



どすぶい
だに☆
第3号

This Page is Blank.

もくじ

ページ

光学距離計の実験と製作	ChaN	2
電光掲示ファンの製作	ChaN	13
FTD232を用いた USB接続AVRマイコンライタの試作	おゆう	16
マクロクールライトみたいなものをつくる	ROM男	20
デジタルオーディオアンプの試作	Kim	28
MPROM制御ソフトウェアと ハードウェア開発の手順と解説	もろぼし☆らむ	30
変更履歴		39
あとがき、奥付		40

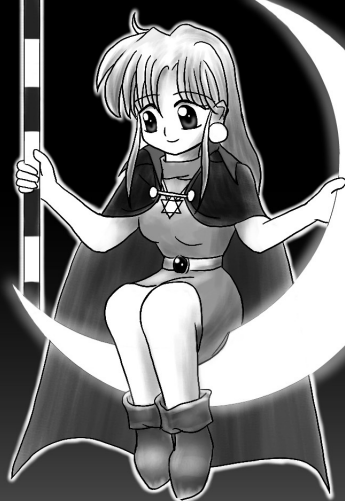


表紙:もろぼし☆らむ

裏表紙:もろぼし☆らむ

光学距離計の 実験と製作

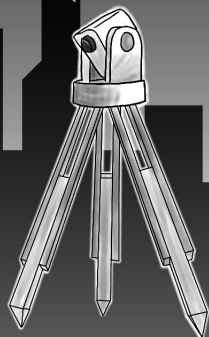
本文・挿絵: ChaN



???



PiPi...
380,000km



自作の距離計といってもいろいろな方式がありますが、製作例としてはマイコン入門によくあるエコー式超音波距離計が代表的です。超音波測距は仕事でもいろいろ開発していて既に飽きが来ていますので、今回はちょっと上を狙って光学測距に挑戦してみました。光学距離計なんてあまり実用性のある製作ではありませんが、以前から興味があって一度やってみたかったというのと、趣味でこんなモノを自作する奴なんてあまりいないだろうという、ハム的チャレンジ精神(笑)からで、今回意を決してプロジェクトに踏み切ったわけです。まあ、論理的に動いて当たり前というデジタルな世界にも飽きてきたことですし。ということで、かなりマイナーな分野なので関連資料も乏しくて、最初から手探り状態。まして光学的ループを含んだ計測システムの自作なんて初めてなので、果たしてどうなることやら。

光学測距の方式

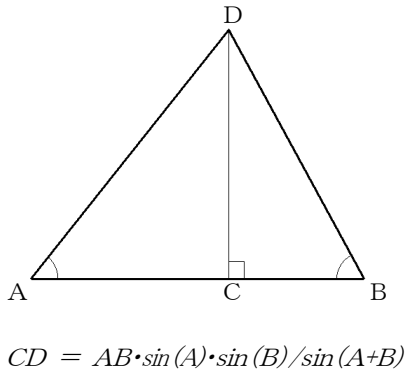
光学測距は主に、幾何学的手段で距離を求める三角法と、光速を元に距離を求める TOF(Time of Flight)法に分類できます。

●三角法

光学的手段(像の合致や、測距光のスポット位置検出)

で基線と対象物のなす三角形の頂点を求めて、三角比の性質を利用して距離を算出する方法で(図1)、いわゆる三角測量の原理です。機械的動作を伴うものでは軍用測距儀やコンパクト・カメラの自動焦点機構として昔からよく用いられてきました。産業用としては、数cmから1m程度までの短距離測定の距離センサー・モジュールなどがこの方式をよく利用しているようです(図2)。

図1. 三角測量

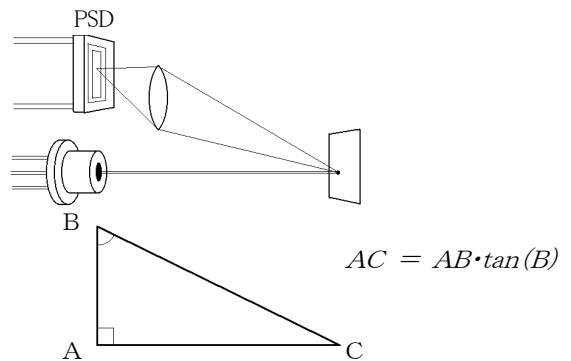


●TOF法

光の進行する速度は真空中で約 $2.998 \times 10^8 \text{m/秒}$ の一定の値を持っています。この性質を利用して、対象物との間を光が往復する時間から対象物までの距離を測定します。TOF方式は時間差測定方法の違いでさらに2種類の方式に分かれます。超音波測距と同様に、パルス光を送出してそれが反射して戻ってくるまでの時間を直接測る「**エコー方式**」(図3)と、変調光を送出して反射光の変調との位相差から伝搬時間を求める「**位相差方式**」(図4)があります。さらに、レーザー光の干渉を利用した微小変位計もあるようですが、一般的でないのでここでは取り上げません。

エコー方式は測距のとき一発だけパルス光を出せばよく、単発ならパルス・レーザーで強力な光を出せるので、自然反射で1km以上の測距が可能です。しかし、光速を

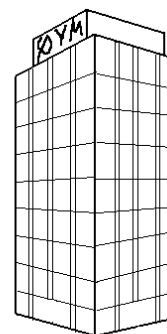
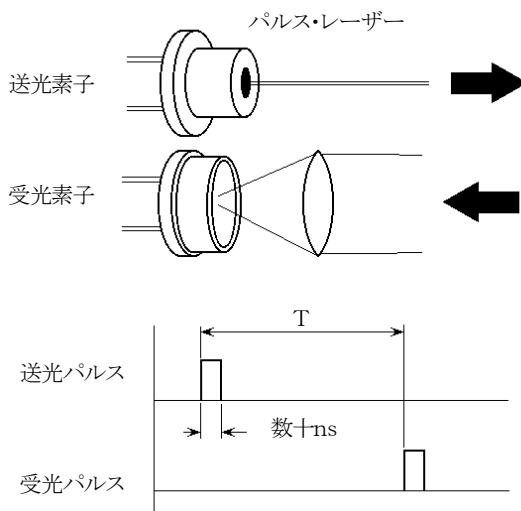
図2. 距離センサ・モジュールの例



直接測ることになるので、あまり高い分解能は得られません。例えば、1mの分解能で距離を測定するには、150MHzの基準クロックが必要になり、一般に市販されているレンジ・ファインダーでも1m程度の分解能になっているようです。

位相差方式でも必要な時間分解能はエコー方式と変わりませんが、後で解説する信号処理で容易に分解能を高めることができます。これにより、位相差方式ではミリ単位での測距が可能になっています。測距時はある程度の時間(または連続で)測距光を出していなければならないので、保安上や消費電力の点であまり測距光出力を上げられず、最大距離は自然反射で数十m、リフレクター(コーナーキューブ・プリズムや再帰反射板)を使用して数km程度が多いようです。今回はこの位相差方式を採用してみました。

図3. エコー方式



$$\text{距離}[m] = T[\text{sec}] \cdot 3 \times 10^8 / 2$$

図4. 位相差方式

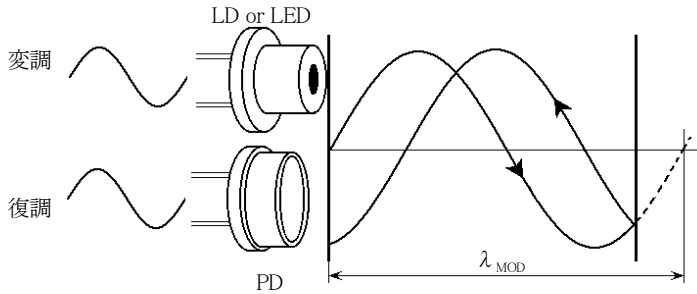
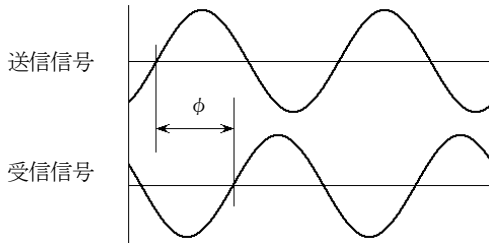


図5. 位相の遅れと距離



$$\text{距離}[m] = \phi [\text{rad}] / 4\pi \times \lambda_{\text{MOD}}[m]$$

位相差方式の原理と実際

●距離と位相差

位相差方式では、一定の周波数 f_{MOD} の信号(以下、測距信号)で AM 変調された測距光を送受信します。測距信号のキャリアである光は光速で伝搬するので、対象物間における測距信号の波長 λ は、

$$\lambda_{\text{MOD}}[m] = 3 \times 10^8 / f_{\text{MOD}} [\text{Hz}]$$

となります。光速は有限であるため、対象物で反射して戻ってきた測距信号の位相は、図5のように距離に比例して遅れ、 $\lambda/2$ の距離で送信端から位相が 2π [rad] 遅れて戻ってくることになります。この送信端と受信端での測距信号の位相差 ϕ を測れば、図5中の式により、対象物までの距離を求めることができます。これが位相差方式の原理です。

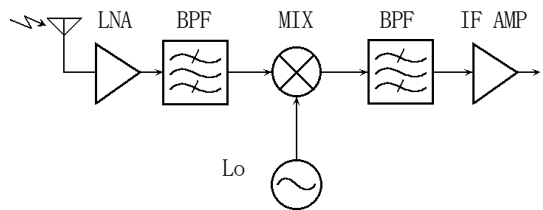
●位相の精密測定

距離に比例して生じる測距信号の位相遅れの実時間は、エコー方式での往復時間と同じなので、直接 mm 単位までカウントするには、150GHz という非現実的な基準クロックが必要です。したがって、受信信号の位相を直接測っていたのでは十分な分解能で測定することが困難で、同じ基準クロックではエコー方式と変わらなくなってしまいます。そこで何らかの方法で位相測定の時間分解能を上げる工夫が必要になります。

それを実現する方法の一つとして、**周波数変換**が用いられています。ラジオ受信機でお馴染みのアレですね

(図6)。周波数変換では、位相情報を保ったまま周波数だけを変換することができるのです。低い周波数に変換すれば、それだけ高い分解能で位相を測定できるようになります。ほかに位相差を直接測定する方法(直交変換方式とか?)でも実用化されているかも知れませんが、探した範囲では見つけれませんでした。

図6. 受信機の周波数変換の例



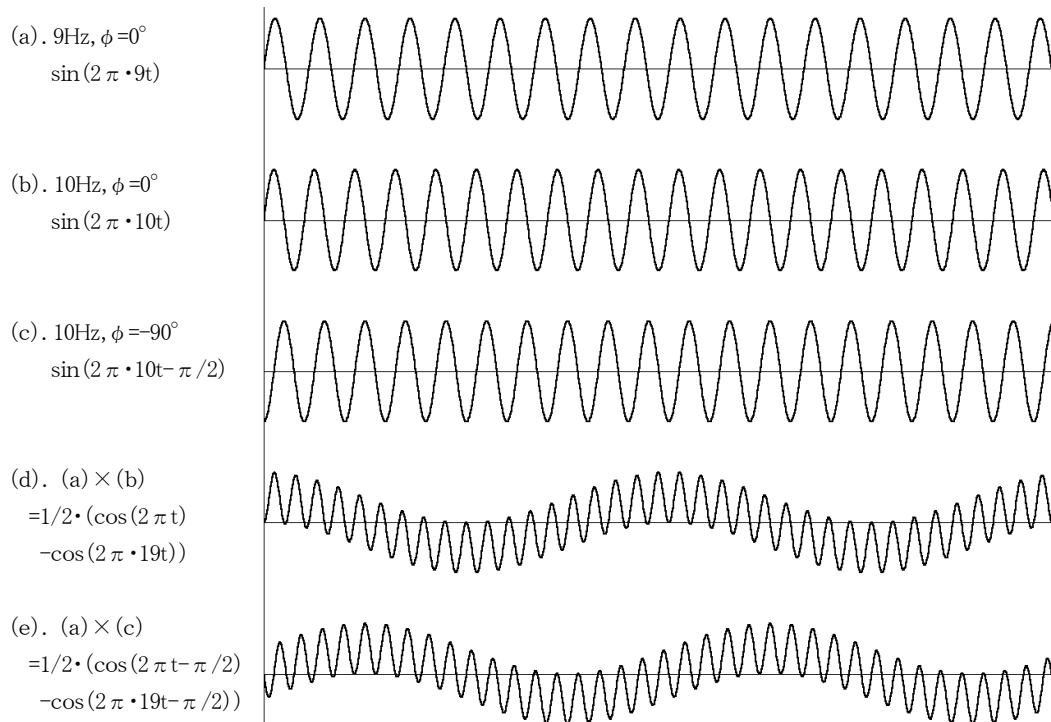
さて、2つ異なる周波数の正弦波信号 f_1 と f_2 をかけ合わせると、その和と差の周波数 $|f_1 - f_2|$ 、 $f_1 + f_2$ の信号が生成することが知られていて、周波数変換はこの性質を利用しています。ミキサーに用いられる乗算回路には、ダイオードやトランジスタの非線形特性を利用したもの、ダイオードのスイッチング動作を利用したDBM、ギルバート・セルを応用した半導体DBMやアナログ乗算器などいろいろありますが、どれも結果的には2つの信号を掛け合わせる信号処理になっています。

ここで、2つの信号 f_1 、 f_2 をそれぞれ $A_1 \cdot \sin(\omega_1 t + \phi_1)$ 、 $A_2 \cdot \sin(\omega_2 t + \phi_2)$ として、周波数変換を数式で表すと式1のようになります。

式1. 周波数変換

$$A_1 \cdot \sin(\omega_1 t + \phi_1) \times A_2 \cdot \sin(\omega_2 t + \phi_2) = \\ 1/2 \cdot A_1 \cdot A_2 \cdot (\cos((\omega_1 - \omega_2)t + \phi_1 - \phi_2) - \cos((\omega_1 + \omega_2)t + \phi_1 + \phi_2))$$

図7. 周波数変換と位相の例



Aと ϕ は、それぞれの信号の振幅と初期位相です。この式から、入力信号の位相の項目が周波数に影響されずに加減算されて、変換出力 $|f_1 - f_2|$ 、 $f_1 + f_2$ の位相の項目にそのまま現れているのが分かります。数式では分かりにくいかも知れないので、周波数が同じで位相の異なる2つの信号をそれぞれ周波数変換した結果をプロットしたのが図7で、これなら直感的に分かるといえます。

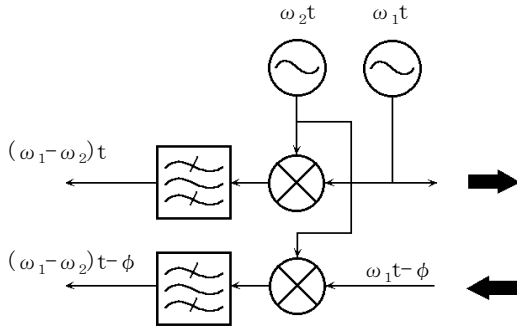
ここで、図7に示す波形を上から順に、(a)周波数変換の為の局発信号、(b)送信測距信号、(c)受信測距信号と考えてみます(周波数は見やすいよう適当に決めています)。(a) $\times$ (b)、(a) $\times$ (c)の変換出力(d)、(e)の波形から、それらは入力信号の和と差の周波数成分を含んでいることが読みとれ、さらに元の信号(b)、(c)の位相差が、生成されたそれぞれの和差成分の位相差に反映されているのが分かります。あとは(d)、(e)からフィルタで差の成分だけ取り出してそれらの位相差を測定すれば、元の信号を直接計測するのに比べ、同じ基準クロックでも変

