メカトロニクスラボ　マイコン編

東京工業大学 工学部 制御システム工学科
Rev. 2.06, Dec. 2014
大学院理工学研究科 機械制御システム専攻 田中正行
mtanaka@ctrl.titech.ac.jp
大学院理工学研究科 機械制御システム専攻 鳥居秋彦
torii@ctrl.titech.ac.jp

目次

0　導入

0.1　石川台 3 号館 310 号室について

ディスプレイとマウス，キーボードは，机の中に格納されており，上に引き出して利用する．

注意事項：

- ディスプレイとマウス，キーボードは出し入れはゆっくり行うこと．
- マウスとキーボードのケーブルを切断しないように，特にしまうときは注意すること．
- アカウントは [ctrl***]．*には自分の 3 桁の自分のアカウント番号が入る．パスワードはスタッフの指示に従うこと．
- 計算機を再起動すると，**全てのデータが失われる**ので注意すること．
- データを保存する際は，ファイルサーバに保存すること．ファイルサーバは「マイネットワーク」からアクセスできる．
- 他人のアカウントのファイルサーバにはアクセスしないこと．
- ファイルサーバをそんなに信用しないこと．

0.2　マイコンについて

「マイコン」とはマイクロコンピュータ (Microcomputer) の略で，直訳すれば「小さな計算機」となる．マイコンとは，何かという厳密な定義があるわけではない．しかし，マイコンといえば，積分などの数値計算を主にに行う計算機ではなく，何かを制御する用途に使われることが多い．そこで，マイコンとは，何かを制御するために使われる比較的小さいな計算機と考えておけば良い．実際，マイコンはマイクロコントローラ (Microcontroller) の略と考えられていることもある．例えば，DC モータの回転数を一定に保つように制御することを考える．この場合，DC モータの回転数を測定 (入力) し，測定した回転数に応じて DC モータへ入力する電流を調節 (出力) する必要がある．そのため，マイコンには一般に入出力ポートが備えられており，制御に便利な周辺機能も備えられているものが多い．

ルネサス エレクトロニクス社製 H8 シリーズ，ATMEL 社製 AVR シリーズ，マイクロチップ・テクノロジー社製 PIC シリーズなど，数多くのメーカから，様々な種類のマイコンが販売されている．本実習では，ルネサステクノロジ社製 H8 シリーズ H8/36064 というマイコンを利用して実習を行う．H8/36064 は，CPU，メモリ，タイマ，A/D 変換器などがワンチップになっている便利なマイコンである．

0.3　マイコンボードについて

当然ではあるが，マイコン単体では，動作できない．マイコンに電源や正確なクロックを供給する必要がある．また，パソコンと通信するためには，シリアル通信の接続が必要である．これら，マイコンに必要な電気回路が搭載されている基板をマイコンボードと呼ぶ．マザーボードと呼ぶこともある．

本実習では，VStone 社製 WRC-003LV というマイコンボードを利用する．このマイコンボードに，ルネサステクノロジ社製 H8 シリーズ H8/36064 が搭載されている．このマイコンボードに搭載されている主なものを以下にまとめる．

- CPU(ルネサステクノロジ社製 H8 シリーズ H8/36064)
- モータドライバ
- USB-シリアル変換
- 水晶発振子 (動作周波数 12.0 [MHz])
- LED 2 個
- ブザー
- 押しボタンスイッチ
- USB コネクタ (mini-b 形式)

0.4　開発環境について

マイコン用のプログラムは，C 言語またはアセンブラ言語で，開発されることが多い．本実習では，C 言語を利用して，プログラムを開発することにする．

ルネサステクノロジ社製が提供している HEW(High-performance Embedded Workshop) を利用して，プログラムの開発を行う．HEW は，H8 シリーズのプログラム開発を目的とした統合開発環境である．統合開発環境とは，テキストエディタ，コンパイラなどを 1 つにまとめた開発環境である．

0.5　ソフトウェアについて

本実習では，主に以下の 4 つのソフトウェアを利用して，マイコンのプログラム開発を行う．

HEW 統合開発環境

VS-WRC003LV シリアルコンバータ USB の動作モードを HID(Human Interface Device) とシリアルポートに変更

Vstone H8Writer マイコンにプログラムを ROM 領域に書き込むソフトウェア

HTERM マイコンとパソコンの通信を行うソフトウェア

モニタプログラム マイコン上で動作し，プログラムを RAM 領域にロードするソフトウェア

0.6 モニタプログラムについて

モニタプログラムとは，マイコンの RAM 領域にプログラムをロードし，実行させるソフトウェアである．モニタプログラムは，プログラムのロード・実行のみならず，マイコンのレジスタの状態などを確認することができる．

本実習では，このモニタプログラムを利用して，実習を進める．

0.7 プロジェクト

プロジェクトとは，1 つのプログラムを開発するための 1 つの単位である．マイコンのプログラムをコンパイルし，実行可能にするためには，C 言語のソースファイルだけではなく，様々なファイルを必要とする．

そのため，コンパイルに必要なファイルをひとまとめにして，プロジェクトという単位で管理している．実際には，プロジェクトはフォルダに対応している．

本実習では，サンプルプログラムを含むサンプルプロジェクトが提供される．そのプロジェクト内の,「MonitorSample.c」と「MonitorIntprg.c」を編集し，コンパイルすることで，実習をすすめる．

プロジェクトを生成するためには，各種設定を行う必要があるが，その手間を省略するためである．

0.8 資料について

「T:/マイコン」から，各種ドキュメントが確認できる．特に「T:/マイコン/doc/index.html」からは，以下のドキュメントを確認できる．

1. マイコンボード VS-WRC004 テクニカルマニュアル
2. VS-WRC003LV シリアルコンバータ 取扱説明書
3. マイコンボード VS-WRC003LV 回路図
4. H8Writer 取扱説明書
5. H8 36064 グループハードウェアマニュアル
6. H8 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.7.00 ユーザーズマニュアル
7. H8 High-performance Embedded Workshop V.4.09 ユーザーズマニュアル
8. モニタプログラム readme.htm
9. モニタプログラム monitor.htm (使い方)

この中で,「H8 36064 グループハードウェアマニュアル」が非常に重要である．

また，参考と文献を以下にあげる．

- H8 マイコンによる組込みプログラミング入門，ヴイストン株式会社，オーム社，2009．[図書館]
- H8/Tiny マイコン完璧マニュアル，島田義人，CQ 出版社，2005．[図書館]
- H8 マイコンで学ぶ組み込み I/O 制御演習，新海吉幸，電波新聞社，2007．[図書館]
- H8 ビギナーズガイド，白土義男，東京電機大学出版，2000．[図書館]
- C 言語による組込み制御入門講座，大須賀威彦，電波新聞社，2006．[図書館]
- H8 マイコン完全マニュアル，藤沢幸穂，オーム社，2000．

なお，[図書館] マークのあるものは，大学図書館に所蔵されている．

0.9 プログラム開発の流れ

以下にマイコンのプログラムを開発する上での大きな流れを示す．

1. プログラム開発の目的を決める．
2. 利用する電気機器の使い方 (仕様) を確認する．このとき，それぞれの電気機器のデータシートを利用する．
3. マイコンと利用する電気機器との接続関係を確認する．このとき,「マイコンボード VS-WRC003LV 回路図」(vs-wrc003lv_20100806_1807.pdf) を利用する．
4. マイコンと電気機器を利用して，どのように目的を達成するか考える．
5. マイコンにおいて，設定に必要なレジスタを確認する．このとき,「H8 36064 グループハードウェアマニュアル」(rjj09b0049_h836064.pdf) を利用する．
6. 必要なレジスタの変数名を見つける．“iodefine.h” ファイルを利用する．
7. 見つけた変数名を使って，必要な設定を行う．

1　モニタプログラム

1.1　目標

1-1. モニタプログラムを理解する．
1-2. プログラムを作成し，モニタプログラムにロードし，実行させる．
1-3. printf を使えるようにする．
1-4. メモリを理解する．

1.2　HID(Human Interface Device) とシリアルポート

本実習で利用するマイコンの USB(Universal Serial Bus) は，HID(Human Interface Device) モードとシリアルポートモードで動作させることが可能である．本実習では，シリアルポートモードで動作させる．なお，マイコンはデフォルトでは，HID モードになっているため,「VS-WRC003LV シリアルコンバータ」というソフトウェアを用いて，シリアルポートモードに変更する必要がある．

シリアルポートモードに変更する手順:

1. USB ケーブルで PC とマイコンを接続する
2. 「VS-WRC003LV シリアルコンバータ」を起動する
3. 図 1.1 に示すように，COM ポート番号を空欄にして,「シリアルポート変更」をクリックし，動作完了まで待つ．

1.3　2 種類の実行方法

自作したプログラムをマイコンで実行する方法は，大きく以下の二つの方法がある．

モニタプログラム利用

モニタプログラムを ROM 領域へ書き込み，自作プログラムを RAM 領域へロードし，実行する．

ROM 領域へ書き込み

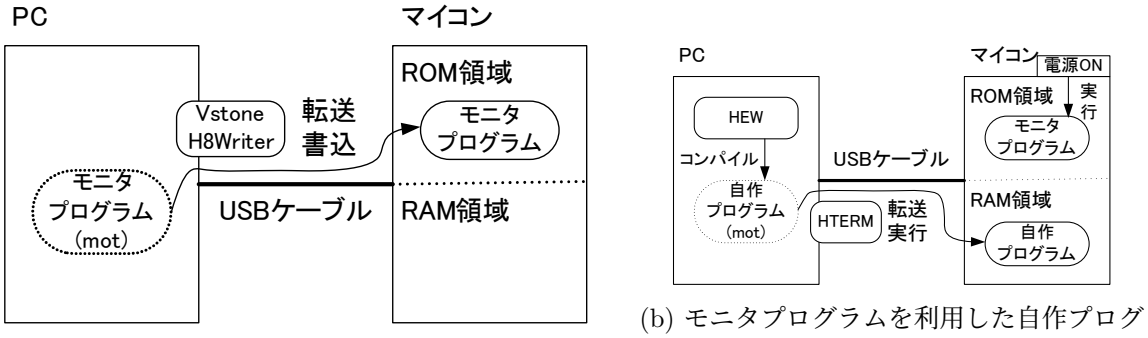
図. 1.1: VS-WRC003LV シリアルコンバータ

Table 1.1: モニタプログラム利用と ROM 領域へ書き込みの比較

	モニタプログラム利用	ROM 領域へ書き込み
書き込み回数	(○) 無制限	(×)1,000 回程度
printf	(○) 利用可能	(×) 利用不可能
デバッグ	(○) デバッグ可能	(×) 専用装置が必要
プログラムサイズ	(×) 小さいサイズのみ	(○) 大きくても大丈夫
電源を切ると	(×) 再ロードが必要	(○) 再ロードは不必要

自作プログラムを，ROM 領域へ書き込み，実行する．

図 1.2 にモニタプログラム利用する場合の概要を，図 1.3 に ROM 領域へ書き込む場合の概要を，それぞれ示す．また，表 1.1 にモニタプログラム利用と ROM 領域へ書き込みの比較を示す．実習では，書込回数が無制限であること，printf が利用可能であることから，モニタプログラムを利用する場合を主に扱う．



(a) モニタプログラムの書き込み（一度だけ）

(b) モニタプログラムを利用した自作プログラムのロードと実行

図. 1.2: モニタプログラムを利用し，自作プログラムを RAM 領域にロード，実行

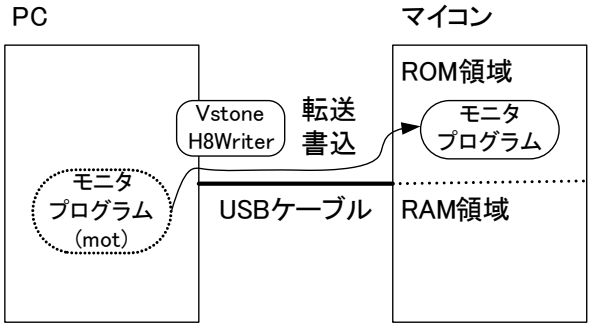


図. 1.3: 自作プログラムを ROM 領域に書き込み，実行

モニタプログラムを利用する場合，自作プログラムをマイコンで実行するまでの手順:

1. モニタプログラムの書込 (Vstone H8Writer)[一度だけ行う]
2. プログラムの編集 (HEW)
3. プログラムのコンパイルし (HEW)，mot ファイルを作成する
4. mot ファイルのロード (HTERM)
5. プログラムの実行 (HTERM)

ここで，括弧の中は利用するソフトウェアを示している．モニタプログラムは既に用意しているので，モニタプログラムを一度だけ ROM 領域に書き込む.

1.4 モニタプログラム

1.4.1 モニタプログラムとは

モニタプログラムとは，マイコンの状態を確認したり，コンパイルして作成される mot ファイルを RAM 領域にロードし，実行する機能を備えた便利なプログラムである．

モニタプログラムは，シリアルインターフェイス (USB ケーブル) でパソコンと通信が可能である．マイコンの状態の確認や mot ファイルのロードは，シリアル通信を介して行う．

1.4.2 モニタプログラムの書き込み方法

モニタプログラムを利用するためには，モニタプログラムを ROM 領域に書き込む必要がある．ROM 領域に書き込むためには，Vstone H8Writer というソフトウェアを利用する．

モニタプログラムの書込手順：

1. MONITOR.MOT を自分のパソコンに用意する．MONITOR.MOT の用意の仕方はスタッフの指示に従うこと．
2. 「H8Writer 取扱説明書」の指示に従い，MONITOR.MOT を書き込む.
3. HTERM を利用して，モニタプログラムが書き込まれたことを確認する．HTERM の使い方は，次節を参照すること．

1.4.3 モニタプログラムの機能

モニタプログラムには，自作のプログラムをロードし，実行する以外に，ステップ実行や，メモリの確認，メモリの書き換えなどの機能が備えられている．詳細は,「モニタプログラム readme.htm」(monitor.htm) を参照すること．

なお，都合により,「繰り返し実行」「コマンド履歴」「アボート管理」はモニタプログラムの機能としては利用できない.「繰り返し実行」「コマンド履歴」については，Hterm に同様の機能が備えられているので，問題ないが,「アボート管理」は利用できないので，注意が必要である．

1.4.4 注意事項

モニタプログラムでは，マイコンの RAM 領域に自作プログラムをロードする．しかし，本実習のマイコンの RAM 領域は，2K バイトとそれほど大きくない．そのため，浮動小数点 (float, double) や printf, scanf などを多用すると，メモリが足りなくなる場合がある．そのような場合は，以下のようなエラーが発生する．

L2321 (E) Section ”B” overlaps section ”P”

このようなエラーが発生した場合は，浮動小数点演算や，printf, scanf などの利用をやめてみること．また，多くのマイコンのプログラムでは，浮動小数点を利用しなければならないことはほとんどなく，固定小数点 (char, int など) だけで，十分である．本十種運において，固定小数点のみの演算で十分である．

1.5 Hterm

1.5.1 Hterm とは

モニタプログラムとパソコンは，シリアル通信を介して，データのやりとりを行う．従って，モニタプログラムを操作するためには，パソコン側でも，シリアル通信をサポートしているターミナルソフトウェアを利用する必要がある．本実習では，ルネサステクノロジが提供している Hterm というターミナル・ソフトウェアを利用する．Hterm は，一般的なターミナル・ソフトウェアの機能に加えて，モニタプログラムに対応した便利な機能を備えている．

1.5.2 Hterm を利用してモニタプログラムと通信を接続する方法

マイコンとパソコンを接続する場合：

1. モニタプログラムが書き込まれたマイコンとパソコンを USB ケーブルで接続する．
2. マイコンに電源が入り，LED1(オレンジ) が点灯する．
3. 「Hterm」を実行する．
4. マイコンの押しボタンスイッチを押す．以下のようなメッセージが表示されれば，成功．

   H8/36064 Series Normal Mode Monitor Ver. 2.0B
   Copyright (C) 2003 Renesas Technology Corp.

マイコンの電源を入れ直す場合：

1. マイコンを USB ケーブルからはずす（電源 OFF).
2. Hterm の「通信」→「切断」を選択し，通信を切断する．
3. マイコンを USB ケーブルに接続する (電源 ON).
4. Hterm の「通信」→「接続」を選択し，通信を接続する．
5. マイコンの押しボタンスイッチを押し，メッセージが表示されることを確認する．

メッセージが表示されない場合 (文字化けしている場合)：

1. Hterm の「通信」→「切断」を選択し，通信を切断する．
2. Hterm の「ファイル」→「プロパティ」を選択し，Hterm のプロパティを表示する．
3. 通信ポート (C)：COM3，ビットレート (B):38400 に設定し,「OK」する．
4. 自動的に接続されるので，マイコンの押しボタンスイッチを押し，確認する．

1.5.3 プログラムのロードと実行

HEW を利用して，プログラムをコンパイルすると，abs ファイル (拡張子が abs のファイル) が作成される．この abs ファイルをモニタプログラムにロードし，プログラムを実行する．

abs ファイルをロードし，実行する場合：

1. Hterm とマイコンの通信が接続している状態で，マイコンの押しボタンスイッチを押し，メッセージが表示されることを確認する．
2. Hterm の「コマンド」→「Load」を選択，または,「F9」を押す．
3. 「ユーザプログラムのダウンロード」というダイアログがあらわれる．ファイルの種類 (T): ELF/DWARF (*.abs) に設定する．
4. 作成した abs ファイルを選択し,「開く」を押す．
5. ロードが終了した後,「ソースプログラムを表示しますか？」というダイアログがあらわれた場合は,「はい」を押す．ソースプログラムが表示される．
6. Hterm の「コマンド」→「Go」を選択，または,「F5」を押すと，プログラムが実行される．

なお，デバッグを行わない場合は，mot ファイルをロードしても良い．

注意

プログラムをロードして，実行しても，すぐにモニタプログラムのプロンプトに戻ってしまうことがある．その場合は，あせらずに，もう 1 度，プログラムをロードして，実行を試みる．どういうわけか，タイマ Z の割り込みを利用した場合，モニタプログラムのプロンプトに戻ってしまうことがある．2 度繰り返すことで，問題なく実行できる．

3 度，4 度と繰り返しても実行できない場合は，スタッフに相談すること．

1.5.4 メモリの確認

モニタプログラムの dump コマンドを利用して，メモリの内容を確認できる．

モニタプログラムが接続している状態で，Hterm の「表示」→「Dump」を選択すると,「ダンプ情報の指定」ダイアログがあらわれる．確認したいメモリの先頭アドレスを指定することで，メモリの内容を確認することができる．サイズについて，バイトのままで良い．モニタプログラムによるメモリの確認では，16 進数表記と，ASCII コードが表示されている．データの表現については，次節を参照．

別の方法としては，Hterm の Console から D コマンドを利用する方法もある．D コマンドの詳細に

ついては,「モニタプログラム monitor.htm (使い方)」(monitor.htm) の「メモリ内容のダンプ 」の項を参照.

1.5.5 周辺機能のレジスタの確認

モニタプログラムのコマンドを利用して，周辺機能のレジスタの状態を確認することができる．

モニタプログラムが接続している状態で，Hterm の「表示」→「Periferal」を選択すると,「周辺機能の選択」ダイアログがあらわれる．そこで，確認したい周辺機能を選択することで，周辺機能のレジスタを確認することができる．

周辺機能と，そのレジスタの意味の詳細は,「H8 36064 グループハードウェアマニュアル」(rjj09b0049_h836064.pdf) を参照.

なお，モニタプログラムのこの機能は，H8 コマンドに対応している．H8 コマンドの詳細については,「モニタプログラム monitor.htm (使い方)」(monitor.htm) の「内蔵周辺機能の状態表示」の項を参照.

1.5.6 データの表現

データは単なる数値であり，その数値のひとまとまりの長さ (バイト，ワードなど) や表現 (2 進数表現，16 進数表現，10 進数表現，ASCII コードなど) は目的に応じて，人間が考える必要がある．

ビット (bit) マイコン内部での最小のデータ単位が 1 ビットである．この単位には，0 か 1 かのどちらかの値を取り，それ以外の値は存在しない．0 と 1 は，電気的に電位のあるなしに対応している．

バイト (byte) 1 ビットを 8 個まとめた単位が，1 バイトである．1 バイトは 2 進数で 8 桁 (例:00110110) で表される．また，1 バイトは 16 進数では 2 桁 (例：36) で表される．1 バイトで表現可能な組合せは，$2^8 = 256$ 通り，0 から 10 進数の 255 までである．

ワード (word) 1 バイトでは表現しきれない内容に対応するために，2 バイトや 4 バイトをまとめた単位がワードやロングワードである．

10 進数表記	0	1	2	3	4	5	6	7
2 進数表記	0000	0001	0010	0011	0100	0101	0110	0111
16 進数表記	0	1	2	3	4	5	6	7

8	9	10	11	12	13	14	15
1000	1001	1010	1011	1100	1101	1110	1111
8	9	A	B	C	D	E	F

2 バイトデータは，本実習で利用するマイコン内部では，上位バイト下位バイトの順に格納されている．このような順に配置されていることをビッグエンディアンと呼ぶ．逆に下位バイト上位バイトの順に格納されている場合，リトルエンディアンと呼ばれる[1].

マイコン内部では，例えば，4 桁の 16 進数の AB12 は，AB 12 の順に格納される．

[1] インテル系の CPU はリトルエンディアンであることが多い.

