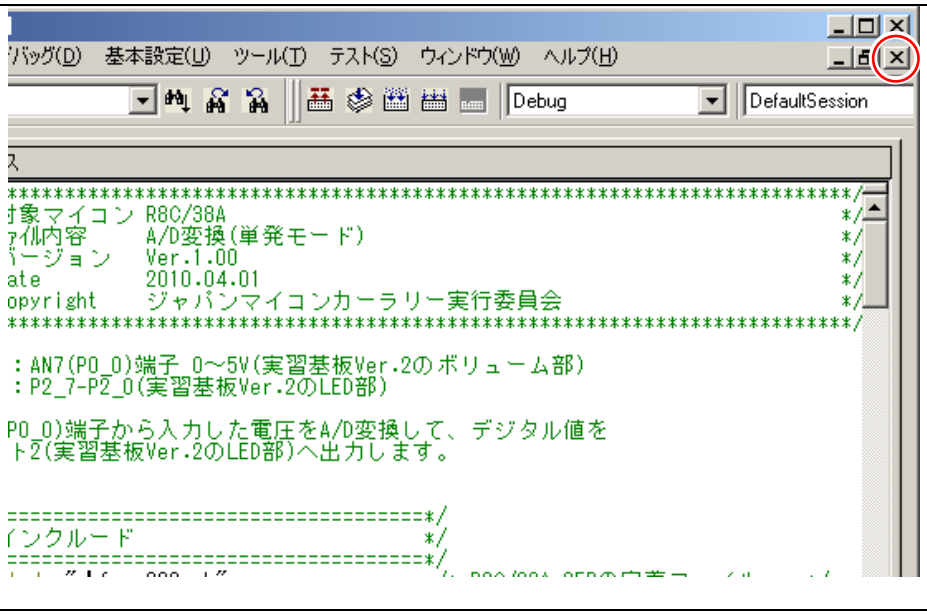
1…「startup.c」をダブルクリックすると、startup.c のエディタウィンドウが開きます。

2…2 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。

4

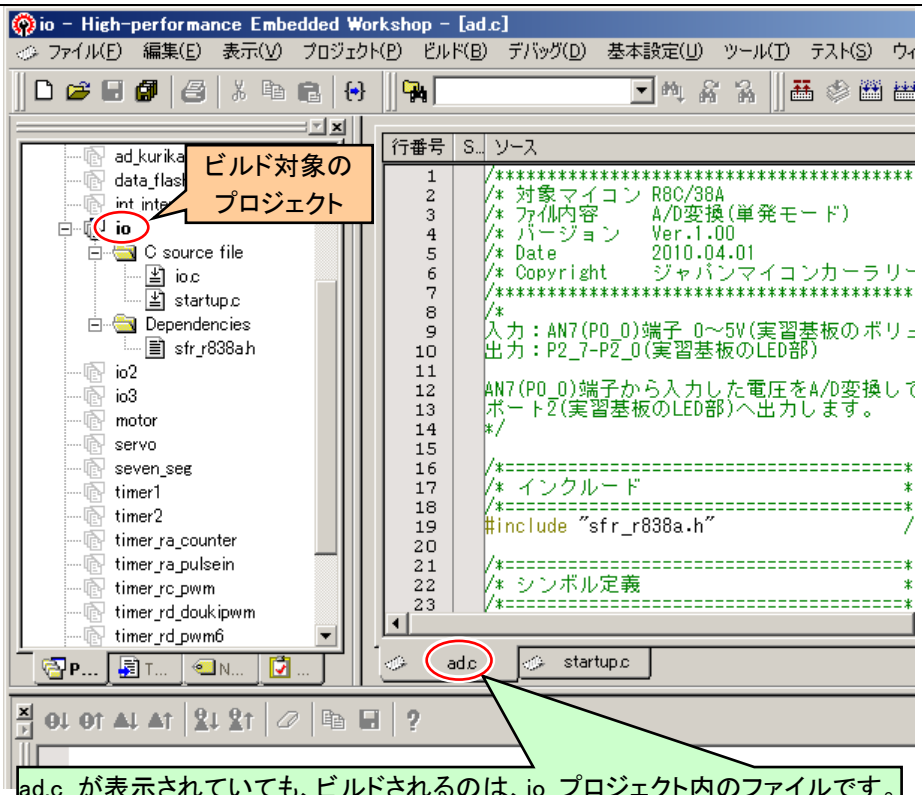


ファイルを編集して内容が変更されると、エディタウィンドウのタブのファイル名の右に「＊」が表示されます。

5		変更した C ファイルは、適宜保存します。保存ボタンは次の 2 つあります。 1…現在編集中のファイルを保存するボタンです。 2…変更したすべてのファイルを保存するボタンです。 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。
6		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。

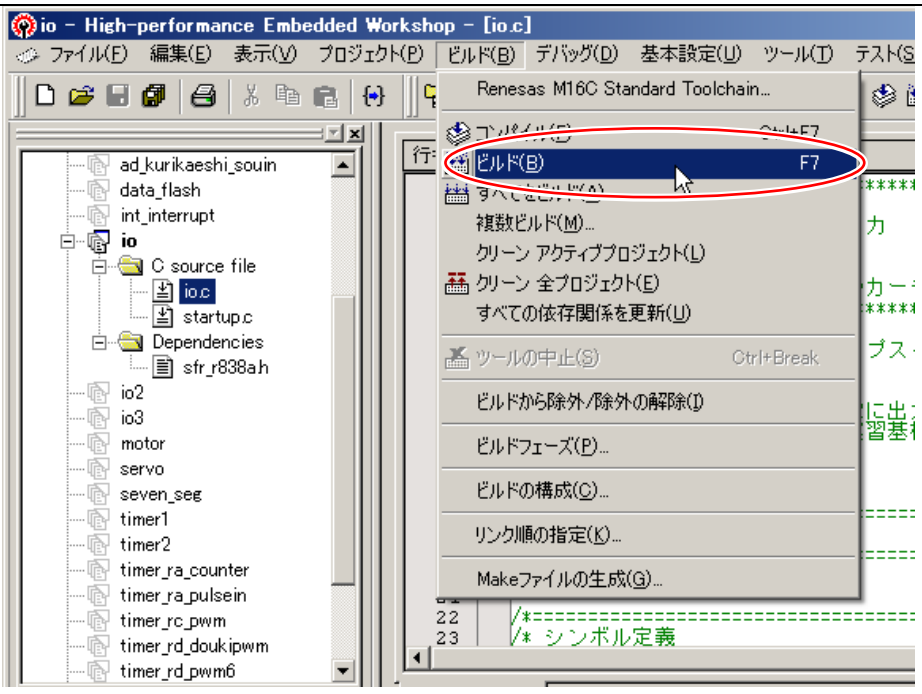
8.4 ビルド（MOT ファイルの作成）

1



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 ファイルを作ることです。
ビルドは、現在有効なプロジェクト内のファイルが対象となります。必ず有効なプロジェクトを確認してください。エディタウィ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、**エディタウィンドウに 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 プロジェクトに関係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。**

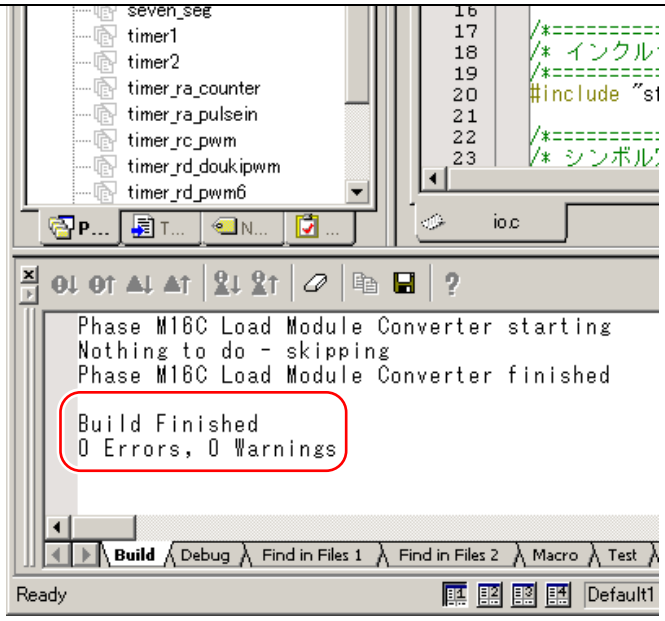
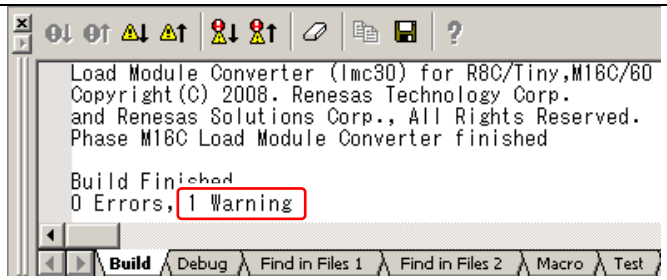
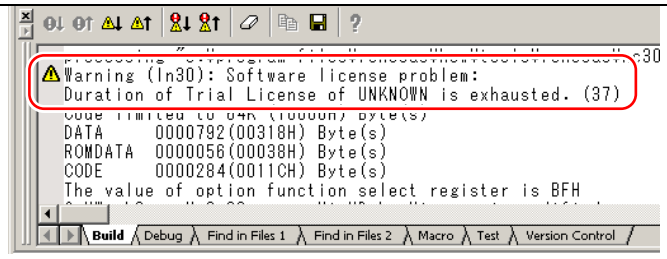
2



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。

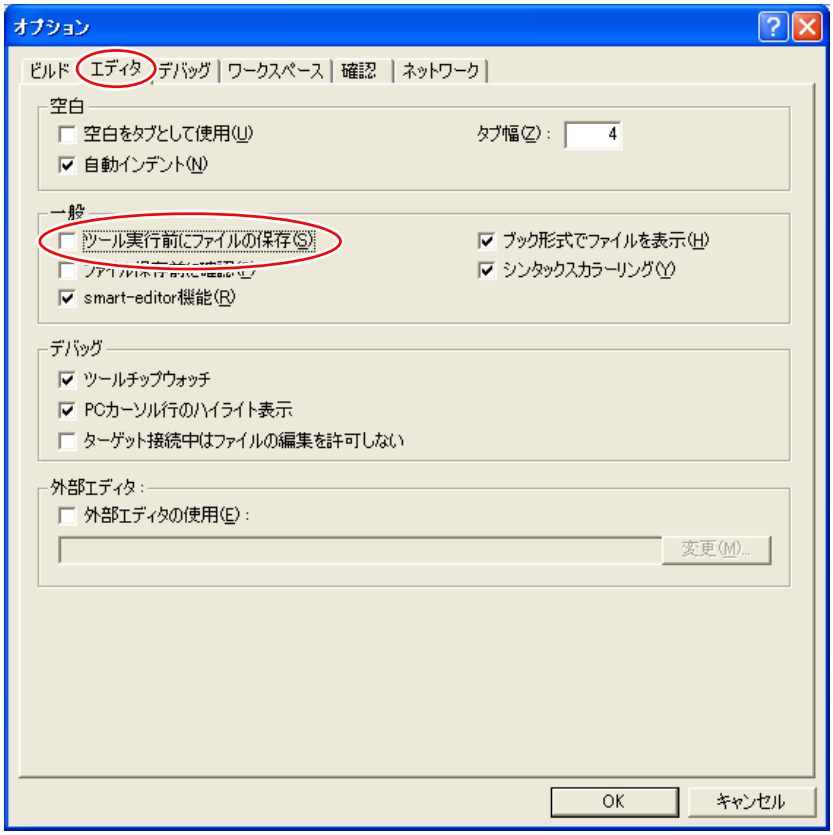
通常は、「ビルド」で問題ありません。

3		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。 **●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インの設定を直します。 **●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。 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。
4		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。
5		ワーニングメッセージを確認しておきます。 Warning (ln30): Software license problem: 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。**

※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。

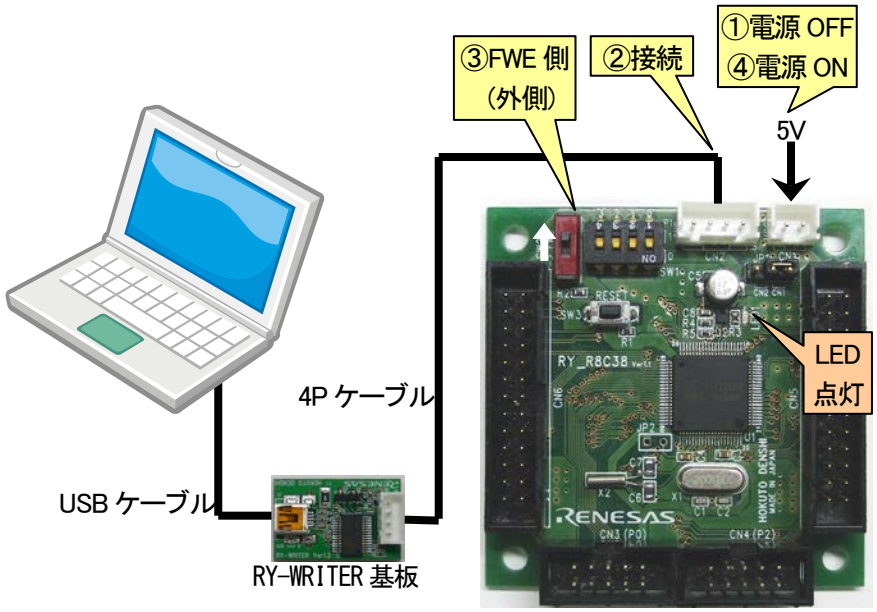
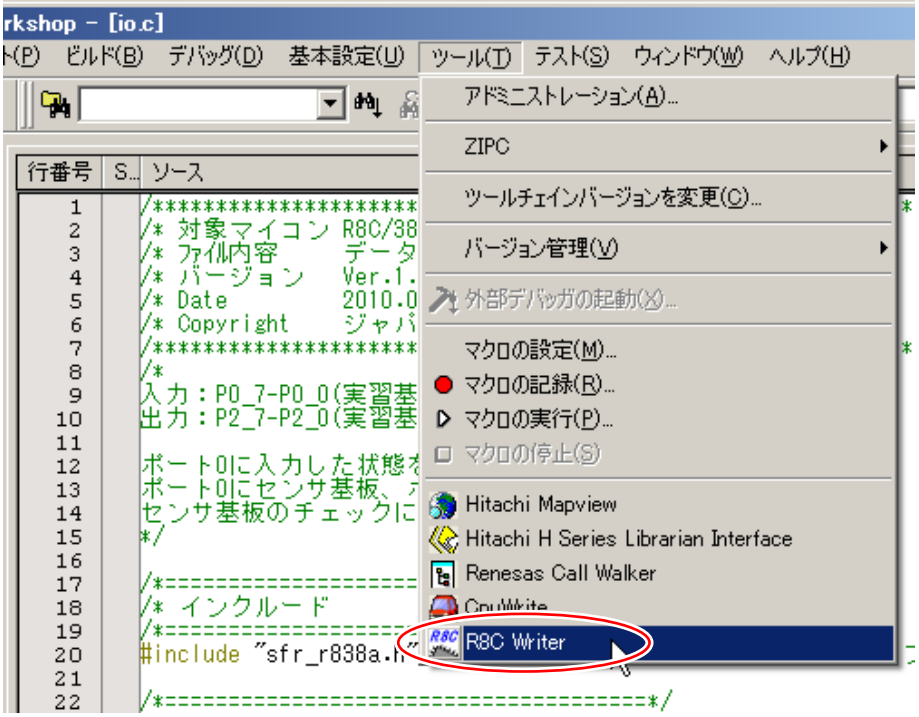
もし、自動保存をしたくない場合は、「基本設定→オプション」でオプション画面を開きます。「エディタ」タブを選び、「一般：ツール実行前にファイルの保存」のチェックを外します。

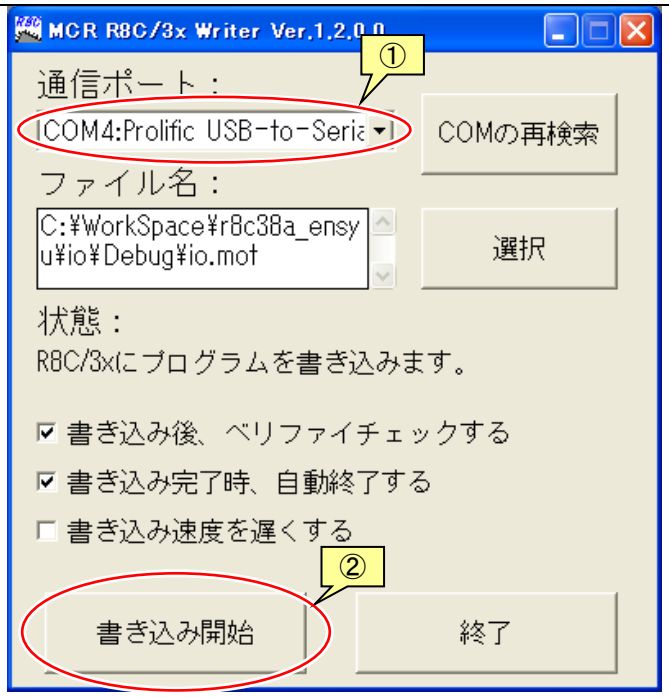
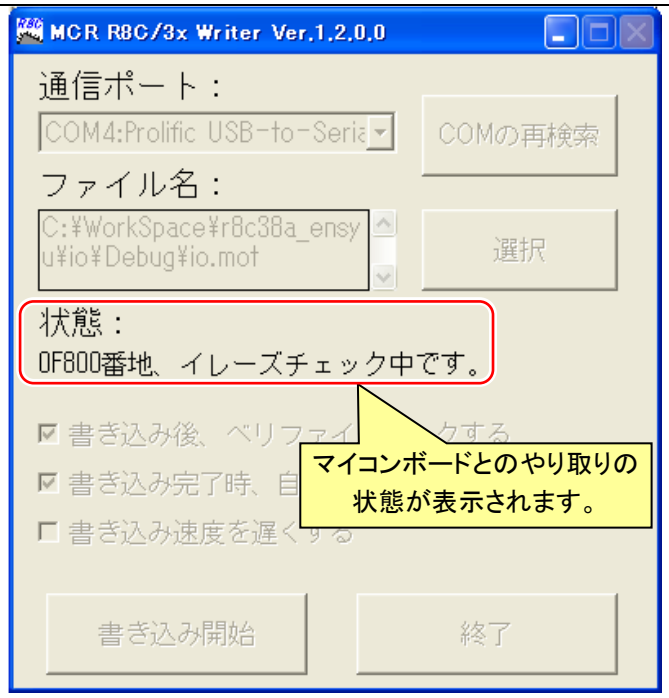
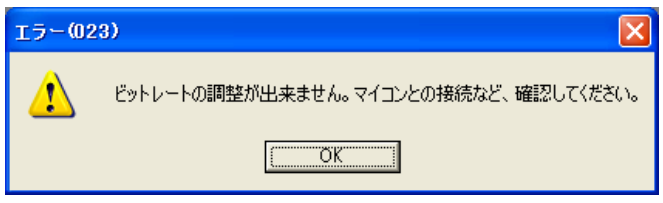


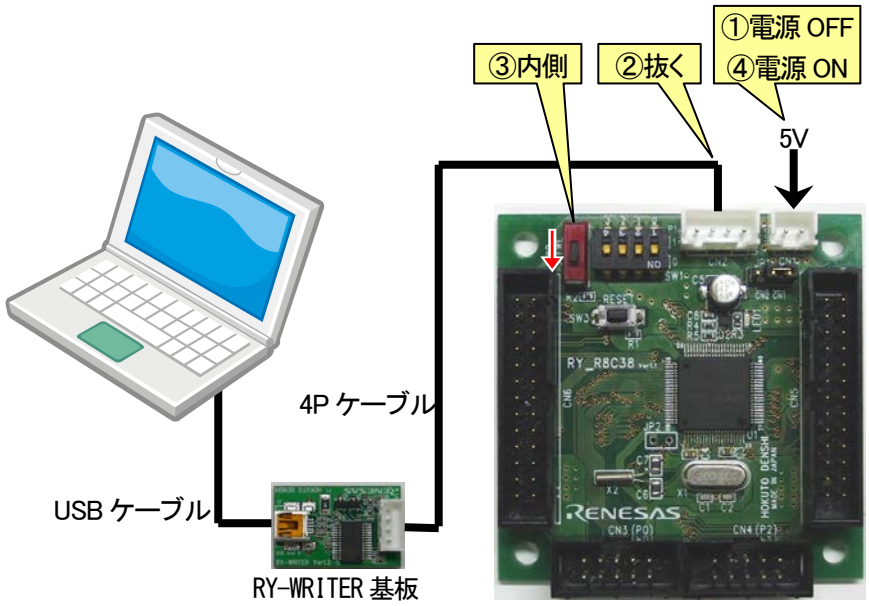
この機能は、複数のファイルを編集中、誤ってビルドボタンを押して、勝手に保存されないようにするときにチェックを外しておくと便利です。

8.5　プログラムの書き込み

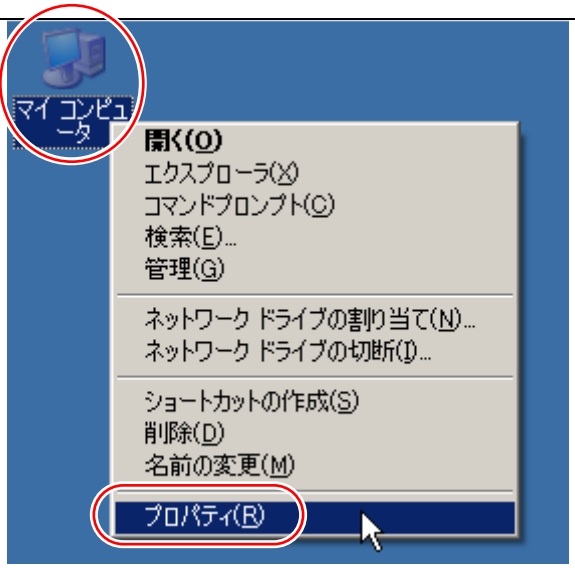
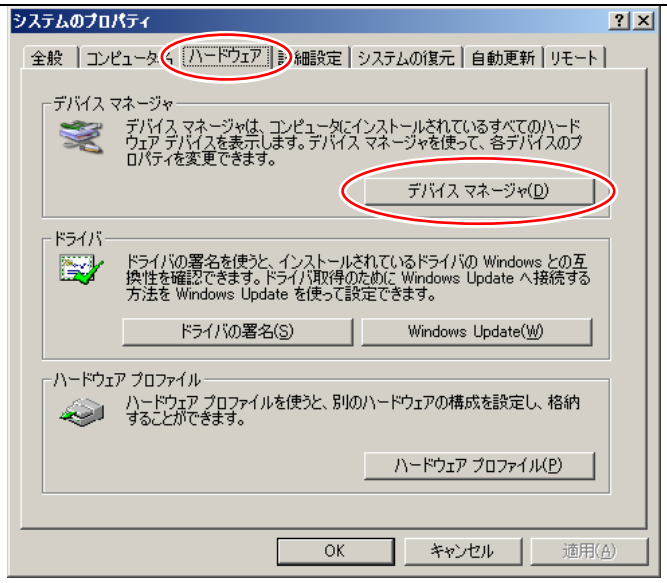
※RY_R8C38 ボードの JP1 は、2-3 ピン間ショート(CN1 から電源供給)されているものとして説明します。

1		①電源は切っておきます。 ②RY-WRITER 基板と RY_R8C38 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で構いません)。 ③SW2 を FWE 側(外側)にします。**これが書き込みモードです。SW2 は電源 OFF の状態で操作します。** ④電源を入れます。 このとき、RY_R8C38 ボードにある LED が点灯します。
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。

3		書き込みソフトが起動します。 ①「通信ポート」の番号を RY-WRITER 基板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。 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。 ※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 を推奨します。 ※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。
4		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場合は、自動で終了します。
5		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。

6		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。 ①電源を OFF にします。 ②USB 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（マイコンカーを走らせる場合はもちろん抜きます）。 ③SW2 を内側にします。**これがプログラム実行モードです。SW2 は電源 OFF の状態で操作します。** ④電源を ON すると、書き込んだプログラムが実行されます。

※通信ポートが分からない場合

1		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。 ※Windows＋Pause キーでもシステムのプロパティ画面を開けます。
2		「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。
3		「ポート（COM と LPT）」をクリックすると、 Prolific USB-to-Serial Comm Port（COM●） があります。●部分が通信ポートの番号です。

9. 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「r8c38a_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	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	ad_pwm	A/D コンバータ（繰り返しモード 0）、タイマ RD	A/D コンバータ（繰り返しモード 0）を使用して 0～5V を読み込み、電圧によってモータの回転速度を調整します。
5	ad_uart0_printf	A/D コンバータ（繰り返しモード 0）、タイマ RB,UART0	A/D コンバータ（繰り返しモード 0）を使用して 0～5V を読み込み、電圧を RS232C 経由（RY_RWITER 基板経由）でパソコンに送信します。
6	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呼び出します。
7	hoco_io		マイコン内蔵の高速オンチップオシレータを使って動作させます。外付けの XIN クロックを使用しません。
8	int_interrupt	$\overline{\text{INT3}}$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。
9	io		ポートの入出力を行います。制御の基本中の基本です。
10	io2		RY_R8C38 ボード上のディップスイッチ（4bit）と LED(1bit)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。
11	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。
12	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。
13	seven_seg	タイマ RB	7 セグメント LED2 桁をダイナミック点灯します。
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 を使って、外部から入力されたパルスの数を数えます。
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_counter	タイマ RD	タイマ RD を使って、外部から入力されたパルスの数を数えます。
20	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。
21	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。
22	timer_re_rtc	タイマ RE	タイマ RE を使用して、リアルタイムクロックで時計機能を動作させます。
23	timer_re_ timerwarikomi	タイマ RE	タイマ RE を使用して、約 1ms ごとに割り込みを発生させます。動作は「timer2」プロジェクトと同じです。
24	timer_rg_counter	タイマ RG	タイマ RG を使って、外部から入力されたパルスの数を数えます。
25	timer_rg_isoukeisu	タイマ RG	タイマ RG を使って、2 相(A 相、B 相)のロータリエンコーダのパルス数を数えます。
26	uart0	UART0	UART0 を使用して、パソコンの RS232C 経由で通信します。
27	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。

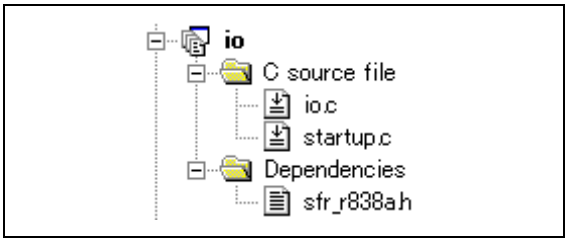
10.　電源が入ってからのマイコンの動作

10.1　概要

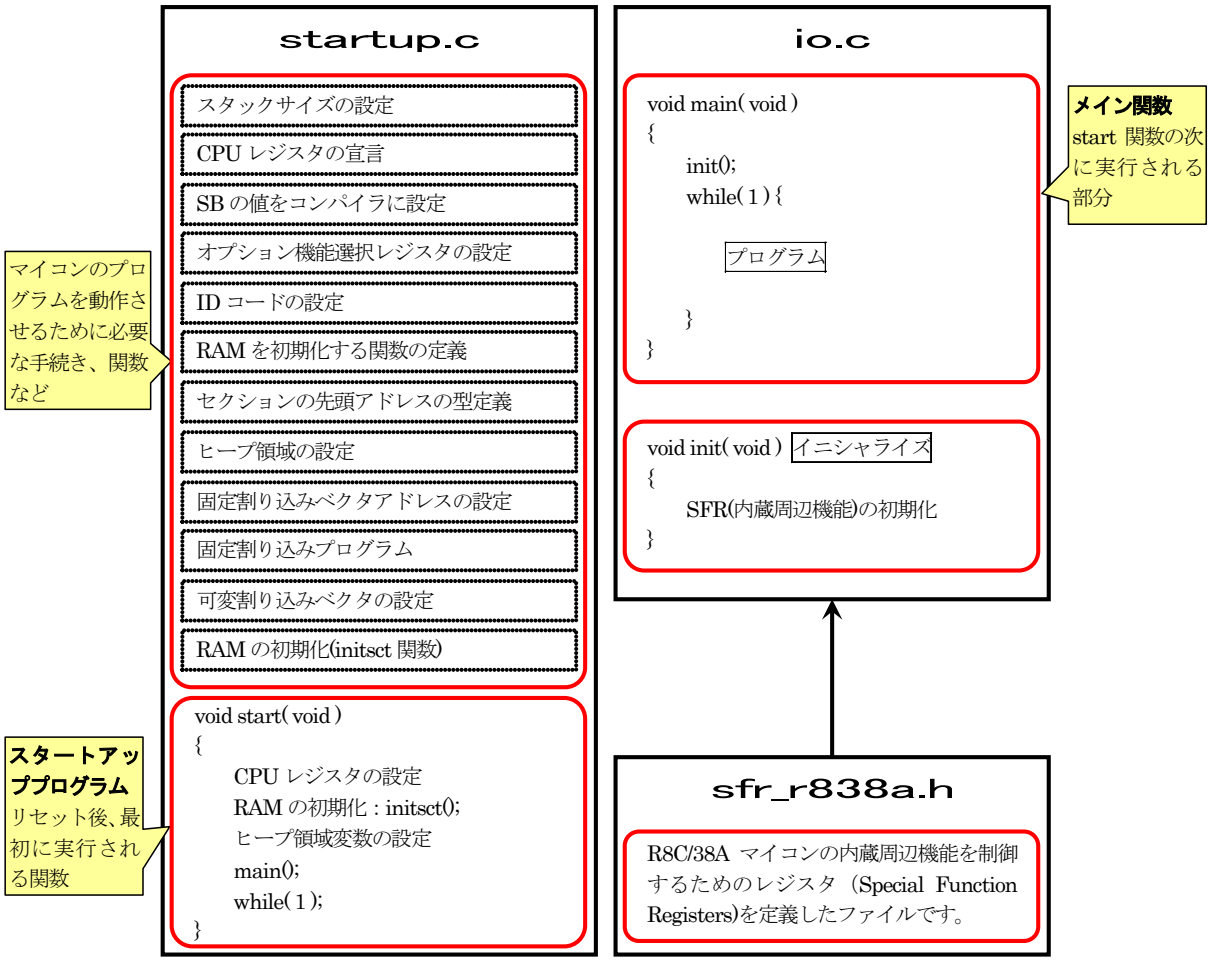
RY_R8C38 ボードの電源が入ってから、どのようにマイコンが動作するのかを説明します。また、これから実習するプロジェクトのファイル構成、実行順について説明します。

10.2　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_r838a.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。



10.3 動作手順

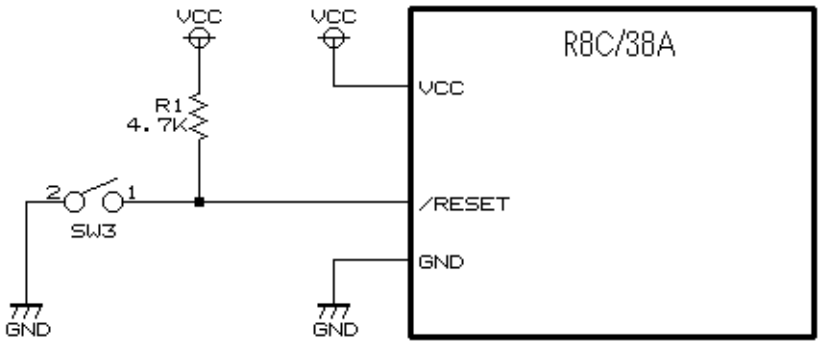
10.3.1 リセットの種類

R8C/38A 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

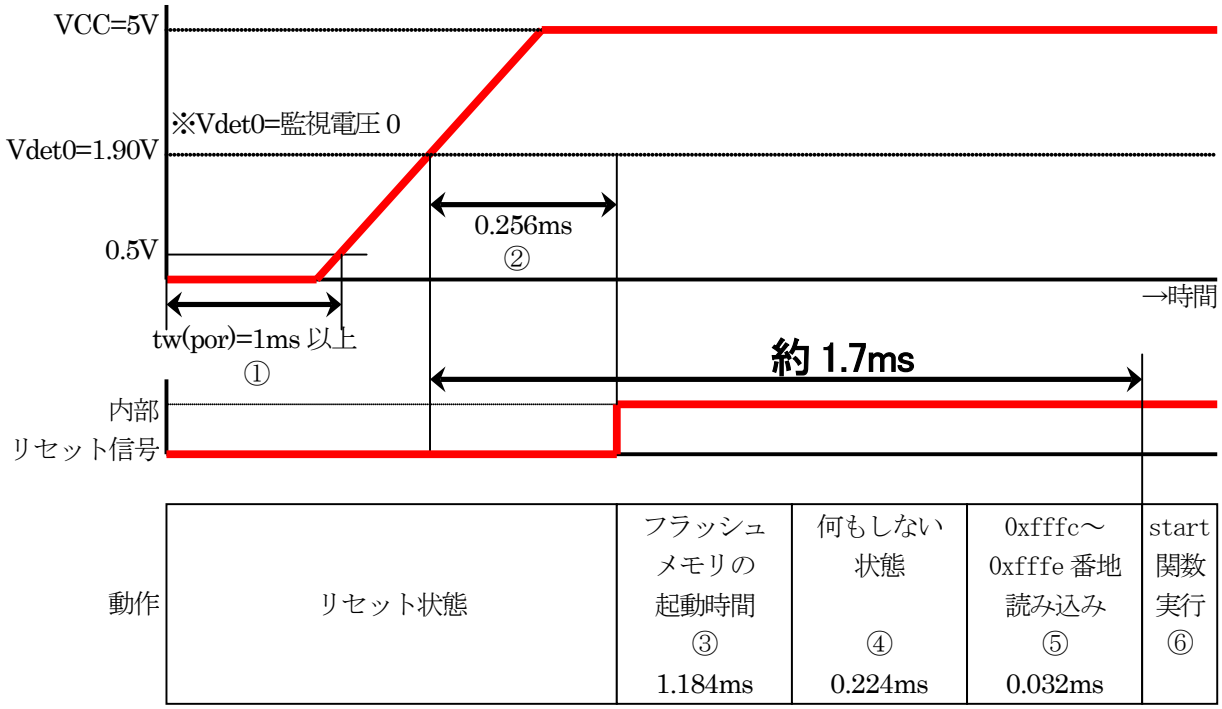
$\overline{\text{RESET}}$端子に抵抗を介して VCC に接続すると、パワーオンリセット機能が有効になります。RY_R8C38 ボードは、4.7kΩの抵抗を介して VCC に接続されており、電源を投入するとパワーオンリセットで立ち上がります。リセット端子の回路を下記に示します。

押しボタンスイッチ SW3 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。SW3 を離すと、パワーオンリセットで立ち上がります。



10.3.2 電源投入してからプログラム実行までの動作

RY_R8C38 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×32 = 1/(125×10^3)×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×148 = 1/(125×10^3)×148 = 1.184ms
④	何もしない状態は、下記の時間です。 1/fOCO-S×28 = 1/(125×10^3)×28 = 0.224ms
⑤	0x0fffc～0x0fffe 番地の読み込みは、下記の時間です。 1/fOCO-S×4 = 1/(125×10^3)×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いておきます。例えば、0x40dc 番地から実行したければ、 0x0fffc 番地＝0xdc 0x0fffd 番地＝0x40 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。

10.4 プログラム「startup.c」

```
/************************************************************************/
/* 対象マイコン R8C/38A                                                 */
/* ﾌｧｲﾙ内容     スタートアッププログラム(C言語版)                       */
/* バージョン   Ver.1.22                                                */
/* Date         2012.03.08                                              */
/* Copyright    ジャパンマイコンカーラリー実行委員会                    */
/*              ルネサスマイコンカーラリー事務局                        */
/*              日立インターメディックス株式会社                        */
/************************************************************************/
/*======================================*/
/* プロトタイプ宣言(ローカル)           */
/*======================================*/
void main( void );

/*======================================*/
/* スタックサイズの設定                 */
/*======================================*/
#pragma STACKSIZE       0x200
#pragma ISTACKSIZE      0x100

/*======================================*/
/* CPUレジスタの宣言                    */
/*======================================*/
#pragma CREG    _flg_   flg
#pragma CREG    _isp_   isp
#pragma CREG    _sp_    sp
#pragma CREG    _sb_    sb
#pragma CREG    _fb_    fb
#pragma CREG    _intbh_ intbh
#pragma CREG    _intbl_ intbl

unsigned int _flg_;
unsigned int _sb_;
unsigned int _fb_;
unsigned int *_sp_;
unsigned int *_isp_;
unsigned int *_intbh_;
unsigned int *_intbl_;

/*======================================*/
/* SBの値をコンパイラに設定             */
/*======================================*/
    _asm("       .glb    __SB__¥n"
         "__SB__ .equ    0400H   ");

/*======================================*/
/* オプション機能選択レジスタの設定     */
/*======================================*/
    _asm("  .ofsreg 0BFH ");            /* OFS = 0xbf (ﾊﾟﾜｰｵﾝﾘｾｯﾄ使用)  */

/*======================================*/
/* IDコードの設定                       */
/*======================================*/
    _asm("  .id ""¥"#FFFFFFFFFFFFFF¥" ");

/*======================================*/
/* RAMを初期化する関数の定義            */
/*======================================*/
#define scopy(X,Y,Z)    _asm("  .initsct "X","Y","Z"¥n"¥
                "   .initsct "X"I,rom"Y",noalign¥n"¥
                "   mov.w   #(topof "X"I)&0ffffH,A0¥n"¥
                "   mov.b   #(topof "X"I)>>16,R1H¥n"¥
                "   mov.w   #(topof "X")&0ffffH,A1¥n"¥
                "   mov.w   #sizeof "X",R3¥n"¥
                "   smovf.b")

#define sclear(X,Y,Z)   _asm("  .initsct "X","Y","Z"¥n"¥
                "   mov.b   #00H,R0L¥n"¥
                "   mov.w   #(topof "X") ,A1¥n"¥
                "   mov.w   #sizeof "X",R3¥n"¥
                "   sstr.b")

/*======================================*/
/* セクションの先頭アドレスの型定義     */
/*======================================*/
extern unsigned int _stack_top;
extern unsigned int _istack_top;
extern unsigned int _vector_top;
```

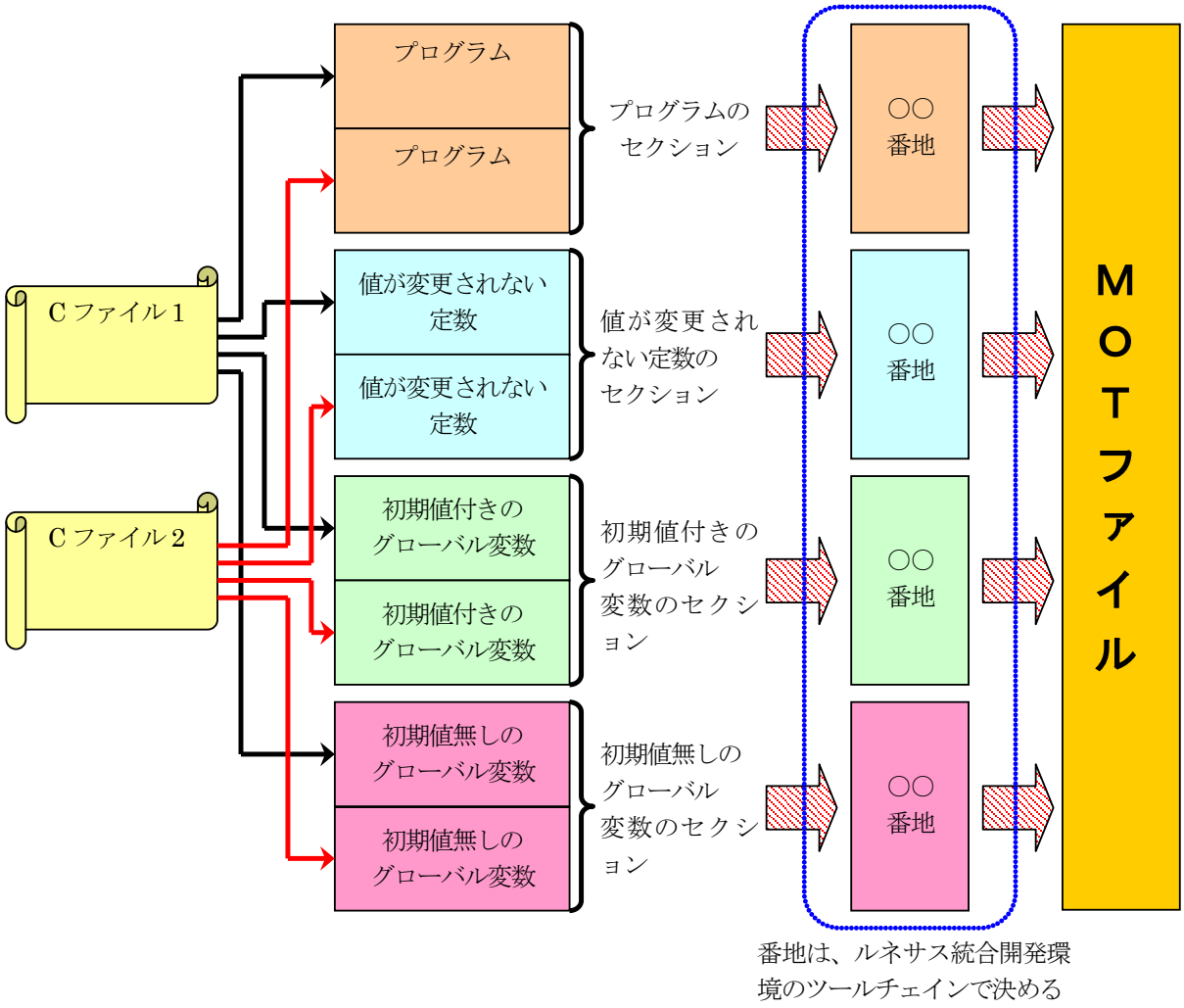
```
 80 :  /*====================================*/
 81 :  /* ヒープ領域の設定                   */
 82 :  /*====================================*/
 83 :  #pragma sectaddress heap_NE,DATA
 84 :  #define __HEAPSIZE__    0x100
 85 :
 86 :  unsigned char heap_area[ __HEAPSIZE__ ]; /* ヒープ領域確保              */
 87 :  extern unsigned char  _far *_mnext;     /* 次に使用できるﾒﾓﾘの先頭ｱﾄﾞﾚｽ */
 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");               /* 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          /* ｼﾝｸﾞﾙｽﾃｯﾌﾟ割り込みベクタ    */
102 :  #pragma interrupt/v _dummy_int          /* ｳｫｯﾁﾄﾞｯｸﾞﾀｲﾏなどの割り込みﾍﾞｸﾀ*/
103 :  #pragma interrupt/v _dummy_int          /* ｱﾄﾞﾚｽﾌﾞ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 initsct(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ﾄﾞﾚｽ設定 */
148 :      _flg_   = 0x0080;                   /* FLGのU="1"                  */
149 :      _sp_    = &_stack_top;              /* USPにﾕｰｻﾞｽﾀｯｸのｱﾄﾞﾚｽ設定    */
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 :  }
162 :
163 :  /*****************************************************************************/
164 :  /* end of file                                                               */
165 :  /*****************************************************************************/
```

10.5　セクション

10.5.1　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イ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。



10.5.2 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。

data または bss または rom または heap＋＿＋①＋②＋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（ニアー）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、0x10000～0xffffff 番地のアドレスのデータです。
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。

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	RAM
	N near データ	**E** 偶数		bss_NE	RAM
		O 奇数		bss_NO	RAM
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	ROM
		O 奇数		rom_NO	ROM
	F far データ	**E** 偶数		far_FE	ROM
		O 奇数		far_FO	ROM
heap ヒープ領域 (RAM)	**N** near データ	**E** 偶数		heap_NE	RAM

アドレスが 0x10000 番地以上の場合は、far データになります。

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「**10.6.6 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。

10.5.3 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_r838a.h"                 /* R8C/38A SFRの定義ファイル     */

 #pragma sbdata c4                      /* SBDATA指定                    */
 #pragma sbdata 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。

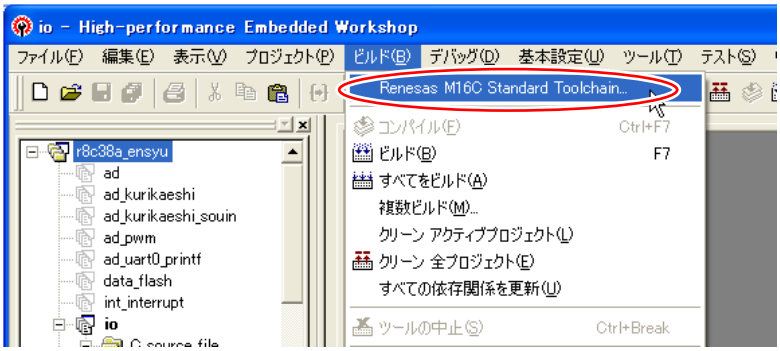
プログラム	配置させるセクション	説明
①const char c1 = 0x00;	rom_NO	const は値を変更できない変数なので、ROM に配置します。char 型は 1 バイト幅なので奇数アドレスで格納します。
②const int i1 = 0x1234;	rom_NE	const は値を変更できない変数なので、ROM に配置します。int 型は 2 バイト幅なので偶数アドレスで格納します。

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。
⑤unsigned char c3;	bss_NO	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。
⑥unsigned int i3;	bss_NE	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。
⑦unsigned char c4;	bss_SO	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。
⑧unsigned int i4 = 1;	data_SEI , data_SE	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。

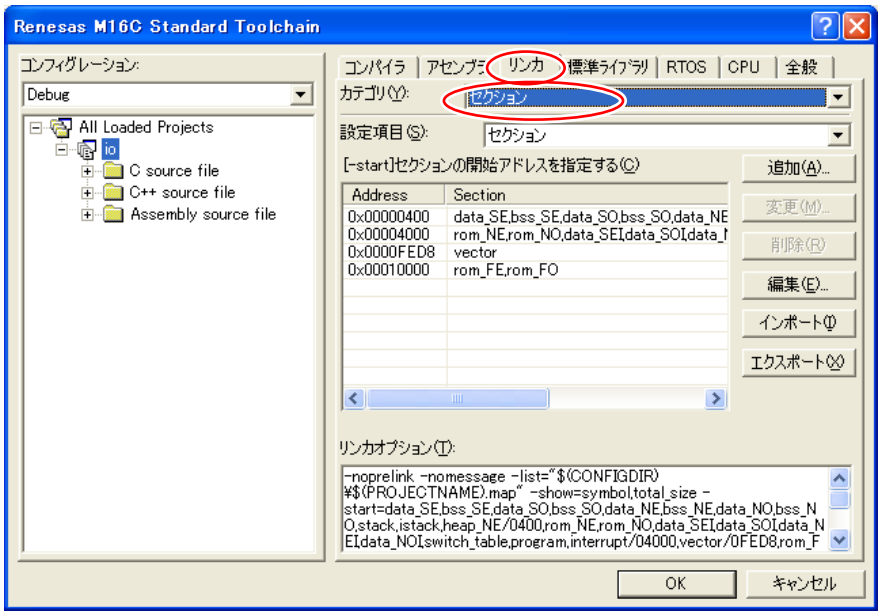
⑨void main(void)
{
prc2 = i4;
pd0 = c1;
pd6 = c2;
 	
while(1) {
c4 = p0;
p6 = c4;
}
}
	program	プログラムです。

10.5.4 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インで設定します。ルネサス統合開発環境の「ビルド→Renesas M16C Standard Toolchain」(ツールチェイン)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インのアドレスと、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、ツールチェインの設定で行ってください。

アドレス	セクション名
0x0400 (RAM)	data_SE
	bss_SE
	data_SO
	bss_SO
	data_NE
	bss_NE
	data_NO
	bss_NO
	stack
	istack
	heap_NE

アドレス	セクション名
0x4000 (ROM)	rom_NE
	rom_NO
	data_SEI
	data_SOI
	data_NEI
	data_NOI
	switch_table
	program
	interrupt
0xfed8 (ROM)	vector
0xffd8 (ROM)	fvector

アドレス	セクション名
0x10000 (ROM)	rom_FE
	rom_FO

※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。

10.5.5 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
```

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。 • 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インの設定では、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インの設定では、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インの設定では、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インで設定するので、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イ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イ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イ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。

10.6 プログラムの解説「startup.c」

10.6.1 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        0x200
19 :  #pragma ISTACKSIZE       0x100
```

18 行	「#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、0x200 個(512 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。 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。 .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
19 行	「#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。 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。 .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

10.6.2 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	「#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。 「#pragma CREG」命令の使い方は下記のとおりです。 `#pragma CREG` C 言語プログラムでの変数名 CPU レジスタ名
32 行～ 38 行	定義した変数(CPU レジスタ)の型を宣言します(※)。

※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 *

10.6.3 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。

10.6.4 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 (ﾊﾟﾜｰｵﾝﾘｾｯﾄ使用)  */
```

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)の 0xffff 番地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" );
```

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使いません。	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。

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
```

10.6.5 ID コードの設定

ID コードを設定します。

```
/*=====================================*/
/* IDコードの設定                      */
/*=====================================*/
    _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
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○○を書き込んで、ID2(0xffe3 番地)に○○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、下記のようになります。

```
_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）。

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 ¥″ ″);
```

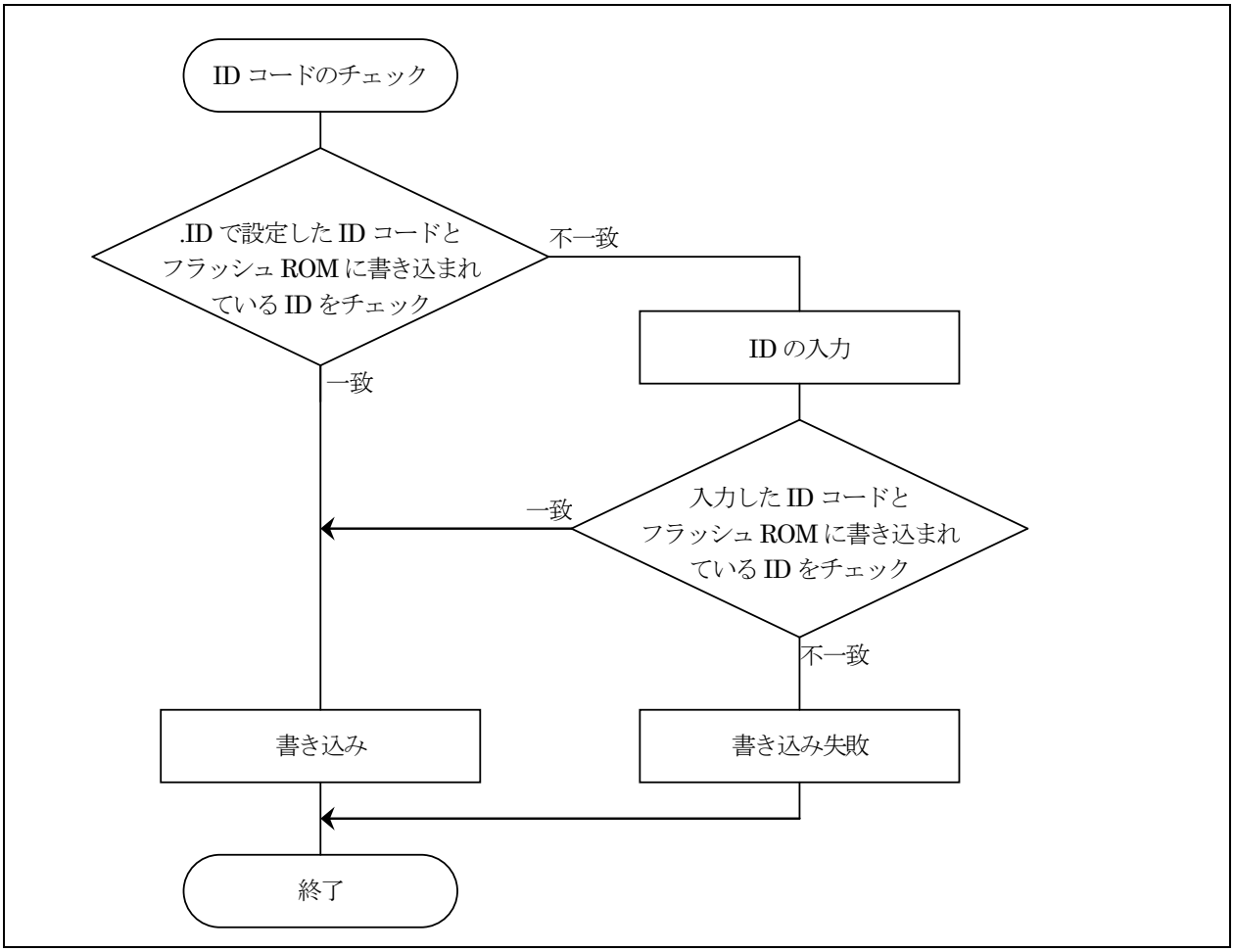
例えば、ID1=0x12、ID2=0x34、ID3=0x56、ID4=0x78、ID5=0x9a、ID6=0xbc、ID7=0xde にしたいとき、下記のように記述します。

```
_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。**

10.6.6 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"I,rom"Y",noalign¥n"¥
 61 :               "   mov.w   #(topof "X"I)&0ffffH,A0¥n"¥
 62 :               "   mov.b   #(topof "X"I)>>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。
初期値無しのグローバル変数	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。

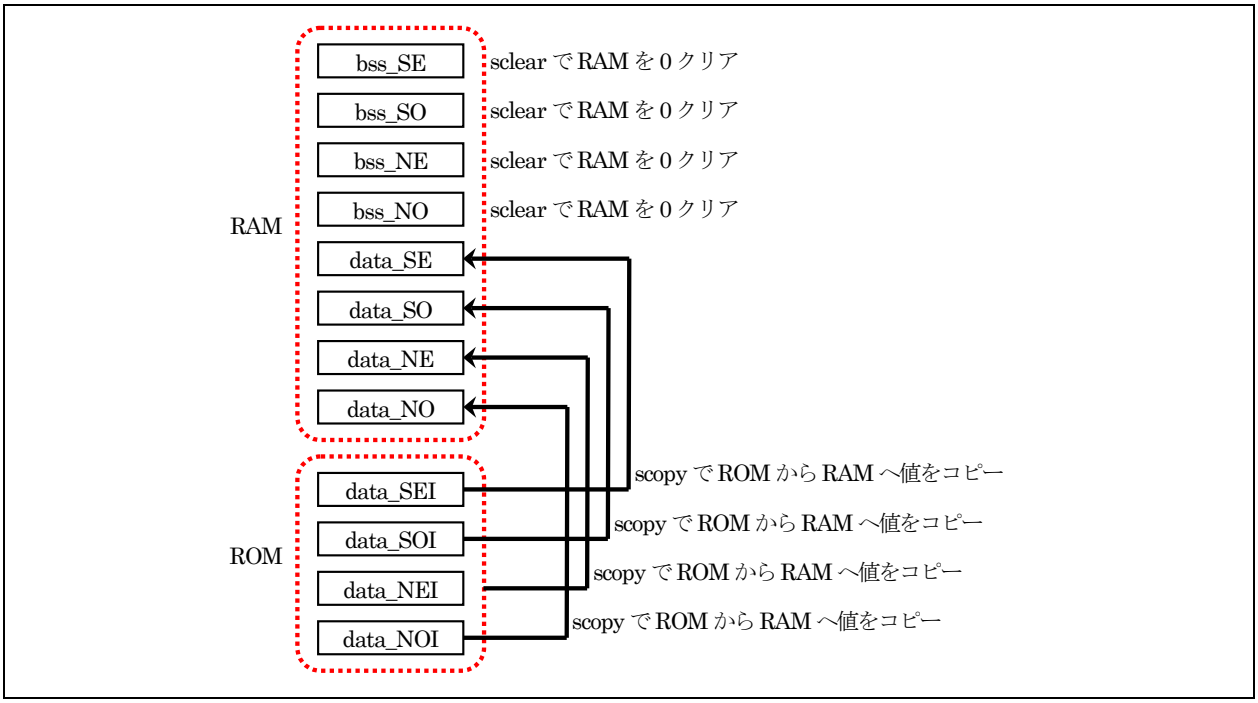
■scopy 関数

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	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。 scopy("data_SE","data","align"); data_SEI→data_SE へコピー scopy("data_SO","data","noalign"); data_SOI→data_SO へコピー scopy("data_NE","data","align"); data_NEI→data_NE へコピー scopy("data_NO","data","noalign"); 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	初期値無しの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	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。 sclear("bss_SE","data","align"); bss_SE を 0 クリア sclear("bss_SO","data","noalign"); bss_SO を 0 クリア sclear("bss_NE","data","align"); bss_NE を 0 クリア sclear("bss_NO","data","noalign"); bss_NO を 0 クリア

グローバル変数の初期化の様子を下記に示します。



10.6.7　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。

```
_+セクション名+_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。

10.6.8 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ﾄﾞﾚｽ */
 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。

```
#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 */

   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なので
                         str[80]は確保されたまま
}
```

```
#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

   続きのプログラム
}
```

10.6.9 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 :                                                          アドレス
 95 :      _asm(" .addr 0FFFFFFFH");   /* 予約                    */  ffd8,ffd9,ffda
 96 :      _asm(" .byte 0FFH");        /* OFS2                    */  ffdb
 97 :  #pragma interrupt/v _dummy_int  /* 未定義命令割り込みベクタ    */  ffdc,ffdd,ffde,ffdf
 98 :  #pragma interrupt/v _dummy_int  /* オーバフロー割り込みベクタ  */  ffe0,ffe1,ffe2,ffe3
 99 :  #pragma interrupt/v _dummy_int  /* BRK命令割り込みベクタ       */  ffe4,ffe5,ffe6,ffe7
100 :  #pragma interrupt/v _dummy_int  /* アドレス一致割り込みベクタ  */  ffe8,ffe9,ffea,ffeb
101 :  #pragma interrupt/v _dummy_int  /* ｼﾝｸﾞﾙｽﾃｯﾌﾟ割り込みベクタ    */  ffec,ffed,ffee,ffef
102 :  #pragma interrupt/v _dummy_int  /* ｳｫｯﾁﾄﾞｯｸﾞﾀｲﾏなどの割り込みﾍﾞｸﾀ*/  fff0,fff1,fff2,fff3
103 :  #pragma interrupt/v _dummy_int  /* ｱﾄﾞﾚｽﾌﾞﾚｲｸ割り込みベクタ    */  fff4,fff5,fff6,fff7
104 :  #pragma interrupt/v _dummy_int  /* 予約                    */  fff8,fff9,fffa,fffb
105 :  #pragma interrupt/v start      /* リセットベクタ              */  fffc,fffd,fffe,ffff
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 :  }
```

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。

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	0xfff0～0xfff3 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、0xfff3 番地は ID5 の値が上書きされます。
103 行	0xfff4～0xfff7 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、0xfff7 番地は ID6 の値が上書きされます。
104 行	0xfff8～0xfffb 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、現在は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せんが、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、0xfffb 番地は ID7 の値が上書きされます。
105 行	0xfffc～0xffff 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、0xffff 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 0x4020 番地の場合、 0x20, 0x40, 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。 • 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	#pragma interrupt _dummy_int()
	void _dummy_int(void)
	{
	　割り込みプログラム
	}
	➡
	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
	}

■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。 • 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	#pragma interrupt intTRB(vect=24)
	void intTRB(void)
	{
	　割り込みプログラム
	}
	➡
	.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
	}

■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	●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	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。 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。

使用例	#pragma interrupt /B intTRB(vect=24)
	void intTRB(void)
	{
	　割り込みプログラム
	}
	➡
	.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
	}

■多重割り込みを許可する割り込み関数の指定

書式	●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	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。

使用例	#pragma interrupt /E intTRB(vect=24)
	void intTRB(void)
	{
	　割り込みプログラム
	}
	➡
	.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
	}

10.6.10 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 割り込み処理関数名(vect=ベクタ番号)」)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。

10.6.11 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ﾄﾞﾚｽ設定 */
148 :      _flg_   = 0x0080;                /* FLGのU="1"                */
149 :      _sp_    = &_stack_top;           /* USPにﾕｰｻﾞｽﾀｯｸのｱﾄﾞﾚｽ設定   */
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 関数名 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。

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、初期値無しのグローバル変数を設定する initsct 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*_mnext 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_msize 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	main 関数を実行します。
160 行	main 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 start 関数が呼ばれた部分へ戻ります。といっても、start 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。

10.6.12 SB 相対アドレッシング

R8C/38A 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）

```
unsigned long cnt_rb;

void timer( unsigned long timer_set )
{
    cnt_rb = 0;
    while( cnt_rb < timer_set );
}
```

SB 相対アドレッシングあり（C言語）

```
#pragma SBDATA cnt_rb
unsigned long cnt_rb;

void timer( unsigned long timer_set )
{
    cnt_rb = 0;
    while( cnt_rb < timer_set );
}
```

SB 相対アドレッシング無し（アセンブリ言語）

```
←マシン語→ ←ア セ ン ブ リ 言語→
            _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

29 バイト

SB 相対アドレッシングあり（アセンブリ言語）

```
←マシン語→ ←ア セ ン ブ リ 言語→
            _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

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）
・ツールチェインで、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。

10.6.13　switch_table セクション

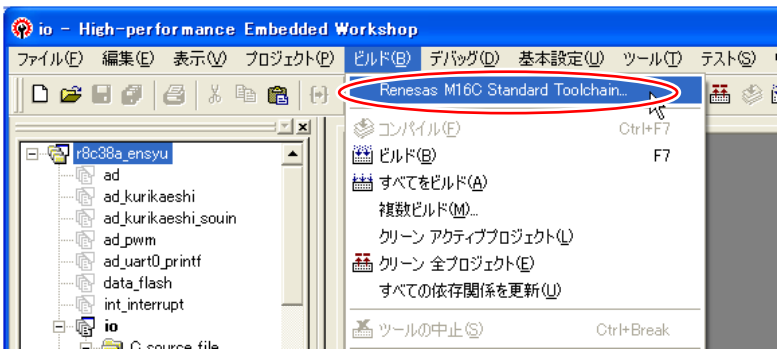
swtich-case 文を使うとき、分岐の方法は次の 2 つの方法があります。

・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

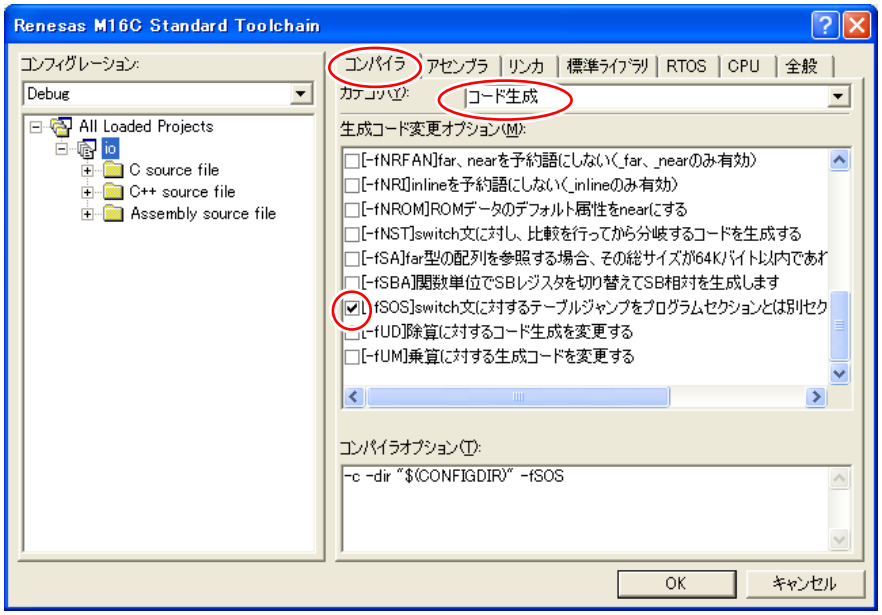
どちらにするかは、プログラムサイズや実行スピードを判定して、ルネサス統合開発環境が自動的に決めます。
ジャンプテーブルを作るとき、このジャンプテーブルを「switch_talbe」セクションに割り当てるか、「program」セクションにプログラムと一緒に割り当てるかを設定することができます。
例えば、プログラムサイズが ROM 領域(0x4000～0x23fff 番地)の 128KB を超えた場合、ジャンプテーブルをデータフラッシュ領域（0x3000～0x3fff 番地)に置くなどすることができます。

■ジャンプテーブルを「switch_table」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」（ツールチェイン）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

```
void main( void )
{
    unsigned char d;

    init();

    while( 1 ) {
        d = p0;
        switch( d ) {
            case 0:
                p6 = 0x01;
                break;
```

（次ページへ続く）

アセンブリソースファイル

```
_main:
    enter   #01H
    jsr _init
L1:
    mov.w   #0001H,R0
    jeq L3
    mov.b   _p0_addr,-1[FB] ;  d
    mov.b   -1[FB],R0L  ;  d
    mov.b   R0L,A0
    cmp.b   #11H,A0     ; case width check
    jnc M1
    mov.w   #10H,A0
M1:
    sha.w   #1,A0
```

（次ページへ続く）

```
            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;
        }
    }
}
```

```
S1:
    jmpi.w  L41[A0]

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:
M3:

    .SECTION    switch_table,ROMDATA,align
L41:
    .words  L7-S1   ; case  0のジャンプ先
    .words  L9-S1   ; case  1のジャンプ先
    .words  L11-S1  ; case  2のジャンプ先
    .words  L13-S1  ; case  3のジャンプ先
    .words  L15-S1  ; case  4のジャンプ先
    .words  L17-S1  ; case  5のジャンプ先
    .words  L19-S1  ; case  6のジャンプ先
    .words  L21-S1  ; case  7のジャンプ先
    .words  L23-S1  ; case  8のジャンプ先
    .words  L25-S1  ; case  9のジャンプ先
    .words  L27-S1  ; case 10のジャンプ先
    .words  L29-S1  ; case 11のジャンプ先
    .words  L31-S1  ; case 12のジャンプ先
    .words  L33-S1  ; case 13のジャンプ先
    .words  L35-S1  ; case 14のジャンプ先
    .words  L37-S1  ; case 15のジャンプ先
    .words  L39-S1  ; defaultのジャンプ先
```

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。

11. I/O ポートの入出力（プロジェクト：io）

11.1 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。

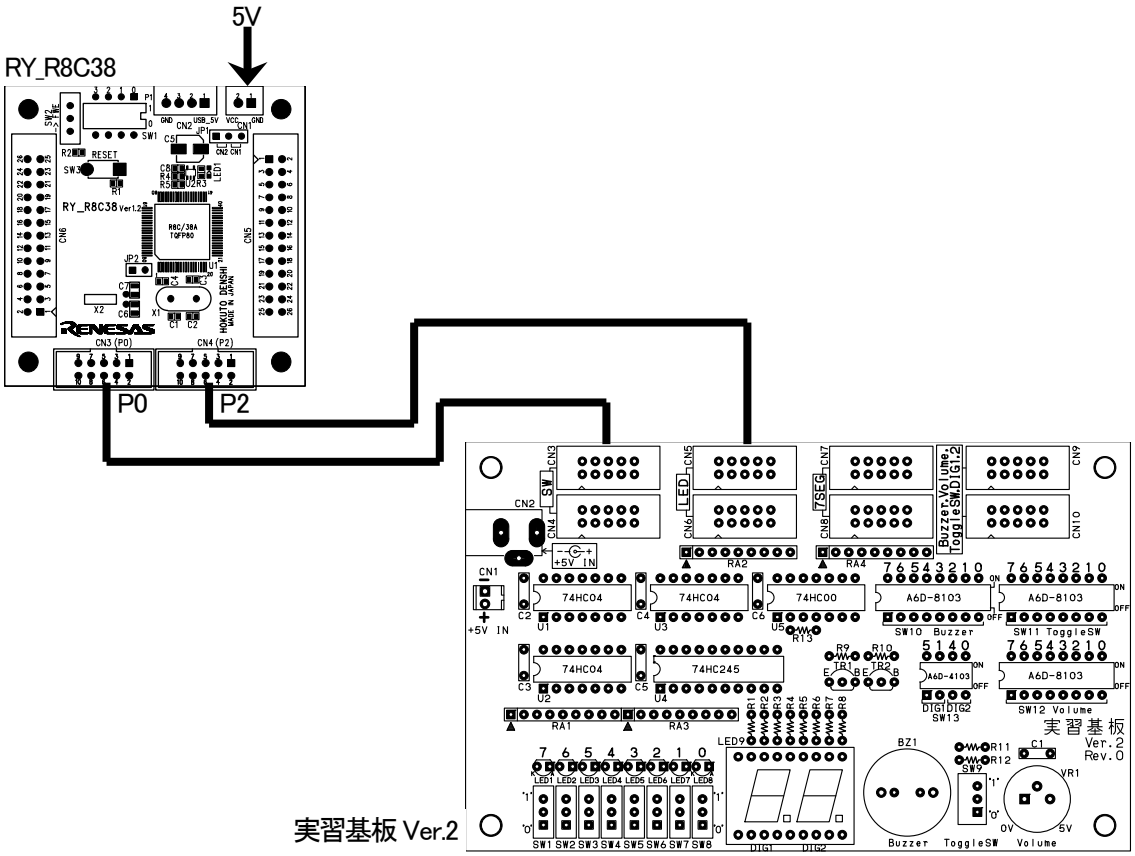
11.2 接続

■使用ポート

マイコンのポート	接続内容
P0 (CN3)	実習基板 Ver.2 のスライドスイッチ部など、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.5 も可能です。
P2 (CN4)	実習基板 Ver.2 の LED 部など、出力機器を接続します。

■接続例

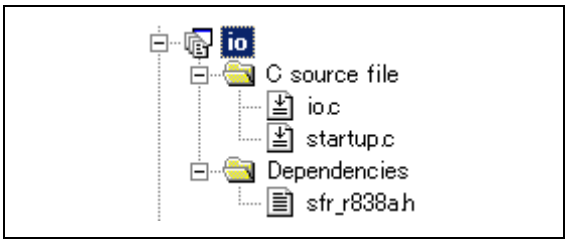
実習基板 Ver.2 を使ったときの接続例を下記に示します。



■操作方法

実習基板 Ver.2 のスライドスイッチ(SW1～8)を ON/OFF すると、それに合わせて LED(LED1～8)が点灯/消灯します。入力したデータを出力する、制御の基本中の基本です。

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。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥io¥startup.c
2	io.c	実際に制御するプログラムが書かれています。R8C/38A の内蔵周辺機能(SFR)の初期化も行います。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥io¥io.c
3	sfr_r838a.h	R8C/38A マイコンの内蔵周辺機能を制御するためのレジスタ（Special Function Registers)を定義したファイルです。 ファイルの位置→C:¥Workspace¥common_r8c38a¥sfr_r838a.h

11.4 プログラム「io.c」

```
 1 :  /***********************************************************************/
 2 :  /* 対象マイコン R8C/38A                                                */
 3 :  /* ﾌｧｲﾙ内容     データの入力、出力                                     */
 4 :  /* バージョン   Ver.1.00                                               */
 5 :  /* Date         2010.04.01                                             */
 6 :  /* Copyright    ジャパンマイコンカーラリー実行委員会                   */
 7 :  /***********************************************************************/
 8 :  /*
 9 :  入力：P0_7-P0_0(実習基板Ver.2のスライドスイッチ部)
10 :  出力：P2_7-P2_0(実習基板Ver.2のLED部)
11 :
12 :  ポート0に入力した状態を、ポート2に出力します。
13 :  ポート0にセンサ基板、ポート2に実習基板Ver.2のLED部分を接続すれば
14 :  センサ基板のチェックになります。
15 :  */
16 :
17 :  /*====================================*/
18 :  /* インクルード                       */
19 :  /*====================================*/
20 :  #include "sfr_r838a.h"                 /* R8C/38A 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 :
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 :          d = p0;
42 :          p2 = d;
43 :      }
44 :  }
45 :
46 :  /************************************************************************/
47 :  /* R8C/38A スペシャルファンクションレジスタ(SFR)の初期化                */
48 :  /************************************************************************/
49 :  void init( void )
50 :  {
51 :      int i;
52 :
53 :      /* クロックをXINクロック(20MHz)に変更 */
54 :      prc0  = 1;                          /* プロテクト解除               */
55 :      cm13  = 1;                          /* P4_6,P4_7をXIN-XOUT端子にする*/
56 :      cm05  = 0;                          /* XINクロック発振              */
57 :      for(i=0; i<50; i++ );               /* 安定するまで少し待つ(約10ms) */
58 :      ocd2  = 0;                          /* システムクロックをXINにする  */
59 :      prc0  = 0;                          /* プロテクトON                 */
60 :
61 :      /* ポートの入出力設定 */
62 :      prc2 = 1;                           /* PD0のプロテクト解除          */
63 :      pd0 = 0x00;                         /* 7-0:スイッチ                 */
64 :      pd1 = 0xd0;                         /* 5:RXD0 4:TXD0 3-0:DIP SW     */
65 :      pd2 = 0xff;                         /* 7-0:LED                      */
66 :      pd3 = 0xff;                         /*                              */
67 :      p4  = 0x20;                         /* P4_5のLED:初期は点灯         */
68 :      pd4 = 0xb8;                         /* 7:XOUT 6:XIN 5:LED 2:VREF    */
69 :      pd5 = 0xff;                         /*                              */
70 :      pd6 = 0xff;                         /*                              */
71 :      pd7 = 0xff;                         /*                              */
72 :      pd8 = 0xff;                         /*                              */
73 :      pd9 = 0x3f;                         /*                              */
74 :      pur0 = 0x04;                        /* P1_3～P1_0のプルアップON     */
75 :  }
76 :
77 :  /************************************************************************/
78 :  /* end of file                                                          */
79 :  /************************************************************************/
```

11.5 プログラムの解説

11.5.1 ヘッダファイルの取り込み

```
20 :  #include "sfr_r838a.h"                  /* R8C/38A 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¥Hew¥Tools¥Renesas¥nc30wa¥v545r00¥inc30 フォルダにあります。波線部分は、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_r838a.h」を取り込んでいます。このヘッダファイルは、R8C/38A マイコンの内蔵周辺機能を制御するためのレジスタ（Special Function 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_r8c38a

11.5.2 init 関数(クロックの選択)

init 関数で、R8C/38A 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①クロックの選択 ②I/O ポートの入出力設定 ③端子のプルアップを行います。ここでは、クロックの選択について説明します。

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

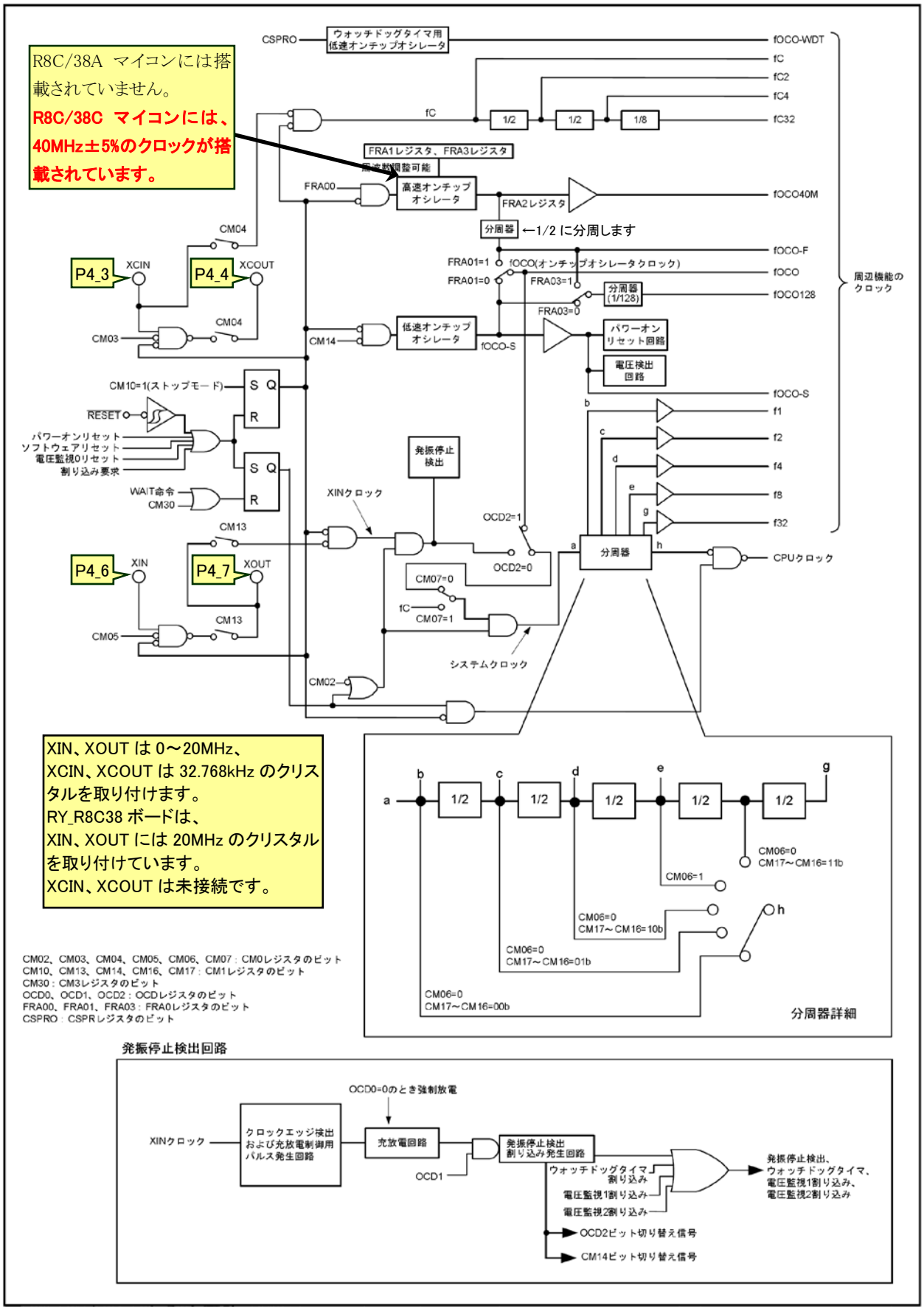
(1) R8C/38A のクロック

R8C/38A 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	RY_R8C38 ボードは 20MHz を実装	32.768kHz	40MHz±5% (38.0～42.0MHz)	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	**リセット後、プログラムでこのクロックに切り替えます。**	オプションです。標準では、取り付けていません。	**R8C/38A には搭載されていません。R8C/38C 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 クロックに切り替えます。	使用しません。

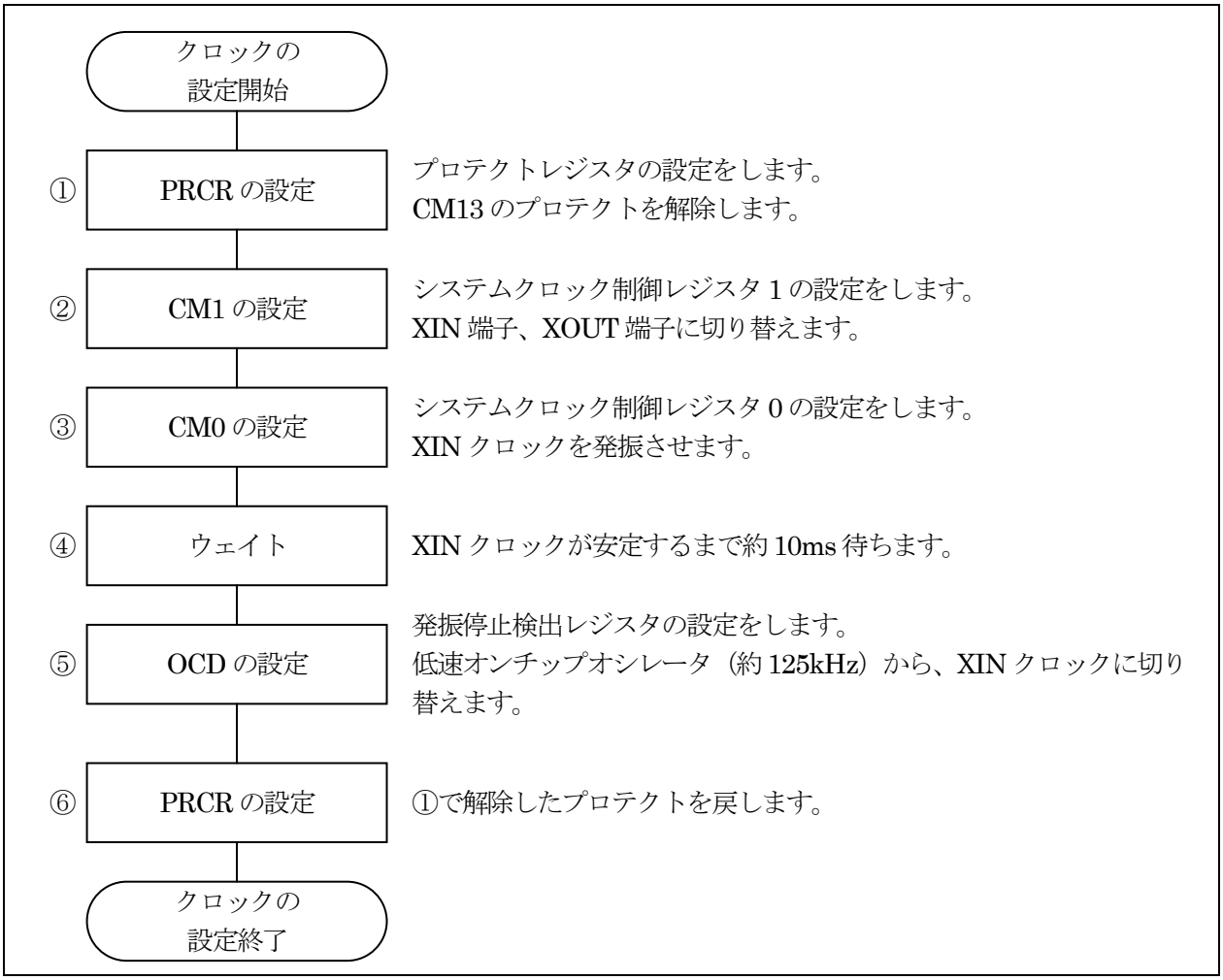
(2) ブロック図

R8C/38A、R8C/38C 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/38A 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/38A 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。

```
54 :      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。

RY_R8C38 ボードは、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。

```
55 :        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き 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。

```
56 :        cm05  = 0;                          /* XIN クロック発振              */
```

④ウエイト

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。

```
57 :      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 2 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 2 はすべて空き端子とします。

```
p2  = 0x00;                      最初に実行 ポート 2 の値をすべて"0"にする
pd2 = 0xff;                      最初に実行 ポート 2 の入出力設定  すべて出力

～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～

p2  = 0xff;                      // ポート 2 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) */
p2  = 0x00;                      // ポート 2 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/38A 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。

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

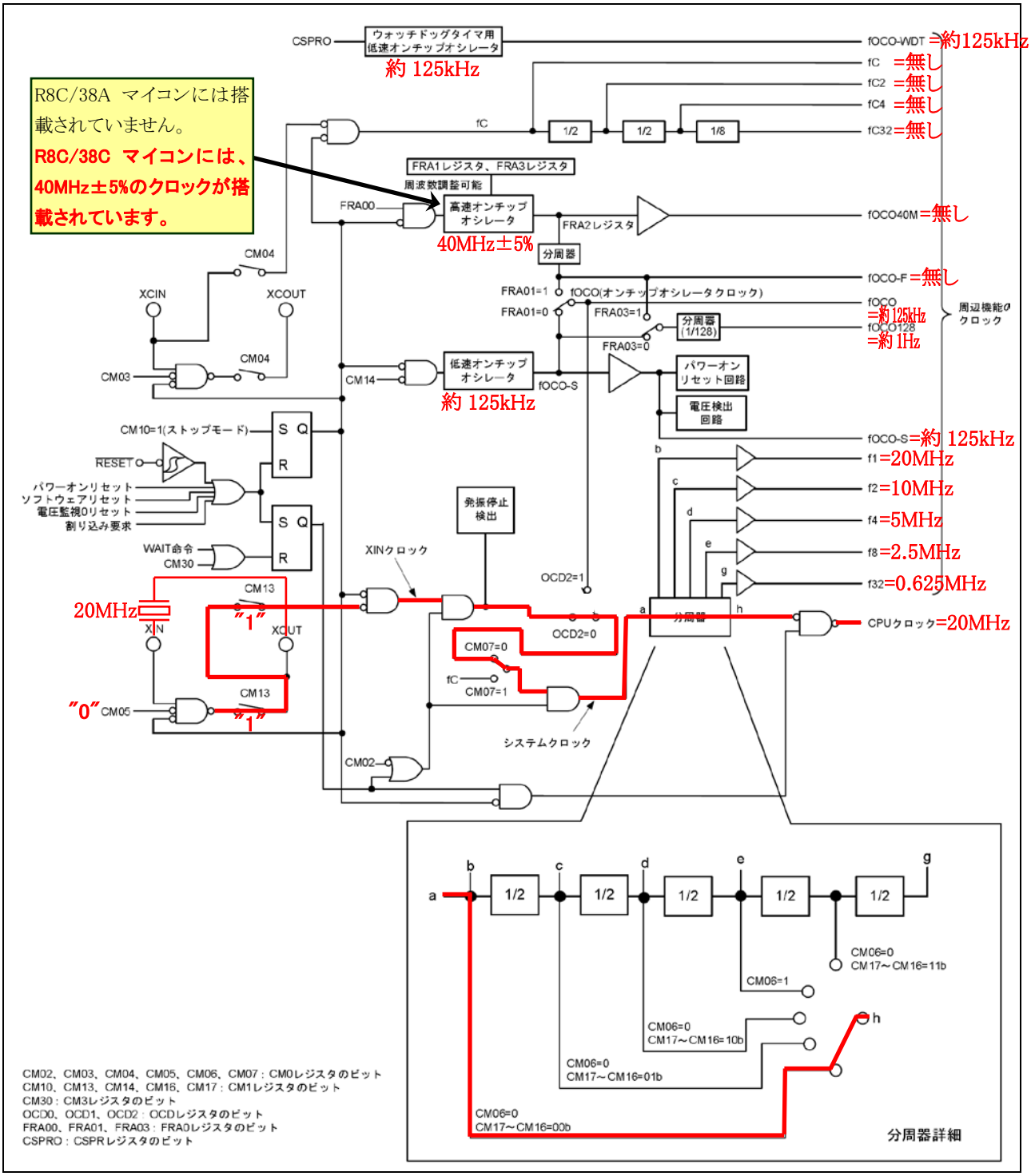
⑥プロテクトレジスタ(PRCR：Protect register)の設定（書き込み禁止）

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）。

```
59 :        prc0  = 0;                          /* プロテクト ON                  */
```

(4) 設定後のクロックの状態

今回の設定をしたときの、クロック発生回路を下記に示します。



11.5.3 init 関数(I/O ポートの入出力設定)

init 関数内で①クロックの選択 ②I/O ポートの入出力設定 ③端子のプルアップを行っています。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。

```
61 :      /* ポートの入出力設定 */
62 :      prc2 = 1;                    /* PD0のプロテクト解除          */
63 :      pd0 = 0x00;                  /* 7-0:スイッチ                 */
64 :      pd1 = 0xd0;                  /* 5:RXD0 4:TXD0 3-0:DIP SW     */
65 :      pd2 = 0xff;                  /* 7-0:LED                      */
66 :      pd3 = 0xff;                  /*                              */
67 :      p4  = 0x20;                  /* P4_5のLED:初期は点灯         */
68 :      pd4 = 0xb8;                  /* 7:XOUT 6:XIN 5:LED 2:VREF    */
69 :      pd5 = 0xff;                  /*                              */
70 :      pd6 = 0xff;                  /*                              */
71 :      pd7 = 0xff;                  /*                              */
72 :      pd8 = 0xff;                  /*                              */
73 :      pd9 = 0x3f;                  /*                              */
```

今回の実習でのポートと接続先の関係、入出力方向について、下表に示します。

ポート	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
0	実習基板 Ver.2 スライドスイッチ 入力	実習基板 Ver.2 スライドスイッチ 入力	実習基板 Ver.2 スライドスイッチ 入力	実習基板 Ver.2 スライドスイッチ 入力	実習基板 Ver.2 スライドスイッチ 入力	実習基板 Ver.2 スライドスイッチ 入力	実習基板 Ver.2 スライドスイッチ 入力	実習基板 Ver.2 スライドスイッチ 入力
1	未接続	未接続	RxD0 入力	TxD0 出力	RY_R8C38 ボード上の SW3 入力	RY_R8C38 ボード上の SW2 入力	RY_R8C38 ボード上の SW1 入力	RY_R8C38 ボード上の SW0 入力
2	実習基板 Ver.2 LED 出力	実習基板 Ver.2 LED 出力	実習基板 Ver.2 LED 出力	実習基板 Ver.2 LED 出力	実習基板 Ver.2 LED 出力	実習基板 Ver.2 LED 出力	実習基板 Ver.2 LED 出力	実習基板 Ver.2 LED 出力
3	未接続	未接続	未接続	未接続	未接続	未接続	未接続	未接続
4	クリスタル 出力	クリスタル 入力	RY_R8C38 ボード上の LED 出力	時計用 クリスタル ※未接続 出力	時計用 クリスタル ※未接続 出力	Vcc 入力		
5	未接続	未接続	未接続	未接続	未接続	未接続	未接続	未接続
6	未接続	未接続	未接続	未接続	未接続	未接続	未接続	未接続
7	未接続	未接続	未接続	未接続	未接続	未接続	未接続	未接続
8	未接続	未接続	未接続	未接続	未接続	未接続	未接続	未接続
9			未接続	未接続	未接続	未接続	未接続	未接続

※リセット後は、全て入力ポートです。
※表の斜線の bit は、端子がない bit です。
※太線枠は、RY_R8C38 ボード上で使っている端子です。

※P4_2 のみ入力専用端子です。他の端子は、入出力可能です。
※P4_2 は、A/D コンバータ用基準電圧を入力するか、通常の入力端子として使用します。今回は、A/D コンバータ用基準電圧として Vcc を接続しています。
※P4_3、P4_4 に時計用クリスタルが接続されている場合は、両ビットとも入力にしてください。

(1) I/O ポートとは

I/O ポート（入出力ポート、Input/Output Port）とは、コンピュータ等の情報機器において、装置の外部に接続する機器との情報の入出力に使用するインタフェースの端子（ポート）を指します(出典：Wikipedia)。
R8C/38A には、ポート 0～ポート 9 まで 10 個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。
1 つの I/O ポートには、2 組のレジスタがあります。レジスタの内容を下表に示します。

PD○	○には 0～9 の数字が入ります。 ポートの入出力方向を設定するレジスタです。 例） pd3 = 0x0f // 0000 1111 bit7～4 は入力端子、bit3～0 は出力端子	
P○	○には 0～9 の数字が入ります。	
	入力ポートの場合	p○の値を読み込むことによって、端子の入力電圧が分かります。 例） c = p0 // ポート 0 の状態を変数 c に代入
	出力ポートの場合	p○に値を書き込むことによって、端子から 0V または 5V を出力します。 p○の値を読み込むと、現在出力している電圧が分かります。 例） p2 = 0x12 // 00010011 をポート 2 の端子から出力

(2) ポートの入出力方向の設定

ポートの入出力設定は、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を設定します。
考え方は、次のとおりです。

①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。
②LED 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。

ポートの入出力方向の設定は、PD0～PD9 の各レジスタで行います。数字はポート番号で PD0 はポート 0、PD1 はポート 1 となります。
PD0～PD9 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、
何も接続せずに出力("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

①～④の考え方を基に、129 ページの表を"1"(出力端子)、"0"(入力端子)で書き換えたポート 0～9 の設定値を下記に示します。

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。2 進数を 16 進数に変換する方が簡単なため、通常は 16 進数に変換します。表のいちばん右の列に 16 進数に変換した値を記載しておきます。

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	0	0	0	0	0xd0
2	1	1	1	1	1	1	1	1	0xff
3	1	1	1	1	1	1	1	1	0xff
4	1	0	1	1	1	0	0	0	0xb8
5	1	1	1	1	1	1	1	1	0xff
6	1	1	1	1	1	1	1	1	0xff
7	1	1	1	1	1	1	1	1	0xff
8	1	1	1	1	1	1	1	1	0xff
9	0	0	1	1	1	1	1	1	0x3f

16 進数に変換した値を、PD0～PD9 に設定すれば、入出力方向の設定は完了です。
プログラムを下記に示します。

```
62 :        prc2 = 1;                       /* PD0のプロテクト解除          */
63 :        pd0 = 0x00;                     /* 7-0:ディップスイッチ         */
64 :        pd1 = 0xd0;                     /* 5:RXD0 4:TXD0 3-0:DIP SW     */
65 :        pd2 = 0xff;                     /* 7-0:LED                      */
66 :        pd3 = 0xff;                     /*                              */
67 :        p4  = 0x20;                     /* P4_5のLED:初期は点灯         */
68 :        pd4 = 0xb8;                     /* 7:XOUT 6:XIN 5:LED 2:VREF    */
69 :        pd5 = 0xff;                     /*                              */
70 :        pd6 = 0xff;                     /*                              */
71 :        pd7 = 0xff;                     /*                              */
72 :        pd8 = 0xff;                     /*                              */
73 :        pd9 = 0x3f;                     /*                              */
```

62 行については、次ページで説明します。

※R8C のコンパイラは、「0b」を付けると、2 進数表記をすることができます。ポート 1 の設定例を下記に示します。

```
64 :        pd1 = 0b11010000;
```

※PD0 を設定するときの注意点

R8C/38A 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～PD9 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（PRCR）の bit2 を"1"にします。このビットは「sfr_r838a.h（R8C/38A 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 PRCR 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=PRCR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

出力端子から、電圧(0V または 5V)を出力するには、P0～P9 の各レジスタに値を書き込みます。数字はポート番号で P0 はポート 0、P1 はポート 1 となります。

P0～P9 の各レジスタへの設定値は、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（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
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

入力端子に入力されている電圧(0V または 5V)を確認するには、P0～P9 の各レジスタの値を読み込みます。数字はポート番号で P0 はポート 0、P1 はポート 1 となります（レジスタ名はデータの出力と同じです）。

P0～P9 の各レジスタの値は、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
```

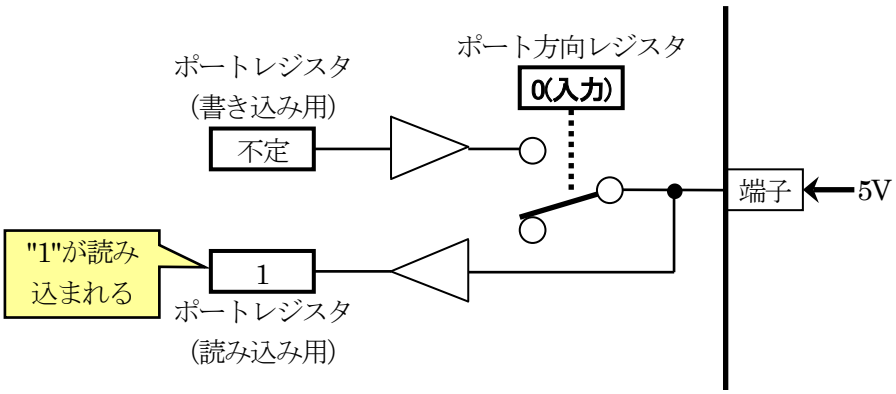
(5) 出力設定にするときのポート方向レジスタ(PD0～PD9)とポートレジスタ（P0～P9)の設定手順

ポート方向レジスタ(PD0～PD9)とポートレジスタ（P0～P9)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～PD9)	0x00(入力設定)
ポートレジスタ(P0～P9)	不定（どのような値か分からない）

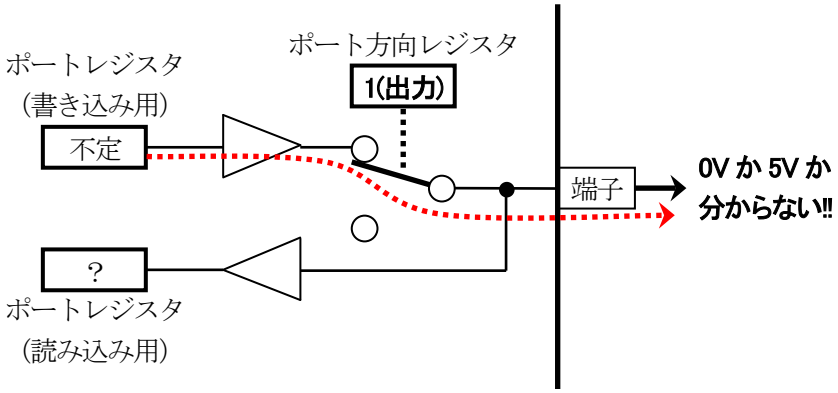
●入力設定の場合

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。

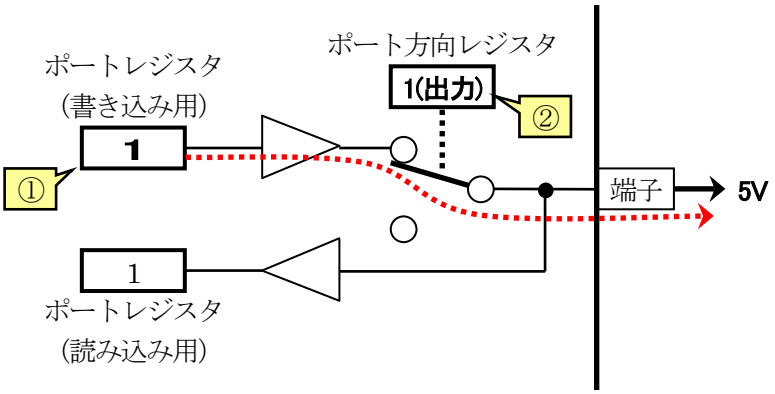


●出力設定の場合

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**。



例えば、RY_R8C38 ボード上の LED(ポート 4 の bit5)は、"0"で消灯、"1"で点灯します。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。

```
    p4   = 0x20;                    /* P4_5のLED:初期は点灯          */
    pd4  = 0xb8;                    /* 7:XOUT 6:XIN 5:LED 2:VREF     */
```

(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
　　0.65Vcc＝0.65×5.0＝3.25V　　∴3.25V～5.0V の電圧なら"1"と判断
"0"と判断する電圧は表より
　　0.4Vcc＝0.4×5.0＝2.0V　　∴0～2.0V 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 ≦ Vcc ≦ 5.5V	—	—	20	MHz
			1.8V ≦ Vcc ＜ 2.7V	—	—	5	MHz
$f_{(XCIN)}$	XCINクロック入力発振周波数		1.8V ≦ Vcc ≦ 5.5V	—	32.768	50	kHz
fOCO40M	タイマRC、タイマRDのカウントソース(注3)		2.7V ≦ Vcc ≦ 5.5V	32	—	40	MHz
fOCO-F	fOCO-F周波数		2.7V ≦ Vcc ≦ 5.5V	—	—	20	MHz
			1.8V ≦ Vcc ＜ 2.7V	—	—	5	MHz
—	システムクロック周波数		2.7V ≦ Vcc ≦ 5.5V	—	—	20	MHz
			1.8V ≦ Vcc ＜ 2.7V	—	—	5	MHz
$f_{(BCLK)}$	CPUクロック周波数		2.7V ≦ Vcc ≦ 5.5V	—	—	20	MHz
			1.8V ≦ Vcc ＜ 2.7V	—	—	5	MHz

注1.　指定のない場合は、Vcc = 1.8V～5.5V、Topr = −20℃～85℃(Nバージョン)/−40℃～85℃(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.　fOCO40MはVcc = 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($I_{OL(sum)}$)。

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 最小	規格値 標準	規格値 最大	単位
V_{OH}	“H” 出力電圧		駆動能力High V_{CC} = 5V	I_{OH} = −20mA	Vcc−2.0	—	Vcc	V
			駆動能力Low V_{CC} = 5V	I_{OH} = −5mA	Vcc−2.0	—	Vcc	V
V_{OL}	“L” 出力電圧		駆動能力High V_{CC} = 5V	I_{OL} = 20mA	—	—	2.0	V
			駆動能力Low V_{CC} = 5V	I_{OL} = 5mA	—	—	2.0	V
V_{T+}-V_{T-}	ヒステリシス	$\overline{INT0}$、$\overline{INT1}$、$\overline{INT2}$、$\overline{INT3}$、$\overline{INT4}$、$\overline{KI0}$、$\overline{KI1}$、$\overline{KI2}$、$\overline{KI3}$、TRAIO、TRBO、TRCIOA、TRCIOB、TRCIOC、TRCIOD、TRDIOA0、TRDIOB0、TRDIOC0、TRDIOD0、TRDIOA1、TRDIOB1、TRDIOC1、TRDIOD1、$\overline{TRCTRG}$、TRCCLK、$\overline{ADTRG}$、RXD0、RXD1、RXD2、CLK0、CLK1、CLK2、SSI、SCL、SDA、SSO		0.1	1.2	—	V
		$\overline{RESET}$			0.1	1.2	—	V
I_{IH}	“H” 入力電流		VI = 5V		—	—	5.0	μA
I_{IL}	“L” 入力電流		VI = 0V		—	—	−5.0	μA
R_{PULLUP}	プルアップ抵抗		VI = 0V		25	50	100	kΩ
R_{fXIN}	帰還抵抗	XIN			—	0.3	—	MΩ
R_{fXCIN}	帰還抵抗	XCIN			—	8	—	MΩ
V_{RAM}	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≦Vcc≦5.5V、Topr = −20℃～85℃(Nバージョン)/−40℃～85℃(Dバージョン)、f(XIN) = 20MHzです。

″1″出力は、(Vcc-2.0)～Vcc の電圧が出力されます。Vcc 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。

11.5.4 init 関数(端子のプルアップ)

```
74 :        pur0 = 0x04;                        /* P1_3～P1_0のプルアップON      */
```

R8C/38A マイコンのポートのほとんどに、プルアップ抵抗が内蔵されています。入力ポートの場合、マイコン内蔵のプルアップ抵抗を ON にすれば、外付けでプルアップ抵抗を付ける必要がありません。
内蔵プルアップの抵抗値は、25～100kΩです。標準は 50kΩです。

今回、マイコンボード上にあるディップスイッチは、P1_3～P1_0 端子に接続されています。ただし、プルアップ抵抗が接続されていないため、ディップスイッチを OFF にしたときは端子が開放状態となり、"0"でもない、"1"でもない状態になります。
そのため、ディップスイッチがある P1_3～P1_0 端子のマイコン内蔵のプルアップ抵抗を、ON にします。
マイコン内蔵のプルアップ抵抗をありにするか、なしにするかを設定するレジスタが、**プルアップ制御レジスタ 0～2（PUR0～PUR2)**という 3 つのレジスタです。これから、3 つのレジスタの意味を説明します。

①プルアップ制御レジスタ 0（PUR0：Pull-Up Control Register 0）の設定

ディップスイッチに接続されている P1_0～P1_3 端子のプルアップを「あり」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	P3_4～P3_7 のプルアップ pu07	0:プルアップなし 1:プルアップあり	0
bit6	P3_0～P3_3 のプルアップ pu06	0:プルアップなし 1:プルアップあり	0
bit5	P2_4～P2_7 のプルアップ pu05	0:プルアップなし 1:プルアップあり	0
bit4	P2_0～P2_3 のプルアップ pu04	0:プルアップなし 1:プルアップあり	0
bit3	P1_4～P1_7 のプルアップ pu03	0:プルアップなし 1:プルアップあり	0
bit2	P1_0～P1_3 のプルアップ pu02	0:プルアップなし **1:プルアップあり** **ディップスイッチは P1_0～P1_3 に接続されているので、この端子のプルアップをありにします。**	1
bit1	P0_4～P0_7 のプルアップ pu01	0:プルアップなし 1:プルアップあり	0
bit0	P0_0～P0_3 のプルアップ pu00	0:プルアップなし 1:プルアップあり	0

プルアップ制御レジスタ 0（PUR0)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**	**0**
16 進数	0				4			

②プルアップ制御レジスタ 1（PUR1：Pull-Up Control Register 1）の設定

今回、PUR1 で設定したいプルアップ抵抗はないため、このレジスタの設定はしていません。プルアップしたい場合は、適宜設定してください。

設定 bit	上：ビット名 下：シンボル	内容	今回の内容
bit7	P7_4～P7_7 のプルアップ pu17	0:プルアップなし 1:プルアップあり	設定せず
bit6	P7_0～P7_3 のプルアップ pu16	0:プルアップなし 1:プルアップあり	設定せず
bit5	P6_4～P6_7 のプルアップ pu15	0:プルアップなし 1:プルアップあり	設定せず
bit4	P6_0～P6_3 のプルアップ pu14	0:プルアップなし 1:プルアップあり	設定せず
bit3	P5_4～P5_7 のプルアップ pu13	0:プルアップなし 1:プルアップあり	設定せず
bit2	P5_0～P5_3 のプルアップ pu12	0:プルアップなし 1:プルアップあり	設定せず
bit1	P4_4～P4_7 のプルアップ pu11	0:プルアップなし 1:プルアップあり	設定せず
bit0	P4_3 のプルアップ pu10	0:プルアップなし 1:プルアップあり	設定せず

③プルアップ制御レジスタ 2（PUR2：Pull-Up Control Register 2）の設定

今回、PUR2 で設定したいプルアップ抵抗はないため、このレジスタの設定はしていません。プルアップしたい場合は、適宜設定してください。

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	設定せず
bit3	P9_4～P9_7 のプルアップ pu23	0:プルアップなし 1:プルアップあり	設定せず
bit2	P9_0～P9_3 のプルアップ pu22	0:プルアップなし 1:プルアップあり	設定せず
bit1	P8_4～P8_7 のプルアップ pu21	0:プルアップなし 1:プルアップあり	設定せず
bit0	P8_0～P8_3 のプルアップ pu20	0:プルアップなし 1:プルアップあり	設定せず

11.5.5 main 関数

