[第8章]

具体的なデバッグ手法とその原因

デバッグの考え方とそのセオリをマスタする

見本

この章では，プログラミングが始めての方にもわかりやすいように，デバッグの方法やWIZ-Cデバッガの使い方を説明します．プログラムの多くは思い込みや勘違いで，動くはずだと思ったものが期待通りに動きません．単純なスペル・ミスもありますし，Cの文法をよく理解しなかったことに原因があったりします．でも，最初はみんなそういう経験をしています．

8-1　デバッグとは

デバッグとはDE-BUG(デ・バグ)のことで，バグを取るという意味があります．バグとは小さな虫のことで，プログラムの間違い箇所を虫に例えてこのように呼ばれています．

デバッグはプログラムを作る上では一番重要なところですが，一番厄介なところでもあります．

プログラムが完成するまでには，いろいろな間違いを探し出さなければなりません．人がやることに間違いは付き物ですが，それを一つずつ根気良くつぶしてきます．それに加え，いろいろな条件で実際に動かしてみて，正常に動いているかも確認します．シミュレーションではうまくいっていても，いざ，実際に何かの装置と組み合わせて動作させてみるとうまく動かないということもあります．これは，仕様の取り違いや，仕様で規定されていない部分などで何らかのミスマッチが起きていることが原因です．

● 間違いの要因

考えられる間違いや問題をいくつか挙げてみます．

- コーディング時のスペル・ミスなど，単純な間違い
- 言語の文法的な間違い，記述上の間違い
- 論理的なミス．たとえば，想定していない値が入ってきたときにおかしな動作をするなど
- PIC内蔵モジュールなどの使用方法，手順の間違い
- I/Oポートの番号の間違い，初期化忘れ
- 変数の初期化忘れ
- 処理の順序，関数呼び出しの順序の問題
- 演算子の優先順位を間違い，演算結果が不正になる

- 処理のタイミングが期待通りにならない
- コンパイラそのものの制限や不具合によるバグ

その他にもいろいろあることでしょう．

コーディングの単純ミスや文法上の間違いなどは，コンパイル時にエラーが表示され，エラー箇所もある程度特定できるので，比較的簡単に修正できます．よくあるのは変数名や関数名が，定義しているところと参照しているところで食い違っているというものです．この場合は未定義シンボルの参照という形でエラーになります．また，関数の場合は，引数の型や数が食い違っているというような間違いもよく起こります．

Appendix Dに間違い事例をいくつか紹介していますが，たとえば，代入する必要があるのに「==」(比較の等号)を書いてしまったり，逆にif文の中の等号を「=」(代入)と書いてしまったという場合，C言語の文法には違反していないため，コンパイル・エラーにはなりません．こういうミスは，わかってしまえばなんということはないですが，見つけるまでなかなか大変です．慣れていても，ついうっかり間違ってしまうこともあります．

コンパイラでチェックできないことは，自分で探し出すしかありません．これが一番労力の必要なところです．

● デバッグ時に確認すること

プログラムが正しく動いているかどうかを確認するポイントは，大雑把に言って次の二つです．

(1) 決められたとおりにプログラムの実行が進んでいるか

(2) 要所で変数に適切な値が入っているか

(1)は条件文やループ文で正しく分岐(またはループ)しているかということがポイントとなります．これは(2)にも関係しますが，条件式で使われている変数の値や式に依存しています．デバッグでは，条件式内の変数の値を変えるなどして，期待通りに分岐するかどうかを確認します．

その他，呼び出した関数から処理が戻ってこないために処理が先に進まない，ということもあります．この場合は，問題箇所を特定して，なぜ戻ってこないか(どこで処理が止まっているか)を探します．

(2)は演算の結果，または演算の過程で，変数の内容や演算結果が正しいか，ということです．桁数が溢れたり，符号付きの変数の符号がなくなったり，演算子の優先順位により期待通りに演算されていない，ということが考えられます．これは演算する前の変数の値と，演算結果の値を確認します．

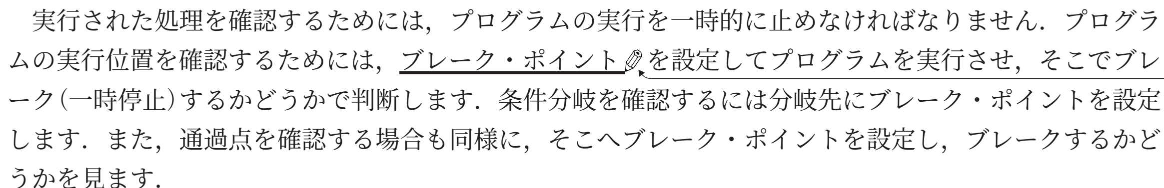
8-2　デバッグの手法

手法にはいくつもありますが，もっとも基本となるのがソース・ファイルを見直すことです．デバッガ(デバッキング・ツール)がない場合には，プログラムをPICに書き込んでは動かし，だめなら，ソース・ファイルを修正して，またPICに書き込んで... ということを繰り返すしかありません．プログラムが小規模のうちはこれでも何とかなりますが，ある程度の大きさになると，不可能ではないにしても，かなりの時間と労力，忍耐が必要になってきます．

