マイコンカーラリー

H8/3048F-ONE

実習マニュアル

ルネサス統合開発環境版

第1.30版より、「c:¥worksapce¥commn」フォルダに『car_printf_3048.c』と『initsct_3048.c』を追加しました。2つのファイルと「car_printf2.c」の関係は、次の通りです。

car_printf2.c ＝ car_printf_3048.c ＋ initsct_3048.c

それぞれのファイルの内容は、次の通りです。
car_printf_3048.c …printf、scanf文を使えるようにするファイル
initsct_3048.c …… RAM領域（セクションB、R)を初期化するファイル

car_printf2.cの場合、printf文を使わなくてもRAM領域の初期化が必要なら、いっしょにプロジェクトに組み込んでしまったためプログラムサイズが大きくなっていました。分割することにより、プロジェクトの内容に合わせてそれぞれのファイルを追加できるようにしました。

第 1.32 版
2009.05.25
ジャパンマイコンカーラリー実行委員会

注 意 事 項 (rev.1.4)

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、事前にマイコンカー公式ホームページ(http://www.mcr.gr.jp/)などを通じて公開される情報に常にご注意ください。

連絡先

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

目　次

1. CPU ボード 1

1.1 仕様 1
1.2 外観 2
1.3 使用できるI/O 数 3
1.4 コネクタのピン配置 4
1.5 回路図 5
1.6 ピン配置 6
1.7 内部機能 7
1.8 メモリマップ 8

2. 実習基板 9

2.1 概要 9
2.2 外観 9
2.3 ディップスイッチ部 10
2.3.1 コネクタ 10
2.3.2 回路 11
2.3.3 回路説明 11
2.4 LED 部 12
2.4.1 コネクタ 12
2.4.2 回路 13
2.4.3 回路説明 13
2.5 トグルスイッチ部 14
2.5.1 コネクタ 14
2.5.2 回路 15
2.5.3 回路説明 16
2.6 ボリューム部 20
2.6.1 コネクタ 20
2.6.2 回路 21
2.6.3 回路説明 21
2.7 ブザー部 22
2.7.1 コネクタ 22
2.7.2 回路 23
2.7.3 ブザーの動作原理 23
2.8 SW4,SW5,SW6 を切り替えるときの注意点 24
2.9 回路図 25

3. サンプルプログラム 26

3.1 ルネサス統合開発環境 26
3.2 サンプルプログラムのインストール 26
3.2.1 CD からソフトを取得する 26
3.2.2 ホームページからソフトを取得する 26
3.2.3 インストール 27

4. 演習手順 28

4.1 ワークスペースを開く 28
4.2 プロジェクトを開く 29

4.3 ファイル編集 31
4.4 ビルド(MOT ファイルの作成) 33
4.5 書き込みソフトの起動 35
4.6 プロジェクトを変更するときの注意点 36

5. プロジェクト「io」 I/O ポート入出力(センサ基板のチェック) 38

5.1 概要 38
5.2 接続 38
5.3 プロジェクトの構成 39
5.4 プログラム「io.c」 40
5.5 プログラム「iostart.src」 40
5.6 プロジェクトのファイル構成 42
5.7 プログラム「iostart.src」の解説 43
5.7.1 電源を入れたときの動作 43
5.7.2 マイコンの動作開始 43
5.7.3 ベクタアドレスからジャンプ先アドレスを取り出す 44
5.7.4 スタートアップルーチンの実行 46
5.7.5 INITSCT 関数へジャンプ 46
5.7.6 IMPORT 宣言 47
5.7.7 main 関数へジャンプ 47
5.7.8 スタートアップルーチンの最後 48
5.8 プログラム「initsct_3048.c」の解説 49
5.9 プログラム「io.c」の解説 49
5.9.1 「machine.h」ファイルの取り込み 49
5.9.2 「h8_3048.h」ファイルの取り込み 49
5.9.3 「h8_3048.h」ファイルの内容 50
5.9.4 プロトタイプ宣言 53
5.9.5 init 関数(I/O ポートの入出力設定) 54
5.9.6 I/O ポートにデータを出力する、読み込む 57
5.9.7 main 関数 62
5.10 ビット操作のプログラムテクニック 62
5.10.1 全ビット反転する 62
5.10.2 特定のビットを"0"にする 62
5.10.3 マスク処理 63
5.10.4 特定のビットを"1"にする 65
5.10.5 特定のビットを反転する 65

6. プロジェクト「timer1」 タイマ(学校祭用電飾プログラムへの応用) 66

6.1 概要 66
6.2 接続 66
6.3 プロジェクトの構成 67
6.4 プログラム「timer1.c」 67
6.5 プログラムの解説 69
6.5.1 I/O ポートの入出力設定 69
6.5.2 timer 関数 69
6.5.3 main 関数 70

7. プロジェクト「timer2」 割り込みによるタイマ 71

7.1 概要 71
7.2 接続 71

7.3 プロジェクトの構成 71
7.4 プログラム「timer2.c」 72
7.5 プログラム「timer2start.src」 73
7.6 割り込みの概要 75
7.6.1 なぜ割り込みが必要か 75
7.6.2 割り込みの種類 76
7.7 プログラムの解説(割り込みの設定手順) 77
7.7.1 割り込みを使う設定、割り込みを許可する 78
7.7.2 割り込みプログラムの作成 87
7.7.3 「#pragma interrupt」の設定 90
7.7.4 全体の割り込みを許可する 92
7.7.5 ベクタアドレスの設定(timer2start.src ファイル) 94
7.7.6 「.IMPORT」の設定(timer2start.src ファイル) 97
7.8 プログラムの解説(割り込み以外) 98
7.8.1 main 関数 98
7.8.2 timer 関数 98
7.9 まとめ 99

8. プロジェクト「ad」 A/D 変換値を LED へ出力 102

8.1 概要 102
8.2 接続 102
8.3 プロジェクトの構成 103
8.4 プログラム「ad.c」 103
8.5 プログラムの解説 104
8.5.1 A/D 変換の初期設定 104
8.5.2 A/D 変換値を読み込む get_ad 関数 106
8.5.3 main 関数 108
8.6 まとめ 109

9. プロジェクト「ad2」 A/D 変換値を LED へ出力(スキャンモード) 110

9.1 概要 110
9.2 接続 110
9.3 プロジェクトの構成 111
9.4 プログラム「ad2.c」 111
9.5 プログラムの解説 113
9.5.1 A/D 変換の初期設定(スキャンモード) 113
9.5.2 get_ad0 関数 115
9.5.3 get_ad1 関数 115
9.5.4 main 関数 115
9.6 まとめ 116

10. プロジェクト「pwm1」 ITU3(または 4)を使った PWM 信号出力 117

10.1 概要 117
10.2 接続 117
10.3 プロジェクトの構成 118
10.4 プログラム「pwm1.c」 118
10.5 プログラムの解説 119
10.5.1 PWM とは? 119
10.5.2 ITU を使った PWM 信号出力 120
10.5.3 時間の単位 121

10.5.4 ITU の初期設定 121
10.5.5 ITU を PWM 機能として使用するポイント 128
10.5.6 main 関数 132
10.5.7 ITU4 を使用する場合 134
10.6 まとめ 135

11. プロジェクト「pwm2」 ITU0(または 1、2)を使った PWM 信号出力 136

11.1 概要 136
11.2 接続 136
11.3 プロジェクトの構成 137
11.4 プログラム「pwm2.c」 137
11.5 プログラムの解説 138
11.5.1 PWM 出力端子 138
11.5.2 ITU0 の初期設定 138
11.5.3 メインプログラム 139
11.6 まとめ 142

12. プロジェクト「pwm21」 ITU0(または 1、2)を使った PWM 信号出力その2 143

12.1 概要 143
12.2 接続 143
12.3 プロジェクトの構成 144
12.4 プログラム「pwm21.c」 144
12.5 プログラムの解説 146
12.5.1 「pwm2.c」のPWM0%は本当に0%か 146
12.5.2 「pwm2.c」のPWM100%は本当に100%か 146
12.5.3 正真正銘の100%PWM出力にする！ 147
12.5.4 正真正銘の0%PWM出力にする！ 148
12.5.5 pwm0 関数 150
12.5.6 main 関数 151
12.6 まとめ 151

13. プロジェクト「pwm3」 リセット同期 PWM モードを使った PWM 信号出力 152

13.1 概要 152
13.2 接続 152
13.3 プロジェクトの構成 153
13.4 プログラム「pwm3.c」 153
13.5 プログラムの解説 155
13.5.1 リセット同期 PWM モードの設定 155
13.5.2 PWM 出力される仕組み 160
13.5.3 リセット同期 PWM モードの注意点 165
13.5.4 main 関数 171
13.6 まとめ 172

14. プロジェクト「servo」 サーボ制御 174

14.1 概要 174
14.2 接続 174
14.3 プロジェクトの構成 175
14.4 プログラム「servo.c」 176
14.5 モータドライブ基板 Vol.3 について 177
14.5.1 概要 177

14.5.2 部品実装図 178
14.6 コネクタ 179
14.6.1 10 ピンコネクタ 179
14.6.2 サーボの制御回路 180
14.7 サーボの制御 181
14.8 プログラムの解説 181
14.8.1 ポート B の入出力設定 181
14.8.2 ポート B に出力する値の初期値 182
14.8.3 PBDR と PBDDR の設定する順番 182
14.8.4 リセット同期 PWM モードの設定 183
14.8.5 mian 関数 185
14.9 まとめ 186

15. プロジェクト「motor」 モータの制御 188

15.1 概要 188
15.2 接続 188
15.3 プロジェクトの構成 189
15.4 プログラム「motor.c」 190
15.5 モータの制御について 191
15.5.1 モータドライブ基板の役割 191
15.5.2 スピード制御の原理 191
15.5.3 正転、逆転、ブレーキの原理 192
15.5.4 H ブリッジ回路 193
15.5.5 Hブリッジ回路のスイッチをFETにする 193
15.5.6 PチャネルとNチャネルの短絡防止回路 196
15.5.7 モータドライブ基板のモータ制御回路 199
15.5.8 モータドライブ基板 Vol.3 の接続 200
15.6 プログラムの解説 201
15.6.1 リセット同期 PWM モードの設定 201
15.6.2 main 関数 201
15.6.3 演習 201
15.7 まとめ 202

16. プロジェクト「enc」 外部のパルスをカウント(1相) 204

16.1 概要 204
16.1.1 ロータリエンコーダとは 204
16.1.2 市販されているロータリエンコーダ(1相出力) 205
16.1.3 ロータリエンコーダの自作 205
16.2 接続 206
16.2.1 ロータリエンコーダ使用 206
16.2.2 実習基板で代用 207
16.2.3 接続信号のチャタリングについて 208
16.3 プロジェクトの構成 208
16.4 プログラム「enc.c」 209
16.5 プログラムの解説 209
16.5.1 ITU2 の初期設定 209
16.5.2 I/O ポートの入出力設定 211
16.5.3 main 関数 212
16.6 まとめ 212

17. プロジェクト「enc2」 外部のパルスをカウント（2相） 213
17.1 概要 213
17.1.1 2 相出力のロータリエンコーダ 213
17.1.2 市販されている 2 相出力のロータリエンコーダ 214
17.2 接続 214
17.2.1 ロータリエンコーダ使用 214
17.2.2 実習基板で代用 215
17.3 プロジェクトの構成 216
17.4 プログラム「enc2.c」 216
17.5 プログラムの解説 217
17.5.1 ITU2 を位相計数モードで使う 217
17.5.2 main 関数 218
17.6 まとめ 219
18. プロジェクト「pulsein」 ラジコン受信機のパルス幅を測定 220
18.1 概要 220
18.2 接続 220
18.3 プロジェクトの構成 221
18.4 プログラム「pulsein.c」 221
18.5 プログラム「pulseinstart.src」 224
18.6 プログラムの解説 225
18.6.1 パルス幅を測定する設定 225
18.6.2 interrupt_imia0 関数 （割り込みプログラム） 227
18.7 まとめ 228
19. プロジェクト「beep2」 ブザーを鳴らす 229
19.1 概要 229
19.2 接続 229
19.2.1 実習基板を使用 229
19.2.2 トレーニングボード(MCR2004A 基板)を使用 230
19.3 プロジェクトの構成 230
19.4 プログラム「beep2.c」 231
19.5 プログラムの解説 233
19.5.1 音階とは 233
19.5.2 音階の define 定義 234
19.5.3 音を鳴らす note 関数 234
20. プロジェクト「beep_haru」 ブザーを鳴らす応用「春の小川」を演奏 235
20.1 概要 235
20.2 接続 235
20.3 プロジェクトの構成 235
20.4 プログラム「beep_haru.c」 235
20.5 プログラムの解説 239
20.5.1 シンボル定義 239
20.5.2 音楽データ 239
20.5.3 割り込み 240
20.5.4 main 関数 240
21. プロジェクト「beep_clock」 ブザーを鳴らす応用「大きな古時計」を演奏 241

21.1 概要 241
21.2 接続 241
21.3 プロジェクトの構成 241
21.4 プログラム「beep_clock.c」 241
21.5 プログラムの解説 245

22. プロジェクト「wdt」 WDT 割り込みによるタイマ 246

22.1 概要 246
22.2 接続 246
22.3 プロジェクトの構成 247
22.4 プログラム「wdt.c」 247
22.5 プログラム「wdtstart.src」 248
22.6 プログラムの解説 249
22.6.1 WDT とは？ 249
22.6.2 WDT の設定 250
22.6.3 割り込みプログラム 252
22.6.4 メインプログラム 253
22.6.5 wdtstart.src の追加、変更 254

23. プロジェクト「rc_timer」 リフレッシュコントローラ割り込みによるタイマ 255

23.1 概要 255
23.2 接続 255
23.3 プロジェクトの構成 256
23.4 プログラム「rc_timer.c」 256
23.5 プログラム「rc_timerstart.src」 257
23.6 プログラムの解説 258
23.6.1 リフレッシュコントローラとは？ 258
23.6.2 リフレッシュコントローラの設定 259
23.6.3 割り込みプログラム 263
23.6.4 メインプログラム 264
23.6.5 rc_timerstart.src の追加、変更 265

24. プロジェクト「sio」 パソコンから数値を入力して LED に出力する 266

24.1 概要 266
24.2 接続 266
24.3 プロジェクトの構成 267
24.4 プログラム「sio.c」 267
24.5 プログラム「car_printf_3048.c」 268
24.6 プログラム「siostart.src」 270
24.7 パソコンとマイコンの通信 272
24.7.1 概要 272
24.7.2 パソコンとの接続方法 273
24.8 プログラムの解説 274
24.8.1 car_printf_3048.c とは？ 274
24.8.2 siostart.src ファイル IMPORT 追加とベクタアドレスの変更 274
24.8.3 sio.c ファイル include 文追加 275
24.8.4 sio.c ファイル init_sci1 関数の追加 275
24.8.5 sio.c ファイル printf 文、scanf 文の使用 277
24.8.6 scanf 文の文字列の意味 278
24.8.7 no_float.h ファイルとは？ 279

24.9 まとめ 280
24.10 演習手順 281
24.10.1 パソコン設定する前準備 281
24.10.2 TeraTermPro のインストール 281
24.10.3 TeraTermPro を使用したマイコンーパソコン通信 285
24.10.4 TeraTermPro を立ち上げた状態でプログラムの書き込みをするには 288
24.10.5 TeraTermPro の画面をクリア 289
24.10.6 TeraTermPro の通信内容を保存 289

25. プロジェクト「adsio」 電圧をパソコンへ出力 291

25.1 概要 291
25.2 接続 291
25.3 プロジェクトの構成 292
25.4 プログラム「adsio.c」 292
25.5 プログラムの解説 294
25.5.1 A/D 変換の初期設定 294
25.5.2 割り込みの設定 294
25.5.3 get_ad 関数 294
25.5.4 interrupt_timer0 関数(割り込みプログラム) 294
25.5.5 main 関数 295

26. プロジェクト「adsio2」 電圧をパソコンへ出力 297

26.1 概要 297
26.2 接続 297
26.3 プロジェクトの構成 298
26.4 プログラム「adsio2.c」 298
26.5 プログラムの解説 300
26.5.1 保存するデータ数 300
26.5.2 グローバル変数 300
26.5.3 main 関数 300
26.5.4 開始待ち 301
26.5.5 データ取得 301
26.5.6 データ出力 301
26.5.7 終了 302

27. プロジェクト「adpwm」 A/D 値に応じて PWM のデューティ比を変える 303

27.1 概要 303
27.2 接続 303
27.3 プロジェクトの構成 304
27.4 プログラム「adpwm.c」 304
27.5 プログラムの解説 306

28. プロジェクト「adpwm2」 A/D 値に応じて PWM のデューティ比を変える(正転、逆転あり) 307

28.1 概要 307
28.2 接続 307
28.3 プロジェクトの構成 307
28.4 プログラム「adpwm2.c」 307
28.5 プログラムの解説 310
28.5.1 data_cnv 関数ーA/D 値の変換 310
28.5.2 main 関数 311

29. 付録 312
29.1 H8 マイコンの変数のサイズ 312
29.2 演算子 313
29.3 優先順位 314
29.4 型が混合したときの演算 315
29.5 printf 関数の使い方 316
29.6 scanf 関数の使い方 318
30. 参考文献 319

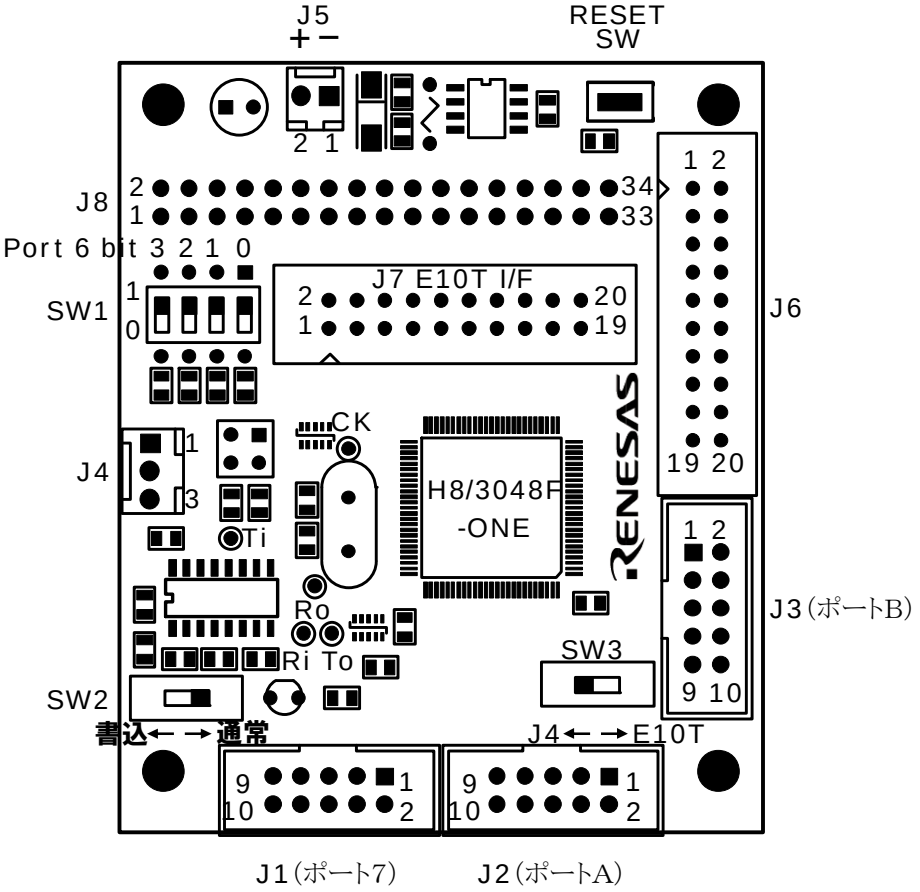
1. CPUボード

1.1 仕様

内容	詳　細
CPU ボード型式	(株)北斗電子　RY3048Fone ※TypeH バージョンは、2005 年度より支給されている型式です。今までのボードとの違いは、SW3 が実装されなくなりました。使用上の違いはまったくありません。
マイコン	(株)ルネサス テクノロジ　H8/3048F-ONE (HD64F3048BF25)
動作クロック	24.576[MHz]
電源電圧	5.0V±10% (4.5～5.5V) **H8/3048F-ONE は電源電圧の適正範囲が狭くなっています。** **アルカリ乾電池は、1.5V4 本ですと、6V 以上となり 5.5V を超えてしまいますので、電源安定化素子を付加して 5.0V 一定になるようにして使用ください。** **単三型二次電池は 1.2V4 本で、4.8V と適正範囲内になっています。**
フラッシュ ROM 書き込み回数	保証している最大書き込み回数:100 回 ただし、あくまでメーカ保証が 100 回というだけで、実際はそれ以上、書き換え可能です。書き換え回数についてはあまり気にしないで大丈夫です。 ※書き込み回数の上限を超えたため書き込みができなくなった CPU は、CPU を交換することによりボードは使えるようになりますが、業者に依頼する場合は工賃の方が高くなり、CPU ボードを買った方が安くなります。 ※明らかに書き換え回数を超えていないのに急に書き込みができなくなった場合は、下記の点を確認、対処してください。 ・SW3が E10T 側になっている　→J4 側へ切り換える ・通信 IC(U2)が壊れた　→新しい IC:SP232ECN または互換品に交換する
外形寸法	W70×D59×H13mm(実寸)
固定穴寸法	60.0×50.0mm　φ3.0mm ネジ　又は 60.96×50.80mm　φ3.0mm ネジ

10 ピン以外の I/O コネクタが未実装になっていますので、使用するときは手持ちのコネクタを使用します。

1.2 外観



J1、J2、J3	10 ピンコネクタは、CPU の各ポートに接続されています。 J1 はポート 7、J2 はポート A、J3 はポート B に接続されています。
J4	RS-232C 接続用 3 ピンコネクタ メス側コネクタは、同封されています。RS-232C 側は線が剥き出しですので RS-232C コネクタを付ける必要があります。ケーブル長は、約 1.5m です。
J5	電源用 2 ピンコネクタ メス側コネクタは、同封されています。ケーブル長は約 30cm です。 **電源電圧は、5V±10%です。極性を間違えると破損しますので気をつけてください。赤色コードが＋側、黒色コードが－側です。**
J6	20 ピン I/O コネクタ コネクタは未実装です。
J7	E10T 用 20 ピンコネクタ E10T とは、パソコンとボードをつないで H8/3048F-ONE に書き込んだプログラムが正しく動作しているかどうかチェックするデバッグ装置のことです。そのためのコネクタ領域です。コネクタは未実装です。通常の I/O としては使用できません。
J8	34 ピン I/O コネクタ コネクタは未実装です。
SW1	4 ビットディップスイッチ ポート6の bit3～0 に接続されています。スイッチに ON と書かれた側が”0”、逆側が”1”ですが、プログラムの中で反転して使用しています。今回マイコンカープログラムでは、このディップスイッチを使って走行のスピードを 16 段階調整できます。

SW2	プログラム書き込みスイッチ 「書込」側が、プログラムを CPU に書き込むときのスイッチ位置、「通常」側がプログラム実行のスイッチ位置です。**このスイッチは、必ず CPU ボードの電源が OFF の状態で操作してください。電源が入った状態で操作しますと最悪の場合、CPU が壊れます。**
SW3	RXD1 切り替えスイッチ RXD1 端子を、J4 の RS-232C 側へ接続するか、E10T 側に接続するかの切り替えです。通常は J4 から書き込みますので、J4 側にしておきます。 **2005 年度より支給のボードは、スイッチが外され、基板のパターンで RS-232C 側に繋がっています。取扱説明書の型式が、「RY3048Fone TypeH」と書かれているボードです。**

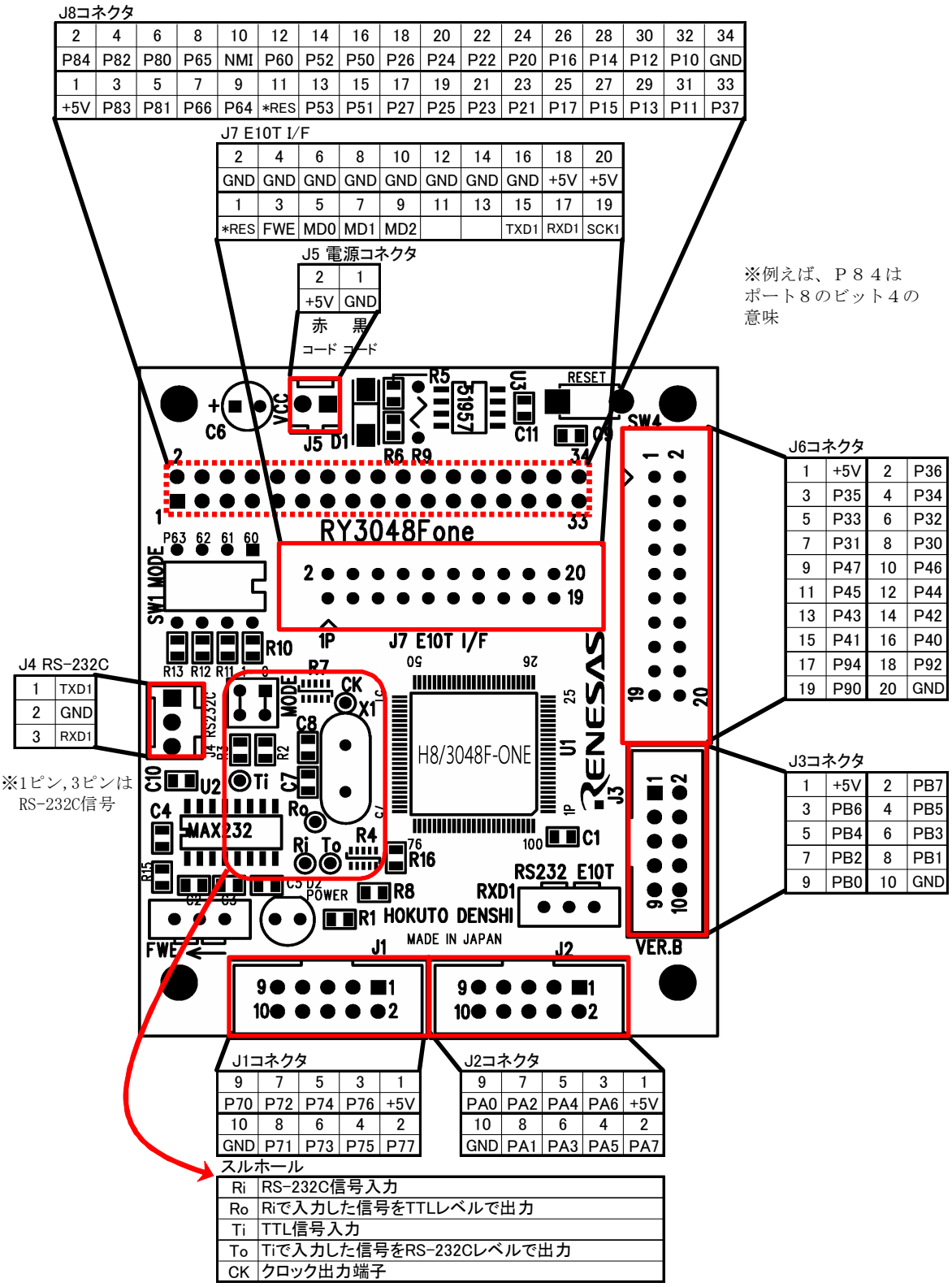
1.3 使用できるI/O数

RY3048Fone ボードは、入力専用として 8bit、入出力用として 63bit、合計 71bit 分の端子が使用可能です。パラレルインターフェース用 LSI の 8255 では 24bit(8bit×3 ポート)ですので、RY3048Fone ボードは 8255 が 3 セット分の端子があるボードということになります。

コネクタ	入力として 使用できる bit 数	入力／出力として 使用できる bit 数	備考
J1	8	－	入力専用ポートです
J2	－	8	
J3	－	8	
J6	－	18	
J7	－	－	通常の I/O としては使用できません
J8	－	29	
合計	8	63	

1.4 コネクタのピン配置

RY3048Fone ボードのコネクタとポートの位置を下図に示します。



J8コネクタ	
2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34	
P84 P82 P80 P65 NMI P60 P52 P50 P26 P24 P22 P20 P16 P14 P12 P10 GND	
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33	
+5V P83 P81 P66 P64 *RES P53 P51 P27 P25 P23 P21 P17 P15 P13 P11 P37	

J7 E10T I/F									
2	4	6	8	10	12	14	16	18	20
GND	GND	GND	GND	GND	GND	GND	GND	+5V	+5V
1	3	5	7	9	11	13	15	17	19
*RES	FWE	MD0	MD1	MD2			TXD1	RXD1	SCK1

J5 電源コネクタ	
2	1
+5V	GND
赤	黒
コード	コード

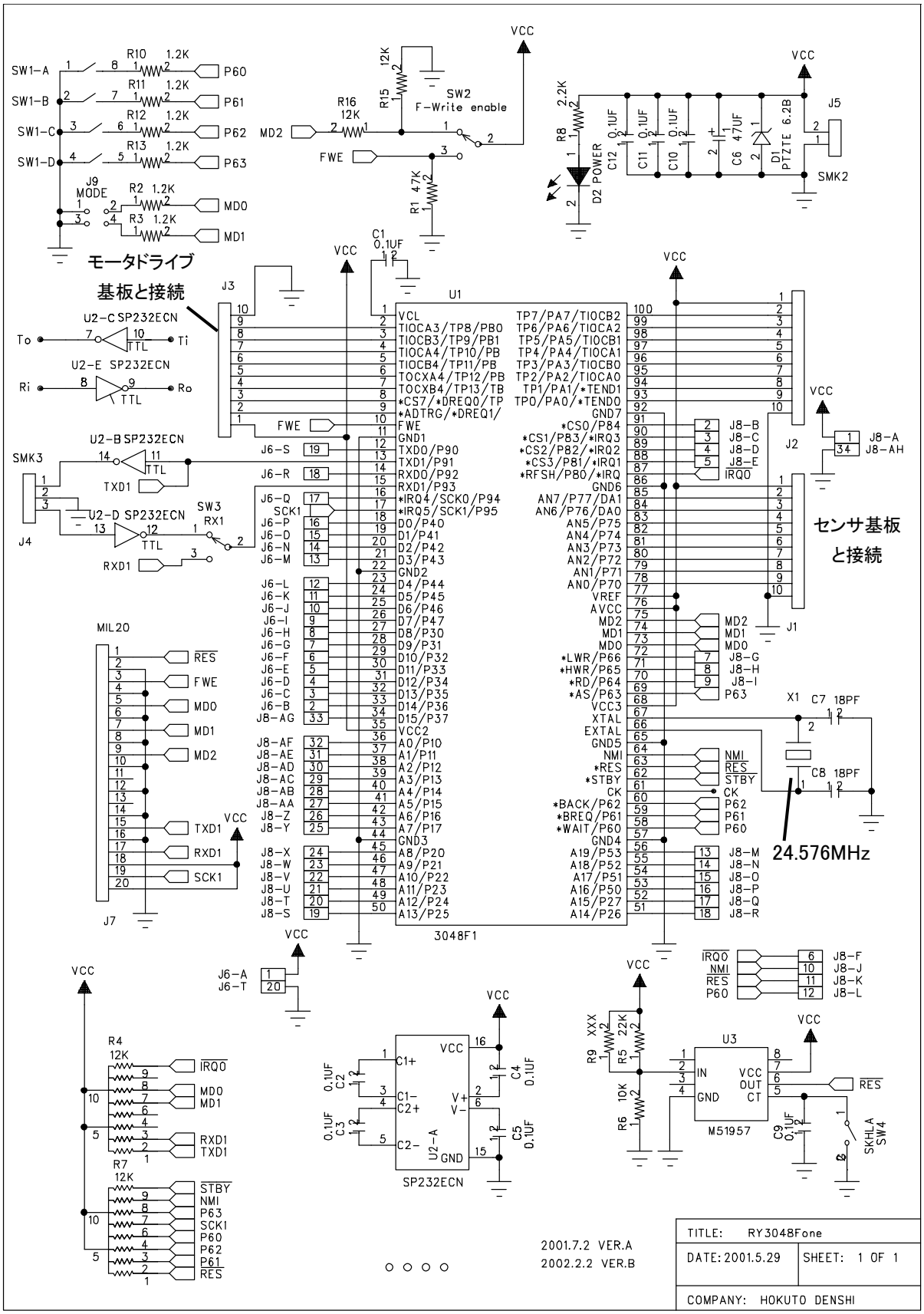
J6コネクタ			
1	+5V	2	P36
3	P35	4	P34
5	P33	6	P32
7	P31	8	P30
9	P47	10	P46
11	P45	12	P44
13	P43	14	P42
15	P41	16	P40
17	P94	18	P92
19	P90	20	GND

J3コネクタ			
1	+5V	2	PB7
3	PB6	4	PB5
5	PB4	6	PB3
7	PB2	8	PB1
9	PB0	10	GND

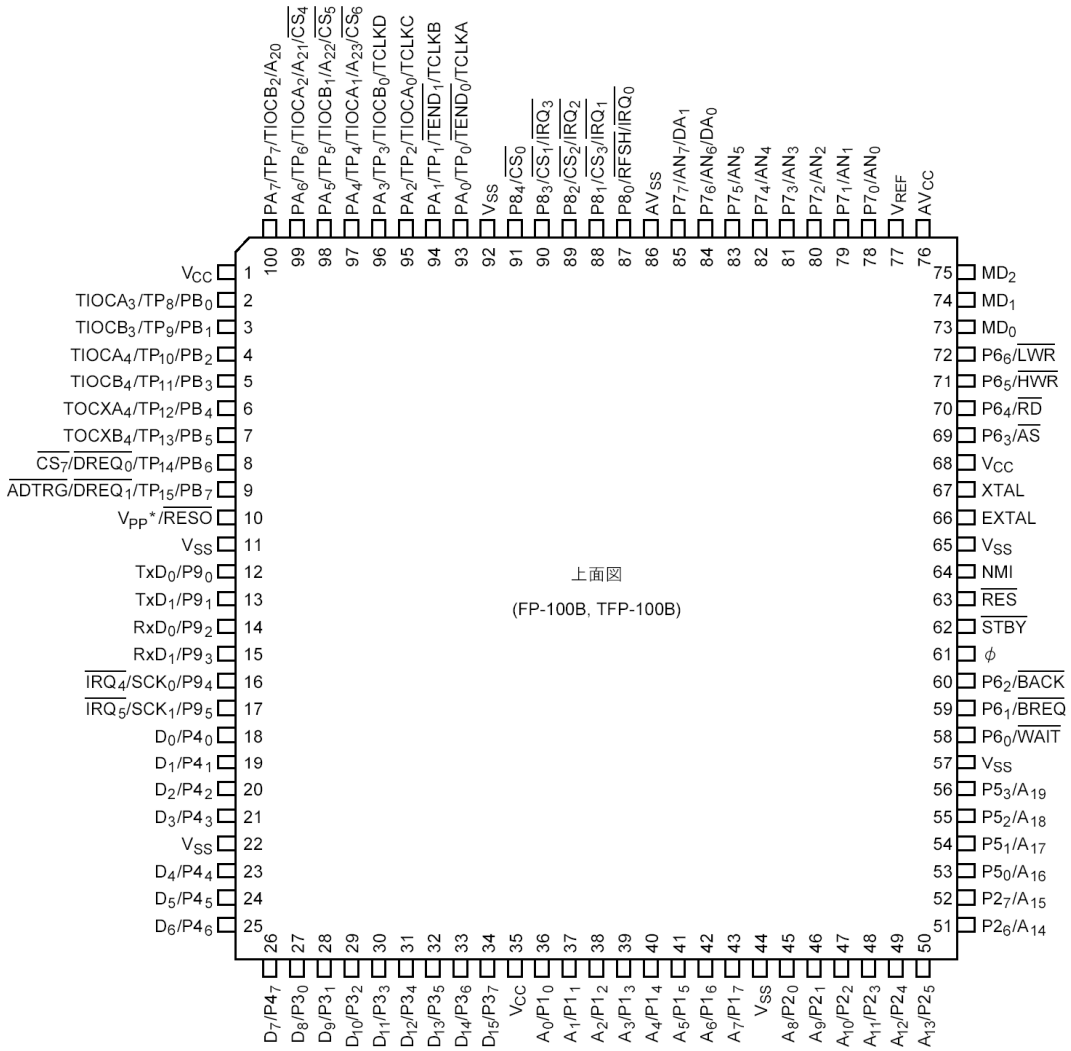
J1コネクタ					J2コネクタ				
9	7	5	3	1	9	7	5	3	1
P70	P72	P74	P76	+5V	PA0	PA2	PA4	PA6	+5V
10	8	6	4	2	10	8	6	4	2
GND	P71	P73	P75	P77	GND	PA1	PA3	PA5	PA7

スルホール	
Ri	RS-232C信号入力
Ro	Riで入力した信号をTTLレベルで出力
Ti	TTL信号入力
To	Tiで入力した信号をRS-232Cレベルで出力
CK	クロック出力端子

1.5 回路図


モータドライブ
基板と接続
センサ基板
と接続
3048F1
SP232ECN
M51957
24.576MHz
2001.7.2 VER.A
2002.2.2 VER.B
TITLE: RY3048Fone
DATE: 2001.5.29
SHEET: 1 OF 1
COMPANY: HOKUTO DENSHI

1.6 ピン配置

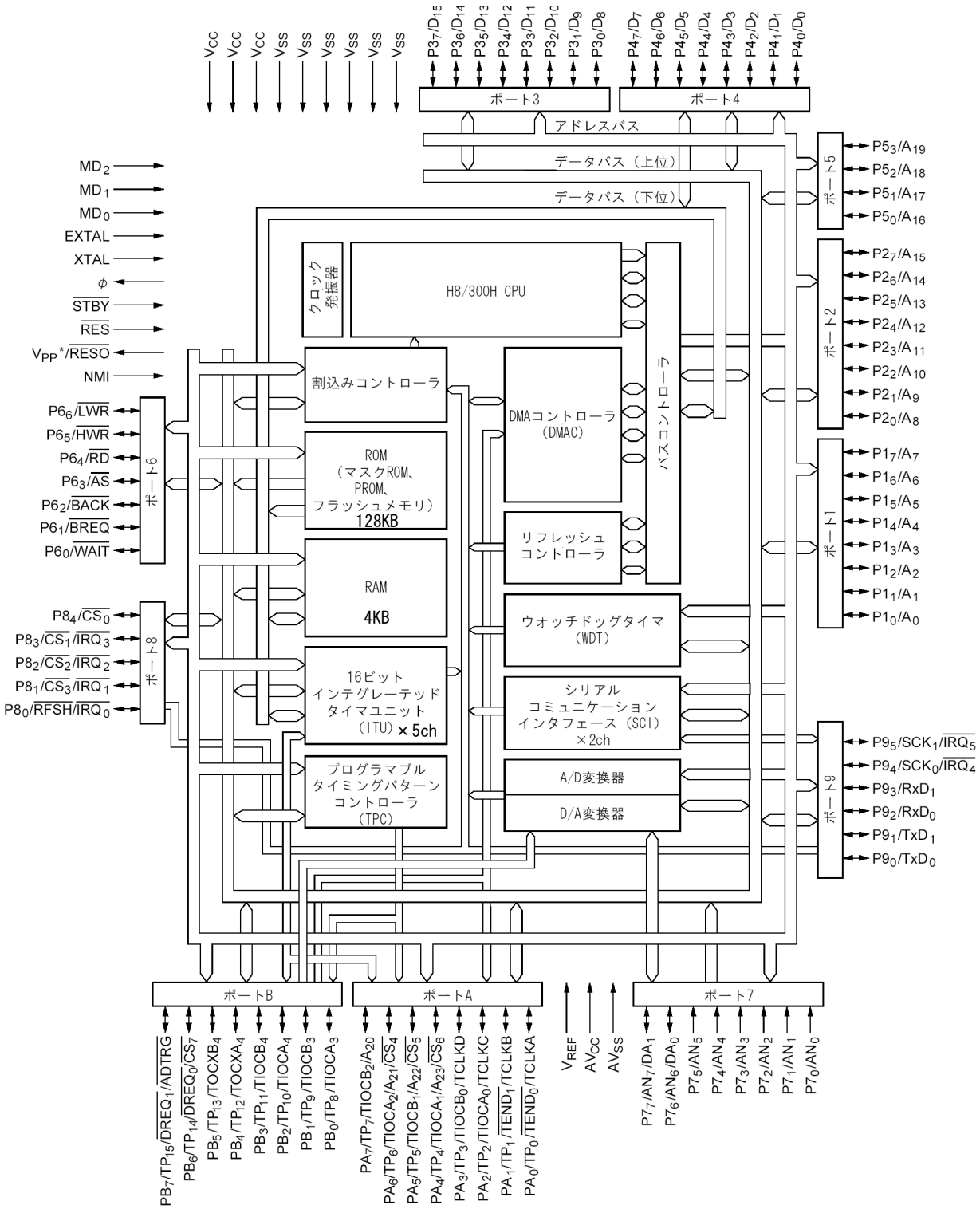


【注】* V_{PP}端子はフラッシュメモリ版のみ対応します。

1 ピンと 100 ピンの間の角が面取りされています。Vcc は＋側電圧の 5V、Vss は－側電圧の 0V を加えます。

1.7 内部機能

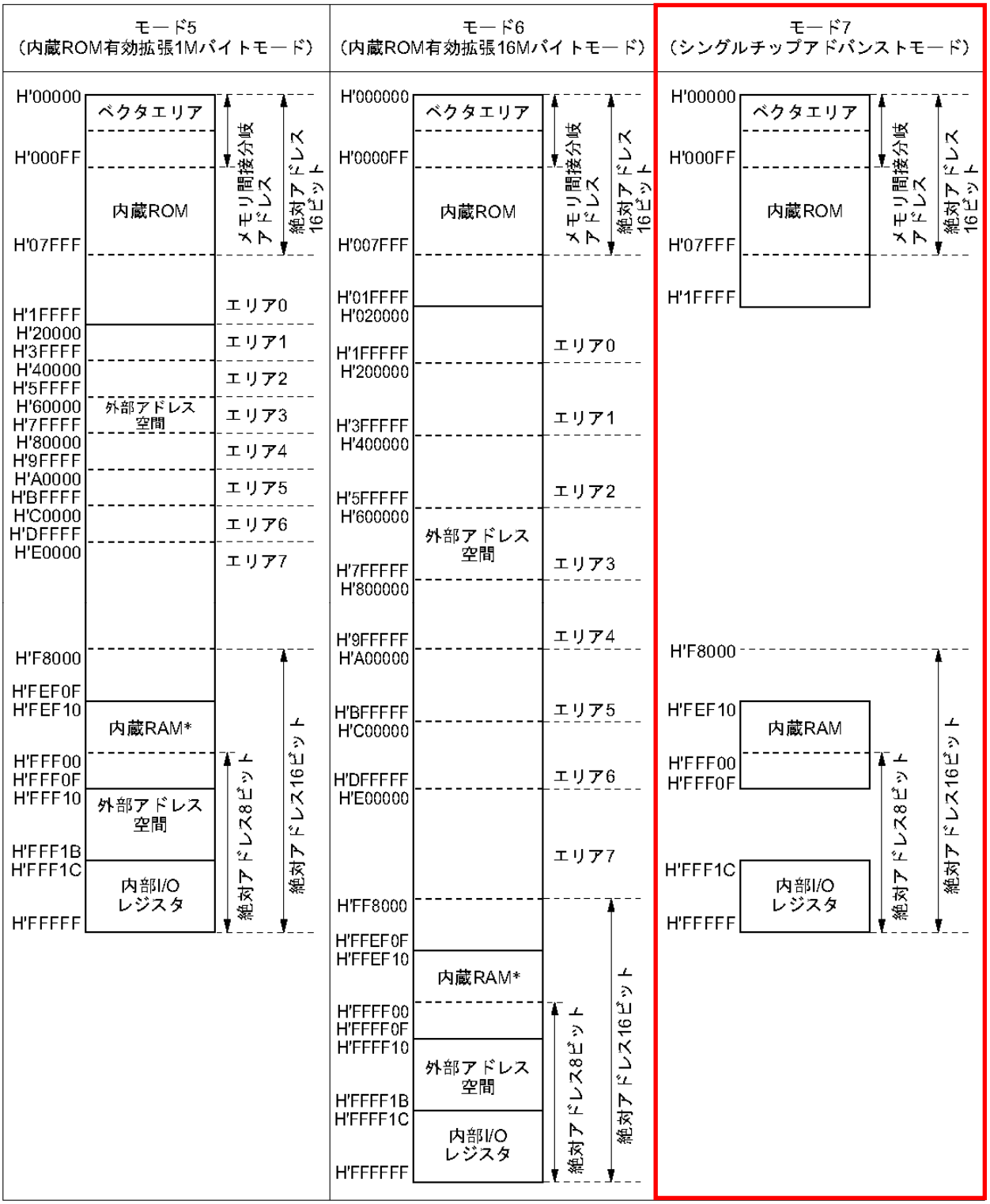
H8/3048F-ONE、および H8/3048F には、下記のような機能が内蔵されています。



【注】＊ V_{PP}端子はフラッシュメモリ版のみ対応します。

1.8 メモリマップ

H8/3048F-ONE の ROM 領域、RAM 領域、I/O レジスタ領域を下記に示します。シングルチップモード（外付けの ROM や RAM などのデバイスがない状態）で動作していますので、モード 7 のメモリマップとなります。



※H8/3048B シリーズ　ハードウェアマニュアル　第2版　3-6 ページより抜粋

この表から、H8/3048F-ONE マイコンは、

・0x00000～0x1ffff 番地が ROM (128KB)

・0xfef10～0xfff0f 番地が RAM (4KB)

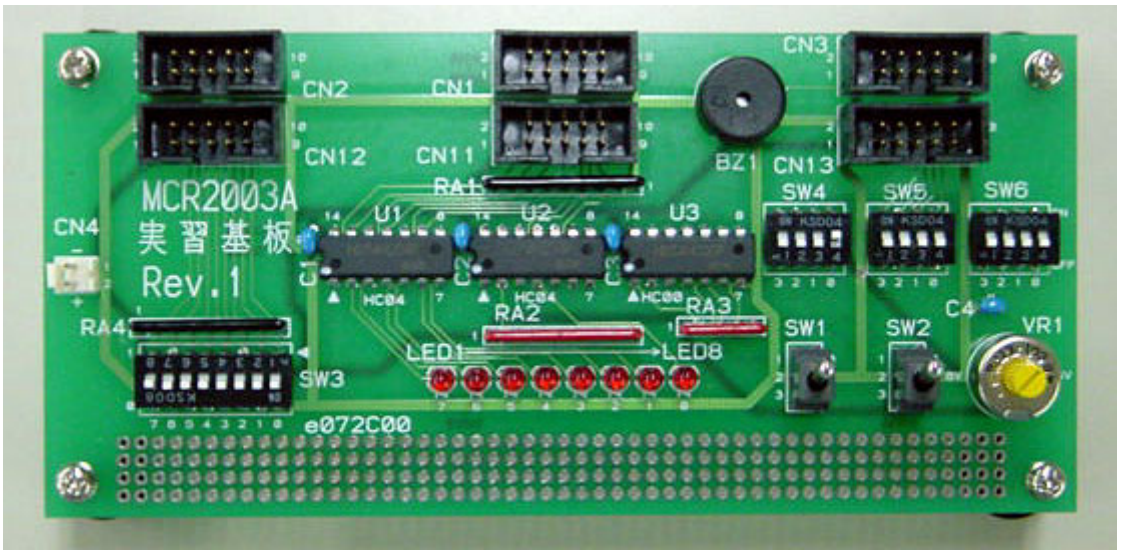
・0xfff1c～0xffffff 番地が I/O レジスタ

となります。

2. 実習基板

2.1 概要

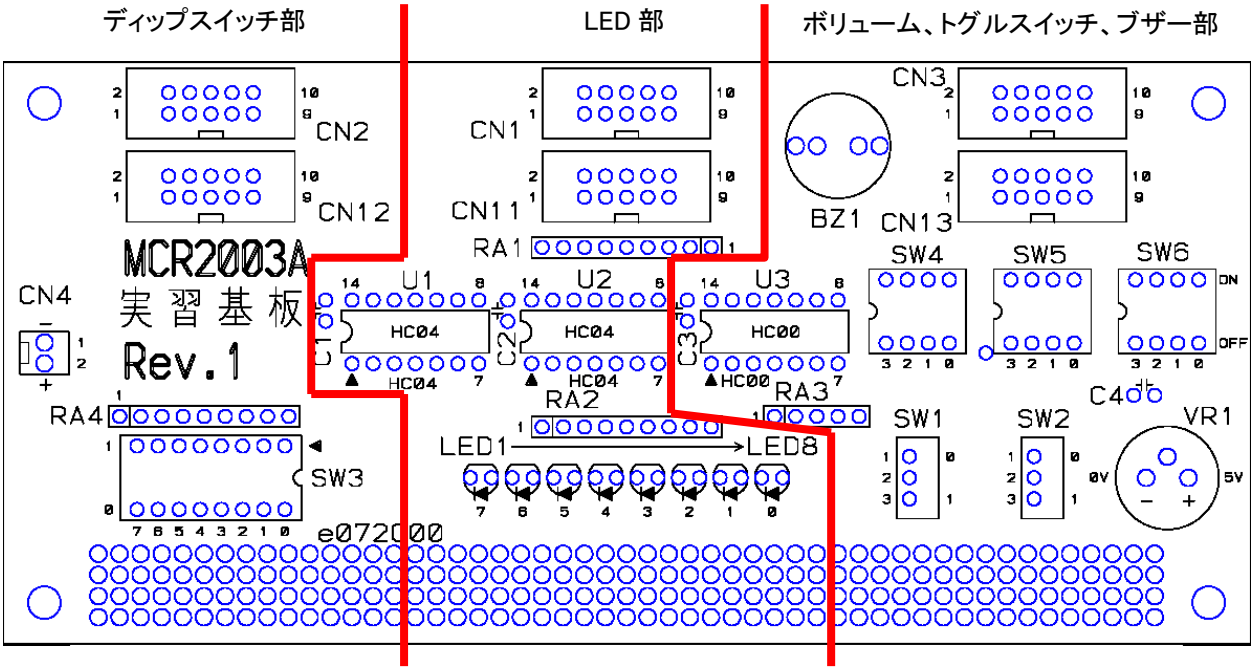
本実習マニュアルは、主に MCR2003A 実習基板を使って H8/3048F-ONE の機能について実習していきます。



▲MCR2003A 実習基板

マニュアル内では「実習基板」として説明していきます。

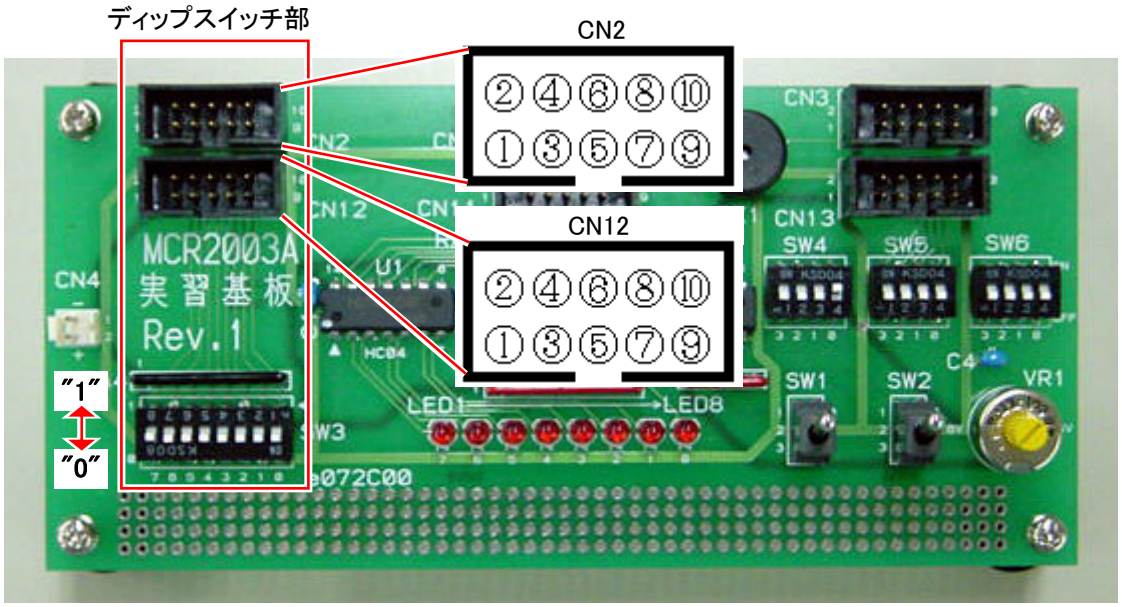
2.2 外観



2.3 ディップスイッチ部

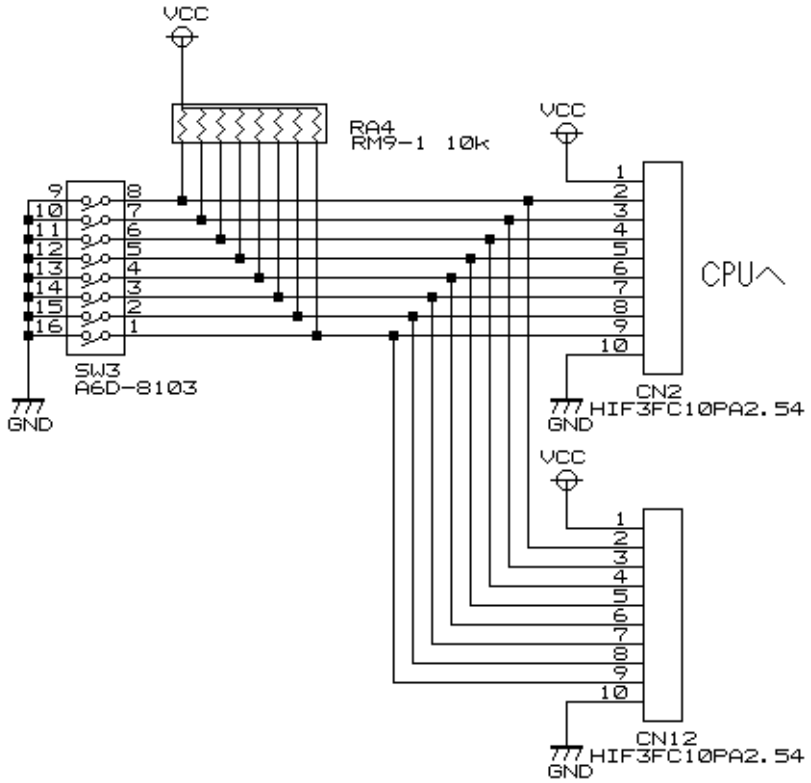
2.3.1 コネクタ

ディップスイッチの信号 8 ビット分を、CN2(CN12)コネクタから外部へ出力します。CN2、CN12 は並列に接続されています。

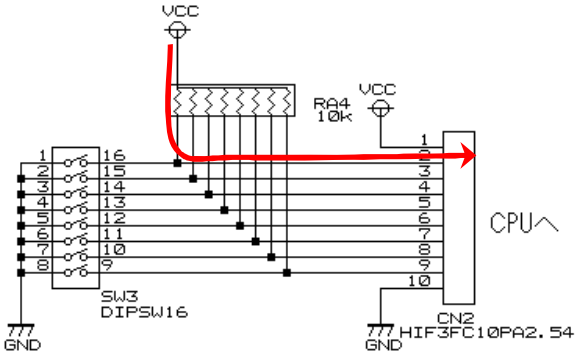


CN2,CN12 ピン番号	信号	“0”	“1”
1	＋5V	－	－
2	ディップスイッチ 7	下側	上側
3	ディップスイッチ 6	下側	上側
4	ディップスイッチ 5	下側	上側
5	ディップスイッチ 4	下側	上側
6	ディップスイッチ 3	下側	上側
7	ディップスイッチ 2	下側	上側
8	ディップスイッチ 1	下側	上側
9	ディップスイッチ 0	下側	上側
10	GND	－	－

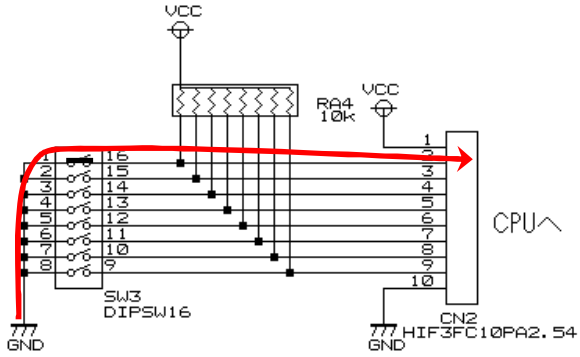
2.3.2 回路



2.3.3 回路説明



スイッチが OFF なら、プルアップ抵抗を通して、"1"が出力されます。

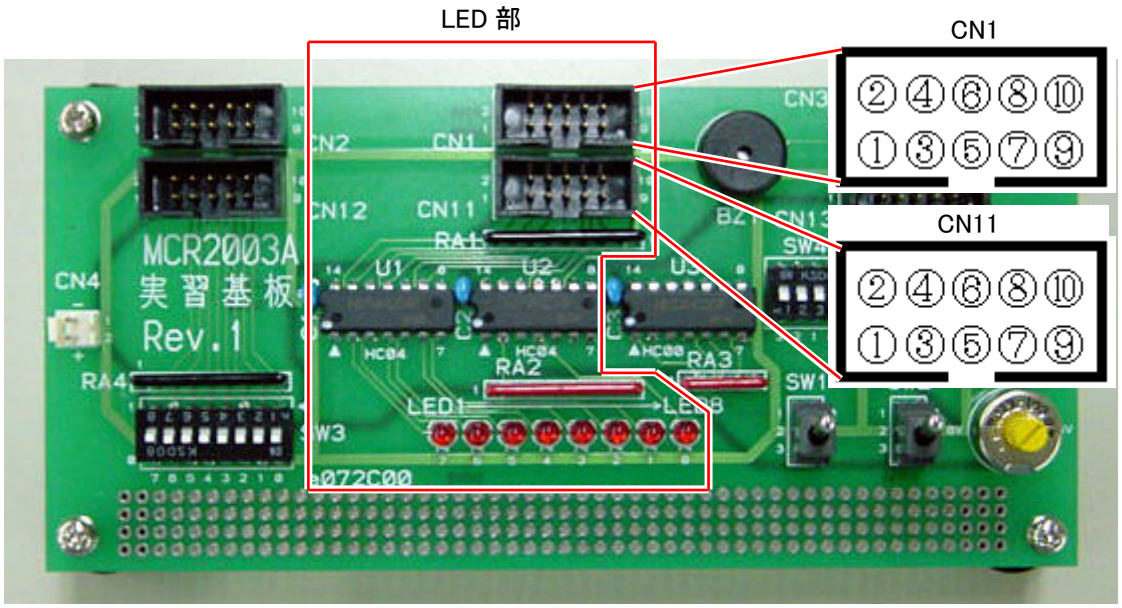


スイッチが ON なら、GND と直結になり、"0"が出力されます。

2.4 LED部

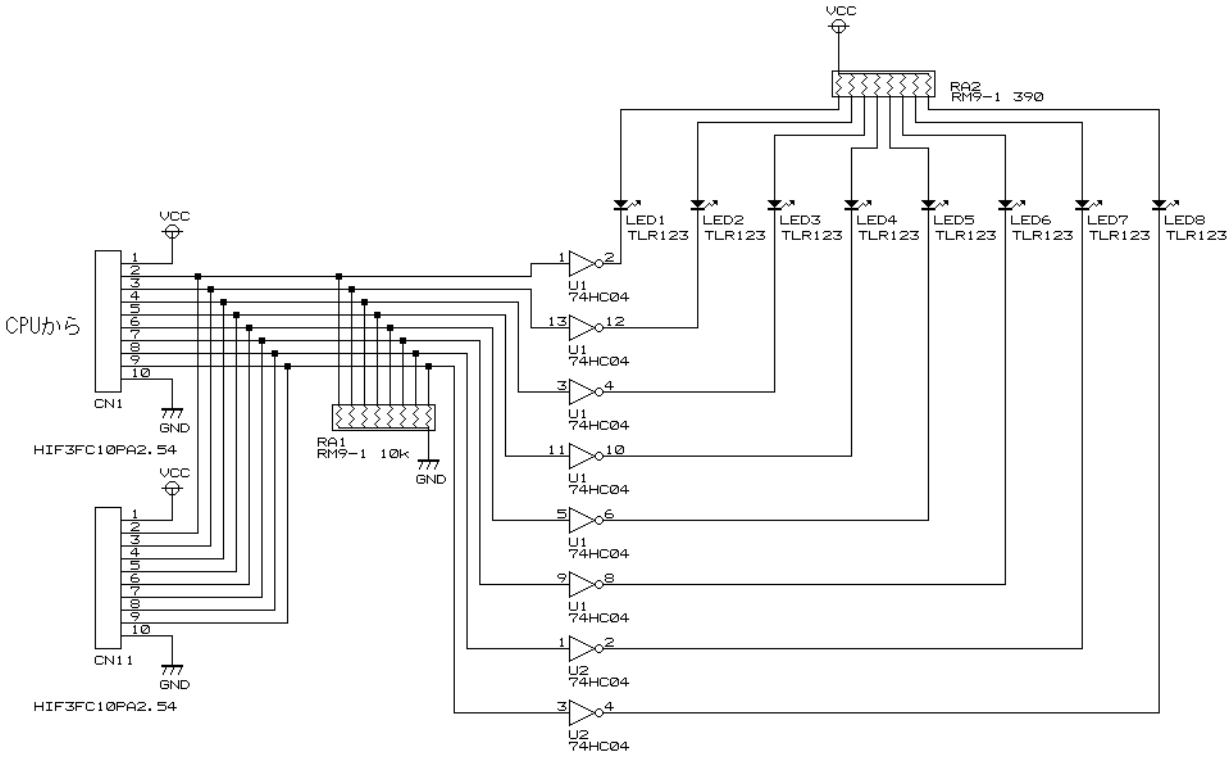
2.4.1 コネクタ

CN1(CN11)コネクタから入力した信号 8 ビット分を、LED に表示します。CN1、CN11 は並列に接続されています。

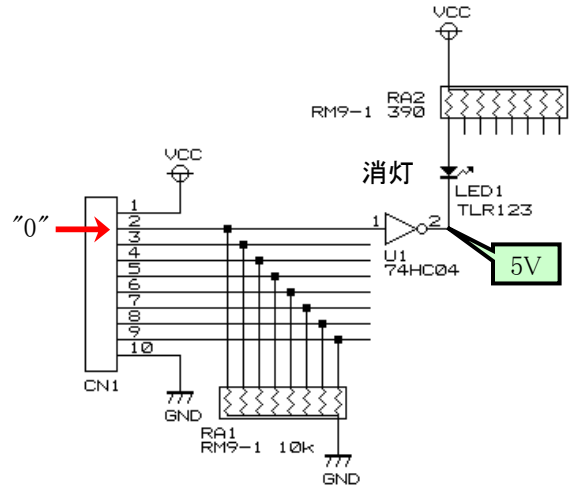


CN1,CN11 ピン番号	信号	“0”	“1”
1	＋5V	－	－
2	LED1(7)	消灯	点灯
3	LED2(6)	消灯	点灯
4	LED3(5)	消灯	点灯
5	LED4(4)	消灯	点灯
6	LED5(3)	消灯	点灯
7	LED6(2)	消灯	点灯
8	LED7(1)	消灯	点灯
9	LED8(0)	消灯	点灯
10	GND	－	－

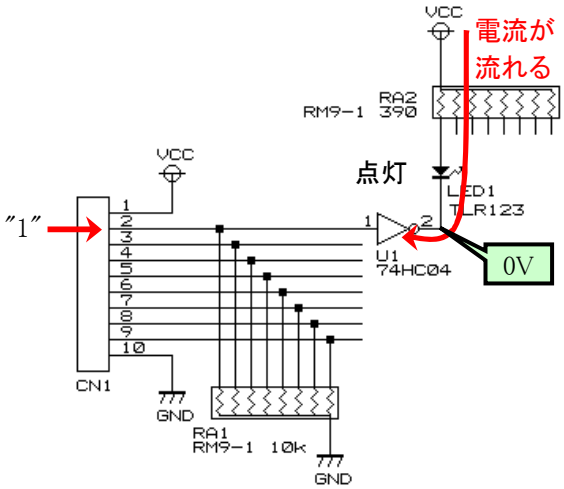
2.4.2 回路



2.4.3 回路説明



入力信号が"0"(0V)なら、LED は消灯です。

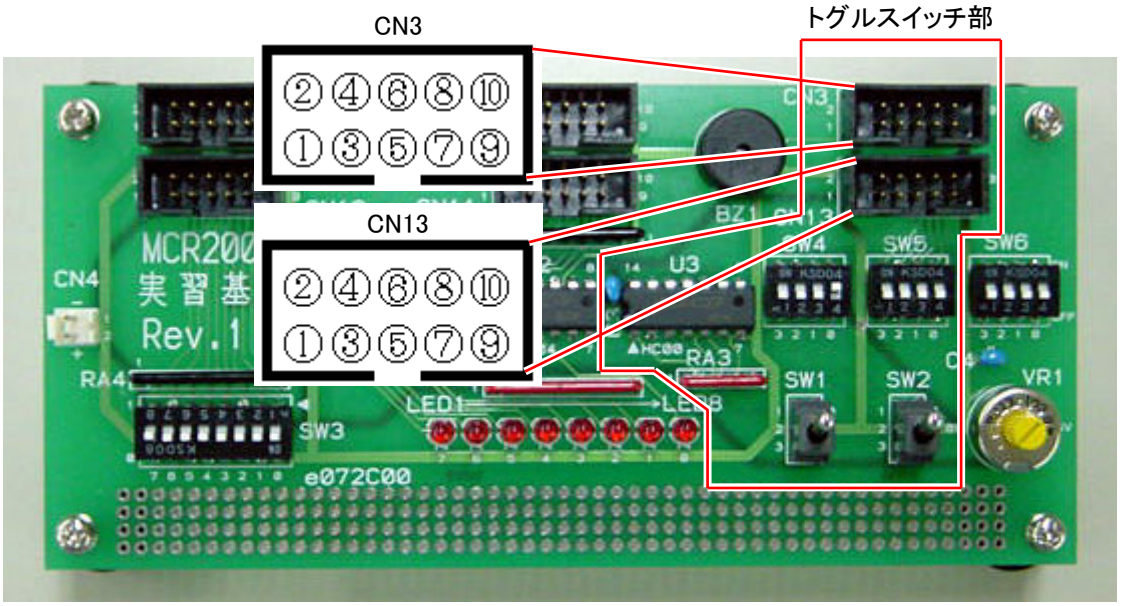


入力信号が"1"(5V)なら、LED は点灯します。

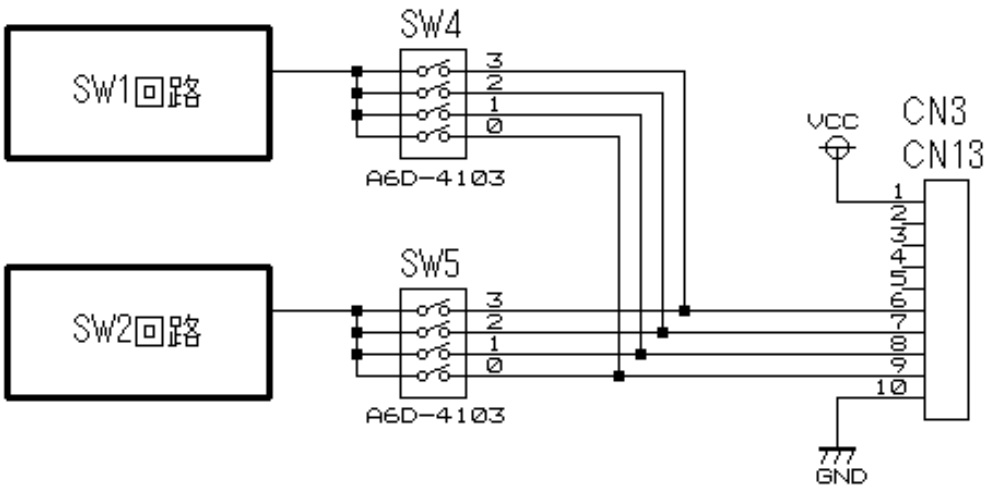
2.5 トグルスイッチ部

2.5.1 コネクタ

SW1,SW2 の信号を、CN3(CN13)コネクタへ出力します。

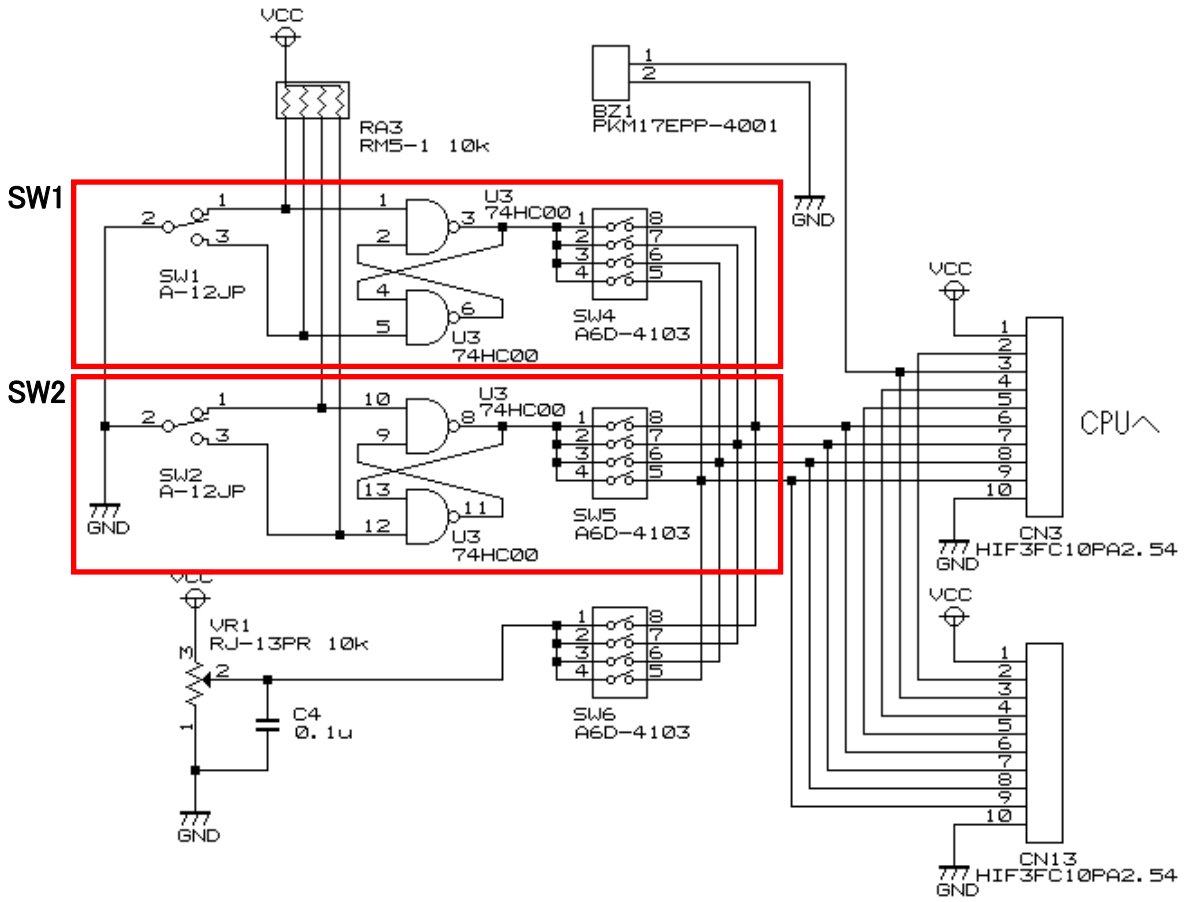


CN3(CN13)のどのピンへ出力するかは、SW1 の信号は SW4 で、SW2 の信号は SW5 で切り替えます。



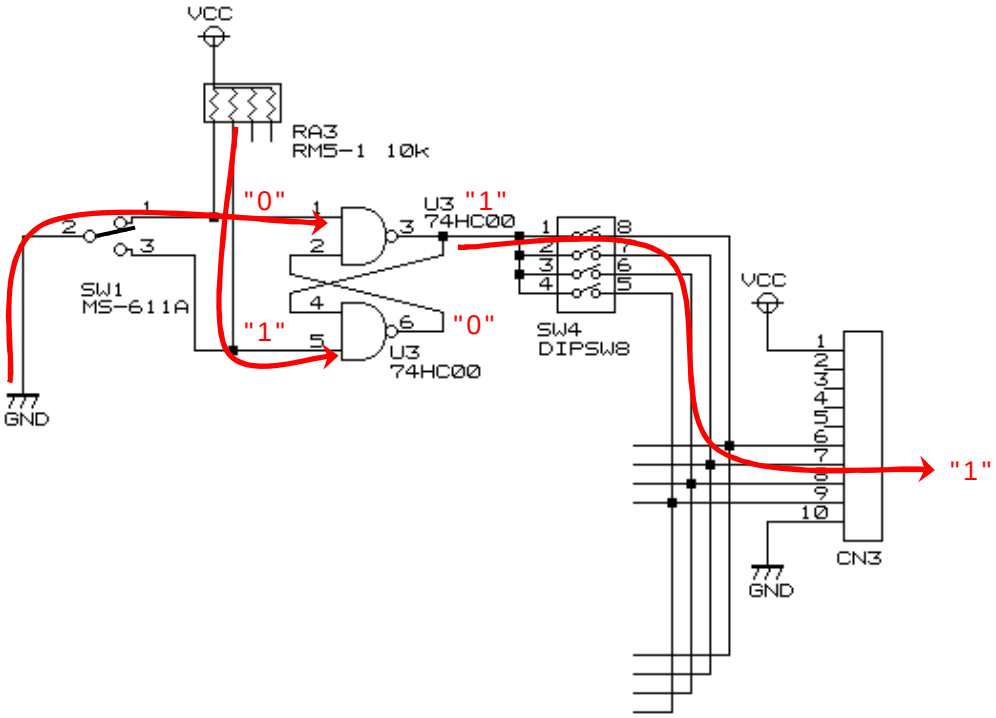
2.5.2 回路

□で囲った部分が、SW1 の回路と、SW2 の回路です。



2.5.3 回路説明

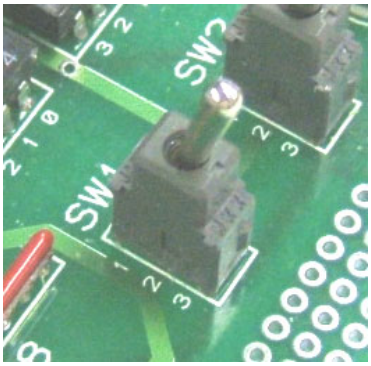
トグルスイッチは下記のような動作原理です。

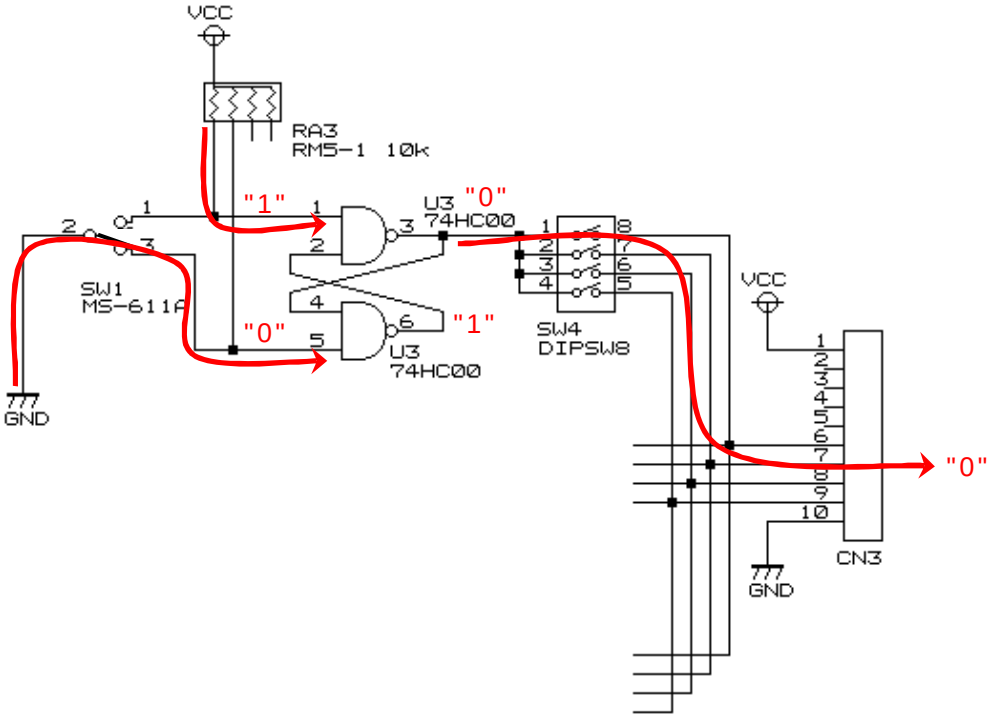


SW1 の 1－2 ピンを短絡させたとき、U3 の端子状態は、

1. 1ピンには、スイッチが繋がったことによりGNDの”0”が入力されます。
2. 5ピンには、スイッチが開放されたことによりプルアップ抵抗の”1”が入力されます。
3. 3ピン出力は、1ピンが”0”であることで2ピンが”0”であろうと、”1”であろうと”1”が出力されます。
4. 6ピンには、4ピン”1”、5ピン”1”が入力され、$\overline{1 \cdot 1} = \overline{1} = 0$でとなります。
5. 2ピンは”0”となりますが、3ピン出力は変わりません。
6. 結果、出力は”1”となります。

ちなみに、1-2 ピンを短絡させた状態は、SW1 を 3 ピン側に倒した状態です(下写真)。倒した側と反対側が短絡するので、注意してください。

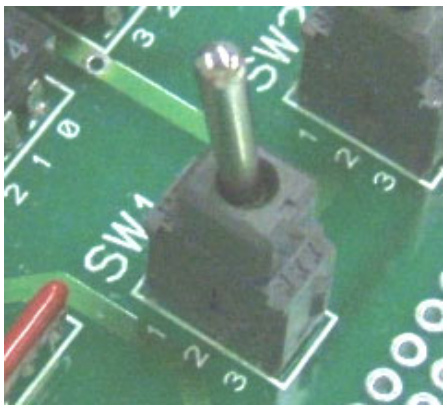




SW1 のスイッチ 2－3 ピンを短絡させたとき、U3 の端子状態は、

1. 5ピンには、スイッチが繋がったことによりGNDの”0”が入力されます。
2. 1ピンには、開放されたことによりプルアップ抵抗の”1”が入力されます。
3. 6ピン出力は、5ピンが”0”であることで4ピンが”0”であろうと、”1”であろうと”1”が出力されます。
4. 3ピンには、1ピン”1”、2ピン”1”が入力され、$\overline{1 \cdot 1} = \overline{1} = 0$でとなります。
5. 結果、出力は”0”となります。

ちなみに、2-3 ピンを短絡させた状態は、SW1 を 1 ピン側に倒した状態です(下写真)。倒した側と反対側が短絡するので、注意してください。



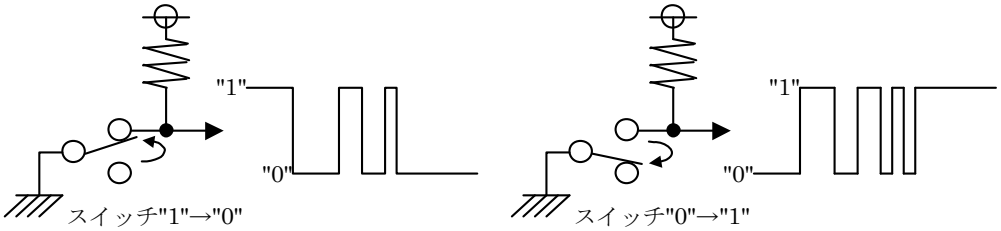
このように、スイッチを上下させることにより、出力が″0″、または″1″と変化します。なぜ単純なディップスイッチ部の様な回路ではいけないのでしょうか。理由は、次ページの参考資料を参照してください。

※参考資料ーチャタリング防止回路

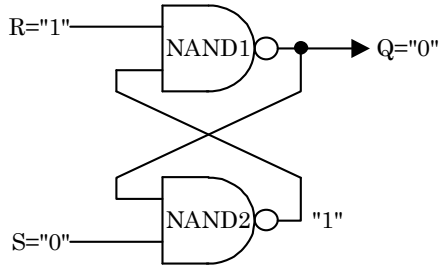
下図のように、通常のプルアップしたスイッチを"1"から"0"、もしくは"0"から"1"に変更した場合、何度か接点が付いたり離れたりして、最終的に"1"や"0"になります。これを「チャタリング」といいます。

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、誤カウントとなってしまいます。

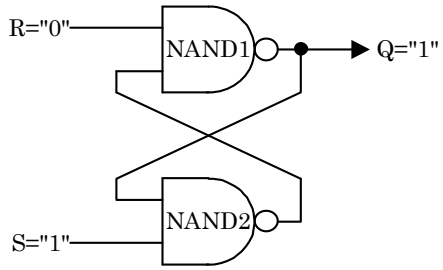
下図の"1"→"0"の例では、一度しか"1"→"0"にしていないつもりでも 3 回もスイッチを上げ下げしたと判断されてしまいます。更にやっかいなことは、発生するパルス数、収束するまでの時間が毎回違うということです。



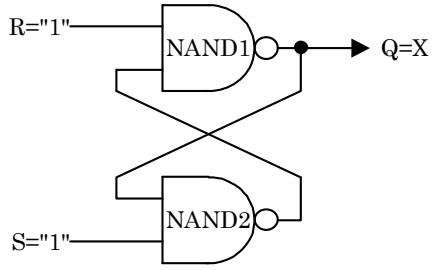
チャタリングを解消する回路の一つに「リセット・セット・フリップフロップ(RS-FF)」という回路があります。NAND 回路をループさせた形の回路です。R(Reset)端子に"1"が入力されると出力 Q は"0"になり、S(Set)端子に"1"が入力されると出力 Q は"1"になることからそう呼ばれています。動作原理は次のとおりです。



●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。



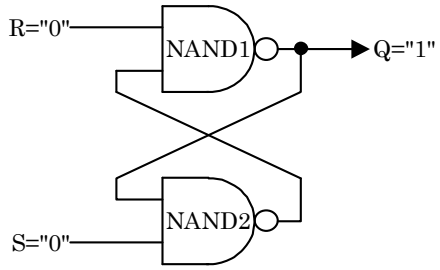
●入力(R,S)=("0","1")のとき
・NAND1 は R="0"のため、もう一方の入力が何であろうと無条件で出力"1"
・NAND2 は S="1"、もう一方の入力は NAND1 の出力"1"なので出力"0"
・NAND1 の R 側でない入力が"0"で確定するが R="0"のため、出力は変わらず"1"
結果、セット信号だけが"1"のため、出力 Q="1"となったと考えることができます。



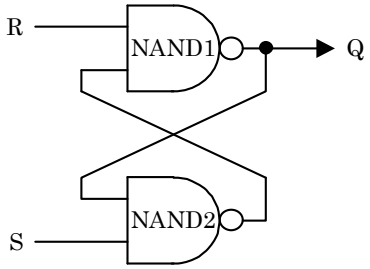
●入力(R,S)=("1","1")のとき
・NAND1、NAND2、共に入力が決まらなければ出力は決定されないので、NAND1 の出力 Q=X と仮定する
・NAND2 の入力は S="1"と X なので、出力は $\overline{X}$ となる
・NAND1 の入力は R="1"と $\overline{X}$ なので

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

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。



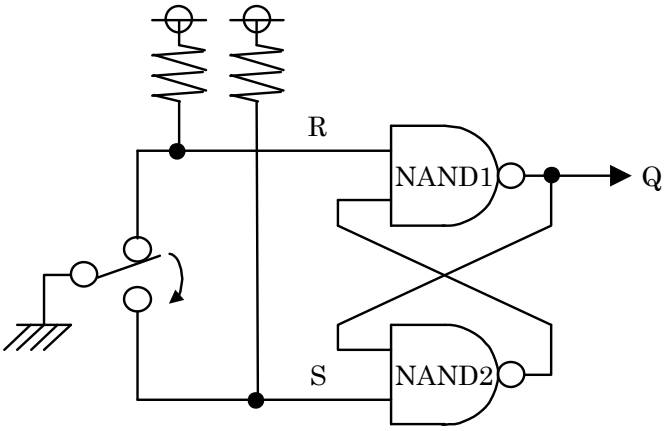
●入力(R,S)=("0","0")のとき
・NAND1 は R="0"、もう一方の入力が何であろうと無条件で"1"
・NAND2 は S="0"、もう一方の入力が何であろうと無条件で"1"
結果、Q="1"となります。
ただし、(R,S)=("0","0")はリセットともセットとも呼べない状態です。Q="1"となりセット状態ではありますが、リセット、セット、どちらかに"1"信号があるものとする論理に反する入力状態のため、通常は「禁止」とされています。ショートなどして回路が壊れるために禁止とするのではなく、論理が合わなくなるため禁止としています。

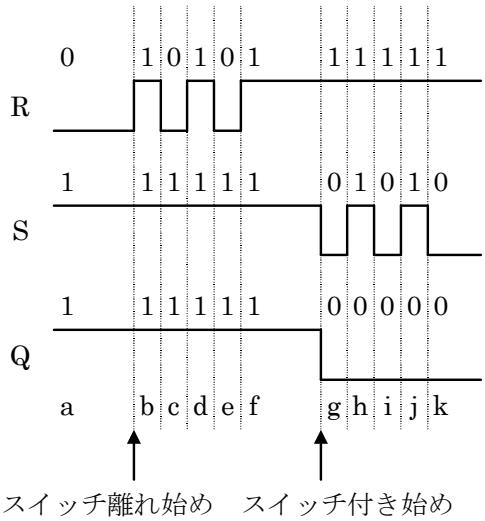


まとめると下表のようになります。

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。



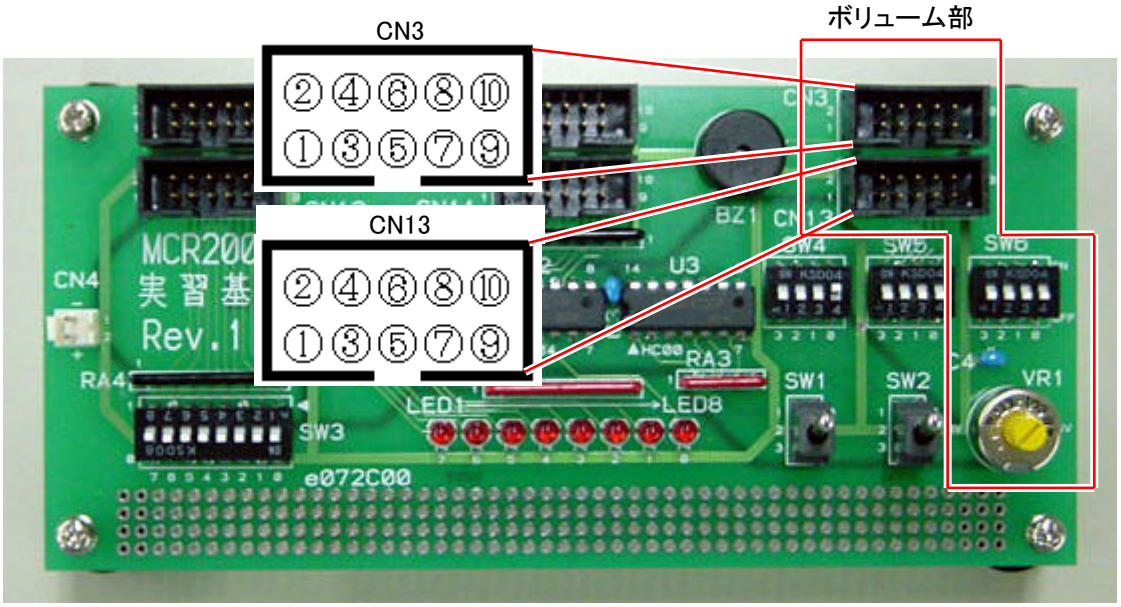


a. 最初、(R,S)=("0","1")のため、Q="1"となります。
b. スイッチを下側に切り替え、R 側から接点が離れ始めた時です。(R,S)=("1","1")のため、Q は前の値を保持します。
c. (R,S)=("0","1")のため、Q="1"となります。
d. (R,S)=("1","1")のため、Q は前の値を保持します。
e. (R,S)=("0","1")のため、Q="1"となります。
f. (R,S)=("1","1")のため、Q は前の値を保持します。接点が中間になりました。R 端子、S 端子共にスイッチの接点には繋がっていない状態ですが、プルアップ抵抗があるので"1"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

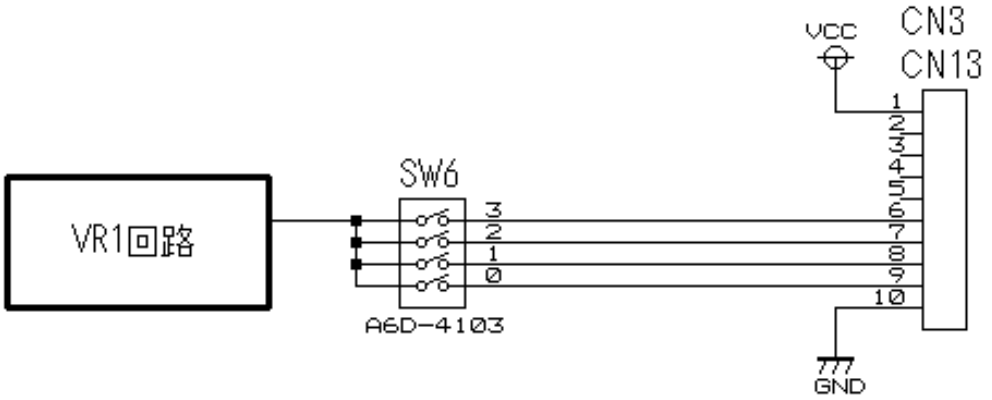
2.6 ボリューム部

2.6.1 コネクタ

VR1 の信号を、CN3(CN13)コネクタへ出力します。

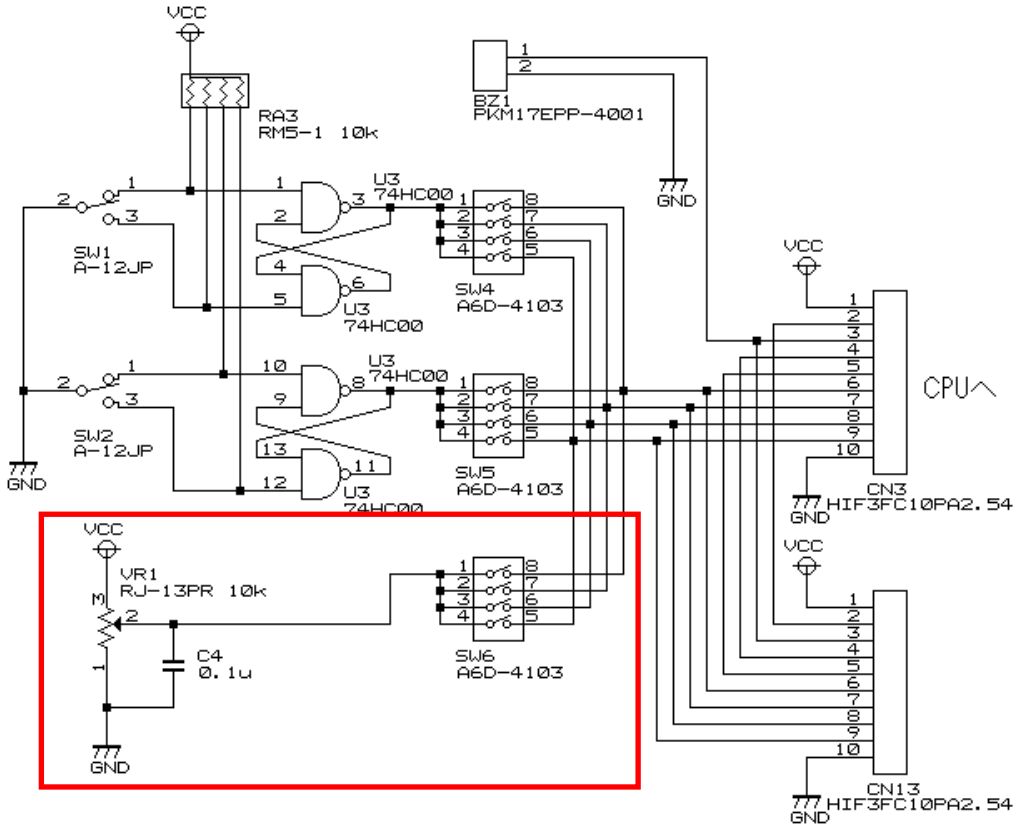


CN3(CN13)のどのピンへ出力するかは、SW6 で切り替えます。

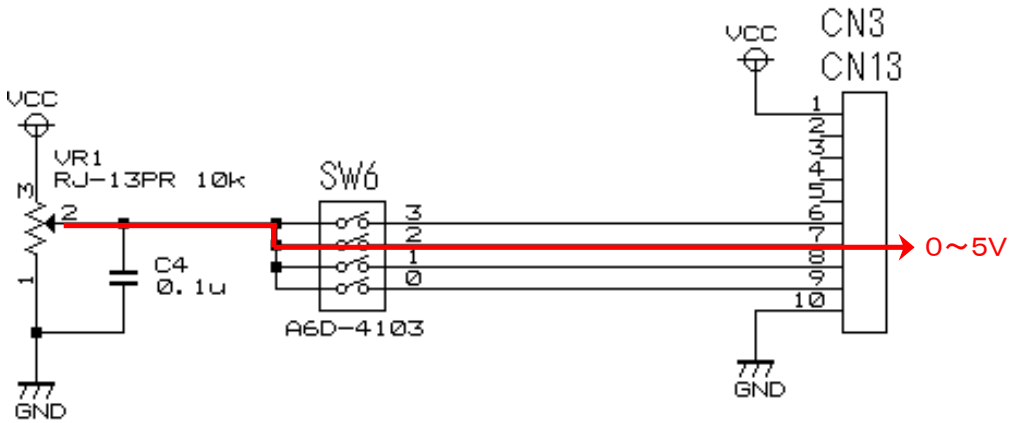


2.6.2 回路

□で囲った部分が、ボリューム部の回路です。



2.6.3 回路説明

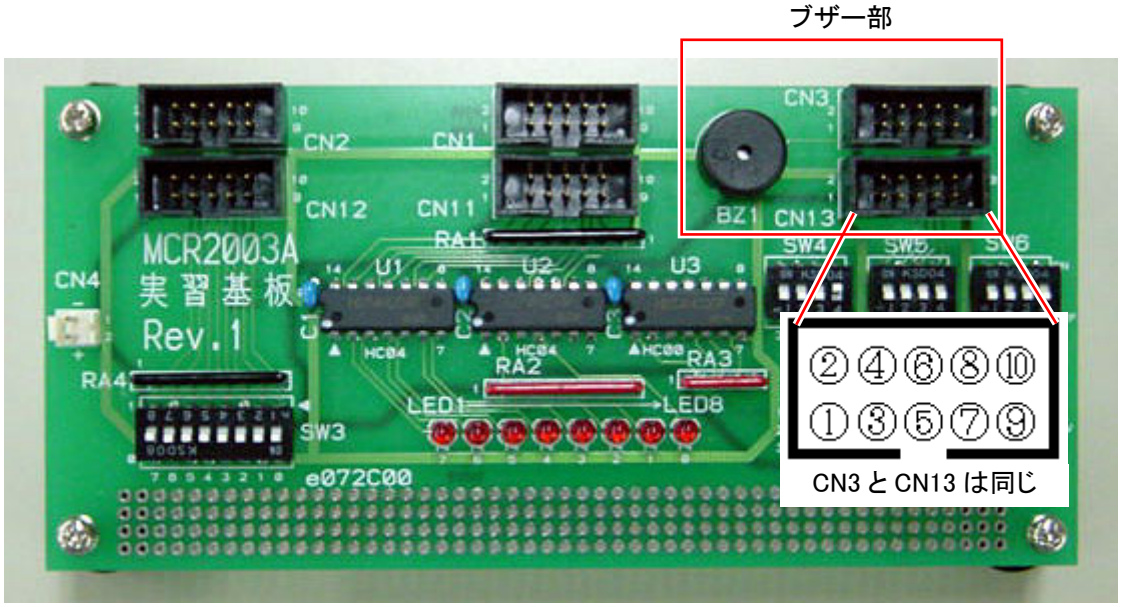


太いラインは、SW6 の 2 を ON にしたときの流れです。ボリュームの電圧が、SW6 を通して CN3 へ出力されます。C4 は、信号にノイズが乗ってアナログ変換値がばらつくのを防ぐ役割です。

2.7 ブザー部

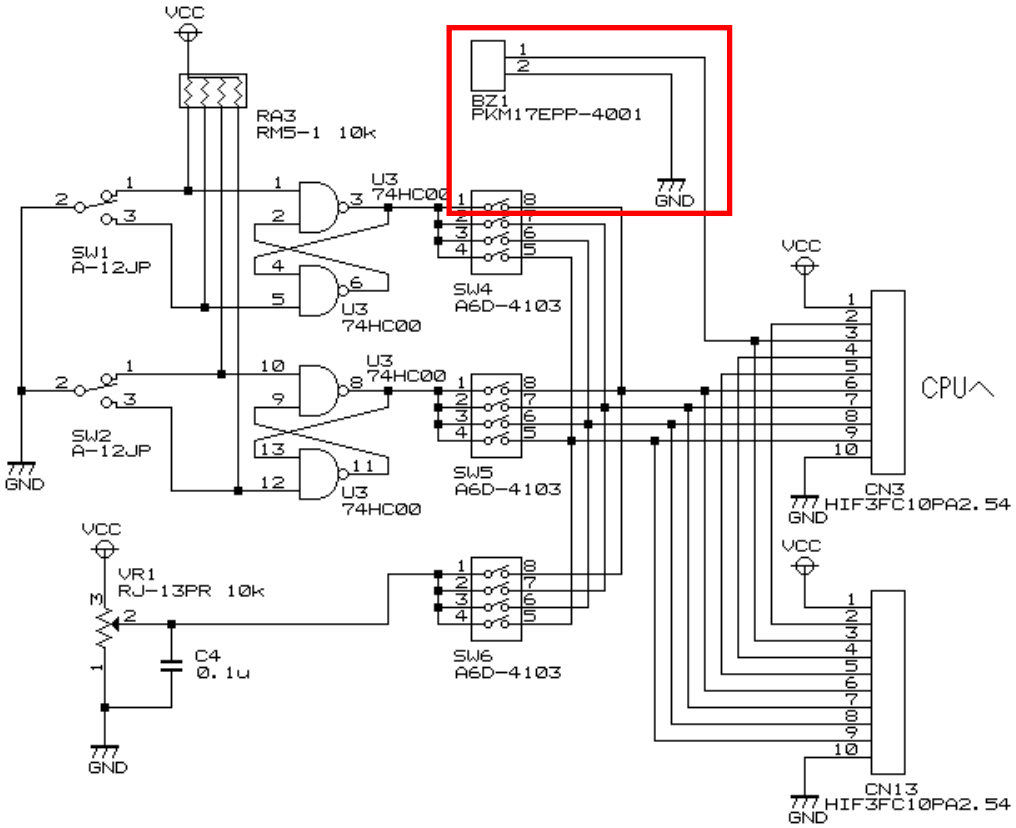
2.7.1 コネクタ

CN3(CN13)コネクタから入力された信号でブザーを鳴らします。



2.7.2 回路

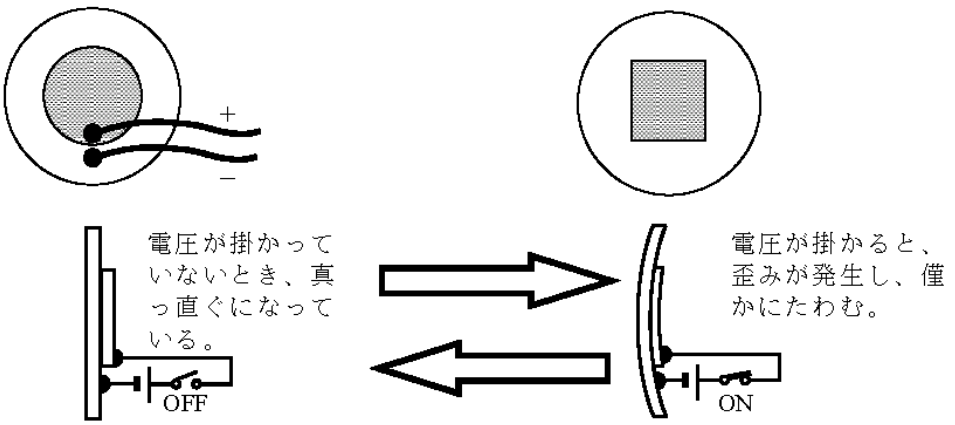
□で囲った部分が、ブザー部の回路です。ブザーは、CN3(CN13)の 3 ピンに接続されています。



2.7.3 ブザーの動作原理

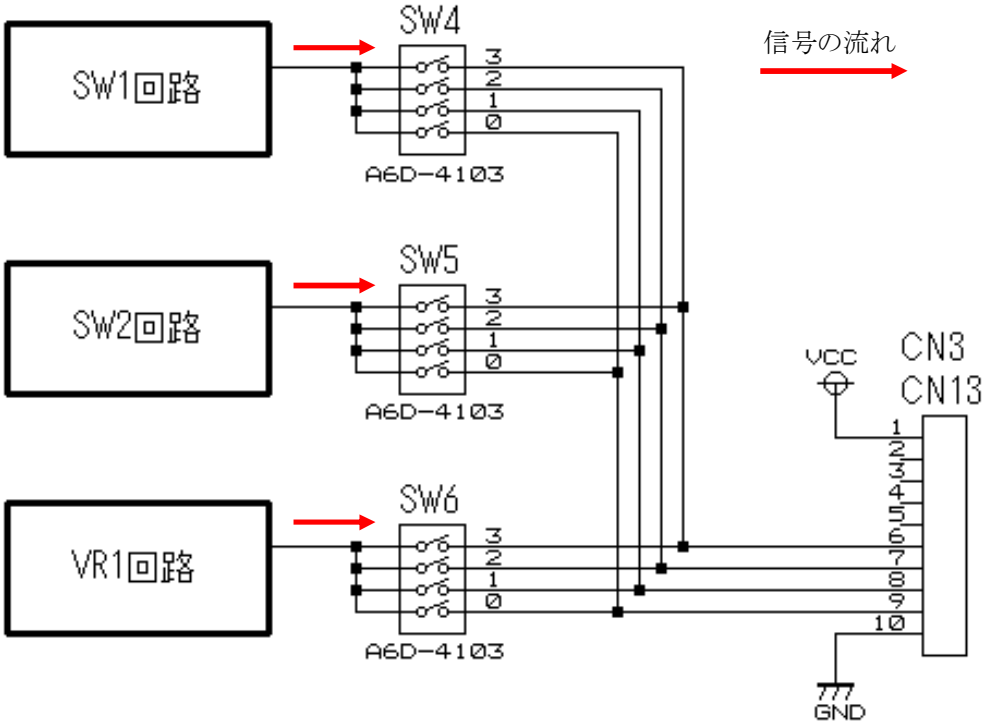
ブザーは、下図のように 2 枚の電極を貼り合わせたような形をしています。2 枚の電極は絶縁されていますが、既定値以上の電圧を加えると壊れてしまいます。外側は薄い鉄板になっていて、内側には特殊な電極が張り付いています。両方とも半田付けができるようになっています。

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。

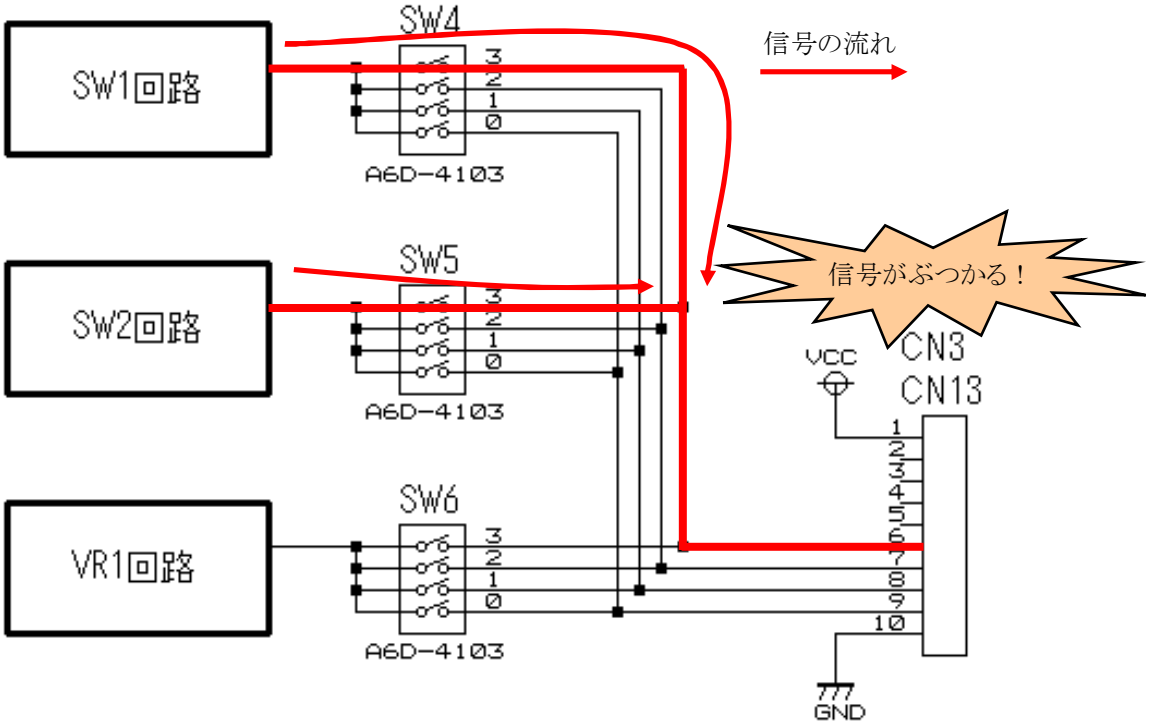


2.8 SW4,SW5,SW6 を切り替えるときの注意点

SW1 回路、SW2 回路、VR1 回路と SW4～6 の関係は下図のようになっています。

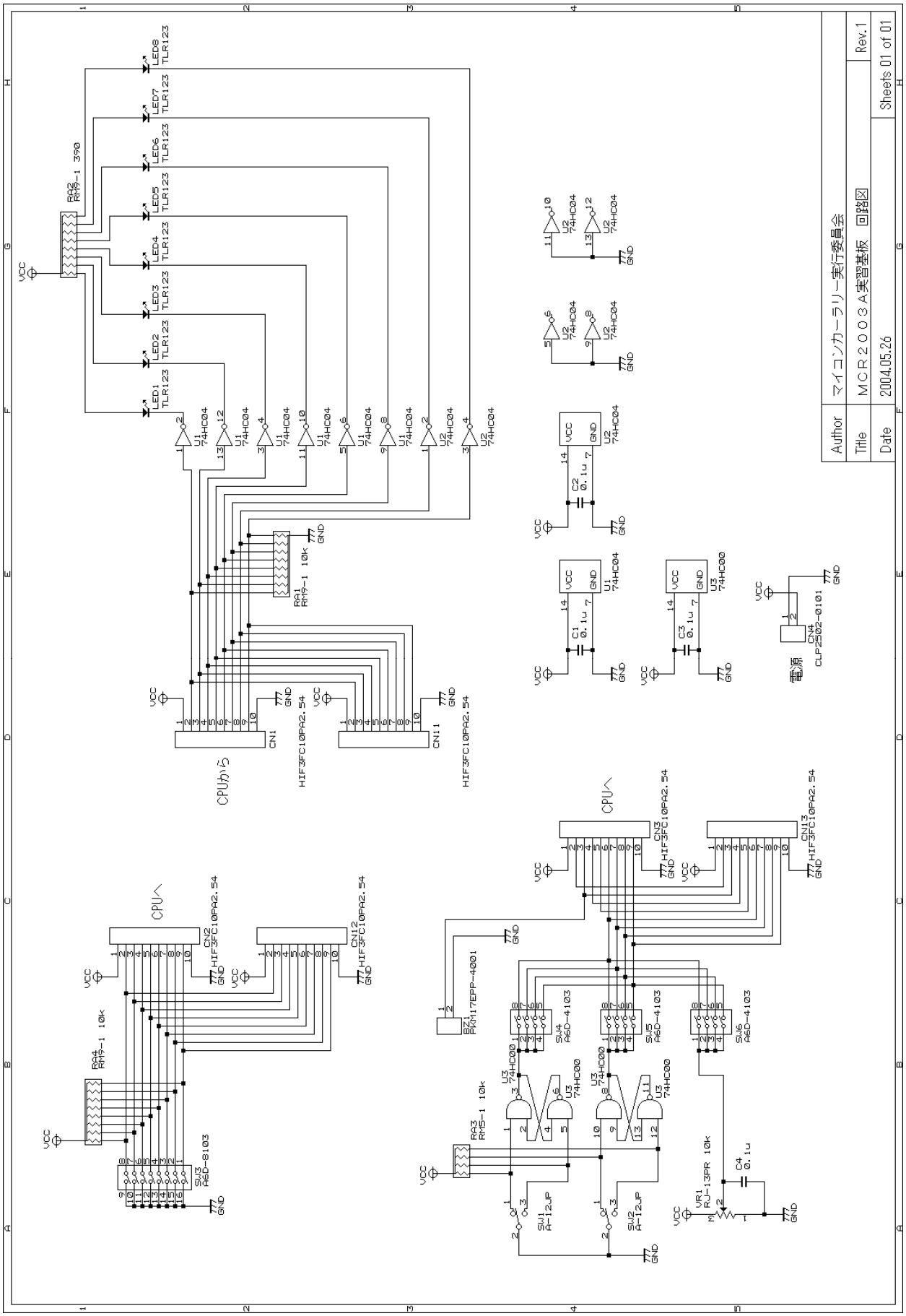


例えば、SW4 の 3 と SW5 の 3 を同時に ON にすると、下図のように信号同士がぶつかり、ショート状態になることがあります。



そのため、**SW4,SW5,SW6 の 3～0 のスイッチを同時に ON にしないようにしてください**。SW4,SW5,SW6 を切り替えるときは、すべてを OFF にしてから、該当のスイッチを ON にしてください。

2.9 回路図


Author
マイコンカーラリー実行委員会
Title
MCR2003A実習基板 回路図
Rev.1
Date
2004.05.26
Sheets 01 of 01
CPUから
CPUへ
CPUへ
電源

3. サンプルプログラム

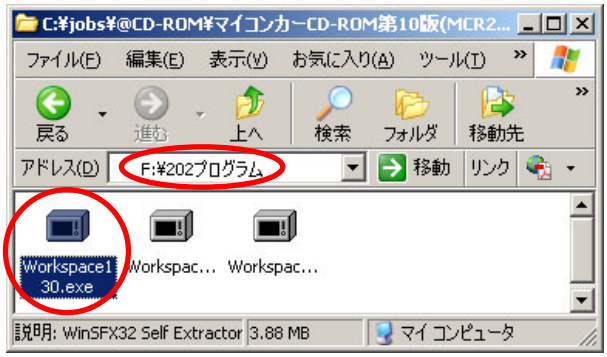
3.1 ルネサス統合開発環境

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

3.2 サンプルプログラムのインストール

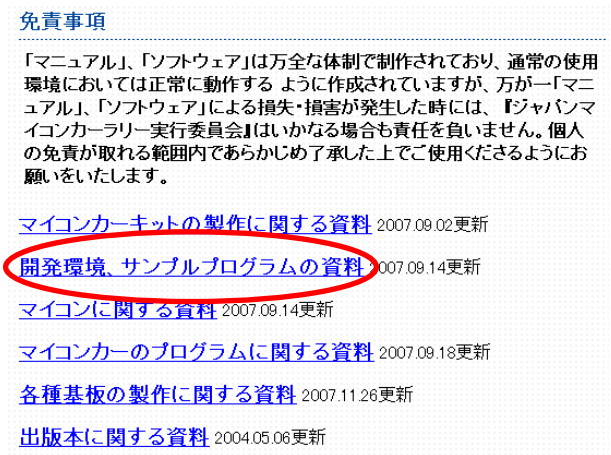
サンプルプログラムをインストールします。

3.2.1 CDからソフトを取得する



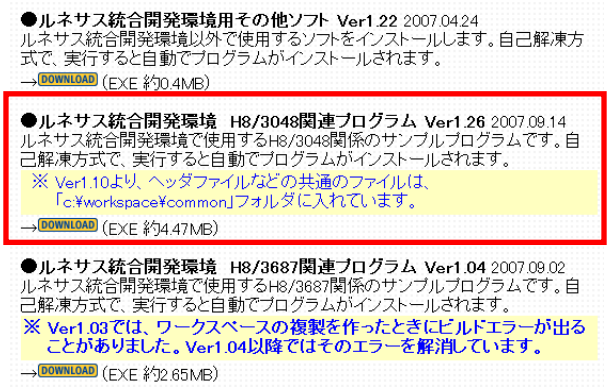
2007 年以降の講習会 CD がある場合、「CD ドライブ→202 プログラム」フォルダにある、「Workspace130.exe」を実行します。数字の 130 は、バージョンにより異なります。

3.2.2 ホームページからソフトを取得する



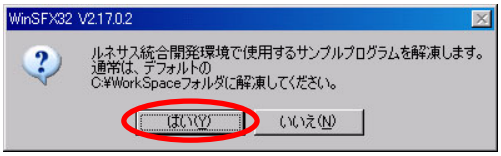
1.マイコンカーラリーサイト「http://www.mcr.gr.jp/」の技術情報→ダウンロード内のページへ行きます。

2.「開発環境、サンプルプログラムの資料」をクリックします。

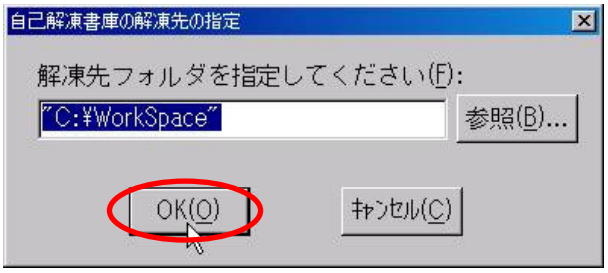


3.「ルネサス統合開発環境　H8/3048 関連プログラム」をダウンロードします。

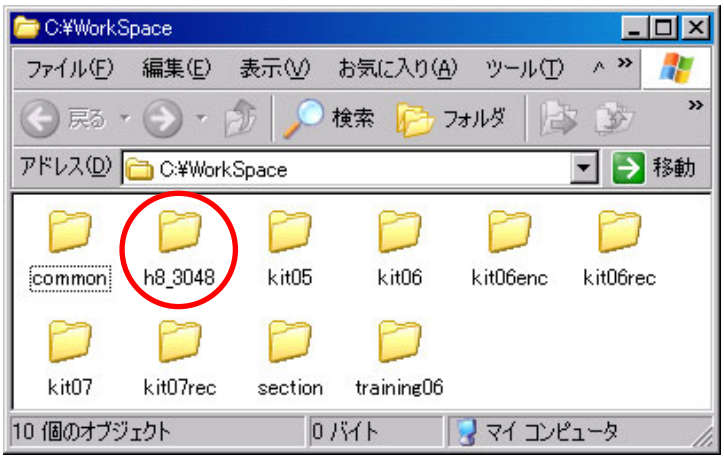
3.2.3 インストール



1.CD またはダウンロードした「Workspace130.exe」を実行します。はいをクリックします。



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



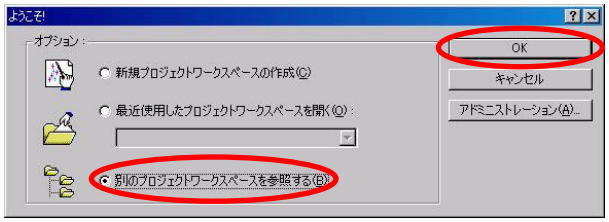
3. 解凍が終わると、「C ドライブ→Workspace」フォルダが自動で開かれます。複数のフォルダがあります。今回使用するのは、「**h8_3048**」です。

4. 演習手順

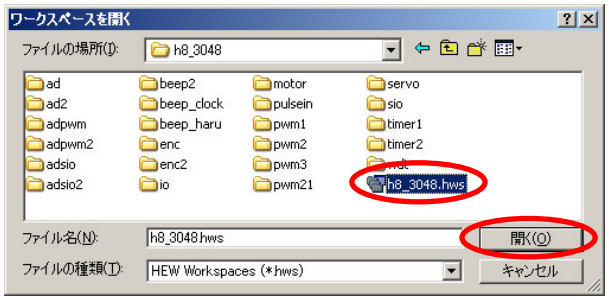
これから、実際にサンプルプログラムを使って、演習をしていきます。その前に、操作手順を説明していきます。詳しくは、「ルネサス統合開発環境　操作マニュアル」を参照してください。

4.1 ワークスペースを開く

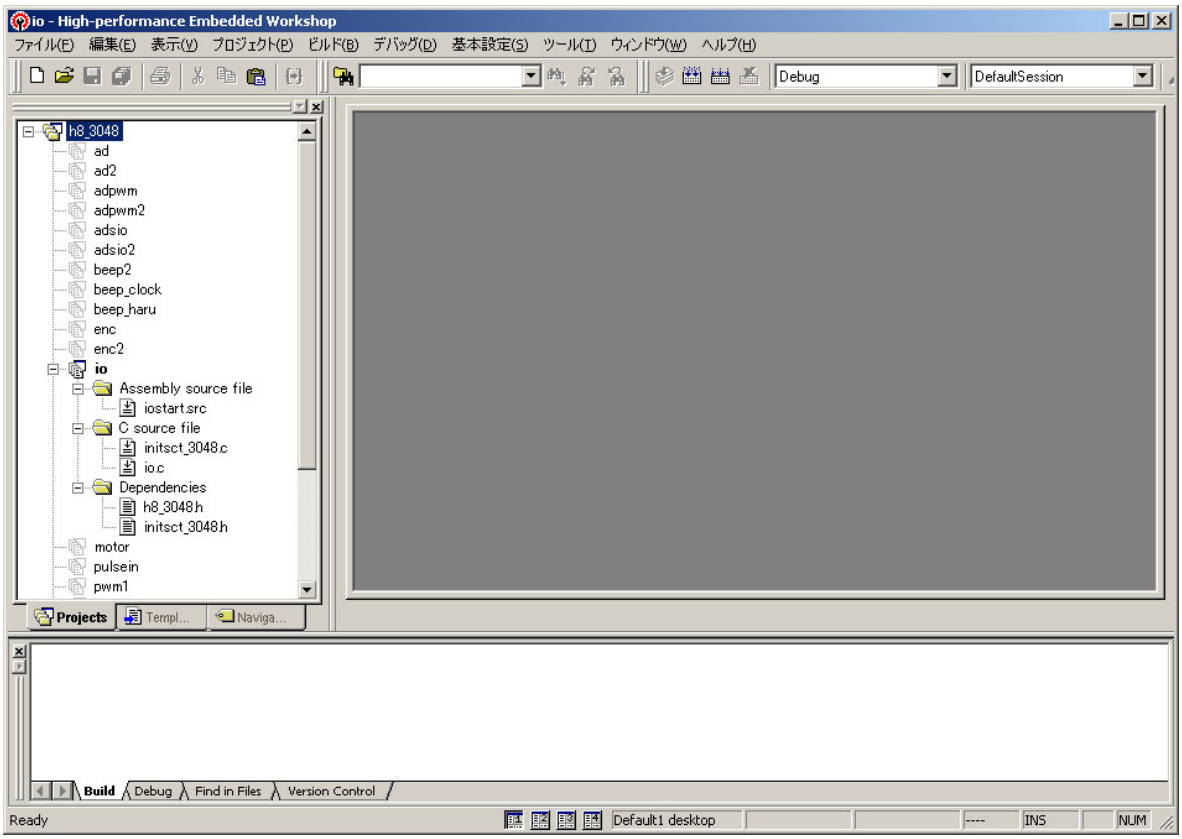
1.ルネサス統合開発環境を実行します。



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。



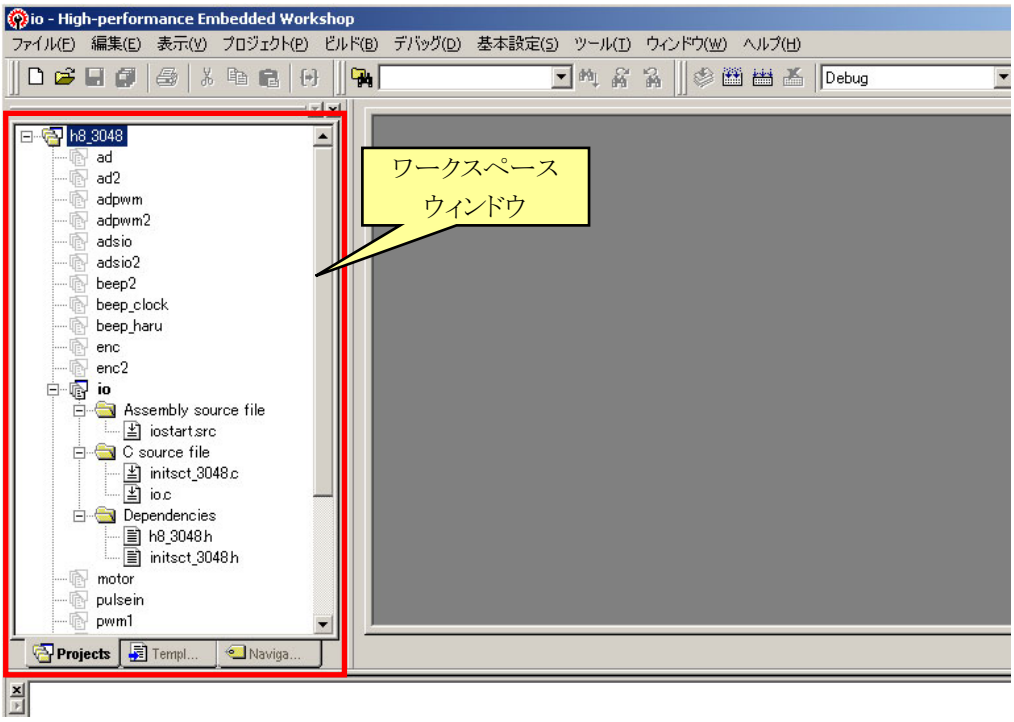
3.「Cドライブ→workspace→h8_3048」の「h8_3048.hws」を選択、開くをクリックします。



