3.4.6 ワークスペースにプロジェクトを追加

ワークスペースを最初に作ったときには、プロジェクトは 1 つしかありません。本テキストでは、H8/3694F のプロジェクトをすべて同一のワークスペース H8_3694F で管理しますので、新しいプロジェクトを追加していくことになります。

ここでは、ワークスペースにプロジェクトを追加する方法を説明します。

「プロジェクト」「プロジェクトの挿入」を選択します (図 3.23)。

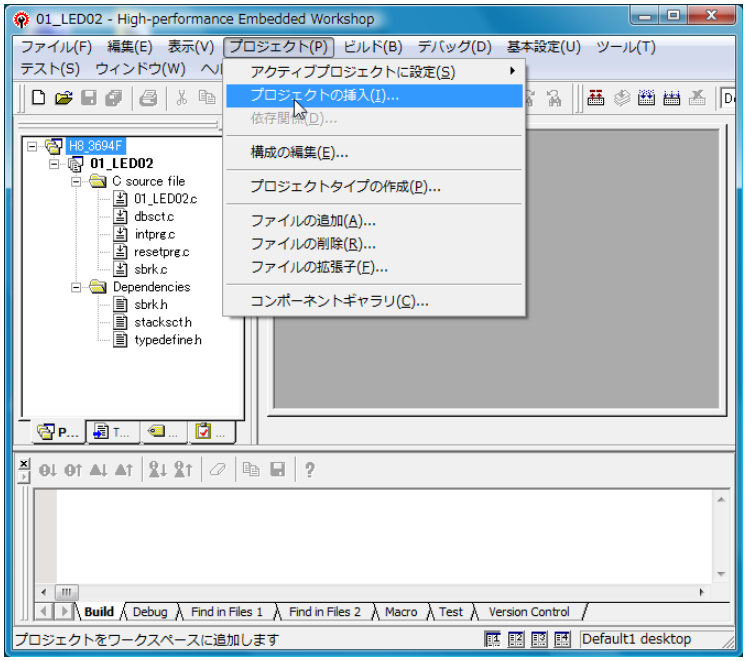

図 3.23　プロジェクトの挿入

この操作は、ワークスペースウィンドウのワークスペース名 (H8_3694F) を右クリックして選択することも可能です (図 3.24)。

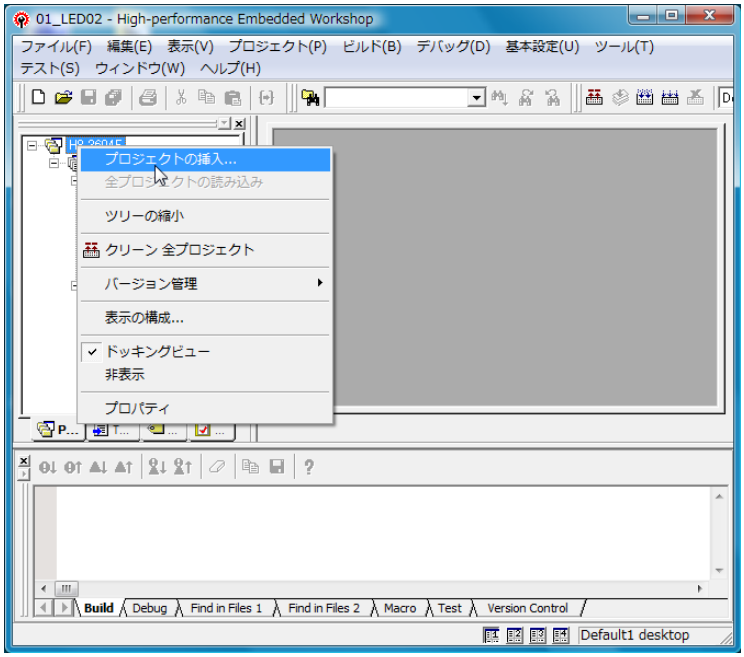

図 3.24　右クリックでプロジェクトの挿入

「プロジェクトの挿入」ダイアログが表示されるので、「新規プロジェクト」を選択し、「OK」をクリックします (図 3.25)。

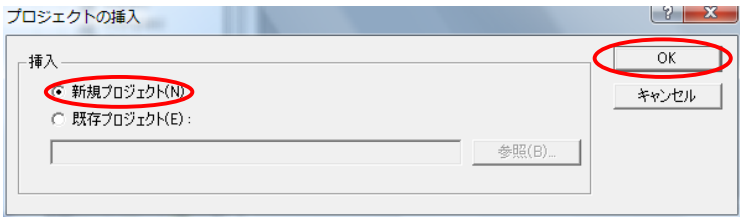

図 3.25　「プロジェクトの挿入」ダイアログ

次に「新規プロジェクトの挿入」ダイアログが表示されます。「プロジェクトタイプ」タブの「Application」を選択し、「プロジェクト名」にプロジェクト名を入れます (図 3.26)。ここでは「01_LED03」追加してみましょう。このとき「ディレクトリ」に自動的にプロジェクト名が追加されることを確認してください。

「OK」をクリックします。

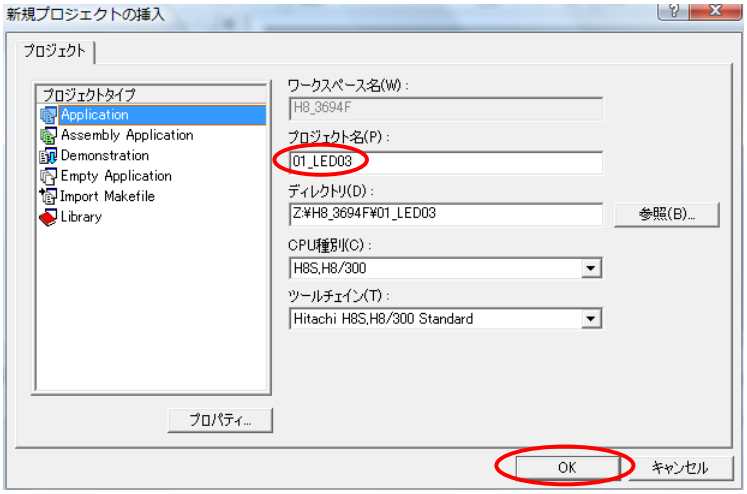

図 3.26　新規プロジェクトの挿入

ここから先は、P.66 の図 3.10 から P.70 の図 3.17 までの操作と同じです。図 3.17 の「プロジェクトの概要」ダイアログで「OK」ボタンをクリックすると、セッション変更の確認ダイアログが表示されます (図 3.27)。「はい」をクリックしましょう。

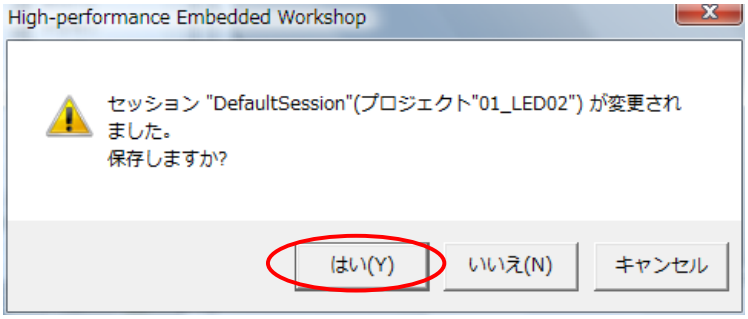

図 3.27　セッション変更の確認

以上の操作を完了すると、プロジェクト「01_LED03」が追加されていることが分かります (図 3.28)。

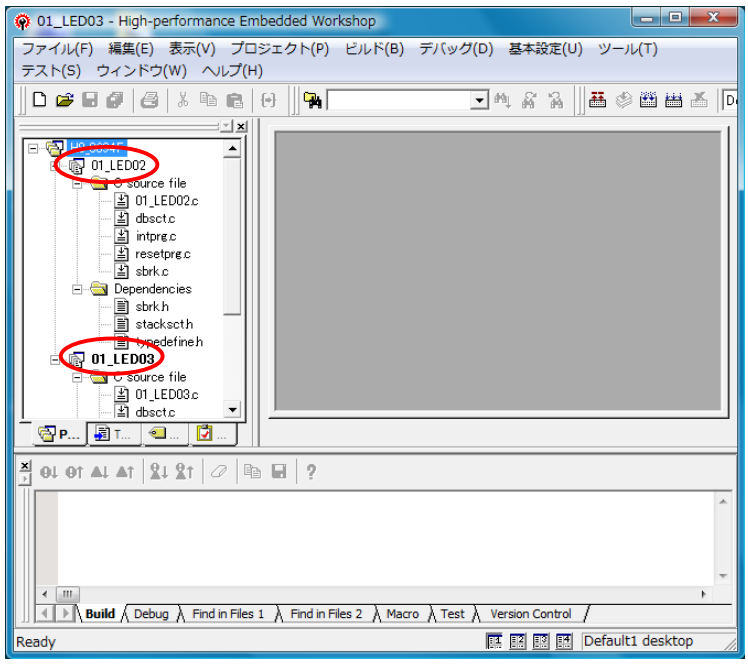

図 3.28　「01_LED03」の追加

プロジェクト名「01_LED03」は太字で、プロジェクト名「01_LED02」は通常の文字で表示されていることに注意してください。

太字のプロジェクトがビルドなどの操作対象となります。

このようなプロジェクトを「アクティブプロジェクト」と呼びます。

右側のウィンドウ (エディタウィンドウ) に表示されているファイルと、アクティブプロジェクトは必ずしも同じではないことに注意してください。ビルドの対象は編集しているファイルではなくて、あくまでもアクティブプロジェクトです。

混乱を避けるためには、エディタウィンドウにはアクティブプロジェクトのファイル以外は表示しないようにすると良いでしょう。

図 3.28 では、プロジェクト「01_LED02」と「01_LED03」はともに、プロジェクトの中にあるファイル名が階層的に表示されています。プロジェクト名の左側に - のマークが付いていることを確認してください。この表示の場合にはプロジェクトの中身が階層的に表示されます。

- のマークをクリックすると、マークが+に代わり、ファイル名が表示されなくなります (図 3.29)。アクティブプロジェクト以外はプログラム名を表示しないほうが扱いやすいかもしれません。

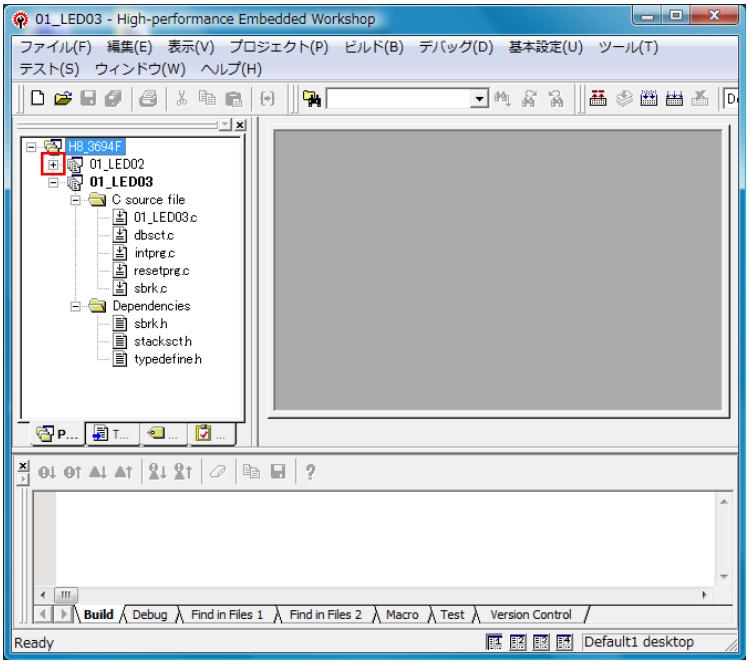

図 3.29　プロジェクトの表示

3.4.7 アクティブプロジェクトの切り替え

すでに説明したように、太字のプロジェクトがビルドなどの操作対象となります (アクティブプロジェクト)。

ここでは、アクティブプロジェクトの切り替えの方法を説明します。

ワークスペースウィンドウでアクティブにしたいプロジェクトを選んで、右クリックします。ここでは、プロジェクト「01_LED03」がアクティブプロジェクトになっているので、「01_LED02」をアクティブプロジェクトに変更してみましょう。ワークスペースウィンドウでプロジェクト「01_LED02」を選んで、右クリックします (図 3.30)。

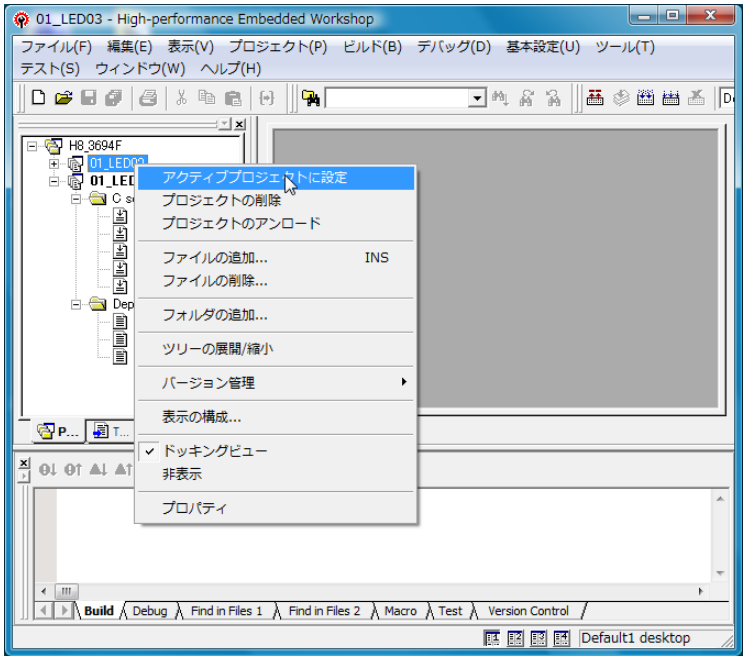

図 3.30　プロジェクトの変更

「アクティブプロジェクトに設定」を選択します。

セッション変更の確認ダイアログが表示されます (図 3.31)。「はい」をクリックしましょう。

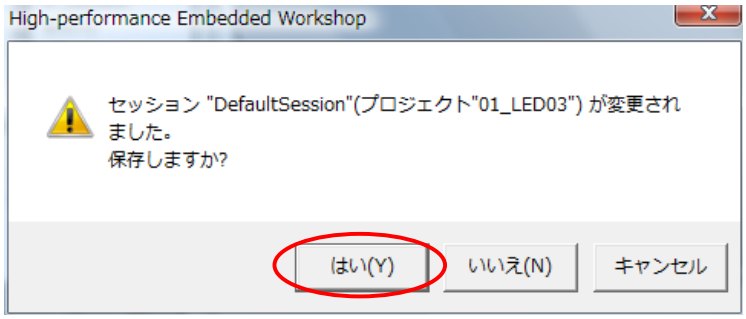

図 3.31　セッション変更の確認

「01_LED02」が太字になり、アクティブプロジェクトが変更されたことが分かります (図 3.32)。

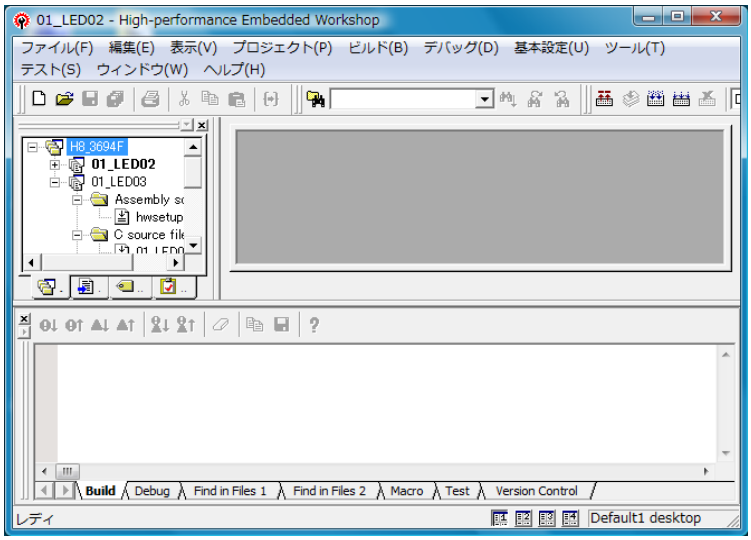

図 3.32　アクティブプロジェクト変更結果

3.5 FDT の利用

HEW でビルドした実行ファイルは、フラッシュ開発ツールキット (Flash Development Toolkit : FDT) を用いて、マイコンのフラッシュメモリに書き込みを行います。