換比の分だけ分解能を上げることができるのです。

●位相差方式のシステム構成

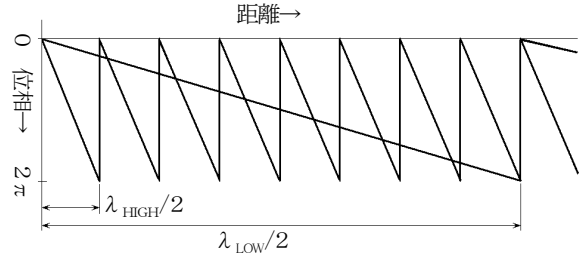
図8に位相差方式の原理ブロック図を示します。送信信号と受信信号をそれぞれ周波数変換して、位相差を測定しています。距離が $\lambda_{MOD}/2$ になると位相がラップ・アラウンドして戻ってしまうので、絶対測距が可能な範囲は、 $\lambda_{MOD}/2$ までです。測距レンジは λ に比例するので、レンジを広げるには測距信号の周波数 f_{MOD} を低くすれば良いことになります。しかし、その測距レンジ全体が測距信号の位相差 $0 \sim 2\pi$ に対応するので、一定の距離分解能で測距レンジを伸ばそうとすると、それだけ高い位相分解能が必要になってしまいます。だからといって変換比を大きく取りすぎると、局発信号や測距信号の位相ノイズまで拡大(相対的に)されてしまって、精度良く測れなくなってしまいます。そこで、一般的には測距信号周波数を切り替えて、低い変調周波数 f_{LOW} で大雑把な絶対位置

図8. 位相差方式での周波数変換の考え方



を出して、高い変調周波数 f_{HIGH} でその付近をさらに細かく測るとい方法がとられているようです(図9)。

図9. 位相差方式の測距レンジ切り替え



実験回路の仕様と設計

図10に今回の実験回路のブロック図を示します。周波数変換など、基本的なところは原理で示したところと同じですが、冗長な部分を削って単純化するためいくらか変更を加えています。

●基準信号の生成

まず、測距信号の周波数と、位相比較のために変換する周波数を決めなければなりません。今回は、測距信号周波数 f_{MOD} を 12.5MHz 単一とし、これを 4kHz に周波数変換して位相を測ることにしました。変換比は 1/3125 です。測距レンジは $\lambda_{\text{MOD}}/2$ より、12m までとなります。これらの周波数は、50MHz のマスター・クロックから生成します。位相差測定の基本クロックにもマスター・クロックを使えるので、位相分解能は、 $4k/50M = 2\pi/12500[\text{rad}]$ となり、約 1mm の距離分解能となります。まあ、実験なのでこの程度で十分でしょう。

それと、もう一つ忘れてはいけなのが局発信号で、測距信号を 4kHz に変換するため、測距信号から 4kHz だけずれた周波数が必要になります。通常、このような周波数比のオシレータは特注になって入手困難ですし、周波数差を正確に保つのも困難です。

そこで、局発信号はPLLを使って生成するようにしました。VXO を 12.496MHz 付近で発振させて、D-FFでこれと 12.5MHz の測距信号との差をとり、これがマスタークロックを 3125 分周した 4kHz の基準信号と位相同期するようにフィードバックをかけます。局発信号の位相ノイズやスプリアスが大きいと変換出力の位相特性に影響する

ので、PLL部は他の回路からの干渉を受けないように電源ライン等に十分アイソレーションをとるなどの配慮も必要です。

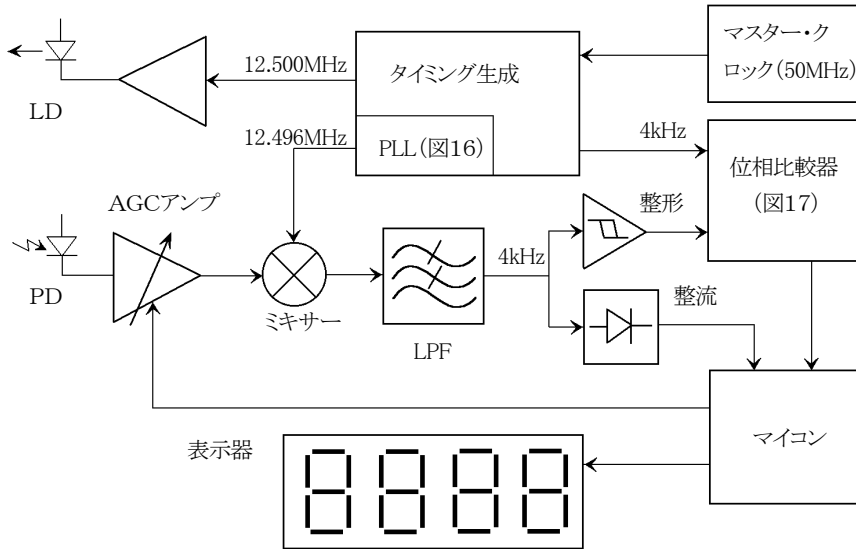
●送信部

測距光の光源には、通信用赤外線LEDまたは半導体レーザーが用いられていますが、昔からあるCD用のレーザー・ダイオードは 780nm の不可視光なので、危険で実験には向いていませんし、通信用赤外線LEDは安全ですが入手が困難です。この実験回路では、DVDの普及によりジャンク屋で手軽に入手できるようになった 650nm のレーザー・ダイオードを使用しました。レーザー・ダイオードは元々高速応答なので、この程度のもので軽く 100MHz 以上での変調が可能です。650nm は可視光で視認性が良く、光軸合わせなどの扱いが容易なので光学実験にはオススメです。

●受信部

対象で反射して戻ってきた測距光は、レンズで集光してPINフォト・ダイオードで光電変換して増幅します。受信される測距光のパワーは、対象までの距離やリフレクターの状態などで大きく変動するので、アンプのゲインが固定ではアンプ出力が飽和したり、必要な信号レベルが得られないなど使い物にならなくなります。実用機では回転濃度フィルターや機械絞りなどの光学アッテネーターで光量を調整しているようですが、必要な光学部品が簡単には入手できなかったため、今回はアンプにAGCをかけて対応することにしました。アンプのゲインを変えると位相に何らかの影響を及ぼす可能性がありますが、この

図10. 実験回路の機能ブロック図



辺は実際にやってみて確認するしかありません。その結果は後ほど...

●周波数変換と位相差測定

必要なレベルまで増幅された測距信号は、ミキサーに入力されて周波数変換されます。ミキサー回路には素直に乗算ICを使用しました。ミキサー出力に現れる和差成分(24.496MHz と 4kHz)を LPF を通して差の成分(4kHz)だけ取り出して位相比較回路に送ります。

ところで、原理図に二つあったミキサーが実験回路には一つしかありません(実用回路でも一つだけです)。まず、原理図では二つの測距信号の位相差が絶対距離を表しているように書いてありますが、実際には変復調・増幅等の過程で生じる不確定な位相シフトによるオフセットが加わって、それらの信号の位相差の絶対性は失われています。だから、比較のための基準位相にも絶対性は必要なく、ある決まった値であれば良いわけです。それならわざわざミキサーで生成しなくても、変換信号を生成するときの 4kHz の基準信号が受信信号の位相測定の基準としてそのまま使えます。

●絶対距離の出し方

得られた送受信信号の位相差が、不確定なオフセット ϕ_{OFS} を含んでいて絶対的ではないとなると、測距対象の絶対距離を出すには、さらに何らかの絶対的な基準が必要になってきます。実際のシステムでは、図11に示すように光学系に内部パスを設けて機械的に光路を切り替え、

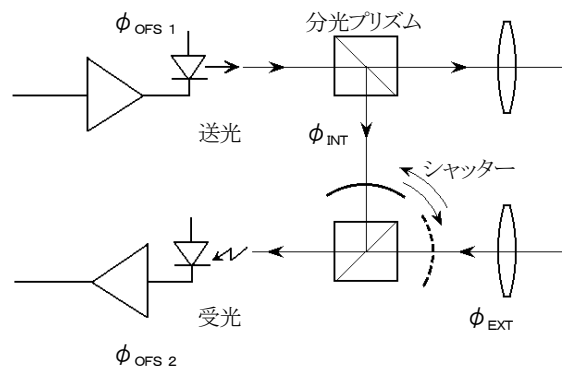
内部パスでの位相($\phi_{\text{INT}} + \phi_{\text{OFS}}$)と、外部パスでの位相($\phi_{\text{EXT}} + \phi_{\text{OFS}}$)の位相差を出しています。これにより、

$$(\phi_{\text{EXT}} + \phi_{\text{OFS}}) - (\phi_{\text{INT}} + \phi_{\text{OFS}}) = \phi_{\text{EXT}} - \phi_{\text{INT}}$$

となり、内部パス(既知の距離)と外部パス(測距対象)の距離の差が確定し、測距対象までの絶対距離が出ます。回路での位相シフト ϕ_{OFS} は温度変化等でドリフトするので、内部と外部の比較動作は測定の都度行われます。

しかし、この実験システムではそういった機械的な基準機構を設けていないので、単体では絶対測定ができません。このため、既知の距離を測って測定値を手動リセットしてから対象距離を測るということでお茶を濁しておくつもりです。まあ、原理実験だからこの程度でも十分ですけどね :-)

図11. 実用機の内部基準機構



実験回路の製作

さて、いよいよ実際の回路の製作に入ります。回路自体は比較的シンプルですが、今回は微少信号を扱うアナログ系が多いので、実装上のノイズ対策はちゃんとやっておく必要はあります。レイアウト上の都合やノイズ対策のため、主にブロック毎に3つの基板に分けて組みました。使用した部品は、どれもそれほど特殊なものではないので、マイコンとPLD以外は全て秋葉で揃いました。

●送信基板

測距信号をキャリアであるレーザー光に乗せて送信するブロックです(図12、写真1)。単純に変調光を出せばよいので、回路的には普通のAPC回路に変調回路を付け足しただけになっています。レーザー・ダイオードは、ジャンク屋で見つけた DL3147-011(三洋)を使用しました。

一般的なレーザー・ダイオードの順方向電流(I_F)－光出力(P_o)特性の典型は、図13に示すようになっていて、順方向電流がスレッシュホールド値(I_{TH})を越えるとレーザー発振が始まります。 I_{TH} は温度によって変化するので、定電流駆動すると光出力を一定に保てず、発振しなかった

り過大出力で壊れたりします。このため、CW(連続発振)用途ではAPC(Automatic Power Control)回路で光出力をフィードバック制御して使用するのが常識です。

この回路では通常のAPC回路の出力電流に変調信号(約 5mA_{P-P} の矩形波)を直接重畳しています。APC回路の制御帯域は変調周波数より十分低いので、光出力の平均バイアス点は一定に保たれます。L1 は原理的には要りませんが、トランジスタのコレクタ・インピーダンスが変調周波数で意外に低かったので、変調電流をブロックするために入れました。

半導体レーザーの放射角はとても広いので、開口率の大きい凸レンズでコリメート光にしてやる必要があります。コリメート・レンズは、CD/DVD の光学ピックアップやレーザー・ポインタから取れます。送信基板は写真のように受光レンズAssyと一体にしておくとうまく光軸合わせがやりやすいです。

●受信基板

受信した測距信号を増幅して周波数変換するブロックです(図14、写真2)。このブロックで処理する信号は非常に微弱なので、基板をシールド・ケースに入れて外来ノイ

図12. 送信基板の回路図

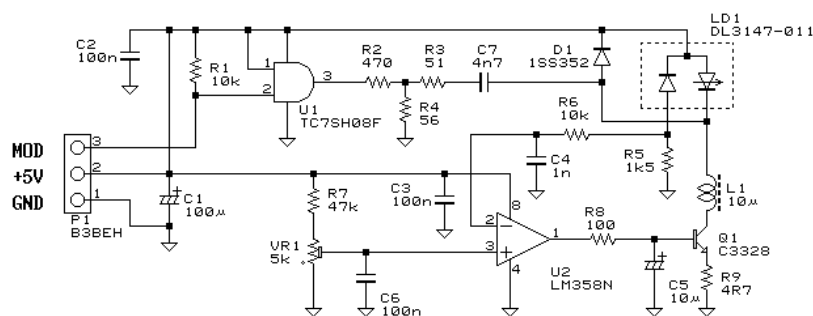


図13. レーザー・ダイオードの I_F － P_o 特性と変調

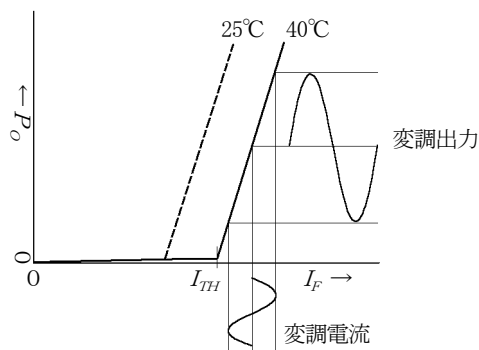


写真1. 送信基板と受光レンズAssy

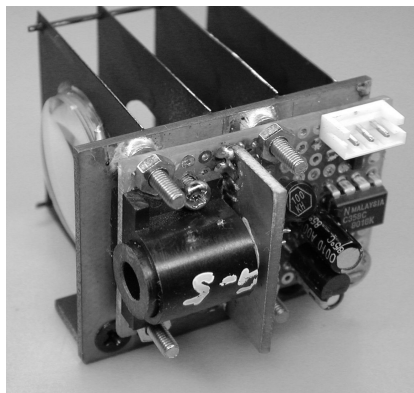


図14. 受信基板の回路図

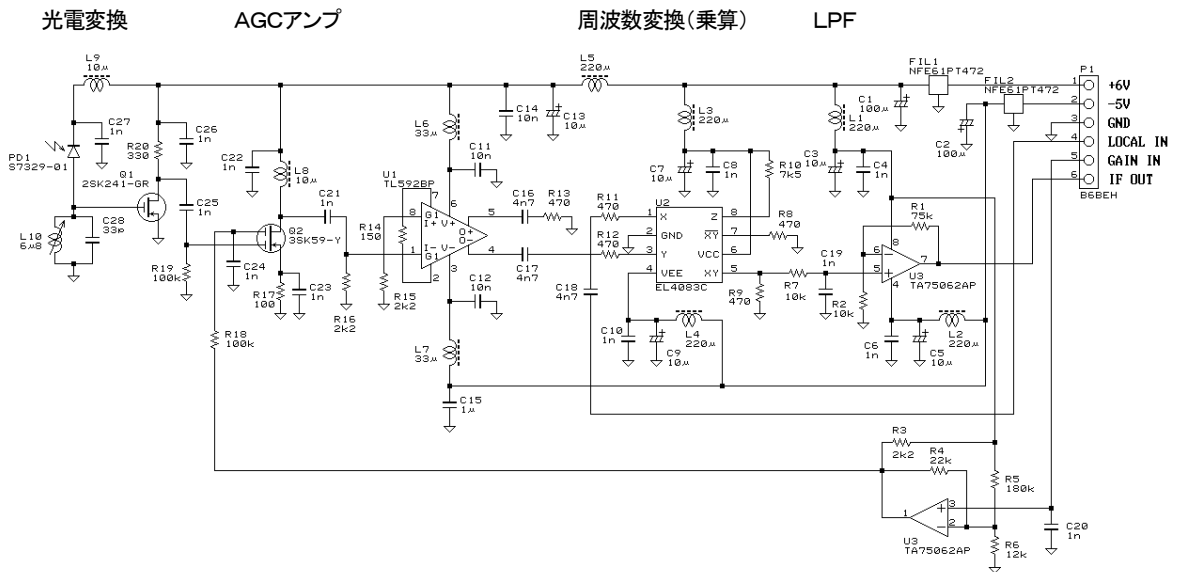
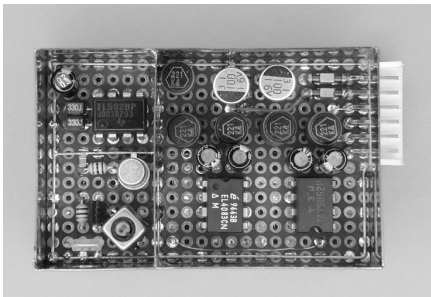