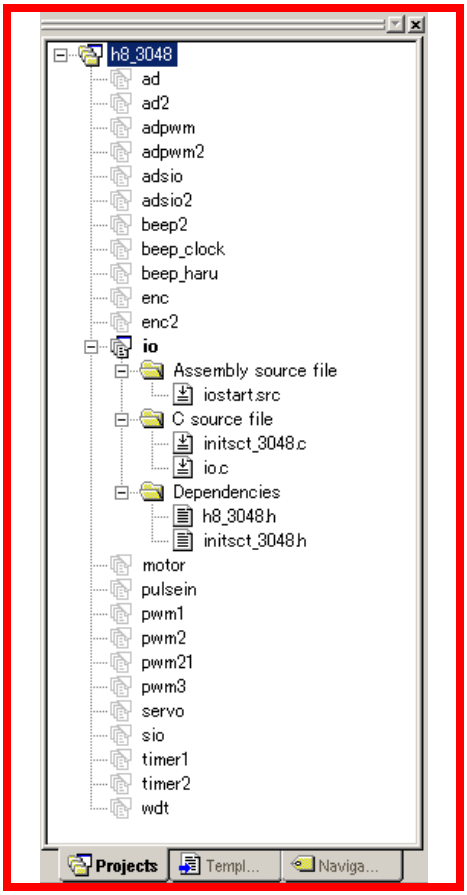
4.h8_3048 というワークスペースが開かれました。本実習マニュアルは、このワークスペースを使用します。

4.2 プロジェクトを開く

例えば、これから「sio」の演習をするとします。

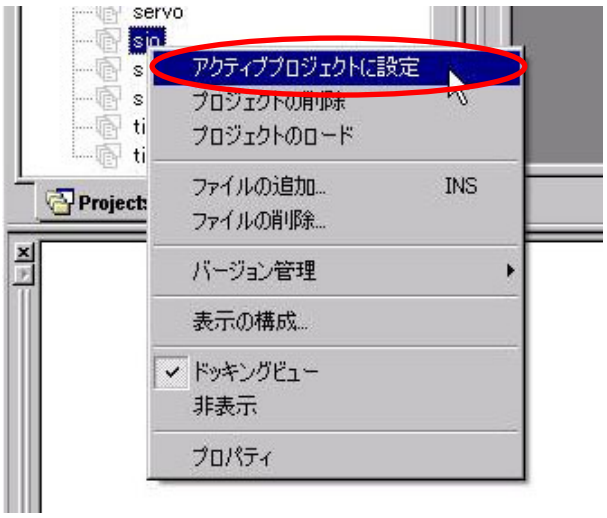


1.ワークスペースウィンドウと呼ばれるスペースにたくさんのプロジェクトが登録されています。

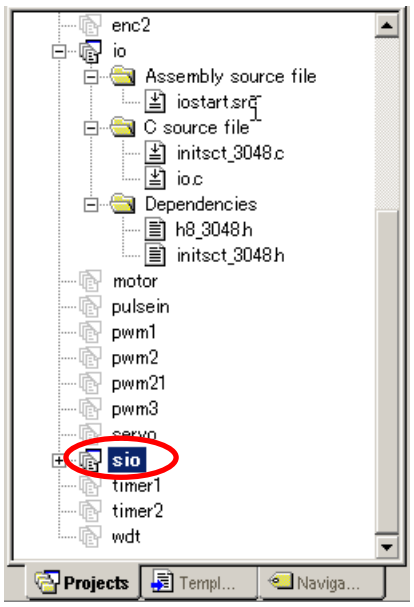


2.上から、
ad , ad2 , adpwm , adpwm2 , adsio , adsio2 , beep2 ,
beep_clock , beep_haru , enc , enc2 , io , motor , pulsein ,
pwm1 , pwm2 , pwm21 , pwm3 , servo , sio , timer1 , timer2 , wdt
というプロジェクトがあります。
プロジェクト＝演習の単位となります。

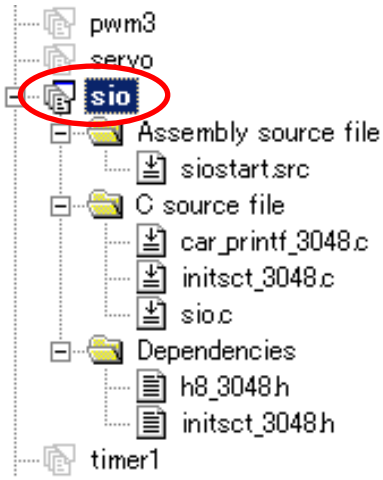
左画面では、プロジェクト「io」が太字になっています。このプロジェクトが現在有効なプロジェクト(アクティブプロジェクト)です。有効なプロジェクトは、
・ビルドの対象
・ツールチェイン(各種設定)の対象
・書き込みの対象
となります。「sio」の演習を行うので、「sio」を有効なプロジェクトにしなければいけません。



3.プロジェクト「sio」上で右クリック、「アクティブプロジェクトに設定」を選択します。



4.「sio」が太字になりました。これでアクティブプロジェクト設定完了です。

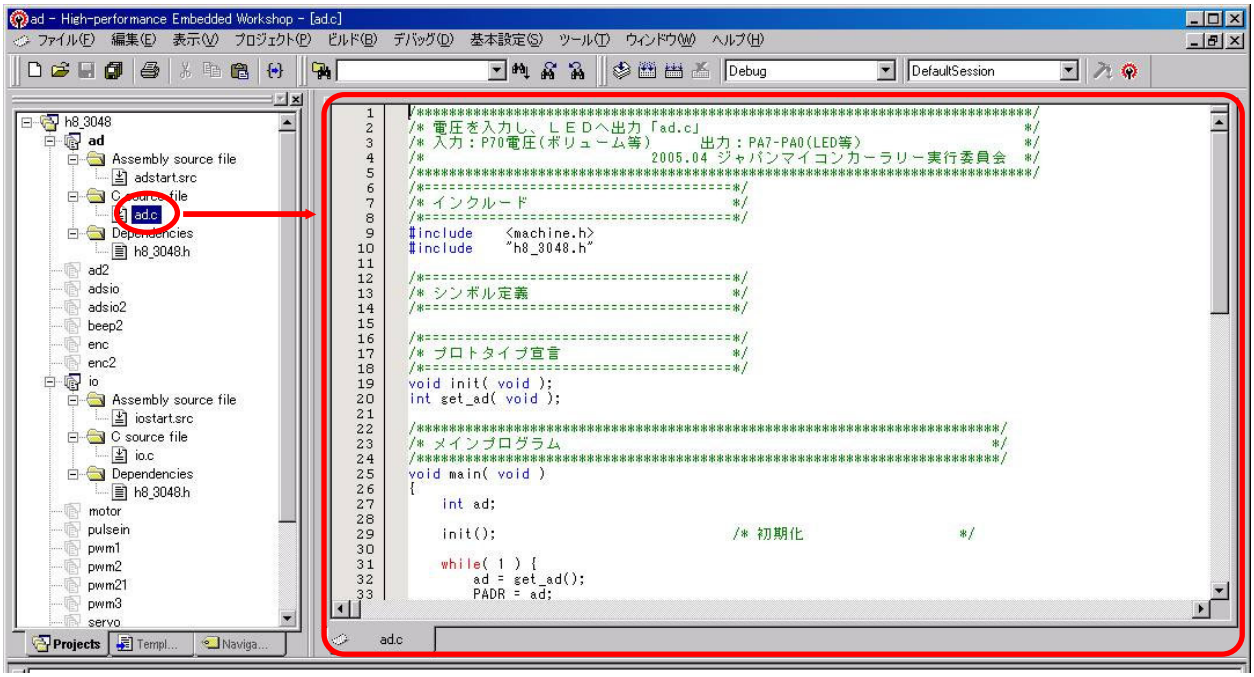


5.「sio」をダブルクリックします。
これがプロジェクト「sio」に登録されているファイルです。
- siostart.src
- car_printf_3048.c
- initsct_3048.c
- sio.c

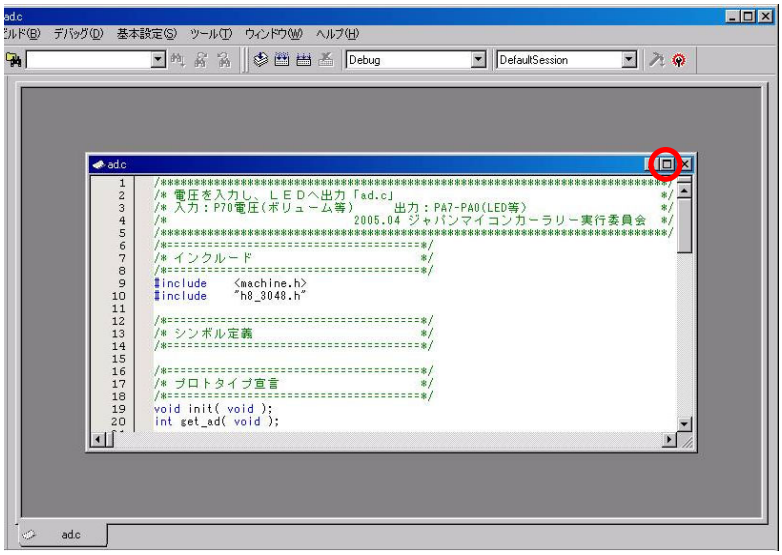
という4ファイルが登録されています。
Dependencies に登録されているファイルを、「依存ファイル」といい、主にヘッダファイルが登録されています。今回は、「h8_3048.h」と「initsct_3048.h」が登録されています。

4.3 ファイル編集

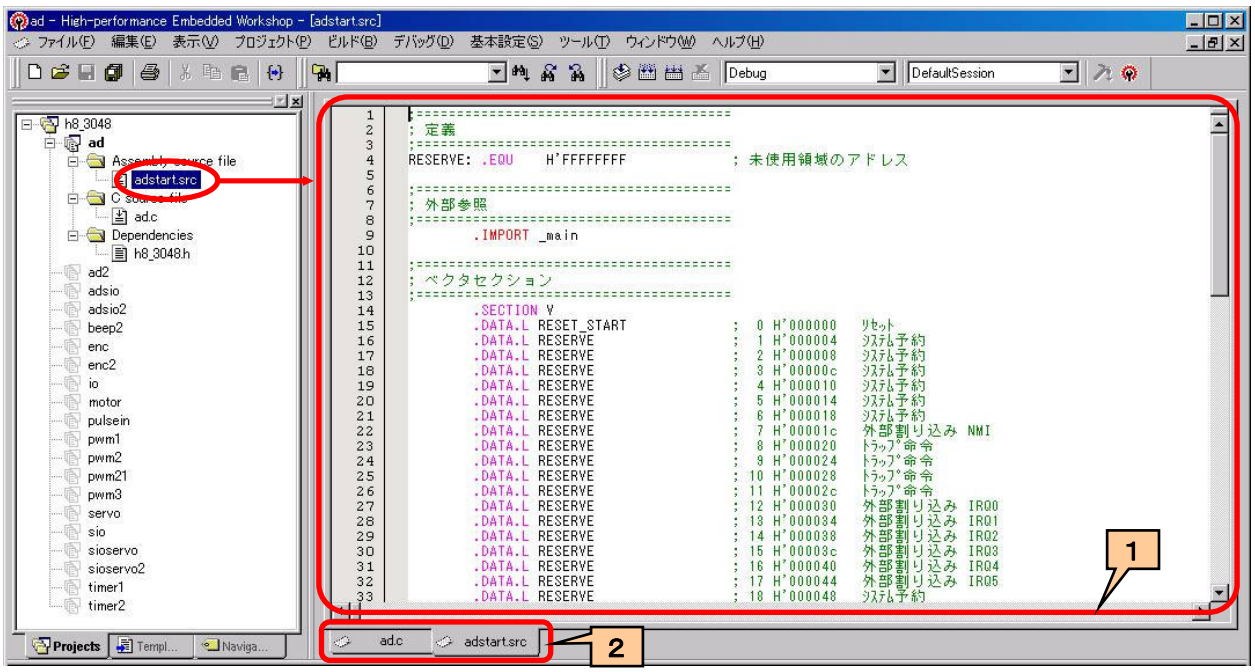
アクティブプロジェクトを「ad」に変更します。



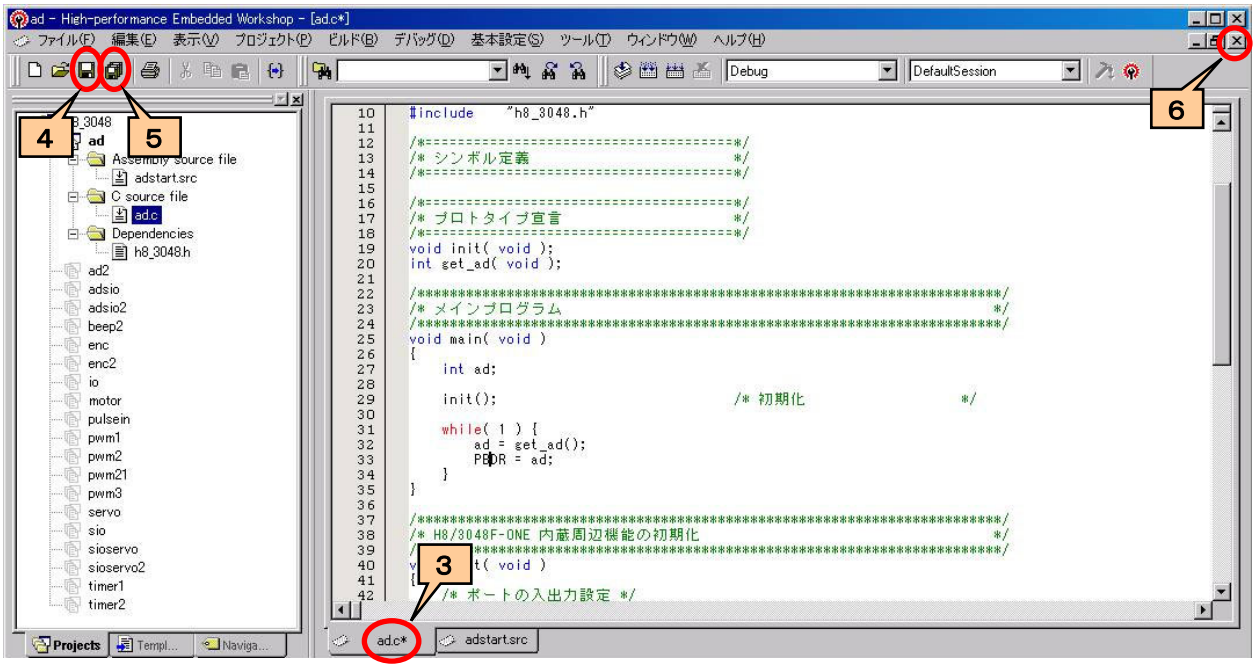
「ad.c」をダブルクリックすると、エディタウィンドウが開きます。ここでファイルを編集します。



左図のようにエディタウィンドウが小さく開いた場合、大きくしたいときは、○部分をクリックすると枠全体に広がります。



「adstart.src」をダブルクリックすると、1部分のように adstart.src のエディタウィンドウが開きます。
2つのファイルが開きました。ファイルを切り換えるには、2部分のタブで編集したいファイル名を選びます。



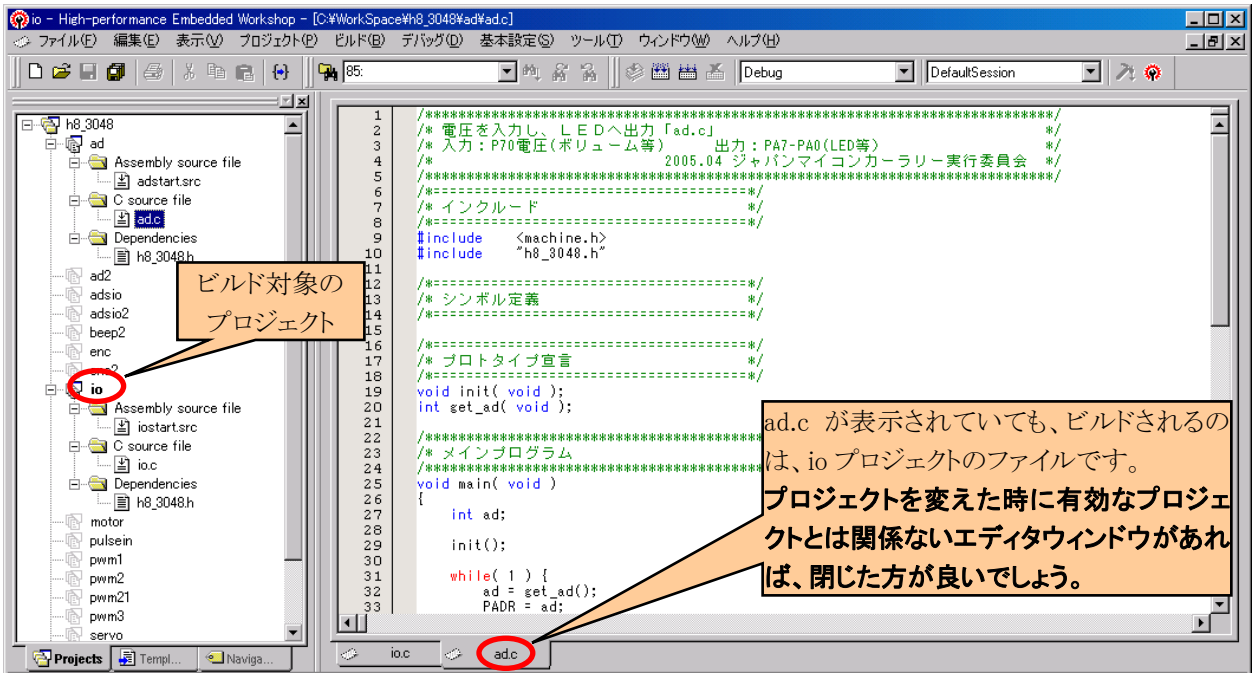
ファイルを編集して内容が変更されると、3部分に「＊」が表示されます。
4のアイコンは、現在編集中のファイルを保存するボタンです。
5のアイコンは、変更したすべてのファイルを保存するボタンです。

5のアイコンを使用すればすべてのファイルが保存されますので、適宜5のアイコンで保存してください。

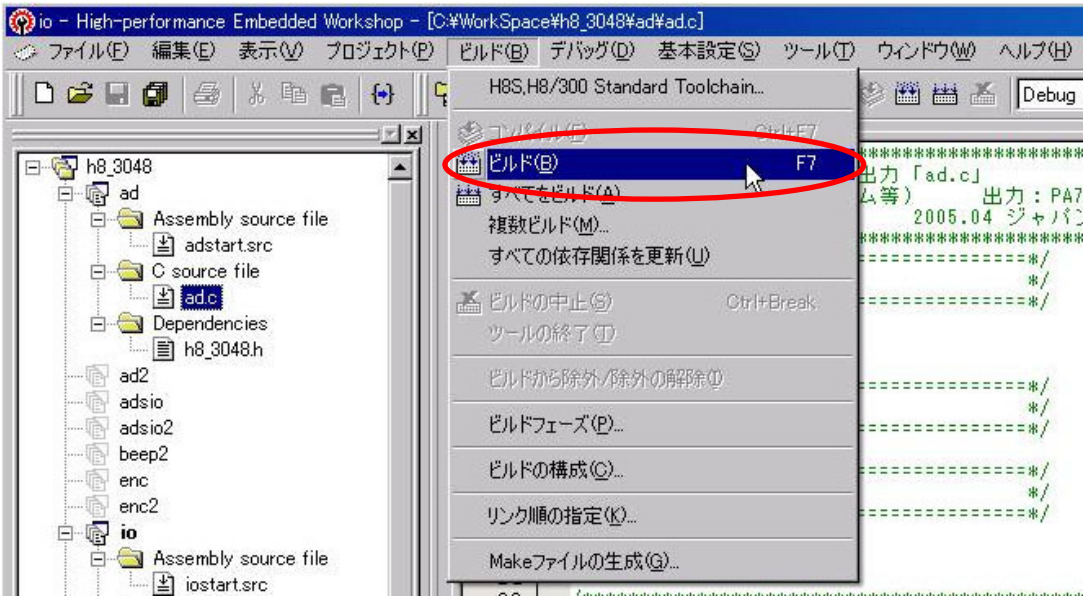
6の×ボタンで、編集中のファイルを閉じます。編集が終わって表示する必要のないファイルは、×ボタンで閉じてください。

4.4 ビルド（MOTファイルの作成）

実行委員会開発環境では、1つのsrcソースファイル、1つの C ファイルが対象でした。ルネサス統合開発環境は、プロジェクトに登録しているファイルすべてが対象となります。



ビルドは、現在有効なプロジェクトが対象となります。必ず有効なプロジェクトを確認してください。エディタウ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「iostart.src」がビルドに関係するファイルです**。



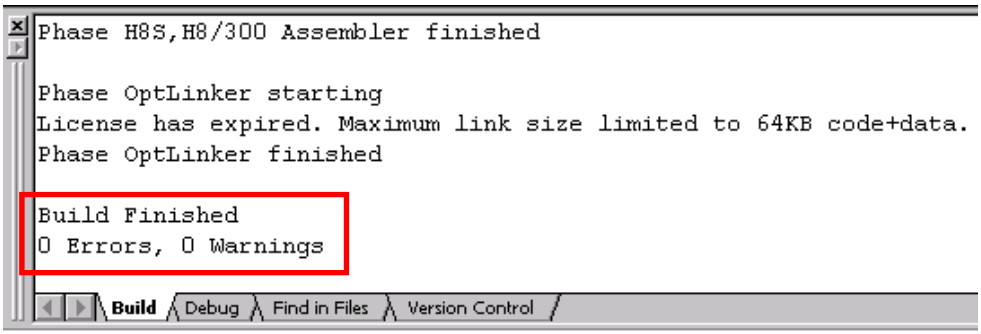
「ビルド」→「ビルド」でビルド作業を開始します。その下に「すべてをビルド」があります。違いは、

●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する

●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する

です。通常は、「ビルド」で全く問題ありません。

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、結果が下のように表示されます。



●*Error とは？*

誤りのことです。これが出た場合は必ずプログラムを直します。

●*Warning とは？*

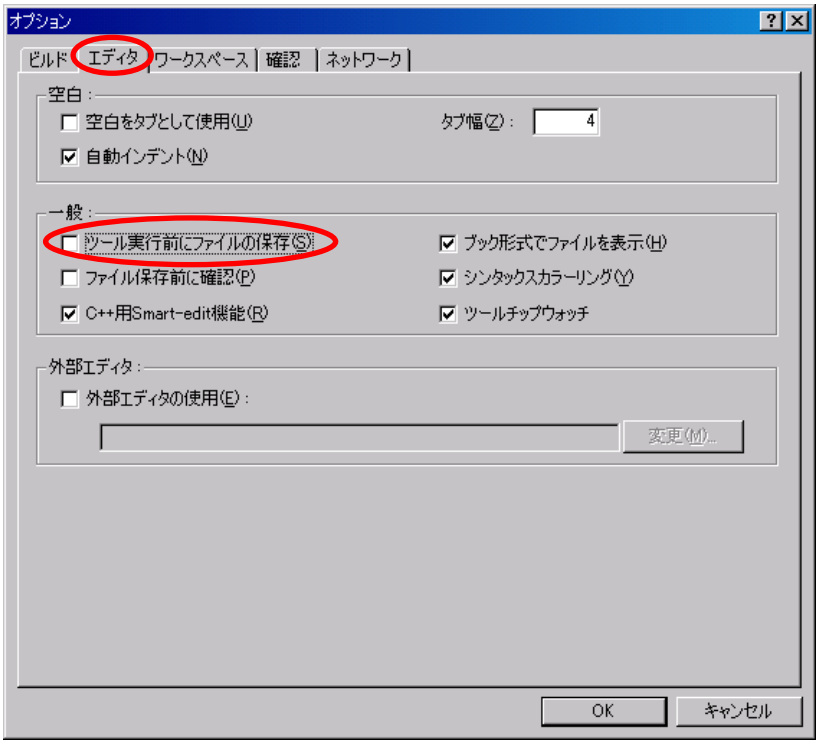
警告です。必ずしも誤っているとは言い切れないけども、間違っている可能性があるので確認してくださいというメッセージです。こちらも必ず直します。

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。

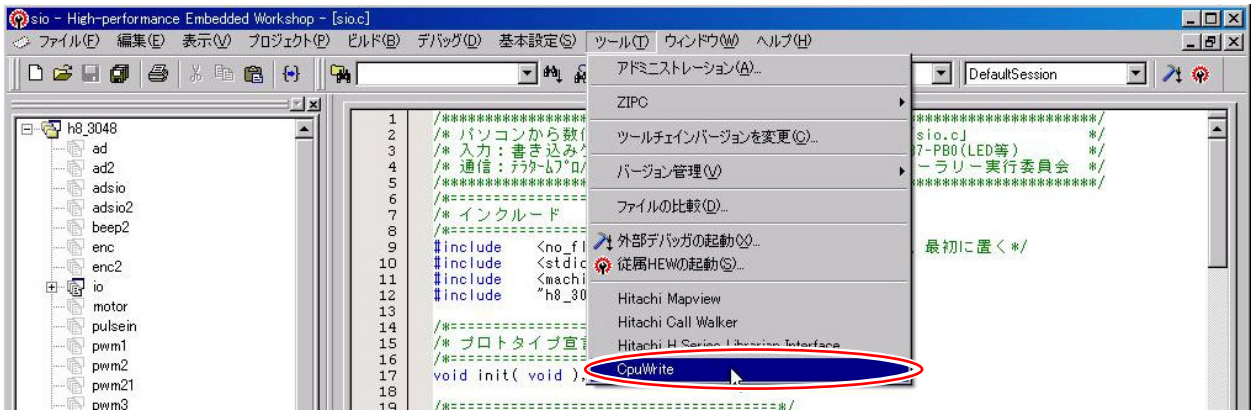
※ファイルの保存について

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。

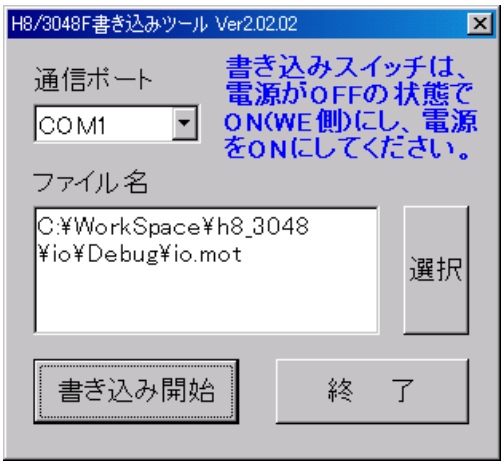
もし、自動保存をしたくない場合は、「基本設定→オプション」でオプション画面を開きます。「エディタ」タブを選び、「一般:ツール実行前にファイルの保存」のチェックを外します。



4.5 書き込みソフトの起動

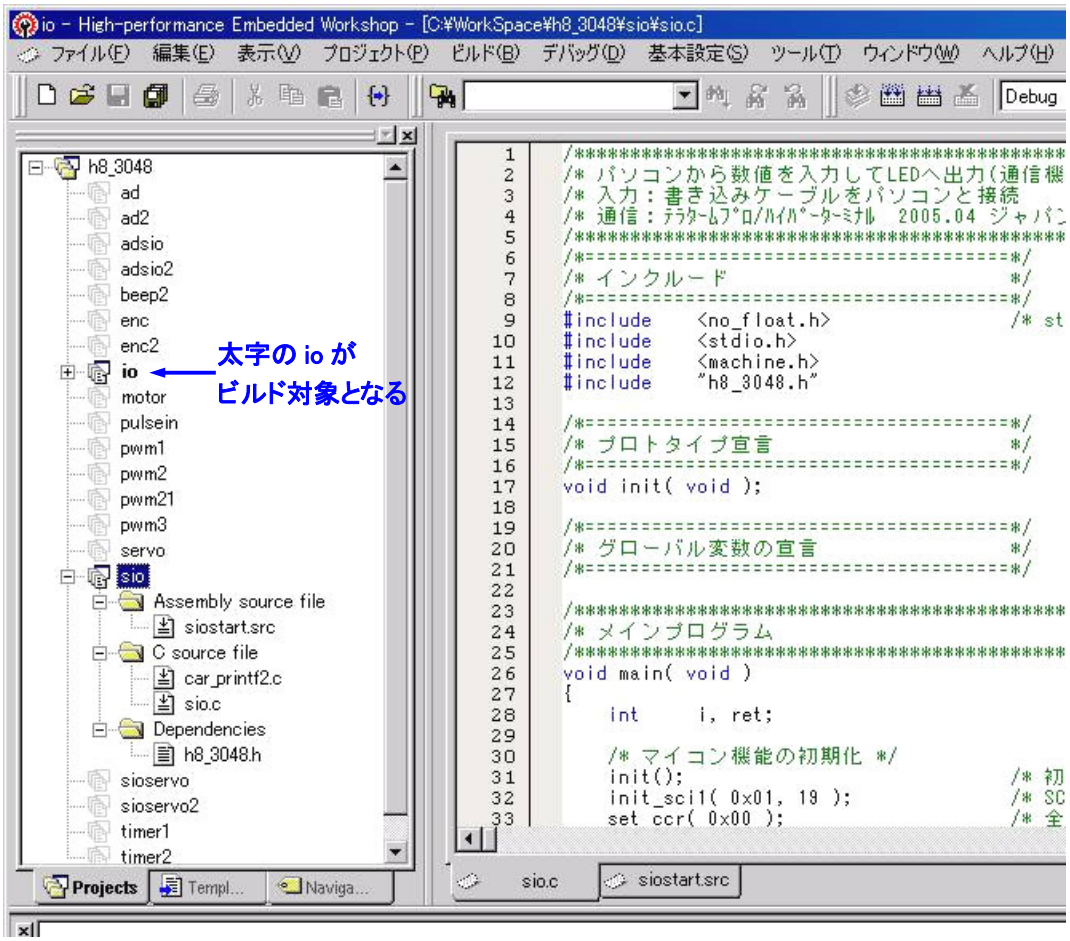


ルネサス統合開発環境の「ツール→CpuWrite」をクリックします。



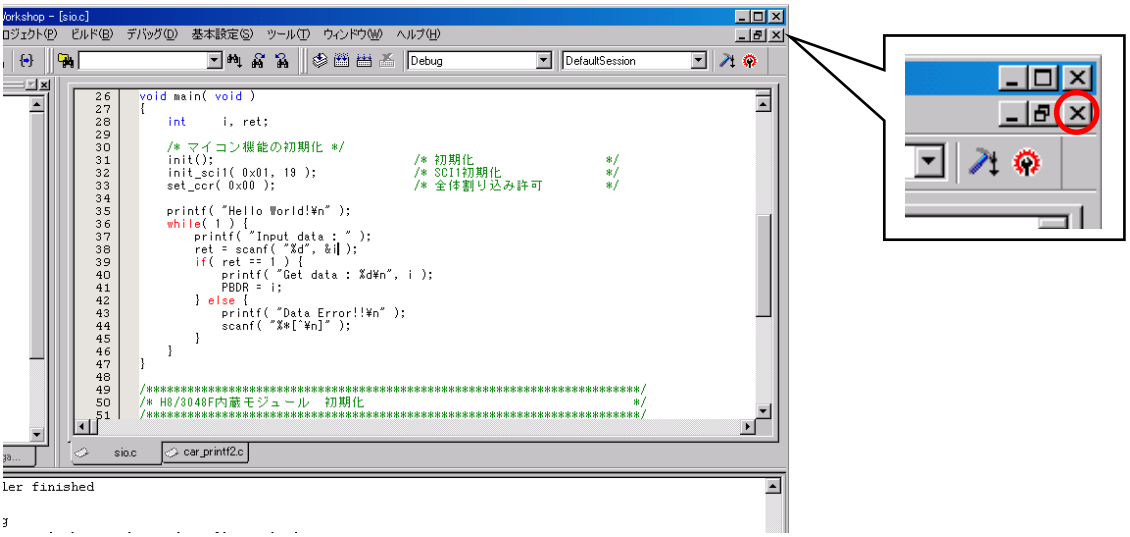
書き込みソフトが起動します。CPU ボードと書き込みソフトを操作してプログラムを書き込んで下さい。

4.6 プロジェクトを変更するときの注意点

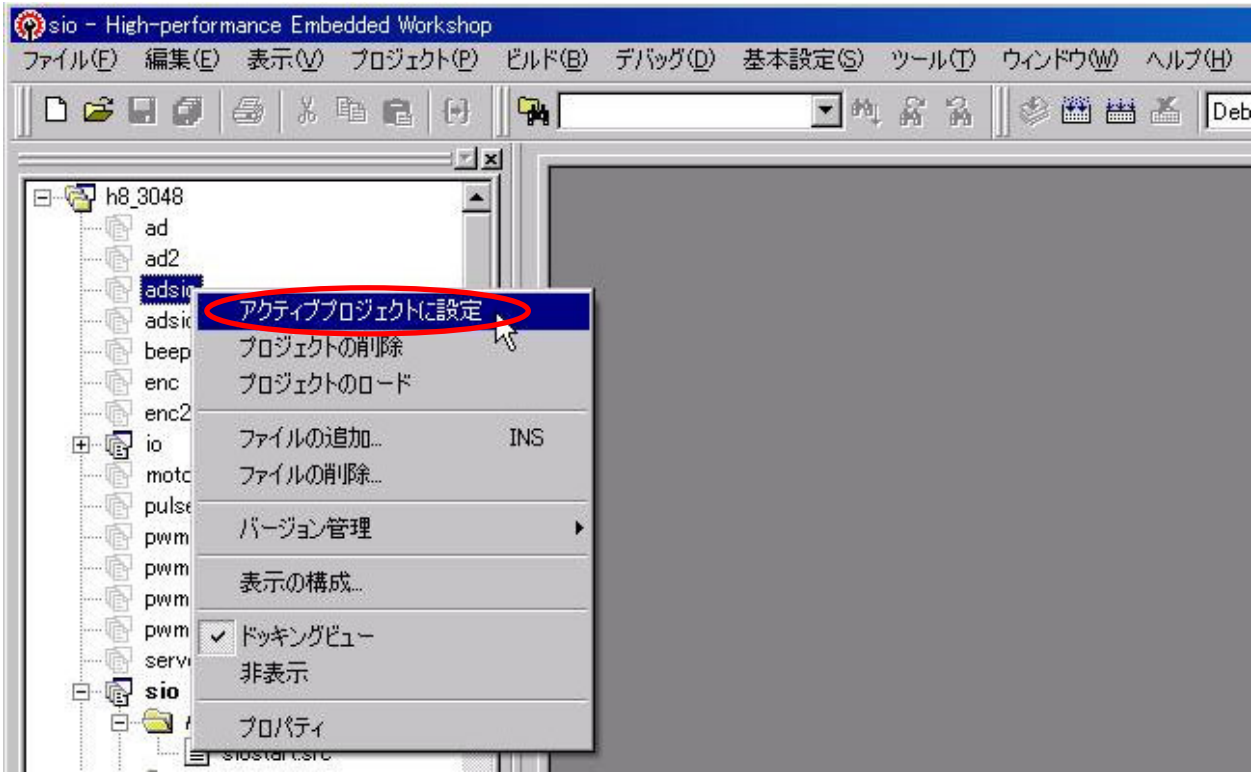


上画面は、「sio.c」、「siostart.src」というファイルを開いています。この状態で「ビルド→ビルド」を実行します。sio プロジェクトに関するファイルがビルドされ・・・・ませんﾞ！ ビルド対象はあくまで太字になっているプロジェクトである「io」プロジェクトです。**開いているファイルや、選択しているプロジェクトは関係ありません。**

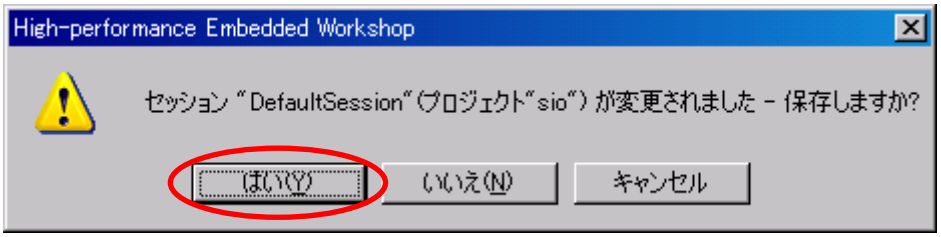
プロジェクトを変えるときは、このような間違えを無くすために必ず開いているファイルを閉じるという癖を付けておくと良いでしょう。



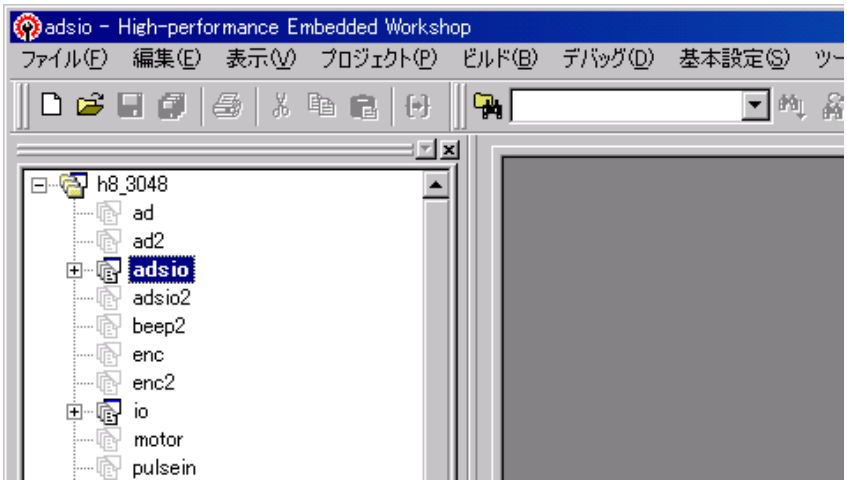
不要なエディタウィンドウは○部分をクリックして閉じておきます。



すべてのファイルを閉じたら、次の演習のプロジェクトを有効にします。例えば、「adsio」を有効にしたければ、「adsio」上で右クリック、「アクティブプロジェクトに設定」を選択します。



もし、上記のようなメッセージが出た場合は、「はい」で保存しておきます。セッションとはデバッグ装置を使うときに使用する機能です。今回は、デバッグ装置を使わないので関係ありませんが、一応保存しておきます。



プロジェクト「adsio」が有効なプロジェクトになりました。

5. プロジェクト「io」 I/Oポート入出力(センサ基板のチェック)

5.1 概要

実習基板のディップスイッチの値をマイコンで読み込みます。その値を LED に出力します。入力したデータを出力する、制御の基本中の基本です。マイコンのポートは、下記を使用します。

・ポート 7 の全ビット・・・ディップスイッチの状態を入力

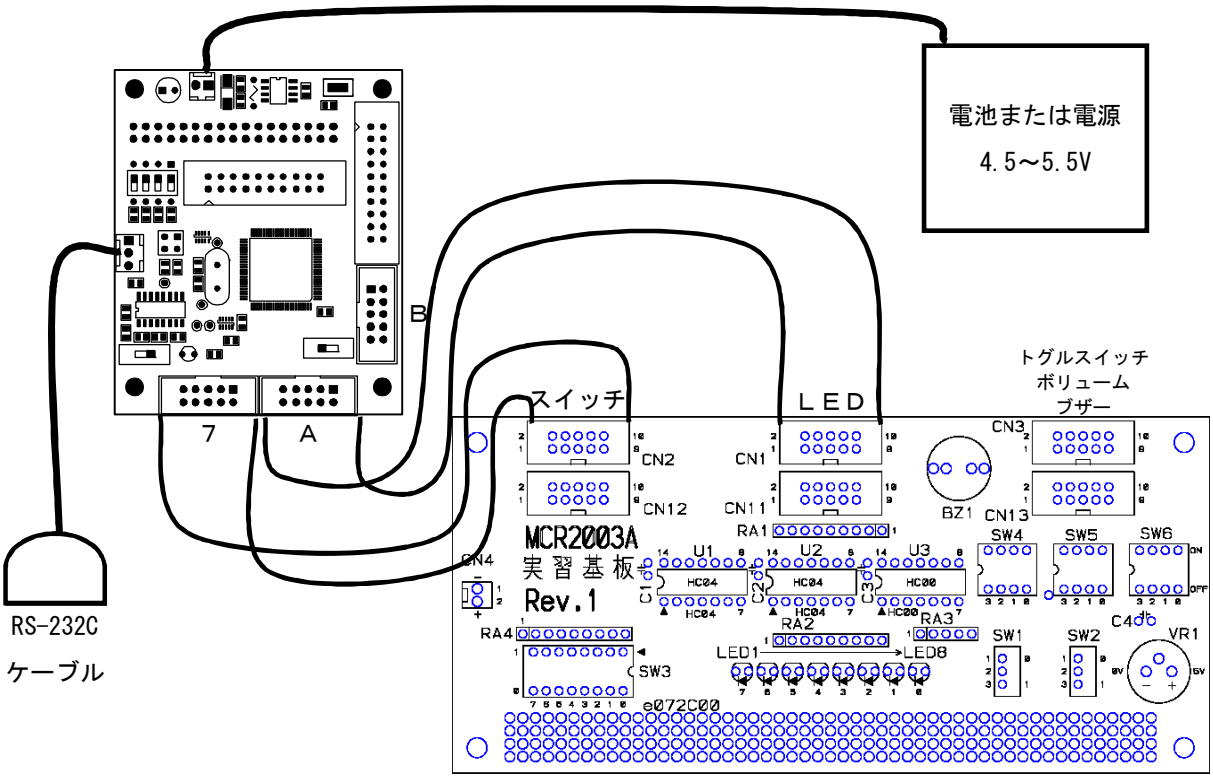
・ポート A の全ビット・・・LED へデータ出力

5.2 接続

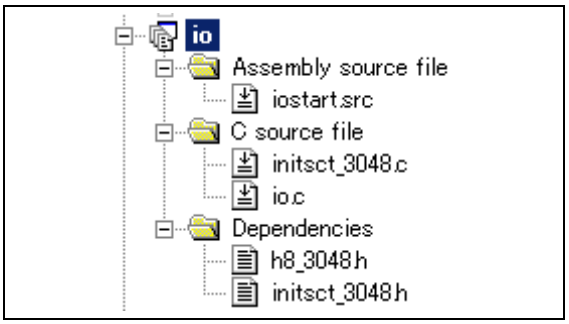
・CPU ボードのポート 7 と、実習基板のスイッチ部をフラットケーブルで接続します。

・CPU ボードのポート A と、実習基板の LED 部をフラットケーブルで接続します。

※ポート 7 のディップスイッチをセンサ基板に変えると、センサの反応を LED に出力することができます。センサ基板のチェックに便利です。



5.3 プロジェクトの構成



	ファイル名	内容
1	iostart.src	アセンブリ言語で記述されたアセンブリソースファイルです。このファイルの構造は下記のようになっています。 ベクタアドレス ＋ スタートアップルーチン
2	initsct_3048.c	初期値のないグローバル変数(セクション B 領域)、初期値のあるグローバル変数(セクション R 領域)の初期化用です。
3	io.c	実際に制御するプログラムが書かれています。H8/3048F-ONE の内蔵周辺機能の初期化も行います。
4	h8_3048.h	H8/3048F-ONE の内蔵周辺機能の I/O レジスタを定義したファイルです。
5	initsct_3048.h	initsct_3048.c のヘッダファイルです。

5.4 プログラム「io.c」

```
 1 :  /*********************************************************************/
 2 :  /* ポート７の状態をポートＡに出力(センサ基板のチェック)「io.c」             */
 3 :  /* 入力：P77-P70(スイッチ8ビットやセンサ基板等)     出力：PA7-PA0(LED等)    */
 4 :  /*                          2005.04 ジャパンマイコンカーラリー実行委員会  */
 5 :  /*********************************************************************/
 6 :  /*====================================*/
 7 :  /* インクルード                       */
 8 :  /*====================================*/
 9 :  #include    <machine.h>
10 :  #include    "h8_3048.h"
11 :
12 :  /*====================================*/
13 :  /* シンボル定義                       */
14 :  /*====================================*/
15 :
16 :  /*====================================*/
17 :  /* プロトタイプ宣言                   */
18 :  /*====================================*/
19 :  void init( void );
20 :
21 :  /*********************************************************************/
22 :  /* メインプログラム                                                  */
23 :  /*********************************************************************/
24 :  void main( void )
25 :  {
26 :      unsigned char d;
27 :
28 :      init();                         /* 初期化                   */
29 :
30 :      while( 1 ) {
31 :          d = P7DR;
32 :          PADR = d;
33 :      }
34 :  }
35 :
36 :  /*********************************************************************/
37 :  /* H8/3048F-ONE 内蔵周辺機能の初期化                                 */
38 :  /*********************************************************************/
39 :  void init( void )
40 :  {
41 :      /* ポートの入出力設定 */
42 :      P1DDR = 0xff;
43 :      P2DDR = 0xff;
44 :      P3DDR = 0xff;
45 :      P4DDR = 0xff;
46 :      P5DDR = 0xff;
47 :      P6DDR = 0xf0;                   /* CPU基板上のDIP SW        */
48 :      P8DDR = 0xff;
49 :      P9DDR = 0xf7;
50 :      PADDR = 0xff;                   /* LED基板                  */
51 :      PBDDR = 0xff;
52 :      /* ポート７は、入力専用なので入出力設定はありません   */
53 :  }
54 :
55 :  /*********************************************************************/
56 :  /* end of file                                                       */
57 :  /*********************************************************************/
```

5.5 プログラム「iostart.src」

```
 1 :  ;====================================
 2 :  ; 定義
 3 :  ;====================================
 4 :  RESERVE: .EQU    H'FFFFFFFF            ; 未使用領域のアドレス
 5 :
 6 :  ;====================================
 7 :  ; 外部参照
 8 :  ;====================================
 9 :          .IMPORT _main
10 :          .IMPORT _INITSCT
11 :
12 :  ;====================================
13 :  ; ベクタセクション
14 :  ;====================================
15 :          .SECTION V
16 :          .DATA.L RESET_START           ;  0 H'000000   ﾘｾｯﾄ
17 :          .DATA.L RESERVE               ;  1 H'000004   ｼｽﾃﾑ予約
18 :          .DATA.L RESERVE               ;  2 H'000008   ｼｽﾃﾑ予約
19 :          .DATA.L RESERVE               ;  3 H'00000c   ｼｽﾃﾑ予約
20 :          .DATA.L RESERVE               ;  4 H'000010   ｼｽﾃﾑ予約
21 :          .DATA.L RESERVE               ;  5 H'000014   ｼｽﾃﾑ予約
```

```
22 :          .DATA.L RESERVE                 ;  6 H'000018   ｼｽﾃﾑ予約
23 :          .DATA.L RESERVE                 ;  7 H'00001c   外部割り込み NMI
24 :          .DATA.L RESERVE                 ;  8 H'000020   ﾄﾗｯﾌﾟ命令
25 :          .DATA.L RESERVE                 ;  9 H'000024   ﾄﾗｯﾌﾟ命令
26 :          .DATA.L RESERVE                 ; 10 H'000028   ﾄﾗｯﾌﾟ命令
27 :          .DATA.L RESERVE                 ; 11 H'00002c   ﾄﾗｯﾌﾟ命令
28 :          .DATA.L RESERVE                 ; 12 H'000030   外部割り込み IRQ0
29 :          .DATA.L RESERVE                 ; 13 H'000034   外部割り込み IRQ1
30 :          .DATA.L RESERVE                 ; 14 H'000038   外部割り込み IRQ2
31 :          .DATA.L RESERVE                 ; 15 H'00003c   外部割り込み IRQ3
32 :          .DATA.L RESERVE                 ; 16 H'000040   外部割り込み IRQ4
33 :          .DATA.L RESERVE                 ; 17 H'000044   外部割り込み IRQ5
34 :          .DATA.L RESERVE                 ; 18 H'000048   ｼｽﾃﾑ予約
35 :          .DATA.L RESERVE                 ; 19 H'00004c   ｼｽﾃﾑ予約
36 :          .DATA.L RESERVE                 ; 20 H'000050   WDT MOVI
37 :          .DATA.L RESERVE                 ; 21 H'000054   REF CMI
38 :          .DATA.L RESERVE                 ; 22 H'000058   ｼｽﾃﾑ予約
39 :          .DATA.L RESERVE                 ; 23 H'00005c   ｼｽﾃﾑ予約
40 :          .DATA.L RESERVE                 ; 24 H'000060   ITU0 IMIA0
41 :          .DATA.L RESERVE                 ; 25 H'000064   ITU0 IMIB0
42 :          .DATA.L RESERVE                 ; 26 H'000068   ITU0 OVI0
43 :          .DATA.L RESERVE                 ; 27 H'00006c   ｼｽﾃﾑ予約
44 :          .DATA.L RESERVE                 ; 28 H'000070   ITU1 IMIA1
45 :          .DATA.L RESERVE                 ; 29 H'000074   ITU1 IMIB1
46 :          .DATA.L RESERVE                 ; 30 H'000078   ITU1 OVI1
47 :          .DATA.L RESERVE                 ; 31 H'00007c   ｼｽﾃﾑ予約
48 :          .DATA.L RESERVE                 ; 32 H'000080   ITU2 IMIA2
49 :          .DATA.L RESERVE                 ; 33 H'000084   ITU2 IMIB2
50 :          .DATA.L RESERVE                 ; 34 H'000088   ITU2 OVI2
51 :          .DATA.L RESERVE                 ; 35 H'00008c   ｼｽﾃﾑ予約
52 :          .DATA.L RESERVE                 ; 36 H'000090   ITU3 IMIA3
53 :          .DATA.L RESERVE                 ; 37 H'000094   ITU3 IMIB3
54 :          .DATA.L RESERVE                 ; 38 H'000098   ITU3 OVI3
55 :          .DATA.L RESERVE                 ; 39 H'00009c   ｼｽﾃﾑ予約
56 :          .DATA.L RESERVE                 ; 40 H'0000a0   ITU4 IMIA4
57 :          .DATA.L RESERVE                 ; 41 H'0000a4   ITU4 IMIB4
58 :          .DATA.L RESERVE                 ; 42 H'0000a8   ITU4 OVI4
59 :          .DATA.L RESERVE                 ; 43 H'0000ac   ｼｽﾃﾑ予約
60 :          .DATA.L RESERVE                 ; 44 H'0000b0   DMAC DEND0A
61 :          .DATA.L RESERVE                 ; 45 H'0000b4   DMAC DEND0B
62 :          .DATA.L RESERVE                 ; 46 H'0000b8   DMAC DEND1A
63 :          .DATA.L RESERVE                 ; 47 H'0000bc   DMCA DEND1B
64 :          .DATA.L RESERVE                 ; 48 H'0000c0   ｼｽﾃﾑ予約
65 :          .DATA.L RESERVE                 ; 49 H'0000c4   ｼｽﾃﾑ予約
66 :          .DATA.L RESERVE                 ; 50 H'0000c8   ｼｽﾃﾑ予約
67 :          .DATA.L RESERVE                 ; 51 H'0000cc   ｼｽﾃﾑ予約
68 :          .DATA.L RESERVE                 ; 52 H'0000d0   SCI0 ERI0
69 :          .DATA.L RESERVE                 ; 53 H'0000d4   SCI0 RXI0
70 :          .DATA.L RESERVE                 ; 54 H'0000d8   SCI0 TXI0
71 :          .DATA.L RESERVE                 ; 55 H'0000dc   SCI0 TEI0
72 :          .DATA.L RESERVE                 ; 56 H'0000e0   SCI1 ERI1
73 :          .DATA.L RESERVE                 ; 57 H'0000e4   SCI1 RXI1
74 :          .DATA.L RESERVE                 ; 58 H'0000e8   SCI1 TXI1
75 :          .DATA.L RESERVE                 ; 59 H'0000ec   SCI1 TEI1
76 :          .DATA.L RESERVE                 ; 60 H'0000f0   A/D ADI
77 :
78 :  ;=====================================
79 :  ; スタートアッププログラム
80 :  ;=====================================
81 :          .SECTION P
82 :  RESET_START:
83 :          MOV.L   #H'FFF10,ER7            ; スタックの設定
84 :          JSR     @_INITSCT               ; RAMエリアの初期化
85 :          JSR     @_main                  ; Ｃ言語のmain()関数へジャンプ
86 :  OWARI:
87 :          BRA     OWARI
88 :
89 :          .END
```

5.6 プロジェクトのファイル構成

プロジェクト「io」は、「iostart.src」と「io.c」と「initsct_3048.c」で構成されています。プロジェクトに登録しているファイルの関数は、ファイルをまたいで呼び出すことができます。例えば、「iostart.src」から「io.c」の main 関数を呼び出すことができます。ただし、

・アセンブリソースファイルから他のファイルの関数を呼び出すときは、「IMPORT」を宣言

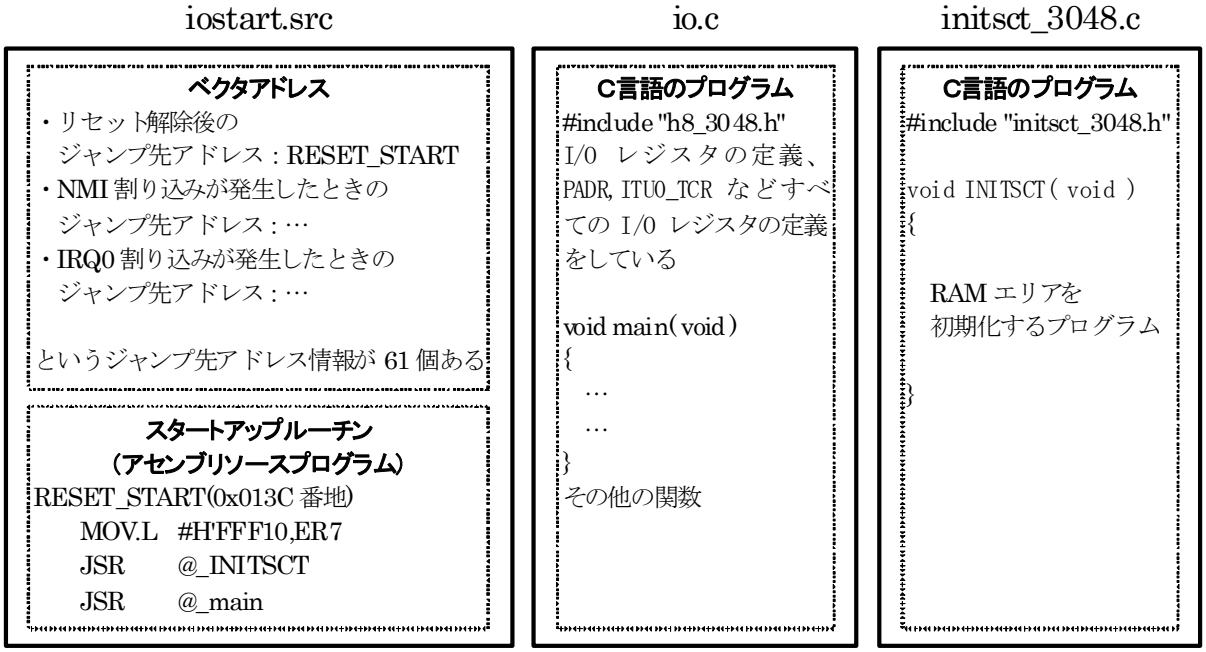
・C 言語ソースファイルから他の C 言語ソースファイルの関数を呼び出すときは、
ヘッダファイル(拡張子 h ファイル)を include
しておく必要があります。後述します。

iostart.src は、アセンブリ言語で記述されたアセンブリソースファイルです。このファイルの構造は下記のようになっています。

iostart.src ＝ ベクタアドレス ＋ スタートアップルーチン

io.c は、C 言語ソースファイルです。このファイルは、H8 内蔵周辺機能の初期化、メインで実行するプログラムが含まれています。

initsct_3048.c は、C 言語ソースファイルです。このファイルは、RAM エリアを初期化するプログラムが入っています。このファイルは、すべてのプロジェクトに入れると覚えておいてください。

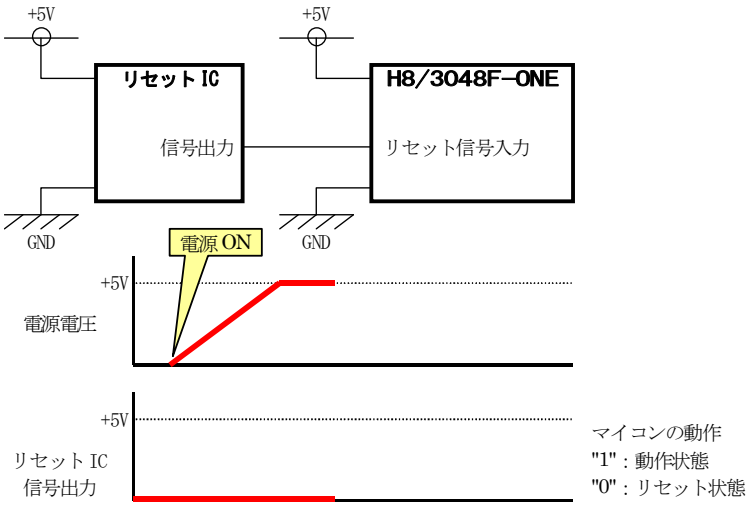


5.7 プログラム「iostart.src」の解説

どのような順番でプログラムが実行されていくのか、説明していきます。

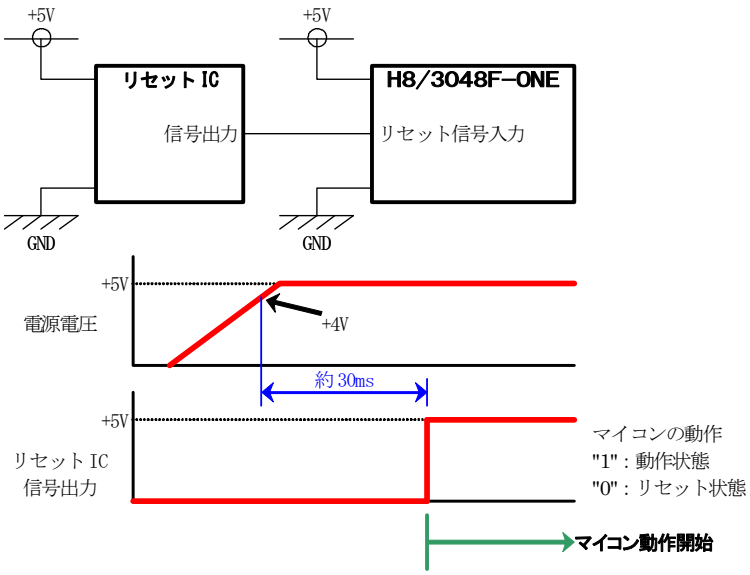
5.7.1 電源を入れたときの動作

マイコンボードの電源を入れます。電源を入れた瞬間は、電圧が安定しません。そのため、外付けのリセット IC の出力信号がしばらくの間、"0"を出力します。出力先は H8 マイコンのリセット端子です。リセット端子="0"でマイコンはリセット状態となるため、何もしません。



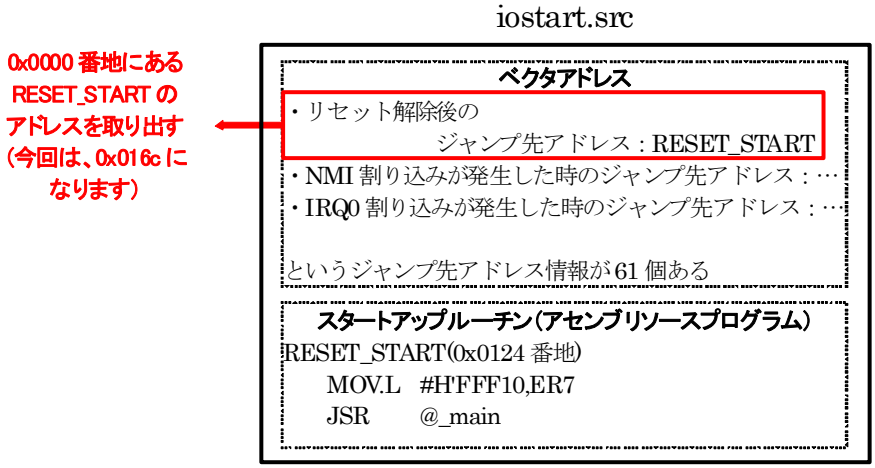
5.7.2 マイコンの動作開始

4V になってから約 30ms たつと、リセット IC は電源が安定しマイコンの準備ができたと判断して"1"を出力します。マイコンは、リセット信号入力端子が"1"となり動き始めます。ちなみに 30ms というのは、RY3048Fone ボードに取り付けているリセット IC により、電源を入れてから 30ms 後に"0"→"1"になるためです。時間はリセット IC の種類によって違います。今回のリセット IC はたまたま 30ms だったと言うだけです。ちなみに、H8/3048F-ONE ハードウェアマニュアルには、「電源投入後 20ms 以上たってからリセット解除すること」と記載されています。



5.7.3 ベクタアドレスからジャンプ先アドレスを取り出す

マイコンが動作を開始して初めに行うことは、ベクタアドレスと呼ばれる領域の「リセットが解除されたときのジャンプ先アドレス」が書いてある部分(0番)から値を取り出すことです。



ベクタアドレスは、0 番～60 番まであります。詳しくは、「(2) 割り込みの種類とベクタアドレス」を参照してください。

あらかじめ、「○○の割り込みが発生したときは、○○番からジャンプ先アドレスを読み込む」と決まっています。割り込みを使うときは、該当する場所(ベクタアドレス)にジャンプ先アドレスを登録しておきます。**リセットを解除したときは、0番(0x0000 番地)からジャンプ先アドレスを読み込みます。**そこには、「RESET_START」のアドレスを登録しておきます。マイコンは、「RESET_START」のアドレスである「0x016c」という値を読み込みます(プロジェクト「io」の場合)。他の割り込みは使っていませんので登録していません。

プログラムでは、下記のようにベクタアドレスを記述します。

```
 15 :          .SECTION V
 16 :          .DATA.L RESET_START             ;  0 H'000000   ﾘｾｯﾄ          ←０番のジャンプ先アドレス
 17 :          .DATA.L RESERVE                 ;  1 H'000004   ｼｽﾃﾑ予約     ←１番のジャンプ先アドレス
 18 :          .DATA.L RESERVE                 ;  2 H'000008   ｼｽﾃﾑ予約     ←２番のジャンプ先アドレス
 19 :          .DATA.L RESERVE                 ;  3 H'00000c   ｼｽﾃﾑ予約     ←３番のジャンプ先アドレス
中略
 75 :          .DATA.L RESERVE                 ; 59 H'0000ec   SCI1 TEI1   ←５９番のジャンプ先アドレス
 76 :          .DATA.L RESERVE                 ; 60 H'0000f0   A/D ADI     ←６０番のジャンプ先アドレス
```

0 番なら、

```
 16 :          .DATA.L RESET_START             ;  0 H'000000   ﾘｾｯﾄ
```

の部分が読み込まれます。「.DATA.L」はロングサイズ(4バイト)でデータを作りなさいという命令です。「RESET_START」のある番地データを作ります。io プロジェクトでは「RESET_START」は、0x16c 番地にあるので

```
 16 :          .DATA.L H'0000016c              ;  0 H'000000   ﾘｾｯﾄ
```

となります。その他の番号には

```
 17 :          .DATA.L RESERVE                 ;  1 H'000004   ｼｽﾃﾑ予約
```

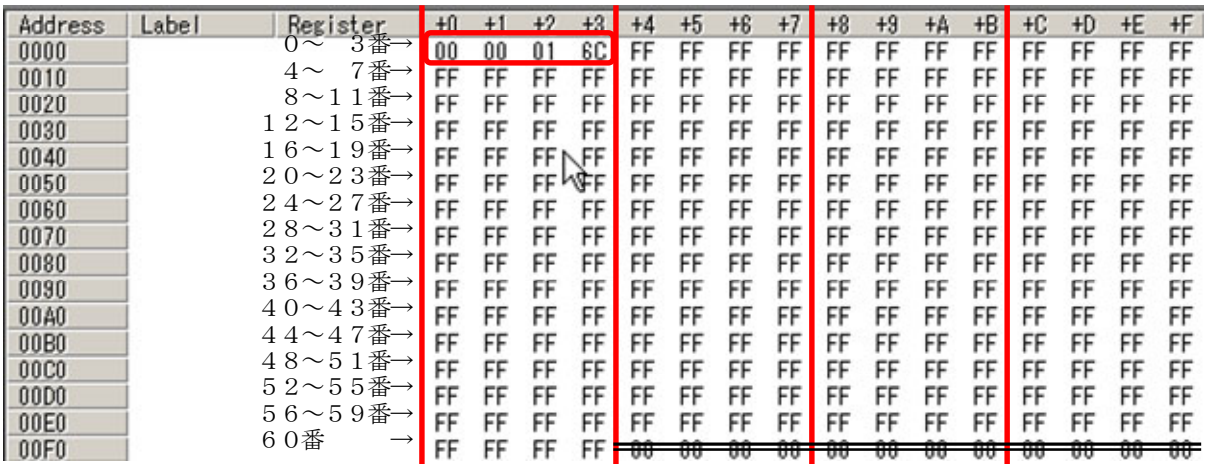
と、「RESERVE」が入っています。登録する必要のない番号には「RESERVE」を入れておきます。「RESERVE」は、

```
4 :  RESERVE: .EQU     H'FFFFFFFF                ; 未使用領域のアドレス
```

と登録しています。.EQU は、「イコール」命令です。「RESERVE」という単語が出てきたら「H'FFFFFFFF」に置き換えなさい、という意味です。登録する必要のない場所は、アドレスが分かりませんので、ダミーとして「H'FFFFFFFF」を入れておきます。結果的には、

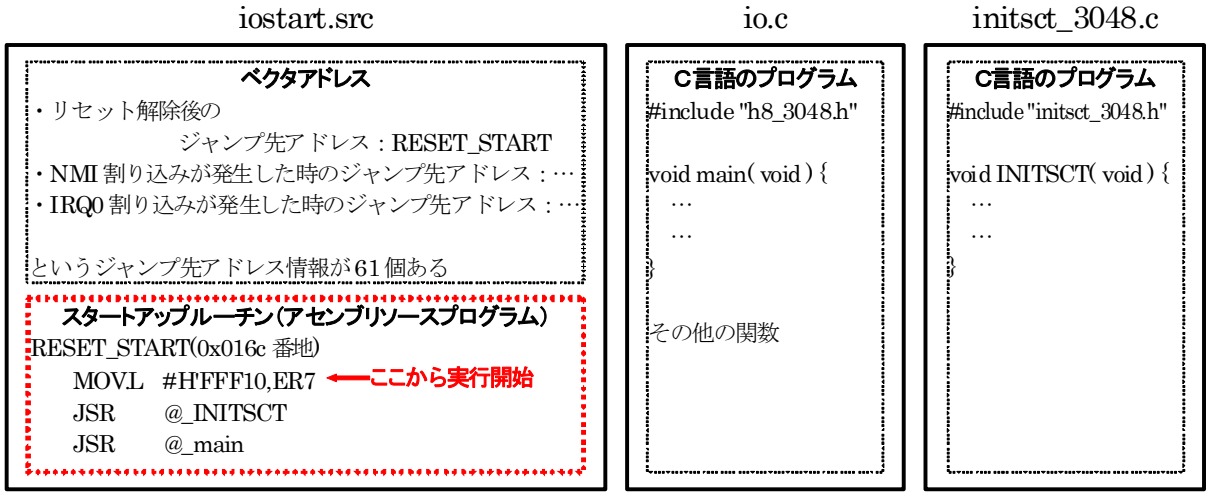
```
17 :            .DATA.L H'FFFFFFFF                ;  1 H'000004    ｼｽﾃﾑ予約
```

ということになります。下記に実際のメモリの値を示します。4 バイトごとに 0～60 番まであります。



5.7.4 スタートアップルーチンの実行

0x016c 番地には、初期設定を行うプログラム(スタートアップルーチン)があります。実際のプログラムは下記のようになっています。

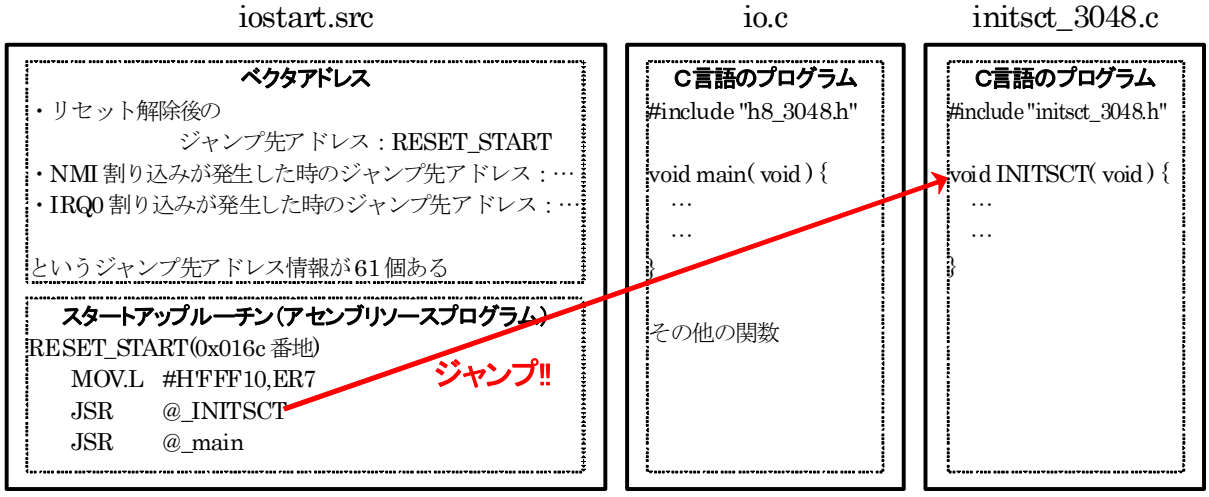


5.7.5 INITSCT関数へジャンプ

初期設定ができたらC言語プログラムの INITSCT 関数へ飛んで、プログラムを続けます。

```
RESET_START:
        MOV.L   #H'FFF10,ER7            ; スタックの設定
        JSR     @_INITSCT               ; RAM エリアの初期化
        JSR     @_main                  ; C言語の main()関数へジャンプ
OWARI:
        BRA     OWARI
```

JSR とは、「ジャンプサブルーチン」の略で、INITSCT へジャンプしてその処理が終わったら戻ってきなさい、という意味です。ちなみに「INITSCT」の前の「__」(アンダーバー)は、アセンブリソースプログラムからC言語ソースプログラムの関数を呼び出す場合は、「__」を付けなければいけないという決まりがあるためです。



5.7.6 IMPORT宣言

ただ、ちょっとした問題があります。アセンブルやコンパイルはファイルごとに行います。

```
84 :            JSR     @_INITSCT                ; RAM エリアの初期化
```

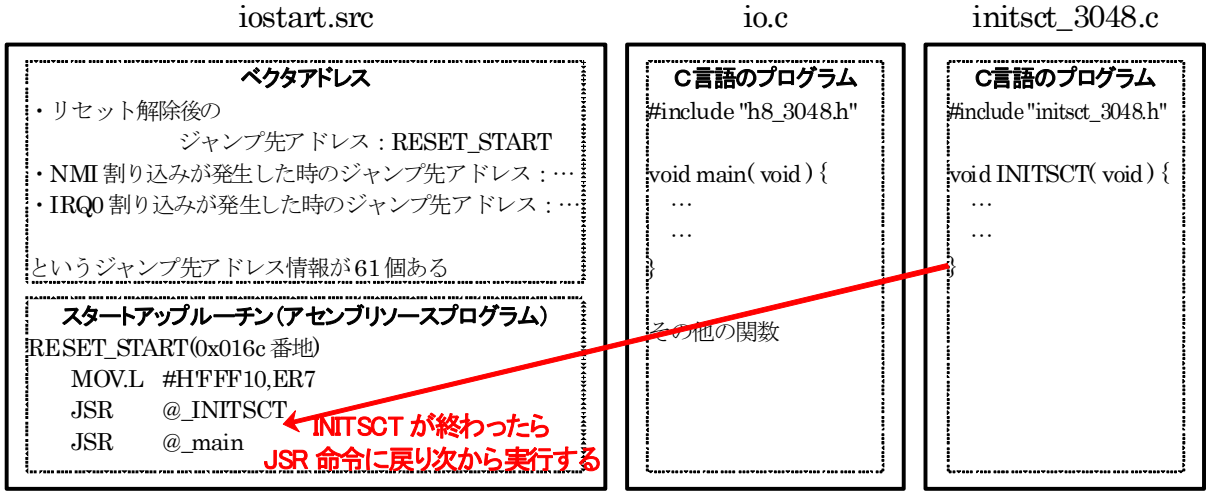
をアセンブルするとき、「_INITSCT」を探します。しかし、「_INITSCT」は iostart.src 内にはありません。「_INITSCT」は、initsct_3048.c ファイル内にあります。そのため、「_INITSCT」は他の場所にあるので、他のファイルを探してください、とアセンブラに知らせる必要があります。その命令が、「IMPORT」命令なのです。

```
10 :            .IMPORT _INITSCT
```

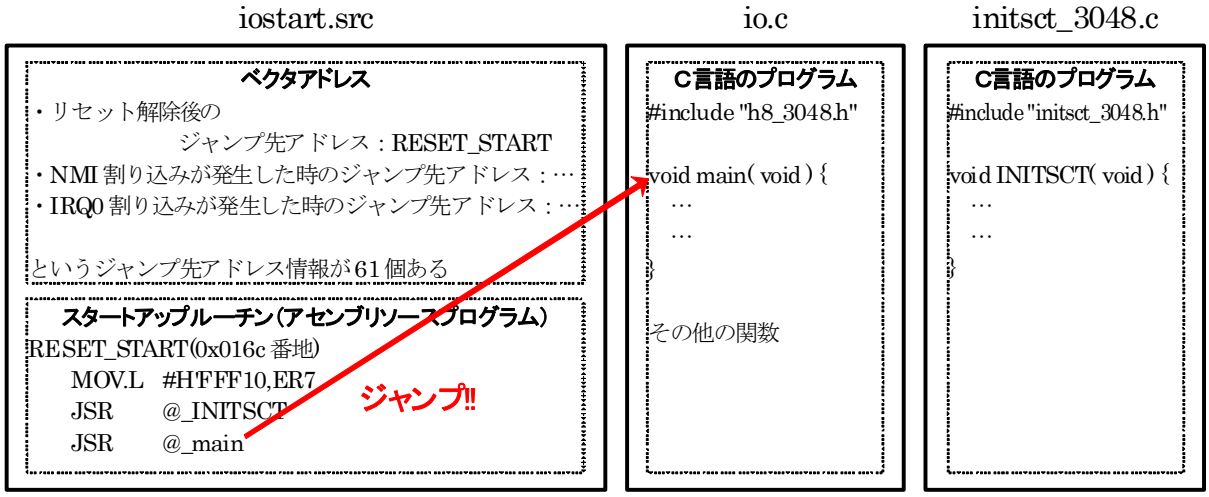
という記述で、リンケージエディタは「_INITSCT」が他のファイルにあることを理解して、84 行は、他のファイルにある「_ INITSCT」というラベル名へジャンプするようマシン語に変換します。

5.7.7 main関数へジャンプ

INITSCT 関数が実行し終わると、呼び出し先の命令の次からプログラムの実行を続けます。



次は、「_main」関数へジャンプしてその処理が終わったら戻ってきなさい、という命令です。



main 関数が実行し終わると、呼び出し先の命令の次からプログラムの実行を続けます。

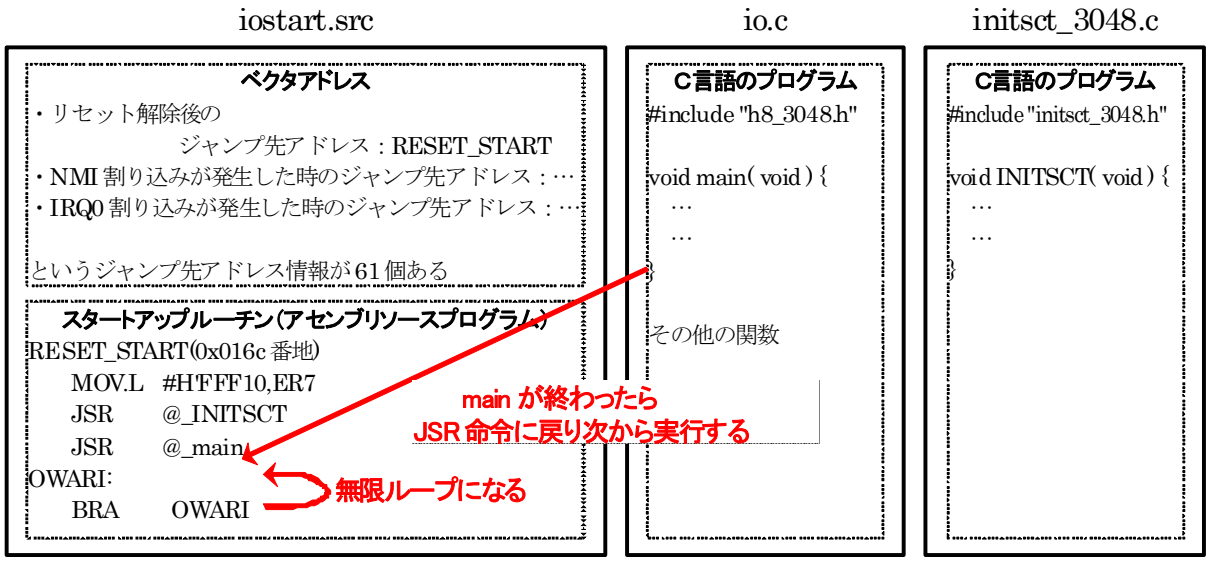
5.7.8 スタートアップルーチンの最後

スタートアップルーチンの最後は、次のようにプログラムしています。

```
86 :  OWARI:
87 :          BRA      OWARI
```

「BRA」は「Branch Always」の略で、「必ず分岐(ジャンプ)しなさい」という意味です。どこにジャンプするかというと、「OWARI」と書いてあるラベルへジャンプします。「OWARI」は 1 行上にあるので、自分自身にジャンプ、すなわち無限ループになります。

main 関数が終わるとどうなるのでしょう。「JSR @_main」は、main 関数の処理が終わると、戻ってきて次の行に進みます。次の行にある命令が、「BRA OWARI」なのです。この命令は、前記の通り無限ループなので、ここで何もせずに待つことになります。電源が OFF になるか、リセットするまでこの状態です。



5.8 プログラム「initsct_3048.c」の解説

ルネサス統合開発環境　操作マニュアル　応用編のセクションに関わる部分を参照してください。

5.9 プログラム「io.c」の解説

5.9.1 「machine.h」ファイルの取り込み

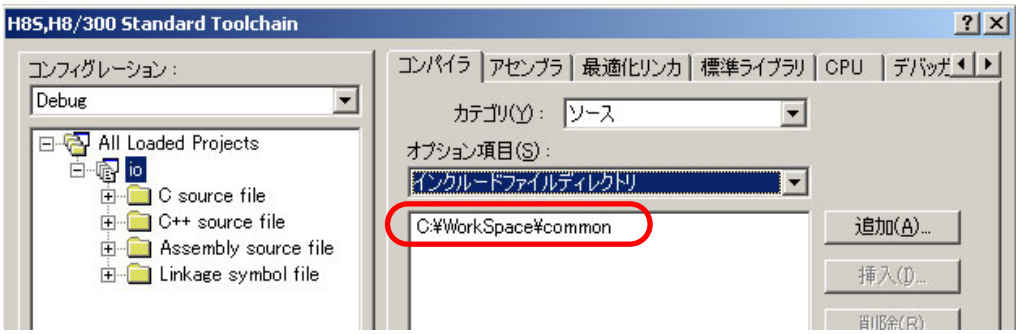
```
9 ： #include    〈machine.h〉
```

「#include」がインクルード命令です。インクルード命令は、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「＜　＞」で囲い、その中に記述します。ここでは、「machine.h」を取り込んでいます。
まず、「machine.h」ファイルを読み込みます。このヘッダファイルは、C 言語で記述できない CPU に特化した機能を提供する組み込み関数です。

5.9.2 「h8_3048.h」ファイルの取り込み

```
10 ： #include    "h8_3048.h"
```

次に「h8_3048.h」ファイルを読み込みます。先ほどとは違い、「"　　"」で囲っています。どう違うのでしょうか。違いは、

＜　＞	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で用意されているファイルは、こちらを使います。 stdio.h , stdlib.h , math.h　などです。ちなみに、「include フォルダ」は、 C:¥Program Files¥Renesas¥Hew¥Tools¥Renesas¥H8¥6_1_2¥include フォルダにあります(「6_1_2」はバージョンによって変わります)。
"　"	次の手順でファイルを探します。 ①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)を探す ②ツールチェインで設定したフォルダから探す(下記)  ③通常の include フォルダから取り込むファイルを探す 自分で作ったファイルは、こちらを使います。

という違いがあります。「h8_3048.h」ファイルは標準で用意されているヘッダファイルではなく、ジャパンマイコンカーラリー実行委員会が作成したヘッダファイルです。そのため、「"　　"」で囲みます。

5.9.3 「h8_3048.h」ファイルの内容

なぜ、「h8_3048.h」という別ファイルにするのでしょうか。中身は、H8/3048F-ONE 用の内蔵周辺機能の I/O レジスタを定義したファイルです。C 言語ソースファイルごとに同じ内容を数百行も記述することになります。そのため、**ファイルごとに共通化できる定義がある場合、ヘッダファイルとして別ファイル化して、インクルードで対応するのが慣習です**。

「h8_3048.h」は、下記のような内容です。

```
 1 :  /************************************************************************/
 2 :  /* H8/3048F-ONE用内蔵周辺機能のI/Oレジスタ定義 Ver1.02                  */
 3 :  /*    2005.04 マイコンカーラリー実行委員会                              */
 4 :  /************************************************************************/
 5 :
 6 :  #define P1DDR           (*(unsigned char*)0xfffc0)
 7 :  #define P1DR            (*(unsigned char*)0xfffc2)
 8 :  #define P2DDR           (*(unsigned char*)0xfffc1)
 9 :  #define P2DR            (*(unsigned char*)0xfffc3)
10 :  #define P3DDR           (*(unsigned char*)0xfffc4)
11 :  #define P3DR            (*(unsigned char*)0xfffc6)
12 :  #define P4DDR           (*(unsigned char*)0xfffc5)
13 :  #define P4DR            (*(unsigned char*)0xfffc7)
14 :  #define P5DDR           (*(unsigned char*)0xfffc8)
15 :  #define P5DR            (*(unsigned char*)0xfffca)
16 :  #define P6DDR           (*(unsigned char*)0xfffc9)
17 :  #define P6DR            (*(unsigned char*)0xfffcb)
18 :  #define P7DR            (*(unsigned char*)0xfffce)
19 :  #define P8DDR           (*(unsigned char*)0xfffcd)
20 :  #define P8DR            (*(unsigned char*)0xfffcf)
21 :  #define P9DDR           (*(unsigned char*)0xfffd0)
22 :  #define P9DR            (*(unsigned char*)0xfffd2)
23 :  #define PADDR           (*(unsigned char*)0xfffd1)
24 :  #define PADR            (*(unsigned char*)0xfffd3)
25 :  #define PBDDR           (*(unsigned char*)0xfffd4)
26 :  #define PBDR            (*(unsigned char*)0xfffd6)
27 :  #define P2PCR           (*(unsigned char*)0xfffd8)
28 :  #define P4PCR           (*(unsigned char*)0xfffda)
29 :  #define P5PCR           (*(unsigned char*)0xfffdb)
30 :
31 :  #define ITU_STR         (*(unsigned char*)0xfff60)
32 :  #define ITU_SNC         (*(unsigned char*)0xfff61)
33 :  #define ITU_MDR         (*(unsigned char*)0xfff62)
34 :  #define ITU_FCR         (*(unsigned char*)0xfff63)
35 :  #define ITU_TOER        (*(unsigned char*)0xfff90)
36 :  #define ITU_TOCR        (*(unsigned char*)0xfff91)
37 :
38 :  #define ITU0_CNT        (*(unsigned int *)0xfff68)
39 :  #define ITU0_TCR        (*(unsigned char*)0xfff64)
40 :  #define ITU0_GRA        (*(unsigned int *)0xfff6a)
41 :  #define ITU0_GRB        (*(unsigned int *)0xfff6c)
42 :  #define ITU0_IER        (*(unsigned char*)0xfff66)
43 :  #define ITU0_TSR        (*(unsigned char*)0xfff67)
44 :  #define ITU0_TIOR       (*(unsigned char*)0xfff65)
45 :
46 :  #define ITU1_CNT        (*(unsigned int *)0xfff72)
47 :  #define ITU1_TCR        (*(unsigned char*)0xfff6e)
48 :  #define ITU1_GRA        (*(unsigned int *)0xfff74)
49 :  #define ITU1_GRB        (*(unsigned int *)0xfff76)
50 :  #define ITU1_IER        (*(unsigned char*)0xfff70)
51 :  #define ITU1_TSR        (*(unsigned char*)0xfff71)
52 :  #define ITU1_TIOR       (*(unsigned char*)0xfff6f)
53 :
54 :  #define ITU2_CNT        (*(unsigned int *)0xfff7c)
55 :  #define ITU2_TCR        (*(unsigned char*)0xfff78)
56 :  #define ITU2_GRA        (*(unsigned int *)0xfff7e)
57 :  #define ITU2_GRB        (*(unsigned int *)0xfff80)
58 :  #define ITU2_IER        (*(unsigned char*)0xfff7a)
59 :  #define ITU2_TSR        (*(unsigned char*)0xfff7b)
60 :  #define ITU2_TIOR       (*(unsigned char*)0xfff79)
61 :
62 :  #define ITU3_CNT        (*(unsigned int *)0xfff86)
63 :  #define ITU3_TCR        (*(unsigned char*)0xfff82)
64 :  #define ITU3_GRA        (*(unsigned int *)0xfff88)
65 :  #define ITU3_GRB        (*(unsigned int *)0xfff8a)
66 :  #define ITU3_BRA        (*(unsigned int *)0xfff8c)
67 :  #define ITU3_BRB        (*(unsigned int *)0xfff8e)
68 :  #define ITU3_IER        (*(unsigned char*)0xfff84)
69 :  #define ITU3_TSR        (*(unsigned char*)0xfff85)
70 :  #define ITU3_TIOR       (*(unsigned char*)0xfff83)
71 :
72 :  #define ITU4_CNT        (*(unsigned int *)0xfff96)
73 :  #define ITU4_TCR        (*(unsigned char*)0xfff92)
74 :  #define ITU4_GRA        (*(unsigned int *)0xfff98)
75 :  #define ITU4_GRB        (*(unsigned int *)0xfff9a)
76 :  #define ITU4_BRA        (*(unsigned int *)0xfff9c)
```

```
 77 :  #define ITU4_BRB        (*(unsigned int *)0xfff9e)
 78 :  #define ITU4_IER        (*(unsigned char*)0xfff94)
 79 :  #define ITU4_TSR        (*(unsigned char*)0xfff95)
 80 :  #define ITU4_TIOR       (*(unsigned char*)0xfff93)
 81 :
 82 :  #define SCI0_SMR        (*(unsigned char*)0xfffb0)
 83 :  #define SCI0_BRR        (*(unsigned char*)0xfffb1)
 84 :  #define SCI0_SCR        (*(unsigned char*)0xfffb2)
 85 :  #define SCI0_TDR        (*(unsigned char*)0xfffb3)
 86 :  #define SCI0_SSR        (*(unsigned char*)0xfffb4)
 87 :  #define SCI0_RDR        (*(unsigned char*)0xfffb5)
 88 :  #define SCI0_SCMR       (*(unsigned char*)0xfffb6)
 89 :
 90 :  #define SCI1_SMR        (*(unsigned char*)0xfffb8)
 91 :  #define SCI1_BRR        (*(unsigned char*)0xfffb9)
 92 :  #define SCI1_SCR        (*(unsigned char*)0xfffba)
 93 :  #define SCI1_TDR        (*(unsigned char*)0xfffbb)
 94 :  #define SCI1_SSR        (*(unsigned char*)0xfffbc)
 95 :  #define SCI1_RDR        (*(unsigned char*)0xfffbd)
 96 :
 97 :  #define AD_DRA          (*(unsigned int *)0xfffe0)
 98 :  #define AD_DRB          (*(unsigned int *)0xfffe2)
 99 :  #define AD_DRC          (*(unsigned int *)0xfffe4)
100 :  #define AD_DRD          (*(unsigned int *)0xfffe6)
101 :  #define AD_CSR          (*(unsigned char*)0xfffe8)
102 :  #define AD_CR           (*(unsigned char*)0xfffe9)
```

以下、略

「#define」は、定義するという意味で、「**ある文字列を他の文字列に置き換える**」という意味です。
例えば、

```
#define         PADDR      (*(unsigned char*)0xfffd1)
```

は、「PADDR」という文字列が出てくると、「(*(unsigned char*)0xfffd1)」という文字列に置き換えなさい、という意味になります。プログラム中で、

```
        PADDR = 0xff;
```

は、コンパイラは、

```
        (*(unsigned char*)0xfffd1) = 0xff;
```

と変換して実行しています。0xfffd1 番地に 0xff を書き込みなさいという意味です。どういうことでしょうか。メモリのアドレス構成は下記のようになっています。

・フラッシュ ROM は、0x00000～0x1ffff 番地にあります。例えば、0x100 番地を読み込むと、0x100 番地にあるデータが読み込まれます。
・RAM は、0xfef10～0xfff0f 番地にあります。例えば、0xfef12 番地に 0x55 を書き込むと 0x55 が保存されます。0xfef12 番地からデータを読み込むと、0x55 が読み込まれます。
・I/O レジスタは、0xfff1c～0xfffff 番地にあります。例えば、0xfffd3 番地に 0xaa を書き込むと、ポート A から「1010 1010」が出力されます(ポート A が出力設定の場合)。

このように、ROM も RAM も I/O レジスタもメモリ上にあり、アドレスによってどの場所を操作するか決まっています。このような方式を**「メモリマップド I/O」**と呼びます。
I/O レジスタと呼ばれる領域は、H8 マイコン内にある内蔵周辺機能を制御するための領域です。
例えば、0xfffd3 番地は PADR と決められています。0xfff1c～0xfffff 番地には決められた名称があります。以下に、0xfffc0～0xfffdb 番地の I/O レジスタの名称を示します。

アドレス	I/O レジスタ名	アドレス	I/O レジスタ名
0xfffc0	P1DDR	0xfffce	P7DR
0xfffc2	P1DR	0xfffcd	P8DDR
0xfffc1	P2DDR	0xfffcf	P8DR
0xfffc3	P2DR	0xfffd0	P9DDR
0xfffc4	P3DDR	0xfffd2	P9DR
0xfffc6	P3DR	0xfffd1	PADDR
0xfffc5	P4DDR	0xfffd3	PADR
0xfffc7	P4DR	0xfffd4	PBDDR
0xfffc8	P5DDR	0xfffd6	PBDR
0xfffca	P5DR	0xfffd8	P2PCR
0xfffc9	P6DDR	0xfffda	P4PCR
0xfffcb	P6DR	0xfffdb	P5PCR

アドレスで設定してみます。
0xfffd0 番地に 0xf7 を設定
0xfffd1 番地に 0xff を設定
0xfffd6 番地に 0xc0 を設定
0xfffd4 番地に 0xfe を設定
何というレジスタに値を設定しているか、分かりますか？分からないと思います。アドレスだけでは、語呂合わせでもしない限り、全く意味が分かりません。そこで、h8_3048.h で I/O レジスタのアドレスとその意味を定義してみました。「h8_3048.h」をインクルードすることにより、
P9DDR に 0xf7 を設定
PADDR に 0xff を設定
PBDR に 0xc0 を設定
PBDDR に 0xfe を設定
と言い換えて記述することができます。これなら、何というI/O レジスタに値を設定しているか一目瞭然です。
このように、直接番地で指定していたら訳が分からなくなってしまうため、h8_3048.h で「PADR とは 0xfffd3 番地のことです」と定義している訳です。

「#define PADR (*(unsigned char*)0xfffd3)」を分解してみると、

#define PADR 0xfffd3	PADR は、0xfffd3 という値ですよ、という意味になります。
↓	
#define PADR *0xfffd3	PADR は、0xfffd3 番地ですよ、という意味になります。
↓	
#define PADR (unsigned char*)0xfffd3	PADR は、符号無し８bit 幅(unsigned char)の 0xfffd3 番地ですよ、という意味になります。
↓	
#define PADR *(unsigned char*)0xfffd3	PADR は、符号無し８bit 幅の 0xfffd3 番地の値ですよ、という意味になります。
↓	
#define PADR (*(unsigned char*)0xfffd3)	同じですが、全体をカッコでくくります。

最後に、全体をカッコでくくっています。これは、なぜでしょう。
例えば、

```
        PADR++;
```

とすると

```
        (*(unsigned char*)0xfffd3)++;
```

と変換されます。これは、0xfffd3 番地の値を+1 しなさい、という意味になります。
しかし、すべてをカッコでくくっていないと、

```
        *(unsigned char*)0xfffd3++;
```

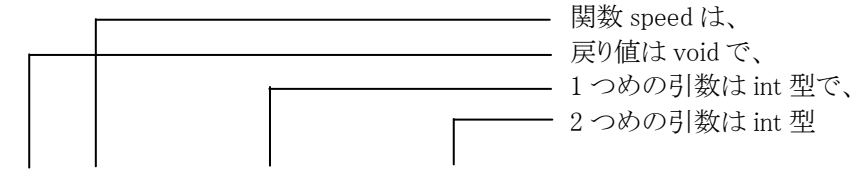
となり、0xfffd3 番地を＋1しなさい、という意味となります。0xfffd3 番地は固定値なので変更はできません。エラーとなります。
このように、計算の優先順位によっては、どの部分に作用するか分からないため、define 定義した内容は全体をカッコでくくるのが慣例です。

h8_3048.h では、マイコンカー制御用に使っている I/O レジスタの他に、使わない I/O レジスタもすべて網羅しています。詳しくは、「H/3048 シリーズ、H8/3048F-ZTAT™ (H8/3048F、H8/3048F-ONE)ハードウェアマニュアル」でレジスタの意味を調べてみてください。

5.9.4 プロトタイプ宣言

```
16 : /*====================================*/
17 : /* プロトタイプ宣言                   */
18 : /*====================================*/
19 : void init( void );
```

プロトタイプ宣言とは、自作した関数の引数の型と個数をチェックするために、関数を使用する前に宣言することです。関数プロトタイプは、関数に「;」を付加したものです。例えば、speed 関数を作ったとします。関数プロトタイプは、次のような情報を宣言しています。



先で説明したとおりプロトタイプ宣言は、「関数の引数の型と個数をチェックするため」に行います。そのため、仮引数名は不要です。下記のような記述でも問題ありません。

```
void speed( int , int );
```

しかし、仮引数名はプログラムを読むときの解読に役立ちます。書いておいたほうが親切です。また、わざわざ

仮引数名を削除して手を加えるより、間違いを無くすという意味で「**関数宣言の行をそのままコピーしてセミコロンを追加する**」と覚えておいて問題有りません。

当然、speed 関数は、プロトタイプ宣言と同じ引数、戻り値を持っていなければいけません。

```
void speed( int accele_l, int accele_r )
{
        プログラム;
}
```

このように、プロトタイプ宣言では、自作したすべての関数を記述して、使用する関数を宣言します。

プロトタイプ宣言をせずに関数を呼び出すと、コンパイラはその関数の引数や戻り値が正しいか分からないため、エラーであるのにエラーだと分からずおかしな動作をしてしまいます。そのため、関数の名前、引数、戻り値だけを先に記述して、こういう関数がありますよとコンパイラに宣言します(下図)。

```
void main( void )
{
    int i;
    i = 10;
    test( i );
}

void test( int *a )
{
    printf( "%d¥n",*a );  ←誤動作
}
```

エラーなのにコンパイルされてしまう

```
void test( int *a ); ←プロトタイプ宣言
void main( void )
{
    int i;
    i = 10;
    test( i );
}

void test( int *a )
{
    printf( "%d¥n",*a );
}
```

エラーと出力される！

ちなみに正しくは「test(&i);」です。

5.9.5 init関数(I/Oポートの入出力設定)

この関数で、H8/3048F-ONE 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。今回の演習では、init 関数内で I/O ポートの入出力設定を行っています。プログラムは下記のようになります。

```
39 : void init( void )
40 : {
41 :     /* ポートの入出力設定 */
42 :     P1DDR = 0xff;
43 :     P2DDR = 0xff;
44 :     P3DDR = 0xff;
45 :     P4DDR = 0xff;
46 :     P5DDR = 0xff;
47 :     P6DDR = 0xf0;                    /* CPU 基板上の DIP SW        */
48 :     P8DDR = 0xff;
49 :     P9DDR = 0xf7;
50 :     PADDR = 0xff;                    /* LED 基板                   */
51 :     PBDDR = 0xff;
52 :     /* ポート7は、入力専用なので入出力設定はありません    */
53 : }
```

(1) I/Oポートとは

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

I/O ポートは 1～B まで 11 個あり、基本的には 8bit 単位ですが、8bit 以下のポートもあります（「1.3 使用できる I/O 数」参照）。**I/O ポートには入出力方向の決定を行う P〇DDR というレジスタと、データの入力や出力を行う P〇DR というレジスタの、2 組で構成されています。〇には、1～B のポート番号が入ります。**

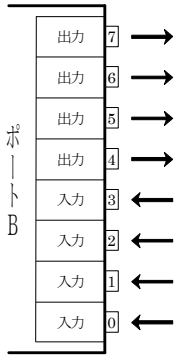
例）ポート 6 の入出力方向を決めるレジスタ…P6DDR
　　ポート A のデータの入力や出力を行うレジスタ…PADR

(2) 入出力設定とDDRレジスタ

各ポートの入出力設定は、P〇DDR というレジスタで行います。DDR は、「Data Direction Register」の略称です。〇には 1～B のポート番号が入ります。ポート 7 のみ入力専用ポートとなっており、P7DDR はありません。必ず入力ポートとなります。

入出力設定は、1 端子ごとに設定することができます。

例えば、ポート B を下記のような入出力設定にしたいとします。例題ですので io.c とは関係ありません。



まとめると、下表のようになります。

bit	7	6	5	4	3	2	1	0
ポート B の入出力方向	出力	出力	出力	出力	入力	入力	入力	入力

ポート B の入出力設定は、PBDDR に値を設定します。PBDDR に設定する値は、

●出力端子にしたいビットを"1"に設定
●入力端子にしたいビットを"0"に設定

します。結果、下表のようになります。

bit	7	6	5	4	3	2	1	0
PBDDR	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**

PBDDR には、2 進数で「1111　0000」を設定すれば良いことになります。ただし、C 言語は 2 進数表記はできないので C 言語で表記できる 10 進数か 16 進数に変換します。2 進数を 16 進数に変換するほうが簡単なため、通常は 16 進数で表記します。プログラムは、

```
    PBDDR = 0xf0;
```

となります。今回はポートＢを例に説明しましたが、他のポートも同様の考え方です。

io.c に戻ります。ポートAは LED 出力なので、すべて出力です。ポート 7 のみ、入力専用のポートで入出力設定はありません。必ず入力です。

```
50 :        PADDR = 0xff;                    /* LED 基板                    */
```

その他のポートはどう設定すれば良いのでしょうか。下表のように設定しています。

ポート	設定値	
P1DDR	0xff	未接続
P2DDR	0xff	未接続
P3DDR	0xff	未接続
P4DDR	0xff	未接続
P5DDR	0xff	未接続
P6DDR	0xf0	bit3～0 は CPU ボード上のディップスイッチ入力、他は未接続
P7		スイッチ部が接続
P8DDR	0xff	未接続
P9DDR	0xf7	bit3 は通信データ入力、bit1 は通信データ出力、他は未接続
PADDR	0xff	LED 部が接続
PBDDR	0xff	未接続

未接続ビットは出力に設定します。

なぜ何も接続されていないビットは出力にするのでしょうか。何も接続されていない状態で入力設定にすると、外部からの雑音(ノイズ)が端子に入ってきて、最悪の場合には H8 マイコンの内部回路が壊れてしまいます。**何も接続されていない端子は、プルアップかプルダウンして入力端子とするか、何も接続せずに出力設定とします。ポート7は常に入力のため、接続しない場合は必ず、プルアップかプルダウンしなければいけません。**

5.9.6 I/Oポートにデータを出力する、読み込む

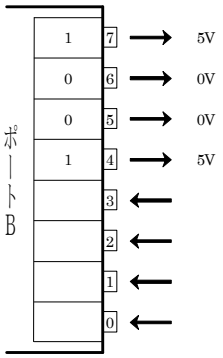
先ほどは、P○DDR を設定して端子の入出力方向を決めました。次は、入力に設定した端子からデータを入力したり、出力に設定した端子へデータを出力したりしてみましょう。

では、実際はどうするのでしょう。ここではポート B を例として、先ほど PBDDR=0xf0 を設定した状態で説明します。これは例ですので、io.c とは関係有りません。

(1) データの入出力とDRレジスタ

各ポートの端子にデータを入力したり出力するには、**P○DR** というレジスタで行います。DR は、「**Data Register**」の略称です。○には 1～B のポート番号が入ります。

(2) ポートBからデータを出力する



左図のようにデータを出力させたいとします。bit3～0 は、入力に設定しているため、今回は関係ありません。

表にまとめると、下表のようになります。

bit	7	6	5	4	3	2	1	0
ポートBの出力値	5V	0V	0V	5V				

ポート B にデータを出力するには、PBDR に値を設定します。PBDR に設定する値は

●5V を出力したければ"1"を設定する
●0V を出力したければ"0"を設定する

結果、下表のようになります。

bit	7	6	5	4	3	2	1	0
PBDR	1	0	0	1	入力端子	入力端子	入力端子	入力端子

PBDR への設定は、出力端子である bit7～4 だけで良いのですが、特定のビットだけ設定することはできず 8bit 全て行う必要があります。そのため、入力端子のビットにも値を設定します。入力端子は"0"を設定しても"1"を設定しても何も変化しません。そのため"0"でも"1"でも良いのですが、"1"にすると何か意味があるのかと思われるため、"0"にしておきます。結果、下表のようになります。

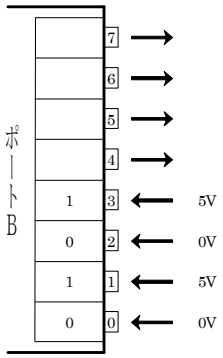
bit	7	6	5	4	3	2	1	0
PBDR	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**

PBDR には、2 進数で「1001　0000」、16 進数に直して 0x90 を設定します。プログラムは、

```
PBDR = 0x90;
```

となります。

(3) PBDRからデータを入力する



左図のようにデータが入力されているとします。bit7～4 は、出力に設定しているため、今回は関係ありません。

まとめると、下表のようになります。

bit	7	6	5	4	3	2	1	0
ポートB の入力電圧					5V	0V	5V	0V

ポートB のデータを読み込むには、PBDR の値を読みます。PBDR を読み込むと

●5V が入力されていれば"1"が読み込まれる
●0V が入力されていれば"0"が読み込まれる

結果、下表のようになります。

bit	7	6	5	4	3	2	1	0
PBDR を読み込んだ値	出力端子	出力端子	出力端子	出力端子	1	0	1	0

PBDR からの読み込みは、入力端子である bit3～0 だけで良いのですが、特定のビットだけ読み込むことはできず 8bit 全て読み込まれます。そのため、出力端子のビットの値も読み込まれます。出力端子は、現在出力している値が読み込まれます。現在、「0x90」を出力しているとすると、下表のようになります。

bit	7	6	5	4	3	2	1	0
PBDR	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**0**

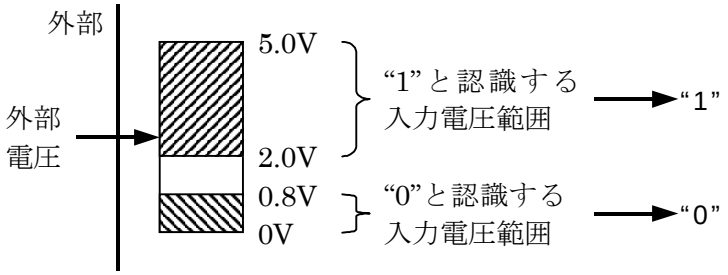
PBDR には、2 進数で「1001　1010」、16 進数に直して 0x9a が読み込まれます。プログラムを次のようにすると、変数 c には、0x9a が代入されます。

```
c = PBDR;
```

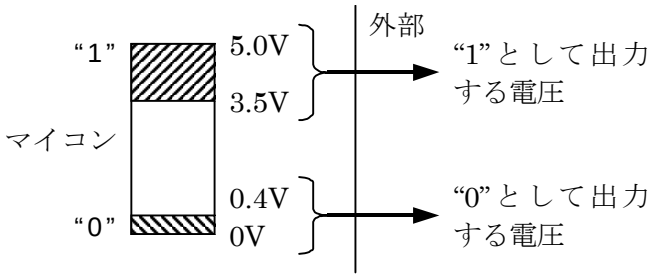
今回はポートBを例に説明しましたが、他のポートも同様の考え方です。

(4) 電圧レベルについて

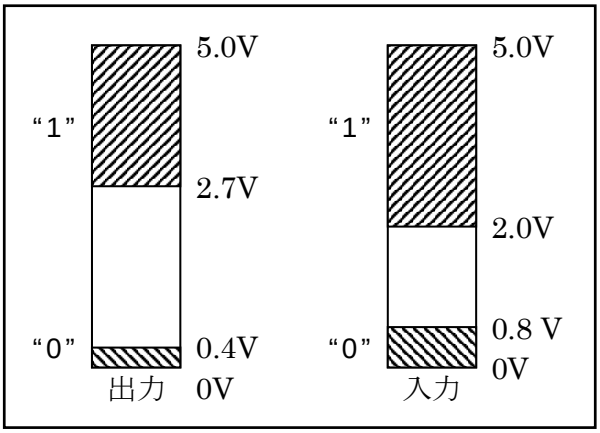
I/O ポートを入力用に設定した場合は、外部からの電圧信号を読み込むことができます。デジタル値で"1"(ハイレベル)とみなすのが 2.0V 以上の電圧、"0"(ローレベル)と見なすのが 0.8V 以下の電圧の場合です。どちらの範囲にもない 0.8～2.0V の電圧を入力した場合は、どちらのレベルになるか分かりません。このような電圧はマイコン(プログラム)の誤動作の元となりますので必ず範囲内の電圧を出力する回路にします。



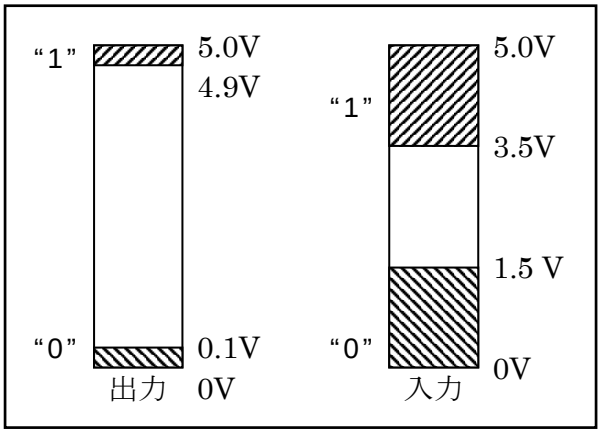
出力用に設定した場合は、"1"(ハイレベル)を書き込んだビットは 3.5V 以上の電圧、"0"(ローレベル)を書き込んだビットは 0.4V 以下の電圧が端子から出力されます。



ちなみに、TTLレベル(74LSシリーズのICなど)と C-MOS レベル(74HCシリーズのICなど)の電圧レベルは下図のようになっています。マイコンの I/O ポートの電圧レベルはどちらでもありませんので注意が必要です。



TTL レベル



CMOS レベル

(5) 出力電流について

H8/3048F-ONE の I/O 出力許容電流は、下記のようになっています。

項目		許容電流
出力 Low レベル許容電流(1端子あたり)	ポート 1、2、5、B	10mA
	上記以外の出力端子	2.0mA
出力 Low レベル許容電流(総和)	ポート 1、2、5、B、28 端子の総和	80mA
	上記を含む、全出力端子の総和	120mA
出力 High レベル許容電流(1端子あたり)	全出力端子	2.0mA
出力 High レベル許容電流(総和)	全出力端子の総和	40mA

※条件　Vcc=3.0～3.6V、AVcc=3.0～3.6V、V_{REF}=3.0～AVcc、Vss=AVss=0V または、
Vcc=5.0±10%、AVcc=5.0±10%、V_{REF}=4.5～AVcc、Vss=AVss=0V

ダーリントントランジスタや LED を直接駆動する場合、出力に必ず電流制限抵抗を挿入してください。

ポート B に Low レベル信号を出力するとき、1端子あたり 10mA の電流を流せます。8 ビットすべて 10mA 流すとポート B だけで 80mA の電流を消費します。「ポート 1、2、5、B の出力 Low レベル許容電流(総和)は 80mA」ですので、ポート 1、2、5 には電流を流すことができません。

ポート 1、2、5、B の Low レベル許容電流は、1 端子あたり 10mA ですが、High レベル許容電流は 2.0mA と High レベルのほうが電流を流せません。回路設計時には注意が必要です。

※参考資料－10 進数、16 進数、2 進数、出力について

LED の出力は、ON か OFF かですので、2 進数と同じです。2 進数の 1 が ON(LED なら点灯)、0 が OFF(LED なら消灯)です。C言語では、残念ながら 2 進数表記はできないため、10 進数か 16 進数で記述することになります。どちらで記述してもいいのですが、2 進数を 10 進数に変換するより、2 進数を 16 進数に変換する方が簡単なため、通常 16 進数でプログラムします。

10 進数	16 進数	2 進数	出力パターン ○＝ON、●＝OFF
0	0	0000	●●●●
1	1	0001	●●●○
2	2	0010	●●○●
3	3	0011	●●○○
4	4	0100	●○●●
5	5	0101	●○●○
6	6	0110	●○○●
7	7	0111	●○○○
8	8	1000	○●●●
9	9	1001	○●●○
10	a	1010	○●○●
11	b	1011	○●○○
12	c	1100	○○●●
13	d	1101	○○●○
14	e	1110	○○○●
15	f	1111	○○○○

例えば、ポート B の bit7～0 を ON,OFF,ON,OFF,OFF,ON,OFF,ON と出力したい場合、まずは 2 進数に変換します。

ON,OFF,ON,OFF,OFF,ON,OFF,ON　→　10100101

次に、4 桁ずつ区切ります。

10100101　→　1010　0101

次に、上の表より、4 桁の 2 進数を 16 進数に変換します。今回は「1010」と「0101」の 2 つ有りますので、変換も 2 回行います。

1010　0101
↓　↓
a　5

16 進数の前には「0x (ゼロ、エックス)」を付けますので、

0xa5

となります。このようにして、出力したいパターンを 16 進数に変換して最終的に下記のようにするとポート B から「ON,OFF,ON,OFF,OFF,ON,OFF,ON」の信号が出力されます。

```
PBDR = 0xa5;
```

5.9.7 main関数

```
24 : void main( void )
25 : {
26 :     unsigned char d;
27 :
28 :     init();                              /* 初期化                    */
29 :
30 :     while( 1 ) {
31 :         d = P7DR;
32 :         PADR = d;
33 :     }
34 : }
```

28 行で、内蔵周辺機能の初期化をする init 関数を呼んでいます。

31 行で、ポート 7 の状態を読み込み変数dへ代入、変数dの値をポート A へ出力します。ポートの値を扱う変数は、符号無し 8bit 幅の unsigned char 型とします。

ポート7は全ビット入力、ポート A は全ビット出力なので、ポート7の状態がそのままポート A へ出力されます。

30 行目の「while(1)」は、while 文のカッコ内が真なら、「 { } 」の中を繰り返すという意味で、1は常に真なので無限ループです。

5.10 ビット操作のプログラムテクニック

5.10.1 全ビット反転する

変数やレジスタの先頭に「~」を付けると、レベルが反転します。例えば、P7DR が 0x55(2 進数で 0101 0101)なら、変数 d には、0xaa(2 進数で 1010 1010)が代入されます。これは、ON で"0"、OFF で"1"となっているスイッチをつないでいる場合などに便利です。

「~」は、チルダと読み、キーボードの¥キー左横の^キーを、シフトを押しながら押すと~になります。

```
30 :     while( 1 ) {
31 :         d = ~P7DR;
32 :         PADR = d;
33 :     }
```

5.10.2 特定のビットを"0"にする

例えば、ポート7の bit3,2,1,0 を"0"にして値を読み込みたいとします。その場合、AND 演算を使います。現在、P7DR は 0x55 とします。

"0"にしたいビットを"0"、そのままにしたいビットを"1"とした値で AND 演算します。

「&」は、そのままアンドと読みます。シフトを押しながら6キーを押すと&になります。

bit	7	6	5	4	3	2	1	0	
P7DR	0	1	0	1	0	1	0	1	
AND する値	1	1	1	1	0	0	0	0	→0xf0
d	0	1	0	1	0	0	0	0	

```
30 :     while( 1 ) {
31 :         d = P7DR & 0xf0;
32 :         PADR = d;
33 :     }
```

5.10.3 マスク処理

マスクとは、「覆う」ことです。チェックに不要なビットを覆って”0”にする、それがマスク処理です。マスクは制御で非常に重要です。マイコンカー制御でも頻繁に使用します。ここで詳しく説明しておきます。

1 ポートの単位は8ビットのため、**1ビット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8ビットまとめてのチェックとなります**。
例えば、センサの左端である bit7 が“1”かどうかチェックしたい場合、

```
if( センサの値==0x80 ) {
    /* ビット７が “1” ならこの中を実行 */
}
```

とすればいいように思えます。しかし、bit6～0 がどのような値になっているか分かりません。例えば、bit7 が“1”、bit0 も”1”なら
センサ値＝10000001（2進数）＝0x81（16進数）
となります。プログラムで0x80かどうかチェックしただけでは bit7 が”1”かどうか判断できません。これでは、うまくチェックできないので、「マスク」という作業をします。マスクとは『覆い隠す』という意味になります。

では、どのようにマスクするのでしょうか。実際の制御では、強制的に”0”にするだけのことです。
プログラムでは、論理演算の論理積、すなわち AND 演算を行います。AND 演算とは、2つの変数 A とBがあるとき（それぞれ0か1の数値）、ともに1であるときのみ1になる演算を言います。Aをセンサの値として考えてみます。

A（センサ値）	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

ここで、Bが0のときに注目します。

A（センサ値）	**B**	A AND B
0	**0**	0
1	**0**	0

Bが0ならA（センサ値）がどの値でも結果は必ず0になります。次に、Bが1のときに注目します。

A（センサ値）	**B**	A AND B
0	**1**	0
1	**1**	1

Bが1なら、結果はA（センサ値）の値そのものとなります。
実際に置き換えると、Aがセンサの値、Bがマスク値にあたります。マスク値は、必要なビットは”1”に、必要のないビットは”0”にします。そして AND 演算を行うと不必要なビットは必ず”0”になるので、プログラムでは必要ないビットは”0”ということを前提にして作成することができます。

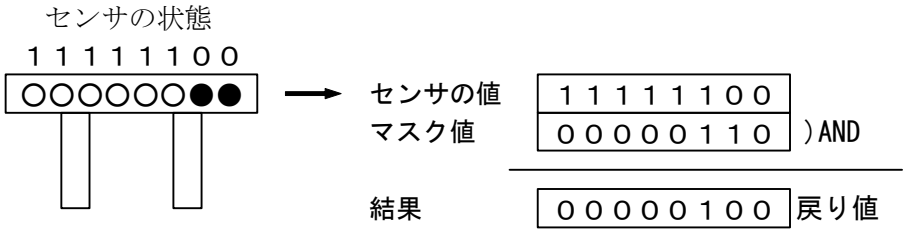
このように、マスクとは AND 処理して不要なビットを強制的に"0"にすることです。
例えば、センサの状態が「白白白白白白黒黒」のとき、センサ値は「11111100」です。bit2,1 のみチェックに必要で、他は必要なしとします。表にまとめます。

bit	7	6	5	4	3	2	1	0
	不要	不要	不要	不要	不要	**必要**	**必要**	不要

不要な部分を0にするためには、不要ビットのマスク値を"0"にして AND 演算を行います。したがって、マスク値は上表の不要部分を"0"に、必要部分を"1"に書き換えれば良いことになります。

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

2進数で「0000 0110」、16進数に直すと「0x06」となります。下図はその計算方法と結果です。



例えば、bit2＝"1"、bit1＝"0"かどうかチェックしたい場合、チェックする値は 16 進数で 0x04 となります。よって次のようなプログラムでチェックできます。

```
                マスク値   チェック値
                   ↓         ↓
if( (センサの値 & 0x06) == 0x04 ) {
    /* bit2＝" 1"、bit1＝" 0" ならこの中を実行 */
}
```

bit2,1 以外はマスクによって強制的に"0"になっていることが分かっているので、安心して bit2,1 のみ調べることができます。

まとめると、

・1ポートの単位は8ビットのため、特定のビットだけチェックはできない
・そのため、チェックに必要のないビットを強制的に"0"にする(これをマスク処理という)
・チェックに必要のないビットは"0"として、チェックしたいビットを調べる

となります。

5.10.4 特定のビットを”1”にする

例えば、ポート7の bit7,6,1,0 を"1"にして値を読み込みたいとします。その場合、OR 演算を使います。現在、P7DR は 0x55 とします。

"1"にしたいビットを"1"、そのままにしたいビットを"0"とした値で OR 演算します。

「｜」は、パイプ、または縦線と読みます。プログラム中ではそのまま「オア」と呼ぶのか慣例です。シフトを押しながら¥キーを押すと｜になります。

bit	7	6	5	4	3	2	1	0	
P7DR	0	1	0	1	0	1	0	1	
OR する値	1	1	0	0	0	0	1	1	→０ｘｃ３
d	1	1	0	1	0	1	1	1	

```
30 :      while( 1 ) {
31 :          d = P7DR | 0xc3;
32 :          PADR = d;
33 :      }
```

5.10.5 特定のビットを反転する

例えば、ポート7の bit7,5,3,1 を反転して値を読み込みたいとします。その場合、XOR 演算を使います。現在、P7DR は 0x55 とします。

反転したいビットを"1"、そのままにしたいビットを"0"とした値で XOR 演算します。

「^」は、アクサンシルコンフレックスと読みます。プログラム中ではそのまま「エクスクルーシブオア」と呼ぶのか慣例です。¥キーの左キーをそのまま押すと^になります。

bit	7	6	5	4	3	2	1	0	
P7DR	0	1	0	1	0	1	0	1	
XOR する値	1	0	1	0	1	0	1	0	→０ｘａａ
d	1	1	1	1	1	1	1	1	

```
30 :      while( 1 ) {
31 :          d = P7DR ^ 0xaa;
32 :          PADR = d;
33 :      }
```

6. プロジェクト「timer1」 タイマ(学校祭用電飾プログラムへの応用)

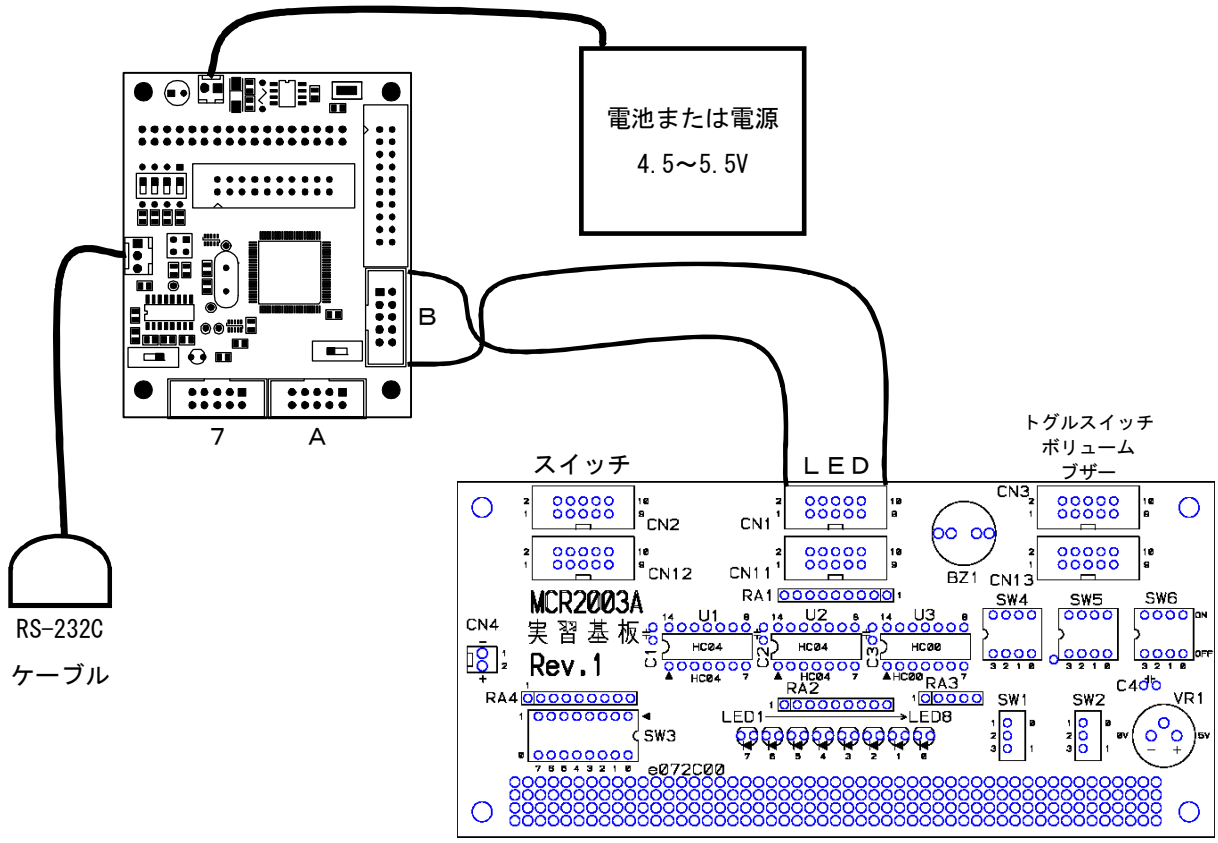
6.1 概要

LED を、0.5 秒ごとに点滅させます。マイコンのポートは、下記を使用します。
・ポート B の全ビット・・・LED へデータ出力

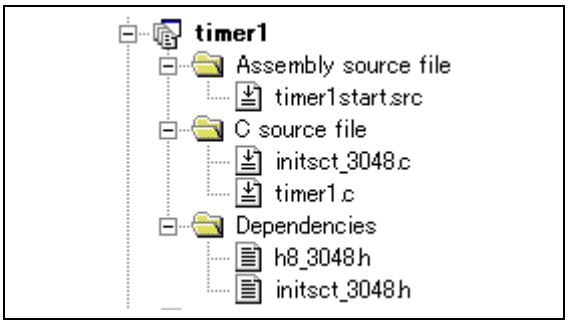
6.2 接続

・CPU ボードのポート B と、実習基板の LED 部をフラットケーブルで接続します。

今回、LED は 8 個のみですが、I/O ポートをすべて使えば 70 個の LED の制御が可能です(RY3048F-ONE ボードはディップスイッチと通信用に 7 ビット使用しているので 63 個になります)。様々な色を使えば目立たせることができます。学校祭の電飾用として最適です。



