Tutorial1A

創造設計第二 TA：田中 秀和

平成 23 年度 10 月 13 日

1. はじめに

試作検討 1A では創造設計第 2 で使用するマイコンの開発環境になれることを主な目的とする．また，試作検討を通じて，マイコンは H8/36064、マイコンボードは，Vstone 社製マイコンボード (VS-WRC003) を使用することを前提としている．

今回の創造設計では使用するマイコンの指定はされていないが，上記以外のマイコン及びマイコンボードを使用する場合は，TA 側でサポートすることは困難であるため，他のマイコンを使用する際は完全に自己責任で行うこと．

使用するボードセットはメカトロニクスラボで使用したものと同一のものなので，今回の試作検討ではメカトロニクスラボの復習が主な内容となっている．実際に取り組む内容としては以下のようになっている．

- 開発環境の理解
- RAM 化
- ポート入出力
- 割り込み（タイマ割り込みと外部割り込み）
- ROM 化

まず，開発環境に慣れるためにサンプルプログラムを実行し，動作を確認する．次に割り込みの方法（タイマ割り込みと外部割り込み）について確認する．最後に試作検討 1B で作成したメカニカルスイッチのチャタリング防止回路を動作確認するために，スイッチの信号を外部入力とする割り込みプログラムを作成し，マイクロコントローラのフラッシュメモリに書き込んで ROM 化を行う．

警告	各グループのメンバー全員が内容を理解できるように，確認し合って進めること．

警告	試作検討のプログラムや資料は必要に応じて創造設計第二のホームページ (http://www.ac.ctrl.titech.ac.jp/ss2_2011/index.html) からダウンロードすること．

2. 開発環境

本年度の創造設計から，H8 のマイコン使用するため開発環境として，ルネサステクノロジ社製の統合開発環境である High-performance Embedded Workshop (HEW) と，デバッガである HTerm を使用します．また，ROM への書き込みには Flash Development Toolkit(FDT) を使用する．

HEW,HTerm,FDT に関する詳細なマニュアルはルネサステクノロジのホームページを参考にすること．

注意	プロジェクトのワーキングディレクトリとして, 全角文字や空白を入れてはいけない

2.1 入出力ポート

VS-WRC003 には多くの入出力ポートが用意されている．デフォルトで用意されているポートの数は table1 のとおりである．

しかし、VS-WRC003 は、I/O 拡張ボード VS-WRC004 を使い、入出力ポートを増やすことができる．

今回は後にメカニカルスイッチを用いるため，ボード上の集合 IO(CN10/EXT) を利用できるようにする．

具体的には VS-WRC004 から必要分ピンを取り出し、ハンダ付けするだけである．

入出力ポートの詳細（ボード番号やレジストリ名）は VS-WRC003 マニュアルやルネサスのサポートページ http://www.vstone.co.jp/products/beauto_chaser/download.html のその他関連情報を参照すること．

Table 1: 入出力ポート数

DC モータ	2
RC サーボ	0 (ボードを拡張することで使用可能)
アナログ入力	2 (最大 4)
ブザー	1
LED	1 (1 つは電源用 LED のため，基本的には使用しない)

3. LED1 を点灯させる

本節では LED1 を点灯させることを通じて，入出力ポートの設定方法を復習すると共に，開発環境に慣れることを目的とする．また、ポートに対応するレジスタの探し方や、I/O レジスタ定義ファイル iodefine.h についても触れる。

3.1 モニタプログラムの書き込み

デフォルトの状態で配布されている VS-WRC003 には LCD(液晶パネル) が取り付けられていないため，本試作検討ではプログラムのデバッグをモニタプログラムを使用して行う．

サンプルプログラムの入った zip ファイルを解凍すると MONITOR.MOT が入っているので，FDT を用いてマイコンの ROM 領域にモニタプログラムを書き込む．

まず，PC とマイコンをシリアルケーブルで接続し，**プッシュスイッチを押しながら**電源を入れブートモードでマイコンを起動する。その際，デバイスマネージャ[マイコンピュータを右クリック → プロパティ→ デバイスマネージャ→ ポート] でマイコンが COM3 で接続されていることを確認する．

次に FDT Basic を起動して、それぞれの所に該当する情報を入力する。

- デバイスとカーネルの選択 Full Name ： H8/36064F
- 通信ポート：COM3(デバイスマネージャで確認した値)
- デバイス設定 入力クロック： 14.7456[MHz]
- 通信タイプと書き込みオプションはそのまま

最後に、FDT Simple Interface の画面になるので、User/Data Area にチェックをいれて、その横に MONITOR.MOT のあるアドレスを指定する。そして、スタートボタンをクリックすれば終わりである。うまくいかない場合はもう一度ブートモードでマイコンが起動しているか確認すること。

3.2 モニタプログラムの動作確認

次に HTerm を使用して，モニタプログラムが正常に書き込まれているか確認する。今、ブートモードでマイコンが起動していると思われるので、一度電源を切って通常モードでマイコンを起動させる。

Hterm を立ち上げると、COM1 が開けませんというエラーメッセージが出ることがある。この場合、ファイル → プロパティを選択して、Hterm のプロパティ画面を出す。ここで、

- 通信ポート：COM3(デバイスマネージャで確認した値)
- ビットレート：38400

をそれぞれ選択する。

次回以降は PC とマイコンが電源 (通常モード) が入っている状態で接続されている時に，HTerm を起動すると，自動的にマイコンと HTerm が接続され，白いウィンドウが表示されるようになる。

白いウィンドウが表示されている状態で，G を入力して Enter もしくは F5 ボタンを押すと，以下のメッセージが表示される．

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

このメッセージが表示されれば，モニタプログラムは正常に作動している．また，同時に HTerm とマイコンが正常に接続されていることもこのメッセージにより確認できる．

Hterm と通信中にマイコンの電源を落とさないこと。切断するか、Hterm を終了してから落とす。以後の Hterm との通信が一時的に不通になる可能性がある。

3.3　サンプルプログラム

本節ではサンプルプログラムを用いて，HEW，HTerm の使い方になれる．

本節の目的はサンプルプログラムを HEW でコンパイルし，HTerm で RAM 領域に書き込み，マイコンの LED1 を点灯させることを目的とする．

3.3.1　サンプルプロジェクト

ダウンロードした zip ファイルの sample_problem というフォルダの中に sample.hws という創造設計第 2 用のサンプルプロジェクトがあるので，ダブルクリックで起動する．

プロジェクトの作成は複雑な手順が必要なので，TA 側が使用するプロジェクト変更を支持した場合を除いて，今後はこのサンプルをコピーして使用する。詳しくは 7 章の個人 PC での開発環境の構築で述べる。

3.3.2　LED1 を点灯させる入出力ポートの特定

VS-wrc003 回路図を開き、LED1 の入出力ポートを探そう。Fig.1 のような部分が見つかるはず、そこにポート番号 (P64) が書いてある。

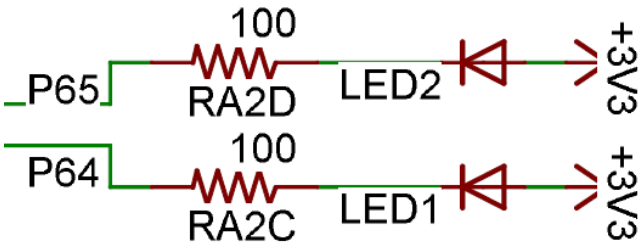


Fig. 1: LED 回路

3.3.3　ポート番号に対応するレジスタの特定

H8/36064 マニュアルを開き、ctrl+F で P64 を pdf 内検索をかけると、P64 はポート 6 に所属していることが分かる。

マニュアル内容から、ポート 6 に属するポートコントロールレジスタ 6(PCR6) とポートデータレジスタ 6(PDR6) の、P64 に対応する 4 ビット目の値を書き換えることによって、LED1 を点灯させることができることが分かる。