以下では、この方法について解説します。

3.5.1 FDT の設定 (初めて FDT を起動する場合)

まずは、H8/3694F 用に FDT を設定します。

「スタート」-「すべてのプログラム」-「Renesas」-「Flash Development Toolkit 4.03」-「Flash Development Toolkit 4.03 Basic」を選択し、Flash Development Toolkit 4.03 Basic を起動します (図 3.33)。

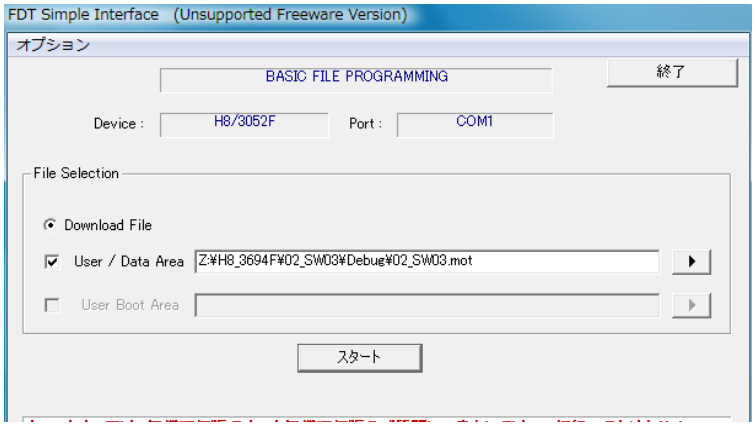

図 3.33　Flash Development Toolkit 4.03 Basic の画面

メニューの「オプション」-「新規作成」を選びます (図 3.34)。

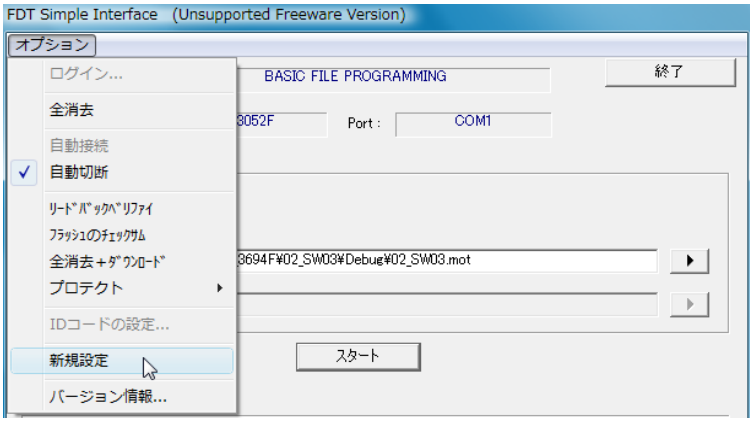

図 3.34　設定の新規作成

「デバイスとカーネルの選択」画面が出るので、「フィルタ」に「H8/3694」と入力します。リストにデバイス名「H8/3694F」が表示されるので、それを選択して「次へ」ボタンをクリックします (図 3.35)。

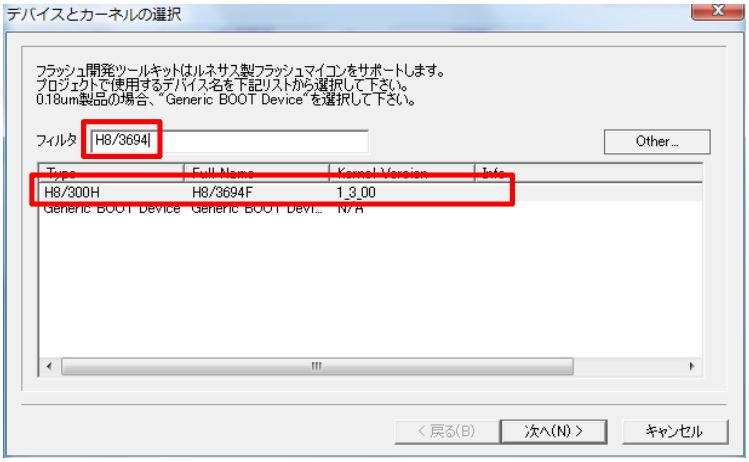

図 3.35 デバイスとカーネルの選択

「通信ポート」の設定画面が表示されます (図 3.36)。Select port が「COM1」になっていることを確認しておいてください。

図 3.36 「通信ポート」の設定画面

「デバイス設定」画面です。「入力クロック」が「20」Mhz になっていることを確認してください (図 3.37)。「次へ」ボタンをクリックします。

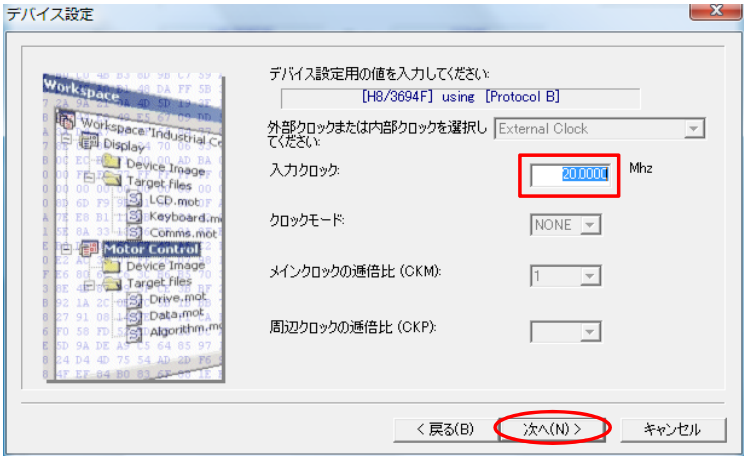

図 3.37　「デバイス設定」画面

「接続タイプ」設定画面です。書込み時の接続方法を設定します。「Use Default」にチェックがついていたら、チェックを外します。「ボーレート (推奨)」はプルダウンメニューから「19200」を選択します (図 3.38)。

「次へ」ボタンをクリックしましょう。

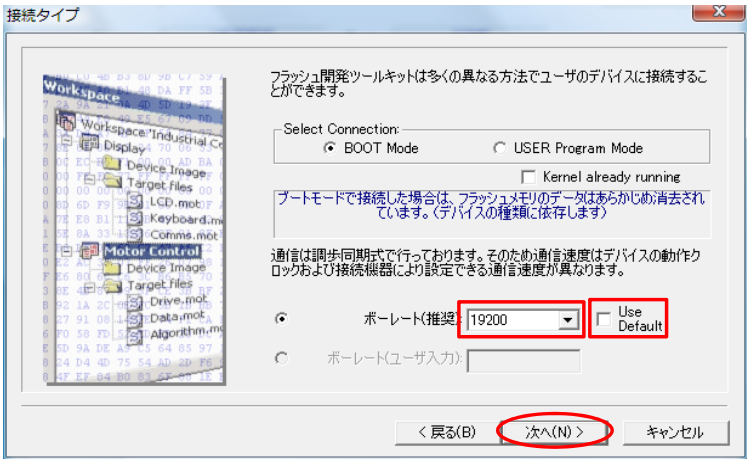

図 3.38　「接続タイプ」設定画面

「書き込みオプション」画面は、そのままの設定で、「完了」をクリックします (図 3.39)。

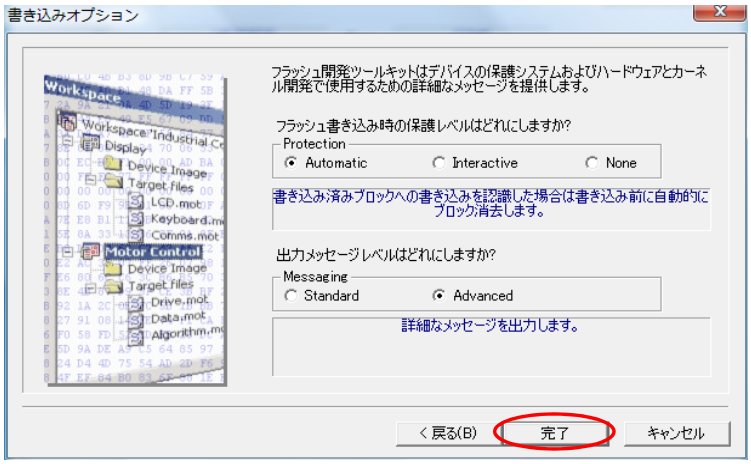

図 3.39 「書き込みオプション」画面

以上で、FDT の設定は終わりです。

3.5.2 実行ファイルの書込み

ここでは、実行ファイルの書き込み方法について説明します。

ここでは、`01_LED02.mot` を書きこんでみましょう。

「FDT Simple Interface」の画面で、「User / Data Area」にチェックをつけます。テキストボックスの右にある矢印ボタンをクリックしましょう (図 3.40)。

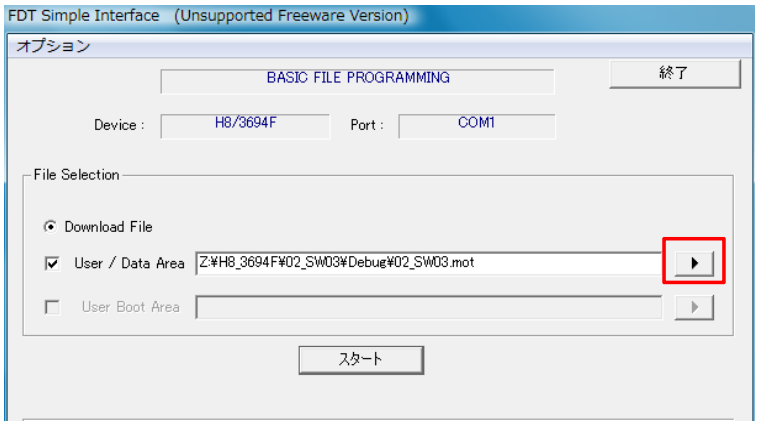

図 3.40 ファイルを選択する

「ファイルを開く」ダイアログが表示されるので、「ファイルの場所」を指定して 01_LED02.mot を選択し (Z:\H8_3694F\02_SW02\Debug\01_LED02.mot)、「開く」ボタンをクリックします (図 3.41)。

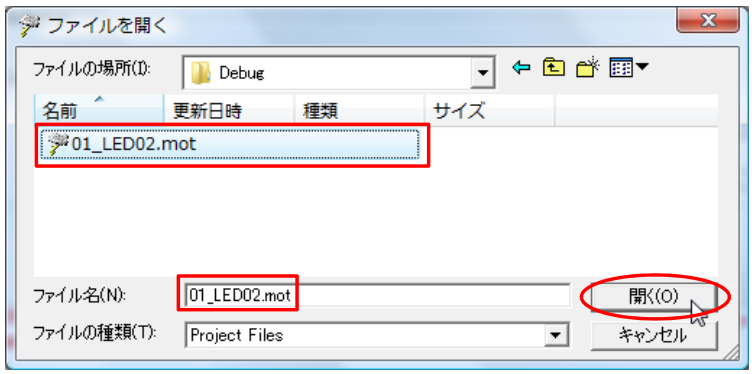

図 3.41　ファイルを選択する

書込み・拡張 I/O ボード MB-RS10 のシリアル通信コネクタ (CN1) にシリアル通信ケーブル (ストレート) のオス側をさします。「モード切り替えスイッチ」を「BOOT」側にして、電源スイッチを入れましょう。

「FDT Simple Interface」画面の「スタート」ボタンをクリックします (図 3.42)。

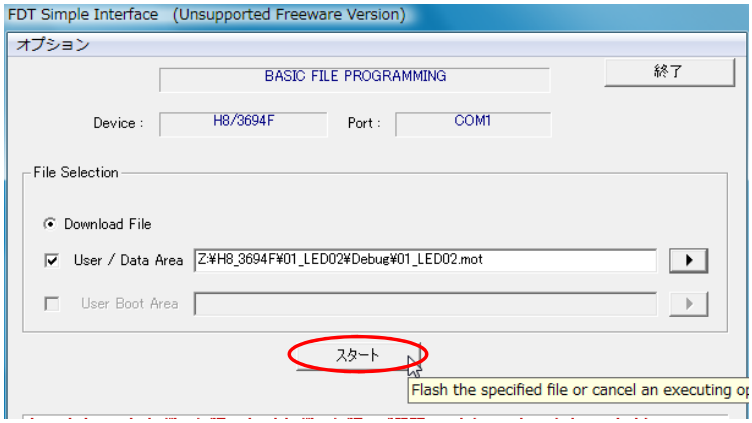

図 3.42　書き込みを実行する

書き込みが始まります。

書き込みが終了したら、書込み・拡張 I/O ボード MB-RS10 の電源スイッチを切り、「モード切り替えスイッチ」を「RUN」側にして、電源スイッチを入れましょう。