1.5.7 デバッグ方法

簡易的なデバッグ方法としては，printf を利用する方法がある．printf を利用したデバッグの方がわかりやすい場合も多い．比較的，簡単にデバッグすることができるので，本実習では，printf を利用したデバッグ方法を主に利用することにする．printf の使い方は次節を参照すること．

モニタプログラムのブレークポイント (b コマンド) やステップ実行 (s コマンド)，Hterm の機能である 1 行実行 (Pstep) を利用して，デバッグを行うこともできる．ステップ実行などを行った場合は，モニタプログラムを利用して，レジスタやメモリの状態を確認することができるので，より詳細なデバッグを行うことが可能である．モニタプログラムのコマンドの詳細については,「モニタプログラム monitor.htm (使い方)」(monitor.htm) の「メモリ内容のダンプ 」の項を参照.

なお，モニタプログラムを利用したデバッグでは，printf や scanf，SET_VECTOR_TABLE といった関数を利用すると正しく動作しないことがあるので，注意が必要である．resetprg.c の中でで，main 関数を呼び出す直前で，SET_VECTOR_TABLE が呼び出されているので，モニタプログラムのデバッグ機能を利用する場合は，SET_VECTOR_TABLE をコメントアウトする必要がある．

1.6 サンプルプログラム

簡単なサンプルプログラムを用意している．このサンプルプログラムを，実際にコンパイルし，実行してみる．C:￥WorkSpace に 010MonitorMemory.zip を解凍する．

1.6.1 HEW の起動方法とビルド方法

パソコンの「すべてのプログラム」→「Renesas」→「High-performance Embedded Workshop」→「High-performance Embedded Workshop」を選択し，実行することで，HEW を起動することができる．その後,「ファイル」→「ワークスペースを開く」から，編集したいワークスペースを開く．

C 言語のソースコードから，マイコンが理解できるコードに変換することをビルドという．ビルドを行うためには,「ビルド」→「ビルド」を選択するか,「F7」を押すとビルドが行われる．下のウィンドウの Build タブに 0 errors と表示されれば，成功である．ビルドが行われると，C:￥WorkSpace ￥010MonitorMemory￥MonitorSample￥Debug に，abs ファイルや mot ファイルなどが生成されている．

この abs ファイルまたは mot ファイルを，Hterm からモニタプログラムにロードし，実行を行う．

1.6.2 printf と scanf

モニタプログラムを利用する利点の 1 つに printf と scanf が利用できることがある．

printf の出力結果はシリアル通信を介して，Hterm に表示される．また，scanf はシリアル通信を介して，Hterm からの入力を受け付ける．

なお，この printf と scanf の機能は，モニタプログラムの機能であるため，自作プログラムを直接 ROM 領域に書き込んだ場合は，printf および scanf は利用できないので，注意が必要である．

printf と scanf については，C 言語の標準ライブラリと利用方法は，ほぼ同じである．

1.6.3 ソースコード

```
/* サンプルコード 1.1 */
void main(void)
{
    int i, offset;
    int array[6];
    char message[16];

    printf( "Input offset: " );
    scanf( "%d", &offset );
    printf( "\n" );

    for( i = 0; i < 6; i++ ){
        array[i] = i*i+offset;
        printf( "%d %3d 0x%02x\n", i, array[i], array[i] );
    }
    printf( "\n" );

    printf( "array dump: %x\n", array );
    printf( "message dump: %x\n", message );

    i = 0;
    message[i++] = 0x48;
    message[i++] = 0x65;
    message[i++] = 0x6C;
    message[i++] = 0x6C;
    message[i++] = 0x6F;
    message[i++] = 0x20;
    message[i++] = 0x54;
    message[i++] = 0x6F;
    message[i++] = 0x6B;
    message[i++] = 0x79;
    message[i++] = 0x6F;
    message[i++] = 0x54;
    message[i++] = 0x65;
    message[i++] = 0x63;
    message[i++] = 0x68;
    message[i++] = 0x21;

    RESET();
}
```

1.7 実習課題

1-1. (必修) モニタプログラムをマイコンの ROM へ書き込み，ヘルプを表示することで，動作確認を行う．
1-2. (必修) サンプルプログラムをビルドし，モニタプログラムにロードし，実行する．ただし，サンプルプログラムには，2 カ所バグがあるので，バグを修正して，ビルドを行うこと．
1-3. (必修) モニタプログラムを利用して，メモリの値を確認する．array にはビッグエンディアンでデータが格納されていることを，message には ASCII コードのメッセージを，それぞれ確認すること．
1-4. (オプション) モニタプログラムのその他の機能を使ってみる．

1-1(必修)	1-2(必修)	1-3(必修)	1-4(オプション)

2　入出力ポート

2.1　目標

2-1. アドレス，メモリを理解する．
2-2. 入出力ポートを利用できるようになる．
2-3. ブザーを鳴らす．

2.2　マイコンと外部入出力

マイコンを利用する目的は，計算を行わせるだけではない．例えば，炊飯器の中のマイコンは何をしているか考えてみよう．炊飯器の中のマイコンは，釜の温度，圧力や経過時間をチェックし，それらに応じて火加減を調整している．

つまり，本実習で考えているマイコンを利用する目的とは，外部の情報を取得し，取得した情報に応じて，外部へなんらかの働きかけをすることである．

そのためには，外部の情報を入力する外部入力と，外部へ働きかけを行う外部出力が必要になる．このような関係を図 2.1 に示す．外部入力の例としては，スイッチ，マウス，キーボード，各種センサ，通信ポートなどがある．外部出力の例としては，LED，モータ，サーボ，各種アクチュエータ，通信ポートなどがある．具体的な外部入出力機器については，メカトロニクスラボ電気編で平行して学習している．

CPU では，外部入力の情報に基づき，どのような値を出力するかはに外部へ出力するかを，計算（演算）する．当然，外部入力が与えられたとしても，CPU のみで計算（演算）ができるわけではない．計算（演算）を行うためには，それを行うプログラムとデータが必要であり，プログラムとデータはメモリに格納されている．また，必要に応じて，メモリ内部のデータは書き換えられる．このような形式のコンピュータはノイマン型コンピュータと呼ばれている．

さらに，マイコンには，CPU とメモリだけでなく，タイマや A/D 変換器といった便利な周辺機能

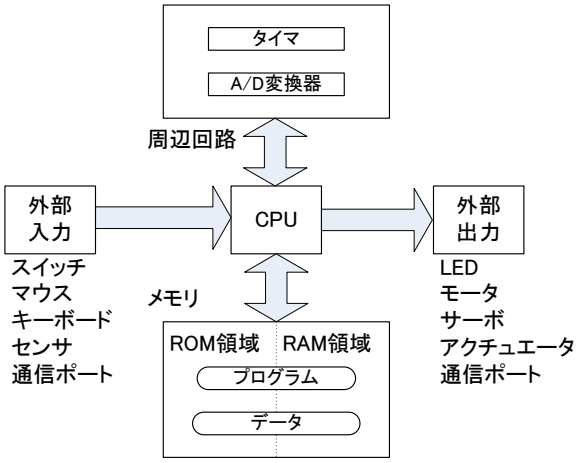


図. 2.1: CPU とメモリ，外部入出力の関係

が搭載されていることが多い．本実習で利用しているマイコンにも，タイマと A/D 変換器が搭載されている．

図 2.1 を見ると，CPU が文字通り中心となり，計算（演算）が行われている様子がわかる．図 2.2 に本実習で利用する Renesas 社製 H8/36064 マイコンの内部ブロック図を示す．

2.3 アドレスとメモリ，外部入出力，周辺機能

ところで，CPU はどのようにデータのやりとりをするのだろうか．図 2.2 を見ると，CPU とメモリや，タイマ，外部入出力がアドレスバスとデータバスで接続されている様子が確認できる．

CPU がデータを読み取る場合，(1) アドレスバスにデータを読み取るアドレス (場所) をセットし，(2) ついでデータバスからデータを読み取る．CPU がデータを書き込み場合は，(1) アドレスバスにデータを書き込むアドレス (場所) をセットし，(2) ついでデータバスにデータを書き込む．

アドレスは，データを読み書きする場所を表している．メモリにアドレスが割り当てられているのは直感的に理解しやすいだろう．メモリとはデータを格納しておく「箱」のようなものなので，全て

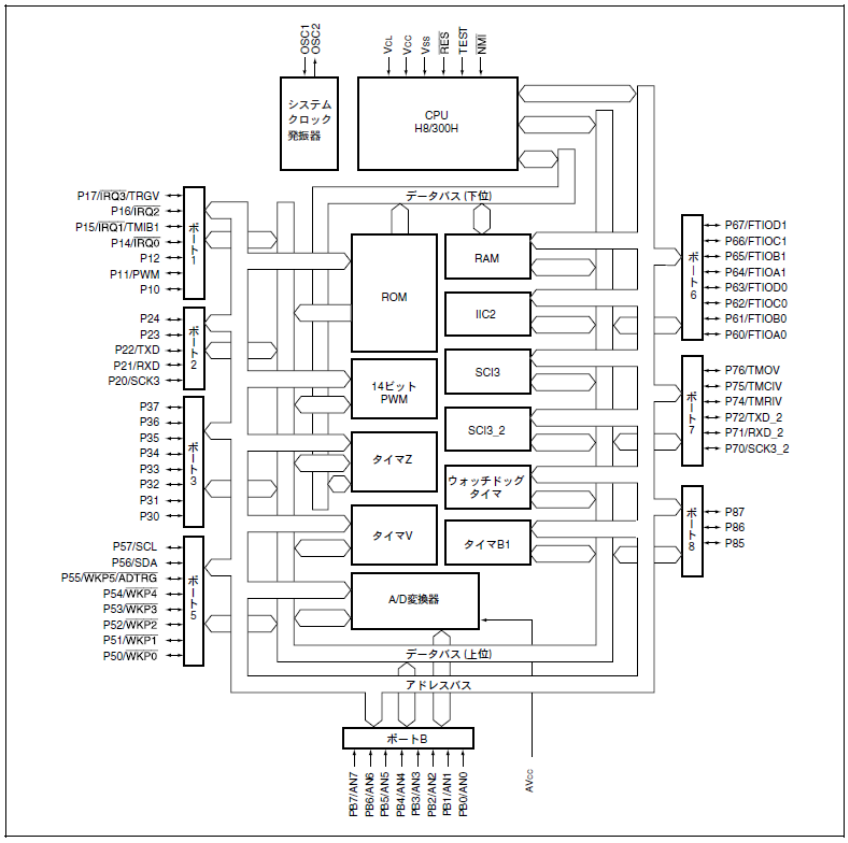


図. 2.2: 内部ブロック図 (Renesas H8/36064 ハードウェアマニュアルより)

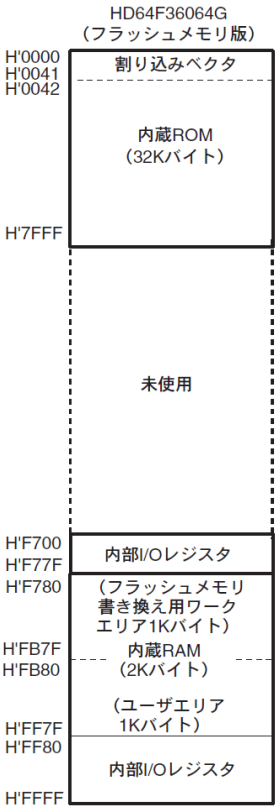


図. 2.3: メモリマップ (Renesas H8/36064 ハードウェアマニュアルより)

の「箱」に場所を表す「住所」つまり，アドレスが割り当てられている．このアドレスは，メモリのみならず，外部入出力や周辺機能にも同様に割り当てられている．このように，アドレスが，外部入出力や周辺機能にも割り当てられていることにより，CPU から考えた場合，外部入出力や周辺機能のデータのやりとりも，メモリとのデータのやりとりと全く同様に行うことができる．

従って，メモリの値を読み取る場合と同様に，(1) 外部入力に対応するアドレスをアドレスバスにセットし，(2) ついでデータバスからデータを読み取ることで，外部入力の値を読み取ることができる．外部出力へ値を出力するときも同様である．

CPU が読み書きできる機器 (メモリ，外部入出力，周辺機能) には全て，唯一のアドレスが割り当てられている．そのため，アドレスバスにアドレスをセットすることにより，間違うことなく対応する機器のデータを読み書きすることができる．

言い換えると，アドレスとは，機器 (メモリ，外部入出力，周辺機能) それぞれを対応させるための「名前」(「住所」,「番地」,「場所」) であると言える．データとは，機器 (メモリ，外部入出力，周辺機能) の状態を表すものである．

図 2.3 に本実習で利用する Renesas 社製 H8/36064 マイコンのメモリマップを示す．

2.4 入出力ポート

2.4.1 入出力ポートとは

入出力ポートとは，外部入出力機器とマイコンを接続する代表的な手段である．具体的には，入出力ポートは，マイコンの「足」に対応しており，外部入出力機器とマイコンの「足」を電気的に接続する．入力ポートにスイッチが接続されていれば，対応するポートの状態 (0 か 1 か) を調べることで，スイッチの状態が確認できる．また，出力ポートに LED が接続されていれば，マイコンから LED を点灯させたり，消灯させたりすることができる．

図 2.4 に本実習で利用する，H8/36064 マイコンのピン配置を示す．この図の中で，P64 や P65 がポートの名前を示している．また，一つのピンに複数の名前が付けられている場合がある．これは，一つのピンで複数の役割を共有しており，実際に利用する場合は，どの役割で利用するかを設定する必要がある．

本実習では Vstone 社製の WRC003LV というマイコンボードを利用している．このマイコンボードでは，既にいくつかのピンはブザーや，シリアル通信用に接続されており，全てのポートが利用できるわけではないので，注意が必要である．

主な接続関係を表 2.2 に示す．詳細は,「マイコンボード VS-WRC003LV 回路図」で確認できる．

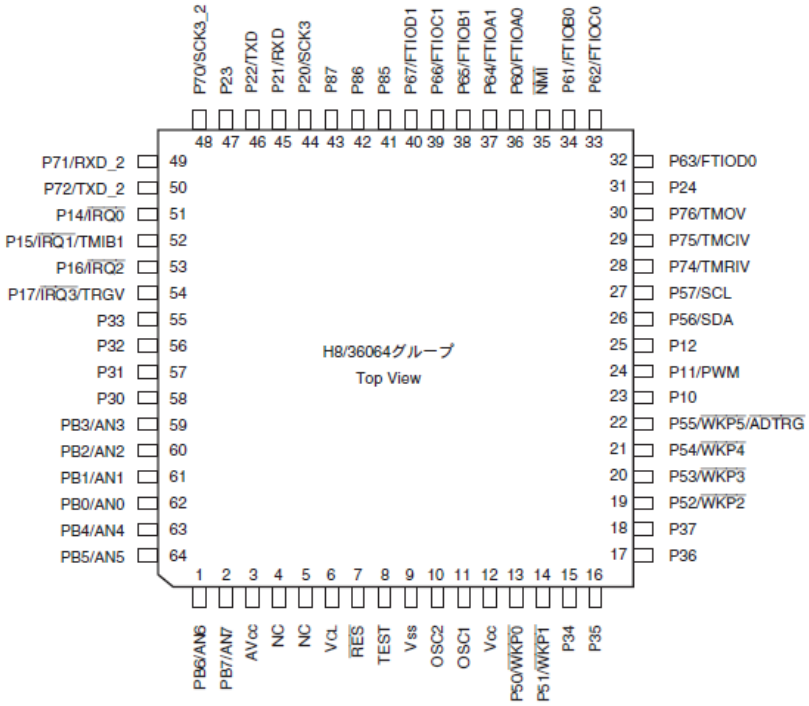


図. 2.4: ピン配置図 (Renesas H8/36064 ハードウェアマニュアルより)

Table 2.2: ポートの接続関係

ポートの名前	接続先	備考
P21/RXD	HID-SERIAL-IC の TXD	シリアル通信（受信）
P22/TXD	HID-SERIAL-IC の RXD	シリアル通信（送信）
P61/FTIOB0	モータドライバ 1 の PWM	タイマ Z の出力，モータドライバ 1 を介して CN3 への出力を制御
P30	モータドライバ 1 の制御 1	モータドライバ 1 を介して CN3 への出力を制御
P31	モータドライバ 1 の制御 2	モータドライバ 1 を介して CN3 への出力を制御
P62/FTIOC0	モータドライバ 2 の PWM	タイマ Z の出力，モータドライバ 2 を介して CN4 への出力を制御
P32	モータドライバ 2 の制御 1	モータドライバ 2 を介して CN4 への出力を制御
P33	モータドライバ 2 の制御 2	モータドライバ 2 を介して CN4 への出力を制御
P74	押しボタンスイッチ	押してるとき 0，離しているとき 1
$\overline{\text{NMI}}$	押しボタンスイッチ	P74 と NMI が接続されている
P60	LED1（オレンジ）	0 で点灯，1 で消灯
P64	LED2（緑）	0 で点灯，1 で消灯
P76/TMOV	ブザー	タイマ V の出力
PB0/AN0	アナログ入力 (CN4)	
PB1/AN1	アナログ入力 (CN5)	
PB2/AN2	アナログ入力 (CN6)	入力コネクタをハンダ付けが必要
PB3/AN3	アナログ入力 (CN7)	入力コネクタをハンダ付けが必要

2.4.2 入出力ポートの名前

既に述べたように，全ての入出力ポートには唯一のアドレスが割り当てられているので，アドレスを指定することにより，入出力ポートにアクセスすることが可能である．しかしながら，実際にプログラムを開発する上では，無味乾燥的なアドレス番号を指定することは，効率的ではなく，バグの原因にもなる．そのため，入出力ポートや周辺機能に対応するアドレスに，名前をつけてプログラムを開発することがほとんどである．

H8 シリーズでは，入出力ポートや周辺機能のアドレスの命名規則が決まっており，その規則に従って，名前が付けられている．利用可能な入出力ポートや周辺機能のアドレスに対応する名前 (変数名) は iodefine.h ファイルで定義されている．

2.4.3 入出力ポートの設定と使い方

入出力ポートを実際に利用する前に，入出力ポートの機能を設定しなければならない．入出力ポートの機能とは，入出力の方向や，ポートのモード，プルアップする/しないなどである．入出力ポートの機能の設定したり，データを入出力するためには，特定のレジスタに値を設定すれば良い．入出力ポートに関連するレジスタを以下にまとめる．

PMR(ポートモードレジスタ) 汎用入出力ポートとして利用するか，特定の機能として利用するかを設定する．ポート毎に機能が決まっており，詳細は「H8 36064 グループハードウェアマニュアル」(rjj09b0049_h836064.pdf) を参照.