写真2. 受信基板

部品面



半田面

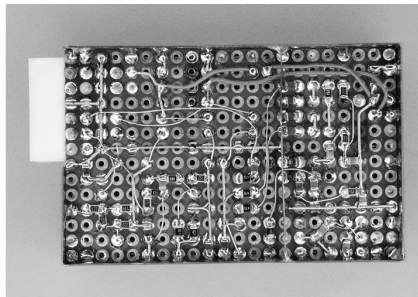
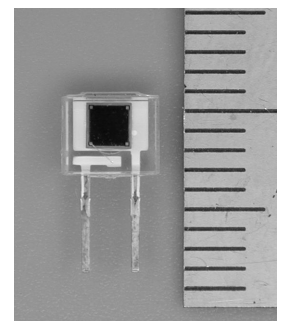


写真3. S7329-01



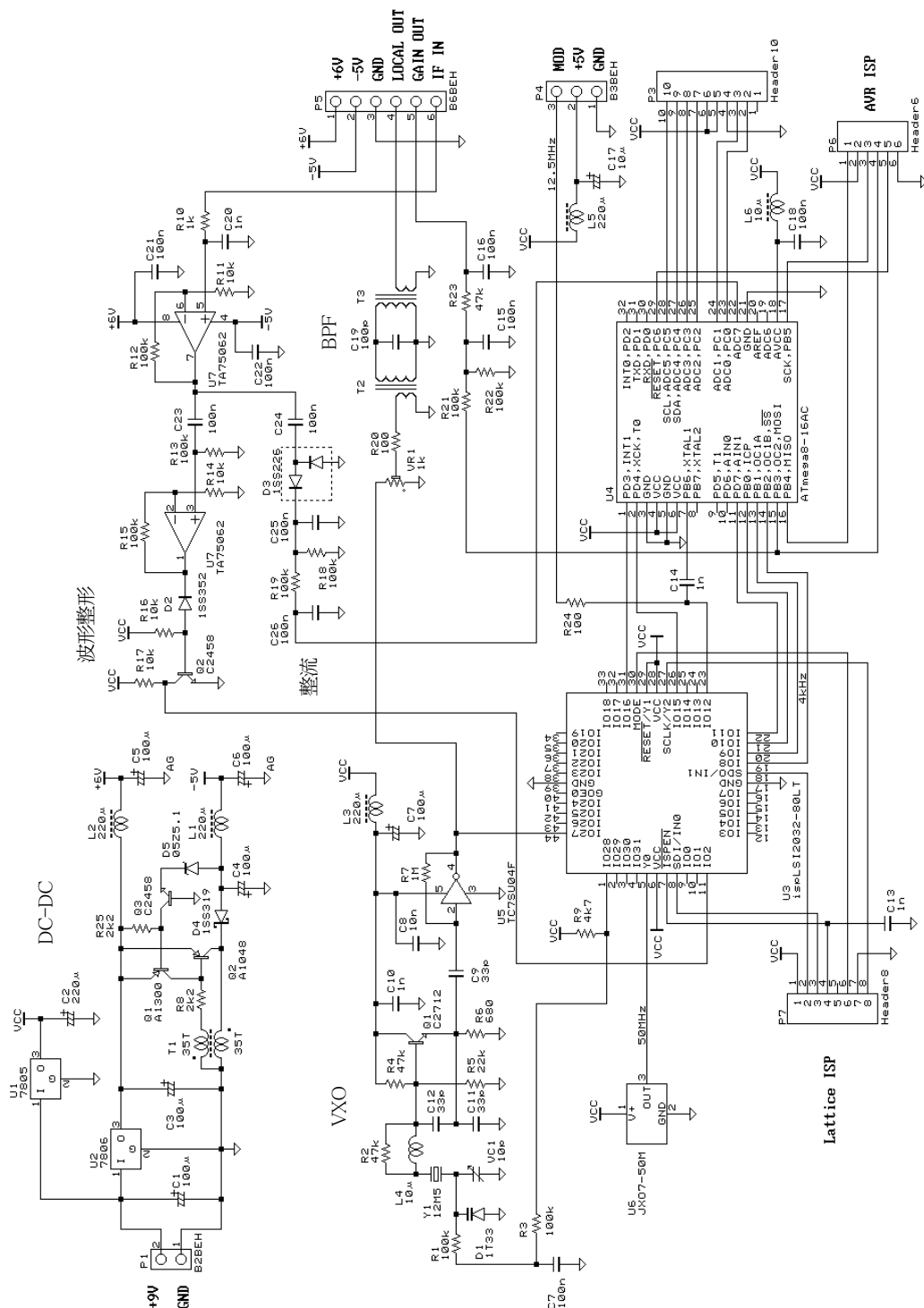
ズを防止します。ランダム・ノイズはフィルタ処理である程度取り除けますが、測距信号に相関性のあるノイズ(送信部からの回り込みなど)が飛び込むと、受信信号の位相が回転してしまって精度を悪化させてしまいます。たとえば、送信部から受信レベル比で -40dB の回り込みがあったとすると、 $\pm \tan^{-1}(0.01)$ の位相エラーとなり、これはフルスケールの $\pm 0.16\%$ に相当します。できれば 60dB 以上の分離は確保したいところです。シールドなしでは、ゲインを上げたとき送信部からの回り込みを拾ってしまって使い物になりませんでした。

写真3が光電変換に使用したPINフォト・ダイオードです。このダイオードは受光エリアが広い($2 \times 2\text{mm}$)割に、 60MHz の帯域があるので、今回の用途には最適です。PINフォト・ダイオードはセオリー通り逆バイアスして、さらに負荷抵抗の代わりに同調回路を使って、不要信号をあ

る程度排除するようになっています。

測距信号は Q1:2SK241、Q2:3SK59、U1:TL592B を経て約 $400\text{mV}_{\text{P-P}}$ まで増幅されます。このうち 3SK59 でトータル・ゲインを制御しています。FETの G_m がドレイン電流に依存することを利用した典型的なゲイン制御回路ですね。ゲイン調整比は 40dB 以上が得られました。最初は共振回路を 3SK59 の G1 に直接接続していたのですが、G2 電圧を変えてゲインを調整すると内部容量が変わってしまうようで、かなりの位相シフトが発生してしまいました。そこで、共振回路との間に Q1 を入れて方向性を持たせて、ついでにゲインも稼いでいます。コントロール基板からのAGC電圧($0 \sim 2.5\text{V}$)は、U3 で $-0.5\text{V} \sim 5.5\text{V}$ に変換されて 3SK59 の G2 に加えられます。シングル・ゲートFETの 2SK241 等でもDCバイアスを調整して動作点を変えれば同様にゲイン制御ができます。

図15. コントロール基板の回路図

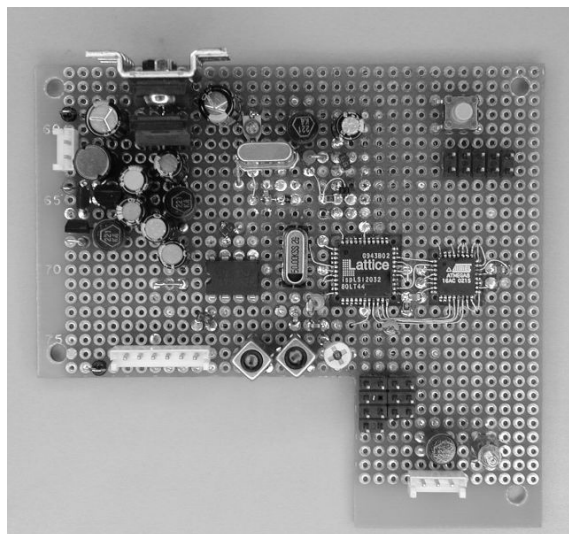


TL592B はオープン・ループの広帯域ビデオ・アンプで、主に磁気ヘッドのプリ・アンプとして使われているものです。ゲインを 400 倍に設定したとき 50MHz の帯域 (GBW=20GHz に相当)があるので、広帯域・高ゲインの必要な用途に最適です。この回路では、60 倍の設定で

使用しています。

一定の振幅に増幅された測距信号は、アナログ乗算器 (U2:EL4083C) で 4kHz 低い局発信号とかけ合わせます。EL4083C は、通信用ミキサーIC (MC1496 など)よりも高価ですが、電流出力でダイナミック・レンジも広く、信号処

写真4. コントロール基板



理用として使い道がととても広いです。生成した和差成分は、LPFで差の成分を取り出してU3で約400mV_{PP}に増幅したあとコントロール基板に送られます。

●コントロール基板と表示基板

図15、写真3にコントロール基板を示します。コントロール基板には、マイコン、位相差カウンタ、基準信号発生回路と電源回路が実装されます。マイコンとPLDには手持ちのATmega8(アトメル)とispLSI2032(ラティス)を使用しました。これらは、H8やXC9500などほかの適当なもので代替できるでしょう。

基準信号発生回路で生成する必要がある周波数は、測距信号(12.5MHz)と、それから4kHzずれた局発信号(12.496MHz)、そして位相基準信号(4kHz)です。12.5MHzと4kHzは50MHzのマスター・クロックを元に生成できますが、12.496MHzはPLLで生成しています。生成された測距信号は送信基板へ、局発信号はBPFで高調波成分を除いてから受信基板に送られます。

受信信号の振幅が変わると波形形成などで位相に影響を及ぼすので、常に一定に保つ必要があります。受信基板からの周波数変換された測距信号は整流されてマイコンのADC入力に入り、マイコンは信号レベルが一定になるようにAGC電圧を制御します。

測距信号は同時に矩形波に成形されて、位相差カウンタで位相基準信号との位相を測定します。位相差測定ロジックはPLD内部に構成しています(図16)。50MHzのマスター・クロックでカウントしているので、位相差を12500分割で測定できることになります。

図16. 12.496MHzの生成

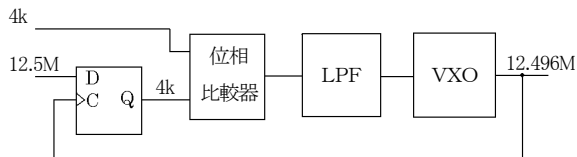
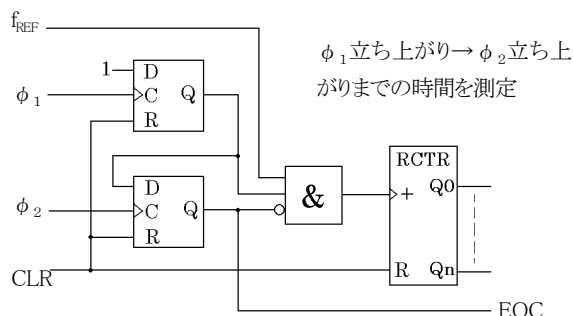


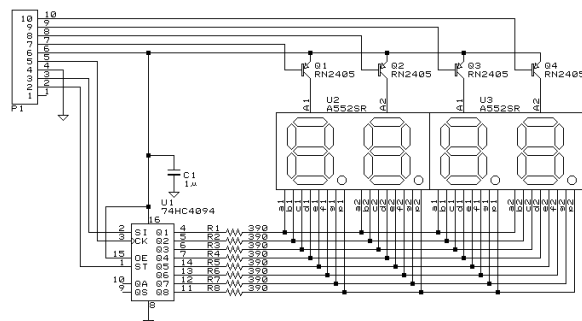
図17. 位相差カウンタ



電源回路はノイズ対策のためアナログ系とデジタル系を分離しています。が、アナログ負電源の生成にスイッチング・レギュレータを使ったりしているので、分離した意味がありません(´_`;)。仕方ないので、フィルタを入れてスイッチング・ノイズが伝わるのを抑えるようにしました。

表示基板(図17)は、メイン基板の上に乗る子亀基板で、ただのLED表示器です。マイコンのポートが足りないときの常套手段ですが、シフト・レジスタで出力を拡張しています。

図17. 表示基板の回路



ソフトウェアの動作

ソフトウェアの動作はとても単純なので、あまり書くところはないです。基本的にAGC動作と位相差測定がメインになっています。

●AGC

受信基板から来る周波数変換された受信信号のレベルは、式1からミキサー入力レベルに比例するので、これが一定になるように制御すれば全体のレベルを一定にできることになります。マイコンは、PWM出力で受信基板へのAGC電圧を調整してADC入力が一定になるようにフィードバック制御します。PWMや整流のリプルを抑えるため、ループ内にLPFが何段も入っていて位相の遅れが大きく、追従性よく制御するにはサーボ・パラメータのチューニングが必要でした。

実用機ではアンプのゲイン調整ではなく、光学アッテネータを駆動して光入力レベルを調整しています。

●位相差測定と距離の算出

位相差は複数回取得して移動平均でランダム・ノイズを排除します。カウンタに得られる値は $0 \sim 12499$ ($0 \sim 2\pi$ [rad]) の範囲ですが、位相には始点・終点というものがなく数値はラップ・アラウンドするので、演算の際はその点を考慮する必要があります。測距対象の距離は、基準距離での位相と対象距離での位相の差を元に算出します。測距信号の周波数が 12.5MHz ($\lambda/2=12\text{m}$) で、位相差分解能が 12500 カウントの場合、

$$\text{対象距離}[m] =$$

$$12 \times \text{位相差カウント数} \div 12500 - \text{基準距離}[m]$$

となります。

実用機では光路を切り替えて基準位置の位相を自動的に取得しますが、本機ではそのような自動基準機構がないので適当な基準位置を SW1 で記憶して、そこを基準に測ることになります。

結果と課題

結果から言えば、一勝一敗ってところでしょうか...:-)。AGC動作と位相差測定は設計通りうまくいきました。しかし、AGC量による位相回転は、やはり発生してしまいました。これによる測距誤差が 2.5% 程度もあったので、かなり大きなエラーといえます。FETの動作点によるゲイン調整は内部容量の変化を伴うため、これと周辺のリアクタン

ス成分との作用で位相が微妙に回ってしまうと考えられます。数千分の1の位相安定度を保つには、光入力パワーから信号レベルを揃えておかないとダメなようです。実用機が光学的方法で入力レベルを調整している理由がよく分かりました(^_^;)。

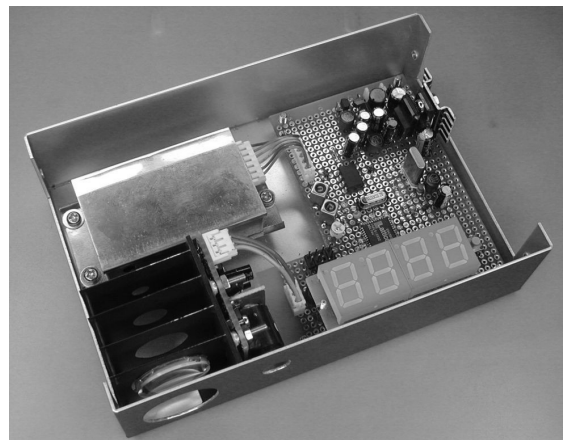
●今後の課題

基本回路はほぼ完成していてまだまだ遊べるので、もう少しいじって次はメカ系(光学絞りや内部基準機構)を追加して実験してみる予定です。アクチュエーターの製作や光学部品がネックですけど、いろいろ工夫すれば何とかなるでしょう(^_^)。

こうなるとメカニクス+オプティクス+エレクトロニクスの3つのテクノロジーの融合(どういう合成語になるんだろ?)ということになりますが、これって日本を代表するカメラ技術そのものなんですよねぇ。うーん、こうやって考えると(ビデオ)カメラ・メーカーって凄いぞー(^_^)。

