AVR/hidspx_tips

======= hidspxに関連するTips ======= †

AVR/HIDaspxのページが長大になり、閲覧に時間がかかるようになってきました。
そのため、Tipsの追加も難しくなってきたので、別ページに分離しました。
今後は、各種のノウハウをこのページを充実させていきます。

なおココに記したものは、HIDaspxのハードを前提にするものはではなく、ライタ指定オプション(-phXX)の変更により、COM-SPIブリッジなどのhidspxがサポートするライタでも同様に使えます。HIDaspx以外の利用者もご活用ください。

HIDaspx用の制御コマンドhidspxは、Windows(2000/XP/Vista), Linux, MacOS Xで利用でき、共通の操作が可能です。

↑

hidspxコマンドの基本的な使い方 †

hidspx-****.zipアーカイブに含まれるReadme-j.txtをお読みください。
また、hidspx.exeを配置したディレクトリに設定ファイルhidspx.iniを置き、適切なオプションを設定してください。hidspxコマンドはCMDプロンプト上で利用しますが、hidspx-GUIやhidspxGを使って、GUI操作も可能です。

HIDapsxをhidspxで使用する場合のコマンド例

1	ターゲットのAVRマイコンのデバイス情報を確認する	hidspx -ph -d4 -r
2	ターゲットのAVRマイコンのFuse情報を確認する	hidspx -ph -d4 -rf
3	main.hexを書き込む	hidspx -ph -d4 main.hex
4	main.hexを照合する	hidspx -ph -d4 -v main.hex
5	AVRマイコンのプログラムを読み出し、HEXファイルに書く	hidspx -ph -d4 -rp >rom.hex
5'	AVRマイコンのプログラムを読み出し、HEXファイルに書く	hidspx -ph -d4 -rp -orom.hex

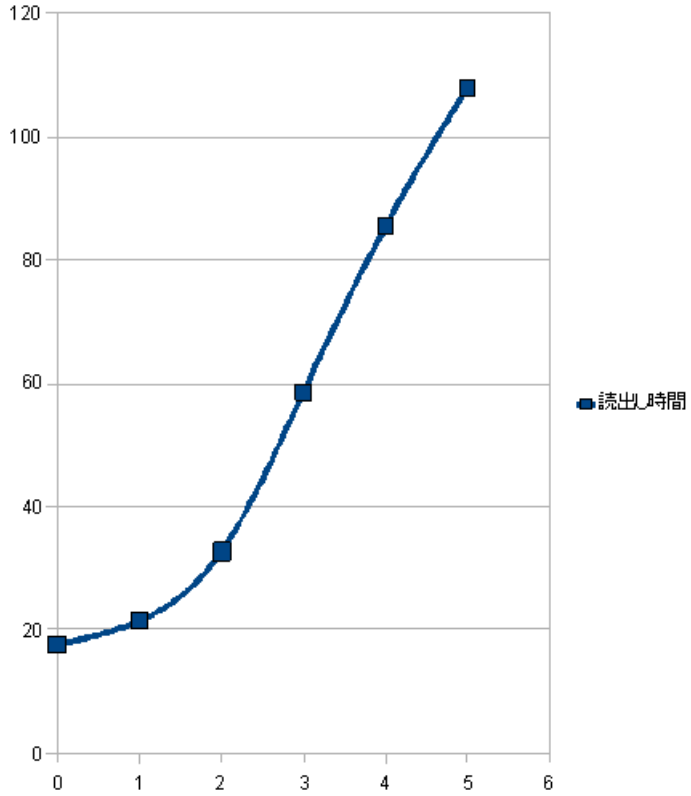
6　書き込まれているプログラムのダンプリスト表示　　hidspx -ph -d4 -rph

「-ph -d1」のようにオプション指定することで、より高速な処理が可能になります。
「-ph -d4」のように、毎回指定が必要なオプション類は、hidspx.iniに設定しておくことで、入力を簡略化できます。その他にも多数の機能がありますので、コマンドの使い方の詳細は、付属の説明書(avrx-tool.txt)をご覧ください。

↑

「-dオプション」と処理時間の関係 †

HIDaspx(AVRライタ全般に共通)では、-dの値によって大きく処理時間が変化します。
そこで、delay値と読み出し時間をグラフ(ディレイ値と時間[秒])を作成しました。
縦軸は、NECのチップによる増設ボードのUSBポートを使い、ATmega128の全メモリの読み出しに要する時間を表しています。



このグラフから、-d0と-d5では5倍もの違いがあることがわかります。
値が小さいほど高速に処理できますが、値が大きいほど認識できる可能性は高まります。
そこで、hidspx.iniには工場出荷時の値でエラー無く処理できるという理由から、「-d4」を設定しています。書き込みと照合に必要な時間は、この時間の約2倍と考えてください。

この特性を理解し、書き込み対象のマイコン速度に合った値を指定し、効率的な利用を行ってください。

指定の例(ATtiny2313の場合)

No	FUSE Low	-dの値	発振周波数	備考
0	----	-d0	18MHz以上	外部クリスタル/セラミック発振子
1	-fL0xe4	-d1	8MHz	14CK+65ms
2	-fL0xe2	-d2	4MHz	14CK+65ms
3	-fL0x64	-d4	1MHz	工場出荷値
4	-fL0x62	-d5	500kHz	14CK+65ms
5	-fL0xe6	-d17	128kHz	14CK+65ms
6	-fL0x66	-d120	16kHz	118, 119では不安定

↑

senshu版、hidspx用GUIフロントエンド †

2009年初めから取組んできた、使い勝手向上の取り組み成果として、ゆきの研究室さん公開のソースを元にした（JA1WXYさんとは別の）GUIフロントエンドをsenshuが作成しました。2009年4月6日から、hidspxパッケージに同梱しています。

これはC#で記述しており、ボタンをマウスでクリックですることでhidspxの主要な機能（Read/Write/Verify/FUSE操作）を利用できます。

またテキストウインドウにはコマンドの実行の様子が表示され、各種コマンドを指定して実行することも可能です。

Webブラウザで各種の開発に役立つページを適切に開く機能も備えています。

動作画面は、次の項をご覧ください。

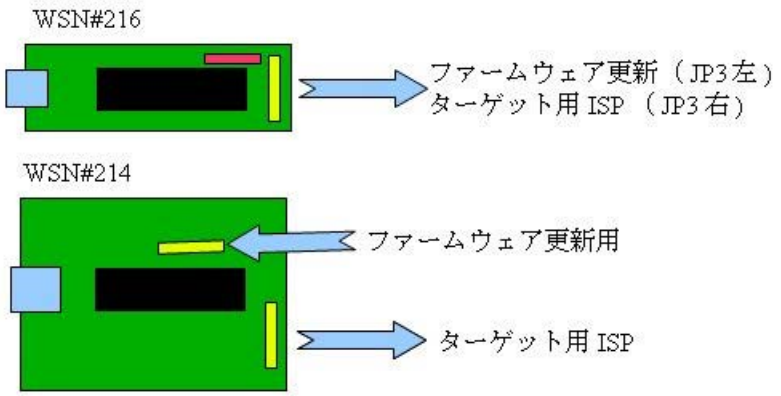
↑

WSN214, 216基板の使い方 †

WSN214 http://wsnak.com/kit/214/index.htm や
WSN216 http://wsnak.com/kit/216/index.htm の利用法を簡単に説明します。

