1chipマイコン（AVR）を使用して、周波数カウンタを作る

2010.6.9
JA1VCW

１．　はじめに

AVRを触ってみようと思い、一番簡単な入門コースのLED点滅を作りましたところ、比較的容易に動作しました。 プログラムはBASCOM-AVRのお試し版です。
さて、次に練習するものは何かと考えて、周波数カウンタ（Fカウンタ）にしました。
以前からLSIを使ったものはあるのですが、マイコンでFカウンタができればいろいろと応用が利くのではないかと思った次第です。
たとえば、アナログVFOのAFCや昔の受信機の周波数表示などです。
AVRのアプリケーションとしてのFカウンタは、インターネットなどで検索しますと、いくつも出てきます。 皆様工夫をされています。
私もいろいろ参考にさせていただきながら、自己流で作って見ました。
今回の方法が最良というわけでもなく、別な回路方式やプログラム方法、矛盾などもいろいろとあります。 Fカウンタを作っただけでご大層なものではないのですが、そのとき考えたことやプログラムの練習についての、これは備忘録です。
枚数が多いのですが、字が大きいためです。（目が追いつかなくて）

２．周波数カウンタの基礎

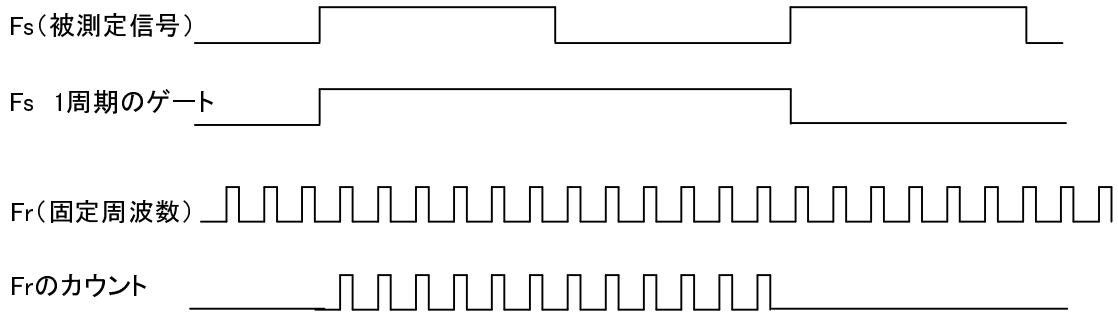
Fカウンタの動作原理については、これもいろいろな所に書いてありますが、復習します。

２．１　直接計測

周波数は1秒間に何回振動しているかということなので、被測定信号を１秒間切り取ってその振動の数を数えればよいことになります。
たとえば被測定信号が100MHzであれば、1秒間の振動数は100,000,000であり、カウンタの表示は100,000,000となります。 9桁の計数器が必要となります。 また、2Hzであれば2という表示になります。
重要なことは、切り取る時間（ゲート時間）を1秒とした場合は
１）測定には1秒以上かかること。（ゲート時間＋表示などの処理時間）
２）測定の最小分解能は1Hzとなること。
周波数はアナログ量ですので、1.7Hzなどということもありますが、この場合は振動の個数を数えているので小数点以下はカウントできません。 1または2の表示になります。
もし被測定周波数が1.7Hzの場合は、ゲート時間を10倍にして10秒間の振動を数えれば17個となって、小数点1位までの測定ができます。
このようにゲート時間と測定精度には関係があって、精度を上げようとすると測定時間が長くなります。 被測定周波数をmHzまで測定しようとすればゲート時間は1000秒となり、現実的ではありません。
しかし工夫はあります。 後に述べるレシプロカルカウンタや、ゲート時間以下の周波数を電圧に変換して電圧計で値を測定し、桁数を上げる方法などがあります。

２．２ レシプロカル・カウンタについて

直接計測では、測定周波数の分解能を上げるにはゲート時間を長くすることが必要でした。
固定周波数(Fr)のパルスを被測定信号(Fs)で切り取って、そのパルス数をカウントすれば被測定信号1周期に対してのFrのパルス数がわかるので、計算で被測定周波数数が求まります。　通常周波数関係は　Fr>>Fsです。



１）たとえばFr = 1MHz = 1us　とする。
２）Fsによってゲートされたパルスをカウントして“12”なので 、Fsの1周期は　1us*12=12us。
３）Frの周波数は　Fr　=　1/T　=　1/12us　　なので、計算を行って83.33.・・・kHz。
４）もし”13”カウントであれば同様に、1/13us　=　76.92kHz
５）従って周波数の分解能は　6.41kHzとなる。

この例では測定分解能がはなはだ悪いですが、分解能をあげるには
１）Frの周波数を高くする。
２）Fsを分周する。
要するに測定値のカウント数が大きくなるようにすれば良いので、 実際には両方を組み合わせて精度を上げるのがよさそうです。
因みにレシプロカルというのは逆数のことです。　まさに逆数を求めています。

３．　使用するマイコン

最近はいろいろな1chipマイコンがありますが、秋葉原で安価で購入できるような物が良いので、あまり考えずにAVRにしました。 他に有名なものには、PICがあります。
AVRにした理由を挙げれば、私個人の主観的なものですが、
１）アセンブラが比較的きれいであること。
２）PICよりも後から発売されたので、恐らくよりスマートであるだろうと考えた。