PCR(ポートコントロールレジスタ) 汎用入出力ポートの入出力の方向を設定する．出力ポートとして利用する場合は 1 を，入力ポートとして利用する場合は 0 を，設定する．

PUCR(ポートプルアップコントロールレジスタ) 入力ポートとして設定されている場合のみ有効である．プルアップ MOS を有効にする場合は 1 を，無効にする場合は 0 を，設定する．

PDR(ポートデータレジスタ) 入力ポートとして設定されている場合は，データを取得する．出力ポートとして設定されている場合は，データを出力する．

ポートのレジスタの名前

PORT	PMR	PCR	PUCR	PDR
P1	IO.PMR1	IO.PCR1	IO.PUCR1	IO.PDR1
P2	IO.PMR3	IO.PCR2	-	IO.PDR2
P3	-	IO.PCR3	-	IO.PDR3
P5	IO.PMR5	IO.PCR5	IO.PUCR5	IO.PDR5
P6	-	IO.PCR6	-	IO.PDR6
P7	-	IO.PCR7	-	IO.PDR7
P8	-	IO.PCR8	-	IO.PDR7
PB	-	-	-	IO.PDRB

注意：プログラムで利用する場合は，BYTE や，BIT.B?などが必要なこともある．
iodefine.h を確認すること．

2.4.4 出力ポートの使用例

LED2（緑）を制御する場合を考える．表 2.2 によれば，P64 が LED2（緑）に接続されている．従って，P64 を利用することで，LED2（緑）を制御することができる．

まず，IO.PCR6 の 4 ビット目を 1 に設定し，P64 の入出力方向を出力方向に設定する．その後，IO.PDR6 の 4 ビット目を 0 に設定することで，LED2（緑）を点灯させることができる．LED2（緑）を点灯させるサンプルプログラムを以下に示す．

```
/* サンプルコード 2.1 */
void main(void)
{
    IO.PCR6 |= 0x10;     /* IO.PCR6 の 4 ビット目を 1 にセット (P64 を出力ポートに設定) */
    IO.PDR6.BIT.B4 = 0; /* IO.PDR6.BIT.B4 を 0 にセット (P64 の出力を Low に設定)        */

    for(;;){             /* 無限ループ */
    }
}
```

2.4.5 入力ポートの使用例

押しボタンが押されているときと，押されていないときで，異なる処理をすることを考える．表 2.2 によれば，P74 が押しボタンに接続されている．従って，P74 の状態に基づき，押しボタンが押されているか，押されていないかを判断することができる．

まず，IO.PCR7 の 4 ビット目を 0 に設定し，P74 を入力ポートに設定する．既にプルアップ抵抗が接続されているので，プルアップは OFF のままでよい．

IO.PDR7 の 4 ビットが，0 であれば押しボタンが押されているとき処理を行い，1 であれば押されていないときの処理を行うことができる．以下にサンプルプログラムを示す．

```
/* サンプルコード 2.2 */
void main(void)
{
    IO.PCR7 &= 0xEF;       /* 4 ビットを 0 に設定 (P74 を入力ポートに設定) */

    for(;;){ /* 無限ループ */
        if( IO.PDR7.BIT.B4 == 0 ){
            /* 押しボタンが押されているときの処理*/
        }else{
            /* 押しボタンが押されていないの処理*/
        }
    }
}
```

2.5 ブザー

表 2.2 よれば，P76/TMOV とブザーが接続されている．このブザーは，ピエゾ素子によって駆動しており，ON/OFF を周期的に変更することにより，音が出る．周期的に ON/OFF を変更するような処理には，周辺機能のタイマを利用するのが良い．タイマの詳細な利用方法は，次章で学習することとし，ここでは，ブザーを鳴らせることのみを目的とする．

以下に，ブザーを鳴らすためのサンプル関数を示す．

```
/* サンプルコード 2.3 */
void BuzzerSet(unsigned char pitch , unsigned char vol)
{
    TV.TCRV0.BYTE=0x00;
    TV.TCRV1.BYTE=0x01;
    TV.TCSRV.BYTE=0x06;

    TV.TCORA=(unsigned char)( ((unsigned int)(pitch) * (unsigned int)(vol)) >> 8 );
    TV.TCORB=pitch;
}

void BuzzerON() /* buzzer start */
{ TV.TCRV0.BIT.CKS = 0x03; }

void BuzzerOFF() /* buzzer stop */
{ TV.TCRV0.BIT.CKS = 0x00; }
```

タイマ V と TMOV は関連付いており，タイマ V の動作に伴って，TMOV の出力が変化する．タイマを使用する場合も，入出力ポートと同様に，まず，タイマの設定を行う必要がある．

関数 BuzzerSet で，タイマの設定を行っている．BuzzerSet は二つの引数 pitch と vol を設定することができる．pitch はおよそブザー音の音の高さに関連しており，vol はおよそブザー音の音の大きさに関連している．これらのパラメータは，およそであり厳密に音の高さや，大きさに関連しているというわけではない．

上記関数の使い方は，まず，ブザーを使用するために，pitch および vol を設定して，BuzzerSet を呼び出し，ブザーを使用する準備を行う．準備ができたら，BuzzerON() を呼び出し，ブザーを鳴らす．一旦，BuzzerON() で，ブザーを鳴らすと，BuzzerOFF() を呼び出すまで，鳴り続けるので注意すること．

マイコンで実行した際に，ブザーが止まらない場合や，動作がおかしい場合は，マイコンの電源を OFF にする (USB ケーブルをはずす) こと．

2.6 実習課題

2-1. (必修) 前回のサンプルプログラムを修正し，LED2（緑）を点灯させるプログラムを作成する．
2-2. (必修) 押しボタンが押されているとき，LED1（オレンジ）点灯，LED2（緑）消灯，押されていないとき，LED1（オレンジ）消灯，LED2（緑）点灯するプログラムを作成する．
2-3. (必修) ブザーのサンプル関数を利用して，押しボタンが押されているとき，ブザーを鳴らす．できる範囲で，タイマ V の動作を，ハードウェアマニュアルで確認する．
2-4. (オプション) タイマ V の設定を変更して，ブザーの音の種類を変更するプログラムを作成する．

2-1(必修)	2-2(必修)	2-3(必修)	2-4(オプション)

3　割り込み処理とタイマ

3.1　目標

3-1. 割り込み処理を理解する．
3-2. タイマを理解する．
3-3. 入力ポートを監視する方法を理解する．

3.2　割り込み処理

3.2.1　フロー駆動型システムとイベント駆動型システム

フロー駆動型システムとは，処理の順番がはじめから決まっており，その順番に処理を行うシステム (プログラム) である．C 言語演習で学習してきたプログラムは，ほとんどフロー駆動型システムに対応している．例えば，あるデータを処理して，ファイルに保存するという処理は，予め決められた処理を順番に行う処理であり，フロー駆動型システムである．

しかしながら，我々が普段に利用しているシステムのほとんどは，イベント駆動型システムである．正確には，イベント駆動型システムとフロー駆動型システムが組み合わさっている場合が多い．ここで，イベントとは，マイコンへ対する何らかの働きかけである．例えば，マウスがクリックされた，ボタンが押されたなどは，マイコン外部からの働きかけ (イベント) であるので，外部イベントと呼ばれる．一方，次章で詳しく説明するようにタイマが一定の値になったなど，マイコンの内部で発生するイベントもあり，内部イベントと呼ばれる．CPU から考えた場合，外部イベントも内部イベントも，区別がなくただ，単なるイベントである．

我々が普段利用しているシステムは，何らかのイベントを待っている場合が多い．例えば，自動ドアでは人が近づいているというイベントを観測してドアを開ける．GUI と呼ばれるアプリケーションでは，マウスがクリックされた場所に応じて，対応する処理が行われる．クリックされた場所がインターネットエクスプローラのアイコンであれば，インターネットエクスプローラを立ち上げるといった具合である．

このように外部と接続されているシステムの多くは，イベントの発生を待ち，イベントが発生したらそのイベントに対応する処理を逐次的に処理するという仕組みになっている．

3.2.2　割り込み処理とは

割り込み処理とは，イベント駆動型システムを実現するために非常に便利な処理方法である．

イベントとそのイベントに対応する処理が対応づけられ，イベントが発生すると対応する処理を行うことができる．マイコンなどで多く利用される割り込み処理では，イベントのことを割り込み要求，対応する処理のことを割り込み処理と呼ばれる[2]．割り込み要求が発生すると，CPU は現在の処理を中断して，割り込み要求に対応する割り込み処理を行う．

[2]Windows もイベント駆動型システムであるが，Windows では割り込み処理ではなく，メッセージという概念が利用されている．

なぜ，このような割り込み処理が必要なのだろうか．割り込み要求は，あるイベントが発生したことを CPU に知らせる仕組みである．このような仕組みが存在しない場合，CPU はイベントが発生したかどうかを定期的に確認する必要がある．

人間と携帯電話の例で考えてみる．ここで，人間が CPU，電話の着信がイベントである．着信音が鳴る携帯電話 (割り込み有り) と，着信音も鳴らずバイブレーション機能もオフにしている携帯電話 (割り込み無し) を例にとる．着信音が鳴らない携帯電話でも，画面を確認することで，着信があるかどうかを確認することはできる．しかしながら，この二つの携帯電話は着信履歴は残らないとする．

着信履歴が残らないので，着信時に必ず対応する必要がある．そのため，着信音が鳴らない携帯電話 (割り込み無し) では常に携帯電話の画面を確認する必要があり，別の仕事をすることができない．一方，着信音がなる携帯電話 (割り込み有り) では，着信音でお知らせ (割り込み要因) してくれるので，常に画面を確認する必要はなく，別の仕事をすることができる．

このように，割り込み処理は，次のような重要な利点がある．

1. 外部イベント (スイッチ入力など) に確実に応答できる．
2. 外部イベントに対する応答が速くなる．
3. 定期的に行う処理 (一定時間毎のサンプリングなど) が確実にできる．
4. 無駄なチェックが不要になるため，処理効率が高くなる．
5. 割り込み要因と割り込み処理が対応しているので，プログラムの構造がわかりやすくなる．

一方，次のような欠点もある．

1. 割り込みの概念を理解する必要がある．一度理解してしまえば，欠点にならない．
2. 割り込みに関連する設定を行う必要がある．
3. 多重割り込みへの対策が必要である．
4. デバッグがしにくい．
5. 割り込み処理では複雑な処理が行いにくい．T[sec] 毎に割り込み処理を行う場合，割り込み処理は T[sec] よりも，十分早く終了する必要がある．

3.2.3 割り込み処理の動作概要

ところで，マイコン内部ではプログラムはどのように実行されているのだろうか．マイコン内部では，プログラムは，マシン語と呼ばれる命令がメモリ内部に格納されている．この命令が順番に実行される．プログラムカウンタ (PC, Program Counter) である．プログラムカウンタは，現在実行すべき命令が格納されているアドレスを，示している．従って，プログラムカウンタを増加させながら，命令を実行していくことにより，マイコンでは命令が順番に実行していく．

さて，割り込み処理では，順番に実行されている処理 (メイン処理) に，割り込んで処理をする必要がある．そのためには，メイン処理を中断し，割り込み処理を行い，メイン処理を再開しなければならない．メイン処理を再開するためには，どの命令からメイン処理を再開させる必要があるかを，記

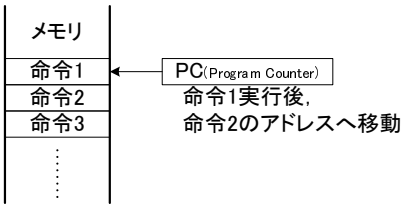


図. 3.1: プログラムの実行と PC

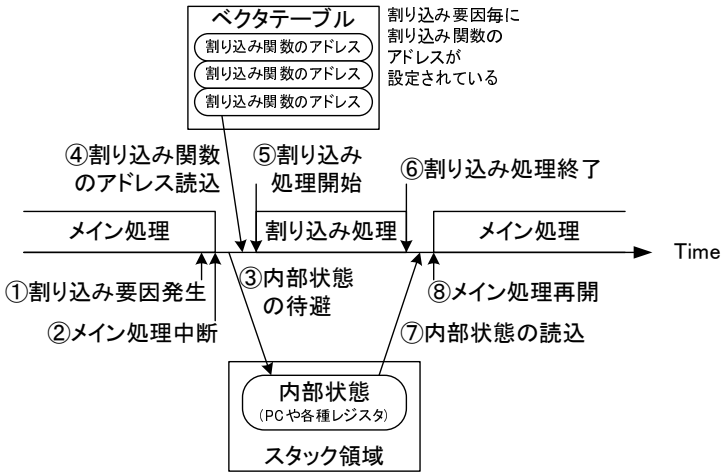


図. 3.2: 割り込み処理の動作概要

憶しておかなければならない．このためには，プログラムカウンタを記憶しておけばよい．また，プログラムカウンタだけでなく，メイン処理を再開するために必要な情報も同時に記憶される．このように，プログラムカウンタなどの情報を一時的に記憶しておく領域は，スタック領域と呼ばれる．

一方，割り込み要因が発生したとき，対応する割り込み処理の命令がアドレスを CPU に知らせる必要がある．このために，ベクタテーブルと呼ばれるメモリ領域が用意されており，割り込み要因に対応する関数のアドレスが，ベクタテーブルに格納されている[3]．つまり，ベクタテーブルにより，割り込み要因と割り込み関数が一対一に対応している．割り込み要因に対応する関数のアドレスをベクタテーブルから，プログラムカウンタに読み込むことで，割り込み処理が開始される．

このような，割り込み処理の動作は以下のようにまとめられる (図 3.2).

1. 割り込み要因発生.
2. メイン処理中断.
3. プログラムカウンタや内部状態を，スタック領域に待避.
4. 割り込み要因に対応する割り込み関数のアドレスを，プログラムカウンタに読み込む.
5. 割り込み処理開始.
6. 割り込み処理終了.
7. 待避したプログラムカウンタや内部状態を，スタック領域から読み込む.
8. メイン処理を再開.

[3]本実習では，ベクタテーブルに，割り込み関数のアドレスが既に格納されている．しかしながら，自動的にベクタテーブルに格納されるわけではないので，割り込み処理を利用する場合は，割り込み関数のアドレスをベクタテーブルに格納 (登録) する必要がある.

3.2.4 割り込み処理の設定

本実習のサンプルプログラムでは，割り込み要因に対応する割り込み処理が，関数として既に提供されている．表 3.3 に割り込み要因と割り込み処理関数をまとめる．本実習のサンプルプログラムでは，表 3.3 内の割り込み関数は，既に MonitorIntprg.c に既に用意されているので，対応する関数を修正することで，割り込み処理が行える．

ただし，INT_NMI 以外は，割り込み処理を許可する設定が必要である．割り込み許可の設定方法は，割り込み要因によって異なる．詳細は「H8 36064 グループハードウェアマニュアル」(rjj09b0049_h836064.pdf) を参照．

なお，タイマ B1 以外のタイマについては各タイマの章に，タイマ B1 と WKP については例外処理の章に，解説されている．

Table 3.3: 割り込み要因と割り込み処理関数 (Renesas H8/36064 ハードウェアマニュアルより一部抜粋)

発生元	割り込み要因	割り込み処理関数	優先度
外部割り込み端子 押しボタンスイッチ	NMI	INT_NMI	高
外部割り込み端子	WKP	INT_WKP	
タイマ V	コンペアマッチ A コンペアマッチ B オーバーフロー	INT_TimerV	
A/D 変換器	A/D 変換終了	INT_ADI	
タイマ Z0	コンペアマッチ インプットキャプチャ オーバーフロー	INT_TimerZ0	
タイマ Z1	コンペアマッチ インプットキャプチャ オーバーフロー	INT_TimerZ1	
タイマ B1	オーバーフロー	INT_TimerB1	低

3.2.5 割り込み処理の優先順位

割り込み処理が行われているときに，別の割り込み要因が発生した場合 (多重割り込み)，どのような動作になるのだろうか．この問題を解決するために，全ての割り込み要因には優先度が決まっており，優先度が高い割り込み要因の処理が優先される．

しかしながら，多重割り込みは，処理が複雑になりデバッグも難しくなることから，勧められない．割り込み処理は，できるだけ，簡潔に短い時間で処理が完了することが大切である．処理時間が短ければ，確率的に多重割り込みが生じる可能性が減少する．また，同様な理由により可能な限り，割り込みの利用は少ない方が望ましい．

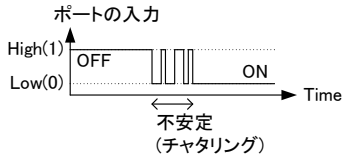
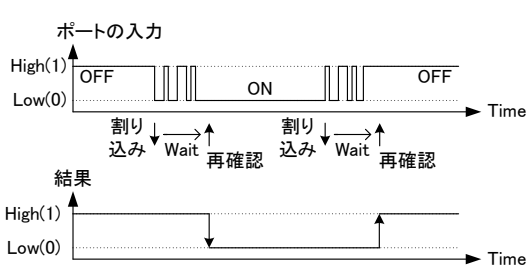
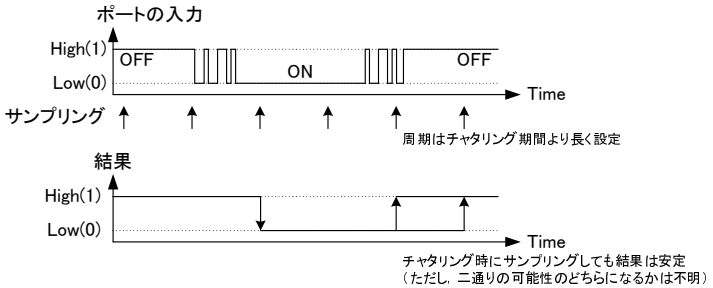


図. 3.3: チャタリング



(a) 割り込み後再確認方式



(b) サンプリング方式

図. 3.4: チャタリング防止

3.2.6　割り込み以外の方法

割り込み以外に，イベントを確認する方法に，常時監視方式とサンプリング方式がある．

常時監視方式は，文字通り，可能な限りずっとイベントを監視する方式である．サンプルコード 2.2 は，P74 のポートの変化を常時監視方式により，確認している．サンプリング方式は，一定間隔毎にイベントを確認する方式である．一定間隔毎にイベントを確認するためには，タイマ割り込みを利用する．タイマからの割り込みを利用しているので，確認したいイベントの変化から直接割り込み要因を発生させれば良いように感じられる．しかしながら，割り込みに対応していないポートに接続されている外部入力や，次節で述べるチャタリングを防止する目的で利用されることがある．なお，タイマに関しては 3.3 節で解説する．

3.2.7　タイマを利用したチャタリング対策

機械的なスイッチを利用する場合，ON と OFF が切り替わる瞬間は非常に不安定であり，短い時間に ON と OFF が不規則に変化する．このような不安定な状態をチャタリングと呼ぶ．図 3.3 にチャタリングの概念図を示す．

例えば，スイッチを押された回数をカウントしている場合，チャタリングの間は，複数回カウントされてしまい，正しい結果が得られない．また，ポートの入力の変化に対して，割り込みを発生させる場合も，チャタリングの間は，多数の割り込みが発生することになり，処理が不安定になる可能性が大きい．そのため，チャタリングに対して対策を行う必要がある．

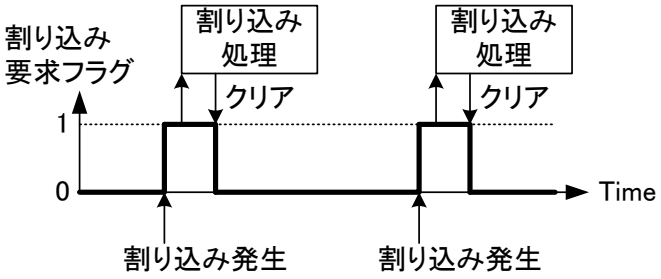
チャタリング対策には，大きく分けてハードウェア対策とソフトウェア対策がある．ハードウェア対策では，機械的なスイッチとマイコンの外部入力の間に回路を付加し，付加回路でチャタリングを

防止する方法である．付加回路には，フリップフロップや，LPF(Low-Pass Filter) とシュミットトリガ (ヒステリシス付の入力バッファ) を組み合わせた回路が利用される．ソフトウェア対策として，図 3.4 に示す割り込み後再確認方式とサンプリング方式がある．

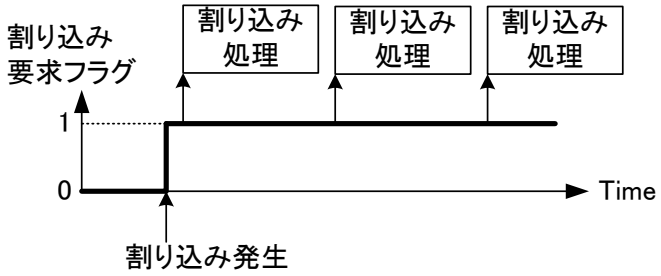
割り込み後再確認方式では，ポートの入力の変化に対して，割り込みを発生させ，チャタリング期間よりも長い間時間経ってから，再度ポートの入力を確認する方式である．この方式は，変化が生じた後，安定してから再度確認する方式である．ポートの入力変化に対して，割り込みを発生させることができる場合に利用できる．

サンプリング方式では，一定の周期でポートの入力を確認する方式である．サンプリングの周期をチャタリング期間よりも長く，人間の感覚よりも十分短くすることで，チャタリング防止が行える．仮にチャタリング期間にサンプリングしたとしても，ON/OFF の切替が多数生じるような不安定な結果にはならない．ただし，チャタリング期間にサンプリングした場合，出力が変化するタイミングが，二通りあり，どちらになるかはわからない．

また，サンプリング方式を利用して，押しボタンスイッチが押されたこと (ON から OFF または，OFF から ON に変化したこと) を検出するためには，1 サンプリング時刻前の状態を保持しておく必要がある．この場合，グローバル変数などを利用して，1 サンプリング時刻前の状態を保持しておき，現サンプリング時刻の状態と比較する必要がある．



(a) クリアする場合（正常動作）



(b) クリアしない場合（異常動作）

図. 3.5: 割り込みフラグと割り込み処理

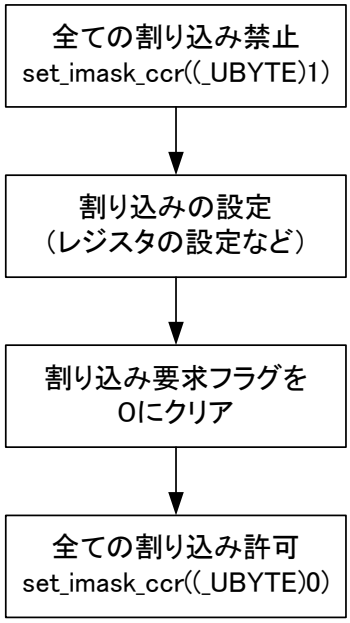


図. 3.6: 割り込み処理の設定手順

3.2.8 割り込みフラグと割り込みマスク, NMI

割り込みが許可されている場合，割り込みが発生すると，対応する割り込みフラグが 1 にセットされ，対応する割り込み処理関数が呼び出される．図 3.5 に概要を示す．このとき，割り込み処理では，必ず割り込みフラグを 0 にクリアすることが必要である．図 3.5-(a) に示すように，割り込みフラグを 0 にクリアすることにより，割り込み処理が正常終了できる．一方，図 3.5-(a) に示すように，割り込みフラグをクリアしないと，一旦，割り込み処理が終了しても，割り込みフラグが 1 のままであるので，再度割り込み処理が呼び出され，永久ループになってしまうので，注意が必要である．割り込み処理関数では，対応する**割り込みフラグを 0 にクリアすること**を忘れてはならない．

ところで，割り込みの設定中に割り込みが発生してしまったらどうなるのだろうか．設定が不完全な状態で，割り込みが発生してしまうと，CPU が正常に動作できなくなり，非常に不安定になってしまう．このような状態を避けるために，割り込みの設定をする前に，全ての割り込みを禁止する必要がある．割り込みを禁止することを，割り込みをマスクするとも呼ぶ．図 3.6 に割り込みの設定手順を示す．割り込みをマスクするためには set_imask_ccr という関数を引数 1 で呼び出す．また，割り込みマスクを解除するためには，set_imask_ccr 関数を引数 0 で呼び出す．

```
set_imask_ccr(1);     /* 割り込みマスクの設定 */
set_imask_ccr(0);     /* 割り込みマスクの解除 */
```

マイコンを利用したシステムでは，緊急事態などが発生したときに非常停止を行いたい場合がある．そこで，多くのマイコンでは，NMI(Non Maskable Interrupt) と呼ばれる非常に優先度が高い処理を行うために特別な割り込みが用意されている．この NMI は，set_imask_ccr 関数でもマスクされず，また，優先度が最も高く，常に割り込みが可能である．非常停止ボタンなどに接続されていると考えると良いだろう．非常停止ボタンが押された場合は，マイコンの内部でどのような処理が行われていたとしても，システムを停止しなければならない．このような用途に利用される．