これらは、どちらもHIDaspx（AVRライタやUSB I/O）としての利用できます。
一般的なAVRマイコン用のライターは2列x3ピン＝6PのISPコネクタを採用していますが、2x3＝6Pコネクタは入手性が悪く、またブレッドボードでの利用にも適していません。

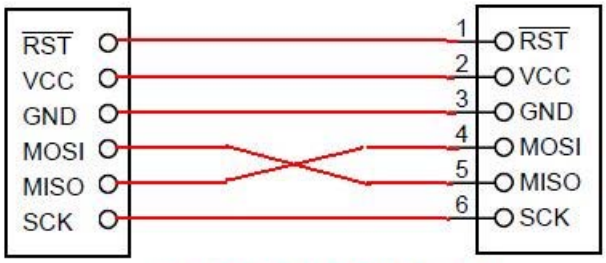
そこで、逆接続時を行っても壊れない1x6pピン配置の6Pコネクタを採用し、専用基板化を行っています。WSNさんで扱っているAVRマイコンボードは、全てこのピン配置を採用している為、ストレートの6Pコネクタで接続すればISP方式で書き込みできます。



WSN214基板はAVRライタとしての利用を主眼に置き、コネクタも兼用ではなく、用途を特定しています。2つの6Pコネクタは、ISP用と内部ファームウェア更新用です。

一方、WSN216は小型化のためファームウェア更新コネクタを実装できない為、ISPコネクタを使ってアップデートを行います。内蔵のファームウェア更新時は、JP3プラグを切り替え、MISO,MOSI をクロス接続したケーブルで接続して書き込んでください。

ISP用の「ストレートケーブル」



ファームウェア更新用のクロスケーブル

この作業が面倒なら、書き込み済みのAVRマイコンを用意すればよいのです。しかし、
HIDaspxファームウェアの更新はほとんど発生しないので、許容できると思います。

WSN216のJP3をISPモードに切り替えれば、WSN214とWSN216は同一のストレートケーブル
(6Pの1対1のケーブル)で同等の利用 (AVRマイコン用への書き込み) が可能です。

↑

AVRマイコン開発環境との統合(for Windows) †

各種の開発環境から、HIDaspxを利用する事ができます。

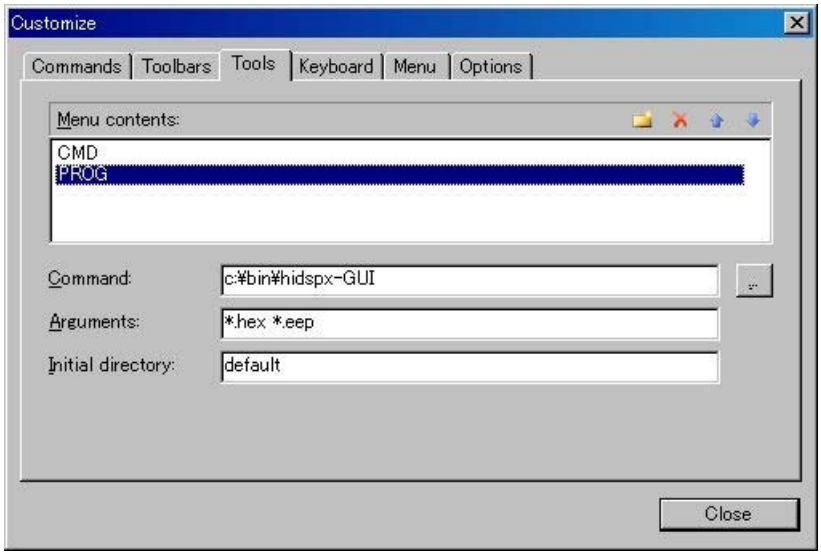
↑

「Atmel社の開発環境(AVR studio)」 †

AVR studioでは、外部ツールを呼び出す仕組みを持っています。

■ AVR studioでの設定方法

「MENUのTools ⇒ Custmize ⇒ Tools」と選択し、以下の内容を登録します。



■ 使い方

プロジェクトでコンパイルを行ない、登録した「PROG」を起動するとファイル名が自動的に
設定され、次に「Wrire(Both)」をクリックし、書き込みます。(名前は任意のものでOK)

hidspx.iniを適切に設定(通常は提供時のままでOK)していれば、AVRマイコンを自動認識する為、これ以外の操作は不要です。

hidspx-GUIを通常利用する場合と同様にFuseデータの確認やChip Eraseなども実行可能です。
また、hidspx-GUIは最新のHEXファイルを必ず読み出し後に書き込むので、対象ファイル名に変更が無い限り、何度でも繰り返し「Write(Both)」で書き込みできます。

多くの方が望んでいた(と思われる)「AVR studioとの一体感のある連携」が可能になりました。

↑

「BASCOM-AVR」でhidspx(または hidspx-GUI)を利用する †

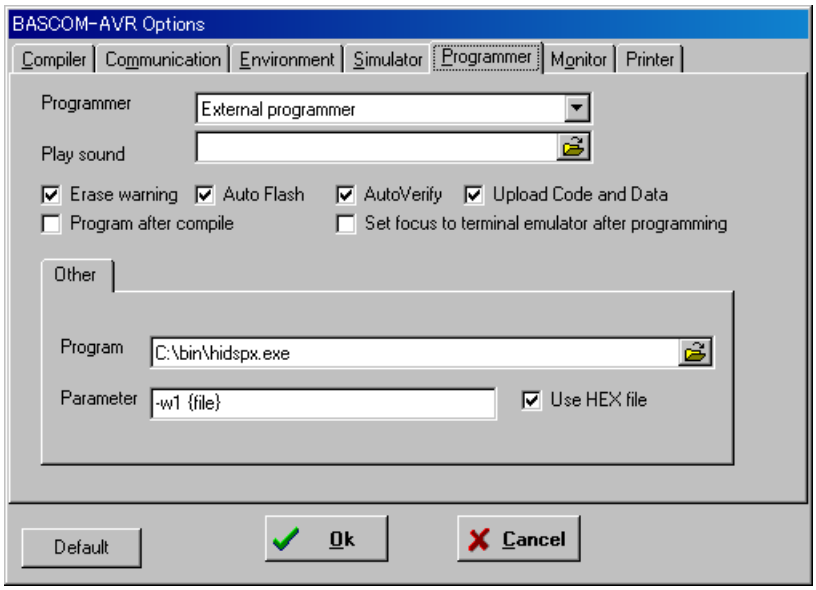
BASCOM-AVRは、一般的なAVR用プログラマを書き込み用プログラマとして設定できます。

BASCOM-AVRのメニューのOptions=>Programmerを選択し、自分が利用しているhidspx.exeのフルパス名を指定します。その後、[F4キー]やメニューの書込みボタンをクリックすれば、hidspxで書き込みできます。「-w1」は完了後も指示のあるまで表示を続けます。2以上の値(例えば-w5)は、書込み完了後にその指定した秒数だけCMD窓を表示します。

