

## MFS08JS (MC9S08JS16 マイコンボード) ファーストステップガイド

IDE (CodeWarrior) の起動から LED 点滅まで

さて、インストール作業も終わったので早速プログラムを組んで書き込み、動かすというところまでの一通りの手順を習得してしまうことにしよう。

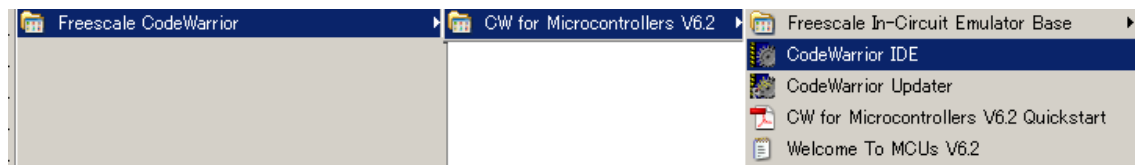
大まかな流れは

- 1) IDE (CodeWarrior) の起動
- 2) プロジェクトの作成
- 3) main.c へのプログラムの記述
- 4) make する
- 5) ボードをブートローダモードで起動 (USB ケーブルで接続)
- 6) BootLoaderGUI を起動して、書き込み

という具合になる。プログラムを組むという段階では細かいところを言い出すといろいろと面倒くさいところもあるのだけど、ここでは手順を習得するのが主な目的なので細かい話はスルーしてしまって、後からゆっくり調べていけば良い。

### 1) IDE の起動

IDE というのは (Integrated Development Environment) の頭文字をとったもので、日本語だと「統合開発環境」なんていう具合に呼ばれている (脱線コラム参照)



さて、スタートメニューから FreeScale CodeWarrior CW for Microcontrollers V6.2 (数字はバージョンによって変わる) と降りていって CodeWarrior IDE を選択しよう。

脱線コラム：統合開発環境

何が「統合」なんだかというと、エディタやコンパイラ、アセンブラ、デバッガなどがひとつの画面から扱えるようになってきているということ。

長年この手の開発ソフトというのは文字が主体のコマンドラインインターフェース・・・Unix系ならシェルとか、Windows などでは「DOS プロンプト」なんていうものの上で、まずエディタを起動して、ソースコードを作り、次にコンパイラやアセンブラを起動してオブジェクトファイルを作り、リンカなどを起動して複数のオブジェクトやライブラリなどをリンクし、最後に書き込み可能なファイル形式へのコンバータを起動し、出来上がったものは書き込みツールを起動し、うまく動かなければ今度はデバッガを起動して・・・という具合でそれぞれバラバラになっていた。たとえばコンパイルひとつでも

```
$/usr/local/arm-tools/bin/arm-elf-gcc -mcpu=cortex-m3 -mthumb -c $< -o gcc_sample.o
```

なんていう具合。

まあ、分かってくるとこの手のやり方というのが分かりやすいっていう部分もあるのだけど、やっぱり GUIの方がわかりやすいよねっていうことで、一連の操作を全部ひとつのGUI上でできるようにしたのが IDE。だいたい今はCコンパイラ(C++っていうのもあるけども)が使われるので main()関数に飛び込んでくるまでの最低限の初期化は自動生成してくれて、ユーザは main()の先だけ考えれば良いようになっているのが普通。

更に最近ではGUI上で設定するだけで何かと面倒なデバイスの初期化部分やら I/O 関係を操作するためのライブラリも自動生成するようになってきている。

という具合で、今のマイコン開発環境では IDE を使うか、あるいは Unix 環境 (Linux を使ったり、Windows 上でエミュレーションする Cygwin なんていうのが多いみたい) でコマンドラインで頑張るかという二択だけど、だいたいのメーカーさんは前者をご推奨で、後者を使った場合は何か変なことがあってもそっちで勝手に頑張って調べてね・・・と、ひんやりやんわりと受け流されることうけあいである。

メーカーさんご推奨 IDE の場合メーカーによって無償提供のところと、有償のところがある。もっとも有償とはいってもお試し用( )の無償版が提供されている。今回使う CodeWarrior は Special Edition と呼ばれる無償版。ちなみに購入すると一番安い BASIC 品でも\$395、一番高いのになると\$4995 (いずれも 2010 年 7 月時点)ということなので、ご家族のある方は財務省の判断を仰がないと厳しいかもしれない。

：物によって機能制限やプログラムサイズの制限、インストールした日からの使用日限などなどを加えている。CodeWarrior の HCS08 版ではオブジェクトが 32K バイトまでという制

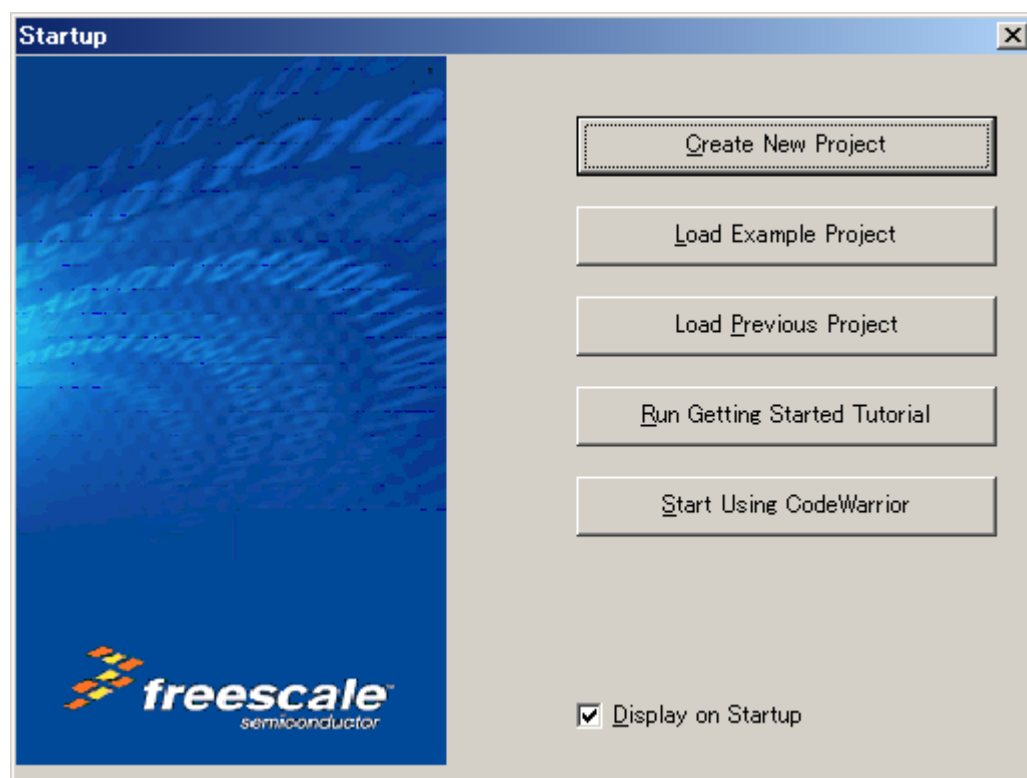
## 2) プロジェクトの作成

こんな感じのウィンドウが表示されるはず。今回は新規プロジェクトの作成なので一番上の「Create New Project」をえらぶ。

ちなみにその下にあるのは

- ・サンプルプロジェクトの読み込み
- ・前回扱ったプロジェクトの読み込み
- ・チュートリアル（Code Warrior の使い方練習：英語）
- ・TIPS（使い方ヒント：これも英語）の表示

という具合。

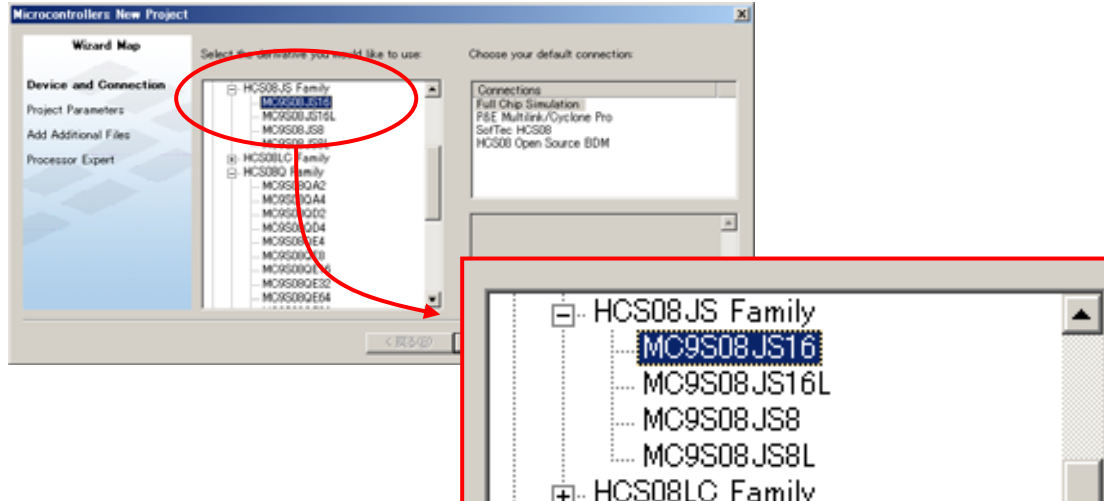
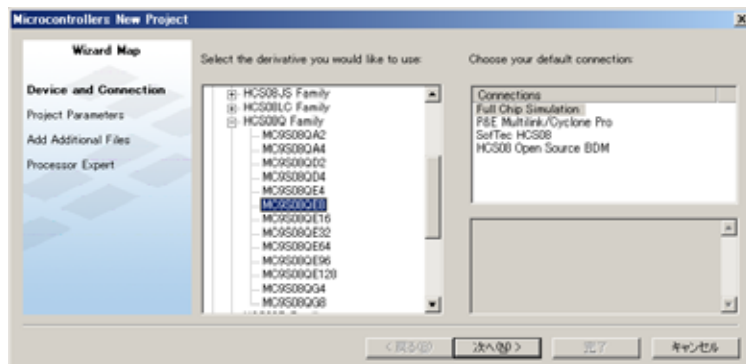


## 2-1) Device and Connection ステップ

すると,こんな按配で,デバイス(左側)とデバugg(右側)の選択画面になる.左の「Wizard Map」の一番上が太字になって最初のステップだぞと分かるようになっている.