参考文献

- (1). 田中政芳, 光学式距離測定システムの設計, トランジスタ技術 1993/7/p288~p303, CQ 出版
- (2). Samuel M. Goldwasser, Sam's Laser FAQ, <http://www.repairfaq.org/sam/lasersam.htm>
- (3). 浜松ホトニクス(株), SiPIN フォトダイオード データシート
- (4). 鳥取三洋電機(株), 赤色半導体レーザダイオード DL-3147-011 データシート
- (5). Elantec Inc., EL4083 Datasheet
- (6). Texas Instruments Inc., TL592B Datasheet



電光掲示ファンの製作

本文: ChaN

昔あった光るオモチャで、棒に LED が並べてあって「ぶん！」と振り回すと、並んだ LED が光って文字が浮かび上がるというのがありましたね。LED 列の軌跡が扇状になるので、決まったタイミングで LED を光らせれば目の残像で2次元のパターンとなって見えるというものでした。しかし、これらは棒を振った瞬間しか見えないので、表示器として使えるような物ではなく、商品化されてもあっという間に消えていったようです(玩具としては危険だというのもあったでしょう)。自作例としては、15年位前の「初ラ」が「ラ製」か「I/O」(どれだっけ?)の記事が最初だったと思います。

これでスチール表示をするには、ドットマトリクスLEDやCRTディスプレイのように、スキャン動作を繰り返し行う必要があります。それには何らかの方法でLED列を絶えず動かしてやらなければなりません。具体的な方法としては、てこ・クランク機構で反復動作させるとか、回転体に取り付けてぐるぐる回すなどが考えられます。どちらの方式も一長一短があって、どの点を重視するかによって適した選択が変わってくると思います。今回は、たまたま手元に種となる手頃なDCファン(12V、 $\square$ 85mm)があったので、これを改造してみることにしました。電光掲示板ならぬ、電光掲示「ファン」の製作です(笑)。



基本設計

●メカニカルスキャン

前書きで述べたように、LEDバーを動かす方法として反復式(図1)と回転式(図2)があります。今回は回転式を選択しましたが、短所に示す信号接続の問題からLED個別に配線するのは困難です。このため、コントローラ部を回転体側に配置して、外部からは電源だけ供給するようにしました。

●電源供給

無限回転する回転体との間で電気信号を授受するのには、ワイヤー接続が使えません。これを実現するには、2通りの方法があります。一つはスリップ・リングとブラシで伝送する方法(図3)で、もう一つはロータリー・トランスで伝送する方法(図4)です。最初はポット・コアを向かい合わせてロータリー・トランスでも自作してみようと思ったのですが、回路が少々複雑になるので途中まで考えて止

めました(´_`;)。メカ的な工作だけで済むスリップ・リングの方が後述の理由もあって手軽なようです。

●同期

表示パターンが静止して見えるようにするためには、それぞれのスキャンのたびに同じ位置でドットを点灯させる同期機構が必要になります。たとえば、1回転200ドットで表示するなら、LED列の位置(角度)を200分割で検出する必要があります。これはロータリー・エンコーダーを使えばできないことはありませんが、あまりスマートとは言えません。でも、回転速度が一定であると仮定すれば、ある決まった角度に来た瞬間(インデックス信号)だけ分かれば、そこから一定のドット・タイミングでスキャンを開始して静止表示することもできます。実際は起動後回転が安定したあとは大きく変動することはないので、これで十分です。インデックス・センサーには図5に示すようにいろいろありますが、今回は新顔の半導体磁気センサー(MRセンサー)を使用してみました。

図1. 反復式メカニカル・スキャン

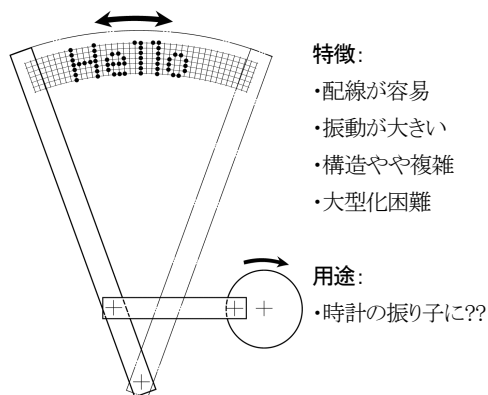


図2. 回転式メカニカル・スキャン

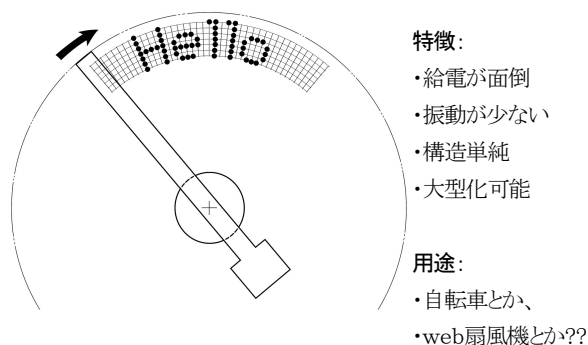


図3. スリップ・リング

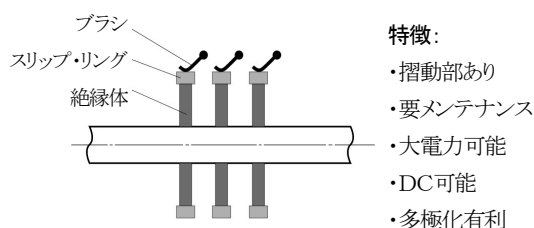


図4. ロータリー・トランス

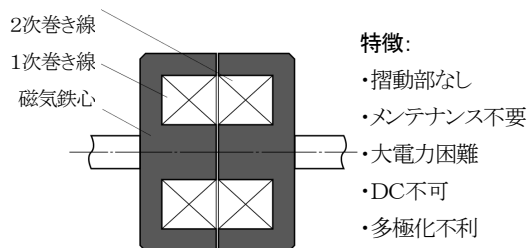
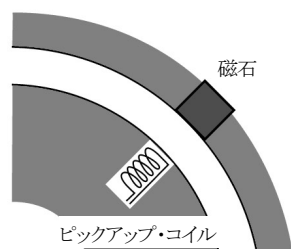
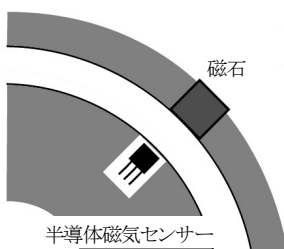


図5. インデックス・センサーのいろいろ

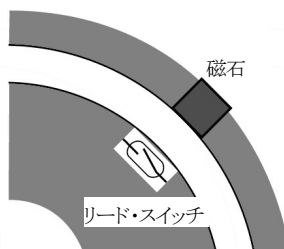
(a). 磁気・誘導式



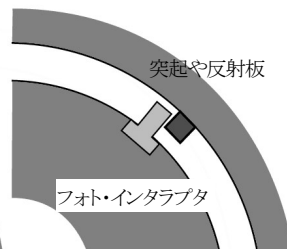
(b). 磁気・半導体式



(c). 磁気・機械式



(d). 光学式



ハードウェアの設計

●電源部(固定部)

写真1に電源部を示します。ファンに 12V を供給すると同時に、ブラシを介してコントローラ部に約 6V を供給します。コントローラのプログラミング等のため、ファンの回転を止められるようにファンへの電源供給はスイッチやコネクタ等でOFFできるようにする必要があります。レギュレーターは発熱を抑えるためにスイッチング方式を使用しましたが、位置的によく冷える(笑)ので3端子レギュレーターでも良かったかも知れません。

基板がブラシの保持を兼ねているので、スリップ・リングとの位置関係に狂いが生じないように、基板はがっちりマ

写真1. 電源部(固定部)

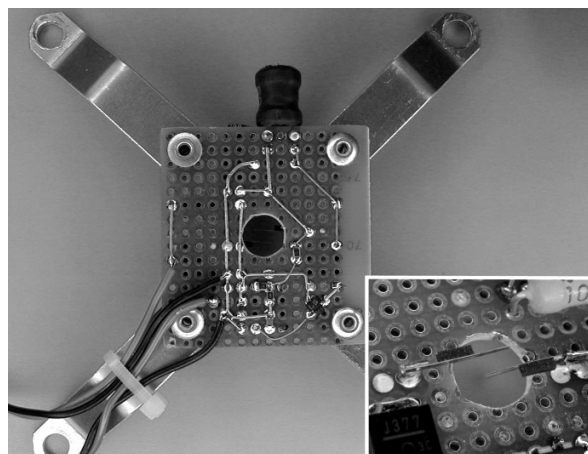
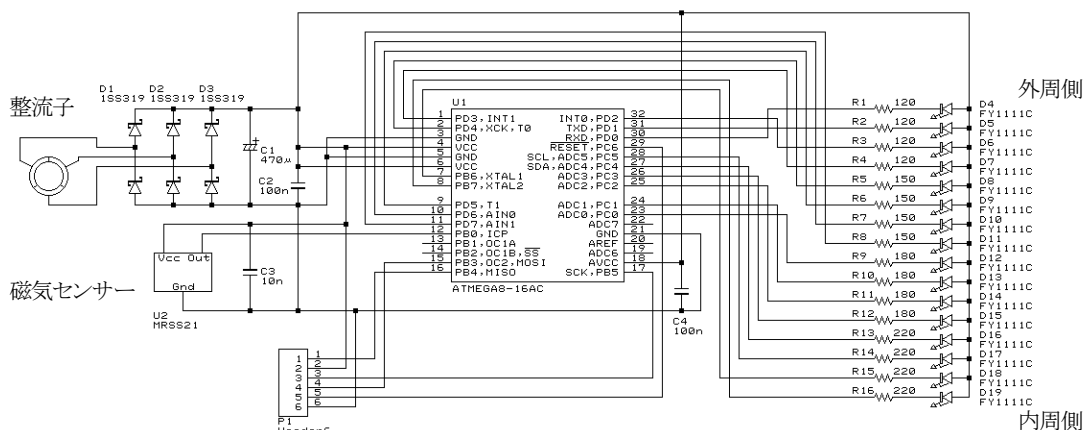


図6. コントローラ部の回路図



ウントする必要があります。

●コントローラ部(回転部)

図7、写真2にコントローラ部の回路と写真を示します。まず、電源はスリップ・リングを介して供給されますが、スリップ・リングの製作がめんどいので、代用としてジャンクの3極DCモーターをバラして、整流子、ブラシ、軸などを流用しました。これは手軽でコンパクトにまとまって良い感じです。ただし、整流子に得られる電圧は3相交流なので、受電側で整流しなければなりません。この分の電圧ロスを見込んで供給側で6Vとしています。

LEDアレーは気流を乱さないように羽根の後縁に取り付けます(写真3)。日本語表示するには16ラインは欲しいところなので、1608サイズの高輝度黄色LEDを16個並べることにしました。内周側と外周側とはLEDの線速度が異なるので、単位線分あたりの光出力が大きく変わるように直列抵抗の値を変えて外周側の電流を多めに設定します。

コントローラーには ATmega8(アトメル)を使用しました。またこれですって? ...いや、外付け部品なしで動いて結構便利なもんで(在庫の消化もありますけど^_^)。

基準位置を検出するセンサーには感度の高いMRセンサーを使いました。バランスも考慮してLEDの反対側にマウントしてあります(写真4)。ベンチレー側には磁石を埋め込んでおきます。整流子は心ぶれが無いように正確に基板を固定します。整流子シャフトの正確な心出しはいろいろ工夫してみてください。

一通り組み立てたら、フル・スピードで回転させてみて、振動が少なくなるようにカウンター・バランス等で重心を調整します。2000rpm 程度までなら特にバランス対策なくても問題は無いようです。静止表示は、1000rpm 以上の速度なら実用的な表示ができるようです。

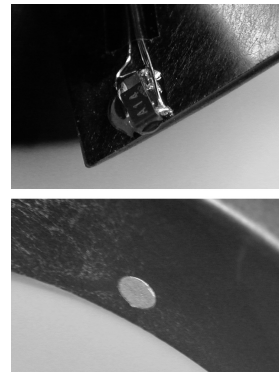
写真2. コントローラ部(可動部)



写真3. LEDアレー



写真4. センサーと磁石



プログラムの動作

はっきり言ってあまり説明するところはないです(^_^;)。インデックス信号でフレーム同期をとりながらあらかじめROM に記録されたパターンとスクリプトで表示動作を繰り返すだけです。創刊号のLED名札の F/W...

FTD232 を用いた USB 接続 AVR マイコンライタの試作

おゆう

1.はじめに

ATMEL AVR マイコン開発環境は安価に用意することができます。

ソフトウェア環境は同社の AVR studio が無料で配布されており、アセンブラによるプログラミングが可能です。また、C言語による開発も gcc を使うことができます。もちろん、無料で使用できますし、コンパイルが終わった windows installer 付きのものも有志により配布¹されています。

デバイスのプログラミングに必要なハードウェアはというと、市販のものから自作のものまであります。アマチュアとしては、(金銭的な理由もありますが) 自作でいきたいところです。幸い、ATMEL 社の Application Note(以下、AN と略す)にて、ライター制作事例やポイントが記載されています。また、各デバイスのデータシートにも FLASH ROM 書き込みに関する詳細な記述がされており、これを参照することで自作が容易です。

今回紹介するライターは、各所で紹介されている AVR マイコンライター²と機能的に同じモノになります。少しだけ異なるのは USB 経由で書き込みを行うことです。USB-serial 変換アダプタを内蔵しただけですが、書き込み制御マイコンにも少々手を加えてみました。

『ライターを作りました。これをどうぞ』では記事にも何にもなりませんので、AN0943 の一部紹介と付加した制御ソフトの紹介もさせていただきます。また、本記事のソフトウェア・ハードウェアについては、最後に紹介する URL にて配布させて頂く予定です。

2. AVR マイコンライター構成

ハードウェア

今回、試作をするに当たり、IPI 社³の USB232/AT90S (図 1) を使用しました。このボードには、FT8U232AM(以下、FT232 と略す)と AT90S2313 (以下、S2313 と略す)が搭載されています。FTDI 社のチップは USB-Serial 変換用の石で、同社の提供する汎用デバイスドライバを使うことで仮想 COM ポートとして認識されます。この COM ポート経由でボード上の S2313 とデータ送受信が可能です。また、ユーザ I/O として 11bit のデータ入出力が可能となっています。これを利用して AVR マイコンへのシリアルプログラミング、動作切り替えスイッチの入力とします。追加基板には、ターゲットボードとのレベル変換用に 74HC126、動作切り替え用のスイッチを載せました。

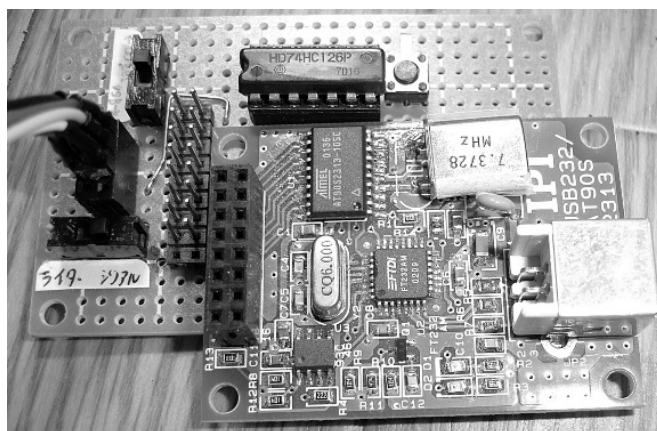


図 1. 試作機外観

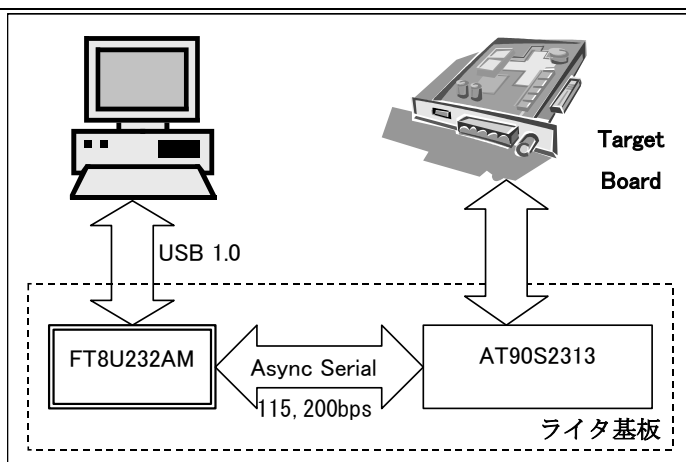


図 2. ライタ接続例

¹ AVRgcc, WinAVR という名称でパッケージがリリースされています。詳しくは AVRfreaks(<http://www.avrfreaks.com/>)を参照ください。

