マイコンカーラリー

H8/3687F

実習マニュアル

ルネサス統合開発環境版

第 1.02 版

2007.09.14

ジャパンマイコンカーラリー実行委員会

注 意 事 項 (rev.1.2)

著作権

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

禁止事項

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

転載、複製

本マニュアルの転載、複製については、文章によるジャパンマイコンカーラリー実行委員会の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、こと前にマイコンカー公式ホームページ(http://www.mcr.gr.jp/)などを通じて公開される情報に常にご注意ください。

連絡先

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

目　次

1. CPU仕様

1.1 仕様

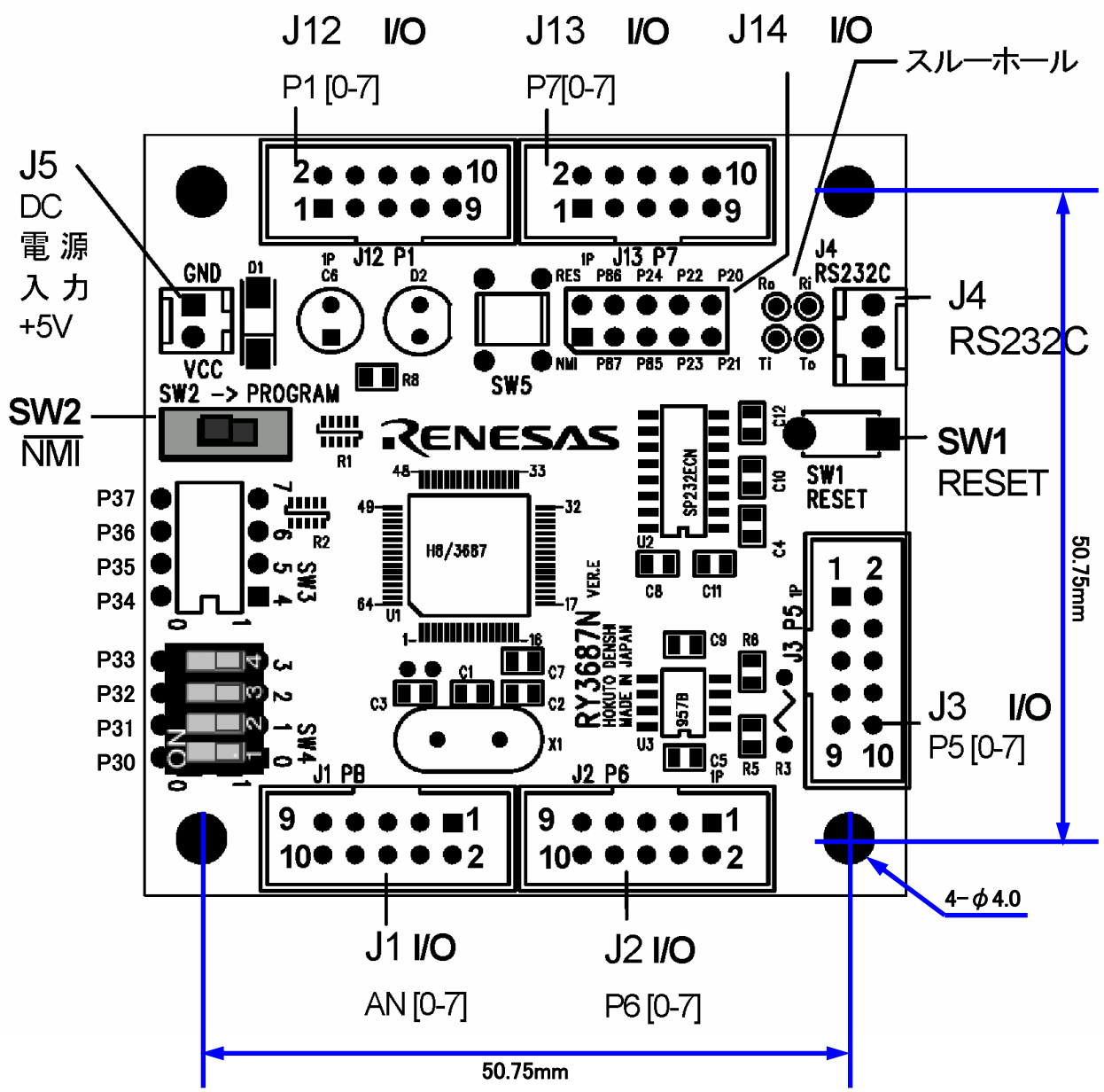
H8/3687F と H8/3048F-ONE の仕様を比較します。今まで H8/3048F-ONE を使用していた場合、どう違うか比べてみてください。

<table>
<tr><th></th><th>RY3048Foneボード</th><th>RY3687Nボード</th><th>解説</th></tr>
<tr><td>CPU</td><td>(株)ルネサステクノロジ製
H8/3048F-ONE</td><td>(株)ルネサステクノロジ製
H8/3687F</td><td>ボードの型式は、RY3687NでCPUの型式とは関係ありません</td></tr>
<tr><td>内蔵メモリ</td><td>フラッシュROM:128KB
RAM: 4KB</td><td>フラッシュROM:56KB
RAM: 4KB
ただし、RAMは2KBごとに分かれている</td><td>ルネサス統合開発環境無償評価版でビルドできる容量は64KBまで。H8/3687Fは、すべてのメモリを使うことができる</td></tr>
<tr><td>アドレス空間</td><td>モード7(外部メモリを接続できないモード):
0x00000～0xfffff (1MB)
モード5(外部メモリを接続できるモード):
0x00000～0xfffff (1MB)
モード6(外部メモリを接続できるモード):
0x000000～0xffffff (16MB)</td><td>0x0000～0xffff (64KB)
外部メモリを接続することはできない</td><td>H8/3687は外部のメモリを接続することはできない</td></tr>
<tr><td>マイコンの動作周波数</td><td>2.0～25.0MHz</td><td>2.0～20.0MHz</td><td>H8/3687Fは最大20MHzまで</td></tr>
<tr><td>ボードのクリスタル値</td><td>24.576MHz</td><td>14.7456MHz</td><td></td></tr>
<tr><td>動作電圧</td><td>4.5～5.5V</td><td>動作周波数が2.0～10.0MHzの場合:3.0～5.5V
動作周波数が10.0～20.0MHzの場合:4.0～5.5V</td><td>H8/3687Fは、10.0MHz以下の動作クロックの場合、3.0Vから動作する</td></tr>
<tr><td>基板外形</td><td>W70×D59×H13mm
(実寸)</td><td>W60×D60×H13mm
(実寸)</td><td></td></tr>
<tr><td>コネクタ</td><td>
<table>
<tr><th>コネクタ(ポート)</th><th>I/O数</th></tr>
<tr><td>J1(P7)</td><td>8※</td></tr>
<tr><td>J2(PA)</td><td>8</td></tr>
<tr><td>J3(PB)</td><td>8</td></tr>
<tr><td>J6(P3,4など)</td><td>18</td></tr>
<tr><td>J8(P1,2,5など)</td><td>29</td></tr>
<tr><td></td><td></td></tr>
<tr><td>合計</td><td>71</td></tr>
</table>
※J1は入力専用</td><td>
<table>
<tr><th>コネクタ(ポート)</th><th>I/O数</th></tr>
<tr><td>J1(PB)</td><td>8※</td></tr>
<tr><td>J2(P6)</td><td>8</td></tr>
<tr><td>J3(P5)</td><td>8</td></tr>
<tr><td>J12(P1)</td><td>7</td></tr>
<tr><td>J13(P7)</td><td>6</td></tr>
<tr><td>J14</td><td>6</td></tr>
<tr><td>合計</td><td>43</td></tr>
</table>
※J1は入力専用</td><td>RY3048Foneボードは、71ビット分の入出力ができる(ポート7の8ビットは入力のみ)

RY3687Nボードは、43ビット分の入出力ができる(ポートBの8ビットは入力のみ)</td></tr>
<tr><td>ディップスイッチ</td><td>P63～P60に接続(4ビット分)</td><td>P33～P30に接続(4ビット分)
ディップスイッチを追加することによりP37～P34に増設可能</td><td></td></tr>
</table>

※P33…ポート 3 の bit3 という意味です。

1.2 外観



J1、J2、J3 J12,J13	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 B、J2 はポート 6、J3 はポート 5、J12 はポート 1、J13 はポート 7 に接続されています。 これらのコネクタは未実装です。各自でコネクタを実装してください。
J4	RS-232C 接続用 3 ピンコネクタ J4 に接続するコネクタは、同封されています。RS-232C 側は線が剥き出しですので RS-232C コネクタを各自で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 J5 に接続するコネクタは、同封されています。ケーブル長は約 30cm 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。** **※アルカリ乾電池は、1.5V4 本で使用すると、6V 以上となります。5.5V を超えてしまいますので、電源安定化素子(三端子レギュレータなど)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 1.2V ですので、4 本で使用すると 4.8V となります。適正範囲内なので使用可能です。**

SW1	リセットスイッチ 押している間、CPU はリセット状態です。離すとリセットが解除されます。
SW2	プログラム書き込みスイッチ CPU ボードにプログラムを書き込むときは、「PROGRAM」側（内側）にします。書き込んだプログラムを実行するときは、その逆側(外側)にします。**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。**
SW4	4 ビットディップスイッチ ポート 3 の bit3～0 に接続されています。オムロン製の A6D-4103 が実装されています。
SW3	4 ビットディップスイッチ（未実装） ポート 3 の bit7～4 に接続されています。未実装ですので、必要なら各自でスイッチを実装してください。実装するスイッチは、オムロン製の A6D-4103 を推奨します。

1.3 コネクタのピン配置

J1 I/O

No	信号名	No	信号名
1	Vcc	2	PB7/AN7
3	PB6/AN6	4	PB5/AN5
5	PB4/AN4	6	PB3/AN3
7	PB2/AN2	8	PB1/AN1
9	PB0/AN0	10	GND

J2 I/O

No	信号名	No	信号名
1	Vcc	2	P67/FTIOD1
3	P66/FTIOC1	4	P65/FTIOB1
5	P64/FTIOA1	6	P63/FTIOD0
7	P62/FTIOC0	8	P61/FTIOB0
9	P60/FTIOA0	10	GND

J3 I/O

No	信号名	No	信号名
1	Vcc	2	P57/SCL
3	P56/SDA	4	P55/*WKP5/*ADTRG
5	P54/*WKP4	6	P53/*WKP3
7	P52/*WKP2	8	P51/*WKP1
9	P50/*WKP0	10	GND

J12 I/O

No	信号名	No	信号名
1	Vcc	2	P17/*IRQ3/TRGV
3	P16/*IRQ2	4	P15/*IRQ1/TMIB1
5	P14/*IRQ0	6	NC
7	P12	8	P11/PWM
9	P10	10	GND

J13 I/O

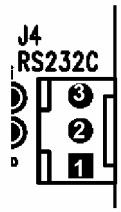
No	信号名	No	信号名
1	Vcc	2	NC
3	P76/TMOV	4	P75/TMCIV
5	P74/TMRIV	6	NC
7	P72/TXD_2	8	P71/RXD_2
9	P70/SCK3_2	10	GND

J14

No	信号名	No	信号名
1	*NMI	2	*RES
3	P87	4	P86
5	P85	6	P24
7	P23	8	P22/TXD
9	P21/RXD	10	P20/SCK3

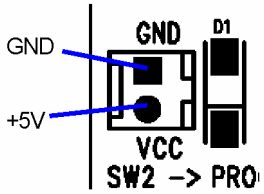
注意！　＊は負論理を示します

J4 RS232C

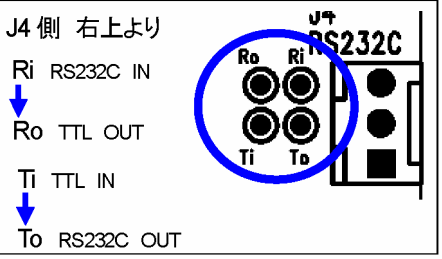


No	信号名
1	P22/TXD
2	GND
3	P21/RXD

J5DC電源入力

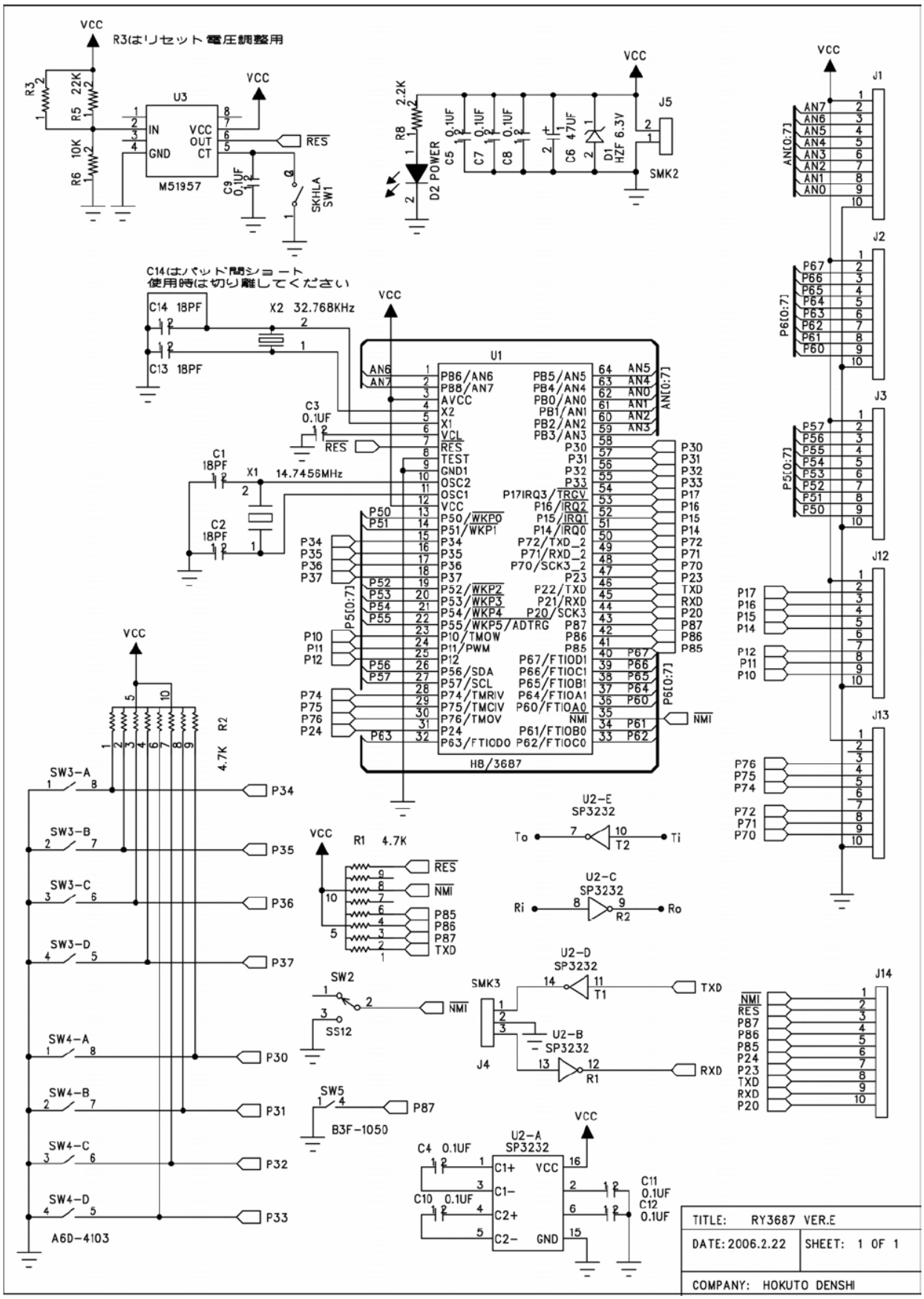


スルーホール

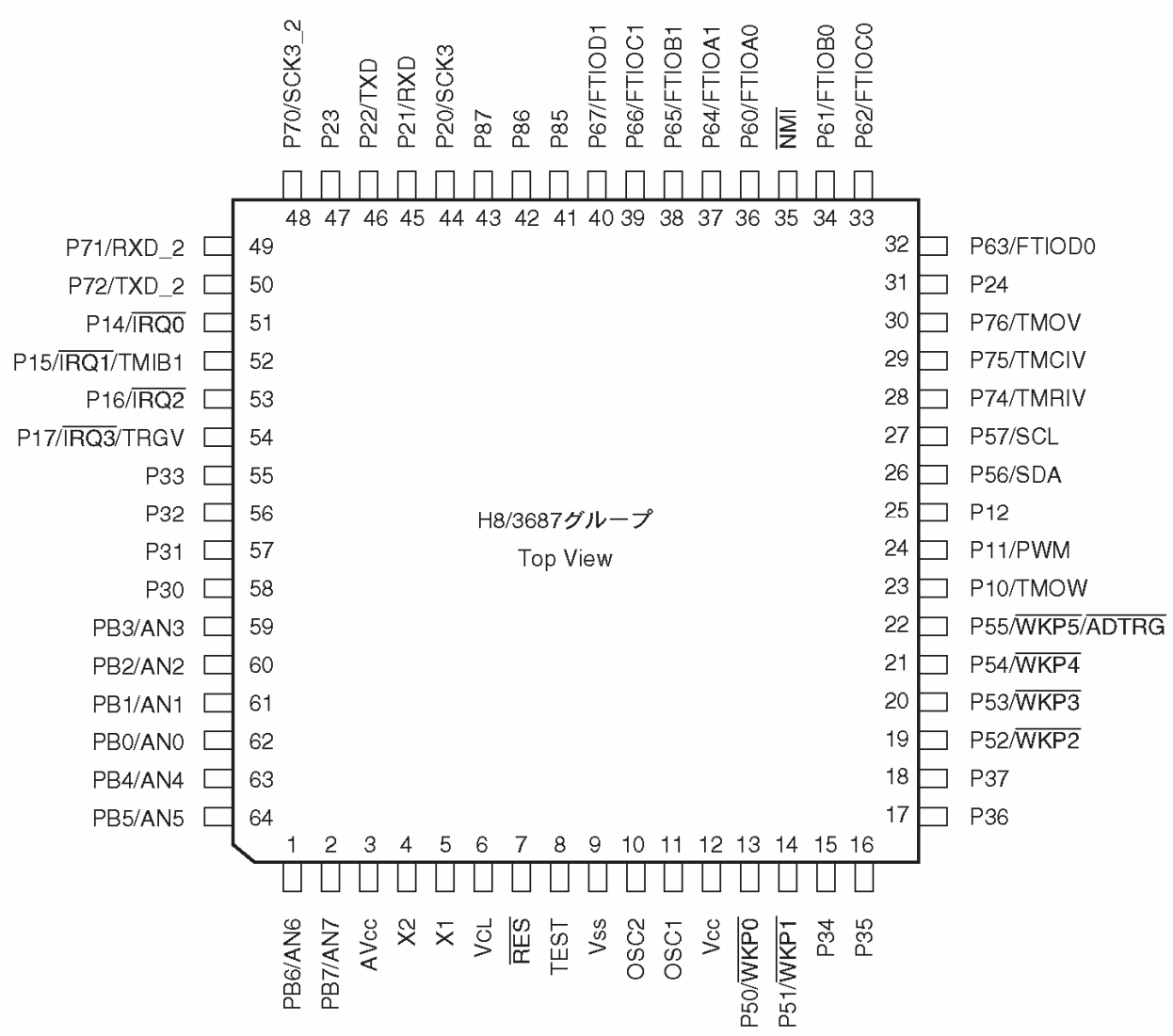


※NC…Non Connect　未接続のことです。

1.4 回路図


R3はリセット電圧調整用
C14はパッド間ショート
使用時は切り離してください
U1
H8/3687
U3
M51957
U2-A
SP3232
U2-B
SP3232
U2-C
SP3232
U2-D
SP3232
U2-E
SP3232
X1 14.7456MHz
X2 32.768KHz
SW3-A
SW3-B
SW3-C
SW3-D
SW4-A
SW4-B
SW4-C
SW4-D
A6D-4103
SW2
SS12
SW5
B3F-1050
J1
J2
J3
J4
J5
J12
J13
J14
SMK2
SMK3
TITLE: RY3687 VER.E
DATE: 2006.2.22
SHEET: 1 OF 1
COMPANY: HOKUTO DENSHI

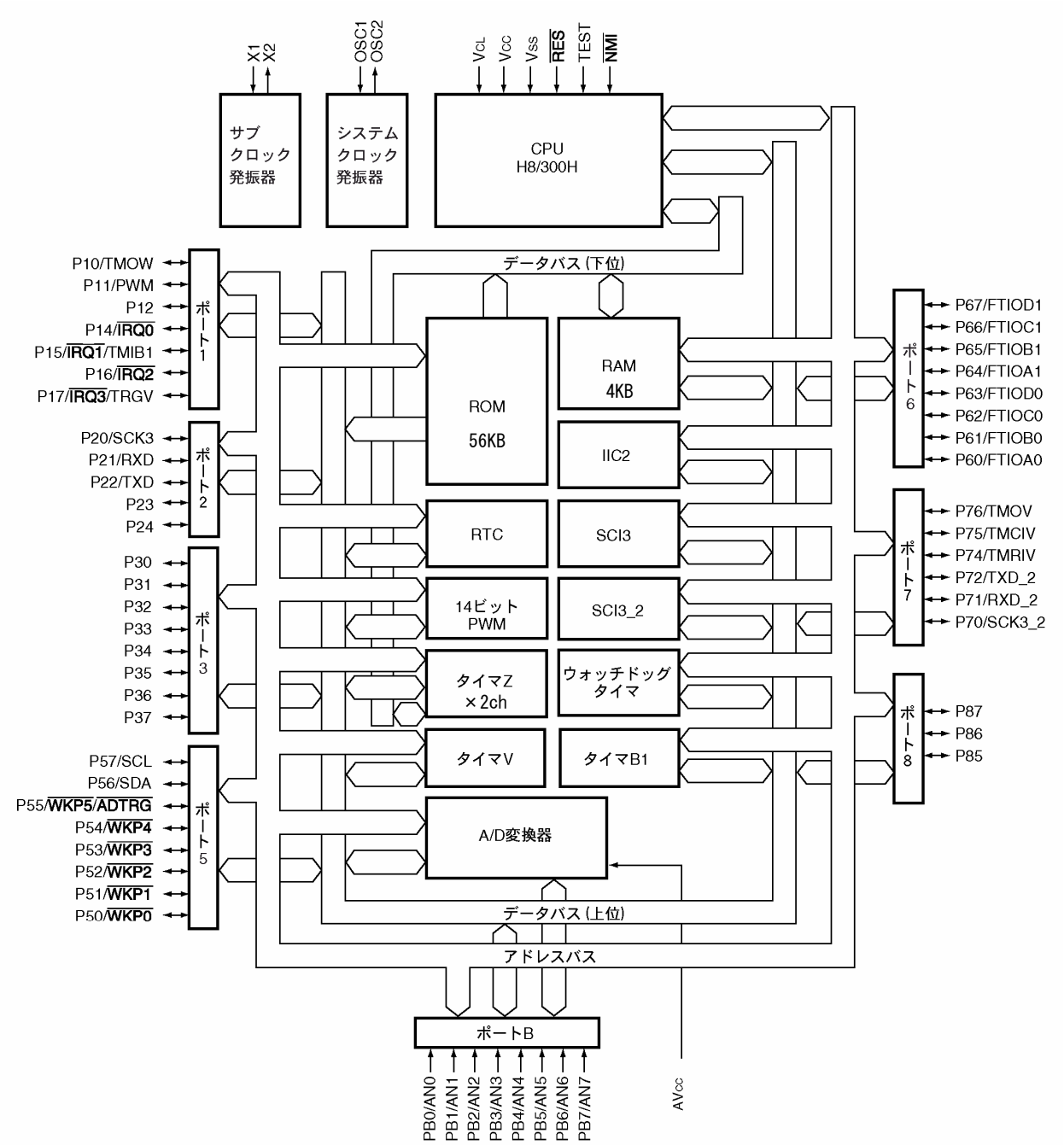
1.5 H8/3687Fのピン配置



1 ピンと 64 ピンの間の角が面取りされています。これを目印に、1 ピンが分かります。Vcc は＋側電圧の 5V、Vss は－側電圧の 0V を加えます。

1.6 内部機能

H8/3687F には、下記のような機能が内蔵されています。

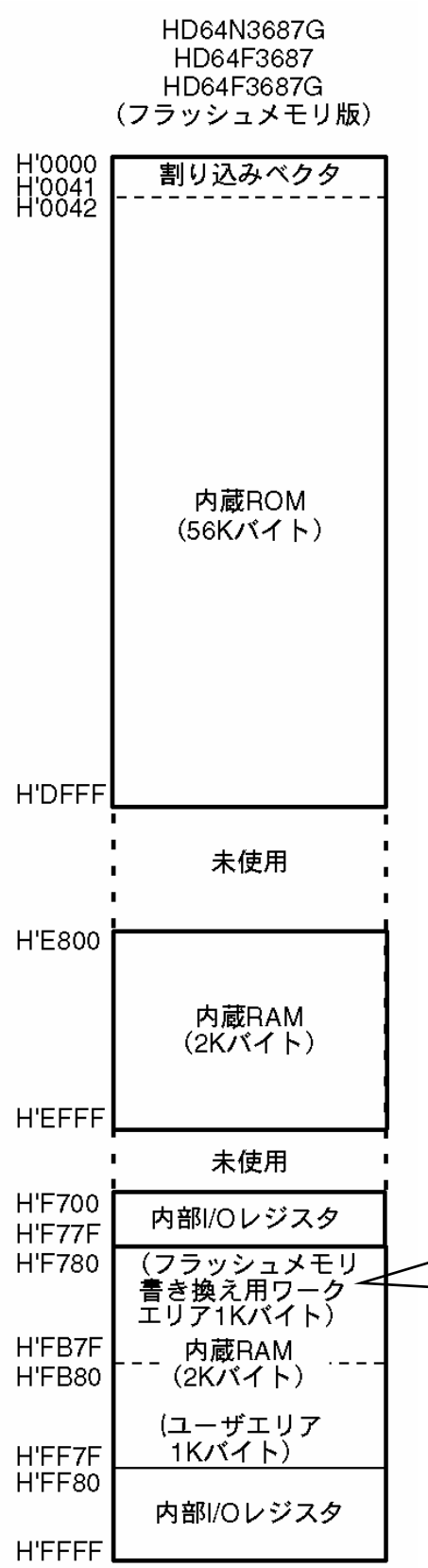


※用語の説明

IIC2	Inter Integrated Circuit	IIC という規格の通信をする機能です。2 は、Ver2 のことです。**2 つあるということではありません**。
SCI3	Serial Communication Interface	コンピュータ同士がデータの受け渡しを行う機能です。3 は Ver3 のことです。**3 つあるということではありません**。SCI3_2 は 2 つ目の SCI3 です。
RTC	Real Time Clock	時計機能のことです。

1.7 メモリマップ

H8/3687F の ROM 領域、RAM 領域、I/O レジスタ領域を下記に示します。



H8/3687F のメモリは、

・0x0000～0xdfff 番地が ROM (56KB)
・0xe800～0xefff 番地が RAM (2KB)
・0xf780～0xff7f 番地が RAM (2KB)

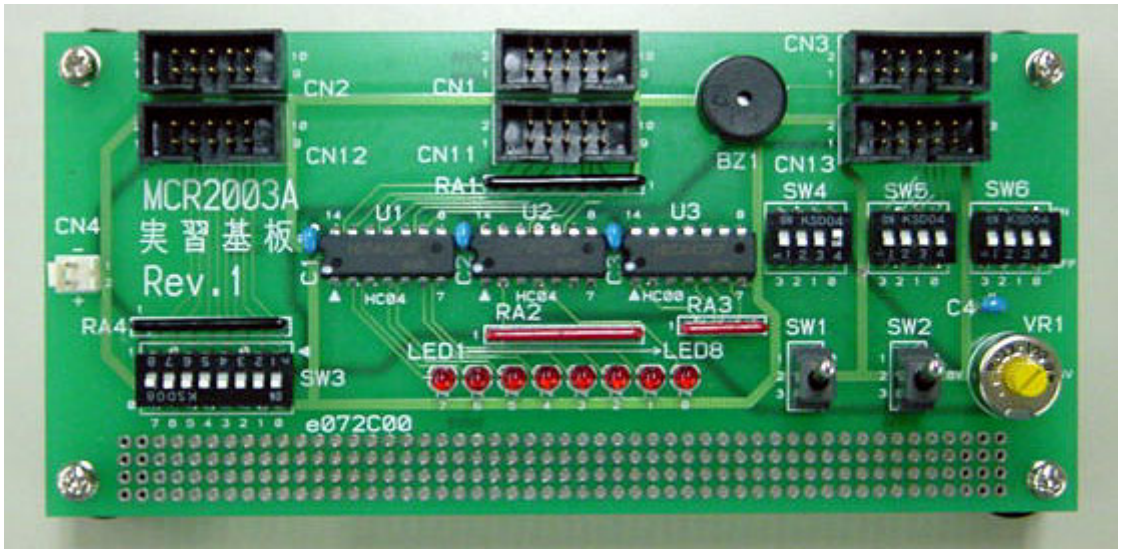
となります。

フラッシュメモリ書き換え用ワークエリアとは、プログラムを書き込むときに、マイコンがこのエリアを作業用として使うということです。プログラムを実行するときは、このエリアを自由に使えます。そのため、このコメントは無視して構いません。0xf780～0xff7f まで自由に使用することができます。

2. 実習基板

2.1 概要

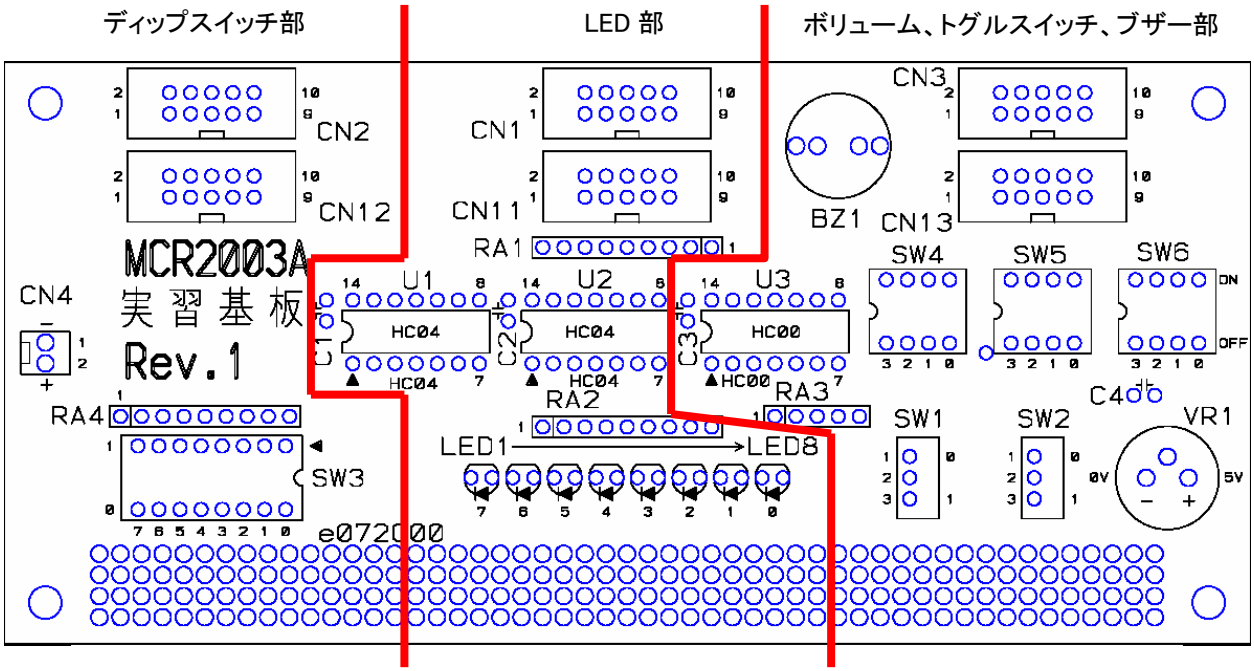
本実習マニュアルは、主に MCR2003A 実習基板を使って H8/3687F の機能について実習していきます。



▲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。

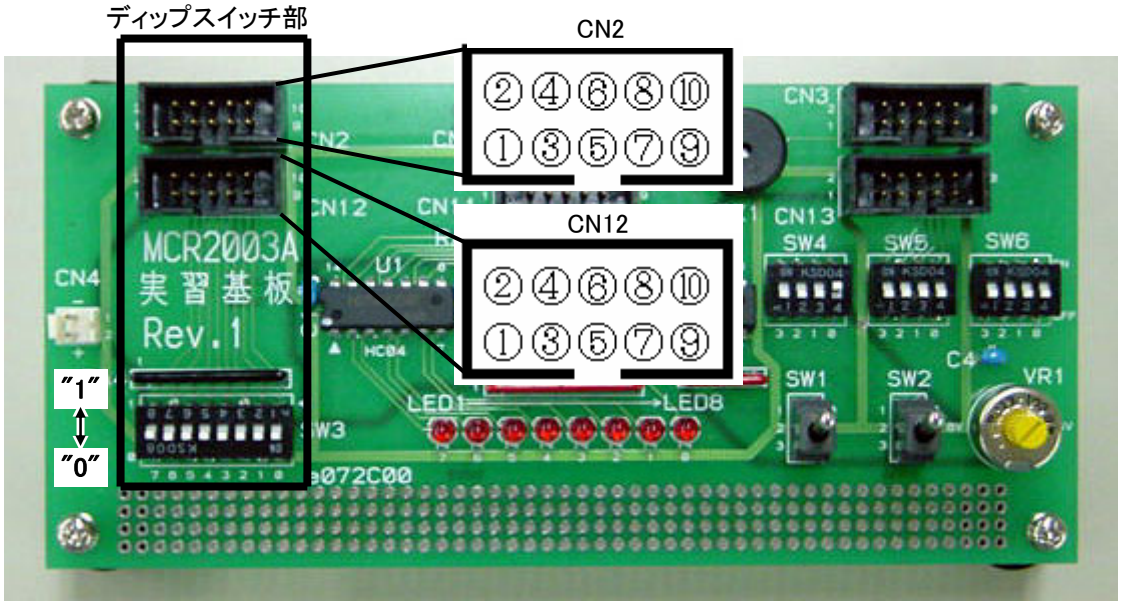
2.2 外観



2.3 ディップスイッチ部

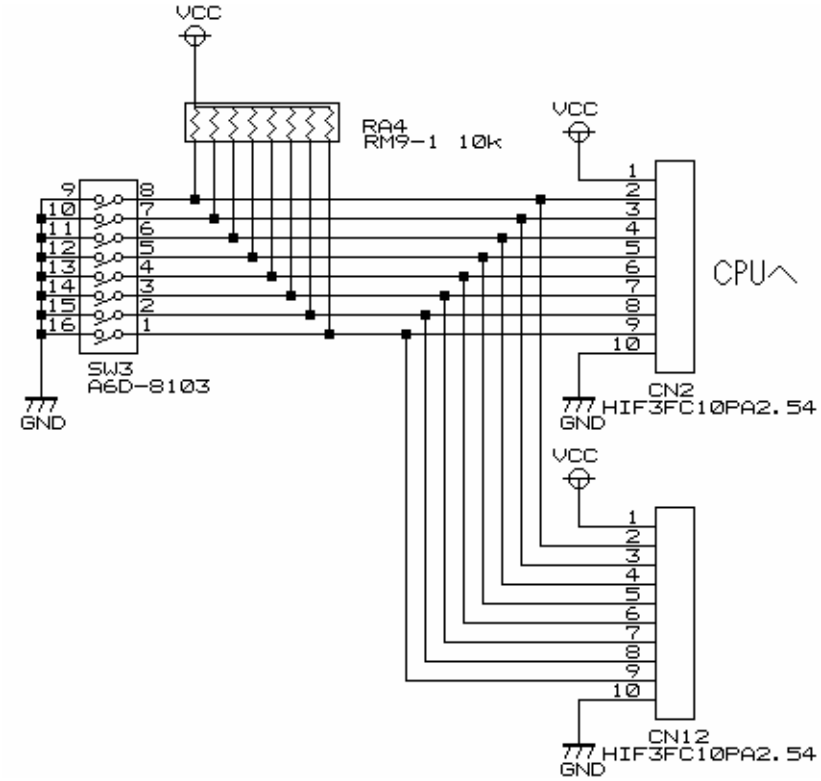
2.3.1 コネクタ

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。

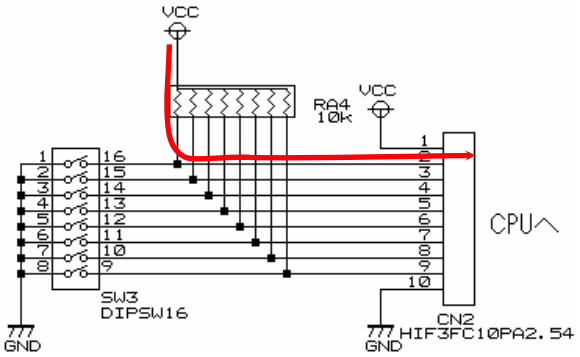


CN2,CN12 ピン番号	信号	“0”	“1”
1	＋5V	－	－
2	ディップスイッチ 7	下側	上側
3	ディップスイッチ 6	下側	上側
4	ディップスイッチ 5	下側	上側
5	ディップスイッチ 4	下側	上側
6	ディップスイッチ 3	下側	上側
7	ディップスイッチ 2	下側	上側
8	ディップスイッチ 1	下側	上側
9	ディップスイッチ 0	下側	上側
10	GND	－	－

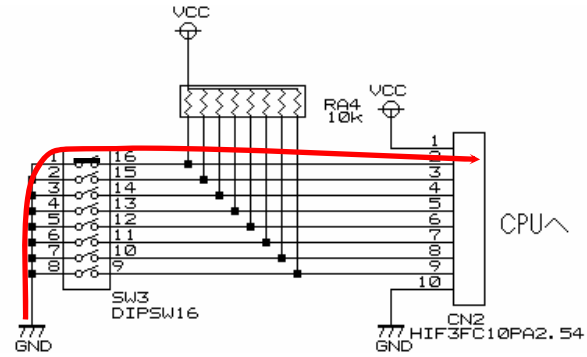
2.3.2 回路



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。

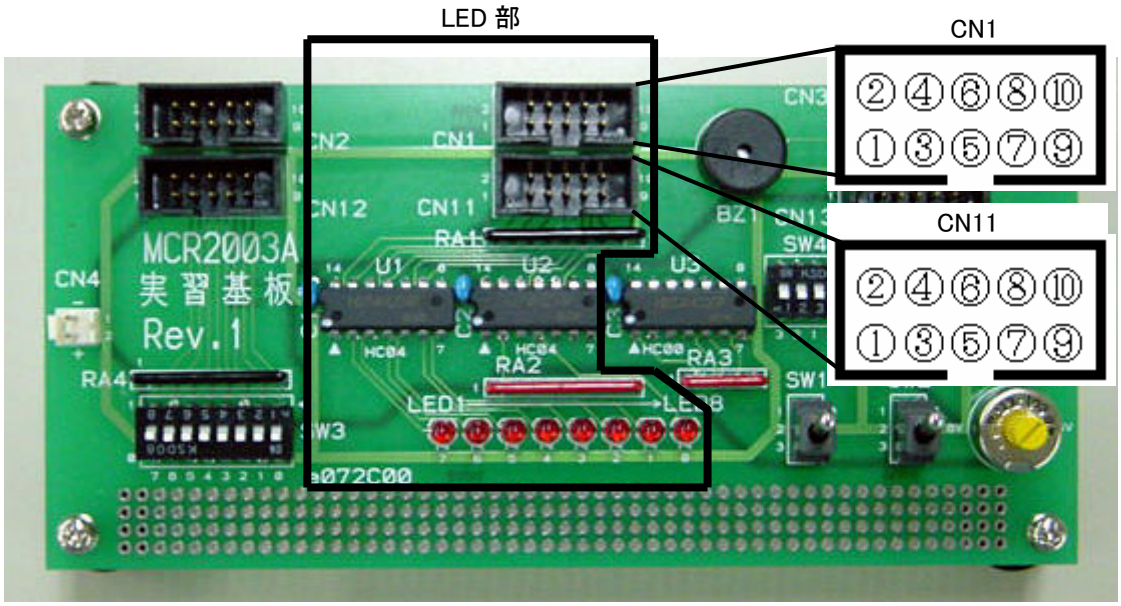


スイッチが ON なら、GND と直結になり、"0"が出力されます。

2.4 LED部

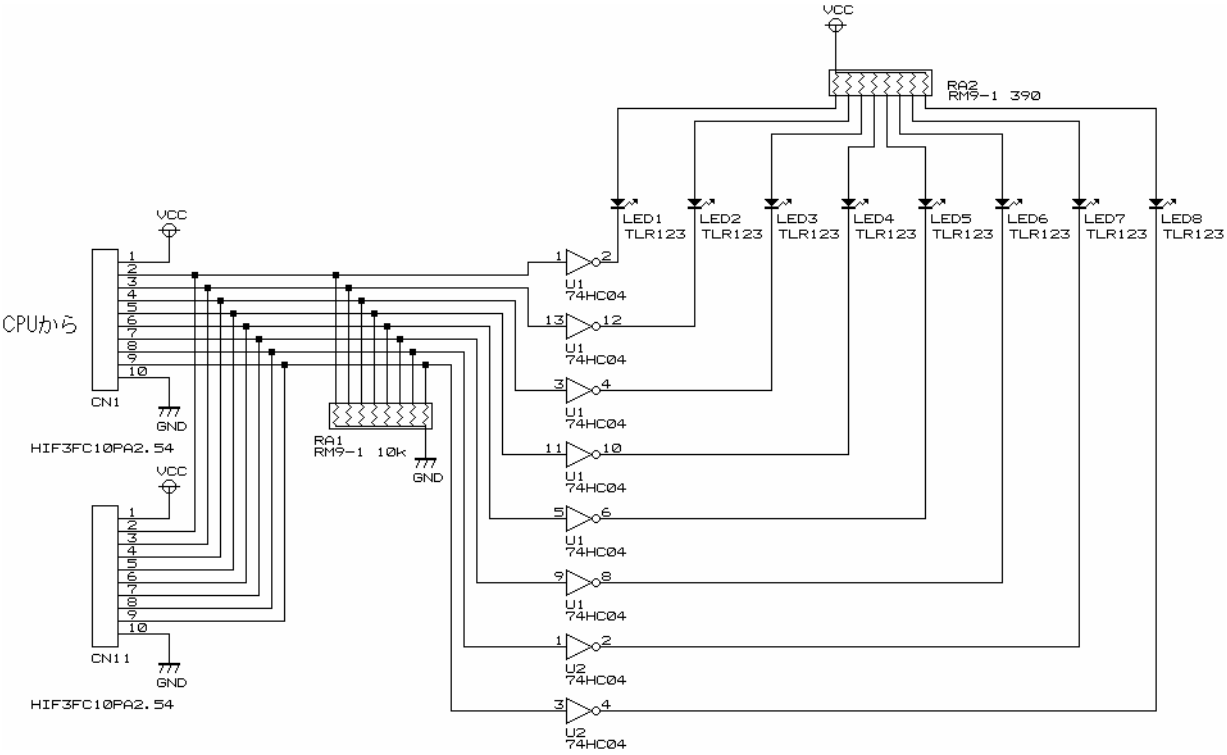
2.4.1 コネクタ

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。

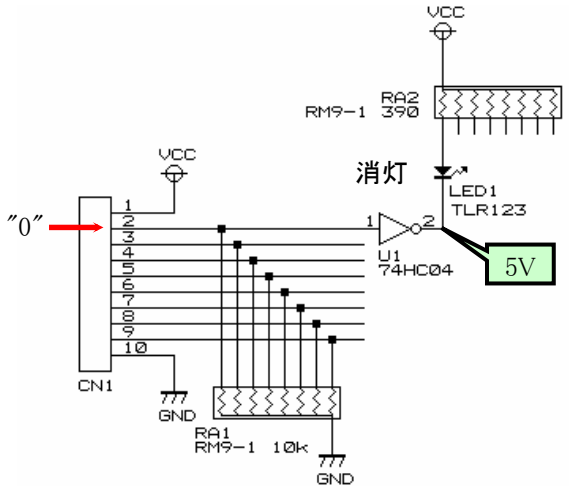


CN1,CN11 ピン番号	信号	“0”	“1”
1	＋5V	－	－
2	LED1(7)	消灯	点灯
3	LED2(6)	消灯	点灯
4	LED3(5)	消灯	点灯
5	LED4(4)	消灯	点灯
6	LED5(3)	消灯	点灯
7	LED6(2)	消灯	点灯
8	LED7(1)	消灯	点灯
9	LED8(0)	消灯	点灯
10	GND	－	－

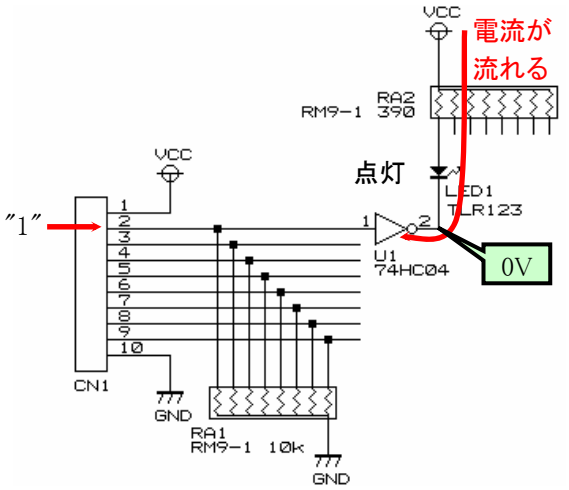
2.4.2 回路



2.4.3 回路説明



入力信号が"0"(0V)なら、LED は消灯です。

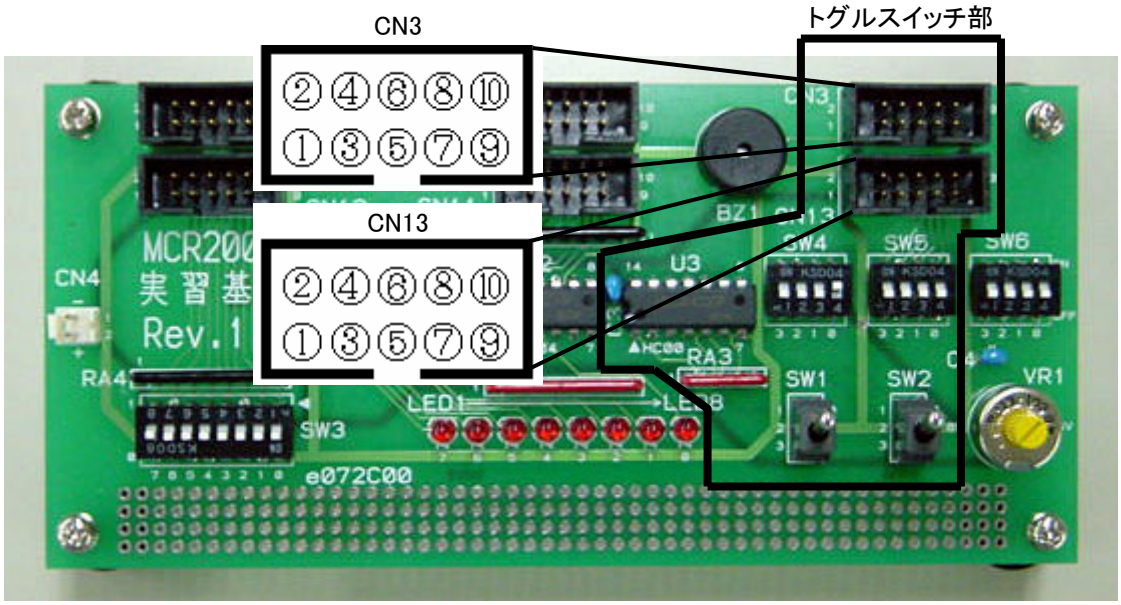


入力信号が"1"(5V)なら、LED は点灯します。

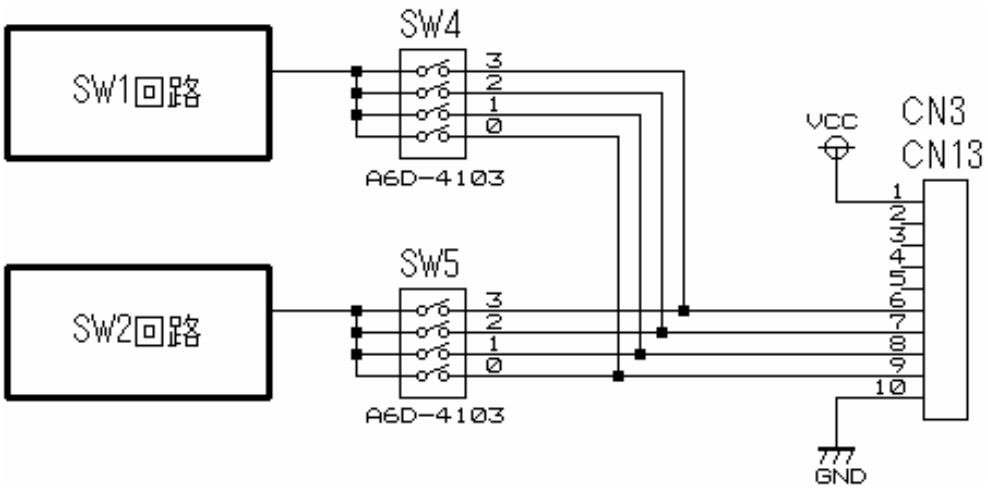
2.5 トグルスイッチ部

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。

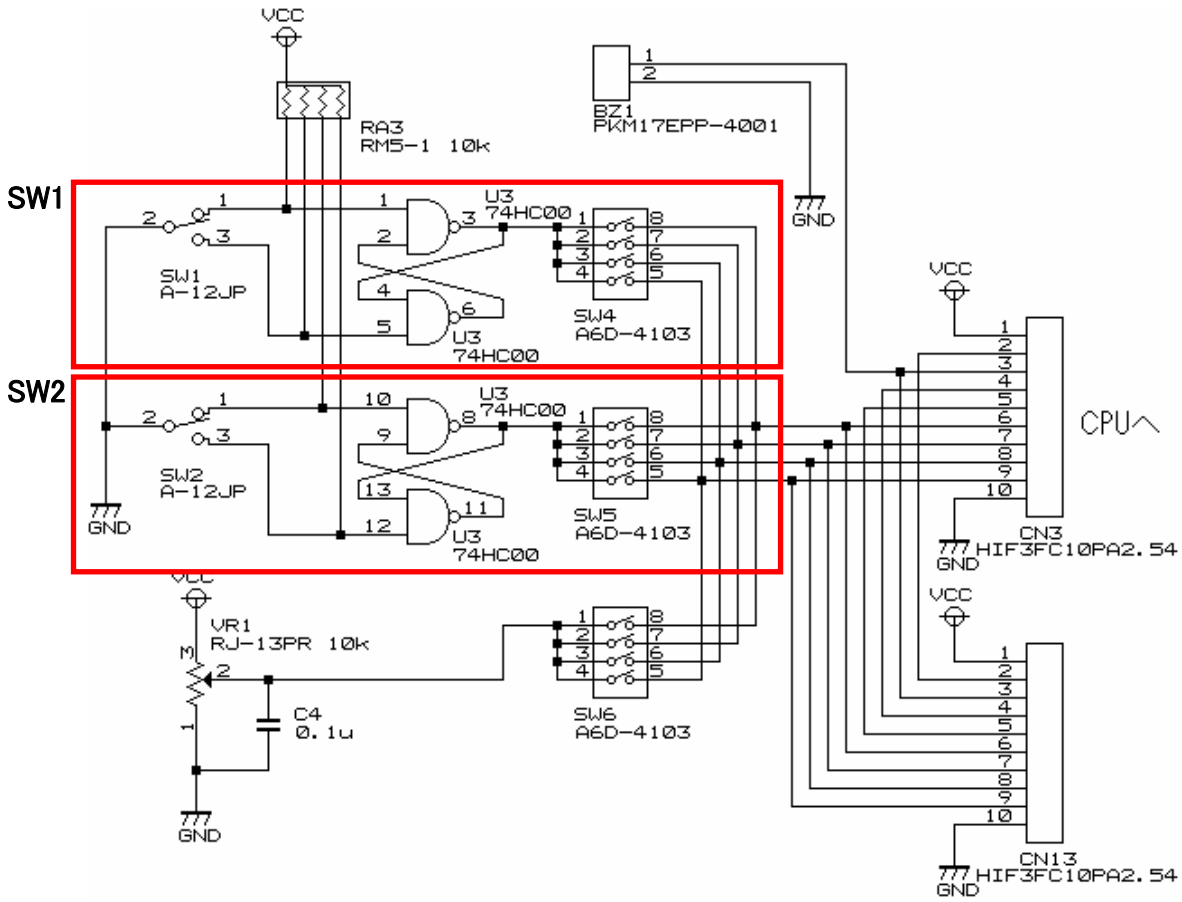


CN3(CN13)のどのピンへ出力するかは、SW1 の信号は SW4 で、SW2 の信号は SW5 で切り替えます。



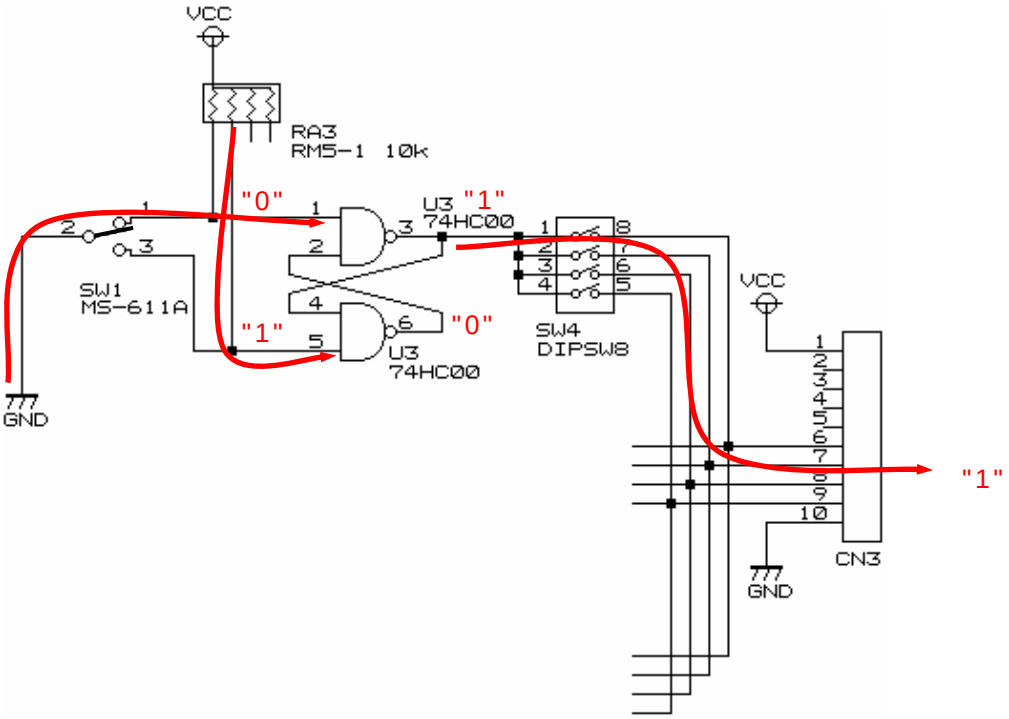
2.5.2 回路

□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

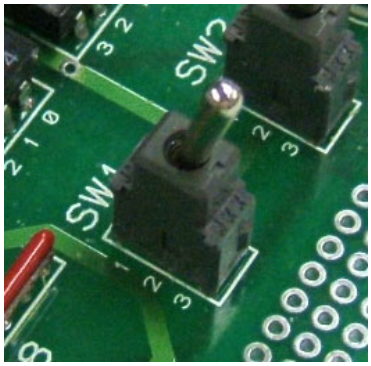
トグルスイッチは下記のような動作原理です。

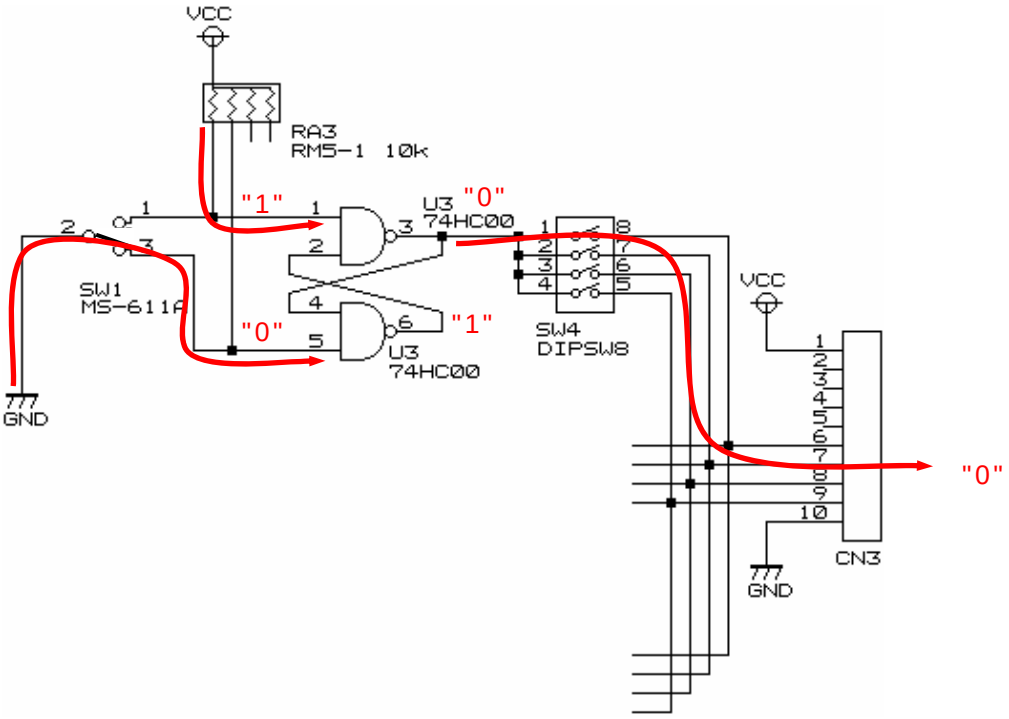


SW1 の 1－2 ピンを短絡させたとき、U3 の端子状態は、

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。

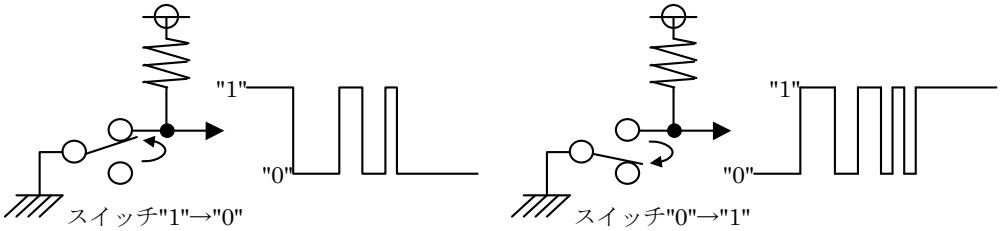
このように、スイッチを上下させることにより、出力が″0″、または″1″と変化します。なぜ単純なディップスイッチ部の様な回路ではいけないのでしょうか。理由は、次ページの参考資料を参照してください。

※参考資料－チャタリング防止回路

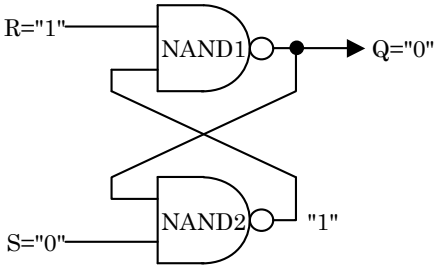
下図のように、通常のプルアップしたスイッチを"1"から"0"、もしくは"0"から"1"に変更した場合、何度か接点が付いたり離れたりして、最終的に"1"や"0"になります。これを「チャタリング」といいます。

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。

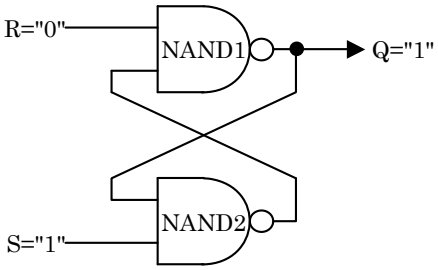
下図の"1"→"0"の例では、一度しか"1"→"0"にしていないつもりでも 3 回もスイッチを上げ下げしたと判断されてしまいます。更にやっかいなことは、発生するパルス数、収束するまでの時間が毎回違うということです。



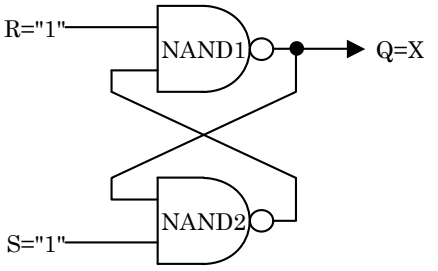
チャタリングを解消する回路の一つに「リセット・セット・フリップフロップ（RS-FF)」という回路があります。NAND 回路をループさせた形の回路です。R(Reset)端子に"1"が入力されると出力 Q は"0"になり、S(Set)端子に"1"が入力されると出力 Q は"1"になることからそう呼ばれています。動作原理は次のとおりです。



●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。



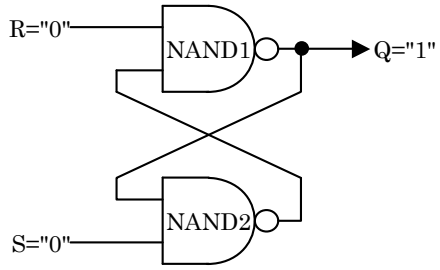
●入力(R,S)=("0","1")のとき
・NAND1 は R="0"のため、もう一方の入力が何であろうと無条件で出力"1"
・NAND2 は S="1"、もう一方の入力は NAND1 の出力"1"なので出力"0"
・NAND1 の R 側でない入力が"0"で確定するが R="0"のため、出力は変わらず"1"
結果、セット信号だけが"1"のため、出力 Q="1"となったと考えることができます。



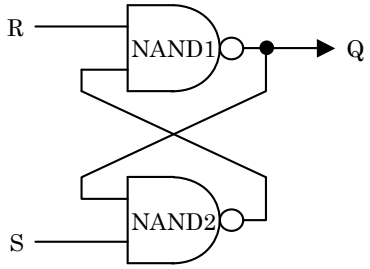
●入力(R,S)=("1","1")のとき
・NAND1、NAND2、共に入力が決まらなければ出力は決定されないので、NAND1 の出力 Q=X と仮定する
・NAND2 の入力は S="1"と X なので、出力は $\overline{X}$ となる
・NAND1 の入力は R="1"と $\overline{X}$ なので

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

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



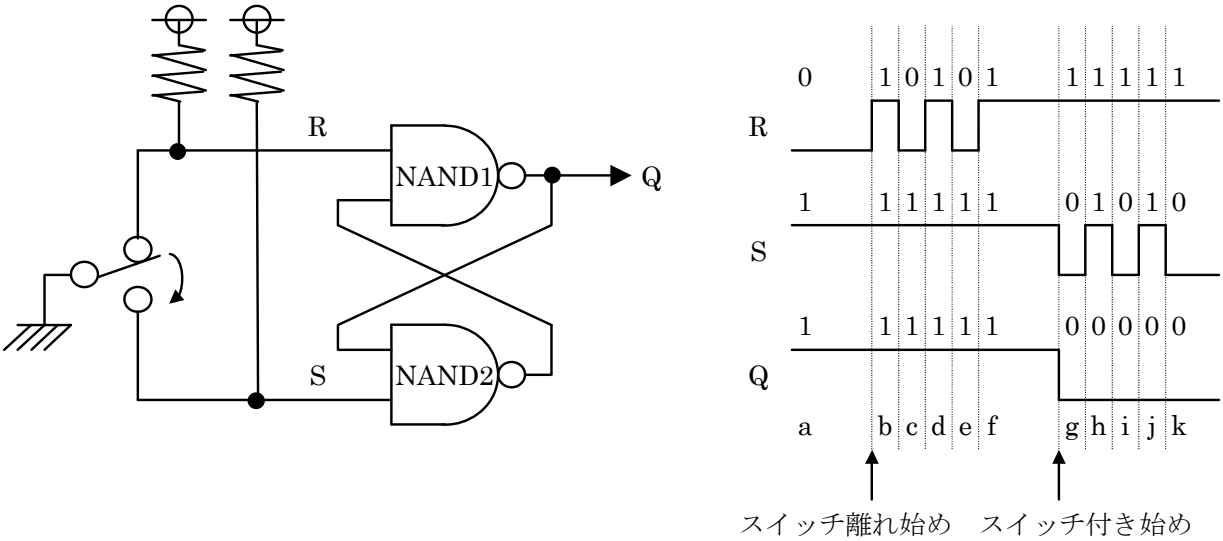
●入力(R,S)=(″0″,″0″)のとき
・NAND1 は R=″0″、もう一方の入力が何であろうと無条件で″1″
・NAND2 は S=″0″、もう一方の入力が何であろうと無条件で″1″
結果、Q=″1″となります。
ただし、(R,S)=(″0″,″0″)はリセットともセットとも呼べない状態です。Q=″1″となりセット状態ではありますが、リセット、セット、どちらかに″1″信号があるものとする論理に反する入力状態のため、通常は「禁止」とされています。ショートなどして回路が壊れるために禁止とするのではなく、論理が合わなくなるため禁止としています。



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。

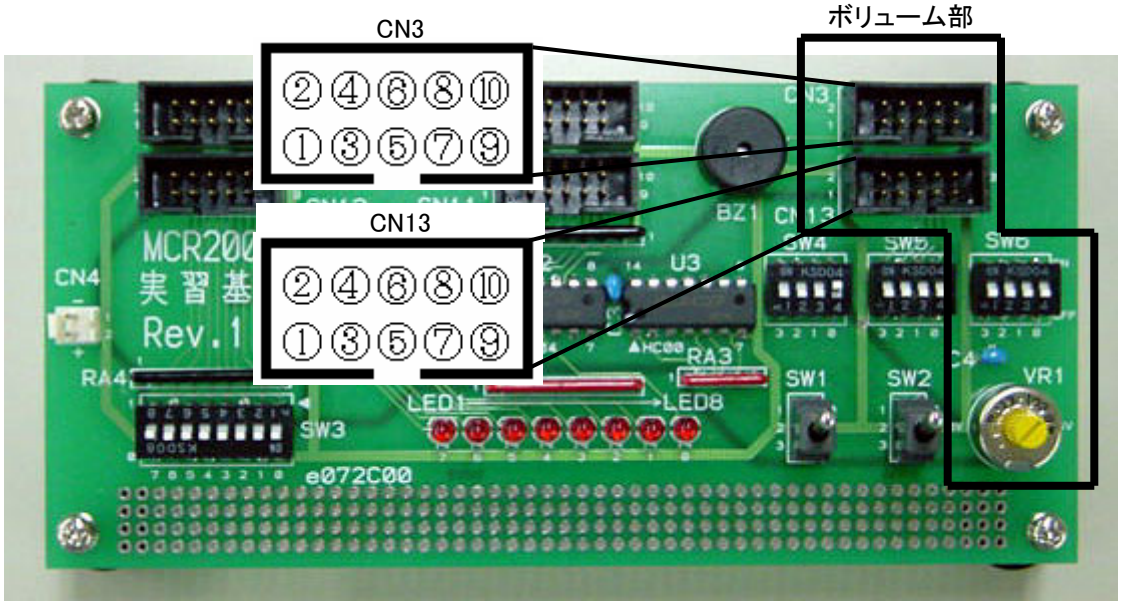


a. 最初、(R,S)=(″0″,″1″)のため、Q=″1″となります。
b. スイッチを下側に切り替え、R 側から接点が離れ始めた時です。(R,S)=(″1″,″1″)のため、Q は前の値を保持します。
c. (R,S)=(″0″,″1″)のため、Q=″1″となります。
d. (R,S)=(″1″,″1″)のため、Q は前の値を保持します。
e. (R,S)=(″0″,″1″)のため、Q=″1″となります。
f. (R,S)=(″1″,″1″)のため、Q は前の値を保持します。接点が中間になりました。R 端子、S 端子共にスイッチの接点には繋がっていない状態ですが、プルアップ抵抗があるので″1″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

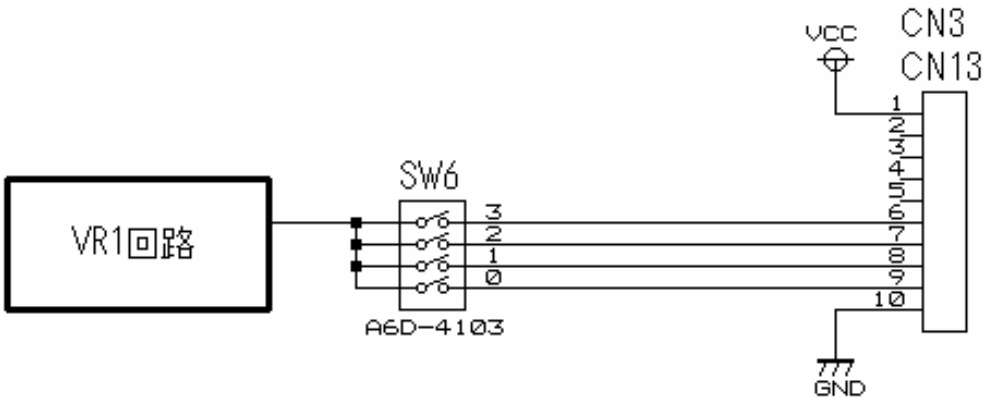
2.6 ボリューム部

2.6.1 コネクタ

VR1 の信号を、CN3(CN13)コネクタへ出力します。

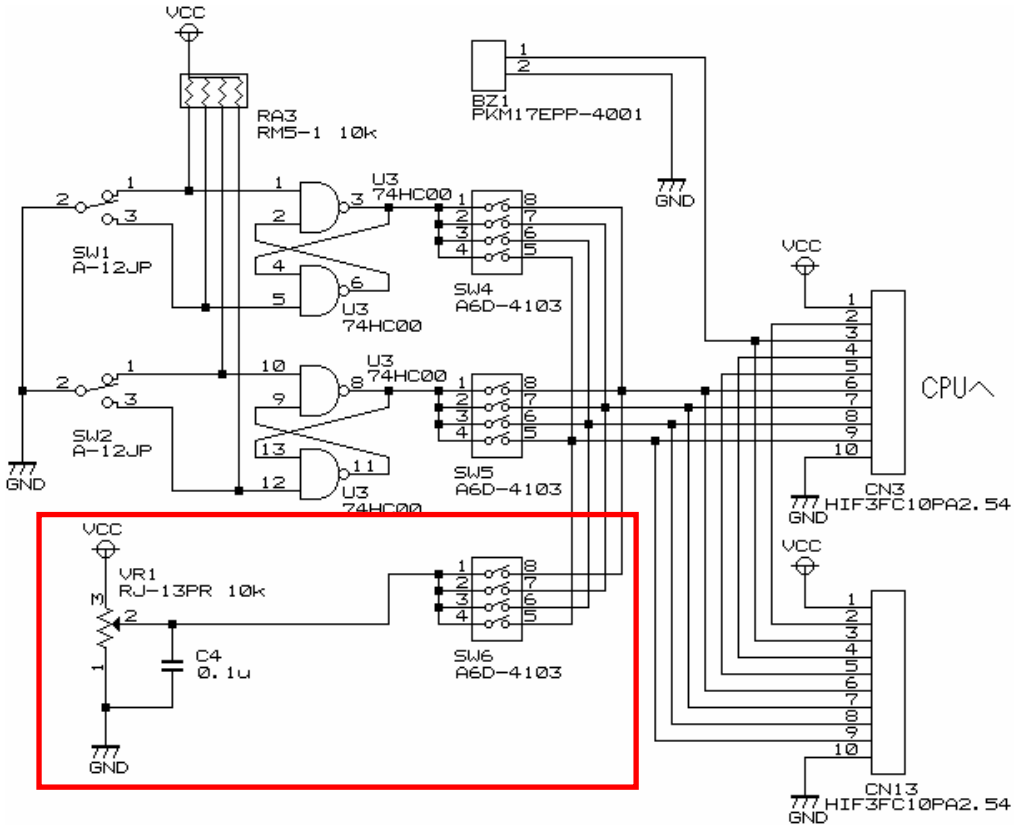


CN3(CN13)のどのピンへ出力するかは、SW6 で切り替えます。

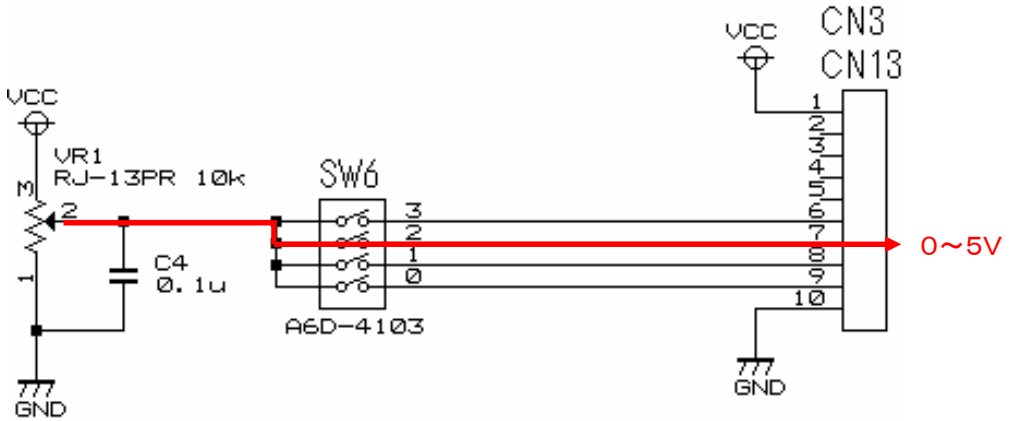


2.6.2 回路

□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

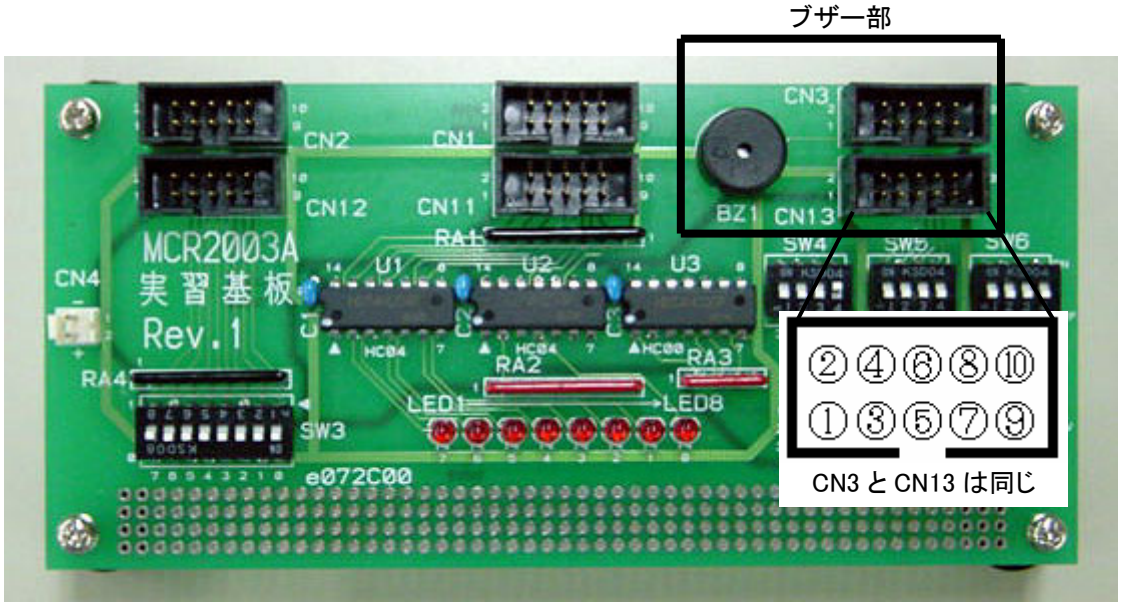


太いラインは、SW6 の 2 を ON にしたときの流れです。ボリュームの電圧が、SW6 を通して CN3 へ出力されます。C4 は、信号にノイズが乗ってアナログ変換値がばらつくのを防ぐ役割です。

2.7 ブザー部

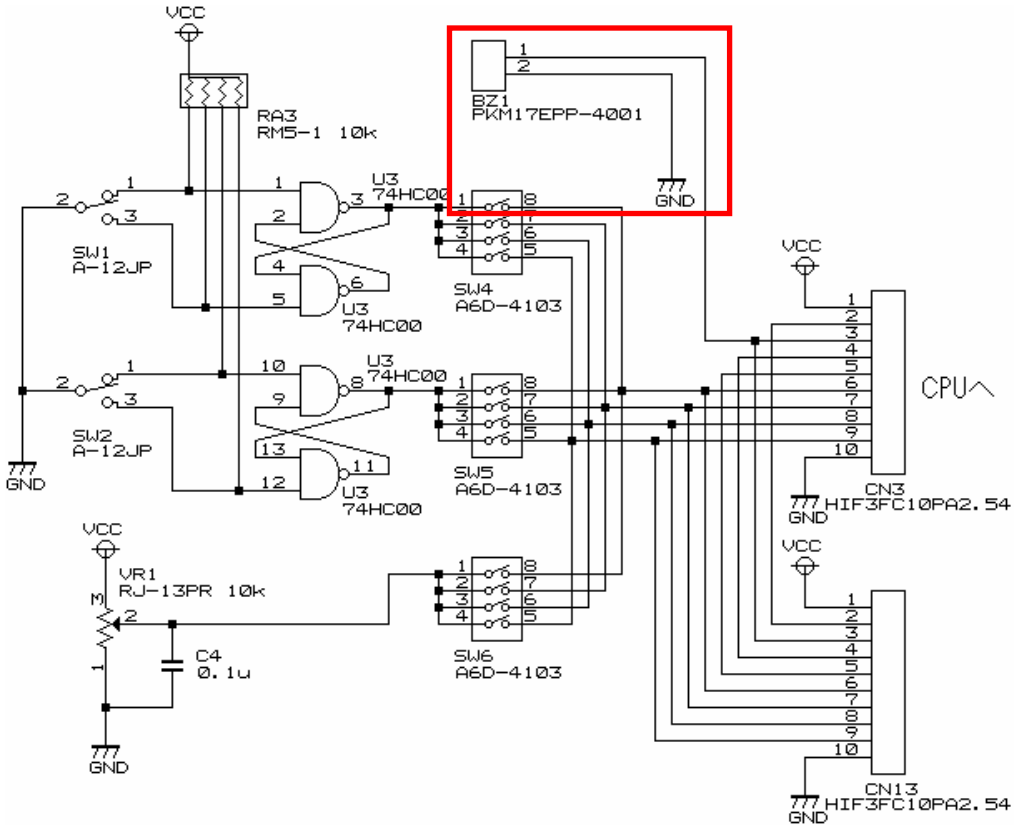
2.7.1 コネクタ

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

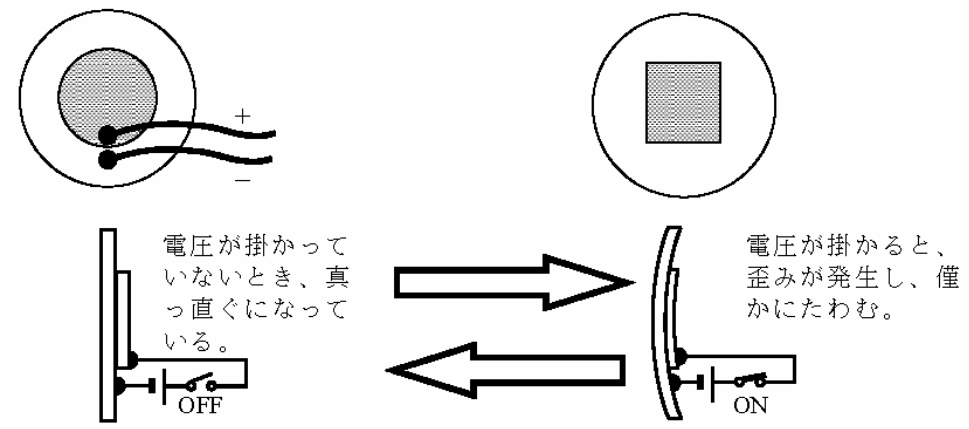
□で囲った部分が、ブザー部の回路です。ブザーは、CN3(CN13)の 3 ピンに接続されています。



2.7.3 ブザーの動作原理

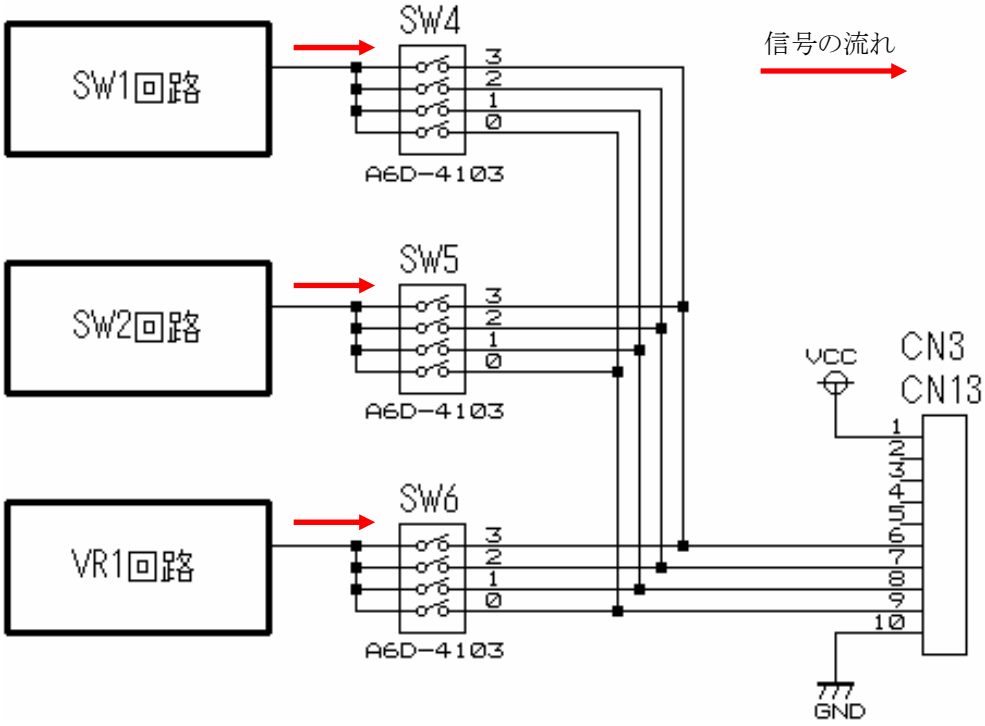
ブザーは、下図のように 2 枚の電極を貼り合わせたような形をしています。2 枚の電極は絶縁されていますが、既定値以上の電圧を加えると壊れてしまいます。外側は薄い鉄板になっていて、内側には特殊な電極が張り付いています。両方とも半田付けができるようになっています。

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。

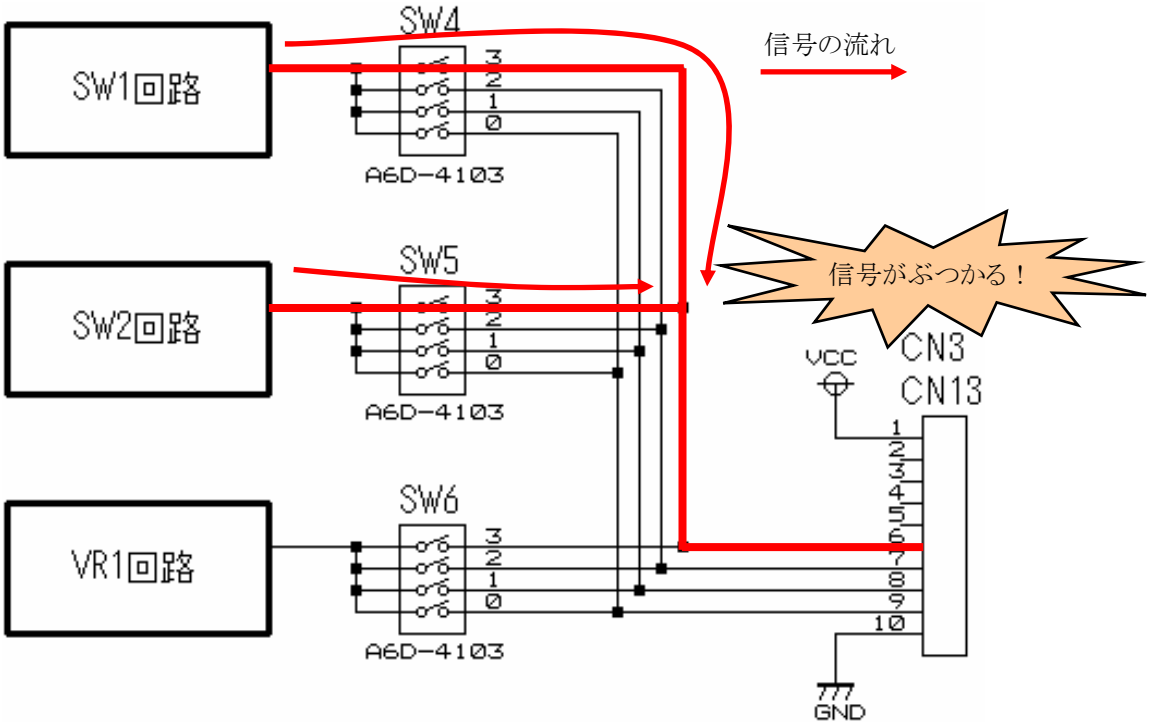


2.8 SW4,SW5,SW6 を切り替えるときの注意点

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。

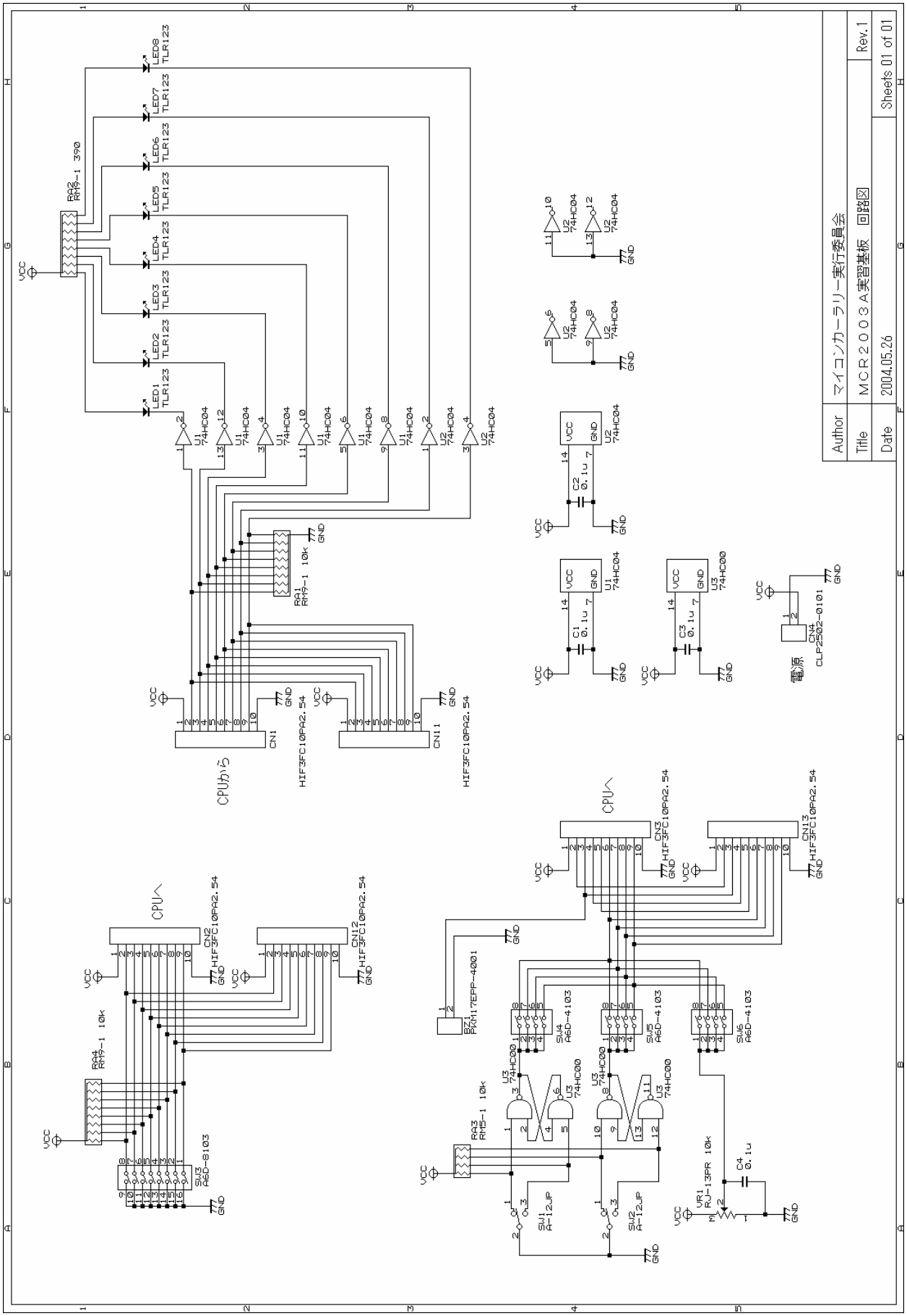


例えば、SW4 の 3 と SW5 の 3 を同時に ON にすると、下図のように信号同士がぶつかり、ショート状態になることがあります。



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

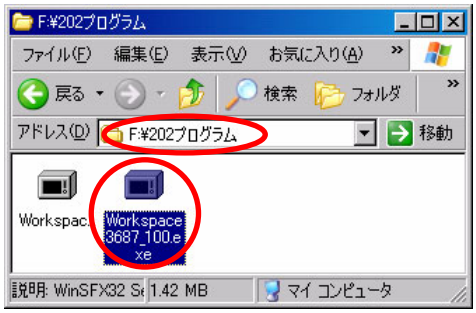
3.1 ルネサス統合開発環境

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。

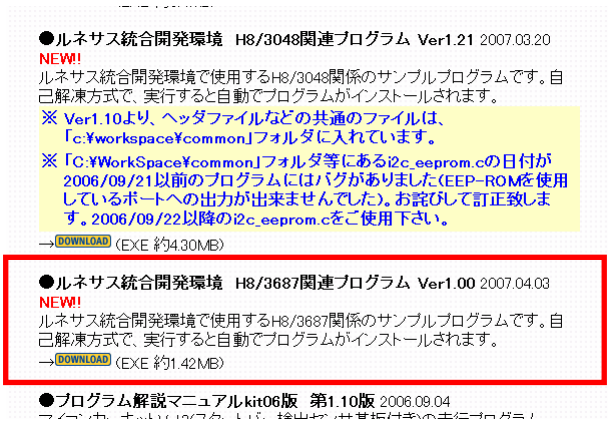
3.2.1 CDからソフトを取得する



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、「Workspace3687_100.exe」を実行します。数字の 100 は、バージョンにより異なります。