本実習で利用しているマイコンボード (WRC003LV) では，押しボタンスイッチが，NMI と P74 に接続されているので，動作には注意が必要である．

3.3 タイマ

3.3.1 タイマ概要

タイマとは，基本的に設定した時間になったら，お知らせしてくれるアラームのようなものである．このタイマとは，図 3.7 に示すように，入力信号のパルス（立ち下がりエッジ，立ち上がりエッジ，または両エッジ）に応じてカウントアップし，カウンタの値をコンペアレジスタの値と比較し，条件を満足すれば，割り込みを発生したり出力の値を変更したりする機能を有する周辺機能である．なお，コンペアレジスタは複数存在する場合もある．カウンタの部分のみをタイマと呼んだり，図 3.7 の構成に対して，ITU(インテグレーテッド・タイマ・ユニット, Integrated Timer Unit) と呼ばれることもあるが，本実習では図 3.7 の構成を単にタイマと呼ぶことにする．

本実習で利用する H8/36064 には，タイマ B1，タイマ V，タイマ Z(2 チャンネル) の合計 4 個のタイマ搭載されており，それぞれ独立に利用可能である．なお，H8/36064 には関連して，WDT(ウォッチドッグタイマ，WatchDog Timer) と 14 ビット PWM が搭載されている．これらの詳細については，「H8 36064 グループハードウェアマニュアル」(rjj09b0049_h836064.pdf) を参照．

さて，タイマは図 3.7 に示すように簡単な構成になっている．この簡単なタイマを上手に利用することによって，様々な機能が実現でき，マイコンにおける重要な周辺機能の一つである．以下にタイマで実現できる主な機能を示す．

インターバルタイマ 設定した時間間隔で周期的に，割り込みを発生したり，出力を変更したりする．

ワンショットタイマ 設定した時間間隔後に一度だけ，割り込みを発生したり，出力を変更したりする．

パルス幅変調 (PWM) 任意のディーティ比の PWM を発生させる．インターバルタイマにより，PWM を発生させることも可能であるが，よく利用する機能であるため，PWM 用のモードが用意されているタイマもある．

イベントカウンタ 外部からのパルスをカウントする．

3.3.2 タイマの主な設定項目

タイマも入出力ポートと同様に，まずタイマの動作を設定する必要がある．タイマの動作を設定するためには対応するレジスタに値をセットすればよい．タイマの主な設定項目を以下に示す．

動作モード インターバルタイマモードや，ワンショットタイマモード，PWM モード，イベントカウンタモードなどを設定する．

カウンタクロックの選択 図 3.7 の入力信号を内部クロックにするのか，外部信号にするのかを設定する．また，内部クロックでは早すぎる場合も多いので，内部クロックの周波数を何分の一にするかといった設定もある．外部信号では，エッジの向きも設定できる場合もある．

カウンタクリア要因の選択 最も単純なカウンタクリア要因はオーバーフローである．コンペアレジスタの値でカウンタをクリアしたり，別の要因でカウンタをクリアすることもできる．

出力の設定割り込みを発生されるのか，対応する出力ポートの値を変更するのかなどを設定する．

これらの設定は，タイマによって異なり，全ての項目が設定できるわけではない．詳細については，「H8 36064 グループハードウェアマニュアル」(rjj09b0049_h836064.pdf) を参照．

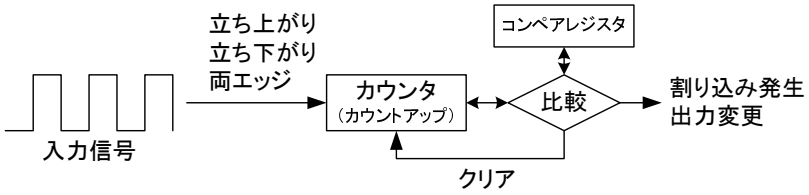


図. 3.7: タイマの構成概要

3.3.3 ブザーがなる仕組み (タイマ V)

前章では，タイマ V を利用して，ブザーを鳴らす方法を学習した．タイマ V がどのように動作して，ブザーを鳴らしていたかを改めて確認する．

タイマ V は 8 ビットタイマであり，タイムコンスタントレジスタを 2 つ有している．タイムコンスタントレジスタはコンペアマッチに利用される．カウンタの値とレジスタの値を比較して，等しい場合がコンペアマッチの状態である．コンペアマッチ時に対応する出力を変更することにより，パルス出力を生成する．

クロックは内部クロックと外部クロックから選択可能である．また，コンペアマッチで，カウンタをクリアしたり，出力の値を変更することが可能である．これらの機能を利用することにより，任意のデューティのパルス出力や PWM 出力をすることができる．

関数 BuzzerSet において，TCRV0，TCRV1, TCSRV, TCORA および TCORB の各レジスタが設定されている．表 3.4 に前章でブザーを鳴らした際の TCRV0，TCRV1 および TCSRV の各レジスタの設定値を，表 3.5 に設定レジスタの意味を，それぞれ表す．なお，リザーブビットに対して，書き込みを行おうとした場合，その書き込みは無視される．また，TCORA および TCORB はカウンタの値と比較を行うコンペアレジスタである．

これらの設定に基づき，関数 BuzzerSet を行った後のタイマ V の状態を確認してみる．

カウンタクロックの選択

表 3.5-(c) により，確認する．関数 BuzzerSet では，| CKS2 | CKS1 | CKS0 | ICKS0 | = | 0 | 0 | 0 | 1 | と設定されている．従って，クロック入力が禁止の状態である．

割り込み発生

割り込み発生に関係する，CMIEB,CMIEA および OVIE が，全て 0 であるので，割り込みは発生しない．

カウンタクリア条件

| CCLR1 | CCLR0 | = | 0 | 0 | と設定されているので，カウンタはクリアされない．ただし，オーバーフローはするので，オーバーフローが発生した時点で，カウンタは 0 になる．

タイムコンスタントレジスタ TCORA と TCORB

TCORB には pitch が，TCORA には pitch × vol/256 が設定される．なお，>> 8 は 8 ビット右にシフトすることを意味しており，整数を 8 ビット右にシフトすることは，256 で割ることと等しい．適宜，C 言語の参考書を参考にすること．また，unsigned char の最大値が 255 であることから，必ず TCORA < TCORB の関係が保たれる．

コンペアマッチ時の出力変化

| OS3 | OS2 | = | 0 | 1 | と設定されているので，カウンタの値 (TCNTV) と TCORB が一致したとき，出力 TMOV を 0 にする．

| OS1 | OS0 | = | 1 | 0 | と設定されているので，カウンタの値 (TCNTV) と TCORA が一致したとき，

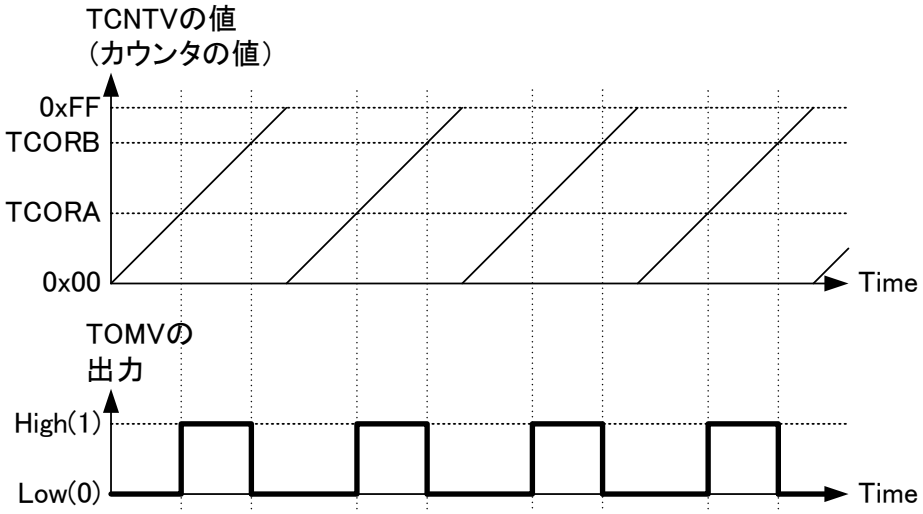


図. 3.8: TOMV 出力とタイマ V の関係

出力 TMOV を 1 にする．

カウンタクロックの選択で，説明したとおり，関数 BuzzerSet を呼び出した状態では，クロック入力が禁止されており，タイマは動作しない．関数 BuzzerStart で，CKS に 0x03 を設定している．関数 BuzzerStart が呼び出されると，$\boxed{\text{CKS2}}\boxed{\text{CKS1}}\boxed{\text{CKS0}}\boxed{\text{ICKS0}} = \boxed{0}\boxed{1}\boxed{1}\boxed{1}$となるので，表 3.5-(c) より，内部クロックΦ/128 でカウントがスタートし，ブザーが鳴る．なお，本実習で，利用しているマイコンボード WRC003LV の内部クロックは，12.0[MHz] である．

また， 関数 BuzzerStop で，CKS に 0x03 を設定している．関数 BuzzerStop が呼び出されると，$\boxed{\text{CKS2}}\boxed{\text{CKS1}}\boxed{\text{CKS0}}\boxed{\text{ICKS0}} = \boxed{0}\boxed{0}\boxed{0}\boxed{1}$となるので，表 3.5-(c) より，内部クロックの入力が禁止され，カウントがストップし，ブザーが止まる．

図 3.8 にタイマ V が動作中（ブザーが鳴っている状態）の，タイマのカウンタ値と出力 TOMV の関係を示す．TOMV はブザーに接続されており，ON/OFF を周期的に繰り返すことにより，ブザーが鳴る仕組みである．カウンタの値が時間とともに増加し，TCORA と一致したところで，TOMV が 1 に設定される．さらに，カウンタの値が増加し，TCORB と一致したところで，TOMV が 0 に設定される．さらに，カウンタの値が増加すると，カウンタの値はオーバーフローし，結果としてカウンタの値は 0 に戻る．

Table 3.4: ブザーを鳴らすためのタイマ V の設定

(a) TCRV0 の設定

ビット	7	6	5	4	3	2	1	0	
ビット名	CMIEB	CMIEA	OVIE	CCLR1	CCLR0	CKS2	CKS1	CKS0	HEX
設定値	0	0	0	0	0	0	0	0	0x00

(b) TCRV1 の設定

ビット	7	6	5	4	3	2	1	0	
ビット名	-	-	-	TVEG1	TVEG0	TRGE	-	ICKS0	HEX
設定値	0	0	0	0	0	0	0	1	0x01

(c) TCSRV の設定

ビット	7	6	5	4	3	2	1	0	
ビット名	CMFV	CMFA	OVF	-	OS3	OS2	OS1	OS0	HEX
設定値	0	0	0	0	0	1	1	0	0x06

Table 3.5: タイマ V の設定レジスタ (Renesas H8/36064 ハードウェアマニュアルより)

(a) TCRV0 の設定レジスタ

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 11.2 を参照してください。

(b) TCRV1 の設定レジスタ

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	TVEG1	0	R/W	TRGV 入力エッジセレクト
3	TVEG0	0	R/W	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 11.2 を参照してください。

(c) クロックとカウント条件 (CKS,ICKS)

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

(d) TCSRV の設定レジスタ

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

3.4 タイマ割り込みを利用してブザーを鳴らすサンプルプログラム

タイマ B1 と割り込みを利用して，ブザーを鳴らすことを考える．

3.4.1 タイマ B1 の概要と出力パルスの関係

タイマ B1 は，8 ビットタイマであり，内部クロックと外部クロック (イベント) を選択することができる．カウンタのオーバーフローで割り込みを発生することができる．コンペアマッチレジスタは有していないが，ロードレジスタを有しており，設定によりオーバーフロー時にロードレジスタの値がカウンタにセットされる．内部クロックとロードレジスタの設定により，オーバーフローが生じる周期をある程度設定することが可能である．

このタイマ B1 と割り込みを利用して，パルスを出力する．出力パルスをブザーへ入力することで，ブザーを鳴らす．図 3.9 に，タイマ B1 と出力の関係を示す．タイマ B1 がカウンタへの入力により，カウントアップする．タイマ B1 がオーバーフローした時点で，オーバーフロー割り込みが発生し，割り込み処理により，出力を反転 (トグル出力) させることにする．ここで，図 3.9 に示すように出力パルスの周期を 10[msec] になるように，タイマを設定することを考える．なお，出力パルスの周期は 10[msec] であるが，トグル出力を利用するので，タイマ B1 のオーバーフロー割り込みが発生する周期は，5[msec] となるようにしたい．

タイマ B1 に関連する設定レジスタを表 3.6 に，タイマ B1 の割り込みに関連するレジスタを表 3.7 に，それぞれ示す．

3.4.2 タイマ B1 の設定

まず，入力クロックを選択する．とりあえず，表 3.6 より，φ/512 とすることを考える．本実習のマイコンボードの内部クロックは 12.0[MHz] であるので，タイマ B1 の入力クロックは 12.0[MHz] / 512 $\simeq$ 23.438[KHz] となる．次に，オーバーフロー割り込みの周期を 5[msec] にするためには，23.438[KHz] のクロックのパルスをいくつカウントすればよいかを考える．これは，簡単に計算でき，5[msec] $\times$ 23.438[KHz] $\simeq$ 117[count] である．タイマ B1 は 8 ビット (256) であるので，カウンタの初期値 (TLB1) に，256-117=139 とすれば良いことがわかる．なお，今回の計算では，適当な近似計算を行っている．また，実際に設定値を決める場合は，計算の試行錯誤が必要になる．

タイマ B1 のカウント許可/禁止に対応するレジスタはなく，常にタイマ B1 はカウントを続けていると考えて良い．

3.4.3 タイマ B1 の割り込み設定

タイマ B1 の割り込み関連のレジスタを表 3.7 に示す．割り込み関連の設定は，割り込み許可/禁止を設定する IENR2 と，割り込み要求フラグ IRR2 の 2 ビット分だけである．

割り込みを許可するためには，IENR2 を 1 に設定する．なお，既に述べているように割り込み関連

の設定を行う前に割り込みのマスクを行い，設定後には割り込みのマスクを解除するのが安全である．

タイマ B1 がオーバーフローすると，割り込み要求フラグ IRR2 に 1 が設定される．このとき，IENR2 が 1 に設定されていれば，対応する割り込み処理関数が呼び出される．割り込み処理関数では，**IRR2 を必ず 0 に設定する**必要がある．割り込み処理関数で，IRR2 を 0 に設定し忘れると，割り込み処理が終了しても，割り込み要求フラグが 1 であるので，再度割り込みが発生し，永久に処理が終了せず，永久ループになってしまう．

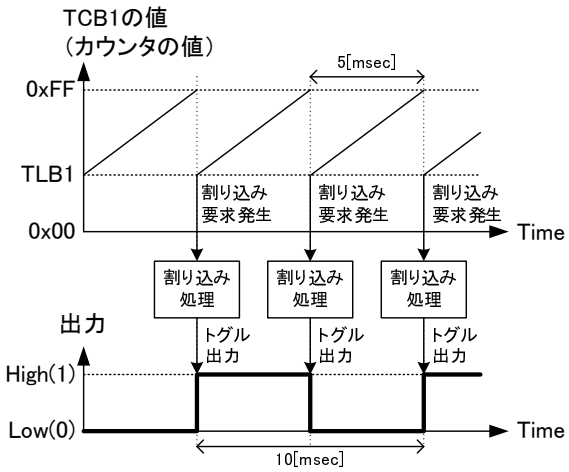


図. 3.9: タイマ B1 と割り込みを利用してブザーを鳴らす

Table 3.6: タイマ B1 の設定レジスタ (TMB1)(Renesas H8/36064 ハードウェアマニュアルより)

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	−	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。
5	−	1	−	
4	−	1	−	
3	−	1	−	
2	TMB12	0	R/W	クロックセレクト
1	TMB11	0	R/W	000：内部クロックφ/8192 でカウント
0	TMB10	0	R/W	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。

Table 3.7: タイマ B1 の割り込み関連レジスタ (Renesas H8/36064 ハードウェアマニュアルより)

(a) 割り込み許可/禁止 (IENR2)

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	−	

(b) 割り込み要求フラグ (IRR2)

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	−	

3.4.4 ソースコード

以下にサンプルプログラムのソースコードを示す．割り込み処理を利用しているので，ブザーへの出力を変更するだけでなく，LED2(緑) への出力も同時に変更することもできる．このサンプルプログラムでは，ブザーへの出力の 1/4 の周期で LED2(緑) への出力を変更している．

MonitorIntprg.c

```
/************************************************************************/
/* global variables                                                     */
/************************************************************************/
unsigned char gCnt;

/************************************************************************/
/* interrupt functions                                                  */
/************************************************************************/
void INT_NMI(void)
{ RESET(); }
void INT_TimerB1(void)
{
    IRR2.BIT.IRRTB1 = 0; /* 割り込み要求をクリア */

    IO.PDR7.BIT.B6 = !IO.PDR7.BIT.B6; /* ブザーへの出力 (P76) を反転 (トグル出力)*/

    gCnt = gCnt + 1;     /* gCnt を増加 */
    gCnt = gCnt & 0x03;  /* 2 ビットマスク */
    if( gCnt == 0 ){     /* 1/4 周期で動作 */
        IO.PDR6.BIT.B4 = !IO.PDR6.BIT.B4; /* LED2 への出力 (P64) を反転（トグル出力）*/
    }
}
```

MonitorSample.c

```
/************************************************************************/
/* main function                                                        */
/************************************************************************/
void main(void)
{
    set_imask_ccr(1);       /* 割り込み処理をマスク */

    IO.PCR6 |= 0x10;        /* IO.PCR6 の 4 ビット目を 1 にセット (P64 を出力ポートに設定) */

    TV.TCSRV.BIT.OS = 0;    /* P76 を汎用入出力ポートとして利用する */
    IO.PCR7 |= 0x40;        /* IO.PCR7 の 6 ビット目を 1 にセット (P76 を出力ポートに設定) */

    TB1.TMB1.BIT.RLD = 1;   /* オートリロード機能を利用 */
    TB1.TMB1.BIT.CKS = 2;   /* [010] 内部クロックΦ/512 でカウント */
    TB1.TLB1 = 139;         /* TLB1 を 139, 144 カウントでオーバーフローする */

    IRR2.BIT.IRRTB1 = 0;    /* TMB1 の割り込み要求をクリア */
    IENR2.BIT.IENTB1 = 1;   /* TMB1 の割り込み許可 */

    set_imask_ccr(0);       /* 割り込み処理をマスク解除 */

    for(;;){}               /* 無限ループ */
}
```

3.5　サンプリング方式で押しボタンスイッチの変化を検出するサンプルプログラム

3.2.7 節で述べたように，サンプリング方式でチャタリングを防止する場合，入力の変化を検出するためには，一サンプリング時刻前の状態をグローバル変数などで保持しておく必要がある．

以下に，サンプリング方式を利用して，押しボタンスイッチが押されたこと，および離されたことを検出し，対応する処理を行うサンプルプログラムを示す．なお，タイマ B1 の割り込みを利用しているが，タイマ B1 と割り込みの設定は適切に行われているとする．

MonitorIntprg.c

```
/************************************************************************/
/* global variables                                                     */
/************************************************************************/
unsigned char gP74Old = 0xFF; /* 一時刻前のサンプリングの P74 の状態 */

/************************************************************************/
/* interrupt functions                                                  */
/************************************************************************/
void INT_NMI(void)
{ RESET(); }
void INT_TimerB1(void)
{
    unsigned char P55;
    IRR2.BIT.IRRTB1 = 0;  /* 割り込み要求をクリア */

    P74 = IO.PDR7.BIT.B4; /* P74 の状態を取得 */
    if( gP74Old == 1 && P74 == 0 ){
        /* 押しボタンスイッチが押されたときの処理 */

    }else if( gP74Old == 0 && P74 == 1 ){
        /* 押しボタンスイッチが離されたときの処理 */

    }

    gP74Old = P74;        /* P74 の状態を保持 */
}
```

3.6 実習課題

3-1. (必修) タイマ割り込みを利用して，LED を点滅させるプログラムを作成する．☆可能であれば，LED の点灯と消灯の間隔 (デューティ比) を変更する．
3-2. (必修) サンプリング方式により，押しボタンが押されるたびに，LED の点灯パターンが変化するプログラムを作成する．LED の点灯パターンは，(LED1,LED2) が，(消灯, 消灯) → (消灯，点灯) → (点灯，消灯) → (点灯，点灯) → (消灯，消灯) を繰り返す．
3-3. (必修) 割り込み処理により，押しボタンが押されるたびに，LED の点灯パターンが変化するプログラムを作成する．割り込み後再確認方式のチャタリング防止は実装しなくても良い．ただし，サンプリング方式との違いを考察すること．
3-4. (オプション) 割り込み処理により，押しボタンが押されるたびに，LED の点灯パターンが変化するプログラムを作成する．割り込み後再確認方式のチャタリング防止を実装すること．
3-5. (オプション) 押しボタンの長押しによるリセットするプログラム

3-1(必修)	3-2(必修)	3-3(必修)	3-4(オプション)	3-5(オプション)

4　アラームタイマの作成

4.1　目標

4-1. 30 秒のアラームタイマを作成する.