6.3 プロジェクトの構成



	ファイル名	内容
1	timer1start.src	アセンブリ言語で記述されたアセンブリソースファイルです。このファイルの構造は下記のようになっています。 ベクタアドレス ＋ スタートアップルーチン
2	initsct_3048.c	初期値のないグローバル変数(セクション B 領域)、初期値のあるグローバル変数(セクション R 領域)の初期化用です。
3	timer1.c	実際に制御するプログラムが書かれています。H8/3048F-ONE の内蔵周辺機能の初期化も行います。
4	h8_3048.h	H8/3048F-ONE の内蔵周辺機能の I/O レジスタを定義したファイルです。
5	initsct_3048.h	initsct_3048.c のヘッダファイルです。

6.4 プログラム「timer1.c」

```
 1 :  /************************************************************************/
 2 :  /* タイマ(プログラムループ)でＬＥＤ等を光らす(電飾への応用)「timer1.c」      */
 3 :  /* 出力：PB7-PB0(LED等)                                                 */
 4 :  /*                          2005.04 ジャパンマイコンカーラリー実行委員会  */
 5 :  /************************************************************************/
 6 :  /*======================================*/
 7 :  /* インクルード                         */
 8 :  /*======================================*/
 9 :  #include    <machine.h>
10 :  #include    "h8_3048.h"
11 :
12 :  /*======================================*/
13 :  /* シンボル定義                         */
14 :  /*======================================*/
15 :
16 :  /*======================================*/
17 :  /* プロトタイプ宣言                     */
18 :  /*======================================*/
19 :  void init( void );
20 :  void timer( unsigned long timer_set );
21 :
22 :  /************************************************************************/
23 :  /* メインプログラム                                                     */
24 :  /************************************************************************/
25 :  void main( void )
26 :  {
27 :      init();                             /* 初期化                       */
28 :
29 :      while( 1 ) {
30 :          PBDR = 0x55;
31 :          timer( 1000 );
32 :          PBDR = 0xaa;
33 :          timer( 1000 );
34 :          PBDR = 0x00;
35 :          timer( 1000 );
36 :      }
37 :  }
38 :
```

```
39 :  /************************************************************************/
40 :  /* H8/3048F-ONE 内蔵周辺機能の初期化                                     */
41 :  /************************************************************************/
42 :  void init( void )
43 :  {
44 :      /* ポートの入出力設定 */
45 :      P1DDR = 0xff;
46 :      P2DDR = 0xff;
47 :      P3DDR = 0xff;
48 :      P4DDR = 0xff;
49 :      P5DDR = 0xff;
50 :      P6DDR = 0xf0;                       /* CPU基板上のDIP SW        */
51 :      P8DDR = 0xff;
52 :      P9DDR = 0xf7;
53 :      PADDR = 0xff;
54 :      PBDDR = 0xff;                       /* LED基板                  */
55 :      /* ポート７は、入力専用なので入出力設定はありません    */
56 :  }
57 :
58 :  /************************************************************************/
59 :  /* タイマ本体                                                           */
60 :  /* 引数    タイマ値 1=1ms                                               */
61 :  /************************************************************************/
62 :  void timer( unsigned long timer_set )
63 :  {
64 :      unsigned long l, m;
65 :
66 :      for( l=0; l<timer_set; l++ ) {
67 :          for( m=0; m<6160; m++ );
68 :      }
69 :  }
70 :
71 :  /************************************************************************/
72 :  /* end of file                                                          */
73 :  /************************************************************************/
```

6.5 プログラムの解説

6.5.1 I/Oポートの入出力設定

```
42 : void init( void )
43 : {
44 :     /* ポートの入出力設定 */
45 :     P1DDR = 0xff;
46 :     P2DDR = 0xff;
47 :     P3DDR = 0xff;
48 :     P4DDR = 0xff;
49 :     P5DDR = 0xff;
50 :     P6DDR = 0xf0;                          /* CPU 基板上の DIP SW        */
51 :     P8DDR = 0xff;
52 :     P9DDR = 0xf7;
53 :     PADDR = 0xff;
54 :     PBDDR = 0xff;                          /* LED 基板                   */
55 :     /* ポート7は、入力専用なので入出力設定はありません    */
56 : }
```

ポートBにLED基板を接続しますので、ポートBは出力に設定します。開放ポートも出力に設定します。

6.5.2 timer関数

```
62 : void timer( unsigned long timer_set )
63 : {
64 :     unsigned long l, m;
65 :
66 :     for( l=0; l<timer_set; l++ ) {
67 :         for( m=0; m<6160; m++ );
68 :     }
69 : }
```

タイマ関数は、時間稼ぎをする関数です。

```
    timer( 時間稼ぎをする時間 [ms] );
```

カッコの中には、ミリ秒単位で値を設定します。例えば、10秒なら10000となります。
プログラムの1命令は、数百ナノ秒から数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す。例え短くとも、何十万回も繰り返すと秒単位の時間となります。ここでは for 文を使って、何もしないことを繰り返すことにより時間稼ぎをしています。

```
66 :     for( l=0; l<timer_set; l++ ) {
67 :         for( m=0; m<6160; m++ );
68 :     }
```

この3行が時間稼ぎをしている部分です。
67行では6160回、変数 m を足しています。そして6160以下かどうかチェックしています。ただこれだけで、

他は何もしていません。この足したりチェックしたりすることで時間がかかります。67 行1行で 1[ms]の時間稼ぎとなります。

1[ms]の時間稼ぎを、引数 timer_set の値だけさらに繰り返します。これが 66 行です。変数 l (エル) を 0 にして、timer_set と比較します。この timer_set が timer 関数の引数の値です。例えば、timer_set が 1000 なら

l (エル) < 1000

が成り立つまで 67 行を繰り返します。1000 回目、式が成り立たなくなり for 文を終了、timer 関数も終了します。

なぜ、6160 が 1[ms]だと分かったのでしょうか。実は、

```
timer( 10000 );
```

として、ストップウォッチで測定、ちょうど 10 秒間になった値が 6160 だったのです。この値は、
・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)
・クリスタルの値
によって違ってきますので、今回の条件固有の数値と覚えておくと良いでしょう。

6.5.3 main関数

```
25 : void main( void )
26 : {
27 :     init();                          /* 初期化                    */
28 :
29 :     while( 1 ) {
30 :         PBDR = 0x55;
31 :         timer( 1000 );
32 :         PBDR = 0xaa;
33 :         timer( 1000 );
34 :         PBDR = 0x00;
35 :         timer( 1000 );
36 :     }
37 : }
```

27 行目で、init 関数を呼んでポートの入出力設定を行います。
29 行目は、while 文がありカッコの中が常に真なので、対応するカッコ閉じである 36 行まで無限ループです。
30 行目で PB に 0x55 を出力、1000ミリ秒時間稼ぎします。
32 行目で PB に 0xaa を出力、1000ミリ秒時間稼ぎします。
34 行目で PB に 0x00 を出力、1000ミリ秒時間稼ぎします。

7. プロジェクト「timer2」 割り込みによるタイマ

7.1 概要

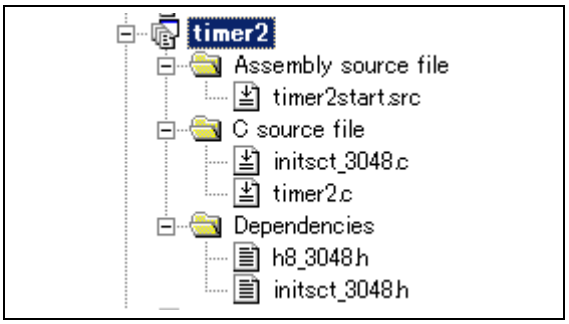
LED 8 個を 1 秒ごとに
●○●○●○●○ (16 進数で 0x55)　　※●＝LED 消灯　○＝LED 点灯
○●○●○●○● (16 進数で 0xaa)
●●●●●●●● (16 進数で 0x00)
を繰り返し出力し続けます。マイコンのポートは、下記を使用します。
・ポート B の全ビット・・・LED へデータ出力

7.2 接続

・CPU ボードのポート B と、実習基板の LED 部をフラットケーブルで接続します。

※timer1.c と同じです。

7.3 プロジェクトの構成



	ファイル名	内容
1	timer2start.src	アセンブリ言語で記述されたアセンブリソースファイルです。このファイルの構造は下記のようになっています。 ベクタアドレス ＋ スタートアップルーチン
2	initsct_3048.c	初期値のないグローバル変数(セクション B 領域)、初期値のあるグローバル変数(セクション R 領域)の初期化用です。
3	timer2.c	実際に制御するプログラムが書かれています。H8/3048F-ONE の内蔵周辺機能の初期化も行います。
4	h8_3048.h	H8/3048F-ONE の内蔵周辺機能の I/O レジスタを定義したファイルです。
5	initsct_3048.h	initsct_3048.c のヘッダファイルです。

7.4 プログラム「timer2.c」

```
 1 : /*********************************************************************/
 2 : /* タイマ(ITU0割り込み)でＬＥＤ等を光らす(電飾への応用)「timer2.c」          */
 3 : /* 出力：PB7-PB0(LED等)                                               */
 4 : /*                          2005.04 ジャパンマイコンカーラリー実行委員会  */
 5 : /*********************************************************************/
 6 : /*====================================*/
 7 : /* インクルード                       */
 8 : /*====================================*/
 9 : #include    <machine.h>
10 : #include    "h8_3048.h"
11 :
12 : /*====================================*/
13 : /* シンボル定義                       */
14 : /*====================================*/
15 :
16 : /*====================================*/
17 : /* プロトタイプ宣言                   */
18 : /*====================================*/
19 : void init( void );
20 : void timer( unsigned long timer_set );
21 :
22 : /*====================================*/
23 : /* グローバル変数の宣言               */
24 : /*====================================*/
25 : unsigned long   cnt0;                 /* ITU0用                   */
26 :
27 : /*********************************************************************/
28 : /* メインプログラム                                                  */
29 : /*********************************************************************/
30 : void main( void )
31 : {
32 :     init();                           /* 初期化                   */
33 :     set_ccr( 0x00 );                  /* 全体割り込み許可         */
34 :
35 :     while( 1 ) {
36 :         PBDR = 0x55;
37 :         timer( 1000 );
38 :         PBDR = 0xaa;
39 :         timer( 1000 );
40 :         PBDR = 0x00;
41 :         timer( 1000 );
42 :     }
43 : }
44 :
45 : /*********************************************************************/
46 : /* H8/3048F-ONE 内蔵周辺機能の初期化                                 */
47 : /*********************************************************************/
48 : void init( void )
49 : {
50 :     /* ポートの入出力設定 */
51 :     P1DDR = 0xff;
52 :     P2DDR = 0xff;
53 :     P3DDR = 0xff;
54 :     P4DDR = 0xff;
55 :     P5DDR = 0xff;
56 :     P6DDR = 0xf0;                     /* CPU基板上のDIP SW        */
57 :     P8DDR = 0xff;
58 :     P9DDR = 0xf7;
59 :     PADDR = 0xff;
60 :     PBDDR = 0xff;                     /* LED基板                  */
61 :     /* ポート７は、入力専用なので入出力設定はありません   */
62 :
63 :     /* ITU0 1ms毎の割り込み  timer関数用 */
64 :     ITU0_TCR = 0x20;
65 :     ITU0_GRA = 24575;
66 :     ITU0_IER = 0x01;
67 :     ITU_STR  = 0x01;
68 : }
69 :
70 : /*********************************************************************/
71 : /* タイマ本体                                                        */
72 : /* 引数   タイマ値 1=1ms                                             */
73 : /*********************************************************************/
74 : void timer( unsigned long timer_set )
75 : {
76 :     cnt0 = 0;
77 :     while( cnt0 < timer_set );
78 : }
79 :
80 : /*********************************************************************/
81 : /* ITU0 割り込み処理                                                 */
82 : /*********************************************************************/
83 : #pragma interrupt( interrupt_timer0 )
84 : void interrupt_timer0( void )
85 : {
```

```
86 :      ITU0_TSR &= 0xfe;                    /* フラグクリア             */
87 :      cnt0++;
88 :  }
89 :
90 :  /************************************************************************/
91 :  /* end of file                                                          */
92 :  /************************************************************************/
```

7.5 プログラム「timer2start.src」

ゴシック体が、timer2.c を使うために追加、変更した行です。

```
 1 :  ;=====================================
 2 :  ; 定義
 3 :  ;=====================================
 4 :  RESERVE: .EQU    H'FFFFFFFF             ; 未使用領域のアドレス
 5 :
 6 :  ;=====================================
 7 :  ; 外部参照
 8 :  ;=====================================
 9 :          .IMPORT _main
10 :          .IMPORT _interrupt_timer0
11 :          .IMPORT _INITSCT
12 :
13 :  ;=====================================
14 :  ; ベクタセクション
15 :  ;=====================================
16 :          .SECTION V
17 :          .DATA.L RESET_START            ;  0 H'000000   ﾘｾｯﾄ
18 :          .DATA.L RESERVE                ;  1 H'000004   ｼｽﾃﾑ予約
19 :          .DATA.L RESERVE                ;  2 H'000008   ｼｽﾃﾑ予約
20 :          .DATA.L RESERVE                ;  3 H'00000c   ｼｽﾃﾑ予約
21 :          .DATA.L RESERVE                ;  4 H'000010   ｼｽﾃﾑ予約
22 :          .DATA.L RESERVE                ;  5 H'000014   ｼｽﾃﾑ予約
23 :          .DATA.L RESERVE                ;  6 H'000018   ｼｽﾃﾑ予約
24 :          .DATA.L RESERVE                ;  7 H'00001c   外部割り込み NMI
25 :          .DATA.L RESERVE                ;  8 H'000020   ﾄﾗｯﾌﾟ命令
26 :          .DATA.L RESERVE                ;  9 H'000024   ﾄﾗｯﾌﾟ命令
27 :          .DATA.L RESERVE                ; 10 H'000028   ﾄﾗｯﾌﾟ命令
28 :          .DATA.L RESERVE                ; 11 H'00002c   ﾄﾗｯﾌﾟ命令
29 :          .DATA.L RESERVE                ; 12 H'000030   外部割り込み IRQ0
30 :          .DATA.L RESERVE                ; 13 H'000034   外部割り込み IRQ1
31 :          .DATA.L RESERVE                ; 14 H'000038   外部割り込み IRQ2
32 :          .DATA.L RESERVE                ; 15 H'00003c   外部割り込み IRQ3
33 :          .DATA.L RESERVE                ; 16 H'000040   外部割り込み IRQ4
34 :          .DATA.L RESERVE                ; 17 H'000044   外部割り込み IRQ5
35 :          .DATA.L RESERVE                ; 18 H'000048   ｼｽﾃﾑ予約
36 :          .DATA.L RESERVE                ; 19 H'00004c   ｼｽﾃﾑ予約
37 :          .DATA.L RESERVE                ; 20 H'000050   WDT MOVI
38 :          .DATA.L RESERVE                ; 21 H'000054   REF CMI
39 :          .DATA.L RESERVE                ; 22 H'000058   ｼｽﾃﾑ予約
40 :          .DATA.L RESERVE                ; 23 H'00005c   ｼｽﾃﾑ予約
41 :          .DATA.L _interrupt_timer0      ; 24 h'000060   ITU0 IMIA0
42 :          .DATA.L RESERVE                ; 25 H'000064   ITU0 IMIB0
43 :          .DATA.L RESERVE                ; 26 H'000068   ITU0 OVI0
44 :          .DATA.L RESERVE                ; 27 H'00006c   ｼｽﾃﾑ予約
45 :          .DATA.L RESERVE                ; 28 H'000070   ITU1 IMIA1
46 :          .DATA.L RESERVE                ; 29 H'000074   ITU1 IMIB1
47 :          .DATA.L RESERVE                ; 30 H'000078   ITU1 OVI1
48 :          .DATA.L RESERVE                ; 31 H'00007c   ｼｽﾃﾑ予約
49 :          .DATA.L RESERVE                ; 32 H'000080   ITU2 IMIA2
50 :          .DATA.L RESERVE                ; 33 H'000084   ITU2 IMIB2
51 :          .DATA.L RESERVE                ; 34 H'000088   ITU2 OVI2
52 :          .DATA.L RESERVE                ; 35 H'00008c   ｼｽﾃﾑ予約
53 :          .DATA.L RESERVE                ; 36 H'000090   ITU3 IMIA3
54 :          .DATA.L RESERVE                ; 37 H'000094   ITU3 IMIB3
55 :          .DATA.L RESERVE                ; 38 H'000098   ITU3 OVI3
56 :          .DATA.L RESERVE                ; 39 H'00009c   ｼｽﾃﾑ予約
57 :          .DATA.L RESERVE                ; 40 H'0000a0   ITU4 IMIA4
58 :          .DATA.L RESERVE                ; 41 H'0000a4   ITU4 IMIB4
59 :          .DATA.L RESERVE                ; 42 H'0000a8   ITU4 OVI4
60 :          .DATA.L RESERVE                ; 43 H'0000ac   ｼｽﾃﾑ予約
61 :          .DATA.L RESERVE                ; 44 H'0000b0   DMAC DEND0A
62 :          .DATA.L RESERVE                ; 45 H'0000b4   DMAC DEND0B
63 :          .DATA.L RESERVE                ; 46 H'0000b8   DMAC DEND1A
64 :          .DATA.L RESERVE                ; 47 H'0000bc   DMCA DEND1B
65 :          .DATA.L RESERVE                ; 48 H'0000c0   ｼｽﾃﾑ予約
66 :          .DATA.L RESERVE                ; 49 H'0000c4   ｼｽﾃﾑ予約
67 :          .DATA.L RESERVE                ; 50 H'0000c8   ｼｽﾃﾑ予約
68 :          .DATA.L RESERVE                ; 51 H'0000cc   ｼｽﾃﾑ予約
69 :          .DATA.L RESERVE                ; 52 H'0000d0   SCI0 ERI0
70 :          .DATA.L RESERVE                ; 53 H'0000d4   SCI0 RXI0
71 :          .DATA.L RESERVE                ; 54 H'0000d8   SCI0 TXI0
72 :          .DATA.L RESERVE                ; 55 H'0000dc   SCI0 TEI0
73 :          .DATA.L RESERVE                ; 56 H'0000e0   SCI1 ERI1
```

```
 74 :          .DATA.L RESERVE                 ; 57 H'0000e4   SCI1 RXI1
 75 :          .DATA.L RESERVE                 ; 58 H'0000e8   SCI1 TXI1
 76 :          .DATA.L RESERVE                 ; 59 H'0000ec   SCI1 TEI1
 77 :          .DATA.L RESERVE                 ; 60 H'0000f0   A/D ADI
 78 :
 79 :  ;====================================
 80 :  ; スタートアッププログラム
 81 :  ;====================================
 82 :          .SECTION P
 83 :  RESET_START:
 84 :          MOV.L   #H'FFF10,ER7            ; スタックの設定
 85 :          JSR     @_INITSCT               ; RAMエリアの初期化
 86 :          JSR     @_main                  ; Ｃ言語のmain()関数へジャンプ
 87 :  OWARI:
 88 :          BRA     OWARI
 89 :
 90 :          .END
```

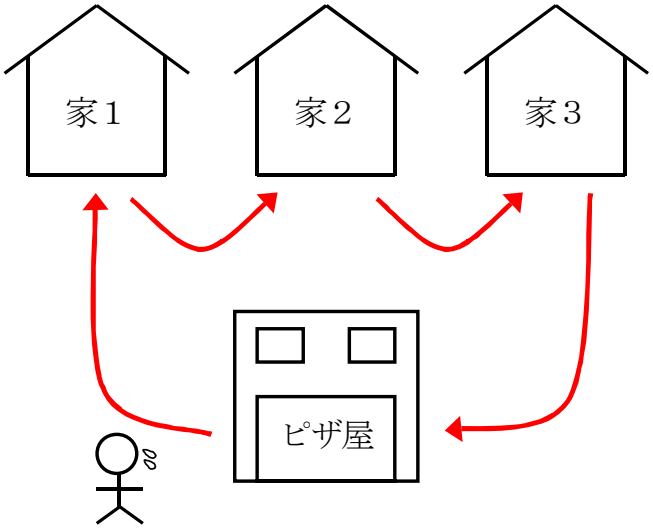
割り込みを使わないプログラムと比べて、ITU0 の割り込みに関する設定の 10 行目が追加、41 行目が変更されています。

7.6 割り込みの概要

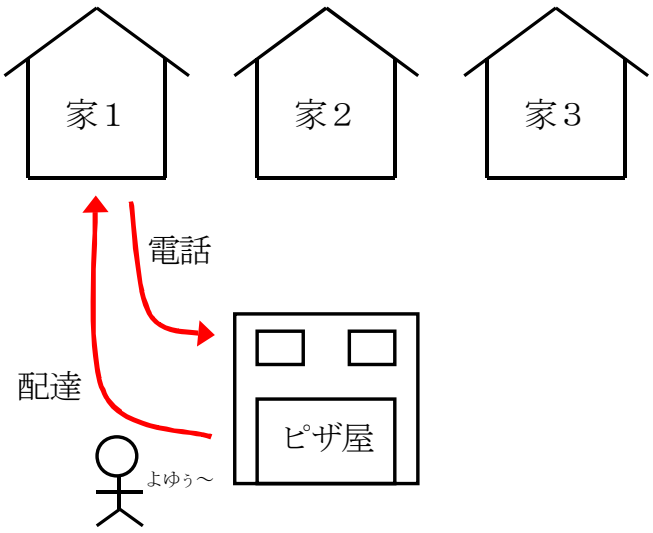
「timer1.c」では、プログラムの繰り返し(ループ)によって時間稼ぎをしていました。「timer2.c」では、割り込みを利用して正確なタイマを作ります。

7.6.1 なぜ割り込みが必要か

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。



そこで、電話で注文を受けることにします。バイト君は、わざわざ各家を回る必要がありません。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。



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。

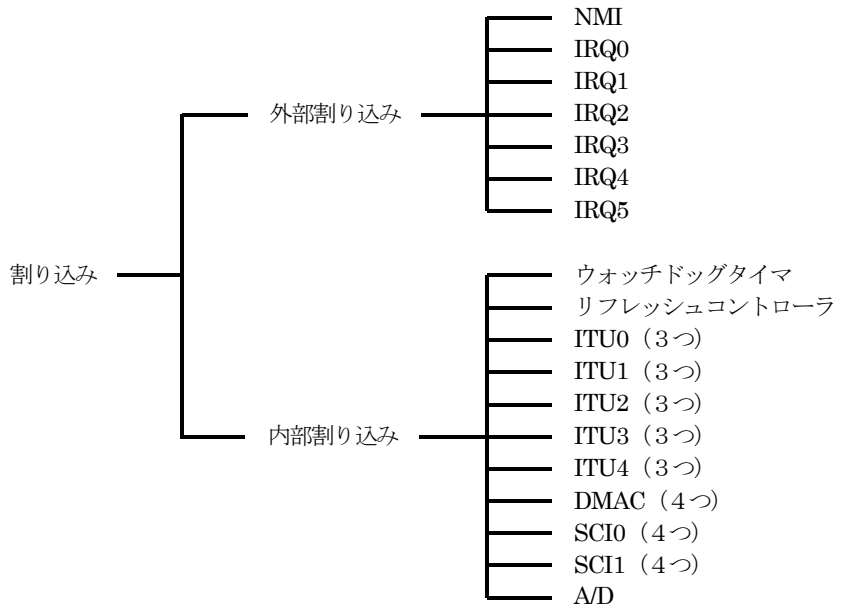
・注文がないか聞きに回る
　制御の用語で「ポーリング」といいます。定期的に監視しなければいけないので、監視する部分が多いと、タイムロスになります。
・電話で注文をうける
　制御の用語(でもないですが)で「割り込み」といいます。電話のようにきっかけがあったときだけ対処すれば良いので効率が良いです。ただし、電話の用意、電話の受け答えをする必要があります。

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	→	プログラムの割り込み設定
ベルが鳴る	→	割り込みの発生
電話対応する	→	割り込みプログラムを実行

7.6.2 割り込みの種類

割り込みには、外部からの信号がきっかけである「**外部割り込み**」、内蔵周辺機能がきっかけである「**内部割り込み**」があります。



外部割り込みは 7 つ、内部割り込みは 30 つあります。周辺機能1つに対して、複数の割り込み要因がある場合があります。例えば、ITU0～ITU4 は、3 つずつあります。
外部割り込み…外部からの信号の変化によって割り込みをかけることができます。
　　例えば、信号が"1"→"0"に変化したなどです。
内部割り込み…内蔵周辺機能に設定したきっかけにより割り込みをかけることができます。
　　例えば、SCI0(通信機能)で受信データを受けた場合などです。

7.7 プログラムの解説(割り込みの設定手順)

C言語ソースプログラムとアセンブリソースプログラムの 2 ファイルに割り込み設定する必要があります。

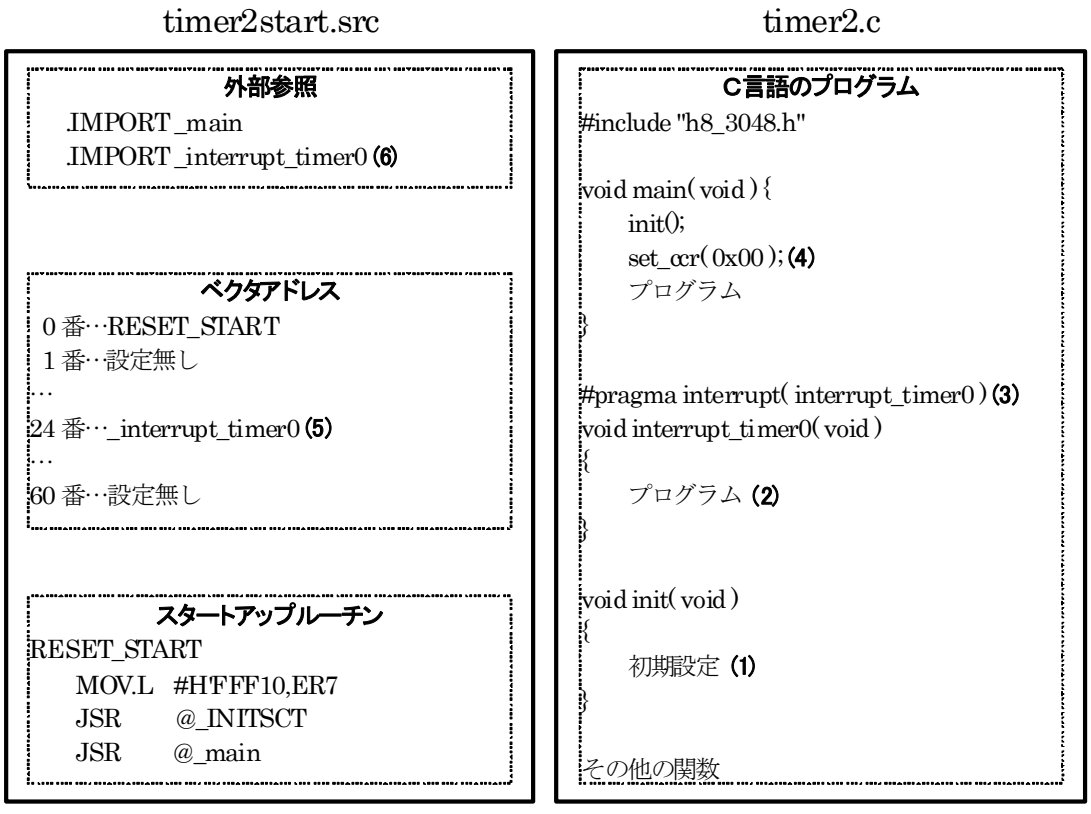
C言語ソースプログラム「timer2.c」の設定手順は、下記のとおりです。

(1) 割り込みを使う設定、割り込みを許可する
(2) 割り込みプログラムの作成
(3) 「#pragma interrupt」の設定
(4) 全体の割り込みを許可する

アセンブリソースプログラム「timer2start.src」の設定手順は、下記のとおりです。

(5) ベクタアドレスの設定(src ファイル)
(6) 「.IMPORT」の設定(src ファイル)

プログラムの構成を簡単に書くと下記のようになります。(1)～(6)が上記の番号の内容を、どの部分に記述するかを示しています。
※initsct_3048.c は割り込みに関係なく必要なため、今回の説明では省略します。



7.7.1 割り込みを使う設定、割り込みを許可する

下記 4 行のプログラムが、ITU0 を使用して 1ms ごとに割り込みを発生させる設定となります。

```
64 :        ITU0_TCR = 0x20;
65 :        ITU0_GRA = 24575;
66 :        ITU0_IER = 0x01;
67 :        ITU_STR  = 0x01;
```

(1) そもそもITUとは？

「ITU」とは、「16 ビットインテグレーテッドタイマユニット」の略称です。「インテグレーテッド(integrated)」とは、集積という意味です。集積は「多くのものを集めて積み重ねること」です。これらから、多くの機能が集まったタイマと言えます。一言では、**高性能タイマ**と言えます。

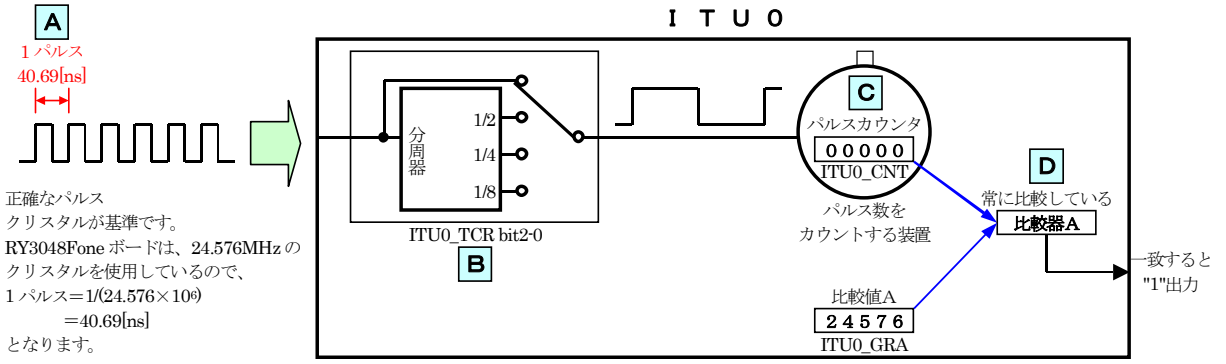
H8/3048F-ONE には ITU0～4 の 5 チャネル分内蔵されています。設定により様々な用途に使用できます。

・タイマ(時間の計測) … 1ms のタイマを作るなど
・PWM 出力 … モータのスピード制御を行うなど
・3 相 PWM 出力 … 3 相モータの制御を行うなど
・パルス入力 … パルス数の計測など
・インプットキャプチャ入力…波形のパルス幅の測定など

マイコンカーでは ITU を使用して下記の内容を行っています。
・タイマ機能を使用することにより正確な時間計測を行います。
・PWM 出力機能によって、サーボ、モータの制御を行います。
・パルス入力機能により、マイコンカーの走行距離を測ったり、走行速度を計ります。

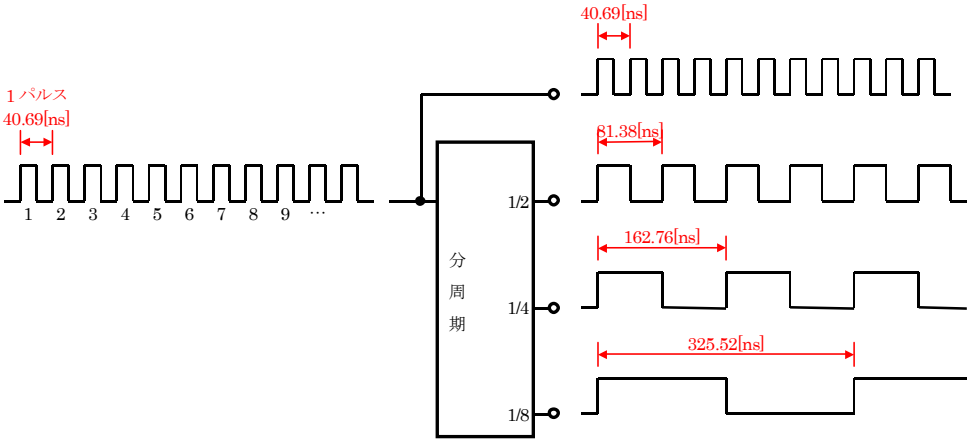
今回の演習では、ITU0 を使用して 1ms ごとに割り込みを発生させます。割り込みプログラム内で変数を+1 します。その変数を通常のプログラムでチェックすることで、時間が来たと判断できます。例えば、「変数が 3000 になるまで待つ」とすれば、変数は 1ms ごとに+1 するので、3000ms 待つことになります。これで 3 秒タイマのでき上がりです。

(2) ITU0 をタイマとして使用する原理



A 基準は、クリスタルの値です。RY3048Fone ボードは 24.576[MHz]のクリスタルが取り付けられていますので、1パルス幅は逆数の 40.69[ns]です。

B ITU の内部にはクリスタルの値を分周する機能があります。分周とは、周波数を 1/n に変換することです。要は、パルス幅を n 倍にすることです。ITU は設定により周波数を、1/1(そのまま)、1/2、1/4、1/8 に変換することができます。(下図)。どのパルスを使うかはプログラムで自由に選ぶことができます。



C パルスを数えます。1 パルスの時間は決まっているので、パルスの合計数で時間が分かります。例えば、1ms たったか計測したい場合は、1パルスが 40.69[ns]とすると、

1[ms]/40.69[ns]＝24576　となります。要は、24576 カウントされたら、1ms と判断できます。

D 比較値 A には、比較したい値である 24576 をセットしておきます。カウンタ値と比較値 A の値は、比較器 A により常に比較されています。一致すると、"1"の信号を出力します。これが割り込みとなります。マイコンは割り込み信号により割り込みプログラムを実行します。

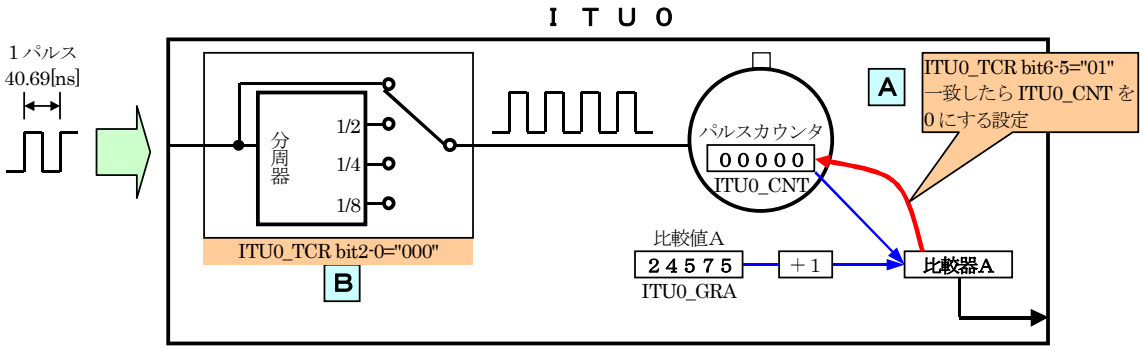
(3) ITU0 のレジスタ設定

これからの説明に出てくるレジスタと意味です。

レジスタ	名称	意味
ITU0_CNT	タイマカウンタ	パルス数をカウントするレジスタです。値によって経過時間が分かります。先ほどのパルスカウンタに当たります。
ITU0_TCR	タイマコントロールレジスタ	ITU0_CNT を 0 にするタイミングや ITU0_CNT に加える周波数(1/1,1/2,1/4,1/8)を選択します。
ITU0_GRA	ジェネラルレジスタ A	ジェネラルは一般的な、全般的な、という意味ですがここでは汎用的な、という意味です。汎用とは、「広くいろいろな方面に用いること」です。ITU0_GRA と ITU0_CNT は、自動的に比較されていて、値が一致すると割り込みを発生させたり、PWM の信号を変えたりと、設定によっていろいろな用途で使用されます。そのため、ジェネラルと呼ばれます。ジェネラルレジスタは2つありそれぞれAとBという名前が付いています。
ITU0_GRB	ジェネラルレジスタ B	ジェネラルレジスタ A と同様、2つ目としてBという名前が付いたジェネラルレジスタがあります。
ITU0_IER	タイマインタラプトイネーブルレジスタ	インタラプトは割り込み、イネーブルは動作可能にする、という意味です。割り込みを動作可能にする→割り込みを許可するかしないかのレジスタです。
ITU_STR	タイマスタートレジスタ	ITU0_CNT は最初、止まっています。ITU0_CNT を動かすのがこのレジスタです。スイッチのような役割です。
ITU0_TSR	タイマステータスレジスタ	ステータスは、地位や身分という意味ですが、コンピュータでは「状態」のことです。ITU0 の状態がどうなっているかを知ることができます。

●ITU0_TCR(タイマコントロールレジスタ)の設定内容

このレジスタは、
・ITU0_CNT を0にするタイミング
・ITU0_CNT が＋1する時間(加えるパルス幅)
を設定します。下図のようです。



各ビットの意味は下記のようになっています。

ビット:	7	6	5	4	3	2	1	0
ITU0_TCR:	－	CCLR1	CCLR0	CKEG1	CKEG0	TPSC2	TPSC1	TPSC0
設定値:	0	0	1	0	0	0	0	0
16 進数:	2				0			

・ビット 6,5:カウンタクリア 1,0
CNT のカウンタクリア要因を選択します。

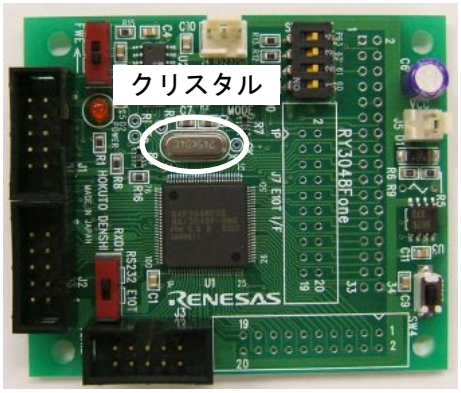
CCLR1	CCLR0	説明
0	0	CNT のクリア禁止
0	1	GRA のコンペアマッチ／インプットキャプチャで CNT をクリア
1	0	GRB のコンペアマッチ／インプットキャプチャで CNT をクリア
1	1	同期クリア

この設定が、「ITU0_CNT を0にするタイミング」です(図の A 部分)。コンペアマッチとは、コンペア＝比較、マッチ＝一致、ですので、ITU0_CNT と(ITU0_GRA＋1)が一致すると ITU0_CNT をクリアにしなさい、という設定にします。なぜ＋1した値かというと、ここが分かりづらい部分なのですが、「ITU の決まり事」としか言いようがありません。一致したと判断するのは「ITU0_CNT＝(ITU0_GRA＋1)」のときです。したがって、ITU0_CNT が 24576 になったら何かをさせたい場合(割り込みやパルスを発生させるなど)、ITU0_GRA には 24575 を入れます。ITU0_CNT が(24575+1)かどうか常に比較して、一致すると割り込みやパルスを発生させます。

・ビット 2～0:タイマプリスケーラ 2～0

CNT のカウントクロックを選択します。

TPSC2	TPSC1	TPSC0	説明
0	**0**	**0**	**内部クロック:φでカウント　　要はそのまま出力**
0	0	1	内部クロック:φ／2でカウント　要はパルスを2倍にして出力
0	1	0	内部クロック:φ／4でカウント　要はパルスを4倍にして出力
0	1	1	内部クロック:φ／8でカウント　要はパルスを8倍にして出力
1	0	0	外部クロックA:TCLKA 端子(PA0)でカウント
1	0	1	外部クロックB:TCLKB 端子(PA1)でカウント
1	1	0	外部クロックC:TCLKC 端子(PA2)でカウント
1	1	1	外部クロックD:TCLKD 端子(PA3)でカウント



この設定が、「ITU0_CNT をカウントさせるパルスの選択」の設定です(図の B 部分)。

内部クロックφとは、クリスタルの値です。クリスタルが時間を作る基準となります。マイコンカーラリー用 RY3048Fone ボードは「**24.576MHz**」のクリスタルが搭載されています。今回の設定は、「000」なので、そのままのパルス幅である 40.69[ns]ごとに ITU0_CNT がカウントされます。

ちなみにクリスタルは誤差±100ppm です。ppm は百万分の 1 ですので、100/1,000,000 の誤差、1/10,000 の誤差、0.01%となります。

外部クロックでカウントとは、クリスタルではなく外部からのパルスをポートから入力してカウントすることです。プロジェクト「enc」で詳しく説明します。

パルスを 1 倍、2 倍、4 倍、8 倍にしたときに計測できる最大時間を計算してみます。カウンタである ITU0_CNT は符号無し 16 ビット幅ですので 0～65535 までカウントすることができます。すなわち 65535 のときが計測できる最大時間です。

TPSC2	TPSC1	TPSC0	説明
0	0	0	内部クロック:φでカウント 24.576MHz でカウント、時間に直すと逆数なので 1パルス＝1÷(24.576×10^6)＝40.69[ns] 最大値は 40.69[ns]×65535＝2.666[ms]
0	0	1	内部クロック:φ／2でカウント 1パルス＝1÷(24.576×10^6／2)＝81.38[ns] 最大値は 81.38[ns]×65535＝5.333[ms]
0	1	0	内部クロック:φ／4でカウント 1パルス＝1÷(24.576×10^6／4)＝162.76[ns] 最大値は 162.76[ns]×65535＝10.67[ms]
0	1	1	内部クロック:φ／8でカウント 1パルス＝1÷(24.576×10^6／8)＝325.52[ns] 最大値は 325.52[ns]×65535＝21.33[ms]

この4つの設定のどれにすれば良いのでしょうか。割り込みをかける間隔で決めます。

・割り込みをかける間隔が 2.666[ms]以下なら、TPSC="000"にします。
・割り込みをかける間隔が 5.333[ms]以下なら、TPSC="001"にします。
・割り込みをかける間隔が 10.67[ms]以下なら、TPSC="010"にします。
・割り込みをかける間隔が 21.33[ms]以下なら、TPSC="011"にします。
・割り込みをかける間隔が 21.33[ms]以上なら、それ以下の間隔で割り込みをかけて割り込み何回目で処理をする、などプログラムで対応します。

今回は、1ms ですので"000"に設定します。

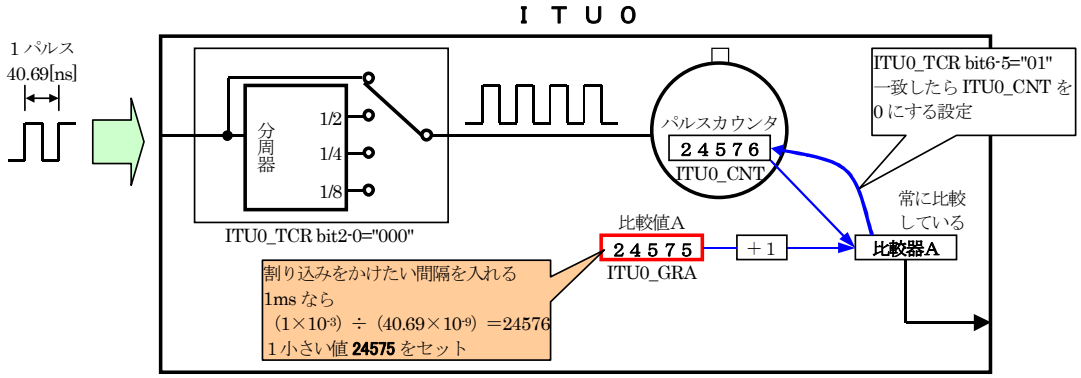
●ITU0_GRA(ジェネラルレジスタ A)の設定内容

ジェネラルは「汎用的な」という意味です。汎用とは、「広くいろいろな方面に用いること」です。ITU0_GRA と ITU0_CNT は、自動的に比較されていて、値が一致すると割り込みを発生させたり、PWM の信号を変えたりと、設定によっていろいろな用途で使用されます。そのため、ジェネラルと呼ばれます。ジェネラルレジスタは2つありそれぞれ A と B という名前が付いています。今回は A を使用します。タイマで使用するときは、ITU0_GRA を比較値 A と呼ぶと分かりやすくなります。

比較値 A には、割り込みをかける間隔を設定します。
今回は、割り込み間隔を 1[ms]にします。パルスカウンタである ITU0_CNT は 40.69[ns]ごとにカウントアップされますので、1[ms]がたつまでのカウント数は

$(1\times10^{-3})\div(40.69\times10^{-9})=24576$

となります。実際の比較は「ITU0_CNT＝(ITU0_GRA＋1)」で行われますので、比較値 A である ITU0_GRA には1小さい値「24575」を設定します。
ITU0_TCR では、「GRA のコンペアマッチ／インプットキャプチャで CNT をクリア」に設定しています。「ITU0_CNT＝(ITU0_GRA＋1)＝24576」になると、ITU0_CNT が 0 になります。要は、「0→1→2…24574→24575→0→1→2…」となります。



●ITU0_TSR(タイマステータスレジスタ)の設定内容

このレジスタをチェックすると、ITU0_CNT と ITU0_GRA、ITU0_GRB が一致したかどうか分かります。

ビット:	7	6	5	4	3	2	1	0
ITU0_TSR:	–	–	–	–	–	OVF	IMFB	IMFA

・ビット 2:オーバフローフラグ

CNT のオーバフロー／アンダフローの発生を示すステータスフラグです。

OVF	説明
0	OVF=1 の状態で、OVF フラグをリードした後、OVF フラグに 0 をライトしたとき
1	CNT の値がオーバフロー(h'FFFF→h'0000)またはアンダフロー(h'0000→h'FFFF)したとき

今回は、関係有りません。

・ビット 1:インプットキャプチャ／コンペアマッチフラグ B

GRB のコンペアマッチまたはインプットキャプチャの発生を示すステータスフラグです。

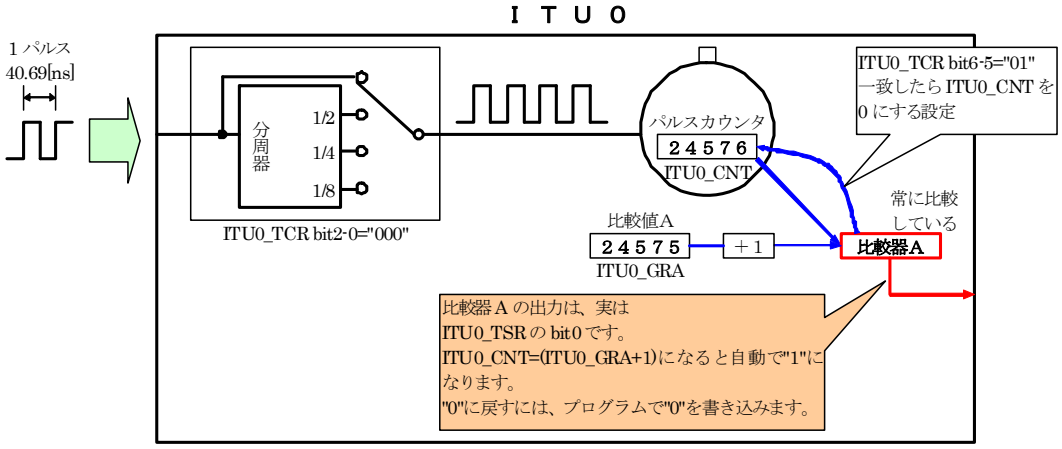
IMFB	説明
0	IMFB=1 の状態で、IMFB フラグをリードした後、IMFB フラグに 0 をライトしたとき
1	(1)GRB がアウトプットコンペアレジスタとして機能している場合、CNT=GRB になったとき (2)GRB がインプットキャプチャレジスタとして機能している場合、インプットキャプチャ信号により CNT の値が GRB に転送されたとき

今回は、関係有りません。

・ビット 0:インプットキャプチャ／コンペアマッチフラグ A

GRA のコンペアマッチまたはインプットキャプチャの発生を示すステータスフラグです。

IMFA	説明
0	IMFA=1 の状態で、IMFA フラグをリードした後、IMFA フラグに 0 をライトしたとき
1	**(1)GRA がアウトプットコンペアレジスタとして機能している場合、CNT=GRA になったとき** (2)GRA がインプットキャプチャレジスタとして機能している場合、インプットキャプチャ信号により CNT の値が GRA に転送されたとき



ITU0_CNT＝(ITU0_GRA＋1)になったときに、ITU0_TSR の bit0 が"1"になります。

ちなみに ITU0_TSR の bit0 のことを IMFA と呼びます。次に出てきますので、次まで覚えておいてください。

●ITU0_IER(タイマインタラプトイネーブルレジスタ)の設定内容

インタラプトは割り込み、イネーブルは動作可能にする、という意味です。割り込みを動作可能にする→割り込みを許可するかしないかを設定するレジスタです。

ビット:	7	6	5	4	3	2	1	0
ITU0_IER:	−	−	−	−	−	OVIE	IMIEB	IMIEA
設定値:	0	0	0	0	0	0	0	1
16 進数:	0				1			

・ビット 2:オーバフローインタラプトイネーブル

TSR の OVF フラグが1にセットされたとき、OVF フラグによる割り込み要求を許可／禁止します。

OVIE	説明
0	OVF フラグによる割り込み(OVI)要求を禁止
1	OVF フラグによる割り込み(OVI)要求を許可

今回は関係ありません。関係ない場合は、元の値のままにしておきます。

・ビット 1:インプットキャプチャ／コンペアマッチインタラプトイネーブル B

TSR の IMFB フラグが1にセットされたとき、IMFB フラグによる割り込み要求を許可／禁止します。

IMIEB	説明
0	IMFB フラグによる割り込み(IMIB)要求を禁止
1	IMFB フラグによる割り込み(IMIB)要求を許可

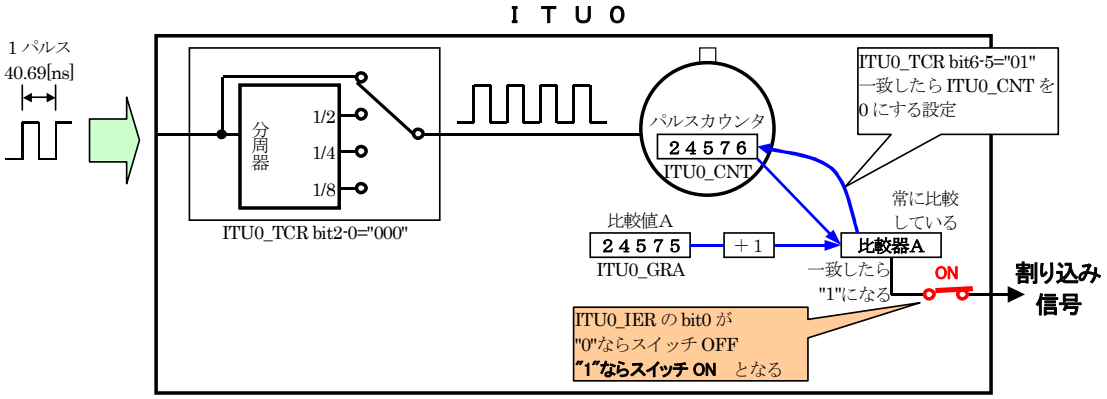
今回は関係ありません。関係ない場合は、元の値のままにしておきます。

・ビット 0:インプットキャプチャ／コンペアマッチインタラプトイネーブル A

TSR の IMFA フラグが1にセットされたとき、IMFA フラグによる割り込み要求を許可／禁止します。

IMIEA	説明
0	IMFA フラグによる割り込み(IMIA)要求を禁止
1	IMFA フラグによる割り込み(IMIA)要求を許可

ITU0_TSR の bit0 が"1"かつ、今回の ITU0_IER の bit0 が"1"なら、割り込みがあったと見なしなさい、ということです。ITU0_TSR の bit0 の、スイッチのような役割です。



●ITU_STR(タイマスタートレジスタ)の設定内容

bit:	7	6	5	4	3	2	1	0
ITU_STR:	–	–	–	ITU4_CNT のカウント開始	ITU3_CNT のカウント開始	ITU2_CNT のカウント開始	ITU1_CNT のカウント開始	ITU0_CNT のカウント開始
設定値:	0	0	0	0	0	0	0	1
16 進数:	0				1			

・ビット 4-0:カウンタスタート 4～0

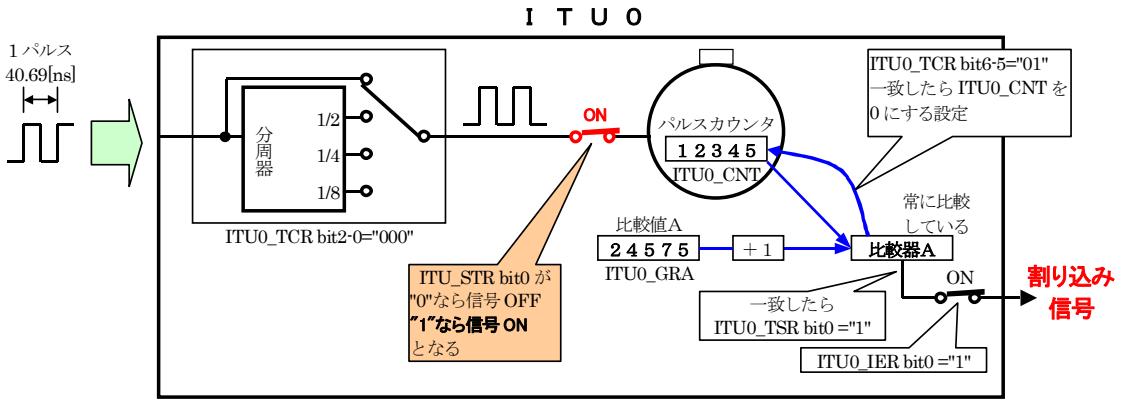
タイマカウンタ x の動作／停止を選択します。

STR4-0	説明
0	ITUx の CNT のカウント動作は停止
1	ITUx の CNT はカウント動作

※ x は 0～4

ITU0_CNT は、実はまだカウントしていません。このレジスタを設定することにより動き出すのです。ITU_STR は今までの ITU0_CNT のように、チャネルを示す数字が付きません。これは、ITU_STR 1つで ITU0～4 チャネルまで共通の設定のためです。

今回は ITU0 をスタートさせるので bit0 を”1”にします。



ITU_STR の bit0 は、ITU0_CNT へ加えるパルス信号線のスイッチのような役割です。

7.7.2 割り込みプログラムの作成

割り込みを使う設定を行い、割り込みを許可しました。次に割り込みが起こったときに実行するプログラムを作ります。下記が、そのプログラムです。

```
84 :  void interrupt_timer0( void )
85 :  {
86 :      ITU0_TSR &= 0xfe;                    /* フラグクリア              */
87 :      cnt0++;
88 :  }
```

(1) 関数名

```
84 :  void interrupt_timer0( void )
85 :  {
              プログラム
88 :  }
```

自由に名前を付けて構いません。一応、割り込みと分かるように「interrupt」、ITU0 をタイマとして使用するので「timer0」とします。これらの名称を合わせて、「interrupt_timer0」とします。
後ほど、ITU0 割り込みが発生したとき実行する関数を登録します。そのときの関数名は「interrupt_timer0」にします。

(2) interrupt_timer0 関数の処理内容

```
87 :      cnt0++;
```

先の設定により、interrupt_timer0 関数が、1ms ごとに割り込みで呼ばれて実行されます。
87 行で cnt0 変数を＋1しています。cnt0 は、大域変数です。どの関数からもアクセスできます。interrupt_timer0 関数で 1ms ごとに+1 して、main 関数などで cnt0 の値をチェックすることにより正確に時間を計測することができます。

(3) ITU0_TSRのクリア

```
86 :      ITU0_TSR &= 0xfe;                    /* フラグクリア              */
```

86 行で、ITU0_TSR を 0xfe で AND 処理しています。ITU0_CNT＝(ITU0_GRA+1)で ITU0_TSR の bit0 が"1"になっていますので、次回(1ms 後)の一致に備えて"0"にしておきます。

●クリア方法

余談ですので、時間がない方はこの部分を飛ばして構いません。

```
    ITU0_TSR = 0x00;
```

としては、だめなのでしょうか。
残念ながらこれではクリアされません。ITU0_TSR の bit0(IMFA)の説明をもう一度読み直してみます。

ITU0_TSR(タイマステータスレジスタ)の設定内容

ビット:	7	6	5	4	3	2	1	0
ITU0_TSR:	−	−	−	−	−	OVF	IMFB	IMFA

・ビット 0:インプットキャプチャ/コンペアマッチフラグ A
GRA のコンペアマッチまたはインプットキャプチャの発生を示すステータスフラグです。

IMFA	説明
0	**IMFA=1 の状態で、IMFA フラグをリードした後、IMFA フラグに 0 をライトしたとき**
1	(1)GRA がアウトプットコンペアレジスタとして機能している場合、CNT=GRA になったとき (2)GRA がインプットキャプチャレジスタとして機能している場合、インプットキャプチャ信号により CNT の値が GRA に転送されたとき

この、「**IMFA フラグをリードした後**」がポイントです。そのまま値を書き込むだけなら「リード(読み込み)」していません。そのため、0x00 を書き込んでも、bit0 は"0"になりません。そこで AND 処理します。

ビット	7	6	5	4	3	2	1	0
ITU0_TSR	?	?	?	?	?	?	?	1
AND する値	1	1	1	1	1	1	1	0
結果	※	※	※	※	※	※	※	0

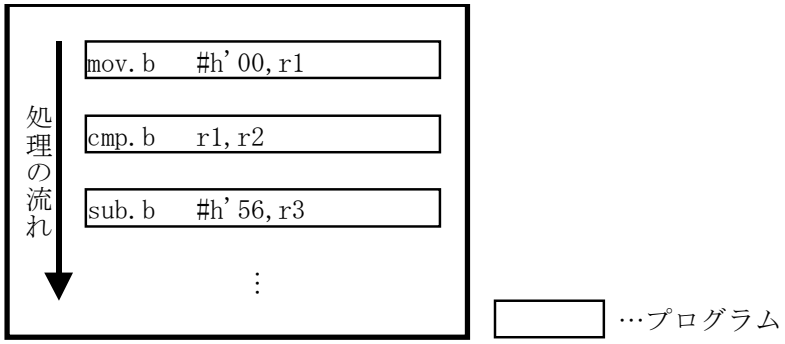
※＝変化無し

AND 処理することにより、ITU0_TSR はいったん CPU の内部レジスタに**読み込まれて**演算し、ITU0_TSR へ書き込まれます。これで ITU0_TSR の bit0 は"0"になります。また、AND 処理には、ITU0_TSR の bit0 以外の値を変えないという理由もあります。

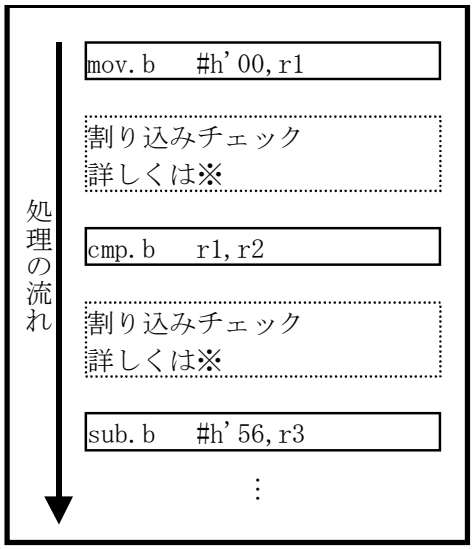
●ITU0_TSR をクリアしないとどうなるのか

余談ですので、時間がない方はこの部分を飛ばして構いません。

割り込みプログラム内で ITU0_TSR の bit0 を"0"にしなければどうなるのでしょうか。
マイコンは、機械語1命令ずつ実行していきます。



H8/3048F-ONE マイコンには外部割り込みが 7 個、内部割り込みが 30 個あります。実は、これらすべての割り込みを機械語1命令が終わるごとにチェックしているのです。



※割り込みチェックの流れ

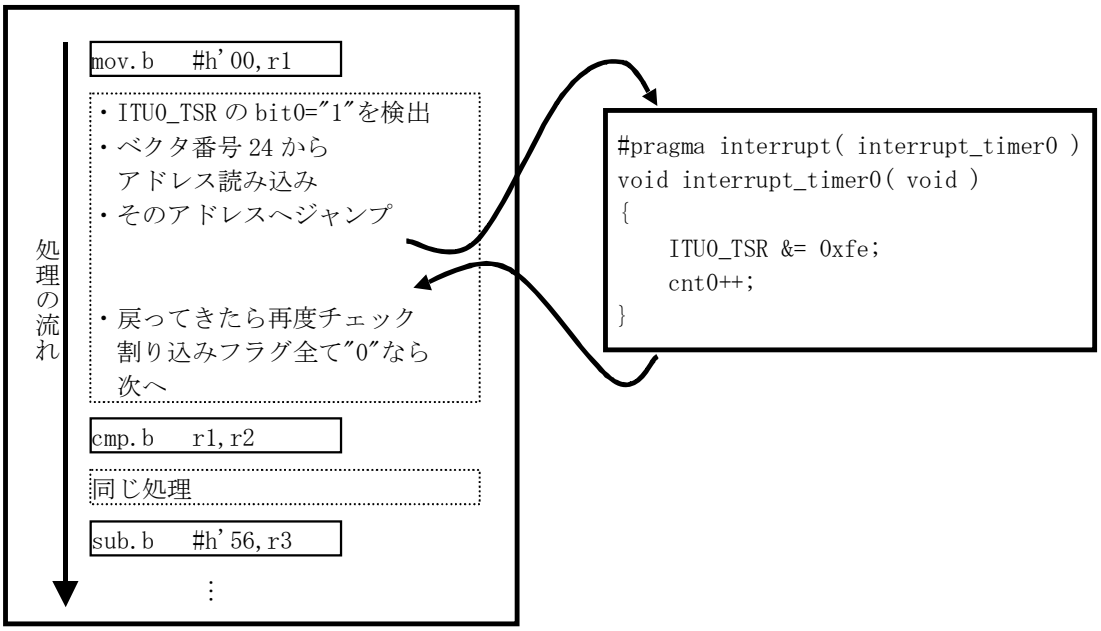
・外部割り込み７個、内部割り込み３０個をチェック
・割り込みがあれば、そのベクタ番号からジャンプするアドレスを読み込み、割り込みプログラムへジャンプ
・割り込みプログラムが終わると、**戻ってきて、また割り込みをチェック**。全ての割り込み信号が"0"なら次のプログラムへ移行する

…プログラム

…プログラムとプログラムの間の処理

ここで、「**戻ってきて、また割り込みをチェック**」に注目します。ITU0_TSR の bit0 が"1"なら割り込み処理終了後、また割り込みが発生したと判断して割り込みプログラムへ移ります。1ms はおろか、1命令も実行していないにもかかわらずです。これは、メインプログラムは実行せずにずっと割り込みプログラムだけを実行する、いわゆる無限ループになります。割り込みプログラム内では、かならず該当するステータスフラグと呼ばれるビット、今回は ITU0_TSR の bit0 をクリアしなければいけません。

下記に ITU0_TSR の bit0="1"を検出したときの処理例を示します。

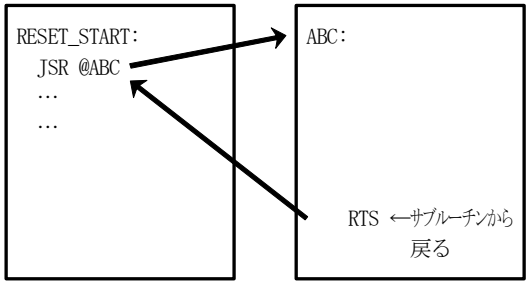


7.7.3 「#pragma interrupt」の設定

※シャープ　プラグマ　インタラプトと読みます。

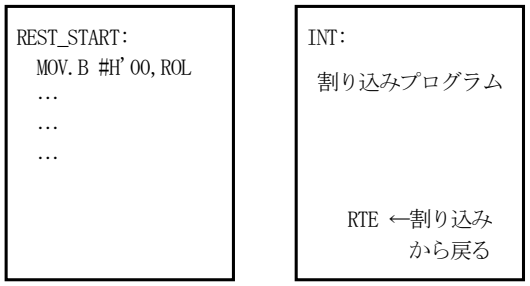
(1) 割り込みプログラム終わりの処理

アセンブリソースプログラムで JSR 命令(Jump to SubRoutine)でサブルーチンが呼ばれた後、呼ばれた場所に戻るのは RTS という命令です。BASIC でいうと、「GOSUB」命令と同じです(下図)。



※RTS＝ReTurn from Subroutine：サブルーチンから戻る

割り込みで実行されたサブルーチンの終了は、RTS 命令ではなく、RTE 命令です。これは、割り込みが発生したときに自動的に保存される CCR レジスタの値を戻してから呼ばれた場所へ戻る命令です(下図)。



RTE＝ReTurn from Exception：割り込みから戻る

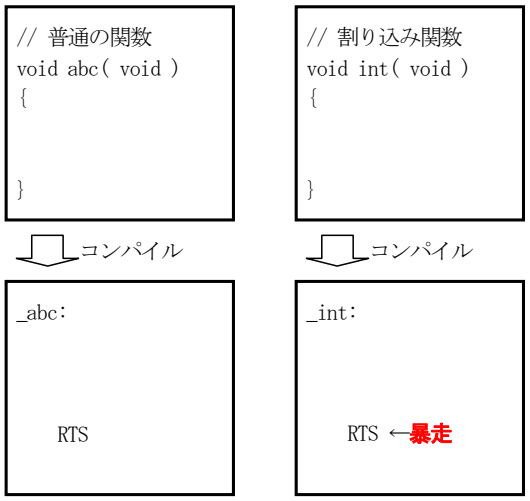
これらは、

・普通のサブルーチンから戻る→RTS 命令を使う

・割り込みから戻る→RTE 命令を使う

と**プログラマが区別して使い分ける必要があります**。

C言語ではどうでしょうか。



C 言語では普通の関数なのか、

割り込み関数なのか判断できない

C言語では、割り込みであろうと無かろうと「}」で関数を終了します。そのため、普通の関数か割り込み関数か判断できません。割り込みプログラムを終了するときは、「RTE」命令でなければいけません。「RTS」命令だとマイコンが暴走してしまいます。そのため、コンパイラに「この関数は割り込み関数なので RTE 命令を使ってください」と知らせる必要があります。それが「#pragma interrupt」宣言です。

(2) 「#pragma interrupt」宣言

```
83 :  #pragma interrupt( interrupt_timer0 )
```

割り込みでジャンプする関数には、必ず「pragma interrupt」宣言しなければ行けません。
「#pragma interrupt(**割り込み関数名**) 」として、カッコ内に関数名を書きます。

```
// 普通の関数
void abc( void )
{


}
```

コンパイル

```
_abc:


    RTS
```

```
// 割り込み関数
#pragma interrupt( int )
void int( void )
{

}
```

コンパイル

```
_int:


    RTE ←OK!!
```

#pragma interrupt で設定した関数は

コンパイラは RTS ではなく RTE にする

割り込みでジャンプする関数には必ず宣言する、と覚えておけば OK です。
ちなみに、**プログラム中から「pragma interrupt」宣言した関数を呼ぶことはできません。**

```
void main( void )
{
    abc();                  ←不可
}
#pragma interrupt( abc )
void abc( void )
{
    printf( "abc!" );
}
```

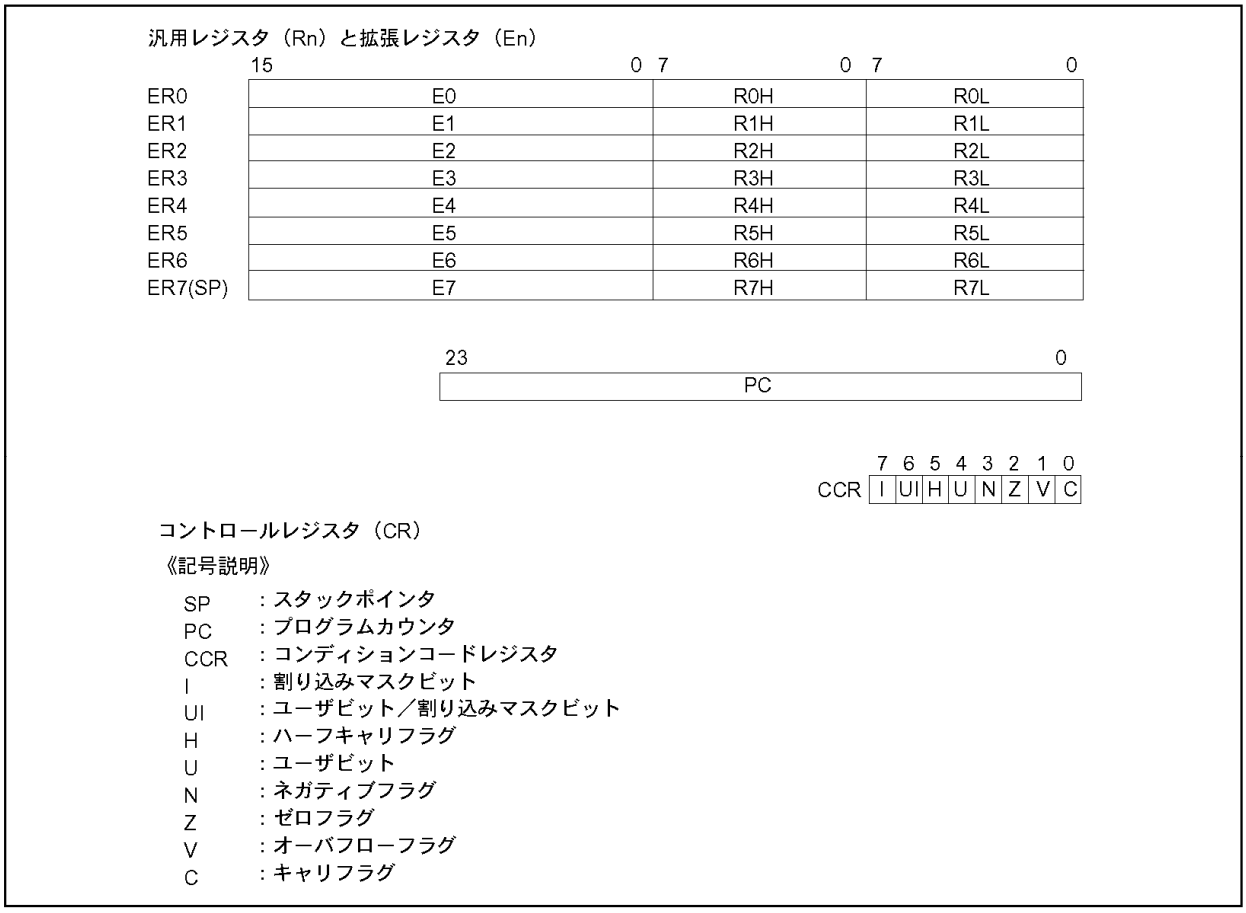
7.7.4 全体の割り込みを許可する

(1) CCRレジスタ

今までの設定をしてもまだ割り込みが発生しません。実はすべての割り込みを許可するかしないか、設定する必要があるのです。初期値は「許可しない」という設定です。

その設定は CPU の内部レジスタである、CCR(コンディションコードレジスタ)という 8 ビット幅のレジスタです。メモリマップド I/O 方式では、アドレス上に ROM、RAM、I/O レジスタがありますが、内部レジスタだけは考え方が全く違います。内部レジスタは、下記のような特徴があります。

・CPU の中にある（メモリとは別）
・番地はなく、特別な名前が付いている
・専用の機能を持っていて使い方が決められている



▲内部レジスタの構成

コンディションは「体の状態」という意味ですので、CCR はマイコンの状態を示すレジスタです。CCR の bit7 は、割り込みマスクビットという名称が付いていて、”0”にすることによりマイコン全体の割り込みが許可されます。初期値は”1”です。

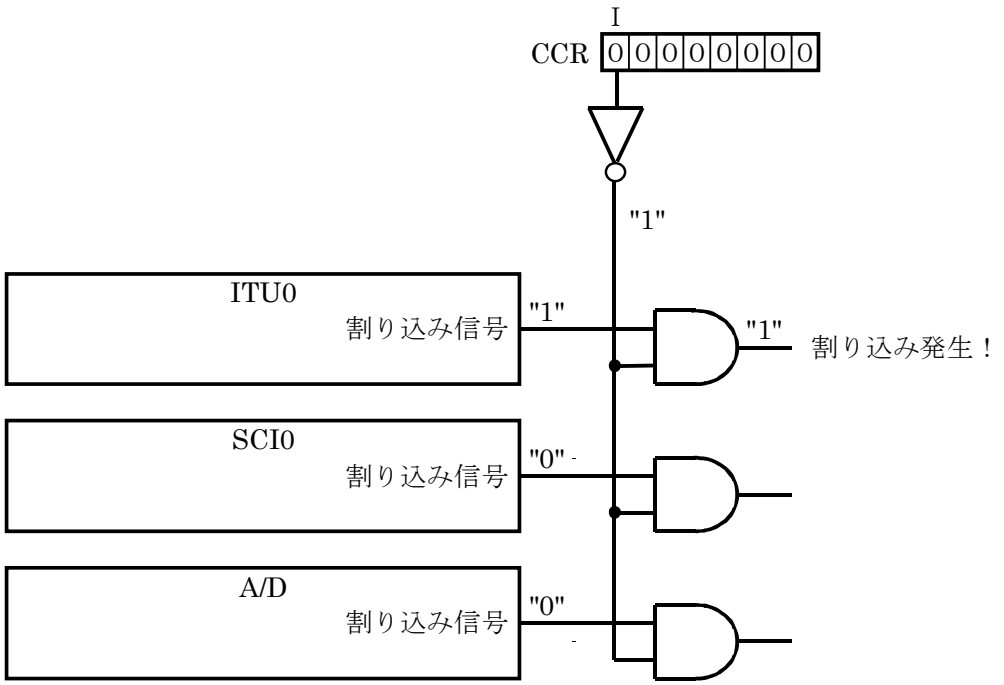
●CCR(コンディションコードレジスタ)の構成内容

ビット:	7	6	5	4	3	2	1	0
CCR:	I	UI	H	U	N	Z	V	C
設定値:	0	–	–	–	–	–	–	–
16 進数:	0				0			

CCR は、CPU 内部の特別なレジスタのため、Cプログラム上から直接変更することはできません。そのため、値を変える関数が用意されています。それが、「set_ccr」という関数です。この関数は、machine.h ファイルをインクルードすることにより使用できます。使い方は、

```
set_ccr( CCR レジスタにセットする値 );
```

となります。本当は I ビットである bit7 のみ"0"にすれば良いのですが、それ以外に"0"を書き込んでもここでは影響はないので、0x00 を代入します。



全体の割り込みをＣＣＲのＩ(アイ)ビットで制御している

(2) プログラム

この命令は内蔵周辺機能の初期化が終わった後に行います。main 関数内にあります。

```
32 :    init();                    /* 初期化                 */
33 :    set_ccr( 0x00 );           /* 全体割り込み許可       */
```

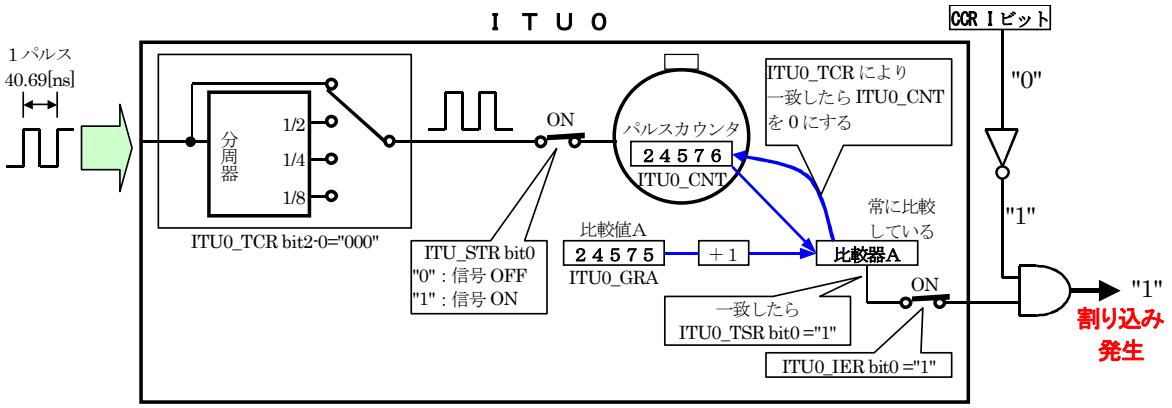
これで CPU 全体の割り込みが許可され、本当に割り込みかかります。内蔵周辺機能(例えば ITU0_IER など)の設定だけでは割り込みは発生しません。

7.7.5 ベクタアドレスの設定（timer2start.srcファイル）

割り込みを使うには、「timer2start.src」ファイルも変更する必要があります。これからその作業手順を説明します。

(1) 割り込みが発生したときのジャンプ先

下記のように、ITU0 を設定、割り込みが発生しました。

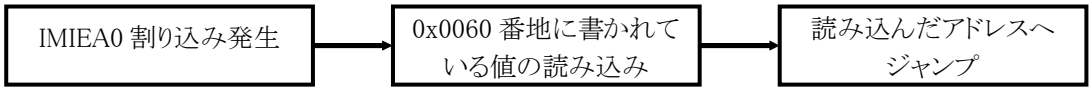


割り込み関数へジャンプします。timer2.c に interrupt_timer0 関数を作りました。たまたま、人間が分かりやすいように「割り込みのタイマ0」という名称にしましたが、マイコンにとってはどこにジャンプすれば良いのか分かりません。

H8/3048F-ONE では、0x0000～0x00fc 番地の ROM 領域を**ベクタアドレス領域**と呼び、特別な番地です。割り込みが発生すると、下記のような動作を行います。

・割り込みの種類によって決められたアドレスから 4 バイトのデータを読み込みます。
・読み込んだデータのアドレスにジャンプします。
・ジャンプして、その番地にあるプログラムから実行し始めます。
・RTE 命令で割り込みを終了、割り込みがかかる前のプログラムに戻ります。

「ITU0_CNT＝(ITU0_GRA＋1)」により割り込みが発生した場合（IMIEA0 割り込み）、0x0060 番地に書かれている値を読み込み、そのアドレスへジャンプします。



(2) 割り込みの種類とベクタアドレス

割り込みの種類とベクタアドレスをまとめた表を、下記に示します。

割込み要因	要因発生元	ベクタ番号	ベクタアドレス	IPR	優先順位
リセット	外部端子	0	H'0000 ～ H'0003	——	
NMI		7	H'001C ～ H'001F	——	高
IRQ_0		12	H'0030 ～ H'0033	IPRA7	↑
IRQ_1		13	H'0034 ～ H'0037	IPRA6	
IRQ_2	外部端子	14	H'0038 ～ H'003B	IPRA5	
IRQ_3		15	H'003C ～ H'003F		
IRQ_4		16	H'0040 ～ H'0043	IPRA4	
IRQ_5		17	H'0044 ～ H'0047		
リザーブ	————	18	H'0048 ～ H'004B		
		19	H'004C ～ H'004F		
WOVI(インターバルタイマ)	ｳｫｯﾁﾄﾞｯｸﾞﾀｲﾏ	20	H'0050 ～ H'0053	IPRA3	
CMI (コンペアマッチ)	ﾘﾌﾚｯｼｭｺﾝﾄﾛｰﾗ	21	H'0054 ～ H'0057		
リザーブ	————	22	H'0058 ～ H'005B		
		23	H'005C ～ H'005F		
IMIA0 (コンペアマッチ/インプットキャプチャA0)	ITU チャネル0	**24**	**H'0060 ～ H'0063**	IPRA2	
IMIB0 (コンペアマッチ/インプットキャプチャB0)		25	H'0064 ～ H'0067		
OVI0 (オーバフロー0)		26	H'0068 ～ H'006B		
リザーブ	————	27	H'006C ～ H'006F		
IMIA1 (コンペアマッチ/インプットキャプチャA1)	ITU チャネル1	28	H'0070 ～ H'0073	IPRA1	
IMIB1 (コンペアマッチ/インプットキャプチャB1)		29	H'0074 ～ H'0077		
OVI1 (オーバフロー1)		30	H'0078 ～ H'007B		
リザーブ	————	31	H'007C ～ H'007F		
IMIA2 (コンペアマッチ/インプットキャプチャA2)	ITU チャネル2	32	H'0080 ～ H'0083	IPRA0	
IMIB2 (コンペアマッチ/インプットキャプチャB2)		33	H'0084 ～ H'0087		
OVI2 (オーバフロー2)		34	H'0088 ～ H'008B		
リザーブ	————	35	H'008C ～ H'008F		
IMIA3 (コンペアマッチ/インプットキャプチャA3)	ITU チャネル3	36	H'0090 ～ H'0093	IPRB7	
IMIB3 (コンペアマッチ/インプットキャプチャB3)		37	H'0094 ～ H'0097		
OVI3 (オーバフロー3)		38	H'0098 ～ H'009B		
リザーブ	————	39	H'009C ～ H'009F		
IMIA4 (コンペアマッチ/インプットキャプチャA4)	ITU チャネル4	40	H'00A0 ～ H'00A3	IPRB6	
IMIB4 (コンペアマッチ/インプットキャプチャB4)		41	H'00A4 ～ H'00A7		
OVI4 (オーバフロー4)		42	H'00A8 ～ H'00AB		
リザーブ	————	43	H'00AC ～ H'00AF		
DEDN0A	DMAC	44	H'00B0 ～ H'00B3	IPRB5	
DEDN0B		45	H'00B4 ～ H'00B7		
DEDN1A		46	H'00B8 ～ H'00BB		
DEDN1B		47	H'00BC ～ H'00BF		
リザーブ	————	48	H'00C0 ～ H'00C3	——	
		49	H'00C4 ～ H'00C7		
		50	H'00C8 ～ H'00CB		
		51	H'00CC ～ H'00CF		

ERI0 (受信エラー0)	SCI チャネル0	52	H'00D0 ～ H'00D3	IPRB3	
RXI0 (受信完了 0)		53	H'00D4 ～ H'00D7		
TXI0 (送信データエンプティ 0)		54	H'00D8 ～ H'00DB		
TEI0 (送信終了 0)		55	H'00DC ～ H'00DF		
ERI1 (受信エラー1)	SCI チャネル1	56	H'00E0 ～ H'00E3	IPRB2	
RXI1 (受信完了 1)		57	H'00E4 ～ H'00E7		
TXI1 (送信データエンプティ 1)		58	H'00E8 ～ H'00EB		
TEI1 (送信終了 1)		59	H'00EC ～ H'00EF		↓
ADI (A/D エンド)	A／D	60	H'00F0 ～ H'00F3	IPRB1	低

(3) プログラム

timer2start.src ファイルの初めには、ベクタ番号 0 から 60 まで順番にジャンプ先アドレスを宣言しています。すべての割り込みのベクタアドレスが記述されており、使わない割り込みは「RESERVE」(予約)と記述します。

```
 16 :          .SECTION V
 17 :          .DATA.L RESET_START              ;  0 H'000000   ﾘｾｯﾄ
 18 :          .DATA.L RESERVE                  ;  1 H'000004   ｼｽﾃﾑ予約
 19 :          .DATA.L RESERVE                  ;  2 H'000008   ｼｽﾃﾑ予約
(中略)
 39 :          .DATA.L RESERVE                  ; 22 H'000058   ｼｽﾃﾑ予約
 40 :          .DATA.L RESERVE                  ; 23 H'00005c   ｼｽﾃﾑ予約
 41 :          .DATA.L _interrupt_timer0        ; 24 h'000060   ITU0 IMIA0
 42 :          .DATA.L RESERVE                  ; 25 H'000064   ITU0 IMIB0
 43 :          .DATA.L RESERVE                  ; 26 H'000068   ITU0 OVI0
(中略)
 76 :          .DATA.L RESERVE                  ; 59 H'0000ec   SCI1 TEI1
 77 :          .DATA.L RESERVE                  ; 60 H'0000f0   A/D ADI
```

上表より、ITU0 の IMIA0 割り込みはベクタ番号 24 なので、25 番目(0 から始まるので)の.DATA.L にジャンプ先関数名「_interrupt_timer0」を記述します。

```
 41 :          .DATA.L _interrupt_timer0        ; 24 h'000060   ITU0 IMIA0
```

「.DATA.L」はロングサイズ(4バイト)でデータを作りなさいという命令です。「_interrupt_timer0」があるプログラムの番地データを作ります。timer2 プロジェクトでは「_interrupt_timer0」は、0x1bc 番地になります。

その他の番号には

```
 18 :          .DATA.L RESERVE                  ;  1 H'000004   ｼｽﾃﾑ予約
```

と、「RESERVE」が入っています。登録する必要のない番号には「RESERVE」を入れておきます。「RESERVE」は、

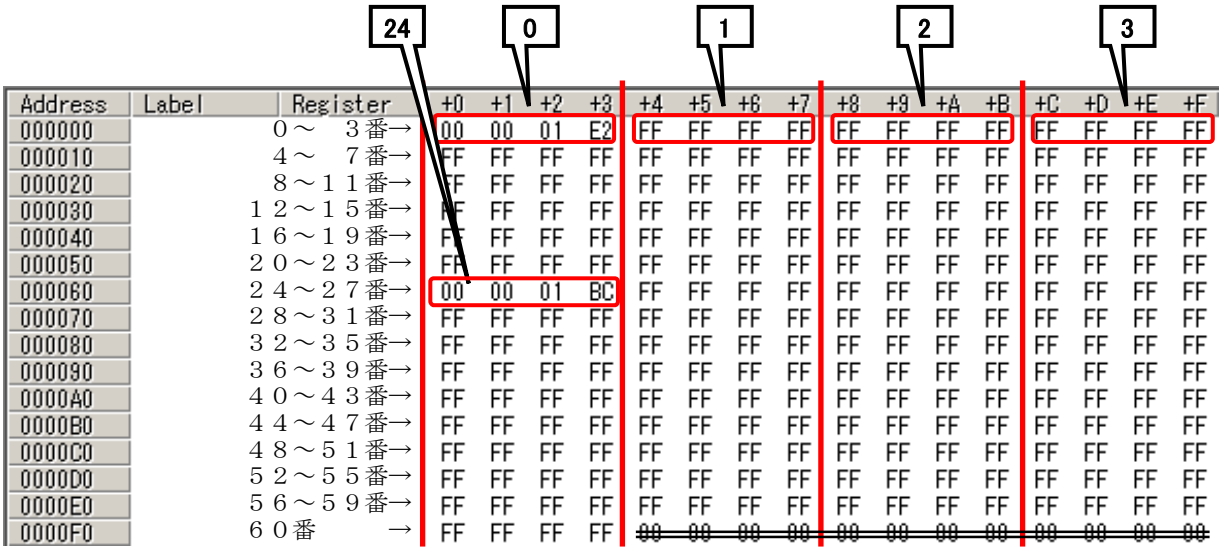
```
  4 :  RESERVE: .EQU     H'FFFFFFFF             ; 未使用領域のアドレス
```

と登録しています。.EQU は、「イコール」命令です。「RESERVE」という単語が出てきたら「H'FFFFFFFF」に置き換えなさい、という意味です。登録する必要のない場所は、アドレスが分かりませんので、ダミーとして「H'FFFFFFFF」を入れておきます。結果的には、

```
 18 :          .DATA.L H'FFFFFFFF               ;  1 H'000004   ｼｽﾃﾑ予約
```

ということになります。

下記に実際のメモリ値を示します。4 バイトごとに 0～60 番まであります。



Address	Label	Register	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
000000		0～ 3番→	00	00	01	E2	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
000010		4～ 7番→	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
000020		8～11番→	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
000030		12～15番→	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
000040		16～19番→	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
000050		20～23番→	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
000060		24～27番→	00	00	01	BC	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
000070		28～31番→	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
000080		32～35番→	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
000090		36～39番→	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0000A0		40～43番→	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0000B0		44～47番→	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0000C0		48～51番→	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0000D0		52～55番→	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0000E0		56～59番→	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0000F0		60番 →	FF	FF	FF	FF	00	00	00	00	00	00	00	00	00	00	00	00

0番のベクタアドレスは、リセットがかかったときのジャンプ先アドレスで 0x01e2 番地です。24 番が ITU0 の IMIA0 割り込みが発生したときのジャンプ先アドレスで 0x01bc 番地となります。

7.7.6 「.IMPORT」の設定(timer2start.srcファイル)

(1) IMPORTはなぜ必要か

アセンブルやコンパイルはファイルごとに行います。

```
41 :          .DATA.L _interrupt_timer0        ; 24 h'000060    ITU0 IMIA0
```

をアセンブルするとき、「_interrupt_timer0」を探します。しかし、timer2start.src 内には、ありません。timer2.c ファイル内にあるためです。そのため、「_interrupt_timer0」は他の場所にあるので、他のファイルを探してください、とアセンブラに知らせる必要があります。その命令が、「.IMPORT」命令なのです。

```
10 :          .IMPORT _interrupt_timer0
```

という記述で、リンケージエディタは_interrupt_timer0」が他のファイルにあることを理解して、41 行は、他のファイルにある「_interrupt_timer0」というラベル名へジャンプするようマシン語に変換します。

(2) アセンブリソースプログラムのファイルからC言語プログラムのファイルを呼ぶ場合

アセンブリソースプログラム「timer2start.src」からC言語ソースプログラム「timer2.c」の関数を呼ぶときは、関数の先頭名に「__」(アンダバー)を付けなければいけないというルールがあります。

C 言語ソースプログラムでは、「interrupt_timer0」ですが、アセンブリソースプログラムでは「_interrupt_timer0」と記述します。

アセンブリソースプログラムの「_interrupt_timer0」 ＝ C言語ソースプログラムの「interrupt_timer0」

これで割り込みの設定は完了です。割り込みを使いこなしましょう！！

7.8 プログラムの解説(割り込み以外)

7.8.1 main関数

```
30 :  void main( void )
31 :  {
32 :      init();                          /* 初期化                  */
33 :      set_ccr( 0x00 );                 /* 全体割り込み許可        */
34 :
35 :      while( 1 ) {
36 :          PBDR = 0x55;
37 :          timer( 1000 );
38 :          PBDR = 0xaa;
39 :          timer( 1000 );
40 :          PBDR = 0x00;
41 :          timer( 1000 );
42 :      }
43 :  }
```

32 行で init 関数を呼んでポートの入出力設定、ITU0 の設定を行います。
33 行で全体の割り込みを許可しています。
35 行には、while 文があり、カッコの中が常に真なので対応するカッコ閉じである 42 行まで無限ループです。
36 行で PB に 0x55 を出力、1000ミリ秒時間稼ぎします。
38 行で PB に 0xaa を出力、1000ミリ秒時間稼ぎします。
40 行で PB に 0x00 を出力、1000ミリ秒時間稼ぎします。
「timer2.c」の timer 関数は、割り込みにより正確に時間をカウントしているため、正確な 1000 ミリ秒の時間となります。

7.8.2 timer関数

```
 74 :  void timer( unsigned long timer_set )
 75 :  {
 76 :      cnt0 = 0;
 77 :      while( cnt0 < timer_set );
 78 :  }
```

76 行目で cnt0 変数を 0 にしています。
77 行目では、cnt0 変数が timer_set 変数より小さいか比較、小さいなら 77 行を繰り返します。イコールか大きくなったら次の行に進みます。次の行は、ありませんので timer 関数の終了です。

7.9 まとめ

(1) 割り込みを使う設定、割り込みを許可する

timer2.c ソースファイルの init 関数内で ITU0 のレジスタを設定します。

設定するレジスタ	詳細
ITU0_TCR	ITU0_CNT の＋1する時間、クリア要因の設定をします。 0x20→40.69[ns]でカウント　0x21→81.38[ns]でカウント 0x22→162.76[ns]でカウント　0x23→325.52[ns]でカウント
ITU0_GRA	割り込み周期を設定します。「設定したい周期÷ITU0_CNT のカウント時間－1」です。周期 1ms、カウント時間 40.69[ns]なら $(1\times10^{-3})\div(40.69\times10^{-9})-1=24575$ となります。
ITU0_IER	割り込みを許可します。0x01 を設定します。
ITU_STR	ITU のカウンタ(CNT)を動作させる設定です。 ITU0 を使用するので、0x01 を設定します。

(2) 割り込みプログラムの作成

timer2.c ソースファイルに interupt_timer0 関数を追加します。

```
void interrupt_timer0( void )
{
    ITU0_TSR &= 0xfe;                    ←フラグのクリアを必ず行う
    cnt0++;                              ←割り込みプログラム
}
```

(3) 「#pragma interrupt」の設定

timer2.c ソースファイルの割り込み関数に「#pragma interrupt」命令を追加します。

```
#pragma interrupt( interrupt_timer0 )    ←割り込みプログラムであることを宣言する
void interrupt_timer0( void )
{
    ITU0_TSR &= 0xfe;                    ←必ず行う
    cnt0++;                              ←割り込みプログラム
}
```

(4) 全体の割り込みを許可する

timer2.c ソースファイルに追加します。

```
void main( void )
{
    init();             /* 初期化             */   ←初期化
    set_ccr( 0x00 );    /* 全体割り込み許可 */   ←全体の割り込み許可、必ず行う
```

(5) ベクタアドレスの設定(srcファイル)

timer2start.src ソースファイルにベクタアドレスを設定します。今回は ITU0 の IMIA0 割り込みが発生するので、ベクタアドレスの表より 24 番部分(0x0060 番地)に「_interrupt_timer0」を記述します。関数名を記述するとき、先頭に「_(アンダーバー)」を追加することを忘れないようにします。

```
        .DATA.L _interrupt_timer0        ; 24 h'000060    ITU0 IMIA0
```

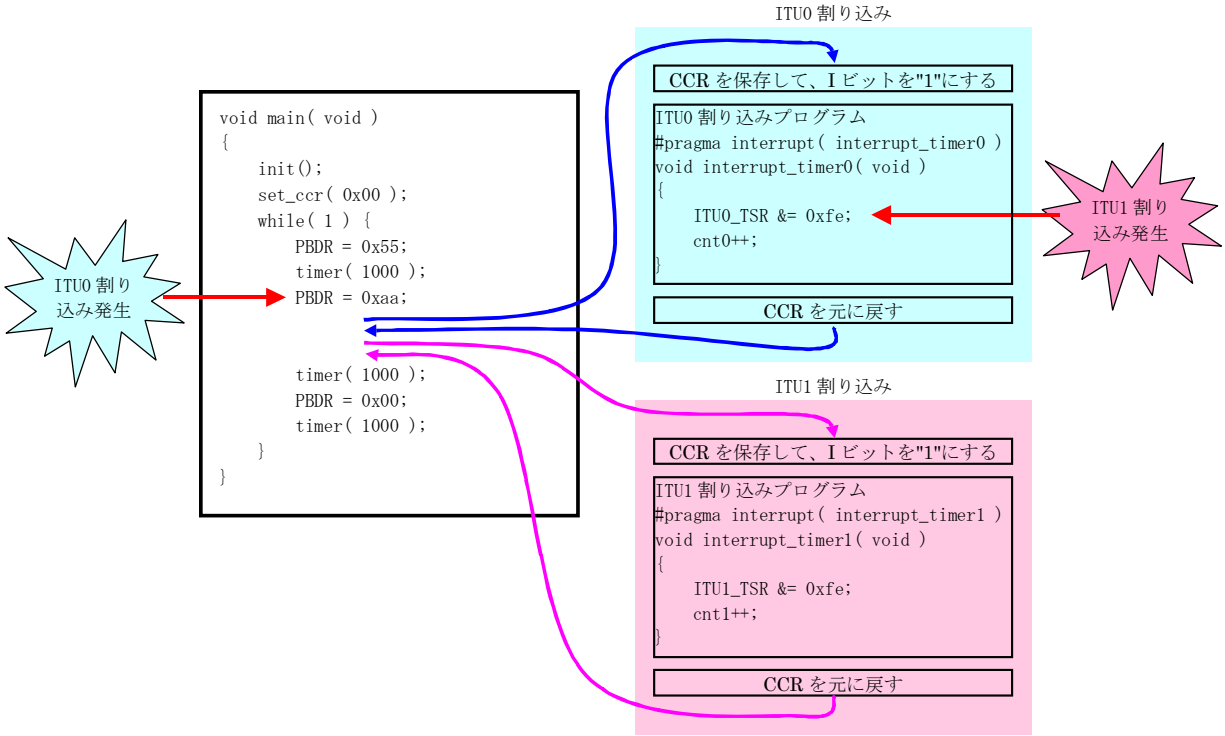
(6) 「.IMPORT」の設定(srcファイル)

「.IMPORT＋関数名」で、別のファイルにこの関数があることを伝えます。

```
        .IMPORT _interrupt_timer0        ; 外部参照
```

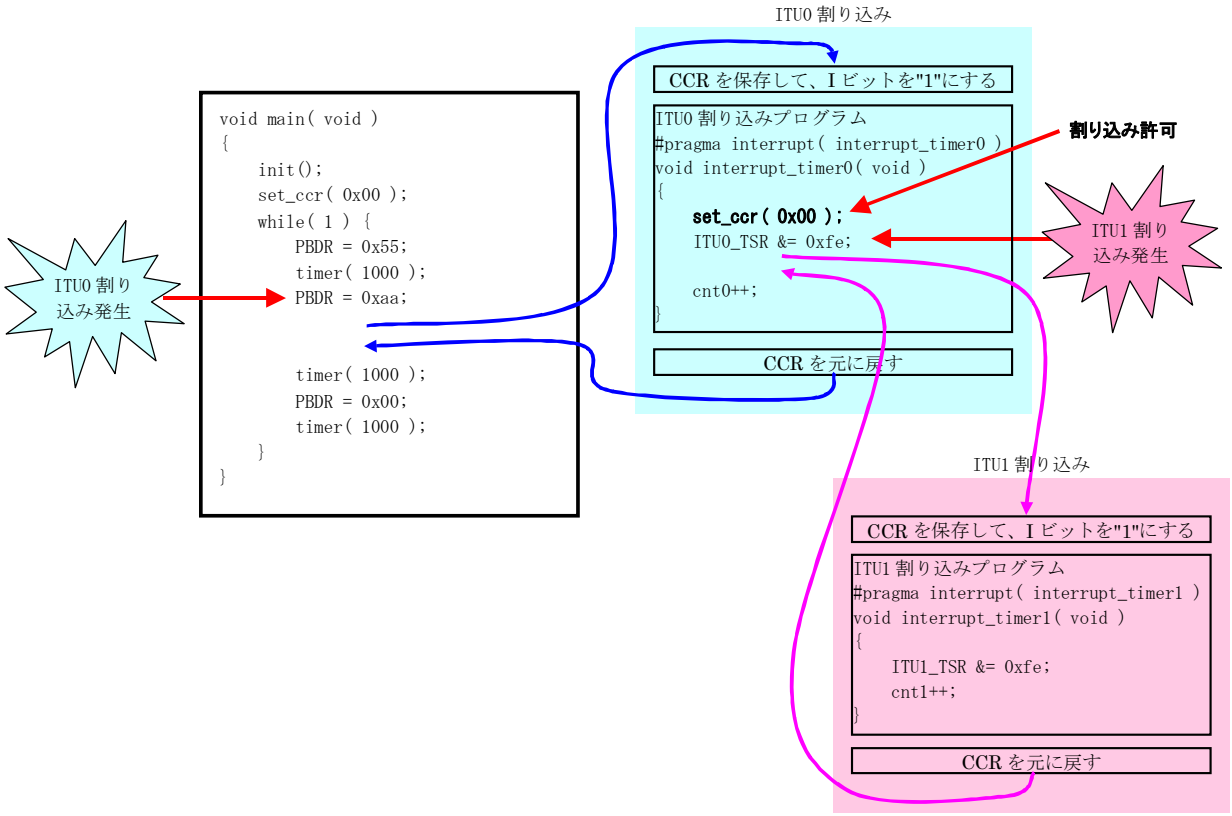
※参考資料　多重割り込みについて

割り込みプログラム実行中に割り込みがあった場合、どうなるのでしょうか。ITU0 割り込みと、ITU1 割り込みを許可していたとします。



1. 「PBDR=0xaa」を実行している最中に、ITU0 割り込みが発生しました。
2. 割り込みプログラムである interrupt_timer0 関数へジャンプします。
3. 実は、interrupt_timer0 関数が実行される前に自動で CCR が保存され、CCR の I ビットが"1"になります。このビットは、「割り込みを許可するか、しないか」を決めるビットで"1"は割り込みを禁止にします。
4. interrupt_timer0 関数を実行中に、ITU1 割り込みが発生しました。しかし、割り込みが禁止されているので、ITU1 割り込みは**保留**されます。**無視ではありません！**
5. interrupt_timer0 関数が終わりました。終わると同時に、CCR が割り込み処理の実行前状態に戻ります。割り込み処理の実行前はもちろん割り込み許可の状態です。
6. 許可されると、保留されていた ITU1 割り込みが実行されます。interrupt_timer1 関数へ処理が移ります。
7. interrupt_timer1 関数が終わりました。もう保留されている割り込みはありませんので、メインプログラムの次の行である「timer(1000)」が実行されます。

では、割り込み処理中に別の割り込みを許可したい場合はどうしたらよいでしょうか。割り込み処理に移ると自動的に CCR のIビットが"1"になり、割り込みを禁止にします。それなら、プログラムで割り込みを許可してしまえばよいのです。



1. interrupt_timer0 関数内で「set_ccr(0x00)」を実行します。CCR の 7 ビットが"0"になり、割り込みが許可されます。
2. 「ITU0_TSR &= 0xfe」命令を実行中に、ITU1 割り込みが発生しました。割り込みが許可されていますので、「ITU0_TSR &= 0xfe」が終わると ITU1 割り込みプログラムである interrupt_timer1 関数へ移ります。
3. interrupt_timer1 関数が終わると、interrupt_timer0 関数へ戻ります。
4. interrupt_timer0 関数が終わるとメイン関数へ戻り、処理を続けます。

このように、割り込みプログラム内で CCR の 7 ビットを"0"にすれば、割り込みが許可されます。割り込み内で割り込みが起これば、多重割り込みとなります。多重割り込みを行いたくなければ、何もしなくとも大丈夫です。元々割り込みが禁止されています。

※割り込み内で割り込みを許可したとき、また同じ割り込みが起こると同じ割り込みを再度実行します。これを再帰コールと呼び、その関数から自分自身を呼び出すことになります。

※CCR レジスタを操作する関数
CCR レジスタに値をセットする関数が、set_ccr 関数です。他にも下記のような関数があります。

set_ccr(引数);	数の値を CCR レジスタにセットする
or_ccr(引数);	引数の値を現在の CCR レジスタの値と OR をとり結果を CCR にセットする
and_ccr(引数);	引数の値を現在の CCR レジスタの値と AND をとり結果を CCR にセットする

8. プロジェクト「ad」 A/D変換値をLEDへ出力

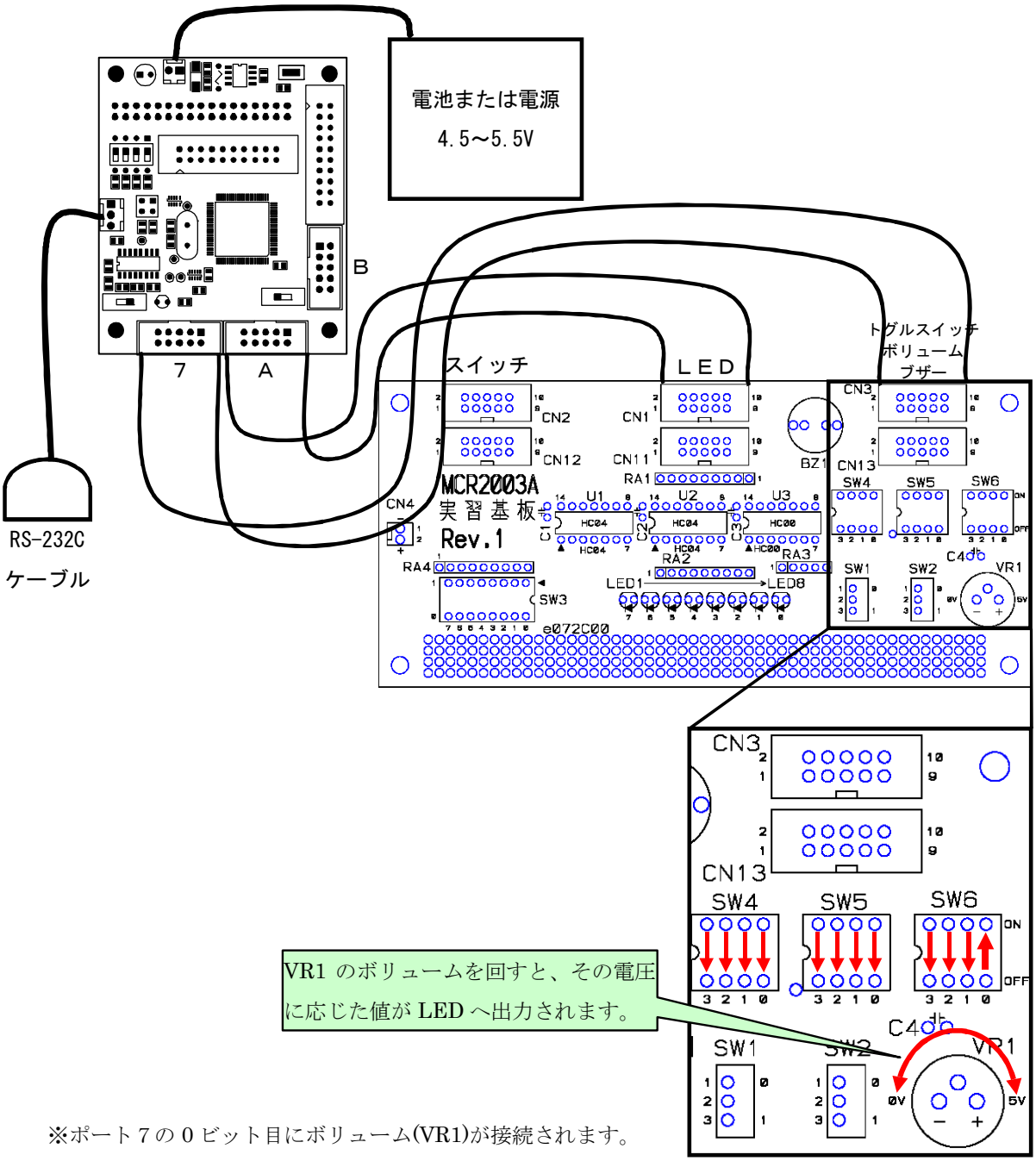
8.1 概要

実習基板のボリューム電圧をマイコンで読み込み A/D 変換します。その値を LED に出力して変換値を見てみます。マイコンのポートは、下記を使用します。

・ポート 7 の 0 ビット・・・アナログ電圧(0～5V)入力
・ポート A の全ビット・・・LED へデータ出力

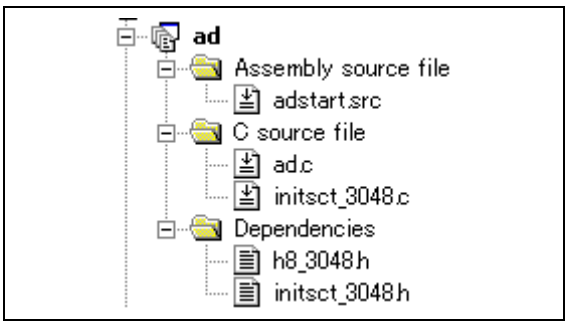
8.2 接続

・CPU ボードのポート 7 と、実習基板のトグルスイッチ・ボリューム部をフラットケーブルで接続します。
・CPU ボードのポート A と、実習基板の LED 部をフラットケーブルで接続します。
・実習基板の SW6 No0 のスイッチを ON、SW4～6 のその他のビットを OFF にします。



※ポート 7 の 0 ビット目にボリューム(VR1)が接続されます。

8.3 プロジェクトの構成



	ファイル名	内容
1	adstart.src	アセンブリ言語で記述されたアセンブリソースファイルです。このファイルの構造は下記のようになっています。 ベクタアドレス ＋ スタートアップルーチン
2	initsct_3048.c	初期値のないグローバル変数(セクション B 領域)、初期値のあるグローバル変数(セクション R 領域)の初期化用です。
3	ad.c	実際に制御するプログラムが書かれています。H8/3048F-ONE の内蔵周辺機能の初期化も行います。
4	h8_3048.h	H8/3048F-ONE の内蔵周辺機能の I/O レジスタを定義したファイルです。
5	initsct_3048.h	initsct_3048.c のヘッダファイルです。

8.4 プログラム「ad.c」

```
 1 :  /*****************************************************************************/
 2 :  /* 電圧を入力し、ＬＥＤへ出力「ad.c」                                        */
 3 :  /* 入力：P70電圧(ボリューム等)       出力：PA7-0(LED等)                      */
 4 :  /*                          2005.04 ジャパンマイコンカーラリー実行委員会   */
 5 :  /*****************************************************************************/
 6 :  /*====================================*/
 7 :  /* インクルード                       */
 8 :  /*====================================*/
 9 :  #include    <machine.h>
10 :  #include    "h8_3048.h"
11 :
12 :  /*====================================*/
13 :  /* シンボル定義                       */
14 :  /*====================================*/
15 :
16 :  /*====================================*/
17 :  /* プロトタイプ宣言                   */
18 :  /*====================================*/
19 :  void init( void );
20 :  int get_ad( void );
21 :
22 :  /*****************************************************************************/
23 :  /* メインプログラム                                                          */
24 :  /*****************************************************************************/
25 :  void main( void )
26 :  {
27 :      int ad;
28 :
29 :      init();                         /* 初期化                    */
30 :
31 :      while( 1 ) {
32 :          ad = get_ad();
33 :          PADR = ad;
34 :      }
35 :  }
36 :
```

```
37 :  /*********************************************************************/
38 :  /* H8/3048F-ONE 内蔵周辺機能の初期化                                  */
39 :  /*********************************************************************/
40 :  void init( void )
41 :  {
42 :      /* ポートの入出力設定 */
43 :      P1DDR = 0xff;
44 :      P2DDR = 0xff;
45 :      P3DDR = 0xff;
46 :      P4DDR = 0xff;
47 :      P5DDR = 0xff;
48 :      P6DDR = 0xf0;                       /* CPU基板上のDIP SW        */
49 :      P8DDR = 0xff;
50 :      P9DDR = 0xf7;
51 :      PADDR = 0xff;                       /* LED基板                  */
52 :      PBDDR = 0xff;
53 :      /* ※ポート7は、入力専用なので入出力設定はありません    */
54 :
55 :      /* A/Dの初期設定 */
56 :      AD_CSR = 0x00;                      /* AN0使用                  */
57 :  }
58 :
59 :  /*********************************************************************/
60 :  /* A/D値読み込み(AN0)                                                */
61 :  /* 引数    なし                                                      */
62 :  /* 戻り値 A/D値 0～255 本当は0-1023ですが下位2bit分を無視しています     */
63 :  /*********************************************************************/
64 :  int get_ad( void )
65 :  {
66 :      int i;
67 :
68 :      AD_CSR |= 0x20;                     /* ADスタート               */
69 :      while( !(AD_CSR & 0x80) );          /* エンドフラグをチェック   */
70 :      AD_CSR &= 0x7f;                     /* エンドフラグクリア       */
71 :      i = AD_DRA >> 8;                    /* 代入                     */
72 :
73 :      return i;
74 :  }
75 :
76 :  /*********************************************************************/
77 :  /* end of file                                                       */
78 :  /*********************************************************************/
```

8.5 プログラムの解説

8.5.1 A/D変換の初期設定

H8/3048F-ONE には 10bit の分解能の A/D 変換器が内蔵されており、0～5V(電源電圧)の電圧を 0～1023 ($2^{10}-1$)の値に変換できます。

アナログ入力端子は、ポート 7 の bit0～7 まで 8 端子あり、プログラムで入力端子を切り替えて 1 端子ずつ A/D 値を取り込むことができます。

init 関数内で、A/D 変換器の初期設定をしています。ここでは、A/D 変換器の初期化をしているだけなので、A/D 変換はしていません。プログラムは下記のとおりです。

```
56 :      AD_CSR = 0x00;                    /* AN0 使用                  */
```

●AD_CSR(A/D コントロール／ステータスレジスタ)の設定内容

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

・ビット 7:A/D エンドフラグ(ADF)

A/D 変換の終了を示すステータスフラグです。

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

初期設定では関係有りません。

・ビット 6:A/D インタラプトイネーブル(ADIE)

A/D 変換の終了による割り込み(ADI)要求の許可／禁止を選択します。

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

A/D 変換が終了したとき、割り込みを発生させることができます。今回は使いません。

・ビット 5:A/D スタート(ADST)

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

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

初期設定では関係有りません。

・ビット 4:スキャンモード(SCAN)

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

SCAN	説明
0	単一モード
1	スキャンモード

単一モードとは、1端子でアナログ入力を行うことです。スキャンモードとは、1～4端子を自動で切り替えながらアナログ入力を行うことです。今回は1端子のみアナログ入力しますので、単一モードとします。

・ビット 3:クロックセレクト(CKS)

A/D 変換時間の設定を行います。変換時間の切り替えは、ADST＝0 の状態で行ってください。

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

1ステートとは、クリスタルの周期のことです。RY3048Fone ボードは、24.576MHz のクリスタルを使用しているので、

1ステート＝1／(24.576×10^6)＝40.69[ns]

となります。したがって、それぞれの最大変換時間は、

CKS=0:266ステート→40.69×10^{-9}×266＝10.82[μs]

CKS=1:133ステート→40.69×10^{-9}×133＝5.41[μs]

となります。

変換が早いと精度が悪い、変換が遅いと精度がよい、と考えがちですが CKS の設定は単純に早いか遅いかだけです。どちらも精度は同じです。ここではどちらにしても動作には影響しないので、0 にしておきます。

・ビット 2-0:チャネルセレクト(CH2-0)

SCAN ビットと共にアナログ入力チャネルを選択します。

チャネル選択と切り替えは、ADST＝0 の状態で行ってください。

CH2	CH1	CH0	単一モード	スキャンモード
0	0	0	P70	P70
0	0	1	P71	P70、P71
0	1	0	P72	P70～P72
0	1	1	P73	P70～P73
1	0	0	P74	P74
1	0	1	P75	P74、P75
1	1	0	P76	P74～P76
1	1	1	P77	P74～P77

この設定がどの端子を使うかの設定です。ここでは、P70 端子を使用するので、"000"を設定します。

8.5.2 A/D変換値を読み込むget_ad関数

get_ad 関数で、A/D 値を読み込みます。

```
64 :  int get_ad( void )
65 :  {
66 :      int i;
67 :
68 :      AD_CSR |= 0x20;                  /* AD スタート                */
69 :      while( !(AD_CSR & 0x80) );       /* エンドフラグをチェック     */
70 :      AD_CSR &= 0x7f;                  /* エンドフラグクリア         */
71 :      i = AD_DRA >> 8;                 /* 代入                       */
72 :
73 :      return i;
74 :  }
```

まず、A/D 変換をスタートします。スタートするには、AD_CSR のスタートビットを"1"にします。

●AD_CSR(A/D コントロール／ステータスレジスタ)の設定内容

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

・ビット 5:A/D スタート(ADST)

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

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

bit5 を"1"にすれば良いことが分かります。bit5 のみ"1"にするので、OR 演算します。

```
68 :        AD_CSR |= 0x20;                    /* AD スタート                */
```

次に、A/D 変換が終了するまで待ちます。最大、CKS で設定した時間分
$1/(24.576\times10^{6})\times266=10.8[\mu s]$
で A/D 変換は完了します。
終了したかどうかは、bit7 をチェックします。

・ビット 7:A/D エンドフラグ(ADF)
A/D 変換の終了を示すステータスフラグです。

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

```
69 :        while( !(AD_CSR & 0x80) );         /* エンドフラグをチェック    */
```

while 文で bit7が"0"かどうかチェックします。成り立てば、要は"0"なら 69 行を繰り返します。A/D 変換が終了して bit7が"1"になると、成立しなくなり次の行へ進みます。
変換中のとき、bit7="0"です。カッコの中は、
!(AD_CSR & 0x80)
=!(0x00 & 0x80)　←AD_CSR の bit6-0 は"0"とする
=!0x00　!a a が 0 なら 1、a が 0 以外なら 0 となる。
=1　while は成り立つ→繰り返す

変換終了時、bit7="1"です。カッコの中は、
!(AD_CSR & 0x80)
=!(0x80 & 0x80)　←AD_CSR の bit6-0 は"0"とする
=!0x80　!a a が 0 なら 1、a が 0 以外なら 0 となる。
=0　while は成り立たない→次の行へ進む

bit7が"1"になったので、次に備えて"0"にします。bit7のみ、"0"にするので AND 演算を行います。

```
70 :        AD_CSR &= 0x7f;                    /* エンドフラグクリア        */
```

最後に、A/D 変換値を読み込みます。

```
71 :        i = AD_DRA >> 8;                   /* 代入                      */
```

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

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

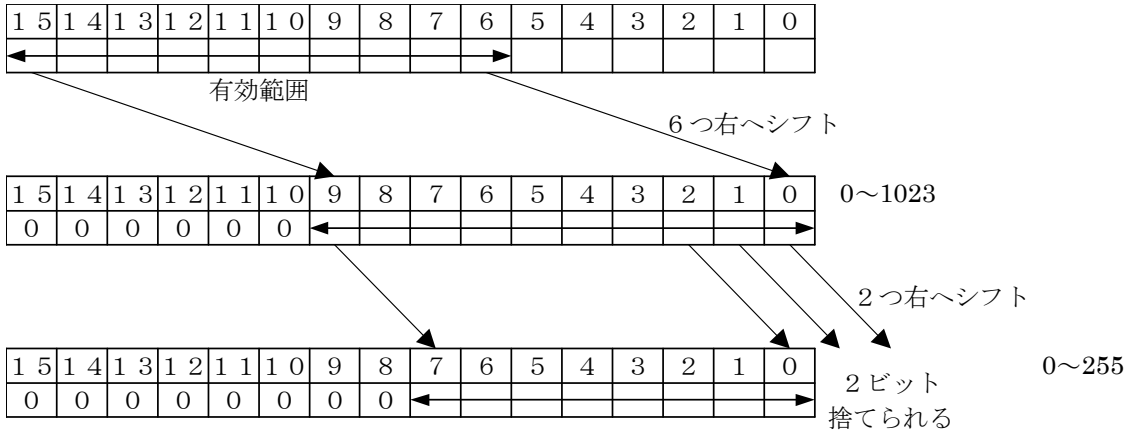
AD_DR は A から D まで4つあり、P70～P77 の端子に入力して A/D 変換された電圧値が左にシフトされた状

態で格納されます。A/D 値は 16bit レジスタに左詰で格納されており、そのうちの 10bit が有効範囲です。プログラムで使うには、6ビット右にシフトして使います。P70～P77 と AD_DRA～AD_DRD との対応は決まっていて下表のようになっています。

電圧を入力する端子	読み込むレジスタ
P70 か P74 を使用	AD_DRA
P71 か P75 を使用	AD_DRB
P72 か P76 を使用	AD_DRC
P73 か P77 を使用	AD_DRD

今回のプログラムでは、P70 端子を使うので、読み込むレジスタは、AD_DRA になります。

通常は、6 ビットシフトして 0～1023 の値にします。今回は LED が 8 ビット分しかないので、更に 2 ビットシフトして、0～255 の値にします。



シフトした桁には、0が入ります。

8.5.3 main関数

```
25 : void main( void )
26 : {
27 :     int ad;
28 :
29 :     init();                              /* 初期化                    */
30 :
31 :     while( 1 ) {
32 :         ad = get_ad();
33 :         PADR = ad;
34 :     }
35 : }
```

int 型で、ad 変数を宣言します。
変数 ad に A/D 取得値の 0～255 の値が入ります。そのままポート A へ出力しています。これを繰り返します。

8.6 まとめ

設定するレジスタ	詳細
AD_CSR	A/D 変換の初期設定とどの端子を A/D 入力端子とするか設定します。 0x00…P70 端子　0x01…P71 端子　0x02…P72 端子　0x03…P73 端子 0x04…P74 端子　0x05…P75 端子　0x06…P76 端子　0x07…P77 端子

A/D 値の読み込みは、下記のように行います。

```
64 :  int get_ad( void )
65 :  {
66 :      int i;
67 :
68 :      AD_CSR |= 0x20;                ←A/D 変換のスタート
69 :      while( !(AD_CSR & 0x80) );     ←A/D 変換が終わったかどうかチェック/
70 :      AD_CSR &= 0x7f;                ←終わったときに"1"になるビットをクリア
71 :      i = AD_DRA >> 8;               ←レジスタより A/D 値の読み込み
72 :                                       8 ビット右シフトなら 0～255 の値、
73 :      return i;                        6 ビット右シフトなら 0～1023 の値が返ってくる
74 :  }
```

71 行のレジスタは、
・P70 か P74 の端子を使用している場合→AD_DRA から読み込み
・P71 か P75 の端子を使用している場合→AD_DRB から読み込み
・P72 か P76 の端子を使用している場合→AD_DRC から読み込み
・P73 か P77 の端子を使用している場合→AD_DRD から読み込み
と、使用する端子により変わります。
AD_CSR の初期設定(56 行)が 0x00 の場合、P70 端子を使用する設定なので、AD_DRA から A/D 値を読み込みます。

※ルネサステクノロジのハードウェアマニュアルでは、ポートをアナログ入力端子として使用するとき、ポート名を P70 と呼ばず、AN0(アナログ入力端子0)と呼んでいます。本テキストでは、混乱を招くためすべてポート7で統一していますが、統一できていない部分もあります。下記のように読み替えてください。
AN0→P70
AN1→P71
AN2→P72
AN3→P73
AN4→P74
AN5→P75
AN6→P76
AN7→P77