```
34 :  void main( void )
35 :  {
36 :      unsigned char d;
37 :
38 :      init();                                /* 初期化                          */
39 :
40 :      while( 1 ) {
41 :          d = p0;
42 :          p2 = d;
43 :      }
44 :  }
```

「io.c」は、main 関数から実行されます。

38 行	init 関数を実行します。
40 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。40 行目の最後の「{」から、43 行目の「}」の間のプログラムが無限に繰り返されます。要は、41 行、42 行が無限に繰り返されます。
41 行	変数 d にポート 0 の状態を読み込み、保存します。
42 行	ポート d の値を、ポート 2 に出力します。

11.6 ビット操作

11.6.1 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

そのままの部分を「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;
p2 = d;                    ←結果をポート2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 1 にスイッチが繋がっていて、bit0 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( p1 == 0x01 ) {
    /* bit0 が“1”ならこの中を実行 */
}
```

しかし、ポート 1 の bit7～1 がどのような値になっているか分かりません。例えば、bit0 が“1”、bit1 も”1”ならポート 0 の値は 2 進数で"0000 0011"、16 進数に直すと 0x03 となります。そのため、次のようになります。

```
if( p1 == 0x01 ) {     ←p1 は 0x03 なので式は成り立たない
    /* bit0 は“1”だが、この中を実行しない！！ */
}
```

bit0 は"1"ですが、bit1 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、**チェックする必要のないビットを 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 処理でチェックする必要のないビットを強制的に"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 0、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？	しない	しない	しない	しない	しない	しない	しない	**する**

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	0	0	0	0	0	0	0	**1**

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、bit7～1 が"0"であることが分かっているので、bit0 だけ"1"か"0"かをチェックすればいいことになります。

bit0 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( (p1 & 0x01) == 0x01 ) {
    /* bit0 が“1”ならこの中を実行 */
}
```

P1 が 0x91 なら、次のような計算になります。

```
    P1      1001 0001
    マスク値 0000 0001  (AND
    結果     0000 0001
```

bit0 は"1"なので式は成り立ち、カッコの中が実行されます。

bit0 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( (p1 & 0x01) == 0x00 ) {
    /* bit0 が “0” ならこの中を実行 */
}
```

P1 が 0x76 なら、次のような計算になります。

P1	0111 0110
マスク値	0000 0001 （AND
結果	0000 0000

bit0 は″0 なので式は成り立ち、カッコの中が実行されます。

11.6.2 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;
    p2 = d;                ←結果をポート2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

11.6.3 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

そのままの部分を「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″^″(アクサンシルコンフレックス)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;
p2 = d;                    ←結果をポート2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

11.6.4 全ビット反転する(NOT 演算)

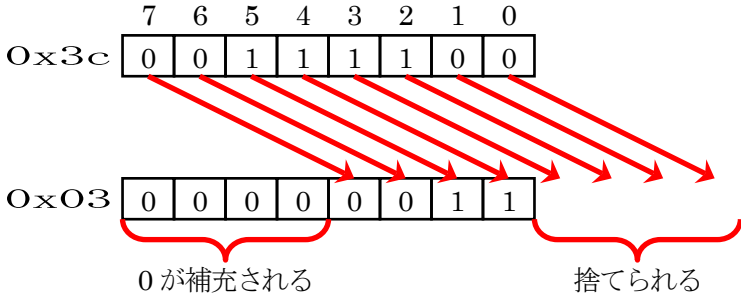
全ビット反転するなら、C 言語では″~″(チルダ)を用います。C 言語で記述すると次のようになります。

```
d = ~p0;
p2 = d;                    ←結果をポート2へ出力
```

反転したい値の先頭に″~″を付けます。

11.6.5 ビットシフトする

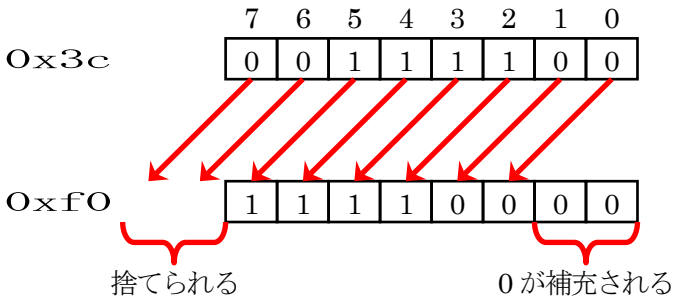
ビットシフトとは、各ビットを右や左にずらすことです。
例えば、ポート 0 を 4 ビット右にシフトするとします。ポート 0 が 0x3c なら、結果は 0x03 になります。



右シフトの記述は C 言語では″>>″（大なり大なり）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
p2 = d;                ←結果をポート2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″<<″（小なり小なり）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
p2 = d;                ←結果をポート2へ出力
```

11.7 演習

(1) ポート0にLED部、ポート2にスライドスイッチ部を接続して、スライドスイッチの状態をLEDへ出力しなさい。

(2) (1)のとき、スライドスイッチの状態 bit7～bit0 を読み込み、読み込んだ値の bit7～bit4 は強制的に″0″にして bit3～bit0 はそのままの値を LED へ出力しなさい(bit7～bit4 の LED は常に消灯、bit3～bit0 はディップスイッチの状態により点灯／消灯する)。

(3) (1)のとき、スライドスイッチの状態 bit7～bit0 を読み込み、読み込んだ値の bit3～bit0 は強制的に″1″にして bit3～bit0 はそのままの値を LED へ出力しなさい(bit3～bit0 の LED は常に点灯、bit7～bit4 はディップスイッチの状態により点灯／消灯する)。

(4) (1)のとき、スライドスイッチの状態 bit7～bit0 を読み込み、読み込んだ値をすべて反転して LED へ出力しなさい。

(5) (1)のとき、スライドスイッチの状態 bit7～bit0 を読み込み、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、スライドスイッチの bit4 以外は″1″か″0″か分からないものとする。

(6) (1)のとき、スライドスイッチの状態 bit7～bit0 を読み込み、読み込んだ値の bit5～2 のみを反転して、その他の bit はそのままの値を LED へ出力しなさい。

12. I/O ポートの入出力 2(プロジェクト:io2)

12.1 概要

本章では、RY_R8C38 ボードのディップスイッチ(4bit)、LED(1bit)を使って、I/O ポートの入出力を行う方法を説明します。ビット操作を行って入出力を行います。

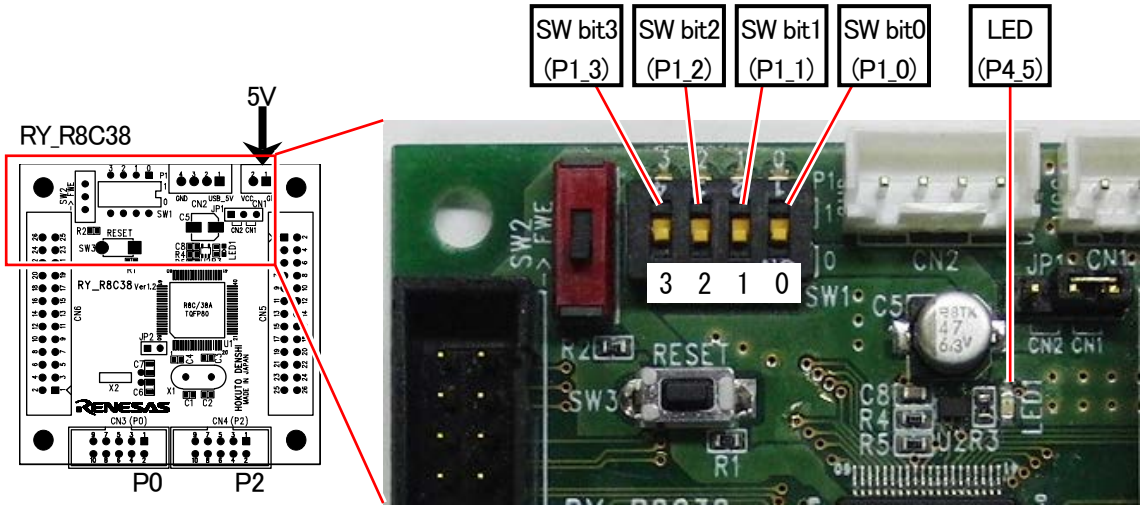
12.2 接続

■使用ポート

マイコンのポート	接続内容
P1_3、P1_2、P1_1、P1_0	RY_R8C38 ボードのディップスイッチです。
P4_5	RY_R8C38 ボードの LED です。

■接続例

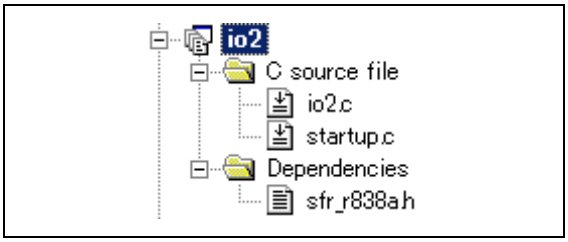
RY_R8C38 ボードだけで実習できます。



■操作方法

RY_R8C38 ボードのディップスイッチ(SW1)の bit0 を上下させると、それに合わせて LED1 が点灯(上側)／消灯(下側)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。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥io2¥startup.c
2	io2.c	実際に制御するプログラムが書かれています。R8C/38A の内蔵周辺機能(SFR)の初期化も行います。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥io2¥io2.c
3	sfr_r838a.h	R8C/38A マイコンの内蔵周辺機能を制御するためのレジスタ（Special Function Registers)を定義したファイルです。 ファイルの位置→C:¥Workspace¥common_r8c38a¥sfr_r838a.h

12.4 プログラム「io2.c」

```
 1 :  /***********************************************************************/
 2 :  /* 対象マイコン  R8C/38A                                               */
 3 :  /* ﾌｧｲﾙ内容      データの入力、出力                                    */
 4 :  /* バージョン    Ver.1.00                                              */
 5 :  /* Date          2010.04.01                                            */
 6 :  /* Copyright     ジャパンマイコンカーラリー実行委員会                  */
 7 :  /***********************************************************************/
 8 :  /*
 9 :  入力：P1_0(マイコンボード上のディップスイッチbit0)
10 :  出力：P4_5(マイコンボード上のLED)
11 :
12 :  マイコンボードのディップスイッチ値(bit0)を、
13 :  マイコンボード上のLEDへ出力します。
14 :  */
15 :
16 :  /*====================================*/
17 :  /* インクルード                       */
18 :  /*====================================*/
19 :  #include "sfr_r838a.h"                 /* R8C/38A 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 :  void led_out_m( unsigned char led );
31 :
```

```
 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 = dipsw_get() & 0x01;
 43 :          led_out_m( d );
 44 :      }
 45 :  }
 46 :
 47 :  /************************************************************************/
 48 :  /* R8C/38A 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;                         /*                              */
 65 :      pd1 = 0xd0;                         /* 5:RXD0 4:TXD0 3-0:DIP SW     */
 66 :      pd2 = 0xff;                         /*                              */
 67 :      pd3 = 0xff;                         /*                              */
 68 :      p4  = 0x20;                         /* P4_5のLED:初期は点灯         */
 69 :      pd4 = 0xb8;                         /* 7:XOUT 6:XIN 5:LED 2:VREF    */
 70 :      pd5 = 0xff;                         /*                              */
 71 :      pd6 = 0xff;                         /*                              */
 72 :      pd7 = 0xff;                         /*                              */
 73 :      pd8 = 0xff;                         /*                              */
 74 :      pd9 = 0x3f;                         /*                              */
 75 :      pur0 = 0x04;                        /* P1_3～P1_0のプルアップON     */
 76 :  }
 77 :
 78 :  /************************************************************************/
 79 :  /* ディップスイッチ値読み込み                                           */
 80 :  /* 戻り値 スイッチ値 0～15                                              */
 81 :  /************************************************************************/
 82 :  unsigned char dipsw_get( void )
 83 :  {
 84 :      unsigned char sw;
 85 :
 86 :      sw = p1 & 0x0f;                     /* P1_3～P1_0読み込み           */
 87 :
 88 :      return  sw;
 89 :  }
 90 :
 91 :  /************************************************************************/
 92 :  /* マイコンボードのLED出力                                              */
 93 :  /* 引数   0:消灯 1:点灯                                                */
 94 :  /************************************************************************/
 95 :  void led_out_m( unsigned char led )
 96 :  {
 97 :      if( led ) {
 98 :          p4 |= 0x20;                     /* LED点灯                      */
 99 :      } else {
100 :          p4 &= 0xdf;                     /* LED消灯                      */
101 :      }
102 :  }
103 :
104 :  /************************************************************************/
105 :  /* end of file                                                          */
106 :  /************************************************************************/
```

12.5 プログラムの解説

12.5.1 dipsw_get 関数

dipsw_get 関数は、RY_R8C38 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。

```
78 :  /************************************************************************/
79 :  /* ディップスイッチ値読み込み                                           */
80 :  /* 戻り値 スイッチ値 0～15                                              */
81 :  /************************************************************************/
82 :  unsigned char dipsw_get( void )
83 :  {
84 :      unsigned char sw;
85 :
86 :      sw = p1 & 0x0f;                    /* P1_3～P1_0読み込み           */
87 :
88 :      return  sw;
89 :  }
```

RY_R8C38 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 P1_3、P1_2、P1_1、P1_0 の各 bit に接続されています。

ポート 1 の bit7～4 は関係ないので、ビット演算を使ってディップスイッチの値だけ取り込みます。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と"1"、下(ON と書いてある側)にすると"0"です。

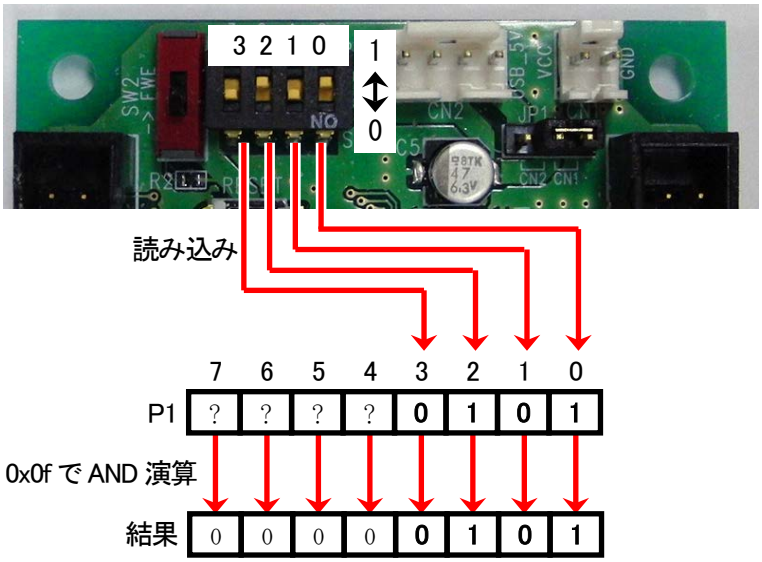
変数 sw にポート 1 の値を読み込みます。

```
86 :      sw = p1 & 0x0f;                    /* P1_3～P1_0読み込み           */
               ①    ②
```

① ポート 1 の値を読み込みます。

② ①の値を 0x0f でマスクします。0x0f は、「0000 1111」なので bit3～0 を有効に、他は強制的に"0"にします。

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。



12.5.2 led_out_m 関数

led_out_m 関数は、RY_R8C38 ボードにある LED 1 個を点灯／消灯させる関数です。

```
 91 :  /************************************************************************/
 92 :  /* マイコンボードのLED出力                                              */
 93 :  /* 引数    0:消灯 1:点灯                                               */
 94 :  /************************************************************************/
 95 :  void led_out_m( unsigned char led )
 96 :  {
 97 :      if( led ) {
 98 :          p4 |= 0x20;                    /* LED点灯                      */
 99 :      } else {
100 :          p4 &= 0xdf;                    /* LED消灯                      */
101 :      }
102 :  }
```

RY_R8C38 ボードに搭載されている LED は、ポート 4 の bit5 に接続されています。LED を制御するときに、ポート 4 の他のビットには影響がないようにします。
led 変数の値が 0 以外なら、ポート 4 の値を 0x20 で OR 演算してポート 4 に代入します。0x20 は 2 進数で「0010 0000」なので、bit5 のみ"1"になり、P4_5 に接続されている LED が点灯します。

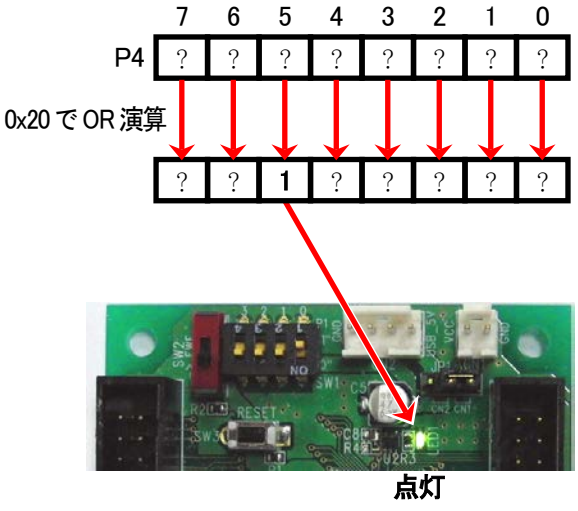
```
 97 :      if( led ) {
 98 :          p4 |= 0x20;                    /* LED点灯                      */
                  ↓
              p4 = p4 | 0x20; と同じ意味です
```

led 変数の値が 0 なら、ポート 4 の値を 0xdf で AND 演算してポート 4 に代入します。0xdf は 2 進数で「1101 1111」なので、bit5 のみ"0"になり、P4_5 に接続されている LED が消灯します。

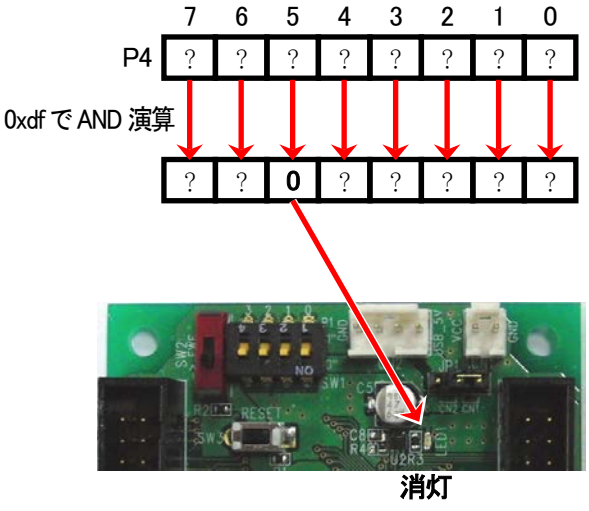
```
 99 :      } else {
100 :          p4 &= 0xdf;                    /* LED消灯                      */
                  ↓
              p4 = p4 & 0xdf; と同じ意味です
```

led_out_m 関数の動きを、下図に示します。

■引数が 0 以外の場合



■引数が 0 の場合



12.5.3　main 関数

```
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 = dipsw_get() & 0x01;
43 :          led_out_m( d );
44 :      }
45 :  }
```

main 関数は次のような動作をします。

42 行	変数 d に RY_R8C38 ボードのディップスイッチの値を読み込みます。このとき、0x01 で AND 演算して、bit0 のみ有効、bit3,bit2,bit1 は強制的に"0"にします。
43 行	RY_R8C38 ボードの LED に、変数 d の値を出力します。

結果、RY_R8C38 ボードのディップスイッチ bit0 の値を、RY_R8C38 ボードの LED へ出力します。

12.6 ビット単位でデータを入力、出力する

R8C/38A マイコンの内蔵周辺機能を制御するためのレジスタ(Special Function Registers)を定義したファイル「sfr_r838a.h」は、ポート Pi レジスタ(P0～P9)をビット単位で設定できるよう定義しています。
シンボル名は、下記のように定義されています。

```
px_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下記に示します。

```
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下記に示します。

```
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
}
```

12.7 演習

(1) 98 行目、100 行目を「Px_y」の形式に書き直しなさい。

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ので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。

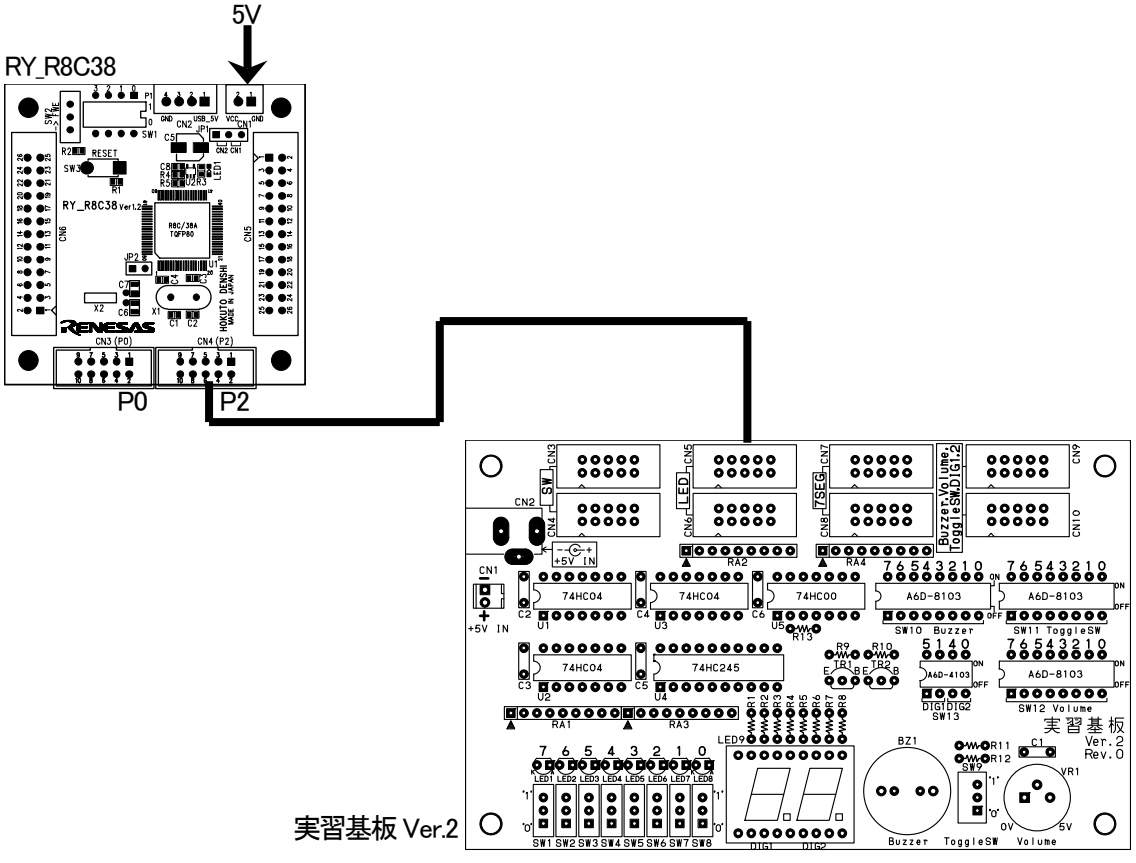
13.2 接続

■使用ポート

マイコンのポート	接続内容
P2 (CN4)	実習基板 Ver.2 の LED 部など、出力機器を接続します。

■接続例

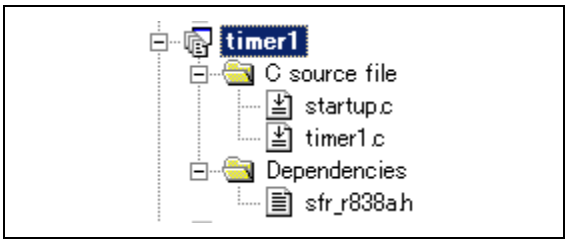
実習基板 Ver.2 を使ったときの接続例を下記に示します。



■操作方法

操作は特にありません。電源を入れると実習基板 Ver.2 の 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。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥timer1¥startup.c
2	timer1.c	実際に制御するプログラムが書かれています。R8C/38A の内蔵周辺機能(SFR)の初期化も行います。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥timer1¥timer1.c
3	sfr_r838a.h	R8C/38A マイコンの内蔵周辺機能を制御するためのレジスタ（Special Function Registers)を定義したファイルです。 ファイルの位置→C:¥Workspace¥common_r8c38a¥sfr_r838a.h

13.4 プログラム「timer1.c」

```
/************************************************************************/
/* 対象マイコン R8C/38A                                                 */
/* ﾌｧｲﾙ内容     ソフトウェアタイマ                                      */
/* バージョン   Ver.1.10                                                */
/* バージョン   Ver.1.00                                                */
/* Date         2010.04.01                                              */
/* Copyright    ジャパンマイコンカーラリー実行委員会                    */
/************************************************************************/
/*
出力：P2_7-P2_0(実習基板Ver.2のLED部)

ポート2に繋いだLEDを1秒間隔で点滅させます。
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。
*/

/*====================================*/
/* インクルード                       */
/*====================================*/
#include "sfr_r838a.h"                  /* R8C/38A SFRの定義ファイル     */

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

/*====================================*/
/* プロトタイプ宣言                   */
/*====================================*/
void init( void );
void timer( unsigned long timer_set );

```

```
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 :          p2 = 0x55;
40 :          timer( 1000 );
41 :          p2 = 0xaa;
42 :          timer( 1000 );
43 :          p2 = 0x00;
44 :          timer( 1000 );
45 :      }
46 :  }
47 :
48 :  /************************************************************************/
49 :  /* R8C/38A 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 = 0x00;                         /*                              */
66 :      pd1 = 0xd0;                         /* 5:RXD0 4:TXD0 3-0:DIP SW     */
67 :      pd2 = 0xff;                         /* LED基板など                  */
68 :      pd3 = 0xff;                         /*                              */
69 :      p4  = 0x20;                         /* P4_5のLED:初期は点灯         */
70 :      pd4 = 0xb8;                         /* 7:XOUT 6:XIN 5:LED 2:VREF    */
71 :      pd5 = 0xff;                         /*                              */
72 :      pd6 = 0xff;                         /*                              */
73 :      pd7 = 0xff;                         /*                              */
74 :      pd8 = 0xff;                         /*                              */
75 :      pd9 = 0x3f;                         /*                              */
76 :      pur0 = 0x04;                        /* P1_3～P1_0のプルアップON     */
77 :  }
78 :
79 :  /************************************************************************/
80 :  /* タイマ本体                                                           */
81 :  /* 引数    タイマ値 1=1ms                                               */
82 :  /************************************************************************/
83 :  void timer( unsigned long timer_set )
84 :  {
85 :      int i;
86 :
87 :      do {
88 :          for( i=0; i<1240; i++ );
89 :      } while( timer_set-- );
90 :  }
91 :
92 :  /************************************************************************/
93 :  /* end of file                                                          */
94 :  /************************************************************************/
```

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。

```
 79 :  /*********************************************************************/
 80 :  /* タイマ本体                                                        */
 81 :  /* 引数    タイマ値 1=1ms                                            */
 82 :  /*********************************************************************/
 83 :  void timer( unsigned long timer_set )
 84 :  {
 85 :      int i;
 86 :
 87 :      do {
 88 :          for( i=0; i<1240; i++ );      この行で 1ms の時間稼ぎ
 89 :      } while( timer_set-- );
 90 :  }
```

(1) timer 関数の使い方

timer 関数の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。使用例を下記に示します。

```
timer( 200 );     // 約 200ms この行で時間稼ぎする
```

(2) プログラムの解説

88 行	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。この繰り返しが 1ms になります。1240 という数字は、実測です。 ※1240 について この数値は、 ・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) ・ツールチェインの設定 ・クリスタルの値 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。

87 行、
89 行
	do {
	　　命令
	} while(条件);
	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。命令部分は、1ms の時間稼ぎなので、回数分×1ms の時間稼ぎとなります。

13.5.2 main 関数

```
34 :  void main( void )
35 :  {
36 :      init();                              /* 初期化                          */
37 :
38 :      while( 1 ) {
39 :          p2 = 0x55;
40 :          timer( 1000 );
41 :          p2 = 0xaa;
42 :          timer( 1000 );
43 :          p2 = 0x00;
44 :          timer( 1000 );
45 :      }
46 :  }
```

39 行	ポート 2 に 0x55(0101 0101)を出力します。
40 行	timer 関数で 1000ms（＝1 秒)の時間稼ぎをします。
41 行	ポート 2 に 0xaa(1010 1010)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 2 に 0x00(0000 0000)を出力します。
44 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μs(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 関数内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 :      init();                 ←init関数内の命令を実行する時間かかる(数百μs程度)
37 :
38 :      while( 1 ) {
39 :          p2 = 0x55;          ←数μs
40 :          timer( 1000 );      ←約1000ms
41 :          p2 = 0xaa;          ←数μs
42 :          timer( 1000 );      ←約1000ms
43 :          p2 = 0x00;          ←数μs
44 :          timer( 1000 );      ←約1000ms
45 :      }
46 :  }
```

13.6 演習

(1) 次の状態をポート 2 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 RY_R8C38 ボードの LED に出力するようにしなさい。
① ″1″を 0.1 秒間
② ″0″を 0.9 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/38A 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。

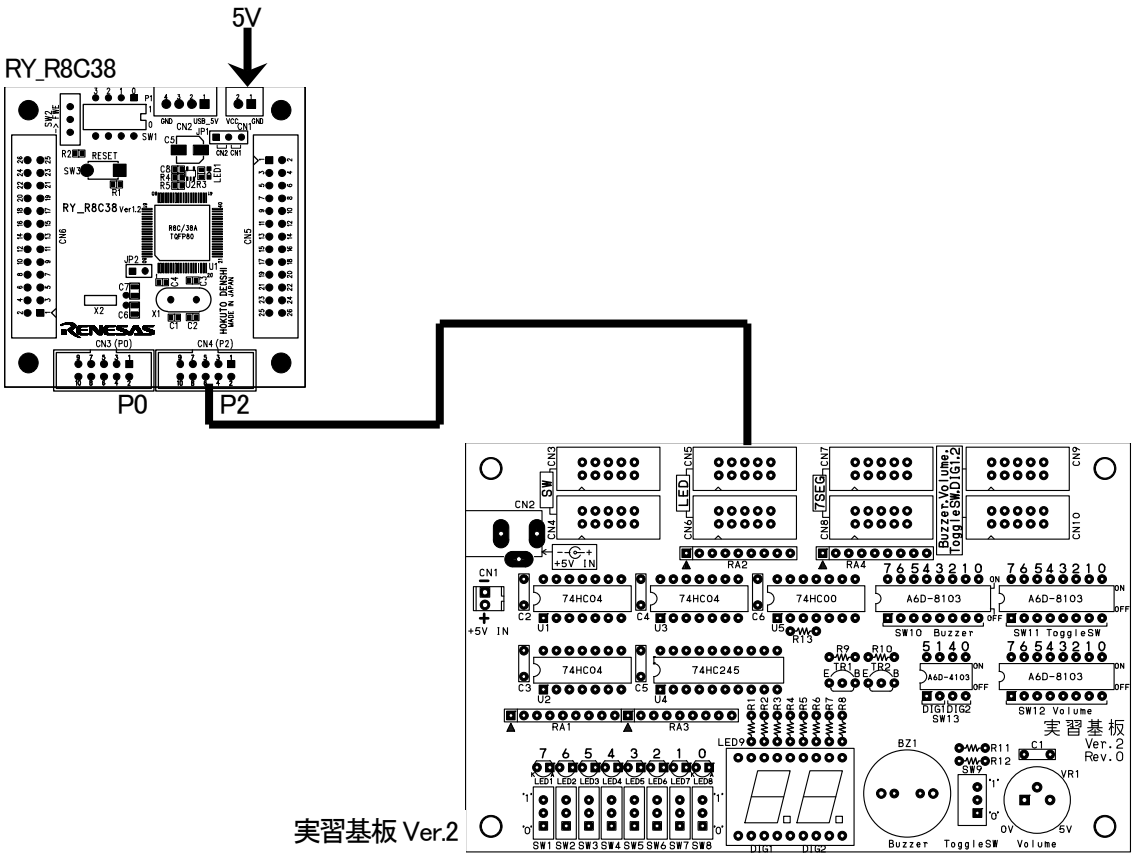
14.2 接続

■使用ポート

マイコンのポート	接続内容
P2 (CN4)	実習基板 Ver.2 の LED 部など、出力機器を接続します。

■接続例

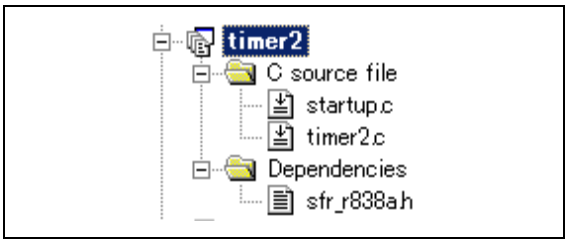
実習基板 Ver.2 を使ったときの接続例を下記に示します。



■操作方法

操作は特にありません。電源を入れると実習基板 Ver.2 の 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。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥timer2¥startup.c
2	timer2.c	実際に制御するプログラムが書かれています。R8C/38A の内蔵周辺機能(SFR)の初期化も行います。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥timer2¥timer2.c
3	sfr_r838a.h	R8C/38A マイコンの内蔵周辺機能を制御するためのレジスタ（Special Function Registers)を定義したファイルです。 ファイルの位置→C:¥Workspace¥common_r8c38a¥sfr_r838a.h

14.4 プログラム「timer2.c」

```
 1 :  /***********************************************************************/
 2 :  /* 対象マイコン R8C/38A                                                */
 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.1.00                                               */
 5 :  /* Date         2010.04.01                                             */
 6 :  /* Copyright    ジャパンマイコンカーラリー実行委員会                   */
 7 :  /***********************************************************************/
 8 :  /*
 9 :  出力：P2_7-P2_0(実習基板Ver.2のLED部)
10 :
11 :  ポート2に繋いだLEDを1秒間隔で点滅させます。
12 :  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。
13 :  */
14 :
15 :  /*====================================*/
16 :  /* インクルード                       */
17 :  /*====================================*/
18 :  #include "sfr_r838a.h"                 /* R8C/38A SFRの定義ファイル     */
19 :
20 :  /*====================================*/
21 :  /* シンボル定義                       */
22 :  /*====================================*/
23 :
24 :  /*====================================*/
25 :  /* プロトタイプ宣言                   */
26 :  /*====================================*/
27 :  void init( void );
28 :  void timer( unsigned long timer_set );
29 :
```

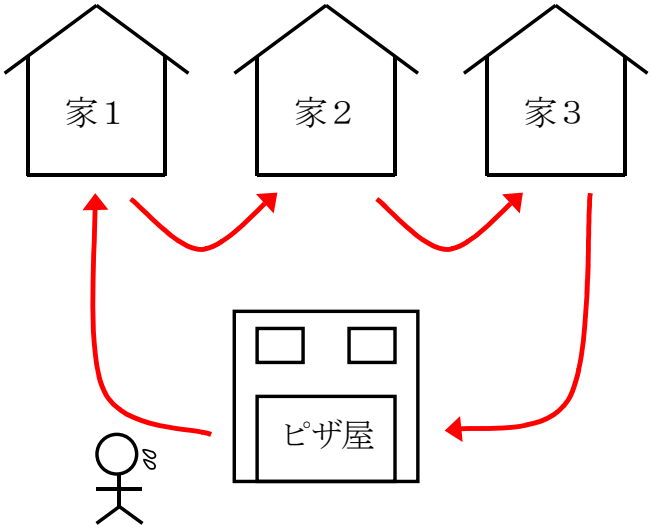
```
 30 :  /*====================================*/
 31 :  /* グローバル変数の宣言               */
 32 :  /*====================================*/
 33 :  unsigned long cnt_rb;                  /* タイマRB用                 */
 34 :
 35 :  /************************************************************************/
 36 :  /* メインプログラム                                                     */
 37 :  /************************************************************************/
 38 :  void main( void )
 39 :  {
 40 :      init();                          /* 初期化                         */
 41 :      asm(" fset I ");                 /* 全体の割り込み許可             */
 42 :
 43 :      while( 1 ) {
 44 :          p2 = 0x55;
 45 :          timer( 1000 );
 46 :          p2 = 0xaa;
 47 :          timer( 1000 );
 48 :          p2 = 0x00;
 49 :          timer( 1000 );
 50 :      }
 51 :  }
 52 :
 53 :  /************************************************************************/
 54 :  /* R8C/38A 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;                        /* PD0のプロテクト解除            */
 70 :      pd0 = 0x00;                      /*                                */
 71 :      pd1 = 0xd0;                      /* 5:RXD0 4:TXD0 3-0:DIP SW       */
 72 :      pd2 = 0xff;                      /* 7-0:LED                        */
 73 :      pd3 = 0xff;                      /*                                */
 74 :      p4  = 0x20;                      /* P4_5のLED:初期は点灯           */
 75 :      pd4 = 0xb8;                      /* 7:XOUT 6:XIN 5:LED 2:VREF      */
 76 :      pd5 = 0xff;                      /*                                */
 77 :      pd6 = 0xff;                      /*                                */
 78 :      pd7 = 0xff;                      /*                                */
 79 :      pd8 = 0xff;                      /*                                */
 80 :      pd9 = 0x3f;                      /*                                */
 81 :      pur0 = 0x04;                     /* P1_3～P1_0のプルアップON       */
 82 :
 83 :      /* タイマRBの設定 */
 84 :      /* 割り込み周期 = 1 / 20[MHz]    * (TRBPRE+1) * (TRBPR+1)
 85 :                      = 1 / (20*10^6) * 200        * 100
 86 :                      = 0.001[s] = 1[ms]
 87 :      */
 88 :      trbmr  = 0x00;                   /* 動作モード、分周比設定         */
 89 :      trbpre = 200-1;                  /* プリスケーラレジスタ           */
 90 :      trbpr  = 100-1;                  /* プライマリレジスタ             */
 91 :      trbic  = 0x07;                   /* 割り込み優先レベル設定         */
 92 :      trbcr  = 0x01;                   /* カウント開始                   */
 93 :  }
 94 :
 95 :  /************************************************************************/
 96 :  /* タイマ本体                                                           */
 97 :  /* 引数   タイマ値 1=1ms                                                */
 98 :  /************************************************************************/
 99 :  void timer( unsigned long timer_set )
100 :  {
101 :      cnt_rb = 0;
102 :      while( cnt_rb < timer_set );
103 :  }
104 :
105 :  /************************************************************************/
106 :  /* タイマRB 割り込み処理                                                */
107 :  /************************************************************************/
108 :  #pragma interrupt intTRB(vect=24)
109 :  void intTRB( void )
110 :  {
111 :      cnt_rb++;
112 :  }
113 :
114 :  /************************************************************************/
115 :  /* end of file                                                          */
116 :  /************************************************************************/
```

14.5 プログラムの解説

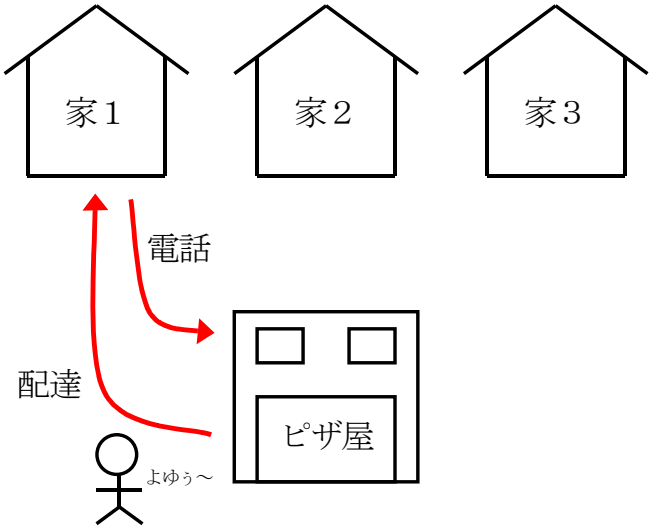
14.5.1 割り込みとは

(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。電話で注文が来れば、その家に届けれ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。

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、監視もれが起こります。

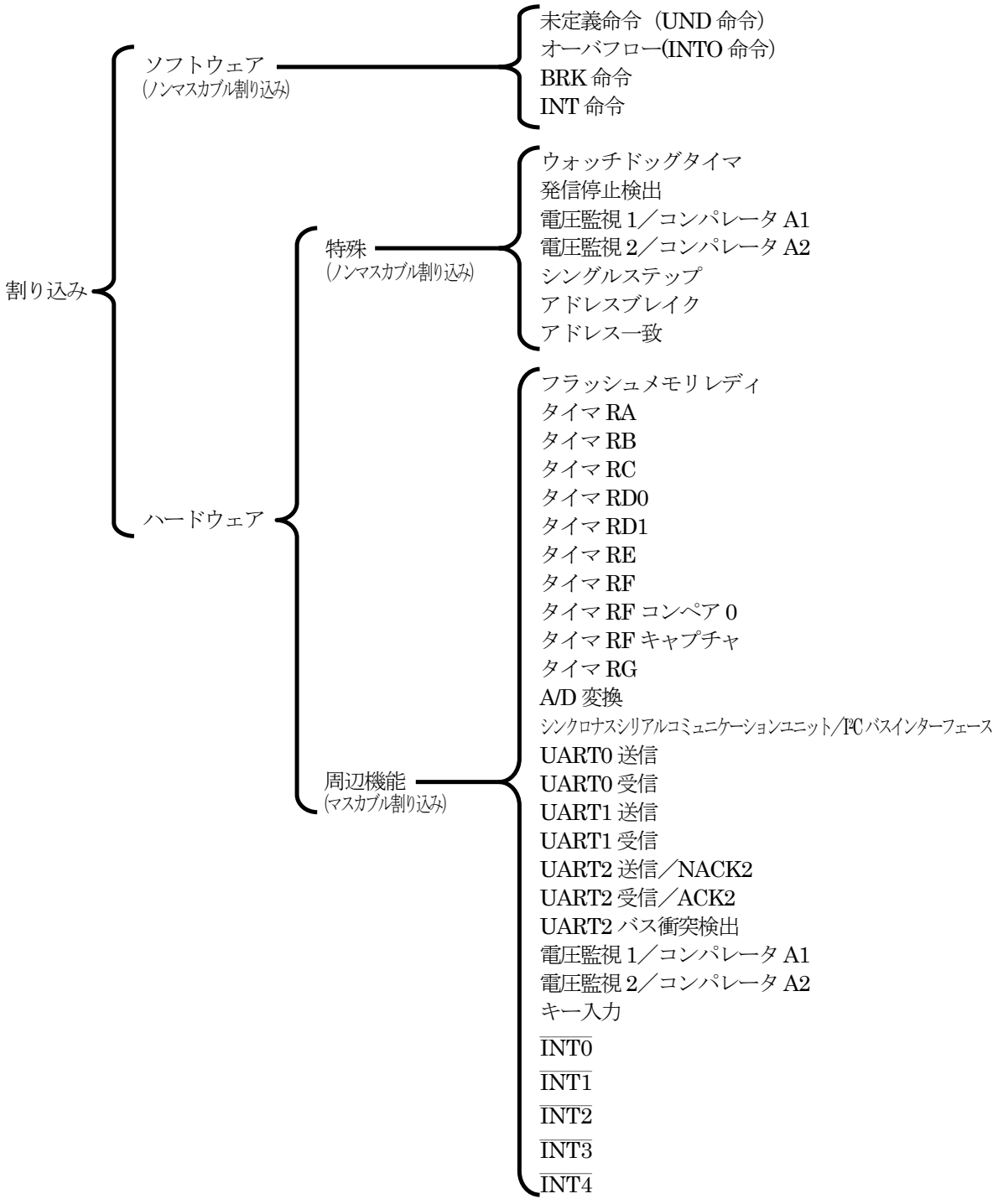
・電話で注文をうける
制御の用語（でもないですが）で「割り込み」といいます。電話のように、きっかけがあったときにだけ対処すれば良いので効率が良いです。ただし、電話の用意、電話の受け答えをする必要があります。

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。

人間の場合		マイコンの場合
電話を用意する	→	割り込みプログラムの設定をする
ベルが鳴る	→	割り込みが発生する
電話対応する	→	割り込みプログラムを実行する

(2) 割り込みの種類

R8C/38A の割り込みの種類を、下表に示します。



マスカブル割り込み	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**
ノンマスカブル割り込み	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。

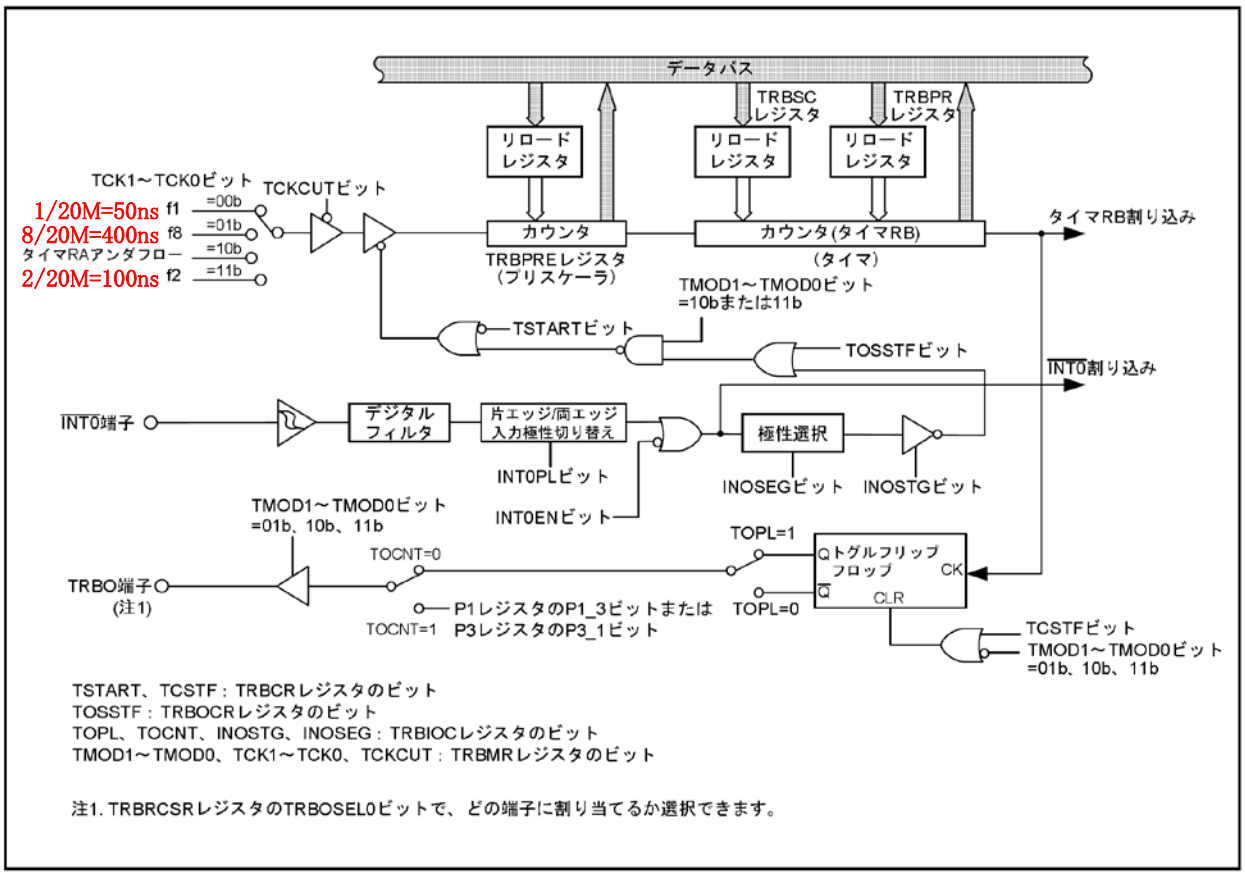
```
 83 :      /* タイマRBの設定 */
 84 :      /* 割り込み周期 = 1 / 20[MHz]    * (TRBPRE+1) * (TRBPR+1)
 85 :                     = 1 / (20*10^-6) * 200        * 100
 86 :                     = 0.001[s] = 1[ms]
 87 :      */
 88 :      trbmr  = 0x00;                    /* 動作モード、分周比設定       */
 89 :      trbpre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

(1) タイマ RB とは

R8C/38A 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。今回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。

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。

（2）タイマ RB のブロック図



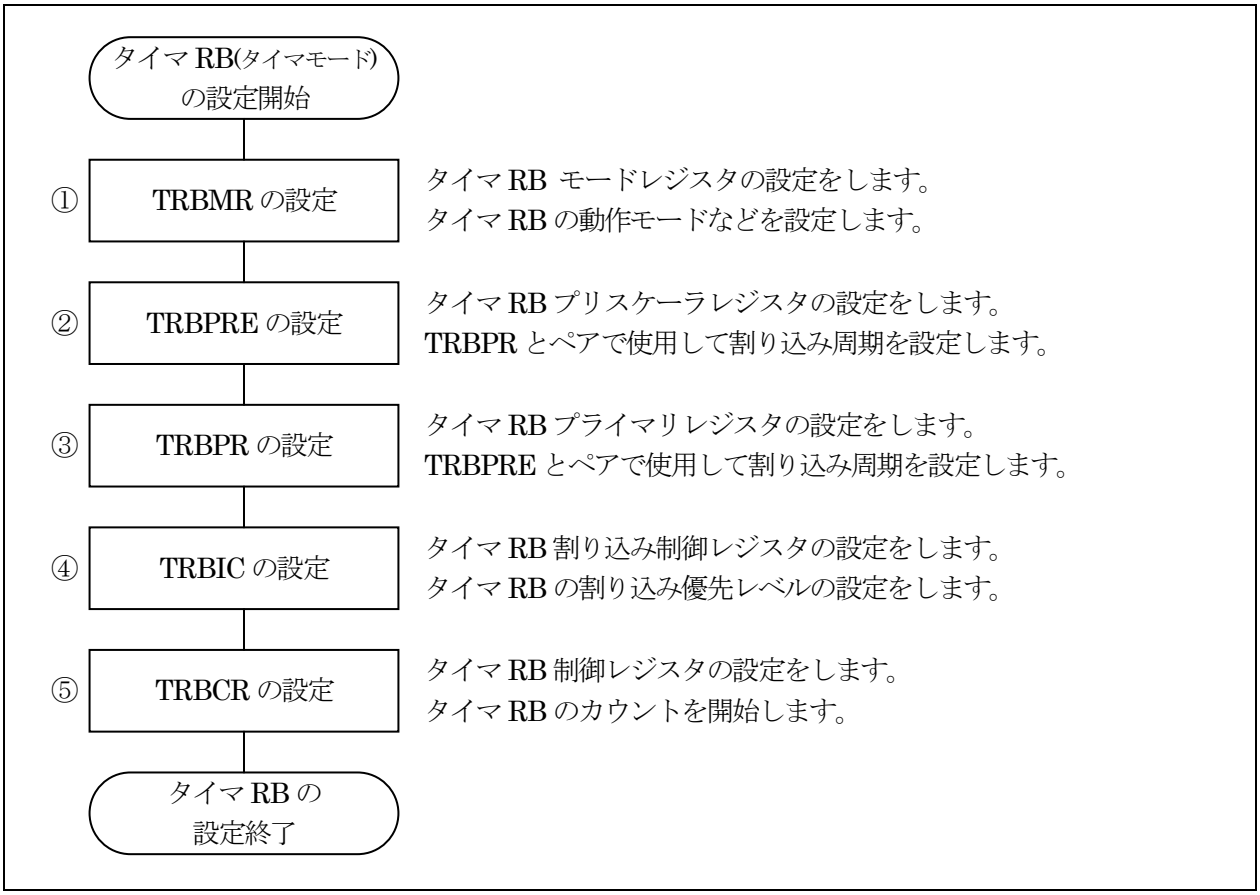
※タイマ RB の端子構成

端子名	割り当てる端子	入出力	機能
TRBO	P1_3 または P3_1	出力	パルス出力(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)

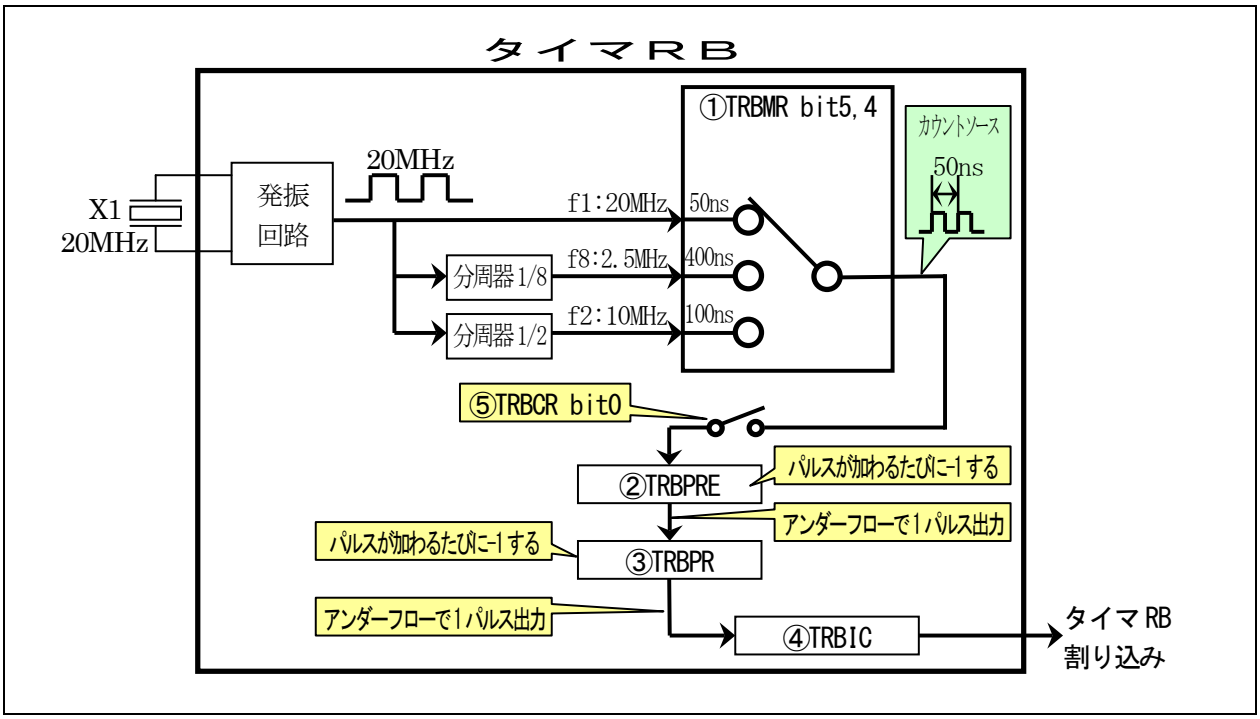
今回は、1ms ごとの割り込みを発生させるだけなので、端子は使いません。

(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。



タイマ RB タイマモードの概略図を下記に示します。



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

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。選択方法は次ページで説明します。	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 プリスケーラレジスタ(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	タイマ RB プリスケーラレジスタ(TRBPRE)がカウントダウンする間隔を、f1(50ns)にします。カウントダウンする間隔は、 f1/20MHz＝1/20MHz＝50ns となります。設定できる割り込み間隔の最大は、 50ns×65,536＝**3.2768ms** となります。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。
11	タイマ RB プリスケーラレジスタ(TRBPRE)がカウントダウンする間隔を、f2(100ns)にします。カウントダウンする間隔は、 f2/20MHz＝2/20MHz＝100ns となります。設定できる割り込み間隔の最大は、 100ns×65,536＝**6.5536ms** となります。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。
01	タイマ RB プリスケーラレジスタ(TRBPRE)がカウントダウンする間隔を、f8(400ns)にします。カウントダウンする間隔は、 f8/20MHz＝8/20MHz＝400ns となります。設定できる割り込み間隔の最大は、 400ns×65,536＝**26.2144ms** となります。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。

今回は、割り込み間隔を 1ms にします。タイマ RB モードレジスタ(TRBMR)の bit5,4 の設定について、
"00"を設定…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
"11"を設定…最大の割り込み間隔は 6.5536ms、今回設定したい 1ms の割り込み間隔を設定できる
"01"を設定…最大の割り込み間隔は 26.2144ms、今回設定したい 1ms の割り込み間隔を設定できる

2 つ以上設定できる場合は、割り込み間隔が小さい設定を選びます。今回は、"00"を設定します。

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	O				O			

②タイマ RB プリスケーラレジスタ（TRBPRE：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 プリスケーラレジスタ（TRBPRE）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。
TRBPRE と TRBPR の値を計算する式を、下記に示します。

タイマ RB 割り込み要求周期＝タイマ RB カウントソース×（TRBPRE＋1）×（TRBPR＋1）

TRBPRE と TRBPR を左辺に移動します。

（TRBPRE＋1）×（TRBPR＋1）＝タイマ RB 割り込み要求周期／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)のカウントソース選択ビット（bit5,4）で設定している内容で、今回は f1（50ns）です。よって、

（TRBPRE＋1）×（TRBPR＋1）＝タイマ RB 割り込み要求周期／タイマ RB カウントソース
（TRBPRE＋1）×（TRBPR＋1）＝（1×10^{-3}）　／（50×10^{-9}）
（TRBPRE＋1）×（TRBPR＋1）＝20,000
B　C　A

次の条件になるよう、A、B、C 部分を設定してください。

A…1～65,536 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 になるよう、値を設定してください。値は整数です。
C…1～256 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×C＝20,000
∴C＝100
となります。
B＝TRBPRE＋1　∴**TRBPRE**＝200－1＝**199**
C＝TRBPR＋1　∴**TRBPR**　＝100－1＝**99**
を設定します。

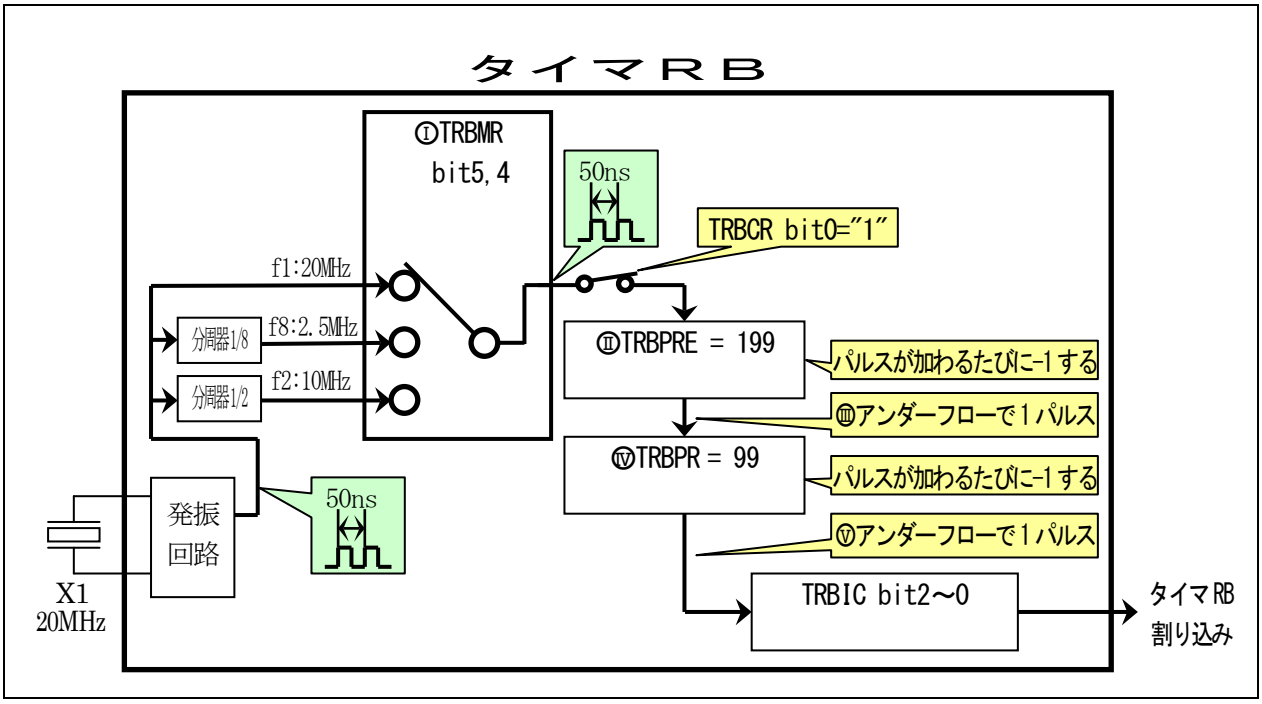
タイマ RB プリスケーラレジスタ（TRBPRE）の設定値を下記に示します。

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**							

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、次のパルスがタイマ RB プリスケーラレジスタに入力されます。 f1=1/20MHz=50ns
ⅱ	タイマ RB プリスケーラレジスタ(TRBPRE)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ます。今回は、199 を設定します。このレジスタは 50ns ごとに、「199→198→・・・→2→1→0」とカウントダウンしていきます。このように 0 も含めて数えるので、200 回カウントしたければ、1 小さい値の 199 を設定します。 0 をカウントダウンさせようとしても、次の値がないので(このレジスタは 0～255 の範囲しか設定できません)アンダーフロー(最小値を超えること)となります。アンダーフローになると、このレジスタに設定している値である 199 になります。 よって、「199→198→・・・→2→1→0→199→198・・・」と繰り返しカウントダウンしていくことになります。
ⅲ	タイマ RB プリスケーラレジスタ(TRBPRE)がアンダーフローに(0→199 に)なった瞬間、1 パルス出力されます。これは TRBPRE にパルスが 200 個入ると 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＝10μs
ⅳ	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し、1 つずつ値が減っていきます。今回は、99 を設定します。このレジスタは 10μs ごとに、「99→98→・・・→2→1→0」とカウントダウンしていきます。このように 0 も含めて数えるので、100 回カウントしたければ、1 小さい値の 99 を設定します。 アンダーフローになると、このレジスタに設定している値である 99 になります。
ⅴ	タイマ RB プライマリレジスタ(TRBPR)がアンダーフローに(0→99 に)なった瞬間、1 パルス出力されます。これは TRBPR にパルスが 100 個入ると 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。 10μs(入力されるパルスの間隔)×100 ＝ 1,000μs ＝ **1ms**

このように、タイマ RB プライマリレジスタ(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。

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 の割り込み優先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
bit7～4		″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、TRBPRE 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。

```
108 :  #pragma interrupt intTRB(vect=24)
109 :  void intTRB( void )
110 :  {
111 :      cnt_rb++;
112 :  }
```

108 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。
109 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。
111 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 されることになります。

※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は、下記のとおりです。

割り込み要因	ベクタ番号(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
BRK 命令(注 3)	+0～+3 (0000h～0003h)	0	
フラッシュメモリレディ	+4～+7 (0004h～0007h)		FMRDYIC
—(予約)		2～5	
INT4	+24～+27 (0018h～001Bh)	6	INT4IC
タイマ RC	+28～+31 (001Ch～001Fh)	7	TRCIC
タイマ RD0	+32～+35 (0020h～0023h)	8	TRD0IC
タイマ RD1	+36～+39 (0024h～0027h)	9	TRD1IC
タイマ RE	+40～+43 (0028h～002Bh)	10	TREIC
UART2 送信/NACK2	+44～+47 (002Ch～002Fh)	11	S2TIC
UART2 受信/ACK2	+48～+51 (0030h～0033h)	12	S2RIC
キー入力	+52～+55 (0034h～0037h)	13	KUPIC
A/D 変換	+56～+59 (0038h～003Bh)	14	ADIC
シンクロナスシリアルコミュニケーション ユニット/I2C バスインタフェース(注 2)	+60～+63 (003Ch～003Fh)	15	SSUIC/ IICIC
タイマ RF コンペア 1	+64～+67 (0040h～0043h)	16	CMP1IC
UART0 送信	+68～+71 (0044h～0047h)	17	S0TIC
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
タイマ RA	+88～+91 (0058h～005Bh)	22	TRAIC
— (予約)		23	
タイマ RB	**+96～+99 (0060h～0063h)**	**24**	**TRBIC**
INT1	+100～+103 (0064h～0067h)	25	INT1IC
INT3	+104～+107 (0068h～006Bh)	26	INT3IC
タイマ RF	+108～+111 (006Ch～006Fh)	27	TRFIC
タイマ RF コンペア 0	+112～+115 (0070h～0073h)	28	CMP0IC
INT0	+116～+119 (0074h～0077h)	29	INT0IC
UART2 バス衝突検出	+120～+123 (0078h～007Bh)	30	U2BCNIC
タイマ RF キャプチャ	+124～+127 (007Ch～007Fh)	31	CAPIC
ソフトウェア(注 3)	+128～+131 (0080h～0083h) ～ +164～+167 (00A4h～00A7h)	32～41	
— (予約)		42	
タイマ RG	+172～+175 (00ACh～00AFh)	43	TRGIC
— (予約)		44～49	
電圧監視 1/~~コンパレータ A1~~	+200～+203 (00C8h～00CBh)	50	VCMP1IC
電圧監視 2/~~コンパレータ A~~	+204～+207 (00CCh～00CFh)	51	VCMP2IC
— (予約)		52～55	
ソフトウェア(注 3)	+224～+227 (00E0h～00E3h) ～ +252～+255 (00FCh～00FFh)	56～63	

R8C/38C で削除された機能です。

注 1. INTB レジスタが示す番地からの相対番地です。
注 2. SSUIICSR 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。

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インの設定により、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。

```
 95 :  /************************************************************************/
 96 :  /* タイマ本体                                                           */
 97 :  /* 引数    タイマ値 1=1ms                                               */
 98 :  /************************************************************************/
 99 :  void timer( unsigned long timer_set )
100 :  {
101 :      cnt_rb = 0;
102 :      while( cnt_rb < timer_set );
103 :  }
```

(1) timer 関数の使い方

timer 関数の使い方を下記に示します。

```
    timer( 時間稼ぎをしたい時間 );
```

引数は、時間稼ぎをしたい時間をミリ秒単位で設定します。使用例を下記に示します。

```
    p2 = 0x55;          ←数μs
    timer( 1000 );      ←1000ms
```

ポート 2 に値を出力してから、timer 関数で 1000ms 待ちます。

(2) プログラムの解説

「timer(1000);」を実行したとして、説明します。

<table>
<tr><td>101 行</td><td colspan="2">cnt_rb を 0 にクリアします。</td></tr>
<tr><td rowspan="4">102 行</td><td>①</td><td>まず下記プログラムが実行されます。

while(cnt_rb < timer_set);

cnt_rb 変数は 101 行で 0 にクリアされています。timer_set 変数は関数の引数です。よって、

while(0 < 1000);

となります。カッコの中が成り立ちませんので、この命令を繰り返し続けます。</td></tr>
<tr><td>②</td><td>1ms 後、intRB 割り込みが発生します。現在実行している while 文の実行を止めて、割り込みプログラムである命令が実行されます。

111 ： cnt_rb++; ← 0++なので、1になる

よって、cnt_rb は 1 になります。割り込みプログラムは終了すると、割り込みが発生した行に戻ります。</td></tr>
<tr><td>③</td><td>戻った行は、while 文です。

while(1 < 1000);

cnt_rb 変数が割り込みプログラム内でプラスされたので 1 になりました。まだカッコの中が成り立ちませんので、この命令を繰り返し続けます。</td></tr>
<tr><td>④</td><td>②を 1000 回実行すると、cnt_rb は 1000 になります。

while(1000 < 1000);

while 文の条件式が成り立たなくなりました。while 文を終えて次の行に行きます。次の行は無いので、timer 関数を終了します。intRB 割り込みは 1ms ごとに実行されます。1000 になるまで while 文を繰り返し続けたと言うことは、1000ms の間、繰り返し続けたということになります。
このように、タイマ RB による 1ms ごとの割り込みを実行した回数を timer 関数で数えて、時間を計ります。</td></tr>
</table>

14.5.5 main 関数

```
 38 :  void main( void )
 39 :  {
 40 :      init();                          /* 初期化                       */
 41 :      asm(" fset I ");                 /* 全体の割り込み許可           */
 42 :
 43 :      while( 1 ) {
 44 :          p2 = 0x55;
 45 :          timer( 1000 );
 46 :          p2 = 0xaa;
 47 :          timer( 1000 );
 48 :          p2 = 0x00;
 49 :          timer( 1000 );
 50 :      }
 51 :  }
```

41 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。

```
; 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,_p2_addr
        push.w  #0000H
        push.w  #03e8H
        jsr     _timer
        add.b   #04H,SP
        mov.b   #0aaH,_p2_addr
                                (右上へ続く)
```

```
(左下から続き)
        push.w  #0000H
        push.w  #03e8H
        jsr     _timer
        add.b   #04H,SP
        mov.b   #00H,_p2_addr
        push.w  #0000H
        push.w  #03e8H
        jsr     _timer
        add.b   #04H,SP
        jmp     L1
L3:
```

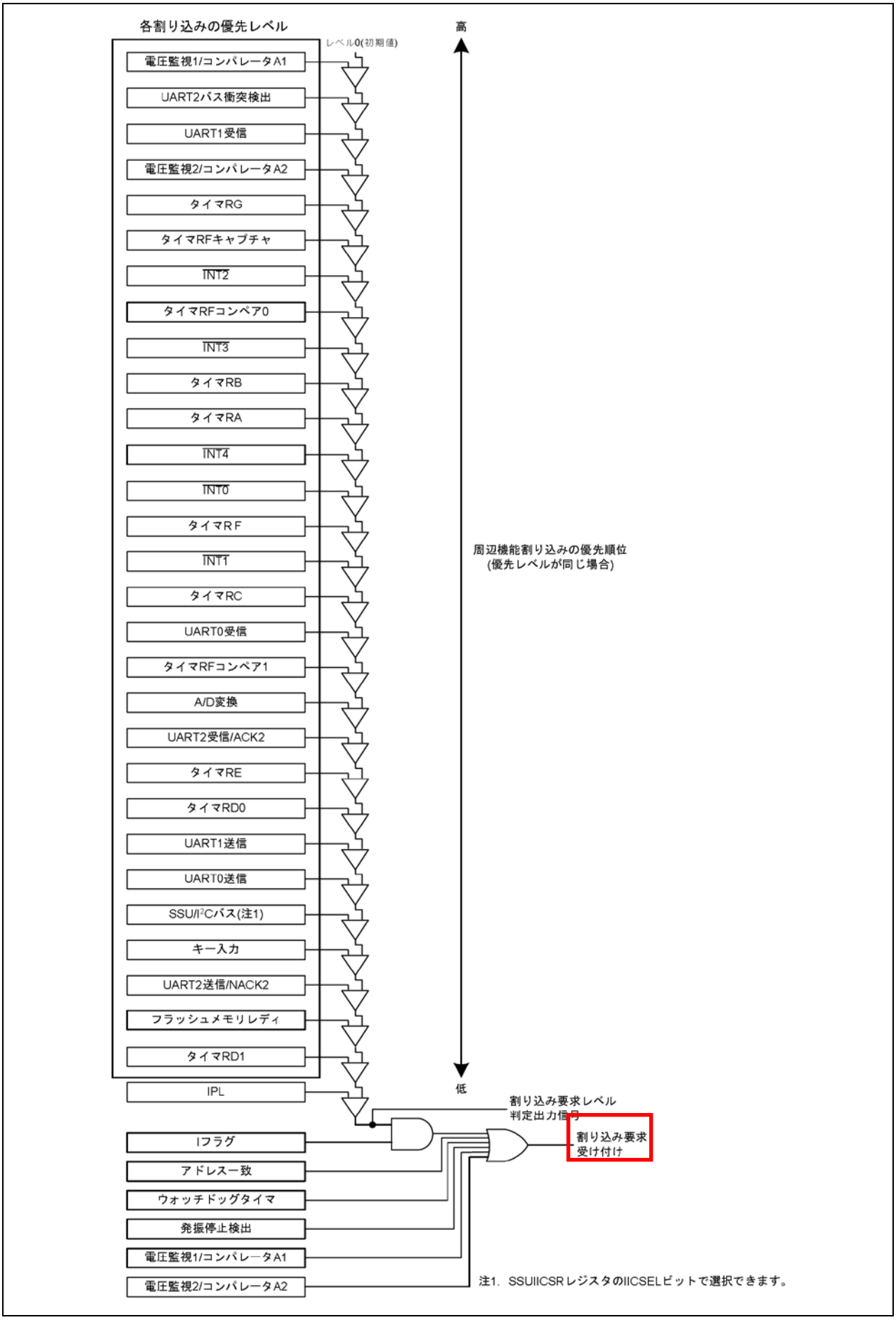
割り込みがあるかどうかは、アセンブリ言語の命令を 1 行終了後、毎回チェックが行われます。
イメージを下記に示します。

```
; 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,_p2_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,_p2_addr
        割り込み要求があるかチェック
                                (右上へ続く)
```

```
(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,_p2_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:
```

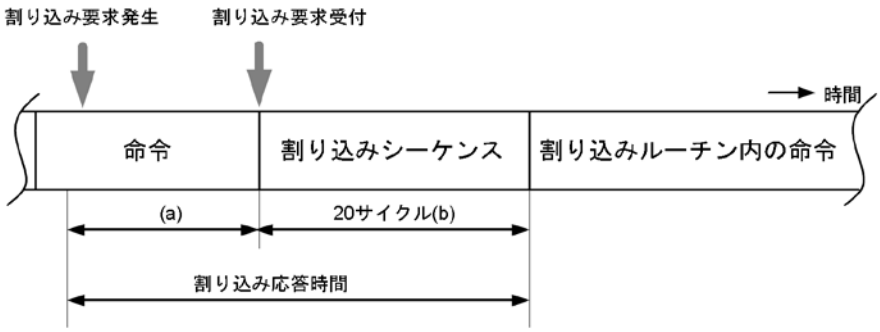
※「割り込み要求があるかチェック」の実行時間は、0 秒です。

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。



「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。



(a)　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。実行している命令によって異なります。この時間が最も長くなる命令はDIVX命令で30サイクル(ウェイトなし、除数がレジスタの場合のサイクル数)です。
(b)　アドレス一致割り込み、シングルステップ割り込みは21サイクルです。

RY_R8C38 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、

1/20MHz＝50ns

です。割り込みシーケンスの時間は、20 サイクルですので、

割り込みシーケンスの時間 ＝ 1 サイクル×20 ＝ 50ns×20 ＝ 1000ns ＝ 1μs

となります。

「**割り込みシーケンス＝割り込みプログラムを実行する前準備**」です。要は、**割り込みが発生するまでの準備に、1μs の時間がかかるということになります**。

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（上図の(a)の部分）の、次の命令から実行を再開します。

14.6 演習

(1) タイマ RB の割り込み間隔を 100μs にして、動作確認しなさい。ただし、timer 関数がある 45 行、47 行、49 行は 1 秒にすること（「timer(10000)」にする）。

(2) タイマ RB の割り込み間隔を 10μs にして、動作確認しなさい。ただし、timer 関数がある 45 行、47 行、49 行は 1 秒にすること（「timer(100000)」にする）。

(3) タイマ RB の割り込み間隔を 1μs にして、動作確認しなさい。ただし、timer 関数がある 45 行、47 行、49 行は 1 秒にすること（「timer(1000000)」にする）。

※(3)は、正しくプログラムを記述しても動作しません。これは、割り込みプログラムを終了した時点で次の割り込みが発生しているため、main 関数のプログラムを実行せずにすぐに割り込みプログラムを実行します。このように、main 関数のプログラムは実行されず、割り込み関数のプログラムだけ実行され続けます。

※(2)も、割り込みプログラムの実行に 10μs 以上の時間がかかる様なプログラムの場合は、(3)と同様に main 関数は実行されません。割り込みプログラムはできるだけ簡潔に記述してください。

15.　7 セグメント LED の表示（プロジェクト：seven_seg）

15.1　概要

本章では、7 セグメント LED（2 桁）を表示させる方法を説明します。回路的には、1 桁ずつしか点灯させることができません。そこで、1ms ごとに割り込みを発生させ、1ms ごとに表示する桁を変える方法を紹介します。

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。

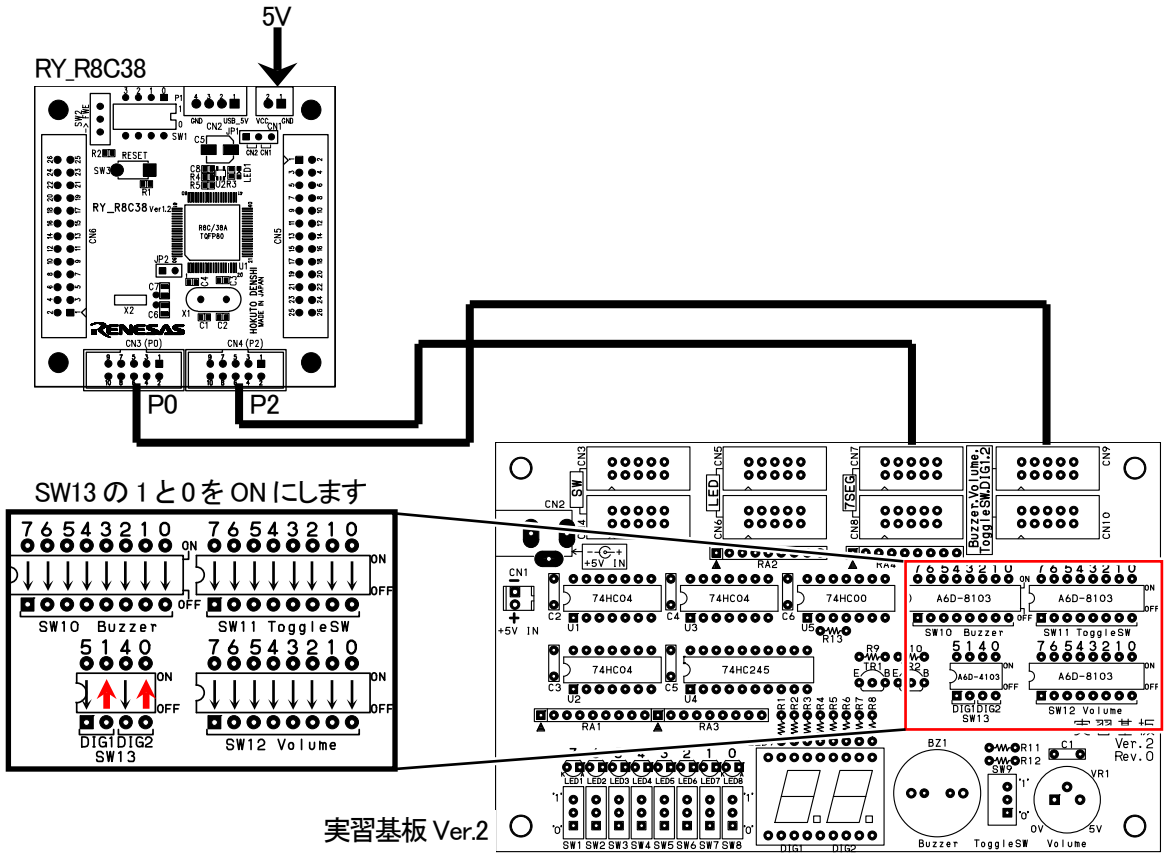
15.2　接続

■使用ポート

マイコンのポート	接続内容
P2 (CN4)	実習基板 Ver.2 の 7 セグメント LED 部に接続します。
P0 (CN3)	実習基板 Ver.2 の 7 セグメント LED 桁選択部に接続します。

■接続例

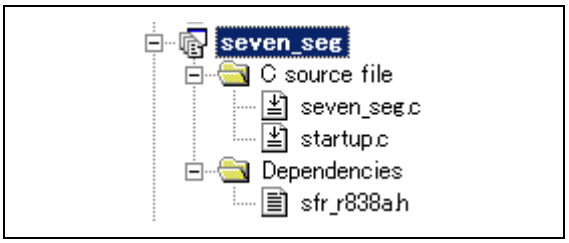
実習基板 Ver.2 を使ったときの接続例を下記に示します。



■操作方法

操作は特にありません。電源を入れると 7 セグメント LED が表示されます。数字の変化をよく観察してください。

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。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥seven_seg¥startup.c
2	seven_seg.c	実際に制御するプログラムが書かれています。R8C/38A の内蔵周辺機能(SFR)の初期化も行います。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥seven_seg¥seven_seg.c
3	sfr_r838a.h	R8C/38A マイコンの内蔵周辺機能を制御するためのレジスタ（Special Function Registers)を定義したファイルです。 ファイルの位置→C:¥Workspace¥common_r8c38a¥sfr_r838a.h

15.4 プログラム「seven_seg.c」

```
/************************************************************************/
/* 対象マイコン  R8C/38A                                                */
/* ﾌｧｲﾙ内容      7セグメントLEDの制御                                   */
/* バージョン    Ver.1.00                                                */
/* Date          2010.04.01                                              */
/* Copyright     ジャパンマイコンカーラリー実行委員会                   */
/************************************************************************/
/*
出力：P2_7-P2_0(実習基板Ver.2の7セグメントLED部)
      P0_1,P0_1(実習基板Ver.2の7セグメントLED桁選択部)

7セグメントLEDを制御します。
1桁ずつしか表示できないので、1msごとに割り込みで桁を切り替えて
2桁分を表示します。
*/

/*====================================*/
/* インクルード                       */
/*====================================*/
#include "sfr_r838a.h"                  /* R8C/38A SFRの定義ファイル     */

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

/*====================================*/
/* プロトタイプ宣言                   */
/*====================================*/
void init( void );
void timer( unsigned long timer_set );

/*====================================*/
/* グローバル変数の宣言               */
/*====================================*/
unsigned long cnt_rb;                   /* タイマRB用                    */
unsigned long cnt_7seg;                 /* 7セグメントLED表示用          */
```

```
 37 :  unsigned char seg_out;                  /* 7セグメントLEDに表示する値   */
 38 :
 39 :  const unsigned char seg_data[] = {
 40 :  /*     0     1     2     3     4     5     6     7     8     9 */
 41 :      0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f, 0x6f
 42 :  };
 43 :
 44 :  /************************************************************************/
 45 :  /* メインプログラム                                                     */
 46 :  /************************************************************************/
 47 :  void main( void )
 48 :  {
 49 :      init();                           /* 初期化                         */
 50 :      asm(" fset I ");                  /* 全体の割り込み許可             */
 51 :
 52 :      while( 1 ) {
 53 :      }
 54 :  }
 55 :
 56 :  /************************************************************************/
 57 :  /* R8C/38A 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 :      cm13  = 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;                         /* PD0のプロテクト解除            */
 73 :      pd0 = 0xff;                       /* 1:7セグDIG1 0:7セグDIG2        */
 74 :      pd1 = 0xd0;                       /* 5:RXD0 4:TXD0 3-0:DIP SW       */
 75 :      pd2 = 0xff;                       /* 7-0:7セグメントLED             */
 76 :      pd3 = 0xff;                       /*                                */
 77 :      p4  = 0x20;                       /* P4_5のLED:初期は点灯           */
 78 :      pd4 = 0xb8;                       /* 7:XOUT 6:XIN 5:LED 2:VREF      */
 79 :      pd5 = 0xff;                       /*                                */
 80 :      pd6 = 0xff;                       /*                                */
 81 :      pd7 = 0xff;                       /*                                */
 82 :      pd8 = 0xff;                       /*                                */
 83 :      pd9 = 0x3f;                       /*                                */
 84 :      pur0 = 0x04;                      /* P1_3～P1_0のプルアップON       */
 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 :      trbmr  = 0x00;                    /* 動作モード、分周比設定         */
 92 :      trbpre = 200-1;                   /* プリスケーラレジスタ           */
 93 :      trbpr  = 100-1;                   /* プライマリレジスタ             */
 94 :      trbic  = 0x07;                    /* 割り込み優先レベル設定         */
 95 :      trbcr  = 0x01;                    /* カウント開始                   */
 96 :  }
 97 :
 98 :  /************************************************************************/
 99 :  /* タイマ本体                                                           */
100 :  /* 引数   タイマ値 1=1ms                                                */
101 :  /************************************************************************/
102 :  void timer( unsigned long timer_set )
103 :  {
104 :      cnt_rb = 0;
105 :      while( cnt_rb < timer_set );
106 :  }
107 :
108 :  /************************************************************************/
109 :  /* タイマRB 割り込み処理                                                */
110 :  /************************************************************************/
111 :  #pragma interrupt intTRB(vect=24)
112 :  void intTRB( void )
113 :  {
114 :      int i;
115 :
116 :      cnt_rb++;
117 :
118 :      // 7セグメントLEDの表示値の更新
119 :      cnt_7seg++;
120 :      if( cnt_7seg >= 100 ) {
121 :          // 100msに1回実行
122 :          cnt_7seg = 0;
123 :          seg_out++;
124 :          if( seg_out >= 100 ) seg_out = 0;
125 :      }
126 :
```

```
127 :      // 7セグメントLED 表示処理
128 :      if( p0 & 0x01 ) {
129 :          // 10の桁(DIG1)表示
130 :          p0 &= 0xfe;
131 :          i = seg_out / 10;
132 :          p2 = seg_data[ i ];
133 :          p0 |= 0x02;
134 :      } else {
135 :          // 1の桁(DIG2)表示
136 :          p0 &= 0xfd;
137 :          i = seg_out % 10;
138 :          p2 = seg_data[ i ];
139 :          p0 |= 0x01;
140 :      }
141 :  }
142 :
143 :  /************************************************************************/
144 :  /* end of file                                                          */
145 :  /************************************************************************/
```

15.5 プログラムの解説

15.5.1 変数領域

```
32 :  /*======================================*/
33 :  /* グローバル変数の宣言                 */
34 :  /*======================================*/
35 :  unsigned long cnt_rb;                   /* タイマRB用                   */
36 :  unsigned long cnt_7seg;                 /* 7セグメントLED表示用         */
37 :  unsigned char seg_out;                  /* 7セグメントLEDに表示する値   */
38 :
39 :  const unsigned char seg_data[] = {
40 :  /*     0     1     2     3     4     5     6     7     8     9 */
41 :      0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f, 0x6f
42 :  };
```

35 行～37 行で、グローバル変数の宣言をしています。

cnt_rb	timer 関数で使う変数です。
cnt_7seg	seg_out 変数を+1 する時間をカウントする変数です。この変数は 1ms ごとに+1 します。今回のプログラムでは、cnt_7seg 変数が 100 になったら、seg_out 変数を+1 します。すなわち、seg_out 変数は 100ms ごとに+1 します。
seg_out	7 セグメント LED の表示する値です。0～99 の範囲です。100ms ごとに+1 されます。99 の次は 0 になります。

39 行～42 行で、7 セグメント LED へ出力するセグメントデータの配列を定義しています。

seg_data[]	7 セグメント LED への表示データを納めた配列です。「0」を表示するとき、0x3f を出力しなければ「0」と表示されません。この配列で、表示する値から 7 セグメント LED に出力する値に変換します。 const 指定をすると、ROM に格納するという意味になり、配列の内容を変更することはできません。配列の内容を変更したい場合(RAM に格納したい場合）は、「const」を取ります。ただし、RAM は容量が限られていますので、内容を変更しない場合は const 指定しておきます。

15.5.2 intTRB 関数(1ms ごとの割り込み)

割り込みの中で、7 セグメント LED の表示処理をしています。

```
111 :  #pragma interrupt intTRB(vect=24)
112 :  void intTRB( void )
113 :  {
114 :      int i;
115 :
116 :      cnt_rb++;
117 :
118 :      // 7セグメントLEDの表示値の更新
119 :      cnt_7seg++;
120 :      if( cnt_7seg >= 100 ) {
121 :          // 100msに1回実行
122 :          cnt_7seg = 0;
123 :          seg_out++;
124 :          if( seg_out >= 100 ) seg_out = 0;
125 :      }
126 :
127 :      // 7セグメントLED 表示処理
128 :      if( p0 & 0x01 ) {
129 :          // 10の桁(DIG1)表示
130 :          p0 &= 0xfe;
131 :          i = seg_out / 10;
132 :          p2 = seg_data[ i ];
133 :          p0 |= 0x02;
134 :      } else {
135 :          // 1の桁(DIG2)表示
136 :          p0 &= 0xfd;
137 :          i = seg_out % 10;
138 :          p2 = seg_data[ i ];
139 :          p0 |= 0x01;
140 :      }
141 :  }
```

111 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、タイマ RB の割り込みは、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。
119～ 125 行	seg_out 変数を 100ms ごとに+1 します。99 の次は 0 に戻します。 割り込みは、1ms ごとに発生します。cnt_7seg 変数は割り込みごとに+1 して、100 になったなら seg_out 変数を+1 すれば、seg_out 変数は 100ms ごとに+1 することになります。
128 行	ポート 0 の bit0 が"1"なら、DIG2(1 の桁)が表示されているので、129 行～133 行を実行して DIG1(10 の桁)を表示する処理をします。"0"なら、DIG1(10 の桁)が表示されている(もしくは、両方とも表示されていない)として、135 行～139 行を実行して DIG2(1 の桁)を表示する処理をします。

129 行
～133 行

DIG1(10 の桁)を表示する処理です。
まず、130 行で下表のビット演算を行い、P0_0 端子を"0"にして、現在表示している 1 の桁の表示を OFF にします。

bit	7	6	5	4	3	2	1	0
元の値 (ポート 0)	P0_7	P0_6	P0_5	P0_4	P0_3	P0_2	P0_1	P0_0
AND 値	1	1	1	1	1	1	1	**0**
結果	P0_7	P0_6	P0_5	P0_4	P0_3	P0_2	P0_1	**0**

131 行で、seg_out の値を 10 で割り、結果を変数 i に代入します。変数 i には、seg_out の 10 の桁の値が代入されます。
132 行で、0～9 の値を、7 セグメント LED へ出力する値に変換して、ポート 2 へ出力します。
133 行で下表のビット演算を行い、P0_1 端子を"1"にします。

bit	7	6	5	4	3	2	1	0
元の値 (ポート 0)	P0_7	P0_6	P0_5	P0_4	P0_3	P0_2	P0_1	P0_0
OR	0	0	0	0	0	0	**1**	0
結果	P0_7	P0_6	P0_5	P0_4	P0_3	P0_2	**1**	P0_0

DIG1(10 の桁)の 7 セグメント LED が表示されます。表示される値は、132 行で出力した内容です。

136～
139 行

DIG2(1 の桁)を表示する処理です。
まず、136 行で下表のビット演算を行い、P0_1 端子を"0"にして、現在表示している 10 の桁の表示を OFF にします。

bit	7	6	5	4	3	2	1	0
元の値 (ポート 0)	P0_7	P0_6	P0_5	P0_4	P0_3	P0_2	P0_1	P0_0
AND 値	1	1	1	1	1	1	**0**	1
結果	P0_7	P0_6	P0_5	P0_4	P0_3	P0_2	**0**	P0_0

137 行で、seg_out の値を 10 で割った余りを変数 i に代入します。変数 i には、seg_out の 1 の桁の値が代入されます。
138 行で、0～9 の値を、7 セグメント LED へ出力する値に変換して、ポート 2 へ出力します。
139 行で下表のビット演算を行い、P0_0 端子を"1"にします。

bit	7	6	5	4	3	2	1	0
元の値 (ポート 0)	P0_7	P0_6	P0_5	P0_4	P0_3	P0_2	P0_1	P0_0
OR	0	0	0	0	0	0	0	**1**
結果	P0_7	P0_6	P0_5	P0_4	P0_3	P0_2	P0_1	**1**

DIG2(1 の桁)の 7 セグメント LED が表示されます。表示される値は、138 行で出力した内容です。

※7 セグメント LED の表示データと配列

seg_data 配列には、7 セグメント LED の表示データが格納されています。添字部分に表示したい数字を入れると、表示データが返ってきます。添字は 0 から指定します。

表示したい値	プログラム指定内容	戻り値
0	seg_data[0]	0x3f
1	seg_data[1]	0x06
2	seg_data[2]	0x5b
3	seg_data[3]	0x4f
4	seg_data[4]	0x66
5	seg_data[5]	0x6d
6	seg_data[6]	0x7d
7	seg_data[7]	0x07
8	seg_data[8]	0x7f
9	seg_data[9]	0x6f

15.5.3 main 関数

```
47 :  void main( void )
48 :  {
49 :      init();                          /* 初期化                    */
50 :      asm(" fset I ");                 /* 全体の割り込み許可        */
51 :
52 :      while( 1 ) {
53 :      }
54 :  }
```

50 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 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。
52 行 ～53 行	7 セグメント LED の表示処理は割り込み内で行っているため、main 関数では何もしません。

15.6 演習

(1) RY_R8C38 ボードのディップスイッチ値(0～15)を、7 セグメント LED に表示するようにしなさい。

(2) 「a」～「f」までの 7 セグメント LED データを作り、「00」～「ff」(10 進数で 0～255)まで 16 進数表記で 0.1 秒ごとに表示内容が＋1するようにしなさい。

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、$\overline{\text{INT3}}$端子を P3_3 に割り当てて、割り込みを発生させます。

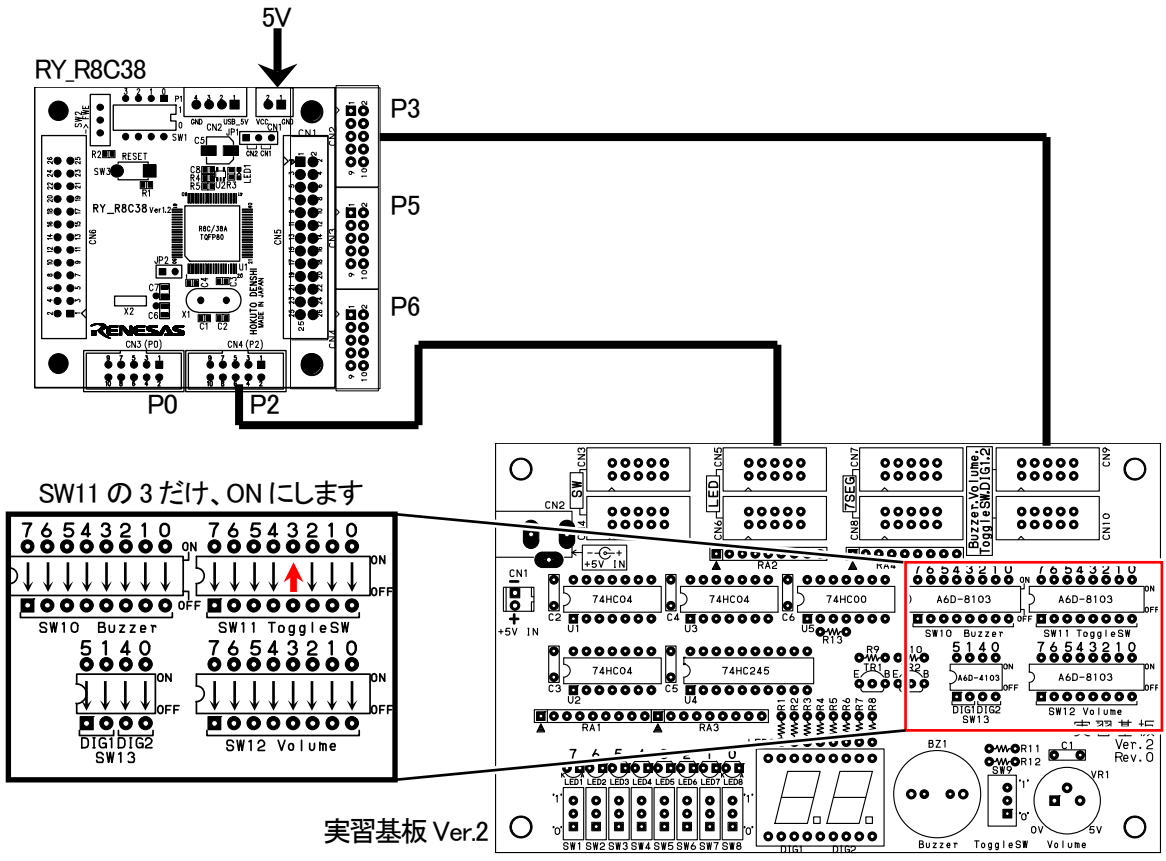
16.2　接続

■使用ポート

マイコンのポート	接続内容
P2 (CN4)	実習基板 Ver.2 の LED 部など、出力機器を接続します。
P3_3 (変換基板 CN2)	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。
P4_5	RY_R8C38 ボードの LED です。

■接続例

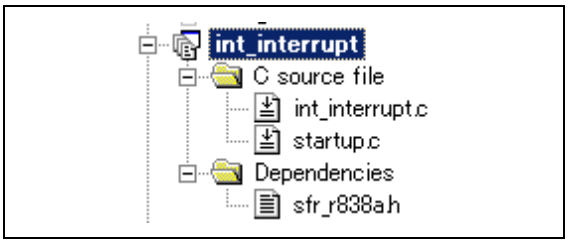
実習基板 Ver.2 を使ったときの接続例を下記に示します。



■操作方法

実習基板 Ver.2 のトグルスイッチ SW9 を上下させると、実習基板 Ver.2 の LED1～LED9 の値が「"0000 0000"→"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、RY_R8C38 ボードの LED が 1 秒ごとに点灯します。

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。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥int_interrupt¥startup.c
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/38A の内蔵周辺機能(SFR)の初期化も行います。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥int_interrupt¥int_interrupt.c
3	sfr_r838a.h	R8C/38A マイコンの内蔵周辺機能を制御するためのレジスタ(Special Function Registers)を定義したファイルです。 ファイルの位置→C:¥Workspace¥common_r8c38a¥sfr_r838a.h

16.4 プログラム「int_interrupt.c」

```
/************************************************************************/
/* 対象マイコン  R8C/38A                                                */
/* ﾌｧｲﾙ内容      INT割り込み                                            */
/* バージョン    Ver.1.00                                               */
/* Date          2010.04.01                                             */
/* Copyright     ジャパンマイコンカーラリー実行委員会                   */
/************************************************************************/
/*
入力：INT3(P3_3)(実習基板Ver.2のトグルスイッチ部などチャタリングのない信号)
出力：P2_7-P2_0(実習基板Ver.2のLED部)

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。
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。
INT3割り込みの発生ごとに、実習基板Ver.2のLED部が＋１していきます。
メイン関数では、マイコンボード上のLEDを点滅させます。
*/

/*====================================*/
/* インクルード                       */
/*====================================*/
#include "sfr_r838a.h"                  /* R8C/38A SFRの定義ファイル     */

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

/*====================================*/
/* プロトタイプ宣言                   */
/*====================================*/
void init( void );
void timer( unsigned long timer_set );
void led_out_m( unsigned char led );
```

```
 33 :
 34 :  /*====================================*/
 35 :  /* グローバル変数の宣言               */
 36 :  /*====================================*/
 37 :  unsigned char cnt_int3;                /* INT3割り込みごとに+1        */
 38 :
 39 :  /************************************************************************/
 40 :  /* メインプログラム                                                     */
 41 :  /************************************************************************/
 42 :  void main( void )
 43 :  {
 44 :      init();                           /* 初期化                       */
 45 :      asm(" fset I ");                  /* 全体の割り込み許可           */
 46 :
 47 :      while( 1 ) {
 48 :          led_out_m( 1 );
 49 :          timer( 1000 );
 50 :          led_out_m( 0 );
 51 :          timer( 1000 );
 52 :      }
 53 :  }
 54 :
 55 :  /************************************************************************/
 56 :  /* R8C/38A スペシャルファンクションレジスタ(SFR)の初期化                */
 57 :  /************************************************************************/
 58 :  void init( void )
 59 :  {
 60 :      int i;
 61 :
 62 :      /* クロックをXINクロック(20MHz)に変更 */
 63 :      prc0  = 1;                        /* プロテクト解除               */
 64 :      cm13  = 1;                        /* P4_6,P4_7をXIN-XOUT端子にする*/
 65 :      cm05  = 0;                        /* XINクロック発振              */
 66 :      for(i=0; i<50; i++ );             /* 安定するまで少し待つ(約10ms) */
 67 :      ocd2  = 0;                        /* システムクロックをXINにする  */
 68 :      prc0  = 0;                        /* プロテクトON                 */
 69 :
 70 :      /* ポートの入出力設定 */
 71 :      prc2 = 1;                         /* PD0のプロテクト解除          */
 72 :      pd0 = 0x00;                       /*                              */
 73 :      pd1 = 0xd0;                       /* 5:RXD0 4:TXD0 3-0:DIP SW     */
 74 :      pd2 = 0xff;                       /* 7-0:LED                      */
 75 :      pd3 = 0xf7;                       /* 3:パルス入力                 */
 76 :      p4  = 0x00;                       /* P4_5のLED:初期は消灯         */
 77 :      pd4 = 0xb8;                       /* 7:XOUT 6:XIN 5:LED 2:VREF    */
 78 :      pd5 = 0xff;                       /*                              */
 79 :      pd6 = 0xff;                       /*                              */
 80 :      pd7 = 0xff;                       /*                              */
 81 :      pd8 = 0xff;                       /*                              */
 82 :      pd9 = 0x3f;                       /*                              */
 83 :      pur0 = 0x04;                      /* P1_3～P1_0のプルアップON     */
 84 :
 85 :      /* INT0～4割り込み設定(今回はINT3を設定) */
 86 :      intsr  = 0x00;                    /* INT1～3の入力端子設定        */
 87 :      inten  = 0x40;                    /* INT0～3の外部入力許可設定    */
 88 :      inten1 = 0x00;                    /* INT4の外部入力許可設定       */
 89 :      intf   = 0xc0;                    /* INT0～3の入力フィルタ選択    */
 90 :      intf1  = 0x00;                    /* INT4の入力フィルタ選択       */
 91 :
 92 :      int0ic = 0x00;                    /* INT0割り込み優先レベル設定   */
 93 :      int1ic = 0x00;                    /* INT1割り込み優先レベル設定   */
 94 :      int2ic = 0x00;                    /* INT2割り込み優先レベル設定   */
 95 :      int3ic = 0x07;                    /* INT3割り込み優先レベル設定   */
 96 :      int4ic = 0x00;                    /* INT4割り込み優先レベル設定   */
 97 :  }
 98 :
 99 :  /************************************************************************/
100 :  /* マイコンボードのLED出力                                              */
101 :  /* 引数   0:消灯 1:点灯                                                 */
102 :  /************************************************************************/
103 :  void led_out_m( unsigned char led )
104 :  {
105 :      if( led ) {
106 :          p4 |= 0x20;                   /* LED点灯                      */
107 :      } else {
108 :          p4 &= 0xdf;                   /* LED消灯                      */
109 :      }
110 :  }
111 :
112 :  /************************************************************************/
113 :  /* タイマ本体                                                           */
114 :  /* 引数   タイマ値 1=1ms                                                */
115 :  /************************************************************************/
116 :  void timer( unsigned long timer_set )
117 :  {
118 :      int i;
119 :
120 :      do {
121 :          for( i=0; i<1240; i++ );
122 :      } while( timer_set-- );
123 :  }
```

```
124 :
125 :  /************************************************************************/
126 :  /* INT3 割り込み処理                                                    */
127 :  /************************************************************************/
128 :  #pragma interrupt intINT3(vect=26)
129 :  void intINT3( void )
130 :  {
131 :      cnt_int3++;
132 :      p2 = cnt_int3;
133 :  }
134 :
135 :  /************************************************************************/
136 :  /* end of file                                                          */
137 :  /************************************************************************/
```

16.5 プログラムの解説

今回は、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(INT3割り込みの設定)

