

[第1章]

WIZ-CとCCS-Cの違い、移植の際の注意事項などを説明

FED WIZ-CとCCS-Cの二つのC言語

本書では、FED WIZ-CとCCS-Cの二種類のPIC用Cコンパイラを使ってC言語でプログラミングしています。この章ではプログラムを作成する上での相違点、注意点を説明します。

マイコンの開発では、開発言語とデバッグ環境の使い勝手によって、その開発効率が左右されます。パソコンなどでは使いやすい統合開発環境が入手できますし、GNUのC言語を始めとする各種開発言語も利用できます。しかし、マイコンの世界では、アセンブラは経験しておかないといけませんし、メインにC言語を使う場合、メーカーによって仕様や利用環境が異なるので、自分にあったものを見つけなければスムーズに開発は進みません。

とくに、I/Oを扱うことが中心となる組み込み用マイコンでは、ビット単位の入出力などの記述方法が、コンパイラによって独自になっている場合もよくあります。したがって、本書では、開発を始める前に、筆者が使いやすいと感じているWIZ-Cとこの業界でよく使われるCCS-Cについて特徴を説明をします。

数年前まで、16Fシリーズではメモリの容量が少なく、C言語での開発よりアセンブラが利用されることが多かったと思われていますが、ここに来て、メモリの容量が拡大された製品が低価格になってきているのでC言語による開発が増えていきそうです。

1-1 WIZ-CとCCS-Cの概要

WIZ-C

FED(Forest Electronic Developments)社のCコンパイラであるWIZ-Cは低価格ながら、12、16シリーズだけでなく18シリーズも含めて多くのPICに対応しています。標準でプロジェクト機能、エディタやデバッガ(ソフトウェア・シミュレータ)、I/Oデバイス設定機能(アプリケーション・デザイナ)などを統合した開発環境(IDE)が用意されています。

スタンダード版はフロート(浮動小数点)演算がサポートされていませんが、PICの開発にとってそれほど不自由はないと思います。フロート演算、マルチ・プロジェクト機能(複数のPICを連携してシミュレーション、デバッグできる)がサポートされているプロフェッショナル版もあります。本書

ではスタンダード版(STD) Ver11.05を使用しています(図1-1)。

見本

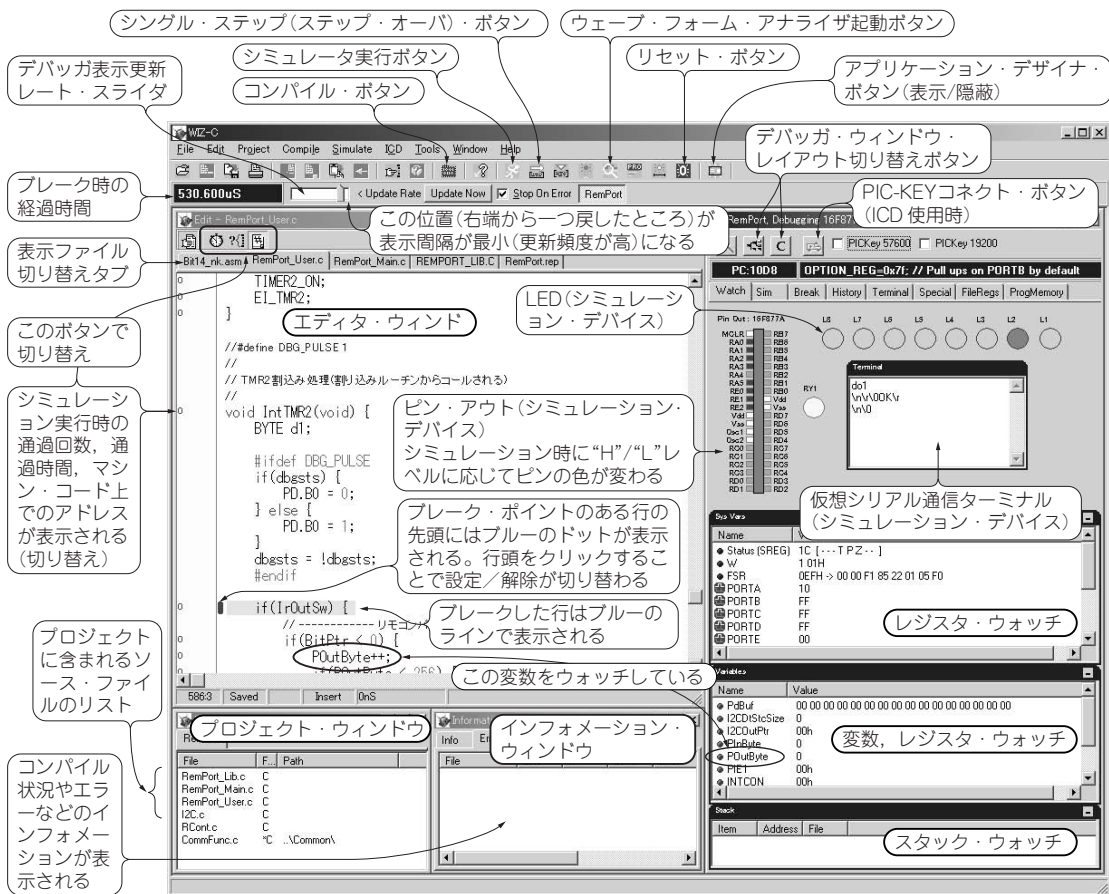


図1-1 FED WIZ-Cのメイン画面(WIZ-C Ver.11.05)

WIZ-CのIDE画面の一例。いくつかのレイアウトがあらかじめプリセットされている。左側はエディタ、プロジェクト、インフォメーションの各ウィンドウ、右半分はシミュレータ(デバッガ)ウィンドウ。この画面は第9章のリモート・ポート開発時のもの。

