

実際に組み込んで使える

開発に役立つタイニ・デバッグ・モニタ PIC16F84A & 877 版の作成

鶴見 恵一

PICのプログラム・メモリに、レジスタ・ファイルとEEPROMの読み書き、さらに任意のアドレスからのサブルーチン実行などをコマンドとして書き込んでおき、パソコンのターミナル・ソフトで操作してデバッグを行います。

マイコンのデバッグ・ツールとしてはかなり古い手法ですが、AVR^①で試みて結構重宝しているのでPIC版も作成して使っています。対象となるデバイスはPIC16F84Aですが、本書のためにPIC16F877版も用意しました。

パソコンとの接続はRS-232Cです。ターミナル・ソフトは何でもかまいません、Windows付属のハイパーターミナルでも十分です。

このアイコンは、章末に用語解説があります

5-1 デバッグの方法とツール

ずいぶん数多くのプログラムをこれまでに書いてきましたが、デバッグなしに一発で動いたのはたいへんまれでした。プログラミングには必ずデバッグ作業が伴うものと思ひましょう。筆者が仕事でソフト開発を請け負う場合も「デバッグ：**時間」と、堂々と見積りに提示します。

最初に、デバッグにはどのような方法があるのでしょうか。

● 机上デバッグ

うまく動かなかったらプログラム・リストを眺めて不具合を探す。見つけたら再試行。これを繰り返す。何のツールも不要なので安上がりですが、効率は最悪です。プログラムを書くときにはこれで良いと思ひ込んでいるわけですから、間違いを自分で発見するのは至難の業です。

● シミュレータ

マイクロチップ・テクノロジー社が無料で公開している開発ツール(MPLAB)にはシミュレータが付属しています。無料ですからこれも安上がりです。

演算ルーチンのように内部レジスタだけの処理ならば便利で有効なツールなのですが、センサ入力やアナログ駆動^②が主なプログラムでは、これらをシミュレートできるわけではなないのでかなり厄介です。

● イン・サーキット・エミュレータ (ICE)

実機のPICデバイスの代わりに、これをエミュレートするツールをつなぎ込んでデバッグします。PICデバイスに接続された周辺装置を実際に動かしながらプログラムのテストができて、機能と使い勝手はシミュレータと同等もしくは以上ですから、最も優れたツールといえます。

難点は少々お金がかかることです。

● タイニ・デバッグ・モニタ

お金をかけずにソフトウェアの工夫だけでシミュレータの欠点を可能な範囲で補い、ICEのもつ機能を適度に妥協してなんとか実用的なツールとしたのがここで紹介するタイニ・デバッグ・モニタです。

5-2 使い方の概要

Z80や6809のような従来のプロセッサでしたら、デバッグ・モニタをROMに常駐させ、アセンブル済みのファイル(HEX形式のファイルなど)をモニタのもつロード機能でRAMに読み込んでデバッグ開始が普通でしたが、プログラム・メモリがフラッシュだけのPIC16F84Aではこれできません。そこで、タイニ・デバッグ・モニタのソース・ファイルをデバッグ対象のソース・ファイルと一緒にアセンブルしてフラッシュ・メモリに書き込み、モニタの機能であるレジスタ・ファイルの読み書きや任意のアドレスからの実行を試みてデバッグを進めます。

タイニ・デバッグ・モニタのソース・ファイルにデバッグ対象のソースを書き加えてもかまいませんが、`#include`命令でデバッグ対象のソースにモニタのソースをインクルードさせたほうが便利と思います。

5-3 プログラムの仕様

組み込まれているコマンドを表5-1に示します。以下はコマンドの詳細です。大文字/小文字のどちらも有効です。

● レジスタ・ファイルの読み書き

コマンド‘M’に続けて16進四桁のアドレスと‘Enter’をタイプします。入力されたアドレスとその内容が表示されます。機能は表5-1を参照してください。

ページ0およびページ1のレジスタ・ファイルもこのコマンドだけで読み書きできます。

例：M0081でOPTIONレジスタを読み書きできます。

● EEPROMデータメモリの読み書き

コマンド‘E’に続けて16進四桁のアドレスと‘Enter’をタイプします。利用方法は‘M’コマンドと同様です。アセンブル・リストに示されるデータ・メモリの先頭アドレスは、ライタの都合で\$2100ですが、チップ内部では先頭が\$0000なので注意してください。このコマンドの‘E’に続くアドレスの上位バイトは常に00になります。