```
 85 :      /* INT0～4割り込み設定(今回はINT3を設定) */
 86 :      intsr  = 0x00;                  /* INT1～3の入力端子設定          */
 87 :      inten  = 0x40;                  /* INT0～3の外部入力許可設定      */
 88 :      inten1 = 0x00;                  /* INT4の外部入力許可設定         */
 89 :      intf   = 0xc0;                  /* INT0～3の入力フィルタ選択      */
 90 :      intf1  = 0x00;                  /* INT4の入力フィルタ選択         */
 91 :
 92 :      int0ic = 0x00;                  /* INT0割り込み優先レベル設定     */
 93 :      int1ic = 0x00;                  /* INT1割り込み優先レベル設定     */
 94 :      int2ic = 0x00;                  /* INT2割り込み優先レベル設定     */
 95 :      int3ic = 0x07;                  /* INT3割り込み優先レベル設定     */
 96 :      int4ic = 0x00;                  /* INT4割り込み優先レベル設定     */
```

(1) INT0～INT4割り込み

INT0～INT4割り込みとは、INT0～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) INT0～INT4割り込み端子

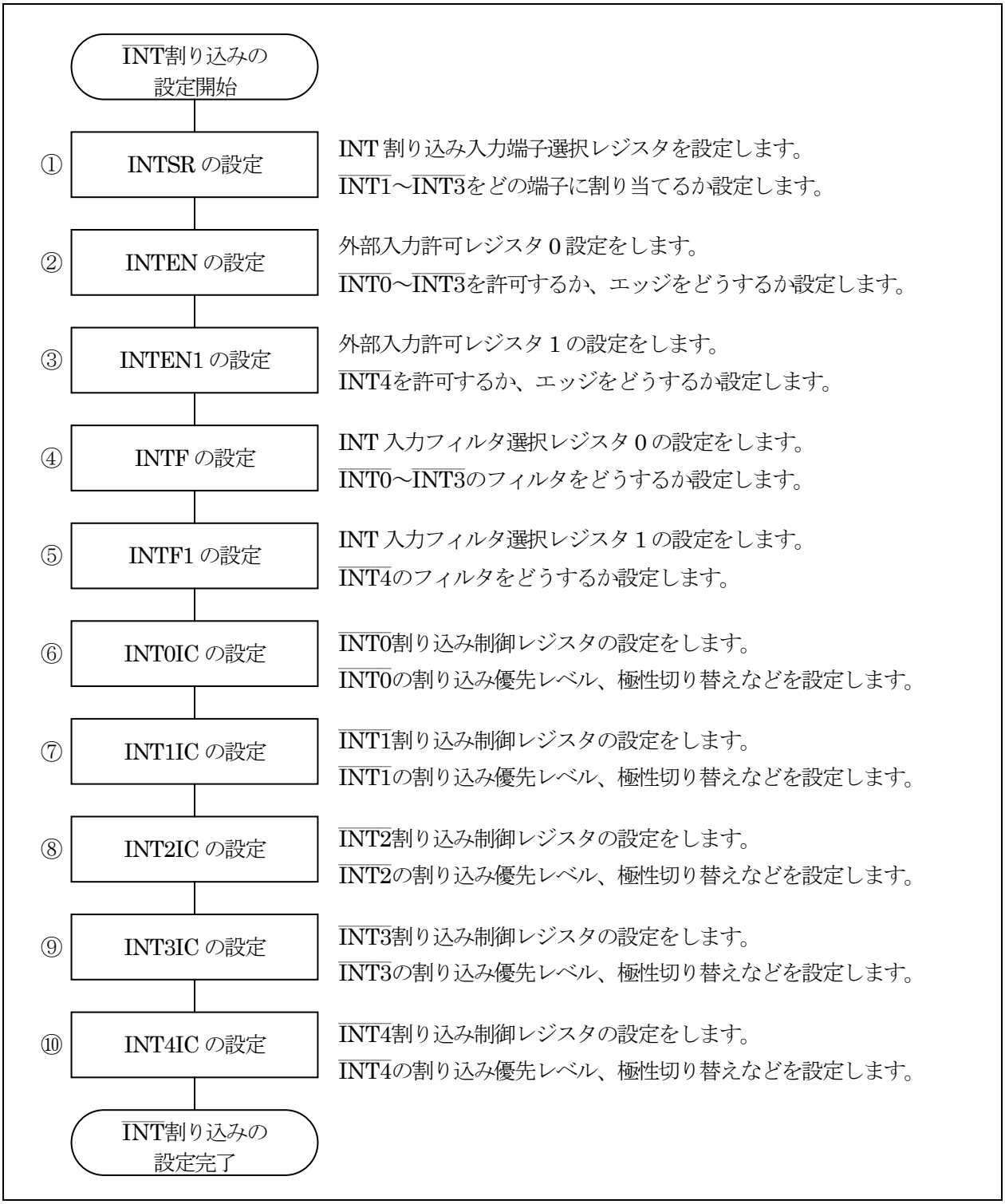
INT0～INT4割り込み端子に割り当てることのできる端子を下表に示します。

端子名	割り当てることのできる端子
INT0端子	P4_5 に設定可能です。
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) $\overline{\text{INT3}}$割り込みの設定

レジスタの設定手順を下記に示します。



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	O				O			

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。$\overline{\text{INT0}}$～$\overline{\text{INT3}}$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。

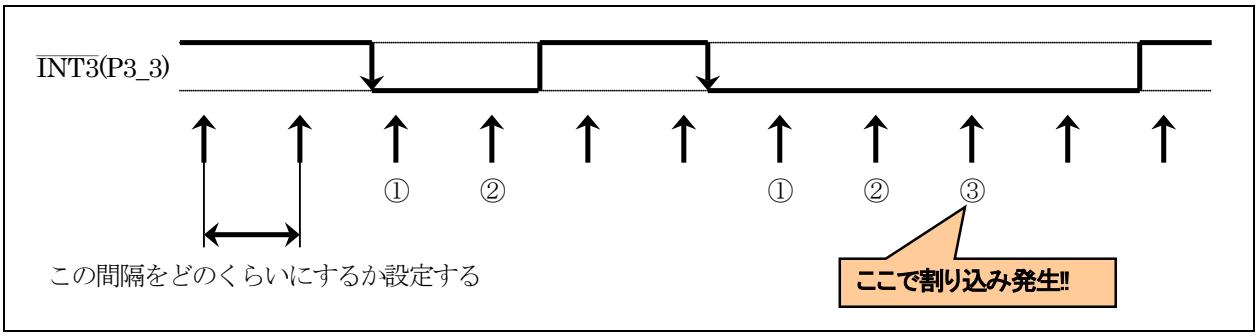
bit	7	6	5	4	3	2	1	0
設定値	0	0	0	0	0	0	0	0
16 進数	O				O			

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。$\overline{\text{INT0}}$～$\overline{\text{INT3}}$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				O			

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	0000 00
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			

⑨$\overline{\text{INT3}}$割り込み制御レジスタ(INT3IC：INT3 interrupt control register)の設定

$\overline{\text{INT3}}$割り込み制御レジスタの設定をします。$\overline{\text{INT3}}$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_int1ic	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_int1ic	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_int1ic bit1:ilvl1_int1ic bit0:ilvl0_int1ic	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$\overline{\text{INT3}}$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、R8C/38A 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、R8C/38A 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。

$\overline{\text{INT3}}$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			

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	O				O			

16.5.2 intINT3 関数（$\overline{\text{INT3}}$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。

```
125 :  /************************************************************************/
126 :  /* INT3 割り込み処理                                                    */
127 :  /************************************************************************/
128 :  #pragma interrupt intINT3(vect=26)
129 :  void intINT3( void )
130 :  {
131 :      cnt_int3++;
132 :      p2 = cnt_int3;
133 :  }
```

128 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。
129 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。
131 行	変数 cnt_int3 を+1 します。
132 行	ポート 2 へ、変数 cnt_int3 の値を出力します。

16.5.3 main 関数

```
39 :  /************************************************************************/
40 :  /* メインプログラム                                                     */
41 :  /************************************************************************/
42 :  void main( void )
43 :  {
44 :      init();                             /* 初期化                       */
45 :      asm(" fset I ");                    /* 全体の割り込み許可           */
46 :
47 :      while( 1 ) {
48 :          led_out_m( 1 );
49 :          timer( 1000 );
50 :          led_out_m( 0 );
51 :          timer( 1000 );
52 :      }
53 :  }
```

45 行	全体の割り込みを許可する命令です。 init 関数内で $\overline{\text{INT3}}$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。
47 行 ～ 52 行	RY_R8C38 ボードの LED を 1 秒点灯、1 秒消灯させます。 timer 関数は、ソフトウェアによるタイマです。詳しくは、プロジェクト「timer1」の解説を参照してください。

16.6 演習

(1) $\overline{\text{INT3}}$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) $\overline{\text{INT1}}$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、左から「下下下下　下上下下」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、実習基板 Ver.2 の LED に出力します。今回の A/D 変換は、単発モードを使います。

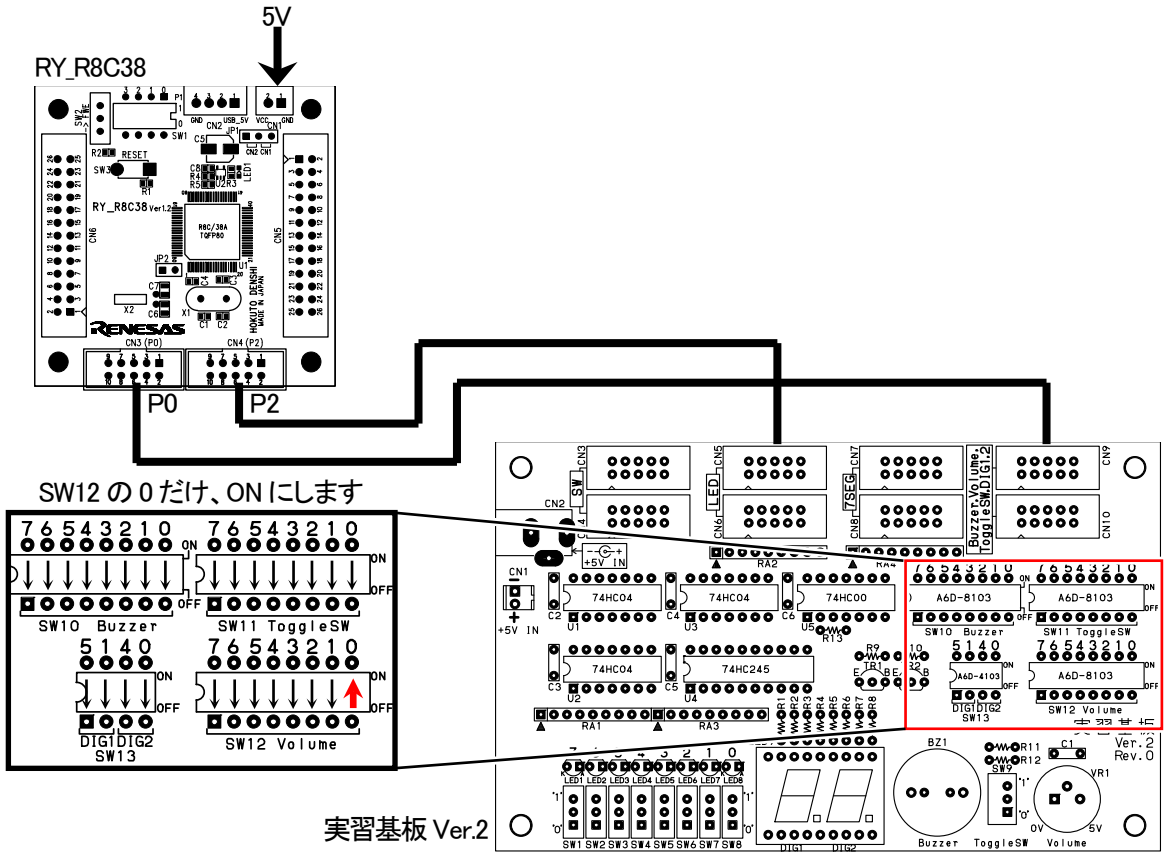
17.2　接続

■使用ポート

マイコンのポート	接続内容
P0_0 (CN3)	実習基板 Ver.2 のボリューム部を接続します。その他のボリュームやアナログセンサなど 0～5V を出力する機器でも構いません。
P2 (CN4)	実習基板 Ver.2 の LED 部など、出力機器を接続します。

■接続例

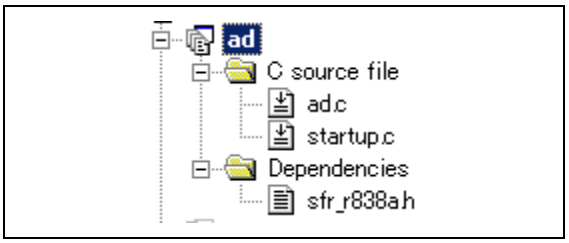
実習基板 Ver.2 を使ったときの接続例を下記に示します。



■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 変換した値を実習基板 Ver.2 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。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥ad¥startup.c
2	ad.c	実際に制御するプログラムが書かれています。R8C/38A の内蔵周辺機能(SFR)の初期化も行います。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥ad¥ad.c
3	sfr_r838a.h	R8C/38A マイコンの内蔵周辺機能を制御するためのレジスタ（Special Function Registers)を定義したファイルです。 ファイルの位置→C:¥Workspace¥common_r8c38a¥sfr_r838a.h

17.4 プログラム「ad.c」

```
 1 :  /***********************************************************************/
 2 :  /* 対象マイコン R8C/38A                                                */
 3 :  /* ﾌｧｲﾙ内容     A/D変換(単発モード)                                    */
 4 :  /* バージョン   Ver.1.00                                               */
 5 :  /* Date         2010.04.01                                             */
 6 :  /* Copyright    ジャパンマイコンカーラリー実行委員会                   */
 7 :  /***********************************************************************/
 8 :  /*
 9 :  入力：AN7(P0_0)端子 0～5V(実習基板Ver.2のボリューム部)
10 :  出力：P2_7-P2_0(実習基板Ver.2のLED部)
11 :
12 :  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
13 :  ポート2(実習基板Ver.2のLED部)へ出力します。
14 :  */
15 :
16 :  /*====================================*/
17 :  /* インクルード                       */
18 :  /*====================================*/
19 :  #include "sfr_r838a.h"                 /* R8C/38A 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 :  int get_ad7( void );
30 :
```

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

    init();                             /* 初期化                       */

    while( 1 ) {
        ad = get_ad7();
        ad = ad >> 2;
        p2 = ad;
    }
}

/************************************************************************/
/* R8C/38A スペシャルファンクションレジスタ(SFR)の初期化                */
/************************************************************************/
void init( void )
{
    int i;

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

    /* ポートの入出力設定 */
    prc2 = 1;                           /* PD0のプロテクト解除          */
    pd0 = 0x00;                         /* 0:ボリュームなどのｱﾅﾛｸﾞ電圧  */
    pd1 = 0xd0;                         /* 5:RXD0 4:TXD0 3-0:DIP SW     */
    pd2 = 0xff;                         /* 7-0:LED                      */
    pd3 = 0xff;                         /*                              */
    p4  = 0x20;                         /* P4_5のLED:初期は点灯         */
    pd4 = 0xb8;                         /* 7:XOUT 6:XIN 5:LED 2:VREF    */
    pd5 = 0xff;                         /*                              */
    pd6 = 0xff;                         /*                              */
    pd7 = 0xff;                         /*                              */
    pd8 = 0xff;                         /*                              */
    pd9 = 0x3f;                         /*                              */
    pur0 = 0x04;                        /* P1_3～P1_0のプルアップON     */
}

/************************************************************************/
/* A/D値読み込み(AN7)                                                   */
/* 引数   なし                                                          */
/* 戻り値 A/D値 0～1023                                                 */
/************************************************************************/
int get_ad7( void )
{
    int i;

    /* A/Dコンバータの設定 */
    admod   = 0x03;                     /* 単発モードに設定             */
    adinsel = 0x07;                     /* 入力端子AN7(P0_0)を選択      */
    adcon1  = 0x30;                     /* A/D動作可能                  */
    asm(" nop ");                       /* φADの1サイクルウエイト入れる*/
    adcon0  = 0x01;                     /* A/D変換スタート              */

    while( adcon0 & 0x01 );             /* A/D変換終了待ち              */

    i = ad7;

    return i;
}

/************************************************************************/
/* end of file                                                          */
/************************************************************************/
```

17.5 プログラムの解説

17.5.1 init 関数（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。今回はポート 0 の全 bit を入力端子にしています。

```
 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;                     /* 0:ボリュームなどのｱﾅﾛｸﾞ電圧  */
 65 :      pd1 = 0xd0;                     /* 5:RXD0 4:TXD0 3-0:DIP SW     */
 66 :      pd2 = 0xff;                     /* 7-0:LED                      */
 67 :      pd3 = 0xff;                     /*                              */
 68 :      p4  = 0x20;                     /* P4_5のLED:初期は点灯         */
 69 :      pd4 = 0xb8;                     /* 7:XOUT 6:XIN 5:LED 2:VREF    */
 70 :      pd5 = 0xff;                     /*                              */
 71 :      pd6 = 0xff;                     /*                              */
 72 :      pd7 = 0xff;                     /*                              */
 73 :      pd8 = 0xff;                     /*                              */
 74 :      pd9 = 0x3f;                     /*                              */
 75 :      pur0 = 0x04;                    /* P1_3～P1_0のプルアップON     */
```

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_ad**7**」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。

```
 78 :  /************************************************************************/
 79 :  /* A/D値読み込み(AN7)                                                   */
 80 :  /* 引数    なし                                                         */
 81 :  /* 戻り値 A/D値 0～1023                                                 */
 82 :  /************************************************************************/
 83 :  int get_ad7( void )
 84 :  {
 85 :      int i;
 86 :
 87 :      /* A/Dコンバータの設定 */
 88 :      admod   = 0x03;                 /* 単発モードに設定             */
 89 :      adinsel = 0x07;                 /* 入力端子AN7(P0_0)を選択      */
 90 :      adcon1  = 0x30;                 /* A/D動作可能                  */
 91 :      asm(" nop ");                   /* φADの1サイクルウエイト入れる*/
 92 :      adcon0  = 0x01;                 /* A/D変換スタート              */
 93 :
 94 :      while( adcon0 & 0x01 );         /* A/D変換終了待ち              */
 95 :
 96 :      i = ad7;
 97 :
 98 :      return i;
 99 :  }
```

(1) A/D コンバータとは

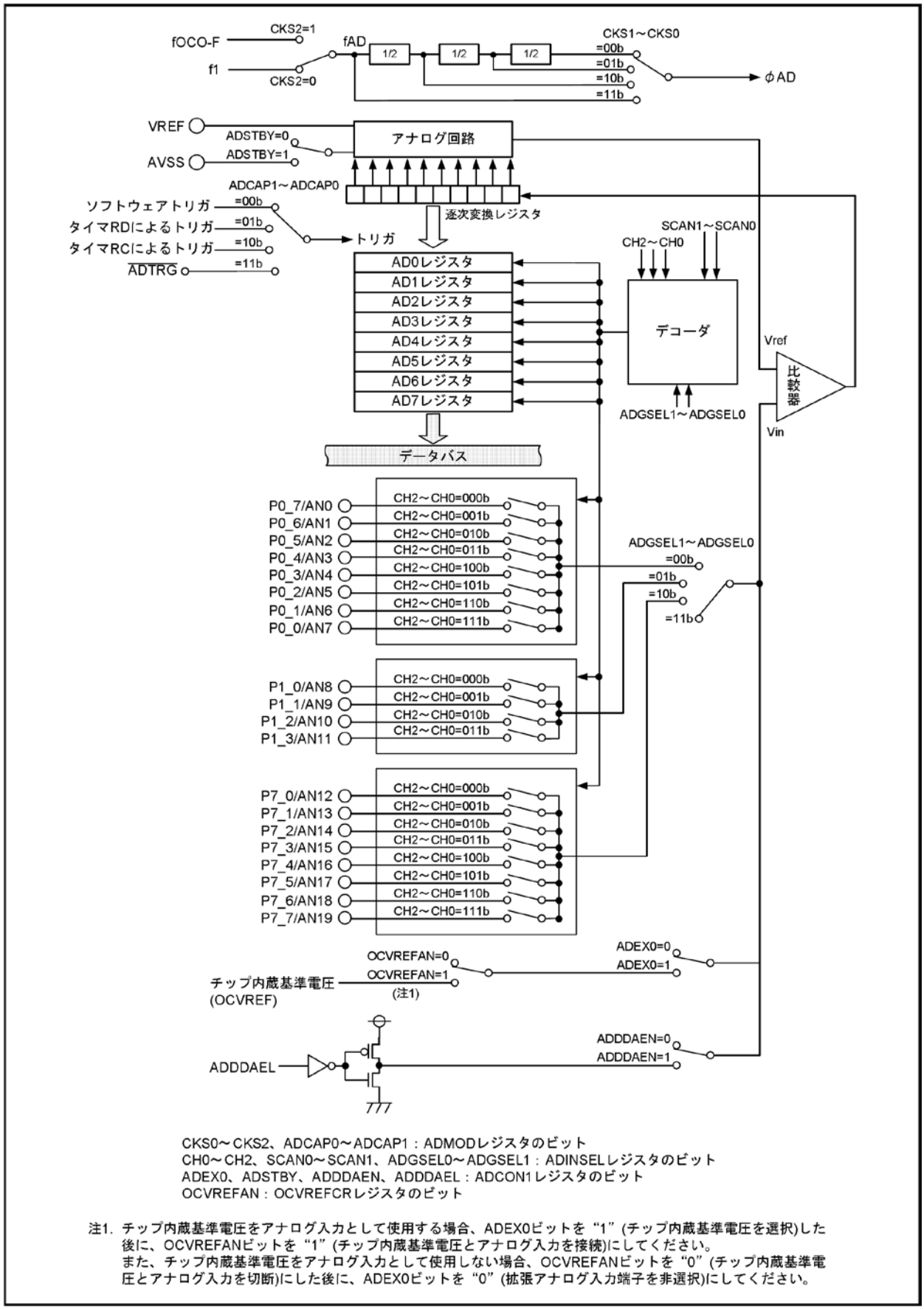
A/D コンバータは、アナログ／デジタルコンバータの略称です。マイコンは"0"か"1"か("なし"か"あり"か)のデジタルで判断するので、外部からの入力電圧も 0V("0")か 5V("1")の 2 通りしか判断することができません。これ以外の電圧が入力されても、ある電圧を境にして"0"または"1"と判断します。

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/38A 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	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)、 AN12(P7_0)、AN13(P7_1)、AN14(P7_2)、AN15(P7_3)、 AN16(P7_4)、AN17(P7_5)、AN18(P7_6)、AN19(P7_7) の 20 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 A/D 変換を行います。どの端子を 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になります。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、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。RY_R8C38 ボードは、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。 $43\times(1/外部クロック)=43\times(1/20\times10^6)=43\times0.05\times10^{-6}=2.15\times10^{-6}=$**2.15 μs** 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 μ s 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～AN19 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～AN19 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～AN19 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～AN19 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～AN19 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。

(2) A/D コンバータのブロック図

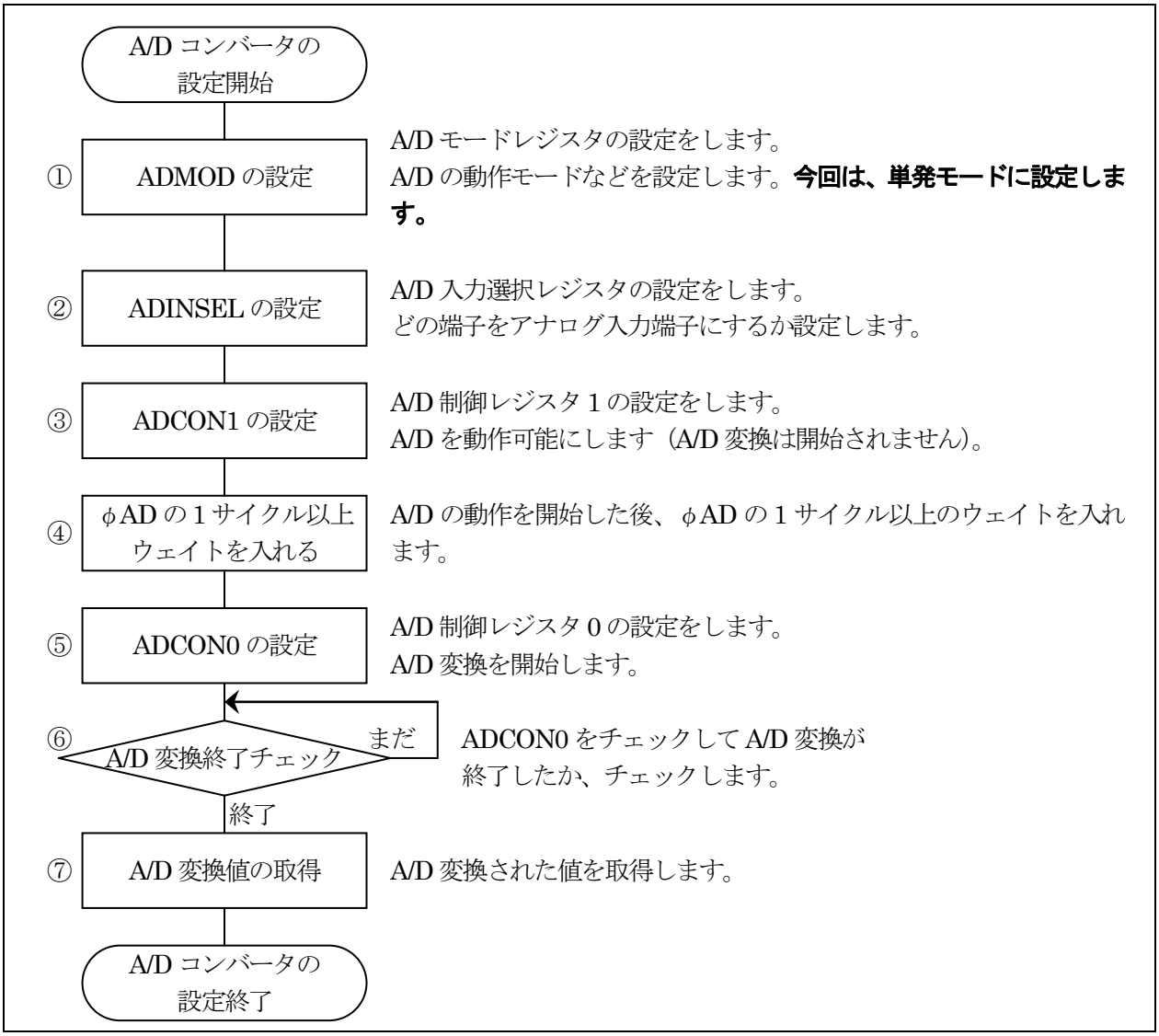


CKS0～CKS2、ADCAP0～ADCAP1：ADMODレジスタのビット
CH0～CH2、SCAN0～SCAN1、ADGSEL0～ADGSEL1：ADINSELレジスタのビット
ADEX0、ADSTBY、ADDDAEN、ADDDAEL：ADCON1レジスタのビット
OCVREFAN：OCVREFCRレジスタのビット

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)した後に、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。

(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。実習基板 Ver.2 のボリュームが接続されている 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)の設定

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)
1	0	0	0	0	0	0	0	AN12(P7_0)
1	0	0	0	0	0	0	1	AN13(P7_1)
1	0	0	0	0	0	1	0	AN14(P7_2)
1	0	0	0	0	0	1	1	AN15(P7_3)
1	0	0	0	0	1	0	0	AN16(P7_4)
1	0	0	0	0	1	0	1	AN17(P7_5)
1	0	0	0	0	1	1	0	AN18(P7_6)
1	0	0	0	0	1	1	1	AN19(P7_7)

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）の設定

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、φA/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、φAD 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			

④φAD 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、φA/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）の設定

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	0000000
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。

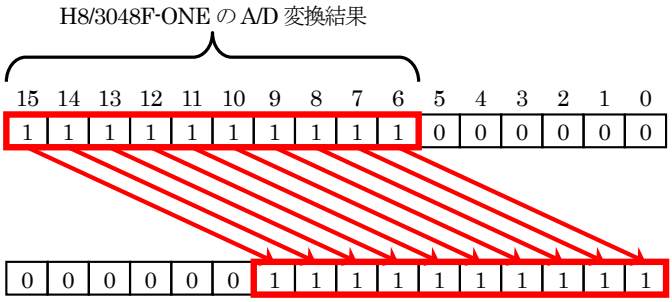
アナログ入力端子	読み込むレジスタ
AN0(P0_7)	AD0
AN1(P0_6)	AD1
AN2(P0_5)	AD2
AN3(P0_4)	AD3
AN4(P0_3)	AD4
AN5(P0_2)	AD5
AN6(P0_1)	AD6
AN7(P0_0)	AD7
AN8(P1_0)	AD0
AN9(P1_1)	AD1
AN10(P1_2)	AD2
AN11(P1_3)	AD3
AN12(P7_0)	AD0
AN13(P7_1)	AD1
AN14(P7_2)	AD2
AN15(P7_3)	AD3
AN16(P7_4)	AD4
AN17(P7_5)	AD5
AN18(P7_6)	AD6
AN19(P7_7)	AD7

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。

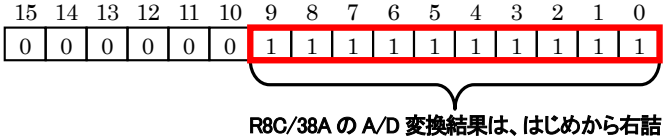
```
i = ad7;
```

※H8/3048F-ONE と R8C/38A 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/38A 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。



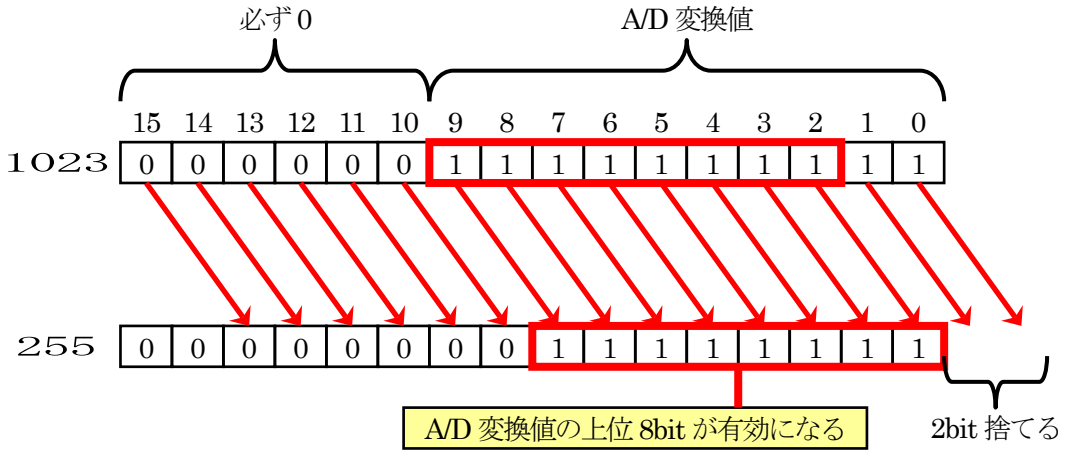
17.5.3　main 関数

A/D 変換値を取得、その値をポート 2 の LED へ出力します。

```
34 :  void main( void )
35 :  {
36 :      int ad;
37 :
38 :      init();                              /* 初期化                          */
39 :
40 :      while( 1 ) {
41 :          ad = get_ad7();
42 :          ad = ad >> 2;
43 :          p2 = ad;
44 :      }
45 :  }
```

41 行	get_ad7 関数で A/D 変換値を取得し、ad 変数に格納します。
42 行	A/D 変換値は、0～1023(2 進数で"11 1111 1111"→10bit 幅)の値です。ポート 2 は 8bit です。そのため今回は、2 進数で 10 桁の A/D 値を 8 桁に変換します。プログラムは、右シフトを 2 ビット分行い、下位の 2 桁を捨てます。その結果、A/D 値は 0～255 の値になり、ad 変数に代入します。
43 行	0～255 に変換した A/D 値をポート 2 へ出力します。

変数 ad の値が 1023 のとき、42 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)の状態で、アナログ入力端子を P0_1 端子に変更して、A/D 変換した値を 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）

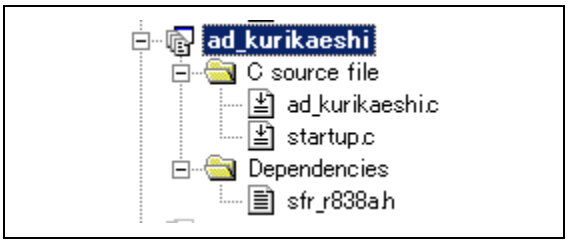
18.1　概要

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、実習基板 Ver.2 の LED 部に出力します。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。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥ad_kurikaeshi¥startup.c
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/38A の内蔵周辺機能(SFR)の初期化も行います。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥ad_kurikaeshi¥ad_kurikaeshi.c
3	sfr_r838a.h	R8C/38A マイコンの内蔵周辺機能を制御するためのレジスタ（Special Function Registers)を定義したファイルです。 ファイルの位置→C:¥Workspace¥common_r8c38a¥sfr_r838a.h

18.4 プログラム「ad_kurikaeshi.c」

```
 1 :  /***********************************************************************/
 2 :  /* 対象マイコン R8C/38A                                                */
 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.1.00                                                */
 5 :  /* Date         2010.04.01                                              */
 6 :  /* Copyright    ジャパンマイコンカーラリー実行委員会                    */
 7 :  /***********************************************************************/
 8 :  /*
 9 :  入力：AN7(P0_0)端子 0～5V(実習基板Ver.2のボリューム部)
10 :  出力：P2_7-P2_0(実習基板Ver.2のLED部)
11 :
12 :  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
13 :  ポート2(実習基板Ver.2のLED部)へ出力します。
14 :  */
15 :
16 :  /*====================================*/
17 :  /* インクルード                       */
18 :  /*====================================*/
19 :  #include "sfr_r838a.h"                 /* R8C/38A 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 :  int get_ad7( void );
30 :
31 :  /***********************************************************************/
32 :  /* メインプログラム                                                    */
33 :  /***********************************************************************/
34 :  void main( void )
35 :  {
36 :      int ad;
37 :
38 :      init();                           /* 初期化                        */
39 :
40 :      while( 1 ) {
41 :          ad = get_ad7();
42 :          ad = ad >> 2;
43 :          p2 = ad;
44 :      }
45 :  }
46 :
47 :  /***********************************************************************/
48 :  /* R8C/38A 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;                       /* 0:ボリュームなどのｱﾅﾛｸﾞ電圧  */
65 :      pd1 = 0xd0;                       /* 5:RXD0 4:TXD0 3-0:DIP SW      */
66 :      pd2 = 0xff;                       /* 7-0:LED                       */
67 :      pd3 = 0xff;                       /*                               */
68 :      p4  = 0x20;                       /* P4_5のLED:初期は点灯          */
69 :      pd4 = 0xb8;                       /* 7:XOUT 6:XIN 5:LED 2:VREF     */
70 :      pd5 = 0xff;                       /*                               */
71 :      pd6 = 0xff;                       /*                               */
72 :      pd7 = 0xff;                       /*                               */
73 :      pd8 = 0xff;                       /*                               */
74 :      pd9 = 0x3f;                       /*                               */
75 :      pur0 = 0x04;                      /* P1_3～P1_0のプルアップON      */
76 :
77 :      /* A/Dコンバータの設定 */
78 :      admod   = 0x13;                   /* 繰り返しモード0に設定         */
79 :      adinsel = 0x07;                   /* 入力端子AN7(P0_0)を選択       */
80 :      adcon1  = 0x30;                   /* A/D動作可能                   */
81 :      asm(" nop ");                     /* φADの1サイクルウエイト入れる*/
82 :      adcon0  = 0x01;                   /* A/D変換スタート               */
83 :  }
84 :
```

```
 85 :  /************************************************************************/
 86 :  /* A/D値読み込み(AN7)                                                   */
 87 :  /* 引数   なし                                                          */
 88 :  /* 戻り値 A/D値 0～1023                                                 */
 89 :  /************************************************************************/
 90 :  int get_ad7( void )
 91 :  {
 92 :      int i;
 93 :
 94 :      /* 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 */
 95 :      i = ad7;
 96 :
 97 :      return i;
 98 :  }
 99 :
100 :  /************************************************************************/
101 :  /* end of file                                                          */
102 :  /************************************************************************/
```

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。今回はポート 0 の全ての bit を入力端子にします。

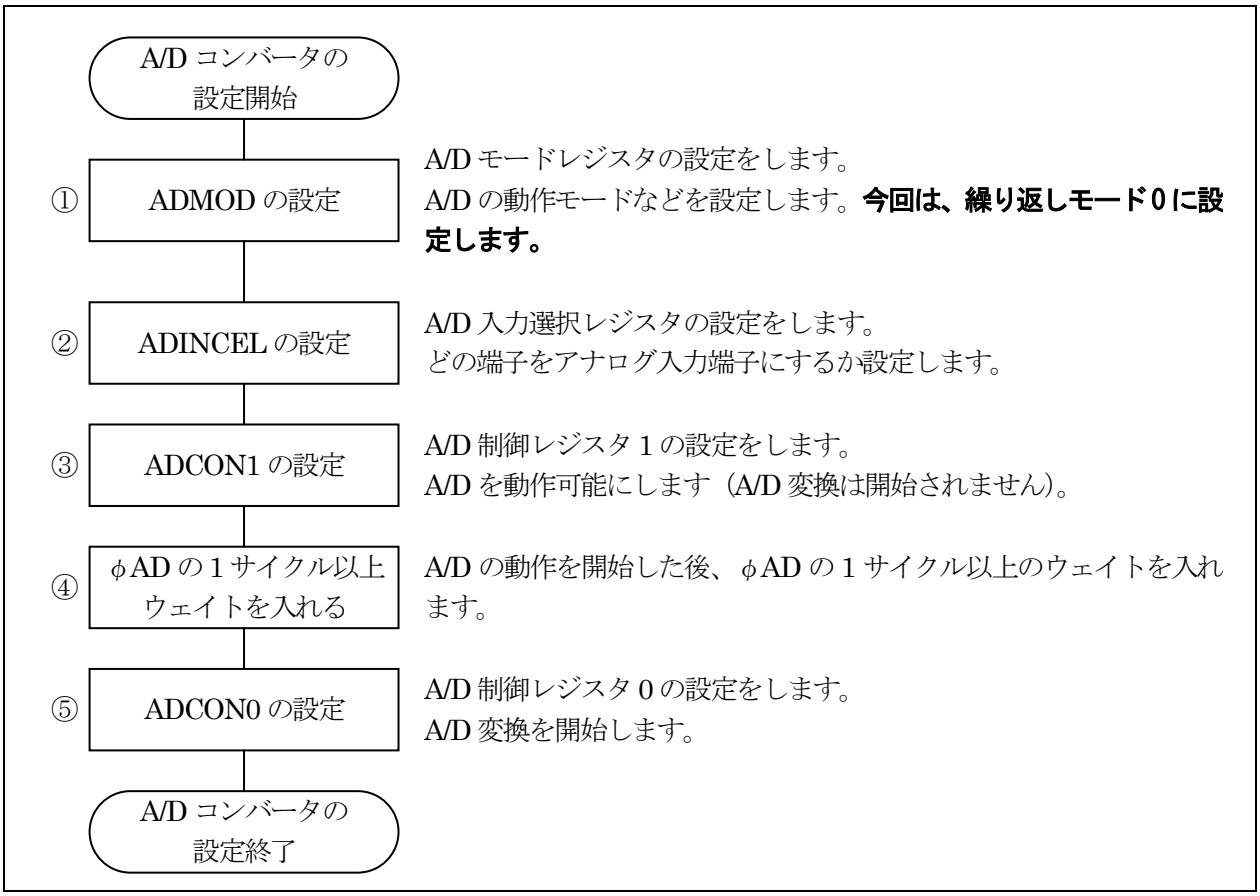
```
 62 :      /* ポートの入出力設定 */
 63 :      prc2 = 1;                   /* PD0のプロテクト解除          */
 64 :      pd0 = 0x00;                 /* 0:ボリュームなどのｱﾅﾛｸﾞ電圧  */
 65 :      pd1 = 0xd0;                 /* 5:RXD0 4:TXD0 3-0:DIP SW     */
 66 :      pd2 = 0xff;                 /* 7-0:LED                      */
 67 :      pd3 = 0xff;                 /*                              */
 68 :      p4  = 0x20;                 /* P4_5のLED:初期は点灯         */
 69 :      pd4 = 0xb8;                 /* 7:XOUT 6:XIN 5:LED 2:VREF    */
 70 :      pd5 = 0xff;                 /*                              */
 71 :      pd6 = 0xff;                 /*                              */
 72 :      pd7 = 0xff;                 /*                              */
 73 :      pd8 = 0xff;                 /*                              */
 74 :      pd9 = 0x3f;                 /*                              */
 75 :      pur0 = 0x04;                /* P1_3～P1_0のプルアップON     */
```

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。

```
77 :      /* A/Dコンバータの設定 */
78 :      admod    = 0x13;                  /* 繰り返しモード0に設定          */
79 :      adinsel = 0x07;                   /* 入力端子AN7(P0_0)を選択        */
80 :      adcon1   = 0x30;                  /* A/D動作可能                    */
81 :      asm(" nop ");                     /* φADの1サイクルウェイト入れる*/
82 :      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。実習基板 Ver.2 のボリュームが接続されている 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)の設定

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)
1	0	0	0	0	0	0	0	AN12(P7_0)
1	0	0	0	0	0	0	1	AN13(P7_1)
1	0	0	0	0	0	1	0	AN14(P7_2)
1	0	0	0	0	0	1	1	AN15(P7_3)
1	0	0	0	0	1	0	0	AN16(P7_4)
1	0	0	0	0	1	0	1	AN17(P7_5)
1	0	0	0	0	1	1	0	AN18(P7_6)
1	0	0	0	0	1	1	1	AN19(P7_7)

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）の設定

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、φA/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、φAD 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			

④φAD 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、φA/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)の設定

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			

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。

```
 85 :  /************************************************************************/
 86 :  /* A/D値読み込み(AN7)                                                   */
 87 :  /* 引数    なし                                                         */
 88 :  /* 戻り値 A/D値 0～1023                                                 */
 89 :  /************************************************************************/
 90 :  int get_ad7( void )
 91 :  {
 92 :      int i;
 93 :
 94 :      /* 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 */
 95 :      i = ad7;
 96 :
 97 :      return i;
 98 :  }
```

95 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
AN12(P7_0)	AD0
AN13(P7_1)	AD1
AN14(P7_2)	AD2
AN15(P7_3)	AD3
AN16(P7_4)	AD4
AN17(P7_5)	AD5
AN18(P7_6)	AD6
AN19(P7_7)	AD7

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。

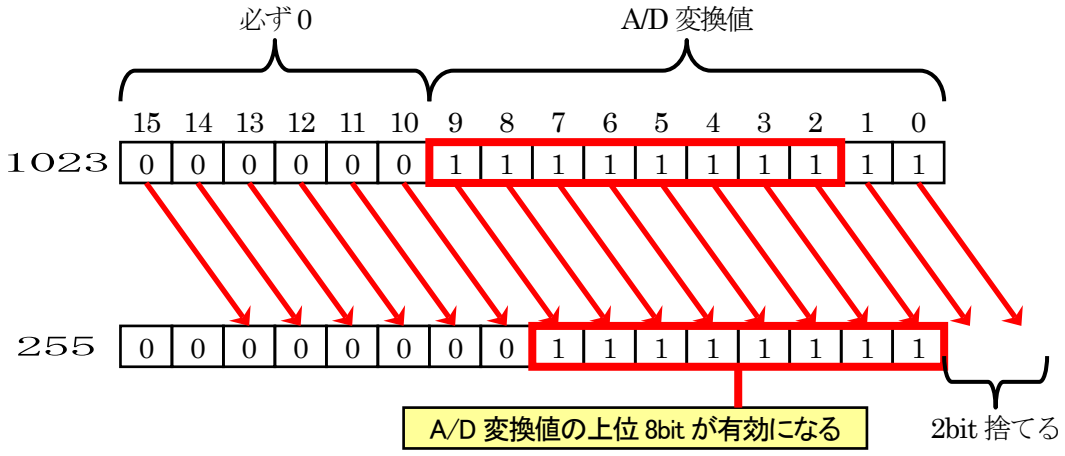
18.5.4　main 関数

A/D 変換値を取得、ポート 2 へ取得した A/D 値を出力します。

```
34 :  void main( void )
35 :  {
36 :      int ad;
37 :
38 :      init();                              /* 初期化                          */
39 :
40 :      while( 1 ) {
41 :          ad = get_ad7();
42 :          ad = ad >> 2;
43 :          p2 = ad;
44 :      }
45 :  }
```

41 行	get_ad7 関数で A/D 変換値を取得し、ad 変数に格納します。
42 行	A/D 変換値は、0～1023(2 進数で"11 1111 1111"→10bit 幅)の値です。ポート 2 は 8bit です。そのため今回は、2 進数で 10 桁の A/D 値を 8 桁に変換します。プログラムは、右シフトを 2 ビット分行い、下位の 2 桁を捨てます。その結果、A/D 値は 0～255 の値になり、ad 変数に代入します。
43 行	0～255 に変換した A/D 値をポート 2 へ出力します。

変数 ad の値が 1023 のとき、42 行のビットシフトの様子を下記に示します。



18.6　演習

(1) ポート 6 に LED 基板(実習基板 Ver.2 の LED 部など)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)の状態で、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。

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、実習基板 Ver.2 の LED 部に出力します。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。

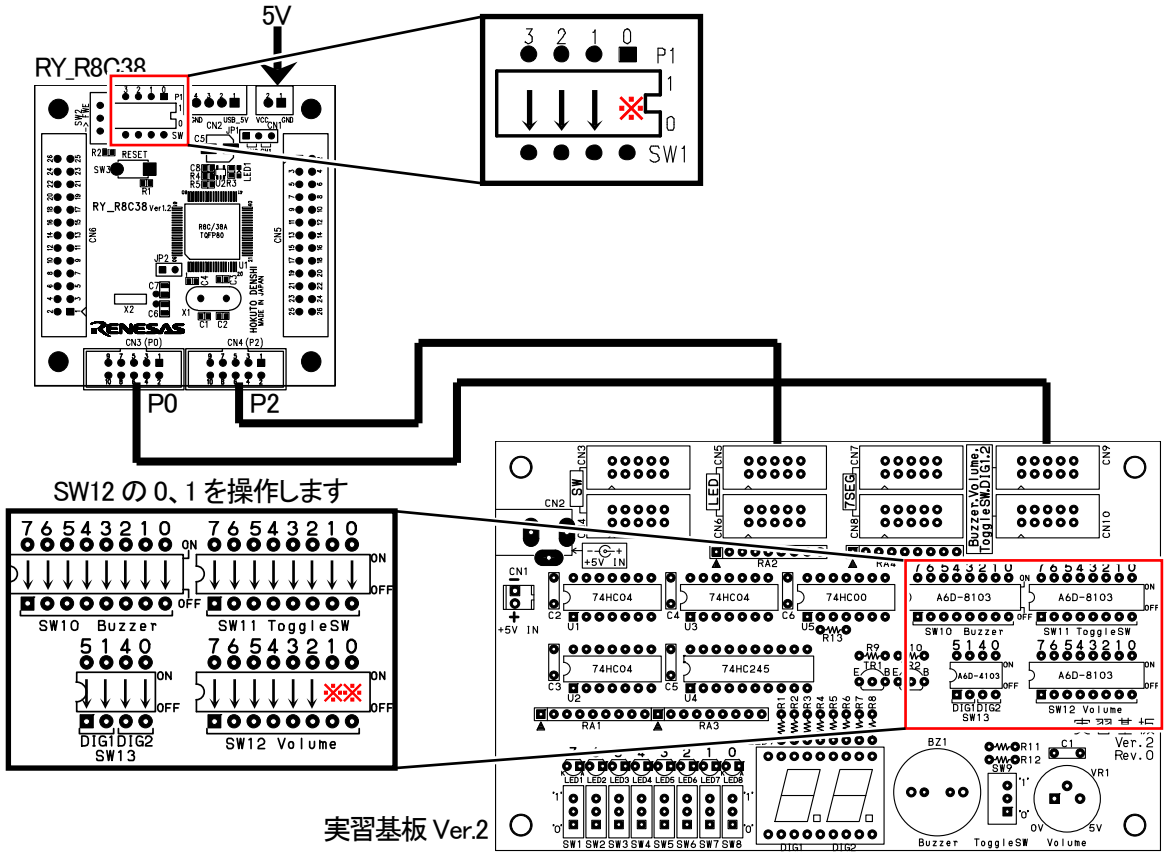
19.2 接続

■使用ポート

マイコンのポート	接続内容
P0_0、P0_1（CN3）	実習基板 Ver.2 のボリューム部を接続します。その他のボリュームやアナログセンサなど 0～5V を出力する機器でも構いません。
P2（CN4）	実習基板 Ver.2 の LED 部など、出力機器を接続します。
P1_3、P1_2、P1_1、P1_0	RY_R8C38 ボードのディップスイッチです。

■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。



※RY_R8C38 ボードのディップスイッチと、実習基板 Ver.2 の SW12 は、A/D 変換をどの端子でおこなうかで決めます。スイッチの状態を下表に示します。″0″が OFF、″1″が ON です。

A/D 変換を行う端子	RY_R8C38 ボードの ディップスイッチ	実習基板 Ver.2 の SW12
AN7(P0_0)	0000（↓↓↓↓）	0000 0001（↓↓↓ ↓↓↓↑）
AN6(P0_1)	0001（↓↓↓↑）	0000 0010（↓↓↓ ↓↓↑↓）

■操作方法

AN7(P0_0)端子で A/D 変換を行うときは、上表の「A/D 変換を行う端子＝AN7(P0_0)」の状態にスイッチを設定します。実習基板 Ver.2 のボリュームを回して、実習基板 Ver.2 の LED の付き方を確認します。

AN6(P0_1)端子で A/D 変換を行うときは、上表の「A/D 変換を行う端子＝AN6(P0_1)」の状態にスイッチを設定します。実習基板 Ver.2 のボリュームを回して、実習基板 Ver.2 の LED の付き方を確認します。

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。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥ad_kurikaeshi_souin¥startup.c
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/38A の内蔵周辺機能(SFR)の初期化も行います。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥ad_kurikaeshi_souin¥ad_kurikaeshi_souin.c
3	sfr_r838a.h	R8C/38A マイコンの内蔵周辺機能を制御するためのレジスタ（Special Function Registers)を定義したファイルです。 ファイルの位置→C:¥Workspace¥common_r8c38a¥sfr_r838a.h

19.4 プログラム「ad_kurikaeshi_souin.c」

```
 1 :  /***********************************************************************/
 2 :  /* 対象マイコン  R8C/38A                                               */
 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.1.00                                              */
 5 :  /* Date          2010.04.01                                            */
 6 :  /* Copyright     ジャパンマイコンカーラリー実行委員会                  */
 7 :  /***********************************************************************/
 8 :  /*
 9 :  入力：AN0(P0_7)～AN7(P0_0)端子 0～5V(実習基板Ver.2のボリューム部)
10 :        P1_3-P1_0(マイコンボードのディップスイッチ)
11 :  出力：P2_7-P2_0(実習基板Ver.2のLED部)
12 :
13 :  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、
14 :  デジタル値をポート2(実習基板Ver.2のLED部)へ出力します。
15 :  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。
16 :  */
17 :
18 :  /*====================================*/
19 :  /* インクルード                       */
20 :  /*====================================*/
21 :  #include "sfr_r838a.h"                 /* R8C/38A 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 :  unsigned char dipsw_get( void );
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 :          switch( dipsw_get() ) {
 44 :          case 0:
 45 :              ad = ad7;                   /* AD7(P0_0)取得                */
 46 :              ad = ad >> 2;
 47 :              p2 = ad;
 48 :              break;
 49 :          case 1:
 50 :              ad = ad6;                   /* AD6(P0_1)取得                */
 51 :              ad = ad >> 2;
 52 :              p2 = ad;
 53 :              break;
 54 :          }
 55 :      }
 56 :  }
 57 :
 58 :  /************************************************************************/
 59 :  /* R8C/38A スペシャルファンクションレジスタ(SFR)の初期化                */
 60 :  /************************************************************************/
 61 :  void init( void )
 62 :  {
 63 :      int i;
 64 :
 65 :      /* クロックをXINクロック(20MHz)に変更 */
 66 :      prc0  = 1;                          /* プロテクト解除               */
 67 :      cm13  = 1;                          /* P4_6,P4_7をXIN-XOUT端子にする*/
 68 :      cm05  = 0;                          /* XINクロック発振              */
 69 :      for(i=0; i<50; i++ );               /* 安定するまで少し待つ(約10ms) */
 70 :      ocd2  = 0;                          /* システムクロックをXINにする  */
 71 :      prc0  = 0;                          /* プロテクトON                 */
 72 :
 73 :      /* ポートの入出力設定 */
 74 :      prc2 = 1;                           /* PD0のプロテクト解除          */
 75 :      pd0 = 0x00;                         /* 7-0:ボリュームなどのｱﾅﾛｸﾞ電圧*/
 76 :      pd1 = 0xd0;                         /* 5:RXD0 4:TXD0 3-0:DIP SW     */
 77 :      pd2 = 0xff;                         /* 7-0:LED                      */
 78 :      pd3 = 0xff;                         /*                              */
 79 :      p4  = 0x20;                         /* P4_5のLED:初期は点灯         */
 80 :      pd4 = 0xb8;                         /* 7:XOUT 6:XIN 5:LED 2:VREF    */
 81 :      pd5 = 0xff;                         /*                              */
 82 :      pd6 = 0xff;                         /*                              */
 83 :      pd7 = 0xff;                         /*                              */
 84 :      pd8 = 0xff;                         /*                              */
 85 :      pd9 = 0x3f;                         /*                              */
 86 :      pur0 = 0x04;                        /* P1_3～P1_0のプルアップON     */
 87 :
 88 :      /* A/Dコンバータの設定 */
 89 :      admod   = 0x33;                     /* 繰り返し掃引モードに設定     */
 90 :      adinsel = 0x30;                     /* 入力端子P0の8端子を選択      */
 91 :      adcon1  = 0x30;                     /* A/D動作可能                  */
 92 :      asm(" nop ");                       /* φADの1サイクルウエイト入れる*/
 93 :      adcon0  = 0x01;                     /* A/D変換スタート              */
 94 :  }
 95 :
 96 :  /************************************************************************/
 97 :  /* ディップスイッチ値読み込み                                           */
 98 :  /* 戻り値 スイッチ値 0～15                                              */
 99 :  /************************************************************************/
100 :  unsigned char dipsw_get( void )
101 :  {
102 :      unsigned char sw;
103 :
104 :      sw = p1 & 0x0f;                     /* P1_3～P1_0読み込み           */
105 :
106 :      return  sw;
107 :  }
108 :
109 :  /************************************************************************/
110 :  /* end of file                                                          */
111 :  /************************************************************************/
```

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)

AN7(P0_0)～AN0(P0_7)をアナログ電圧入力端子にします。ポートの入出力設定は入力にします。出力にした端子は、A/D 変換できません。忘れやすいので、気をつけてください。

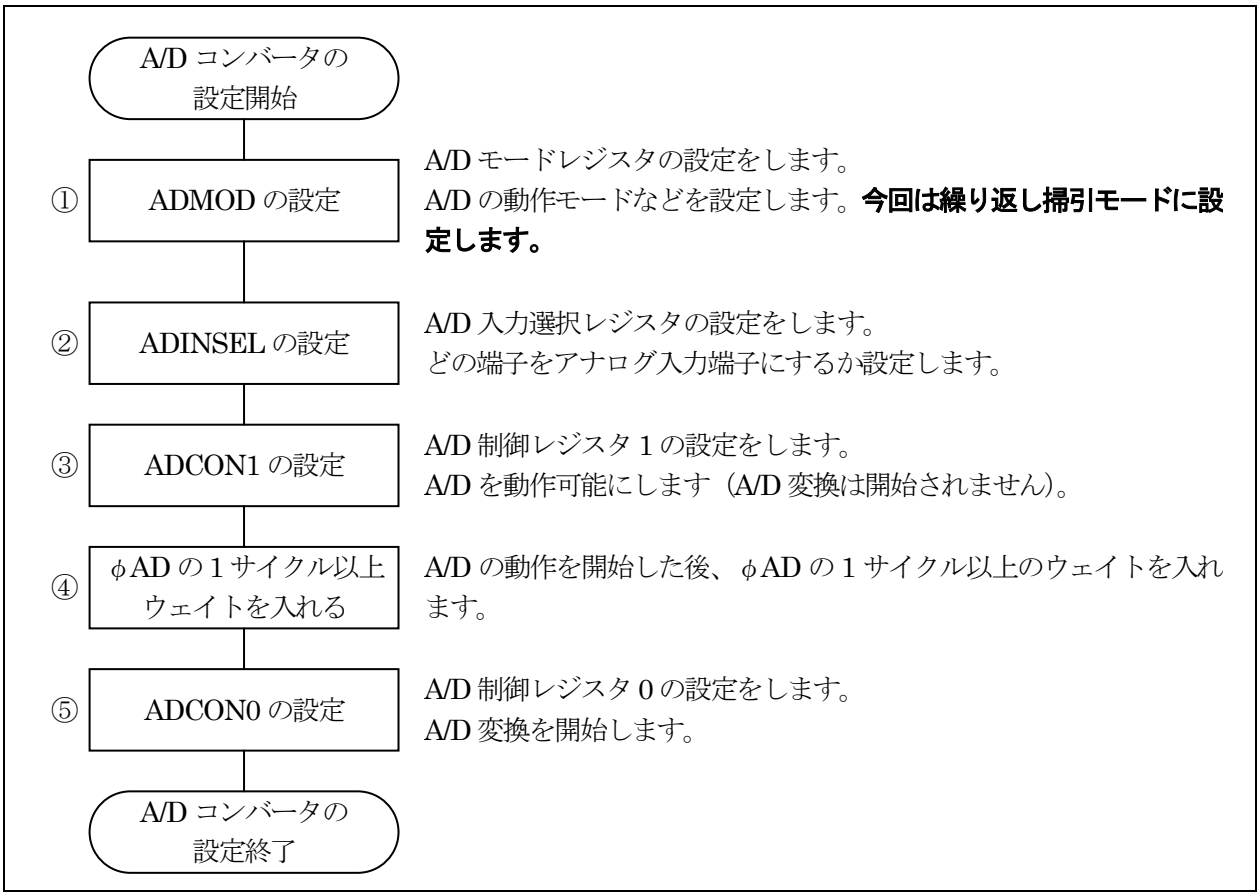
```
 73 :      /* ポートの入出力設定 */
 74 :      prc2 = 1;                       /* PD0のプロテクト解除          */
 75 :      pd0 = 0x00;                     /* 7-0:ボリュームなどのｱﾅﾛｸﾞ電圧*/
 76 :      pd1 = 0xd0;                     /* 5:RXD0 4:TXD0 3-0:DIP SW     */
 77 :      pd2 = 0xff;                     /* 7-0:LED                      */
 78 :      pd3 = 0xff;                     /*                              */
 79 :      p4  = 0x20;                     /* P4_5のLED:初期は点灯         */
 80 :      pd4 = 0xb8;                     /* 7:XOUT 6:XIN 5:LED 2:VREF    */
 81 :      pd5 = 0xff;                     /*                              */
 82 :      pd6 = 0xff;                     /*                              */
 83 :      pd7 = 0xff;                     /*                              */
 84 :      pd8 = 0xff;                     /*                              */
 85 :      pd9 = 0x3f;                     /*                              */
 86 :      pur0 = 0x04;                    /* P1_3～P1_0のプルアップON     */
```

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。

```
88 :      /* A/Dコンバータの設定 */
89 :      admod   = 0x33;                  /* 繰り返し掃引モードに設定      */
90 :      adinsel = 0x30;                  /* 入力端子P0の8端子を選択       */
91 :      adcon1  = 0x30;                  /* A/D動作可能                   */
92 :      asm(" nop ");                    /* φADの1サイクルウェイト入れる*/
93 :      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。
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
1	0	0	0	0	0	0	0	AN12(P7_0)～AN13(P7_1)の 2 端子
1	0	0	1	0	0	0	0	AN12(P7_0)～AN15(P7_3)の 4 端子
1	0	1	0	0	0	0	0	AN12(P7_0)～AN17(P7_5)の 6 端子
1	0	1	1	0	0	0	0	AN12(P7_0)～AN19(P7_7)の 8 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				O			

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) adddael	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) adddaen	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、φA/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、φAD 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			

④φAD 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、φA/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、実習基板 Ver.2 の LED 部へ A/D 値を出力します。

```
 36 :  void main( void )
 37 :  {
 38 :      int ad;
 39 :
 40 :      init();                             /* 初期化                       */
 41 :
 42 :      while( 1 ) {
 43 :          switch( dipsw_get() ) {
 44 :          case 0:
 45 :              ad = ad7;                   /* AD7(P0_0)取得                */
 46 :              ad = ad >> 2;
 47 :              p2 = ad;
 48 :              break;
 49 :          case 1:
 50 :              ad = ad6;                   /* AD6(P0_1)取得                */
 51 :              ad = ad >> 2;
 52 :              p2 = ad;
 53 :              break;
 54 :          }
 55 :      }
 56 :  }
```

43 行	dipsw_get 関数で、RY_R8C38 ボードのディップスイッチの値を取得します。
45 行	ディップスイッチの値が 0 なら、45 行にジャンプします。 AD7 端子(P0_0)の A/D 変換値を取得し、ad 変数に格納します。
46 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 2 ビット分行い、下位の 2 を捨てます。その結果、A/D 値は 0～255 の値になります。
47 行	0～255 に変換した A/D 値を実習基板 Ver.2 の LED 部に出力します。
50 行	ディップスイッチの値が 1 なら、50 行にジャンプします。 AD6 端子(P0_1)の A/D 変換値を取得し、ad 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 2 ビット分行い、下位の 2 桁を捨てます。その結果、A/D 値は 0～255 の値になります。
52 行	0～255 に変換した A/D 値を実習基板 Ver.2 の LED 部に出力します。

※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
AN12(P7_0)	AD0
AN13(P7_1)	AD1
AN14(P7_2)	AD2
AN15(P7_3)	AD3
AN16(P7_4)	AD4
AN17(P7_5)	AD5
AN18(P7_6)	AD6
AN19(P7_7)	AD7

19.6 演習

(1) RY_R8C38 ボードのディップスイッチの値 2～7 によって、AN0～5 端子から電圧を取り込むようにしなさい。このとき、取り込んだ A/D 変換値を実習基板 Ver.2 の LED へ出力するようにしなさい。
※プログラム実行時、A/D 変換する端子に合わせて、実習基板 Ver.2 のボリュームの端子切り替えスイッチ SW12 の設定も替えてください。例えば、AN5 なら SW12 の 2 を ON に、AN0 なら SW12 の 7 を ON にしてください。

(2) (1)の状態から A/D 変換の結果を 7 セグメント LED へ出力するようにしなさい。7 セグメント LED は、16 進数で表示するようにしなさい。

20.　パルスカウント（プロジェクト：timer_rg_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 RG で数える方法を説明します。タイマ RG 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）。マイコンカーでは、ロータリエンコーダのパルスカウントでの使用に最適です。

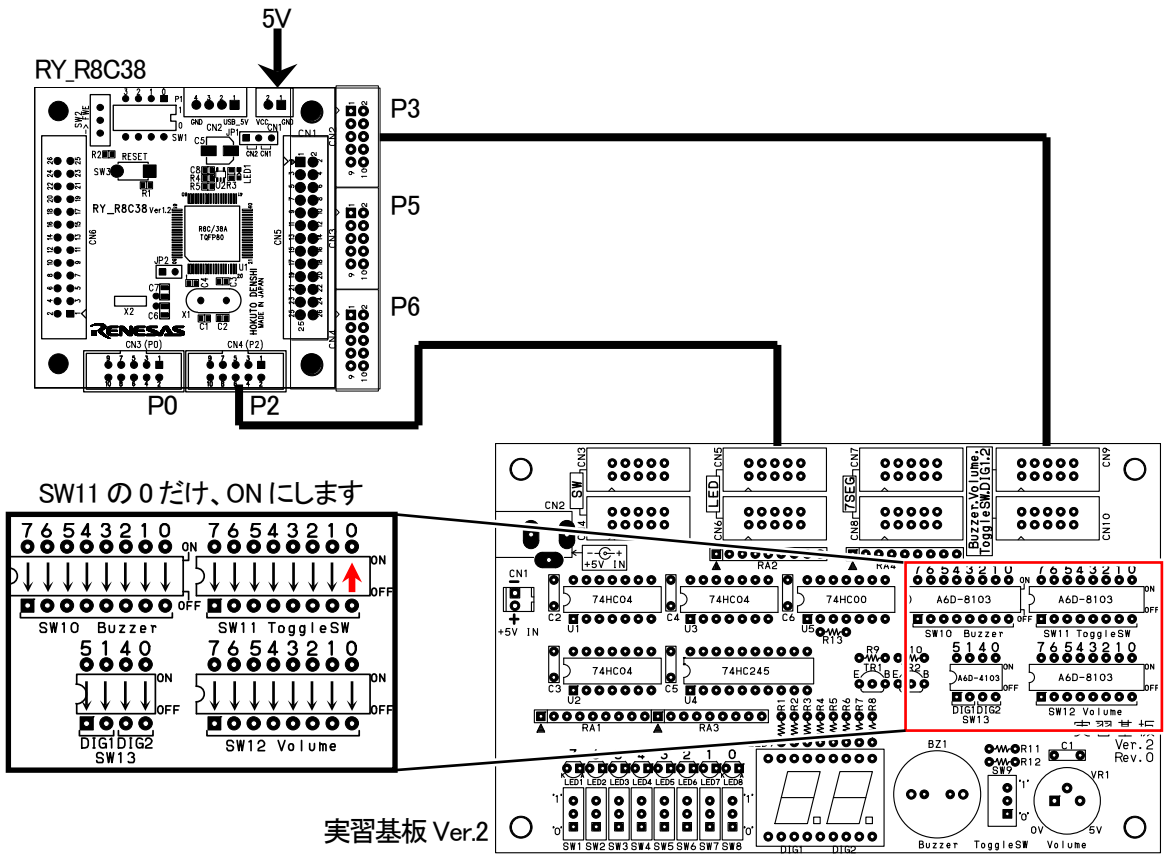
20.2　接続

■使用ポート

マイコンのポート	接続内容
P3_0 (変換基板 CN2)	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。
P2 (CN4)	実習基板 Ver.2 の LED 部など、出力機器を接続します。

■接続例

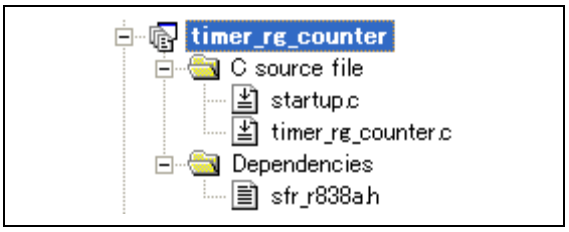
実習基板 Ver.2 を使ったときの接続例を下記に示します。



■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。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥timer_rg_counter¥startup.c
2	timer_rg_counter.c	実際に制御するプログラムが書かれています。R8C/38A の内蔵周辺機能(SFR)の初期化も行います。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥timer_rg_counter¥timer_rg_counter.c
3	sfr_r838a.h	R8C/38A マイコンの内蔵周辺機能を制御するためのレジスタ（Special Function Registers)を定義したファイルです。 ファイルの位置→C:¥Workspace¥common_r8c38a¥sfr_r838a.h

20.4 プログラム「timer_rg_counter.c」

```
 1 :  /***********************************************************************/
 2 :  /* 対象マイコン R8C/38A                                                */
 3 :  /* ﾌｧｲﾙ内容     タイマRGによるパルス数計測                             */
 4 :  /* バージョン   Ver.1.00                                               */
 5 :  /* Date         2010.04.01                                             */
 6 :  /* Copyright    ジャパンマイコンカーラリー実行委員会                   */
 7 :  /***********************************************************************/
 8 :  /*
 9 :  入力：TRGCLKA端子(P3_0)（エンコーダ、または実習基板Ver.2のトグルスイッチ部）
10 :  出力：P2_7-P2_0(実習基板Ver.2のLED部)
11 :
12 :  P3_0に入力したパルスの数を、ポート2の実習基板Ver.2のLED部へ出力します。
13 :  */
14 :
15 :  /*====================================*/
16 :  /* インクルード                       */
17 :  /*====================================*/
18 :  #include "sfr_r838a.h"                 /* R8C/38A SFRの定義ファイル    */
19 :
20 :  /*====================================*/
21 :  /* シンボル定義                       */
22 :  /*====================================*/
23 :
24 :  /*====================================*/
25 :  /* プロトタイプ宣言                   */
26 :  /*====================================*/
27 :  void init( void );
28 :
29 :  /***********************************************************************/
30 :  /* メインプログラム                                                    */
31 :  /***********************************************************************/
32 :  void main( void )
33 :  {
34 :      init();                           /* 初期化                       */
35 :
36 :      while( 1 ) {
37 :          p2 = trg;
38 :      }
39 :  }
40 :
41 :  /***********************************************************************/
42 :  /* R8C/38A スペシャルファンクションレジスタ(SFR)の初期化               */
43 :  /***********************************************************************/
44 :  void init( void )
45 :  {
46 :      int i;
47 :
48 :      /* クロックをXINクロック(20MHz)に変更 */
49 :      prc0  = 1;                        /* プロテクト解除               */
50 :      cm13  = 1;                        /* P4_6,P4_7をXIN-XOUT端子にする*/
51 :      cm05  = 0;                        /* XINクロック発振              */
52 :      for(i=0; i<50; i++ );             /* 安定するまで少し待つ(約10ms) */
53 :      ocd2  = 0;                        /* システムクロックをXINにする  */
54 :      prc0  = 0;                        /* プロテクトON                 */
55 :
56 :      /* ポートの入出力設定 */
57 :      prc2 = 1;                         /* PD0のプロテクト解除          */
58 :      pd0 = 0x00;                       /*                              */
59 :      pd1 = 0xd0;                       /* 5:RXD0 4:TXD0 3-0:DIP SW     */
60 :      pd2 = 0xff;                       /* 7-0:LED                      */
61 :      pd3 = 0xfe;                       /* 0:パルス入力                 */
62 :      p4  = 0x20;                       /* P4_5のLED:初期は点灯         */
63 :      pd4 = 0xb8;                       /* 7:XOUT 6:XIN 5:LED 2:VREF    */
64 :      pd5 = 0xff;                       /*                              */
65 :      pd6 = 0xff;                       /*                              */
66 :      pd7 = 0xff;                       /*                              */
67 :      pd8 = 0xff;                       /*                              */
68 :      pd9 = 0x3f;                       /*                              */
69 :      pur0 = 0x04;                      /* P1_3～P1_0のプルアップON     */
70 :
71 :      /* タイマRG タイマモード(両エッジでカウント)の設定 */
72 :      timsr = 0x40;                     /* TRGCLKA端子 P3_0に割り当てる */
73 :      trgcr = 0x15;                     /* TRGCLKA端子の両エッジでカウント*/
74 :      trgmr = 0x80;                     /* TRGのカウント開始            */
75 :  }
76 :
77 :  /***********************************************************************/
78 :  /* end of file                                                         */
79 :  /***********************************************************************/
```

20.5 ロータリエンコーダ（1相）

20.5.1 概要

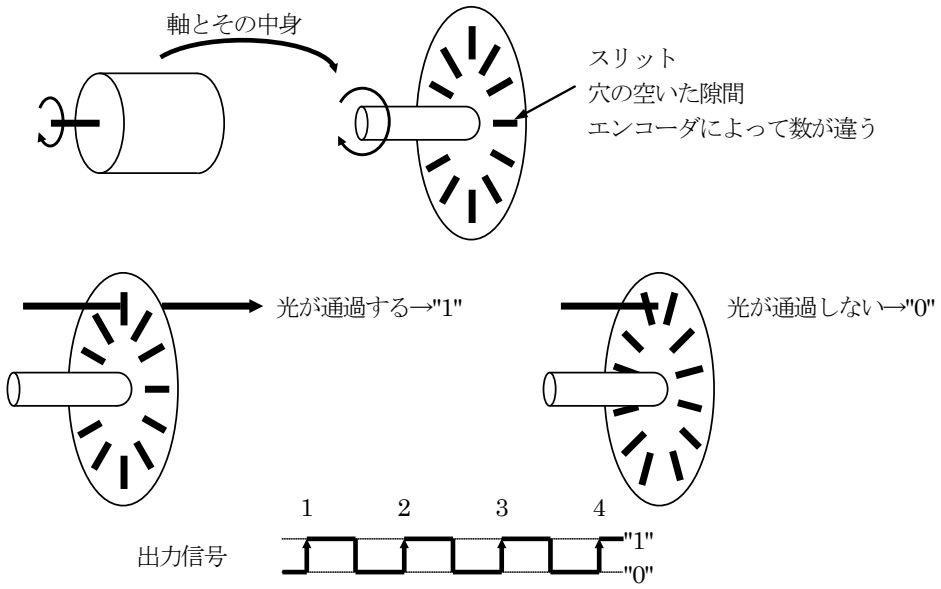
マイコンカーの後ろに、モータと形状が似ていてタイヤが付いているマシンがあります。これがロータリエンコーダと呼ばれる装置です。ロータリエンコーダを使用すると、走行距離や現在のスピードが分かります。

ロータリエンコーダの例
日本電産ネミコン（株） OME-100-1CA-105-015-00

20.5.2 原理

ロータリエンコーダとは、どのような物でしょうか。「ロータリ(rotary)」は、「回転する」という意味です。「エンコーダ(encoder)」は、電気でよく使われる言葉で「符号化する装置」という意味です。合わせると「ロータリエンコーダ」は、「回転を符号化（数値化）する装置」ということになります。

原理は、回転軸に薄い円盤が付いています。その円盤にはスリットと呼ばれる小さい隙間を空けておきます。円盤のある一点に光を通して、通過すれば″1″、しなければ″0″とします。スリットの数は、1 つの円盤に 10 個程度から数千個程度まで様々あります。当然スリット数の多い方が、値段が高くなります。



″0″から″1″になる回数を数えれば、距離が分かります。また、ある一定時間、例えば 1 秒間の回数をカウントして、多ければ回転が速い（＝スピードが速い）、少なければ回転が遅い（＝スピードが遅い）と判断できます。

20.5.3 市販されているロータリエンコーダ（1相出力）

市販されているロータリエンコーダでマイコンカーに使用できそうなエンコーダを以下に示します。他にもたくさんありますので、調べてみると良いでしょう。

メーカ	型式	特徴
日本電産ネミコン（株）	OME-100-1CA-105-015-00	方形波が出力されるので、マイコンで扱いやすいです。プルアップ抵抗だけで使用可能です。 1 回転 100 パルス～300 パルス程度まであります。
日本電産コパル（株）	RE12D-100-101-1	方形波が出力されるので、マイコンで扱いやすいです。プルアップされているのでプルアップ抵抗不要です。φ12mm と小型です。

20.5.4 ロータリエンコーダの自作

ロータリエンコーダは精密機器のため、高価です。マイコンカーラリー販売サイトから安価なエンコーダキットが販売されています。1 回転のパルス数が少ないですが、マイコンカーでの使用では問題ない範囲です。

販売メーカ	型式	特徴
日立インターメディックス（株）	M-S145	1 回転 36 パルスで少ないですが、プログラムで立ち上がり、立ち下がりでカウントに設定することにより 1 回転 72 パルスにすることができます。市販のエンコーダに近い性能となっています。 1 セットで 2 台分のエンコーダが製作できます。

※日立インターメディックス　マイコンカーラリー販売サイトアドレス　http://www2.himdx.net/mcr/

20.6 プログラムの解説

20.6.1 init 関数(ポートの入出力設定)

今回パルス入力として使用する P3_0 端子を、入力に設定します。忘れやすいので、気をつけてください。

```
56 :      /* ポートの入出力設定 */
57 :      prc2 = 1;                       /* PD0のプロテクト解除          */
58 :      pd0 = 0x00;                     /*                              */
59 :      pd1 = 0xd0;                     /* 5:RXD0 4:TXD0 3-0:DIP SW     */
60 :      pd2 = 0xff;                     /* 7-0:LED                      */
61 :      pd3 = 0xfe;                     /* 0:パルス入力                 */
62 :      p4  = 0x20;                     /* P4_5のLED:初期は点灯         */
63 :      pd4 = 0xb8;                     /* 7:XOUT 6:XIN 5:LED 2:VREF    */
64 :      pd5 = 0xff;                     /*                              */
65 :      pd6 = 0xff;                     /*                              */
66 :      pd7 = 0xff;                     /*                              */
67 :      pd8 = 0xff;                     /*                              */
68 :      pd9 = 0x3f;                     /*                              */
69 :      pur0 = 0x04;                    /* P1_3～P1_0のプルアップON     */
```

20.6.2 init 関数(タイマ RG の設定)

タイマ RG を設定するプログラムは、次のようになります。

```
71 :      /* タイマRG タイマモード(両エッジでカウント)の設定 */
72 :      timsr = 0x40;                   /* TRGCLKA端子 P3_0に割り当てる */
73 :      trgcr = 0x15;                   /* TRGCLKA端子の両エッジでカウント*/
74 :      trgmr = 0x80;                   /* TRGのカウント開始            */
```

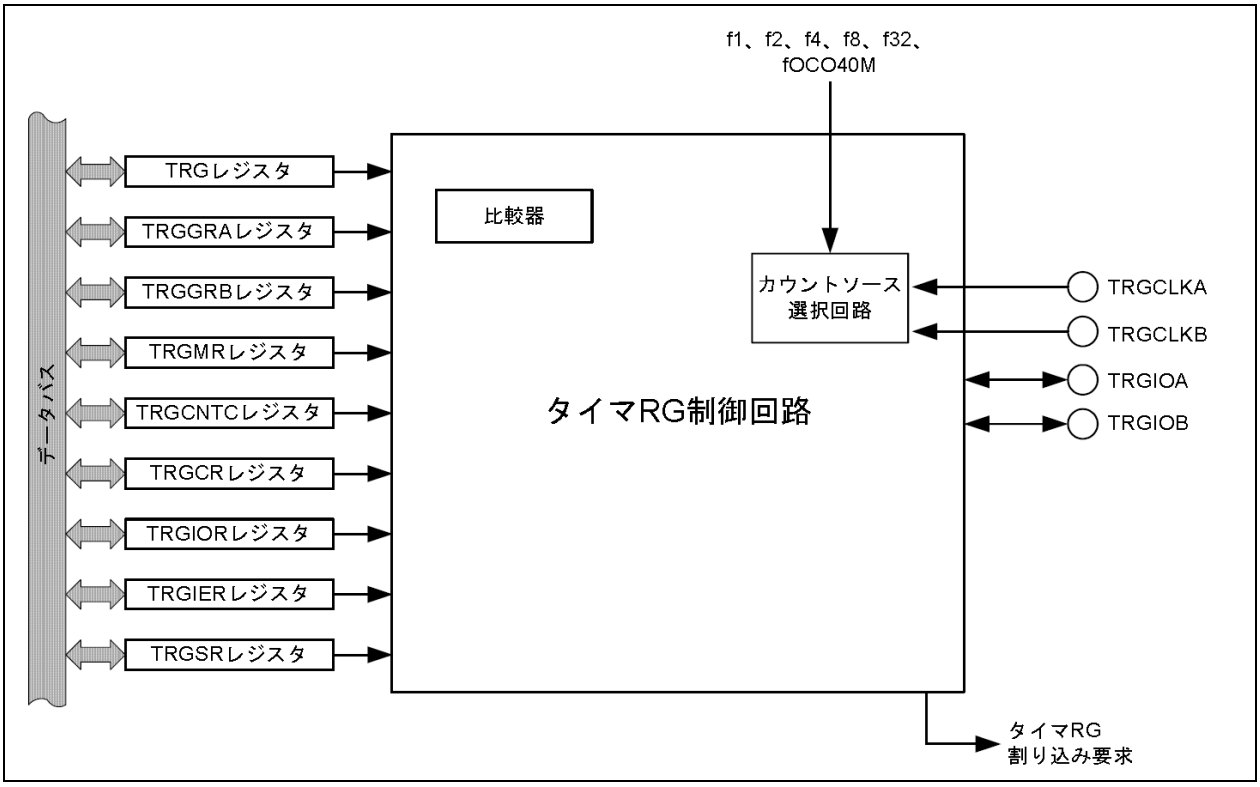
(1) タイマ RG とは

R8C/38A には、タイマ RG というタイマが 1 個内蔵されています。タイマ RG には、3 つのモードがあります。

モード	詳細
タイマモード	●インプットキャプチャ機能 立ち上がりエッジ、立ち下がりエッジ、立ち上がり/ 立ち下がりの両エッジでカウント ●アウトプットコンペア機能 “L”出力/“H”出力/トグル出力
PWM モード	任意デューティの PWM 出力が可能
位相計数モード	2 相エンコーダのカウント数の自動計測が可能

今回は、タイマモードで使いますが、インプットキャプチャ機能もアウトプットコンペア機能も使いません。外部からのパルスを数えるだけです。信号は、立ち上がりエッジ、立ち下がりエッジ、両エッジでのカウントが可能です。

(2) タイマ RG のブロック図



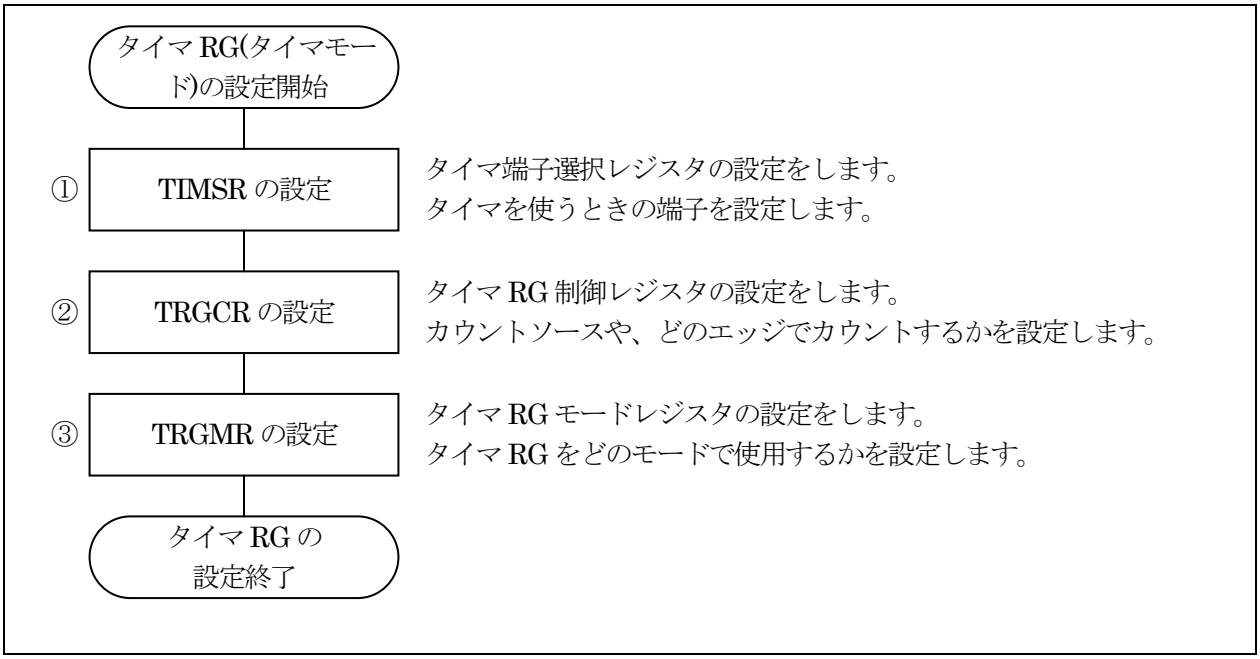
※タイマ RG の端子構成

端子名	割り当てる端子	入出力	機能
TRGCLKA	P3_0	入力	●位相計数モード時 A 相入力 ●位相計数モード時以外 外部クロック A 入力
TRGCLKB	P3_2	入力	●位相計数モード時 B 相入力 ●位相計数モード時以外 外部クロック B 入力
TRGIOA	P5_6	入出力	●タイマモード(アウトプットコンペア機能)時 TRGGRA アウトプットコンペア出力 ●タイマモード(インプットキャプチャ機能)時 TRGGRA インプットキャプチャ入力 ●PWM モード時 PWM 出力
TRGIOB	P5_7	入出力	●タイマモード(アウトプットコンペア機能)時 TRGGRB アウトプットコンペア出力 ●タイマモード(インプットキャプチャ機能)時 TRGGRB インプットキャプチャ入力

今回のプログラムでは、P3_0 端子を外部クロック A 入力端子として使用します。他に P3_2 端子を外部クロック B 入力端子としてパルス入力することができます。

(3) タイマ RG の設定（タイマモード）

今回は、タイマ RG をタイマモードに設定して、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タイマ端子選択レジスタ（TIMSR：Timer Pin Select Register）の設定

タイマを使うときの端子を設定します。

設定 bit	上：ビット名 下：シンボル	内容	今回の内容
bit7	TRGCLKB 端子選択ビット trgclkbsel	0:割り当てない 1:P3_2 に割り当てる 今回は、割り当てません。P3_2 端子でパルスカウントする場合は、この bit を設定してください。	0
bit6	TRGCLKA 端子選択ビット trgclkasel	0:割り当てない 1:P3_0 に割り当てる 今回は、P3_0 を使用するので"1"を設定します。	1
bit5	TRGIOB 端子選択ビット trgiobsel	0:割り当てない 1:P5_7 に割り当てる 使いません。	0
bit4	TRGIOA 端子選択ビット trgioasel	0:割り当てない 1:P5_6 に割り当てる 使いません。	0
bit3		"0"を設定	0
bit2	TRFI 端子選択ビット trfisel0	0:割り当てない 1:P8_3 に割り当てる 使いません。	0
bit1		"0"を設定	0
bit0	TREO 端子選択ビット treosel0	0：P0_4 に割り当てる 1：P6_0 に割り当てる 使いません。	0

タイマ端子選択レジスタ（TIMSR）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			

②タイマ RG 制御レジスタ(TRGCR：Timer RG Control Register)の設定

カウントソースやエッジのカウント方法を設定します。

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
bit6,5	TRG レジスタクリア要因選択ビット bit6:cclr1_trgcr bit5:cclr0_trgcr	00：クリア禁止 01：インプットキャプチャまたは TRGGRA のコンペア一致で TRG レジスタをクリア 10：インプットキャプチャまたは TRGGRB のコンペア一致で TRG レジスタをクリア 11：設定しないでください クリアしません。″00″を設定します。	00
bit4,3	外部クロック有効エッジ選択ビット(注 1) bit4:ckeg1_trgcr bit3:ckeg0_trgcr	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：立ち上がり/立ち下がりの両エッジでカウント 11：設定しないでください 両エッジでカウントします。″10″を設定します。	10
bit2～0	カウントソース選択ビット(注 1) bit2:tck2_trgcr bit1:tck1_trgcr bit0:tck0_trgcr	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：TRGCLKA 入力(P3_0 端子から入力) 110：fOCO40M 111：TRGCLKB 入力(P3_2 端子から入力) P3_0 端子から入力するので″101″を設定します。	101

注 1. 位相計数モードのとき、TCK0～TCK2 ビット、および CKEG0～CKEG1 ビット設定は無効になり、位相計数モードの動作が優先されます。

タイマ RG 制御レジスタ(TRGCR)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**	**1**	**0**	**1**
16 進数	1				5			

③タイマ RG モードレジスタ(TRGMR：Timer RG Mode Register)の設定

タイマ RG の使用モード、カウントの開始を設定します。

設定 bit	上：ビット名 下：シンボル	内容	今回の内容
bit7	TRG カウント開始ビット tstart_trgmr	0：カウント停止 1：カウント開始 TRG(タイマ RG カウンタ)のカウント開始します。″1″を設定します。	1
bit6		″0″を設定	0
bit5,4	デジタルフィルタ機能で使用するクロック選択ビット bit5:dfck1_trgmr bit4:dfck0_trgmr	00：f32 01：f8 10：f1 11：TRGCR レジスタの TCK0～2 で選択したクロック デジタルフィルタ機能は使用しませんのでどれを設定しても構いませんが、″00″を設定しておきます。	00
bit3	TRGIOB 端子のデジタルフィルタ機能選択ビット dfb_trgmr	0：デジタルフィルタ機能なし 1：デジタルフィルタ機能あり デジタルフィルタ機能は使いません。	0
bit2	TRGIOA 端子のデジタルフィルタ機能選択ビット dfa_trgmr	0：デジタルフィルタ機能なし 1：デジタルフィルタ機能あり デジタルフィルタ機能は使いません。	0
bit1	位相計数モード選択ビット mdf_trgmr	0：アップカウント 1：位相計数モード アップカウントです。″0″を設定します。	0
bit0	PWM モード選択ビット pwm_trgmr	0：タイマモード 1：PWM モード タイマモードで使用します。″0″を設定します。	0

タイマ RG モードレジスタ(TRGMR)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			

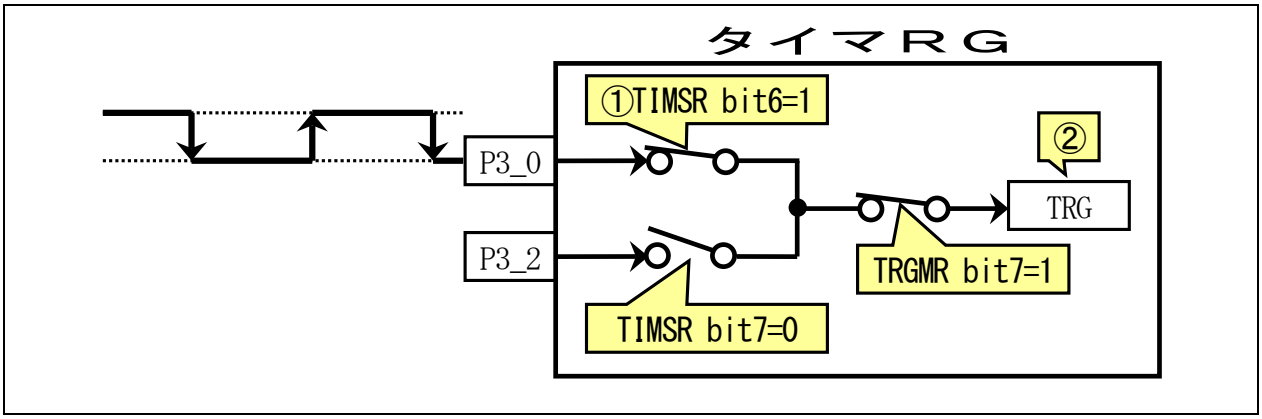
20.6.3 main 関数

タイマ RG でパルスカウントを行い、カウントした値をポート 2 へ出力します。

```
29 :  /************************************************************************/
30 :  /* メインプログラム                                                     */
31 :  /************************************************************************/
32 :  void main( void )
33 :  {
34 :      init();                             /* 初期化                       */
35 :
36 :      while( 1 ) {
37 :          p2 = trg;
38 :      }
39 :  }
```

37 行	タイマ RG カウンタ(TRG)は、P3_0 端子からパルスが入力されるたびに値が増えていきます。信号は、タイマ RG 制御レジスタ(TRGCR)で、「立ち上がり/立ち下がりの両エッジでカウント」を設定しているため、1 パルスで 2 カウントします。 ロータリエンコーダで 1 回転のカウント数が少ない場合は有効な機能です。

タイマ RG カウンタ(TRG)は、P3_0 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 RG 制御レジスタ(TRGCR)のカウントソース選択ビットで設定した端子です。今回は、P3_0 端子に割り当てたので、P3_0 端子に入力されたパルスになります。 また、同レジスタの外部クロック有効エッジ選択ビットで、立ち上がりエッジでカウントするか、立ち下がりエッジでカウントするか、両方でカウントするか選びます。今回は両方でカウントします。
②	タイマ RG カウンタ(TRG)の値はパルスが入力されると、0→1→2・・・と値が増えていきます。最大は、65535 でこの次は 0 に戻ります。

20.7 演習

(1) パルス入力端子を、P3_2 端子にしなさい（入出力設定を忘れないようにしてください）。

21. 2 相エンコーダのパルスカウント(プロジェクト:timer_rg_isoukeisu)

21.1 概要

本章ではタイマ RG を使用して、2 相エンコーダからの A 相、B 相信号でパルスを数える方法を説明します。タイマ RG を使うと、プログラムで正転か逆転かチェックしなくても自動でカウンタの値を増減させることができます。

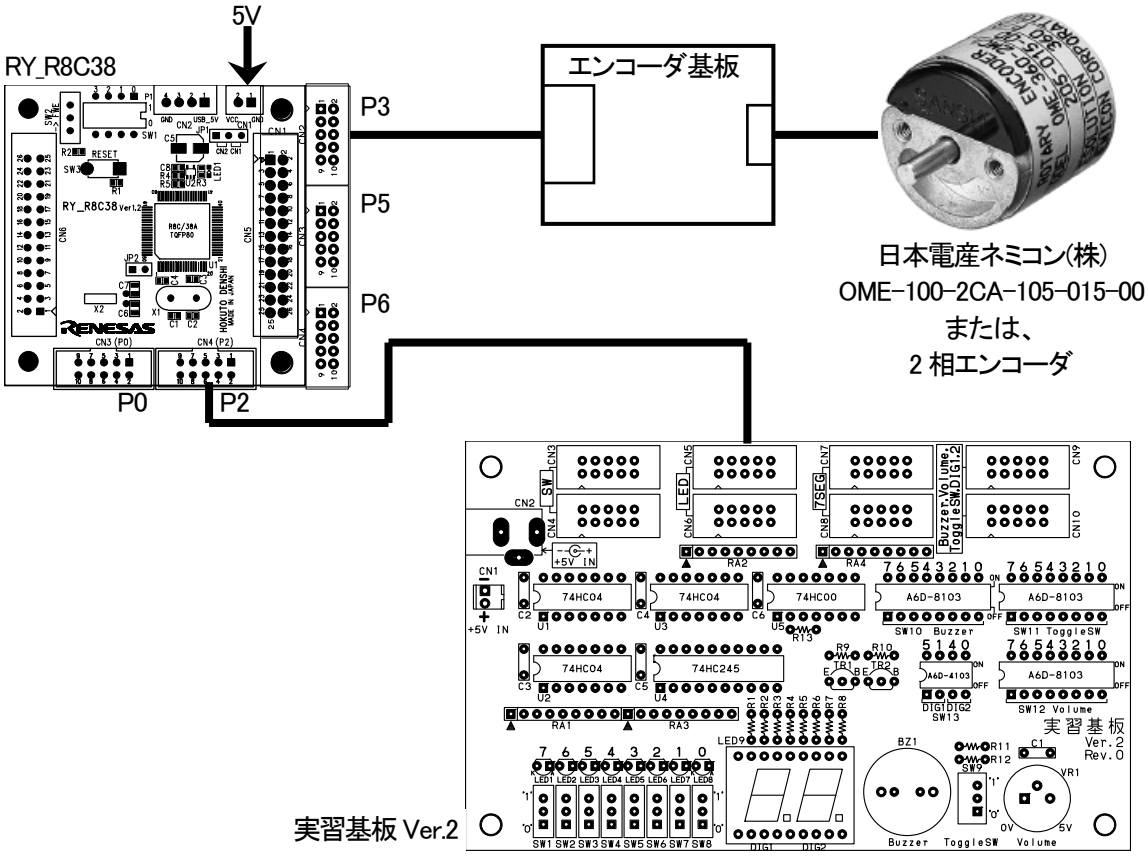
21.2 接続

■使用ポート

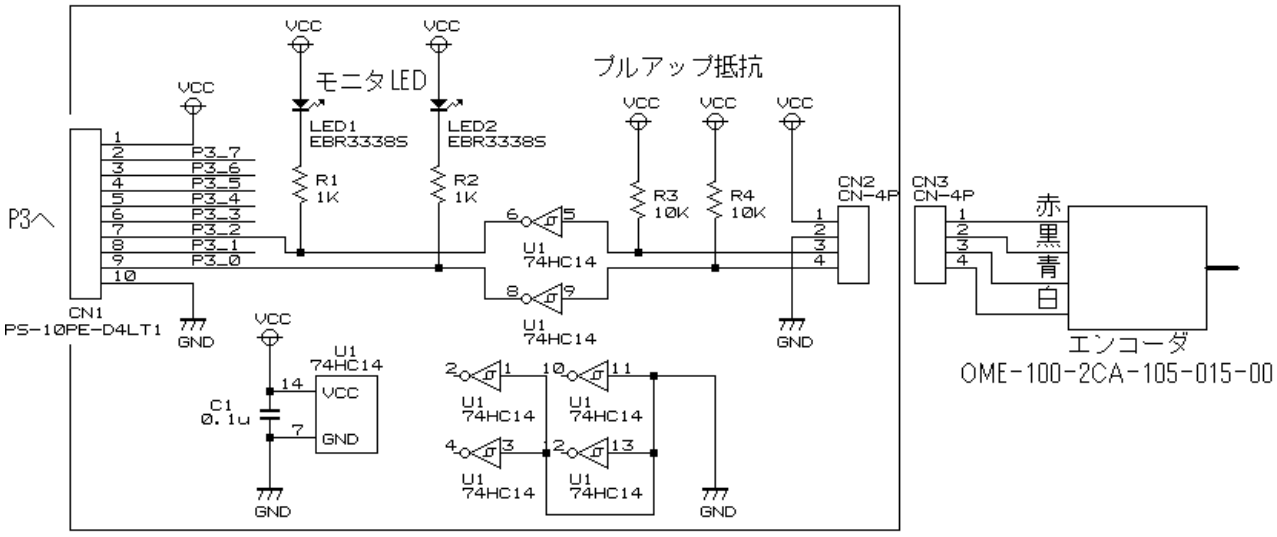
マイコンのポート	接続内容
P3_0、P3_2 (変換基板 CN2)	2 相のエンコーダを接続します。P3_0 には A 相、P3_2 には B 相を接続します。
P2 (CN4)	実習基板 Ver.2 の LED 部など、出力機器を接続します。

■接続例

実習基板 Ver.2 と 2 相エンコーダを使ったときの接続例を下記に示します。2 相エンコーダは、日本電産ネミコン(株)の OME-100-2CA-105-015-00 を使用した回路例も示します。



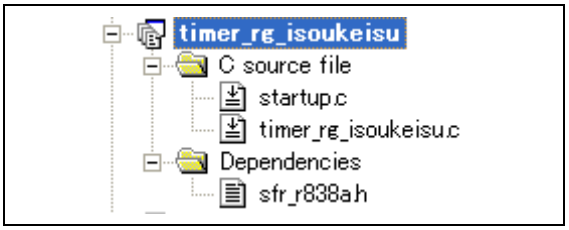
エンコーダ基板の回路例を下図に示します。



■操作方法

ロータリエンコーダの軸を回転させると、実習基板 Ver.2 の LED の値が増減します。値が増えるときの回転方向と減るときの回転方向を確かめてください。また、エンコーダの軸が 1 周するときの LED の増分(減分)を確かめてください。

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。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥timer_rg_isoukeisu¥startup.c
2	timer_rg_isoukeisu.c	実際に制御するプログラムが書かれています。R8C/38A の内蔵周辺機能(SFR)の初期化も行います。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥timer_rg_isoukeisu¥timer_rg_isoukeisu.c
3	sfr_r838a.h	R8C/38A マイコンの内蔵周辺機能を制御するためのレジスタ(Special Function Registers)を定義したファイルです。 ファイルの位置→C:¥Workspace¥common_r8c38a¥sfr_r838a.h

21.4 プログラム「timer_rg_isoukeisu.c」

```
/************************************************************************/
/* 対象マイコン R8C/38A                                                 */
/* ﾌｧｲﾙ内容     タイマRGによるパルス数計測                              */
/* バージョン   Ver.1.00                                                */
/* Date         2010.04.01                                              */
/* Copyright    ジャパンマイコンカーラリー実行委員会                    */
/************************************************************************/
/*
入力：TRGCLKA端子(P3_0)とTRGCLKB端子(P3_2) (2相のエンコーダ)
出力：P2_7-P2_0(実習基板Ver.2のLED部)

2相のエンコーダから入力したパルスの数を、
ポート2の実習基板Ver.2のLED部へ出力します。
*/

/*====================================*/
/* インクルード                       */
/*====================================*/
#include "sfr_r838a.h"                  /* R8C/38A SFRの定義ファイル    */

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

/*====================================*/
/* プロトタイプ宣言                   */
/*====================================*/
void init( void );

/************************************************************************/
/* メインプログラム                                                     */
/************************************************************************/
void main( void )
{
    init();                             /* 初期化                       */

    while( 1 ) {
        p2 = trg;
    }
}

/************************************************************************/
/* R8C/38A スペシャルファンクションレジスタ(SFR)の初期化                */
/************************************************************************/
void init( void )
{
    int i;

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

    /* ポートの入出力設定 */
    prc2 = 1;                           /* PD0のプロテクト解除          */
    pd0 = 0x00;                         /*                              */
    pd1 = 0xd0;                         /* 5:RXD0 4:TXD0 3-0:DIP SW     */
    pd2 = 0xff;                         /* 7-0:LED                      */
    pd3 = 0xfa;                         /* 2,0:2相のエンコーダ          */
    p4  = 0x20;                         /* P4_5のLED:初期は点灯         */
    pd4 = 0xb8;                         /* 7:XOUT 6:XIN 5:LED 2:VREF    */
    pd5 = 0xff;                         /*                              */
    pd6 = 0xff;                         /*                              */
    pd7 = 0xff;                         /*                              */
    pd8 = 0xff;                         /*                              */
    pd9 = 0x3f;                         /*                              */
    pur0 = 0x04;                        /* P1_3～P1_0のプルアップON     */

    /* タイマRG(位相計数モード)の設定 */
    timsr = 0xc0;                       /* TRGCLKA,TRGCLKB端子割り当て  */
    trgcntc = 0xff;                     /* 位相計数ﾓｰﾄﾞのｶｳﾝﾄ方法指定   */
    trgmr = 0x82;                       /* TRGのカウント開始            */
}

/************************************************************************/
/* end of file                                                          */
/************************************************************************/
```

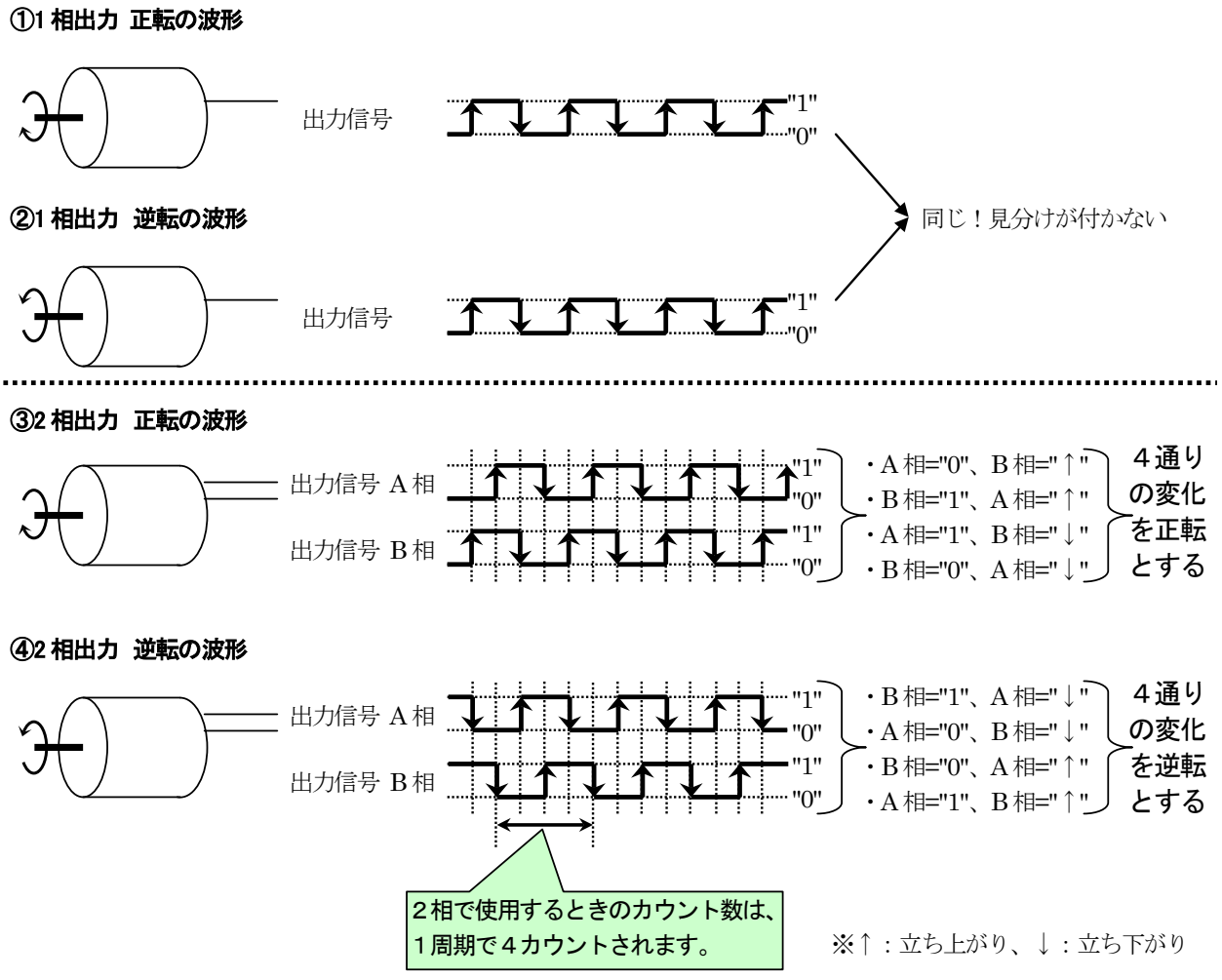
21.5 ロータリエンコーダ（2相）

21.5.1 概要

プロジェクト「timer_rg_counter」を実習すると、あることに気づきます。正転させても逆転させても LED の値が増えていきます。1 相のエンコーダの場合、正転の出力信号と逆転の出力信号では、区別が付かないためです。
2 相出力のあるエンコーダを使えば正転と逆転を判別することができます。