FED WIZ-Cスタンダード版(1万数千円*)

(* 価格は参考値)

CCS-C

CCS(Custom Computer Services)社のCコンパイラであるCCS-Cは、PICの種類によりコンパイラにいくつかの種類があり、それらを使い分ける必要があります。また、それらを統合してさらにプロジェクト機能、エディタやICD用デバッガ、内蔵デバイスの初期設定機能(プロジェクト・ウィザード)などを加えた統合環境(IDE)版のPCW、PCWHという製品も存在します。本書ではPCWHを使用していますが、掲載のプログラムをコンパイルする場合はPCW、PCMも使用可能です。

見本

CB 12ビット・コア用 コマンドライン・コンパイラ(3万円弱*)

Column 1-1 PCMを使ってコマンドラインでコンパイル(CCS-C)

PCWやPCWHは高価でなかなか手が出ないかもしれませんが、比較的安価なPCM単体でコンパイルする方法を紹介します。PCMはMPLABに統合して使用できるほか、単体でもコマンドライン・コンパイラとして使用できます。ここで紹介する方法は、MPLABも必要ありません。

第6章のシンプル・タイマを例にして説明します。この例では、次のようなフォルダ構造になっているものとします([]はフォルダを示す)。

C:

```
[CCSC_SRC]
  [SmpTimer]
    SmpTimer.c……ソース・ファイル
  [Common]
    CommFunc.c……共通関数ソース・ファイル
  reg12f683.h ……レジスタなどの定義ファイル
```

(1) まず、コマンドプロンプトを起動します。Windowsの“スタート”ボタンをクリックして“ファイル名を指定して実行”で“cmd”(Win

dows 98, MEの場合は“command”)を実行します。これでコマンドプロンプト・ウィンドウが開きます。

(2) コンパイル対象のソース・ファイルがあるドライブ、フォルダへ移ります。

```
C:¥>cd CCSC_SRC¥SmpTimer [ENT]
```

(3) コンパイルします。

```
C:¥CCSC_SRC¥SmpTimer¥CCSC +P SmpTimer.c [ENT]
```

“+P”はコンパイル終了時にステータス・ウィンドウの表示を画面に残すというコンパイル・オプションです。ほかのオプションを指定する場合は、スペースで区切ってソース・ファイル名の前に並べます。オプションが多い場合は、バッチ・ファイルを作っておくと便利です。

(4) コンパイルを始めると、Windows上に進行具合などを示すステータス・ウィンドウが表示されます。コンパイル結果は生成されたファイルでも確認できます。“SmpTimer.err”は、エラー・ステータス、“SmpTimer.lst”はメモリやスタックの使用量とアセンブリ・リストが入っています。

PCM 12,16シリーズ(14ビット・コア)コマンドライン・コンパイラ(3万円弱)

PCH 18シリーズ(16ビット・コア)コマンドライン・コンパイラ(3万円強)

PCW PCB + PCM + IDE(統合環境)付き(6万円強)

PCWH PCB + PCM + PCH + IDE(統合環境)付き(7万円程度)

(* 価格は参考値)

コマンドライン・コンパイラとは、Windowsのコマンド・プロンプトからコマンドを入力してコンパイルするタイプのことで、バッチ・ファイルや“makefile”を利用する人にはこちらのほうが使いやすい場合があります(Column1-1参照)。筆者も以前はPCMでMS-DOS版の“make”を使ってDOS窓からコンパイルしていました。ただし、I/Oを視覚的に設定してコードを自動生成するというような機能はないので、自分でコーディングしなければなりません。

また、コマンドライン・コンパイラはMPLABに組み込んで、MPLABのプロジェクトから使用

見本

することもできます。各コンパイラはCCS社のWebサイトなどからダウンロード版を直接購入すれば安価に入手できま

このアイコンは、章末に用語解説があります

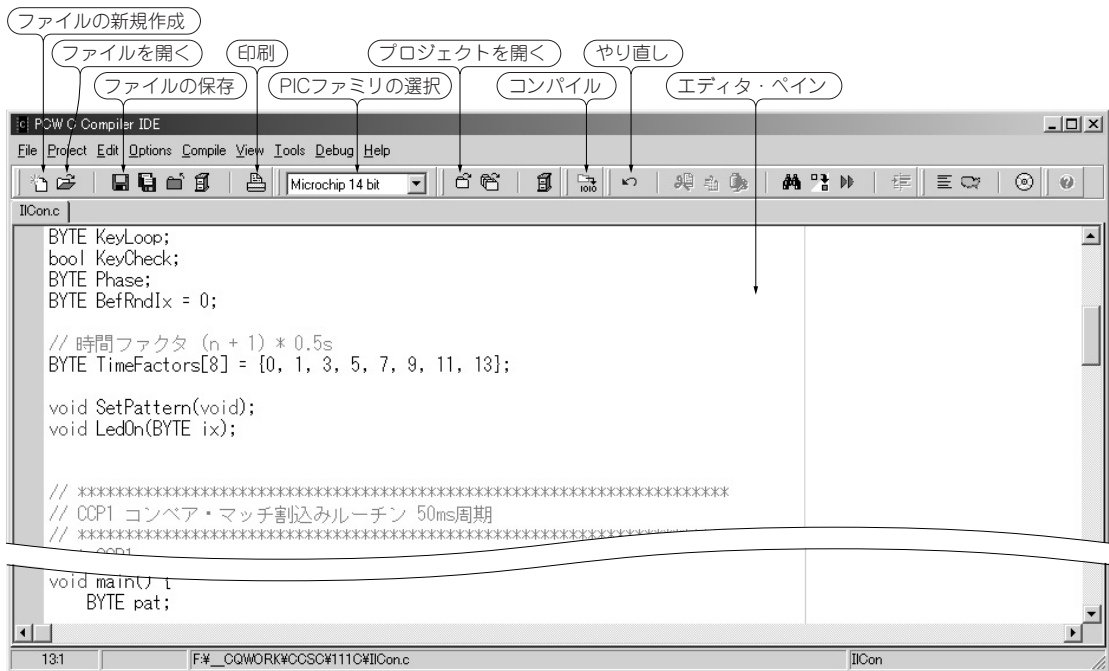


図1-2 CCS-Cのメイン画面 (CCS-C PCW/PCWH)