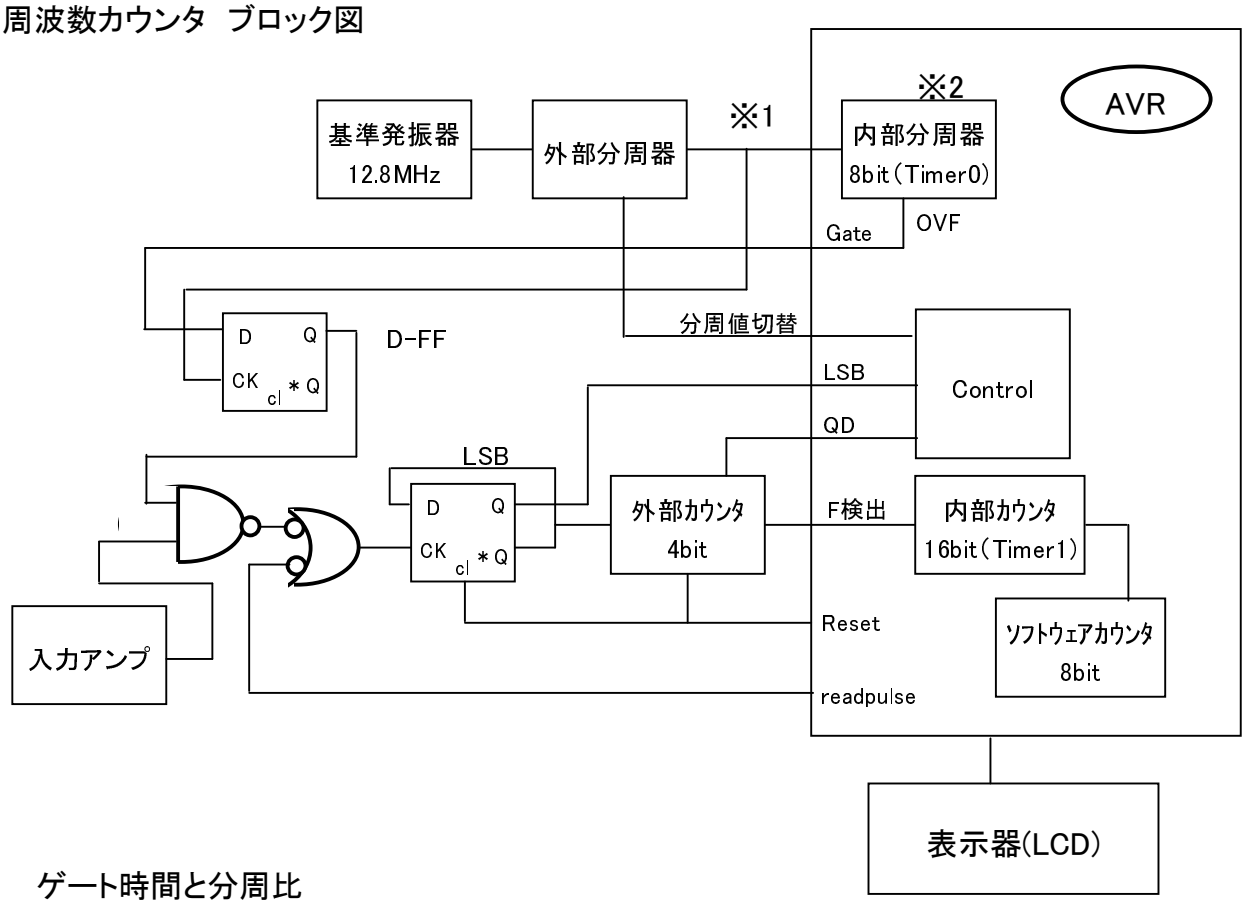
どちらも性能は同じ様で、一方でできて他方で出来ないなどと言うことはほとんど無いと思っています。（よく調べたわけではありませんが・・・）

4.　マイコンを使用して周波数カウンタを作ることについて
それではマイコンはこの機能の中でどの部分を担当させればよいかということを考えます。

4．1　全体の構造
今回使用したAVRマイコンには8bit*2個,16bit*1個のカウンタが内蔵されているのでそれらを使用して、次の仕事を担当させます。

1)ゲートタイムを作る。(Timer0)
2)被測定周波数をカウントする。（ LSB + 外部カウンタ + Timer1 + ソフトウェアカウンタ）
3)LCDモジュールが簡単に接続できるようになっているので、それを接続して値の表示。

周波数カウンタ　ブロック図



ゲート時間と分周比

※1　　※2 （Timer0の分周比）

外部分周器出力		Timer 0	全分周比	ゲート時間
周波数(Hz)	間隔(ms)	分周比		(s)
12.5	80	125	128000000	10
100	10	100	12800000	1
100	10	10	1280000	0.1
1000	1	10	128000	0.01
8000	0.125	8	12800	0.001

４．２　こうなった理由
外付けICや回路がおおいのですが、理由はなるべく正確なカウンタにしたかったためです。
Fカウンタの性能が基準発振器の精度で決まるようにしました。
そのために次のように考えました。

１）ゲート時間を作るカウンタ(ゲートカウンタ）のオーバーフローや一致をインタラプトなどで
検出して被測定周波数のカウンタ（メインカウンタ）をストップさせる方式がありますが、
その場合はプログラムの実行時間のばらつきが精度に影響しますので、それを避ける
ためにゲートカウンター部をディスクリートのICしました。
２）同様にAVRのプログラムカウンタの出力を、Dフリップフロップを使用して基準発振器を
分周したクロックと同期をとりました。
３）２）の精度の高いゲート時間を、ディスクリートのゲートICでゲートしました。
４）被測定周波数をなるべく高くしたいので、5bitのプリスケーラをつけました。
５）ゲート時間を1ms～10sまでディケードで変えたいため、外部分周器を付けしました。

１）では例えばゲートカウンタのオーバーフロー（一致出力でも同じ）を処理するプログラムが
アセンブラプログラムで1ステップのばらつきが発生したら、12.8MHzのCPUクロックでは
約78nsのばらつきになります。 1秒ゲートでの誤差は 78ns/1s = 78e-9の誤差になります。
このことは 10MHzを測定した場合0.78Hzの誤差が発生する可能性があります。
通常インタラプトの処理のばらつきがアセンブラで1ステップに収まると言うことは考えずらい
のです。 まして、Basicではインタラプトあるいは変数のチェックの実行時間がどれだけか
わからない現状では、プログラムでのゲートの開閉（メインカウンタのstart/stop）は避けたいと
考えました。

２）というわけでハードでゲート時間を作りました。

３）同様にゲートもハードで実行。

４）AVRのカウンタ入力周波数は、CPUクロックの1/2以下にしろと書いてあります。
これはカウンタの入力信号をCPUクロックで同期を取っているためです。
今回12.8MHzなので、6.4MHz以下にする必要があります。
プリスケーラは4bitの場合、102.4MHz(6.4MHz*16）までの周波数は扱えます。
4bitのbinaryカウンタICはSN74HC161があって、30MHzまでカウントを保障しています。
そのカウンタの前にD-FFを付けて、1/2分周を行っていますので、60MHz + マージン程度の
測定範囲になります。（74AC161を使用すれば4bitで125MHzまでカウントできる可能性が
ありますが、今回は手持ちの関係でこのようになりました）

５）ゲート時間は1ms（kHz単位）と1s（Hz単位）でかまわないとも思ったのですが、そんなに
手間でもないので 1ms、10ms,100ms,1s,10sの5レンジにしました。

12.8MHzと内部のカウンタでは最大1024*256=262144分周なのでAVR内部だけではゲート時間は約20msまでとなります。 これ以上の長いゲート時間を得るためにはソフトのカウンタを作る必要があります。
ソフトカウンタを作って基準発振器と同期を取る方法もうまく行くと思われます。
いずれの場合も基準発振器と同期を取る必要があります。
今回はゲート時間を作るためにソフトのカウンタを使用する方法は採用していません。
（後から追加したレシプロカルカウンタはソフトでゲートをon/offしています）