- PCR6 はポートを入力として使用するか、出力として使用するか決定する。

  PCR6 の詳細を Fig.2 に示す。

- 出力ポートとする場合、PDR6 により H レベル (1) 出力か L レベル (0) 出力かを決定する。

  PDR6 の詳細を Fig.3 に示す。

- 入力ポートとする場合、PDR6 は設定する必要はない。

今回はマイコンを操作して LED を光らせるので、P64 を**出力ポート**として使用する。また、Fig.1 の LED の向きを見れば、**L レベル出力**にすればよいことが分かる。

PCR6 はポート 6 の汎用入出力ポートとして使用する端子の入出力をビットごとに選択します。

ビット	ビット名	初期値	R/W	説　明
7	PCR67	0	W	汎用入出力ポートの機能が選択されているとき、このビットを 1 にセットすると対応する端子は出力ポートとなり、0 にクリアすると入力ポートとなります。
6	PCR66	0	W	
5	PCR65	0	W	
4	PCR64	0	W	
3	PCR63	0	W	
2	PCR62	0	W	
1	PCR61	0	W	
0	PCR60	0	W	

Fig. 2: PCR6

PDR6 はポート 6 の汎用入出力ポートデータレジスタです。

ビット	ビット名	初期値	R/W	説　　明
7	P67	0	R/W	ポート 6 の出力値を格納します。
6	P66	0	R/W	このレジスタをリードすると、PCR6 がセットされているビットはこのレジスタの値が読み出されます。PCR6 がクリアされているビットはこのレジスタの値にかかわらず端子の状態が読み出されます。
5	P65	0	R/W	
4	P64	0	R/W	
3	P63	0	R/W	
2	P62	0	R/W	
1	P61	0	R/W	
0	P60	0	R/W	

Fig. 3: PDR6

3.3.4　I/O レジスタ定義ファイル **iodefine.h** を使用したレジスタの値の変更方法

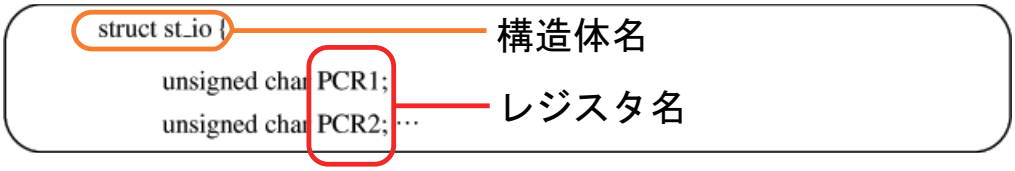
レジスタの値を変更するには、H8/36064 マニュアルの 20.1 節を参考にして、内部 I/O レジスタのアドレスをポインタで指定して直接変える方法があるが、いささか面倒である。また、他人がプログラムソースを読んだときに直感で理解しにくい。

そこで、ルネサスが提供している内部 I/O レジスタを定義しているファイルを利用して、プログラムを組むことにする。

sample.hws を起動すると、左側のワークスペースウィンドウ上に iodefine.h のファイルがあるので、ダブルクリックしてソースを見よう。

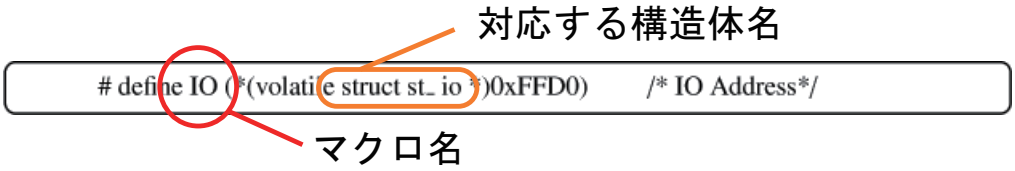
このソースの構成を示す。

- 構造体

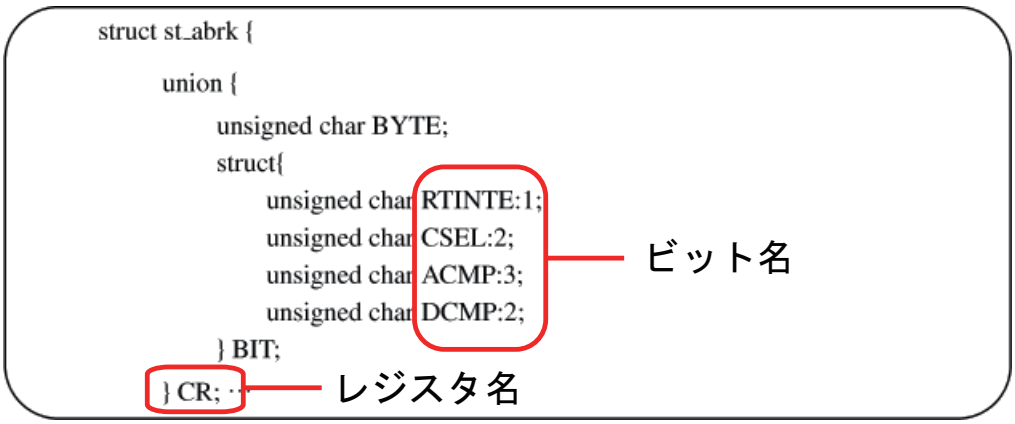


構造体 (struct) を使い内部 I/O レジスタを使いやすいように一つの塊にしている。

これに対応するのが、ソースの下の方にある、#define IO である。



- 共用体



共用体 (union) を使い、一つのレジスタのアドレスに対して、異なる型を処理できるようにしており、Bit 単位でアクセスすることが多いものは共用体で記述されている。CSEL:2; などビット名の後ろに書いてある数字（ここでは 2）は数字のビット分（ここでは 2 ビット分）のデータ量を持っていることを意味する。

以下に、レジスタの値を変更する場合の記述方法を記す。構造体型変数の中にある内部 I/O レジスタを参照するには、「マクロ名」,「レジスタ名」,「ビット名」をそれぞれドット演算子と呼ばれる”.”（ピリオド）を使ってつなげて記述する。

- **共用体で記述されていないとき**（構造体のみで記述されているとき）

```
[マクロ名].[レジスタ名] = 0x??
```

と記述する。0x は後ろに 16 進数が来ることを意味する。?に 0～F が入り、一桁で 4 ビット分の値を決定する。
例えば、PCR1 の値の 0 ビット目と 5 ビット目の値を 1 にしたい場合、IO.PCR1 = 0x21 の様に記述する。
レジスタ IENR2 のように、構造体でビットを指定している場合もある。その時は、

```
[マクロ名 (=レジスタ名)].BIT.[ビット名] = ?
```