コマンド	パラメータ	機能
M	www	レジスタ・ファイルの読み書き。 ‘Enter’で次のアドレス ‘/’で前のアドレス 変更はスペースに続いて二桁の16進数 終了は‘Enter’,‘/’,スペース以外の文字
E	www	EEPROMの読み書き。 ‘Enter’で次のアドレス ‘/’で前のアドレス 変更はスペースに続いて二桁の16進数 終了は‘Enter’,‘/’,スペース以外の文字
G	www	サブルーチンの実行
D	www bb	レジスタ・ファイルのダンプ

wwwは16進四桁のアドレス値
bbは16進二桁、バイト数

表5-1 タイニ・デバッグ・モニタのコマンドデバッグに最低必要と思われるコマンドを用意した。実行前のレジスタ・ファイルの設定が必要であれば‘M’コマンドで設定し、サブルーチン実行後のレジスタの内容は‘M’コマンドまたは‘D’コマンドで読み出す。大文字/小文字、どちらでも可。

● サブルーチンの実行

コマンド‘G’に続けて16進四桁のアドレスと‘Enter’をタイプします。レジスタ退避に割り当てたレジスタ・ファイルからFSRとSTATUSレジスタに値をロードして入力されたアドレスから実行し、サブルーチンの終了(return命令)でW, STATUS, FSRの各レジスタを元のレジスタ・ファイルに退避してコマンド待ちに戻ります。

リスト5-1にアセンブル・リストから退避領域の定義を抜き出しました。

(注) wtempにはWレジスタの実行後の結果が残されますが、実行開始時のWレジスタの値は不定です。STATUSレジスタには実行開始時にstatustempからロードされますが、その際にZフラグだけは影響を受けてしまうので注意してください。

● レジスタ・ファイルのダンプ

コマンド‘D’に続けて16進四桁のアドレスとデリミタ‘,’さらに2桁のバイト数をタイプします。デリミタは\$2f以下のコードなら何でも有効です。

リスト5-1 レジスタ・ファイルの退避領域

実行結果の確認で重要なのはWレジスタとStatusレジスタ。‘G’コマンドで実行後、必要ならば‘M’コマンドでこのアドレスを参照する。

このアドレスで都合が悪ければ、ソース・ファイルで‘org 0x0c’の行を書き換えて別のアドレスに配置できる。

```

00031 ;レジスタ・ファイル:ワークエリア
00032                org    0x0c
00033 ; レジスタ退避
00034 wtemp         res    1        ;Wレジスタ退避
00035 statustemp    res    1        ;Statusレジスタ退避
00036 pclathtemp    res    1
00037 fsrtemp       res    1        ;FSRレジスタ退避

```

網掛けした部分がアドレス

5-4 プログラムの構成と動作 (ソースは秋月版とマイクロチップ社純正版)

本モニタは(秋月電子通商)のキットに付属されているアセンブラ“PA.EXE”を使って作りましたが、マイクロチップ社が提供するアセンブラとは違いがあるのでマイクロチップ社版も用意しました。両者を見比べるとソース文記述方法の違いが容易に比較できると思います。

PIC16F84AはUARTを内蔵していないので、RS-232C通信にはソフトウェアUARTを使います。この部分は前述のキットに付属のサンプル・プログラム(RS232C.ASM)を引用させていただきました。送信データはRA₀、受信データはRA₁に割り付けました。

PIC16F877版での違いは、この章の最後で説明します。

● 秋月版

ソース・ファイルの本体は“AkiMonV2.asm”です。特殊機能レジスタやビットの定義ファイルはPA.EXEに付属の“16F84.H”を使い、これを以下の行のようにインクルード(include)しています。

```
include 16f84.h
```

● 純正版

ソース・ファイルの本体は“PicMonV2.asm”です。特殊機能レジスタやビットの定義ファイルはマイクロチップ社提供のツールに付属している“p16F84A.inc”を使い、これをインクルードします。

アセンブラは両者とも基本命令以外にマクロ拡張された命令をもっていますが、秋月版で利用した拡張命令の一部をここでも使用するため、マクロ・ファイル“MonMacro.inc”を用意しました。これも以下のようにインクルードします。

```
#include <p16F84A.inc>
#include <MonMacro.inc>
```