² トラ技誌で紹介されたライターとして、ChaN 氏の公開されているライター(<http://elm-chan.org/>)が、PC のシリアル or パラレルポートと直結でプログラミング可能です。メインのライターとして使用させて頂いております。

ライタ用の S2313 へ書き込み指示を出す側のソフト (PC 側ライタソフト) は、試作ということもあり AVR studio 添付のソフトを用います。これは、AN のライタコマンドと互換を保つことで使用可能です。将来的にはライタソフトも自作して、より高速なプログラミングができるようにしたいと考えております。

FLASH へのプログラムアルゴリズムは、データシートや AN.0943 で示されているアルゴリズムにしたがいます。また、プログラム制御用ソフトウェアは AN と同様に公開されている AVR910.ASM を流用し、追加修正を行います。このソフトは AT90S1200 (以下、S1200 と略す) 用に作られており、スタックやハードウェア UART 機能を使用していません (S1200 が持っていない)。今回は S2313 を使用しますので、これらの機能を有効に使用します。また、IPI 社のオリジナルの X'tal では UART のビットレートに誤差が出るので、7.3728MHz に換装しておきます。

⇒ AVR910.ASM の修正箇所

アセンブラソースのため、コメントをあわせて千行を超えてしまっています。紙面を割いて載せるほどのものではないので、S1200 から S2313 へとマイコンをリプレースする際の変更点を記します。

(1) スタック・周辺デバイス等の初期化処理追加

スタックのあるデバイスへと変わりましたので、初期化処理にスタックポインタ設定・周辺デバイスの初期化処理追加・修正を行います。周辺機能のうち、タイマ・UART・I/O ポートの初期設定を行います。PC との通信速度は、これまでの実績とデータの転送時間を短縮することを考慮して 115,200bps としました。タイマは、UART 等の割り込み処理が走るため、書き込み待ちなどの遅延時間を正確に得るために使用することにしました。

また、制御 PC との USB の通信が確立されるまでは、初期化処理完了で停止するようにしておきます。FT232 の USBEN 端子を監視して⁴、enable になるのを待ちます。

(2) 書き込み完了待ち処理の高速化

オリジナルのソフトではフラッシュ ROM の書き込み完了待ちは、時間待ちで処理しているため効率が悪いです。AN0943 にも記載されている⁵ように、ポーリング処理に置き換えます。記載内容の概略は表 1 の通りです。

(一部、別のデータシートから補完。) すり替える場所は、フラッシュへの書き込み完了時間待ちをしているところになります。デバイスにより下記の値 M や遅延 N が変わるので、分岐処理も必要です。

(3) ISP 用の I/O bit 割り当て変更

もちろん、ライタに使うハードウェアが異なるので、それに合わせた修正が必要となります。初期化とビット制御まわりの処理をハードウェアにあったものに置き換えます。

表 1. AN0943 : ポーリング処理の概略

- デバイスによっては、フラッシュまたは EEPROM への書き込み完了待ちにポーリングが使える。
- バイトデータが、プログラムされる時、プログラム中のアドレスを読み出すと値 M (多くは 0xFF) が得られる。
- デバイスが新しいバイトデータの書き込みが完了したとき、プログラムした値は正常に読み出される。
- 値 M の書き込み完了ポーリングがうまくできないとき、次の値を書き込む前に遅延 N を用いて書き込み完了を待つこと。
- erase コマンドで消去した箇所は 0xFF になっているので、同値であればプログラムは不要。
- 値 M のプログラムは、ポーリングができないので遅延 N を用いて書き込み完了ウェイトをとること。
- 値 M ・遅延 N はそれぞれのデバイスデータシートに記載されている。遅延 N は電圧にも依存している。

⇒ VR910.ASM への機能追加

単なるライタとしてだけでは勿体ないので、資料[5], ChaN 氏の COM ポートを使ったライタと同様に、ISP 端子を用いてターゲットデバイスとのシリアル通信を行えるようにします。ここで、簡単のためターゲットデバイス側のシリアル転送速度を 38,400bps 固定とします。PC との接続は、115,200bps としていますので、レート変換を行う必要があります。ライタ機能と追加する通信機能は、ソフトの作成を容易にするために外部のスイッチで切り替えることにします。タイマ 1 の使い分けをせず、Software UART はタイマ 0 で行います。

制御マイコンとターゲットマイコンとの通信は、双方ともに Software UART となります。あらかじめ原発をキリの良い値にしておいたので、ハード的な要因で大きく誤差が蓄積されることはありません。そこらかしこで見かける

³ AVR, PIC, SX マイコンを各種取り扱っておられます。また、FTDI 社の正規代理店でもあります (<http://www.ipishop.com/>)

⁴ 最初、TXDEN も監視していたのですが、よく見ると RS485 の送信可信号でした... 使用するのは RS232 ですから監視は不要です。

⁵ AN0943 p.6 の「FLASH PROGRAM MEMORY ACCESS」参照。本文中の概要は意識しています。

Software UART で恐縮ですが、今回実装した全二重通信の概要を記述させていただきます。

前提として、通信速度の倍の速度（1ビット長の半分の時間）でタイマ割り込みを発生させます。送信または受信が有効なときにだけ割り込みを有効とした方が省エネにはなりますが、今回はシリアル転送速度の変換のみに注力するため、実装の楽な方法をとりました。送信・受信処理の状態遷移図を図2・図3に示します。

タイマ割り込みハンドラでは、優先順位の高いものから順に処理を記述します。まずは必要レジスタの待避を行ってから、次の割り込み時間をセットします。ここで用いているのはタイマ0ですので、カウンタオーバーフローで割り込み発生となります。

次に処理するのは、受信ビットの取得です。サンプリング間隔を一定に保つため、受信ピンのデータを記憶しておきます。その後で、送信処理を行います。送信データも等間隔で変化させられることが望ましいですが、簡単のため、処理を先に行うことにしました。その結果、数サイクル程度の誤差が出ますが、受信側でカバーできる範囲のようなので OK としました。

概略のみ記述いたしましたが、ソースコードを追ってみてください。それぞれの状態に割り当てる状態値をうまく具合にすることで、状態遷移をインクリメント発で回せるようにしています。割り込み処理中の負荷を軽減させるため、状態値は、ここで示した遷移図通りではなく、ループしているところを展開した形にしています。

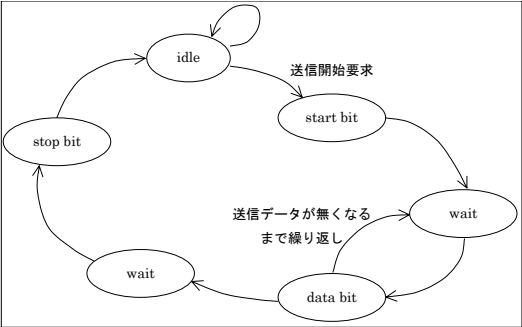


図 2. 送信処理状態遷移

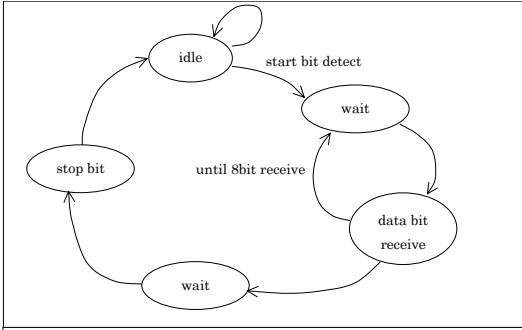


図 3. 受信処理状態遷移

3. 結果

試作したライタと、ChaN 氏のライタ(NT 系 OS 対応の navrss)との書き込み速度比較を行いました。書き込むデータは 0x00, 0x01, 0x02,...,0xFF の繰り返しデータをデバイスのプログラムメモリいっぱい書き込み～ベリファイを実行するのに要した時間です。明らかに試作機のほうが遅いですね。試しに、試作機の書き込み確認をポーリングではなく、6 ミリ秒遅延として比較・測定してみましたが、表と同様の結果となりました。ライタマイコンと PC とのプロトコルがネックになっているものと考えられます。1 byte 書き込み・読み込みを行うために、コマンド発行・結果待ちを繰り返すからです。

表を見てのとおり、Win98 系で使える avrss が最も早く、他との差が歴然となっています。NT 系 Windows では、やはり直接ハードを叩くのはおいしくない環境であるといえます。そのために外部に処理を任せようと、今回の試作を行ったのですが、さすがに S1200 用に作られたもので、かつ速度面についてあまり考えられていない(?)ものをベースに使ってしまったのは失敗でした。(S1200 で作ったこともないんで、速度比較はできませんが…)

表 1. 書き込み速度比較

DEVICE	本機	navrss	avrss
AT90S2313(2KB)	49.9 秒	37.3 秒	8.8 秒
AT90S8515(4KB)	98.9 秒	73.7 秒	14.0 秒
AT90S8515(8KB)	197.6 秒	124.2 秒	28.5 秒
※ Xtal はそれぞれ次の通り. S2313:8MHz, S8515:7.3728M.			
※ 使用 PC: PC-PJ2X4(P2-333) avrss は win98, 他は Win2k にて.			

4. 改善方法

今回の結果より、改めてライタをオリジナルで作ることにします。PC 側の制御ソフト作成が手間となるので、制御マイコン側に簡易 UI を載せてしまいましょう。書き込むソフトウェアは intel HEX ファイルをそのまま流し込むのが(ユーザに)としては便利ですね。こうなると下りデータのみハードウェアフローを入れておく必要があります。今回使用した基板では既に RTS/CTS が結線されているので、個人的に弄るのがイヤです。そこで、試作よりも基板サイズが大きくなるかもしれませんが、自作しようかと思っています。作成予定の回路図を図4に示します。

単純に待っている箇所を修正すれば高速化が図れると踏んでいたのですが、現実はそのなかに甘くなかったようです。本稿執筆時には、回路・ソフトのほうは未だ完成しておりません。なんとか配布できるよう注力します。入門書も執筆中でしたが、年末以降になりそうです。

今回は、『改善したつもりが駄目でした』という結論で、ご容赦くださいませ... (´Д`)

參考資料

- [1] AVR910: In-System Programming Rev. 0943C-11/00, Atmel Corporation
- [2] FT8U232AM Preliminary Information Rev 0.8, Future Technology Devices Intl.
- [3] FT232BM Designers Guide Version 1.1, Future Technology Devices Intl.
- [4] Application Note AN232-*, Future Technology Devices Intl.
- [5] AVR ライタ製作集 ライタソフト添付の Tips, ChaN 氏

ライタ制作に必要な資料配付場所 / 問い合わせ先

U1 lab. : <http://homepage2.nifty.com/u1lab/danyo3/>

E-mail: u1lab@nifty.com

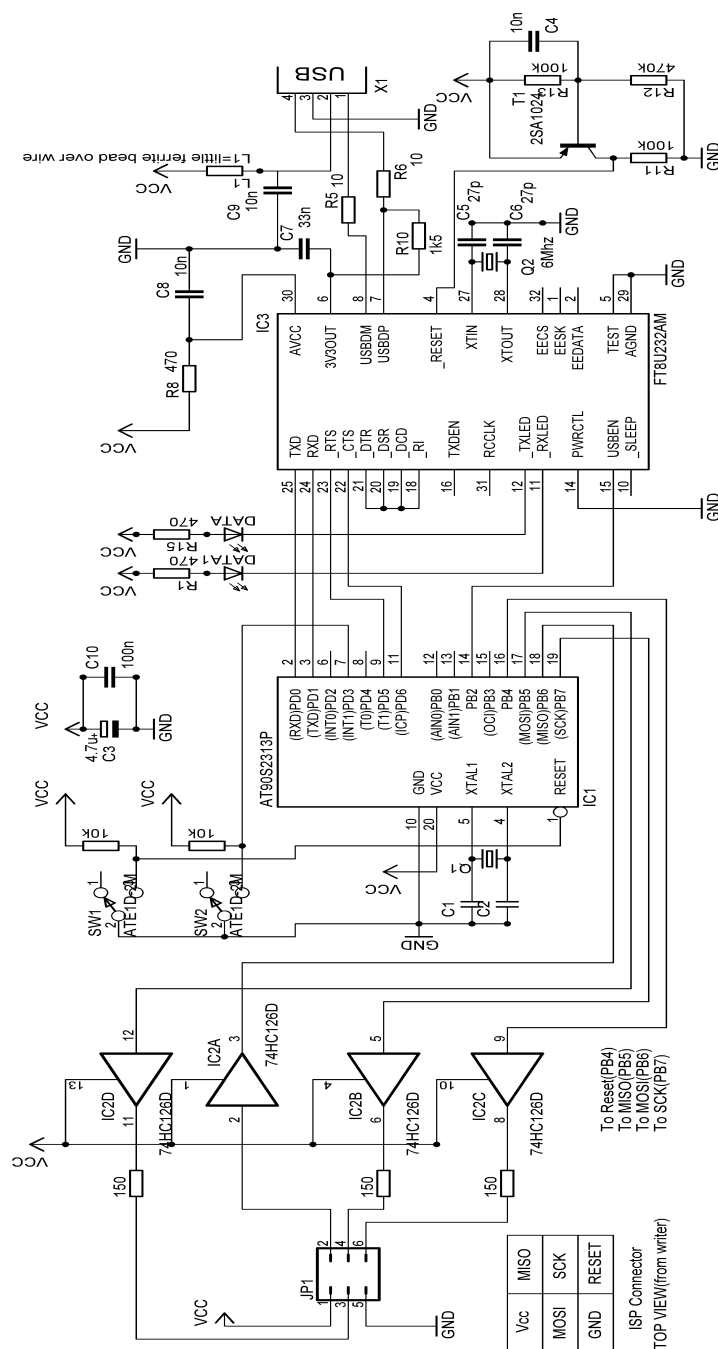


図4. ライタ回路(制作予定)

注:コネクタのピン番号と、一般のコネクタのピン番号と一致してはいけません。

EAGLE のライブラリを適当に漁って付けているので (ライブラリも自作しないと駄目ですね)

マクロークールライトみたいなものを作る

2003 年 8 月 17 日

ROM男 (cma59466@biglobe.ne.jp)

1 なぜ作るの？

Nikon の CoolPix900/4500 シリーズ用に Macro Cool Light という製品が出ています。非常に面白そうな製品で買いたいという気になりましたが、以下のような点が気にかかりました。

- 3φ の白色 LED が 8 個しか使われていないので暗そう (Macro Cool Light を複数購入して LED 基板を組み合わせで増灯されている方もいます)
 - リチウム一次電池を使うなんてもったいない
- その上“スパコンを作ろう”で使ったプリント基板 AE-2M の大きさが Macro Cool Light とほとんど同じというところから自前のクールライトもどきを作りはじめました。

2 回路

2.1 設計

LED は、5φ のものを 30 個使うことにしました。3φ のより 2 倍明るいこととあわせるとおよそ 7.5 倍の輝度になるはずですが。ChaN さんの回路は LED を二つ直列にしたユニットを使っていたのですがここでは一ユニットあたり LED を三つ直列に繋ぐことにします。

次に電源はニッケル水素電池を使うことにします。LED ユニット一つには、約 21.2mA 流れると考えられます。よってすべてのユニットで 212mA 流れます。電圧はおよそ 10.9V になるでしょう。各定数は TL497 のデータシートを元に概算しました。