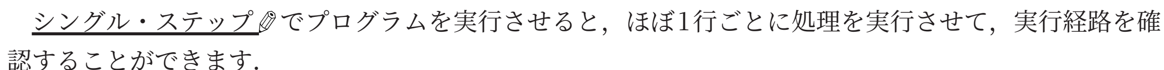
幸い，FED WIZ-Cには優秀なデバッガ(シミュレータ)が付いているので，はるかに効率的なデバッグができます．確認できないこともありますが，デバッガでの確認作業は要約すると次のようになります．

- プログラムの実行が予定通り進むか(想定ポイントを通っているか)，順を追って確認する

- 分岐点（if文などの条件判定箇所）で適切に処理が切り替わるかを確認する
- 特定の変数に注目し，その値がポイントごとに適切な値になっているかを確認する
- 関数の戻り値が適切な値（または結果）になっているかを確認する
- 割り込み発生時に，正常に割り込みルーチンに制御が移っているかを確認する
- 処理時間を測定，確認する



実行された処理を確認するためには，プログラムの実行を一時的に止めなければなりません．プログラムの実行位置を確認するためには，ブレーク・ポイントを設定してプログラムを実行させ，そこでブレーク（一時停止）するかどうかで判断します．条件分岐を確認するには分岐先にブレーク・ポイントを設定します．また，通過点を確認する場合も同様に，そこへブレーク・ポイントを設定し，ブレークするかどうかを見ます．



シングル・ステップでプログラムを実行させると，ほぼ1行ごとに処理を実行させて，実行経路を確認することができます．

特定位置の変数の値を見る場合にも，該当箇所にブレーク・ポイントを設定してそこでブレークさせて，変数の内容を見るということになります．

このアイコンは，章末に用語解説があります

＜実行位置の確認＞

- 適切な分岐先へ分岐しているか
- 特定行を実行しているか
- 割り込み処理に入ってくるか
- 割り込み処理を終わって，通常処理に戻るか
- 関数に入ってくるか
- 関数から戻ってくるか

＜演算結果，変数の確認＞

- 関数の戻り値が適正か
- 特定行の変数値が適正か
- 演算過程の変数値は適正か
- 条件文の中の変数値は適正か

8-3　PICデバッグのテクニック

● AUTO変数

WIZ-Cスタンダード版では，AUTO変数の値を直接デバッガで参照したり，設定したりすることはできません．これは，AUTO変数が宣言されるたびにスタック領域に作られ，そのたびにアドレス変わるためです．ブレークした時点でスタックの中身をのぞいてアドレスを調べれば読み書きできないこともないですが，非常に面倒です．

そこでRAM容量に余裕があれば，デバッグのときにAUTO変数を一時的に外部変数に変更してしまいます．こうすれば，デバッガで直接扱うことができます．

Column…1　思い込みは禁物

これは筆者にも当てはまることですが，この部分は大丈夫なはず，と思って確認を怠ると，そういう所に限ってバグが発生するということがあります．単純なミスは間違った思い込みや勘違いでも起こります．うまく動かない場合はとくに，すべてを疑うようにしたほうが無難です．

● 複雑な式

1行で複数の演算子を使って一気に演算させる場合，式をあまり複雑にすると正しく演算できないことがあります．また，途中の演算過程を確認するすべがありません．演算子の実行順位は決められていますが，式が複雑になると間違えたり，わからなくなることもあります．

このような場合は，外部変数の一時変数を用意して，演算を小分けにし，そのたびに一時変数に値を残すようにします．ブレーク・ポイントを設置するなどして，小分けにした演算ごとに，この一時変数の内容を確認していけば，どこで不具合が起きているかを特定しやすくなります．一時変数を外部変数にするのは，デバッガで参照/変更ができるようにするためです．

● ほかのコンパイラでロジックを確認する

これは少し特殊な方法ですが，ロジック（プログラムの動作）を確認するために，筆者がたまに使う方法です．MS-DOSやWindowsで使えるCコンパイラがあり，それでプログラムが作れることが前提ですが，特定の関数をそのコンパイラで作成して実行させ，正しく動いているかどうかを確認した後でPICへ移植するという方法です．

この方法の利点は，デバッガの機能が充実しているのと，パラメータをいろいろ変えながら，演算結果などを画面上に表示できるということです．

ただ，変数の型のビット数が違ったり，コンパイラの癖が違っていたりで，PICに移植してもすんなり動かない場合があり，ここでまた余計なデバッグが必要になる場合もあります．複雑な処理にはそれなりに利点もありますが，あまりお勧めできません．参考までに．