5. プログラム言語について
マイコンですから、プログラムを作成しなければ動きません。
最近はほとんどのマイコンがいわゆる高級言語をサポートしています。
AVRも C言語、BASICなどが使用できます。
私の場合はBASICが好きで、今回はBASCOM-AVRのお試し版を使用してみました。
これは無料ですが、プログラムメモリが4kbyteまでという制限があります。
まっ、4kbyteでどのくらいの仕事をするかということも興味の一部です。

6.　各部の考え方

6．1　ゲート時間の作成　（次ページタイミング図も参照）

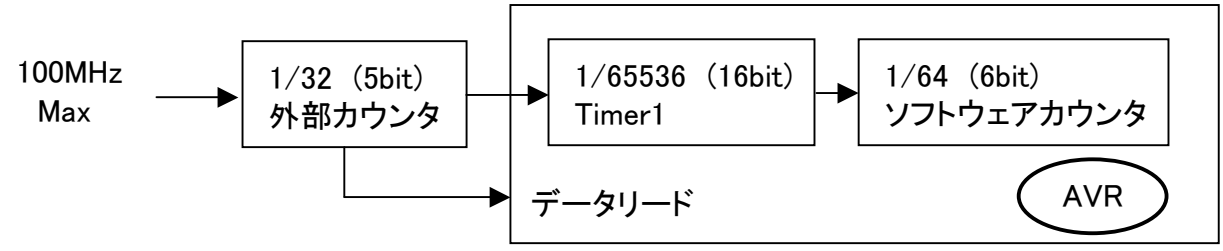
ゲート時間が1msの場合の動作を示します。 この場合次の設定や条件があります。
D-FFのQ = Gate = 8kHz = 125us ,　 Timer0の必要カウント数 = 8
Timer0比較値 = 7

1）Resetを解除することでゲート回路の計数が開始します。
2）外部分周器の最終段のQD出力（390_Dout）が“H”となった時点で、Timer0がカウントアップします。
3）基準発振器に同期して（D-FFのQ）が“H”となるとGateが開いて、信号の計数が開始します。
4）Timer0は（390_Dout）の出力を計数し、8パルスの入力後比較出力が“L”となります。
5）4）の結果、次のD-FFのclk入力で出力Qが“L”となり、Gateがクローズします。
　これで 0.125ms*8カウント ＝ 1ms となります。
6）注意しなければならないことは、Timer0の入力、出力ともにAVR内部でCPUクロックで同期が取られていることです。 この事はTimer0の出力は最大2*CPUクロックばらつく可能性があります。（CPUクロックとゲートの基準クロックの周波数が違った場合）
　このD-FFのclk入力は少なくとも2*CPUクロック以下の周波数にする必要があります。

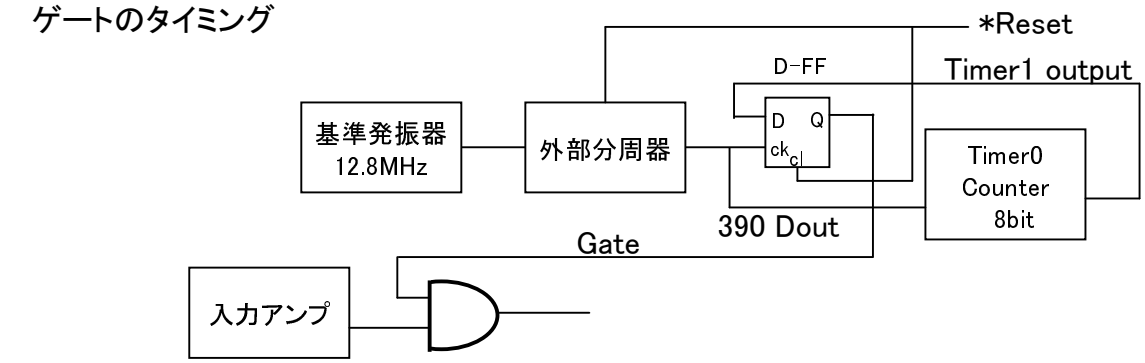
6．2　信号カウンタ

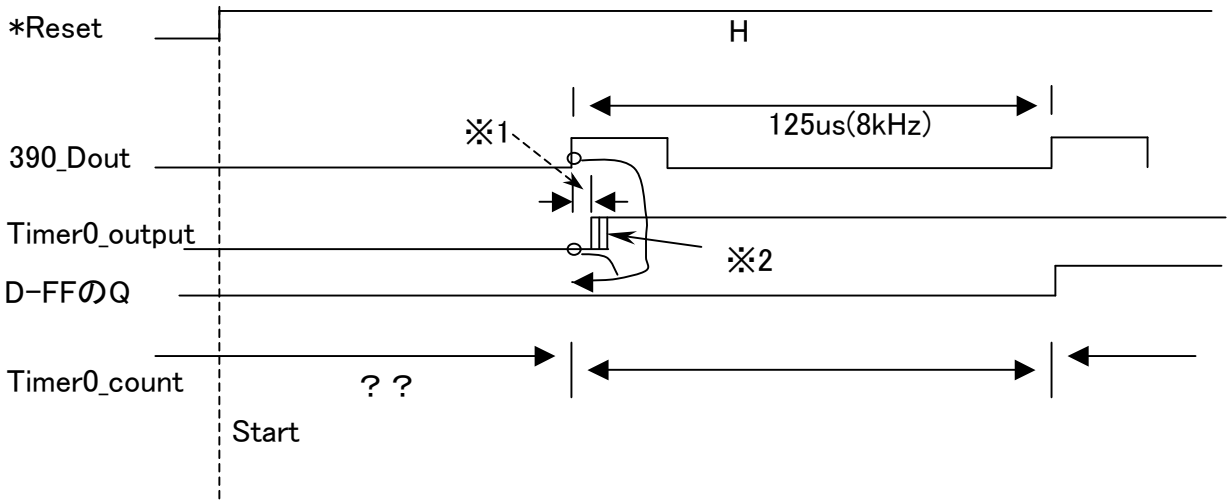
最高周波数99.999 999MHzの表示としたいので、最大の桁数を8桁とします。
この場合2進数だと 5F5E0FF (Hex) となるので、27bit必要となります。
16bit分はAVRの内部カウンタTimer1を使用します。
またTimer1の入力周波数はIOクロック12.8MHzの1/2.5以下とすると 5.12MHz以下です。
（取説にTimer1の入力はAVR内部でIOクロックと同期をとっているので、入力のdutycycleが約50%の時に、IOクロック半分以下の周波数にしろ、1/2.5以下が推奨と記述してあります）
信号入力周波数は100MHzとすると、Timer1の許容入力周波数は5.12MHzなので、
この間には外部分周器を入れます。 分周比は 100/5.12=19.6 => 20分周 あればよい。
16分周(4bit)では不足なので、FFの半分を追加して32分周(5bit)としました。
Timer1の最大入力周波数は 100MHz/32=3.125MHz となります。