-w5で5秒間、書込み結果を確認できます。{file} {EEPROM}は、それぞれ、Flash, EEPROM用のファイルです。{}パラメータ(引数)は空白で区切ります。

また、hidspx-GUI.exeを指定すればマウス操作で書き込みでき、FUSE操作も可能です。ただし、HEXファイル書き込みが目的なら、hidspx.exeを登録するのがワンクリックで書き込みができる便利な方法です。hidspxG.exeでも全く同様の利用が可能です。

hidspx-GUI.exeを使えば、FUSE操作も可能ですが、マウスで[Write]ボタンのクリック操作が必要になり、書き込み手順は増えます。なお、hidspx-GUIでは「-wや -phなどのオプションは無視する」ので、hidspx用の設定は取り除く必要はありません。




```
-w5 {file} {EEPROM}   .... 指定パラメータは、空白文字に注意し入力すること
```

↑

hidspx-GUIが機能しない原因(設定上の注意点) †

kawanaさんらの質問とその対策です。

hidspx-GUIを起動しても、ライタ機能が利用できません。
ログ窓に以下のように表示されます。

```
>hidspx-GUI -h
# version Check ....OK
>hidspx-GUI -!
# Get user bookmark .... NG
>hidspx-GUI -d4 -rl
# Fuse Read ....NG
>hidspx-GUI -rf
# Command execute ....OK
```

```
（デバイス）Read ボタン をクリック
>hidspx-GUI -rf
# Chip status ....OK
と表示され、
デバイス欄は「Not Connect」表示のまま
```

```
>hidspx-GUI -!
#Get user bookmark .... NG
```

hidspx-GUIコマンドが -! の実行でエラーになっていることがわかります。
hidspx関連の不具合が予想されますが、kawanaさんからの報告は意外なものでした。

```
[hidspx.exe File]のBOXに「hidspx-GUI.exe」をセットした
```

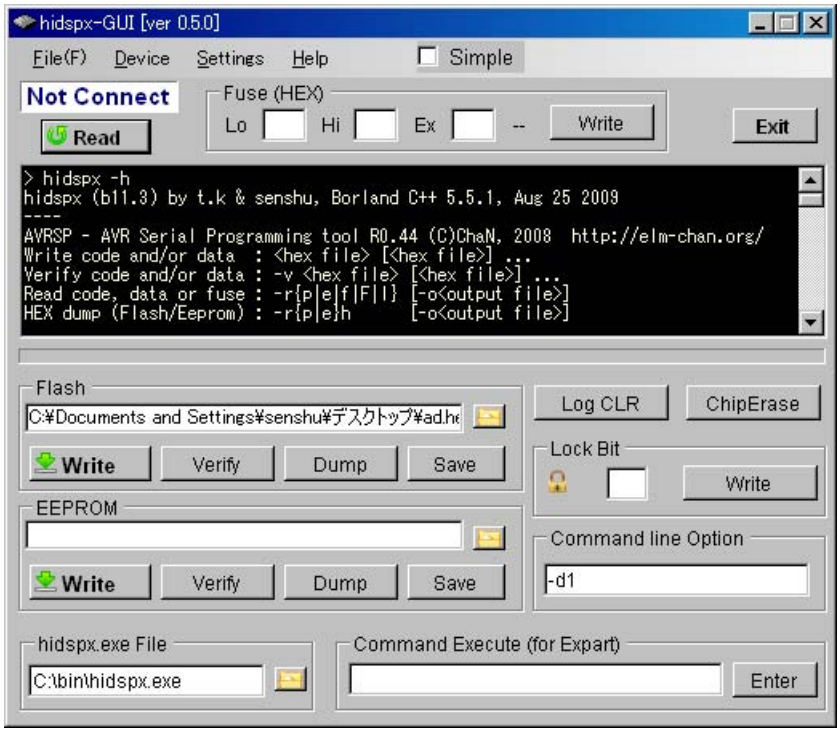
とのことです。

このBOXにはhidspx.exeやhidspx-gcc.exeを指定することを想定しています。利便性を考え、空欄で起動すると、自動でhidspx.exeを設定しますが、任意の値を入力できます。また、hidspx-GUI.exeは複数個の起動は出来ないようなっていますので、今回のようにhidspx-GUI.exeが設定されると、報告のような動作をします（全く機能しません）。

設定ダイアログではhidspx*.exeをリストするため、誤ってhidspx-GUI.exeを選択すると、こうした設定を行う可能性があるようです。

この設定でhidspx-GUIを操作したところ、ログの一部にOKの文字が表示されるため、設定ミスに気づくことが遅れたようです。そこで、2009-0826版では、hidspx-GUI, hidspxG共に、適切なエラーコードを返却する仕様に変更しました。これによって、実行結果は、全てNGを表示します。

正しくは、以下に示す図のように「hidspx.exeのフルパス名」を設定します。



なお、基本的なことですが、hidspx.exeの利用には以下のファイル群が必要です。

```
hidspx.exe, hidspx.ini, libusb0.dll, fuse.txt, hidspx-GUI.exe（GUIを使う場合）
```

これらを漏れなくコピーしてください。c:\binをPATH環境変数に追加しておけば、hidspxを展開した場所にあるbinフォルダ中のsetup.batをクリックするだけで、バージョンアップが可能です。

なお、-phを追加せずにhidspxを実行する理由は、COM-SPIブリッジなどでもこのGUIを利用可能にするためです。hidspx.iniに個人が利用するライタの固有情報を設定すれば（通常はその設定を行って利用します）、GUI側では無設定で、HIDaspx以外のライタも利用可能です。

ただし、一時的な設定を確認するなら、「Command line option」に-phのような指定を加え、hidspx.iniを変更することなくライタを切り替えができます。

もう一つの方法には、hidspx.iniを利用せずに、「Command line option」に-ph -d4を入力することで、問題を解決できます。複数のライタは「Command line option」でライタ指定を行うと便利です。

なお、hidspx.iniにはライタ指定以外にもブックマークも登録されており、添付のhidspx.iniをコピーしておくと有益なページを開く機能が利用できます。

↑

「MikroC PRO 2008 for AVR」でhidspxを利用する †

AVR freaks でmikroE社から、AVRマイコン用のC言語開発環境が2008年12月にリリースされました。

今までは、BASICとPascalだけでしたが、新たにC言語が追加され、ユーザーインターフェースもスマートなスタイルに一新しました。安定性その他は不明ですが、長らく待たれていたC言語用の開発環境です。

今まで使ってきたPICマイコン用と類似の関数を持ち、使い方もほぼ同様です。これが安定して利用できると、かなり便利になります。

WinAVRも十分便利なのですが、Atmel純正のシミュレータを利用するには、WinAVRとAVR studioを合計すると200MBを超えるdisk領域が必要ですが、この開発環境は32MBほどのサイズで利用でき、手持ちのAVRライタを開発環境に組み込むことも可能（HIDaspxライタもOK）です。