9. プロジェクト「ad2」 A/D変換値をLEDへ出力(スキャンモード)

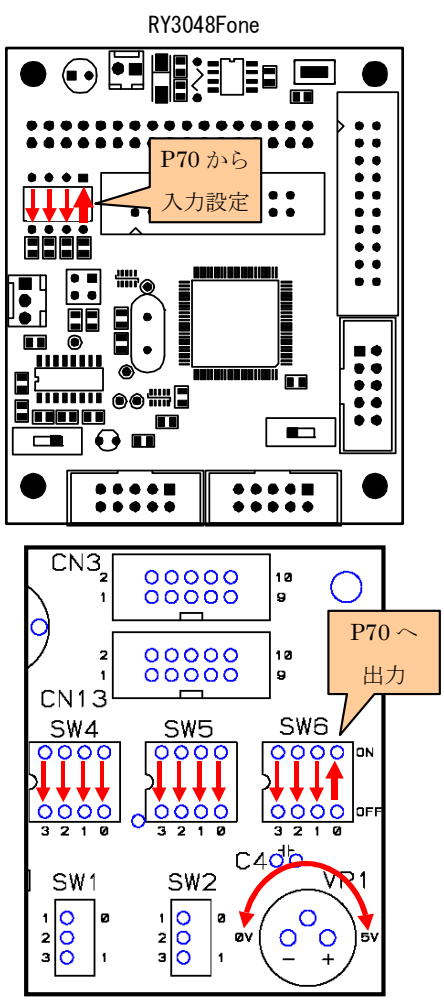
9.1 概要

実習基板のボリューム電圧をマイコンで読み込み A/D 変換します。その値を LED に出力して変換値を見てみます。マイコンのポートは下記を使用します。
・ポート 7 の 0 ビットと 1 ビット・・・アナログ電圧(0～5V)入力
・ポート A の全ビット・・・LED へデータ出力

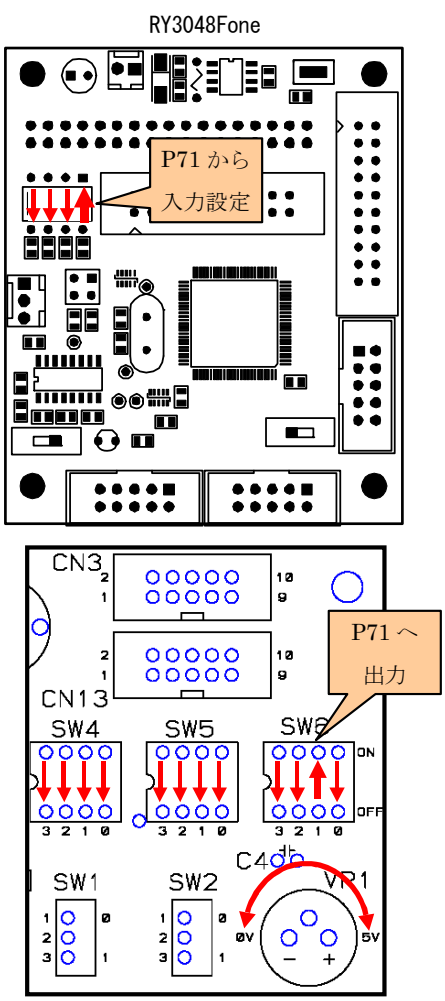
P70 と P71 のどちらの A/D 変換値を LED 出力するかは、下記のようにディップスイッチを使って決めます。
・CPU ボードのディップスイッチ P60 が OFF 側なら、P70 の A/D 変換値をポート A へ出力
・CPU ボードのディップスイッチ P60 が ON 側なら、P71 の A/D 変換値をポート A へ出力

9.2 接続

ad.c と同じです。CPU ボードのディップスイッチ P60 によって、SW6 の 0 を ON にするか、1 を ON にするか変えます。

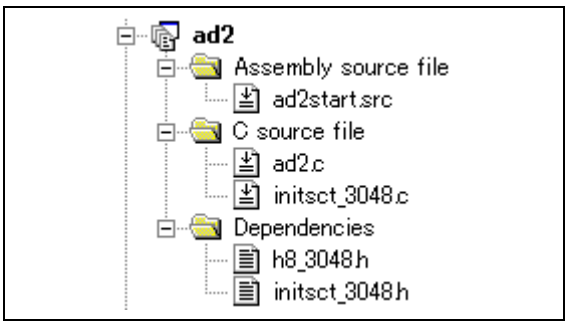


CPU ボードのディップスイッチ P60 が"1"なら、P70 からのアナログ値が読み込まれます。



CPU ボードのディップスイッチ P60 が"0"なら、P71 からのアナログ値が読み込まれます。

9.3 プロジェクトの構成



	ファイル名	内容
1	ad2start.src	アセンブリ言語で記述されたアセンブリソースファイルです。このファイルの構造は下記のようになっています。 ベクタアドレス ＋ スタートアップルーチン
2	initsct_3048.c	初期値のないグローバル変数(セクション B 領域)、初期値のあるグローバル変数(セクション R 領域)の初期化用です。
3	ad2.c	実際に制御するプログラムが書かれています。H8/3048F-ONE の内蔵周辺機能の初期化も行います。
4	h8_3048.h	H8/3048F-ONE の内蔵周辺機能の I/O レジスタを定義したファイルです。
5	initsct_3048.h	initsct_3048.c のヘッダファイルです。

9.4 プログラム「ad2.c」

```
 1 :  /************************************************************************/
 2 :  /* 2端子から電圧を入力し、ＬＥＤへ出力(ADのスキャンモード使用)「ad2.c」      */
 3 :  /* 入力：P70,P71電圧(ボリューム等)       出力：PA7-PA0(LED等)            */
 4 :  /*                              2005.04 ジャパンマイコンカーラリー実行委員会  */
 5 :  /************************************************************************/
 6 :  /*======================================*/
 7 :  /* インクルード                         */
 8 :  /*======================================*/
 9 :  #include    <machine.h>
10 :  #include    "h8_3048.h"
11 :
12 :  /*======================================*/
13 :  /* シンボル定義                         */
14 :  /*======================================*/
15 :
16 :  /*======================================*/
17 :  /* プロトタイプ宣言                     */
18 :  /*======================================*/
19 :  void init( void );
20 :  int get_ad0( void );
21 :  int get_ad1( void );
22 :
23 :  /************************************************************************/
24 :  /* メインプログラム                                                     */
25 :  /************************************************************************/
26 :  void main( void )
27 :  {
28 :      int ad;
29 :
30 :      init();                             /* 初期化                       */
31 :
32 :      while( 1 ) {
33 :          if( P6DR & 0x01 ) {
34 :              ad = get_ad0();
35 :              PADR = ad;
36 :          } else {
```

```
37 :              ad = get_ad1();
38 :              PADR = ad;
39 :          }
40 :      }
41 :  }
42 :
43 :  /************************************************************************/
44 :  /* H8/3048F-ONE 内蔵周辺機能の初期化                                    */
45 :  /************************************************************************/
46 :  void init( void )
47 :  {
48 :      /* ポートの入出力設定 */
49 :      P1DDR = 0xff;
50 :      P2DDR = 0xff;
51 :      P3DDR = 0xff;
52 :      P4DDR = 0xff;
53 :      P5DDR = 0xff;
54 :      P6DDR = 0xf0;                       /* CPU基板上のDIP SW        */
55 :      P8DDR = 0xff;
56 :      P9DDR = 0xf7;
57 :      PADDR = 0xff;                       /* LED基板                  */
58 :      PBDDR = 0xff;
59 :      /* ※ポート７は、入力専用なので入出力設定はありません    */
60 :
61 :      /* A/Dの初期設定 */
62 :      AD_CSR = 0x11;                      /* スキャンモード使用AN0-AN1*/
63 :      AD_CSR |= 0x20;                     /* ADスタート               */
64 :  }
65 :
66 :  /************************************************************************/
67 :  /* A/D値読み込み(AN0)                                                   */
68 :  /* 引数    なし                                                         */
69 :  /* 戻り値 A/D値 0～255 本当は0-1023ですが下位2bit分を無視しています     */
70 :  /************************************************************************/
71 :  int get_ad0( void )
72 :  {
73 :      return AD_DRA >> 8;
74 :  }
75 :
76 :  /************************************************************************/
77 :  /* A/D値読み込み(AN1)                                                   */
78 :  /* 引数    なし                                                         */
79 :  /* 戻り値 A/D値 0～255 本当は0-1023ですが下位2bit分を無視しています     */
80 :  /************************************************************************/
81 :  int get_ad1( void )
82 :  {
83 :      return AD_DRB >> 8;
84 :  }
85 :
86 :  /************************************************************************/
87 :  /* end of file                                                          */
88 :  /************************************************************************/
```

9.5 プログラムの解説

9.5.1 A/D変換の初期設定(スキャンモード)

ad.c では、P70 端子のみアナログ入力端子としました。ここでは、スキャンモードを使用して、P70 と P71 をアナログ入力端子として、アナログ変換値を読み込みます。プログラムは下記のとおりです。

```
62 :      AD_CSR = 0x11;                    /* スキャンモード使用 AN0-AN1*/
63 :      AD_CSR |= 0x20;                   /* AD スタート               */
```

●AD_CSR(A/D コントロール／ステータスレジスタ)の設定内容

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

・ビット 7:A/D エンドフラグ(ADF)

A/D 変換の終了を示すステータスフラグです。

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

今回は関係有りません。

・ビット 6:A/D インタラプトイネーブル(ADIE)

A/D 変換の終了による割り込み(ADI)要求の許可／禁止を選択します。

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

A/D 変換が終了したとき、割り込みを発生させることができます。今回は使いません。

・ビット 5:A/D スタート(ADST)

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

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

初期設定では関係有りません。

・ビット 4:スキャンモード(SCAN)

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

SCAN	説明
0	単一モード
1	スキャンモード

スキャンモードを使用します。

・ビット 3:クロックセレクト(CKS)

A/D 変換時間の設定を行います。

変換時間の切り替えは、ADST＝0 の状態で行ってください。

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

変換時間は最大 266 ステートとします。

・ビット 2-0:チャネルセレクト(CH2-0)

SCAN ビットと共にアナログ入力チャネルを選択します。

チャネル選択と切り替えは、ADST＝0 の状態で行ってください。

CH2	CH1	CH0	単一モード	スキャンモード
0	0	0	P70	P70
0	0	1	P71	P70、P71
0	1	0	P72	P70～P72
0	1	1	P73	P70～P73
1	0	0	P74	P74
1	0	1	P75	P74、P75
1	1	0	P76	P74～P76
1	1	1	P77	P74～P77

この設定がどの端子を使うかの設定です。スキャンモードで使用しますので、太枠で囲った部分が対象です。今回は P70 と P71 を使用するので、"001"を設定します。

スキャンモードは、1～4 端子を順番に A/D 変換していきます。このとき、使用できる端子は表のとおりです。例えば、"010"なら P70, P71, P72 端子が有効になります。P73 と P74 と P77 を使用したい、と思ってもそれはできません。回路側で表と対応するように合わせておく必要があります。

単一モードでは、A/D 変換したいときに、A/D 変換をスタートしていましたが、スキャンモードは一度設定すると、P70→P71→P70… というように自動で変換し続けます。そのため、init 関数内で、スタートしておきます。

モード(bit4)の切り替えは、bit5 が"0"のときに行います。そのため、

・62 行…初期設定

・63 行…A/D 変換スタート

というように 2 回に分けるようにします。AD_CSR の設定が 2 回続いているので、同時に設定しても良いような気がしますがそれはできません。

9.5.2 get_ad0 関数

```
71 : int get_ad0( void )
72 : {
73 :     return AD_DRA >> 8;
74 : }
```

get_ad0 関数は、P70 端子のアナログ変換結果が格納されている AD_DRA を読み込む関数です。右へ 8 ビットシフトして、0～255 の値にしています。

0～1023 の値を取得したい場合は、6 ビットだけシフトします。

9.5.3 get_ad1 関数

```
81 : int get_ad1( void )
82 : {
83 :     return AD_DRB >> 8;
84 : }
```

get_ad1 関数は、P71 端子のアナログ変換結果が格納されている AD_DRB を読み込む関数です。右へ 8 ビットシフトして、0～255 の値にしています。

0～1023 の値を取得したい場合は、6 ビットのシフトにします。

9.5.4 main関数

```
26 : void main( void )
27 : {
28 :     int ad;
29 :
30 :     init();                              /* 初期化                    */
31 :
32 :     while( 1 ) {
33 :         if( P6DR & 0x01 ) {
34 :             ad = get_ad0();
35 :             PADR = ad;
36 :         } else {
37 :             ad = get_ad1();
38 :             PADR = ad;
39 :         }
40 :     }
41 : }
```

33 行目で、P6DR の bit0 をチェックしています。これは、CPU ボード上のディップスイッチをチェックしています。

値が 1 なら(0 以外なら)、34～35 行目を実行します。1 というのは、ディップスイッチの P60 が OFF の状態です。P70 端子のアナログ電圧を読み込んでポート A に繋がっている LED へ出力します。

それ以外なら、すなわち値が 0 なら 37～38 行目を実行します。0 というのは、ディップスイッチの P60 が ON の状態です。P71 端子のアナログ電圧を読み込んでポート A に繋がっている LED へ出力します。

9.6 まとめ

設定するレジスタ	詳細
AD_CSR	A/D 変換の初期設定と、どの端子を A/D 入力端子とするか設定します。 0x10…70 端子　　0x11…P70～P71 端子 0x12…P70～P72 端子　　0x13…P70～P73 端子 0x14…74 端子　　0x15…P74～P75 端子 0x16…P74～P76 端子　　0x17…P74～P77 端子 上記設定後、 AD_CSR |= 0x20; として A/D 変換をスタートします。スタート後は A/D 変換は自動で繰り返し行われ、AD_DRA、AD_DRB、AD_DRC、AD_DRD の値が更新されていきます。

A/D 値の読み込みは、下記のように行います。

```
71 : int get_ad0( void )
72 : {
73 :     return AD_DRA >> 8;          ←レジスタより A/D 値の読み込み
74 : }                                  8 ビット右シフトなら 0～255 の値、
                                        6 ビット右シフトなら 0～1023 の値が返ってくる
```

73 行の読み込むレジスタは、
・P70 か P74 の端子を使用している場合→AD_DRA から読み込み
・P71 か P75 の端子を使用している場合→AD_DRB から読み込み
・P72 か P76 の端子を使用している場合→AD_DRC から読み込み
・P73 か P77 の端子を使用している場合→AD_DRD から読み込み
と、使用する端子により変わります。
AD_CSR の初期設定(62 行)が 0x11 の場合、P70 端子と P71 端子を使用する設定なので、AD_DRA と AD_DRB レジスタから A/D 値を読み込みます。

10. プロジェクト「pwm1」 ITU3(または 4)を使ったPWM信号出力

10.1 概要

ITU3 を使って PWM 波形を端子に出力します。周期は、16[ms]に設定します。デューティ比は、CPU ボード上のディップスイッチにより 16 段階に切り替えることができます。

本プログラムを改造して、ITU4 と置き換えることもできます。ITU0～2 を使用して PWM 出力する場合は、pwm2.c を参照してください。

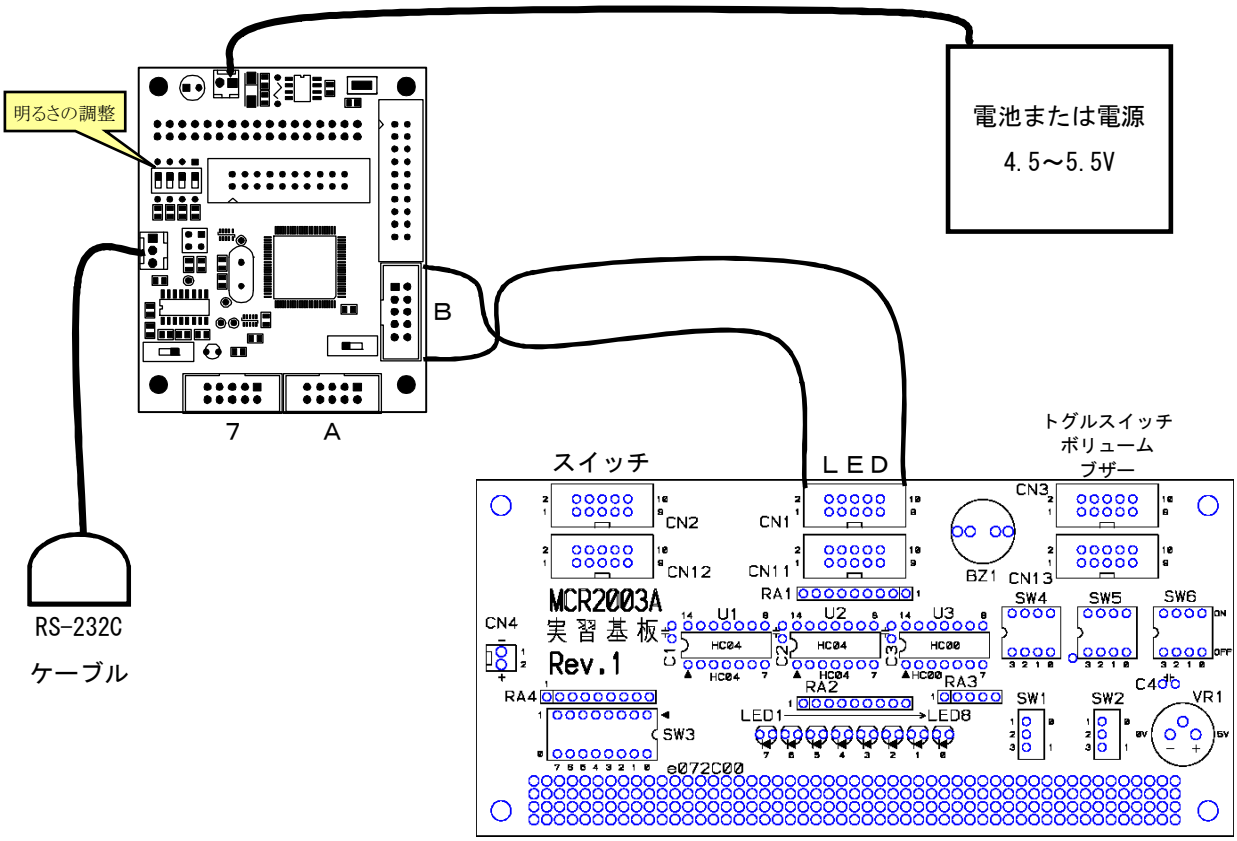
マイコンのポートは下記を使用します。

・ポート B の 0 ビット・・・LED へデータ出力(PWM 出力)

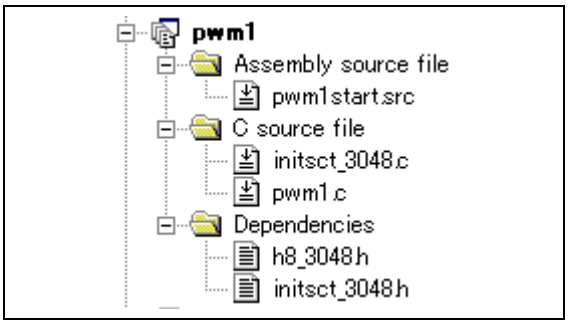
10.2 接続

・CPU ボードのポート B と実習基板の LED 部を、フラットケーブルで接続します。

※LED の明るさの調整は、CPU ボードのディップスイッチで行います。



10.3 プロジェクトの構成



	ファイル名	内容
1	pwm1start.src	アセンブリ言語で記述されたアセンブリソースファイルです。このファイルの構造は下記のようになっています。 ベクタアドレス ＋ スタートアップルーチン
2	initsct_3048.c	初期値のないグローバル変数(セクション B 領域)、初期値のあるグローバル変数(セクション R 領域)の初期化用です。
3	pwm1.c	実際に制御するプログラムが書かれています。H8/3048F-ONE の内蔵周辺機能の初期化も行います。
4	h8_3048.h	H8/3048F-ONE の内蔵周辺機能の I/O レジスタを定義したファイルです。
5	initsct_3048.h	initsct_3048.c のヘッダファイルです。

10.4 プログラム「pwm1.c」

```
 1 :  /************************************************************************/
 2 :  /* PWM波形出力(ITU3またはITU4使用)「pwm1.c」                              */
 3 :  /* 入力:CPUボードのディップスイッチ(P63-P60)    出力:PB0(LED等)           */
 4 :  /*                          2005.04 ジャパンマイコンカーラリー実行委員会  */
 5 :  /************************************************************************/
 6 :  /*====================================*/
 7 :  /* インクルード                       */
 8 :  /*====================================*/
 9 :  #include    <machine.h>
10 :  #include    "h8_3048.h"
11 :
12 :  /*====================================*/
13 :  /* シンボル定義                       */
14 :  /*====================================*/
15 :
16 :  /*====================================*/
17 :  /* プロトタイプ宣言                   */
18 :  /*====================================*/
19 :  void init( void );
20 :  unsigned char dipsw_get( void );
21 :
22 :  /************************************************************************/
23 :  /* メインプログラム                                                     */
24 :  /************************************************************************/
25 :  void main( void )
26 :  {
27 :      init();                             /* 初期化                   */
28 :
29 :      while( 1 ) {
30 :          ITU3_BRB = 49150 * dipsw_get() / 15;
31 :      }
32 :  }
33 :
34 :  /************************************************************************/
35 :  /* H8/3048F-ONE 内蔵周辺機能の初期化                                    */
36 :  /************************************************************************/
37 :  void init( void )
38 :  {
```

```
39 :      /* ポートの入出力設定 */
40 :      P1DDR = 0xff;
41 :      P2DDR = 0xff;
42 :      P3DDR = 0xff;
43 :      P4DDR = 0xff;
44 :      P5DDR = 0xff;
45 :      P6DDR = 0xf0;                     /* CPU基板上のDIP SW        */
46 :      P8DDR = 0xff;
47 :      P9DDR = 0xf7;
48 :      PADDR = 0xff;
49 :      PBDDR = 0xff;                     /* LED基板                  */
50 :      /* ポート7は、入力専用なので入出力設定はありません    */
51 :
52 :      ITU3_TCR = 0x23;                  /* カウンタ、クリアの設定   */
53 :      ITU_FCR  = 0x02;                  /* バッファの設定           */
54 :      ITU3_GRA = 49151;                 /* PWM周期                  */
55 :      ITU3_GRB = ITU3_BRB = 0;          /* デューティ比設定         */
56 :      ITU_MDR  = 0x08;                  /* PWMモード設定            */
57 :      ITU_STR  = 0x08;                  /* タイマスタート           */
58 :  }
59 :
60 :  /************************************************************************/
61 :  /* ディップスイッチ値読み込み                                           */
62 :  /* 戻り値 スイッチ値 0～15                                              */
63 :  /************************************************************************/
64 :  unsigned char dipsw_get( void )
65 :  {
66 :      unsigned char sw;
67 :
68 :      sw  = ~P6DR;                      /* ディップスイッチ読み込み */
69 :      sw &= 0x0f;
70 :
71 :      return  sw;
72 :  }
73 :
74 :  /************************************************************************/
75 :  /* end of file                                                          */
76 :  /************************************************************************/
```

10.5 プログラムの解説

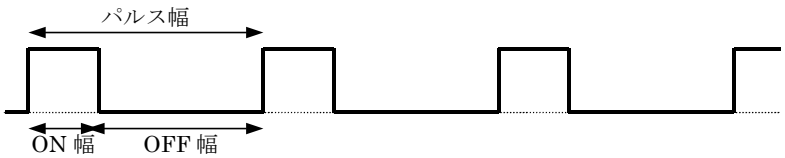
10.5.1 PWMとは？

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。

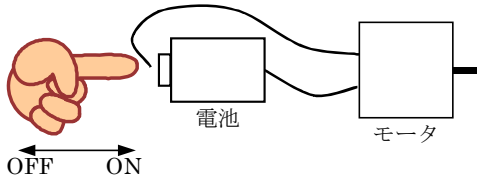
ボリュームを使えば電圧を可変することができます。しかし、モータへは大電流が流れるため、非常に大きな抵抗が必要です。また、モータに加えなかった分は、抵抗の熱となってしまいます。

そこで、スイッチで ON、OFF 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」と言います。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＝ON 幅／パルス幅(ON幅＋OFF幅)
です。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。

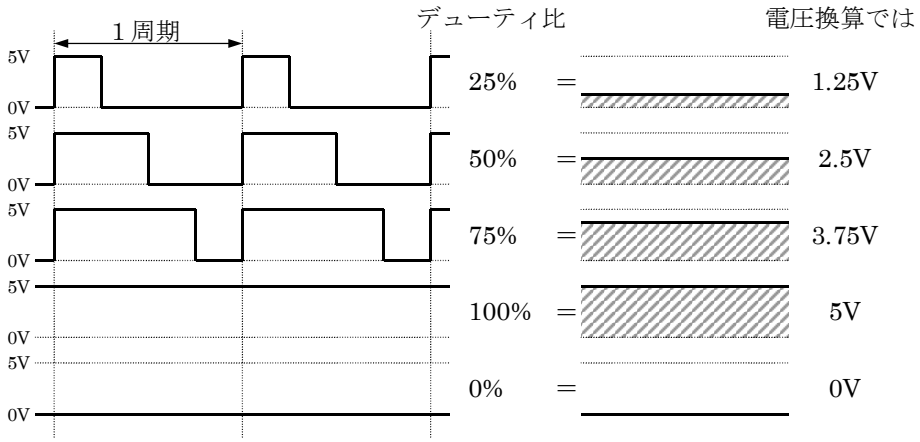
「PWM」と聞くと、何か難しく感じてしまいますが、下記のように手でモータと電池の線を「繋ぐ」、「離す」の繰り返し、それも PWM と言えます。繋いでいる時間が長いとモータは速く回ります。離している時間が長いとモータは少ししか回りません。人なら「繋ぐ」、「離す」の動作をコンマ数秒でしか行えませんが、マイコンなら数ミリ秒で行うことができます。



下図のように、0V と 5V を出力するような波形で考えてみます。1周期に対して ON の時間が長ければ長いほど平均化した値は大きくなります。すべて 5V にすればもちろん平均化しても 5V、これが最大の電圧です。ON の時間を半分の 50%にするとどうでしょうか。平均化すると 5V×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。CPU 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。CPU 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**。

10.5.2 ITUを使ったPWM信号出力

「ITU」とは、「16 ビットインテグレーテッドタイマユニット」の略称です。H8/3048F-ONE には ITU0～4 の 5 チャネルが内蔵されています。

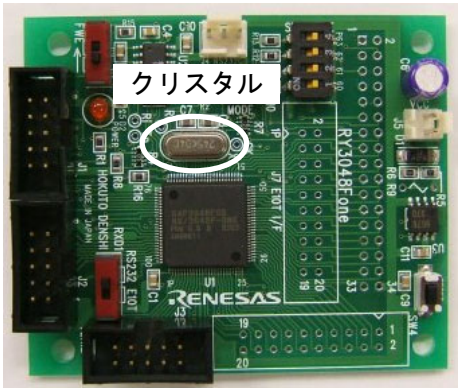
ITU に PWM の設定をいったん行えば、あとは自動的に PWM 波形を出力してくれるため便利です。ITU を使わないとどうなるのでしょう。1ms 周期、0.6ms が ON、0.4ms が OFF の PWM 波形をプログラム的に作るとすると、端子を"1"にして 0.6ms 待つ、端子を"0"にして 0.4ms 待つ、ということを繰り返します。これだけなら良いのですが、PWM を 2 つ、3 つ、更にはセンサをチェックしたり、エンコーダのカウント値をチェックするなど、他の制御が加わると正確に 0.6ms と時間を計ることはできません。その点、ITU では、いったん設定してしまえばプログラムとは独立して PWM 波形が自動的に出力するので、設定した時間きっちりの PWM 波形を出力することができます。

ITU を PWM モードに設定したとき、その波形が出力される端子は決まっています。残念ながら出力端子の変更はできません。

ITU のチャネル	PWM 波形出力端子
0	ポート A の bit2 端子
1	ポート A の bit4 端子
2	ポート A の bit6 端子
3	ポート B の bit0 端子
4	ポート B の bit2 端子

今回は ITU3 を使用しますので、PB0 端子から PWM 波形が出力されます。

10.5.3 時間の単位



人間の世界では、時間の単位は秒です。1 秒はセシウム原子の共鳴周波数の 9192631770Hz 分(約 92 億ヘルツ)と決められています。

マイコンにも時間の単位があります。これはマイコンに繋がっているクリスタルの値になります。RY3048Fone ボードには「24.576 MHz」のクリスタルが搭載されています。パルス幅は周波数の逆数なので

パルス幅＝1／φ＝1/24.476×10^6＝40.69[ns]

がマイコンの動作する基準の時間となります。「内部クロック」や「φ」という用語が出てきますが、これはクリスタルの値である「24.576MHz」、時間に直すと「40.69ns」のことです。

クリスタルの誤差は、±100ppm です。ppm は百万分の 1 ですので、100/1,000,000 の誤差、1/10,000 の誤差、0.01%となります。非常に正確です。

10.5.4 ITUの初期設定

```
52 :      ITU3_TCR = 0x23;                /* カウンタ、クリアの設定   */
53 :      ITU_FCR  = 0x02;                /* バッファの設定           */
54 :      ITU3_GRA = 49151;               /* PWM周期                  */
55 :      ITU3_GRB = ITU3_BRB = 0;        /* デューティ比設定         */
56 :      ITU_MDR  = 0x08;                /* PWMモード設定            */
57 :      ITU_STR  = 0x08;                /* タイマスタート           */
```

この 6 行を設定することにより、該当する PWM 端子から PWM 信号が出力されます。今回は ITU3 を設定していますので、PB0 から PWM 信号が出力されます。これからどのレジスタがどのような意味を持つか説明していきます。

●レジスタ

これから出てくるレジスタと意味です。

レジスタ	名称	意味
ITU3_CNT	タイマカウンタ	パルス数をカウントするレジスタです。値によって経過時間が分かります。
ITU3_TCR	タイマコントロールレジスタ	ITU3_CNT を0にするタイミングや ITU3_CNT がカウントするパルス幅(1/1 , 1/2 , 1/4 , 1/8)を選択します。
ITU3_GRA	ジェネラルレジスタ A	ジェネラルは一般的な、全般的な、という意味ですがここでは汎用的な、という意味です。汎用とは、「広くいろいろな方面に用いること」です。ITU3_GRA と ITU3_CNT は、自動的に比較されています。**PWM モードで使用するときは、値が一致すると PWM 端子を"1"にします**。
ITU3_GRB	ジェネラルレジスタ B	ITU3_GRB と ITU3_CNT は、自動的に比較されています。**PWM モードで使用するときは、値が一致すると PWM 端子を"0"にします**。
ITU_MDR	タイマモードレジスタ	ITU を PWM モードで使用するときに設定します。
ITU_STR	タイマスタートレジスタ	ITU3_CNT へのパルス入力は、最初はされていません。ITU3_CNT へパルス入力を ON にするのがこのレジスタです。スイッチのような役割です。
ITU_FCR	タイマファンクション コントロールレジスタ	バッファを使用するかしないかの設定を行います。

●ITU3_TCR(タイマコントロールレジスタ)の設定内容

このレジスタは、ITU3_CNT(カウンタ)をどういう動きにするかを設定するレジスタです。ITU3_TCR では、
・ITU3_CNT を0にするタイミング
・ITU3_CNT が＋1する時間(加えるパルス幅)
を設定します。

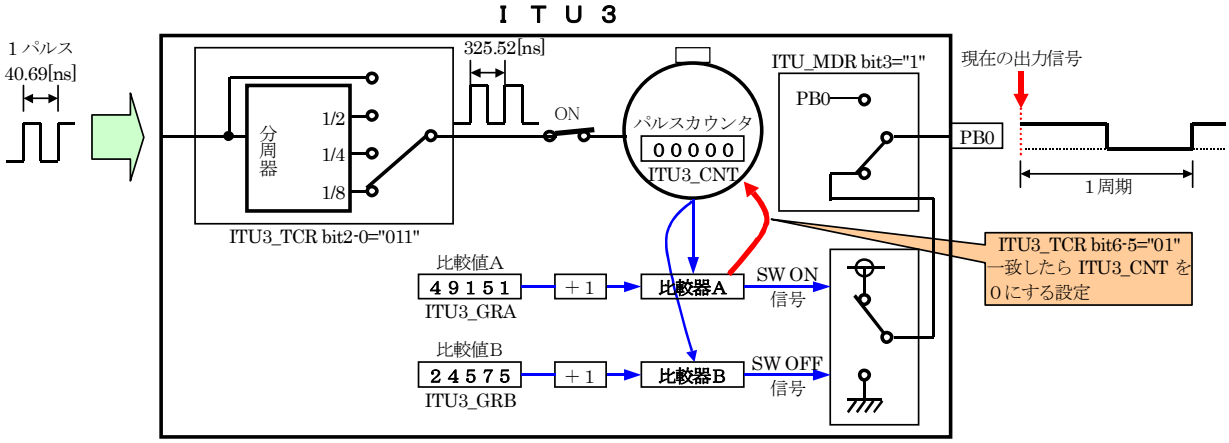
ITU3_TCR の各ビットの意味は、下記のようになっています。

ビット:	7	6	5	4	3	2	1	0
ITU3_TCR:	－	CCLR1	CCLR0	CKEG1	CKEG0	TPSC2	TPSC1	TPSC0
設定値:	0	0	1	0	0	0	1	1
16 進数:	2				3			

・ビット 6,5:カウンタクリア 1,0
CNT のカウンタクリア要因を選択します。

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

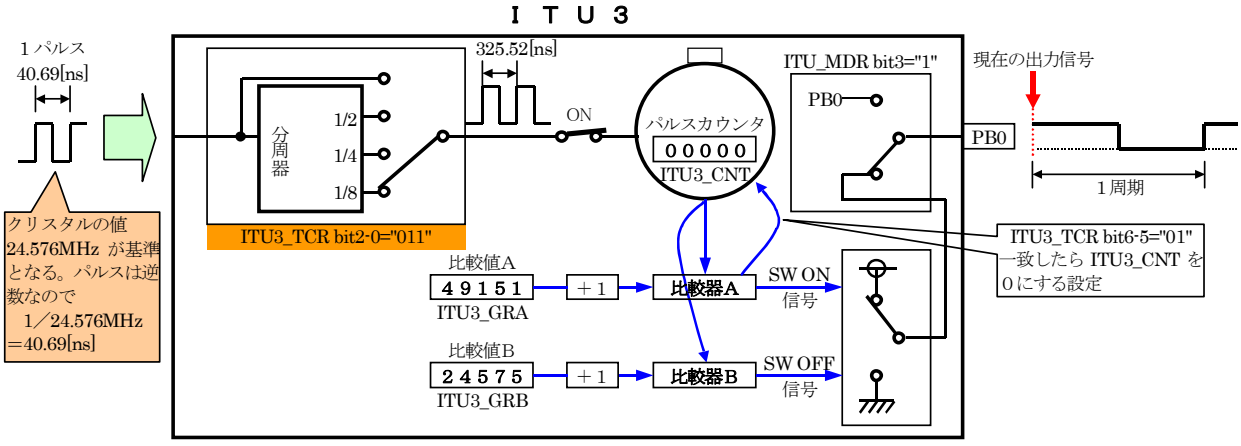
この設定が、「ITU3_CNT を0にするタイミング」です。コンペアマッチとは、コンペア＝比較、マッチ＝一致、ですので、ITU3_CNT と(ITU3_GRA＋1)が一致すると ITU3_CNT をクリアにしなさい、という設定にします。なぜ ITU3_GRA ではなく、(ITU3_GRA＋1)かというと、ここが分かりづらい部分なのですが、「ITU の決まり事」としか言いようがありません。一致したと判断するのは「ITU3_CNT＝(ITU3_GRA＋1)」のときです。したがって、ITU3_CNT が 1000 になったら何かをさせたい場合、ITU3_GRA には 999 を入れます。ITU3_CNT が(999+1)かどうか常に比較して、一致すると ITU0_CNT をクリアします。



・ビット 2～0：タイマプリスケーラ 2～0

CNT のカウントクロックを選択します。

TPSC2	TPSC1	TPSC0	説明
0	0	0	内部クロック：φでカウント　　要はそのまま出力
0	0	1	内部クロック：φ／2でカウント　要はパルス幅を 2 倍にして出力
0	1	0	内部クロック：φ／4でカウント　要はパルス幅を 4 倍にして出力
0	**1**	**1**	**内部クロック：φ／8でカウント　要はパルス幅を 8 倍にして出力**
1	0	0	外部クロックA：TCLKA 端子(PA0)でカウント
1	0	1	外部クロックB：TCLKB 端子(PA1)でカウント
1	1	0	外部クロックC：TCLKC 端子(PA2)でカウント
1	1	1	外部クロックD：TCLKD 端子(PA3)でカウント



ITU3_CNT は、入力されたパルスを数える機能があります。パルス幅は、クリスタルの値である「40.69[ns]」です。すなわち 40.69[ns]ごとに＋1します。その他に、設定で 2 倍、4 倍、8 倍のパルス幅に変換して、それを ITU3_CNT へ加えることができます。その設定が、ITU3_TCR の bit2～0 になります。

ITU3_CNT は符号無し 16 ビット幅ですので 0～65535 までカウントすることができます。すなわち 65535 のときが計測できる最大時間です。パルスを 1 倍、2 倍、4 倍、8 倍にしたときに計測できる最大時間を計算してみます。

TPSC2	TPSC1	TPSC0	説明
0	0	0	内部クロック: φ でカウント 24.576MHz でカウント、時間に直すと逆数なので $1 \div (24.576 \times 10^6) = 40.69$[ns] 計測できる最大時間は 40.69[ns]×65535＝2.666[ms]
0	0	1	内部クロック: φ ／2でカウント $1 \div (24.576 \times 10^6 / 2) = 81.38$[ns] 計測できる最大時間は 81.38[ns]×65535＝5.333[ms]
0	1	0	内部クロック: φ ／4でカウント $1 \div (24.576 \times 10^6 / 4) = 162.76$[ns] 計測できる最大時間は 162.76[ns]×65535＝10.67[ms]
0	1	1	内部クロック: φ ／8でカウント $1 \div (24.576 \times 10^6 / 8) = 325.52$[ns] 計測できる最大時間は 325.52[ns]×65535＝21.33[ms]

これから詳しく説明しますが、ITU3_CNT の値が○○になったら端子を"1"にしなさい、○○になったら端子を"0"にしなさい、という方法で PWM 信号を作ります。そのため、ITU3_CNT が 65535 以下で設定できる周期である必要があります。それ以上になると、0 に戻ってしまい比較ができないためです。

4つの設定がありますが、選び方は、
・PWM 周期が 2.666[ms]以下なら、TPSC="000"にします。
・PWM 周期が 5.333[ms]以下なら、TPSC="001"にします。
・PWM 周期が 10.67[ms]以下なら、TPSC="010"にします。
・PWM 周期が 21.33[ms]以下なら、TPSC="011"にします。
・PWM 周期が 21.33[ms]以上なら、設定できません。
今回は、周期を 16[ms]に設定するので"011"を設定します。ITU3_CNT は 325.52[ns]ごとに＋1します。

ITU3_CNT を0にするタイミングと ITU3_CNT が一定間隔で＋1する時間の設定を合わせると、0x23 となります。

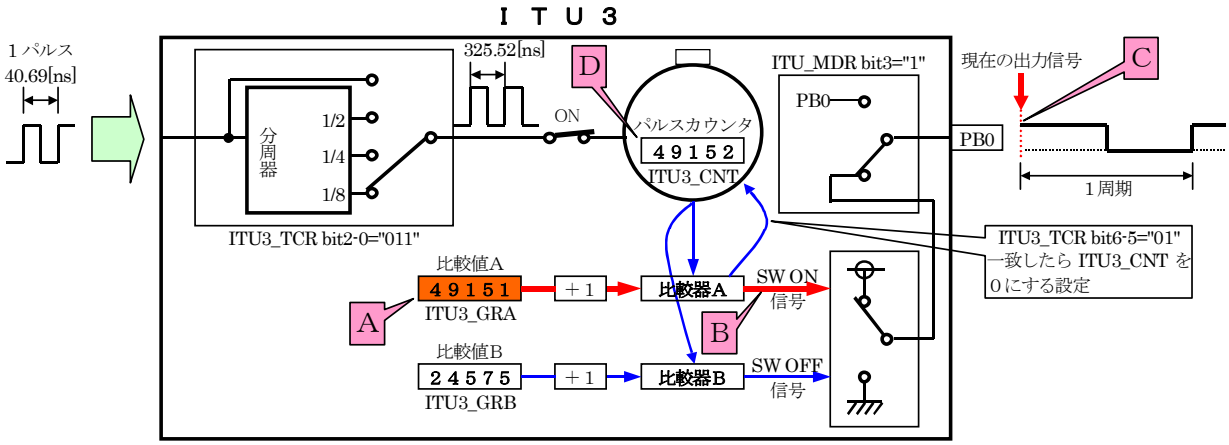
```
52 :        ITU3_TCR = 0x23;                    /* カウンタ、クリアの設定    */
```

●ITU3_GRA(ジェネラルレジスタ A)の設定内容

ジェネラルは「汎用的な」という意味です。汎用とは、「広くいろいろな方面に用いること」です。ITU3_GRA と ITU3_CNT は、自動的に比較されていて、値が一致すると割り込みを発生させたり、PWM の信号を変えたりと、設定によっていろいろな用途で使用されます。そのため、ジェネラルと呼ばれます。ジェネラルレジスタは2つありそれぞれ A と B という名前が付いています。ITU3_GRA を比較値 A と呼ぶと分かりやすくなります。

比較値 A には、PWM の周期を設定します。
今回は、周期を 16[ms]にします。パルスカウンタである ITU3_CNT は 325.52[ns]ごとにカウントアップされますので、16[ms]がたつまでのカウント数は
$(16 \times 10^{-3}) \div (325.52 \times 10^{-9}) = 49152$
となります。実際の比較は「ITU3_CNT＝(ITU3_GRA＋1)」で行われますので、比較値 A である ITU3_GRA には1小さい値「**49151**」を設定します(A部分)。

比較器Aでは、ITU3_CNT の値と ITU3_GRA の値を常に比較してします。「ITU3_CNT＝(ITU3_GRA＋1)」になると、スイッチ ON 信号をスイッチに出力します(B部分)。PWM 端子は"1"になります(C部分)。



ITU3_TCR では、「GRA のコンペアマッチ／インプットキャプチャで CNT をクリア」に設定しています。「ITU3_CNT＝(ITU3_GRA＋1)＝49152」になると、ITU3_CNT が 0 になります(D部分)。要は、「0→1→2…49150→49151→0→1→2…」となります。図では ITU3_CNT の値が 49152 になっていますが、49152 になる瞬間に 0 になるため、**実際には 49152 になることはありません**。

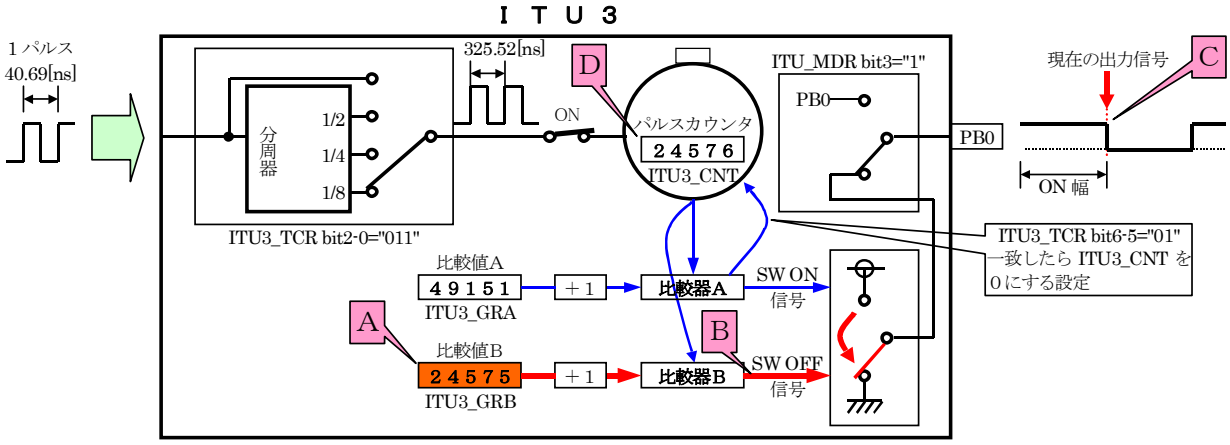
下記が実際にプログラムで設定している部分です。**ITU3_GRA には周期を設定します**。

```
54 :        ITU3_GRA = 49151;                    /* PWM周期                    */
```

●ITU3_GRB(ジェネラルレジスタ B)の設定内容

ITU3_GRB は、比較値 B と言い換えると分かりやすくなります。比較値 B には、PWM の ON 幅を設定します。
今回、ON 幅は 8[ms]とします。パルスカウンタである ITU3_CNT は 325.52[ns]ごとにカウントアップされますので、8[ms]がたつまでのカウント数は
$(8\times10^{-3})\div(325.52\times10^{-9})=24576$
となります。実際の比較は「ITU3_CNT＝(ITU3_GRB＋1)」で行われますので、比較値 B である ITU3_GRB には1小さい値「24575」を設定します(A部分)。

比較器Bでは、ITU3_CNT の値と ITU3_GRB の値を常に比較してします。「ITU3_CNT＝(ITU3_GRB＋1)」になると、スイッチ OFF 信号をスイッチに出力します(B部分)。PWM 端子は"0"になります(C部分)。



下記が実際にプログラムで設定している部分です。**ITU3_GRB には ON 幅を設定します**。ITU3_BRB については後述します。

```
55 :        ITU3_GRB = ITU3_BRB = 0;              /* デューティ比設定          */
```

●ITU_MDR(タイマモードレジスタ)の設定内容

ITU を PWM 出力として使用するかどうかを設定します。ITU_MDR は今までの ITU3_CNT のように、数字が付きません。これは、ITU_MDR　1つで ITU0～4 まで共通の設定のためです。

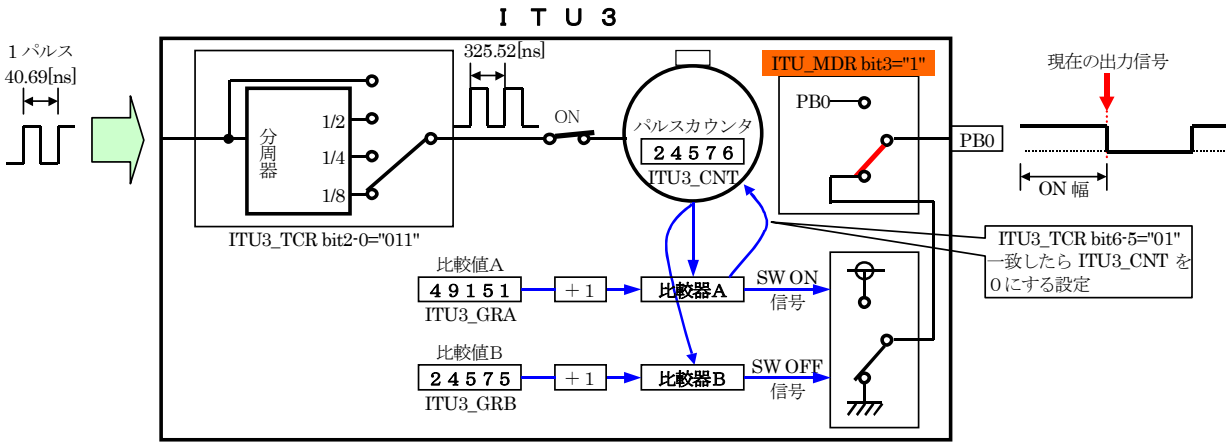
ビット:	7	6	5	4	3	2	1	0
ITU_MDR:	－	MDF	FDIR	ITU4 を PWM モードにする	ITU3 を PWM モードにする	ITU2 を PWM モードにする	ITU1 を PWM モードにする	ITU0 を PWM モードにする
設定値:	0	0	0	0	1	0	0	0
16 進数:	0				8			

・ビット 4～0:PWM モード 4～0

チャネル x を通常動作させるか、PWM モードで動作させるかを選択します。

PWM x	説明
0	チャネル x は通常動作
1	チャネル x は PWM モード

※ x は 0～4



ここまでの設定では、PWM 端子から PWM 信号が出力されていません。ITU_MDR を設定することにより、はじめて ITU3 の出力端子である PB0 から、PWM 信号が出力されます。ITU のチャネルと PWM 波形出力端子の関係を下表に示します。対応は決まっていて、必ず下表のようになります。例えば、ポート B の bit5 から PWM 波形を出力する設定はできません。

ITU のチャネル	PWM 波形出力端子
0	ポート A の bit2 端子
1	ポート A の bit4 端子
2	ポート A の bit6 端子
3	ポート B の bit0 端子
4	ポート B の bit2 端子

プログラムは下記のようになります。

```
56 :        ITU_MDR  = 0x08;                    /* PWMモード設定              */
```

●ITU_STR(タイマスタートレジスタ)の設定内容

ビット:	7	6	5	4	3	2	1	0
ITU_STR:	−	−	−	ITU4_CNT のカウント開始	ITU3_CNT のカウント開始	ITU2_CNT のカウント開始	ITU1_CNT のカウント開始	ITU0_CNT のカウント開始
設定値:	0	0	0	0	1	0	0	0
16 進数:	0				8			

・ビット 4-0:カウンタスタート 4～0

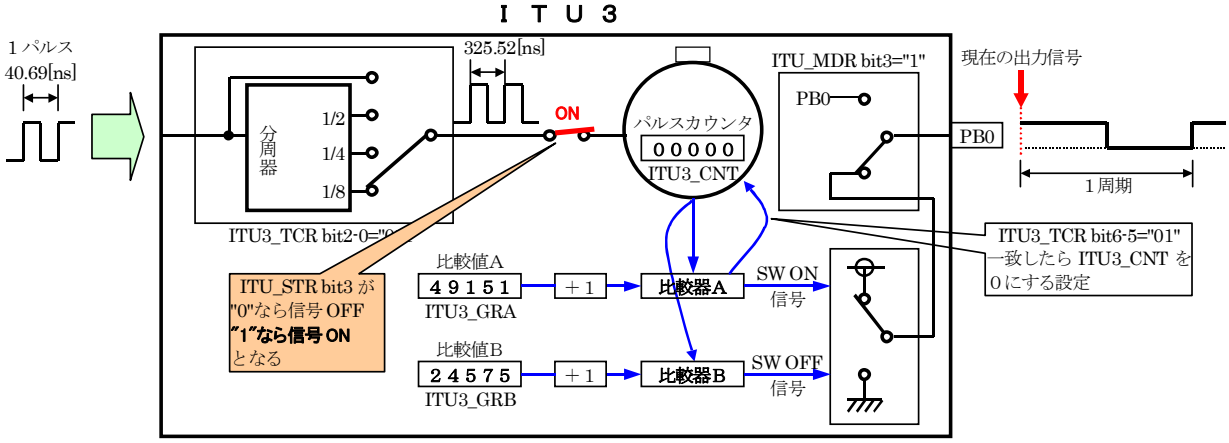
タイマカウンタ x の動作／停止を選択します。

STR4-0	説明
0	ITUx の CNT のカウント動作は停止
1	ITUx の CNT はカウント動作

※ x は 0～4

ITU3_CNT は、まだカウントし始めていません。ITU3_CNT へパルスを加えるか加えないかを決めるレジスタが ITU_STR レジスタです。このレジスタを設定することにより ITU3_CNT は動き出します。ITU_STR は今までの ITU3_CNT のように、数字が付きません。これは、ITU_STR 1つで ITU0～4 まで共通の設定のためです。

今回は ITU3 をスタートさせますので ITU_STR の bit3 を”1”にします。



ITU_STR の bit3 は、ITU3_CNT へ加えるパルス信号線のスイッチのような役割です。

```
57 :        ITU_STR  = 0x08;                    /* タイマスタート             */
```

10.5.5 ITUをPWM機能として使用するポイント

ITU を PWM 機能として使用するに当たり、予期せぬ動作をすることがあります。予期せぬ動作をしないようにプログラマ側で対策しなければいけません。この辺が分かりづらくしている部分でもあります。

(1) ITUx_GRAとITUx_GRBが一致している場合の対処

ポイント1…ITUx_GRA と ITUx_GRB が一致している場合、該当の PWM 出力端子は変化しない

周期 16ms とします。ITU3_CNT に加えるパルス幅を 325.52[ns]とすると、ITU3_GRA に設定する値は、

ITU3_GRA＝(16×10^{-3})÷(325.52×10^{-9})－1＝49151

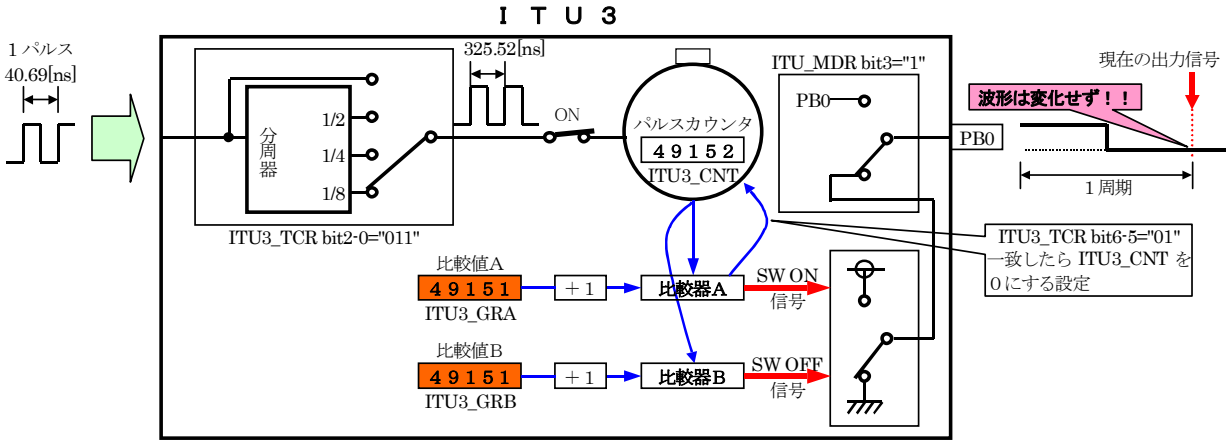
となります。

ON 幅も 16ms、要は 100%の波形を出力したいとします。ITU3_GRB に設定する値は、

ITU3_GRB＝(16×10^{-3})÷(325.52×10^{-9})－1＝49151

となります。

ITU3_GRA と ITU3_GRB の値が同じになりました。実は、下記のような予期せぬ動作になってしまいます。



ITU3_CNT が 49152 になりました。

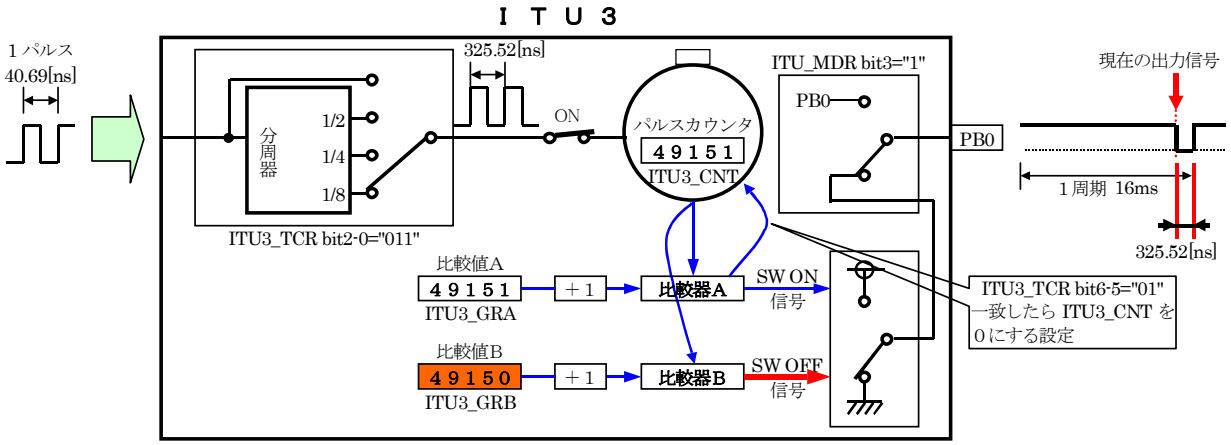
ITU3_CNT=(ITU3_GRA+1)になったので、ON 信号を出力して、PWM 波形を"1"にします。

ITU3_CNT=(ITU3_GRB+1)になったので、OFF 信号を出力して、PWM 波形を"0"にします。

同時に、ON 信号、OFF 信号が出力されてしまいました。どうなるのでしょうか。このときは、**波形は変化しません**。これでは、100%出力にならず、問題ありです。

そこで、100%にしたいときは、ITU3_GRB の値を ITU3_GRA より 1 小さい値にします。

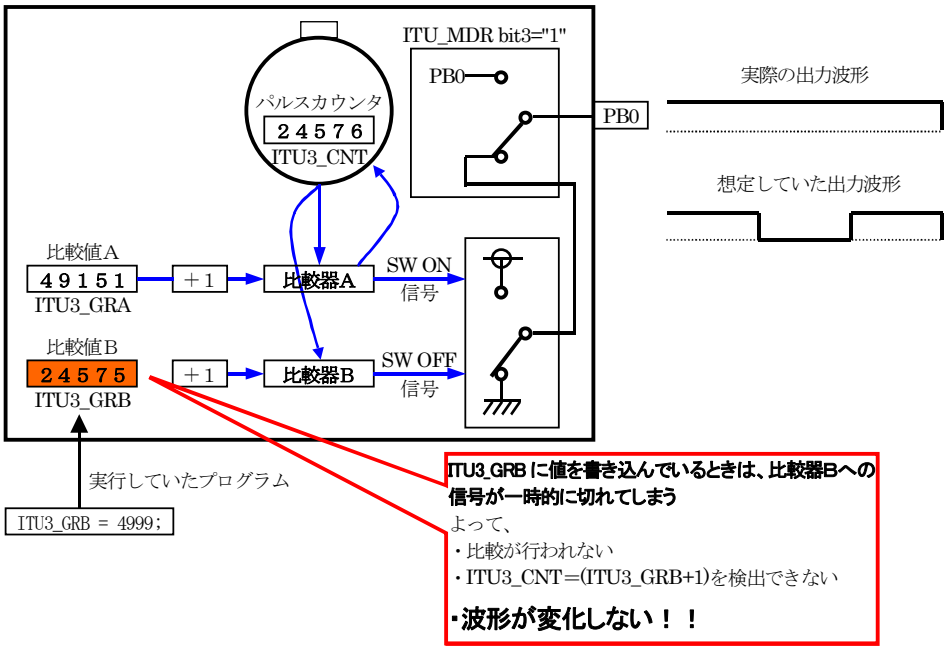
ITU3_CNT=(ITU3_GRB+1)で波形が"0"になります。1つ増えるとすぐに ITU3_CNT=(ITU3_GRA+1)となり波形が"1"になります。1カウント分の 325.52[ns] OFF になりますが、これを 100%とします。



(2) ITUx_GRAやITUx_GRBのレジスタに値を書き換え中に、タイマカウンタとレジスタが一致した場合の対処

ポイント2 … ITUx_GRA や ITUx_GRB のレジスタに値を書き換え中に、タイマカウンタとレジスタが一致した場合、信号は変化しません

これは大変な問題です。実際に問題が起こったときの状態を図解してみます。

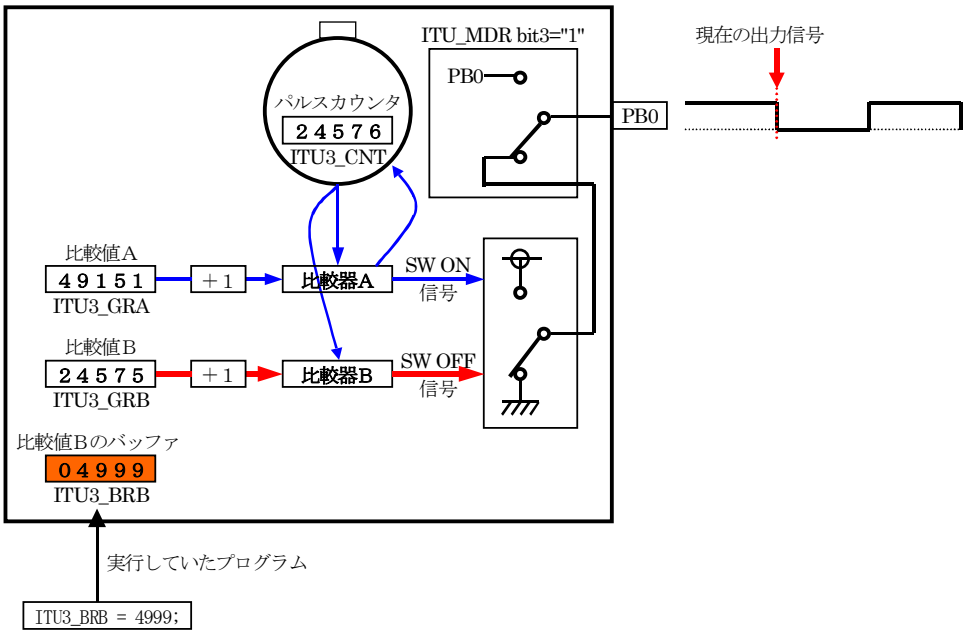


これを防止する機能が備わっています。それがバッファレジスタというレジスタです。

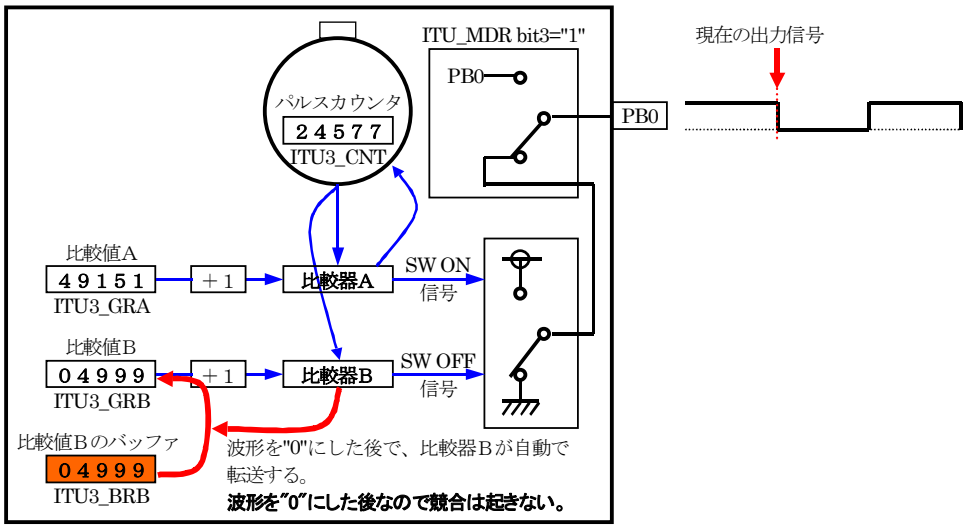
レジスタ名	説明
ITU3_BRA	ITU3_GRA のバッファ(ITU3バッファレジスタ A)です。今回は、ITU3_GRA は周期設定用でいったん設定すると変えることはないのでバッファにしません。
ITU3_BRB	ITU3_GRB のバッファ(ITU3バッファレジスタ B)です。プログラムでは ITU3_GRB へ値を書き込まずに ITU3_BRB へ書き込み、競合を回避します。
ITU4_BRA	ITU4_GRA のバッファ(ITU4バッファレジスタ A)です。ITU4 を使用するときに使います。プログラムでは ITU4_GRA へ値を書き込まずに ITU4_BRA へ書き込み、競合を回避します。
ITU4_BRB	ITU4_GRB のバッファ(ITU4バッファレジスタ B)です。ITU4 を使用するときに使います。プログラムでは ITU4_GRB へ値を書き込まずに ITU4_BRB へ書き込み、競合を回避します。

バッファレジスタを使用に設定すると波形が変化した後に、**自動でバッファレジスタの値がジェネラルレジスタへ転送されます**。メインプログラムはあくまでもバッファレジスタに値を書き込んでいますので、競合が起きることは有りません。

プログラムでは、ITU3_BRB に値を書き込みます。ITU3_GRB には書き込んでいないので競合が起きることはありません(下図)。



「ITU3_CNT=ITU3_GRB+1」を検出して波形を"0"にしてから、ITU3_BRB の値を ITU3_GRB に**自動で転送します**(下図)。



この「**自動で転送する**」設定が、ITU_FCR です。

●ITU_FCR(タイマファンクションコントロールレジスタ)の設定内容

ビット:	7	6	5	4	3	2	1	0
ITU_FCR:	−	−	CMD1	CMD0	BFB4	BFA4	BFB3	BFA3
設定値:	0	0	0	0	0	0	1	0
16 進数:	0				2			

・ビット 5,4:コンビネーションモード 1,0

チャネル 3,4 を通常動作させるか、相補 PWM モードまたはリセット同期 PWM モードで動作させるかを選択します。

CMD1	CMD0	説明
0	0	チャネル 3,4 は通常動作
0	1	
1	0	チャネル 3,4 を組み合わせ、相補 PWM モードで動作
1	1	チャネル 3,4 を組み合わせ、リセット同期 PWM モードで動作

ITU3 と 4 を組み合わせて、相補 PWM モードやリセット同期 PWM モードに設定することができます。今回は通常動作です。通常動作といっても、ITU_MDR を設定すれば PWM 出力に設定できます。

・ビット 3:バッファ動作 B4(BFB4)

ITU4_GRB を通常動作とするか、ITU4_GRB と ITU4_BRB を組み合わせてバッファ動作とするかを選択します。

BFB4	説明
0	ITU4_GRB は通常動作
1	ITU4_BRB はバッファ動作

ITU4 は使いません。通常動作です。

・ビット 2:バッファ動作 A4(BFA4)

ITU4_GRA を通常動作とするか、ITU4_GRA と ITU4_BRA を組み合わせてバッファ動作とするかを選択します。

BFA4	説明
0	ITU4_GRA は通常動作
1	ITU4_BRA はバッファ動作

ITU4 は使いません。通常動作です。

・ビット 1:バッファ動作 B3(BFB3)

ITU3_GRB を通常動作とするか、ITU3_GRB と ITU3_BRB を組み合わせてバッファ動作とするかを選択します。

BFB3	説明
0	ITU3_GRB は通常動作
1	ITU3_BRB はバッファ動作

ITU3_GRB の変わりにバッファである ITU3_BRB へ値を書き込みます。使用するので"1"です。

・ビット 0:バッファ動作 A3(BFA3)

ITU3_GRA を通常動作とするか、ITU3_GRA と ITU3_BRA を組み合わせてバッファ動作とするかを選択します。

BFA3	説明
0	ITU3_GRA は通常動作
1	ITU3_BRA はバッファ動作

ITU3_GRA は周期を設定します。一度設定するともう変えることがないので、バッファにする必要はありません。

```
53 :       ITU_FCR  = 0x02;                    /* バッファの設定            */
```

10.5.6 main関数

```
25 :  void main( void )
26 :  {
27 :      init();                              /* 初期化                    */
28 :
29 :      while( 1 ) {
30 :          ITU3_BRB = 49150 * dipsw_get() / 15;
31 :      }
32 :  }
```

29 行目の while 文のカッコ内は1で常に真なので 30 行目は繰り返されます。

30 行目で、ディップスイッチの値により ITU3_BRB へ代入する数値を決めてデューティ比を可変しています。ディップスイッチの値とデューティ比をまとめておきます。

ディップスイッチの値	ITU3_BRB の値	デューティ比
0	49150 * 0 / 15 = 0	0% ※1
1	49150 * 1 / 15 = 3276	6.6%
2	49150 * 2 / 15 = 6553	13.3%
3	49150 * 3 / 15 = 9830	20.0%
4	49150 * 4 / 15 = 13106	26.7%
5	49150 * 5 / 15 = 16383	33.3%
6	49150 * 6 / 15 = 19660	40.0%
7	49150 * 7 / 15 = 22936	46.7%
8	49150 * 8 / 15 = 26213	53.3%
9	49150 * 9 / 15 = 29490	60.0%
10	49150 * 10 / 15 = 32766	66.7%
11	49150 * 11 / 15 = 36043	73.3%
12	49150 * 12 / 15 = 39320	80.0%
13	49150 * 13 / 15 = 42596	86.7%
14	49150 * 14 / 15 = 45873	93.3%
15	49150 * 15 / 15 = 49150	100% ※2

※1…ITU3_BRB を0にしても、ITU3_CNT＝(ITU3_GRB+1)＝(0＋1)＝1で一致と見なされます。したがって、完全な 0%ではなく、ITU3_CNT の 1 カウント分（325.52ns）だけ ON になります。

※2…ITU3_BRB を 49151 とすると、ポイント1の説明のとおり、波形の変化が無くなくなるため、1 小さい値を設定しています。したがって、完全な 100%ではなく、ITU3_CNT の 1 カウント分（325.52ns）だけ OFF に

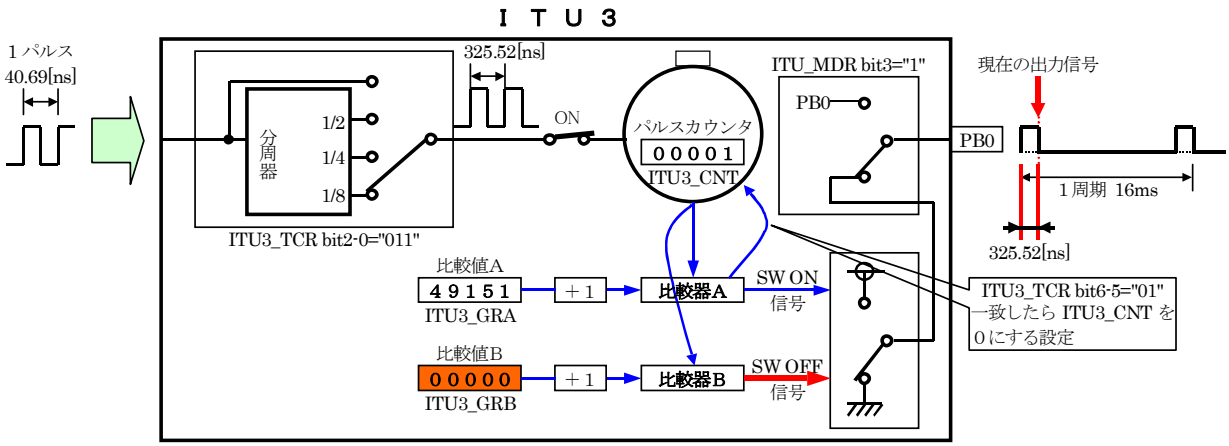
なります。

(1) 0%出力

0%出力、要は ON 幅 0[ms]にするとき、ITU3_GRB は、

$0 \div (325.52 \times 10^{-9}) = 0$

となります。しかし、比較は、「ITU3_CNT＝(ITU3_GRB＋1)」で行われるので、「ITU3_CNT＝(0＋1)」→「ITU3_CNT＝1」が最低の設定となります。必ず、1カウント分は"1"になってしまうのです(下図)。



今回、周期は 16[ms]です。それに対して ON 幅 325.52[ns]ですので、デューティ比は、

$(325.52 \times 10^{-9}) / (16 \times 10^{-3}) = 0.00002 = 0.002\%$

となり、モータ制御なら 0%としても良い範囲です。

(2) 100%出力

ポイント1のとおり、1カウント分 OFF になります。

参考資料ー30 行目の計算について

```
30 :          ITU3_BRB = 49150 * dipsw_get() / 15;
```

30 行目は、どのように計算されるのでしょうか。C 言語では、前記したように演算には優先順位があります。この式では左結合性より、左から順番に計算されます。まずは、

```
49150 * dipsw_get()
```

が、計算されます。

ここで型に注目します。

・49150 は、**long 型**

・dipsw_get()の戻り値は、unsigned char 型

したがって、

```
49150 * dipsw_get() → (long) * (unsigned char) → (long) * (long)
```

の型に変換されます。

dipsw_get()の範囲を思い出してみると、最大が 15 です。そのため、

```
49150 * 15 → 答え 737250
```

となります。こちらは問題有りません。

では、例えば 49150 が 10000 ならどうなるでしょうか。

```
ITU3_BRB = 10000 * dipsw_get() / 15;      /* 例えば 10000 なら…これは正しくない*/
```

まずは、
10000 * dipsw_get()
が、計算されます。
ここで型に注目します。
・10000 は、**int 型 ←先ほどとは違うことに注意！**
・dipsw_get()の戻り値は、unsigned char 型
したがって、
10000 * dipsw_get() → (**int**) * (unsigned char) → (int) * (int)
の型に変換されます。
dipsw_get()の範囲を思い出してみると、最大が 15 です。そのため、
10000 * 15 → 答え ~~150000~~ **不定!!**
と正しい答えになりません。実は**答えも計算したときと同じ型の int 型なのです**。int 型の上限は 32767 です。これを越える答えは不定となります。この後に 15 で除算しますので更に違った値となってしまいます。
このように、定数の型に気をつける必要があります。

・10進数定数が int 型で表せるなら int 型になる
・10進数定数が int 型で表しきれないときは、long 型になる
・long 型でも表しきれなければ、unsigned long 型になる

定数の値によって、型が違います。答えが計算式の型を超えてしまう場合、強制的に大きな型に変換してオーバフローしないようにプログラマが考慮する必要があります。ここで、キャスト演算子という演算子を使います。数値の前に(long)とつけると、10000 という数値は強制的に long 型に変換され、dipsw_get()も long 型に変換されます。
(long)10000 * 15 → 答え 150000
今度こそ正しい値になります。

```
ITU3_BRB = (long)10000 * dipsw_get() / 15;      /* OK! */
```

ちなみに、16進数の定数の型は、10進数とは違います。一緒に説明しておきます。

・16進数定数が int 型で表せるなら int 型になる
・int 型で表しきれないときは、unsigned int 型になる
・unsigned int 型で表しきれないときは、long 型になる
・long 型でも表しきれなければ、unsigned long 型になる

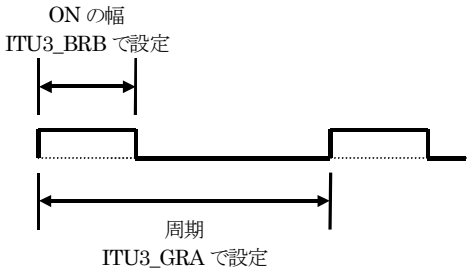
10.5.7 ITU4 を使用する場合

下記の箇所が変更になります。

```
30 :          ITU4_BRB = 49150 * dipsw_get() / 15;
中略
52 :      ITU4_TCR = 0x23;                  /* カウンタ、クリアの設定   */
53 :      ITU_FCR  = 0x08;                  /* バッファの設定           */
54 :      ITU4_GRA = 49151;                 /* PWM 周期                 */
55 :      ITU4_GRB = ITU4_BRB = 0;          /* デューティ比設定         */
56 :      ITU_MDR  = 0x10;                  /* PWM モード設定           */
57 :      ITU_STR  = 0x10;                  /* タイマスタート           */
```

ITU3 を ITU4 へ変更するのはもちろんですが、53 行の ITU_FCR レジスタの設定を ITU4_BRB のバッファを使用する設定に変更するところがポイントです。

10.6 まとめ



設定するレジスタ	詳細
ITU3_TCR	ITU3_CNT の値が＋1する時間、クリア要因の設定をします。 0x20…周期が 2.666ms 以下の場合(＋1する時間は 40.69[ns]ごと) 0x21…周期が 5.333ms 以下の場合(＋1する時間は 81.38[ns]ごと) 0x22…周期が 10.67ms 以下の場合(＋1する時間は 162.76[ns]ごと) 0x23…周期が 21.33ms 以下の場合(＋1する時間は 325.52[ns]ごと) 今回は、周期 16ms なので **0x23** を設定します。
ITU_FCR	バッファレジスタを使用するかの設定します。 ITU3 を使用する場合、**0x02** を設定します。ITU4 を使用する場合、**0x08** を設定します。
ITU3_GRA	周期を設定します。計算は、「設定したい周期÷(ITU3_CNT の値が＋1する時間)－1」です。周期 16ms、ITU3_TCR が 0x23 なら＋1する時間は 325.52ns となり $(16\times10^{-3})\div(325.52\times10^{-9})-1=49151$
ITU3_BRB (ITU3_GRB)	ON 幅を設定します。 計算は、「設定したい ON 幅÷(ITU3_CNT の値が＋1する時間)－1」です。ON 幅 8ms、＋1する時間 325.52ns なら $(8\times10^{-3})\div(325.52\times10^{-9})-1=24575$ ITU3_GRB は、最初だけ ITU3_BRB と同じ値を設定します。2 回目以降は、ITU3_BRB のみ設定します。最大値は、ITU3_GRA－1の値です。1 カウント分 OFF になります。
ITU_MDR	それぞれの ITU のチャネルで PWM 波形を出力するかしないか選択します。 ″1″:PWM波形出力する ″0″:しない bit 7 6 5 4 3 2 1 0 0固定 0固定 0固定 ITU4 ITU3 ITU2 ITU1 ITU0 今回は ITU3 を使用するので、ITU3 の部分が″1″、他は″0″なので **0x08** を設定します。 ITU4 なら **0x10** を設定します。
ITU_STR	それぞれの ITU のチャネルで ITU_CNT をカウント動作させるかどうか選択します。要は、ITU を使うか使わないかの設定です。 ″1″:使用する ″0″:使用しない bit 7 6 5 4 3 2 1 0 0固定 0固定 0固定 ITU4 ITU3 ITU2 ITU1 ITU0 今回は ITU3 を使用するので、ITU3 の部分が″1″、他は″0″なので **0x08** を設定します。 ITU4 なら **0x10** を設定します。

ITU_MDR の該当ビットを″1″にすると、**DDR の設定が入力だろうが出力だろうが PWM 出力端子となります**。一応、DDR の設定も出力設定にしておきましょう。

11. プロジェクト「pwm2」 ITU0(または 1、2)を使ったPWM信号出力

11.1 概要

ITU0 を使って PWM 波形を端子に出力します。周期は、16[ms]に設定します。デューティ比は、CPU ボード上のディップスイッチにより 16 段階に切り替えることができます。

本プログラムを改造して、ITU1 または ITU2 と置き換えることもできます。ITU3～4 を使用して PWM 出力する場合は、pwm1.c を参照ください。

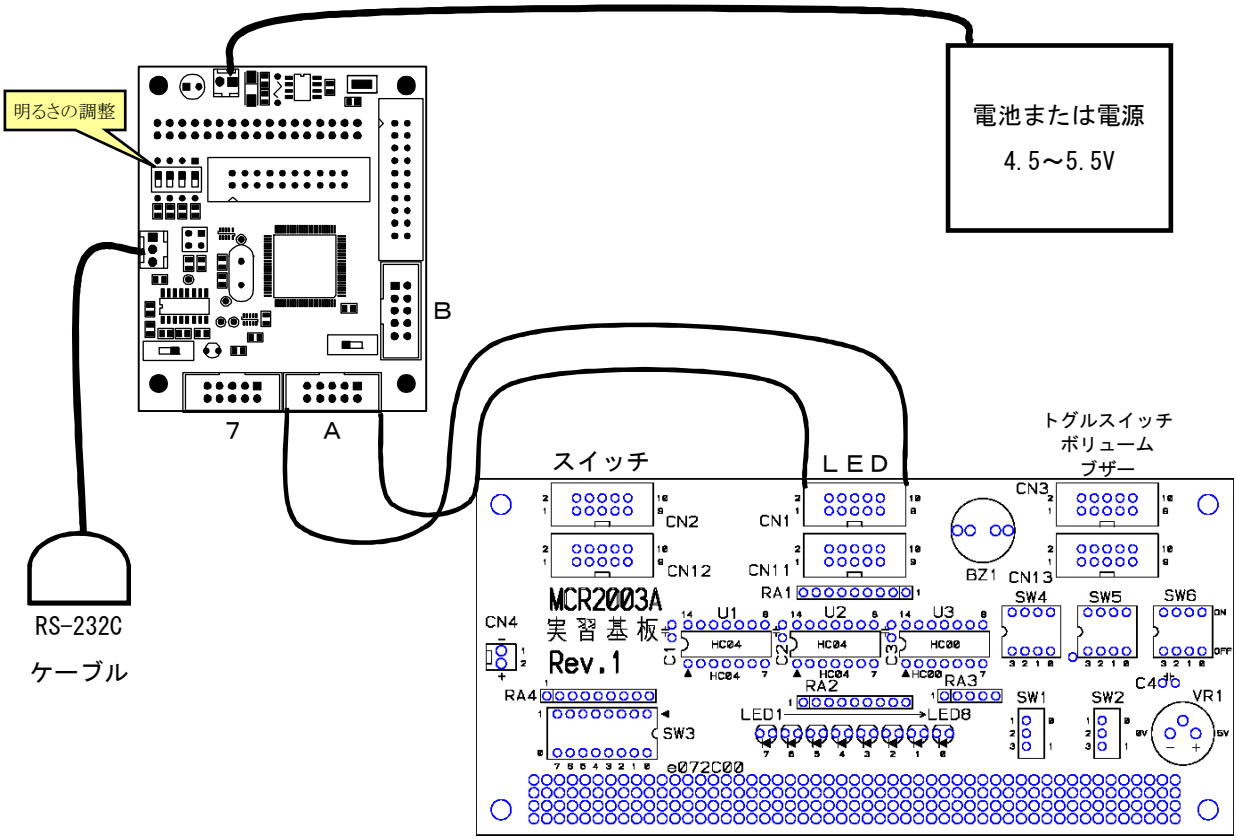
マイコンのポートは下記を使用します。

・ポート A の 2 ビット・・・LED へデータ出力(PWM 出力)

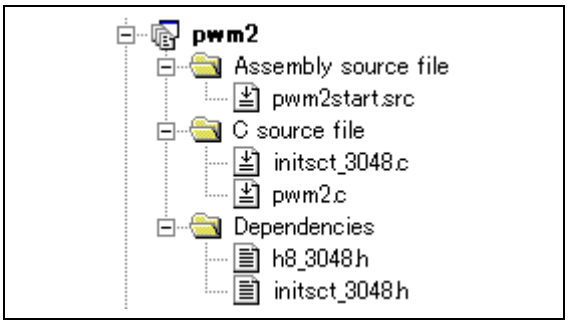
11.2 接続

・CPU ボードのポート A と実習基板の LED 部を、フラットケーブルで接続します。

※LED の明るさの調整は、CPU ボードのディップスイッチで行います。



11.3 プロジェクトの構成



	ファイル名	内容
1	pwm2start.src	アセンブリ言語で記述されたアセンブリソースファイルです。このファイルの構造は下記のようになっています。 ベクタアドレス ＋ スタートアップルーチン
2	initsct_3048.c	初期値のないグローバル変数(セクション B 領域)、初期値のあるグローバル変数(セクション R 領域)の初期化用です。
3	pwm2.c	実際に制御するプログラムが書かれています。H8/3048F-ONE の内蔵周辺機能の初期化も行います。
4	h8_3048.h	H8/3048F-ONE の内蔵周辺機能の I/O レジスタを定義したファイルです。
5	initsct_3048.h	initsct_3048.c のヘッダファイルです。

11.4 プログラム「pwm2.c」

```
 1 : /************************************************************************/
 2 : /* PWM波形出力(ITU0またはITU1またはITU2使用)「pwm2.c」                   */
 3 : /* 入力：CPUボードのディップスイッチ(P63-P60)    出力：PA2(LED等)        */
 4 : /*                          2005.04 ジャパンマイコンカーラリー実行委員会  */
 5 : /************************************************************************/
 6 : /*======================================*/
 7 : /* インクルード                         */
 8 : /*======================================*/
 9 : #include    <machine.h>
10 : #include    "h8_3048.h"
11 :
12 : /*======================================*/
13 : /* シンボル定義                         */
14 : /*======================================*/
15 :
16 : /*======================================*/
17 : /* プロトタイプ宣言                     */
18 : /*======================================*/
19 : void init( void );
20 : unsigned char dipsw_get( void );
21 :
22 : /************************************************************************/
23 : /* メインプログラム                                                     */
24 : /************************************************************************/
25 : void main( void )
26 : {
27 :     init();                             /* 初期化                       */
28 :
29 :     while( 1 ) {
30 :         /* GRAとのコンペアマッチが起きるまで待つ */
31 :         while( ITU0_CNT >= ITU0_GRA-10 );
32 :         /* GRBとのコンペアマッチが起きるまで待つ */
33 :         while((ITU0_CNT >= ITU0_GRB-10) && (ITU0_CNT <= ITU0_GRB));
34 :         /* デューティの設定 */
35 :         ITU0_GRB = 49150 * dipsw_get() / 15;
36 :     }
37 : }
38 :
```

```
/************************************************************************/
/* H8/3048F-ONE 内蔵周辺機能の初期化                                    */
/************************************************************************/
void init( void )
{
    /* ポートの入出力設定 */
    P1DDR = 0xff;
    P2DDR = 0xff;
    P3DDR = 0xff;
    P4DDR = 0xff;
    P5DDR = 0xff;
    P6DDR = 0xf0;                       /* CPU基板上のDIP SW        */
    P8DDR = 0xff;
    P9DDR = 0xf7;
    PADDR = 0xff;
    PBDDR = 0xff;                       /* LED基板                  */
    /* ポート７は、入力専用なので入出力設定はありません    */

    ITU0_TCR = 0x23;                    /* カウンタ、クリアの設定   */
    ITU0_GRA = 49151;                   /* PWM周期                  */
    ITU0_GRB = 0;                       /* デューティ比設定         */
    ITU_MDR  = 0x01;                    /* PWMモード設定            */
    ITU_STR  = 0x01;                    /* タイマスタート           */
}

/************************************************************************/
/* ディップスイッチ値読み込み                                           */
/* 戻り値 スイッチ値 0～15                                              */
/************************************************************************/
unsigned char dipsw_get( void )
{
    unsigned char sw;

    sw  = ~P6DR;                        /* ディップスイッチ読み込み */
    sw &= 0x0f;

    return  sw;
}

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

11.5 プログラムの解説

11.5.1 PWM出力端子

ITU を PWM モードに設定したとき、その波形が出力される端子は決まっています。出力端子の変更はできません。

ITU のチャネル	PWM 波形出力端子
0	ポート A の bit2 端子
1	ポート A の bit4 端子
2	ポート A の bit6 端子
3	ポート B の bit0 端子
4	ポート B の bit2 端子

今回は ITU0 を使用しますので、PA2 から PWM 信号が出力されます。

11.5.2 ITU0 の初期設定

```
57 :      ITU0_TCR = 0x23;                  /* カウンタ、クリアの設定   */
58 :      ITU0_GRA = 49151;                 /* PWM 周期                 */
59 :      ITU0_GRB = 0;                     /* デューティ比設定         */
60 :      ITU_MDR  = 0x01;                  /* PWM モード設定           */
61 :      ITU_STR  = 0x01;                  /* タイマスタート           */
```

ITU0 関連のレジスタを設定します。周期は、16[ms]とします。

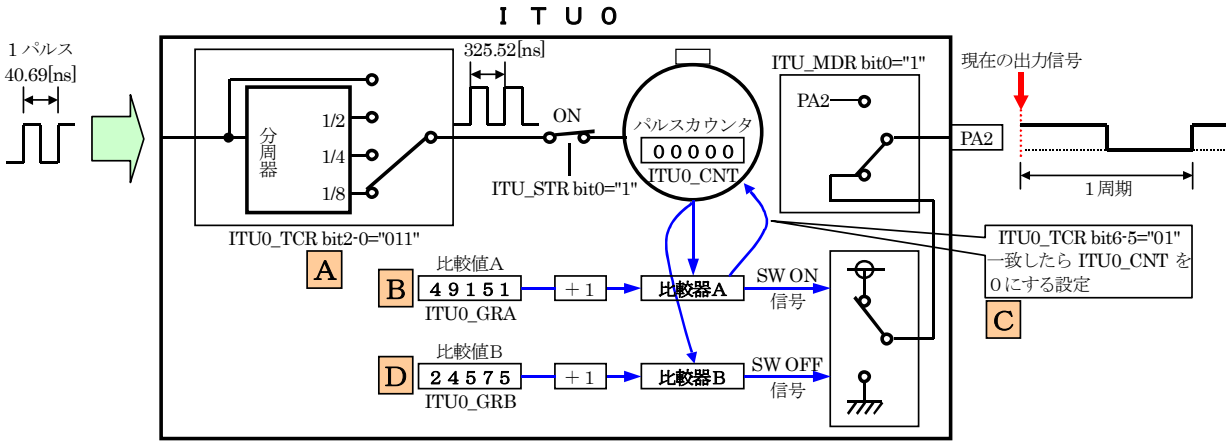
11.5.3 メインプログラム

```
25 :  void main( void )
26 :  {
27 :      init();                              /* 初期化                    */
28 :
29 :      while( 1 ) {
30 :          /* GRA とのコンペアマッチが起きるまで待つ */
31 :          while( ITU0_CNT >= ITU0_GRA-10 );
32 :          /* GRB とのコンペアマッチが起きるまで待つ */
33 :          while((ITU0_CNT >= ITU0_GRB-10) && (ITU0_CNT <= ITU0_GRB));
34 :          /* デューティの設定 */
35 :          ITU0_GRB = 49150 * dipsw_get() / 15;
36 :      }
37 :  }
```

(1) 基本的なプログラム

基本のプログラムは下記のようです。このプログラムは、ある特定の条件になった場合、正しく動作しないことがあります。30～33 行のプログラムは、不具合を防止するために入れています。

```
25 :  void main( void )
26 :  {
27 :      init();                              /* 初期化                    */
28 :
29 :      while( 1 ) {
34 :          /* デューティの設定 */
35 :          ITU0_GRB = 49150 * dipsw_get() / 15;
36 :      }
37 :  }
```



A ITU0_TCR bit2-0 の設定により、ITU0_CNT は 325.52[ns]ごとに＋1します。
B ITU0_GRA には周期を設定します。「ITU0_CNT=(ITU0_GRA+1)」になると PA2 端子が"1"になります。
例えば、周期を 16[ms]にしたい場合、
$(16\times10^{-3})\div(325.52\times10^{-9})-1=49151$
を設定します。
C ITU0_TCR bit6-5 の設定により「ITU0_CNT=(ITU0_GRA+1)」になると ITU0_CNT が 0 になります。
D ITU0_GRB には ON 幅を設定します。「ITU0_CNT=(ITU0_GRB+1)」になると PA2 端子が"0"になります。

例えば、ON 幅を 1[ms]にしたい場合、
$(1\times10^{-3})\div(325.52\times10^{-9})-1=3071$
を設定します。

メインプログラムでは、

```
38 :            ITU0_GRB = 49150 * dipsw_get() / 15;
```

として、ON 幅をディップスイッチの割合に応じて変えています。

(2) ITU0_GRBに値を代入するときの対処

ITU0_GRB に値を代入するときに、pwm1.c であった問題が発生します。もう一度、思い出してみます。

ポイント2 … ITUx_GRA や ITUx_GRB のレジスタに値を書き換え中に、タイマカウンタとレジスタが一致した場合、信号は変化しません

ITU0_GRB を直接書き換えると、PWM 波形が変化しないことがあります。そこでバッファレジスタを使えると良いのですが、実は **ITU0～2 にはバッファレジスタが無いのです**。

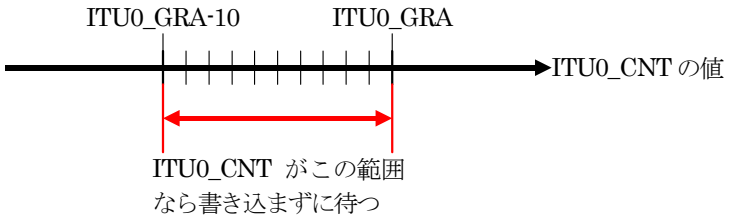
ITU のチャネル	バッファレジスタの有無
0	**なし**
1	**なし**
2	**なし**
3	あり
4	あり

このままでは波形が変化しない信号が出力されることがあります。そこで、プログラムの中で下記のようにします。

・ITU0_GRA と ITU0_CNT が一致しそうなときは、ITU0_GRA の書き込みを行わずに待つ
・ITU0_GRB と ITU0_CNT が一致しそうなときは、ITU0_GRB の書き込みを行わずに待つ

このプログラムを入れることによりポイント 2 は回避できます。

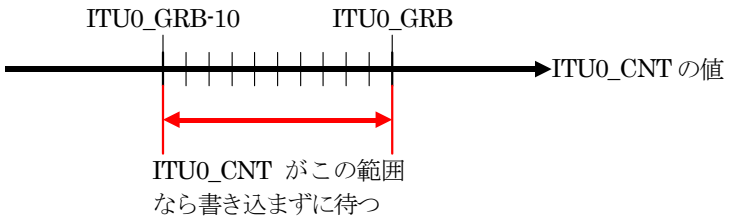
まず、ITU0_GRA の場合です。ITU0_GRA と ITU0_CNT の値が近い場合は、ITU0_CNT が ITU0_GRA の値を超えるまで待ちます。とりあえず 10 以内なら待つようにします。



図のように、ITU0_CNT が、(ITU0_GRA-10)～ITU0_GRA の間なら待つ、というプログラムにします。
ただし、ITU0_CNT=(ITU0_GRA+1)になると、ITU0_CNT はOになるので、ITU0_CNT が(ITU0_GRA-10)以上なら待つ、というプログラムでも同じです。これをプログラム化します。

```
30 :            /* GRA とのコンペアマッチが起きるまで待つ */
31 :            while( ITU0_CNT >= ITU0_GRA-10 );
```

次に、ITU0_GRB の場合です。ITU0_GRB と ITU0_CNT の値が近い場合は、ITU0_CNT が ITU0_GRB の値を超えるまで待ちます。とりあえず 10 以内なら待つようにします。



図のように、
ITU0_CNT が、(ITU0_GRB-10)～ITU0_GRB の間なら待つ、というプログラムにします。これをプログラム化します。

```
32 :            /* GRB とのコンペアマッチが起きるまで待つ */
33 :            while((ITU0_CNT >= ITU0_GRB-10) && (ITU0_CNT <= ITU0_GRB));
```

これで、ポイント2を回避できます。
ITU0、1、2 を PWM モードで使用する場合、この方法を使います。

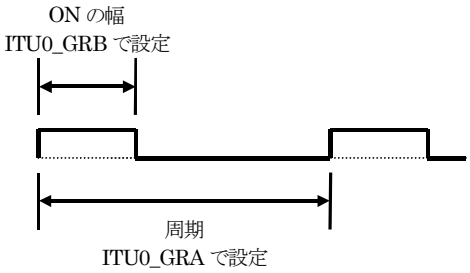
待った後、ITU0_GRB に値を書き込みます。

```
35 :            ITU0_GRB = 49150 * dipsw_get() / 15;
```

●ポイント

・ITU3、4 を PWM モードで使用する場合、バッファレジスタを使って競合を回避する。
・ITU0、1、2 を PWM モードで使用する場合、プログラムで競合を回避する。

11.6 まとめ



設定するレジスタ	詳細
ITU0_TCR	ITU0_CNT の値が一定間隔で＋1する時間、クリア要因の設定をします。 0x20…周期が 2.666ms 以下の場合(＋1する時間は 40.69[ns]ごと) 0x21…周期が 5.333ms 以下の場合(＋1する時間は 81.38[ns]ごと) 0x22…周期が 10.67ms 以下の場合(＋1する時間は 162.76[ns]ごと) 0x23…周期が 21.33ms 以下の場合(＋1する時間は 325.52[ns]ごと) 今回は、周期 16ms なので **0x23** を設定します。
ITU0_GRA	周期を設定します。計算は、「設定したい周期÷(ITU0_CNT の値が＋1する時間)－1」です。周期 16ms、ITU0_TCR が 0x23 なら＋1する時間は 325.52ns となり $(16\times10^{-3})\div(325.52\times10^{-9})-1=49151$
ITU0_GRB	ON 幅を設定します。 計算は、「設定したい ON 幅÷ITU0_CNT のカウント時間－1」です。ON 幅 8ms、＋1する時間 325.52ns なら $(8\times10^{-3})\div(325.52\times10^{-9})-1=24575$ **ITU0_GRB へ値を設定するときは、競合に気をつけます。**
ITU_MDR	それぞれの ITU のチャネルで PWM 波形を出力するかしないか選択します。 ″1″:PWM波形出力する ″0″:しない bit 7 6 5 4 3 2 1 0 0固定 0固定 0固定 ITU4 ITU3 ITU2 ITU1 ITU0 今回は ITU0 を使用するので、ITU0 の部分が″1″、他は″0″なので **0x01** を設定します。 ITU1 なら 0x02、ITU2 なら 0x04 を設定します。
ITU_STR	それぞれの ITU のチャネルで ITU_CNT をカウント動作させるかどうか選択します。要は、ITU を使うか使わないかの設定です。 ″1″:使用する ″0″:使用しない bit 7 6 5 4 3 2 1 0 0固定 0固定 0固定 ITU4 ITU3 ITU2 ITU1 ITU0 今回は ITU0 を使用するので、ITU0 の部分が″1″、他は″0″なので **0x01** を設定します。 ITU1 なら 0x02、ITU2 なら 0x04 を設定します。

ITU_MDR の該当ビットを″1″にすると、**DDR の設定が入力だろうが出力だろうが PWM 出力端子となります**。一応、DDR の設定も出力設定にしておきましょう。

12. プロジェクト「pwm21」 ITU0(または 1、2)を使ったPWM信号出力その2

12.1 概要

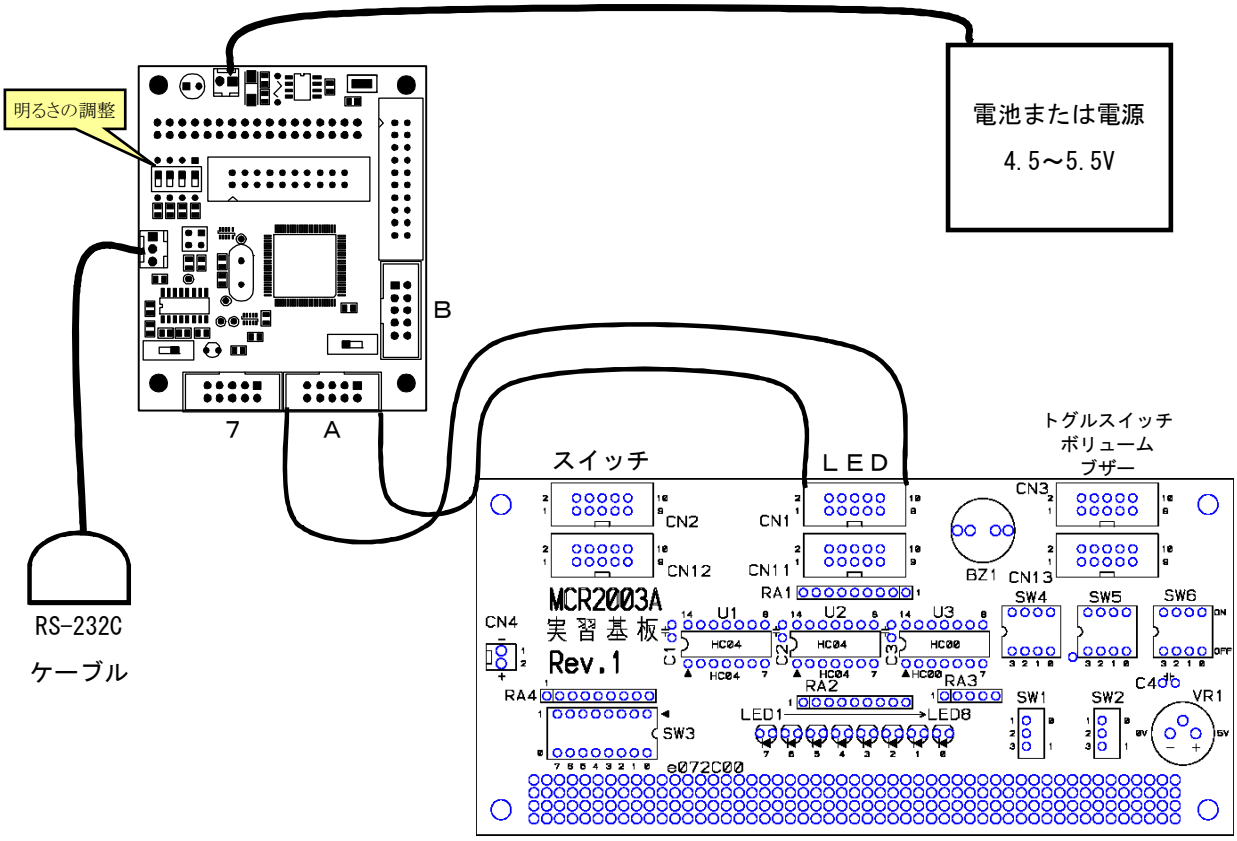
プロジェクト「pwm2」は、プログラムを簡単にするため、0%出力、100%出力はできません。「pwm21.c」は、0%出力、100%出力ができるように対策します。マイコンのポートは、下記を使用します。

・ポート A の 2 ビット・・・LED へデータ出力(PWM 出力)

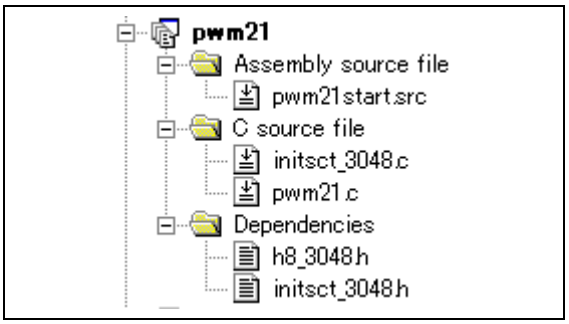
12.2 接続

・CPU ボードのポート A と実習基板の LED 部を、フラットケーブルで接続します。

※LED の明るさの調整は、CPU ボードのディップスイッチで行います。



12.3 プロジェクトの構成



	ファイル名	内容
1	pwm21start.src	アセンブリ言語で記述されたアセンブリソースファイルです。このファイルの構造は下記のようになっています。 ベクタアドレス ＋ スタートアップルーチン
2	initsct_3048.c	初期値のないグローバル変数(セクション B 領域)、初期値のあるグローバル変数(セクション R 領域)の初期化用です。
3	pwm21.c	実際に制御するプログラムが書かれています。H8/3048F-ONE の内蔵周辺機能の初期化も行います。
4	h8_3048.h	H8/3048F-ONE の内蔵周辺機能の I/O レジスタを定義したファイルです。
5	initsct_3048.h	initsct_3048.c のヘッダファイルです。

12.4 プログラム「pwm21.c」

```
/************************************************************************/
/* 実習用サンプルプログラム「pwm21.c」                                  */
/*                                  2006.04 マイコンカーラリー実行委員会 */
/************************************************************************/
/*
●プログラム解説
CPUボードのディップスイッチに応じて、PA2端子から出力するPWMの
デューティ比を変更します。「pwm2.c」に比べ、より完全なPWMとなっています。
●接続
入力：CPUボードのディップスイッチ(P63-P60)
出力：PA2端子(LED等を接続)
*/
/*======================================*/
/* インクルード                         */
/*======================================*/
#include    <machine.h>
#include    "h8_3048.h"

/*======================================*/
/* シンボル定義                         */
/*======================================*/
#define         PWM_CYCLE       49151   /* PWM のサイクル  16ms     */

/*======================================*/
/* プロトタイプ宣言                     */
/*======================================*/
void init( void );
unsigned char dipsw_get( void );
void pwm0( int value );

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

    while( 1 ) {
```

```
 39 :          pwm0( dipsw_get() * 10 );       /* ディップスイッチ10以上は */
 40 :                                          /* 常に100%になります       */
 41 :      }
 42 :  }
 43 :
 44 :  /************************************************************************/
 45 :  /* H8/3048F-ONE 内蔵周辺機能の初期化                                    */
 46 :  /************************************************************************/
 47 :  void init( void )
 48 :  {
 49 :      /* ポートの入出力設定 */
 50 :      P1DDR = 0xff;
 51 :      P2DDR = 0xff;
 52 :      P3DDR = 0xff;
 53 :      P4DDR = 0xff;
 54 :      P5DDR = 0xff;
 55 :      P6DDR = 0xf0;                       /* CPU基板上のDIP SW        */
 56 :      P8DDR = 0xff;
 57 :      P9DDR = 0xf7;
 58 :      PADDR = 0xff;
 59 :      PBDDR = 0xff;                       /* LED基板                  */
 60 :      /* ポート７は、入力専用なので入出力設定はありません    */
 61 :
 62 :      ITU0_TCR = 0x23;                    /* カウンタ、クリアの設定   */
 63 :      ITU0_GRA = PWM_CYCLE;               /* PWM周期                  */
 64 :      ITU0_GRB = 0;                       /* デューティ比設定         */
 65 :      ITU_MDR  = 0x01;                    /* PWMモード設定            */
 66 :      ITU_STR  = 0x01;                    /* タイマスタート           */
 67 :  }
 68 :
 69 :  /************************************************************************/
 70 :  /* ディップスイッチ値読み込み                                           */
 71 :  /* 戻り値 スイッチ値 0～15                                              */
 72 :  /************************************************************************/
 73 :  unsigned char dipsw_get( void )
 74 :  {
 75 :      unsigned char sw;
 76 :
 77 :      sw  = ~P6DR;                        /* ディップスイッチ読み込み */
 78 :      sw &= 0x0f;
 79 :
 80 :      return  sw;
 81 :  }
 82 :
 83 :  /************************************************************************/
 84 :  /* ＰＷＭ値出力                                                         */
 85 :  /* 引　数 ０～１００％                                                  */
 86 :  /************************************************************************/
 87 :  void pwm0( int value )
 88 :  {
 89 :      /* 0%,100%及びコンペアマッチ接近のcheck */
 90 :      if( value == 0 ) {
 91 :          /* 0%なら */
 92 :          ITU0_TCR = 0x43;
 93 :          ITU0_GRB = PWM_CYCLE -1;
 94 :          ITU0_CNT = PWM_CYCLE -2;
 95 :      } else if( value < 100 ) {
 96 :          /* 1～99%なら */
 97 :          /* GRAとのコンペアマッチが起きるまで待つ */
 98 :          while( ITU0_CNT >= ITU0_GRA-10 );
 99 :          /* GRBとのコンペアマッチが起きるまで待つ */
100 :          while((ITU0_CNT >= ITU0_GRB-10) && (ITU0_CNT <= ITU0_GRB));
101 :          ITU0_TCR = 0x23;
102 :          ITU0_GRB = (unsigned long)PWM_CYCLE * value / 100;
103 :      } else {
104 :          /* 100%なら */
105 :          ITU0_TCR = 0x23;
106 :          ITU0_GRB = PWM_CYCLE + 1;
107 :          ITU0_CNT = PWM_CYCLE - 1;
108 :      }
109 :  }
110 :
111 :  /************************************************************************/
112 :  /* end of file                                                          */
113 :  /************************************************************************/
```

12.5 プログラムの解説

12.5.1 「pwm2.c」のPWM0%は本当に0%か

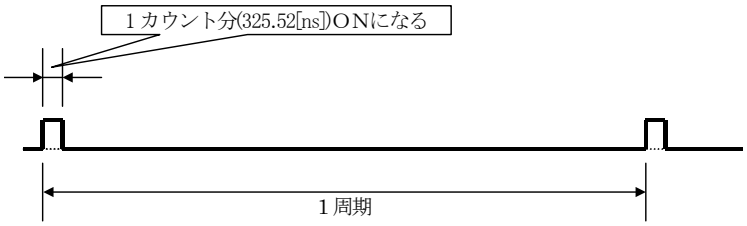
ITU0_CNT と(ITU0_GRB+1)が一致したとき、PA2 端子が"0"になります。したがって、ITU0_GRB に 0 を入れれば、0%出力になるはずです。
しかし、"0"になるきっかけは「**ITU0_GRB+1**」です。ITU0_GRB に 0 を代入すると「0＋1＝1」で最小値が1になります。そのため、

・ITU0_CNT=(ITU0_GRA+1)で端子出力が"1"になり、ITU0_CNT が 0 になる
・ITU0_CNT=(ITU0_GRB+1)＝(0+1)＝1 で、端子出力が"0"になる

という動作になります。必ず1つ分は"1"になるのです。1 カウントは先の計算で 325.52[ns]と分かっていますので、結論としては、

ITU0_GRB にOを入れても 325.52[ns]は端子出力が"0"になる

ということになります。



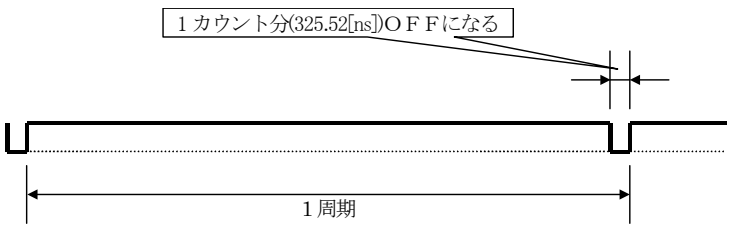
12.5.2 「pwm2.c」のPWM100%は本当に100%か

ITU0_CNT と(ITU0_GRB+1)が一致したとき、PA2 端子が"0"になります。したがって、端子を"0"にする ITU0_GRB の値を ITU0_GRA と同じ値にすれば、100%出力になるはずです。しかし、ポイント2の問題が起こってしまい 100%出力にできません。ITU0_GRB には、ITU0_GRA より1小さい値を代入します。そのため、

・ITU0_CNT=(ITU0_GRB+1)＝49151 で、端子出力が"0"になる
・ITU0_CNT=(ITU0_GRA+1)＝49152 で、端子出力が"1"になり、ITU0_CNT がOになる

という動作になります。必ず1つ分は"0"になるのです。1 カウントは先の計算で 325.52[ns]と分かっていますので、結論としては、

ITU0_GRB に誤動作しない最大限の値である「ITU0_GRA-1」を入れても 325.52[ns]間は端子出力が"0"になる
ということになります。

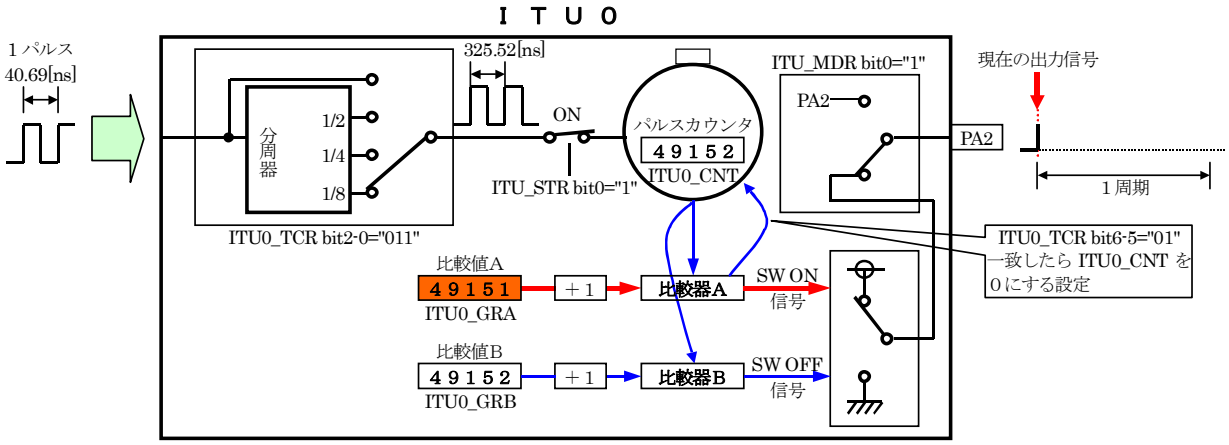


12.5.3 正真正銘の100%PWM出力にする！

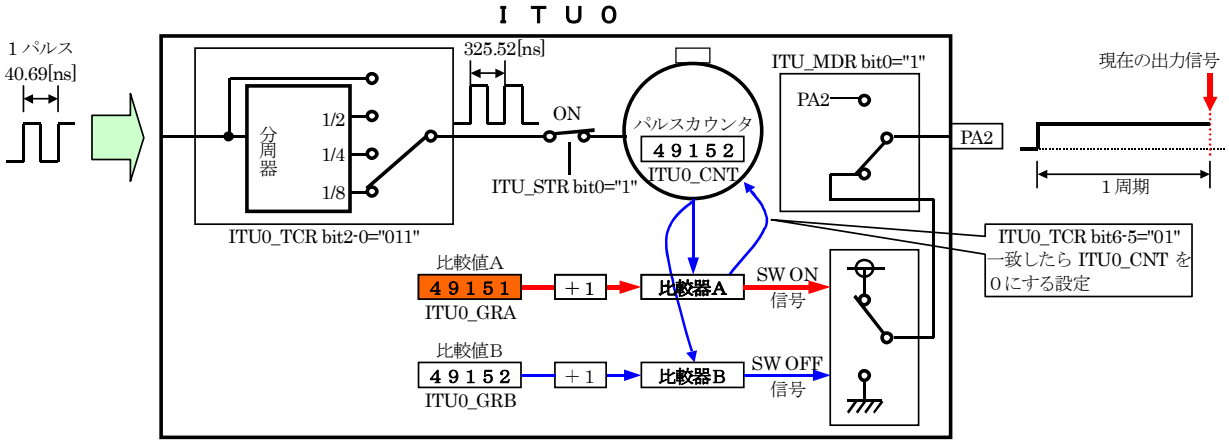
先のプログラムでは、100%にしたくとも ITU0_CNT の1カウント分は"0"になってしまいます。どうにかして正真正銘 100%にできないのでしょうか。ITU0_CNT、ITU0_GRA、ITU0_GRB の関係をもう一度まとめると、

・ITU0_CNT＝(ITU0_GRB+1)になると出力端子が"0"になる
・ITU0_CNT＝(ITU0_GRA+1)になると出力端子が"1"になり、ITU0_CNT が 0 になる

ということは、「ITU0_CNT=ITU0_GRB+1」の条件が起こらなければ"0"になることはありません。ではどうすればよいのでしょうか。「ITU0_CNT=ITU0_GRA+1」で ITU0_CNT の値が 0 になります。したがって、ITU0_GRB の値を ITU0_GRA より大きい値にすれば「ITU0_CNT＝(ITU0_GRB+1)」にはなりません。
例として、ITU0_GRA に 49151、ITU0_GRB に 49152 を代入したときの動作を図解してみます。



「ITU0_CNT＝(ITU0_GRA+1)」になったので出力端子が"1"になり、ITU0_CNT が 0 になります。そして、ITU0_CNT が 1、2、3…と増えていきます。



ITU0_CNT が増えていき、また 49152 になりました。「ITU0_CNT＝(ITU0_GRA+1) → 49152＝(49151+1)」となったので、波形が"1"になり、ITU0_CNT がクリアされます。この間、「ITU0_CNT＝(ITU0_GRB+1)」になることはないので、"0"になりません(上図)。正真正銘の 100%です！

12.5.4 正真正銘の0%PWM出力にする！

先のプログラムでは、0%にしたくとも ITU0_CNT の1カウント分"1"になってしまいます。どうにかして正真正銘の0%にできないのでしょうか。

100%出力のときは、

・ITU0_GRB の値を ITU0_GRA より大きくする
・ITU0_CNT＝(ITU0_GRA+1)になると出力端子が"1"になり、**ITU0_CNT が 0 になる**
・ITU0_GRB は ITU0_GRA より大きいので、ITU0_CNT＝(ITU0_GRB+1)になる前に ITU0_CNT が 0 になる
・結果、出力端子は常に"1"となる

となり、100%になりました。今度は、「ITU0_CNT＝(ITU0_GRA+1)」にならないようにすれば、出力端子が"1"になることがありません。下記のような動作にすれば 0%になりそうです。

・ITU0_GRA の値を ITU0_GRB より大きくする
・ITU0_CNT＝(ITU0_GRB+1)になると出力端子が"0"になり、**ITU0_CNT が 0 になるようにする！**
・ITU0_GRA は ITU0_GRB より大きいので、ITU0_CNT＝(ITU0_GRA+1)になる前に ITU0_CNT が 0 になる
・結果、出力端子は常に"0"となる

ITU0_CNT の動作は、ITU0_TCR を設定することにより決定します。もう一度設定を見てみます。

●ITU0_TCR(タイマコントロールレジスタ)の設定内容

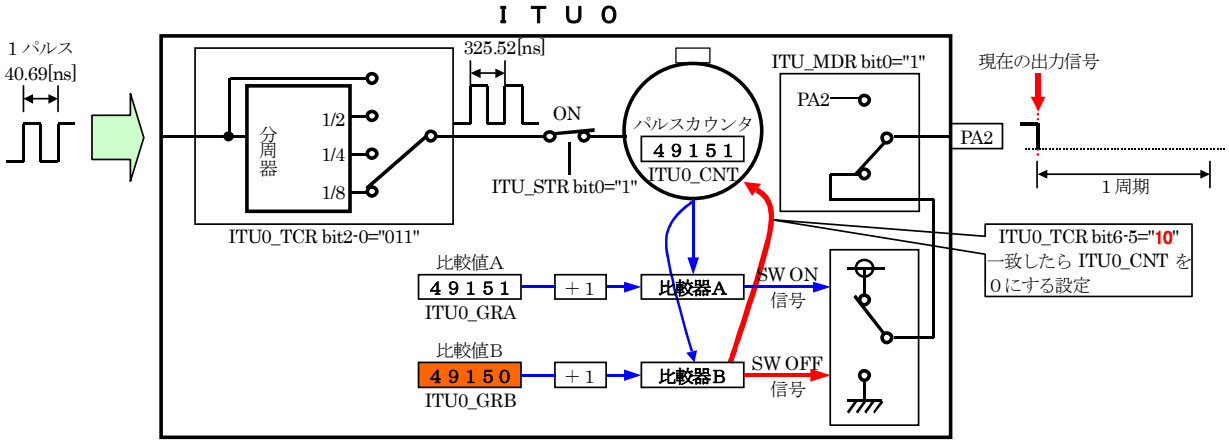
ビット:	7	6	5	4	3	2	1	0
ITU0_TCR:	－	CCLR1	CCLR0	CKEG1	CKEG0	TPSC2	TPSC1	TPSC0
設定値:	0	1	0	0	0	0	0	0
16 進数:	4				0			

・ビット 6,5：カウンタクリア 1,0
CNT のカウンタクリア要因を選択します。

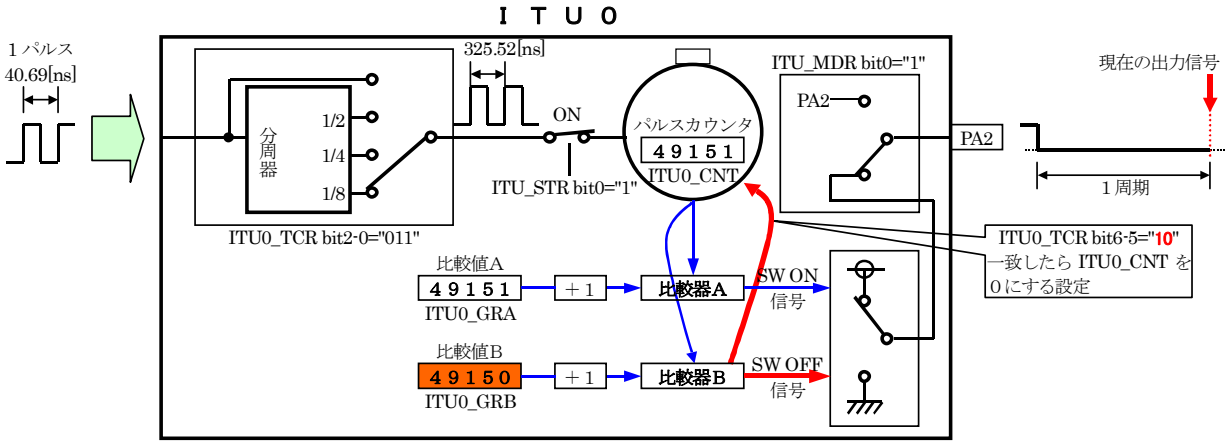
CCLR1	CCLR0	説明
0	0	CNT のクリア禁止
0	1	GRA のコンペアマッチ／インプットキャプチャで CNT をクリア
1	0	GRB のコンペアマッチ／インプットキャプチャで CNT をクリア
1	1	同期クリア

この設定が、「ITU0_CNT をOにするタイミング」です。コンペアマッチとは、コンペア＝比較、マッチ＝一致、なので、ITU0_CNT と（ITU0_GRB＋1）が一致すると ITU0_CNT をクリアにしなさい、という設定になります。

図解すると下図のようになります。



「ITU0_CNT＝(ITU0_GRB+1) → 49151＝(49150+1)」となったので、波形が"0"になり、ITU0_CNT がクリアされます（上図）。そして、ITU0_CNT が 1、2、3…と増えていきます。



ITU0_CNT が増えていき、また 49151 になりました。「ITU0_CNT＝(ITU0_GRB+1) → 49151＝(49150+1)」となったので、波形が"0"になり、ITU0_CNT がクリアされます。この間、「ITU0_CNT＝(ITU0_GRA+1)」になることはないので、"1"になりません（上図）。正真正銘の 0%です！

12.5.5 pwm0 関数

引数に 0～100 の値を設定すると、PWM値が 0～100%になる pwm0 関数を作りました。

```
 83 :  /*********************************************************************/
 84 :  /* ＰWM値出力                                                        */
 85 :  /* 引　数 ０～１００％                                               */
 86 :  /*********************************************************************/
 87 :  void pwm0( int value )
 88 :  {
 89 :      /* 0%,100%及びコンペアマッチ接近の check */
 90 :      if( value == 0 ) {
 91 :          /* 0%なら */
 92 :          ITU0_TCR = 0x43;
 93 :          ITU0_GRB = PWM_CYCLE -1;
 94 :          ITU0_CNT = PWM_CYCLE -2;
 95 :      } else if( value < 100 ) {
 96 :          /* 1～99%なら */
 97 :          /* GRA とのコンペアマッチが起きるまで待つ */
 98 :          while( ITU0_CNT >= ITU0_GRA-10 );
 99 :          /* GRB とのコンペアマッチが起きるまで待つ */
100 :          while((ITU0_CNT >= ITU0_GRB-10) && (ITU0_CNT <= ITU0_GRB));
101 :          ITU0_TCR = 0x23;
102 :          ITU0_GRB = (unsigned long)PWM_CYCLE * value / 100;
103 :      } else {
104 :          /* 100%なら */
105 :          ITU0_TCR = 0x23;
106 :          ITU0_GRB = PWM_CYCLE + 1;
107 :          ITU0_CNT = PWM_CYCLE - 1;
108 :      }
109 :  }
```

動作は、

・0%なら、91～94 行を実行　　…正真正銘の0%PWM出力にする！を参照
・1～99%なら、96～102 行を実行　　…pwm2.c と同じ
・100%以上なら、104～107 行を実行　　…正真正銘の100%PWM出力にする！を参照

として、引数により処理を分けました。

12.5.6 main関数