と記述する。

- **共用体で記述されているとき**

```
[マクロ名].[レジスタ名].BIT.[ビット名] = ?
```

と記述する。?にはデータ量が n ビットの場合 0～(2^n – 1) の値が 10 進数で入る。
例えば、1 ビットデータ量の RTINTE の値を変更したい場合、ABRK.CR.BIT.RTINTE = 1 の様に記述する。 3 ビットデータ量の ACMP の値を変更して 3 ビットとも 1 にしたい場合、ABRK.CR.BIT.ACMP = 7 の様に記述する。
共用体にもバイト単位でアクセスことが可能である。その場合、

```
[マクロ名].[レジスタ名].BYTE = 0x??
```

と記述する。

> **注意** H8/36064 マニュアルと iodefine.h でのレジスタ名、ビット名が異なる場合がある。エラー回避のため、**iodefine.h で定義されているものを使うこと。**

3.3.5 LED を点灯させるプログラムの作成

サンプルプロジェクトをそのままコンパイルしても，何も実行されない．そこで，本節では sample.c 内の main 関数を書き換えて，LED1 を点灯させる．

課題 1a-1：LED1 の点灯

1. iodefine.h を開き、LED1 を点灯するために必要なレジスタ PCR6 のマクロ名、レジスタ名を調べろ。
2. 同様に、PDR6 のマクロ名、レジスタ名、ビット名を調べろ。
3. sample.c から課題 1a-1 に関連する部分のコメントアウトを外せ。(2 箇所)
4. 3.3 節を参考に sample.c 内の課題 1a-1 の?の部分を変更し，LED1(緑) のみを点灯させるプログラムを作成せよ。

sample.c の変更ができたら，コンパイルを行う．ビルドには Debug ビルドと Release ビルドがある。Dubug ビルドは Hterm 上でプログラムを走らせるときに使用して、Release ビルドは ROM 化を行う時に使用する。HEW の画面右上、デバックアイコンの右側に Debug と書いてあり、Debug ビルドに設定されいることを確認した後に，「すべてをビルド」を行う．
0 Errors となれば，コンパイルが完了である．同時に，実行ファイルである，sample.abs が Debug フォルダに生成されている．

3.4 RAM 領域への書き込みと実行

HEW はプログラムを ROM の先頭アドレスから順に配置することを想定している。しかし、試作検討では ROM にはモニタプログラムが書き込まれおり、プログラムを RAM のアドレスに配置する必要がある。
Fig.7 にマイコンのメモリマップを示す。ここから内臓 RAM が H’F780～H’FF7F 番地に割り当てられていることが分かる（H’ は後ろに 16 進数が来ること明記する記号）。ただし、モニタプログラム等の使用領域があるので、プログラムは H’F890 番地以降に配置することにする。
HEW を開き、ビルド →H8S, H8/300 Standard Toolchain... を選択する。最適化リンカを選び、カテゴリからセクションを選択すると Fig.4 のような画面が表示される。

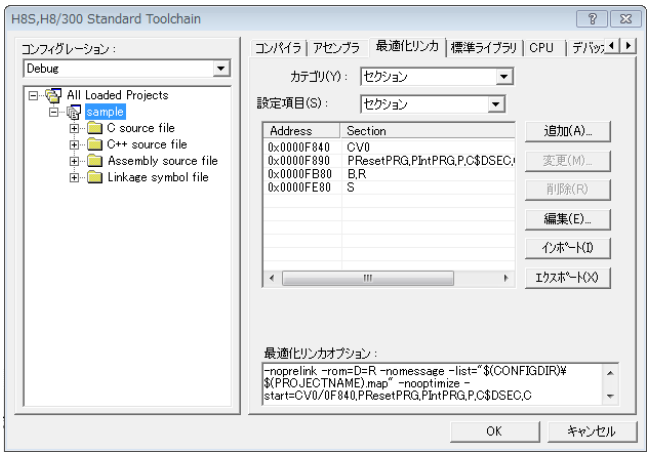


Fig. 4: H8S, H8/300 Standard Toolchain

編集ボタンを押して、Fig.5 のようにセクションを設定する。サンプルプログラムでは、すでに設定されているが、個人 PC で開発する時は変更が必要である。関連する内容を 7 章で述べる。Fig.6 に各セクションの意味を示す。

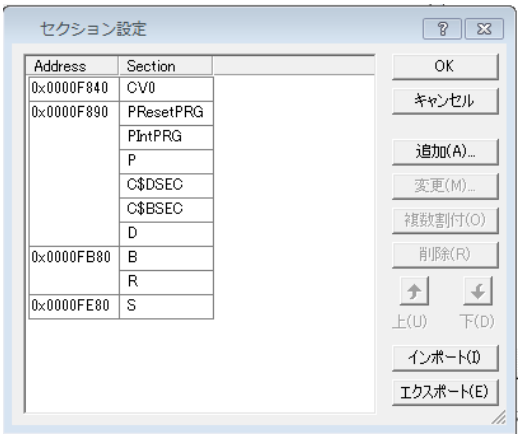


Fig. 5: Section 設定

PResetPRG	(HEW) resetprg.c にて #pragma section で生成されるプログラムコード。このセクションの先頭がプログラムのスタートアドレスになる
PIntPRG	(HEW) intprg.c にて #pragma section で生成されるプログラムコード
P	(C/C++) : プログラムコード
C	(C/C++) : 定数
C$DSEG	(HEW) dbsct.c の #pragma $DSEC にて生成される定数
C$BSEG	(HEW) dbsct.c の #pragma $BSEC にて生成される定数
D	(C/C++) 初期化データ
B	(C/C++) 未初期化データ
R	(C/C++) ROM 化支援で多重化された初期化データ
S	(HEW) stacksct.h の #pragma stacksize にて生成されるスタック領域

Fig. 6: 各 Section の意味

以上が終了したら、マイコンの電源を入れて、通常モードで起動し、HTerm を開いてコマンド → ロードもしくは F9 で sample.abs を選択する。ロードが完了すると「ソースプログラムを表示しますか？」と聞かれるので「はい」をクリックする。これで実行ファイルを RAM 領域に書き込むことができた。

この状態で G→Enter もしくは F5 を押すことで、プログラムが実行されるので、LED1(緑) が点灯すれば、完了である。

注意	この状態ではプログラムに終了コードが設定されていないため，Hterm を終了してから、ボードの電源を切ることで強制終了する．その後に，電源を入れて再接続を行う．

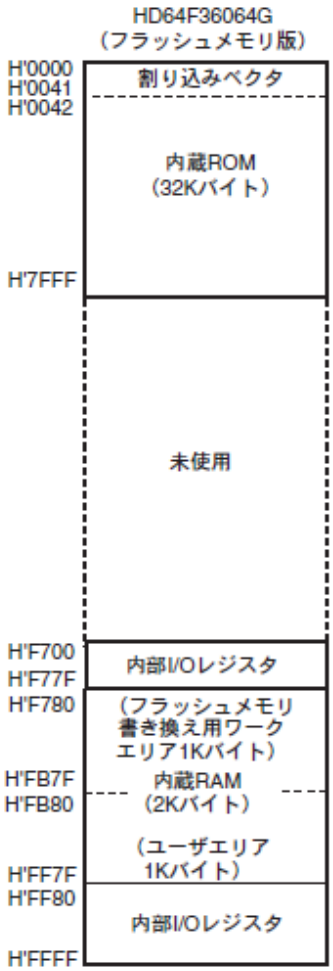


Fig. 7: H8/36064 メモリマップ

4. ポーリング

一定の間隔で繰り返しポートの値を読み込むことをポーリングという．入力ポートの電圧 (H/L) をプログラム中で変数の値 (0/1) として扱う[1]．前回の値を保持しておき今回の値と比較すれば，入力の変化 (エッジ) を検出することができる．次節でタイマ割り込みを用いた, ポーリングによるリアルタイム制御プログラムについて説明する．