また、ANSI-C準拠度が高く、C言語を知っている方なら新たに学習する部分は極わずかです。

DEMO版でも、2kW(4kB)までのコードを作成できます。この程度のサイズまで利用できれば、かなり複雑なレベルまで利用できます。（ATtiny2313やATmega48なら制限なしです。）

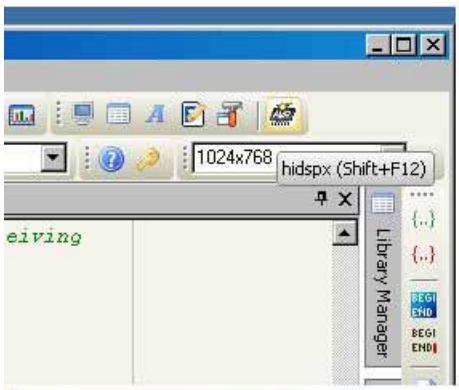
ただし、ツール自体のUIは非常に凝りすぎていてBUGが心配です。

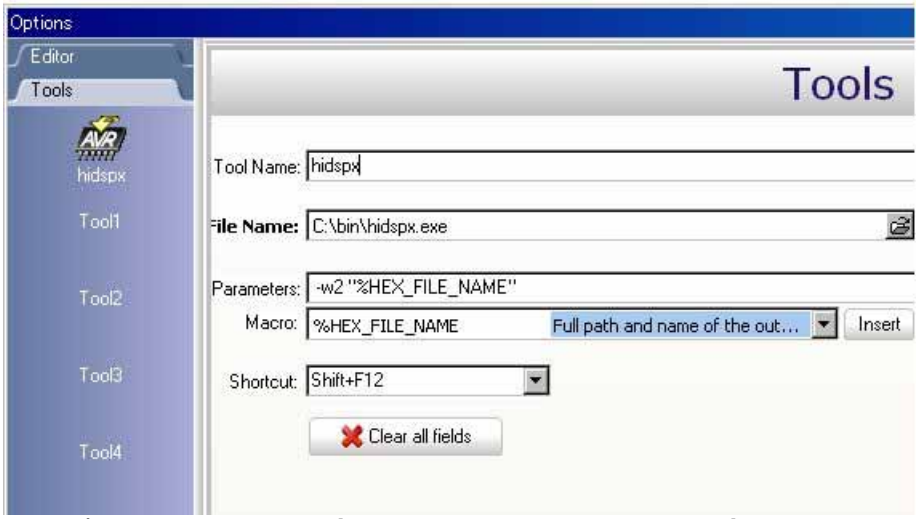
http://www.mikroe.com/en/compilers/mikroc/avr/

また、AVR-GCCと異なり、const宣言したものはプログラムメモリ(Flash領域)に割り付けます。

実機が無くとも、内蔵のシミュレータでDEBUGが可能で、実行時間も詳細にチェックできます。
また、豊富なライブラリが付属するので、各種のI/Oを短いコードで利用可能です。

以下のように外部ツールを設定すると、アイコンクリック（あるいはShif+F12）でHIDaspxを使って、AVRマイコンに書き込みが可能になります。ワンクリックとはいえませんが、ワンキーで便利に書き込みができます。





設定のポイントは、parametersに「-w2 "%HEX_FILE_NAME"」のように「"」で囲むことです。
これは空白を含むファイルネームの時に有効に機能します。

以下のプログラムでは問題なく動作しました。

```
/*
 * Project name:
     UART (Simple usage of UART module library functions)
*/

char uart_rd;

void main() {

  UART1_Init(9600);               // Initialize UART module at 9600 bps
  Delay_ms(100);                  // Wait for UART module to stabilize

  while (1) {                     // Endless loop
   if (UART1_Data_Ready()) {      // If data is received,
     uart_rd = UART1_Read();      //   read the received data,
     UART1_Write(uart_rd);        //   and send data via UART
    }
  }
}
```

- senshu 2009-01-27 (火) 18:01:58
  こうしたツールを利用すると、RAINさん作のAVRライタのアイコンが映えます。
  正に、どこかの製品の感じがします (気のせい？)

↑

JA1WXY版、hidspx用のGUIフロントエンド †

JA1WXYさんのページに、hidspx用の簡易GUI(D&Dに対応)が公開されています。

http://www15.plala.or.jp/ja1wyx/seisaku/hidaspx/hidaspx1.html

ソースコード公開なので改良は容易です。各自が希望する機能を追加してみましょう。

↑

FUSEデータ確認方法(Windowsの場合) †

avrx-tool.txtに書かれていない場合には、データシートを読むのが一番なのですが、必ずしも読みやすくはありません。どういう設定を行えば希望する動作が出来るかは、経験を必要とします。

そこで、hidspxにWebブラウザとの連携機能を追加しました。CMD窓で「start URL」と入力すると該当のページを開くことができますが、この機能を積極的に利用するわけです。

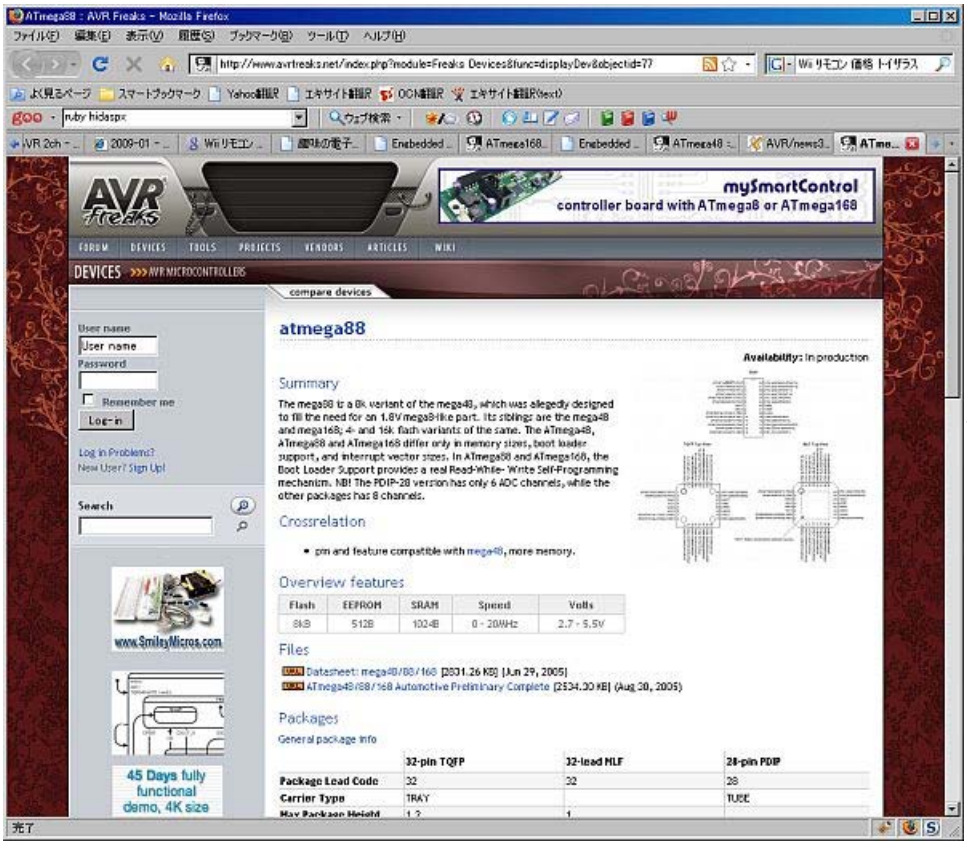
機能を拡張で考慮したことは、大幅な修正は行わない、比較的簡単に行える修正で利便性を増す、の2点です。そこで、avrライタに接続しているAVRマイコンに関する情報を適切に提供できる機能を追加するのが望ましいと考え、ライタから得られるFUSE情報を元に適切なページを開く機能を追加しました。