リスト5-2 ボー・レートの設定

違うクロック周波数を使用する場合は囲み線で示された計算式でbtimeの値を求め、equの後ろの数値を書き直す。ボー・レートも9600にこだわる必要はないが、第3章と第4章の事例2では9600bpsを前提に話を進めているので、最初はそのままが無難と思う。

```
; RS232C送受信
;
; 送信・受信フォーマット:
;     9600bps, 8ビット, 1ストップ・ビット
;     パリティなし, フロー制御なし

; btime = {(動作周波数(Hz) ÷ 転送スピード(bps) ÷ 4) - 10} ÷ 3

LIST p=16F84A
#include <p16F84A.inc>
__config _HS_OSC & _WDT_OFF & _PWRTE_OFF;Set oscillator to HS,

#include <MonMacro.inc>

; btime equ .33 ; 9600bps @4.19MHz
btime equ .66 ; 9600bps @8MHz.
```

● 共通の注意：ボー・レート

RS-232Cのボー・レートはクロック・オシレータの周波数に依存するので、使用する水晶またはセラミック振動子の周波数に合わせてソース文の変更が必要です。リスト5-2はソースの冒頭部分です。

変更を要するのは網掛けで示した**btime**の値だけです。ボー・レートが9600bpsでオシレータが4.19MHzと8MHzでの二つの例があり、4.19MHzの例をコメント・アウトしています。これ以外の周波数で使用する場合は囲み線で示された計算式で**btime**の値を求め、**equ**の後ろの数値を書き直してください。

● 動作

リセット・スタートするとRS-232Cで使用するポートを初期化してモニタ・コマンド待ちのループ(**Mprompt**)を実行します。リスト5-3がこのループの部分です。メインテナンスの用意として、実行プログラムを中断または終了してデバッグ・モニタに移る場合は**Mprompt**のアドレスに**goto**してください。

'G'コマンドはレジスタ・ファイルのロードと退避が伴い、少々工夫を要する部分なので、この部分をリスト5-4に抜き出しました。

網掛けをした行に注目してください。実行アドレスの入力とFSRレジスタを復帰した後で、実行アドレスにジャンプするルーチンに移るのですが、この時に**call**命令を使用しています。つまり'G'コマンドで指示したサブルーチンからリターンした後は網掛けをした行の次から実行し、サブルーチン終了時のレジスタの内容が確認できるように、**W**, **STATUS**, **FSR**の各レジスタを退避アドレスに書き込んでコマンド待ちになります。

リスト5-3 モニタ・コマンド待ちの部分

文字入力の後でサブルーチン'toupper'をコールして大文字に変換するので、小文字大文字どちらも有効。


```
;モニタ・コマンド待ち
Mprompt    call    sendscr        ;CR
             movlw   '*'
             movwf  ch
             call   transmit
             call   insecho
             call   toupper
             movlw  'M'
             subwf  ch,0
             jnz   Mnlpm1
             goto  MnMcmd        ;'M'ならばレジスタ・ファイルの読み書き

Mnlpm1      movlw  'E'
             subwf  ch,0
             jnz   Mnlpm2
             goto  MnEcmd        ;'E'ならばEEPROMの読み書き

Mnlpm2      movlw  'G'
             subwf  ch,0
             jnz   Mnlpm3
             goto  MnGcmd        ;'G'ならばサブルーチンを実行

Mnlpm3      movlw  'D'
             subwf  ch,0
             jnz   Mprompt
             goto  MnDcmd        ;'D'ならばレジスタ・ダンプ
```


● 制約と妥協

プログラム・コードはフラッシュROMに書き込むため、ソフトウェアによる手段ではブレイク・ポイント  を設定できません。このため、サブルーチン単位で実行させて、実行結果と前後のレジスタ・ファイルの内容を確認することで妥協しています。

プログラム・サイズにも制約があり、限られたメモリ空間をデバッグ・モニタができる限り占有しないように必要最小限の機能とし、デバッグ終了後もメンテナンスのためにそのまま残しておいても邪魔にならない程度とするように努めました。

‘G’ コマンドでの実行開始時にWレジスタが不定なのも不満のある所です。拡張命令の一部ではマクロ内部でWレジスタを使用するために、サブルーチンへのパラメータの受け渡しにWレジスタを使用するのは適当でないこともあり、妥協していますが筆者の知識不足かと思います。