CCSC (PCW/PCWH) の IDE 画面。ICD を使用すれば IDE のデバッグ機能が使用できる。単独ではソフトウェアによるシミュレーションはできない。

すが、当然ながらサポートは CCS 社から直接受けなければなりません (図1-2)。なお CCS-C は売り切りなので、1 ヶ月を過ぎるとバージョンアップはできません (次回は新規購入になる)。

1-2 開発環境の違い

● プロジェクトの新規作成

プログラムを作成し始めるときに、まずしなければならないことは I/O ポートの割り付けやその入出力の定義、TIMER や USART などの内蔵モジュールの定義、設定です。

WIZ-C や CCS-C (PCW, PCWH) には、それらをサポートする機能が用意されています。

WIZ-C

WIZ-C ではアプリケーション・デザイナーと呼ばれる、アイコンのドラッグ&ドロップで I/O ポートや PIC 内蔵モジュール、ライブラリ関数を登録、設定できる使いやすいツールが標準で用意されています。

このツールの特徴は、視覚的に I/O やモジュールをピンに割り当てたり、パラメータをプロパティとして定義できる点ですが、さらに、いったんプログラムを自動生成したあとでも、モジュールの追加/削除、設定値の変更などの操作ができます (Windows アプリケーションの開発言語の IDE では一般的なことだが)。アプリケーション・デザイナーで加えた変更は、再コンパイルのたびに自動的

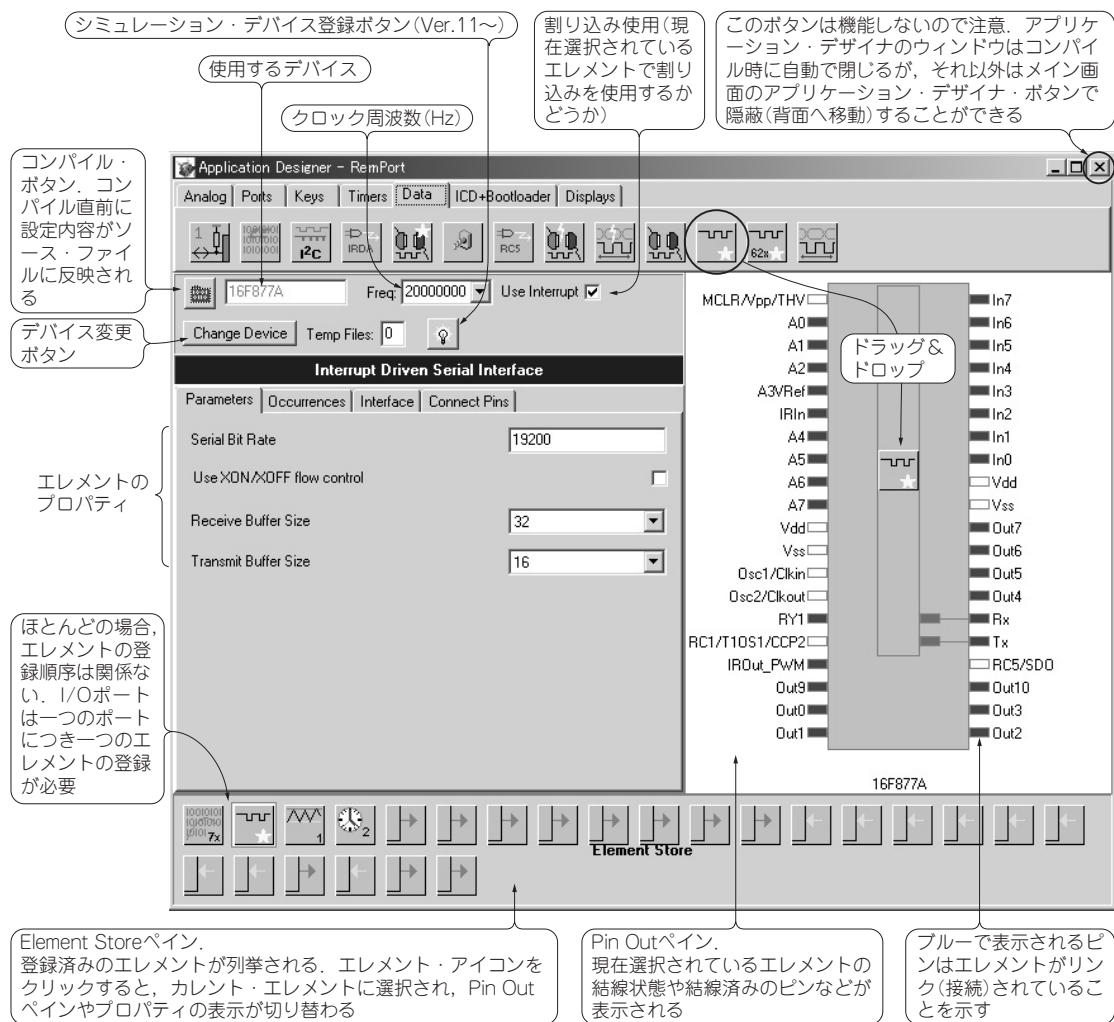


図 1-3 WIZ-C のアプリケーション・デザイナー画面 (WIZ-C Ver.11.05)
 デバイスや I/O を視覚的に設定できる WIZ-C の標準ツール。Ver.11 ではシミュレーション・デバイスがワンタッチで登録できるボタンが追加されたため、キーボードや LCD の登録が非常に便利になった。