(1) 従来とおり「hidspx -rf」で確認出来ますが、詳細な情報が得られないため、希望する設定に変更するためには、データシートを参照する必要がありました。

```
Detected device is ATmega88.

Low: 11111111
     ||||++++-- CKSEL[3:0] システムクロック選択
     ||++-- SUT[1:0] 起動時間
     |+-- CKOUT (0:PB0にシステムクロックを出力)
     +-- CKDIV8 クロック分周初期値 (1:1/1, 0:1/8)

High:11-11101
     |||||+++-- BODLEVEL[2:0] (111:無, 110:1.8V, 101:2.7V, 100:4.3V)
     ||||+-- EESAVE (消去でEEPROMを 1:消去, 0:保持)
     |||+-- WDTON (1:WDT通常動作, 0:WDT常時ON)
     ||+-- SPIEN (1:ISP禁止, 0:ISP許可) ※Parallel時のみ
     |+-- DWEN (On-Chipデバッグ 1:無効, 0:有効)
     +-- RSTDISBL (RESETピン 1:有効, 0:無効(PC6))

Ext: -----000
          ||+-- BOOTRST ※データシート参照
          ++-- BOOTSZ[1:0] ※データシート参照

Cal: 170
```

そこで、この機能を拡張し「hidspx -ri」では、得られたFUSE情報を元に、以下のURLを開きます。このページを利用すれば、データシートが手元に無くとも(あったほうが良いですが)、FUSEの設定内容を確認し、変更内容を詳細に検討できます。

(2)「hidspx -rd」では、AVRマイコンのピン接続やデータシートのページを参照できます。このページでは、ピン接続やデータシートの入手、閲覧、類似機能を持つマイコンとの比較などが可能です。



「--atmel-avr」や「--avr-devices」もお試しください。教育現場の実習では、実習用の多数のPCにAVR関連のブックマークが登録されていることは期待できません。しかし、この機能があれば容易にAVR関連情報に到達できます。

これら機能の追加に要したコードはニ百行未満だと思いますが、かなり使い勝手が向上していると考えます。(段階的に機能を追加したので、何度も公開ファイルを差し替えましたが)

私なりに、コンパイルを行わなくとも各種の連携が可能なように、利用者が考えてコマンドやファイルとの連携を可能にする仕組みを追加しました。

基本はブックマーク登録機能ですが、各種のアプリケーションも起動できます。
特に、付属の説明書はかなりの時間を費やして書いていますが、なかなか読んでもらえません。--Helpや--hidspxで説明書を開く機能を追加しましたので、読んでくれる方が増えるのを期待します。

↑

FUSEデータ確認方法(Linux, MacOSの場合) †

-riオプションにより、オープンすべきURLが表示されます。

```
hidspx -ri ...  「`」は逆クォート(日本語キーボードでは「Shift+@」)です
```

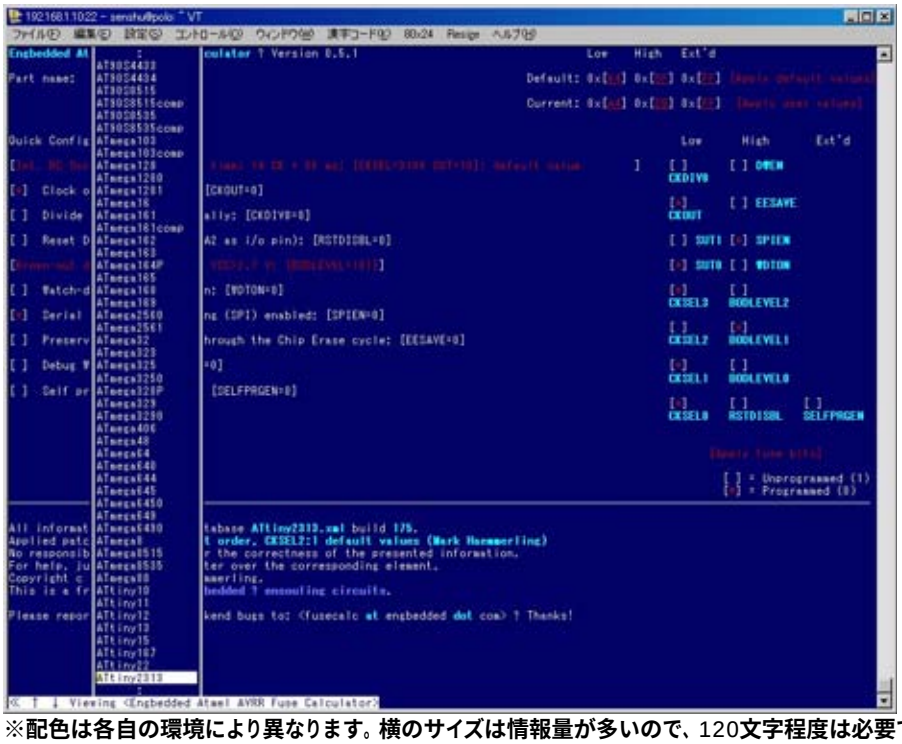
X環境で端末を使っている場合には、URL部分をクリックすれば、そのURLを開くことが できます。マウス操作が面倒だったり、テキストブラウザを利用したい場合には、以下の 方法を利用してください。

↑

w3mを利用する †

w3mをインストールしていない方は、インストール後に利用してください。
(w3mは、山形大学の伊藤さんの作です)

```
w3m `hidspx -ri` ...  「`」は逆クォート(日本語キーボードでは「Shift+@」)です
```