デバuggは今回は使わない(そもそもデバuggが無い)ので,何でも良い.ちなみに,この一番下にある「HCS08 Open Source BDM」というのはフリーのデバuggとして回路図やプログラムが公開されているオープンソースなデバugg. MC9S08JS もサポート対象になっているので,興味のある方は調べてみると良い.

デバイスの方は当然何でも良い・・・というわけにはいかない.今回使用する MC9S08JS16 を選ぶ. HCS08 Familyの中から MC9S08JS16 を選んで「次へ」をクリックである.



(つづやき)ファミリ名と製品型名の頭の文字が違って何かとややこしいというのはいろいろ複雑な社内事情と歴史的背景ってところもあるのだろうけど,やっぱりややこしい.

## 2-2) Project Parameters ステップ

Wizard Map でいうところの第二段階の「Project Parameters」になってこんな画面がでてきたはず。

左側はプログラムの記述に使用するプログラミング言語の選択，右がプロジェクトの名称 (Project name) とプロジェクトを置くフォルダ/ディレクトリ (Location)の指定である。

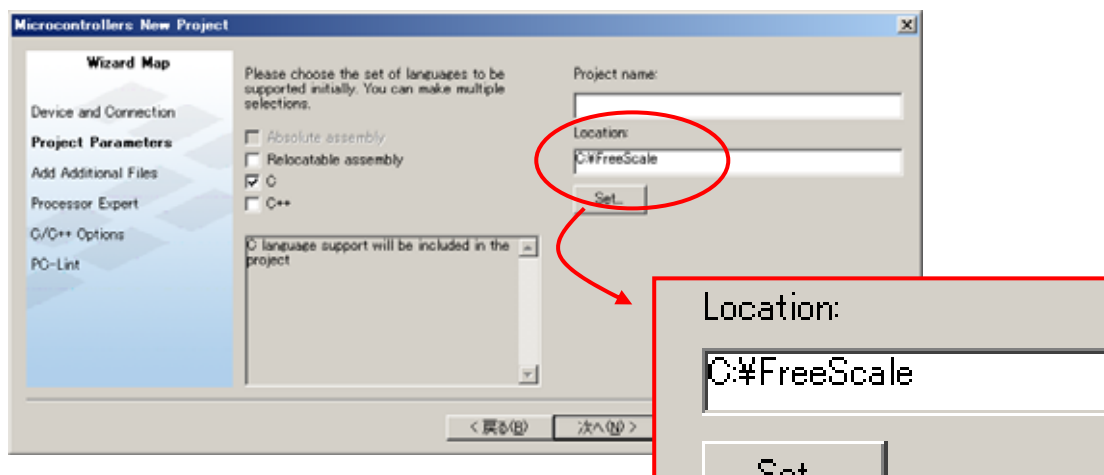
言語の方はデフォルトの C 言語にしておけば良い。一見「を・・・C++も使える?」と思うところだけど，クリックするとわかるとおりで，これはそれなりのお値段のする Professional Edition か Standard Edition が必要なので今回はパス。

Project name と Location の指定はそれぞれ

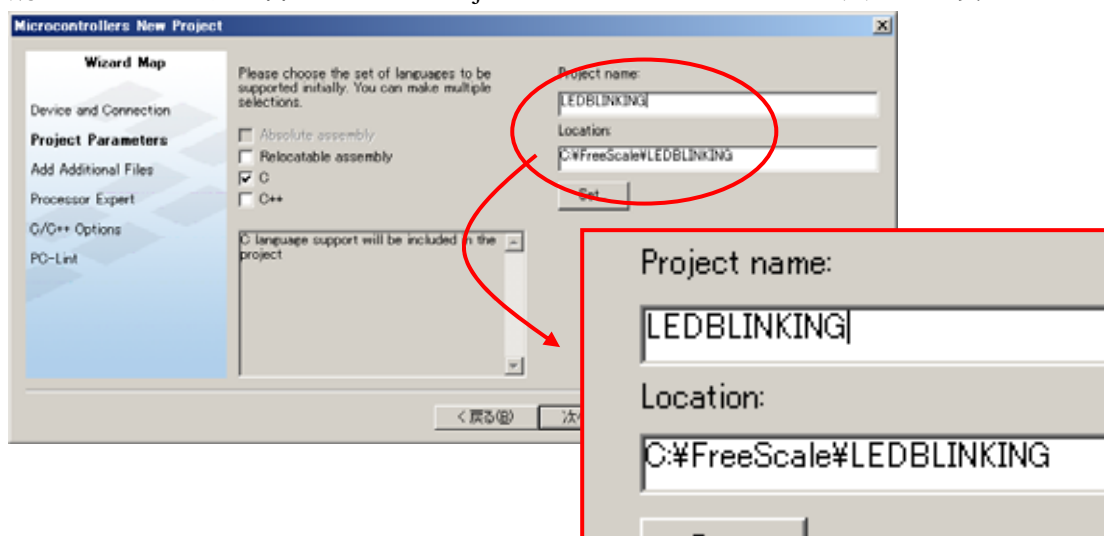
- LEDBLINKING
- C:\FreeScale\LEDBLINKING

としてみた。フォルダ名の最後のところは自動的にプロジェクト名と同じものになるので，入力は不要である。

まず，フォルダ名を設定。これは Location に C:\FreeScale と入れておく



続いてプロジェクト名。こちらは Projectname に LEDBLINKING と入れれば良い



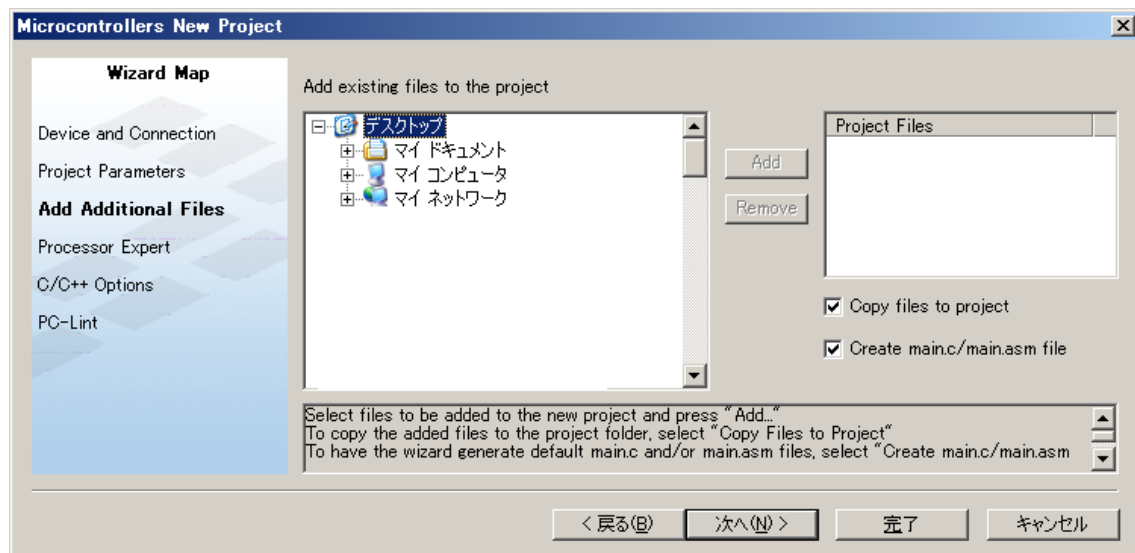
これでプロジェクト作成のための最低限の手続きは終わり．ここから先はおまけみたいなもので，後は全部自力でやるぜというタフな方は「完了」してしまえば良いけど，この「おまけ」がなかなか便利だったりするし，せっかく用意してくれているのだから使ってみる．

「次へ」をクリックである．

## 2-3) Add Additional Files

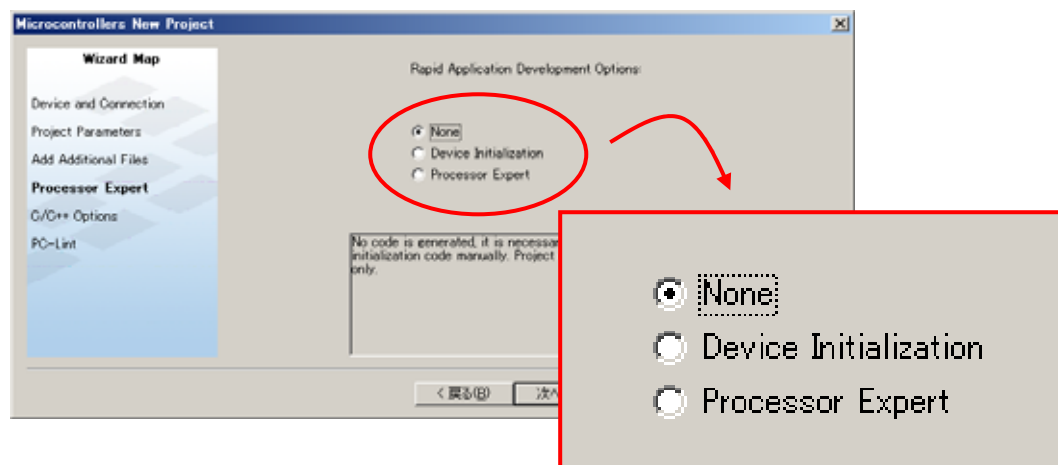
と，ファイル追加の画面になる．自動生成されたファイルだけでなく自前で用意したライブラリやヘッダファイルなどをプロジェクトに追加しておきたいときはここで指定すれば良い．右下には，指定したファイルをプロジェクトのフォルダ内にコピーするのか，main.c（C言語を使う場合）や main.asm（アセンブリ言語を使う場合）