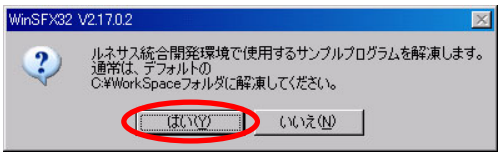
3.2.2 ホームページからソフトを取得する

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。

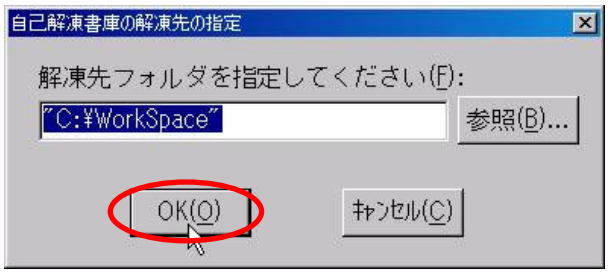


2.「ルネサス統合開発環境 **H8/3687** 関連プログラム」をダウンロードします。H8/3048 ではありませんので気をつけてください。

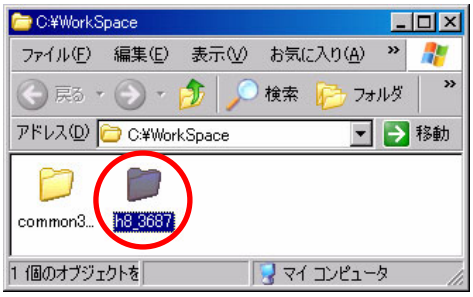
3.2.3 インストール



1.「**Workspace3687_100.exe**」を実行します。はいをクリックします。※100 はバージョンにより異なります。



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。**



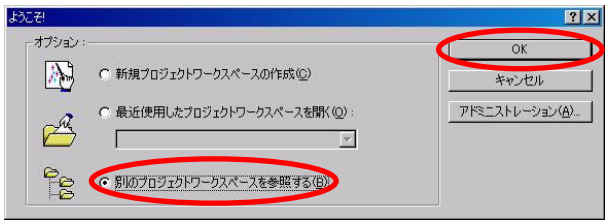
3. 解凍が終わったら、エクスプローラで「C ドライブ→Workspace」フォルダを開いてみてください。複数のフォルダがあります。今回使用するのは、「**h8_3687**」フォルダ内にあるワークスペースやプログラムです。

4. 演習手順

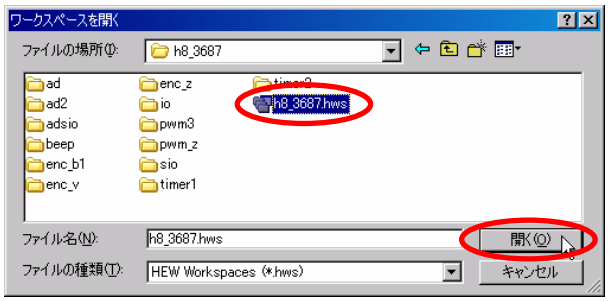
これから、実際にサンプルプログラムを使って、演習をしていきます。その前に、操作手順を説明していきます。詳しくは、「ルネサス統合開発環境　操作マニュアル」を参照してください。

4.1 ワークスペースを開く

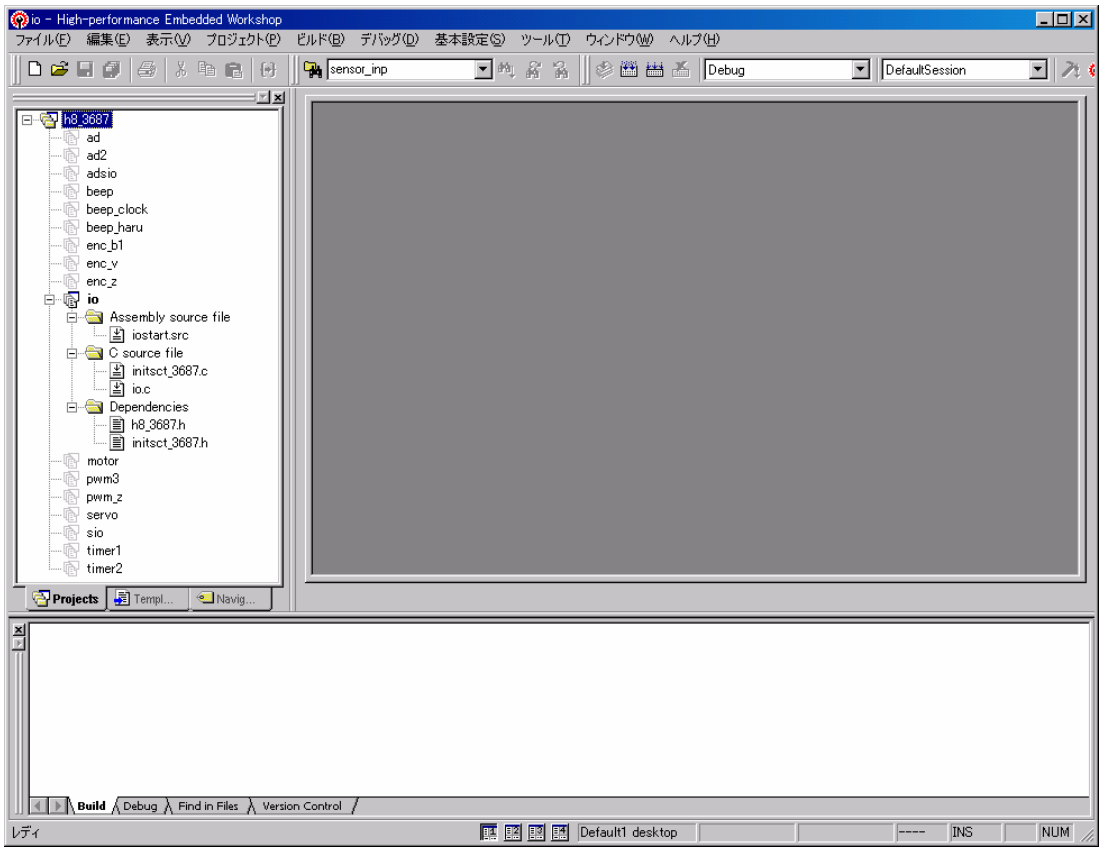
1.ルネサス統合開発環境を実行します。



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。



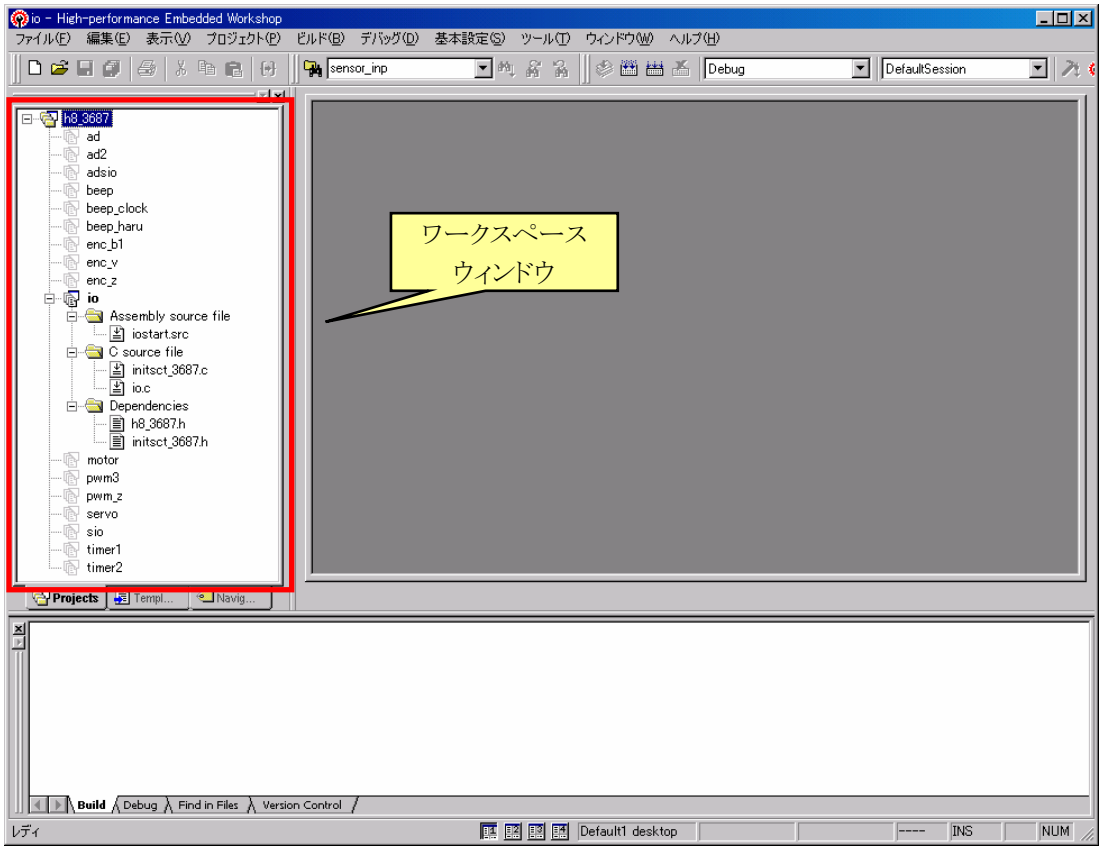
3.Cドライブ→workspace→h8_3687 の「h8_3687.hws」を選択、開くをクリックします。



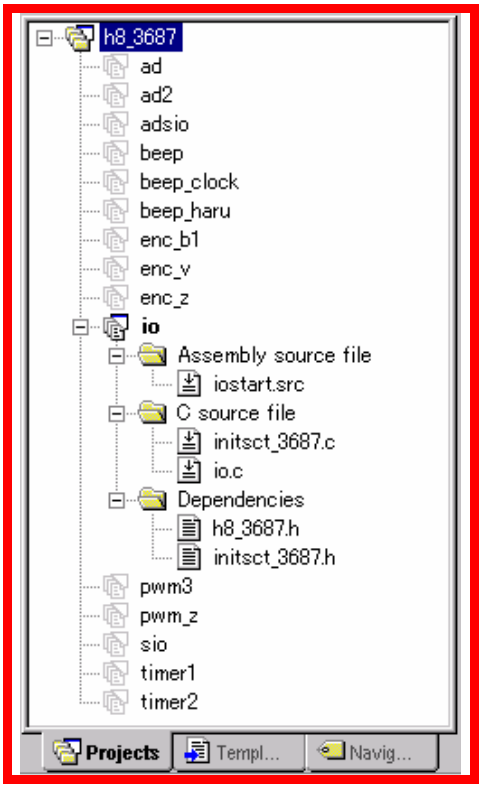
4.h8_3687 というワークスペースが開かれました。本実習マニュアルは、このワークスペースを使用します。

4.2 プロジェクトを開く

例えば、これから「ad」の演習をするとします。

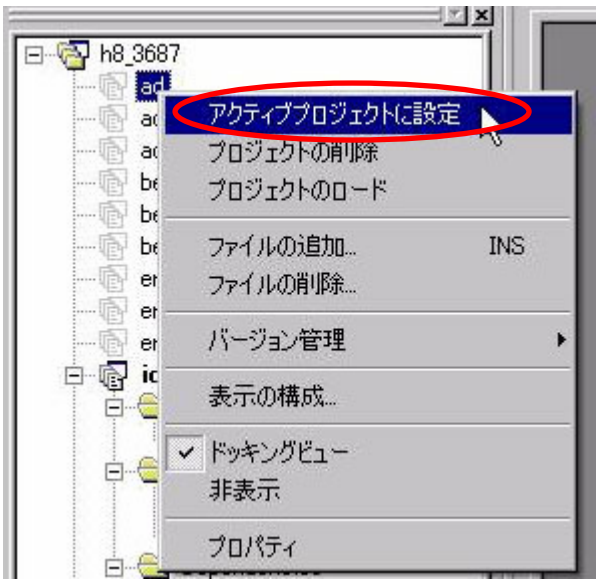


1.ワークスペースウィンドウと呼ばれるスペースにたくさんのプロジェクトが登録されています。

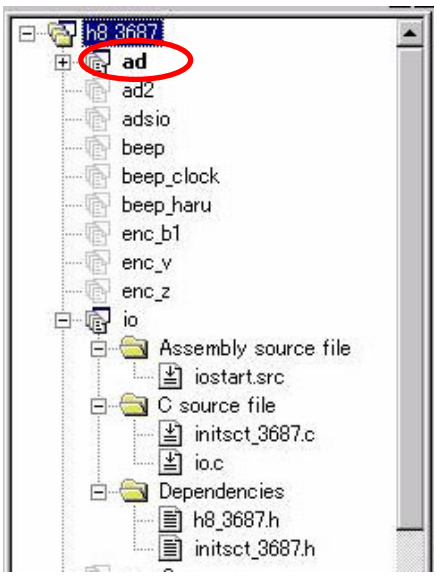


2.上から、
ad，ad2，adsio，beep，beep_clock，beep_haru，
enc_b1，enc_v，enc_z，io，pwm3，pwm_z，sio，timer1，timer2
というプロジェクトがあります。
プロジェクト＝演習の単位となります。

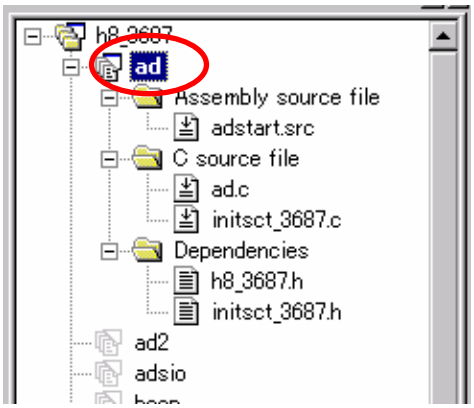
左画面では、プロジェクト「io」が太字になっています。このプロジェクトが現在有効なプロジェクト(アクティブプロジェクト)です。有効なプロジェクトは、
・ビルドの対象
・ツールチェイン(各種設定)の対象
・書き込みの対象
となります。「ad」の演習を行うので、「ad」を有効なプロジェクトにしなければいけません。



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



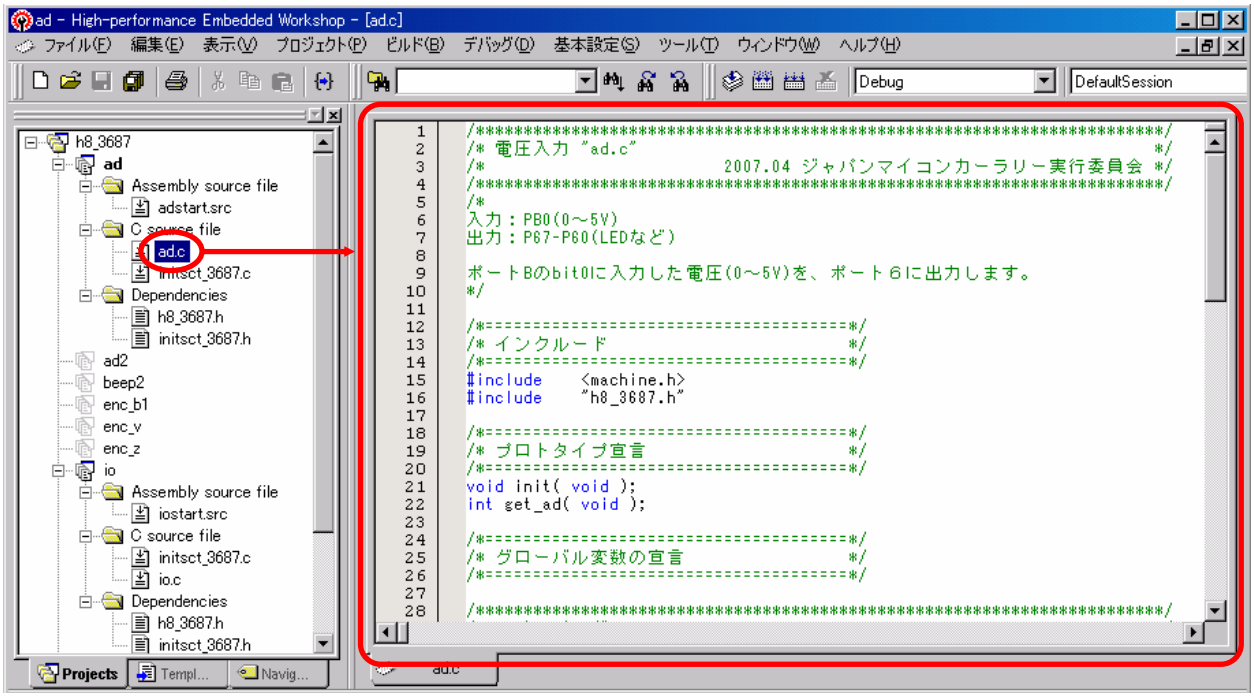
4.「ad」が太字になりました。これでアクティブプロジェクト設定完了です。



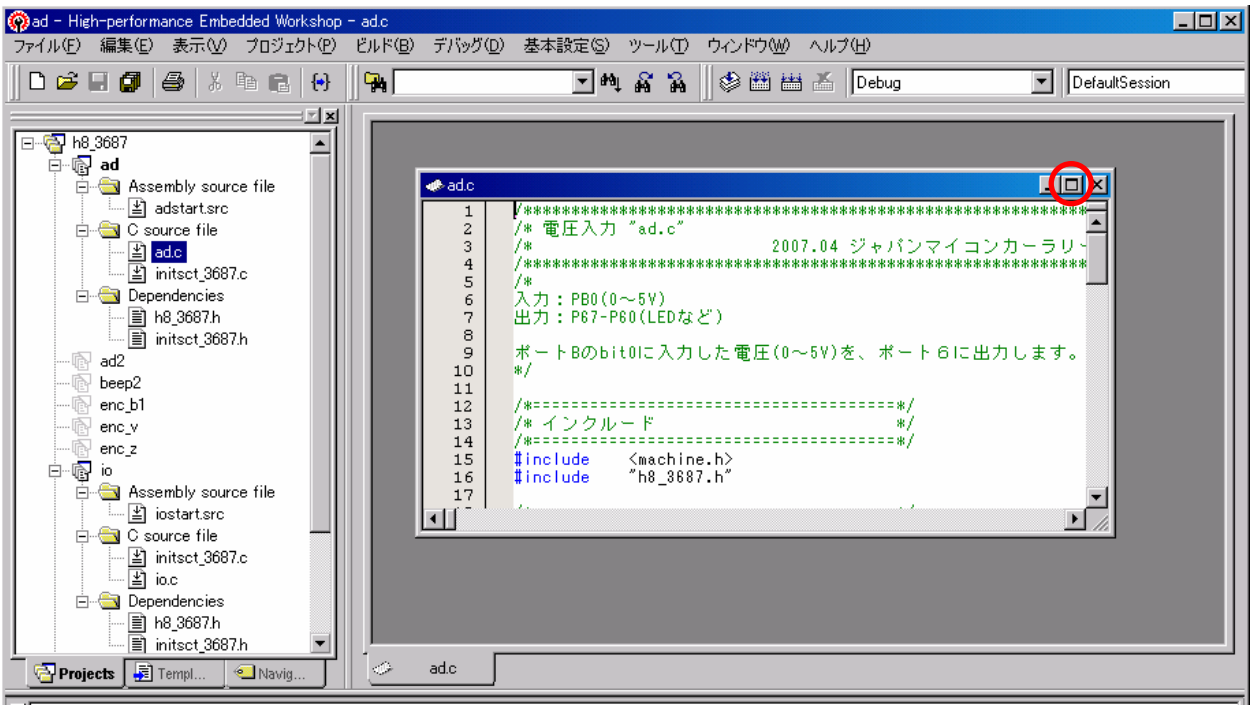
5.「ad」をダブルクリックします。
これがプロジェクト「ad」に登録されているファイルです。
●adstart.src
●ad.c
●initsct_3687.c
という 3 ファイルが登録されています。
Dependencies は、「従属する」という意味で、3 ファイル内でインクルードされているファイルがこの欄に表示されます。今回は、h8_3687.h ファイルと initsct_3687.h が表示されています。

4.3 ファイル編集

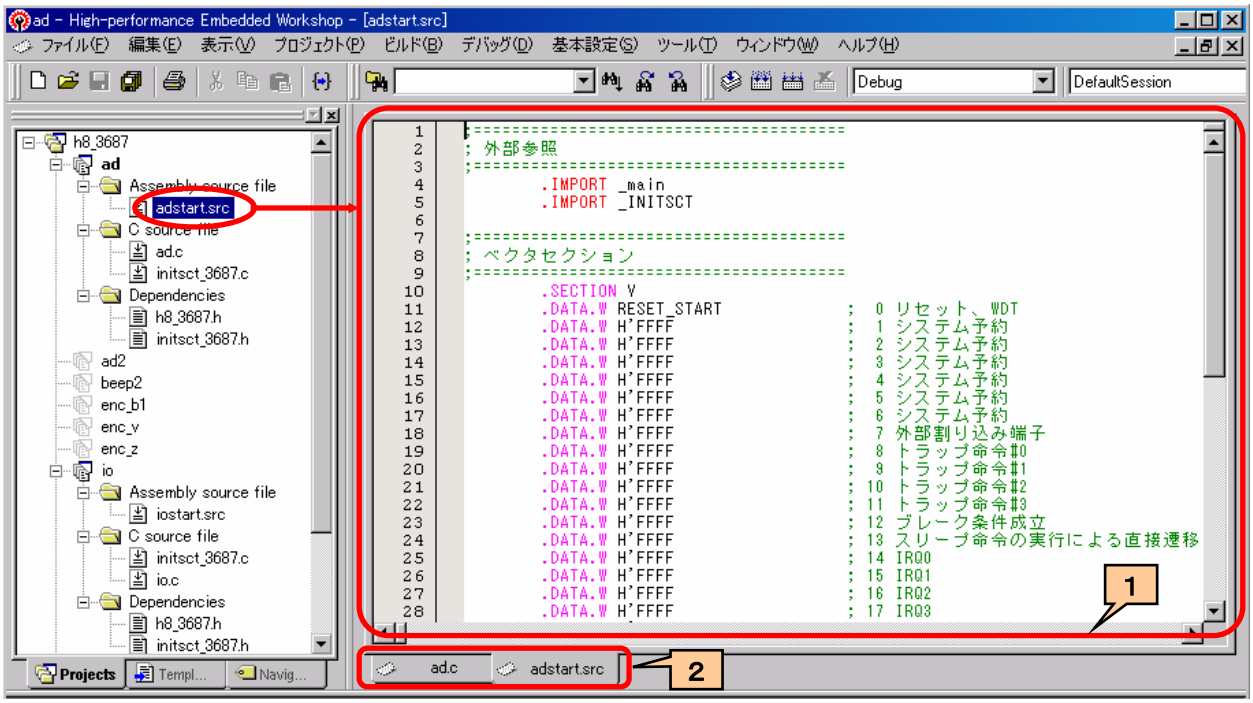
アクティブプロジェクトが「ad」を例に説明します。プロジェクト「ad」がアクティブでない場合、アクティブにしてください。



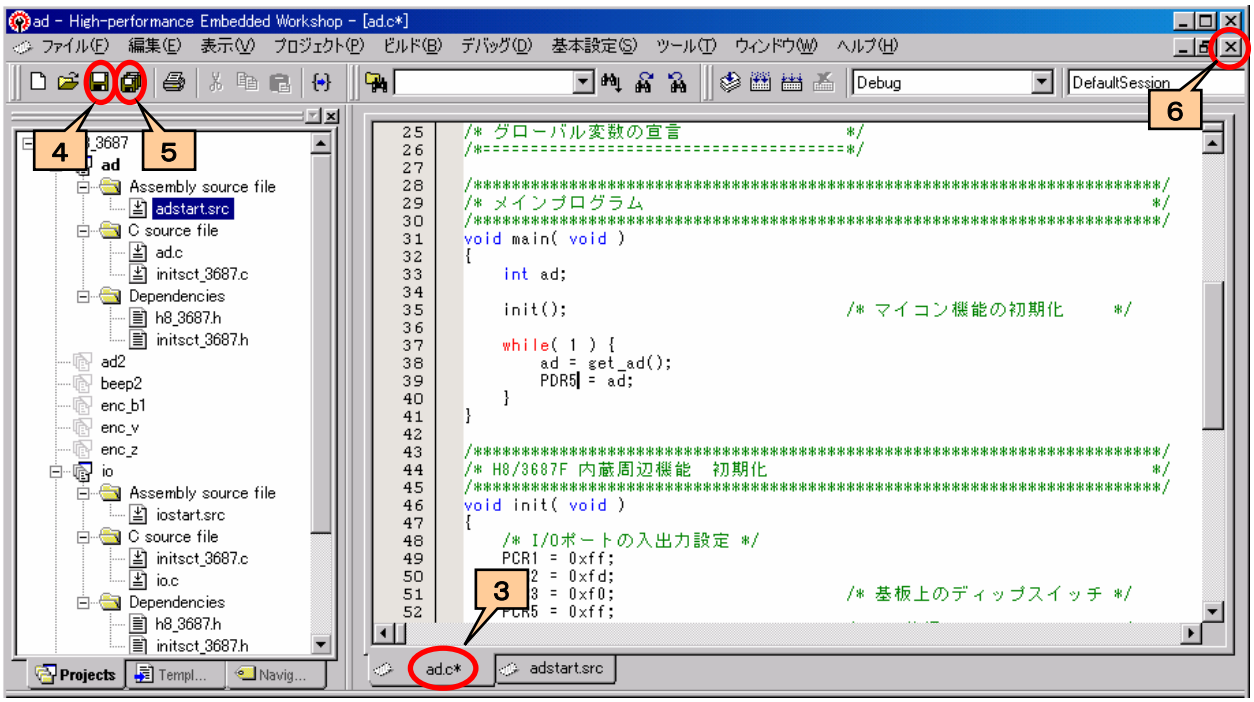
「ad.c」をダブルクリックすると、エディタウィンドウが開きます。ここでファイルを編集します。



上画面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。



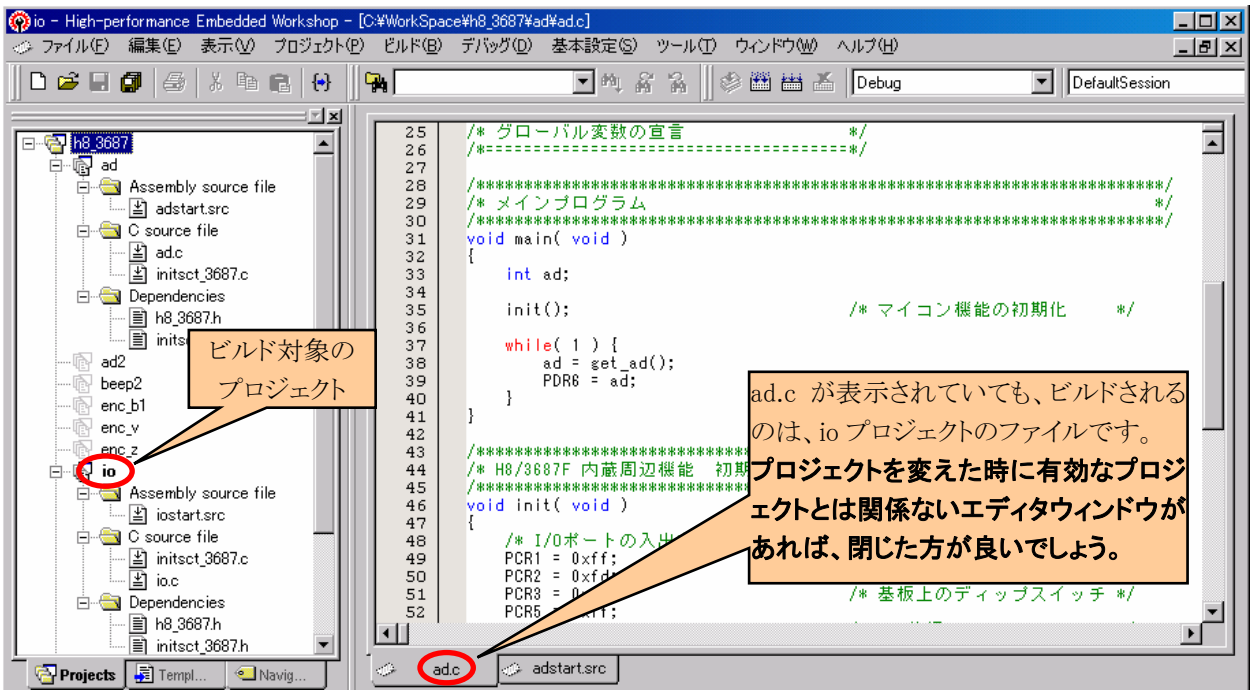
ファイルを編集して内容が変更されると、3部分に「＊」が表示されます。
4のアイコンは、現在編集中のファイルを保存するボタンです。
5のアイコンは、変更したすべてのファイルを保存するボタンです。

5のアイコンにすればすべて保存されますので、適宜5のアイコンで保存してください。

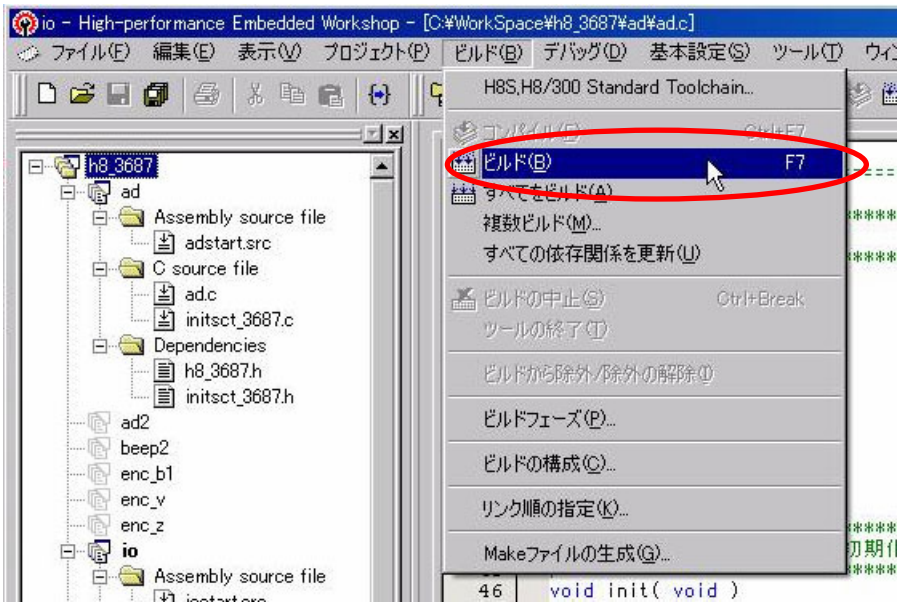
6の×ボタンで、編集中のファイルを閉じます。編集が終わって表示する必要のないファイルは、×ボタンで閉じてください。

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**。



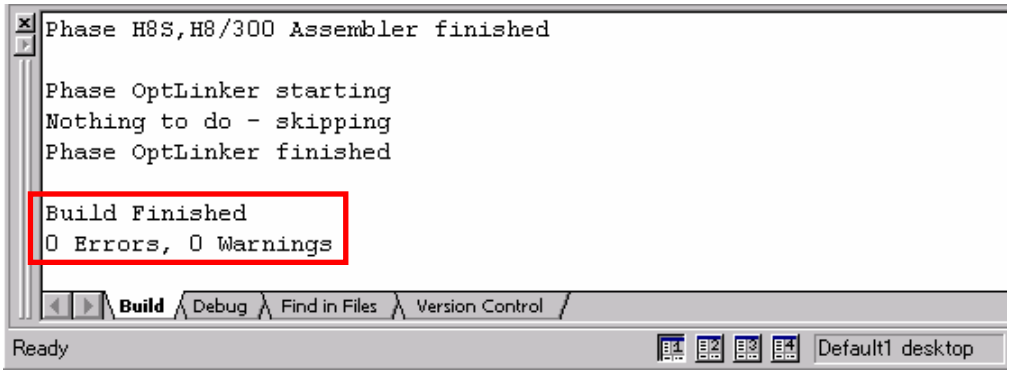
「ビルド」→「ビルド」でビルド作業を開始します。その下に「すべてをビルド」があります。違いは、

●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？*

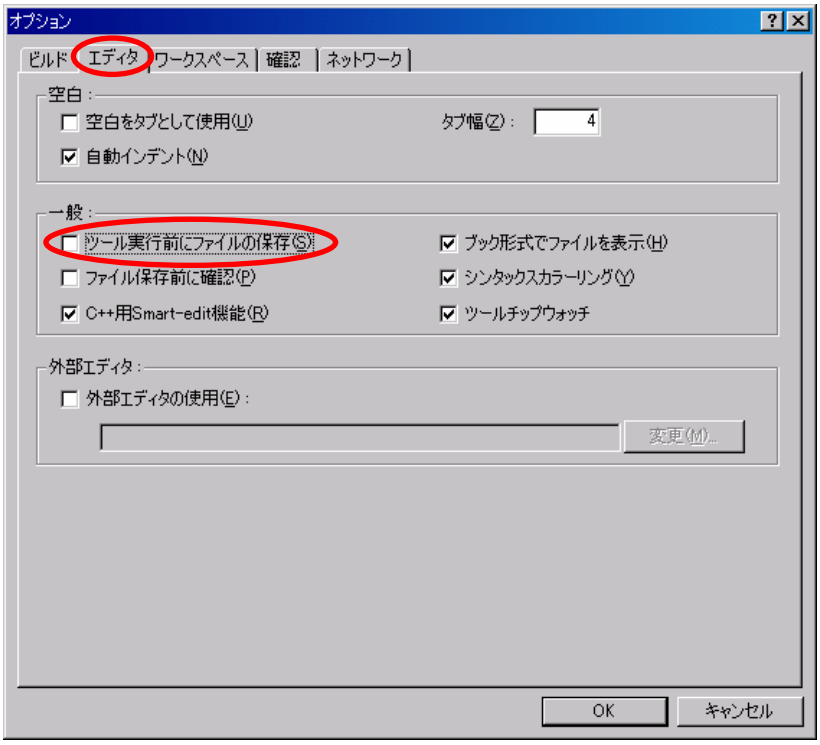
警告です。必ずしも誤っているとは言い切れないけども、間違っている可能性があるので確認してくださいというメッセージです。こちらも必ず直します。

Errors や Warnings が ″0″なら、プログラムに誤りはないということです。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、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。

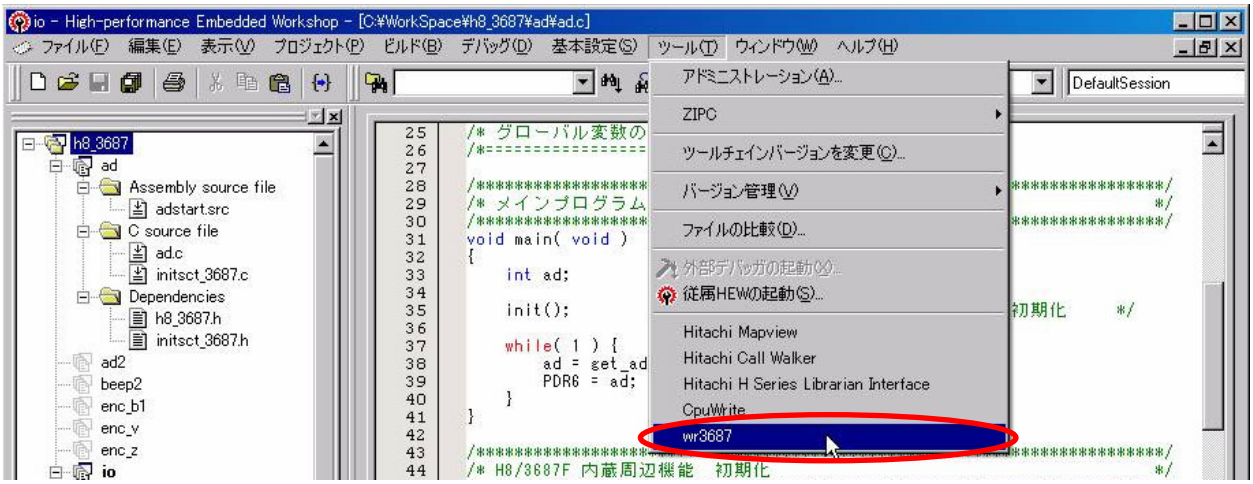
※ファイルの保存について

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。

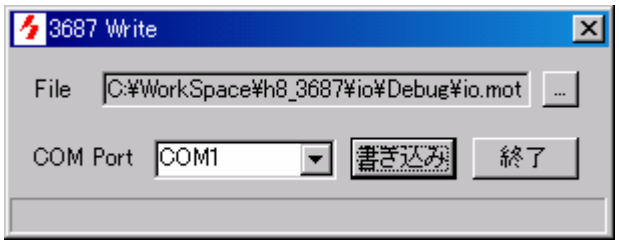


4.5 書き込みソフトの起動



ルネサス統合開発環境の「ツール→wr3687」をクリックします。

※CpuWrite ではありません。CpuWrite は RY3048Fone ボード用です。間違えないようにしてください。



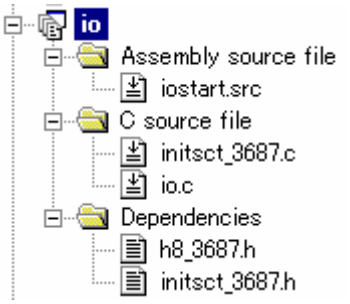
書き込みソフトが起動します。CPU ボードと書き込みソフトを操作してプログラムを書き込んで下さい。詳しくは、「ルネサス統合開発環境操作マニュアル 導入編」を参照してください。

5. プロジェクト内のファイルの関わりと実行順

5.1 概要

ここでは、ワークスペース「h8_3687」のプロジェクト「io」を例にして、下記について説明します。
・プロジェクト内にあるファイルがどう関わっているのか
・電源を入れてから、どの様な順番で実行されていくのか

5.2 プロジェクトのファイル構成



プロジェクト「io」は、下記のファイルから構成されています。

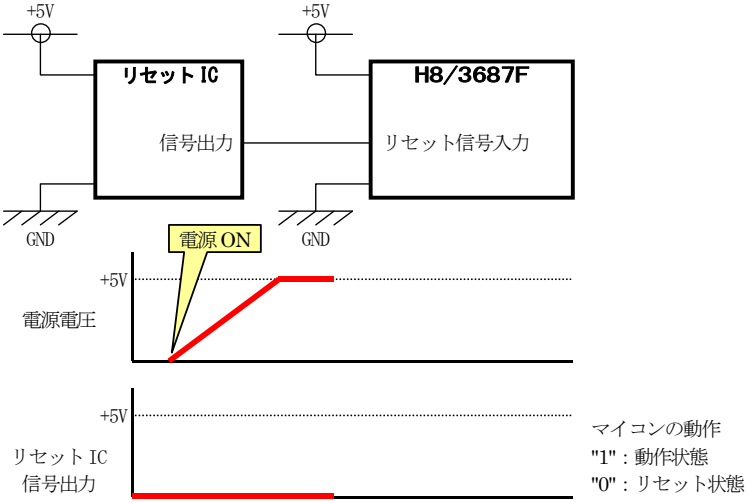
	ファイル名	内容
1	iostart.src	アセンブリ言語で記述されたアセンブリソースファイルです。このファイルの構造は下記のようになっています。 iostart.src ＝ ベクタアドレス ＋ スタートアップルーチン
2	initsct_3687.c	C 言語ソースファイルです。初期値のないグローバル変数（セクション B 領域）、初期値のあるグローバル変数（セクション R 領域）の初期化用です。
3	io.c	C 言語ソースファイルです。このファイルは、H8/3687F の内蔵周辺機能の初期化、メインで実行するプログラムが含まれています。
4	h8_3687.h	H8/3687F の内蔵周辺機能の I/O レジスタを定義したファイルです。
5	initsct_3687.h	initsct_3687.c のヘッダファイルです。

5.3 プログラムの実行順

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。

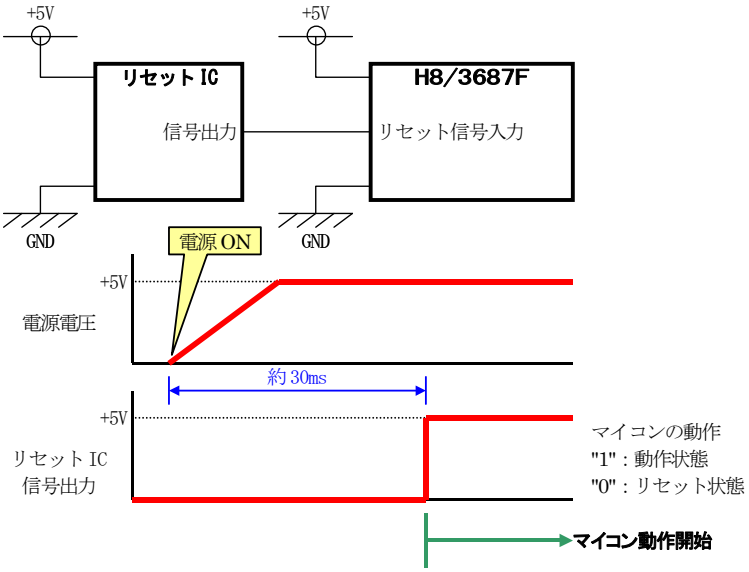
5.3.1 電源を入れたときの動作

マイコンボードの電源を入れます。電源を入れる瞬間は、電圧が安定しません。そのため、RY3687N ボードに取り付けているリセット 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。マイコンはリセット端子=″0″でリセット状態となるため、何もしません。



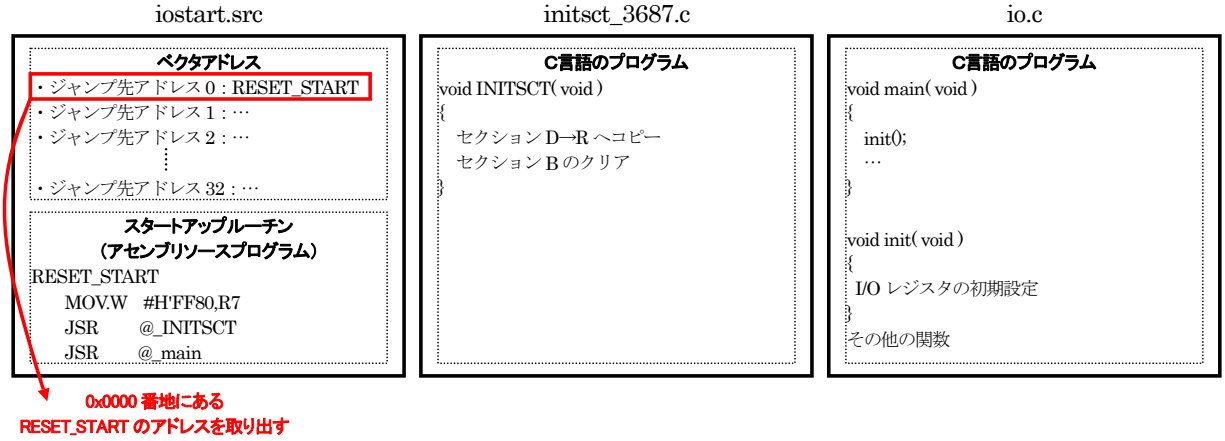
5.3.2 マイコンの動作開始

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。H8 マイコンのリセット信号入力端子が″1″になります。この瞬間から、マイコンは動き出します。ちなみに 30ms というのは、RY3687N ボードに取り付けているリセット IC の機能により、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。



5.3.3 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すると、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」が書いている部分（0 番）から値を取り出します。



ベクタアドレスは、0 番～32 番まであります。

0 番:　　リセット解除後のジャンプ先アドレス
1～6 番:　なし
7 番:　　NMI割り込み発生時のジャンプ先アドレス
8 番:　　トラップ命令#0 割り込み発生時のジャンプ先アドレス
…

とあらかじめ、「○○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。それらの割り込みを使うときは、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番からジャンプ先アドレスを読み込みます。**

ベクタアドレスは、ROM の 0x0000～0x0041 番地の範囲で、ベクタ番号 0 番は 0x0000 番地、ベクタ番号 1 番は 0x0002 番地・・・と決められています。ベクタ番号とベクタアドレス、割り込みの発生元の関係は、次の表のようになっています。

発生元	例外処理要因	ベクタ番号	ベクタアドレス	優先度
$\overline{\text{RES}}$ 端子 ウォッチドッグタイマ	リセット	0	H'0000～H'0001	高
－	システム予約	1～6	H'0002～H'000D	
外部割り込み端子	NMI	7	H'000E～H'000F	
CPU	トラップ命令 #0	8	H'0010～H'0011	
	トラップ命令 #1	9	H'0012～H'0013	
	トラップ命令 #2	10	H'0014～H'0015	
	トラップ命令 #3	11	H'0016～H'0017	
アドレスブレーク	ブレーク条件成立	12	H'0018～H'0019	
CPU	スリープ命令の実行による直接遷移	13	H'001A～H'001B	
外部割り込み端子	IRQ0 低電圧検出割り込み*	14	H'001C～H'001D	
	IRQ1	15	H'001E～H'001F	
	IRQ2	16	H'0020～H'0021	
	IRQ3	17	H'0022～H'0023	
	WKP	18	H'0024～H'0025	
RTC	オーバフロー	19	H'0026～H'0027	
－	システム予約	20	H'0028～H'0029	
タイマV	コンペアマッチA コンペアマッチB オーバフロー	22	H'002C～H'002D	
SCI3	受信データフル 送信データエンプティ 送信終了 受信エラー	23	H'002E～H'002F	
IIC2	送信データエンプティ、送信終了、受信データフル、 アービトレーションロスト／オーバランエラー NACK 検出、停止条件検出	24	H'0030～H'0031	
A/D 変換器	A/D 変換終了	25	H'0032～H'0033	
タイマ Z	コンペアマッチ／インプットキャプチャ A0～D0 オーバフロー	26	H'0034～H'0035	
	コンペアマッチ／インプットキャプチャ A1～D1 オーバフロー、アンダフロー	27	H'0036～H'0037	
タイマ B1	オーバフロー	29	H'003A～H'003B	
SCI3_2	受信データフル 送信データエンプティ 送信終了 受信エラー	32	H'0040～H'0041	低

例えば、リセット後、「RESET_START」ラベルのある場所からプログラムを開始したいとします。その場合、iostart.src プログラムに、下記のようにベクタアドレスを記述します。

```
 10 :          .SECTION V
 11 :          .DATA.W RESET_START          ;  0 リセット、WDT          ←0番のジャンプ先アドレス
 12 :          .DATA.W H'FFFF               ;  1 システム予約           ←1番のジャンプ先アドレス
 13 :          .DATA.W H'FFFF               ;  2 システム予約           ←2番のジャンプ先アドレス

中略

 42 :          .DATA.W H'FFFF               ; 31                        ←31番のジャンプ先アドレス
 43 :          .DATA.W H'FFFF               ; 32 SCI3_2                 ←32番のジャンプ先アドレス
```

「.DATA.W」はワードサイズ(2 バイト)でデータを作りなさいという命令です。11 行目は、「RESET_START」ラベルがある番地を数値にします。例えば、「RESET_START」のラベルがある場所が 0x100 番地なら、下記と同じことです。

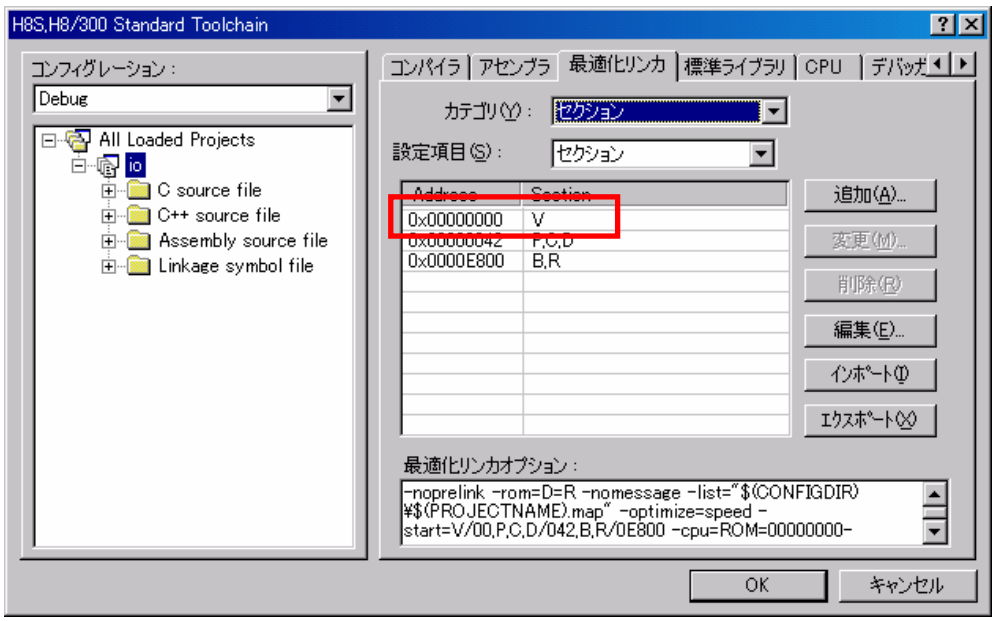
```
 11 :          .DATA.W H'0100               ;  0 リセット、WDT
```

登録する必要のない番号には 0xffff を入れておきます。

```
 12 :          .DATA.W H'FFFF               ;  1 システム予約
 13 :          .DATA.W H'FFFF               ;  2 システム予約
・・・・・
 42 :          .DATA.W H'FFFF               ; 31
 43 :          .DATA.W H'FFFF               ; 32 SCI3_2
```

ちなみに、10 行目の記述は「ここからはセクション V という領域ですよ」とルネサス統合開発環境(リンカ)に知らせている命令です。ルネサス統合開発環境はツールチェインの設定によって、セクション V を何番地にするか決めます。

下記が実際のツールチェインの設定です。ルネサス統合開発環境の「ビルド→H8S,H8/300 Standard Toolchain」を選択、「最適化リンカ、カテゴリ:セクション、設定項目:セクション」を選択すると確認できます。

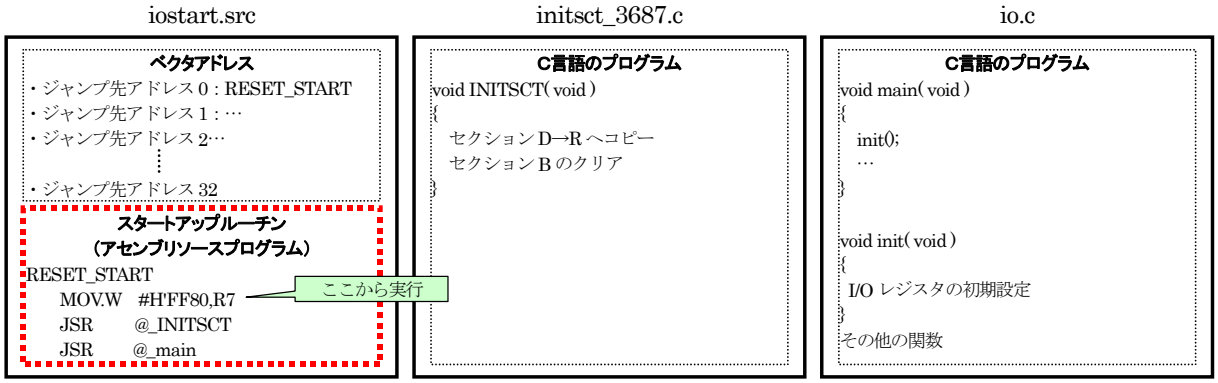


ここで、「セクション V は、0x0000 番地にしなさい」と設定されているので、ベクタアドレスであるセクション V 部分は 0x0000 番地から配置されるのです。「ベクタアドレスは 0x0000 番地から開始する」というのは、決まり事な

ので変更することはできません。ツールチェインやセクションについての詳しい説明は、「ルネサス統合開発環境操作マニュアル　応用編」を参照してください。

5.3.4 スタートアップルーチンの実行

マイコンは、ベクタ番号 0 番に書かれている番地、すなわち「RESET_START」部分から実行します。ここには、初期設定を行うプログラムを用意しておきます。実際のプログラムは下記のようになっています。

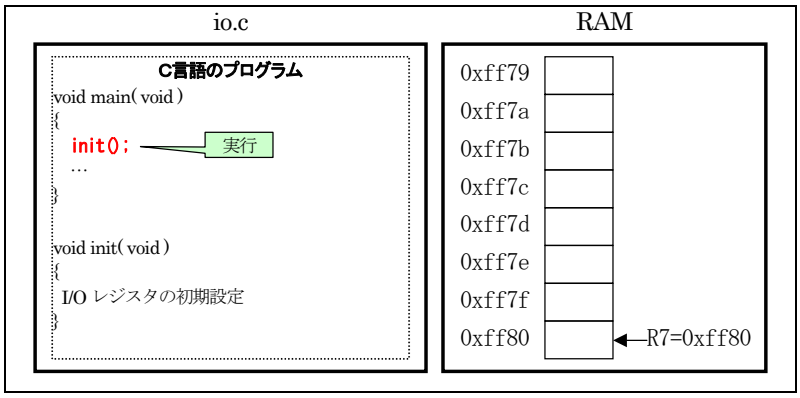


5.3.5 スタックポインタの設定

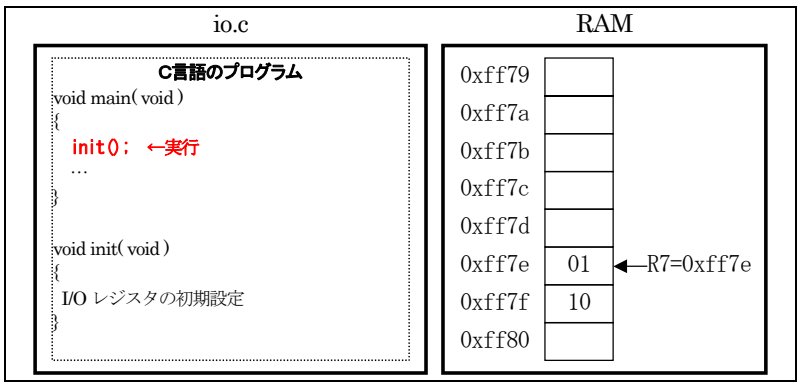
まず、スタックポインタの設定を行います。

```
RESET_START:
        MOV.W   #H'FF80,R7          ; スタックポインタの設定
        JSR     @_INITSCT           ; セクション D,R,B の設定
        JSR     @_main              ; C言語の main()関数へジャンプ
OWARI:
        BRA     OWARI
```

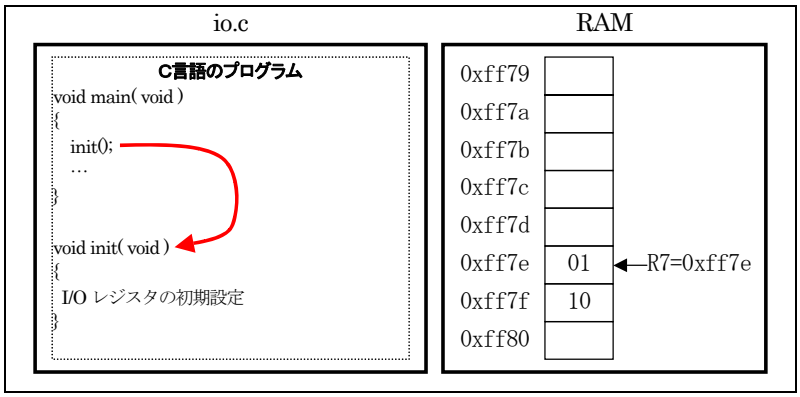
スタックポインタとは、番地やデータを一時的に待避させるアドレスのことです。
例えば、init 関数を呼んだとします（下図）。



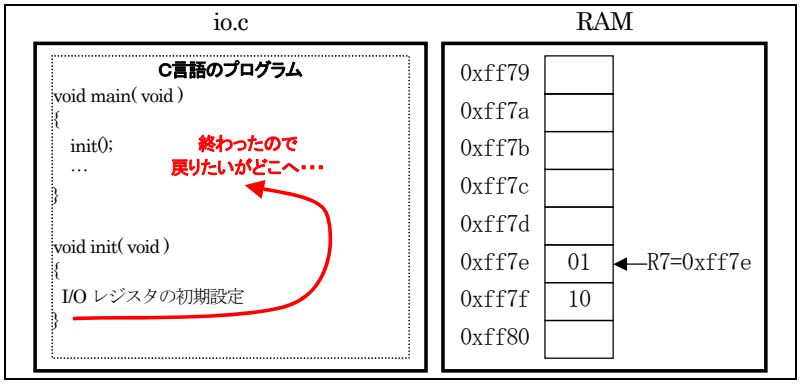
マイコンは、スタックポインタ(R7)の値を-2 します。その番地である 0xff7e 番地に、今実行しているプログラムの次のプログラムがある番地を書き込みます。例えば、その番地が 0x0110 番地なら、0xff7e 番地には 0x0110 と保存されます(下図)。



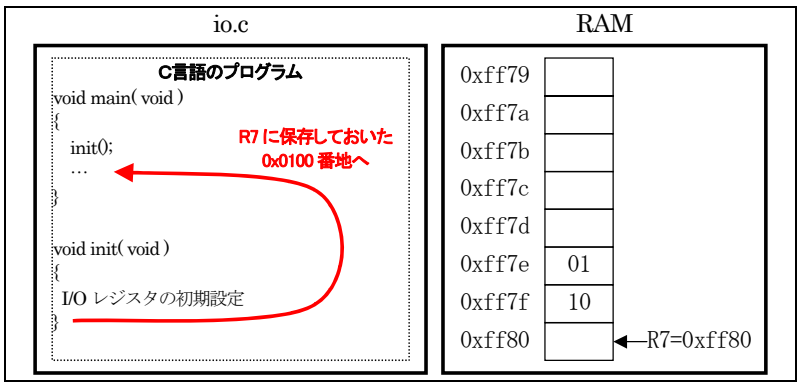
保存後、init 関数へジャンプして、init 関数を実行します(下図)。



実行が終わったら、呼ばれた部分へ戻ります。しかし、それは何処だったでしょうか？(下図)



ここで、スタックポインタの意味があるのです。init 関数を呼んだとき、戻り先をスタックポインタで示している番地に保存しました。スタックポインタ(R7)が示している番地のデータを読み込み、その番地を実行すればよいのです。スタックポインタ(R7)の値は+2 しておきます(下図)。

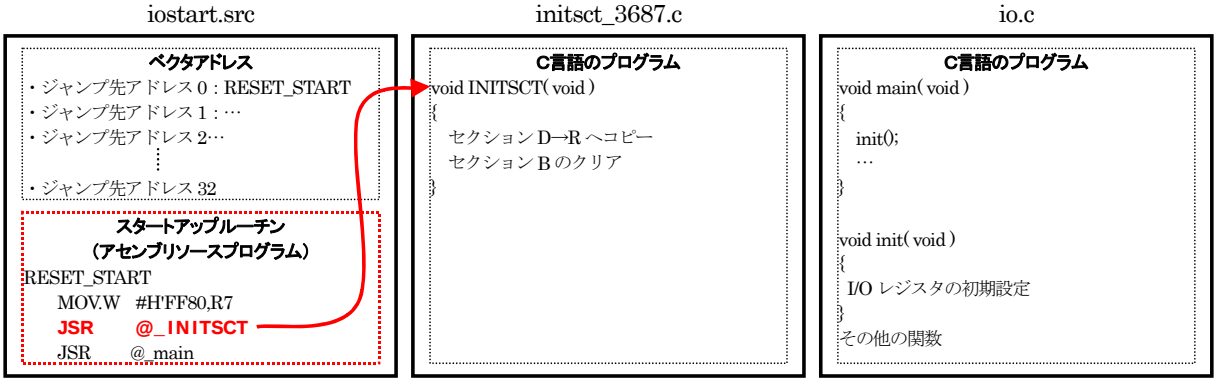


このように、スタックポインタは戻る番地や値を保存しておきます。**スタックポインタを設定しないと、戻り先が分からなくなるので、プログラムが暴走します。そのため、プログラムを実行するとき、一番最初にスタックポインタを設定しなければ行けないのです**。

それ以外にも、割り込み発生時の戻り先や CCR を保存したりします。詳しくは「スタックポインタ」という単語でインターネットを検索するか、制御の教科書を参照してください。

5.3.6 INITSCT関数の実行

次に「JSR @_INITSCT」を実行します。JSR とは、「ジャンプサブルーチン」の意味です。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。



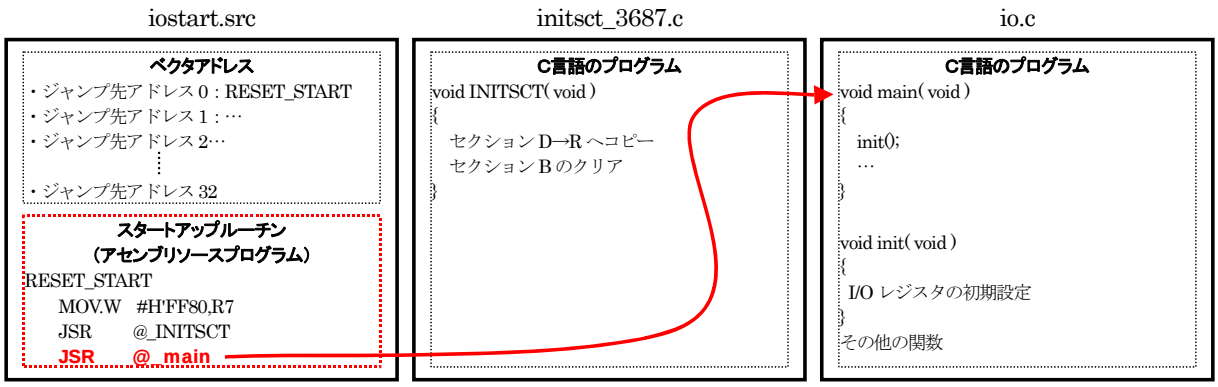
INITSCT 関数を実行します。この関数は何をしているのでしょうか。下記のような役割です。

・初期値のあるグローバル変数(セクション R 領域)の値を設定
・初期値のないグローバル変数(セクション B 領域)をクリア

詳しくは、「ルネサス統合開発環境操作マニュアル 応用編」のセクションを参照してください。

5.3.7 main関数の実行

初期値のあるグローバル変数の設定、初期値のないグローバル変数のクリアが終わったら、main 関数を実行します。main 関数内には、読んで字の如く、メインのプログラムを入れておきます。



5.3.8 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。

```
52 :            JSR     @_main                  ; C言語の main()関数へジャンプ
```

をアセンブルするとき、「_main」を探します。しかし、「_main」は iostart.src 内にはありません。「_main」は、io.c ファイル内にあります。そのため、「_main」は他の場所にあるので、他を探してください、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。

```
4 :             .IMPORT _main
```

という記述で、アセンブラは「_main」が他のファイルにあることを理解して「_main」というラベル名があれば、予約だけしておきます。その場合、リンケージエディタがリンク時に実際のアドレスを入れます。

同様に、「_INITSCT」も IMPORT 宣言しておきます。

```
5 :             .IMPORT _INITSCT
```

6. 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)

6.1 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 5 の全ビット・・・ディップスイッチの状態を入力

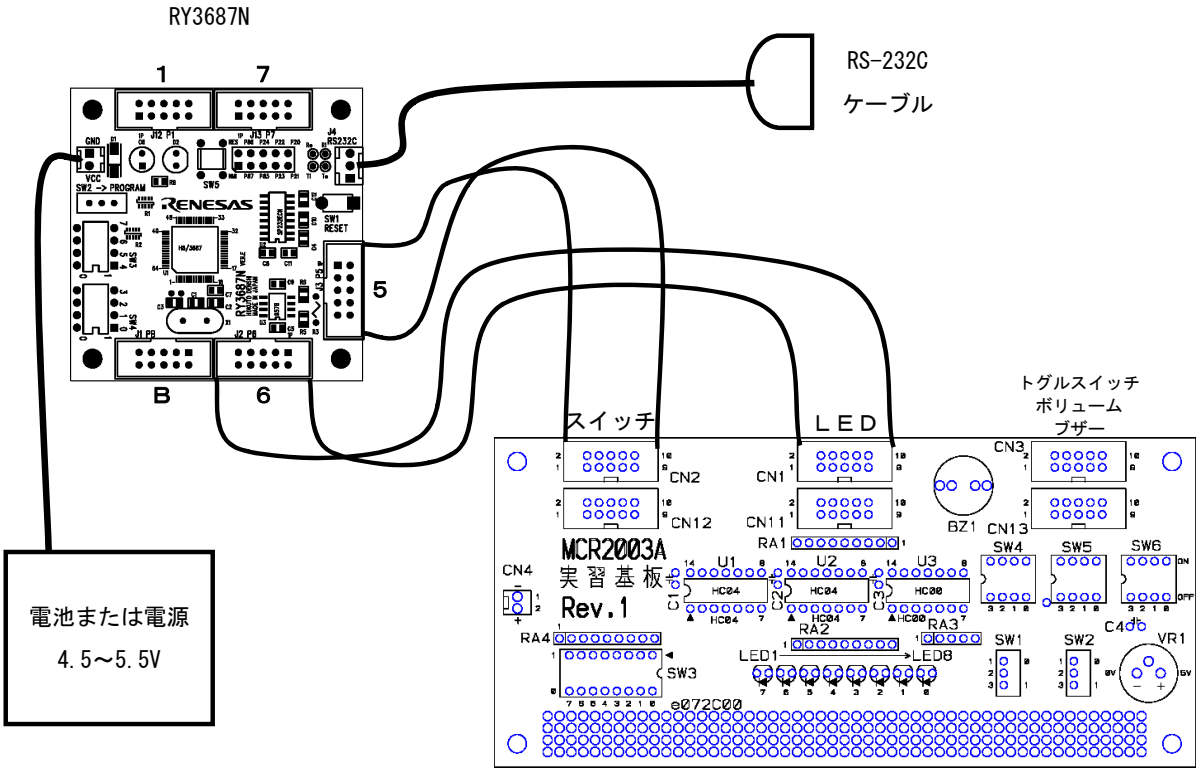
・ポート 6 の全ビット・・・LED へデータ出力

6.2 接続

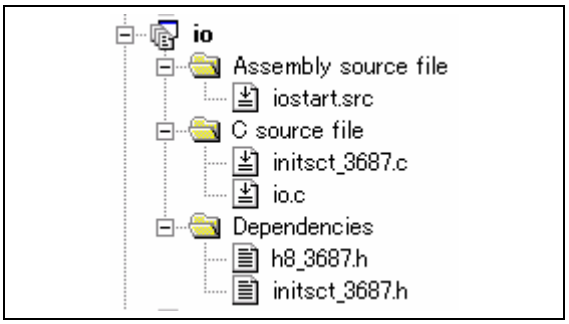
・CPU ボードのポート 5 と、実習基板のスイッチ部をフラットケーブルで接続します。

・CPU ボードのポート 6 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 5 のディップスイッチをマイコンカーの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。



6.3 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_3687.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。H8/3687F の内蔵周辺機能の初期化も行います。
4	h8_3687.h	H8/3687F 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_3687.h	initsct_3687.c のヘッダファイルです。

6.4 プログラム「io.c」

```
 1 :  /************************************************************************/
 2 :  /* ポート５の状態をポート６に出力(センサ基板のチェック)"io.c"           */
 3 :  /*                           2007.04 ジャパンマイコンカーラリー実行委員会 */
 4 :  /************************************************************************/
 5 :  /*
 6 :  入力：P57-P50(ディップスイッチなど)
 7 :  出力：P67-P60(LEDなど)
 8 :
 9 :  ポート５に入力した状態を、ポート６に出力します。
10 :  ポート５にセンサ基板、ポート６に実習基板のLED部分を接続すれば
11 :  センサ基板のチェックになります。
12 :  */
13 :
14 :  /*====================================*/
15 :  /* インクルード                       */
16 :  /*====================================*/
17 :  #include    <machine.h>
18 :  #include    "h8_3687.h"
19 :
20 :  /*====================================*/
21 :  /* プロトタイプ宣言                   */
22 :  /*====================================*/
23 :  void init( void );
24 :
25 :  /*====================================*/
26 :  /* グローバル変数の宣言               */
27 :  /*====================================*/
28 :
29 :  /************************************************************************/
30 :  /* メインプログラム                                                     */
31 :  /************************************************************************/
32 :  void main( void )
33 :  {
34 :      unsigned char d;
35 :
36 :      init();                         /* マイコン機能の初期化      */
37 :
```

```
38 :      while( 1 ) {
39 :          d = PDR5;
40 :          PDR6 = d;
41 :      }
42 :  }
43 :
44 :  /************************************************************************/
45 :  /* H8/3687F 内蔵周辺機能  初期化                                        */
46 :  /************************************************************************/
47 :  void init( void )
48 :  {
49 :      /* I/Oポートの入出力設定 */
50 :      PCR1 = 0xff;
51 :      PCR2 = 0xfd;                        /* 通信ビットP22:TxD P21:RxD*/
52 :      PCR3 = 0xf0;                        /* 基板上のディップスイッチ */
53 :      PCR5 = 0x00;                        /* スイッチ基板             */
54 :      PCR6 = 0xff;                        /* LED基板                  */
55 :      PCR7 = 0xff;
56 :      PCR8 = 0xff;
57 :      /* ポートBは、入力専用なので入出力設定はありません。          */
58 :      /* また、A/D変換器のADCSRによって指定された1ビットだけは、   */
59 :      /* 入力ポートとしては使えません。                           */
60 :  }
61 :
62 :  /************************************************************************/
63 :  /* End of file                                                          */
64 :  /************************************************************************/
```

6.5 プログラム「iostart.src」

```
 1 :  ;======================================
 2 :  ; 外部参照
 3 :  ;======================================
 4 :          .IMPORT _main
 5 :          .IMPORT _INITSCT
 6 :
 7 :  ;======================================
 8 :  ; ベクタセクション
 9 :  ;======================================
10 :          .SECTION V
11 :          .DATA.W RESET_START             ;  0 リセット、WDT
12 :          .DATA.W H'FFFF                  ;  1 システム予約
13 :          .DATA.W H'FFFF                  ;  2 システム予約
14 :          .DATA.W H'FFFF                  ;  3 システム予約
15 :          .DATA.W H'FFFF                  ;  4 システム予約
16 :          .DATA.W H'FFFF                  ;  5 システム予約
17 :          .DATA.W H'FFFF                  ;  6 システム予約
18 :          .DATA.W H'FFFF                  ;  7 外部割り込み端子
19 :          .DATA.W H'FFFF                  ;  8 トラップ命令#0
20 :          .DATA.W H'FFFF                  ;  9 トラップ命令#1
21 :          .DATA.W H'FFFF                  ; 10 トラップ命令#2
22 :          .DATA.W H'FFFF                  ; 11 トラップ命令#3
23 :          .DATA.W H'FFFF                  ; 12 ブレーク条件成立
24 :          .DATA.W H'FFFF                  ; 13 スリープ命令の実行による直接遷移
25 :          .DATA.W H'FFFF                  ; 14 IRQ0
26 :          .DATA.W H'FFFF                  ; 15 IRQ1
27 :          .DATA.W H'FFFF                  ; 16 IRQ2
28 :          .DATA.W H'FFFF                  ; 17 IRQ3
29 :          .DATA.W H'FFFF                  ; 18 WKP
30 :          .DATA.W H'FFFF                  ; 19 オーバフロー
31 :          .DATA.W H'FFFF                  ; 20 システム予約
32 :          .DATA.W H'FFFF                  ; 21
33 :          .DATA.W H'FFFF                  ; 22 タイマV
34 :          .DATA.W H'FFFF                  ; 23 SCI3
35 :          .DATA.W H'FFFF                  ; 24 IIC2
36 :          .DATA.W H'FFFF                  ; 25 AD変換器
37 :          .DATA.W H'FFFF                  ; 26 タイマZ A0-D0
38 :          .DATA.W H'FFFF                  ; 27 タイマZ A1-D1
39 :          .DATA.W H'FFFF                  ; 28
40 :          .DATA.W H'FFFF                  ; 29 タイマB1
41 :          .DATA.W H'FFFF                  ; 30
42 :          .DATA.W H'FFFF                  ; 31
43 :          .DATA.W H'FFFF                  ; 32 SCI3_2
44 :
45 :  ;======================================
46 :  ; スタートアッププログラム
47 :  ;======================================
48 :          .SECTION P
49 :  RESET_START:
50 :          MOV.W   #H'FF80,R7              ; スタックポインタの設定
51 :          JSR     @_INITSCT               ; セクションD,R,Bの設定
52 :          JSR     @_main                  ; C言語のmain()関数へジャンプ
53 :  LOOP:
54 :          BRA     LOOP
55 :
56 :          .END
```

6.6 プログラム「io.c」の解説

6.6.1 「machine.h」ファイルの取り込み

```
17 :  #include    〈machine.h〉
```

「#include」は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マイコンに特化した機能を提供する組み込み関数です。

6.6.2 「h8_3687.h」ファイルの取り込み

```
18 :  #include    "h8_3687.h"
```

次に「h8_3687.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。下記のような違いがあります。

＜　＞	通常の include フォルダから取り込むファイルを探します。 標準ライブラリとしてルネサス統合開発環境が用意しているファイルは、こちらを使います。 例えば、stdio.h , stdlib.h , math.h などです。ちなみに、これらのファイルは、 C:¥Program Files¥Renesas¥Hew¥Tools¥Renesas¥H8¥6_1_2¥include フォルダにあります。
“　”	まず、カレント・フォルダ（Cプログラムファイルがあるフォルダ）を探して、そこに無ければ通常の include フォルダから取り込むファイルを探します。自分で作ったファイルは、こちらを使います。

「h8_3687.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。

6.6.3 「h8_3687.h」ファイルの内容

なぜ、「h8_3687.h」という別ファイルにするのでしょうか。中身は、H8/3687F マイコンの内蔵周辺機能の 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。これでは非効率なので、**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_3687.h」は、下記のような内容です。

```
 1 :  /************************************************************************/
 2 :  /* H8/3687用内蔵周辺機能のI/Oレジスタ定義 Ver1.03                       */
 3 :  /*    2007.04 ジャパンマイコンカーラリー実行委員会                      */
 4 :  /************************************************************************/
 5 :
 6 :  #define TCR_0    (*(volatile unsigned char*)0xf700)  /*ﾀｲﾏｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ_0*/
 7 :  #define TIORA_0  (*(volatile unsigned char*)0xf701)  /*ﾀｲﾏI/OｺﾝﾄﾛｰﾙﾚｼﾞｽﾀA_0*/
 8 :  #define TIORC_0  (*(volatile unsigned char*)0xf702)  /*ﾀｲﾏI/OｺﾝﾄﾛｰﾙﾚｼﾞｽﾀC_0*/
 9 :  #define TSR_0    (*(volatile unsigned char*)0xf703)  /*ﾀｲﾏｽﾃｰﾀｽﾚｼﾞｽﾀ_0*/
10 :  #define TIER_0   (*(volatile unsigned char*)0xf704)  /*ﾀｲﾏｲﾝﾀﾗﾌﾟﾄｲﾈｰﾌﾞﾙﾚｼﾞｽﾀ_0*/
11 :  #define POCR_0   (*(volatile unsigned char*)0xf705)  /*PWMﾓｰﾄﾞｱｳﾄﾌﾟｯﾄﾚﾍﾞﾙｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ_0*/
12 :  #define TCNT_0   (*(volatile unsigned int*)0xf706)   /*ﾀｲﾏｶｳﾝﾀ_0*/
13 :  #define GRA_0    (*(volatile unsigned int*)0xf708)   /*ｼﾞｪﾈﾗﾙﾚｼﾞｽﾀA_0*/
14 :  #define GRB_0    (*(volatile unsigned int*)0xf70a)   /*ｼﾞｪﾈﾗﾙﾚｼﾞｽﾀB_0*/
15 :  #define GRC_0    (*(volatile unsigned int*)0xf70c)   /*ｼﾞｪﾈﾗﾙﾚｼﾞｽﾀC_0*/
16 :  #define GRD_0    (*(volatile unsigned int*)0xf70e)   /*ｼﾞｪﾈﾗﾙﾚｼﾞｽﾀD_0*/
17 :  #define TCR_1    (*(volatile unsigned char*)0xf710)  /*ﾀｲﾏｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ_1*/
18 :  #define TIORA_1  (*(volatile unsigned char*)0xf711)  /*ﾀｲﾏI/OｺﾝﾄﾛｰﾙﾚｼﾞｽﾀA_1*/
```

```
 19 :  #define TIORC_1 (*(volatile unsigned char*)0xf712)  /*ﾀｲﾏI/OｺﾝﾄﾛｰﾙﾚｼﾞｽﾀC_1*/
 20 :  #define TSR_1   (*(volatile unsigned char*)0xf713)  /*ﾀｲﾏｽﾃｰﾀｽﾚｼﾞｽﾀ_1*/
 21 :  #define TIER_1  (*(volatile unsigned char*)0xf714)  /*ﾀｲﾏｲﾝﾀﾗﾌﾟﾄｲﾈｰﾌﾞﾙﾚｼﾞｽﾀ_1*/
 22 :  #define POCR_1  (*(volatile unsigned char*)0xf715)  /*PWMﾓｰﾄﾞｱｳﾄﾌﾟｯﾄﾚﾍﾞﾙｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ_1*/
 23 :  #define TCNT_1  (*(volatile unsigned int*)0xf716)   /*ﾀｲﾏｶｳﾝﾀ_1*/
 24 :  #define GRA_1   (*(volatile unsigned int*)0xf718)   /*ｼﾞｪﾈﾗﾙﾚｼﾞｽﾀA_1*/
 25 :  #define GRB_1   (*(volatile unsigned int*)0xf71a)   /*ｼﾞｪﾈﾗﾙﾚｼﾞｽﾀB_1*/
 26 :  #define GRC_1   (*(volatile unsigned int*)0xf71c)   /*ｼﾞｪﾈﾗﾙﾚｼﾞｽﾀC_1*/
 27 :  #define GRD_1   (*(volatile unsigned int*)0xf71e)   /*ｼﾞｪﾈﾗﾙﾚｼﾞｽﾀD_1*/
 28 :  #define TSTR    (*(volatile unsigned char*)0xf720)  /*ﾀｲﾏｽﾀｰﾄﾚｼﾞｽﾀ*/
 29 :  #define TMDR    (*(volatile unsigned char*)0xf721)  /*ﾀｲﾏﾓｰﾄﾞﾚｼﾞｽﾀ*/
 30 :  #define TPMR    (*(volatile unsigned char*)0xf722)  /*ﾀｲﾏPWMﾓｰﾄﾞﾚｼﾞｽﾀ*/
 31 :  #define TFCR    (*(volatile unsigned char*)0xf723)  /*ﾀｲﾏﾌｧﾝｸｼｮﾝｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ*/
 32 :  #define TOER    (*(volatile unsigned char*)0xf724)  /*ﾀｲﾏｱｳﾄﾌﾟｯﾄﾏｽﾀｲﾈｰﾌﾞﾙﾚｼﾞｽﾀ*/
 33 :  #define TOCR    (*(volatile unsigned char*)0xf725)  /*ﾀｲﾏｱｳﾄﾌﾟｯﾄｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ*/
 34 :  #define RSECDR  (*(volatile unsigned char*)0xf728)  /*秒ﾃﾞｰﾀﾚｼﾞｽﾀ/ﾌﾘｰﾗﾝｶｳﾝﾀﾃﾞｰﾀﾚｼﾞｽﾀ*/
 35 :  #define RMINDR  (*(volatile unsigned char*)0xf729)  /*分ﾃﾞｰﾀﾚｼﾞｽﾀ*/
 36 :  #define RHRDR   (*(volatile unsigned char*)0xf72a)  /*時ﾃﾞｰﾀﾚｼﾞｽﾀ*/
 37 :  #define RWKDR   (*(volatile unsigned char*)0xf72b)  /*曜日ﾃﾞｰﾀﾚｼﾞｽﾀ*/
 38 :  #define RTCCR1  (*(volatile unsigned char*)0xf72c)  /*RTCｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ1*/
 39 :  #define RTCCR2  (*(volatile unsigned char*)0xf72d)  /*RTCｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ2*/
 40 :  #define RTCCSR  (*(volatile unsigned char*)0xf72f)  /*ｸﾛｯｸｿｰｽｾﾚｸﾄﾚｼﾞｽﾀ*/
 41 :  #define LVDCR   (*(volatile unsigned char*)0xf730)  /*低電圧検出ｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ*/
 42 :  #define LVDSR   (*(volatile unsigned char*)0xf731)  /*低電圧検出ｽﾃｰﾀｽﾚｼﾞｽﾀ*/
 43 :  #define SMR_2   (*(volatile unsigned char*)0xf740)  /*ｼﾘｱﾙﾓｰﾄﾞﾚｼﾞｽﾀ_2*/
 44 :  #define BRR_2   (*(volatile unsigned char*)0xf741)  /*ﾋﾞｯﾄﾚｰﾄﾚｼﾞｽﾀ_2*/
 45 :  #define SCR3_2  (*(volatile unsigned char*)0xf742)  /*ｼﾘｱﾙｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ3_2*/
 46 :  #define TDR_2   (*(volatile unsigned char*)0xf743)  /*ﾄﾗﾝｽﾐｯﾄﾃﾞｰﾀﾚｼﾞｽﾀ_2*/
 47 :  #define SSR_2   (*(volatile unsigned char*)0xf744)  /*ｼﾘｱﾙｽﾃｰﾀｽﾚｼﾞｽﾀ_2*/
 48 :  #define RDR_2   (*(volatile unsigned char*)0xf745)  /*ﾚｼｰﾌﾞﾃﾞｰﾀﾚｼﾞｽﾀ_2*/
 49 :  #define ICCR1   (*(volatile unsigned char*)0xf748)  /*I2Cﾊﾞｽｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ1*/
 50 :  #define ICCR2   (*(volatile unsigned char*)0xf749)  /*I2Cﾊﾞｽｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ2*/
 51 :  #define ICMR    (*(volatile unsigned char*)0xf74a)  /*I2Cﾊﾞｽﾓｰﾄﾞﾚｼﾞｽﾀ*/
 52 :  #define ICIER   (*(volatile unsigned char*)0xf74b)  /*I2Cﾊﾞｽｲﾝﾀﾗﾌﾟﾄｲﾈｰﾌﾞﾙﾚｼﾞｽﾀ*/
 53 :  #define ICSR    (*(volatile unsigned char*)0xf74c)  /*I2Cﾊﾞｽｽﾃｰﾀｽﾚｼﾞｽﾀ*/
 54 :  #define SAR     (*(volatile unsigned char*)0xf74d)  /*ｽﾚｰﾌﾞｱﾄﾞﾚｽﾚｼﾞｽﾀ*/
 55 :  #define ICDRT   (*(volatile unsigned char*)0xf74e)  /*I2Cﾊﾞｽ送信ﾃﾞｰﾀﾚｼﾞｽﾀ*/
 56 :  #define ICDRR   (*(volatile unsigned char*)0xf74f)  /*I2Cﾊﾞｽ受信ﾃﾞｰﾀﾚｼﾞｽﾀ*/
 57 :  #define TMB1    (*(volatile unsigned char*)0xf760)  /*ﾀｲﾏﾓｰﾄﾞﾚｼﾞｽﾀB1*/
 58 :  #define TCB1    (*(volatile unsigned char*)0xf761)  /*ﾀｲﾏｶｳﾝﾀB1*/
 59 :  #define TLB1    (*(volatile unsigned char*)0xf761)  /*ﾀｲﾏﾛｰﾄﾞﾚｼﾞｽﾀB1*/
 60 :  #define FLMCR1  (*(volatile unsigned char*)0xff90)  /*ﾌﾗｯｼｭﾒﾓﾘｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ1*/
 61 :  #define FLMCR2  (*(volatile unsigned char*)0xff91)  /*ﾌﾗｯｼｭﾒﾓﾘｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ2*/
 62 :  #define FLPWCR  (*(volatile unsigned char*)0xff92)  /*ﾌﾗｯｼｭﾒﾓﾘﾊﾟﾜｰｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ*/
 63 :  #define EBR1    (*(volatile unsigned char*)0xff93)  /*ﾌﾞﾛｯｸ指定ﾚｼﾞｽﾀ1*/
 64 :  #define FENR    (*(volatile unsigned char*)0xff9b)  /*ﾌﾗｯｼｭﾒﾓﾘｲﾈｰﾌﾞﾙﾚｼﾞｽﾀ*/
 65 :  #define TCRV0   (*(volatile unsigned char*)0xffa0)  /*ﾀｲﾏｺﾝﾄﾛｰﾙﾚｼﾞｽﾀV0*/
 66 :  #define TCSRV   (*(volatile unsigned char*)0xffa1)  /*ﾀｲﾏｺﾝﾄﾛｰﾙ/ｽﾃｰﾀｽﾚｼﾞｽﾀV*/
 67 :  #define TCORA   (*(volatile unsigned char*)0xffa2)  /*ﾀｲﾑｺﾝｽﾀﾝﾄﾚｼﾞｽﾀA*/
 68 :  #define TCORB   (*(volatile unsigned char*)0xffa3)  /*ﾀｲﾑｺﾝｽﾀﾝﾄﾚｼﾞｽﾀB*/
 69 :  #define TCNTV   (*(volatile unsigned char*)0xffa4)  /*ﾀｲﾏｶｳﾝﾀV*/
 70 :  #define TCRV1   (*(volatile unsigned char*)0xffa5)  /*ﾀｲﾏｺﾝﾄﾛｰﾙﾚｼﾞｽﾀV1*/
 71 :  #define SMR     (*(volatile unsigned char*)0xffa8)  /*ｼﾘｱﾙﾓｰﾄﾞﾚｼﾞｽﾀ*/
 72 :  #define BRR     (*(volatile unsigned char*)0xffa9)  /*ﾋﾞｯﾄﾚｰﾄﾚｼﾞｽﾀ*/
 73 :  #define SCR3    (*(volatile unsigned char*)0xffaa)  /*ｼﾘｱﾙｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ3*/
 74 :  #define TDR     (*(volatile unsigned char*)0xffab)  /*ﾄﾗﾝｽﾐｯﾄﾃﾞｰﾀﾚｼﾞｽﾀ*/
 75 :  #define SSR     (*(volatile unsigned char*)0xffac)  /*ｼﾘｱﾙｽﾃｰﾀｽﾚｼﾞｽﾀ*/
 76 :  #define RDR     (*(volatile unsigned char*)0xffad)  /*ﾚｼｰﾌﾞﾃﾞｰﾀﾚｼﾞｽﾀ*/
 77 :  #define ADDRA   (*(volatile unsigned int*)0xffb0)   /*A/DﾃﾞｰﾀﾚｼﾞｽﾀA*/
 78 :  #define ADDRB   (*(volatile unsigned int*)0xffb2)   /*A/DﾃﾞｰﾀﾚｼﾞｽﾀB*/
 79 :  #define ADDRC   (*(volatile unsigned int*)0xffb4)   /*A/DﾃﾞｰﾀﾚｼﾞｽﾀC*/
 80 :  #define ADDRD   (*(volatile unsigned int*)0xffb6)   /*A/DﾃﾞｰﾀﾚｼﾞｽﾀD*/
 81 :  #define ADCSR   (*(volatile unsigned char*)0xffb8)  /*A/Dｺﾝﾄﾛｰﾙ/ｽﾃｰﾀｽﾚｼﾞｽﾀ*/
 82 :  #define ADCR    (*(volatile unsigned char*)0xffb9)  /*ｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ*/
 83 :  #define PWDRL   (*(volatile unsigned char*)0xffbc)  /*PWMﾃﾞｰﾀﾚｼﾞｽﾀL*/
 84 :  #define PWDRU   (*(volatile unsigned char*)0xffbd)  /*PWMﾃﾞｰﾀﾚｼﾞｽﾀU*/
 85 :  #define PWCR    (*(volatile unsigned char*)0xffbe)  /*PWMｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ*/
 86 :  #define TCSRWD  (*(volatile unsigned char*)0xffc0)  /*ﾀｲﾏｺﾝﾄﾛｰﾙ/ｽﾃｰﾀｽﾚｼﾞｽﾀWD*/
 87 :  #define TCWD    (*(volatile unsigned char*)0xffc1)  /*ﾀｲﾏｶｳﾝﾀWD*/
 88 :  #define TMWD    (*(volatile unsigned char*)0xffc2)  /*ﾀｲﾏﾓｰﾄﾞﾚｼﾞｽﾀWD*/
 89 :  #define ABRKCR  (*(volatile unsigned char*)0xffc8)  /*ｱﾄﾞﾚｽﾌﾞﾚｰｸｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ*/
 90 :  #define ABRKSR  (*(volatile unsigned char*)0xffc9)  /*ｱﾄﾞﾚｽﾌﾞﾚｰｸｽﾃｰﾀｽﾚｼﾞｽﾀ*/
 91 :  #define BARH    (*(volatile unsigned char*)0xffca)  /*ﾌﾞﾚｰｸｱﾄﾞﾚｽﾚｼﾞｽﾀH*/
 92 :  #define BARL    (*(volatile unsigned char*)0xffcb)  /*ﾌﾞﾚｰｸｱﾄﾞﾚｽﾚｼﾞｽﾀL*/
 93 :  #define BDRH    (*(volatile unsigned char*)0xffcc)  /*ﾌﾞﾚｰｸﾃﾞｰﾀﾚｼﾞｽﾀH*/
 94 :  #define BDRL    (*(volatile unsigned char*)0xffcd)  /*ﾌﾞﾚｰｸﾃﾞｰﾀﾚｼﾞｽﾀL*/
 95 :  #define PUCR1   (*(volatile unsigned char*)0xffd0)  /*ﾎﾟｰﾄﾌﾟﾙｱｯﾌﾟｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ1*/
 96 :  #define PUCR5   (*(volatile unsigned char*)0xffd1)  /*ﾎﾟｰﾄﾌﾟﾙｱｯﾌﾟｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ5*/
 97 :  #define PDR1    (*(volatile unsigned char*)0xffd4)  /*ﾎﾟｰﾄﾃﾞｰﾀﾚｼﾞｽﾀ1*/
 98 :  #define PDR2    (*(volatile unsigned char*)0xffd5)  /*ﾎﾟｰﾄﾃﾞｰﾀﾚｼﾞｽﾀ2*/
 99 :  #define PDR3    (*(volatile unsigned char*)0xffd6)  /*ﾎﾟｰﾄﾃﾞｰﾀﾚｼﾞｽﾀ3*/
100 :  #define PDR5    (*(volatile unsigned char*)0xffd8)  /*ﾎﾟｰﾄﾃﾞｰﾀﾚｼﾞｽﾀ5*/
101 :  #define PDR6    (*(volatile unsigned char*)0xffd9)  /*ﾎﾟｰﾄﾃﾞｰﾀﾚｼﾞｽﾀ6*/
102 :  #define PDR7    (*(volatile unsigned char*)0xffda)  /*ﾎﾟｰﾄﾃﾞｰﾀﾚｼﾞｽﾀ7*/
103 :  #define PDR8    (*(volatile unsigned char*)0xffdb)  /*ﾎﾟｰﾄﾃﾞｰﾀﾚｼﾞｽﾀ8*/
104 :  #define PDRB    (*(volatile unsigned char*)0xffdd)  /*ﾎﾟｰﾄﾃﾞｰﾀﾚｼﾞｽﾀB*/
105 :  #define PMR1    (*(volatile unsigned char*)0xffe0)  /*ﾎﾟｰﾄﾓｰﾄﾞﾚｼﾞｽﾀ1*/
106 :  #define PMR5    (*(volatile unsigned char*)0xffe1)  /*ﾎﾟｰﾄﾓｰﾄﾞﾚｼﾞｽﾀ5*/
107 :  #define PMR3    (*(volatile unsigned char*)0xffe2)  /*ﾎﾟｰﾄﾓｰﾄﾞﾚｼﾞｽﾀ3*/
108 :  #define PCR1    (*(volatile unsigned char*)0xffe4)  /*ﾎﾟｰﾄｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ1*/
109 :  #define PCR2    (*(volatile unsigned char*)0xffe5)  /*ﾎﾟｰﾄｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ2*/
```

```
110 :  #define PCR3    (*(volatile unsigned char*)0xffe6)  /*ﾎﾟｰﾄｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ3*/
111 :  #define PCR5    (*(volatile unsigned char*)0xffe8)  /*ﾎﾟｰﾄｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ5*/
112 :  #define PCR6    (*(volatile unsigned char*)0xffe9)  /*ﾎﾟｰﾄｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ6*/
113 :  #define PCR7    (*(volatile unsigned char*)0xffea)  /*ﾎﾟｰﾄｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ7*/
114 :  #define PCR8    (*(volatile unsigned char*)0xffeb)  /*ﾎﾟｰﾄｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ8*/
115 :  #define SYSCR1  (*(volatile unsigned char*)0xfff0)  /*ｼｽﾃﾑｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ1*/
116 :  #define SYSCR2  (*(volatile unsigned char*)0xfff1)  /*ｼｽﾃﾑｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ2*/
117 :  #define IEGR1   (*(volatile unsigned char*)0xfff2)  /*割り込みｴｯｼﾞｾﾚｸﾄﾚｼﾞｽﾀ1*/
118 :  #define IEGR2   (*(volatile unsigned char*)0xfff3)  /*割り込みｴｯｼﾞｾﾚｸﾄﾚｼﾞｽﾀ2*/
119 :  #define IENR1   (*(volatile unsigned char*)0xfff4)  /*割り込みｲﾈｰﾌﾞﾙﾚｼﾞｽﾀ1*/
120 :  #define IENR2   (*(volatile unsigned char*)0xfff5)  /*割り込みｲﾈｰﾌﾞﾙﾚｼﾞｽﾀ2*/
121 :  #define IRR1    (*(volatile unsigned char*)0xfff6)  /*割り込みﾌﾗｸﾞﾚｼﾞｽﾀ1*/
122 :  #define IRR2    (*(volatile unsigned char*)0xfff7)  /*割り込みﾌﾗｸﾞﾚｼﾞｽﾀ2*/
123 :  #define IWPR    (*(volatile unsigned char*)0xfff8)  /*ｳｪｲｸｱｯﾌﾟ割り込みﾌﾗｸﾞﾚｼﾞｽﾀ*/
124 :  #define MSTCR1  (*(volatile unsigned char*)0xfff9)  /*ﾓｼﾞｭｰﾙｽﾀﾝﾊﾞｲｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ1*/
125 :  #define MSTCR2  (*(volatile unsigned char*)0xfffa)  /*ﾓｼﾞｭｰﾙｽﾀﾝﾊﾞｲｺﾝﾄﾛｰﾙﾚｼﾞｽﾀ2*/
126 :  #define EKR     (*(volatile unsigned char*)0xff10)  /*EEPROMｷｰﾚｼﾞｽﾀ*/
```