21.5.2 2 相出力のロータリエンコーダ

エンコーダには、1 相出力と 2 相出力があります。
2 相出力のロータリエンコーダは回転検出部分を 2 組用意し、2 組のパルスが出力されるようにします。信号の名前は、一方を A 相、もう一方を B 相とします。A 相と B 相は、同じ信号を出力しても意味が無いので、B 相の光検出を 90 度分ずらして、A 相より 90 度分ずれるようにしています。出力波形の様子を下図に示します。



身近な例では、パソコンのマウス（光学式ではなくボール式）には 2 相のエンコーダが2つ付いています。1 つ目が左右の検出、もう一つで上下の検出をしています。

21.5.3 市販されているロータリエンコーダ(2 相出力)

市販されている 2 相出力のエンコーダで、マイコンカーに使用できそうなエンコーダを以下に示します。他にもたくさんありますので、調べてみると良いでしょう。

メーカ	型式	特徴
日本電産ネミコン(株)	OME-100-**2**CA-105-015-00	方形波が出力されるので、マイコンで扱いやすい。プルアップ抵抗だけで使用可能。
日本電産コパル(株)	RE12D-100-**2**01-1	方形波が出力されるので、マイコンで扱いやすい。プルアップも不要。φ12mm と小型。

21.6 プログラムの解説

21.6.1 init 関数(ポートの入出力設定)

P3_0 端子、P3_2 端子を入力に設定します。忘れやすいので、気をつけてください。

```
57 :      /* ポートの入出力設定 */
58 :      prc2 = 1;                    /* PD0のプロテクト解除          */
59 :      pd0 = 0x00;                  /*                              */
60 :      pd1 = 0xd0;                  /* 5:RXD0 4:TXD0 3-0:DIP SW     */
61 :      pd2 = 0xff;                  /* 7-0:LED                      */
62 :      pd3 = 0xfa;                  /* 2,0:2相のエンコーダ          */
63 :      p4  = 0x20;                  /* P4_5のLED:初期は点灯         */
64 :      pd4 = 0xb8;                  /* 7:XOUT 6:XIN 5:LED 2:VREF    */
65 :      pd5 = 0xff;                  /*                              */
66 :      pd6 = 0xff;                  /*                              */
67 :      pd7 = 0xff;                  /*                              */
68 :      pd8 = 0xff;                  /*                              */
69 :      pd9 = 0x3f;                  /*                              */
70 :      pur0 = 0x04;                 /* P1_3～P1_0のプルアップON     */
```

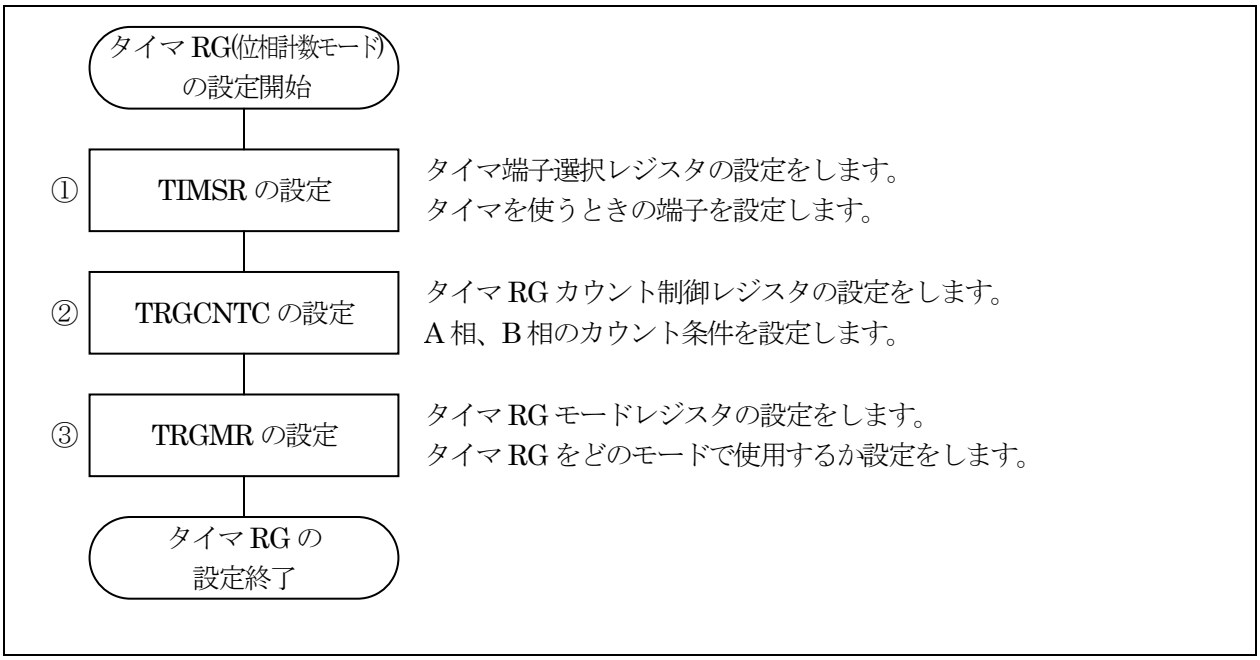
今回のプログラムでは、P3_0 端子をＡ相入力、P3_2 端子をＢ相入力に使用します。この端子は決められており、他の端子に変更することはできません。

21.6.2 init 関数(タイマ RG の設定)

タイマ RG を設定するプログラムは、次のようになります。

```
72 :      /* タイマRG(位相計数モード)の設定 */
73 :      timsr = 0xc0;                    /* TRGCLKA,TRGCLKB端子割り当て   */
74 :      trgcntc = 0xff;                  /* 位相計数ﾓｰﾄﾞのｶｳﾝﾄ方法指定    */
75 :      trgmr = 0x82;                    /* TRGのカウント開始             */
```

今回は、タイマ RG を位相計数モードにして使用します。
レジスタの設定手順を下記に示します。



①タイマ端子選択レジスタ（TIMSR：Timer Pin Select Register）の設定

タイマを使うときの端子を設定します。

設定 bit	上：ビット名 下：シンボル	内容	今回の内容
bit7	TRGCLKB 端子選択ビット trgclkbsel	0:割り当てない 1:P3_2 に割り当てる 今回は、P3_2 をエンコーダの B 相信号入力として使用するので"1"を設定します。	1
bit6	TRGCLKA 端子選択ビット trgclkasel	0:割り当てない 1:P3_0 に割り当てる 今回は、P3_0 をエンコーダの A 相信号入力として使用するので"1"を設定します。	1
bit5	TRGIOB 端子選択ビット trgiobsel	0:割り当てない 1:P5_7 に割り当てる 使いません。	0
bit4	TRGIOA 端子選択ビット trgioasel	0:割り当てない 1:P5_6 に割り当てる 使いません。	0
bit3		"0"を設定	0
bit2	TRFI 端子選択ビット trfisel0	0:割り当てない 1:P8_3 に割り当てる 使いません。	0
bit1		"0"を設定	0
bit0	TREO 端子選択ビット treosel0	0：P0_4 に割り当てる 1：P6_0 に割り当てる 使いません。	0

タイマ端子選択レジスタ（TIMSR）の設定値を下記に示します。

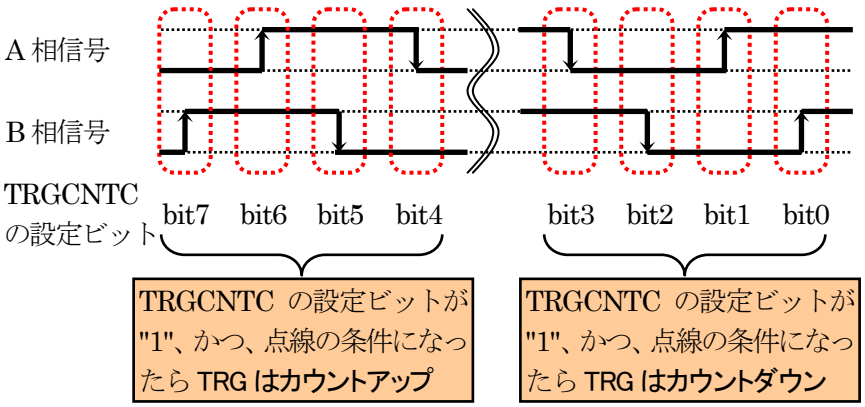
bit	7	6	5	4	3	2	1	0
設定値	**1**	**1**	0	0	0	0	0	0
16 進数	C				O			

②タイマ RG カウント制御レジスタ(TRGCNTC：Timer RG Count Control Register)の設定

2 相の信号のカウントアップ条件、カウントダウン条件を設定します。

設定 bit	上：ビット名 下：シンボル	内容	今回の内容
bit7	カウント許可ビット 7 cnten7_trgcntc	0：無効 1：アップカウント TRGCLKA 入力が“L”レベルで、TRGCLKB 入力の立ち上がりエッジのとき	1
bit6	カウント許可ビット 6 cnten6_trgcntc	0：無効 1：アップカウント TRGCLKB 入力が“H”レベルで、TRGCLKA 入力の立ち上がりエッジのとき	1
bit5	カウント許可ビット 5 cnten5_trgcntc	0：無効 1：アップカウント TRGCLKA 入力が“H”レベルで、TRGCLKB 入力の立ち下がりエッジのとき	1
bit4	カウント許可ビット 4 cnten4_trgcntc	0：無効 1：アップカウント TRGCLKB 入力が“L”レベルで、TRGCLKA 入力の立ち下がりエッジのとき	1
bit3	カウント許可ビット 3 cnten3_trgcntc	0：無効 1：ダウンカウント TRGCLKB 入力が“H”レベルで、TRGCLKA 入力の立ち下がりエッジのとき	1
bit2	カウント許可ビット 2 cnten2_trgcntc	0：無効 1：ダウンカウント TRGCLKA 入力が“L”レベルで、TRGCLKB 入力の立ち下がりエッジのとき	1
bit1	カウント許可ビット 1 cnten1_trgcntc	0：無効 1：ダウンカウント TRGCLKB 入力が“L”レベルで、TRGCLKA 入力の立ち上がりエッジのとき	11
bit0	カウント許可ビット 0 cnten0_trgcntc	0：無効 1：ダウンカウント TRGCLKA 入力が“H”レベルで、TRGCLKB 入力の立ち上がりエッジのとき	1

設定ビットと波形の関係を下記に示します。今回は全ビット″1″なので、下記の条件すべてでカウントダウン、またはカウントアップします。



タイマ RG カウント制御レジスタ(TRGCNTC)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			

③タイマ RG モードレジスタ(TRGMR：Timer RG Mode Register)の設定

タイマ RG をどのモードで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	TRG カウント開始ビット tstart_trgmr	0：カウント停止 1：カウント開始 TRG(タイマ RG カウンタ)のカウント開始します。"1"を設定します。	1
bit6		"0"を設定	0
bit5,4	デジタルフィルタ機能で使用するクロック選択ビット bit5:dfck1_trgmr bit4:dfck0_trgmr	00：f32 01：f8 10：f1 11：TRGCR レジスタの TCK0～2 で選択したクロック デジタルフィルタ機能は使用しませんのでどれを設定しても構いませんが、"00"を設定しておきます。	00
bit3	TRGIOB 端子のデジタルフィルタ機能選択ビット dfb_trgmr	0：デジタルフィルタ機能なし 1：デジタルフィルタ機能あり デジタルフィルタ機能は使いません。	0
bit2	TRGIOA 端子のデジタルフィルタ機能選択ビット dfa_trgmr	0：デジタルフィルタ機能なし 1：デジタルフィルタ機能あり デジタルフィルタ機能は使いません。	0
bit1	位相計数モード選択ビット mdf_trgmr	0：アップカウント 1：位相計数モード 位相計数モードです。"1"を設定します。	1
bit0	PWM モード選択ビット pwm_trgmr	0：タイマモード 1：PWM モード タイマモードで使用します。"0"を設定します。	0

タイマ RG モードレジスタ(TRGMR)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	**1**	0
16 進数	8				2			

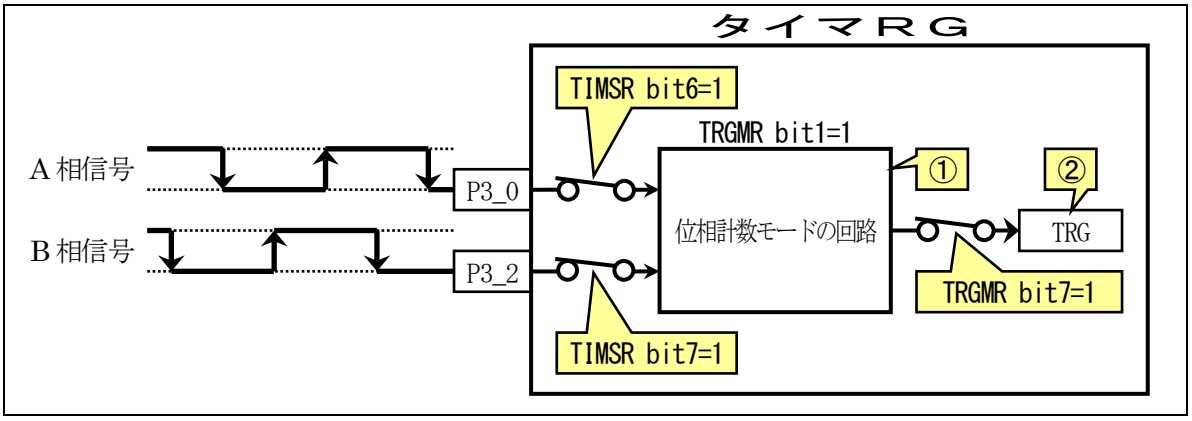
21.6.3 main 関数

タイマ RG の位相計数モードでパルスカウントを行い、カウントした値をポート 2 へ出力します。

```
30 :  /************************************************************************/
31 :  /* メインプログラム                                                     */
32 :  /************************************************************************/
33 :  void main( void )
34 :  {
35 :      init();                             /* 初期化                       */
36 :
37 :      while( 1 ) {
38 :          p2 = trg;
39 :      }
40 :  }
```

38 行	A 相、B 相の入力信号によりタイマ RG カウンタ(TRG)がカウントアップ、カウントダウンされます。その値をポート 2 に出力しています。

タイマ RG カウンタ(TRG)の値は、P3_0 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 RG により、A 相、B 相信号を入力、カウントアップ処理、またはカウントダウン処理します。
②	①の結果により、タイマ RG カウンタ(TRG)はカウントアップ、またはカウントダウンします。最大は 65535 で、この状態でカウントアップすると 0 になります。最小は 0 で、この状態でカウントダウンすると 65535 になります。

22. 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）

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。

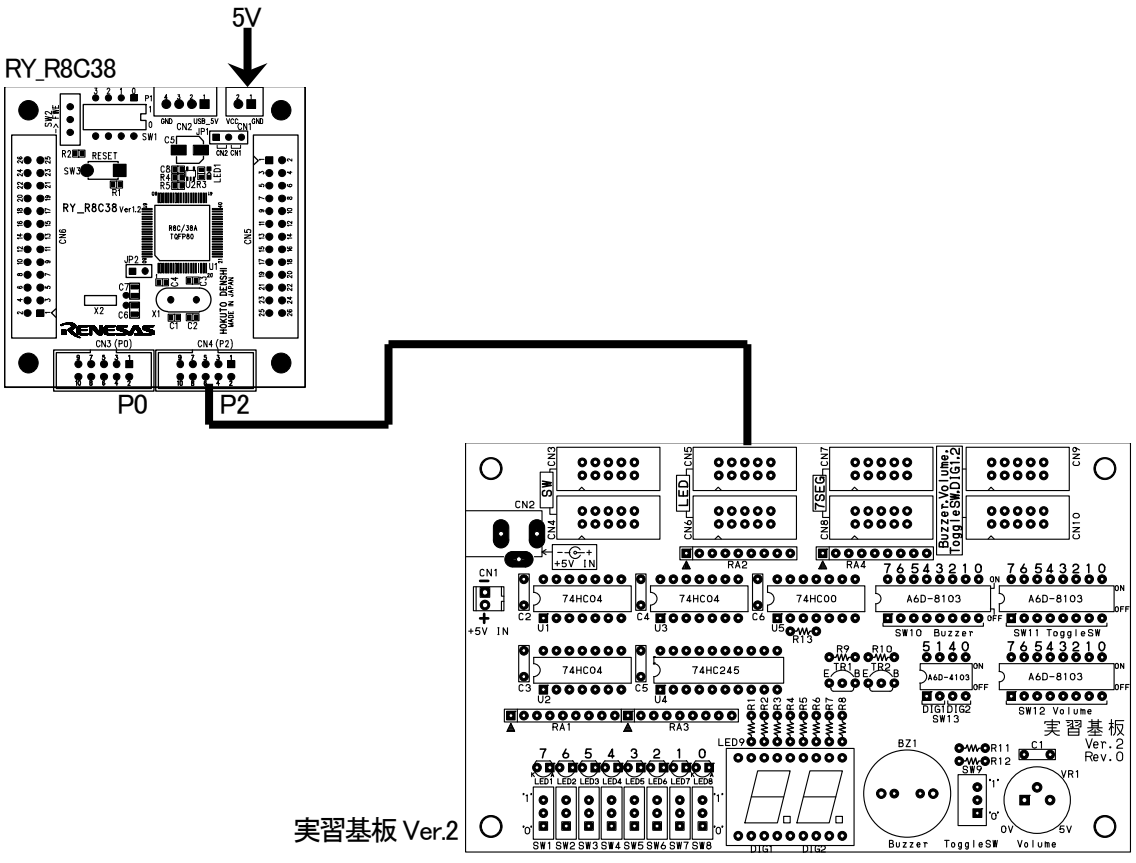
22.2 接続

■使用ポート

マイコンのポート	接続内容
P1_3、P1_2、P1_1、P1_0	RY_R8C38 ボードのディップスイッチです。
P2 (CN4)	実習基板 Ver.2 の LED 部など、出力機器を接続します。

■接続例

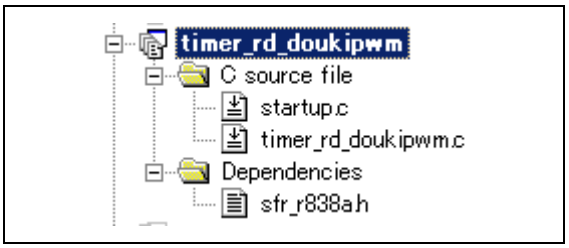
実習基板 Ver.2 を使ったときの接続例を下記に示します。



■操作方法

RY_R8C38 ボードのディップスイッチ(SW1)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。

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。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥timer_rd_doukipwm¥startup.c
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/38A の内蔵周辺機能(SFR)の初期化も行います。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥timer_rd_doukipwm¥timer_rd_doukipwm.c
3	sfr_r838a.h	R8C/38A マイコンの内蔵周辺機能を制御するためのレジスタ（Special Function Registers)を定義したファイルです。 ファイルの位置→C:¥Workspace¥common_r8c38a¥sfr_r838a.h

22.4 プログラム「timer_rd_doukipwm.c」

```
/************************************************************************/
/* 対象マイコン R8C/38A                                                 */
/* ﾌｧｲﾙ内容     タイマRDによるリセット同期PWM                           */
/* バージョン   Ver.1.00                                                */
/* Date         2010.04.01                                              */
/* Copyright    ジャパンマイコンカーラリー実行委員会                    */
/************************************************************************/
/*
入力：マイコンボードのディップスイッチ
出力：P2_1端子～P2_7端子からPWM出力（実習基板Ver.2のLED部）

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。
*/

/*====================================*/
/* インクルード                       */
/*====================================*/
#include "sfr_r838a.h"                  /* R8C/38A SFRの定義ファイル    */

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

/*====================================*/
/* プロトタイプ宣言                   */
/*====================================*/
void init( void );
unsigned char dipsw_get( void );

```

```
  30 :  /************************************************************************/
  31 :  /* メインプログラム                                                     */
  32 :  /************************************************************************/
  33 :  void main( void )
  34 :  {
  35 :      init();                             /* 初期化                       */
  36 :
  37 :      while( 1 ) {
  38 :          trdgrd0 = 39998 * dipsw_get() / 15;
  39 :      }
  40 :  }
  41 :
  42 :  /************************************************************************/
  43 :  /* R8C/38A スペシャルファンクションレジスタ(SFR)の初期化                */
  44 :  /************************************************************************/
  45 :  void init( void )
  46 :  {
  47 :      int i;
  48 :
  49 :      /* クロックをXINクロック(20MHz)に変更 */
  50 :      prc0  = 1;                          /* プロテクト解除               */
  51 :      cm13  = 1;                          /* P4_6,P4_7をXIN-XOUT端子にする*/
  52 :      cm05  = 0;                          /* XINクロック発振              */
  53 :      for(i=0; i<50; i++ );               /* 安定するまで少し待つ(約10ms) */
  54 :      ocd2  = 0;                          /* システムクロックをXINにする  */
  55 :      prc0  = 0;                          /* プロテクトON                 */
  56 :
  57 :      /* ポートの入出力設定 */
  58 :      prc2 = 1;                           /* PD0のプロテクト解除          */
  59 :      pd0 = 0x00;                         /*                              */
  60 :      pd1 = 0xd0;                         /* 5:RXD0 4:TXD0 3-0:DIP SW     */
  61 :      pd2 = 0xff;                         /* 7-0:LED                      */
  62 :      pd3 = 0xff;                         /*                              */
  63 :      p4  = 0x20;                         /* P4_5のLED:初期は点灯         */
  64 :      pd4 = 0xb8;                         /* 7:XOUT 6:XIN 5:LED 2:VREF    */
  65 :      pd5 = 0xff;                         /*                              */
  66 :      pd6 = 0xff;                         /*                              */
  67 :      pd7 = 0xff;                         /*                              */
  68 :      pd8 = 0xff;                         /*                              */
  69 :      pd9 = 0x3f;                         /*                              */
  70 :      pur0 = 0x04;                        /* P1_3～P1_0のプルアップON     */
  71 :
  72 :      /* タイマRD リセット同期PWMモードの設定*/
  73 :      trdpsr0 = 0x68;                     /* TRDIOB0,C0,D0端子設定        */
  74 :      trdpsr1 = 0x55;                     /* TRDIOA1,B1,C1,D1端子設定     */
  75 :      trdmr   = 0xf0;                     /* バッファレジスタ設定         */
  76 :      trdfcr  = 0x01;                     /* リセット同期PWMモードに設定  */
  77 :      trdcr0  = 0x23;                     /* ソースカウントの選択:f8      */
  78 :      trdgra0 = trdgrc0 = 39999;          /* 周期                         */
  79 :      trdgrb0 = trdgrd0 = 0;              /* P2_2端子のON幅設定           */
  80 :      trdgra1 = trdgrc1 = 0;              /* P2_4端子のON幅設定           */
  81 :      trdgrb1 = trdgrd1 = 0;              /* P2_5端子のON幅設定           */
  82 :      trdoer1 = 0x01;                     /* 出力端子の選択               */
  83 :      trdstr  = 0x0d;                     /* TRD0カウント開始             */
  84 :  }
  85 :
  86 :  /************************************************************************/
  87 :  /* ディップスイッチ値読み込み                                           */
  88 :  /* 戻り値 スイッチ値 0～15                                              */
  89 :  /************************************************************************/
  90 :  unsigned char dipsw_get( void )
  91 :  {
  92 :      unsigned char sw;
  93 :
  94 :      sw = p1 & 0x0f;                     /* P1_3～P1_0読み込み           */
  95 :
  96 :      return  sw;
  97 :  }
  98 :
  99 :  /************************************************************************/
 100 :  /* end of file                                                          */
 101 :  /************************************************************************/
```

22.5　PWM

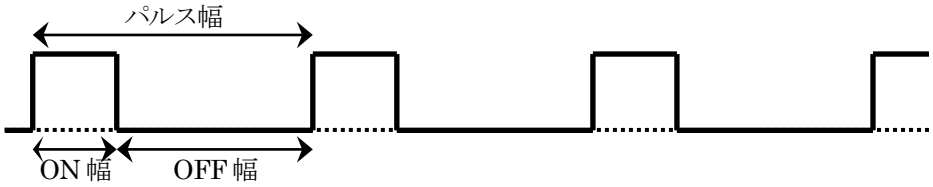
22.5.1　PWM とは？

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よいだけです。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。

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**といいます。パルス幅に対する ON の割合のことを**デューティ比**といいます。周期に対する ON 幅を 50%にするとき、デューティ比 50%といいます。他にも PWM50%とか、単純にモータ 50%といいます。



デューティ比は下記で表すことができます。

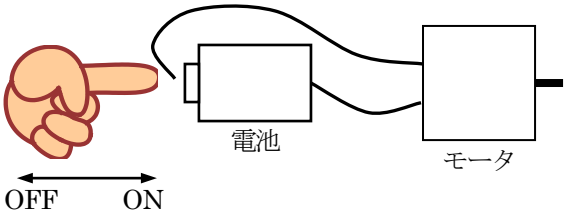
デューティ比＝ON 幅／パルス幅（ON 幅＋OFF 幅）

例えば、100ms のパルスに対して、ON 幅が 60ms なら、

デューティ比＝60ms/100ms＝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ますが、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。

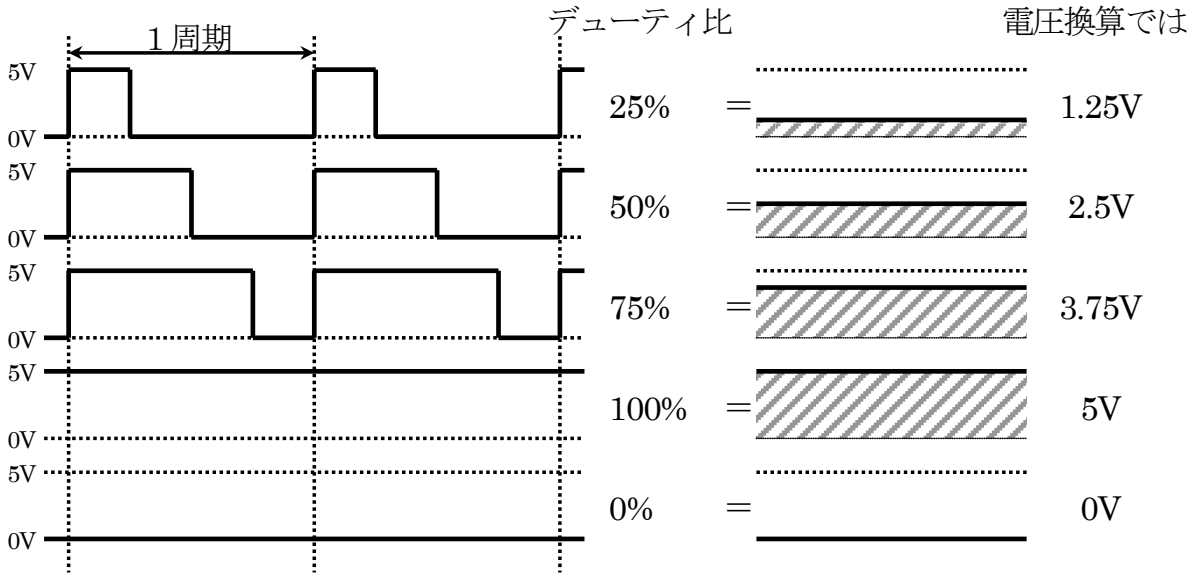


22.5.2 PWM 波形を電圧換算する

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×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変得意ですが、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**。

22.6 プログラムの解説

22.6.1 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。

```
72 :      /* タイマRD リセット同期PWMモードの設定*/
73 :      trdpsr0 = 0x68;                    /* TRDIOB0,C0,D0端子設定          */
74 :      trdpsr1 = 0x55;                    /* TRDIOA1,B1,C1,D1端子設定       */
75 :      trdmr   = 0xf0;                    /* バッファレジスタ設定           */
76 :      trdfcr  = 0x01;                    /* リセット同期PWMモードに設定    */
77 :      trdcr0  = 0x23;                    /* ソースカウントの選択:f8        */
78 :      trdgra0 = trdgrc0 = 39999;         /* 周期                           */
79 :      trdgrb0 = trdgrd0 = 0;             /* P2_2端子のON幅設定             */
80 :      trdgra1 = trdgrc1 = 0;             /* P2_4端子のON幅設定             */
81 :      trdgrb1 = trdgrd1 = 0;             /* P2_5端子のON幅設定             */
82 :      trdoer1 = 0x01;                    /* 出力端子の選択                 */
83 :      trdstr  = 0x0d;                    /* TRD0カウント開始               */
```

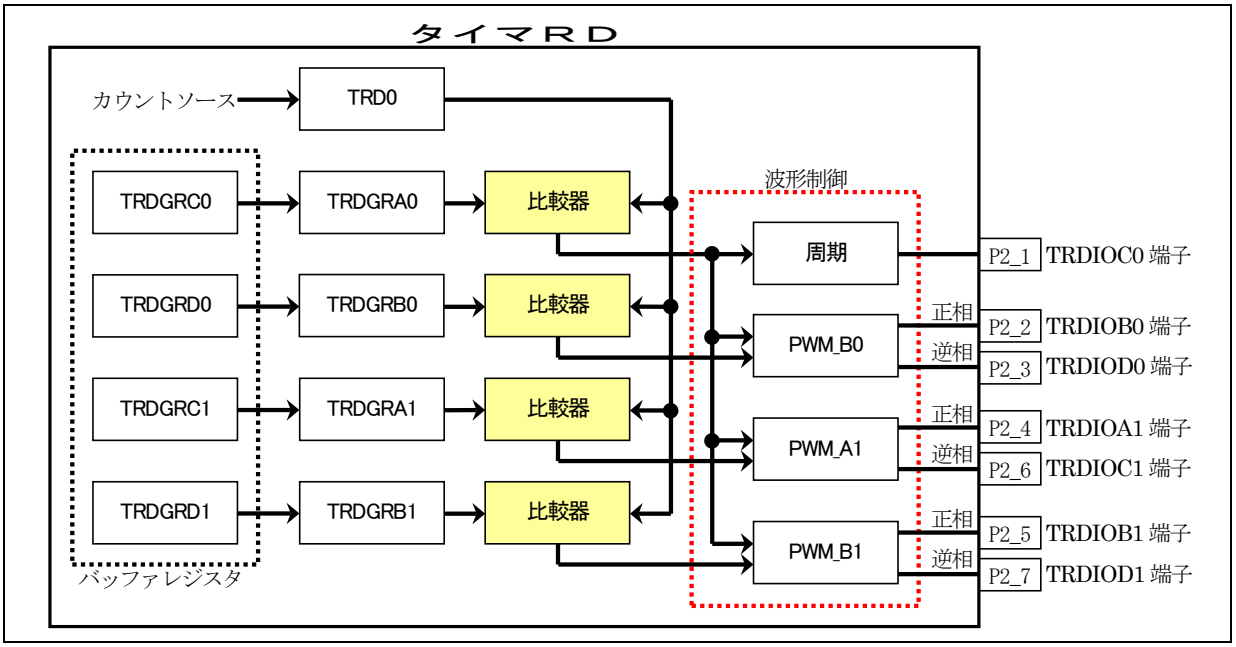
(1) タイマ RD とは

R8C/38A 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。

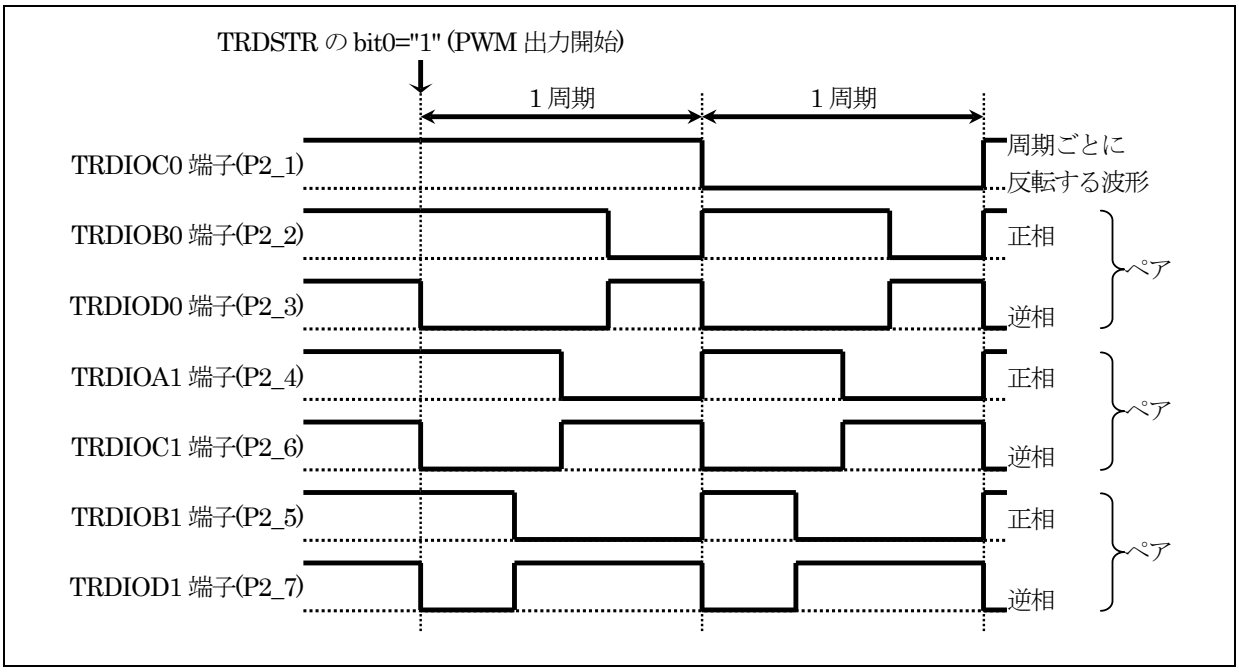
モード	内容
タイマモード	タイマモードには、次の 2 つの機能があります。 ●インプットキャプチャ機能 外部信号をトリガにしてカウンタの値をレジスタに取り込む機能 ●アウトプットコンペア機能 カウンタとレジスタの値の一致を検出する機能(検出時に端子出力変更可能)
PWM モード	任意の幅のパルスを連続して出力するモード
リセット同期 PWM モード	**鋸波変調、短絡防止時間なしの三相波形(6 本)を出力するモード**
相補 PWM モード	三角波変調、短絡防止時間ありの三相波形(6 本)を出力するモード
PWM3 モード	同一周期の PWM 波形(2 本)を出力するモード

(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 のチャネル 0 とチャネル 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 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。

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 trdiod0sel0	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			

②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。

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 trdiod1sel0	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			

③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			

④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。

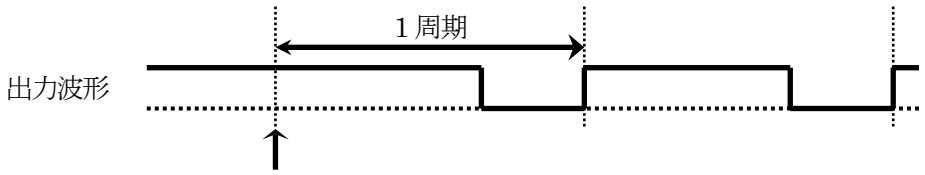
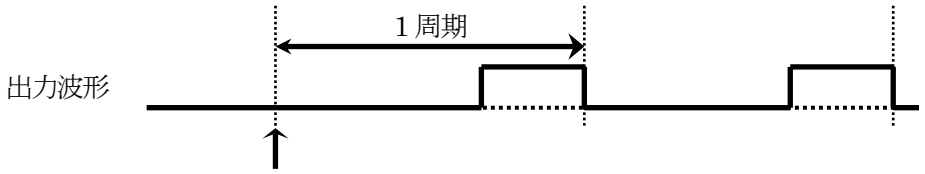
設定 bit	上:ビット名 下:シンボル	内容	今回の内容
bit7		″0″を設定	0
bit6	外部クロック入力選択ビット stclk_trdfcr	0:外部クロック入力無効 1:外部クロック入力有効 今回は、内部のクロックを使用しますので、無効を選択します。	0
bit5,4		″00″を設定	00
bit3	逆相出力レベル選択ビット (リセット同期 PWM モードまたは相補 PWM モード時) ols1_trdfcr	0:初期出力“H”、アクティブレベル“L” 1:初期出力“L”、アクティブレベル“H” 今回は、″0″を設定します。	0
bit2	正相出力レベル選択ビット (リセット同期 PWM モードまたは相補 PWM モード時) ols0_trdfcr	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_trdfcr bit0:cmd0_trdfcr	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			

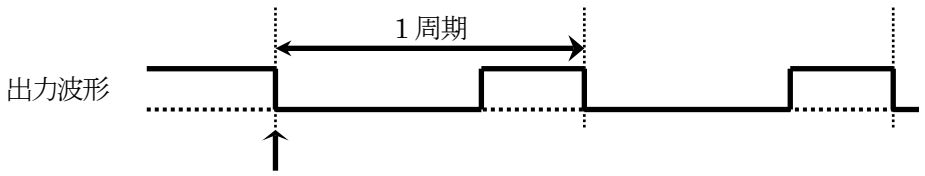
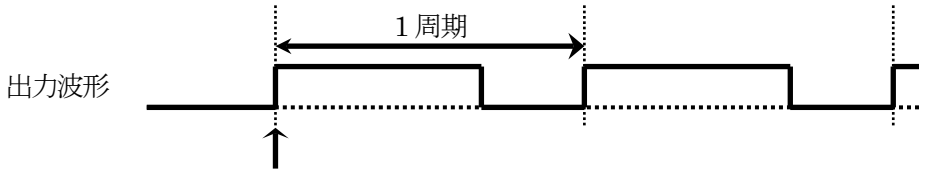
※正相出力レベル選択ビットと出力波形の関係

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)		"1"からスタートする設定です。
1		"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)		"0"からスタートする設定です。
1		"1"からスタートする設定です。

⑤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、 f1/20MHz＝1/20MHz＝50ns 設定できる PWM 周期の最大は、 50ns×65,536＝**3.2768ms** 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、 f2/20MHz＝2/20MHz＝100ns 設定できる PWM 周期の最大は、 100ns×65,536＝**6.5536ms** 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、 f4/20MHz＝4/20MHz＝200ns 設定できる PWM 周期の最大は、 200ns×65,536＝**13.1072ms** 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、 f8/20MHz＝8/20MHz＝400ns 設定できる PWM 周期の最大は、 400ns×65,536＝**26.2144ms** 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、 f32/20MHz＝32/20MHz＝1600ns 設定できる PWM 周期の最大は、 1600ns×65,536＝**104.8576ms** 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。

⑥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。

PWM 周期＝タイマ RD カウンタ 0 のカウントソース×(TRDGRA0＋1)

TRDGRA0 を左辺に移動して、TRDGRA0 を求める式に変形します。

TRDGRA0＝PWM 周期／タイマ RD カウンタ 0 のカウントソースー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。

TRDGRA0＝PWM 周期 ／カウントソースー1
TRDGRA0＝(16×10^{-3}) ／(400×10^{-9})－1
TRDGRA0＝40000－1 ＝ 39999

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。

⑦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。

P2_2 端子から出力される PWM 波形の ON 幅＝タイマ RD カウンタ 0 のカウントソース×(TRDGRB0＋1)

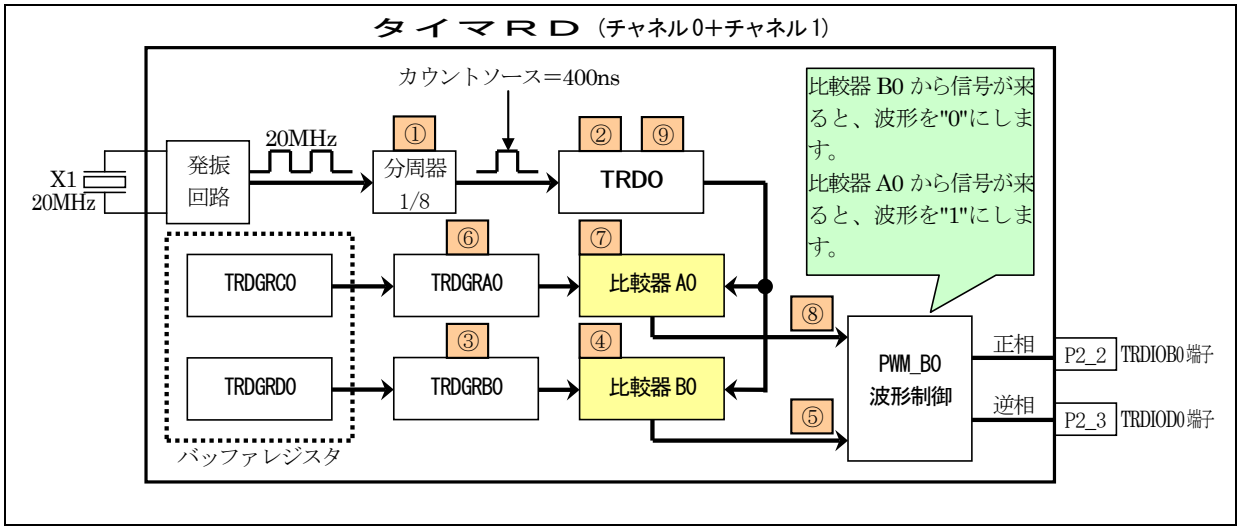
TRDGRB0 を左辺に移動して、TRDGRB0 を求める式に変形します。

TRDGRB0＝P2_2 端子から出力される PWM 波形の ON 幅／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。

TRDGRB0＝ON 幅　／カウントソース－1
TRDGRB0＝$(1\times10^{-3})/(400\times10^{-9})-1$
TRDGRB0＝2500－1　＝　2499

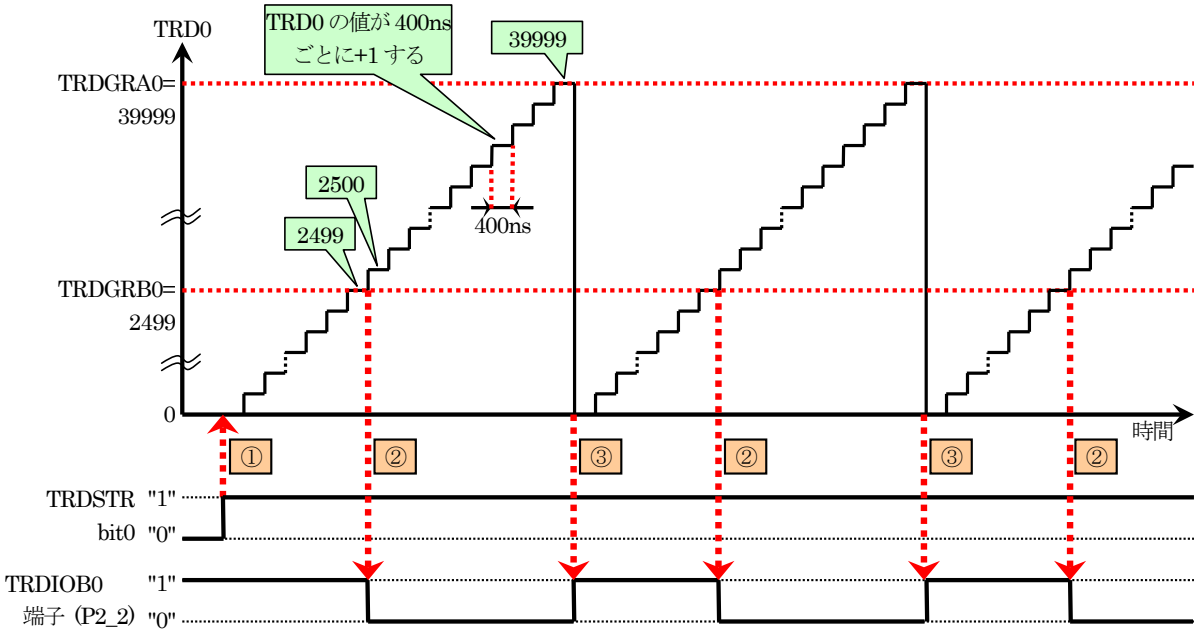
■仕組み



①	分周器には、クリスタルの 20HMz 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 分周するということです。よって分周器からの出力は、 8／20MHz＝2.5MHz のパルスが出力されます。時間は、 1／2.5M＝400ns 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。

③ ④	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。 TRDGRB0＋１＝TRD0 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 2499（TRDGRB0）＋１≠　0（TRD0） となり、成り立ちません。400ns 後、TRD0 は＋1されて 1 になります。 2499（TRDGRB0）＋１≠ 1（TRD0） まだ成り立ちません。式が成り立つ TRD0 が 2500 になるには 2500×400ns＝1ms です。よって、1ms 経つと 2499（TRDGRB0）＋１＝ 2500（TRD0） 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。
⑤	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。
⑥ ⑦	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。 TRDGRA0＋１＝TRD0 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 39999（TRDGRA0）＋１≠ 0（TRD0） となり、成り立ちません。400ns 後、TRD0 は＋1されて 1 になります。 39999（TRDGRA0）＋１≠ 1（TRDGRA0） まだ成り立ちません。式が成り立つ TRD0 が 40000 になるには 40000×400ns＝16ms です。よって、16ms 経つと 39999（TRDGRA0）＋１＝ 40000（TRD0） 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。
⑧ ⑨	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。

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。

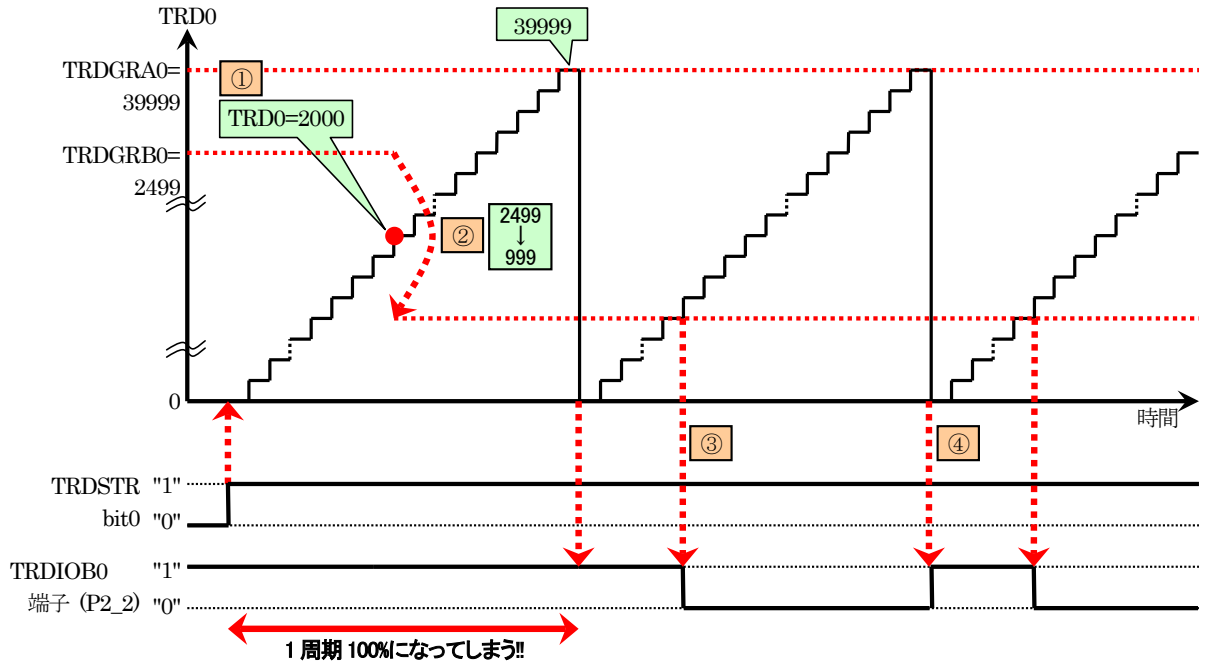


①	TRDSTR の bit0 を″1″にすると、TRD0 のカウントが開始されます。
②	TRD0＝(TRDGRB0＋1)になると、P2_2 端子が″0″になります。 TRDGRB0 の値は 2499、TRD0 が+1 する間隔は 400ns です。 よって、P2_2 端子が″1″の時間は、 P2_2 端子が″1″の時間＝　(TRDGRB0＋1)×TRD0 が+1 する間隔 ＝　(2499＋1)×400ns ＝　1,000,000 [ns] ＝ 1[ms]
③	TRD0＝(TRDGRA0＋1)になると、P2_2 端子が″1″になります。同時に TRD0 が 0 にクリアされ、また 0 から値が増えていきます。 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] 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。

■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。

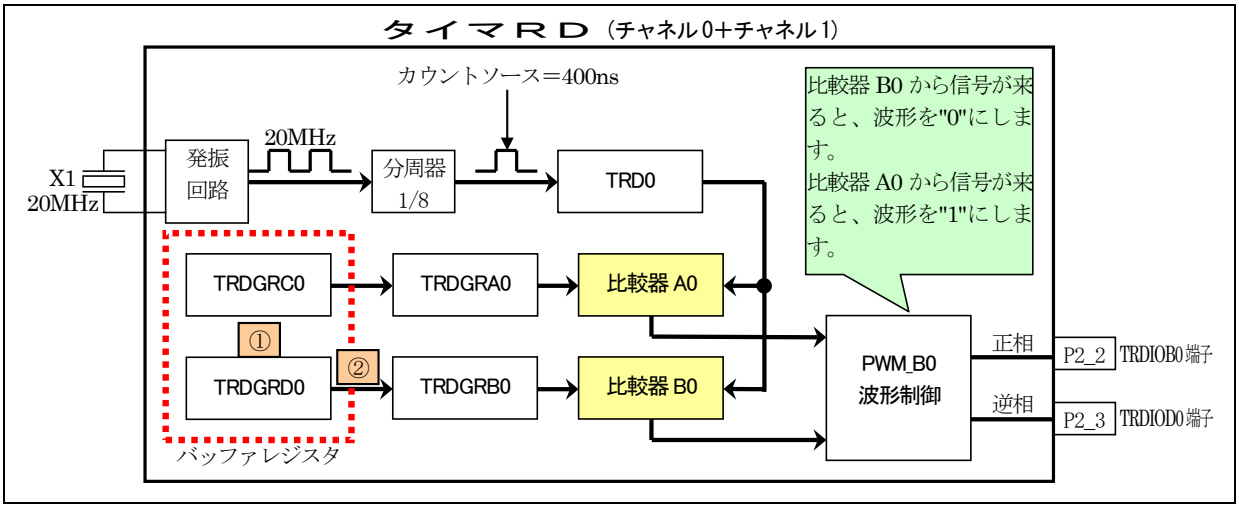
例えば、TRDGRB0=2499、TRD0=2000 のときに、プログラムで TRDGRB0 の値を 999 に変更したとします。



① ②	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。
③	次の周期は、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。同時に TRD0 が 0 になります。

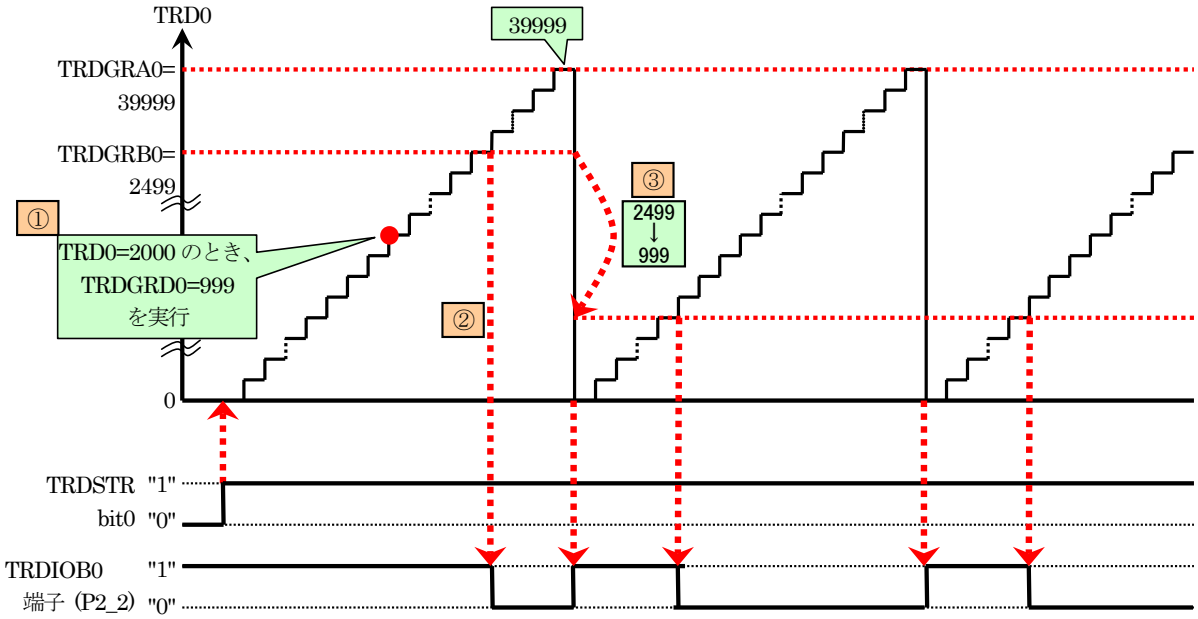
このように、TRDGRB0 の値を変えるタイミングによっては PWM100%という想定外の波形になってしまいます。モータを制御していたら、一瞬ですが 100%の回転となってしまい、暴走してしまうかもしれません。

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)。



①	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。
②	「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。** **TRDGRB0 の値は、TRD0 の値が 0 になるの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になります。これがバッファレジスタを使う目的です。**

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	「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 になります。 **・TRDGRD0 の値が、TRDGRB0 に転送されます。** **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。

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、下記のようでした。

P2_2 端子が"1"の時間＝ (TRDGRB0＋1)×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。

P2_2 端子が"1"の時間＝ (0＋1)×400ns
＝ 1×400ns ＝ 400ns

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、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します。

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。

⑧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。

P2_4 端子から出力される PWM 波形の ON 幅＝タイマ RD カウンタ 0 のカウントソース×(TRDGRA1＋1)

TRDGRA1 を左辺に移動して、TRDGRA1 を求める式に変形します。

TRDGRA1＝P2_4 端子から出力される PWM 波形の ON 幅／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。

TRDGRA1＝ON 幅　／カウントソース－1
TRDGRA1＝$(10\times10^{-3})/(400\times10^{-9})-1$
TRDGRA1＝25000－1 ＝ 24999

■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。

⑨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。

P2_5 端子から出力される PWM 波形の ON 幅＝タイマ RD カウンタ 0 のカウントソース×(TRDGRB1＋1)

TRDGRB1 を左辺に移動して、TRDGRB1 を求める式に変形します。

TRDGRB1＝P2_5 端子から出力される PWM 波形の ON 幅／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。

TRDGRB1＝ON 幅　／カウントソース－1
TRDGRB1＝$(15\times10^{-3})/(400\times10^{-9})-1$
TRDGRB1＝37500－1 ＝ 37499

■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 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。
①、②で割り当てた端子を、出力許可にしてください。

設定 bit	上:ビット名 下:シンボル	内容	今回の内容
bit7	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			

⑪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**			

22.6.2 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、RY_R8C38 ボードのディップスイッチで設定します。

```
33 :  void main( void )
34 :  {
35 :      init();                                 /* 初期化                        */
36 :
37 :      while( 1 ) {
38 :          trdgrd0 = 39998 * dipsw_get() / 15;
39 :      }
40 :  }
```

RY_R8C38 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%

38 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するには、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%

22.7 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 ポートにしなさい。このとき、この端子は出力端子として"0"を出力しなさい。

(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 ポートにしなさい。このとき、この端子は出力端子として"0"を出力しなさい。
※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なさい。出力の仕方は、サンプルプログラム 38 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なさい。出力の仕方は、サンプルプログラム 38 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 ポートにしなさい。このとき、この端子は出力端子として"0"を出力しなさい。
※プログラムは、(4)の状態から改造するものとする。

23. 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)

23.1 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、今回の 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。

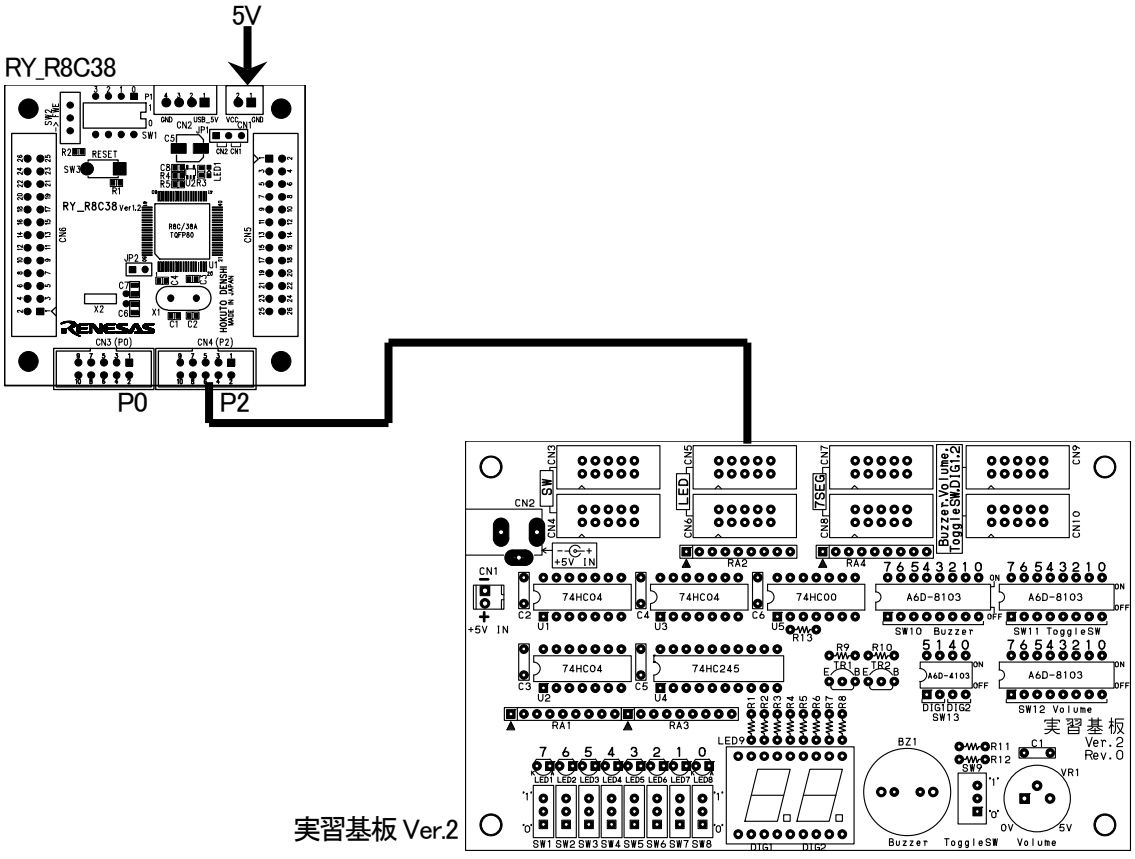
23.2 接続

■使用ポート

マイコンのポート	接続内容
P1_3、P1_2、P1_1、P1_0	RY_R8C38 ボードのディップスイッチです。
P2 (CN4)	実習基板 Ver.2 の LED 部など、出力機器を接続します。

■接続例

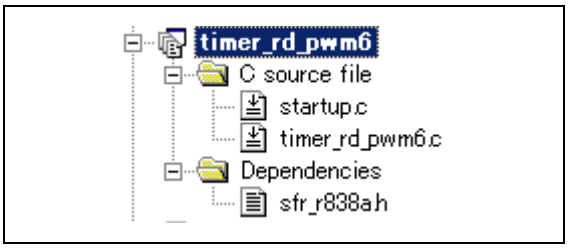
実習基板 Ver.2 を使ったときの接続例を下記に示します。



■操作方法

RY_R8C38 ボードのディップスイッチ(SW1)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。

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。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥timer_rd_pwm6¥startup.c
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/38A の内蔵周辺機能(SFR)の初期化も行います。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥timer_rd_pwm6¥timer_rd_pwm6.c
3	sfr_r838a.h	R8C/38A マイコンの内蔵周辺機能を制御するためのレジスタ（Special Function Registers）を定義したファイルです。 ファイルの位置→C:¥Workspace¥common_r8c38a¥sfr_r838a.h

23.4 プログラム「timer_rd_pwm6.c」

```
 1 :  /***********************************************************************/
 2 :  /* 対象マイコン R8C/38A                                                */
 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.1.00                                               */
 5 :  /* Date         2010.04.01                                             */
 6 :  /* Copyright    ジャパンマイコンカーラリー実行委員会                   */
 7 :  /***********************************************************************/
 8 :  /*
 9 :  入力：マイコンボードのディップスイッチ(4bit)
10 :  出力：P2_1,P2_2,P2_3,P2_5,P2_6,P2_7端子からPWM出力
11 :
12 :  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、
13 :  ポート2の6端子からPWM波形を出力します。出力値を確認するため
14 :  実習基板Ver.2のLED部を接続します。
15 :  */
16 :
17 :  /*====================================*/
18 :  /* インクルード                       */
19 :  /*====================================*/
20 :  #include "sfr_r838a.h"                 /* R8C/38A 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 :
 39 :      init();                             /* 初期化                       */
 40 :
 41 :      while( 1 ) {
 42 :          if( dipsw_get() == 0 ) {
 43 :              // trdgra0,trdgra1と同じにすると0%出力
 44 :              pwm = 39999;
 45 :          } else if( dipsw_get() == 15 ) {
 46 :              // trdgra0,trdgra1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/38A 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;                         /*                              */
 80 :      pd1 = 0xd0;                         /* 5:RXD0 4:TXD0 3-0:DIP SW     */
 81 :      pd2 = 0xff;                         /* 7-0:LED                      */
 82 :      pd3 = 0xff;                         /*                              */
 83 :      p4  = 0x20;                         /* P4_5のLED:初期は点灯         */
 84 :      pd4 = 0xb8;                         /* 7:XOUT 6:XIN 5:LED 2:VREF    */
 85 :      pd5 = 0xff;                         /*                              */
 86 :      pd6 = 0xff;                         /*                              */
 87 :      pd7 = 0xff;                         /*                              */
 88 :      pd8 = 0xff;                         /*                              */
 89 :      pd9 = 0x3f;                         /*                              */
 90 :      pur0 = 0x04;                        /* P1_3～P1_0のプルアップON     */
 91 :
 92 :      /* タイマRD PWMモードの設定(6本のPWM出力) */
 93 :      trdpsr0  = 0x68;                    /* TRDIOB0,C0,D0端子設定        */
 94 :      trdpsr1  = 0x54;                    /* TRDIOB1,C1,D1端子設定        */
 95 :      trdpmr   = 0x77;                    /* 各端子PWMﾓｰﾄﾞにするかしないか*/
 96 :      trdfcr   = 0x80;                    /* PWMモードに設定              */
 97 :      trdocr   = 0x00;                    /* 各端子初期出力設定           */
 98 :      trdcr0   = 0x23;                    /* TRD0ソースカウントの選択:f8  */
 99 :      trdcr1   = 0x23;                    /* TRD1ソースカウントの選択:f8  */
100 :      trdpocr0 = 0x00;                    /* B0～D0のアクティブレベル設定 */
101 :      trdpocr1 = 0x00;                    /* B1～D1のアクティブレベル設定 */
102 :      trdgra0  = 39999;                   /* TRD0の周期設定               */
103 :      trdgra1  = 39999;                   /* TRD1の周期設定               */
104 :      trdgrb0  = 0;                       /* P2_2のON幅設定               */
105 :      trdgrc0  = 0;                       /* P2_1のON幅設定               */
106 :      trdgrd0  = 0;                       /* P2_3のON幅設定               */
107 :      trdgrb1  = 0;                       /* P2_5のON幅設定               */
108 :      trdgrc1  = 0;                       /* P2_6のON幅設定               */
109 :      trdgrd1  = 0;                       /* P2_7のON幅設定               */
110 :      trdoer1  = 0x11;                    /* 各端子出力許可するかしないか */
111 :      trdstr   = 0x0f;                    /* TRD0,TRD1カウント開始        */
112 :  }
113 :
```

```
114 :  /************************************************************************/
115 :  /* ディップスイッチ値読み込み                                           */
116 :  /* 戻り値 スイッチ値 0～15                                               */
117 :  /************************************************************************/
118 :  unsigned char dipsw_get( void )
119 :  {
120 :      unsigned char sw;
121 :
122 :      sw = p1 & 0x0f;                     /* P1_3～P1_0読み込み           */
123 :
124 :      return  sw;
125 :  }
126 :
127 :  /************************************************************************/
128 :  /* end of file                                                          */
129 :  /************************************************************************/
```

23.5 プログラムの解説

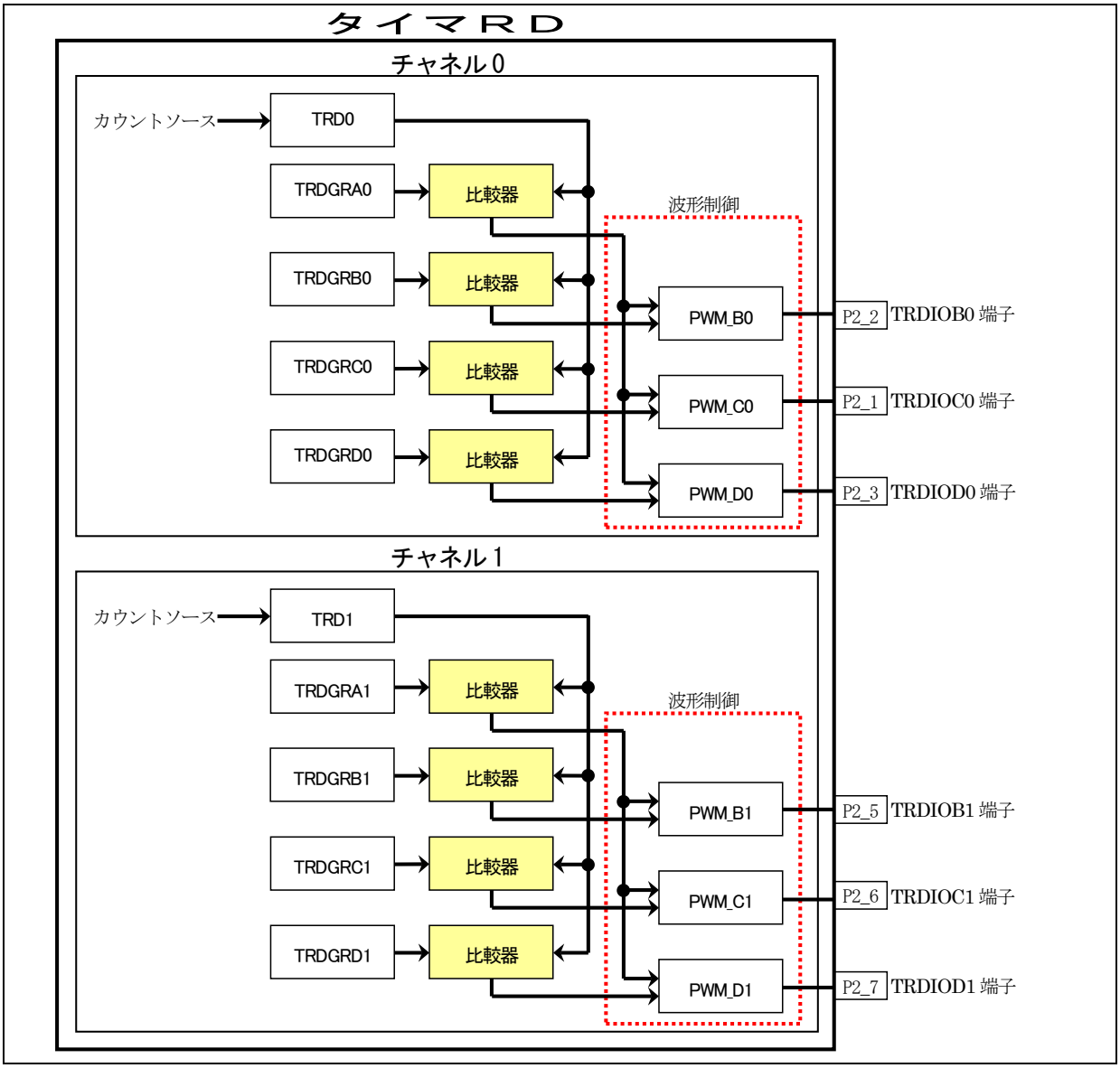
23.5.1 init 関数(タイマ RD の設定)

タイマ RD を使った PWM モードの設定を行います。

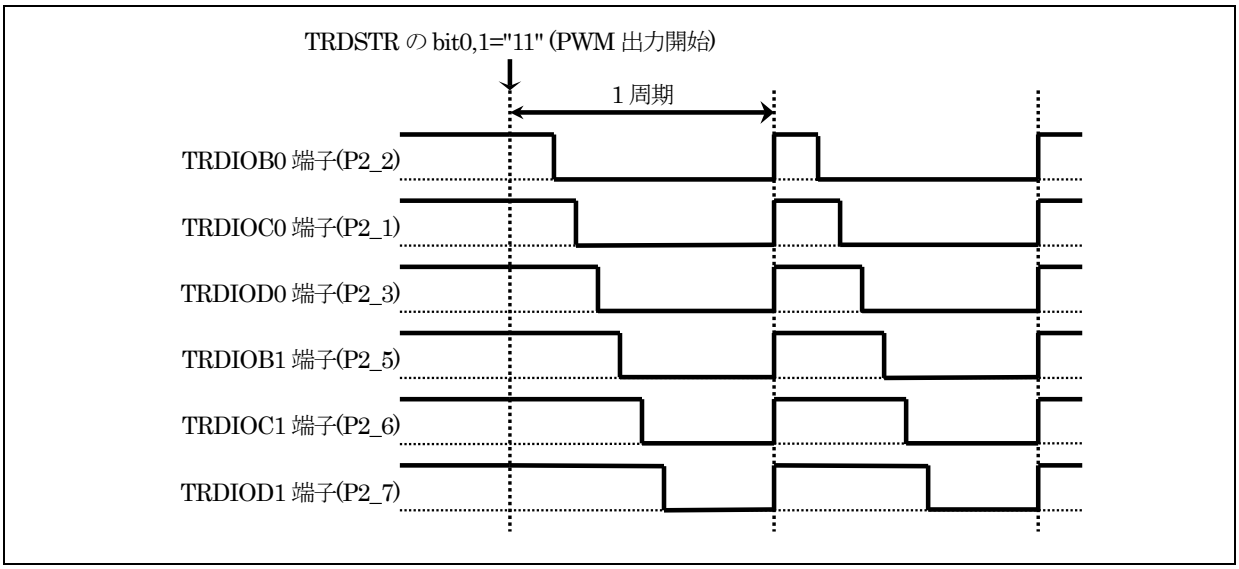
```
 92 :      /* タイマRD PWMモードの設定(6本のPWM出力) */
 93 :      trdpsr0  = 0x68;                    /* TRDIOB0,C0,D0端子設定        */
 94 :      trdpsr1  = 0x54;                    /* TRDIOB1,C1,D1端子設定        */
 95 :      trdpmr   = 0x77;                    /* 各端子PWMﾓｰﾄﾞにするかしないか*/
 96 :      trdfcr   = 0x80;                    /* PWMモードに設定              */
 97 :      trdocr   = 0x00;                    /* 各端子初期出力設定           */
 98 :      trdcr0   = 0x23;                    /* TRD0ソースカウントの選択:f8  */
 99 :      trdcr1   = 0x23;                    /* TRD1ソースカウントの選択:f8  */
100 :      trdpocr0 = 0x00;                    /* B0～D0のアクティブレベル設定 */
101 :      trdpocr1 = 0x00;                    /* B1～D1のアクティブレベル設定 */
102 :      trdgra0  = 39999;                   /* TRD0の周期設定               */
103 :      trdgra1  = 39999;                   /* TRD1の周期設定               */
104 :      trdgrb0  = 0;                       /* P2_2のON幅設定               */
105 :      trdgrc0  = 0;                       /* P2_1のON幅設定               */
106 :      trdgrd0  = 0;                       /* P2_3のON幅設定               */
107 :      trdgrb1  = 0;                       /* P2_5のON幅設定               */
108 :      trdgrc1  = 0;                       /* P2_6のON幅設定               */
109 :      trdgrd1  = 0;                       /* P2_7のON幅設定               */
110 :      trdoer1  = 0x11;                    /* 各端子出力許可するかしないか */
111 :      trdstr   = 0x0f;                    /* TRD0,TRD1カウント開始        */
```

(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。

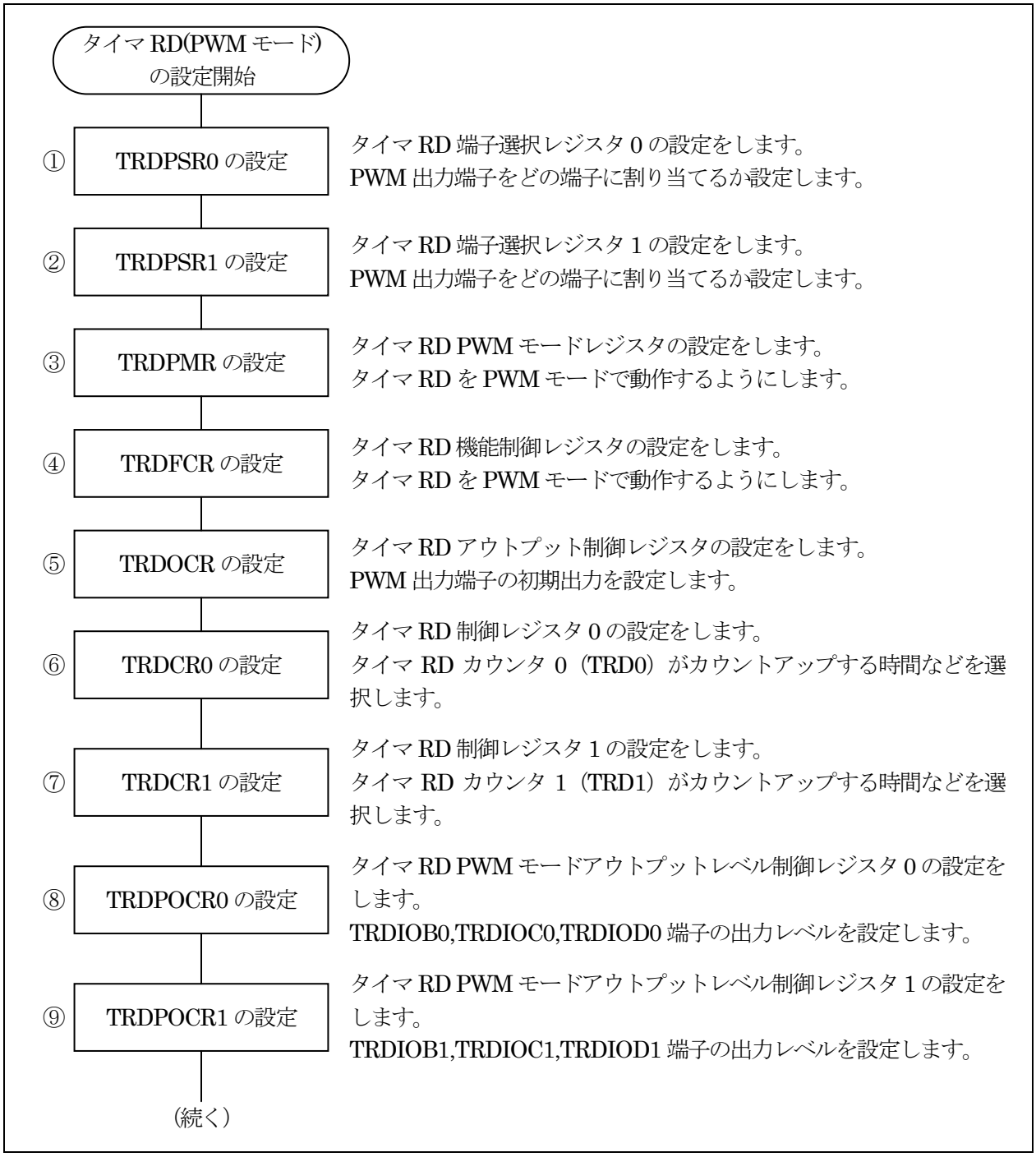


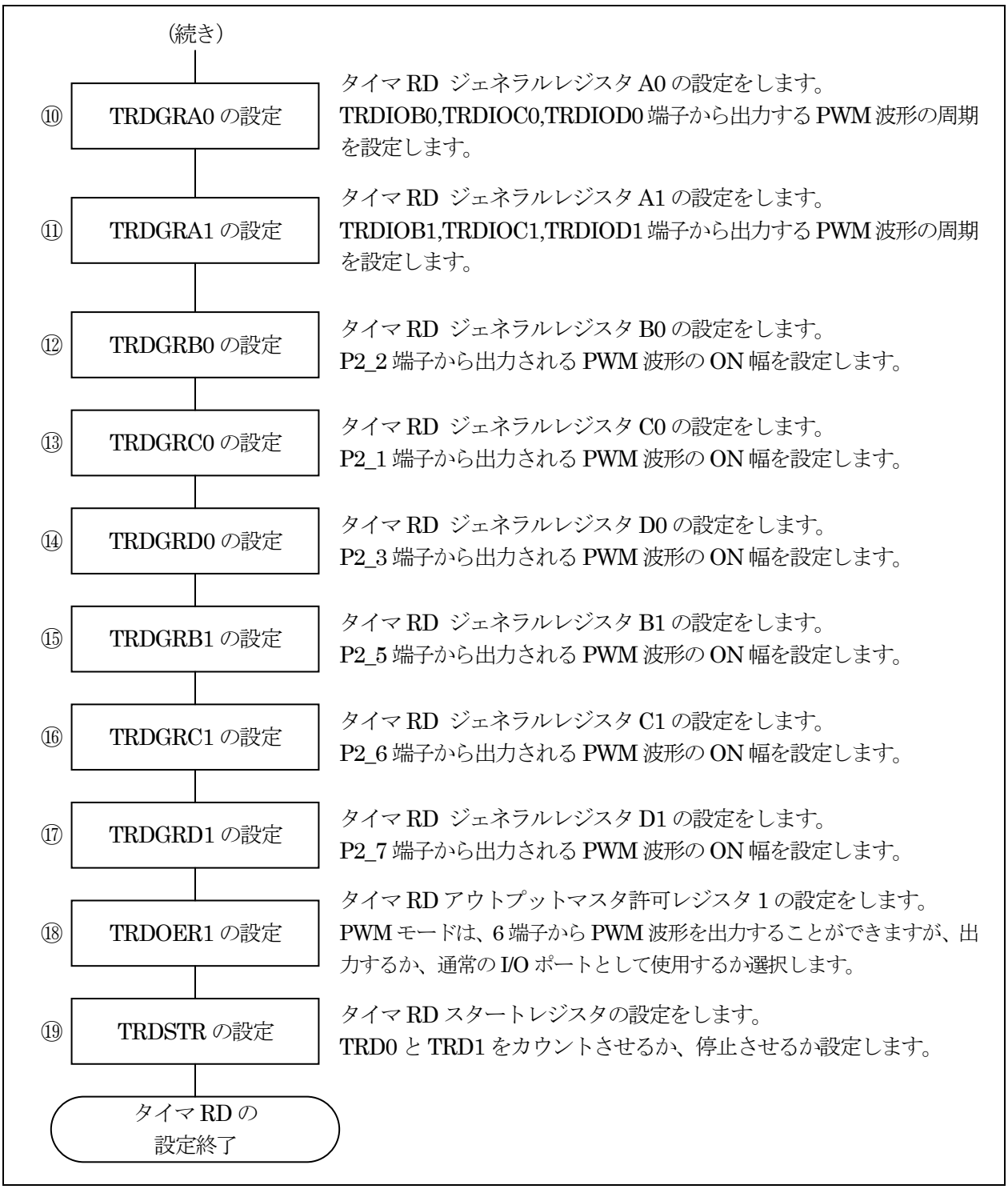
PWM 波形の出力例を下記に示します。



(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。




（続き）
⑩ TRDGRA0 の設定
タイマ RD ジェネラルレジスタ A0 の設定をします。
TRDIOB0,TRDIOC0,TRDIOD0 端子から出力する PWM 波形の周期を設定します。
⑪ TRDGRA1 の設定
タイマ RD ジェネラルレジスタ A1 の設定をします。
TRDIOB1,TRDIOC1,TRDIOD1 端子から出力する PWM 波形の周期を設定します。
⑫ TRDGRB0 の設定
タイマ RD ジェネラルレジスタ B0 の設定をします。
P2_2 端子から出力される PWM 波形の ON 幅を設定します。
⑬ TRDGRC0 の設定
タイマ RD ジェネラルレジスタ C0 の設定をします。
P2_1 端子から出力される PWM 波形の ON 幅を設定します。
⑭ TRDGRD0 の設定
タイマ RD ジェネラルレジスタ D0 の設定をします。
P2_3 端子から出力される PWM 波形の ON 幅を設定します。
⑮ TRDGRB1 の設定
タイマ RD ジェネラルレジスタ B1 の設定をします。
P2_5 端子から出力される PWM 波形の ON 幅を設定します。
⑯ TRDGRC1 の設定
タイマ RD ジェネラルレジスタ C1 の設定をします。
P2_6 端子から出力される PWM 波形の ON 幅を設定します。
⑰ TRDGRD1 の設定
タイマ RD ジェネラルレジスタ D1 の設定をします。
P2_7 端子から出力される PWM 波形の ON 幅を設定します。
⑱ TRDOER1 の設定
タイマ RD アウトプットマスタ許可レジスタ 1 の設定をします。
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選択します。
⑲ TRDSTR の設定
タイマ RD スタートレジスタの設定をします。
TRD0 と TRD1 をカウントさせるか、停止させるか設定します。
タイマ RD の
設定終了

①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/38A では、割り当てる端子は決まっており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 trdiod0sel0	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			

②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/38A では、割り当てる端子は決まっており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 trdiod1sel0	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			

③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			

④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_trdfcr	″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_trdfcr	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_trdfcr bit0:cmd0_trdfcr	PWM モードでは“00”（タイマモード、PWM モード、PWM3 モード）にしてください	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			

⑤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_trdocr	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_trdocr	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_trdocr	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_trdocr	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_trdocr	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_trdocr	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_trdocr	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_trdocr	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	O				O			

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	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			

⑦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			

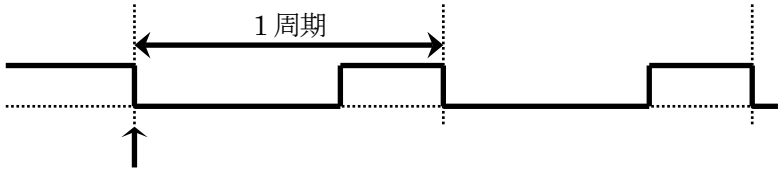
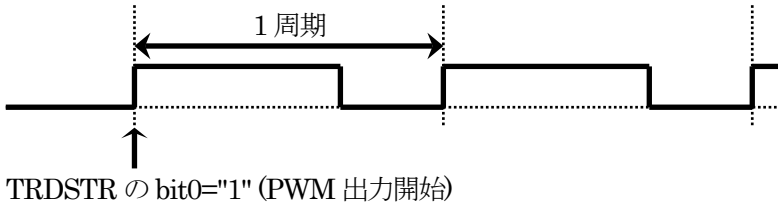
⑧タイマ RD PWM モードアウトプットレベル制御レジスタ 0
(TRDPOCR0： Timer RD PWM mode output level control register 0)の設定

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	出力波形 	″0″からスタートする設定です。
“L”アクティブ	出力波形 	″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	O				O			

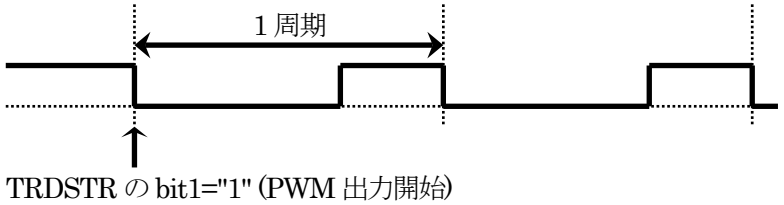
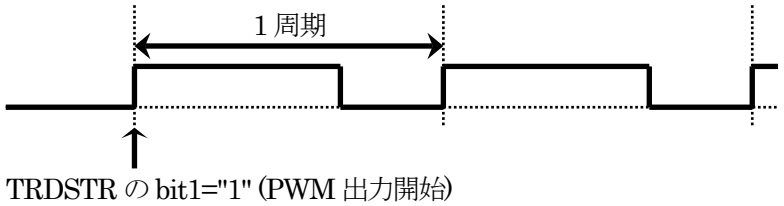
⑨タイマ RD PWM モードアウトプットレベル制御レジスタ 1
(TRDPOCR1： Timer RD PWM mode output level control register 1)の設定

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:TRDIOC1 の出力レベルは“L”アクティブ 1:TRDIOC1 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:TRDIOB1 の出力レベルは“L”アクティブ 1:TRDIOB1 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	出力波形 	″0″からスタートする設定です。
“L”アクティブ	出力波形 	″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	O				O			

⑩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。

PWM 周期＝タイマ RD カウンタ 0 のカウントソース×(TRDGRA0＋1)

TRDGRA0 を左辺に移動して、TRDGRA0 を求める式に変形します。

TRDGRA0＝PWM 周期／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。

TRDGRA0＝PWM 周期／カウントソース－1
TRDGRA0＝$(16\times10^{-3})/(400\times10^{-9})-1$
TRDGRA0＝40000－1 ＝ 39999

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。

⑪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。

PWM 周期＝タイマ RD カウンタ 1 のカウントソース×(TRDGRA1＋1)

TRDGRA1 を左辺に移動して、TRDGRA1 を求める式に変形します。

TRDGRA1＝PWM 周期／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。

TRDGRA1＝PWM 周期／カウントソース－1
TRDGRA1＝$(16\times10^{-3})/(400\times10^{-9})-1$
TRDGRA1＝40000－1 ＝ 39999

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。

⑫タイマ RD ジェネラルレジスタ B0(TRDGRB0：Timer RD General register B0)の設定
⑬タイマ RD ジェネラルレジスタ C0(TRDGRC0：Timer RD General register C0)の設定
⑭タイマ RD ジェネラルレジスタ D0(TRDGRD0：Timer RD General register D0)の設定
⑮タイマ RD ジェネラルレジスタ B1(TRDGRB1：Timer RD General register B1)の設定
⑯タイマ RD ジェネラルレジスタ C1(TRDGRC1：Timer RD General register C1)の設定
⑰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
⑫	タイマ RD ジェネラルレジスタ B0	TRDGRB0	P2_2	TRDGRB0=ON 幅×タイマ RD カウンタ 0 のカウントソース-1
⑬	タイマ RD ジェネラルレジスタ C0	TRDGRC0	P2_1	TRDGRC0=ON 幅×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 D0	TRDGRD0	P2_3	TRDGRD0=ON 幅×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 B1	TRDGRB1	P2_5	TRDGRB1=ON 幅×タイマ RD カウンタ 1 のカウントソース-1
⑯	タイマ RD ジェネラルレジスタ C1	TRDGRC1	P2_6	TRDGRC1=ON 幅×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 D1	TRDGRD1	P2_7	TRDGRD1=ON 幅×タイマ RD カウンタ 1 のカウントソース-1

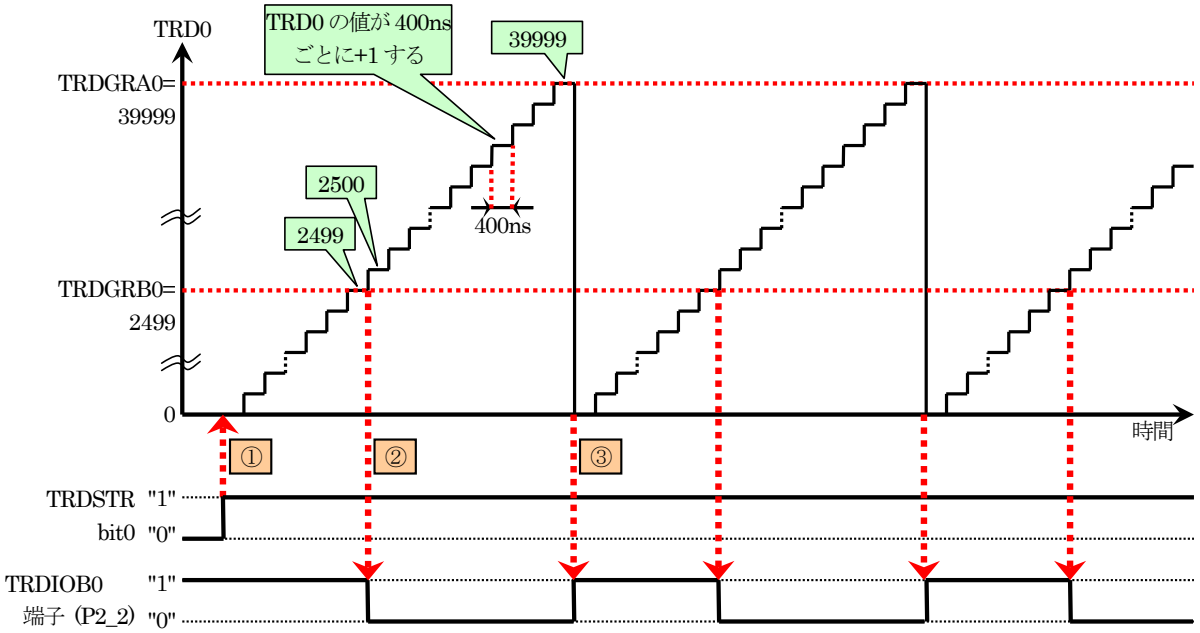
例として、タイマ RD ジェネラルレジスタ B0(TRDGRB0)を計算してみます。TRDGRB0 を設定することにより、P2_2 端子から出力される PWM 波形の ON 幅を設定します。
P2_2 端子から出力される PWM 波形の ON 幅は、下記の式で決まります。

TRDGRB0＝P2_2 端子から出力される PWM 波形の ON 幅／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 幅　／カウントソース－1
TRDGRB0＝$(1\times10^{-3})/(400\times10^{-9})-1$
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	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]
③	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] 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。

⑱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

①、②でタイマ RD の端子として割り当てた場合は出力許可、使用しない場合は出力禁止にしてください。

設定 bit	上：ビット名 下：シンボル	内容	今回の内容
bit7	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**			

⑲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			

23.5.2 main 関数

RY_R8C38 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,trdgra1と同じにすると0%出力
 44 :              pwm = 39999;
 45 :          } else if( dipsw_get() == 15 ) {
 46 :              // trdgra0,trdgra1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 :  }
```

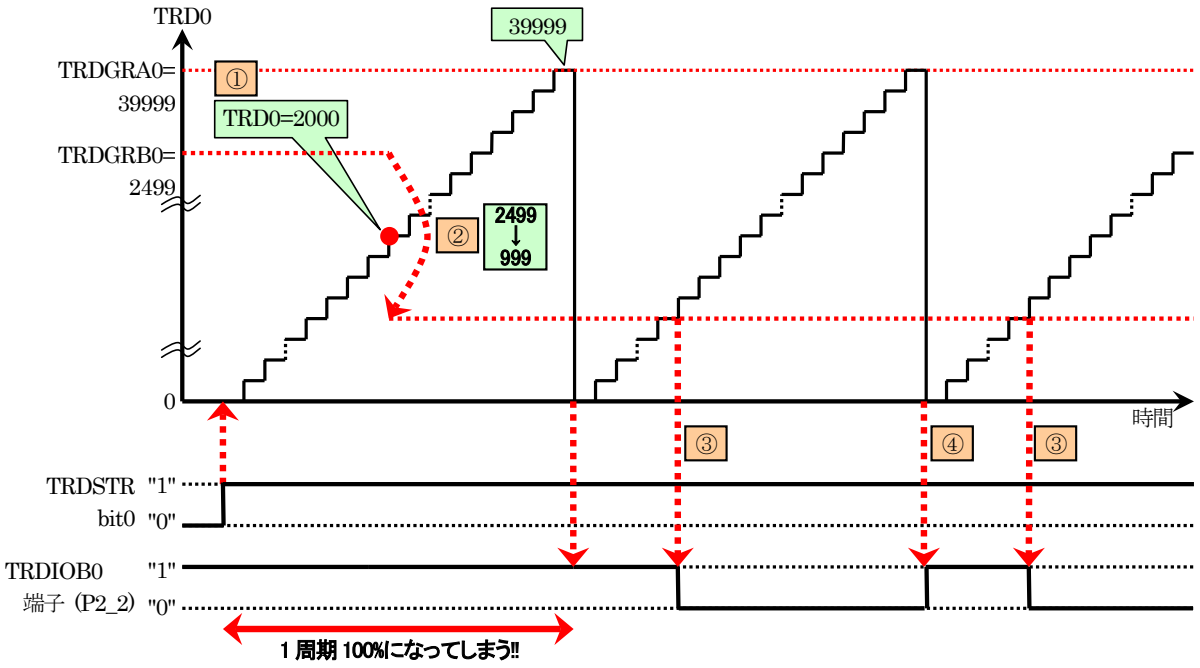
42 行	ディップスイッチの値が 0 かどうかチェックします。0 なら、44 行を実行します。
44 行	PWM を 0%出力するために、pwm 変数に TRDGRA0（TRDGRA1）と同じ値の 39999 を設定します。周期を替えた場合は、TRDGRA0（TRDGRA1）の値は 39999 では無くなりますので、TRDGRA0（TRDGRA1）と同じ値を設定してください。 詳しくは、「**22. タイマ RD による PWM 波形出力（リセット同期 PWM モード）（プロジェクト：timer_rd_doukipwm）**」を参照してください。
45 行	ディップスイッチの値が 15 かどうかチェックします。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、「**22. 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	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。
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い、暴走してしまうかもしれません。

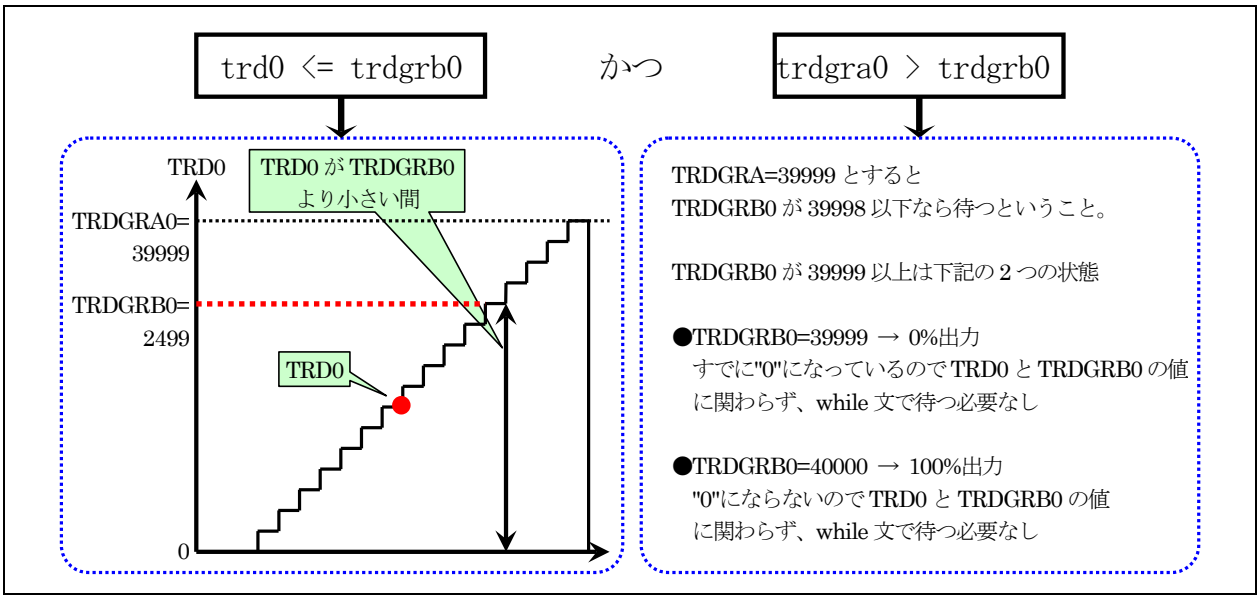
リセット同期 PWM モードの場合、バッファレジスタがありましたが、PWM モードにはバッファレジスタの機能がありません。そのため、プログラムでこのような不具合が起こらないようにします。

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。



23.6 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(P2_5)=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。

24. モータの制御（プロジェクト：motor）

24.1 概要

本章では、マイコンカー用のモータドライブ基板 Ver.5 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。

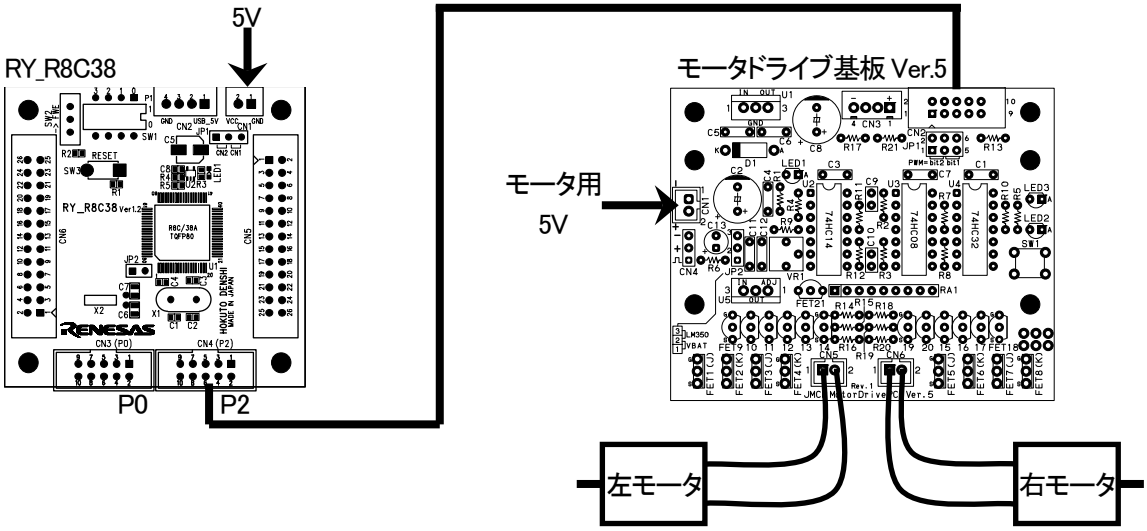
24.2 接続

■使用ポート

マイコンのポート	接続内容
P2 (CN4)	マイコンカー用モータドライブ基板 Ver.5 を接続

■接続例

RY_R8C38 ボードとモータドライブ基板 Ver.5 をフラットケーブルで接続します。

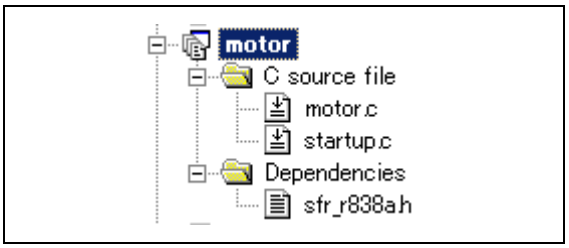


※モータドライブ基板 Ver.4 も接続可能です。プログラムも同一のもので動作します。

■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。モータが回りますので、モータ付属のギヤボックス、タイヤなどは浮かした状態で実験してください。

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。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥motor¥startup.c
2	motor.c	実際に制御するプログラムが書かれています。R8C/38A の内蔵周辺機能(SFR)の初期化も行います。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥motor¥motor.c
3	sfr_r838a.h	R8C/38A マイコンの内蔵周辺機能を制御するためのレジスタ（Special Function Registers)を定義したファイルです。 ファイルの位置→C:¥Workspace¥common_r8c38a¥sfr_r838a.h

24.4 プログラム「motor.c」

```
 1 :  /***********************************************************************/
 2 :  /* 対象マイコン R8C/38A                                                */
 3 :  /* ﾌｧｲﾙ内容     マイコンカー用モータドライブ基板Ver.5のモータ制御      */
 4 :  /* バージョン   Ver.1.02                                               */
 5 :  /* Date         2012.10.18                                             */
 6 :  /* Copyright    ジャパンマイコンカーラリー実行委員会                   */
 7 :  /***********************************************************************/
 8 :  /*
 9 :  入力：マイコンボード上のディップスイッチ
10 :  出力：マイコンカー用モータドライブ基板Ver.5の右モータ、左モータ
11 :
12 :  マイコンカー用モータドライブ基板Ver.5の右モータ、左モータを制御します。
13 :  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。
14 :  */
15 :
16 :  /*=====================================*/
17 :  /* インクルード                        */
18 :  /*=====================================*/
19 :  #include "sfr_r838a.h"                 /* R8C/38A 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 :
26 :  /*=====================================*/
27 :  /* プロトタイプ宣言                    */
28 :  /*=====================================*/
29 :  void init( void );
30 :  void timer( unsigned long timer_set );
31 :  void motor( int data1, int data2 );
32 :  unsigned char dipsw_get( void );
33 :
```

```
 34 :  /*====================================*/
 35 :  /* グローバル変数の宣言               */
 36 :  /*====================================*/
 37 :  unsigned long cnt_rb;                  /* タイマRB用                 */
 38 :
 39 :  /************************************************************************/
 40 :  /* メインプログラム                                                     */
 41 :  /************************************************************************/
 42 :  void main( void )
 43 :  {
 44 :      init();                          /* 初期化                         */
 45 :      asm(" fset I ");                 /* 全体の割り込み許可             */
 46 :
 47 :      while( 1 ) {
 48 :          motor( 100 , 0);
 49 :          timer( 1000 );
 50 :          motor( 0, 80 );
 51 :          timer( 1000 );
 52 :          motor( -60, 0 );
 53 :          timer( 1000 );
 54 :          motor( 0, -40 );
 55 :          timer( 1000 );
 56 :          motor( 0, 0 );
 57 :          timer( 1000 );
 58 :      }
 59 :  }
 60 :
 61 :  /************************************************************************/
 62 :  /* R8C/38A スペシャルファンクションレジスタ(SFR)の初期化                */
 63 :  /************************************************************************/
 64 :  void init( void )
 65 :  {
 66 :      int i;
 67 :
 68 :      /* クロックをXINクロック(20MHz)に変更 */
 69 :      prc0  = 1;                       /* プロテクト解除                 */
 70 :      cm13  = 1;                       /* P4_6,P4_7をXIN-XOUT端子にする*/
 71 :      cm05  = 0;                       /* XINクロック発振                */
 72 :      for(i=0; i<50; i++ );            /* 安定するまで少し待つ(約10ms) */
 73 :      ocd2  = 0;                       /* システムクロックをXINにする  */
 74 :      prc0  = 0;                       /* プロテクトON                   */
 75 :
 76 :      /* ポートの入出力設定 */
 77 :      prc2 = 1;                        /* PD0のプロテクト解除            */
 78 :      pd0 = 0x00;                      /*                                */
 79 :      pd1 = 0xd0;                      /* 5:RXD0 4:TXD0 3-0:DIP SW       */
 80 :      pd2 = 0xfe;                      /* 7-0:モータドライブ基板Ver.5  */
 81 :      pd3 = 0xff;                      /*                                */
 82 :      p4  = 0x20;                      /* P4_5のLED:初期は点灯           */
 83 :      pd4 = 0xb8;                      /* 7:XOUT 6:XIN 5:LED 2:VREF      */
 84 :      pd5 = 0xff;                      /*                                */
 85 :      pd6 = 0xff;                      /*                                */
 86 :      pd7 = 0xff;                      /*                                */
 87 :      pd8 = 0xff;                      /*                                */
 88 :      pd9 = 0x3f;                      /*                                */
 89 :      pur0 = 0x04;                     /* P1_3～P1_0のプルアップON     */
 90 :
 91 :      /* タイマRBの設定 */
 92 :      /* 割り込み周期 = 1 / 20[MHz]   * (TRBPRE+1) * (TRBPR+1)
 93 :                      = 1 / (20*10^6) * 200        * 100
 94 :                      = 0.001[s] = 1[ms]
 95 :      */
 96 :      trbmr  = 0x00;                   /* 動作モード、分周比設定         */
 97 :      trbpre = 200-1;                  /* プリスケーラレジスタ           */
 98 :      trbpr  = 100-1;                  /* プライマリレジスタ             */
 99 :      trbic  = 0x07;                   /* 割り込み優先レベル設定         */
100 :      trbcr  = 0x01;                   /* カウント開始                   */
101 :
102 :      /* タイマRD リセット同期PWMモードの設定*/
103 :      /* PWM周期 = 1 / 20[MHz]   * カウントソース * (TRDGRA0+1)
104 :                 = 1 / (20*10^6) * 8              * 40000
105 :                 = 0.016[s] = 16[ms]
106 :      */
107 :      trdpsr0 = 0x08;                  /* TRDIOB0,C0,D0端子設定          */
108 :      trdpsr1 = 0x05;                  /* TRDIOA1,B1,C1,D1端子設定       */
109 :      trdmr   = 0xf0;                  /* バッファレジスタ設定           */
110 :      trdfcr  = 0x01;                  /* リセット同期PWMモードに設定  */
111 :      trdcr0  = 0x23;                  /* ソースカウントの選択: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 = 3750;        /* P2_5端子のON幅設定             */
116 :      trdoer1 = 0xcd;                  /* 出力端子の選択                 */
117 :      trdstr  = 0x0d;                  /* TRD0カウント開始               */
118 :  }
119 :
```