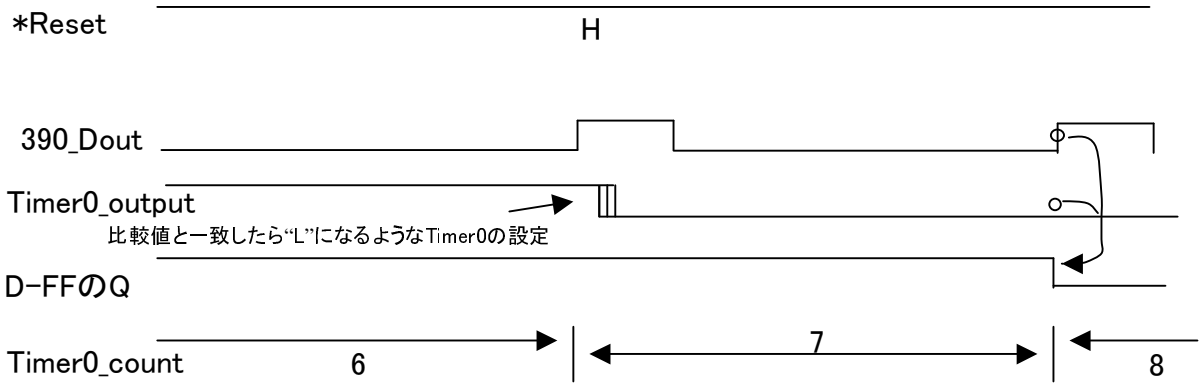
全体では27bit必要で、外部分周器(5bit) ＋Timer1 (16bit) = 21bit となるので、残り6bitはソフトウェアのカウンタを使用します。
この時その最大周波数のときのソフトウェアのカウント周波数は
100MHz / 32 / 65536 = 約48Hzなので 約20ms弱でカウントアップします。
この時間はソフトウェアで十分対応できます。



ゲートのタイミング





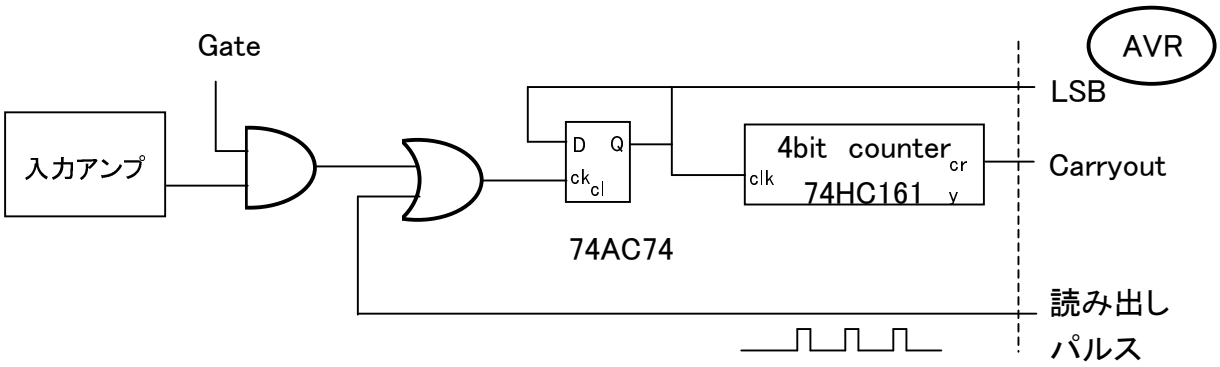


※1　Timer0 内部の遅れ。
※2　CPUクロックと基準発振器の周波数が異なると、ばらつきが発生する場合が有る。 またソフトでコントロールする場合、命令によってばらつきが発生する。

いずれもTimer0_outputの変化は、場合D-FFのクロック（390Dout）の立ち上がりと、その次の立ち上がりの間に収まっていなければならない。

６．３　外部カウンタの構成とそのデータリード

図のように　FF(74AC74)と4bitCounter(74HC161)で5bitのカウンタを構成します。
これら外部カウンタのカウント値をAVRに読み込む必要があります。
カウンタ出力の5本ラインをそのままAVRのポートに接続すれば良いのですが、ポートが
不足してきれいに5本の入力を接続することができません。 ポートを多重に使ったり、
切替回路を入れればできないことはないのですが、今回は3本のポートで代用します。
（無理に複雑にすることもないのですが、今回ソフト／ハードの練習でもあります）



１）Gateが閉じて入力の計数を終了すると、5bitのカウンタの値が決まります。
２）4bitCounterの出力にCarryがあります。 これはカウンタの出力4bit(QA-QD）が
　すべて“H”になった時、つまりカウント出力が“F”になったときのみ“H”になる機能です。
３）読み出しパルスをソフトで出力してLSBとCarry の両方が“H”になったら、読み出し
　パルスを止めます。 このときは5bitのカウンタの値が1F(Hex)になったことを意味します。
　読み出しパルスを出力してから1F(Hex)になるまでのパルス数を数えておけば、カウンタ
　のGateが閉じて止まった時の値が逆算できます。
４）読み出しパルスの数、　5bitカウンタ初期値(Hex)

0	1F
1	1E
2	1D
・	・
31	0

　欠点は時間がかかることですが、人間が見るような使い方でしたら問題ないでしょう。

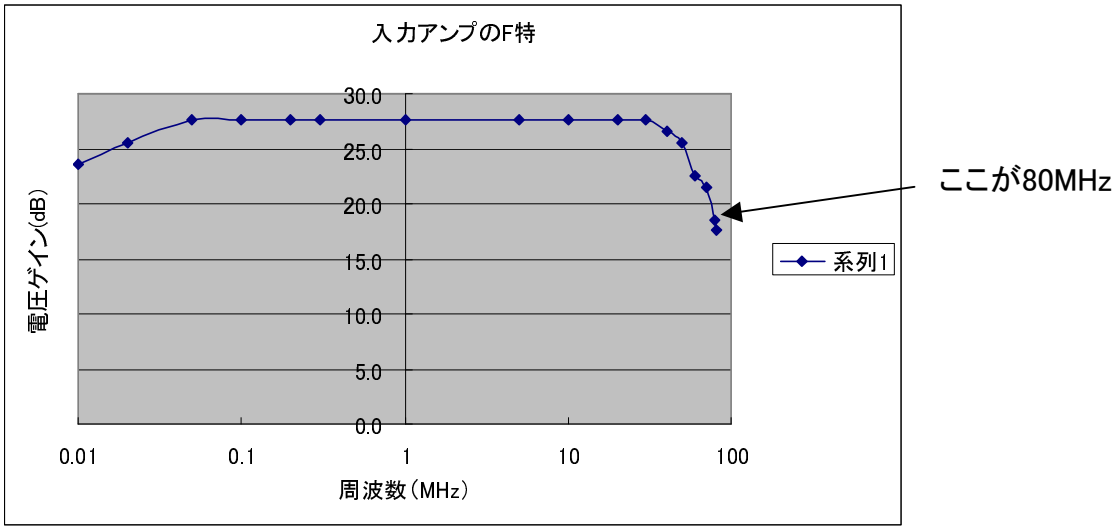
これを実現するためには読み出し用のパルスが必要です。
また、読み出しの時はGateが閉じていること、Gateが開いている時は読み出しパルスが
入力されていない事が必須です。（上図参照）