今回は特に追加するファイルは無いので，単に「次へ」をクリックで良い．



## 2-4) Processor Expert

続いて Processor Expert というプログラム作成のためのお助けツール ( Rapid Application Development Options ) を使うか否かの設定画面になる。残念ながら MC9S08JS は Processor Expert がまだ対応していないようなので、ここは「None」を選択して次に行くしかない。



### 脱線コラム：Processor Expert

マイコン系のプログラム開発が Windows アプリケーションなどと比べて面倒な点はいくつかあるが、その中でも、CPU のリセット直後からの動作や CPU や内蔵 I/O 類の初期設定、割り込みのエントリ部分など細かい部分まですべてプログラマが面倒を見なくてはならないということは何かと面倒や間違いをしやすい要注意ポイントである。

このあたりを IDE の方で支援してやろうというのが Processor Expert で、

- ・ None は Processor Expert による
- ・ Device Initialization ならデバイスの初期化関係を GUI で行う
- ・ Processor Expert ならその I/O 関係なども行う

という感じ。今のところ MC9S08JS はサポートしていないけれどもいずれ対応することに期待しよう。

## 2-5) C/C++ Options

C/C++言語のための設定である．とりあえずデフォルトのままそのまま進んでしまっても良い．  
というだけだと気持ち悪いと思うので一応簡単に解説だけ．

一番上は ANSI に準拠するのか否かの設定．ANSI startup code を選ぶと初期化していない変数のゼロクリアやら初期値を与えた変数の初期化などを行うコードが生成される．まあ，だいたいはこちらを選んでおいて良いのだけれども，組み込み用途などではこうした初期化をやってほしくない．つまりリセットがかかる寸前のデータを保持しておいて欲しいということもある．そんなときは minimal startup code を選ぶと必要最小限の初期化だけやって main() にくるようになる．今回は ANSI startup code を選んでおけば良い．

二番目はメモリモデルの指定．HCS08 の場合，0000H～00FFH 番地までの 256 バイトの範囲は「ゼロページ」と呼ばれる領域になっており，この範囲を効率良く（少ない命令サイズで）アクセスする命令が用意されている．MC9S08JS ではゼロページのうち 00H～7FH は内部 I/O 用レジスタとして利用されていて，80H～FFH は RAM 領域になっている．

Tiny モデルではこのゼロページの RAM 領域に変数を割り付ける．一方，Small の場合にはゼロページ外（100H 以降）の領域に割り付ける．変数や配列などをあまり使わないときは Tiny にしておくコンパクトになるという理屈．

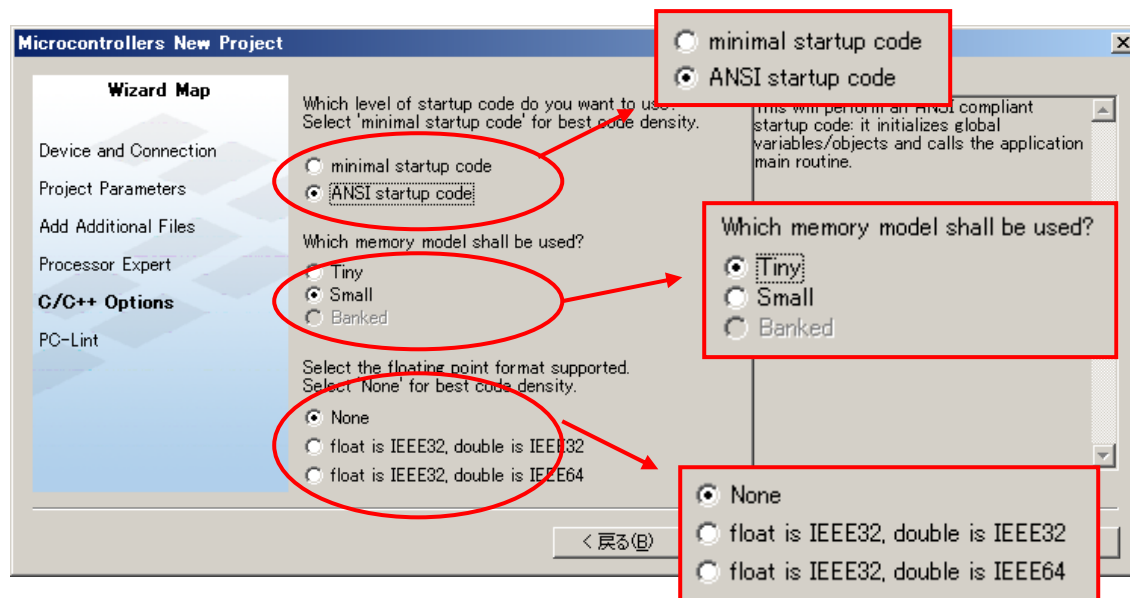
三番目は浮動小数点演算演算をサポートするのか，サポートするときに double 型のビット長（データフォーマット）をどうするのかの指定．浮動小数点演算をやるつもりがないならメモリ容量節約のためにも None にしておくのが良いだろう．

下の二つは C 言語の double 型を使ったときのデータ形式選択．C 言語で float 型を指定したときは 32 ビット長だけど，double にしたときには float と同じ 32 ビット長になるのか，64 ビット長になるのかをここで指定する．

浮動小数点の形式として広く利用されているのは 32 ビット長（符号 1 ビット + 指数 8 ビット + 仮数部 23 ビット）のフォーマットだけれども，これでは精度が足りない，あるいは扱う数値の範囲が表現しきれないという場合に対応して 64 ビット長のフォーマット（符号 1 ビット + 指数 11 ビット + 仮数部 52 ビット）が用意されていて，前者を IEEE32，後者を IEEE64 と呼んでいる．I

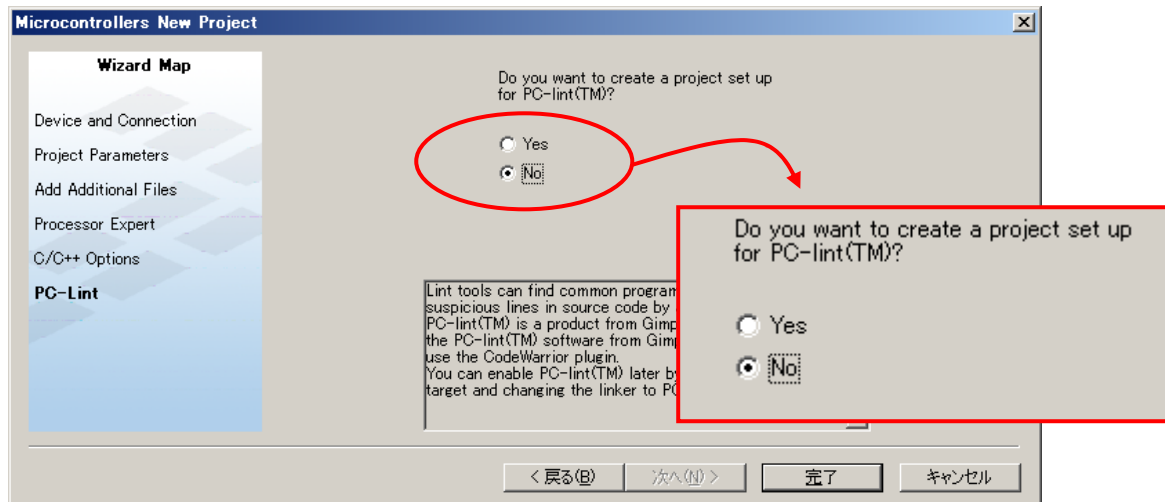
そもそもこのクラスのマイコンで浮動小数点演算というのは結構重い処理でもあるし，IEEE64 となると更に重い．





## 2-6) PC-lint

プロジェクトを PC-lint に対応させるか否かの設定 . PC-lint は別プロダクトになっていて , 別途インストールしなくてはならないので , ここでは使わない . 「No」のままにして「完了」させて良い .



## 脱線コラム : lint

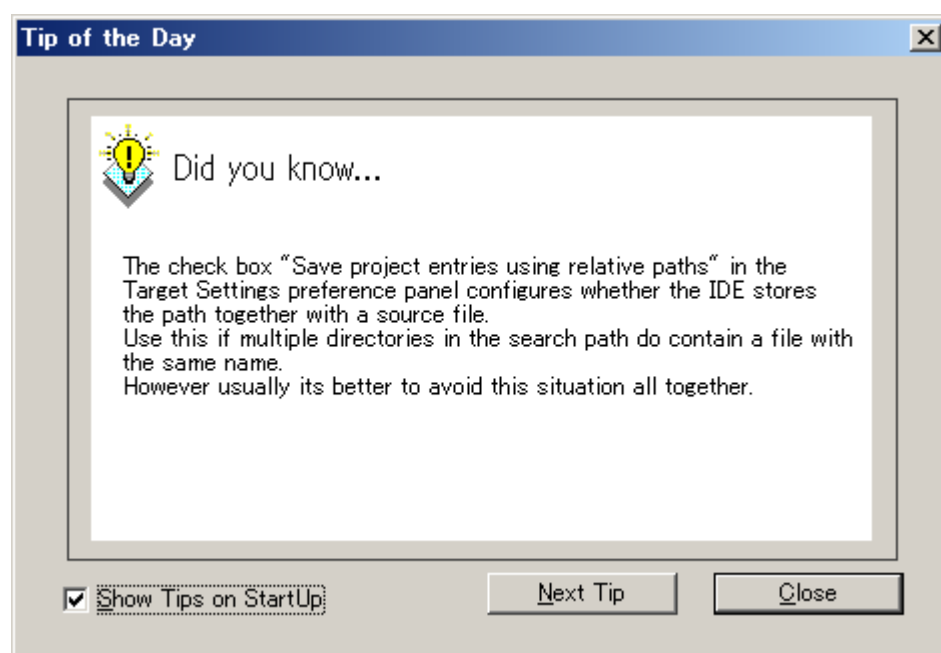
lint というのは C 言語の文法チェッカとして昔から unix 上で使われてきたツール . コンパイラもそれなりのチェックはするけれども , lint は更に細かいチェックまで行うので , つまらないバグが出にくくなる .