```
120 :  /************************************************************************/
121 :  /* ディップスイッチ値読み込み                                           */
122 :  /* 戻り値 スイッチ値 0～15                                              */
123 :  /************************************************************************/
124 :  unsigned char dipsw_get( void )
125 :  {
126 :      unsigned char sw;
127 :
128 :      sw = p1 & 0x0f;                     /* P1_3～P1_0読み込み           */
129 :
130 :      return  sw;
131 :  }
132 :
133 :  /************************************************************************/
134 :  /* タイマ本体                                                           */
135 :  /* 引数    タイマ値 1=1ms                                               */
136 :  /************************************************************************/
137 :  void timer( unsigned long timer_set )
138 :  {
139 :      cnt_rb = 0;
140 :      while( cnt_rb < timer_set );
141 :  }
142 :
143 :  /************************************************************************/
144 :  /* タイマRB 割り込み処理                                                */
145 :  /************************************************************************/
146 :  #pragma interrupt intTRB(vect=24)
147 :  void intTRB( void )
148 :  {
149 :      cnt_rb++;
150 :  }
151 :
152 :  /************************************************************************/
153 :  /* モータ速度制御                                                       */
154 :  /* 引数    左モータ:-100～100、右モータ:-100～100                       */
155 :  /*         0で停止、100で正転100%、-100で逆転100%                       */
156 :  /* 戻り値 なし                                                          */
157 :  /************************************************************************/
158 :  void motor( int data1, int data2 )
159 :  {
160 :      int    motor_r, motor_l, sw_data;
161 :
162 :      sw_data = dipsw_get() + 5;
163 :      motor_l = data1 * sw_data / 20;
164 :      motor_r = data2 * sw_data / 20;
165 :
166 :      /* 左モータ制御 */
167 :      if( motor_l >= 0 ) {
168 :          p2 &= 0xfd;
169 :          trdgrd0 = (long)( PWM_CYCLE - 1 ) * motor_l / 100;
170 :      } else {
171 :          p2 |= 0x02;
172 :          trdgrd0 = (long)( PWM_CYCLE - 1 ) * ( -motor_l ) / 100;
173 :      }
174 :
175 :      /* 右モータ制御 */
176 :      if( motor_r >= 0 ) {
177 :          p2 &= 0xf7;
178 :          trdgrc1 = (long)( PWM_CYCLE - 1 ) * motor_r / 100;
179 :      } else {
180 :          p2 |= 0x08;
181 :          trdgrc1 = (long)( PWM_CYCLE - 1 ) * ( -motor_r ) / 100;
182 :      }
183 :  }
184 :
185 :  /************************************************************************/
186 :  /* end of file                                                          */
187 :  /************************************************************************/
```

24.5 モータの制御

24.5.1 モータドライブ基板の役割

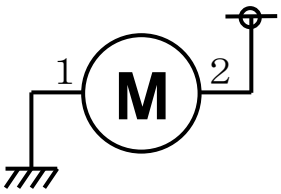
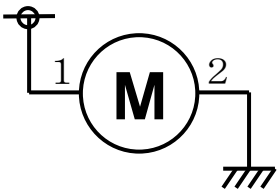
モータドライブ基板は、マイコンからの命令によってモータを動かします。マイコンからの「モータを回せ、止めろ」という信号は非常に弱く、その信号線に直接モータをつないでもモータはまったく動きません。この弱い信号をモータが動くための数百～数千 mA という大きな電流が流せる信号に変換します。

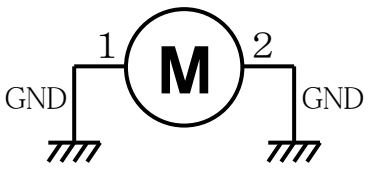
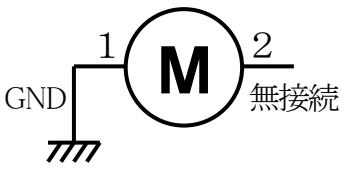
信号が弱いため、モータは回らない　　大電流に変換、モータ駆動！

モータのスピード制御については「PWM」を参照してください。

24.5.2 正転、逆転、ブレーキ、フリーの原理

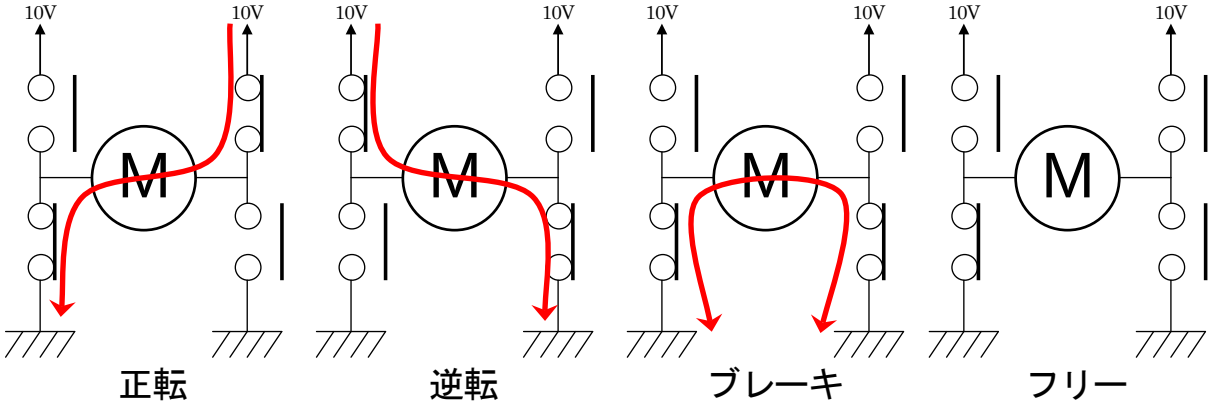
モータドライブ基板 Ver.5 は、モータを「正転、逆転、ブレーキ、フリー」に動作させることができます。これは、モータの端子に加える電圧を変えて制御しています。動作と端子の関係を下表に示します。

正転	逆転
	
1 ピンは GND（0V）、2 ピンはプラスに接続します。	1 ピンはプラス、2 ピンは GND（0V）に接続します。

ブレーキ	フリー
	
両端子を GND に接続します。結果、両端子がショートした状態になります。	片側を無接続にします。結果、どこにも繋いでいない状態になります。

24.5.3 H ブリッジ回路

実際は下図のように、モータを中心として H 型に4つのスイッチを付けます。この 4 つのスイッチをそれぞれ ON／OFF することにより、正転、逆転、ブレーキ、フリー制御を行います。H 型をしていることから「H ブリッジ回路」と呼ばれています。

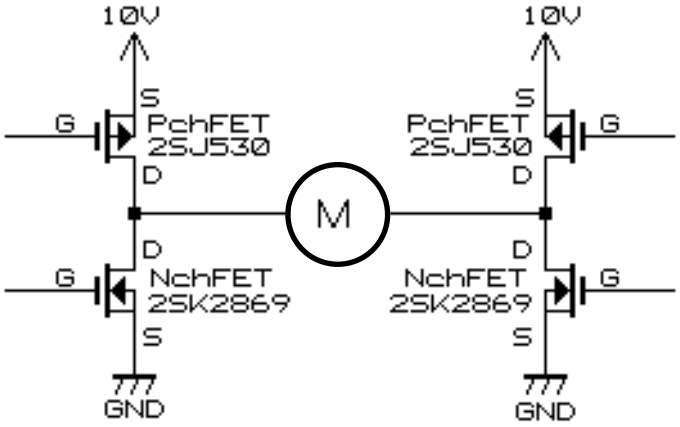


24.5.4 H ブリッジ回路のスイッチをFETにする

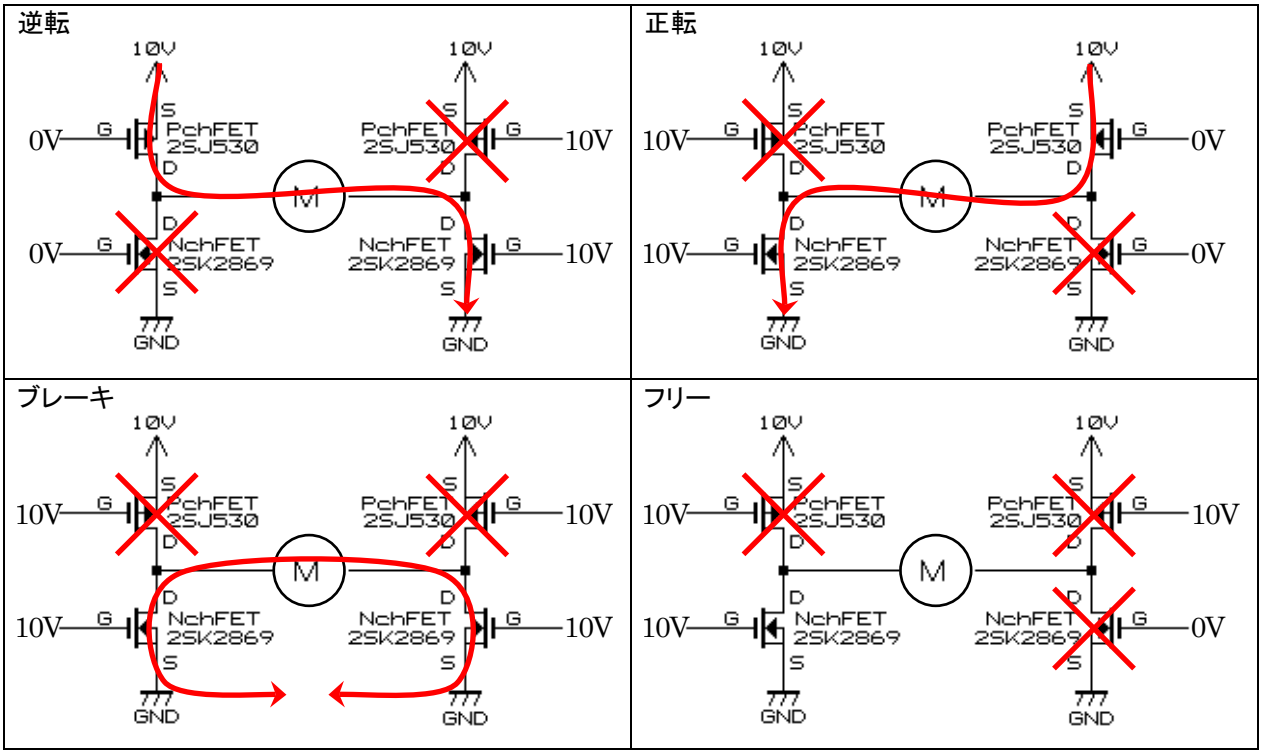
実際の回路では、前記のスイッチに変わる部分に FET を使います。電源のプラス側に P チャネル FET（2SJ タイプ）、マイナス側に N チャネル FET（2SK タイプ）を使用します。

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

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

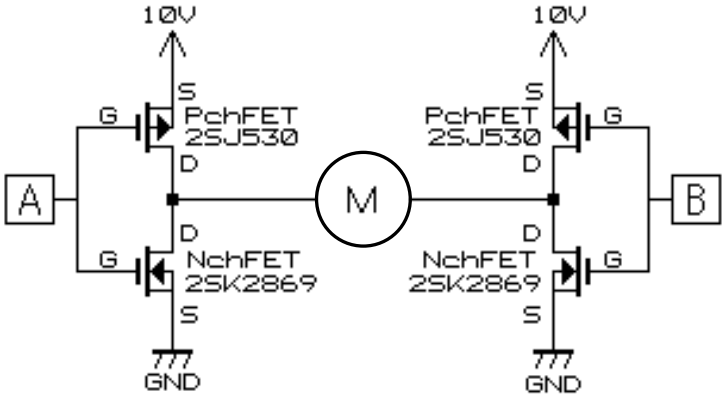


これら 4 つの FET のゲートに加える電圧を変えることにより、正転、逆転、ブレーキの動作を行います。



注意点は、絶対に左側 2 個もしくは右側 2 個の FET を同時に ON させてはいけないということです。10V から GND へ何の負荷もないまま繋がりますのでショートと同じです。FET が燃えるかパターンが燃えるか…いずれにせよ危険です。

4 つのゲート電圧を見ると、左側の P チャネル FET と N チャネル FET、右側の P チャネル FET と N チャネル FET に加える電圧が共通であることが分かります。そのため、下記のような回路にしてみました。



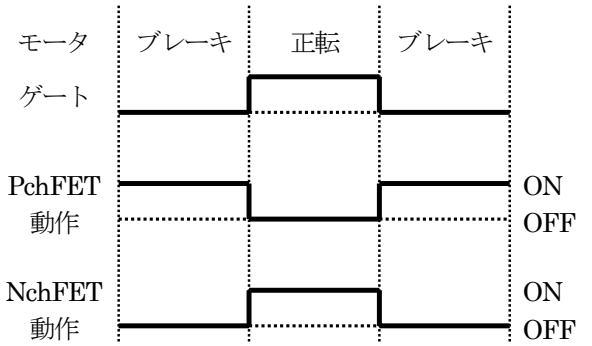
A	B	動作
0V	0V	ブレーキ
0V	10V	逆転
10V	0V	正転
10V	10V	ブレーキ

※G（ゲート）端子にはモータ用の電源電圧が 10V であったとすれば、その電圧がそのまま加えられたり 0V が加えられたりします。″0″、″1″の制御信号とは異なるので注意しましょう。

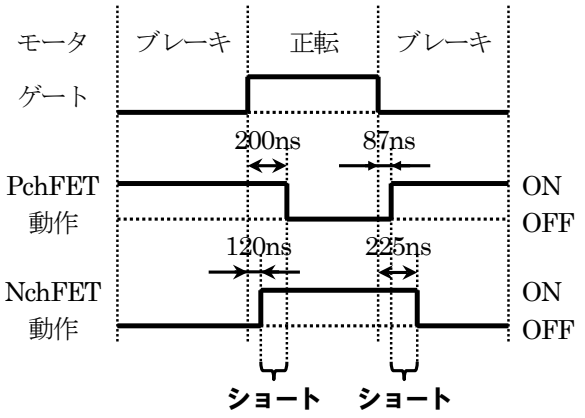
この回路を実際に組んで PWM 波形を加え動作させると、FET が非常に熱くなりました。どうしてでしょうか。

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

理想的な波形



実際の波形



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

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

モータドライブ基板 Ver.5 で使用しているルネサス エレクトロニクス製の FET「2SJ530」と「2SK2869」の電気的特性を下記に示します。

2SJ530(P チャネル)

電気的特性

(Ta=25℃)

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

注) 4. パルス測定

OFF→ON は 87ns 遅れる

ON→OFF は 200ns 遅れる

2SK2869(Nチャネル)

電気的特性

(Ta=25℃)

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

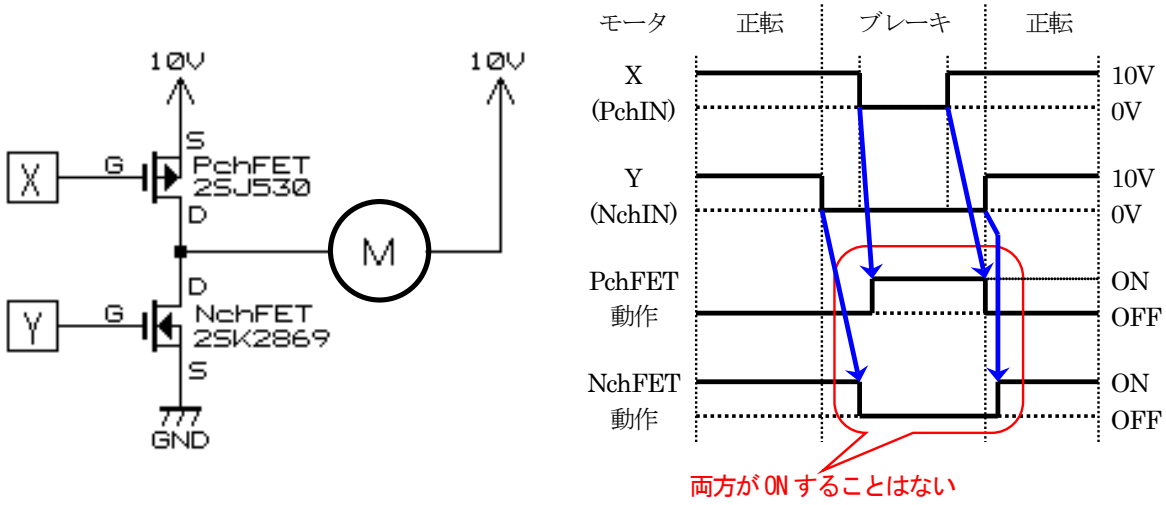
注) 1. パルス測定

OFF→ON は 120ns 遅れる

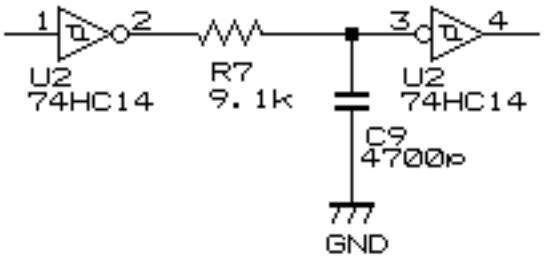
ON→OFF は 225ns 遅れる

24.5.5 PチャネルFETとNチャネルFETの短絡防止回路

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



この時間をずらす部分を、積分回路で作ります。積分回路については、多数の専門書があるので、そちらを参照してください。下記に積分回路を示します。



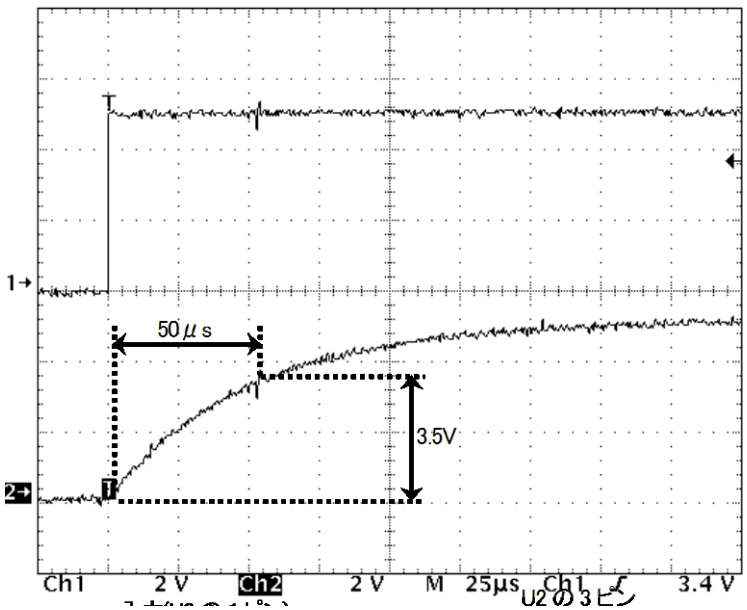
遅延時間は下記の式で計算することができます。

時定数T＝CR［s］

今回は 9.1kΩ、4700pF なので、計算すると下記の時間になります。

$T = (9.1 \times 10^{3}) \times (4700 \times 10^{-12})$

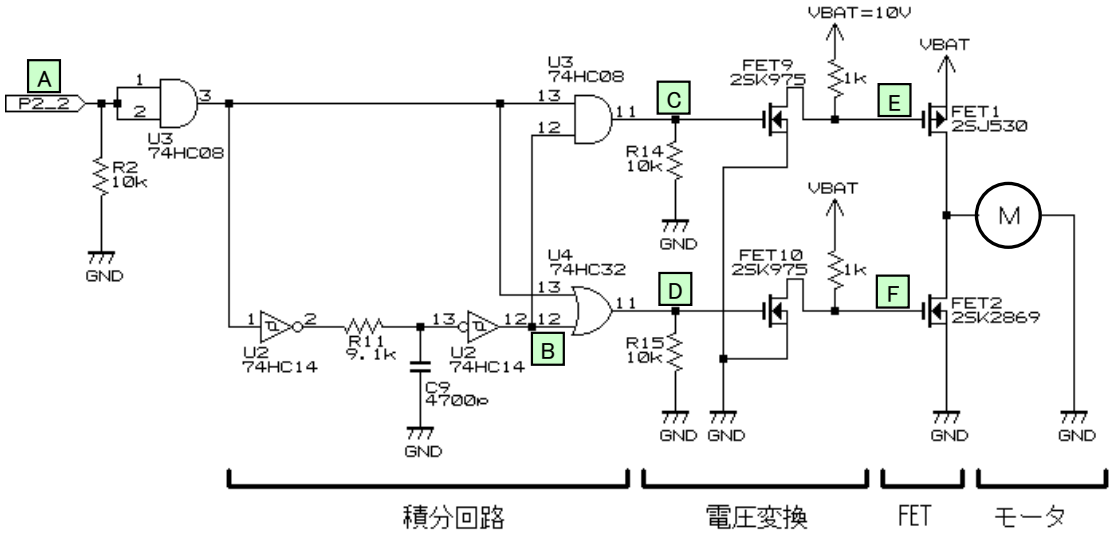
$= 42.77[\mu s]$

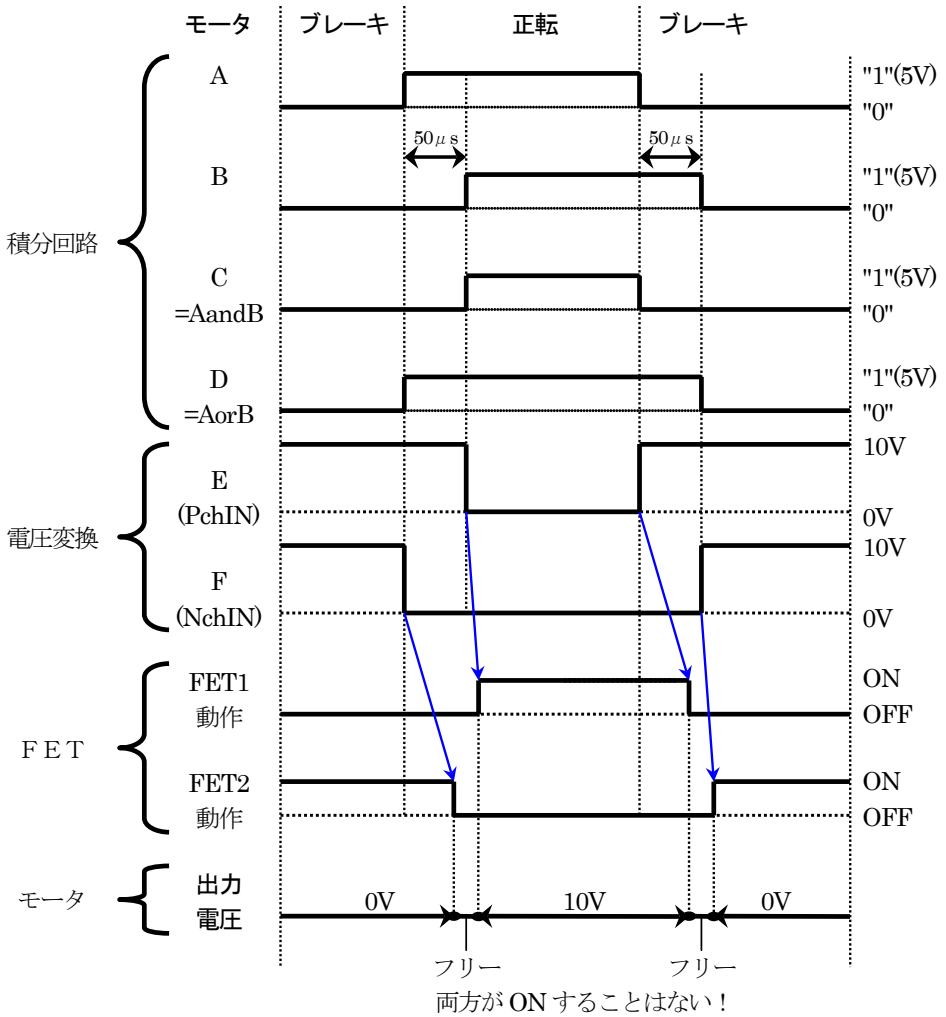


74HC シリーズは 3.5V 以上の入力電圧があると"1"とみなします。
実際に波形を観測し、3.5V になるまでの時間を計ると約 50μs になりました。

先ほどの「実際の波形」の図では最高でも 225ns のずれしかありませんが、積分回路では 50μs の遅延時間を作っています。これは、FET 以外にも、電圧変換用のデジタルトランジスタの遅延時間、FET のゲートのコンデンサ成分による遅れなどを含めたためです。

積分回路と FET を合わせた回路は下記のようになります。





(1) ブレーキ→正転に変えるとき

1. A点の信号は"0"でブレーキ、"1"で正転です。A点の出力を"0"(ブレーキ)から"1"(正転)へ変えます。
2. B点は積分回路により、50μs 遅れた波形が出力されます。
3. C点は、A and B の波形が出力されます。
4. D点は、A or B の波形が出力されます。
5. E点は、FET9 で電圧変換された信号が出力されます。C点の 0V－5V 信号が、10V－0V 信号へと変換されます。
6. F点も同様にD点の 0V－5V 信号が、10V－0V 信号へと変換されます。
7. A点の信号を"0"→"1"にかえると、FET2 のゲートが 10V→0V となり FET2 は OFF になります。ただし、遅延時間があるため遅れて OFF になります。この時点では、FET1 も FET2 も OFF 状態のため、モータはフリー状態となります。
8. A点の信号を変えてから 50μs 後、今度は FET1 のゲートが 0V→10V となり ON します。10V がモータに加えられ正転します。

(2) 正転→ブレーキに変えるとき

1. A点の信号を"1"(正転)から"0"(ブレーキ)にかえると、FET1 のゲート電圧が 0V から VBAT となり FET1 は OFF になります。ただし、遅延時間があるため遅れて OFF になります。この時点では、FET1 も FET2 も OFF 状態のため、モータはフリー状態となります。
2. A点の信号を変えてから 50μs 後、今度は FET2 のゲートが 0V→10V となり ON します。0V がモータに加えられ、両端子 0V なのでブレーキ動作になります。

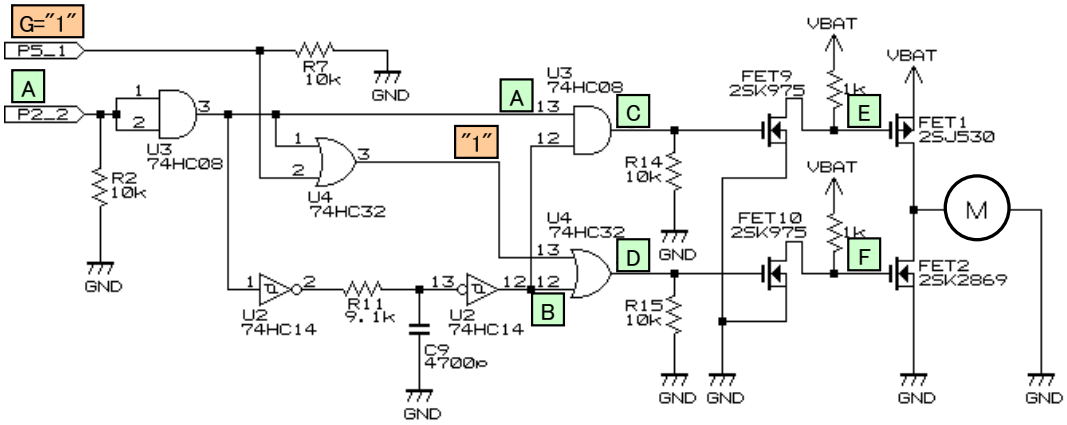
このように、動作を切り替えるときはいったん、両 FET ともフリー状態を作って、短絡するのを防いでいます。

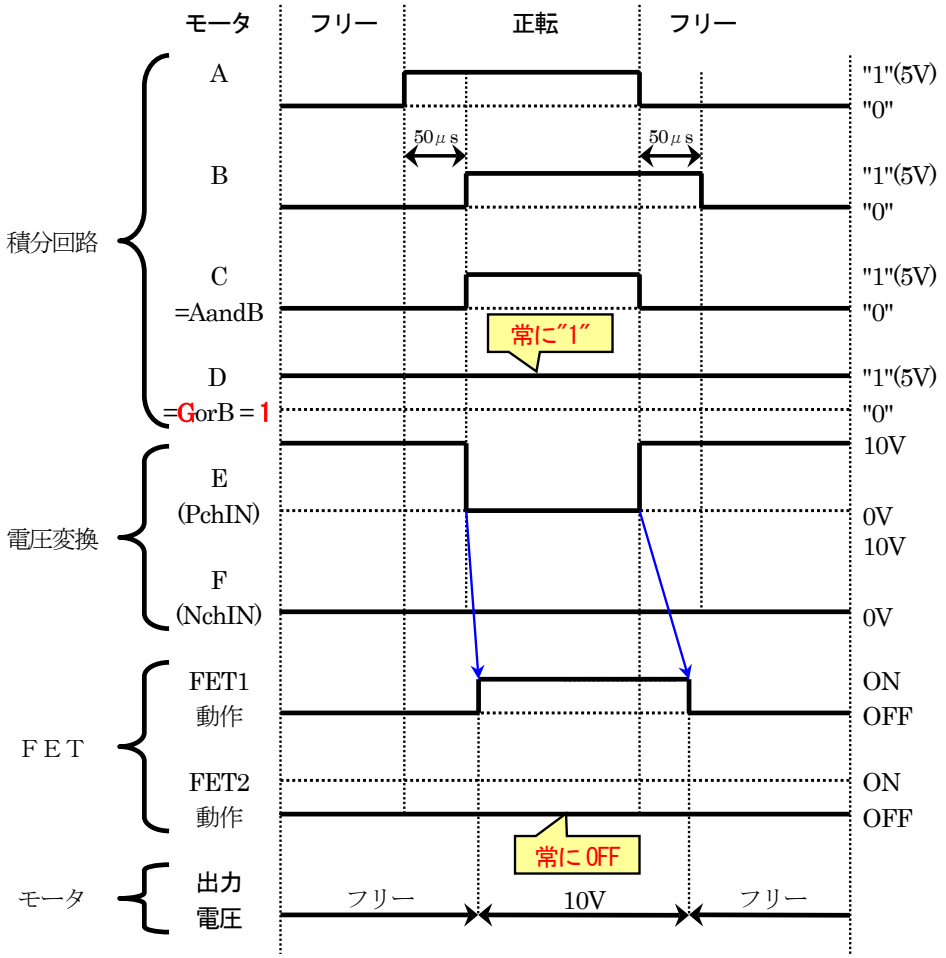
※ゲートに加える電圧の 10V は例です。実際は電源電圧(VBAT)になります。

24.5.6 フリー回路

ここでいうフリー回路は、P チャネル FET と N チャネル FET の短絡防止ではなく、モータの停止状態をブレーキとフリーにすることです。

モータドライブ基板 Ver.5 は「フリー追加セット」を追加することにより、停止動作をブレーキとフリーにすることができます。G 点が"1"のときの様子を下図に示します。





G 点が"1"なら、D 点は A 点や B 点の状態に関わらず"1"になります。よって、FET2 は常に OFF になり、モータは正転とフリーを繰り返します。

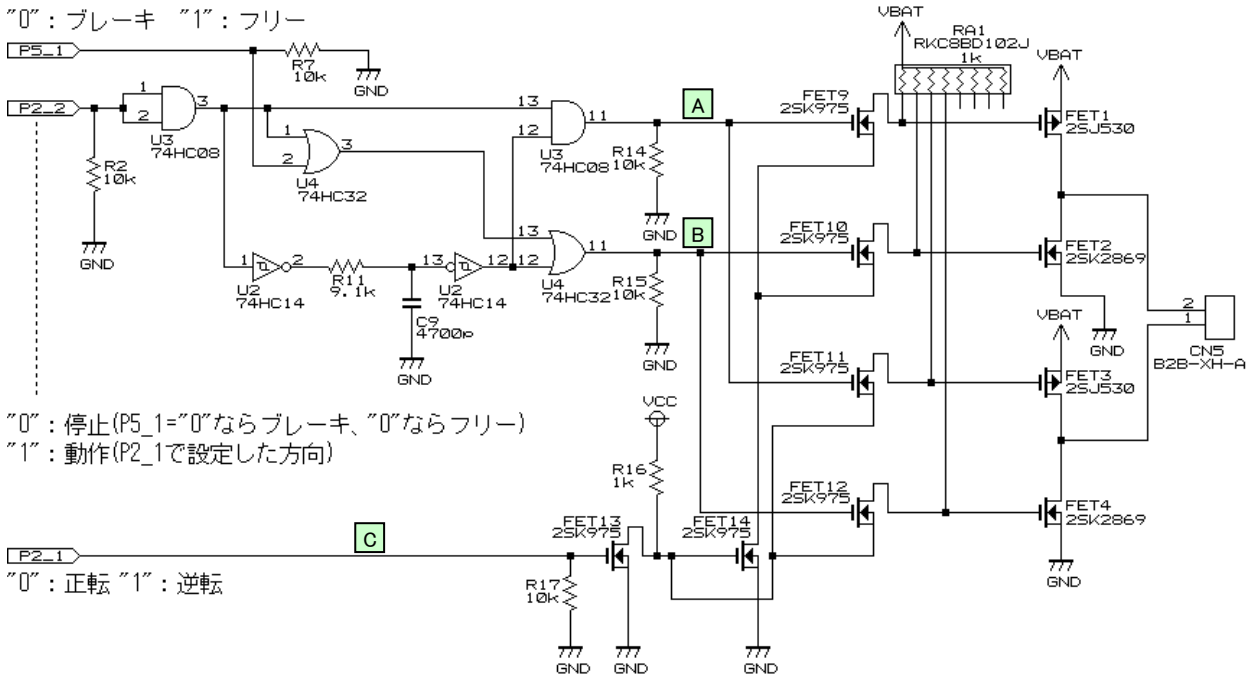
G 点が"0"のときは前記のとおり、正転とブレーキを繰り返します。

24.5.7 実際の回路

実際の回路は、積分回路、FET 回路、フリー回路の他、正転／逆転切り替え用回路が付加されています。下記回路は、左モータ用の回路です。端子は、下記の 3 本あります。

●P2_2…PWM を加える端子
●P2_1…正転／逆転を切り替える端子
●P5_1…ブレーキ／フリーを切り替える端子です。

回路図



方向：正転、停止：ブレーキのときの信号のレベルとモータの動作

A	B	C	FET1 のゲート	FET2 のゲート	FET3 のゲート	FET4 のゲート	CN5 2 ピン	CN5 1 ピン	モータ動作
0	0		10V (OFF)	10V (ON)	10V (OFF)	10V (ON)	0V	0V	ブレーキ
0	1		10V (OFF)	0V (OFF)	10V (OFF)	10V (ON)	フリー (開放)	0V	フリー
1	1	0	0V (ON)	0V (OFF)	10V (OFF)	10V (ON)	10V	0V	**正転**
0	1		10V (OFF)	0V (OFF)	10V (OFF)	10V (ON)	フリー (開放)	0V	フリー
0	0		10V (OFF)	10V (ON)	10V (OFF)	10V (ON)	0V	0V	ブレーキ

※A,B,C："0"=0V、"1"=5V

(3) 方向:逆転、停止:ブレーキのときの信号のレベルとモータの動作

A	B	C	FET1 のゲート	FET2 のゲート	FET3 のゲート	FET4 のゲート	CN5 2 ピン	CN5 1 ピン	モータ動作
0	0	1	10V (OFF)	10V (ON)	10V (OFF)	10V (ON)	0V	0V	ブレーキ
0	1		10V (OFF)	10V (ON)	10V (OFF)	0V (OFF)	0V	フリー (開放)	フリー
1	1		10V (OFF)	10V (ON)	0V (ON)	0V (OFF)	0V	10V	**逆転**
0	1		10V (OFF)	10V (ON)	10V (OFF)	0V (OFF)	0V	フリー (開放)	フリー
0	0		10V (OFF)	10V (ON)	10V (OFF)	10V (ON)	0V	0V	ブレーキ

(4) 方向:正転、停止:フリーのときの信号のレベルとモータの動作

A	B	C	FET1 のゲート	FET2 のゲート	FET3 のゲート	FET4 のゲート	CN5 2 ピン	CN5 1 ピン	モータ動作
0	1	0	10V (OFF)	0V (OFF)	10V (OFF)	10V (ON)	フリー (開放)	0V	**フリー**
0	1		10V (OFF)	0V (OFF)	10V (OFF)	10V (ON)	フリー (開放)	0V	フリー
1	1		0V (ON)	0V (OFF)	10V (OFF)	10V (ON)	10V	0V	正転
0	1		10V (OFF)	0V (OFF)	10V (OFF)	10V (ON)	フリー (開放)	0V	フリー
0	1		10V (OFF)	0V (OFF)	10V (OFF)	10V (ON)	フリー (開放)	0V	**フリー**

24.5.8 左モータの動作

左モータは、P2_1、P2_2、P5_1 の 3 端子で制御します。フリー追加セットが無い場合、P5_1 の部分は常に″0″になります。

左モータ方向 P2_1	左モータ PWM P2_2	左モータ停止動作 P5_1	モータ動作
0	PWM	0	PWM=″1″なら正転、″0″ならブレーキ
0	PWM	1	PWM=″1″なら正転、″0″ならフリー
1	PWM	0	PWM=″1″なら逆転、″0″ならブレーキ
1	PWM	1	PWM=″1″なら逆転、″0″ならフリー

左モータを正転とブレーキで動作させたい場合、P2_1=″0″、P5_1=″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。このときの停止は、ブレーキ動作になります。

24.5.9 右モータの動作

右モータは、P2_3、P2_4、P5_0 の 3 端子で制御します。フリー追加セットが無い場合、P5_0 の部分は常に″0″になります。

右モータ方向 P2_3	右モータ PWM P2_4	右モータ停止動作 P5_0	モータ動作
0	PWM	0	PWM=″1″なら正転、″0″ならブレーキ
0	PWM	1	PWM=″1″なら正転、″0″ならフリー
1	PWM	0	PWM=″1″なら逆転、″0″ならブレーキ
1	PWM	1	PWM=″1″なら逆転、″0″ならフリー

右モータを正転とフリーで動作させたい場合、P2_3=″0″、P5_0=″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。このときの停止は、フリー動作になります。

24.5.10 RY_R8C38 ボードとモータドライブ基板 Ver.5 の接続

RY_R8C38 ボードとモータドライブ基板 Ver.5 の接続を下表に示します。

RY_R8C38 ボードの CN4(ポート 2)とモータドライブ基板をフラットケーブルで接続すると、右モータ PWM 端子は P2_4、左モータ PWM 端子は P2_2 と接続されます。タイマ RD によるリセット同期 PWM モードを使用してこれらの端子から PWM 波形を出力、左右のモータを制御します。

モータドライブ基板 Ver.5		RY_R8C38 ボード CN4(ポート 2)			
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	LED1	2	P2_7	出力	P2_7 は通常の I/O ポートです。
3	LED0	3	P2_6	出力	P2_6 は通常の I/O ポートです。
4	サーボ	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	**右モータ 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	右モータ回転方向	6	P2_3	出力	P2_3 は通常の I/O ポートです。
7	**左モータ 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	左モータ回転方向	8	P2_1	出力	P2_1 は通常の I/O ポートです。
9	プッシュスイッチ	9	P2_0	入力	プッシュスイッチの状態を入力します。
10	GND	10	GND		

↑
フラットケーブルの結線

24.6 プログラムの解説

24.6.1 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。

```
 21 :  /*====================================*/
 22 :  /* シンボル定義                       */
 23 :  /*====================================*/
 24 :  #define PWM_CYCLE   39999              /* モータPWMの周期              */

中略

102 :       /* タイマRD リセット同期PWMモードの設定*/
103 :       /* PWM周期 = 1 / 20[MHz]   * カウントソース *（TRDGRA0+1）
104 :                   = 1 / (20*10^6) * 8              * 40000
105 :                   = 0.016[s] = 16[ms]
106 :       */
107 :       trdpsr0 = 0x08;                   /* TRDIOB0,C0,D0端子設定        */
108 :       trdpsr1 = 0x05;                   /* TRDIOA1,B1,C1,D1端子設定     */
109 :       trdmr   = 0xf0;                   /* バッファレジスタ設定         */
110 :       trdfcr  = 0x01;                   /* リセット同期PWMモードに設定  */
111 :       trdcr0  = 0x23;                   /* ソースカウントの選択: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 = 3750;         /* P2_5端子のON幅設定           */
116 :       trdoer1 = 0xcd;                   /* 出力端子の選択               */
117 :       trdstr  = 0x0d;                   /* TRD0カウント開始             */
```

PWM の周期は 16ms です。112 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 24 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、24 行にある「39999」を変更してください。

①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/38A では、割り当てる端子は決まっており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 trdiod0sel0	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			

②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/38A では、割り当てる端子は決まっており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 trdiod1sel0	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			

③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			

④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_trdfcr	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_trdfcr	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_trdfcr	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_trdfcr bit0:cmd0_trdfcr	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			

⑤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、 f1/20MHz＝1/20MHz＝50ns 設定できる PWM 周期の最大は、 50ns×65,536＝**3.2768ms** 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、 f2/20MHz＝2/20MHz＝100ns 設定できる PWM 周期の最大は、 100ns×65,536＝**6.5536ms** 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、 f4/20MHz＝4/20MHz＝200ns 設定できる PWM 周期の最大は、 200ns×65,536＝**13.1072ms** 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、 f8/20MHz＝8/20MHz＝400ns 設定できる PWM 周期の最大は、 400ns×65,536＝**26.2144ms** 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、 f32/20MHz＝32/20MHz＝1600ns 設定できる PWM 周期の最大は、 1600ns×65,536＝**104.8576ms** 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。

⑥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。

PWM 周期＝タイマ RD カウンタ 0 のカウントソース×(TRDGRA0＋1)

TRDGRA0 を左辺に移動して、TRDGRA0 を求める式に変形します。

TRDGRA0＝PWM 周期／タイマ RD カウンタ 0 のカウントソースー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。

TRDGRA0＝周期　　／カウントソースー1
TRDGRA0＝$(16\times10^{-3})/(400\times10^{-9})-1$
TRDGRA0＝40000－1 ＝ 39999

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   */
    ～～～～～～～～～～～～～～～～
}
```

⑦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。

P2_2 端子から出力される PWM 波形の ON 幅＝タイマ RD カウンタ 0 のカウントソース×(TRDGRB0＋1)

TRDGRB0 を左辺に移動して、TRDGRB0 を求める式に変形します。

TRDGRB0＋1＝P2_2 端子から出力される PWM 波形の ON 幅／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。

TRDGRB0＋1＝ON 幅　／カウントソース
TRDGRB0＋1＝$(1\times10^{-3})/(400\times10^{-9})$
TRDGRB0＋1＝2500
TRDGRB0　　＝2499

■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。

⑧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。

P2_4 端子から出力される PWM 波形の ON 幅＝タイマ RD カウンタ 0 のカウントソース×(TRDGRA1＋1)

TRDGRA1 を左辺に移動して、TRDGRA1 を求める式に変形します。

TRDGRA1＝P2_4 端子から出力される PWM 波形の ON 幅／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。

TRDGRA1＝ON 幅　　／カウントソース－1
TRDGRA1＝$(10\times10^{-3})/(400\times10^{-9})-1$
TRDGRA1＝25000－1 ＝ 24999

■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。

⑨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。

P2_5 端子から出力される PWM 波形の ON 幅＝タイマ RD カウンタ 0 のカウントソース×(TRDGRB1＋1)

TRDGRB1 を左辺に移動して、TRDGRB1 を求める式に変形します。

TRDGRB1＝P2_5 端子から出力される PWM 波形の ON 幅／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 1.5ms 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。

TRDGRB1＝ON 幅　／カウントソース－1
TRDGRB1＝$(1.5\times10^{-3})/(400\times10^{-9})-1$
TRDGRB1＝3750－1 ＝ 3749

■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 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			

⑪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			

24.6.2 motor 関数

左モータ、右モータへ PWM を出力する関数です。また、引数の符号により正転、逆転の制御も行います。

```
158 :  void motor( int data1, int data2 )
159 :  {
160 :      int    motor_r, motor_l, sw_data;
161 :
162 :      sw_data = dipsw_get() + 5;
163 :      motor_l = data1 * sw_data / 20;
164 :      motor_r = data2 * sw_data / 20;
165 :
166 :      /* 左モータ制御 */
167 :      if( motor_l >= 0 ) {
168 :          p2 &= 0xfd;
169 :          trdgrd0 = (long)( PWM_CYCLE - 1 ) * motor_l / 100;
170 :      } else {
171 :          p2 |= 0x02;
172 :          trdgrd0 = (long)( PWM_CYCLE - 1 ) * ( -motor_l ) / 100;
173 :      }
174 :
175 :      /* 右モータ制御 */
176 :      if( motor_r >= 0 ) {
177 :          p2 &= 0xf7;
178 :          trdgrc1 = (long)( PWM_CYCLE - 1 ) * motor_r / 100;
179 :      } else {
180 :          p2 |= 0x08;
181 :          trdgrc1 = (long)( PWM_CYCLE - 1 ) * ( -motor_r ) / 100;
182 :      }
183 :  }
```

(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。

左モータに出力される PWM ＝ −70×(12＋5)÷20 ＝ −70×0.85 ＝ −59.5 ＝ −59%
右モータに出力される PWM ＝ 100×(12＋5)÷20 ＝ 100×0.85 ＝ 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

```
162 :     sw_data = dipsw_get() + 5;        dipsw_get() = ディップスイッチの値0～15
163 :     motor_l = data1 * sw_data / 20;
164 :     motor_r = data2 * sw_data / 20;
```

162 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。
163 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。
164 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。

```
166 :      /* 左モータ制御 */
167 :      if( motor_l >= 0 ) {
168 :          p2 &= 0xfd;              p2 = p2 AND 0xfd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 :          trdgrd0 = (long)( PWM_CYCLE - 1 ) * ( -motor_l ) / 100;
173 :      }
```

167 行	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 168～169 行、負の数なら 171～172 行を実行します。
168 行 ～ 169 行	正の数なら 168～169 行を実行します。 P2_1=″0″を設定し、P2_2 端子から PWM 出力すると、PWM の割合に応じて左モータが正転します。 168 行目で下表のビット演算を行い、P2_1 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	1	1	1	1	1	**0**	1
結果	P2_7	P2_6	P2_5	P2_4	P2_3	P2_2	**0**	P2_0

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。

$$TRDGRD0 = (PWM\_CYCLE - 1) \times \frac{motor\_l\ (0 \sim 100)}{100}$$

$$= 39998 \times \frac{motor\_l\ (0 \sim 100)}{100}$$

例えば motor_l=80 なら、TRDGRD0 は次のように計算されます。

TRDGRD0＝39998×80／100＝31998.4＝31998

171 行
～
172 行

負の数なら 171～172 行を実行します。
P2_1=″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逆転します。
171 行目で下表のビット演算を行い、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。

$$TRDGRD0 = (PWM\_CYCLE - 1) \times \frac{-motor\_l\ (-1 \sim -100)}{100}$$

$$= 39998 \times \frac{-motor\_l\ (-1 \sim -100)}{100}$$

ポイントは、motor_l 変数が負の数ということです。回路的には P2_1=″1″で逆転の設定になっています。そのため motor_l は、正の数に直して計算します。直し方は、計算式の中で「−motor_l」としています。例えば motor_l=−50 なら、TRDGRD0 は次のように計算されます。

TRDGRD0＝39998×{−(−50)}／100＝39998×50／100＝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。

```
175 :       /* 右モータ制御 */
176 :       if( motor_r >= 0 ) {
177 :           p2 &= 0xf7;             p2 = p2 AND 0xf7という意味です
178 :           trdgrc1 = (long)( PWM_CYCLE - 1 ) * motor_r / 100;
179 :       } else {
180 :           p2 |= 0x08;             p2 = p2 OR  0x08という意味です
181 :           trdgrc1 = (long)( PWM_CYCLE - 1 ) * ( -motor_r ) / 100;
182 :       }
```

176 行	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 177～178 行、負の数なら 180～181 行を実行します。
177 行 ～ 178 行	正の数なら 177～178 行を実行します。 P2_3="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正転します。 177 行目で下表のビット演算を行い、P2_3 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 / 1 / 1 / 1 / **0** / 1 / 1 / 1 結果 / P2_7 / P2_6 / P2_5 / P2_4 / **0** / P2_2 / P2_1 / P2_0 178 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。 $TRDGRC1 = (PWM\_CYCLE - 1) \times \frac{motor\_r\ (0 \sim 100)}{100}$ $= 39998 \times \frac{motor\_r\ (0 \sim 100)}{100}$ 例えば motor_r=20 なら、TRDGRC1 は次のように計算されます。 TRDGRC1＝39998×20／100＝7999.6＝7999

180 行
～
181 行

負の数なら 180～181 行を実行します。
P2_3=″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逆転します。
180 行目で下表のビット演算を行い、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

181 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。

$$TRDGRC1 = (PWM\_CYCLE - 1) \times \frac{-motor\_r\ (-1 \sim -100)}{100}$$

$$= 39998 \times \frac{-motor\_r\ (-1 \sim -100)}{100}$$

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。

TRDGRC1＝39998×{−(−90)}／100＝39998×90／100＝35998.2＝35998

24.6.3 main 関数

```
 42 :  void main( void )
 43 :  {
 44 :      init();                              /* 初期化                       */
 45 :      asm(" fset I ");                     /* 全体の割り込み許可           */
 46 :
 47 :      while( 1 ) {
 48 :          motor( 100 , 0);
 49 :          timer( 1000 );
 50 :          motor( 0, 80 );
 51 :          timer( 1000 );
 52 :          motor( -60, 0 );
 53 :          timer( 1000 );
 54 :          motor( 0, -40 );
 55 :          timer( 1000 );
 56 :          motor( 0, 0 );
 57 :          timer( 1000 );
 58 :      }
 59 :  }
```

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、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。
45 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。
47 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 48 行～57 行が実行され続けます。
48 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、実際にモータへ出力される PWM 波形は、ディップスイッチの割合も加わります。
49 行	1000ms の時間、この行で待ちます。
50 行～ 57 行	同様にモータを制御します。

25.　サーボの制御（プロジェクト：servo）

25.1　概要

本章では、マイコンカー用のモータドライブ基板 Ver.5 のサーボを制御する方法を紹介します。サーボの角度を±90 度、自由に変えることができます。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。

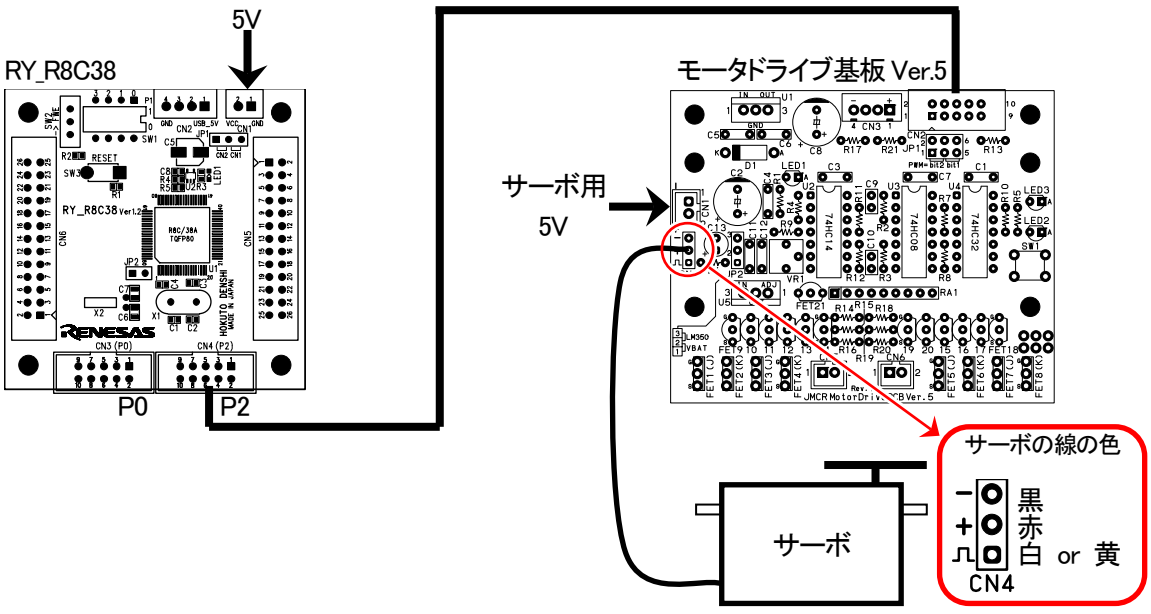
25.2　接続

■使用ポート

マイコンのポート	接続内容
P2 (CN4)	マイコンカー用モータドライブ基板 Ver.5 を接続

■接続例

RY_R8C38 ボードとモータドライブ基板 Ver.5 をフラットケーブルで接続します。

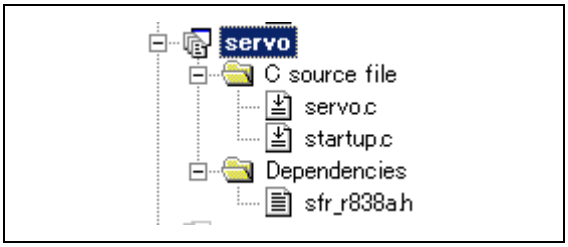


※モータドライブ基板 Ver.4 も接続可能です。プログラムも同一のもので動作します。

■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。

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。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥servo¥startup.c
2	servo.c	実際に制御するプログラムが書かれています。R8C/38A の内蔵周辺機能(SFR)の初期化も行います。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥servo¥servo.c
3	sfr_r838a.h	R8C/38A マイコンの内蔵周辺機能を制御するためのレジスタ（Special Function Registers)を定義したファイルです。 ファイルの位置→C:¥Workspace¥common_r8c38a¥sfr_r838a.h

25.4 プログラム「servo.c」

```
 1 :  /***********************************************************************/
 2 :  /* 対象マイコン R8C/38A                                                */
 3 :  /* ﾌｧｲﾙ内容     マイコンカー用モータドライブ基板Ver.5のサーボ制御      */
 4 :  /* バージョン   Ver.1.02                                               */
 5 :  /* Date         2011.10.18                                             */
 6 :  /* Copyright    ジャパンマイコンカーラリー実行委員会                   */
 7 :  /***********************************************************************/
 8 :  /*
 9 :  入力：マイコンボード上のディップスイッチ
10 :  出力：マイコンカー用モータドライブ基板Ver.5のサーボ
11 :
12 :  マイコンカー用モータドライブ基板Ver.5のサーボを制御します。
13 :  */
14 :
15 :  /*=====================================*/
16 :  /* インクルード                        */
17 :  /*=====================================*/
18 :  #include "sfr_r838a.h"                 /* R8C/38A SFRの定義ファイル    */
19 :
20 :  /*=====================================*/
21 :  /* シンボル定義                        */
22 :  /*=====================================*/
23 :  #define PWM_CYCLE       39999          /* モータPWMの周期              */
24 :  #define SERVO_CENTER    3750           /* サーボのセンタ値             */
25 :  #define HANDLE_STEP     22             /* 1°分の値                    */
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 handle( int angle );
33 :
34 :  /*=====================================*/
35 :  /* グローバル変数の宣言                */
36 :  /*=====================================*/
37 :  unsigned long cnt_rb;                  /* タイマRB用                   */
38 :
```

```
 39 :  /************************************************************************/
 40 :  /* メインプログラム                                                     */
 41 :  /************************************************************************/
 42 :  void main( void )
 43 :  {
 44 :      init();                             /* 初期化                       */
 45 :      asm(" fset I ");                    /* 全体の割り込み許可           */
 46 :
 47 :      while( 1 ) {
 48 :          handle( 0 );
 49 :          timer( 1000 );
 50 :          handle( 30 );
 51 :          timer( 1000 );
 52 :          handle( 0 );
 53 :          timer( 1000 );
 54 :          handle( -30 );
 55 :          timer( 1000 );
 56 :      }
 57 :  }
 58 :
 59 :  /************************************************************************/
 60 :  /* R8C/38A スペシャルファンクションレジスタ(SFR)の初期化                */
 61 :  /************************************************************************/
 62 :  void init( void )
 63 :  {
 64 :      int i;
 65 :
 66 :      /* クロックをXINクロック(20MHz)に変更 */
 67 :      prc0  = 1;                          /* プロテクト解除               */
 68 :      cm13  = 1;                          /* P4_6,P4_7をXIN-XOUT端子にする*/
 69 :      cm05  = 0;                          /* XINクロック発振              */
 70 :      for(i=0; i<50; i++ );               /* 安定するまで少し待つ(約10ms) */
 71 :      ocd2  = 0;                          /* システムクロックをXINにする  */
 72 :      prc0  = 0;                          /* プロテクトON                 */
 73 :
 74 :      /* ポートの入出力設定 */
 75 :      prc2 = 1;                           /* PD0のプロテクト解除          */
 76 :      pd0 = 0x00;                         /*                              */
 77 :      pd1 = 0xd0;                         /* 5:RXD0 4:TXD0 3-0:DIP SW     */
 78 :      pd2 = 0xfe;                         /* 7-0:モータドライブ基板Ver.5  */
 79 :      pd3 = 0xff;                         /*                              */
 80 :      p4  = 0x20;                         /* P4_5のLED:初期は点灯         */
 81 :      pd4 = 0xb8;                         /* 7:XOUT 6:XIN 5:LED 2:VREF    */
 82 :      pd5 = 0xff;                         /*                              */
 83 :      pd6 = 0xff;                         /*                              */
 84 :      pd7 = 0xff;                         /*                              */
 85 :      pd8 = 0xff;                         /*                              */
 86 :      pd9 = 0x3f;                         /*                              */
 87 :      pur0 = 0x04;                        /* P1_3～P1_0のプルアップON     */
 88 :
 89 :      /* タイマRBの設定 */
 90 :      /* 割り込み周期 = 1 / 20[MHz]   * (TRBPRE+1) * (TRBPR+1)
 91 :                      = 1 / (20*10^6) * 200        * 100
 92 :                      = 0.001[s] = 1[ms]
 93 :      */
 94 :      trbmr  = 0x00;                      /* 動作モード、分周比設定       */
 95 :      trbpre = 200-1;                     /* プリスケーラレジスタ         */
 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 :      trdpsr0 = 0x08;                     /* TRDIOB0,C0,D0端子設定        */
106 :      trdpsr1 = 0x05;                     /* TRDIOA1,B1,C1,D1端子設定     */
107 :      trdmr   = 0xf0;                     /* バッファレジスタ設定         */
108 :      trdfcr  = 0x01;                     /* リセット同期PWMモードに設定  */
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 :      trdoer1 = 0xcd;                     /* 出力端子の選択               */
115 :      trdstr  = 0x0d;                     /* TRD0カウント開始             */
116 :  }
117 :
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;
125 :
126 :      sw = p1 & 0x0f;                     /* P1_3～P1_0読み込み           */
127 :
128 :      return  sw;
129 :  }
```

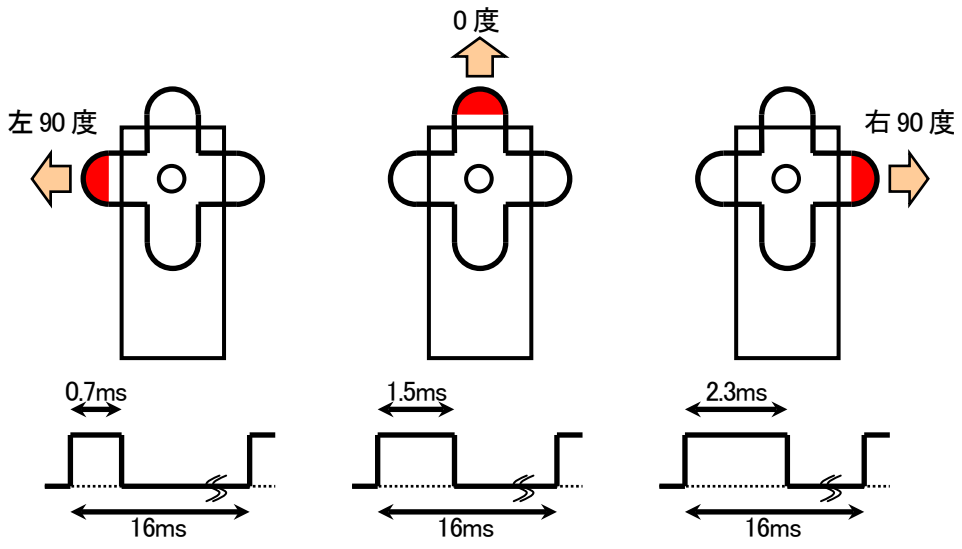
```
130 :
131 :  /************************************************************************/
132 :  /* タイマ本体                                                           */
133 :  /* 引数    タイマ値 1=1ms                                               */
134 :  /************************************************************************/
135 :  void timer( unsigned long timer_set )
136 :  {
137 :      cnt_rb = 0;
138 :      while( cnt_rb < timer_set );
139 :  }
140 :
141 :  /************************************************************************/
142 :  /* タイマRB 割り込み処理                                                */
143 :  /************************************************************************/
144 :  #pragma interrupt intTRB(vect=24)
145 :  void intTRB( void )
146 :  {
147 :      cnt_rb++;
148 :  }
149 :
150 :  /************************************************************************/
151 :  /* サーボハンドル操作                                                   */
152 :  /* 引数    サーボ操作角度：-90～90                                      */
153 :  /*         -90で左へ90度、0でまっすぐ、90で右へ90度回転                  */
154 :  /* 戻り値 なし                                                          */
155 :  /************************************************************************/
156 :  void handle( int angle )
157 :  {
158 :      /* サーボが左右逆に動く場合は、「-」を「+」に替えてください */
159 :      trdgrd1 = SERVO_CENTER - angle * HANDLE_STEP;
160 :  }
161 :
162 :  /************************************************************************/
163 :  /* end of file                                                          */
164 :  /************************************************************************/
```

25.5　サーボ

25.5.1　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。



・PWM 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。

- PWM 周期 16[ms]
- ON 幅 0.7～2.3[ms]

25.5.2 RY_R8C38 ボードとモータドライブ基板 Ver.5 の接続

RY_R8C38 ボードとモータドライブ基板 Ver.5 の接続を下表に示します。

RY_R8C38 ボードの CN4（ポート 2）とモータドライブ基板をフラットケーブルで接続すると、サーボ端子が P2_5 と接続されます。タイマ RD によるリセット同期 PWM モードを使用してこの端子から PWM 波形を出力、サーボを制御します。

モータドライブ基板 Ver.5			RY_R8C38 ボード CN4（ポート 2）			
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	LED1		2	P2_7	出力	P2_7 は通常の I/O ポートです。
3	LED0		3	P2_6	出力	P2_6 は通常の I/O ポートです。
4	**サーボ**		**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	右モータ 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	右モータ回転方向		6	P2_3	出力	P2_3 は通常の I/O ポートです。
7	左モータ 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	左モータ回転方向		8	P2_1	出力	P2_1 は通常の I/O ポートです。
9	プッシュスイッチ		9	P2_0	入力	プッシュスイッチの状態を入力します。
10	GND		10	GND		

↑
フラットケーブルの結線

25.6 プログラムの解説

25.6.1 init 関数(タイマ RD の設定)

```
 20 :  /*====================================*/
 21 :  /* シンボル定義                       */
 22 :  /*====================================*/
 23 :  #define PWM_CYCLE       39999          /* モータPWMの周期              */
 24 :  #define SERVO_CENTER    3750           /* サーボのセンタ値             */
 25 :  #define HANDLE_STEP     22             /* 1° 分の値                    */

中略

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 :      trdpsr0 = 0x08;                    /* TRDIOB0,C0,D0端子設定        */
106 :      trdpsr1 = 0x05;                    /* TRDIOA1,B1,C1,D1端子設定     */
107 :      trdmr   = 0xf0;                    /* バッファレジスタ設定         */
108 :      trdfcr  = 0x01;                    /* リセット同期PWMモードに設定  */
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 :      trdoer1 = 0xcd;                    /* 出力端子の選択               */
115 :      trdstr  = 0x0d;                    /* TRD0カウント開始             */
```

23 行	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。 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のようになります。 TRDGRA0 ＝PWM 周期／タイマ RD カウンタ 0 のカウントソース－1 ＝(16×10^{-3})／(400×10^{-9})－1 ＝40000-1＝39999 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。
24 行	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。 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のようになります。 TRDGRB1 ＝P2_5 端子から出力される PWM 波形の ON 幅／タイマ RD カウンタ 0 のカウントソース－1 ＝(1.5×10^{-3})／(400×10^{-9})－1 ＝3750－1 ＝ 3749 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**。

25 行	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。 • 左 90 度の PWM の ON 幅 TRDGRB1 ＝P2_5 端子から出力される PWM 波形の ON 幅／タイマ RD カウンタ 0 のカウントソース－1 ＝$(0.7\times10^{-3})/(400\times10^{-9})-1$ ＝1750-1 ＝ 1749 • 右 90 度の PWM の ON 幅 TRDGRB1 ＝P2_5 端子から出力される PWM 波形の ON 幅／タイマ RD カウンタ 0 のカウントソース－1 ＝$(2.3\times10^{-3})/(400\times10^{-9})-1$ ＝5750-1 ＝ 5749 • 1 度当たりの値 (右－左)/180 ＝ (5749－1749)/180 ＝ 22.22 ≒ 22 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。
105～ 115 行	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。 • 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。

25.6.2 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。

```
150 :  /************************************************************************/
151 :  /* サーボハンドル操作                                                   */
152 :  /* 引数    サーボ操作角度：-90～90                                      */
153 :  /*         -90で左へ90度、0でまっすぐ、90で右へ90度回転                  */
154 :  /* 戻り値 なし                                                          */
155 :  /************************************************************************/
156 :  void handle( int angle )
157 :  {
158 :      /* サーボが左右逆に動く場合は、「-」を「+」に替えてください */
159 :      trdgrd1 = SERVO_CENTER - angle * HANDLE_STEP;
160 :  }
```

(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。
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。

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

```
159 :      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

```
TRDGRD1 = SERVO_CENTER - angle * HANDLE_STEP
        = 3749         - 0     * 22
        = 3749
```

• 30 度のとき

```
TRDGRD1 = SERVO_CENTER - angle * HANDLE_STEP
        = 3749         - 30    * 22
        = 3749         - 660
        = 3089
```

• -45 度のとき

```
TRDGRD1 = SERVO_CENTER - angle * HANDLE_STEP
        = 3749         - (-45) * 22
        = 3749         - (-990)
        = 4739
```

25.6.3 main 関数

```
42 :  void main( void )
43 :  {
44 :      init();                              /* 初期化                       */
45 :      asm(" fset I ");                     /* 全体の割り込み許可           */
46 :
47 :      while( 1 ) {
48 :          handle( 0 );
49 :          timer( 1000 );
50 :          handle( 30 );
51 :          timer( 1000 );
52 :          handle( 0 );
53 :          timer( 1000 );
54 :          handle( -30 );
55 :          timer( 1000 );
56 :      }
57 :  }
```

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、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。
45 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。
47 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 48 行～55 行が実行され続けます。
48 行	handle 関数を使って、サーボを 0 度にします。
49 行	1000ms の時間、この行で待ちます。
50 行～ 55 行	同様にサーボを制御します。

26.　通信（プロジェクト：uart0）

26.1　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。

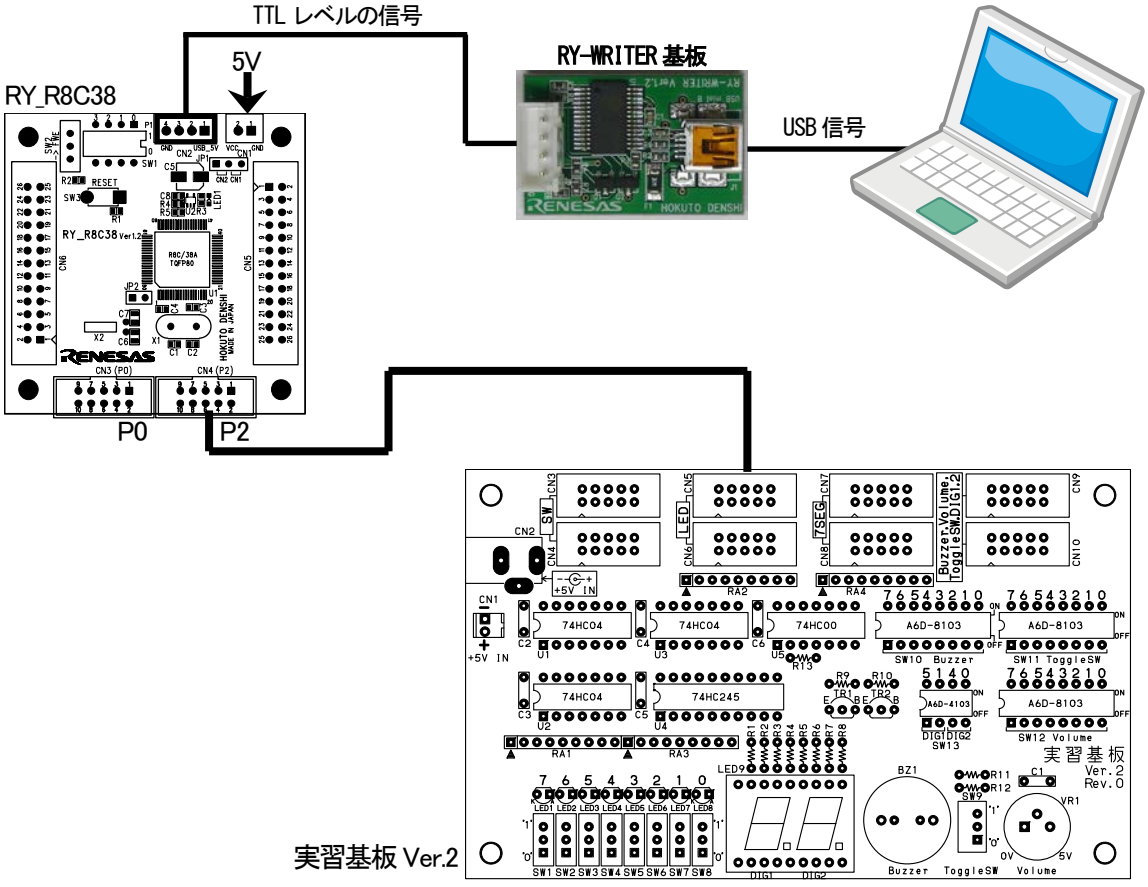
26.2　接続

■使用ポート

マイコンのポート	接続内容
4 ピンコネクタ (CN2)	CN2 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マイコンで受信し、その値を LED へ出力します。
P2 (CN4)	実習基板 Ver.2 の LED 部など、出力機器を接続します。

■接続例

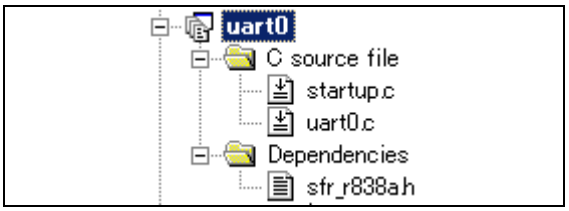
実習基板 Ver.2 を使ったときの接続例を下記に示します。



■操作方法

パソコン側は Tera Term 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2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、「**26.7　実習手順**」を参照してください。

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。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥uart0¥startup.c
2	uart0.c	実際に制御するプログラムが書かれています。R8C/38A の内蔵周辺機能(SFR)の初期化も行います。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥uart0¥uart0.c
3	sfr_r838a.h	R8C/38A マイコンの内蔵周辺機能を制御するためのレジスタ（Special Function Registers)を定義したファイルです。 ファイルの位置→C:¥Workspace¥common_r8c38a¥sfr_r838a.h

26.4　プログラム「uart0.c」

```
 1 :  /***********************************************************************/
 2 :  /* 対象マイコン  R8C/38A                                               */
 3 :  /* ﾌｧｲﾙ内容      UART0の使用例                                         */
 4 :  /* バージョン    Ver.1.00                                              */
 5 :  /* Date          2010.04.01                                            */
 6 :  /* Copyright     ジャパンマイコンカーラリー実行委員会                  */
 7 :  /***********************************************************************/
 8 :  /*
 9 :  入力：UART0（パソコンのキーボードで入力したデータ）
10 :        ※パソコンはTeraTermProなどの通信ソフトを使用します
11 :  出力：UART0（通信ソフトの画面）
12 :        P2_7-P2_0（実習基板Ver.2のLED部）
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 :  通信ソフトの画面にそのまま表示されます。
17 :  */
18 :
19 :  /*====================================*/
20 :  /* インクルード                       */
21 :  /*====================================*/
22 :  #include <stdio.h>
23 :  #include "sfr_r838a.h"                 /* R8C/38A 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 :  int get_uart0( unsigned char *s );
34 :  int put_uart0( unsigned char r );
35 :
```

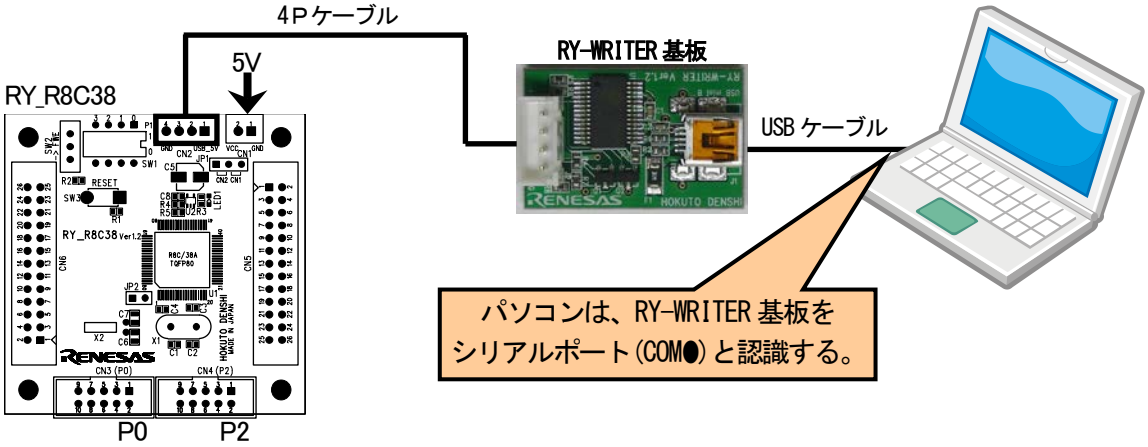
```
 36 :  /************************************************************************/
 37 :  /* メインプログラム                                                     */
 38 :  /************************************************************************/
 39 :  void main( void )
 40 :  {
 41 :      unsigned char   d;
 42 :      int             ret;
 43 :
 44 :      init();                             /* SFRの初期化                  */
 45 :
 46 :      while( 1 ) {
 47 :          ret = get_uart0( &d );
 48 :          p2 = d;
 49 :          if( ret == 1 ) put_uart0( d );
 50 :      }
 51 :  }
 52 :
 53 :  /************************************************************************/
 54 :  /* R8C/38A 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;                           /* PD0のプロテクト解除          */
 70 :      pd0 = 0x00;                         /*                              */
 71 :      pd1 = 0xd0;                         /* 5:RXD0 4:TXD0 3-0:DIP SW     */
 72 :      pd2 = 0xff;                         /* 7-0:LED                      */
 73 :      pd3 = 0xff;                         /*                              */
 74 :      p4  = 0x20;                         /* P4_5のLED:初期は点灯         */
 75 :      pd4 = 0xb8;                         /* 7:XOUT 6:XIN 5:LED 2:VREF    */
 76 :      pd5 = 0xff;                         /*                              */
 77 :      pd6 = 0xff;                         /*                              */
 78 :      pd7 = 0xff;                         /*                              */
 79 :      pd8 = 0xff;                         /*                              */
 80 :      pd9 = 0x3f;                         /*                              */
 81 :      pur0 = 0x04;                        /* P1_3～P1_0のプルアップON     */
 82 :
 83 :      /* UART0の設定 */
 84 :      /* U0BRG = ｶｳﾝﾄｿｰｽの周波数/(設定したいbps*16)-1
 85 :               = 20MHz          /(     9600    *16)-1
 86 :               = 129.208 = 129
 87 :      */
 88 :      u0sr = 0x05;                        /* P14=TXD0,P15=RXD0に設定      */
 89 :      u0c0 = 0x00;                        /* カウントソースなどの設定     */
 90 :      u0c1 = 0x05;                        /* 送信､受信許可               */
 91 :      u0brg = 129;                        /* 通信速度=9600pbs             */
 92 :      u0mr = 0x05;                        /* UART0 ﾃﾞｰﾀ長8bit 1ｽﾄｯﾌﾟﾋﾞｯﾄ  */
 93 :  }
 94 :
 95 :  /************************************************************************/
 96 :  /* １文字受信                                                           */
 97 :  /* 引数    受信文字格納アドレス                                         */
 98 :  /* 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            */
 99 :  /************************************************************************/
100 :  int get_uart0( unsigned char *s )
101 :  {
102 :      int ret = 0, i;
103 :      unsigned int data;
104 :
105 :      if (ri_u0c1 == 1){                  /* 受信データあり？             */
106 :          data = u0rb;
107 :          *s = (unsigned char)data;
108 :          ret = 1;
109 :          if( data & 0xf000 ) {           /* エラーあり？                 */
110 :              /* エラー時は再設定 */
111 :              re_u0c1 = 0;
112 :              for( i=0; i<50; i++ );
113 :              re_u0c1 = 1;
114 :
115 :              ret = -1;
116 :          }
117 :      }
118 :      return ret;
119 :  }
120 :
```

```
121 :  /************************************************************************/
122 :  /* 1文字出力                                                            */
123 :  /* 引数   送信データ                                                    */
124 :  /* 戻り値 0:送信中のため、送信できず 1:送信セット完了                   */
125 :  /************************************************************************/
126 :  int put_uart0( unsigned char r )
127 :  {
128 :      if(ti_u0c1 == 1) {                  /* 送信データなし？             */
129 :          u0tbl = r;
130 :          return 1;
131 :      } else {
132 :          /* 先に送信中(今回のデータは送信せずに終了) */
133 :          return 0;
134 :      }
135 :  }
136 :
137 :  /************************************************************************/
138 :  /* end of file                                                          */
139 :  /************************************************************************/
```

26.5　パソコンと RY_R8C38 ボードの通信

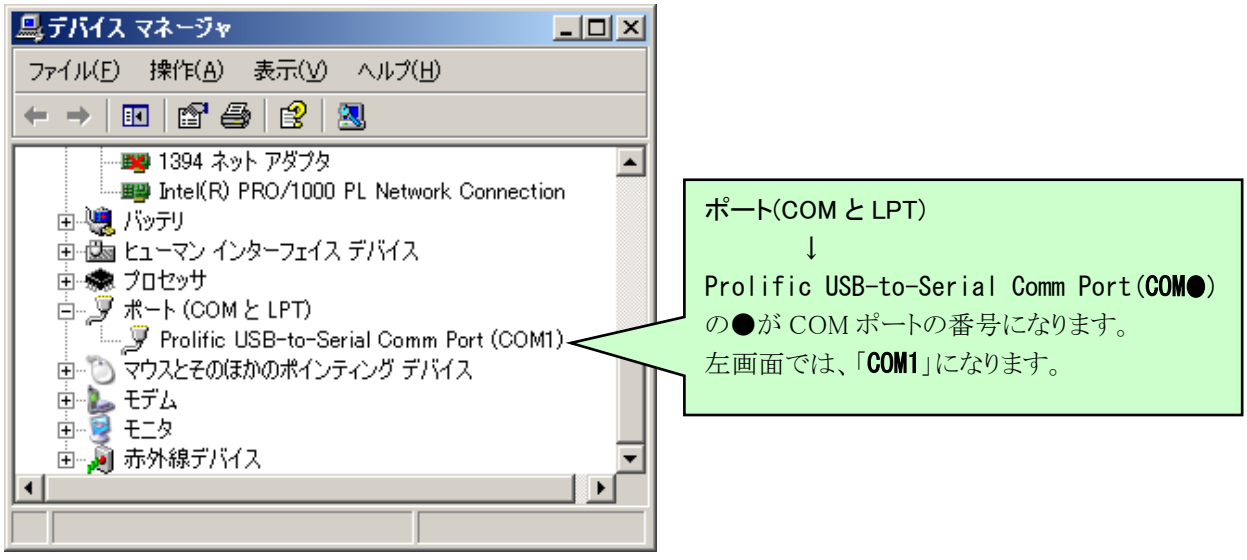
26.5.1　RY_R8C38 ボードの通信方法

RY_R8C38 ボードは、TTL 信号で通信を行います。そのため、パソコンと RY_R8C38 ボードの間に RY-WRITER 基板などの USB 信号を TTL 信号に変換する基板を入れて通信を行います。RY-WRITER 基板には、USB-シリアル変換 IC が搭載されており、パソコン(Tera Term やハイパーターミナルなどの通信ソフト)は RY_R8C38 ボードを COM ポート(シリアルポート)として認識します。



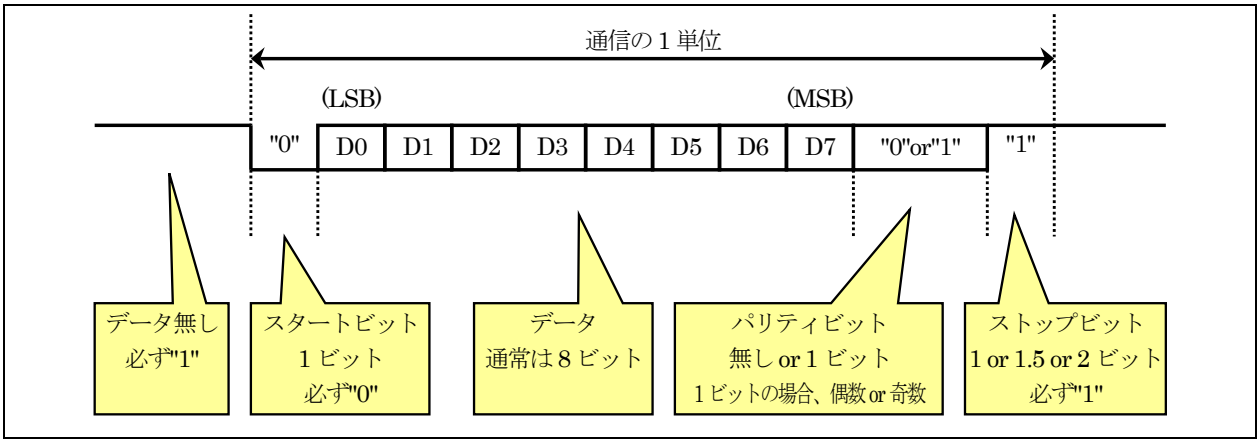
26.5.2　COM ポートの確認

USB-TTL 変換基板をパソコンの 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ーをクリックします。



26.5.3 シリアル通信の設定

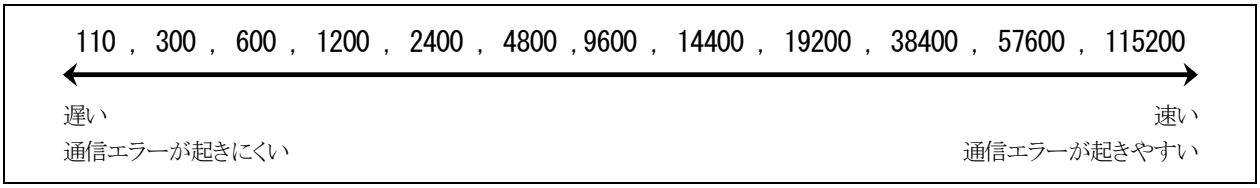
1 文字のデータを送る場合、文字データだけ送るわけではありません。受信側で正しく受け取るために送信側は、「スタートビット、データ、パリティビット、ストップビット」という形で送ります（下図）。



※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。Tera Term 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ければいけません。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、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

26.6 プログラムの解説

26.6.1 init 関数(UART0 の設定)

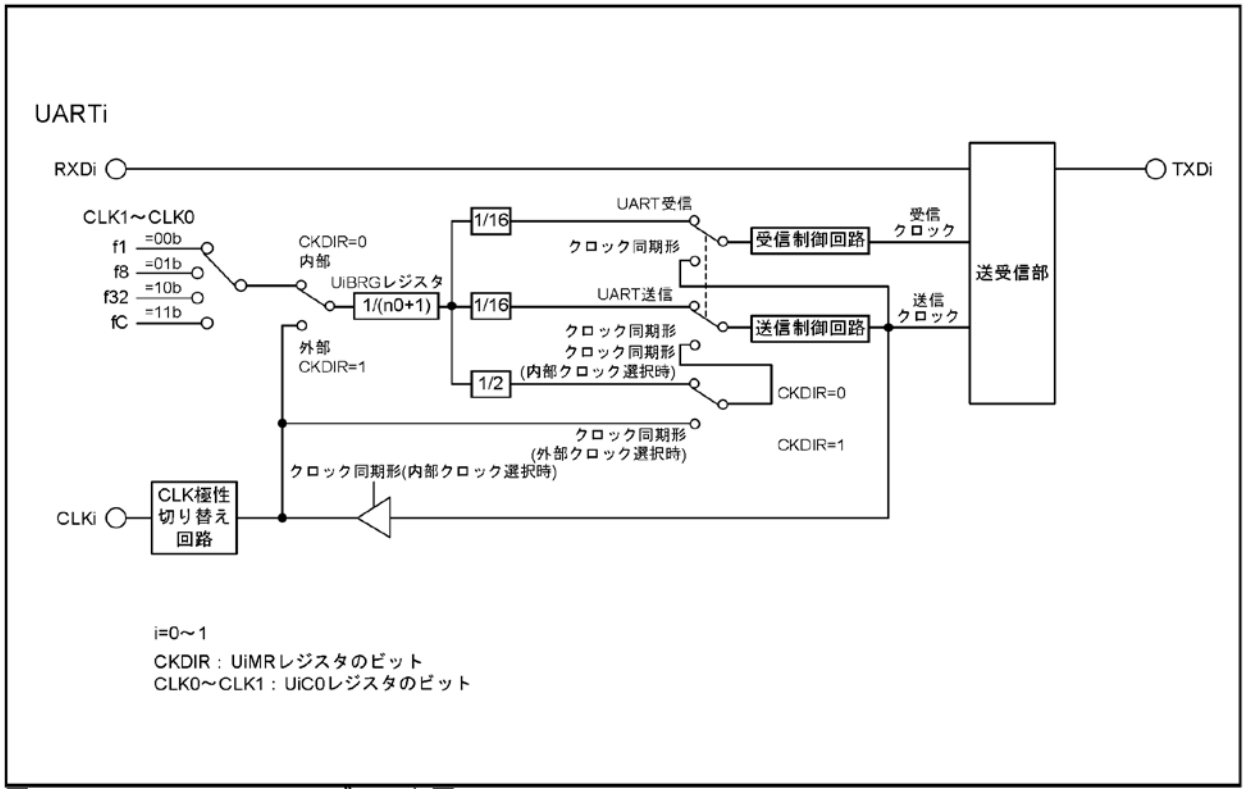
シリアルインターフェース 0(UART0)の設定プログラムは、次のようになります。

```
 83 :       /* UART0の設定 */
 84 :       /* U0BRG = ｶｳﾝﾄｿｰｽの周波数/(設定したいbps*16)-1
 85 :                 = 20MHz            /(      9600      *16)-1
 86 :                 = 129.208 = 129
 87 :       */
 88 :       u0sr = 0x05;                          /* P14=TXD0,P15=RXD0に設定       */
 89 :       u0c0 = 0x00;                          /* カウントソースなどの設定      */
 90 :       u0c1 = 0x05;                          /* 送信､受信許可                 */
 91 :       u0brg = 129;                          /* 通信速度=9600pbs              */
 92 :       u0mr = 0x05;                          /* UART0 ﾃﾞｰﾀ長8bit 1ｽﾄｯﾌﾟﾋﾞｯﾄ  */
```

(1) シリアルインターフェース

R8C/38A には、シリアルインタフェース(UART)が UART0～UART2 の 3 チャネルあります。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せんが「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、シリアルインター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。



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 RY_R8C38 ボードの CN2 の 2 ピンに接続されています。
RXD0	P1_5	入力	シリアルデータ入力 RY_R8C38 ボードの CN2 の 3 ピンに接続されています。
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	O				O			

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
Ⅰ	00	U0BRG＝1,250,000　／設定したいビットレート－1
Ⅱ	01	U0BRG＝　156,250　／設定したいビットレート－1
Ⅲ	10	U0BRG＝　 39,062.5／設定したいビットレート－1
Ⅳ	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 Ⅱで再計算
・Ⅱ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 Ⅲで再計算
・Ⅲ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 ≒ 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。UART0 送受信制御レジスタ 0(U0C0)の bit1,0 は"00"にしておきます。

```
        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			

26.6.2 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。

```
 95 :  /**********************************************************************/
 96 :  /* 1 文字受信                                                         */
 97 :  /* 引数    受信文字格納アドレス                                       */
 98 :  /* 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          */
 99 :  /**********************************************************************/
100 :  int get_uart0( unsigned char *s )
101 :  {
102 :      int ret = 0, i;
103 :      unsigned int data;
104 :
105 :      if (ri_u0c1 == 1){                  /* 受信データあり？           */
106 :          data = u0rb;
107 :          *s = (unsigned char)data;
108 :          ret = 1;
109 :          if( data & 0xf000 ) {           /* エラーあり？               */
110 :              /* エラー時は再設定 */
111 :              re_u0c1 = 0;
112 :              for( i=0; i<50; i++ );
113 :              re_u0c1 = 1;
114 :
115 :              ret = -1;
116 :          }
117 :      }
118 :      return ret;
119 :  }
```

(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。

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

```
102 :       int ret = 0, i;
103 :       unsigned int data;
```

102 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。
103 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。

```
105 :       if (ri_u0c1 == 1){                  /* 受信データあり？              */
106 :           data = u0rb;
107 :           *s = (unsigned char)data;
108 :           ret = 1;
```

105 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。
106 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。このデータを変数 data に代入します。 U0RB の内容を下記に示します。 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。

107 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。
108 行	変数 ret を 1 にして、戻り値を受信ありにします。

(4) 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。

```
109 :          if( data & 0xf000 ) {          /* エラーあり？                  */
110 :              /* エラー時は再設定 */
111 :              re_u0c1 = 0;
112 :              for( i=0; i<50; i++ );
113 :              re_u0c1 = 1;
114 :
115 :              ret = -1;
116 :          }
```

109 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、110 行以降を実行します。
111 行 ～113 行	エラーがあった場合、R8C/38A 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、111 行で RE_U0C1 を"0"、112 行でウエイトを入れ、113 行で RE_U0C1 を"1"にします。
115 行	戻り値をエラー(－1)にします。

(5) 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。

```
118 :      return ret;
119 :  }
```

26.6.3 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。

```
121 :  /************************************************************************/
122 :  /* 1文字出力                                                            */
123 :  /* 引数    送信データ                                                   */
124 :  /* 戻り値 0:送信中のため、送信できず 1:送信セット完了                   */
125 :  /************************************************************************/
126 :  int put_uart0( unsigned char r )
127 :  {
128 :      if(ti_u0c1 == 1) {                  /* 送信データなし？           */
129 :          u0tbl = r;
130 :          return 1;
131 :      } else {
132 :          /* 先に送信中(今回のデータは送信せずに終了) */
133 :          return 0;
134 :      }
135 :  }
```

(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_urat0 関数を実行します。

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

```
128 :      if(ti_u0c1 == 1) {                  /* 送信データなし？              */
129 :          u0tbl = r;
130 :          return 1;
```

128 行	「TI_U0C1」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。
129 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、TI_U0C1 が自動で"0"(データあり)になります。送信が完了すると、TI_U0C1 が自動で"1"(データなし)になります。
130 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

```
131 :      } else {
132 :          /* 先に送信中(今回のデータは送信せずに終了) */
133 :          return 0;
134 :      }
```

133 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。

26.6.4 main 関数

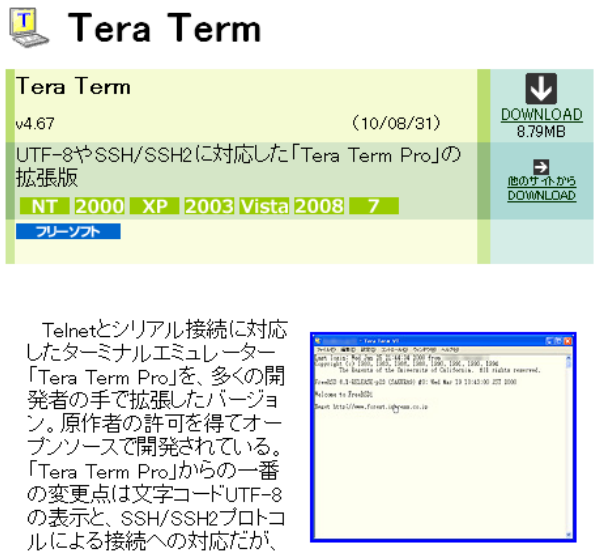
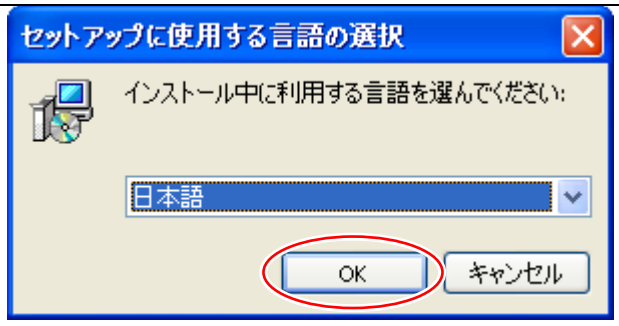
```
 39 :  void main( void )
 40 :  {
 41 :      unsigned char   d;
 42 :      int             ret;
 43 :
 44 :      init();                                 /* SFRの初期化                  */
 45 :
 46 :      while( 1 ) {
 47 :          ret = get_uart0( &d );
 48 :          p2 = d;
 49 :          if( ret == 1 ) put_uart0( d );
 50 :      }
 51 :  }
```

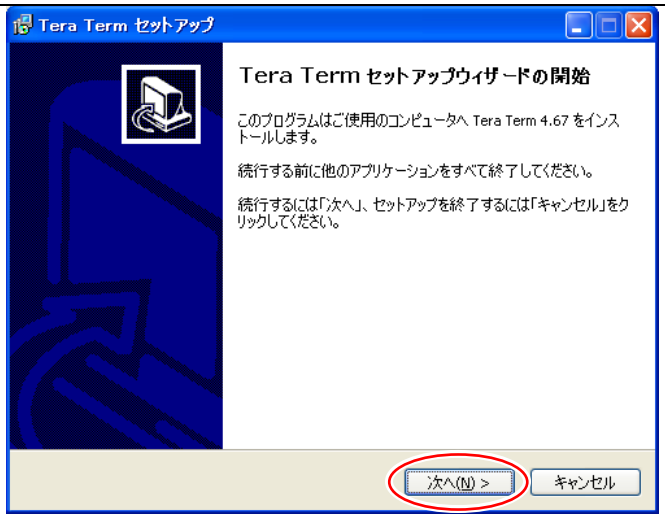
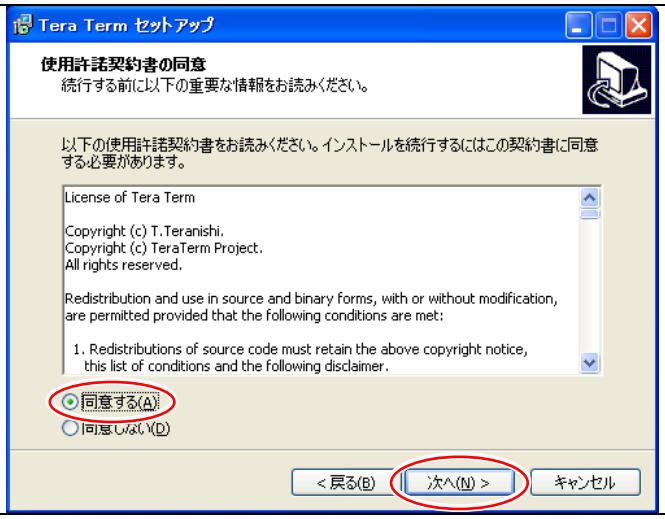
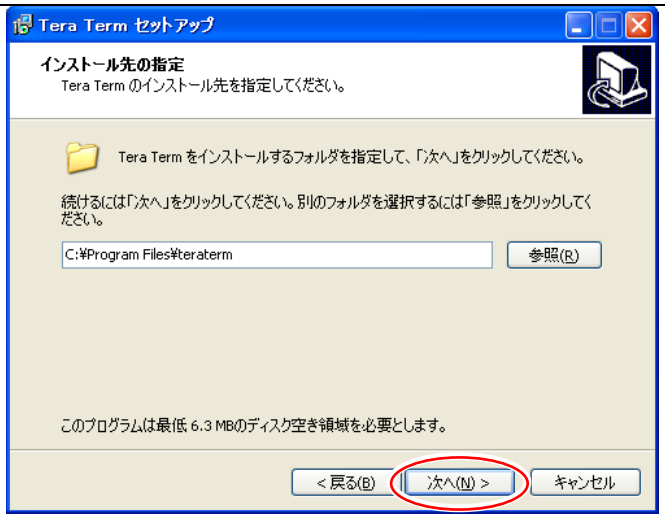
44 行	init 関数を実行します。 init 関数では、ポートの入出力設定、シリアルコミュニケーション 0(UART0)の設定を行っています。
47 行	get_uart0 関数で、パソコンから送られてきたデータを受信します。 受信結果は変数 ret に、受信文字は変数 d に代入します。
48 行	受信したデータをポート 2 に出力します。
49 行	変数 ret が 1 なら、すなわち受信データがあるなら、put_uart0 関数で受信したデータと同じデータをパソコンに送信します。

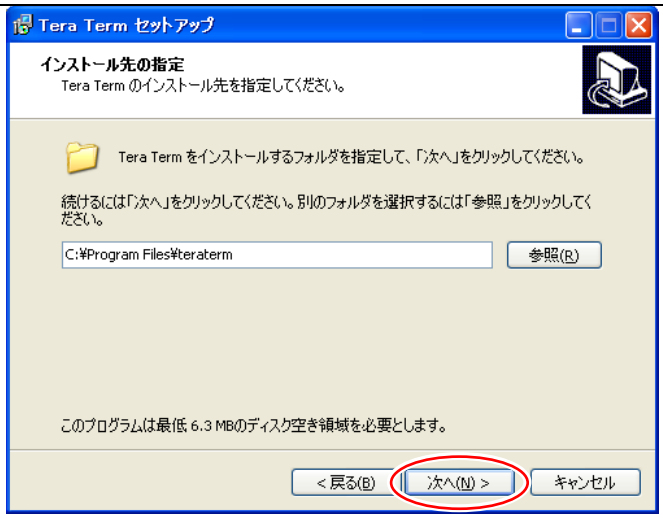
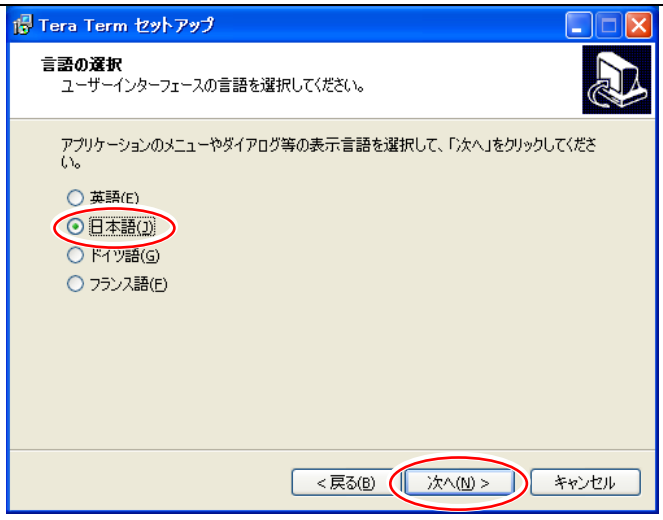
26.7　実習手順

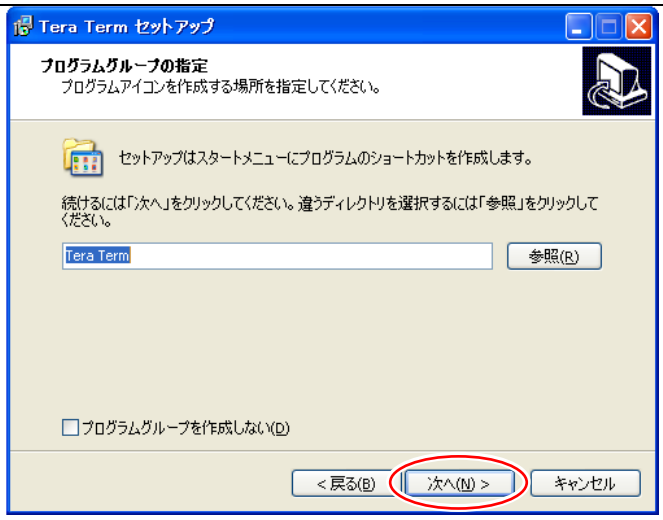
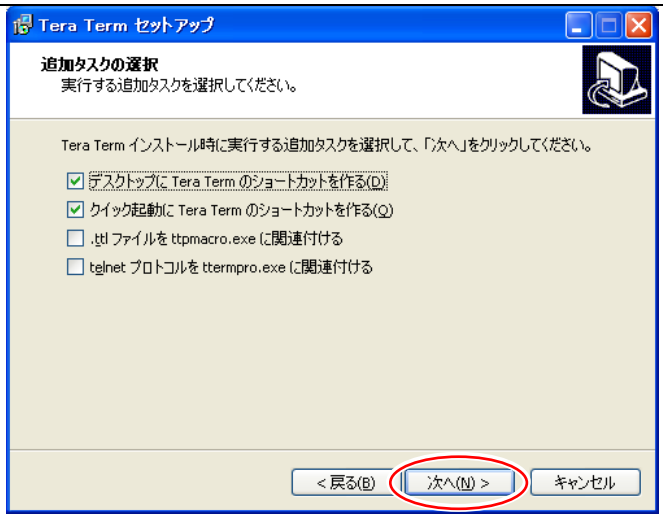
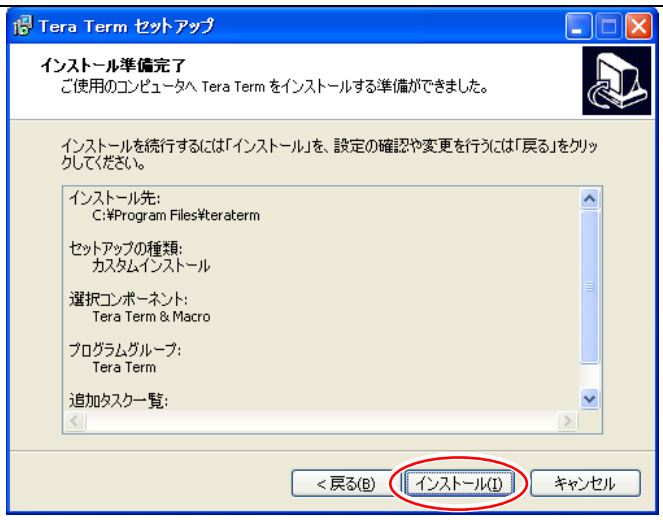
26.7.1　Tera Term のインストール

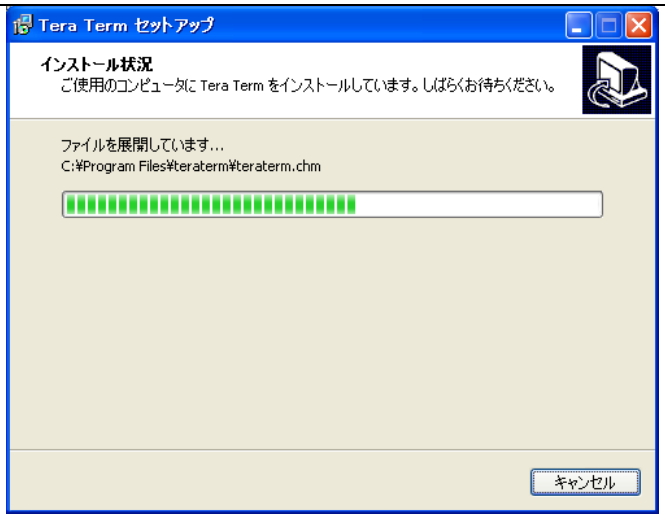
フリーソフトで通信のできる「**Tera Term**」というソフトを使用して、マイコンと通信をおこないます。ここではインストール手順を説明します。既に Tera Term をインストールしている場合は、インストールする必要はありません。