にソース・ファイルに反映されます (図 1-3)。

CCS-C (PCW, PCWH)

CCS-C では、プロジェクト・ウィザード (PIC ウィザード) と呼ばれるキック・スタート用のツールが用意されています。これは、新規プロジェクト作成時に I/O やモジュールを定義するツールで、プログラムのテンプレート (雛型) を作成してくれます。モジュールやそれに関する初期化ルーチンも自動で組み込まれます。

見本 このプロジェクト・ウィザードは、WIZ-C のアプリケーション・デザイナーと違い、最初にコードを生成してくれるだけで、いったん確定すると、それ以降の変更はソース・ファイルを自分で書き換

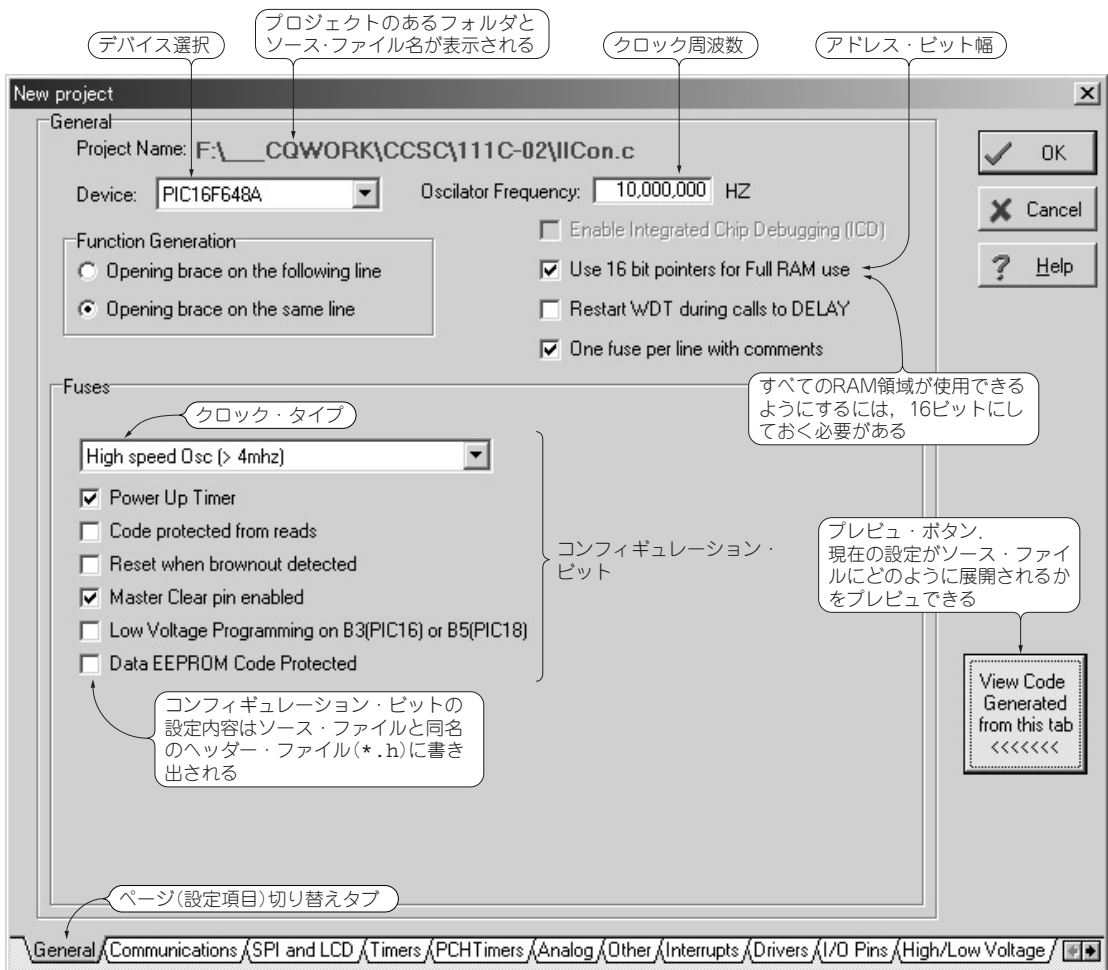


図1-4 CCS-CのPICウィザード(CCS-C PCW/PCWH)

PCWまたはPCWHのIDEから呼び出されるウィザードの表示例。“OK”ボタンを押して確定させた後は変更できないので注意(手作業で修正が必要)。手修正の際は仮のプロジェクトを作って、修正した内容をプレビュー表示させ、その内容をコピーしてソース・ファイルに貼り付ければよい。

えるか、新規にプロジェクトを作成し直さなければなりません。なお、既存のプロジェクト名(ファイル)を指定して新規作成すると、既存のソース・ファイルが上書きされてしまうので注意してください(図1-4)。

● デバッガ/シミュレータ

WIZ-C

見本 WIZ-Cには使いやすいソフトウェア・シミュレータが内蔵されています。これは、シミュレーションデバイス(External Device)を使ったシミュレータで、パソコン上の仮想PICで動作する仮

想デバイスが、実機が動いているようにパソコンの画面上で動きます。このデバイスにはLEDをはじめ、7セグメントLED、LCD(液晶表示器)、4×4キーパッド、仮想ターミナル(仮想PICとシリアル通信を模擬)などがあります。場合によってはこれだけでPICにプログラムを書き込む前にデバッグが終わります(図1-1の右側参照)。

また、出力ポートの状態をロジック・アナライザのようにグラフ上に表示させるツール(Waveform Analyser)も標準で付属しているので、シミュレーション実行した後に出力状態を確認し、パルス幅を測定するというようなこともできます(図9-9参照)。