LED が正しく接続されていれば、LED1 が点灯するはずです。今のところ、LED を接続していないので、何も起こりません。

なお、「FDT Simple Interface」の「オプション」から「自動切断」にチェックをつけておくと、プログラムの書き込み後に、マイコンとの接続を自動的に切断してくれるので便利です (図 3.43)。

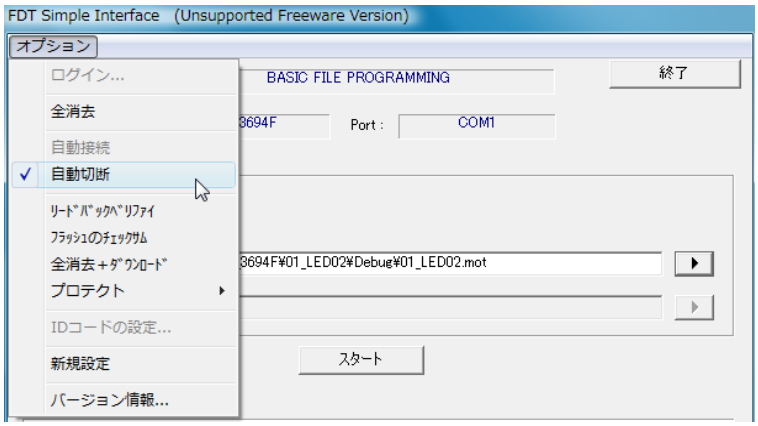

図 3.43 「自動切断」を設定する

3.5.3 FDT の 2 回目以降の起動

FDT を 2 回目以降に起動したときには、以下のように前回の設定が引き継がれています (図 3.44)。

書き込みたいファイルを選択して、「スタート」ボタンをクリックしてください。

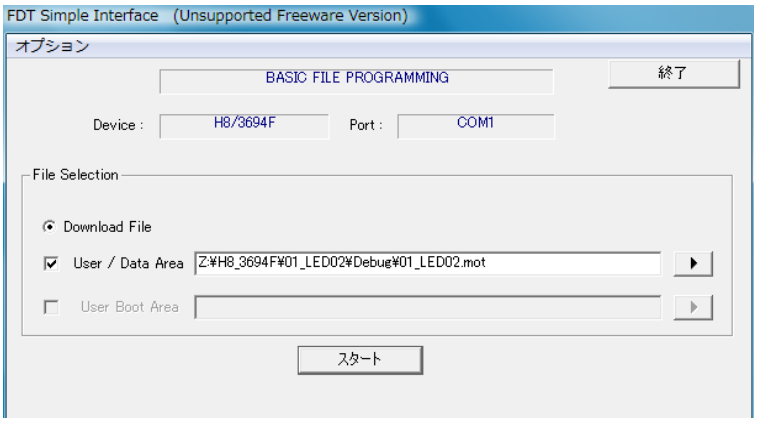

図 3.44 前回の設定が保存されている

4 LEDの点滅(マイコンプログラミングの基本)

4.1 LEDの回路

マイコンプログラムの基本を段階的に学ぶため、しばらくの間 LED の点滅を例に説明していきます。

マイコンのプログラミングはハードウェアを理解せずに行うことはできません。例えば、回路の構成が変更されると、プログラムもそれに応じた変更が必要になるわけです。

ここでは、LED の点滅をプログラムをする際に必要な、ハードウェアに関して説明しておきます。具体的には、LED の特性や回路の構成です。また、必要なレジスタの説明もしておきます。

4.1.1 I/O ポート (Input/Output Port) とは

LED やスイッチなどの外部回路 (機器) とマイコンの間でデータなどをやり取りするインターフェイス (端子) のことを I/O ポートといいます。

H8/3694F では一部の専用制御端子を除いて、端子は I/O ポート端子として利用することができます。

最大 8 本でまとめられ、ポートの名称がつけられています。ひとつの端子は 0 または 1 の値をやり取りすることができます。つまり I/O ポートは最大 8 ビットのデータを並列にやり取りすることができるわけです。

H8/3694F にはポート 1、ポート 2、ポート 5、ポート 7、ポート 8、ポート B の 6 本の I/O ポートがあります。

ポート 1 は 7 ビット、ポート 2 は 3 ビット、ポート 5 は 8 ビット、ポート 7 は 3 ビット、ポート 8 は 8 ビットの入出力ポートで、ポート B は 8 ビットの入力専用ポートです。合計 29 本 (29 ビット) の入出力ポートと 8 本 (8 ビット) の入力ポートを持っていることになります。

このうちポート 8 は大電流ポートで Low レベル出力時 20mA（VOL=1.5V 時）駆動できます。

ポートは他の機能 (P.61 表 3.2) と兼用になっています。

機能を選択したり、入力か出力かを設定したり、データの読み書きをするためにレジスタが用意されて

います。たとえば、ポート B は A/D コンバータと兼用になっていて、レジスタを設定することで機能を切り替えることができるのです。

本テキストでは、この I/O レジスタの設定の仕方を学ぶことになります。

4.1.2 LED(Light Emitting Diode) とは

LED は Light Emitting Diode の略で、発光ダイオードとも呼ばれます。pn 接合に順方向の電流が流れるときに、半導体に混ざっている不純物の種類によって特定の波長の光を放出します。この原理を用いたものが LED です。

人間に見える色だけでなく赤外線や紫外線など、さまざまな色の LED が存在します。

電圧を加える方向が決まっていて (「極性がある」といいます)、2 本の足のうち長いほう (アノード) を電源に、短いほう (カソード) をグランドにつなぎます (図 4.1)。

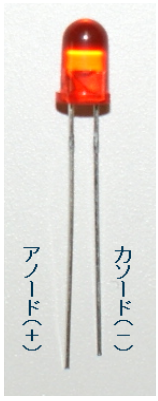

図 4.1　LED の概観

蛍光灯や白熱灯などに比べて、安価であり衝撃に強く寿命も長いなど多くの利点を備えているために、現在ある照明器具も LED に置き換えが進んでいくと思われます。

赤色 LED の順方向の電圧降下は 2V 程度です。順方向電流 I_F を 10mA 程度にするには、電流制限抵抗 R は、

$$R = \frac{V_{CC} - V_F}{I_F} = \frac{5.0V - 2.0V}{10 \times 10^{-3} A} = 300\Omega$$

となります。

これより少々大きな抵抗をつなげばよいことが分かります。

LED はポート 8 の P85,P86,P87 に接続します。LED を点滅させるプログラムを例に、I/O ポートからデータを出力する方法を学びましょう。

4.1.3 回路図

すでに説明したように、ポート 8 は大電流ポートで Low レベル出力時 20mA（VOL=1.5V 時）駆動できます。このため、LED は Low レベル出力時 (0 を出力した場合) に点灯になるように接続します (図 4.2)。

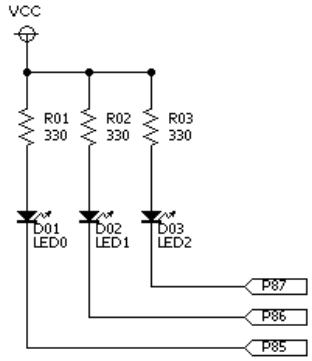

図 4.2 LED の回路図

ポートの HIGH(1)，LOW(0) と LED の点灯・消灯の関係は、以下のようになります (表 4.1)。

表 4.1 ポートデータと LED 点滅の関係

P85	LED0	P86	LED1	P87	LED2
HIGH(1)	消灯	HIGH(1)	消灯	HIGH(1)	消灯
LOW(0)	点灯	LOW(0)	点灯	LOW(0)	点灯

0 で点灯、1 で消灯というのは感覚的に分かりにくく、点灯がシフトするプログラムを作る際など、プログラムもしにくく感じるかもしれません。

必要に応じて、NOT の回路が入った 74HC04 などを使って、1 で点灯、0 で消灯になるように回路を変更すると良いでしょう。その場合には、このテキストの以降のプログラムを適宜変更してください。

4.1.4 接続例

回路図 4.2 に合わせて、ブレッドボード上に回路を構成してください。

図 4.3 に接続した様子を示しておきます。

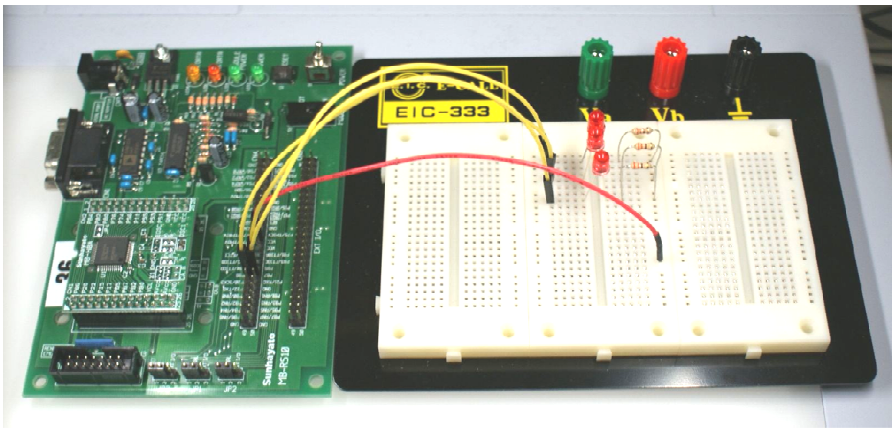

図 4.3　LED の接続例

4.1.5 関連レジスタ

上記回路において、LED の点滅操作に必要なレジスタは以下のとおりです。

- ポートコントロールレジスタ 8(PCR8)
- ポートデータレジスタ 8(PDR8)

I/O レジスタの場所 (アドレス) は決まっていて、上記のレジスタは次のアドレスに割り振られています。

表 4.2　P8 レジスタのアドレス

レジスタ名	アドレス
ポートコントロールレジスタ 8(PCR8)	H'FFEB
ポートデータレジスタ 8(PDR8)	H'FFDB

ポートコントロールレジスタ 8(PCR8)

PCR8 は、ポート 8 を汎用入出力ポートとして使用する場合、端子を入力にするか出力にするかをビットごとに選択するために使用します。

表 4.3　ポートコントロールレジスタ 8(PCR8)

ビット	ビット名	初期値	R/W	説 明
7	PCR87	0	W	ポート 8 が汎用入出力ポートに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	PCR86	0	W	
5	PCR85	0	W	
4	PCR84	0	W	
3	PCR83	0	W	
2	PCR82	0	W	
1	PCR81	0	W	
0	PCR80	0	W	

ポートデータレジスタ 8(PDR8)

PDR8 はポート 8 の汎用入出力ポートデータレジスタです。出力に設定した端子の出力データを設定します。設定した値は保持され、新たに設定しなおすまで変化しません。

表 4.4　ポートデータレジスタ 8

ビット	ビット名	初期値	R/W	説 明
7	P87	0	R/W	汎用出力ポートの出力値を格納します。
6	P86	0	R/W	
5	P85	0	R/W	
4	P84	0	R/W	
3	P83	0	R/W	
2	P82	0	R/W	
1	P81	0	R/W	
0	P80	0	R/W	

4.2　ポインタの機能 (復習)

パソコン上で C 言語のプログラムをする際に、ポインタとはどのような機能を持っていたのかを簡単に復習しておきましょう。

ポインタを用いて変数の値を書き換えるプログラムを作ってみます。

04211pointer01.c

```
 1  /*********************************************************************/
 2  /*                                                                   */
 3  /*  DESCRIPTION :ポインターの機能                                    */
 4  /*  CPU TYPE    :PC                                                  */
 5  /*  NAME        :Ono Yasuji                                          */
 6  /*                                                                   */
 7  /*********************************************************************/
 8
 9  #include<stdio.h>
10
11  int main(void)
12  {
13    unsigned char a=1;
14    unsigned char *p;
15
16    printf("&a=%x\n", &a);
17    printf("&p=%x\n", &p);
18    printf("a=%d\n", a);
19    p=&a;
```

```
20    *p=5;
21    printf("a=%d\n", a);
22    return(0);
23  }
```

End Of List

実行結果 (Borland C++ Compiler 5.5)

>bcc32 04211pointer01.c ↵

```
Borland C++ 5.5.1 for Win32 Copyright (c) 1993, 2000 Borland
04211pointer01.c:
Turbo Incremental Link 5.00 Copyright (c) 1997, 2000 Borland
```

>04211pointer01.exe ↵

```
&a=18ff53
&p=18ff4c
a=1
a=5

>
```

実行結果の

```
&a=18ff53
&p=18ff4c
```

は、コンパイラが変数を割り付けたアドレスですので、自分の環境で実行した場合に必ずしもこれと同じ値になるわけではありません。

プログラム解説 (04211pointer01.c)

```
13    unsigned char a=1;
```

unsigned char 型の変数 a を宣言し、値 1 を代入しています。変数 a はコンパイラによってメモリ上のあいた部分に自動的に割り付けられますが、実行結果からそのアドレスが 0x18ff53 であることが分かります。

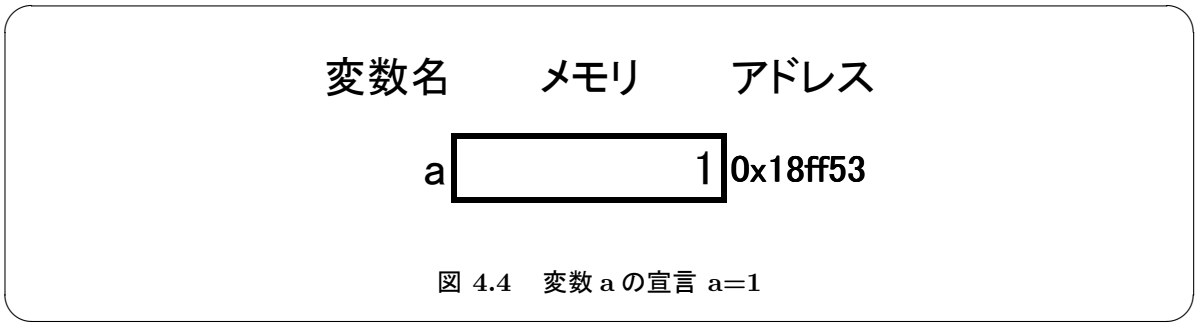

図 4.4　変数 a の宣言 a=1

```
14    unsigned char *p;
```

unsigned char 型のポインタ変数 p を宣言しています。実行結果からそのアドレスが 0x18ff4c であることが分かります。

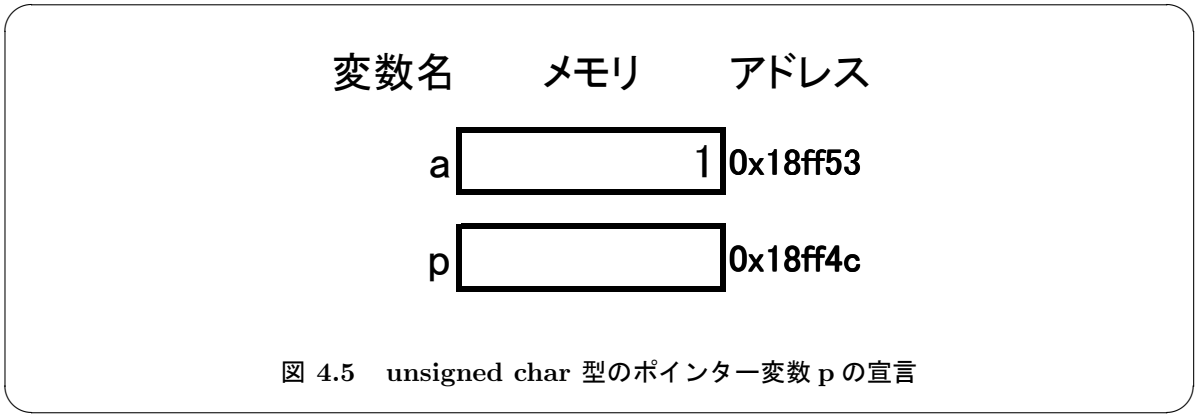

図 4.5 unsigned char 型のポインター変数 p の宣言

```
16    printf("&a=%x\n", &a);
17    printf("&p=%x\n", &p);
18    printf("a=%d\n", a);
```