[1]電圧とレジスタ値の関係は正論理である (つまりレジスタを 1 にすると電圧が出る)．ポート入力は負論理となっている

5. 割り込み

割り込みとは，実行中の処理を一時的に中断して，他の処理（割り込み処理）を行うことをいう．また，割り込み処理が終了した後には元の実行していた処理を再開させることができる．割り込み処理を行う主な利点としては次のようなものがある．

- 外部イベント (スイッチやセンサ入力などの外部入力によって行う処理) を確実に実行できる。
- 外部イベントに対する応答が速くなる。
- 定期的な処理を確実に実行できる。
- プログラム全体の処理効率が上が。
- 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。

- 多重割り込み (割り込み処理中に割り込みが発生すること) への対策が必要である。
- デバックがしにくい。
- 一定時間ごとに発生する割り込みでは、その割り込み周期を超えるような時間のかかる処理ができない。

割り込みについて、アナロジーを挙げて考えてみる。ゲームをやっていて、何も持っていない状態で主人公が公園に行っても通常イベントしか起きないが、アメを持って公園に行くとアメを子どもに渡すという、割り込みイベントが起こったというシーンを考える。

この場合、アメをアイテムとしてゲットすると割り込みイベントが発生する条件が整うことが分かる。

このように、割り込みイベントが発生する条件が整うことを**フラグがセットされた**（立った）といい、逆にイベントが発生する条件を失うことを**フラグがクリアされた**と呼ぶ。

さて、アメを持っているだけでは目的のイベントは起きない。当然公園に行く必要がある。すなわち、**フラグが立ってもイベントが起こることが確定するわけではない**。イベントの発生が許可される必要がある。イベント発生が許可されることを**要求がイネーブルになる**と呼ぶ。この例では、公園に行くことで割り込みイベントの発生要求がイネーブルになる。

では、アメを子どもに渡すイベントを終わらせるにはどうすれば良いだろうか。当然、アメが子どもの手に渡ればこのイベントは終了する。アメというフラグがクリアされるのである。

割り込み処理を行う場合、処理の中で**フラグをクリアするのを忘れてはならない**。割り込み処理が起こり続けることになってしまう。この例では、子どもの前でアメを見せ続けて与えないまま時間が経過し続けることに当たるだろうか。

ところで、割り込み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。設定が不完全なまま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ってしまい、非常に不安定になってしまう。これを避けるために、割り込みの設定する前に、全ての割り込みを禁止する必要がある。これを**割り込みをマスクする**とも呼ぶ。そして、割り込みの設定が終わったら割り込みマスクを解除すればよい。

全ての割り込みを禁止するコマンド：set_imask_ccr(1);

全ての割り込みを許可するコマンド：set_imask_ccr(0);

以上をまとめると、

- 割り込みを起こすためには
  - フラグをセットする
  - 割り込み要求をイネーブルにする
- 割り込み処理の中でフラグをクリアする
- 割り込みの設定前に割り込みをマスクし、設定後に解除する

フラグのセット・クリアや割り込みの許可は、全て該当するレジスタの値を変更することで行える。

5.1 タイマ割り込み

タイマによるカウント値を利用して，一定周期で割り込みを行うことをタイマ割り込みという。

本ボードには割り込み用のタイマとして，タイマ B1, タイマ V, タイマ Z0, タイマ Z1 が用意されている。各タイマの詳細はマニュアルを参照すること。本試作検討ではタイマ B1 を用いて 5[ms] 毎のタイマ割り込みを行う。

本節の目標はタイマ割り込みを用いて LED1 を 0.5 秒間隔で点滅させることである。

5.1.1　タイマ割り込みの設定

本節では，タイマ B1 を用いたタイマ割り込みの具体的なレジスタの設定を述べる。

- TCB1(タイマカウンタレジスタ)

タイマ B1 のカウントを行うレジスタである。入力クロック周波数の逆数 (つまり入力クロックの周期) が 1 カウントする時間になる。入力クロック周波数はマイコン内部のクロック周波数を利用している。詳しくは TMB1 内のビット、TMB10～TMB12 を説明するところで述べる。8 ビットカウンタなので $2^8 = 256$ がカウントされるとオーバーフローを起こす。

タイマ B1 を設定するために必要なレジスタは TMB1,TLB1,IENR2,IRR2 の 4 つである。タイマ B1 を動かすためにはこの 4 つのレジスタの値を変更する必要がある。

- IENR2(割り込みイネーブルレジスタ) と IRR2(割り込みフラグレジスタ)

割り込みを許可/禁止するレジスタと割り込みのフラグをセット/クリアするレジスタである。詳細を Fig.8 と Fig.9 に示す。

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。

Fig. 8: IENR2

レジスタ内の各ビットについて説明する。IENTB1 は値が 0 だと割り込みが禁止され、1 だと許可される。IRRTB1 は値が 0 だとフラグがクリアされ、1 だとフラグがセットされる。また、タイマカウント TCB1 のカウンタが 256 になるとオーバーフローを起こし、自動的に IRRTB1 が 1 になり、フラグがセットされる。

- TMB1(タイマモードレジスタ) と TLB1(タイマロードレジスタ)

TLB1 には 0～255 の間の整数が入り、タイマをオートリロード機能（後述）として使用する場合に使う。TMB1 は一回の割り込み時間を設定するために必要なレジスタである。詳細を Fig.10 に示す。

ここからは TMB1 レジスタ内の各ビットについて説明する。TMB17(RLD) はタイマの機能を設定するビットである。

オートリロード機能

TCB1 の値がオーバーフローして割り込みが発生すると TCB1 の値をクリアして、TLB1 に代入されている値が TCB1 に入り、カウントが行う機能。こちらの機能を使うことで、TLB1 の値を決めると割り込みの周期をより細かく指定することができる。

インターバル機能

TCB1 の値がオーバーフローして割り込みが発生するとタイマカウント TCB1 の値をクリアして、0 からカウントを行う機能。

割り込みフラグレジスタ 2（IRR2）

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	−	

Fig. 9: IRR2

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	RLD
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	CKS
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。	

Fig. 10: TMB1

TMB10～TMB12(CKS) は入力クロック周波数を決め、TCB1 が 1 カウントする時間を設定するビットである。カウントする速さは基本、内部クロックの周波数によって決まるが、そのまま内部クロックを入力クロック周波数とすると、速過ぎる場合がある。一般的にタイマにはプリスケーラというものがついており、カウントする速さを内部クロック周波数の 2 乗オーダーで遅くすることができる。また、3 ビットとも 1 とした場合、外部パルスを取り込み入力クロック周波数とするモードになる。この場合、外部パルスの周波数によりカウントする速さが決定する。

タイマ B1 の設定するプログラム

- sample.c

```
void main(void){
  //すべての割り込みを禁止
  set_imask_ccr(1);
  // TB1 : 5 [ms] ごとに割り込みを入れる
  TB1.TMB1.BIT.RLD = 1;
  TB1.TMB1.BIT.CKS = 2;
  TB1.TLB1 = ?;
  IENR2.BIT.IENTB1 = ?;        // 割り込みを許可
  IRR2.BIT.IRRTB1 = ?;         // フラグをクリア
  // すべての割り込みを許可
  set_imask_ccr(0);
}
//TCB1 がカウントを開始する
//TCB1 が 256 になりオーバーフローすると
//IRR2.BIT.IRRTB1 = 1; になり、割り込みが発生する
```