WIZ-CではICD(In-Circuit Debugger)としてPIC-KEYが使用できます。これはPICプログラマ(ICSP ; In-Circuit Serial Programming)としても使用できます。PIC-KEYについては第11章で説明します(写真11-1参照)。

CCS-C

他方、CCS-Cの場合はC言語ソース・レベルで動くシミュレータは内蔵されていません。MPLABを併用してアセンブラ・ソース・レベルでシミュレートしなければなりません。

ICDを使えばデバッグが使えます。その場合はマイクロチップ社純正かCCS社のICDを使用します。PIC-KEYは使用できません。CCSのデバッグICD-U40については、第11章で説明します(写真11-2参照)。

1-3 プログラムの構造、言語仕様上の違い

● 自動生成されるテンプレート・ファイル

WIZ-Cはアプリケーション・デザイナ、CCS-C(PCW, PCWH)ではプロジェクト(PIC)ウィザードを使用すれば、テンプレート(雛型)ファイルが自動で生成されます。

WIZ-C

WIZ-Cではアプリケーション・デザイナを使用する場合、プロジェクト新規作成時に“xxx_main.c”と“xxx_user.c”という二つのファイルが自動生成されます。“xxx”の部分にはプロジェクト名が入ります。“xxx_main.c”にはmain関数と割り込みルーチンが作られますが、ユーザは通常は“xxx_main.c”のほうには手を加えることはありません。main関数からはUserLoop関数、割り込みルーチンからはUserInterrupt関数がコールされるようになっていて、この二つの関数は“xxx_user.c”のほうに作られます。

ユーザは、これらの関数の中に処理を追加していきます。UserLoop関数はmain関数の無限ループの中からコールされているため、UserLoop関数の中で処理をループさせる必要はありません(*)。

なお、アプリケーション・デザイナを使う設定になっている場合は、“xxx_main.c”のほうはコンパイルのたびに上書きされるため、修正を加えても無効になります。修正したい場合はアプリ

見本 (*) プログラムの説明でメイン・ループと表現している場合があるが、WIZ-Cの場合、とくに注記がない場合はユーザ・ループ(UserLoop)を指す。読み替えてください。

ケーション・デザイナをOFFに切り替えてから修正します。

CCS-C

CCS-Cではプロジェクトの新規作成時に生成される“xxx.c”ファイルの中にmain関数とブランチの割り込みルーチンが作られます。ウィザードで設定している場合は設定に応じてmain関数の中に内蔵デバイスの初期化処理やI/Oポートの初期化処理などが自動で展開されます。ユーザはこのmain関数や割り込みルーチンの中に処理を追加していくことになります。

ウィザードで厳密に設定しなくても、PIC種別、クロック周波数やコンフィギュレーション・ビットなどを大雑把に設定しておいて、後から自分でソース・ファイルに処理を加えたり、修正していてもかまいません。

● 文字の扱い

WIZ-C

WIZ-Cでは標準で大文字、小文字を区別します。オプションで区別なしにも設定できます。コメントに日本語を使う場合は文字コードにユニコードが使われます。ほかのテキスト・エディタでソース・ファイルを閲覧、編集する場合はユニコードに対応したものがが必要です。

また、コメント内の日本語で一部の文字がコンパイル・エラーを引き起こすものがあります。筆者が認識しているものを挙げておきますので、WIZ-Cではこの文字は使わないようにしてください(文脈によってはエラーにならない場合もある)。なお、原因がわからないコンパイル・エラーがある場合は日本語文字を疑ってみてください。

「最」, 「上」, 漢数字の「一」, 「言」

たとえば、「最上位」, 「一致」, 「宣言」などがだめです。最初の頃はこれでたいぶ泣かされました。

CCS-C

CCS-Cではデフォルトでは英文字の大文字、小文字を区別しません。BASICなどと違い、通常のCコンパイラは区別があるのが普通なので、プログラムを移植する際に、大文字と小文字で区別しているような変数名、関数名がある場合は注意してください。

● 型(データ・タイプ)の違い

WIZ-Cのスタンダード版はfloatが使えません(プロフェッショナル版では使用可)。

CCS-Cは比較的高価なだけあって、標準でfloat(浮動小数点)が使えます。CCS-Cの最初の頃はlong(32ビット整数)がなかったのですが、Ver3から追加されました。

その他、注意しなければならないのが、shortです。CCS-Cのshortは1ビットですが、WIZ-Cを含む通常のコンパイラではshortは8ビット(32ビット・プロセッサの場合は16ビット)です。

CCS-Cで普通のshort intのつもりでうっかりshortを使うとなかなか気づかないバグに悩まされることになります。shortは混乱を避けるためにも使用しないほうがよいでしょう。型の対比

見本

表1-1 主なデータ・タイプ(型, 同じ型名でも WIZ-C と CCS-C でビット幅が違うものがあるので注意)

主なデータ・タイプ(型)

	FED WIZ-C [ビット] (Ver.11 STD版)	CCS-C [ビット]
int	16	8
short	8	1
char	8	8
long	32	16
bit	1	-
int1	-	1
int8	-	8
int16	-	16
int32	-	32
float	-	32 浮動小数点
BYTE *	8(unsigned)	-
WORD *	16(unsigned)	-

表現できる数値範囲

データ長 [ビット]	signed	unsigned
8	- 128 ~ 127	0 ~ 255
16	- 32,768 ~ 32,767	0 ~ 65,535
32	- 2,147,483,648 ~ 2,147,483,647	0 ~ 4,294,967,295