16 行目では unsigned char 型の変数 a のアドレスを、17 行目では unsigned char 型のポインタ変数 p のアドレスを 16 進数で表示しています。18 行目では変数 a の値を表示しています。今の場合、13 行目で代入した 1 が表示されます。

```
19    p=&a;
```

unsigned char 型のポインタ変数 p に変数 a のアドレスを代入しています。変数の前に&を付けると、その変数のアドレスを意味します。今の場合ポインタ変数 p には変数 a のアドレス 0x18ff53 が代入されます。

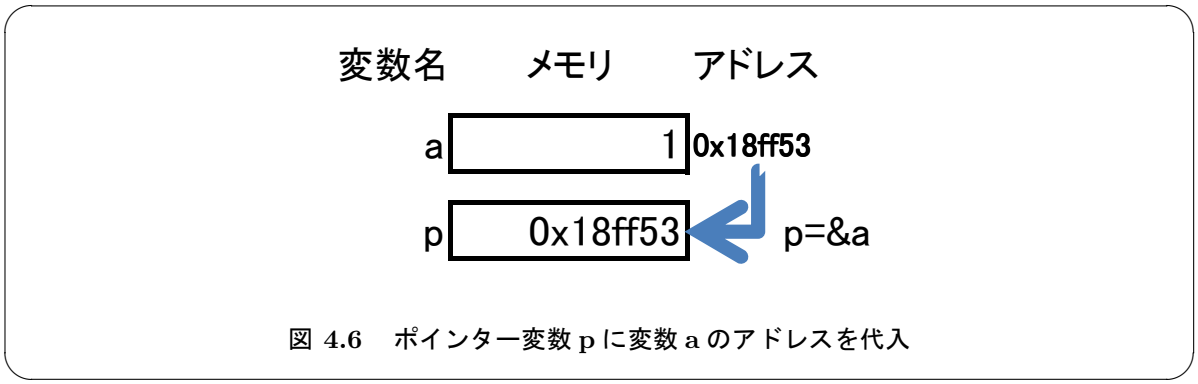

図 4.6 ポインター変数 p に変数 a のアドレスを代入

```
20    *p=5;
```

unsigned char 型のポインタ変数 p に代入されたアドレスのメモリを書き換えています。ポインタ変数の前に*を付けると、そのポインタ変数に格納されたアドレスのメモリを意味します。p には変数 a のアドレスが代入されているので、変数 a の値を書き換えることになります。

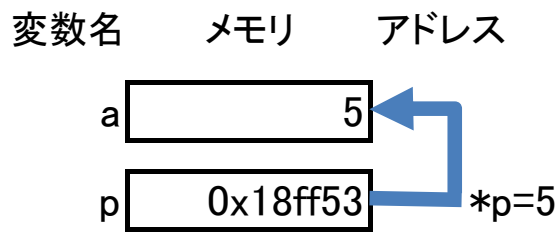

図 4.7　変数 a の値の書き換え

```
21    printf("a=%d\n", a);
```

20 行目で変数 a の値を書き換えたので、結果は 5 になります。

このように、ポインタ変数というのはアドレスを格納して、そのアドレスのメモリの中身を操作することができる変数でした。

今プログラムしたいのは、マイコンの I/O レジスタの設定を変更することです。I/O レジスタはアドレスが分かっています (P.92 表 4.2)。その中身を操作するのにはポインタを使えばよいことになります。

4.3　LED 点灯プログラム (ポインタ変数を用いて)

まずは、P.92 表 4.2 の I/O レジスタのアドレスを用いて LED0(P85 に接続されている LED) を点灯し、他の二つの LED を消灯するプログラムを作りましょう。。

01_LED01.c

```
 1  /*********************************************************************/
 2  /*                                                                   */
 3  /*  FILE        :01_LED01.c                                          */
 4  /*  DESCRIPTION :LED0 の点灯 (ポインタの利用)                        */
 5  /*  CPU TYPE    :H8/3694F                                            */
 6  /*                                                                   */
 7  /*  LED0 P85                                                         */
 8  /*  LED1 P86                                                         */
 9  /*  LED2 P87                                                         */
10  /*                                                                   */
11  /*  This file is generated by Renesas Project Generator (Ver.4.9).   */
12  /*                                                                   */
13  /*********************************************************************/
14
15  void main(void)
16  {
17      volatile unsigned char *pcr8, *pdr8;
18
19    pcr8 =(volatile unsigned char *)0xFFEB;
20    pdr8 =(volatile unsigned char *)0xFFDB;
21
```

```
22    /*********************************************
23      LED 接続端子の初期化
24    *********************************************/
25    *pcr8 =0xE0;
26
27    /*********************************************
28      LED0 を点灯他は消灯
29    *********************************************/
30    *pdr8 =(~0x20);
31
32    /*********************************************
33      無限ループ
34    *********************************************/
35    while(1){
36      ;
37    }
38  }
```

End Of List

実行結果

LED0(P85 に接続されている LED) が点灯し、他の二つの LED は消灯する。

プログラム解説 (01_LED01.c)

```
17    volatile unsigned char *pcr8, *pdr8;
```

unsigned char 型のポインタ変数 pcr8 と pdr8 を宣言しています。ポートコントロールレジスタ 8(PCR8) もポートデータレジスタ 8(PDR8) も 1 バイトで、正の値のみですから、unsigned char 型のポインタを使います。

ポインタ変数 pcr8 と pdr8 の型宣言には、unsigned char の前に volatile というのが付いています。

コンパイラは最適化という作業を行うことがあります。これは、プログラムを変更することによって、実行結果に影響を与えず実行ファイルのサイズを小さくし、実行速度を早くするという作業です。

ところが、場合によっては最適化によって実行結果に影響が出てしまうことがあります。たとえば、

```
*pdr8 =0x00;
*pdr8 =(~0x20);
*pdr8 =0x00;
```

というプログラムがあったとします。pdr8 はポートデータレジスタ 8 で、ポートにつながっている周辺機器を制御しているとしましょう。この場合 3 回データを送ることが必要で、省略できないとします。

ところが、コンパイラは結局最後は 0 に設定しているのなら、途中の操作を省略して 0 に設定するだけでよいと判断するかもしれません。このような場合には思った制御ができなくなってしまいます。

そこで、最適化を行わないように volatile を追加するのです。

HEW では外部変数に対して「外部変数の volatile 化」の設定ができますが、移植性を考えて本テキストではプログラム内に明示的に記述することにしています。

```
19    pcr8 =(volatile unsigned char *)0xFFEB;
20    pdr8 =(volatile unsigned char *)0xFFDB;
```

19 行目では、unsigned char 型のポインタ変数 pcr8 に、ポートコントロールレジスタ 8(PCR8) のアドレスを代入しています。同様に 20 行目では、unsigned char 型のポインタ変数 pdr8 に、ポートデータレジスタ 8(PDR8) のアドレスを代入しています。

```
pcr8 =0xFFEB;
```

のように、数値を直接代入してしまうと、アドレスであるということが分かりません。そこで、unsigned char 型のポインタにキャストして、アドレスであることを明示しています。

```
25    *pcr8 =0xE0;
```

ポートコントロールレジスタ 8(PCR8) に 16 進数の値 0xE0 を設定しています。ポート 8 の P85,P86,P87 に LED がつながっているので、ここを出力に設定します。他は 0 に設定することにし、$(11100000)_2$ を 16 進数に直した 0xE0 にしました。LED がつながっていないビットを 0 にしたのは、ポートコントロールレジスタ 8 の初期値が 0 だからですが、後ほど LED に関係ないビットは設定しないように修正したいと思います。

```
30    *pdr8 =(~0x20);
```

ポートデータレジスタ 8(PDR8) に値 (~0x20) を設定しています。値を 0 にすると点灯、1 にすると消灯だということを再度確認しておきましょう。ただし、感覚的には 1 が点灯 0 が消灯としたほうが分かりやすそうなので、LED0(P85) のみを点灯させるためのデータを、$(00100000)_2$ を反転させた (~0x20) で書きました。もちろん、0xDF と書いてもかまいません。

```
35    while(1){
36        ;
37    }
```

while(1) が無限ループになっていることに注意してください。

パソコン用のプログラムでは、main 関数の最後は return 文でした。これはプログラムが終了したら制御を OS に返すことを意味しています。しかし、今の場合マイコンに OS は入っていませんから、制御を戻すとどのような動作になるか分かりません。場合によってはリセットがかかったり暴走したりすることになります。

OS を導入しない組込みマイコンの場合には、最後に return 文を書かずに、プログラムが終了しないような処理をします。

何かの動作を永遠に繰り返すか、ここでのように、何もせずに無限ループを実行するかです。

4.4 LED1 の点灯 (キャストの利用)

プログラム 01_LED01.c ではポインタ変数を用いましたが、パソコンの場合とは違って、レジスタのアドレスは最初から分かっています。

ですから、わざわざ変数を宣言しなくても、そのアドレスに直接値を入れてしまえばよいわけです。アドレスが単なる数値ではないということを明示するために、unsigned char 型のポインタにキャストする必要があります。

プログラムの変更が分かるように、今回は LED1 を点灯するプログラムにして、プログラム 01_LED01.c を書き変えてみましょう。

01_LED02.c

```
 1  /*********************************************************************/
 2  /*                                                                   */
 3  /*  FILE        :01_LED02.c                                          */
 4  /*  DESCRIPTION :LED1 の点灯 (キャストの利用)                        */
 5  /*  CPU TYPE    :H8/3694F                                            */
 6  /*                                                                   */
 7  /*  LED0 P85                                                         */
 8  /*  LED1 P86                                                         */
 9  /*  LED2 P87                                                         */
10  /*                                                                   */
11  /*  This file is generated by Renesas Project Generator (Ver.4.9).   */
12  /*                                                                   */
13  /*********************************************************************/
14
15  void main(void)
16  {
17    /*********************************************
18      LED 接続端子の初期化
19    *********************************************/
20    (*((volatile unsigned char *)0xFFEB)) = 0xE0;
21
22    /*********************************************
23      LED1 を点灯他は消灯
24    *********************************************/
25    (*((volatile unsigned char *)0xFFDB)) = (~0x40);
26
27    /*********************************************
28      無限ループ
29    *********************************************/
30    while(1){
31      ;
32    }
33  }
```

End Of List

実行結果

LED1(P86 に接続されている LED) が点灯し、他の二つの LED は消灯する。

プログラム解説 (01_LED02.c)

```
20    (*((volatile unsigned char *)0xFFEB)) = 0xE0;
```

ポートコントロールレジスタ 8(PCR8) に値 0xE0 を代入しています。PCR8 のアドレス 0xFFEB を明示的に (volatile unsigned char *) でキャストしています。

```
25    (*((volatile unsigned char *)0xFFDB)) = (~0x40);
```

ポートデータレジスタ 8(PDR8) に値 (~0x40) を代入しています。PDR8 のアドレス 0xFFDB を明示的に (volatile unsigned char *) でキャストしています。

前述のように、1 が点灯になるように値を記述し、反転させました。

4.5 LED2 の点灯 (#define 文の利用)

01_LED02.c では、キャストを利用して、ポインタ変数を用いずにレジスタに値を代入しました。

しかし、レジスタを利用するたびに (volatile unsigned char *) と書いていたのでは煩雑です。そこで、レジスタを操作するための (キャストを含めた) アドレスを、最初に他の名前で再定義して使うことが考えられます。

たとえば、(*((volatile unsigned char *)0xFFEB)) を PCR8 という名前で再定義して、ポートコントロールレジスタ 8(PCR8) に値を代入する場合にはこれを使うようにするわけです。

この方針でプログラムを再度書きなおしてみましょう。今回は LED2 だけを点灯させてみます。

01_LED03.c

```
 1  /************************************************************************/
 2  /*                                                                      */
 3  /*  FILE        :01_LED03.c                                             */
 4  /*  DESCRIPTION :LED2 の点灯 (#define 文の利用)                         */
 5  /*  CPU TYPE    :H8/3694F                                               */
 6  /*                                                                      */
 7  /*  LED0 P85                                                            */
 8  /*  LED1 P86                                                            */
 9  /*  LED2 P87                                                            */
10  /*                                                                      */
11  /*  This file is generated by Renesas Project Generator (Ver.4.9).      */
12  /*                                                                      */
13  /************************************************************************/
14
15  #define PCR8 (*((volatile unsigned char *)0xFFEB))
16  #define PDR8 (*((volatile unsigned char *)0xFFDB))
17
18  void main(void)
19  {
20    /*********************************************
21      LED 接続端子の初期化
22    *********************************************/
23    PCR8 = 0xE0;
24
25    /*********************************************
26      LED2 を点灯他は消灯
27    *********************************************/
28    PDR8 = (~0x80);
29
30    /*********************************************
31    無限ループ
32    *********************************************/
33    while(1){
34      ;
35    }
36  }
```

End Of List

実行結果

LED2(P87 に接続されている LED) が点灯し、他の二つの LED は消灯する。

プログラム解説 (01_LED03.c)

```
15    #define PCR8 (*((volatile unsigned char *)0xFFEB))
16    #define PDR8 (*((volatile unsigned char *)0xFFDB))
```

15 行目で (*((volatile unsigned char *)0xFFEB)) を PCR8 という名前に、16 行目で (*((volatile unsigned char *)0xFFDB)) を PDR8 という名前に定義しています。以後 PCR8 は (*((volatile unsigned char *)0xFFEB)) に置き換えられます。