- intprg.c

```
// 割り込み関数の記述
__interrupt(vect=29) void INT_TimerB1(void)
{
  IRR2.BIT.IRRTB1 = ?; //フラグをクリア

  割り込み処理の内容
}
```

今回は 5[ms] ごとに割り込みを発生させたい。そこで、オートリロード機能を使用する。よって TB1.TMB1.BIT.RLD = 1; である。

それでは TLB1 の値を決定し、5[ms] ごとに割り込みが起こるようにしよう。ここで、より一般化して割り込み周期 T[s]、入力クロック周波数 f[Hz] とすると、TLB1 は以下のような式で表される。

$$T = \frac{1}{f} \times (2^8 - \mathrm{TLB1}) \tag{1}$$

2^8 は 8 ビットタイマカウンタ TCB1 がオーバーフローする値である。また、オートリロード機能を使用した場合、TCB1 の初期値は TLB1 の値になるので、その差がオーバーフローするまでのカウント数である。これに 1 カウントする時間をかける事で割り込みが発生する時間が求まる。

今回割り込み周期 T は 5[ms] にしたい。入力クロック周波数 f は TB1.TMB1.BIT.CKS = 2;（つまり 010）として、ϕ/512 を選択した。ただし、ϕ は内部クロック周波数を表し、ϕ = 14.7456[MHz] である。

$$T = 5 \times 10^{-3} \tag{2}$$

$$f = \frac{14.7456 \times 10^6}{512} \tag{3}$$

式 (1)〜(3) により、今回使用する TLB1 の値を求めることができる。（求めて TB1.TLB1 = ?; の?を求めた値に設定する。今回は割り切れるが、割り切れない場合は近似値を使うなど工夫が必要である）

5.1.2 タイマ割り込みの記述、割り込み関数

割り込みを発生させる場合、割り込み関数を使い、関数内に割り込み処理の内容を記述する必要がある。

割り込み関数はコマンドプロンプトを開いて、アセンブラファイルに割り込みベクタアドレスを記述することで、新たに定義することができる。今回、それについては H8S,H8/300 シリーズクロスアセンブラユーザーマニュアルを参照することにして、もともと定義されている関数を使用する。

HEW 左側のワークスペースウィンドウ上に intprg.c のファイルを見ると、その中に割り込みとそれに対応する関数が載っている。この intprg.c の関数内に割り込み処理内容を記述する。[2]

タイマ B1 の割り込み関数を探すと、以下のような部分が見つかる。__interrupt で始まる一文が割り込み関数である。

```
// vector 29 Timer B1
__interrupt(vect=29) void INT_TimerB1(void)
```

まず、この割り込み関数を使用する宣言をしなくてはならない。宣言については resetprg.c の 89 行目から 92 行目に記述してある。

```
#pragma interrupt (INT_TimerB1)
void INT_TimerB1(void);
```

resetprg.c は電源投入時に実行するプログラムである。なお、ここで宣言した関数と、intprg.c で処理内容を記述する関数は一致していないといけない。よって、課題 1a-3,4 用の割り込み関数 INT_WKP も宣言されているので、intprg.c 内では__interrupt(vect=18) void INT_WKP(void) についてもコメントアウトが外されている。逆に#pragma interrupt で宣言されていない関数に関してはコメントアウトをつけておく必要がある。

また、intprg.c の最初に sample.c で使用した変数と同じものを extern をつけて宣言する (21 行目から 26 行目)。extern 宣言を使うと、異なるソースファイルで変数を共有することができる。

intprg.c の INT_TimerB1(void) 内に処理を記述すればよいが、割り込み関数の中にフラグをクリアするコマンドを忘れずに記述すること。今回は IRR2.BIT.IRRTB1 = 0; である。

課題 1a-2：LED1 の点滅

1. サンプルコード内の課題 1a-2 の部分のコメントアウトを外す。sample.c と intprg.c との両方にある。(3 箇所)
2. 課題 1a-1 ではずしたコメントアウトはそのままでよい。
   (以降の課題もコメントアウトを一度外したらそのままでよい)
3. サンプル中の？を変更して，0.5 秒間隔で LED1 を点滅させる。
   LED1flag という 0,1 の値をとるフラグが用意されている．main 関数内に LED1flag の値によって，LED1 を点灯，消灯させる条件分岐が記述されている．

この課題によって，電源スイッチの横のボタンでプログラムを終了させることができるようになっている．原理は，ボタンのレジスタの値 IO.PDR5.BIT.B5 の値によって，reset というフラグを切り替え，reset = 1 のときに，無限ループを break によって抜け出るようにしている．

[2]割り込み処理が行われることは 15 行目の# pragma section IntPRG で宣言している。この宣言の後に割り込み処理を記述すればよいので、sample.c の方で# pragma section IntPRG の宣言をして割り込み処理を記述してもよい。

5.2 イベント割り込み

イベント割り込みとは、入力ポートの電圧が変化した時に起こる割り込みのことである．例えば、メカニカルスイッチなどの外部パルスの立ち上がり（または立ち下がり）を検出し、割り込み処理を実行することができる。**立ち上がり**とは電圧がL レベルから H レベルに変化することである。**立ち下り**はその逆である。外部割込みは、不定期にやってくる処理に対して便利であるので、例えば、先日壊れた南 5 号館の自動ドアに、付随するセンサーなどに使われているはずである。

タイマー割り込みのように、マイコン内部で起こった要因によって起こった割り込みを、**内部割込み**と呼ぶのに対し、イベント割り込みのように、スイッチなど、マイコン外部のイベントによって起こる割り込みを、**外部割込み**と呼ぶ。

5.2.1 イベント割り込みの設定

本節では、イベント割り込みを起こすための具体的なレジスタの設定を述べる。

このマイコンには外部割込みが3 種類あり、NMI,IRQ,WKP である。この中で NMI は優先度が一番高い割り込みであり、非常時の緊急停止などに利用されることが多い。マイコン基盤の押しボタンスイッチにも NMI の効果が割り当てられてあるが、それを MONITOR.MOT で無効に設定してあるので、Hterm で起動する際は、スイッチを押してもその割り込みは発生しない。IRQ と WKP では IRQ の方が優先度が高い。詳しくは H8/36064 マニュアルの表 3.1 を参考にすること。

今回は WKP を利用する。WKP は WKP0～WKP5 まであり、最大 6 種類の WKP を使用した割り込みを起こすことができる。しかし、割り込み関数は全て INT_WKP が実行されるので注意が必要である。[3] 今回は WKP0 を使用する。外部割り込み WKP0 を設定するために必要なレジスタは IEGR2,IENR1,IWPR,PMR5 である。

- IEGR2(割り込みエッジセレクトレジスタ)

エッジとは、ここでは電圧の値が変化する瞬間のことである。レジスタの詳細を Fig.11 に示す。立ち上がり検出で割り込みが起こるか、立ち下り検出で割り込みが起こるかを決めるレジスタである。今回使うスイッチは on にすると、電圧が H レベルから L レベルに変化し立ち下がりが起きる。逆にスイッチを off にすると、立ち上がりが起きる。0 ビット目の WPEG0 を 0 にして立ち下がりを検出するようにして、今回は on にした瞬間、割り込みが起こるようにする。