1		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。 ※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。
2		「DOWNLOAD」をクリックし、ファイルをダウンロードします。
3		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。
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。
7		次へをクリックします。

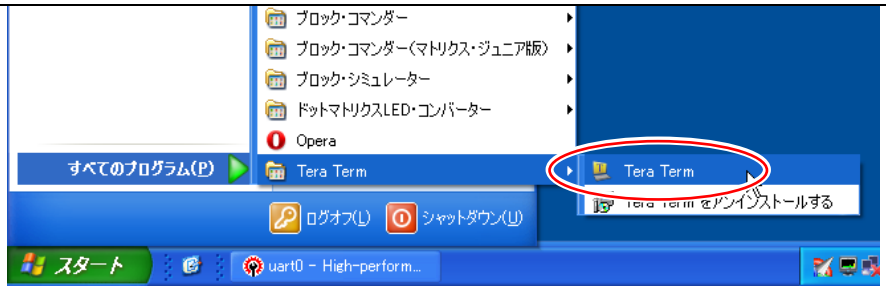
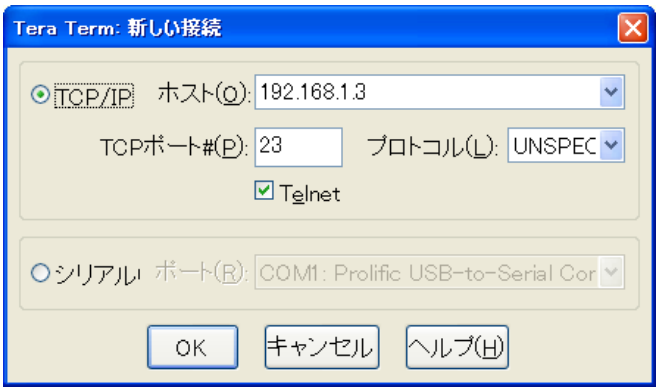
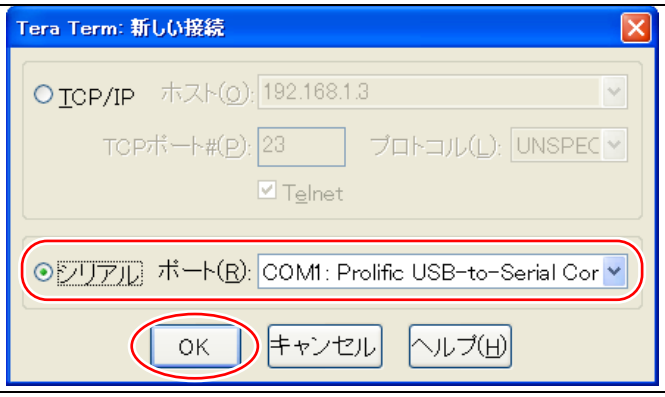
8		次へをクリックします。
9		コンポーネントの選択です。どのコンポーネントも使いませんので、すべてのチェックを外します。次へをクリックします。
10		「日本語」を選択して、次へをクリックします。

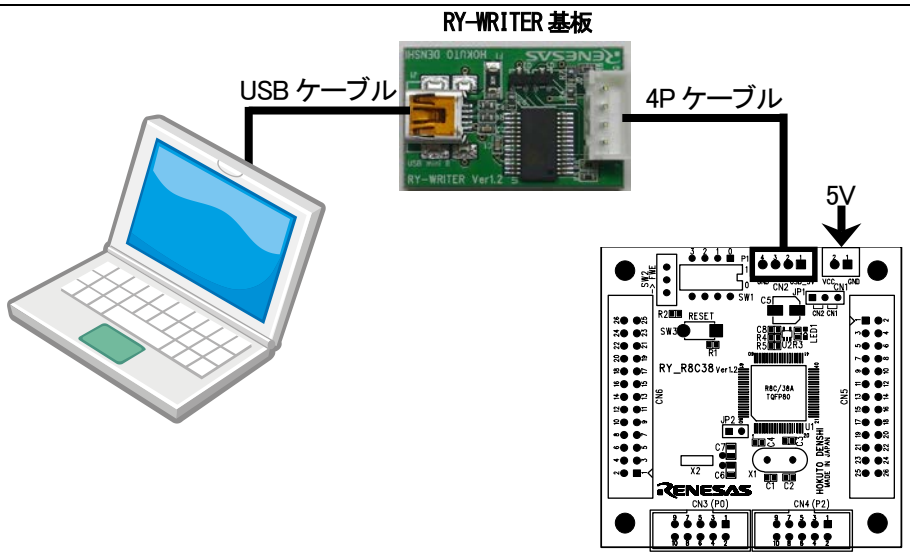
11		次へをクリックします。
12		追加するタスクを選択して（通常は変更なしで大丈夫です）、次へをクリックします。
13		インストールをクリックします。

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。

26.7.2　Tera Term を使って、マイコンと通信しよう

※あらかじめ「uart0.mot」をマイコンに書き込んでおきます。

1		「スタート」→「すべてのプログラム、またはプログラム」→「Tera Term」→「Tera Term」で Tera Term を立ち上げます。
2		最初に、接続先を確認する画面が出てきます。
3		「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。RY-WRITER 基板の場合は、「Prolific USB-to-Serial Comm 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。
4		立ち上がりました。

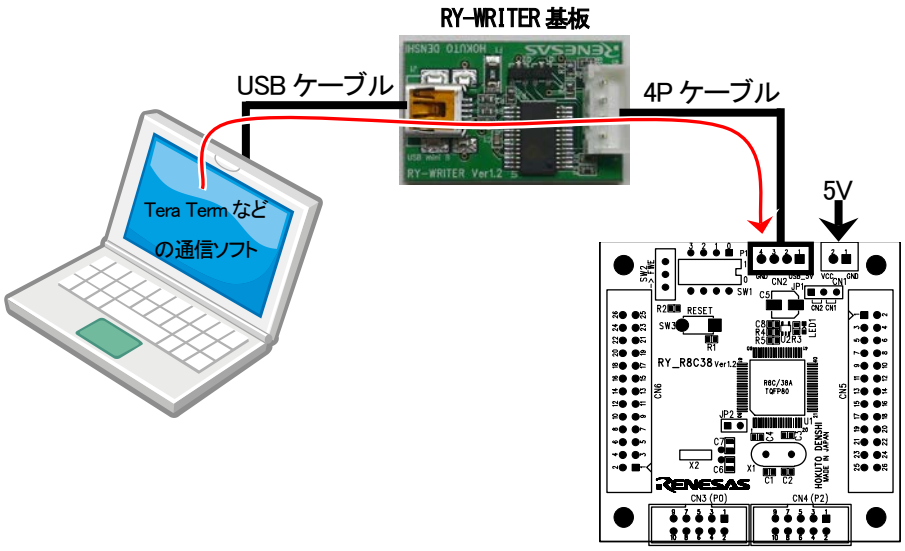
5		パソコンと RY_R8C38 ボードの間に RY-WRITER 基板（USB-TTL 変換基板）を入れて、ボードの電源を入れます。
6		これで準備が整いました。 a キーを押してみてください。 Tera Term のウィンドウ内に「a」が表示されます。 もし、表示されない場合は、 ・マイコンボードと USB 接続できていない ・COM ポートの番号が合っていない ・Tera Term の設定が合っていない などの理由が考えられます。もう一度、手順を確認してください。

26.7.3　Tera Term 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。Tera Term 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 RY_R8C38 ボードで受信

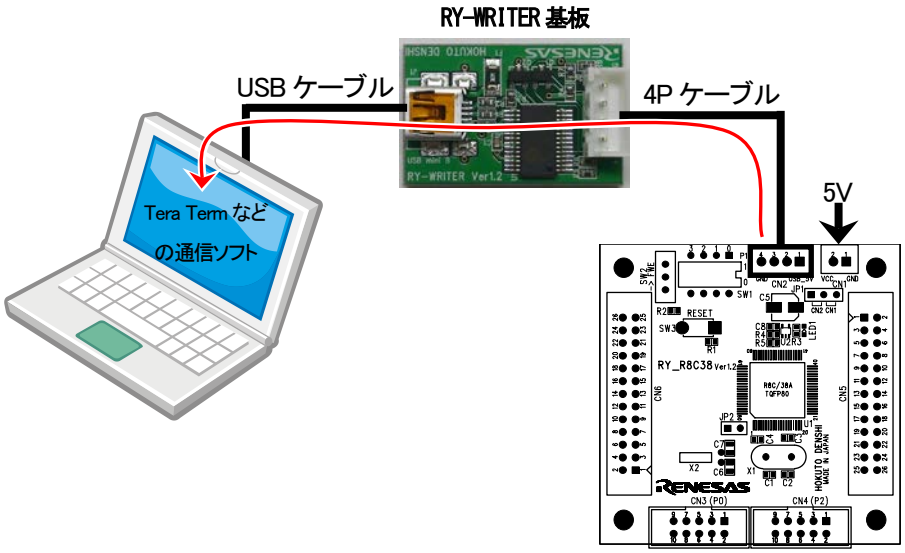
通信ソフト（Tera Term など）は、キーボードに入力された文字を、COM ポートを通じて RY_R8C38 ボードへ送ります。その様子を下図に示します。



キーボード→通信ソフト→COM ポート(USB)→RY_R8C38 ボードへ

(2) RY_R8C38 ボードから送られたデータを、パソコンに表示

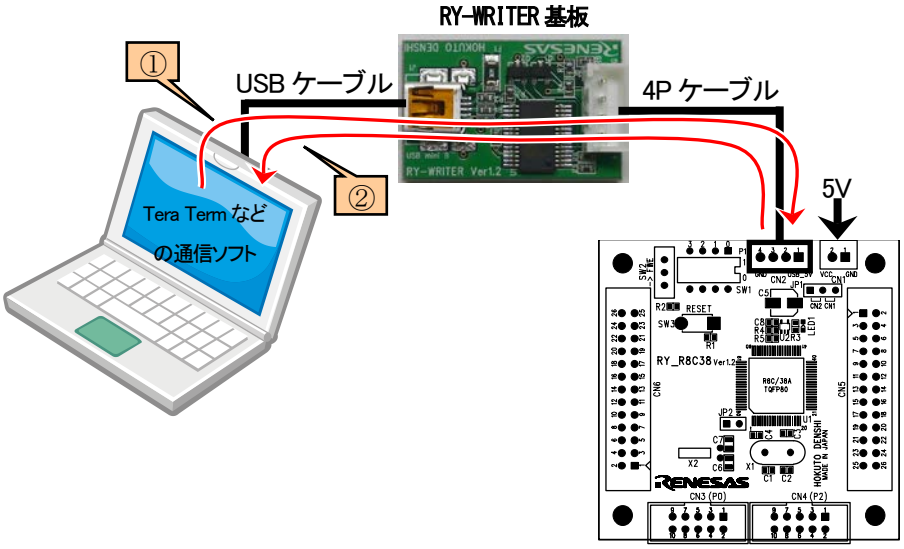
通信ソフト（Tera Term など）は、RY_R8C38 ボードから送られてきた文字データを、COM ポートを通じて画面に表示します。その様子を下図に示します。



RY_R8C38 ボード→COM ポート(USB)→通信ソフト→ディスプレイへ

(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？

26.8　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、Tera Term 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
    }
}
```

27.　printf、scanf 文を使った通信（プロジェクト：uart0_printf）

27.1　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。

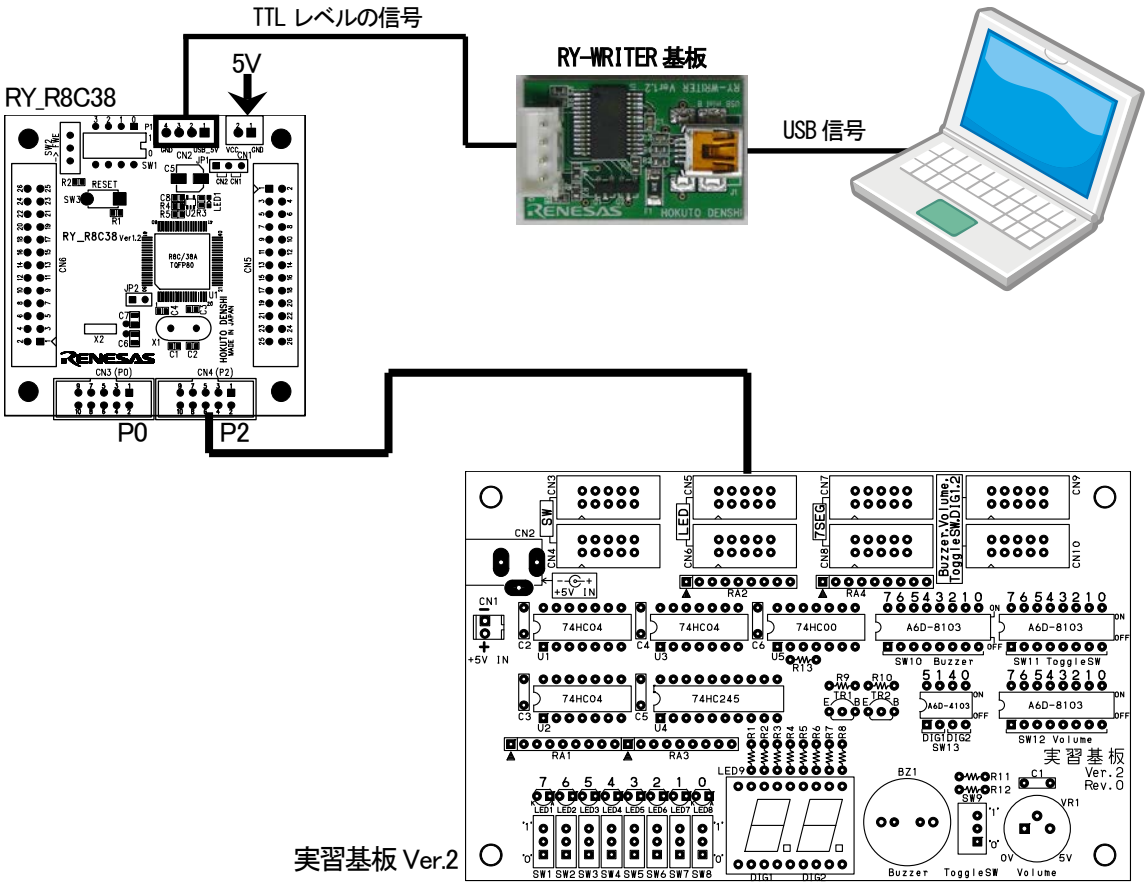
27.2　接続

■使用ポート

マイコンのポート	接続内容
4 ピンコネクタ（CN2）	CN2 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マイコンで受信し、その値を LED へ出力します。
P2（CN4）	実習基板 Ver.2 の LED 部など、出力機器を接続します。

■接続例

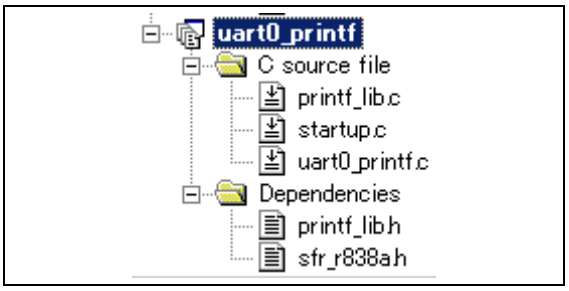
実習基板 Ver.2 を使ったときの接続例を下記に示します。



■操作方法

パソコン側は Tera Term 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。

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。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥uart0_printf¥startup.c
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/38A の内蔵周辺機能(SFR)の初期化も行います。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥uart0_printf¥uart0_printf.c
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。** ファイルの位置→C:¥Workspace¥common_r8c38a¥printf_lib.c
4	sfr_r838a.h	R8C/38A マイコンの内蔵周辺機能を制御するためのレジスタ（Special Function Registers)を定義したファイルです。 ファイルの位置→C:¥Workspace¥common_r8c38a¥sfr_r838a.h

27.4 プログラム「uart0_printf.c」

```
/************************************************************************/
/* 対象マイコン R8C/38A                                                 */
/* ﾌｧｲﾙ内容     printf文、scanf文の使用例                               */
/* バージョン   Ver.1.00                                                */
/* Date         2010.04.01                                              */
/* Copyright    ジャパンマイコンカーラリー実行委員会                    */
/************************************************************************/
/*
入力：UART0(パソコンのキーボードで入力したデータ)
      ※パソコンはTeraTermProなどの通信ソフトを使用します
出力：P2_7-P2_0(実習基板Ver.2のLED部)

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
ポート2に接続した実習基板Ver.2のLED部へ出力します。
C言語でおなじみのprintf文、scanf文を使用した演習です。
*/

/*=====================================*/
/* インクルード                        */
/*=====================================*/
#include <stdio.h>
#include "sfr_r838a.h"                  /* R8C/38A SFRの定義ファイル    */
#include "printf_lib.h"                 /* printf使用ライブラリ         */

```

```
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 :
34 :  /************************************************************************/
35 :  /* メインプログラム                                                     */
36 :  /************************************************************************/
37 :  void main( void )
38 :  {
39 :      int     i, ret;
40 :
41 :      init();                             /* SFRの初期化                  */
42 :      init_uart0_printf( SPEED_9600 );    /* UART0とprintf関連の初期化    */
43 :      asm(" fset I ");                    /* 全体の割り込み許可           */
44 :
45 :      printf( "Hello World!¥n" );
46 :
47 :      while( 1 ) {
48 :          printf( "Input data : " );
49 :          ret = scanf( "%d", &i );
50 :          if( ret == 1 ) {
51 :              printf( "Get data : %d¥n", i );
52 :              p2 = i;
53 :          } else {
54 :              printf( "Data Error!!¥n" );
55 :              scanf( "%*[^¥n]" );
56 :          }
57 :      }
58 :  }
59 :
60 :  /************************************************************************/
61 :  /* R8C/38A 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 = 0x00;                         /*                              */
78 :      pd1 = 0xd0;                         /* 5:RXD0 4:TXD0 3-0:DIP SW     */
79 :      pd2 = 0xff;                         /* 7-0:LED                      */
80 :      pd3 = 0xff;                         /*                              */
81 :      p4  = 0x20;                         /* P4_5のLED:初期は点灯         */
82 :      pd4 = 0xb8;                         /* 7:XOUT 6:XIN 5:LED 2:VREF    */
83 :      pd5 = 0xff;                         /*                              */
84 :      pd6 = 0xff;                         /*                              */
85 :      pd7 = 0xff;                         /*                              */
86 :      pd8 = 0xff;                         /*                              */
87 :      pd9 = 0x3f;                         /*                              */
88 :      pur0 = 0x04;                        /* P1_3～P1_0のプルアップON     */
89 :  }
90 :
91 :  /************************************************************************/
92 :  /* end of file                                                          */
93 :  /************************************************************************/
```

27.5　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。

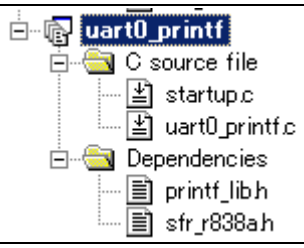
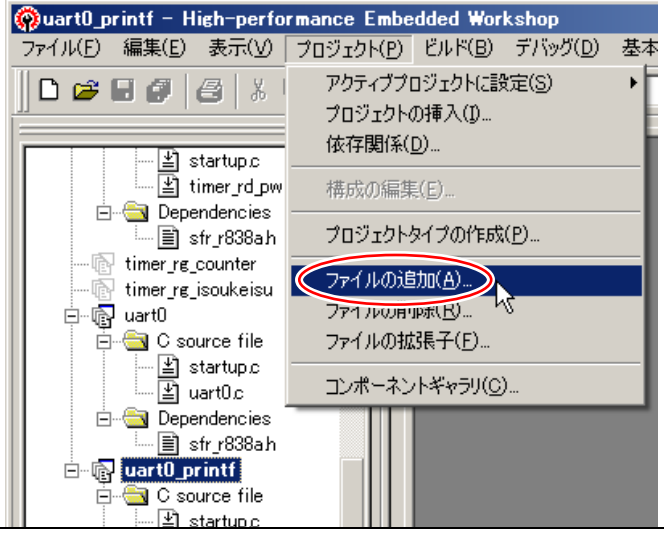
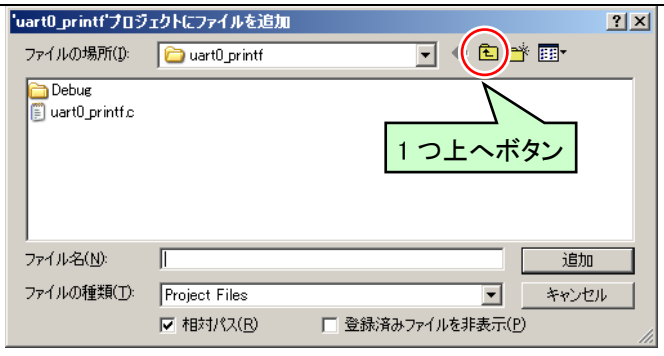
RY_R8C38 ボードにプログラムを書き込むとき、RY_R8C38 ボードとパソコンの間には、RY-WRITER 基板などの USB-シリアル変換基板を接続します。パソコンは RY_R8C38 ボードを COM ポートとして認識します。この COM ポートを通して、パソコンの画面やキーボードをあたかも RY_R8C38 ボードの装置であるかのごとく使用することができます。ただ、パソコンの画面やキーボードは RY_R8C38 ボードとパソコンを USB ケーブルで接続するだけでは使用できず、「ハイパーターミナル」や「Tera Term」などの通信ソフトを経由して使用します。

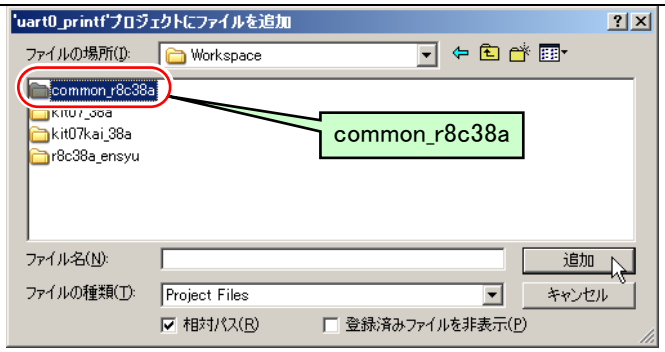
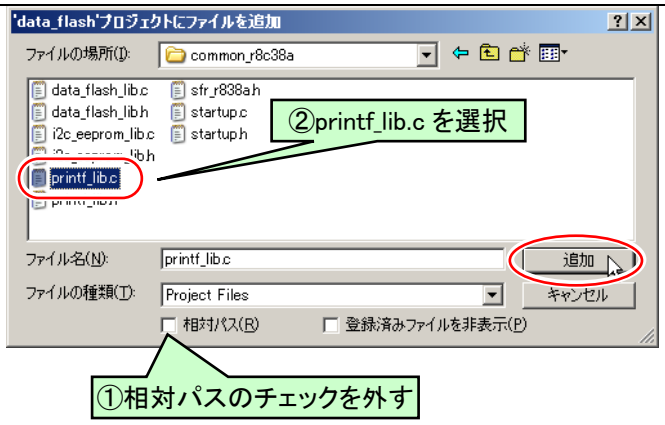
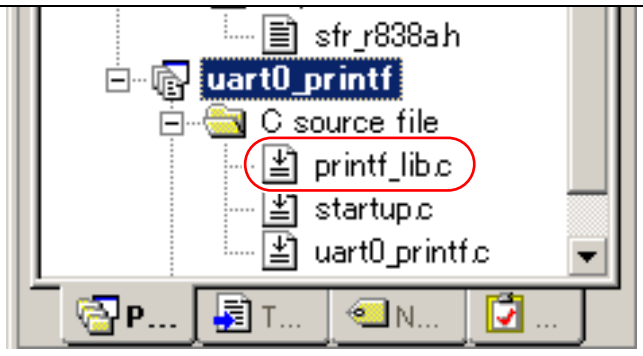
※printf 関数、scanf 関数は多くのメモリを消費するため、R8C マイコンでは「%e, %E, %f, %g, %G」の変換指定文字は使用できません。

27.6 プログラムの解説「printf_lib.c」

「printf_lib.c」は、R8C/38A 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。

27.6.1 「printf_lib.c」の追加

1		「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）。
2		「プロジェクト→ファイルの追加」を選択します。
3		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。

4		C ドライブ ↓ Workspace ↓ common_r8c38a フォルダを開きます。
5		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。
6		「printf_lib.c」が追加されました。

27.6.2 関数一覧

「printf_lib.c」ファイルを追加して、追加された関数を下記に示します。

■init_uart0_printf 関数

書式	void init_uart0_printf(int sp)
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	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

■printf 関数

書式	int printf(const char *format, ...);
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	printf("i は、%d です。", i);　// i の値を表示
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。 #include　<stdio.h>

■scanf 関数

書式	int scanf(const char *format, ...);
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	scanf("%d, &i);　// i に 10 進数を読み込む
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。 #include　<stdio.h>

27.7 プログラムの解説「uart0_printf.c」

27.7.1 インクルード部分

pirntf 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。

```
18 :  /*====================================*/
19 :  /* インクルード                       */
20 :  /*====================================*/
21 :  #include <stdio.h>
22 :  #include "sfr_r838a.h"                /* R8C/38A SFRの定義ファイル     */
23 :  #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_r8c38a」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。**

27.7.2 main 関数（初期化部分）

```
34 :  /*****************************************************************************/
35 :  /* メインプログラム                                                          */
36 :  /*****************************************************************************/
37 :  void main( void )
38 :  {
39 :      int     i, ret;
40 :
41 :      init();                          /* SFRの初期化                  */
42 :      init_uart0_printf( SPEED_9600 );  /* UART0とprintf関連の初期化    */
43 :      asm(" fset I ");                 /* 全体の割り込み許可           */
```

41 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**。
42 行	init_uart0_printf 関数で、UART0 の初期化と pirntf 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。
43 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。

27.7.3 main 関数（通信部分）

```
 45 :      printf( "Hello World!¥n" );
 46 :
 47 :      while( 1 ) {
 48 :          printf( "Input data : " );
 49 :          ret = scanf( "%d", &i );
 50 :          if( ret == 1 ) {
 51 :              printf( "Get data : %d¥n", i );
 52 :              p2 = i;
 53 :          } else {
 54 :              printf( "Data Error!!¥n" );
 55 :              scanf( "%*[^¥n]" );
 56 :          }
 57 :      }
 58 :  }
```

45 行	最初に「Hello World!」と表示します。
48 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。
49 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。
50 行	scanf 関数の戻り値をチェックします。
51,52 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 2 へ出力します。
54,55 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。

27.7.4 受信バッファのクリア

プログラムの 55 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のようになります。

```
 %   *  [^ ¥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。

まとめると、「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」となります。「¥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、

```
 49 :          ret = scanf( "%d", &i );
```

ここで、「aaa(改行)」を入力したとします（改行＝エンタキー）。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、数値入力しようと、

```
        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。

```
 50 :          if( ret == 1 ) {
 51 :              printf( "Get data : %d¥n", i );
 52 :              p2 = 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 2 へその値を出力します。

```
 53 :          } else {
 54 :              printf( "Data Error!!¥n" );
 55 :              scanf( "%*[^¥n]" );
 56 :          }
```

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。**

28. データフラッシュ（プロジェクト：data_flash）

28.1 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。

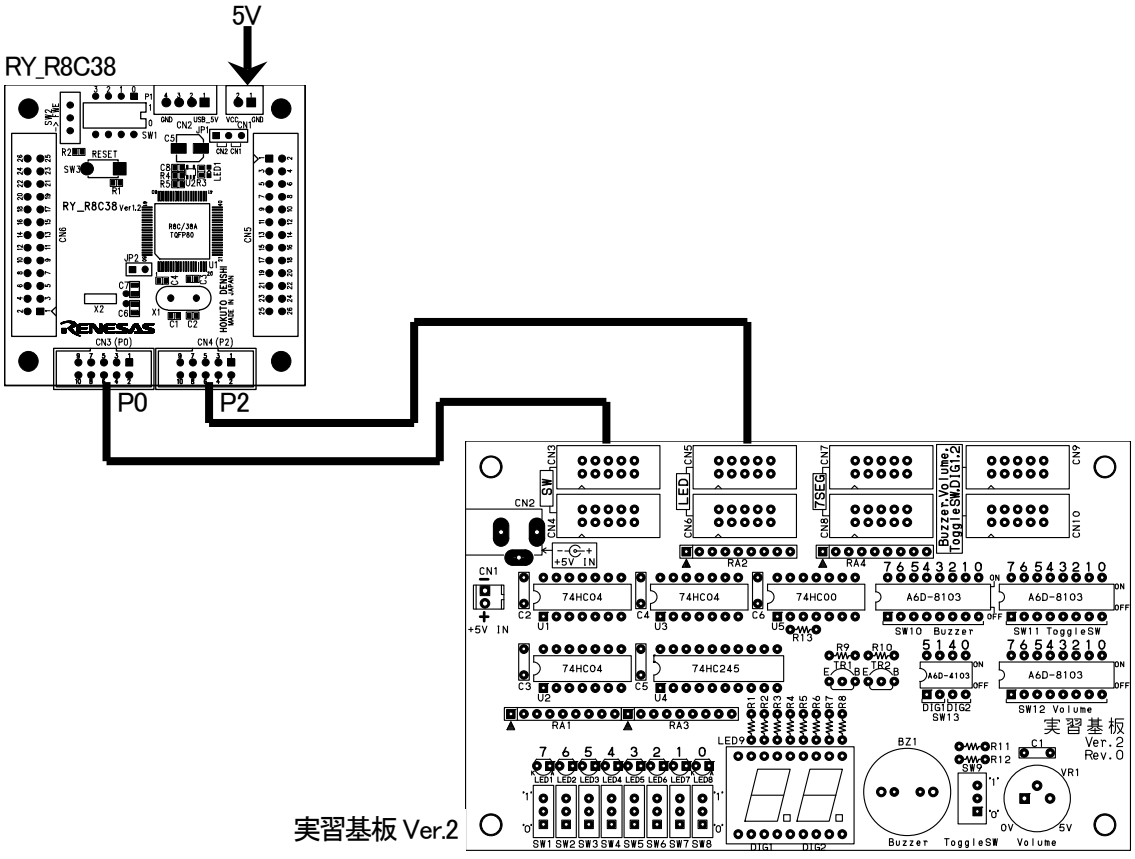
28.2 接続

■使用ポート

マイコンのポート	接続内容
P1_3、P1_2、P1_1、P1_0	RY_R8C38 ボードのディップスイッチです。
P0 (CN3)	実習基板 Ver.2 のスライドスイッチ部など、入力機器を接続します。
P2 (CN4)	実習基板 Ver.2 の LED 部など、出力機器を接続します。

■接続例

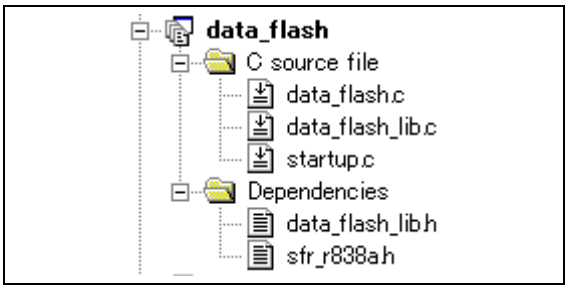
実習基板 Ver.2 を使ったときの接続例を下記に示します。



■操作方法

「28.8 実習手順」を参照してください。

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。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥data_flash¥startup.c
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/38A の内蔵周辺機能(SFR)の初期化も行います。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥data_flash¥data_flash.c
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。** ファイルの位置→C:¥Workspace¥common_r8c38a¥data_flash_lib.c
4	sfr_r838a.h	R8C/38A マイコンの内蔵周辺機能を制御するためのレジスタ（Special Function Registers）を定義したファイルです。 ファイルの位置→C:¥Workspace¥common_r8c38a¥sfr_r838a.h

28.4 プログラム「data_flash.c」

```
 1 :  /***********************************************************************/
 2 :  /* 対象マイコン R8C/38A                                                */
 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.1.02                                               */
 5 :  /* Date         2011.03.18                                             */
 6 :  /* Copyright    ジャパンマイコンカーラリー実行委員会                   */
 7 :  /***********************************************************************/
 8 :  /*
 9 :  入力：P0_7-P0_0(実習基板Ver.2のスライドスイッチ部)
10 :  出力：P2_7-P2_0(実習基板Ver.2のLED部)
11 :
12 :  データフラッシュに書き込んでいる値を実習基板Ver.2のLED部に出力します。
13 :  マイコンボード上のディップスイッチの値を変えると、
14 :  実習基板Ver.2のスライドスイッチの値をデータフラッシュに保存します。
15 :  */
16 :
17 :  /*======================================*/
18 :  /* インクルード                         */
19 :  /*======================================*/
20 :  #include "sfr_r838a.h"                  /* R8C/38A SFRの定義ファイル    */
21 :  #include "data_flash_lib.h"             /* データフラッシュライブラリ   */
22 :
```

```
 23 :  /*====================================*/
 24 :  /* シンボル定義                       */
 25 :  /*====================================*/
 26 :  /* データフラッシュ関連 */
 27 :  #define     DF_SIZE             256     /* 読み書きサイズ               */
 28 :
 29 :  #define     DF_CHECK            0       /* データフラッシュチェック     */
 30 :  #define     DF_DATA             1       /* データ                       */
 31 :
 32 :  /*====================================*/
 33 :  /* プロトタイプ宣言                   */
 34 :  /*====================================*/
 35 :  void init( void );
 36 :  unsigned char dipsw_get( void );
 37 :  void timer( unsigned long timer_set );
 38 :
 39 :  /*====================================*/
 40 :  /* グローバル変数の宣言               */
 41 :  /*====================================*/
 42 :  /* データフラッシュ関連 */
 43 :  signed char data_buff[ DF_SIZE ];       /* 一時保存エリア               */
 44 :
 45 :  /************************************************************************/
 46 :  /* メインプログラム                                                     */
 47 :  /************************************************************************/
 48 :  void main( void )
 49 :  {
 50 :      int             ret;
 51 :      unsigned char   c;
 52 :
 53 :      init();                             /* 初期化                       */
 54 :
 55 :      readDataFlash( 0x3000, data_buff, DF_SIZE );
 56 :
 57 :      if( data_buff[DF_CHECK] != 0x38 ) {
 58 :          data_buff[DF_CHECK] = 0x38;
 59 :          data_buff[DF_DATA]  = 0;
 60 :      }
 61 :
 62 :      p2 = data_buff[DF_DATA];
 63 :
 64 :      c = dipsw_get();
 65 :      while( c == dipsw_get() );
 66 :
 67 :      data_buff[DF_DATA]  = p0;
 68 :      blockEraseDataFlash( 0x3000 );
 69 :      ret = writeDataFlash( 0x3000, data_buff, DF_SIZE );
 70 :      if( ret == 0 ) {
 71 :          /* 書き込みエラーなら */
 72 :          while( 1 ) {
 73 :              p2 = 0xf0;
 74 :              timer( 100 );
 75 :              p2 = 0x0f;
 76 :              timer( 100 );
 77 :          }
 78 :      }
 79 :
 80 :      /* 書き込みが正常にできたなら */
 81 :      while( 1 ) {
 82 :          p2 = 0x55;
 83 :          timer( 200 );
 84 :          p2 = 0xaa;
 85 :          timer( 200 );
 86 :      }
 87 :  }
 88 :
 89 :  /************************************************************************/
 90 :  /* R8C/38A スペシャルファンクションレジスタ(SFR)の初期化                */
 91 :  /************************************************************************/
 92 :  void init( void )
 93 :  {
 94 :      int i;
 95 :
 96 :      /* クロックをXINクロック(20MHz)に変更 */
 97 :      prc0  = 1;                          /* プロテクト解除               */
 98 :      cm13  = 1;                          /* P4_6,P4_7をXIN-XOUT端子にする*/
 99 :      cm05  = 0;                          /* XINクロック発振              */
100 :      for(i=0; i<50; i++ );               /* 安定するまで少し待つ(約10ms) */
101 :      ocd2  = 0;                          /* システムクロックをXINにする  */
102 :      prc0  = 0;                          /* プロテクトON                 */
103 :
104 :      /* ポートの入出力設定 */
105 :      prc2 = 1;                           /* PD0のプロテクト解除          */
106 :      pd0 = 0x00;                         /* 7-0:スイッチ                 */
107 :      pd1 = 0xd0;                         /* 5:RXD0 4:TXD0 3-0:DIP SW     */
108 :      pd2 = 0xff;                         /* 7-0:LED                      */
109 :      pd3 = 0xff;                         /*                              */
110 :      p4  = 0x20;                         /* P4_5のLED:初期は点灯         */
111 :      pd4 = 0xb8;                         /* 7:XOUT 6:XIN 5:LED 2:VREF    */
112 :      pd5 = 0xff;                         /*                              */
113 :      pd6 = 0xff;                         /*                              */
```

```
114 :      pd7 = 0xff;                         /*                              */
115 :      pd8 = 0xff;                         /*                              */
116 :      pd9 = 0x3f;                         /*                              */
117 :      pur0 = 0x04;                        /* P1_3～P1_0のプルアップON     */
118 :  }
119 :
120 :  /************************************************************************/
121 :  /* ディップスイッチ値読み込み                                           */
122 :  /* 戻り値 スイッチ値 0～15                                               */
123 :  /************************************************************************/
124 :  unsigned char dipsw_get( void )
125 :  {
126 :      unsigned char sw;
127 :
128 :      sw = p1 & 0x0f;                     /* P1_3～P1_0読み込み           */
129 :
130 :      return  sw;
131 :  }
132 :
133 :  /************************************************************************/
134 :  /* タイマ本体                                                           */
135 :  /* 引数　タイマ値 1=1ms                                                 */
136 :  /************************************************************************/
137 :  void timer( unsigned long timer_set )
138 :  {
139 :      int i;
140 :
141 :      do {
142 :          for( i=0; i<1240; i++ );
143 :      } while( timer_set-- );
144 :  }
145 :
146 :  /************************************************************************/
147 :  /* end of file                                                          */
148 :  /************************************************************************/
```

28.5 データフラッシュを使おう！

28.5.1 概要

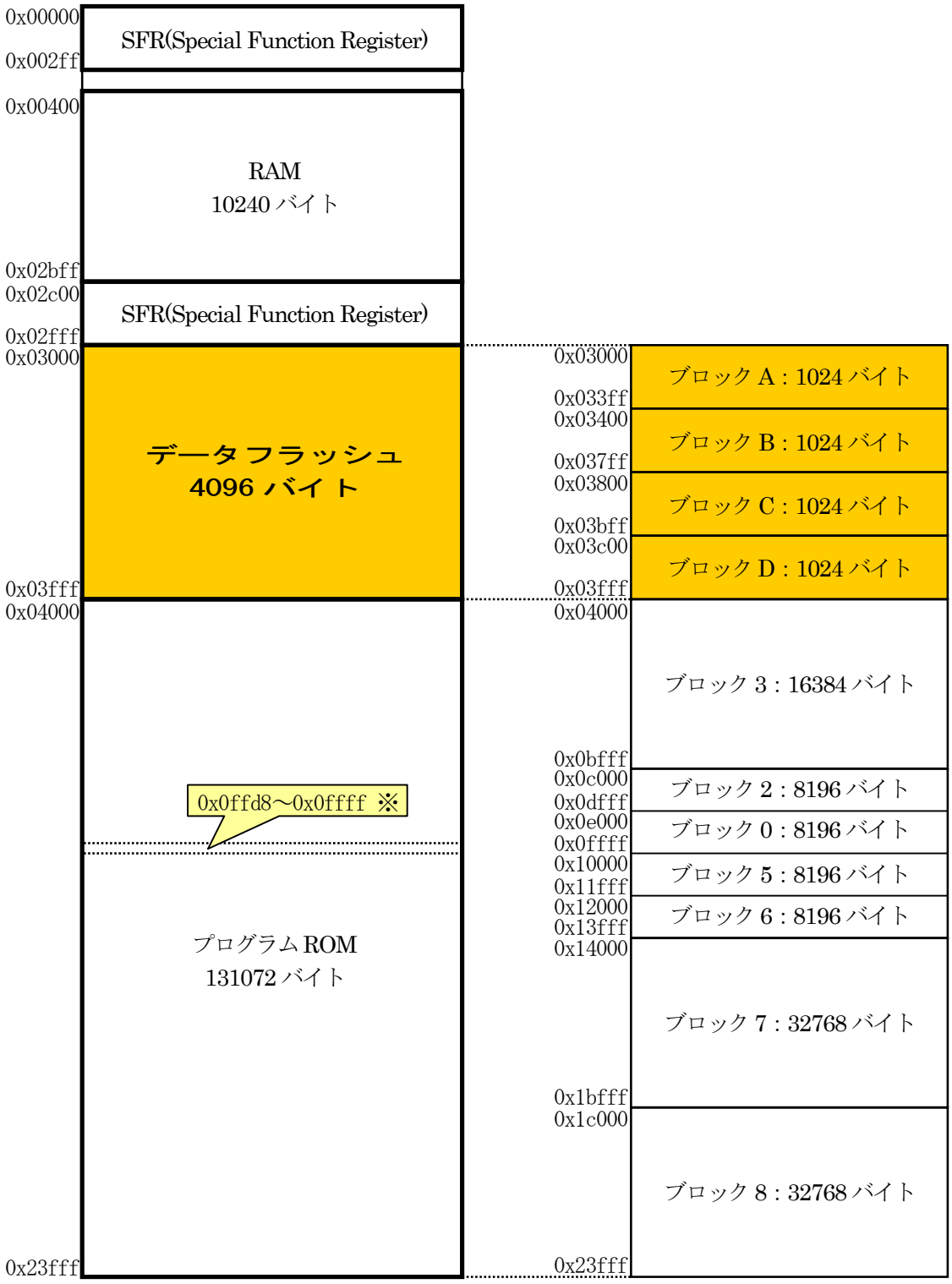
R8C/38A には、電気的に書き換え可能なフラッシュメモリが内蔵されています。フラッシュメモリは 2 種類あり、次のような特徴があります。

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	128KB(131,072 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 3:0x04000～0x0bfff 番地の 32KB ブロック 2:0x0c000～0x0dfff 番地の 8KB ブロック 0:0x0e000～0x0ffff 番地の 8KB ブロック 5:0x10000～0x11fff 番地の 8KB ブロック 6:0x12000～0x13fff 番地の 8KB ブロック 7:0x14000～0x1bfff 番地の 32KB ブロック 8:0x1c000～0x23fff 番地の 32KB	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。

28.5.2　データフラッシュのメモリマップ

データフラッシュは、0x03000～0x03fff 番地にあります。

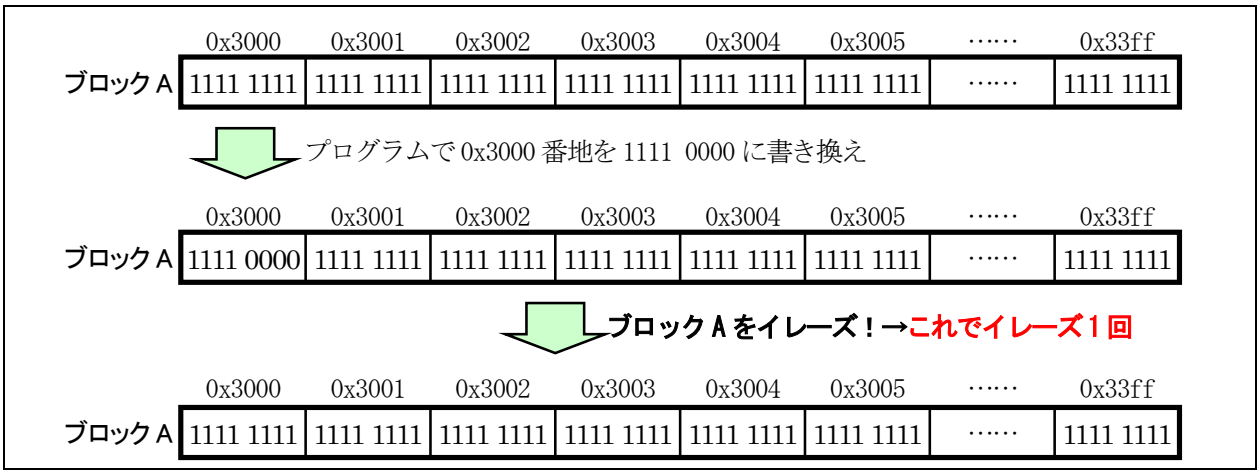


※0x0ffd8～0x0ffdb 番地は予約領域、0x0ffdc～0x0ffff 番地は固定割り込みベクタテーブルの領域のため、プログラムは置けません。

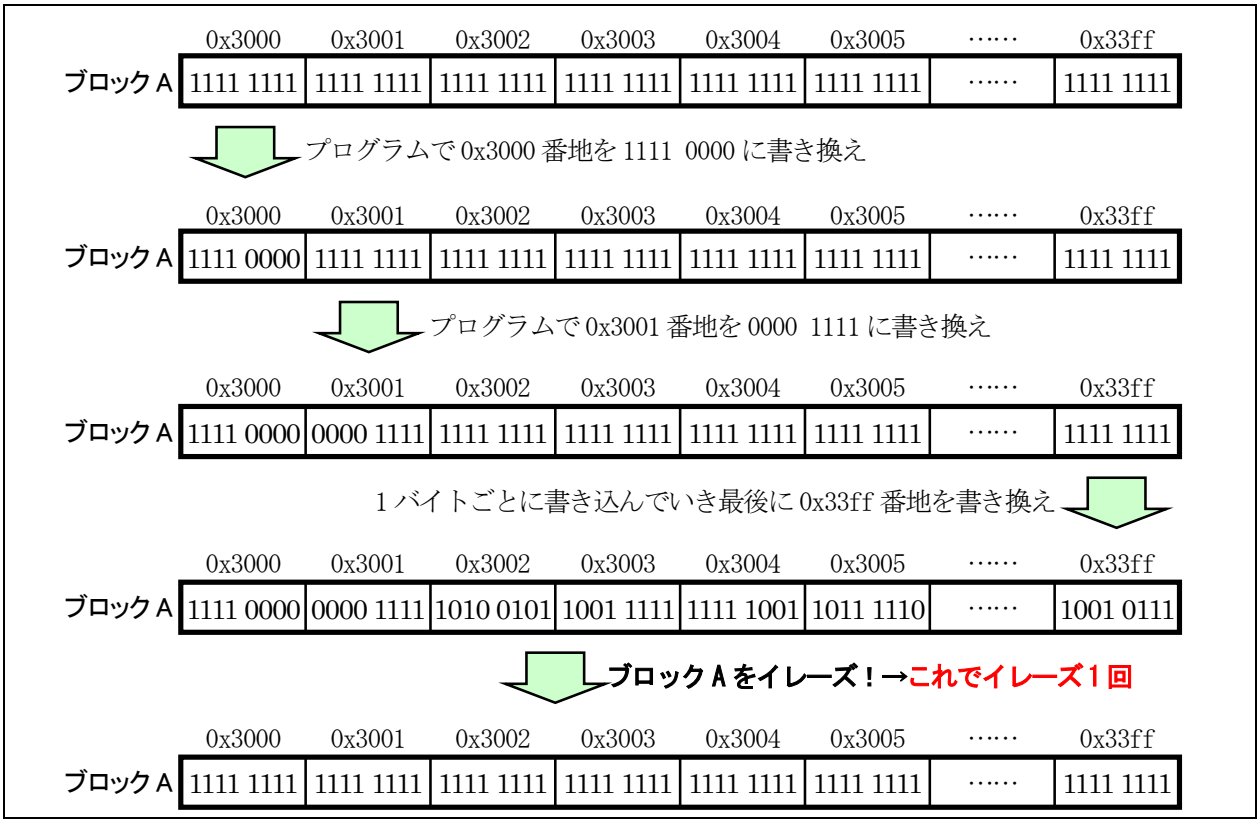
28.5.3 データフラッシュを効率的に使う

データフラッシュの初期値は 0xff(1111 1111_2)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がイレーズ回数は 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）。

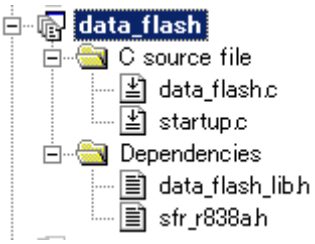
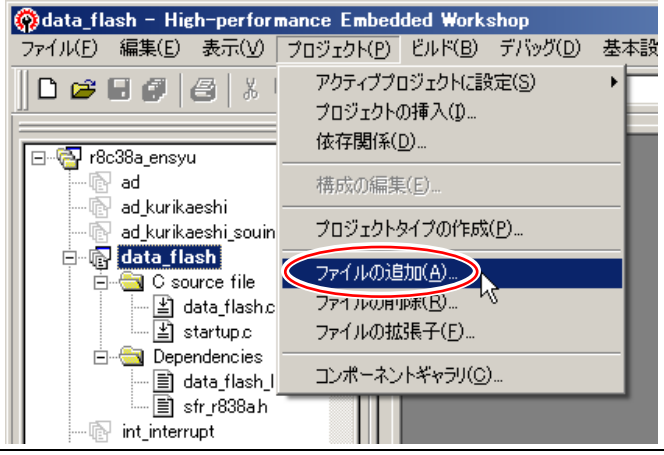
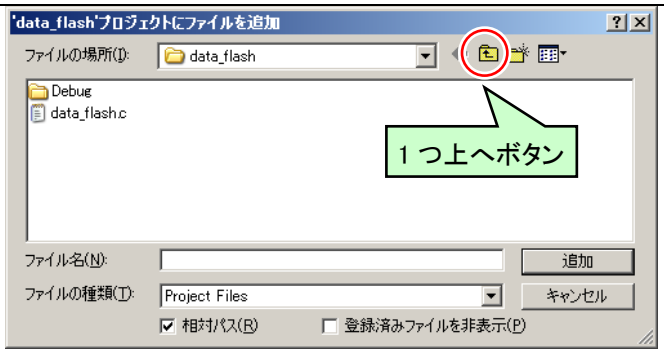
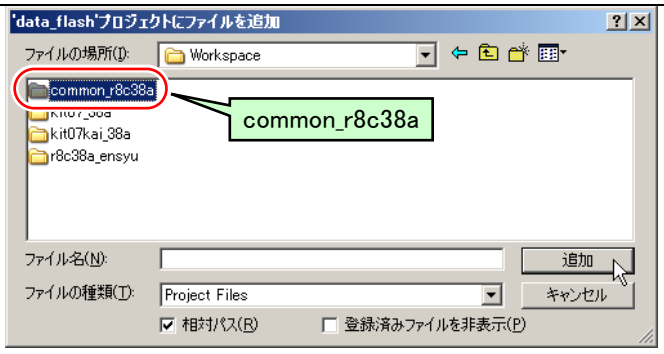


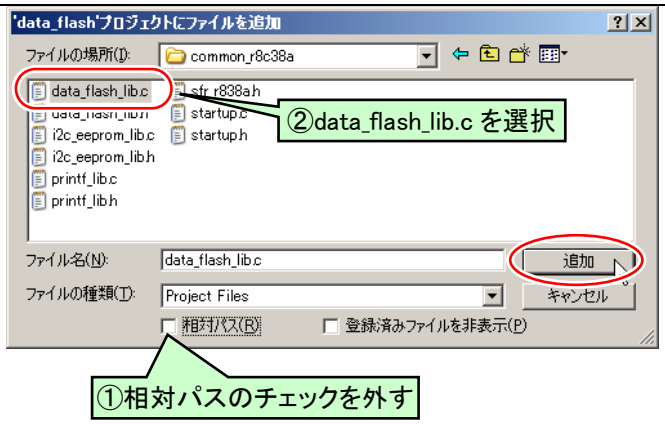
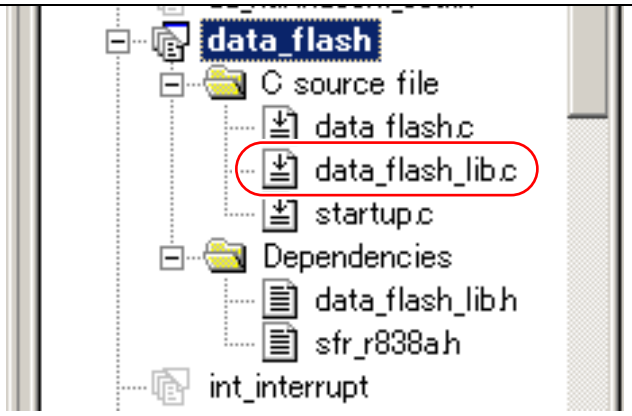
このように、イレーズ回数を減らすことによってデータフラッシュを効率的（長く）に使うことが出来ます。プログラムでは、1 ブロック使い切るまで書き込んでいき、使い切った時点でイレーズするようにします。

28.6 プログラムの解説「data_flash_lib.c」

「data_flash_lib.c」は、R8C/38A 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。

28.6.1 「data_flash_lib.c」の追加

1		「data_flash_lib.c」は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)。
2		「プロジェクト→ファイルの追加」を選択します。
3		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。
4		C ドライブ ↓ Workspace ↓ common_r8c38a フォルダを開きます。

5		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。
6		「data_flash_lib.c」が追加されました。

28.6.2　関数一覧

「data_flash_lib.c」ファイルを追加して、追加された関数を下記に示します。

■readDataFlash 関数

書式	void readDataFlash(unsigned int r_address, signed char *w_address, int count)
内容	データフラッシュから、データを読み込みます。
引数	unsigned int　　読み込み元アドレス 0x3000～0x3fff signed char*　　読み込み先アドレス int　　読み込むデータ数
戻り値	無し
使用例	signed char　data_buff[256];　　/* データフラッシュ用保管エリア */ readDataFlash(0x3000, data_buff, 256); /* 0x3000 番地から、data_buff 配列に、256 個のデータを読み込みます。 */

■writeDataFlash 関数

書式	int writeDataFlash(unsigned int w_address, signed char *r_address, int count)
内容	データフラッシュに、データを書き込みます。
引数	unsigned int　書き込み先アドレス 0x3000～0x3fff signed char*　書き込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	signed char　data_buff[256];　/* データフラッシュ用保管エリア */ writeDataFlash(0x3000, data_buff, 256); /* 0x3000 番地から、data_buff 配列のデータを、256 個書き込みます。 */
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 256 個のデータを書き込むとき、0x33ff 番地の次は、0x3400 番地になります。ブロックは、A から B に変わります。これはできません。 例) writeDataFlash(0x33f0, data_buff, 256); /* これはできない */

■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。

28.7 プログラムの解説「data_flash.c」

28.7.1 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。

```
 17 :  /*====================================*/
 18 :  /* インクルード                       */
 19 :  /*====================================*/
 20 :  #include "sfr_r838a.h"                /* R8C/38A SFRの定義ファイル    */
 21 :  #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_r8c38a」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。**

28.7.2 変数領域

```
 23 :  /*====================================*/
 24 :  /* シンボル定義                       */
 25 :  /*====================================*/
 26 :  /* データフラッシュ関連 */
 27 :  #define     DF_SIZE             256     /* 読み書きサイズ               */
 28 :
 29 :  #define     DF_CHECK            0       /* データフラッシュチェック     */
 30 :  #define     DF_DATA             1       /* データ                       */
中略
 39 :  /*====================================*/
 40 :  /* グローバル変数の宣言               */
 41 :  /*====================================*/
 42 :  /* データフラッシュ関連 */
 43 :  signed char data_buff[ DF_SIZE ];       /* 一時保存エリア               */
```

27 行～30 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 256 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。256 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)。

43 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 256 個になります。

28.7.3 main 関数（データフラッシュから読み込み）

```
48 :  void main( void )
49 :  {
50 :      int             ret;
51 :      unsigned char   c;
52 :
53 :      init();                             /* 初期化                       */
54 :
55 :      readDataFlash( 0x3000, data_buff, DF_SIZE );
56 :
57 :      if( data_buff[DF_CHECK] != 0x38 ) {
58 :          data_buff[DF_CHECK] = 0x38;
59 :          data_buff[DF_DATA]  = 0;
60 :      }
61 :
62 :      p2 = data_buff[DF_DATA];
63 :
64 :      c = dipsw_get();
65 :      while( c == dipsw_get() );
```

53 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。
55 行	readDataFlash 関数で、 ・0x3000 番地から ・data_buff 配列に ・DF_SIZE 個(256 個)のデータを 読み込みます。
57 行	data_buff 配列の DF_CHECK 番目(0 番目)のデータが 0x38 かチェックします。
58,59 行	57 行のチェックで、0x38 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 0x38 を ・DF_DATA 番目(1 番目)に 0 を 設定します。 ちなみに 0x38 なら、DF_DATA には前回保存したデータが読み込まれます。
62 行	ポート 2(実習基板 Ver.2 の LED など)に、前回保存したデータを出力します。もし、今回初めて 0x3000 番地を読み込んだ場合は、59 行目に設定している 0 を出力します(実習基板 Ver.2 の LED は点灯しません)。
64,65 行	RY_R8C38 ボードのディップスイッチを違う値にするまで、65 行を繰り返し続けます。

28.7.4 main 関数（データフラッシュのイレーズ、書き込み）

```
67 :      data_buff[DF_DATA]  = p0;
68 :      blockEraseDataFlash( 0x3000 );
69 :      ret = writeDataFlash( 0x3000, data_buff, DF_SIZE );
```

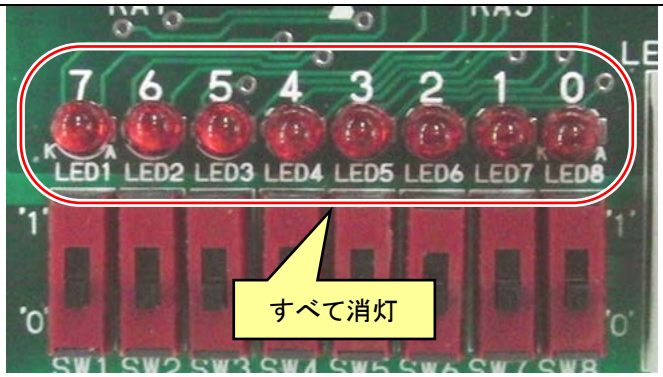
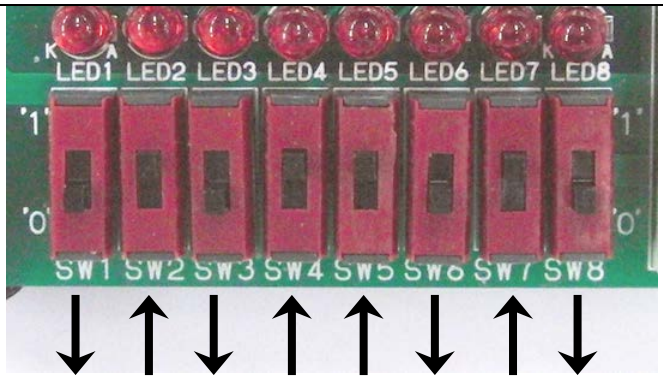
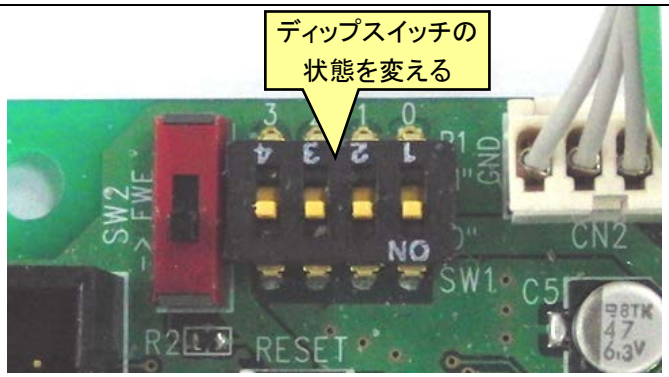
67 行	RY_R8C38 ボードのディップスイッチの値(P0 の値)を、data_buff 配列の DF_DATA 番目(1 番目)に保存します。
68 行	blockEraseDataFlash 関数で、0x3000 番地を含むブロックをイレーズします。 **注意点は、0x3000 番地のブロックであるブロック A のデータをすべてイレーズすることです。**ブロック A は 0x3000～0x33ff 番地なので、このアドレスのデータがイレーズされます。この範囲のアドレスで消したくないデータは、内蔵 RAM に保存しておきます。ただし、内蔵 RAM は 10KB しかありませんので、RAM の容量がオーバーしないよう気をつけてください。
69 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(256 個) 書き込みます。 書き込みが正常にできれは変数 ret には 1 が、書き込みできなければ 0 が入ります。

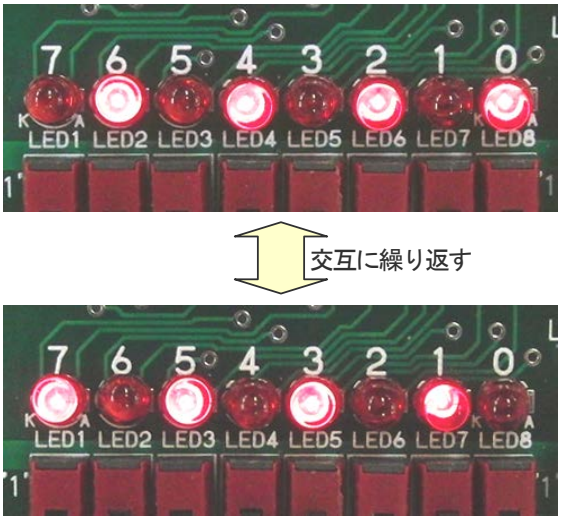
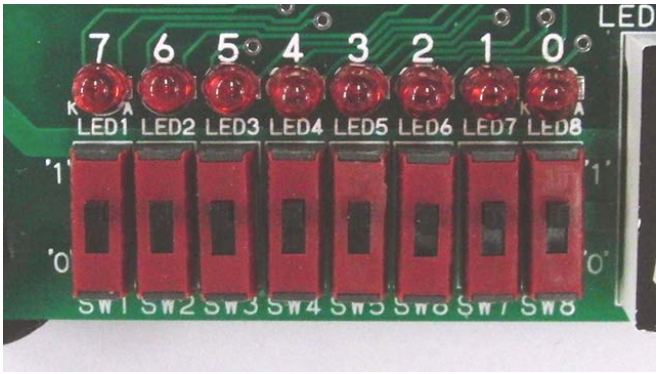
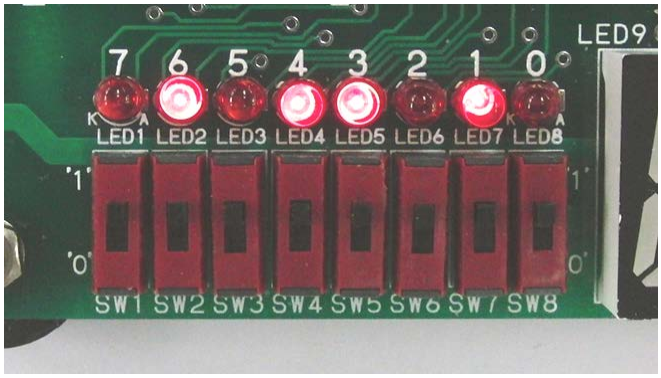
28.7.5 main 関数（無限ループ部分）

```
70 :      if( ret == 0 ) {
71 :          /* 書き込みエラーなら */
72 :          while( 1 ) {
73 :              p2 = 0xf0;
74 :              timer( 100 );
75 :              p2 = 0x0f;
76 :              timer( 100 );
77 :          }
78 :      }
79 :
80 :      /* 書き込みが正常にできたなら */
81 :      while( 1 ) {
82 :          p2 = 0x55;
83 :          timer( 200 );
84 :          p2 = 0xaa;
85 :          timer( 200 );
86 :      }
87 :  }
```

70 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。
72～ 77 行	正常に書き込みができなければ、ポート 2 に接続している機器(LED など)に 0xf0 と 0x0f を 0.1 秒ごとに表示させます。
81～ 86 行	正常に書き込みができれば、ポート 2 に接続している機器(LED など)に 0x55 と 0xaa を 0.2 秒ごとに表示させます。

28.8 実習手順

1		・「data_flash.mot」を書き込みます。 ・RY_R8C38 ボードの電源を入れます。 実習基板 Ver.2 の LED（LED1～LED8）は消灯しています。 ※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。
2		実習基板 Ver.2 のスライドスイッチ（SW1～SW8）を、好きな状態に変えます。 写真は、左から、「下上下上　上下上下」です。 下が"0"、上が"1"なので、左から「01011010」になります。スライドスイッチの状態を覚えておいてください。
3		RY_R8C38 ボードのディップスイッチの状態を変えます。どの bit を変えても構いません。 変えた瞬間、実習基板 Ver.2 のスライドスイッチの値をデータフラッシュに書き込みます。

4		LED が"0101 0101"と"10101 1010"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。
5		RY_R8C38 ボードの電源を切って、実習基板 Ver.2 のスライドスイッチを先ほどとは違う値にします。
6		再度、電源を入れると、先ほどディップスイッチを変えたときの実習基板 Ver.2 のスライドスイッチ値をデータフラッシュから呼び出して、実習基板 Ver.2 の LED に表示します。

28.9　演習

(1)「data_flash.c」の 68 行にある blockEraseDataFlash 関数部分をコメント文（または削除）にしたとき、動作がどうなるか確かめなさい。

29. 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)

29.1 概要

本書では、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。

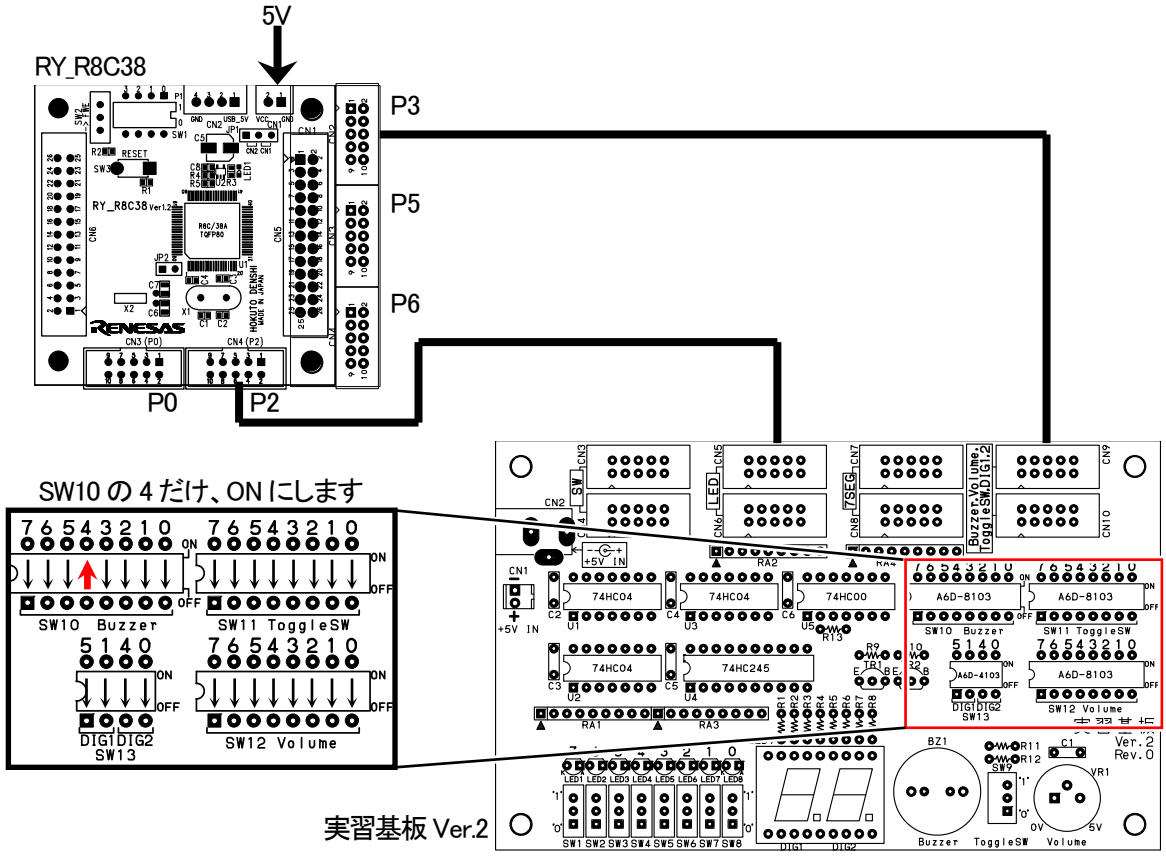
29.2 接続

■使用ポート

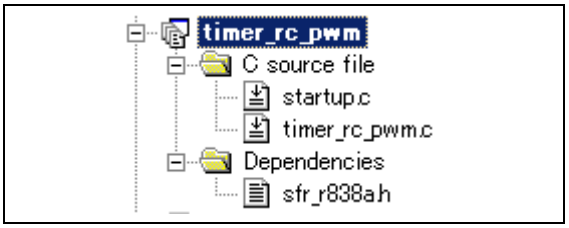
マイコンのポート	接続内容
P3_4 (変換基板 CN2)	実習基板 Ver.2 のブザーを接続します。

■接続

実習基板 Ver.2 を使ったときの接続例を下記に示します。



29.3 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。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥timer_rc_pwm¥startup.c
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/38A の内蔵周辺機能(SFR)の初期化も行います。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥timer_rc_pwm¥timer_rc_pwm.c
3	sfr_r838a.h	R8C/38A マイコンの内蔵周辺機能を制御するためのレジスタ（Special Function Registers)を定義したファイルです。 ファイルの位置→C:¥Workspace¥common_r8c38a¥sfr_r838a.h

29.4 プログラム「timer_rc_pwm.c」

```
 1 :  /***********************************************************************/
 2 :  /* 対象マイコン R8C/38A                                                */
 3 :  /* ﾌｧｲﾙ内容     タイマRCによるPWM出力                                  */
 4 :  /* バージョン   Ver.1.01                                               */
 5 :  /* Date         2010.04.01                                             */
 6 :  /* Copyright    ジャパンマイコンカーラリー実行委員会                   */
 7 :  /***********************************************************************/
 8 :  /*
 9 :  出力：TRCIOC端子(P3_4)にブザーを接続(実習基板Ver.2のブザー部)
10 :
11 :  TRCIOC端子(P3_4)に接続したブザーから、音を鳴らします。
12 :  */
13 :
14 :  /*=====================================*/
15 :  /* インクルード                        */
16 :  /*=====================================*/
17 :  #include "sfr_r838a.h"                 /* R8C/38A SFRの定義ファイル    */
18 :
19 :  /*=====================================*/
20 :  /* シンボル定義                        */
21 :  /*=====================================*/
22 :  /* 3オクターブ目の音階 */
23 :  #define         DO_3        38226     /* ド                           */
24 :  #define         DOU_3       36081     /* ド＃                         */
25 :  #define         RE_3        34056     /* レ                           */
26 :  #define         REU_3       32144     /* レ＃                         */
27 :  #define         MI_3        30340     /* ミ                           */
28 :  #define         FA_3        28637     /* ファ                         */
29 :  #define         FAU_3       27030     /* ファ＃                       */
30 :  #define         SO_3        25513     /* ソ                           */
31 :  #define         SOU_3       24081     /* ソ＃                         */
32 :  #define         RA_3        22729     /* ラ                           */
33 :  #define         RAU_3       21454     /* ラ＃                         */
34 :  #define         SI_3        20250     /* シ                           */
35 :
36 :  /* 4オクターブ目の音階 */
37 :  #define         DO_4        19113     /* ド                           */
38 :  #define         DOU_4       18040     /* ド＃                         */
39 :  #define         RE_4        17028     /* レ                           */
40 :  #define         REU_4       16072     /* レ＃                         */
41 :  #define         MI_4        15170     /* ミ                           */
42 :  #define         FA_4        14319     /* ファ                         */
43 :  #define         FAU_4       13515     /* ファ＃                       */
44 :  #define         SO_4        12756     /* ソ                           */
45 :  #define         SOU_4       12041     /* ソ＃                         */
46 :  #define         RA_4        11365     /* ラ                           */
47 :  #define         RAU_4       10727     /* ラ＃                         */
48 :  #define         SI_4        10125     /* シ                           */
49 :
50 :  /* 5オクターブ目の音階 */
51 :  #define         DO_5        9557      /* ド                           */
52 :  #define         DOU_5       9020      /* ド＃                         */
53 :  #define         RE_5        8514      /* レ                           */
54 :  #define         REU_5       8036      /* レ＃                         */
55 :  #define         MI_5        7585      /* ミ                           */
56 :  #define         FA_5        7159      /* ファ                         */
57 :  #define         FAU_5       6758      /* ファ＃                       */
58 :  #define         SO_5        6378      /* ソ                           */
59 :  #define         SOU_5       6020      /* ソ＃                         */
60 :  #define         RA_5        5682      /* ラ                           */
61 :  #define         RAU_5       5363      /* ラ＃                         */
62 :  #define         SI_5        5062      /* シ                           */
63 :
64 :  #define         TEMPO       60        /* テンポ                       */
65 :
66 :  /*=====================================*/
67 :  /* プロトタイプ宣言                    */
68 :  /*=====================================*/
69 :  void init( void );
70 :  void beep( unsigned int tone );
71 :
72 :  /*=====================================*/
73 :  /* グローバル変数の宣言                */
74 :  /*=====================================*/
75 :  unsigned long   cnt_rb;                /* タイマRB用                   */
76 :  int             music_dim;             /* 音楽データ配列の位置         */
77 :  int             music_flag;            /* 音楽データならすかどうか     */
78 :
```

```
 79 :  const int music_data[][2] = {          /* 大きな古時計　音楽データ      */
 80 :  /*
 81 :  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。
 82 :  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。
 83 :  */
 84 :  /*  音階,    長さ     */
 85 :      0,       0,  /* スタート*/
 86 :
 87 :      RE_4,    4,  /* お */
 88 :      SO_4,    4,  /* お */
 89 :      FAU_4,   2,  /* き */
 90 :      SO_4,    2,  /* な */
 91 :      RA_4,    4,  /* のっ */
 92 :      SO_4,    2,  /* ぽ */
 93 :      RA_4,    2,  /* の */
 94 :      SI_4,    2,  /* ふ */
 95 :      SI_4,    2,  /* る */
 96 :      DO_5,    2,  /* ど */
 97 :      SI_4,    2,  /* け */
 98 :      MI_4,    4,  /* い */
 99 :      RA_4,    2,  /* お */
100 :      RA_4,    2,  /* じ */
101 :      SO_4,    4,  /* い */
102 :      SO_4,    2,  /* さ */
103 :      SO_4,    2,  /* ん */
104 :      FAU_4,   4,  /* の */
105 :      MI_4,    2,  /* と */
106 :      FAU_4,   2,  /* け */
107 :      SO_4,    10, /* い */
108 :      0,       2,
109 :
110 :      RE_4,    2,  /* ひゃ */
111 :      RE_4,    2,  /* く */
112 :      SO_4,    4,  /* ねん */
113 :      FAU_4,   2,  /* い */
114 :      SO_4,    2,  /* つ */
115 :      RA_4,    4,  /* も */
116 :      SO_4,    2,  /* うご */
117 :      RA_4,    2,  /* ご */
118 :      SI_4,    4,  /* い */
119 :      DO_5,    2,  /* て */
120 :      SI_4,    2,  /* い */
121 :      MI_4,    4,  /* た */
122 :      RA_4,    2,  /* ご */
123 :      RA_4,    2,  /* じ */
124 :      SO_4,    4,  /* ま */
125 :      SO_4,    2,  /* ん */
126 :      SO_4,    2,  /* の */
127 :      FAU_4,   4,  /* と */
128 :      MI_4,    2,  /* け */
129 :      FAU_4,   2,  /* い */
130 :      SO_4,    10, /* さ */
131 :      0,       2,
132 :
133 :      SO_4,    2,  /* お */
134 :      SI_4,    2,  /* じ */
135 :      RE_5,    4,  /* い */
136 :      SI_4,    2,  /* さ */
137 :      RA_4,    2,  /* ん */
138 :      SO_4,    4,  /* の */
139 :      FAU_4,   2,  /* う */
140 :      SO_4,    2,  /* ま */
141 :      RA_4,    2,  /* れ */
142 :      SO_4,    2,  /* た */
143 :      FAU_4,   2,  /* あ */
144 :      MI_4,    2,  /* さ */
145 :      RE_4,    4,  /* に */
146 :      SO_4,    2,  /* か */
147 :      SI_4,    2,  /* っ */
148 :      RE_5,    4,  /* て */
149 :      SI_4,    2,  /* き */
150 :      RA_4,    2,  /* た */
151 :      SO_4,    4,  /* と */
152 :      FAU_4,   2,  /* け */
153 :      SO_4,    2,  /* い */
154 :      RA_4,    10, /* さ */
155 :      0,       2,
156 :
157 :      RE_4,    2,  /* い */
158 :      SO_4,    2,  /* ま */
159 :      SO_4,    2,  /* は */
160 :      0,       4,
161 :      RA_4,    2,  /* もう */
162 :      0,       2,
163 :      0,       4,
164 :      SI_4,    2,  /* う */
165 :      SI_4,    2,  /* ご */
166 :      DO_5,    2,  /* か */
167 :      SI_4,    2,  /* な */
168 :      MI_4,    4,  /* い */
169 :      RA_4,    2,  /* そ */
```

```
170 :      RA_4,    2,  /* の */
171 :      SO_4,    8,  /* と */
172 :      FAU_4,   8,  /* け */
173 :      SO_4,    8,  /* い */
174 :      0,       2,
175 :
176 :      RE_4,    2,  /* ひゃ */
177 :      RE_4,    2,  /* く */
178 :      SO_4,    4,  /* ねん */
179 :      RE_4,    2,  /* や */
180 :      RE_4,    2,  /* す */
181 :      MI_4,    2,  /* ま */
182 :      RE_4,    2,  /* ず */
183 :      RE_4,    4,  /* に */
184 :      SI_3,    2,  /* ちっく */
185 :      0,       2,
186 :      RE_4,    2,  /* たっく */
187 :      0,       2,
188 :      SI_3,    2,  /* ちっく */
189 :      0,       2,
190 :      RE_4,    2,  /* たっく */
191 :      RE_4,    2,  /* お */
192 :      SO_4,    4,  /* じー */
193 :      RE_4,    2,  /* さん */
194 :      RE_4,    2,  /* と */
195 :      MI_4,    4,  /* いっ */
196 :      RE_4,    2,  /* しょ */
197 :      RE_4,    2,  /* に */
198 :      SI_3,    2,  /* ちっく */
199 :      0,       2,
200 :      RE_4,    2,  /* たっく */
201 :      0,       2,
202 :      SI_3,    2,  /* ちっく */
203 :      0,       2,
204 :      RE_4,    2,  /* たっく */
205 :
206 :      RE_4,    2,  /* い */
207 :      SO_4,    2,  /* ま */
208 :      SO_4,    2,  /* は */
209 :      0,       4,
210 :      RA_4,    2,  /* もう */
211 :      0,       2,
212 :      0,       4,
213 :      SI_4,    2,  /* う */
214 :      SI_4,    2,  /* ご */
215 :      DO_5,    2,  /* か */
216 :      SI_4,    2,  /* な */
217 :      MI_4,    4,  /* い */
218 :      RA_4,    2,  /* そ */
219 :      RA_4,    2,  /* の */
220 :      SO_4,    8,  /* と */
221 :      FAU_4,   8,  /* け */
222 :      SO_4,    12, /* い */
223 :      0,       4,
224 :
225 :      -1,      0   /* 終了 */
226 :  };
227 :
228 :  /************************************************************************/
229 :  /* メインプログラム                                                     */
230 :  /************************************************************************/
231 :  void main( void )
232 :  {
233 :      unsigned char d;
234 :
235 :      init();                             /* 初期化                       */
236 :      asm(" fset I ");                    /* 全体の割り込み許可           */
237 :
238 :      music_flag = 1;                     /* 音楽スタート                 */
239 :
240 :      while( 1 ) {
241 :      }
242 :  }
243 :
244 :  /************************************************************************/
245 :  /* ブザーを鳴らす                                                       */
246 :  /* 引数  音階のPWM値                                                    */
247 :  /************************************************************************/
248 :  void beep( unsigned int tone )
249 :  {
250 :      if( tone ) {
251 :          trcmr  &= 0x7f;                 /* TRCのカウント停止            */
252 :          trc     = tone - 1;
253 :          trcgra  = tone - 1;             /* 周期設定                     */
254 :          trcgrc  = tone / 2 - 1;         /* ON幅設定                     */
255 :          trcmr  |= 0x80;                 /* TRCのカウント開始            */
256 :      } else {
257 :          trcmr  &= 0x7f;                 /* TRCのカウント停止            */
258 :          trc     = 0;
259 :          trcgra  = 0;                    /* 周期設定                     */
260 :          trcgrc  = 0;                    /* ON幅設定                     */
```

```
261 :          trcmr  |= 0x80;                  /* TRCのカウント開始            */
262 :      }
263 :  }
264 :
265 :  /************************************************************************/
266 :  /* R8C/38A スペシャルファンクションレジスタ(SFR)の初期化                */
267 :  /************************************************************************/
268 :  void init( void )
269 :  {
270 :      int i;
271 :
272 :      /* クロックをXINクロック(20MHz)に変更 */
273 :      prc0  = 1;                          /* プロテクト解除               */
274 :      cm13  = 1;                          /* P4_6,P4_7をXIN-XOUT端子にする*/
275 :      cm05  = 0;                          /* XINクロック発振              */
276 :      for(i=0; i<50; i++ );               /* 安定するまで少し待つ(約10ms) */
277 :      ocd2  = 0;                          /* システムクロックをXINにする  */
278 :      prc0  = 0;                          /* プロテクトON                 */
279 :
280 :      /* ポートの入出力設定 */
281 :      prc2 = 1;                           /* PD0のプロテクト解除          */
282 :      pd0 = 0x00;                         /*                              */
283 :      pd1 = 0xd0;                         /* 5:RXD0 4:TXD0 3-0:DIP SW     */
284 :      pd2 = 0xff;                         /*                              */
285 :      pd3 = 0xff;                         /* 4:ブザー                     */
286 :      p4  = 0x20;                         /* P4_5のLED:初期は点灯         */
287 :      pd4 = 0xb8;                         /* 7:XOUT 6:XIN 5:LED 2:VREF    */
288 :      pd5 = 0xff;                         /*                              */
289 :      pd6 = 0xff;                         /*                              */
290 :      pd7 = 0xff;                         /*                              */
291 :      pd8 = 0xff;                         /*                              */
292 :      pd9 = 0x3f;                         /*                              */
293 :      pur0 = 0x04;                        /* P1_3～P1_0のプルアップON     */
294 :
295 :      /* タイマRBの設定 */
296 :      /* 割り込み周期 = 1 / 20[MHz]    * (TRBPRE+1) * (TRBPR+1)
297 :                      = 1 / (20*10^6) * 200        * 100
298 :                      = 0.001[s] = 1[ms]
299 :      */
300 :      trbpre = 200-1;                     /* プリスケーラレジスタ         */
301 :      trbpr  = 100-1;                     /* プライマリレジスタ           */
302 :      trbmr  = 0x00;                      /* 動作モード、分周比設定       */
303 :      trbic  = 0x07;                      /* 割り込み優先レベル設定       */
304 :      trbcr  = 0x01;                      /* カウント開始                 */
305 :
306 :      /* タイマRC PWMモード設定 */
307 :      trcpsr0 = 0x00;                     /* TRCIOA,B端子の設定           */
308 :      trcpsr1 = 0x02;                     /* TRCIOC,D端子の設定           */
309 :      trcmr   = 0x0a;                     /* TRCIOCはPWMモード            */
310 :      trccr1  = 0xa0;                     /* カウントソース,初期出力の設定*/
311 :      trccr2  = 0x00;                     /* 出力レベルの設定             */
312 :      trcgra  = 0;                        /* 周期設定                     */
313 :      trcgrc  = 0;                        /* ON幅設定                     */
314 :      trcoer  = 0x0b;                     /* TROIOC出力許可               */
315 :  }
316 :
317 :  /************************************************************************/
318 :  /* タイマRB 割り込み処理                                                */
319 :  /************************************************************************/
320 :  #pragma interrupt intTRB(vect=24)
321 :  void intTRB( void )
322 :  {
323 :      cnt_rb++;
324 :
325 :      if( music_flag ) {
326 :          /* ブザー処理 */
327 :          if( cnt_rb >= 15000L * music_data[music_dim][1] / TEMPO ) {
328 :              /* 次の音階をならす */
329 :              cnt_rb = 0;
330 :              music_dim++;
331 :              if( music_data[music_dim][0] == -1 ) {
332 :                  /* -1なら終了 */
333 :                  beep( 0 );
334 :                  music_flag = 0;
335 :              } else {
336 :                  /* -1でないなら次の音階セット */
337 :                  beep( music_data[music_dim][0] );
338 :              }
339 :          }
340 :      }
341 :  }
342 :
343 :  /************************************************************************/
344 :  /* end of file                                                          */
345 :  /************************************************************************/
```

29.5 プログラムの解説

29.5.1 init 関数(タイマ RC の設定)

```
306 :       /* タイマRC PWMモード設定 */
307 :       trcpsr0 = 0x00;                 /* TRCIOA,B端子の設定              */
308 :       trcpsr1 = 0x02;                 /* TRCIOC,D端子の設定              */
309 :       trcmr   = 0x0a;                 /* TRCIOCはPWMモード               */
310 :       trccr1  = 0xa0;                 /* カウントソース,初期出力の設定*/
311 :       trccr2  = 0x00;                 /* 出力レベルの設定                */
312 :       trcgra  = 0;                    /* 周期設定                        */
313 :       trcgrc  = 0;                    /* ON幅設定                        */
314 :       trcoer  = 0x0b;                 /* TROIOC出力許可                  */
```

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/38A 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	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。 ・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て波形を出力します。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。

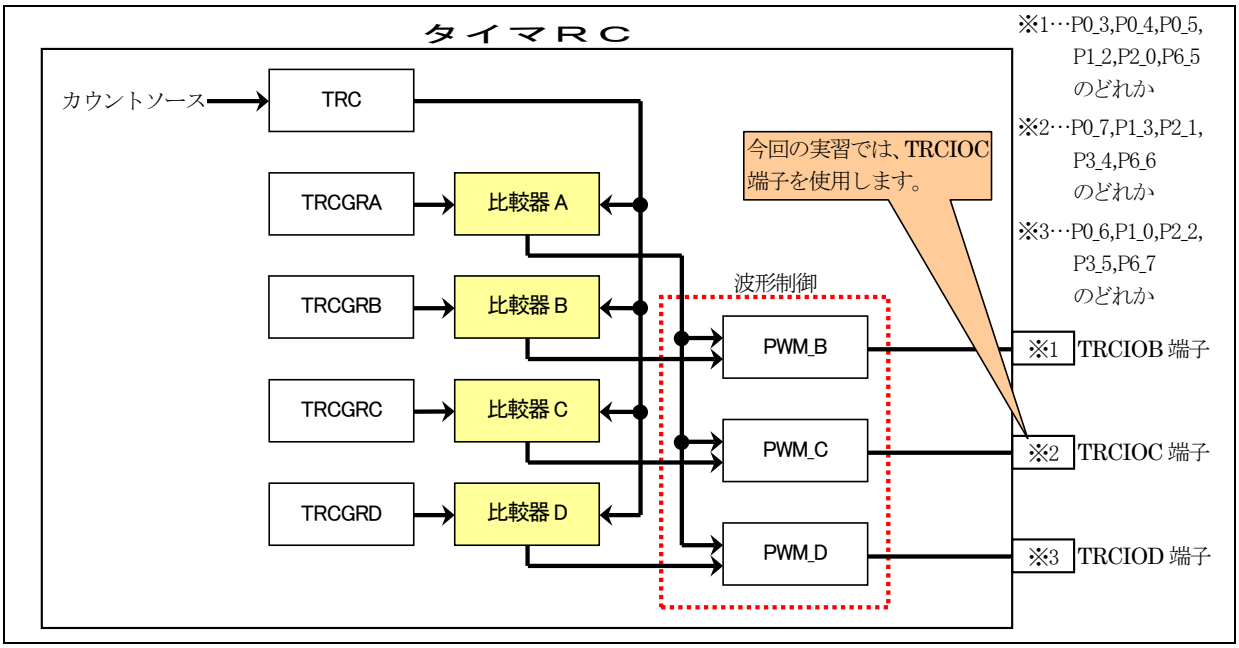
端子名	割り当てる端子	入出力	機能
TRCIOA	P0_0、P0_1、P0_2 または P1_1	入出力	モードによって機能が異なります。詳細は各モードを参照してください。
TRCIOB	P0_3、P0_4、P0_5、P1_2、P2_0 または P6_5		
TRCIOC	P0_7、P1_3、P2_1、P3_4 または P6_6		
TRCIOD	P0_6、P1_0、P2_2、P3_5 または P6_7		
TRCCLK	P1_4 または P3_3	入力	外部クロック入力
TRCTRG	P0_0、P0_1、P0_2 または P1_1	入力	PWM2 モードの外部トリガ入力

(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。

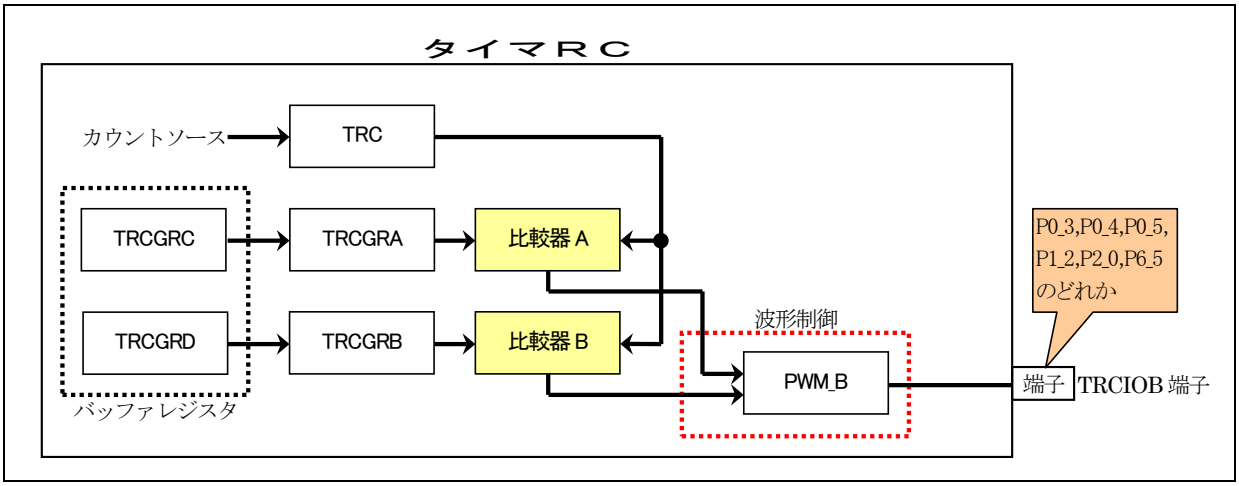
●バッファレジスタを使わない場合

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。



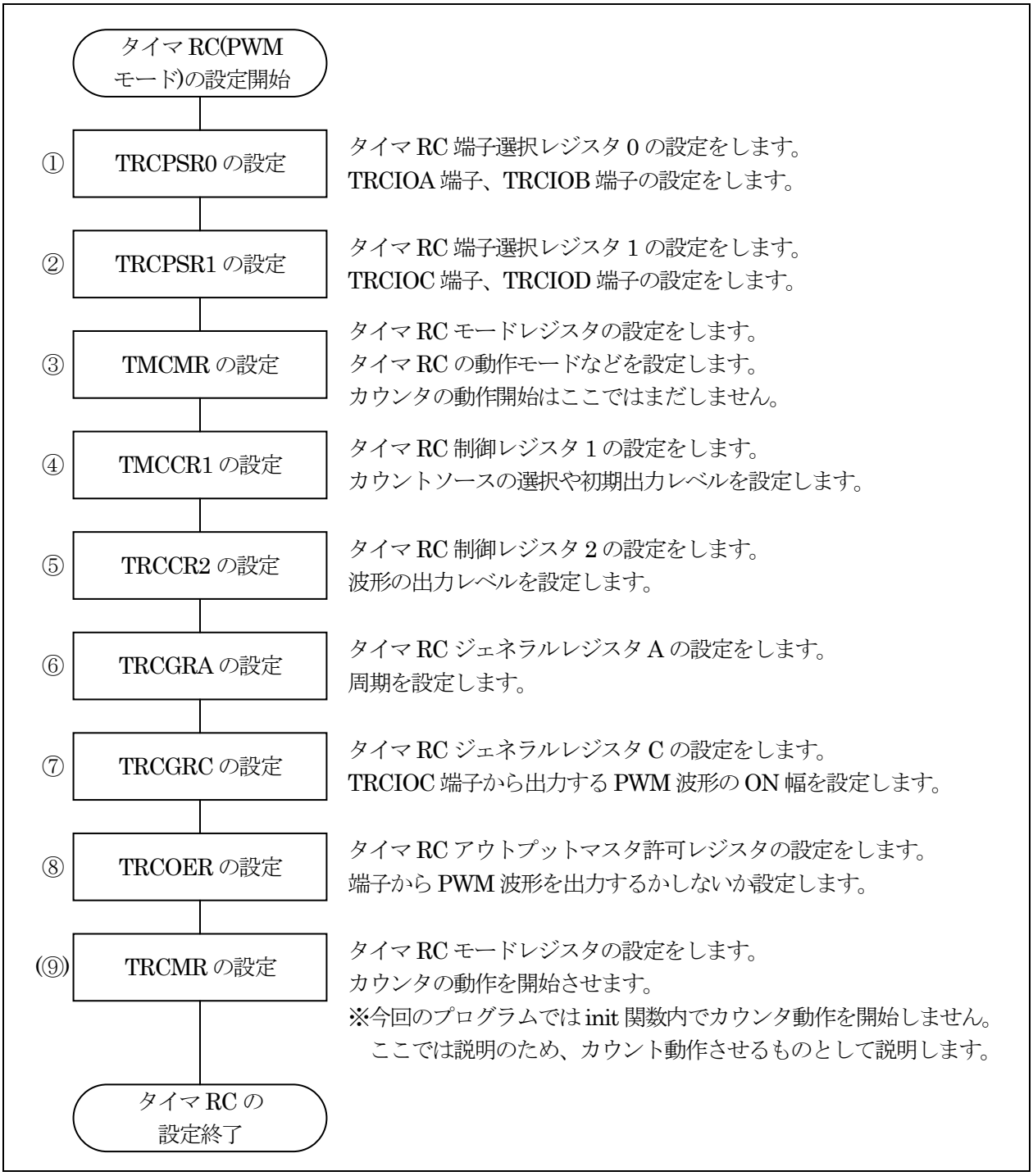
●バッファレジスタを使う場合

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必要ありません。



(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 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	O				O			

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

TRCIOC 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	TRCIOC 端子選択ビット bit2:trciocsel2 bit1:trciocsel1 bit0:trciocsel0	000：TRCIOC 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			

③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) pwmd_trcmr	0：タイマモード 1：PWM モード TRCIOD は使用しませので"0"を設定しておきます。	0
bit1	TRCIOC PWM モード選択ビット(注 1) pwmc_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) pwmb_trcmr	0：タイマモード 1：PWM モード TRCIOB は使用しませので"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「R8C/38A 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。

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	O				a			

④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				O			

※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の設定方法

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(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。

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、 f1/20MHz＝1/20MHz＝50ns 設定できる PWM 周期の最大は、 50ns×65,536＝**3.2768ms** 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、 f2/20MHz＝2/20MHz＝100ns 設定できる PWM 周期の最大は、 100ns×65,536＝**6.5536ms** 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、 f4/20MHz＝4/20MHz＝200ns 設定できる PWM 周期の最大は、 200ns×65,536＝**13.1072ms** 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、 f8/20MHz＝8/20MHz＝400ns 設定できる PWM 周期の最大は、 400ns×65,536＝**26.2144ms** 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、 f32/20MHz＝32/20MHz＝1600ns 設定できる PWM 周期の最大は、 1600ns×65,536＝**104.8576ms** 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[ms]

です。この周波数を設定できる PWM 周期に設定します。

- "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
- "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
- "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。

⑤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	TRCTRG 入力エッジ選択ビット(注 3) bit7:tceg1_trccr2 bit6:tceg0_trccr2	00：TRCTRG 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

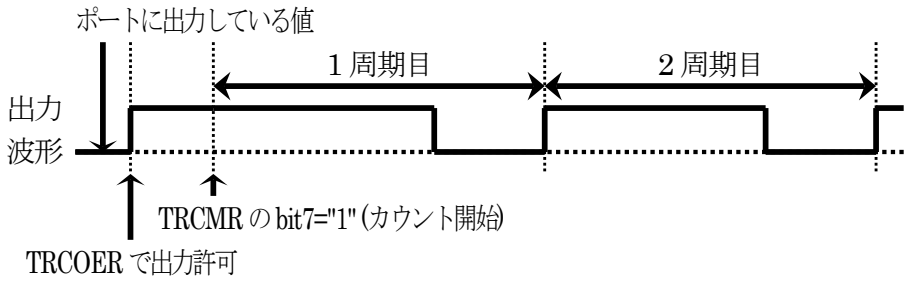
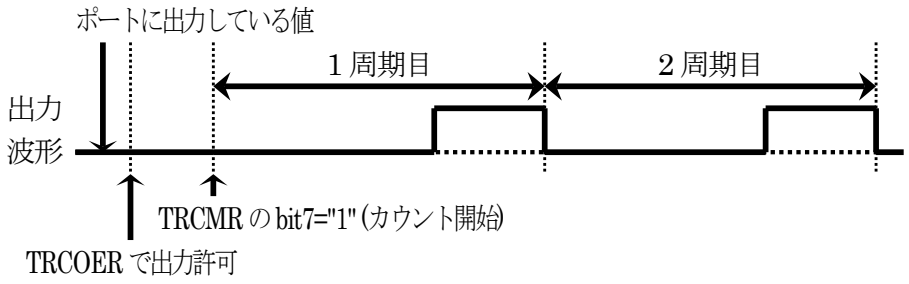
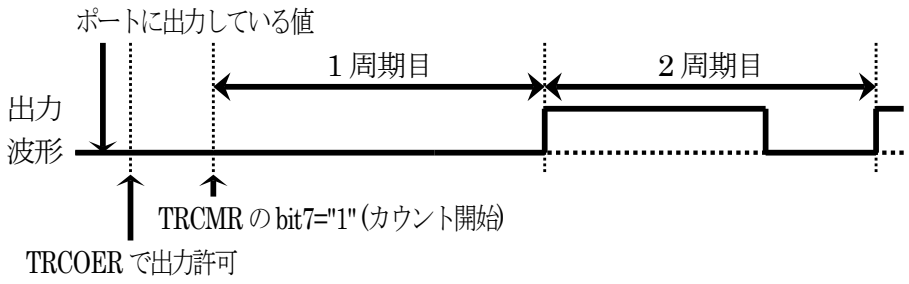
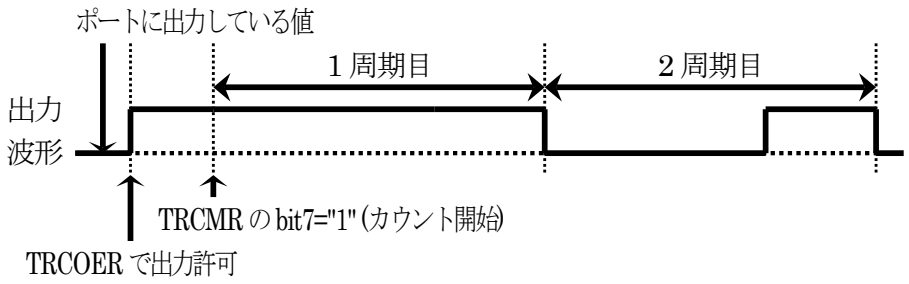
注 1. PWM モードのとき有効です。
注 2. アウトプットコンペア機能、PWM モード、PWM2 モードのとき有効です。PWM2 モード時の注意事項は「ハードウェアマニュアル 19.9.6 PWM2 モード時の TRCMR レジスタ」を参照してください。
注 3. PWM2 モードのとき有効です。

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	O				O			

※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	
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	
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	
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	

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。

⑥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。

PWM 周期＝タイマ RC カウンタのカウントソース×（TRCGRA＋1）

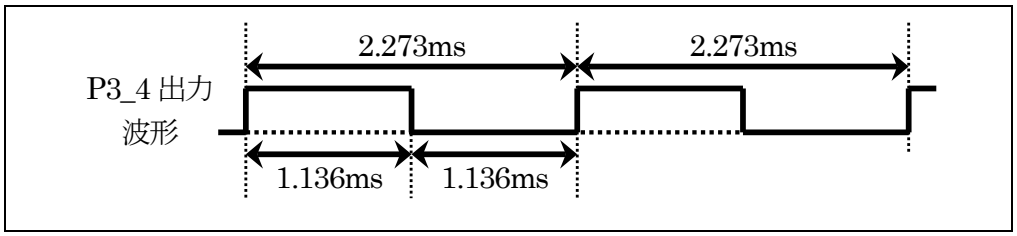
TRCGRA を左辺に移動して、TRCGRA を求める式に変形します。

TRCGRA＝PWM 周期／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＝2.273ms

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。

TRCGRA＝周期　　／カウントソース－1
TRCGRA＝$(2.273\times10^{-3})/(200\times10^{-9})$　－1
TRCGRA＝11364.7－1 ≒ 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。

⑦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。

TRCIOC 端子(P3_4)の ON 幅＝タイマ RC カウンタのカウントソース×(TRCGRC＋1)

TRCGRC を左辺に移動して、TRCGRC を求める式に変形します。

TRCGRC＝TRCIOC 端子(P3_4)の ON 幅／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」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。

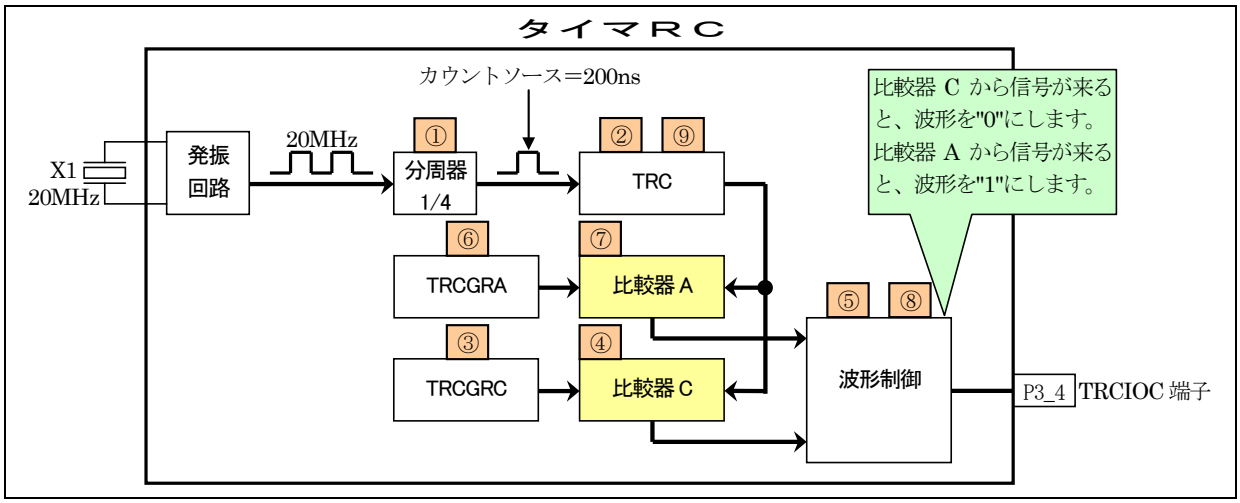
1/440.0＝2.273ms

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。

TRCGRC＝ON 幅　　／カウントソース－1
TRCGRC＝$(1.136\times10^{-3})/(200\times10^{-9})$　－1
TRCGRC＝5682.4－1 ≒ 5683－1 ＝ 5682