６．４　入力アンプについて

入力インピーダンスを上げることと、若干のゲインを持たすためのアンプを入力につけます。
波形整形ロジック（74AC14)の動作電圧は、2Vpp程度として、カウンタの入力電圧100mVpp
とすれば、26dBくらいのゲインがあれば良いことになります。 測定の大半は発振器などの
周波数の特定なので、そんなに小さいレベルを測る子他は無いだろうと考えました。
回路はいろいろなホームページを参考にして、且つ手持ちの部品で作りました。
特性を取ってみましたが、あまり上手にできませんでした。

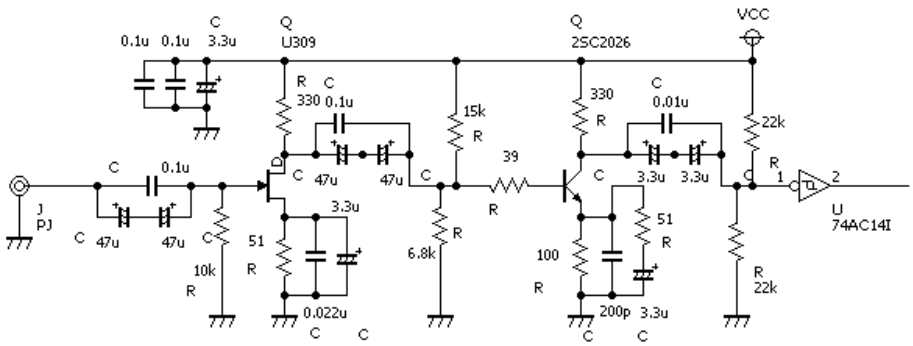
周波数特性

結果だけですが下のようになりました。 高域がゲイン不足ですが、80MHzあたりでおよそ
150mVpp、1MHzあたりで50mVpp程度でカウント開始するようです。



レシプロカルカウンタのための低い周波数についての特性は考慮していません。
データも取っていません。
低い周波数にゲインを持たせる場合、カップリングコンデンサの値は再考する必要があります。

2段目のエミッタのバイパスコンデンサ(200pF)が重要で、高い周波数の特性が決まります。
大きすぎると、全体のゲインもあがってしまいフラットな特性から外れてしまいます。
カットアンドトライできめましたが、エミッタのどの定数を変えてもF特に影響します。



7. プログラムについて

プログラムの流れです。 おおよそのイメージです。

AVRのタイプ宣言
CPUクロック周波数宣言
Portの宣言
変数の宣言
定数の設定
Portの初期設定
インタラプトの宣言
インタラプト イネーブル

→ ゲート時間のSW読み込み
ゲート時間のSWによる外部カウンタの設定

Timer0 （Pwm，入力エッジ，カウントモード）
Timer1 （カウンタモード，入力エッジ，カウンタクリアモード）
Timer0の比較値設定
外部カウンタのリセット解除

ゲートが閉じるのを待つ

Timer0,Timer1停止
5bitカウンタのデータリード(Ctl)
16bitカウンタのデータリード(Ctm)
ソフトカウンタ(Cth)
周波数 = Ctl+(CTm*32)+(Cth*2097152)

└ LCDに 周波数を表示

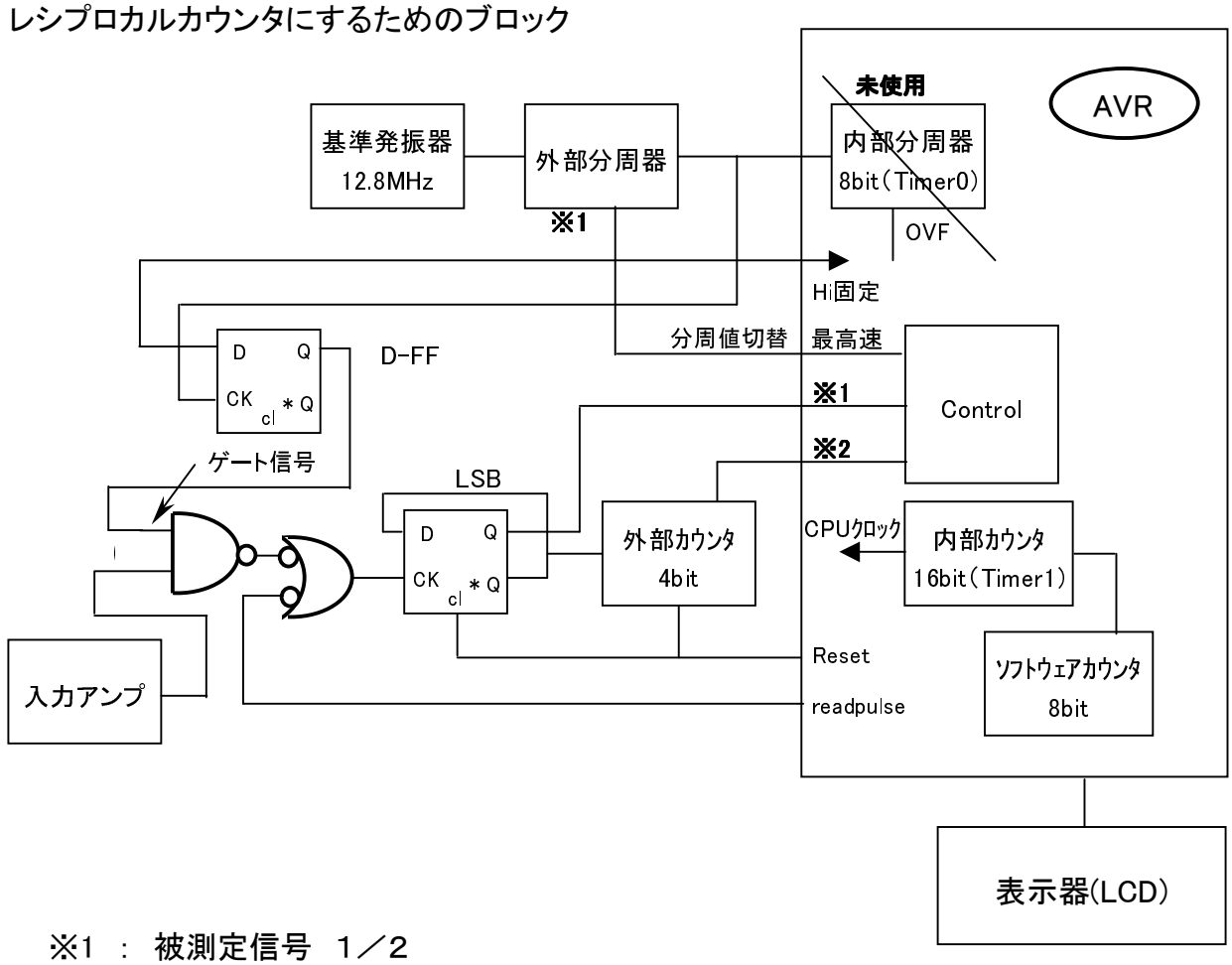
8.　レシプロカルカウンタの追加方針
周波数カウンタを作成しましたが、ついでにレシプロカルカウンタ動作も付けてみました。
ハードで実現するには単にゲート信号に既知の周波数を入力すればよいだけです。
このためには切替スイッチとそのコントロール信号、測定終了信号などが必要です。