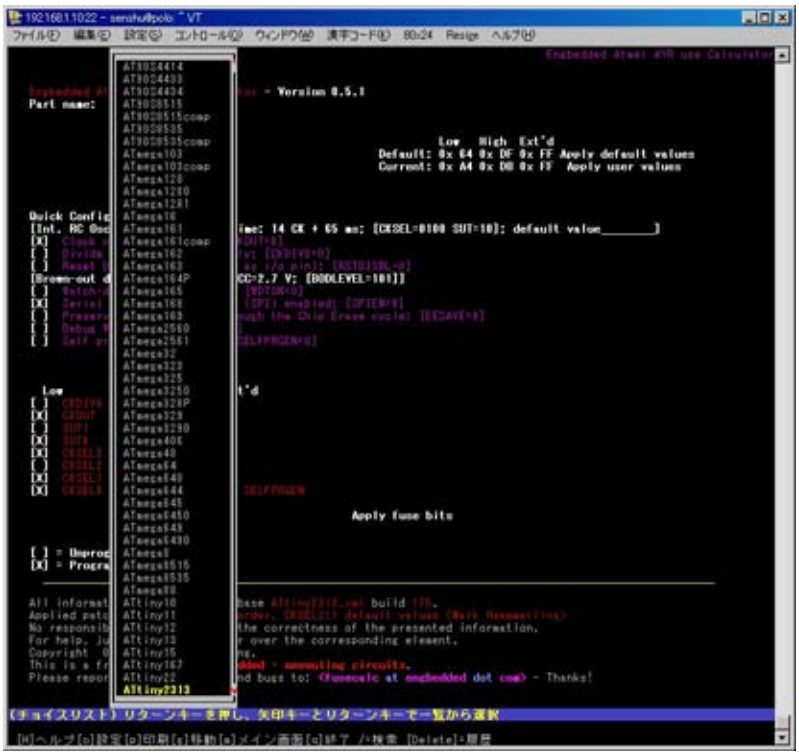
※配色は各自の環境により異なります。横のサイズは情報量が多いので、120文字程度は必要です。

↑

lynxブラウザ †

長い歴史を持つlynxを使った例です。レイアウトはw3mに比べ見劣りがしますが、lynxの方が
横の文字数は少なくても操作が出来ます。

```
lynx `hidspx -ri`
```



↑

文字ベースのWebブラウザ利用上の注意点 †

慣れは必要ですが、どちらもTABとEnter、矢印キーで操作してください。qで終了できます。
（慣れれば、通常のWeb利用もこれでOKという方もいらっしゃいます。lynxは数年間、
Windowsの日本語版を私的にメンテしていました。）

↑

GUIブラウザ †

firefoxやkonquerorなどのGUIブラウザ使う場合も同様です。X-Window-systemを使っている場合に利用できます。リモートログインではこの方法は機能しませんので注意してください。

```
firefox `hidspx -ri`&  ... (`&のように、続けて&をつけて起動してください)
 あるいは
konqueror `hidspx -ri`&
```

この後、firefoxが起動し、制御はそちらに移りますが、ターミナルに制御を戻し、Enterキーを押下すれば、操作を継続できます。

```
$ firefox `hidspx -rd`&
# hidspx -d4 -ph --new-mode -rd
[1] 6663
$ Detected device is ATtiny2313.    -> Enterキー押下

[1]+  Done                    firefox `hidspx -rd`
```

↑

リモートログイン時にWebブラウザで開く †

WindowsからTeraTerm経由でLinuxを使っている場合には、表示されたURLをマウスでダブルクリックすればWindows上の既定のブラウザが開きます。

↑

時間計測機能について †

Linux版では、当初は時間計測機能しませんでしたが、谷岡さんの協力により、以下のようにWindows版と同様の表示が可能になりました。

```
usl-5p:~/hidspx-2009-0126b/src# time ./hidspx -d1 /home/landisk/avrsttest.hex
# hidspx -d4 -ph --new-mode -d1 /home/landisk/avrsttest.hex
Detected device is ATtiny2313.
Erase Flash memory.
Flash memory...
Writing   [##################################################]   1000,    0.64s
Verifying [##################################################]   1000,    0.36s
Passed.
Total read/write size = 2000 B / 1.37 s (1.43 kB/s)

real    0m1.382s
user    0m0.000s
sys     0m0.032s
```

↑

hidspx(HIDaspx)とavrdude(AVRISP-mkII)の速度比較 †

avrdude ver5.6の不具合が解消したので、HIDaspxとの比較を行いました。

■ hidspx (USB HUB経由) でATmega88の全領域を読み出し、ファイルに書き出す

```
>hidspx -d1 -rp -oa1.hex
Detected device is ATmega88.
Flash Memory...
Reading   [##################################################]   8192,    1.52s
Passed.
Total read/write size = 8192 B / 1.61 s (4.97 kB/s)
```

■ avrdudeとAVRISP-mkIIを使い、同様の操作を行う

```
>timeit avrdude -p atmega88 -P usb -c avrisp2 -U flash:r:a.hex:i

avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.02s
avrdude: Device signature = 0x1e930a
```

```
avrdude: reading flash memory:
Reading | ################################################## | 100% 2.39s
avrdude: writing output file "a.hex"

avrdude: safemode: Fuses OK
avrdude done.  Thank you.

Elapsed Time:     0:00:02.906
```

```
>diff -u a.hex a1.hex
--- a.hex       Thu Mar 19 11:52:05 2009
+++ a1.hex      Thu Mar 19 11:51:31 2009
@@ -65,5 +65,5 @@
 :200800003B5A0BB9D0F2279528F4515029F4220F0000F9CF012756E027950BB920F451509B
 :2008200021F4220FF9CF012756E0299133230BB921F6037F10914701110FC651D0400BB9EB
 :2008400011F01093410111E01CBB08601AB1137F402F437F5F9100C000C00BB91AB94BB9E9
-:0608600066CFFFCF5AFF36
+:2008600066CFFFCF5AFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF36
 :00000001FF
```

結果ですが、最後のデータが32バイト区切りではないために違いが表示されていますが、
全く同じものです。

AVR用書き込み器の重要な機能にFUSE操作があります。

■ **hidspx(USB HUB経由)でFUSEデータを表示する(その1)**

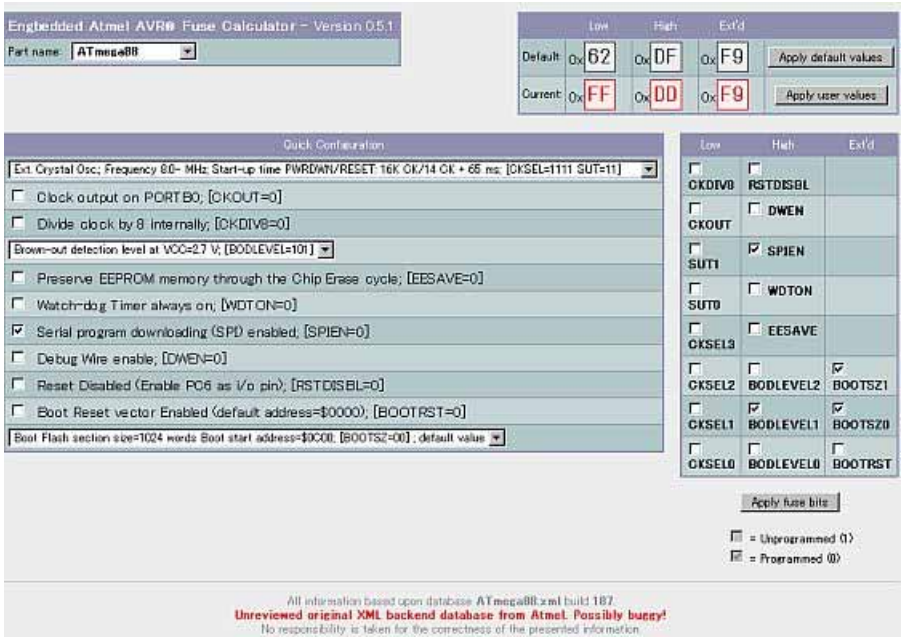
```
>hidspx -rF
Detected device is ATmega88.

DEVICE=atmega88
### hidspx command line example ###
hidspx -qmega88 -d10 -fL0xFF -fH0xDD -fX0xF9

### avrdude command line example ###
avrdude -c avrdoper -P stk500v2  -p m88 -U flash:w:main.hex:i ¥
 -u -U lfuse:w:0xff:m -U hfuse:w:0xdd:m -U efuse:w:0xf9:m
```