もともと C コンパイラというのは高水準アセンブラのようなもので , あまり禁止事項を設けず , プログラマの責任で何でもできるのが魅力 . とはいえ , C 言語というのはたとえば `if(i==3)` ( i と 3 が等しいなら成立 ) と書くべきところを `if (i=3)` ( i に 3 を代入して結果が 0 以外なら成立 ) にしても OK というくらい結構危ういもの .

やっぱりチェックくらいしてくれたほうが . . . ということで , 文法チェッカとしての lint が作られたということらしい .

もっとも , 最近はコンパイラのチェックも結構厳しくなっているんで警告をちゃんと確認しておけばそれほど痛い目にはあわないだろう .

ということで、終了すると、こういう TIPS が表示される。別に読まなくてはならないというものでもないの、そのまま Close してしまっても良い。



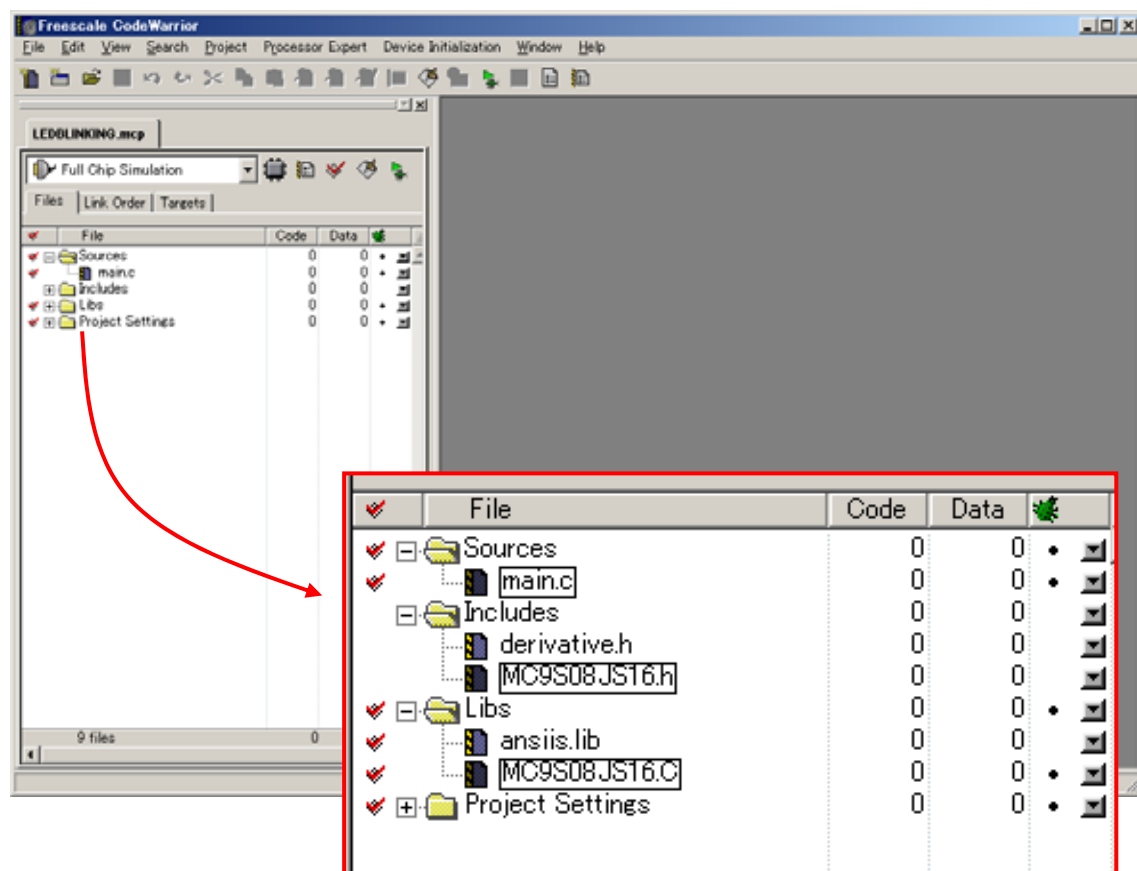
### 3) main.c へのプログラムの記述

さて、プロジェクトが作成された後の画面はこんな感じ。ちょっと拡大しているけども、Sources の下に main.c が、そして Includes や Libs の下にヘッダファイル（拡張子が.h のファイル）やライブラリ（.lib や.c のファイル）が用意されている。

この中で MC9S08JS16 の内部レジスタのビット配置や名称の定義をしているのが MC9S08JS16.h と MC9S08JS16.C。

なにぶん今のマイコンは内部レジスタの数も多く、無味乾燥なアドレスを指定してアクセスするのはとっても面倒・・・ということで、データシートに書いてあるような名称でアクセスできるようにしてくれているというわけ。

ちなみに、この二つのファイルはプロジェクトのディレクトリには無い。実際の場所はファイルを右クリックして現れるメニューから「Open in Windows Explorer」を選択すれば良い。



さて、それではプログラムを記述していくことにしよう。まずは基板の上にある LED を点滅させてみる。とりあえず、次のようなプログラムを書いておく。分かりやすいように日本語コメントを入れているけども CodeWarrior のエディタは日本語対応していないので、文字化けする。

<pre>#include &lt;hidef.h&gt; #include "derivative.h" void Init_Port() { // PTA は全部ユーザ開放（とりあえず全ピンプルアップ） PTAD = 0x00; PTADD = 0x00; PTAPE = 0xff; PTASE = 0x00; PTADS = 0x00;  // PTB // PB[0] : --- // PB[1] : ResetSwitch // PB[2] : BKGD // PB[3] : BLMS スイッチ/LED // PB[4] : XTAL(12MHz) // PB[5]=&gt;XTAL(12MHz) // PB[6]=&gt; N/A // PB[7]=&gt; N/A PTBD = 0x08; PTBDD = 0x08; PTBPE = 0x03; PTBSE = 0x00; PTBDS = 0x00; }</pre>	<pre>void delay() { unsigned int i,j,k; for(k=0;k&lt;20;k++) { for (i=0; i&lt;100; i++) { for (j=0; j&lt;100; j++) ; } } }  void main(void) { unsigned char c; SOPT1 = 0x13; // WDT 停止 . PB1 をリセット入力 // Init_Clock() Init_Port(); // PTA,PTB の初期化 c = 0; for (;;) { if (c &amp; 1) { PTBD  = 0x08; } else { PTBD &amp;= ~0x08; } c++; delay(); } }</pre>
--	--

こまごましたことはまた後ほど．とりあえずここでは Init\_Port で I/O ポートを初期化して，

1)PTBD レジスタにデータを書き込んで LED を点灯 / 消灯して

2)delay()で時間稼ぎ

3)1)に戻る

というループを繰り返して LED を点滅させている．

ところで CPU はどのくらいの速度で動くのだろうかということを疑問に思うのは至極当然．基板には外付けの水晶（基板では 12MHz）が付いているけれども，この状態ではまだリセット後の初期状態のままなので，内部クロック（32KHz 程度）を 512 倍（1024 倍 ÷ 2）した約 16MHz のクロックで動いている．