```
34 :  void main( void )
35 :  {
36 :      init();                              /* 初期化                       */
37 :
38 :      while( 1 ) {
39 :          pwm0( dipsw_get() * 10 );        /* ディップスイッチ 10 以上は */
40 :                                           /* 常に 100%になります          */
41 :      }
42 :  }
```

CPU ボード上のディップスイッチの値により、デューティ比は下表のようになります。

ディップ スイッチの値	pwm0 関数 の引数	ITU0_GRA (一致すると"1")	ITU0_GRB (一致すると"0")	ITU0_CNT の クリア要因	デューティ比
0	0	49,151	49,150	ITU0_GRB	0%
1	10	49,151	4,915	ITU0_GRA	10%
2	20	49,151	9,830	ITU0_GRA	20%
3	30	49,151	14,745	ITU0_GRA	30%
4	40	49,151	19,660	ITU0_GRA	40%
5	50	49,151	24,575	ITU0_GRA	50%
6	60	49,151	29,490	ITU0_GRA	60%
7	70	49,151	34,405	ITU0_GRA	70%
8	80	49,151	39,320	ITU0_GRA	80%
9	90	49,151	44,235	ITU0_GRA	90%
10	100	49,151	49,152	ITU0_GRA	100%
11	110	49,151	49,152	ITU0_GRA	100%
12	120	49,151	49,152	ITU0_GRA	100%
13	130	49,151	49,152	ITU0_GRA	100%
14	140	49,151	49,152	ITU0_GRA	100%
15	150	49,151	49,152	ITU0_GRA	100%

pwm0 関数は、100 以上を代入しても 100%となります。そのため、ディップスイッチの値が 10 以上の場合は 100%となります。それ以外のときは、ディップスイッチ値の 10 倍の PWM 値が出力されます。

12.6 まとめ

ITU0、1、2 を使って PWM 出力する場合、
・0%出力
・100%出力
・それ以外の出力
でプログラムを分ける必要があります。今回は、pwm0 という関数を作り、行っています。

13. プロジェクト「pwm3」 リセット同期PWMモードを使ったPWM信号出力

13.1 概要

リセット同期 PWM モードを使用して、PWM 波形を出力します。

ITU を PWM 設定にすると PWM 波形を出力できますが、1チャネル1出力です。リセット同期 PWM モードを使用すると ITU3 と ITU4 の 2 チャネルを組み合わせて 3 つの PWM 波形を出力でき、同時に逆相の波形も出力されます。デューティ比は、CPU ボード上のディップスイッチにより 16 段階に切り替えます。

周期は、16[ms]とします。

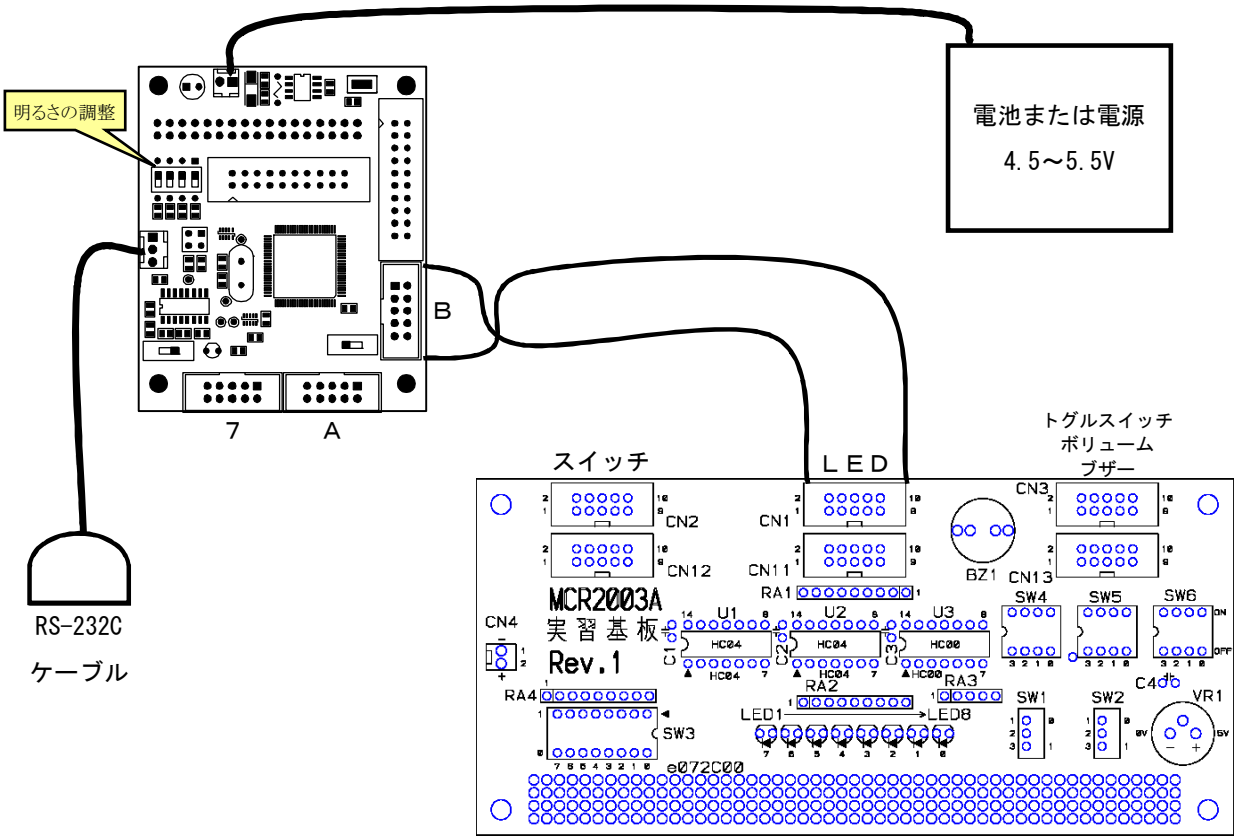
マイコンのポートは下記を使用します。

・ポート B の 0 ビット～5 ビット・・・LED へデータ出力(PWM 出力)

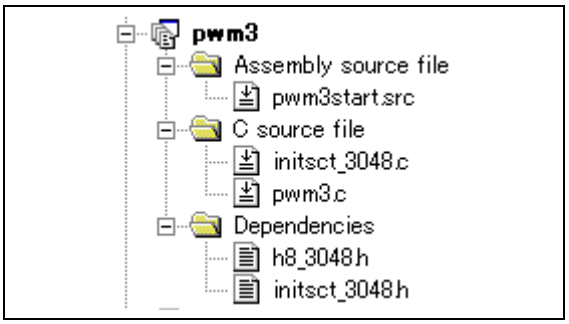
13.2 接続

・CPU ボードのポート B と実習基板の LED 部を、フラットケーブルで接続します。

※LED の明るさの調整は、CPU ボードのディップスイッチで行います。



13.3 プロジェクトの構成



	ファイル名	内容
1	pwm3start.src	アセンブリ言語で記述されたアセンブリソースファイルです。このファイルの構造は下記のようになっています。 ベクタアドレス ＋ スタートアップルーチン
2	initsct_3048.c	初期値のないグローバル変数(セクション B 領域)、初期値のあるグローバル変数(セクション R 領域)の初期化用です。
3	pwm3.c	実際に制御するプログラムが書かれています。H8/3048F-ONE の内蔵周辺機能の初期化も行います。
4	h8_3048.h	H8/3048F-ONE の内蔵周辺機能の I/O レジスタを定義したファイルです。
5	initsct_3048.h	initsct_3048.c のヘッダファイルです。

13.4 プログラム「pwm3.c」

```
 1 :  /*********************************************************************/
 2 :  /* PWM波形出力(リセット同期PWMモード使用)「pwm3.c」                  */
 3 :  /* 入力:CPUボードのディップスイッチ(P63-P60)   出力:PB0-PB5(LED等)   */
 4 :  /*                          2005.04 ジャパンマイコンカーラリー実行委員会  */
 5 :  /*********************************************************************/
 6 :  /*====================================*/
 7 :  /* インクルード                       */
 8 :  /*====================================*/
 9 :  #include    <machine.h>
10 :  #include    "h8_3048.h"
11 :
12 :  /*====================================*/
13 :  /* シンボル定義                       */
14 :  /*====================================*/
15 :
16 :  /*====================================*/
17 :  /* プロトタイプ宣言                   */
18 :  /*====================================*/
19 :  void init( void );
20 :  unsigned char dipsw_get( void );
21 :
22 :  /*********************************************************************/
23 :  /* メインプログラム                                                  */
24 :  /*********************************************************************/
25 :  void main( void )
26 :  {
27 :      init();                             /* 初期化                    */
28 :
29 :      while( 1 ) {
30 :          ITU3_BRB = 49150 * dipsw_get() / 15;
31 :      }
32 :  }
33 :
34 :  /*********************************************************************/
35 :  /* H8/3048F-ONE 内蔵周辺機能の初期化                                 */
36 :  /*********************************************************************/
37 :  void init( void )
38 :  {
```

```
39 :      /* ポートの入出力設定 */
40 :      P1DDR = 0xff;
41 :      P2DDR = 0xff;
42 :      P3DDR = 0xff;
43 :      P4DDR = 0xff;
44 :      P5DDR = 0xff;
45 :      P6DDR = 0xf0;                     /* CPU基板上のDIP SW        */
46 :      P8DDR = 0xff;
47 :      P9DDR = 0xf7;
48 :      PADDR = 0xff;
49 :      PBDDR = 0xff;                     /* LED基板                  */
50 :      /* ポート７は、入力専用なので入出力設定はありません    */
51 :
52 :      /* リセット同期式ＰＷＭモードの設定 */
53 :      ITU3_TCR = 0x23;                  /* カウンタ、クリアの設定   */
54 :      ITU_FCR  = 0x3e;                  /* リセット同期PWMモード    */
55 :      ITU3_GRA = 49151;                 /* 周期                     */
56 :      ITU3_GRB = ITU3_BRB = 0;          /* デューティ比設定 PB0,1   */
57 :      ITU4_GRA = ITU4_BRA = 0;          /* デューティ比設定 PB2,4   */
58 :      ITU4_GRB = ITU4_BRB = 0;          /* デューティ比設定 PB3,5   */
59 :      ITU_TOER = 0x3f;                  /* 出力端子の設定           */
60 :      ITU_STR  = 0x08;                  /* タイマスタート           */
61 :  }
62 :
63 :  /************************************************************************/
64 :  /* ディップスイッチ値読み込み                                           */
65 :  /* 戻り値 スイッチ値 0～15                                              */
66 :  /************************************************************************/
67 :  unsigned char dipsw_get( void )
68 :  {
69 :      unsigned char sw;
70 :
71 :      sw  = ~P6DR;                      /* ディップスイッチ読み込み */
72 :      sw &= 0x0f;
73 :
74 :      return  sw;
75 :  }
76 :
77 :  /************************************************************************/
78 :  /* end of file                                                          */
79 :  /************************************************************************/
```

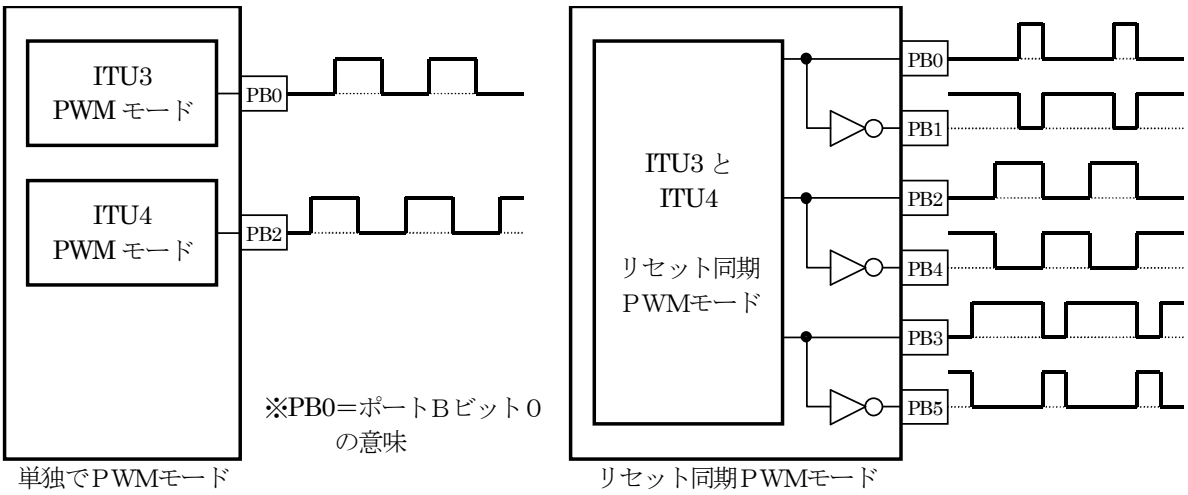
13.5 プログラムの解説

13.5.1 リセット同期PWMモードの設定

```
53 :      ITU3_TCR = 0x23;                /* カウンタ、クリアの設定    */
54 :      ITU_FCR  = 0x3e;                /* リセット同期 PWM モード    */
55 :      ITU3_GRA = 49151;               /* 周期                      */
56 :      ITU3_GRB = ITU3_BRB = 0;        /* デューティ比設定 PB0,1    */
57 :      ITU4_GRA = ITU4_BRA = 0;        /* デューティ比設定 PB2,4    */
58 :      ITU4_GRB = ITU4_BRB = 0;        /* デューティ比設定 PB3,5    */
59 :      ITU_TOER = 0x3f;                /* 出力端子の設定            */
60 :      ITU_STR  = 0x08;                /* タイマスタート            */
```

ITU は 5 チャネルあります。通常 ITU 一つにつき、1 つの PWM 波形を出力することができます。

H8/3048F-ONE には、**リセット同期 PWM モード**という設定があります。これは、ITU3 と ITU4 を組み合わせて使い、PWM 波形を 3 つ出力するというモードです。更に、3 つの PWM 波形を反転した波形も同時に出力します。合計6端子から PWM 出力できます。欠点は、周期を個別に変更できないことです。



●ITU3_TCR(タイマコントロールレジスタ)の設定内容

このレジスタは、
・ITU3_CNT を0にするタイミング
・ITU3_CNT が＋1する時間
を設定します。
各ビットの意味は下記のようになっています。

ビット:	7	6	5	4	3	2	1	0
ITU3_TCR:	－	CCLR1	CCLR0	CKEG1	CKEG0	TPSC2	TPSC1	TPSC0
設定値:	0	0	1	0	0	0	1	1
16 進数:	2				3			

・ビット 6,5:カウンタクリア 1,0

CNT のカウンタクリア要因を選択します。

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

ITU3_CNT＝(ITU3_GRA＋1)になると ITU3_CNT をクリアしなさい、という設定です。

・ビット 2～0:タイマプリスケーラ 2～0

CNT のカウントクロックを選択します。

TPSC2	TPSC1	TPSC0	説明
0	0	0	内部クロック:φでカウント 24.576MHz でカウント、時間に直すと逆数なので 1÷(24.576×10^6)＝40.69[ns] 最大値は 40.69[ns]×65535＝2.666[ms]
0	0	1	内部クロック:φ／2でカウント 1÷(24.576×10^6／2)＝81.38[ns] 最大値は 81.38[ns]×65535＝5.333[ms]
0	1	0	内部クロック:φ／4でカウント 1÷(24.576×10^6／4)＝162.76[ns] 最大値は 162.76[ns]×65535＝10.67[ms]
0	1	1	内部クロック:φ／8でカウント 1÷(24.576×10^6／8)＝325.52[ns] 最大値は 325.52[ns]×65535＝21.33[ms]
1	0	0	外部クロックA:TCLKA 端子(PA0)でカウント
1	0	1	外部クロックB:TCLKB 端子(PA1)でカウント
1	1	0	外部クロックC:TCLKC 端子(PA2)でカウント
1	1	1	外部クロックD:TCLKD 端子(PA3)でカウント

今回は周期が 16[ms]です。周期を 16[ms]にできるのは TPSC＝"011"しかありません。

●ITU_FCR(タイマファンクションコントロールレジスタ)の設定内容

このレジスタでは、チャネル 3,4 をどのモードで使うかの設定、およびバッファ動作の設定を行います。

ビット:	7	6	5	4	3	2	1	0
ITU_FCR:	−	−	CMD1	CMD0	BFB4	BFA4	BFB3	BFA3
設定値:	0	0	1	1	1	1	1	0
16 進数:	3				e			

・ビット 5,4:コンビネーションモード 1,0

チャネル 3,4 を通常動作させるか、相補 PWM モードまたはリセット同期 PWM モードで動作させるかを選択します。

CMD1	CMD0	説明
0	0	チャネル 3,4 は通常動作
0	1	
1	0	チャネル 3,4 を組み合わせ、相補 PWM モードで動作
1	1	チャネル 3,4 を組み合わせ、リセット同期 PWM モードで動作

ITU3,4 は、通常動作、相補 PWM モード、リセット同期 PWM モードを選択することができます。今回は、リセット同期 PWM モードを使用しますので"11"にします。

その他のビットは後述します。

●ITU_TOER(タイマアウトプットマスタイネーブルレジスタ)の設定内容

PWM 出力端子にするか、通常の I/O ポートとするかの設定です。

ビット:	7	6	5	4	3	2	1	0
ITU_TOER:	−	−	EXB4	EXA4	EB3	EB4	EA4	EA3
設定値:	0	0	1	1	1	0	0	0
16 進数:	3				8			

・ビット 5:マスタイネーブル TOCXB4(EXB4)

TOCXB4 端子の ITU 出力を許可／禁止します。

EXB4	説明
0	TFCR の設定にかかわらず TOCXB4 端子の出力は禁止 (TOCXB4 端子は入出力ポートとして動作) XTGD＝0 の状態で、チャネル 1 のインプットキャプチャ A が発生したとき 0 にクリア
1	TFCR の設定に従い TOCXB4 端子の出力は許可

・ビット 4:マスタイネーブル TOCXA4(EXA4)

TOCXA4 端子の ITU 力を許可／禁止します。

EXA4	説明
0	TFCR の設定にかかわらず TOCXA4 端子の出力は禁止 (TOCXA4 端子は入出力ポートとして動作) XTGD＝0 の状態で、チャネル 1 のインプットキャプチャ A が発生したとき 0 にクリア
1	TFCR の設定に従い TOCXA4 端子の出力は許可

・ビット 3:マスタイネーブル TIOCB3(EB3)

TIOCB3 端子の ITU 出力を許可／禁止します。

EB3	説明
0	TIOR3、TFCR の設定にかかわらず TIOCB3 端子の出力は禁止 XTGD＝0 の状態で、チャネル 1 のインプットキャプチャ A が発生したとき 0 にクリア
1	TIOR3、TFCR の設定に従い TIOCB3 端子の出力は許可

・ビット 2:マスタイネーブル TIOCB4(EB4)

TIOCB4 端子の ITU 出力を許可／禁止します。

EB4	説明
0	TIOR4、TFCR の設定にかかわらず TIOCB4 端子の出力は禁止 (TIOCB4 端子は入出力ポートとして動作) XTGD＝0 の状態で、チャネル 1 のインプットキャプチャ A が発生したとき 0 にクリア
1	TIOR4、TFCR の設定に従い TIOCB4 端子の出力は許可

・ビット 1: マスタイネーブル TIOCA4(EA4)

TIOCA4 端子の ITU 出力を許可／禁止します。

EA4	説明
0	TIOR4、TMDR、TFCR の設定にかかわらず TIOCA4 端子の出力は禁止 (TIOCA4 端子は入出力ポートとして動作) XTGD＝0 の状態で、チャネル 1 のインプットキャプチャ A が発生したとき 0 にクリア
1	TIOR4、TMDR、TFCR の設定に従い TIOCA4 端子の出力は許可

・ビット 0:マスタイネーブル TIOCA3(EA3)

TIOCA3 端子の ITU 出力を許可／禁止します。

EA3	説明
0	TIOR3、TMDR、TFCR の設定にかかわらず TIOCA3 端子の出力は禁止 (TIOCA3 端子は入出力ポートとして動作) XTGD＝0 の状態で、チャネル 1 のインプットキャプチャ A が発生したとき 0 にクリア
1	TIOR3、TMDR、TFCR の設定に従い TIOCA3 端子の出力は許可

ハードウェアマニュアルの内容をそのまま書きましたが、難しい表現が多すぎてよく分かりません。リセット同期 PWM モードを使用すると、PB5～0 から PWM 波形が自動的に出力されます。このとき、PWM 波形は必要なく、通常の I/O ポートとして使用したい場合、**ITU_TOER レジスタを設定することにより PWM 出力を OFF にすることができます**。

分かりやすいように表にして書き直しました。今回は、実習なのですべて出力に設定します。

TOER	7	6	5	4	3	2	1	0
"0"のとき	－	－	PB5 は通常の入出力ポート	PB4 は通常の入出力ポート	PB1 は通常の入出力ポート	PB3 は通常の入出力ポート	PB2 は通常の入出力ポート	PB0 は通常の入出力ポート
"1"のとき	－	－	PB5 は PWM 出力	PB4 は PWM 出力	PB1 は PWM 出力	PB3 は PWM 出力	PB2 は PWM 出力	PB0 は PWM 出力
今回の設定	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**

ちなみに、モータドライブ基板 Vol.3 は、PB5 はサーボ用 PWM、PB4 は右モータ用 PWM、PB1 は左モータ PWM として使用します。それ以外は通常の入出力ポートとして使用します。下記のようになります。

	7	6	5	4	3	2	1	0
用途	－	－	PB5 は サーボ	PB4 は 右モータ	PB1 は 左モータ	PB3 は通常の入出力ポート	PB2 は通常の入出力ポート	PB0 は通常の入出力ポート
TOER の設定	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**0**

16 進数に直して、0x38 を ITU_TOER へ書き込みます。

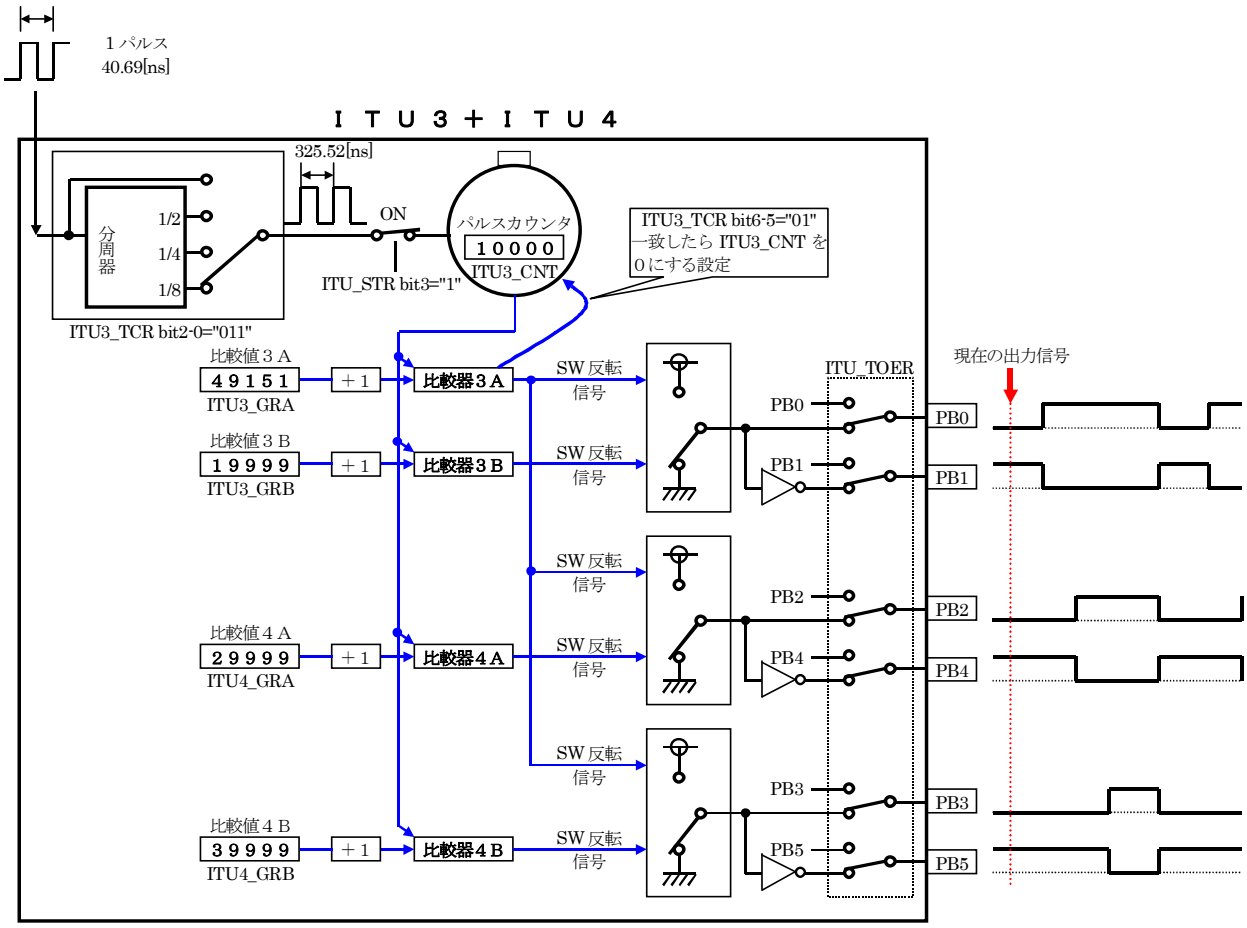
●ITU3_CNT,ITU4_CNT,ITU3_GRA,ITU3_GRB,ITU4_GRA,ITU4_GRB の内容

リセット同期 PWM モードを使用したとき、各レジスタの機能は下表のようになります。

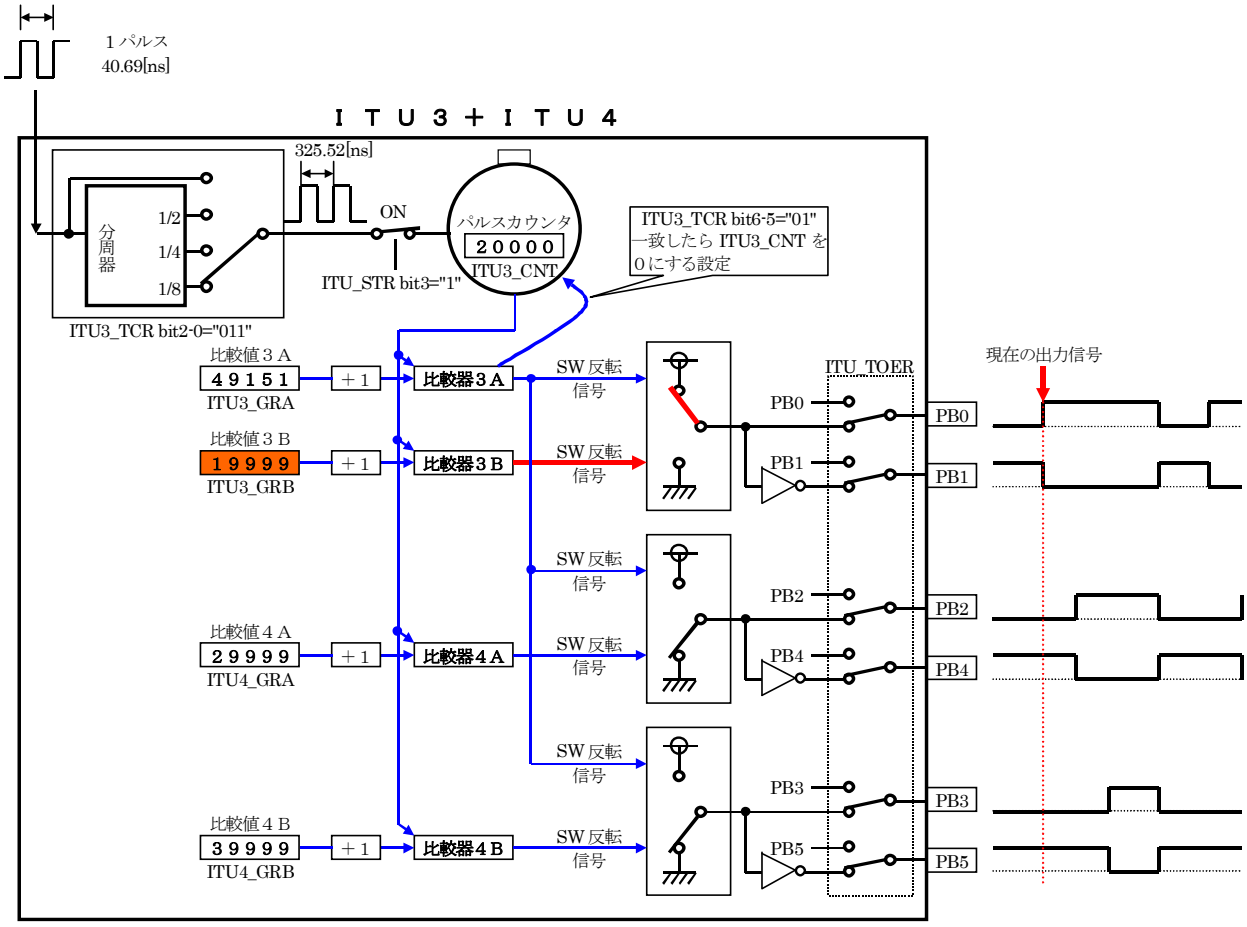
レジスタ	リセット同期 PWM モードで使用したときの機能
ITU3_CNT	カウンタとして使用します。
ITU4_CNT	未使用です。
ITU3_GRA	PWM 周期を設定します。
ITU3_GRB	PB0,PB1 端子より出力される PWM 波形の設定 PB0 端子より ITU3_GRB で設定したデューティ比の波形が出力されます。PB1 端子からは、PB0 端子の反転波形が出力されます。
ITU4_GRA	PB2,PB4 端子より出力される PWM 波形の設定 PB2 端子より ITU4_GRA で設定したデューティ比の波形が出力されます。PB4 端子からは、PB2 端子の反転波形が出力されます。
ITU4_GRB	PB3,PB5 端子より出力される PWM 波形の設定 PB3 端子より ITU4_GRB で設定したデューティ比の波形が出力されます。PB5 端子からは、PB3 端子の反転波形が出力されます。

13.5.2 PWM出力される仕組み

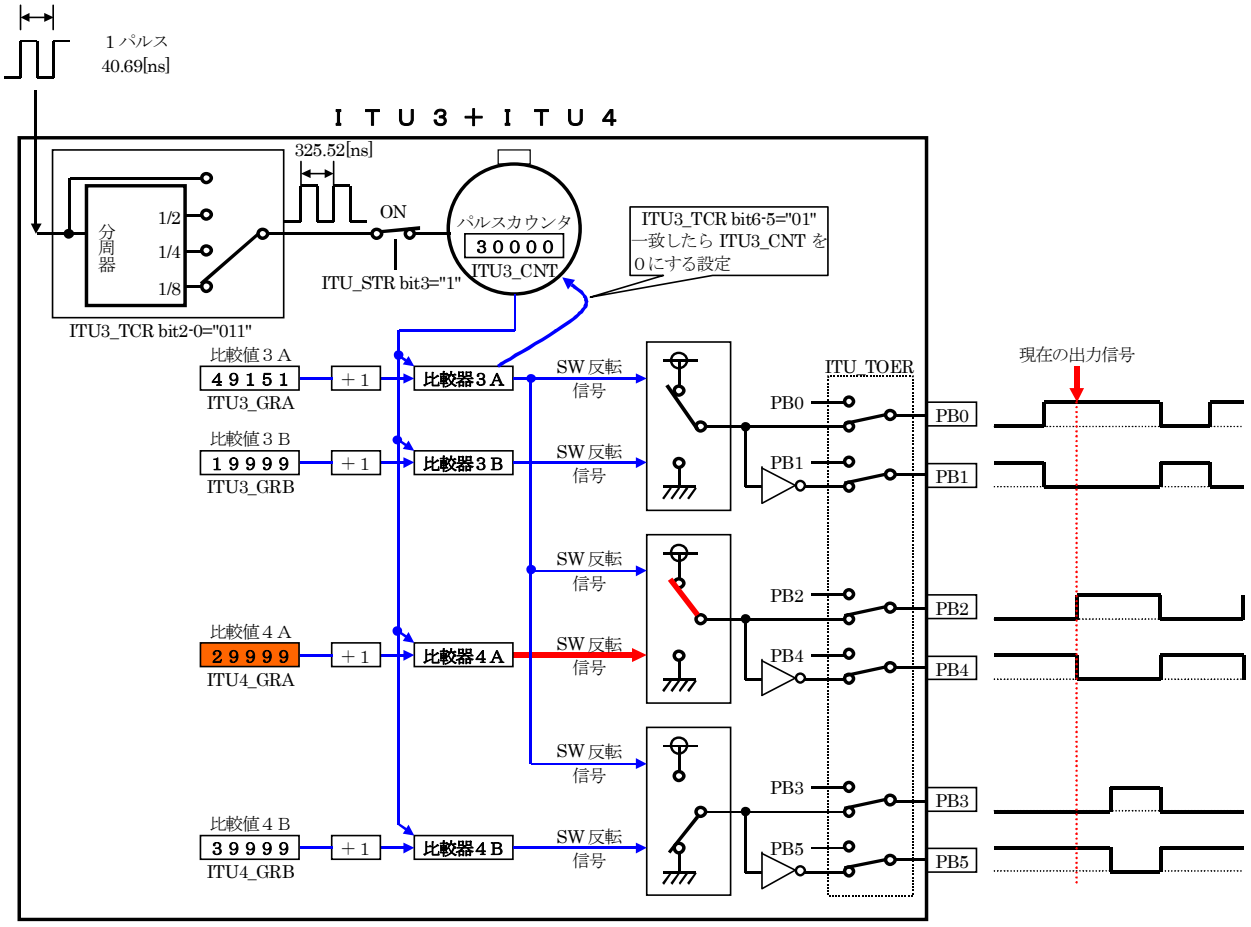
PWM 出力される仕組みを図解で説明します。



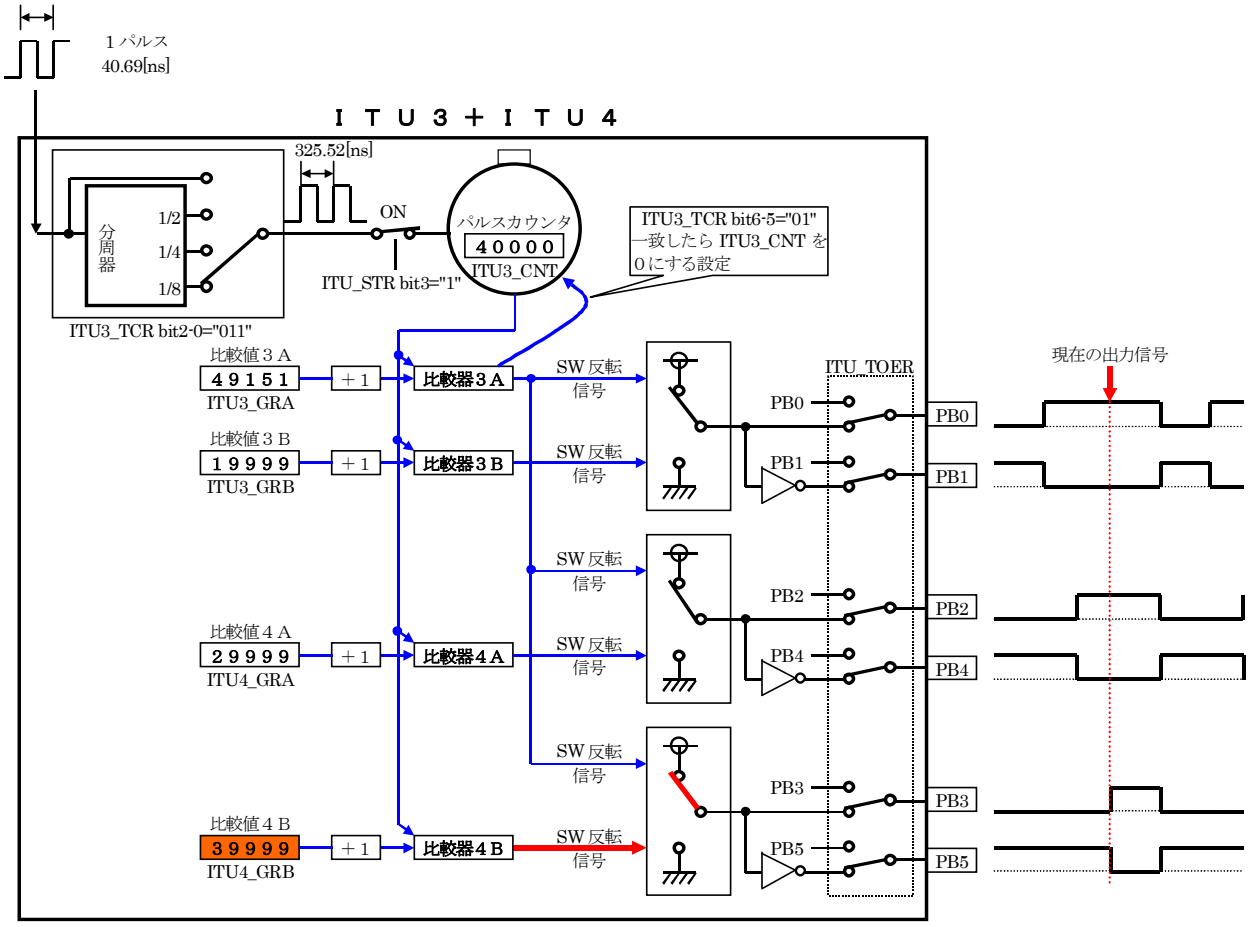
1. リセット同期 PWM モードは、ITU3_CNT を共通にして、3 つの PWM 波形と反転された波形が出力されます。
2. それぞれ、PWM 出力端子が決まっており、ITU3_GRB に関わる端子は PB0 と PB1、ITU4_GRA に関わる端子は PB2 と PB4、ITU4_GRB に関わる端子は PB3 と PB5 になります。PB0,2,3 は"0"から始まる PWM 波形、PB1,4,5 はそれぞれの端子出力の反転波形が出力されます。
3. ITU_TOER でそれぞれの端子を PWM 出力として使用するか、入出力ポートとして使用するかを選択することができます。



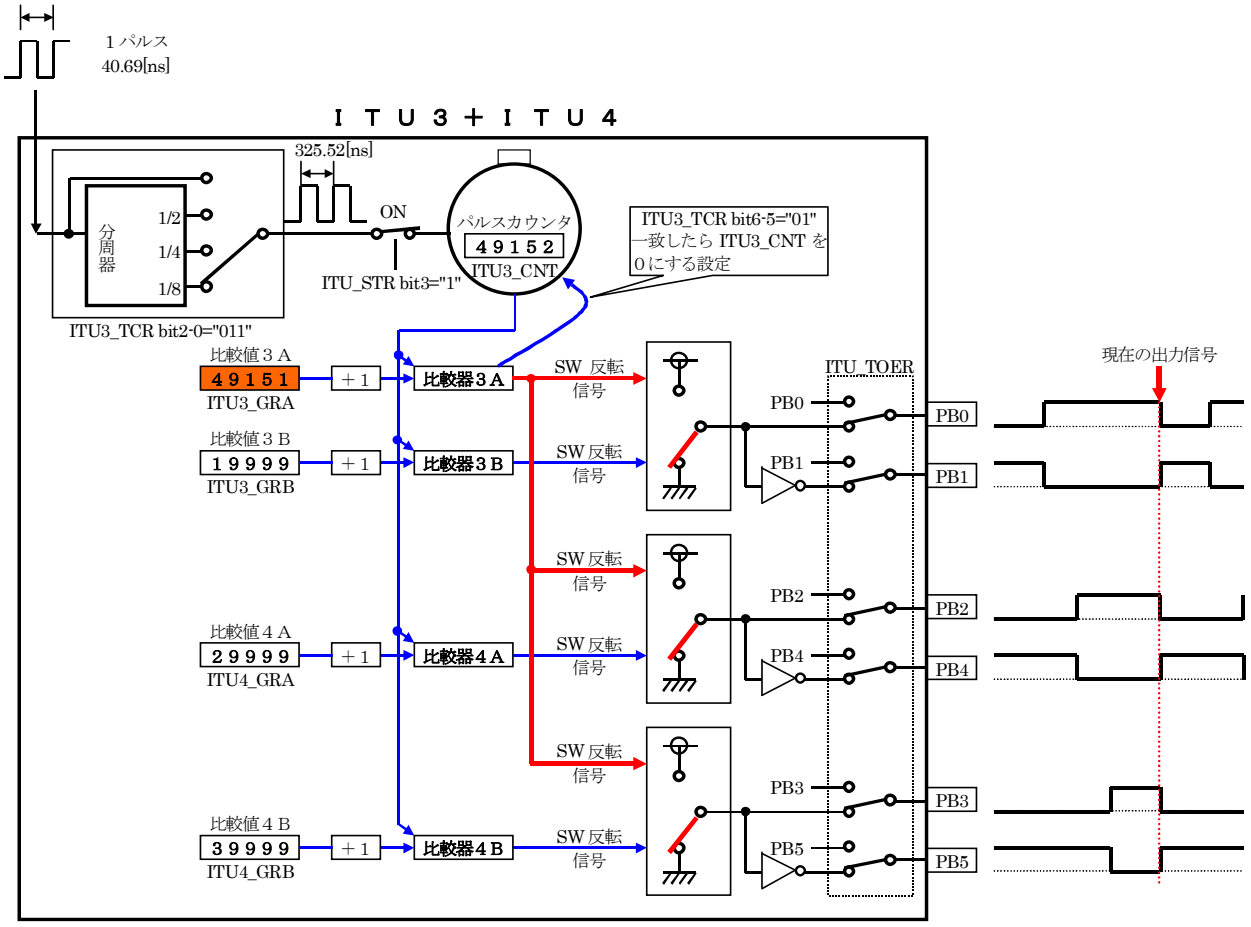
4.「ITU3_CNT＝(ITU3_GRB+1)」になると、PB0 と PB1 の出力波形は反転します。
PB0 は今まで"0"でしたが、反転して"1"になります。PB1 の OFF 幅は
ITU3_CNT の1カウント分の時間×(ITU3_GRB+1)
となります。
PB1 は今まで"1"でしたが、反転して"0"になります。PB1 の ON 幅は
ITU3_CNT の1カウント分の時間×(ITU3_GRB+1)
となります。



5.「ITU3_CNT＝(ITU4_GRA+1)」になると、PB2 と PB4 の出力波形は反転します。
PB2 は今まで"0"でしたが、反転して"1"になります。PB2 の OFF 幅は
ITU3_CNT の1カウント分の時間×(ITU4_GRA+1)
となります。
PB4 は今まで"1"でしたが、反転して"0"になります。PB4 の ON 幅は
ITU3_CNT の1カウント分の時間×(ITU4_GRA+1)
となります。

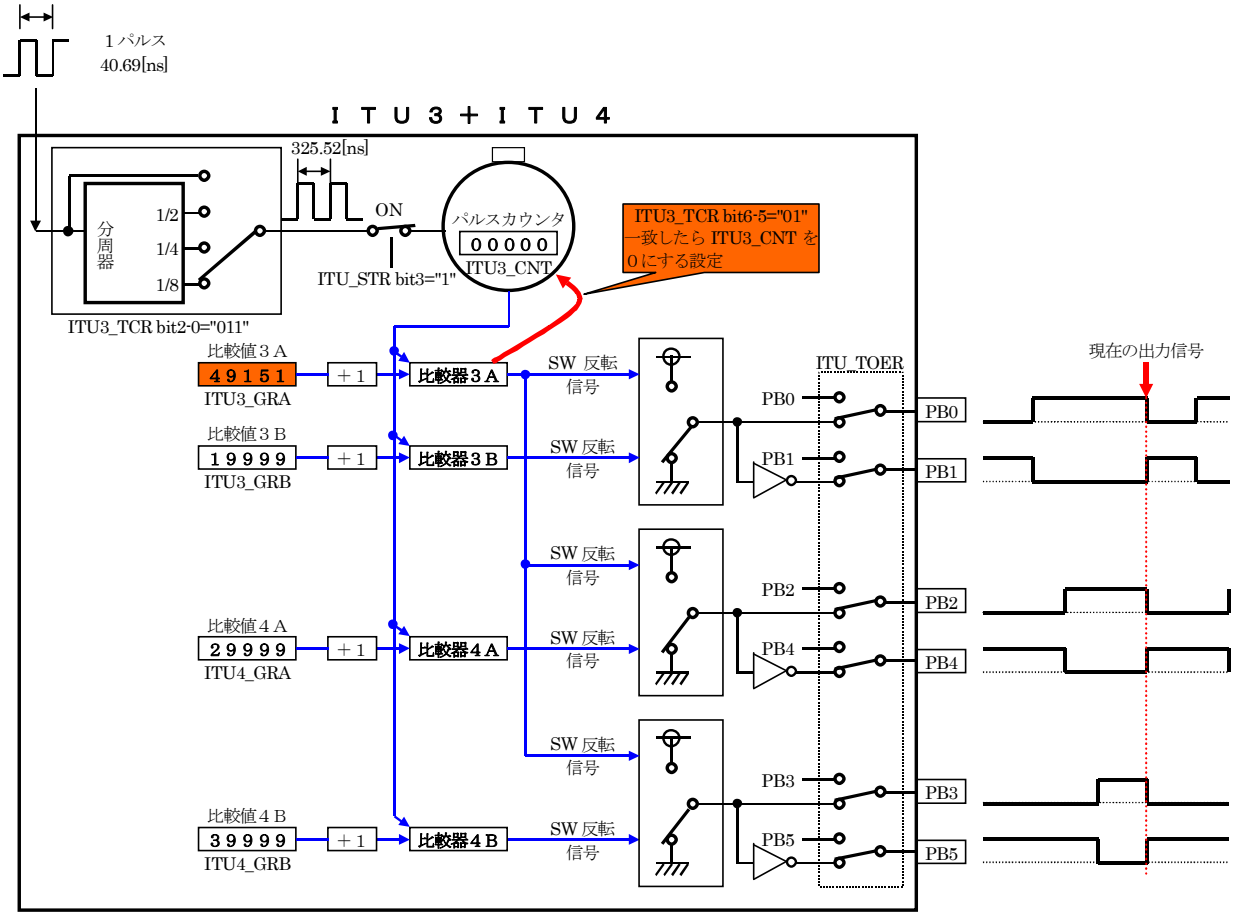


6.「ITU3_CNT＝(ITU4_GRB+1)」になると、PB3 と PB5 の出力波形は反転します。
PB3 は今まで"0"でしたが、反転して"1"になります。PB3 の OFF 幅は
ITU3_CNT の1カウント分の時間×(ITU4_GRB+1)
となります。
PB5 は今まで"1"でしたが、反転して"0"になります。PB5 の ON 幅は
ITU3_CNT の1カウント分の時間×(ITU4_GRB+1)
となります。



7.「ITU3_CNT＝(ITU3_GRA+1)」になると、全波形が反転します。

※次で説明しますが、「ITU3_CNT＝(ITU3_GRA+1)」になる瞬間に ITU3_CNT はクリアされます。そのため、実際には 49152 にはなりません。

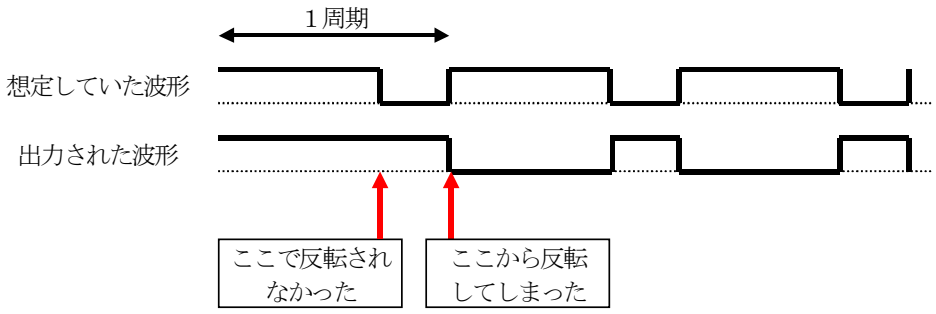


8. ITU3_TCR で、「ITU3_CNT＝(ITU3_GRA+1)になるとITU3_CNT をクリア」と設定しているので、ITU3_CNT は0になりまたカウントを開始します。ITU_CNT は、「0 → 1 → 2 → … → 49150 → 49151 → 0 → 1→ …」という動作になります。

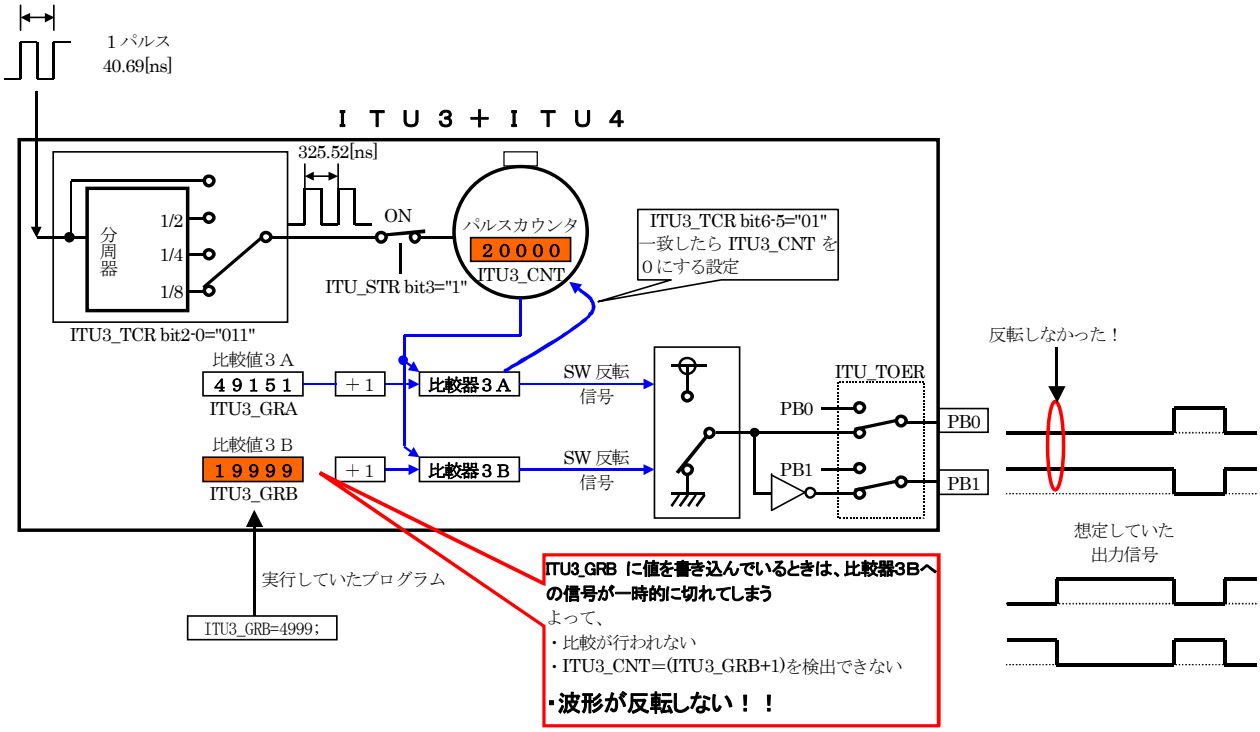
13.5.3 リセット同期PWMモードの注意点

リセット同期 PWM モードを使うに当たって、いくつか注意点があります。

(1) 波形がおかしくなることがある



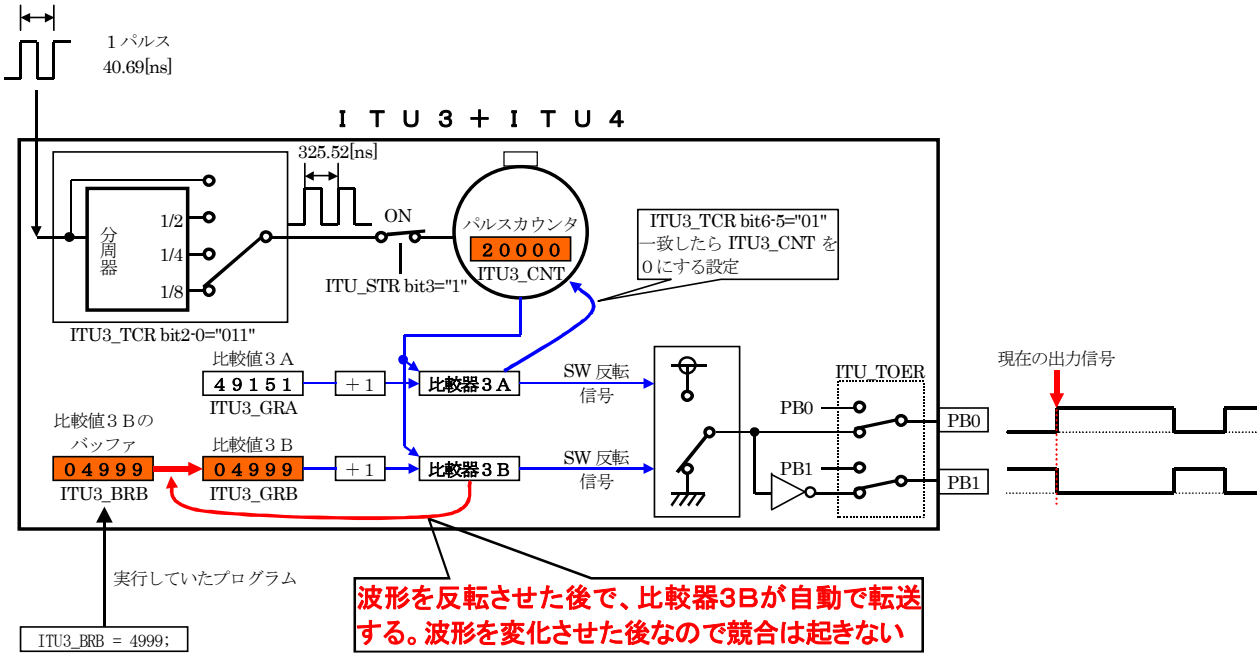
デューティ比を変えるために、ITU3_GRB、ITU4_GRA、ITU4_GRB を書き換えます。各ジェネラルレジスタの値の書き換えと、(ITU3_CNT＝各ジェネラルレジスタ＋1)になるときが重なってしまうと、**波形の反転がされずに書き換えを優先的に行ってしまいます。**



これを防止する機能が備わっています。それがバッファレジスタというレジスタです。

レジスタ名	説明
ITU3_BRA	ITU3_GRA のバッファ(ITU3バッファレジスタ A)です。今回は、ITU3_GRA は周期設定用でいったん設定すると変えることはないのでバッファにしません。
ITU3_BRB	ITU3_GRB のバッファ(ITU3バッファレジスタ B)です。プログラムでは ITU3_GRB へ値を書き込まずに ITU3_BRB へ書き込み、競合を回避します。
ITU4_BRA	ITU4_GRA のバッファ(ITU4バッファレジスタ A)です。プログラムでは ITU4_GRA へ値を書き込まずに ITU4_BRA へ書き込み、競合を回避します。
ITU4_BRB	ITU4_GRB のバッファ(ITU4バッファレジスタ B)です。プログラムでは ITU4_GRB へ値を書き込まずに ITU4_BRB へ書き込み、競合を回避します。

バッファレジスタを使用に設定すると波形が反転した後に、**自動でバッファレジスタの値がジェネラルレジスタへ転送されます**。メインプログラムではあくまでもバッファレジスタに値を書き込んでいますので競合が起きることは有りません。



この「**自動で転送する**」ようにする設定が、先ほど説明した ITU_FCR なのです。

●ITU_FCR(タイマファンクションコントロールレジスタ)の設定内容

ビット:	7	6	5	4	3	2	1	0
ITU_FCR:	−	−	CMD1	CMD0	BFB4	BFA4	BFB3	BFA3
設定値:	0	0	1	1	1	1	1	0
16 進数:	3				e			

・ビット 5,4:コンビネーションモード 1,0

チャネル 3,4 を通常動作させるか、相補 PWM モードまたはリセット同期 PWM モードで動作させるかを選択します。

CMD1	CMD0	説明
0	0	チャネル 3,4 は通常動作
0	1	
1	0	チャネル 3,4 を組み合わせ、相補 PWM モードで動作
1	1	チャネル 3,4 を組み合わせ、リセット同期 PWM モードで動作

これは先ほど説明したとおりです。

・ビット 3:バッファ動作 B4(BFB4)

ITU4_GRB を通常動作とするか、ITU4_GRB と ITU4_BRB を組み合わせてバッファ動作とするかを選択します。

BFB4	説明
0	ITU4_GRB は通常動作
1	ITU4_BRB はバッファ動作

ITU4_GRB を使いますので、バッファ動作とします。

・ビット 2:バッファ動作 A4(BFA4)

ITU4_GRA を通常動作とするか、ITU4_GRA と ITU4_BRA を組み合わせてバッファ動作とするかを選択します。

BFA4	説明
0	ITU4_GRA は通常動作
1	ITU4_BRA はバッファ動作

ITU4_GRA を使いますので、バッファ動作とします。

・ビット 1:バッファ動作 B3(BFB3)

ITU3_GRB を通常動作とするか、ITU3_GRB と ITU3_BRB を組み合わせてバッファ動作とするかを選択します。

BFB3	説明
0	ITU3_GRB は通常動作
1	ITU3_BRB はバッファ動作

ITU3_GRB を使いますので、バッファ動作とします。

・ビット 0:バッファ動作 A3(BFA3)

ITU3_GRA を通常動作とするか、ITU3_GRA と ITU3_BRA を組み合わせてバッファ動作とするかを選択します。

BFA3	説明
0	ITU3_GRA は通常動作
1	ITU3_BRA はバッファ動作

ITU3_GRA は周期レジスタで、いったん設定するともう変更することはありません。競合することはないので、こちらは通常動作とします。

```
55 :      ITU3_GRA = 49151;                  /* 周期                        */
```

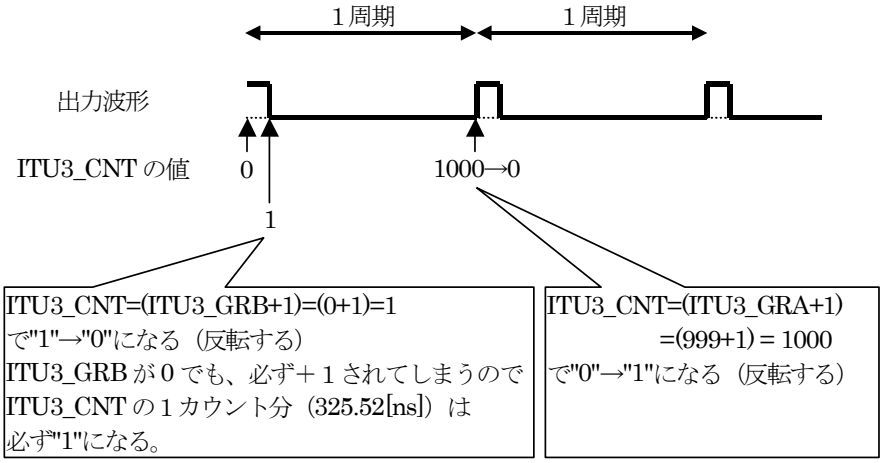
周期を設定します。

```
56 :      ITU3_GRB = ITU3_BRB = 0;           /* デューティ比設定 PB0,1    */
57 :      ITU4_GRA = ITU4_BRA = 0;           /* デューティ比設定 PB2,4    */
58 :      ITU4_GRB = ITU4_BRB = 0;           /* デューティ比設定 PB3,5    */
```

デューティ比を設定します。最初は0にしておきます。同時にバッファレジスタもクリアしておきます。ITU3_GRB、ITU4_GRA、ITU4_GRB に値を書き込むのは、この一回限りです。

(2) 0%出力できない

ITU3_GRB で反転した波形(PB1)を例にします。波形が反転するタイミングは、
・ITU3_CNT＝(ITU3_GRB+1) このとき、"1"→"0"になる(反転する)
・ITU3_CNT＝(ITU3_GRA+1) このとき、"0"→"1"になる(反転する)
のときです。
もちろん、0%にしたいのでデューティ比を決める ITU3_GRB には0を設定します。ITU3_GRA は 999 とします。



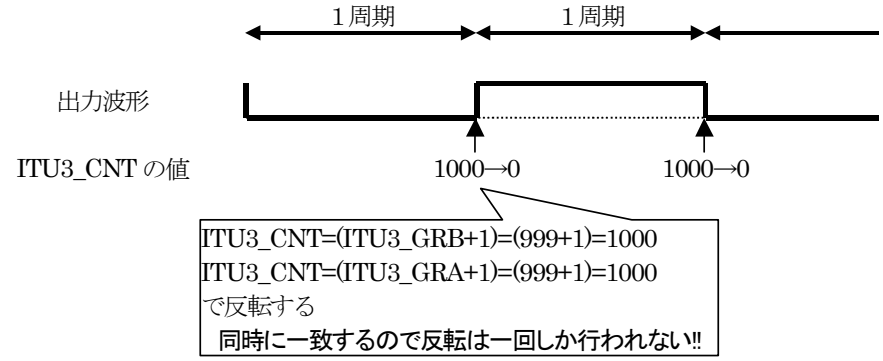
このように完全な 0%はできません。必ず ITU3_CNT の1カウント分ONになってしまいます。今回は周期 16[ms]です。"1"になる割合は、

$(325.52\times10^{-9})\div(16\times10^{-3})=0.00002=0.002\%$

ほとんど無視できます。0%と言っても問題ないです。

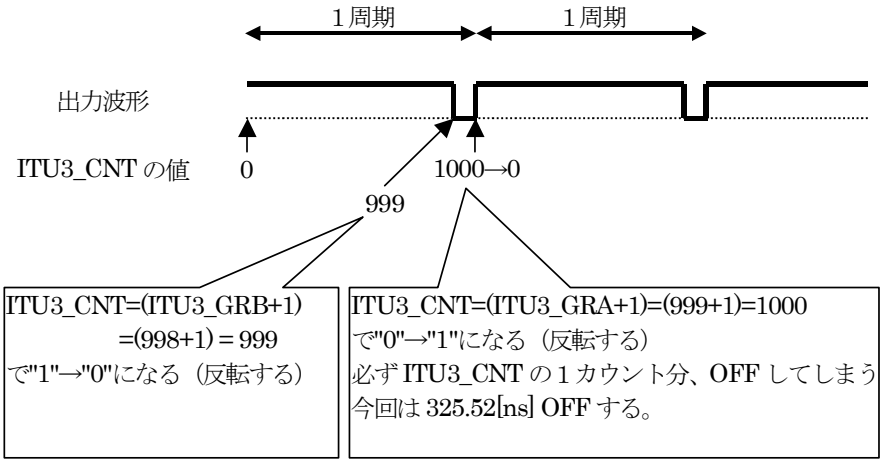
(3) 100%出力できない

ITU3_GRB で反転した波形(PB1)を例にします。波形が反転するタイミングは、
・ITU3_CNT＝(ITU3_GRB+1) このとき、"1"→"0"になる(反転する)
・ITU3_CNT＝(ITU3_GRA+1) このとき、"0"→"1"になる(反転する)
のときです。
もちろん、100%にしたいのでデューティ比を決める ITU3_GRB には周期である ITU3_GRA の値を設定します。ITU3_GRA は 999 とします。



このように ITU3_GRA(周期)と ITU3_GRB(デューティ比)の値を同じにすると、ITU3_CNT と一致するタイミングが同じになってしまい反転が一回しか行われません。結果、1 周期 0%、次の 1 周期 100%という波形になり周期 16[ms]の PWM ではない波形になってしまいます。

そこで、ITU3_GRB には ITU3_GRA より1小さい値を書き込みます。



このように、完全な 100%は出力できません。必ず ITU3_CNT の 1 カウント分 OFF になってしまいます。

13.5.4 main関数

```
25 :  void main( void )
26 :  {
27 :      init();                                /* 初期化                    */
28 :
29 :      while( 1 ) {
30 :          ITU3_BRB = 49150 * dipsw_get() / 15;
31 :      }
32 :  }
```

29 行目の while 文のカッコ内は1で常に真なので 30 行目は繰り返されます。

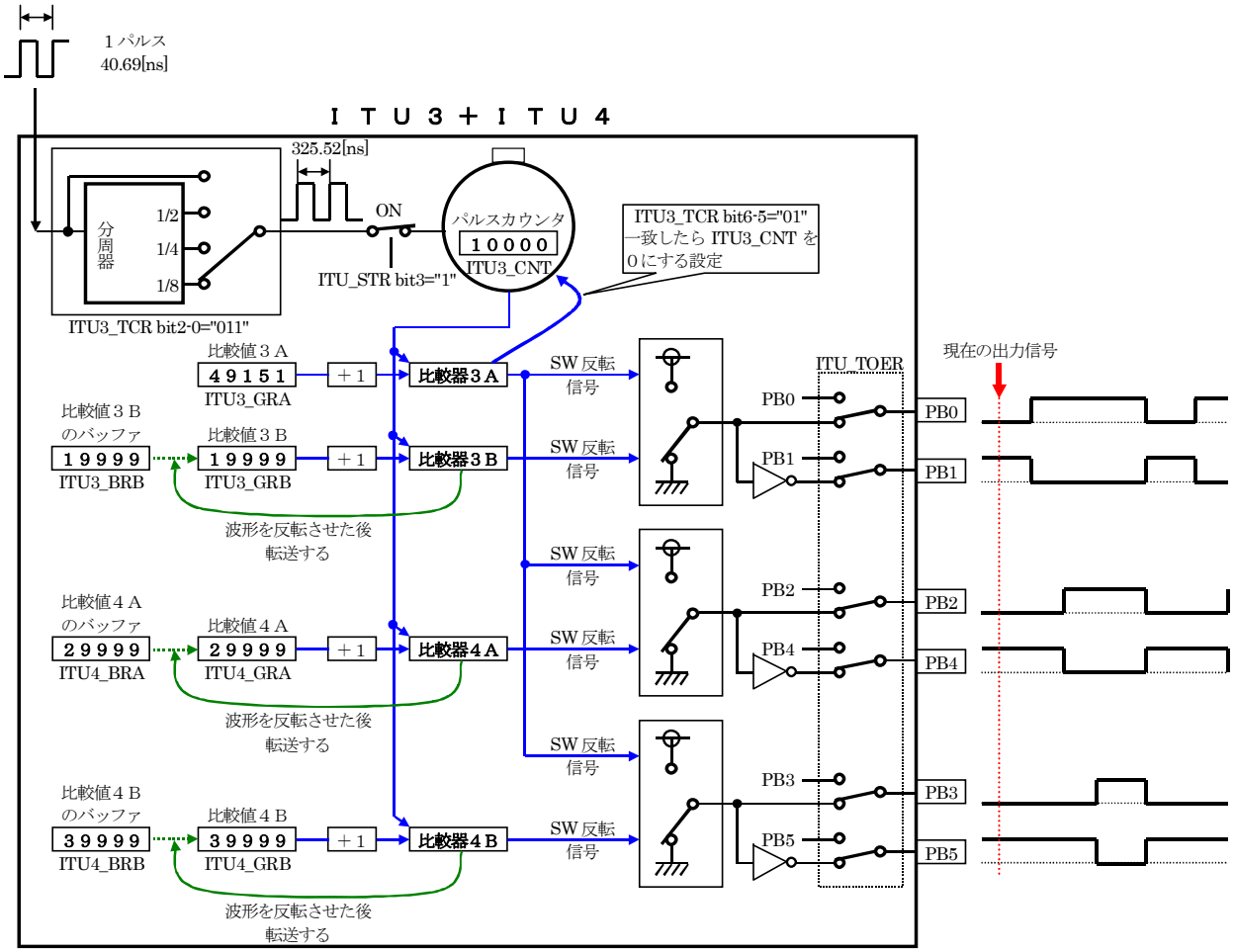
30 行目で、ディップスイッチの値により ITU3_BRB へ代入する数値を決めてデューティ比を可変にしています。ディップスイッチの値とデューティ比をまとめておきます。

ディップスイッチの値	ITU3_BRB の値	デューティ比
0	49150 * 0 / 15 = 0	0% ※1
1	49150 * 1 / 15 = 3276	6.6%
2	49150 * 2 / 15 = 6553	13.3%
3	49150 * 3 / 15 = 9830	20.0%
4	49150 * 4 / 15 = 13106	26.7%
5	49150 * 5 / 15 = 16383	33.3%
6	49150 * 6 / 15 = 19660	40.0%
7	49150 * 7 / 15 = 22936	46.7%
8	49150 * 8 / 15 = 26213	53.3%
9	49150 * 9 / 15 = 29490	60.0%
10	49150 * 10 / 15 = 32766	66.7%
11	49150 * 11 / 15 = 36043	73.3%
12	49150 * 12 / 15 = 39320	80.0%
13	49150 * 13 / 15 = 42596	86.7%
14	49150 * 14 / 15 = 45873	93.3%
15	49150 * 15 / 15 = 49150	100% ※2

※1…完全な 0%ではなく、ITU3_CNT の1カウント分(325.52ns) だけ ON になります。
※2…完全な 100%ではなく、ITU3_CNT の 1 カウント分 (325.52ns) だけ OFF になります。

プログラムの ITU3_BRB を、ITU4_BRA や ITU4_BRB に変えて、ディップスイッチに応じてそれぞれの PWM 出力端子から PWM 出力信号が出力されるか確かめてみてください。

13.6 まとめ



設定するレジスタ	詳細
ITU3_TCR	ITU3_CNT の値が＋1する時間、クリア要因の設定をします。 0x20…周期が 2.666ms 以下の場合(＋1する時間は 40.69[ns]ごと) 0x21…周期が 5.333ms 以下の場合(＋1する時間は 81.38[ns]ごと) 0x22…周期が 10.67ms 以下の場合(＋1する時間は 162.76[ns]ごと) 0x23…周期が 21.33ms 以下の場合(＋1する時間は 325.52[ns]ごと) 今回は、周期 16ms なので **0x23** を設定します。
ITU_FCR	リセット同期PWMモードの設定と、バッファレジスタを使用するかの設定します。 **0x3e** を設定します。PB0～PB5 の端子からは PWM 波形がされます。出力したくない場合は、ITU_TOER を設定することにより通常の I/O ポートとして使用できます。
ITU3_GRA	周期を設定します。計算は、「設定したい周期÷(ITU3_CNT の値が＋1する時間)－1」です。周期 16ms、＋1する時間 325.52ns なら $(16\times10^{-3})\div(325.52\times10^{-9})-1=49151$
ITU3_BRB ITU3_GRB	PB0 の OFF 幅、または PB1 の ON 幅を設定します。 計算は、「設定したい幅÷(ITU3_CNT の値が＋1する時間)－1」です。 ITU3_GRB は、最初だけ ITU3_BRB と同じ値を設定します。その後は、バッファレジスタ ITU3_BRB に設定します。

<table>
<tr><td>ITU4_BRA
ITU4_GRA</td><td>PB2 の OFF 幅、または PB4 の ON 幅を設定します。
計算は、「設定したい幅÷(ITU3_CNT の値が＋1する時間)－1」です。
ITU4_GRA は、最初だけ ITU4_BRA と同じ値を設定します。その後は、バッファレジスタ ITU4_BRA に設定します。</td></tr>
<tr><td>ITU4_BRB
ITU4_GRB</td><td>PB3 の OFF 幅、または PB5 の ON 幅を設定します。
計算は、「設定したい幅÷(ITU3_CNT の値が＋1する時間)－1」です。
ITU4_GRB は、最初だけ ITU4_BRB と同じ値を設定します。その後は、バッファレジスタ ITU4_BRB に設定します。</td></tr>
<tr><td>ITU_TOER</td><td>PWM 波形を出力するかしないか選択します。
″1″:PWM波形出力 ″0″:通常の I/O ポートとして使用
bit 7 6 5 4 3 2 1 0
0固定 0固定 PB5 PB4 PB1 PB3 PB2 PB0

例)PB5,1,2 を PWM として使用、他は I/O ポートなら
PB5,1,2 の部分を″1″に、PB4,3,0 の部分を″0″にします。
bit 7 6 5 4 3 2 1 0
値 0 0 1 0 1 0 1 0

0010 1010→0x2a を設定します。</td></tr>
<tr><td>ITU_STR</td><td>それぞれの ITU のチャネルで ITU_CNT をカウント動作させるかどうか選択します。要は、ITU を使うか使わないかの設定です。
″1″:使用する ″0″:使用しない
bit 7 6 5 4 3 2 1 0
0固定 0固定 0固定 ITU4 ITU3 ITU2 ITU1 ITU0

今回は ITU3 を使用するので、ITU3 の部分が″1″、他は″0″なので 0x08 を設定します。
ITU4 なら 0x10 を設定します。リセット同期 PWM モードは ITU3 と ITU4 を組み合わせて使用しますが、ITU3 の部分のみ″1″にするだけで OK です。</td></tr>
</table>

●リセット同期 PWM モードの良いところ

・ITU3、ITU4 の 2 つの ITU で、3 つの PWM 波形とそれぞれの反転した PWM 波形を出力できる。
・バッファレジスタを使うことによりプログラムの負担が減る。

●リセット同期 PWM モードの使いづらいところ

・ITU3 と ITU4 のペアでしかリセット同期 PWM モードを設定できない。
・完全な 0%、完全な 100%出力ができない。
・周期は、共通にしか設定できない。

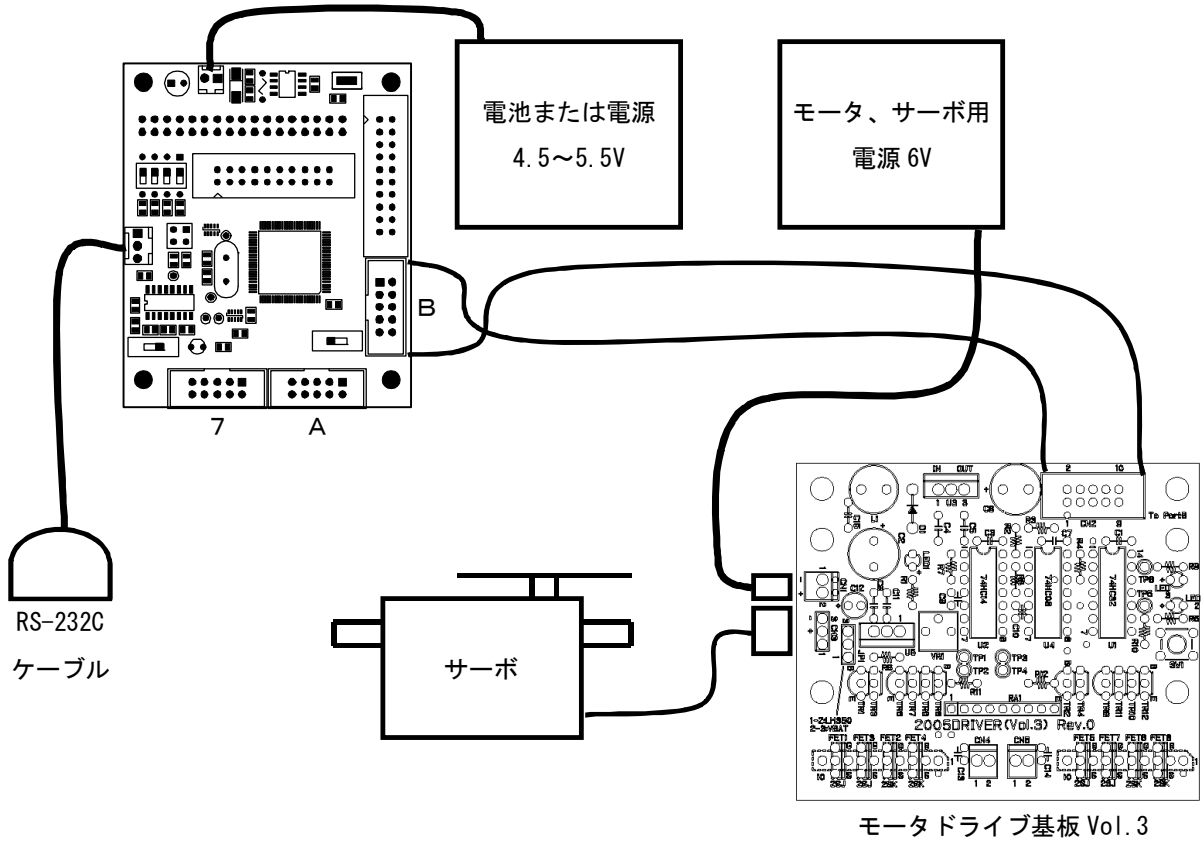
14. プロジェクト「servo」 サーボ制御

14.1 概要

サーボを制御します。マイコンのポートは、下記を使用します。
・ポート B・・・モータドライブ基板 Vol.3 を接続

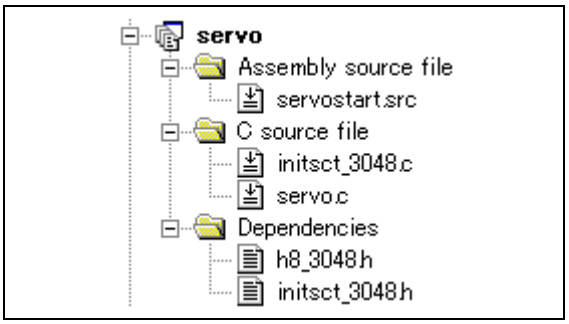
14.2 接続

・CPU ボードのポート B と、モータドライブ基板 Vol.3 をフラットケーブルで接続します。
・モータドライブ基板には、サーボとモータ・サーボ用電源を接続します。



モータドライブ基板 Vol.3

14.3 プロジェクトの構成



	ファイル名	内容
1	servostart.src	アセンブリ言語で記述されたアセンブリソースファイルです。このファイルの構造は下記のようになっています。 ベクタアドレス ＋ スタートアップルーチン
2	initsct_3048.c	初期値のないグローバル変数(セクション B 領域)、初期値のあるグローバル変数(セクション R 領域)の初期化用です。
3	servo.c	実際に制御するプログラムが書かれています。H8/3048F-ONE の内蔵周辺機能の初期化も行います。
4	h8_3048.h	H8/3048F-ONE の内蔵周辺機能の I/O レジスタを定義したファイルです。
5	initsct_3048.h	initsct_3048.c のヘッダファイルです。

14.4 プログラム「servo.c」

```
 1 :  /*********************************************************************/
 2 :  /* サーボのテスト「servo.c」                                         */
 3 :  /* 入力:DIP SW  出力：PB←モータドライブ基板(Vol.3)←サーボモータ     */
 4 :  /*                          2005.04 ジャパンマイコンカーラリー実行委員会  */
 5 :  /*********************************************************************/
 6 :  /*====================================*/
 7 :  /* インクルード                       */
 8 :  /*====================================*/
 9 :  #include    <machine.h>
10 :  #include    "h8_3048.h"
11 :
12 :  /*====================================*/
13 :  /* シンボル定義                       */
14 :  /*====================================*/
15 :
16 :  /*====================================*/
17 :  /* プロトタイプ宣言                   */
18 :  /*====================================*/
19 :  void init( void );
20 :  unsigned char dipsw_get( void );
21 :
22 :  /*********************************************************************/
23 :  /* メインプログラム                                                  */
24 :  /*********************************************************************/
25 :  void main( void )
26 :  {
27 :      init();                             /* 初期化                    */
28 :
29 :      while( 1 ) {
30 :          ITU4_BRB = 5000 + dipsw_get() * 26;
31 :      }
32 :  }
33 :
34 :  /*********************************************************************/
35 :  /* H8/3048F-ONE 内蔵周辺機能の初期化                                 */
36 :  /*********************************************************************/
37 :  void init( void )
38 :  {
39 :      /* ポートの入出力設定 */
40 :      P1DDR = 0xff;
41 :      P2DDR = 0xff;
42 :      P3DDR = 0xff;
43 :      P4DDR = 0xff;
44 :      P5DDR = 0xff;
45 :      P6DDR = 0xf0;                       /* CPU基板上のDIP SW         */
46 :      P8DDR = 0xff;
47 :      P9DDR = 0xf7;
48 :      PADDR = 0xff;
49 :      PBDR  = 0xc0;
50 :      PBDDR = 0xfe;                       /* モータドライブ基板Vol.3  */
51 :      /* ポート７は、入力専用なので入出力設定はありません    */
52 :
53 :      /* ITU3,4 リセット同期PWMモード 左右モータ、サーボ用 */
54 :      ITU3_TCR = 0x23;                    /* カウンタ、クリアの設定    */
55 :      ITU_FCR  = 0x3e;                    /* リセット同期PWMモード     */
56 :      ITU3_GRA = 49151;                   /* 周期の設定                */
57 :      ITU3_GRB = ITU3_BRB = 0;            /* 左モータのPWM設定         */
58 :      ITU4_GRA = ITU4_BRA = 0;            /* 右モータのPWM設定         */
59 :      ITU4_GRB = ITU4_BRB = 5000;         /* サーボのPWM設定           */
60 :      ITU_TOER = 0x38;                    /* 出力端子の設定            */
61 :      ITU_STR  = 0x08;                    /* タイマスタート            */
62 :  }
63 :
64 :  /*********************************************************************/
65 :  /* ディップスイッチ値読み込み                                        */
66 :  /* 戻り値 スイッチ値 0～15                                           */
67 :  /*********************************************************************/
68 :  unsigned char dipsw_get( void )
69 :  {
70 :      unsigned char sw;
71 :
72 :      sw  = ~P6DR;                        /* ディップスイッチ読み込み */
73 :      sw &= 0x0f;
74 :
75 :      return  sw;
76 :  }
77 :
78 :  /*********************************************************************/
79 :  /* end of file                                                       */
80 :  /*********************************************************************/
```

14.5 モータドライブ基板Vol.3 について

14.5.1 概要

下記に、モータドライブ基板 Vol.1～3 の仕様をまとめます。マイコンカーキット Ver.4 で使用されているモータドライブ基板は Vol.3 です。

名称	モータドライブ 基板(Vol.1)	モータドライブ 基板(Vol.2)	**モータドライブ 基板(Vol.3)**
略称	ドライブ基板1	ドライブ基板2	ドライブ基板3
販売開始 時期	1998 年ごろ	2002 年 4 月	2005 年 4 月
モータの 動作	正転、逆転、ブレーキ	正転、フリー、ブレーキ	正転、逆転、ブレーキ
CPU ボード との接続	ポート B	ポート A	ポート B
PWM	リセット同期 PWM モード使用	1 チャネルごとの PWM 使用	リセット同期 PWM モード使用
周期	モータ:16ms　サーボ:16ms 個別設定不可	モータ:1ms　サーボ:16ms 個別に設定可能	モータ:16ms　サーボ:16ms 個別設定不可
使用する FET	4AM12×2 個	4AM12×1 個	2SJ タイプ×4 個＋ 2SK タイプ×4 個
制御系 電圧	DC5.0V±10%	DC5.0V±10%	DC5.0V±10%
駆動系 電圧	5V	5～15V	5～15V
駆動系電圧 6V 以上のときの対 応	回路的にできない	制御系の 5V 生成用回路の領域あり(三端子レギュレータ)、サーボ用の 6V 生成用回路の領域はなし(外付け)	制御系の 5V 生成用回路の領域、サーボ用の 6V 生成用回路の領域共にあり(三端子レギュレータ)
標準 プログラム	tmc4.c	kit2.c　または　kit04.c	kit05.c または kit06.c または kit07.c
寸法	最大 W76×D49×H15mm (実測)	最大 W80×D50×H15mm (実測)	最大 W80×D65×H20mm (実測)
その他	販売終了	販売終了	販売中 (2007.05 現在)

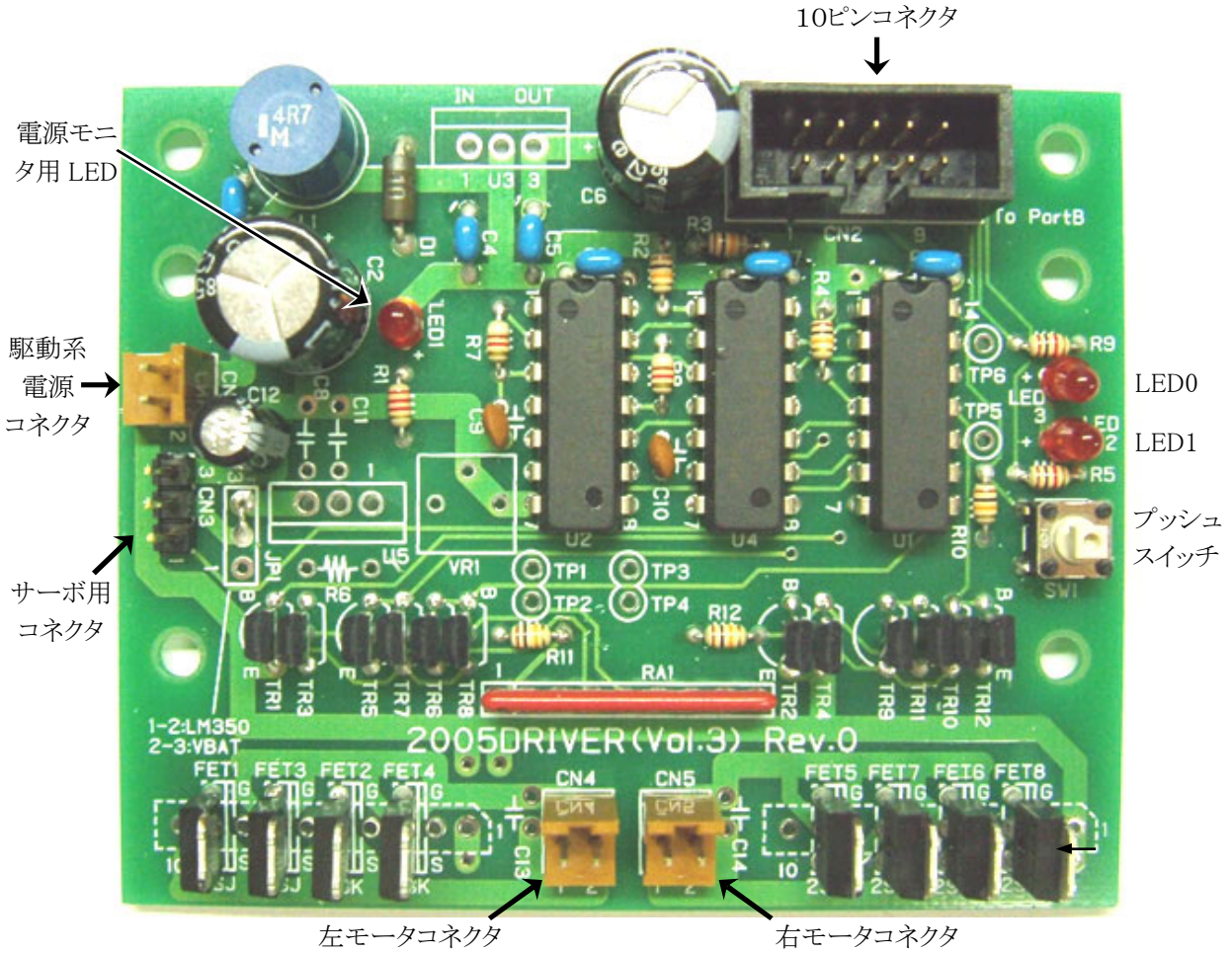
ドライブ基板1ではリセット同期 PWM モードというモードでモータ、サーボを制御していました。これはプログラムが比較的簡単なのですが、右モータ、左モータ、サーボに加える PWM 周期を同じにしかできないという制限がありました。サーボには 16[ms]の周期を加えることと規格で決まっており、モータに加える PWM も 16[ms]となります。しかし、モータ制御に周期 16[ms]では間隔が長すぎ、モータがガクガクした動きとなってしまいました。

ドライブ基板 2 では、1 チャネルごとの PWM を使用して、右モータ、左モータの周期を 1[ms]、サーボの周期を 16[ms]と、それぞれ独立して周期を設定できるようにしました。モータの制御は滑らかになりましたが、プログラムが複雑になってしまい非常に理解しづらくなってしまいました。

そこで、ドライブ基板 3 では、リセット同期 PWM モードに戻しました。ドライブ基板 1 の説明のとおり、モータがガクガクしてしまいます。しかし、サーボの周期は規格では 16[ms]ですが、実験で周期を短くしても動作することが分かりました。そこで、どうしてもガクガクが気になる場合は、サーボが動作する範囲内で周期を短くして対応します。

14.5.2 部品実装図

ドライブ基板3は、部品を実装すると下記写真のようになります。



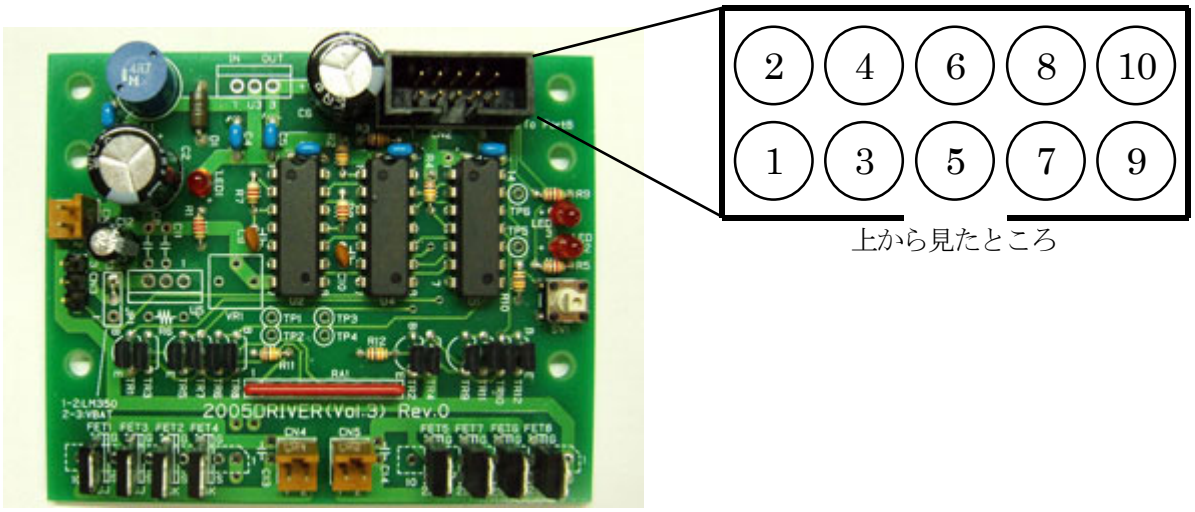
10 ピンコネクタ	フラットケーブルで CPU ボードと接続します。ポート B(J3)のコネクタに接続します。
駆動系 電源コネクタ	モータとサーボに供給する電源です。74HC08、74HC14、74HC32 などの制御系部品は、10 ピンコネクタから供給される 5V で動作します。標準キットでは入力電圧 6V までに対応していますが、それ以上の電圧にするときは、**サーボに加える電圧を 6V 一定にする必要があります**。LM350 追加セットの部品を追加すると、サーボ電圧を一定にすることができます。
右モータコネクタ	右モータと接続します。
左モータコネクタ	左モータと接続します。
サーボ用コネクタ	サーボと接続します。3 ピンで信号の順番が、「1:サーボ信号、2:＋電源、3:GND」となっています。この順番でないメーカのサーボは、サーボ側のピンを入れ替える必要があります。
電源モニタ用 LED	電源コネクタに電圧が供給されていると光ります。
LED0	10 ピンコネクタに接続した CPU ボードのポート B の bit6 と接続されています。この bit を出力用に設定して、LED0 を点灯／消灯させます。

LED1	10 ピンコネクタに接続した CPU ボードのポート B の bit7 と接続されています。この bit を出力用に設定して、LED1 を点灯／消灯させます。
プッシュスイッチ	10 ピンコネクタに接続した CPU ボードのポート B の bit0 と接続されています。この bit を入力用に設定して、状態を読み込むことによりスイッチが押されているかどうかチェックします。

14.6 コネクタ

14.6.1 10 ピンコネクタ

フラットケーブルで CPU ボードと接続します。



上から見たところ

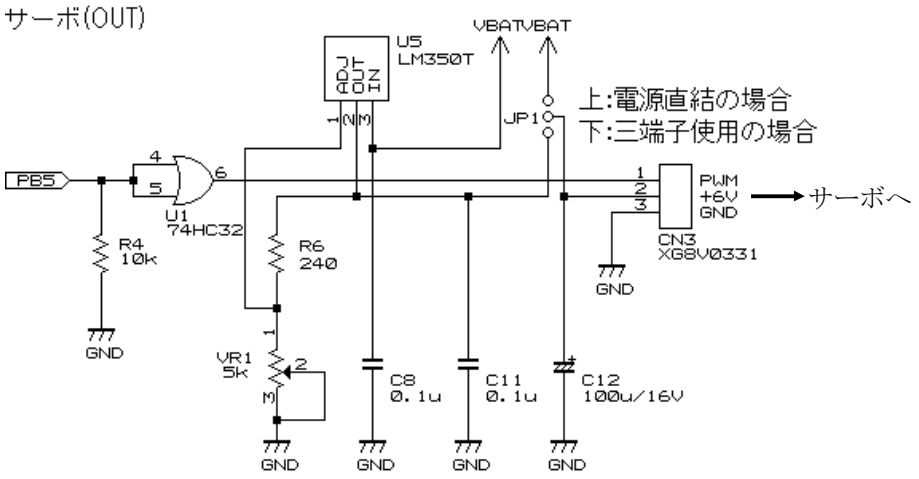
ピン番	信号、方向※	詳細	“0”	“1”	備考
1	―	+5V			
2	基板←PB7	LED1	点灯	消灯	
3	基板←PB6	LED0	点灯	消灯	
4	**基板←PB5**	**サーボ信号**	**PWM 信号**		**PWM 信号** **ITU4_BRB でデューティ比設定**
5	基板←PB4	右モータ PWM	停止	動作	PWM 信号 ITU4_BRA でデューティ比設定
6	基板←PB3	右モータ回転方向	正転	逆転	
7	基板←PB2	左モータ回転方向	正転	逆転	
8	基板←PB1	左モータ PWM	停止	動作	PWM 信号 ITU3_BRB でデューティ比設定
9	基板→PB0	プッシュスイッチ	押された	押されてない	
10	―	GND			

※「基板→PB○」 は、ドライブ基板からの出力信号をマイコンのポートで読み込みます(ポートは入力)。
「基板←PB○」 は、マイコンからの出力信号をドライブ基板が入力し動作します。

表のとおり、PB5 にサーボが接続されています。プログラムはリセット同期 PWM モードを使用しているので、PB5 のデューティ比を変えるには、ITU4_BRB の値を変えます。

14.6.2 サーボの制御回路

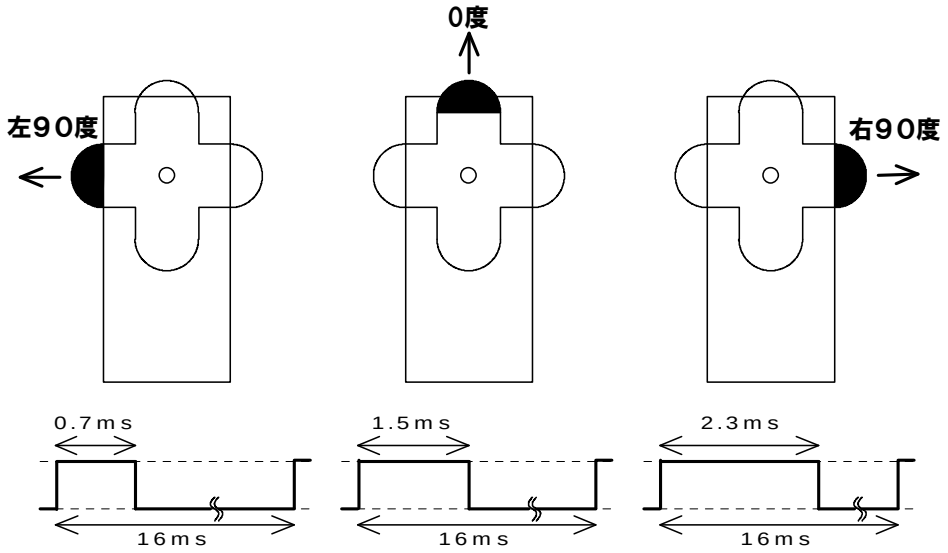
モータドライブ基板 Vol.3 のサーボ制御回路は下記のようになっています。



1. ポートBのビット5から PWM 信号を出力します。プログラムは、ITU4_BRB の値を変えると ON 幅が変わります。
2. PB5 とサーボの1ピンの間には、バッファとして OR 回路(74HC32)を入れています。例えば、1 ピンに間違って電源を接続したりノイズが混入して端子が壊れてしまった場合、PB5 とサーボの 1 ピンが直結ならマイコンのポートを壊します。これは致命的です。74HC32 なら安価で 14 ピンなので簡単に交換できます。
3. サーボの 2 ピンは、電源端子です。モータ用電源が電池 4 本以下の場合、JP1 の上側をショートして電源と直結します。それ以上の電圧の場合、サーボの定格を超えますので LM350 という 3A 電流を流せる三端子レギュレータにて電圧を 6V 一定にします。JP1 は下側をショートさせます。

14.7 サーボの制御

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 ON のパルス幅の関係は、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になります。



・周期は 16[ms]
・中心は 1.5[ms]の ON パルス、±0.8[ms]で±90 度のサーボ角度変化

リセット同期式 PWM モードで下記のような PWM 信号を出力して、サーボを制御します。
・周期 16[ms]
・ON 幅 0.7～2.3[ms]

14.8 プログラムの解説

14.8.1 ポートBの入出力設定

ポート B にモータドライブ基板を接続します。入出力方向は、下記のとおりです。

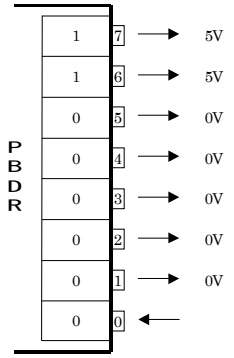
ピン番	信号、方向	詳細	“0”	“1”	入出力	PBDDR の値
1	－	+5V				
2	基板←PB7	LED1	点灯	消灯	出力	1
3	基板←PB6	LED0	点灯	消灯	出力	1
4	基板←PB5	サーボ信号	PWM 信号		出力	1
5	基板←PB4	右モータ PWM	停止	動作	出力	1
6	基板←PB3	右モータ回転方向	正転	逆転	出力	1
7	基板←PB2	左モータ回転方向	正転	逆転	出力	1
8	基板←PB1	左モータ PWM	停止	動作	出力	1
9	基板→PB0	プッシュスイッチ	押された	押されてない	入力	0
10	－	GND				

入力は”0”、出力は”1”を設定するので、PBDDR の値は下記のようになります。
PBDDR ＝ 1111 1110 ＝ 0xfe

14.8.2 ポートBに出力する値の初期値

ポート B に最初に出力する値は、モータドライブ基板に接続されている機器をどうするかによって決めます。下表の太線のような動作にします。

ピン番	信号、方向	詳細	"0"	"1"	PBDR の値
1	―	+5V			
2	基板←PB7	LED1	点灯	消灯	**1**
3	基板←PB6	LED0	点灯	消灯	**1**
4	基板←PB5	サーボ信号	PWM 信号		**0**
5	基板←PB4	右モータ PWM	停止	動作	**0**
6	基板←PB3	右モータ回転方向	正転	逆転	**0**
7	基板←PB2	左モータ回転方向	正転	逆転	**0**
8	基板←PB1	左モータ PWM	停止	動作	**0**
9	基板→PB0	プッシュスイッチ	押された	押されてない	**どちらでも良い**
10	―	GND			



最初、LED は消灯させたいので、"1"を出力します。モータは停止させたいので"0"、サーボはどちらでも良いのですが、とりあえず"0"としておきます。PB0 は入力用なので、何を書き込んでも変わりません。ただ"1"にすると、何か意味があるのかという疑問がわいてくるので、"0"を書き込んでおきます。

PBDR ＝ 1100 0000 ＝ 0xc0

となります。

14.8.3 PBDRとPBDDRの設定する順番

プログラムでは最初に PBDR へ 0xc0 をセットして、次に PBDDR へ 0xfeをセットしています。

```
49 :      PBDR  = 0xc0;
50 :      PBDDR = 0xfe;                    /* モータドライブ基板 Vol.3  */
```

なぜ入出力設定の前にデータをセットするのでしょうか？これは、リセットした直後のレジスタの値がどのようになっているかに関係してきます。PBDDR の初期値は「0x00」で、端子は入力になっています。PBDR の初期値は「0x00」です。そのため、先に PBDDR でポートを出力にセットすると、PBDR の値「0x00」が出力されてしまいます。LED は"0"で点灯するので一瞬 LED が点灯します。次の PBDR への書き込みですぐに消灯するので問題ないと言えば無いのですが、仮に他のデバイスに接続されていた場合、一瞬有効になったことにより誤動作するかもしれません。このような小さなミスを無くすために、先に PBDR を設定して LED を消灯状態にしておいてから、PBDDR の設定を行っています。

14.8.4 リセット同期PWMモードの設定

リセット同期 PWM モードを設定します。8 行です。

```
54 :      ITU3_TCR = 0x23;                  /* カウンタ、クリアの設定   */
55 :      ITU_FCR  = 0x3e;                  /* リセット同期 PWM モード    */
56 :      ITU3_GRA = 49151;                 /* 周期の設定               */
57 :      ITU3_GRB = ITU3_BRB = 0;          /* 左モータの PWM 設定        */
58 :      ITU4_GRA = ITU4_BRA = 0;          /* 右モータの PWM 設定        */
59 :      ITU4_GRB = ITU4_BRB = 5000;       /* サーボの PWM 設定          */
60 :      ITU_TOER = 0x38;                  /* 出力端子の設定           */
61 :      ITU_STR  = 0x08;                  /* タイマスタート           */
```

各レジスタの設定について、簡単に説明していきます。詳細は、pwm3.c を参照してください。

```
54 :      ITU3_TCR = 0x23;                  /* カウンタ、クリアの設定   */
```

・ITU3_CNT を0にするタイミング　→ ITU3_CNT が(TU3_GRA+1)になったとき
・ITU3_CNT が＋1する時間　→ φ/8 でカウント
を設定します。
RY3048F-ONE ボードのクリスタルは、24.576[MHz]ですので、＋1する時間は、
1／(φ／8)＝1／(24.576×10^6／8)＝325.52[ns]
となります。

```
55 :      ITU_FCR  = 0x3e;                  /* リセット同期 PWM モード    */
```

ビット:	7	6	5	4	3	2	1	0
ITU_FCR:	−	−	CMD1	CMD0	BFB4	BFA4	BFB3	BFA3
設定値:	0	0	1	1	1	1	1	0
16 進数:	3				e			

ビット 5, 4 で、リセット同期 PWM モードを使用するよう設定します。また、ITU4_BRB、ITU4_BRA、ITU3_BRB のバッファを使用します。

```
56 :      ITU3_GRA = 49151;                 /* 周期の設定               */
```

PWM の周期の設定です。ITU3_CNT は 325.52[ns]で＋1するので、16[ms]にするためには、
(16×10^{-3})÷(325.52×10^{-9})＝49,152
となります。設定は、1小さい値を設定します。

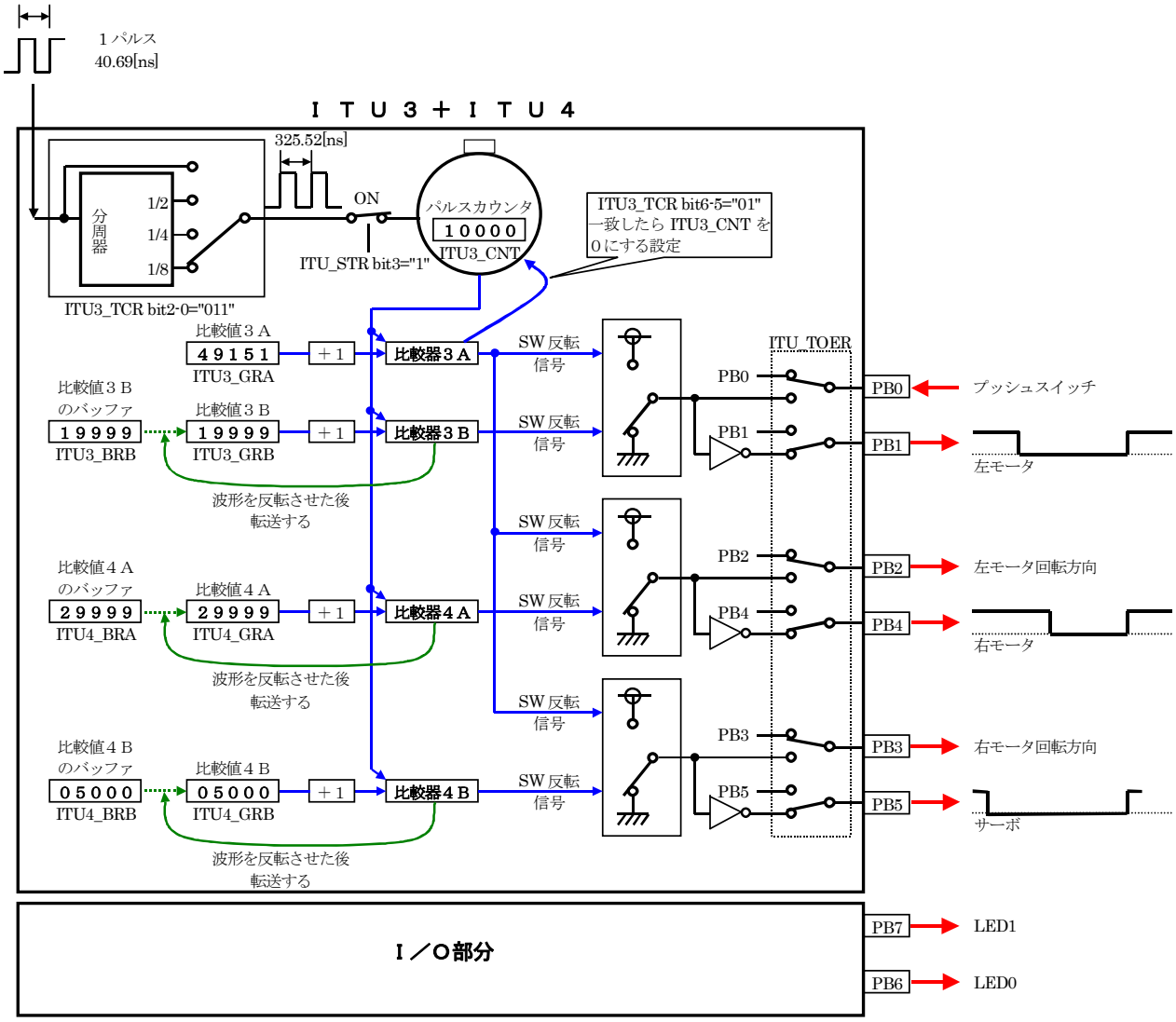
```
57 :      ITU3_GRB = ITU3_BRB = 0;          /* 左モータの PWM 設定        */
58 :      ITU4_GRA = ITU4_BRA = 0;          /* 右モータの PWM 設定        */
59 :      ITU4_GRB = ITU4_BRB = 5000;       /* サーボの PWM 設定          */
```

それぞれのデューティ比を設定します。
サーボをまっすぐに向かせるには、1.5[ms]の時間"1"にします。そのため、設定値は、
(1.5×10^{-3})÷(325.52×10^{-9})＝4608

となります。しかし、サーボのセンタは、サーボ自体の誤差、サーボホーンに挿すギザギザのかみ合わせ方などの影響ですべてのサーボで違う値になります。例えるなら、人間の指紋のようなものでしょうか。そのため、ここではきりのいい 5,000 という値にしています。この値は、直進時にまっすぐに向くように調整、変更します。

```
60 :        ITU_TOER = 0x38;                    /* 出力端子の設定          */
```

どの端子を PWM 出力にするかの設定です。ポートBの接続を再確認しましょう。次ページの図のようになっています。



PB0～PB6 が PWM 出力端子になっていますが、PB0、PB2、PB3 は通常の I/O ポートとして使用します。そのため、PWM 出力を OFF にします。

TOER	7	6	5	4	3	2	1	0
"0"のとき	—	—	PB5 は通常の入出力ポート	PB4 は通常の入出力ポート	PB1 は通常の入出力ポート	PB3 は通常の入出力ポート	PB2 は通常の入出力ポート	PB0 は通常の入出力ポート
"1"のとき	—	—	PB5 は PWM 出力	PB4 は PWM 出力	PB1 は PWM 出力	PB3 は PWM 出力	PB2 は PWM 出力	PB0 は PWM 出力
今回の設定	0	0	1	1	1	0	0	0

16 進数に変換して、ITU_TOER には 0x38 を設定します。

```
61 :      ITU_STR  = 0x08;                    /* タイマスタート          */
```

最後に、ITU3_CNT のカウントをスタートさせて、リセット同期 PWM モードの設定が完了です。

14.8.5 mian関数

```
25 :  void main( void )
26 :  {
27 :      init();                             /* 初期化                  */
28 :
29 :      while( 1 ) {
30 :          ITU4_BRB = 5000 + dipsw_get() * 26;
31 :      }
32 :  }
```

サーボは PB5 へ接続されています。PB5 のデューティ比を変えるレジスタは、ITU4_GRB です。今回はバッファを使用しますので、プログラムで変更するのは ITU4_BRB となります。

計算式は、

ITU4_BRB = 5000 + dipsw_get() * 26;

①　②　③

となっています。

①サーボのセンタ値

5000は先ほど計算したとおり、センタ値です。

②CPUボードのディップスイッチの値

0～15の値です。

③1度分の数値

ディップスイッチの値に応じて0～15まで変化しますが、これでは変化量が小さすぎて、サーボがほとんど動きません。そのため、ディップスイッチの数値1当たり、1度動作するようにしています。

サーボが、右に 90 度向くときは 2.3ms のパルスなので、

$(2.3\times10^{-3})\div(325.52\times10^{-9})=7,065$

サーボが、左に 90 度向くときは 0.7ms のパルスなので、

$(0.7\times10^{-3})\div(325.52\times10^{-9})=2,150$

よって、±90度の移動量は、

(右 90 度)－(左 90 度)＝7,065-2,150＝4,915

となります。これを180で割れば1度当たりの移動量が分かります。

4,915÷180≒27

となります。正確に計算すると 27 がサーボ 1 度分の値です。ただし、前のプログラム kit04.c では 26 としていたので、過去との互換を考えて、26 とします。

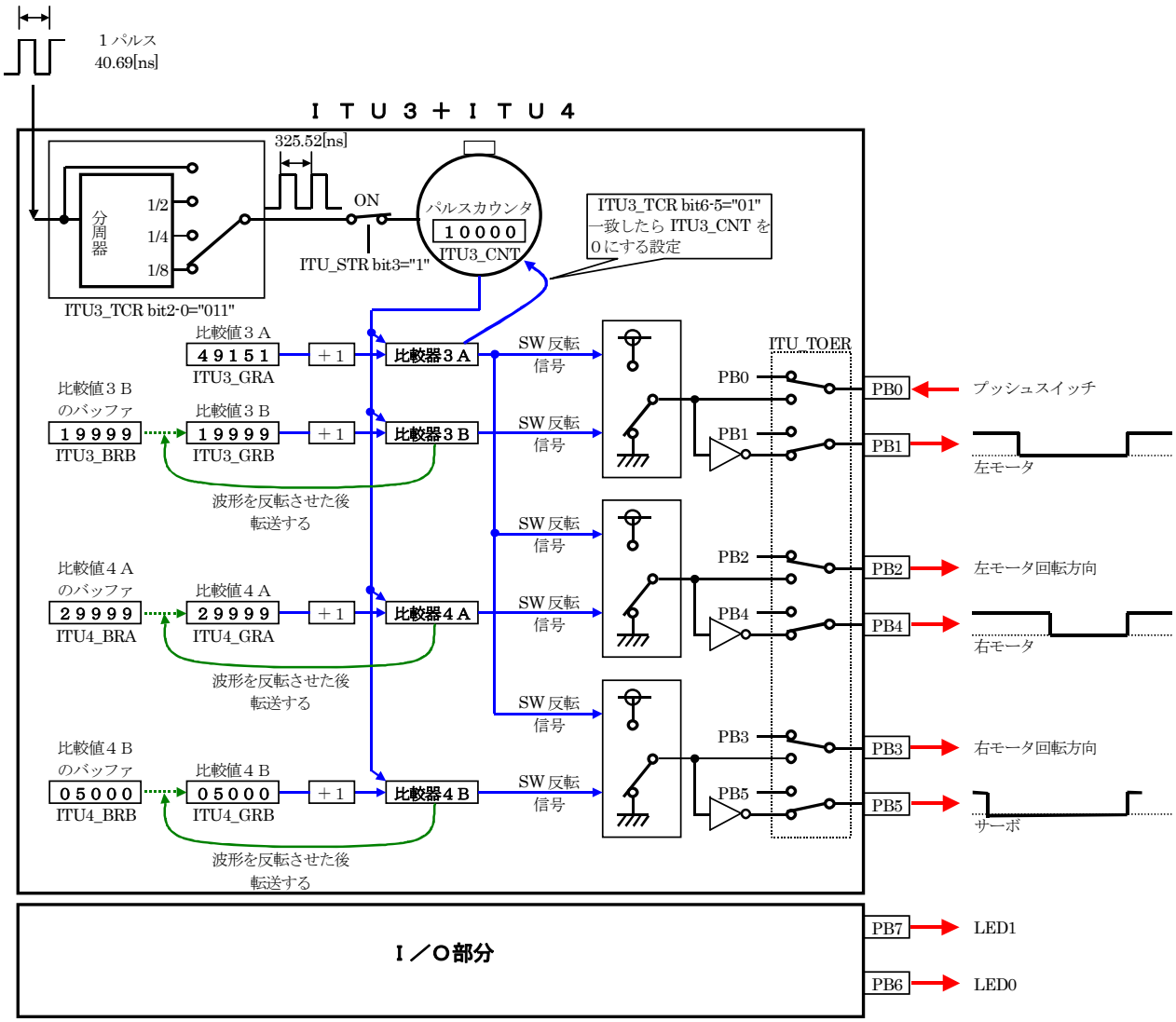
これで、ディップスイッチを動かすと0度～15度までサーボが変化します。センタがずれている場合は、5000の数値を変えてみてください。

このように、サーボに加えるパルス幅を変えるだけでサーボの制御ができます。

このプログラムでは片方にしかサーボが動きません。下記のように dipsw_get 関数の前の「＋」を「－」に変えて、逆にも動くことを確かめてみてください。

```
29 :      while( 1 ) {
30 :          ITU4_BRB = 5000 - dipsw_get() * 26;
31 :      }
```

14.9 まとめ



設定するレジスタ	詳細
ITU3_TCR	ITU3_CNT の値が＋1する時間、クリア要因の設定をします。 0x20…周期が 2.666ms 以下の場合(＋1する時間は 40.69[ns]ごと) 0x21…周期が 5.333ms 以下の場合(＋1する時間は 81.38[ns]ごと) 0x22…周期が 10.67ms 以下の場合(＋1する時間は 162.76[ns]ごと) 0x23…周期が 21.33ms 以下の場合(＋1する時間は 325.52[ns]ごと) 今回は、周期 16ms なので **0x23** を設定します。
ITU_FCR	リセット同期PWMモードの設定と、バッファレジスタを使用するかの設定します。 **0x3e** を設定します。PB0～PB5 の端子からは PWM 波形がされます。出力したくない場合は、ITU_TOER を設定することにより通常の I/O ポートとして使用できます。
ITU3_GRA	周期を設定します。計算は、「設定したい周期÷(ITU3_CNT の値が＋1する時間)－1」です。周期 16ms、＋1する時間 325.52ns なら $(16\times10^{-3})\div(325.52\times10^{-9})-1=49151$
ITU3_BRB (ITU3_GRB)	PB0 の OFF 幅、または PB1 の ON 幅を設定します。PB0 端子は通常の I/O として使用、PB1 端子には左モータが接続されています。 計算は、「設定したい幅÷(ITU3_CNT の値が＋1する時間)－1」です。 ITU3_GRB は、最初だけ ITU3_BRB と同じ値を設定します。
ITU4_BRA (ITU4_GRA)	PB2 の OFF 幅、または PB4 の ON 幅を設定します。PB2 端子は通常の I/O として使用、PB4 端子には右モータが接続されています。 計算は、「設定したい幅÷(ITU3_CNT の値が＋1する時間)－1」です。 ITU4_GRA は、最初だけ ITU4_BRA と同じ値を設定します。
ITU4_BRB (ITU4_GRB)	PB3 の OFF 幅、または PB5 の ON 幅を設定します。PB3 端子は通常の I/O として使用、PB5 端子にはサーボが接続されています。 計算は、「設定したい幅÷(ITU3_CNT の値が＋1する時間)－1」です。 ITU4_GRB は、最初だけ ITU4_BRB と同じ値を設定します。 例えば、サーボのセンタ値である 1.5[ms]の間、ON としたい場合、 $(1.5\times10^{-3})\div(325.52\times10^{-9})-1=4607$ となります。
ITU_TOER	PWM 波形を出力するかしないか選択します。 "1":PWM波形出力 "0":通常の I/O ポートとして使用 bit 7 6 5 4 3 2 1 0 0固定 0固定 PB5 PB4 PB1 PB3 PB2 PB0 今回は、PB5,4,1 を PWM として使用、他は I/O ポートなので PB5,4,1 の部分を"1"に、PB4,3,0 の部分を"0"にします。 bit 7 6 5 4 3 2 1 0 値 0 0 1 1 1 0 0 0 0011 1000→**0x38** を設定します。
ITU_STR	それぞれの ITU のチャネルで ITU_CNT をカウント動作させるかどうか選択します。要は、ITU を使うか使わないかの設定です。 "1":使用する "0":使用しない bit 7 6 5 4 3 2 1 0 0固定 0固定 0固定 ITU4 ITU3 ITU2 ITU1 ITU0 リセット同期 PWM モードでは ITU3 と 4 を使用します。しかしカウンタは ITU3 のみしか使用しません。そのため、ITU3 のみのカウンタを動作させます。 bit 7 6 5 4 3 2 1 0 値 0 0 0 0 1 0 0 0 0000 1000 →**0x08** を設定します。

15. プロジェクト「motor」 モータの制御

15.1 概要

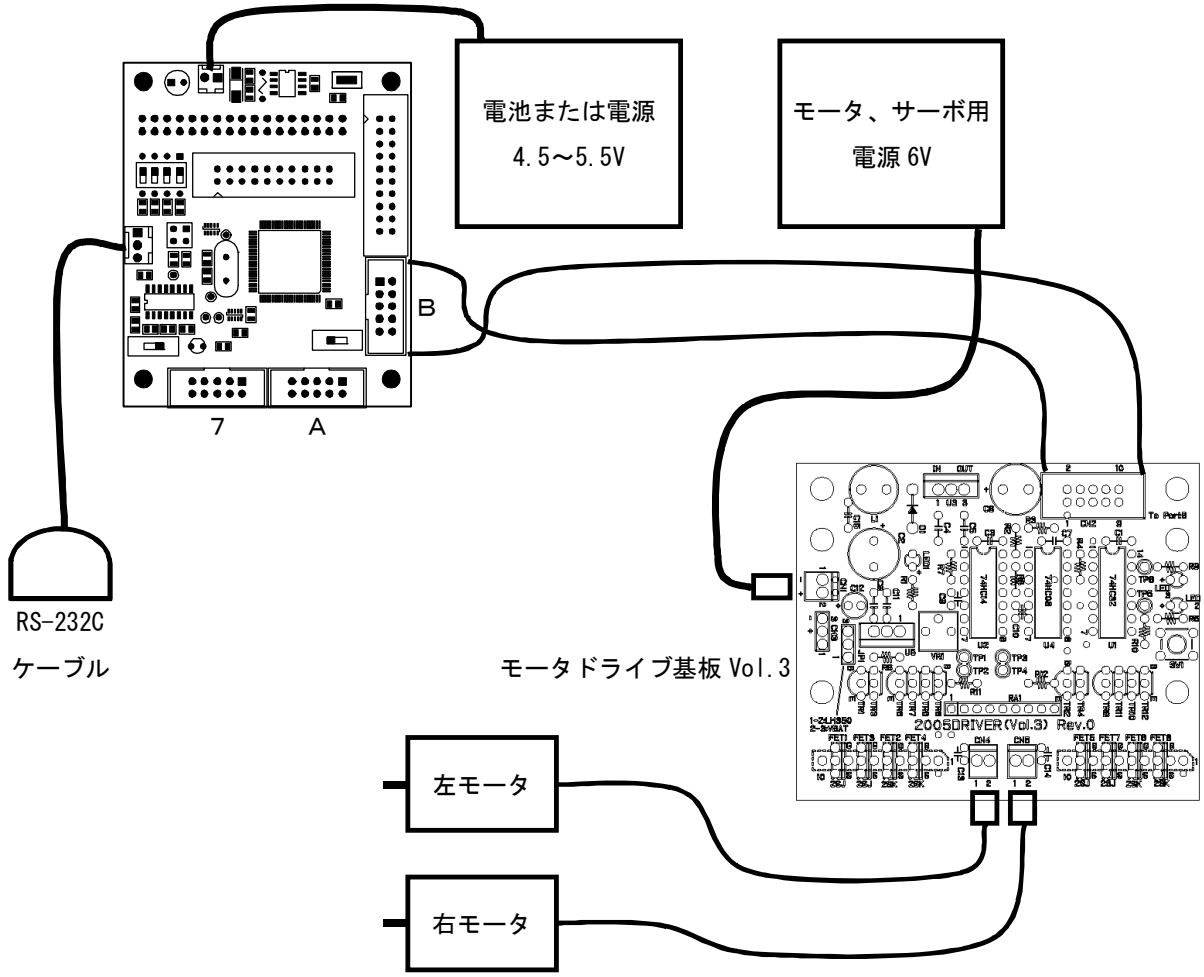
モータを制御します。マイコンのポートは、下記を使用します。

・ポート B・・・モータドライブ基板 Vol.3 を接続

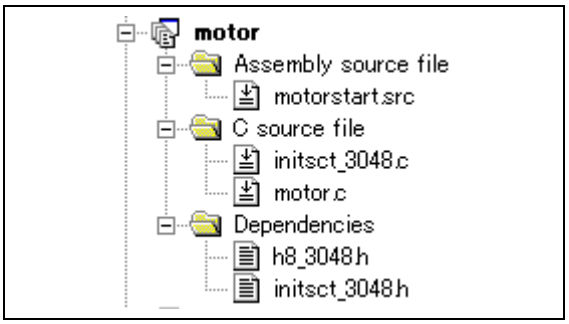
15.2 接続

・CPU ボードのポート B と、モータドライブ基板 Vol.3 をフラットケーブルで接続します。

・モータドライブ基板には、右モータ、左モータとモータ・サーボ用電源を接続します。



15.3 プロジェクトの構成



	ファイル名	内容
1	motorstart.src	アセンブリ言語で記述されたアセンブリソースファイルです。このファイルの構造は下記のようになっています。 ベクタアドレス ＋ スタートアップルーチン
2	initsct_3048.c	初期値のないグローバル変数(セクション B 領域)、初期値のあるグローバル変数(セクション R 領域)の初期化用です。
3	motor.c	実際に制御するプログラムが書かれています。H8/3048F-ONE の内蔵周辺機能の初期化も行います。
4	h8_3048.h	H8/3048F-ONE の内蔵周辺機能の I/O レジスタを定義したファイルです。
5	initsct_3048.h	initsct_3048.c のヘッダファイルです。

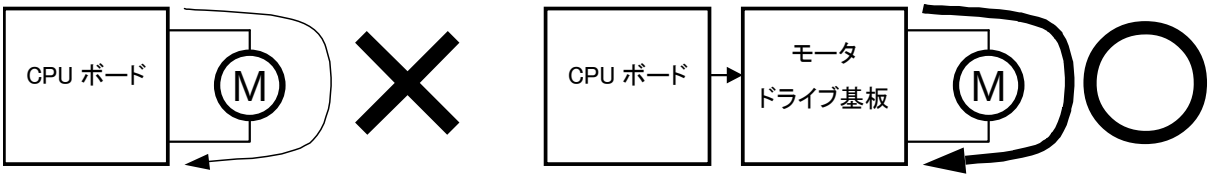
15.4 プログラム「motor.c」