■ **hidspx(USB HUB経由)でFUSEデータを表示する(その2)**

```
>hidspx -rf
Detected device is ATmega88.

Low: 11111111
     ||||++++-- CKSEL[3:0] システムクロック選択
     ||++-- SUT[1:0] 起動時間
     |+-- CKOUT (0:PB0にシステムクロックを出力)
     +-- CKDIV8 クロック分周初期値 (1:1/1, 0:1/8)

High:11-11101
     |||||+++-- BODLEVEL[2:0] (111:無, 110:1.8V, 101:2.7V, 100:4.3V)
     ||||+-- EESAVE (消去でEEPROMを 1:消去, 0:保持)
     |||+-- WDTON (1:WDT通常動作, 0:WDT常時ON)
     ||+-- SPIEN (1:ISP禁止, 0:ISP許可) ※Parallel時のみ
     |+-- DWEN (On-Chipデバッグ 1:無効, 0:有効)
     +-- RSTDISBL (RESETピン 1:有効, 0:無効(PC6))

Ext: -----001
          ||+-- BOOTRST ※データシート参照
          ++-- BOOTSZ[1:0] ※データシート参照
```

■ **hidspx(USB HUB経由)でFUSEデータを表示する(その3)**

```
> hidspx -ri
```



■ avrdudeとAVRISP-mkIIを使い、同様の操作を行う

```
>avrdude -p atmega88 -P usb -c avrisp2 -U lfuse:r:-:h -U hfuse:r:-:h -U efuse:r:-:h -q

avrdude: AVR device initialized and ready to accept instructions
avrdude: Device signature = 0x1e930a

avrdude: reading lfuse memory:0xff

avrdude: reading hfuse memory:0xdd

avrdude: reading efuse memory:0x1

avrdude: safemode: Fuses OK
```

avrdudeは私が手をいれた ver 5.6で評価しました。#による進捗表示は類似していますが、hidspxは非常にコマンドが簡潔で、全実行時間や読み出しサイズ情報も確認できます。

素朴な疑問ですが、hidspxでは通常のAVRマイコン名、avrdudeではデバイスシグニチャを表示しますが、avrdudeの表示ではデータシートが必須で、正しく認識されたかが不明です。こうした仕様になっているのは何故でしょうか。デバイス名を省略できないので、これでも問題無いのかも知れませんが、ソースを修正しAVRマイコン名を表示するようにしました。

私はhidspxに慣れているので、avrdudeの哲学はよくわかっていません。コードを読む限り、堅実な実装に徹している印象ですが、機械的に整形した（と思われる）ソースはあまり読み易くはありません。（もっと読みやすく整形して欲しいです）

複数の開発者が並行して開発しているので、内部の情報をできるだけ開示しながら動作するように書かれているのだと思います。（-qを指定しない時の表示はDEBUGモードと錯覚します）

さらに、AVRマイコン用の諸データをプログラムには埋め込まずに外部のテキストファイルに格納するという柔軟な手法を取り入れています。このデータの解析にもそれなりの時間が必要です。hidspxでは、プログラムの中に埋め込んでいるので解析する必要はありませんが、新規のデバイスに対応するには、ソースの修正が必要です。

avrdudeがAVRISP-mkIIに最適化されているとは限りませんので、HIDaspxに有利な評価試験ですが、どちらも3秒未満です。この程度の違いは通常の利用では全く問題ではありません。

このように、USB HUB経由でHIDaspxを利用すると市販ライタを上回る高速性が得られる場合もあることがわかります。

単なる速度試験では信頼性や使い勝手などは比較できませんが、この評価試験でHIDaspxの実力を再確認できました。

↑

hidspxで読み出し内容を16進形式で表示する †

hidspx でターゲットマイコンのフラッシュメモリを読み出すと、

```
>hidspx -rp
Detected device is ATtiny2313.
Flash Memory...
Reading   [##################################################]   2048,    1.11s
Passed.
:200000001FC01FC137C036C035C035C033C032C031C030C02FC02EC02DC02CC02BC02AC0F9
:2000200029C028C027C0022324232222022323032225050223242322220223262223440003D
:2000400011241FBECFEDCDBF10E0A0E6B0E0E2E3F3E003C0C89531960D92A437B107D1F7C7
:2000600010E0A4E7B0E001C01D92A638B107E1F71AD15EC1C5CF1F920F920FB60F9211240C
:200080002F933F934F938F939F93EF93FF9340917600442309F4C2C020917700300917800F4
  :
:2003200085B78F7D85BF87EE88BB8AE187BBE5CFFFCF100820000000000B3179A1ADC1D9B
:20034000EE135E160000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF42
:00000001FF
Total read/write size = 2048 B / 1.25 s (1.60 kB/s)
```

のように表示されます。これはファイルに書き出すとそのままHEXファイルとして扱える
ので便利なのですが、大多数の人間にとって読み易いものではありません。

そこで私は、srec_catと組み合わせて利用しています。

```
>hidspx -rp | srec_cat -o - -hexdump - -i
```

以下の指定で特定番地(この例では300番地台)を抜き出すこともできます。

```
>hidspx -rp | srec_cat -o - -hexdump - -i | grep 000003[0-9]0
Detected device is ATtiny2313.
Flash Memory...
Reading   [##################################################]   2048,    1.12s
Passed.
Total read/write size = 2048 B / 1.25 s (1.60 kB/s)
00000300: E1 F7 90 91 81 00 99 23 C1 F7 87 EE 88 BB 97 BB  aw.....#Aw.n.;.;
00000310: 85 B7 8F 7A 80 61 85 BF 85 B7 80 62 85 BF 88 95  .7.z.a.?.7.b.?..
00000320: 85 B7 8F 7D 85 BF 87 EE 88 BB 8A E1 87 BB E5 CF  .7.}.?.n.;.a.;e0
00000330: FF CF 10 08 20 00 00 00 00 00 B3 17 9A 1A DC 1D  .0..  .....3...¥.
00000340: EE 13 5E 16 00 00 FF FF FF FF FF FF FF FF FF FF  n.^.............
00000350: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
```

最後の部分をless(more)に置き換えれば、画面毎にじっくり内容をチェックできます。

```
>hidspx -rp | srec_cat -o - -hexdump - -i | less
```

hidspxは、一般的なUNIX系のツールと同様にツール間の連携が容易です。

かなり前(20年位？)に読んだUNIXの解説本に、UNIXの効果的な利用法は
「できるだけコードは書かずに、既にあるものを利用して問題を解決する」
と書いてありました。既存のツールを利用し、開発効率を上げるという考え方です。