「#define」は、「**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、

```
101 :  #define PDR6     (*(volatile unsigned char*)0xffd9)  /*ﾎﾟｰﾄﾃﾞｰﾀﾚｼﾞｽﾀ 6*/
```

は、「PDR6」という文字列が出てくると、「(*(volatile unsigned char*)0xffd9)」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、

```
        PDR6 = 0xff;
```

は、コンパイラは、

```
        (*(volatile unsigned char*)0xffd9) = 0xff;
```

と変換して実行しています。0xffd9 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、0x0000～0xdfff 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 ROM データが読み込まれます。
・RAM は、0xe800～0xefff 番地と 0xfb80～0xff7f 番地にあります。例えば、0xe810 番地に 0x55 を書き込むと 0x55 が保存されます。0xe810 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、0xf700～0xf77f 番地と 0xff80～0xffff 番地にあります。例えば、0xffd9 番地に 0xaa を書き込むと、ポート 6 から「1010 1010」が出力されます(ポート 6 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、0xffd9 番地は PDR6 と決められています。0xf700～0xf77f 番地と 0xff80～0xffff 番地には決められた名称があります。以下に、0xffd4～0xffeb 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
0xffd4	PDR1	0xffe0	PMR1
0xffd5	PDR2	0xffe1	PMR5
0xffd6	PDR3	0xffe2	PMR3
0xffd7		0xffe3	
0xffd8	PDR5	0xffe4	PCR1
0xffd9	PDR6	0xffe5	PCR2
0xffda	PDR7	0xffe6	PCR3
0xffdb	PDRB	0xffe7	
0xffdc	PDR8	0xffe8	PCR5
0xffdd	PDRB	0xffe9	PCR6
0xffde		0xffea	PCR7
0xffdf		0xffeb	PCR8

アドレスで設定してみます。
0xffd4 番地に 0xf7 を設定
0xffd5 番地に 0xff を設定
0xffd6 番地に 0xc0 を設定
0xffd8 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_3687.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_3687.h」をインクルードすることにより、
PDR1 に 0xf7 を設定
PDR2 に 0xff を設定
PDR3 に 0xc0 を設定
PDR5 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_3687.h で「PDR6 とは 0xffd9 番地のことです」と定義している訳です。

「#define PDR6　　(*(unsigned char*)0xffd9)」を分解してみると、

#define PDR6 0xffd9	PDR6 は、0xffd9 という値ですよ、という意味になります。
↓	
#define PDR6 *0xffd9	PDR6 は、0xffd9 番地ですよ、という意味になります。
↓	
#define PDR6 (unsigned char*)0xffd9	PDR6 は、符号無し８bit 幅(unsigned char)の 0xffd9 番地ですよ、という意味になります。
↓	
#define PDR6 *(unsigned char*)0xffd9	PDR6 は、符号無し８bit 幅の 0xffd9 番地の値ですよ、という意味になります。
↓	
#define PDR6 (*(unsigned char*)0xffd9)	同じ意味ですが、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、

```
        PDR6++;
```

とすると

```
        (*(unsigned char*)0xffd9)++;
```

と変換されます。これは、0xffd9 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*)0xffd9++;
```

となり、0xffd9 番地を＋1しなさい、という意味となります。0xffd9 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_3687.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、「H8/3687 シリーズ 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。

　最後に、すべての定義には「volatile」命令が付いています（下記）。

```
101 :  #define PDR6     (*(volatile unsigned char*)0xffd9)  /*ﾎﾟｰﾄﾃﾞｰﾀﾚｼﾞｽﾀ 6*/
```

　これは、コンパイラに対してプログラムの最適化をしないようにする命令です。例えば下記のようなプログラムがあったとします。volatile 命令がないとコンパイラが最適化を行い、プログラムを自動的に変換してしまいます。

```
PDR6 = 0xff;
PDR6 = 0x00;
```

コンパイラが
最適化してしまう

```
PDR6 = 0x00;
```

　そこで、「volatile」命令で、最適化してはいけないことをコンパイラに伝えます。コンパイラは最適化せず、プログラマの思い通りのプログラムになります。

```
PDR6 = 0xff;
PDR6 = 0x00;
```

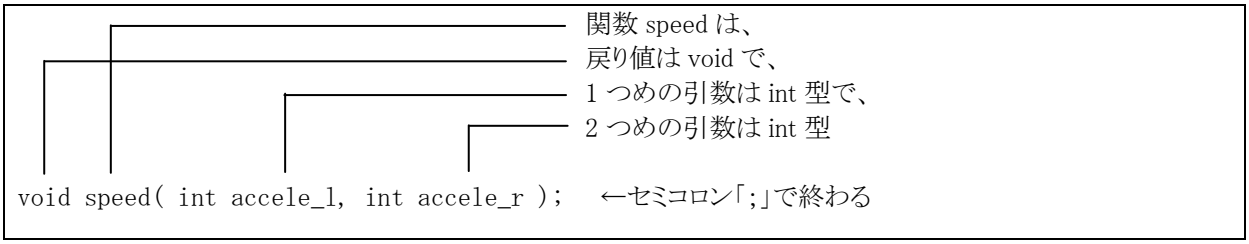
コンパイラは
最適化しない

```
PDR6 = 0xff;
PDR6 = 0x00;
```

6.6.4 プロトタイプ宣言

```
20 :  /*=====================================*/
21 :  /* プロトタイプ宣言                    */
22 :  /*=====================================*/
23 :  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。

6.6.5 init関数(I/Oポートの入出力設定)

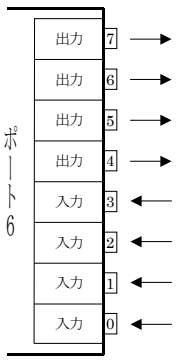
この関数で、H8/3687F マイコンに内蔵されている機能の初期化を行います。「init」とは、「initialize(イニシャライズ)」の略で、初期化の意味です。今回の演習では、init 関数内で I/O ポートの入出力設定を行っています。プログラムは下記のようになります。

```
44 :  void init( void )
45 :  {
46 :      /* I/Oポートの入出力設定 */
47 :      PCR1 = 0xff;
48 :      PCR2 = 0xfd;                          /* 通信ビットP22:TxD P21:RxD*/
49 :      PCR3 = 0xf0;                          /* 基板上のディップスイッチ */
50 :      PCR5 = 0x00;                          /* スイッチ基板             */
51 :      PCR6 = 0xff;                          /* LED基板                  */
52 :      PCR7 = 0xff;
53 :      PCR8 = 0xff;
54 :      /* ポートBは、入力専用なので入出力設定はありません。          */
55 :      /* また、A/D変換器のADCSRによって指定された1ビットだけは、    */
56 :      /* 入力ポートとしては使えません。                            */
57 :  }
```

(1) I/Oポートとは

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

(2) 入出力設定とポートコントロールレジスタ(PCR)



I/O ポートは 1～3、5～8、B の 8 ポートあります。1 ポートは、基本的には 8bit 単位ですが、それ以下のポートもあります(「**1.1 仕様**」のコネクタ欄を参照してください)。ポート 1～B は、それぞれポートコントロールレジスタ(PCR)とポートデータレジスタ(PDR)の 2 つのレジスタで構成されています。前者は入出力方向の決定、後者はデータの入力や出力を行っています。ただし、**ポート B のみ入力専用ポートとなっており、ポートコントロールレジスタ B(PCRB)はありません。必ず入力ポートとなります**。

ポート 6 を例として説明します。例題ですので io.c とは関係有りません。今回は左図のようにしたいとします。

ポートの入出力の設定は、ポートコントロールレジスタ(PCR)に値を設定することで行います。**入出力方向は各ビット自由に設定することができます**。

ポート 6 の入出力設定は、ポートコントロールレジスタ(PCR) の末尾にポート番号を付け、「PCR6」というレジスタ名になります。PCR1～PCR8 まであります。□部分が 1～8 まで変わります。ポート B は入力専用なので、ポートコントロールレジスタ B(PCRB)はありません。

端子を入力用にするか、出力用にするかは、下記のようにポートコントロールレジスタを設定します。

端子を入力用にしたい(初期値)　→ ポートコントロールレジスタ(PCR)の端子と同じビットを"0"にする
端子を出力用にしたい　→ ポートコントロールレジスタ(PCR)の端子と同じビットを"1"にする

先ほどの入出力設定を表にまとめます。今回はポート 6 の入出力設定を行うので、ポートコントロールレジスタ 6(PCR6)に設定することになります。

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

単純に、入力は”0”、出力は”1”にするだけでポートコントロールレジスタ 6(PCR6)に設定する内容になります。

bit	7	6	5	4	3	2	1	0
ポート 6 の 入出力設定	1	1	1	1	0	0	0	0

PCR6 には、「1111　0000」を設定します。C 言語では 2 進数表記ができないので、16 進数に変換してプログラムします。よって、

PCR6 ＝　0xf0

を設定します。

io.c に戻ります。ポート 6 は LED に接続しているので、すべて出力です。ポート 5 はスイッチに接続しているので、すべて入力です。ポート B のみ、入力専用ポートなので入出力設定はありません。必ず入力です。

```
50 :        PCR5 = 0x00;                    /* スイッチ基板          */
51 :        PCR6 = 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。下のように設定します。

ポート	設定値	
PCR1	0xff	未接続
PCR2	0xfd	bit1 は通信データ入力、bit2 は通信データ出力、他は未接続
PCR3	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
PCR5	0x00	スイッチ部が接続
PCR6	0xff	LED 部が接続
PCR7	0xff	未接続
PCR8	0xff	未接続
ポート B		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 B 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。**

6.6.6 データを出力する、読み込む

先ほどは、ポートコントロールレジスタ(PCR)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。

ここではポート 6 を例として、先ほどポートコントロールレジスタ 6(PCR6)=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) データの入出力とポートデータレジスタ(PDR)

各ポートにデータを入出力するには、**ポートデータレジスタ(PDR)** に値を設定することで行います。

ポート 6 にデータを入出力するには、ポートデータレジスタ(PDR) の末尾にポート番号を付け、「PDR6」というレジスタ名になります。PDR1～PDRB までであります。□部分が 1～B まで変わります。

◎端子が入力設定(PCR=0)の場合、ポートデータレジスタ(PDR)を読み込むことによりその端子の入力レベルが分かります。
端子に"0"(ローレベル)の電圧が入力されている場合→PDR の該当ビットが"0"になる
端子に"1"(ハイレベル)の電圧が入力されている場合→PDR の該当ビットが"1"になる

※PDR に値を書き込んでも何も起こりません。

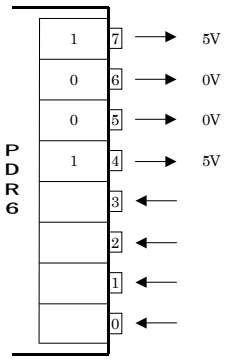
◎端子が出力設定(PCR=1)の場合、ポートデータレジスタ(PDR) にデータを書き込むことによりその端子から電圧が出力されます。
端子から"0"(ローレベル)の電圧を出力する場合→PDR の該当ビットを"0"にする
端子から"1"(ハイレベル)の電圧を出力する場合→PDR の該当ビットを"1"にする

※PDR を読み込むと、現在出力している値が読み込まれます。

まとめると下のようになります。

PCR の設定	PDR を読み込むと	PDR にデータを書き込むと
"0" 入力設定	端子の入力レベルが読み込まれます。	何も起こりません。
"1" 出力設定	現在、出力している値が読み込まれます。	端子にデータを出力します。

(2) ポートデータレジスタからデータ出力



左図のようにデータを出力させたいとします。

bit	7	6	5	4	3	2	1	0
ポート 6 から出力する値	1	0	0	1	入力端子	入力端子	入力端子	入力端子

入力端子へは、何を書き込んでも何も起こりません。ただ、"1"にすると、何か意味があるのかと疑問を持つかもしれないため、"0"にしておきます。まとめると、下記のようになります。

bit	7	6	5	4	3	2	1	0
ポート 6 から出力する値	1	0	0	1	0	0	0	0

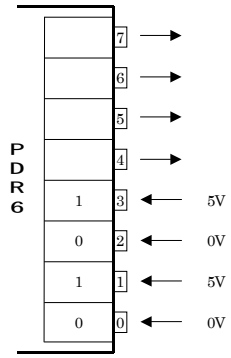
2 進数を 16 進数に変換してポートデータレジスタ 6(PDR6)へ設定します。

```
PDR6 = 0x90;
```

となります。

(3) ポートデータレジスタ 6(PDR6)からデータ入力

左図のように電圧が入力されているとします。



先ほどとは逆にポートデータレジスタ 6(PDR6)の値を読み込むと、入力状態が分かります。読み込んだ結果は、5V なら"1"、0V なら"0"となります。

bit	7	6	5	4	3	2	1	0
ポート 6 を 読み込んだ値 (入力端子)					1	0	1	0

では、出力に設定しているビットを読み込むと、どうなるのでしょう。それは現在出力している値が読み込まれます。例えば、先ほど 0x90 を設定しました。下記のようになります。

bit	7	6	5	4	3	2	1	0
ポート 6 を 読み込んだ値 (出力端子)	1	0	0	1				

ポートデータレジスタ 6(PDR6)には入力値と出力値が合わさった値が読み込まれます。下記ようなイメージです。

bit	7	6	5	4	3	2	1	0
ポート 6 を 読み込んだ値 (結論)	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**0**

プログラムで

```
c = PDR6;
```

とすると、変数 c には
1001 1010 ＝ 0x9a
が読み込まれます。

(4) 入力電圧

ポートを入力用に設定した場合、端子に 5V が入力されていれば"1"、0V が入力されていれば"0"となります。しかし実際には 0～5V の、どの電圧が入力されているか分かりません。2.5V が入力された場合、"0"になるのでしょうか？ "1"になるのでしょうか？ 端子や CPU ボードの電圧によって下記のようになります。

●"1"とみなす電圧

（特記なき場合、Vcc＝3.0～5.5V、Vss＝0.0V、Ta＝－20～＋75℃）

項目	記号	適用端子	測定条件	規格値			単位	備考
				Min	Typ	Max		
入力 High レベル電圧	V_{IH}	$\overline{RES}$、$\overline{NMI}$ $\overline{WKP0}$～$\overline{WKP5}$ $\overline{IRQ0}$～$\overline{IRQ3}$ $\overline{ADTRG}$、TMIB1 TMRIV、TMCIV	Vcc=4.0～5.5V	Vcc×0.8	—	Vcc＋0.3	V	
		FTIOA0～FTIOD0 FTIOA1～FTIOD1 SCK3、SCK3_2 TRGV		Vcc×0.9	—	Vcc＋0.3	V	
		RXD、RXD_2 SCL、SDA P10～P12 P14～P17 P20～P24	Vcc=4.0～5.5V	Vcc×0.7	—	Vcc＋0.3	V	
		P30～P37 P50～P57 P60～P67 P70～P72 P74～P76 P85～P87		Vcc×0.8	—	Vcc＋0.3	V	
		PB0～PB7	Vcc=4.0～5.5V	Vcc×0.7	—	AVcc＋0.3	V	
				Vcc×0.8	—	AVcc＋0.3	V	
		OSC1	Vcc=4.0～5.5V	Vcc－0.5	—	Vcc＋0.3	V	
				Vcc－0.3	—	Vcc＋0.3	V	

【注】 TEST 端子は Vss に接続してください。

●″0″とみなす電圧

（特記なき場合、Vcc＝3.0～5.5V、Vss＝0.0V、Ta＝－20～＋75℃）

項目	記号	適用端子	測定条件	規格値			単位	備考
				Min	Typ	Max		
入力 Low レベル電圧	V_{IL}	$\overline{RES}$、$\overline{NMI}$ $\overline{WKP0}$～$\overline{WKP5}$ $\overline{IRQ0}$～$\overline{IRQ3}$ $\overline{ADTRG}$、TMIB1 TMRIV、TMCIV	Vcc=4.0～5.5V	－0.3	—	Vcc×0.2	V	
		FTIOA0～FTIOD0 FTIOA1～FTIOD1 SCK3、SCK3_2 TRGV		－0.3	—	Vcc ×0.1	V	
		RXD、RXD_2 SCL、SDA P10～P12 P14～P17 P20～P24 P30～P37	Vcc=4.0～5.5V	－0.3	—	Vcc×0.3	V	
		P50～P57 P60～P67 P70～P72 P74～P76 P85～P87 PB0～PB7		－0.3	—	Vcc×0.2	V	
		OSC1	Vcc=4.0～5.5V	－0.3	—	0.5	V	
				－0.3	—	0.3	V	

Vcc は 5.0V とします。例えば、P60 の入力電圧を確かめてみます。
″1″と見なす電圧は、下記のようになります。
表より Min＝Vcc×0.7＝5.0×0.7＝3.5V
Max＝Vcc+0.3＝5.0+0.3＝5.3V
よって、3.5V～5.3V の電圧が入力されると″1″とみなします。

″0″と見なす電圧は、下記のようになります。
表より Min＝-0.3V
Max＝Vcc×0.3＝5.0×0.3＝1.5V
よって、-0.3V～1.5V の電圧が入力されると″0″とみなします。

ちなみに、それ以外の電圧
・-0.3V 以下
・1.5～3.5V
・5.3V 以上
が入力されたとき、どうなるのでしょうか。それは不定です。″0″か″1″か分かりません。実際にポートデータレジスタ(PDR)の値を読まないと分からないのです。

例えば、2.0V が入力されているとき、ポートデータレジスタ(PDR)を読み込むと"0"のときもあれば、"1"のときもあるかもしれません。"0"か"1"かで動作するプログラムを分けている場合、どちらのプログラムを実行するか分かりません。このような電圧が入力されないように回路で対応する必要があります。

ちなみに、端子に-0.3V 以下、または 7.0V 以上の電圧を加えると、端子が壊れます。そのような電圧は加えないように回路で対応してください。

(5) 出力電圧

ポートを出力用に設定した場合、ポートデータレジスタ(PDR)に"1"を設定すれば 5V が、"0"を設定すれば 0V が出力されます。しかしこれは理想です。実際には電圧の変動があります。変動幅は下表のようになります。端子によって違いがあります。

(特記なき場合、Vcc＝3.0～5.5V、Vss＝0.0V、Ta＝－20～＋75℃)

項目	記号	適用端子	測定条件	規格値			単位	備考
				Min	Typ	Max		
出力 High レベル電圧	V_{OH}	P10～P12 P14～P17 P20～P24 P30～P37	Vcc＝4.0～5.5V $-I_{OH}$＝1.5mA	Vcc－1.0	—	—	V	
		P50～P55 P60～P67 P70～P72 P74～P76 P85～P87	$-I_{OH}$＝0.1mA	Vcc－0.5	—	—	V	
		P56、P57	Vcc＝4.0～5.5V $-I_{OH}$＝0.1mA	Vcc－2.5	—	—	V	
			Vcc＝3.0～4.0V $-I_{OH}$＝0.1mA	Vcc－2.0	—	—	V	
出力 Low レベル電圧	V_{OL}	P10～P12 P14～P17 P20～P24 P30～P37	Vcc＝4.0～5.5V I_{OL}＝1.6mA	—	—	0.6	V	
		P50～P57 P70～P72 P74～P76 P85～P87	I_{OL}＝0.4mA	—	—	0.4	V	
		P60～P67	Vcc＝4.0～5.5V I_{OL}＝20.0mA	—	—	1.5	V	
			Vcc＝4.0～5.5V I_{OL}＝10.0mA	—	—	1.0	V	
			Vcc＝4.0～5.5V I_{OL}＝1.6mA	—	—	0.4	V	
			I_{OL}＝0.4mA	—	—	0.4	V	
		SCL、SDA	Vcc＝4.0～5.5V I_{OL}＝6.0mA	—	—	0.6	V	
			I_{OL}＝3.0mA	—	—	0.4	V	

Vcc は 5.0V とします。例えば、P60 の出力電圧を確かめてみます。

″1″を設定して出力される電圧は、下記のようになります。

表より　I_{OH}=1.5mA（端子から流れる電流が 1.5mA）のとき、Min＝Vcc−1.0＝5.0−1.0＝4.0V

I_{OH}=0.1mA（端子から流れる電流が 0.5mA）のとき、Min＝Vcc−0.5＝5.0−0.5＝4.5V

よって、流れる電流が 1.5mA のとき最小でも 4.0V の電圧が出力されます。流れる電流が 0.5mA のとき、最小でも 4.5V の電圧が出力されます。ちなみに、最高は書かれていませんが、電源電圧となります。

″0″を設定して出力される電圧は、下記のようになります。

表より　I_{OL}=20.0mA（端子に流れてくる電流が 20.0mA）のとき、Max＝1.5V

I_{OL}=10.0mA（端子に流れてくる電流が 10.0mA）のとき、Max＝1.0V

I_{OL}=1.6mA（端子に流れてくる電流が 1.6mA）のとき、Max＝0.4V

よって、流れてくる電流が 20.0mA のとき最大でも 1.5V の電圧が出力されます。流れてくる電流が 10.0mA のとき最大でも 1.0V の電圧が出力されます。流れてくる電流が 1.6mA のとき最大でも 0.4V の電圧が出力されます。ちなみに、最低は書かれていませんが、GND 電圧、要は 0V となります。

(6) 出力電流

端子に流すことのできる電流は、下表のようになっています。

（特記なき場合、Vcc＝3.0～5.5V、Vss＝0.0V、Ta＝－20～＋75℃）

項目	記号	適用端子	測定条件	規格値			単位
				Min	Typ	Max	
出力 Low レベル許容電流（1 端子あたり）	I_{OL}	ポート 6、SCL、SDA 以外の出力端子	Vcc=4.0～5.5V	—	—	2.0	mA
		ポート 6		—	—	20.0	mA
		ポート 6、SCL、SDA 以外の出力端子		—	—	0.5	mA
		ポート 6		—	—	10.0	mA
		SCL、SDA		—	—	6.0	mA
出力 Low レベル許容電流（総和）	ΣI_{OL}	ポート 6、SCL、SDA 以外の出力端子	Vcc=4.0～5.5V	—	—	40.0	mA
		ポート 6、SCL、SDA		—	—	80.0	mA
		ポート 6、SCL、SDA 以外の出力端子		—	—	20.0	mA
		ポート 6、SCL、SDA		—	—	40.0	mA
出力 High レベル許容電流（1 端子あたり）	$\lvert -I_{OH} \rvert$	全出力端子	Vcc=4.0～5.5V	—	—	2.0	mA
				—	—	0.2	mA
出力 High レベル許容電流（総和）	$\lvert -\Sigma I_{OH} \rvert$	全出力端子	Vcc=4.0～5.5V	—	—	30.0	mA
				—	—	8.0	mA

Vcc は 5.0V とします。例えば、P60 の出力電流を確かめてみます。

″0″(Low レベル)を設定したとき流せる電流は、表より下記のようになります。

Max＝20.0mA

ただし、総和に気をつける必要があります。表より″0″を設定したときに流せる全端子の合計電流は 80.0mA までとなります。例えば、P60=20.0mA、P61=20.0mA、P62=20.0mA、P63=20.0mA、P64=20.0mA を流したとき、1 端子だけ見たときは、表の 20.0mA 以下に当てはまるので問題ないような気もしますが、合計は 100.0mA となり、80.0mA を超えてしまいます。この電流の流し方はできません。

″1″(High レベル)を設定したとき流せる電流は、表より下記のようになります。

Max＝2.0mA

ただし、こちらも総和に気をつける必要があります。表より″1″を設定したときに流せる全端子の合計電流は 30.0mA までとなります。例えば、20 ビット分の端子すべてに 2.0mA を流したとき、1 端子だけ見たときは、表の 2.0mA 以下に当てはまるので問題ないような気もしますが、合計は 40.0mA となり、30.0mA を超えてしまいます。この電流の流し方はできません。

6.6.7 main 関数

```
 32 :  void main( void )
 33 :  {
 34 :      unsigned char d;
 35 :
 36 :      init();                              /* マイコン機能の初期化      */
 37 :
 38 :      while( 1 ) {
 39 :          d = PDR5;
 40 :          PDR6 = d;
 41 :      }
 42 :  }
```

36 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。

39 行で、ポート 5 の状態を読み込み変数dへ代入、変数dの値をポート 6 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 5 は全ビット入力、ポート 6 は全ビット出力なので、ポート 5 の状態がそのままポート 6 へ出力されます。

38 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。

6.7 ビット操作のプログラムテクニック

6.7.1 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、ポートデータレジスタ 5(PDR5)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。

```
38 :      while( 1 ) {
39 :          d = ~PDR5;
40 :          PDR6 = d;
41 :      }
```

6.7.2 特定のビットを"0"にする

例えば、ポート 5 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、ポートデータレジスタ 5(PDR5)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	
PDR5	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	

```
38 :      while( 1 ) {
39 :          d = PDR5 & 0xf0;
40 :          PDR6 = d;
41 :      }
```

6.7.3 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ビット7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。マスクというと風邪をひいたときに口元に付けるマスクを連想します。風邪用マスクは風邪の菌をまき散らさないためにつけますが、ここでいうマスクはその見た目をいいます。マスクを付けると口が見えません。隠しています。そうです。ここでいうマスクは『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 処理して不必要なビットを強制的に"0"にすることです**。

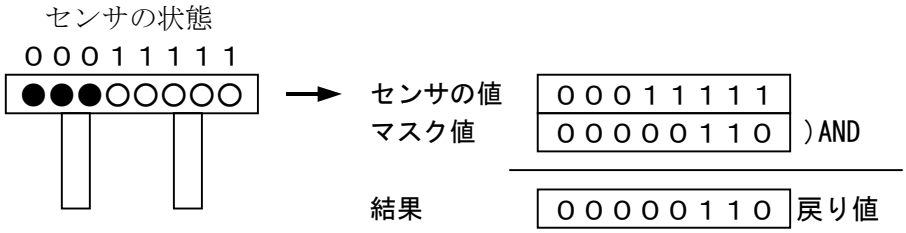
例えば、センサの状態が「黒黒黒白白白白白」のとき、センサ値は「00011111」です。bit2,1 のみチェックに必要で、他は必要なしとします。

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「00000110」、16進数に直すと「0x06」となります。下表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、下記のようになります。

```
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。

6.7.4 特定のビットを”1”にする

例えば、ポート 5 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、ポートデータレジスタ 5(PDR5)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	
PDR5	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	

```
38 :      while( 1 ) {
39 :          d = PDR5 | 0xc3;
40 :          PDR6 = d;
41 :      }
```

6.7.5 特定のビットを反転する

例えば、ポートデータレジスタ 5(PDR5)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、PDR5 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	
PDR5	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	

```
38 :      while( 1 ) {
39 :          d = PDR5 ^ 0xaa;
40 :          PDR6 = d;
41 :      }
```

※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 かなので、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は 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 6 から「ON,OFF,ON,OFF,OFF,ON,OFF,ON」の信号が出力されます。

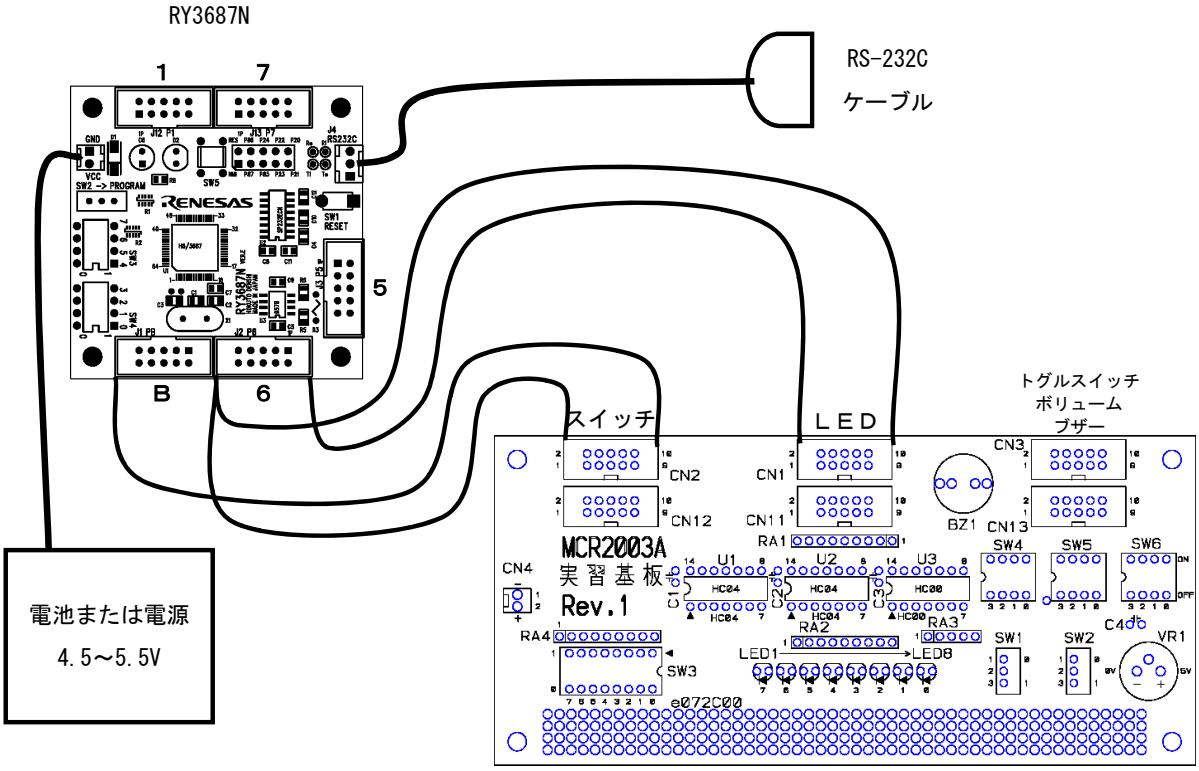
```
PDR6 = 0xa5;
```

6.8 ポートBを使用する場合の注意点

ポート B を使用する場合、注意しなければいけないことがあります。

6.8.1 接続

・CPU ボードの**ポート B** 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 6 と、実習基板の LED 部をフラットケーブルで接続します。



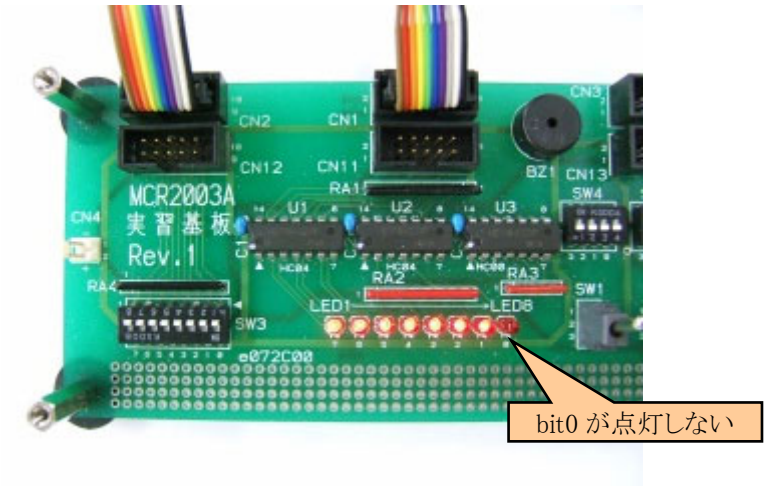
6.8.2 プログラム

スイッチの入力を、ポート B にします。ポート B には、PCR がありませんので設定不要です。

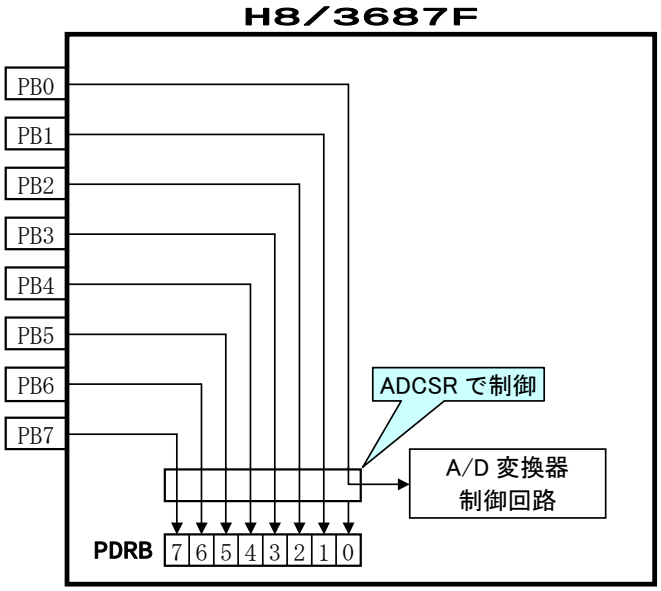
```
void main( void )
{
    unsigned char d;

    init();                                /* マイコン機能の初期化       */

    while( 1 ) {
        d = PDRB;
        PDR6 = d;
    }
}
```

6.8.3 動作確認

ディップスイッチを動かして、LED の点き方を確認します。bit0 の LED だけ、スイッチをどう動かしても点灯しません。コネクタの接触不良？端子が壊れた？　原因は何でしょうか。



実は、この動作で正常です。ポート B は通常のデジタル値入力の他、アナログ電圧入力端子でもあります。**ポート B は、必ず 1 端子以上はアナログ入力端子なのです**。初期は、bit0 がアナログ電圧入力端子となっています。アナログ入力端子は、ポートデータレジスタ B（PDRB）とは内部回路が切り離されています。そのため、PDRB では読めないのです（下図）。



どの端子をアナログ入力端子にするかは、A/D コントロール／ステータスレジスタ(ADCSR)で制御します。

ADCSR			内容
bit2	bit1	bit0	
0	0	0	PB0 をアナログ入力端子にする(**初期値**)
0	0	1	PB1 をアナログ入力端子にする
0	1	0	PB2 をアナログ入力端子にする
0	1	1	PB3 をアナログ入力端子にする
1	0	0	PB4 をアナログ入力端子にする
1	0	1	PB5 をアナログ入力端子にする
1	1	0	PB6 をアナログ入力端子にする
1	1	1	PB7 をアナログ入力端子にする

例えば、ADCSR=0x01 を設定すると、PB1 端子がアナログ入力端子となり、ポートデータレジスタ B(PDRB)とは切り離されます。

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

    init();                                 /* マイコン機能の初期化       */
    ADCSR = 0x01;

    while( 1 ) {
        d = PDRB;
        PDR6 = d;
    }
}
```

●ポイント

ポート B は、必ず 1 端子はアナログ入力端子となり、その端子はポートデータレジスタ B(PDRB)から読みこめません。

6.8.4 ポートBの端子を 8bit読み込むには

前記の通り、ポート B は 8 端子中、7 端子しか読み込むことができません。どうしようもないのでしょうか。2 回に分けて読み込めばよいのです。

ADCSR = 0x00;
d1 = PDRB;
ADCSR = 0x01;
d2 = PDRB;
d = d1 | d2;
PDR6 = d;
	PB0 端子をアナログ入力端子にする
	PB7～PB1 を読み込み(**1回目**)
	PB1 端子をアナログ入力端子にする
	PB0 を読み込み(**2回目**)
	PB7～PB1 の電圧レベルと PB0 の電圧レベルの OR をとる
	ポート 6 へ出力

変数の宣言も含めてプログラム化します。

```
void main( void )
{
    unsigned char d, d1, d2;

    init();                              /* マイコン機能の初期化      */

    while( 1 ) {
        ADCSR = 0x00;
        d1 = PDRB;
        ADCSR = 0x01;
        d2 = PDRB;
        d = d1 | d2;
        PDR6 = d;
    }
}
```

7. 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）

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）

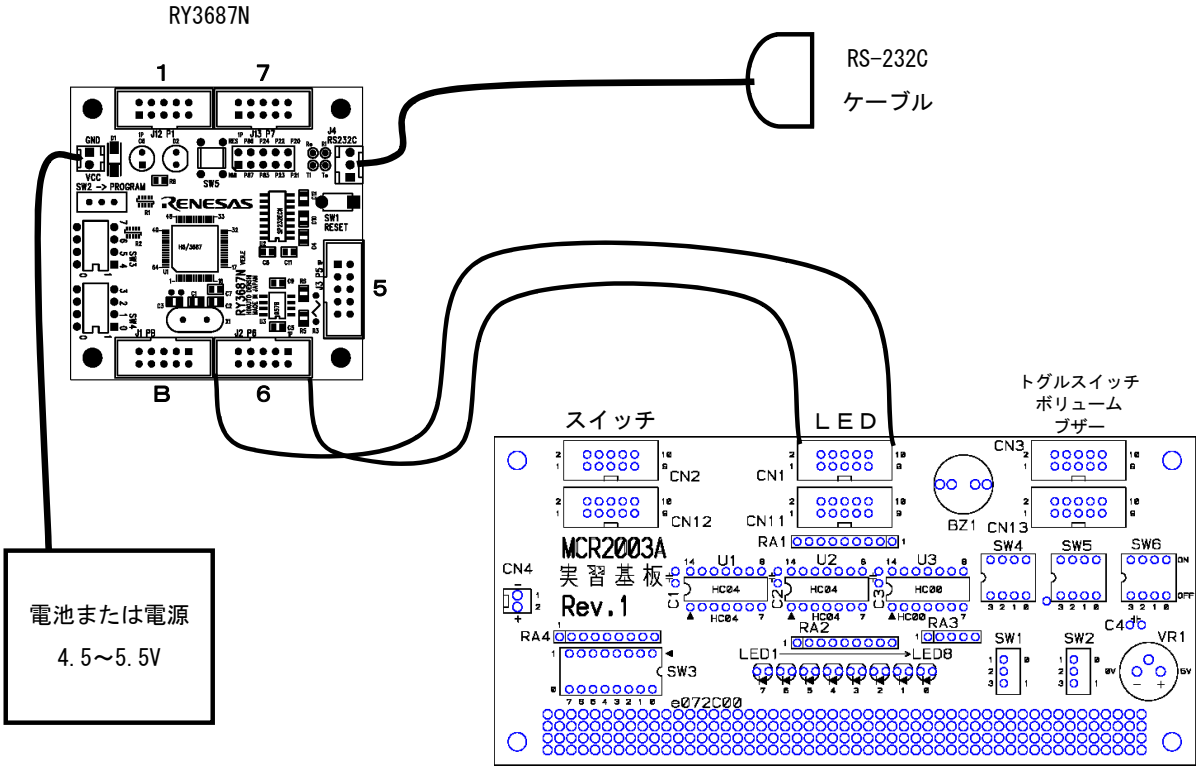
を繰り返し出力し続けます。マイコンのポートは、下記を使用します。

・ポート 6 の全ビット・・・LED へデータ出力

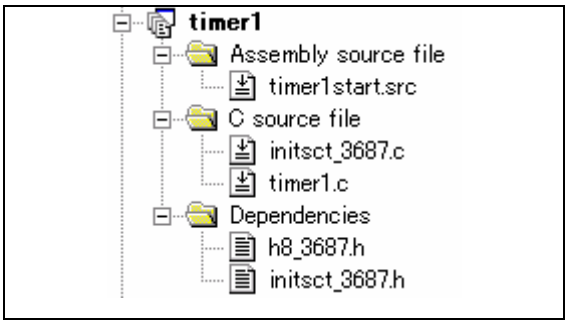
7.2 接続

・CPU ボードのポート 6 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、RY3687N ボードのコネクタをすべて使用すれば 35 個の LED の制御が可能です（PB は入力専用なので LED の点灯はできません）。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。



7.3 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_3687.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。H8/3687F の内蔵周辺機能の初期化も行います。
4	h8_3687.h	H8/3687F 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_3687.h	initsct_3687.c のヘッダファイルです。

7.4 プログラム「timer1.c」

```
 1 :  /*********************************************************************/
 2 :  /* ソフトタイマ "timer1.c"                                           */
 3 :  /*                          2007.04 ジャパンマイコンカーラリー実行委員会 */
 4 :  /*********************************************************************/
 5 :  /*
 6 :  出力：P67-P60(LEDなど)
 7 :
 8 :  ポート６に繋いだLEDを1秒間隔で点滅させます。
 9 :  */
10 :
11 :  /*====================================*/
12 :  /* インクルード                       */
13 :  /*====================================*/
14 :  #include    <machine.h>
15 :  #include    "h8_3687.h"
16 :
17 :  /*====================================*/
18 :  /* プロトタイプ宣言                   */
19 :  /*====================================*/
20 :  void init( void );
21 :  void timer( unsigned long timer_set );
22 :
23 :  /*====================================*/
24 :  /* グローバル変数の宣言               */
25 :  /*====================================*/
26 :
27 :  /*********************************************************************/
28 :  /* メインプログラム                                                  */
29 :  /*********************************************************************/
30 :  void main( void )
31 :  {
32 :      init();                             /* マイコン機能の初期化      */
33 :
34 :      while( 1 ) {
35 :          PDR6 = 0x55;
36 :          timer( 1000 );
37 :          PDR6 = 0xaa;
38 :          timer( 1000 );
```

```
39 :          PDR6 = 0x00;
40 :          timer( 1000 );
41 :      }
42 :  }
43 :
44 :  /************************************************************************/
45 :  /* H8/3687F 内蔵周辺機能  初期化                                        */
46 :  /************************************************************************/
47 :  void init( void )
48 :  {
49 :      /* I/Oポートの入出力設定 */
50 :      PCR1 = 0xff;
51 :      PCR2 = 0xfd;                        /* 通信ビットP22:TxD P21:RxD*/
52 :      PCR3 = 0xf0;                        /* 基板上のディップスイッチ */
53 :      PCR5 = 0xff;
54 :      PCR6 = 0xff;                        /* LED基板                  */
55 :      PCR7 = 0xff;
56 :      PCR8 = 0xff;
57 :      /* ポートBは、入力専用なので入出力設定はありません。          */
58 :      /* また、A/D変換器のADCSRによって指定された1ビットだけは、   */
59 :      /* 入力ポートとしては使えません。                             */
60 :  }
61 :
62 :  /************************************************************************/
63 :  /* タイマ本体                                                           */
64 :  /* 引数   タイマ値 1=1ms                                                */
65 :  /* 戻り値 なし                                                          */
66 :  /************************************************************************/
67 :  #pragma option speed = noloop
68 :  void timer( unsigned long timer_set )
69 :  {
70 :      unsigned long m, n;
71 :
72 :      for( m=0; m<timer_set; m++ ) {
73 :          for( n=0; n<2456; n++ );
74 :      }
75 :  }
76 :  #pragma option speed
77 :
78 :  /************************************************************************/
79 :  /* End of file                                                          */
80 :  /************************************************************************/
```

7.5 プログラムの解説

7.5.1 I/Oポートの入出力設定

```
 47 :  void init( void )
 48 :  {
 49 :      /* I/Oポートの入出力設定 */
 50 :      PCR1 = 0xff;
 51 :      PCR2 = 0xfd;                          /* 通信ビットP22:TxD P21:RxD*/
 52 :      PCR3 = 0xf0;                          /* 基板上のディップスイッチ */
 53 :      PCR5 = 0xff;
 54 :      PCR6 = 0xff;                          /* LED基板                  */
 55 :      PCR7 = 0xff;
 56 :      PCR8 = 0xff;
 57 :      /* ポートBは、入力専用なので入出力設定はありません。          */
 58 :      /* また、A/D変換器のADCSRによって指定された1ビットだけは、   */
 59 :      /* 入力ポートとしては使えません。                            */
 60 :  }
```

ポート 6 に LED 部を接続しますので、ポート 6 は出力に設定します。開放ポートも出力に設定します。

7.5.2 timer関数

```
 67 :  #pragma option speed = noloop
 68 :  void timer( unsigned long timer_set )
 69 :  {
 70 :      unsigned long m, n;
 71 :
 72 :      for( m=0; m<timer_set; m++ ) {
 73 :          for( n=0; n<2456; n++ );
 74 :      }
 75 :  }
 76 :  #pragma option speed
```

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。

```
 72 :      for( m=0; m<timer_set; m++ ) {     ←この行で 1ms を何回繰り返すかチェック
 73 :          for( n=0; n<2456; n++ );       ←この行で 1ms の時間稼ぎ
 74 :      }
```

この 3 行が時間稼ぎをしている部分です。

73 行では 2456 回、変数 n を足しています。そして 2456 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。73 行の1行だけで 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 72 行です。変数 m 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

m ＜ 1000

が成り立つまで 73 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、2456 が 1[ms]だと分かったのでしょうか。実は、

```
timer( 100000 );
```

として、ストップウォッチで測定、ちょうど 100 秒間になった値が 2456 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。

7.5.3 #pragmaについて

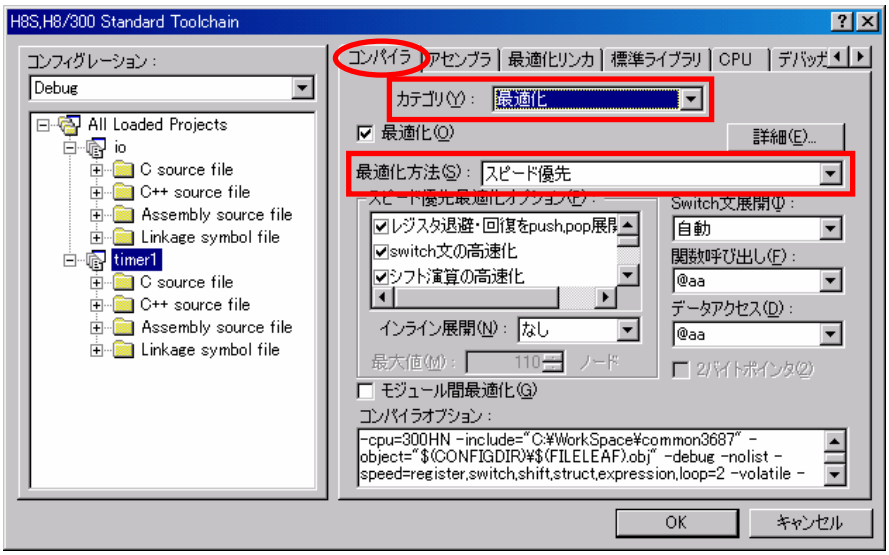
timer 関数の前後には、＃が先頭にある文が 2 行あります。

```
67 :  #pragma option speed = noloop
68 :  void timer( unsigned long timer_set )
69 :  {
70 :      unsigned long m, n;
71 :
72 :      for( m=0; m<timer_set; m++ ) {
73 :          for( n=0; n<2456; n++ );
74 :      }
75 :  }
76 :  #pragma option speed
```

これはどの様な意味でしょうか。その前に、コンパイラの設定について説明します。

コンパイラが C 言語プログラムを機械語に変換するとき、下記のどちらで変換するか、選ぶことができます。
・スピード優先 →スピードは速いが、機械語に変換したサイズが大きめの設定
・サイズ優先 →機械語に変換したサイズは小さいが、スピードが遅めの設定

この設定は、次の操作で確認することができます。ルネサス統合開発環境の「ビルド→H8S,H8/300 Standard Toolchain」を選択します。コンパイラを選び、「カテゴリ:最適化」を選択します。



「最適化方法」欄が、その設定です。プロジェクト「timer1」は、
・スピード優先→スピードは速いが、機械語に変換したサイズが大きめの設定
設定になっています。
timer 関数をもう一度確認してみます。

```
68 :  void timer( unsigned long timer_set )
69 :  {
70 :      unsigned long m, n;
71 :
72 :      for( m=0; m<timer_set; m++ ) {
73 :          for( n=0; n<2456; n++ );
74 :      }
75 :  }
```

for 文は**何もしないことを繰り返しているので**、コンパイラが自動的に最適化して実行回数を減らしてしまうことがある
どのように最適化するかは、for 文内の 2456 の数字によって変わる

for 文は**何もしないことを繰り返しているので**、コンパイラが自動的に最適化して実行回数を減らしてしまうことがあります。どのように最適化するかは、for 文内の 2456 の数字によって変わります。下記は、コンパイラがアセンブリソースリストに変換したした結果です。

```
    67:    void timer( unsigned long timer_set )
0038                    _timer:                         ; function: timer
0038  01006DF5              PUSH.L      ER5
    68:    {
    69:        unsigned long m, n;
    70:
    71:        for( m=0; m<timer_set; m++ ) {
003C  1AD5                  SUB.L       ER5,ER5
003E  4000                  BRA         L67:8
0040                    L66:
    72:            for( n=0; n<2456; n++ );
0040  7A0100000998          MOV.L       #2456,ER1
0046                    L68:
0046  1B71                  DEC.L       #1,ER1
0048  1B71                  DEC.L       #1,ER1
004A  4600                  BNE         L68:8
004C  0B75                  INC.L       #1,ER5
004E                    L67:
004E  1F85                  CMP.L       ER0,ER5
0050  4500                  BLO         L66:8
    73:        }
    74:    }
0052  01006D75              POP.L       ER5
0056  5470                  RTS
```

コンパイラが早くループを終わらせようとして、自動的に 1 回で 2 つ引いてしまう
そのため、予期していない動作になる

そこでコンパイラに、「timer 関数内は、最適化しないようにしてください」とお願いします。それが＃の付いた行です。

```
67:  #pragma option speed = noloop        この行以降は、ループの最適化を禁止

     timer 関数

76 :  #pragma option speed                この行以降から解除
```

オプションを付けることにより、コンパイラは自動的に最適化せず、プログラムした通りの動きになります。

#pragma 文については、H8S、H8/300 シリーズ C/C++コンパイラ、アセンブラ、最適化リンケージエディタ コンパイラパッケージ Ver.6.01 ユーザーズマニュアルの「10.2.1 #pragma 拡張子、キーワード」を参照してください。

7.5.4 main関数

```
30 :  void main( void )
31 :  {
32 :      init();                              /* マイコン機能の初期化      */
33 :
34 :      while( 1 ) {
35 :          PDR6 = 0x55;
36 :          timer( 1000 );
37 :          PDR6 = 0xaa;
38 :          timer( 1000 );
39 :          PDR6 = 0x00;
40 :          timer( 1000 );
41 :      }
42 :  }
```

32 行目で、init 関数を呼んでポートの入出力設定を行います。
34 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 41 行まで無限ループです。
35 行目でポート 6 に 0x55 を出力、1000 ミリ秒時間稼ぎします。
37 行目でポート 6 に 0xaa を出力、1000 ミリ秒時間稼ぎします。
39 行目でポート 6 に 0x00 を出力、1000 ミリ秒時間稼ぎします。

8. プロジェクト「timer2」 割り込みによるタイマ

8.1 概要

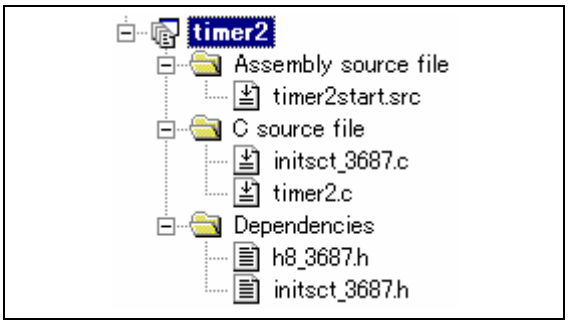
LED 8 個を 1 秒ごとに
●○●○●○●○ (16 進数で 0x55)　　※●＝LED 消灯　○＝LED 点灯
○●○●○●○● (16 進数で 0xaa)
●●●●●●●● (16 進数で 0x00)
を繰り返し出力し続けます。マイコンのポートは、下記を使用します。
・ポート 6 の全ビット・・・LED へデータ出力

8.2 接続

・CPU ボードのポート 6 と、実習基板の LED 部をフラットケーブルで接続します。

※プロジェクト「timer1」と同じです。

8.3 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 この演習は、割り込みを使うので今までの src ファイルとは一部異なります。
2	initsct_3687.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。H8/3687F の内蔵周辺機能の初期化も行います。
4	h8_3687.h	H8/3687F 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_3687.h	initsct_3687.c のヘッダファイルです。

8.4 プログラム「timer2.c」

```
 1 : /*********************************************************************/
 2 : /* タイマB1の割り込みによるタイマ "timer2.c"                         */
 3 : /*                          2007.04 ジャパンマイコンカーラリー実行委員会 */
 4 : /*********************************************************************/
 5 : /*
 6 : 出力：P67-P60(LEDなど)
 7 :
 8 : ポート６に繋いだLEDを1秒間隔で点滅させます。
 9 : */
10 :
11 : /*====================================*/
12 : /* インクルード                       */
13 : /*====================================*/
14 : #include    <machine.h>
15 : #include    "h8_3687.h"
16 :
17 : /*====================================*/
18 : /* プロトタイプ宣言                   */
19 : /*====================================*/
20 : void init( void );
21 : void timer( unsigned long timer_set );
22 :
23 : /*====================================*/
24 : /* グローバル変数の宣言               */
25 : /*====================================*/
26 : unsigned long cnt0;                    /* タイマB1                 */
27 :
28 : /*********************************************************************/
29 : /* メインプログラム                                                  */
30 : /*********************************************************************/
31 : void main( void )
32 : {
33 :     init();                           /* マイコン機能の初期化     */
34 :     set_ccr( 0x00 );                  /* 全体割り込み許可         */
35 :
36 :     while( 1 ) {
37 :         PDR6 = 0x55;
38 :         timer( 1000 );
39 :         PDR6 = 0xaa;
40 :         timer( 1000 );
41 :         PDR6 = 0x00;
42 :         timer( 1000 );
43 :     }
44 : }
45 :
46 : /*********************************************************************/
47 : /* H8/3687F 内蔵周辺機能　初期化                                     */
48 : /*********************************************************************/
49 : void init( void )
50 : {
51 :     /* I/Oポートの入出力設定 */
52 :     PCR1 = 0xff;
53 :     PCR2 = 0xfd;                      /* 通信ビットP22:TxD P21:RxD*/
54 :     PCR3 = 0xf0;                      /* 基板上のディップスイッチ */
55 :     PCR5 = 0xff;
56 :     PCR6 = 0xff;                      /* LED基板                  */
57 :     PCR7 = 0xff;
58 :     PCR8 = 0xff;
59 :     /* ポートBは、入力専用なので入出力設定はありません。        */
60 :     /* また、A/D変換器のADCSRによって指定された1ビットだけは、  */
61 :     /* 入力ポートとしては使えません。                           */
62 :
63 :     /* タイマB1 1msごとの割り込み設定 */
64 :     TMB1 = 0x84;                      /* 入力クロックの設定等     */
65 :     TLB1 = 26;                        /* カウンタ初期値設定       */
66 :     IENR2 = 0x20;                     /* 割り込み要求許可         */
67 : }
68 :
69 : /*********************************************************************/
70 : /* タイマ本体                                                        */
71 : /* 引数　タイマ値 1=1ms                                              */
72 : /* 戻り値 なし                                                       */
73 : /*********************************************************************/
74 : void timer( unsigned long timer_set )
75 : {
76 :     cnt0 = 0;
77 :     while( cnt0 < timer_set );
78 : }
79 :
```

```
/************************************************************************/
/* タイマB1 割り込み処理                                                */
/************************************************************************/
#pragma interrupt( interrupt_timerB1 )
void interrupt_timerB1( void )
{
    IRR2 &= 0xdf;                       /* フラグクリア             */
    cnt0++;
}

/************************************************************************/
/* End of file                                                          */
/************************************************************************/
```

8.5 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。

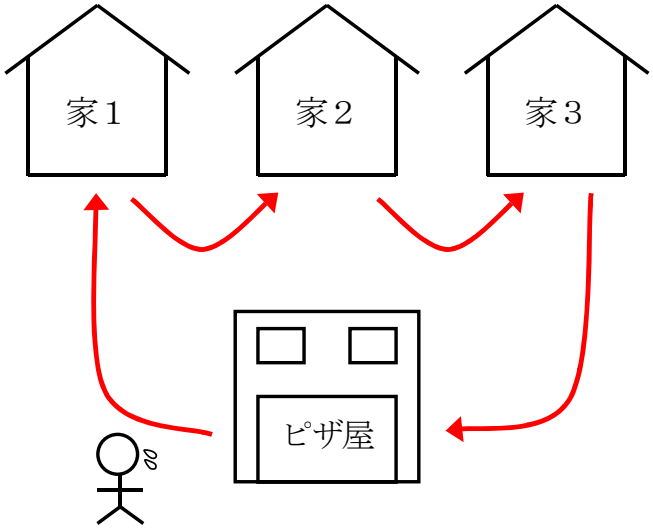
```
;=====================================
; 外部参照
;=====================================
        .IMPORT _main
        .IMPORT _INITSCT
        .IMPORT _interrupt_timerB1;                          ←追加

;=====================================
; ベクタセクション
;=====================================
        .SECTION V
        .DATA.W RESET_START             ;  0 リセット、WDT
        .DATA.W H'FFFF                  ;  1 システム予約
        .DATA.W H'FFFF                  ;  2 システム予約
        .DATA.W H'FFFF                  ;  3 システム予約
        .DATA.W H'FFFF                  ;  4 システム予約
        .DATA.W H'FFFF                  ;  5 システム予約
        .DATA.W H'FFFF                  ;  6 システム予約
        .DATA.W H'FFFF                  ;  7 外部割り込み端子
        .DATA.W H'FFFF                  ;  8 トラップ命令#0
        .DATA.W H'FFFF                  ;  9 トラップ命令#1
        .DATA.W H'FFFF                  ; 10 トラップ命令#2
        .DATA.W H'FFFF                  ; 11 トラップ命令#3
        .DATA.W H'FFFF                  ; 12 ブレーク条件成立
        .DATA.W H'FFFF                  ; 13 スリープ命令の実行による直接遷移
        .DATA.W H'FFFF                  ; 14 IRQ0
        .DATA.W H'FFFF                  ; 15 IRQ1
        .DATA.W H'FFFF                  ; 16 IRQ2
        .DATA.W H'FFFF                  ; 17 IRQ3
        .DATA.W H'FFFF                  ; 18 WKP
        .DATA.W H'FFFF                  ; 19 オーバフロー
        .DATA.W H'FFFF                  ; 20 システム予約
        .DATA.W H'FFFF                  ; 21
        .DATA.W H'FFFF                  ; 22 タイマV
        .DATA.W H'FFFF                  ; 23 SCI3
        .DATA.W H'FFFF                  ; 24 IIC2
        .DATA.W H'FFFF                  ; 25 AD変換器
        .DATA.W H'FFFF                  ; 26 タイマZ A0-D0
        .DATA.W H'FFFF                  ; 27 タイマZ A1-D1
        .DATA.W H'FFFF                  ; 28
        .DATA.W _interrupt_timerB1      ; 29 タイマB1                ←変更
        .DATA.W H'FFFF                  ; 30
        .DATA.W H'FFFF                  ; 31
        .DATA.W H'FFFF                  ; 32 SCI3_2

;=====================================
; スタートアッププログラム
;=====================================
        .SECTION P
RESET_START:
        MOV.W   #H'FF80,R7              ; スタックポインタの設定
        JSR     @_INITSCT               ; セクションD,R,Bの設定
        JSR     @_main                  ; C言語のmain()関数へジャンプ
LOOP:
        BRA     LOOP

        .END
```

8.6 割り込みの概要

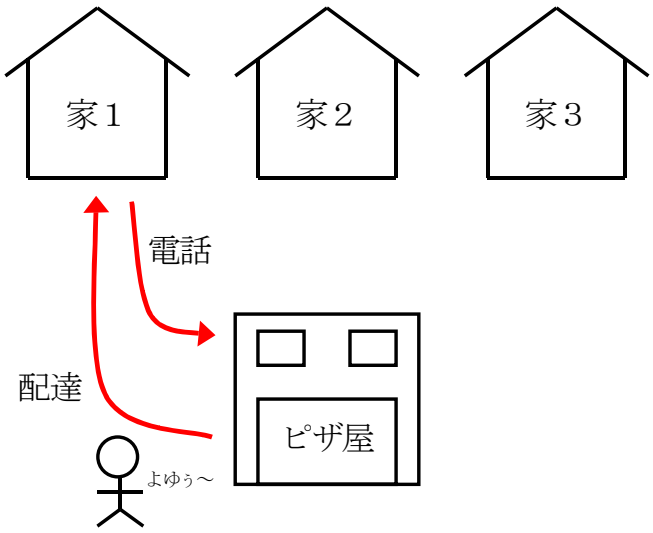
「timer1.c」では、プログラムのループによって時間稼ぎをしていました。「timer2.c」では、割り込みを利用して正確なタイマを作ります。

8.6.1 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。

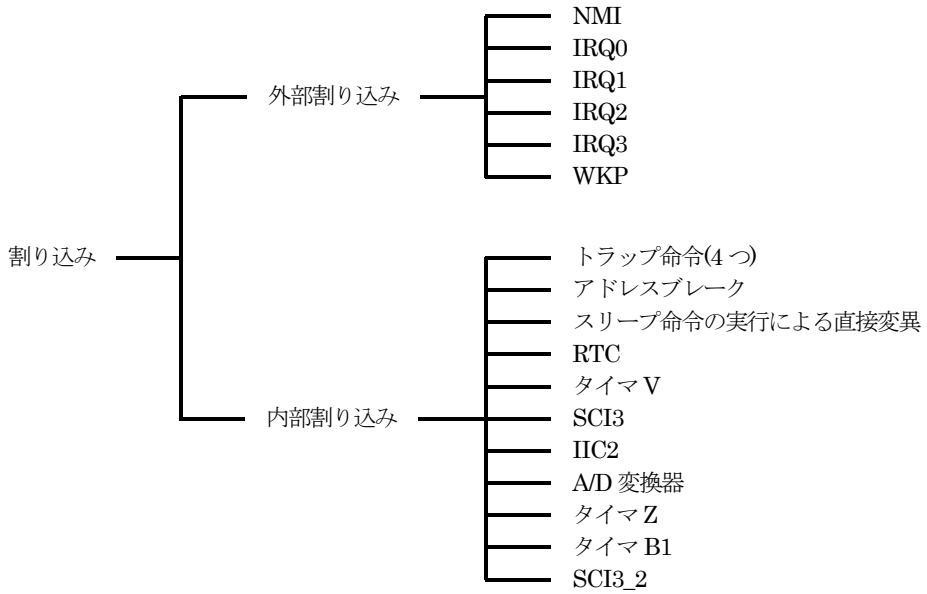
・注文がないか聞きに回る
　制御の用語で「ポーリング」といいます。定期的に監視しなければいけないので、監視する部分が多いと、監視が遅れたり、監視もれが起こります。
・電話で注文をうける
　制御の用語（でもないですが）で「割り込み」といいます。電話のように、きっかけがあったときだけ対処すれば良いので効率が良いです。ただし、電話の用意、電話の受け答えをする必要があります。

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する
ベルが鳴る	→	割り込みが発生する
電話対応する	→	割り込みプログラムを実行する

8.6.2 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 6 つ、内部割り込みは 14 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。
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、SCI3(通信機能)で受信データを受けた場合などです。

8.7 割り込みプログラムの作成方法

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」の設定内容は、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」の設定内容は、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)は、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。

timer2start.src

外部参照

```
.IMPORT _main
.IMPORT _interrupt_timerB1 (6)
```

ベクタアドレス

```
0 番…RESET_START
1 番…設定無し
…
29 番…_interrupt_timerB1 (5)
…
32 番…設定無し
```

スタートアップルーチン

```
RESET_START
    MOV.W   #H'FF80,R7
    JSR     @_INITSCT
    JSR     @_main
```

timer2.c

C言語のプログラム

```
#include "h8_3687.h"

void main( void ) {
    init();
    set_ccr( 0x00 ); (4)
    プログラム
}

#pragma interrupt( interrupt_timerB1 ) (3)
void interrupt_timerB1( void )
{
    プログラム (2)
}

void init( void )
{
    初期設定 (1)
}

その他の関数
```

8.7.1 割り込みを使う設定、割り込みを許可する

H8/3687F の内蔵周辺機能の初期化をする、init 関数内でタイマ B1 の設定を行います。設定は主に、下記の手順で行います。

・どの機能を使うか
・どのきっかけで割り込みをかけるか
・その割り込みを許可

今回は、

・タイマ B1 を使用
・1ms ごとに割り込みを発生させる
・タイマ B1 が割り込みを発生させることを許可

の設定を行います。

```
void init( void )
{
    タイマ B1 を使って 1ms ごとに割り込みが発生するように設定
    タイマ B1 の割り込みを許可
}
```

詳しい内容は、後述します。

8.7.2 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。

割り込みが起こったときに実行する関数名は自由に付けて構いません。この演習では関数名を、割り込みと分かるように「interrupt」、タイマ B1 の割り込みなので「timerB1」とします。これらの名称を合わせて、「interrupt_timerB1」とします。

```
void interrupt_timerB1( void )
{
    1ms ごとに実行するプログラム
}
```

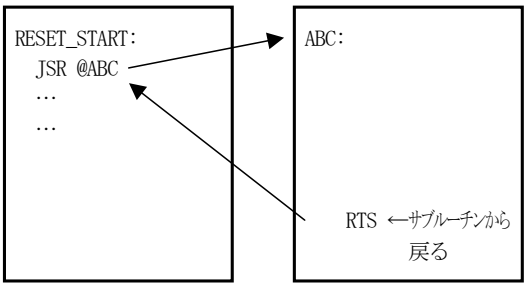
実行するプログラムの内容については後述します。

8.7.3 「#pragma interrupt」の設定

※シャープ　プラグマ　インタラプトと読みます。

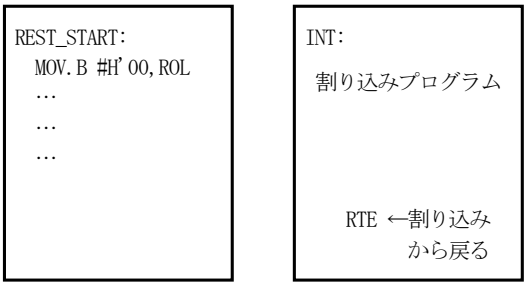
(1) 割り込みプログラム終わりの処理

関数を終えた後、その関数が実行された場所に戻ります。C 言語ソースプログラムは、アセンブリソースプログラムに変換すると下図のようになります。



※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

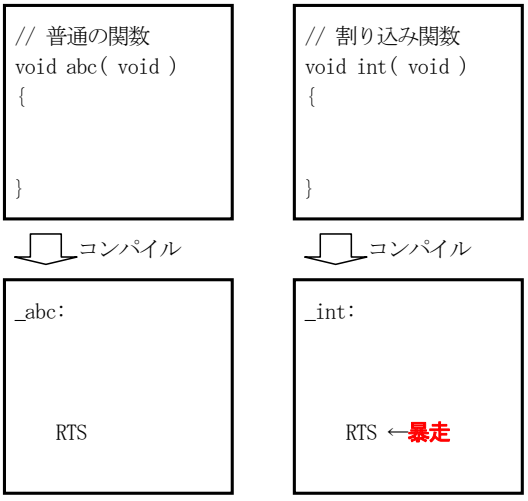
これらは、

・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

宣言の仕方は、下記のようになります。

```
#pragma interrupt( 割り込み関数名 )
```

この宣言は、何処に記述しても良いですが、分かりやすいように割り込み処理する関数の前に記述すると分かりやすくなります。

```
#pragma interrupt( interrupt_timerB1 )
void interrupt_timerB1( void )
{
    1ms ごとに実行するプログラム
}
```

コンパイラは、#pragma interrupt 宣言した関数の終わりは、下記のように「RTE」命令にします。

```
// 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!" );
}
```

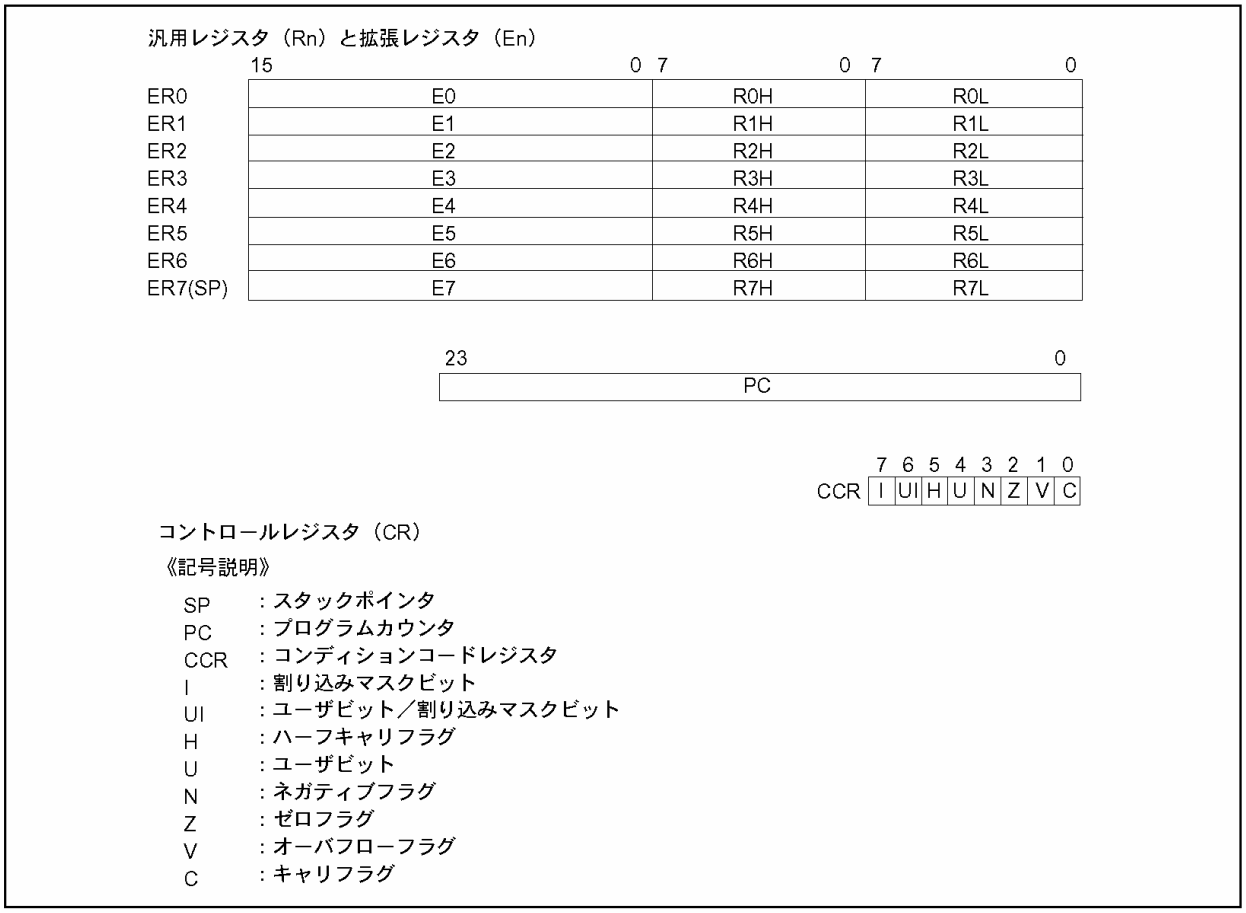
8.7.4 全体の割り込みを許可する

(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 CPU の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。

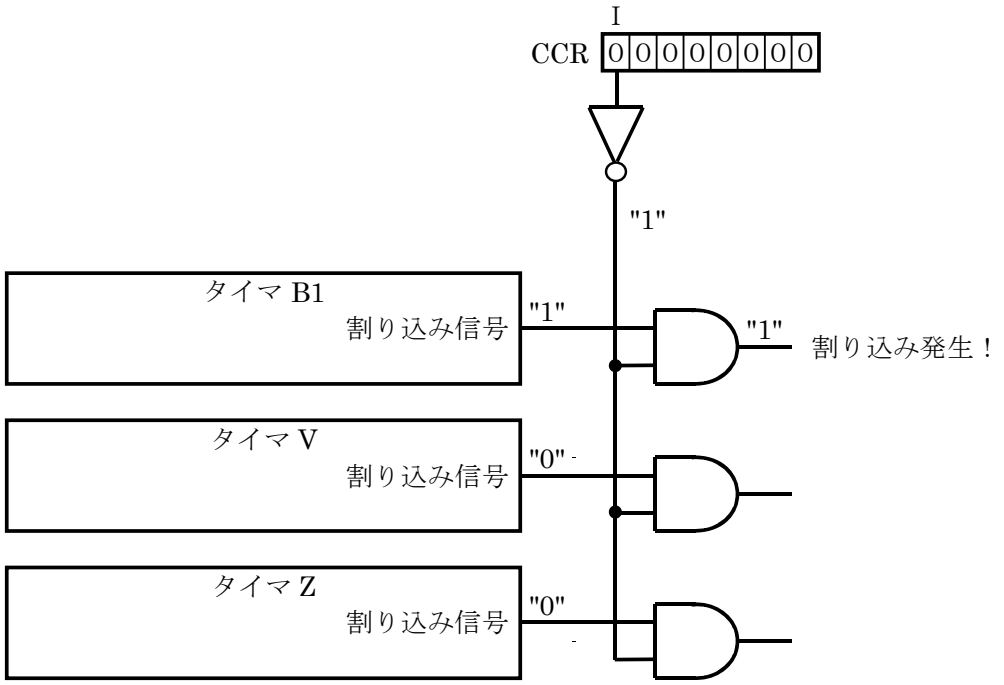
●CCR(コンディションコードレジスタ)の構成内容

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。



etc...

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。

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

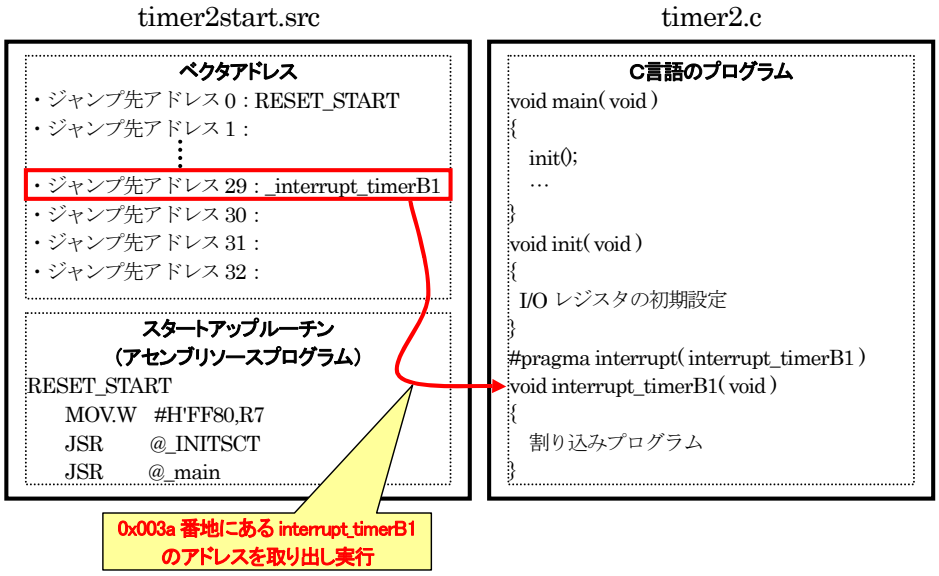
これで CPU 全体の割り込みが許可され、本当に割り込みかかります。内蔵周辺機能(例えばタイマ B1 など)の設定だけでは割り込みは発生しません。

8.7.5 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

タイマ B1 で、1ms ごとに割り込みが発生するように設定しました。
割り込みが発生すると、timer2.c 内にある interrupt_timerB1 関数へ移り、その中のプログラムを実行します。ただし、この関数名は、たまたま分かりやすいように「interrupt_timerB1」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。
「5.3.3 ベクタアドレスからジャンプ先アドレスを取り出す」にあるとおり、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。タイマ B1 は、ベクタ番号 29 です。そのため、ベクタ番号 29 部分に、「interrupt_timerB1 関数のある番地」を記述すればよいのです。



(2) プログラム

timer2start.src ファイル内に記述します。29 番部分に「**_interrupt_timerB1**」を記述すれば、アセンブルするときに interrupt_timerB1 関数のある番地が記述されます。

```
 11 :          .SECTION V
 12 :          .DATA.W RESET_START          ;  0 リセット、WDT
 13 :          .DATA.W H'FFFF               ;  1 システム予約
 14 :          .DATA.W H'FFFF               ;  2 システム予約
(中略)
 39 :          .DATA.W H'FFFF               ; 27 タイマ Z A1-D1
 40 :          .DATA.W H'FFFF               ; 28
 41 :          .DATA.W _interrupt_timerB1   ; 29 タイマ B1
 42 :          .DATA.W H'FFFF               ; 30
 43 :          .DATA.W H'FFFF               ; 31
 44 :          .DATA.W H'FFFF               ; 32 SCI3_2
```

8.7.6 「.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.W _interrupt_timerB1       ; 29 タイマ B1
```

をアセンブルするとき、「_interrupt_timerB1」を探します。しかし、timer2start.src 内にはこのラベルはありません。「_interrupt_timerB1」は、timer2.c ファイル内にあるためです。そのため、「_interrupt_timerB1」は他の場所にあるので、他を探してください、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。

```
 6 :            .IMPORT _interrupt_timerB1
```

という記述で、アセンブラは「_interrupt_timerB1」が他のファイルにあることを理解します。そして「_interrupt_timerB1」というラベル名があれば、予約だけしておきます。その場合、リンケージエディタがリンク時に実際のアドレスを入れます。

(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「__」(アンダーバー)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_timerB1」ですが、アセンブリソースプログラムでは「_interrupt_timerB1」と記述します。

アセンブリソースプログラムの「_interrupt_timerB1」 ＝ C言語ソースプログラムの「interrupt_timerB1」

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！！

8.8 H8/3687Fのタイマ

H8/3687F には、3 種類のタイマがあります。

	タイマ B1	タイマ V	タイマ Z
チャネル数	1	1	2
カウンタのビット数	8 ビット	8 ビット	16 ビット
主な機能	・カウンタのカウントアップは、内部クロック、外部クロックを選択可能 ・カウンタのオーバフローで割り込み発生	・カウンタのカウントアップは、内部クロック、外部クロックを選択可能 ・カウンタのクリアが指定可能（コンペアマッチ A、コンペアマッチ B） ・任意の PWM 出力が可能	・カウンタのカウントアップは、内部クロック、外部クロックを選択可能 ・1 チャネルで 3 つの PWM 出力が可能（競合による問題はプログラムで対応） ・2 チャネルを組み合わせることにより、リセット同期 PWM モードとし、正相、逆相の PWM 波形を 3 つ出力可能 ・2 チャネルを組み合わせることにより、相補 PWM モードとし、3 相モータなど制御可能
実用的な用途	・正確なタイマ ・外部パルスカウント	・正確なタイマ ・外部パルスカウント ・PWM 出力(1 つ)	・正確なタイマ ・外部パルスカウント ・PWM 出力(3 つ) ・3 相モータの制御

3 種類、4 つのタイマを上手く組み合わせて、使用します。マイコンカーでは下記の使い方をしています。

・タイマ B1 を使用して、正確な時間計測を行います。
・タイマ V を使用して、ロータリエンコーダのパルス入力を行います。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。
・タイマ Z を使用して、3 つ分の PWM 出力を行います。サーボ、左モータ、右モータの制御を行います。

8.9 タイマB1 のレジスタ

タイマ B1 に関するレジスタは、下記の 3 レジスタあります。

・タイマモードレジスタ B1（TMB1）
・タイマカウンタ B1（TCB1）
・タイマロードレジスタ B1（TLB1）

また、タイマ B1 の割り込みに関わるレジスタは、下記の 2 レジスタあります。

・割り込みイネーブルレジスタ 2（IENR2）
・割り込みフラグレジスタ 1（IRR1）

8.9.1 タイマモードレジスタB1（TMB1）

TMB1 はオートリロード機能の選択、および入力クロックの選択を行います。

ビット	ビット名	初期値	R/W	説　　明
7	TMB17	0	R/W	オートリロード機能選択 0：インターバル機能を選択 1：オートリロード機能を選択
6 5 4 3	－ － － －	1 1 1 1	－ － － －	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。
2 1 0	TMB12 TMB11 TMB10	0 0 0	R/W R/W R/W	クロックセレクト 000：内部クロックφ/8192 でカウント 001：内部クロックφ/2048 でカウント 010：内部クロックφ/512 でカウント 011：内部クロックφ/256 でカウント 100：内部クロックφ/64 でカウント 101：内部クロックφ/16 でカウント 110：内部クロックφ/4 でカウント 111：外部イベント（TMIB1）の立ち上がりエッジまたは立ち下がりエッジでカウント* 【注】 * 外部イベントのエッジ選択は、割り込みエッジセレクトレジスタ 1（IEGR1）の IEG1 により設定します。詳細は「3.2.1 割り込みエッジセレクトレジスタ 1（IEGR1）」を参照してください。なお TMB12～TMB10 をそれぞれ 1 にセットする前に、必ずポートモードレジスタ 1（PMR1）の IRQ1 を 1 にセットしてください。

8.9.2 タイマカウンタB1（TCB1）

TCB1 は 8 ビットのリード可能なアップカウンタで、入力する内部クロックによりカウントアップされます。入力するクロックは、TMB1 の TMB12～TMB10 により選択します。TCB1 の値は、CPU から常にリードできます。TCB1 がオーバフロー（H'FF→H'00 または H'FF→TLB1 の設定値）すると、IRR2 の IRRTB1 フラグが 1 にセットされます。TCB1 は、TLB1 と同一のアドレスに割り付けられます。TCB1 の初期値は H'00 です。

8.9.3 タイマロードレジスタB1（TLB1）

TLB1 は 8 ビットのライト専用レジスタで、TCB1 のリロード値を設定します。TLB1 にリロード値を設定すると、同時にその値は TCB1 にもロードされ、TCB1 はその値からカウントアップを開始します。またオートリロード動作時に TCB1 がオーバフローすると、TCB1 に TLB1 の値がロードされます。したがって、オーバフロー周期を 1～256 入力クロックの範囲で設定することができます。TLB1 は、TCB1 と同一のアドレスに割り付けられています。TLB1 の初期値は H'00 です。

8.9.4 割り込みイネーブルレジスタ 2（IENR2）

IENR2 はタイマ B1 のオーバフロー割り込みをイネーブルにします。

ビット	ビット名	初期値	R/W	説　　明
7	－	0	－	リザーブビットです。リードすると常に 0 が読み出されます。
6	－	0	－	
5	IENTB1	0	R/W	タイマ B1 割り込み要求イネーブル このビットを 1 にセットするとタイマ B1 のオーバフロー割り込み要求がイネーブルになります。
4	－	1	－	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。
3	－	1	－	
2	－	1	－	
1	－	1	－	
0	－	1	－	

割り込みイネーブルレジスタをクリアすることにより割り込み要求をディスエーブルにする場合、または割り込みフラグレジスタをクリアする場合は、割り込み要求をマスクした状態（I＝1）で行ってください。I＝0 の状態で上記の操作を行うと、命令の実行と当該割り込み要求の発生が競合した場合には、当該操作命令の実行終了時に発生した割り込み要求に対応する例外処理を実行します。

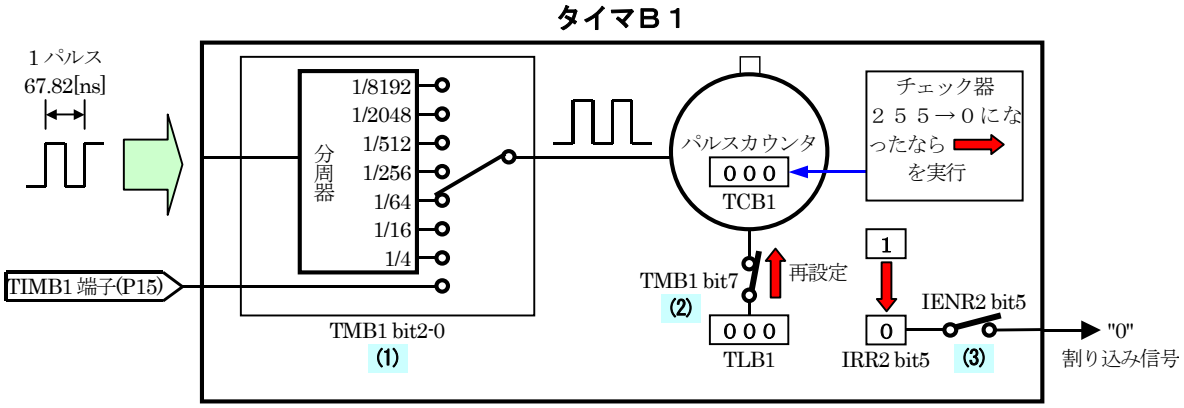
8.9.5 割り込みフラグレジスタ 1（IRR1）

IRR2 はタイマ B1 割り込み要求ステータスフラグレジスタです。

ビット	ビット名	初期値	R/W	説　　明
7	－	0	－	リザーブビットです。リードすると常に 0 が読み出されます。
6	－	0	－	
5	IRRTB1	0	R/W	タイマ B1 割り込み要求フラグ ［セット条件］ タイマ B1 がオーバフローしたとき ［クリア条件］ 0 をライトしたとき
4	－	1	－	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。
3	－	1	－	
2	－	1	－	
1	－	1	－	
0	－	1	－	

8.10 タイマB1 の設定

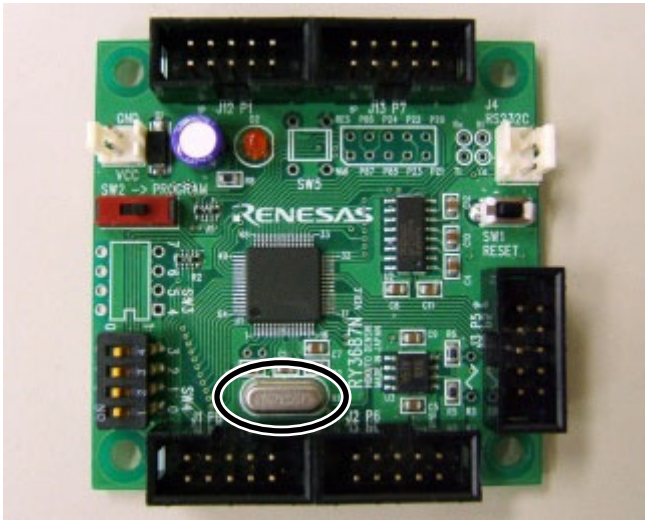
8.10.1 設定手順



(1)パルスカウンタをカウントするタイミングの設定を行います。
(2)オートリロード機能を使うか設定します。
(3)パルスカウンタがオーバフローしたとき、割り込みを発生させるかどうか設定します。

8.10.2 φについて

φとは、クリスタルの周波数のことです。RY3687N ボードには、下記○部分にクリスタルが付いています。



このクリスタルの周波数は、
φ＝14.7456[MHz]＝14.7456×10^6[Hz]
です。1 パルス幅は、周波数の逆数なので、
1 パルス幅＝1/14.7456×10^6≒67.8168[ns] ≒67.82[ns]
となります。この時間が、CPU が動作するすべての基準となります。

8.10.3 パルスカウンタをカウントするタイミングの設定

タイマカウンタ B1(TCB1)は、8 ビットのパルスカウンタです。入力されたパルスを数えます。タイマモードレジスタ B1(TMB1)の bit2～0 を設定することにより、入力するパルスの幅を変えることができます。1 パルス幅は、クリスタルを基準にしており、正確な間隔で送られてきます。そのため、**パルスを数えることは、時間を計ることと同じです**。下記に、設定値と 1 パルス幅の計算をします。また、オーバフロー(255→0 になるとき)するまでの時間も計算しておきます。