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。

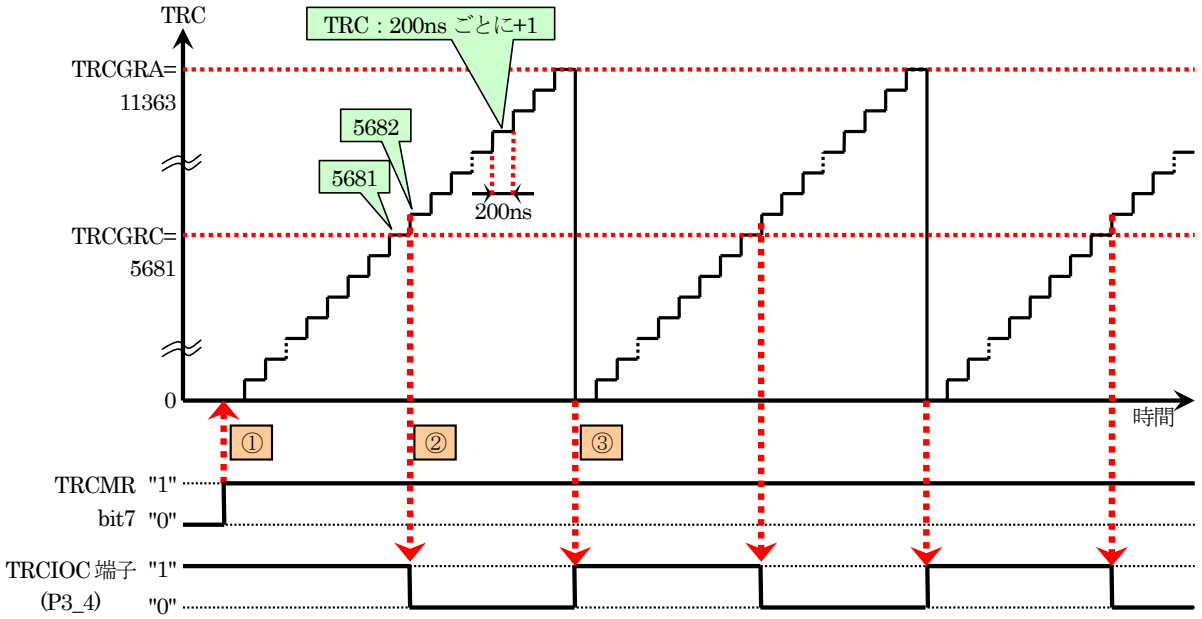
■仕組み



①	分周器には、クリスタルの 20HMz 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(TRCCR1)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 f4 を選択しています。これは、4 分周するということです。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	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。
③ ④	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。 TRCGRC＋１＝TRC 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 5681(TRCGRC) ＋１≠ 0(TRC) となり、成り立ちません。200ns 後、TRC は＋1されて 1 になります。 5681(TRCGRC) ＋１≠ 1(TRC) まだ成り立ちません。式が成り立つ TRC が 5682 になるには 5682×200ns＝1.1364ms です。よって、1.1364ms 経つと 5681(TRCGRC) ＋１＝ 5682(TRC) 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。
⑤	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。
⑥ ⑦	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。 TRCGRA＋１＝TRC 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 11363(TRCGRA) ＋１≠ 0(TRD0) となり、成り立ちません。200ns 後、TRC は＋1されて 1 になります。 11363(TRCGRA) ＋１≠ 1(TRD0) まだ成り立ちません。式が成り立つ TRC が 11364 になるには 11364×200ns＝2.273ms です。よって、2.273ms 経つと 11363(TRCGRA) ＋１＝ 11364(TRC) 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。
⑧ ⑨	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。

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＝(TRCGRC＋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[ns] ＝ 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。

⑧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

①、②で割り当てた場合は出力許可、使用しない場合は出力禁止を設定してください。

設定 bit	上:ビット名 下:シンボル	内容	今回の内容
bit7	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			

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) pwmd_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。	変更せず
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。	変更せず

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

29.5.2 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。

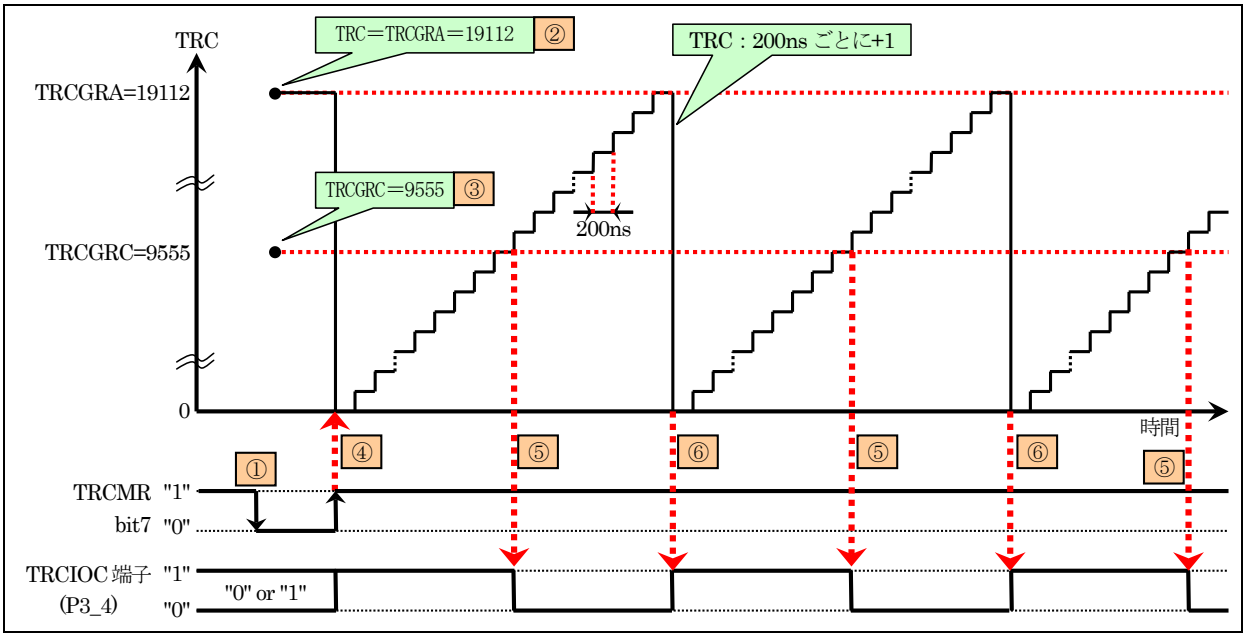
```
244 :  /************************************************************************/
245 :  /* ブザーを鳴らす                                                       */
246 :  /* 引数　音階のPWM値                                                    */
247 :  /************************************************************************/
248 :  void beep( unsigned int tone )
249 :  {
250 :      if( tone ) {
251 :          trcmr  &= 0x7f;                 /* TRCのカウント停止            */
252 :          trc     = tone - 1;
253 :          trcgra  = tone - 1;             /* 周期設定                     */
254 :          trcgrc  = tone / 2 - 1;         /* ON幅設定                     */
255 :          trcmr  |= 0x80;                 /* TRCのカウント開始            */
256 :      } else {
257 :          trcmr  &= 0x7f;                 /* TRCのカウント停止            */
258 :          trc     = 0;
259 :          trcgra  = 0;                    /* 周期設定                     */
260 :          trcgrc  = 0;                    /* ON幅設定                     */
261 :          trcmr  |= 0x80;                 /* TRCのカウント開始            */
262 :      }
263 :  }
```

250 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 251 行へ、ブザーを止めるなら 257 行へジャンプします。
251 行	タイマ RC モードレジスタ(TRCMR)の bit7 を"0"にして、TRC のカウント動作を停止します。PWM 波形出力が止まるので、ブザーが鳴りやみます。
252 行～ 254 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。
255 行	タイマ RC モードレジスタ(TRCMR)の bit7 を"1"にして、TRC のカウント動作を開始します。P3_4 端子から PWM 波形が出力され、ブザーが鳴ります。
257 行	タイマ RC モードレジスタ(TRCMR)の bit7 を"0"にして、TRC のカウント動作を停止します。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。
258 行～ 260 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。
261 行	タイマ RC モードレジスタ(TRCMR)の bit7 を"1"にして、TRC のカウント動作を開始します。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。

```
251 :          trcmr  &= 0x7f;                  /* TRCのカウント停止          */
252 :          trc    = 19113 - 1;
253 :          trcgra = 19113 - 1;              /* 周期設定                   */
254 :          trcgrc = 19113 / 2 - 1;          /* ON幅設定                   */
255 :          trcmr  |= 0x80;                  /* TRCのカウント開始          */
```

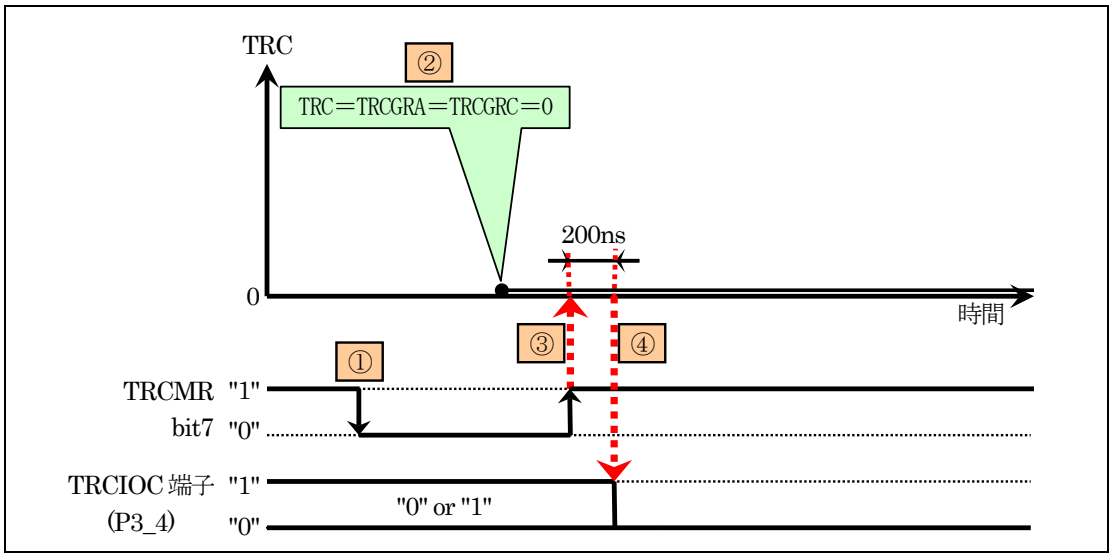


①	251 行	タイマ RC カウンタ(TRC)を停止させます。
②	252 行 253 行	タイマ RC ジェネラルレジスタ A(TRCGRA)に周期を設定します。タイマ RC カウンタ(TRC)に、TRCGRA と同じ値を設定します。
③	254 行	タイマ RC ジェネラルレジスタ C(TRCGRC)に ON 幅を設定します。周期の半分の時間です。
④	255 行	タイマ RC カウンタ(TRC)を動作させます。 動作させてから、200ns 経つと TRCGRA+1＝TRC が成り立ち、P3_4 が"1"に、TRC が 0 になります。
⑤		TRCGRC+1＝TRC が成り立つと、P3_4 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。

```
257 :           trcmr  &= 0x7f;                 /* TRCのカウント停止          */
258 :           trc     = 0;
259 :           trcgra  = 0;                    /* 周期設定                  */
260 :           trcgrc  = 0;                    /* ON幅設定                  */
261 :           trcmr  |= 0x80;                 /* TRCのカウント開始          */
```



①	257 行	タイマ RC カウンタ(TRC)を停止させます。
②	258～ 260 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	261 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。**

29.5.3 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。

```
317 :  /************************************************************************/
318 :  /* タイマRB 割り込み処理                                                */
319 :  /************************************************************************/
320 :  #pragma interrupt intTRB(vect=24)
321 :  void intTRB( void )
322 :  {
323 :      cnt_rb++;
324 :
325 :      if( music_flag ) {
326 :          /* ブザー処理 */
327 :          if( cnt_rb >= 15000L * music_data[music_dim][1] / TEMPO ) {
328 :              /* 次の音階をならす */
329 :              cnt_rb = 0;
330 :              music_dim++;
331 :              if( music_data[music_dim][0] == -1 ) {
332 :                  /* -1なら終了 */
333 :                  beep( 0 );
334 :                  music_flag = 0;
335 :              } else {
336 :                  /* -1でないなら次の音階セット */
337 :                  beep( music_data[music_dim][0] );
338 :              }
339 :          }
340 :      }
341 :  }
```

323 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。
325 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。
327 行	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。 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 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。
329 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。
330 行	music_dim 変数を＋1して、次の音階データの参照先にします。

331～ 334 行	音階データが「-1」なら、音階データが終わりと判断して、beep 関数でブザーへの出力を止め、music_flag を 0 にします。
337 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。

29.5.4 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。

```
231 :  void main( void )
232 :  {
233 :      unsigned char d;
234 :
235 :      init();                          /* 初期化                    */
236 :      asm(" fset I ");                 /* 全体の割り込み許可        */
237 :
238 :      music_flag = 1;                  /* 音楽スタート              */
239 :
240 :      while( 1 ) {
241 :      }
242 :  }
```

235 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。
236 行	割り込みを許可します。
238 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。
240～ 241 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。

29.6 音階

29.6.1 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。

$$周波数＝261.6×2^{(x/12)}\ [Hz]$$

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。

ｵｸﾀｰﾌﾞ	音	x	計算	周波数[Hz]	周期[ms]	TRCGRA の値
0	ド	-48	$261.6\times2^{(-48/12)}$	16.4	61.16	305810※
1	ド	-36	$261.6\times2^{(-36/12)}$	32.7	30.58	152905※
2	ド	-24	$261.6\times2^{(-24/12)}$	65.4	15.29	76453※
3	ド	-12	$261.6\times2^{(-12/12)}$	130.8	7.65	38226
4	ド	0	$261.6\times2^{(0/12)}$	261.6	3.82	19113
	ド＃	1	$261.6\times2^{(1/12)}$	277.2	3.61	18040
	レ	2	$261.6\times2^{(2/12)}$	293.6	3.41	17028
	レ＃	3	$261.6\times2^{(3/12)}$	311.1	3.21	16072
	ミ	4	$261.6\times2^{(4/12)}$	329.6	3.03	15170
	ファ	5	$261.6\times2^{(5/12)}$	349.2	2.86	14319
	ファ＃	6	$261.6\times2^{(6/12)}$	370.0	2.70	13515
	ソ	7	$261.6\times2^{(7/12)}$	392.0	2.55	12756
	ソ＃	8	$261.6\times2^{(8/12)}$	415.3	2.41	12041
	ラ	9	$261.6\times2^{(9/12)}$	440.0	2.27	11365
	ラ＃	10	$261.6\times2^{(10/12)}$	466.1	2.15	10727
	シ	11	$261.6\times2^{(11/12)}$	493.8	2.02	10125
5	ド	12	$261.6\times2^{(12/12)}$	523.2	1.91	9557
6	ド	24	$261.6\times2^{(24/12)}$	1046.4	0.96	4778
7	ド	36	$261.6\times2^{(36/12)}$	2092.8	0.48	2389
8	ド	48	$261.6\times2^{(48/12)}$	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。

ドの周期／タイマ RC カウンタ(TRC)のカウントアップする間隔
$=(3.82\times10^{-3})/(200\times10^{-9})=19113$

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みてください。

29.6.2 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。

```
36 :  /* 4 オクターブ目の音階 */
                    名称    TRCGRA の設定値
37 :  #define       DO_4       19113      /* ド                  */
38 :  #define       DOU_4      18040      /* ド＃                */
39 :  #define       RE_4       17028      /* レ                  */
40 :  #define       REU_4      16072      /* レ＃                */
41 :  #define       MI_4       15170      /* ミ                  */
42 :  #define       FA_4       14319      /* ファ                */
43 :  #define       FAU_4      13515      /* ファ＃              */
44 :  #define       SO_4       12756      /* ソ                  */
45 :  #define       SOU_4      12041      /* ソ＃                */
46 :  #define       RA_4       11365      /* ラ                  */
47 :  #define       RAU_4      10727      /* ラ＃                */
48 :  #define       SI_4       10125      /* シ                  */
```

29.6.3 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 回分演奏する早さです。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。
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。

時間　＝(60/テンポ)×(長さ/4)×1000ms
　　＝15/テンポ×長さ×1000
　　＝15000×長さ/テンポ［ms］

計算の順番は、割り算を最後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。

```
64 :  #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。

```
327 :           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。

29.6.4 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。

30. モータのスピードをボリュームで制御（プロジェクト：ad_pwm）

30.1 概要

本章では、マイコンカー用のモータドライブ基板 Ver.5 の左モータの回転をボリュームで制御する方法を紹介します。ボリュームの回転により正転、停止、逆転させることができます。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 RD によるリセット同期 PWM モード、A/D コンバータを使っていますがここでは説明していません。タイマ RD によるリセット同期 PWM モードについては、**「22. 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）」**、A/D コンバータについては、**「18. A/D コンバータ(繰り返しモード 0)（プロジェクト：ad_kurikaeshi）」**を参照してください。

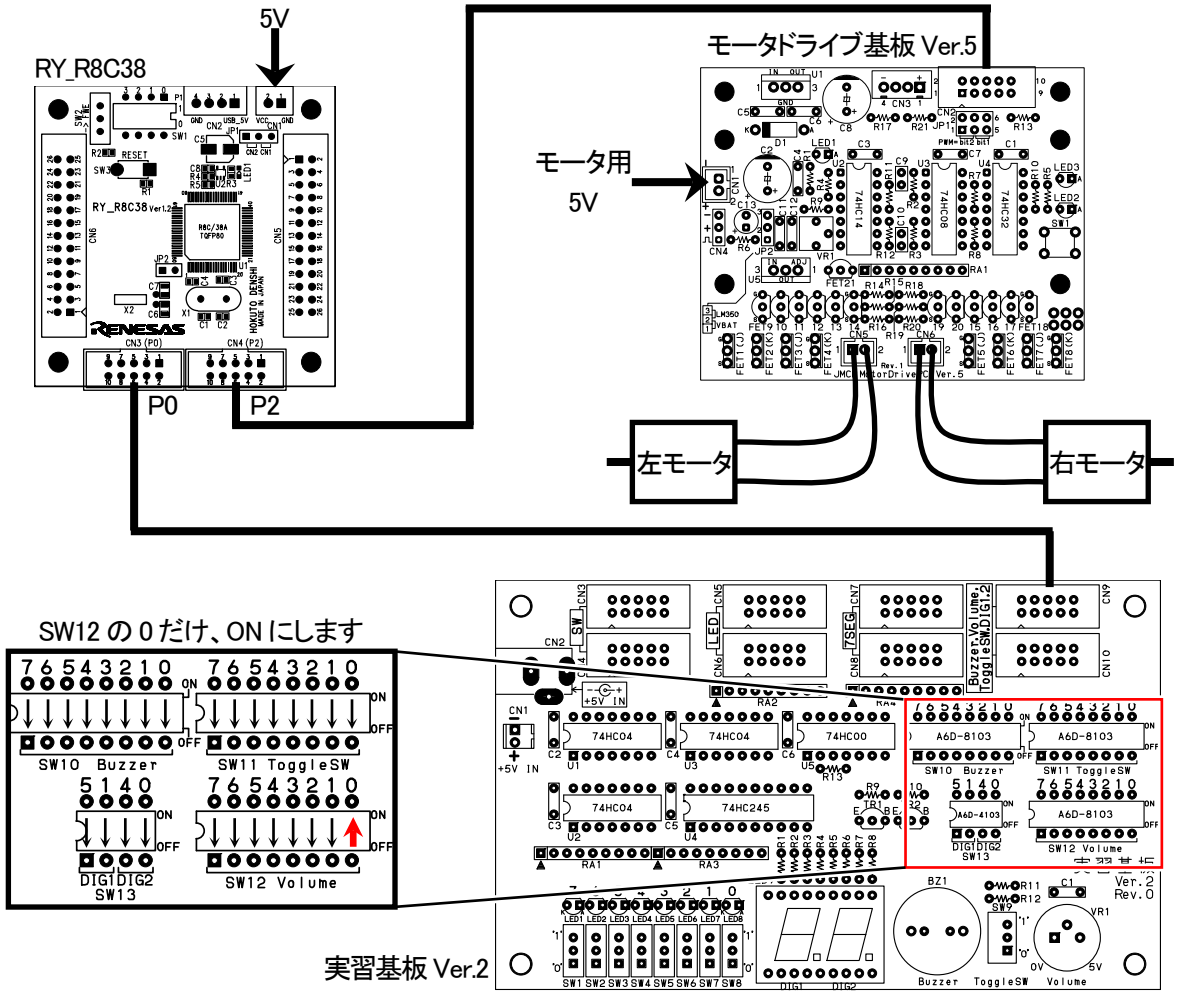
30.2 接続

■使用ポート

マイコンのポート	接続内容
P0_0 (CN3)	実習基板 Ver.2 のボリューム部など、0～5V の電圧が変化する入力機器を接続します。
P2 (CN4)	マイコンカー用モータドライブ基板 Ver.5 を接続

■接続例

RY_R8C38 ボードと実習基板 Ver.2、モータドライブ基板 Ver.5 をフラットケーブルで接続します。



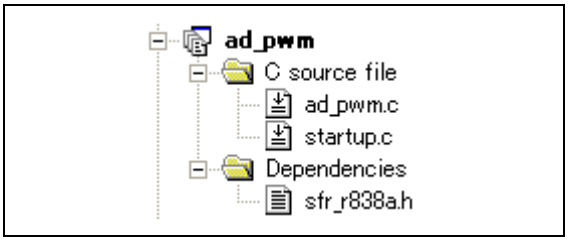
※モータドライブ基板 Ver.4 も接続可能です。プログラムも同一のもので動作します。

■操作方法

モータ(マイコンカーの場合はタイヤ)が回りますので、周辺には何も置かないでください。

実習基板 Ver.2 のボリュームを中間(2.5V 付近)にして電源を入れると、左モータは動きません。ボリュームをゆっくりと右に回すと、左モータがゆっくりと正転します。ボリュームをいちばん右まで回すと最高速で正転します。次にボリュームをゆっくりと左に回すと、左モータがゆっくりと逆転します。ボリュームをいちばん左まで回すと最高速で逆転します。

30.3 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。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥motor¥startup.c
2	ap_pwm.c	実際に制御するプログラムが書かれています。R8C/38A の内蔵周辺機能(SFR)の初期化も行います。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥ad_pwm¥ad_pwm.c
3	sfr_r838a.h	R8C/38A マイコンの内蔵周辺機能を制御するためのレジスタ（Special Function Registers)を定義したファイルです。 ファイルの位置→C:¥Workspace¥common_r8c38a¥sfr_r838a.h

30.4 プログラム「ad_pwm.c」

```
 1 :  /***********************************************************************/
 2 :  /* 対象マイコン R8C/38A                                                */
 3 :  /* ﾌｧｲﾙ内容     モータのスピードをボリュームで制御                     */
 4 :  /* バージョン   Ver.1.01                                               */
 5 :  /* Date         2012.10.18                                             */
 6 :  /* Copyright    ジャパンマイコンカーラリー実行委員会                   */
 7 :  /***********************************************************************/
 8 :  /*
 9 :  入力：AN7(P0_0)端子 0～5V(実習基板Ver.2のボリューム部)
10 :  出力：PB7-PB0(モータドライブ基板Ver.5)
11 :
12 :  AN7(P0_0)端子から入力した電圧でモータを回します。
13 :  中心で停止、ボリュームを左に回すと正転、右に回すと逆転します。
14 :  */
15 :
16 :  /*====================================*/
17 :  /* インクルード                       */
18 :  /*====================================*/
19 :  #include "sfr_r838a.h"                 /* R8C/38A 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 :
26 :  /*====================================*/
27 :  /* プロトタイプ宣言                   */
28 :  /*====================================*/
29 :  void init( void );
30 :  int get_ad7( void );
31 :  int data_cnv( int data );
32 :
33 :  /***********************************************************************/
34 :  /* メインプログラム                                                    */
35 :  /***********************************************************************/
36 :  void main( void )
37 :  {
38 :      int ad, pwm;
39 :
40 :      init();                           /* 初期化                       */
41 :
```

```
 42 :     while( 1 ) {
 43 :         ad = get_ad7();
 44 :         pwm = data_cnv( ad );           /* A/D値→PWM値に変換       */
 45 :
 46 :         if( pwm > 0 ) {
 47 :             /* 正転 */
 48 :             p2 &= 0xfd;
 49 :             trdgrd0 = (long)( PWM_CYCLE - 1 ) * pwm / 100;
 50 :         } else if( pwm < 0 ) {
 51 :             /* 逆転 */
 52 :             p2 |= 0x02;
 53 :             trdgrd0 = (long)( PWM_CYCLE - 1 ) * ( -pwm ) / 100;
 54 :         }
 55 :     }
 56 : }
 57 :
 58 : /************************************************************************/
 59 : /* R8C/38A スペシャルファンクションレジスタ(SFR)の初期化                */
 60 : /************************************************************************/
 61 : void init( void )
 62 : {
 63 :     int i;
 64 :
 65 :     /* クロックをXINクロック(20MHz)に変更 */
 66 :     prc0  = 1;                          /* プロテクト解除               */
 67 :     cm13  = 1;                          /* P4_6,P4_7をXIN-XOUT端子にする*/
 68 :     cm05  = 0;                          /* XINクロック発振              */
 69 :     for(i=0; i<50; i++ );               /* 安定するまで少し待つ(約10ms) */
 70 :     ocd2  = 0;                          /* システムクロックをXINにする  */
 71 :     prc0  = 0;                          /* プロテクトON                 */
 72 :
 73 :     /* ポートの入出力設定 */
 74 :     prc2 = 1;                           /* PD0のプロテクト解除          */
 75 :     pd0 = 0x00;                         /* 0:ボリュームなどのｱﾅﾛｸﾞ電圧  */
 76 :     pd1 = 0xd0;                         /* 5:RXD0 4:TXD0 3-0:DIP SW     */
 77 :     pd2 = 0xfe;                         /* 7-0:モータドライブ基板Ver.5  */
 78 :     pd3 = 0xff;                         /*                              */
 79 :     p4  = 0x20;                         /* P4_5のLED:初期は点灯         */
 80 :     pd4 = 0xb8;                         /* 7:XOUT 6:XIN 5:LED 2:VREF    */
 81 :     pd5 = 0xff;                         /*                              */
 82 :     pd6 = 0xff;                         /*                              */
 83 :     pd7 = 0xff;                         /*                              */
 84 :     pd8 = 0xff;                         /*                              */
 85 :     pd9 = 0x3f;                         /*                              */
 86 :     pur0 = 0x04;                        /* P1_3～P1_0のプルアップON     */
 87 :
 88 :     /* A/Dコンバータの設定 */
 89 :     admod   = 0x13;                     /* 繰り返しモード0に設定        */
 90 :     adinsel = 0x07;                     /* 入力端子AN7(P0_0)を選択      */
 91 :     adcon1  = 0x30;                     /* A/D動作可能                  */
 92 :     asm(" nop ");                       /* φADの1サイクルウエイト入れる*/
 93 :     adcon0  = 0x01;                     /* A/D変換スタート              */
 94 :
 95 :     /* タイマRD リセット同期PWMモードの設定*/
 96 :     /* PWM周期 = 1 / 20[MHz]   * カウントソース * (TRDGRA0+1)
 97 :                = 1 / (20*10^6) * 8              * 40000
 98 :                = 0.016[s] = 16[ms]
 99 :     */
100 :     trdpsr0 = 0x08;                     /* TRDIOB0,C0,D0端子設定        */
101 :     trdpsr1 = 0x05;                     /* TRDIOA1,B1,C1,D1端子設定     */
102 :     trdmr   = 0xf0;                     /* バッファレジスタ設定         */
103 :     trdfcr  = 0x01;                     /* リセット同期PWMモードに設定  */
104 :     trdcr0  = 0x23;                     /* ソースカウントの選択:f8      */
105 :     trdgra0 = trdgrc0 = PWM_CYCLE;      /* 周期                         */
106 :     trdgrb0 = trdgrd0 = 0;              /* P2_2端子のON幅設定           */
107 :     trdgra1 = trdgrc1 = 0;              /* P2_4端子のON幅設定           */
108 :     trdgrb1 = trdgrd1 = 3750;           /* P2_5端子のON幅設定           */
109 :     trdoer1 = 0xcd;                     /* 出力端子の選択               */
110 :     trdstr  = 0x0d;                     /* TRD0カウント開始             */
111 : }
112 :
113 : /************************************************************************/
114 : /* A/D値読み込み(AN7)                                                   */
115 : /* 引数    なし                                                         */
116 : /* 戻り値 A/D値 0～1023                                                 */
117 : /************************************************************************/
118 : int get_ad7( void )
119 : {
120 :     int i;
121 :
122 :     /* 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 */
123 :     i = ad7;
124 :
125 :     return i;
126 : }
127 :
```

```
128 :  /************************************************************************/
129 :  /* A/D値→PWM値に変換 2.5V(中心)で0%、0Vで100%、5Vで-100%                */
130 :  /* 引数    AD値  0-1023                                                 */
131 :  /* 戻り値 -100～100                                                     */
132 :  /************************************************************************/
133 :  int data_cnv( int data )
134 :  {
135 :      int ret;
136 :
137 :      if( data <= 400 ) {
138 :          /* 正転 0～100*/
139 :          ret = 400 - data;
140 :          ret /= 4;
141 :      } else if( data >= 623 ){
142 :          /* 逆転 -100 ～ 0 */
143 :          ret = 623 - data;
144 :          ret /= 4;
145 :      } else {
146 :          /* 停止 */
147 :          ret = 0;
148 :      }
149 :      return ret;
150 :  }
151 :
152 :  /************************************************************************/
153 :  /* end of file                                                          */
154 :  /************************************************************************/
```

30.5 プログラムの解説

30.5.1 data_cnv 関数（データ変換関数）

data_cnv 関数は、A/D 値(0～1023)を-100～0 の値に変換する関数です。

```
128 :  /************************************************************************/
129 :  /* A/D値→PWM値に変換 2.5V(中心)で0%、0Vで100%、5Vで-100%                */
130 :  /* 引数    AD値  0-1023                                                 */
131 :  /* 戻り値 -100～100                                                     */
132 :  /************************************************************************/
133 :  int data_cnv( int data )
134 :  {
135 :      int ret;
136 :
137 :      if( data <= 400 ) {
138 :          /* 正転 0～100*/
139 :          ret = 400 - data;
140 :          ret /= 4;
141 :      } else if( data >= 623 ){
142 :          /* 逆転 -100 ～ 0 */
143 :          ret = 623 - data;
144 :          ret /= 4;
145 :      } else {
146 :          /* 停止 */
147 :          ret = 0;
148 :      }
149 :      return ret;
150 :  }
```

変換の内容を、下表に示します。

A/D 値 （引数）	電圧	PWM 変換値 （戻り値）	説明
0～400	0V～1.96V	100%～0%	ボリュームを左いっぱい回したときを正転 100%にします。
401～622	1.96V～3.04V	0%	0～2.5V を正転、2.5V～5.0V を逆転にすると 0%の範囲が狭すぎて、停止がうまくできません。そのため、2.5V±約 0.5V の範囲を 0%にします。
623～1023	3.04V～5.00V	-0%～-100%	ボリュームを右いっぱい回したときを逆転 100%にします。

プログラムの説明を下表に示します。

139～ 140 行	A/D 値が 0～400 のとき、次の計算を行います。 ret = 400 － A/D 値(0～400) A/D 値が 0～400 のとき、ret 変数には 400～0 が入ります。大・小を、逆にします。 次に、4 で割ります。よって、400～0 の値を 100～0 にします。 これで、A/D 値が 0(左いっぱいに回したとき)のときに 100%、A/D 値が 400 のときに 0%にします。
143～ 144 行	A/D 値が 623～1023 のとき、次の計算を行います。 ret = 623 － A/D 値(623～1023) A/D 値が 623～1023 のとき、ret 変数には 0～-400 が入ります。623 を 0、1023 を-400 にします。 次に、4 で割ります。よって、0～-400 の値を 0～-100 にします。 これで、A/D 値が 1023(右いっぱいに回したとき)のときに-100%、A/D 値が 623 のときに 0%にします。
147 行	A/D 値が 401～622 のとき、0%にします。

30.5.2 main 関数

A/D 変換値(0～1023)を 0～100 の値に変換して、タイマ RD によるリセット同期 PWM モードを使って、デューティ比(ON の割合)を変更します。

```
 36 :  void main( void )
 37 :  {
 38 :      int ad, pwm;
 39 :
 40 :      init();                                 /* 初期化                          */
 41 :
 42 :      while( 1 ) {
 43 :          ad = get_ad7();
 44 :          pwm = data_cnv( ad );               /* A/D値→PWM値に変換               */
 45 :
 46 :          if( pwm > 0 ) {
 47 :              /* 正転 */
 48 :              p2 &= 0xfd;
 49 :              trdgrd0 = (long)( PWM_CYCLE - 1 ) * pwm / 100;
 50 :          } else if( pwm < 0 ) {
 51 :              /* 逆転 */
 52 :              p2 |= 0x02;
 53 :              trdgrd0 = (long)( PWM_CYCLE - 1 ) * ( -pwm ) / 100;
 54 :          }
```

```
55 :        }
56 :  }
```

43 行	get_ad7 関数で AN7 端子(P0_0)の A/D 変換値を取得し、ad 変数に格納します。
44 行	data_cnv 関数で、0～1023 の A/D 値を-100～100 の PWM 値に変換します。結果は、pwm 変数に格納されます。
48～ 49 行	pwm 変数が 1～100 なら、48～49 行を実行します。 P2_1=″0″を設定し、P2_2 端子から PWM 出力すると、PWM の割合に応じて左モータが正転します。 48 行目で下表のビット演算を行い、P2_1 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 / 1 / 1 / 1 / 1 / 1 / **0** / 1 結果 / P2_7 / P2_6 / P2_5 / P2_4 / P2_3 / P2_2 / **0** / P2_0 49 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。 $TRDGRD0 = (PWM\_CYCLE - 1) \times \frac{pwm\ (0 \sim 100)}{100}$ $= 39998 \times \frac{pwm\ (0 \sim 100)}{100}$ 例えば pwm=60 なら、TRDGRD0 は次のように計算されます。 TRDGRD0＝39998×60／100＝23998.8＝23998
52～ 53 行	pwm 変数が-1～-100 なら、52～53 行を実行します。 P2_1=″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逆転します。 52 行目で下表のビット演算を行い、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 53 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。 $TRDGRD0 = (PWM\_CYCLE - 1) \times \frac{-pwm\ (-1 \sim -100)}{100}$ $= 39998 \times \frac{-pwm\ (-1 \sim -100)}{100}$ ポイントは、pwm 変数が負の数ということです。回路的には P2_1=″1″で逆転の設定になっています。そのため pwm 変数は、正の数に直して計算します。直し方は、計算式の中で「**-pwm**」としています。例えば pwm=-30 なら、TRDGRD0 は次のように計算されます。 TRDGRD0＝39998×{-(-30)}／100＝39998×30／100＝11999.4＝11999

31. タイマ RE によるリアルタイムクロック（プロジェクト：timer_re_rtc）

31.1 概要

本章では、タイマ RE によるリアルタイムクロック機能を説明します。リアルタイムクロックとは時計機能のことで、タイマ RE を使えば、簡単に月、日、曜日、時、分、秒のカウントができます。

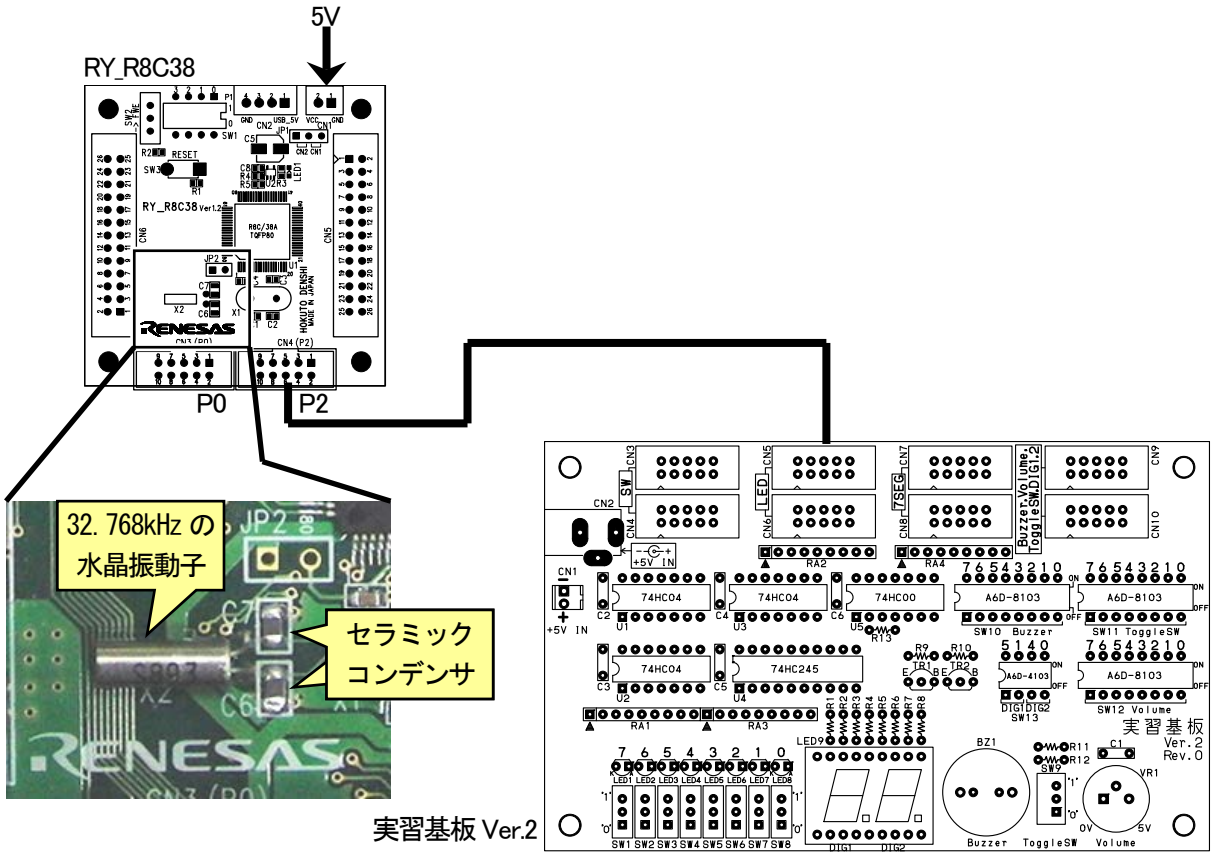
31.2 接続

■使用ポート

マイコンのポート	接続内容
P2 (CN4)	実習基板 Ver.2 の LED 部など、出力機器を接続します。
P4_3,P4_4	X2 に 32.768kHz の水晶振動子、C6 と C7 には 2125 サイズのチップセラミックコンデンサを実装すると、P4_3、P4_4 に 32.768kHz のクロックが入力されます。C6、C7 には水晶振動子にあった容量のセラミックコンデンサを実装してください。エプソントヨコム(株)の C-002RX の場合、10～20pF 程度です。

■接続例

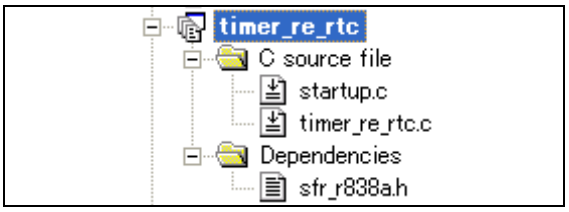
実習基板 Ver.2 を使ったときの接続例を下記に示します。



■操作方法

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

31.3 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。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥timer_re_rtc¥startup.c
2	timer_re_rtc.c	実際に制御するプログラムが書かれています。R8C/38A の内蔵周辺機能(SFR)の初期化も行います。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥timer_re_rtc¥timer_re_rtc.c
3	sfr_r838a.h	R8C/38A マイコンの内蔵周辺機能を制御するためのレジスタ（Special Function Registers)を定義したファイルです。 ファイルの位置→C:¥Workspace¥common_r8c38a¥sfr_r838a.h

31.4 プログラム「timer_re_rtc.c」

```
 1 :  /***********************************************************************/
 2 :  /* 対象マイコン R8C/38A                                                */
 3 :  /* ﾌｧｲﾙ内容     タイマREによるリアルタイムクロック(時計機能)           */
 4 :  /* バージョン   Ver.1.00                                               */
 5 :  /* Date         2011.05.16                                             */
 6 :  /* Copyright    ジャパンマイコンカーラリー実行委員会                   */
 7 :  /***********************************************************************/
 8 :  /*
 9 :  出力：P2_7-P2_0(実習基板Ver.2のLED部)
10 :
11 :  ポート2に接続した実習基板Ver.2のLED部分に秒を出力します。
12 :  X2の時計用クリスタル,C6,7のセラミックコンデンサはオプションです。
13 :  これらの部品を実装していないと本プログラムは動作しません。
14 :  */
15 :
16 :  /*====================================*/
17 :  /* インクルード                       */
18 :  /*====================================*/
19 :  #include "sfr_r838a.h"                 /* R8C/38A 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 :  /* グローバル変数の宣言               */
32 :  /*====================================*/
33 :  unsigned long   cnt_re;                /* タイマRE用                    */
34 :
```

```
 35 :  /************************************************************************/
 36 :  /* メインプログラム                                                     */
 37 :  /************************************************************************/
 38 :  void main( void )
 39 :  {
 40 :      init();                             /* 初期化                       */
 41 :      asm(" fset I ");                    /* 全体の割り込み許可           */
 42 :
 43 :      while( 1 ) {
 44 :          p2 = tresec;
 45 :  //        p2 = cnt_re;
 46 :      }
 47 :  }
 48 :
 49 :  /************************************************************************/
 50 :  /* R8C/38A 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 :      /* XCINクロック(時計クロック)を有効にする */
 65 :      xcsel = 1;                          /* XCINをP4_3,XCOUTをP4_4に設定 */
 66 :      prc0  = 1;                          /* プロテクト解除               */
 67 :      cm03  = 0;                          /* XCINクロック発振             */
 68 :      cm04  = 1;                          /* XCIN-XCOUTピンを有効にする   */
 69 :      prc0  = 0;                          /* プロテクトON                 */
 70 :
 71 :      /* ポートの入出力設定 */
 72 :      prc2 = 1;                           /* PD0のプロテクト解除          */
 73 :      pd0 = 0x00;                         /*                              */
 74 :      pd1 = 0xd0;                         /* 5:RXD0 4:TXD0 3-0:DIP SW     */
 75 :      pd2 = 0xff;                         /* 7-0:LED                      */
 76 :      pd3 = 0xff;                         /*                              */
 77 :      p4  = 0x20;                         /* P4_5のLED:初期は点灯         */
 78 :      pd4 = 0xa0;                         /* 7:XOUT 6:XIN 5:LED 4:XCOUT 3:XCIN 2:VREF */
 79 :      pd5 = 0xff;                         /*                              */
 80 :      pd6 = 0xff;                         /*                              */
 81 :      pd7 = 0xff;                         /*                              */
 82 :      pd8 = 0xff;                         /*                              */
 83 :      pd9 = 0x3f;                         /*                              */
 84 :      pur0 = 0x04;                        /* P1_3～P1_0のプルアップON     */
 85 :
 86 :      /* タイマREの設定 */
 87 :      trecr1 = 0x10;                      /* タイマREレジスタクリア       */
 88 :      trecr1 = 0x00;
 89 :      trecsr = 0x08;                      /* RTCクロックモードに選択      */
 90 :      tresec = 0;                         /* 「秒」の初期値を設定         */
 91 :      tremin = 0;                         /* 「分」の初期値を設定         */
 92 :      trehr  = 0;                         /* 「時」の初期値を設定         */
 93 :      trewk  = 0;                         /* 「曜日」の初期値を設定(0=Sun)*/
 94 :      trecr1 = 0x48;                      /* 動作モード:24ｈモード        */
 95 :      trecr2 = 0x01;                      /* 割り込み要因：秒             */
 96 :      treic  = 0x02;                      /* 割り込み優先レベルの設定     */
 97 :      trecr1 |= 0x80;                     /* 動作開始                     */
 98 :  }
 99 :
100 :  /************************************************************************/
101 :  /* タイマRE 割り込み処理                                                */
102 :  /************************************************************************/
103 :  #pragma interrupt intTRE(vect=10)
104 :  void intTRE( void )
105 :  {
106 :      cnt_re++;
107 :  }
108 :
109 :  /************************************************************************/
110 :  /* end of file                                                          */
111 :  /************************************************************************/
```

31.5 プログラムの解説

31.5.1 init 関数(XCIN クロック(32.768kHz)を有効にする)

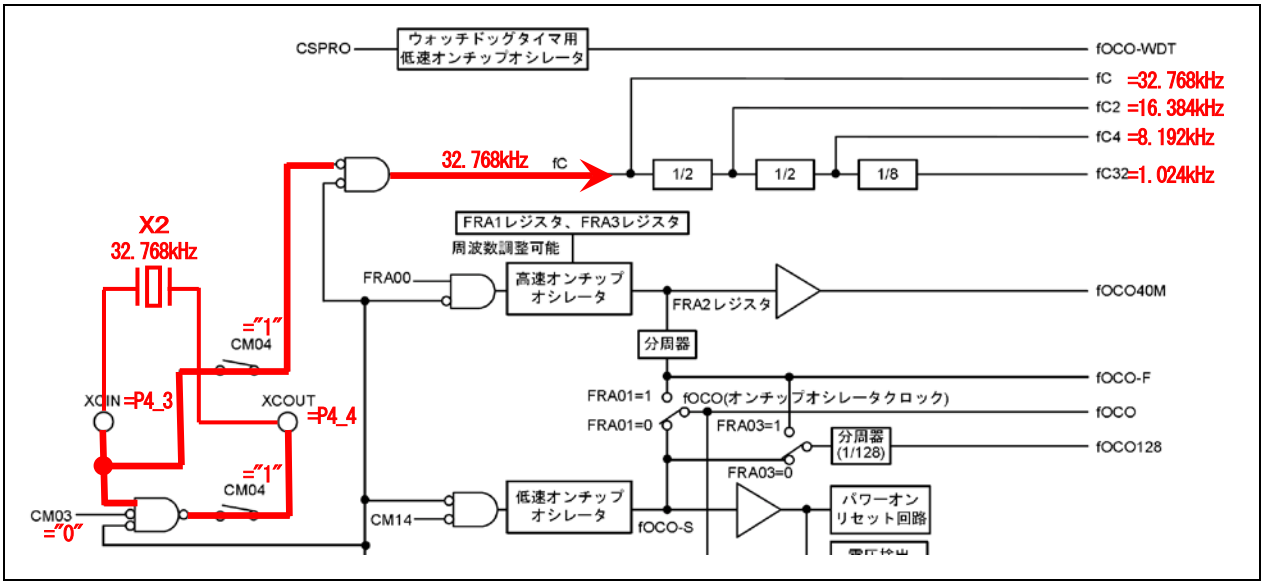
XCIN クロックを有効にするプログラムは、次のようになります。

```
64 :      /* XCINクロック(時計クロック)を有効にする */
65 :      xcsel = 1;                    /* XCINをP4_3,XCOUTをP4_4に設定 */
66 :      prc0  = 1;                    /* プロテクト解除               */
67 :      cm03  = 0;                    /* XCINクロック発振             */
68 :      cm04  = 1;                    /* XCIN-XCOUTピンを有効にする   */
69 :      prc0  = 0;                    /* プロテクトON                 */
```

(1) XCIN クロック

R8C/38A には、XCIN クロック発振回路が内蔵されています。これは、32.768kHz の水晶発振子を XCIN 端子(P4_3)、XCOUT 端子(P4_4)に接続することにより、時計用のクロック(fC、fC2、fC4、fC32)を生成することができます。

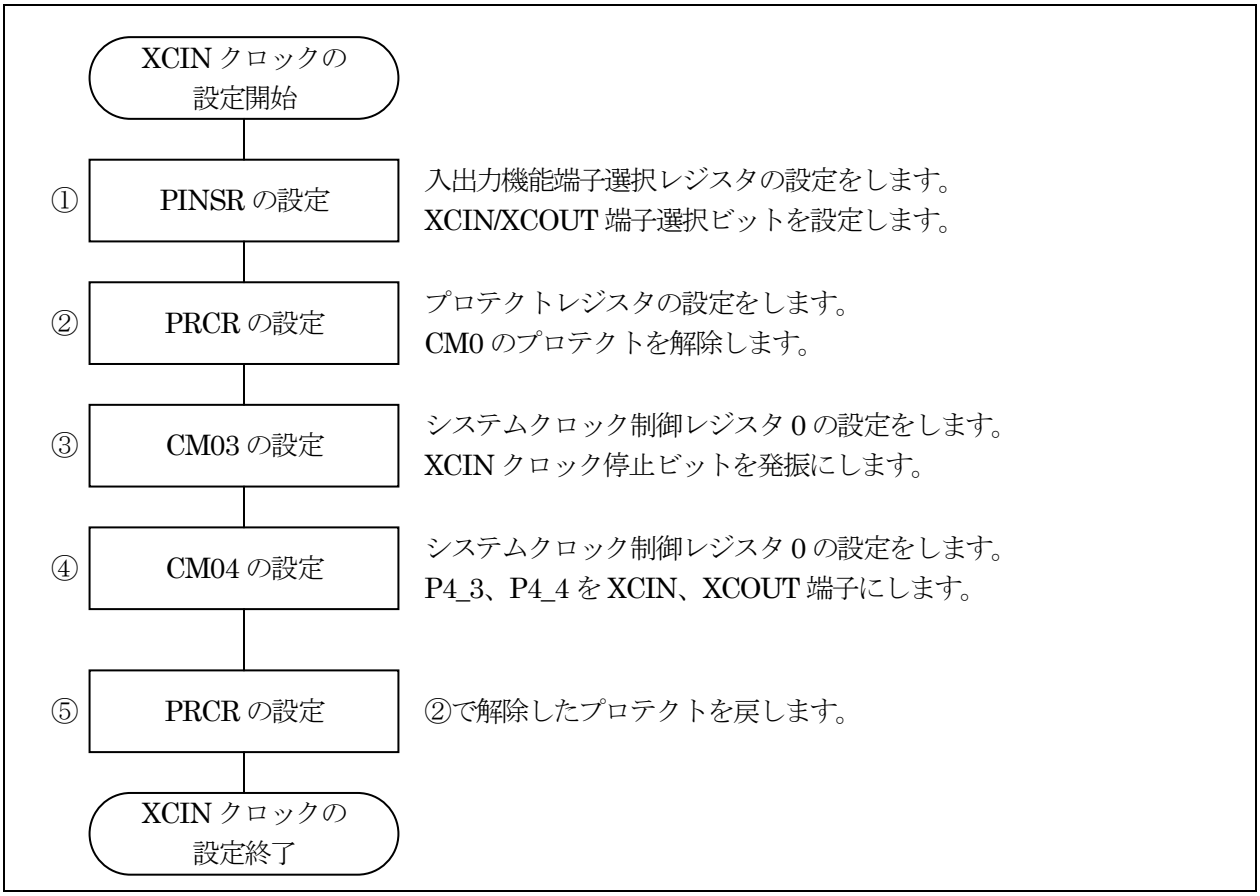
R8C/38A のクロック発生回路のうち、XCIN 部分を下記に示します。



init 関数内の設定を行うことにより、赤線の経路で信号が接続され、fC、fC2、fC4、fC32 からパルスが出力されるようになります。

(2) XCIN クロックの設定

XCIN(P4_3)端子、XCOUT(P4_4)端子は起動時、通常の I/O ポートです。init 関数内のプログラムで XCIN クロック発振回路に切り替えます。レジスタの設定手順を下記に示します。



①入出力機能端子選択レジスタ(PINSR:I/O Function Pin Select Register)の設定

XCIN/XCOUT 端子接続ビットを設定して、XCIN を P4_3、XCOUT を P4_4 に接続します。

設定 bit	上:ビット名 下:シンボル	内容	今回の内容
bit7,6	SDA 端子デジタル遅延選択ビット bit7:sdadly1 bit6:sdadly0	00:3×f1 サイクルのデジタル遅延 01:11×f1 サイクルのデジタル遅延 10:19×f1 サイクルのデジタル遅延 11:設定しないでください	変更なし
bit5	I^2C 転送レート 1/2 倍選択ビット iictchalf	0:ICCR1 レジスタの CKS0～CKS3 ビットの設定値通りの転送レート 1:ICCR1 レジスタの CKS0～CKS3 ビットの設定値の 1/2 倍の転送レート	変更なし
bit4	I^2C 転送レート 2 倍選択ビット iictctwi	0:ICCR1 レジスタの CKS0～CKS3 ビットの設定値通りの転送レート 1:ICCR1 レジスタの CKS0～CKS3 ビットの設定値の 2 倍の転送レート	変更なし
bit3	I/O ポート入力機能選択ビット ioinsel	0:I/O ポートの入力機能は PDi (i=0～9)レジスタに依存 PDi レジスタの PDi_j (j=0～7)ビットが“0”(入力モード)のとき、端子の入力レベルを読む。PDi レジスタの PDi_j ビットが“1”(出力モード)のとき、ポートラッチを読む。 1:I/O ポートの入力機能は PDi レジスタに関係なく、端子の入力レベルを読む	変更なし
bit2,1		″0″を設定	変更なし
bit0	XCIN/XCOUT 端子接続ビット xcsel	0:XCIN を P4_3、XCOUT を P4_4 に接続しない 1:XCIN を P4_3、XCOUT を P4_4 に接続する 水晶振動子を接続するので、″1″に設定します。	1

XCIN/XCOUT 端子接続ビットを″1″にして、XCIN を P4_3、XCOUT を P4_4 に接続します。入出力機能端子選択レジスタ(PINSR)の bit0 のシンボル名は「XCSEL」なので、この名称を使って設定します。

```
65 :      xcsel = 1;                          /* XCIN を P4_3, XCOUT を P4_4 に設定 */
```

②プロテクトレジスタ(PRCR：Protect register)の設定（書き込み許可）

R8C/38A 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。

③、④で CM0 レジスタの設定をします。CM0 レジスタを書き込み許可するためにプロテクトレジスタ(PRCR)のプロテクトビット 0 を″1″にします。プロテクトレジスタ(PRCR)のプロテクトビット 0（bit0）のシンボル名は、「PRC0」なので、この名称を使って設定します。

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

③④システムクロック制御レジスタ 0(CM0：System clock control register0)の設定

XCIN クロックを発振させます。システムクロック制御レジスタ 0（CM0）は、プロテクトレジスタ(PRCR)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)	変更なし
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) ″1″の XCIN、XCOUT 端子に設定します。	④で 1
bit3	XCIN クロック停止ビット cm03	0:発振 1:停止 ″0″の発振に設定します。	③で 0
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き 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。

XCIN クロック停止ビットを″0″にして、XCIN クロックを発振させます。システムクロック制御レジスタ 0(CM0)の XCIN クロック停止ビット(bit3)のシンボル名は「CM03」なので、この名称を使って設定します。

```
67 :        cm03  = 0;                          /* XCIN クロック発振              */
```

次に、ポート XCIN-XCOUT 切り替えビットを"1"にして、P4_3 端子を XCIN 端子に、P4_4 端子を XCOUT 端子に切り替えます。システムクロック制御レジスタ 0(CM0)のポート XCIN-XCOUT 切り替えビット(bit4)のシンボル名は「CM04」なので、この名称を使って設定します。

```
68 :        cm04  = 1;                          /* XCIN-XCOUT ピンを有効にする    */
```

⑤プロテクトレジスタ(PRCR:Protect register)の設定（書き込み禁止）

②で書き込みを許可しました。それを、元に戻し書き込み禁止にします。

```
69 :        prc0  = 0;                          /* プロテクト ON                  */
```

31.5.2 init 関数(ポートの入出力設定)

ポートの入出力方向の設定プログラムを、下記に示します。

```
71 :        /* ポートの入出力設定 */
72 :        prc2 = 1;                           /* PD0のプロテクト解除            */
73 :        pd0 = 0x00;                         /*                                */
74 :        pd1 = 0xd0;                         /* 5:RXD0 4:TXD0 3-0:DIP SW       */
75 :        pd2 = 0xff;                         /* 7-0:LED                        */
76 :        pd3 = 0xff;                         /*                                */
77 :        p4  = 0x20;                         /* P4_5のLED:初期は点灯           */
78 :        pd4 = 0xa0;                         /* 7:XOUT 6:XIN 5:LED 4:XCOUT 3:XCIN 2:VREF */
79 :        pd5 = 0xff;                         /*                                */
80 :        pd6 = 0xff;                         /*                                */
81 :        pd7 = 0xff;                         /*                                */
82 :        pd8 = 0xff;                         /*                                */
83 :        pd9 = 0x3f;                         /*                                */
84 :        pur0 = 0x04;                        /* P1_3～P1_0のプルアップON       */
```

今回の実習で関係するポート 2 とポート 4 の入出力設定を、下表に示します。

ポート	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
2	実習基板 Ver.2 LED 出力	実習基板 Ver.2 LED 出力	実習基板 Ver.2 LED 出力	実習基板 Ver.2 LED 出力	実習基板 Ver.2 LED 出力	実習基板 Ver.2 LED 出力	実習基板 Ver.2 LED 出力	実習基板 Ver.2 LED 出力
4	クリスタル (XOUT 端子) 出力	クリスタル (XIN 端子) 入力	RY_R8C38 ボード上の LED 出力	時計用 クリスタル (XCOUT 端子) **入力**	時計用 クリスタル (XCIN 端子) **入力**	Vcc 入力		

システムクロック制御レジスタ 0 の注 6 の説明に「**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**」とあるので、P4_3 端子、P4_4 端子は入力に設定します。
ポート 2 は、実習基板 Ver.2 の LED 部に接続するので、出力に設定します。

31.5.3 init 関数(タイマ RE の設定)

タイマ RE を使って、リアルタイムクロック(時計機能)を動作させます。

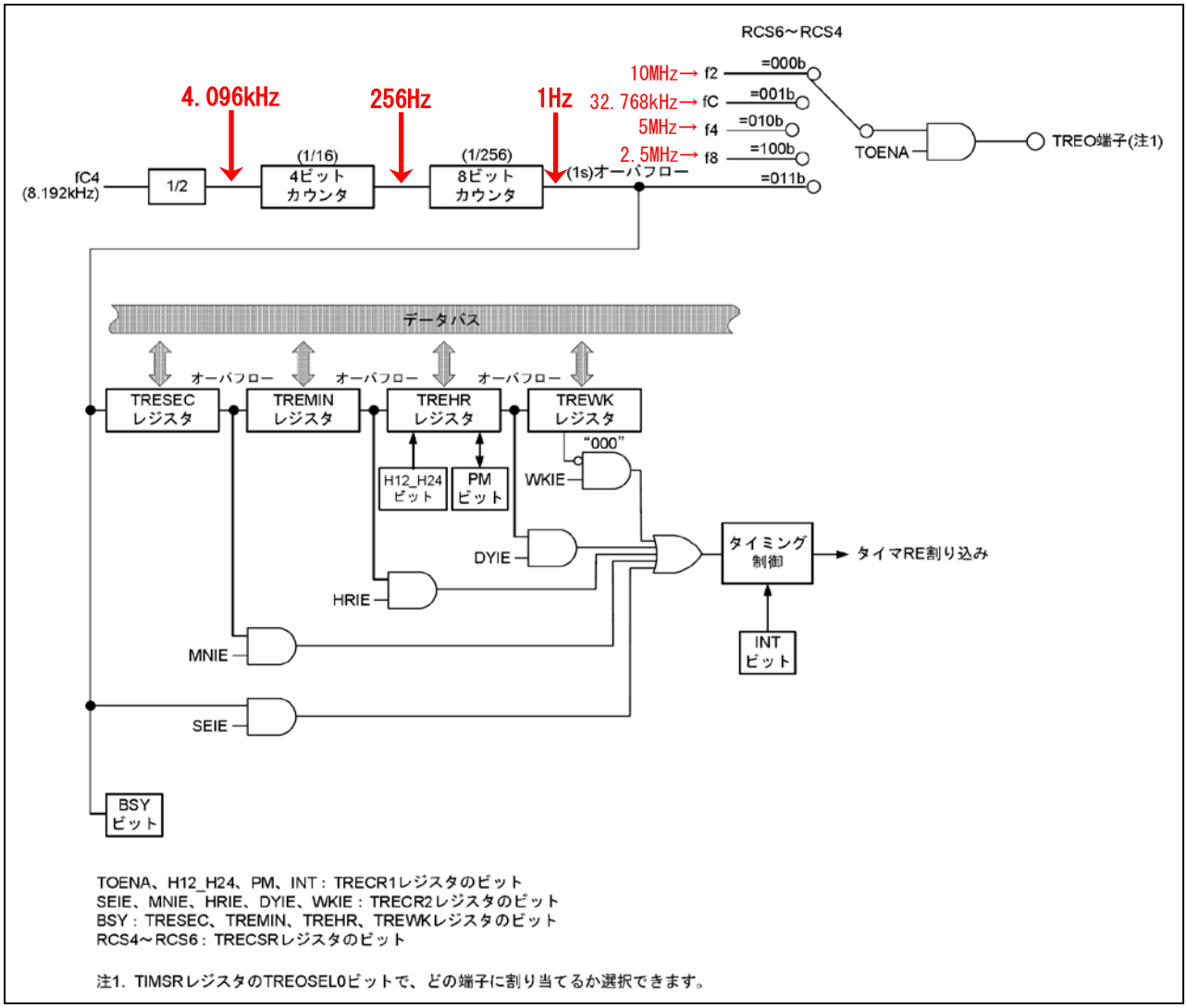
```
64 :      /* XCINクロック(時計クロック)を有効にする */
65 :      pinsr = 0x01;                        /* XCINをP4_3,XCOUTをP4_4に設定 */
66 :      prc0  = 1;                           /* プロテクト解除               */
67 :      cm03  = 0;                           /* XCINクロック発振             */
68 :      cm04  = 1;                           /* XCIN-XCOUTピンを有効にする   */
69 :      prc0  = 0;                           /* プロテクトON                 */
```

(1) タイマ RE とは

R8C/38A には、タイマ RE というタイマが 1 個内蔵されています。タイマ RE には、次の 2 種類のモードがあります。今回はリアルタイムクロックモードを使い、1 秒ごとに割り込みを発生させます。

モード	詳細
リアルタイムクロックモード	**fC4 から 1s を作り、秒、分、時、曜日をカウントするモードです。**
アウトプットコンペアモード	カウントソースをカウントし、コンペア一致を検出するモードです。

(2) タイマ RE のブロック図



※タイマ RE の端子構成

端子名	割り当てる端子	入出力	機能
TREO	P0_4 または P6_0	出力	リアルタイムクロックモードのときは、f2(10MHz)、f4(5MHz)、f8(2.5MHz)、fC(32.768 kHz)、1 秒のパルスを出力することができます。

今回は、TREO 端子は使いません。

(3) タイマ RE の設定(リアルタイムクロックモード)

タイマ RE のレジスタの設定手順を下記に示します。

①②タイマ RE 制御レジスタ1（TRECR1：Timer RE Control Register 1)の設定

タイマ RE に関するレジスタをリセットします。

設定 bit	上：ビット名 下：シンボル	内容	今回の内容
bit7	タイマ RE カウント開始ビット tstart_trecr1	0：カウント停止 1：カウント開始	変更なし
bit6	H12_H24 動作モード選択ビット h12_h24_trecr1	0：12 時間モード 1：24 時間モード	変更なし
bit5	午前/午後ビット pm_trecr1	12_H24 ビットが“0”(12 時間モード)のとき(注 1) 0：午前 1：午後 H12_H24 ビットが“1”(24 時間モード)のとき、不定	変更なし
bit4	タイマ RE リセットビット trerst_trecr1	このビットを“1”にした後、“0”にすると次の状態になります。 ·TRESEC、TREMIN、TREHR、TREWK、TRECR2 レジスタが“00h” ·TRECR1 レジスタの TCSTF、INT、PM、H12_H24、TSTART ビットが“0” ·8 ビットカウンタが“00h”、4 ビットカウンタが“0h”	1 ↓ 0
bit3	INT 割り込み要求タイミングビット int_trecr1	リアルタイムクロックモードでは“1”にしてください	変更なし
bit2	TOENA TREO 端子出力許可ビット toena_trecr1	0：クロック出力禁止 1：クロック出力許可	変更なし
bit1	TCSTF タイマ RE カウントステータスフラグ tcstf_trecr1	0：カウント停止中 1：カウント中	変更なし
bit0		″0″を設定	0

タイマ RE 制御レジスタ 1（TRECR1)のタイマ RE リセットビットを″1″にした後、″0″にして、タイマ RE に関するレジスタを初期化します。

```
87 :      trecr1 = 0x10;                    /* タイマREレジスタクリア        */
88 :      trecr1 = 0x00;
```

③タイマ RE カウントソース選択レジスタ(TRECSR:Timer RE Count Source Select Register)の設定

タイマ RE をリアルタイムクロックモードにします。

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
bit6～4	クロック出力選択ビット(注 1) bit6:rcs6_trecsr bit5:rcs5_trecsr bit4:rcs4_trecsr	000:f2 001:fC 010:f4 011:1Hz 100:f8 上記以外:設定しないでください 今回は、クロック出力をしないので何を設定しても構いません。″000″にしておきます。	000
bit3	リアルタイムクロックモード選択ビット rcs3_trecsr	リアルタイムクロックモードでは“1”にしてください	1
bit2	4 ビットカウンタ選択ビット rcs2_trecsr	リアルタイムクロックモードでは“0”にしてください	0
bit1,0	カウントソース選択ビット bit1:rcs1_trecsr bit0:rcs0_trecsr	リアルタイムクロックモードでは“00b”にしてください	00

注 1. RCS4～RCS6 ビットは、TRECR1 レジスタの TOENA ビットが“0”(クロック出力禁止)のとき、書いてください。

タイマ RE カウントソース選択レジスタ(TRECSR)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			

④タイマ RE 秒データレジスタ(TRESEC：Timer RE Second Data Register)の設定

「秒」の初期値を設定します。

設定 bit	上：ビット名 下：シンボル	内容	今回の内容
bit7	タイマ RE ビジーフラグ bsy_tresec	TRESEC、TREMIN、TREHR、TREWK レジスタが更新中、“1”になります このビットが″1″のときは、TRESEC の値を読み込まないでください。	0
bit6～4	秒十位カウントビット bit6:sc12_tresec bit5:sc11_tresec bit4:sc10_tresec	0 から 5 をカウントして、60 秒をカウント 設定範囲：0～5(BCD コード)	000
bit3～0	秒一位カウントビット bit3:sc03_tresec bit2:sc02_tresec bit1:sc01_tresec bit0:sc00_tresec	1 秒ごとに 0 から 9 をカウント。桁上がりが発生すると、秒十位が 1 加算される。 設定範囲：0～9(BCD コード)	0000

今回は、初期値を 0 秒にします。タイマ RE 秒データレジスタ(TRESEC)には、0 を設定します。

```
90 :      tresec = 0;                    /* 「秒」の初期値を設定          */
```

※タイマ RE 秒データレジスタ(TRESEC)の扱う値

このレジスタの値は、BCD コードで扱います。BCD コードは「Binary Coded Decimal」の略で、日本語では「二進化十進数」といいます。BCD コードとは、「2 進数の値を 4 桁用いて、10 進数の 1 桁分の値を表現すること」です。

10 進数、16 進数、BCD コードの対応表を下記に示します。

10 進数表記	0	1	2	3	…	9	10	11	12	…	18	19	20	21	22
16 進数表記	0x00	0x01	0x02	0x03	…	0x09	0x0a	0x0b	0x0c	…	0x12	0x13	0x14	0x15	0x16
BCD コード表記	0x00	0x01	0x02	0x03	…	0x09	**0x10**	**0x11**	**0x12**	…	**0x18**	**0x19**	**0x20**	**0x21**	**0x22**

BCD コードでは、0x09 の次は、0x10 となります。

16 進数で 0x16 のとき、10 進数では 22 ですが、すぐには 0x16＝22 と思い浮かばないと思います。BCD コードを使えば、BCD コードが 0x22 のとき 10 進数は 22 となり直感的に分かりやすくなります。

このように BCD コードは、16 進数を 10 進数のように扱います。BCD コードでは、a～f を含む値は扱いません。

BCD コードを使うと、値が直感的に分かりやすくなることを説明しました。さらにプログラムでも扱いやすくなります。たとえば、data 変数に入っている値が 35 のとき、10 の桁の「3」と、1 の桁の「5」に分けるプログラムを作るとき、BCD コードで扱わないときと、扱ったときのプログラムを下記に示します。

```
// data 変数を BCD コードで扱わない場合
sec10 = data / 10;    // 割り算は時間がかかる
sec1  = data % 10;    // 「%」は剰余算演算子で余りを求めます。こちらも割り算を行って余りを
                      // 求めるので時間がかかります。

// data 変数を BCD コードで扱った場合
sec10 = data >> 4;    // 10 の位は、bit7～4 なので 4 ビット右にシフトすれば 10 の桁が分かる
sec1  = data & 0x0f   // 1 の位は、bit3～0 なので bit3～0 のみ有効にすれば 1 の桁が分かる
```

10 の桁と 1 の桁に分ける場合、BCD コードでない場合は割り算します。これは処理時間がかかると共に、プログラムサイズも大きくなります。BCD コードの場合は上位 4 桁が 10 の位、下位 4 桁が 1 の位なので、AND 演算やビットシフトで簡単に求まります。こちらは処理が高速ですしプログラムサイズも小さくなります。

BCD コードのデメリットは、通常の 8bit 幅のレジスタであれば 0～255 までの値を扱えますが、BCD コードだと、0x00～0x99 までの 100 個しか扱えなくなります。

もし、タイマ RE 秒データレジスタ(TRESEC)に 31 秒を設定したい場合、下記のようになります。

```
90 :      tresec = 0x31;     // BCD コードで設定
          tresec = 31;       // 16 進数で 0x1f になるので、この設定は間違い!!
```

⑤タイマ RE 分データレジスタ(TREMIN：Timer RE Minute Data Register)の設定

「分」の初期値を設定します。

設定 bit	上:ビット名 下:シンボル	内容	今回の内容
bit7	タイマ RE ビジーフラグ bsy_tremin	TRESEC、TREMIN、TREHR、TREWK レジスタが更新中、“1”になります このビットが″1″のときは、TREMIN の値を読み込まないでください。	0
bit6～4	分十位カウントビット bit6:mn12_tremin bit5:mn11_tremin bit4:mn10_tremin	0 から 5 をカウントして、60 分をカウント 設定範囲:0～5(BCD コード)	000
bit3～0	分一位カウントビット bit3:mn03_tremin bit2:mn02_tremin bit1:mn01_tremin bit0:mn00_tremin	1 分ごとに 0 から 9 をカウント。桁上がりが発生すると、分十位が 1 加算される。 設定範囲:0～9(BCD コード)	0000

今回は、初期値を 0 分にします。タイマ RE 分データレジスタ(TREMIN)には、0 を設定します。

```
91 :      tremin = 0;                    /* 「分」の初期値を設定          */
```

⑥タイマ RE 時データレジスタ(TREHR:Timer RE Hour Data Register)の設定

「時」の初期値を設定します。

設定 bit	上:ビット名 下:シンボル	内容	今回の内容
bit7	タイマ RE ビジーフラグ bsy_trehr	TRESEC、TREMIN、TREHR、TREWK レジスタが更新中、“1”になります このビットが″1″のときは、TREHR の値を読み込まないでください。	0
bit6		″0″を設定	0
bit5～4	時十位カウントビット bit5:hr11_trehr bit4:hr10_trehr	H12_H24 ビットが“0”(12 時間モード)のとき、0 から 1 をカウント。 H12_H24 ビットが“1”(24 時間モード)のとき、0 から 2 をカウント。 設定範囲:0～2(BCD コード)	00
bit3～0	時一位カウントビット bit3:hr03_trehr bit2:hr02_trehr bit1:hr01_trehr bit0:hr00_trehr	1 時間ごとに 0 から 9 をカウント。桁上がりが発生すると、時十位が 1 加算される。 設定範囲:0～9(BCD コード)	0000

今回は、初期値を 0 時にします。タイマ RE 時データレジスタ(TREHR)には、0 を設定します。

```
92 :      trehr  = 0;                    /* 「時」の初期値を設定          */
```

⑦タイマ RE 曜日データレジスタ(TREWK：Timer RE Day of Week Data Register)の設定

「曜日」の初期値を設定します。

設定 bit	上：ビット名 下：シンボル	内容	今回の内容
bit7	タイマ RE ビジーフラグ bsy_trewk	TRESEC、TREMIN、TREHR、TREWK レジスタが更新中、“1”になります このビットが″1″のときは、TREWK の値を読み込まないでください。	0
bit6～3		″0000″を設定	0000
bit2～0	曜日カウントビット bit2:wk2_trewk bit1:wk1_trewk bit0:wk0_trewk	000：日 001：月 010：火 011：水 100：木 101：金 110：土 111：設定しないでください	000

今回は、初期値を日曜日にします。タイマ RE 曜日データレジスタ(TREWK)には、0 を設定します。

```
93 :        trewk  = 0;                          /* 「曜日」の初期値を設定(0=Sun)*/
```

⑧タイマ RE 制御レジスタ 1（TRECR1：Timer RE Control Register 1)の設定

タイマ RE に関する動作モード、TREO 端子出力の設定を行います。

設定 bit	上：ビット名 下：シンボル	内容	今回の内容
bit7	タイマ RE カウント開始ビット tstart_trecr1	0：カウント停止 1：カウント開始	0
bit6	H12_H24 動作モード選択ビット h12_h24_trecr1	0：12 時間モード 1：24 時間モード 今回は、24 時間モードにします。	1
bit5	午前/午後ビット pm_trecr1	12_H24 ビットが“0”(12 時間モード)のとき(注 1) 0：午前 1：午後 H12_H24 ビットが“1”(24 時間モード)のとき、不定 今回は 24 時間モードに設定しているので設定の必要はありませんが、″0″を設定しておきます。	0
bit4	タイマ RE リセットビット trerst_trecr1	このビットを“1”にした後、“0”にすると次の状態になります。 ·TRESEC、TREMIN、TREHR、TREWK、TRECR2 レジスタが“00h” ·TRECR1 レジスタの TCSTF、INT、PM、H12_H24、TSTART ビットが“0” ·8 ビットカウンタが“00h”、4 ビットカウンタが“0h”	0
bit3	INT 割り込み要求タイミングビット int_trecr1	リアルタイムクロックモードでは“1”にしてください	1
bit2	TOENA TREO 端子出力許可ビット toena_trecr1	0：クロック出力禁止 1：クロック出力許可 今回は、信号出力しません。	0
bit1	TCSTF タイマ RE カウントステータスフラグ tcstf_trecr1	0：カウント停止中 1：カウント中	0
bit0		″0″を設定	0

タイマ RE 制御レジスタ 1（TRECR1)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	**1**	0	0	0
16 進数	4				8			

⑨タイマ RE 制御レジスタ 2（TRECR2：Timer RE Control Register 2)の設定

秒、分、時、日、曜日の、どの周期で割り込みを発生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
bit7,6		″00″を設定	00
bit5	コンペア一致割り込み許可ビット comie_trecr2	リアルタイムクロックモードでは“0”にしてください	0
bit4	週周期割り込み許可ビット(注 1) wkie_trecr2	0：週周期割り込み禁止 1：週周期割り込み許可	0
bit3	日周期割り込み許可ビット(注 1) dyie_trecr2	0：日周期割り込み禁止 1：日周期割り込み許可	0
bit2	時周期割り込み許可ビット(注 1) hrie_trecr2	0：時周期割り込み禁止 1：時周期割り込み許可	0
bit1	分周期割り込み許可ビット(注 1) mnie_trecr2	0：分周期割り込み禁止 1：分周期割り込み許可	0
bit0	秒周期割り込み許可ビット(注 1) seie_trecr2	0：秒周期割り込み禁止 1：秒周期割り込み許可 今回は、1 秒ごとに割り込みが発生するように設定します。	1

注 1. 複数の許可ビットを“1”(割り込み許可)にしないでください。

タイマ RE 制御レジスタ 2（TRECR2)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			

⑩タイマ RE 割り込み制御レジスタ（TREIC：Timer RE Interrupt Control Register)の設定

タイマ RE の割り込み優先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
bit7～4		″0″を設定	0000
bit3	割り込み要求ビット ir_treic	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_treic bit1: ilvl1_treic bit0: ilvl0_treic	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。今回は、低めのレベル 2 に設定します。 ※「printf_lib.c」の送信割り込みがレベル 1 です（今回は使っていません）。その次に高い値である 2 にしました。	010

タイマ RE 割り込み制御レジスタ(TREIC)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			

⑪タイマ RE 制御レジスタ 1（TRECR1：Timer RE Control Register 1)の設定

タイマ RE 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	タイマ RE カウント開始ビット tstart_trecr1	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	H12_H24 動作モード選択ビット h12_h24_trecr1	0：12 時間モード 1：24 時間モード	変更なし
bit5	午前/午後ビット pm_trecr1	12_H24 ビットが“0”(12 時間モード)のとき(注 1) 0：午前 1：午後 H12_H24 ビットが“1”(24 時間モード)のとき、不定	変更なし
bit4	タイマ RE リセットビット trerst_trecr1	このビットを“1”にした後、“0”にすると次の状態になります。 ・TRESEC、TREMIN、TREHR、TREWK、TRECR2 レジスタが“00h” ・TRECR1 レジスタの TCSTF、INT、PM、H12_H24、TSTART ビットが“0” ・8 ビットカウンタが“00h”、4 ビットカウンタが“0h”	変更なし
bit3	INT 割り込み要求タイミングビット int_trecr1	リアルタイムクロックモードでは“1”にしてください	変更なし
bit2	TOENA TREO 端子出力許可ビット toena_trecr1	0：クロック出力禁止 1：クロック出力許可	変更なし
bit1	TCSTF タイマ RE カウントステータスフラグ tcstf_trecr1	0：カウント停止中 1：カウント中	変更なし
bit0		″0″を設定	0

タイマ RE 制御レジスタ 1（TRECR1)の設定値を下記に示します。bit7 のみ″1″にするので、0x80 で OR 演算しています。

bit	7	6	5	4	3	2	1	0
現在の設定値	**1**	?	?	?	?	?	?	?
OR 値	**1**	0	0	0	0	0	0	0
結果	**1**	変更なし	変更なし	変更なし	変更なし	変更なし	変更なし	変更なし

31.5.4 intTRE 関数(1 秒ごとに実行される関数)

先の設定で、タイマ RE 秒データレジスタ(TRESEC1)の値が変わるごとに(1 秒ごとに)、割り込みが発生するように設定しました。intTRE 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。

```
100 :  /************************************************************************/
101 :  /* タイマRE 割り込み処理                                                */
102 :  /************************************************************************/
103 :  #pragma interrupt intTRE(vect=10)
104 :  void intTRE( void )
105 :  {
106 :      cnt_re++;
107 :  }
```

103 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。タイマ RE 割り込みは表より、10 番です。 よって、10 番の割り込みが発生したときに intTRE 関数を実行するよう、「#pragma interrupt」で設定します。
104 行	タイマ RE 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。
106 行	cnt_re 変数を+1 します。この関数は 1 秒ごとに実行されるので、cnt_re は 1 秒ごとに+1 されることになります。

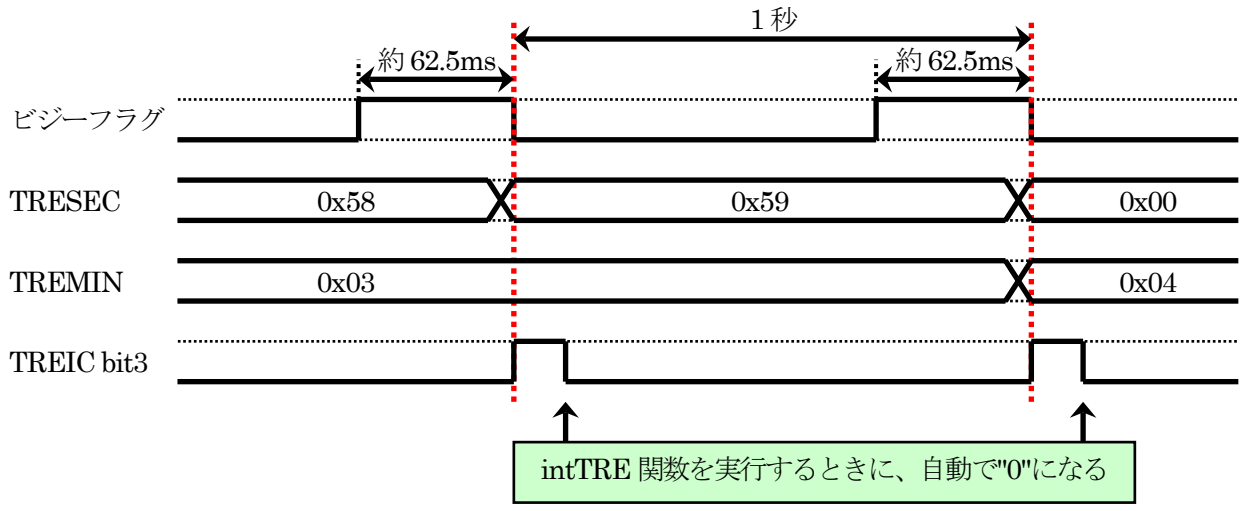
31.5.5 main 関数

```
35 :  /************************************************************************/
36 :  /* メインプログラム                                                     */
37 :  /************************************************************************/
38 :  void main( void )
39 :  {
40 :      init();                             /* 初期化                       */
41 :      asm(" fset I ");                    /* 全体の割り込み許可           */
42 :
43 :      while( 1 ) {
44 :          p2 = tresec;
45 :  //        p2 = cnt_re;
46 :      }
47 :  }
```

41 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 RE 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。
44 行	タイマ RE 秒データレジスタ(TRESEC)の値をポート 2 に接続している LED に出力します。 BCD コードであること、bit7 のタイマ RE ビジーフラグの点灯の仕方をよく見てみてください。

31.5.6 ビジーフラグと値の読み込み方

ビジーフラグは各レジスタの値が更新中であることを示しています。ビジーフラグが"1"のときは値を読み出さないでください。ビジーフラグ、タイマ RE 秒データレジスタ(TRESEC)、タイマ RE 分データレジスタ(TREMIN)、タイマ RE 割り込み制御レジスタ(TREIC)の割り込み要求ビット(bit3)の様子を下図に示します。



タイマ RE 秒データレジスタ(TRESEC)、タイマ RE 分データレジスタ(TREMIN)、タイマ RE 時データレジスタ(TREHR)、タイマ RE 曜日データレジスタ(TREWK)の各レジスタは、1 秒ごとに更新されていきます。値を読み込むときは、ビジーフラグが"0"のときに読み込んでください。連続して値を読み込むときは、各レジスタのビジーフラグをチェック、もし"1"になったなら"0"になるまで待ち、"0"になったら最初から読み直してください。

32.　パルスカウント（プロジェクト：timer_ra_counter）

32.1　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（アンダフローは除く）。マイコンカーでは、ロータリエンコーダのパルスカウントでの使用に最適です。

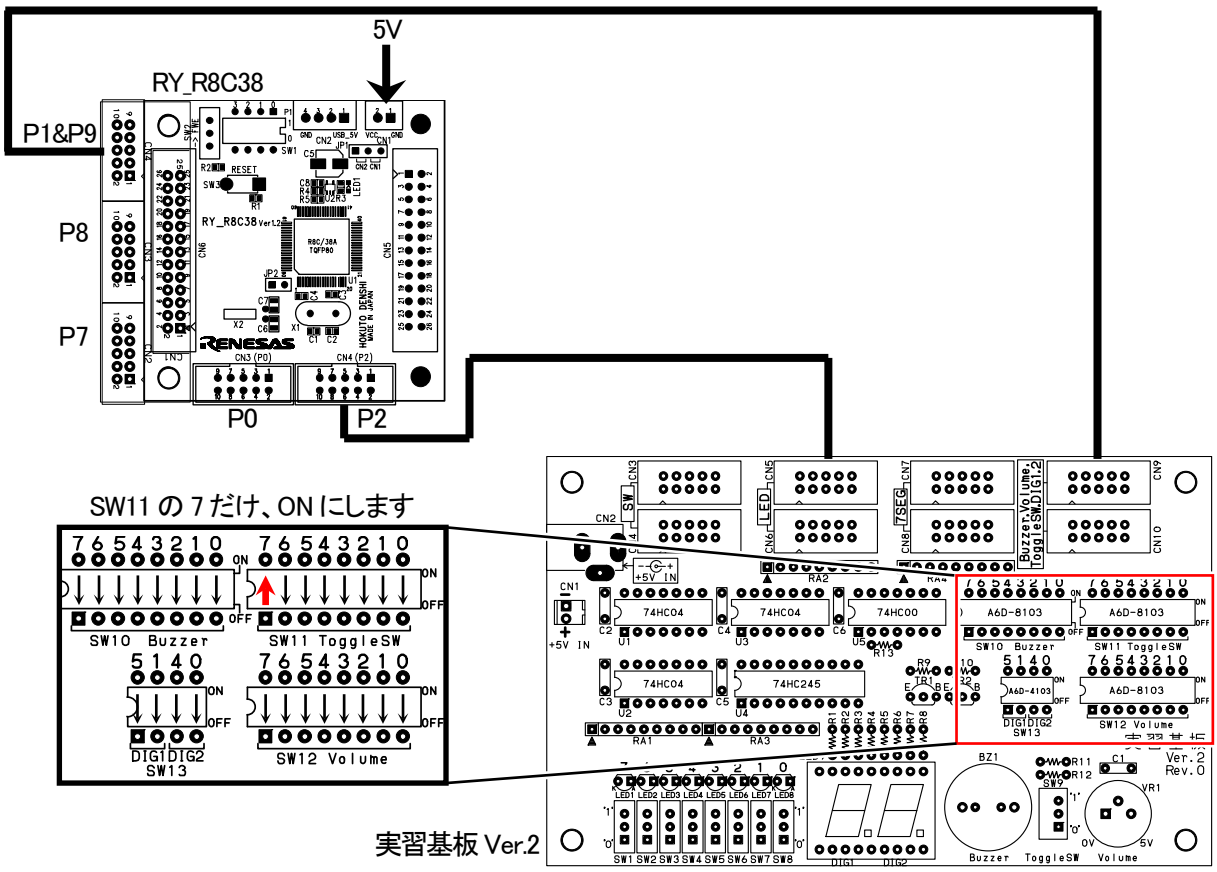
32.2　接続

■使用ポート

マイコンのポート	接続内容
P1_7 (変換基板 CN4)	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。
P2 (CN4)	実習基板 Ver.2 の LED 部など、出力機器を接続します。

■接続例

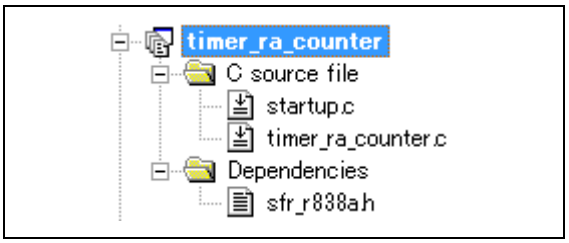
実習基板 Ver.2 を使ったときの接続例を下記に示します。



■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)。

32.3 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。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥timer_ra_counter¥startup.c
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/38A の内蔵周辺機能(SFR)の初期化も行います。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥timer_ra_counter¥timer_ra_counter.c
3	sfr_r838a.h	R8C/38A マイコンの内蔵周辺機能を制御するためのレジスタ（Special Function Registers)を定義したファイルです。 ファイルの位置→C:¥Workspace¥common_r8c38a¥sfr_r838a.h

32.4 プログラム「timer_ra_counter.c」

```
 1 :  /***********************************************************************/
 2 :  /* 対象マイコン R8C/38A                                                */
 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.1.01                                               */
 5 :  /* Date         2013.04.01                                             */
 6 :  /* Copyright    ジャパンマイコンカーラリー実行委員会                   */
 7 :  /***********************************************************************/
 8 :  /*
 9 :  入力：TRAIO端子(P1_7)（エンコーダ、または実習基板Ver.2のトグルスイッチ部）
10 :  出力：P2_7-P2_0(実習基板Ver.2のLED部)
11 :
12 :  P1_7に入力したパルスの数を、実習基板Ver.2のLED部へ出力します。
13 :  */
14 :
15 :  /*====================================*/
16 :  /* インクルード                       */
17 :  /*====================================*/
18 :  #include "sfr_r838a.h"                 /* R8C/38A SFRの定義ファイル    */
19 :
20 :  /*====================================*/
21 :  /* シンボル定義                       */
22 :  /*====================================*/
23 :
24 :  /*====================================*/
25 :  /* プロトタイプ宣言                   */
26 :  /*====================================*/
27 :  void init( void );
28 :
29 :  /***********************************************************************/
30 :  /* メインプログラム                                                    */
31 :  /***********************************************************************/
32 :  void main( void )
33 :  {
34 :      init();                           /* 初期化                       */
35 :
36 :      while( 1 ) {
37 :          p2 = 255 - trapre;            /* パルスカウント下位8bit       */
38 :        //p2 = 255 - tra;               /* パルスカウント上位8bit       */
39 :      }
40 :  }
41 :
42 :  /***********************************************************************/
43 :  /* R8C/38A スペシャルファンクションレジスタ(SFR)の初期化               */
44 :  /***********************************************************************/
45 :  void init( void )
46 :  {
47 :      int i;
48 :
49 :      /* クロックをXINクロック(20MHz)に変更 */
50 :      prc0  = 1;                        /* プロテクト解除               */
51 :      cm13  = 1;                        /* P4_6,P4_7をXIN-XOUT端子にする*/
52 :      cm05  = 0;                        /* XINクロック発振              */
53 :      for(i=0; i<50; i++ );             /* 安定するまで少し待つ(約10ms) */
54 :      ocd2  = 0;                        /* システムクロックをXINにする  */
55 :      prc0  = 0;                        /* プロテクトON                 */
56 :
57 :      /* ポートの入出力設定 */
58 :      prc2 = 1;                         /* PD0のプロテクト解除          */
59 :      pd0 = 0x00;                       /*                              */
60 :      pd1 = 0x50;                       /* 7:パルス入力 5:RXD0 4:TXD0 3-0:DIP SW */
61 :      pd2 = 0xff;                       /* 7-0:LED                      */
62 :      pd3 = 0xfb;                       /* 2:パルス入力                 */
63 :      p4  = 0x20;                       /* P4_5のLED:初期は点灯         */
64 :      pd4 = 0xb8;                       /* 7:XOUT 6:XIN 5:LED 2:VREF    */
65 :      pd5 = 0xff;                       /*                              */
66 :      pd6 = 0xff;                       /*                              */
67 :      pd7 = 0xff;                       /*                              */
68 :      pd8 = 0xff;                       /*                              */
69 :      pd9 = 0x3f;                       /*                              */
70 :      pur0 = 0x04;                      /* P1_3～P1_0のプルアップON     */
71 :
72 :      /* 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 */
73 :      tramr  = 0x02;                    /* イベントカウンタモードに設定 */
74 :      trasr  = 0x01;                    /* TRAIO端子:P1_7に設定         */
75 :      traioc = 0x00;                    /* 立ち上がりエッジでカウント   */
76 :      tracr  = 0x01;                    /* TRAのカウント開始            */
77 :  }
78 :
79 :  /***********************************************************************/
80 :  /* end of file                                                         */
81 :  /***********************************************************************/
```

32.5 プログラムの解説

32.5.1 init 関数(ポートの入出力設定)

今回パルス入力として使用する P1_7 端子を入力に設定します。忘れやすいので、気をつけてください。

```
57 :      /* ポートの入出力設定 */
58 :      prc2 = 1;                    /* PD0のプロテクト解除          */
59 :      pd0 = 0x00;                  /*                              */
60 :      pd1 = 0x50;                  /* 7:パルス入力 5:RXD0 4:TXD0 3-0:DIP SW */
61 :      pd2 = 0xff;                  /* 7-0:LED                      */
62 :      pd3 = 0xfb;                  /* 2:パルス入力                 */
63 :      p4  = 0x20;                  /* P4_5のLED:初期は点灯         */
64 :      pd4 = 0xb8;                  /* 7:XOUT 6:XIN 5:LED 2:VREF    */
65 :      pd5 = 0xff;                  /*                              */
66 :      pd6 = 0xff;                  /*                              */
67 :      pd7 = 0xff;                  /*                              */
68 :      pd8 = 0xff;                  /*                              */
69 :      pd9 = 0x3f;                  /*                              */
70 :      pur0 = 0x04;                 /* P1_3～P1_0のプルアップON     */
```

32.5.2 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。

```
72 :      /* 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 */
73 :      tramr  = 0x02;               /* イベントカウンタモードに設定 */
74 :      trasr  = 0x01;               /* TRAIO端子:P1_7に設定         */
75 :      traioc = 0x00;               /* 立ち上がりエッジでカウント   */
76 :      tracr  = 0x01;               /* TRAのカウント開始            */
```

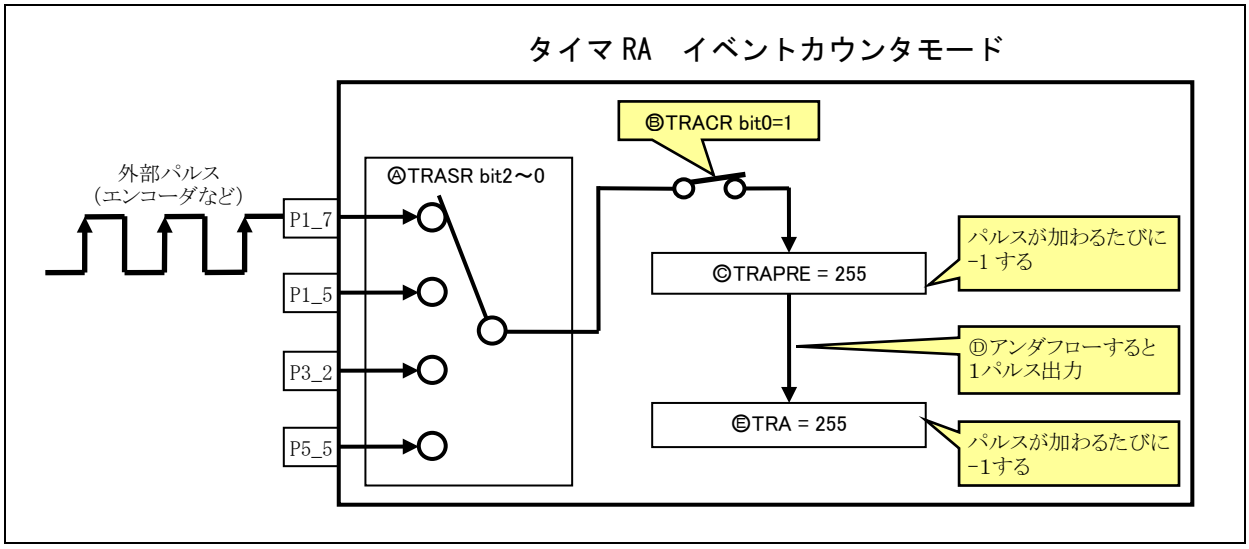
(1) タイマ RA とは

R8C/38A 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
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
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**
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
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

今回は、イベントカウンタモードを使い、外部パルスをカウントします。イベントカウンタモードは、立ち上がりエッジ、立ち下がりエッジでのカウントが可能です。

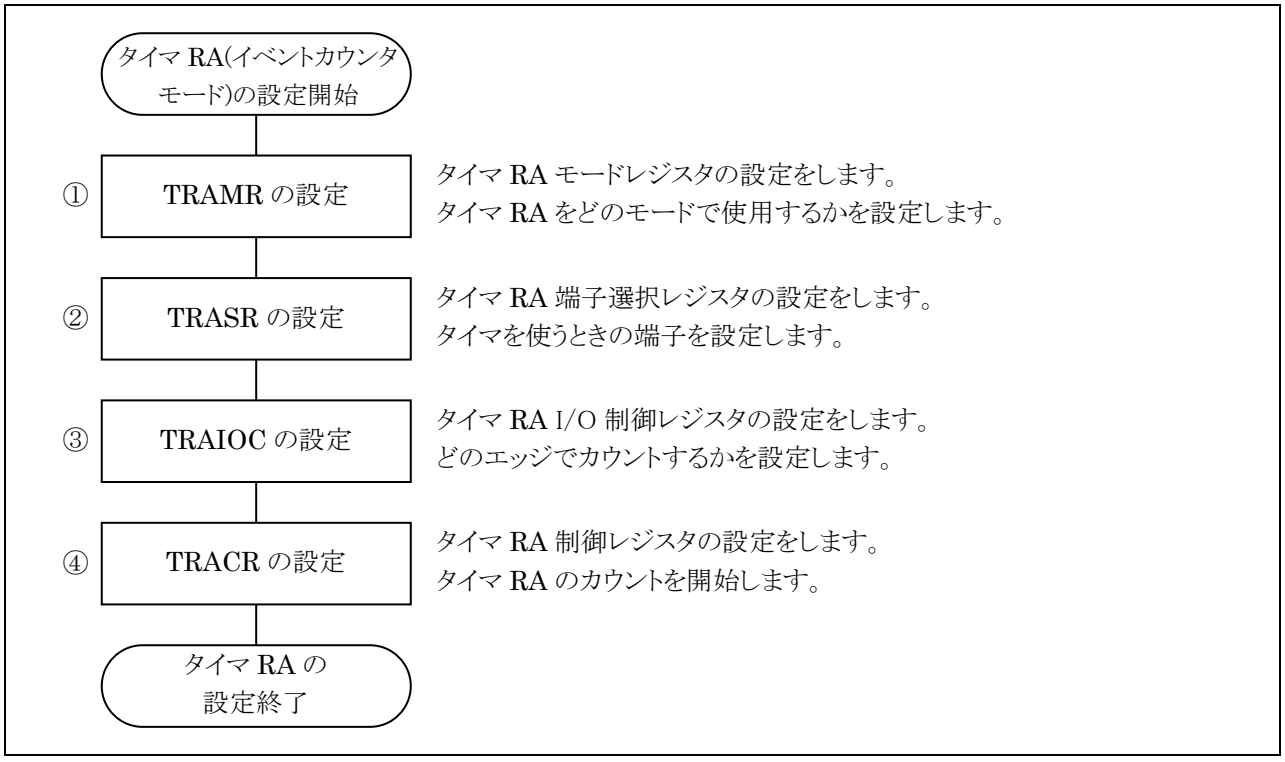
(2) タイマ RA のブロック図



Ⓐ	タイマ RA 端子選択レジスタ（TRASR）の TRAIO 端子選択ビットで、4 つのどの端子からパルスを入力するか設定します。今回は P1_7から外部パルスを入力します。
Ⓑ	タイマ RA 制御レジスタのタイマ RA カウント開始ビットを"1"にするとカウントが開始します。
Ⓒ	タイマ RA プリスケーラレジスタ（TRAPRE）はダウンカウントです。また、タイマ RA は 8bit タイマですので TRAPRE のカウントは、「255→254→253･･･2→1→0」のようにダウンカウントしてきます。カウンタがアンダフロー（0 以下）になると値が 255 に戻ります。
Ⓓ	タイマ RA プリスケーラレジスタ（TRAPRE）がアンダフローになった瞬間、1 パルス出力されます。TRAPRE にパルスが 255 個入ると 1 パルス出力されるということです。
Ⓔ	タイマ RA レジスタ（TRA）はダウンカウントです。また、タイマ RA は 8bit タイマですので、TRA のカウントは、「255→254→253･･･2→1→0」のようにダウンカウントしていきます。TRA は、TRAPRE がアンダフローになった瞬間に出力されるパルスをカウントします。カウンタがアンダフローになると値が 255 に戻ります。

(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、外部からのパルスを数えるように設定します。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 RA カウントソース遮断ビット tckcut	0:カウントソース供給 1:カウントソース遮断 カウントソース供給にします。"0"を設定しておきます。	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 001:f8 010:fOCO 011:f2 100:fC32 101:設定しないでください 110:fC 111:設定しないでください 使いません。	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に設定します。"010"を設定します。	010

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 Pin Select Register)の設定

タイマを使うときの端子を設定します。

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		″0″を設定	0
bit5		″0″を設定	0
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 使いません。	00
bit2～0	TRAIO 端子選択ビット bit2:traiosel2 bit1:traiosel1 bit0:traiosel0	000：TRAIO 端子は使用しない 001：P1_7 に割り当てる 010：P1_5 に割り当てる 011：P3_2 に割り当てる 100：P5_2 に割り当てる 上記以外：設定しないでください 今回は、P1_7 端子からパルスを入力します。″001″を設定します。	001

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**	**0**	**1**
16 進数	**0**				**1**			

③タイマ RA I/O 制御レジスタ（TRAIOC：Timer RA I/O Control Register)の設定

エッジのカウント方法を設定します。

設定 bit	上：ビット名 下：シンボル	内容	今回の内容
bit7～6	TRAIO 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：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 常にイベント入力有効にします。"0"を設定しておきます。	00
bit5～4	TRAIO 入力フィルタ選択ビット（注 1） bit5: tipf1_traioc bit4: 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 フィルタ機能は使いません。"0"を設定しておきます。	00
bit3	ハードウェア LIN 機能選択ビット tiosel_traioc	"0"にしてください "0"を設定をします。	0
bit2	TRAO 出力許可ビット toena_traioc	0：TRAO 出力禁止 1：TRAO 出力 TRAO 端子は使いません。"0"を設定しておきます。	0
bit1	TRAIO 出力制御ビット topcr_traioc	イベントカウンタモードでは、"0"にしてください "0"を設定します。	0
bit0	TRAIO 極性切り替えビット tedgsel_traioc	0：TRAIO 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：TRAIO 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 立ち上がりエッジでカウントします。"0"を設定します。	0

注1. TRAIO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確定します。

注2. INT2のイベント入力有効を使用する場合は、次の設定をしてください。

・INTENレジスタのINT2ENビットを"1"(INT2入力許可)、INT2PLビットを"0"(片エッジ)にする。

・INT2の極性をINT2ICレジスタのPOLビットで選択する。

POLビットを"0"(立ち下がりエッジを選択)にすると、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。

・INT2端子に割り当てたポートのPDi (i=3、6)レジスタのPDi_j (j=2、6)ビットを"0"(入力モード)にする。

・INT2のデジタルフィルタをINTFレジスタのINT2F1～INT2F0ビットで選択する。

なお、INT2ICレジスタのPOLビットとINTENレジスタのINT2PLビットの選択と、INT2端子入力の変更にしたがって、INT2ICレジスタのIRビットが"1"(割り込み要求あり)になります。(ハードウェアマニュアルの「11.8 割り込み使用上の注意」参照)。割り込みの詳細はハードウェアマニュアルの「11. 割り込み」を参照してください。

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	O				O			

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		″0″を設定	0
bit6		″0″を設定	0
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 アンダフローになったかどうかをチェックするフラグです。書く場合は、″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） 使いません。書く場合は、″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″。 使いません。″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 使いません。″0″を設定します。	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 TRA（タイマ RA カウンタ）のカウントを開始します。″1″を設定します。	1

注1 TSTART、TCSTFビットの使用上の注意事項については、「17.8 タイマRA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、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			

32.5.3 main 関数

タイマ RA でパルスカウントを行い、カウントした値をポート 2 へ出力します。

```
29 :  /************************************************************************/
30 :  /* メインプログラム                                                     */
31 :  /************************************************************************/
32 :  void main( void )
33 :  {
34 :      init();                         /* 初期化                           */
35 :
36 :      while( 1 ) {
37 :          p2 = 255 - trapre;          /* パルスカウント下位8bit           */
38 :        //p2 = 255 - tra;             /* パルスカウント上位8bit           */
39 :      }
40 :  }
```

37 行	タイマ RA プリスケーラレジスタ（TRAPRE）は、P1_7 からパルスが入力されるたびに「255→254→253」と値が減っていき、アンダフローすると値が 255 に戻り、カウントを継続します。ダウンカウントのままでは、何パルスカウントしたのか分かりません。そこで、255 から TRAPRE の値を引くと下記のように、パルス数に置き換えることができます。 0 パルス ： 0 = 255 – TRAPER(255) 1 パルス ： 1 = 255 – TRAPER(254) 2 パルス ： 2 = 255 – TRAPER(253) パルス数に置き換えたものを P2 へ出力しています。 P1_7 の信号は、タイマ RA I/O 制御レジスタ（TRAIOC）で「立ち上がりエッジでカウント」を設定しているため、1パルス1カウントします。
38 行	タイマ RA レジスタ（TRA）は、タイマ RA プリスケーラレジスタ（TRAPER）がアンダフローしたとき、「255→254→253」と値が減っていきます。TRA がアンダフローすると、値は 255 に戻り、カウントを継続します。 タイマ RA は 8bit タイマです。タイマ RA（TRA）とタイマ RA プリスケーラレジスタ（TRAPRE）は、それぞれ 0～255 までカウントすることができます。TRA と TRAPRE の関係は、TRA が上位 8bit、TRAPER が下位 8bit のカウント値になります。上位と下位を合わせると 16bit となり、0～65535 までカウントすることができます。 例えば、TRA = 10、TRAPRE = 23、カウントしたとすると、 TRA = 10　　TRAPRE = 23 上位ビット　　下位ビット 0000 1010　　0001 0111 0000 1010 0001 0111　＝　2583 パルス 2583 パルスということが分かります。このように、TRA と TRAPRE を組み合わせることによって、16bit タイマと同じ、0～65535 までカウントすることができます

33.　パルスカウント（プロジェクト：timer_rd_counter）

33.1　概要

本章では、外部から入力したパルスの数をタイマ RD で数える方法を説明します。

マイコンカーは、タイマ RD をリセット同期 PWM モードとして駆動モータ 2 個、ステアリングモータ（サーボモータ）1 個を制御しています。ただし、タイマ RD の一部の機能を使っていません。今回は、リセット同期 PWM モードで使っていない部分を使ってパルスカウントを行います。タイマ RD 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（オーバフローは除く）。マイコンカーでは、ロータリエンコーダのパルスカウントでの使用に最適です。