● プログラムの単純化

単純なモデルでプログラムを作成し，それで確実に動くようになってから本来の機能を肉付けしていく方法です．逆に，作ってしまってからうまく動かない場合は，さしあたり必要ない部分を一時的に取り去り，単純化して動作を確認してから，少しずつ処理を戻していくという方法もあります．

どちらの場合も，部品（関数や特定部分の処理）が確実に動くようなものを先に作り，それを組み立てていくという，ボトムアップ手法です．

先に述べた，「ほかのコンパイラでロジックを確認する」方法にも，同様の意味があります．

8-4　WIZ-Cデバッガ/シミュレータの使い方

● ブレーク・ポイントの設定

図8-1にブレーク・ポイントを設定したときのエディタ・ウィンドウの表示例を示します．ブレーク・ポイントを設定するには，行の左端をクリックします．ブレーク・ポイントが正しく設定されると，図のように青のマークが表示されます．この状態で，さらにマーク上をクリックすると，マークがグレーに変わります．これは，ブレーク・ポイントが無効になっている状態です．さらにマーク上をクリックするとマークが消えます．これでブレーク・ポイントは削除されます．

ブレーク・ポイントが設定できるのは，実行行として有効な行のみです．宣言文などでは設定できないことがあります．また，コメント行や空白行，一つの実行文が複数行に分かれている箇所など，マシン・コードが割り当てられていない行にはブレーク・ポイントは設定できません．なお，ブレーク・ポイント

で止まった時点ではまだその行は実行されていません．

図8-1の画面左端にある数字は，行ごとの実行回数を表しています．その数字のある行が実行されると，そこを通るたびにこの数が＋1されます．また，**図8-2**，**図8-3**のようにエディタ・ウィンドウ左上の表示切り替えボタンで切り替えることにより，実行時間かマシン・コードのアドレスに表示を切り替えることもできます．これらの行情報の表示は，ブレークしたときに更新されます．

図8-4のように，ブレーク・ポイントはデバッガ・ウィンドウに一覧として表示させることもできます．ここでもブレーク・ポイントの有効/無効を切り替えたり，ブレーク・ポイントそのものを削除することができます．

この一覧上で，チェックがついているものが有効，チェックが外れているのが無効です．ブレーク・ポイントを選択してキーボードからDeleteキーを押すと削除することができます．

● 変数/レジスタのウォッチ登録，参照

AUTO変数を除く変数や，レジスタの内容を表示させたり，値を変更することができます．WIZ-Cのメイン・ウィンドウ左側には，**図8-5**に示すようなデバッガ・ウィンドウがあります．このウィンドウの“Watch”ページ（ウォッチ・ペイン）には，レジスタ名/変数名とその値の一覧が表示されています．ここに任意のレジスタ，変数を登録して変数の値を参照したり，変更することができます．ただし，AUTO変数はここに登録することはできません．