* typedef で定義されているもの

は表1-1のようになります。

● CCS-Cの関数の引数, ポインタの制約

CCS-Cでは, 関数の引数やポインタに特殊な仕様があるので注意が必要です。

関数の引数でリテラル(定数)のポインタを渡せません。たとえば, 次のような場合です。

```
void LCDString(char *str);
```

というような関数がある場合,

```
LCDString("test"); // ×CCS-C ○WIZ-C
```

は, CCS-Cではコンパイル・エラーとなります。ROM上にあるものがポインタで参照できないということです。C言語の仕様としては規格外ですが, PICでは仕方がないのかもしれませんが(WIZ-Cでは問題なし)。ただし, printfなどの引数の中にはリテラルを含められます。

```
printf("%02d", val); // ○CCS-C ○WIZ-C
```

```
strcpy(buf, "1234"); // ○CCS-C ○WIZ-C
```

また, ポインタの定義についても特殊な仕様があります。普通のC言語では, 初期値付きの変数は次のように定義できますが, CCS-Cではエラーになります。

```
char *Msg = "Hello"; // ×CCS-C ○WIZ-C
```

これは, 次のように定義する必要があります。

```
char Msg[] = "Hello";
```

また, LCDの出力に“lcd_putc”という組み込み関数がありますが,

```
lcd_putc('a'); // 1文字の出力
```

この関数は, 次のようにすれば文字列を出力することができます。

```
lcd_putc("bcd"); // 文字列の出力
```

これもおかしなものですが, 実際はコンパイラが次のように展開します。

```
lcd_putc('b');
```

```
lcd_putc('c');
```

見本

Column 1-2 割り込みとその要因

割り込みとは、文字通り本来の処理に割り込んで処理することで、一般には緊急度の高い処理を実行させますが、場合によっては発生頻度の低い(めったに起こらない)処理に利用するという使い方もあります。

割り込み処理は通常の処理からは独立したものになっていて、通常処理側はいつ割り込みが発生するかはわかりません。ただ、通常処理中に割り込まれたくない場合は、あるプログラム区間を割り込み禁止にすることができます。XXIEフラグをクリアしておくことと特定の要因に対する割り込みを禁止することができます。ただし、禁止された要因そのものでは割り込みは発生しませんが、要因フラグのXXIFはセットされていることがある、ということに注意してください。INTCONのGIEをクリアしておくこととすべての割り込みを一括して禁止/許可することもできます(グローバルな割り込みの禁止/許可)。

● 割り込み要因

割り込みは、複数の要因がORされた結果として発生します(複数のうちのいずれかの要因が発生したら割り込みが発生)。したがって、複数の要因が同時に発生しても起こるのは一つ[※]の割り込みだけです。

そこで割り込みルーチンでは、まず初めにどの要因の割り込みが発生しているかを特定しなければなりません。また、割り込み要因によっては、割り込みを受け付けたことをPICの割り込み回路へ知らせるために、明示的に要因フラグをクリアする必要があります。クリアするのを忘れると、

(※) 18シリーズでは高優先と低優先の二つの割り込み種別があるため、割り込みルーチンも二つもてる。また、要因ごとに高優先にするか低優先にするかを選択できる。

割り込みルーチンを抜けた後に再び割り込みが発生し、これを永遠に繰り返して通常処理に戻らない、というようなことも起こります。他方、シリアル・データの受信割り込みでは、受信データをRCREGから取り出すことでRCIFがクリアされるため、直接RCIFをクリアする必要はないというものもあります。

なお、I²Cモジュール使用時のSSP^①割り込みなど、同じ要因フラグでも状況により意味が変わるものもあります。たとえば、SSPIFはI²Cの通信状況に応じてストップ・コンディション発行完了、ACK受信完了など違った意味でセットされます。

WIZ-Cの場合はこのような要因の特定、要因フラグのクリアなどの処理は、ユーザがプログラミングする必要があります。CCS-Cの場合はコンパイラが自動で要因を特定してそれに応じた割り込みルーチンへジャンプし、要因フラグは自動でクリアされるようになっています。

どちらが優れているとはいえませんが、WIZ-Cの方法はプログラムの手間はかかりますが、自由度が高いということはいえるでしょう。

● おもな割り込みの種類

次に、おもな割り込みの種類を示します。デバイスによっては実装されていないものもあります。

- ◆ タイマ(カウンタ)のオーバフロー、比較一致
- ◆ CCPの比較一致、キャプチャ完了
- ◆ RB0の外部割り込み
- ◆ RB3~RB7の状態変化
- ◆ シリアル通信の1バイト送受信完了
- ◆ SSP(I2C, SPI)の各種動作完了
- ◆ A-D変換完了
- ◆ データEEPROM書き込み完了
- ◆ その他

見本

```
lcd_putc('d');
```

このように展開してくれるのは、この“lcd_putc”だけのようです。

● 割り込み処理の記述方法の違い

WIZ-C

WIZ-Cの場合は割り込み発生時はUserInterrupt関数が呼び出されるだけなので、その中で割り込み要因^⑧を自分で判断し、場合によってはその要因フラグをクリアしてやらなければなりません(割り込みの復帰処理は自動で組み込まれる)。

WIZ-Cには割り込みの種類別に「クイック割り込み」と「通常割り込み」の2種類ありますが、アプリケーション・デザイナを使う場合はクイック割り込みに固定されています。両者の違いは割り込み発生時/終了時に退避/復帰させるレジスタやワーク・エリアの数の違いで、クイック割り込みのほうが退避/復帰数が少なく、オーバヘッドが小さくなります(処理速度が向上)。

その代わりに、割り込み処理の中で実行できることに制限がでます。処理できるのは単純なビット演算や加減算、比較などに制限されます(剰余算など不可)。ローカル変数の使用やC言語の関数の呼び出しはできない(アセンブラで作成した関数は可)とされていますが、実際には使用できる場合もあります。ただ、たまたま動いているだけかもしれないので、使わないようにしたほうがよいでしょう。