- $I_{pk} = 2 \times I_{o_max} [V_o / V_i] = 2 \times (21.2 \text{ mA} \times 10) \times [10.9V / 2V] = 2.31A$
- $L (\mu H) = V_i / I_{pk} \times T_{ON} (\mu sec) = 2.0 / 2.31 \times 100 (10kHz \text{ と仮定}) = 86.6 \mu H$
- FT-82-43 を使うと $557nH / turn$ より $\sqrt{(86.6^4 / 557^2)} \sim 12.5turn$ (発振周波数に依るので調整が必要)
- $I_i = 0.212A \times 10.9V / \{2 \times 0.6 (\text{レギュレーターの効率})\} = 1.92A$
単 3 型のニッケル水素電池 (1 本あたり 1600mAh~2000mAh) で約 50 分~62.5 分利用可能?
- $C (\text{平滑用}, \mu F) = T_{ON} \times \{(V_i / V_o \times I_{pk}) + I_o\} / V_{ripple \text{ pk}} = 100 \times \{(2 / 10.9 \times 1.92) + 0.212\} / 0.5 = 112.9 \mu F$ としてこの値より大きな 220μF を使用する

2.2 パーツ

2.2.1 トランジスタとSBD

ドライブ用トランジスタは 6A 前後で V_{sat} が低く、かつ V_{CEO} が $10.9V + 2V = 12.9V$ 以上のものを選ぶことにしました。東芝の製品からは適切な銘柄を見つけることができず (60V 12A の容量の 2SC3709A あたりが適切といえば適切ですが)、NEC のカタログを探した末に 2SC4553 (100V, 7.5A) を使いました。制御用のトランジスタは ChaN さんの回路でドライブ用に使われている 2SC3279 を使いました。SBD に必要な容量は 3A かそれ以上ということになります。

サトー電気のカatalogとにらめっこした結果 40V 5A の 5GWJZ47 を使うことにしました。前述の式は回路を作った後に見直したもので当初 I_i は 3.6A と見積もっていました。半固定抵抗の値は 2SC4553 の h_{fe} ($IC = 5A$) が 150 なので $2V / 3.6A \times 150 = 83.3 \Omega$ と考え、ほぼ中央でこの値となる 200Ω のものを使いました。ただ、後でバイパスコンデンサをつけたのでこの抵抗はもっと大きく (500Ω 程度?) ても動いたはずだと思っています。

2.2.2 フェライトコア

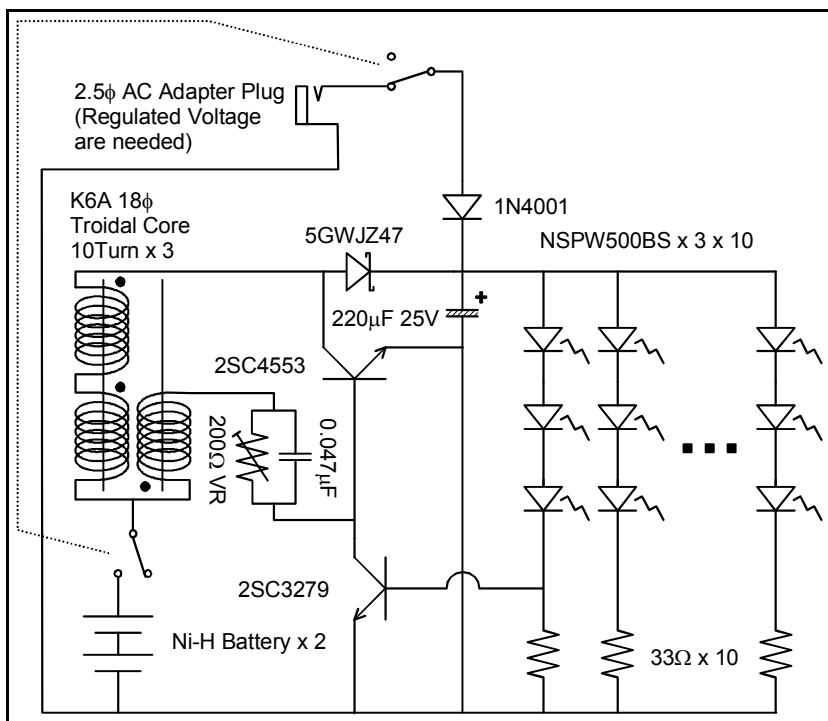
最初はアミドンのフェライトコアを使おうか? と考

えていましたが、東京ラジオデパート 3F の斉藤電機
商会で 18φ の TDK 製トロイダルコア (K6A 材) を見
つけたので使用しました。このコアは 60nH/turn な
ので 10 回巻で 6μH、20 回巻で 24μH になります。
二つのコアを貼りあわせ、0.5φ のポリウレタン線を
巻きました。TDK のコアなら通信販売をしている商
社 (<http://www.hi-seiko.co.jp/shop.html>) があります。
H5A 材などを選べるので利用されてはいかがでしょ
うか。巻き数はかなり調整しました。最初は 10turn
×2 としたところ、比較的安定して発振したものの
2SC4553 が結構発熱することが分かりました。そこ
で巻数を入力側 10turn, 出力側 20turn に変更たので
すが発振が不安定になり、発振が停止すると 2SC4553
もあつという間に熱くなりました。そこで手持ちの
セラミックコンデンサ (0.047μF) をバイパスコンデン
サとして半固定 VR と並列に入れたところ問題なく
動くようになり、2SC4553 もほとんど発熱しなくな

りました。このときの発振周波数は 20kHz よりかなり上のように、音が聞こえません。

2.2.3 スイッチ・AC アダプタ・電圧降下用ダイオード

電池から流れる電流はおおよそ 1.6A 程度になります。最初の計算では 3A 程度電流が流れると考えていたのによく目に付く 3A, 125VA のスイッチは使いたくありませんでした。ヒロセパーツに寄ったときに 6A 125VA の二連スイッチ (ミヤマ MS-500F-B) を見つけたのでそれを使うことにしましたが 3A のもので十分だったようです。AC アダプタは秋月電子で売っているスイッチングタイプ 12V のを使いました。このとき、電池駆動よりもやや高めの電圧がかかることになります。そのままだでも別にかまわないのですが、明るさを出来るだけあわせるために整流用ダイオード (1N4001) を入れることにしました。



マクロクールライトみたいなものの回路図

2.2.4 電池ボックス

電池からは大電流が流れるので、電池ボックスと電池の間に生じる接触抵抗を出来るだけ減らさなければ

ばなりません。今回は損失が少ないと謳っている秋月電子通商の電池ボックスを使いました。プリント基板に直接固定する余裕はないので 10mm のジュラコンスペーサーを介して取り付けています。

2.2.5 ライトのカメラへの固定

ケンコーのステップアップリング (28mm → 37mm) を使い、カメラ本体に固定することにしました。

2.3 店

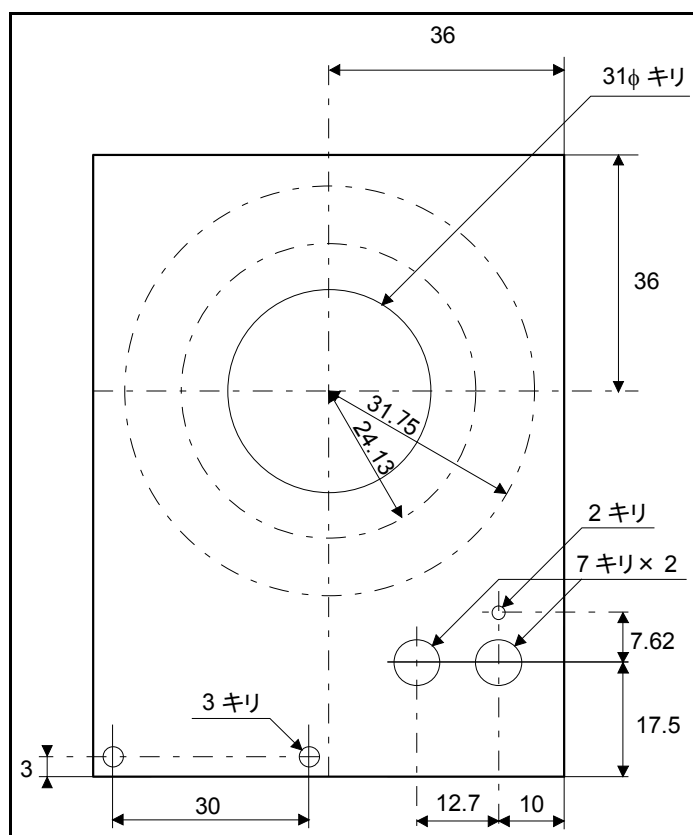
今回使った部品は以下の店で手に入れました。

2SC4553	サトー電気	電池ケース	秋月電子
2SC3279	サトー電気	プリント基板 (AE-2M)	秋月電子
5GWJZ47	サトー電気		
200Ω 半固定	サトー電気	AC 電源, 12V	秋月電子
フェライトコア	斉藤電気商会	スイッチ MS-500F-B	ヒロセ
NSPW500BS	千石電商	ジュラコンス ペーサー	ヒロセ
33Ω 抵抗	千石電商	ねじなど	ヒロセ
0.5φ UEW	千石電商	ステップアッ プリング	ヨドバシカメ ラ
1N4001	千石電商		

3 組み立て

3.1 穴あけ

プリント基板への穴あけ位置を示します。いろいろ数値を書いています、現物あわせや蛇の目の穴に合わせた穴あけであり厳密にはずれているものもあるかもしれません。左下の 3φ の穴二つはジュラコンスペーサーで隙間を作った上で電池ボックスを固定するためのものです。右側の 7φ の穴にスイッチと AC アダプタ用のソケットを組み込みます。一点鎖線で描いた円弧が LED を配置しようとした位置です。31φ の穴はカメラのレンズを通すためのもので、会社のホールソーで開けました。

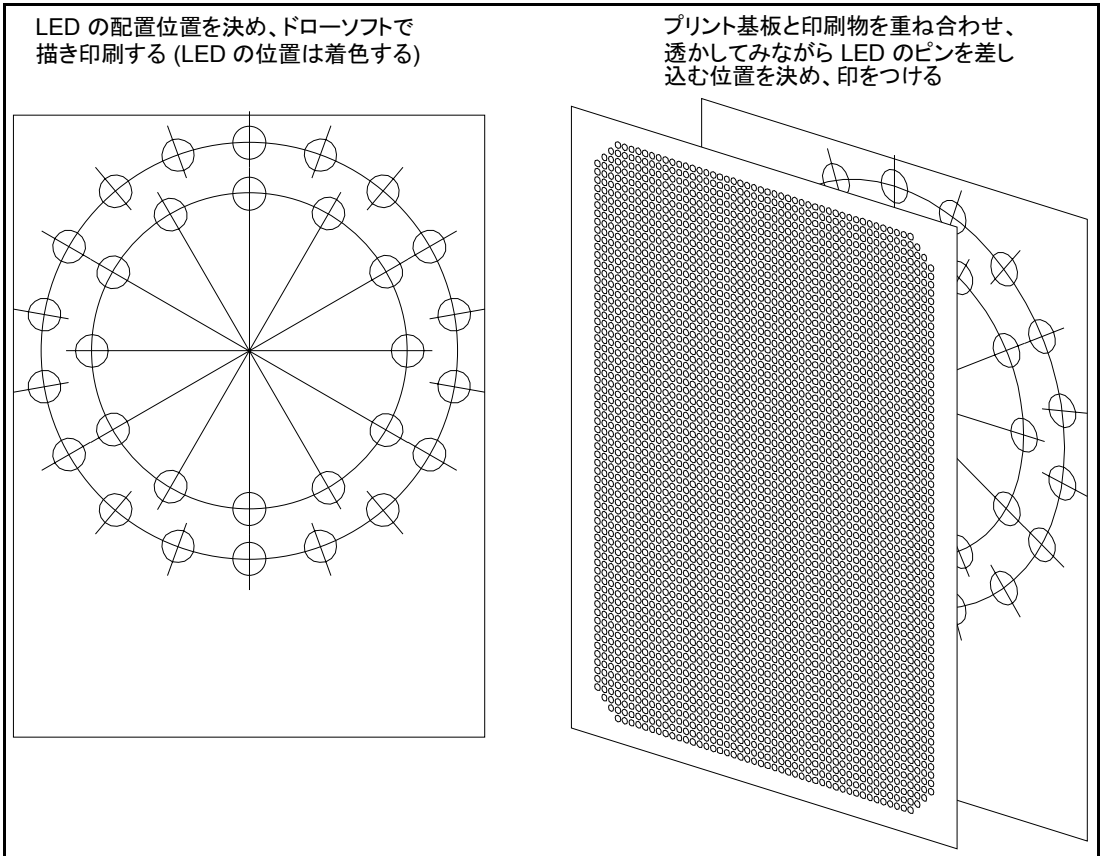


プリント基板への穴あけ位置図

3.2 LED の配置

LED の配置位置は次の図で説明したような方法で決めました。正確な図を描けるドローソフトとプリンタが安価に手に入るようになったので簡単に位置決

めが出来るようになりました。LED はできるだけ同心円状に並べたかったのですが蛇の目の位置はそれほど望んだようには並んでいませんから妥協する必要があります。ただし、左右・上下対称な位置になるようには心がけました。



LED 配置位置の決め方

3.3 フェライトコア

フェライトコアを用いたトランスの作り方を次のページに示します、といっても二つのコアを貼り付けた上で UEW を巻いただけです。ただし巻線は正しく結ばねばなりませんからその点は気をつけてください。

3.4 スイッチと AC アダプタ用プラグの位置

スイッチを指で動かすことを考えた結果スイッチは向かって下右端 (使用時は左端) に来よう配置しました。

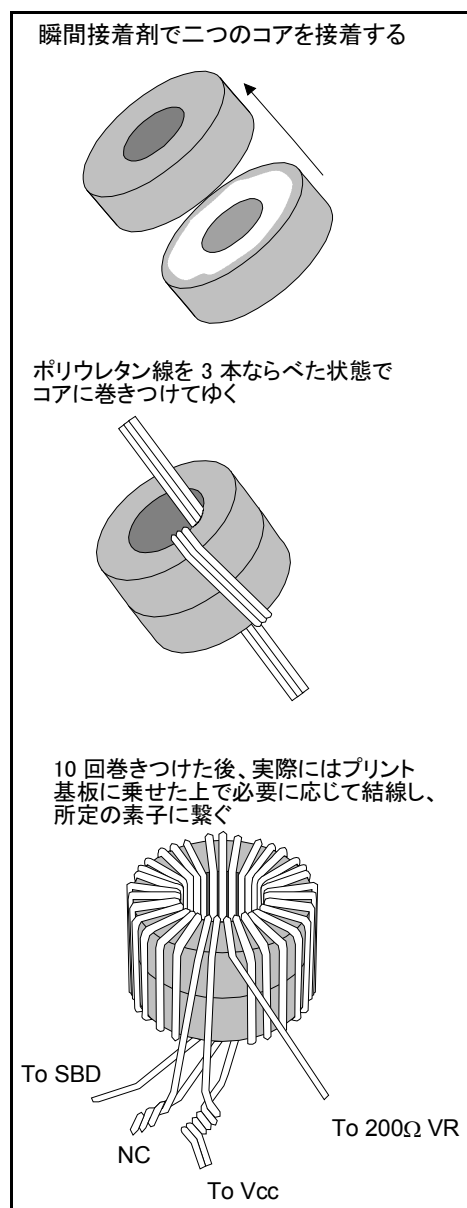
3.5 その他の素子の配置

トランジスタや整流用ダイオード、抵抗は行き当たりばったりと並べたのでバランスが良くありません。マクロール泰斗みたいなものを製作される方は素子の配置をよく考えた方がいいと思います。