参考ページ ⇒ srec_catの使い方
http://www-ice.yamagata-cit.ac.jp/ken/senshu/sitedev/index.php?embtech%2F01#content_1_4
audinさん ⇒ http://avr.paslog.jp/

なお、この使い方では問題はありませんが、特定のツール(Notepad++Portable)と組み
合わせた時に、「stderr/stdoutの表示の一部が混ざり合う」不具合を確認しましたので、
標準出力のバッファリングを行単位にする、という変更を実施しました。(2009/03/07以降)

上記のコマンド列が覚えられない場合には、bashなどのalias機能を使い、コマンドを
.bashrcファイルなどに登録します。

```
＊.bashrcなどのRun Commandに設定する
--------
alias avrdump=hidspx -rp | srec_cat -o - -hexdump - -i
--------
```

avrdump でこの(やや長い)コマンド列を簡単に呼び出すことができます。BATファイルにするのも一案ですがaliasなら無駄がありません(Windowsでは、小さなファイルでも最低でも数十kBのdiskを占有します)

```
avrdump | less
```

のようにすれば、専用プログラムのように利用できます。常用するコマンドを登録し、使い勝手向上することができます。

↑

hidspx-***.zipファイルの解凍が遅い †

kumanさんより (2009-02-07 (土) 14:11:27)

```
HIDaspx用の hidspx-2009-0115はWinXPのエクスプローラのドラッグ&ドロップ
機能で解凍できるのですが、2009-0121a, 2009-0126は途中でフリーズして解凍
できません。私のPCだけでしょうか。Lhaplusで解凍できましたが、hidspx.exeの
解凍時に20秒ほど時間がかかります。Lhaexでも解凍できましたがこのときは同じ
exeファイルの解凍に40秒ほど かかります。0115版では連続して解凍されます。~
```

という問い合わせがありました。

やり取りを行った結果、**ウイルスバスターが原因と判明**しました。

kumanさん 2009-02-08 (日) 02:44:19

```
わかりました。
抗ウイルスソフトを禁止するとほぼ瞬時に解凍できました。
ソフトはウイルスバスターですが、これの影響でした。
0115と違って0126はこのソフトにとって気に入らないところが
あるのでしょうね。
お騒がせしました。ありがとうございました。
```

ウィルスと誤認されたのでは困りますね。2009-0307版では、どうでしょうね。

↑

実行時間の計測方法 †

現在公開中のhidspxには、実行時間を計測する機能が備わっています。この機能は2009年以降に追加した機能ですので、以前の版を利用している方は、次の項の説明を参考にしてください。(特に理由がなければ、アップデートを強くお薦めします)

```
■ 2009年以降の版での動作例
C:¥hidspx-2009-0125b¥bin>hidspx.exe -d1 avrsttest.hex
Detected device is ATtiny2313.
Erase Flash memory.
Flash memory...
Writing   [################################################]   1000,    0.45s
Verifying [################################################]   1000,    0.36s
Passed.
Total read/write size = 2000 B / 1.11 s (1.76 kB/s)
```

HIDaspxの動作報告では、実行時間の報告をお願いしています。

時計があれば計ることができますが、短い時間を正確に計測することはできません。

そこで、作成したプログラムの性能評価のためのストップウォッチ的なソフトウェアが必要になります。

私は、動作時間の計測には、MS社の提供する無償ツールtimeitコマンドを使い、1/1000秒の分解能(実際には1/100秒程度?)で計測を行っています。

以下が、timeitコマンドで、ATtiny2313の全メモリを書き出し・照合に要する時間を計測した結果です。8MHzで動作しているATtiny2313の2kBの書込み・照合を1.03秒で完了していることがわかります。手動計測では、人間の反応時間を計っているようなものであり、この種のツールを利用しなければ計測は不可能です。

```
>timeit hidspx -d1 2313.hex
 ^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
Detected device is ATtiny2313.
Erase Flash memory.
Verify  Flash: 2048/2048 B

Passed.

Version Number:    Windows NT 5.0 (Build 2195)
Exit Time:         10:40 am, Thursday, October 30 2008
Elapsed Time:      0:00:01.031
Process Time:      0:00:00.265
System Calls:      296309
Context Switches: 1945
Page Faults:       616
Bytes Read:        10106
Bytes Written:     4876
Bytes Other:       8326
```

詳細は、@ITの紹介記事
http://www.atmarkit.co.jp/fwin2k/win2ktips/422timecmd/timecmd.htm **を参考にしてください。**

コマンドプロンプトから利用しますが、計測の対象はWindows上で動作する全アプリケーション
です。実に便利で、標準でOSに含めて欲しいツールです。各種のスイッチを持っていますが、
非常にUNIXの香りのするツールです。

```
Usage: TIMEIT [-f filename] [-a] [-c] [-i] [-d] [-s] [-t] [-k keyname | -r keyname]
 [-m mask] [commandline...]
where:        -f specifies the name of the database file where TIMEIT
                 keeps a history of previous timings.  Default is .¥timeit.dat
              -k specifies the keyname to use for this timing run
              -r specifies the keyname to remove from the database.  If
                 keyname is followed by a comma and a number then it will
                 remove the slowest (positive number) or fastest (negative)
                 times for that keyname.
              -a specifies that timeit should display average of all timings
                 for the specified key.
              -i specifies to ignore non-zero return codes from program
              -d specifies to show detail for average
              -s specifies to suppress system wide counters
              -t specifies to tabular output
              -c specifies to force a resort of the data base
              -m specifies the processor affinity mask
```

実行結果は、STDERR**に出力されますので、**

```
(timeit 計測するコマンド) 2>&1 > 結果
                          ^^^^
```

と指定すれば、ファイルに書き出すことが出来ます。()で括ることで、コマンドの
範囲を明確にします。このコマンドは、過去に実行したコマンドの統計情報を出力する
こともできる強力なツールです。

↑

コメント †

- senshu 2009-08-24 (月) 12:37:19
  実際に試してみての感想をお書きください。

  hidspx**のコマンドはシンプルであり、その使い方に慣れた方には、特に**GUI**のメリット**
  は感じないかも知れません。しかし、シンプルであっても多少の学習は必要です。
  hidspx-GUI**ならマウスによる操作が可能で、コマンドオプションを知らなくとも基本**
  操作ができます。さらに、Fuse Calc**とのスムーズな連携、**HEX**ダンプ表示のスクロール、**
  表示データのコピーなども可能です。

  CUI**の環境では多少の学習が必要になる操作でも、**GUI**環境では学習なしでも、それなりの**
  操作が可能です。HELP**でNews**を選択すれば、最新のアーカイブのダウンロード先が開きま**
  す。どこから DownLoad**するのですか？といった疑問も生じません。**

  私が特に力を注いだのは、AVR studio**との連携機能です。**BASCOM-AVR**やMikroC**では**GUI**は**
  不要ですが、AVR studio**では、**hidspx**を呼び出すだけでは実現できませんでした。**

  そのため、hidspx-GUI**やhidspxG**の機能にこの連携を可能する機能を盛り込みました。

このページに書かれているような使い方をすれば、利点は自ずと理解できると思います。

お名前:

コメントの挿入