処理が複雑になる場合は、通常割り込みを使用してください。第3章のイルミネータ(拡張版)、第9章のリモート・ポートでは、通常割り込みを使用しています。

CCS-C

CCS-Cの場合は、割り込み要因ごとのプリプロセッサ(#int_XXX)の直後に書かれた関数とその要因に対する割り込みルーチンとなります。割り込み要因ごとに分かれた関数がコールされるように

Column 1-3 WIZ-Cの割り込みタイプの変更方法

WIZ-Cでアプリケーション・デザイナを使用している場合は、割り込みタイプはクイック割り込みに固定されています。これを、通常割り込みに変更する手順を説明します。

(1) アプリケーション・デザイナで一通り設定が終わったら、一度  (Generate Application) でコンパイルします。

(2) メイン画面に戻り、メニューの“Project”-“Use Application Designer”をクリックして同項目のチェックを外し、アプリケーション・デザイナを使用しないように設定します。

(3) “xxx_main.c”の“Interrupt”関数の直前にあるステートメントを次のように修正します。

```
const int QuickInt=1;
→ const int NormalInt=1;
```

(4) 最後にコンパイルし直します。

なお、再びアプリケーション・デザイナをONにしてコンパイルすると、この変更は元に戻ってしまうので注意してください。その場合は、もう一度同じ作業が必要です。

見本

なります。要因フラグ(XXIF)のクリアは不要です。

WIZ-Cのほうが多少面倒ですが、割り込みの順序や複数の割り込みが関係するような処理を作るときはWIZ-Cのほうが自由度があって都合がよい場合もあります。なお、CCS-Cにも同等の機能が用意されています。

PICの割り込みで注意すべきことがあります。それは特定の要因の割り込みを個別にプログラムの中で許可、禁止する場合に起こることです。PICの場合、xxIEフラグを‘0’に設定して割り込みが発生しないように設定したつもりでも、割り込みの起動がかからないだけで、xxIFはセットされていることがあります。この状態で、ほかの要因の割り込みが発生すると、xxIEでマスクしたはずのxxIFの処理が動いてしまいます。

たとえば、TIMER0は割り込みを禁止(TMR0IE=0)していてもオーバフローするたびにTMR0IFがセットされます。リセットしない限りセットされたままになるので、TIMER0の割り込み処理で要因の判断にTMR0IFだけをチェックしていると、ほかの要因の割り込みが発生した際にTIMER0の割り込み処理が誤って動いてしまいます。

これを防止するには、要因のチェック時にxxIFだけでなく、xxIEもセットされているかを調べる必要があります。気が付きにくい問題なので、おかしいと思ったときにはチェックしてみてください。グローバル(一括)で許可/禁止する場合は問題ありません。なお、CCS-Cの現在のバージョンではこの問題はありません。

● EEPROMの書き込みの違い(ライブラリ)

WIZ-C、CCS-CともにPICに内蔵されているEEPROMの読み書きができるライブラリ関数が用意されていますが、書き込みの処理に相違があります。

WIZ-C

WIZ-Cの“writeEEData”は、EEPROMへ書き込んでも書き込みの終了は待たずに関数を抜けます。EEPROMの書き込みには数msかかるので、続けて書き込もうとすると直前の書き込みが終了していないため書き込みは失敗します。

CCS-C

CCS-Cの“write_eeprom”は書き込みが完了するまで先に進まないため、続けて書き込んでも書き込みに失敗することはありません。

使うときには少し注意が必要ですが、“read_eeprom”、“write_eeprom”はマクロなので、多用するとプログラム量が増大します。多用する場合は別に関数を作って、その中で“read_eeprom”、“write_eeprom”をコールするようすればよいでしょう。

一見、WIZ-Cのほうが使いにくいように見えますが、書き込みの完了を待たないということはその次の処理(EEPROMの書き込み以外の処理)にすぐ移れるということです。CCS-Cの場合は書き込みが完了するまで数msの間、ほかの処理ができません。本書で今

見本

回掲載したプログラムではWIZ-Cでも書き込みを完了するまでは次の処理に移らないように簡易的な作りになっていますが、全体の処理効率を上げたい場合は、この数msのむだ時間が問題になる場合もあります。とくに書き込むバイト数が多い場合はなおのことです。

この場合は1バイトのデータを書き込んだら、メイン・ループまたは割り込み処理でそのデータの書き込み完了を検知して、次のデータを書き込むという処理が必要です。こういう用途にはCCS-Cの“write_eeprom”は使用できないので、独自に関数を作成しなければなりません。

● インライン・アセンブラ

C言語の中にアセンブリ・コードを記述することができます。WIZ-CとCCS-Cでは記述方法に違いがあるので注意してください。また、コメントの付け方にも注意してください。

Column 1-4 WIZ-Cのライブラリ拡張

標準のインストールをただけでは、WIZ-Cの一部のライブラリはインストールされません(WIZ-C Ver.11以前)。本書でも使用している“atoi”はその一つです。そのため拡張ライブラリをあらかじめインストールしておかなければなりません(製品に付属)。

追加の手順を次に示します。

(1) WIZ-CのセットアップCD-ROMから“Library Extensions”をインストールします。

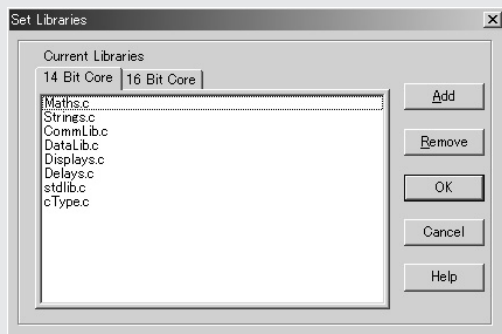


図1-A ライブラリの追加画面

ライブラリを追加するためのウィンドウ。14ビット、16ビットそれぞれのシートに必要なに応じてライブラリ・ファイル(*.c)を追加する。

(2) WIZ-Cを立ち上げてメニューから“File” - “Libraries”をクリックします。図1-Aのようなウィンドウが表示されるので、“14 Bit Core”のページを表示させた上で“Add”ボタンをクリックし、追加したいライブラリ・ファイル(*.c)を選択します。本書では“stdlib.c”を追加しています。WIZ-CがCドライブの標準的な位置にインストールされている場合は、ライブラリ・ファイルは“C:\Program Files\FED\PIXIE\Libs”にあります。