```
 1 :  /*********************************************************************/
 2 :  /* モータのテスト「motor.c」                                         */
 3 :  /* 入力、出力：ポートＢにモータドライブ基板(Vol.3)を接続             */
 4 :  /*                          2005.04 ジャパンマイコンカーラリー実行委員会  */
 5 :  /*********************************************************************/
 6 :  /*====================================*/
 7 :  /* インクルード                       */
 8 :  /*====================================*/
 9 :  #include    <machine.h>
10 :  #include    "h8_3048.h"
11 :
12 :  /*====================================*/
13 :  /* シンボル定義                       */
14 :  /*====================================*/
15 :
16 :  /*====================================*/
17 :  /* プロトタイプ宣言                   */
18 :  /*====================================*/
19 :  void init( void );
20 :  unsigned char dipsw_get( void );
21 :
22 :  /*********************************************************************/
23 :  /* メインプログラム                                                  */
24 :  /*********************************************************************/
25 :  void main( void )
26 :  {
27 :      init();                             /* 初期化                    */
28 :
29 :      while( 1 ) {
30 :          ITU3_BRB = 49150 * dipsw_get() / 15;    /* 左モータ          */
31 :      }
32 :  }
33 :
34 :  /*********************************************************************/
35 :  /* H8/3048F-ONE 内蔵周辺機能の初期化                                 */
36 :  /*********************************************************************/
37 :  void init( void )
38 :  {
39 :      /* ポートの入出力設定 */
40 :      P1DDR = 0xff;
41 :      P2DDR = 0xff;
42 :      P3DDR = 0xff;
43 :      P4DDR = 0xff;
44 :      P5DDR = 0xff;
45 :      P6DDR = 0xf0;                       /* CPU基板上のDIP SW         */
46 :      P8DDR = 0xff;
47 :      P9DDR = 0xf7;
48 :      PADDR = 0xff;
49 :      PBDR  = 0xc0;
50 :      PBDDR = 0xfe;                       /* モータドライブ基板Vol.3   */
51 :      /* ポート７は、入力専用なので入出力設定はありません    */
52 :
53 :      /* ITU3,4 リセット同期PWMモード 左右モータ、サーボ用 */
54 :      ITU3_TCR = 0x23;                    /* カウンタ、クリアの設定    */
55 :      ITU_FCR  = 0x3e;                    /* リセット同期PWMモード     */
56 :      ITU3_GRA = 49151;                   /* 周期の設定                */
57 :      ITU3_GRB = ITU3_BRB = 0;            /* 左モータのPWM設定         */
58 :      ITU4_GRA = ITU4_BRA = 0;            /* 右モータのPWM設定         */
59 :      ITU4_GRB = ITU4_BRB = 5000;         /* サーボのPWM設定           */
60 :      ITU_TOER = 0x38;                    /* 出力端子の設定            */
61 :      ITU_STR  = 0x08;                    /* タイマスタート            */
62 :  }
63 :
64 :  /*********************************************************************/
65 :  /* ディップスイッチ値読み込み                                        */
66 :  /* 戻り値 スイッチ値 0～15                                           */
67 :  /*********************************************************************/
68 :  unsigned char dipsw_get( void )
69 :  {
70 :      unsigned char sw;
71 :
72 :      sw  = ~P6DR;                        /* ディップスイッチ読み込み */
73 :      sw &= 0x0f;
74 :
75 :      return  sw;
76 :  }
77 :
78 :  /*********************************************************************/
79 :  /* end of file                                                       */
80 :  /*********************************************************************/
```

15.5 モータの制御について

15.5.1 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。



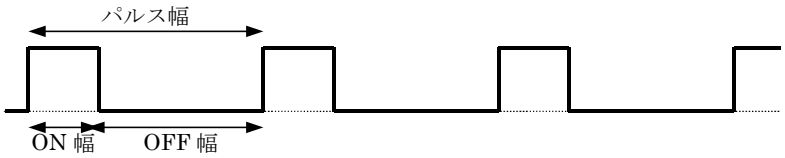
信号が弱いため、モータは回らない　　大電流に変換、モータ駆動！

15.5.2 スピード制御の原理

モータを回したければ、電圧を加えれば回ります。止めたければ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。

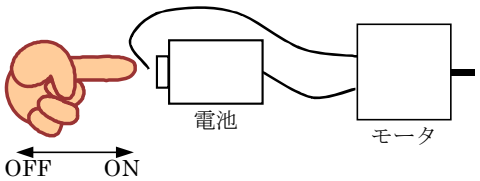
ボリュームを使えば電圧を落とすことができます。しかし、モータへは大電流が流れるため、非常に大きな抵抗が必要です。また、モータに加えなかった分は、抵抗の熱となってしまいます。

そこで、スイッチで ON、OFF 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」となります。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＝ON 幅／パルス幅(ON幅＋OFF幅)
です。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。

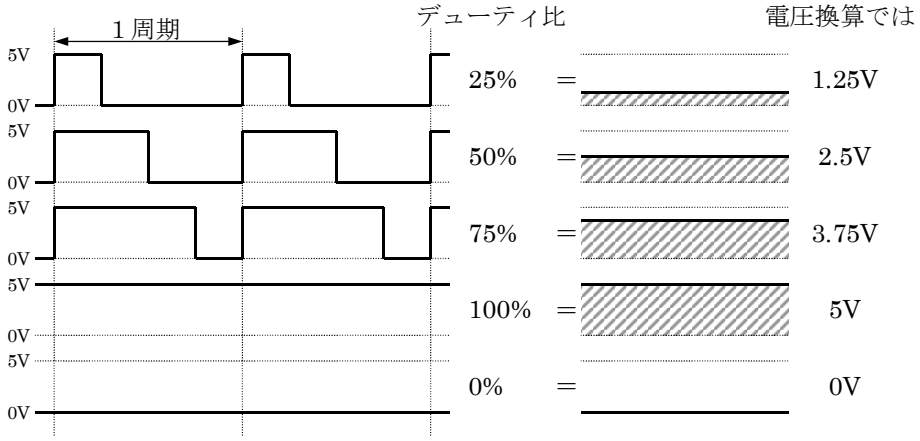
「PWM」と聞くと、何か難しく感じてしまいますが、下記のように手でモータと電池の線を「繋ぐ」、「離す」の繰り返し、それも PWM と言えます。繋いでいる時間が長いとモータは速く回ります。離している時間が長いとモータは少ししか回りません。人なら「繋ぐ」、「離す」動作をコンマ数秒でしか行えませんがマイコンなら数ミリ秒で行えます。



下図のように、0V と 5V を出力するような波形で考えてみます。1周期に対して ON の時間が長ければ長いほど平均化した値は大きくなります。すべて 5V にすればもちろん平均化しても 5V、これが最大の電圧です。ON の時間を半分の 50%にするとどうでしょうか。平均化すると 5V×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。CPU 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。

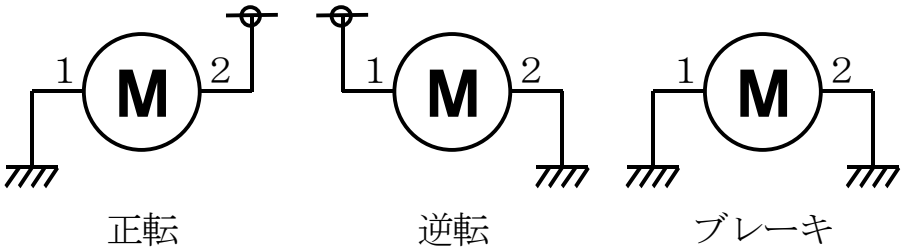


なぜ電圧制御ではなくパルス幅制御でモータのスピードを制御するのでしょうか。CPU は"0"か"1"かのデジタル値の取り扱いは大変得意ですが、何 V というアナログ的な値は不得意です。そのため、″0″と″1″の幅を変えて、あたかも電圧制御しているように振る舞います。これが PWM 制御です。

15.5.3 正転、逆転、ブレーキの原理

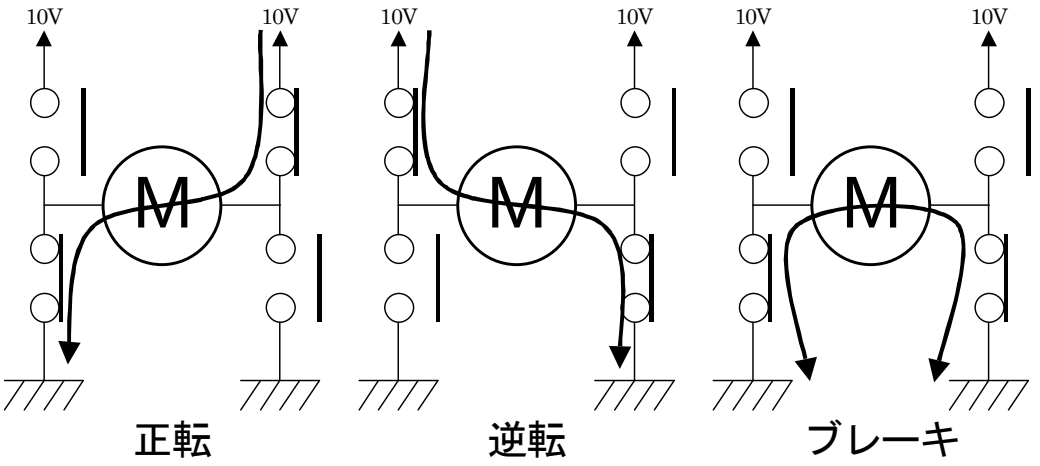
モータドライブ基板(Vol.3)では、モータを「正転、逆転、ブレーキ」制御することができます。これは、モータの端子に加える電圧を下表のように変えることにより、制御しています。

動作	端子1	端子2
正転	GND接続	＋接続
逆転	＋接続	GND接続
ブレーキ	GND接続	GND接続



15.5.4 Hブリッジ回路

実際のモータ制御は、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。

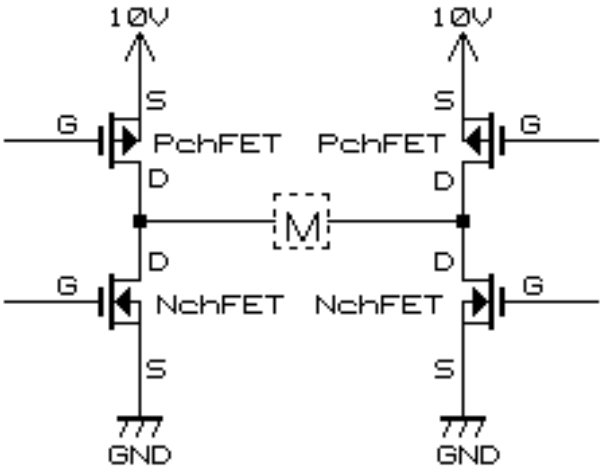


15.5.5 Hブリッジ回路のスイッチをFETにする

この、スイッチに変わる素子を FET で行います。電源のプラス側に P チャネル FET(2SJ タイプ)、マイナス側に N チャネル FET(2SK タイプ)を使用します。

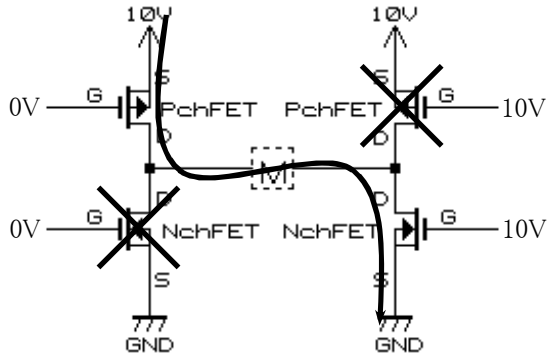
P チャネル FET は、V_G(ゲート電圧)＜V_S(ソース電圧)のとき、D-S(ドレイン－ソース)間に電流が流れます。

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

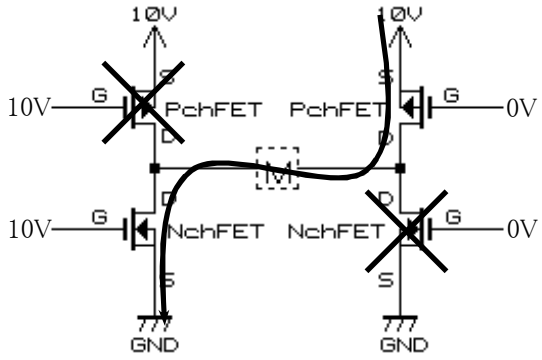


これら 4 つの FET のゲートに加える電圧を変えることにより、正転、逆転、ブレーキの動作を行います。

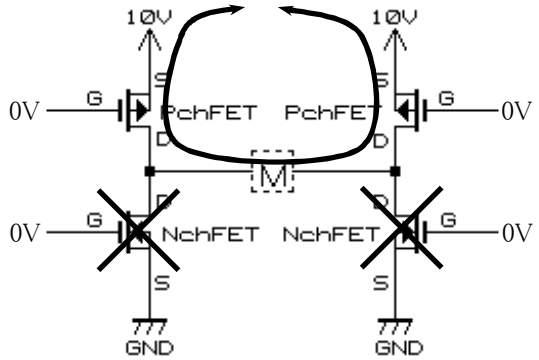
逆転



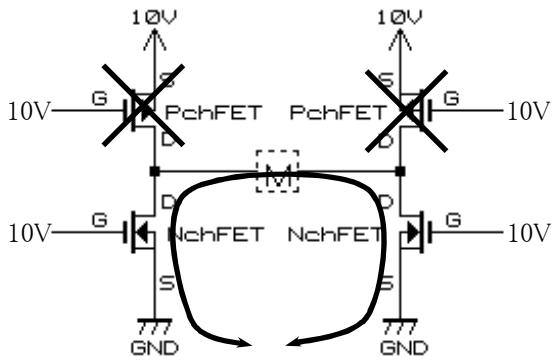
正転



ブレーキ1

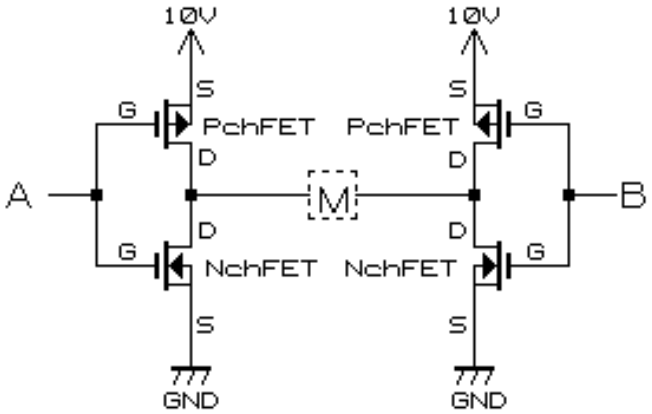


ブレーキ2



注意点は、絶対に左側もしくは右側の FET を同時に ON させてはいけないと言うことです。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
0V	0V	ブレーキ

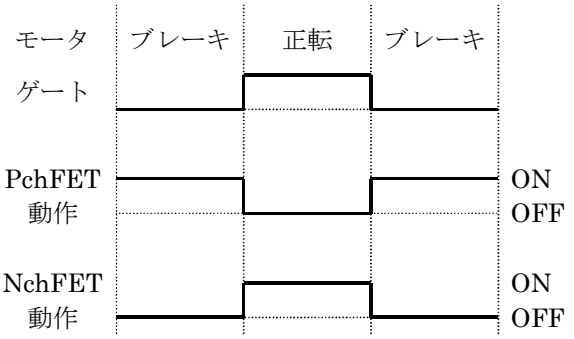
※G(ゲート)端子にはモータ用の電源電圧が 10V であったとすれば、その電圧がそのまま加えられたり 0V が加えられたりします。″0″、″1″の制御信号とは異なるので注意しましょう。

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。

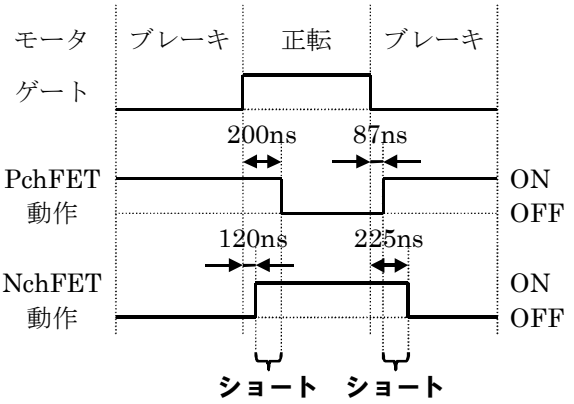
FET のゲートから信号を入力し、ドレイン・ソース間が ON／OFF するとき、事項の左図「理想的な波形」のように、P チャネル FET と N チャネル FET がすぐに反応してブレーキと正転がスムーズに切り替わりるように思えま

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。

参考までに FET 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

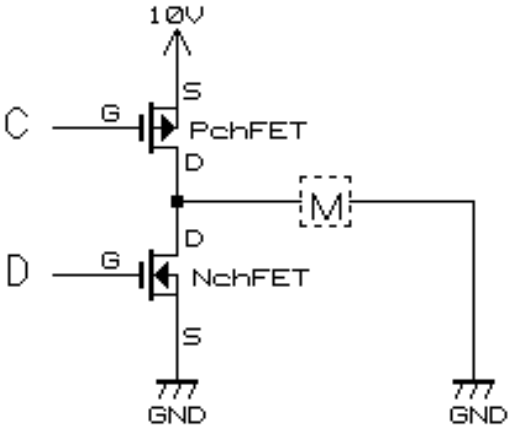
注) 1. パルス測定

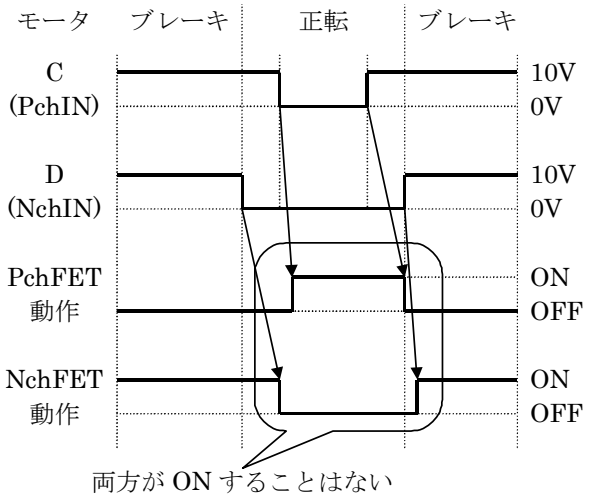
OFF→ON は 120ns 遅れる

ON→OFF は 225ns 遅れる

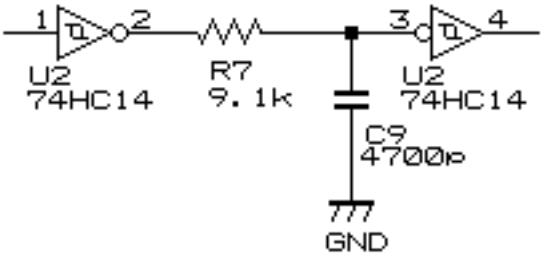
15.5.6 PチャネルとNチャネルの短絡防止回路

解決策としては、先ほどの回路図にある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。

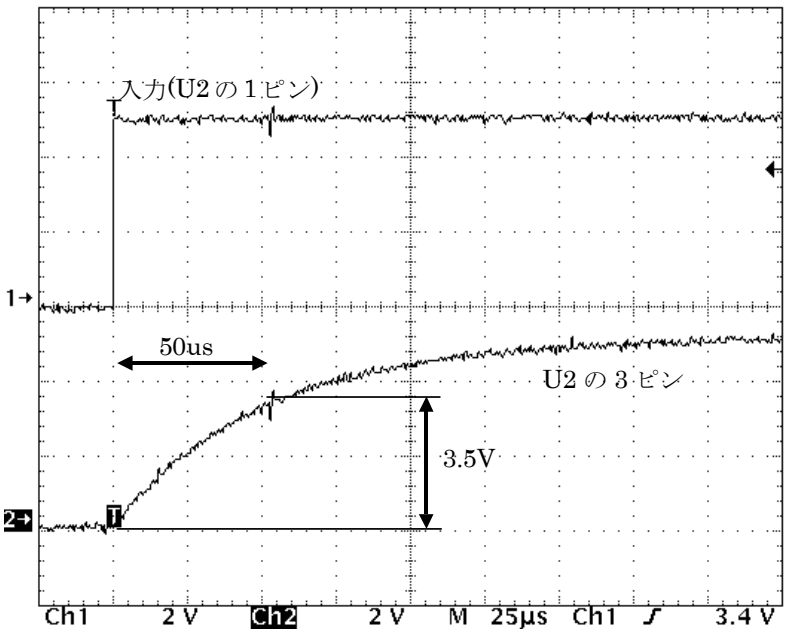




この時間をずらす部分を、積分回路で作ります。積分回路については、多数の専門書があるので、そちらを参照してください。



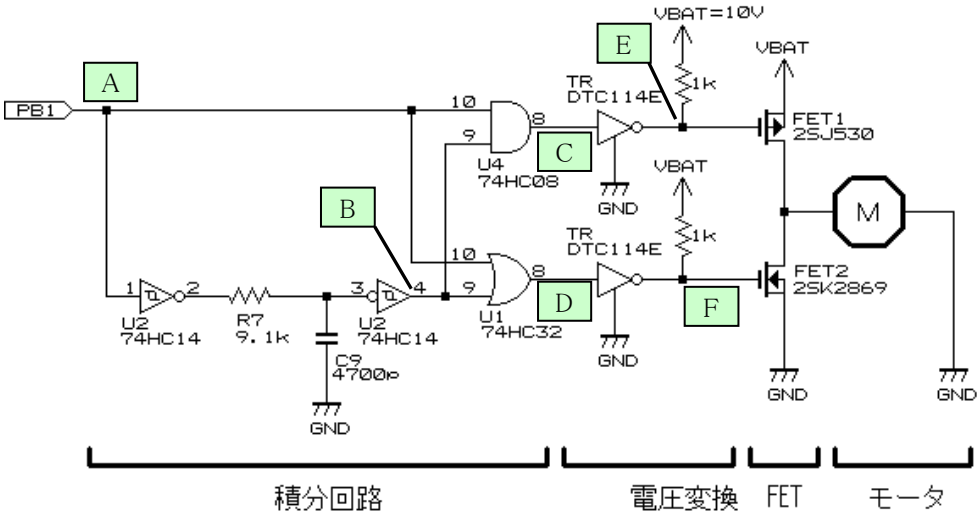
遅延時間はだいたい
時定数T＝CR［s］
で計算することができます。
今回は9.1kΩ、4700pFなので、計算すると
$T=9.1\times10^{3}\times4700\times10^{-12}$
$=42.77[\mu s]$
となります。

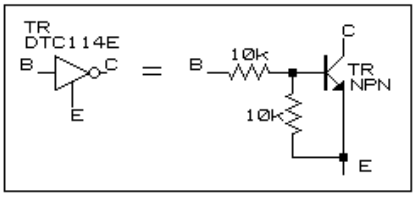


74HCシリーズは3.5V以上の入力電圧があると"1"とみなします。実際に波形を観測し、3.5Vになるまでの時間を計ると約50μsになりました。

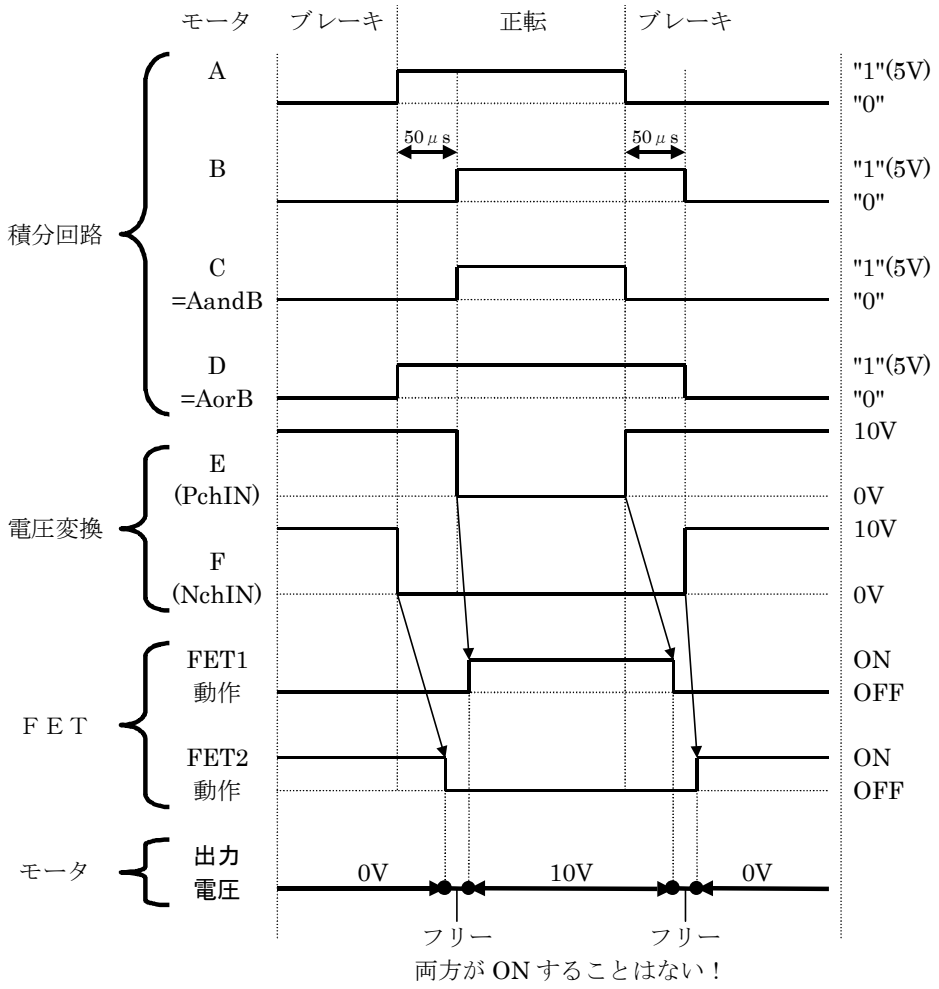
先ほどの「実際の波形」の図では最高でも225nsのずれしかありませんが、積分回路では50μsもの遅延時間を作っています。これは、FET以外にも、電圧変換用のデジタルトランジスタの遅延時間、FETのゲートのコンデンサ成分による遅れなどを含めたためです。

積分回路とFETを合わせた回路は下記のようになります。





デジタルトランジスタで
入力0V→出力10V(オープンコレクタ)
入力5V→出力0V
に変換します



(1) ブレーキ→正転に変えるとき

1. ポートからの信号は"0"でブレーキ、"1"で正転です。ブレーキから正転へ変えます(A点)。
2. 積分回路により 50μs 遅れた波形がB点より出力されます。
3. C点は、AandB の波形が出力されます。
4. D点は、AorB の波形が出力されます。
5. E点は、デジタルトランジスタ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

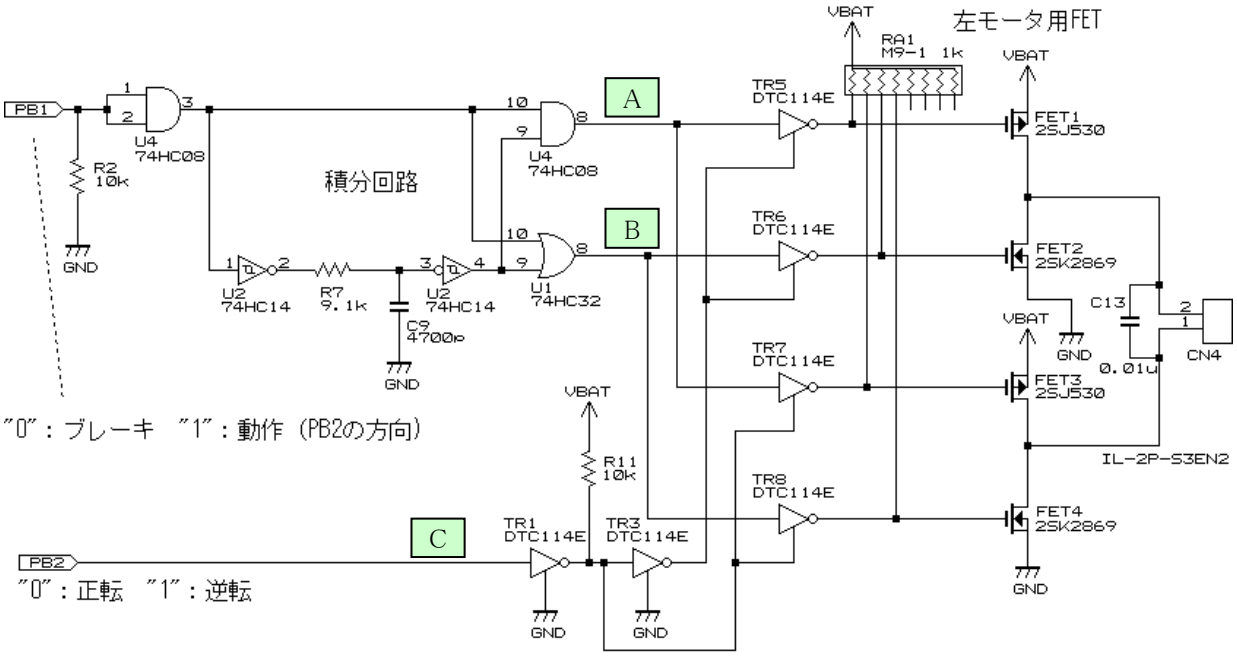
2. A点の信号を変えてから 50μs 後、今度は FET2 のゲートが 0V→10V となり ON します。0V がモータに加えられ、両端子 0V なのでブレーキ動作になります。

このように、動作を切り替えるときはいったん、両 FET とも OFF の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)と同じにします。

15.5.7 モータドライブ基板のモータ制御回路

実際の回路は、積分回路、FET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。PB1 が PWM を加える端子、PB2 が正転／逆転を切り替える端子です。



A	B	C	FET1 のゲート	FET2 のゲート	FET3 のゲート	FET4 のゲート	CN4 2 ピン	CN4 1 ピン	モータ動作
0	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		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	FET1 のゲート	FET2 のゲート	FET3 のゲート	FET4 のゲート	CN4 2 ピン	CN4 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

※A,B,C:"0"=0V、"1"=5V

※フリーについて

フリーは、PchFET と NchFET のショートを避けるために積分回路で作っています。そのため、プログラムでフリーにすることはできません。モータドライブ基板 Vol.3 の停止はすべてブレーキです。

フリーの時間を変えたい場合は、積分回路の C と R の値を変えます。

15.5.8 モータドライブ基板Vol.3 の接続

ピン番	信号、方向※	詳細	“0”	“1”	詳細
1	－	+5V			
2	基板←PB7	LED1	点灯	消灯	
3	基板←PB6	LED0	点灯	消灯	
4	基板←PB5	サーボ信号	PWM 信号		ITU4_BRB でデューティ比設定
5	**基板←PB4**	**右モータ PWM**	**停止**	**動作**	**ITU4_BRA でデューティ比設定**
6	基板←PB3	右モータ回転方向	正転	逆転	
7	基板←PB2	左モータ回転方向	正転	逆転	
8	**基板←PB1**	**左モータ PWM**	**停止**	**動作**	**ITU3_BRB でデューティ比設定**
9	基板→PB0	プッシュスイッチ	押された	押されてない	
10	－	GND			

PB4 が右モータ制御で、PWM 信号を加えることでスピード制御しています。右モータの正転／逆転は PB3 です。

PB1 が左モータ制御で、PWM 信号を加えることでスピード制御しています。左モータの正転／逆転は PB2 です。

15.6 プログラムの解説

15.6.1 リセット同期PWMモードの設定

servo.c と同じです。

```
54 :      ITU3_TCR = 0x23;                    /* カウンタ、クリアの設定   */
55 :      ITU_FCR  = 0x3e;                    /* リセット同期 PWM モード   */
56 :      ITU3_GRA = 49151;                   /* 周期の設定               */
57 :      ITU3_GRB = ITU3_BRB = 0;            /* 左モータの PWM 設定       */
58 :      ITU4_GRA = ITU4_BRA = 0;            /* 右モータの PWM 設定       */
59 :      ITU4_GRB = ITU4_BRB = 5000;         /* サーボの PWM 設定         */
60 :      ITU_TOER = 0x38;                    /* 出力端子の設定           */
61 :      ITU_STR  = 0x08;                    /* タイマスタート           */
```

バッファを使用するので ITU3_BRB で左モータ、ITU4_BRA で右モータのデューティ比を設定します。
最初、左モータ、右モータの PWM は0%に設定しています。

15.6.2 main関数

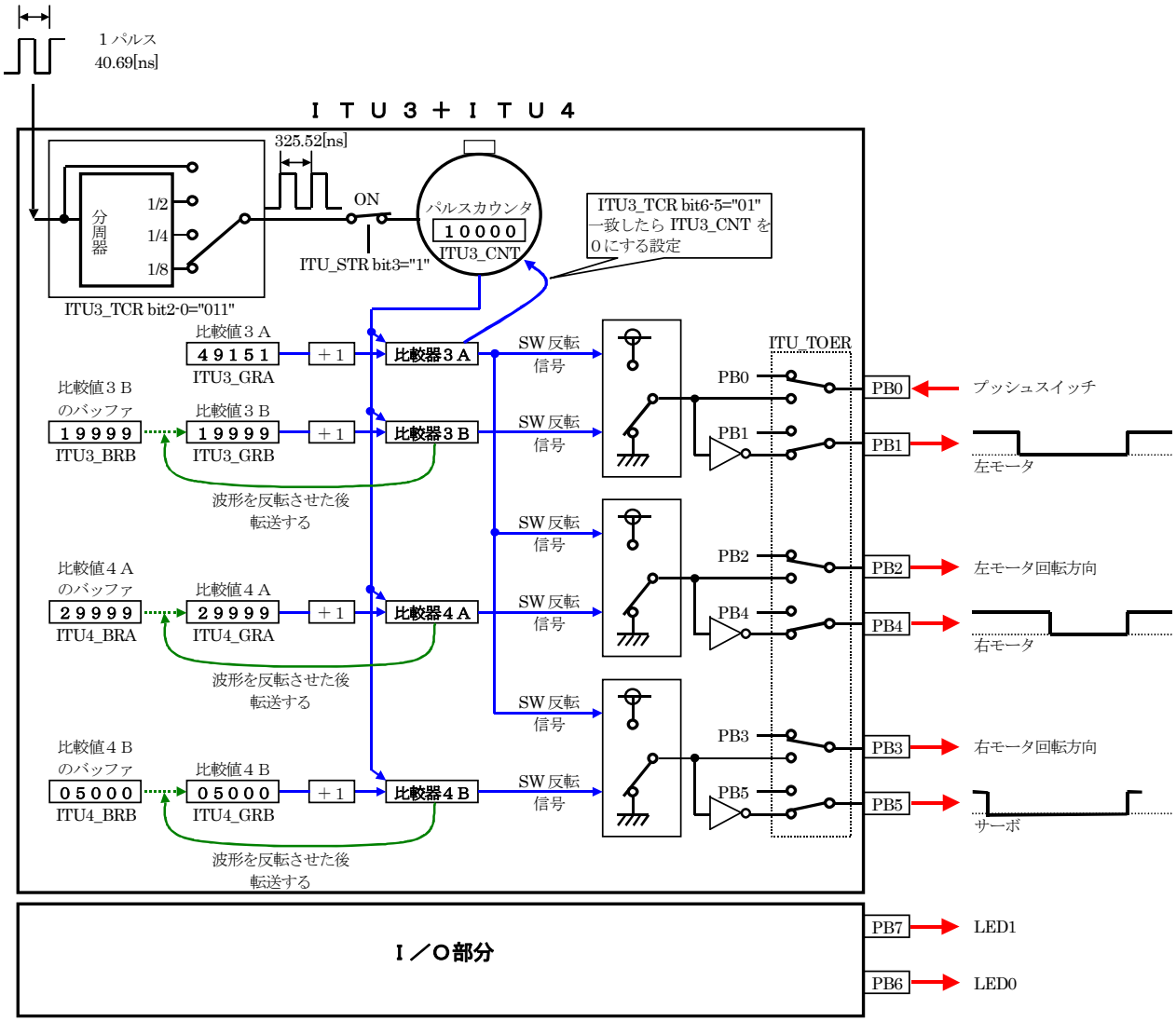
```
25 :  void main( void )
26 :  {
27 :      init();                             /* 初期化                   */
28 :
29 :      while( 1 ) {
30 :          ITU3_BRB = 49150 * dipsw_get() / 15;    /* 左モータ         */
31 :      }
32 :  }
```

左モータに繋がっている端子の PWM 波形を操作するので、ITU3_BRB に設定します。

15.6.3 演習

・30 行を変えて、右モータを制御するようにして見ましょう。
・PBDR に出力する値を変えて、モータを逆転させて見ましょう。

15.7 まとめ



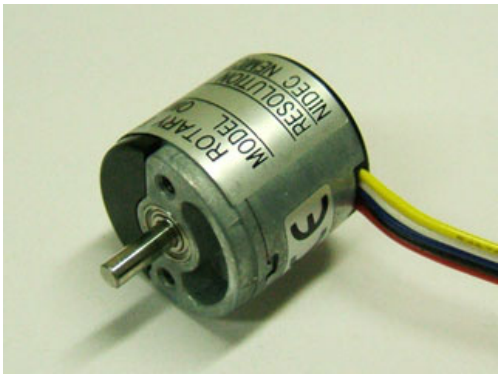
設定するレジスタ	詳細
ITU3_TCR	ITU3_CNT の値が＋1する時間、クリア要因の設定をします。 0x20…周期が 2.666ms 以下の場合(＋1する時間は 40.69[ns]ごと) 0x21…周期が 5.333ms 以下の場合(＋1する時間は 81.38[ns]ごと) 0x22…周期が 10.67ms 以下の場合(＋1する時間は 162.76[ns]ごと) 0x23…周期が 21.33ms 以下の場合(＋1する時間は 325.52[ns]ごと) 今回は、周期 16ms なので **0x23** を設定します。
ITU_FCR	リセット同期PWMモードの設定と、バッファレジスタを使用するかの設定します。 **0x3e** を設定します。PB0～PB5 の端子からは PWM 波形がされます。出力したくない場合は、ITU_TOER を設定することにより通常の I/O ポートとして使用できます。
ITU3_GRA	周期を設定します。計算は、「設定したい周期÷(ITU3_CNT の値が＋1する時間)－1」です。周期 16ms、＋1する時間 325.52ns なら $(16\times10^{-3})\div(325.52\times10^{-9})-1=49151$

ITU3_BRB (ITU3_GRB)	PB0 の OFF 幅、または PB1 の ON 幅を設定します。PB0 端子は通常の I/O として使用、PB1 端子には左モータが接続されています。 計算は、「設定したい幅÷(ITU3_CNT の値が＋1する時間)－1」です。 ITU3_GRB は、最初だけ ITU3_BRB と同じ値を設定します。 例えば、左モータの ON 幅を 1.6[ms](10%)としたい場合、 $(1.6\times10^{-3})\div(325.52\times10^{-9})-1=4914$ となります。
ITU4_BRA (ITU4_GRA)	PB2 の OFF 幅、または PB4 の ON 幅を設定します。PB2 端子は通常の I/O として使用、PB4 端子には右モータが接続されています。 計算は、「設定したい幅÷(ITU3_CNT の値が＋1する時間)－1」です。 ITU4_GRA は、最初だけ ITU4_BRA と同じ値を設定します。 例えば、右モータの ON 幅を 11.2[ms](70%)としたい場合、 $(11.2\times10^{-3})\div(325.52\times10^{-9})-1=34405$ となります。
ITU4_BRB (ITU4_GRB)	PB3 の OFF 幅、または PB5 の ON 幅を設定します。PB3 端子は通常の I/O として使用、PB5 端子にはサーボが接続されています。 計算は、「設定したい幅÷(ITU3_CNT の値が＋1する時間)－1」です。 ITU4_GRB は、最初だけ ITU4_BRB と同じ値を設定します。
ITU_TOER	PWM 波形を出力するかしないか選択します。 ″1″:PWM波形出力 ″0″:通常の I/O ポートとして使用 bit 7 6 5 4 3 2 1 0 0固定 0固定 PB5 PB4 PB1 PB3 PB2 PB0 今回は、PB5,4,1 を PWM として使用、他は I/O ポートなので PB5,4,1 の部分を″1″に、PB4,3,0 の部分を″0″にします。 bit 7 6 5 4 3 2 1 0 値 0 0 1 1 1 0 0 0 0011 1000→**0x38** を設定します。
ITU_STR	それぞれの ITU のチャネルで ITU_CNT をカウント動作させるかどうか選択します。要は、ITU を使うか使わないかの設定です。 ″1″:使用する ″0″:使用しない bit 7 6 5 4 3 2 1 0 0固定 0固定 0固定 ITU4 ITU3 ITU2 ITU1 ITU0 リセット同期 PWM モードでは ITU3 と 4 を使用します。しかしカウンタは ITU3 のみしか使用しません。そのため、ITU3 のみのカウンタを動作させます。 bit 7 6 5 4 3 2 1 0 値 0 0 0 0 1 0 0 0 0000 1000 →**0x08** を設定します。

16. プロジェクト「enc」 外部のパルスをカウント(1相)

16.1 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。

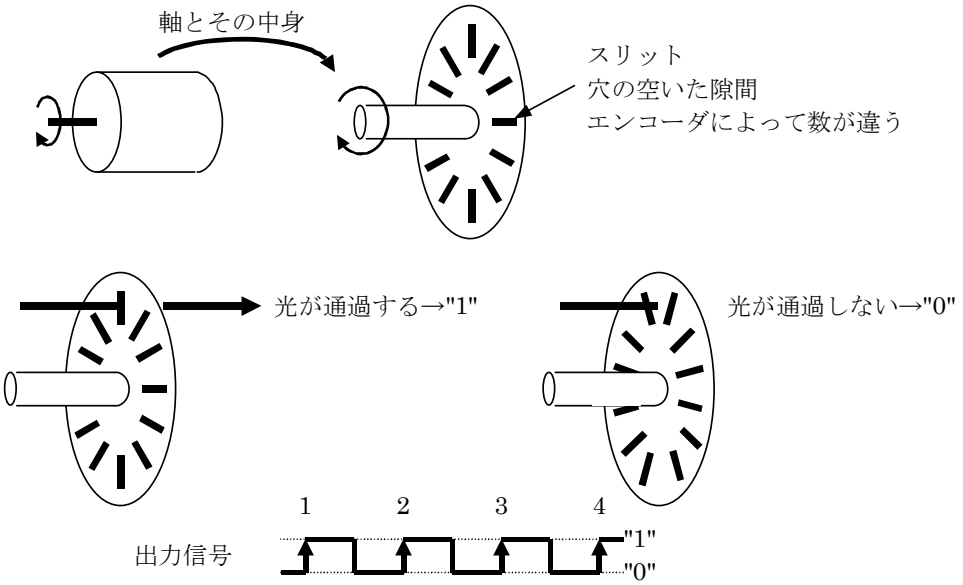


日本電産ネミコン(株) OME-100-1CA-105-015-00

16.1.1 ロータリエンコーダとは

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。この頃、パソコンに映像を取り込んだり、音声を取り込んだりすることが流行っていますが、ビデオ信号を MPEG データに変換したり、音声信号を PCM データに変換することをエンコード(符号化)すると言います。これらから、「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。

16.1.2 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	値段
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。	6500 円程度
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。	7000 円程度

※価格は参考です。必ず各自で調べてください

16.1.3 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、上記のとおり非常に高価です。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、マイコンカーで使用するにはほとんど問題ありません。

販売メーカ	型式	特徴	値段
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。	1 セット(2 台分)1575 円

※価格は参考です。必ず各自で調べてください

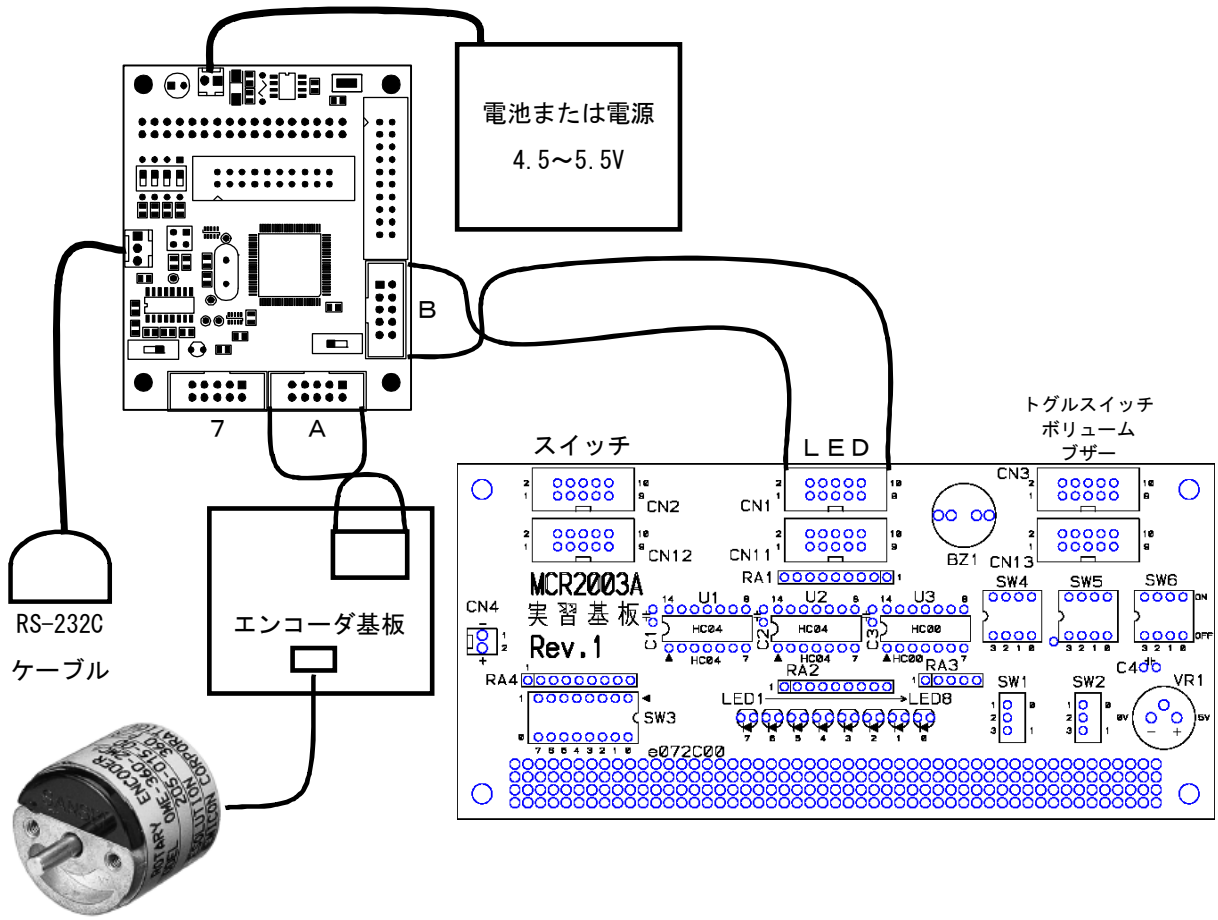
※日立インターメディックス販売サイトアドレス･･･http://www2.himdx.net/mcr/

16.2 接続

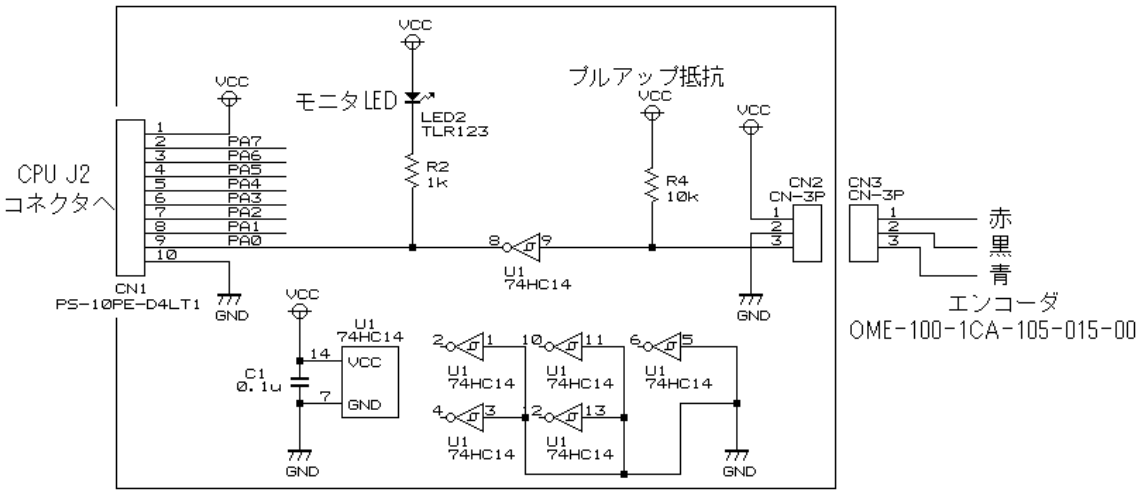
16.2.1 ロータリエンコーダ使用

・CPU ボードのポート A のビット 0 と、ロータリエンコーダの出力信号を接続します。
・CPU ボードのポート B と、実習基板の LED 部をフラットケーブルで接続します。

日本電産ネミコン(株)「OME-100-1CA-105-015-00」を使用した回路を下記に示します。出力信号は、デジタル信号なのでそのままポートに入力可能です。ただ、オープンコレクタなのでプルアップだけは必要です。一応、74HC14(シュミットトリガのインバータ)で波形整形します。他のエンコーダでも 0V と 5V が出力されるようにして、PA0 へ接続します。

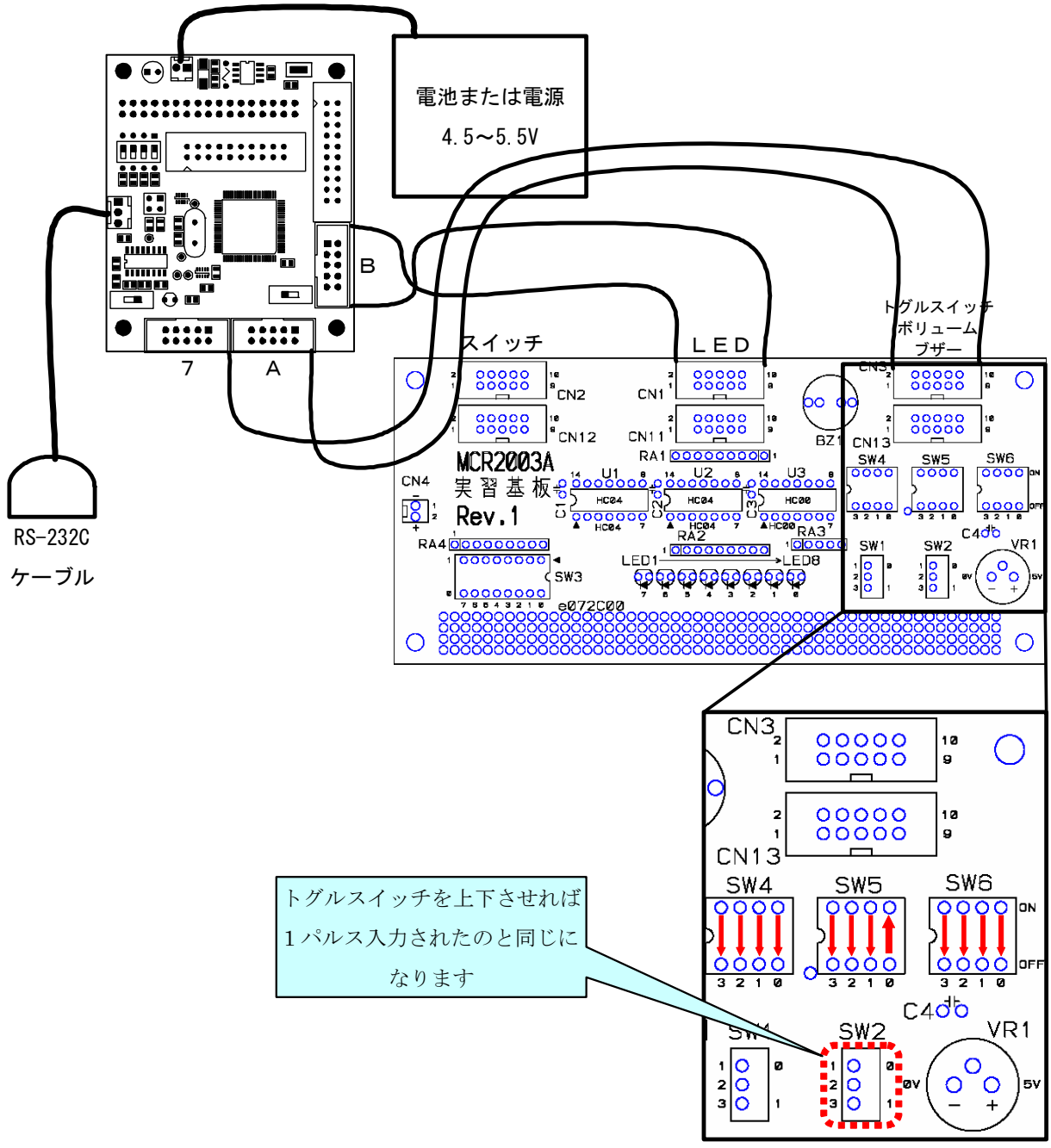


▼エンコーダ基板例



16.2.2 実習基板で代用

・CPU ボードのポート A と、トグルスイッチ・ボリューム部をフラットケーブルで接続します。
・CPU ボードのポート B と、LED 部をフラットケーブルで接続します。
・実習基板の SW5 No0 のスイッチを ON、SW4～6 のその他のビットを OFF にします。

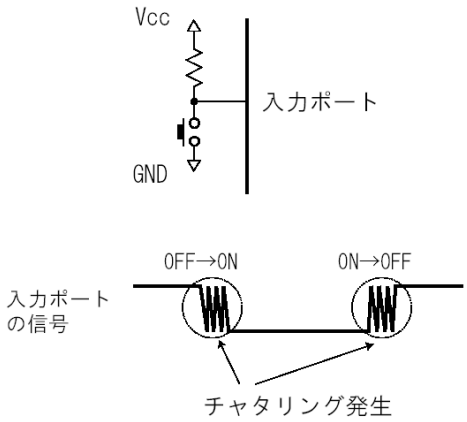


16.2.3 接続信号のチャタリングについて

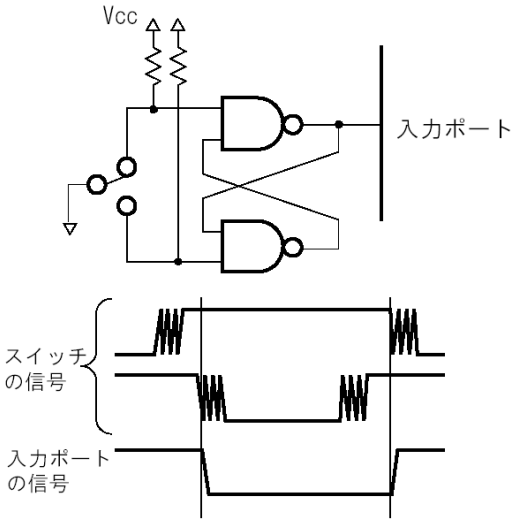
入力する信号は、ただのスイッチによるプルアップ回路ではチャタリングのため、誤カウントしてしまいます(下左図)。

チャタリング防止回路例を下右図に示します。実習基板には、下記チャタリング防止回路のあるトグルスイッチが 2 つあります(SW1、SW2)。実習基板を使うときは、ディップスイッチではなく、トグルスイッチを接続します。

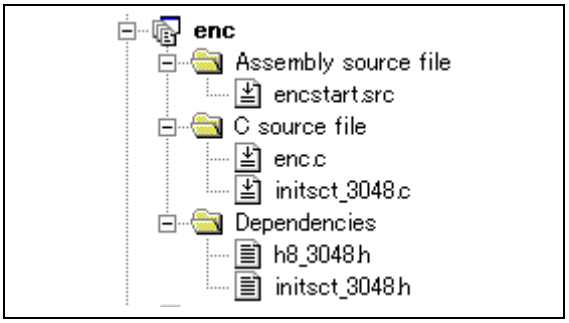
▼チャタリングの起こる回路



▼チャタリング防止回路



16.3 プロジェクトの構成



	ファイル名	内容
1	encstart.src	アセンブリ言語で記述されたアセンブリソースファイルです。このファイルの構造は下記のようになっています。 ベクタアドレス ＋ スタートアップルーチン
2	initsct_3048.c	初期値のないグローバル変数(セクション B 領域)、初期値のあるグローバル変数(セクション R 領域)の初期化用です。
3	enc.c	実際に制御するプログラムが書かれています。H8/3048F-ONE の内蔵周辺機能の初期化も行います。
4	h8_3048.h	H8/3048F-ONE の内蔵周辺機能の I/O レジスタを定義したファイルです。
5	initsct_3048.h	initsct_3048.c のヘッダファイルです。

16.4 プログラム「enc.c」

```
 1 :  /************************************************************************/
 2 :  /* 外部パルスをカウント(１相) 「enc.c」                                 */
 3 :  /* 入力：PA0(チャタリングのないパルス、エンコーダ等)　出力：PB7-PB0(LED等) */
 4 :  /*                              2005.04 ジャパンマイコンカーラリー実行委員会 */
 5 :  /************************************************************************/
 6 :  /*======================================*/
 7 :  /* インクルード                         */
 8 :  /*======================================*/
 9 :  #include    <machine.h>
10 :  #include    "h8_3048.h"
11 :
12 :  /*======================================*/
13 :  /* シンボル定義                         */
14 :  /*======================================*/
15 :
16 :  /*======================================*/
17 :  /* プロトタイプ宣言                     */
18 :  /*======================================*/
19 :  void init( void );
20 :
21 :  /************************************************************************/
22 :  /* メインプログラム                                                     */
23 :  /************************************************************************/
24 :  void main( void )
25 :  {
26 :      init();                             /* 初期化                   */
27 :
28 :      while( 1 ) {
29 :          PBDR = ITU2_CNT;
30 :      }
31 :  }
32 :
33 :  /************************************************************************/
34 :  /* H8/3048F-ONE 内蔵周辺機能の初期化                                    */
35 :  /************************************************************************/
36 :  void init( void )
37 :  {
38 :      /* ポートの入出力設定 */
39 :      P1DDR = 0xff;
40 :      P2DDR = 0xff;
41 :      P3DDR = 0xff;
42 :      P4DDR = 0xff;
43 :      P5DDR = 0xff;
44 :      P6DDR = 0xf0;                       /* CPU基板上のDIP SW        */
45 :      P8DDR = 0xff;
46 :      P9DDR = 0xf7;
47 :      PADDR = 0xfe;                       /* トグルスイッチ           */
48 :      PBDDR = 0xff;                       /* LED基板                  */
49 :      /* ポート７は、入力専用なので入出力設定はありません　*/
50 :
51 :      /* ITU2 パルス入力の設定 */
52 :      ITU2_TCR = 0x04;                    /* PA0端子のパルスでカウント*/
53 :      ITU_STR  = 0x04;                    /* タイマスタート           */
54 :  }
55 :
56 :  /************************************************************************/
57 :  /* end of file                                                          */
58 :  /************************************************************************/
```

16.5 プログラムの解説

16.5.1 ITU2 の初期設定

今まで ITU を使用するときは、CPU のクロックφを使って ITU の CNT をカウントしていました。このカウントするタイミングを、外部のパルスによって行います。ITU0～ITU4 のどれを使っても構いません。今回は、ITU のチャネル 2 を使います。

```
52 :      ITU2_TCR = 0x04;                  /* PA0 端子のパルスでカウント*/
53 :      ITU_STR  = 0x04;                  /* タイマスタート           */
```

●ITU2_TCR(タイマコントロールレジスタ)の設定内容

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

・ビット 6,5:カウンタクリア 1,0

CNT のカウンタクリア要因を選択します。

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

今までは、「GRA のコンペアマッチ／インプットキャプチャで CNT をクリア」の設定でしたが、今回は ITU2_CNT をクリアする必要はないので、クリアしません。

・ビット 4,3:クロックエッジ 1,0

外部クロック選択時に、外部クロックの入力エッジを選択します。

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

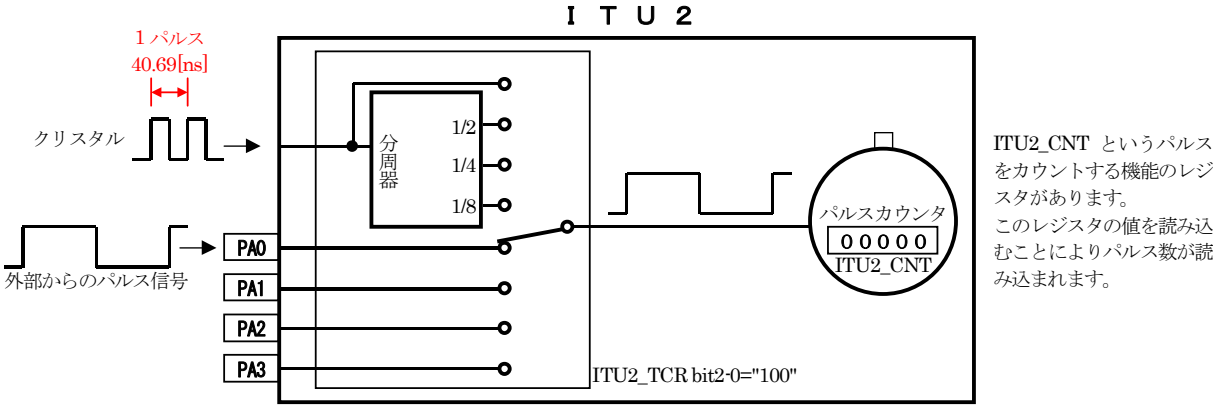
外部パルスの立ち上がりで ITU2_CNT が＋1します。

・ビット 2～0:タイマプリスケーラ 2～0

CNT のカウントクロックを選択します。

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

イメージとしては下図のようになります。



今までは、CPU ボード上のクリスタルのクロックによってカウンタの値を＋1していましたが、今回は、エンコーダからのパルスで増やしていきます。入力端子はポート A の bit0～3 から選ぶことができます。今回は、PA0 に接続します。

外部パルスをカウントする場合、ポート A の bit0～3 の端子以外を使用することはできません。

●ITU_STR(タイマスタートレジスタ)の設定内容

ビット:	7	6	5	4	3	2	1	0
ITU_STR:	－	－	－	STR4	STR3	STR2	STR1	STR0
設定値:	0	0	0	0	0	1	0	0
16進数:	0				4			

・ビット 4-0:カウンタスタート 4～0

タイマカウンタ x の動作／停止を選択します。

STR4-0	説明
0	ITUx の CNT のカウント動作は停止
1	ITUx の CNT はカウント動作

※ x は 0～4

ITU2 を使うので、ビット 2 を"1"にします。

16.5.2 I/Oポートの入出力設定

```
38 :      /* ポートの入出力設定 */
39 :      P1DDR = 0xff;
40 :      P2DDR = 0xff;
41 :      P3DDR = 0xff;
42 :      P4DDR = 0xff;
43 :      P5DDR = 0xff;
44 :      P6DDR = 0xf0;                 /* CPU 基板上の DIP SW        */
45 :      P8DDR = 0xff;
46 :      P9DDR = 0xf7;
47 :      PADDR = 0xfe;                 /* トグルスイッチ             */
48 :      PBDDR = 0xff;                 /* LED 基板                   */
```

PA0 はパルス入力端子です。入力端子なので PADDR の bit0="0"とします。忘れないようにします。

16.5.3 main関数

```
24 :  void main( void )
25 :  {
26 :      init();                              /* 初期化                    */
27 :
28 :      while( 1 ) {
29 :          PBDR = ITU2_CNT;
30 :      }
31 :  }
```

パルスの入力により ITU2_CNT が増えていきます。その値をそのままポート B へ出力します。LED は 8bit しか無いため、ITU2_CNT の下位 8bit の値が LED へ出力されます。ITU2_CNT は 65535 までカウントできます。65535 の次は 0 になります。

このサンプルプログラムでは、ITU2_CNT の値を読み込んだままですが、マイコンカーでの応用として 10[ms]ごとに ITU2_CNT の値を読み込み、long 型の大域変数に足していき、前回との差分を計算することにより 10ms ごとに進んだ距離が分かります。

long 型の変数は、21 億回のカウント値を保存できますので、すべてのカウントを加算すればコースの距離が分かります。

応用としては、クランク 0.5～1m 手前のクロスライン(横線)を検出後、40cm 以上進まなければ直角検出ルーチンへ行かないようにすれば誤動作を防げます。

また、10[ms]ごとにカウント値を読み込んで、ある値以上ならブレーキ、ある値以下になったらモータの PWM を 80%にする、などの制御をするとクランクでのブレーキが長すぎてタイムロスしたり、スピードが乗りすぎて直角を曲がりきれないということが無くなります。

16.6 まとめ

設定するレジスタ	詳細
ITU2_TCR	ITU2_CNT をカウントさせる端子の設定をします。 0x04…PA0 端子　0x05…PA1 端子 0x06…PA2 端子　0x07…PA3 端子 今回は、**0x04** を設定します。
ITU_STR	それぞれの ITU のチャネルで ITU_CNT をカウント動作させるかどうか選択します。要は、ITU を使うか使わないかの設定です。 "1":使用する "0":使用しない bit 7 6 5 4 3 2 1 0 0固定 0固定 0固定 ITU4 ITU3 ITU2 ITU1 ITU0 **0 0 0 0 0 1 0 0** 今回は ITU2 を使用するので、ITU2 の部分が"1"、他は"0"なので **0x04** を設定します。

設定後は、プログラム内で ITU2_CNT の値を読み込めば、パルス値が入力されています。

17. プロジェクト「enc2」 外部のパルスをカウント(2相)

17.1 概要

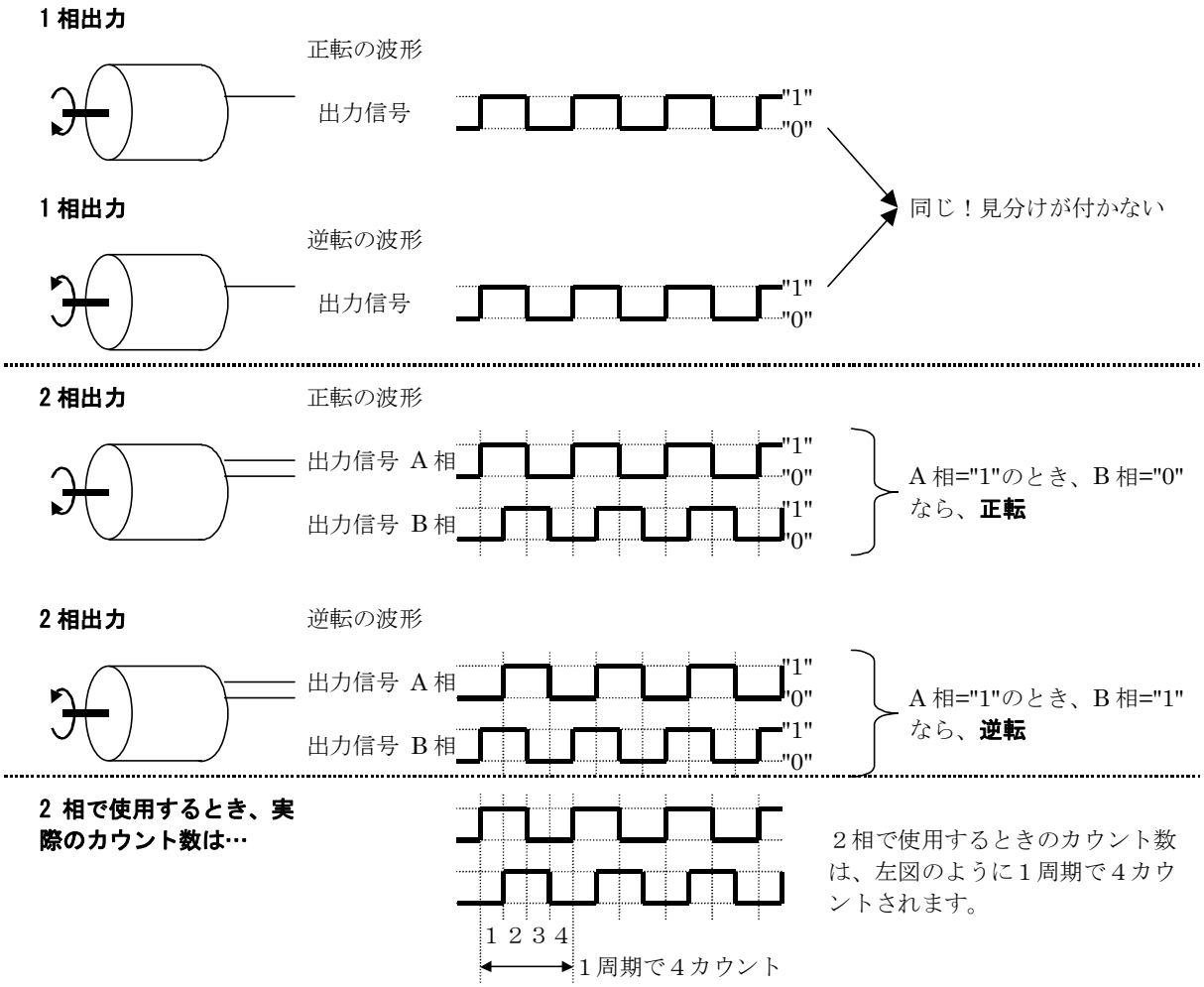
プロジェクト「enc」を実習すると、あることに気づきます。正転させても逆転させても LED の値が増えていきます。それは、1 相のエンコーダの場合、回転が正転か逆転か分かりません。どちらも"1"と"0"の信号でしかないためです。

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。

17.1.1 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。1 相の場合、回転が正転か逆転か分かりません。どちらも"1"と"0"の信号でしかないためです。

そこで、出力を A 相という名前と、B 相という名前の 2 つ出力します。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。



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。

17.1.2 市販されている 2 相出力のロータリエンコーダ

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	値段
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。	8000 円程度
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。	8000 円程度

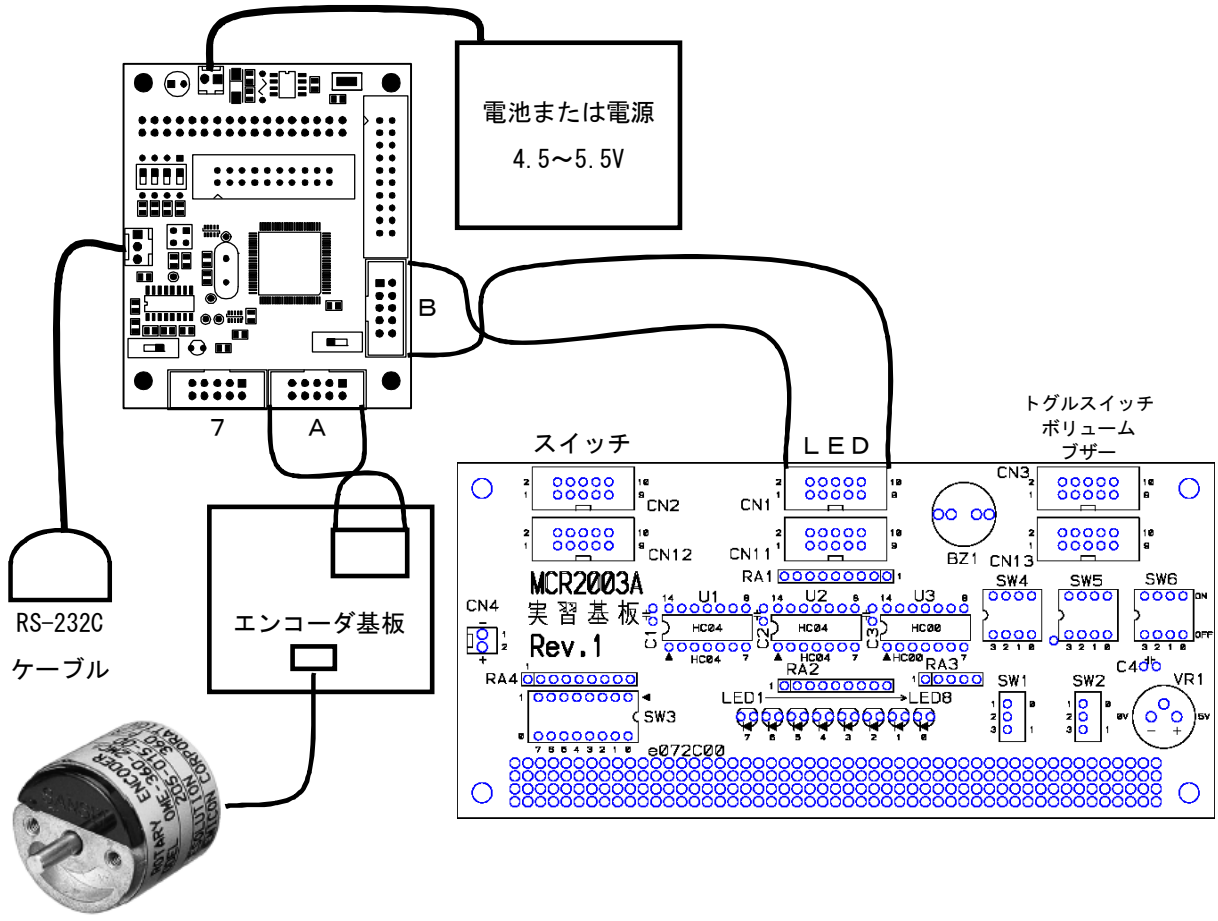
※価格は参考です。必ず各自で調べてください

17.2 接続

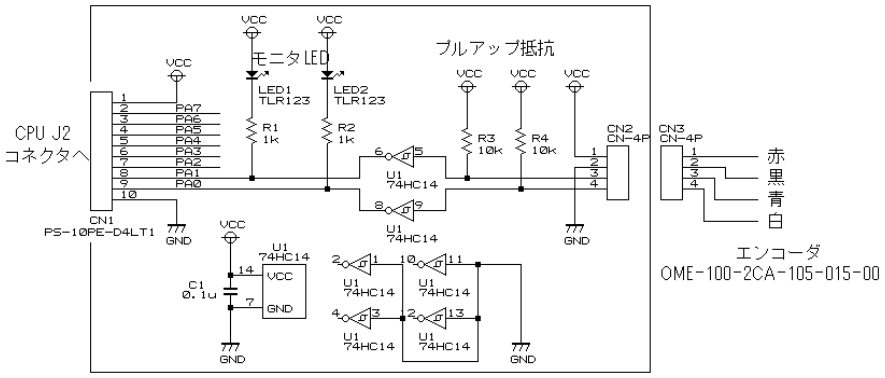
17.2.1 ロータリエンコーダ使用

・CPU ボードのポート A のビット 0 と、ロータリエンコーダの A 相出力信号を接続します。
・CPU ボードのポート A のビット 1 と、ロータリエンコーダの B 相出力信号を接続します。
・CPU ボードのポート B と、実習基板の LED 部をフラットケーブルで接続します。

日本電産ネミコン(株)「OME-100-2CA-105-015-00」を使用した回路を下記に示します。出力信号は、デジタル信号なのでそのままポートに入力可能です。ただ、オープンコレクタなのでプルアップだけは必要です。ただ、オープンコレクタなのでプルアップだけは必要です。一応、74HC14(シュミットトリガのインバータ)で波形整形します。他のエンコーダでも 0V と 5V が出力されるようにして、PA0 と PA1 へ接続します。

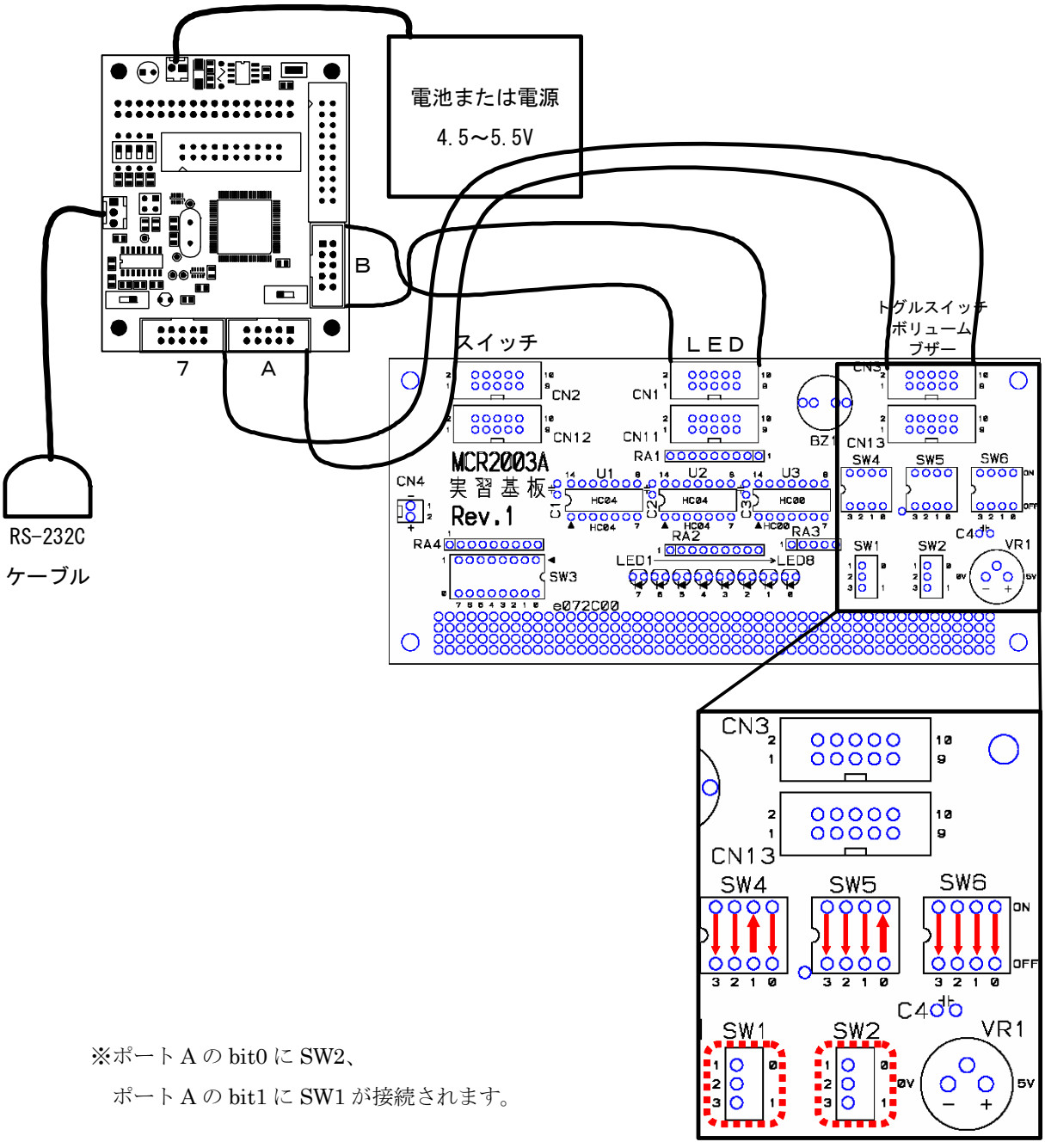


▼エンコーダ基板例



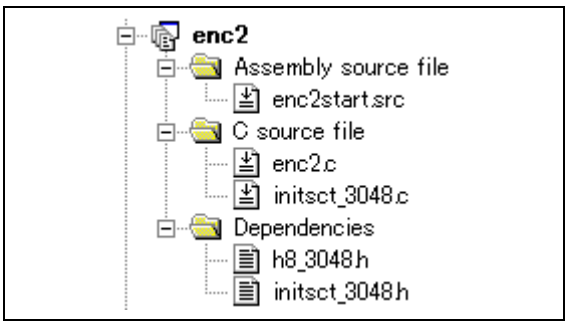
17.2.2 実習基板で代用

・CPU ボードのポート A と、トグルスイッチ・ボリューム部をフラットケーブルで接続します。
・CPU ボードのポート B と、LED 部をフラットケーブルで接続します。
・実習基板の SW4 No0 のスイッチを ON、SW5 No1 を ON、SW4～6 のその他のビットを OFF にします。



※ポート A の bit0 に SW2、
ポート A の bit1 に SW1 が接続されます。

17.3 プロジェクトの構成



	ファイル名	内容
1	enc2start.src	アセンブリ言語で記述されたアセンブリソースファイルです。このファイルの構造は下記のようになっています。 ベクタアドレス ＋ スタートアップルーチン
2	initsct_3048.c	初期値のないグローバル変数(セクション B 領域)、初期値のあるグローバル変数(セクション R 領域)の初期化用です。
3	enc2.c	実際に制御するプログラムが書かれています。H8/3048F-ONE の内蔵周辺機能の初期化も行います。
4	h8_3048.h	H8/3048F-ONE の内蔵周辺機能の I/O レジスタを定義したファイルです。
5	initsct_3048.h	initsct_3048.c のヘッダファイルです。

17.4 プログラム「enc2.c」

```
 1 :  /************************************************************************/
 2 :  /* 外部パルスをカウント(２相) 「enc2.c」                                */
 3 :  /* 入力：PA0,PA1(２相のロータリエンコーダ)       出力：PB7-PB0(LED等)    */
 4 :  /*                              2005.04 ジャパンマイコンカーラリー実行委員会  */
 5 :  /************************************************************************/
 6 :  /*======================================*/
 7 :  /* インクルード                         */
 8 :  /*======================================*/
 9 :  #include    <machine.h>
10 :  #include    "h8_3048.h"
11 :
12 :  /*======================================*/
13 :  /* シンボル定義                         */
14 :  /*======================================*/
15 :
16 :  /*======================================*/
17 :  /* プロトタイプ宣言                     */
18 :  /*======================================*/
19 :  void init( void );
20 :
21 :  /************************************************************************/
22 :  /* メインプログラム                                                     */
23 :  /************************************************************************/
24 :  void main( void )
25 :  {
26 :      init();                             /* 初期化                       */
27 :
28 :      while( 1 ) {
29 :          PBDR = ITU2_CNT;
30 :      }
31 :  }
32 :
33 :  /************************************************************************/
34 :  /* H8/3048F-ONE 内蔵周辺機能の初期化                                    */
35 :  /************************************************************************/
36 :  void init( void )
```

```
37 :  {
38 :      /* ポートの入出力設定 */
39 :      P1DDR = 0xff;
40 :      P2DDR = 0xff;
41 :      P3DDR = 0xff;
42 :      P4DDR = 0xff;
43 :      P5DDR = 0xff;
44 :      P6DDR = 0xf0;                    /* CPU基板上のDIP SW        */
45 :      P8DDR = 0xff;
46 :      P9DDR = 0xf7;
47 :      PADDR = 0xfc;                    /* エンコーダ               */
48 :      PBDDR = 0xff;                    /* LED基板                  */
49 :      /* ポート７は、入力専用なので入出力設定はありません    */
50 :
51 :      /* ITU2 位相計数モードの設定 */
52 :      ITU_MDR = 0x40;                  /* ITU2は位相計数モード     */
53 :      ITU_STR = 0x04;                  /* タイマスタート           */
54 :  }
55 :
56 :  /************************************************************************/
57 :  /* end of file                                                          */
58 :  /************************************************************************/
```

17.5 プログラムの解説

17.5.1 ITU2 を位相計数モードで使う

2 相のエンコーダのパルスをカウントするために、位相計数モードという機能があります。これは ITU2 のみで使用可能です。

```
52 :      ITU_MDR = 0x40;                  /* ITU2 は位相計数モード     */
53 :      ITU_STR = 0x04;                  /* タイマスタート            */
```

●ITU_MDR(タイマモードレジスタ)の設定内容

ビット:	7	6	5	4	3	2	1	0
ITU_MDR:	－	MDF	FDIR	PWM4	PWM3	PWM2	PWM1	PWM0
設定値:	0	1	0	0	0	0	0	0
16 進数:	4				0			

・ビット 6:位相計数モード(MDF)

チャネル 2 を通常動作させるか、位相計数モードで動作させるかを選択します。

MDF	説明
0	チャネル 2 は通常動作
1	チャネル 2 は位相計数モード

ITU2 を位相計数モードで使用するので、"1"にします。

・ビット 5:フラグディレクション(FDIR)

TSR2 の OVF フラグのセット条件を設定します。本ビットの設定は、チャネル 2 がいずれのモードで動作していても有効となります。

FDIR	説明
0	TSR2 の OVF フラグは、TCNT2 がオーバフローまたはアンダフローしたときに1にセット
1	TSR2 の OVF フラグは、TCNT2 がオーバフローしたときに1にセット

特に関係有りません。元の設定のままにします。

・ビット 4～0:PWM モード 4～0

チャネル x を通常動作させるか、PWM モードで動作させるかを選択します。

PWM x	説明
0	チャネル x は通常動作
1	チャネル x は PWM モード

※ x は 0～4

PWM は使用しないので、どのビットも 0 です。

今回は ITU2_TCR の設定はありません。**ITU_MDR のビット 6 を"1"にして位相計数モードで使用にすると、自動的に PA0 と PA1 がパルス入力端子になります**。そのため、ITU2_TCR は設定しなくとも良いのです。
位相計数モードにすると、PA0 と PA1 は I/O ポートとして使用できません。

●ITU_STR(タイマスタートレジスタ)の設定内容

ビット:	7	6	5	4	3	2	1	0
ITU_STR:	−	−	−	STR4	STR3	STR2	STR1	STR0
設定値:	0	0	0	0	0	1	0	0
16進数:	0				4			

・ビット 4-0:カウンタスタート 4～0

タイマカウンタ x の動作／停止を選択します。

STR4-0	説明
0	ITUx の CNT のカウント動作は停止
1	ITUx の CNT はカウント動作

※ x は 0～4

ITU2 を使いますので、ビット2を"1"にします。

17.5.2 main関数

```
24 :  void main( void )
25 :  {
26 :      init();                              /* 初期化                    */
27 :
28 :      while( 1 ) {
29 :          PBDR = ITU2_CNT;
30 :      }
31 :  }
```

位相計数モードにしたときの ITU2_CNT の変化の仕方を下記に示します。

カウント方向	ITU2_CNT カウントダウン				ITU2_CNT カウントアップ			
TCLKA(PA0)端子	↑	"1"	↓	"0"	↑	"0"	↓	"1"
TCLKB(PA1)端子	"0"	↑	"1"	↓	"1"	↑	"0"	↓

2 相のロータリエンコーダの回転方向により、ITU2_CNT がカウントダウンしたりカウントアップします。
ポートBへ ITU2_CNT の値を出力していますので、ロータリエンコーダの回転方向により、LED の数値が増えたり減ったりします。

17.6 まとめ

設定するレジスタ	詳細
ITU_MDR	ITU2 を位相計数モードというモードに設定します。**0x40** を設定します。 端子は、自動的に PA0 と PA1 が2相ロータリエンコーダのパルス入力になります。 ちなみに、位相計数モードは ITU2 のみでしか使用できません。
ITU_STR	enc.c と同様です。

後は、プログラム内で ITU2_CNT の値を読み込めば、パルス値が入力されています。

18. プロジェクト「pulsein」 ラジコン受信機のパルス幅を測定

18.1 概要

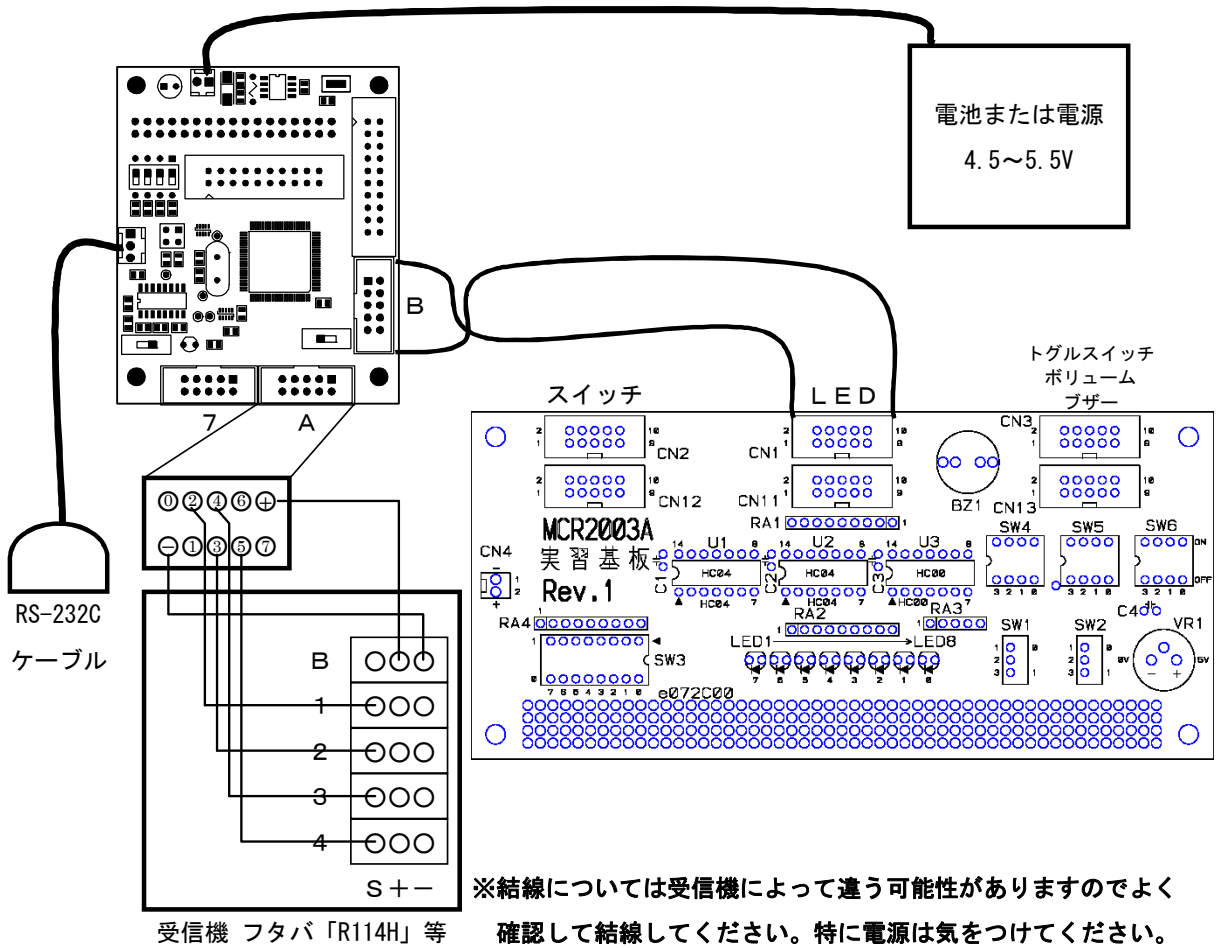
▲フタバ　R114H

プロポの操作信号を受信する受信機があります。その受信機が出力される PWM 信号をマイコンで取り込んで、ON 幅を測定します。

今までは、サーボを制御するために PWM 信号を作っていましたが、今回は PWM 信号を受信して ON の幅を測定、LED へ出力します。LED の変わりに様々な機器を接続すれば、ラジコンのプロポで様々な制御ができます。

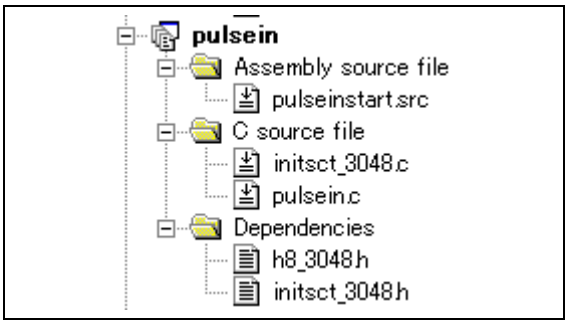
18.2 接続

・CPU ボードのポート B と、実習基板の LED 部をフラットケーブルで接続します。
・CPU ボードのポート A の bit2,3,4,5 と、受信機の PWM 出力信号を接続します。



※結線については受信機によって違う可能性がありますのでよく確認して結線してください。特に電源は気をつけてください。

18.3 プロジェクトの構成



	ファイル名	内容
1	pulseinstart.src	アセンブリ言語で記述されたアセンブリソースファイルです。このファイルの構造は下記のようになっています。 ベクタアドレス ＋ スタートアップルーチン
2	initsct_3048.c	初期値のないグローバル変数(セクション B 領域)、初期値のあるグローバル変数(セクション R 領域)の初期化用です。
3	pulsein.c	実際に制御するプログラムが書かれています。H8/3048F-ONE の内蔵周辺機能の初期化も行います。
4	h8_3048.h	H8/3048F-ONE の内蔵周辺機能の I/O レジスタを定義したファイルです。
5	initsct_3048.h	initsct_3048.c のヘッダファイルです。

18.4 プログラム「pulsein.c」

```
/************************************************************************/
/* プロポの受信機から出力されるPWMのON幅測定「pulsein.c」                */
/* 入力：PA2-PA5(プロポの受信機信号)     出力：PB7-PB0(LED等)            */
/*                          2004.04 ジャパンマイコンカーラリー実行委員会  */
/************************************************************************/
/*======================================*/
/* インクルード                         */
/*======================================*/
#include    <machine.h>
#include    "h8_3048.h"

/*======================================*/
/* シンボル定義                         */
/*======================================*/
#define         TIMER_CYCLE     3071    /* タイマのサイクル 1ms     */
                                        /* φ/8で使用する場合、     */
                                        /* φ/8 = 325.5[ns]         */
                                        /* ∴TIMER_CYCLE =           */
                                        /*      1[ms] / 325.5[ns]   */
                                        /*              = 3072      */

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

/*======================================*/
/* グローバル変数の宣言                 */
/*======================================*/
unsigned long   cnt0;                   /* タイマ用                 */

unsigned int    uPulseWidth0Before;     /* 前回値 ITU0_GRA          */
unsigned int    uPulseWidth1Before;     /* 前回値 ITU0_GRB          */
unsigned int    uPulseWidth2Before;     /* 前回値 ITU1_GRA          */
unsigned int    uPulseWidth3Before;     /* 前回値 ITU1_GRB          */
unsigned int    uPulseWidth0;           /* パルス幅 TIOCA0(PA2)     */
```

```
unsigned int    uPulseWidth1;           /* パルス幅 TIOCB0(PA3)     */
unsigned int    uPulseWidth2;           /* パルス幅 TIOCA1(PA4)     */
unsigned int    uPulseWidth3;           /* パルス幅 TIOCB1(PA5)     */
/*
1あたり、1 / 24.576MHz = 40.69ns
0.8ms / 40.69ns = 19661
1.5ms / 40.69ns = 36864
2.3ms / 40.69ns = 56525
*/

int             flag;                   /* パルスの変化があると     */
                                        /* 該当ビットが"1"になる    */

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

    /* マイコン機能の初期化 */
    init();                             /* 初期化                   */
    set_ccr( 0x00 );                    /* 全体割り込み許可         */

    while( 1 ) {
        switch( dipsw_get() ) {
        case 0:
            PBDR = uPulseWidth0 >> 8;   /* PA2のパルス幅 下位8bit   */
            break;
        case 1:
            PBDR = uPulseWidth1 >> 8;   /* PA3のパルス幅 下位8bit   */
            break;
        case 2:
            PBDR = uPulseWidth2 >> 8;   /* PA4のパルス幅 下位8bit   */
            break;
        case 3:
            PBDR = uPulseWidth3 >> 8;   /* PA5のパルス幅 下位8bit   */
            break;
        default:
            PBDR = 0xff;
            break;
        }
    }
}

/************************************************************************/
/* H8/3048F内蔵モジュール　初期化                                       */
/************************************************************************/
void init( void )
{
    /* ポートの入出力設定 */
    P1DDR = 0xff;
    P2DDR = 0xff;
    P3DDR = 0xff;
    P4DDR = 0xff;
    P5DDR = 0xff;
    P6DDR = 0xf0;                       /* CPU基板上のDIP SW        */
    P8DDR = 0xff;
    P9DDR = 0xf7;                       /* 通信ポート               */
    PADDR = 0xc3;                       /* PA5:TIOCB1 PA4:TIOCA1    */
                                        /* PA3:TIOCB0 PA2:TIOCA0    */
    PBDDR = 0xff;                       /* LED基板                  */
    /* ポート７は、入力専用なので入出力設定はありません    */

    /* ITU0 インプットキャプチャ入力 */
    ITU0_TCR  = 0x00;                   /* CNTのカウントはφ        */
    ITU0_TIOR = 0x77;                   /* TIOCA0=input TIOCB0=input */
    ITU0_IER  = 0x03;                   /* IMIA,IMIB許可            */

    /* ITU1 インプットキャプチャ入力 */
    ITU1_TCR  = 0x00;                   /* CNTのカウントはφ        */
    ITU1_TIOR = 0x77;                   /* TIOCA1=input TIOCB1=input */
    ITU1_IER  = 0x03;                   /* IMIA,IMIB許可            */

    /* ITU2 1ms毎の割り込み 内部時間用 */
    ITU2_TCR = 0x23;
    ITU2_GRA = TIMER_CYCLE;
    ITU2_IER = 0x01;                    /* 割り込み許可             */

    ITU_STR = 0x07;                     /* ITUのカウントスタート    */

    uPulseWidth0 = 0;
    uPulseWidth1 = 0;
    uPulseWidth2 = 0;
    uPulseWidth3 = 0;
}

/************************************************************************/
/* ITU0 IMIA0割り込み処理                                               */
/************************************************************************/
#pragma interrupt( interrupt_imia0 )
```

```
130 :  void interrupt_imia0( void )
131 :  {
132 :      ITU0_TSR &= 0xfe;                   /* フラグクリア             */
133 :      if( !(PADR & 0x04) ) {
134 :          /* "1"なら */
135 :          uPulseWidth0 = ITU0_GRA - uPulseWidth0Before;
136 :          uPulseWidth0 -= 8;              /* プログラム実行分の時間を引く */
137 :          flag |= 0x01;
138 :      }
139 :      uPulseWidth0Before = ITU0_GRA;
140 :  }
141 :
142 :  /************************************************************************/
143 :  /* ITU0 IMIB0割り込み処理                                               */
144 :  /************************************************************************/
145 :  #pragma interrupt( interrupt_imib0 )
146 :  void interrupt_imib0( void )
147 :  {
148 :      ITU0_TSR &= 0xfd;                   /* フラグクリア             */
149 :      if( !(PADR & 0x08) ) {
150 :          /* "1"なら */
151 :          uPulseWidth1 = ITU0_GRB - uPulseWidth1Before;
152 :          uPulseWidth1 -= 8;              /* プログラム実行分の時間を引く */
153 :          flag |= 0x02;
154 :      }
155 :      uPulseWidth1Before = ITU0_GRB;
156 :  }
157 :
158 :  /************************************************************************/
159 :  /* ITU1 IMIA1割り込み処理                                               */
160 :  /************************************************************************/
161 :  #pragma interrupt( interrupt_imia1 )
162 :  void interrupt_imia1( void )
163 :  {
164 :      ITU1_TSR &= 0xfe;                   /* フラグクリア             */
165 :      if( !(PADR & 0x10) ) {
166 :          /* "1"なら */
167 :          uPulseWidth2 = ITU1_GRA - uPulseWidth2Before;
168 :          uPulseWidth2 -= 8;              /* プログラム実行分の時間を引く */
169 :          flag |= 0x04;
170 :      }
171 :      uPulseWidth2Before = ITU1_GRA;
172 :  }
173 :
174 :  /************************************************************************/
175 :  /* ITU1 IMIB1割り込み処理                                               */
176 :  /************************************************************************/
177 :  #pragma interrupt( interrupt_imib1 )
178 :  void interrupt_imib1( void )
179 :  {
180 :      ITU1_TSR &= 0xfd;                   /* フラグクリア             */
181 :      if( !(PADR & 0x20) ) {
182 :          /* "1"なら */
183 :          uPulseWidth3 = ITU1_GRB - uPulseWidth3Before;
184 :          uPulseWidth3 -= 8;              /* プログラム実行分の時間を引く */
185 :          flag |= 0x08;
186 :      }
187 :      uPulseWidth3Before = ITU1_GRB;
188 :  }
189 :
190 :  /************************************************************************/
191 :  /* ITU2 割り込み処理                                                    */
192 :  /************************************************************************/
193 :  #pragma interrupt( interrupt_timer2 )
194 :  void interrupt_timer2( void )
195 :  {
196 :      ITU2_TSR &= 0xfe;                   /* フラグクリア             */
197 :      cnt0++;
198 :  }
199 :
200 :  /************************************************************************/
201 :  /* タイマ本体                                                           */
202 :  /* 引数    タイマ値 1=1ms                                               */
203 :  /************************************************************************/
204 :  void timer( unsigned long timer_set )
205 :  {
206 :      cnt0     = 0;
207 :      while( cnt0 < timer_set );
208 :  }
209 :
210 :  /************************************************************************/
211 :  /* ディップスイッチ値読み込み                                           */
212 :  /* 戻り値 スイッチ値 0～15                                              */
213 :  /************************************************************************/
214 :  unsigned char dipsw_get( void )
215 :  {
216 :      unsigned char sw;
217 :
218 :      sw  = ~P6DR;                        /* ディップスイッチ読み込み */
219 :      sw &= 0x0f;
220 :
```

```
221 :      return  sw;
222 :  }
223 :
224 :  /************************************************************************/
225 :  /* end of file                                                          */
226 :  /************************************************************************/
```

18.5 プログラム「pulseinstart.src」

```
  1 :  ;=====================================
  2 :  ; 定義
  3 :  ;=====================================
  4 :  RESERVE: .EQU    H'FFFFFFFF              ; 未使用領域のアドレス
  5 :
  6 :  ;=====================================
  7 :  ; 外部参照
  8 :  ;=====================================
  9 :          .IMPORT _main
 10 :          .IMPORT _interrupt_imia0
 11 :          .IMPORT _interrupt_imib0
 12 :          .IMPORT _interrupt_imia1
 13 :          .IMPORT _interrupt_imib1
 14 :          .IMPORT _interrupt_timer2
 15 :          .IMPORT _INITSCT
 16 :
 17 :  ;=====================================
 18 :  ; ベクタセクション
 19 :  ;=====================================
 20 :          .SECTION V
 21 :          .DATA.L RESET_START             ;  0 H'000000   ﾘｾｯﾄ
 22 :          .DATA.L RESERVE                 ;  1 H'000004   ｼｽﾃﾑ予約
 23 :          .DATA.L RESERVE                 ;  2 H'000008   ｼｽﾃﾑ予約
 24 :          .DATA.L RESERVE                 ;  3 H'00000c   ｼｽﾃﾑ予約
 25 :          .DATA.L RESERVE                 ;  4 H'000010   ｼｽﾃﾑ予約
 26 :          .DATA.L RESERVE                 ;  5 H'000014   ｼｽﾃﾑ予約
 27 :          .DATA.L RESERVE                 ;  6 H'000018   ｼｽﾃﾑ予約
 28 :          .DATA.L RESERVE                 ;  7 H'00001c   外部割り込み NMI
 29 :          .DATA.L RESERVE                 ;  8 H'000020   ﾄﾗｯﾌﾟ命令
 30 :          .DATA.L RESERVE                 ;  9 H'000024   ﾄﾗｯﾌﾟ命令
 31 :          .DATA.L RESERVE                 ; 10 H'000028   ﾄﾗｯﾌﾟ命令
 32 :          .DATA.L RESERVE                 ; 11 H'00002c   ﾄﾗｯﾌﾟ命令
 33 :          .DATA.L RESERVE                 ; 12 H'000030   外部割り込み IRQ0
 34 :          .DATA.L RESERVE                 ; 13 H'000034   外部割り込み IRQ1
 35 :          .DATA.L RESERVE                 ; 14 H'000038   外部割り込み IRQ2
 36 :          .DATA.L RESERVE                 ; 15 H'00003c   外部割り込み IRQ3
 37 :          .DATA.L RESERVE                 ; 16 H'000040   外部割り込み IRQ4
 38 :          .DATA.L RESERVE                 ; 17 H'000044   外部割り込み IRQ5
 39 :          .DATA.L RESERVE                 ; 18 H'000048   ｼｽﾃﾑ予約
 40 :          .DATA.L RESERVE                 ; 19 H'00004c   ｼｽﾃﾑ予約
 41 :          .DATA.L RESERVE                 ; 20 H'000050   WDT MOVI
 42 :          .DATA.L RESERVE                 ; 21 H'000054   REF CMI
 43 :          .DATA.L RESERVE                 ; 22 H'000058   ｼｽﾃﾑ予約
 44 :          .DATA.L RESERVE                 ; 23 H'00005c   ｼｽﾃﾑ予約
 45 :          .DATA.L _interrupt_imia0        ; 24 h'000060   ITU0 IMIA0
 46 :          .DATA.L _interrupt_imib0        ; 25 h'000064   ITU0 IMIB0
 47 :          .DATA.L RESERVE                 ; 26 H'000068   ITU0 OVI0
 48 :          .DATA.L RESERVE                 ; 27 H'00006c   ｼｽﾃﾑ予約
 49 :          .DATA.L _interrupt_imia1        ; 28 h'000070   ITU1 IMIA1
 50 :          .DATA.L _interrupt_imib1        ; 29 h'000074   ITU1 IMIB1
 51 :          .DATA.L RESERVE                 ; 30 H'000078   ITU1 OVI1
 52 :          .DATA.L RESERVE                 ; 31 H'00007c   ｼｽﾃﾑ予約
 53 :          .DATA.L _interrupt_timer2       ; 32 h'000080   ITU2 IMIA2
 54 :          .DATA.L RESERVE                 ; 33 H'000084   ITU2 IMIB2
 55 :          .DATA.L RESERVE                 ; 34 H'000088   ITU2 OVI2
 56 :          .DATA.L RESERVE                 ; 35 H'00008c   ｼｽﾃﾑ予約
 57 :          .DATA.L RESERVE                 ; 36 H'000090   ITU3 IMIA3
 58 :          .DATA.L RESERVE                 ; 37 H'000094   ITU3 IMIB3
 59 :          .DATA.L RESERVE                 ; 38 H'000098   ITU3 OVI3
 60 :          .DATA.L RESERVE                 ; 39 H'00009c   ｼｽﾃﾑ予約
 61 :          .DATA.L RESERVE                 ; 40 H'0000a0   ITU4 IMIA4
 62 :          .DATA.L RESERVE                 ; 41 H'0000a4   ITU4 IMIB4
 63 :          .DATA.L RESERVE                 ; 42 H'0000a8   ITU4 OVI4
 64 :          .DATA.L RESERVE                 ; 43 H'0000ac   ｼｽﾃﾑ予約
 65 :          .DATA.L RESERVE                 ; 44 H'0000b0   DMAC DEND0A
 66 :          .DATA.L RESERVE                 ; 45 H'0000b4   DMAC DEND0B
 67 :          .DATA.L RESERVE                 ; 46 H'0000b8   DMAC DEND1A
 68 :          .DATA.L RESERVE                 ; 47 H'0000bc   DMCA DEND1B
 69 :          .DATA.L RESERVE                 ; 48 H'0000c0   ｼｽﾃﾑ予約
 70 :          .DATA.L RESERVE                 ; 49 H'0000c4   ｼｽﾃﾑ予約
 71 :          .DATA.L RESERVE                 ; 50 H'0000c8   ｼｽﾃﾑ予約
 72 :          .DATA.L RESERVE                 ; 51 H'0000cc   ｼｽﾃﾑ予約
 73 :          .DATA.L RESERVE                 ; 52 H'0000d0   SCI0 ERI0
 74 :          .DATA.L RESERVE                 ; 53 H'0000d4   SCI0 RXI0
 75 :          .DATA.L RESERVE                 ; 54 H'0000d8   SCI0 TXI0
 76 :          .DATA.L RESERVE                 ; 55 H'0000dc   SCI0 TEI0
 77 :          .DATA.L RESERVE                 ; 56 H'0000e0   SCI1 ERI1
```

```
78 :          .DATA.L RESERVE                ; 57 H'0000e4   SCI1 RXI1
79 :          .DATA.L RESERVE                ; 58 H'0000e8   SCI1 TXI1
80 :          .DATA.L RESERVE                ; 59 H'0000ec   SCI1 TEI1
81 :          .DATA.L RESERVE                ; 60 H'0000f0   A/D ADI
82 :
83 :  ;===================================
84 :  ; スタートアッププログラム
85 :  ;===================================
86 :          .SECTION P
87 :  RESET_START:
88 :          MOV.L   #H'FFF10,ER7           ; スタックの設定
89 :          JSR     @_INITSCT              ; RAMエリアの初期化
90 :          JSR     @_main                 ; Ｃ言語のmain()関数へジャンプ
91 :  OWARI:
92 :          BRA     OWARI
93 :
94 :          .END
```

18.6 プログラムの解説

18.6.1 パルス幅を測定する設定

```
103 :       /* ITU0 インプットキャプチャ入力 */
104 :       ITU0_TCR  = 0x00;                  /* CNT のカウントはφ          */
105 :       ITU0_TIOR = 0x77;                  /* TIOCA0=input TIOCB0=input */
106 :       ITU0_IER  = 0x03;                  /* IMIA,IMIB 許可             */
107 :
108 :       /* ITU1 インプットキャプチャ入力 */
109 :       ITU1_TCR  = 0x00;                  /* CNT のカウントはφ          */
110 :       ITU1_TIOR = 0x77;                  /* TIOCA1=input TIOCB1=input */
111 :       ITU1_IER  = 0x03;                  /* IMIA,IMIB 許可             */
```

パルス幅を測定するための設定です。TIOR というレジスタを設定します。

●TU0_TIOR(タイマ I/O コントロールレジスタ)の設定内容

ビット:	7	6	5	4	3	2	1	0
ITU0_TIOR:	－	IOB2	IOB1	IOB0	－	IOA2	IOA1	IOA0
設定値:	0	1	1	1	0	1	1	1
16 進数:	7				7			

・ビット 6-4:I/O コントロール B2～0

GRB の機能を選択します。

IOB2	IOB1	IOB0	説明	
0	0	0	GRB はアウトプットコンペアレジスタ	コンペアマッチによる端子出力禁止(初期値)
0	0	1		GRB のコンペアマッチで0出力
0	1	0		GRB のコンペアマッチで1出力
0	1	1		GRB のコンペアマッチでトグル出力 (チャネル2のみ1出力)
1	0	0	GRB はインプットキャプチャレジスタ	立ち上がりエッジで GRB へインプットキャプチャ
1	0	1		立ち下がりエッジで GRB へインプットキャプチャ
1 **1**	1 **1**	0 **1**		立ち上がり／立ち下がりの両エッジでインプットキャプチャ

・ビット 2-0:I/O コントロール A2～0
GRA 機能を選択します。

IOA2	IOA1	IOA0	説明	
0	0	0	GRA はアウトプットコンペアレジスタ	コンペアマッチによる端子出力禁止(初期値)
0	0	1		GRA のコンペアマッチで0出力
0	1	0		GRA のコンペアマッチで1出力
0	1	1		GRA のコンペアマッチでトグル出力 (チャネル2のみ1出力)
1	0	0	GRA はインプットキャプチャレジスタ	立ち上がりエッジで GRA へインプットキャプチャ
1	0	1		立ち下がりエッジで GRA へインプットキャプチャ
1 **1**	1 **1**	0 **1**		立ち上がり／立ち下がりの両エッジでインプットキャプチャ

TIOR レジスタの設定により、下記端子が出力用になるか、入力用になるか変わります。使用する ITU のチャネルにより、端子が変わります。今回は ITU0 と 1 を使用するため、PA2～5 がパルス幅測定用の入力端子となります。ITU は 5 チャネルありますので、最大で 10 信号分のパルス幅を測定することができますが、通常のタイマや PWM が使えなくなりますので、うまく考えて選ぶ必要があります。

名称	端子	説明
TIOCA0	PA2	ITU0 の GRA で使用する端子
TIOCB0	PA3	ITU0 の GRB で使用する端子
TIOCA1	PA4	ITU1 の GRA で使用する端子
TIOCB1	PA5	ITU1 の GRB で使用する端子
TIOCA2	PA6	ITU2 の GRA で使用する端子
TIOCB2	PA7	ITU2 の GRB で使用する端子
TIOCA3	PB0	ITU3 の GRA で使用する端子
TIOCB3	PB1	ITU3 の GRB で使用する端子
TIOCA4	PB2	ITU4 の GRA で使用する端子
TIOCB4	PB3	ITU4 の GRB で使用する端子

105 行で、ITU0_TIOR の設定を行っています。

```
105 :       ITU0_TIOR = 0x77;                    /* TIOCA0=input TIOCB0=input */
```

bit6-4="111": GRB はインプットキャプチャレジスタ
立ち上がり／立ち下がりの両エッジでインプットキャプチャ
bit2-0="111": GRA はインプットキャプチャレジスタ
立ち上がり／立ち下がりの両エッジでインプットキャプチャ

横文字ばかりで意味が分からないと思います。言い換えると、インプットキャプチャレジスタとは、信号が切り替わったときに CNT の現在値を取り込むレジスタという意味です。信号が切り替わったときとは、今回"111"に設定していますので、先の説明から「立ち上がり／立ち下がりの両エッジでインプットキャプチャ」ということになります。ITU0_GRA で例えると、TIOCA0(PA2)端子に立ち上がり信号("0"から"1"に変わった瞬間)と立ち下がり信号("1"から"0"に変わった瞬間)が入力されたとき、ITU0_CNT の値を ITU0_GRA へコピーします。
ITU0_GRA、GRB 共に両エッジでインプットキャプチャするよう設定しています。

106 行で割り込みの設定をしています。

```
106 :       ITU0_IER  = 0x03;                   /* IMIA,IMIB 許可            */
```

bit0="1":IMIA を許可

にしています。これは、ITU0_CNT の値が ITU0_GRA へコピーされたとき、割り込みを発生させるかさせないかの設定です。割り込みを発生させるようにしています。コピーされるとき、それは先ほどの設定のとおり、PA2 端子が立ち上がり、または立ち下がりの信号を検出したときです。「pulseinstart.src」のベクタ番号24を「_interrupt_imia0」にします。割り込みが発生すると、interrupt_imia0 関数へ実行を移します。

```
45 :          .DATA.L _interrupt_imia0         ; 24 h'000060    ITU0 IMIA0
```

※pulseinstart.src ファイル内の 45 行目です。

割り込みプログラム内で、今の ITU0_GRA の値と1回前の ITU0_GRA の値を引くと、差分がでます。

ITU0_TCR の設定は 0x00 なので、ITU0_CNT のカウントはφの時間で1つ進みます。計算すると、

$$1／\phi$$
$$=1／(24.576\times10^{6})$$
$$=40.69[ns]$$

となります。また、

bit1="1":IMIB を許可

ということで、IMIB も許可していますので、同様に両エッジで ITU0_CNT の値が ITU0_GRB にコピーされると同時に割り込みが発生します。

18.6.2 interrupt_imia0 関数（割り込みプログラム）