このウォッチ上の値は，ブレークしたときに表示が更新されるほか，シミュレーション実行中も一定周

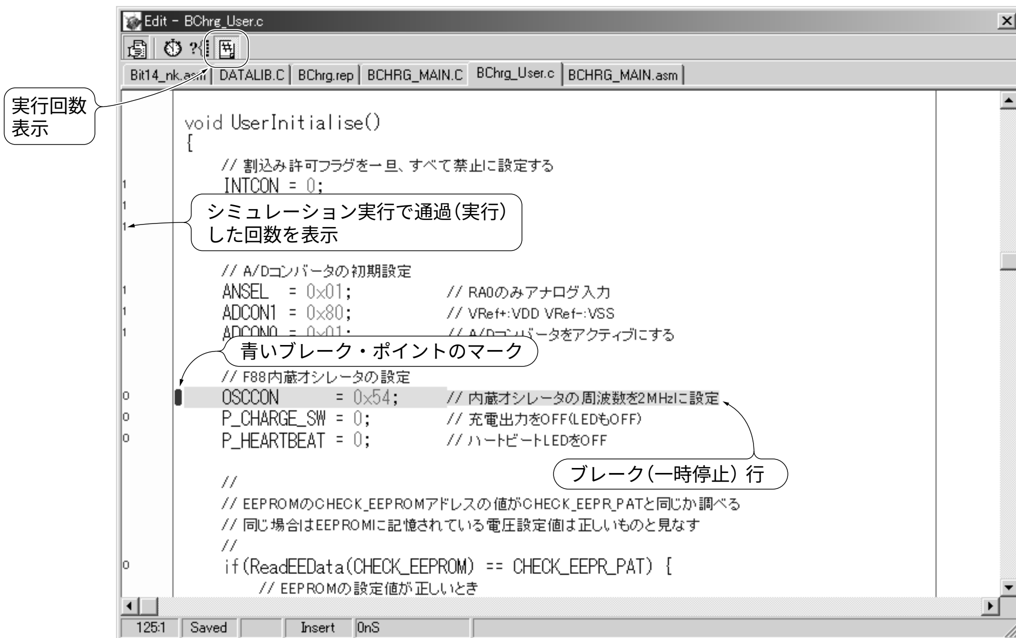


図8-1　エディタ・ウィンドウ（実行回数表示）
C言語のソース・ファイル上でブレーク・ポイントを設定することができる．シミュレーション実行してブレークした位置が水色のラインで表示される．画面左端の数字は，その行が実行された回数を表している．ブレーク位置はまだ実行されていないので，実行回数はゼロ．

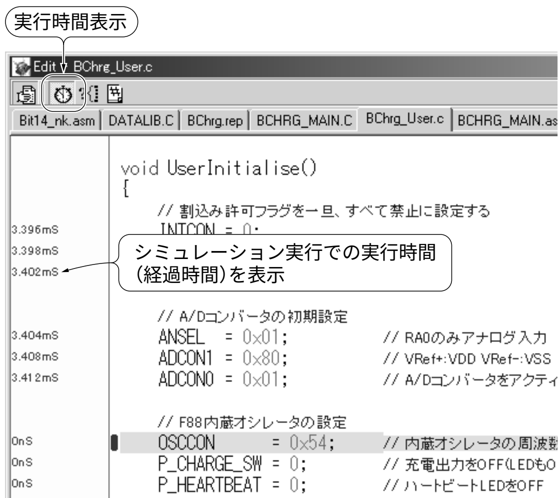


図8-2　エディタ・ウィンドウ(実行時間)
シミュレーション実行が始まってから(リセット直後から)，その行までの実行時間(経過時間)が表示される．二つの実行行の時間差を計算することで，行ごとの実行時間を知ることもできる．

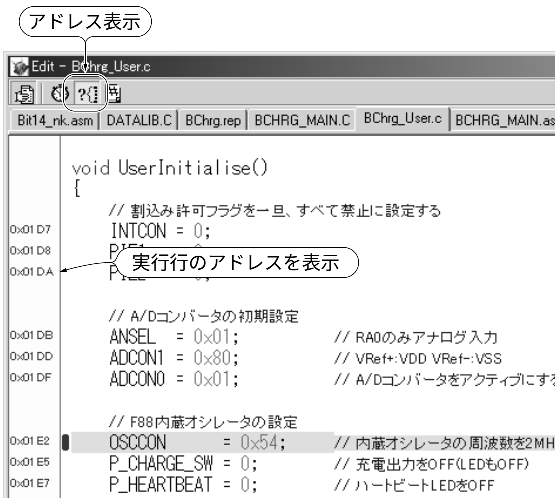


図8-3　エディタ・ウィンドウ(アドレス表示)
実行行のアドレスを表示している．バイナリ・コード(アセンブラ・リスト)を見るときに有用．

期で更新されます．この更新の周期を設定するのが，メイン・ウィンドウ左上の“Update Rate”スライダです．

＜ウォッチ登録手順1＞

ウォッチ・ペインに変数やレジスタを登録するには，ウォッチ・ペイン上でマウスを右クリックしてポップアップ・メニューを表示させ，[Add Watch]をクリックして“Debug Watch”ウィンドウを表示させます．そこで“Expression”入力欄にレジスタか変数を設定します．レジスタ名は[Browse]ボタンをクリックすると一覧が表示されるので，そこから選択することもできます．レジスタ名を直接入力してもかまいません．変数名は直接入力します．実在しない(または設定できない)名前を指定しても[OK]ボタンが有効になりません．

次に“Memory”項で表示形式(変数型)などを設定します．

まとめると，次のような手順になります．

- 変数一覧(ウォッチ・ペイン)の上で右クリック
- ポップアップ・メニューで[Add Watch]をクリック
- Debug Watchウィンドウでレジスタ名か変数名を“Expression”欄に設定
- 同ウィンドウの“Memory”項で表示形式(変数型)などを設定

＜ウォッチ設定手順2＞

ソース・ファイルの変数名またはレジスタ名の上でマウスを右クリックしてポップアップ・メニューを表示させ，そこから[Set Watch on Item]をクリックして追加することもできます．この場合は，続けてデバッガ・ウィンドウのウォッチ・ペイン上で，登録した変数(レジスタ)名の上でマウスを右クリックしてポップアップ・メニューを表示させます．[Modify Watch]をクリックして“Debug Watch”ウィンドウを開き，“Memory”項を設定します．この方法は変数/レジスタ名の入力が省けます．

まとめると，次のようになります．