MC9S08JS は 48MHz で動作できるので，この状態の約 3 倍まで動作を速くすることができる．以下のような Init\_Clock()関数を追加して，main()でコメントになっている Init\_Clock()の呼び出しを有効にすれば良い．

```
//
// リセット後の内部クロックモード（約 16MHz）から
// 12MHz の外付け水晶を使った PLL モード（48MHz）への切り替
// え
//
void Init_Clock()
{
    MCGC2 = MCGC2_HGO_MASK      // High ゲインモード
        | MCGC2_EREFS_MASK      // 外部水晶を使う
        | MCGC2_RANGE_MASK      // 1-16MHz の間にある
        | MCGC2_ERCLKEN_MASK;   // 外部リファレンスを使う
    while (MCGSC_OSCINIT == 0)
        ;

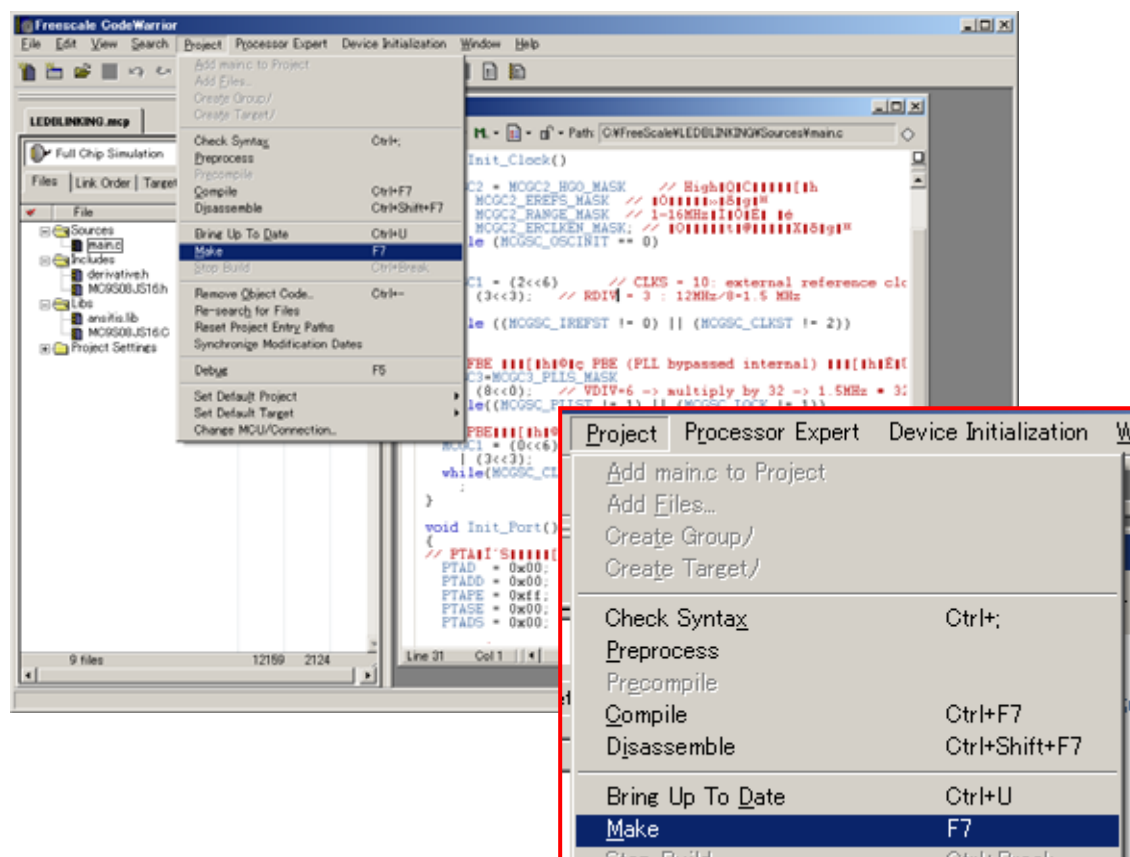
    MCGC1 = (2<<6) // external reference clock.
        | (3<<3);    // RDIV = 3 : 12MHz/8=1.5 MHz
    while ((MCGSC_IREFST != 0) || (MCGSC_CLKST != 2))
        ;
}
```

```
// FBE PBE (PLL bypassed internal) モード
// に移行
MCGC3=MCGC3_PLLS_MASK
        | (8<<0); // multiply by 32 ->
1.5MHz*32=48MHz
while((MCGSC_PLLST != 1) || (MCGSC_LOCK !=
1))
    ;
// PBE モードから PEE
// (PLL enabled external mode)
// モードに移行
MCGC1 = (0<<6) //PLL or FLL Clock
        | (3<<3); // 12MHz/8=1.5 MHz
while(MCGSC_CLKST !=3)
    ;
}
```

## 4) make する

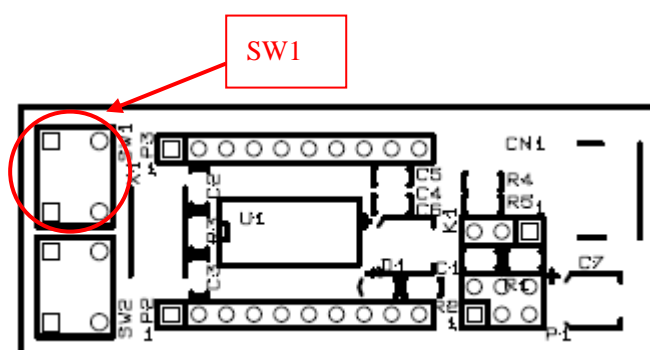
プログラムの入力が終わったら make して書き込み可能なファイルを作成する。別段難しいことはなくて、単に Project メニューから Make を選択すれば良い。

エラーや Warning などが出てしまったらソースコードを良く確認しよう。

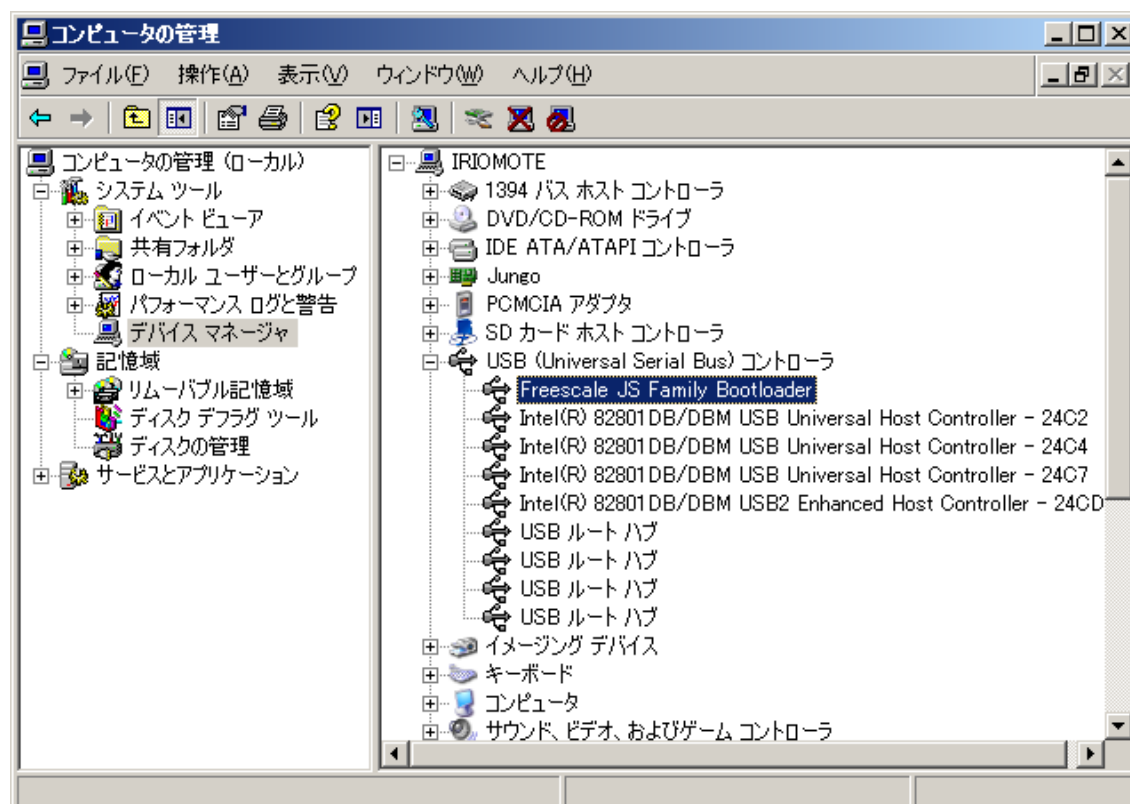


## 5) ボードをブートローダモードで起動 (USB ケーブルで接続)

さて、無事に終わったら、書き込みの準備をしよう。SW1 (下図参照) を押しながら USB ケーブルを接続すると、PC に認識される筈。



デバイスマネージャで眺めるとこんな感じで、USB ( Universal Serial Bus)コントローラのところに「Freescape JS Family Bootloader」が居るはず。