```
129 :  #pragma interrupt( interrupt_imia0 )
130 :  void interrupt_imia0( void )
131 :  {
132 :      ITU0_TSR &= 0xfe;                   /* フラグクリア              */
133 :      if( !(PADR & 0x04) ) {
134 :          /* "1"なら */
135 :          uPulseWidth0 = ITU0_GRA - uPulseWidth0Before;
136 :          uPulseWidth0 -= 8;              /* プログラム実行分の時間を引く */
137 :          flag |= 0x01;
138 :      }
139 :      uPulseWidth0Before = ITU0_GRA;
140 :  }
```

割り込みプログラム内では現在の IUT0_GRA 値と、1回前に発生した割り込みの ITU0_GRA 値の差分を計算してパルスの ON 幅を計算します。差分は uPulseWidth0 という大域変数に代入され、メインプログラム内でこの変数を使って様々な処理をします。

133 行で、端子が"1"か"0"かチェックしています。両エッジで割り込みが発生しますので、"0"→"1"の割り込みか、"1"→"0"の割り込みか分かりません。そこで直接 PA2 端子をチェックしてどちらの割り込みか判断します。ON 幅を測定しますので"0"かどうかチェックしています("1"→"0"での割り込みのため)。

135 行で、(現在の GRA 値)－(過去の GRA 値)を計算、uPulseWidth0 変数に代入します。過去の ITU0_GRA 値とは、uPulseWidth0Before 変数のことです。uPulseWidth0 の値1つで、40.69[ns]のパルス幅になります。もし、uPulseWidth0 が 1000 なら、40.69[μs]のパルス幅となります。

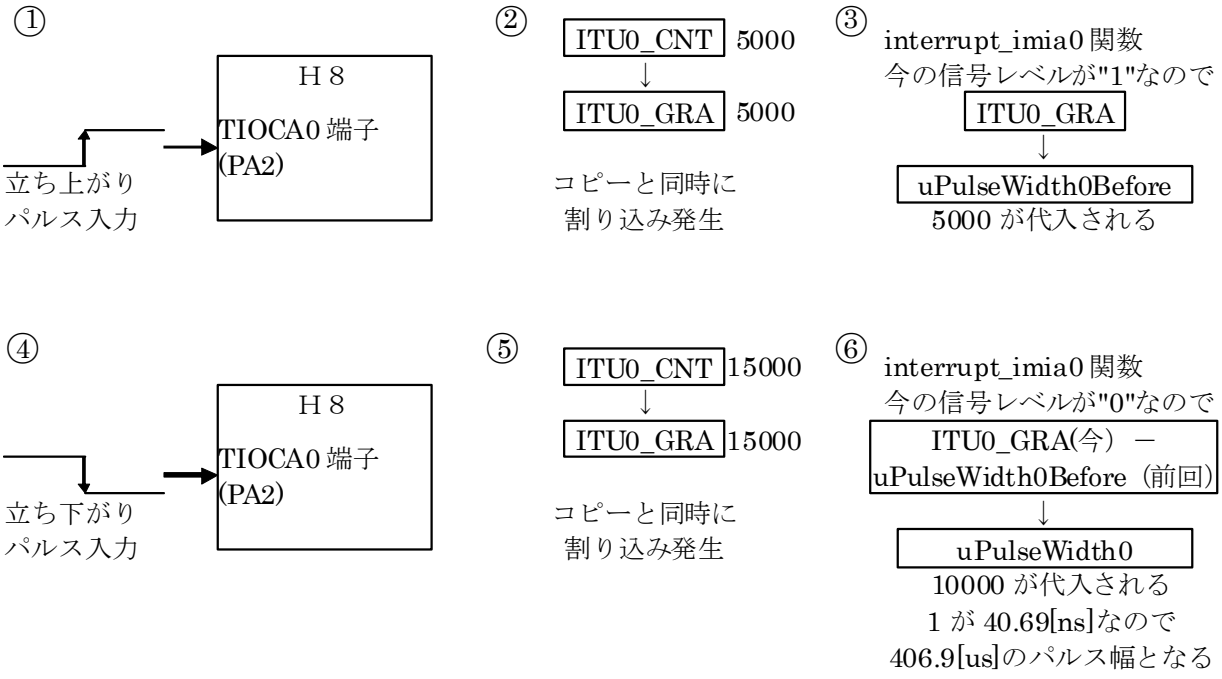
136 行で、計算した結果から 8 引いています。これは、立ち上がり、または立ち下がりエッジを検出して割り込みが発生、計算するまで、8 つ分の時間がかかってしまうためです。実際に入力したパルス幅と uPulseWidth0 の値を比較して、8 と割り出しました。

139 行で、ITU0_GRA の値を uPulseWidth0Before 変数に代入します。次に割り込みが発生したとき、この変

数の値が前回の値となります。
他の ITU0_GRB、ITU1_GRA、ITU1_GRB も同様の動作です。

18.7 まとめ

ITU0_GRA（PA2 端子）を例にまとめると、下図のようになります。



19. プロジェクト「beep2」 ブザーを鳴らす

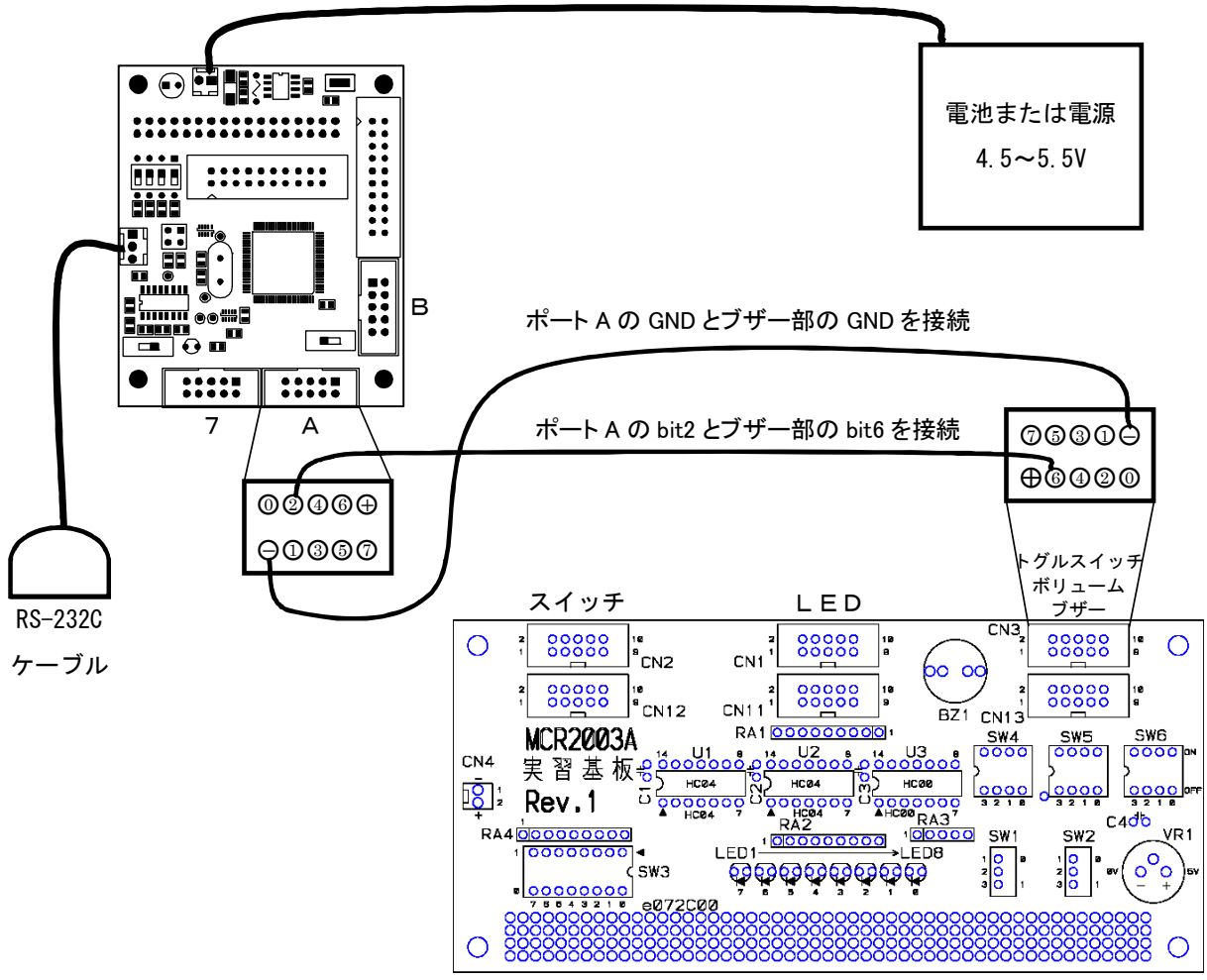
19.1 概要

PWM 機能を使って、圧電ブザーの音を鳴らします。マイコンのポートは、下記を使用します。
・ポート A のビット 2・・・圧電ブザーやアンプなど

19.2 接続

19.2.1 実習基板を使用

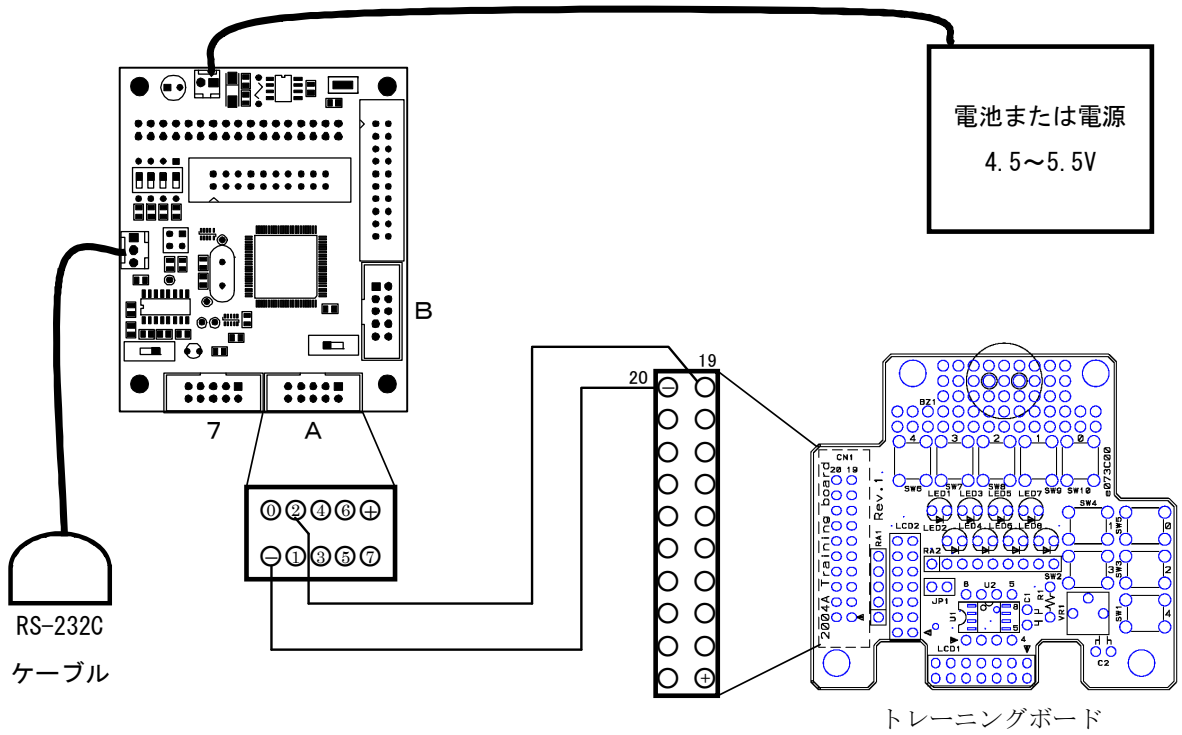
・CPU ボードのポート A のビット 2(J2 コネクタの 7 ピン)と、ブザー部のビット 6 をピンコードで接続します。
・CPU ボードのポート A の GND(J2 コネクタの 10 ピン)と、ブザー部の GND をピンコードで接続します。



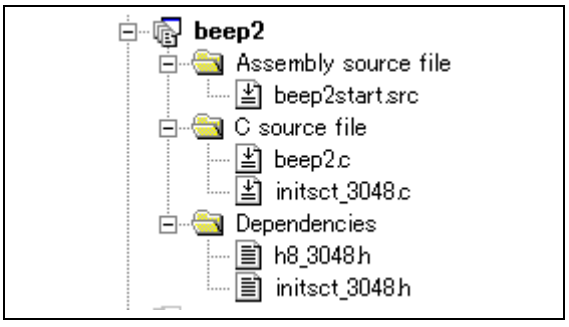
19.2.2 トレーニングボード(MCR2004A基板)を使用

トレーニングボード(MCR2004A 基板)には、ブザーが付いています。それを利用して実習する場合の結線方法を、下図に示します。

・CPU ボードのポート A のビット 2(J2 コネクタの 7 ピン)に、トレーニングボードの 19 ピンを接続します。
・CPU ボードのポート A の GND(J2 コネクタの 10 ピン)に、トレーニングボードの 20 ピンを接続します。



19.3 プロジェクトの構成



	ファイル名	内容
1	beep2start.src	アセンブリ言語で記述されたアセンブリソースファイルです。このファイルの構造は下記のようになっています。 ベクタアドレス ＋ スタートアップルーチン
2	initsct_3048.c	初期値のないグローバル変数(セクション B 領域)、初期値のあるグローバル変数(セクション R 領域)の初期化用です。
3	beep2.c	実際に制御するプログラムが書かれています。H8/3048F-ONE の内蔵周辺機能の初期化も行います。

4	h8_3048.h	H8/3048F-ONE の内蔵周辺機能の I/O レジスタを定義したファイルです。
5	initsct_3048.h	initsct_3048.c のヘッダファイルです。

19.4 プログラム「beep2.c」

```
 1 : /*********************************************************************/
 2 : /* ブザーを鳴らす(ITU0のPWM使用)「beep2.c」                           */
 3 : /* 出力：PA2(圧電ブザー、アンプ等)                                    */
 4 : /*                          2005.04 ジャパンマイコンカーラリー実行委員会  */
 5 : /*********************************************************************/
 6 : /*====================================*/
 7 : /* インクルード                       */
 8 : /*====================================*/
 9 : #include    <machine.h>
10 : #include    "h8_3048.h"
11 :
12 : /*====================================*/
13 : /* シンボル定義                       */
14 : /*====================================*/
15 :
16 : /* 定数設定 */
17 : #define         TIMER_CYCLE     3071    /* タイマのサイクル 1ms     */
18 :                                         /* φ/8で使用する場合、     */
19 :                                         /* φ/8 = 325.5[ns]          */
20 :                                         /* ∴TIMER_CYCLE =           */
21 :                                         /*      1[ms] / 325.5[ns]   */
22 :                                         /*                = 3072    */
23 :
24 : #define         M_DO        11739       /* ド                       */
25 : #define         M_DOU       11080       /* ド#                      */
26 : #define         M_RE        10458       /* レ                       */
27 : #define         M_REU       9871        /* レ#                      */
28 : #define         M_MI        9317        /* ミ                       */
29 : #define         M_FA        8793        /* ファ                     */
30 : #define         M_FAU       8300        /* ファ#                    */
31 : #define         M_SO        7834        /* ソ                       */
32 : #define         M_SOU       7394        /* ソ#                      */
33 : #define         M_RA        6979        /* ラ                       */
34 : #define         M_RAU       6587        /* ラ#                      */
35 : #define         M_SI        6217        /* シ                       */
36 : #define         H1_DO       5868        /* ド                       */
37 :
38 : /*====================================*/
39 : /* プロトタイプ宣言                   */
40 : /*====================================*/
41 : void init( void );
42 : void timer( unsigned long timer_set );
43 : unsigned char dipsw_get( void );
44 : void note(unsigned int length,unsigned int tone);
45 :
46 : /*====================================*/
47 : /* グローバル変数の宣言               */
48 : /*====================================*/
49 : unsigned long   cnt0;                  /* ITU3用                   */
50 :
51 : /*********************************************************************/
52 : /* メインプログラム                                                  */
53 : /*********************************************************************/
54 : void main( void )
55 : {
56 :     init();                             /* マイコン機能の初期化     */
57 :     set_ccr( 0x00 );                    /* 全体割り込み許可         */
58 :
59 :     while( 1 ) {
60 :         note( 1000, M_DO  );
61 :         note( 1000, M_RE  );
62 :         note( 1000, M_MI  );
63 :         note( 1000, M_FA  );
64 :         note( 1000, M_SO  );
65 :         note( 1000, M_RA  );
66 :         note( 1000, M_SI  );
67 :         note( 1000, H1_DO );
68 :         note( 1000, 0     );
69 :     }
70 : }
71 :
72 : /*********************************************************************/
73 : /* 音を鳴らす                                                        */
74 : /* 引数  長さ[ms]                                                    */
75 : /*       トーンのPWM値                                               */
76 : /*********************************************************************/
```

```
 77 :  void note(unsigned int length,unsigned int tone)
 78 :  {
 79 :      if( tone ) {
 80 :          ITU_STR &= 0xfe;                /* PWM停止                  */
 81 :          ITU0_CNT = tone - 1;
 82 :          ITU0_GRA = tone;                /* トーン設定               */
 83 :          ITU0_GRB = tone / 2;            /* デューティ比は５０％     */
 84 :          ITU_STR |= 0x01;                /* PWM開始                  */
 85 :      }
 86 :      timer( length );                    /* 時間カウント             */
 87 :
 88 :      /* 出力OFF */
 89 :      ITU_STR &= 0xfe;
 90 :      ITU0_CNT = 0;
 91 :      ITU0_GRA = 1;
 92 :      ITU0_GRB = 2;
 93 :      ITU_STR |= 0x01;
 94 :  }
 95 :
 96 :  /************************************************************************/
 97 :  /* H8/3048F内蔵モジュール　初期化                                       */
 98 :  /************************************************************************/
 99 :  void init( void )
100 :  {
101 :      /* ポートの入出力設定 */
102 :      P1DDR = 0xff;
103 :      P2DDR = 0xff;
104 :      P3DDR = 0xff;
105 :      P4DDR = 0xff;
106 :      P5DDR = 0xff;
107 :      P6DDR = 0xf0;                       /* CPU基板上のDIP SW        */
108 :      P8DDR = 0xff;
109 :      P9DDR = 0xf7;                       /* 通信ポート               */
110 :      PADDR = 0xff;                       /* bit2:圧電ブザー          */
111 :      PBDDR = 0xff;
112 :      /* ポート７は、入力専用なので入出力設定はありません　*/
113 :
114 :      /* ITU0 ブザー用PWM */
115 :      ITU0_TCR = 0x23;
116 :      ITU0_GRA = 1;
117 :      ITU0_GRB = 2;
118 :      ITU0_CNT = 0;
119 :      ITU_MDR |= 0x01;                    /* PWMモード設定            */
120 :
121 :      /* ITU3 1ms毎の割り込み　timer関数用 */
122 :      ITU3_TCR = 0x23;
123 :      ITU3_GRA = TIMER_CYCLE;
124 :      ITU3_CNT = 0;
125 :      ITU3_IER = 0x01;                    /* 割り込み許可             */
126 :
127 :      ITU_STR = 0x09;                     /* ITUのカウントスタート    */
128 :  }
129 :
130 :  /************************************************************************/
131 :  /* ITU3 割り込み処理                                                    */
132 :  /************************************************************************/
133 :  #pragma interrupt( interrupt_timer3 )
134 :  void interrupt_timer3( void )
135 :  {
136 :      ITU3_TSR &= 0xfe;                   /* フラグクリア             */
137 :      cnt0++;
138 :  }
139 :
140 :  /************************************************************************/
141 :  /* タイマ本体                                                           */
142 :  /* 引数　タイマ値 1=1ms                                                 */
143 :  /************************************************************************/
144 :  void timer( unsigned long timer_set )
145 :  {
146 :      cnt0    = 0;
147 :      while( cnt0 < timer_set );
148 :  }
149 :
150 :  /************************************************************************/
151 :  /* ディップスイッチ値読み込み                                           */
152 :  /* 戻り値 スイッチ値 0～15                                              */
153 :  /************************************************************************/
154 :  unsigned char dipsw_get( void )
155 :  {
156 :      unsigned char sw;
157 :
158 :      sw  = ~P6DR;                        /* ディップスイッチ読み込み */
159 :      sw &= 0x0f;
160 :
161 :      return  sw;
162 :  }
163 :
164 :  /************************************************************************/
165 :  /* end of file                                                          */
166 :  /************************************************************************/
```

19.5 プログラムの解説

19.5.1 音階とは

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%の周期の PWM を圧電スピーカに出力すれば、「ドレミファソラシド」と音を鳴らすことができます。

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

261.6×2 の(x／12)乗

で求められます。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。

音	x	計算	周波数[Hz]	ITU0_GRA の値
ド	0	$261.6\times2^{(0/12)}$	261.6	11739
ド＃	1	$261.6\times2^{(1/12)}$	277.2	11080
レ	2	$261.6\times2^{(2/12)}$	293.6	10458
レ＃	3	$261.6\times2^{(3/12)}$	311.1	9871
ミ	4	$261.6\times2^{(4/12)}$	329.6	9317
ファ	5	$261.6\times2^{(5/12)}$	349.2	8793
ファ＃	6	$261.6\times2^{(6/12)}$	370.0	8300
ソ	7	$261.6\times2^{(7/12)}$	392.0	7834
ソ＃	8	$261.6\times2^{(8/12)}$	415.3	7394
ラ	9	$261.6\times2^{(9/12)}$	440.0	6979
ラ＃	10	$261.6\times2^{(10/12)}$	466.1	6587
シ	11	$261.6\times2^{(11/12)}$	493.8	6217
ド	12	$261.6\times2^{(12/12)}$	523.2	5868

「ド」を例に ITU0_GRA の値を計算すると、表より周期は 261.6[Hz]ですので

周期＝1／周波数＝1/261.6＝3.823[ms]

となります。ITU0_CNT は一つ値が増える時間は、325.52[ns]ですので、「ド」の周期分のカウント値は、

$3.823\times10^{-3}\div325.52\times10^{-9}=11740$

となります。ITU0_GRA に設定する値は1小さい値にするので最終的には、11739 を代入します。すべて計算すると、上表のようになります。

19.5.2 音階のdefine定義

先ほどの計算値を define で定義して、プログラム中で使いやすくします。

```
24 :  #define        M_DO      11739      /* ド                  */
25 :  #define        M_DOU     11080      /* ド＃                */
26 :  #define        M_RE      10458      /* レ                  */
27 :  #define        M_REU     9871       /* レ＃                */
28 :  #define        M_MI      9317       /* ミ                  */
29 :  #define        M_FA      8793       /* ファ                */
30 :  #define        M_FAU     8300       /* ファ＃              */
31 :  #define        M_SO      7834       /* ソ                  */
32 :  #define        M_SOU     7394       /* ソ＃                */
33 :  #define        M_RA      6979       /* ラ                  */
34 :  #define        M_RAU     6587       /* ラ＃                */
35 :  #define        M_SI      6217       /* シ                  */
36 :  #define        H1_DO     5868       /* ド                  */
```

19.5.3 音を鳴らすnote関数

音を鳴らす note 関数を作りました。引数に、鳴らす時間[ms]、音階の PWM 値を設定します。

```
        note( 鳴らす時間[ms] , 音階の PWM 値 );
```

例えば、「レ」を 0.3 秒鳴らす場合は、

```
        note( 300 , M_RE );
```

とします。

サンプルプログラムでは、「ド・レ・ミ・ファ・ソ・ラ・シ・ド」を 1 秒ごとに鳴らします。note 関数を使って、作曲してみましょう。

20. プロジェクト「beep_haru」 ブザーを鳴らす応用「春の小川」を演奏

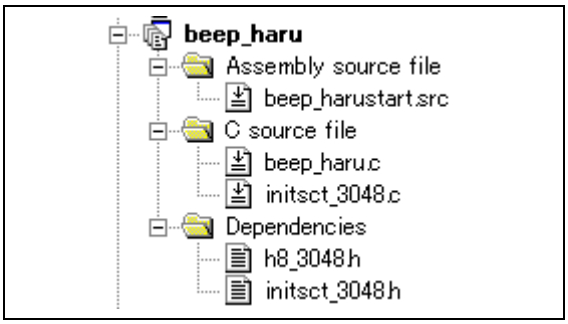
20.1 概要

プロジェクト「beep2」の応用です。春の小川を演奏します。1ms ごとの割り込みを追加しています。割り込みの詳細は、プロジェクト「timer2」を参照してください。演奏処理は、すべて割り込みプログラム内で行います。

20.2 接続

プロジェクト「beep2」と同様です。

20.3 プロジェクトの構成



	ファイル名	内容
1	beep_harustart.src	アセンブリ言語で記述されたアセンブリソースファイルです。このファイルの構造は下記のようになっています。 ベクタアドレス ＋ スタートアップルーチン
2	initsct_3048.c	初期値のないグローバル変数(セクション B 領域)、初期値のあるグローバル変数(セクション R 領域)の初期化用です。
3	beep_haru.c	実際に制御するプログラムが書かれています。H8/3048F-ONE の内蔵周辺機能の初期化も行います。
4	h8_3048.h	H8/3048F-ONE の内蔵周辺機能の I/O レジスタを定義したファイルです。
5	initsct_3048.h	initsct_3048.c のヘッダファイルです。

20.4 プログラム「beep_haru.c」

```
 1 :  /*********************************************************************/
 2 :  /* 音階出力 春が来た版 "beep_haru.c"                                 */
 3 :  /* 出力：PA2(圧電ブザー、アンプ等)                                   */
 4 :  /*                          2008.09 ジャパンマイコンカーラリー実行委員会  */
 5 :  /*********************************************************************/
 6 :  /*====================================*/
 7 :  /* インクルード                       */
 8 :  /*====================================*/
 9 :  #include    <machine.h>
10 :  #include    "h8_3048.h"
```

```
 11 :
 12 :  /*====================================*/
 13 :  /* シンボル定義                       */
 14 :  /*====================================*/
 15 :
 16 :  /* 定数設定 */
 17 :  #define         TIMER_CYCLE     3071    /* タイマのサイクル 1ms     */
 18 :                                          /* φ/8で使用する場合、      */
 19 :                                          /* φ/8 = 325.5[ns]          */
 20 :                                          /* ∴TIMER_CYCLE =           */
 21 :                                          /*      1[ms] / 325.5[ns]   */
 22 :                                          /*                = 3072    */
 23 :
 24 :  /* 低い音階１ */
 25 :  #define         L1_DO       23486       /* ド                       */
 26 :  #define         L1_DOU      22168       /* ド＃                     */
 27 :  #define         L1_RE       20924       /* レ                       */
 28 :  #define         L1_REU      19750       /* レ＃                     */
 29 :  #define         L1_MI       18641       /* ミ                       */
 30 :  #define         L1_FA       17595       /* ファ                     */
 31 :  #define         L1_FAU      16607       /* ファ＃                   */
 32 :  #define         L1_SO       15675       /* ソ                       */
 33 :  #define         L1_SOU      14795       /* ソ＃                     */
 34 :  #define         L1_RA       13965       /* ラ                       */
 35 :  #define         L1_RAU      13181       /* ラ＃                     */
 36 :  #define         L1_SI       12441       /* シ                       */
 37 :
 38 :  /* 標準の音階 */
 39 :  #define         M_DO        11743       /* ド                       */
 40 :  #define         M_DOU       11084       /* ド＃                     */
 41 :  #define         M_RE        10462       /* レ                       */
 42 :  #define         M_REU       9875        /* レ＃                     */
 43 :  #define         M_MI        9321        /* ミ                       */
 44 :  #define         M_FA        8797        /* ファ                     */
 45 :  #define         M_FAU       8304        /* ファ＃                   */
 46 :  #define         M_SO        7838        /* ソ                       */
 47 :  #define         M_SOU       7398        /* ソ＃                     */
 48 :  #define         M_RA        6983        /* ラ                       */
 49 :  #define         M_RAU       6591        /* ラ＃                     */
 50 :  #define         M_SI        6221        /* シ                       */
 51 :
 52 :  /* 高い音階１ */
 53 :  #define         H1_DO       5872        /* ド                       */
 54 :  #define         H1_DOU      5542        /* ド＃                     */
 55 :  #define         H1_RE       5231        /* レ                       */
 56 :  #define         H1_REU      4937        /* レ＃                     */
 57 :  #define         H1_MI       4660        /* ミ                       */
 58 :  #define         H1_FA       4399        /* ファ                     */
 59 :  #define         H1_FAU      4152        /* ファ＃                   */
 60 :  #define         H1_SO       3919        /* ソ                       */
 61 :  #define         H1_SOU      3699        /* ソ＃                     */
 62 :  #define         H1_RA       3491        /* ラ                       */
 63 :  #define         H1_RAU      3295        /* ラ＃                     */
 64 :  #define         H1_SI       3110        /* シ                       */
 65 :
 66 :  #define         TEMPO       60          /* テンポ                   */
 67 :
 68 :  /*====================================*/
 69 :  /* プロトタイプ宣言                   */
 70 :  /*====================================*/
 71 :  void init( void );
 72 :  void timer( unsigned long timer_set );
 73 :  unsigned char dipsw_get( void );
 74 :  void note( unsigned int tone );
 75 :
 76 :  /*====================================*/
 77 :  /* グローバル変数の宣言               */
 78 :  /*====================================*/
 79 :  unsigned long   cnt0;                   /* timer用                  */
 80 :  unsigned long   cnt1;                   /* 音楽用                   */
 81 :  int             music_dim;              /* 音楽データ配列の位置     */
 82 :  int             music_flag;             /* 音楽データならすかどうか */
 83 :
 84 :  const int music_data[][2] = {           /* 春の小川 音楽データ      */
 85 :  /*
 86 :  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。
 87 :  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。
 88 :  */
 89 :  /*  音階,   長さ    */
 90 :      0,      0,  /* スタート*/
 91 :
 92 :      M_MI,   4,  /* は */
 93 :      M_SO,   4,  /* ー */
 94 :      M_RA,   4,  /* る */
 95 :      M_SO,   4,  /* の */
 96 :      M_MI,   4,  /* お */
 97 :      M_SO,   4,  /* が */
 98 :      H1_DO,  4,  /* わ */
 99 :      H1_DO,  4,  /* は */
100 :      M_RA,   4,  /* さ */
101 :      M_RA,   4,  /* ら */
```

```
102 :      M_SO,    4,  /* さ */
103 :      M_MI,    4,  /* ら */
104 :      M_DO,    4,  /* い */
105 :      M_RE,    4,  /* く */
106 :      M_MI,    4,  /* よ */
107 :      0,       4,
108 :
109 :      M_MI,    4,  /* き */
110 :      M_SO,    4,  /* ー */
111 :      M_RA,    4,  /* し */
112 :      M_SO,    4,  /* の */
113 :      M_MI,    4,  /* す */
114 :      M_SO,    4,  /* み */
115 :      H1_DO,   4,  /* れ */
116 :      H1_DO,   4,  /* や */
117 :      M_RA,    4,  /* れ */
118 :      M_RA,    4,  /* ん */
119 :      M_SO,    4,  /* げ */
120 :      M_MI,    4,  /* の */
121 :      M_RE,    4,  /* は */
122 :      M_MI,    4,  /* な */
123 :      M_DO,    4,  /* に */
124 :      0,       4,
125 :
126 :      M_RE,    4,  /* す */
127 :      M_MI,    4,  /* ー */
128 :      M_RE,    4,  /* が */
129 :      M_SO,    4,  /* た */
130 :      M_RA,    4,  /* や */
131 :      M_RA,    4,  /* さ */
132 :      M_SO,    4,  /* し */
133 :      M_RA,    4,  /* く */
134 :      H1_DO,   4,  /* い */
135 :      H1_DO,   4,  /* ろ */
136 :      M_SI,    4,  /* う */
137 :      M_RA,    4,  /* つ */
138 :      M_SO,    4,  /* く */
139 :      M_SO,    4,  /* し */
140 :      M_MI,    4,  /* く */
141 :      0,       4,
142 :
143 :      M_MI,    4,  /* さ */
144 :      M_SO,    4,  /* ー */
145 :      M_RA,    4,  /* い */
146 :      M_SO,    4,  /* て */
147 :      M_MI,    4,  /* い */
148 :      M_SO,    4,  /* る */
149 :      H1_DO,   4,  /* ね */
150 :      H1_DO,   4,  /* と */
151 :      M_RA,    4,  /* さ */
152 :      M_RA,    4,  /* さ */
153 :      M_SO,    4,  /* や */
154 :      M_MI,    4,  /* き */
155 :      M_RE,    4,  /* な */
156 :      M_MI,    4,  /* が */
157 :      M_DO,    4,  /* ら */
158 :      0,       4,
159 :
160 :      -1,      0   /* 終了 */
161 :  };
162 :
163 :  /************************************************************************/
164 :  /* メインプログラム                                                     */
165 :  /************************************************************************/
166 :  void main( void )
167 :  {
168 :      init();                          /* マイコン機能の初期化     */
169 :      set_ccr( 0x00 );                 /* 全体割り込み許可         */
170 :
171 :      music_flag = 1;                  /* 音楽スタート             */
172 :
173 :      while( 1 ) {
174 :      }
175 :  }
176 :
177 :  /************************************************************************/
178 :  /* 音を鳴らす                                                           */
179 :  /* 引数  長さ[ms]                                                       */
180 :  /*       トーンのPWM値                                                  */
181 :  /************************************************************************/
182 :  void note( unsigned int tone )
183 :  {
184 :      if( tone ) {
185 :          ITU_STR &= 0xfe;             /* PWM停止                  */
186 :          ITU0_CNT = tone - 1;
187 :          ITU0_GRA = tone;             /* トーン設定               */
188 :          ITU0_GRB = tone / 2;         /* デューティ比は５０％     */
189 :          ITU_STR |= 0x01;             /* PWM開始                  */
190 :      } else {
191 :          ITU_STR &= 0xfe;             /* PWM停止                  */
192 :          ITU0_CNT = 0;                /* 0%                       */
```

```
193 :          ITU0_GRA = 1;
194 :          ITU0_GRB = 2;
195 :          ITU_STR |= 0x01;                /* PWM開始                  */
196 :      }
197 :  }
198 :
199 :  /************************************************************************/
200 :  /* H8/3048F内蔵モジュール　初期化                                       */
201 :  /************************************************************************/
202 :  void init( void )
203 :  {
204 :      /* ポートの入出力設定 */
205 :      P1DDR = 0xff;
206 :      P2DDR = 0xff;
207 :      P3DDR = 0xff;
208 :      P4DDR = 0xff;
209 :      P5DDR = 0xff;
210 :      P6DDR = 0xf0;                       /* CPU基板上のDIP SW        */
211 :      P8DDR = 0xff;
212 :      P9DDR = 0xf7;                       /* 通信ポート               */
213 :      PADDR = 0xff;                       /* bit2:圧電ブザー          */
214 :      PBDDR = 0xff;
215 :      /* ポート７は、入力専用なので入出力設定はありません　*/
216 :
217 :      /* ITU0 ブザー用PWM */
218 :      ITU0_TCR = 0x23;
219 :      ITU0_GRA = 1;
220 :      ITU0_GRB = 2;
221 :      ITU0_CNT = 0;
222 :      ITU_MDR |= 0x01;                    /* PWMモード設定            */
223 :
224 :      /* ITU3 1ms毎の割り込み　timer関数用 */
225 :      ITU3_TCR = 0x23;
226 :      ITU3_GRA = TIMER_CYCLE;
227 :      ITU3_CNT = 0;
228 :      ITU3_IER = 0x01;                    /* 割り込み許可             */
229 :
230 :      ITU_STR = 0x09;                     /* ITUのカウントスタート    */
231 :  }
232 :
233 :  /************************************************************************/
234 :  /* ITU3 割り込み処理                                                    */
235 :  /************************************************************************/
236 :  #pragma interrupt( interrupt_timer3 )
237 :  void interrupt_timer3( void )
238 :  {
239 :      ITU3_TSR &= 0xfe;                   /* フラグクリア             */
240 :      cnt0++;
241 :      cnt1++;
242 :
243 :      if( music_flag ) {
244 :          /* 音階処理 */
245 :          if( cnt1 >= 15000L*music_data[music_dim][1]/TEMPO ) {
246 :              /* 次の音階をならす */
247 :              cnt1 = 0;
248 :              music_dim++;
249 :              if( music_data[music_dim][0] == -1 ) {
250 :                  /* -1なら終了 */
251 :                  note( 0 );
252 :                  music_flag = 0;
253 :              } else {
254 :                  /* -1でないなら次の音階セット */
255 :                  note( music_data[music_dim][0] );
256 :              }
257 :          }
258 :      }
259 :  }
260 :
261 :  /************************************************************************/
262 :  /* タイマ本体                                                           */
263 :  /* 引数　タイマ値 1=1ms                                                 */
264 :  /************************************************************************/
265 :  void timer( unsigned long timer_set )
266 :  {
267 :      cnt0     = 0;
268 :      while( cnt0 < timer_set );
269 :  }
270 :
271 :  /************************************************************************/
272 :  /* ディップスイッチ値読み込み                                           */
273 :  /* 戻り値 スイッチ値 0～15                                              */
274 :  /************************************************************************/
275 :  unsigned char dipsw_get( void )
276 :  {
277 :      unsigned char sw;
278 :
279 :      sw  = ~P6DR;                        /* ディップスイッチ読み込み */
280 :      sw &= 0x0f;
281 :
282 :      return  sw;
283 :  }
```

```
284 :
285 :  /************************************************************************/
286 :  /* end of file                                                           */
287 :  /************************************************************************/
```

20.5 プログラムの解説

20.5.1 シンボル定義

```
 24 :  /* 低い音階１ */
 25 :  #define          L1_DO       23486       /* ド                   */
 26 :  #define          L1_DOU      22168       /* ド＃                 */
中略
 38 :  /* 標準の音階 */
 39 :  #define          M_DO        11743       /* ド                   */
 40 :  #define          M_DOU       11084       /* ド＃                 */
中略
 52 :  /* 高い音階１ */
 53 :  #define          H1_DO       5872        /* ド                   */
 54 :  #define          H1_DOU      5542        /* ド＃                 */
中略
 66 :  #define          TEMPO       60          /* テンポ               */
```

標準の音階は、「medium」(中間)の意味で「M」を先頭に付けます。1つ低い音は「low」(低い)の意味で「L1」を先頭に付けます。1 つ高い音は「high」(高い)の意味で「H1」を付けます。

「TEMPO」定数にテンポを入力します。

20.5.2 音楽データ

配列「music_data」に、音楽データを登録します。二次元配列です。

```
const int music_data[][2] = {           /* 春の小川 音楽データ      */
    0,      0,  /* スタート*/  1
    M_MI,   4,  /* は */       2
    0,      4,                 3
    -1,     0   /* 終了 */     4
};
```

1…スタートです。

2…音楽データです。

1 個目は音階を入れます。周波数を入力すれば良いのですが、数値を直接入力すると後で見たときにドか、レか分かりづらいです。そのため、分かりやすくするため、定数宣言で定義した音階データを入力します。

2 個目は、長さを入れます。

16…全音符　8…二分音符　4…四分音符　2…八分音符　1…16 分音符

となります。

3…休符です。

1 個目は 0 を入れます。

2 個目は、長さを入れます。

16…全休符　8…二分休符　4…四分休符　2…八分休符　1…16 分休符

となります。

4…終了です。

20.5.3 割り込み

音を鳴らす処理は、すべて割り込みで行っています。

```
233 :  /************************************************************************/
234 :  /* ITU3 割り込み処理                                                    */
235 :  /************************************************************************/
236 :  #pragma interrupt( interrupt_timer3 )
237 :  void interrupt_timer3( void )
238 :  {
239 :      ITU3_TSR &= 0xfe;                   /* フラグクリア             */
240 :      cnt0++;
241 :      cnt1++;
242 :
243 :      if( music_flag ) {
244 :          /* 音階処理 */
245 :          if( cnt1 >= 15000L*music_data[music_dim][1]/TEMPO ) {
246 :              /* 次の音階をならす */
247 :              cnt1 = 0;
248 :              music_dim++;
249 :              if( music_data[music_dim][0] == -1 ) {
250 :                  /* -1 なら終了 */
251 :                  note( 0 );
252 :                  music_flag = 0;
253 :              } else {
254 :                  /* -1 でないなら次の音階セット */
255 :                  note( music_data[music_dim][0] );
256 :              }
257 :          }
258 :      }
259 :  }
```

music_flag が 0 なら音楽を鳴らすプログラムを実行しません。0 以外なら音楽を鳴らす処理を行います

現在の音を鳴らす時間は終わったかチェック

音階データが-1 なら演奏終了

音は鳴らさない

0 にして演奏終了

新しい音階を鳴らす

20.5.4 main関数

演奏作業はすべて割り込みないで行っているため、main 関数では、演奏開始の合図以外、何もしていません。while 文中に何か制御する命令を追加すれば、音楽を鳴らしながら処理を平行してできます。これは一定時間ごとの割り込み(今回は ITU を使用して 1ms ごとに割り込みを発生)と ITU によるパルス出力を使わなければできません。

```
166 :  void main( void )
167 :  {
168 :      init();                             /* マイコン機能の初期化     */
169 :      set_ccr( 0x00 );                    /* 全体割り込み許可         */
170 :
171 :      music_flag = 1;                     /* 音楽スタート             */
172 :
173 :      while( 1 ) {
174 :      }
175 :  }
```

演奏開始

無限ループで何もしない

21. プロジェクト「beep_clock」 ブザーを鳴らす応用「大きな古時計」を演奏

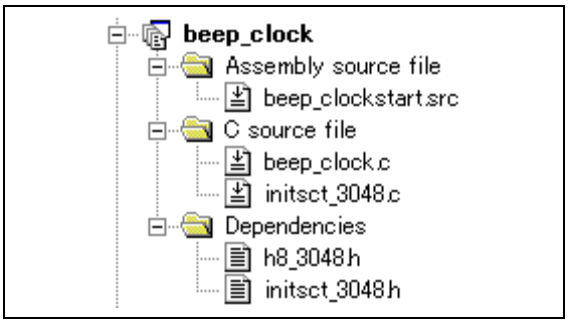
21.1 概要

プロジェクト「beep」の応用です。大きな古時計を演奏します。プロジェクト「beep_haru」の、音楽データのみ替えています。

21.2 接続

プロジェクト「beep2」と同様です。

21.3 プロジェクトの構成



	ファイル名	内容
1	beep_clockstart.src	アセンブリ言語で記述されたアセンブリソースファイルです。このファイルの構造は下記のようになっています。 ベクタアドレス ＋ スタートアップルーチン
2	initsct_3048.c	初期値のないグローバル変数(セクション B 領域)、初期値のあるグローバル変数(セクション R 領域)の初期化用です。
3	beep_clock.c	実際に制御するプログラムが書かれています。H8/3048F-ONE の内蔵周辺機能の初期化も行います。
4	h8_3048.h	H8/3048F-ONE の内蔵周辺機能の I/O レジスタを定義したファイルです。
5	initsct_3048.h	initsct_3048.c のヘッダファイルです。

21.4 プログラム「beep_clock.c」

```
 1 :  /************************************************************************/
 2 :  /* 音階出力　大きな古時計版 "beep_clokc.c"                              */
 3 :  /* 出力：PA2(圧電ブザー、アンプ等)                                      */
 4 :  /*                          2008.09 ジャパンマイコンカーラリー実行委員会 */
 5 :  /************************************************************************/
 6 :  /*======================================*/
 7 :  /* インクルード                         */
 8 :  /*======================================*/
 9 :  #include    <machine.h>
10 :  #include    "h8_3048.h"
11 :
12 :  /*======================================*/
```

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

/* 定数設定 */
#define         TIMER_CYCLE     3071    /* タイマのサイクル 1ms     */
                                        /* φ/8で使用する場合、      */
                                        /* φ/8 = 325.5[ns]          */
                                        /* ∴TIMER_CYCLE =           */
                                        /*      1[ms] / 325.5[ns]   */
                                        /*                = 3072    */

/* 低い音階１ */
#define         L1_DO       23486       /* ド                       */
#define         L1_DOU      22168       /* ド＃                     */
#define         L1_RE       20924       /* レ                       */
#define         L1_REU      19750       /* レ＃                     */
#define         L1_MI       18641       /* ミ                       */
#define         L1_FA       17595       /* ファ                     */
#define         L1_FAU      16607       /* ファ＃                   */
#define         L1_SO       15675       /* ソ                       */
#define         L1_SOU      14795       /* ソ＃                     */
#define         L1_RA       13965       /* ラ                       */
#define         L1_RAU      13181       /* ラ＃                     */
#define         L1_SI       12441       /* シ                       */

/* 標準の音階 */
#define         M_DO        11743       /* ド                       */
#define         M_DOU       11084       /* ド＃                     */
#define         M_RE        10462       /* レ                       */
#define         M_REU       9875        /* レ＃                     */
#define         M_MI        9321        /* ミ                       */
#define         M_FA        8797        /* ファ                     */
#define         M_FAU       8304        /* ファ＃                   */
#define         M_SO        7838        /* ソ                       */
#define         M_SOU       7398        /* ソ＃                     */
#define         M_RA        6983        /* ラ                       */
#define         M_RAU       6591        /* ラ＃                     */
#define         M_SI        6221        /* シ                       */

/* 高い音階１ */
#define         H1_DO       5872        /* ド                       */
#define         H1_DOU      5542        /* ド＃                     */
#define         H1_RE       5231        /* レ                       */
#define         H1_REU      4937        /* レ＃                     */
#define         H1_MI       4660        /* ミ                       */
#define         H1_FA       4399        /* ファ                     */
#define         H1_FAU      4152        /* ファ＃                   */
#define         H1_SO       3919        /* ソ                       */
#define         H1_SOU      3699        /* ソ＃                     */
#define         H1_RA       3491        /* ラ                       */
#define         H1_RAU      3295        /* ラ＃                     */
#define         H1_SI       3110        /* シ                       */

#define         TEMPO       60          /* テンポ                   */

/*=======================================*/
/* プロトタイプ宣言                      */
/*=======================================*/
void init( void );
void timer( unsigned long timer_set );
unsigned char dipsw_get( void );
void note( unsigned int tone );

/*=======================================*/
/* グローバル変数の宣言                  */
/*=======================================*/
unsigned long   cnt0;                   /* timer用                  */
unsigned long   cnt1;                   /* 音楽用                   */
int             music_dim;              /* 音楽データ配列の位置     */
int             music_flag;             /* 音楽データならすかどうか */

const int music_data[][2] = {           /* 大きな古時計　音楽データ */
/*
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。
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。
*/
/*  音階,   長さ    */
    0,      0,  /* スタート*/

    M_RE,   4,  /* お */
    M_SO,   4,  /* お */
    M_FAU,  2,  /* き */
    M_SO,   2,  /* な */
    M_RA,   4,  /* のっ */
    M_SO,   2,  /* ぽ */
    M_RA,   2,  /* の */
    M_SI,   2,  /* ふ */
    M_SI,   2,  /* る */
    H1_DO,  2,  /* ど */
    M_SI,   2,  /* け */
    M_MI,   4,  /* い */
```

beep_haru.c の音楽データを
変更しています。

```
104 :       M_RA,   2,  /* お */
105 :       M_RA,   2,  /* じ */
106 :       M_SO,   4,  /* い */
107 :       M_SO,   2,  /* さ */
108 :       M_SO,   2,  /* ん */
109 :       M_FAU,  4,  /* の */
110 :       M_MI,   2,  /* と */
111 :       M_FAU,  2,  /* け */
112 :       M_SO,   10, /* い */
113 :       0,      2,
114 :
115 :       M_RE,   2,  /* ひゃ */
116 :       M_RE,   2,  /* く */
117 :       M_SO,   4,  /* ねん */
118 :       M_FAU,  2,  /* い */
119 :       M_SO,   2,  /* つ */
120 :       M_RA,   4,  /* も */
121 :       M_SO,   2,  /* う */
122 :       M_RA,   2,  /* ご */
123 :       M_SI,   4,  /* い */
124 :       H1_DO,  2,  /* て */
125 :       M_SI,   2,  /* い */
126 :       M_MI,   4,  /* た */
127 :       M_RA,   2,  /* ご */
128 :       M_RA,   2,  /* じ */
129 :       M_SO,   4,  /* ま */
130 :       M_SO,   2,  /* ん */
131 :       M_SO,   2,  /* の */
132 :       M_FAU,  4,  /* と */
133 :       M_MI,   2,  /* け */
134 :       M_FAU,  2,  /* い */
135 :       M_SO,   10, /* さ */
136 :       0,      2,
137 :
138 :       M_SO,   2,  /* お */
139 :       M_SI,   2,  /* じ */
140 :       H1_RE,  4,  /* い */
141 :       M_SI,   2,  /* さ */
142 :       M_RA,   2,  /* ん */
143 :       M_SO,   4,  /* の */
144 :       M_FAU,  2,  /* う */
145 :       M_SO,   2,  /* ま */
146 :       M_RA,   2,  /* れ */
147 :       M_SO,   2,  /* た */
148 :       M_FAU,  2,  /* あ */
149 :       M_MI,   2,  /* さ */
150 :       M_RE,   4,  /* に */
151 :       M_SO,   2,  /* か */
152 :       M_SI,   2,  /* っ */
153 :       H1_RE,  4,  /* て */
154 :       M_SI,   2,  /* き */
155 :       M_RA,   2,  /* た */
156 :       M_SO,   4,  /* と */
157 :       M_FAU,  2,  /* け */
158 :       M_SO,   2,  /* い */
159 :       M_RA,   10, /* さ */
160 :       0,      2,
161 :
162 :       M_RE,   2,  /* い */
163 :       M_SO,   2,  /* ま */
164 :       M_SO,   2,  /* は */
165 :       0,      4,
166 :       M_RA,   2,  /* もう */
167 :       0,      2,
168 :       0,      4,
169 :       M_SI,   2,  /* う */
170 :       M_SI,   2,  /* ご */
171 :       H1_DO,  2,  /* か */
172 :       M_SI,   2,  /* な */
173 :       M_MI,   4,  /* い */
174 :       M_RA,   2,  /* そ */
175 :       M_RA,   2,  /* の */
176 :       M_SO,   8,  /* と */
177 :       M_FAU,  8,  /* け */
178 :       M_SO,   8,  /* い */
179 :       0,      2,
180 :
181 :       M_RE,   2,  /* ひゃ */
182 :       M_RE,   2,  /* く */
183 :       M_SO,   4,  /* ねん */
184 :       M_RE,   2,  /* や */
185 :       M_RE,   2,  /* す */
186 :       M_MI,   2,  /* ま */
187 :       M_RE,   2,  /* ず */
188 :       M_RE,   4,  /* に */
189 :       L1_SI,  2,  /* ちっく */
190 :       0,      2,
191 :       M_RE,   2,  /* たっく */
192 :       0,      2,
193 :       L1_SI,  2,  /* ちっく */
194 :       0,      2,
```

```
195 :      M_RE,   2,  /* たっく */
196 :      M_RE,   2,  /* お */
197 :      M_SO,   4,  /* じー */
198 :      M_RE,   2,  /* さん */
199 :      M_RE,   2,  /* と */
200 :      M_MI,   4,  /* いっ */
201 :      M_RE,   2,  /* しょ */
202 :      M_RE,   2,  /* に */
203 :      L1_SI,  2,  /* ちっく */
204 :      0,      2,
205 :      M_RE,   2,  /* たっく */
206 :      0,      2,
207 :      L1_SI,  2,  /* ちっく */
208 :      0,      2,
209 :      M_RE,   2,  /* たっく */
210 :
211 :      M_RE,   2,  /* い */
212 :      M_SO,   2,  /* ま */
213 :      M_SO,   2,  /* は */
214 :      0,      4,
215 :      M_RA,   2,  /* もう */
216 :      0,      2,
217 :      0,      4,
218 :      M_SI,   2,  /* う */
219 :      M_SI,   2,  /* ご */
220 :      H1_DO,  2,  /* か */
221 :      M_SI,   2,  /* な */
222 :      M_MI,   4,  /* い */
223 :      M_RA,   2,  /* そ */
224 :      M_RA,   2,  /* の */
225 :      M_SO,   8,  /* と */
226 :      M_FAU,  8,  /* け */
227 :      M_SO,   12, /* い */
228 :      0,      4,
229 :
230 :      -1,     0   /* 終了 */
231 :  };
232 :
233 :  /************************************************************************/
234 :  /* メインプログラム                                                     */
235 :  /************************************************************************/
236 :  void main( void )
237 :  {
238 :      init();                             /* マイコン機能の初期化     */
239 :      set_ccr( 0x00 );                    /* 全体割り込み許可         */
240 :
241 :      music_flag = 1;                     /* 音楽スタート             */
242 :
243 :      while( 1 ) {
244 :      }
245 :  }
246 :
247 :  /************************************************************************/
248 :  /* 音を鳴らす                                                           */
249 :  /* 引数   長さ[ms]                                                      */
250 :  /*        トーンのPWM値                                                 */
251 :  /************************************************************************/
252 :  void note( unsigned int tone )
253 :  {
254 :      if( tone ) {
255 :          ITU_STR &= 0xfe;                /* PWM停止                  */
256 :          ITU0_CNT = tone - 1;
257 :          ITU0_GRA = tone;                /* トーン設定               */
258 :          ITU0_GRB = tone / 2;            /* デューティ比は５０％     */
259 :          ITU_STR |= 0x01;                /* PWM開始                  */
260 :      } else {
261 :          ITU_STR &= 0xfe;                /* PWM停止                  */
262 :          ITU0_CNT = 0;                   /* 0%                       */
263 :          ITU0_GRA = 1;
264 :          ITU0_GRB = 2;
265 :          ITU_STR |= 0x01;                /* PWM開始                  */
266 :      }
267 :  }
268 :
269 :  /************************************************************************/
270 :  /* H8/3048F内蔵モジュール　初期化                                       */
271 :  /************************************************************************/
272 :  void init( void )
273 :  {
274 :      /* ポートの入出力設定 */
275 :      P1DDR = 0xff;
276 :      P2DDR = 0xff;
277 :      P3DDR = 0xff;
278 :      P4DDR = 0xff;
279 :      P5DDR = 0xff;
280 :      P6DDR = 0xf0;                       /* CPU基板上のDIP SW        */
281 :      P8DDR = 0xff;
282 :      P9DDR = 0xf7;                       /* 通信ポート               */
283 :      PADDR = 0xff;                       /* bit2:圧電ブザー          */
284 :      PBDDR = 0xff;
285 :      /* ポート７は、入力専用なので入出力設定はありません    */
```

```
286 :
287 :       /* ITU0 ブザー用PWM */
288 :       ITU0_TCR = 0x23;
289 :       ITU0_GRA = 1;
290 :       ITU0_GRB = 2;
291 :       ITU0_CNT = 0;
292 :       ITU_MDR |= 0x01;                    /* PWMモード設定            */
293 :
294 :       /* ITU3 1ms毎の割り込み  timer関数用 */
295 :       ITU3_TCR = 0x23;
296 :       ITU3_GRA = TIMER_CYCLE;
297 :       ITU3_CNT = 0;
298 :       ITU3_IER = 0x01;                    /* 割り込み許可             */
299 :
300 :       ITU_STR = 0x09;                     /* ITUのカウントスタート    */
301 :   }
302 :
303 :   /************************************************************************/
304 :   /* ITU3 割り込み処理                                                    */
305 :   /************************************************************************/
306 :   #pragma interrupt( interrupt_timer3 )
307 :   void interrupt_timer3( void )
308 :   {
309 :       ITU3_TSR &= 0xfe;                   /* フラグクリア             */
310 :       cnt0++;
311 :       cnt1++;
312 :
313 :       if( music_flag ) {
314 :           /* 音階処理 */
315 :           if( cnt1 >= 15000L*music_data[music_dim][1]/TEMPO ) {
316 :               /* 次の音階をならす */
317 :               cnt1 = 0;
318 :               music_dim++;
319 :               if( music_data[music_dim][0] == -1 ) {
320 :                   /* -1なら終了 */
321 :                   note( 0 );
322 :                   music_flag = 0;
323 :               } else {
324 :                   /* -1でないなら次の音階セット */
325 :                   note( music_data[music_dim][0] );
326 :               }
327 :           }
328 :       }
329 :   }
330 :
331 :   /************************************************************************/
332 :   /* タイマ本体                                                           */
333 :   /* 引数   タイマ値 1=1ms                                                */
334 :   /************************************************************************/
335 :   void timer( unsigned long timer_set )
336 :   {
337 :       cnt0     = 0;
338 :       while( cnt0 < timer_set );
339 :   }
340 :
341 :   /************************************************************************/
342 :   /* ディップスイッチ値読み込み                                           */
343 :   /* 戻り値 スイッチ値 0～15                                              */
344 :   /************************************************************************/
345 :   unsigned char dipsw_get( void )
346 :   {
347 :       unsigned char sw;
348 :
349 :       sw  = ~P6DR;                        /* ディップスイッチ読み込み */
350 :       sw &= 0x0f;
351 :
352 :       return  sw;
353 :   }
354 :
355 :   /************************************************************************/
356 :   /* end of file                                                          */
357 :   /************************************************************************/
```

21.5 プログラムの解説

プロジェクト「beep_haru」と同じです。

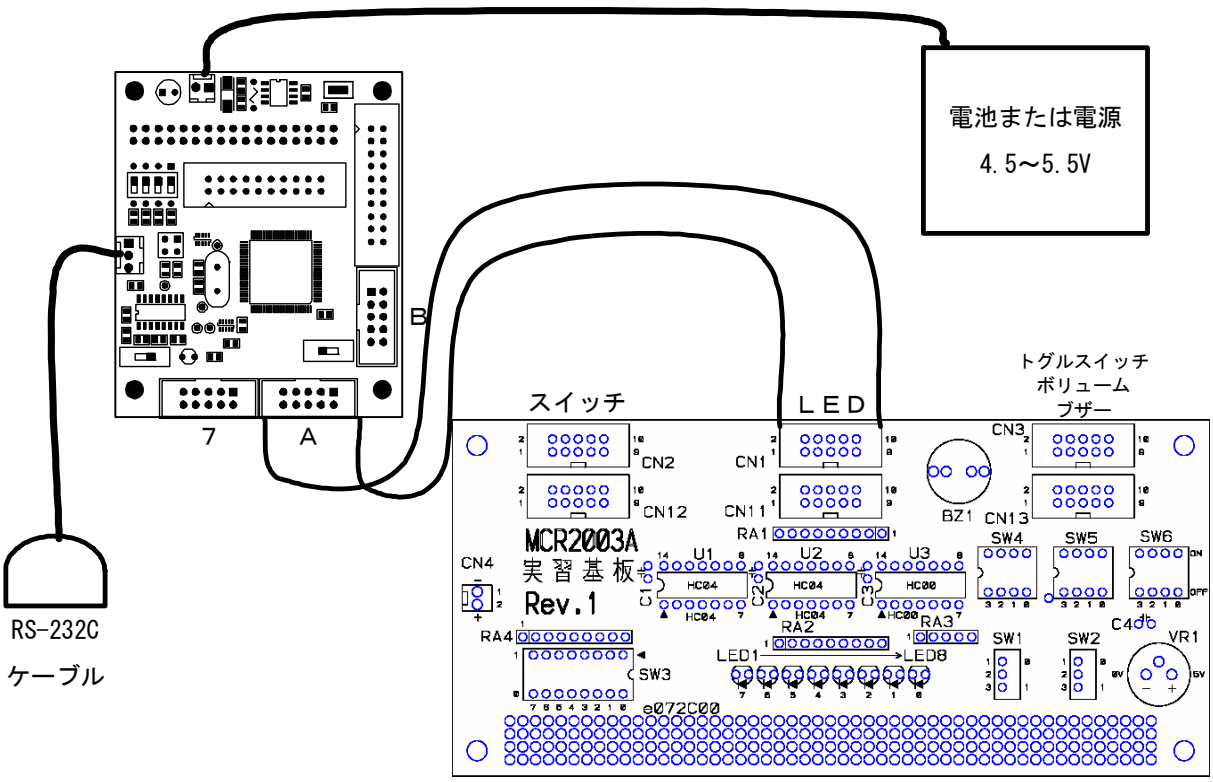
22. プロジェクト「wdt」 WDT割り込みによるタイマ

22.1 概要

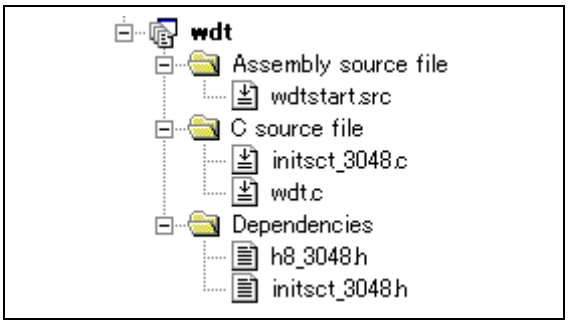
カウンタの値を 1 秒ごとに増やしていき、その値を LED へ出力します。時間の計測には、WDT による割り込みを使用します。マイコンのポートは、下記を使用します。
・ポート A の全ビット・・・LED へデータ出力

22.2 接続

・CPU ボードのポート A と、実習基板の LED 部をフラットケーブルで接続します。



22.3 プロジェクトの構成



	ファイル名	内容
1	wdtstart.src	アセンブリ言語で記述されたアセンブリソースファイルです。このファイルの構造は下記のようになっています。 ベクタアドレス ＋ スタートアップルーチン
2	initsct_3048.c	初期値のないグローバル変数(セクション B 領域)、初期値のあるグローバル変数(セクション R 領域)の初期化用です。
3	wdt.c	実際に制御するプログラムが書かれています。H8/3048F-ONE の内蔵周辺機能の初期化も行います。
4	h8_3048.h	H8/3048F-ONE の内蔵周辺機能の I/O レジスタを定義したファイルです。
5	initsct_3048.h	initsct_3048.c のヘッダファイルです。

22.4 プログラム「wdt.c」

```
 1 : /************************************************************************/
 2 : /* タイマ(WDT割り込み)でＬＥＤ等を光らす「wdt.c」                       */
 3 : /* 出力：PA7-PA0(LED等)                                                 */
 4 : /*                              2008.06 ジャパンマイコンカーラリー実行委員会  */
 5 : /************************************************************************/
 6 : /*======================================*/
 7 : /* インクルード                         */
 8 : /*======================================*/
 9 : #include    <machine.h>
10 : #include    "h8_3048.h"
11 :
12 : /*======================================*/
13 : /* シンボル定義                         */
14 : /*======================================*/
15 :
16 : /*======================================*/
17 : /* プロトタイプ宣言                     */
18 : /*======================================*/
19 : void init( void );
20 :
21 : /*======================================*/
22 : /* グローバル変数の宣言                 */
23 : /*======================================*/
24 : unsigned long   cnt_wdt;               /* WDT用                */
25 : int             cnt_wdt2;              /* WDT内部処理用        */
26 :
27 : /************************************************************************/
28 : /* メインプログラム                                                     */
29 : /************************************************************************/
30 : void main( void )
31 : {
32 :     unsigned char d = 0;
33 :
34 :     init();                            /* 初期化               */
35 :     set_ccr( 0x00 );                   /* 全体割り込み許可     */
36 :
37 :     while( 1 ) {
38 :         if( cnt_wdt >= 1000 ) {        /* 1sたったか？         */
```

```
39 :              cnt_wdt = 0;
40 :              PADR = ++d;
41 :          }
42 :      }
43 :  }
44 :
45 :  /************************************************************************/
46 :  /* H8/3048F-ONE 内蔵周辺機能の初期化                                    */
47 :  /************************************************************************/
48 :  void init( void )
49 :  {
50 :      /* ポートの入出力設定 */
51 :      P1DDR = 0xff;
52 :      P2DDR = 0xff;
53 :      P3DDR = 0xff;
54 :      P4DDR = 0xff;
55 :      P5DDR = 0xff;
56 :      P6DDR = 0xf0;                       /* CPU基板上のDIP SW        */
57 :      P8DDR = 0xff;
58 :      P9DDR = 0xf7;
59 :      PADDR = 0xff;                       /* LED基板                  */
60 :      PBDDR = 0xff;
61 :      /* ポート７は、入力専用なので入出力設定はありません    */
62 :
63 :      /* WDT設定 */
64 :      TCSR_W = 0xa521;                    /* WDT=インターバルタイマ   */
65 :
66 :      cnt_wdt  = 0;
67 :      cnt_wdt2 = 0;
68 :  }
69 :
70 :  /************************************************************************/
71 :  /* WDT 割り込み処理 0.333msごと(1/3000)の割り込み                       */
72 :  /************************************************************************/
73 :  #pragma interrupt( interrupt_wdt )
74 :  void interrupt_wdt( void )
75 :  {
76 :      unsigned char work;
77 :
78 :      work = TCSR & 0x7f;                 /* フラグリード             */
79 :      TCSR_W = 0xa500 | work;             /* フラグクリア             */
80 :
81 :      if( ++cnt_wdt2 >= 3 ) {
82 :          /* この中は1msごとに実行 */
83 :          cnt_wdt2 = 0;
84 :          cnt_wdt++;
85 :      }
86 :  }
87 :
88 :  /************************************************************************/
89 :  /* end of file                                                          */
90 :  /************************************************************************/
```

22.5 プログラム「wdtstart.src」

ゴシック体が、WDT 割り込みを使うために追加、変更した行です。

```
1 :  ;======================================
2 :  ; 定義
3 :  ;======================================
4 :  RESERVE: .EQU    H'FFFFFFFF            ; 未使用領域のアドレス
5 :
6 :  ;======================================
7 :  ; 外部参照
8 :  ;======================================
9 :          .IMPORT _main
10 :         .IMPORT _interrupt_wdt
11 :         .IMPORT _INITSCT
12 :
13 :  ;======================================
14 :  ; ベクタセクション
15 :  ;======================================
16 :         .SECTION V
17 :         .DATA.L RESET_START             ;  0 H'000000   ﾘｾｯﾄ
18 :         .DATA.L RESERVE                 ;  1 H'000004   ｼｽﾃﾑ予約
19 :         .DATA.L RESERVE                 ;  2 H'000008   ｼｽﾃﾑ予約
20 :         .DATA.L RESERVE                 ;  3 H'00000c   ｼｽﾃﾑ予約
21 :         .DATA.L RESERVE                 ;  4 H'000010   ｼｽﾃﾑ予約
22 :         .DATA.L RESERVE                 ;  5 H'000014   ｼｽﾃﾑ予約
23 :         .DATA.L RESERVE                 ;  6 H'000018   ｼｽﾃﾑ予約
24 :         .DATA.L RESERVE                 ;  7 H'00001c   外部割り込み NMI
25 :         .DATA.L RESERVE                 ;  8 H'000020   ﾄﾗｯﾌﾟ命令
26 :         .DATA.L RESERVE                 ;  9 H'000024   ﾄﾗｯﾌﾟ命令
27 :         .DATA.L RESERVE                 ; 10 H'000028   ﾄﾗｯﾌﾟ命令
28 :         .DATA.L RESERVE                 ; 11 H'00002c   ﾄﾗｯﾌﾟ命令
```

```
29 :          .DATA.L RESERVE                ; 12 H'000030   外部割り込み IRQ0
30 :          .DATA.L RESERVE                ; 13 H'000034   外部割り込み IRQ1
31 :          .DATA.L RESERVE                ; 14 H'000038   外部割り込み IRQ2
32 :          .DATA.L RESERVE                ; 15 H'00003c   外部割り込み IRQ3
33 :          .DATA.L RESERVE                ; 16 H'000040   外部割り込み IRQ4
34 :          .DATA.L RESERVE                ; 17 H'000044   外部割り込み IRQ5
35 :          .DATA.L RESERVE                ; 18 H'000048   ｼｽﾃﾑ予約
36 :          .DATA.L RESERVE                ; 19 H'00004c   ｼｽﾃﾑ予約
37 :          .DATA.L _interrupt_wdt         ; 20 H'000050   WDT MOVI
38 :          .DATA.L RESERVE                ; 21 H'000054   REF CMI
39 :          .DATA.L RESERVE                ; 22 H'000058   ｼｽﾃﾑ予約
40 :          .DATA.L RESERVE                ; 23 H'00005c   ｼｽﾃﾑ予約
41 :          .DATA.L RESERVE                ; 24 H'000060   ITU0 IMIA0
42 :          .DATA.L RESERVE                ; 25 H'000064   ITU0 IMIB0
43 :          .DATA.L RESERVE                ; 26 H'000068   ITU0 OVI0
44 :          .DATA.L RESERVE                ; 27 H'00006c   ｼｽﾃﾑ予約
45 :          .DATA.L RESERVE                ; 28 H'000070   ITU1 IMIA1
46 :          .DATA.L RESERVE                ; 29 H'000074   ITU1 IMIB1
47 :          .DATA.L RESERVE                ; 30 H'000078   ITU1 OVI1
48 :          .DATA.L RESERVE                ; 31 H'00007c   ｼｽﾃﾑ予約
49 :          .DATA.L RESERVE                ; 32 H'000080   ITU2 IMIA2
50 :          .DATA.L RESERVE                ; 33 H'000084   ITU2 IMIB2
51 :          .DATA.L RESERVE                ; 34 H'000088   ITU2 OVI2
52 :          .DATA.L RESERVE                ; 35 H'00008c   ｼｽﾃﾑ予約
53 :          .DATA.L RESERVE                ; 36 H'000090   ITU3 IMIA3
54 :          .DATA.L RESERVE                ; 37 H'000094   ITU3 IMIB3
55 :          .DATA.L RESERVE                ; 38 H'000098   ITU3 OVI3
56 :          .DATA.L RESERVE                ; 39 H'00009c   ｼｽﾃﾑ予約
57 :          .DATA.L RESERVE                ; 40 H'0000a0   ITU4 IMIA4
58 :          .DATA.L RESERVE                ; 41 H'0000a4   ITU4 IMIB4
59 :          .DATA.L RESERVE                ; 42 H'0000a8   ITU4 OVI4
60 :          .DATA.L RESERVE                ; 43 H'0000ac   ｼｽﾃﾑ予約
61 :          .DATA.L RESERVE                ; 44 H'0000b0   DMAC DEND0A
62 :          .DATA.L RESERVE                ; 45 H'0000b4   DMAC DEND0B
63 :          .DATA.L RESERVE                ; 46 H'0000b8   DMAC DEND1A
64 :          .DATA.L RESERVE                ; 47 H'0000bc   DMCA DEND1B
65 :          .DATA.L RESERVE                ; 48 H'0000c0   ｼｽﾃﾑ予約
66 :          .DATA.L RESERVE                ; 49 H'0000c4   ｼｽﾃﾑ予約
67 :          .DATA.L RESERVE                ; 50 H'0000c8   ｼｽﾃﾑ予約
68 :          .DATA.L RESERVE                ; 51 H'0000cc   ｼｽﾃﾑ予約
69 :          .DATA.L RESERVE                ; 52 H'0000d0   SCI0 ERI0
70 :          .DATA.L RESERVE                ; 53 H'0000d4   SCI0 RXI0
71 :          .DATA.L RESERVE                ; 54 H'0000d8   SCI0 TXI0
72 :          .DATA.L RESERVE                ; 55 H'0000dc   SCI0 TEI0
73 :          .DATA.L RESERVE                ; 56 H'0000e0   SCI1 ERI1
74 :          .DATA.L RESERVE                ; 57 H'0000e4   SCI1 RXI1
75 :          .DATA.L RESERVE                ; 58 H'0000e8   SCI1 TXI1
76 :          .DATA.L RESERVE                ; 59 H'0000ec   SCI1 TEI1
77 :          .DATA.L RESERVE                ; 60 H'0000f0   A/D ADI
78 :
79 :  ;=====================================
80 :  ; スタートアッププログラム
81 :  ;=====================================
82 :          .SECTION P
83 :  RESET_START:
84 :          MOV.L   #H'FFF10,ER7           ; スタックの設定
85 :          JSR     @_INITSCT              ; RAMエリアの初期化
86 :          JSR     @_main                 ; Ｃ言語のmain()関数へジャンプ
87 :  OWARI:
88 :          BRA     OWARI
89 :
90 :          .END
```

22.6 プログラムの解説

22.6.1 WDTとは？

WDT は、ウォッチドッグタイマ(watchdog timer)の略称です。日本語ではよく、「番犬」と例えます。
WDT は、2 つの機能があり、どちらかを選んで使用することができます。

機能	詳細
ウォッチドッグタイマ	WDT のカウンタをクリアするプログラムを入れておきます。WDT のカウンタを所定時間内にクリアできないと、システムが暴走したと判断して、強制的にリセットさせます。 番犬と言われるゆえんです。
インターバルタイマ	WDT をインターバルタイマとして使用することできます。

プロジェクト「timer2」では、ITU による割り込みでタイマの機能を実現していました。ただ、ITU は PWM 機能(モータ制御)やパルスカウント(エンコーダ用)として利用することが多く、割り込みとして使いたくないことがあります。そこで、WDT をインターバルタイマとして利用し、ITU を別な用途で使えるように空けておきましょう。

22.6.2 WDTの設定

init 関数内で、WDT をインターバルタイマとして利用できるよう設定しています。プログラムは下記のとおりです。

```
64 :        TCSR_W = 0xa521;                    /* WDT=インターバルタイマ    */
```

●TCNT(タイマカウンタ)の設定内容

TCNT は、8 ビットのリード／ライト可能なアップカウンタです。

ビット:	7	6	5	4	3	2	1	0
TCNT:								

TCSR(タイマコントロール／ステータスレジスタ)の TME(タイマネーブル)ビットを 1 にすると、TSCR の CKS2～CKS0 ビットで選択された内部クロックにより、カウントアップを開始します。また、TCNT の値がオーバフロー(0xff→0x00)すると、TCSR の OVF フラグが 1 にセットされます。また、TCNT はリセット、または TME=0 のとき 0x00 にイニシャライズされます。

●TCSR(タイマコントロール／ステータスレジスタ)の設定内容

ビット:	7	6	5	4	3	2	1	0
TCSR:	OVF	WT/$\overline{IT}$	TME	-	-	CKS2	CKS1	CKS0
設定値:	0	0	1	0	0	0	0	1
16 進数:	2				1			

・ビット 7:オーバフローフラグ(OVF)

TCNT がオーバフロー(0xff→0x00)したことを示すステータスフラグです。

WT/$\overline{IT}$	説明
0	[クリア条件] OVF=1 の状態で、OVF フラグをリード後、OVF フラグに 0 をライトしたとき
1	[セット条件] TCNT が 0xff→0x00 に変化したとき

・ビット 6:タイマモードセレクト(WT/$\overline{IT}$)

WDT をウオッチドッグタイマとして使用するか、インターバルタイマとして使用するかを選択するビットです。インターバルタイマ時は TCNT(タイマカウンタ)のオーバフローでインターバルタイマ割り込み要求を発生します。また、ウオッチドッグタイマ時は TCNT のオーバフローでリセット信号を発生します。

WT/$\overline{IT}$	説明
0	**インターバルタイマを選択:インターバルタイマ割り込み要求**
1	ウォッチドッグタイマを選択:リセット信号を発生

WDT をインターバルタイマとして使用するので"0"を選択します。

・ビット 5:タイマイネーブル(TME)

TCNT の動作／停止を選択します。WT/$\overline{IT}$=1 の場合、SYSCR のソフトウェアスタンバイビット(SSBY)を 0 にクリアしてから TME を 1 にセットしてください。また、SSBY を 1 にセットするときは TME を 0 にクリアしてください。

TME	説明
0	TCNT を H'00 にイニシャライズし、カウント動作は停止
1	**TCNT はカウント動作、CPU への割り込み要求を許可**

インターバルタイマを動作させるので、"1"を選択します。このビットを"1"にすることにより、割り込みが許可されます。WDT のインターバルタイマの割り込みベクタ番号は、20 番になります。

・ビット 2～0:クロックセレクト 2～0

システムクロックφを分周して得られる 8 種類の内部クロックから TCNT に入力するクロックを選択するビットです。

CKS2	CKS1	CKS0	説明
0	0	0	**φ/2** 1パルス＝1÷(24.576×10^6／2)＝81.38[ns] 割り込み周期は 81.38[ns]×256＝20.83[μs] 1 秒間に割り込みが 48,000 回実行される計算です。
0	**0**	**1**	**φ/32** 1パルス＝1÷(24.576×10^6／32)＝1.302[μs] 割り込み周期は 1.302[μs]×256＝333.3[μs] 1 秒間に割り込みが 3,000 回実行される計算です。
0	1	0	**φ/64** 1パルス＝1÷(24.576×10^6／64)＝2.604[μs] 割り込み周期は 2.604[μs]×256＝6.667[μs] 1 秒間に割り込みが 1,500 回実行される計算です。
0	1	1	**φ/128** 1パルス＝1÷(24.576×10^6／128)＝5.208[μs] 割り込み周期は 5.208[μs]×256＝1.333[ms] 1 秒間に割り込みが 750 回実行される計算です。
1	0	0	**φ/256** 1パルス＝1÷(24.576×10^6／256)＝10.42[μs] 割り込み周期は 10.42[μs]×256＝2.667[ms] 1 秒間に割り込みが 375 回実行される計算です。
1	0	1	**φ/512** 1パルス＝1÷(24.576×10^6／512)＝20.83[μs] 割り込み周期は 20.83[μs]×256＝5.333[ms] 1 秒間に割り込みが 187.5 回実行される計算です。
1	1	0	**φ/2048** 1パルス＝1÷(24.576×10^6／2048)＝83.33[μs] 割り込み周期は 83.33[μs]×256＝21.33[ms] 1 秒間に割り込みが 46.875 回実行される計算です。
1	1	1	**φ/4096** 1パルス＝1÷(24.576×10^6／4096)＝166.67[μs] 割り込み周期は 166.67[μs]×256＝42.67[ms] 1 秒間に割り込みが 23.4375 回実行される計算です。

TCNT(タイマカウンタ)をどのくらいの間隔でカウントアップさせるかの設定です。今回は、"001"を設定します。これは、TCNT が 1.302[μs]ごとに+1 する設定です。TCNT は 8 ビット幅なので、255 の次が 0 になります。この瞬間に OVF(bit7)が"1"になります。OVF が"1"になる間隔は、

1.302[μs]×256＝333.3[μs]

となります。これが割り込み周期になります。1 秒間に何回割り込みがかかるかというと、

1[s]÷333.3[μs]＝3,000

となり、1 秒間に 3,000 回割り込みがかかる計算になります。変えたい場合は、CKS2～CKS0 を変えます。

●TCSR の書き込み方、読み込み方

WDT(ウオッチドッグタイマ)の TCSR(タイマコントロール／ステータスレジスタ)は、簡単に書き換えられないよう書き込み方法が一般のレジスタとは異なります。書き換え、読み込み方法を次に示します。

(1) TCSR への書き込み方法

TCSR へ値を書き込みたいとき、レジスタ名は TCSR_W を使います。TCSR_W には「**書き込みたい値＋0xa500**」の値を書き込みます。

例)書き込みたい値が 0x21 のとき

TCSR_W = 0xa500 + 0x21 = **0xa521**

(2) TCSR の値を読み込む方法

読み込み方は通常のレジスタと同じです。

例)変数 c に TCSR の値を代入

```
c = TCSR;
```

22.6.3 割り込みプログラム

WDT のインターバルタイマで割り込みがかかったとき、実行される関数は interrupt_wdt 関数です。この関数は 0.333ms ごとに実行されます。1 秒間に 3000 回実行されます。

```
70 :  /************************************************************************/
71 :  /* WDT 割り込み処理 0.333ms ごと(1/3000)の割り込み                      */
72 :  /************************************************************************/
73 :  #pragma interrupt( interrupt_wdt )
74 :  void interrupt_wdt( void )
75 :  {
76 :      unsigned char work;
77 :
78 :      work = TCSR & 0x7f;                 /* フラグリード             */
79 :      TCSR_W = 0xa500 | work;             /* フラグクリア             */
80 :
81 :      if( ++cnt_wdt2 >= 3 ) {
82 :          /* この中は 1ms ごとに実行 */
83 :          cnt_wdt2 = 0;
84 :          cnt_wdt++;
85 :      }
86 :  }
```

78 行で TCSR(タイマコントロール／ステータスレジスタ)のオーバフローフラグ(OVF)が"1"になっているので、"0"にクリアします。クリア条件は「**OVF=1 の状態で、OVF フラグをリード後、OVF フラグに 0 をライトしたとき**」なので、直接"0"を書き込むのでなく、いったん work 変数に読み込んで AND 処理で OVF フラグである bit7 をクリアします。その値を、TCSR に書き込みます。ただし、書き込みは「TCSR_W に 0xa500 を足した値を書き込む」という決まりがあるので、79 行目でその決まりどおりにプログラムしています。

0.333ms ごとに割り込みが発生すると言うことは、3 回ごとに処理をすれば 1ms ごとにプログラムを実行することになります。そこで、81 行の if 文で 3 回目かチェックして、3 回目であるならばカッコの中を実行、カッコの中のプログラムは 1ms ごとに実行されることになります。カッコの中は、cnt_wdt 変数を＋1する命令を書いています。よって、cnt_wdt 変数の値 1 あたり、1ms となります。

WDT は割り込み周期を 8 種類しか選べないので、今回は 1 秒間に 3000 回実行する設定として、割り込みプログラムで 3 回に 1 回 cnt_wdt 変数を＋1するようにしています。

22.6.4 メインプログラム

```
27 :  /************************************************************************/
28 :  /* メインプログラム                                                     */
29 :  /************************************************************************/
30 :  void main( void )
31 :  {
32 :      unsigned char d = 0;
33 :
34 :      init();                         /* 初期化                   */
35 :      set_ccr( 0x00 );                /* 全体割り込み許可         */
36 :
37 :      while( 1 ) {
38 :          if( cnt_wdt >= 1000 ) {     /* 1s たったか？            */
39 :              cnt_wdt = 0;
40 :              PADR = ++d;
41 :          }
42 :      }
43 :  }
```

WDT のインターバルタイマ割り込みを使用するので、35 行で全体の割り込みを許可します。

他は特に難しいことはなく、38 行目で cnt_wdt 変数が 1000 になったなら、すなわち 1000ms たったなら if 文のカッコの中を実行します。カッコの中は、変数 d の値を＋1して、その値をポート A に出力します。

WDT のインターバルタイマ割り込みを使用するので、35 行で忘れずに全体の割り込みを許可します。

22.6.5 wdtstart.srcの追加、変更

```
  1 :  ;=====================================
  2 :  ; 定義
  3 :  ;=====================================
  4 :  RESERVE: .EQU    H'FFFFFFFF              ; 未使用領域のアドレス
  5 :
  6 :  ;=====================================
  7 :  ; 外部参照
  8 :  ;=====================================
  9 :          .IMPORT _main
 10 :          .IMPORT _interrupt_wdt
 11 :          .IMPORT _INITSCT
 12 :
 13 :  ;=====================================
 14 :  ; ベクタセクション
 15 :  ;=====================================
 16 :          .SECTION V
 17 :          .DATA.L RESET_START             ;  0 H'000000   ﾘｾｯﾄ
 18 :          .DATA.L RESERVE                 ;  1 H'000004   ｼｽﾃﾑ予約
 19 :          .DATA.L RESERVE                 ;  2 H'000008   ｼｽﾃﾑ予約
 20 :          .DATA.L RESERVE                 ;  3 H'00000c   ｼｽﾃﾑ予約
 21 :          .DATA.L RESERVE                 ;  4 H'000010   ｼｽﾃﾑ予約
 22 :          .DATA.L RESERVE                 ;  5 H'000014   ｼｽﾃﾑ予約
 23 :          .DATA.L RESERVE                 ;  6 H'000018   ｼｽﾃﾑ予約
 24 :          .DATA.L RESERVE                 ;  7 H'00001c   外部割り込み NMI
 25 :          .DATA.L RESERVE                 ;  8 H'000020   ﾄﾗｯﾌﾟ命令
 26 :          .DATA.L RESERVE                 ;  9 H'000024   ﾄﾗｯﾌﾟ命令
 27 :          .DATA.L RESERVE                 ; 10 H'000028   ﾄﾗｯﾌﾟ命令
 28 :          .DATA.L RESERVE                 ; 11 H'00002c   ﾄﾗｯﾌﾟ命令
 29 :          .DATA.L RESERVE                 ; 12 H'000030   外部割り込み IRQ0
 30 :          .DATA.L RESERVE                 ; 13 H'000034   外部割り込み IRQ1
 31 :          .DATA.L RESERVE                 ; 14 H'000038   外部割り込み IRQ2
 32 :          .DATA.L RESERVE                 ; 15 H'00003c   外部割り込み IRQ3
 33 :          .DATA.L RESERVE                 ; 16 H'000040   外部割り込み IRQ4
 34 :          .DATA.L RESERVE                 ; 17 H'000044   外部割り込み IRQ5
 35 :          .DATA.L RESERVE                 ; 18 H'000048   ｼｽﾃﾑ予約
 36 :          .DATA.L RESERVE                 ; 19 H'00004c   ｼｽﾃﾑ予約
 37 :          .DATA.L _interrupt_wdt          ; 20 H'000050   WDT MOVI
 38 :          .DATA.L RESERVE                 ; 21 H'000054   REF CMI
以下、略
```

WDT のインターバルタイマの割り込みベクタ番号は、20 番になります。37 行の 20 番には割り込みがかかったときに実行する関数「interrupt_wdt」を記述します。アセンブリソースプログラム(今回は wdtstart.src)から C 言語プログラム(今回は wdt.c)の関数を呼ぶときは、関数名の先頭に「__」(アンダバー)を付けなければいけない決まりがあるので、「_interrupt_wdt」としています。

10 行は、「_interrupt_wdt」が他のファイル(今回は wdt.c)にあることを知らせる記述です。

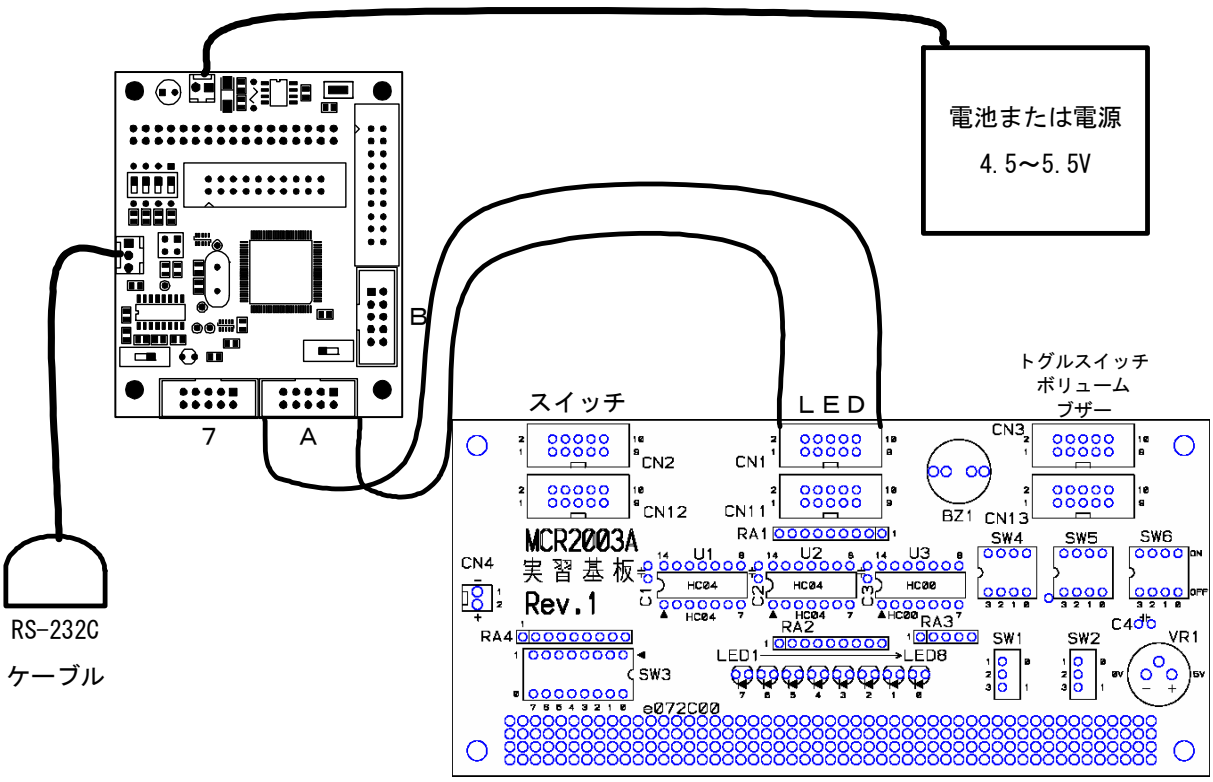
23. プロジェクト「rc_timer」 リフレッシュコントローラ割り込みによるタイマ

23.1 概要

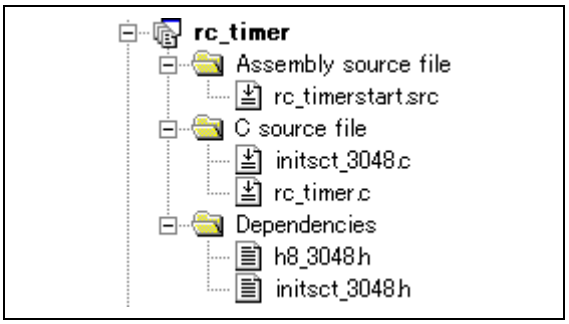
カウンタの値を 1 秒ごとに増やしていき、その値を LED へ出力します。時間の計測には、リフレッシュコントローラによる割り込みを使用します。マイコンのポートは、下記を使用します。
・ポート A の全ビット・・・LED へデータ出力

23.2 接続

・CPU ボードのポート A と、実習基板の LED 部をフラットケーブルで接続します。



23.3 プロジェクトの構成



	ファイル名	内容
1	rc_timerstart.src	アセンブリ言語で記述されたアセンブリソースファイルです。このファイルの構造は下記のようになっています。 ベクタアドレス ＋ スタートアップルーチン
2	initsct_3048.c	初期値のないグローバル変数(セクション B 領域)、初期値のあるグローバル変数(セクション R 領域)の初期化用です。
3	rc_timer.c	実際に制御するプログラムが書かれています。H8/3048F-ONE の内蔵周辺機能の初期化も行います。
4	h8_3048.h	H8/3048F-ONE の内蔵周辺機能の I/O レジスタを定義したファイルです。
5	initsct_3048.h	initsct_3048.c のヘッダファイルです。

23.4 プログラム「rc_timer.c」

```
 1 : /************************************************************************/
 2 : /* タイマ(ﾘﾌﾚｯｼｭｺﾝﾄﾛｰﾗによる割り込み)でLEDなどを光らす「rt_timer.c」         */
 3 : /* 出力：PA7-PA0(LED等)                                                 */
 4 : /*                              2009.03 ジャパンマイコンカーラリー実行委員会  */
 5 : /************************************************************************/
 6 : /*====================================*/
 7 : /* インクルード                       */
 8 : /*====================================*/
 9 : #include    <machine.h>
10 : #include    "h8_3048.h"
11 :
12 : /*====================================*/
13 : /* シンボル定義                       */
14 : /*====================================*/
15 :
16 : /*====================================*/
17 : /* プロトタイプ宣言                   */
18 : /*====================================*/
19 : void init( void );
20 :
21 : /*====================================*/
22 : /* グローバル変数の宣言               */
23 : /*====================================*/
24 : unsigned long   cnt_rc;                /* ﾘﾌﾚｯｼｭｺﾝﾄﾛｰﾗ用          */
25 :
26 : /************************************************************************/
27 : /* メインプログラム                                                     */
28 : /************************************************************************/
29 : void main( void )
30 : {
31 :     unsigned char d = 0;
32 :
33 :     init();                          /* 初期化                      */
34 :     set_ccr( 0x00 );                 /* 全体割り込み許可            */
35 :
36 :     while( 1 ) {
37 :         if( cnt_rc >= 1000 ) {       /* 1sたったか？                */
38 :             cnt_rc = 0;
```

```
            PADR = ++d;
        }
    }
}

/************************************************************************/
/* H8/3048F-ONE 内蔵周辺機能の初期化                                    */
/************************************************************************/
void init( void )
{
    /* ポートの入出力設定 */
    P1DDR = 0xff;
    P2DDR = 0xff;
    P3DDR = 0xff;
    P4DDR = 0xff;
    P5DDR = 0xff;
    P6DDR = 0xf0;                       /* CPU基板上のDIP SW        */
    P8DDR = 0xff;
    P9DDR = 0xf7;
    PADDR = 0xff;                       /* LED基板                  */
    PBDDR = 0xff;
    /* ポート7は、入力専用なので入出力設定はありません    */

    /* リフレッシュコントローラの設定 */
    RFSHCR = 0x00;                      /* RC=インターバルタイマ    */
    RTCOR  = 191;                       /* 1ms/5.208μs-1＝191       */
    RTMCSR = 0x60;                      /* φ/128、割り込み許可     */
}

/************************************************************************/
/* リフレッシュコントローラ 割り込み処理 1.000msごと                    */
/************************************************************************/
#pragma interrupt( interrupt_rc )
void interrupt_rc( void )
{
    RTMCSR &= 0x7f;                     /* フラグリード             */

    cnt_rc++;
}

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

23.5 プログラム「rc_timerstart.src」

ゴシック体が、リフレッシュコントローラによる 8 ビットインターバルタイマ割り込みを使うために追加、変更した行です。

```
;=====================================
; 定義
;=====================================
RESERVE: .EQU    H'FFFFFFFF             ; 未使用領域のアドレス

;=====================================
; 外部参照
;=====================================
        .IMPORT _main
        .IMPORT _interrupt_rc
        .IMPORT _INITSCT

;=====================================
; ベクタセクション
;=====================================
        .SECTION V
        .DATA.L RESET_START             ;  0 H'000000   ﾘｾｯﾄ
        .DATA.L RESERVE                 ;  1 H'000004   ｼｽﾃﾑ予約
        .DATA.L RESERVE                 ;  2 H'000008   ｼｽﾃﾑ予約
        .DATA.L RESERVE                 ;  3 H'00000c   ｼｽﾃﾑ予約
        .DATA.L RESERVE                 ;  4 H'000010   ｼｽﾃﾑ予約
        .DATA.L RESERVE                 ;  5 H'000014   ｼｽﾃﾑ予約
        .DATA.L RESERVE                 ;  6 H'000018   ｼｽﾃﾑ予約
        .DATA.L RESERVE                 ;  7 H'00001c   外部割り込み NMI
        .DATA.L RESERVE                 ;  8 H'000020   ﾄﾗｯﾌﾟ命令
        .DATA.L RESERVE                 ;  9 H'000024   ﾄﾗｯﾌﾟ命令
        .DATA.L RESERVE                 ; 10 H'000028   ﾄﾗｯﾌﾟ命令
        .DATA.L RESERVE                 ; 11 H'00002c   ﾄﾗｯﾌﾟ命令
        .DATA.L RESERVE                 ; 12 H'000030   外部割り込み IRQ0
        .DATA.L RESERVE                 ; 13 H'000034   外部割り込み IRQ1
        .DATA.L RESERVE                 ; 14 H'000038   外部割り込み IRQ2
        .DATA.L RESERVE                 ; 15 H'00003c   外部割り込み IRQ3
        .DATA.L RESERVE                 ; 16 H'000040   外部割り込み IRQ4
        .DATA.L RESERVE                 ; 17 H'000044   外部割り込み IRQ5
        .DATA.L RESERVE                 ; 18 H'000048   ｼｽﾃﾑ予約
```

```
36 :          .DATA.L RESERVE                ; 19 H'00004c   ｼｽﾃﾑ予約
37 :          .DATA.L RESERVE                ; 20 H'000050   WDT MOVI
38 :          .DATA.L _interrupt_rc          ; 21 H'000054   REF CMI
39 :          .DATA.L RESERVE                ; 22 H'000058   ｼｽﾃﾑ予約
40 :          .DATA.L RESERVE                ; 23 H'00005c   ｼｽﾃﾑ予約
41 :          .DATA.L RESERVE                ; 24 H'000060   ITU0 IMIA0
42 :          .DATA.L RESERVE                ; 25 H'000064   ITU0 IMIB0
43 :          .DATA.L RESERVE                ; 26 H'000068   ITU0 OVI0
44 :          .DATA.L RESERVE                ; 27 H'00006c   ｼｽﾃﾑ予約
45 :          .DATA.L RESERVE                ; 28 H'000070   ITU1 IMIA1
46 :          .DATA.L RESERVE                ; 29 H'000074   ITU1 IMIB1
47 :          .DATA.L RESERVE                ; 30 H'000078   ITU1 OVI1
48 :          .DATA.L RESERVE                ; 31 H'00007c   ｼｽﾃﾑ予約
49 :          .DATA.L RESERVE                ; 32 H'000080   ITU2 IMIA2
50 :          .DATA.L RESERVE                ; 33 H'000084   ITU2 IMIB2
51 :          .DATA.L RESERVE                ; 34 H'000088   ITU2 OVI2
52 :          .DATA.L RESERVE                ; 35 H'00008c   ｼｽﾃﾑ予約
53 :          .DATA.L RESERVE                ; 36 H'000090   ITU3 IMIA3
54 :          .DATA.L RESERVE                ; 37 H'000094   ITU3 IMIB3
55 :          .DATA.L RESERVE                ; 38 H'000098   ITU3 OVI3
56 :          .DATA.L RESERVE                ; 39 H'00009c   ｼｽﾃﾑ予約
57 :          .DATA.L RESERVE                ; 40 H'0000a0   ITU4 IMIA4
58 :          .DATA.L RESERVE                ; 41 H'0000a4   ITU4 IMIB4
59 :          .DATA.L RESERVE                ; 42 H'0000a8   ITU4 OVI4
60 :          .DATA.L RESERVE                ; 43 H'0000ac   ｼｽﾃﾑ予約
61 :          .DATA.L RESERVE                ; 44 H'0000b0   DMAC DEND0A
62 :          .DATA.L RESERVE                ; 45 H'0000b4   DMAC DEND0B
63 :          .DATA.L RESERVE                ; 46 H'0000b8   DMAC DEND1A
64 :          .DATA.L RESERVE                ; 47 H'0000bc   DMCA DEND1B
65 :          .DATA.L RESERVE                ; 48 H'0000c0   ｼｽﾃﾑ予約
66 :          .DATA.L RESERVE                ; 49 H'0000c4   ｼｽﾃﾑ予約
67 :          .DATA.L RESERVE                ; 50 H'0000c8   ｼｽﾃﾑ予約
68 :          .DATA.L RESERVE                ; 51 H'0000cc   ｼｽﾃﾑ予約
69 :          .DATA.L RESERVE                ; 52 H'0000d0   SCI0 ERI0
70 :          .DATA.L RESERVE                ; 53 H'0000d4   SCI0 RXI0
71 :          .DATA.L RESERVE                ; 54 H'0000d8   SCI0 TXI0
72 :          .DATA.L RESERVE                ; 55 H'0000dc   SCI0 TEI0
73 :          .DATA.L RESERVE                ; 56 H'0000e0   SCI1 ERI1
74 :          .DATA.L RESERVE                ; 57 H'0000e4   SCI1 RXI1
75 :          .DATA.L RESERVE                ; 58 H'0000e8   SCI1 TXI1
76 :          .DATA.L RESERVE                ; 59 H'0000ec   SCI1 TEI1
77 :          .DATA.L RESERVE                ; 60 H'0000f0   A/D ADI
78 :
79 :  ;=====================================
80 :  ; スタートアッププログラム
81 :  ;=====================================
82 :          .SECTION P
83 :  RESET_START:
84 :          MOV.L   #H'FFF10,ER7           ; スタックの設定
85 :          JSR     @_INITSCT              ; RAMエリアの初期化
86 :          JSR     @_main                 ; Ｃ言語のmain()関数へジャンプ
87 :  OWARI:
88 :          BRA     OWARI
89 :
90 :          .END
```

23.6 プログラムの解説

23.6.1 リフレッシュコントローラとは？

リフレッシュコントローラとは、H8/3048F-ONE と DRAM(Dynamic Random Access Memory:ダイナミック・ランダム・アクセス・メモリ、略してディーラム)または PSRAM(Pseudo SRAM:擬似 SRAM)を接続したときに、DRAM または PSRAM をリフレッシュする機能のことです。詳しくは、インターネットや技術書で調べてください。

リフレッシュコントローラは 3 つの機能があり、どれらかを選んで使用することができます。

機能	詳細
DRAM リフレッシュ制御	DRAM をリフレッシュすることができます。
PSRAM リフレッシュ制御	PSRAM をリフレッシュすることができます。
インターバルタイマ	リフレッシュコントローラをインターバルタイマとして使用することできます。

プロジェクト「timer2」では、ITU による割り込みでタイマの機能を実現していました。ただ、ITU は PWM 機能(モータ制御)やパルスカウント(エンコーダ用)として利用することが多く、割り込みとして使いたくないことがあります。そこで、リフレッシュコントローラをインターバルタイマとして利用し、ITU を別な用途で使えるように空けておきましょう。

23.6.2 リフレッシュコントローラの設定

init 関数内で、リフレッシュコントローラをインターバルタイマとして利用できるよう設定しています。プログラムは下記のとおりです。

```
63 :    RFSHCR = 0x00;                  /* RC=インターバルタイマ     */
64 :    RTCOR  = 191;                   /* 1ms/5.208μs-1＝191        */
65 :    RTMCSR = 0x60;                  /* φ/128、割り込み許可      */
```

●RFSHCR(リフレッシュコントロールレジスタ)の設定内容

RFSHCR は、8 ビットのリード／ライト可能なレジスタで、リフレッシュコントローラの動作モードを選択します。

ビット:	7	6	5	4	3	2	1	0
RFSHCR:	SRFMD	PSRAME	DRAME	CAS/$\overline{WE}$	M9/$\overline{M8}$	RFSHE	-	RCYCE
設定値:	0	0	0	0	0	0	0	0
16 進数:	0				0			

RFSHCR は、リセット、またはハードウェアスタンバイモード時に H'02 にイニシャライズされます。

・ビット 7:セルフリフレッシュモード(SRFMD)

ソフトウェアスタンバイモード時、DRAM または PSRAM のセルフリフレッシュを指定します。

PSRAME＝1、DRAME＝0 のとき、SRFMD ビットを 1 にセットした後に、ソフトウェアスタンバイモードに遷移すると、PSRAM のセルフリフレッシュが可能となります。

また、PSRAME＝0、DRAME＝1 のとき、SRFMD ビットを 1 にセットした後に、ソフトウェアスタンバイモードに遷移すると、DRAM のセルフリフレッシュが可能となります。

いずれの場合もソフトウェアスタンバイモードの解除により、通常のアクセス状態に戻ります。

SRFMD	説明
0	**ソフトウェアスタンバイモード時に、DRAM または PSRAM のセルフリフレッシュを禁止**
1	ソフトウェアスタンバイモード時に、DRAM または PSRAM のセルフリフレッシュが可能

・ビット 6:PSRAM イネーブル(PSRAME)

・ビット 5:DRAM イネーブル(DRAME)

外部アドレス空間のエリア 3 に対して、DRAM または PSRAM の接続を許可／禁止します。

DRAM または PSRAM を接続する場合、エリア 3 のバスサイクルおよびリフレッシュサイクルは ASTCR の設定にかかわらず、3 ステートアクセスとなります。ただし、ウェイトステートは、ASTCR の AST3＝0 の場合、挿入することはできません。

PSRAME ビットまたは DRAME ビットが 1 にセットされていると、RFSHCR のビット 0、2、3、4、および RTMCSR、RTCNT、RTCOR へのライトはできません。ただし、RTMCSR の CMF フラグについては、フラグをクリアするための 0 ライトのみ可能です。

PSRAME	DRAME	説明
0	**0**	**インターバルタイマとして使用可能 (DRAM、PSRAM の直接接続不可能)**
0	1	DRAM の直接接続が可能
1	0	PSRAM の直接接続が可能
1	1	使用禁止

リフレッシュコントローラをインターバルタイマとして使用するので"00"を選択します。

・ビット 4:ストローブモードセレクト(CAS/$\overline{\text{WE}}$)

2$\overline{\text{CAS}}$方式か 2$\overline{\text{WE}}$方式のいずれかを選択します。

本ビットの設定は PSRAME=0、DRAME=1 のとき有効となります。本ビットは、PSRAME ビットまたは DRAME ビットが 1 にセットされているとライトすることはできません。

CAS/$\overline{\text{WE}}$	説明
0	2$\overline{\text{WE}}$方式を選択
1	2$\overline{\text{CAS}}$方式を選択

・ビット 3:アドレスマルチプレクスモードセレクト(M9/$\overline{\text{M8}}$)

8 ビットカラムアドレスまたは 9 ビットカラムアドレスのいずれかを選択します。

本ビットの設定は PSRAME=0、DRAME=1 のとき有効となります。本ビットは、PSRAME ビットまたは DRAME ビットが 1 にセットされているとライトすることはできません。

M9/$\overline{\text{M8}}$	説明
0	8 ビットカラムモードを選択
1	9 ビットカラムモードを選択

・ビット 2:リフレッシュ端子イネーブル(RFSHE)

$\overline{\text{RFSH}}$端子のリフレッシュ信号出力を許可/禁止します。

本ビットは、PSRAME ビットまたは DRAME ビットが 1 にセットされているとライトすることはできません。

RFSHE	説明
0	$\overline{\text{RFSH}}$端子のリフレッシュ信号出力を禁止 ($\overline{\text{RFSH}}$端子は入出力ポートとして使用可)
1	$\overline{\text{RFSH}}$端子のリフレッシュ信号出力を許可

・ビット 1:リザーブビット

リザーブビットです。リードすると常に 1 が読み出されます。ライトは無効です。

・ビット 0:リフレッシュサイクルイネーブル(RCYCE)

リフレッシュサイクルの挿入を許可または禁止します。本ビットは PSRAME=1、または DRAME=1 のときに有効となります。PSRAME=0 かつ DRAME ビット=0 のときは、本ビットの設定にかかわらずリフレッシュサイクルは挿入されません。

RCYCE	説明
0	リフレッシュサイクルを禁止
1	エリア 3 に対するリフレッシュサイクルを許可

●RTMCSR(リフレッシュタイマコントロールステータスレジスタ）の設定内容

RTMCSR は、8 ビットのリード／ライト可能なレジスタで、RTCNT に入力するクロックの選択を行います。また、インターバルタイマとして使用する場合は、割り込み要求の許可／禁止も行います。

ビット:	7	6	5	4	3	2	1	0
RTMCSR:	CMF	CMIE	CKS2	CKS1	CKS0	-	-	-
設定値:	0	1	1	0	0	0	0	0
16 進数:	6				0			

ビット 7、6 は、リセット、またはスタンバイモード時にイニシャライズされます。
ビット 5～3 は、リセット、またはハードウェアスタンバイモード時にイニシャライズされますが、ソフトウェアスタンバイモード時にはソフトウェアスタンバイモードに遷移する前の状態を保持しています。

・ビット 7：コンペアマッチフラグ（CMF）
RTCNT と RTCOR の値が一致したことを示すステータスフラグです。

CMF	説明
0	［クリア条件］ CMF＝1 の状態で、CMF フラグをリードした後、CMF フラグに 0 をライトしたとき
1	［セット条件］ RTCNT＝RTCOR になったとき

・ビット 6：コンペアマッチインタラプトイネーブル（CMIE）
RTCSR の CMF フラグが 1 にセットされたとき、CMF フラグによる割り込み（CMI）要求を許可／禁止します。
PSRAME＝1、または DRAME＝1 のとき、CMIE ビットは常に 0 にクリアされています。

CMIE	説明
0	CMF フラグによる割り込み（CMI）要求を禁止
1	**CMF フラグによる割り込み（CMI）要求を許可**

インターバルタイマを動作させるので、"1"を選択します。このビットを"1"にすることにより、割り込みが許可されます。リフレッシュコントローラのインターバルタイマの割り込みベクタ番号は、21 番になります。

・ビット 5～3:クロックセレクト 2～0(CKS2～CKS0)

RTCNT に入力するクロックを内部クロックから選択します。リフレッシュコントローラとして使用する場合は、RTCNT と RTCOR のコンペアマッチによりリフレッシュ要求を周期的に発生します。インターバルタイマとして使用する場合は、コンペアマッチにより CMI 割り込み要求を周期的に発生します。

本ビットは、PSRAME ビットまたは DRAME ビットが 1 にセットされているとライトすることはできません。

CKS2	CKS1	CKS0	説明
0	0	0	クロック入力禁止
0	0	1	**φ/2** 1パルス＝1÷(24.576×10^6／2)＝81.38[ns] 最大割り込み周期は 81.38[ns]×256＝20.83[μs] 1 秒間に割り込みが 48,000 回実行される計算です。 256 にあたる部分が RTCOR(リフレッシュタイムコンスタントレジスタ)です。0～255 の値を設定することができます。後ほど説明します。 (以下、すべて同じです)
0	1	0	**φ/8** 1パルス＝1÷(24.576×10^6／8)＝325.52[ns] 最大割り込み周期は 325.52[ns]×256＝83.33[μs] 1 秒間に割り込みが 12,000 回実行される計算です。
0	1	1	**φ/32** 1パルス＝1÷(24.576×10^6／32)＝1.302[μs] 最大割り込み周期は 1.302[μs]×256＝333.3[μs] 1 秒間に割り込みが 3,000 回実行される計算です。
1	**0**	**0**	**φ/128** 1パルス＝1÷(24.576×10^6／128)＝5.208[μs] 最大割り込み周期は 5.208[μs]×256＝1.333[ms] 1 秒間に割り込みが 750 回実行される計算です。
1	0	1	**φ/512** 1パルス＝1÷(24.576×10^6／512)＝20.83[μs] 最大割り込み周期は 20.83[μs]×256＝5.333[ms] 1 秒間に割り込みが 187.5 回実行される計算です。
1	1	0	**φ/2048** 1パルス＝1÷(24.576×10^6／2048)＝83.33[μs] 最大割り込み周期は 83.33[μs]×256＝21.33[ms] 1 秒間に割り込みが 46.875 回実行される計算です。
1	1	1	**φ/4096** 1パルス＝1÷(24.576×10^6／4096)＝166.67[μs] 最大割り込み周期は 166.67[μs]×256＝42.67[ms] 1 秒間に割り込みが 23.4375 回実行される計算です。

今回は、割り込みを 1[ms]にします。設定値は、小さいクロックから順番に

最大割り込み周期 ≧ 設定したい割り込み周期

かどうかチェックして、成り立ったクロックにします。

今回は、 "001"→最大割り込み周期 20.83[μs] → 1[ms]以下なので NG

"010"→最大割り込み周期 83.33[μs] → 1[ms]以下なので NG

"011"→最大割り込み周期 333.3[μs] → 1[ms]以下なので NG

"100"→最大割り込み周期 1.333[ms] → **1[ms]以上なので OK**

よって、今回は、最大割り込み周期 1.333[ms]である**"100"**を設定します。

●RTCNT(リフレッシュタイマカウンタ)の設定内容

RTCNT は、リード／ライト可能な 8 ビットのアップカウンタです。

ビット:	7	6	5	4	3	2	1	0
RTCNT:								

RTMCSR(リフレッシュタイマコントロールステータスレジスタ)の CKS2～CKS0 ビットで選択された内部クロックにより、カウントアップを開始します。今回は、"100"を設定しています。これは、RTCNT が 5.208[μs]ごとに+1 する設定です。

●RTCOR(リフレッシュタイムコンスタントレジスタ)の設定内容

RTCOR は、8 ビットのリード／ライト可能なレジスタで、RTCNT とのコンペアマッチ周期を設定します。

ビット:	7	6	5	4	3	2	1	0
RTCOR:								

「RTCNT＝RTCOR+1」になると、割り込みが発生します。
今回は、1ms ごとに割り込みを発生させます。RTMCSR(リフレッシュタイマコントロールステータスレジスタ)の CKS2～CKS0 ビットで選択された内部クロックにより、RTCNT は 5.208[μs]ごとに+1 します。そのため、

RTCOR ＝割り込みを発生させたい周期÷RTCNT のカウント時間－1
＝1[ms]÷5.208[μs]－1
＝192－1
＝191

RTCOR に 191 を設定すれば 1[ms]ごとに割り込みが発生することになります。RTCNT が 191 から 192 になった瞬間に RTCNT は 0 にクリアされ、0 からカウントを続けていきます。

23.6.3 割り込みプログラム

リフレッシュコントローラのインターバルタイマで割り込みがかかったとき、実行される関数は interrupt_rc 関数です。この関数は 1ms ごとに実行されます。

```
68 :  /************************************************************************/
69 :  /* リフレッシュコントローラ 割り込み処理 1.000ms ごと                    */
70 :  /************************************************************************/
71 :  #pragma interrupt( interrupt_rc )
72 :  void interrupt_rc( void )
73 :  {
74 :      RTMCSR &= 0x7f;                     /* フラグリード             */
75 :
76 :      cnt_rc++;
77 :  }
```

74 行で、"1"になっている RTMCSR(リフレッシュタイマコントロールステータスレジスタ)のコンペアマッチフラグ(CMF)を"0"にクリアします。クリア条件は「**CMF＝1 の状態で、CMF フラグをリードした後、CMF フラグに 0 をライトしたとき**」なので、直接"0"を書き込むのでなく、AND 処理して CMF フラグである bit7 をクリアします。
76 行で、cnt_rc 変数を＋1する命令を書いています。よって、cnt_rc 変数の値 1 あたり 1ms となります。

23.6.4 メインプログラム

```
 26 :  /*********************************************************************/
 27 :  /* メインプログラム                                                  */
 28 :  /*********************************************************************/
 29 :  void main( void )
 30 :  {
 31 :      unsigned char d = 0;
 32 :
 33 :      init();                          /* 初期化                   */
 34 :      set_ccr( 0x00 );                 /* 全体割り込み許可         */
 35 :
 36 :      while( 1 ) {
 37 :          if( cnt_rc >= 1000 ) {       /* 1s たったか？            */
 38 :              cnt_rc = 0;
 39 :              PADR = ++d;
 40 :          }
 41 :      }
 42 :  }
```

リフレッシュコントローラのインターバルタイマ割り込みを使用するので、34 行で全体の割り込みを許可します。

他は特に難しいことはなく、37 行目で cnt_rc 変数が 1000 になったなら、すなわち 1000ms たったなら if 文のカッコの中を実行します。カッコの中は、変数 d の値を＋1して、その値をポート A に出力します。

リフレッシュコントローラのインターバルタイマ割り込みを使用するので、34 行で忘れずに全体の割り込みを許可します。

23.6.5 rc_timerstart.srcの追加、変更

```
  1 :  ;=====================================
  2 :  ; 定義
  3 :  ;=====================================
  4 :  RESERVE: .EQU    H'FFFFFFFF              ; 未使用領域のアドレス
  5 :
  6 :  ;=====================================
  7 :  ; 外部参照
  8 :  ;=====================================
  9 :          .IMPORT _main
 10 :          .IMPORT _interrupt_rc
 11 :          .IMPORT _INITSCT
 12 :
 13 :  ;=====================================
 14 :  ; ベクタセクション
 15 :  ;=====================================
 16 :          .SECTION V
 17 :          .DATA.L RESET_START             ;  0 H'000000   ﾘｾｯﾄ
 18 :          .DATA.L RESERVE                 ;  1 H'000004   ｼｽﾃﾑ予約
 19 :          .DATA.L RESERVE                 ;  2 H'000008   ｼｽﾃﾑ予約
 20 :          .DATA.L RESERVE                 ;  3 H'00000c   ｼｽﾃﾑ予約
 21 :          .DATA.L RESERVE                 ;  4 H'000010   ｼｽﾃﾑ予約
 22 :          .DATA.L RESERVE                 ;  5 H'000014   ｼｽﾃﾑ予約
 23 :          .DATA.L RESERVE                 ;  6 H'000018   ｼｽﾃﾑ予約
 24 :          .DATA.L RESERVE                 ;  7 H'00001c   外部割り込み NMI
 25 :          .DATA.L RESERVE                 ;  8 H'000020   ﾄﾗｯﾌﾟ命令
 26 :          .DATA.L RESERVE                 ;  9 H'000024   ﾄﾗｯﾌﾟ命令
 27 :          .DATA.L RESERVE                 ; 10 H'000028   ﾄﾗｯﾌﾟ命令
 28 :          .DATA.L RESERVE                 ; 11 H'00002c   ﾄﾗｯﾌﾟ命令
 29 :          .DATA.L RESERVE                 ; 12 H'000030   外部割り込み IRQ0
 30 :          .DATA.L RESERVE                 ; 13 H'000034   外部割り込み IRQ1
 31 :          .DATA.L RESERVE                 ; 14 H'000038   外部割り込み IRQ2
 32 :          .DATA.L RESERVE                 ; 15 H'00003c   外部割り込み IRQ3
 33 :          .DATA.L RESERVE                 ; 16 H'000040   外部割り込み IRQ4
 34 :          .DATA.L RESERVE                 ; 17 H'000044   外部割り込み IRQ5
 35 :          .DATA.L RESERVE                 ; 18 H'000048   ｼｽﾃﾑ予約
 36 :          .DATA.L RESERVE                 ; 19 H'00004c   ｼｽﾃﾑ予約
 37 :          .DATA.L RESERVE                 ; 20 H'000050   WDT MOVI
 38 :          .DATA.L _interrupt_rc           ; 21 H'000054   REF CMI
 39 :          .DATA.L RESERVE                 ; 22 H'000058   ｼｽﾃﾑ予約
以下、略
```

リフレッシュコントローラのインターバルタイマの割り込みベクタ番号は、21 番になります。38 行の 21 番には割り込みがかかったときに実行する関数「interrupt_rc」を記述します。アセンブリソースプログラム(今回は rc_timerstart.src)から C 言語プログラム(今回は rc_timer.c)の関数を呼ぶときは、関数名の先頭に「__」(アンダバー)を付けなければいけない決まりがあるので、「_interrupt_rc」としています。

10 行は、「_interrupt_rc」が他のファイル(今回は rc_timer.c)にあることを知らせる記述です。

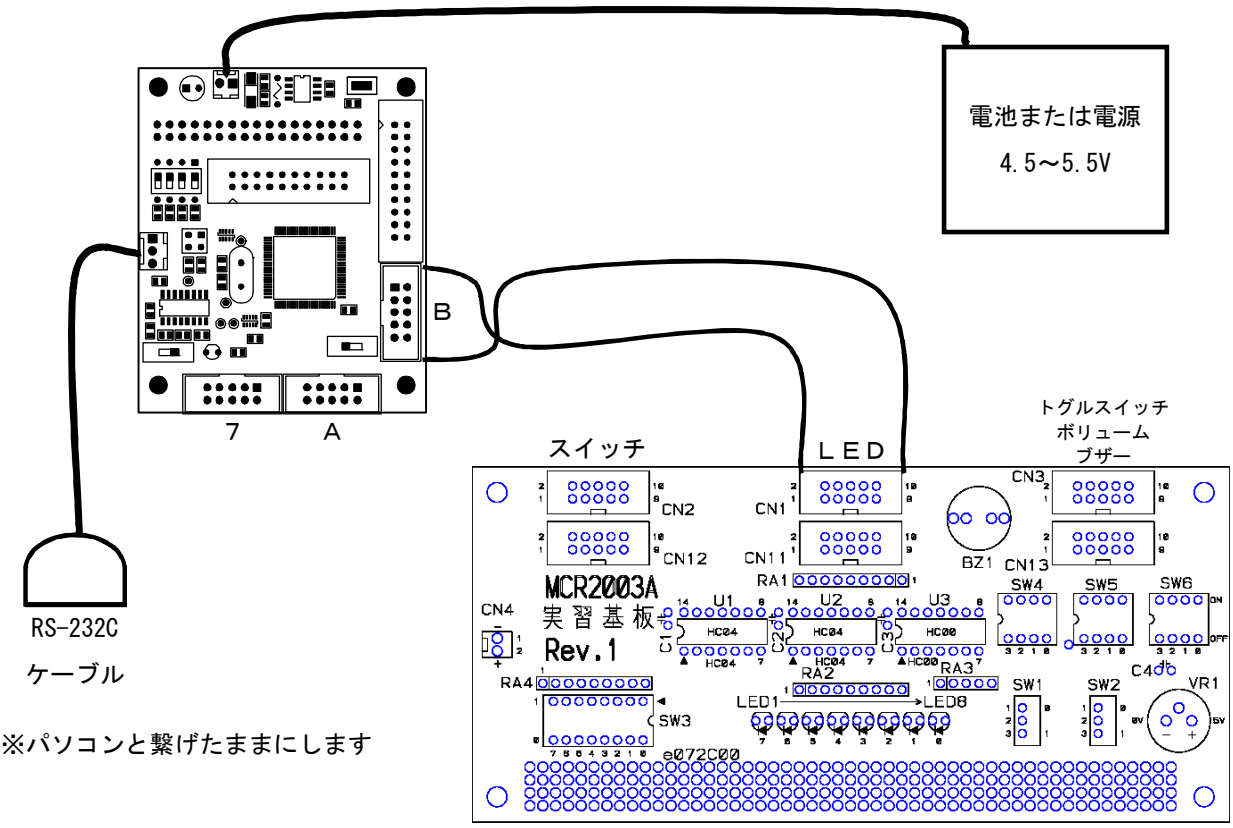
24. プロジェクト「sio」 パソコンから数値を入力してLEDに出力する

24.1 概要

パソコンから入力した数値を LED へ出力します。

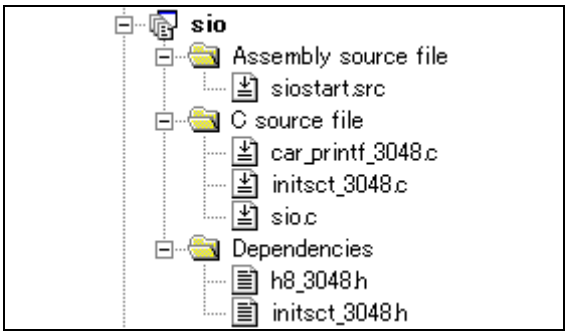
24.2 接続

・CPU ボードのポート B と、実習基板の LED 部をフラットケーブルで接続します。
・通信ケーブルは、パソコンの RS-232C コネクタに接続したままにしておきます。



※パソコンと繋げたままにします

24.3 プロジェクトの構成



	ファイル名	内容
1	siostart.src	アセンブリ言語で記述されたアセンブリソースファイルです。このファイルの構造は下記のようになっています。 ベクタアドレス ＋ スタートアップルーチン
2	initsct_3048.c	初期値のないグローバル変数(セクション B 領域)、初期値のあるグローバル変数(セクション R 領域)の初期化用です。
3	sio.c	実際に制御するプログラムが書かれています。H8/3048F-ONE の内蔵周辺機能の初期化も行います。
4	h8_3048.h	H8/3048F-ONE の内蔵周辺機能の I/O レジスタを定義したファイルです。
5	initsct_3048.h	initsct_3048.c のヘッダファイルです。
6	car_printf_3048.c	・通信するための設定 ・printf 関数の出力先、scanf 関数の入力元を通信にするための設定 を行っています。 プロジェクト「sio」に限らず、printf、scanf 関数を使うプロジェクトは、car_printf_3048.c ファイル追加します。

24.4 プログラム「sio.c」

ゴシック体が、printf、scanf を使うために追加した行です。

```
 1 :  /************************************************************************/
 2 :  /* パソコンから数値を入力してLEDへ出力(通信機能の利用)「sio.c」          */
 3 :  /* 入力：書き込みケーブルをパソコンと接続        出力：PB7-PB0(LED等)    */
 4 :  /* 通信：ﾃﾗﾀｰﾑﾌﾟﾛ/ﾊｲﾊﾟｰﾀｰﾐﾅﾙ  2005.04 ジャパンマイコンカーラリー実行委員会  */
 5 :  /************************************************************************/
 6 :  /*======================================*/
 7 :  /* インクルード                         */
 8 :  /*======================================*/
 9 :  #include    <no_float.h>                /* stdioの簡略化 最初に置く*/
10 :  #include    <stdio.h>
11 :  #include    <machine.h>
12 :  #include    "h8_3048.h"
13 :
14 :  /*======================================*/
15 :  /* プロトタイプ宣言                     */
16 :  /*======================================*/
17 :  void init( void );
18 :
19 :  /*======================================*/
20 :  /* グローバル変数の宣言                 */
21 :  /*======================================*/
22 :
23 :  /************************************************************************/
24 :  /* メインプログラム                                                     */
25 :  /************************************************************************/
```

```
26 :  void main( void )
27 :  {
28 :      int     i, ret;
29 :
30 :      /* マイコン機能の初期化 */
31 :      init();                             /* 初期化                   */
32 :      init_sci1( 0x00, 79 );              /* SCI1初期化               */
33 :      set_ccr( 0x00 );                    /* 全体割り込み許可         */
34 :
35 :      printf( "Hello World!¥n");
36 :      while( 1 ) {
37 :          printf( "Input data : " );
38 :          ret = scanf( "%d", &i );
39 :          if( ret == 1 ) {
40 :              printf( "Get data : %d¥n", i );
41 :              PBDR = i;
42 :          } else {
43 :              printf( "Data Error!!¥n" );
44 :              scanf( "%*[^¥n]" );
45 :          }
46 :      }
47 :  }
48 :
49 :  /************************************************************************/
50 :  /* H8/3048F内蔵モジュール  初期化                                       */
51 :  /************************************************************************/
52 :  void init( void )
53 :  {
54 :      /* ポートの入出力設定 */
55 :      P1DDR = 0xff;
56 :      P2DDR = 0xff;
57 :      P3DDR = 0xff;
58 :      P4DDR = 0xff;
59 :      P5DDR = 0xff;
60 :      P6DDR = 0xf0;                       /* CPU基板上のDIP SW        */
61 :      P8DDR = 0xff;
62 :      P9DDR = 0xf7;
63 :      PADDR = 0xff;
64 :      PBDDR = 0xff;                       /* LED基板                  */
65 :      /* ポート７は、入力専用なので入出力設定はありません   */
66 :  }
67 :
68 :  /************************************************************************/
69 :  /* end of file                                                          */
70 :  /************************************************************************/
```

24.5 プログラム「car_printf_3048.c」

```
 1 :  /************************************************************************/
 2 :  /* マイコンカー用printf,scanf使用プログラム Ver2.1                      */
 3 :  /*                          2006.04 ジャパンマイコンカーラリー実行委員会 */
 4 :  /************************************************************************/
 5 :
 6 :  /*====================================*/
 7 :  /* インクルード                       */
 8 :  /*====================================*/
 9 :  #include    <no_float.h>              /* stdioの簡略化 最初に置く*/
10 :  /*
11 :  printf,scanf文でfloatやdouble型を使わなければ、stdio.hをインクルードする前に
12 :  no_float.hをインクルードすることにより、MOTファイルサイズを小さくすることが
13 :  出来ます。もし、double型を使用するのであれば、インクルードしないでください。
14 :  no_float.hはルネサス統合開発環境でのみ使用出来ます。
15 :  */
16 :  #include    <stdio.h>
17 :  #include    <machine.h>
18 :  #include    "h8_3048.h"
19 :
20 :  /*====================================*/
21 :  /* シンボル定義                       */
22 :  /*====================================*/
23 :  #define         SEND_BUFF_SIZE  64      /* 送信バッファサイズ       */
24 :  #define         RECV_BUFF_SIZE  32      /* 受信バッファサイズ       */
25 :
26 :  /*====================================*/
27 :  /* グローバル変数の宣言               */
28 :  /*====================================*/
29 :  /* 送信バッファ */
30 :  char            send_buff[SEND_BUFF_SIZE];
31 :  char            *send_w = send_buff;
32 :  char            *send_r = send_buff;
33 :  unsigned int    send_count = 0;
34 :
35 :  /* 受信バッファ */
36 :  char            recv_buff[RECV_BUFF_SIZE];
37 :  char            *recv_w = recv_buff;
38 :  char            *recv_r = recv_buff;
```