今回はプログラムの勉強を兼ねていますので、この改造は行わずハードウェアの追加無しで
ソフトウェアのみで実施しました。

8．1　レシプロカルカウンタの実現方法
AVRの動作を変えるだけで、外部のハードウェアの変更はしません。

1）メインのカウンタにTimer1を使用して、CPUのクロック(12.8MHz)をカウントします。
2）Timet1を被測定信号でゲートします。 ソフトウェアでTimer1をstart/stopします。
3）カウント値を計算して周波数に変換します。
4）被測定信号の1/2と1/32の分周回路があるので、切替て測定分解能を上げられるように
　しました。

レシプロカルカウンタにするためのブロック



※1 ： 被測定信号　1／2
※2 ： 被測定信号　1／32
これらの信号の立ち上り（立ち下り）でTimer1をスタートし、立ち下り（立ち上り）で
ストップする。

5）被測定信号と基準信号周波数（CPUクロック：12.8MHz）の関係は表のようになります。
1mHz分解能を実現するためには（1mHz分解能の根拠は無い。私が決めた）入力周波数が100Hz以下であればよいことがわかります。
6）分周回路を使用すれば、分周したぶんだけ分解能が上がります。
7）測定周波数の最小を1Hzとした場合、カウント値は12800000 = C35000(hex)となるので、メインのカウンタの長さは24bit必要です。
8）逆数を計算するときの計算誤差が発生します。

測定周波数と基準信号周波数、分解能の関係

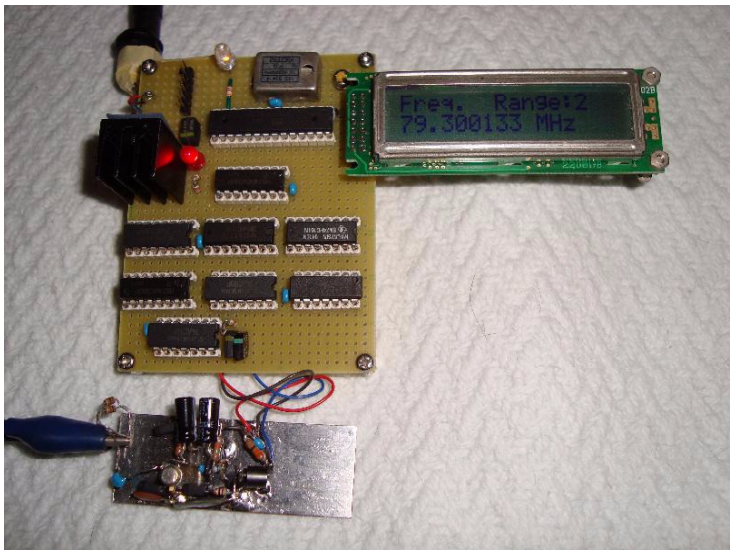
基準信号周波数： 12.8MHz

入力Freq.(Hz)	時間(ms)	カウント数	分解能(Hz)	表示
1	1000	12800000	0.0000	1.00000008
10	100	1280000	0.0000	10.00000781
100	10	128000	0.0008	100.00078124
1,000	1	12800	0.078	1,000.07811890
10,000	0.1	1280	7.806	10,007.80640125
100,000	0.01	128	775.194	100,775.19379845

分解能の計算は、たとえば100Hzを測定した場合、
基準信号周波数を12.8MHzとしたときは、基準信号周期78.125ns。
これを100Hzでゲートすると、1/100=10msなのでカウント値は128000。
分解能は測定カウント値1カウントぶんなので、128000と128001がカウントされる。
これを周波数になおして F1 = 1／（78.125ns*128000） と F2 = 1／（78.125ns*128001）
それぞれ 100Hz、100.0007815Hzとなるので分解能はおよそ0.0008Hzとなる。

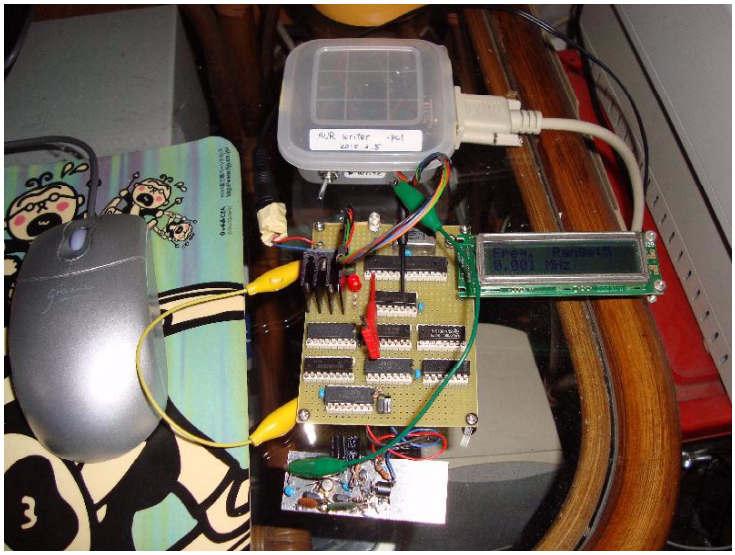
また、カウント値を周波数に変換するには、次のようになる。
測定周波数 = 分周比 ／（基準信号1周期の時間 * カウント）
= 分周比 * 基準信号周波数 ／ カウント