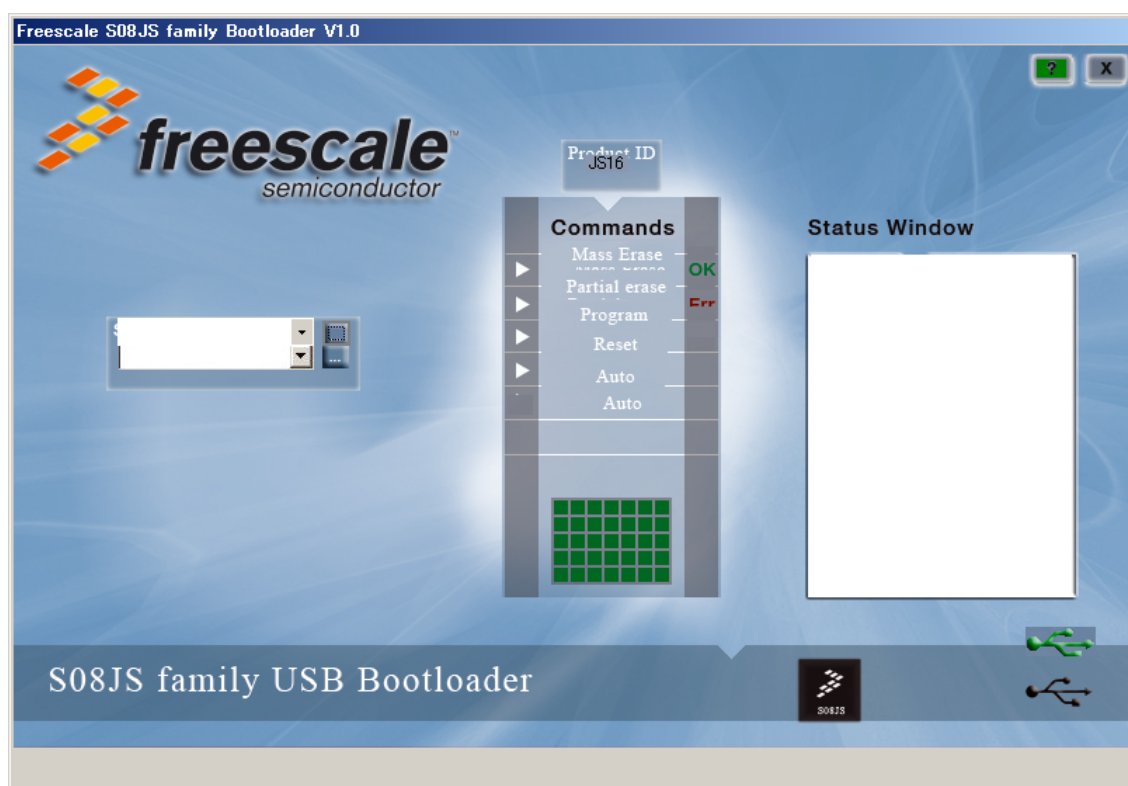
6) BootLoaderGUI を起動して、書き込み

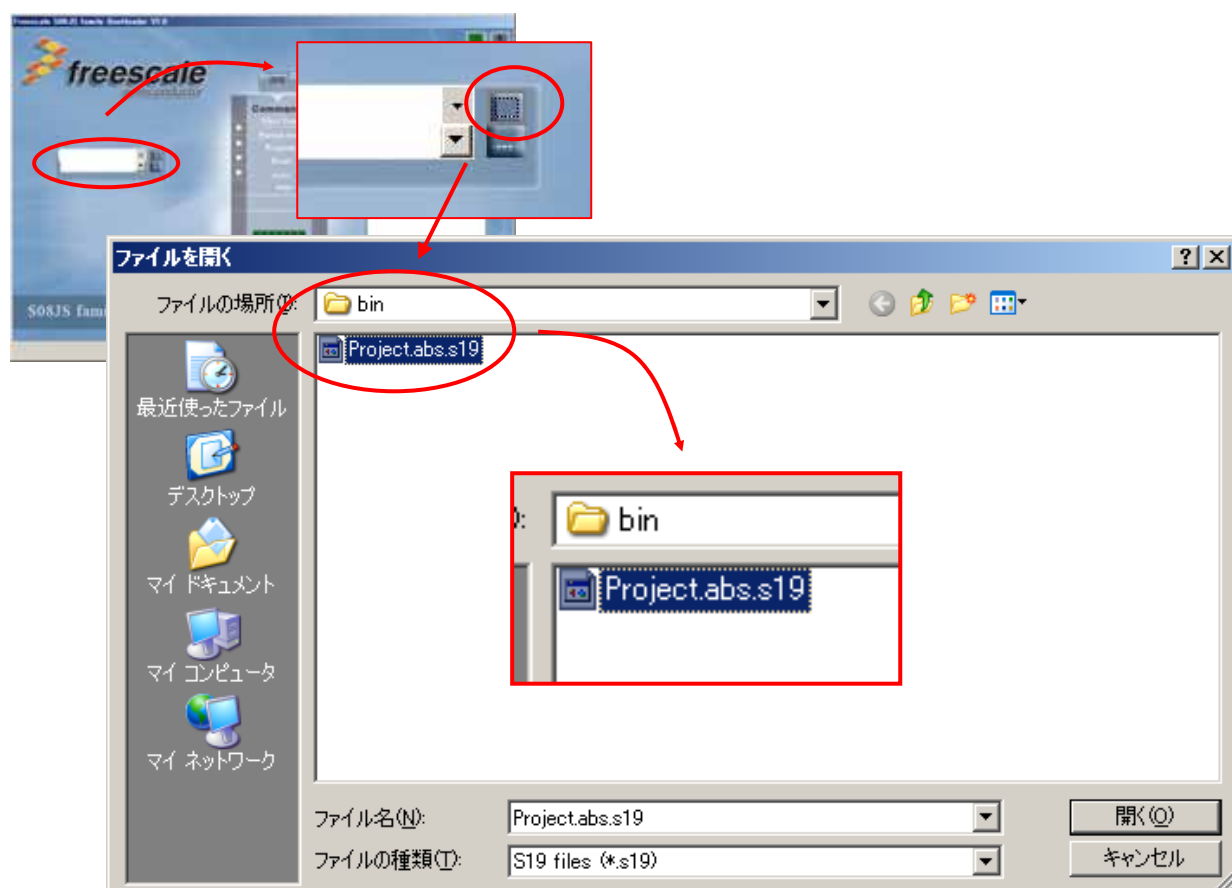
次に書き込みツールを起動して書き込みを行う。本当は CodeWarrior からそのまま書き込み実行できれば良いのだけど、まだそうならないようで、独立したツールの「Bootloader JS Family GUI」を使う。

スタートメニューから Freescale JS Family BootLoader Bootloader JS Family GUI を選択する。



と、こんな感じの画面になる。なんだか若干画面が崩れているような部分があるけれども、そういうものらしいので気にしないで良い。





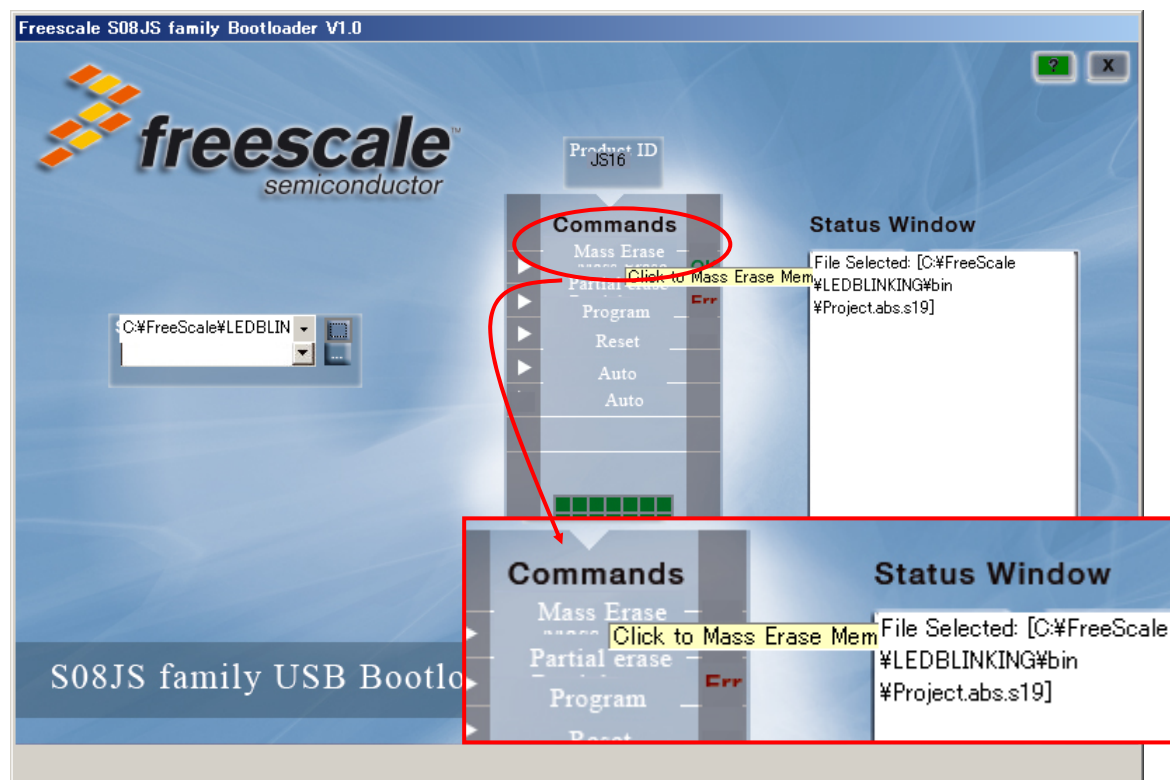
左上のリストボックスの横にあるボタンをクリックする。ファイル選択画面になるので、プロジェクトの下に bin ディレクトリの中にある Project.abs.s19 ファイルを選択する。

書き込みは、

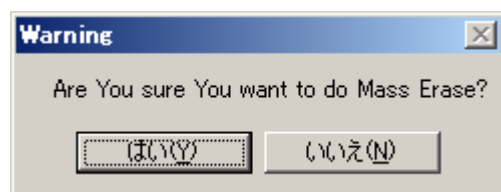
- (1)Mass Erase (チップ全体の消去)
  - (2)Program (書き込み)
  - (3)Reset (リセットして書き込んだプログラムを実行)
- というステップで行います。