- IENR1(割り込みイネーブルレジスタ)

詳細を Fig.12 に示す。WKP による割り込みを許可するレジスタである。5 ビット目の IENWP を 1 にすると許可される。

- IWPR(ウェイクアップ割り込みフラグレジスタ)

詳細を Fig.13 に示す。割り込みフラグをセット/クリアするレジスタである。エッジが検出されるとフラグがセットされる。

- PMR5(ポートモードレジスタ)

詳細を Fig.14 に示す。WKP0 と対応するポートは P50 である。今回はスイッチからの信号を受け取り割り込みを起こすので、入力端子に設定する。すなわち、0 ビット目の WKP0 を 1 にする。

5.2.2 割り込み関数

WKP による割り込み関数は以下のようになる。

```
//  vector 18 WKP
__interrupt(vect=18) void INT_WKP(void){ }
```

この割り込み関数を使用する宣言は以下のようになる。

```
#pragma interrupt (INT_WKP)
void INT_WKP(void);
```

[3]それに対して、IRQ は IRQ0～IRQ3 まであり、それぞれ別の割り込み関数をもつ。優先順位は数字が小さいものほど高い。

IEGR2 は $\overline{\text{ADTRG}}$ 端子、$\overline{\text{WKP5}}$～$\overline{\text{WKP0}}$ 端子の割り込み要求を発生させるエッジの方向を選択します。

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	－	1	－	
5	WPEG5	0	R/W	WKP5 エッジセレクト 0：$\overline{\text{WKP5}}$ 端子（$\overline{\text{ADTRG}}$ 端子）入力の立ち下がりエッジを検出 1：$\overline{\text{WKP5}}$ 端子（$\overline{\text{ADTRG}}$ 端子）入力の立ち上がりエッジを検出
4	WPEG4	0	R/W	WKP4 エッジセレクト 0：$\overline{\text{WKP4}}$ 端子入力の立ち下がりエッジを検出 1：$\overline{\text{WKP4}}$ 端子入力の立ち上がりエッジを検出
3	WPEG3	0	R/W	WKP3 エッジセレクト 0：$\overline{\text{WKP3}}$ 端子入力の立ち下がりエッジを検出 1：$\overline{\text{WKP3}}$ 端子入力の立ち上がりエッジを検出
2	WPEG2	0	R/W	WKP2 エッジセレクト 0：$\overline{\text{WKP2}}$ 端子入力の立ち下がりエッジを検出 1：$\overline{\text{WKP2}}$ 端子入力の立ち上がりエッジを検出
1	WPEG1	0	R/W	WKP1 エッジセレクト 0：$\overline{\text{WKP1}}$ 端子入力の立ち下がりエッジを検出 1：$\overline{\text{WKP1}}$ 端子入力の立ち上がりエッジを検出
0	WPEG0	0	R/W	WKP0 エッジセレクト 0：$\overline{\text{WKP0}}$ 端子入力の立ち下がりエッジを検出 1：$\overline{\text{WKP0}}$ 端子入力の立ち上がりエッジを検出

Fig. 11: IEGR2

IENR1 は直接遷移割り込み、および外部端子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	IENDT	0	R/W	直接遷移割り込み要求イネーブル このビットを 1 にセットすると直接遷移割り込み要求がイネーブルになります。
6	−	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。
5	IENWP	0	R/W	ウェイクアップ割り込み要求イネーブル このビットは $\overline{\text{WKP5}}$~$\overline{\text{WKP0}}$ 端子共通のイネーブルビットで、1 にセットすると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	IEN3	0	R/W	IRQ3 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 $\overline{\text{IRQ3}}$ 端子の割り込み要求がイネーブルになります。
2	IEN2	0	R/W	IRQ2 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 $\overline{\text{IRQ2}}$ 端子の割り込み要求がイネーブルになります。
1	IEN1	0	R/W	IRQ1 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 $\overline{\text{IRQ1}}$ 端子の割り込み要求がイネーブルになります。
0	IEN0	0	R/W	IRQ0 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 $\overline{\text{IRQ0}}$ 端子の割り込み要求がイネーブルになります。

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。

Fig. 12: IENR1

IWPR は $\overline{\text{WKP5}}$～$\overline{\text{WKP0}}$ 端子の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	－	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	IWPF5	0	R/W	WKP5 割り込み要求フラグ ［セット条件］ $\overline{\text{WKP5}}$ 端子が割り込み入力に設定され、指定されたエッジを検出したとき ［クリア条件］ 0 をライトしたとき
4	IWPF4	0	R/W	WKP4 割り込み要求フラグ ［セット条件］ $\overline{\text{WKP4}}$ 端子が割り込み入力に設定され、指定されたエッジを検出したとき ［クリア条件］ 0 をライトしたとき
3	IWPF3	0	R/W	WKP3 割り込み要求フラグ ［セット条件］ $\overline{\text{WKP3}}$ 端子が割り込み入力に設定され、指定されたエッジを検出したとき ［クリア条件］ 0 をライトしたとき
2	IWPF2	0	R/W	WKP2 割り込み要求フラグ ［セット条件］ $\overline{\text{WKP2}}$ 端子が割り込み入力に設定され、指定されたエッジを検出したとき ［クリア条件］ 0 をライトしたとき
1	IWPF1	0	R/W	WKP1 割り込み要求フラグ ［セット条件］ $\overline{\text{WKP1}}$ 端子が割り込み入力に設定され、指定されたエッジを検出したとき ［クリア条件］ 0 をライトしたとき
0	IWPF0	0	R/W	WKP0 割り込み要求フラグ ［セット条件］ $\overline{\text{WKP0}}$ 端子が割り込み入力に設定され、指定されたエッジを検出したとき ［クリア条件］ 0 をライトしたとき

Fig. 13: IWPR

ビット	ビット名	初期値	R/W	説　　明
7 6	POF57 POF56	0 0	－ －	このビットを 1 にセットすると対応する端子は PMOS がカットオフし NMOS オープンドレイン出力となり、0 にクリアすると CMOS 出力となります。
5	WKP5	0	R/W	P55/WKP5/ADTRG 端子の機能を選択します。 0：汎用入出力ポート 1：WKP5 入力端子および ADTRG 入力端子
4	WKP4	0	R/W	P54/WKP4 端子の機能を選択します。 0：汎用入出力ポート 1：WKP4 入力端子
3	WKP3	0	R/W	P53/WKP3 端子の機能を選択します。 0：汎用入出力ポート 1：WKP3 入力端子
2	WKP2	0	R/W	P52/WKP2 端子の機能を選択します。 0：汎用入出力ポート 1：WKP2 入力端子
1	WKP1	0	R/W	P51/WKP1 端子の機能を選択します。 0：汎用入出力ポート 1：WKP1 入力端子
0	WKP0	0	R/W	P50/WKP0 端子の機能を選択します。 0：汎用入出力ポート 1：WKP0 入力端子

Fig. 14: PMR5

5.2.3　ブザー

発射メロディーを鳴らしたい場合や、過去にブザーの音程を使いデバックを行う人がいた例を除けば、そこまで本筋には絡まないので、ブザーについての説明は割愛する。

課題 1a-3：ブザーの ON/OFF を切り替える

1. サンプルプログラムから課題 1a-3 に関連する部分のコメントアウトを外す。（3 箇所）
2. sensor という変数の値により，ブザーの ON/OFF を行うコードを main 関数のループの中に記述する．
3. イベントの発生により，sensor の値が反転するようにイベント割り込みの処理を記述する．