4.2　アラームタイマの仕様

- はじめは待機状態．LED1 と LED2 は点灯.
- 押しボタンを押して，カウントスタート.
- 1 秒ごとに,（LED1 点灯，LED2 消灯）と (LED1 消灯，LED2 点灯) を繰り返す．(カウント状態)
- 30 秒たったら，ブザーがなり，LED1 と LED2 は点滅．(アラーム状態)
- 押しボタンを押したら，ブザーが止まり，LED1 と LED2 は点灯し，待機状態に戻る.

4.3　ステートマシン (状態遷移マシン)

押しボタンスイッチが 2 つあれば，それぞれを「スタートボタン」と「アラームストップボタン」に割り当てることができる．しかしながら，実習で利用しているマイコンボードには押しボタンスイッチが 1 つしかない．このような場合，1 つの押しボタンスイッチで,「スタートボタン」と「アラームストップボタン」の機能を果たす必要がある.

このような場合，ステートマシンという概念を利用するのが便利である．ステートマシンとは，いくつかの「状態 (ステート)」が定義されており，その「状態」が変化 (状態遷移) していくと考えるものである．例えば，同じ入力であっても，ステートマシンの (内部) 状態が異なれば，異なる動作をすることが可能である.

例えば，パソコンで利用されているローマ字での日本語入力を考えてみる．図 4.1 に示すように，初期状態は「入力待ち状態」である．この状態で，母音 (a,i,u,e,o) が入力されたならば,「あ」「い」「う」「え」「お」が出力され,「入力待ち状態」に戻る．一方，子音 (k,s,t,n,h,m,y,r,w,...) が入力されたなら

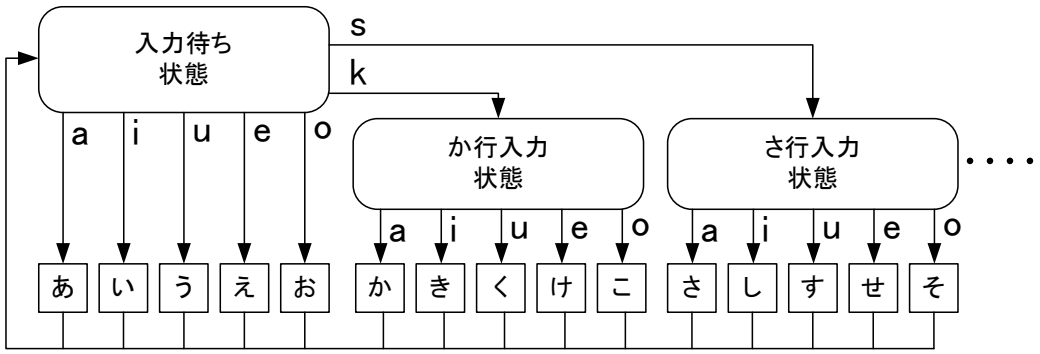


図. 4.1: ローマ字での日本語入力における状態遷移図

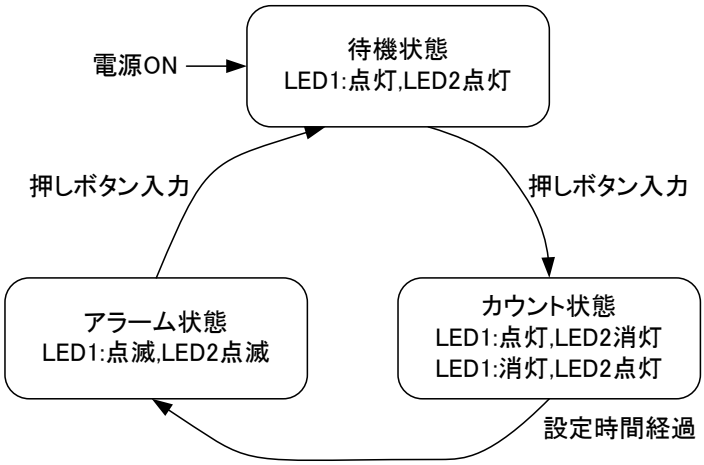


図. 4.2: アラームタイマの状態遷移図

ば，それぞれの行の入力待ち状態に状態が遷移する．例えば，k が入力された場合を考えると,「入力待ち状態」から「か行入力待ち状態」に状態が遷移する．この「か行入力待ち状態」において，母音 (a,i,u,e,o) が入力されたならば,「か」「き」「く」「け」「こ」が出力され,「入力待ち状態」に戻る．

このように，同じ入力であったとしても，ステートマシンの状態により，出力を変化させることができる．例えば，同じ a が入力されたとしても，ステートマシンの状態が「入力待ち状態」であれば「あ」が出力され,「か行入力待ち状態」であれば「か」が出力され,「さ行入力待ち状態」であれば「さ」が出力されることになる．

図 4.1 のように，ステートマシンの状態と状態遷移が表された図を状態遷移図と呼ぶ．ステートマシンの考え方に基づくことにより，限られた入力を利用して，複雑な動作を行うことが可能である．

アラームタイマの状態遷移図を図 4.2 に示す．電源が ON になった初期状態は，待機状態である．この待機状態では，押しボタンスイッチが入力されることを待つ．待機状態で，押しボタンスイッチが入力されると，カウント状態に遷移する．カウント状態では，設定時間が経過するまで，カウントを続ける．なお，このカウント状態で，押しボタンスイッチが入力されたとしても，無視される．ついで，設定時間が経過した後，アラーム状態に遷移する．アラーム状態では，ブザーを鳴らす．アラーム状態で，押しボタンスイッチが入力されると，待機状態に戻る．

このように，一連の処理は，状態の遷移とともに実行される．実際にプログラムを作成する上では，例えば，gState などといった状態を表すグローバル変数を用意し，状態を保持しておくことになる．

4.4 アラームタイマの設計例

状態遷移図 図 4.2 の状態遷移図を利用する．

ブザー タイマ V を利用する．TOMV の出力をコンペアマッチ (TCROA,TCROB) を利用して，変化させ，パルス信号をブザーに出力する．2.5 節を参照すること．

LED1 と LED2 LED1 は P60 に，LED2 は P64 に，それぞれ接続されている．P60 と P64 を，それぞれ制御することで，LED1 と LED2 の点灯/消灯を制御できる．2.4.4 節を参考にできる．

押しボタンスイッチ 押しボタンスイッチは P76 に接続されている．P76 を確認することで，押しボタンスイッチの状態を確認できる．2.4.5 節を参考にできる．

チャタリング防止 サンプリング方式によるチャタリング防止を行う．タイマ B1 を利用して，設定した周期毎に割り込みを発生させ，押しボタンスイッチの状態を確認することにする．3.2.7 節，3.4 節を参考にできる．

時間測定 チャタリング防止のために，タイマ B1 を利用して周期的に割り込みを発生させている．これを利用して，割り込みが発生する毎に，カウントを増加させ，1 秒と 30 秒を判断する．1 秒経過で LED の点灯パターンを変更し，30 秒経過で，アラーム状態に遷移する．3.4 節を参考にできる．

プログラムの構成 MonitorSample.c の main 関数では，ポートやタイマの設定を行う．MonitorIntprg.c の方に，状態を表す変数 (gStatus)，カウントを表す変数 (gCount) を，グローバル変数として用意する．主なコードはタイマ B1 のオーバーフロー割り込みに対応している INT_TimerB1 関数に記述する．

4.5 新規プロジェクトの作成と ROM 領域への書き込み

4.5.1 新規プロジェクトの作成

本実習では，与えられたプロジェクト (フォルダ) をコピーし，MonitorSample.c と MonitorIntprg.c を修正することで，プログラムを作成している．これは，モニタプログラムを利用するために必要な煩雑な処理を，受講生に意識させないようにするためである．

しかし，例えば，ROM 領域へ書き込みを行うプログラムを作成するためには，プロジェクトを新規に作成する必要がある．プロジェクトの新規生成は，プロジェクト生成ウィザードを利用することで簡単に行うことができる．

本節では，ROM 領域へ書き込みを行うための新規プロジェクトの生成手順を示す．

新規プロジェクトの生成方法:

1. 「すべてのプログラム」→「Renesas」→「High-performance Embedded Workshop」→「High-performance Embedded Workshop」で，HEW を起動する
2. ようこそ！ダイアログから,「新規プロジェクトワークスペースの作成」を選択して「OK」を押すと，プロジェクト生成ウィザードをスタートさせる．既に HEW が立ち上がっている場合は,「ファイル」→「新規ワークスペース」を選択して，プロジェクト生成ウィザードをスタートさせる．
3. 「新規プロジェクトワークスペース」ダイアログで次のように設定し,「OK」を押す.

プロジェクトタイプ	:	Application
ワークスペース名	:	app (任意の名前を設定できる)
プロジェクト名	:	app (ワークスペース名を設定すると，自動的に設定される)
ディレクトリ	:	C￥WorkSpace￥app (ワークスペース名が自動的に追加される)
CPU 種別	:	H8S,H8/300
ツールチェイン	:	Hitachi H8S,H8/300 Standard

4. 「新規プロジェクト-1/9-CPU」ダイアログで次のように設定し,「完了」を押す．1/9 であるが，以降の設定はデフォルト設定で問題ない．必要がある場合には，設定を変更すること．

ツールチェインバージョン	:	7.0.0.0
CPU シリーズ	:	300H
CPU タイプ	:	36064

   なお，CPU シリーズを選択してからでないと，CPU タイプは選択できない．
5. 「概要」ダイアログがあらわれるので,「OK」を押す.
6. 自動的に新規プロジェクトが生成される．
7. (必要があれば，) iodefine.h も自動的に生成されているので，app.c と intprg.c に#include ”iodefine.h” を追加する．なお，どこかに#include ”iodefine.h”を書くことで，Dependencies に iodefine.h があらわれる．
8. set_imask_ccr 関数を利用する場合は，app.c に#include <machine.h> を追加する．
9. (必要があれば，) WDT(WatchDog Timer) を停止する．
10. app.c に main 関数があり，intprg.c に割り込み関数が，自動的に生成されているので，適宜編集する．

4.5.2 printf，scanf とデバッガ

printf と scanf はモニタプログラムの機能として提供されている．従って，モニタプログラムを利用せず，ROM 領域にプログラムを書き込んだ場合，当然ながら，printf と scanf を利用することはできない．E8a というデバッグシステムを利用することで，ROM 領域に書き込んだプログラムをデバッグすることも可能であるが，本実習では取り扱わないことにする．

4.5.3 intprg.c にグローバル変数を宣言する際の注意

intprg.c にグローバル変数を宣言する際は，#pragma section IntPRG よりも,「前」に宣言すると良い．サンプルを以下に示す．

```
/************************************************************************/
/*                                                                      */
/*  FILE          :intprg.c                                             */
/*  DATE          :xxx, xxx xx, 20xx                                    */
/*  DESCRIPTION :Interrupt Program                                      */
/*  CPU TYPE      :H8/36064                                             */
/*                                                                      */
/*  This file is generated by Renesas Project Generator (Ver.4.9).      */
/*                                                                      */
/************************************************************************/

#include "iodefine.h" /* 追加*/
#include <machine.h>


/************************************************************************/
/* global variables                                                     */
/************************************************************************/

/* ここにグローバル変数を宣言する */


#pragma section IntPRG

//  vector 1 Reserved

//  vector 2 Reserved

//  vector 3 Reserved

//  vector 4 Reserved

//  vector 5 Reserved

//  vector 6 Reserved

/*
・
・
・
*/
```

4.5.4 書き込み方法

新規プロジェクトを作成し，プログラムが正しくコンパイルされたならば,「「H8Writer」取扱説明書」の指示に従い，H8Writer を利用して，mot ファイルを書き込む.

4.5.5 WDT(WatchDog Timer) とその停止

一種のバグとも考えられるが，ときとして，開発者の意図しない異常な動作をシステムが行う場合がある．WDT(WatchDog Timer，ウォッチドッグタイマ) は，このような異常な動作が行われた場合に，安全にシステムをリセットさせる機能を提供している.

WDT はタイマの一つであり，WDT がオーバーフローすると，システムがリセットされる仕組みになっている．そのため，システムがリセットされないようにするためには，WDT がオーバーフローする前に，WDT のカウンタを 0 にリセットする必要がある.

WDT のオーバーフローの前に，必ず WDT のタイマのカウンタを 0 にリセットするようにプログラムを作成しておく．異常動作では，WDT のタイマのカウンタが 0 にリセットされないので，WDT オーバーフローし，システムがリセットされる.

この WDT は，システムの異常動作を検出し，安全にリセットを行う仕組みとして，単純でありかつ有効である．しかしながら，WDT を利用した異常動作を検出する仕組みを利用するためには，WDT のオーバーフローの前に，必ず WDT のカウンタを 0 にリセットするようにプログラムを作成しなければならない．WDT カウンタを 0 にリセットするのを忘れてしまうと，システムが自動的にリセットされてしまう.

WDT のカウンタを 0 にリセットする処理が煩雑になってしまうため，本実習では，WDT は利用しないことにしてる．しかしながら，マイコン (H8/36064) の初期設定では，WDT が動作する設定になっている．モニタプログラムを利用していた場合，モニタプログラムが WDT を停止させていたが，ROM 領域に自作プログラムを書き込む場合は，自作プログラム内で，WDT を停止させる必要がある.

WDT の設定は，少しわかりにくいので，WDT を停止するサンプルプログラムのみを示す．詳細は「H8 36064 グループハードウェアマニュアル」(rjj09b0049_h836064.pdf) を参照．なお，以下のサンプルプログラムは，新規プロジェクトを生成したときに，自動生成されるコードに，#include ”iodefine.h” と WDT の停止を追加したものである.

```
/************************************************************************/
/*                                                                      */
/*  FILE        :app.c                                                  */
/*  DATE        :xxx, xxx xx, 20xx                                      */
/*  DESCRIPTION :Main Program                                           */
/*  CPU TYPE    :H8/36064                                               */
/*                                                                      */
/*  This file is generated by Renesas Project Generator (Ver.4.9).      */
/*                                                                      */
/************************************************************************/

#include "iodefine.h" /* 追加 */
#include <machine.h>  /* 追加 */

void main(void);
#ifdef __cplusplus
extern "C" {
void abort(void);
}
#endif

void main(void)
{
  /* WDT OFF */
  WDT.TCSRWD.BYTE=0x1E; /* TCSRWD write enable && WDON: 0 && WRST: 0 (HWM 13.2.1) */
  WDT.TCSRWD.BYTE=0x80; /* TCSRWD write disnable (HWM 13.2.1) */
  WDT.TCWD=0x00;        /* Timer Counter Reset */

}

#ifdef __cplusplus
void abort(void)
{

}
#endif
```

4.6 実習課題

4-1. (必修)30 秒のアラームタイマを作成する (モニタプログラム上で動作).
4-2. (必修) 新規プロジェクトを作成し，ROM 領域に書き込む，動作するアラームタイマを作成する (ROM 領域上で動作).
4-3. (オプション)5 秒ジャストゲームを作成する．押しボタンスイッチを一度押し，計測スタート．5 秒経過したと思ったときに，もう一度押しボタンスイッチを押し，5 秒とのずれを，printf で表示する (モニタプログラム上で動作).
4-4. (オプション) ストップウォッチを作成する．printf を利用すると作りやすい (モニタプログラム上で動作).

4-1(必修)	4-2(必修)	4-3(オプション)	4-4(オプション)

5　PWM 制御と RC サーボ

5.1　目標

5-1. モータドライバ回路の役割を理解する．
5-2. PWM 制御により，DC モータを回転させる．
5-3. RC サーボを動かす．

5.2　DC モータの PWM 制御

ここでは，図 5.1 のように接続された DC モータを PWM 制御する方法を解説する．

5.2.1　PWM(Pulse Width Modulation) 制御

マイコンのポートからの出力は，0(Low) か 1(High) の 2 値である[4]．2 値の出力により，DC モータを制御しようとした場合，回転させる (1) と停止させる (0) と 2 つの状態しか制御することができない．しかしながら，DC モータの回転数を制御したいことが多い．

このような場合，PWM(Pulse Width Modulation，パルス幅変調) を利用する．図 5.2 に PWM の例を示す．PWM とは，図 5.2 に示すように，パルス幅を変える変調方法である．パルスの幅を変えるとはすなわち，1 を出力する時間と 0 を出力する時間を変えることである．一般に，PWM では周期を一定にして，パルス幅を変更する．出力 1 が DC モータ回転に，出力 0 が DC モータ停止に，対応しているとすれば，1 を出力する時間が長いほど高出力であることが直感的にも理解できる．

例えば，パルス幅 (出力 1 の時間) が周期の 10%であれば出力 10%であり，パルス幅 (出力 1 の時間) が周期の 75%であれば出力 75%となる．周期に対するパルス幅の比は，デューティ比と呼ばれる．つまり，PWM では，出力の値は 0 と 1 の 2 値であるが，出力する時間 (パルス幅) を変更することにより，出力を連続的に変化させることができる．図 5.3 にアナログ値制御と PWM 制御の関係を示す．1

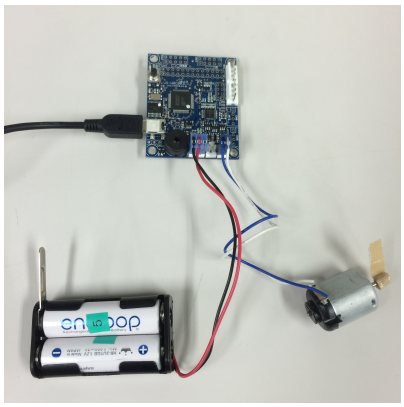


図. 5.1: DC モータと電池ボックスの接続

[4]D/A 変換を利用して，アナログ値を出力できる場合もあるが，そのような場合でも D/A 変換出力が可能なポートが非常に限られている．実際，本実習で利用している H8/36064 は，D/A 変換出力を行う機能がない．

周期分積分した結果 (面積) は，アナログ値制御と PWM 制御で等しいことが確認できる．

ところで，図 5.2 のようなパルス信号を DC モータへ入力した場合，DC モータがガタガタしてしまうと考えるかもしれない．しかし，一般的に DC モータは時定数が大きく，DC モータの入力が積分されて，出力 (回転) される．従って，DC モータの時定数に比べて，PWM の周期が十分小さければ，ある時間入力が積分された結果が，回転数となって出力される．逆に，時定数が小さいアクチュエータを利用する場合は，注意が必要である．

5.2.2 DC モータドライバ回路 (H ブリッジ回路)

マイコンのポートからの出力電流は一般に非常に小さく，LED 程度であれば直接点灯させることもできるが，DC モータなどのアクチュエータを駆動するためには，出力電流が小さすぎる．

そのため，通常は，マイコンの出力を増幅し，DC モータや各種アクチュエータに接続する．このようにモータや各種アクチュエータを駆動するための回路は駆動回路やドライバ回路と呼ばれる．

また，単純にマイコンの出力を増幅させただけでは，DC モータを一方向へ回転させることしかできない．そこで，正転と逆転ができるようにすることも，DC モータのドライバ回路の役割となる．

DC モータのドライバ回路には，図 5.4 に示すような H ブリッジと呼ばれる回路が良く用いられる．なお，図 5.4 のスイッチ (SW) は電気的に ON/OFF が可能なスイッチである．また，実際のドライバ回路では，回路を保護するためのフリーホイールダイオードなどが加えられているが，動作モードを理解することを目的としているので省略している．図 5.4 に示すように H ブリッジ回路は，モータを中心に 4 つのスイッチを組み合わせた構成になっており，その回路図が H に似ているので，H ブリッジ回路と呼ばれている．

図 5.4 に示すように 4 つのスイッチの ON/OFF を組み合わせることにより，(a) 正転モード，(b) 逆転モード，(c) ショートブレーキモード，および (d) ストップモードを実現できる．正転モードと逆転モードは，DC モータを正転および逆転させるモードである．ショートブレーキモードは，DC モータへの入力である O1 と O2 が接続 (ショート) されている．DC モータは，回転軸を回転させることで発電機としても，動作する．DC モータへの入力を接続させて，DC モータの回転軸を回転させると，

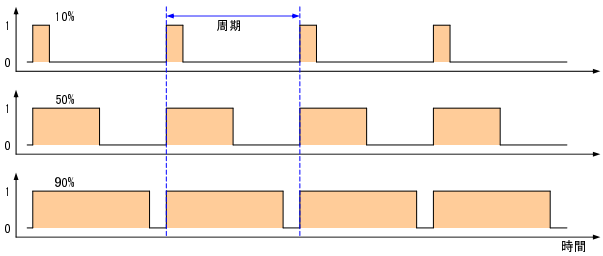


図. 5.2: PWM の例

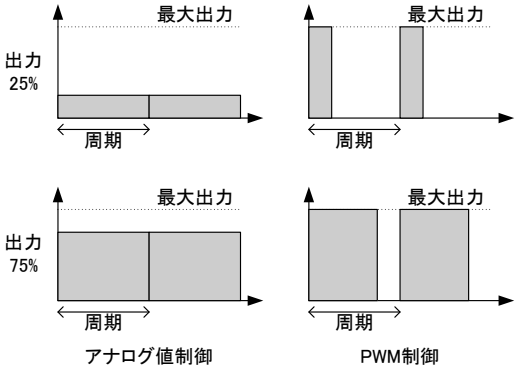
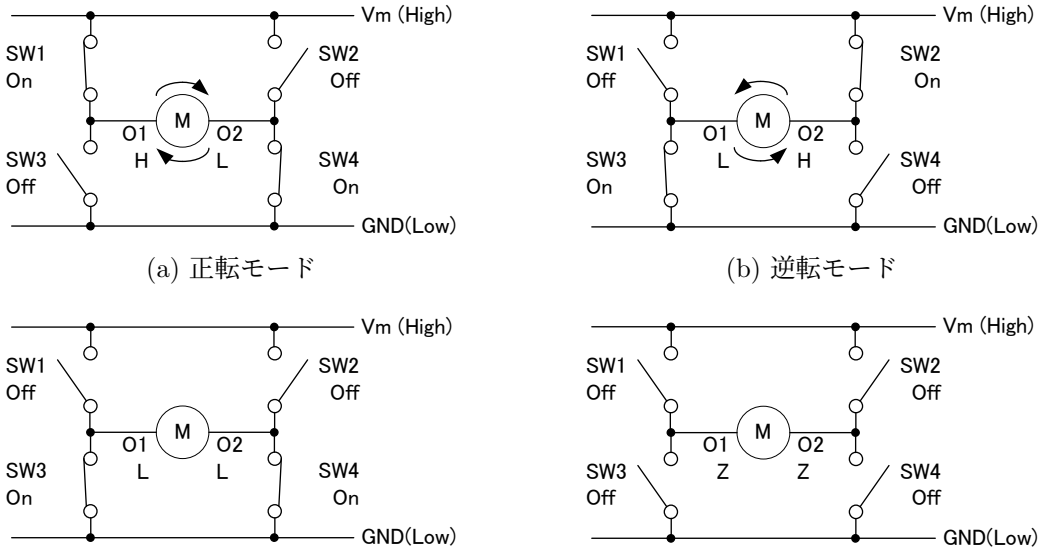