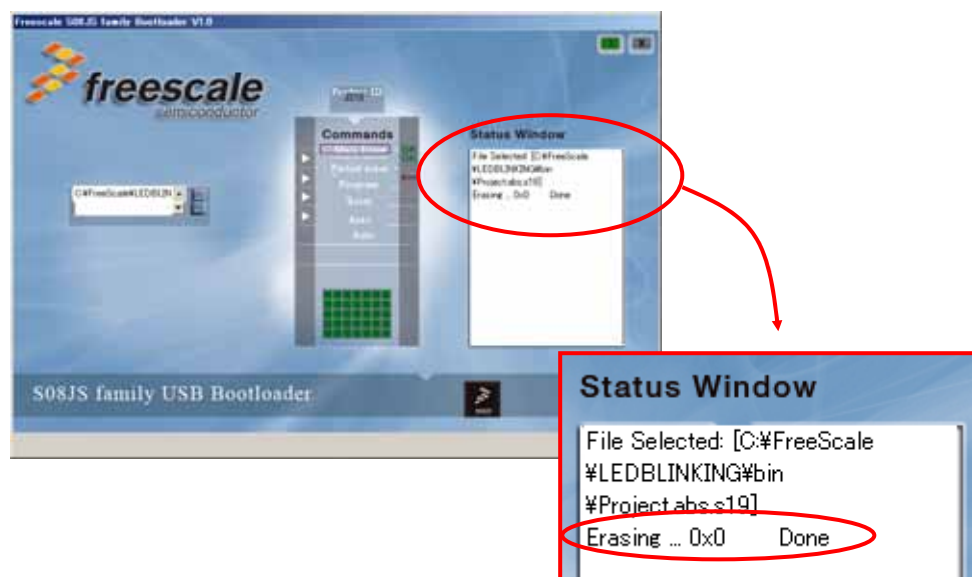
(1)Mass Erase

中央部分のメニューで「Mass Erase」をクリックする



次のようなダイアログが出る。「本当に全部消しちゃっていいの?」ということなので「はい」で良い。

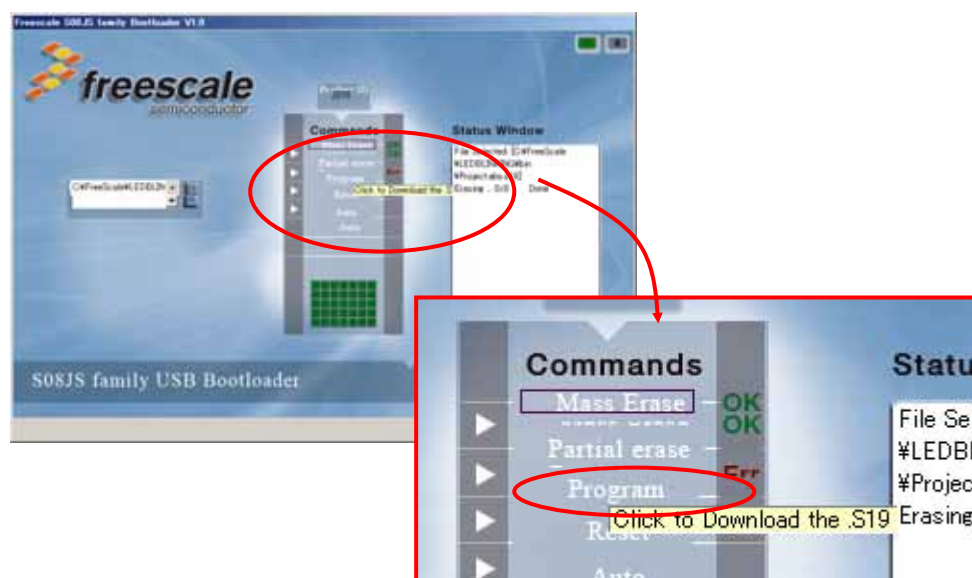




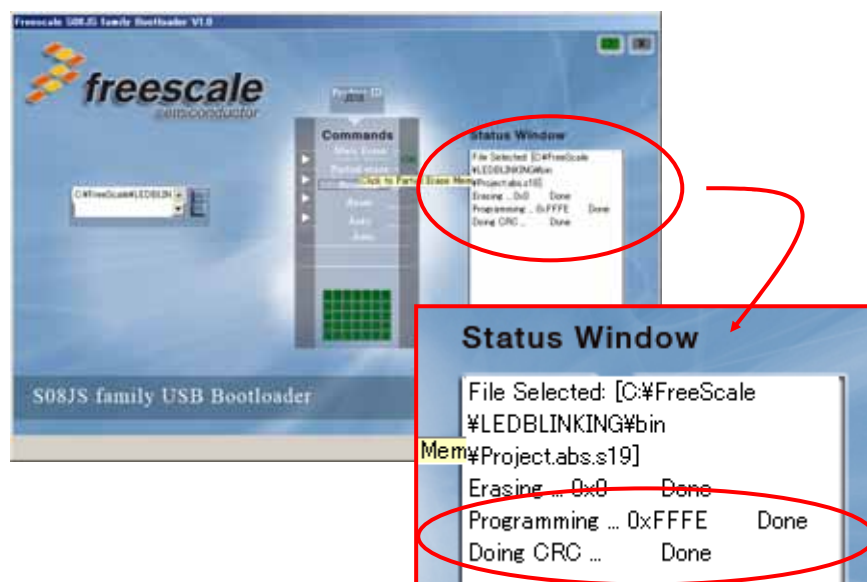
消去できれば、図のように「Erasing...0x0 Done」と消去完了メッセージが出る。

## (2)Program

次に「Program」をクリック。



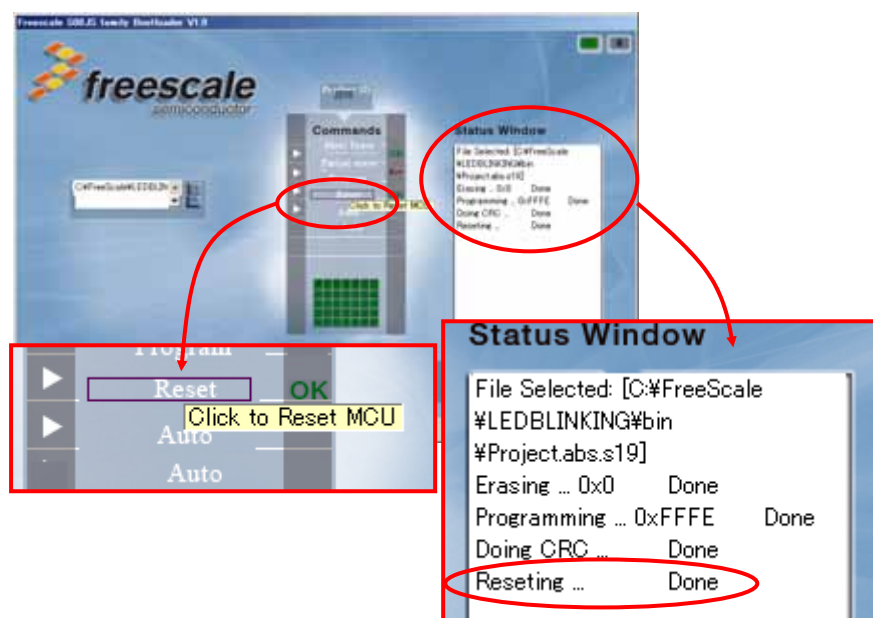
プログラミングが進み、完了すると図のようなメッセージが出る。最後の Doing CRC というのは正常に書き込めたかどうかのエラーチェックコードをチェックして正常に書き込めたかを確認するステップ。



(3)Reset (リセットして書き込んだプログラムを実行)

最後に Reset をクリックして ,CPU をリセットして今書き込んだプログラムを実行する .  
ボード上の LED が点滅するはずである .

Init\_Clock()の有無によって点滅速度が変わる ( CPU の動作速度が変わる ) ことも確認しておこう .



## Appendix

### A : I/O ポート周りについて

ここで、I/O ポート絡みの部分について少し補足しておこう。

#### A-1) LED まわりの回路

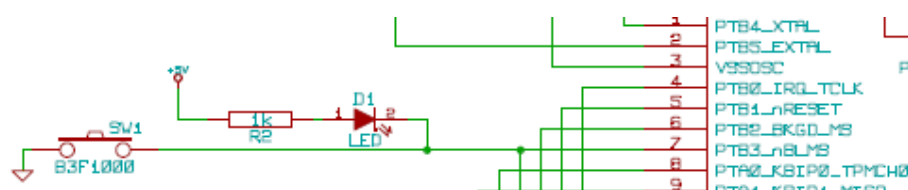
MFD08JS 基板の LED 部分の回路はこんな具合になっている。PTB3( PortB のビット 3 )が LED に繋がっていて、

- ・'0'を書くと点灯
- ・'1'を書くと消灯

となる。

MC9S08JS の場合、PTA ( Port A ) と PTB ( Port B ) の二つの I/O ポートを持っている。各ポートは 8 ビット幅あって、ビットごとに入力 / 出力のどちらで使用するのか設定できるようになっている。

端子は内部 I/O 用の入出力信号端子としても利用される。内部 I/O 用の端子として設定した場合にはプログラムでポートを制御したり状態を読み出すことはできなくなる。



## A-2) MC9S08JS のポートの設定とアクセス

MC9S08JS のポート制御用のレジスタには次のようなものがある。

- ・ PTxDD ( x は'A'または'B' : 以下同様 ) : データディレクションレジスタ  
ピンを出力ポートにするのか('1') 入力ポートにするのか('0') の設定
- ・ PTxD : データレジスタ  
出力に設定したピンに出力するデータを設定
- ・ PTxPE : プルアップイネーブルレジスタ  
入力に設定したピンのプルアップ抵抗をイネーブルにする ('1') か否か ('0')
- ・ PTxSE : スルーレートコントロールイネーブルレジスタ  
出力時に出力状態を変化させたときの電圧の変化率 (スルーレート) 制御を行う ('1') か否か ('0')
- ・ PTxDS : ドライブストレングスレジスタ  
出力時のポート駆動力が大きいモード('1')か小さいモード ('0') にするか