3.6 配線やらなにやら

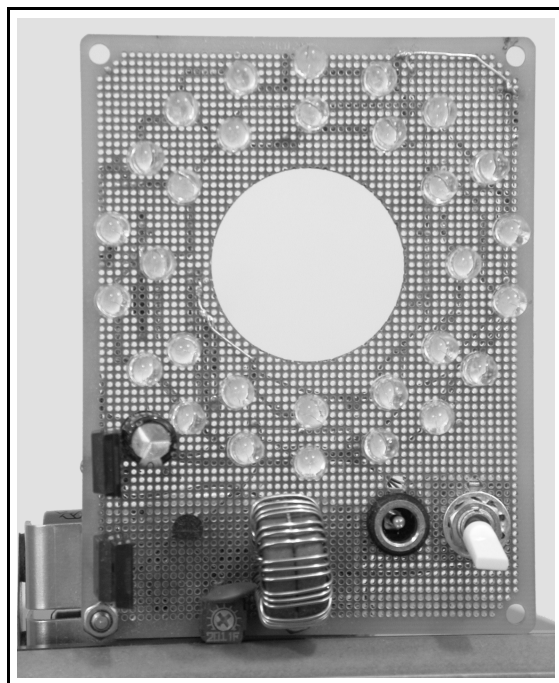
配線は各素子の足と 0.5φ UEW を使いました。UEW 以外の部分は剥き出しです。本当はこういうものをそのまま取り付けたらカメラに傷がついてしまいます。私はそういうところは全然かまわないのでそのまま取り付けた。しかしそういう部分を気にか

ける方は適切なクッションをカメラとライトの間に
入れるべきでしょう。

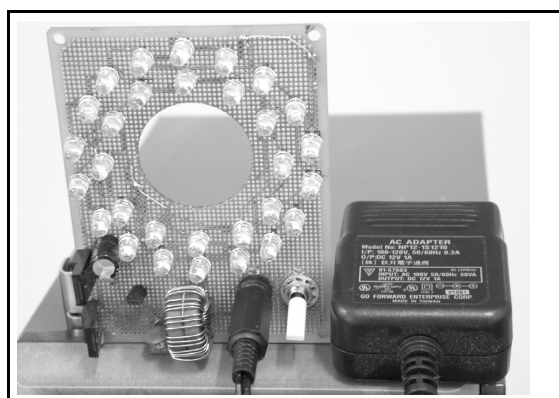


トランスの作り方

トは強いようです。悔やまれるのが部品配置、および
配線を通した位置です。もう少し考えて並べてゆけ
ばコンパクトに収めることが出来、後述の“縁を丸
め”たり“LED 用カバーを取り付け”たりも難なく
出来たと思うのですが…。



ライト本体



AC アダプタを取り付けた状態

4 完成品

4.1 画像

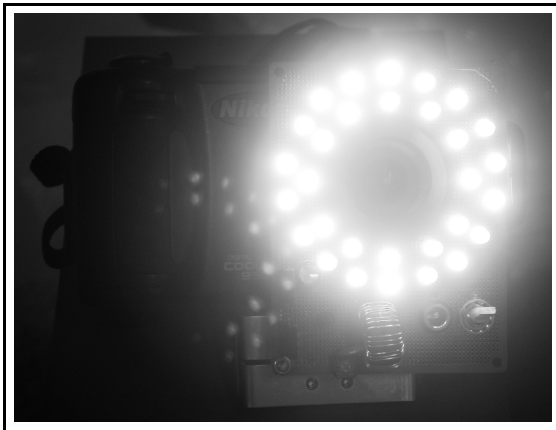
ライト本体の写真、カメラに取り付けたときの状態
などを次に示します。非常に不恰好ですがインパク

電源スイッチは端につけた方が操作しやすいわけ
ですがこのことに気が付いたのは一回組み立てた後で
した。慌てて AC アダプタのプラグの位置とスイッ
チの位置を交換しました。



CoolPix 950 に取り付けた状態

CoolPix 4500 に取り付けるより 950 に取り付けた方が格好いいのでやってみました。



ライトを点灯したところ



点灯しているところを横から撮影したところ

4.2 撮影例



ライトを使って CoolPix 950 を撮影したところ

わざとらしく LED を映り込ませてみました。しかしこの場合でもシャッター速度は 1/15 秒までしか短縮できず、カメラを三脚に押し付けて撮影しないとブレてしまいました。

4.3 持続時間

ベストケースでどのくらい光りつづけることができるのか試してみました。東芝の 2000mAh のニッケル水素電池をフル充電して点灯させました。時間とともにわずかに明るさは変化していったようですが、2003 年 4 月 1 日に調べたところ、97 分たったところで急激に明るさが衰えました。本物の Macro Cool Light の電池寿命は 6 時間くらいだそうです。それからすれば 1/4 しかもたない、ということになりますがこれで十分じゃないでしょうか？いざとなれば予備の電池を持ち歩けばいいのですから。

5 使ってみた感想(反省点)

5.1 カメラに取り付けにくく外しにくい

ステップアップリングで取り付ける際に、カメラのレンズに触りやすく指紋をつけてしまいます。また、ステップアップリングをねじ込んだ後もライト自体はぶらぶらしているのがちょっと頼りない感じがします。更にステップアップリングを外すのも結構面倒

です。この問題について、

「フィールドスコープにデジタルカメラを取り付けるためのアダプタ、ケンコー FS28 を使ってはどうか？」というアイデアをいただきました。これはなかなか面白そうです。接続のためには適切なサイズ (外径 34φ) のパイプを探さねばならないこと、顕微鏡の接眼鏡に取り付けるアダプタとして使用したときに耐久性に問題があったという Web ページもあったことからすぐに試すことは出来ませんでした but 興味を持っています。しかし…ケンコーのサイトには FS28 の紹介がないのでしょうか？

5.2 三脚に取り付けられない

電池ボックスがぶつかるので三脚に取り付けられませんでした。むしろ被写体側に電池ボックスを置くべきだったと感じています。

5.3 LED が曲がりやすい

手が触れないようにカバーをつけるべきだと思います。が、このあたりなかなか難しいです。ジュラコンスペーサーをいくつか建ててカバーを取り付けられるようにしたらよかったかな？

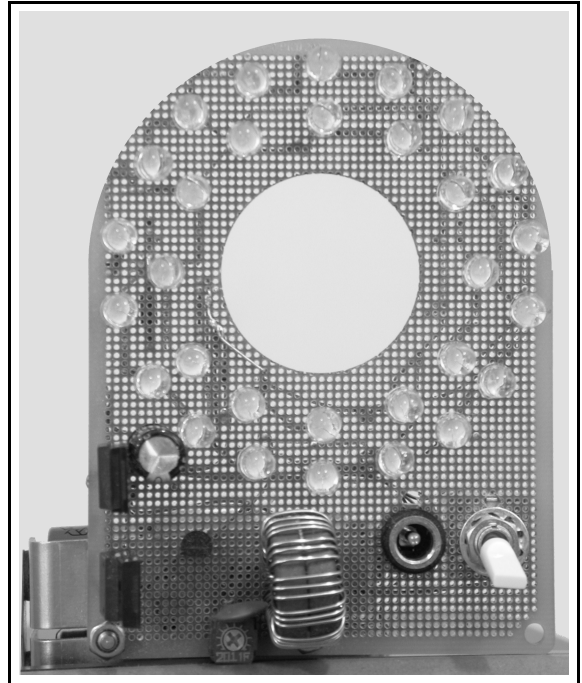
5.4 上半分は縁を取った方が良かった

という意見がありました。もっと Macro Cool Light っぽくなるのでそうすべきだったと 気づきましたがそのとき既に何も考えずに配線を通してしまいましたので…。右にそれっぽいコラージュを作ってみました。2 ページ前の元の絵より格好いいでしょうか？

5.5 明るさだけは負けません

これだけは負けていませんし、このことだけでもマクロクールライトみたいなものを作った甲斐がありました but 明るさを抑えたい場合がある、といわれました。そこで電流制限方法をいろいろ考えてみま

したが、結局次のページのような回路が妥当だろうと考えました。あまり好みではない制御方法ですが簡単な回路で済むところがメリットでしょうか。



Macro Cool Light もどきの上半分の縁を丸めたモデル (コラージュです)

6 最後に

他愛もない製作記に長々とお付き合いいただきありがとうございます。もっと高度な製作もやってみたいと思いますがなかなか実力が伴いません。本音を言えば今回の製作は

「トロイダルコアに線を巻いてみたいよ～。SBD を使ってみたいよ～」といったところから始まっています。この文章を見て

『何か作りたくなった』と思ってもらえれば幸いです。また、LED ライトの世界は凝りまくっている方々がインターネット上にたくさんいらっしゃいますので検索して訪問してみたいかがででしょうか？

デジタル オーディオ アンプの製作

Kim

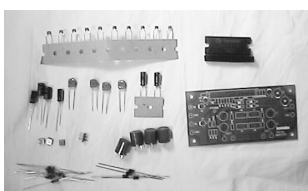


☆デジタルアンプって？

最近、デジタルアンプを用いた製品をよく見かけるようになりました。デジタルアンプの多くはPWM (Pulse Width Modulation) という、入力信号に応じて幅を変えたパルスをローパスフィルターに通す方式です。デジタルアンプというよりはスイッチングアンプと呼ぶほうがしっくりくるでしょう。デジタルアンプは高効率、大出力、小型化が可能であり、AVやカー用を中心に普及してきました。最近ではシャープの1bitアンプやTacT audio など、本格的なピュアオーディオ用途でも使われています。今回製作するのは、トライパス社製 TA2020-020 というICを使ったオーディオアンプです。このICは+12Vの単一電源で動作する仕様で扱いやすく、(株)カマデンから作りやすいキットも発売されています。
(<http://www.kamaden.com/index.html>) 今回はこのキットを製作します。



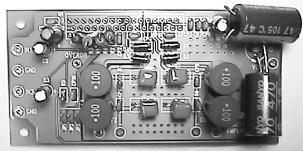
【写真1】カマデンから発売されているキット。コイズミ無線にて4800円で購入。



【写真2】部品点数は少なめ。

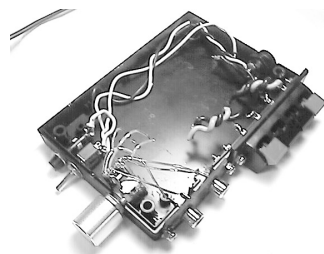
☆キットの製作

キットにはトライパスのICと周辺部品が入っています。電源回路、入出力端子やボリューム、筐体はお好きなものを組み合わせてください。まずは基板を作成しましょう。



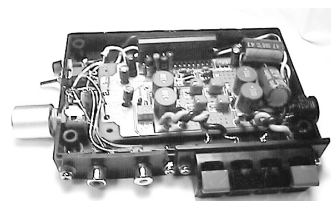
【写真3】IC以外の部品をハンダづけしたところ。一部抵抗をオーディオ用に、電解コンデンサを470 μ FのOSコンに変更した。

説明書にしたがって部品をハンダづけしていきます。ハンダづけに慣れてない人にはICの足のピッチが狭いかもしれませんが、丁寧に作業すれば大丈夫でしょう。C2、C3の電解コンデンサはロットによっては入力側がプラスになっていますが、入力側はマイナスが正しいので注意してください。全ての部品が組み込まれると写真3のようになります。



【写真4】筐体に入出力端子、ボリューム、電源スイッチを取り付け。電源はACアダプタを使用。

次に筐体を作成します。基板についていない入出力端子やボリュームを取りつけます。今回は電源にACアダプタを使用します。



【写真5】筐体に基板を取りつけたところ。配線は最短にした。

筐体に基板を取りつけます。このICは発熱が少なく、かなり音量を上げてほんのり熱くなるだけなので、ヒートシンクは無くても大丈夫でしょう。

この段階で音出しです。テスト用スピーカーを接続してみます。このアンプは電源投入時にスピーカーから「ポツ」というポップ音がします。わずかな直流漏れがあるのかもしれません。



【写真6】音源、スピーカーをつなげて音だし。大型の3wayをつなげても面白い。

音源とスピーカーを接続して完成です。今回は汎用で使えるようにボリュームをつけましたが、P C専用で使う場合には、ボリュームを省略してもいいでしょう。A Cアダプタ方式にしたので、普通のトランス式電源の他に、スイッチング電源やバッテリー駆動を使用して音比べも可能です。

音質は非常に正確でクッキリハッキリした印象です。情報量も多く、サイズから想像できない音が楽しめます。

☆更に手を加えてみよう！

このデジタルアンプキットはそのままでも充分いい音を楽しめますが、さらに手を加えるのもまた面白いです。ここではそのポイントをご紹介します。ただし、くれぐれも安全に注意してください。あと、基板のパターンが剥がれやすいので何度も半田づけをする際には注意してください。

①電源コンデンサの大容量化・OSコン化

C5、C6の電解コンデンサは電源の平滑コンデンサです。これを大容量にすればするほど、低音に余裕が出ます。また、OSコンにするのもいいでしょう。トランス式電源ならば、3万 μ Fくらいつなげても平気ですが、スイッチング電源の場合には程ほどにしておいた方がいいでしょう。また、フィルムコンをパラに接続

すると瞬発力が出るでしょう。

②入力に直列の抵抗を交換

R1～R4の抵抗は音楽信号が通りますので、これをオーディオグレードに変更すると良いと思われます。リケノームの1/2Wなら、基板のパターンにぴったりおさまります。ヴィンシェイなど、いろんな抵抗を聞き比べてみるのも楽しいでしょう。

③インダクター交換

L1～L4のコイルをさらにグレードの高いものにすると、音の詰まった感じが取れて伸びやかになります。東光のDASLあたりが通販で購入できて便利です。

(<http://www.koei-denshi.com/>) 通販メニューには入っていませんが、メールで申し込みは購入できます。また、どうしても手に入らない場合には1mmくらいのエナメル線を自分で巻いて空芯コイルを作るという手もあります。フィルムケースに15回ほどきっちり巻きつけると、だいたい10 μ Hになります。空芯コイルの巻き数を計算してくれるソフトもあります。(<http://www.vector.co.jp/soft/win95/business/se192441.html>) ただし、この部品には大電流の高周波が流れるので、周囲に強烈なノイズを撒き散らします。また、出来るだけ配線は短くする必要があります。さらに、空芯コイルは音がきつuitとを感じる人もいます。よって、出来ればしっかりシールドしてある市販品を使いましょう。

④入力カップリングコンデンサ交換

C2、C3のコンデンサは音楽信号が直接通りますので、これを交換すると音が変わります。ただ、良く感じるかどうかは個人の好み

によって違うので一概にはいえません。フィルムコンデンサやOSコンなど、いろいろ聞き比べてみましょう。

⑤+5V 電源の独立供給化

TA2020-020の30番ピンからは+5Vの電源が出力されており、これが基板パターンを通過して2番ピンと8番ピンに供給されています。ここに低ノイズの電源を供給すると、ぐっと音が静かになります。このパターンをカットして途中でフィルタを挟むだけでも効果があるようですが、バッテリーで給電したらかなり効果がありました。レギュレータで独立給電にするのも面白いかも知れません。

☆オーディオ工作のススメ

オーディオ工作というスピーカーエンクロージャーの自作と真空管アンプが二大巨頭でしたが、最近はD/Aコンバーターやデジタルアンプなど、デジタル物もじわじわ広がってきました。デジタルアンプは、真空管アンプに比較して、高価な真空管やアウトプットトランスなどは使わないですし、アナログアンプのような大掛かりな放熱設計もいりません。難しそうなデジタルアンプですが、今回製作したようなキットを使えば、案外ほかのオーディオ工作よりも手軽と言えるでしょう。自作のオーディオでお気に入りの音楽を聴くのもオツなものです。