9.　カウンタの写真とデバッグ状態



上部にある長いICが　ATmega328P
蛇の目基板で配線

下の小さな基板が入力アンプ



上の白い箱が書込器。
SN74HC125Nが1個入っています。

作業台の近くにPCが無いため
PCの近くにカウンタとオシロを持ってきた

１０．　結果

周波数カウンタ
　測定周波数範囲　：　10Hz～75MHz
　入力感度　：　100mVpp　@　70MHz
　測定精度　：　4ppm　／　1年、常温　（水晶発振器に依存）

レシプロカルカウンタ
　測定周波数範囲　：　50Hz～300Hz
　測定精度　：　？？　　※1

※1　内部のクロックを測定した場合は全くばらつきが発生せず、mHz程度まで正しく
　　表示する。（測定周波数は781.25Hz）
　　コンピュータによる発生器（wg.exe）の100Hzを入力した場合は、入力レベを適正にし、
　　且つ入力アンプの帯域を補正した場合にはばらつきは発生せずに測定できる。
　　100Hzの分解能の理想値はおよそ0.8mHzである。
　　周波数カウンタの10秒ゲートで測定した値とは、0.1Hzまでは合っている。

１１．　感想

1）動作しました。 測定できます。
　手元にある信号を複数測定しましたが、期待の値が測定できました。
　精度についていろいろと述べてきましたが、現時点では確認しようがありません。
　基準発振器が安価な簡易型なので、その程度の性能です。
　しかし、この値段でこの精度（10-6程度）はりっぱなものです。
　OCXOのジャンク（10-8程度）がありますので、ケースに収めるようなことになったら、
　切替を考えます。
2）表示の出し方はいろいろありますので、暫定的にこのようにしてあります。
　表示を凝りだすときりがありません。
3）プログラムはBASCOM-AVRお試し版（無料）で、REM、空白を含めておよそ350行。
　プログラム容量はコンパイル時11%と表示されますので、およそ32kbyte*11%=3.52kbyte。
　いろいろな関数があって便利です。 非常にシンプルであると感じました。
4）レシプロカルカウンタも動作しています。
　以下はこの動作のときの結果です。 項目　4）～8）まで。
　自分の内部の信号を測定しますと、1/2分周時と1/32分周時で結果がわずかに違いました。
　計算誤差によるものと思われます。
5）この動作のときは、周波数が低いほどカウント値が大きくなるので、どこかで
　オーバーフローを起こして測定値がかけ離れた値になるはずですが調べていません。
　エラーチェックをプログラムに組み込みますと、これもきりがありませんので現在は入れて
　いません。（勿論余裕があれば入れるべきです）

6）難しいのは入力アンプです。 低い周波数から80MHz程までの帯域をカバーしなければなりません。 レシプロカルカウンタなど低い周波数を主に測定する場合本当はDCからの帯域を持たせないといけないのですが、今回簡易的なアンプにしています。

7）低周波で誤動作しました。 300Hz以下で測定が安定しませんでした。 入力を50Ωでターミネートしたり、アンプの出力にコンデンサを入れて高域を落としたりすると安定です。最終的にトランジスタのベースに抵抗を入れたり、コレクタにフェライトビーズをいれた後現象が出ていません。

8）7）のようなことがありましたが、最終的に10kHz程度以下の低い周波数は安定しません。入力アンプが低い周波数で波形が2つに 割れたり、細いヒゲが乗ったりしています。
入力レベルや波形によってもカウントが変わったりする場合があります。
周波数とレベルがうまく設定されると、レシプロカルカウンタもほとんどばらつき無く測定が出来ますので、カウンタ動作自体はうまく動いていそうです。

9）電圧ゲイン10倍で出力振幅2Vpp程度、F特DC～80MHzくらいのアンプは、恐らく探せば良いICがあると思いますが当分このままです。

10）メーカー製のカウンタは、オシロスコープなどと同じ様に入力にスレッシュホールドを設けたり、入力レンジ(アッテネータ）があって適性な入力範囲が選択できるようになっています。 自作の場合もできれば見習いたい。

11）AVRにはいろいろな機能がたっぷりありますので、ほんの一部を使ってみた結果になりました。 まだ、A/D変換器、D/A（PWM）、ウォッチドッグタイマ、PINの割り込みなどは未知です。 機能を使うことだけを目的として実験をするのは面白くないので、これらは必要になったときに、再び勉強することにします。

12）カウンタとしては表示が小さく見にくい。 LEDにするとハードを追加しないといけないしせっかくBasicで楽に表示できたのがむだになってしまう。 もう少し大きいLCDがあると良いのですが・・・・・
測定器の機能はこの程度でできてしまうのですが、測定器としてはもっと大きくて重いほうが良い。 測定ケーブルに引っ張られて台から落ちます。

１２．　BASCOM-AVRに関して

今回1chipCPUでAVRを選択し、プログラム言語として、BASCOM-AVRを使用しました。
その感想です。

１）BASCOMを使用して良いと感じたのは、表示や計算が楽であることでした。
　高級言語なのであたりまえなのですが、実際に使用すると有り難みがわかります。
　ハードを直接コントロールするような場合は、BASCOMの命令の仕様がわからないこと
　（私の使用方法の範囲で）があって、その場合は実際に命令の動作を確かめながら
　プログラムを作る必要がありました。

２）楽な分メモリをたくさん消費しそうです。 フローティング演算や超越関数を1行でも書くと
　数百バイト簡単に消費します。 その後はあまり増えませんので、そのような機能を使うと
　まあライブラリか何かをリンクするのだと思います。
　しかし、dB変換のためのlog などはアセンブラで処理しようとするとどうしたら良いか
　わかりませんし、フローティング演算なども同様ですので、ずいぶん有難いと思います。

３）カウンタでおよそ4kbyte消費しました。 お試し版の最大容量です。 これ以上入りません。
　でもCPU自体は64kbyteのプログラムROMを持っていますので（ATmega328P）、あと16倍
　のプログラムが内蔵できます。 正規版を買えばROM全部使えるはずです。
　現在¥12k～15kのようです。 この手のプログラムとしては高くないと思います。
　私については買うかと言うと・・・・買わないでしょうね。 4kbyte以内でこそこそと・・・・

４）Basicプログラム中にアセンブラを書き込めます。 Basicだけで事足りるかというと、今後
　恐らく必要になると思います。 Basicだけですと本当に細かい部分、たとえば割り込みや
　時間を気にする部分、ハードと密着した処理などは隔靴掻痒となります。
　まだ勉強不足でアセンブラまで手がまわりませんが、AVRを使い続けるならばいずれ必要
　になると思っています。