TMB1			内容
bit2	bit1	bit0	
0	0	0	φ/8192 でカウント 1 パルス＝1/(14.7456×10^6/8192)＝555.56[μs] 最大計測時間＝555.56×10^{-6}×256＝142.2[ms]
0	0	1	φ/2048 でカウント 1 パルス＝1/(14.7456×10^6/2048)＝138.89[μs] 最大計測時間＝138.89×10^{-6}×256＝35.56[ms]
0	1	0	φ/512 でカウント 1 パルス＝1/(14.7456×10^6/512)＝34.72[μs] 最大計測時間＝34.72×10^{-6}×256＝8.89[ms]
0	1	1	φ/256 でカウント 1 パルス＝1/(14.7456×10^6/256)＝17.36[μs] 最大計測時間＝17.36×10^{-6}×256＝4.44[ms]
1	0	0	φ/64 でカウント 1 パルス＝1/(14.7456×10^6/64)＝4.34[μs] 最大計測時間＝4.34×10^{-6}×256＝1.11[ms]
1	0	1	φ/16 でカウント 1 パルス＝1/(14.7456×10^6/16)＝1.085[μs] 最大計測時間＝1.085×10^{-6}×256＝0.278[ms]
1	1	0	φ/4 でカウント 1 パルス＝1/(14.7456×10^6/4)＝0.271[μs] 最大計測時間＝0.271×10^{-6}×256＝0.069[ms](69.44[μs])
1	1	1	外部端子 TMIB1(P15)のパルスでカウント ※割り込みエッジセレクトレジスタ 1(IEGR1)の IEG1(bit1)=″0″なら 立ち下がりエッジでカウント、″1″なら立ち上がりエッジでカウント

今回は、1ms ごとに割り込みをかけます。最大計測時間は 1ms 以上である必要があります。最大計測時間が短い順に最大計測時間を見ていきます。1ms 以下の場合は、計ることができませんので設定できません。一番最初に 1ms 以上になったときが、設定する値です。

・φ/4 のとき、最大計測時間は 0.069[ms] → 1ms 以下なので不可
・φ/16 とき、最大計測時間は 0.278[ms] → 1ms 以下なので不可
・φ/64 のとき、最大計測時間は 1.11[ms] → 1ms 以上なので適合

よって、タイマモードレジスタ B1(TMB1) bit2～0 の設定は、φ/64 である″100″にします。

タイマモードレジスタ B1(TMB1)							
7	6	5	4	3	2	1	0
オートリロード 機能選択					クロックセレクト 2～0		
					1	**0**	**0**

8.10.4 オートリロード機能を使うかの設定

パルスカウンタであるタイマカウンタ B1(TCB1)がオーバフローして、255 から 0 になったとき、割り込みを発生させることができます(割り込みについて詳しくは後述します)。ということは、256 カウントごとに割り込みがかかることになります。

先に計算したとおり、タイマモードレジスタ B1(TMB1)の bit2～0 を"100"に設定すると下記のようになりました。

1 パルス＝1/(14.7456×10^6/64)＝4.34[μs]
256 カウントする間での時間(最大計測時間)＝4.34×10^{-6}×256＝1.11[ms]

今回は 1ms ごとに割り込みをかけたいのですが、これでは 1.11[ms]という中途半端な間隔でしか割り込みをかけることができません。そこで、オートリロードという機能を使います。

通常、タイマカウンタ B1(TCB1)が 255 になった次は 0 です。オートリロード機能とは、255 の次を 0 ではなく指定した値にすることを言います。

割り込み間隔を 1[ms]にしたい場合、255 の次は何にすれば良いのでしょうか。

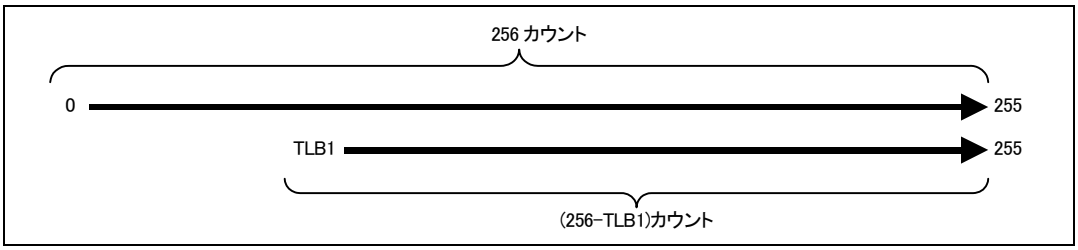
1 パルスは 4.34[μs]ですので、1ms になるまでのカウント数は

1ms のカウント数＝(1×10^{-3})÷(4.34×10^{-6})＝230.4≒230

よって、230 カウントさせれば良いことになります。そのため、最初の数値は

256-230＝26

となります。この数値をタイマロードレジスタ B1(TLB1)というレジスタに設定することにより、タイマカウンタ B1(TCB1)がオーバフローしたとき 0 ではなく、設定した数値にすることができます。図解すると下記のようなります。



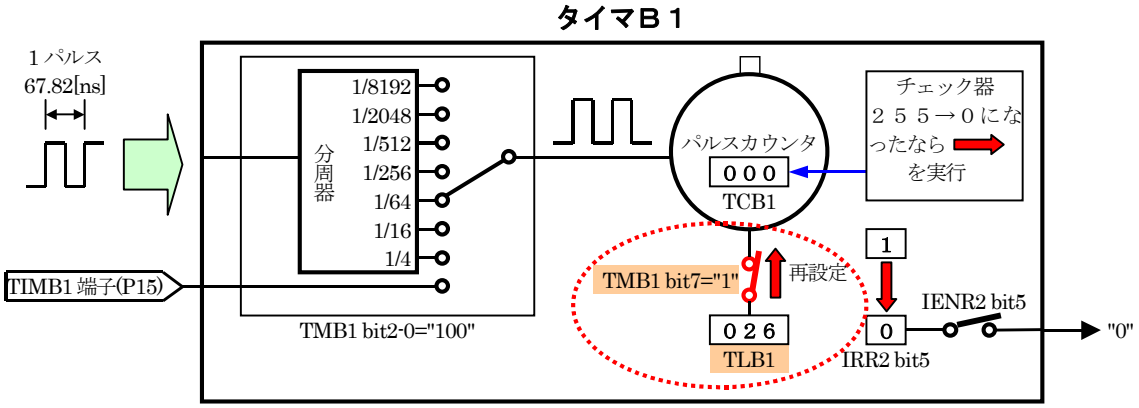
プログラムは、下記のようになります。

```
TLB1 = 26;
```

そして最後に、「タイマロードレジスタ B1(TLB1)を使います」という設定を行います。それがタイマモードレジスタ B1(TMB1)の bit7 です。

TMB1 の bit7=**"1":オートリロード機能を選択**

"0":インターバル機能を選択(インターバル機能とは、オートリロード機能を使わないことです)



パルス幅の設定は、bit2-0="100"でした。今回の設定と合わせてタイマモードレジスタ B1(TMB1)を設定します。意味のないビットは、"0"にしておきます。

タイマモードレジスタ B1(TMB1)							
7	6	5	4	3	2	1	0
オートリロード機能選択					クロックセレクト 2～0		
1					**1**	**0**	**0**

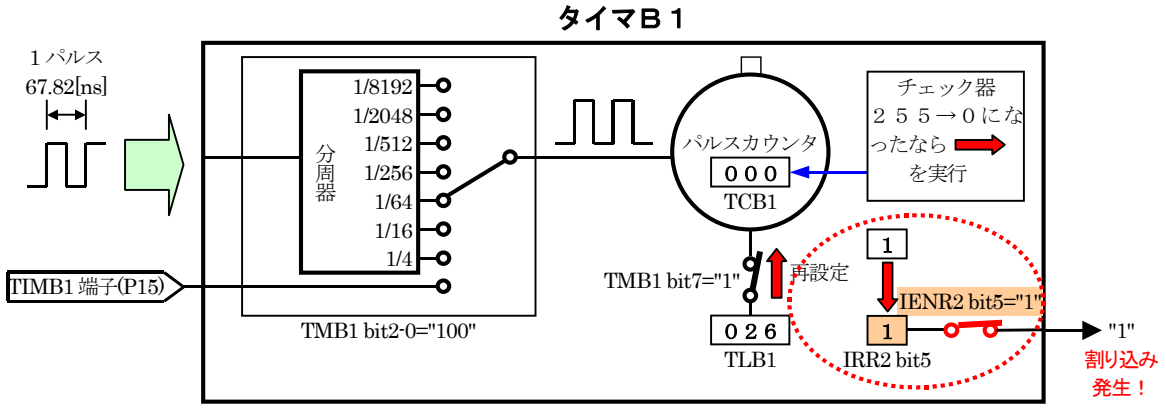
プログラムは下記のようになります。

```
TMB1 = 0x84;              /* 1??? ?100 */   ?は特に意味はなし、よって 0 にする
```

8.10.5 パルスカウンタがオーバフローしたとき、割り込みを発生させる設定

先の設定で、タイマカウンタ B1(TCB1)がオーバフローする間隔は、1[ms]ごとになりました。このときに、割り込みをかければ、1ms ごとに割り込みがかかることになります。

タイマカウンタ B1(TCB1)がオーバフローしたとき、割り込みフラグレジスタ 2(IRR2)の bit5 が"1"になります。このことを検出して、割り込みを発生させることができます。割り込みイネーブルレジスタ 2(IENR2)の bit5 がスイッチのような役割をしています。下記は、割り込みイネーブルレジスタ 2(IENR2) の bit5="1"のとき、タイマカウンタ B1(TCB1)が 255 から 0 にオーバフローした瞬間です。



割り込みイネーブルレジスタ 2(IENR2)							
7	6	5	4	3	2	1	0
		タイマ B1 割り込み要求イネーブル					
		1					

プログラムは下記のようになります。

```
IENR2 = 0x20;     /* ??1? ???? */   ?は特に意味はなし、よって 0 にする
```

8.10.6 まとめ

init 関数内で、タイマ B1 に関する 3 つのレジスタの設定を行います。

```
TMB1 = 0x84;                    /* 入力クロックの設定等      */
TLB1 = 26;                      /* カウンタ初期値設定        */
IENR2 = 0x20;                   /* 割り込み要求許可          */
```

8.11 動作

8.11.1 設定内容

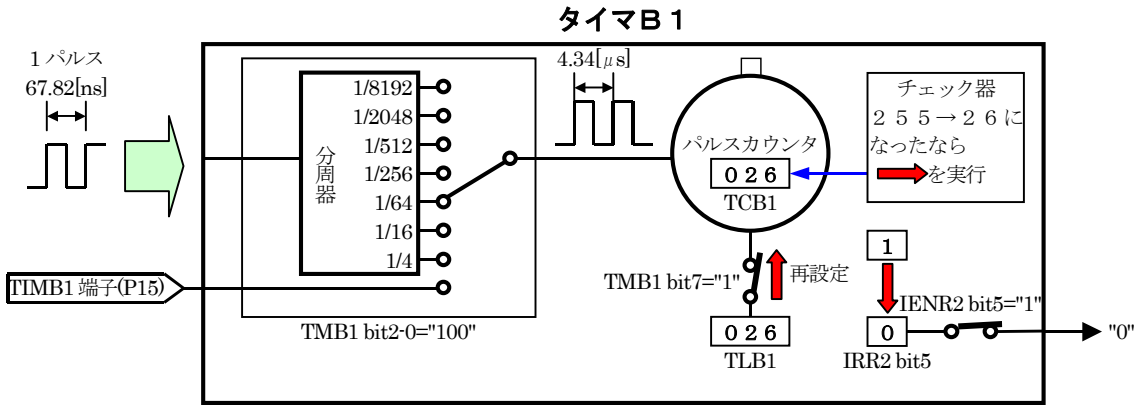
```
63 :      /* タイマ B1 1ms1 ごとの割り込み設定 */
64 :      TMB1 = 0x84;                    /* 入力クロックの設定等      */
65 :      TLB1 = 26;                      /* カウンタ初期値設定        */
66 :      IENR2 = 0x20;                   /* 割り込み要求許可          */
```

上記設定により、

・タイマカウンタ B1(TCB1)の値が増える間隔は、4.34[μs]です。

・タイマカウンタ B1(TCB1)の初期値は、タイマロードレジスタ B1(TLB1)に設定した 26 です。

・タイマカウンタ B1(TCB1)がオーバフローすると、割り込みが発生します。



8.11.2 オーバフロー（割り込み発生）

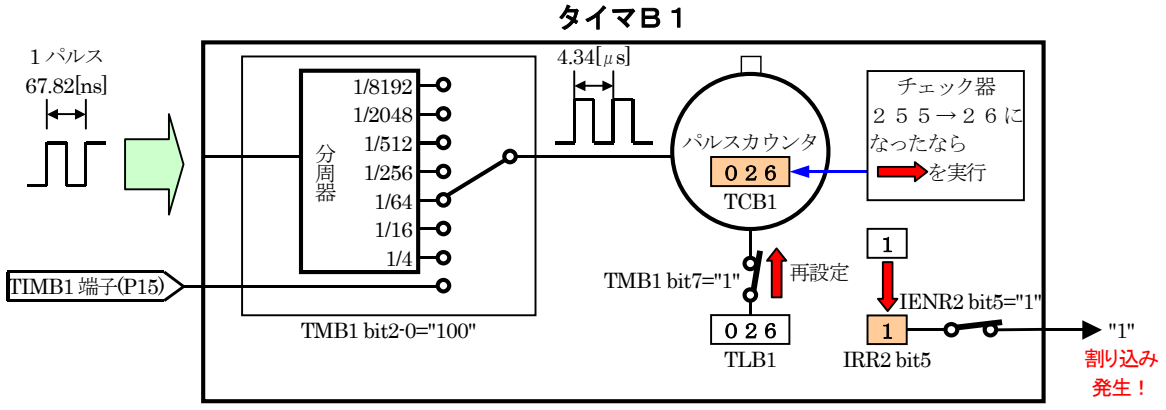
タイマカウンタ B1(TCB1)が 255 から 1 つ増えてオーバフローします。タイマカウンタ B1(TCB1)は 26 がセットされます。そして、割り込みが発生します。

割り込み間隔は、

26 から 256 になるまでのカウント数は、 256-26＝230

1 カウント、4.34[μs]なので、$4.34\times10^{-6}\times230=0.998$[ms]≒**1[ms]**

となります。

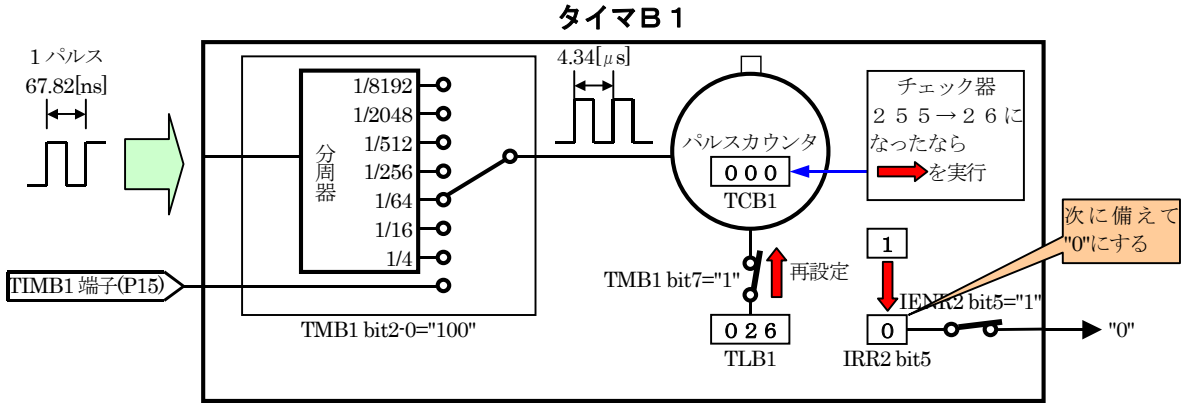


8.11.3 割り込みプログラム

interrupt_timerB1 関数が、1ms ごとに実行されます。

```
 83 :  #pragma interrupt( interrupt_timerB1 )
 84 :  void interrupt_timerB1( void )
 85 :  {
 86 :      IRR2 &= 0xdf;                       /* フラグクリア             */
 87 :      cnt0++;
 88 :  }
```

まず、86 行で割り込みフラグレジスタ 2(IRR2)の bit5 をクリアしています。このビットは、タイマカウンタ B1 (TCB1)がオーバフローしたときに"1"になるビットです。このビットは自動的に"0"になりません、そのため、次のオーバフローに備えて、プログラマが"0"にする必要があります。そのため、この部分で"0"にしています。"0"にするときは、bit5 のみ処理するため、AND 演算しています。



その後、変数 cnt0 を1つ増やします。この関数は 1ms ごとに実行されますので、変数 cnt0 は 1ms ごとに増えることになります。

8.12 プログラムの解説

8.12.1 main関数

```
 31 :  void main( void )
 32 :  {
 33 :      init();                              /* マイコン機能の初期化      */
 34 :      set_ccr( 0x00 );                     /* 全体割り込み許可          */
 35 :
 36 :      while( 1 ) {
 37 :          PDR6 = 0x55;
 38 :          timer( 1000 );
 39 :          PDR6 = 0xaa;
 40 :          timer( 1000 );
 41 :          PDR6 = 0x00;
 42 :          timer( 1000 );
 43 :      }
 44 :  }
```

33 行で init 関数を呼んでポートの入出力設定、タイマ B1 の設定を行います。
34 行で全体の割り込みを許可しています。
36 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 43 行まで無限ループです。
37 行でポートデータレジスタ 6(PDR6)に 0x55 を出力、1000 ミリ秒時間稼ぎします。
39 行でポートデータレジスタ 6(PDR6)に 0xaa を出力、1000 ミリ秒時間稼ぎします。
41 行でポートデータレジスタ 6(PDR6)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。

8.12.2 timer関数

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

76 行で、cnt0 変数を 0 にしています。
77 行で、cnt0 変数が timer_set 変数より小さいか比較します。小さいなら 77 行を繰り返します。同じ値になるか、または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。

8.13 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 関数内でタイマ B1 のレジスタを設定します。

設定するレジスタ	詳細
タイマモード レジスタ B1 TMB1	オートリロード機能の選択、クロックのセレクト 0x80→555.56[μs]でカウント　0x81→138.89[μs]でカウント 0x82→34.72[μs]でカウント　0x83→17.36[μs]でカウント 0x84→4.34[μs]でカウント　0x85→1.085[μs]でカウント 0x86→0.271[μs]でカウント　0x87→外部端子 P15 でカウント
タイマロード レジスタ B1 TLB1	タイマカウンタ B1(TCB1)がオーバフローしたときに、設定する値 TMB1=0x84 のとき、1ms ごとに割り込みを発生させたいなら、 $(1\times10^{-3})/(4.34\times10^{-6})=230.4\fallingdotseq230$ ∴TLB1=256-230=26
割り込み イネーブルレジスタ 2 IENR2	割り込みを許可します。0x20 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_timerB1 関数を追加します。

```
void interrupt_timerB1( void )
{
    IRR2 &= 0xdf;                          ←フラグのクリアを必ず行う
    cnt0++;                                ←割り込みプログラム（今回は 1 行）
}
```

(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_timerB1 )     ←割り込みプログラムであることを宣言する
void interrupt_timerB1( void )
{
    IRR2 &= 0xdf;                          ←フラグのクリアを必ず行う
    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
    set_ccr( 0x00 );                           ←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。今回はタイマ B1 割り込みが発生するので、ベクタアドレスの表より 29 番部分(0x003a 番地)に「_interrupt_timerB1」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。

```
 41 :           .DATA.W _interrupt_timerB1       ; 29 タイマ B1
```

(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_timerB1       ; 外部参照
```

9. プロジェクト「timer3」 メイン処理をしながらLED点滅処理

9.1 概要

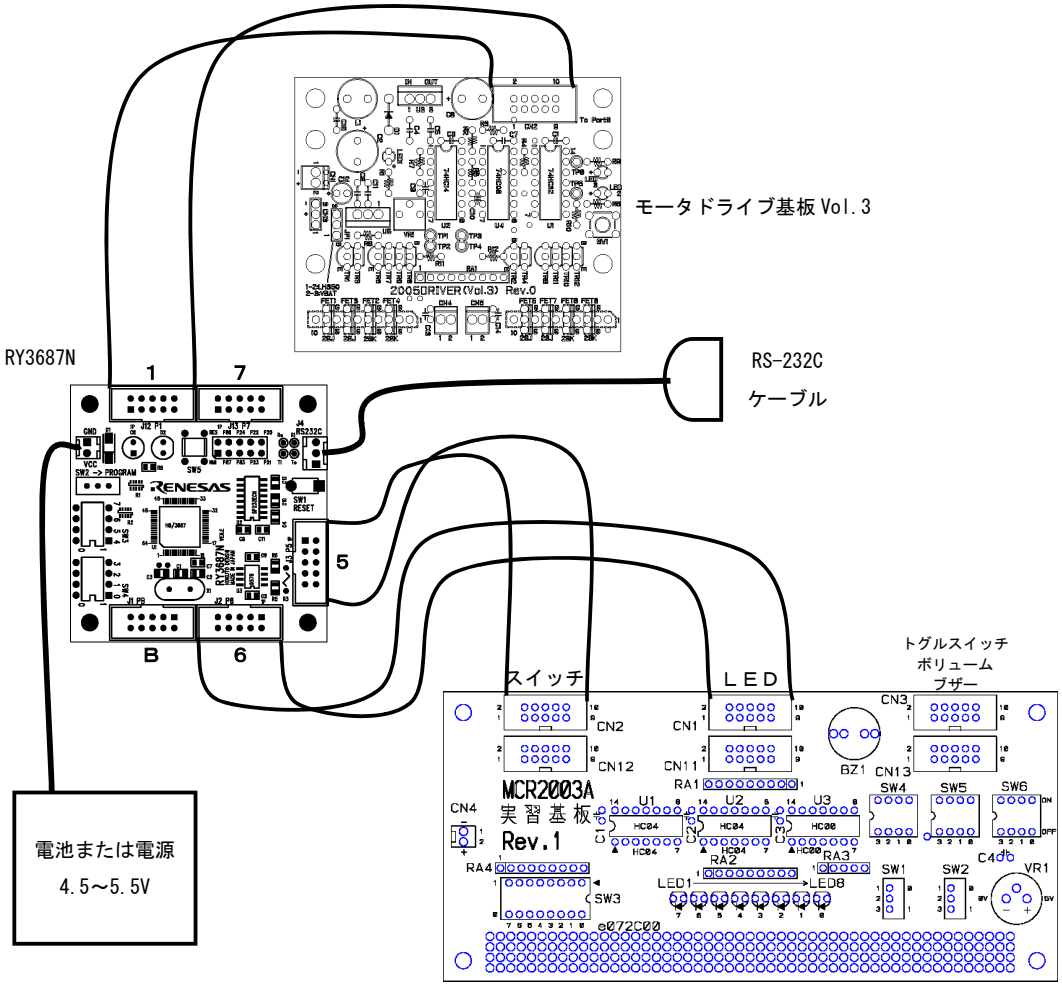
timer2.c を応用して、下記の動作を行ってみましょう。
・メインプログラムでは、ポート 5 に接続されているディップスイッチの値をポート 6 の LED へ出力
・割り込みプログラムでは、ポート1に接続されているモータドライブ基板のLED 2個を0.5秒間隔で交互に点滅

マイコンのポートは、下記を使用します。
・ポート 1・・・モータドライブ基板 Vol.3 と接続（使用するのは LED 2 個）
・ポート 5 の全ビット・・・ディップスイッチの状態を入力
・ポート 6 の全ビット・・・LED へデータ出力

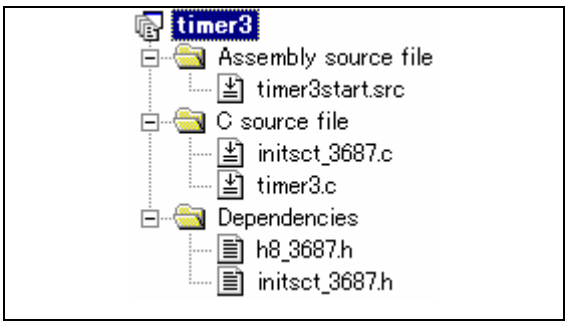
timer2.c を応用すれば改造できるプログラムなので、最初は各自で改造してみましょう。完成したら timer3.c と見比べてみましょう。

9.2 接続

・CPU ボードのポート 5 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 6 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 1 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。



9.3 プロジェクトの構成



	ファイル名	内容
1	timer3start.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_3687.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	timer3.c	実際に制御するプログラムが書かれています。H8/3687F の内蔵周辺機能の初期化も行います。
4	h8_3687.h	H8/3687F 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_3687.h	initsct_3687.c のヘッダファイルです。

9.4 プログラム「timer3.c」

太字が、timer2.c と比べて追加したところや変更したところです。

```
 1 :  /*********************************************************************/
 2 :  /* メイン処理をしながらLED点滅処理 "timer3.c"                        */
 3 :  /*                          2007.04 ジャパンマイコンカーラリー実行委員会 */
 4 :  /*********************************************************************/
 5 :  /*
 6 :  出力：P17,P16 モータドライブ基板Vol.3のLED
 7 :  入力：P57-P50(ディップスイッチなど)
 8 :  出力：P67-P60(LEDなど)
 9 :
10 :  メインプログラムでは、ポート5に接続されているディップスイッチの値を
11 :  ポート6のLEDへ出力します。
12 :  割り込みでは、ポート1に接続されているモータドライブ基板のLED 2個を
13 :  0.5秒間隔で交互に点滅させます。
14 :  */
15 :
16 :  /*====================================*/
17 :  /* インクルード                       */
18 :  /*====================================*/
19 :  #include    <machine.h>
20 :  #include    "h8_3687.h"
21 :
22 :  /*====================================*/
23 :  /* プロトタイプ宣言                   */
24 :  /*====================================*/
25 :  void init( void );
26 :  void timer( unsigned long timer_set );
27 :
28 :  /*====================================*/
29 :  /* グローバル変数の宣言               */
30 :  /*====================================*/
31 :  unsigned long cnt0;                    /* タイマB1                 */
32 :  unsigned long cnt1;                    /* 割り込み内ＬＥＤ制御用   */
```

```
 33 :
 34 : /*********************************************************************/
 35 : /* メインプログラム                                                  */
 36 : /*********************************************************************/
 37 : void main( void )
 38 : {
 39 :     init();                             /* マイコン機能の初期化     */
 40 :     set_ccr( 0x00 );                    /* 全体割り込み許可         */
 41 :
 42 :     while( 1 ) {
 43 :         PDR6 = PDR5;
 44 :     }
 45 : }
 46 :
 47 : /*********************************************************************/
 48 : /* H8/3687F 内蔵周辺機能  初期化                                     */
 49 : /*********************************************************************/
 50 : void init( void )
 51 : {
 52 :     /* I/Oポートの入出力設定 */
 53 :     PCR1 = 0xfe;                        /* モータドライブ基板追加   */
 54 :     PCR2 = 0xfd;                        /* 通信ビットP22:TxD P21:RxD*/
 55 :     PCR3 = 0xf0;                        /* 基板上のディップスイッチ */
 56 :     PCR5 = 0x00;                        /* ディップスイッチ追加     */
 57 :     PCR6 = 0xff;                        /* LED基板                  */
 58 :     PCR7 = 0xff;
 59 :     PCR8 = 0xff;
 60 :     /* ポートBは、入力専用なので入出力設定はありません。          */
 61 :     /* また、A/D変換器のADCSRによって指定された1ビットだけは、  */
 62 :     /* 入力ポートとしては使えません。                            */
 63 :
 64 :     /* タイマB1 1msごとの割り込み設定 */
 65 :     TMB1 = 0x84;                        /* 入力クロックの設定等     */
 66 :     TLB1 = 26;                          /* カウンタ初期値設定       */
 67 :     IENR2 = 0x20;                       /* 割り込み要求許可         */
 68 : }
 69 :
 70 : /*********************************************************************/
 71 : /* タイマ本体                                                        */
 72 : /* 引数   タイマ値 1=1ms                                             */
 73 : /* 戻り値 なし                                                       */
 74 : /*********************************************************************/
 75 : void timer( unsigned long timer_set )
 76 : {
 77 :     cnt0 = 0;
 78 :     while( cnt0 < timer_set );
 79 : }
 80 :
 81 : /*********************************************************************/
 82 : /* タイマB1 割り込み処理                                             */
 83 : /*********************************************************************/
 84 : #pragma interrupt( interrupt_timerB1 )
 85 : void interrupt_timerB1( void )
 86 : {
 87 :     IRR2 &= 0xdf;                       /* フラグクリア             */
 88 :     cnt0++;
 89 :
 90 :     /* LED点滅処理 */
 91 :     cnt1++;
 92 :     if( cnt1 <= 500 ) {
 93 :         PDR1 = 0x80;
 94 :     } else if( cnt1 <= 1000 ) {
 95 :         PDR1 = 0x40;
 96 :     } else {
 97 :         cnt1 = 0;
 98 :     }
 99 : }
100 :
101 : /*********************************************************************/
102 : /* End of file                                                       */
103 : /*********************************************************************/
```

9.5 プログラムの解説

9.5.1 ポートの接続

ポート 1 とモータドライブ基板 Vol.3 が接続されています。各ビットの接続内容は下記のようです。

bit	7	6	5	4	3	2	1	0
内容	**LED1 出力 "0":点灯 "1"消灯**	**LED0 出力 "0":点灯 "1"消灯**	サーボ出力	右モータ出力	右モータ出力	左モータ出力	左モータ出力	スイッチ入力

ポート 1 の入出力設定は、bit0 は入力、他のビットは出力となります。

```
PCR1 = 0xfe;
```

LED1 を点灯、LED0 を消灯させるには、bit7="0"、bit6="1"にします。他のビットは"0"にします。

```
PDR1 = 0x40;
```

LED1 を消灯、LED0 を点灯させるには、bit7="1"、bit6="0"にします。他のビットは"0"にします。

```
PDR1 = 0x80;
```

よって、LED を交互に点滅させるには PDR1 に 0x80 と 0x40 を、0.5 秒ごとに設定すれば良いことになります。

9.5.2 main関数

```
37 :  void main( void )
38 :  {
39 :      init();                          /* マイコン機能の初期化      */
40 :      set_ccr( 0x00 );                 /* 全体割り込み許可          */
41 :
42 :      while( 1 ) {
43 :          PDR6 = PDR5;
44 :      }
45 :  }
```

main 関数内では、ポート 5 の値をポート 6 に出力しているだけです。モータドライブ基板 Vol.3 の LED 制御は一切行っていません。

9.5.3 割り込みプログラム

```
84 :  #pragma interrupt( interrupt_timerB1 )
85 :  void interrupt_timerB1( void )
86 :  {
87 :      IRR2 &= 0xdf;                       /* フラグクリア             */
88 :      cnt0++;
89 :
90 :      /* LED 点滅処理 */
91 :      cnt1++;
92 :      if( cnt1 <= 500 ) {
93 :          PDR1 = 0x80;
94 :      } else if( cnt1 <= 1000 ) {
95 :          PDR1 = 0x40;
96 :      } else {
97 :          cnt1 = 0;
98 :      }
99 :  }
```

割り込みプログラムで、PDR1 の値を操作して、モータドライブ基板の LED の点滅処理を行っています。割り込みは 1ms ごとに実行されることが分かっているので、cnt1 変数で回数を数えることにより、0.5 秒ごとに LED を点滅させています。

10. プロジェクト「ad」 A/D変換値をLEDへ出力

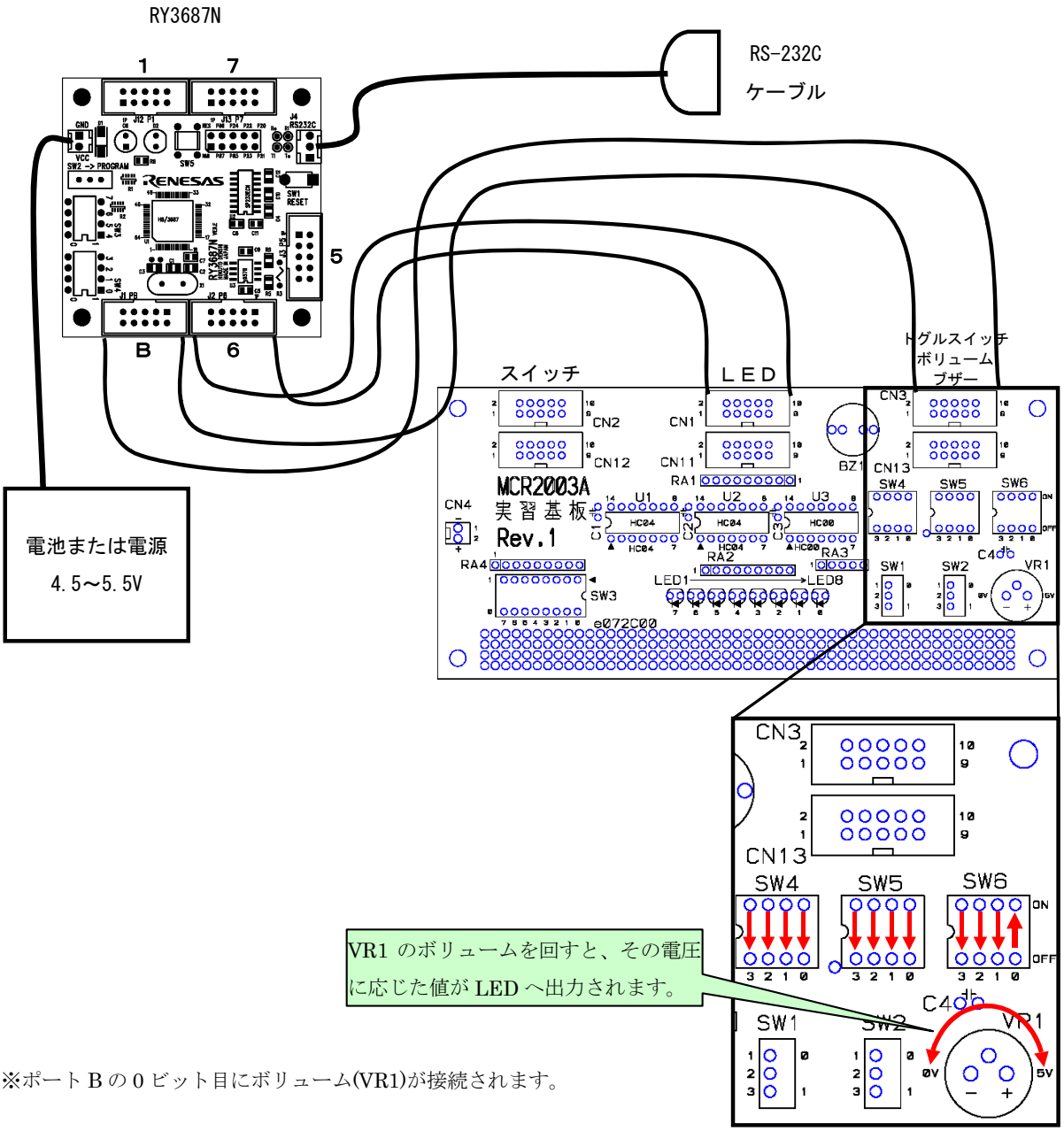
10.1 概要

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。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 B の 0 ビット・・・アナログ電圧(0～5V)入力
・ポート 6 の全ビット・・・LED へデータ出力

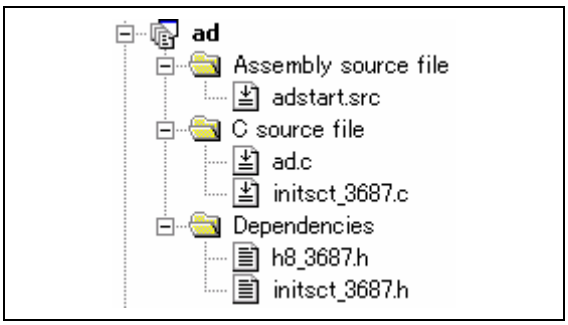
10.2 接続

・CPU ボードのポート B 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 6 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 B の 0 ビット目にボリューム(VR1)が接続されます。

10.3 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_3687.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。H8/3687F の内蔵周辺機能の初期化も行います。
4	h8_3687.h	H8/3687F 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_3687.h	initsct_3687.c のヘッダファイルです。

10.4 プログラム「ad.c」

```
 1 :  /*********************************************************************/
 2 :  /* 電圧入力 "ad.c"                                                   */
 3 :  /*                          2007.04 ジャパンマイコンカーラリー実行委員会 */
 4 :  /*********************************************************************/
 5 :  /*
 6 :  入力：PB0(0～5V)
 7 :  出力：P67-P60(LEDなど)
 8 :
 9 :  ポートBのbit0に入力した電圧(0～5V)を、ポート６に出力します。
10 :  */
11 :
12 :  /*====================================*/
13 :  /* インクルード                       */
14 :  /*====================================*/
15 :  #include    <machine.h>
16 :  #include    "h8_3687.h"
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 :
28 :  /*********************************************************************/
29 :  /* メインプログラム                                                  */
30 :  /*********************************************************************/
31 :  void main( void )
32 :  {
33 :      int ad;
34 :
35 :      init();                             /* マイコン機能の初期化      */
36 :
```

```
 37 :      while( 1 ) {
 38 :          ad = get_ad();
 39 :          PDR6 = ad;
 40 :      }
 41 :  }
 42 :
 43 :  /************************************************************************/
 44 :  /* H8/3687F 内蔵周辺機能　初期化                                        */
 45 :  /************************************************************************/
 46 :  void init( void )
 47 :  {
 48 :      /* I/Oポートの入出力設定 */
 49 :      PCR1 = 0xff;
 50 :      PCR2 = 0xfd;                        /* 通信ビットP22:TxD P21:RxD*/
 51 :      PCR3 = 0xf0;                        /* 基板上のディップスイッチ */
 52 :      PCR5 = 0xff;
 53 :      PCR6 = 0xff;                        /* LED基板                  */
 54 :      PCR7 = 0xff;
 55 :      PCR8 = 0xff;
 56 :      /* ポートBは、入力専用なので入出力設定はありません。          */
 57 :      /* また、A/D変換器のADCSRによって指定された1ビットだけは、   */
 58 :      /* 入力ポートとしては使えません。                             */
 59 :
 60 :      /* A/Dの初期設定 */
 61 :      ADCSR = 0x00;
 62 :  }
 63 :
 64 :  /************************************************************************/
 65 :  /* A/D値読み込み(AN0)                                                   */
 66 :  /* 引数　 なし                                                          */
 67 :  /* 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     */
 68 :  /************************************************************************/
 69 :  int get_ad( void )
 70 :  {
 71 :      int i;
 72 :
 73 :      ADCSR |= 0x20;                      /* ADスタート               */
 74 :      while( !(ADCSR & 0x80) );           /* エンドフラグをチェック   */
 75 :      ADCSR &= 0x7f;                      /* エンドフラグクリア       */
 76 :      i = ADDRA >> 8;                     /* 代入                     */
 77 :
 78 :      return i;
 79 :  }
 80 :
 81 :  /************************************************************************/
 82 :  /* End of file                                                          */
 83 :  /************************************************************************/
```

10.5 A/D変換器のレジスタ

A/D 変換器に関するレジスタは、下記の 6 レジスタあります。

・A/D データレジスタ A(ADDRA)
・A/D データレジスタ B(ADDRB)
・A/D データレジスタ C(ADDRC)
・A/D データレジスタ D(ADDRD)
・A/D コントロール／ステータスレジスタ(ADCSR)
・A/D コントロールレジスタ(ADCR)

10.5.1 A/Dデータレジスタ A～D（ADDRA～D）

A/D データレジスタは A/D 変換結果を格納するための 16 ビットのリード専用レジスタで、ADDRA～ADDRD の 4 本あります。各アナログ入力チャネルの変換結果が格納される A/D データレジスタは下表のとおりです。10 ビットの変換データは A/D データレジスタのビット 15 からビット 6 に格納されます。下位 6 ビットの読み出し値は常に 0 です。CPU との間のデータバスは 8 ビット幅で、上位バイトは CPU から直接リードできますが、下位バイトは上位バイトリード時にテンポラリレジスタに転送されたデータが読み出されます。このため A/D データレジスタをリードする場合は、ワードアクセスするか、バイトアクセス時は上位バイト、下位バイトの順でリードしてください。ADDR の初期値は H'0000 です。

アナログ入力チャネル		変換結果が格納される
グループ 0	グループ 1	A/D データレジスタ
AN0	AN4	ADDRA
AN1	AN5	ADDRB
AN2	AN6	ADDRC
AN3	AN7	ADDRD

10.5.2 A/Dコントロール／ステータスレジスタ（ADCSR）

ADCSR は A/D 変換器の制御ビットと変換終了ステータスビットで構成されています。

ビット	ビット名	初期値	R/W	説　　明
7	ADF	0	R/W	A/D エンドフラグ ［セット条件］ • 単一モードで A/D 変換が終了したとき • スキャンモードで選択されたすべてのチャネルの変換が 1 回終了したとき ［クリア条件］ • 1 の状態をリードした後、0 をライトしたとき
6	ADIE	0	R/W	A/D インタラプトイネーブル このビットを 1 にセットすると ADF による A/D 変換終了割り込み要求（ADI）がイネーブルになります。
5	ADST	0	R/W	A/D スタート このビットを 1 にセットすると A/D 変換を開始します。単一モードでは A/D 変換を終了すると自動的にクリアされます。スキャンモードではソフトウェア、リセット、またはスタンバイモードによってクリアされるまで選択されたチャネルを順次連続変換します。
4	SCAN	0	R/W	スキャンモード A/D 変換のモードを選択します。 0：単一モード 1：スキャンモード
3	CKS	0	R/W	クロックセレクト A/D 変換時間の設定を行います。 0：変換時間＝134 ステート（max） 1：変換時間＝70 ステート（max） 変換時間の切換えは、ADST＝0 の状態で行ってください。

ビット	ビット名	初期値	R/W	説　　明
2	CH2	0	R/W	チャネルセレクト 2～0
1	CH1	0	R/W	アナログ入力チャネルを選択します。
0	CH0	0	R/W	SCAN＝0 のとき　　SCAN＝1 のとき 000：AN0　　000：AN0 001：AN1　　001：AN0～AN1 010：AN2　　010：AN0～AN2 011：AN3　　011：AN0～AN3 100：AN4　　100：AN4 101：AN5　　101：AN4～AN5 110：AN6　　110：AN4～AN6 111：AN7　　111：AN4～AN7

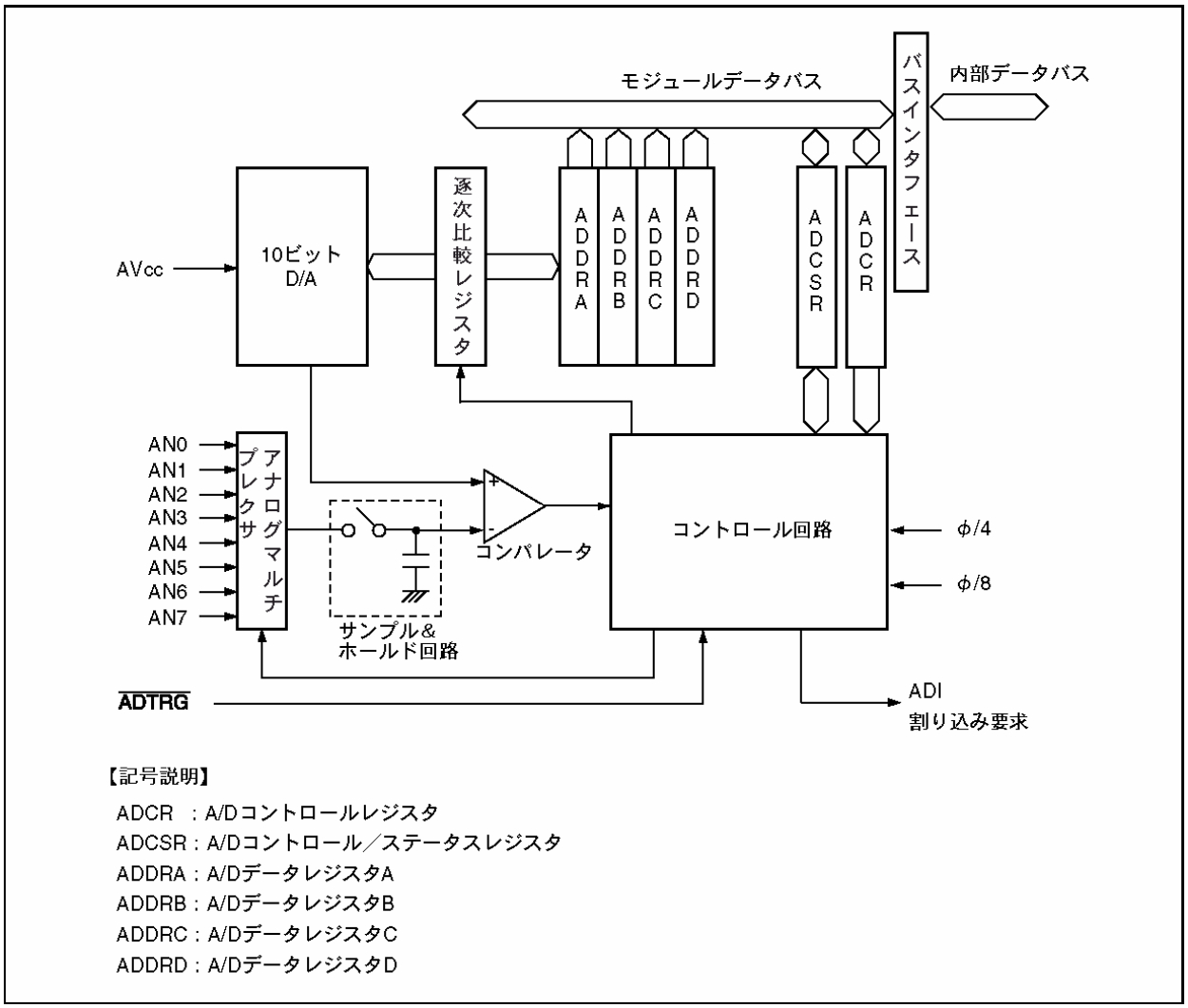
10.5.3 A/Dコントロールレジスタ（ADCR）

ADCR は外部トリガによる A/D 変換開始をイネーブルにします。

ビット	ビット名	初期値	R/W	説　　明
7	TRGE	0	R/W	トリガイネーブル このビットを 1 にセットすると外部トリガ端子（$\overline{\text{ADTRG}}$）の立ち上がり、立ち下がりエッジでも A/D 変換を開始します。 外部トリガ端子（$\overline{\text{ADTRG}}$）の立ち上がり、立ち下がりエッジ選択は割り込みエッジセレクトレジスタ 2（IEGR2）の WPEG5 の設定に従います。
6～1	－	すべて 1	－	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。
0	－	0	R/W	リザーブビットです。リード／ライト可能ですが、1 に設定しないでください。

10.6 A/D変換器の設定(単一モード)

10.6.1 設定手順



A/D 変換器を使用するには、下記の手順で設定します。

(1)A/D コントロール／ステータスレジスタ(ADCSR)で下記を設定します。
 (a)スキャンモードの選択(単一モードに設定)
 (b)クロックセレクト
 (c)チャネルセレクト
(2)ソフトウェアまたは外部トリガ入力によって ADCSR の ADST ビットが 1 にセットされると、選択されたチャネルの A/D 変換を開始します。
(3)A/D 変換が終了すると A/D 変換結果がそのチャネルに対応する A/D データレジスタに転送されます。
(4)A/D 変換終了時、ADCSR の ADF フラグが 1 にセットされます。このとき、ADIE ビットが 1 にセットされていると、ADI 割り込み要求を発生します。
(5)ADST ビットは A/D 変換中は 1 を保持し、変換が終了すると自動的にクリアされて A/D 変換器は待機状態になります。

10.6.2 A/Dコントロール／ステータスレジスタ(ADCSR)の設定

(1)スキャンモードの選択

A/D 変換するとき、2つのモードがあります。

●単一モード
1 チャネルを 1 回ごとに A/D 変換します。ADCSR の bit4 には"0"を設定します。
●スキャンモード
1～4 チャネルを連続して A/D 変換します。ADCSR の bit4 には"1"を設定します。

今回は、1 チャネルの A/D 変換を行いますので、単一モードになります。A/D コントロール／ステータスレジスタ(ADCSR)の bit4 は、"0"を設定します。

A/D コントロール／ステータスレジスタ(ADCSR)							
7	6	5	4	3	2	1	0
A/D エンド フラグ	A/D インタラプト イネーブル	A/D スタート	スキャン モード	クロック セレクト	チャネルセレクト 2～0		
			0				

(2)クロックセレクト

A/D 変換時間の設定を行います。
0:変換時間＝134 ステート(max)
1:変換時間＝70 ステート(max)
※変換時間の切り替えは、ADST＝0 の状態で行ってください。

1 ステートとは、φ のことです。したがって、
134 ステート＝1 クロック×134＝(1/14.7456×10^6)×134＝9.087[μs]
70 ステート＝1 クロック× 70＝(1/14.7456×10^6)× 70＝4.747[μs]

一般的に、速いが精度が悪い、遅いが精度がよい、と考え勝ちですが、ここでは単純に速いか遅いかだけです。精度はいっさい変わりません。今回は、どちらでも構いませんが"0"を設定します。

A/D コントロール／ステータスレジスタ(ADCSR)							
7	6	5	4	3	2	1	0
A/D エンド フラグ	A/D インタラプト イネーブル	A/D スタート	スキャン モード	クロック セレクト	チャネルセレクト 2～0		
				0			

(3)チャネルセレクト

どの端子を使うか選択します。先に SCAN=0 を設定しているので、「SCAN=0 のとき」部分が対象です。

ビット	ビット名	初期値	R/W	説　　明
2	CH2	0	R/W	チャネルセレクト 2～0
1	CH1	0	R/W	アナログ入力チャネルを選択します。
0	CH0	0	R/W	SCAN＝0 のとき　　SCAN＝1 のとき
				000：AN0　　000：AN0
				001：AN1　　001：AN0～AN1
				010：AN2　　010：AN0～AN2
				011：AN3　　011：AN0～AN3
				100：AN4　　100：AN4
				101：AN5　　101：AN4～AN5
				110：AN6　　110：AN4～AN6
				111：AN7　　111：AN4～AN7

今回は、AN0 端子をアナログ入力端子にします。A/D コントロール／ステータスレジスタ(ADCSR)の bit2～0 には、"000"を設定します。

A/D コントロール／ステータスレジスタ(ADCSR)							
7	6	5	4	3	2	1	0
A/D エンド フラグ	A/D インタラプト イネーブル	A/D スタート	スキャン モード	クロック セレクト	チャネルセレクト 2～0		
					0	**0**	**0**

(1)～(3)の設定を合わせると、プログラムでは下記のようになります。

```
ADCSR = 0x00;
```

10.7 A/D変換動作(単一モード)

10.7.1 設定内容

```
ADCSR = 0x00;
```

AN0 端子をアナログ電圧入力端子にします。

10.7.2 A/D変換の開始

ソフトウェアまたは外部トリガ入力によって A/D コントロール／ステータスレジスタ(ADCSR)の ADST ビットが 1 にセットされると、選択されたチャネルの A/D 変換を開始します。ADCSR の ADST ビットのみ"1"にするため、ADCSR は OR 演算で bit5 を"1"にします。

```
    ADCSR |= 0x20;
```

10.7.3 A/D変換の終了のチェック

A/D 変換が終了するとA/D 変換結果がそのチャネルに対応するA/D データレジスタに転送されます。
A/D 変換終了時、ADCSR の ADF フラグが 1 にセットされます。このとき、ADIE ビットが 1 にセットされていると、ADI 割り込み要求を発生します。
ADST ビットは A/D 変換中は 1 を保持し、変換が終了すると自動的にクリアされて A/D 変換器は待機状態になります。

A/D コントロール／ステータスレジスタ(ADCSR)							
7	6	5	4	3	2	1	0
A/D エンド フラグ	A/D インタラプト イネーブル	A/D スタート	スキャン モード	クロック セレクト	チャネルセレクト 2～0		
1		**0**					

変換が終了すると"1"になる　　変換が終了すると"0"になる

そのため、A/D 変換が終了したかどうかは、「ADCSR の ADF(bit7)が"1"になったなら終了」と判断できます。"1"になったので、次に備えて"0"にしておきます。

```
    while( !(ADCSR & 0x80) );       /* A/D変換中はこの行でループ、変換終了したら次の行へ  */
    ADCSR &= 0x7f;                  /* "0"にしておく                                      */
```

while のカッコの中の意味は、下記のようになります。
変換中の時、bit7＝"0"です。カッコの中は、
!(ADCSR & 0x80)
＝!(0x00 & 0x80)　←ADCSR 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、
!(ADCSR & 0x80)
＝!(0x80 & 0x80)　←ADCSR 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

10.7.4 A/D変換値を取り込むレジスタ

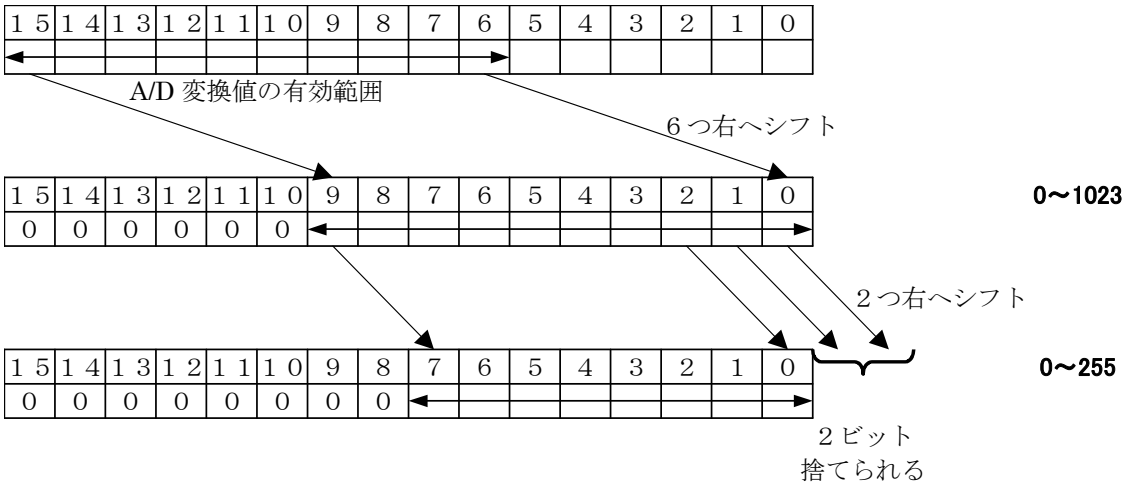
A/D 変換値を取り込むレジスタは、どの端子から読み込んだかによって変わります。

電圧を入力する端子	読み込むレジスタ
AN0 か AN4 を使用	ADDRA
AN1 か AN5 を使用	ADDRB
AN2 か AN6 を使用	ADDRC
AN3 か AN7 を使用	ADDRD

今回のプログラムでは、AN0 端子を使うので、読み込むレジスタは ADDRA になります。

10.7.5 A/D変換値の加工

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 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。



プログラムは、下記のようなります。変数 i には、A/D 変換値に応じて 0～255 の値が代入されます。

```
    i = ADDRA >> 8;
```

10.8 プログラムの解説

10.8.1 A/D変換の初期設定(単一モード)

init関数で、I/Oポートの入出力設定終了後、A/Dコントロール／ステータスレジスタ(ADCSR)の初期設定を行います。設定内容は、「**10.6.2 A/Dコントロール／ステータスレジスタ(ADCSR)の設定**」を参照してください。

```
61 :      ADCSR = 0x00;
```

10.8.2 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。

```
69 :  int get_ad( void )
70 :  {
71 :      int i;
72 :
73 :      ADCSR |= 0x20;                    /* AD スタート                */
74 :      while( !(ADCSR & 0x80) );         /* エンドフラグをチェック     */
75 :      ADCSR &= 0x7f;                    /* エンドフラグクリア         */
76 :      i = ADDRA >> 8;                   /* 代入                       */
77 :
78 :      return i;
79 :  }
```

73 行から 75 行目は、A/D 変換を開始してから終了までの部分です。この部分は常に同じです。
76 行目の **A/D データレジスタは、読み込む端子によって A～D が変わります**。端子を変えた場合は、読み込む A/D データレジスタも忘れずに変更するようにしてください。

10.8.3 main関数

```
31 :  void main( void )
32 :  {
33 :      int ad;
34 :
35 :      init();                           /* マイコン機能の初期化       */
36 :
37 :      while( 1 ) {
38 :          ad = get_ad();
39 :          PDR6 = ad;
40 :      }
41 :  }
```

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 6 へ出力しています。これを繰り返します。

10.9 演習

AN3 端子(PB3)をアナログ電圧入力端子として、A/D 変換値をポート 6 に出力してみましょう。下記は回答例です。

```
void init( void )
{

    中略

    /* A/Dの初期設定 */
    ADCSR = 0x03;
}

int get_ad( void )
{
    int i;

    ADCSR |= 0x20;                  /* ADスタート              */
    while( !(ADCSR & 0x80) );       /* エンドフラグをチェック  */
    ADCSR &= 0x7f;                  /* エンドフラグクリア      */
    i = ADDRD >> 8;                 /* 代入                    */

    return i;
}
```

10.10 まとめ

設定するレジスタ	詳細
ADCSR	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…AN0 端子　0x01…AN1 端子　0x02…AN2 端子　0x03…AN3 端子 0x04…AN4 端子　0x05…AN5 端子　0x06…AN6 端子　0x07…AN7 端子

A/D 値の読み込みは、下記のように行います。

```
69 :  int get_ad( void )
70 :  {
71 :      int i;
72 :
73 :      ADCSR |= 0x20;                  ←A/D 変換のスタート
74 :      while( !(ADCSR & 0x80) );       ←A/D 変換が終わったかどうかチェック/
75 :      ADCSR &= 0x7f;                  ←終わったときに"1"になるビットをクリア
76 :      i = ADDRA >> 8;                 ←レジスタより A/D 値の読み込み
77 :                                        8 ビット右シフトなら 0～255 の値、
78 :      return i;                         6 ビット右シフトなら 0～1023 の値が返ってくる
79 :  }
```

76 行の A/D データレジスタ A (ADDRA)は、下記のように使用する端子によって変わります。

・AN0 か AN4 の端子の A/D 変換値を読み込むとき→ADDRA から読み込み
・AN1 か AN5 の端子の A/D 変換値を読み込むとき→ADDRB から読み込み
・AN2 か AN6 の端子の A/D 変換値を読み込むとき→ADDRC から読み込み
・AN3 か AN7 の端子の A/D 変換値を読み込むとき→ADDRD から読み込み

A/D コントロール／ステータスレジスタ(ADCSR)の初期設定(61 行)が 0x00 の場合、AN0 端子を使用する設定なので、ADDRA から A/D 変換値を読み込みます。

※H8/3687 シリーズのハードウェアマニュアルでは、アナログ入力端子として使用するとき、ポート名を PB0 と呼ばず、AN0(アナログ入力端子 0)と呼んでいます。下記のように読み替えてください。

AN0→PB0
AN1→PB1
AN2→PB2
AN3→PB3
AN4→PB4
AN5→PB5
AN6→PB6
AN7→PB7

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

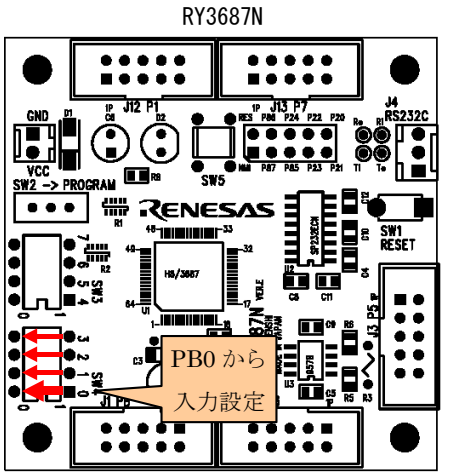
11.1 概要

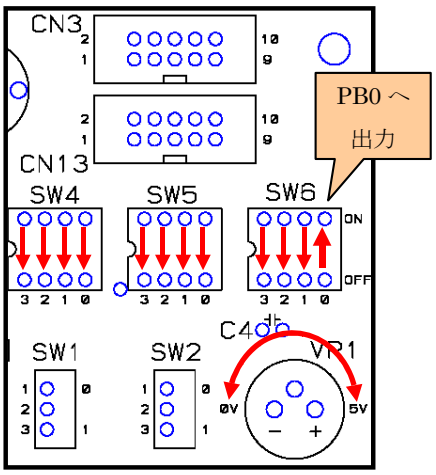
実習基板のボリューム電圧をマイコンで読み込み A/D 変換します。その値を LED に出力して変換値を見てみます。マイコンのポートは下記を使用します。
・ポート B の 0 ビットと 1 ビット・・・アナログ電圧(0～5V)入力
・ポート 6 の全ビット・・・LED へデータ出力

PB0 と PB1 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 P30 が"0"なら、PB0 の A/D 変換値をポート 6 へ出力
・CPU ボードのディップスイッチ P30 が"1"なら、PB1 の A/D 変換値をポート 6 へ出力

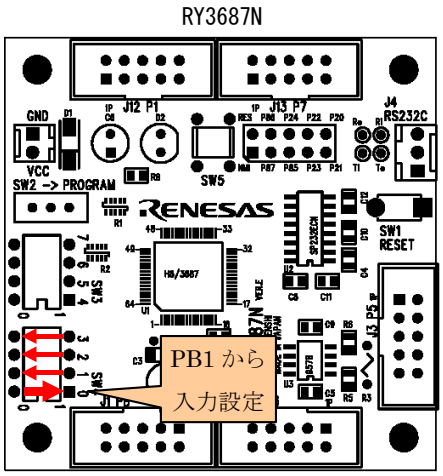
11.2 接続

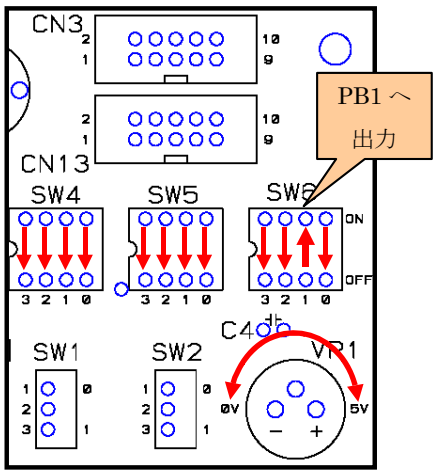
ad.c と同じです。CPU ボードのディップスイッチ P30 によって、SW6 の 0 を ON にするか、1 を ON にするか変えます。





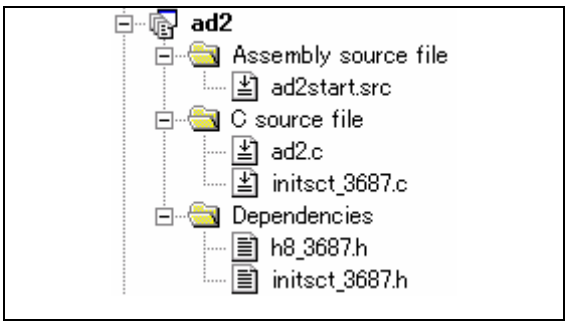
CPU ボードのディップスイッチ P30 が"0"なら、PB0 からのアナログ値が読み込まれます。





CPU ボードのディップスイッチ P30 が"1"なら、PB1 からのアナログ値が読み込まれます。

11.3 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_3687.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。H8/3687F の内蔵周辺機能の初期化も行います。
4	h8_3687.h	H8/3687F 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_3687.h	initsct_3687.c のヘッダファイルです。

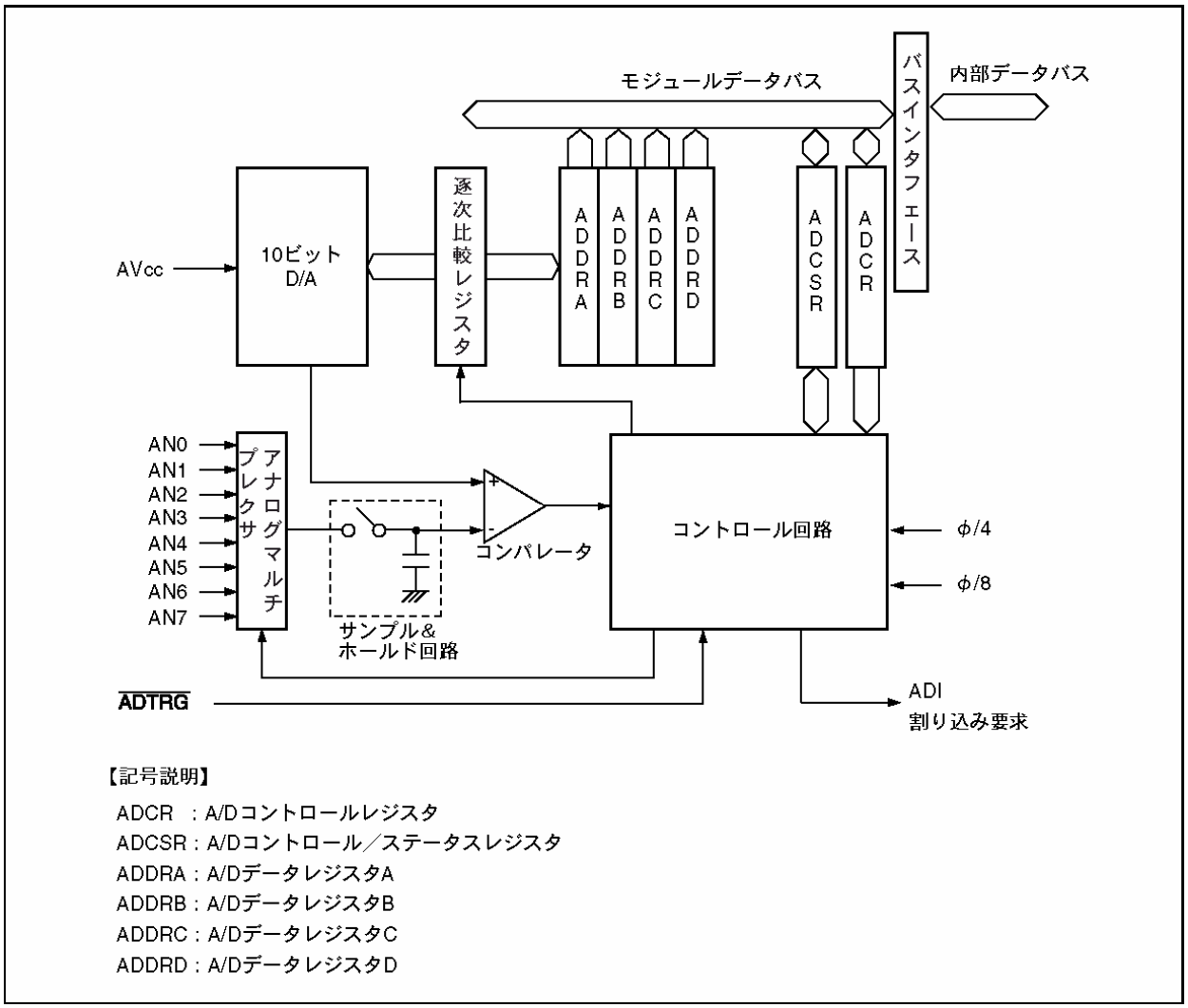
11.4 プログラム「ad2.c」

```
 1 :  /*********************************************************************/
 2 :  /* 電圧入力(スキャンモード) "ad2.c"                                  */
 3 :  /*                          2007.04 ジャパンマイコンカーラリー実行委員会 */
 4 :  /*********************************************************************/
 5 :  /*
 6 :  入力：PB0とPB1(0～5V)
 7 :  出力：P67-P60(LEDなど)
 8 :
 9 :  CPUボードのディップスイッチP30が"0"ならPB0の電圧、"1"ならPB1の電圧を
10 :  ポート６に出力します。
11 :  */
12 :
13 :  /*====================================*/
14 :  /* インクルード                       */
15 :  /*====================================*/
16 :  #include    <machine.h>
17 :  #include    "h8_3687.h"
18 :
19 :  /*====================================*/
20 :  /* プロトタイプ宣言                   */
21 :  /*====================================*/
22 :  void init( void );
23 :  int get_ad0( void );
24 :  int get_ad1( void );
25 :
26 :  /*====================================*/
27 :  /* グローバル変数の宣言               */
28 :  /*====================================*/
29 :
```

```
30 :  /************************************************************************/
31 :  /* メインプログラム                                                     */
32 :  /************************************************************************/
33 :  void main( void )
34 :  {
35 :      int ad;
36 :
37 :      init();                             /* マイコン機能の初期化      */
38 :
39 :      while( 1 ) {
40 :          if( !(PDR3 & 0x01) ) {
41 :              ad = get_ad0();
42 :              PDR6 = ad;
43 :          } else {
44 :              ad = get_ad1();
45 :              PDR6 = ad;
46 :          }
47 :      }
48 :  }
49 :
50 :  /************************************************************************/
51 :  /* H8/3687F 内蔵周辺機能  初期化                                        */
52 :  /************************************************************************/
53 :  void init( void )
54 :  {
55 :      /* I/Oポートの入出力設定 */
56 :      PCR1 = 0xff;
57 :      PCR2 = 0xfd;                        /* 通信ビットP22:TxD P21:RxD*/
58 :      PCR3 = 0xf0;                        /* 基板上のディップスイッチ */
59 :      PCR5 = 0xff;
60 :      PCR6 = 0xff;                        /* LED基板                  */
61 :      PCR7 = 0xff;
62 :      PCR8 = 0xff;
63 :      /* ポートBは、入力専用なので入出力設定はありません。          */
64 :      /* また、A/D変換器のADCSRによって指定された1ビットだけは、   */
65 :      /* 入力ポートとしては使えません。                             */
66 :
67 :      /* A/Dの初期設定 */
68 :      ADCSR = 0x11;                       /* スキャンモード使用AN0-AN1*/
69 :      ADCSR |= 0x20;                      /* ADスタート               */
70 :  }
71 :
72 :  /************************************************************************/
73 :  /* A/D値読み込み(AN0)                                                   */
74 :  /* 引数    なし                                                         */
75 :  /* 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     */
76 :  /************************************************************************/
77 :  int get_ad0( void )
78 :  {
79 :      return ADDRA >> 8;
80 :  }
81 :
82 :  /************************************************************************/
83 :  /* A/D値読み込み(AN1)                                                   */
84 :  /* 引数    なし                                                         */
85 :  /* 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     */
86 :  /************************************************************************/
87 :  int get_ad1( void )
88 :  {
89 :      return ADDRB >> 8;
90 :  }
91 :
92 :  /************************************************************************/
93 :  /* End of file                                                          */
94 :  /************************************************************************/
```

11.5 A/D変換器の設定(スキャンモード)

11.5.1 設定手順



(1)A/D コントロール／ステータスレジスタ(ADCSR)の下記を設定します。
 (a)スキャンモードの選択(スキャンモードに設定)
 (b)クロックセレクト
 (c)チャネルセレクト
(2)ソフトウェアまたは外部トリガ入力によって ADCSR の ADST ビットが 1 にセットされると、グループの第 1 チャネル(CH2＝0 のとき AN0、CH2＝1 のとき AN4)から A/D 変換を開始します。
(3)それぞれのチャネルの A/D 変換が終了すると A/D 変換結果は順次そのチャネルに対応する A/D データレジスタに転送されます。
(4)選択されたすべてのチャネルの A/D 変換が終了すると ADCSR の ADF フラグが 1 にセットされます。このとき、ADIE ビットが 1 にセットされていると、ADI 割り込み要求を発生します。A/D 変換器は再びグループの第 1 チャネルから A/D 変換を開始します。
(5)ADST ビットは自動的にはクリアされず、1 にセットされている間は、(3)～(4)を繰り返します。ADST ビットを 0 にクリアすると A/D 変換は停止します。

11.5.2 A/Dコントロール／ステータスレジスタ(ADCSR)の設定

(1)スキャンモードの選択

A/D 変換するとき、2つのモードがあります。

●単一モード
1 チャネルを 1 回ごとに A/D 変換します。ADCSR の bit4 には"0"を設定します。
●スキャンモード
1～4 チャネルを連続して A/D 変換します。ADCSR の bit4 には"1"を設定します。

今回は、AN0 と AN1 の 2 チャネルの A/D 変換を行いますので、スキャンモードを設定します。A/D コントロール／ステータスレジスタ(ADCSR)の bit4 には、"1"を設定します。

A/D コントロール／ステータスレジスタ(ADCSR)							
7	6	5	4	3	2	1	0
A/D エンド フラグ	A/D インタラプト イネーブル	A/D スタート	スキャン モード	クロック セレクト	チャネルセレクト 2～0		
			1				

(2)クロックセレクト

A/D 変換時間の設定を行います。
0:変換時間＝134 ステート(max)
1:変換時間＝70 ステート(max)
※変換時間の切り替えは、ADST＝0 の状態で行ってください。

1 ステートとは、φ のことです。したがって、
134 ステート＝1 クロック×134＝(1/14.7456×10^6)×134＝9.087[μs]
70 ステート＝1 クロック× 70＝(1/14.7456×10^6)× 70＝4.747[μs]

一般的に、速いが精度が悪い、遅いが精度がよい、と考え勝ちですが、ここでは単純に速いか遅いかだけです。精度はいっさい変わりません。今回は、どちらでも構いませんが"0"を設定します。

A/D コントロール／ステータスレジスタ(ADCSR)							
7	6	5	4	3	2	1	0
A/D エンド フラグ	A/D インタラプト イネーブル	A/D スタート	スキャン モード	クロック セレクト	チャネルセレクト 2～0		
				0			

(3)チャネルセレクト

どの端子を使うか選択します。先に SCAN=1 を設定しているので、「SCAN=1 のとき」部分が対象です。

ビット	ビット名	初期値	R/W	説　　明	
2	CH2	0	R/W	チャネルセレクト 2～0	
1	CH1	0	R/W	アナログ入力チャネルを選択します。	
0	CH0	0	R/W	SCAN＝0 のとき	SCAN＝1 のとき
				000：AN0	000：AN0
				001：AN1	001：AN0～AN1
				010：AN2	010：AN0～AN2
				011：AN3	011：AN0～AN3
				100：AN4	100：AN4
				101：AN5	101：AN4～AN5
				110：AN6	110：AN4～AN6
				111：AN7	111：AN4～AN7

今回は、AN0 端子と AN1 端子をアナログ入力端子にします。A/D コントロール／ステータスレジスタ(ADCSR)の bit2～0 には、"001"を設定します。

A/D コントロール／ステータスレジスタ(ADCSR)							
7	6	5	4	3	2	1	0
A/D エンドフラグ	A/D インタラプトイネーブル	A/D スタート	スキャンモード	クロックセレクト	チャネルセレクト 2～0		
					0	**0**	**1**

これらの設定を合わせると、プログラムでは下記のようになります。

```
ADCSR = 0x11;
```

11.6 A/D変換動作(単一モード)

11.6.1 設定内容

```
ADCSR = 0x11;
```

AN0 端子、AN1 端子をアナログ電圧入力端子にします。

11.6.2 変換の開始

ソフトウェアまたは外部トリガ入力によって A/D コントロール／ステータスレジスタ(ADCSR)の ADST ビットが 1 にセットされると、選択されたチャネルの A/D 変換を開始します。ADCSR の ADST ビットのみ"1"にするため、ADCSR は OR 演算で bit5 を"1"にします。

```
    ADCSR |= 0x20;
```

11.6.3 A/D変換の終了のチェック

スキャンモードの場合は、順番に変換し続けます。今回の場合は、AN0 を A/D 変換、終わったら AN1 を A/D 変換、終わったら AN0 に戻ります。そのため、必要なときに A/D データレジスタ A～D を読み込めば最新の A/D 変換値が読み込めます。特に、終了のチェックは必要ありません。

11.6.4 A/D変換値の取り込みレジスタ

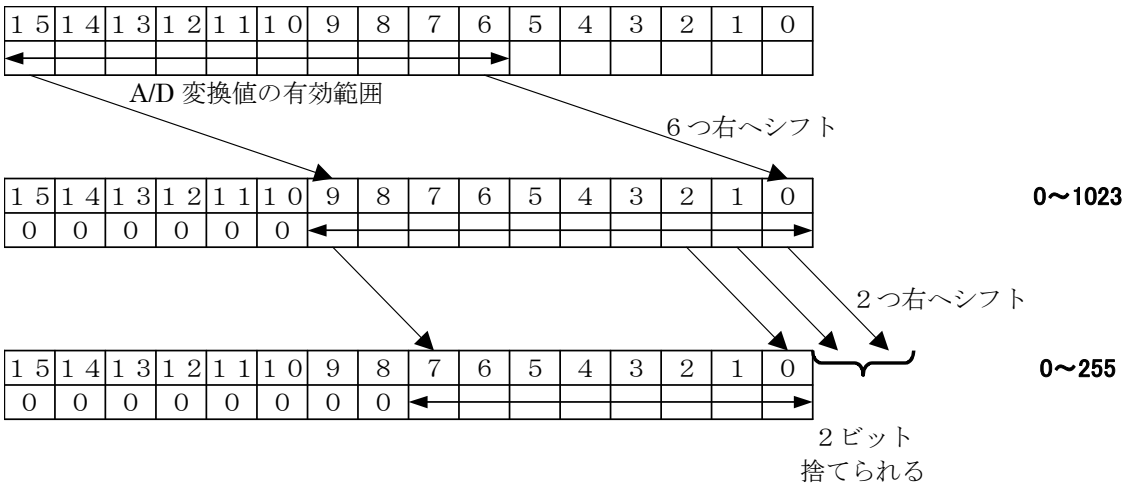
A/D 変換値を取り込むレジスタは、どの端子から読み込んだかによって変わります。

電圧を入力する端子	読み込むレジスタ
AN0 か AN4 を使用	ADDRA
AN1 か AN5 を使用	ADDRB
AN2 か AN6 を使用	ADDRC
AN3 か AN7 を使用	ADDRD

今回のプログラムでは、AN0 と AN1 端子を使うので、読み込むレジスタは、下記のようになります。
・AN0 端子の A/D 変換値は ADDRA を読み込みます
・AN1 端子の A/D 変換値は ADDRB を読み込みます

11.6.5 A/D変換値の加工

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 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。



プログラムは、下記のようなります。変数 i、変数 j に 0～255 の値が代入されます。

```
    i = ADDRA >> 8;         /* AN0 端子 */
    j = ADDRB >> 8;         /* AN1 端子 */
```

11.7 プログラムの解説

11.7.1 A/D変換の初期設定（スキャンモード）

init関数で、I/Oポートの入出力設定終了後、A/Dコントロール／ステータスレジスタ(ADCSR)の初期設定を行います。設定内容は、「**11.5.2 A/Dコントロール／ステータスレジスタ(ADCSR)の設定**」を参照してください。

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

11.7.2 A/D変換値を読み込むget_ad0 関数

get_ad0 関数で、AN0 端子の A/D 値を読み込みます。

```
77 :  int get_ad0( void )
78 :  {
79 :      return ADDRA >> 8;
80 :  }
```

スキャンモードの場合、常に変換し続けています。A/D データレジスタ A(ADDRA)から値を読めば、最新の AN0 端子の A/D 変換値を読み込むことができます。値は、8 ビット右シフトしているので 0～255 の値が読み込まれます。

11.7.3 A/D変換値を読み込むget_ad1 関数

get_ad1 関数で、AN1 端子の A/D 値を読み込みます。

```
87 :  int get_ad1( void )
88 :  {
89 :      return ADDRB >> 8;
90 :  }
```

スキャンモードの場合、常に変換し続けています。A/D データレジスタ B(ADDRB)から値を読めば、最新の AN1 端子の A/D 変換値を読み込むことができます。値は、8 ビット右シフトしているので 0～255 の値が読み込まれます。

11.7.4 main関数

```
 33 :  void main( void )
 34 :  {
 35 :      int ad;
 36 :
 37 :      init();                              /* マイコン機能の初期化       */
 38 :
 39 :      while( 1 ) {
 40 :          if( !(PDR3 & 0x01) ) {
 41 :              ad = get_ad0();
 42 :              PDR6 = ad;
 43 :          } else {
 44 :              ad = get_ad1();
 45 :              PDR6 = ad;
 46 :          }
 47 :      }
 48 :  }
```

CPU ボードのディップスイッチ P30 が"0"ならこの部分を実行します。

CPU ボードのディップスイッチ P30 が"1"ならこの部分を実行します。

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 6 へ出力しています。これを繰り返します。

CPU ボードのディップスイッチによって、AN0 端子の A/D 変換値をポート 6 に出力するか、AN1 端子の A/D 変換値をポート 6 に出力するか選択します。

11.8 まとめ

設定するレジスタ	詳細
ADCSR	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…AN0 端子　　0x11…AN0～AN1 端子 0x12…AN0～AN2 端子　　0x13…AN0～AN3 端子 0x14…AN4 端子　　0x15…AN4～AN5 端子 0x16…AN4～AN6 端子　　0x17…AN4～AN7 端子 上記設定後、 ADCSR |= 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、ADDRA、ADDRB、ADDRC、ADDRD 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。

```
77 : int get_ad0( void )
78 : {
79 :     return ADDRA >> 8;              ←レジスタより A/D 値の読み込み
80 : }                                     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
```

79 行の A/D データレジスタ A (ADDRA)は、使用する端子によって下記のように変わります。
・AN0 か AN4 の端子の A/D 変換値を読み込むとき→ADDRA から読み込み
・AN1 か AN5 の端子の A/D 変換値を読み込むとき→ADDRB から読み込み
・AN2 か AN6 の端子の A/D 変換値を読み込むとき→ADDRC から読み込み
・AN3 か AN7 の端子の A/D 変換値を読み込むとき→ADDRD から読み込み

A/D コントロール／ステータスレジスタ(ADCSR)の初期設定(68 行)が 0x11 の場合、AN0 端子と AN1 端子使用する設定なので、ADDRA と ADDRB レジスタから A/D 値を読み込みます。

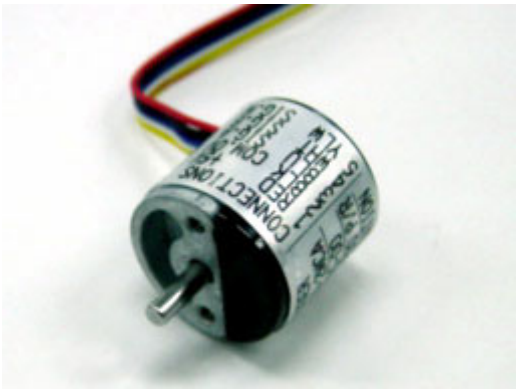
※H8/3687 シリーズのハードウェアマニュアルでは、アナログ入力端子として使用するとき、ポート名を PB0 と呼ばず、AN0(アナログ入力端子 0)と呼んでいます。下記のように読み替えてください。
AN0→PB0
AN1→PB1
AN2→PB2
AN3→PB3
AN4→PB4
AN5→PB5
AN6→PB6
AN7→PB7

12. プロジェクト「enc_v」 外部のパルスをカウント（タイマV使用）

12.1 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。

今回の演習では、H8/3687F のタイマ V という機能を使って、簡単にパルスをカウントします。

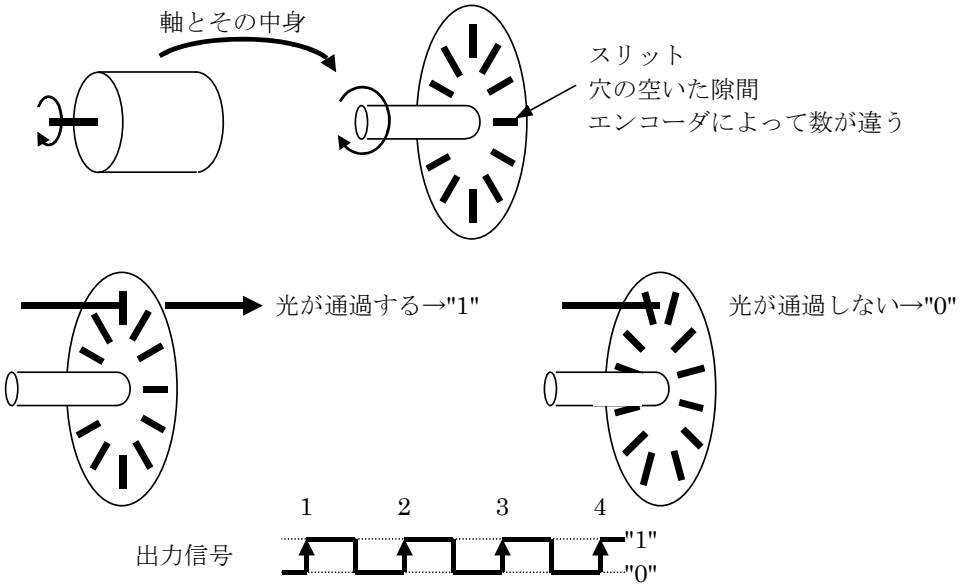


日本電産ネミコン（株） OME-100-1CA-105-015-00

12.1.1 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。

12.1.2 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

12.1.3 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。マイコンカー関係部材を扱っているヒタチインターメディックスの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分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-S41	1 回転 8 パルスで少ないですが、プログラムで 2 倍にしています。そのため、1 回転 16 パルスとなります。 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 1115 円

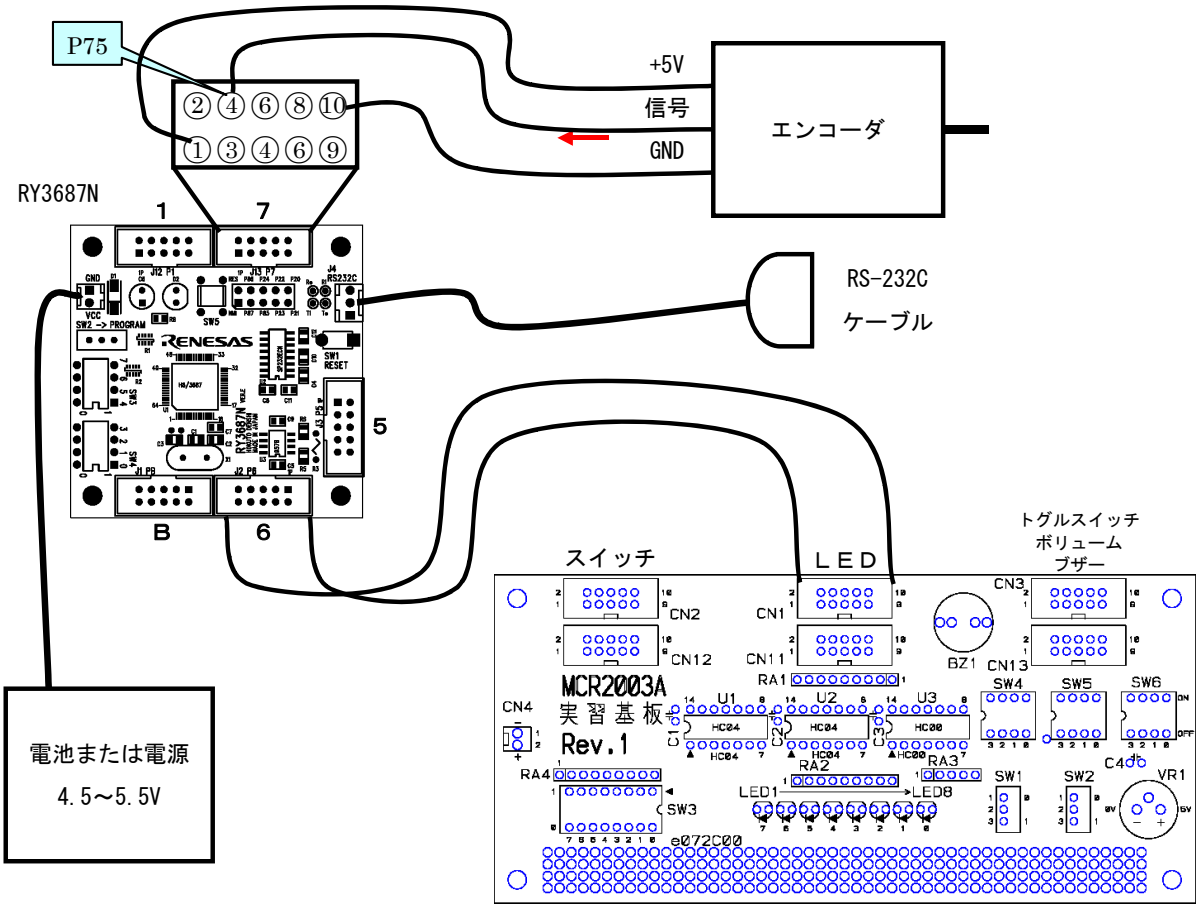
※価格は参考です。必ず各自で調べてください

※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/

12.2 接続

12.2.1 ロータリエンコーダを使用する場合

・CPU ボードのポート 7 のビット 5 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 6 と、実習基板の LED 部をフラットケーブルで接続します。



12.2.2 RY3048Fone用コネクタ付きのロータリエンコーダを使用する場合

RY3048Fone ボードでロータリエンコーダを使用するとき、PA0 端子がパルス入力端子となります。ロータリエンコーダを RY3048Fone ボードに直結できるコネクタを取り付けたとき、ロータリエンコーダの信号出力は 9 番ピンになるようにしています（下表）。

RY3048Fone ボードの PA コネクタ

ピン番号	端子名
1	+5V
2	
3	
4	
5	
6	
7	
8	
9	パルス入力
10	GND

直結

ロータリエンコーダのコネクタ

ピン番号	端子名
1	+5V
2	
3	
4	
5	
6	
7	
8	
9	パルス出力
10	GND

今回の演習は、ポート 7 コネクタの 4 番ピンである、P75 端子がパルス入力端子になります。既存のコネクタ付きロータリエンコーダを取り付けても、ピン番号が合わないのでパルスカウントできません（下表）。

RY3687N ボードの P7 コネクタ

ピン番号	端子名
1	+5V
2	
3	
4	パルス入力
5	
6	
7	
8	
9	
10	GND

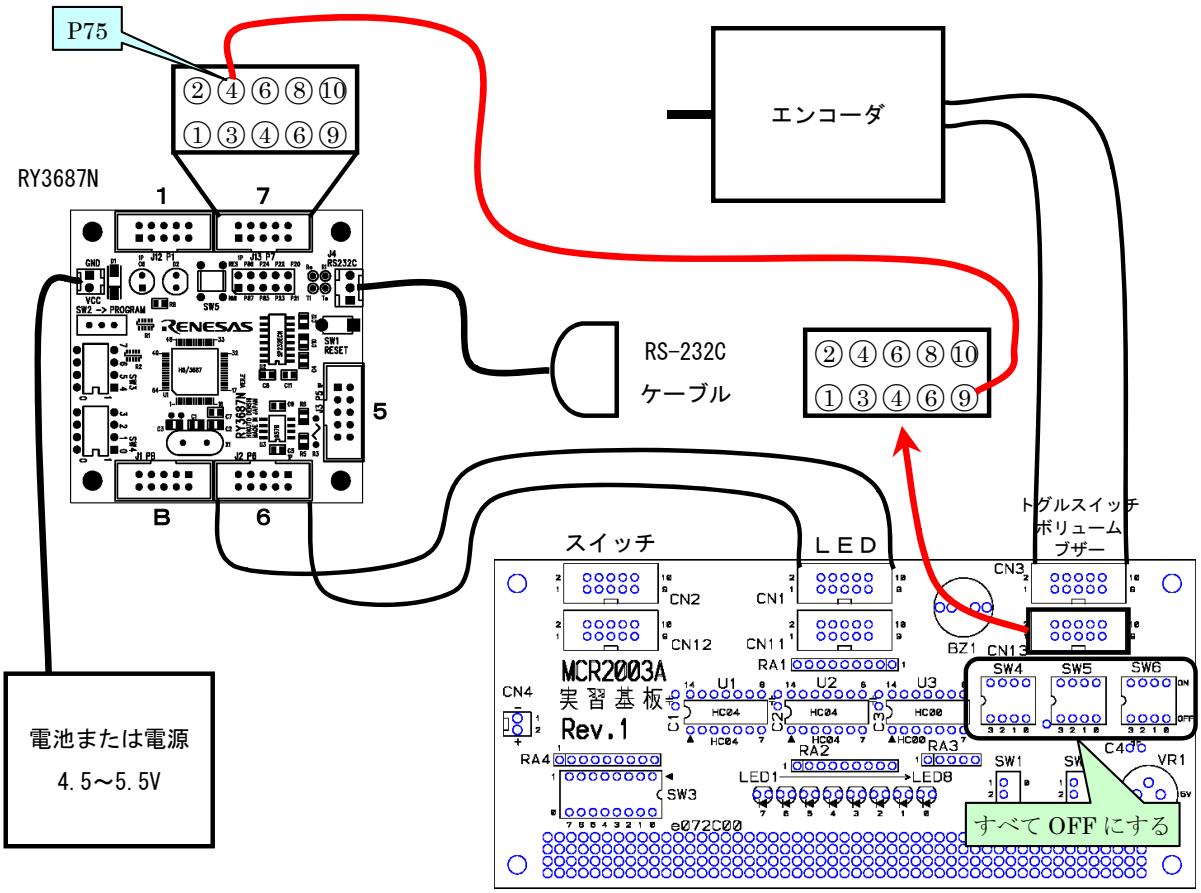
直結できない

ロータリエンコーダのコネクタ

ピン番号	端子名
1	+5V
2	
3	
4	
5	
6	
7	
8	
9	パルス出力
10	GND

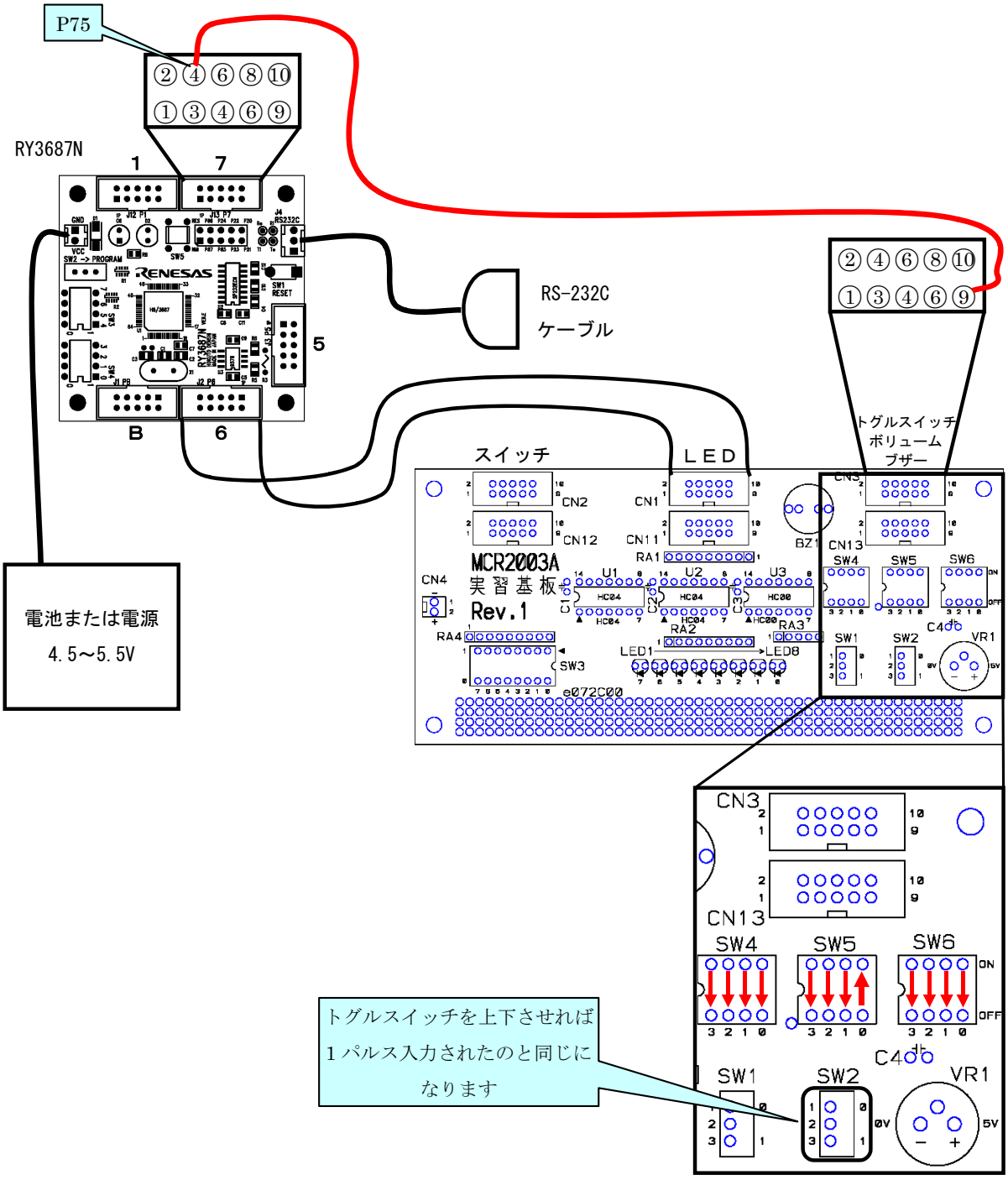
そこで、実習基板とピン付きコードを使って、P75 端子に接続します。

・実習基板の SW4～SW6 はすべて OFF にします。
・ロータリエンコーダのコネクタと、実習基板のトグルスイッチ部を接続します。
・実習基板のトグルスイッチ部の 9 ピンと、CPU ボードの P75 端子であるポート 7 の 4 番ピンを
ピン付きコードで接続します。
・CPU ボードのポート 6 と、実習基板の LED 部をフラットケーブルで接続します。



12.2.3 実習基板で代用

・実習基板のトグルスイッチ部の 9 ピンと、CPU ボードの P75 端子であるポート 7 の 4 番ピンを
ピン付きコードで接続します。
・CPU ボードのポート 6 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。

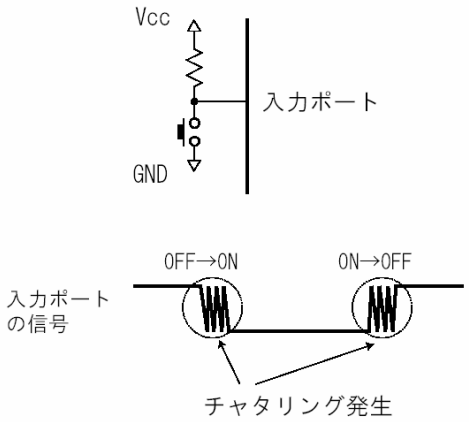


12.2.4 接続信号のチャタリングについて

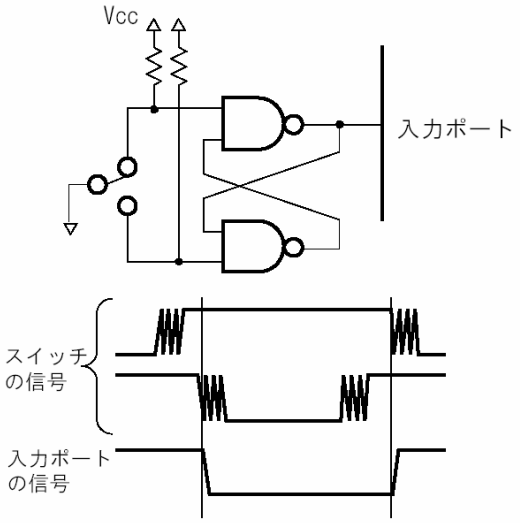
入力する信号は、ただのスイッチによるプルアップ回路ではチャタリングのため、誤カウントしてしまいます（下左図）。

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。

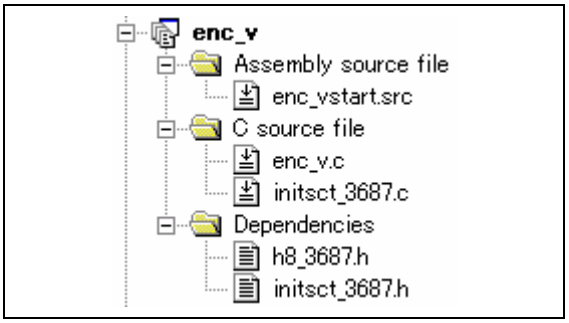
▼チャタリングの起こる回路



▼チャタリング防止回路



12.3 プロジェクトの構成



	ファイル名	内容
1	enc_vstart.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_3687.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_v.c	実際に制御するプログラムが書かれています。H8/3687F の内蔵周辺機能の初期化も行います。
4	h8_3687.h	H8/3687F 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_3687.h	initsct_3687.c のヘッダファイルです。

12.4 プログラム「enc_v.c」