このうち、PTxSE と PTxDS レジスタはとりあえず全部 0x00 (リセット後の初期状態) にしておけば良い。

今回のサンプルでは次のように設定した。ポート A は使っていないので、とりあえず全部入力にして、プルアップ抵抗をイネーブルにした。

ポート B はビット 0 は未使用、ビット 1 はリセットスイッチに使っているので、この両方をプルアップしている。

### ・ ポート A

```
PTAD  = 0x00;
PTADD = 0x00;
PTAPE = 0xff;
PTASE = 0x00;
PTADS = 0x00;
```

### ・ ポート B

```
PTBD  = 0x08;
PTBDD = 0x08;
PTBPE = 0x03;
PTBSE = 0x00;
PTBDS = 0x00;
```



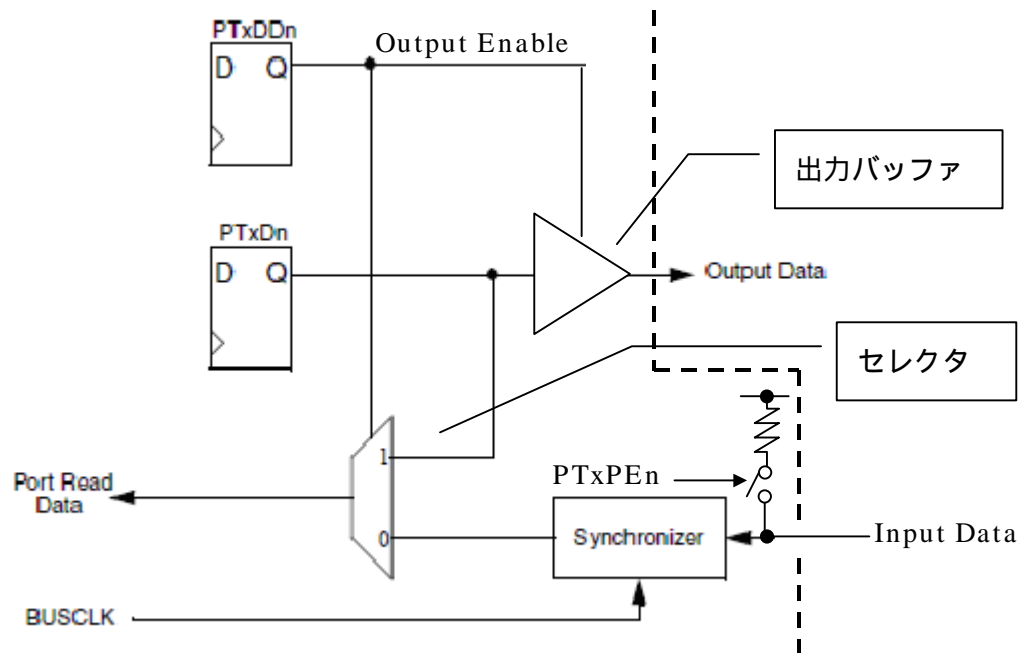


Figure 6-1. Parallel I/O Block Diagram

左上の PTxDDn や PTxDn というのは CPU から操作するレジスタでプログラムで値を設定するもの。PTxDn は「Data Register」で、ピンの状態を設定するもの、PTxDDn は「Data Direction Register」で、ピンが入力用なのか出力用なのかを設定するもので、「1」にすると出力用、「0」なら入力用となる。

PTxDDn の x 部分はポートの種別 (MC9S08JS なら 'A' か 'B') が入り、n はビット位置を示している。ここでは LED 制御用として PB3 (PortB のビット 3) を操作するので、扱うのは PTBDD / PTBD のビット 3 ということになる。

たとえば、PTBDD に 0x30 を設定すれば PTB (PortB) のビット 5 とビット 4 が出力、他は入力となり、0x08 を設定すれば、PTB のビット 3 だけが出力、他は入力ということになる。

PTB3 が出力モードのときに PTBD に \$00 を書き込めば PTB のビット 3 は '0' ('L'レベル) になり、PTBD に \$08 を書き込めば PTB3 は '1' ('H'レベル) になる。

右側の三角形は出力バッファで、上から入っている信号が '1' の時に ON 状態になる。この場合には PTxDDn が '1' になっていると、PTxDn に設定した値が端子 (Output Data) に出力される。

このバッファのドライブ能力や立ち上がり / 立下り時のスルーレート制御をしているのが PTxDS, PTxSE レジスタである。

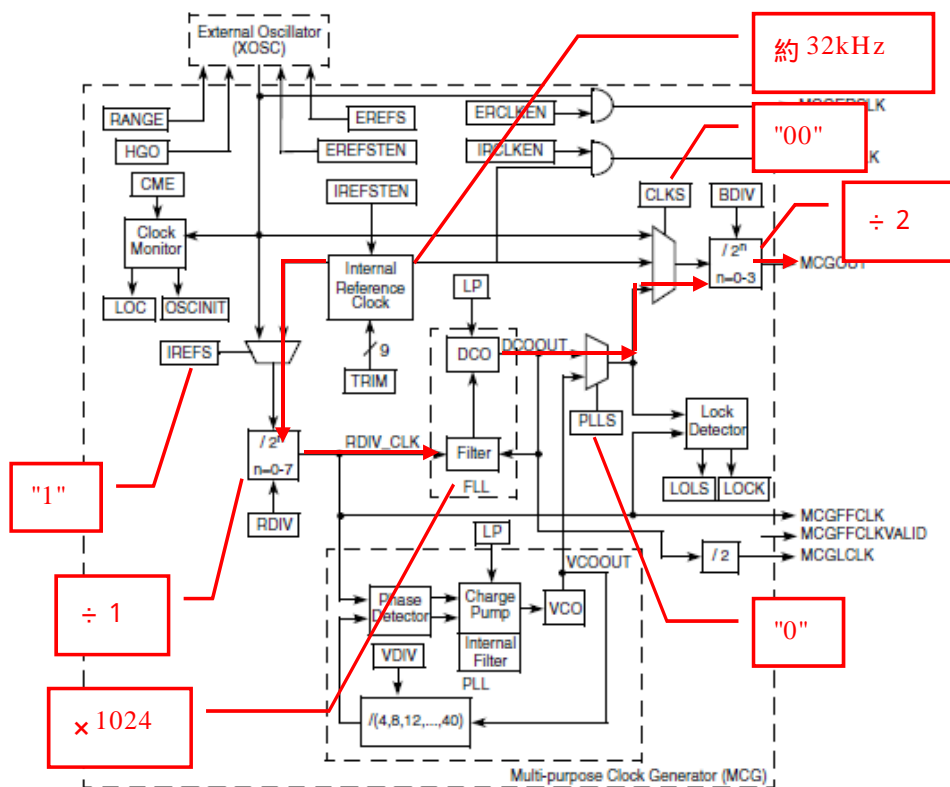
## B：クロック関係の設定

MC9S08JS のクロック系統は下図のような構造になっている。

MC9S08JS のクロック生成の大元になるクロックは左上にある「External Oscillator」と中央やや上の方にある「Internal Reference Clock」である。External Oscillator は今回の基板では 12MHz の水晶が付いているので、発振回路を動作させると 12MHz のクロックが得られる。一方、Internal の方はというと約 32kHz の周波数発振回路になっている。

PLL（中央下部）と FLL（中央部分）はこれら的大元となるクロックを元にしてより高い周波数を生成するもので、PLL は 4,8,12・・・といって最大 40 倍まで、FLL の方は 1024 倍の周波数を得られるようになっていて。（もちろん、上限はある）

リセット後の状態ではクロック信号の流れは図の赤線のようになっていて、32KHz の Internal Clock を FLL で 1024 倍してから 1/2 にした、約 16MHz のクロックが生成されて、これが CPU に与えられる。この設定状態は FEI(FLL Engaged Internal)モードと呼ばれている。



**FEI モード時のクロック信号の流れ**

## B-1)モード切替

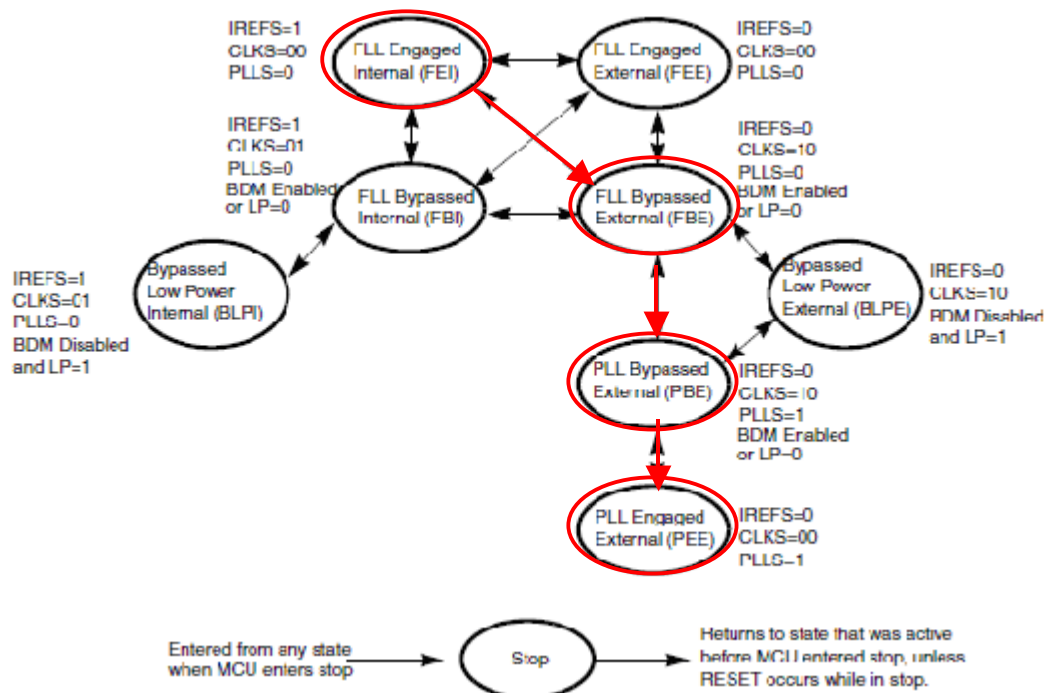
なにかと便利な内蔵発信器ではあるが 32KHz の発信器は精度や安定度が悪く、このままでは USB などには使えない。また、動作周波数もリセット後のデフォルト状態のままでは 16MHz と少々遅い。

これを外部水晶（今回の MFS08JS では 12MHz）を使い、PLL を使って 4 倍にして 48MHz のクロックを得るのがサンプルで示した Init\_Clock()関数。

この切り替えは一発ではいけず、図で赤で示したように

FEI FBE PBE PEE

という具合に移動する。



## クロックモードの遷移

なんだか難しそうだけど、要するに

- 1) デフォルトの内部クロック => FLL で駆動 (FEI)
- 2) CLKS を"10"にして外部クロック直結動作モードにする (FBE)
- 3) 外部クロック直結のまま FLL と PLL のセクタ (PLLS)を PLL 側に切替 (PBE)
- 4) CLKS を"00"にして PLL 出力に切り替え (PEE)

という具合

どんな風にな変わっていくのか、ブロック図で追ってみると良いだろう。