図. 5.3: アナログ値制御と PWM 制御の関係

DC モータは発電機として動作し，回転方向とは逆の方向へ回転させようとする電流が発生する．その結果として，ブレーキのような動作をすることになる．ストップモードは，DC モータへの入力がどこにも接続されていない状態である．

マイコンを利用して，PWM 制御により，DC モータを駆動させようとした場合，PWM 信号に基づき，H ブリッジ回路の 4 つのスイッチを適切に ON/OFF させる制御回路が，H ブリッジ回路とは別に必要になる．これらの，電気回路を作成することはもちろん可能ではあるが，H ブリッジ回路と制御回路がセットになっている DC モータドライバ IC が各社から市販されている．

本実習では，マイコンボードに搭載済みの DC モータドライバ IC を利用する．



(a) 正転モード　(b) 逆転モード

(c) ショートブレーキモード　(d) ストップモード

図. 5.4: H ブリッジ回路の基本モード, Z はハイインピーダンスを表す

Table 5.8: DC モータドライバの入出力関係, Z はハイインピーダンスを表す

動作	入力			出力		
目的	IN1	IN2	PWM	O1	O2	モード
正回転	H	L	H	H	L	正転
			L	L	L	ショートブレーキ
逆回転	L	H	H	L	H	逆転
			L	L	L	ショートブレーキ
ブレーキ	H	H	H	L	L	ショートブレーキ
			L			
フリー	L	L	H	Z	Z	ストップ
			L			

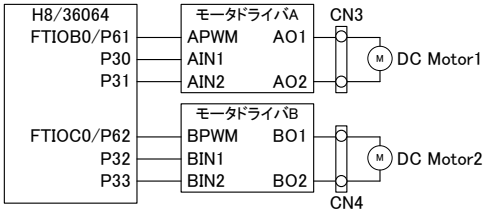


図. 5.5: マイコンとドライバ，モータの接続関係

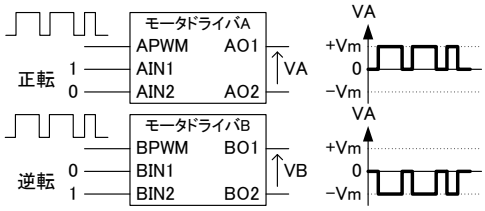


図. 5.6: PWM 信号と出力電圧の関係

5.2.3 DC モータドライバを利用した PWM 制御

マイコン (H8/36064) と DC モータドライバ回路，DC モータは図 5.5 に示すように接続されている．また，DC モータドライバ回路の入出力関係を表 5.8 に示す．

図 5.6 に示すように，動作の目的を考え，対応する値を IN1 と IN2 に設定し，所望のデューティ比で PWM 信号を出力することにより，DC モータを制御することができる．

5.3 RC サーボの制御

5.3.1 RC サーボとは

RC サーボとは，制御用の信号で回転角度を指定できるアクチュエータである．

RC サーボは，DC モータ，ギア，ポテンショメータ，制御回路，駆動回路などから構成されている．通常の DC モータであるが，ポテンショメータで回転角度検出し，所望の角度まで回転させるように制御する制御回路も搭載されている．また，DC モータの駆動回路も搭載されているので，電源をつなぎ，マイコンの出力を，制御用の信号に入力させることで，動作する．

なお演習では，図 5.7 のような接続用回路を介して，図 5.8 のように接続をおこなう．

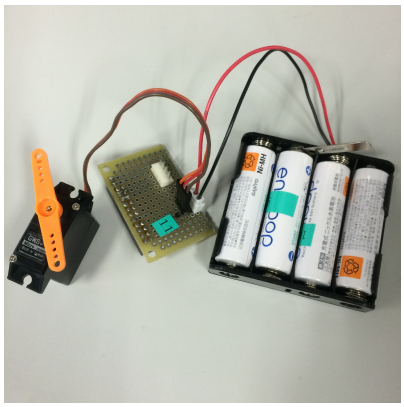


図. 5.7: RC サーボと接続用回路

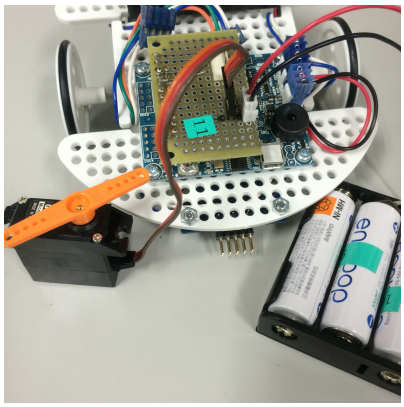


図. 5.8: RC サーボの接続

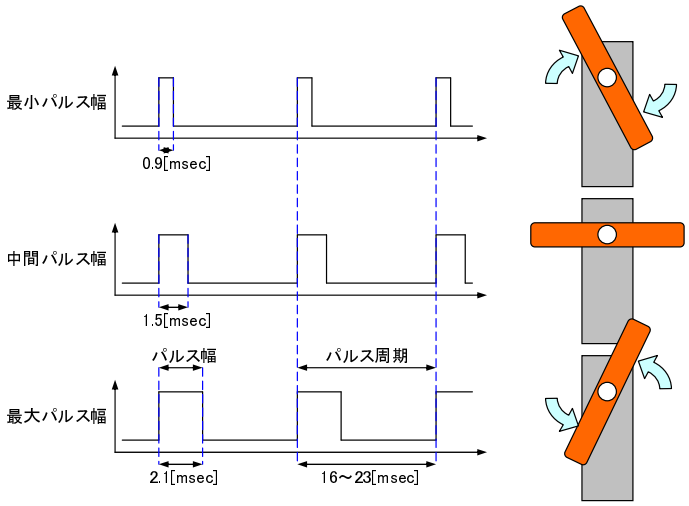


図. 5.9: 制御信号と RC サーボの回転角

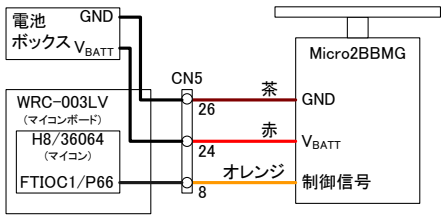


図. 5.10: マイコンボードと RC サーボの接続

5.3.2 GWS 社製 Micro2BBMG

本実習では，GWS 社製 Micro2BBMG という RC サーボを利用する．制御用の信号は，PWM 信号であり，パルス幅で RC サーボの回転角度を指定する．表 5.9 に主な仕様を示す．パルス幅 1.5[msec] が中立な位置で，このパルス幅に対して ±0.6[msec] 分だけ，パルス幅を変更することで，約 ±60[deg] 回転する．従って，0.1[msec] が 10[deg] に対応している．なお，比較的回転角度誤差が大きいので，使用に際しては注意が必要である．

PWM 信号の周期は，16～23 [msec] に設定する．周期が長すぎると，反応が遅くなる．また，逆に短すぎると，RC サーボが動作しない．

マイコンボードとは，拡張コネクタ CN5 を介して，図 5.10 のように接続することにする．配布されたキットには，既に図 5.10 のように接続されている．

5.4 タイマ Z

H8/36064 に搭載されている 2 つのタイマ Z を利用して PWM 信号を生成する．

Table 5.9: GWS 社製 Micro2BBMG の主な仕様

位置	パルス幅	角度
最小位置	0.9[msec]	約－ 60[deg]
中立位置	1.5[msec]	0[deg]
最大位置	2.1[msec]	約＋ 60[deg]

パルス周期	16 ～ 23 [msec]
駆動電圧	4.8 ～ 7.5 [V]
回転速度	0.17 [sec] / 60[deg]
トルク	5.4 [kg cm]

5.4.1 タイマ Z 概要

H8/36064 には，タイマ Z0 とタイマ Z1 の 2 チャンネル (2 つ) 搭載されている．2 つのタイマの機能は，同じであり，独立に利用することができる．また，同期させることも可能である．

タイマ Z は 16 ビットタイマであり，16 進数で表すと 0x0000 から 0xFFFF まで，10 進数で表すと 0 から 65535 まで，カウント可能である．

各タイマに 4 つのジェネラルレジスタ (GRA, GRB, GRC, GRD) が搭載されており，コンペアマッチ/インプットキャプチャに利用可能である．各ジェネラルレジスタは，それぞれ入出力ポートに対応しており，最大 8 種類の入出力を制御することができる．表 5.10 にタイマ Z のジェネラルレジスタとポートの対応関係を示す．また，各ジェネラルレジスタの値で，カウンタをクリアすることもできる．コンペアマッチ/インプットキャプチャ，およびオーバーフロー割り込みが可能である．

なお，いくつかのモードが用意されているが，本実習では，通常動作のみを考慮する．

5.4.2 タイマ Z を用いた PWM 生成

3.3.3 節で述べたように，タイマ V では，2 つのタイムコンスタントレジスタ (TCROA,TCROB) を利用して，1 つの出力ポートを制御していた．そのため，TCROA とマッチしたときに出力 1，TCROB とマッチしたときに出力 0 とすることで，パルスを生成することができた．

一方，タイマ Z では，表 5.10 に示すように，1 つのジェネラルレジスタが 1 つの出力ポートに対応している．このため，コンペアマッチによる出力制御のみでは，パルスを生成することができない．そこで，割り込み処理を利用することを考える．ジェネラルレジスタとのコンペアマッチによっても割り込みを発生させることもできるが，割り込みの発生は可能な限り少ない方が良い．そこで，コンペアマッチによる出力制御と，オーバーフロー割り込みを併用することにする．

具体的には，オーバーフロー割り込み処理で，出力を 1 にセットし，ジェネラルレジスタとのコンペアマッチにより，出力を 0 にクリアする．このように動作させることにより，1 度の割り込み処理で任意の幅のパルスを生成することができる．また，ジェネラルレジスタの値がパルス幅に比例する．図 5.11 にパルス生成の動作概要を示す．

なお，ジェネラルレジスタとのコンペアマッチにより出力を 1 にセットし，オーバーフロー割り込み処理で出力を 0 にクリアすることもでき，一般的にはこのようにする場合が多い．しかしながら，この場合，ジェネラルレジスタの値がパルス幅に比例しないので，直感的に分かりにくい．

Table 5.10: タイマ Z のジェネラルレジスタとポートの対応関係

タイマ Z0	
GRA0	P60/FTIOA0
GRB0	P61/FTIOB0
GRC0	P62/FTIOC0
GRD0	P63/FTIOD0

タイマ Z1	
GRA1	P64/FTIOA1
GRB1	P65/FTIOB1
GRC1	P66/FTIOC1
GRD1	P67/FTIOD1

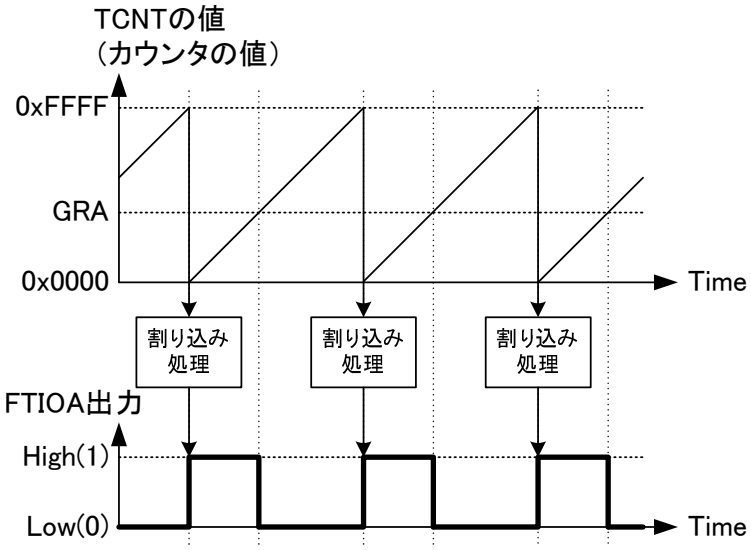


図. 5.11: コンペアマッチと割り込みを併用したパルス生成

5.4.3 タイマ Z の設定

タイマ Z0 を利用して，DC モータを制御する場合の設定を考える．

タイマモードの設定 (TMDR,TPMR,TFCR) タイマモードの設定のためには，TMDR,TPMR, および TFCR の設定を行う必要がある．ただし，通常動作の場合は，全ての初期値のままの良いため，改めて設定を行う必要はない．

カウンタクロックの選択 (TCR) カウンタクロックを選択するためには，TCR レジスタの TPSC を設定する．タイマ Z をΦ/4 でカウントすることにする場合は，$\boxed{\text{TPSC2}}\boxed{\text{TPSC1}}\boxed{\text{TPSC0}} = \boxed{0}\boxed{1}\boxed{0}$と設定すればよい．

カウンタクリア条件の設定 (TCR) カウンタクリア条件を設定するためには，TCR レジスタの CCLR を設定する．オーバーフローによるクリアのみをする場合は，$\boxed{\text{CCLR2}}\boxed{\text{CCLR1}}\boxed{\text{CCLR0}} = \boxed{0}\boxed{0}\boxed{0}$ と設定すればよい．

割り込みの設定 (TIER,TSR) 割り込み許可の設定は TIER レジスタで行う．また，割り込みフラグは TSR である．オーバーフロー割り込みを許可する場合は，$\boxed{\text{OVIE}} = \boxed{1}$ とする．割り込み処理で，TSR レジスタの割り込みフラグは OVF を 0 にクリアすることを忘れないこと．

出力モードの設定 (TOER) P61/FTIOB0 と P62/FTIOC0 を，コンペアマッチで制御する必要がある．コンペアマッチで出力の制御を可能にするためには TOER レジスタを設定する．P61/FTIOB0 と P62/FTIOC0 を制御可能にする場合は，$\boxed{\text{EB0}} = \boxed{0}$ および $\boxed{\text{EC0}} = \boxed{0}$ と設定すればよい．

コンペアマッチ時の出力変化 (TIORA, TIORC) コンペアマッチ時の出力変化を設定するためには，TIORA および TIORC を設定する．FTIOA と FTIOB の出力変化の設定には TIORA を，

FTIOC と FTIOD の出力変化の設定には TIORC を，それぞれ設定する．FTIOB0 をコンペアマッチ時に出力 0 とする場合は，IOB2 | IOB1 | IOB0 = 0 | 0 | 1 と設定すればよい．また，FTIOC0 をコンペアマッチ時に出力 0 とする場合は，IOC2 | IOC1 | IOC0 = 0 | 0 | 1 と設定すればよい．

ジェネラルレジスタの設定 (GRA,GRB,GRC,GRD) ジェネラルレジスタ GRB0 は出力 FTIOB0 のデューティ比を，ジェネラルレジスタ GRC0 は出力 FTIOC0 のデューティ比を，それぞれ表す．従って，所望のデューティ比に対応する値を，それぞれ設定する．なお，デューティ比は，GRB0 または GRC0 の値と，カウンタの周期 0xFFFF の比と等しい．

タイマのスタート/ストップ (TSTR) タイマ Z0 をスタートさせるためには，TSTR レジスタの STR0 を 1 に設定する必要がある．逆に，TSTR レジスタの STR0 を 0 に設定すれば，タイマはストップする．

5.4.4 タイマ Z のオーバーフロー割り込み処理

パルス生成に必要な，オーバーフロー時の割り込み処理は，以下の二つである．

割り込みフラグを 0 にクリアする (TSR) TSR レジスタの割り込みフラグは OVF を 0 にクリアする．

出力を 1 にセットする (TOCR) 出力を制御するためには，TOCR レジスタを設定する．FTIOB0 の出力を 1 にする場合は，TOB0 = 1 と設定すればよい．また，FTIOC0 の出力を 1 にする場合は，TOC0 = 1 と設定すればよい．

5.4.5 タイマ Z の割り込み使用時の注意

プログラムをロードして，実行しても，すぐにモニタプログラムのプロンプトに戻ってしまうことがある．その場合は，あせらずに，もう 1 度，プログラムをロードして，実行を試みる．どういうわけか，タイマ Z の割り込みを利用した場合，モニタプログラムのプロンプトに戻ってしまうことがある．2 度繰り返すことで，問題なく実行できる．

3 度，4 度と繰り返しても実行できない場合は，スタッフに相談すること．

5.5 実習課題

5-1. (必修)PWM 信号により，DC モータを回転させる．
5-2. (必修)PWM 信号により，RC サーボを制御する．
5-3. (オプション)DC モータを制御する．タイマと組み合わせる，押しボタンスイッチと組み合わせる，scanf と組み合わせるなど，いろいろと DC モータを制御してみる．

5-1(必修)	5-2(必修)	5-3(オプション)

6　アナログ入力とエンコーダ

6.1　目標

6-1. アナログ入力の値を読み取る．
6-2. エンコーダを利用して，パルスをカウントする．

6.2　アナログ入力

6.2.1　可変抵抗

可変抵抗とは，抵抗値を変えることができる抵抗である．回転軸を回すと抵抗値が変化する．本実習では，可変抵抗は，マイコンと図 6.1 に示すとおりに接続することにする．回転軸を回すことにより，② の位置が変化し，結果として抵抗値が変化する．

マイコンへの入力電圧を計算するため，図 6.1 の回路を図 6.2 のように書き換える．マイコンへの入力電圧は次式のように計算できる．

$$V = \frac{R_1 + 680}{\frac{33000 \times R_0}{33000 + R_0} + R_1 + 680} V_{cc} \tag{6.1}$$

ここで，R_0 と R_1 は可変抵抗の回転軸の位置に定まる抵抗値を表す．

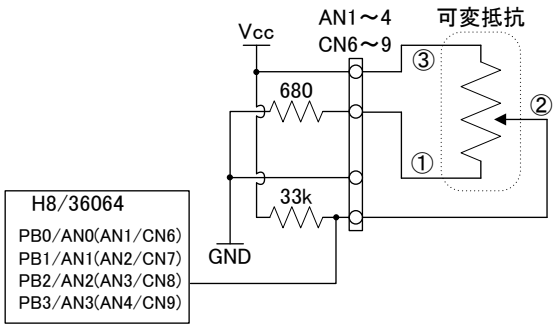


図. 6.1: 可変抵抗とその接続

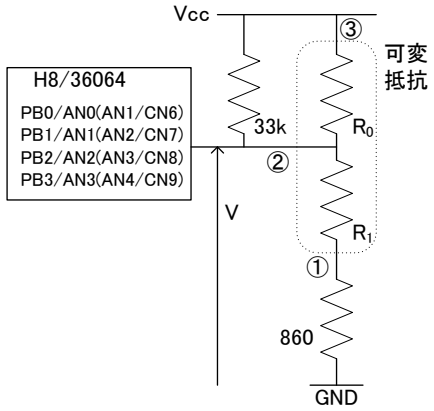


図. 6.2: 電圧の計算

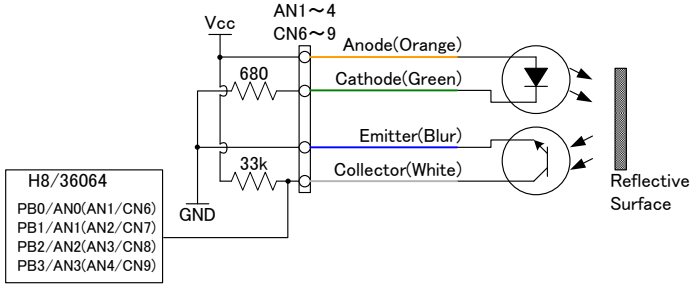


図. 6.3: 赤外線センサとその接続

6.2.2 赤外線センサ

本実習では，Vstone 社から発売されている赤外線センサを利用し，アナログ入力の学習を行う．

図 6.3 に示すように，赤外線センサは赤外線 LED と，赤外線に反応するフォトトランジスタから構成されている．また，図 6.3 に示すようにマイコンボードおよびマイコンに接続されているので，マイコンボードの電源をいれるだけで，赤外線 LED は発光を開始する．ただし，赤外線であるので発光しているかどうかは確認できない．携帯電話のカメラなどで撮影すると赤外線の発光を確認することができる[5]．

赤外線 LED で発光し，反射体で反射した光を，フォトトランジスタで測定する仕組みになっている．フォトトランジスタは，光の強さが強いほどに応じて多くの電流が流れる．

一般に，このような光を利用したセンサを利用する場合は，外乱の影響を避けるため変復調が利用される．例えば，センサではないが家電製品のリモコンでは，赤外線を 38[KHz] で変復調して利用している．しかし，本実習で利用するマイコンボード (WRC-003LV) との接続では変復調が利用されていないので，注意が必要である．

反射体と赤外線センサとの距離に応じて，フォトトランジスタが受ける光の強さが変わることを利用して，簡易的な距離センサとして利用することが可能である．ただし，反射体の反射率や形状，向きによっても，フォトトランジスタが受ける光の強さが変化するので，注意が必要である．

反射体と赤外線センサとの距離が同じであっても，反射体の反射率が異なれば，フォトトランジスタが受ける光の強さも異なる．この性質を利用して，白地に書かれた黒い線を検出することなどもできる．

6.2.3 A/D 変換器

A/D 変換器は，アナログ量をディジタル量に変換する電気回路である．AD コンバータ，ADC とも呼ばれる．逐次比較形，デルタシグマ形など，いくつもの方法が知られている．いずれにしても，入力のアナログ信号をディジタル化する方法である．

図 6.4 に A/D 変換器の概要を示す．図 6.4 に示すように，入力電圧 Vin がが A/D 変換器によって，4 ビットのディジタル量に変換されていることが確認される．当然のことではあるが，A/D 変換を行うためには，ある程度の時間が必要であり，その間は入力の電圧を一定に保つ必要がある．このようにある時間間隔電圧を一定に保つ機能は，サンプル・ホールド機能と呼ばれる．

6.2.4 H8/36064 周辺機能 A/D 変換器の特徴

H8/36064 に搭載されている A/D 変換器の特徴を以下にまとめる．

- 10 ビット分解能.
- 8 チャンネル入力 (PB0/AN0 から PB7/AN7 までの 8 チャンネル) .