(3) 16ビット版(18Fxxxなど)でも使用する場合は、“16 Bit Core”のページでも同様の操作でライブラリを追加します。ライブラリによっては14ビット版と16ビット版の二つが供給されている場合があります。そのときは、対応するページにそれぞれ追加してください。16ビット版は“xxx16.c”というファイル名になっています。

(4) 最後に“OK”をクリックしてウィンドウを閉じます。

なお、Ver.12からは、標準的なライブラリはすでに登録されているので、あらためて登録する必要はありません。

見本

WIZ-C

```
#asm                // 複数行
    NOP             ; no operation
    CLRWDT         // clear WDT

#asmend
#asmline NOP        // no operation(単一行)
#asmline CLRWDT    // clear WDT(単一行)
```

“pragma”は省略しています。

CCS-C

```
#asm                // 複数行
    NOP             // no operation
    CLRWDT         // clear WDT

#endasm
```

CCS-Cの場合、コメントの開始にアセンブラのようなセミコロン“;”は使用できません。

● WIZ-CのOccurrence(イベント・ハンドラ)

WIZ-Cでは、エレメントによってはOccurrenceと呼ばれるイベント処理が実装できます。たとえば、シリアル通信でデータを受信したときにOccurrence関数がコールされるというものです。キーボードの場合はキーが押されたときにそれに応じたOccurrence関数がコールされます。

このOccurrence関数は、いわゆるイベント・ハンドラと呼ばれるものです。このような処理は割り込み処理などでフラグを立てて、それをメイン・ループでチェックするようなことで独自に作ることも容易ですが、アプリケーション・デザイナーを使用すればこのしくみがmain関数の中に自動で組み込まれ、ユーザ・プログラムから分離・隠蔽されるため、ユーザ・プログラムではOccurrence関数を記述するだけで済むという、プログラムのすっきりしたものになります。場合によってはUserLoop関数は空でOccurrence関数だけでプログラムができあがるというケースもあります。

このOccurrenceを使うかどうかの設定(Occurrenceが発生したときコールされる関数の登録)は、アプリケーション・デザイナーで行えます。

● WIZ-Cのスタック

WIZ-Cは一般的なC言語のようにオプションでスタック・エリアをRAM上に確保することができます。そのため、12、16シリーズのPICがもつスタックの8レベルを超えたスタック・エリアを使うこともできます。その代償としてスタック用のメモリ・エリアが必要になると、そのPUSH-POP処理に伴うオーバーヘッド(プログラムの増加や処理速度の低下)が発生します。デフォルトではPIC内蔵のスタックを使用するようになっています。

なお、アプリケーション・デザイナーを使用して作成したプログラムは、main関数からUser

Loop関数をコールした時点でスタックを1レベル消費しているので注意してください。

1-4 エlement, ライブラリの比較

WIZ-C

WIZ-Cのライブラリ関数は、アプリケーション・デザイナーによりElementの一部として使用されます。一つのElementには複数のライブラリ関数を含んでいるものもあります。たとえば、EEPROM Elementは“WriteEEData”, “ReadEEData”の二つのライブラリ関数を含んでいます。アプリケーション・デザイナーを使用しない場合は、ライブラリ関数として直接使用することもできます。

WIZ-Cは現行バージョン(Ver.11.05)ではアナログ・コンパレータ関係のものが皆無なので、使用する場合は自分でレジスタを操作しなければなりません。また、シミュレーションもできません。本書ではアナログ・コンパレータは使用していませんが、PIC16F648Aではリセット直後はアナログ・コンパレータがONになっているため、PORTAをデジタルI/Oポートとして使用する場合には、コンパレータをOFFにしなければなりません。

このために、UserInitialise関数の初めに“CMCON=7;”というコードを入れていますが、第5章のシリアル端末では、main関数で先にLCDの初期化処理が実行された後にUserInitialise関数が実行されるため、LCDの初期化が失敗します(PORTAがまだアナログ入力のままのため)。そこで16F648A用にアナログ・コンパレータをOFFするElementを作成しました。このElementは一番最後に登録してください。16F877Aの場合は初期状態がコンパレータOFFとなっているので、使わないときは何もする必要はありません(ADCピンをデジタルI/Oに設定する処理は必要)。

PIC内蔵のモジュール(TIMERやCCP, ADC, USARTなど)もElementという形で登録します。Elementを登録するとそれに必要なヘッダー・ファイルなどは自動的にインクルードされます。また、内蔵モジュールを使用するためのライブラリ関数も使用できるようになります。

LCD ElementのDB₇~DB₄(データ)信号は上位4ビット(b₇~b₄)にアサインするという制限はありますが、PORTB~PORTDのどのポートにもアサインできます。また、“RS”, “E”, “RW”の制御信号は、任意のポートにアサインできます。キーパッド・Elementは4×4の範囲で任意の行、

Column 1-5 WIZ-Cのスタック・オプションの変更

WIZ-Cでは、標準環境ではPIC内蔵のスタック^①を使用するようになっていますが、オプションでRAMエリア上にスタック・エリアをもたせて、スタック・サイズを増やすことができます。その手順を説明します。

(1) メニューで“Project”-“Project Options”を

クリックして“Compiler Options for xxx”ダイアログ・ボックスを開きます。

(2) “Optimisations”のページで“Use PIC Call Stack”のチェックを外します。

(3) 最後にコンパイルし直します。

見本