```
 1 : /*********************************************************************/
 2 : /* パルスを数えてポート6に出力(タイマV使用) "enc_v.c"                 */
 3 : /*                          2007.04 ジャパンマイコンカーラリー実行委員会 */
 4 : /*********************************************************************/
 5 : /*
 6 : 入力：P75(チャタリングのないパルス)
 7 : 出力：P67-P60(LEDなど)
 8 :
 9 : ポート7のbit5に入力したパルス数を数え、ポート6に出力します。
10 : */
11 :
12 : /*====================================*/
13 : /* インクルード                       */
14 : /*====================================*/
15 : #include    <machine.h>
16 : #include    "h8_3687.h"
17 :
18 : /*====================================*/
19 : /* プロトタイプ宣言                   */
20 : /*====================================*/
21 : void init( void );
22 :
23 : /*====================================*/
24 : /* グローバル変数の宣言               */
25 : /*====================================*/
26 :
27 : /*********************************************************************/
28 : /* メインプログラム                                                  */
29 : /*********************************************************************/
30 : void main( void )
31 : {
32 :     init();                         /* マイコン機能の初期化      */
33 :
34 :     while( 1 ) {
35 :         PDR6 = TCNTV;
36 :     }
37 : }
38 :
39 : /*********************************************************************/
40 : /* H8/3687F 内蔵周辺機能　初期化                                     */
41 : /*********************************************************************/
42 : void init( void )
43 : {
44 :     /* I/Oポートの入出力設定 */
45 :     PCR1 = 0xff;
46 :     PCR2 = 0xfd;                    /* 通信ビットP22:TxD P21:RxD*/
47 :     PCR3 = 0xf0;                    /* 基板上のディップスイッチ */
48 :     PCR5 = 0xff;
49 :     PCR6 = 0xff;                    /* LED基板                  */
50 :     PCR7 = 0xdf;                    /* P75:パルス入力           */
51 :     PCR8 = 0xff;
52 :     /* ポートBは、入力専用なので入出力設定はありません。        */
53 :     /* また、A/D変換器のADCSRによって指定された1ビットだけは、  */
54 :     /* 入力ポートとしては使えません。                           */
55 :
56 :     /* タイマVの設定  */
57 :     TCRV0 = 0x05;                   /* 外部クロックの立ち上がり選択 */
58 : }
59 :
60 : /*********************************************************************/
61 : /* End of file                                                       */
62 : /*********************************************************************/
```

12.5 使用するタイマ

H8/3687F には、タイマ V、タイマ B1、タイマ Z の 3 種類あります。また、タイマ Z は、チャネル 0 とチャネル 1 の 2 つあります。どのタイマを使うかによって、パルス入力端子がどれになるか決まります。下記にタイマ名と端子を示します。

タイマ名	タイマ B1	タイマ V	タイマ Z チャネル 0	タイマ Z チャネル 1
パルス入力端子	TMIB1(P15)	TMCIV(P75)	FTIOA0(P60)	FTIOA0(P60)

タイマとパルス入力端子の関係は、必ず上表のようになります。例えば、タイマ V を使って P70 をパルス入力端子にしたい、ということはできません。タイマ V を使うと必ず P75 がパルス入力端子になります。
マイコンカーでは、主に下記のように使用します。

・タイマ B1・・・1ms ごとの割り込み
・タイマ Z・・・・PWM 出力による、左モータ、右モータ、サーボの制御

よって、タイマ V をパルスカウントとして使用することにします。

12.6 タイマVのレジスタ

タイマ V に関するレジスタは、下記の 6 レジスタあります。

・タイマカウンタ V（TCNTV）
・タイムコンスタントレジスタ A（TCORA）
・タイムコンスタントレジスタ B（TCORB）
・タイマコントロールレジスタ V0（TCRV0）
・タイマコントロール／ステータスレジスタ V（TCSRV）
・タイマコントロールレジスタ V1（TCRV1）

12.6.1 タイマカウンタV（TCNTV）

TCNTV は、8 ビットのアップカウンタです。クロックは TCRV0 の CKS2～CKS0 により選択します。TCNTV の値は CPU から常にリード／ライトできます。TCNTV は、外部リセット入力信号またはコンペアマッチ信号 A、コンペアマッチ信号 B によりクリアすることができます。いずれの信号でクリアするかは、TCRV0 の CCLR1、CCLR0 により選択します。また、TCNTV がオーバフローすると、TCSRV の OVF が 1 にセットされます。TCNTV の初期値は H'00 です。

12.6.2 タイムコンスタントレジスタA、B（TCORA、TCORB）

TCORA と TCORB は同一機能をもっています。
TCORA は 8 ビットのリード／ライト可能なレジスタです。TCORA の値は TCNTV と常に比較され、一致すると TCSRV の CMFA が 1 にセットされます。このとき TCRV0 の CMIEA が 1 なら CPU に対して割り込み要求を発生します。ただし、TCORA へのライトサイクルの T3 ステートでの比較は禁止されています。また、この一致信号（コンペアマッチ A）と TCSRV の OS3～OS0 の設定により、TMOV 端子からのタイマ出力を制御することができます。
TCORA、TCORB の初期値は H'FF です。

12.6.3 タイマコントロールレジスタV0（TCRV0）

TCRV0 は TCNTV の入力クロックの選択、TCNTV のクリア条件指定、各割り込み要求の制御を行います。

ビット	ビット名	初期値	R/W	説　　明
7	CMIEB	0	R/W	コンペアマッチインタラプトイネーブル B 1 のとき TCSRV の CMFB による割り込み要求がイネーブルになります。
6	CMIEA	0	R/W	コンペアマッチインタラプトイネーブル A 1 のとき TCSRV の CMFA による割り込み要求がイネーブルになります。
5	OVIE	0	R/W	タイマオーバフローインタラプトイネーブル 1 のとき TCSRV の OVF による割り込み要求がイネーブルになります。
4 3	CCLR1 CCLR0	0 0	R/W R/W	カウンタクリア 1～0 TCNTV のクリア条件を指定します。 00：クリアされません。 01：コンペアマッチ A でクリアされます。 10：コンペアマッチ B でクリアされます。 11：TMRIV 端子の立ち上がりエッジにでクリアされます。 クリア後の TCNTV の動作は TCRV1 の TRGE によって異なります。
2 1 0	CKS2 CKS1 CKS0	0 0 0	R/W R/W R/W	クロックセレクト 2～0 TCRV1 の ICKS0 との組合わせで、TCNTV に入力するクロックとカウント条件を選択します。表 12.2 を参照してください。

表 12.2　TCNTV に入力するクロックとカウント条件

TCRV0			TCRV1	説　　明
ビット 2	ビット 1	ビット 0	ビット 0	
CKS2	CKS1	CKS0	ICKS0	
0	0	0	－	クロック入力禁止
0	0	1	0	内部クロックφ/4 立ち下がりエッジでカウント
0	0	1	1	内部クロックφ/8 立ち下がりエッジでカウント
0	1	0	0	内部クロックφ/16 立ち下がりエッジでカウント
0	1	0	1	内部クロックφ/32 立ち下がりエッジでカウント
0	1	1	0	内部クロックφ/64 立ち下がりエッジでカウント
0	1	1	1	内部クロックφ/128 立ち下がりエッジでカウント
1	0	0	－	クロック入力禁止
1	0	1	－	外部クロックの立ち上がりエッジでカウント
1	1	0	－	外部クロックの立ち下がりエッジでカウント
1	1	1	－	外部クロックの立ち上がり／立ち下がり両エッジでカウント

12.6.4 タイマコントロール／ステータスレジスタV（TCSRV）

TCSRV はステータスフラグの表示およびコンペアマッチによる出力制御を行います。

ビット	ビット名	初期値	R/W	説　　明
7	CMFB	0	R/W	コンペアマッチフラグ B ［セット条件］ TCNTV の値と TCORB の値が一致したとき ［クリア条件］ CMFB＝1 の状態で、CMFB をリードした後、CMFB に 0 をライトしたとき
6	CMFA	0	R/W	コンペアマッチフラグ A ［セット条件］ TCNTV の値と TCORA の値が一致したとき ［クリア条件］ CMFA＝1 の状態で、CMFA をリードした後、CMFA に 0 をライトしたとき
5	OVF	0	R/W	タイマオーバフローフラグ ［セット条件］ TCNTV の値が H'FF から H'00 にオーバフローしたとき ［クリア条件］ OVF＝1 の状態で、OVF をリードした後、OVF に 0 をライトしたとき
4	－	1	－	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。
3 2	OS3 OS2	0 0	R/W R/W	アウトプットセレクト 3～2 TCORB と TCNTV のコンペアマッチによる TMOV 端子の出力方法を選択します。 00：変化しない。 01：0 出力 10：1 出力 11：トグル出力
1 0	OS1 OS0	0 0	R/W R/W	アウトプットセレクト 1～0 TCORA と TCNTV のコンペアマッチによる TMOV 端子の出力方法を選択します。 00：変化しない。 01：0 出力 10：1 出力 11：トグル出力

OS3 と OS2 はコンペアマッチ B による出力方法を選択し、OS1 と OS0 はコンペアマッチ A による出力方法を選択し、それぞれ独立に設定することができます。リセット後、最初のコンペアマッチが起こるまでのタイマ出力は 0 です。

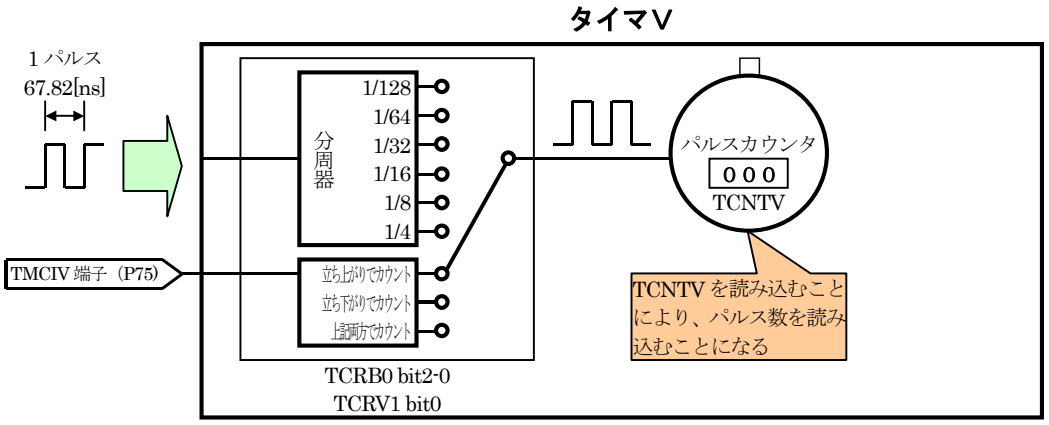
12.6.5 タイマコントロールレジスタV1（TCRV1）

TCRV1 は TRGV 端子のエッジセレクト、TRGV 入力イネーブル、TCNTV の入力クロックの選択を行います。

ビット	ビット名	初期値	R/W	説　　明
7～5	－	すべて 1	－	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。
4 3	TVEG1 TVEG0	0 0	R/W R/W	TRGV 入力エッジセレクト TRGV 端子の入力エッジを選択します。 00：TRGV からのトリガ入力を禁止 01：立ち上がりエッジを選択 10：立ち下がりエッジを選択 11：立ち上がり／立ち下がり両エッジを選択
2	TRGE	0	R/W	TVEG1、TVEG0 で選択されたエッジの入力により、TCNTV カウントアップが開始します。 0：TRGV 端子入力による TCNTV カウントアップの開始とコンペアマッチによる TCNTV クリア時の TCNTV カウントアップの停止を禁止 1：TRGV 端子入力による TCNTV カウントアップの開始とコンペアマッチによる TCNTV クリア時の TCNTV カウントアップの停止を許可
1	－	1	－	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。
0	ICKS0	0	R/W	インターナルクロックセレクト 0 TCRV0 の CKS2～CKS0 との組合せで、TCNTV に入力するクロックを選択します。表 12.2 を参照してください。

12.7 タイマVの設定

12.7.1 設定手順



(1)タイマコントロールレジスタ V0（TCRV0）とタイマコントロールレジスタ V1（TCRV1）の設定によって、タイマカウンタ V（TCNTV）をカウントアップするタイミングを TMCIV 端子(P75)に設定します。
(2)パルス入力されるたびに、タイマカウンタ V（TCNTV）がカウントアップされるので、このレジスタの値を読み込むことにより、入力されたパルス数が分かります。

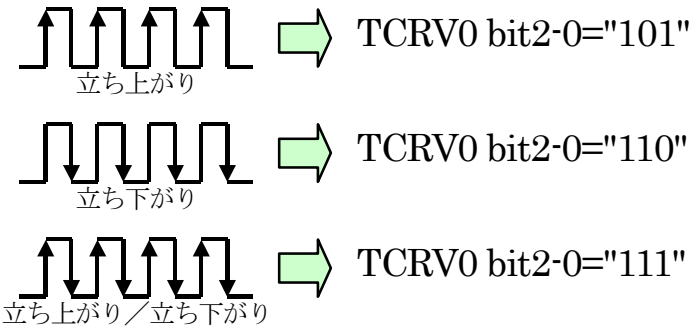
12.7.2 タイマコントロールレジスタV0（TCRV0）の設定

タイマコントロールレジスタ V0（TCRV0）の bit2～0 を設定することにより、タイマカウンタ V（TCNTV）がどのタイミングでカウントアップするか設定します。

今回は、TMCIV 端子(P75)からパルスを入力します。下記の□で囲った 3 つの設定が、TMCIV 端子(P75)からパルスを入力してカウントする設定です。パルスの立ち上がり、立ち下がりのどの段階でカウントアップするか選択することができます。

TCRV0			TCRV1	説　　明
ビット 2	ビット 1	ビット 0	ビット 0	
CKS2	CKS1	CKS0	ICKS0	
0	0	0	–	クロック入力禁止
0	0	1	0	内部クロック φ/4 立ち下がりエッジでカウント
0	0	1	1	内部クロック φ/8 立ち下がりエッジでカウント
0	1	0	0	内部クロック φ/16 立ち下がりエッジでカウント
0	1	0	1	内部クロック φ/32 立ち下がりエッジでカウント
0	1	1	0	内部クロック φ/64 立ち下がりエッジでカウント
0	1	1	1	内部クロック φ/128 立ち下がりエッジでカウント
1	0	0	–	クロック入力禁止
1	0	1	–	外部クロックの立ち上がりエッジでカウント
1	1	0	–	外部クロックの立ち下がりエッジでカウント
1	1	1	–	外部クロックの立ち上がり／立ち下がり両エッジでカウント

下記の矢印部分のような信号の変化が入力されたとき、タイマカウンタ V（TCNTV）がカウントアップされます。



今回は、信号の立ち上がりでカウントアップする設定にします。

タイマコントロールレジスタ V0（TCRV0）							
7	6	5	4	3	2	1	0
コンペアマッチインタラプトイネーブル B	コンペアマッチインタラプトイネーブル A	タイマオーバフローインタラプトイネーブル	カウンタクリア 1～0		クロックセレクト 2～0		
					1	**0**	**1**

他のビットは特に設定する必要はないので"0"にします。

```
TCRV0 = 0x05;
```

12.8 プログラムの解説

12.8.1 タイマVの初期設定

init関数で、I/Oポートの入出力設定終了後、タイマコントロールレジスタV0(TCRV0)の初期設定を行います。設定内容は、「**12.7.2 タイマコントロールレジスタV0(TCRV0)の設定**」を参照してください。

```
57 :      TCRV0 = 0x05;                          /* 外部クロックの立ち上がり選択 */
```

12.8.2 main関数

```
30 :  void main( void )
31 :  {
32 :      init();                                /* マイコン機能の初期化      */
33 :
34 :      while( 1 ) {
35 :          PDR6 = TCNTV;
36 :      }
37 :  }
```

タイマカウンタ V(TCNTV)にパルス数が入力されます。タイマカウンタ V(TCNTV)の値をそのままポート 6 へ出力します。これを繰り返します。

13. プロジェクト「pwm_z」 タイマZを使ったPWM信号出力

13.1 概要

タイマ Z のチャネル 0 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、タイマ Z のチャネル 1 と置き換えることもできます。

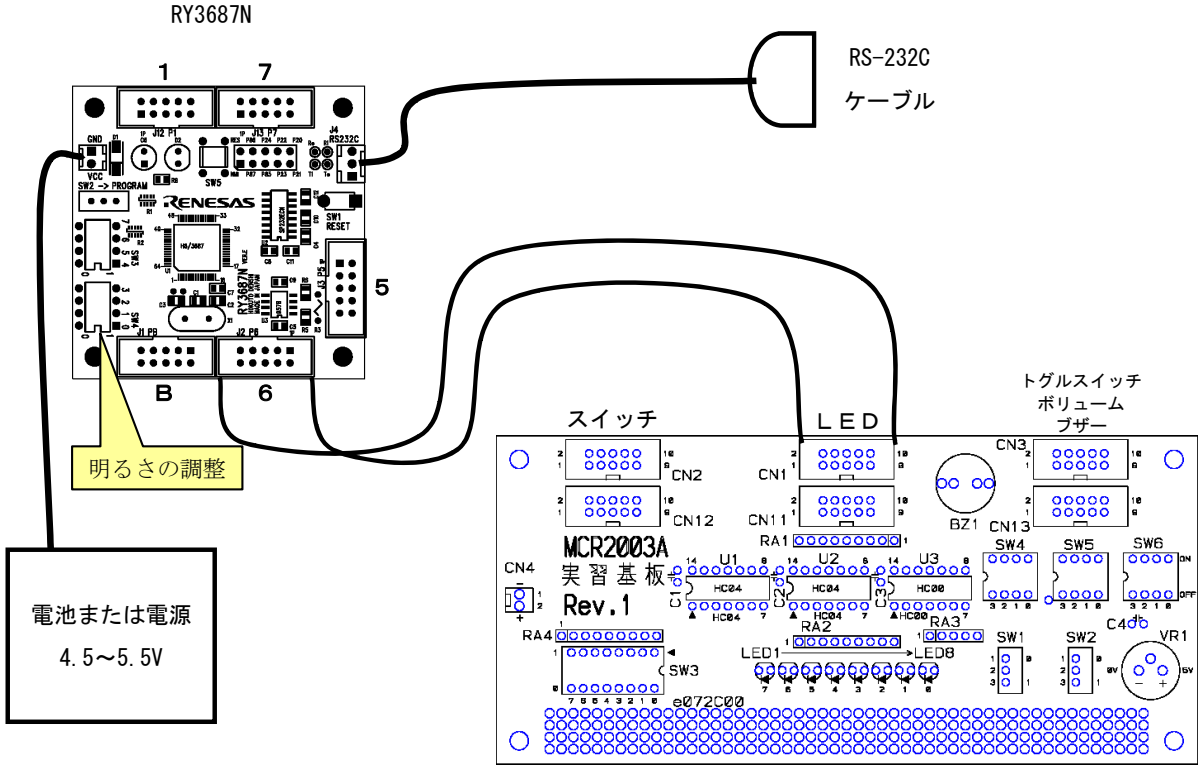
マイコンのポートは下記を使用します。

・ポート 6 の bit1、bit2、bit3・・・LED へ PWM 出力

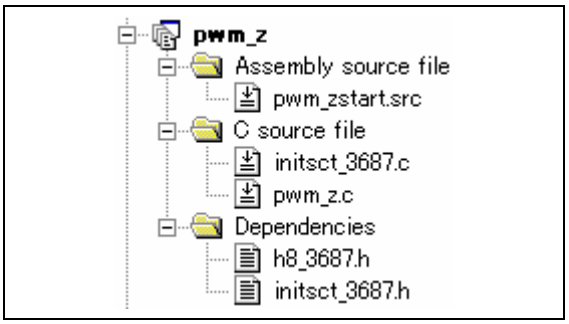
13.2 接続

・CPU ボードのポート 6 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	pwm_zstart.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_3687.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	pwm_z.c	実際に制御するプログラムが書かれています。H8/3687F の内蔵周辺機能の初期化も行います。
4	h8_3687.h	H8/3687F 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_3687.h	initsct_3687.c のヘッダファイルです。

13.4 プログラム「pwm_z.c」

```
 1 :  /*********************************************************************/
 2 :  /* タイマZによるPWM信号出力 "pwm_z.c"                                */
 3 :  /*                          2007.04 ジャパンマイコンカーラリー実行委員会 */
 4 :  /*********************************************************************/
 5 :  /*
 6 :  入力：P33-P30(CPUボード上のディップスイッチ)
 7 :  出力：P67-P60(LEDなど)
 8 :
 9 :  ポート６にリセット同期PWMモードで生成したPWM信号を出力します。
10 :  */
11 :
12 :  /*====================================*/
13 :  /* インクルード                       */
14 :  /*====================================*/
15 :  #include    <machine.h>
16 :  #include    "h8_3687.h"
17 :
18 :  /*====================================*/
19 :  /* プロトタイプ宣言                   */
20 :  /*====================================*/
21 :  void init( void );
22 :  unsigned char dipsw_get( void );
23 :
24 :  /*====================================*/
25 :  /* グローバル変数の宣言               */
26 :  /*====================================*/
27 :
```

```
28 : /************************************************************************/
29 : /* メインプログラム                                                     */
30 : /************************************************************************/
31 : void main( void )
32 : {
33 :     init();                             /* マイコン機能の初期化     */
34 :
35 :     while( 1 ) {
36 :         GRB_0 = (long)58982 * dipsw_get() / 15;     /* P61          */
37 :         GRC_0 = (long)0;                            /* P62          */
38 :         GRD_0 = (long)0;                            /* P63          */
39 :     }
40 : }
41 :
42 : /************************************************************************/
43 : /* H8/3687F 内蔵周辺機能  初期化                                        */
44 : /************************************************************************/
45 : void init( void )
46 : {
47 :     /* I/Oポートの入出力設定 */
48 :     PCR1 = 0xff;
49 :     PCR2 = 0xfd;                        /* 通信ビットP22:TxD P21:RxD*/
50 :     PCR3 = 0xf0;                        /* 基板上のディップスイッチ */
51 :     PCR5 = 0xff;
52 :     PCR6 = 0xff;                        /* LED基板                  */
53 :     PCR7 = 0xff;
54 :     PCR8 = 0xff;
55 :     /* ポートBは、入力専用なので入出力設定はありません。           */
56 :     /* また、A/D変換器のADCSRによって指定された1ビットだけは、     */
57 :     /* 入力ポートとしては使えません。                               */
58 :
59 :     /* タイマZ ch0 PWMモード */
60 :     TCR_0  = 0x22;                      /* カウンタクロック設定     */
61 :     TPMR   = 0x07;                      /* PWMモードの設定          */
62 :     TOCR   = 0x00;                      /* 初期出力値設定           */
63 :     POCR_0 = 0x00;                      /* 出力レベル設定           */
64 :     GRA_0  = 58981;                     /* 周期の設定               */
65 :     GRB_0  = 0;                         /* ON幅の設定               */
66 :     GRC_0  = 0;                         /* ON幅の設定               */
67 :     GRD_0  = 0;                         /* ON幅の設定               */
68 :     TOER   = 0xf1;                      /* PWM出力の許可            */
69 :     TSTR   = 0x01;                      /* タイマZ ch0 スタート     */
70 : }
71 :
72 : /************************************************************************/
73 : /* ディップスイッチ値読み込み                                           */
74 : /* 引数   なし                                                          */
75 : /* 戻り値 スイッチ値 0～15                                              */
76 : /************************************************************************/
77 : unsigned char dipsw_get( void )
78 : {
79 :     unsigned char sw;
80 :
81 :     sw  = PDR3;                         /* ディップスイッチ読み込み */
82 :     sw &= 0x0f;
83 :     return  sw;
84 : }
85 :
86 : /************************************************************************/
87 : /* End of file                                                          */
88 : /************************************************************************/
```

13.5 タイマZのレジスタ

タイマ Z に関するレジスタは、下記があります。

●共通
チャネル 0 とチャネル 1 の設定を同時にするレジスタです。例えば、TOER は、bit7～4 がチャネル 1 の設定、bit3～0 がチャネル 0 の設定です。

・タイマスタートレジスタ（TSTR）
・タイマモードレジスタ（TMDR）
・タイマ PWM モードレジスタ（TPMR）
・タイマファンクションコントロールレジスタ（TFCR）
・タイマアウトプットマスタイネーブルレジスタ（TOER）
・タイマアウトプットコントロールレジスタ（TOCR）

●チャネル 0
・タイマコントロールレジスタ_0（TCR_0）
・タイマ I/O コントロールレジスタ A_0（TIORA_0）
・タイマ I/O コントロールレジスタ C_0（TIORC_0）
・タイマステータスレジスタ_0（TSR_0）
・タイマインタラプトイネーブルレジスタ_0（TIER_0）
・PWM モードアウトプットレベルコントロールレジスタ_0（POCR_0）
・タイマカウンタ_0（TCNT_0）
・ジェネラルレジスタ A_0（GRA_0）
・ジェネラルレジスタ B_0（GRB_0）
・ジェネラルレジスタ C_0（GRC_0）
・ジェネラルレジスタ D_0（GRD_0）

●チャネル 1
・タイマコントロールレジスタ_1（TCR_1）
・タイマ I/O コントロールレジスタ A_1（TIORA_1）
・タイマ I/O コントロールレジスタ C_1（TIORC_1）
・タイマステータスレジスタ_1（TSR_1）
・タイマインタラプトイネーブルレジスタ_1（TIER_1）
・PWM モードアウトプットレベルコントロールレジスタ_1（POCR_1）
・タイマカウンタ_1（TCNT_1）
・ジェネラルレジスタ A_1（GRA_1）
・ジェネラルレジスタ B_1（GRB_1）
・ジェネラルレジスタ C_1（GRC_1）
・ジェネラルレジスタ D_1（GRD_1）

13.5.1 タイマスタートレジスタ（TSTR）

TSTR は TCNT の動作／停止を選択します。

ビット	ビット名	初期値	R/W	説　　明
7～2	－	すべて 1	－	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。
1	STR1	0	R/W	チャネル 1 カウンタスタート 0：TCNT_1 はカウント動作停止 1：TCNT_1 はカウント動作
0	STR0	0	R/W	チャネル 0 カウンタスタート 0：TCNT_0 はカウント動作停止 1：TCNT_0 はカウント動作

13.5.2 タイマモードレジスタ（TMDR）

TMDR はバッファ動作の設定、同期動作を選択します。

ビット	ビット名	初期値	R/W	説　　明
7	BFD1	0	R/W	バッファ動作 D1 0：GRD_1 は通常動作 1：GRB_1 と GRD_1 はバッファ動作
6	BFC1	0	R/W	バッファ動作 C1 0：GRC_1 は通常動作 1：GRA_1 と GRC_1 はバッファ動作
5	BFD0	0	R/W	バッファ動作 D0 0：GRD_0 は通常動作 1：GRB_0 と GRD_0 はバッファ動作
4	BFC0	0	R/W	バッファ動作 C0 0：GRC_0 は通常動作 1：GRA_0 と GRC_0 はバッファ動作
3～1	－	すべて 1	－	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。
0	SYNC	0	R/W	タイマ同期 0：TCNT_1、TCNT_0 はそれぞれ別々のタイマとして動作 1：TCNT_1、TCNT_0 は同期動作 各チャネルとも同期プリセット／同期クリアが可能

13.5.3 タイマPWM モードレジスタ（TPMR）

TPMR は端子を PWM モードに設定することができます。

ビット	ビット名	初期値	R/W	説　　明
7	－	1	－	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。
6	PWMD1	0	R/W	PWM モード D1 0：FTIOD1 は通常動作 1：FTIOD1 は PWM モード
5	PWMC1	0	R/W	PWM モード C1 0：FTIOC1 は通常動作 1：FTIOC1 は PWM モード
4	PWMB1	0	R/W	PWM モード B1 0：FTIOB1 は通常動作 1：FTIOB1 は PWM モード
3	－	1	－	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。
2	PWMD0	0	R/W	PWM モード D0 0：FTIOD0 は通常動作 1：FTIOD0 は PWM モード
1	PWMC0	0	R/W	PWM モード C0 0：FTIOC0 は通常動作 1：FTIOC0 は PWM モード
0	PWMB0	0	R/W	PWM モード B0 0：FTIOB0 は通常動作 1：FTIOB0 は PWM モード

13.5.4 タイマファンクションコントロールレジスタ（TFCR）

TFCR は各動作モードの設定や出力レベルの選択を行います。

ビット	ビット名	初期値	R/W	説　　明
7	－	1	－	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。
6	STCLK	0	R/W	外部クロック入力セレクト 0：外部クロック入力は無効 1：外部クロック入力は有効
5	ADEG	0	R/W	A/D トリガエッジセレクト A/D モジュールを外部トリガで A/D 変換開始の設定にしてください。 0：相補 PWM モード時、山で A/D トリガ 1：相補 PWM モード時、谷で A/D トリガ
4	ADTRG	0	R/W	外部トリガディスエーブル 0：相補 PWM モード時、PWM 周期の A/D トリガを無効 1：相補 PWM モード時、PWM 周期の A/D トリガを有効
3	OLS1	0	R/W	出力レベルセレクト 1 リセット同期 PWM モード／相補 PWM モード時に逆相の出力レベルを選択します。 0：初期出力はハイレベル、アクティブレベルはローレベル 1：初期出力はローレベル、アクティブレベルはハイレベル
2	OLS0	0	R/W	出力レベルセレクト 0 リセット同期 PWM モード／相補 PWM モード時に、正相の出力レベルを選択します。 0：初期出力はハイレベル、アクティブレベルはローレベル 1：初期出力はローレベル、アクティブレベルはハイレベル OLS1＝0、OLS0＝0 の場合のリセット同期 PWM モードおよび相補 PWM モードの出力例を図 13.4 に示します。
1 0	CMD1 CMD0	0 0	R/W R/W	コンビネーションモード 1～0 00：チャネル 0、1 は通常動作 01：チャネル 0、1 を組み合わせ、リセット同期 PWM モードで動作 10：チャネル 0、1 を組み合わせ、相補 PWM モードで動作（谷で転送） 11：チャネル 0、1 を組み合わせ、相補 PWM モードで動作（山で転送） 【注】　これらのビットによりリセット同期 PWM モード、または相補 PWM モードに設定した場合、TPMR の各ビットによる PWM モードの設定よりも優先されます。なお、リセット同期 PWM モード、および相補 PWM モードの設定は、TCNT_0、TCNT_1 を停止させた状態で行ってください。

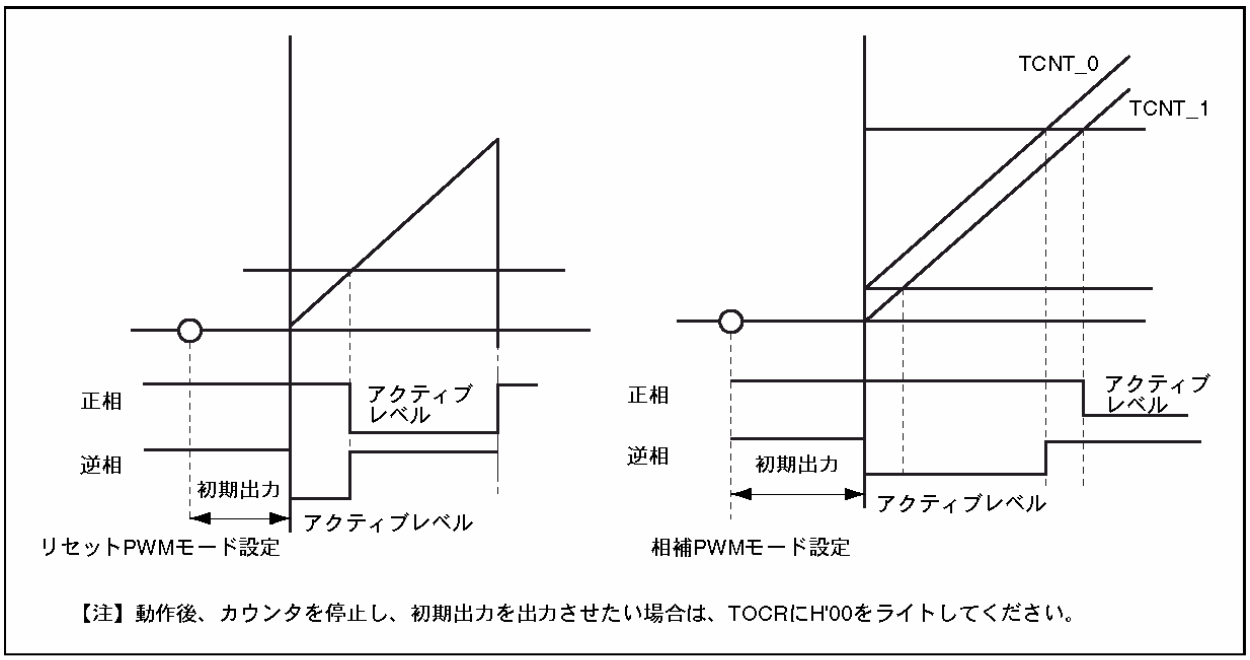


図 13.4　リセット同期 PWM モードおよび相補 PWM モードの出力例

13.5.5 タイマアウトプットマスタイネーブルレジスタ（TOER）

TOER はチャネル 0、1 の出力を許可／禁止します。$\overline{\text{WKP4}}$ 入力設定時に、$\overline{\text{WKP4}}$ に Low レベルを入力すると各ビットが 1 にセットされ、タイマ Z の出力は禁止されます。

ビット	ビット名	初期値	R/W	説　　明
7	ED1	1	R/W	マスタイネーブル D1 0：TPMR、TFCR、TIORC_1 の設定に従い、FTIOD1 端子の出力は許可 1：TPMR、TFCR、TIORC_1 の設定にかかわらず FTIOD1 端子の出力は禁止 （FTIOD1 端子は入出力ポートとして動作）
6	EC1	1	R/W	マスタイネーブル C1 0：TPMR、TFCR、TIORC_1 の設定に従い、FTIOC1 端子の出力は許可 1：TPMR、TFCR、TIORC_1 の設定にかかわらず FTIOC1 端子の出力は禁止 （FTIOC1 端子は入出力ポートとして動作）
5	EB1	1	R/W	マスタイネーブル B1 0：TPMR、TFCR、TIORA_1 の設定に従い、FTIOB1 端子の出力は許可 1：TPMR、TFCR、TIORA_1 の設定にかかわらず FTIOB1 端子の出力は禁止 （FTIOB1 端子は入出力ポートとして動作）

ビット	ビット名	初期値	R/W	説　　明
4	EA1	1	R/W	マスタイネーブル A1 0：TPMR、TFCR、TIORA_1 の設定に従い、FTIOA1 端子の出力は許可 1：TPMR、TFCR、TIORA_1 の設定にかかわらず FTIOA1 端子の出力は禁止 （FTIOA1 端子は入出力ポートとして動作）
3	ED0	1	R/W	マスタイネーブル D0 0：TPMR、TFCR、TIORC_0 の設定に従い、FTIOD0 端子の出力は許可 1：TPMR、TFCR、TIORC_0 の設定にかかわらず FTIOD0 端子の出力は禁止 （FTIOD0 端子は入出力ポートとして動作）
2	EC0	1	R/W	マスタイネーブル C0 0：TPMR、TFCR、TIORC_0 の設定に従い、FTIOC0 端子の出力は許可 1：TPMR、TFCR、TIORC_0 の設定にかかわらず FTIOC0 端子の出力は禁止 （FTIOC0 端子は入出力ポートとして動作）
1	EB0	1	R/W	マスタイネーブル B0 0：TPMR、TFCR、TIORA_0 の設定に従い、FTIOB0 端子の出力は許可 1：TPMR、TFCR、TIORA_0 の設定にかかわらず FTIOB0 端子の出力は禁止 （FTIOB0 端子は入出力ポートとして動作）
0	EA0	1	R/W	マスタイネーブル A0 0：TPMR、TFCR、TIORA_0 の設定に従い、FTIOA0 端子の出力は許可 1：TPMR、TFCR、TIORA_0 の設定にかかわらず FTIOA0 端子の出力は禁止 （FTIOA0 端子は入出力ポートとして動作）

13.5.6 タイマアウトプットコントロールレジスタ（TOCR）

TOCR はコンペアマッチが最初に起こるまでの初期出力を設定します。なお、リセット同期 PWM モード、相補 PWM モードの場合、本レジスタの設定には依存せず、TFCR の OLS1、OLS0 ビットの設定に従います。

ビット	ビット名	初期値	R/W	説　　明
7	TOD1	0	R/W	出力レベルセレクト D1 0：FTIOD1 は 0 出力* 1：FTIOD1 は 1 出力*
6	TOC1	0	R/W	出力レベルセレクト C1 0：FTIOC1 は 0 出力* 1：FTIOC1 は 1 出力*
5	TOB1	0	R/W	出力レベルセレクト B1 0：FTIOB1 は 0 出力* 1：FTIOB1 は 1 出力*
4	TOA1	0	R/W	出力レベルセレクト A1 0：FTIOA1 は 0 出力* 1：FTIOA1 は 1 出力*
3	TOD0	0	R/W	出力レベルセレクト D0 0：FTIOD0 は 0 出力* 1：FTIOD0 は 1 出力*
2	TOC0	0	R/W	出力レベルセレクト C0 0：FTIOC0 は 0 出力* 1：FTIOC0 は 1 出力*
1	TOB0	0	R/W	出力レベルセレクト B0 0：FTIOB0 は 0 出力* 1：FTIOB0 は 1 出力*
0	TOA0	0	R/W	出力レベルセレクト A0 0：FTIOA0 は 0 出力* 1：FTIOA0 は 1 出力*

【注】　*　出力値は変更した時点で反映されます。

13.5.7 タイマカウンタ（TCNT）

TCNT は、対応する GRA、GRB、GRC、GRD とのコンペアマッチ、または GRA、GRB、GRC、GRD へのインプットキャプチャにより H’0000 にクリアすることができます（カウンタクリア機能）。TCNT がオーバフローすると、対応するチャネルの TSR の OVF フラグが 1 にセットされます。TCNT_1 がアンダフローすると、TSR の UDF フラグが 1 にセットされます。なお TCNT カウンタの 8 ビット単位でのアクセスは禁止です。常に 16 ビット単位でアクセスしてください。

13.5.8 ジェネラルレジスタ A、B、C、D（GRA、GRB、GRC、GRD）

GR は 16 ビットのリード／ライト可能なレジスタで、各チャネルに 4 本、計 8 本あります。

アウトプットコンペアレジスタとインプットキャプチャレジスタの機能の切り換えを TIORA、TIORC により行います。

アウトプットコンペアレジスタとして使用しているときは、GR と TCNT の値は常に比較されています。両者の値が一致すると TSR の IMFA～IMFD フラグが 1 にセットされます。TIORA、TIORC によりコンペアマッチ出力を設定することができます。

インプットキャプチャレジスタとして使用しているときは、外部からの信号を検出して TCNT の値を格納します。このとき対応する TSR の IMFA～IMFD フラグが 1 にセットされます。インプットキャプチャ信号の検出エッジ選択は TIORA、TIORC により行います。

PWM モード、相補 PWM モード、またはリセット同期 PWM モードに設定されている場合には、TIORA、TIORC の設定値は無視されます。GR はリセット時にアウトプットコンペアレジスタ（端子出力なし）に設定され、H'FFFF に初期化されます。なお GR の 8 ビット単位でのアクセスは禁止です。常に 16 ビット単位でアクセスしてください。

13.5.9 タイマコントロールレジスタ（TCR）

TCR は TCNT のカウンタクロック選択、外部クロック選択時のエッジ選択、およびカウンタクリア要因の選択を行います。TCR は各チャネルに 1 本、計 2 本の TCR があります。

ビット	ビット名	初期値	R/W	説　　明
7	CCLR2	0	R/W	カウンタクリア 2～0
6	CCLR1	0	R/W	000：TCNT のクリア禁止
5	CCLR0	0	R/W	001：GRA のコンペアマッチ／インプットキャプチャで TCNT クリア*1
				010：GRB のコンペアマッチ／インプットキャプチャで TCNT クリア*1
				011：同期クリア。同期動作をしている他のチャネルのカウンタクリアに同期して TCNT をクリア*2
				100：TCNT のクリア禁止
				101：GRC のコンペアマッチ／インプットキャプチャで TCNT クリア*1
				110：GRD のコンペアマッチ／インプットキャプチャで TCNT クリア*1
				111：同期クリア。同期動作をしている他のチャネルのカウンタクリアに同期して TCNT をクリア*2
4	CKEG1	0	R/W	クロックエッジ 1～0
3	CKEG0	0	R/W	00：立ち上がりエッジでカウント
				01：立ち下がりエッジでカウント
				1X：立ち上がり／立ち下がりの両エッジでカウント
2	TPSC2	0	R/W	タイマプリスケーラ 2～0
1	TPSC1	0	R/W	000：内部クロック：φでカウント
0	TPSC0	0	R/W	001：内部クロック：φ/2 でカウント
				010：内部クロック：φ/4 でカウント
				011：内部クロック：φ/8 でカウント
				1XX：外部クロック：FTIOA0（TCLK）端子入力でカウント

【注】 *1 GR がアウトプットコンペアレジスタとして機能しているとき、コンペアマッチによりクリアされます。GR がインプットキャプチャとして機能しているとき、インプットキャプチャによりクリアされます。

*2 同期動作の設定は TMDR によって行います。

X： Don't care

13.5.10 タイマ I/Oコントロールレジスタ（TIORA、TIORC）

TIOR は GR の制御を行います。TIOR は TIORA と TIORC から構成されており、各チャネルに 2 本、計 4 本あります。相補 PWM モード、リセット同期 PWM モードを含む PWM モードに設定したとき、TIOR の設定は無効となります。

●TIORA

TIORA は GRA、GRB をアウトプットコンペアレジスタとして使用するか、インプットキャプチャレジスタとして使用するかを選択します。アウトプットコンペアレジスタを選択した場合は出力設定を選択し、インプットキャプチャレジスタを選択した場合はインプットキャプチャ信号の入力エッジを選択します。また FTIOA 端子、FTIOB 端子の機能を選択します。

ビット	ビット名	初期値	R/W	説　　明
7	－	1	－	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。
6	IOB2	0	R/W	I/O コントロール B2～0
5	IOB1	0	R/W	GRB はアウトプットコンペアレジスタ
4	IOB0	0	R/W	000：コンペアマッチによる端子出力禁止
				001：GRB のコンペアマッチで 0 出力
				010：GRB のコンペアマッチで 1 出力
				011：GRB のコンペアマッチでトグル出力
				GRB はインプットキャプチャレジスタ
				100：立ち上がりエッジで GRB へインプットキャプチャ
				101：立ち下がりエッジで GRB へインプットキャプチャ
				11X：立ち上がり／立ち下がりの両エッジで GRB へインプットキャプチャ
3	－	1	－	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。
2	IOA2	0	R/W	I/O コントロール A2～0
1	IOA1	0	R/W	GRA はアウトプットコンペアレジスタ
0	IOA0	0	R/W	000：コンペアマッチによる端子出力禁止
				001：GRA のコンペアマッチで 0 出力
				010：GRA のコンペアマッチで 1 出力
				011：GRA のコンペアマッチでトグル出力
				GRA はインプットキャプチャレジスタ
				100：立ち上がりエッジで GRA へインプットキャプチャ
				101：立ち下がりエッジで GRA へインプットキャプチャ
				11X：立ち上がり／立ち下がりの両エッジで GRA へインプットキャプチャ

【注】X：Don't care

●TIORC

TIORC は GRC、GRD をアウトプットコンペアレジスタとして使用するか、インプットキャプチャレジスタとして使用するかを選択します。アウトプットコンペアレジスタを選択した場合は出力設定を選択し、インプットキャプチャレジスタを選択した場合はインプットキャプチャ信号の入力エッジを選択します。また FTIOC 端子、 FTIOD 端子の機能を選択します。

ビット	ビット名	初期値	R/W	説　　明
7	－	1	－	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。
6	IOD2	0	R/W	I/O コントロール D2～0
5	IOD1	0	R/W	GRD はアウトプットコンペアレジスタ
4	IOD0	0	R/W	000：コンペアマッチによる端子出力禁止
				001：GRD のコンペアマッチで 0 出力
				010：GRD のコンペアマッチで 1 出力
				011：GRD のコンペアマッチでトグル出力
				GRD はインプットキャプチャレジスタ
				100：立ち上がりエッジで GRD へインプットキャプチャ
				101：立ち下がりエッジで GRD へインプットキャプチャ
				11X：立ち上がり／立ち下がりの両エッジで GRD へインプットキャプチャ
3	－	1	－	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。
2	IOC2	0	R/W	I/O コントロール C2～0
1	IOC1	0	R/W	GRC はアウトプットコンペアレジスタ
0	IOC0	0	R/W	000：コンペアマッチによる端子出力禁止
				001：GRC のコンペアマッチで 0 出力
				010：GRC のコンペアマッチで 1 出力
				011：GRC のコンペアマッチでトグル出力
				GRC はインプットキャプチャレジスタ
				100：立ち上がりエッジで GRC へインプットキャプチャ
				101：立ち下がりエッジで GRC へインプットキャプチャ
				11X：立ち上がり／立ち下がりの両エッジで GRC へインプットキャプチャ

【注】X：Don't care

13.5.11 タイマステータスレジスタ（TSR）

TSR は TCNT のオーバフロー／アンダフローの発生、および GRA、GRB、GRC、GRD のコンペアマッチ／インプットキャプチャの発生を示します。これらのフラグは割り込み要因であり、TIER の対応するビットにより割り込みが許可されると CPU に割り込みを要求します。TSR は各チャネル 1 本、計 2 本あります。

ビット	ビット名	初期値	R/W	説　　明
7	－	1	－	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。
6	－	1	－	
5	UDF*	0	R/W	アンダフローフラグ ［セット条件］ • TCNT_1 がアンダフローしたとき ［クリア条件］ • 1 の状態をリードした後、0 をライトしたとき
4	OVF	0	R/W	オーバフローフラグ ［セット条件］ • TCNT の値がオーバフローしたとき ［クリア条件］ • 1 の状態をリードした後、0 をライトしたとき
3	IMFD	0	R/W	インプットキャプチャ／コンペアマッチフラグ D ［セット条件］ • GRD 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、TCNT＝GRD になったとき • GRD 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 TCNT の値が GRD に転送されたとき ［クリア条件］ • 1 の状態をリードした後、0 をライトしたとき
2	IMFC	0	R/W	インプットキャプチャ／コンペアマッチフラグ C ［セット条件］ • GRC 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、TCNT＝GRC になったとき • GRC 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 TCNT の値が GRC に転送されたとき ［クリア条件］ • 1 の状態をリードした後、0 をライトしたとき

【注】　*　TSR_0 には、UDF フラグはありません。TSR_0 のビット 5 は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。

ビット	ビット名	初期値	R/W	説　　明
1	IMFB	0	R/W	インプットキャプチャ／コンペアマッチフラグ B ［セット条件］ • 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、TCNT＝GRB になったとき • 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 TCNT の値が GRB に転送されたとき ［クリア条件］ • 1 の状態をリードした後、0 をライトしたとき
0	IMFA	0	R/W	インプットキャプチャ／コンペアマッチフラグ A ［セット条件］ • 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、TCNT＝GRA になったとき • 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 TCNT の値が GRA に転送されたとき ［クリア条件］ • 1 の状態をリードした後、0 をライトしたとき

13.5.12 タイマインタラプトイネーブルレジスタ（TIER）

TIER はオーバフロー割り込み要求、GR のコンペアマッチ／インプットキャプチャ割り込み要求の許可／禁止を制御します。TIER は各チャネルに 1 本、計 2 本あります。

ビット	ビット名	初期値	R/W	説　　明
7～5	－	すべて 1	－	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。
4	OVIE	0	R/W	オーバフローインタラプトイネーブル 0：OVF、UDF フラグによる割り込み(OVI)要求を禁止 1：OVF、UDF フラグによる割り込み(OVI)要求を許可
3	IMIED	0	R/W	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 D 0：IMFD フラグによる割り込み(IMID)要求を禁止 1：IMFD フラグによる割り込み(IMID)要求を許可
2	IMIEC	0	R/W	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 C 0：IMFC フラグによる割り込み(IMIC)要求を禁止 1：IMFC フラグによる割り込み(IMIC)要求を許可
1	IMIEB	0	R/W	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 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
0	IMIEA	0	R/W	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 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

13.5.13 PWMモードアウトプットレベルコントロールレジスタ（POCR）

POCR は PWM モード時のアクティブレベルの制御をします。POCR は各チャネルに 1 本、計 2 本あります。

ビット	ビット名	初期値	R/W	説　　明
7～3	－	すべて 1	－	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。
2	POLD	0	R/W	PWM モードアウトプットレベルコントロール D 0：FTIOD の出力レベルはローアクティブ 1：FTIOD の出力レベルはハイアクティブ
1	POLC	0	R/W	PWM モードアウトプットレベルコントロール C 0：FTIOC の出力レベルはローアクティブ 1：FTIOC の出力レベルはハイアクティブ
0	POLB	0	R/W	PWM モードアウトプットレベルコントロール B 0：FTIOB の出力レベルはローアクティブ 1：FTIOB の出力レベルはハイアクティブ

13.6 PWM

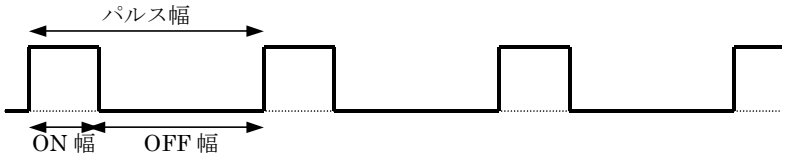
13.6.1 PWMとは？

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。

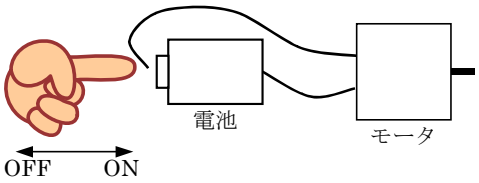
ボリュームを使えば電圧を可変することができます。しかし、モータへは大電流が流れるため、非常に大きな抵抗が必要です。また、モータに加えなかった分は、抵抗の熱となってしまいます。

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。

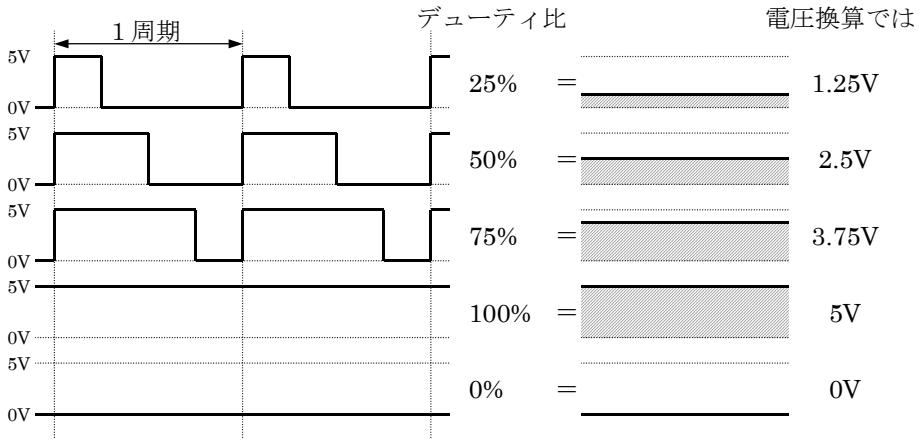
「PWM」と聞くと、何か難しく感じてしまいますが、下記のように手でモータと電池の線を「繋ぐ」、「離す」の繰り返し、それも PWM と言えます。繋いでいる時間が長いとモータは速く回ります。離している時間が長いとモータは少ししか回りません。人なら「繋ぐ」、「離す」の動作をコンマ数秒でしか行えませんが、マイコンなら数ミリ秒で行うことができます。



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**。

13.6.2 PWM出力端子

PWM 信号を出力できる端子は決まっています。自由に決めることはできません。

内蔵周辺機能	タイマ Z チャネル 0	タイマ Z チャネル 1	タイマ V	14 ビット PWM
PWM 出力数	3	3	1	1
出力端子	FTIOB0(P61)端子 FTIOC0(P62)端子 FTIOD0(P63)端子	FTIOB1(P65)端子 FTIOC1(P66)端子 FTIOD1(P67)端子	TMOV(P76)端子	PWM(P11)端子
カウンタのビット	16bit	16bit	8bit	8bit (周期は 2 種類しか選べない)

今回の演習は、タイマ Z チャネル 0 を使います。したがって、P61,P62,P63 端子から PWM 信号を出力します。

13.7 タイマZの設定

13.7.1 設定手順

タイマ Z のチャネル 0 を PWM モードで設定するには、下記のような手順で各レジスタを設定します。

(1) TCR の TPSC2～TPSC0 ビットでカウンタクロックを選択してください。外部クロックを選択した場合は、TCR の CKEG1、CKEG0 ビットにより外部クロックのエッジを選択してください。
(2) TCR の CCLR2、CCLR1、CCLR0 ビットによりカウンタクリア要因を選択してください。
(3) TPMR の PWMB0～PWMD0、PWMB1～PWMD1 ビットで PWM モードを選択してください。
(4) TOCR の TOB0～TOD0、TOB1～TOD1 ビットで初期出力値を設定してください。
(5) POCR の POLB～POLD ビットで出力レベルを設定してください。
(6) GRA に周期を設定し、他の GR にデューティを設定してください。
(7) TOER でタイマ出力の許可／禁止を設定してください。
(8) TSTR の STR ビットを 1 にセットし、カウンタ動作を開始してください。

※タイマ Z はチャネル 0 とチャネル 1 の 2 つあります。この演習では、チャネル 0 を対象として説明します。

13.7.2 パルスカウンタをカウントするタイミングの設定

タイマカウンタ(TCNT)、タイマコントロールレジスタ(TCR)は 2 つあり、タイマ Z のチャネルによって番号が異なります。
・タイマ Z チャネル 0…タイマカウンタ_0(TCNT_0)、タイマコントロールレジスタ_0(TCR_0)
・タイマ Z チャネル 1…タイマカウンタ_1(TCNT_1)、タイマコントロールレジスタ_1(TCR_1)

タイマカウンタ_0(TCNT_0)は、16 ビットのパルスカウンタです。入力されたパルスを数えます。タイマコントロールレジスタ_0(TCR_0)の bit2～0 を設定することにより、入力するパルスの幅を変えることができます。1 パルス幅は、クリスタルを基準にしており、正確な間隔で送られてきます。そのため、**パルスを数えることは、時間を計ることと同じです**。下記に、設定値と 1 パルス幅の計算をします。また、最大 PWM 周期(TCNT_0 が 65536 になるまでの時間)も計算しておきます。

TCR_0			内容
bit2	bit1	bit0	
0	0	0	φでカウント 1 パルス＝1/(14.7456×10^6)＝67.72[ns] 最大 PWM 周期＝67.72×10^{-9}×65536＝4.44[ms]
0	0	1	φ/2 でカウント 1 パルス＝1/(14.7456×10^6/2)＝135.63[ns] 最大 PWM 周期＝135.63×10^{-9}×65536＝8.89[ms]
0	1	0	φ/4 でカウント 1 パルス＝1/(14.7456×10^6/4)＝271.27[ns] 最大 PWM 周期＝271.27×10^{-9}×65536＝17.78[ms]
0	1	1	φ/8 でカウント 1 パルス＝1/(14.7456×10^6/8)＝542.53[ns] 最大 PWM 周期＝542.53×10^{-9}×65536＝35.56[ms]
1	0	0	外部クロック FTIOA0(TCLK)端子から入力したパルスでカウント

今回、PWM 周期は 16[ms]にします。どの設定にすればよいのでしょうか。最大 PWM 周期が短い順に見ていきます。最大 PWM 周期が 16ms 以下の場合は、計ることができませんので設定できません。一番最初に 16ms 以上になったときが、設定する値です。

・φのとき、最大 PWM 周期は 4.44[ms] → 16ms 以下なので不可
・φ/2 のとき、最大 PWM 周期は 8.89[ms] → 16ms 以下なので不可
・φ/4 のとき、最大 PWM 周期は 17.78[ms] → 16ms 以上なので適合

よって、タイマコントロールレジスタ_0(TCR_0) bit2～0 の設定は、φ/4 である"010"にします。

タイマコントロールレジスタ_0(TCR_0)							
7	6	5	4	3	2	1	0
カウンタクリア 2～0			クロックエッジ 1～0		タイマプリスケーラ 2～0		
					0	**1**	**0**

13.7.3 カウンタクリア要因の設定

PWM 動作のときは、タイマコントロールレジスタ_0（TCR_0）の bit7～5 は"001"としてください。

タイマコントロールレジスタ 0（TCR_0）							
7	6	5	4	3	2	1	0
カウンタクリア 2～0			クロックエッジ 1～0		タイマプリスケーラ 2～0		
0	**0**	**1**					

パルスカウンタをカウントするタイミングの設定と合わせて、プログラムは下記のようになります。

```
TCR_0 = 0x22;
```

13.7.4 PWMモードの選択

端子を PWM 出力用に切り替えます。タイマ PWM モードレジスタ（TPMR）で設定します。タイマ Z チャネル 0 は bit2～0 が対象です。今回は、全ビットを PWM 出力にします。

タイマ PWM モードレジスタ（TPMR）							
タイマ Z チャネル 1				**タイマ Z チャネル 0**			
7	6	5	4	3	2	1	0
—	FTIOD1(P67)端子を PWM 出力端子にするなら"1"	FTIOC1(P66)端子を PWM 出力端子にするなら"1"	FTIOB1(P65)端子を PWM 出力端子にするなら"1"	—	FTIOD0(P63)端子を PWM 出力端子にするなら"1"	FTIOC0(P62)端子を PWM 出力端子にするなら"1"	FTIOB0(P61)端子を PWM 出力端子にするなら"1"
					1	**1**	**1**

プログラムは下記のようになります。

```
TPMR = 0x07;
```

13.7.5 初期出力値の設定

PWM 出力端子の初期出力値を設定します。タイマアウトプットコントロールレジスタ（TOCR）でコンペアマッチが最初に起こるまでの初期出力レベルを設定します。

タイマアウトプットコントロールレジスタ（TOCR）							
タイマ Z チャネル 1				**タイマ Z チャネル 0**			
7	6	5	4	3	2	1	0
FTIOD1(P67)端子の初期出力を"1"にするなら"1"	FTIOC1(P66)端子の初期出力を"1"にするなら"1"	FTIOB1(P65)端子の初期出力を"1"にするなら"1"	FTIOA1(P64)端子の初期出力を"1"にするなら"1"	FTIOD0(P63)端子の初期出力を"1"にするなら"1"	FTIOC0(P62)端子の初期出力を"1"にするなら"1"	FTIOB0(P61)端子の初期出力を"1"にするなら"1"	FTIOA0(P60)端子の初期出力を"1"にするなら"1"
				0	**0**	**0**	

プログラムは下記のようになります。

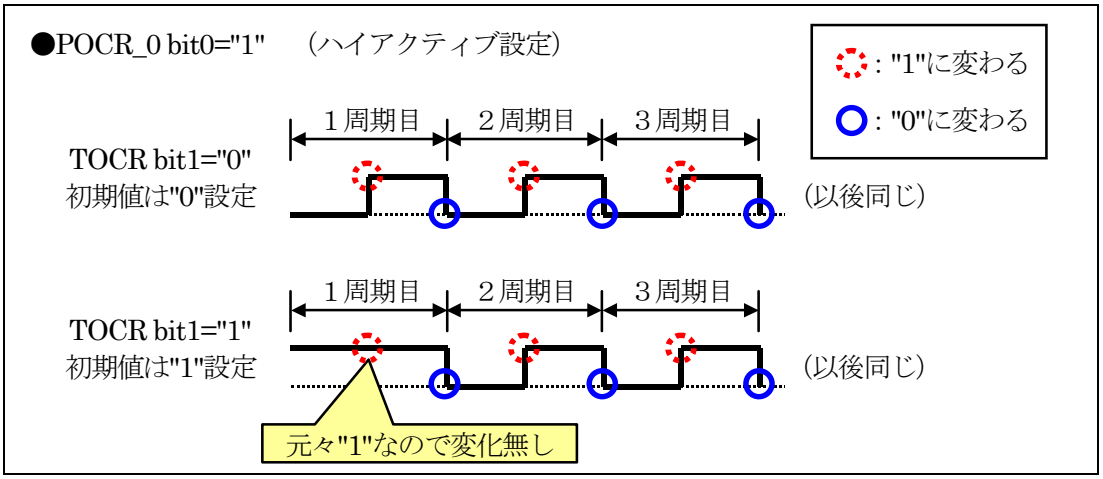
```
TOCR = 0x00;
```

13.7.6 出力レベルの設定

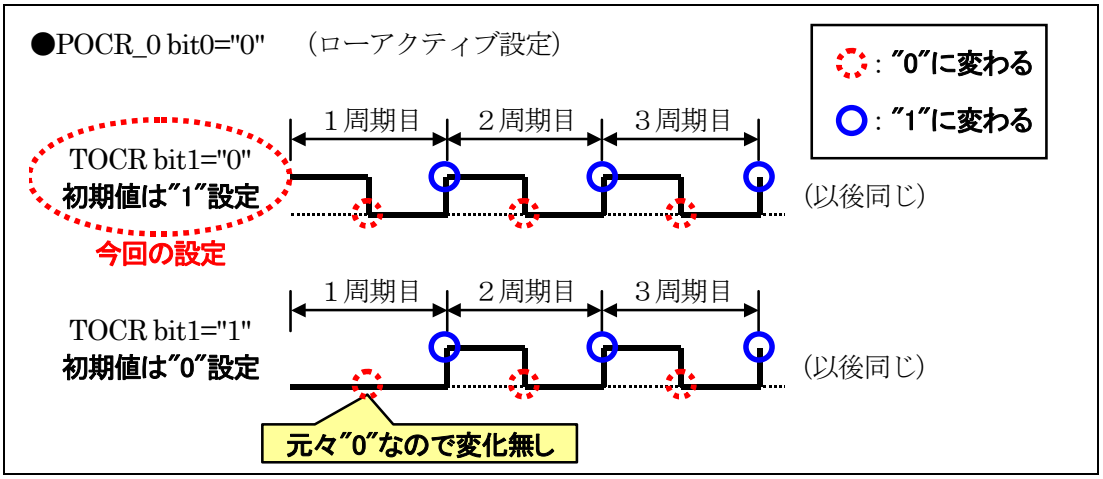
PWM 出力端子の出力レベルを設定します。PWM モードアウトプットレベルコントロールレジスタ(POCR)で設定します。このレジスタは 2 つあり、タイマ Z のチャネルによって番号が異なります。
・タイマ Z チャネル 0…PWM モードアウトプットレベルコントロールレジスタ_0(POCR_0)
・タイマ Z チャネル 1…PWM モードアウトプットレベルコントロールレジスタ_1(POCR_1)

PWM モードアウトプットレベルコントロールレジスタ_0(POCR_0)							
7	6	5	4	3	2	1	0
—	—	—	—	—	FTDOD0(P63) の出力レベル 0:ローアクティブ 1:ハイアクティブ	FTDOC0(P62) の出力レベル 0:ローアクティブ 1:ハイアクティブ	FTDOB0(P61)の 出力レベル 0:ローアクティブ 1:ハイアクティブ
					0	**0**	**0**

P61 端子を例にします。ハイアクティブとは、下記のような波形です。タイマアウトプットコントロールレジスタ(TOCR)の値によって、1 周期目の最初のレベルが"0"か"1"かになります。TOCR の値が"1"の場合、1 周期目はすべて"1"となります(下図)。



ローアクティブとは、ハイアクティブのときと比べ波形を反転させることです。初期出力値の設定も反転します(下図)。



今回は、ローアクティブ(POCR_0 の該当ビット"0")、初期出力は"0"(ローアクティブなので出力は"1")の設定にします。

プログラムは下記のようになります。

```
        POCR_0 = 0x00;
```

13.7.7 周期の設定

ジェネラルレジスタ A (GRA)に周期を設定します。このレジスタは 2 つあり、タイマ Z のチャネルによって番号が異なります。
・タイマ Z チャネル 0…ジェネラルレジスタ A_0 (GRA_0)
・タイマ Z チャネル 1…ジェネラルレジスタ A_1 (GRA_1)

今回は、周期 16[ms]に設定します。GRA_0 に設定する値は、下記のようになります。

GRA_0＝設定したい周期／TCR_0 で設定した 1 パルス幅－1
＝16[ms] ／ 271.27[ns] － 1
＝$(16\times10^{-3})/(271.27\times10^{-9})$ － 1
＝58982.4 - 1
＝58981.4
≒58981　　※小数点の指定はできないので、四捨五入します。

プログラムは下記のようになります。

```
        GRA_0 = 58981;
```

13.7.8 ON幅の設定

タイマ Z のチャネル 0 を使用する場合、各端子とジェネラルレジスタ B_0、C_0、D_0(GRB_0、GRC_0、GRD_0)の関係は下記のようになります。

・FTIOB0 端子(P61)の ON 幅は、ジェネラルレジスタ B_0(GRB_0)に設定します。
・FTIOC0 端子(P62)の ON 幅は、ジェネラルレジスタ C_0(GRC_0)に設定します。
・FTIOD0 端子(P63)の ON 幅は、ジェネラルレジスタ D_0(GRD_0)に設定します。

例えば、FTIOB0 端子(P61)の ON 幅を 8[ms]に設定するとします。GRB_0 に設定する値は、下記のようになります。

GRB_0＝ON 幅／TCR_0 で設定した 1 パルス幅－1
＝8[ms] ／ 271.27[ns] － 1
＝$(8\times10^{-3})/(271.27\times10^{-9})$ － 1
＝29491.2 - 1
＝29490.2
≒29490　　※小数点の指定はできないので、四捨五入します。

プログラムは下記のようになります。

```
        GRB_0 = 29490;
```

GRC_0 や GRD_0 も同様に計算します。

13.7.9 タイマ出力の許可

端子に PWM 信号を出力するかどうかの設定をします。タイマアウトプットマスタイネーブルレジスタ(TOER)で設定します。

タイマアウトプットマスタイネーブルレジスタ(TOER)							
タイマ Z チャネル 1				タイマ Z チャネル 0			
7	6	5	4	3	2	1	0
FTIOD1(P67)端子の出力を許可するなら"0" (初期値は"1")	FTIOC1(P66)端子の出力を許可するなら"0" (初期値は"1")	FTIOB1(P65)端子の出力を許可するなら"0" (初期値は"1")	FTIOA1(P64)端子の出力を許可するなら"0" (初期値は"1")	FTIOD0(P63)端子の出力を許可するなら"0" (初期値は"1")	FTIOC0(P62)端子の出力を許可するなら"0" (初期値は"1")	FTIOB0(P61)端子の出力を許可するなら"0" (初期値は"1")	FTIOA0(P60)端子の出力を許可するなら"0" (初期値は"1")
1	1	1	1	**0**	**0**	**0**	1

いつもの設定とは逆で、"0"で許可、"1"が禁止ですので気をつけます。プログラムは下記のようになります。

```
TOER = 0xf1;
```

13.7.10 カウンタ動作開始

タイマカウンタ_0(TCNT_0)を動作させるかの設定です。タイマスタートレジスタ(TSTR)で設定します。

タイマスタートレジスタ(TSTR)							
7	6	5	4	3	2	1	0
–	–	–	–	–	–	TCNT_1 を動作させるなら"1"	TCNT_0 を動作させるなら"1"
						0	**1**

プログラムは下記のようになります。

```
TSTR = 0x01;
```

13.7.11 まとめ

```
TCR_0  = 0x22;          /* カウンタクロック設定     */
TPMR   = 0x07;          /* PWMモードの設定          */
TOCR   = 0x00;          /* 初期出力値設定           */
POCR_0 = 0x00;          /* 出力レベル設定           */
GRA_0  = 58981;         /* 周期の設定               */
GRB_0  = 0;             /* ON幅設定                 */
GRC_0  = 0;             /* ON幅設定                 */
GRD_0  = 0;             /* ON幅設定                 */
TOER   = 0xf1;          /* PWM出力の許可            */
TSTR   = 0x01;          /* タイマZ ch0 スタート     */
```

※ON 幅の設定は 0 とします。

13.8 PWMモードの動作

13.8.1 設定内容

ジェネラルレジスタ A_0、B_0、C_0、D_0(GRA_0、GRB_0、GRC_0、GRD_0)には、下記のように設定したとします。

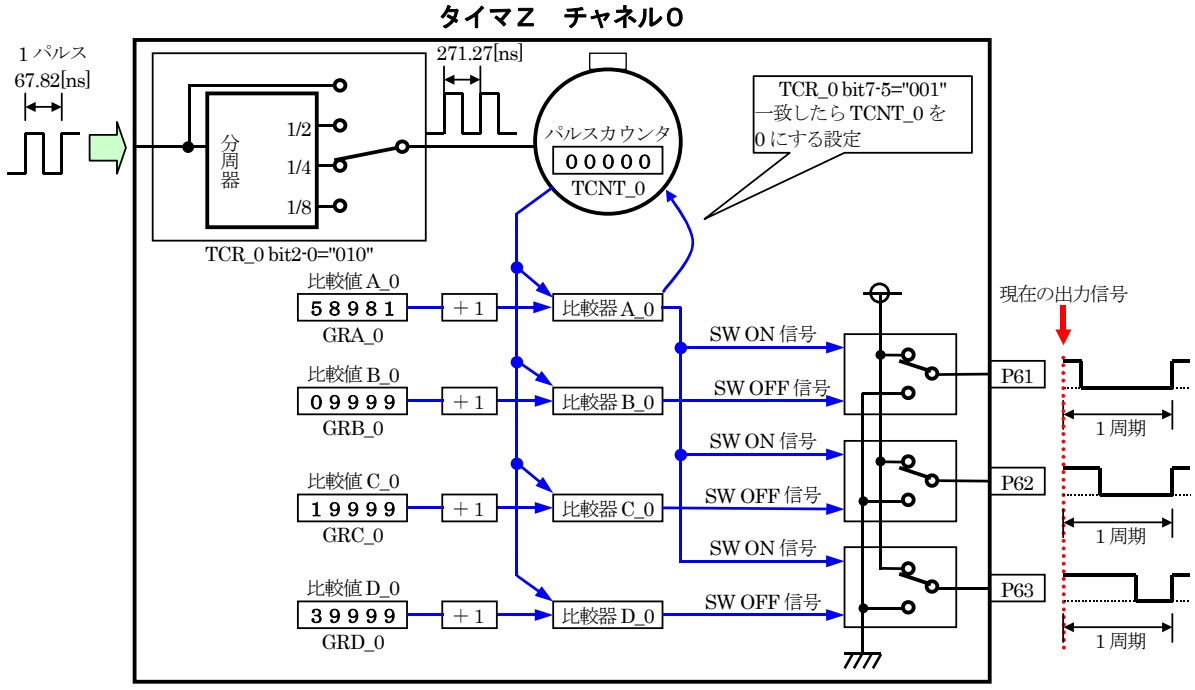
```
GRA_0  = 58981;
GRB_0  = 9999;
GRC_0  = 19999;
GRD_0  = 39999;
```

13.8.2 スタート

最初、各端子は"1"出力です。
タイマコントロールレジスタ_0(TCR_0)の bit2～0 には、"010"を設定してますので、タイマカウンタ_0(TCNT_0)には、φ/4 の間隔でパルスが送られてきます。計算すると

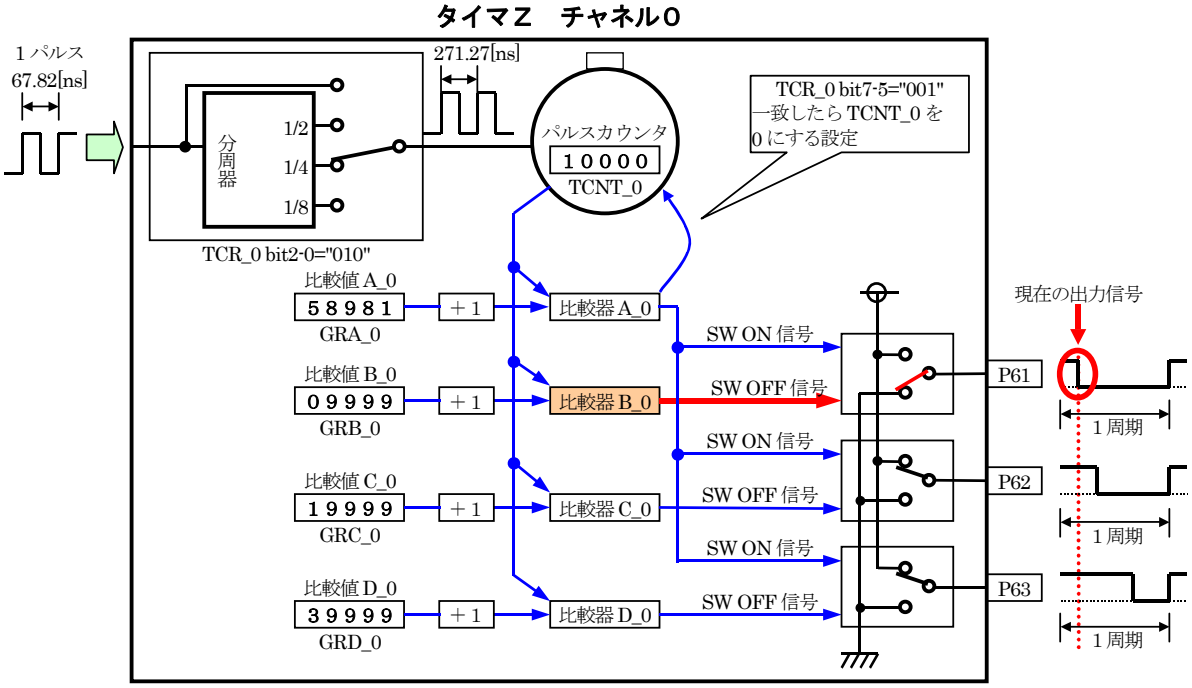
パルス幅＝1/(φ/4)＝1/(14.7456×10⁶/4)＝271.27[ns]

よって、タイマカウンタ_0(TCNT_0)は、271.27[ns]ごとに増えていくことになります。タイマカウンタ_0(TCNT_0)の合計によって、時間が分かります。



13.8.3 TCNT_0 とGRB_0 が一致

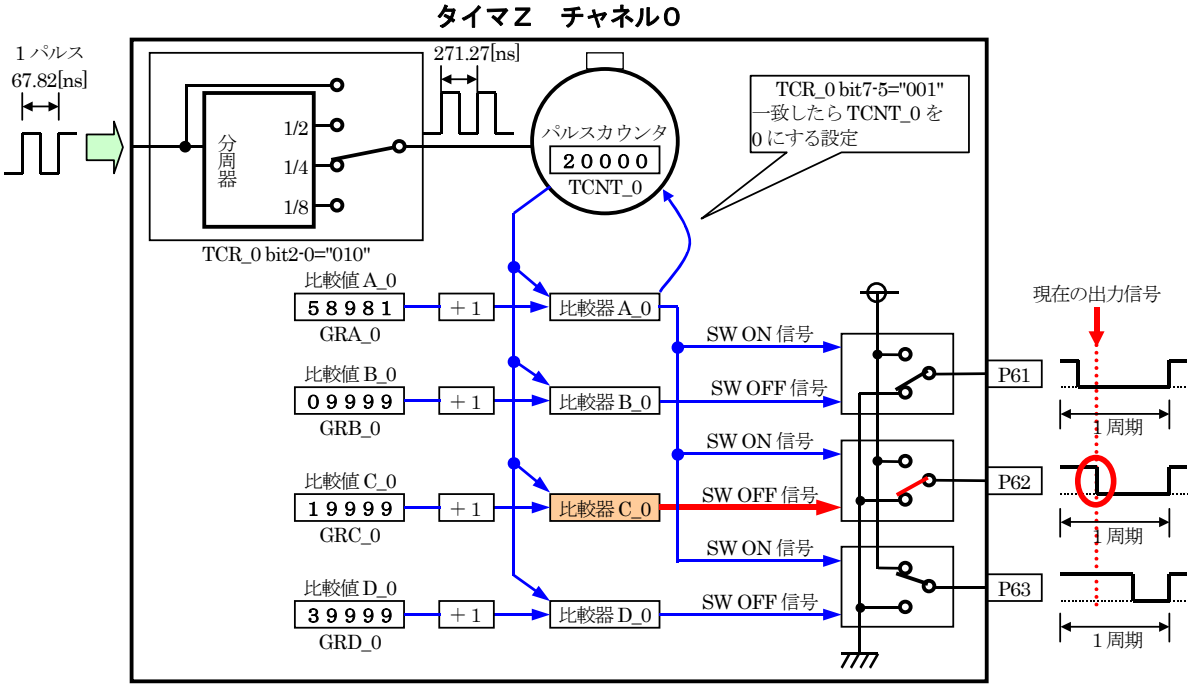
タイマカウンタ_0（TCNT_0）の値が 10000 になりました。
TCNT_0＝GRB_0＋1
が成り立ったので、比較器 B_0 が反応して、OFF 信号を出力します。P61 端子の出力が"0"になります。
1 あたり、271.27[ns]なので、P61 端子は下記の時間分、"1"になります。
$271.27 \times 10^{-9} \times 10000 = 2.71$[ms]



ちなみに、TCNT_0 と GRB_0 を比較するとき、GRB_0 の値は、1つ分足された値で比較されます。そのため、GRB_0 には、計算値より1つ小さい値をセットしておきます。GRA_0～GRD_0、GRA_1～GRD_1 すべて同様です。

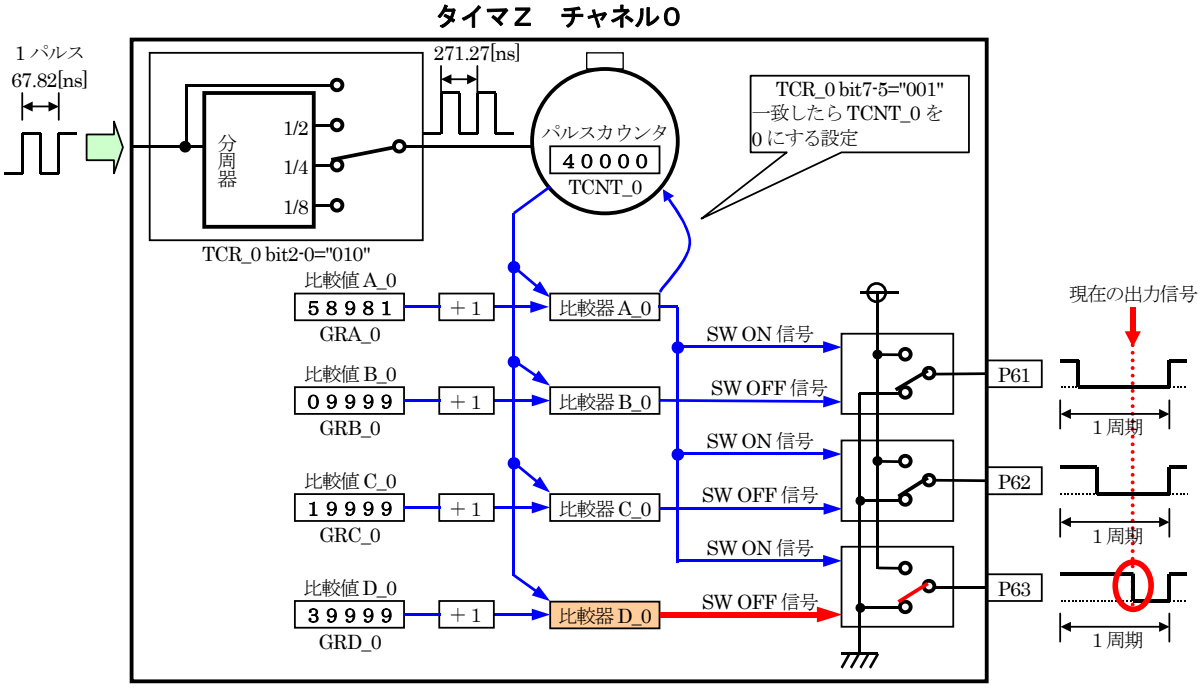
13.8.4 TCNT_0 とGRC_0 が一致

タイマカウンタ_0（TCNT_0）の値が 20000 になりました。
TCNT_0＝GRC_0＋1
が成り立ったので、比較器 C_0 が反応して、OFF 信号を出力します。P62 端子の出力が"0"になります。
1 あたり、271.27[ns]なので、P62 端子は下記の時間分、"1"になります。
$271.27 \times 10^{-9} \times 20000 = 5.43$[ms]



13.8.5 TCNT_0 とGRD_0 が一致

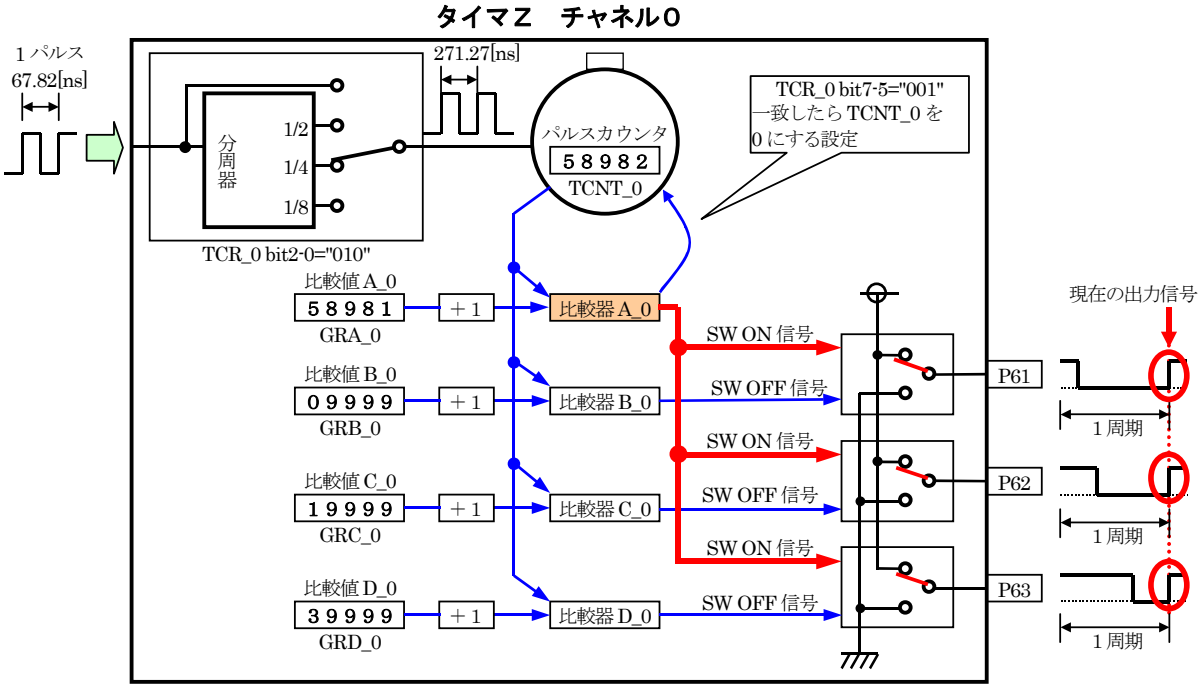
タイマカウンタ_0（TCNT_0）の値が 40000 になりました。
TCNT_0＝GRD_0＋1
が成り立ったので、比較器 D_0 が反応して、OFF 信号を出力します。P63 端子の出力が"0"になります。
1 あたり、271.27[ns]なので、P63 端子は下記の時間分、"1"になります。
$271.27 \times 10^{-9} \times 40000 = 10.85$[ms]



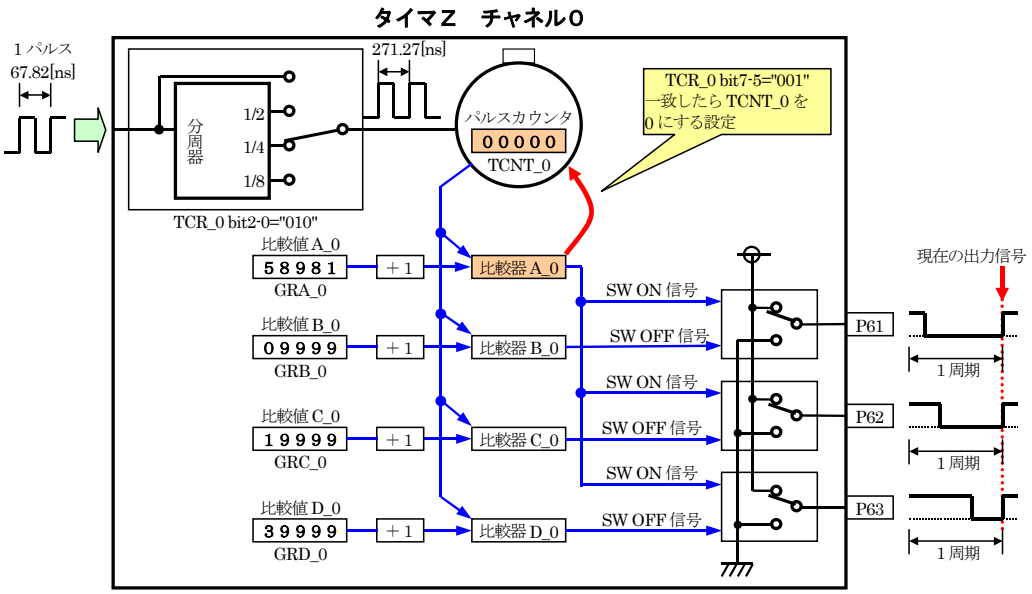
13.8.6 TCNT_0 とGRA_0 が一致

タイマカウンタ_0（TCNT_0）の値が 58982 になりました。
TCNT_0＝GRA_0＋1
が成り立ったので、比較器 A_0 が反応して、ON 信号を出力します。P61 端子、P62 端子、P63 端子の出力が一斉に"1"になります。
1 あたり、271.27[ns]なので、波形が ON から次に ON になるまでの間隔は下記のようになります。
$271.27\times10^{-9}\times58982=15.999[ms]\fallingdotseq16.00[ms]$

要は、周期が 16.00[ms]ということです。



そして、タイマカウンタ_0（TCNT_0）が 58982 になった瞬間、タイマコントロールレジスタ_0（TCR_0）の bit7～5 の設定により、TCNT_0 が 0 になります。



このようにして、3 端子から PWM 波形が繰り返し、出力され続けます。

13.8.7 0%出力にしたい場合

P61 端子を例に説明します。
0%出力にしたい場合、ON 幅を 0 にするだけです。GRB_0 を 0 にします。
ただし、忘れてはならない事柄があります。TCNT_0 と GRB_0 を比較するとき、GRB_0 の値は、1つ分足された値で比較されます。そのため、計算は下記のようになります。

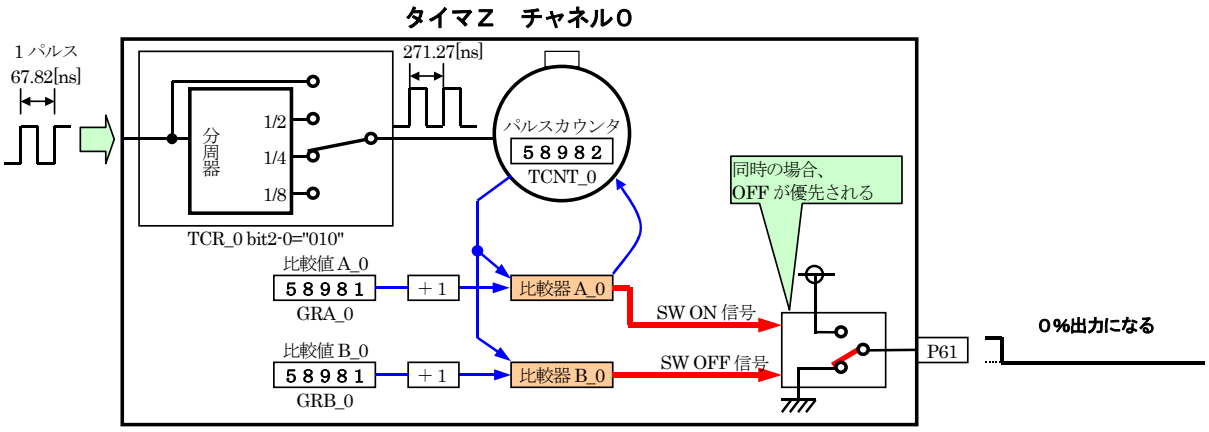
TCNT_0＝GRB_0 + 1
＝ 0 + 1
＝ 1

となり、必ず 1 カウント分である 271.27[ns]は、ON になってしまいます。

タイマ Z を PWM モードで使用するとき、下記のような決まりがあります。

TCNT_0＝GRB_0+1（端子を"0"にするタイミング） TCNT_0＝GRA_0+1（端子を"1"にするタイミング） が同時の場合、**端子を"0"にすることが優先されます**。

そのため、0%出力にしたい端子のジェネラルレジスタには、ジェネラルレジスタ A_0（GRA_0）と同じ値を設定します。



13.8.8 100%出力にしたい場合

P61 端子を例に説明します。

100%出力にしたい場合、ON 幅を 100%にするだけです。といって、周期と同じ値を設定すると、前記の通り、0%になってしまいます。

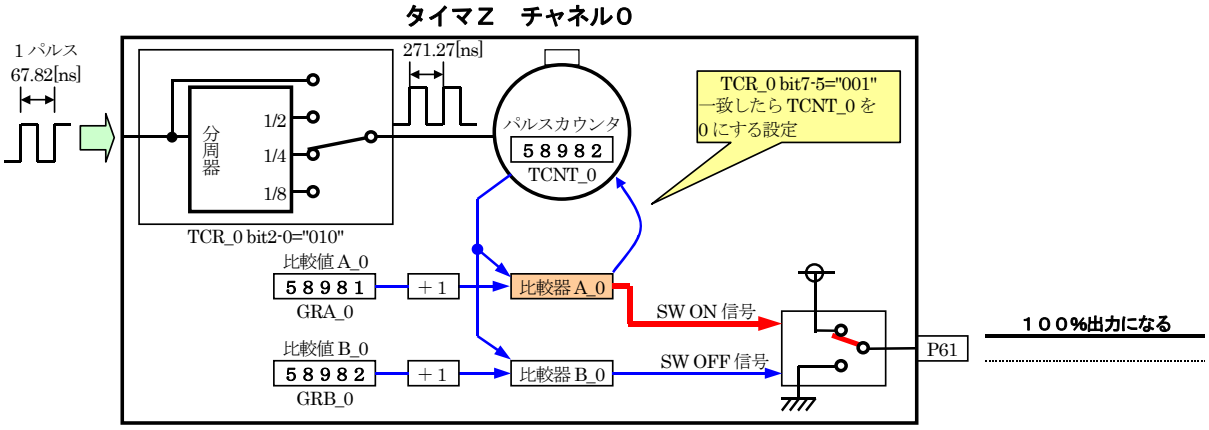
100%出力にしたいとき、ジェネラルレジスタ B_0（GRB_0）は下記の値を設定します。

GRB_0 ＝ GRA_0＋1
＝ 58981 ＋1
＝ 58982

TCNT_0 が増えていき、58982 の値になると、下記の条件が成り立ちます。

TCNT_0＝GRA_0 + 1

ON 信号が出力され、端子は"1"になります。そして、**TCR_0 の bit7～5 の設定により TCNT_0 は 0 になります**。端子を"0"にする条件である「TCNT_0＝GRB_0 ＋ 1」になることはありません。そのため、100%出力になるのです。



13.8.9 まとめ

●カウンタが+1 する時間の設定

タイマコントロールレジスタ_0（TCR_0）に、タイマカウンタ_0（TCNT_0）が 1 つ増える時間を設定します。
0x20…周期が 4.44[ms]以下の場合（＋1する時間は、67.72[ns]ごと）
0x21…周期が 8.89[ms]以下の場合（＋1する時間は、135.63[ns]]ごと）
0x22…周期が 17.78[ms]以下の場合（＋1する時間は、271.27[ns]ごと）
0x23…周期が 35.56[ms]以下の場合（＋1する時間は、542.53[ns]ごと）

今回は、周期を 16[ms]にするので、0x22 を設定します。

●周期の設定

PWM 波形の周期を設定します。今回は、周期を 16[ms]に設定します。
GRA_0＝設定したい周期／TCR_0 で設定した 1 パルス幅－1
GRA_0＝(16×10^{-3})／(271.27×10^{-9})－1
＝(16×10^{-3})／(271.27×10^{-9}) － 1
≒58981　　※小数点の指定はできないので、四捨五入します。

●ON 幅の設定

ON 幅を設定します。
・FTIOB0 端子(P61)の ON 幅は、ジェネラルレジスタ B_0（GRB_0）に設定します。
・FTIOC0 端子(P62)の ON 幅は、ジェネラルレジスタ C_0（GRC_0）に設定します。
・FTIOD0 端子(P63)の ON 幅は、ジェネラルレジスタ D_0（GRD_0）に設定します。

例えば、P61 端子の ON 幅を 1[ms]にするとき
GRB_0＝ON 幅／TCR_0 で設定した 1 パルス幅－1
＝(1×10^{-3})／(271.27×10^{-9})－1
＝3686.4 - 1
＝3685.4
≒**3685**

※ON 幅を 0%出力するとき
GRA_0 と同じ値を設定します。
例） P61 端子を 0%出力にするとき
GRB_0＝GRA_0＝**58981**

※100%出力するとき
GRA_0 より1つ大きい値を設定します。
例） P61 端子を 100%出力にするとき
GRB_0＝GRA_0 + 1＝58981+1＝**58982**

13.9 プログラムの解説

13.9.1 タイマZチャネル 0 の初期設定

init 関数で、I/O ポートの入出力設定終了後、タイマ Z のチャネル 0 を PWM モードで使用する設定にします。端子は、P61、P62、P63 の 3 つを PWM 出力にします。

ON 幅は、main 関数で設定することとして、ここでは 0 としておきます。