なお、(*((volatile unsigned char *)0xFFEB)) は全体をカッコでくくっていることにも注意してください。

```
23    PCR8 = 0xE0;
```

15 行目で (*((volatile unsigned char *)0xFFEB)) を PCR8 という名前に定義しました。したがって、この 23 行目は (*((volatile unsigned char *)0xFFEB))=0xE0 に置き換えられます。

01_LED02.c の 20 行目と比較してください。全く同じであることが分かります。

```
28    PDR8 = (~0x80);
```

16 行目で (*((volatile unsigned char *)0xFFDB)) を PDR8 という名前に定義しました。したがって、この 28 行目は (*((volatile unsigned char *)0xFFDB))=(~0x80) に置き換えられます。

01_LED02.c の 25 行目と比較してください。

4.6 LED0,1 の点灯 (不要なレジスタは変更しない)

さて、では不要なレジスタは変更しないように書きなおしてみましょう。この場合、0 に設定する演算と、1 に設定する演算の 2 回が必要になるのでした。

今の場合、分かりやすいように値が 1 のビットを点灯、0 のビットを消灯になるように書いてみましょう。LED0 を点灯させるために、0x20 という値を用いてプログラムをするわけです。

不要なビットをマスクし、0 のビットを消灯 (1 に設定) に、1 のビットを点灯 (0 に設定) にするわけですから、P.55 の表 2.9 の演算を行えばよいことが分かります。

01_LED03_2.c

```
1  /*********************************************************************/
2  /*                                                                   */
3  /*  FILE        :01_LED03_2.c                                        */
4  /*  DESCRIPTION :LED0 の点灯 (不要なレジスタは変更しない)              */
5  /*  CPU TYPE    :H8/3694F                                            */
6  /*                                                                   */
7  /*  LED0 P85                                                         */
8  /*  LED1 P86                                                         */
```

```
 9  /*  LED2 P87                                                          */
10  /*                                                                    */
11  /*  This file is generated by Renesas Project Generator (Ver.4.9).    */
12  /*                                                                    */
13  /**********************************************************************/
14
15  #define PCR8 (*((volatile unsigned char *)0xFFEB))
16  #define PDR8 (*((volatile unsigned char *)0xFFDB))
17
18  #define LED_MASK 0xE0
19
20  void main(void)
21  {
22      /*********************************************
23        LED 接続端子の初期化
24      *********************************************/
25      PCR8 |= LED_MASK;
26
27      /*********************************************
28        LED0 を点灯
29      *********************************************/
30      PDR8 &= (~(0x20 & LED_MASK)); /* 0 をセット */
31      PDR8 |= ((~0x20) & LED_MASK); /* 1 をセット */
32
33      /*********************************************
34        無限ループ
35      *********************************************/
36      while(1){
37          ;
38      }
39  }
```

End Of List

実行結果

LED0(P85 に接続されている LED) が点灯し、他の二つの LED は消灯する。

プログラム解説 (01_LED03_2.c)

```
18    #define LED_MASK 0xE0
```

マスクの値を分かりやすいように LED_MASK と定義しなおしました。LED がつながっている 3 ビットのみ 1 にしています。

```
25    PCR8 |= LED_MASK;
```

18 行目の定義を用いてポートコントロールレジスタ 8(PCR8) の設定をしています。LED が接続されたビットのみを出力に設定しています。

```
30    PDR8 &= (~(0x20 & LED_MASK)); /* 0 をセット */
31    PDR8 |= ((~0x20) & LED_MASK); /* 1 をセット */
```

30 行目で P85 を 0 に設定しています。31 行目では P86 と P87 を 1 に設定しています。分からない場合には第 2 章 (P.11 まで) を復習してください。

さて、こうして書きなおしてみましたが、LED が右端のビット (0 ビット目) からつながっていない場合には、シフトを使ってより分かりやすくすることができます。

具体的には、上では LED0 を点灯させるのに値 0x20 を用いていますが、シフトを用いれば値 0x01 を用いることができます。こちらのほうが直感的に分かりやすく、ミスも少なくなるのではないかと思います。

もちろん、このような演算がより重要になる場面は、関数を用いたときです。これについてはまたいずれ説明します。

それでは、プログラム 01_LED03_2.c をシフトを用いて書きなおしてみましょう。

01_LED04.c

```
 1  /*********************************************************************/
 2  /*                                                                   */
 3  /*  FILE        :01_LED04.c                                          */
 4  /*  DESCRIPTION :LED0,1 の点灯 (不要なレジスタは変更しない)          */
 5  /*  CPU TYPE    :H8/3694F                                            */
 6  /*                                                                   */
 7  /*  LED0 P85                                                         */
 8  /*  LED1 P86                                                         */
 9  /*  LED2 P87                                                         */
10  /*                                                                   */
11  /*  This file is generated by Renesas Project Generator (Ver.4.9).   */
12  /*                                                                   */
13  /*********************************************************************/
14
15  #define PCR8 (*((volatile unsigned char *)0xFFEB))
16  #define PDR8 (*((volatile unsigned char *)0xFFDB))
17
18  #define LED_SHIFT 5
19  #define LED_MASK (0x07 << LED_SHIFT)
20
21  void main(void)
22  {
23    /*********************************************
24      LED 接続端子の初期化
25    *********************************************/
26    PCR8 |= LED_MASK;
27
28    /*********************************************
29      LED0,1 を点灯 LED2 は消灯
30    *********************************************/
31    PDR8 &= (~((0x03 << LED_SHIFT) & LED_MASK)); /* 0 をセット */
32    PDR8 |= ((~(0x03 << LED_SHIFT)) & LED_MASK); /* 1 をセット */
33
34    /*********************************************
35      無限ループ
36    *********************************************/
37    while(1){
38      ;
39    }
40  }
```

End Of List

実行結果

LED0,1(P85、P86 に接続されている LED) が点灯し、LED2(P87 に接続されている LED) は消灯する。

プログラム解説 (01_LED04.c)

```
18    #define LED_SHIFT 5
```

シフトするビット数 5 を LED_SHIFT と定義しました。シフトを用いることで、LED が一番右のビット (0 ビット目) からつながっていなくても、0 ビット目からつながっていると思ってプログラムを組むことができるようになります。

```
19    #define LED_MASK (0x07<<LED_SHIFT)
```

18 行目の LED_SHIFT を用いて、マスクの値も LED が 0 ビット目からつながっていると思って、プログラムすることができるようになります。LED_MASK の値は 01_LED03_2.c と同様に 0xE0 になります。自分で実際にシフトして確認してみましょう。

```
26    PCR8 |= LED_MASK;
```

P.30 の 2.5.6 「必要なビットだけ 1 に設定し、他は変えない (OR)」で、論理和 (OR) は必要なビットだけ 1 に設定するという説明をしました。ここでは、LED_MASK を用いて LED がつながったビットだけ 1(出力) に設定しています。

```
31    PDR8 &= (~((0x03<<LED_SHIFT) & LED_MASK)); /* 0 をセット */
32    PDR8 |= ((~(0x03<<LED_SHIFT)) & LED_MASK); /* 1 をセット */
```

P.56 の表 2.10 にまとめたように、与えられた値をシフトし、必要な部分にマスクをかけて、0 のビットに対応するレジスタを 1 に、1 のビットに対応するレジスタを 0 に設定する演算です。

4.7 LED0,2 の点滅 (空ループ版)

ここまでは、ある LED を点灯させたり消灯させたりということを行ってきました。

しかし、点灯消灯は最初に設定されたパターンから変化はしませんでした。

ここでは、LED の点滅のパターンが変化するプログラムを書いてみましょう。そのためには無限ループの中に LED 点滅のプログラムを書く必要があります。

また、マイコンの処理は早いので、人間の目で見てわかるように、点灯や消灯をある程度の時間保つ必要があります。この目的のために、ここでは何もしないループをまわして時間稼ぎをすることにしました。

では、以下に LED 点滅のサンプルプログラムを紹介します。

01_LED05.c

```
 1  /*********************************************************************/
 2  /*                                                                   */
 3  /*  FILE        :01_LED05.c                                          */
 4  /*  DESCRIPTION :LED0,2 の点滅 (空ループ版)                          */
 5  /*  CPU TYPE    :H8/3694F                                            */
 6  /*                                                                   */
 7  /*  LED0 P85                                                         */
 8  /*  LED1 P86                                                         */
 9  /*  LED2 P87                                                         */
10  /*                                                                   */
11  /*  This file is generated by Renesas Project Generator (Ver.4.9).   */
12  /*                                                                   */
13  /*********************************************************************/
14
15  #define PCR8 (*((volatile unsigned char *)0xFFEB))
16  #define PDR8 (*((volatile unsigned char *)0xFFDB))
17
18  #define LED_SHIFT 5
19  #define LED_MASK (0x07 << LED_SHIFT)
```

```
20
21  void main(void)
22  {
23    unsigned long count;
24
25    /*********************************************
26      LED 接続端子の初期化
27    *********************************************/
28    PCR8 |= LED_MASK;
29
30    while(1){
31      /*********************************************
32        LED0 を点灯 LED1,2 は消灯
33      *********************************************/
34      PDR8 &= (~((0x01 << LED_SHIFT) & LED_MASK)); /* 0 をセット */
35      PDR8 |= ((~(0x01 << LED_SHIFT)) & LED_MASK); /* 1 をセット */
36      for(count=0; count<0x3FFFFFL; count++){
37        ;
38      }
39      /*********************************************
40        LED2 を点灯 LED0,1 は消灯
41      *********************************************/
42      PDR8 &= (~((0x04 << LED_SHIFT) & LED_MASK)); /* 0 をセット */
43      PDR8 |= ((~(0x04 << LED_SHIFT)) & LED_MASK); /* 1 をセット */
44      for(count=0; count<0x3FFFFFL; count++){
45        ;
46      }
47    }
48  }
```

End Of List

実行結果

最初は LED0 だけが点灯。
その後 LED0 と LED2 が交互に点滅する。

プログラム解説 (01_LED05.c)

```
34    PDR8 &= (~((0x01<<LED_SHIFT) & LED_MASK)); /* 0 をセット */
35    PDR8 |= ((~(0x01<<LED_SHIFT)) & LED_MASK); /* 1 をセット */
```

P85,P86,P87 を $(110)_2$ に設定しています。LED0 のみ点灯します。

```
36    for(count=0; count<0x3FFFFFL; count++){
37        ;
38    }
```

何もしないループを回して時間を稼いでいます。マイコンの処理は 1 秒間に何百万回と行われます。この早さで LED を点滅させても、我々は点滅を感知することができません。

そこで、点滅が分かるように何分の 1 秒かの「待ち」を作る必要があります。それをやっているのがこのループです。

ただし、C 言語で書かれたプログラムはどのような実行ファイルに翻訳されるか分かりませんので、正確な待ち時間を作ることは困難です。

そのような目的にはタイマを利用します。

```
42    PDR8 &= (~((0x04<<LED_SHIFT) & LED_MASK)); /* 0 をセット */
43    PDR8 |= ((~(0x04<<LED_SHIFT)) & LED_MASK); /* 1 をセット */
```

P85,P86,P87 を $(011)_2$ に設定しています。LED2 のみ点灯します。

4.8 LED1,2 の点滅 (関数化など)

プログラム 01_LED05.c では、時間稼ぎのための待ちループが 2 度出てきました (36 行目から 38 行目までと、44 行目から 46 行目まで)。

このように同じプログラムを複数回利用する場合には、関数にして必要な場所で呼び出すほうがプログラムも小さくなりますし、デバッグもしやすくなります。

以下では排他的論理和 (XOR) を用いて LED の点滅を実現しました。そのため、待ちループを 2 度書く必要がなくなり、一つになりました。関数を複数回使うことはなくなりますが、関数化することで見通しは良くなるのではないでしょうか。

関数化してみましょう。

01_LED06.c

```
 1  /*********************************************************************/
 2  /*                                                                   */
 3  /*  FILE        :01_LED06.c                                          */
 4  /*  DESCRIPTION :LED1,2 の点滅 (関数化など)                          */
 5  /*  CPU TYPE    :H8/3694F                                            */
 6  /*                                                                   */
 7  /*  LED0 P85                                                         */
 8  /*  LED1 P86                                                         */
 9  /*  LED2 P87                                                         */
10  /*                                                                   */
11  /*  This file is generated by Renesas Project Generator (Ver.4.9).   */
12  /*                                                                   */
13  /*********************************************************************/
14
15  #define PCR8 (*((volatile unsigned char *)0xFFEB))
16  #define PDR8 (*((volatile unsigned char *)0xFFDB))
17
18  #define LED_SHIFT 5
19  #define LED_MASK (0x07 << LED_SHIFT)
20  #define LED12 (0x06 << LED_SHIFT)
21
22  #define WAIT_LOOP 0x3FFFFFL
23
24  void WaitLoop(void)
25  {
26    unsigned long count;
27
28    for(count=0; count<WAIT_LOOP; count++){
29      ;
30    }
31  }
32
33  void main(void)
34  {
35
36    /*********************************************
37      LED 接続端子の初期化
38    *********************************************/
39    PCR8 |= LED_MASK;
40
41    /*********************************************
42      LED1 を点灯 LED0,2 は消灯
43    *********************************************/
44    PDR8 &= (~((0x02 << LED_SHIFT) & LED_MASK));  /* 0 をセット */
45    PDR8 |= ((~(0x02 << LED_SHIFT)) & LED_MASK); /* 1 をセット */
46
47    while(1){
48      WaitLoop();
49      PDR8 ^= LED12;  /* LED1,2 を反転 */
```

```
50    }
51  }
```

End Of List

実行結果

最初は LED1 だけが点灯する。
LED1 と LED2 が交互に点滅する。

プログラム解説 (01_LED06.c)

```
20    #define LED12 (0x06 << LED_SHIFT)
```

今回は LED の点滅を排他的論理和 (XOR) を使って演算します (P.35 の 2.5.9 「必要なビットだけ値を反転させる、他は変えない (XOR)」参照)。

そのための値を LED12 という名前で定義しました。LED12 は 0xC0(11000000_2) です。

```
22    #define WAIT_LOOP 0x3FFFFFL
```

ループの回数も WAIT_LOOP という名前で定義しました。この部分を変更すれば点滅の速度が変えられます。

```
24    void WaitLoop(void)
25    {
26      unsigned long count;
27
28      for(count=0; count<WAIT_LOOP; count++){
29        ;
30      }
31    }
```