5-5 プログラムの作成とデバッグ

汎用レジスタは13バイトがワーク・エリア  としてデバッグ・モニタで使用されています。この定義の最後に“WorkRamEnd”がラベルとして付けられているので、使用する汎用レジスタの先頭アドレスは以下のように定義してください。

```
org    WorkRamEnd
```

リスト5-4 Gコマンド実行の部分

指定されたアドレスへコンピュートド goto を実行し、リターン時のレジスタの状態を実行結果の確認のために保存する。

```
; コマンド'G' サブルーチンを実行
MnGcmd    call    get4hex        ; 16進四桁を入力, hexH, hexLに保管
           call    insecho
           movlw   0dh           ; 'Enter'が打たれたら実行
           subwf  ch, 0
           jnz    Mprompt
MnG01     clrfs  STATUS         ; Bank 0を選択
           movf   fsrtemp, 0     ; FSRレジスタを復帰
           movwf  FSR
           call   gotarget       ; 次のアドレスをpush, ターゲット・アドレスへジャンプ
;
; サブルーチンからの復帰で次行からが実行される
           movwf  wtemp         ; Wレジスタ保存
           movf   STATUS, 0      ; Statusレジスタをリード
           clrfs  STATUS         ; Bank 0を選択
           movwf  statustemp     ; Statusレジスタを保存
           movf   FSR, 0        ; FSRレジスタをリード
           movwf  fsrtemp       ; FSRレジスタを保存
           goto   Mprompt       ; モニタ・コマンド待ち
; ターゲット・アドレスへジャンプ
gotarget
           movf   hexH, 0        ; PCLATHにアドレス上位バイトをセット
           movwf  PCLATH
           movf   statustemp, 0  ; Statusレジスタ復帰, Zは影響受ける
           movwf  STATUS
           movf   hexL, 0        ; ターゲット・アドレス下位バイトをWレジスタへ
           movwf  PCL            ; Wレジスタをpclに移動してターゲット・ジャンプ
```

ソース・ファイルを作成したらその冒頭でデバッグ・モニタのソースをインクルードします。秋月版の“PA.EXE”を使用する場合は“AkiMonV2.asm”，マイクロチップ社純正版では“PicMonV2.asm”です。“PicMonV2.asm”ではマクロ定義の“MonMacro.inc”もパスが有効なフォルダ(ソースと同じフォルダが確実)に格納するのも忘れないでください。

これをアセンブルし、チップに書き込み、RS-232C接続とパソコンのターミナル・ソフトが正常であればPICの電源投入後にプロンプトのアスタリスク‘*’がターミナル画面に表示されるはずで、これでデバッグ・モニタのコマンドを使ってデバッグ開始です。

● ハイパーターミナルの設定例

Windows付属のハイパーターミナルを使用する場合の設定例を以下に示しておきます。

最初に起動すると「接続の設定」ダイアログが開きます。表示されない場合は左上のメニュー「ファイル」から「新しい接続」を選びます。名前を入力と好みのアイコンを選択してOKをクリックします。

次の「電話番号の情報」を入力する画面に変わりますが、番号は空欄でかまいません。「接続方法」で使用するシリアル・ポートのナンバ(COM1, COM2など)を選んでOKをクリックします。

次の「ポートの設定」ではビット/秒を使用するボー・レート、データ・ビットを8、パリティなし、ストップ・ビット1、フロー制御なしを選んでOKをクリックします。

これで使用可能になりますが、後のためにこの設定を保存しましょう。左上のメニュー「ファイル」から「名前を付けて保存」を選び、後は表示に従います。

● 実行例

デバッグ・モニタの最後の部分に簡単なテスト・プログラムを用意してあります。

リスト5-5にアセンブル・リストからその部分を抜き出しました。このリストはマイクロチップ社純正版の“PicMonV2.asm”をアセンブルしたものです。秋月版ではアドレスが異なっているので注意してください。