[5]多くの携帯のカメラでは赤外線領域にも感度があり，強い赤外線に反応することが多い.

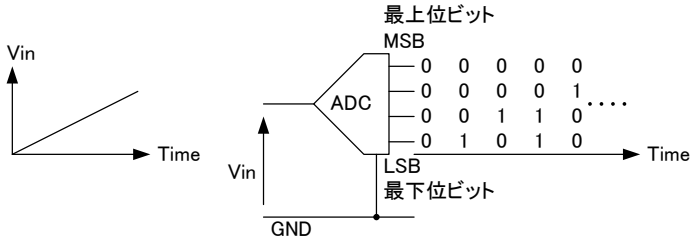


図. 6.4: A/D 変換器の概要

- 変換時間，約 6[μ sec][6].
- 1 チャンネルのみを A/D 変換する単一モードと，複数チャンネルを連続 A/D 変換するスキャンモードが，利用可能.
- サンプル・ホールド機能付き.
- ソフトウエアによる A/D 変換のみならず，外部トリガ信号により A/D 変換を開始することも可能.
- A/D 変換終了時に割り込みを発生させることが可能.

6.2.5 A/D 変換の動作と設定

単一モードとスキャンモードの基本的な使い方を説明する．ここで，述べる使い方以外にも，工夫次第で様々な使い方が可能である.

6.2.5.1 データレジスタについて

表 6.11 に示すように，A/D 変換結果を格納するレジスタは，二つのチャネルで共有するようになっている．そのため，データを読み出すときなどには注意が必要である.

また，4 つデータレジスタ (ADDRA, ADDRB, ADDRC, および ADDRD) は，16 ビットデータ (unsigned int) として定義されている．しかしながら，H8/36064 に搭載されている A/D 変換器は 10 ビットである．そのため，表 6.12 に示すように，データレジスタの下位 4 ビットは必ず 0 が読み出される．このため，データを読み出した後に，4 ビットシフト (C 言語では，“>> 4”) するなどすれば良い．また，A/D 変換器の下位のビットはノイズの影響などにより，それほど信頼性の高いものではない．そのため，10 ビット A/D 変換器ではあるものの，上位 8 ビットのみを利用することも多い.

[6] ハードウェアマニュアルの 3.5[μ sec](20 [MHz] 動作時) から動作周波数で換算.

Table 6.11: A/D 変換のチャネルとデータレジスタの関係

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

Table 6.12: A/D 変換のデータレジスタとビットの並び

データレジスタ	B_{15}	B_{14}	B_{13}	B_{12}	B_{11}	B_{10}	B_9	B_8	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0
A/D 変換結果	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	0	0	0	0	0	0

6.2.5.2 単一モード

単一モードでは，一つのチャネルを必要なときに一度だけ，A/D 変換する場合に便利である．基本的な利用方法と，関連するレジスタの設定を以下に示す．

1. **単一モードに設定する．** ADCSR レジスタの SCAN を 0 に設定する．
2. **A/D 変換を行うチャネル (ポート) を選択する．** ADCSR レジスタの CH にチャネルを設定する．
3. **A/D 変換を開始する．** ADCSR レジスタの ADST を 1 に設定する．
4. **A/D 変換が終了するまで待つ．** ADCSR レジスタの ADST が 1 の間，ループして待つ．
5. **A/D 変換の結果を読み出す．** ADDRA, ADDRB, ADDRC または ADDRD の内から，A/D 変換を行ったチャネルに対応したデータレジスタからデータを読み出す．
6. **上位ビットを取り出す．** 必要に応じて，上位ビットのみを取り出す．

なお，A/D 変換のレジスタの詳細な設定は,「H8 36064 グループハードウェアマニュアル」(rjj09b0049_h836064.pdf) を参照．

6.2.5.3 スキャンモード

スキャンモードは，指定したチャネルを逐次的に常に A/D 変換を行う．常に A/D 変換を行っているので，データレジスタを読み出すだけで，最後に行った A/D 変換の結果を読み出すことができる．

この方法は，簡便な方法であるが，常に A/D 変換を行っているので，消費電力が大きい．また，データレジスタから読み出す値が，常に過去のデータであり，いつ A/D 変換が行われたかが明確でないという欠点がある．

基本的な利用方法と，関連するレジスタの設定を以下に示す．

1. **スキャンモードに設定する．**ADCSR レジスタの SCAN を 1 に設定する．
2. **A/D 変換を行うチャネル (ポート) を選択する．**ADCSR レジスタの CH にチャネルを設定する．
3. **A/D 変換を開始する．**ADCSR レジスタの ADST を 1 に設定する．
4. **データを読み出す．**必要なときに，データレジスタからデータを読み出す．スキャンモードでは，A/D 変換が常に行われているので，最後に A/D 変換が行われたときの，値を読み出すことになる．

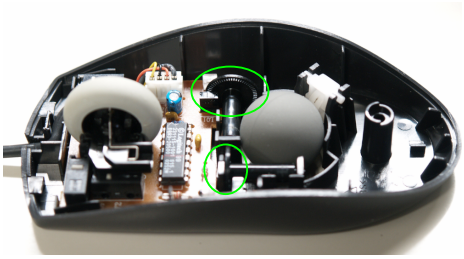


図. 6.5: ボール式マウスの中のロータリエンコーダ

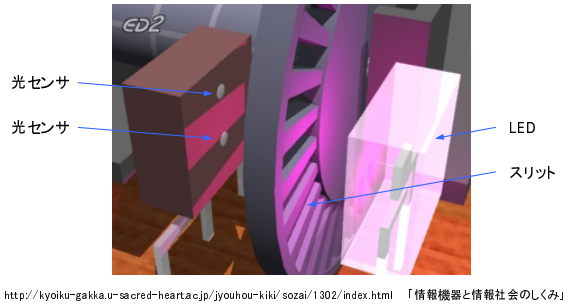


図. 6.6: ロータリーエンコーダの構成例

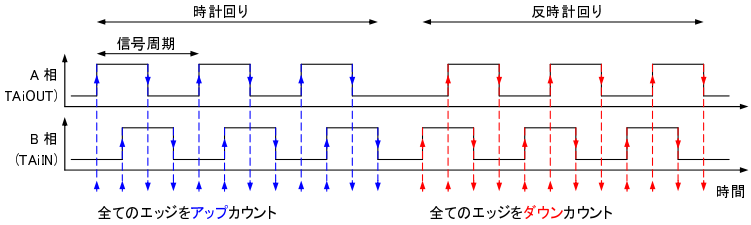


図. 6.7: ロータリーエンコーダの A 相/B 相出力と回転方向識別

5. **上位ビットを取り出す.** 必要に応じて，上位ビットのみを取り出す.
6. **A/D 変換を終了する.** A/D 変換の必要がないときは，ADCSR レジスタの ADST を 0 に設定し，A/D 変換を行わない.

なお，A/D 変換のレジスタの詳細な設定は,「H8 36064 グループハードウェアマニュアル」(rjj09b0049_h836064.pdf) を参照.

6.3 ロータリエンコーダ

6.3.1 ロータリエンコーダとは

ロータリエンコーダとは，回転角度と回転方向を検出することができるセンサ (機器) である．図 6.5 に示すようにボール式のマウスにも利用されている．ロータリエンコーダは，図 6.6 に示すように，穴の空いた円盤と，LED および光センサから，構成されている.

穴の空いた円盤を回転させる．LED から出る光は，穴がある場合は，光センサに到達し，穴がない箇所では，光は遮られ，光センサには到達しない.

このような簡単な仕組みにより，穴の空いた円盤の回転角度に応じたパルスが，光センサから出力されることになる.

ところで，光センサが一つしかない場合は，回転の方向を判別することができない．そこで，2 個のセンサを利用し，回転角度と回転方向を判別可能にしている．2 個のセンサの出力は，A 相/B 相と

Table 6.13: A 相/B 相と 1,2,4 逓倍の関係

A 相	立ち上がり↑		立ち下がり↓		High(1)	Low(0)	High(1)	Low(0)
B 相	High(1)	Low(0)	High(1)	Low(0)	立ち上がり↑		立ち下がり↓	
回転方向	逆 (-)	正 (+)	正 (+)	逆 (-)	正 (+)	逆 (-)	逆 (-)	正 (+)
1 逓倍	○カウント		×なし		×なし		×なし	
2 逓倍	○カウント		○カウント		×なし		×なし	
4 逓倍	○カウント		○カウント		○カウント		○カウント	

呼ばれ，図 6.7 に示すように，回転スリットの周期 (穴の空いた円盤の穴と穴の間隔) よりも，センサの間隔が狭くなるように配置されている．もう少し具体的には，1/4 周期だけずれて配置されている．このような配置することにより，回転角度のみならず，回転の方向が判別可能になる．

時計回りのときは，A 相の立ち上がりエッジが生じるときは B 相の出力は必ず Low(0) レベルである．一方，反時計回りのときは，A 相の立ち上がりエッジが生じているときは B 相の出力は必ず High(1) レベルである．このように，2 個のセンサの出力を組み合わせることで，回転角度と回転方向を判別が可能である．

実習で利用するロータリエンコーダは，円周に白と黒のパターンがあり，このパターンを計測することによりパルスを発生させている．

6.3.2 1,2,4 逓倍 (ていばい) カウント

図 6.7 の A 相の立ち上がりエッジのみに着目して，回転角度と回転方向を検出することができる．実習で利用するロータリーエンコーダは一回転で 5 パルス出力するので，72[deg] の分解能で，検出することができる．

図 6.7 からもわかるように，A 相の立ち上がりエッジのみならず，A 相の立ち下がりエッジを利用することで分解能を向上させることができる．さらに，B 相のエッジを利用すると，さらに分解能が向上する．

A 相の立ち上がりエッジのみをカウントする方法は，1 逓倍カウントと呼ばれ，分解能は 72[deg] である．A 相の立ち上がりエッジと立ち下がりエッジをカウントする方法は，2 逓倍カウントと呼ばれ，分解能は 1 逓倍カウントの 2 倍の 36[deg] である．さらに，A 相の立ち上がりエッジと立ち下がりエッジおよび B 相の立ち上がりエッジと立ち下がりエッジをカウントする方法は，4 逓倍カウントと呼ばれ，分解能は 1 逓倍カウントの 4 倍の 18[deg] となる．表 6.13 に，1,2 および 4 逓倍カウントと，エッジの関係をまとめる．

同じロータリーエンコーダを利用しても，カウント方法を変更するだけで，分解能が異なる．

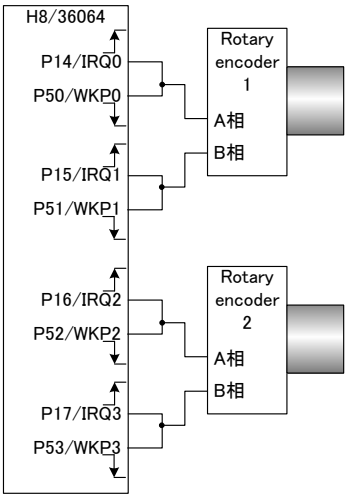


図. 6.8: ロータリエンコーダ 2 つを 4 逓倍カウントする場合の接続例

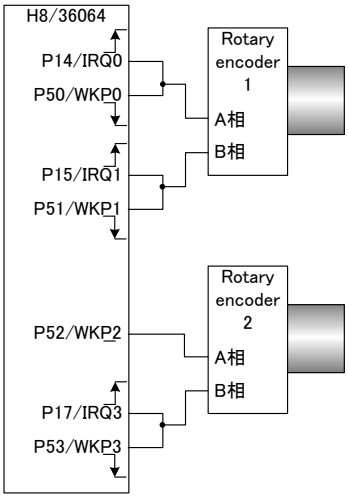


図. 6.9: モニタプログラムの IRQ2 のバグを避けるための接続例

6.3.3 マイコンとの接続

A 相と B 相が 1/4 周期ずれたロータリーエンコーダは広く普及しているため，タイマと組合せ，A 相 B 相を特定のポートとつなぐことで，タイマを応用し，4 逓倍カウントで，カウントする機能を有するマイコンも，存在する．

しかしながら，本実習で利用しているマイコン (H8/36064) にはそのような機能も存在しない．また，4 逓倍カウントを利用するためには，立ち上がりと立ち下がりの両エッジを検出する必要がある．エッジの検出には割り込み処理を利用可能であるが，マイコン (H8/36064) では両エッジを検出可能なポートが存在しない．

このような理由により，ロータリエンコーダをマイコン (H8/36064) に接続するためには少しの工夫が必要となる．

マイコン (H8/36064) で利用可能な外部割り込みが可能なポートは，P14/IRQ0, P15/IRQ1, P16/IRQ2, P17/IRQ3, P50/WKP0, P51/WKP1, P52/WKP2, P53/WKP3, P54/WKP4, および P55/WKP5 の 10 ポートである[7]．

これらのポートは，立ち上がりエッジで割り込み要求を発生させるか，立ち下がりエッジで割り込み要求を発生させるか，を選択することは可能である．しかし，両エッジで割り込み要求を発生させることはできない．

このような制約を考慮して，図 6.8 に示すようにマイコンと接続されている．

[7]NMI も外部割り込みの 1 つであるが，通常は非常用に利用されるため，通常の利用は行えないものと考える．

6.3.3.1 ロータリエンコーダを 4 逓倍カウント

図 6.8 に，ロータリエンコーダを 4 逓倍カウントで使用するための接続例を示す．4 逓倍カウントで使用するためには，A 相と B 相ともに両エッジを検出する必要がある．マイコン (H8/36064) で A 相の両エッジを検出するために，A 相の出力を P14/IRQ0 と P50/WKP0 の 2 つのポートに接続し，P14/IRQ0 で立ち上がりエッジを，P50/WKP0 で立ち下がりエッジを，それぞれ検出することにする．また，B 相の出力についても，同様に P15/IRQ1 と P51/WKP1 に接続し，2 つのポートを利用して，両エッジを検出することにする．

ロータリエンコーダ 2 についても同様である．

6.3.3.2 モニタプログラムの IRQ2 に対するバグ

モニタプログラムに IRQ2 の割り込みに関するバグがあるようである．そのため，モニタプログラムを利用して，図 6.8 のように 2 つのエンコーダを 4 逓倍カウントで利用することはできない．このバグを避けるため，図 6.9 のように接続されていると考え，ロータリエンコーダ 2 を 2 逓倍カウントとして利用することは可能である．

なお，このバグはモニタプログラムに起因するバグのようであるため，ROM 焼きの場合は，図 6.8 と考え，2 つのエンコーダを 4 逓倍カウントで利用することができる．

6.3.4 サンプルプログラム

図 6.8 に示すような接続をした場のロータリエンコーダ 1 に対するサンプルプログラムを以下に示す.

MonitorIntprg.c

```
/************************************************************************/
/* global variables                                                     */
/************************************************************************/
unsigned int gEncCnt1;
unsigned int gEncCnt2;


/************************************************************************/
/* encoder function                                                     */
/************************************************************************/
void encoder(void)
{
    /* A 相: P14(IRQ0) 立ち上がり，P50(WKP0) 立ち下がり
       B 相: P15(IRQ1) 立ち上がり，P51(WKP1) 立ち下がり */

    if( IRR1.BIT.IRRI0 == 1 ){
        /* A 相立ち上がり処理 */
        if( IO.PDR5.BIT.B1 == 0 ){
            gEncCnt1++;
        }else{
            gEncCnt1--;
        }
        IRR1.BIT.IRRI0 = 0; /* 割り込み要求フラグクリア */
    }
    else if( IWPR.BIT.IWPF0 == 1 ){
        /* A 相立ち下がり処理 */
        if( IO.PDR5.BIT.B1 == 0 ){
            gEncCnt1--;
        }else{
            gEncCnt1++;
        }
        IWPR.BIT.IWPF0 = 0; /* 割り込み要求フラグクリア */
    }
    else if( IRR1.BIT.IRRI1 == 1 ){
        /* B 相立ち上がり処理 */
        if( IO.PDR5.BIT.B0 == 0 ){
            gEncCnt1--;
        }else{
            gEncCnt1++;
        }
        IRR1.BIT.IRRI1 = 0; /* 割り込み要求フラグクリア */
    }
    else if( IWPR.BIT.IWPF1 == 1 ){
        /* B 相立ち下がり処理 */
        if( IO.PDR5.BIT.B0 == 0 ){
            gEncCnt1++;
        }else{
            gEncCnt1--;
        }
        IWPR.BIT.IWPF1 = 0; /* 割り込み要求フラグクリア */
    }
}

/************************************************************************/
```

```
/* interrupt functions                                                    */
/************************************************************************/
void INT_IRQ0(void)
{ encoder(); }
void INT_IRQ1(void)
{ encoder(); }
void INT_IRQ2(void)
{ encoder(); }
void INT_IRQ3(void)
{ encoder(); }
void INT_WKP(void)
{ encoder(); }
```

MonitorSample.c

```
/*********************************************************************/
/* external variables                                                */
/*********************************************************************/
extern unsigned int gEncCnt1; /* 外部変数宣言 */
extern unsigned int gEncCnt2; /* 外部変数宣言 */

/*********************************************************************/
/* main function                                                     */
/*********************************************************************/
void main(void)
{
    set_imask_ccr(1);       /* 割り込み処理をマスク */

    gEncCnt1 = 0;           /* カウント値を初期化 */
    gEncCnt2 = 0;           /* カウント値を初期化 */


    IO.PMR1.BIT.IRQ0 = 1;   /* P14/IRQ0: IRQ 入力として利用 */
    IO.PMR1.BIT.IRQ1 = 1;   /* P15/IRQ1: IRQ 入力として利用 */

    IO.PMR5.BIT.WKP0 = 1;   /* P50/WKP0: WKP 入力として利用 */
    IO.PMR5.BIT.WKP1 = 1;   /* P51/WKP1: WKP 入力として利用 */

    IEGR1.BIT.IEG0 = 1;     /* IRQ0:立ち上がりエッジ検出 */
    IEGR1.BIT.IEG1 = 1;     /* IRQ1:立ち上がりエッジ検出 */

    IEGR2.BIT.WPEG0 = 0;    /* WKP0:立ち下がりエッジ検出 */
    IEGR2.BIT.WPEG1 = 0;    /* WKP1:立ち下がりエッジ検出 */

    IRR1.BIT.IRRI0 = 0;     /* IRQ0 の割り込み要求フラグクリア */
    IRR1.BIT.IRRI1 = 0;     /* IRQ1 の割り込み要求フラグクリア */

    IWPR.BIT.IWPF0 = 0;     /* WKP0 の割り込み要求フラグをクリア */
    IWPR.BIT.IWPF1 = 0;     /* WKP1 の割り込み要求フラグをクリア */


    IENR1.BIT.IEN0 = 1;     /* IRQ0 の割り込み許可 */
    IENR1.BIT.IEN1 = 1;     /* IRQ1 の割り込み許可 */
    IENR1.BIT.IENWP = 1;    /* WKP0～WKP5 の割り込み許可 */

    IRR2.BIT.IRRTB1 = 0;    /* TMB1 の割り込み要求をクリア */
    IENR2.BIT.IENTB1 = 1;   /* TMB1 の割り込み許可 */

    set_imask_ccr(0);       /* 割り込み処理をマスク解除 */

    for(;;){}               /* 無限ループ */
}
```

6.4 実習課題

6-1. (必修)AD 変換の結果を printf 出力するプログラムを作成する．なお，AD 変換に際してはスキャンモードを利用すること．
6-2. (必修) ロータリエンコーダの回転方向を判別して，LED を点灯させる．
6-3. (必修) ロータリエンコーダの A 相の出力を LED1 に，B 相の出力を LED2 に出力し，一定時間間隔でロータリエンコーダのカウントを printf で出力するプログラムを作成する．なお，4 逓倍カウントを行うこと．
6-4. (オプション)AD 変換の結果に応じて，PWM のデューティ比を変化させ，DC モータを制御する．
6-5. (オプション) ロータリエンコーダを利用して，一定回転した後に停止するなど，DC モータを制御する．