無駄ループの関数本体です。WaitLoop という名前にしました。

呼び出し元の main 関数の前に書いたので、プロトタイプ宣言がなくても利用できます。

```
48    WaitLoop();
```

関数 WaitLoop を呼び出しています。

引数も戻り値もないので、このような呼び出しになります。

```
49    PDR8 ^= LED12;  /* LED1,2 を反転 */
```

LED12 は 0xC0(11000000_2) なので、P86、P87 を反転させることになります (P.35 の 2.5.9 「必要なビットだけ値を反転させる、他は変えない (XOR)」参照)。

4.9 LED のカウントアップ

三つの LED を 3 桁の 2 進数とみなし、消灯を 0、点灯を 1 として、$(000)_2$ から $(111)_2$ まで 1 ずつ表示が増えていくプログラムを作りましょう。

プログラムの書き方は何通りもありますが、ここでは変数 led_data と for ループを使って、以下のようなプログラムを書いてみました。

01_LED07.c

```
 1  /************************************************************************/
 2  /*                                                                      */
 3  /*  FILE        :01_LED07.c                                             */
 4  /*  DESCRIPTION :LED のカウントアップ                                   */
 5  /*  CPU TYPE    :H8/3694F                                               */
 6  /*                                                                      */
 7  /*  LED0 P85                                                            */
 8  /*  LED1 P86                                                            */
 9  /*  LED2 P87                                                            */
10  /*                                                                      */
11  /*  This file is generated by Renesas Project Generator (Ver.4.9).      */
12  /*                                                                      */
13  /************************************************************************/
14
15  #define PCR8 (*((volatile unsigned char *)0xFFEB))
16  #define PDR8 (*((volatile unsigned char *)0xFFDB))
17
18  #define LED_SHIFT 5
19  #define LED 0x07
20  #define LED_MASK (LED << LED_SHIFT)
21
22  #define WAIT_LOOP 0x3FFFFFL
23
24  void WaitLoop(void)
25  {
26    unsigned long count;
27
28    for(count=0; count<WAIT_LOOP; count++){
29      ;
30    }
31  }
32
33  void main(void)
34  {
35    unsigned char led_data=0;
36
37    /*******************************************
38      LED 接続端子の初期化
39    *******************************************/
40    PCR8 |= LED_MASK;
41
42    /*******************************************
43      LED はすべて消灯
44    *******************************************/
45    PDR8 &= (~((led_data<<LED_SHIFT) & LED_MASK)); /* 0 をセット */
46    PDR8 |= ((~(led_data<<LED_SHIFT)) & LED_MASK); /* 1 をセット */
47
48    while(1){
49      for(led_data=0; led_data<=LED; led_data++){
50        PDR8 &= (~((led_data<<LED_SHIFT) & LED_MASK)); /* 0 をセット */
51        PDR8 |= ((~(led_data<<LED_SHIFT)) & LED_MASK); /* 1 をセット */
52        WaitLoop();
53      }
54    }
55  }
```

End Of List

実行結果

最初はすべての LED が消灯。
LED の表示がカウントアップしていく。$(111)_2$ までいったら、$(000)_2$ に戻る。

プログラム解説 (01_LED07.c)

```
19    #define LED 0x07
```

LED がつながっているビット数だけ 1 を立てています。シフトをするとマスクのための値になります。

ここでは、変数 led_data の最大値としても使っています。

```
35    unsigned char led_data=0;
```

この変数を用いてレジスタを設定します。変数 led_data の値は最下位 3 ビット分が LED の点滅を表すデータになっています。

これをビット操作して、対応するレジスタを適切に設定します。

```
45    PDR8 &= (~((led_data<<LED_SHIFT) & LED_MASK)); /* 0 をセット */
46    PDR8 |= ((~(led_data<<LED_SHIFT)) & LED_MASK); /* 1 をセット */
```

これは LED の初期状態を設定しています。この 2 行はなくても動作は変わりません。

しかし、初期状態を明示するためにあえて入れておきました。実際に削って動作が変わらないことを確認してみてください。

```
49    for(led_data=0; led_data<=LED; led_data++){
50      PDR8 &= (~((led_data<<LED_SHIFT) & LED_MASK)); /* 0 をセット */
51      PDR8 |= ((~(led_data<<LED_SHIFT)) & LED_MASK); /* 1 をセット */
52      WaitLoop();
53    }
```

for ループを使って、led_data が 0 のときから、7 になるまで値を 1 ずつ増やしていきます。50 行目と 51 行目でその値をもとにレジスタの設定を行っています。52 行目は待ちループです。

課題 4.9.1　(提出) LED の点灯プログラム

3 つの LED が点灯の状態から、LED2 が消え、LED2 と LED1 が消え、すべてが消灯し、すべて点灯の状態に戻るプログラムを作ってください。

課題 4.9.1　(提出) LED の点灯プログラム (続き)

表 4.5　LED の点滅の仕方 (○:点灯、●:消灯)

LED0	LED1	LED2
○	○	○
○	○	●
○	●	●
●	●	●

プロジェクト名：e01_LED07

4.10　LED 点灯の左シフト

LED0 だけ点灯した状態から、LED1 だけ点灯、LED2 だけ点灯となり、LED0 だけ点灯した状態に戻るプログラムを、シフト演算を用いてプログラムしてみましょう。

表 4.6　LED の点滅の仕方 (○:点灯、●:消灯)

LED0	LED1	LED2
○	●	●
●	○	●
●	●	○

今回も、以下の例以外にも回答はいくつも存在します。ここであげたサンプル以上に分かりやすいプログラムを考えてみるとよいでしょう。

01_LED08.c

```
 1  /************************************************************************/
 2  /*                                                                      */
 3  /*  FILE        :01_LED08.c                                             */
 4  /*  DESCRIPTION :LED 点灯の左シフト                                     */
 5  /*  CPU TYPE    :H8/3694F                                               */
 6  /*                                                                      */
 7  /*  LED0 P85                                                            */
 8  /*  LED1 P86                                                            */
 9  /*  LED2 P87                                                            */
10  /*                                                                      */
11  /*  This file is generated by Renesas Project Generator (Ver.4.9).      */
12  /*                                                                      */
13  /************************************************************************/
14
15  #define PCR8 (*((volatile unsigned char *)0xFFEB))
16  #define PDR8 (*((volatile unsigned char *)0xFFDB))
17
18  #define LED_SHIFT 5
19  #define LED 0x07
20  #define LED_MASK (LED << LED_SHIFT)
21
22  #define WAIT_LOOP 0x3FFFFFL
23
24  void WaitLoop(void)
25  {
26    unsigned long count;
27
28    for(count=0; count<WAIT_LOOP; count++){
```

```
29      ;
30    }
31  }
32
33  void main(void)
34  {
35    unsigned char led_data=0x01;
36
37    /*********************************************
38      LED 接続端子の初期化
39    *********************************************/
40    PCR8 |= LED_MASK;
41
42    /*********************************************
43      LED1 のみ点灯
44    *********************************************/
45    PDR8 &= (~((led_data<<LED_SHIFT) & LED_MASK)); /* 0 をセット */
46    PDR8 |= ((~(led_data<<LED_SHIFT)) & LED_MASK); /* 1 をセット */
47
48    while(1){
49      for(led_data=0x01; led_data<LED; led_data<<=1){
50        PDR8 &= (~((led_data<<LED_SHIFT) & LED_MASK)); /* 0 をセット */
51        PDR8 |= ((~(led_data<<LED_SHIFT)) & LED_MASK); /* 1 をセット */
52        WaitLoop();
53      }
54    }
55  }
```

End Of List

実行結果

- LED0 だけ点灯 (○ ● ●)
- LED1 だけ点灯 (● ○ ●)
- LED2 だけ点灯 (● ● ○)
- 最初に戻って以下繰り返し

プログラム解説 (01_LED08.c)

```
35    unsigned char led_data=0x01;
```

unsigned char 型の変数 led_data を宣言しています。初期値は LED0 のみ点灯する値を設定しています。

```
49    for(led_data=0x01; led_data<LED; led_data<<=1){
```

LED は 0x07 なので、このループは

- led_data=0x01
- led_data=0x02
- led_data=0x04
- led_data=0x08 になるので for ループを一旦抜ける

という動作になります。その後、while(1) で再度 for ループを繰り返します。

課題 4.10.1　(提出) LED のシフトプログラム

左シフトして、LED2 が点灯した状態の後、右シフトをして LED の点灯が行ったりきたりするプログラムを作ってください。

具体的に書き下すと

- LED0 のみ点灯 (○ ● ●)
- LED1 のみ点灯 (● ○ ●)
- LED2 のみ点灯 (● ● ○)
- LED1 のみ点灯 (● ○ ●)
- 最初に戻って以下繰り返し

となります。

プロジェクト名：e01_LED08

4.11　LED0,2 の点滅 (関数化)

ここまでのプログラムで、レジスタを設定する演算は必ず出てくることが分かります。具体的には、

```
PDR8 &= (~((led_data<<LED_SHIFT) & LED_MASK)); /* 0 をセット */
PDR8 |= ((~(led_data<<LED_SHIFT)) & LED_MASK); /* 1 をセット */
```

というプログラムです。

毎回この長い演算を入力するのは手間ですし、プログラムも見づらくなります。

そこで、LED 関連の機能を関数としてまとめておきましょう。

関数の作り方も必ずしも一つではないでしょう。

ここでは、以下のような関数を作ることにしました。

```
LED 関連関数 (H8/3694F)

LED0 P85
LED1 P86
LED2 P87

*****************************************************************
void LedInit(void);

    形式  : void LedInit(void);
    機能  :LED で使用する I/O ポートを初期化する関数。
           初期状態は LED すべて消灯。
           LED を使用する前に実行すること。
    引数  :なし
    戻り値:なし

*****************************************************************
void LedOn(unsigned char led_data);

    形式  : void LedOn(unsigned char led_data);
    機能  :1 で指定された LED を点灯する。
           ビットパターンで同時に複数の LED を指定できる。
           1 で指定された LED 以外 (0 で指定された LED) には変化が無い。
    引数  :unsigned char led_data
           点灯する LED の指定 (LED0 or LED1 or LED2)
           下位 3 ビット分のみ有効。
    戻り値:なし

*****************************************************************
void LedOff(unsigned char led_data);

    形式  : void LedOff(unsigned char led_data);
    機能  :1 で指定された LED を消灯する。
           ビットパターンで同時に複数の LED を指定できる。
           1 で指定された LED 以外 (0 で指定された LED) には変化が無い。
    引数  :unsigned char led_data
           消灯する LED の指定 (LED0 or LED1 or LED2)
           下位 3 ビット分のみ有効。
    戻り値:なし

*****************************************************************
void LedSet(unsigned char led_data);

    形式  : void LedSet(unsigned char led_data);
    機能  :1 で指定された LED を点灯し、
           0 で指定された LED を消灯する。
           ビットパターンで同時に 3 ビット分の LED を指定する。
    引数  :unsigned char led_data
           点灯する LED の指定 (LED0 or LED1 or LED2)
           下位 3 ビット分のみ有効。
    戻り値:なし
```

この仕様に沿ってプログラムを関数化してみました。

各関数をそれを呼び出しているプログラムの前におき、プロトタイプ宣言をしていません。

LED の個数はともかくとして、この関数の仕様で他のマイコンのプログラムも作ることがでるわけです。関数の中身だけマイコンに合わせてプログラムすれば、それを用いて作ったプログラムは変更する必要はなくなります。

01_LED09.c

```
 1  /************************************************************************/
 2  /*                                                                      */
 3  /*  FILE        :01_LED09.c                                             */
 4  /*  DESCRIPTION :LED0,2 の点滅 (関数化)                                 */
 5  /*  CPU TYPE    :H8/3694F                                               */
 6  /*                                                                      */
 7  /*  LED0 P85                                                            */
 8  /*  LED1 P86                                                            */
 9  /*  LED2 P87                                                            */
10  /*                                                                      */
11  /*  This file is generated by Renesas Project Generator (Ver.4.9).      */
12  /*                                                                      */
13  /************************************************************************/
14
15  #define PCR8 (*((volatile unsigned char *)0xFFEB))
16  #define PDR8 (*((volatile unsigned char *)0xFFDB))
17
18  #define LED0 0x01
19  #define LED1 0x02
20  #define LED2 0x04
21
22  #define LED_SHIFT 5
23  #define LED_MASK ((LED0 | LED1 | LED2) << LED_SHIFT)
24
25  #define WAIT_LOOP 0x3FFFFFL
26
27  void WaitLoop(void)
28  {
29    unsigned long count;
30
31    for(count=0; count<WAIT_LOOP; count++){
32      ;
33    }
34  }
35
36  /***************************************
37    LED 接続端子の初期化 (LED はすべて消灯)
38  ***************************************/
39  void LedInit(void)
40  {
41    PCR8 |= LED_MASK;
42    PDR8 |= LED_MASK;
43  }
44
45  /*******************************
46    1 を指定した LED を点灯
47  *******************************/
48  void LedOn(unsigned char led_data)
49  {
50    PDR8 &= (~((led_data << LED_SHIFT) & LED_MASK)); /* 0 をセット */
51  }
52
53  /*******************************
54    1 を指定した LED を消灯
55  *******************************/
56  void LedOff(unsigned char led_data)
57  {
58    PDR8 |= ((led_data << LED_SHIFT) & LED_MASK); /* 1 をセット */
59  }
60
61  /*******************************
62    指定した LED を点灯
63    それ以外は消灯
64  *******************************/
65  void LedSet(unsigned char led_data)
66  {
67    LedOn(led_data);
68    LedOff(~led_data);
69  }
70
71  void main(void)
72  {
73    unsigned char led_data=(LED0 | LED2);
74
75    LedInit();
76
77    while(1){
78      LedOn(led_data);
79      WaitLoop();
80      LedOff(led_data);
81      WaitLoop();
82    }
83  }
```

End Of List

実行結果

- すべての LED が消灯 (● ● ●)
- LED0,2 が点灯 (○ ● ○)
- 最初に戻って以下繰り返し

プログラム解説 (01_LED09.c)

```
18    #define LED0 0x01
19    #define LED1 0x02
20    #define LED2 0x04
```

P.113 の LED 関連の関数の仕様にあるように、LED0,LED1,LED2 という名前で LED を指定できるように、対応する値に名前をつけています。

以降では、LED0 と書けば 0x01 に置き換えられることになります。この名前を用いてレジスタの設定をする場合には、シフトをする必要があることに注意してください。

```
23    #define LED_MASK ((LED0 | LED1 | LED2) <<LED_SHIFT)
```

18 行目から 20 行目で定義した LED0,LED1,LED2 を用いて、LED_MASK を書き直しました。値自体は前のプログラムと同じ 0xE0 です。

```
39    void LedInit(void)
40    {
41      PCR8 |= LED_MASK;
42      PDR8 |= LED_MASK;
43    }
```