６）BASCOMの整数型変数は　-2,147,483,648 ～ 2,147,483,647　の制限があります。
　精度をなるべく保つために、乗除では掛け算を先に行います。
　定数などを被除数にした場合は、数割り算を行う前に範囲ぎりぎりまで桁をおおきくして
　計算します。
　また、割り算を行う場合に商と余りを使って、小数点以下を計算してゆく方法もあります。
　桁数のダイナミックレンジが不足の場合に有効です。

７）ソフトでカウンタを作る場合、プログラムの実行時間のばらつきを考慮しないと精度が悪く
　なる可能性があります。 たとえばプログラムでカウンタのゲートを開閉するような場合、
　ゲートを閉じようとした時にインタラプトが重なると、インタラプトが優先されてゲートを閉じる
　タイミングが遅れ、結果的にゲート時間がインタラプトの処理の分長くなるような事です。
　この不良の検出は難しく、実際には
　・ほとんどの場合は正しい測定値となる。
　・ある周波数で測定値の誤差が大きくなる。 どんな周波数でその不良現象が発生するかは
　　実際に測定してみないとわからない。
　・誤差が発生しているとわかるくらいの値のずれであれば測定が正しくないと認識できるが、
　　たとえば数十MHzで数十カウントの誤差などというと、誤差が発生していること自体が
　　わからない。

こんなことがおきます。 これを検証するのは大変です。 実際どうしてよいか解りません。設計時に良く考えて、考える範囲内では絶対に不良は発生しないという回路、方式を採用しなければいけません。

8）AVRは無料の“C”言語もありますが、“C”は慣れていないのと個人の趣味ですがいまいち好きになれません。 主流は“C”であることは間違いないのですが、またやって出来ないことはないとは思いますが・・・・

１３．　AVR　書き込み器（ハードウェア）について

１）書き込み器はシリアルポート、SN74HC125Nを1個使用したタイプです。

２）他に抵抗4本で出来たタイプがあります。 これは2種類あって、SCKが2ピンのもの(A)と、5ピンのもの(B）があります。

(A)タイプは、書き込みソフトavrsp.exeで書き込めるはずであるが不可。 理由は不明。

(B)タイプは、BASCOM-AVRから直接書き込みが出来る。 avrsp.exeで書き込めるかは調査していない。

最初2）を作ったがよくわからなくて、１）の書き込み器を作り、その後２）は使っていません。
avrsp.exeは有名なAVRの書き込みプログラムで、インターネットを見ていますと各所に出現します。 ちょっと検索すればダウンロードできるサイトが見つかります。

１３．１　AVR　書き込み方法

１）avrsp.exeを入手し、適当なディレクトリに置きます。

２）BASCOM-AVRでプログラムを作成し、コンパイルを行うと、xxx.hexの拡張子が付いたファイルが作成されます。

３）（ア） DOSのコマンドプロムプトを開きます。 コマンドプロムプトはWindowsXPではアクセサリの中にあります。　ショートカットを作っておくと便利。

（イ） １）のavrsp.exeをコマンドプロムプトの窓にD&Dします。 必要に応じてスペースをキーインすること。

（ウ） -pc1をキーインします。 これはcom1を使用するためのコマンド。
avrsp.exeとおなじディレクトリにavrsp.iniを作ってその中に-pc1と書いておくと、このコマンドは不要です。（avrspの取説参照）

（エ） スペースを入れた後、xxx.hexをコマンドプロムプトの窓にD&Dします。

（オ） 書き込み器のSWを、writeの位置ににしてから、エンターkeyを押します。

（カ） これでDevice名が表示され、最後にpassが表示されれば書き込み完了。

（キ） 書き込み器のSWを戻せば、reset-startします。

４）デバッグを連続的に行う場合はコマンドプロムプトを閉じないで、2)のコンパイルを行った後に、コマンドプロムプトに移動して、F5を押すと直前の命令が表示される。
もし前回の書き込み操作後、コマンドプロムプトに何もキーインしていなければ、
SWの操作とエンターキーで次の書き込みを行うことが出来る。

１３．２ ヒューズの書き換え

外部クロックを使用する場合は、ヒューズビットを書き換えなければなりません。
また、水晶発振子やセラミック発振子を使用する場合も同様です。
書き換えはavrsp.exeのコマンドを使用。 方法は同取説参照。
ヒューズバイトの下位にビットが含まれているのでそれを書き換えます。
イニシャル時は0110　0010となっている。 これを1110　0000に変更することによって
外部クロック・クロック分周　“1”　となってCPUその他が外部クロックの周波数で動作します。

１４．　プログラムのデバグ方法

プログラムを作って、ROMに書き込んで、ハードウェアを動かすわけですが、1回で動くことはありません。（少なくとも私には） なのでプログラムの不具合を見つけなければなりません。
そのためには次の方法があります。
１）プログラムリストを一所懸命見て、順番にプログラムを追ってゆき、不具合を見つけます。
２）BASCOM-AVRはシミュレーションの機能があるので、それを使用してデバグを行います。
３）実際に動作させながら、変数の値などをチェックして意図に沿った動きをしているか確認。

１）は初期の段階ではかなり有効です。 プログラムを作ったときは問題ないと思っていてももう一度（あるいは何度も）見直すと、実は落とし穴があったりします。
２）は私の場合は時間がかかって、途中でやめてしまいました。 CPUの速度のせいでしょうか。ハードがいらないので、プログラムさえあれば動作させて見ることが出来ます。
３）LCDディスプレイが付いていないとうまくゆきませんが、ついていればプログラムの要所に変数の値を表示させる命令を追加して値を確認できます。
また、使用していないポートを出力にしてパルスを出すようなサブルーチンを作っておいて、プログラムの必要な場所にそのサブルーチンをコールするように命令を追加して、オシロスコープなどで観測すれば、プログラムのその部分を実行したかどうかがわかります。
同様に、そのパルスをトリガにしてオシロスコープで関係各所の波形を観測できます。

このような方法でプログラムをデバグしてゆきます。
特筆すべきことは、プログラムの変更がとても容易なので3）のようなことがどんどんできます。
プログラムのコンパイル=>書き込み　が15秒もあれば終了しますので、プログラムを変更して確認し、元に戻して次の部分をチェックということを繰り返して行ってもそんなに時間がかかりません。
また、ROMの書き換え回数も10000回以上もあるということで、気にしなくても良いようです。

ただし注意しなければいけないことは、うっかりデバグに関係ない部分をまちがって修正して気が付かなかったり、元に戻すときにミスをしないように等、細心の注意をしながら行わないとひどい目にあいます。

このようにしてデバグを進めてゆきます。

以上