```
 59 :      /* タイマ Z ch0 PWM モード */
 60 :      TCR_0  = 0x22;                  /* カウンタクロック設定      */
 61 :      TPMR   = 0x07;                  /* PWM モードの設定          */
 62 :      TOCR   = 0x00;                  /* 初期出力値設定            */
 63 :      POCR_0 = 0x00;                  /* 出力レベル設定            */
 64 :      GRA_0  = 58981;                 /* 周期の設定                */
 65 :      GRB_0  = 0;                     /* ON 幅の設定               */
 66 :      GRC_0  = 0;                     /* ON 幅の設定               */
 67 :      GRD_0  = 0;                     /* ON 幅の設定               */
 68 :      TOER   = 0xf1;                  /* PWM 出力の許可            */
 69 :      TSTR   = 0x01;                  /* タイマ Z ch0 スタート     */
```

13.9.2 dipsw_get関数

```
 77 :  unsigned char dipsw_get( void )
 78 :  {
 79 :      unsigned char sw;
 80 :
 81 :      sw  = PDR3;                     /* ディップスイッチ読み込み */
 82 :      sw &= 0x0f;
 83 :      return  sw;
 84 :  }
```

CPU ボードにある 4 ビットのディップスイッチを入力する関数です。

81 行で、ポート 3 の状態を読み込みます。

82 行で、下位 4 ビットのみ有効にします。ディップスイッチは、下位 4 ビットに繋がっています。上位 4 ビットは無効ですので、マスクして強制的に 0 にします。sw 変数は、ディップスイッチの状態に応じて、0～15 になります。

83 行で、sw 変数を戻り値として、関数を終了します。

13.9.3 main関数

```
 31 :  void main( void )
 32 :  {
 33 :      init();                                  /* マイコン機能の初期化      */
 34 :
 35 :      while( 1 ) {
 36 :          GRB_0 = (long)58982 * dipsw_get() / 15;      /* P61          */
 37 :          GRC_0 = (long)0;                             /* P62          */
 38 :          GRD_0 = (long)0;                             /* P63          */
 39 :      }
 40 :  }
```

36 行で、ディップスイッチの値によりジェネラルレジスタ B_0(GRB_0)へ代入する数値を決めて ON 幅を可変しています。ディップスイッチの値と ON 幅をまとめておきます。

ディップスイッチの値	GRB_0 の値	ON 幅
0	58982 * 0 / 15 = 0	0% ※
1	58982 * 1 / 15 = 3932	6.6%
2	58982 * 2 / 15 = 7864	13.3%
3	58982 * 3 / 15 = 11796	20.0%
4	58982 * 4 / 15 = 15729	26.7%
5	58982 * 5 / 15 = 19661	33.3%
6	58982 * 6 / 15 = 23593	40.0%
7	58982 * 7 / 15 = 27525	46.7%
8	58982 * 8 / 15 = 31457	53.3%
9	58982 * 9 / 15 = 35389	60.0%
10	58982 * 10 / 15 = 39321	66.7%
11	58982 * 11 / 15 = 43253	73.3%
12	58982 * 12 / 15 = 47186	80.0%
13	58982 * 13 / 15 = 51118	86.7%
14	58982 * 14 / 15 = 55050	93.3%
15	58982 * 15 / 15 = 58982	100%

※GRB_0 を 0 にしても、TCNT_0＝(GRB_0+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、TCNT_0 の 1 カウント分（271.26ns）だけ ON になります。

参考資料－36 行目の計算について

```
        GRB_0 = 58982* dipsw_get() / 15;      /* P61           */
```

36 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、
58982* dipsw_get()
が、計算されます。
ここで型に注目します。
・58982、**long 型**
・dipsw_get()の戻り値は、unsigned char 型
したがって、

58982* 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、
58982* 15 → 答え 884730
となります。こちらは問題有りません。

では、例えば 58982 が 10000 ならどうなるでしょうか。

```
        GRB_0 = 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。これを超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。

```
GRB_0 = (long)10000 * dipsw_get() / 15;      /* OK! */
```

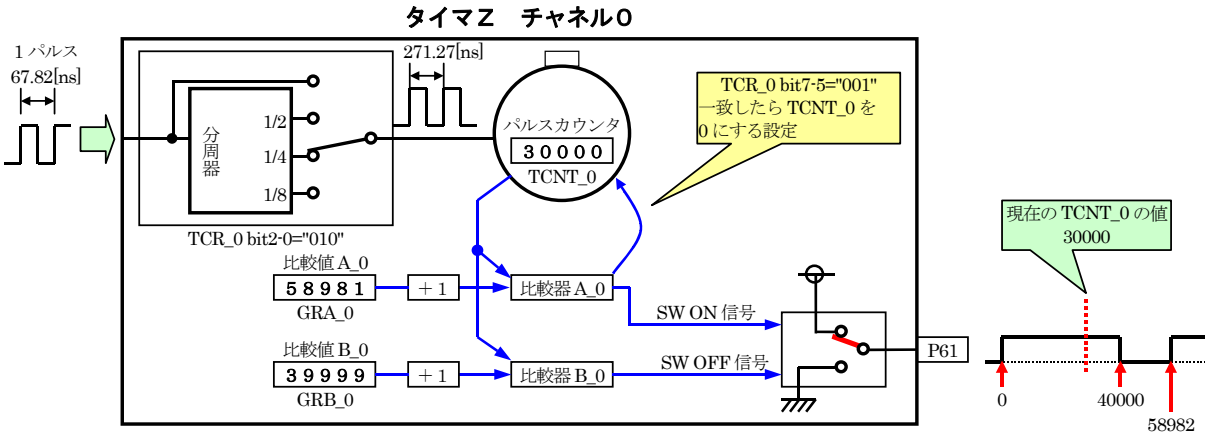
ちなみに、16進数の定数の型は、10進数とは違います。一緒に説明しておきます。

・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

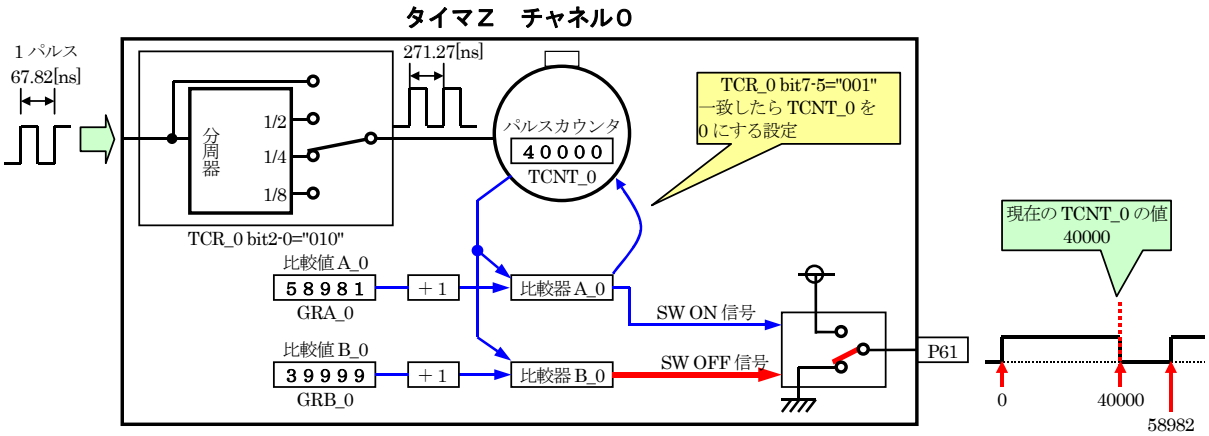
今回の設定値は 58982 なので、キャスト演算子は必要ありません。しかし、32767 以下の値に変更した場合を考えて、あらかじめ「(long)」をつけています。

13.10 問題点

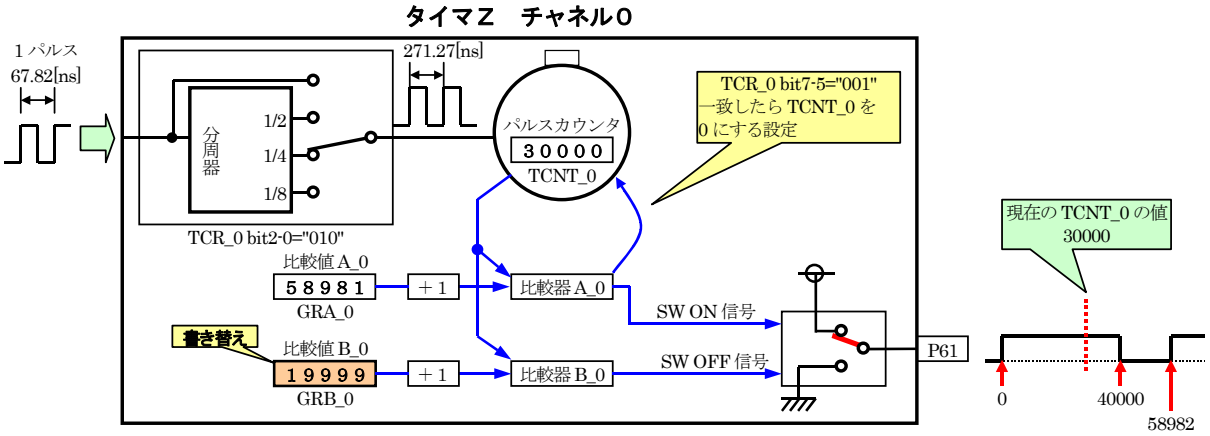
例えば、現在のタイマカウンタ_0(TCNT_0)の値が 30000、ジェネラルレジスタ B_0(GRB_0)の値が 39999 とします(下図)。



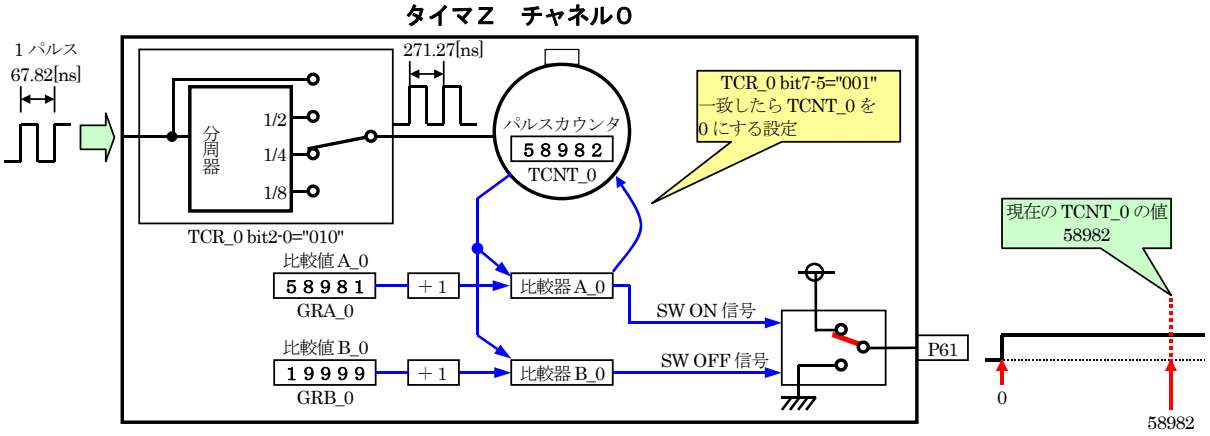
タイマカウンタ_0(TCNT_0)の値が 40000 になると、「TCNT0＝GRB_0＋1」が成り立ち、波形が"0"になります(下図)。



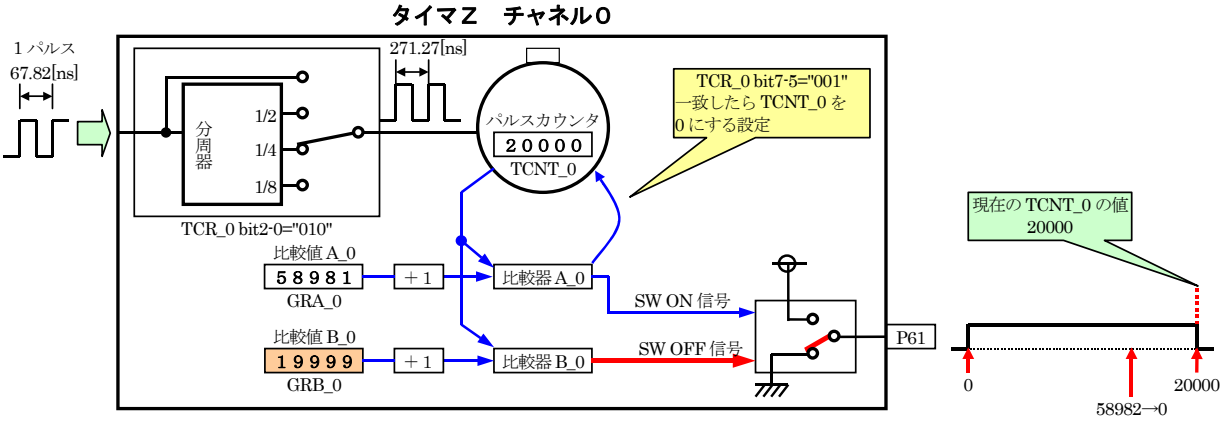
今度は、タイマカウンタ_0（TCNT_0）の値が 30000 のとき、**ジェネラルレジスタ B_0（GRB_0）の値を 19999 にプログラムで書き換えたとします**（下図）。



タイマカウンタ_0（TCNT_0）の値が増えていっても波形が"0"になりません。「TCNT0＝GRB_0＋1」が成り立つ前に、GRB_0 の値を TCNT_0 より小さい値に書き替えてしまったためです。結果、**1 周期分、100%出力となってしまいます**（下図）。



ただし、次の周期はタイマカウンタ_0（TCNT_0）の値が 20000 になると波形が"0"になります（下図）。このように、ジェネラルレジスタ B_0（GRB_0）を更新するタイミングによっては、1 周期"1"になることがあります。



対策としては GRB_0 の変更をするとき、TCNT_0 の値より GRB_0 の値を小さくしないようにします。

例） TCNT_0 の値が 10000、GRB_0 の値が 11000 のとき、GRB_0＝9000 の設定は不可

プログラムは、GRB_0 の値が TCNT_0 より小さいなら、GRB_0 の値を設定するようにします。大きいなら小さくなるまで待ちます(下記プログラム)。ただし、while 文を使うと、タイマと同じで処理がその行で止まってしまうので、止めたくない場合は、if 文を使うなど工夫してみてください。

```
    GRA_0 = 20000;    /* 周期 */
    GRB_0 = 10000;    /* ON 幅 */
    while( 1 ) {
        /* プログラム */
        while( TCNT_0 >= GRB_0+1 );  /* GRB_0+1 の値が TCNT_0 より大きくなるまで待つ */
        GRB_0 = 5000;                /* ON 幅の書き換え */
    }
```

GRB_0 の値が周期(GRA_0)と同じなら、上記の条件は起きないので無限ループになってしまいます。この場合は、1 周期"1"になる可能性がありますがチェックせずに書き替えるほかありません(できないことはないですがチェックが複雑になります)。

この問題は、**GRB0_の値を違う値に書き替えたとき 1 周期だけ発生する問題なので、この現象が発生しても問題ない機器を制御している場合は、対策する必要はありません**。

13.11 演習

13.11.1 P62 端子、P63 端子への出力

P61 端子の他、P62 端子、P63 端子へも、ディップスイッチの値により PWM 出力するようにしてみましょう。プログラム例は下記のようです。

```
void main( void )
{
    init();                              /* マイコン機能の初期化      */

    while( 1 ) {
        GRB_0 = (long)58982 * dipsw_get() / 15;     /* P61          */
        GRC_0 = (long)58982 * dipsw_get() / 15;     /* P62          */
        GRD_0 = (long)58982 * dipsw_get() / 15;     /* P63          */
    }
}
```

13.11.2 0%出力

サンプルプログラムは、0%出力でも 1 カウント分 ON になってしまいます。正真正銘の 0%になるようにプログラムを改造してみましょう。プログラム例は下記のようです。

```
void main( void )
{
    init();                             /* マイコン機能の初期化      */

    while( 1 ) {
        if( dipsw_get() == 0 ) {
            GRB_0 = 58981;                          /* P61 = "0"    */
        } else {
            GRB_0 = (long)58982 * dipsw_get() / 15; /* P61          */
        }
    }
}
```

13.11.3 タイマZのチャネル 1 を使用する場合

タイマ Z のチャネル 1 を使用して、PWM 出力してみましょう。プログラム例は下記のようです。

```
void main( void )
{
    init();                             /* マイコン機能の初期化      */

    while( 1 ) {
        GRB_1 = (long)58982 * dipsw_get() / 15;     /* P61          */
        GRC_1 = (long)0;                            /* P62          */
        GRD_1 = (long)0;                            /* P63          */
    }
}

void init( void )
{
    /* I/O ポートの入出力設定 */
    PCR1 = 0xff;
    PCR2 = 0xfd;                        /* 通信ビット P22:TxD P21:RxD*/
    PCR3 = 0xf0;                        /* 基板上のディップスイッチ */
    PCR5 = 0xff;
    PCR6 = 0xff;                        /* LED 基板                  */
    PCR7 = 0xff;
    PCR8 = 0xff;

    /* タイマ Z ch0 PWM モード */
    TCR_1  = 0x22;                      /* カウンタクロック設定      */
    TPMR   = 0x70;                      /* PWM モードの設定          */
    TOCR   = 0x00;                      /* 初期出力値設定            */
    POCR_1 = 0x00;                      /* 出力レベル設定            */
    GRA_1  = 58981;                     /* 周期の設定                */
    GRB_1  = 0;                         /* ON 幅の設定               */
    GRC_1  = 0;                         /* ON 幅の設定               */
    GRD_1  = 0;                         /* ON 幅の設定               */
    TOER   = 0x1f;                      /* PWM 出力の許可            */
    TSTR   = 0x02;                      /* タイマ Z ch0 スタート     */
}
```

GRA_0 を GRA_1 にするなど、チャネル番号をへ変更するのはもちろんですが、チャネル 0、チャネル 1 共通レジスタの設定を変更するところがポイントです。

14. 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

14.1 概要

リセット同期 PWM モードは、タイマ Z のチャネル 0 とチャネル 1 を組み合わせて使用することにより、正相と逆相が 1 組になった PWM 波形を 3 組出力することができます。

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。

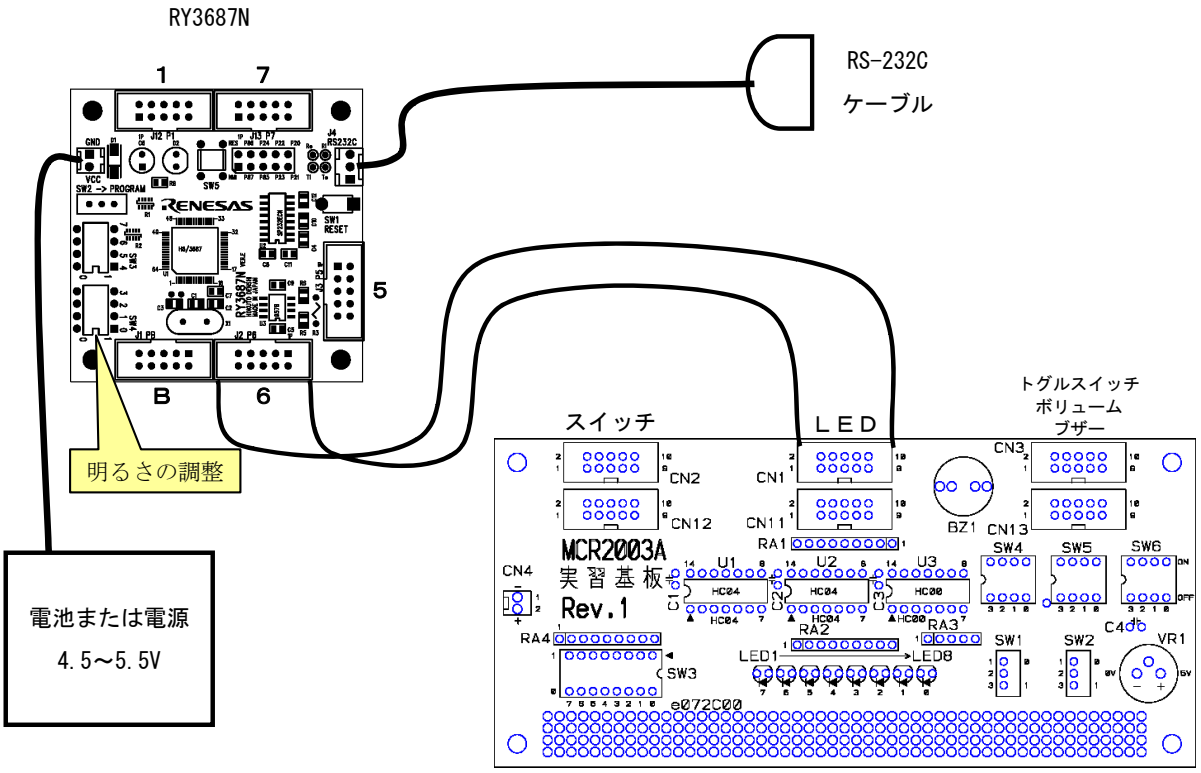
マイコンのポートは下記を使用します。

・ポート 6 の bit1～bit7・・・LED へ PWM 出力

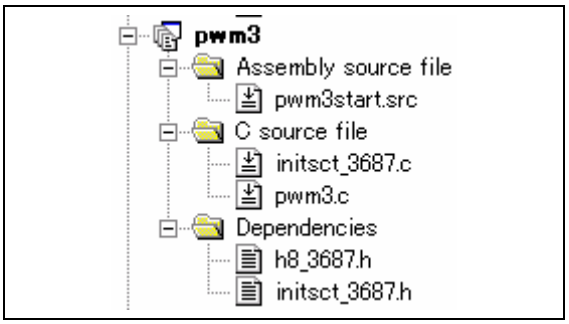
14.2 接続

・CPU ボードのポート 6 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。



14.3 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_3687.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。H8/3687F の内蔵周辺機能の初期化も行います。
4	h8_3687.h	H8/3687F 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_3687.h	initsct_3687.c のヘッダファイルです。

14.4 プログラム「pwm3.c」

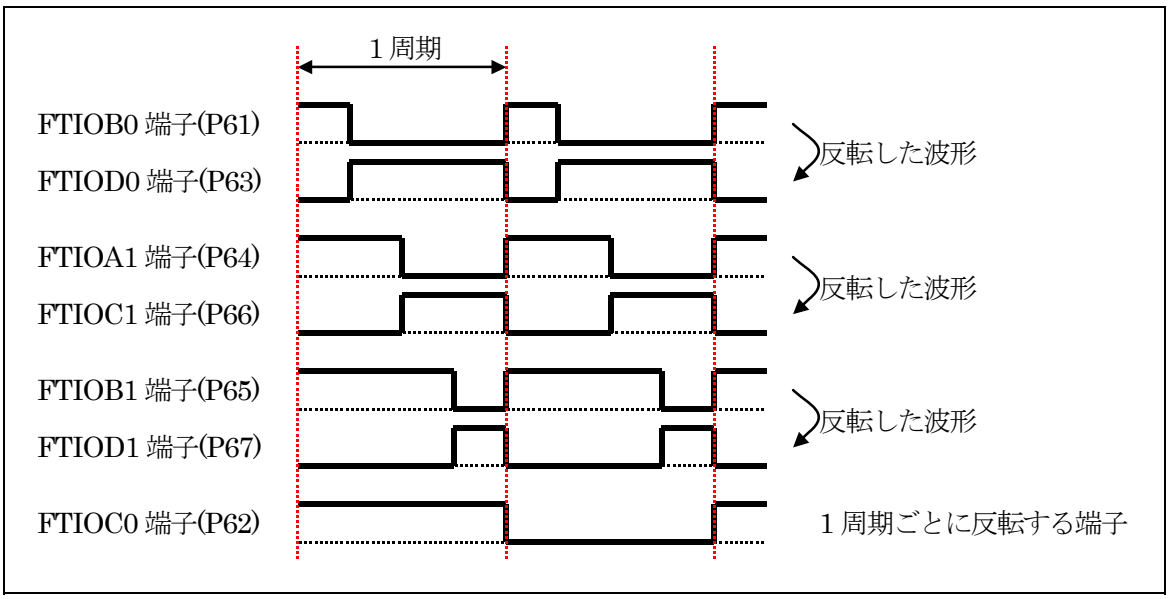
```
 1 :  /************************************************************************/
 2 :  /* リセット同期PWMモード "pwm3.c"                                       */
 3 :  /*                          2007.04 ジャパンマイコンカーラリー実行委員会 */
 4 :  /************************************************************************/
 5 :  /*
 6 :  入力：P33-P30(CPUボード上のディップスイッチ)
 7 :  出力：P67-P60(LEDなど)
 8 :
 9 :  ポート６にリセット同期PWMモードで生成したPWM信号を出力します。
10 :  */
11 :
12 :  /*====================================*/
13 :  /* インクルード                       */
14 :  /*====================================*/
15 :  #include    <machine.h>
16 :  #include    "h8_3687.h"
17 :
18 :  /*====================================*/
19 :  /* プロトタイプ宣言                   */
20 :  /*====================================*/
21 :  void init( void );
22 :  unsigned char dipsw_get( void );
23 :
24 :  /*====================================*/
25 :  /* グローバル変数の宣言               */
26 :  /*====================================*/
27 :
28 :  /************************************************************************/
29 :  /* メインプログラム                                                     */
30 :  /************************************************************************/
31 :  void main( void )
32 :  {
33 :      init();                             /* マイコン機能の初期化      */
34 :
35 :      while( 1 ) {
36 :          GRB_0 = (long)58982 * dipsw_get() / 15;     /* P61,P63      */
37 :          GRA_1 = (long)0;                            /* P64,P66      */
38 :          GRB_1 = (long)0;                            /* P65,P67      */
```

```
39 :     }
40 : }
41 :
42 : /************************************************************************/
43 : /* H8/3687F 内蔵周辺機能　初期化                                        */
44 : /************************************************************************/
45 : void init( void )
46 : {
47 :     /* I/Oポートの入出力設定 */
48 :     PCR1 = 0xff;
49 :     PCR2 = 0xfd;                        /* 通信ビットP22:TxD P21:RxD*/
50 :     PCR3 = 0xf0;                        /* 基板上のディップスイッチ */
51 :     PCR5 = 0xff;
52 :     PCR6 = 0xff;                        /* LED基板                  */
53 :     PCR7 = 0xff;
54 :     PCR8 = 0xff;
55 :     /* ポートBは、入力専用なので入出力設定はありません。          */
56 :     /* また、A/D変換器のADCSRによって指定された1ビットだけは、   */
57 :     /* 入力ポートとしては使えません。                             */
58 :
59 :     /* タイマZ ch0,ch1 リセット同期PWMモード */
60 :     TCR_0  = 0x22;                      /* カウンタクロック設定     */
61 :     TFCR   = 0x01;                      /* リセット同期PWMモード    */
62 :     TOCR   = 0x00;                      /* PWM信号の出力設定        */
63 :     TCNT_0 = 0;                         /* カウンタクリア           */
64 :     GRA_0  = 58981;                     /* 周期の設定               */
65 :     GRB_0  = 0;                         /* P61,P63                  */
66 :     GRA_1  = 0;                         /* P64,P66                  */
67 :     GRB_1  = 0;                         /* P65,P67                  */
68 :     TOER   = 0x01;                      /* 出力端子の設定           */
69 :     TSTR   = 0x01;                      /* タイマスタート           */
70 : }
71 :
72 : /************************************************************************/
73 : /* ディップスイッチ値読み込み                                           */
74 : /* 引数　なし                                                           */
75 : /* 戻り値 スイッチ値 0～15                                              */
76 : /************************************************************************/
77 : unsigned char dipsw_get( void )
78 : {
79 :     unsigned char sw;
80 :
81 :     sw  = PDR3;                         /* ディップスイッチ読み込み */
82 :     sw &= 0x0f;
83 :     return  sw;
84 : }
85 :
86 : /************************************************************************/
87 : /* End of file                                                          */
88 : /************************************************************************/
```

14.5 リセット同期PWMモード

14.5.1 リセット同期PWMモードとは？

リセット同期 PWM モードは、タイマ Z のチャネル 0 とチャネル 1 を組み合わせて使用することにより、正相と逆相が 1 組になった PWM 波形を 3 組出力するモードです。また、1 周期ごとに反転する端子が 1 端子あります。



14.5.2 PWM出力端子

リセット同期 PWM モードを使用したとき、下記のような端子から PWM 信号が出力されます。

端子名	ポート	内容
FTIOC0	P62	1 周期ごとに反転
FTIOB0	P61	PWM 出力 1
FTIOD0	P63	PWM 出力 1 の反転波形
FTIOA1	P64	PWM 出力 2
FTIOC1	P66	PWM 出力 2 の反転波形
FTIOB1	P65	PWM 出力 3
FTIOD1	P67	PWM 出力 3 の反転波形

14.6 リセット同期PWMモードの設定

14.6.1 設定手順

リセット同期 PWM を設定するには、下記のような手順で各レジスタを設定します。

(1) TSTR の STR0 ビットを 0 にクリアし、TCNT_0 のカウンタ動作を停止してください。リセット同期 PWM モードの設定は、TCNT_0 が停止した状態で行ってください。
(2) TCR の TPSC2～TPSC0 ビットでカウンタクロックを選択してください。外部クロックを選択した場合は、TCR の CKEG1、CKEG0 ビットにより外部クロックのエッジを選択してください。
(3) TCR の CCLR1、CCLR0 ビットによりカウンタクリア要因を GRA_0 を選択してください。
(4) TFCR の CMD1、CMD0 ビットでリセット同期 PWM モードを設定してください。FTIOB0～FTIOD0 および FTIOA1～FTIOD1 は自動的に PWM 出力端子になります。
(5) TOCR に H'00 を設定してください。
(6) TCNT_0 は H'0000 としてください。TCNT_1 は設定する必要はありません。
(7) GRA_0 は周期レジスタです。GRA_0 には周期を設定してください。GRB_0、GRA_1、GRB_1 には PWM 出力波形変化タイミングを設定してください。
(8) TOER でタイマ出力の許可／禁止を設定してください。
(9) TSTR の STR ビットを 1 にセットしてカウンタ動作を開始してください。

14.6.2 タイマカウンタ_0（TCNT_0）の動作停止

もし、タイマ Z のチャネル 0 を使っている場合は、タイマスタートレジスタ（TSTR）でタイマカウンタ_0（TCNT_0）の動作を停止します。リセット直後など使用していない場合は、元々停止しているので、何もしなくとも問題ありません。今回の演習では、何もしません。

タイマスタートレジスタ（TSTR）							
7	6	5	4	3	2	1	0
–	–	–	–	–	–	TCNT_1 を動作させるなら"1"	TCNT_0 を動作させるなら"1"
							0

14.6.3 カウンタクロックの選択

タイマコントロールレジスタ(TCR)の TPSC2～TPSC0 ビットでカウンタクロックを選択してください。外部クロックを選択した場合は、TCR の CKEG1、CKEG0 ビットにより外部クロックのエッジを選択してください。

TCR_0			内容
bit2	bit1	bit0	
0	0	0	φ でカウント 1 パルス＝1/(14.7456×10^6)＝67.72[ns] 最大 PWM 周期＝67.72×10^{-9}×65536＝4.44[ms]
0	0	1	φ /2 でカウント 1 パルス＝1/(14.7456×10^6/2)＝135.63[ns] 最大 PWM 周期＝135.63×10^{-9}×65536＝8.89[ms]
0	1	0	φ /4 でカウント 1 パルス＝1/(14.7456×10^6/4)＝271.27[ns] 最大 PWM 周期＝271.27×10^{-9}×65536＝17.78[ms]

0	1	1	φ/8 でカウント 1 パルス＝1/(14.7456×10^6/8)＝542.53[ns] 最大 PWM 周期＝542.53×10^{-9}×65536＝35.56[ms]
1	0	0	外部クロック FTIOA0(TCLK)端子から入力したパルスでカウント

今回、PWM 周期は 16[ms]にします。どの設定にすればよいのでしょうか。最大 PWM 周期が短い順に見ていきます。最大 PWM 周期が 16ms 以下の場合は、計ることができませんので設定できません。一番最初に 16ms 以上になったときが、設定する値です。

・φのとき、最大 PWM 周期は 4.44[ms] → 16ms 以下なので不可
・φ/2 のとき、最大 PWM 周期は 8.89[ms] → 16ms 以下なので不可
・φ/4 のとき、最大 PWM 周期は 17.78[ms] → 16ms 以上なので適合

よって、タイマコントロールレジスタ_0(TCR_0) bit2～0 の設定は、φ/4 である"010"にします。

タイマコントロールレジスタ_0(TCR_0)							
7	6	5	4	3	2	1	0
カウンタクリア 2～0			クロックエッジ 1～0		タイマプリスケーラ 2～0		
					0	**1**	**0**

14.6.4 カウンタクリア要因の選択

タイマコントロールレジスタ(TCR)の CCLR1、CCLR0 ビットによりカウンタクリア要因を GRA_0 を選択してください。

タイマコントロールレジスタ 0(TCR_0)							
7	6	5	4	3	2	1	0
カウンタクリア 2～0			クロックエッジ 1～0		タイマプリスケーラ 2～0		
0	**0**	**1**					

パルスカウンタをカウントするタイミングの設定と合わせて、プログラムは下記のようになります。

```
TCR_0 = 0x22;
```

14.6.5 リセット同期PWMモードの設定

タイマファンクションコントロールレジスタ(TFCR)の CMD1、CMD0 ビットでリセット同期 PWM モードを設定してください。FTIOB0～FTIOD0 および FTIOA1～FTIOD1 は自動的に PWM 出力端子になります。

タイマファンクションコントロールレジスタ(TFCR)							
7	6	5	4	3	2	1	0
—	外部クロック 入力セレクト	A/D トリガ エッジセレクト	外部トリガ ディスエーブル	出力レベル セレクト 1	出力レベル セレクト 0	コンビネーションモード 1～0 00:チャネル 0、1 は通常動作 **01:チャネル 0、1 を組み合わせ、 リセット同期 PWM モードで動作**	
						0	**1**

プログラムは下記のようになります。

```
        TFCR = 0x01;
```

14.6.6 TOCRの設定

タイマアウトプットコントロールレジスタ(TOCR)には、必ず 0x00 を設定します。リセット同期 PWM モードを使う上での決まり事です。

```
        TOCR = 0x00;
```

14.6.7 TCNT_0 の設定

タイマカウンタ_0(TCNT_0)には、0 を設定します。タイマカウンタ_1(TCNT_1)は設定する必要はありません。

```
        TCNT_0 = 0;
```

14.6.8 周期の設定

ジェネラルレジスタ A_0 (GRA_0)に周期を設定します。今回は、周期 16[ms]に設定するとします。GRA_0 に設定する値は、下記のようになります。

GRA_0＝設定したい周期／TCR_0 で設定した 1 パルス幅－1
　　＝16[ms] ／ 271.27[ns] － 1
　　＝$(16\times10^{-3})/(271.27\times10^{-9})$ － 1
　　＝58982.4 - 1
　　＝58981.4
　　≒58981　　※小数点の指定はできないので、四捨五入します。

プログラムは下記のようになります。

```
        GRA_0 = 58981;
```

14.6.9 ON幅の設定

各端子とジェネラルレジスタ B_0、C_0、D_0(GRB_0、GRC_0、GRD_0)の関係は下記のようになります。

端子名	ポート	内容	レジスタ
FTIOC0	P62	1 周期ごとに反転	
FTIOB0	P61	PWM 出力 1	GRB_0 で ON 幅を設定
FTIOD0	P63	PWM 出力 1 の反転波形	GRB_0 で OFF 幅を設定
FTIOA1	P64	PWM 出力 2	GRA_1 で ON 幅を設定
FTIOC1	P66	PWM 出力 2 の反転波形	GRA_1 で OFF 幅を設定
FTIOB1	P65	PWM 出力 3	GRB_1 で ON 幅を設定
FTIOD1	P67	PWM 出力 3 の反転波形	GRB_1 で OFF 幅を設定

例えば、FTIOB0 端子(P61)の ON 幅を 8[ms]に設定するとします。GRB_0 に設定する値は、下記のようになります。

GRB_0＝ON 幅／TCR_0 で設定した 1 パルス幅－1
＝8[ms] ／ 271.27[ns] － 1
＝$(8\times10^{-3})/(271.27\times10^{-9})$ － 1
＝29491.2 - 1
＝29490.2
≒29490　　※小数点の指定はできないので、四捨五入します。

プログラムは下記のようになります。

```
GRB_0 = 29490;
```

GRC_0 や GRD_0 も同様に計算します。

14.6.10 タイマ出力の許可／禁止の設定

タイマアウトプットマスタイネーブルレジスタ(TOER)でタイマ出力の許可／禁止を設定します。

タイマアウトプットマスタイネーブルレジスタ(TOER)							
7	6	5	4	3	2	1	0
FTIOD1(P67)端子の出力を許可するなら"0" (初期値は"1")	FTIOC1(P66)端子の出力を許可するなら"0" (初期値は"1")	FTIOB1(P65)端子の出力を許可するなら"0" (初期値は"1")	FTIOA1(P64)端子の出力を許可するなら"0" (初期値は"1")	FTIOD0(P63)端子の出力を許可するなら"0" (初期値は"1")	FTIOC0(P62)端子の出力を許可するなら"0" (初期値は"1")	FTIOB0(P61)端子の出力を許可するなら"0" (初期値は"1")	FTIOA0(P60)端子の出力を許可するなら"0" (初期値は"1")
0	**0**	**0**	**0**	**0**	**0**	**0**	1

いつもの設定とは逆で、"0"で許可、"1"が禁止ですので気をつけます。プログラムは下記のようになります。

```
TOER = 0x01;
```

14.6.11 カウンタ動作開始

タイマカウンタ_0(TCNT_0)を動作させるかの設定です。タイマスタートレジスタ(TSTR)で設定します。

タイマスタートレジスタ(TSTR)							
7	6	5	4	3	2	1	0
－	－	－	－	－	－	TCNT_1 を動作させるなら"1"	TCNT_0 を動作させるなら"1"
						0	**1**

プログラムは下記のようになります。

```
TSTR = 0x01;
```

14.7 リセット同期PWMモードの動作

14.7.1 設定内容

ジェネラルレジスタ A_0、B_0、C_0、D_0(GRA_0、GRB_0、GRC_0、GRD_0)には、下記のように設定したとします。

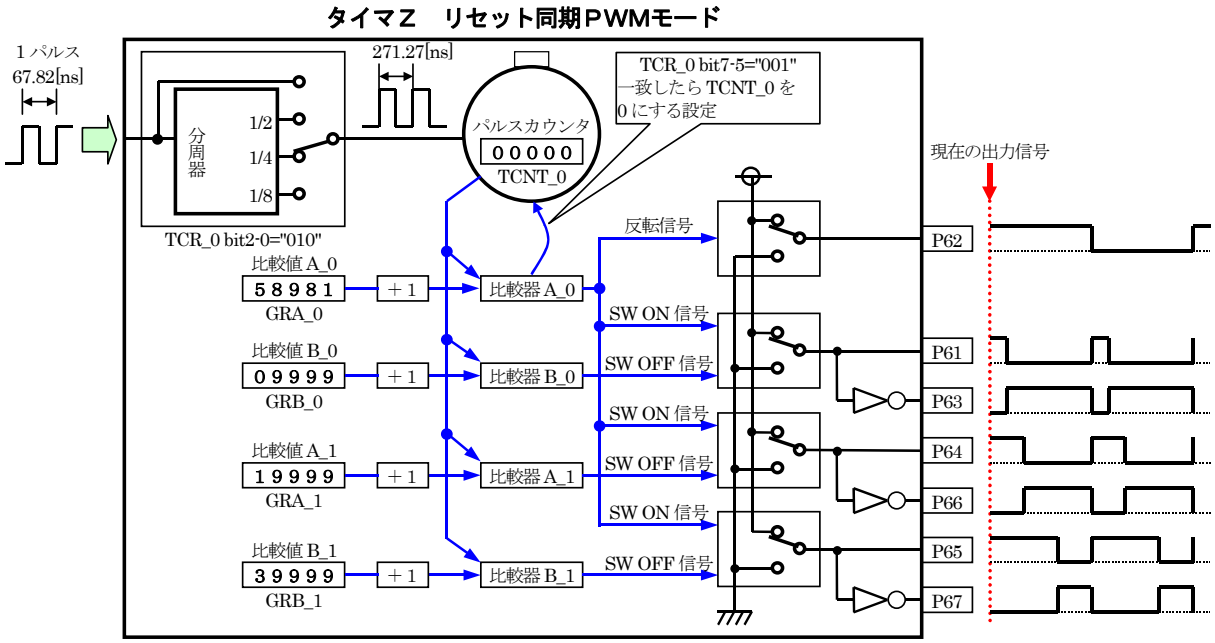
```
GRA_0  = 58981;
GRB_0  = 9999;
GRA_1  = 19999;
GRB_1  = 39999;
```

14.7.2 スタート

各端子は"1"出力です。
タイマコントロールレジスタ_0(TCR_0)の bit2～0 には、"010"を設定してますので、タイマカウンタ_0(TCNT_0)には、φ/4 の間隔でパルスが送られてきます。計算すると

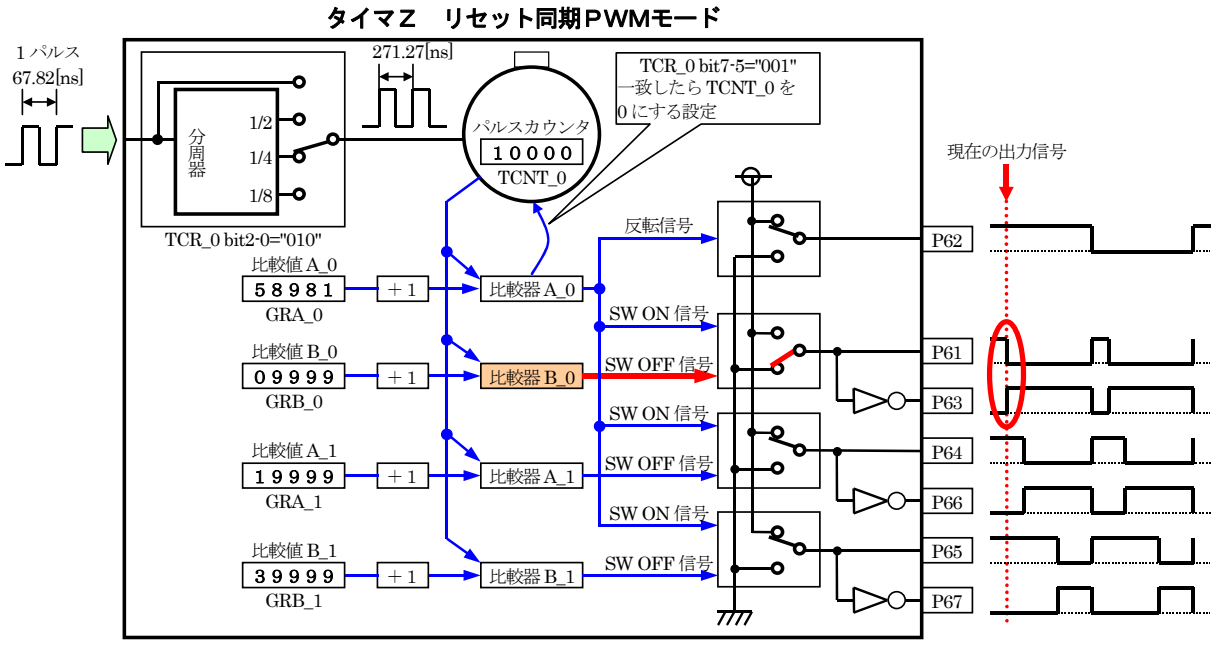
パルス幅＝1/(φ/4)＝1/(14.7456×10⁶/4)＝271.27[ns]

よって、タイマカウンタ_0(TCNT_0)は、271.27[ns]ごとに増えていくことになります。タイマカウンタ_0(TCNT_0)の合計によって、時間が分かります。



14.7.3 TCNT_0 とGRB_0 が一致

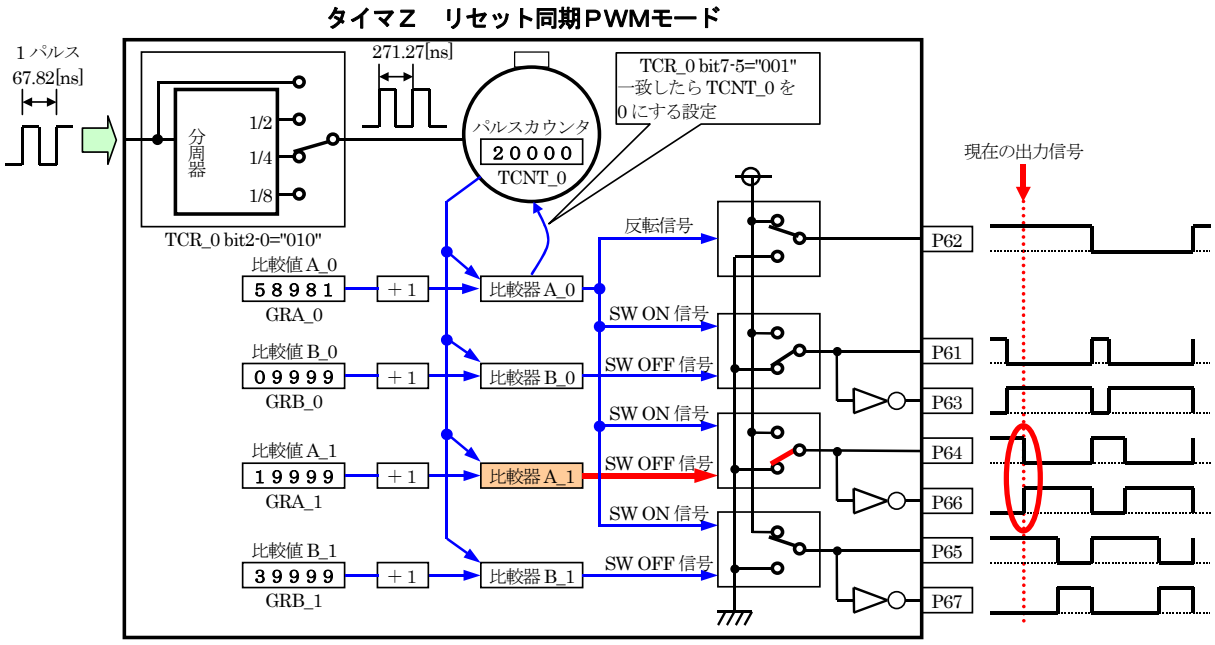
タイマカウンタ_0（TCNT_0）の値が 10000 になりました。
TCNT_0＝GRB_0＋1
が成り立ったので、比較器 B_0 が反応して、OFF 信号を出力します。P61 端子の出力が"0"になります。
1 あたり、271.27[ns]なので、P61 端子は下記の時間分、"1"になります。
$271.27 \times 10^{-9} \times 10000 = 2.71$[ms]
P63 端子は、反転した波形が出力されます。



ちなみに、TCNT_0 と GRB_0 を比較するとき、GRB_0 の値は、1つ分足された値で比較されます。そのため、GRB_0 には、計算値より1つ小さい値をセットしておきます。GRA_0～GRD_0、GRA_1～GRD_1 すべて同様です。

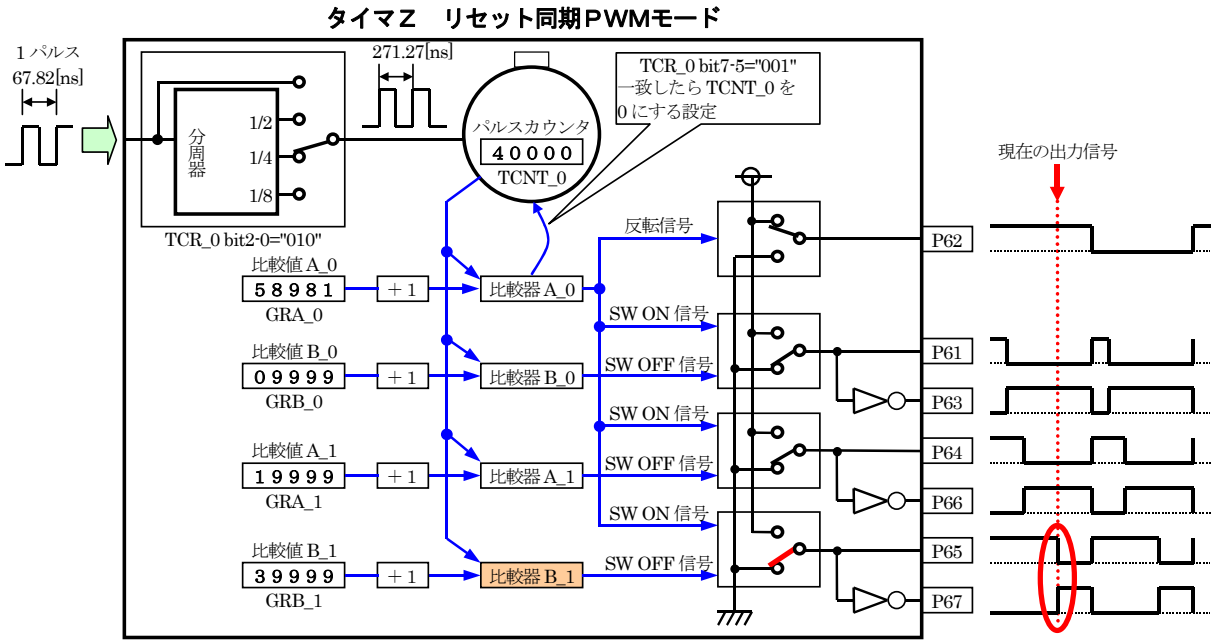
14.7.4 TCNT_0 とGRA_1 が一致

タイマカウンタ_0（TCNT_0）の値が 20000 になりました。
TCNT_0＝GRA_1＋1
が成り立ったので、比較器 A_1 が反応して、OFF 信号を出力します。P64 端子の出力が"0"になります。
1 あたり、271.27[ns]なので、P64 端子は下記の時間分、"1"になります。
$271.27 \times 10^{-9} \times 20000 = 5.43$[ms]
P66 端子は、反転した波形が出力されます。



14.7.5 TCNT_0 とGRB_1 が一致

タイマカウンタ_0（TCNT_0）の値が 40000 になりました。
TCNT_0＝GRB_1＋1
が成り立ったので、比較器 B_1 が反応して、OFF 信号を出力します。P65 端子の出力が"0"になります。
1 あたり、271.27[ns]なので、P65 端子は下記の時間分、"1"になります。
$271.27 \times 10^{-9} \times 40000 = 10.85$[ms]
P67 端子は、反転した波形が出力されます。



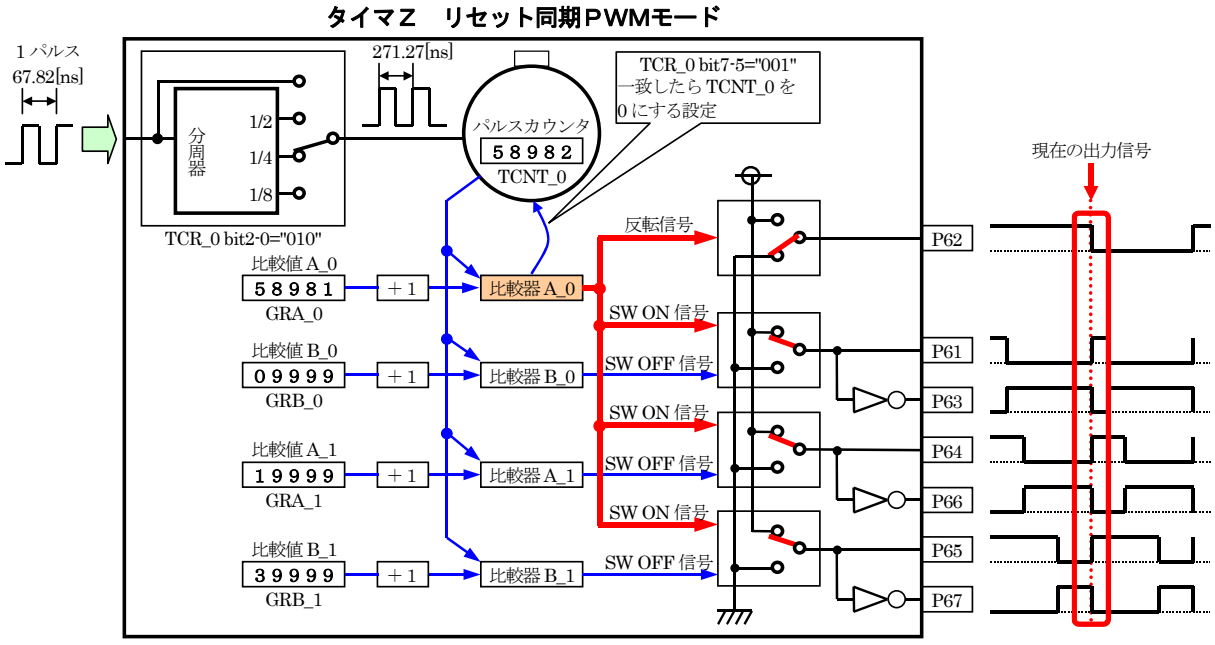
14.7.6 TCNT_0 とGRA_0 が一致

タイマカウンタ_0（TCNT_0）の値が 58982 になりました。
TCNT_0＝GRA_0＋1
が成り立ったので、比較器 A_0 が反応して、下記のようになります。

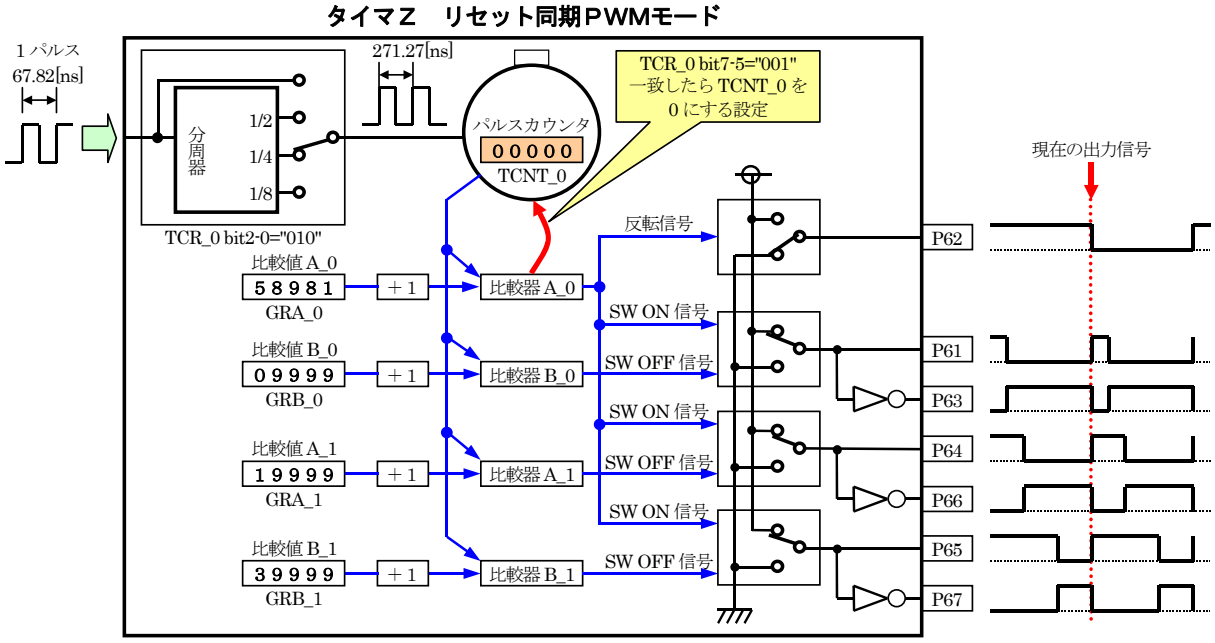
・P61 端子には ON 信号を出力します。P61 端子の出力が"1"になります。P63 端子は、反転した波形が出力されます。
・P64 端子には ON 信号を出力します。P64 端子の出力が"1"になります。P66 端子は、反転した波形が出力されます。
・P65 端子には ON 信号を出力します。P65 端子の出力が"1"になります。P67 端子は、反転した波形が出力されます。
・P62 端子には反転信号を出力します。P62 端子の出力が"0"になります。

1 あたり、271.27[ns]なので、上記動作をする間隔は下記のようになります。
$271.27\times10^{-9}\times58982=15.9999[ms]\fallingdotseq16.00[ms]$

要は、周期が 16.00[ms]ということです。



そして、タイマカウンタ_0(TCNT_0)が 58982 になった瞬間、タイマコントロールレジスタ_0(TCR_0)の bit7～5 の設定により、TCNT_0 が 0 になります。



14.7.7 0%出力にしたい場合

P61 端子を例に説明します。
0%出力にしたい場合、ON 幅を 0 にするだけです。GRB_0 を 0 にします。
ただし、忘れてはならない事柄があります。TCNT_0 と GRB_0 を比較するとき、GRB_0 の値は、1つ分足された値で比較されます。そのため、計算は下記のようになります。

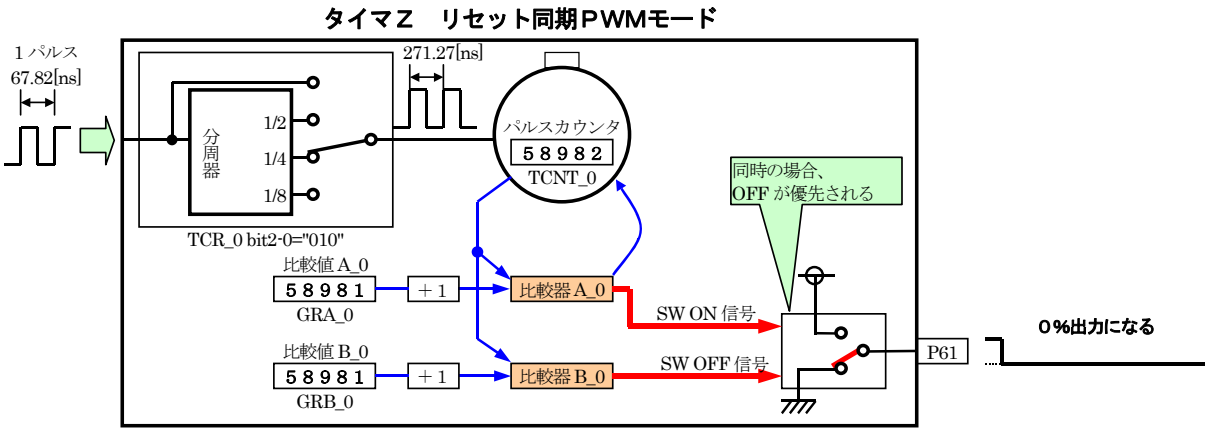
TCNT_0＝GRB_0 + 1
＝ 0 + 1
＝ 1

必ず 1 カウント分である 271.27[ns]は、ON になってしまいます。

リセット同期 PWM モードを使用するとき、下記のような決まりがあります。

TCNT_0＝GRB_0+1（端子を"0"にするタイミング） TCNT_0＝GRA_0+1（端子を"1"にするタイミング） が同時の場合、**端子を"0"にすることが優先されます**。

そのため、0%出力にしたい端子のジェネラルレジスタには、ジェネラルレジスタ A_0（GRA_0）と同じ値を設定します。



14.7.8 100%出力にしたい場合

P61 端子を例に説明します。

100%出力にしたい場合、ON 幅を 100%にするだけです。といって、周期と同じ値を設定すると、前記の通り、0%になってしまいます。

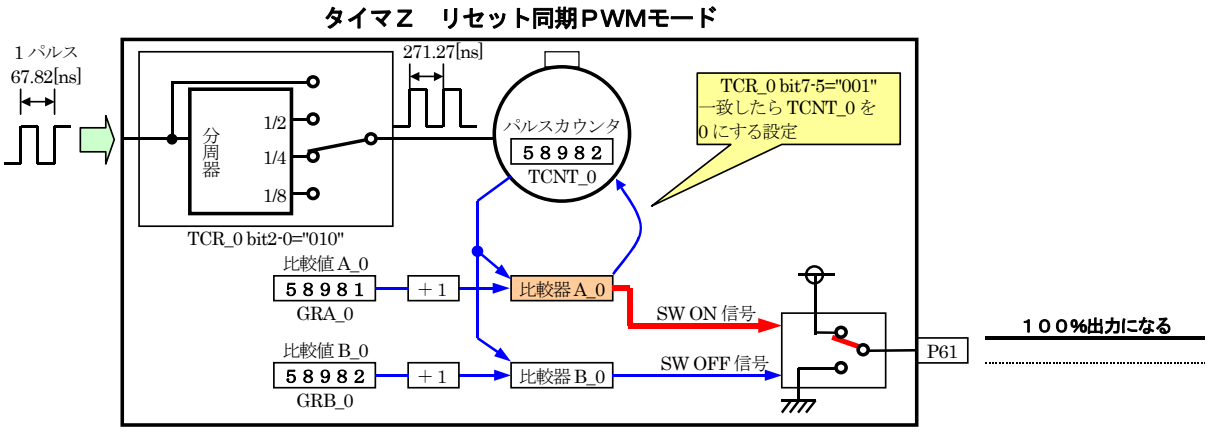
100%出力にしたいとき、ジェネラルレジスタ B_0（GRB_0）は下記の値を設定します。

GRB_0 ＝ GRA_0＋1
＝ 58981 ＋1
＝ 58982

TCNT_0 が増えていき、58982 の値になると、下記の条件が成り立ちます。

TCNT_0＝GRA_0 + 1

ON 信号が出力され、端子は"1"になります。そして、**TCR_0 の bit7～5 の設定により TCNT_0 は 0 になります**。端子を"0"にする条件である「TCNT_0＝GRB_0 ＋ 1」になることはありません。そのため、100%出力になるのです。



14.7.9 まとめ

●カウンタが+1 する時間の設定

タイマコントロールレジスタ_0（TCR_0）に、タイマカウンタ_0（TCNT_0）が 1 つ増える時間を設定します。
0x20…周期が 4.44[ms]以下の場合（＋1する時間は、67.72[ns]ごと）
0x21…周期が 8.89[ms]以下の場合（＋1する時間は、135.63[ns]]ごと）
0x22…周期が 17.78[ms]以下の場合（＋1する時間は、271.27[ns]ごと）
0x23…周期が 35.56[ms]以下の場合（＋1する時間は、542.53[ns]ごと）

今回は、周期を 16[ms]にするので、0x22 を設定します。

●周期の設定

PWM 波形の周期を設定します。今回は、周期を 16[ms]に設定します。
GRA_0＝設定したい周期／TCR_0 で設定した 1 パルス幅－1
GRA_0＝(16×10^{-3})／(271.27×10^{-9})－1
＝(16×10^{-3})／(271.27×10^{-9}) － 1
≒58981　　※小数点の指定はできないので、四捨五入します。

●ON 幅の設定

ON 幅を設定します。
・P61 の ON 幅は、ジェネラルレジスタ B_0（GRB_0）に設定します。P63 には反転した波形が出力されます。
・P64 の ON 幅は、ジェネラルレジスタ A_1（GRA_1）に設定します。P66 には反転した波形が出力されます。
・P65 の ON 幅は、ジェネラルレジスタ B_1（GRB_1）に設定します。P67 には反転した波形が出力されます。

例えば、P61 端子の ON 幅を 2[ms]にするとき
GRB_0＝ON 幅／TCR_0 で設定した 1 パルス幅－1
＝(2×10^{-3})／(271.27×10^{-9})－1
＝7372.8 - 1
＝7371.8
≒**7372**

※ON 幅を 0%出力するとき
GRA_0 と同じ値を設定します。
例） P61 端子を 0%出力にするとき
GRB_0＝GRA_0＝**58981**

※100%出力するとき
GRA_0 より1つ大きい値を設定します。
例） P61 端子を 100%出力にするとき
GRB_0＝GRA_0 + 1＝58981+1＝**58982**

14.8 プログラムの解説

14.8.1 リセット同期PWMモードの設定

init 関数で、I/O ポートの入出力設定終了後、タイマ Z のチャネル 0、チャネル 1 を使ってリセット同期 PWM モードで使用する設定にします。端子は、P61～P67 端子が PWM 出力になります。

```
 59 :      /* タイマ Z ch0,ch1 リセット同期 PWM モード */
 60 :      TCR_0  = 0x22;                    /* カウンタクロック設定      */
 61 :      TFCR   = 0x01;                    /* リセット同期 PWM モード    */
 62 :      TOCR   = 0x00;                    /* PWM 信号の出力設定        */
 63 :      TCNT_0 = 0;                       /* カウンタクリア            */
 64 :      GRA_0  = 58981;                   /* 周期の設定                */
 65 :      GRB_0  = 0;                       /* P61,P63                   */
 66 :      GRA_1  = 0;                       /* P64,P66                   */
 67 :      GRB_1  = 0;                       /* P65,P67                   */
 68 :      TOER   = 0x01;                    /* 出力端子の設定            */
 69 :      TSTR   = 0x01;                    /* タイマスタート            */
```

14.8.2 main関数

```
 31 :  void main( void )
 32 :  {
 33 :      init();                           /* マイコン機能の初期化      */
 34 :
 35 :      while( 1 ) {
 36 :          GRB_0 = (long)58982 * dipsw_get() / 15;     /* P61,P63      */
 37 :          GRA_1 = (long)0;                            /* P64,P66      */
 38 :          GRB_1 = (long)0;                            /* P65,P67      */
 39 :      }
 40 :  }
```

36 行で、ディップスイッチの値によりジェネラルレジスタ B_0(GRB_0)へ代入する数値を決めて P61 端子の ON 幅を可変しています。ディップスイッチの値と ON 幅をまとめておきます。

ディップスイッチの値	GRB_0 の値	ON 幅
0	58982 * 0 / 15 = 0	0% ※
1	58982 * 1 / 15 = 3932	6.6%
2	58982 * 2 / 15 = 7864	13.3%
3	58982 * 3 / 15 = 11796	20.0%
4	58982 * 4 / 15 = 15729	26.7%
5	58982 * 5 / 15 = 19661	33.3%
6	58982 * 6 / 15 = 23593	40.0%
7	58982 * 7 / 15 = 27525	46.7%
8	58982 * 8 / 15 = 31457	53.3%
9	58982 * 9 / 15 = 35389	60.0%
10	58982 * 10 / 15 = 39321	66.7%
11	58982 * 11 / 15 = 43253	73.3%
12	58982 * 12 / 15 = 47186	80.0%

13	58982 * 13 / 15 = 51118	86.7%
14	58982 * 14 / 15 = 55050	93.3%
15	58982 * 15 / 15 = 58982	100%

※GRB_0 を 0 にしても、TCNT_0＝(GRB_0 + 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、TCNT_0 の1カウント分(271.26ns)だけ ON になります。

14.9 問題点

リセット同期PWMモードを使用した場合も「**13.10 問題点**」(181ページ)の現象が発生してしまいます。

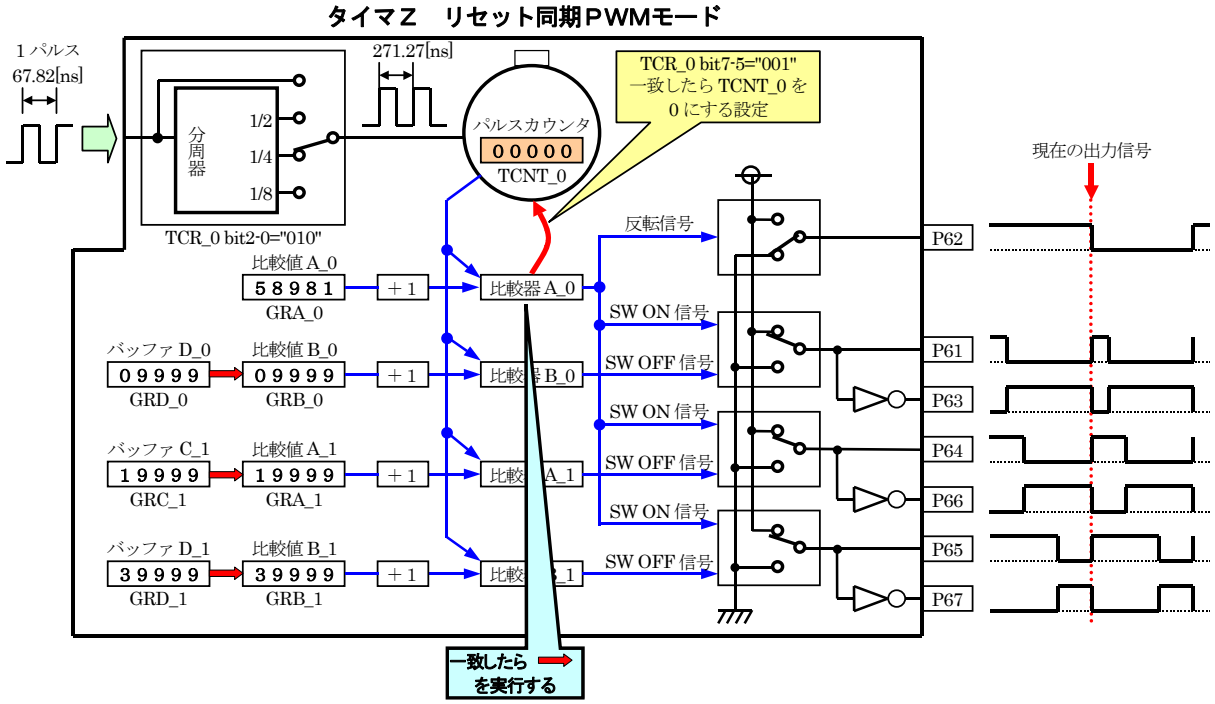
14.9.1 バッファ動作とは

バッファ動作とは、問題点が起きないようにするために、**「TCNT_0＝GRA_0＋1」になったときに GRB_0、GRA_1、GRB_1 の値を自動的に書き替える仕組みです。**

実際の動作は、ON 幅を変えるとき各ジェネラルレジスタを直接書き替えずに、変わりとなるレジスタに ON 幅を設定します。変わりとなるレジスタをバッファレジスタと呼びます。**「TCNT_0＝GRA_0＋1」になったときにバッファレジスタの値が、ジェネラルレジスタに自動的に転送されます。**

各ジェネラルレジスタに対応するバッファレジスタは下表のようになります。

ジェネラルレジスタ	対応するバッファレジスタ
ジェネラルレジスタ B_0(GRB_0)	ジェネラルレジスタ D_0(GRD_0)
ジェネラルレジスタ A_1(GRA_1)	ジェネラルレジスタ C_1(GRC_1)
ジェネラルレジスタ B_1(GRB_1)	ジェネラルレジスタ D_1(GRD_1)



14.9.2 タイマモードレジスタ(TMDR)の設定

バッファレジスタとして使用するかしないかは、タイマモードレジスタ(TMDR)で設定します。

タイマモードレジスタ(TMDR)							
7	6	5	4	3	2	1	0
0:GRD_1 は通常動作 1:GRD_1 は GRB_1 の バッファとして動作	0:GRC_1 は通常動作 1:GRC_1 は GRA_1 の バッファとして動作	0:GRD_0 は通常動作 1:GRD_0 は GRB_0 の バッファとして動作	0:GRC_0 は通常動作 1:GRC_0 は GRA_0 の バッファとして動作	−	−	−	0:TCNT_1 と TCNT_0 は 独立動作 1:同期動作
1	**1**	**1**	**0**				**0**

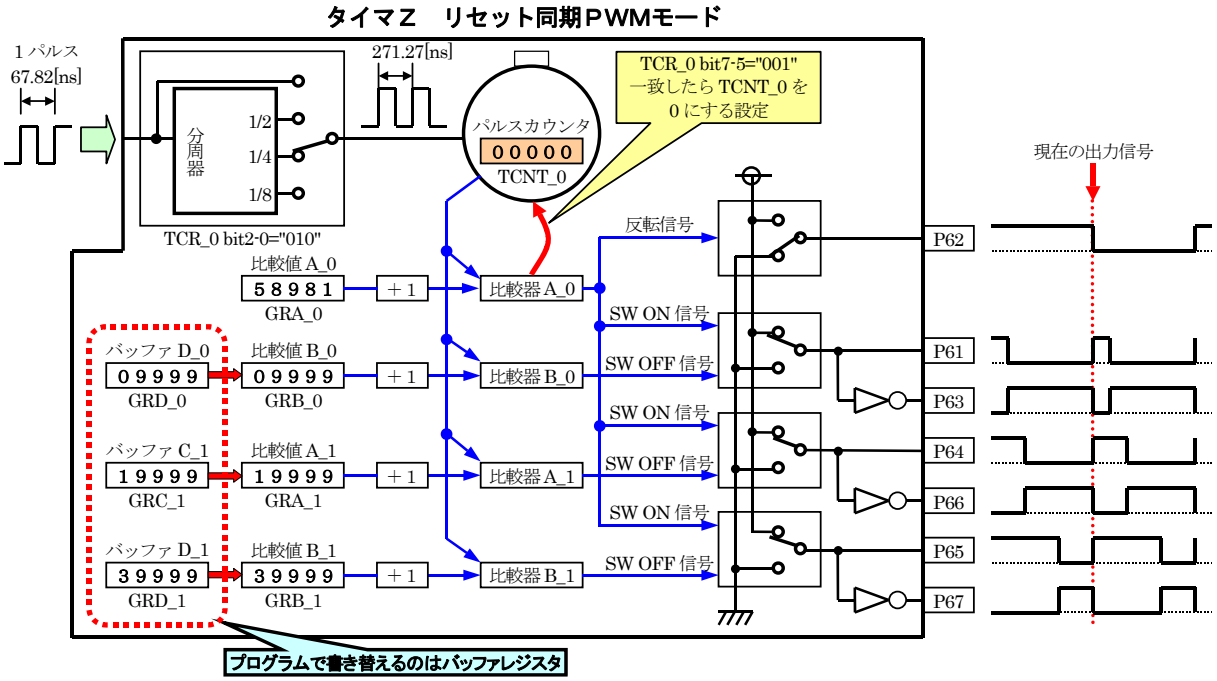
プログラムは、リセット同期 PWM モードの初期設定時にタイマモードレジスタ(TMDR)の設定を追加します。また、バッファレジスタの設定も追加します。

```
59 :    /* タイマZ ch0,ch1 リセット同期PWMモード */
60 :    TCR_0  = 0x22;          /* カウンタクロック設定      */
61 :    TFCR   = 0x01;          /* リセット同期PWMモード     */
62 :    TOCR   = 0x00;          /* PWM信号の出力設定         */
63 :    TCNT_0 = 0;             /* カウンタクリア            */
        TMDR   = 0xe0;          /* バッファ動作設定          */
64 :    GRA_0  = 58981;         /* 周期の設定                */
65 :    GRB_0  = GRD_0 = 0;     /* P61,P63                   */
66 :    GRA_1  = GRC_1 = 0;     /* P64,P66                   */
67 :    GRB_1  = GRD_1 = 0;     /* P65,P67                   */
68 :    TOER   = 0x01;          /* 出力端子の設定            */
69 :    TSTR   = 0x01;          /* タイマスタート            */
```

14.9.3 ON幅の設定

プログラムでは、ジェネラルレジスタの値を書き替えるのではなく、バッファレジスタの値を書き替えます。「TCNT_0＝GRA_0＋1」になると自動的にバッファレジスタの値が各ジェネラルレジスタに転送されます。

```
GRD_0 = 19999;          /* P61,P63   */
GRC_1 = 29999;          /* P64,P66   */
GRD_1 = 39999;          /* P65,P67   */
```



14.10 演習

14.10.1 バッファ動作

バッファ動作になるようプログラムを改造してみましょう。

14.10.2 P64 端子、P65 端子への出力

P61 端子の他、P64 端子、P65 端子へも、ディップスイッチの値により PWM 出力するようにしてみましょう。ただし、バッファ動作になっているものとします。プログラム例は下記のようです。

```
void main( void )
{
    init();                                 /* マイコン機能の初期化      */

    while( 1 ) {
        GRD_0 = (long)58982 * dipsw_get() / 15;     /* P61,P63       */
        GRC_1 = (long)58982 * dipsw_get() / 15;     /* P64,P66       */
        GRD_1 = (long)58982 * dipsw_get() / 15;     /* P65,P67       */
    }
}
```

14.10.3 0%出力

サンプルプログラムは、0%出力でも 1 カウント分 ON になってしまいます。正真正銘の 0%出力になるようにプログラムを改造してみましょう。ただし、バッファ動作になっているものとします。プログラム例は下記のようです。

```
void main( void )
{
    init();                                 /* マイコン機能の初期化      */

    while( 1 ) {
        if( dipsw_get() == 0 ) {
            GRD_0 = 58981;                          /* P61 = "0"     */
        } else {
            GRD_0 = (long)58982 * dipsw_get() / 15; /* P61           */
        }
    }
}
```