LED のポートを初期化する関数です。41 行目でポートコントロールレジスタ 8(PCR8) の設定をしています。LED が接続されたビットのみを出力に設定しています。

42 行目でポートデータレジスタ 8(PDR8) の設定をしています。LED をすべて消灯に設定しています。

```
48    void LedOn(unsigned char led_data)
49    {
50      PDR8 &= (~((led_data<<LED_SHIFT) & LED_MASK)); /* 0 をセット */
51    }
```

LED を点灯させる関数です。

50 行目は今までやってきたように、led_data の 1 のビットに対応したレジスタを 0 に設定します。その部分の LED は点灯します。

```
56    void LedOff(unsigned char led_data)
57    {
58      PDR8 |= ((led_data<<LED_SHIFT) & LED_MASK); /* 1 をセット */
59    }
```

LED を消灯させる関数です。

この関数の仕様が「1 で指定された LED を消灯する」であることに気をつけてください。1 で指定されたビットを 1 に設定するので、値の反転がありません。58 行目でこの操作をしています。

```
65    void LedSet(unsigned char led_data)
66    {
67      LedOn(led_data);
68      LedOff(~led_data);
69    }
```

3 ビット分の LED の点灯消灯をしている関数 LedSet は、点灯関数 LedOn と消灯関数 LedOff を用いて作ることができます。LedOff の引数が反転になっていることに注意してください。

```
73    unsigned char led_data=(LED0 | LED2);
```

18 行目から 20 行目で定義した LED0,LED1,LED2 を用いて、led_data に値を代入しています。単に数値を代入するよりも分かりやすいのではないでしょうか。

```
75    LedInit();
```

LED の初期化関数を呼び出しています。

```
78    LedOn(led_data);
79    WaitLoop();
80    LedOff(led_data);
81    WaitLoop();
```

LED の点滅を LedOn と LedOff を用いてこのように記述しました。LedSet と排他的論理和を用いて書くこともできますし、他にも書き方はいろいろあるでしょう。

課題 4.11.1　(提出) LED0,2 の点滅

プログラム 01_LED09.c を排他的論理和 (XOR) を使って書き直してください。

プロジェクト名：e01_LED09

4.12　LED 点灯の右シフト (ファイルの分割)

4.11 では LED 関連の関数を作りました。しかし、この関数を他のプログラムで利用するためには、コピーアンドペーストする必要がありそうです。

LED 関連の関数を一つのファイルに集めておけば、そのファイルを include するだけで LED の関数が使えて便利です。

ここでは、LED 関連の関数は led.c というファイルにすべて入れることにします。また、関数のプロトタイプ宣言などは led.h に入れておくことにしましょう。

今回はプロジェクト名と同じ C 言語ソースファイル 01_LED10.c 以外にも、led.h と led.c が必要です。以下に HEW のプロジェクトにファイルを追加する方法 (ソースファイルを入力する場合) を説明しておきます。

4.12.1 プロジェクトにソースファイルを追加 (入力)

ソースファイルを入力して、プロジェクトに追加する方法を説明します。

ここでは例として、led.h という名前のファイルを作成して、01_LED10 というプロジェクトに追加してみます。

メニューから「ファイル」「新規作成」を選びます (図 4.8)。

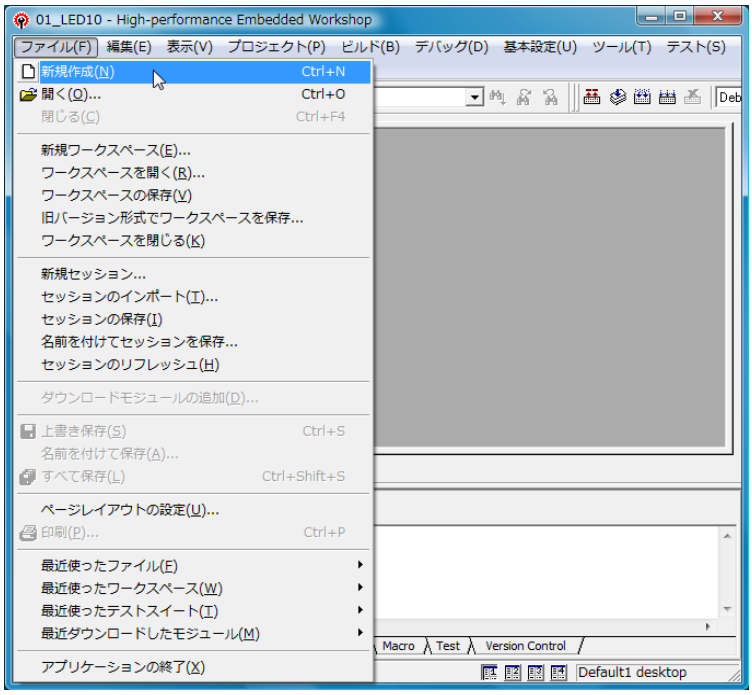

図 4.8　ファイルの新規作成

右上のエディタウィンドに「Document1」という名前の新しいファイルが表示されます (図 4.9)。

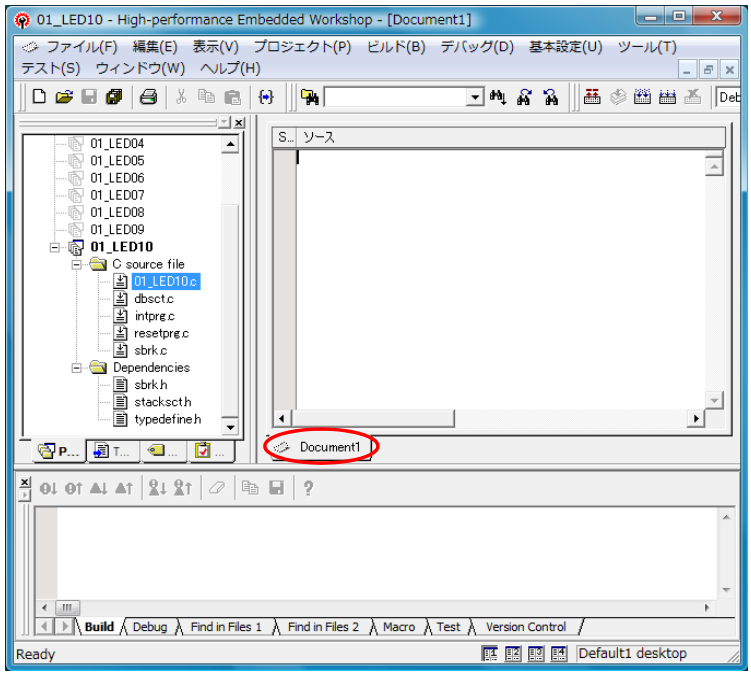

図 4.9　新規ファイル名

プログラムの内容を入力します (図 4.10)。ここではまず led.h を入力します。

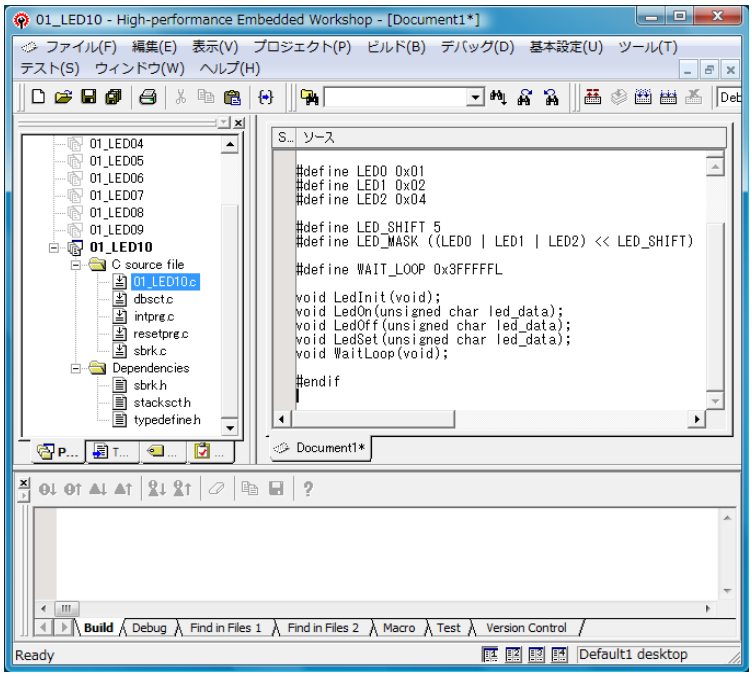

図 4.10　led.h を入力

プログラムは以下の通りです。これは、P.112 の 4.11 で示した 01_LED09.c の一部です。

led.h

```
 1  #ifndef _LED_H_
 2  #define _LED_H_
 3
 4  #define PCR8 (*((volatile unsigned char *)0xFFEB))
 5  #define PDR8 (*((volatile unsigned char *)0xFFDB))
 6
 7  #define LED0 0x01
 8  #define LED1 0x02
 9  #define LED2 0x04
10
11  #define LED_SHIFT 5
12  #define LED_MASK ((LED0 | LED1 | LED2) << LED_SHIFT)
13
14  #define WAIT_LOOP 0x3FFFFFL
15
16  void LedInit(void);
17  void LedOn(unsigned char led_data);
18  void LedOff(unsigned char led_data);
19  void LedSet(unsigned char led_data);
20  void WaitLoop(void);
21
22  #endif
```

End Of List

メニューから「ファイル」「名前を付けて保存」を選びます (図 4.11)。

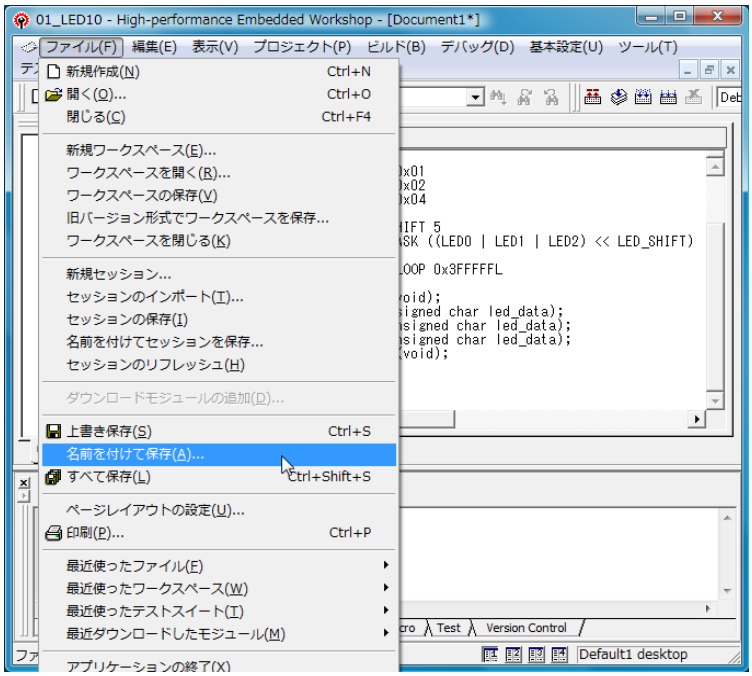

図 4.11　名前をつけて保存

「保存する場所」を指定して、「ファイル名」を入力して、 「保存」をクリックします (図 4.12)。

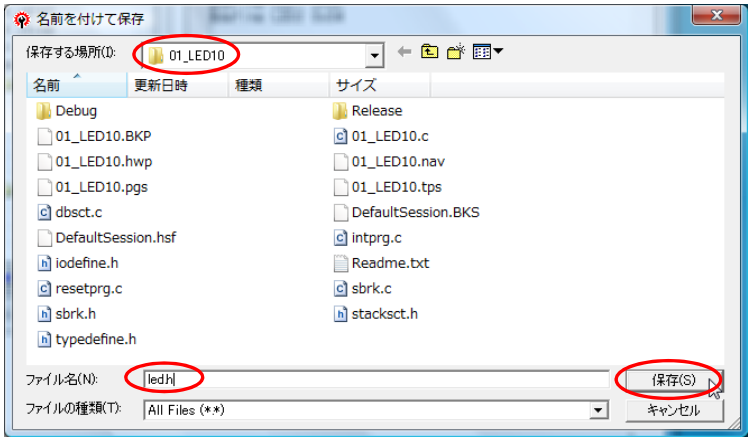

図 4.12　名前をつけて保存

ファイル名が「led.h」になっていることを確認しましょう (図 4.13)。

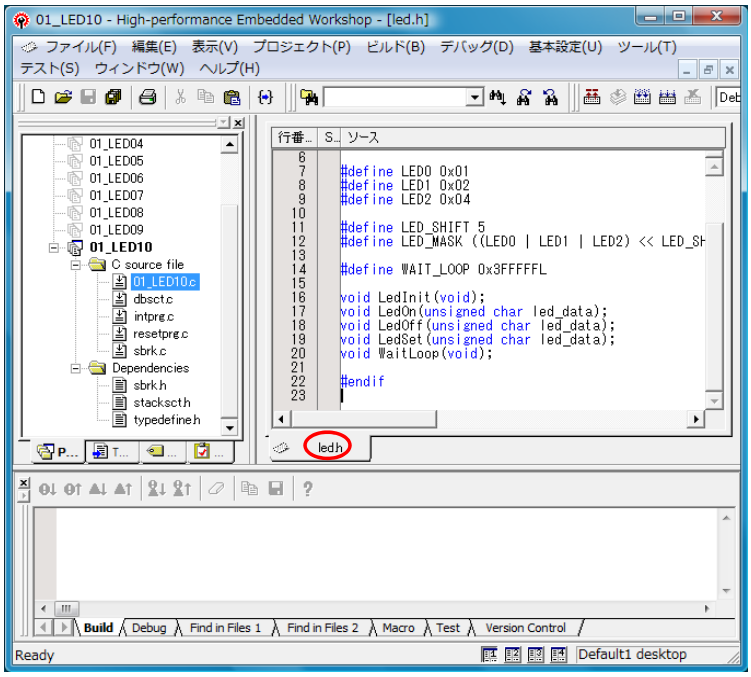

図 4.13　ファイル名が表示される

同様にして、led.c も入力し、保存します (図 4.14)。

プログラムは以下の通りです。これも、P.112 の 4.11 で示した 01_LED09.c の一部です。

led.c

```
 1  /*********************************************************************/
 2  /*  FILE        :led.c                                               */
 3  /*  DESCRIPTION :LED 用関数                                          */
 4  /*********************************************************************/
 5
 6  #include"led.h"
 7
 8  void WaitLoop(void)
 9  {
10    unsigned long count;
11
12    for(count=0; count<WAIT_LOOP; count++){
13      ;
14    }
15  }
16
17  /*************************************
18    LED 接続端子の初期化 (LED はすべて消灯)
19  *************************************/
20  void LedInit(void)
21  {
22    PCR8 |= LED_MASK;
23    PDR8 |= LED_MASK;
24  }
25
26  /*******************************
27    1 を指定した LED を点灯
28  *******************************/
29  void LedOn(unsigned char led_data)
30  {
31    PDR8 &= (~((led_data<<LED_SHIFT) & LED_MASK)); /* 0 をセット */
32  }
33
34  /*******************************
35    1 を指定した LED を消灯
36  *******************************/
37  void LedOff(unsigned char led_data)
38  {
39    PDR8 |= ((led_data<<LED_SHIFT) & LED_MASK); /* 1 をセット */
40  }
41
42  /*******************************
43    指定した LED を点灯
44    それ以外は消灯
45  *******************************/
46  void LedSet(unsigned char led_data)
47  {
48    LedOn(led_data);
49    LedOff(~led_data);
50  }
```