```
 39 :
 40 :  /* printf、scanfを使う為の変数設定 */
 41 :  unsigned char   sml_buf[4];
 42 :  FILE _iob[] = { { &sml_buf[2],0,&sml_buf[0],3,_IOREAD          ,0,0 },
 43 :                  { &sml_buf[3],0,&sml_buf[3],1,_IOWRITE|_IOUNBUF,0,1 } };
 44 :
 45 :  volatile int    _errno;
 46 :
 47 :  /************************************************************************/
 48 :  /* SCI1の初期化                                                         */
 49 :  /* 引数    ボーレートレジスタ設定値                                     */
 50 :  /* 戻り値 なし                                                          */
 51 :  /************************************************************************/
 52 :  void init_sci1( int smr, int brr )
 53 :  {
 54 :      int i;
 55 :
 56 :      SCI1_SCR = 0x00;
 57 :      SCI1_SMR = smr;
 58 :      SCI1_BRR = brr;
 59 :      for( i=0; i<10000; i++ );
 60 :      SCI1_SCR = 0x30;                    /* 送受信許可                   */
 61 :      SCI1_SSR &= 0x80;                   /* エラーフラグクリア           */
 62 :  }
 63 :
 64 :  /************************************************************************/
 65 :  /* １文字受信                                                           */
 66 :  /* 引数    受信文字格納アドレス                                         */
 67 :  /* 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            */
 68 :  /************************************************************************/
 69 :  int get_sci( char *s )
 70 :  {
 71 :      int i;
 72 :
 73 :      if( SCI1_SSR & 0x38 ) {
 74 :          /* 受信エラー */
 75 :          SCI1_SSR &= 0xc7;
 76 :          return -1;
 77 :      } else if( SCI1_SSR & 0x40 ) {
 78 :          /* 受信有り */
 79 :          *s = SCI1_RDR;
 80 :          SCI1_SSR &= 0xbf;
 81 :          return 1;
 82 :      }
 83 :      /* 受信なし */
 84 :      return 0;
 85 :  }
 86 :  /************************************************************************/
 87 :  /* 送信バッファに保存                                                   */
 88 :  /* 引数    格納文字                                                     */
 89 :  /* 戻り値 なし                                                          */
 90 :  /* メモ    バッファがフルの場合、空くまで待ちます                       */
 91 :  /************************************************************************/
 92 :  void setSendBuff(char c)
 93 :  {
 94 :      /* バッファが空くまで待つ */
 95 :      while( SEND_BUFF_SIZE == send_count );
 96 :
 97 :      or_ccr( 0x80 );                     /* 割り込み禁止                 */
 98 :
 99 :      *send_w++ = c;
100 :      if( send_w >= send_buff+SEND_BUFF_SIZE ) send_w = send_buff;
101 :      send_count++;
102 :
103 :      and_ccr( 0x7f );                    /* 割り込み許可                 */
104 :
105 :      /* 送信割り込み許可 */
106 :      SCI1_SCR |= 0x80;
107 :  }
108 :
109 :  /************************************************************************/
110 :  /* 送信割り込み                                                         */
111 :  /* 引数    なし                                                         */
112 :  /* 戻り値 なし                                                          */
113 :  /************************************************************************/
114 :  #pragma interrupt (intTXI1)
115 :  void intTXI1( void )
116 :  {
117 :      /* 送信データをレジスタにセット */
118 :      SCI1_TDR = *send_r++;
119 :      if( send_r >= send_buff+SEND_BUFF_SIZE ) send_r = send_buff;
120 :
121 :      /* 送信開始 */
122 :      SCI1_SSR &= 0x7f;
123 :
124 :      /* これが最後のデータなら次以降の割り込み禁止 */
125 :      send_count--;
126 :      if( !send_count ) SCI1_SCR &= 0x7f;
127 :  }
128 :
129 :  /************************************************************************/
```

```
130 :  /* printfで呼び出される関数                                                */
131 :  /* ユーザーからは呼び出せません                                            */
132 :  /************************************************************************/
133 :  char *sbrk(size_t size)
134 :  {
135 :      return (char *)-1;
136 :  }
137 :
138 :  /************************************************************************/
139 :  /* printfで呼び出される関数                                                */
140 :  /* ユーザーからは呼び出せません                                            */
141 :  /************************************************************************/
142 :  int write(int fileno,char *buf,unsigned int cnt)
143 :  {
144 :      int i;
145 :      static int (*func)(const char *,...) = printf;
146 :
147 :      if( !cnt ) return 0;
148 :
149 :      if( *buf == '¥n' )  {
150 :          setSendBuff( '¥r' );
151 :      } else if( *buf == '¥b' ) {
152 :          setSendBuff( '¥b' );
153 :          setSendBuff( ' ' );
154 :      }
155 :      setSendBuff( *buf );
156 :      return cnt;
157 :  }
158 :
159 :  /************************************************************************/
160 :  /* scanfで呼び出される関数                                                 */
161 :  /* ユーザーからは呼び出せません                                            */
162 :  /************************************************************************/
163 :  int read(int fileno,char *buf,unsigned int cnt)
164 :  {
165 :      static int (*func)(const char *,...) = scanf;
166 :
167 :      if( !cnt ) return 0;
168 :
169 :      if( recv_r == recv_w ) {
170 :          do {
171 :              /* 受信待ち */
172 :              while( !(SCI1_SSR & 0x40) )
173 :              SCI1_SSR &= 0xc0;
174 :              *buf = SCI1_RDR;
175 :              SCI1_SSR &= 0xbf;
176 :
177 :              switch( *buf ) {
178 :                  case '¥b':  /* バックスペース */
179 :                      /* 何もバッファにないならBSは無効 */
180 :                      if( recv_r == recv_w ) continue;
181 :                      /* あるなら一つ戻る */
182 :                      recv_w--;
183 :                      break;
184 :                  case '¥r':  /* Enterキー */
185 :                      *recv_w++ = *buf = '¥n';
186 :                      *recv_w++ = '¥r';
187 :                      break;
188 :                  default:
189 :                      if( recv_w >= recv_buff+RECV_BUFF_SIZE-2 ) continue;
190 :                      *recv_w++ = *buf;
191 :                      break;
192 :              }
193 :              /* エコーバック 入力された文字を返す */
194 :              write( fileno, buf, cnt );
195 :          } while( *buf != '¥n' );
196 :      }
197 :      *buf = *recv_r++;
198 :      if( recv_r == recv_w ) recv_r = recv_w = recv_buff;
199 :
200 :      return 1;
201 :  }
202 :
203 :  /************************************************************************/
204 :  /* end of file                                                          */
205 :  /************************************************************************/
```

24.6 プログラム「siostart.src」

ゴシック体が、printf、scanfを使うために追加、変更した行です。

```
1 :  ;======================================
2 :  ; 定義
3 :  ;======================================
4 :  RESERVE: .EQU    H'FFFFFFFF              ; 未使用領域のアドレス
```

```
 5 :
 6 : ;====================================
 7 : ; 外部参照
 8 : ;====================================
 9 :         .IMPORT _main
10 :         .IMPORT _intTXI1
11 :         .IMPORT _INITSCT
12 :
13 : ;====================================
14 : ; ベクタセクション
15 : ;====================================
16 :         .SECTION V
17 :         .DATA.L         RESET_START             ;  0 H'000000   ﾘｾｯﾄ
18 :         .DATA.L         RESERVE                 ;  1 H'000004   ｼｽﾃﾑ予約
19 :         .DATA.L         RESERVE                 ;  2 H'000008   ｼｽﾃﾑ予約
20 :         .DATA.L         RESERVE                 ;  3 H'00000c   ｼｽﾃﾑ予約
21 :         .DATA.L         RESERVE                 ;  4 H'000010   ｼｽﾃﾑ予約
22 :         .DATA.L         RESERVE                 ;  5 H'000014   ｼｽﾃﾑ予約
23 :         .DATA.L         RESERVE                 ;  6 H'000018   ｼｽﾃﾑ予約
24 :         .DATA.L         RESERVE                 ;  7 H'00001c   外部割り込み NMI
25 :         .DATA.L         RESERVE                 ;  8 H'000020   ﾄﾗｯﾌﾟ命令
26 :         .DATA.L         RESERVE                 ;  9 H'000024   ﾄﾗｯﾌﾟ命令
27 :         .DATA.L         RESERVE                 ; 10 H'000028   ﾄﾗｯﾌﾟ命令
28 :         .DATA.L         RESERVE                 ; 11 H'00002c   ﾄﾗｯﾌﾟ命令
29 :         .DATA.L         RESERVE                 ; 12 H'000030   外部割り込み IRQ0
30 :         .DATA.L         RESERVE                 ; 13 H'000034   外部割り込み IRQ1
31 :         .DATA.L         RESERVE                 ; 14 H'000038   外部割り込み IRQ2
32 :         .DATA.L         RESERVE                 ; 15 H'00003c   外部割り込み IRQ3
33 :         .DATA.L         RESERVE                 ; 16 H'000040   外部割り込み IRQ4
34 :         .DATA.L         RESERVE                 ; 17 H'000044   外部割り込み IRQ5
35 :         .DATA.L         RESERVE                 ; 18 H'000048   ｼｽﾃﾑ予約
36 :         .DATA.L         RESERVE                 ; 19 H'00004c   ｼｽﾃﾑ予約
37 :         .DATA.L         RESERVE                 ; 20 H'000050   WDT MOVI
38 :         .DATA.L         RESERVE                 ; 21 H'000054   REF CMI
39 :         .DATA.L         RESERVE                 ; 22 H'000058   ｼｽﾃﾑ予約
40 :         .DATA.L         RESERVE                 ; 23 H'00005c   ｼｽﾃﾑ予約
41 :         .DATA.L         RESERVE                 ; 24 H'000060   ITU0 IMIA0
42 :         .DATA.L         RESERVE                 ; 25 H'000064   ITU0 IMIB0
43 :         .DATA.L         RESERVE                 ; 26 H'000068   ITU0 OVI0
44 :         .DATA.L         RESERVE                 ; 27 H'00006c   ｼｽﾃﾑ予約
45 :         .DATA.L         RESERVE                 ; 28 H'000070   ITU1 IMIA1
46 :         .DATA.L         RESERVE                 ; 29 H'000074   ITU1 IMIB1
47 :         .DATA.L         RESERVE                 ; 30 H'000078   ITU1 OVI1
48 :         .DATA.L         RESERVE                 ; 31 H'00007c   ｼｽﾃﾑ予約
49 :         .DATA.L         RESERVE                 ; 32 H'000080   ITU2 IMIA2
50 :         .DATA.L         RESERVE                 ; 33 H'000084   ITU2 IMIB2
51 :         .DATA.L         RESERVE                 ; 34 H'000088   ITU2 OVI2
52 :         .DATA.L         RESERVE                 ; 35 H'00008c   ｼｽﾃﾑ予約
53 :         .DATA.L         RESERVE                 ; 36 H'000090   ITU3 IMIA3
54 :         .DATA.L         RESERVE                 ; 37 H'000094   ITU3 IMIB3
55 :         .DATA.L         RESERVE                 ; 38 H'000098   ITU3 OVI3
56 :         .DATA.L         RESERVE                 ; 39 H'00009c   ｼｽﾃﾑ予約
57 :         .DATA.L         RESERVE                 ; 40 H'0000a0   ITU4 IMIA4
58 :         .DATA.L         RESERVE                 ; 41 H'0000a4   ITU4 IMIB4
59 :         .DATA.L         RESERVE                 ; 42 H'0000a8   ITU4 OVI4
60 :         .DATA.L         RESERVE                 ; 43 H'0000ac   ｼｽﾃﾑ予約
61 :         .DATA.L         RESERVE                 ; 44 H'0000b0   DMAC DEND0A
62 :         .DATA.L         RESERVE                 ; 45 H'0000b4   DMAC DEND0B
63 :         .DATA.L         RESERVE                 ; 46 H'0000b8   DMAC DEND1A
64 :         .DATA.L         RESERVE                 ; 47 H'0000bc   DMCA DEND1B
65 :         .DATA.L         RESERVE                 ; 48 H'0000c0   ｼｽﾃﾑ予約
66 :         .DATA.L         RESERVE                 ; 49 H'0000c4   ｼｽﾃﾑ予約
67 :         .DATA.L         RESERVE                 ; 50 H'0000c8   ｼｽﾃﾑ予約
68 :         .DATA.L         RESERVE                 ; 51 H'0000cc   ｼｽﾃﾑ予約
69 :         .DATA.L         RESERVE                 ; 52 H'0000d0   SCI0 ERI0
70 :         .DATA.L         RESERVE                 ; 53 H'0000d4   SCI0 RXI0
71 :         .DATA.L         RESERVE                 ; 54 H'0000d8   SCI0 TXI0
72 :         .DATA.L         RESERVE                 ; 55 H'0000dc   SCI0 TEI0
73 :         .DATA.L         RESERVE                 ; 56 H'0000e0   SCI1 ERI1
74 :         .DATA.L         RESERVE                 ; 57 H'0000e4   SCI1 RXI1
75 :         .DATA.L         _intTXI1                ; 58 H'0000e8   SCI1 TXI1
76 :         .DATA.L         RESERVE                 ; 59 H'0000ec   SCI1 TEI1
77 :         .DATA.L         RESERVE                 ; 60 H'0000f0   A/D ADI
78 :
79 : ;====================================
80 : ; スタートアッププログラム
81 : ;====================================
82 :         .SECTION P
83 : RESET_START:
84 :         MOV.L   #H'FFF10,ER7            ; スタックの設定
85 :         JSR     @_INITSCT               ; RAMエリアの初期化
86 :         JSR     @_main                  ; C言語のmain()関数へジャンプ
87 : OWARI:
88 :         BRA     OWARI
89 :
90 :         .END
```

24.7 パソコンとマイコンの通信

24.7.1 概要

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ディスプレイに、

```
Hello World!
```

と表示されます。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。

しかしマイコンを使用する装置は、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 という規格に準じているため printf や scanf などの関数を実行できます。しかし、出力先や入力元が無い(分からない)ため、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！！

24.7.2 パソコンとの接続方法

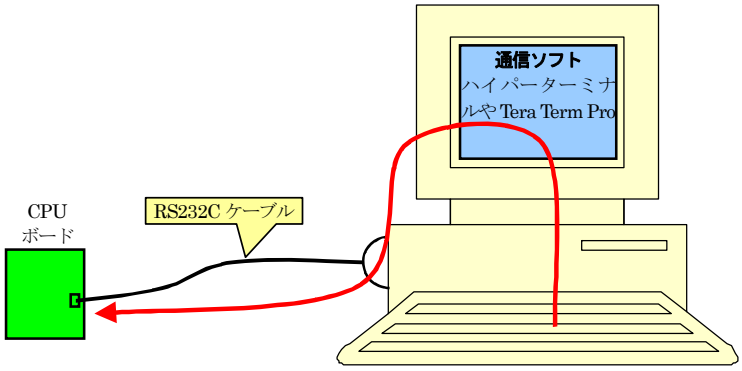
といっても、CPU ボードとディスプレイやキーボードを簡単に接続することはできません。接続する回路や制御プログラムが大変になります。

CPU ボードにプログラムを書き込むときを思い出してみます。CPU ボードとパソコンを RS232C ケーブルで接続します。このケーブル通して、パソコンから CPU ボードへプログラムを書き込んでいました。この RS232C ケーブルを利用して CPU ボードとパソコンを繋ぎます。

パソコンには、「ハイパーターミナル」や「Tera Term Pro」などの通信ソフトを立ち上げておきます。

(1) キーボードから入力したデータをCPUボードで受信

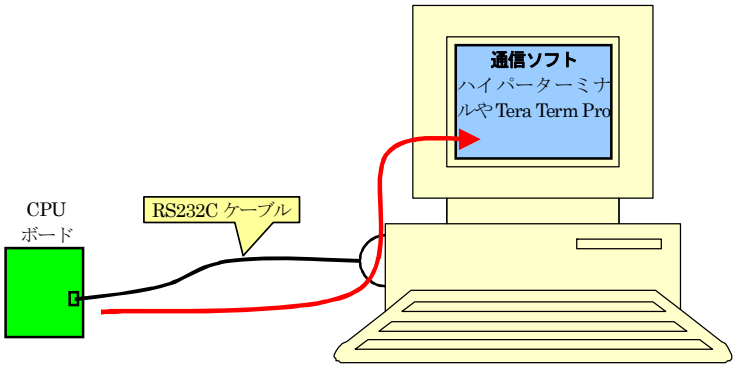
通信ソフトは、キーボードから入力された文字を、RS232C ケーブルを通じて CPU ボードへ送ります(下図)。



キーボード→通信ソフト→RS232C ケーブル→CPU ボードへ

(2) CPUボードから送ったデータを、パソコンに表示

通信ソフトは、CPU ボードから送られてきたデータを RS232C ケーブルを通じて受信して、通信ソフトの画面上に表示します(下図)。



CPU ボード→RS232C ケーブル→通信ソフトの画面へ

今回は、キーボードから 10 進数でデータを入力します。CPU ボードはそのデータを受信して、ポート B に接続されている LED 基板へそのデータを出力します。

24.8 プログラムの解説

24.8.1 car_printf_3048.cとは？

プロジェクト「sio」は、ヘッダファイルを除くと 4 ファイルで構成されています。初めてCソースファイルが 4 つになりました。追加された「car_printf_3048.c」というファイルは何のために追加されたのでしょうか。4 ファイルそれぞれの主な役割を下記に書いてみます。

ファイル名	内容
siostart.src	ベクタアドレスの設定、スタートアップルーチンが含まれるファイルです。
sio.c	C言語のメインプログラムファイルです。
initsct_3048.c	初期値のないグローバル変数(セクション B 領域)、初期値のあるグローバル変数(セクション R 領域)の初期化用です。
car_printf_3048.c	**・通信するための設定 ・printf 関数の出力先、scanf 関数の入力元を通信にするための設定 を行っています。 プロジェクト「sio」に限らず、これらを行うプロジェクトには、car_printf_3048.c ファイルを追加します。**

car_printf_3048.c を追加するだけでは、printf や scanf 関数は動作しません。sio.c、siostart.src ファイルにも手を加える必要があります。

24.8.2 siostart.srcファイル IMPORT追加とベクタアドレスの変更

```
 6 :  ;=====================================
 7 :  ; 外部参照
 8 :  ;=====================================
 9 :          .IMPORT _main
10 :          .IMPORT _intTXI1
11 :          .IMPORT _INITSCT
```

外部参照部分に「**_intTXI1**」を追加します。

```
75 :          .DATA.L         _intTXI1                ; 58 H'0000e8   SCI1 TXI1
```

ベクタ番号 58 のジャンプ先を「_intTXI1」にします。printf 文で文字を送信するとき、通信の送信割り込みを使用します。

24.8.3 sio.cファイル include文追加

```
 6 :  /*====================================*/
 7 :  /* インクルード                       */
 8 :  /*====================================*/
 9 :  #include    <no_float.h>              /* stdioの簡略化 最初に置く*/
10 :  #include    <stdio.h>
11 :  #include    <machine.h>
12 :  #include    "h8_3048.h"
```

printf関数、scanf関数を実行するために、stdio.hファイルをインクルードします。これはパソコンなどでも同様です。stdio.hをインクルードする前に、no_float.hファイルをインクルードしています。このファイルは、**ルネサス統合開発環境のみで有効なヘッダファイルです**。詳細は、「24.8.7 no_float.hファイルとは」を参照してください。

24.8.4 sio.cファイル init_sci1 関数の追加

```
23 :  /*****************************************************************************/
24 :  /* メインプログラム                                                          */
25 :  /*****************************************************************************/
26 :  void main( void )
27 :  {
28 :      int     i, ret;
29 :
30 :      /* マイコン機能の初期化 */
31 :      init();                           /* 初期化                  */
32 :      init_sci1( 0x00, 79 );            /* SCI1 初期化              */
33 :      set_ccr( 0x00 );                  /* 全体割り込み許可        */
```

H8/3048F-ONE の内蔵周辺機能である SCI1 を初期化する関数を実行します。カッコ内の「0x00」と「79」でボーレート(通信スピード)の設定しています。通信スピードは通常、1200bps、4800bps、9600bps、19200bps、38400bps を設定します。

今回は、9600bps に設定したいとします。

```
init_sci1(SMR, BRR );
```

SMR と BRR の計算方法は、下記の手順です。

BRR＝φ÷(A×設定したいボーレート)－1

※A＝32,128,512,2048 のどれか
※φ＝クリスタルの値、RY3048Fone ボードは 24.576MHz

A の値は 4 種類あるので、4 とおり計算します。φ＝24.576MHz、設定したいボーレートは 9600 ですので、代入すると、

A＝32 のとき → BRR＝$24.576\times10^6\div(32\times9600)-1$ ＝79.00
A＝128 のとき → BRR＝$24.576\times10^6\div(128\times9600)-1$ ＝19.00
A＝512 のとき → BRR＝$24.576\times10^6\div(512\times9600)-1$ ＝4.00
A＝2048 のとき → BRR＝$24.576\times10^6\div(2048\times9600)-1$ ＝0.25

init_sci1 関数に値を設定しようとしても4種類もあり、どれを設定すればよいか困ってしまいます。選定基準は、

(1) BRR の値が 0～255 の範囲である設定値を選びます。
(2) 整数の(小数点のない)設定値を選びます。
(3) 1と2に当てはまる設定値が 2 つ以上ある場合は、A の値が小さい設定を選びます。
(4) 1と2に当てはまる設定値が無い場合、BRR を四捨五入した値が「0～255」の範囲内であることを条件に、下記の計算を行います。

実際の通信速度＝φ÷(A×(BRR を四捨五入した値＋1))

この計算で、計算された「実際の通信速度」と「設定したい通信速度」の差が一番小さい A の値を使います。差が同じ場合は、A の値が小さい設定を選びます。

ちなみに

誤差率 ＝ (実際の通信速度－設定したい通信速度)/ 設定したい通信速度

が1％以上ある場合は、通信エラーが出る可能性があります。その場合は、通信速度を変えるか、クリスタルの値を変更するなどしてください。

今回は、
(1)の条件→すべて可能
(2)の条件→A が 32、128、512 のとき可能、2048 は不可
(3)の条件→A の値が一番小さい A=32、BRR=79
が設定値となります。

A の値を直接 init_sci1 関数に設定するわけではありません。下表のような SMR 値となります。

A の値	SMR の設定値
32	0x00
128	0x01
512	0x02
2048	0x03

A=32 なので、SMR は 0x00 となります。

```
init_sci1( 0x00, 79 );              /* SCI1 初期化                  */
           ↑    ↑
          SMR  BRR
```

他のボードの例として下記の2種類の例を取り上げます。

●RY3048Fボードを使用

クリスタル値は 14.7456MHz です。設定したいボーレートを 9600bps とすると、

A＝32 のとき → BRR＝$14.7456\times10^6\div(32\times9600)-1$ ＝47.00
A＝128 のとき → BRR＝$14.7456\times10^6\div(128\times9600)-1$ ＝11.00
A＝512 のとき → BRR＝$14.7456\times10^6\div(512\times9600)-1$ ＝2.00
A＝2048 のとき → BRR＝$14.7456\times10^6\div(2048\times9600)-1$ ＝-0.25

(1)の条件→A が 2048 のとき不可
(2)の条件→A が 32、128、512 のとき可能
(3)の条件→A の値が一番小さい A=32、BRR=47、よって SMR=0x00
が設定値となります。

●AKI-H8 ボードを使用

クリスタル値は 16.000MHz です。設定したいボーレートを 9600bps とすると、

A＝32 のとき	→	BRR＝16.000×10^6÷(32×9600)－1	＝51.08	→四捨五入すると 51
A＝128 のとき	→	BRR＝16.000×10^6÷(128×9600)－1	＝12.02	→四捨五入すると 12
A＝512 のとき	→	BRR＝16.000×10^6÷(512×9600)－1	＝2.26	→四捨五入すると 2
A＝2048 のとき	→	BRR＝16.000×10^6÷(2048×9600)－1	＝-0.17	→設定不可

(1)の条件→A が 2048 のとき不可
(2)の条件→A が 32、128、512 のときのどれも不可
4 種類どれも不可なので、(4)の条件が適用されます。

(4)の条件→ A＝32 のとき	→	ボーレート＝16.000×10^6÷(32×(51+1))	＝9615.4
A＝128 のとき	→	ボーレート＝16.000×10^6÷(128×(12+1))	＝9615.4
A＝512 のとき	→	ボーレート＝16.000×10^6÷(2048×(2+1))	＝10416.7

となります。A=32,128 どちらも同じですので、A の値が少ない方を選びます。A=32、BRR=51、よって SMR=0x00 が設定値となります。
ちなみに誤差率は、
誤差率 ＝ (実際の通信速度－設定したい通信速度)/ 設定したい通信速度
＝(9615.4-9600)/9600 ＝ 0.16%
となります。1%以下なので問題ありません。

24.8.5 sio.cファイル printf文、scanf文の使用

これまでの追加で、printf 関数、scanf 関数を使用するための準備が完了しました。後は、main 関数内でパソコンのC言語のように printf 関数、scanf 関数を使用することができます。

```
35 :     printf( "Hello World!¥n");
36 :     while( 1 ) {
37 :         printf( "Input data : " );
38 :         ret = scanf( "%d", &i );
39 :         if( ret == 1 ) {
40 :             printf( "Get data : %d¥n", i );
41 :             PBDR = i;
42 :         } else {
43 :             printf( "Data Error!!¥n" );
44 :             scanf( "%*[^¥n]" );
45 :         }
46 :     }
```

35 行で最初に「Hello World!」と表示します。
36 行は、while 文のカッコ内は常に真なので無限ループです。
37 行で「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。
38 行で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。
39 行で scanf 関数の戻り値をチェックします。
40、41 行で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 B へ出力します。
43、44 行でデータが異常なら、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。
44 行の意味は、「**24.8.6 scanf文の文字列の意味**」を参照してください。

※printf 関数については、「**28.5 printf 関数の使い方**」を参照してください。
※scanf 関数については、「**28.6 scanf 関数の使い方**」を参照してください。

24.8.6 scanf文の文字列の意味

44 行の scanf 文は、バッファのクリア部分でこの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。

総合すると、バッファを読み捨てますが、「¥n」のみ読み捨てるのを飛ばします。すなわち「¥n」のみが残ります。「¥n」により scanf 関数はバッファの終了と判断して次へ進みます。
もう一度、最初の入力部分の scanf 関数を見ると、

```
38 :          ret = scanf( "%d", &i );
```

ここで、「aaa(改行)」を入力したとします(改行＝エンタ)。入力バッファには、

```
aaa¥n
```

と文字が保存されます。改行が入力されると変換が開始されます。変換指定文字「%d」は、10 進数入力です。それ以外のアルファベットなどの文字が入力されてしまうと「**28.6 scanf 関数の使い方**」の(D)の説明のとおり、文字を読み込まずに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 );
```

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され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。

```
39 :          if( ret == 1 ) {
40 :              printf( "Get data : %d¥n", i );
41 :              PBDR = 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 B へその値を出力します。

```
42 :          } else {
43 :              printf( "Data Error!!¥n" );
44 :              scanf( "%*[^¥n]" );
```

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。本によっては 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。

24.8.7 no_float.hファイルとは？

このファイルは、ルネサス統合開発環境専用のヘッダファイルです。**printf 文、scanf 文で float 型や double 型を使わなければ、stdio.h をインクルードする前に no_float.h をインクルードすることにより、MOT ファイルサイズを小さくすることができます**。もし、float 型や double 型を使用するのであれば、no_float.h は入れないでください。
注意点は、すべてのCソースプログラムファイルで入れるなら入れる、入れないなら入れないと統一してください。今回は、「sio.c」と「car_printf2.c」ファイルの 2 ファイルになります。

sio.c のインクルード部分です。

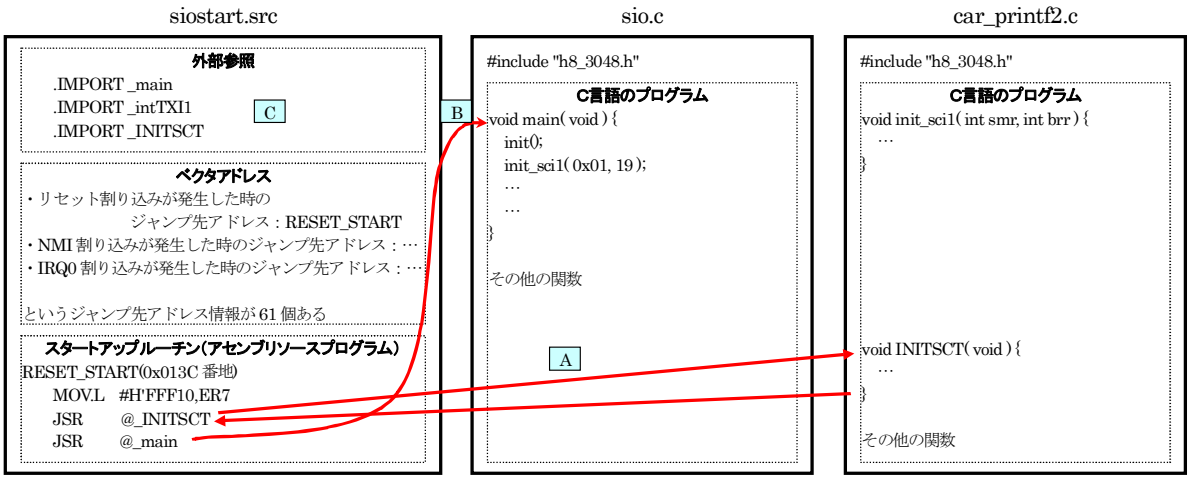
```
 6 :  /*====================================*/
 7 :  /* インクルード                       */
 8 :  /*====================================*/
 9 :  #include    <no_float.h>                /* stdio の簡略化 最初に置く*/
10 :  #include    <stdio.h>
11 :  #include    <machine.h>
12 :  #include    "h8_3048.h"
```

car_printf_3048.c のインクルード部分です。

```
 6 :  /*====================================*/
 7 :  /* インクルード                       */
 8 :  /*====================================*/
 9 :  #include    <no_float.h>                /* stdio の簡略化 最初に置く*/
10 :  /*
11 :  printf, scanf 文で float や double 型を使わなければ、stdio.h をインクルードする前に
12 :  no_float.h をインクルードすることにより、MOT ファイルサイズを小さくすることが
13 :  出来ます。もし、double 型を使用するのであれば、インクルードしないでください。
14 :  no_float.h はルネサス統合開発環境でのみ使用出来ます。
15 :  */
16 :  #include    <stdio.h>
17 :  #include    <machine.h>
18 :  #include    "h8_3048.h"
```

printf 文、scanf 文で float 型や double 型を使わなければ、上記のように no_float.h を 2 ファイルに追加します。使う場合は、両ファイルとも「**#include <no_float.h>**」を削除します。

24.9 まとめ



A…アセンブリソースプログラムから、「car_printf2.c」にある INITSCT 関数を呼んでいます。
B…アセンブリソースプログラムから、「sio.c」にある main 関数を呼んでいます。
C…アセンブリソースプログラムからCソースファイルにある関数を呼ぶとき、外部から探してくださいという宣言をします。「sio.c」にある関数でも「car_printf2.c」にある関数でも、「.IMPORT」命令でOKです。リンク時にリンカが自動で探してくれます。

●ポイント

・アセンブリソースプログラムからCソースプログラムの関数を呼ぶ場合(A,B 部分)は、「.IMPORT」命令で外部に関数があることを知らせる必要があります(C 部分)。

24.10 演習手順

24.10.1 パソコン設定する前準備

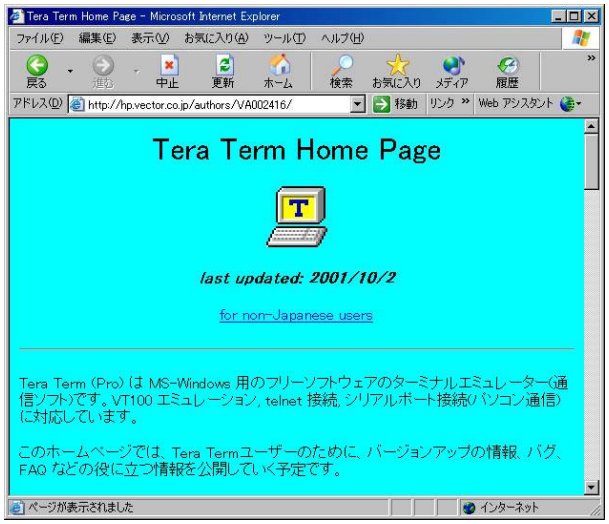
これからパソコンの設定をします。その前に下記作業をあらかじめ済ませておきます。

・H8 マイコンに「sio.mot」ファイルを書き込みます。書き込みが終了したら、CPU ボードの電源は切っておきます(書き込みスイッチも元に戻しておいてください)。

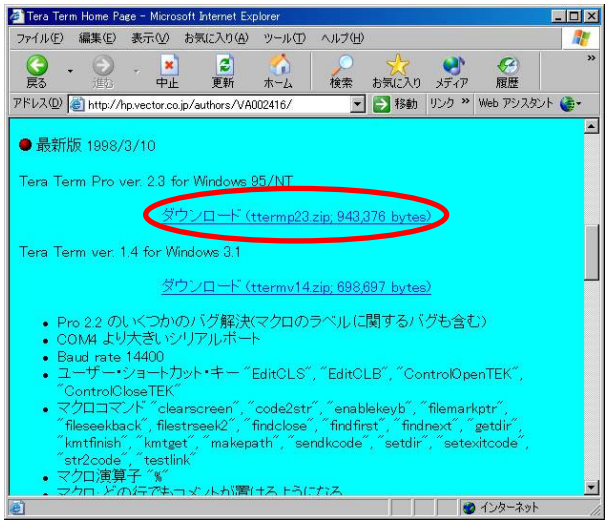
・通信ケーブルは接続したままにしておきます。

24.10.2 TeraTermProのインストール

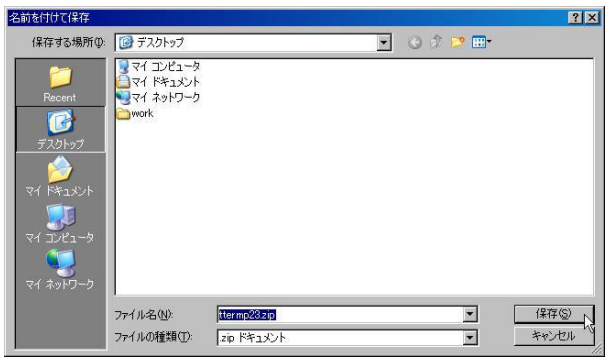
通信ソフトとして、ハイパーターミナルというソフトがあります。ハイパーターミナルは、Windows 標準で入っているためインストール不要で利用できます。しかし、古い Windows では入っていないことがある、通信できないことがある、など不具合が発生しやすいソフトでもあります。そこで、フリーソフトで通信のできる**「Tera Term Pro」**というソフトを使用します。



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://hp.vector.co.jp/authors/VA002416/
   を開きます。
   または、講習会 CD がある場合は、
   CD ドライブ→関連ソフト→ttermp23→setup.exe
   を実行してください。その場合、7 へ進んでください。



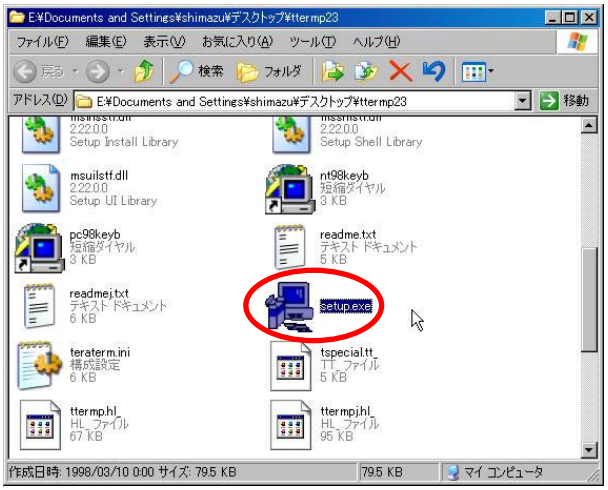
2. 下の方に「ダウンロード (ttermp23.zip; 943,376 bytes)」とあるので、クリックして保存します。



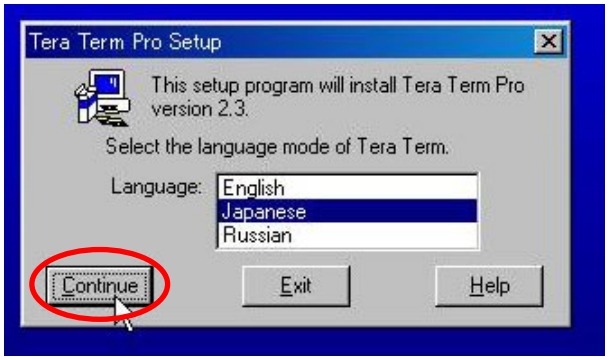
3. 保存は何処でも良いですが、ここではディスクトップに保存します。

4. 保存されました。

5. ttermp23.zip は ZIP 形式で圧縮された形式なので、解凍します。解凍ソフトは、フリーソフトでたくさんありますので、インターネットなどで探してください。図は、ttermp23 というフォルダに解凍したところです。



6. 解凍後のファイルです。setup.exe を実行します。



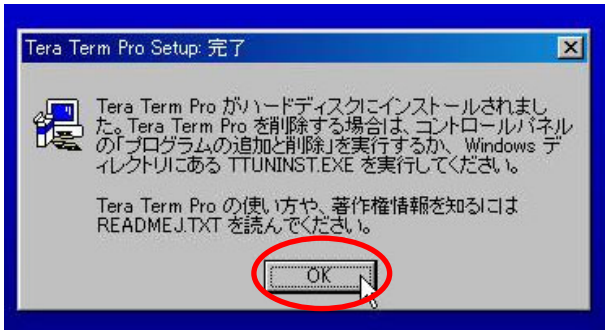
7. 言語の選択です。日本になっていますのでそのまま Continue(続ける)をクリックします。

8. 注意が出ます。Continue(続ける)をクリックします。

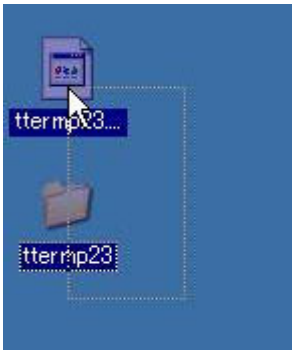
9. キーボードの選択です。Continue(続ける)をクリックします。

10. インストール先を確認して、問題なければ Continue(続ける)をクリックします。インストールが開始されます。

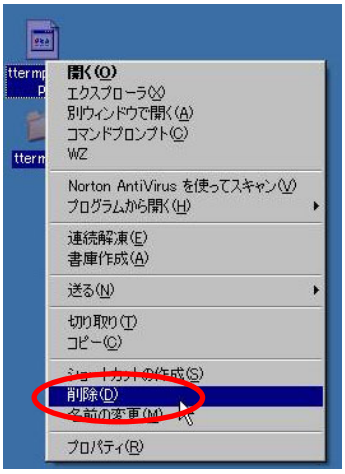
11. メニューが立ち上がります。×をクリックして閉じます。



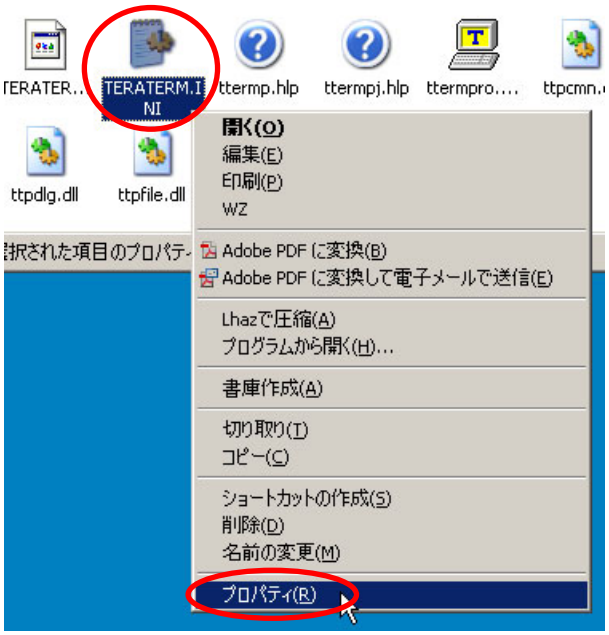
12. OK をクリックして、インストールを完了します。

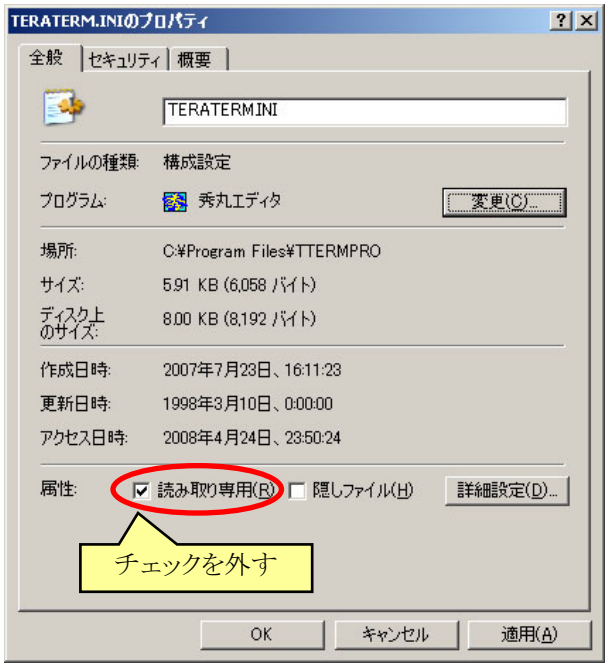


13. インターネットからファイルをダウンロードした場合、ダウンロードファイルを削除します。ttermp23.zip と ttermp23 フォルダを選択します。CD からインストールした場合は不要です。



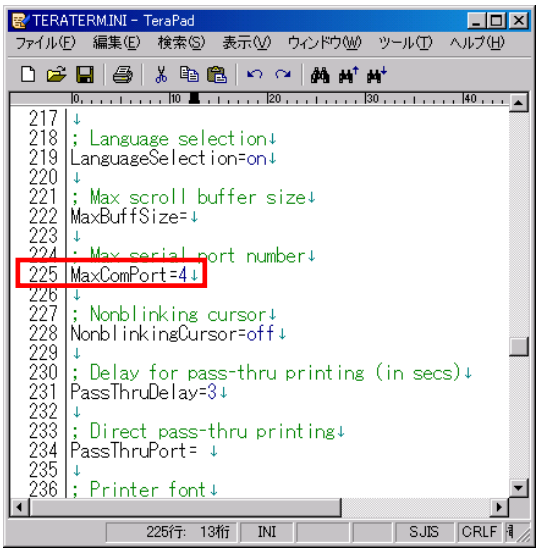
14. どちらかのファイルの上で右クリックして「削除」をクリック、ファイルを削除します。





15. 標準では COM(通信)ポートが 1～4 までしか選択できません。USB-232C 変換を使ったときなど、通信ポートの番号が COM5 以上になることがあります。そのため、COM5 以上も選択できるように変えておきましょう。
「C ドライブ→Program Files→TTERMPRO→TERATERM.INI」を右クリック、「プロパティ」を選択します。

16. 「読み取り専用」のチェックが付いている場合、チェックを外します。



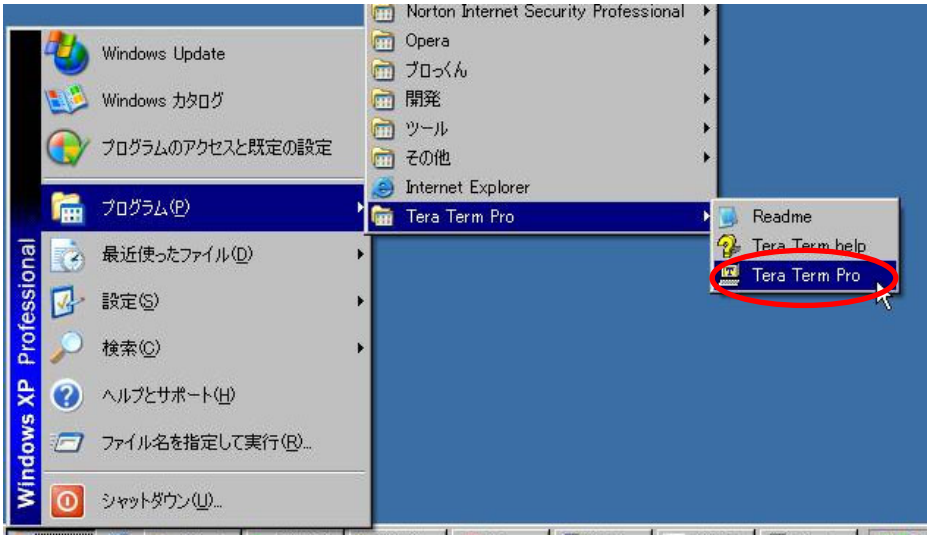
17. 再度、「TERATERM.INI」を右クリックし、今度はファイルを開きます。

18. 225 行に

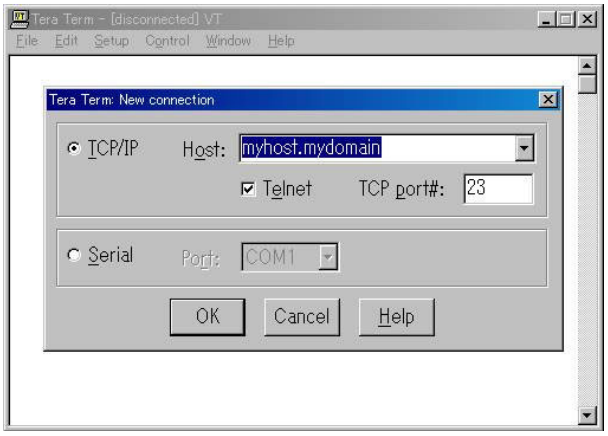
MaxComPort=4

とあります。この「4」という数字が COM 番号の最大値です。
この数値を「**16**」に書き換えておきましょう。
COM16 まで開くことができます(TeraTermPro は 16 が最大です)。
上書き保存して、完了です。

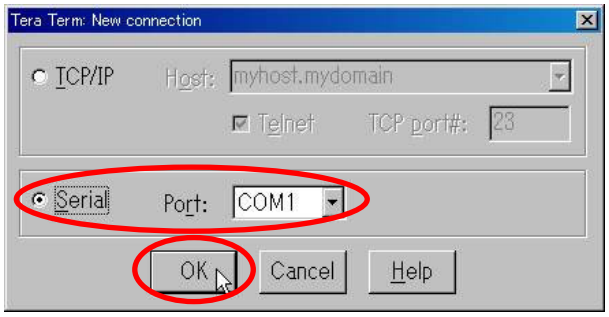
24.10.3 TeraTermProを使用したマイコンーパソコン通信



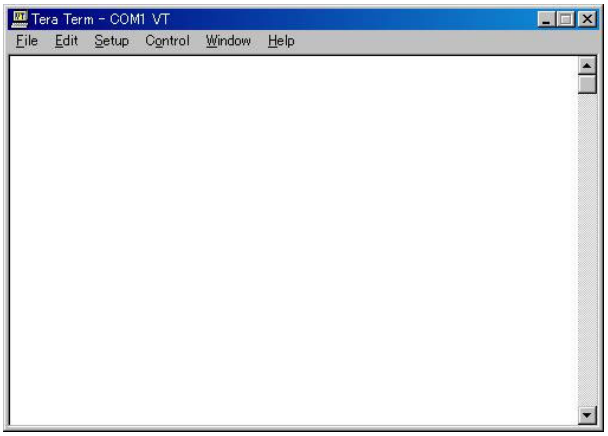
1. 「スタート」→「すべてのプログラム、またはプログラム」→「Tera Term Pro」→「Tera Term Pro」
   で Tera Term Pro が立ち上がります。



2. 最初にどこと接続するか確認する画面が出てきます。



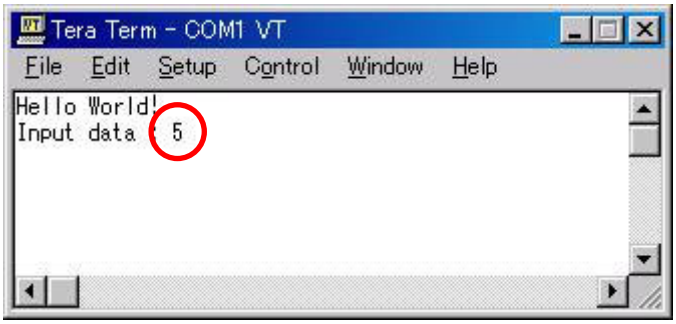
3. 「Serial」を選んで、ポート番号を選びます。選択後、OKをクリックして次へ進みます。



4. 立ち上がりました。

もし、メッセージは表示されるのに、キーを入力しても何も表示されない場合、RS-232C コネクタの 7-8 ピンがショートされていることを確認してください。

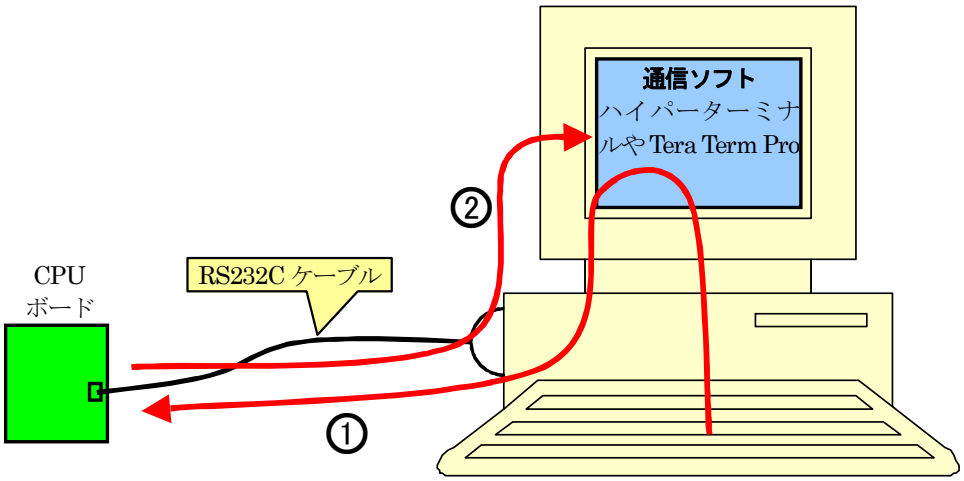
5.H8 の電源を入れると上記のように画面が表示されます。表示されなければ通信の設定が間違っているか接続がうまくいっていません。
この状態で、LED へ出力したいデータを 0～255 の範囲で入力、エンタキーを押すと、入力した値が表示されて LED へデータが出力されます。



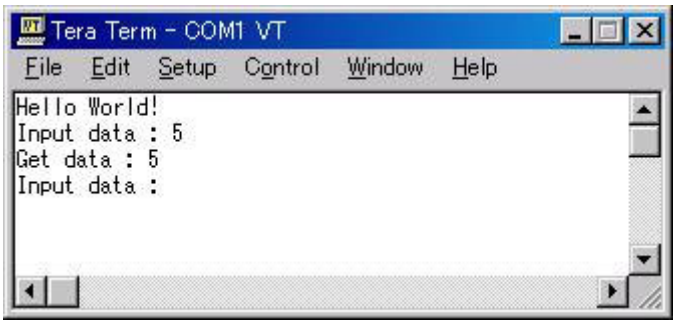
6.キーボードの「5」キーを入力します。TeraTermPro の画面に「5」が表示されました。これは、
キーボード→通信ソフトの画面
という流れで表示されたわけではありません。
「5」というデータは、次の順で CPU ボードへ送られます。

①キーボード→通信ソフト→RS232C ケーブル→CPU ボード
CPU ボードは送られてきたデータをそのままパソコンへ送ります(scanf 関数で実行しています)。

②CPU ボード→RS232C ケーブル→通信ソフトの画面
そのため、通信ソフトの画面に表示されるのです。



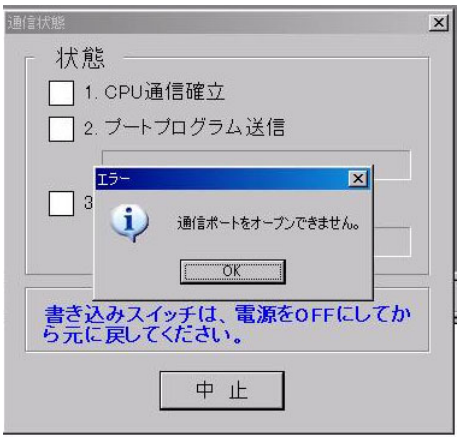
試しに、CPU ボードの電源を切ってみてください。キーボードに何を入力しても、通信ソフトの画面上には何も表示されません。



7.「5」の次にエンターを押すと、LED は、「0000 0101」出力となります。H8 側からも「Get data : 5」という文字列が返ってきて、5 が入力されたことが分かります。いろいろな値を入力してどうなるか試してみてください。

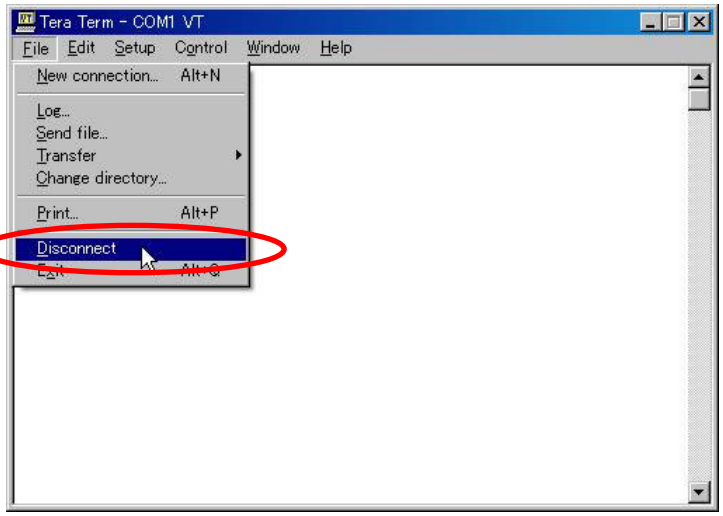
8.数値データ以外を入力すると、「Data Error!」と表示してエラーであることを知らせます。

24.10.4 TeraTermProを立ち上げた状態でプログラムの書き込みをするには

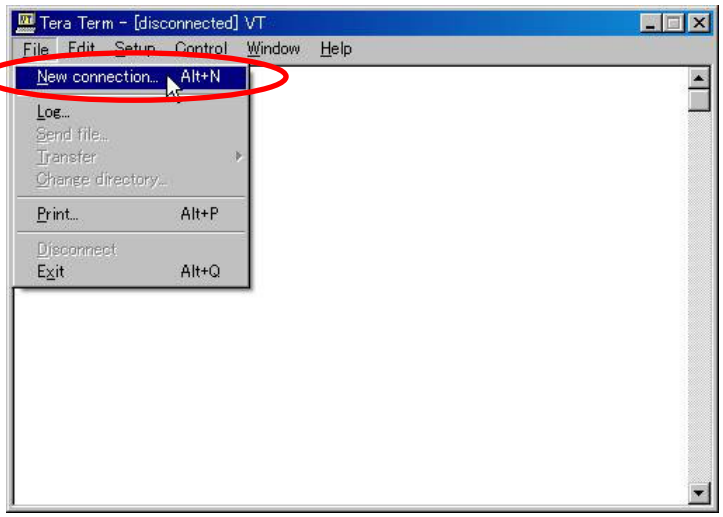


TeraTermPro を立ち上げた状態でプログラムを書き込もうとすると、エラーが出ます。これは、TeraTermPro が通信ポートを使用しているため、書き込みソフトが通信ポートを使えないためです。

TeraTermPro を終了すればよいですが、一回一回終了して、立ち上げるのは大変です。そこで次の手順で、TeraTermPro の接続をいったん切ります。そうすれば通信ポートが空くので、書き込みできます。

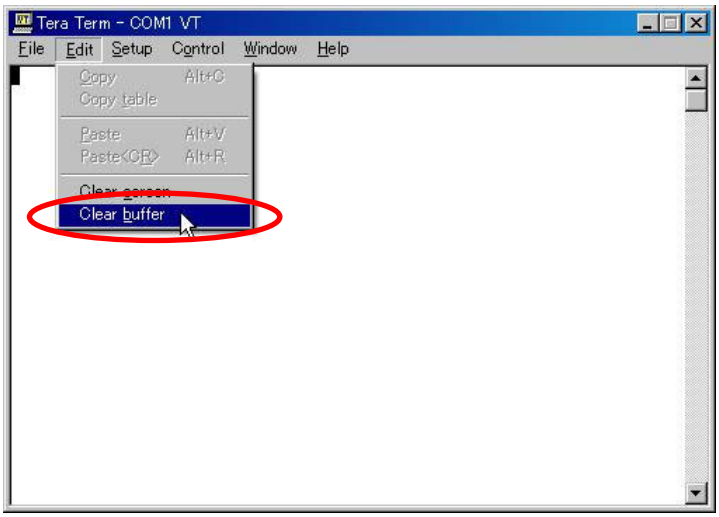


「File→Disconnect」をクリックします。これで TeraTermPro は、通信ポートを空けます。



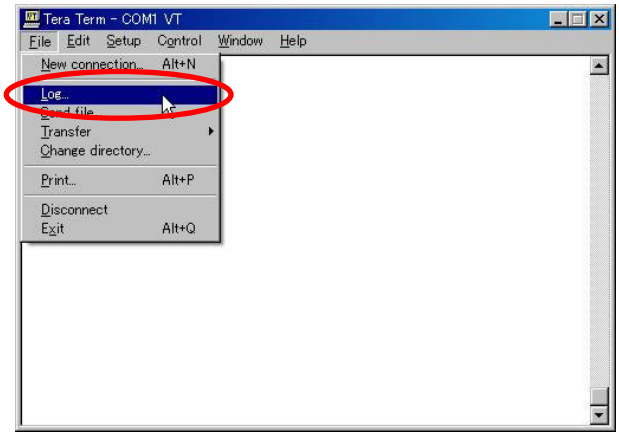
「File→New connection」で再度、接続します。

24.10.5 TeraTermProの画面をクリア

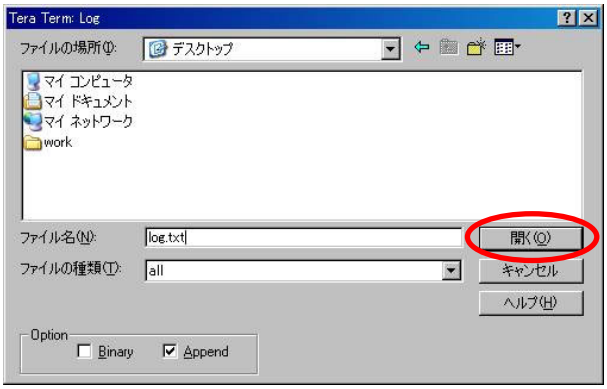


画面をクリアしたい場合は、「Edit→Clear buffer」です。

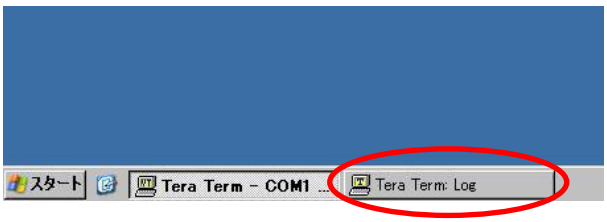
24.10.6 TeraTermProの通信内容を保存



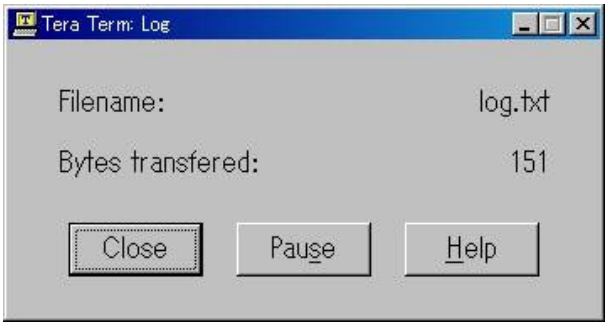
やり取りしているデータをファイルに保存することができます。「File→Log」を選択します。



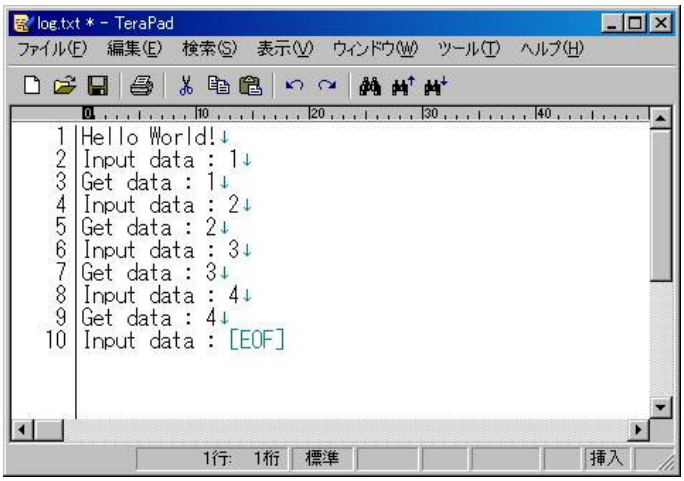
保存ファイル名を入力します。ここでは「log.txt」と入力します。開くをクリックして完了です。



「Tera Term: Log」と出てきます。



選択すると、現在の保存状況が表示されます。保存を終えるには、Close をクリックします。



「log.txt」をテキストエディタで開くと、やり取りした内容が保存されています。

25. プロジェクト「adsio」 電圧をパソコンへ出力

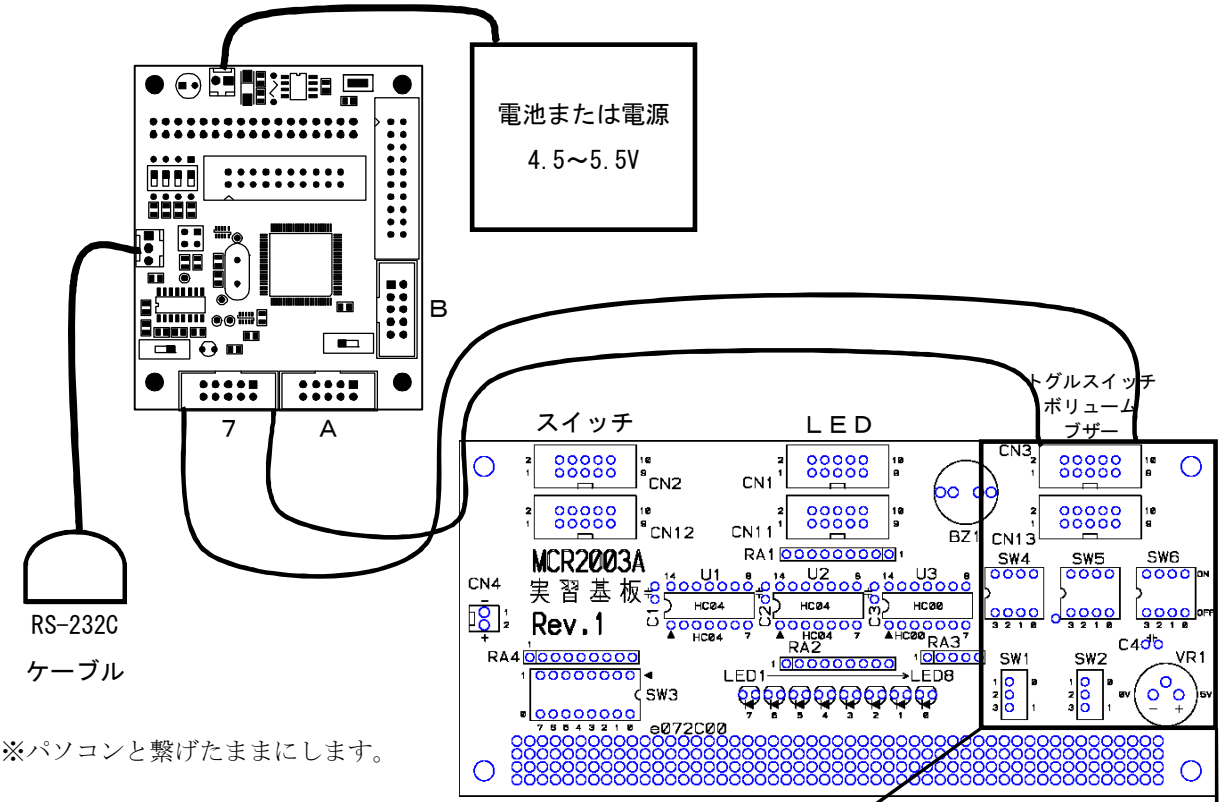
25.1 概要

H8 の A/D 変換器を使って、アナログ電圧をデジタル値に変換します。その値をパソコンへ電圧値として出力します。マイコンのポートは、下記を使用します。

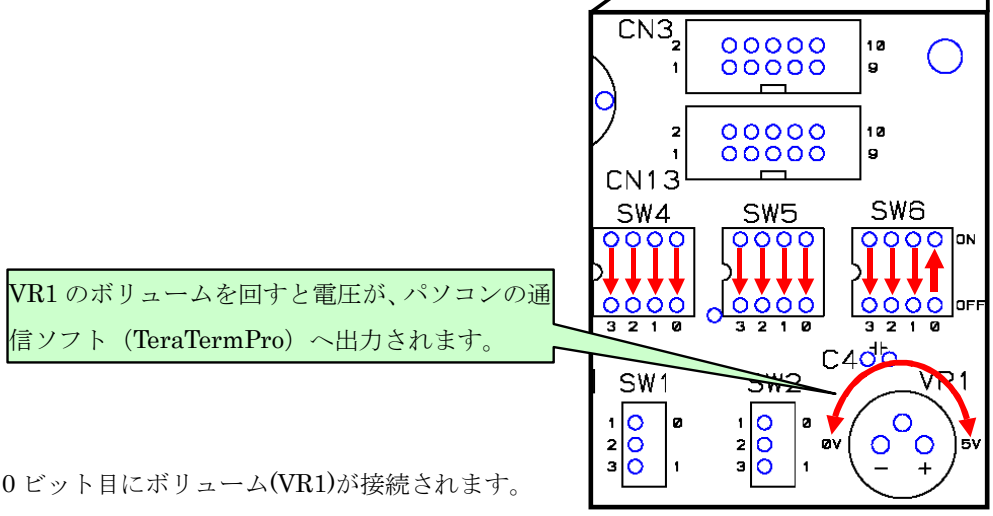
・ポート 7 の 0 ビット・・・アナログ電圧入力
・RS-232C ケーブル・・・パソコンと通信

25.2 接続

・CPU ボードのポート 7 と、実習基板のトグルスイッチ・ボリューム部をフラットケーブルで接続します。
・通信ケーブルは、パソコンの RS-232C コネクタに接続したままにしておきます。
・実習基板の SW6 No0 のスイッチを ON、SW4～6 のその他のビットを OFF にします。

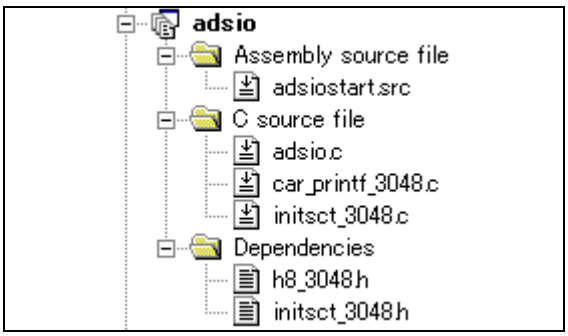


※パソコンと繋げたままにします。



※ポート 7 の 0 ビット目にボリューム(VR1)が接続されます。

25.3 プロジェクトの構成



	ファイル名	内容
1	adsiostart.src	アセンブリ言語で記述されたアセンブリソースファイルです。このファイルの構造は下記のようになっています。 ベクタアドレス ＋ スタートアップルーチン
2	initsct_3048.c	初期値のないグローバル変数(セクション B 領域)、初期値のあるグローバル変数(セクション R 領域)の初期化用です。
3	adsio.c	実際に制御するプログラムが書かれています。H8/3048F-ONE の内蔵周辺機能の初期化も行います。
4	h8_3048.h	H8/3048F-ONE の内蔵周辺機能の I/O レジスタを定義したファイルです。
5	initsct_3048.h	initsct_3048.c のヘッダファイルです。
6	car_printf_3048.c	・通信するための設定 ・printf 関数の出力先、scanf 関数の入力元を通信にするための設定 を行っています。

25.4 プログラム「adsio.c」

```
 1 :  /************************************************************************/
 2 :  /* 電圧を入力してパソコンへ出力「adsio.c」                              */
 3 :  /* 入力：実習基板CN3, P70(アナログ電圧)  出力：伝送ケーブル             */
 4 :  /* 通信：ﾃﾗﾀｰﾑﾌﾟﾛ/ﾊｲﾊﾟｰﾀｰﾐﾅﾙ  2005.04 ジャパンマイコンカーラリー実行委員会  */
 5 :  /************************************************************************/
 6 :  /*======================================*/
 7 :  /* インクルード                         */
 8 :  /*======================================*/
 9 :  #include    <no_float.h>              /* stdioの簡略化 最初に置く*/
10 :  #include    <stdio.h>
11 :  #include    <machine.h>
12 :  #include    "h8_3048.h"
13 :
14 :  /*======================================*/
15 :  /* シンボル定義                         */
16 :  /*======================================*/
17 :
18 :  /*======================================*/
19 :  /* プロトタイプ宣言                     */
20 :  /*======================================*/
21 :  void init( void );
22 :  int get_ad( void );
23 :
24 :  /*======================================*/
25 :  /* グローバル変数の宣言                 */
26 :  /*======================================*/
27 :  unsigned long   cnt1;                   /* main内で使用             */
28 :
29 :  /************************************************************************/
30 :  /* メインプログラム                                                     */
```

```
/************************************************************************/
void main( void )
{
    int     i;

    /* マイコン機能の初期化 */
    init();                             /* 初期化                   */
    init_sci1( 0x00, 79 );              /* SCI1初期化               */
    set_ccr( 0x00 );                    /* 全体割り込み許可         */

    printf( "Voltage Meter¥n");
    while( 1 ) {
        if( cnt1 >= 500 ) {
            cnt1 = 0;
            i = get_ad();
            i = (long)500 * i / 1023;
            printf( "%01d", i/100 );
            printf( "." );
            printf( "%02d¥n", i%100 );
        }
    }
}

/************************************************************************/
/* H8/3048F-ONE 内蔵周辺機能の初期化                                    */
/************************************************************************/
void init( void )
{
    /* ポートの入出力設定 */
    P1DDR = 0xff;
    P2DDR = 0xff;
    P3DDR = 0xff;
    P4DDR = 0xff;
    P5DDR = 0xff;
    P6DDR = 0xf0;                       /* CPU基板上のDIP SW        */
    P8DDR = 0xff;
    P9DDR = 0xf7;
    PADDR = 0xff;
    PBDDR = 0xff;                       /* LED基板                  */
    /* ポート７は、入力専用なので入出力設定はありません    */

    /* A/Dの初期設定 */
    AD_CSR = 0x00;

    /* ITU0 1ms毎の割り込み */
    ITU0_TCR = 0x20;
    ITU0_GRA = 24575;
    ITU0_IER = 0x01;
    ITU_STR  = 0x01;
}

/************************************************************************/
/* A/D値読み込み(AN0)                                                   */
/* 引数    なし                                                         */
/* 戻り値 A/D値 0～1023                                                 */
/************************************************************************/
int get_ad( void )
{
    int i;

    AD_CSR |= 0x20;                     /* ADスタート               */
    while( !(AD_CSR & 0x80) );          /* エンドフラグをチェック   */
    AD_CSR &= 0x7f;                     /* エンドフラグクリア       */
    i = AD_DRA >> 6;                    /* 代入                     */

    return i;
}

/************************************************************************/
/* ITU0 割り込み処理                                                    */
/************************************************************************/
#pragma interrupt( interrupt_timer0 )
void interrupt_timer0( void )
{
    ITU0_TSR &= 0xfe;                   /* フラグクリア             */
    cnt1++;
}

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

25.5 プログラムの解説

25.5.1 A/D変換の初期設定

```
73 :       AD_CSR = 0x00;
```

AN0(P70)端子をアナログ入力端子にします。

25.5.2 割り込みの設定

```
76 :       ITU0_TCR = 0x20;
77 :       ITU0_GRA = 24575;
78 :       ITU0_IER = 0x01;
79 :       ITU_STR  = 0x01;
```

ITU0 を使って、1[ms]ごとに割り込みが発生するように設定します。

25.5.3 get_ad関数

```
87 :  int get_ad( void )
88 :  {
89 :      int i;
90 :
91 :      AD_CSR |= 0x20;                    /* AD スタート                */
92 :      while( !(AD_CSR & 0x80) );         /* エンドフラグをチェック    */
93 :      AD_CSR &= 0x7f;                    /* エンドフラグクリア        */
94 :      i = AD_DRA >> 6;                   /* 代入                       */
95 :
96 :      return i;
97 :  }
```

ad.c と同じですが、94 行のシフトを 6 ビットのみにして、戻り値は 0～1023 にしています。

25.5.4 interrupt_timer0 関数(割り込みプログラム)

```
102 : #pragma interrupt( interrupt_timer0 )
103 : void interrupt_timer0( void )
104 : {
105 :     ITU0_TSR &= 0xfe;                  /* フラグクリア              */
106 :     cnt1++;
107 : }
```

この関数が、1[ms]ごとに実行されます。cnt1 変数を＋1します。メインプログラム内で、この変数を使用して時間測定用に使います。

25.5.5 main関数

```
 32 : void main( void )
 33 : {
 34 :     int     i;
 35 :
 36 :     /* マイコン機能の初期化 */
 37 :     init();                              /* 初期化                    */
 38 :     init_sci1( 0x00, 79 );               /* SCI1 初期化               */
 39 :     set_ccr( 0x00 );                     /* 全体割り込み許可          */
 40 :
 41 :     printf( "Voltage Meter¥n");
 42 :     while( 1 ) {
 43 :         if( cnt1 >= 500 ) {
 44 :             cnt1 = 0;
 45 :             i = get_ad();
 46 :             i = (long)500 * i / 1023;
 47 :             printf( "%01d", i/100 );
 48 :             printf( "." );
 49 :             printf( "%02d¥n", i%100 );
 50 :         }
 51 :     }
 52 : }
```

37 行で、内蔵周辺機能の初期化、38 行は通信の設定です。割り込みを使いますので、39 行で全体の割り込みを許可しています。

43 行で、500 ミリ秒たったかチェックし、経ったなら 44 行以降を実行します。44 行で cnt1 をクリアしているので、500 ミリ秒に1回実行されることになります。

45 行で、A/D 値を取得します。0～1023 です。

46 行で、0～1023 の値を 0～500 に変換します。i は最大 1023 です。500×1023 は、int 型の上限 32767 を超えるので long 型に型変換しておきます。

47 行で i を 100 で割った値をパソコンへ出力します。int 型なので下2桁は切り捨てられます。

48 行で「.」(ピリオド)をパソコンへ出力します。

49 行で i を 100 で割った値の余りをパソコンへ出力します。

例えば、get_ad 関数の戻り値が 678 とします。

$$i = 500 \times \frac{A/D値}{1023}$$
$$= 500 \times 678 \div 1023$$
$$= 331$$

となります。

次に i を 100 で割ります。

331/100＝3

3がパソコンへ出力されます。

次にピリオド「.」がパソコンへ出力されます。

次に i を 100 で割った余りが出力されます。

331/100＝3　余り 31

よって、31 がパソコンへ出力されます。

結果、パソコンへは、

```
３．３１
```

が出力されます。A/D 値 678 は 3.31V です。A/D 値を少数第 2 桁の電圧に変換してパソコンに出力しています。

ちなみに、0～5V が 0～1023 の値なので1当たり

$$5／1023≒0.00489[V]＝4.89[mV]$$

となります。

```
43 :          if( cnt1 >= 500 ) {
```

43 行の 500 という数字が、出力する時間の間隔です。最少間隔は通信速度を考えると 50ms くらいでしょうか。最高は cnt1 が unsigned long 型なので約 42 億ミリ秒($2^{32}-1$)まで大丈夫です。ちなみに 49.7 日です。

26. プロジェクト「adsio2」 電圧をパソコンへ出力

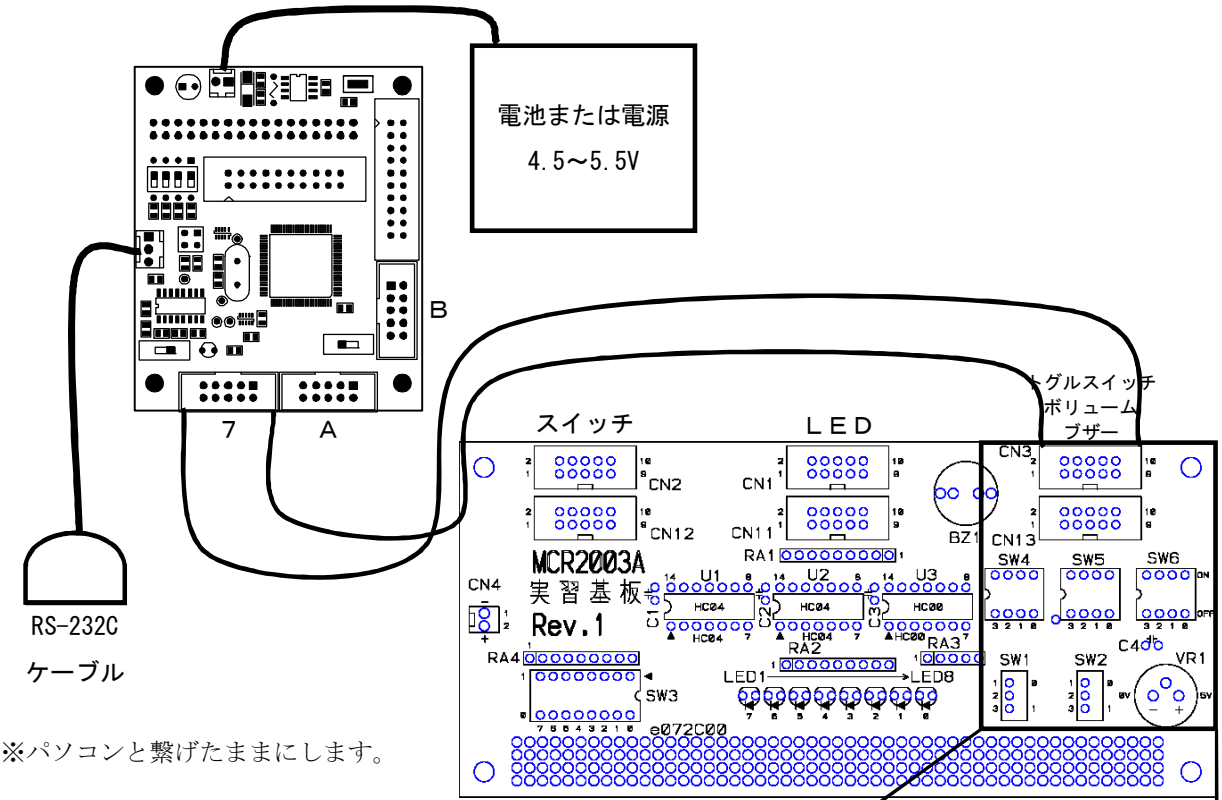
26.1 概要

H8 の A/D 変換器を使って、アナログ電圧をデジタル値に変換します。変換データ 100 個分を、いったん H8 のメモリに保存します。保存後、一括でパソコンへ電圧値として出力します。マイコンのポートは、下記を使用します。

・ポート 7 の 0 ビット・・・アナログ電圧入力
・RS-232C ケーブル・・・パソコンと通信

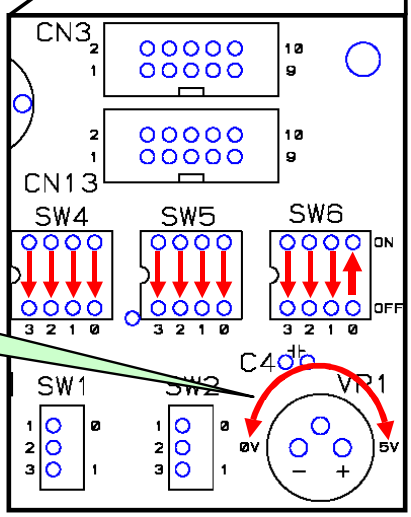
26.2 接続

・CPU ボードのポート 7 と、実習基板のトグルスイッチ・ボリューム部をフラットケーブルで接続します。
・通信ケーブルは、パソコンの RS-232C コネクタに接続したままにしておきます。
・実習基板の SW6 No0 のスイッチを ON、SW4～6 のその他のビットを OFF にします。



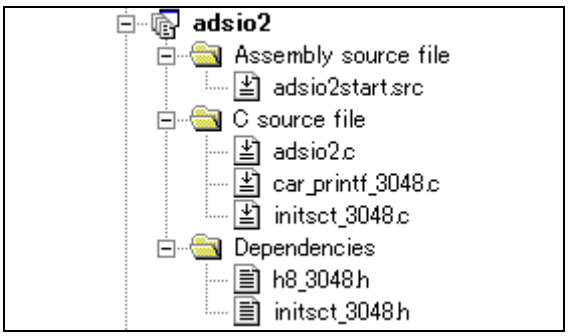
CPU ボードのディップスイッチをどれでも良いので変化させると A/D 変換をスタートします。50ms ごとに 100 回分のボリューム電圧を保存します。保存後、パソコンの通信ソフト（TeraTermPro など）へ A/D 値を一気に出力します。

VR1 のボリュームを回すと電圧が、パソコンの通信ソフト（TeraTermPro）へ出力されます。



※ポート 7 の 0 ビット目にボリューム(VR1)が接続されます。

26.3 プロジェクトの構成



	ファイル名	内容
1	adsio2start.src	アセンブリ言語で記述されたアセンブリソースファイルです。このファイルの構造は下記のようになっています。 ベクタアドレス ＋ スタートアップルーチン
2	initsct_3048.c	初期値のないグローバル変数(セクション B 領域)、初期値のあるグローバル変数(セクション R 領域)の初期化用です。
3	adsio2.c	実際に制御するプログラムが書かれています。H8/3048F-ONE の内蔵周辺機能の初期化も行います。
4	h8_3048.h	H8/3048F-ONE の内蔵周辺機能の I/O レジスタを定義したファイルです。
5	initsct_3048.h	initsct_3048.c のヘッダファイルです。
6	car_printf_3048.c	・通信するための設定 ・printf 関数の出力先、scanf 関数の入力元を通信にするための設定 を行っています。

26.4 プログラム「adsio2.c」

```
 1 :  /*****************************************************************************/
 2 :  /* 電圧を１００回分入力して一気にパソコンへ出力「adsio2.c」                  */
 3 :  /* 入力：実習基板CN3  P70(アナログ電圧)      出力：CN1  伝送ケーブル          */
 4 :  /* 通信：ﾃﾗﾀｰﾑﾌﾟﾛ/ﾊｲﾊﾟｰﾀｰﾐﾅﾙ  2005.04 ジャパンマイコンカーラリー実行委員会  */
 5 :  /*****************************************************************************/
 6 :  /*====================================*/
 7 :  /* インクルード                       */
 8 :  /*====================================*/
 9 :  #include    <no_float.h>               /* stdioの簡略化 最初に置く*/
10 :  #include    <stdio.h>
11 :  #include    <machine.h>
12 :  #include    "h8_3048.h"
13 :
14 :  /*====================================*/
15 :  /* シンボル定義                       */
16 :  /*====================================*/
17 :  #define     DATA_MAX    100            /* データ数                 */
18 :
19 :  /*====================================*/
20 :  /* プロトタイプ宣言                   */
21 :  /*====================================*/
22 :  void init( void );
23 :  int get_ad( void );
24 :
25 :  /*====================================*/
26 :  /* グローバル変数の宣言               */
27 :  /*====================================*/
28 :  unsigned long   cnt1;                  /* main内で使用             */
29 :  int             buff[ DATA_MAX ];      /* データ保存用バッファ     */
30 :
```

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

    /* マイコン機能の初期化 */
    init();                             /* 初期化                   */
    init_sci1( 0x00, 79 );              /* SCI1初期化               */
    set_ccr( 0x00 );                    /* 全体割り込み許可         */

start:
    /* 開始待ち */
    PADR = 0x01;
    i = P6DR;
    while( P6DR == i );

    /* データ取得 */
    cnt1 = 0;
    for( i=0; i<DATA_MAX; i++ ) {
        PADR = i;
        while( cnt1 < 50 );
        buff[ i ] = get_ad();
        cnt1 = 0;
    }

    /* データ出力 */
    PADR = 0x80;
    printf( "¥n");
    printf( "Data Start!¥n");
    for( i=0; i<DATA_MAX; i++ ) {
        printf( "%d¥n", buff[ i ] );
    }

    goto start;
}

/************************************************************************/
/* H8/3048F-ONE 内蔵周辺機能の初期化                                    */
/************************************************************************/
void init( void )
{
    /* ポートの入出力設定 */
    P1DDR = 0xff;
    P2DDR = 0xff;
    P3DDR = 0xff;
    P4DDR = 0xff;
    P5DDR = 0xff;
    P6DDR = 0xf0;                       /* CPU基板上のDIP SW        */
    P8DDR = 0xff;
    P9DDR = 0xf7;
    PADDR = 0xff;                       /* LED基板                  */
    PBDDR = 0xff;
    /* ポート７は、入力専用なので入出力設定はありません    */

    /* A/Dの初期設定 */
    AD_CSR = 0x00;

    /* ITU0 1ms毎の割り込み */
    ITU0_TCR = 0x20;
    ITU0_GRA = 24575;
    ITU0_IER = 0x01;
    ITU_STR  = 0x01;
}

/************************************************************************/
/* A/D値読み込み(AN0)                                                   */
/* 引数    なし                                                         */
/* 戻り値 A/D値 0～1023                                                 */
/************************************************************************/
int get_ad( void )
{
    int i;

    AD_CSR |= 0x20;                     /* ADスタート               */
    while( !(AD_CSR & 0x80) );          /* エンドフラグをチェック   */
    AD_CSR &= 0x7f;                     /* エンドフラグクリア       */
    i = AD_DRA >> 6;                    /* 代入                     */

    return i;
}

/************************************************************************/
/* ITU0 割り込み処理                                                    */
/************************************************************************/
#pragma interrupt( interrupt_timer0 )
void interrupt_timer0( void )
{
    ITU0_TSR &= 0xfe;                   /* フラグクリア             */
    cnt1++;
```

```
122 :  }
123 :
124 :  /************************************************************************/
125 :  /* end of file                                                          */
126 :  /************************************************************************/
```

26.5 プログラムの解説

adsio.c では、A/D 値をすぐにデータをパソコンへ送信しました。今回はデータを一時的に保存して、その後一気にパソコンへ送信します。

パソコンへの送信は若干時間がかかるので、短い間隔でデータ取得するときはこの方法を使います。また、通信ケーブルを外した状態でデータを取得して、取得後にパソコンと接続してデータを送信する場合もこの方法を用います。実は、後者はまさにマイコンカーです。マイコンカーを走らせ走行データを保存、走行後にパソコンと繋いで走行データを取得などの応用があります。

26.5.1 保存するデータ数

```
17 :  #define     DATA_MAX    100             /* データ数                 */
```

保存するデータ数を記述します。DATA_MAX という文字でデータ数を置き換えます。DATA_MAX＝100です。なぜ、プログラム中に直接100と書かないかというと、プログラムを見渡すと define 文以外で DATA_MAX が3個あります。もし、データ数を変えたい場合、3箇所変えなければいけません。DATA_MAX というように定数にすると、define 文で宣言している1行だけ変えればすべて書き換わり、簡単にデータ数を変えることができます。今回は、3箇所だけでしたが、プログラムが大きくなり10箇所や100箇所！にもなるととても書き換えられません。

変更しやすい部分を define 宣言しておくと、後で改造しやすくなります。

26.5.2 グローバル変数

```
28 :  unsigned long   cnt1;                   /* main 内で使用            */
29 :  int             buff[ DATA_MAX ];       /* データ保存用バッファ     */
```

cnt1 はタイマ用です。buff は配列で、DATA_MAX 個分の値を保存します。

buff は int 型なので、メモリは 2 バイト使用します。H8/3048F-ONE の RAM 領域は 4KB なので、2000 個確保すると、メモリを使い果たしてしまいます。ただし、関数の戻り先アドレスの保存、割り込み時の値保存など、マイコン側でも使用します。そのため、最大で約 2.5～3.0KB の確保が限界です(プログラムの規模が大きければさらに小さくなります)。int 型だと、1500 個くらいが限界です。ちなみに char 型は 1 バイトなので、そのまま 3000 個程度、long 型は 4 バイトなので 750 個程度確保できます。メモリを確保しすぎると、プログラムの動作がおかしくなります。おかしい動作をした場合は、確保する量を小さくしてみてください。

26.5.3 main関数

```
34 :  void main( void )
35 :  {
36 :      int     i;
37 :
38 :      /* マイコン機能の初期化 */
39 :      init();                             /* 初期化                   */
40 :      init_sci1( 0x00, 79 );              /* SCI1 初期化              */
41 :      set_ccr( 0x00 );                    /* 全体割り込み許可         */
```

マイコンの内蔵周辺機能の初期化、通信の初期化、全体割り込みの許可をします。

26.5.4 開始待ち

```
43 :  start:
44 :      /* 開始待ち */
45 :      PADR = 0x01;
46 :      i = P6DR;
47 :      while( P6DR == i );
```

A/D 値の取得をスタートするまでの待ち状態です。
46 行目で、ポート 6 のデータを読み込みます。
47 行目で、そのデータをチェック、変化が有れば次へ進みます。ポート 6 は CPU ボード上のディップスイッチに繋がっていますので、ディップスイッチの値が変わるとスタートです。
ちなみに、ポート 6 は P66～P60 まであり、ディップスイッチは P63～P60 です。P66～P64 は何も繋がっていません。出力用に設定にしているので、現在出力している値が読み込まれます。このプログラムでは、P66～P64 を何も変更していませんので変化がありません。もし開始待ちのときに、P66～P64 の出力値を変えたり、入力設定にして入力値が変わったりする場合は、ディップスイッチの値を変えていないのにポート 6 の値が変わってしまいます。当然、次に進んでしまいます。その場合は P66～P64 をマスクして、ディップスイッチのあるビットだけを読み込みます。

26.5.5 データ取得

```
49 :      /* データ取得 */
50 :      cnt1 = 0;
51 :      for( i=0; i<DATA_MAX; i++ ) {
52 :          PADR = i;
53 :          while( cnt1 < 50 );
54 :          buff[ i ] = get_ad();
55 :          cnt1 = 0;
56 :      }
```

DATA_MAX の数値分、データを取得します。buff 配列へ1個ずつ保存します。保存する間隔は、53 行の 50 という数字です。今回は 50ms ごとに保存しています。同時に、ポート A と接続した LED の表示が増えていき、保存されていることを示します。無くとも動作自体には全く影響しませんが、現在データ取得中ということを外部に知らせます。

26.5.6 データ出力

```
58 :      /* データ出力 */
59 :      PADR = 0x80;
60 :      printf( "¥n");
61 :      printf( "Data Start!¥n");
62 :      for( i=0; i<DATA_MAX; i++ ) {
63 :          printf( "%d¥n", buff[ i ] );
64 :      }
```

保存したデータを、一気にパソコンへ送信します。adsio2.c では電圧に変換せずに A/D 値をそのまま出力します。

26.5.7 終了

```
66 :      goto start;
```

送信後、start へ戻ります。戻り先は開始待ち部分です。次にディップスイッチの値が変わるまで待ち続けます。

C言語は構造化プログラムです。構造化とは大きな機能を細分化し、小さな機能の集合にすることです。具体的には、プログラムを小さいサブルーチンにし、処理の流れが1目で分かるように表現します。

本マニュアルは、C言語についての内容を説明するマニュアルではありませんので詳しくは説明しませんが、構造化プログラムで goto 文は歓迎されません。それは、処理の流れが goto 文により大きく変わるためです。

しかし、多くの入れ子を抜ける場合やエラー処理で別な場所へ実行を移したい場合など、goto を使うことによって分かりやすくなることもあります。ただし、多くの入れ子にすること自体、プログラムの作りが悪いという人もいて難しいところです。

今回は、goto 文も使えるという例としてプログラムを作りました。goto 文を使わなければ下記のようになります。

```
/* goto を使用しないプログラム */
void main( void )
{
    while( 1 ) {
        プログラム
    }
}
```

```
/* goto を使用したプログラム */
void main( void )
{
start:
    プログラム
    goto start;
}
```

27. プロジェクト「adpwm」 A/D値に応じてPWMのデューティ比を変える

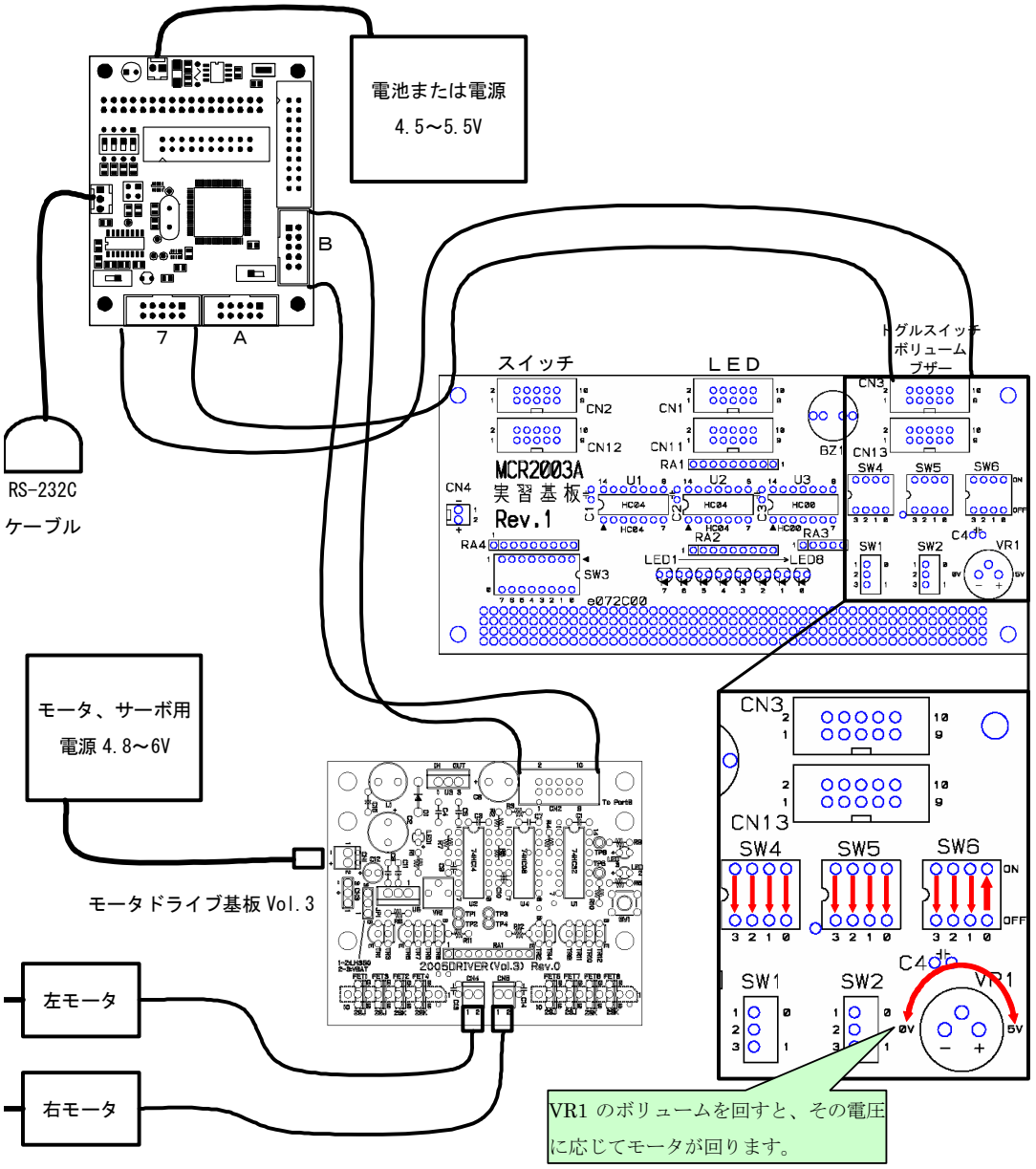
27.1 概要

実習基板のボリュームの電圧により、モータドライブ基板 Vol.3 の左モータの回転を制御します。
・ポート 7 の 0 ビット・・・アナログ電圧(0～5V)入力
・ポート B の 1 ビット・・・PWM 出力(モータドライブ基板 Vol.3 の左モータ)

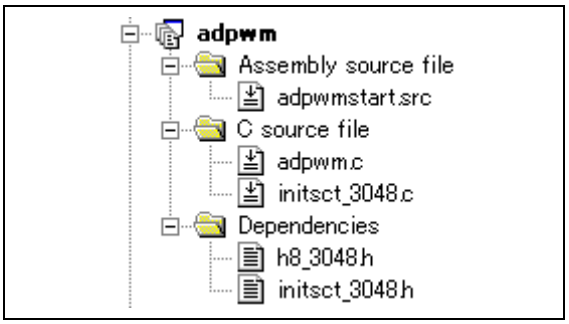
27.2 接続

・CPU ボードのポート 7 と、実習基板のトグルスイッチ・ボリューム部をフラットケーブルで接続します。
・CPU ボードのポート B と、モータドライブ基板 Vol.3 をフラットケーブルで接続します。
・実習基板の SW6 No0 のスイッチを ON、SW4～6 のその他のビットを OFF にします。

※モータドライブ基板が無い場合、ポート B を LED 部に接続します。ボリュームを回すことにより、LED の明るさが変わります。



27.3 プロジェクトの構成



	ファイル名	内容
1	adpwmstart.src	アセンブリ言語で記述されたアセンブリソースファイルです。このファイルの構造は下記のようになっています。 ベクタアドレス ＋ スタートアップルーチン
2	initsct_3048.c	初期値のないグローバル変数(セクション B 領域)、初期値のあるグローバル変数(セクション R 領域)の初期化用です。
3	adpwm.c	実際に制御するプログラムが書かれています。H8/3048F-ONE の内蔵周辺機能の初期化も行います。
4	h8_3048.h	H8/3048F-ONE の内蔵周辺機能の I/O レジスタを定義したファイルです。
5	initsct_3048.h	initsct_3048.c のヘッダファイルです。

27.4 プログラム「adpwm.c」

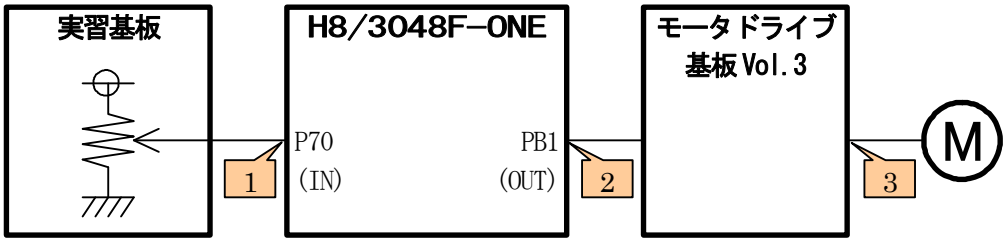
```
 1 :  /************************************************************************/
 2 :  /* 電圧に応じて、PWMのデューティ比を変える "adpwm.c"                    */
 3 :  /*                          2008.06 ジャパンマイコンカーラリー実行委員会 */
 4 :  /************************************************************************/
 5 :  /*
 6 :  入力：P70(0～5V)
 7 :  出力：PB7-PB0(モータドライブ基板、無い場合はLED)
 8 :
 9 :  ポート7のbit0に入力した電圧(0～5V)に応じて、PWMのデューティ比を変えます。
10 :  リセット同期PWMモードのバッファ動作を使用しています。
11 :  */
12 :
13 :  /*====================================*/
14 :  /* インクルード                       */
15 :  /*====================================*/
16 :  #include    <machine.h>
17 :  #include    "h8_3048.h"
18 :
19 :  /*====================================*/
20 :  /* シンボル定義                       */
21 :  /*====================================*/
22 :
23 :  /*====================================*/
24 :  /* プロトタイプ宣言                   */
25 :  /*====================================*/
26 :  void init( void );
27 :  int get_ad( void );
28 :
29 :  /************************************************************************/
30 :  /* メインプログラム                                                     */
31 :  /************************************************************************/
32 :  void main( void )
33 :  {
34 :      int ad;
35 :
36 :      init();                             /* マイコン機能の初期化         */
37 :
38 :      while( 1 ) {
```

```
39 :          ad = get_ad();
40 :          ITU3_BRB = (long)49150 * ad / 1023; /* PB1(左モータ)のPWM設定   */
41 :      }
42 :  }
43 :
44 :  /************************************************************************/
45 :  /* H8/3048F-ONE 内蔵周辺機能の初期化                                     */
46 :  /************************************************************************/
47 :  void init( void )
48 :  {
49 :      /* ポートの入出力設定 */
50 :      P1DDR = 0xff;
51 :      P2DDR = 0xff;
52 :      P3DDR = 0xff;
53 :      P4DDR = 0xff;
54 :      P5DDR = 0xff;
55 :      P6DDR = 0xf0;                       /* CPU基板上のDIP SW        */
56 :      P8DDR = 0xff;
57 :      P9DDR = 0xf7;
58 :      PADDR = 0xff;
59 :      PBDR  = 0xc0;
60 :      PBDDR = 0xfe;                       /* モータドライブ基板Vol.3  */
61 :      /* ポート７は、入力専用なので入出力設定はありません    */
62 :
63 :      /* ITU3,4 リセット同期PWMモード 左右モータ、サーボ用 */
64 :      ITU3_TCR = 0x23;                    /* カウンタ、クリアの設定   */
65 :      ITU_FCR  = 0x3e;                    /* リセット同期PWMモード    */
66 :      ITU3_GRA = 49151;                   /* 周期の設定               */
67 :      ITU3_GRB = ITU3_BRB = 0;            /* 左モータのPWM設定        */
68 :      ITU4_GRA = ITU4_BRA = 0;            /* 右モータのPWM設定        */
69 :      ITU4_GRB = ITU4_BRB = 5000;         /* サーボのPWM設定          */
70 :      ITU_TOER = 0x38;                    /* 出力端子の設定           */
71 :      ITU_STR  = 0x08;                    /* タイマスタート           */
72 :
73 :      /* A/Dの初期設定 */
74 :      AD_CSR = 0x00;                      /* AN0使用                  */
75 :  }
76 :
77 :  /************************************************************************/
78 :  /* A/D値読み込み(AN0)                                                   */
79 :  /* 引数    なし                                                         */
80 :  /* 戻り値 A/D値 0～1023                                                 */
81 :  /************************************************************************/
82 :  int get_ad( void )
83 :  {
84 :      int i;
85 :
86 :      AD_CSR |= 0x20;                     /* ADスタート               */
87 :      while( !(AD_CSR & 0x80) );          /* エンドフラグをチェック   */
88 :      AD_CSR &= 0x7f;                     /* エンドフラグクリア       */
89 :      i = AD_DRA >> 6;                    /* 代入                     */
90 :
91 :      return i;
92 :  }
93 :
94 :  /************************************************************************/
95 :  /* end of file                                                          */
96 :  /************************************************************************/
```

27.5 プログラムの解説

基本的なプログラムは、プロジェクト「ad」とプロジェクト「motor」を組み合わせた内容です。詳しくはそれぞれのプロジェクトを参照してください。

信号の流れは、下図のようになります。



番号	内容
1	0～5V の信号をマイコンの P70 から入力します。A/D 変換器で 0～1023 の値に変換します。
2	A/D 変換値 0～1023 の値を PWM のデューティ比 0～100%に変換します。計算は、 ディーティ比 ＝PWM100%の時の ITU3_BRB の値×A/D 変換値÷1023 ＝49150×(0～1023)÷1023
3	モータドライブ基板で、マイコンからの PWM 信号をモータを回すことのできる大電流信号に変換します。

28. プロジェクト「adpwm2」 A/D値に応じてPWMのデューティ比を変える(正転、逆転あり)

28.1 概要

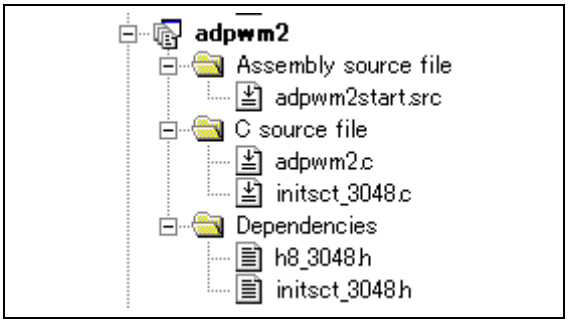
実習基板にあるボリュームの電圧をマイコンで読み込み、A/D 変換します。2.5V を 0%として、電圧を下げるとモータが正転、電圧を上げるとモータが逆転します。

・ポート 7 の 0 ビット・・・アナログ電圧(0～5V)入力
・ポート B の 1 ビット・・・PWM 出力(モータドライブ基板 Vol.3 の左モータ)
・ポート B の 2 ビット・・・回転方向出力(モータドライブ基板 Vol.3 の左モータ)

28.2 接続

「27.2 接続」と同様です。

28.3 プロジェクトの構成



	ファイル名	内容
1	adpwm2start.src	アセンブリ言語で記述されたアセンブリソースファイルです。このファイルの構造は下記のようになっています。 ベクタアドレス ＋ スタートアップルーチン
2	initsct_3048.c	初期値のないグローバル変数(セクション B 領域)、初期値のあるグローバル変数(セクション R 領域)の初期化用です。
3	adpwm2.c	実際に制御するプログラムが書かれています。H8/3048F-ONE の内蔵周辺機能の初期化も行います。
4	h8_3048.h	H8/3048F-ONE の内蔵周辺機能の I/O レジスタを定義したファイルです。
5	initsct_3048.h	initsct_3048.c のヘッダファイルです。

28.4 プログラム「adpwm2.c」

```
 1 :  /*********************************************************************/
 2 :  /* ジョイステック使用時のデューティ比を変える "adpwm2c"               */
 3 :  /*                          2008.06 ジャパンマイコンカーラリー実行委員会 */
 4 :  /*********************************************************************/
 5 :  /*
 6 :  入力：P70(0～5V)
 7 :  出力：PB7-PB0(モータドライブ基板、無い場合はLED)
 8 :
 9 :  ポート7のbit0に入力した電圧(0～5V)に応じて、PWMのデューティ比を変えます。
10 :  リセット同期PWMモードのバッファ動作を使用しています。
11 :
12 :  */
13 :
14 :  /*====================================*/
```

```
 15 :  /* インクルード                          */
 16 :  /*======================================*/
 17 :  #include    <machine.h>
 18 :  #include    "h8_3048.h"
 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 :  int get_ad( void );
 29 :  int data_cnv( int data );
 30 :
 31 :  /************************************************************************/
 32 :  /* メインプログラム                                                     */
 33 :  /************************************************************************/
 34 :  void main( void )
 35 :  {
 36 :      int ad, pwm;
 37 :
 38 :      init();                             /* マイコン機能の初期化     */
 39 :
 40 :      while( 1 ) {
 41 :          ad = get_ad();                  /* A/D値  取得              */
 42 :          pwm = data_cnv( ad );           /* A/D値→PWM値に変換       */
 43 :          if( pwm > 0 ) {
 44 :              /* 正転 */
 45 :              PBDR &= 0xfb;
 46 :          } else if( pwm < 0 ) {
 47 :              /* 逆転 */
 48 :              PBDR |= 0x04;
 49 :              pwm = -pwm;
 50 :          }
 51 :          ITU3_BRB = (long)49150 * pwm / 100; /* PB1(左モータ)のPWM設定 */
 52 :      }
 53 :  }
 54 :
 55 :  /************************************************************************/
 56 :  /* H8/3048F-ONE 内蔵周辺機能の初期化                                    */
 57 :  /************************************************************************/
 58 :  void init( void )
 59 :  {
 60 :      /* ポートの入出力設定 */
 61 :      P1DDR = 0xff;
 62 :      P2DDR = 0xff;
 63 :      P3DDR = 0xff;
 64 :      P4DDR = 0xff;
 65 :      P5DDR = 0xff;
 66 :      P6DDR = 0xf0;                       /* CPU基板上のDIP SW        */
 67 :      P8DDR = 0xff;
 68 :      P9DDR = 0xf7;
 69 :      PADDR = 0xff;
 70 :      PBDR  = 0xc0;
 71 :      PBDDR = 0xfe;                       /* モータドライブ基板Vol.3  */
 72 :      /* ポート7は、入力専用なので入出力設定はありません    */
 73 :
 74 :      /* ITU3,4 リセット同期PWMモード 左右モータ、サーボ用 */
 75 :      ITU3_TCR = 0x23;                    /* カウンタ、クリアの設定   */
 76 :      ITU_FCR  = 0x3e;                    /* リセット同期PWMモード    */
 77 :      ITU3_GRA = 49151;                   /* 周期の設定               */
 78 :      ITU3_GRB = ITU3_BRB = 0;            /* 左モータのPWM設定        */
 79 :      ITU4_GRA = ITU4_BRA = 0;            /* 右モータのPWM設定        */
 80 :      ITU4_GRB = ITU4_BRB = 5000;         /* サーボのPWM設定          */
 81 :      ITU_TOER = 0x38;                    /* 出力端子の設定           */
 82 :      ITU_STR  = 0x08;                    /* タイマスタート           */
 83 :
 84 :      /* A/Dの初期設定 */
 85 :      AD_CSR = 0x00;                      /* AN0使用                  */
 86 :  }
 87 :
 88 :  /************************************************************************/
 89 :  /* A/D値読み込み(AN0)                                                   */
 90 :  /* 引数    なし                                                         */
 91 :  /* 戻り値 A/D値 0～1023                                                 */
 92 :  /************************************************************************/
 93 :  int get_ad( void )
 94 :  {
 95 :      int i;
 96 :
 97 :      AD_CSR |= 0x20;                     /* ADスタート               */
 98 :      while( !(AD_CSR & 0x80) );          /* エンドフラグをチェック   */
 99 :      AD_CSR &= 0x7f;                     /* エンドフラグクリア       */
100 :      i = AD_DRA >> 6;                    /* 代入                     */
101 :
102 :      return i;
103 :  }
104 :
105 :  /************************************************************************/
```

```
106 :  /* A/D値→PWM値に変換 2.5V(中心)で0%、0Vで100%、5Vで-100%                */
107 :  /* 引数   AD値  0-1023                                                 */
108 :  /* 戻り値 -100～100                                                    */
109 :  /*********************************************************************/
110 :  int data_cnv( int data )
111 :  {
112 :      int ret;
113 :
114 :      if( data <= 400 ) {
115 :          /* 正転 0～100*/
116 :          ret = 400 - data;
117 :          ret /= 4;
118 :      } else if( data >= 623 ){
119 :          /* 逆転 -100 ～ 0 */
120 :          ret = 623 - data;
121 :          ret /= 4;
122 :      } else {
123 :          /* 停止 */
124 :          ret = 0;
125 :      }
126 :      return ret;
127 :  }
128 :
129 :  /*********************************************************************/
130 :  /* end of file                                                       */
131 :  /*********************************************************************/
```

28.5 プログラムの解説

28.5.1 data_cnv関数－A/D値の変換

```
105 :  /*********************************************************************/
106 :  /* A/D値→PWM値に変換 2.5V(中心)で0%、0Vで100%、5Vで-100%              */
107 :  /* 引数    AD値  0-1023                                               */
108 :  /* 戻り値 -100～100                                                   */
109 :  /*********************************************************************/
110 :  int data_cnv( int data )
111 :  {
112 :      int ret;
113 :
114 :      if( data <= 400 ) {
115 :          /* 正転 0～100*/
116 :          ret = 400 - data;
117 :          ret /= 4;
118 :      } else if( data >= 623 ){
119 :          /* 逆転 -100 ～ 0 */
120 :          ret = 623 - data;
121 :          ret /= 4;
122 :      } else {
123 :          /* 停止 */
124 :          ret = 0;
125 :      }
126 :      return ret;
127 :  }
```

下表のように、A/D 値を PWM 値に変換します。

A/D 値	PWM 値の計算	A/D 値が小さいときの PWM 値	A/D 値が大きいときの PWM 値
0～400	(400-A/D 値)/4	A/D 値が 0 のとき、(400-**0**)/4=100	A/D 値が 400 のとき、(400-**400**)/4=0
623～1023	(623-A/D 値)/4	A/D 値が 623 のとき、(623-**623**)/4=0	A/D 値が 1023 のとき、(623-**1023**)/4=-100
その他 (401～622)	常に 0	A/D 値が 401 のとき、0	A/D 値が 622 のとき、0

A/D 値が 401～622 のとき、すなわち電圧が 1.96～3.03V のときは PWM 値を 0%にしています。これは車のハンドルの遊びと同じで、0%のときの範囲を広くしてゆとりを持たせています。この範囲が無いと、PWM を 0%にするのは非常に大変です。試しに、A/D 値が 0～511 のとき、PWM 値を 100～0%に、512～1023 のとき、PWM 値を 0～-100%にするようプログラムを作って実験してみましょう。なぜその他(PWM0%)の部分をわざわざ作っているか、分かると思います。

28.5.2 main関数

```
31 :  /*********************************************************************/
32 :  /* メインプログラム                                                  */
33 :  /*********************************************************************/
34 :  void main( void )
35 :  {
36 :      int ad, pwm;
37 :
38 :      init();                             /* マイコン機能の初期化      */
39 :
40 :      while( 1 ) {
41 :          ad = get_ad();                  /* A/D値　取得               */
42 :          pwm = data_cnv( ad );           /* A/D値→PWM値に変換        */
43 :          if( pwm > 0 ) {
44 :              /* 正転 */
45 :              PBDR &= 0xfb;
46 :          } else if( pwm < 0 ) {
47 :              /* 逆転 */
48 :              PBDR |= 0x04;
49 :              pwm = -pwm;
50 :          }
51 :          ITU3_BRB = (long)49150 * pwm / 100; /* PB1(左モータ)のPWM設定 */
52 :      }
53 :  }
```

41 行目で、ad 変数に A/D 値(0～1023)の値を代入します。
42 行目で、A/D 値を-100～100%の PWM 値に変換して、pwm 変数に代入します。
43～50 行目で、-100～100 の PWM 値を下表のように変換します。

pwm 値	モータ動作
0 以上なら	PB2="0"(正転)にします。
0 以下なら	PB2="1"(逆転)にして、pwm 変数を正の数にします。

51 行で、PWM 値 0～100%を ITU3_BRB に設定する値に変換します。式は次のようになります。

$$\text{ITU3\_BRB} = 49150 \times \frac{\text{pwm変数}(0\sim100)}{100}$$

回転方向は PB2 で決めますので、ITU3_BRB は正転、逆転の符号に関係なく、0～49150 の値を代入します。

29. 付録

29.1 H8 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。

H8/300H のコンパイラは以下のようになっています。

・整数型(H8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 (signed char 型として扱われる)	-128 ～ 127	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(H8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)	7.0064923216240862e-46f (0x00000001)
double long double	8バイト	1.7976931348623158e+308 (0x7FEFFFFFFFFFFFFF)	4.9406564584124655e-324 (0x0000000000000001)

H8/300H 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。上記の値の範囲は、H8 マイコンのコンパイラのみ対応です。他の種類のマイコンは違う可能性が高いので必ず確認してください。

29.2 演算子

演算子を用いると、値をいろいろと加工することができます。演算というのはちょっと硬い表現ですが、簡単に言えば、足す、引く、掛ける、割るなどの計算です。

下表に、演算子の機能をまとめておきます。

演算子		機　能	備　　考	例
単項演算子	－	負		-a
	＋	正		+b
	～	ビットごとの反転	チルダと読む	~a
	－－	デクリメント		a--
	＋＋	インクリメント		b++
	＆	変数のアドレス	＆a はaの変数が格納されているアドレス	&a
	＊	ポインタ変数の指す内容	＊p はpの指す内容	*p
2項演算子	－	減算		a = b - c;
	＋	加算		a = b + c;
	＊	積		a = b * c;
	／	商		a = b / c;
	％	整数除算の余り		a = b % c;
	＆	ビットごとの論理積		a = b & 0x7f;
	｜	ビットごとの論理和		a = b | 0x80;
	＾	ビット毎の排他的論理和		a = b ^ 0x55;
	＆＆	論理積	答えは真か偽	if(a==b && c==d)
	｜｜	論理和	答えは真か偽	if(a==b || c==d)
	＞＞	右シフト	(変数名)＞＞(シフトするビット数)	a = a >> 2;
	＜＜	左シフト	(変数名)＜＜(シフトするビット数)	a = a << 2;
代入演算子	＝	代入		a = b;
	＋＝	加算して代入		a += b;
	－＝	減算して代入		a -= b;
	／＝	除算して代入		a /= b;
	％＝	剰余演算して代入		a %= b;
	＜＜＝	左シフト演算して代入		a <<= 5;
	＞＞＝	右シフト演算して代入		a >>= 2;
	＆＝	論理積して代入		a &= 0x55;
	｜＝	論理和して代入		a |= b;
	＾＝	排他的論理和して代入		a ^= b;
比較演算子	＝＝	等しい		if(a == b)
	！＝	等しくない		if(a != b)
	＞	より大きい		if(a > b)
	＜	より小さい		if(a < b)
	＞＝	より大きいか等しい		if(a >= b)
	＜＝	より小さいか等しい		if(a <= b)

29.3 優先順位

式は優先順位の高い順に評価され、同順位なら結合規則にしたがって、左→右または右→左に評価されます。優先順位を変えるには( )を用います。

優先順位 / 演算子	1 高	2	3	4	5	6	7	8	9	10	11	12	13	14	15 低
関数、カッコ	( )														
配列	[ ]														
構造体	. ->														
型		(型) sizeof													
ポインタ		* &													
インクリメント ／デクリメント		++ --													
算術		+ -	* / %	+ -	※優先順位２は、単項演算子　例）-a 優先順位４は、二項演算子　例）a-b										
関係						< <= > >=	== !=								
ビット		~			<< >>			&	^	|					
論理		!									&&	||			
条件													?:		
代入														= += *= など	
コンマ															,
結合規則	左→右	左←右	左　→　右										左←右		左→右
演算子 / 優先順位	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

29.4 型が混合したときの演算

違う型が混合した計算の場合、下記のような決まりで演算されます。

・char と unsigned char と short は int
・unsigned short は unsigned int
・float は double

} に変換され、

long double＞double＞unsigned long＞long＞unsigned int＞int

の優先度で型の高い方に変換されて演算されます。
(ただし、char＜short＝int とする)

29.5 printf関数の使い方

printf 関数の呼び出しは次の形式で記述します。

```
        ret = printf( fmt , arg1 , arg2 , ... ) ;

ただし  ret             :int 型。出力した文字数(エラー時は -1)。
        fmt             :char 型へのポインタ型。フォーマット変換を指定する文字列。
        arg1,arg2,...   :定数または表示データの格納された変数や式(書式に依存)。
```

(A) printf 関数は、fmt で示される文字列をそのまま表示します。

ただし

(B) 文字列の中に「%」があると、それに続く文字により、2番目以降(arg1)の引数の示す値を変換し、変換結果を文字列に埋め込んで表示します。

(C) 「%」の後に続くものはオプションと変換指定文字です。

%[オプション]変換指定文字　　[ ]は省略可

(1)変換指定文字

変換指定文字	引数の型	表示のされ方
d	int	10 進数
o	int	8進数
x	int	16 進数
u	int	符号なし 10 進数
e	double	[-]m.nnnnnne[±]xx の形式の 10 進浮動小数点数 (n の桁数はオプションで可変だが標準では6桁)
f	double	[-]m.nnnnnn の形式の 10 進浮動小数点数 (n の桁数はオプションで可変だが標準では6桁)
c	int	単一文字
s	char *	文字列
p	void *	ポインタ
%		%

(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

```
 2: #include <stdio.h>
 3:
 4: void main( void )
 5: {
 6:   int    a = 64  ,  b = -64;
 7:   double c = 6.4;
 8:   char   data[20] = { "I Love You !" };
 9:
10:   printf("decimal     :%20d%20d¥n",a,b);
11:   printf("unsigned    :%20u%20u¥n",a,b);
12:   printf("octal       :%20o%20o¥n",a,b);
13:   printf("hexadecimal :%20x%20x¥n",a,b);
14:   printf("character   :%20c¥n",a);
15:   printf("double      :%20f¥n",c);
16:   printf("double - e  :%20e¥n",c);
17:   printf("string      :%20s¥n",data);
18: }
```

実行結果

```
decimal     :                  64                 -64
unsigned    :                  64               65472
octal       :                 100              177700
hexadecimal :                  40                ffc0
character   :                   @
double      :            6.400000
double - e  :        6.400000e+00
string      :        I Love You !
```

解説

2 行目: printf 関数を使用するため「 stdio.h 」をインクルードします。

6～8 行目: 必要となる変数を初期値付きで宣言します。

10 行目: 変換指定 %d により 64 と -64 が表示されます。

11 行目: 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 行目: 変換指定 %o により 64 と -64 の 8 進表現が表示されます。

13 行目: 変換指定 %x により 64 と -64 の 16 進表現が表示されます。

14 行目: 変換指定 %c により 64 を文字に変換し @ が表示されます。

15 行目: 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 行目: 変換指定 %e により 6.4 が指数形式で表示されます。

17 行目: 変換指定 %s により char 型配列の内容が文字列として表示されます。

29.6 scanf関数の使い方

scanf 関数の呼び出しは次の形式で記述します。

```
        ret = scanf( fmt , arg1 , arg2 , ... ) ;

ただし  ret             :int 型。読み込んで変換されたデータの数(エラー時は -1)。
        fmt             :char 型へのポインタ型。フォーマット変換を指定する文字列。
        arg1,arg2,...   :変換したデータの格納先を示すアドレスや式(書式に依存)。
```

(A) キーボードから改行キーが入力されるまで文字をバッファに入力する。改行が入力されてはじめて変換作業が始まり、バッファから読み込まれて処理される。バッファの管理はライブラリ関数側で自動的に行われる。

(B) fmt が示す文字列中の「%」に続く「オプション」(省略可)と「変換指定文字」に従って変換を行い、結果を2番目以降(arg1)の引数が示す格納先に格納する。

(C) fmt が示す文字列は、原則として「%」と「変換指定文字」の列で構成する。ただし、例外的に「 % 」の付かない文字を挿入することがある。(F)参照

(D) 変換指定文字が「c」以外の場合、改行やスペースなどの非印字文字は区切り記号とみなされる。また、変換データよりも前にある場合は読み飛ばされ、後の場合はバッファ内に読み残される。したがって、文字列の読み込みの際にスペースも含めて入力したい場合、変換指定文字の「s」は使えない。
変換結果が正常でない場合、例えば 10 進入力にもかかわらず数字以外のものが入力されていた場合は、その文字を読み込まずに作業は終了する。

(E) 変換指定文字が「c」の場合は、ASCII コードはすべて(改行やスペースも含め)処理の対象となる。そのため、それ以前の scanf 関数の処理によってバッファに読み残された非印字文字があれば、それを読んでしまうことになる。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。

30. 参考文献

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

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

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

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

http://japan.renesas.com/

の「マイコン」→「H8 ファミリ」、または「マイコン」→「Tiny」でご覧頂けます

※リンクは、2009 年 2 月現在の情報です。