デバッグ対象のソースがなければデバッグ・モニタのソースだけでかまいません。アセンブルと書き込みを行って実行してみましょう。

リスト5-5 デバッグ・モニタの最後に組み込まれたテスト用のプログラム例

デバッグ・モニタの実行確認用。メモリの節約のために、確認がすんだらこの部分はソースから削除してかまわない。

```
00438 ; テスト・プログラム
0146 300A 00439 Test1 movlw .10
0147 0097 00440        movwf bcnt
0148 300A 00441 Test2 movlw .10
0149 0797 00442        addwf bcnt,F
014A 2113 00443        call  sendscr
014B 0830 00444        movf  0x30,W ;SRAM 30HのASCIIコードをTX
014C 0090 00445        movwf  ch
014D 200E 00446 tstltp call  transmit
00447 djnz  bcnt,tstltp
014E 0B97 M          decfsz bcnt,f
014F 294D M          goto  tstltp
0150 0831 00448        movf  0x31,W ;SRAM 31HのデータをWにもってリターン
0151 0008 00449        return
```

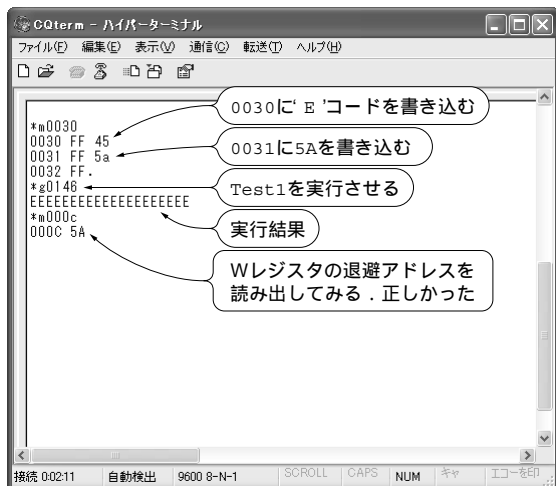


図5-1 パソコンのターミナル・ソフトを使ってデバッグ・モニタのコマンドを実行した様子

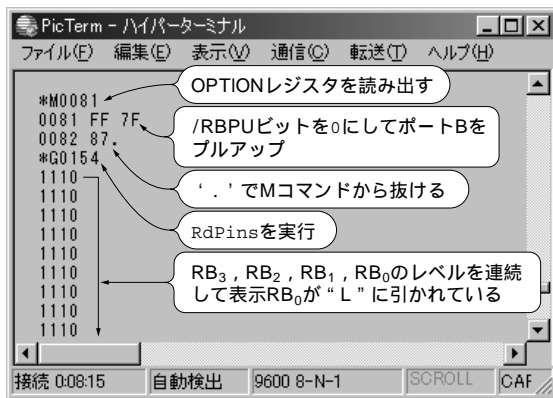


図5-2 別なテスト・プログラム PortTest.asm を実行した様子。

Test1を実行するとレジスタ・ファイル\$0030のASCIIコードを20文字送信し、\$0031の内容をWレジスタに格納してリターンします。図5-1が実行の様子です。

最後に読み出している000cはWレジスタの退避アドレスです。プログラム・リストではラベルとしてwtempが付されています。

● 別なテスト・プログラム

テスト・プログラムをもう一つ用意しました。リスト5-6がソース・リスト、ファイル名は“PortTest.asm”です。ポートの入力レベルを読み出し、‘1’か‘0’の文字で連続して吐き出します。メカトロの工事現場ではこのようなプログラムをその場で作成し、センサやスイッチの確認をすることがしばしばあります。

RB₃, RB₂, RB₁, RB₀の状態を1行として表示しますが、別なピンの確認が必要でしたらポート名とピン名を書き換えてください。

このプログラムはデバッグ・モニタの“PicMonV2.asm”をインクルードします。PicMonV2.asmの最後の行‘end ;’を削除するかコメント・アウトしてください。これを忘れるとモニタ・ソースの‘end’でアセンブルが終了してテスト・プログラムがアセンブルされません。

マイクロチップ社のアセンブラでアセンブルし、実行してみましょう。図5-2が実行の様子です。

最初に‘M’コマンドでポートBのプルアップを有効にしています。OPTIONレジスタはページ1にマッピングされていますが、‘M’コマンドはFSRレジスタを使った間接アドレス指定で読み書きするので、図5-2のようにバンク切り替えなしで読み書きできます(FSRのビット7がページ・セレクトのRP0に割り当てられている)。

‘G’コマンドで実行するのはプログラム中のラベル‘RdPins’からです。アセンブル・リストからこのラベル名を検索してアドレスを調べます。0154でした。

終了するにはEnterキーを何度も叩くか押し続けてください。終了条件はRS-232Cの入力ピンをサンプリングしているだけなので、タイミングによって読みこぼしがあるからです。