以上のコードが書けたらスイッチ回路を接続した後、コンパイルをして，実行させる．
その際に，イベントとして試作検討 1B の Fig.2 のスイッチ回路[4] を使用する。
スイッチ回路が作成できたら，回路に Vcc,GND,Signal を接続する．ただし，Signal は P50 ポートに接続すること．どの I/O 端子に Signal を接続するかは VSWRC003 マニュアルのコネクタ仕様説明を参考にする．
スイッチの ON/OFF を切り替えて，Buzzer が ON/OFF されることを確認する。

⚠ 危険	V_{CC} と GND は絶対にショートさせないこと。最悪の場合、過電流によってマイコンが焼けて壊れる。特に端子ピンを大きなクリップで留めると、クリップどうしが触れ合ってショートする可能性があるので、IC テストクリップ等を利用する。

[4]今回は後でチャタリング防止機能をプログラム上で実装するので、試作検討 1BFig.1 のスイッチ回路は使用しない。

5.2.4　チャタリング防止

前節の状態で，プログラムを実行し，スイッチを ON/OFF すると，チャタリングが起きたはずである．チャタリングとは ON/OFF が切り替わる瞬間に，高周波で ON/OFF が切り替わってしまう現象である。本節ではチャタリングの防止を行い，前節の目標を達成することを目標とする．

チャタリング防止を行う方法として，ソフトウェア上で実現する方法と，ハードウェア（回路）上で実現する方法がある．回路上でチャタリング防止を行う方法は試作検討 1B で解説するため，ここではソフトウェア上で実現する方法を紹介する．

チャタリングは ON/OFF の切り替わりの瞬間のみで生じるので，最初の立ち上がり（もしくは立ち下がり）を検出したら，ある一定時間（チャタリングが落ち着くまでの時間），ブザーを ON/OFF するフラグを変更しないようにすれば良い．

課題 1a-4：チャタリング防止

1. 最初の立ち下がりが発生した時刻を記録し，0.1 秒間 sensor フラグの値が変更されないように課題 1a-3 で作成したプログラムを変更する。

Hint　ここで，立ち上がり時刻を記録する変数として，sensorONtime という変数が用意してある．プログラムが起動してからの経過時間は timer に記録されているので、二つの変数をうまく組み合わせて if 文を使い記述する。

```
/* 時間にゆとりがない人に、さらにヒント */
if (sensorONtime + ? < timer){  // sensor フラグが切り替わってから 0.1 秒間は
                                // フラグを反転させない.
sensor   =      ?      ;     // 割り込んだたびに sensor の値を反転させる.
     ?   =   ?   ;           // sersor フラグが反転した時刻を記憶
}
```

コンパイルして実行し、プログラムが正常に動作すれば、マイコンは Hterm 上で、以下のような動作を行うことができるようになっているはずです．

- 0.5 秒おきに LED1 が点滅する．
- メカニカルスイッチにより，ブザーを ON/OFF できる．
- 電源スイッチ横のボタンにより，プログラムを終了できる．

6.　ROM 化

ROM [5] は、読み出し専用のメモリである。マイコンの ROM 自体にプログラムを書き込む能力はない。ROM に書き込みたい場合は、PC などに接続する必要がある。

この章では、マイコンにプログラムを書き込む方法を示す。これにより、Hterm を使用せずにマイコンの電源を入れることでプログラムが実行されるようになる。

注意	H8/36064 マイコンの ROM の書き込み回数は 1000 回程度であり、多いとは言えない。デバックはできるだけ Hterm 上で済ませて、できるだけ ROM に書き込む回数を減らす方がよいであろう。 RAM の容量は 2k バイトしかないので、一般的には創造設計第二を進めていくと、RAM 不足になり ROM に書き込む機会が増えてくる。

まず、HEW を開き、画面の右上あたりにある Debug を Release に変更する。これで Release ビルドに変更された。[6]

その後、すべてをビルドを実行する。warning が出ても気にしなくてよい。(プログラム名).mot のファイルがプロジェクト内の Release ファイルの中に生成するので、モニタプログラムを書き込んだ時と同じように、FDT を用いて書き込みを行えば完了である。

注意	FDT でファイルを参照するときには必ず、Release から mot ファイルを選択すること。Debug 内の mot ファイルは選ばないこと。

書き込み後、スイッチを入れて正常に動くようなら、成功です。

次回以降も Hterm を使うので、モニタプログラムを FDT に入れなおして終了です。お疲れ様でした。

[5] Read-Only Memory の略である。Read Only Member の略として使われることもある。

[6] この時、セクションも Release 用に変更されるので、3.4 節で行ったセクションの変更をし直す必要はない。

7. 個人 PC での開発環境の構築

個人の PC での開発において、試作検討での回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、個人の責任において行ってください。これらの使用に起因し個人または第三者に生じた損害に関し、当授業関係者は、一切その責任を負いません。

7.1 マニュアルについて

マニュアルについては授業で使ったものをコピーして使う。参考までに URL を載せておく

- H8/36064 グループハードウェアマニュアル
  - http://japan.renesas.com/products/mpumcu/h8/h8300h_tiny/h836064/Documentation.jsp#
- VS-WRC003 テクニカルマニュアル
  - http://www.vstone.co.jp/products/vs_wrc003/download.html#03
- VS-WRC003 回路図
  - http://www.vstone.co.jp/products/vs_wrc003/download.html#03
- H8S,H8/300 シリーズクロスアセンブラユーザーマニュアル
  - http://documentation.renesas.com/jpn/products/tool/j702038_h8s.pdf

7.2 HEW(High-Performance Embedded Workshop)

7.2.1 ダウンロード

「【無償評価版】H8SX,H8S,H8 ファミリ用 C/C++コンパイラパッケージ V.7.00 Release 00」を http://japan.renesas.com/support/downloads/download_results/C2000801-C2000900/evaluation_software_h8c.jsp からダウンロードできる。

1. 上記 URL にアクセスする。
2. 内容を読んでよければ、同意するをクリックする。
3. 右下の Download をクリックする。
4. ログインするとダウンロードが開始される。（My Renesas に登録する必要がある）

7.3 インストール

ダウンロードした h8v7000_ev.exe を開く。特に思うところがなければ、next や次をクリックしていけばよい。ただし、使用契約許諾に同意できない場合はインストールできない。

- インストールは標準インストール (推奨) を選択する。
- インストール製品の選択の画面では、H8SX，H8S，H8 ファミリ用 C/C++コンパイラパッケージの方にチェックを入れてインストールする。
- 言語の選択では好きな方を選択する。

7.4 FDT(Flash Development Toolkit)

7.4.1 ソフトウェアのダウンロード

「【無償評価版】フラッシュ開発ツールキット V.4.07 Release 01」を http://japan.renesas.com/support/downloads/download_results/C2006401-C2006500/evaluation_software_fdt_v4.jsp からダウンロードできる。

1. 上記 URL にアクセスする。
2. 内容を読んでよければ、同意するをクリックする。
3. 右下の Download をクリックする。
4. ログインするとダウンロードが開始される。（My Renesas に登録する必要がある）

7.5 インストール

ダウンロードした fdtv407r01.exe を開く。特に思うところがなければ、next や次をクリックしていけばよい。ただし、使用権許諾契約書に同意できない場合はインストールできない。