6-1(必修)	6-2(必修)	6-3(必修)	6-4(オプション)	6-5(オプション)

付録:レジスタ・変数名対応表

タイマZ0

レジスタ名	バイトアクセス変数名	ビットアクセス変数名
TCR	TZ0.TCR.BYTE	TZ0.TCR.BIT.CCLR
		TZ0.TCR.BIT.CKEG
		TZ0.TCR.BIT.TPSC
TIORA	TZ0.TIORA.BYTE	TZ0.TIORA.BIT.IOB
		TZ0.TIORA.BIT.IOA
TIORC	TZ0.TIORC.BYTE	TZ0.TIORC.BIT.IOD
		TZ0.TIORC.BIT.IOC
TSR	TZ0.TSR.BYTE	TZ0.TSR.BIT.OVF
		TZ0.TSR.BIT.IMFD
		TZ0.TSR.BIT.IMFC
		TZ0.TSR.BIT.IMFB
		TZ0.TSR.BIT.IMFA
TIER	TZ0.TIER.BYTE	TZ0.TIER.BIT.OVIE
		TZ0.TIER.BIT.IMIED
		TZ0.TIER.BIT.IMIEC
		TZ0.TIER.BIT.IMIEB
		TZ0.TIER.BIT.IMIEA
POCR	TZ0.POCR.BYTE	TZ0.POCR.BIT.POLD
		TZ0.POCR.BIT.POLC
		TZ0.POCR.BIT.POLB
TCNT	TZ0.TCNT (unsigned int)	
GRA	TZ0.GRA (unsigned int)	
GRB	TZ0.GRB (unsigned int)	
GRC	TZ0.GRC (unsigned int)	
GRD	TZ0.GRD (unsigned int)	

タイマZ1

レジスタ名	バイトアクセス変数名	ビットアクセス変数名
TCR	TZ1.TCR.BYTE	TZ1.TCR.BIT.CCLR
		TZ1.TCR.BIT.CKEG
		TZ1.TCR.BIT.TPSC
TIORA	TZ1.TIORA.BYTE	TZ1.TIORA.BIT.IOB
		TZ1.TIORA.BIT.IOA
TIORC	TZ1.TIORC.BYTE	TZ1.TIORC.BIT.IOD
		TZ1.TIORC.BIT.IOC
TSR	TZ1.TSR.BYTE	TZ1.TSR.BIT.UDF
		c:¥tmp¥iodefine03.ps
		TZ1.TSR.BIT.IMFD
		TZ1.TSR.BIT.IMFC
		TZ1.TSR.BIT.IMFB
		TZ1.TSR.BIT.IMFA
TIER	TZ1.TIER.BYTE	TZ1.TIER.BIT.OVIE
		TZ1.TIER.BIT.IMIED
		TZ1.TIER.BIT.IMIEC
		TZ1.TIER.BIT.IMIEB
		TZ1.TIER.BIT.IMIEA
POCR	TZ1.POCR.BYTE	TZ1.POCR.BIT.POLD
		TZ1.POCR.BIT.POLC
		TZ1.POCR.BIT.POLB
TCNT	TZ1.TCNT (unsigned int)	
GRA	TZ1.GRA (unsigned int)	
GRB	TZ1.GRB (unsigned int)	
GRC	TZ1.GRC (unsigned int)	
GRD	TZ1.GRD (unsigned int)	

タイマZ共通

レジスタ名	バイトアクセス変数名	ビットアクセス変数名
TSTR	TZ.TSTR.BYTE	TZ.TSTR.BIT.STR1
		TZ.TSTR.BIT.STR0
TMDR	TZ.TMDR.BYTE	TZ.TMDR.BIT.BFD1
		TZ.TMDR.BIT.BFC1
		TZ.TMDR.BIT.BFD0
		TZ.TMDR.BIT.BFC0
		TZ.TMDR.BIT.SYNC
TPMR	TZ.TPMR.BYTE	TZ.TPMR.BIT.PWMD1
		TZ.TPMR.BIT.PWMC1
		TZ.TPMR.BIT.PWMB1
		TZ.TPMR.BIT.PWMD0
		TZ.TPMR.BIT.PWMC0
		TZ.TPMR.BIT.PWMB0
TFCR	TZ.TFCR.BYTE	TZ.TFCR.BIT.STCLK
		TZ.TFCR.BIT.ADEG
		TZ.TFCR.BIT.ADTRG
		TZ.TFCR.BIT.OLS1
		TZ.TFCR.BIT.OLS0
		TZ.TFCR.BIT.CMD
TOER	TZ.TOER.BYTE	TZ.TOER.BIT.ED1
		TZ.TOER.BIT.EC1
		TZ.TOER.BIT.EB1
		TZ.TOER.BIT.EA1
		TZ.TOER.BIT.ED0
		TZ.TOER.BIT.EC0
		TZ.TOER.BIT.EB0
		TZ.TOER.BIT.EA0
TOCR	TZ.TOCR.BYTE	TZ.TOCR.BIT.TOD1
		TZ.TOCR.BIT.TOC1
		TZ.TOCR.BIT.TOB1
		TZ.TOCR.BIT.TOA1
		TZ.TOCR.BIT.TOD0
		TZ.TOCR.BIT.TOC0
		TZ.TOCR.BIT.TOB0
		TZ.TOCR.BIT.TOA0

低電圧検出回路LVD

レジスタ名	バイトアクセス変数名	ビットアクセス変数名
CR	LVD.CR.BYTE	LVD.CR.BIT.LVDE
		LVD.CR.BIT.LVDSEL
		LVD.CR.BIT.LVDRE
		LVD.CR.BIT.LVDDE
		LVD.CR.BIT.LVDUE
SR	LVD.SR.BYTE	LVD.SR.BIT.LVDDF
		LVD.SR.BIT.LVDUF

シリアルコミュニケーションインタフェースSCI_2

レジスタ名	バイトアクセス変数名	ビットアクセス変数名
SMR	SCI3_2.SMR.BYTE	SCI3_2.SMR.BIT.COM
		SCI3_2.SMR.BIT.CHR
		SCI3_2.SMR.BIT.PE
		SCI3_2.SMR.BIT.PM
		SCI3_2.SMR.BIT.STOP
		SCI3_2.SMR.BIT.MP
		SCI3_2.SMR.BIT.CKS
SCR3	SCI3_2.SCR3.BYTE	SCI3_2.SCR3.BIT.TIE
		SCI3_2.SCR3.BIT.RIE
		SCI3_2.SCR3.BIT.TE
		SCI3_2.SCR3.BIT.RE
		SCI3_2.SCR3.BIT.MPIE
		SCI3_2.SCR3.BIT.TEIE
		SCI3_2.SCR3.BIT.CKE
SSR	SCI3_2.SSR.BYTE	SCI3_2.SSR.BIT.TDRE
		SCI3_2.SSR.BIT.RDRF
		SCI3_2.SSR.BIT.OER
		SCI3_2.SSR.BIT.FER
		SCI3_2.SSR.BIT.PER
		SCI3_2.SSR.BIT.TEND
		SCI3_2.SSR.BIT.MPBR
		SCI3_2.SSR.BIT.MPBT
BRR	SCI3_2.BRR	-
TDR	SCI3_2.TDR	-
RDR	SCI3_2.RDR	-

IIC2

レジスタ名	バイトアクセス変数名	ビットアクセス変数名
ICCR1	IIC2.ICCR1.BYTE	IIC2.ICCR1.ICE
		IIC2.ICCR1.RCVD
		IIC2.ICCR1.MST
		IIC2.ICCR1.TRS
		IIC2.ICCR1.CKS
ICCR2	IIC2.ICCR2.BYTE	IIC2.ICCR2.BBSY
		IIC2.ICCR2.SCP
		IIC2.ICCR2.SDAO
		IIC2.ICCR2.SDAOP
		IIC2.ICCR2.SCLO
		IIC2.ICCR2.IICRST
ICMR	IIC2.ICMR.BYTE	IIC2.ICMR.MLS
		IIC2.ICMR.WAIT
		IIC2.ICMR.BCWP
		IIC2.ICMR.BC
ICIER	IIC2.ICIER.BYTE	IIC2.ICIER.TIE
		IIC2.ICIER.TEIE
		IIC2.ICIER.RIE
		IIC2.ICIER.NAKIE
		IIC2.ICIER.STIE
		IIC2.ICIER.ACKE
		IIC2.ICIER.ACKBR
		IIC2.ICIER.ACKBT
ICSR	IIC2.ICSR.BYTE	IIC2.ICSR.TDRE
		IIC2.ICSR.TEND
		IIC2.ICSR.RDRF
		IIC2.ICSR.NACKF
		IIC2.ICSR.STOP
		IIC2.ICSR.ALOVE
		IIC2.ICSR.AAS
		IIC2.ICSR.ADZ
SAR	IIC2.SAR.BYTE	IIC2.SAR.SVA
		IIC2.SAR.FS
ICDRT	IIC2.ICDRT	-
ICDRR	IIC2.ICDRR	-

TB1

レジスタ名	バイトアクセス変数名	ビットアクセス変数名
TMB1	TB1.TMB1.BYTE	TB1.TMB1.RLD
		TB1.TMB1.CKS
TCB1	TB1.TCB1 (unsigned char)	

FLASH

レジスタ名	バイトアクセス変数名	ビットアクセス変数名
FLMCR1	FLASH.FLMCR1.BYTE	FLASH.FLMCR1.SWE
		FLASH.FLMCR1.ESU
		FLASH.FLMCR1.PSU
		FLASH.FLMCR1.EV
		FLASH.FLMCR1.PV
		FLASH.FLMCR1.E
		FLASH.FLMCR1.P
FLMCR2	FLASH.FLMCR2.BYTE	FLASH.FLMCR1.FLER
WK1	FLASH.wk1 (char)	
EBR1	FLASH.EBR1.BYTE	FLASH.EBR1.EB4
		FLASH.EBR1.EB3
		FLASH.EBR1.EB2
		FLASH.EBR1.EB1
		FLASH.EBR1.EB0
WK2	FLASH.wk2[7] (char)	
FENR	FLASH.FENR.BYTE	FLASH.FENR.FLSHE

TV

レジスタ名	バイトアクセス変数名	ビットアクセス変数名
TCRV0	TV.TCRV0.BYTE	TV.TCRV0.CMIEB
		TV.TCRV0.CMIEA
		TV.TCRV0.OVIE
		TV.TCRV0.CCLR
		TV.TCRV0.CKS
TCSRV	TV.TCSRV.BYTE	TV.TCSRV.CMFB
		TV.TCSRV.CMFA
		TV.TCSRV.OVF
		TV.TCSRV.OS
TCORA	TV.TCORA	-
TCORB	TV.TCORB	-
TCNTV	TV.TCNTV	-
TCRV1	TV.TCRV1.BYTE	TV.TCRV1.TVEG
		TV.TCRV1.TRGE
		TV.TCRV1.ICKS

SCI3

レジスタ名	バイトアクセス変数名	ビットアクセス変数名
SMR	SCI3.SMR.BYTE	SCI3.SMR.COM
		SCI3.SMR.CHR
		SCI3.SMR.PE
		SCI3.SMR.PM
		SCI3.SMR.STOP
		SCI3.SMR.MP
		SCI3.SMR.CKS
BRR	SCI3.BRR	-
SCR3	SCI3.SCR3.BYTE	SCI3.SCR3.TIE
		SCI3.SCR3.RIE
		SCI3.SCR3.TE
		SCI3.SCR3.RE
		SCI3.SCR3.MPIE
		SCI3.SCR3.TEIE
		SCI3.SCR3.CKE
TDR	SCI3.TDR	-
SSR	SCI3.SSR.BYTE	SCI3.SSR.TDRE
		SCI3.SSR.RDRF
		SCI3.SSR.OER
		SCI3.SSR.FER
		SCI3.SSR.PER
		SCI3.SSR.TEND
		SCI3.SSR.MPBR
		SCI3.SSR.MPBT
RDR	SCI3.RDR	-

AD

レジスタ名	バイトアクセス変数名	ビットアクセス変数名
ADDRA	AD.ADDRA (unsigned int)	
ADDRB	AD.ADDRB (unsigned int)	
ADDRC	AD.ADDRC (unsigned int)	
ADDRD	AD.ADDRD (unsigned int)	
ADCSR	AD.ADCSR.BYTE	AD.ADCSR.ADF
		AD.ADCSR.ADIE
		AD.ADCSR.ADST
		AD.ADCSR.SCAN
		AD.ADCSR.CKS
		AD.ADCSR.CH
ADCR	AD.ADCR.BYTE	AD.ADCR.TRGE

PWM

レジスタ名	バイトアクセス変数名	ビットアクセス変数名
PWDRL	PWM.PWDRL	-
PWDRU	PWM.PWDRU	-
PWCR	PWM.PWCR.BYTE	PWM.PWCR.CKS

WDT

レジスタ名	バイトアクセス変数名	ビットアクセス変数名
TCSRWD	WDT.TCSRWD.BYTE	WDT.TCSRWD.B6WI
		WDT.TCSRWD.TCWE
		WDT.TCSRWD.B4WI
		WDT.TCSRWD.TCSRWE
		WDT.TCSRWD.B2WI
		WDT.TCSRWD.WDON
		WDT.TCSRWD.B0WI
		WDT.TCSRWD.WRST
TCWD	WDT.TCWD	-
TMWD	WDT.TMWD.BYTE	WDT.TMWD.CKS

ABRK

レジスタ名	バイトアクセス変数名	ビットアクセス変数名
ABRKCR	ABRK.ABRKCR.BYTE	ABRK.ABRKCR.RTINTE
		ABRK.ABRKCR.CSEL
		ABRK.ABRKCR.ACMP
		ABRK.ABRKCR.DCMP
ABRKSR	ABRK.ABRKSR.BYTE	ABRK.ABRKSR.ABIF
		ABRK.ABRKSR.ABIE
BDR	ABRK.BDR (unsigned int)	

IO

レジスタ名	バイトアクセス変数名	ビットアクセス変数名
PUCR1	IO.PUCR1.BYTE	IO.PUCR1.BIT.B7
		IO.PUCR1.BIT.B6
		IO.PUCR1.BIT.B5
		IO.PUCR1.BIT.B4
		IO.PUCR1.BIT.B2
		IO.PUCR1.BIT.B2
		IO.PUCR1.BIT.B1
PUCR5	IO.PUCR5.BYTE	IO.PUCR5.BIT.B5
		IO.PUCR5.BIT.B4
		IO.PUCR5.BIT.B3
		IO.PUCR5.BIT.B2
		IO.PUCR5.BIT.B1
		IO.PUCR5.BIT.B0
wk1	IO.wk1 (char[2])	
PDR1	IO.PDR1.BYTE	IO.PDR1.BIT.B7
		IO.PDR1.BIT.B6
		IO.PDR1.BIT.B5
		IO.PDR1.BIT.B4
		IO.PDR1.BIT.B2
		IO.PDR1.BIT.B1
		IO.PDR1.BIT.B0
PDR2	IO.PDR2.BYTE	IO.PDR2.BIT.B4
		IO.PDR2.BIT.B3
		IO.PDR2.BIT.B2
		IO.PDR2.BIT.B1
		IO.PDR2.BIT.B0
PDR3	IO.PDR3.BYTE	IO.PDR3.BIT.B7
		IO.PDR3.BIT.B6
		IO.PDR3.BIT.B5
		IO.PDR3.BIT.B4
		IO.PDR3.BIT.B3
		IO.PDR3.BIT.B2
		IO.PDR3.BIT.B1
		IO.PDR3.BIT.B0
wk2	IO.wk2 (char)	
PDR5	IO.PDR5.BYTE	IO.PDR5.BIT.B7
		IO.PDR5.BIT.B6
		IO.PDR5.BIT.B5
		IO.PDR5.BIT.B4
		IO.PDR5.BIT.B3
		IO.PDR5.BIT.B2
		IO.PDR5.BIT.B1
		IO.PDR5.BIT.B0
PDR6	IO.PDR6.BYTE	IO.PDR6.BIT.B7
		IO.PDR6.BIT.B6
		IO.PDR6.BIT.B5
		IO.PDR6.BIT.B4
		IO.PDR6.BIT.B3
		IO.PDR6.BIT.B2
		IO.PDR6.BIT.B1
		IO.PDR6.BIT.B0

PDR7	IO.PDR7.BYTE	IO.PDR7.BIT.B6
		IO.PDR7.BIT.B5
		IO.PDR7.BIT.B4
		IO.PDR7.BIT.B2
		IO.PDR7.BIT.B1
		IO.PDR7.BIT.B0
PDR8	IO.PDR8.BYTE	IO.PDR8.BIT.B7
		IO.PDR8.BIT.B6
		IO.PDR8.BIT.B5
wk3	IO.wk3 (char)	
PDRB	IO.PDRB.BYTE	IO.PDRB.BIT.B7
		IO.PDRB.BIT.B6
		IO.PDRB.BIT.B5
		IO.PDRB.BIT.B4
		IO.PDRB.BIT.B3
		IO.PDRB.BIT.B2
		IO.PDRB.BIT.B1
		IO.PDRB.BIT.B0
wk4	IO.wk4 (char[2])	
PMR1	IO.PMR1.BYTE	IO.PMR1.BIT.IRQ3
		IO.PMR1.BIT.IRQ2
		IO.PMR1.BIT.IRQ1
		IO.PMR1.BIT.IRQ0
		IO.PMR1.BIT.TXD2
		IO.PMR1.BIT. PWM
		IO.PMR1.BIT.TXD
PMR5	IO.PMR5.BYTE	IO.PMR5.BIT.WKP5
		IO.PMR5.BIT.WKP4
		IO.PMR5.BIT.WKP3
		IO.PMR5.BIT.WKP2
		IO.PMR5.BIT.WKP1
		IO.PMR5.BIT.WKP0
PMR3(Port	IO.PMR3.BYTE	IO.PMR3.POF3
		IO.PMR3.POF4
wk5	IO.wk5 (char)	
PCR1	IO.PCR1	-
PCR2	IO.PCR2	-
PCR3	IO.PCR3	-
wk6	IO.wk6 (char)	
PCR5	IO.PCR5	-
PCR6	IO.PCR6	-
PCR7	IO.PCR7	-
PCR8	IO.PCR8	-

SYSCR1(System Control Reg 1)

	バイトアクセス変数名	ビットアクセス変数名
SYSCR1	SYSCR1.BYTE	SYSCR1.BIT.SSBY
		SYSCR1.BIT.STS

SYSCR2(System Control Reg 2)

レジスタ名	バイトアクセス変数名	ビットアクセス変数名
SYSCR2(Sy	SYSCR2.BYTE	SYSCR2.MA.SMSEL
		SYSCR2.BIT.DTON
		SYSCR2.BIT.MA

IEGR1(Int. edge select reg. 1)

レジスタ名	バイトアクセス変数名	ビットアクセス変数名
IEGR1	IEGR1.BYTE	IEGR.BIT.NMIEG
		IEGR1.BIT.IEG3
		IEGR1.BIT.IEG2
		IEGR1.BIT.IEG1
		IEGR1.BIT.IEG0

IEGR2(Int. edge select reg. 2)

レジスタ名	バイトアクセス変数名	ビットアクセス変数名
IEGR2	IEGR2.BYTE	IEGR2.BIT.WPEG5
		IEGR2.BIT.WPEG4
		IEGR2.BIT.WPEG3
		IEGR2.BIT.WPEG2
		IEGR2.BIT.WPEG1
		IEGR2.BIT.WPEG0

IENR1(Int. enable register 1)

レジスタ名	バイトアクセス変数名	ビットアクセス変数名
IENR1	INER1.BYTE	INER1.BIT.IENDT
		INER1.BIT.IENWP
		INER1.BIT.IEN3
		INER1.BIT.IEN2
		INER1.BIT.IEN1
		INER1.BIT.IEN0

IENR2(Int. enable register 2)

レジスタ名	バイトアクセス変数名	ビットアクセス変数名
IENR2	INER2.BYTE	INER2.BIT.IENTB1

IRR1(Int. flag register 1)

レジスタ名	バイトアクセス変数名	ビットアクセス変数名
IRR1	IRR1.BYTE	IRR1.BIT.IRRDT
		IRR1.BIT.IRRI3
		IRR1.BIT.IRRI2
		IRR1.BIT.IRRI1
		IRR1.BIT.IRRI0

IRR2(Int. flag register 2)

レジスタ名	バイトアクセス変数名	ビットアクセス変数名
IRR2	IRR2.BYTE	IRR2.BIT.IRRTB1

IWPR(wake up flag register)

レジスタ名	バイトアクセス変数名	ビットアクセス変数名
IWPR	IWPR.BYTE	IWPR.BIT.IWPF5
		IWPR.BIT.IWPF4
		IWPR.BIT.IWPF3
		IWPR.BIT.IWPF2
		IWPR.BIT.IWPF1
		IWPR.BIT.IWPF0

MSTCR1 (Mod standby contr. 1)

レジスタ名	バイトアクセス変数名	ビットアクセス変数名
MSTCR1	MSTCR1.BYTE	MSTCR1.BIT.MSTIIC
		MSTCR1.BIT.MSTS3
		MSTCR1.BIT.MSTAD
		MSTCR1.BIT.MSTWD
		MSTCR1.BIT.MSTTV

MSTCR2 (Mod standby contr. 2)

レジスタ名	バイトアクセス変数名	ビットアクセス変数名
MSTCR2	MSTCR2.BYTE	MSTCR2.BIT.MSTS3 2
		MSTCR2.BIT.MSTTB1
		MSTCR2.BIT.MSTTZ
		MSTCR2.BIT.MSTPWM