End Of List

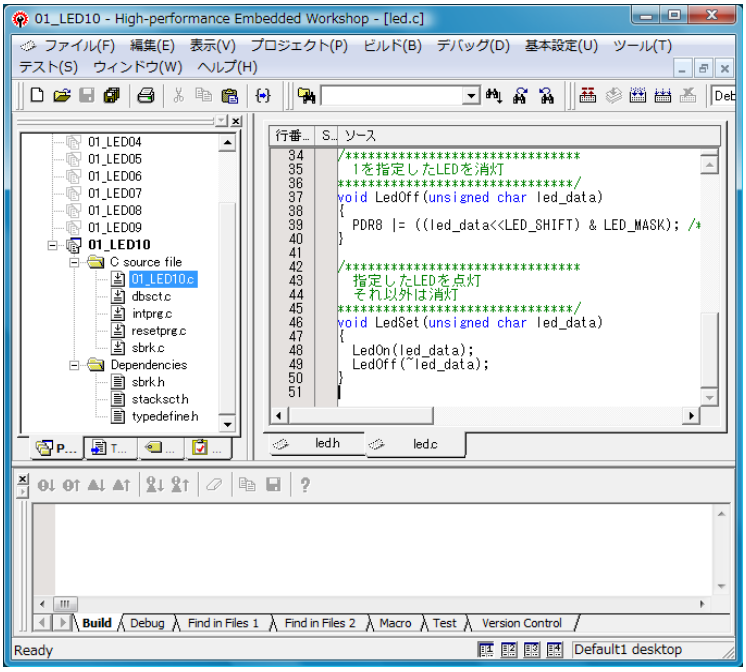

図 4.14　led.c を作成

プロジェクトの名前 (01_LED10) の上で右クリックし、「ファイルの追加」を選択します (図 4.15)。このとき、プロジェクトはアクティブになっていないといけません。

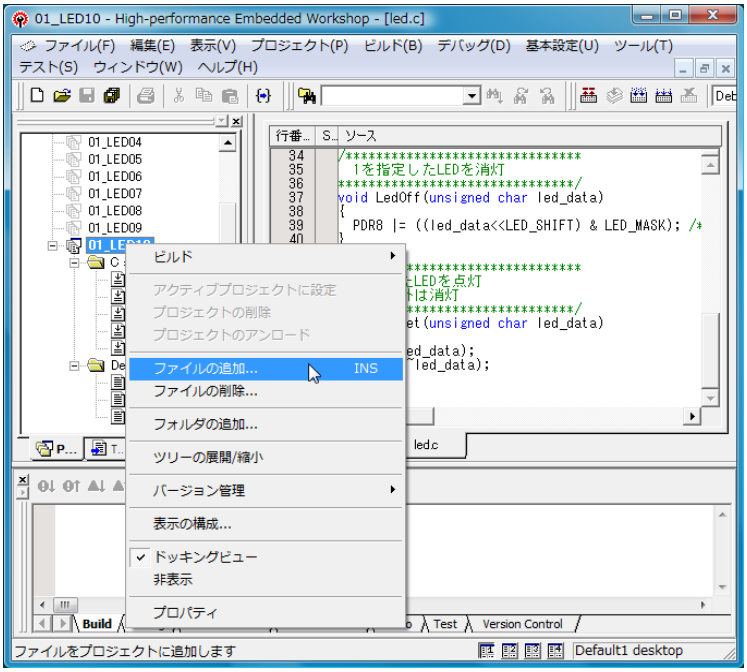

図 4.15　ファイルを追加

追加するファイル (led.c) を選択し、「追加」ボタンをクリックします (図 4.16)。

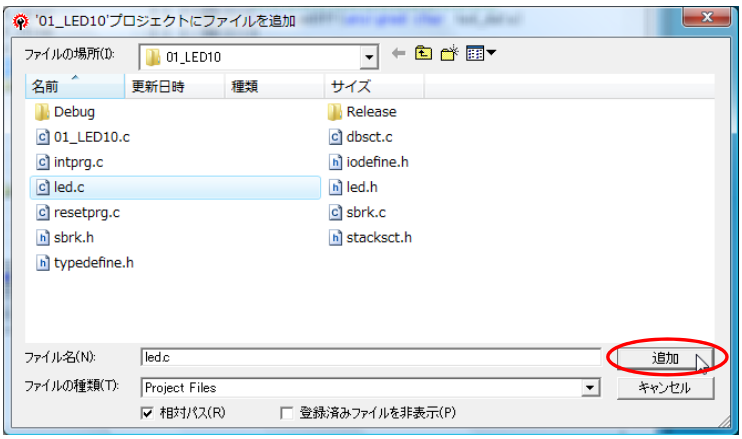

図 4.16　led.c を追加

ワークスペースウィンドにファイル (今の場合 led.c) が追加されていることを確認しましょう (図 4.17)。

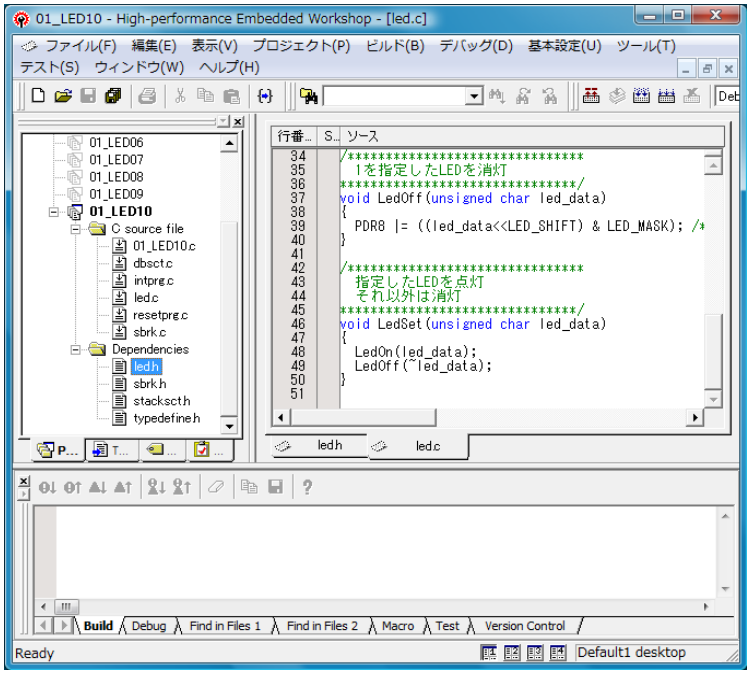

図 4.17　led.c が表示される

ヘッダファイルをインクルードしている場合には、 該当のヘッダファイルは自動的に追加されます (図 4.18)。

今の場合 led.c から led.h が呼び出されているので、ワークスペースウィンドウに自動的に led.h が追加

されます。

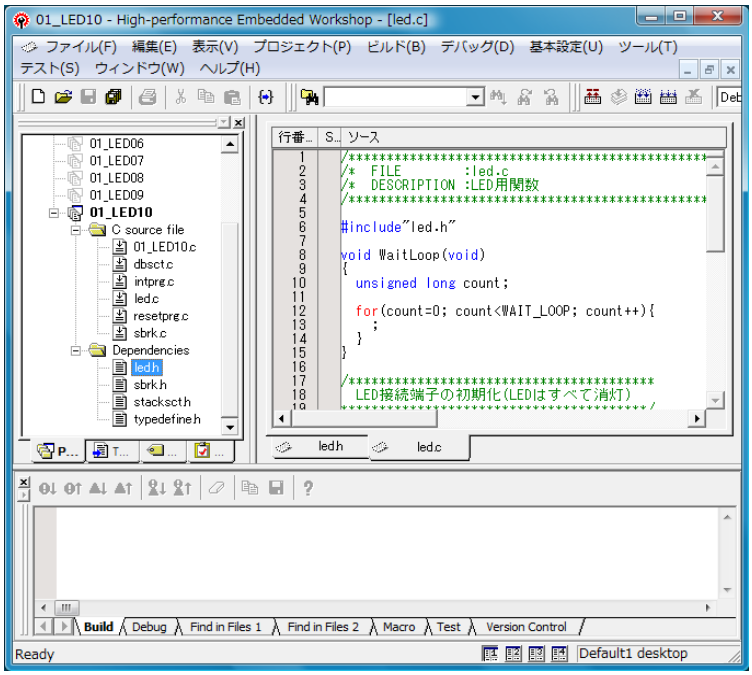

図 4.18　インクルードファイル led.h が表示される

4.12.2 サンプルプログラム

以下は、関数化してファイルを分割した LED のプログラムのサンプルです。4.12.1 で led.h と led.c を示しましたが、ここでは改めてすべてのソースコードを提示しておきます。

led.h

```
 1  #ifndef _LED_H_
 2  #define _LED_H_
 3
 4  #define PCR8 (*((volatile unsigned char *)0xFFEB))
 5  #define PDR8 (*((volatile unsigned char *)0xFFDB))
 6
 7  #define LED0 0x01
 8  #define LED1 0x02
 9  #define LED2 0x04
10
11  #define LED_SHIFT 5
12  #define LED_MASK ((LED0 | LED1 | LED2) << LED_SHIFT)
13
14  #define WAIT_LOOP 0x3FFFFFL
15
16  void LedInit(void);
17  void LedOn(unsigned char led_data);
18  void LedOff(unsigned char led_data);
19  void LedSet(unsigned char led_data);
20  void WaitLoop(void);
21
22  #endif
```

End Of List

led.c

```
 1  /************************************************************************/
 2  /*  FILE        :led.c                                                  */
 3  /*  DESCRIPTION :LED 用関数                                             */
 4  /************************************************************************/
 5
 6  #include"led.h"
 7
 8  void WaitLoop(void)
 9  {
10    unsigned long count;
11
12    for(count=0; count<WAIT_LOOP; count++){
13      ;
14    }
15  }
16
17  /*************************************
18    LED 接続端子の初期化 (LED はすべて消灯)
19  *************************************/
20  void LedInit(void)
21  {
22    PCR8 |= LED_MASK;
23    PDR8 |= LED_MASK;
24  }
25
26  /*******************************
27    1 を指定した LED を点灯
28  *******************************/
29  void LedOn(unsigned char led_data)
30  {
31    PDR8 &= (~((led_data<<LED_SHIFT) & LED_MASK)); /* 0 をセット */
32  }
33
34  /*******************************
35    1 を指定した LED を消灯
36  *******************************/
37  void LedOff(unsigned char led_data)
38  {
39    PDR8 |= ((led_data<<LED_SHIFT) & LED_MASK); /* 1 をセット */
40  }
41
42  /*******************************
43    指定した LED を点灯
44    それ以外は消灯
45  *******************************/
46  void LedSet(unsigned char led_data)
47  {
48    LedOn(led_data);
49    LedOff(~led_data);
50  }
```

End Of List

01_LED10.c

```
 1  /************************************************************************/
 2  /*                                                                      */
 3  /*  FILE        :01_LED10.c                                             */
 4  /*  DESCRIPTION :LED 点灯の右シフト (ファイルの分割)                    */
 5  /*  CPU TYPE    :H8/3694F                                               */
 6  /*                                                                      */
 7  /*  LED0 P85                                                            */
 8  /*  LED1 P86                                                            */
 9  /*  LED2 P87                                                            */
10  /*                                                                      */
11  /*  This file is generated by Renesas Project Generator (Ver.4.9).      */
12  /*                                                                      */
13  /************************************************************************/
14
15  #include"led.h"
16
17  void main(void)
18  {
19    unsigned char led_data=LED2;
20    int i;
21
22    LedInit();
23
24    while(1){
25      for(i=0; i<3; i++){
26        LedSet(led_data >> i);
27        WaitLoop();
28      }
29    }
30  }
```

End Of List

実行結果

LED 点灯が右シフトする。
動作としては、

- LED2 のみが点灯 (●●○)
- LED1 のみが点灯 (●○●)
- LED0 のみが点灯 (○●●)
- 最初に戻って以下繰り返し

となります。

プログラム解説 (led.h)

```
 1    #ifndef _LED_H_
 2    #define _LED_H_

22    #endif
```

拡張子 h が付いたファイルをヘッダファイルといいます。

ヘッダファイルには関数のプロトタイプ宣言や複数のファイルで利用する変数などを定義しておきます。

しかし、複数のファイルから同じヘッダファイルを include してしまうと、同じ宣言を複数回することになります。このような場合、定義が重複しているというエラーが出ます。

これを避けるために、プリプロセッサが処理する条件コンパイルを記述しておきます。

1 行目は、もし_LED_H_が定義されていなかったらという意味です。初めて読み込むときには定義されていないので、次の行以降を読み込みます。

2 行目は_LED_H_を定義しろという命令です。2 回目以降に読み込むと、_LED_H_が定義されているため、1 行目で条件に合わないので 22 行目まで無視されることになります。

このようにして、2 回目以降は読み込まれないようになるわけです。

なお、_LED_H_という名前はファイル名をもとにつけています。

```
16    void LedInit(void);
17    void LedOn(unsigned char led_data);
18    void LedOff(unsigned char led_data);
19    void LedSet(unsigned char led_data);
20    void WaitLoop(void);
```

関数のプロトタイプ宣言です。

このヘッダファイルを include することによって、これらの関数が使えるようになります。

プログラム解説 (led.c)

led.c は LED 用関数をそのまま移し変えました。

プログラム解説 (led10.c)

```
15    #include"led.h"
```

ファイル led.h を include しています。

LED 用の関数を利用するのに必要です。

```
19    unsigned long led_data=LED2;
```

今回のプログラムは右シフトなので、LED の初期状態は LED2 だけ点灯にします。

LED2 は led.h で定義されており、値 0x04 につけた名前です。

```
25    for(i=0; i<3; i++){
26      LedSet(led_data >> i);
27      WaitLoop();
28    }
```

関数 LedSet に渡す引数を、右シフトして渡しています。

最初は 0x04 を次に 0x02 を、その次に 0x01 を渡すことになります。その後 for ループを抜けます。