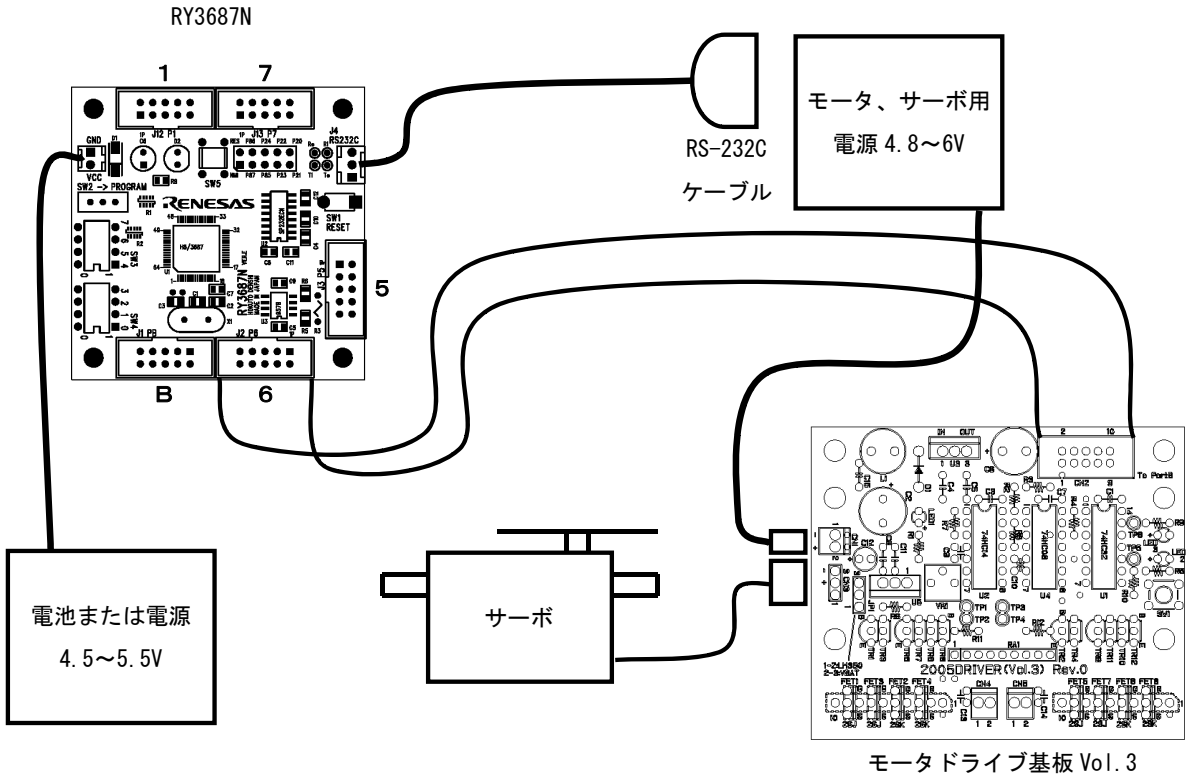
15. プロジェクト「servo」 サーボの制御

15.1 概要

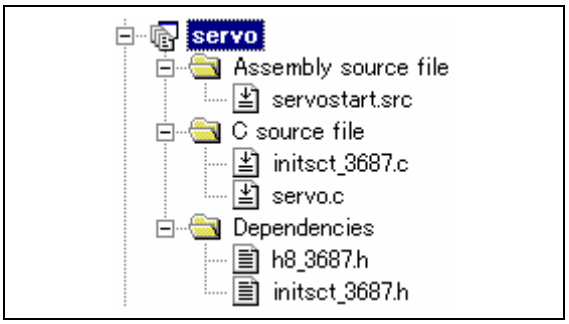
サーボを制御します。マイコンのポートは、下記を使用します。
・ポート 6・・・モータドライブ基板 Vol.3 を接続

15.2 接続

・CPU ボードのポート 6 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。



15.3 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_3687.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。H8/3687F の内蔵周辺機能の初期化も行います。
4	h8_3687.h	H8/3687F 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_3687.h	initsct_3687.c のヘッダファイルです。

15.4 プログラム「servo.c」

```
 1 :  /************************************************************************/
 2 :  /* サーボの制御 "servo.c"                                               */
 3 :  /*                          2007.04 ジャパンマイコンカーラリー実行委員会 */
 4 :  /************************************************************************/
 5 :  /*
 6 :  入力：P33-P30(CPUボード上のディップスイッチ)
 7 :  出力：ポート6にモータドライブ基板Vol.3＋サーボ
 8 :
 9 :  ポート６に接続しているモータドライブ基板Vol.3のサーボを制御します。
10 :  */
11 :
12 :  /*====================================*/
13 :  /* インクルード                       */
14 :  /*====================================*/
15 :  #include    <machine.h>
16 :  #include    "h8_3687.h"
17 :
18 :  /*====================================*/
19 :  /* プロトタイプ宣言                   */
20 :  /*====================================*/
21 :  void init( void );
22 :  unsigned char dipsw_get( void );
23 :
24 :  /*====================================*/
25 :  /* グローバル変数の宣言               */
26 :  /*====================================*/
27 :
28 :  /************************************************************************/
29 :  /* メインプログラム                                                     */
30 :  /************************************************************************/
31 :  void main( void )
32 :  {
33 :      init();                             /* マイコン機能の初期化      */
34 :
35 :      while( 1 ) {
36 :          GRD_1 = 2700 + dipsw_get() * 16;
37 :      }
38 :  }
```

```
39 :
40 : /*********************************************************************/
41 : /* H8/3687F 内蔵周辺機能　初期化                                     */
42 : /*********************************************************************/
43 : void init( void )
44 : {
45 :     /* I/Oポートの入出力設定 */
46 :     PCR1 = 0xff;
47 :     PCR2 = 0xfd;                        /* 通信ビットP22:TxD P21:RxD*/
48 :     PCR3 = 0xf0;                        /* 基板上のディップスイッチ */
49 :     PCR5 = 0xff;
50 :     PDR6 = 0xc0;
51 :     PCR6 = 0xfe;                        /* モータドライブ基板       */
52 :     PCR7 = 0xff;
53 :     PCR8 = 0xff;
54 :     /* ポートBは、入力専用なので入出力設定はありません。          */
55 :     /* また、A/D変換器のADCSRによって指定された1ビットだけは、   */
56 :     /* 入力ポートとしては使えません。                             */
57 :
58 :     /* タイマZ ch0,ch1 リセット同期PWMモード */
59 :     TCR_0  = 0x23;                      /* カウンタクロック設定     */
60 :     TFCR   = 0x01;                      /* リセット同期PWMモード    */
61 :     TOCR   = 0x00;                      /* PWM信号の出力設定        */
62 :     TCNT_0 = 0;                         /* カウンタクリア           */
63 :     TMDR   = 0xe0;                      /* バッファ動作設定         */
64 :     GRA_0  = 29490;                     /* 周期の設定               */
65 :     GRB_0  = GRD_0 = 0;                 /* 左モータのPWM設定        */
66 :     GRA_1  = GRC_1 = 0;                 /* 右モータのPWM設定        */
67 :     GRB_1  = GRD_1 = 2700;              /* サーボのPWM設定          */
68 :     TOER   = 0xcd;                      /* 出力端子の設定           */
69 :     TSTR   = 0x01;                      /* タイマスタート           */
70 : }
71 :
72 : /*********************************************************************/
73 : /* ディップスイッチ値読み込み                                        */
74 : /* 引数　なし                                                        */
75 : /* 戻り値 スイッチ値 0～15                                           */
76 : /*********************************************************************/
77 : unsigned char dipsw_get( void )
78 : {
79 :     unsigned char sw;
80 :
81 :     sw  = PDR3;                         /* ディップスイッチ読み込み */
82 :     sw &= 0x0f;
83 :     return  sw;
84 : }
85 :
86 : /*********************************************************************/
87 : /* End of file                                                       */
88 : /*********************************************************************/
```

15.5 モータドライブ基板Vol.3 について

15.5.1 概要

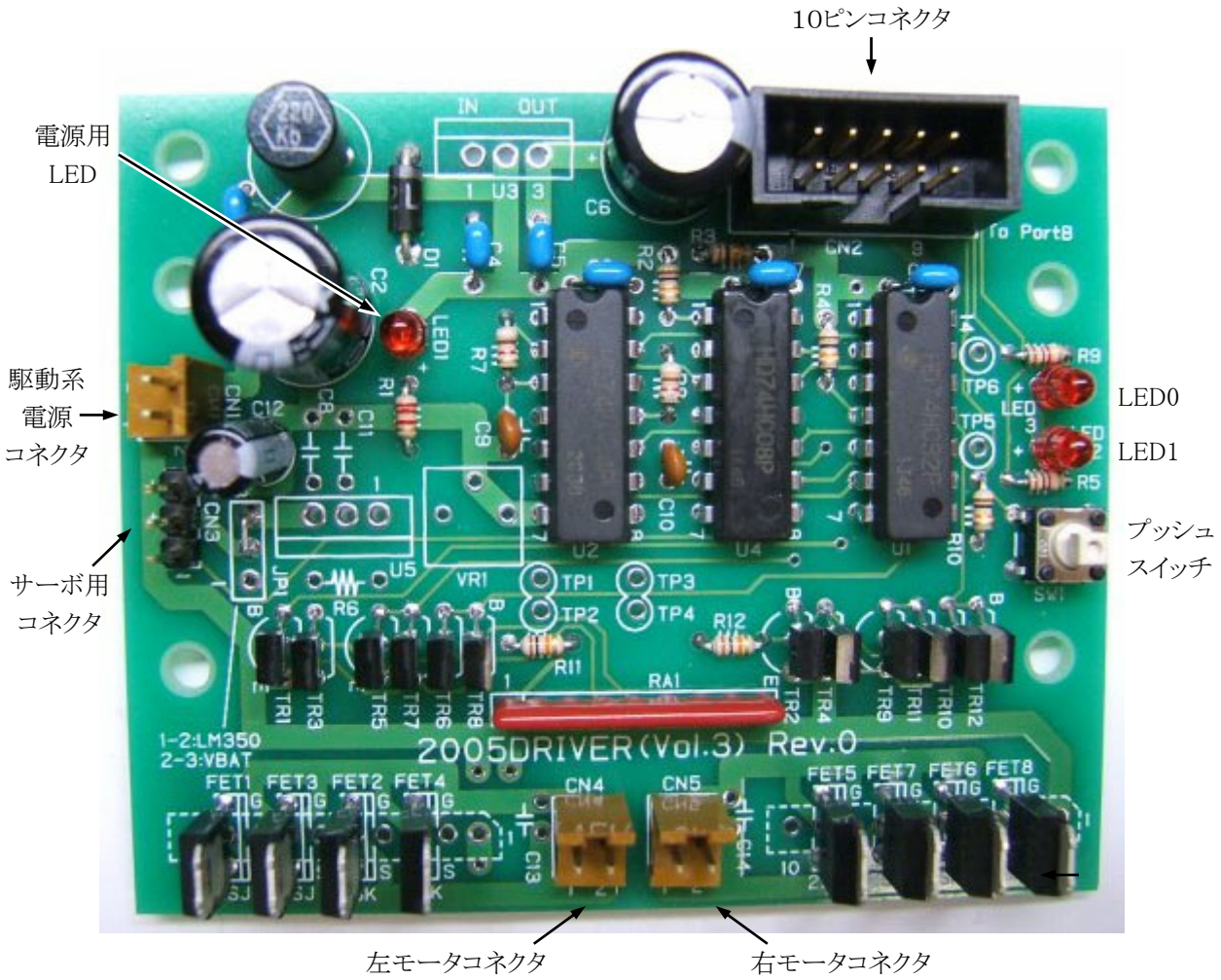
モータドライブ基板 Vol.3 には、2 個の LED、1 個のプッシュスイッチ、1個のサーボ、2 個のモータの制御を行うことができます(LED は 3 個ありますが、1 個は電源モニタ用です)。モータは、それぞれ「正転」、「ブレーキ」、「逆転」の制御を行うことができ、速度制御は H8/3687F の PWM 機能(リセット同期 PWM モード)を使用することにより細かなスピード調整を行うことができます。また、駆動系対応電圧は約 5～15Vですので、モータに加える電圧を上げてチューンナップすることも可能です(ただし、LM350 追加セットを追加する必要があります)。

下記に、モータドライブ基板 Vol.1～3 の仕様をまとめます。

名称	モータドライブ 基板(Vol.2)	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
使用されて いるキット	キット Vol.1	キット Vol.2	キット Vol.3 またはキット Ver.4
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	実績無し	実績無し	RY3687N ボードは **ポート 6**
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 または　4AM12×2 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、 サーボは…	できない	6V 一定にする回路を外付け する必要有り	あらかじめ基板にパターン有 り、LM350 追加セットの部品 を取り付け
標準 プログラム			**kit07_3687.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)

15.5.2 部品実装図

ドライブ基板 3 は、部品を実装すると下図のようになります。

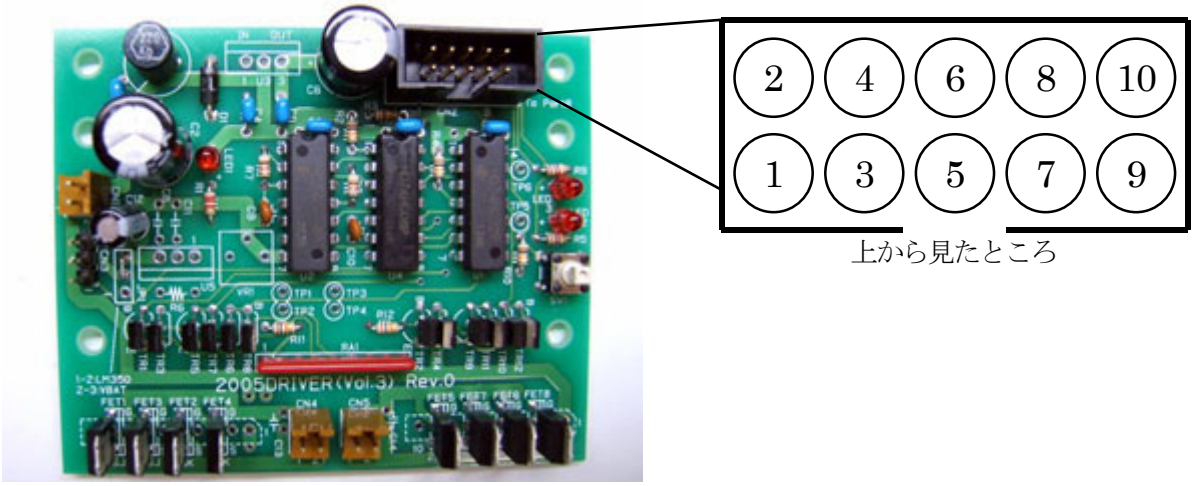


10 ピンコネクタ	フラットケーブルで CPU ボードと接続します。ポート 6(J2)のコネクタに接続します。
駆動系 電源コネクタ	モータとサーボに供給する電源です。74HC08、74HC14、74HC32 などの制御系部品は、10 ピンコネクタから供給される5Vで動作します。標準キットでは入力電圧 6V までに対応していますが、それ以上の電圧にするときは、**サーボに加える電圧を 6V 一定にする必要があります**。LM350 追加セットの部品を追加すると、サーボ電圧を一定にすることができます。
右モータコネクタ	右モータを接続します。
左モータコネクタ	左モータを接続します。
サーボ用コネクタ	サーボを接続します。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 6 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 6 の bit7 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。
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 6 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。

15.5.3 10 ピンコネクタの信号

モータドライブ基板の10ピンコネクタの信号は下記のとおりです。

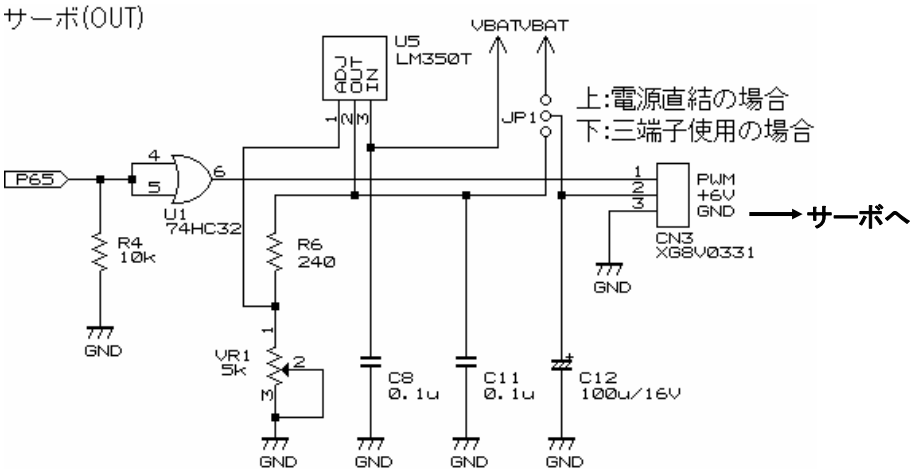


ピン番	信号、方向	詳細	“0”	“1”	詳細
1	―	+5V			
2	基板←P67	LED1	点灯	消灯	
3	基板←P66	LED0	点灯	消灯	
4	**基板←P65**	**サーボ信号**	**PWM 信号**		**GRD_1 でデューティ比設定**
5	基板←P64	右モータ PWM	停止	動作	GRC_1 でデューティ比設定
6	基板←P63	右モータ回転方向	正転	逆転	
7	基板←P62	左モータ回転方向	正転	逆転	
8	基板←P61	左モータ PWM	停止	動作	GRD_0 でデューティ比設定
9	基板→P60	プッシュスイッチ	押された	押されてない	
10	―	GND			

表のとおり、P65 がサーボ制御です。リセット同期 PWM モードを使用していますので P65 のデューティ比を変えるには GRD_1 の値を変えます。

15.5.4 サーボの制御回路

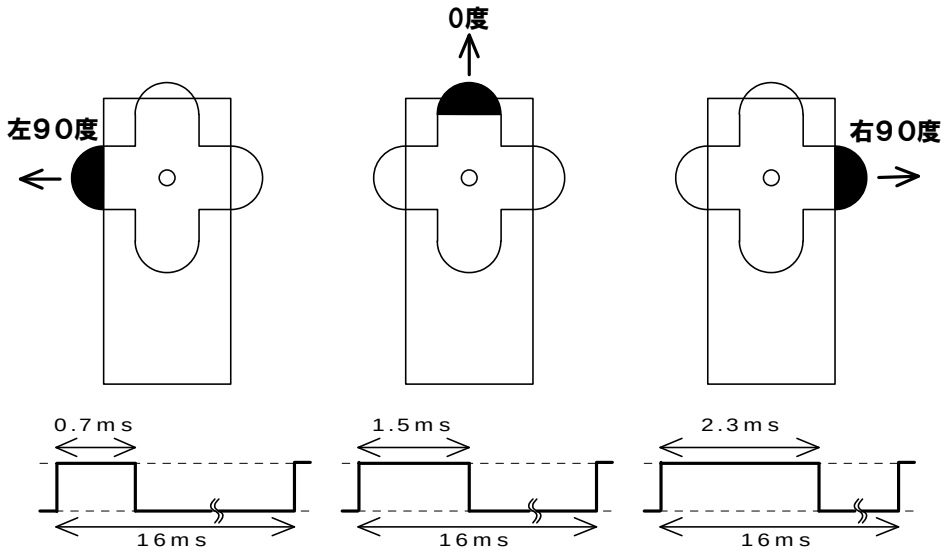
モータドライブ基板 Vol.3 のサーボ制御回路は下記のようになっています。



1. ポート 6 の bit5 から PWM 信号を出力します。プログラムは、ジェネラルレジスタ D_1（GRD_1）の値を変えると ON 幅が変わります。
2. P65 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、P65 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。

15.6 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]

15.7 プログラムの解説

15.7.1 ポート 6 の入出力設定

ポート 6 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。

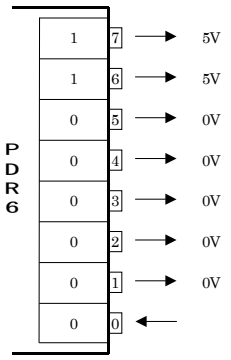
ピン番	信号、方向	詳細	“0”	“1”	入出力	PCR6 の値
1	－	+5V				
2	基板←P67	LED1	点灯	消灯	出力	1
3	基板←P66	LED0	点灯	消灯	出力	1
4	基板←P65	サーボ信号	PWM 信号		出力	1
5	基板←P64	右モータ PWM	停止	動作	出力	1
6	基板←P63	右モータ回転方向	正転	逆転	出力	1
7	基板←P62	左モータ回転方向	正転	逆転	出力	1
8	基板←P61	左モータ PWM	停止	動作	出力	1
9	基板→P60	プッシュスイッチ	押された	押されてない	入力	0
10	－	GND				

入力は"0"、出力は"0"を設定するので、ポートコントロールレジスタ 6(PCR6)の値は下記のようになります。
PCR6 ＝ 1111 1110 ＝ 0xfe

15.7.2 ポート 6 に出力する値の初期値

ポート 6 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”	PDR6 の値
1	―	+5V			
2	基板←P67	LED1	点灯	消灯	**1**
3	基板←P66	LED0	点灯	消灯	**1**
4	基板←P65	サーボ信号	PWM 信号		**0**
5	基板←P64	右モータ PWM	停止	動作	**0**
6	基板←P63	右モータ回転方向	正転	逆転	**0**
7	基板←P62	左モータ回転方向	正転	逆転	**0**
8	基板←P61	左モータ PWM	停止	動作	**0**
9	基板→P60	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 PWM 信号を出食しなければ行けないのですが、とりあえず″0″としておきます。P60 は入力用なので、どちら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。

PDR6 ＝ 1100 0000 ＝ 0xc0

となります。

15.7.3 PCR6 とPDR6 の設定する順番

プログラムでは最初にポートデータレジスタ 6(PDR6)へ 0xc0 をセットして、次にポートコントロールレジスタ 6(PCR6)へ 0xfeをセットしています。

```
50 :      PDR6 = 0xc0;
51 :      PCR6 = 0xfe;                    /* モータドライブ基板      */
```

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。PCR6 の初期値は「0x00」で、端子は入力になっています。PDR6 の初期値は「0x00」です。そのため、先に PCR6 でポートを出力にセットすると、PDR6 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 PCR6 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 PDR6 を設定して LED を消灯状態にしておいてから、PCR6 の設定を行っています。

15.7.4 リセット同期PWMモードの設定

タイマ Z をリセット同期 PWM モードに設定します。

```
59 :    TCR_0  = 0x23;                  /* カウンタクロック設定      */
60 :    TFCR   = 0x01;                  /* リセット同期 PWM モード    */
61 :    TOCR   = 0x00;                  /* PWM 信号の出力設定        */
62 :    TCNT_0 = 0;                     /* カウンタクリア            */
63 :    TMDR   = 0xe0;                  /* バッファ動作設定          */
64 :    GRA_0  = 29490;                 /* 周期の設定                */
65 :    GRB_0  = GRD_0 = 0;             /* 左モータの PWM 設定       */
66 :    GRA_1  = GRC_1 = 0;             /* 右モータの PWM 設定       */
67 :    GRB_1  = GRD_1 = 2700;          /* サーボの PWM 設定         */
68 :    TOER   = 0xcd;                  /* 出力端子の設定            */
69 :    TSTR   = 0x01;                  /* タイマスタート            */
```

各レジスタの設定については、プロジェクト「pwm3」を参照してください。

モータドライブ基板の接続は下記のようになります。太字の P65、P64、P61 が PWM 信号出力、他は PWM 信号出力にはしません。

ポート 6 に接続するモータドライブ基板の信号							
7	6	5	4	3	2	1	0
LED1	LED0	**サーボ信号**	**右モータ PWM 信号**	右モータ回転方向	左モータ回転方向	**左モータ PWM 信号**	プッシュスイッチ

タイマアウトプットマスタイネーブルレジスタ(TOER)でタイマ出力の許可／禁止を設定します。

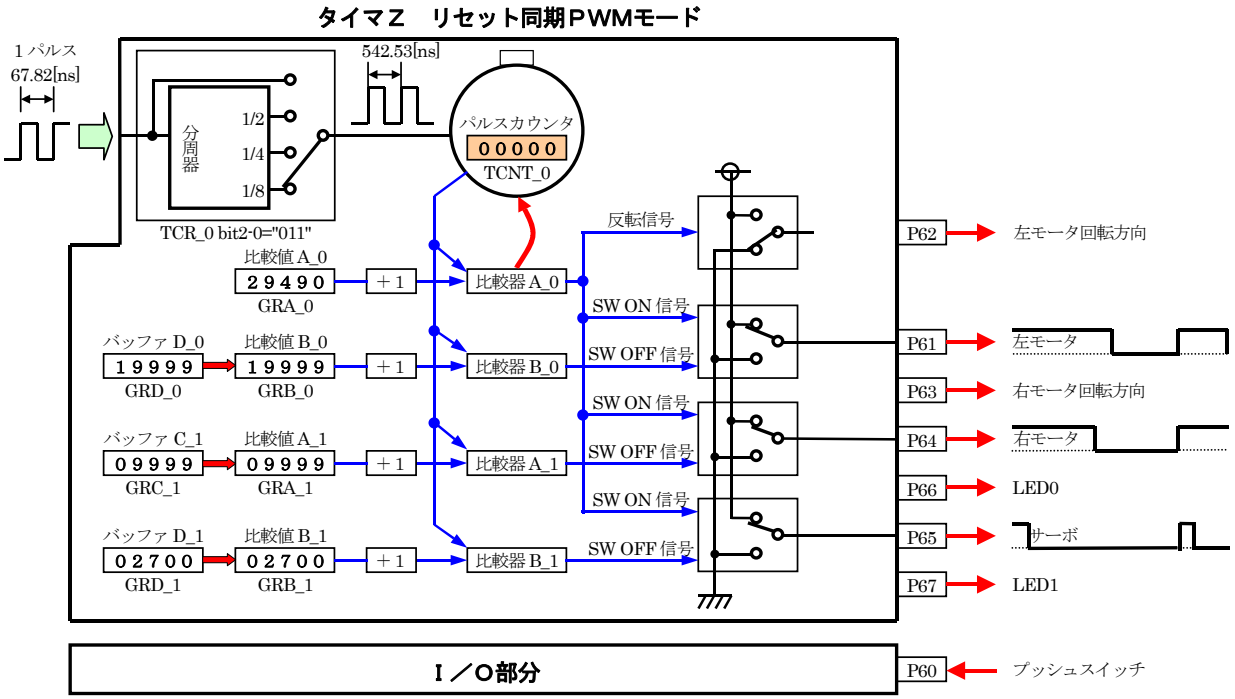
タイマアウトプットマスタイネーブルレジスタ(TOER)							
7	6	5	4	3	2	1	0
FTIOD1(P67)端子の出力を許可するなら"0"(初期値は"1")	FTIOC1(P66)端子の出力を許可するなら"0"(初期値は"1")	FTIOB1(**P65**)端子の出力を許可するなら"0"(初期値は"1")	FTIOA1(**P64**)端子の出力を許可するなら"0"(初期値は"1")	FTIOD0(P63)端子の出力を許可するなら"0"(初期値は"1")	FTIOC0(P62)端子の出力を許可するなら"0"(初期値は"1")	FTIOB0(**P61**)端子の出力を許可するなら"0"(初期値は"1")	FTIOA0(P60)端子の出力を許可するなら"0"(初期値は"1")
1	1	**0**	**0**	1	1	**0**	1

TOER = 1100 1101 = 0xcd
を設定します。

```
68 :    TOER   = 0xcd;                  /* 出力端子の設定            */
```

15.7.5 ON幅の設定

ポート 6 の各ビットとモータドライブ基板の接続は下図のようになっています。



よって、

・左モータの ON 幅を決めるレジスタは、GRD_0
・右モータの ON 幅を決めるレジスタは、GRC_1
・サーボの ON 幅を決めるレジスタは、GRD_1

となります。それぞれのレジスタの値を設定して ON 幅を設定します。ジェネラルレジスタ B_0、A_1、B_1 の設定は最初の1回だけ行います。init 関数内でのそれぞれのレジスタの設定は、

・左モータは停止させるので、
　GRD_0=0
・右モータは停止させるので、
　GRC_1=0
・サーボは、0 度になるよう設定、ON 幅は 1.5[ms]なので、
　$(1.5\times10^{-3})\div(542.53\times10^{-9})=2765\fallingdotseq2700$

を設定します（下記）。

```
65 :      GRB_0  = GRD_0 = 0;              /* 左モータの PWM 設定      */
66 :      GRA_1  = GRC_1 = 0;              /* 右モータの PWM 設定      */
67 :      GRB_1  = GRD_1 = 2700;           /* サーボの PWM 設定        */
```

15.7.6 mian関数

```
 31 :  void main( void )
 32 :  {
 33 :      init();                                /* マイコン機能の初期化      */
 34 :
 35 :      while( 1 ) {
 36 :          GRD_1 = 2700 + dipsw_get() * 16;
 37 :      }
 38 :  }
```

サーボは P65 へ接続されています。P65 のデューティ比を変えるレジスタは、GRB_1 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 GRD_1 となります。

計算式は、

GRD_1 = 2700 + dipsw_get() * 16;

①　②　③

となっています。

①サーボのセンタ値

先ほどの計算の通り、2700 となります。

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度動くようにしています。

サーボが、右に 90 度向くときは 2.3ms のパルスなので、

$(2.3\times10^{-3})\div(542.53\times10^{-9})=4239$

サーボが、左に 90 度向くときは 0.7ms のパルスなので、

$(0.7\times10^{-3})\div(542.53\times10^{-9})=1290$

よって、±90 度の移動量は、

（右 90 度）－（左 90 度）＝4239-1290＝2949

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。

2949÷180＝16.38≒**16**

となります。

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、2700 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。

15.8 演習

15.8.1 サーボを逆側に動かす

サーボを逆側に動かしてみましょう。プログラム例は下記のようです。

```
while( 1 ) {
    GRD_1 = 2700 - dipsw_get() * 16;
}
```

15.8.2 サーボを逆側に動かす

・ディップスイッチの 8 を 0 度、9 以上で左へ、7 以下で右へ動くようにしてみましょう。プログラム例は下記のようです。

```
while( 1 ) {
    GRD_1 = 2700 + (dipsw_get() - 8) * 16;
}
```

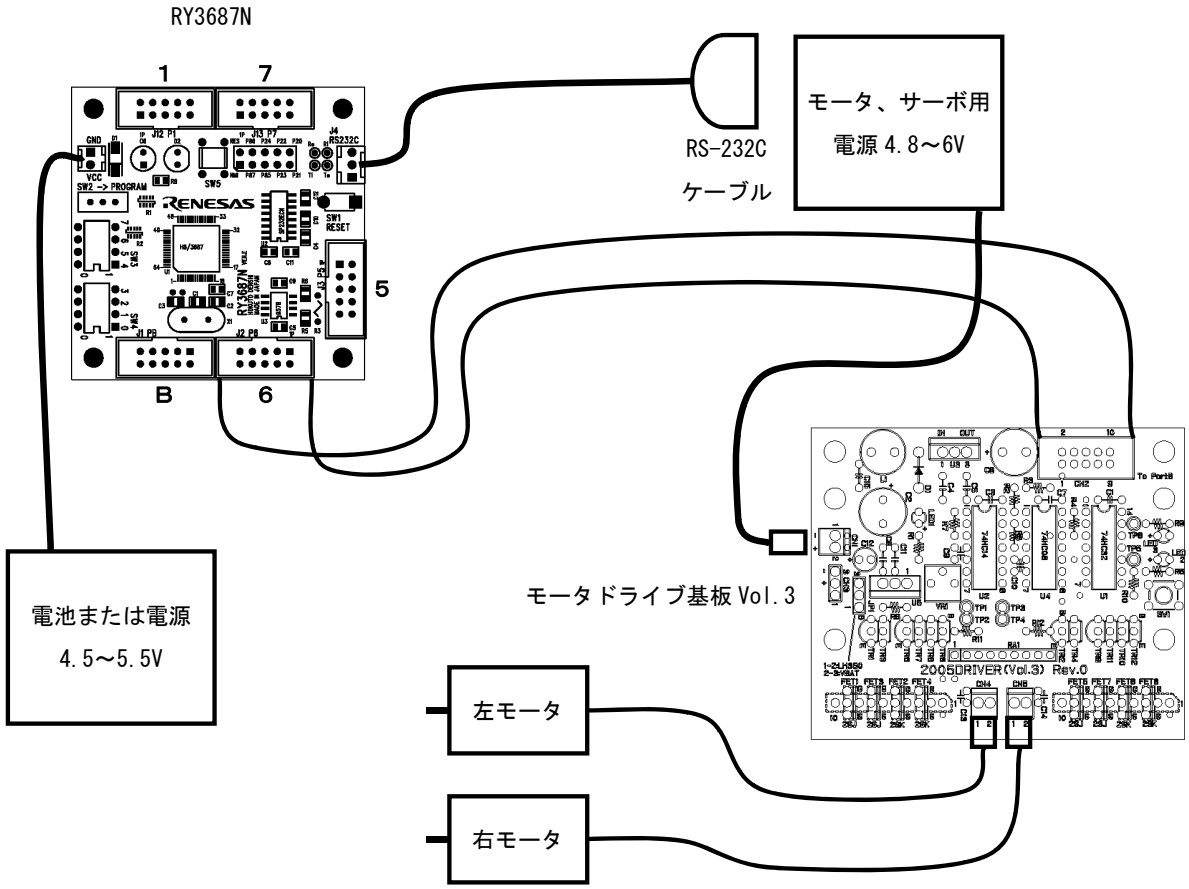
16. プロジェクト「motor」 モータの制御

16.1 概要

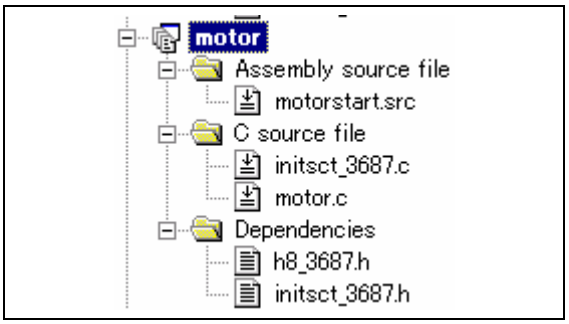
モータを制御します。マイコンのポートは、下記を使用します。
・ポート 6・・・モータドライブ基板 Vol.3 を接続

16.2 接続

・CPU ボードのポート 6 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、左モータ、右モータとモータ・サーボ用電源を接続します。



16.3 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_3687.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。H8/3687F の内蔵周辺機能の初期化も行います。
4	h8_3687.h	H8/3687F 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_3687.h	initsct_3687.c のヘッダファイルです。

16.4 プログラム「motor.c」

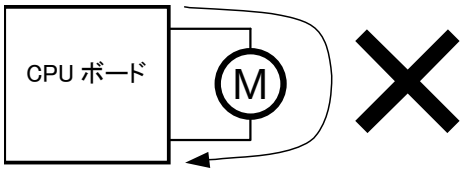
```
 1 :  /*********************************************************************/
 2 :  /* モータの制御 "motor.c"                                            */
 3 :  /*                          2007.04 ジャパンマイコンカーラリー実行委員会 */
 4 :  /*********************************************************************/
 5 :  /*
 6 :  入力：P33-P30(CPUボード上のディップスイッチ)
 7 :  出力：ポート6にモータドライブ基板Vol.3＋左モータ＋右モータ
 8 :
 9 :  ポート６に接続しているモータドライブ基板Vol.3のモータを制御します。
10 :  */
11 :
12 :  /*====================================*/
13 :  /* インクルード                       */
14 :  /*====================================*/
15 :  #include    <machine.h>
16 :  #include    "h8_3687.h"
17 :
18 :  /*====================================*/
19 :  /* プロトタイプ宣言                   */
20 :  /*====================================*/
21 :  void init( void );
22 :  unsigned char dipsw_get( void );
23 :
24 :  /*====================================*/
25 :  /* グローバル変数の宣言               */
26 :  /*====================================*/
27 :
28 :  /*********************************************************************/
29 :  /* メインプログラム                                                  */
30 :  /*********************************************************************/
31 :  void main( void )
32 :  {
33 :      init();                             /* マイコン機能の初期化      */
34 :
35 :      while( 1 ) {
36 :          GRD_0 = (long)29491 * dipsw_get() / 15; /* 左モータ          */
37 :      }
38 :  }
```

```
39 :
40 : /*********************************************************************/
41 : /* H8/3687F 内蔵周辺機能　初期化                                     */
42 : /*********************************************************************/
43 : void init( void )
44 : {
45 :     /* I/Oポートの入出力設定 */
46 :     PCR1 = 0xff;
47 :     PCR2 = 0xfd;                        /* 通信ビットP22:TxD P21:RxD*/
48 :     PCR3 = 0xf0;                        /* 基板上のディップスイッチ */
49 :     PCR5 = 0xff;
50 :     PDR6 = 0xc0;
51 :     PCR6 = 0xfe;                        /* モータドライブ基板       */
52 :     PCR7 = 0xff;
53 :     PCR8 = 0xff;
54 :     /* ポートBは、入力専用なので入出力設定はありません。          */
55 :     /* また、A/D変換器のADCSRによって指定された1ビットだけは、   */
56 :     /* 入力ポートとしては使えません。                             */
57 :
58 :     /* タイマZ ch0,ch1 リセット同期PWMモード */
59 :     TCR_0  = 0x23;                      /* カウンタクロック設定     */
60 :     TFCR   = 0x01;                      /* リセット同期PWMモード    */
61 :     TOCR   = 0x00;                      /* PWM信号の出力設定        */
62 :     TCNT_0 = 0;                         /* カウンタクリア           */
63 :     TMDR   = 0xe0;                      /* バッファ動作設定         */
64 :     GRA_0  = 29490;                     /* 周期の設定               */
65 :     GRB_0  = GRD_0 = 0;                 /* 左モータのPWM設定        */
66 :     GRA_1  = GRC_1 = 0;                 /* 右モータのPWM設定        */
67 :     GRB_1  = GRD_1 = 2700;              /* サーボのPWM設定          */
68 :     TOER   = 0xcd;                      /* 出力端子の設定           */
69 :     TSTR   = 0x01;                      /* タイマスタート           */
70 : }
71 :
72 : /*********************************************************************/
73 : /* ディップスイッチ値読み込み                                        */
74 : /* 引数　なし                                                        */
75 : /* 戻り値 スイッチ値 0～15                                           */
76 : /*********************************************************************/
77 : unsigned char dipsw_get( void )
78 : {
79 :     unsigned char sw;
80 :
81 :     sw  = PDR3;                         /* ディップスイッチ読み込み */
82 :     sw &= 0x0f;
83 :     return  sw;
84 : }
85 :
86 : /*********************************************************************/
87 : /* End of file                                                       */
88 : /*********************************************************************/
```

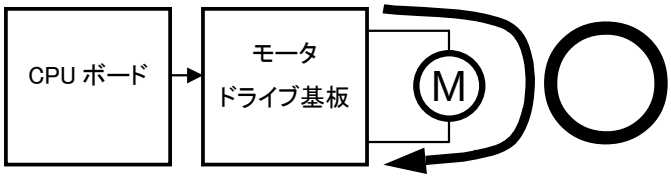
16.5 モータの制御について

16.5.1 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



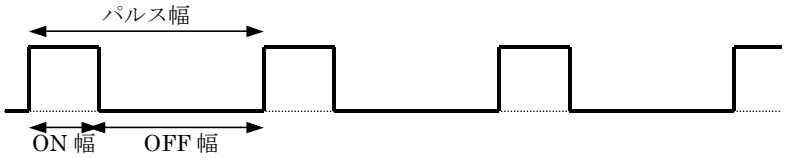
大電流に変換、モータ駆動！

16.5.2 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。

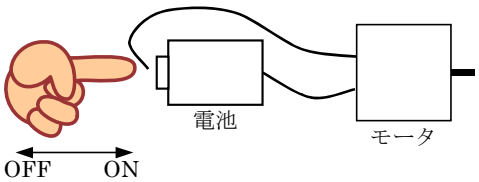
ボリュームを使えば電圧を落とすことができます。しかし、モータへは大電流が流れるため、非常に大きな抵抗が必要です。また、モータに加えなかった分は、抵抗の熱となってしまいます。

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。

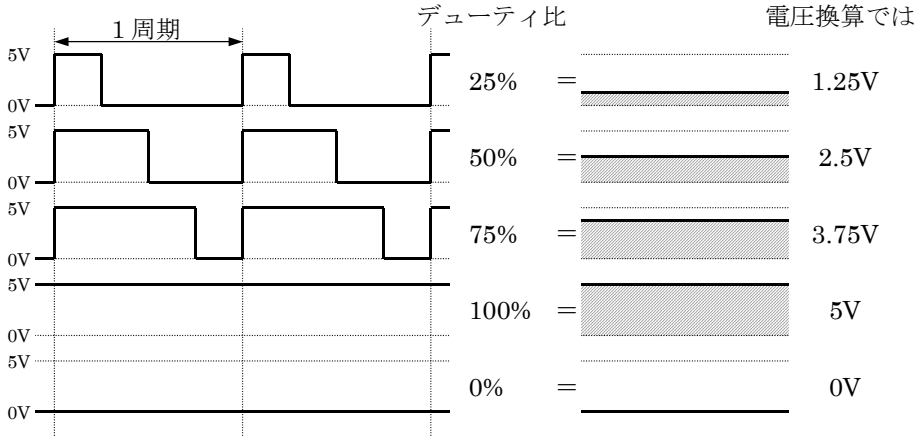
「PWM」と聞くと、何か難しく感じてしまいますが、下記のように手でモータと電池の線を「繋ぐ」、「離す」の繰り返し、それも PWM と言えます。繋いでいる時間が長いとモータは速く回ります。離している時間が長いとモータは少ししか回りません。人なら「繋ぐ」、「離す」動作をコンマ数秒でしか行えませんがマイコンなら数ミリ秒で行えます。



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。

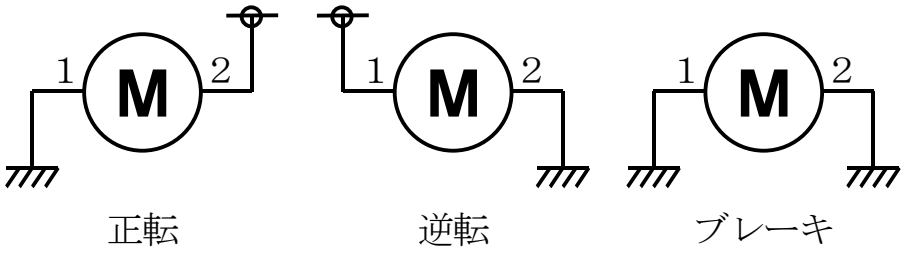


なぜ電圧制御ではなくパルス幅制御でモータのスピードを制御するのでしょうか。CPU は”0”か”1”かのデジタル値の取り扱いは大変得意ですが、何 V というアナログ的な値は不得意です。そのため、″0″と″1″の幅を変えて、あたかも電圧制御しているように振る舞います。これが PWM 制御です。

16.5.3 正転、逆転、ブレーキの原理

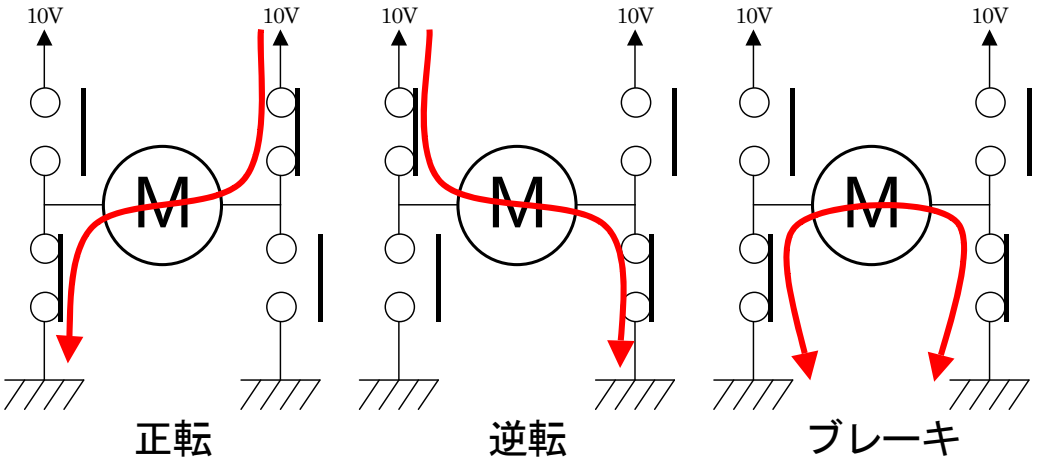
モータドライブ基板(Vol.3)では、モータを「正転、逆転、ブレーキ」制御することができます。これは、モータの端子に加える電圧を下表のように変えることにより、制御しています。

動作	モータ端子 1	モータ端子 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



16.5.4 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。

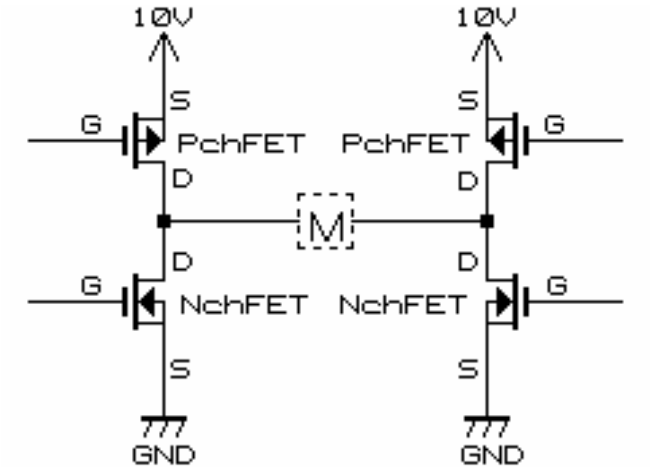


16.5.5 Hブリッジ回路のスイッチをFETにする

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。

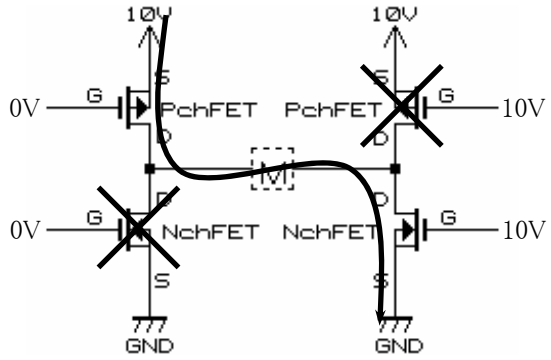
P チャネル FET は、V_G（ゲート電圧）＜V_S（ソース電圧）のとき、D-S（ドレイン－ソース）間に電流が流れます。

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。

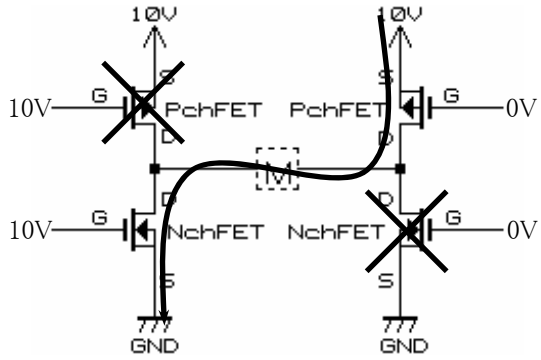


これら 4 つの FET のゲートに加える電圧を変えることにより、正転、逆転、ブレーキの動作を行います。

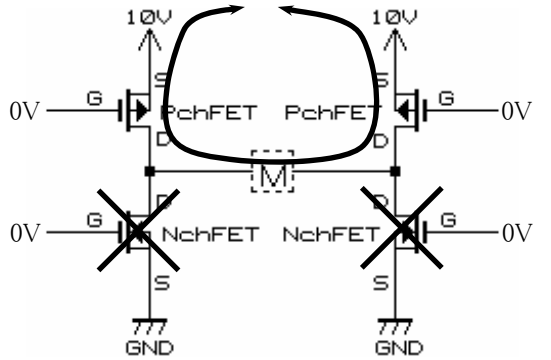
逆転



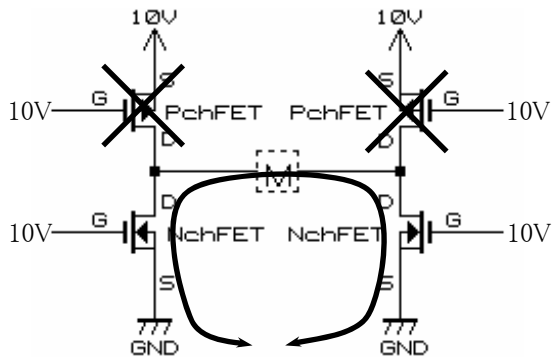
正転



ブレーキ1

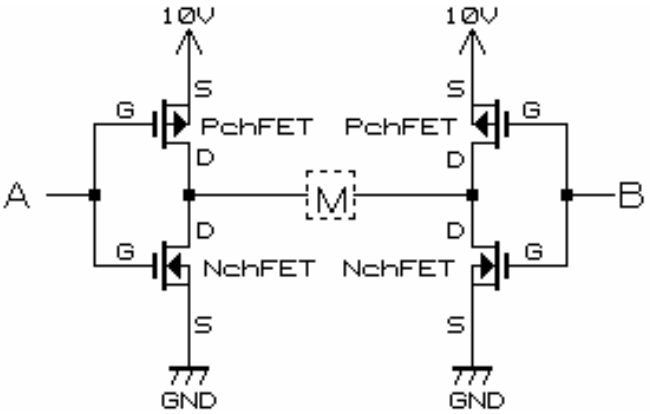


ブレーキ2



注意点は、絶対に左側 2 個もしくは右側 2 個の FET を同時に ON させてはいけないと言うことです。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

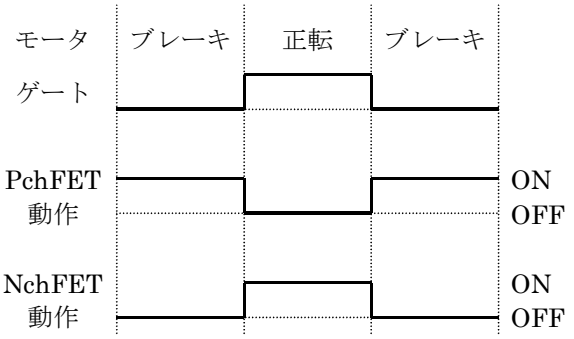
※G（ゲート）端子にはモータ用の電源電圧が 10V であったとすれば、その電圧がそのまま加えられたり 0V が加えられたりします。″0″、″1″の制御信号とは異なるので注意しましょう。

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。

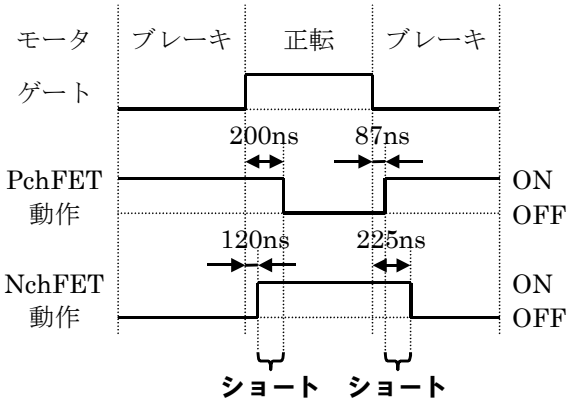
FET のゲートから信号を入力し、ドレイン・ソース間が ON／OFF するとき、事項の左図「理想的な波形」のように、P チャネル FET と N チャネル FET がすぐに反応してブレーキと正転がスムーズに切り替わりるように思えま

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

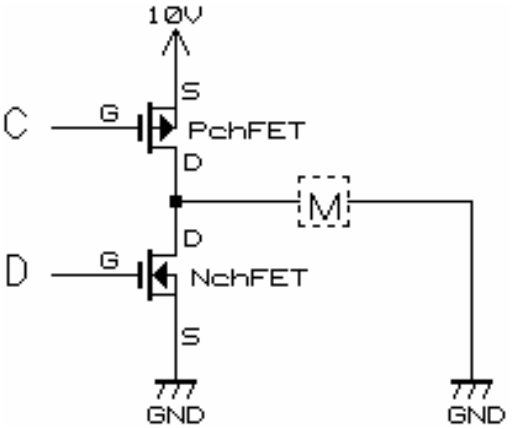
注) 1. パルス測定

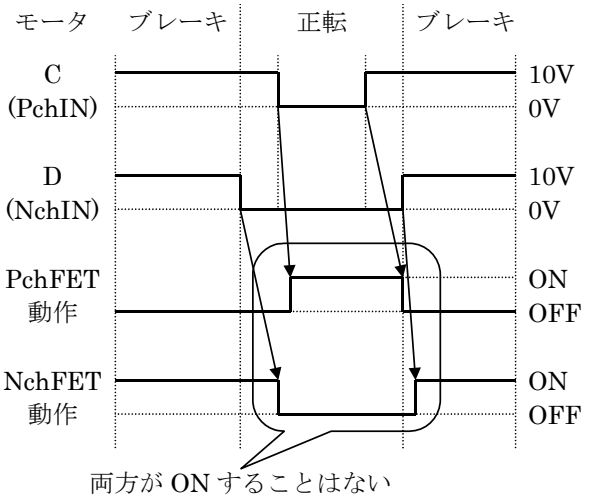
OFF→ON は 120ns 遅れる

ON→OFF は 225ns 遅れる

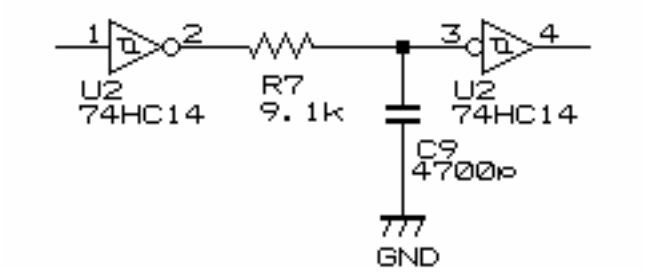
16.5.6 PチャネルとNチャネルの短絡防止回路

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。

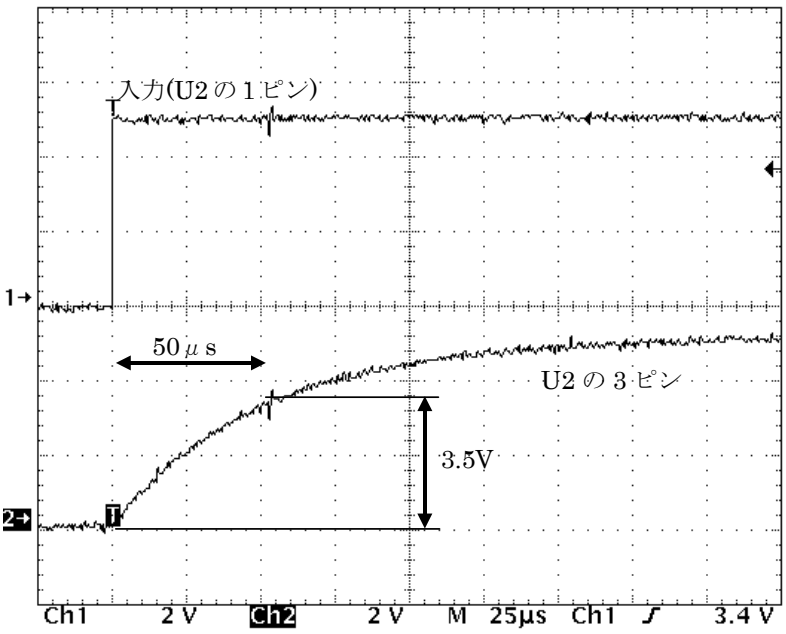




この時間をずらす部分を、積分回路で作ります。積分回路については、多数の専門書があるので、そちらを参照してください。

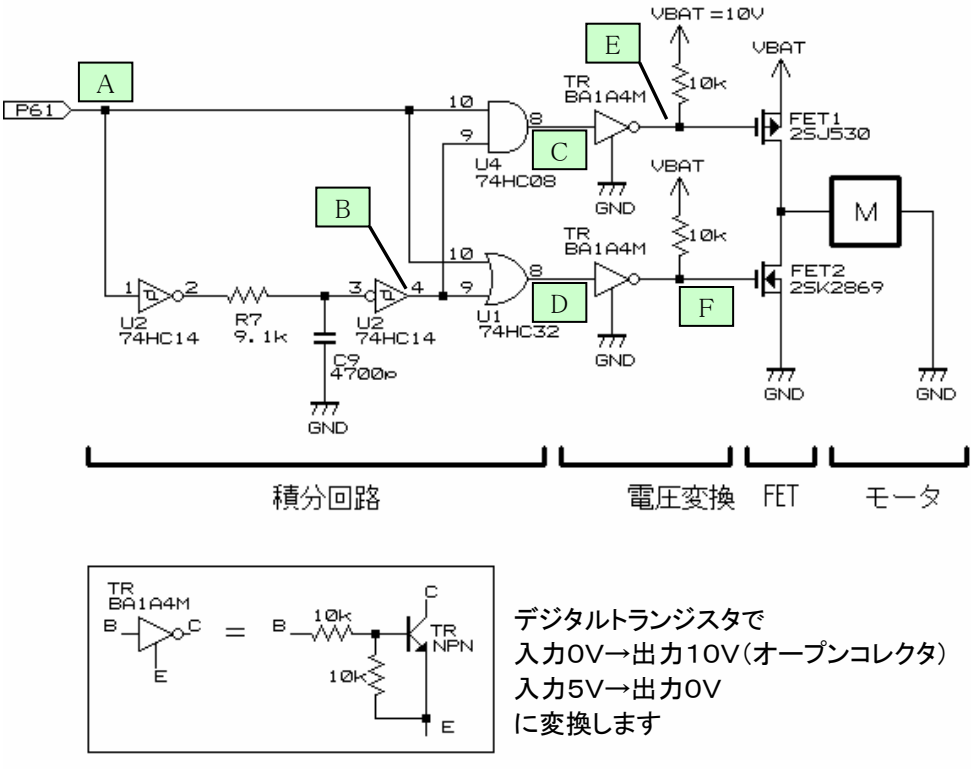


遅延時間はだいたい
時定数T＝CR［s］
で計算することができます。
今回は9.1kΩ、4700pFなので、計算すると
$T=9.1\times10^{3}\times4700\times10^{-12}$
$=42.77[\mu s]$
となります。

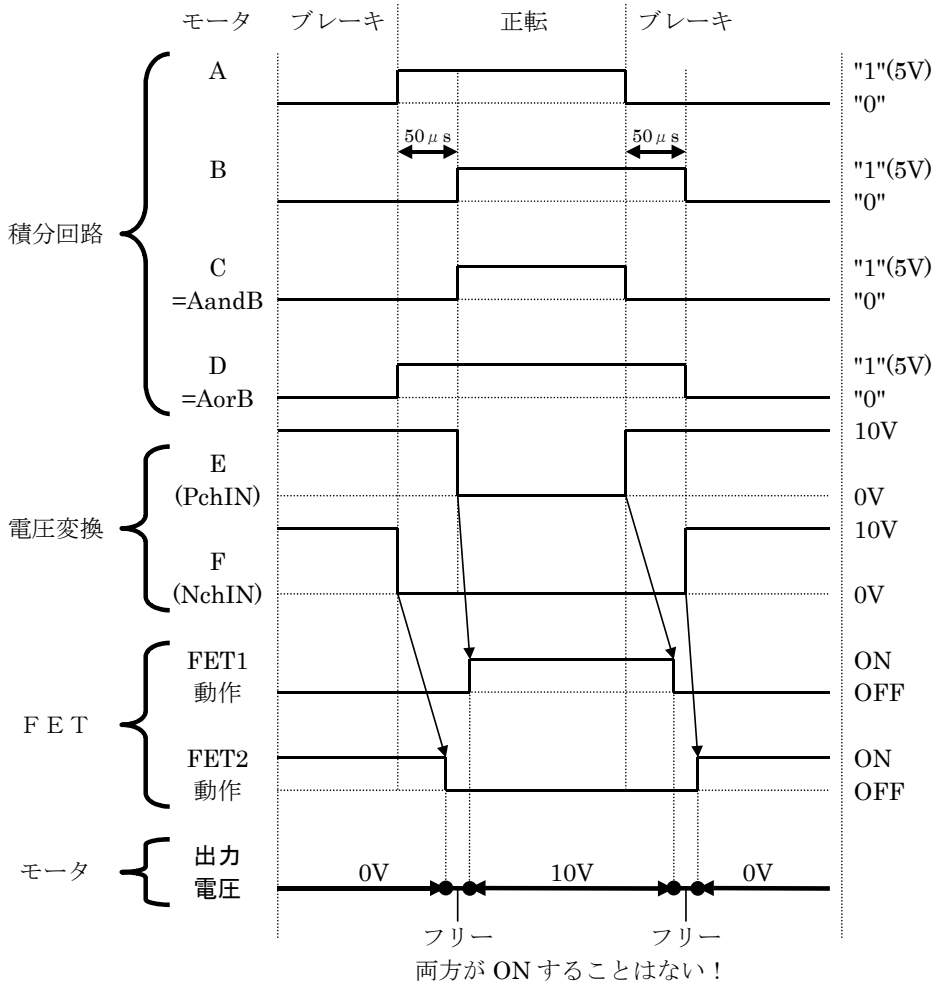


74HCシリーズは3.5V以上の入力電圧があると"1"とみなします。実際に波形を観測し、3.5Vになるまでの時間を計ると約50μsになりました。
先ほどの「実際の波形」の図では最高でも225nsのずれしかありませんが、積分回路では50μsもの遅延時間を作っています。これは、FET以外にも、電圧変換用のデジタルトランジスタの遅延時間、FETのゲートのコンデンサ成分による遅れなどを含めたためです。

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

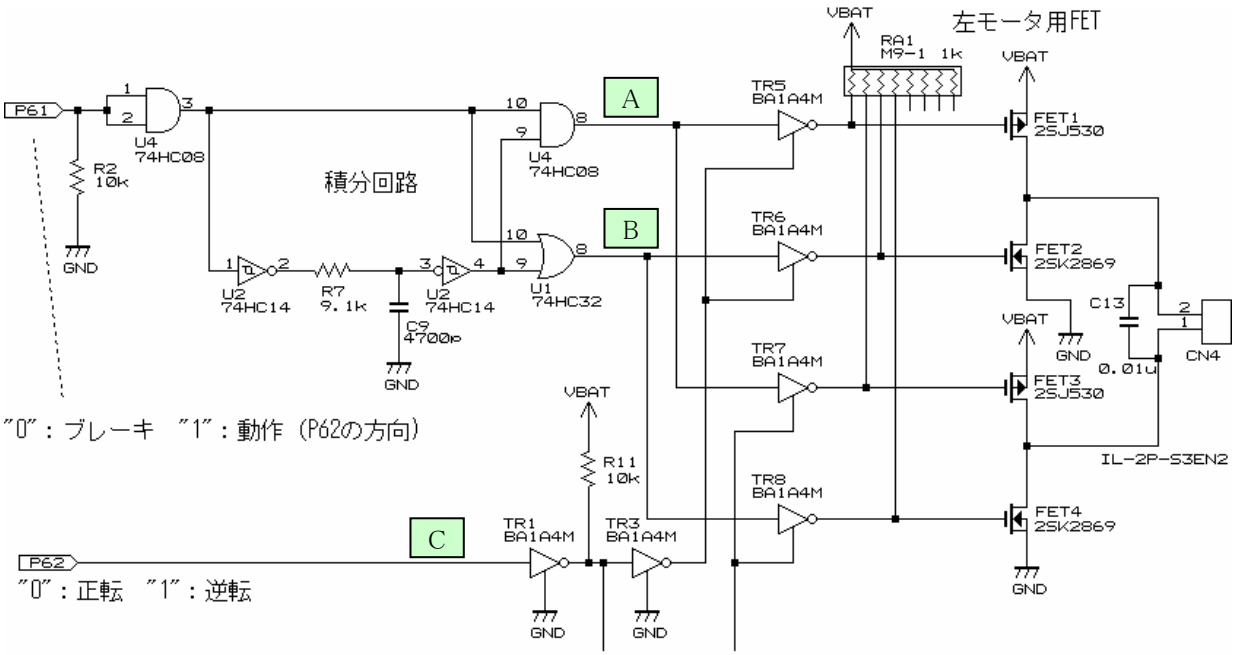
1. A点の信号を"1"→"0"にかえると、FET1 のゲートが 10V→0V となり FET1 は OFF になります。ただし、遅延時間があるため遅れて OFF になります。この時点では、FET1 も FET2 も OFF 状態のため、モータはフリー状態となります。
2. A点の信号を変えてから 50μs 後、今度は FET2 のゲートが 0V→10V となり ON します。0V がモータに加えられ、両端子 0V なのでブレーキ動作になります。

このように、動作を切り替えるときはいったん、両 FET とも 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。

16.5.7 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。P61 が PWM を加える端子、P62 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。

16.5.8 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←P67	LED1	点灯	消灯	
3	基板←P66	LED0	点灯	消灯	
4	基板←P65	サーボ信号	PWM 信号		GRD_1 でデューティ比設定
5	**基板←P64**	**右モータ PWM**	**停止**	**動作**	**GRC_1 でデューティ比設定**
6	基板←P63	右モータ回転方向	正転	逆転	
7	基板←P62	左モータ回転方向	正転	逆転	
8	**基板←P61**	**左モータ PWM**	**停止**	**動作**	**GRD_0 でデューティ比設定**
9	基板→P60	プッシュスイッチ	押された	押されてない	
10	－	GND			

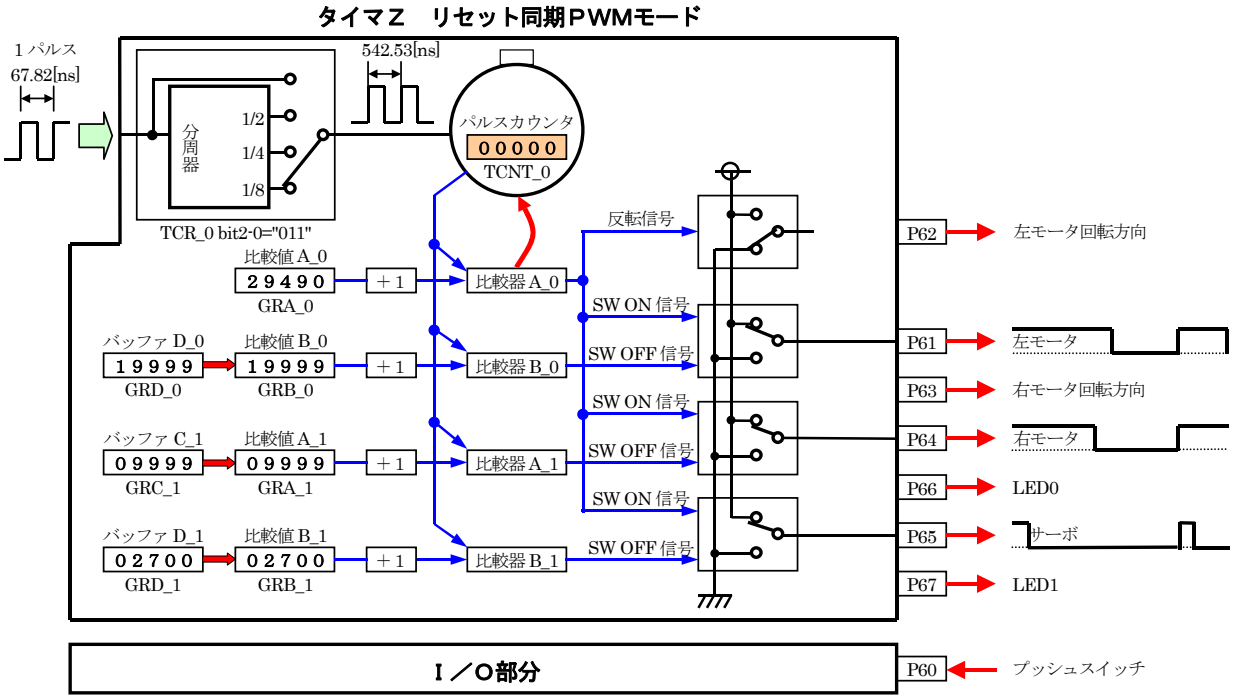
P64 が右モータ制御で、PWM 信号を加えることでスピード制御しています。右モータの正転／逆転は P63 です。

P61 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 P62 です。

16.6 プログラムの解説

16.6.1 ON幅の設定

ポート 6 の各ビットとモータドライブ基板の接続は下図のようになっています。



よって、

・左モータの ON 幅を決めるレジスタは、GRD_0

・右モータの ON 幅を決めるレジスタは、GRC_1

・サーボの ON 幅を決めるレジスタは、GRD_1

となります。それぞれのレジスタの値を設定して ON 幅を設定します。ジェネラルレジスタ B_0、A_1、B_1 の設定は最初の1回だけ行います。init 関数内でのそれぞれのレジスタの設定は、

・左モータは停止させるので、

GRD_0=0

・右モータは停止させるので、

GRC_1=0

・サーボは、0 度になるよう設定、ON 幅は 1.5[ms]なので、

$(1.5\times10^{-3})\div(542.53\times10^{-9})=2765\fallingdotseq2700$

を設定します（下記）。

```
65 :      GRB_0  = GRD_0 = 0;          /* 左モータの PWM 設定      */
66 :      GRA_1  = GRC_1 = 0;          /* 右モータの PWM 設定      */
67 :      GRB_1  = GRD_1 = 2700;       /* サーボの PWM 設定        */
```

16.6.2 main関数

```
 31 :  void main( void )
 32 :  {
 33 :      init();                              /* マイコン機能の初期化      */
 34 :
 35 :      while( 1 ) {
 36 :          GRD_0 = (long)29491 * dipsw_get() / 15; /* 左モータ          */
 37 :      }
 38 :  }
```

左モータに繋がっている端子の PWM 波形を操作するので、GRD_0 に値を設定します。

16.7 演習

16.7.1 右モータの制御

右モータを制御するようにして見ましょう。プログラム例は下記のようです。

```
void main( void )
{
    init();                              /* マイコン機能の初期化      */

    while( 1 ) {
        GRC_1 = (long)29491 * dipsw_get() / 15; /* 右モータ          */
    }
}
```

16.7.2 モータの逆転

PDR6 に出力する値を変えて、モータを逆転させて見ましょう。プログラム例は下記のようです。

```
void main( void )
{
    init();                              /* マイコン機能の初期化      */

    PDR6 |= 0x04;                        /* 左モータ逆転              */
    while( 1 ) {
        GRD_0 = (long)29491 * dipsw_get() / 15; /* 左モータ          */
    }
}
```

17. 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

17.1 概要

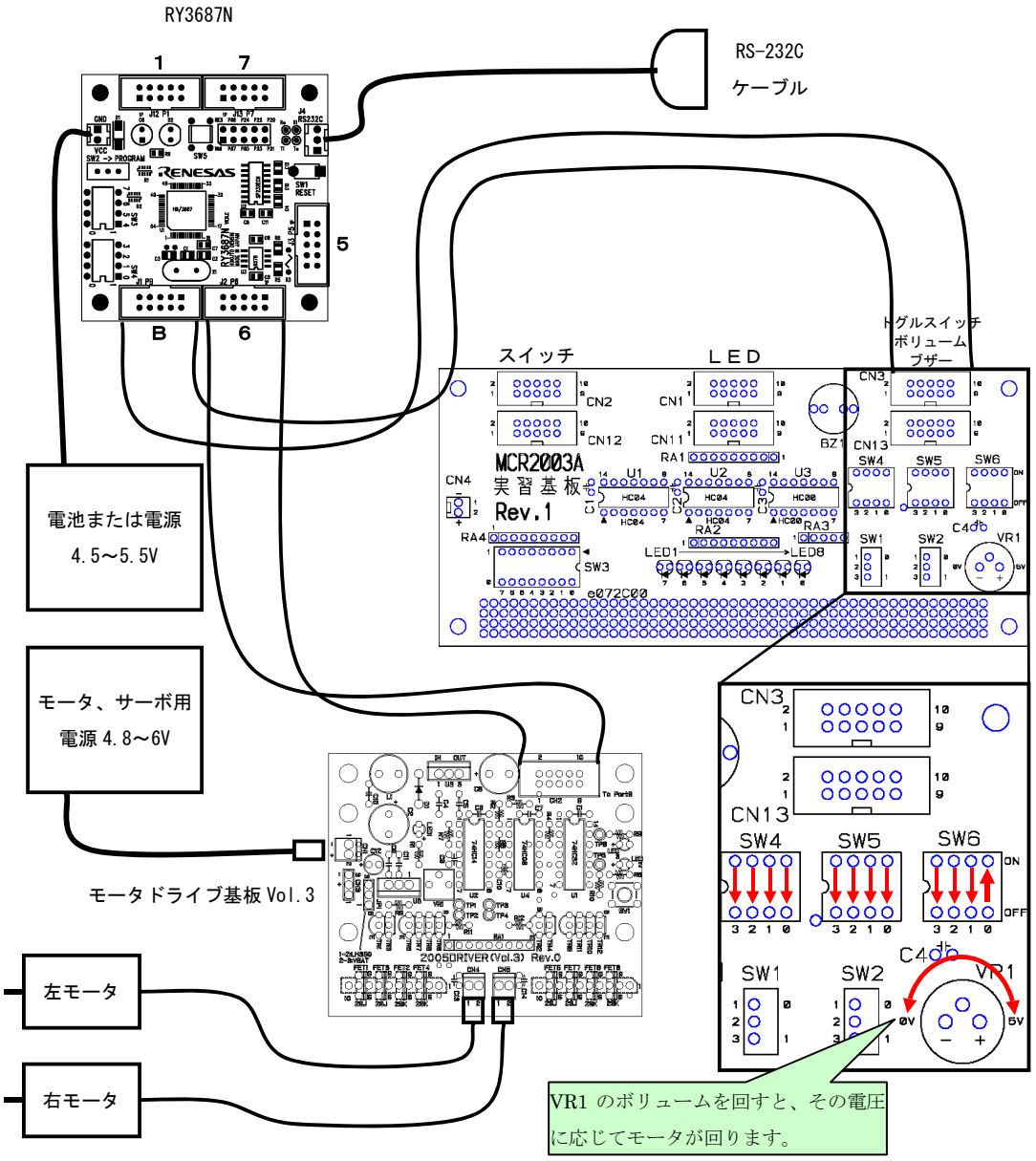
実習基板にあるボリュームの電圧をマイコンで読み込み、A/D 変換します。0V を 0%、5V を 100%として、その割合を PWM のデューティ比として P61 端子から出力します。

・ポート B の 0 ビット・・・アナログ電圧(0～5V)入力
・ポート 6 の 1 ビット・・・PWM 出力(モータドライブ基板)

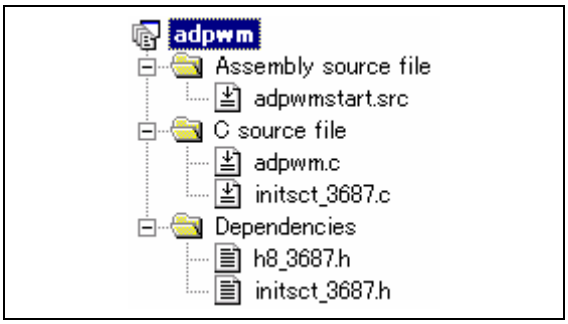
17.2 接続

・CPU ボードのポート B 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 6 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 6 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。



17.3 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_3687.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。H8/3687F の内蔵周辺機能の初期化も行います。
4	h8_3687.h	H8/3687F 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_3687.h	initsct_3687.c のヘッダファイルです。

17.4 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 :  /*                          2007.04 ジャパンマイコンカーラリー実行委員会 */
 4 :  /*********************************************************************/
 5 :  /*
 6 :  入力：PB0(0～5V)
 7 :  出力：P67-P60(モータドライブ基板、無い場合はLED)
 8 :
 9 :  ポートB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_3687.h"
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 :
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 :          GRD_0 = (long)29491 * ad / 255; /* P61(左モータ)のPWM設定   */
41 :      }
42 :  }
43 :
44 :  /************************************************************************/
45 :  /* H8/3687F 内蔵周辺機能  初期化                                        */
46 :  /************************************************************************/
47 :  void init( void )
48 :  {
49 :      /* I/Oポートの入出力設定 */
50 :      PCR1 = 0xff;
51 :      PCR2 = 0xfd;                        /* 通信ビットP22:TxD P21:RxD*/
52 :      PCR3 = 0xf0;                        /* 基板上のディップスイッチ */
53 :      PCR5 = 0xff;
54 :      PCR6 = 0xfe;                        /* モータドライブ基板       */
55 :      PCR7 = 0xff;
56 :      PCR8 = 0xff;
57 :      /* ポートBは、入力専用なので入出力設定はありません。          */
58 :      /* また、A/D変換器のADCSRによって指定された1ビットだけは、   */
59 :      /* 入力ポートとしては使えません。                             */
60 :
61 :      /* タイマZ ch0,ch1 リセット同期PWMモード */
62 :      TCR_0  = 0x23;                      /* カウンタクロック設定     */
63 :      TFCR   = 0x01;                      /* リセット同期PWMモード    */
64 :      TOCR   = 0x00;                      /* PWM信号の出力設定        */
65 :      TCNT_0 = 0;                         /* カウンタクリア           */
66 :      TMDR   = 0xe0;                      /* バッファ動作設定         */
67 :      GRA_0  = 29490;                     /* 周期の設定               */
68 :      GRB_0  = GRD_0 = 0;                 /* 左モータのPWM設定        */
69 :      GRA_1  = GRC_1 = 0;                 /* 右モータのPWM設定        */
70 :      GRB_1  = GRD_1 = 2700;              /* サーボのPWM設定          */
71 :      TOER   = 0xcd;                      /* 出力端子の設定           */
72 :      TSTR   = 0x01;                      /* タイマスタート           */
73 :
74 :      /* A/Dの初期設定 */
75 :      ADCSR = 0x00;
76 :  }
77 :
78 :  /************************************************************************/
79 :  /* A/D値読み込み(AN0)                                                   */
80 :  /* 引数    なし                                                         */
81 :  /* 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     */
82 :  /************************************************************************/
83 :  int get_ad( void )
84 :  {
85 :      int i;
86 :
87 :      ADCSR |= 0x20;                      /* ADスタート               */
88 :      while( !(ADCSR & 0x80) );           /* エンドフラグをチェック   */
89 :      ADCSR &= 0x7f;                      /* エンドフラグクリア       */
90 :      i = ADDRA >> 8;                     /* 代入                     */
91 :
92 :      return i;
93 :  }
94 :
95 :  /************************************************************************/
96 :  /* End of file                                                          */
97 :  /************************************************************************/
```

17.5 プログラムの解説

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。

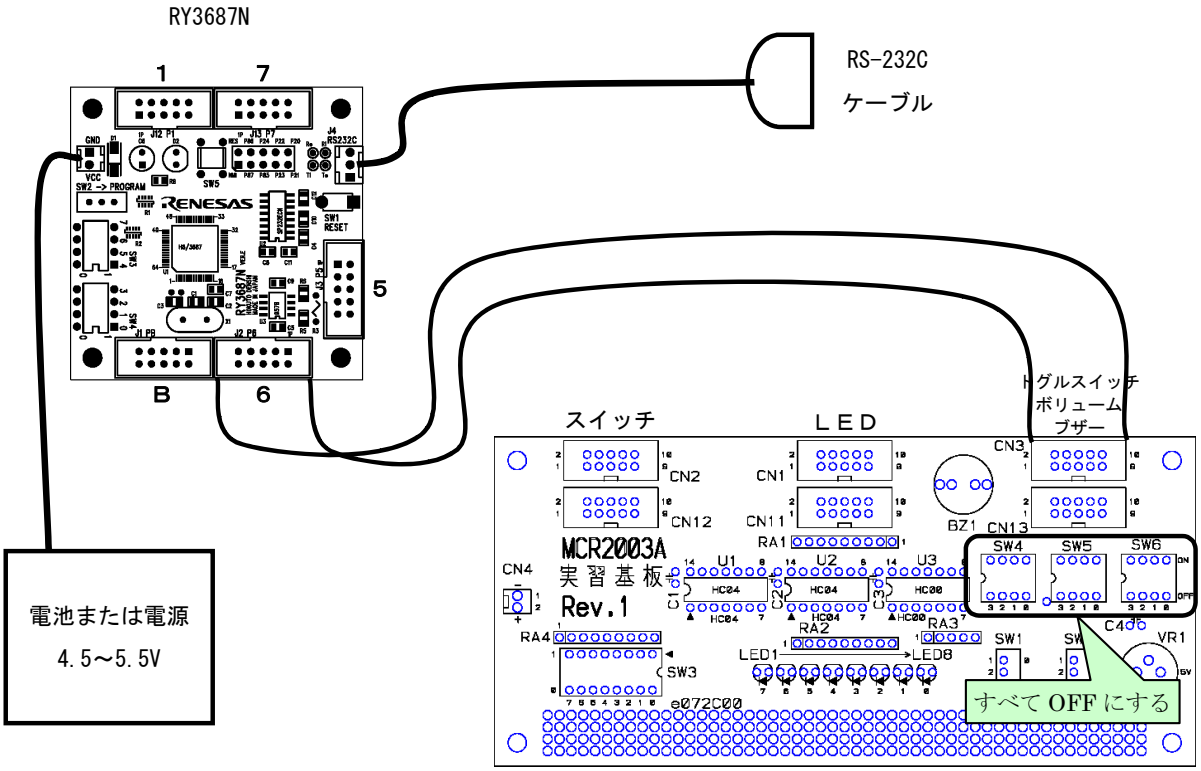
18. プロジェクト「beep」 ブザーを鳴らす

18.1 概要

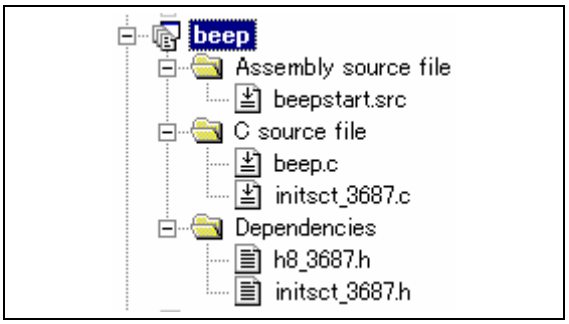
PWM 機能を使って、圧電ブザーの音を鳴らします。マイコンのポートは、下記を使用します。
・ポート 6 の bit6・・・圧電ブザーやアンプなど

18.2 接続

・実習基板の SW4～SW6 はすべて OFF にします。
・CPU ボードのポート 6 と実習基板のトグルスイッチ、ボリューム、ブザー部を、フラットケーブルで接続します。



18.3 プロジェクトの構成



	ファイル名	内容
1	beepstart.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_3687.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.c	実際に制御するプログラムが書かれています。H8/3687F の内蔵周辺機能の初期化も行います。
4	h8_3687.h	H8/3687F 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_3687.h	initsct_3687.c のヘッダファイルです。

18.4 プログラム「beep.c」

```
 1 :  /*********************************************************************/
 2 :  /* 音階出力 "beep.c"                                                 */
 3 :  /*                          2007.04 ジャパンマイコンカーラリー実行委員会 */
 4 :  /*********************************************************************/
 5 :  /*
 6 :  出力：P66(スピーカや圧電ブザーなど)
 7 :
 8 :  ポート6のbit6に接続したスピーカや圧電ブザーに
 9 :  音階を出力します。
10 :  */
11 :
12 :  /*====================================*/
13 :  /* インクルード                       */
14 :  /*====================================*/
15 :  #include    <machine.h>
16 :  #include    "h8_3687.h"
17 :
18 :  /*====================================*/
19 :  /* シンボル定義                       */
20 :  /*====================================*/
21 :  #define         M_DO        7045        /* ド                  */
22 :  #define         M_DOU       6648        /* ド＃                */
23 :  #define         M_RE        6277        /* レ                  */
24 :  #define         M_REU       5924        /* レ＃                */
25 :  #define         M_MI        5598        /* ミ                  */
26 :  #define         M_FA        5277        /* ファ                */
27 :  #define         M_FAU       4981        /* ファ＃              */
28 :  #define         M_SO        4701        /* ソ                  */
29 :  #define         M_SOU       4437        /* ソ＃                */
30 :  #define         M_RA        4188        /* ラ                  */
31 :  #define         M_RAU       3954        /* ラ＃                */
32 :  #define         M_SI        3732        /* シ                  */
33 :  #define         H1_DO       3522        /* ド                  */
34 :
35 :  /*====================================*/
36 :  /* プロトタイプ宣言                   */
37 :  /*====================================*/
38 :  void init( void );
```

```
 39 :  void timer( unsigned long timer_set );
 40 :  void note( unsigned int tone );
 41 :
 42 :  /*====================================*/
 43 :  /* グローバル変数の宣言               */
 44 :  /*====================================*/
 45 :
 46 :  /************************************************************************/
 47 :  /* メインプログラム                                                     */
 48 :  /************************************************************************/
 49 :  void main( void )
 50 :  {
 51 :      init();                             /* マイコン機能の初期化     */
 52 :      set_ccr( 0x00 );                    /* 全体割り込み許可         */
 53 :
 54 :      while( 1 ) {
 55 :          note( M_DO  );
 56 :          timer( 1000 );
 57 :          note( M_RE  );
 58 :          timer( 1000 );
 59 :          note( M_MI  );
 60 :          timer( 1000 );
 61 :          note( M_FA  );
 62 :          timer( 1000 );
 63 :          note( M_SO  );
 64 :          timer( 1000 );
 65 :          note( M_RA  );
 66 :          timer( 1000 );
 67 :          note( M_SI  );
 68 :          timer( 1000 );
 69 :          note( H1_DO );
 70 :          timer( 1000 );
 71 :          note( 0     );
 72 :          timer( 1000 );
 73 :      }
 74 :  }
 75 :
 76 :  /************************************************************************/
 77 :  /* H8/3687F 内蔵周辺機能  初期化                                        */
 78 :  /************************************************************************/
 79 :  void init( void )
 80 :  {
 81 :      /* I/Oポートの入出力設定 */
 82 :      PCR1 = 0xff;
 83 :      PCR2 = 0xfd;                        /* 通信ビットP22:TxD P21:RxD*/
 84 :      PCR3 = 0xf0;                        /* 基板上のディップスイッチ */
 85 :      PCR5 = 0xff;
 86 :      PCR6 = 0xff;                        /* P66:ブザー               */
 87 :      PCR7 = 0xff;
 88 :      PCR8 = 0xff;
 89 :      /* ポートBは、入力専用なので入出力設定はありません。         */
 90 :      /* また、A/D変換器のADCSRによって指定された1ビットだけは、   */
 91 :      /* 入力ポートとしては使えません。                           */
 92 :
 93 :      /* タイマZ ch1 */
 94 :      TCR_1  = 0x23;                      /* カウンタクロック設定     */
 95 :      TPMR   = 0x20;                      /* PWMモードの設定          */
 96 :      TOCR   = 0x00;                      /* 初期出力値設定           */
 97 :      POCR_1 = 0x00;                      /* 出力レベル設定           */
 98 :      GRA_1  = 1;                         /* 周期の設定               */
 99 :      GRC_1  = 1;                         /* デューティ比設定         */
100 :      TOER   = 0xb1;                      /* PWM出力の許可            */
101 :  }
102 :
103 :  /************************************************************************/
104 :  /* 音を鳴らす                                                           */
105 :  /* 引数   トーンのPWM値                                                 */
106 :  /* 戻り値 なし                                                          */
107 :  /************************************************************************/
108 :  void note( unsigned int tone )
109 :  {
110 :      if( tone ) {
111 :          TSTR  &= 0xfd;                  /* PWM停止                  */
112 :          TCNT_1 = tone - 1;
113 :          GRA_1  = tone;                  /* トーン設定               */
114 :          GRC_1  = tone / 2;              /* デューティ比は５０％     */
115 :          TSTR  |= 0x02;                  /* PWM開始                  */
116 :      } else {
117 :          TSTR  &= 0xfd;                  /* PWM停止                  */
118 :          TCNT_0 = 0;
119 :          GRA_1  = 1;
120 :          GRC_1  = 1;
121 :          TSTR  |= 0x02;                  /* PWM開始                  */
122 :      }
123 :  }
124 :
125 :  /************************************************************************/
126 :  /* タイマ本体                                                           */
127 :  /* 引数   タイマ値 1=1ms                                                */
128 :  /* 戻り値 なし                                                          */
129 :  /************************************************************************/
```

```
130 :  void timer( unsigned long timer_set )
131 :  {
132 :  #pragma option speed = noloop
133 :
134 :      unsigned long m, n;
135 :
136 :      for( m=0; m<timer_set; m++ ) {
137 :          for( n=0; n<2456; n++ );
138 :      }
139 :
140 :  #pragma option speed
141 :  }
142 :
143 :  /*****************************************************************/
144 :  /* End of file                                                   */
145 :  /*****************************************************************/
```

18.5 音階

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]	GRA_1 の値
ド	0	$261.6\times2^{(0/12)}$	261.6	7045
ド＃	1	$261.6\times2^{(1/12)}$	277.2	6648
レ	2	$261.6\times2^{(2/12)}$	293.6	6277
レ＃	3	$261.6\times2^{(3/12)}$	311.1	5924
ミ	4	$261.6\times2^{(4/12)}$	329.6	5598
ファ	5	$261.6\times2^{(5/12)}$	349.2	5277
ファ＃	6	$261.6\times2^{(6/12)}$	370.0	4981
ソ	7	$261.6\times2^{(7/12)}$	392.0	4701
ソ＃	8	$261.6\times2^{(8/12)}$	415.3	4437
ラ	9	$261.6\times2^{(9/12)}$	440.0	4188
ラ＃	10	$261.6\times2^{(10/12)}$	466.1	3954
シ	11	$261.6\times2^{(11/12)}$	493.8	3732
ド	12	$261.6\times2^{(12/12)}$	523.2	3522

「ド」を例にジェネラルレジスタ A_1(GRA_1)の値を計算すると、表より周期は 261.6[Hz]なので

周期＝1／周波数＝1/261.6＝3.823[ms]

となります。タイマカウンタ_1(TCNT_1)がカウントアップする間隔は、542.53[ns]なので、「ド」の周期分のカウント値は、

$(3.823\times10^{-3})／(542.53\times10^{-9})＝7046$

となります。GRA_1 に設定する値は1小さい値にするので最終的には、7045 を代入します。すべて計算すると、上表のようになります。

ちなみに高い音階は、xの値が 12、13、14・・・と増えて行きます。低い音階はxの値が-1、-2、-3・・・と減っていきます。エクセルなどで表にして、計算してみてください。

18.6 プログラムの解説

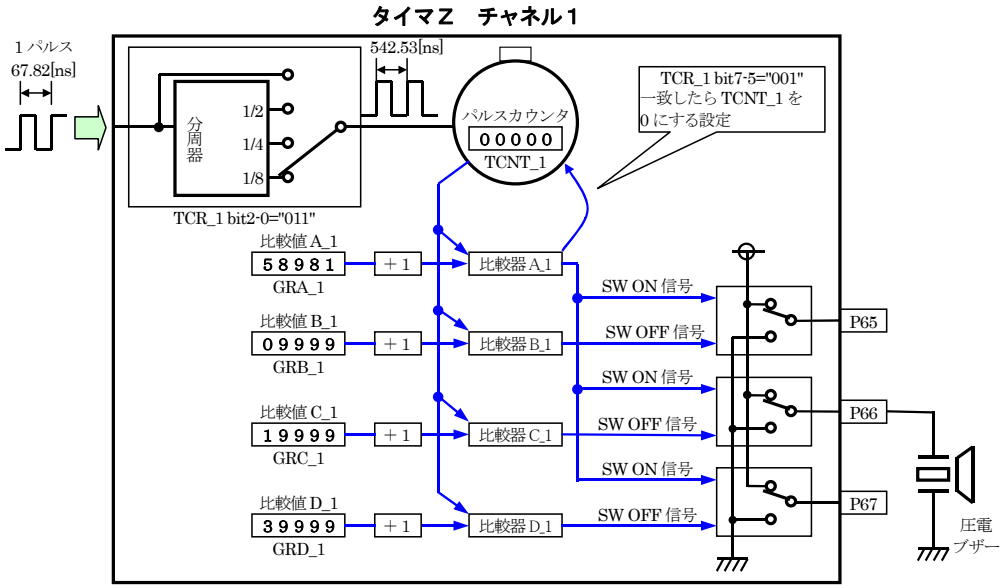
18.6.1 どのタイマを使うか

実習基板のブザーは、3 ピンに繋がっています。これは、各ポートの bit6 です。
タイマにより PWM 信号を出力できる端子は決まっています。自由に決めることはできません。下表のように bit6 に出力できる内蔵周辺機能は、タイマ Z のチャネル 1 とタイマ V です。

内蔵周辺機能	タイマ Z チャネル 0	タイマ Z チャネル 1	タイマ V	14 ビット PWM
PWM 出力数	3	3	1	1
出力端子	FTIOB0(P61)端子 FTIOC0(P62)端子 FTIOD0(P63)端子	FTIOB1(P65)端子 FTIOC1(**P66**)端子 FTIOD1(P67)端子	TMOV(**P76**)端子	PWM(P11)端子
カウンタのビット	16bit	16bit	8bit	8bit (周期が 2 つしか選べない)

この演習では、タイマ Z のチャネル 1 を使って P66 端子から PWM 信号を出力、実習基板の圧電ブザーと接続します。

18.6.2 タイマZチャネル 1 の設定



・タイマカウンタ_1（TCNT_1）のカウント間隔
周期によって、分周器の設定をどうするか決めますが、周期は音階によって変わります。そのため、長く設定できるよういちばん周期の長い 1/8 を設定します。

・周期
周期は、ジェネラルレジスタ A_1（GRA_1）に設定します。

・ON 幅
P66 端子に出力するので、ジェネラルレジスタ C_1（GRC_1）に ON 幅を設定します。ON 幅は周期の 1/2 になります。最初は、常に OFF になるように設定します。

・PWM 出力端子
PWM 出力端子は P66 のみです。その他の端子は特に出力する必要はないので、通常の I/O 端子に設定します。

これらから、タイマ Z のチャネル 1 関係のレジスタには、下記を設定します。