- Select Language では好きなものを選択する。
- Select Features では全ての項目にチェックする。
- Select Options では.mot にチェックをいれる。(全ての拡張子にチェック)

7.6 Hterm とモニタプログラム

http://japan.renesas.com/support/seminar/child_folder/sample_program/seminar_sample_downh83h.jsp?のページに必要なものがそろっている。

7.6.1 ソフトウェアのダウンロード

「Hterm」は http://japan.renesas.com/media/support/seminar/sample_program/seminar_sample_downh83h/htermmdi.exe にアクセスするとダウンロードできる。

7.6.2 インストール

htermmdi.exe をダブルクリックして開く。展開先を聞いてくるので特に変更がなければそのまま OK をクリックする。新しくできたフォルダ C:/hterm に実行ファイル、Hterm.exe がある。

7.6.3 モニタプログラム

モニタプログラムは授業中に使用した MONITOR.MOT をコピーして、FDT で書き込めばよい。書き込む方法は 3.1 節を参照すること。

自分でモニタプログラムをカスタムしたい場合

http://japan.renesas.com/media/support/seminar/sample_program/seminar_sample_downh83h/300thew3.exe にアクセスする。

ダウンロードが終わると 300thew3.exe が作られるので、ダブルクリックして開く。展開先を聞いてくるので特に変更がなければそのまま OK をクリックする。新しくできたフォルダ C:/300t に monitor.hws があるので、ダブルクリックして HEW を立ち上げる。

ここで自分がカスタムしたいようにプログラムを書き換えることができる。終わったらすべてをビルドをクリックすると、MONITOR.MOT が作成される。プログラムのカスタム例が、田中正行先生の作成したページ、モニタプログラムの作成 (http://www.ok.ctrl.titech.ac.jp/ mtanaka/microcomputer/monitor.html#make) に載っているので、参考にするとよい。

また、ここではビットレートを指定できるが、変更した場合、Hterm の方のビットレートも変更するのを忘れないこと。Hterm のビットレートの変更については 3.2 節を参照すること。

提示したアドレスに関して not found が出る方へ

1. pdf からアドレスを正しくコピペできているか確認する。

   特に「_」がうまくコピペできず、「%20」が代わりに入っている場合がある。また、.jsp などの末尾までコピーされているか確認する。

2. それでも駄目な場合、ページが更新されている可能性がある。諦めて検索エンジンを使い、ダウンロードできるページを探す。

7.7 HEW 新規プロジェクトの作成

HEW を立ち上げると、ようこそ！のウインドウが出るので、新規プロジェクトワークスペースの作成にチェックを入れて、OK をクリックする。

Fig.15 の画面が出るので、ワークスペース名を英字で好きに入れて OK をクリックする。ここでは便宜上 abc とする。

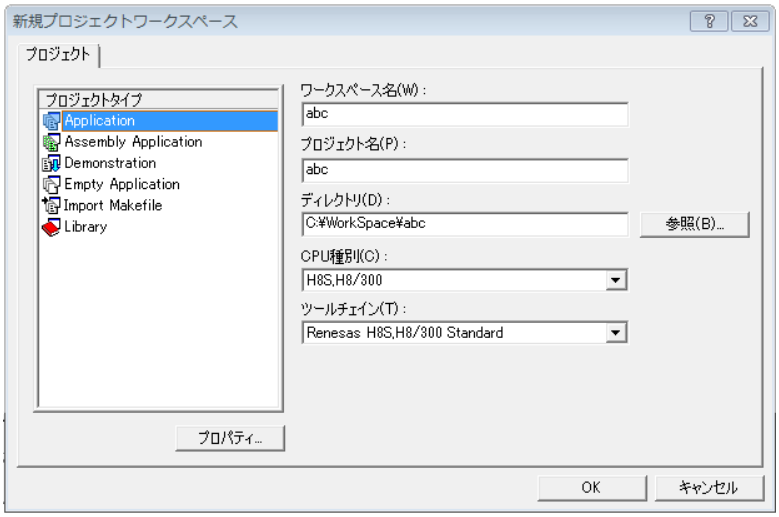


Fig. 15: 新規プロジェクトワークスペース

Fig.16 の画面が出るので、CPU シリーズから 300H を選択。CPU タイプから 36064 を選択して完了をクリックする。確認画面が出たら、CPU SERIES と CPU TYPE が選択されたものになっているのを確認して OK をクリックする。

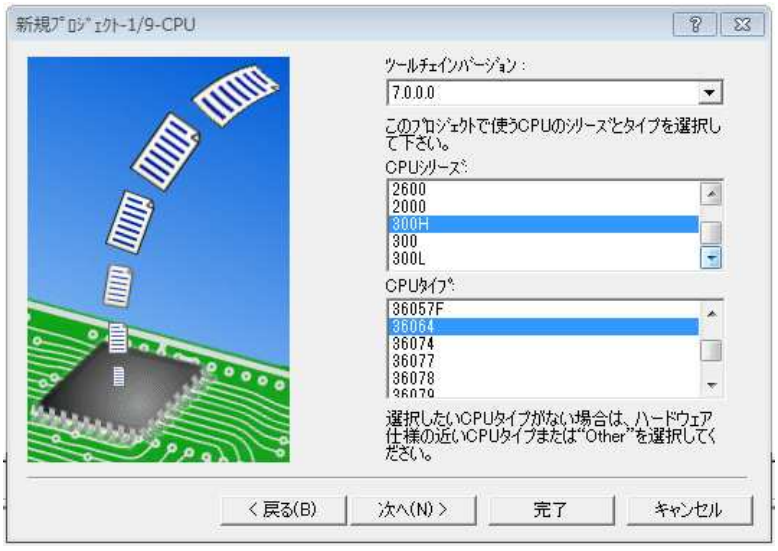


Fig. 16: new project

以上が終わったら、セクションを設定する。→3.4 節:RAM 領域への書き込みと実行を参照

最後に HEW で自動生成したプログラムを上書きする。一旦 HEW を閉じて、試作検討第一回目の sample_problem フォルダの中の resetprg.c,intprg.c,iodefine.h をコピーし、workspace の abc フォルダ (自分付けた名前) 内にある resetprg.c,intprg.c,iodefine.h に上書きする。HEW 開いても、ワークスペースウインドウ (左側) に iodefine.h のファイルは見当たらないが、全てをビルドをすると生成されるので問題はない。

自分で自動生成したプログラムをカスタムする場合は、割り込み関数や WDT(ウオッチドッグ・タイマ) の扱い等に注意が必要である。詳細は割愛する。

最後に、メインプログラムの始めに、#include <machine.h> と#include ”iodefine.h”の一文をそれぞれ書き加えれば終了である。

7.8 試作検討のプロジェクトの利用して新規プロジェクトを作成する

7.7 節で書いた新規プロジェクトの作成が面倒な場合、試作検討のプログラムを改変して使用する方法もある。ホームページからダウンロードした sample_problem フォルダ内の sample.hws を開き、sample.c の 24 行目から 66 行目の変数宣言、ブザーを鳴らす関数と 71 行目から 167 行目の void main(void) 内を消せばよい。また、resetprg.c の 89 行目から 92 行目で行っている、割り込み関数宣言や intprg.c の割り込み関数 (どちらも 5.1.2 節参照) の処理内容も適宜書き換えればよい。