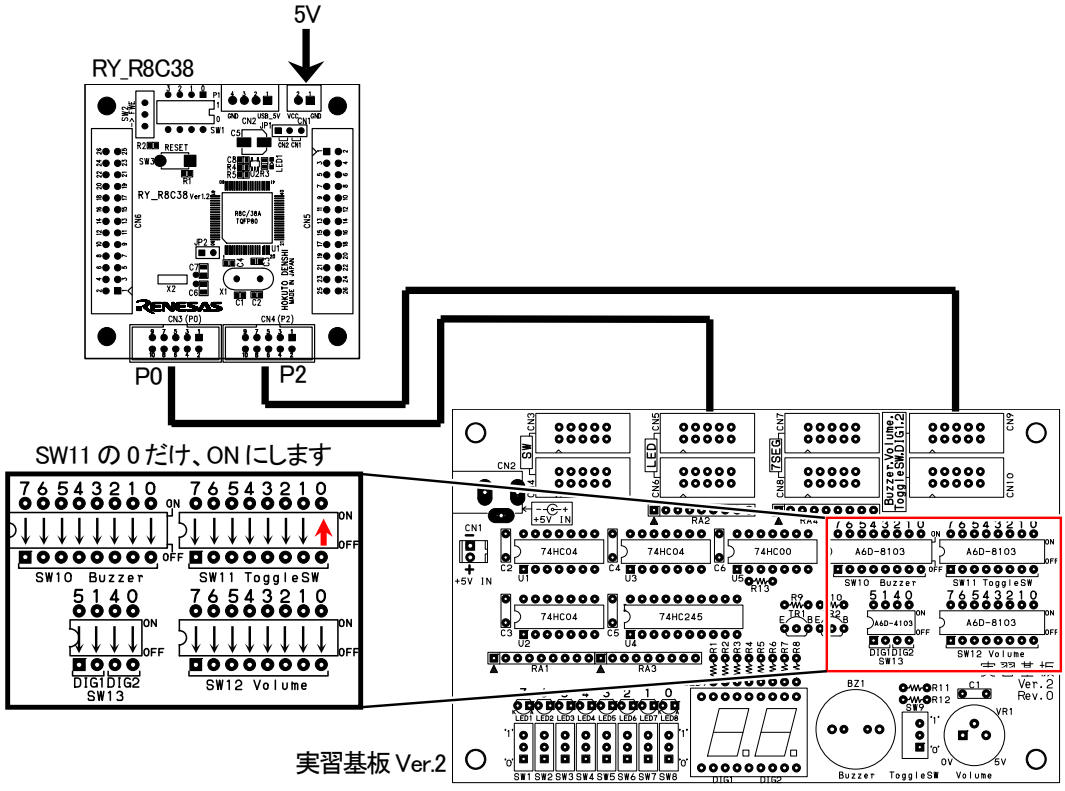
33.2　接続

■使用ポート

マイコンのポート	接続内容
P2_0 (CN4)	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。
P0 (CN3)	実習基板 Ver.2 の LED 部など、出力機器を接続します。

■接続例

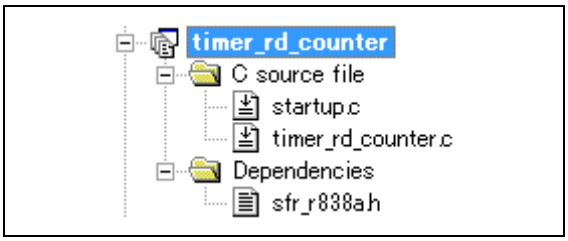
実習基板 Ver.2 を使ったときの接続例を下記に示します。



■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)。

33.3 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。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥timer_rd_counter¥startup.c
2	timer_rd_counter.c	実際に制御するプログラムが書かれています。R8C/38A の内蔵周辺機能(SFR)の初期化も行います。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥timer_rd_counter¥timer_rd_counter.c
3	sfr_r838a.h	R8C/38A マイコンの内蔵周辺機能を制御するためのレジスタ（Special Function Registers)を定義したファイルです。 ファイルの位置→C:¥Workspace¥common_r8c38a¥sfr_r838a.h

33.4 プログラム「timer_rd_counter.c」

```
/***********************************************************************/
/* 対象マイコン R8C/38A                                                */
/* ﾌｧｲﾙ内容     タイマRDによるパルス数計測                             */
/* バージョン   Ver.1.00                                               */
/* Date         2013.04.01                                             */
/* Copyright    ジャパンマイコンカーラリー実行委員会                   */
/***********************************************************************/
/*
入力：TRDCLK端子(P2_0)（エンコーダ、または実習基板Ver.2のトグルスイッチ部）
出力：P0_7-P0_0(実習基板Ver.2のLED部)

P2_0に入力したパルスの数を、実習基板Ver.2のLED部へ出力します。
*/

/*====================================*/
/* インクルード                       */
/*====================================*/
#include "sfr_r838a.h"                  /* R8C/38A SFRの定義ファイル     */

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

/*====================================*/
/* プロトタイプ宣言                   */
/*====================================*/
void init( void );

```

```
29 :  /************************************************************************/
30 :  /* メインプログラム                                                     */
31 :  /************************************************************************/
32 :  void main( void )
33 :  {
34 :      init();                             /* 初期化                       */
35 :
36 :      while( 1 ) {
37 :          p0 = trd1;
38 :      }
39 :  }
40 :
41 :  /************************************************************************/
42 :  /* R8C/38A スペシャルファンクションレジスタ(SFR)の初期化                */
43 :  /************************************************************************/
44 :  void init( void )
45 :  {
46 :      int i;
47 :
48 :      /* クロックをXINクロック(20MHz)に変更 */
49 :      prc0  = 1;                          /* プロテクト解除               */
50 :      cm13  = 1;                          /* P4_6,P4_7をXIN-XOUT端子にする*/
51 :      cm05  = 0;                          /* XINクロック発振              */
52 :      for(i=0; i<50; i++ );               /* 安定するまで少し待つ(約10ms) */
53 :      ocd2  = 0;                          /* システムクロックをXINにする  */
54 :      prc0  = 0;                          /* プロテクトON                 */
55 :
56 :      /* ポートの入出力設定 */
57 :      prc2 = 1;                           /* PD0のプロテクト解除          */
58 :      pd0 = 0xff;                         /*                              */
59 :      pd1 = 0xd0;                         /* 5:RXD0 4:TXD0 3-0:DIP SW     */
60 :      pd2 = 0xfe;                         /* 0:パルス入力                 */
61 :      pd3 = 0xfb;                         /* 2:パルス入力                 */
62 :      p4  = 0x20;                         /* P4_5のLED:初期は点灯         */
63 :      pd4 = 0xb8;                         /* 7:XOUT 6:XIN 5:LED 2:VREF    */
64 :      pd5 = 0xff;                         /*                              */
65 :      pd6 = 0xff;                         /*                              */
66 :      pd7 = 0xff;                         /*                              */
67 :      pd8 = 0xff;                         /*                              */
68 :      pd9 = 0x3f;                         /*                              */
69 :      pur0 = 0x04;                        /* P1_3～P1_0のプルアップON     */
70 :
71 :      /* タイマRD 外部入力（パルスカウント） */
72 :      trdpsr0 |= 0x01;                    /* TRDCLK端子:P2_0に設定        */
73 :      trdfcr  |= 0x40;                    /* 外部クロック入力有効に設定   */
74 :      trdcr1   = 0x15;                    /* TRDCLK端子 両エッジカウント  */
75 :
76 :      /* TRD1カウント開始 */
77 :      trdstr  |= 0x0e;                    /* TRD1カウント開始             */
78 :  }
79 :
80 :  /************************************************************************/
81 :  /* end of file                                                          */
82 :  /************************************************************************/
```

33.5 プログラムの解説

33.5.1 init 関数(ポートの入出力設定)

今回パルス入力として使用する P2_0 端子を、入力に設定します。忘れやすいので、気をつけてください。

```
56 :      /* ポートの入出力設定 */
57 :      prc2 = 1;                           /* PD0のプロテクト解除          */
58 :      pd0 = 0xff;                         /*                              */
59 :      pd1 = 0xd0;                         /* 5:RXD0 4:TXD0 3-0:DIP SW     */
60 :      pd2 = 0xfe;                         /* 0:パルス入力                 */
61 :      pd3 = 0xfb;                         /* 2:パルス入力                 */
62 :      p4  = 0x20;                         /* P4_5のLED:初期は点灯         */
63 :      pd4 = 0xb8;                         /* 7:XOUT 6:XIN 5:LED 2:VREF    */
64 :      pd5 = 0xff;                         /*                              */
65 :      pd6 = 0xff;                         /*                              */
66 :      pd7 = 0xff;                         /*                              */
67 :      pd8 = 0xff;                         /*                              */
68 :      pd9 = 0x3f;                         /*                              */
69 :      pur0 = 0x04;                        /* P1_3～P1_0のプルアップON     */
```

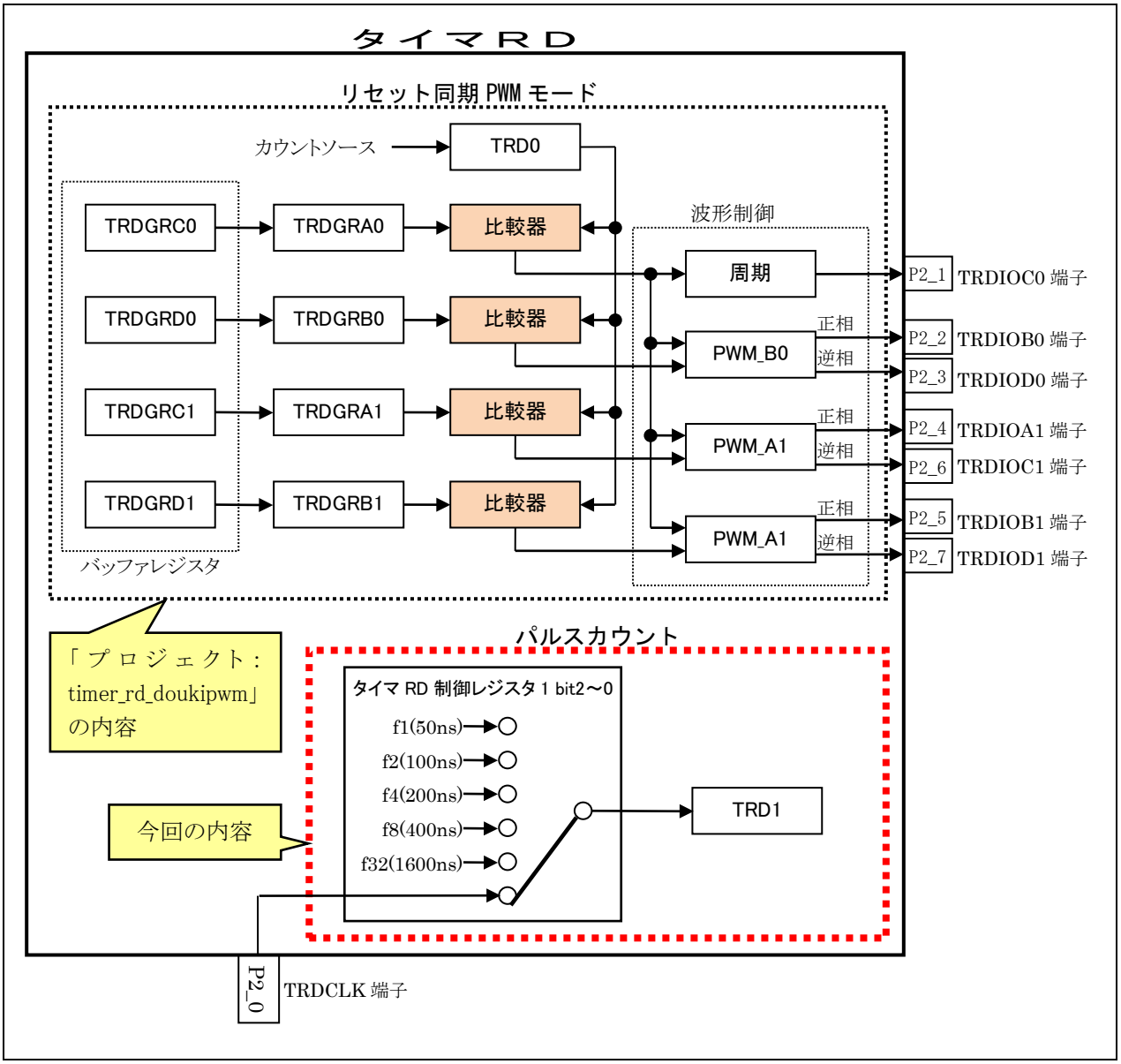
33.5.2 init 関数(タイマ RD の設定)

タイマ RD を設定するプログラムを、下記に示します。

```
71 :      /* タイマRD 外部入力（パルスカウント） */
72 :      trdpsr0 |= 0x01;                  /* TRDCLK端子:P2_0に設定          */
73 :      trdfcr  |= 0x40;                  /* 外部クロック入力有効に設定     */
74 :      trdcr1   = 0x15;                  /* TRDCLK端子 両エッジカウント    */
75 :
76 :      /* TRD1カウント開始 */
77 :      trdstr  |= 0x0e;                  /* TRD1カウント開始               */
```

(1) タイマ RD のブロック図

リセット同期 PWM モード部分は、「プロジェクト:timer_rd_doukipwm」の内容です。今回はそれに加え「パルスカウント」の部分を使用して、パルスカウントを行います。

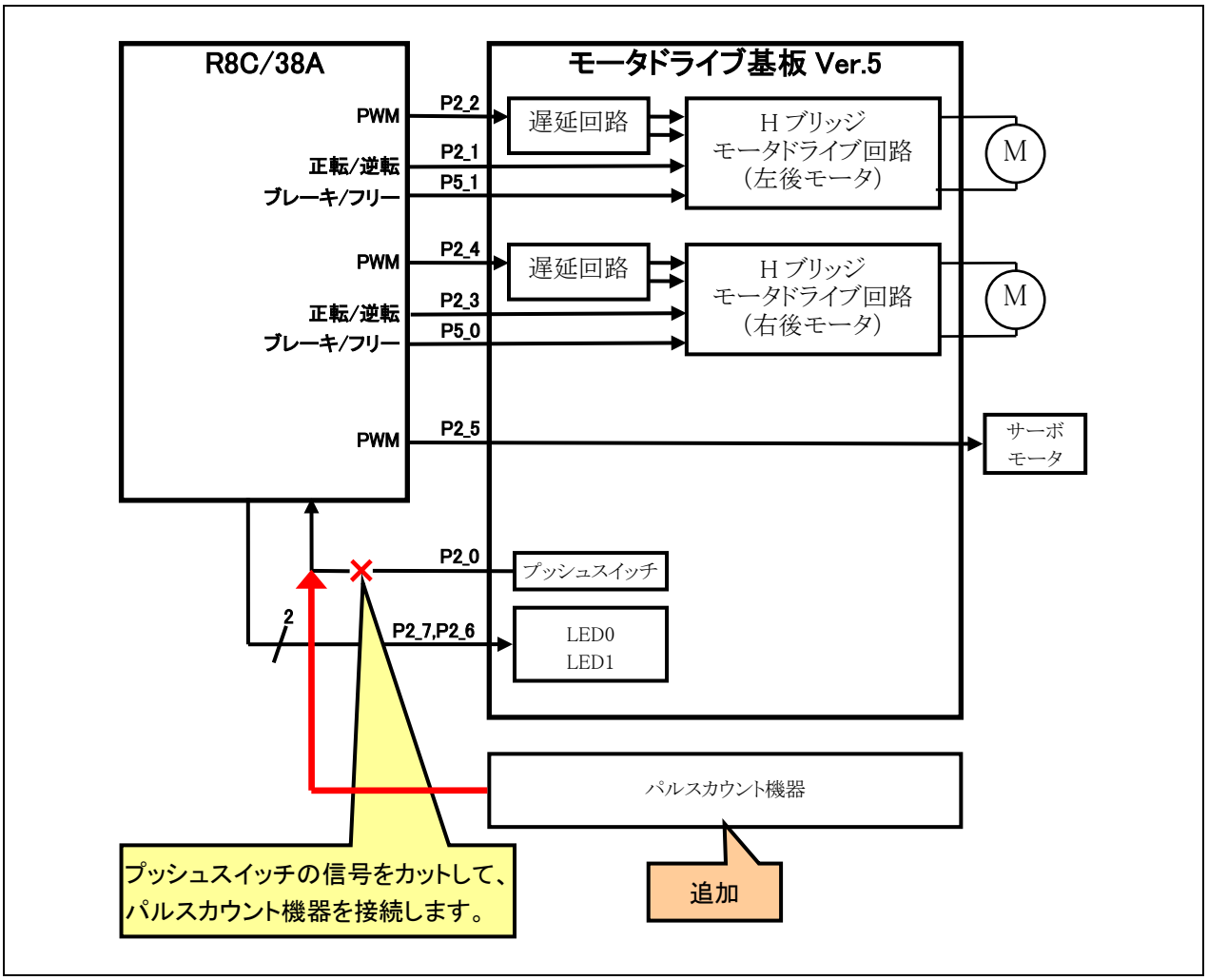


※タイマ RD の端子構成

端子名	割り当てる端子	入出力	機能
TRDIOA0/TRDCLK	P2_0	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。
TRDIOB0	P2_2	入出力	
TRDIOC0	P2_1	入出力	
TRDIOD0	P2_3	入出力	
TRDIOA1	P2_4	入出力	
TRDIOB1	P2_5	入出力	
TRDIOC1	P2_6	入出力	
TRDIOD1	P2_7	入出力	

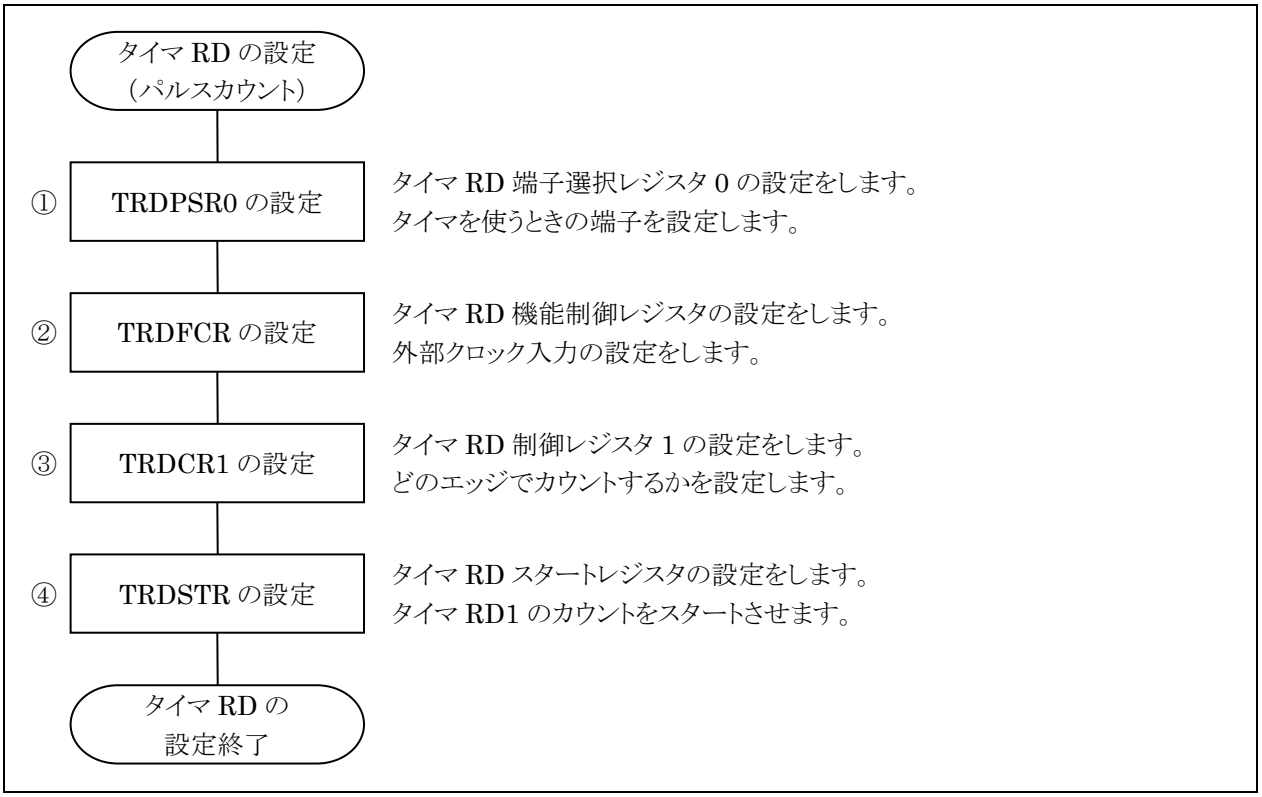
(2) モータドライブ基板との接続

今回のプログラムは、P2_0 端子を外部入力端子として使用します。マイコンカーでは、P2_0 端子をすでに使用しており、モータドライブ基板 Ver.5 の場合はプッシュスイッチの信号として使用しています。TRD1 を使ったパルスカウントとして使用したい場合は、これらの機能を別の端子へ移動してください。



(3) タイマ RD の設定（パルスカウント）

今回は、既にタイマ RD はリセット同期 PWM モードを使用している前提として、外部からのパルスを数える機能を追加します。レジスタの設定手順を下記に示します。



①タイマ RD 端子選択レジスタ 0（TRDPSR0：Timer RD Pin Select Register 0)の設定

タイマを使うときの端子を設定します。

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 trdiod0sel0	0：TRDIOD0 端子は使用しない 1：P2_3 に割り当てる 使いません。″0″を設定しておきます。	0
bit5～4	TRDIOC0 端子選択ビット bit5:trdioc0sel0 bit4:trdioc0sel1	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 使いません。″0″を設定しておきます。	00
bit3～2	TRDIOB0 端子選択ビット bit3:trdiob0sel0 bit2:trdiob0sel1	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 使いません。″0″を設定しておきます。	00
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 P2_0 端子からパルスを入力するので″1″を設定します。	1

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	0	0	0	**1**
16 進数	**0**				**1**			

②タイマ RD 機能制御レジスタ(TRDFCR：Timer RD Function Control Register)の設定

外部クロック入力を有効にします。

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 （注 3） pwm3_trdfcr	リセット同期 PWM モードでは無効です ″0″を設定	0
bit6	外部クロック入力選択ビット stclk_trdfcr	0：外部クロック入力無効 1：外部クロック入力有効 外部クロック入力を有効に設定します。″1″を設定します。	1
bit5	A/D トリガエッジ選択ビット （相補 PWM モード時） adeg_trdfcr	リセット同期 PWM モードでは無効です ″0″を設定	0
bit4	A/D トリガ許可ビット （相補 PWM モード時） adtrg_trdfcr		0
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_trdfcr	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″ 使いません。″0″を設定しておきます。	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_trdfcr		0
bit1～0	コンビネーションモード選択ビット（注 1、2） bit1:cmd1_trdfcr bit0:cmd0_trdfcr	リセット同期PWMモードでは″01b″（リセット同期PWMモード）にしてください 今回は、パルスカウントの設定ですので、″00″を設定しておきます。	00

注1. CMD1～CMD0ビットを″01b″、″10b″、″11b″に設定したとき、TRDPMRレジスタの設定にかかわらず、リセット同期PWMモードまたは相補PWMモードになります。
注2. CMD1～CMD0ビットはTRDSTRレジスタのTSTART0、TSTART1ビットがともに″0″(カウント停止)のときに書いてください。
注3. CMD1～CMD0ビットが″00b″(タイマモード・PWMモード・PWM3モード)のとき、PWM3ビットの設定が有効になります。

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	1	0	0	0	0	0	0
16 進数	4				0			

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

どのエッジでカウント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
bit7～5	TRD1 カウンタクリア選択ビット bit7:cclr2_trdcr1 bit6:cclr1_trdcr1 bit5:cclr0_trdcr1	"000b"(クリア禁止(フリーランニング動作))にしてください クリア禁止に設定します。"000"を設定します。	000
bit4～3	外部クロックエッジ選択ビット （注 3、4） 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 今回は、両エッジカウントを設定します。"10"を設定します。	10
bit2～0	カウントソース選択ビット （注 3） bit2:tck2_trdcr1 bit1:tck1_trdcr1 bit0:tck0_trdcr1	000：f1 001：f2 010：f4 011：f8 100：f32 101：TRDCLK 入力（注 1）または fC2（注 2） 110：fOCO40M 111：fOCO-F（注 5） TRDCLK 端子からパルスを入力します。"101"を設定します。	101

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. TRDCR0レジスタとTRDCR1レジスタの、TCK0～TCK2ビット、CKEG0～CKEG1ビットの設定は、同じにしてください。
注4. TCK2～TCK0ビットが"101b"(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 5. 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	0	**1**	**0**	**1**	**0**	**1**
16 進数	**1**				**5**			

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

タイマ RD1 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		"0"を設定	0
bit6		"0"を設定	0
bit5		"0"を設定	0
bit4		"0"を設定	0
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 TRDGRA1 レジスタとのコンペア一致後もカウント継続にします。"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 使いません。初期値の"1"を設定しておきます。	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 TRD1 のカウントを開始します。"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 使いません。"0"を設定しておきます。	0

注1. CSEL0ビットが"1"に設定されているとき、TSTART0ビットへ"0"を書いてください。
注2. CSEL1ビットが"1"に設定されているとき、TSTART1ビットへ"0"を書いてください。
注3. CSEL0ビットが"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. CSEL1ビットが"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	0
16 進数	0				e			

(4) タイマ RD のレジスタ設定

ここまでの設定で、タイマ RD によるパルスカウントの設定ができました。マイコンカーでは、駆動モータ 2 個、ステアリングモータ(サーボモータ)1 個をタイマ RD のリセット同期 PWM モードを使って制御しています。今回は、リセット同期 PWM モードと併用して、パルスカウントの設定をします。リセット同期 PWM モードとパルスカウントで設定しているレジスタに重複があります。ただし、設定するビットに重複はありません。

タイマ RD リセット同期 PWM モードとパルスカウトのレジスタ設定を下記に示します(**太字が重複するレジスタです**)。

```
    :     /* タイマRD リセット同期PWMモードの設定*/
    :     trdpsr0 = 0x68;              /* TRDIOB0,C0,D0端子設定        */
    :     trdpsr1 = 0x55;              /* TRDIOA1,B1,C1,D1端子設定     */
    :     trdmr   = 0xf0;              /* バッファレジスタ設定         */
    :     trdfcr  = 0x01;              /* リセット同期PWMモードに設定  */
    :     trdcr0  = 0x23;              /* ソースカウントの選択:f8      */
    :     trdgra0 = trdgrc0 = 39999;   /* 周期                         */
    :     trdgrb0 = trdgrd0 = 0;       /* P2_2端子のON幅設定           */
    :     trdgra1 = trdgrc1 = 0;       /* P2_4端子のON幅設定           */
    :     trdgrb1 = trdgrd1 = 0;       /* P2_5端子のON幅設定           */
    :     trdoer1 = 0x01;              /* 出力端子の選択               */
    :     trdstr  = 0x0d;              /* TRD0カウント開始             */
-------------------------------------------------------------------------
 71 :     /* タイマRD 外部入力（パルスカウント） */
 72 :     trdpsr0  = 0x01;             /* TRDCLK端子:P2_0に設定        */
 73 :     trdfcr   = 0x40;             /* 外部クロック入力有効に設定   */
 74 :     trdcr1   = 0x15;             /* TRDCLK端子 両エッジカウント  */
 75 :
 76 :     /* TRD1カウント開始 */
 77 :     trdstr   = 0x0e;             /* TRD1カウント開始             */
```

レジスタ設定に重複があるとどのようになるのでしょうか。始めに、リセット同期 PWM モードのレジスタを設定し、その後にパルスカウントのレジスタを設定すると、パルスカウントで設定したレジスタの値で上書きされ、リセット同期 PWM モードの設定値が消えてしまいます。そこで、リセット同期 PWM モードで設定した値が変わらないようにします。

重複するレジスタは、論理和(OR)することにより、特定のビットだけ"1"にすることができます。

例えば、TRDPSR0 のレジスタを見てみると、リセット同期 PWM モードは、

```
    trdpsr0 = 0x68;              /* TRDIOB0,C0,D0端子設定        */
```

です。パルスカウントの設定は、

```
    trdpsr0  = 0x01;             /* TRDCLK端子:P2_0に設定        */
```

です。

これらの値を OR すると、

```
      0110 1000    リセット同期 PWM モードの設定値
OR)   0000 0001    パルスカウントの設定値
-------------------
      0110 1001  = TRDPSR0 の設定値
```

となります。リセット同期 PWM モードの設定値を上書きすることなく、パルスカウントの設定ができています。今回は、論理和(OR)することにより、設定値を上書きしないようにします。

論理和(OR)を使ったパルスカウントの設定は、下記のようになります。

```
71 :      /* タイマRD 外部入力（パルスカウント） */
72 :      trdpsr0 |= 0x01;                  /* TRDCLK端子:P2_0に設定          */
73 :      trdfcr  |= 0x40;                  /* 外部クロック入力有効に設定     */
74 :      trdcr1   = 0x15;                  /* TRDCLK端子 両エッジカウント    */
75 :
76 :      /* TRD1カウント開始 */
77 :      trdstr  |= 0x0e;                  /* TRD1カウント開始               */
```

リセット同期 PWM モードを使用しているプログラムにパルスカウントを追加する場合は、リセット同期 PWM モードを設定した後、上記のプログラムを追加するようにしてください。

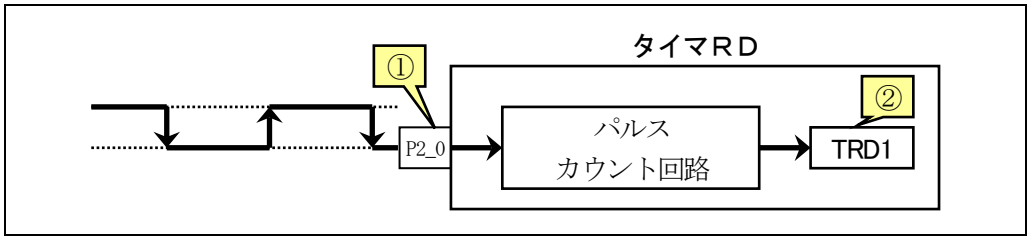
33.5.3 main 関数

タイマ RD でパルスカウントを行い、カウントした値をポート 0 へ出力します。

```
29 : /************************************************************************/
30 : /* メインプログラム                                                     */
31 : /************************************************************************/
32 : void main( void )
33 : {
34 :     init();                           /* 初期化                         */
35 :
36 :     while( 1 ) {
37 :         p0 = trd1;
38 :     }
39 : }
```

37 行	タイマ RD1 カウンタ(TRD1)は、P2_0 端子からパルスが入力されるたびに値が増えていきます。 信号は、タイマ RD 制御レジスタ 1(TRGCR1)で、「両エッジでカウント」を設定しているため、1 パルスで 2 カウントします（信号が"0"→"1"になった瞬間と、"0"から"1"になった瞬間）。ロータリエンコーダで 1 回転のカウント数が少ない場合は有効な機能です。

タイマ RD1 カウンタ(TRD1)の値は、P2_0 端子に入力されるパルスにより値が増えていきます。



①	タイマ RD 端子選択レジスタ(TRDPSR0)の TRDCLK 端子選択ビットで設定した端子です。今回は、P2_0 端子に割り当てたので、P2_0 端子に入力されたパルスになります。 タイマ RD 制御レジスタ 1（TRDCR1）の外部クロックエッジ選択ビットで、立ち上がりエッジでカウントするか、立ち下がりエッジでカウントするか、両エッジでカウントするかを選びます。今回は両エッジでカウントにします。
②	タイマ RD1 カウンタ(TRD1)の値はパルスが入力されると、0→1→2･･･と値が増えていきます。タイマ RD は 16bit タイマですので、0～65535 までカウントできます。TRD1 がオーバフローすると、0 に戻ります(65535 の次の値は 0 になります)。

34. タイマ RE による割り込み（プロジェクト：timer_re_timerwarikomi）

34.1 概要

本章は、タイマ RE で約 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。動作はプロジェクト「timer2」と同じですが、「timer2」はタイマ RB を使った割り込みを使います。

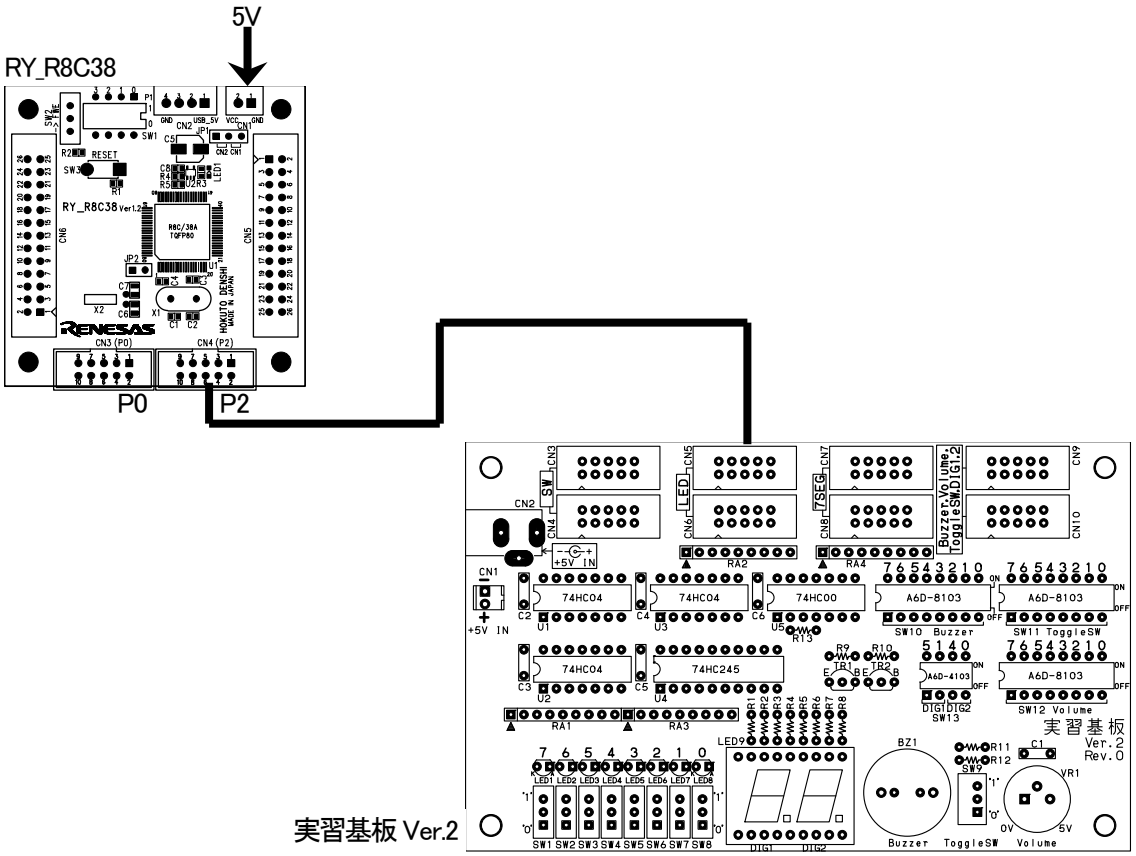
例えば、タイマ RB で 1ms ごとに割り込みプログラムを実行したとき、割り込みプログラムが長くなり 1ms で終わらないとき、10ms ごとに実行しても問題なく優先度の低いプログラム（スイッチ処理や液晶処理など）があれば、今回のタイマ RE 割り込み処理に移動させ、優先度の高いプログラムはタイマ RB 割り込みで 1ms ごとに実行、優先度の低いプログラムはタイマ RE 割り込みで 10ms ごとに実行など、処理を分散させることができます。

34.2 接続

マイコンのポート	接続内容
P2 (CN4)	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。電源を入れると実習基板 Ver.2 の 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。

34.3 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。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥timer_re_timerwarikomi¥startup.c
2	timer_re_ timerwarikomi.c	実際に制御するプログラムが書かれています。R8C/38A の内蔵周辺機能(SFR)の初期化も行います。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥timer_re_timerwarikomi¥ timer_re_timerwarikomi.c
3	sfr_r838a.h	R8C/38A マイコンの内蔵周辺機能を制御するためのレジスタ（Special Function Registers)を定義したファイルです。 ファイルの位置→C:¥Workspace¥common_r8c38a¥sfr_r838a.h

34.4 プログラム「timer_re_timerwarikomi.c」

```
/************************************************************************/
/* 対象マイコン R8C/38A                                                 */
/* ﾌｧｲﾙ内容     タイマRE割り込みによるタイマ                            */
/* バージョン   Ver.1.00                                                */
/* Date         2014.05.12                                              */
/* Copyright    ジャパンマイコンカーラリー実行委員会                    */
/************************************************************************/
/*
出力：P2_7-P2_0(実習基板Ver.2のLED部)

ポート2に繋いだLEDを1秒間隔で点滅させます。
タイマはタイマRE割り込みによる正確なタイマを使用します。
*/

/*====================================*/
/* インクルード                       */
/*====================================*/
#include "sfr_r838a.h"                  /* R8C/38A SFRの定義ファイル     */

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

/*====================================*/
/* プロトタイプ宣言                   */
/*====================================*/
void init( void );
void timer_re( unsigned long timer_set );

/*====================================*/
/* グローバル変数の宣言               */
/*====================================*/
unsigned long cnt_re;                   /* タイマRE用                   */

```

```
/************************************************************************/
/* メインプログラム                                                     */
/************************************************************************/
void main( void )
{
    init();                             /* 初期化                       */
    asm(" fset I ");                    /* 全体の割り込み許可           */

    while( 1 ) {
        p2 = 0x55;
        timer_re( 1000 );
        p2 = 0xaa;
        timer_re( 1000 );
        p2 = 0x00;
        timer_re( 1000 );
    }
}

/************************************************************************/
/* R8C/38A スペシャルファンクションレジスタ(SFR)の初期化                */
/************************************************************************/
void init( void )
{
    int i;

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

    /* ポートの入出力設定 */
    prc2 = 1;                           /* PD0のプロテクト解除          */
    pd0 = 0x00;                         /*                              */
    pd1 = 0xd0;                         /* 5:RXD0 4:TXD0 3-0:DIP SW     */
    pd2 = 0xff;                         /* 7-0:LED                      */
    pd3 = 0xff;                         /*                              */
    p4  = 0x20;                         /* P4_5のLED:初期は点灯         */
    pd4 = 0xb8;                         /* 7:XOUT 6:XIN 5:LED 2:VREF    */
    pd5 = 0xff;                         /*                              */
    pd6 = 0xff;                         /*                              */
    pd7 = 0xff;                         /*                              */
    pd8 = 0xff;                         /*                              */
    pd9 = 0x3f;                         /*                              */
    pur0 = 0x04;                        /* P1_3～P1_0のプルアップON     */

    /* タイマREの設定 */
    /* 割り込み周期 = 1 / 20[MHz]   * 32 * (TREMIN+1)
                    = 1 / (20*10^6) * 32 * 157
                    = 1.0048[ms] ※ほぼ1ms
    */
    trecsr        = 0x04;               /* ｶｳﾝﾄｿｰｽ=f4,4ﾋﾞｯﾄｶｳﾝﾀ使用    */
    tremin        = 157 - 1;            /* 割り込み周期の設定           */
    comie_trecr2  = 1;                  /* コンペア一致割り込み許可     */
    treic         = 0x01;               /* 割り込み優先レベル設定       */
    tstart_trecr1 = 1;                  /* タイマREカウント開始         */
}

/************************************************************************/
/* タイマ本体                                                           */
/* 引数   タイマ値 1=1ms                                                */
/************************************************************************/
void timer_re( unsigned long timer_set )
{
    cnt_re = 0;
    while( cnt_re < timer_set );
}

/************************************************************************/
/* タイマRE 割り込み処理(優先順の低い割り込み処理)                      */
/************************************************************************/
#pragma interrupt intTRE(vect=10)
void intTRE( void )
{
    asm(" fset I ");                    /* 多重割り込み許可   */

    cnt_re++;
}

/************************************************************************/
/* end of file                                                          */
/************************************************************************/
```

34.5 プログラムの解説

34.5.1 init 関数(タイマ RE の設定)

タイマ RE を使って、1.0048ms ごと(以下、本章では 1ms ごととする)に割り込みを発生させます。

```
 83 :      /* タイマREの設定 */
 84 :      /* 割り込み周期 = 1 / 20[MHz]   * 32 * (TREMIN+1)
 85 :                     = 1 / (20*10^6) * 32 * 157
 86 :                     = 1.0048[ms] ※ほぼ1ms
 87 :      */
 88 :      trecsr        = 0x04;              /* ｶｳﾝﾄｿｰｽ=f4,4ﾋﾞｯﾄｶｳﾝﾀ使用      */
 89 :      tremin        = 157 - 1;           /* 割り込み周期の設定          */
 90 :      comie_trecr2  = 1;                 /* コンペア一致割り込み許可    */
 91 :      treic         = 0x01;              /* 割り込み優先レベル設定      */
 92 :      tstart_trecr1 = 1;                 /* タイマREカウント開始        */
```

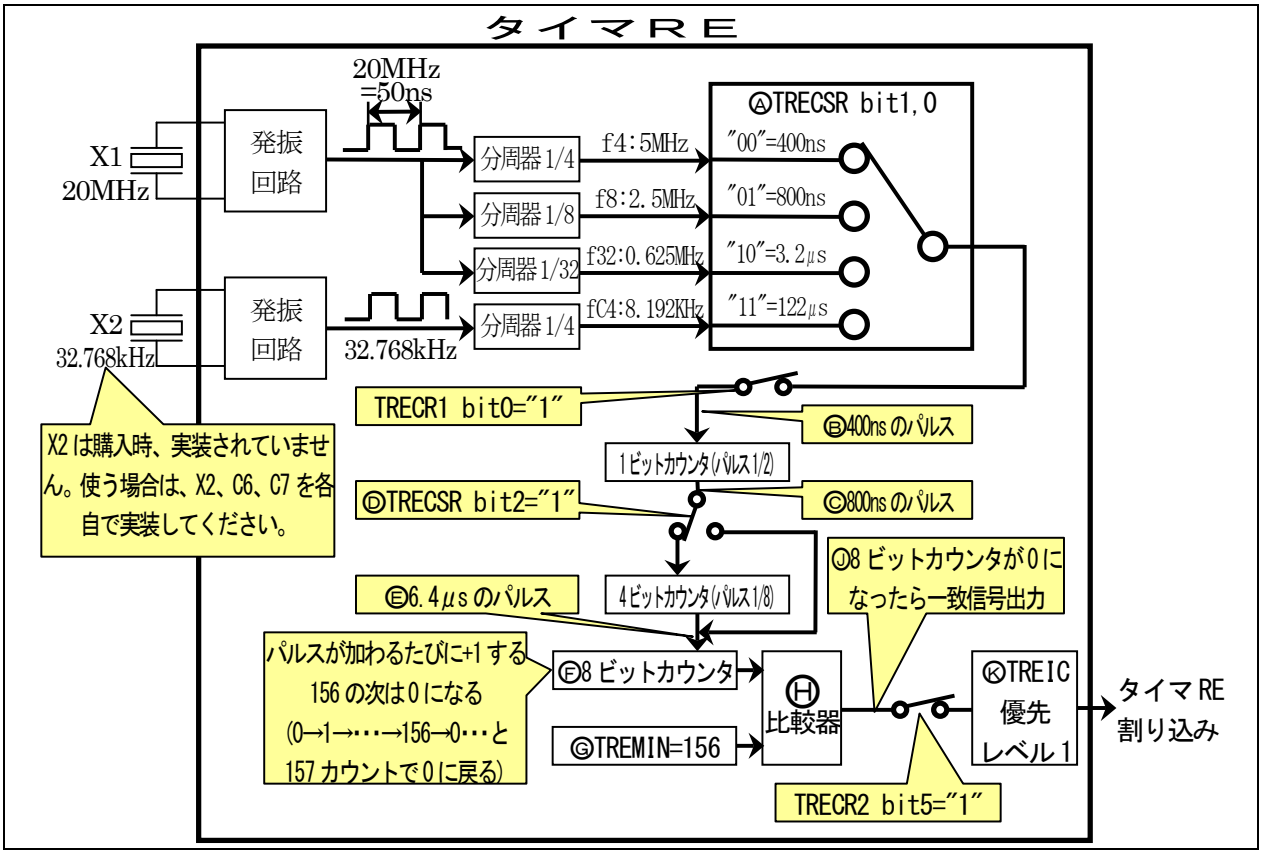
(1) タイマ RE の設定(アウトプットコンペアモード)

今回はタイマ RE をアウトプットコンペアモードで使用して、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。

※TREMIN は、リアルタイムクロックモードで使用するときは「タイマ RE 分データレジスタ」、今回のアウトプットコンペアモードで使用するときは「タイマ RE コンペアデータレジスタ」となります。

(2) ブロック図と動作原理

今回はタイマ RE をアウトプットコンペアモードで使用して、1ms ごとに割り込みを発生させるように設定にします。レジスタを設定した後の動作原理を下記に示します。



Ⓐ Ⓑ	タイマ RE カウントソース選択レジスタ(TRECSR)の RCS1～0(bit1,0)の設定をして、どのクロックを使うか選択します。今回は、"00"の f4 を選択します。400ns のパルスが出力されます。
Ⓒ	パルスは 1/2 され、800ns のパルスになります。
Ⓓ Ⓔ	タイマ RE カウントソース選択レジスタ(TRECSR)の 4 ビットカウンタ選択ビット(bit2)を"1"にして、4 ビットカウンタを通します。パルスは $1/2^4=1/8$ になり 6.4μs になります。
Ⓕ Ⓖ Ⓗ	8 ビットカウンタ(Ⓕ)が 6.4μs ごとに＋1します。8 ビットカウンタと TREMIN(Ⓖ)は、比較器(Ⓗ)により常に比較されています。「8 ビットカウンタ＝TREMIN」になった次のクロックで 8 ビットカウンタは 0 になります。 今回、TREMIN には 156 がセットされています。8 ビットカウンタは「0→1→・・・→156→0→1・・・」と 157 回ごとに 0 になります。これが割り込み周期となります。今回の割り込み周期を下記に示します。 6.4μs×157＝10048μs＝1.0048ms **※8 ビットカウンタは、TREMIN に設定した値の次で 0 に戻ります。よって TREMIN の設定値は、「計算値－1」の値を設定します。例えば 157 を設定したいなら、156 を設定します。**
Ⓘ	8 ビットカウンタが 0 になった瞬間に、比較器から一致信号が出力されます。
Ⓚ	タイマ RE の割込優先レベルが 1 以上なら、割り込みが発生します。今回は 1 を設定しているので割り込みが発生します。タイマ RE 割り込みプログラムが実行されると、自動的に一致信号が"0"になります。

このように、タイマ RE によって 1ms ごとに割り込みが発生します。

①タイマ RE カウントソース選択レジスタ(TRECSR:TimerRE count source select register)の設定

タイマ RE のカウントソースを設定します。

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
bit6,5,4	クロック出力選択ビット(注 2) bit6:rcs6_trecsr bit5:rcs5_trecsr bit4:rcs4_trecsr	000:f2 (2/20MHz＝100ns) 001:fC (32.768kHz＝30.52μs) ※X2 が実装されている場合のみ使用可能 010:f4 (4/20MHz＝200ns) 100:f8 (8/20MHz＝400ns) 110:コンペアマッチ出力 今回は使用しません。	000
bit3		"0"を設定	0
bit2	4 ビットカウンタ選択ビット(注 1) rcs2_trecsr	0:使用しない 1:使用する 使用するので"1"を設定します。	1
bit1,0	カウントソース選択ビット(注 1) bit1:rcs1_trecsr bit0:rcs0_trecsr	00:f4 (4/20MHz＝200ns) 01:f8 (8/20MHz＝400ns) 10:f32 (32/20MHz＝1600ns) 11:fC4 (4/32.768kHz＝122.04μs) ※fC4 は X2 が実装されている場合のみ使用可能 f4 を選択します。選択方法は次ページで説明します。	00

注 1. RCS0～RCS2 ビットは、TRECR1 レジスタの TCSTF ビットが“0”(カウント停止中)のとき、書いてください。
注 2. RCS4～RCS6 ビットは、TRECR1 レジスタの TOENA ビットが“0”(クロック出力禁止)のとき、書いてください。

※タイマ RE カウントソース選択レジスタ(TRECSR)のカウントソース選択ビットの設定方法

タイマ RE カウントソース選択レジスタ(TRECSR)のカウントソース選択ビット(bit1,0)で、8 ビットカウンタがどのくらいの間隔で+1 するか設定します。
タイマ RE カウントソース選択レジスタ(TRECSR)の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。

TRECSR bit1,0	内容
00	8 ビットカウンタがカウントする間隔は、6.4μs(f4(200ns)×2×16)になります。※2 倍は、ブロック図にあるとおり、必ず 2 倍されます。16 倍は、4 ビットカウンタ選択ビット(bit4)を"1"にしているためです。 設定できる割り込み間隔の最大は、 6.4μs×256＝**1.6384ms** となります。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。
01	8 ビットカウンタがカウントする間隔は、12.8μs(f8(400ns)×2×16)になります。 設定できる割り込み間隔の最大は、 12.8μs×256＝**3.2768ms** となります。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、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。
10	8 ビットカウンタがカウントする間隔は、51.2μs(f32(1600ns)×2×16)になります。 設定できる割り込み間隔の最大は、 51.2μs×256＝**13.1072ms** となります。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。ただし、X2(32.768kHz)のクロックを実装している場合は、下記の"11"の設定も可能です。
11	8 ビットカウンタがカウントする間隔は、39.0625ms(fC4(122.04μ)×2×16)になります。 設定できる割り込み間隔の最大は、 39.0625 ms×256＝**1.000s** となります。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"を設定します。 ※この設定ができるのは、X2(32.768kHz)のクロックを実装している場合のみです。

今回は、割り込み間隔を 1ms にします。タイマ RE カウントソース選択レジスタ(TRECSR)の bit1,0 の設定について、
"00"を設定…最大の割り込み間隔は 1.6384ms、今回設定したい 1ms の割り込み間隔を設定できる
"01"を設定…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
"10"を設定…最大の割り込み間隔は 13.1072ms、今回設定したい 1ms の割り込み間隔を設定できる

2 つ以上設定できる場合は、割り込み間隔が小さい設定を選びます。今回は、"00"を設定します。

タイマ RE カウントソース選択レジスタ(TRECSR)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**	**0**
16 進数	0				4			

②タイマ RE コンペアデータレジスタ(TREMIN：Compare Data Register)の設定

※TREMIN は、リアルタイムクロックモードで使用するときは「タイマ RE 分データレジスタ」、今回のアウトプットコンペアモードで使用するときは「タイマ RE コンペアデータレジスタ」となります。

タイマ RE コンペアデータレジスタ（TREMIN）は、割り込み周期を設定します。

TREMIN の値を計算する式を、下記に示します。

タイマ RE 割り込み要求周期＝タイマ RE カウントソース×（TREMIN＋1）

TREMIN を左辺に移動します。

TREMIN＋1＝タイマ RE 割り込み要求周期／タイマ RE 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 RE カウントソースは、①のタイマ RE カウントソース選択レジスタ(TRECSR)のカウントソース選択ビット（bit1,0）と 4 ビットカウンタ選択ビット(bit2)で設定している内容で、今回は 6.4μs です。TREMIN の計算を下記に示します。

TREMIN＋1＝タイマ RE 割り込み要求周期／タイマ RE カウントソース
TREMIN＋1＝(1×10^{-3})　／(6.4×10^{-6})
TREMIN＋1＝(1×10^{-3})　／(6.4×10^{-6})
TREMIN＋1＝156.25
∴**TREMIN**＝156.25－1＝**155.25　、切り捨てた場合は 155、切り上げた場合は 156**

設定できる値は整数のみで小数点は設定できないので、切り捨てか、切り上げます。今回、それぞれの時間は、
切り捨てた場合＝タイマ RE カウントソース×（TREMIN＋1）＝6.4μ×（155+1）＝0.9984ms（誤差 0.20%）
切り上げた場合＝タイマ RE カウントソース×（TREMIN＋1）＝6.4μ×（156+1）＝1.0048ms（誤差 0.48%）

通常誤差が少ない方にしますが、今回のタイマ RE の用途は、タイマ RB 割り込みより優先度の低い割り込みを使いたい（＝回数は少なくても良い）ということなので、割り込み周期が長い方の 156 の値にしておきます。用途に応じて切り上げ、切り捨てを判断してください。

タイマ RE コンペアデータレジスタ（TREMIN）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 5 6**							

③タイマ RE 制御レジスタ 2(TRECR2：Timer RE Control Register 2)の設定

タイマ RE 制御レジスタ 2 のコンペア一致割り込み許可ビット(bit5)の設定をします。"1"にして、許可に設定します。

設定 bit	上:ビット名 下:シンボル	内容	今回の内容
bit7、6		"0"を設定	00
bit5	コンペア一致割り込み許可ビット comie_trecr2	0:コンペア一致割り込み禁止 1:コンペア一致割り込み許可 8 ビットカウンタと TREMIN の値が一致したとき、割り込みを発生させるか設定します。今回は発生させるので"1"を設定します。	1
bit4～0		"0"を設定	00000

タイマ RE 制御レジスタ 2(TRECR2)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	0	0
16 進数	2				0			

プログラムでは、bit5 のみ"1"にするため、下記のようにしています。

```
comie_trecr2  = 1;                    /* コンペア一致割り込み許可      */
```

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

タイマ RE の割り込み優先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
bit7～4		"0"を設定	0000
bit3	割り込み要求ビット ir_treic	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_treic bit1: ilvl1_treic bit0: ilvl0_treic	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 割り込みより優先順位の低い割り込みという想定なので、レベル 1～7 でいちばん低い"001"を設定します。	001

タイマ RE 割り込み制御レジスタ(TREIC)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**			

⑤タイマ RE 制御レジスタ 1（TRECR1：Timer RE Control Register1）の設定

タイマ RE 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	タイマ RE カウント開始ビット tstart_trecr1	0：カウント停止 1：カウント開始 カウントを開始するので、″1″にします。	1
bit6、5		″0″を設定	00
bit4	タイマ RE リセットビット trerst_trecr1	このビットを“1”にした後、“0”にすると次の状態になります。 • TRESEC、TREMIN、TREHR、TREWK、TRECR2 レジスタが“00h” • TRECR1 レジスタの TCSTF、INT、PM、H12_H24、TSTART ビットが“0” • 8 ビットカウンタが“00h”、4 ビットカウンタが“0h” タイマ RE をリセット状態にするビットです。今回は特にしませんので″0″を設定します。	0
bit3		″0″を設定	0
bit2	TREO 端子出力許可ビット toena_trecr1	0：クロック出力禁止 1：クロック出力許可 クロックは出力しませんので″0″を設定します。	0
bit1	タイマ RE カウントステータスフラグ tcstf_trecr1	0：カウント停止中 1：カウント中 カウント中かどうかチェックするビットです。″0″を設定します。	0
bit0		″0″を設定	0

タイマ RE 制御レジスタ 1（TRECR1）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			

プログラムでは、bit7 のみ″1″にするため、下記のようにしています。

```
tstart_trecr1 = 1;                  /* タイマREカウント開始          */
```

34.5.2 intTRE 関数（1ms ごとに実行される関数）

先の設定で、タイマ RE を 1ms ごとに割り込みを発生させる設定にしました。intTRE 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。

```
108 :  #pragma interrupt intTRE(vect=10)
109 :  void intTRE( void )
110 :  {
111 :      asm(" fset I ");                    /* 多重割り込み許可    */
112 :
113 :      cnt_re++;
114 :  }
```

108 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 RE 割り込みは表より、10 番です。 よって、10 番の割り込みが発生したときに intTRE 関数を実行するよう、「#pragma interrupt」で設定します。
109 行	タイマ RE 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。
111 行	intTRE 関数を実行中に、その他の優先順位の高い割り込みを許可するようにします。 例えば、タイマ RE 割り込み制御レジスタ(TREIC)の優先レベルが 1、タイマ RB 割り込み制御レジスタ(TRBIC)の優先レベルが 7 のとき、intTRE 関数を実行中でタイマ RB 割り込みが発生すると、タイマ RB 割り込みプログラムを実行します。優先レベルが低い割り込みは実行されません。 **※この割り込み許可命令を実行しないと、優先レベルの高い割り込みも実行されません。優先レベルの高い割り込みを発生させたい場合は、必ずこの命令を実行してください。**
113 行	cnt_re 変数を+1 します。この関数は 1ms ごとに実行されるので、cnt_re は 1ms ごとに+1 されることになります。

※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は、下記のとおりです。

割り込み要因	ベクタ番号(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
BRK 命令(注 3)	+0～+3（0000h～0003h）	0	
フラッシュメモリレディ	+4～+7（0004h～0007h）		FMRDYIC
—(予約)		2～5	
INT4	+24～+27（0018h～001Bh）	6	INT4IC
タイマ RC	+28～+31（001Ch～001Fh）	7	TRCIC
タイマ RD0	+32～+35（0020h～0023h）	8	TRD0IC
タイマ RD1	+36～+39（0024h～0027h）	9	TRD1IC
タイマ RE	**+40～+43（0028h～002Bh）**	**10**	**TREIC**
UART2 送信/NACK2	+44～+47（002Ch～002Fh）	11	S2TIC
UART2 受信/ACK2	+48～+51（0030h～0033h）	12	S2RIC
キー入力	+52～+55（0034h～0037h）	13	KUPIC
A/D 変換	+56～+59（0038h～003Bh）	14	ADIC
シンクロナスシリアルコミュニケーションユニット/I2C バスインタフェース(注 2)	+60～+63（003Ch～003Fh）	15	SSUIC/ IICIC
タイマ RF コンペア 1	+64～+67（0040h～0043h）	16	CMP1IC
UART0 送信	+68～+71（0044h～0047h）	17	S0TIC
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
タイマ RA	+88～+91（0058h～005Bh）	22	TRAIC
— (予約)		23	
タイマ RB	+96～+99(0060h～0063h)	24	TRBIC
INT1	+100～+103（0064h～0067h）	25	INT1IC
INT3	+104～+107（0068h～006Bh）	26	INT3IC
タイマ RF	+108～+111（006Ch～006Fh）	27	TRFIC
タイマ RF コンペア 0	+112～+115（0070h～0073h）	28	CMP0IC
INT0	+116～+119（0074h～0077h）	29	INT0IC
UART2 バス衝突検出	+120～+123（0078h～007Bh）	30	U2BCNIC
タイマ RF キャプチャ	+124～+127（007Ch～007Fh）	31	CAPIC
ソフトウェア(注 3)	+128～+131（0080h～0083h）～ +164～+167（00A4h～00A7h）	32～41	
— (予約)		42	
タイマ RG	+172～+175（00ACh～00AFh）	43	TRGIC
— (予約)		44～49	
電圧監視 1	+200～+203（00C8h～00CBh）	50	VCMP1IC
電圧監視 2	+204～+207（00CCh～00CFh）	51	VCMP2IC
— (予約)		52～55	
ソフトウェア(注 3)	+224～+227（00E0h～00E3h）～ +252～+255（00FCh～00FFh）	56～63	

注 1. INTB レジスタが示す番地からの相対番地です。
注 2. SSUIICSR 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。

34.5.3 timer_re 関数（割り込みを使った時間稼ぎ）

timer_re 関数は、実行した行で時間稼ぎをする関数です。タイマ RE による時間稼ぎなので「_re」を追加しています。timer_re 関数の使い方は、「**14. 割り込みによるタイマ（プロジェクト：timer2）**」と同様です。こちらを参照してください。

```
 95 :  /************************************************************************/
 96 :  /* タイマ本体                                                           */
 97 :  /* 引数    タイマ値 1=1ms                                               */
 98 :  /************************************************************************/
 99 :  void timer_re( unsigned long timer_set )
100 :  {
101 :      cnt_re = 0;
102 :      while( cnt_re < timer_set );
103 :  }
```

34.5.4 main 関数

動作原理は、「**14. 割り込みによるタイマ（プロジェクト：timer2）**」の main 関数と同様です。こちらを参照してください。

```
 38 :  void main( void )
 39 :  {
 40 :      init();                          /* 初期化                       */
 41 :      asm(" fset I ");                 /* 全体の割り込み許可           */
 42 :
 43 :      while( 1 ) {
 44 :          p2 = 0x55;
 45 :          timer_re( 1000 );
 46 :          p2 = 0xaa;
 47 :          timer_re( 1000 );
 48 :          p2 = 0x00;
 49 :          timer_re( 1000 );
 50 :      }
 51 :  }
```

35. 高速オンチップオシレータの使用(プロジェクト:hoco_io)

35.1 概要

本章では、マイコンの動作クロックとして高速オンチップオシレータを使用したプログラムについて説明します。プログラムの動作は「11. I/O ポートの入出力(プロジェクト:io)」を参照してください。

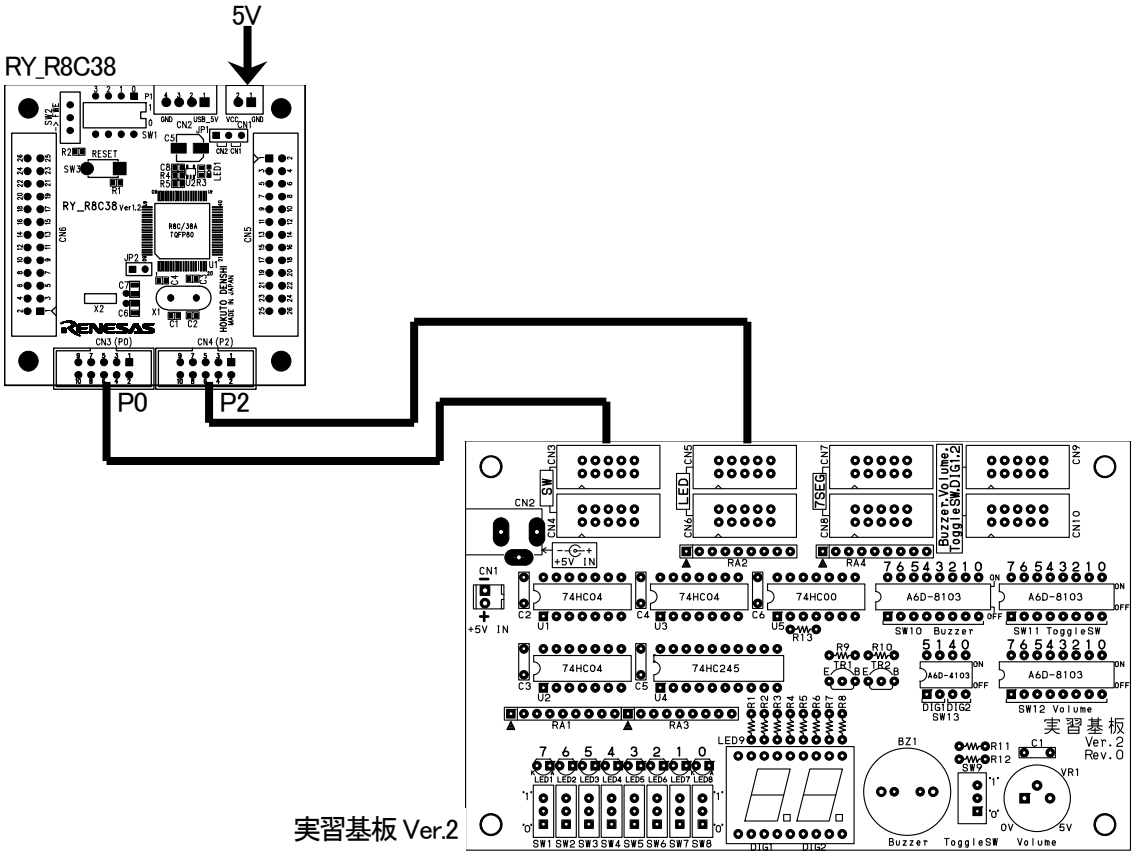
35.2 接続

■使用ポート

マイコンのポート	接続内容
P0 (CN3)	実習基板 Ver.2 のスライドスイッチ部など、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.5 も可能です。
P2 (CN4)	実習基板 Ver.2 の LED 部など、出力機器を接続します。

■接続例

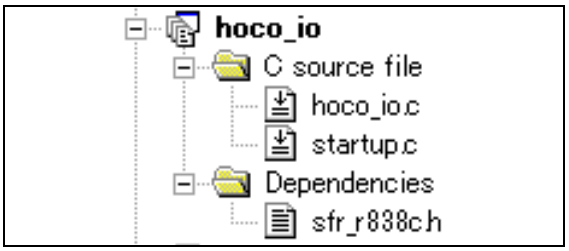
実習基板 Ver.2 を使ったときの接続例を下記に示します。



■操作方法

実習基板 Ver.2 のスライドスイッチ(SW1～8)を ON/OFF すると、それに合わせて LED(LED1～8)が点灯/消灯します。入力したデータを出力する、制御の基本中の基本です。

35.3 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。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥hoco_io¥startup.c
2	hoco_io.c	実際に制御するプログラムが書かれています。R8C/38A の内蔵周辺機能(SFR)の初期化も行います。 ファイルの位置→C:¥Workspace¥r8c38a_ensyu¥hoco_io¥hoco_io.c
3	sfr_r838a.h	R8C/38A マイコンの内蔵周辺機能を制御するためのレジスタ（Special Function Registers)を定義したファイルです。 ファイルの位置→C:¥Workspace¥common_r8c38a¥sfr_r838a.h

35.4 プログラム「hoco_io.c」

```
 1 :  /***********************************************************************/
 2 :  /* 対象マイコン R8C/38C(今回のﾌﾟﾛｸﾞﾗﾑはR8C/38Aでは動作を保証していません)    */
 3 :  /* ﾌｧｲﾙ内容     データの入力、出力(動作ｸﾛｯｸに高速ｵﾝﾁｯﾌﾟｵｼﾚｰﾀ(HOCO)使用       */
 4 :  /* バージョン   Ver.1.01                                               */
 5 :  /* Date         2014.05.12                                             */
 6 :  /* Copyright    ジャパンマイコンカーラリー実行委員会                   */
 7 :  /***********************************************************************/
 8 :  /*
 9 :  入力：P0_7-P0_0(実習基板Ver.2のスライドスイッチ部)
10 :  出力：P2_7-P2_0(実習基板Ver.2のLED部)
11 :
12 :  ポート0に入力した状態を、ポート2に出力します。
13 :  ポート0にセンサ基板、ポート2に実習基板Ver.2のLED部分を接続すれば
14 :  センサ基板のチェックになります。
15 :  */
16 :
17 :  /*====================================*/
18 :  /* インクルード                       */
19 :  /*====================================*/
20 :  #include "sfr_r838c.h"                 /* R8C/38C 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 :
```

```
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 :          d = p0;
42 :          p2 = d;
43 :      }
44 :  }
45 :
46 :  /************************************************************************/
47 :  /* R8C/38C スペシャルファンクションレジスタ(SFR)の初期化                */
48 :  /************************************************************************/
49 :  void init( void )
50 :  {
51 :      int i;
52 :
53 :      /* ｸﾛｯｸを高速ｵﾝﾁｯﾌﾟｵｼﾚｰﾀ(40MHz)に変更(ﾌﾟﾛｸﾞﾗﾑは20MHzで動作) */
54 :      prc0  = 1;                          /* プロテクト解除               */
55 :      fra00 = 1;                          /* 高速オンチップオシレータ発振 */
56 :      for(i=0; i<50; i++ );               /* 安定するまで少し待つ(約10ms) */
57 :      fra01 = 1;                          /* 高速オンチップオシレータ選択 */
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 = 0x00;                         /* 7-0:スイッチ                 */
63 :      pd1 = 0xd0;                         /* 5:RXD0 4:TXD0 3-0:DIP SW     */
64 :      pd2 = 0xff;                         /* 7-0:LED                      */
65 :      pd3 = 0xff;                         /*                              */
66 :      p4  = 0x20;                         /* P4_5のLED:初期は点灯         */
67 :      pd4 = 0xb8;                         /* 7:XOUT 6:XIN 5:LED 2:VREF    */
68 :      pd5 = 0xff;                         /*                              */
69 :      pd6 = 0xff;                         /*                              */
70 :      pd7 = 0xff;                         /*                              */
71 :      pd8 = 0xff;                         /*                              */
72 :      pd9 = 0x3f;                         /*                              */
73 :      pur0 = 0x04;                        /* P1_3～P1_0のプルアップON     */
74 :  }
75 :
76 :  /************************************************************************/
77 :  /* end of file                                                          */
78 :  /************************************************************************/
```

35.5 プログラムの解説

35.5.1 ヘッダファイルの取り込み

高速オンチップオシレータは、R8C/38C マイコンに内蔵されています。R8C/38A マイコンには内蔵されていません。インクルードする SFR の定義ファイルは、R8C/38C 用を定義します。

```
20 :  #include "sfr_r838c.h"                  /* R8C/38C SFR の定義ファイル     */
```

35.5.2 init 関数(クロックの選択)

init 関数で、R8C/38C マイコンに内蔵されている機能の初期化を行います。hoco_io.c では、init 関数内で①クロックの選択 ②I/O ポートの入出力設定 ③端子のプルアップを行います。ここでは、クロックの選択について説明します。

```
 49 :  void init( void )
 50 :  {
 51 :      int i;
 52 :
 53 :      /* ｸﾛｯｸを高速ｵﾝﾁｯﾌﾟｵｼﾚｰﾀ(40MHz)に変更(ﾌﾟﾛｸﾞﾗﾑは20MHzで動作) */
 54 :      prc0  = 1;                          /* プロテクト解除                */
 55 :      fra00 = 1;                          /* 高速オンチップオシレータ発振 */
 56 :      for(i=0; i<50; i++ );               /* 安定するまで少し待つ(約10ms) */
 57 :      fra01 = 1;                          /* 高速オンチップオシレータ選択 */
 58 :      prc0  = 0;                          /* プロテクトON                  */
```

(1) R8C/38C のクロック

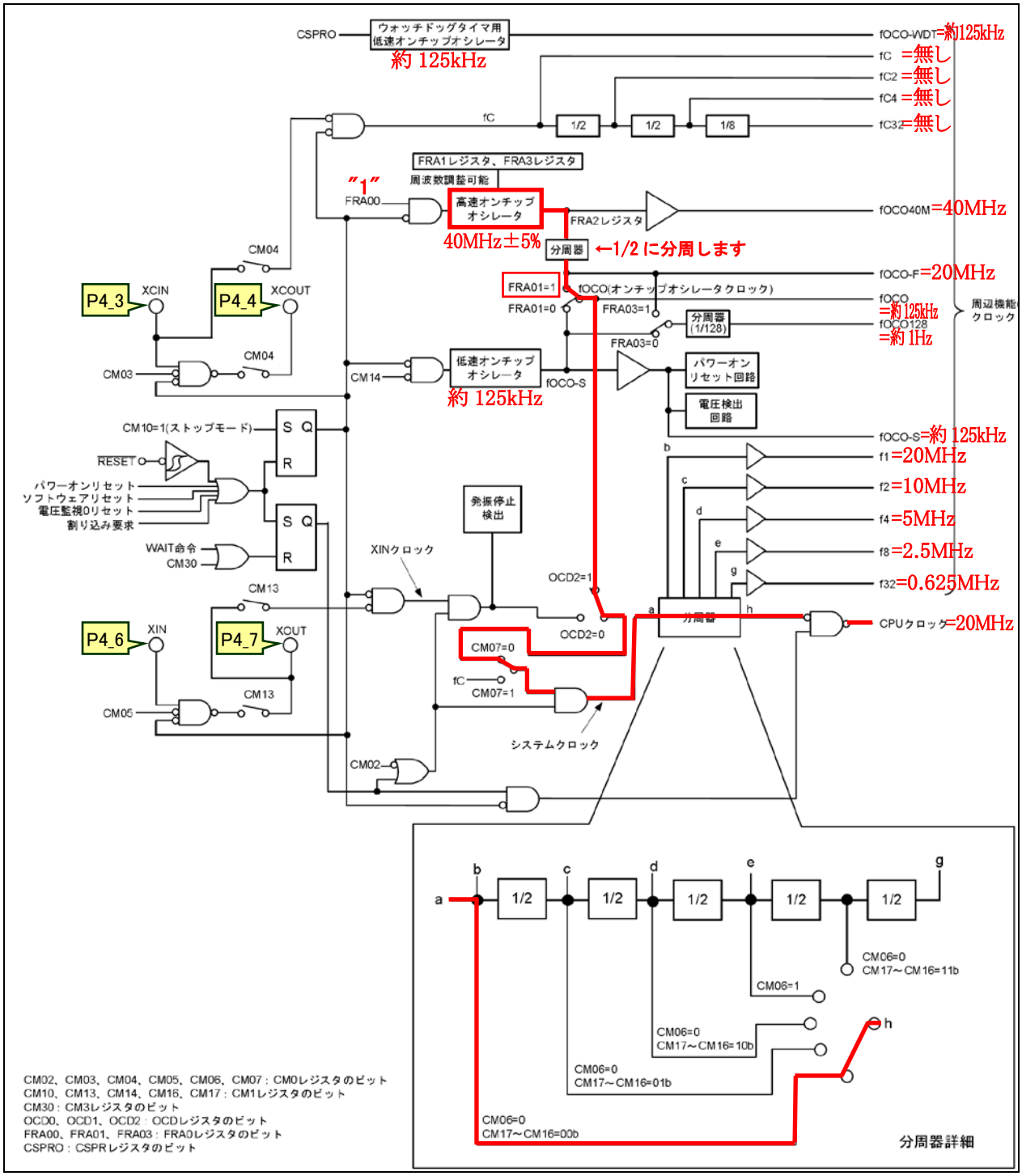
R8C/38C は、3 種類のオシレータ(発振器)が内蔵されています。また、2 種類のクリスタルを外付けすることができます。どのクロックを使うかは、プログラムで設定します。下表にまとめます。

	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)	ｳｫｯﾁﾄﾞｯｸﾞﾀｲﾏ用 低速ｵﾝﾁｯﾌﾟｵｼﾚｰﾀ (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	RY_R8C38 ボードは 20MHz を実装	32.768kHz	40MHz±5% (38.0～42.0MHz)	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		オプションです。標準では、取り付けていません。	**リセット後、プログラムでこのクロックに切り替えます。**	リセット直後のみ使用します。すぐに、高速オンチップオシレータに切り替えます。	使用しません。

マイコンが動作すると低速オンチップオシレータで動作します。プログラムで高速オンチップオシレータで動作するように切り替えます。

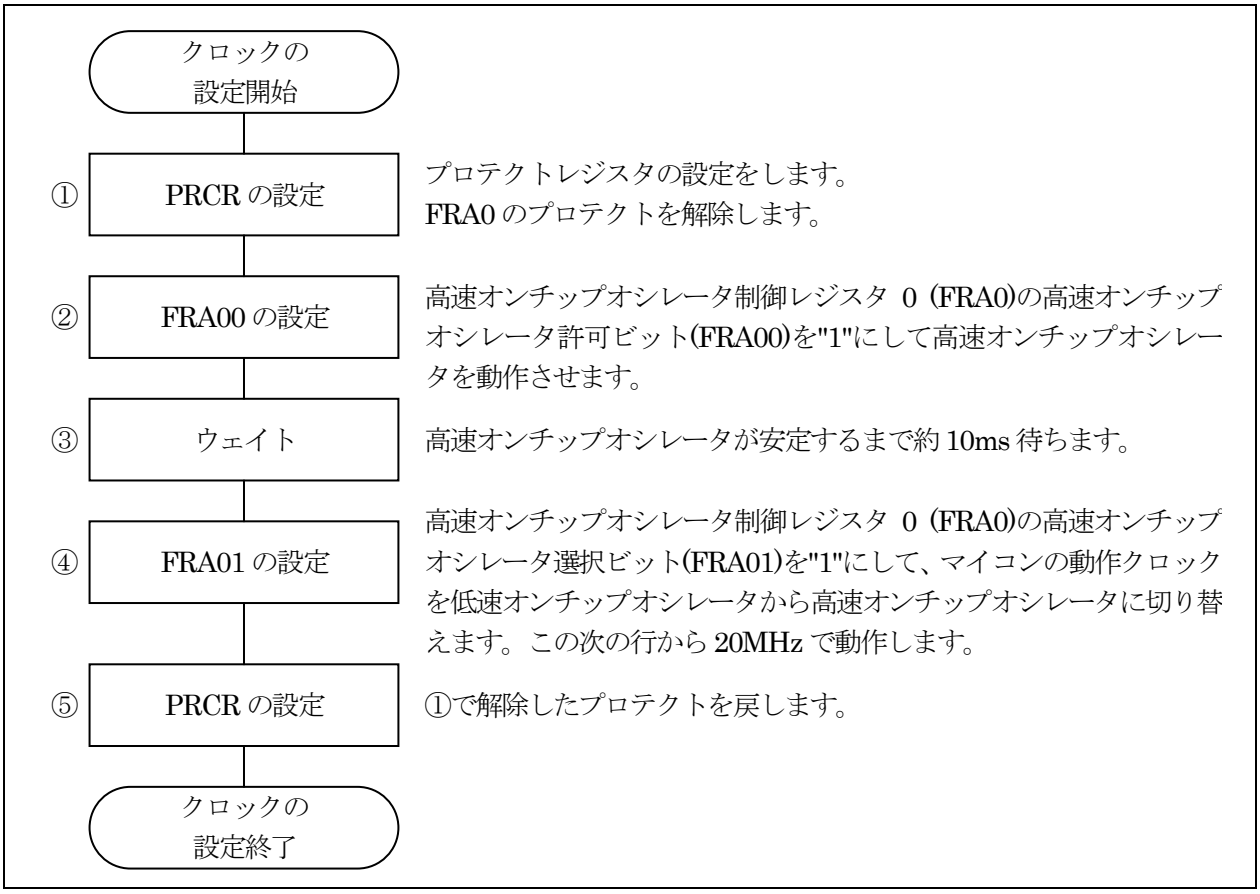
(2) ブロック図

R8C/38C のクロック発生回路を下記に示します。太線のように切り替えて、高速オンチップオシレータで動作させます。



(3) クロックの設定

R8C/38C 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高速オンチップオシレータに切り替えて、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/38C 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 FRA0 の設定をします。このレジスタはプロテクトレジスタ(PRCR)のプロテクトビット 0(PRC0)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、このレジスタを書き込み許可するためにプロテクトビット 0 を″1″にします。下記にプログラムを示します。

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

②高速オンチップオシレータ制御レジスタ 0(FRA0：High-speed on-chip oscillator control register 0)の設定

低速オンチップオシレータから、高速オンチップオシレータに切り替えます。FRA0 は保護されています。このレジスタの値を書き替えるときは、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
bit7～4		″0″を設定	変更なし
bit3	fOCO128 クロック選択ビット fra03	0：fOCO-S の 128 分周を選択 1：fOCO-F の 128 分周を選択	変更なし
bit2		″1″を設定	変更なし
bit1	高速オンチップオシレータ選択ビット(注 1) fra01	0：低速オンチップオシレータ選択(注 2) 1：高速オンチップオシレータ選択(注 3)	変更なし
bit0	高速オンチップオシレータ許可ビット fra00	0：高速オンチップオシレータ停止 1：高速オンチップオシレータ発振	1

注 1. FRA01 ビットは次の条件のとき変更してください。
- ·FRA00=1 (高速オンチップオシレータ発振)
- ·CM1 レジスタの CM14=0 (低速オンチップオシレータ発振)
- ·FRA2 レジスタの FRA22～FRA20 ビットが
  VCC=2.7V～5.5V の場合は全分周モード設定可能 “000b”～“111b”
  VCC=1.8V～5.5V の場合は 8 分周以上の分周比 “110b”～“111b”(8 分周モード以上)

注 2. FRA01 ビットに“0”(低速オンチップオシレータ選択)を書くとき、同時に FRA00 ビットに“0”(高速オンチップオシレータ停止)を書かないでください。FRA01 ビットを“0”にした後、FRA00 ビットを“0”にしてください。

注 3. FRA01 ビットに“1”(高速オンチップオシレータ選択)を設定し、低速オンチップオシレータを停止する場合は、低速オンチップオシレータの1サイクル以上を待ってから、CM1 レジスタの CM14 ビットを“1”(低速オンチップオシレータ停止)にしてください。

リセット後、高速オンチップオシレータは発振していません。FRA00 ビットを″1″にして発振させます。

```
55 :        fra00 = 1;                          /* 高速オンチップオシレータ発振 */
```

③ウエイト

②で高速オンチップオシレータを発振させました。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、何もせずに繰り返す」命令を実行します。実測で約 10ms です。

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

④高速オンチップオシレータ制御レジスタ 0(FRA0：High-speed on-chip oscillator control register 0)の設定

高速オンチップオシレータの発振が安定したら、このクロックに切り替えます。高速オンチップオシレータ制御レジスタ 0 の高速オンチップオシレータ選択ビット(FRA01)を"1"にします。

```
57 :        fra01 = 1;                          /* 高速オンチップオシレータ選択 */
```

⑤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）。

```
58 :        prc0  = 0;                          /* プロテクト ON                */
```

35.5.3 その他のプログラム

その他のプログラムは「プロジェクト：io」と同様です。

36. 付録

36.1 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くてはなりません。

```
void main( void )
{
  int a,b,c ;
  a = 100 ; b = 2000 ;
  c = a * b ;
  printf("¥n 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”と言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(65,536×3＋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 (0x7F7FFFFF)	7.00649232162408 62e-46f (0x00000001)
double long double	8 バイト	1.7976931348623158e+308 (0x7FEFFFFFFFFFFFFF)	4.9406564584124655e-324 (0x0000000000000001)

36.2 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)

36.3 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

36.4 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）

36.5　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。

プログラムを下記に示します。

```
 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: }
```

表示結果を下記に示します。

```
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 !
```

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。

36.6　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が終了し、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。

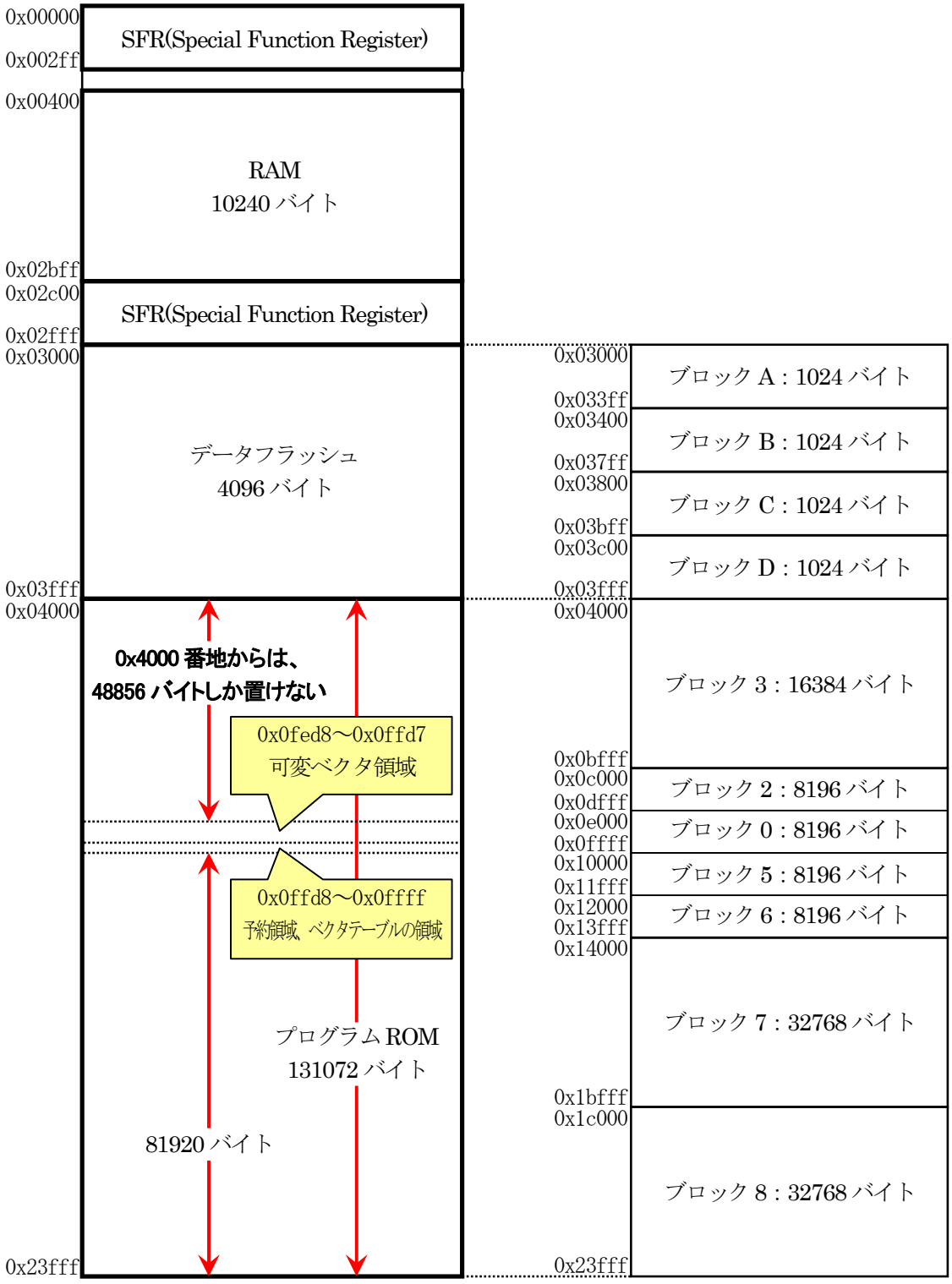
※scanf 関数は多くのメモリを消費するため R8C マイコンでは、「%e, %E, %f, %g, %G」の変換指定文字は使用できません。

36.7 プログラムの容量が 48KB 以上になったとき

36.7.1 概要

RY_R8C38 ボードの R8C/38A マイコン(R5F2138CANFP)は、プログラム ROM：128KB、データフラッシュ：4KB、RAM：10KB が内蔵されています。

プログラムは、プログラム ROM である 0x4000～0x23fff 番地に置きます。ただし、下図のように 0xfed8～0xffff 番地は、可変ベクタ領域、予約領域、固定割り込みベクタテーブルの領域のため、0x4000 番地からの約 48KB と、0x10000 番地からの約 80KB に分けなければいけません。



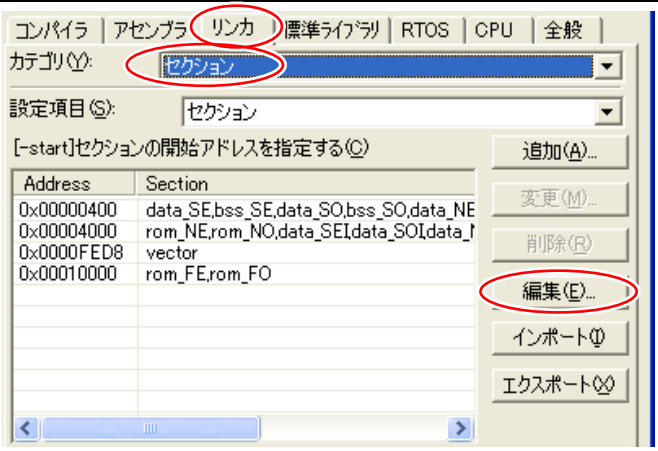
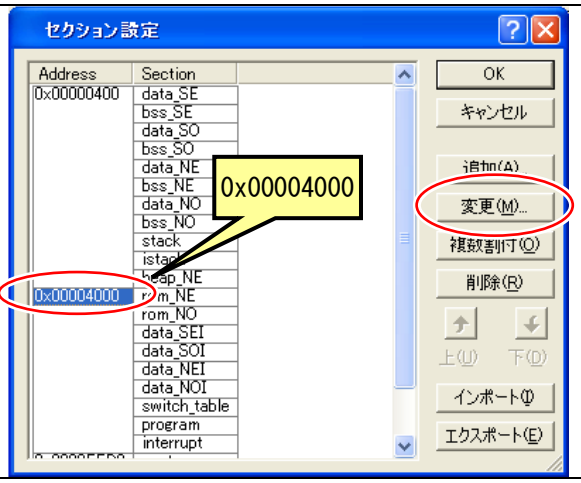
プログラムは、0x4000 番地から配置しているので、0xfed8 番地を超えると(48856 バイト以上になると)、下記のように、セクションがオーバーラップ(重なっている)というエラーが発生します。

```
hase OptLinker starting
Software license problem:
         Duration of Trial License of UNKNOWN is exhausted. (37)
Maximum link size limited to 64KB code+data.
L2321 (E) Section "vector" overlaps section "program"
L2321 (E) Section "fvector" overlaps section "program"
L2321 (E) Section "rom_FE" overlaps section "program"
L2321 (E) Section "rom_FO" overlaps section "program"
Optimizing Linkage Editor Abort
Phase OptLinker finished

Build Finished
4 Errors, 0 Warnings
```

この対策を 2 例、説明します。

36.7.2　データフラッシュを使う

パラメータの保存などで、データフラッシュのブロック A(0x3000～0x33ff 番地)を使っていても、ブロック B～D(0x3400～0x3fff 番地)を使っていることは少ないと思います。データフラッシュはパラメータの保存などに使われることが多いですが、プログラムを置いても問題ありません。そこで、ROM の先頭を 0x4000 番地から 0x3400 番地に変更して、3K バイト分の容量を確保します。

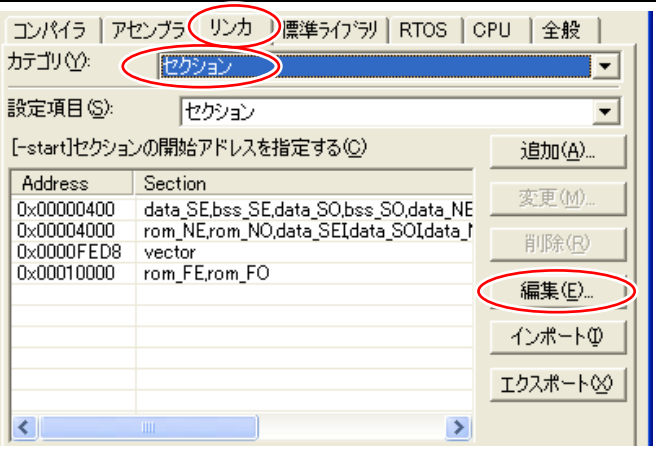
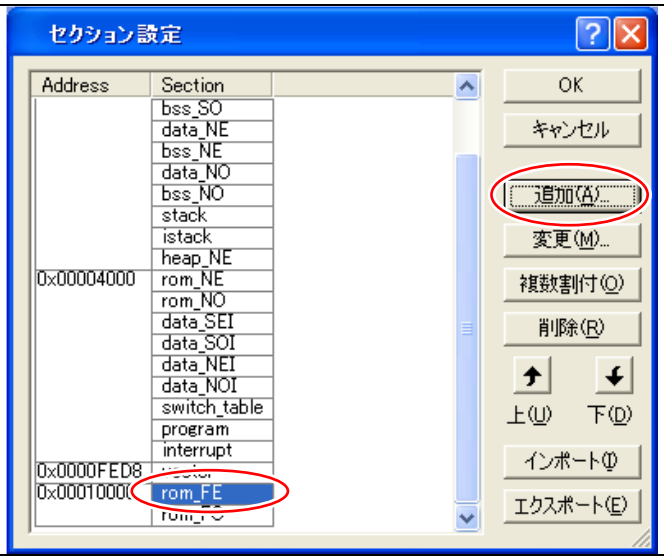
1		ルネサス統合開発環境の「ツール→Renesas M16C Standard Toolchain」を開き、「リンカ」タブを選択、「カテゴリ:セクション」を選択します。 編集をクリックします。
2		「0x00004000」部分をクリック、変更をクリックします。 「0x3400」に変更して、OK をクリックします。 ※データフラッシュを使っていなければ、「0x3000」にしても構いません。

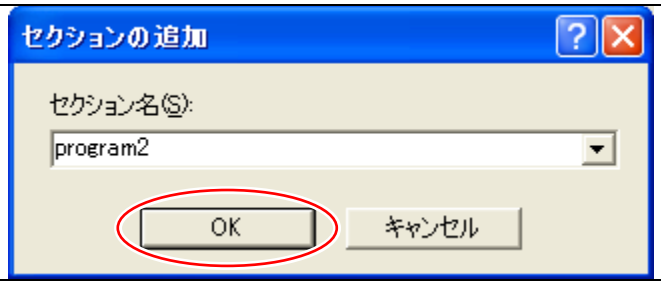
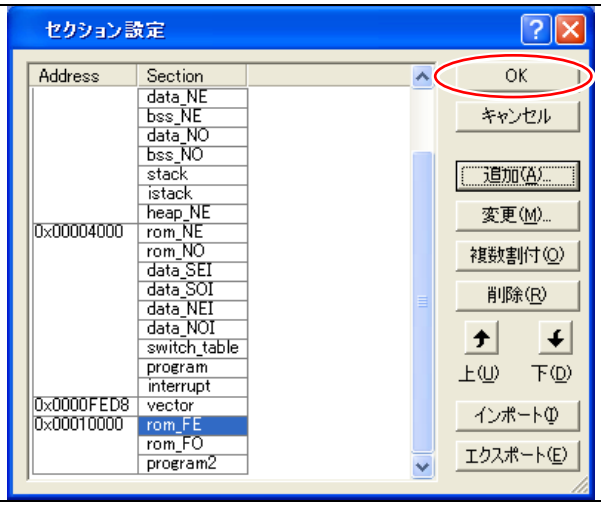
```
Phase OptLinker starting
Software license problem:
         Duration of Trial License of UNKNOWN is exhausted. (37)
Maximum link size limited to 64KB code+data.
RAMDATA SECTION:  00000cfd Byte(s)
ROMDATA SECTION:  00000a8a Byte(s)
PROGRAM SECTION:  0000c1ea Byte(s)
Optimizing Linkage Editor Completed
Phase OptLinker finished

Build Finished
0 Errors, 0 Warnings
```

3	今回の方法で 0xfed8 番地を超えなければ、ビルドを実行すると上記のように 0 Errors になります。

36.7.3　走りに関係しない関数を 0x10000 番地から配置する

0x4000 番地からのプログラムと、0x10000 番地からのプログラムに分けて、プログラム ROM に書き込みます。ただし、プログラムを 10000 番地以降に置くと、命令によっては実行スピードが遅くなることがあります(アセンブリ言語の MOV 命令(一般命令)が LDE(特殊命令)に変わるなど、します)。そのため、実行スピードが遅くなっても問題ない関数、マイコンカーの場合は初期化や液晶など走りに関係しない関数を、0x10000 番地以降に配置します。

1		ルネサス統合開発環境の「ツール→Renesas M16C Standard Toolchain」を開き、「リンカ」タブを選択、「カテゴリ:セクション」を選択します。 編集をクリックします。
2		「0x0010000」のセクション(rom_FE など)部分をクリック、追加をクリックします。

3		「program2」と入力して、OK をクリックします。
4		これで、0x10000 番地に「program2」セクションが追加されました。OK をクリックします。

```
/************************************************************************/
/* R8C/38A スペシャルファンクションレジスタ(SFR)の初期化                */
/************************************************************************/
#pragma section program program2      ←①　これ以降のプログラムは「program2」セクションに配置される
void init( void )
{
中略
}

/************************************************************************/
/* タイマ本体                                                           */
/* 引数    タイマ値 1=1ms                                               */
/************************************************************************/
#pragma section program program       ←②　これ以降のプログラムは「program」セクションに配置される
void timer( unsigned long timer_set )
{
    cnt_rb = 0;
    while( cnt_rb < timer_set );
}
```

5	上記のように、program2 セクションにしたい関数の 1 つ前に①部分を追加、0x4000 番地部分に戻したい関数の 1 つ前に②部分を追加します。 「ビルド→ビルド」を実行すると、program2 セクションに指定した関数は、0x10000 番地以降に配置されます。 ※ルネサス統合開発環境の無償評価版を使っている場合は、合計容量が 64KB を超えるとエラーが出力され、MOT ファイルはできません。この場合は、プログラムを見直して 64KB 以内になるようにしてください。

36.8　switch 文が長い場合（L2330 (E) Relocation size overflow エラー）

swtich 文部分のプログラムが大きく 32KB を超えてしまった場合、下記のようなエラーが出ます。

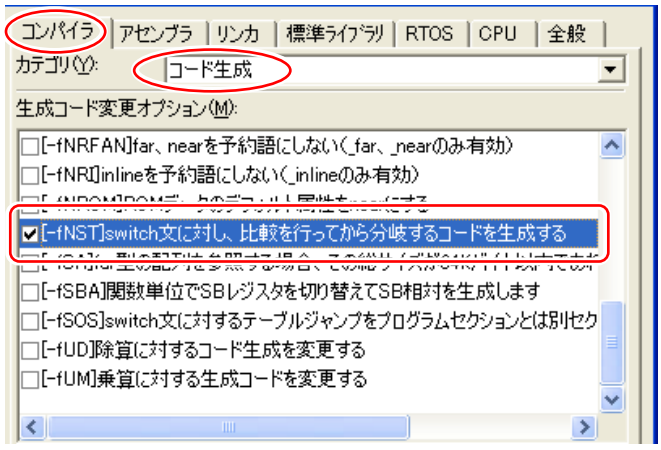
```
swtich( pattern ) {
case 0:
    プログラム
    break;
case 1:
    プログラム
    break;
  :
  数千行に及ぶプログラム
  :
}
```

プログラム容量が 32KB 以上の場合

↓　ビルドすると

```
L2330          (E)          Relocation          size          overflow          :
"C:¥WorkSpace¥sample¥sample¥Debug¥sample.obj"-"program"-"000000d5"
Optimizing Linkage Editor Abort
```

switch 文は、式（上記プログラムだと、pattern 変数）と case 文で指定した値が一致するか探し、一致したなら case 文があるアドレスにジャンプして、プログラムを実行します。このとき、ツールチェインのデフォルトの設定では、ジャンプ先アドレスが 32KB 以上先になるとアドレス指定ができなくなり、エラーがでてしまいます。
次のオプション設定をすることにより、解決できます。

1		ルネサス統合開発環境の「ツール→Renesas M16C Standard Toolchain」を開き、「コンパイラ」タブを選択、「カテゴリ:コード生成」を選択します。 **[-fNST]swtich 文に対し、比較を行ってから分岐するコードを生成する** にチェックを付けます。 OK をクリックして設定完了です。 ビルドを実行して、エラーがないか確認してください。

36.9 コンパイラオプションの最適化について

36.9.1 概要

最適化とは、C 言語プログラムをそのままアセンブリコードに変換するのではなく、同じ動作ですが実行時間を短くしたり、プログラムサイズを小さくしたりすることです。例を下記に示します。

C 言語	変換されたアセンブリ言語	マシン語
int a; 　: a = a / 4;	mov.w　-2[FB],R0 exts.w　R0 div.w　#0004H mov.w　R0,-2[FB]	73 B0 FE 7C F3 7D E1 04 00 73 0B FE

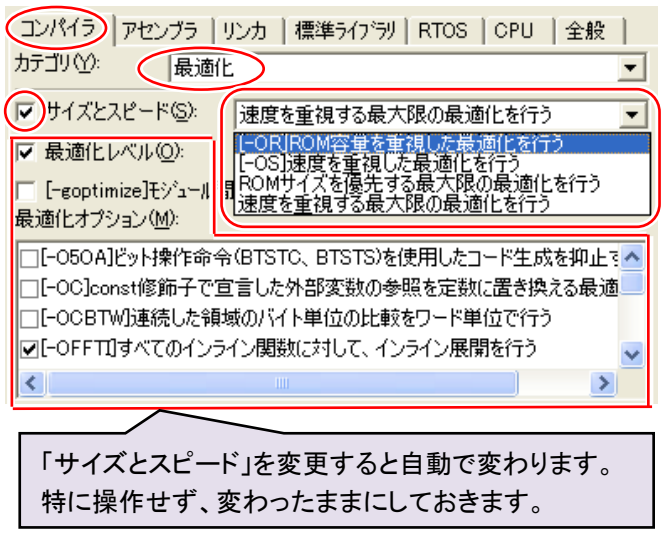
4 で割るだけなら、2 ビット右シフトしても同じです。2 ビット右シフトしたときの例を下記に示します。

C 言語	変換されたアセンブリ言語	マシン語
int a; 　: a = a >> 2;	sha.w　#-02H,-2[FB]	F1 9B FE

結果は同じですが、プログラムサイズも実行時間も短くなりました。このように、割り算をシフト命令に置き換える最適化をしました。この作業はコンパイラが自動で判断して行います。
ただし、最適化してほしくない部分も最適化してしまうことがあり、プログラムを組む場合は、若干注意が必要です（注意点は「**32.9.3　最適化すると動作しなくなるプログラム例と対策**」を参照してください）。

36.9.2 最適化を変更する

ルネサス統合開発環境の Renesas M16C Standard Toolchain Ver.6.00.00 では、下記の操作で最適化の内容を変更することができます。

1		ルネサス統合開発環境の「ツール→Renesas M16C Standard Toolchain」を開き、「コンパイラ」タブを選択、「カテゴリ:最適化」を選択します。「サイズとスピード」のチェックを付けると、最適化項目を4つ選べます。

各最適化の内容で、「kit07_38a.c」(標準のマイコンカー走行プログラム)をビルドしたときのプログラム容量、実行時間を下表に示します。

最適化内容	ROM 容量		パターン 0 内の実行時間 μs	パターン 1 内の実行時間 μs	パターン 11 内の実行時間	
	KB	最適化なしを 100 としたときの値			μs	最適化なしを 100 としたときの値
なし	2,793	100.0	11	11	80	100.0
ROM 容量重視	2,159	**77.3**	8	9	65	81.3
速度重視	2,647	94.8	8	9	65	81.3
ROM サイズ優先を最大限	2,003	**71.7**	9	9	50	**62.5**
速度重視を最大限	2,476	88.7	8	8	44	**55.0**

※ゴシック体は、1 番目、2 番目に良い項目

各最適化の内容で、「kit07_38a.c」に、液晶、microSD、ロータリエンコーダのプログラムを追加してビルドしたときのプログラム容量、実行時間を下表に示します。

最適化内容	ROM 容量		パターン 0 内の実行時間 μs	パターン 1 内の実行時間 μs	パターン 11 内の実行時間	
	KB	最適化なしを 100 としたときの値			μs	最適化なしを 100 としたときの値
なし	27,856	100.0	11	10	80	100.0
ROM 容量重視	24,710	88.7	8	7	64	80.0
速度重視	21,650	**77.7**	8	7	60	75.0
ROM サイズ優先を最大限	24,484	**87.9**	8	8	50	**62.5**
速度重視を最大限	32,332	116.1	8	7	46	**57.5**

※ゴシック体は、1 番目、2 番目に良い項目

プログラムの内容によって、最適化される内容も変わりますが、今回の内容では**「ROM サイズ優先を最大限」が、プログラム容量、実行時間ともに効果的に最適化されています**。

36.9.3 最適化すると動作しなくなるプログラム例と対策

「最適化なし」では動いていたプログラムが、最適化すると動かなくなる、もしくは動作がおかしくなることがあります。プログラムを下記に示します。

```
// グローバル変数
unsigned long cnt_rb;

void wait1s( void )
{
    cnt_rb = 0;
    while( cnt_rb < 1000 );  ←cnt_rb 変数が 1000 以下ならこの行を繰り返す
}

// タイマ RB 割り込み 1ms ごとに割り込み発生
#pragma interrupt intTRB(vect=24)
void intTRB( void )
{
    cnt_rb++;
}
```

wait1s 関数内で、cnt_rb 変数を 0 にしてから、cnt_rb 変数が 1000 以下なら while 文を繰り返しています。この部分だけ見ると、cnt_rb 変数が 1000 以上ということはあり得ないのでコンパイラは、「cnt_rb は 0 なので無限ループにしてしまおう」ということで while 文の行を「while(1);」に書き替えてしまいます。実際はタイマ RB の 1ms ごとの割り込み内で cnt_rb 変数を＋1していますので、1000ms 後に cnt_rb 変数は 1000 になり、ループを抜けているはずでした。

そこで、cnt_rb 変数に最適化を抑止する「**volatile（ボラタイル）**」命令を追加します。

```
// グローバル変数
volatile unsigned long cnt_rb;
以下略
```

volatile を付けるとコンパイラは「最適化をしてはいけない変数だな」ということで、cnt_rb 変数に関わるプログラム部分は最適化をしません。

プロジェクト「timer2」は、最適化するとうまく動作しません。どう動作しないか確かめてみてください。また、割り込みプログラム内で操作している変数に「volatile」を付けるとどうなるか、確かめてみてください。

最適化をするときのプログラムのポイントを下記に示します。

①グローバル変数は「volatile」を必ず付けます。

②関数内で使う自動変数（通常の変数）、静的変数（static を付けた変数）も、最適化されるような使い方をする場合は「volatile」を付けます（下記プログラム）。

```
void wait( void )
{
    volatile int i;

    for( i=0; i<1000; i++ );  // i が 0～999 の間、繰り返す（だけ）
}
```

37. 参考文献

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

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

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

R8C マイコンについての詳しい情報は、ルネサス エレクトロニクス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