```
 93 :      /* タイマ Z ch1 */
 94 :      TCR_1  = 0x23;                  /* カウンタクロック設定      */
 95 :      TPMR   = 0x20;                  /* PWM モードの設定          */
 96 :      TOCR   = 0x00;                  /* 初期出力値設定            */
 97 :      POCR_1 = 0x00;                  /* 出力レベル設定            */
 98 :      GRA_1  = 1;                     /* 周期の設定                */
 99 :      GRC_1  = 1;                     /* デューティ比設定          */
100 :      TOER   = 0xb1;                  /* PWM 出力の許可            */
```

18.6.3 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。

```
 21 :  #define        M_DO       7045      /* ド                  */
 22 :  #define        M_DOU      6648      /* ド＃                */
 23 :  #define        M_RE       6277      /* レ                  */
 24 :  #define        M_REU      5924      /* レ＃                */
 25 :  #define        M_MI       5598      /* ミ                  */
 26 :  #define        M_FA       5277      /* ファ                */
 27 :  #define        M_FAU      4981      /* ファ＃              */
 28 :  #define        M_SO       4701      /* ソ                  */
 29 :  #define        M_SOU      4437      /* ソ＃                */
 30 :  #define        M_RA       4188      /* ラ                  */
 31 :  #define        M_RAU      3954      /* ラ＃                */
 32 :  #define        M_SI       3732      /* シ                  */
 33 :  #define        H1_DO      3522      /* ド                  */
```

18.6.4 音を鳴らすnote関数

音を鳴らす note 関数を作りました。引数は、音階の PWM 値を設定します。

```
        note(音階の PWM 値 );
```

時間は、プロジェクト「timer1」で演習した、timer 関数を使います。例えば、「レ」を 0.3 秒鳴らす場合は、

```
        note( M_RE );
        timer( 300 );
```

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。

19. プロジェクト「beep_haru」 ブザーを鳴らす応用「春の小川」を演奏

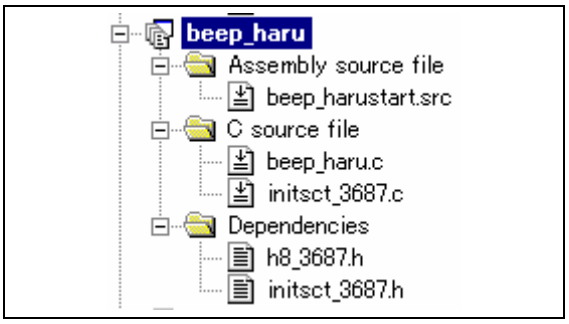
19.1 概要

プロジェクト「beep」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。

19.2 接続

プロジェクト「beep」と同様です。

19.3 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_3687.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。H8/3687F の内蔵周辺機能の初期化も行います。
4	h8_3687.h	H8/3687F 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_3687.h	initsct_3687.c のヘッダファイルです。

19.4 プログラム「beep_haru.c」

```
/*********************************************************************/
/* 音階出力 春が来た版 "beep_haru.c"                                 */
/*                          2007.04 ジャパンマイコンカーラリー実行委員会 */
/*********************************************************************/
/*
出力：P66(スピーカや圧電ブザーなど)

ポート6のbit6に接続したスピーカや圧電ブザーに
音階を出力します。
*/

/*====================================*/
/* インクルード                       */
/*====================================*/
#include    <machine.h>
#include    "h8_3687.h"

/*====================================*/
/* シンボル定義                       */
/*====================================*/
/* 低い音階１ */
#define         L1_DO       14092       /* ド                 */
#define         L1_DOU      13301       /* ド＃               */
#define         L1_RE       12554       /* レ                 */
#define         L1_REU      11850       /* レ＃               */
#define         L1_MI       11185       /* ミ                 */
#define         L1_FA       10557       /* ファ               */
#define         L1_FAU      9964        /* ファ＃             */
#define         L1_SO       9405        /* ソ                 */
#define         L1_SOU      8877        /* ソ＃               */
#define         L1_RA       8379        /* ラ                 */
#define         L1_RAU      7909        /* ラ＃               */
#define         L1_SI       7465        /* シ                 */

/* 標準の音階 */
#define         M_DO        7045        /* ド                 */
#define         M_DOU       6648        /* ド＃               */
#define         M_RE        6277        /* レ                 */
#define         M_REU       5924        /* レ＃               */
#define         M_MI        5598        /* ミ                 */
#define         M_FA        5277        /* ファ               */
#define         M_FAU       4981        /* ファ＃             */
#define         M_SO        4701        /* ソ                 */
#define         M_SOU       4437        /* ソ＃               */
#define         M_RA        4188        /* ラ                 */
#define         M_RAU       3954        /* ラ＃               */
#define         M_SI        3732        /* シ                 */

/* 高い音階１ */
#define         H1_DO       3523        /* ド                 */
#define         H1_DOU      3325        /* ド＃               */
#define         H1_RE       3139        /* レ                 */
#define         H1_REU      2962        /* レ＃               */
#define         H1_MI       2796        /* ミ                 */
#define         H1_FA       2639        /* ファ               */
#define         H1_FAU      2491        /* ファ＃             */
#define         H1_SO       2351        /* ソ                 */
#define         H1_SOU      2219        /* ソ＃               */
#define         H1_RA       2095        /* ラ                 */
#define         H1_RAU      1977        /* ラ＃               */
#define         H1_SI       1866        /* シ                 */

#define         TEMPO       100         /* テンポ             */

/*====================================*/
/* プロトタイプ宣言                   */
/*====================================*/
void init( void );
void note( unsigned int tone );

/*====================================*/
/* グローバル変数の宣言               */
/*====================================*/
unsigned long   cnt0;                   /* タイマB1           */
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] = {           /* 春の小川 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*/

```

```
 86 :      M_MI,    4,  /* は */
 87 :      M_SO,    4,  /* ー */
 88 :      M_RA,    4,  /* る */
 89 :      M_SO,    4,  /* の */
 90 :      M_MI,    4,  /* お */
 91 :      M_SO,    4,  /* が */
 92 :      H1_DO,   4,  /* わ */
 93 :      H1_DO,   4,  /* は */
 94 :      M_RA,    4,  /* さ */
 95 :      M_RA,    4,  /* ら */
 96 :      M_SO,    4,  /* さ */
 97 :      M_MI,    4,  /* ら */
 98 :      M_DO,    4,  /* い */
 99 :      M_RE,    4,  /* く */
100 :      M_MI,    4,  /* よ */
101 :      0,       4,
102 :
103 :      M_MI,    4,  /* き */
104 :      M_SO,    4,  /* ー */
105 :      M_RA,    4,  /* し */
106 :      M_SO,    4,  /* の */
107 :      M_MI,    4,  /* す */
108 :      M_SO,    4,  /* み */
109 :      H1_DO,   4,  /* れ */
110 :      H1_DO,   4,  /* や */
111 :      M_RA,    4,  /* れ */
112 :      M_RA,    4,  /* ん */
113 :      M_SO,    4,  /* げ */
114 :      M_MI,    4,  /* の */
115 :      M_RE,    4,  /* は */
116 :      M_MI,    4,  /* な */
117 :      M_DO,    4,  /* に */
118 :      0,       4,
119 :
120 :      M_RE,    4,  /* す */
121 :      M_MI,    4,  /* ー */
122 :      M_RE,    4,  /* が */
123 :      M_SO,    4,  /* た */
124 :      M_RA,    4,  /* や */
125 :      M_RA,    4,  /* さ */
126 :      M_SO,    4,  /* し */
127 :      M_RA,    4,  /* く */
128 :      H1_DO,   4,  /* い */
129 :      H1_DO,   4,  /* ろ */
130 :      M_SI,    4,  /* う */
131 :      M_RA,    4,  /* つ */
132 :      M_SO,    4,  /* く */
133 :      M_SO,    4,  /* し */
134 :      M_MI,    4,  /* く */
135 :      0,       4,
136 :
137 :      M_MI,    4,  /* さ */
138 :      M_SO,    4,  /* ー */
139 :      M_RA,    4,  /* い */
140 :      M_SO,    4,  /* て */
141 :      M_MI,    4,  /* い */
142 :      M_SO,    4,  /* る */
143 :      H1_DO,   4,  /* ね */
144 :      H1_DO,   4,  /* と */
145 :      M_RA,    4,  /* ささ */
146 :      M_RA,    4,  /* ささ */
147 :      M_SO,    4,  /* や */
148 :      M_MI,    4,  /* き */
149 :      M_RE,    4,  /* な */
150 :      M_MI,    4,  /* が */
151 :      M_DO,    4,  /* ら */
152 :      0,       4,
153 :
154 :      -1,      0   /* 終了 */
155 :  };
156 :
157 :  /************************************************************************/
158 :  /* メインプログラム                                                     */
159 :  /************************************************************************/
160 :  void main( void )
161 :  {
162 :      init();                             /* マイコン機能の初期化      */
163 :      set_ccr( 0x00 );                    /* 全体割り込み許可          */
164 :
165 :      music_flag = 1;
166 :
167 :      while( 1 ) {
168 :      }
169 :  }
170 :
171 :  /************************************************************************/
172 :  /* H8/3687F 内蔵周辺機能　初期化                                        */
173 :  /************************************************************************/
174 :  void init( void )
175 :  {
176 :      /* I/Oポートの入出力設定 */
```

```
177 :      PCR1 = 0xff;
178 :      PCR2 = 0xfd;                        /* 通信ビットP22:TxD P21:RxD*/
179 :      PCR3 = 0xf0;                        /* 基板上のディップスイッチ */
180 :      PCR5 = 0xff;
181 :      PCR6 = 0xff;                        /* P66:ブザー               */
182 :      PCR7 = 0xff;
183 :      PCR8 = 0xff;
184 :      /* ポートBは、入力専用なので入出力設定はありません。          */
185 :      /* また、A/D変換器のADCSRによって指定された1ビットだけは、   */
186 :      /* 入力ポートとしては使えません。                            */
187 :
188 :      /* タイマB1 1msごとの割り込み設定 */
189 :      TMB1 = 0x84;                        /* 入力クロックの設定等     */
190 :      TLB1 = 26;                          /* カウンタ初期値設定       */
191 :      IENR2 = 0x20;                       /* 割り込み要求許可         */
192 :
193 :      /* タイマZ ch1 */
194 :      TCR_1  = 0x23;                      /* カウンタクロック設定     */
195 :      TPMR   = 0x20;                      /* PWMモードの設定          */
196 :      TOCR   = 0x00;                      /* 初期出力値設定           */
197 :      POCR_1 = 0x00;                      /* 出力レベル設定           */
198 :      GRA_1  = 1;                         /* 周期の設定               */
199 :      GRC_1  = 1;                         /* デューティ比設定         */
200 :      TOER   = 0xb1;                      /* PWM出力の許可            */
201 :  }
202 :
203 :  /************************************************************************/
204 :  /* 音を鳴らす                                                           */
205 :  /* 引数   トーンのPWM値                                                 */
206 :  /* 戻り値 なし                                                          */
207 :  /************************************************************************/
208 :  void note( unsigned int tone )
209 :  {
210 :      if( tone ) {
211 :          TSTR  &= 0xfd;                  /* PWM停止                  */
212 :          TCNT_1 = tone - 1;
213 :          GRA_1  = tone;                  /* トーン設定               */
214 :          GRC_1  = tone / 2;              /* デューティ比は５０％     */
215 :          TSTR  |= 0x02;                  /* PWM開始                  */
216 :      } else {
217 :          TSTR  &= 0xfd;                  /* PWM停止                  */
218 :          TCNT_0 = 0;
219 :          GRA_1  = 1;
220 :          GRC_1  = 1;
221 :          TSTR  |= 0x02;                  /* PWM開始                  */
222 :      }
223 :  }
224 :
225 :  /************************************************************************/
226 :  /* タイマB1 割り込み処理                                                */
227 :  /************************************************************************/
228 :  #pragma interrupt( interrupt_timerB1 )
229 :  void interrupt_timerB1( void )
230 :  {
231 :      IRR2 &= 0xdf;                       /* フラグクリア             */
232 :      cnt0++;
233 :
234 :      if( music_flag ) {
235 :          /* 音階処理 */
236 :          if( cnt0 >= 15000L*music_data[music_dim][1]/TEMPO ) {
237 :              /* 次の音階をならす */
238 :              cnt0 = 0;
239 :              music_dim++;
240 :              if( music_data[music_dim][0] == -1 ) {
241 :                  /* -1なら終了 */
242 :                  note( 0 );
243 :                  music_flag = 0;
244 :              } else {
245 :                  /* -1でないなら次の音階セット */
246 :                  note( music_data[music_dim][0] );
247 :              }
248 :          }
249 :      }
250 :  }
251 :
252 :  /************************************************************************/
253 :  /* End of file                                                          */
254 :  /************************************************************************/
```

19.5 プログラムの解説

19.5.1 シンボル定義

```
  21 :  /* 低い音階１ */
  22 :  #define        L1_DO       14092       /* ド                  */
  23 :  #define        L1_DOU      13301       /* ド＃                */
中略
  35 :  /* 標準の音階 */
  36 :  #define        M_DO        7045        /* ド                  */
  37 :  #define        M_DOU       6648        /* ド＃                */
中略
  49 :  /* 高い音階１ */
  50 :  #define        H1_DO       3523        /* ド                  */
  51 :  #define        H1_DOU      3325        /* ド＃                */
中略
  63 :  #define        TEMPO       100         /* 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。

19.5.2 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。

19.5.3 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。

```
225 :  /************************************************************************/
226 :  /* タイマ B1 割り込み処理                                               */
227 :  /************************************************************************/
228 :  #pragma interrupt( interrupt_timerB1 )
229 :  void interrupt_timerB1( void )
230 :  {
231 :      IRR2 &= 0xdf;                       /* フラグクリア             */
232 :      cnt0++;
233 :
234 :      if( music_flag ) {
235 :          /* 音階処理 */
236 :          if( cnt0 >= 15000L*music_data[music_dim][1]/TEMPO ) {
237 :              /* 次の音階をならす */
238 :              cnt0 = 0;
239 :              music_dim++;
240 :              if( music_data[music_dim][0] == -1 ) {
241 :                  /* -1 なら終了 */
242 :                  note( 0 );
243 :                  music_flag = 0;
244 :              } else {
245 :                  /* -1 でないなら次の音階セット */
246 :                  note( music_data[music_dim][0] );
247 :              }
248 :          }
249 :      }
250 :  }
```

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

19.5.4 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。これは割り込み(タイマ B1)とタイマ Z を使わなければできません。

```
160 :  void main( void )
161 :  {
162 :      init();                             /* マイコン機能の初期化     */
163 :      set_ccr( 0x00 );                    /* 全体割り込み許可         */
164 :
165 :      music_flag = 1;
166 :
167 :      while( 1 ) {
168 :      }
169 :  }
```

演奏開始

無限ループで何もしない

20. 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

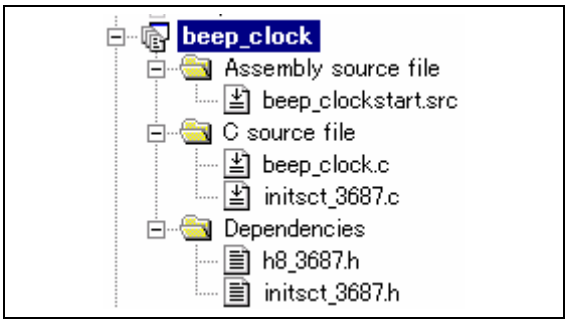
20.1 概要

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。

20.2 接続

プロジェクト「beep」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_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_3687.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。H8/3687F の内蔵周辺機能の初期化も行います。
4	h8_3687.h	H8/3687F 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_3687.h	initsct_3687.c のヘッダファイルです。

20.4 プログラム「beep_clock.c」

```
/*********************************************************************/
/* 音階出力　大きな古時計版 "beep_clokc.c"                           */
/*                          2007.04 ジャパンマイコンカーラリー実行委員会 */
/*********************************************************************/
/*
出力：P66(スピーカや圧電ブザーなど)

ポート6のbit6に接続したスピーカや圧電ブザーに
音階を出力します。
*/

/*====================================*/
/* インクルード                       */
/*====================================*/
#include    <machine.h>
#include    "h8_3687.h"

/*====================================*/
/* シンボル定義                       */
/*====================================*/
/* 低い音階１ */
#define         L1_DO       14092       /* ド                 */
#define         L1_DOU      13301       /* ド＃               */
#define         L1_RE       12554       /* レ                 */
#define         L1_REU      11850       /* レ＃               */
#define         L1_MI       11185       /* ミ                 */
#define         L1_FA       10557       /* ファ               */
#define         L1_FAU      9964        /* ファ＃             */
#define         L1_SO       9405        /* ソ                 */
#define         L1_SOU      8877        /* ソ＃               */
#define         L1_RA       8379        /* ラ                 */
#define         L1_RAU      7909        /* ラ＃               */
#define         L1_SI       7465        /* シ                 */

/* 標準の音階 */
#define         M_DO        7045        /* ド                 */
#define         M_DOU       6648        /* ド＃               */
#define         M_RE        6277        /* レ                 */
#define         M_REU       5924        /* レ＃               */
#define         M_MI        5598        /* ミ                 */
#define         M_FA        5277        /* ファ               */
#define         M_FAU       4981        /* ファ＃             */
#define         M_SO        4701        /* ソ                 */
#define         M_SOU       4437        /* ソ＃               */
#define         M_RA        4188        /* ラ                 */
#define         M_RAU       3954        /* ラ＃               */
#define         M_SI        3732        /* シ                 */

/* 高い音階１ */
#define         H1_DO       3523        /* ド                 */
#define         H1_DOU      3325        /* ド＃               */
#define         H1_RE       3139        /* レ                 */
#define         H1_REU      2962        /* レ＃               */
#define         H1_MI       2796        /* ミ                 */
#define         H1_FA       2639        /* ファ               */
#define         H1_FAU      2491        /* ファ＃             */
#define         H1_SO       2351        /* ソ                 */
#define         H1_SOU      2219        /* ソ＃               */
#define         H1_RA       2095        /* ラ                 */
#define         H1_RAU      1977        /* ラ＃               */
#define         H1_SI       1866        /* シ                 */

#define         TEMPO       60          /* テンポ             */

/*====================================*/
/* プロトタイプ宣言                   */
/*====================================*/
void init( void );
void note( unsigned int tone );

/*====================================*/
/* グローバル変数の宣言               */
/*====================================*/
unsigned long   cnt0;                   /* タイマB1           */
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。
*/
```

```
 83 :  /*  音階,    長さ    */
 84 :      0,      0,  /* スタート*/
 85 :
 86 :      M_RE,   4,  /* おお */
 87 :      M_SO,   4,  /* おき */
 88 :      M_FAU,  2,  /* きな */
 89 :      M_SO,   2,  /* なっ */
 90 :      M_RA,   4,  /* のっ */
 91 :      M_SO,   2,  /* ぽの */
 92 :      M_RA,   2,  /* のふ */
 93 :      M_SI,   2,  /* ふる */
 94 :      M_SI,   2,  /* る */
 95 :      H1_DO,  2,  /* どけ */
 96 :      M_SI,   2,  /* けい */
 97 :      M_MI,   4,  /* いお */
 98 :      M_RA,   2,  /* おじ */
 99 :      M_RA,   2,  /* じい */
100 :      M_SO,   4,  /* いさ */
101 :      M_SO,   2,  /* さん */
102 :      M_SO,   2,  /* ん */
103 :      M_FAU,  4,  /* のと */
104 :      M_MI,   2,  /* とけ */
105 :      M_FAU,  2,  /* けい */
106 :      M_SO,   10, /* い */
107 :      0,      2,
108 :
109 :      M_RE,   2,  /* ひゃ */
110 :      M_RE,   2,  /* く */
111 :      M_SO,   4,  /* ねん */
112 :      M_FAU,  2,  /* いい */
113 :      M_SO,   2,  /* いつ */
114 :      M_RA,   4,  /* もう */
115 :      M_SO,   2,  /* うご */
116 :      M_RA,   2,  /* ごい */
117 :      M_SI,   4,  /* いて */
118 :      H1_DO,  2,  /* て */
119 :      M_SI,   2,  /* いた */
120 :      M_MI,   4,  /* たこ */
121 :      M_RA,   2,  /* こじ */
122 :      M_RA,   2,  /* じ */
123 :      M_SO,   4,  /* まん */
124 :      M_SO,   2,  /* んの */
125 :      M_SO,   2,  /* の */
126 :      M_FAU,  4,  /* のと */
127 :      M_MI,   2,  /* とけ */
128 :      M_FAU,  2,  /* いさ */
129 :      M_SO,   10, /* さ */
130 :      0,      2,
131 :
132 :      M_SO,   2,  /* おじ */
133 :      M_SI,   2,  /* じ */
134 :      H1_RE,  4,  /* いさ */
135 :      M_SI,   2,  /* さん */
136 :      M_RA,   2,  /* ん */
137 :      M_SO,   4,  /* のう */
138 :      M_FAU,  2,  /* うま */
139 :      M_SO,   2,  /* まれ */
140 :      M_RA,   2,  /* れた */
141 :      M_SO,   2,  /* たあ */
142 :      M_FAU,  2,  /* あさ */
143 :      M_MI,   2,  /* さに */
144 :      M_RE,   4,  /* にか */
145 :      M_SO,   2,  /* かっ */
146 :      M_SI,   2,  /* っ */
147 :      H1_RE,  4,  /* てき */
148 :      M_SI,   2,  /* きた */
149 :      M_RA,   2,  /* たと */
150 :      M_SO,   4,  /* とけ */
151 :      M_FAU,  2,  /* けい */
152 :      M_SO,   2,  /* いさ */
153 :      M_RA,   10, /* さ */
154 :      0,      2,
155 :
156 :      M_RE,   2,  /* い */
157 :      M_SO,   2,  /* ま */
158 :      M_SO,   2,  /* は */
159 :      0,      4,
160 :      M_RA,   2,  /* もう */
161 :      0,      2,
162 :      0,      4,
163 :      M_SI,   2,  /* うご */
164 :      M_SI,   2,  /* ご */
165 :      H1_DO,  2,  /* か */
166 :      M_SI,   2,  /* ない */
167 :      M_MI,   4,  /* いそ */
168 :      M_RA,   2,  /* そ */
169 :      M_RA,   2,  /* の */
170 :      M_SO,   8,  /* とけ */
171 :      M_FAU,  8,  /* とけ */
172 :      M_SO,   8,  /* い */
173 :      0,      2,
```

beep_haru.c の音楽データを
変更しています。

```
174 :
175 :       M_RE,    2,  /* ひゃ */
176 :       M_RE,    2,  /* く */
177 :       M_SO,    4,  /* ねん */
178 :       M_RE,    2,  /* や */
179 :       M_RE,    2,  /* す */
180 :       M_MI,    2,  /* ま */
181 :       M_RE,    2,  /* ず */
182 :       M_RE,    4,  /* に */
183 :       L1_SI,   2,  /* ちっく */
184 :       0,       2,
185 :       M_RE,    2,  /* たっく */
186 :       0,       2,
187 :       L1_SI,   2,  /* ちっく */
188 :       0,       2,
189 :       M_RE,    2,  /* たっく */
190 :       M_RE,    2,  /* お */
191 :       M_SO,    4,  /* じー */
192 :       M_RE,    2,  /* さん */
193 :       M_RE,    2,  /* と */
194 :       M_MI,    4,  /* いっ */
195 :       M_RE,    2,  /* しょ */
196 :       M_RE,    2,  /* に */
197 :       L1_SI,   2,  /* ちっく */
198 :       0,       2,
199 :       M_RE,    2,  /* たっく */
200 :       0,       2,
201 :       L1_SI,   2,  /* ちっく */
202 :       0,       2,
203 :       M_RE,    2,  /* たっく */
204 :
205 :       M_RE,    2,  /* い */
206 :       M_SO,    2,  /* ま */
207 :       M_SO,    2,  /* は */
208 :       0,       4,
209 :       M_RA,    2,  /* もう */
210 :       0,       2,
211 :       0,       4,
212 :       M_SI,    2,  /* う */
213 :       M_SI,    2,  /* ご */
214 :       H1_DO,   2,  /* か */
215 :       M_SI,    2,  /* な */
216 :       M_MI,    4,  /* い */
217 :       M_RA,    2,  /* そ */
218 :       M_RA,    2,  /* の */
219 :       M_SO,    8,  /* と */
220 :       M_FAU,   8,  /* け */
221 :       M_SO,    12, /* い */
222 :       0,       4,
223 :
224 :       -1,      0   /* 終了 */
225 :   };
226 :
227 :   /************************************************************************/
228 :   /* メインプログラム                                                     */
229 :   /************************************************************************/
230 :   void main( void )
231 :   {
232 :       init();                             /* マイコン機能の初期化     */
233 :       set_ccr( 0x00 );                    /* 全体割り込み許可         */
234 :
235 :       music_flag = 1;
236 :
237 :       while( 1 ) {
238 :       }
239 :   }
240 :
241 :   /************************************************************************/
242 :   /* H8/3687F 内蔵周辺機能  初期化                                        */
243 :   /************************************************************************/
244 :   void init( void )
245 :   {
246 :       /* I/Oポートの入出力設定 */
247 :       PCR1 = 0xff;
248 :       PCR2 = 0xfd;                        /* 通信ビットP22:TxD P21:RxD*/
249 :       PCR3 = 0xf0;                        /* 基板上のディップスイッチ */
250 :       PCR5 = 0xff;
251 :       PCR6 = 0xff;                        /* P66:ブザー               */
252 :       PCR7 = 0xff;
253 :       PCR8 = 0xff;
254 :       /* ポートBは、入力専用なので入出力設定はありません。           */
255 :       /* また、A/D変換器のADCSRによって指定された1ビットだけは、     */
256 :       /* 入力ポートとしては使えません。                               */
257 :
258 :       /* タイマB1 1msごとの割り込み設定 */
259 :       TMB1 = 0x84;                        /* 入力クロックの設定等     */
260 :       TLB1 = 26;                          /* カウンタ初期値設定       */
261 :       IENR2 = 0x20;                       /* 割り込み要求許可         */
262 :
263 :       /* タイマZ ch1 */
264 :       TCR_1  = 0x23;                      /* カウンタクロック設定     */
```

```
265 :      TPMR   = 0x20;                      /* PWMモードの設定          */
266 :      TOCR   = 0x00;                      /* 初期出力値設定           */
267 :      POCR_1 = 0x00;                      /* 出力レベル設定           */
268 :      GRA_1  = 1;                         /* 周期の設定               */
269 :      GRC_1  = 1;                         /* デューティ比設定         */
270 :      TOER   = 0xb1;                      /* PWM出力の許可            */
271 :  }
272 :
273 :  /************************************************************************/
274 :  /* 音を鳴らす                                                           */
275 :  /* 引数    トーンのPWM値                                                */
276 :  /* 戻り値 なし                                                          */
277 :  /************************************************************************/
278 :  void note( unsigned int tone )
279 :  {
280 :      if( tone ) {
281 :          TSTR  &= 0xfd;                  /* PWM停止                  */
282 :          TCNT_1 = tone - 1;
283 :          GRA_1  = tone;                  /* トーン設定               */
284 :          GRC_1  = tone / 2;              /* デューティ比は５０％     */
285 :          TSTR  |= 0x02;                  /* PWM開始                  */
286 :      } else {
287 :          TSTR  &= 0xfd;                  /* PWM停止                  */
288 :          TCNT_0 = 0;
289 :          GRA_1  = 1;
290 :          GRC_1  = 1;
291 :          TSTR  |= 0x02;                  /* PWM開始                  */
292 :      }
293 :  }
294 :
295 :  /************************************************************************/
296 :  /* タイマB1 割り込み処理                                                */
297 :  /************************************************************************/
298 :  #pragma interrupt( interrupt_timerB1 )
299 :  void interrupt_timerB1( void )
300 :  {
301 :      IRR2 &= 0xdf;                       /* フラグクリア             */
302 :      cnt0++;
303 :
304 :      if( music_flag ) {
305 :          /* 音階処理 */
306 :          if( cnt0 >= 15000L*music_data[music_dim][1]/TEMP0 ) {
307 :              /* 次の音階をならす */
308 :              cnt0 = 0;
309 :              music_dim++;
310 :              if( music_data[music_dim][0] == -1 ) {
311 :                  /* -1なら終了 */
312 :                  note( 0 );
313 :                  music_flag = 0;
314 :              } else {
315 :                  /* -1でないなら次の音階セット */
316 :                  note( music_data[music_dim][0] );
317 :              }
318 :          }
319 :      }
320 :  }
321 :
322 :  /************************************************************************/
323 :  /* End of file                                                          */
324 :  /************************************************************************/
```

20.5 プログラムの解説

プロジェクト「beep_haru」と同じです。

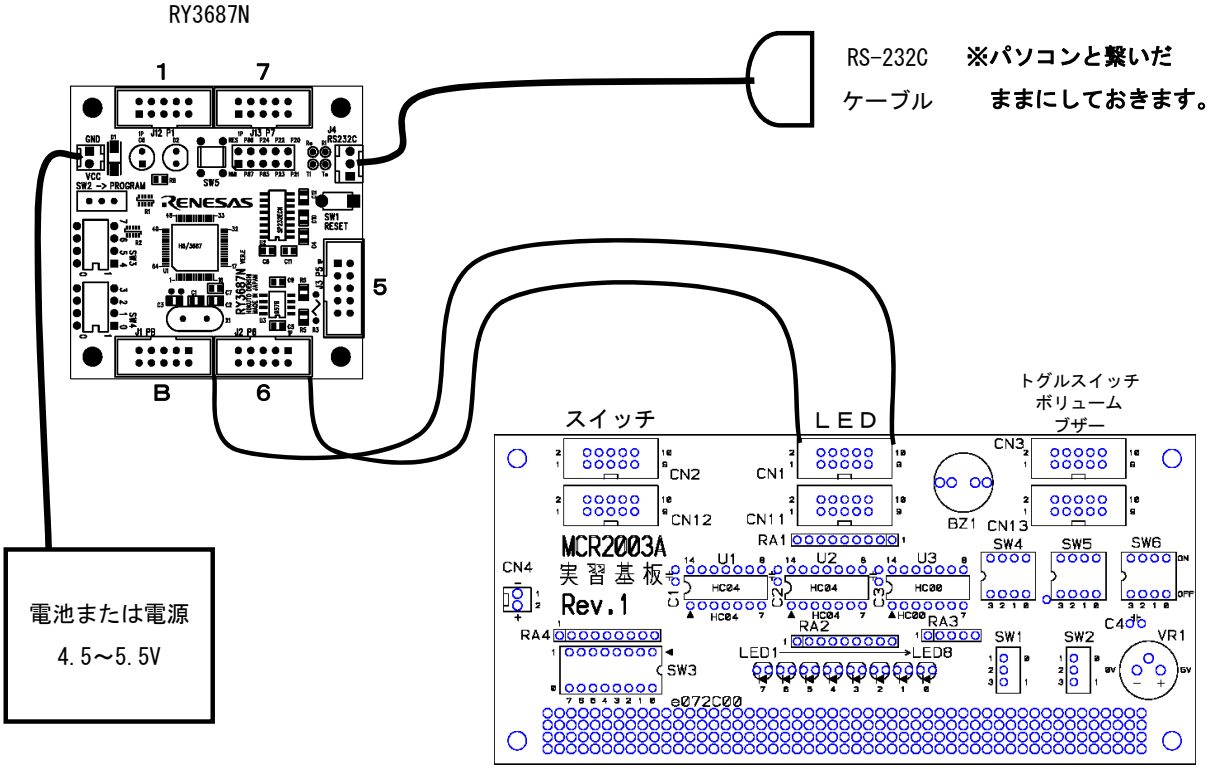
21. プロジェクト「sio」 パソコンから数値を入力してLEDに出力する

21.1 概要

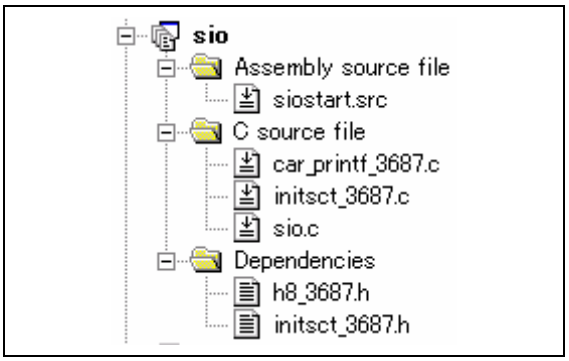
パソコンから入力した数値を LED へ出力します。

21.2 接続

・CPU ボードのポート 6 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。



21.3 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_3687.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。H8/3687F の内蔵周辺機能の初期化も行います。
4	h8_3687.h	H8/3687F 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_3687.h	initsct_3687.c のヘッダファイルです。
6	car_printf_3687.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_3687.c ファイル追加します。**

21.4 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 :  /*                          2007.04 ジャパンマイコンカーラリー実行委員会 */
 4 :  /*********************************************************************/
 5 :  /*
 6 :  入力：SCI3(パソコンのキーボードで入力したデータ)
 7 :        ※パソコンはTeraTermProなどの通信ソフトを使用します
 8 :  出力：P67-P60(LED等)
 9 :
10 :  TeraTermProなどの通信ソフトを通して、キーボードから入力した数値を
11 :  ポート6に接続したLEDなどへ出力します。
12 :  C言語でおなじみのprintf文、scanf文を使用した演習です。
13 :  */
14 :
15 :  /*====================================*/
16 :  /* インクルード                       */
17 :  /*====================================*/
18 :  #include    <no_float.h>              /* stdioの簡略化 最初に置く*/
19 :  #include    <stdio.h>
20 :  #include    <machine.h>
21 :  #include    "h8_3687.h"
22 :  #include    "initsct_3687.h"
23 :
24 :  /*====================================*/
25 :  /* プロトタイプ宣言                   */
26 :  /*====================================*/
27 :  void init( void );
28 :
29 :  /*====================================*/
30 :  /* グローバル変数の宣言               */
31 :  /*====================================*/
32 :
33 :  /*********************************************************************/
34 :  /* メインプログラム                                                  */
35 :  /*********************************************************************/
36 :  void main( void )
37 :  {
38 :      int     i, ret;
39 :
40 :      init();                           /* マイコン機能の初期化     */
41 :      init_sci3( 0x00, 47 );            /* SCI3初期化               */
42 :      set_ccr( 0x00 );                  /* 全体割り込み許可         */
43 :
44 :      printf( "Hello World!¥n" );
45 :      printf( "compile date : %s¥n" , COMPILE_DATE );
46 :      printf( "compile time : %s¥n" , COMPILE_TIME );
47 :      while( 1 ) {
48 :          printf( "Input data : " );
49 :          ret = scanf( "%d", &i );
50 :          if( ret == 1 ) {
51 :              printf( "Get data : %d¥n", i );
52 :              PDR6 = i;
53 :          } else {
54 :              printf( "Data Error!!¥n" );
55 :              scanf( "%*[^¥n]" );
56 :          }
57 :      }
58 :  }
59 :
60 :  /*********************************************************************/
61 :  /* H8/3687F 内蔵周辺機能  初期化                                     */
62 :  /*********************************************************************/
63 :  void init( void )
64 :  {
65 :      /* I/Oポートの入出力設定 */
66 :      PCR1 = 0xff;
67 :      PCR2 = 0xfd;                      /* 通信ビットP22:TxD P21:RxD*/
68 :      PCR3 = 0xf0;                      /* 基板上のディップスイッチ */
69 :      PCR5 = 0xff;
70 :      PCR6 = 0xff;                      /* LED基板                  */
71 :      PCR7 = 0xff;
72 :      PCR8 = 0xff;
73 :      /* ポートBは、入力専用なので入出力設定はありません。        */
74 :      /* また、A/D変換器のADCSRによって指定された1ビットだけは、  */
75 :      /* 入力ポートとしては使えません。                           */
76 :  }
77 :
78 :  /*********************************************************************/
79 :  /* End of file                                                       */
80 :  /*********************************************************************/
```

21.5 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 :  ; 外部参照
3 :  ;=====================================
4 :          .IMPORT _main
5 :          .IMPORT _INITSCT
6 :          .IMPORT _intSCI3
7 :
8 :  ;=====================================
9 :  ; ベクタセクション
10 :  ;=====================================
11 :         .SECTION V
12 :         .DATA.W RESET_START             ;  0 リセット、WDT
13 :         .DATA.W H'FFFF                  ;  1 システム予約
14 :         .DATA.W H'FFFF                  ;  2 システム予約
15 :         .DATA.W H'FFFF                  ;  3 システム予約
16 :         .DATA.W H'FFFF                  ;  4 システム予約
17 :         .DATA.W H'FFFF                  ;  5 システム予約
18 :         .DATA.W H'FFFF                  ;  6 システム予約
19 :         .DATA.W H'FFFF                  ;  7 外部割り込み端子
20 :         .DATA.W H'FFFF                  ;  8 トラップ命令#0
21 :         .DATA.W H'FFFF                  ;  9 トラップ命令#1
22 :         .DATA.W H'FFFF                  ; 10 トラップ命令#2
23 :         .DATA.W H'FFFF                  ; 11 トラップ命令#3
24 :         .DATA.W H'FFFF                  ; 12 ブレーク条件成立
25 :         .DATA.W H'FFFF                  ; 13 スリープ命令の実行による直接遷移
26 :         .DATA.W H'FFFF                  ; 14 IRQ0
27 :         .DATA.W H'FFFF                  ; 15 IRQ1
28 :         .DATA.W H'FFFF                  ; 16 IRQ2
29 :         .DATA.W H'FFFF                  ; 17 IRQ3
30 :         .DATA.W H'FFFF                  ; 18 WKP
31 :         .DATA.W H'FFFF                  ; 19 オーバフロー
32 :         .DATA.W H'FFFF                  ; 20 システム予約
33 :         .DATA.W H'FFFF                  ; 21
34 :         .DATA.W H'FFFF                  ; 22 タイマV
35 :         .DATA.W _intSCI3                ; 23 SCI3
36 :         .DATA.W H'FFFF                  ; 24 IIC2
37 :         .DATA.W H'FFFF                  ; 25 AD変換器
38 :         .DATA.W H'FFFF                  ; 26 タイマZ A0-D0
39 :         .DATA.W H'FFFF                  ; 27 タイマZ A1-D1
40 :         .DATA.W H'FFFF                  ; 28
41 :         .DATA.W H'FFFF                  ; 29 タイマB1
42 :         .DATA.W H'FFFF                  ; 30
43 :         .DATA.W H'FFFF                  ; 31
44 :         .DATA.W H'FFFF                  ; 32 SCI3_2
45 :
46 :  ;=====================================
47 :  ; スタートアッププログラム
48 :  ;=====================================
49 :         .SECTION P
50 :  RESET_START:
51 :         MOV.W   #H'FF80,R7              ; スタックポインタの設定
52 :         JSR     @_INITSCT               ; セクションD,R,Bの設定
53 :         JSR     @_main                  ; C言語のmain()関数へジャンプ
54 :  LOOP:
55 :         BRA     LOOP
56 :
57 :         .END
```

21.6 パソコンとマイコンの通信

21.6.1 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！！

21.6.2 パソコンとの接続方法

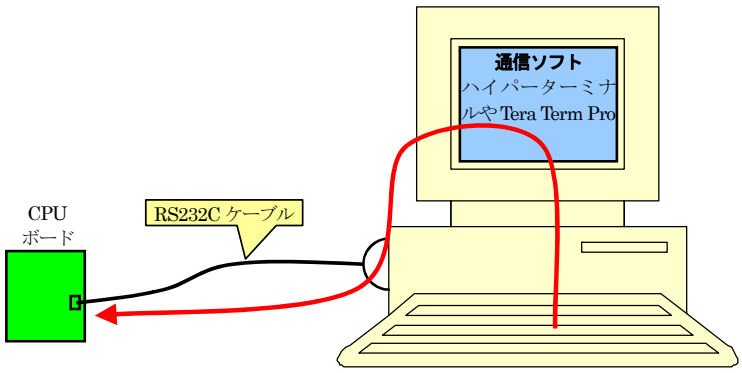
といっても、CPU ボードとディスプレイやキーボードを簡単に接続することはできません。接続する回路や制御プログラムが大変になります。

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

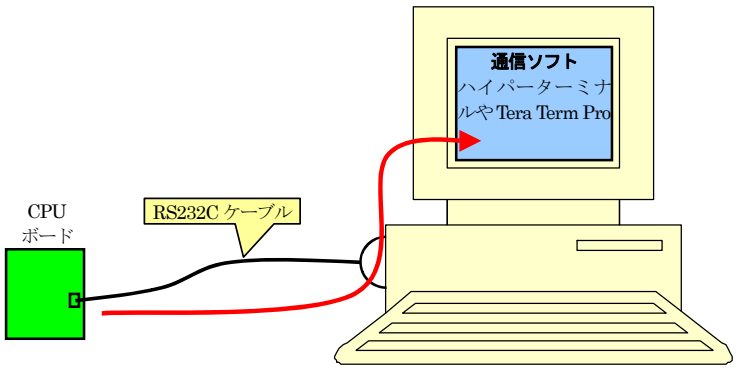
通信ソフトは、キーボードから入力された文字を、RS232C ケーブルを通じて CPU ボードへ送ります(下図)。



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 6 に接続されている LED へデータを出力します。

21.7 プログラムの解説

21.7.1 car_printf_3687.cとは？

プロジェクト「sio」は、「car_printf_3687.c」ファイルを追加しています。これは、

・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_3687.c ファイルを追加します。

21.7.2 car_printf_3687.cのシンボル定義

car_printf_3687.c は、下記のようなシンボル定義をしています。

設定行数	内容
23 行	23 ： #define SEND_BUFF_SIZE 32 /* 送信バッファサイズ */ printf 関数を使用するとき、一度に貯めておける文字数を設定します。 小さいとプログラムの実行速度が遅くなります。大きいと RAM の消費が増えてプログラムが実行できなくなることがあります。問題がなければ、標準値の 32 としておきます。
24 行	24 ： #define RECV_BUFF_SIZE 32 /* 受信バッファサイズ */ scanf 関数を使用するとき、一度に貯めておける文字数を設定します。 小さいと scanf 文で受信できる文字数が少なくなります。大きいと RAM の消費が増えてプログラムが実行できなくなることがあります。問題がなければ、標準値の 32 としておきます。

21.7.3 siostart.srcファイル IMPORT追加とベクタアドレスの変更

car_printf_3687.c ファイルを追加して sio.c プログラム内を改造しただけでは、まだ printf 関数は使えません。src ファイル内も追加、変更します。

```
 1 ： ;=====================================
 2 ： ; 外部参照
 3 ： ;=====================================
 4 ：         .IMPORT _main
 5 ：         .IMPORT _INITSCT
 6 ：         .IMPORT _intSCI3
```

外部参照部分に「**_intSCI3**」を追加します。

```
35 ：         .DATA.W _intSCI3              ; 23 SCI3
```

ベクタ番号 23 のジャンプ先を「_intSCI3」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。

21.7.4 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*/
16 :  #include    <stdio.h>
17 :  #include    <machine.h>
18 :  #include    "h8_3687.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。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**。

21.7.5 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。

ポイントは、プロジェクトにあるすべての 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_printf_3687.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。

```
15 :  /*====================================*/
16 :  /* インクルード                       */
17 :  /*====================================*/
18 :  #include    <no_float.h>              /* stdioの簡略化 最初に置く*/
19 :  #include    <stdio.h>
20 :  #include    <machine.h>
21 :  #include    "h8_3687.h"
22 :  #include    "initsct_3687.h"
```

car_printf_3687.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 :  できます。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できます。
15 :  */
16 :  #include    <stdio.h>
17 :  #include    <machine.h>
18 :  #include    "h8_3687.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>**」の行を削除します。

21.7.6 sio.cファイル init_sci3 関数の追加

```
 33 :  /*********************************************************************/
 34 :  /* メインプログラム                                                  */
 35 :  /*********************************************************************/
 36 :  void main( void )
 37 :  {
 38 :      int     i, ret;
 39 :
 40 :      init();                         /* マイコン機能の初期化      */
 41 :      init_sci3( 0x00, 47 );          /* SCI3初期化                */
 42 :      set_ccr( 0x00 );                /* 全体割り込み許可          */
```

H8/3687F マイコンの内蔵周辺機能である SCI3 を初期化する関数を実行します。カッコ内の「0x00」と「47」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 に設定します。

```
   init_sci3(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、RY3687N ボードは 14.7456MHz

A の値は 4 種類あるので、4 通り計算します。φ＝14.7456MHz、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＝$14.7456\times10^6\div(32\times9600)-1$　＝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

init_sci3 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→A が 32、128、512 のとき可能、2048 は不可
(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=47
が設定値となります。

A の値を直接 init_sci3 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_sci3( 0x00, 47 );          /* SCI3 初期化          */
                   ↑   ↑
                  SMR  BRR
```

●クリスタル値が 16MHz の場合

下記にクリスタル値が 16MHz の場合の計算例を取り上げます。

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。

21.7.7 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。

```
 44 :      printf( "Hello World!¥n" );
 45 :      printf( "compile date : %s¥n" , COMPILE_DATE );
 46 :      printf( "compile time : %s¥n" , COMPILE_TIME );
 47 :      while( 1 ) {
 48 :          printf( "Input data : " );
 49 :          ret = scanf( "%d", &i );
 50 :          if( ret == 1 ) {
 51 :              printf( "Get data : %d¥n", i );
 52 :              PDR6 = i;
 53 :          } else {
 54 :              printf( "Data Error!!¥n" );
 55 :              scanf( "%*[^¥n]" );
 56 :          }
 57 :      }
```

44 行で最初に「Hello World!」と表示します。

45 行から 46 行で、コンパイルした日付と時間を表示します。これはあくまでパソコンの時間です。パソコンの時間がずれていれば、実際の時間とは異なります。

48 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。

49 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。

50 行で scanf 関数の戻り値をチェックします。

51、52 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 6 へ出力します。

54、55 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。

21.7.8 コンパイルした日付と時間の出力

今回のプロジェクトでは、コンパイルした日付と時間を表示します。

```
 45 :      printf( "compile date : %s¥n" , COMPILE_DATE );
 46 :      printf( "compile time : %s¥n" , COMPILE_TIME );
```

コンパイルした日付は、「COMPILE_DATE」に記述されています。

コンパイルした時間は、「COMPILE_TIME」に記述されています。

これらを使用するには、「initsct_3687.h」をインクルードしておく必要があります（下記）。インクルードしていないと、エラーとなります。

```
 22 :  #include    "initsct_3687.h"
```

21.7.9 受信バッファのクリア

プログラムの 55 行目で、バッファのクリアをするために代入抑止文字を使っています。

```
            scanf( "%*[^¥n]" );
```

scanf 内の文字を分解すると下記のようになります。

```
 %   *   [^ ¥n ]
 ①   ②   ③   ④   ⑤
```

① 変換指定文字の開始です。
② 読み捨てる意味です。以後の書式を読み捨てます。
③ ⑤と対で、その中の文字のみを読み飛ばします。
④ 読み飛ばす文字は¥n、すなわち改行コードです。

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、

```
 49 :           ret = scanf( "%d", &i );
```

ここで、「aaa(改行)」を入力したとします(改行＝エンタ)。入力バッファには、

```
aaa¥n
```

と文字が保存されます。改行が入力されると変換が開始されます。変換指定文字「%d」は、10 進数入力です。それ以外のアルファベットなどの文字が入力されてしまうと「10 進入力にもかかわらず数字以外のものが入力されていた場合は、その文字を読み込まずに作業は終了する」に当てはまってしまいます。この場合、エラー終了してしまいます。バッファはクリアされません。ここで再度、数値入力しようと、

```
          ret = scanf( "%d", &i );
```

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。

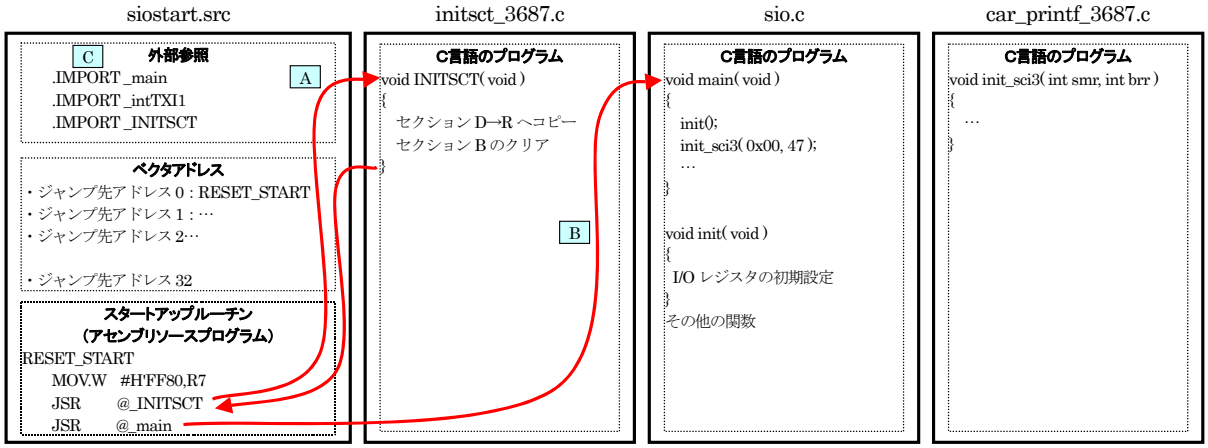
```
 50 :           if( ret == 1 ) {
 51 :               printf( "Get data : %d¥n", i );
 52 :               PDR6 = i;
```

戻り値を格納した ret 変数が 1 なら、printf 関数で値を表示して、ポート6へその値を出力します。

```
53 :          } else {
54 :              printf( "Data Error!!¥n" );
55 :              scanf( "%*[^¥n]" );
56 :          }
```

else 文、すなわち戻り値が1でなくエラーであれば、入力エラーと表示して、バッファをクリアします。**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。**

21.8 まとめ



A…アセンブリソースプログラムから、「initsct_3687.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_printf_3687.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)。

21.9 演習手順

21.9.1 パソコン設定する前準備

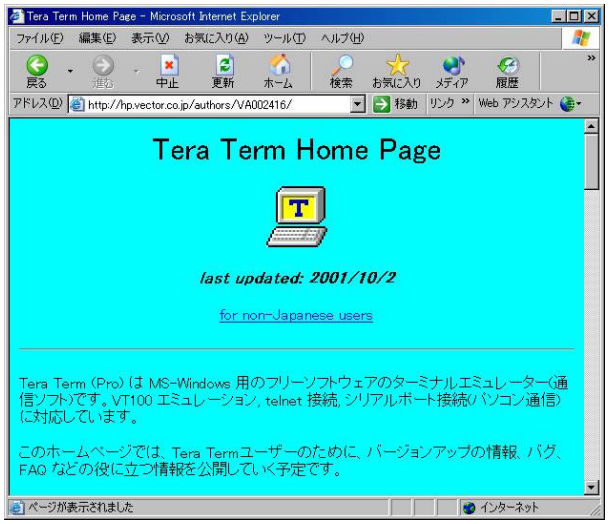
これからパソコンの設定をします。その前に下記作業をあらかじめ済ませておきます。

・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）。

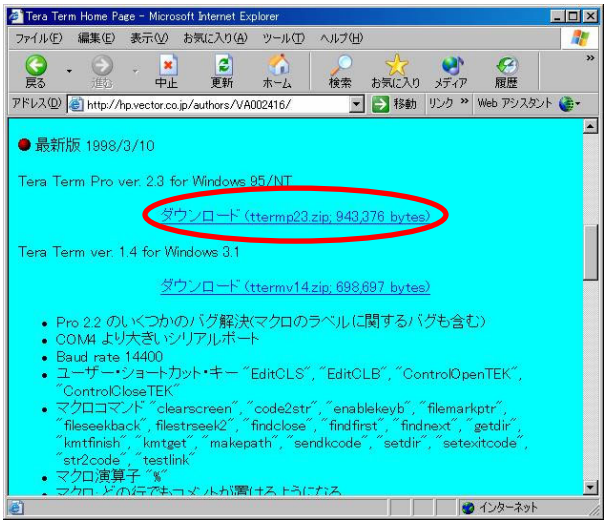
・通信ケーブルは接続したままにしておきます。

21.9.2 TeraTermProのインストール

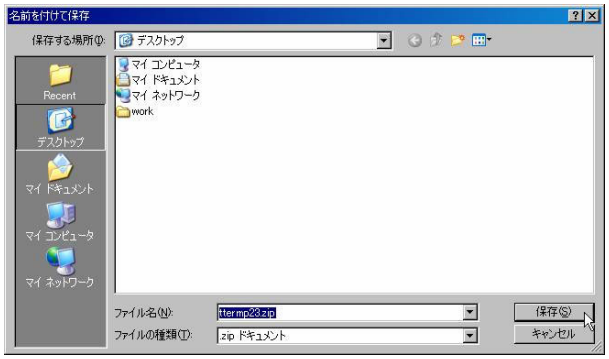
通信ソフトとして、ハイパーターミナルというソフトがあります。ハイパーターミナルは、Windows 標準で入っているためインストール不要で利用できます。しかし、古い Windows では入っていないことがある、通信できないことがある、など不具合が発生しやすいソフトでもあります。そこで、フリーソフトで通信のできる**「Tera Term Pro」**というソフトを使用します。



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。



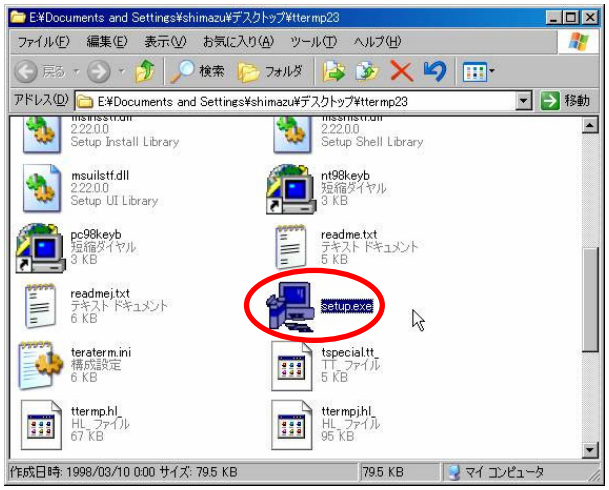
2. 下の方に「ダウンロード (ttermp23.zip; 943,376 bytes)」とあるので、クリックして保存します。



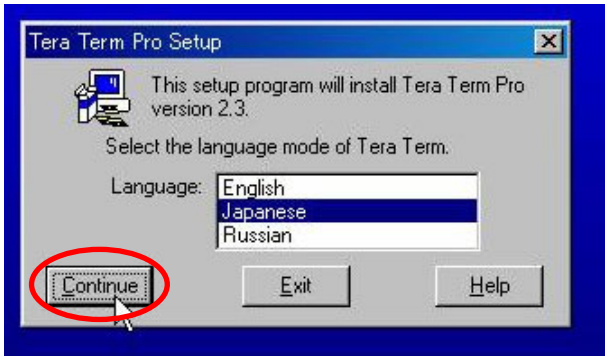
3. 保存は何処でも良いですが、ここではディスクトップに保存します。

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。



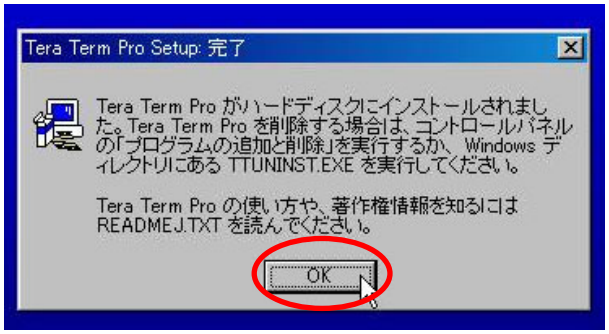
7. 言語の選択です。日本になっていますのでそのまま Continue(続ける)をクリックします。

8. 注意が出ます。Continue(続ける)をクリックします。

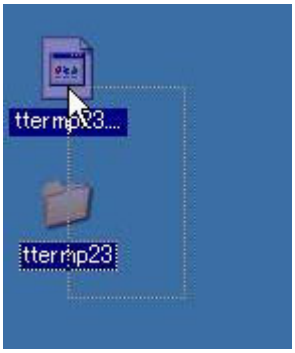
9. キーボードの選択です。Continue(続ける)をクリックします。

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。

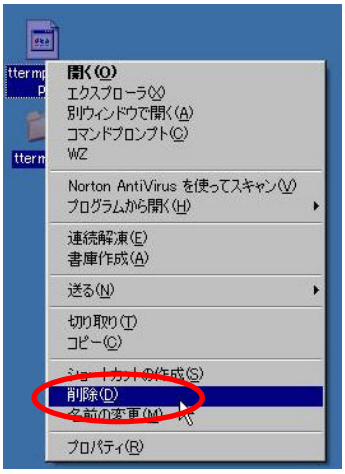
11. メニューが立ち上がります。×をクリックして閉じます。



12. OK をクリックして、インストールを完了します。

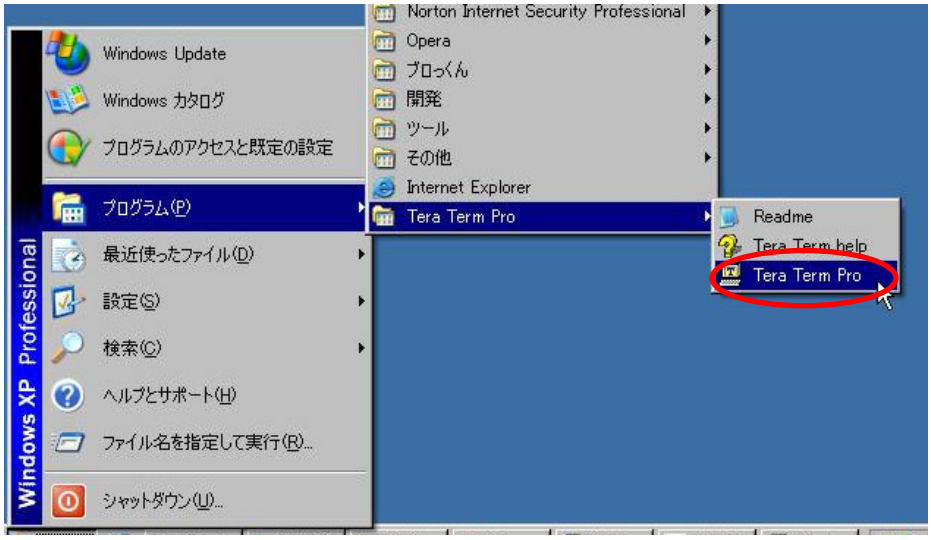


13. インターネットからファイルをダウンロードした場合、ダウンロードしたファイルを削除します。ttermp23.zip と ttermp23 フォルダを選択します。CD からインストールした場合は不要です。

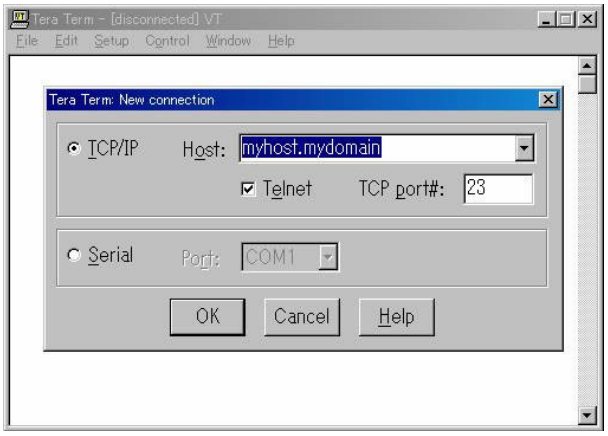


14. どちらかのファイルの上で右クリックして「削除」をクリック、ファイルを削除します。

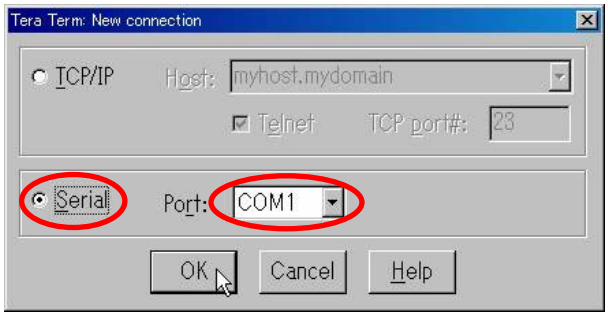
21.9.3 TeraTermProを使用したマイコンーパソコン通信



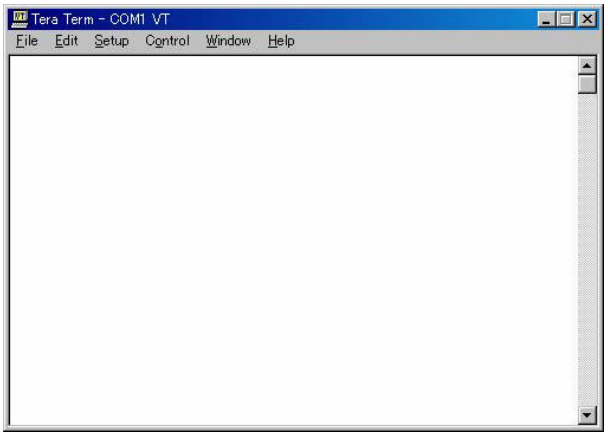
1. 「スタート」→「すべてのプログラム、またはプログラム」→「Tera Term Pro」→「Tera Term Pro」
   で Tera Term Pro が立ち上がります。



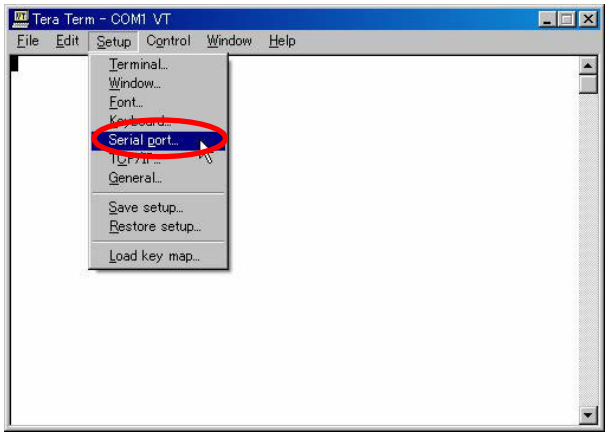
2. 最初にどこと接続するか確認する画面が出てきます。



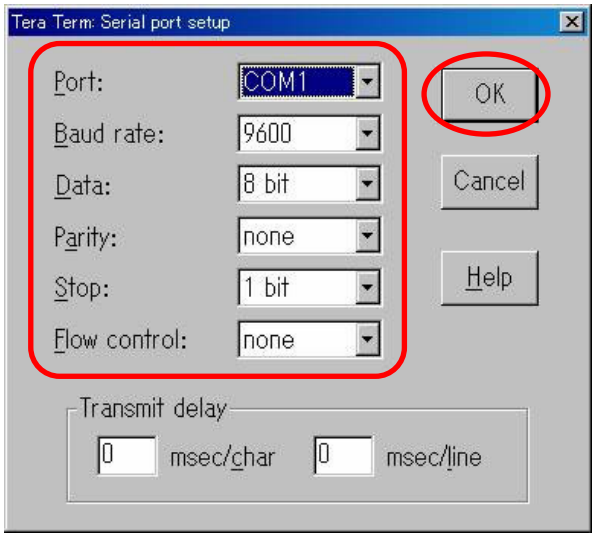
3. 「Serial」を選んで、ポート番号を選びます。選択後、OKをクリックして次へ進みます。



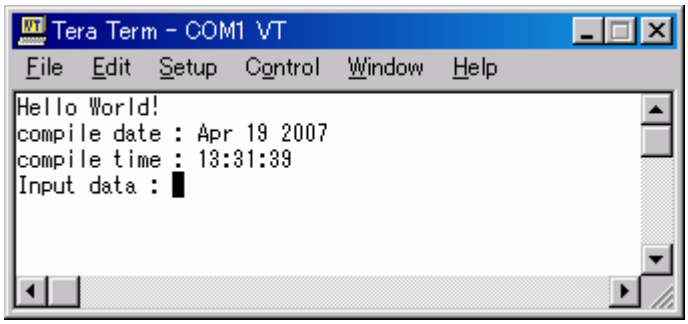
4. 立ち上がりました。
   詳細設定をします。



5. 「Setup→Serial port」を選択します。

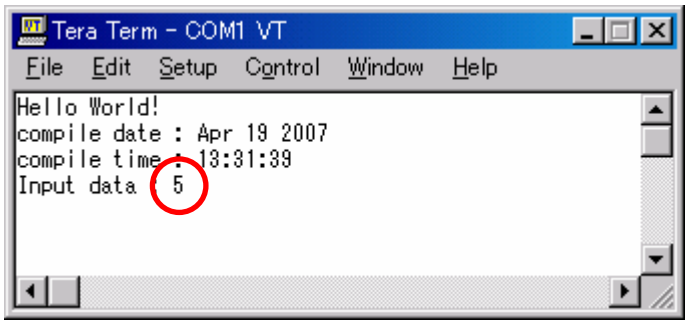


6.通信設定を確認します。画面のように設定して、OK をクリックします。「Port」は、それぞれの通信ポートの番号に合わせてください。



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。

7.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。



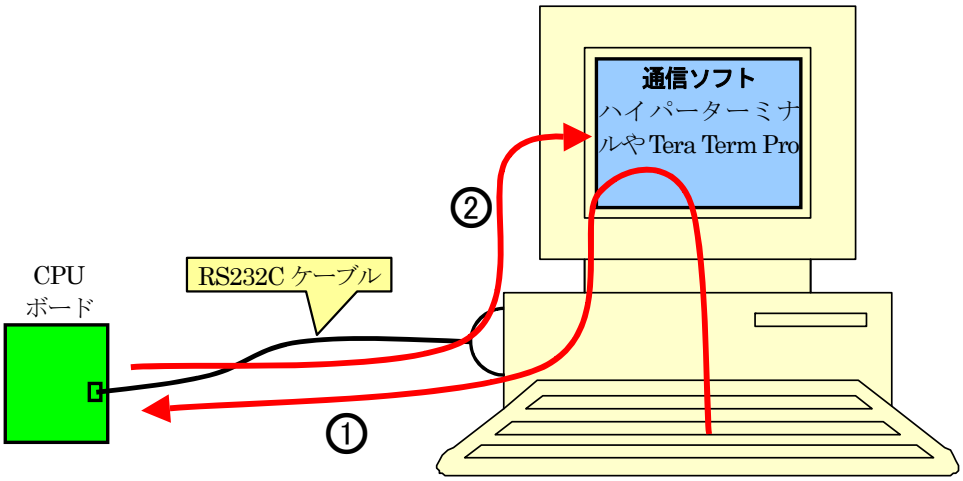
8.キーボードの「5」キーを入力します。TeraTermPro の画面に「5」が表示されました。これは、
キーボード→通信ソフトの画面
という流れで表示されたわけではありません。
「5」というデータは、次の順で CPU ボードへ送られます。

①キーボード→通信ソフト→RS232Cケーブル→CPUボード

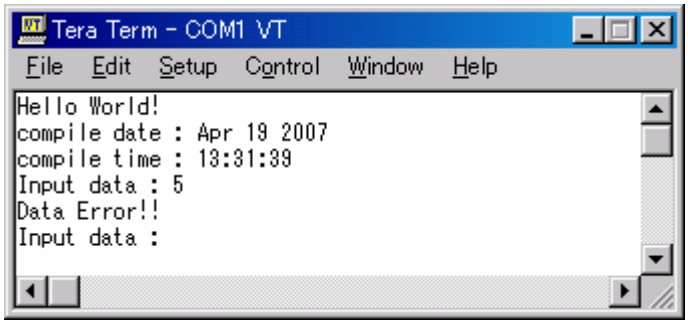
CPU ボードは送られてきたデータをそのままパソコンへ送ります（scanf 関数で実行しています）。

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。



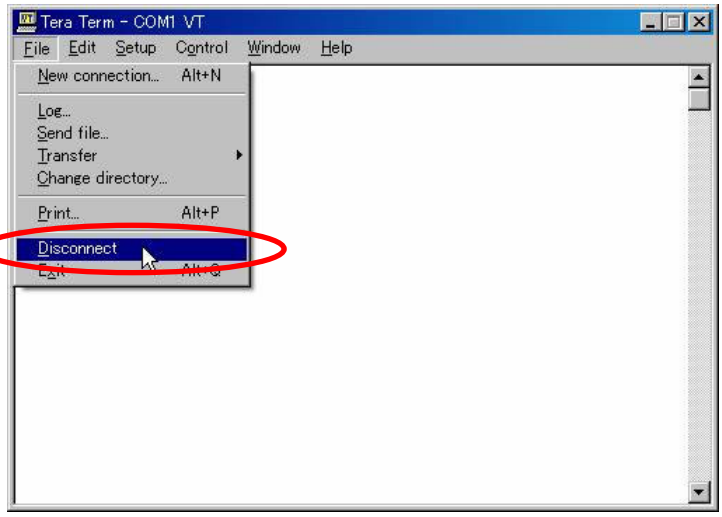
9.「5」の次にエンターを押すと、LED は、「0000 0101」出力となります。H8 側からも「Get data : 5」という文字列が返ってきて、5 が入力されたことが分かります。いろいろな値を入力してどうなるか試してみてください。

10.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。

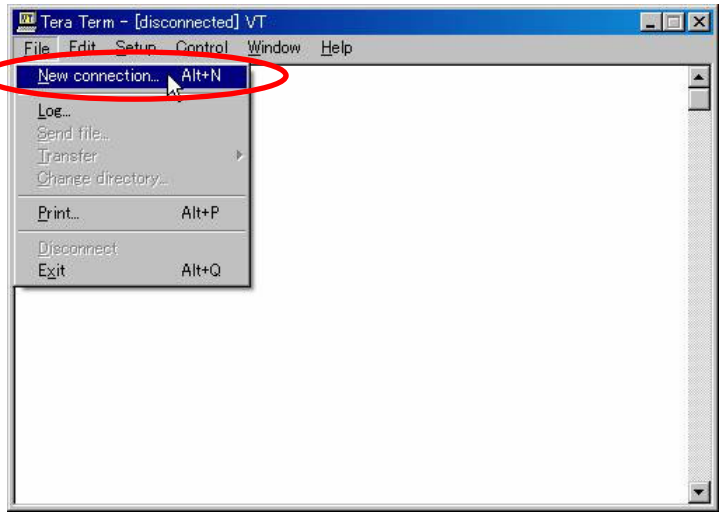
21.9.4 TeraTermProを立ち上げた状態でプログラムの書き込みをするには

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。

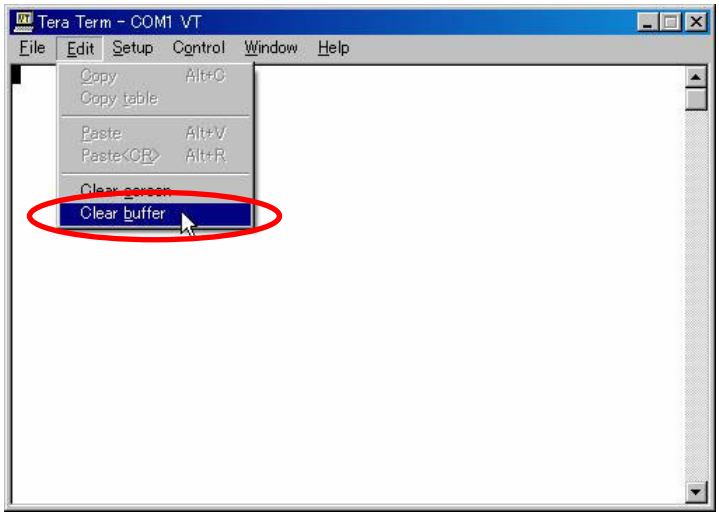


「File→Disconnect」をクリックします。これで TeraTermPro は、通信ポートを空けます。



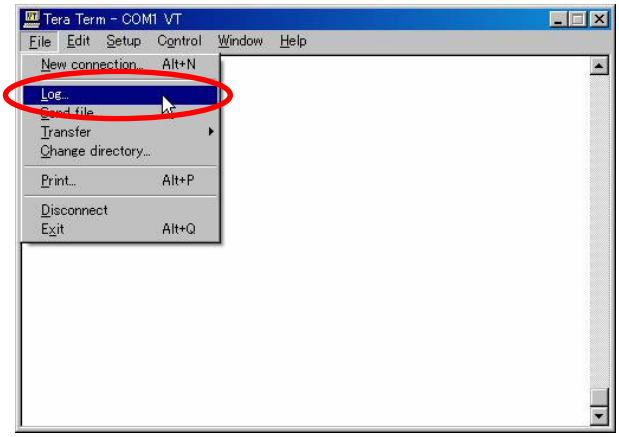
「File→New connection」で再度、接続します。

21.9.5 TeraTermProの画面をクリア

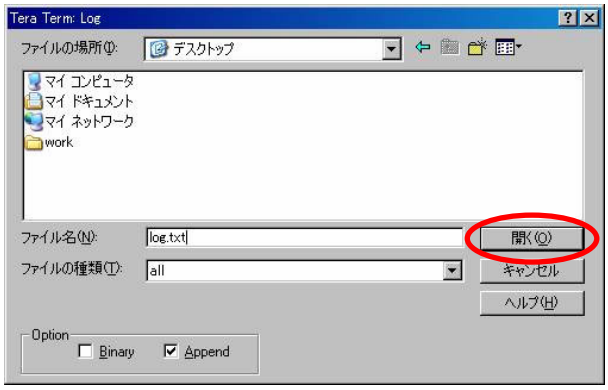


画面をクリアしたい場合は、「Edit→Clear buffer」です。

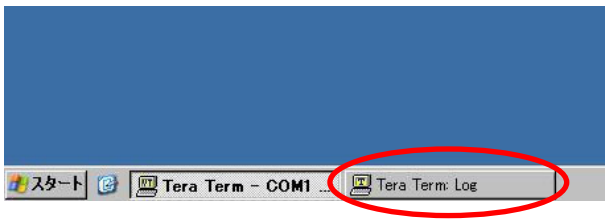
21.9.6 TeraTermProの通信内容を保存



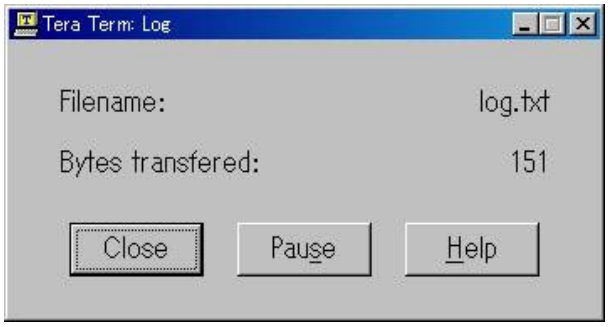
やり取りしているデータをファイルに保存することができます。「File→Log」を選択します。



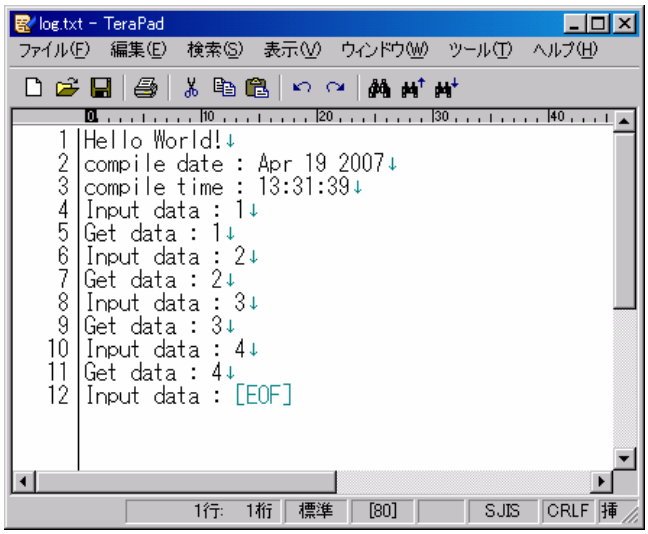
保存ファイル名を入力します。ここでは「log.txt」と入力します。開くをクリックして完了です。



「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。

22. プロジェクト「adsio」 電圧をパソコンへ出力

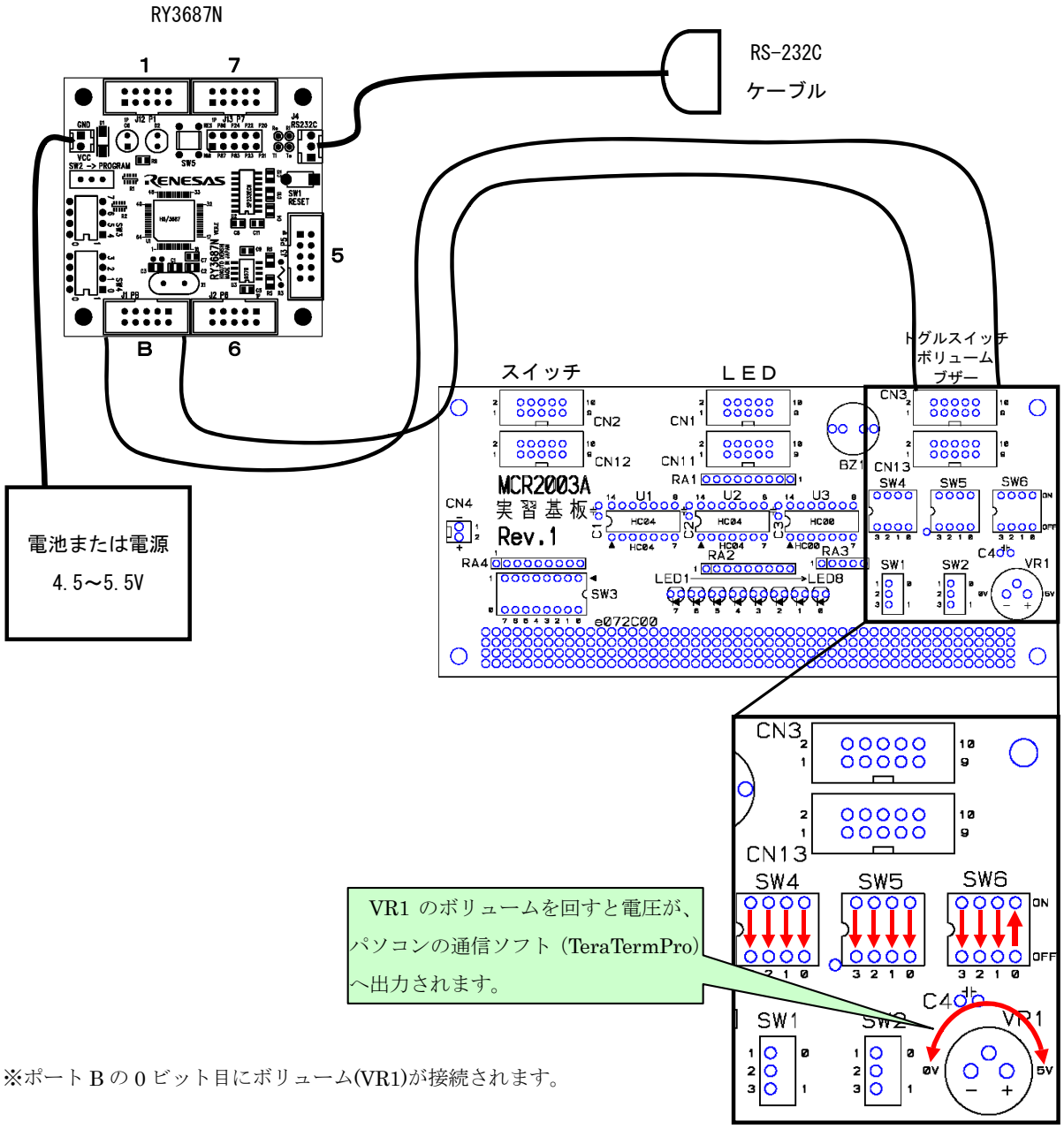
22.1 概要

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 B の bit0・・・アナログ電圧入力
・RS-232C ケーブル・・・パソコンと通信

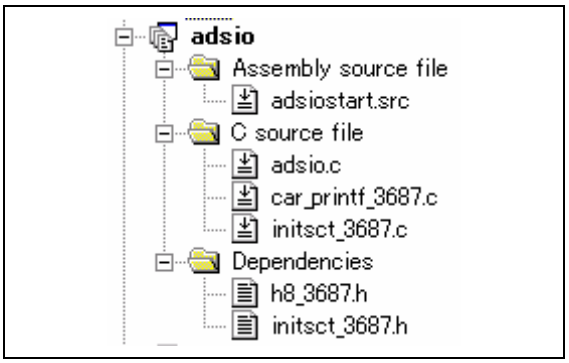
22.2 接続

・CPU ボードのポート B と、実習基板のトグルスイッチ・ボリューム・ブザー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。



※ポート B の 0 ビット目にボリューム(VR1)が接続されます。

22.3 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_3687.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。H8/3687F の内蔵周辺機能の初期化も行います。
4	h8_3687.h	H8/3687F 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_3687.h	initsct_3687.c のヘッダファイルです。
6	car_printf_3687.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「adsio」に限らず、printf、scanf 関数を使うプロジェクトは、car_printf_3687.c ファイルを追加します。

22.4 プログラム「adsio.c」

```
 1 :  /*********************************************************************/
 2 :  /* アナログ変換値をパソコンへ送信(通信機能の利用) "sio.c"             */
 3 :  /*                          2007.04 ジャパンマイコンカーラリー実行委員会 */
 4 :  /*********************************************************************/
 5 :  /*
 6 :  入力：PB0のアナログ電圧
 7 :  出力：通信ポート
 8 :        ※パソコンはTeraTermProなどの通信ソフトを使用します
 9 :
10 :  ポート6のbit0に入力された0～5Vの電圧を、0～1023の値にA/D変換します。
11 :  変換値を0.5秒ごとに0.00～5.00の文字列に変換して、通信ポートへ送ります。
12 :  送られた文字列は、TeraTermProなどの通信ソフトの画面に表示されます。
13 :  */
14 :
15 :  /*====================================*/
16 :  /* インクルード                       */
17 :  /*====================================*/
18 :  #include    <no_float.h>              /* stdioの簡略化 最初に置く*/
19 :  #include    <stdio.h>
20 :  #include    <machine.h>
21 :  #include    "h8_3687.h"
22 :  #include    "initsct_3687.h"
23 :
24 :  /*====================================*/
25 :  /* プロトタイプ宣言                   */
26 :  /*====================================*/
27 :  void init( void );
```

```
 28 :  int get_ad( void );
 29 :
 30 :  /*====================================*/
 31 :  /* グローバル変数の宣言               */
 32 :  /*====================================*/
 33 :  unsigned long   cnt0;                  /* main内で使用            */
 34 :
 35 :  /************************************************************************/
 36 :  /* メインプログラム                                                     */
 37 :  /************************************************************************/
 38 :  void main( void )
 39 :  {
 40 :      int     i;
 41 :
 42 :      /* マイコン機能の初期化 */
 43 :      init();                           /* マイコン機能の初期化     */
 44 :      init_sci3( 0x00, 47 );            /* SCI3初期化               */
 45 :      set_ccr( 0x00 );                  /* 全体割り込み許可         */
 46 :
 47 :      printf( "Voltage Meter¥n");
 48 :      while( 1 ) {
 49 :          if( cnt0 >= 500 ) {
 50 :              cnt0 = 0;
 51 :              i = get_ad();
 52 :              i = (long)500 * i / 1023;
 53 :              printf( "%01d", i/100 );
 54 :              printf( "." );
 55 :              printf( "%02d¥n", i%100 );
 56 :          }
 57 :      }
 58 :  }
 59 :
 60 :  /************************************************************************/
 61 :  /* H8/3687F 内蔵周辺機能  初期化                                        */
 62 :  /************************************************************************/
 63 :  void init( void )
 64 :  {
 65 :      /* I/Oポートの入出力設定 */
 66 :      PCR1 = 0xff;
 67 :      PCR2 = 0xfd;                      /* 通信ビットP22:TxD P21:RxD*/
 68 :      PCR3 = 0xf0;                      /* 基板上のディップスイッチ */
 69 :      PCR5 = 0xff;
 70 :      PCR6 = 0xff;
 71 :      PCR7 = 0xff;
 72 :      PCR8 = 0xff;
 73 :      /* ポートBは、入力専用なので入出力設定はありません。          */
 74 :      /* また、A/D変換器のADCSRによって指定された1ビットだけは、   */
 75 :      /* 入力ポートとしては使えません。                             */
 76 :
 77 :      /* A/Dの初期設定 */
 78 :      ADCSR = 0x00;
 79 :
 80 :      /* タイマB1 1msごとの割り込み設定 */
 81 :      TMB1 = 0x84;                      /* 入力クロックの設定等     */
 82 :      TLB1 = 26;                        /* カウンタ初期値設定       */
 83 :      IENR2 = 0x20;                     /* 割り込み要求許可         */
 84 :  }
 85 :
 86 :  /************************************************************************/
 87 :  /* A/D値読み込み(AN0)                                                   */
 88 :  /* 引数    なし                                                         */
 89 :  /* 戻り値 A/D値 0～1023                                                 */
 90 :  /************************************************************************/
 91 :  int get_ad( void )
 92 :  {
 93 :      int i;
 94 :
 95 :      ADCSR |= 0x20;                    /* ADスタート               */
 96 :      while( !(ADCSR & 0x80) );         /* エンドフラグをチェック   */
 97 :      ADCSR &= 0x7f;                    /* エンドフラグクリア       */
 98 :      i = ADDRA >> 6;                   /* 代入                     */
 99 :
100 :      return i;
101 :  }
102 :
103 :  /************************************************************************/
104 :  /* タイマB1 割り込み処理                                                */
105 :  /************************************************************************/
106 :  #pragma interrupt( interrupt_timerB1 )
107 :  void interrupt_timerB1( void )
108 :  {
109 :      IRR2 &= 0xdf;                     /* フラグクリア             */
110 :      cnt0++;
111 :  }
112 :
113 :  /************************************************************************/
114 :  /* End of file                                                          */
115 :  /************************************************************************/
```

22.5 プログラムの解説

22.5.1 A/D変換の初期設定

```
 77 :      /* A/D の初期設定 */
 78 :      ADCSR = 0x00;
```

AN0(PB0)端子をアナログ入力端子にします。

22.5.2 割り込みの設定

```
 80 :      /* タイマ B1 1ms ごとの割り込み設定 */
 81 :      TMB1 = 0x84;                    /* 入力クロックの設定等      */
 82 :      TLB1 = 26;                      /* カウンタ初期値設定        */
 83 :      IENR2 = 0x20;                   /* 割り込み要求許可          */
```

タイマ B1 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。

22.5.3 get_ad関数

```
 91 :  int get_ad( void )
 92 :  {
 93 :      int i;
 94 :
 95 :      ADCSR |= 0x20;                  /* AD スタート               */
 96 :      while( !(ADCSR & 0x80) );       /* エンドフラグをチェック    */
 97 :      ADCSR &= 0x7f;                  /* エンドフラグクリア        */
 98 :      i = ADDRA >> 6;                 /* 代入                      */
 99 :
100 :      return i;
101 :  }
```

ad.c と同じですが、98 行のシフトを 6 ビットにして、戻り値は 0～1023 にしています。

22.5.4 interrupt_timerB1 関数(割り込みプログラム)

```
106 :  #pragma interrupt( interrupt_timerB1 )
107 :  void interrupt_timerB1( void )
108 :  {
109 :      IRR2 &= 0xdf;                   /* フラグクリア              */
110 :      cnt0++;
111 :  }
```

この関数が、1[ms]ごとに実行されます。cnt0 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。

22.5.5 main関数

```
 38 :  void main( void )
 39 :  {
 40 :      int     i;
 41 :
 42 :      /* マイコン機能の初期化 */
 43 :      init();                             /* マイコン機能の初期化      */
 44 :      init_sci3( 0x00, 47 );              /* SCI3 初期化               */
 45 :      set_ccr( 0x00 );                    /* 全体割り込み許可          */
 46 :
 47 :      printf( "Voltage Meter¥n");
 48 :      while( 1 ) {
 49 :          if( cnt0 >= 500 ) {
 50 :              cnt0 = 0;
 51 :              i = get_ad();               i＝0～1023
 52 :              i = (long)500 * i / 1023;   i＝0～500 に変換
 53 :              printf( "%01d", i/100 );    1 桁目出力
 54 :              printf( "." );
 55 :              printf( "%02d¥n", i%100 );  2～3 桁目出力
 56 :          }
 57 :      }
 58 :  }
```

43 行で、内蔵周辺機能の初期化、44 行は通信の設定です。割り込みを使いますので、45 行で全体の割り込みを許可しています。

49 行で、500 ミリ秒たったかチェックし、たったなら 50 行以降を実行します。50 行で cnt0 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。

51 行で、A/D 値を取得します。0～1023 です。

52 行で、0～1023 の値を 0～500 に変換します。i の最大は 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。

53 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。

54 行で「.」(ピリオド)をパソコンへ出力します。

55 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。

```
 49 :          if( cnt0 >= 500 ) {
```

49 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 cnt0 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。

23. 付録

23.1 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。

23.2 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)

23.3 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

23.4 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）

23.5 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。

23.6 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。

24. 参考文献

・(株)ルネサス テクノロジ
　H8/3687 シリーズ ハードウェアマニュアル 第 3 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、2007 年 9 月現在の情報です。