<参考文献>

- ・トランジスタ技術 2003年8月号 デジタルアンプ特集
- ・2ちゃんねる(www.2ch.net/) ピュアオーディオ板・トライパススレッド たくさんのアイデアを教えてくださいました。この場を借りてトライパススレッドの皆さんに深く感謝致します。

MPROM制御ソフトウェア 開発とハードウェア開発 の手順と解説☆



Written by もろぼし☆らむ
ある意味ネタ切れ位

今回はついにネタなしとなってしまいました(汗 それなりに時間があって興味が向いていても、いまいち本に書けるようなネタがないというのが原因なわけです。ここ最近では、RF 系でいろいろ遊んではいますが、所詮は実験レベルのものという感じです。

そんなわけで、今回は穴埋め企画(笑)として初心者向け中規模のマイコンシステム開発方法について紹介したいと思います。開発方法については、昔本書で紹介しました MPROM のシステムを取り上げて、どうやって比較的規模の大きなこのシステムを作り上げていったかということを解説します。

はじめに

ある程度単機能の自作モノですと、製作のテーマとなることに絞って考えれば大体事足りるのですが、MPROM のようにハードやソフトが十分に重く比較的規模が大きくなってくると、開発手順をきちんと考えて開発を行わないと、いつまで経ってもモノができません。今回は、あまりそういうことが書かれることが少ないようですので、それを中心に解説していこうかと思います。

ちなみにMPROMを知らない方や、どすぶいだによ1号を読んでいない方向けに簡単に解説しますと、いわゆるCD-ROMに記録されたMP3ファイルを再生する装置です。一応、ChaN氏のWebにPDF版の1号が掲載されていますので、この記事を読む際に併読して頂けるとよりわかりやすいかと思います。

仕様を決める

まずはこれです。とにかく、自分のやりたいことを見極めておく必要があります。そうでないと、どうでもいい部分に力を注いでしまい、開発に向けるパワーの配分に失敗してしまいます。また、どうしてもハード上、実現が難しかったりするものに対して、どの程度で妥協するか、あるいは別のアプローチがないか、などを考えるベースになります。

MPROM の場合は、以下の点を最低限実現することとしました。

- MP3 ファイルの再生(320Kbps まで)
- ドットマトリクス LCD によるグラフィカルな表示
- 漢字表示のサポート
- 最低限 ISO-9660、Joliet ファイルシステムのサポート

仕様を実現するハード構成を決める

さて仕様を決めたら、それを実現するハード構成を決めます。MPROM でのキーデバイスは次の通り。

- CPU V53A 12.288MHz(原発 24.576MHz)
- ROM Nor Flash ROM 8Mbit
- RAM SRAM 4Mbit
- 制御 CPLD
- 表示 ドットマトリクス LCD
- MP3 Micronas MP3 chipset
- CD-ROM ATAPI 接続の CD-ROM ドライブ
- KEY 8255 互換 PPI

まずはコントロールする CPU です。この MPROM では x86 系で高速な NEC の V53A を使用しました。当時 H8 などの CPU も出回っていましたが、以下のポイントで使うことにしました。

- とりあえず、x86 系は使い慣れていたのと、自作のアセンブラグラフィックライブラリを持っていた
- デバッグの容易さ(AT 互換機でのデバッグ)
- UNICODE 解析のための変換表、あるいは漢字を表示するためのフォントを格納する大きなメモリ空間
- ATAPI CD-ROM をアクセスするには CMOS-TTL 変換とアドレスデコーダを仕込むだけでよい

上記の内容で、ソフト開発の面を考慮してあるのは重要です。勿論ハード開発も大切ですが、後工程にソフト開発が控えていることをちゃんと考えておかねばなりません。この際、どれだけデバッグが簡単にできるか、という点は全体の開発効率を考えると、非常に重要な要素になります。

動作クロックは 12.288MHz(原発 24.576MHz/2)としています。CPU は 16MHz 版ですので、まだ余裕はあるのですが、これ以上上げても ROM のウェイトを増やさなければな

りませんし、RAM もついてこななくなりますので、あまり意味がありません。また 12.288MHz としたのは、このクロックをタイマーで分周してRS232Cの基準クロックを作る場合、最大通信速度の38400bpsが作れるからでもあります。

主要なロジック部分は、さすがにこの規模のシステムを標準ロジック IC で作るとあまりに大きな基板になってしまいますので、CPLDを使用することにします。実はCPLDを使うのも、HDLをまともに使うのも初めてでしたので、あまり凝ったことをするとハマることが予想されました。このため、MPROMではあまり大きな機能は極力CPLDに入れないことにしました(^^;; とりあえず、自分の能力をちゃんと見極めることは大切です。

さて、表示です。表示はLCDを使うことは決めていましたが、そのためのハードウェアが問題になります。MPROMで使ったLCDパネルは、組み込みではちょっと使いにくいVRAM非内蔵のもので、表示中は常に画面情報をLCDに対して送り出さなければならないタイプです。そのため、常にRAMをアクセスして表示情報を読み出し、LCDに送り出すという回路をいれなければなりません。しかし、ちょうど手持ちで持っていたものがこれだけなので、他に選択の余地がありません。このようなタイプのドットマトリックスモノクロLCDパネルの入手は、一般には難しいものなのです。

そんなわけで、手持ちであるこのLCDを使ってなんとか表示しなければなりません。このLCDを使う場合、一番システムの的に負荷が軽いのは、独立した表示専用VRAMを接続し、CPLDに表示するためのVRAMへのタイミング生成をすべてやらせて、CPUはCPLDの中にあるSRAMインターフェイスを経由して、専用VRAMにアクセスするというものです。こうすると、CPUのアクセスするメインRAMに表示のためのアクセスが発生せず、独立したVRAMのみにアクセスが発生するためパフォーマンスはあがりますが、RAMを追加する必要がありますし、回路規模が大きくなるため、CPLDに入らなくなる可能性があります。

このためMPROMでは、メインRAMにVRAMの機能も持たせることにしました。これをV53AのDMA機能を使って実現することにします。これにより、CPLDはCPUに対してDMARQとDMAKを処理し、LCDに対してはフレーム信号を出してあげるだけでよいことになります。もし、DMAを使わずCPLDだけで行くと、CPLDアクセス中のCPUとのバスアクセスの調停回路(バスアービター)など、かなりいろいろと面倒なものをいれなければなりませんが、DMAであればある程度はCPUが勝手にここはやってくれますので、かなり回路を簡単にすることができます。すでにあるものは、なるべく使いましょう(笑)

MP3デコーダとアンプのICはMicronasのチップを使います。というか、これが手に入ったからMPROMを作ることになったようなものですが(^^;; これらのICを制御するのに

必要なのは、制御用のI²Cバスとデータを転送するシリアル転送バスの実現です。I²Cバスは、組み込み系のCPUでは最初からサポートしているものもありますが、V53Aではそんなものはありません。

I²Cバスの制御回路をCPLDに組み込むという方法も考えられますが、あまりに規模が大きすぎます。このため、MPROMでは完全にソフト制御することにしました。つまり、転送データを1ビットずつソフトで立て、さらに転送クロックも1ビットずつ作ります。I²Cバスは規格上最大でも400KHz程度でしか動きませんし、マスタデバイス=CPUが制御の実権を握るバスです。やりとりするデータもちょっとしたコマンドだけです。ソフト制御により少々遅くなっても特に問題ありません。ただ、バスがプルアップされた状態で、出力をオープンコレクタで制御しますので、出力に外付けトランジスタを使い、入力にはCPLDの端子を使う変則的なものとなります。CPLDは、これらの端子の入出力を切り替えたり、端子をCPUからI/Oポートのレジスタとして見えるようにするだけです。

逆にMP3ファイルデータを送るシリアルバスは、単に連続でデータを垂れ流すだけの簡単なものです。もちろんI²Cバスと同様にCPUでソフト的にシリアル変換して1ビットずつ送るという手もありますが、この転送では高速性がなにより要求されますので、ソフトによってデータを送るのは現実的ではありません。そのため、ここもV53AのDMA機能を使い読み込んだデータをシリアル変換にしてMP3デコーダに送るようにします。これにより、CPUはMP3データのあるメモリをDMAにセットさえすれば、あとはハードウェアがデコーダにデータを送ってくれることになります。

さて、メモリです。今回はx86系のCPUですので、1Mバイトのメモリ空間が使えます。最初にターゲットとして考えたMPROMの仕様上、ROMとして必要なサイズは結構大きくなります。特に以下の2つのデータベースが大きいです。

- ・漢字フォント格納領域 (約200Kbyte)

- ・Joliet ファイルシステム実現のための UNICODE-SJIS 変換テーブル (128KByte)

合計すると300Kバイト以上にもなってしまいます。このため、最低ROMはプログラムも入れて512Kbyte(4Mbit)は必要であろうことは見込まれます。MPROMでは、余裕をみて8Mbitのフラッシュメモリを使うことにしました。そして、残りの空間512KをまるまるSRAMに割り当てます。x86系CPUの場合はリセット番地はFFFF:0000ですので、ROMはアドレス上位(80000-FFFF)に、RAMは00000-7FFFFに割り当てるようにCPLDにアドレスマッピング回路を実装します。さらにROMは高速化のため16ビットバスで使います。一応、使用したROMは8/16bit共用タイプでいいしたので、どちらでもよいのですが、V53Aはダイナミックバスサイジング機能を持っており、簡単に動的にバ

ス幅を変えられますので、特にバスサイズの異なるデバイスが混ざっても関係ありません。ただ、CPLD に対して、特定のデバイスがアクセスされたときに、適切なバスサイズを選択する回路(CPU の/BS 端子の制御)が必要になります。

使用する RAM は一般的な 8 ビットバス接続の 4Mbit の SRAM で、アクセススピードは 70ns です。あまりアクセスタイムが長いとノーウェイトで動作させられなくなり、LCD へのデータの DMA 転送に影響をきたします。このため、CPU がノーウェイトでアクセスできるぐらい速い RAM を使う必要があります。

CD-ROM ドライブについては ATAPI 接続タイプを使います。元々、IDE のバスは HDD コントローラの LSI につながるバスをのぼしたものだという歴史的経緯のため I/O の読み書きをするがごとく、アドレスとライト、リード、チップセレクト信号が出ており、その読み書きのタイミングも 86 系のバスとほぼ同じです。このため、適切にデバイスのアドレスマッピングを CPLD で行うだけで、普通にドライブに対して、コマンドやデータの読み書きを行うことができます。ただちょっと違うのが、バスが TTL レベルで動作することです。このため、バスラインを HCT タイプのバストランシーバを経由して接続することになります。

さらに ATAPI ドライブの場合、アクセス方法が単なる I/O の読み書きだけで行う PIO タイプから、DMA を使って高速転送するものなどいろいろあります。これらは、ドライブによってサポートしているものが異なります。MPROM で使用したドライブはスリムタイプのノート用ドライブですが、PIO モードまで対応しか対応していませんので、これで制御を行うこととします。よって、DMA を使いません。

残りがキーになりますが、ここは安直にパラレル I/O の LSI である i8255 互換品を使っています。あんまりキーもないので、とりあえず 1 ポート 1 キーという、恐ろしく贅沢(無駄とも言う(^^;;))な構成としています。

さらに、上記のような実際に使用する機能だけでなく、開発用のデバッグインターフェイスをつけます。といっても大それたものでなく、単なる RS232C ポートです。このポートを使って、デバッグ用モニタの表示や入力を行います。いきなり LCD をつけてやるというのは結構無謀でし、なにより表示してもすぐに画面から流れたりしますから、あまり実用的ではありません。

以上のように構成を考えて、わかりやすいブロック図面にしておきます。これを元に回路を作ります。

回路の設計と組立をする

設計する MPROM では、特に以下の点に注意が必要になります。

1. バスサイズが異なるデバイスが混在するため、アドレスバスとデータバスの接続に注意する
2. 電源電圧の異なるデバイスがあるので、各デバイスの信号レベルと、電源の切り分けに注意する
3. ピン数の多い石が多いので、端子処理を確実にする
4. アナログ系の回路とデジタル系の回路が混在するので、それぞれの配置と電源周りの回路に注意する

まず、1. です。今回は ATAPI CD-ROM のインターフェイスとフラッシュ ROM については 16 ビットアクセスとなります。ATAPI インターフェイスは特に問題ないですが、フラッシュ ROM の接続は間違えやすいので注意が必要です。アドレスラインは 16 ビット接続のとき、CPU の A1 にフラッシュ ROM の A0 をつなぐようにしなければなりません。このときのフラッシュ ROM のアドレス指定はワード単位だからです。これを間違えると全く動きません(^^;;

次に2です。今回は電源電圧が異なるデバイスが混在した作りになります。使用したデバイスのうち、LCD パネルと MP3 デコーダと専用 DSP が 3.3V 動作で、CPU などのほかのデバイスは 5V で動作しますので、信号のインターフェイスに対してはすべて電圧変換して接続します。使い易いデバイスでは、そのデバイスの電源電圧よりも高い電圧の信号を受け付けるようになっていますが、今回はどちらもそのようにはなっていないので、5V の信号を入れると一発でデバイスが昇天する可能性があります。このため、LCD についてはレベル変換 IC をかまして接続し、MP3 デコーダなどに対してはトランジスタと LCD の電圧変換で使った変換 IC のあまりのピンを使って接続します。レベル変換 IC は2つの Vcc 入力を持っており、各々双方向にレベルの異なる信号の接続ができるようになっています。MP3 デコーダで使う信号のうち、5V 系の回路と繋がるのは、I²C バスと MP3 データ転送用シリアルバスしかありません。I²C バスはオープンコレクタ接続ですのでトランジスタを使い、シリアルバスは LCD の接続で使用したレベル変換 IC の余りのピンの出力を入れることになります。

次に3. です。とにかく CPU はピン数が 100 ピンを越えるセラミック PGA タイプの石のため、多くの端子が存在します。それぞれにおいて、使用する際に使わないピンをプルアップ/プルダウンするかがデータシートに書いてありますので、もれなく確実に処理します。V53A の場合、ほかにシリアルなどの内蔵ペリフェラルの割り込み端子が内部接続でなく外部で接続しますので、きちんとそれらを割り込みコントローラの割り込み入力端子に接続します。

割り込みはシリアル、DMA 完了割り込み、ATAPI バス割り込み、さらにタイマー割り込みの4種類を使います。割り込みは優先順位があるため、きちんと機能的に優先しなければならないものを上位の割り込みにしておく必要があります。V53A では割り込みを発生する機能のある内蔵ペリフェラル

においても、まったく割り込みコントローラに内部で接続する機能はなく、それらの割り込み要求の出力は外部端子に出ています。このため、それらの端子をすべて割り込みコントローラの割り込み入力端子に接続する必要がありますので、回路設計の段階でどの割り込みをどのくらいの優先レベルにするかを考慮しておく必要があります。

前述の機能の中で、必ずシリアルは一番上に設定します。シリアルは割り込みが遅れると、データを取りこぼす可能性があるためですが、デバッグ時にしか使いませんので、実使用上では無視して構わなくなります。次に DMA 完了割り込みを入れます。この割り込みで、MP3 デコーダへのデータ転送準備を行いますので、通常動作時では一番優先度を高くします。その次に ATAPI の割り込みを入れます。この割り込みはコマンド発行後に動作準備が完了したときなどに発生する割り込みです。これは、CD-ROM の読み出しで使用しますし、ある程度時間がかかっても構わないものですので、低めの優先順位にします。そして、最後にタイマーを入れます。このタイマーは単にウェイトを作るためのカウンタを数え上げるだけの機能しかありませんので、あまり優先度が高くなく、すこしぐらい割り込みを落としても問題ないので、一番下に入れます。

最後にアナログ系です。アナログ系は電源周りと、電源の分離が結構重要になります。特にオーディオのアナログ回路にデジタル回路が混在すると、デジタル回路のノイズを拾って、音質が悪化します。これはデジタル回路では矩形波が信号として大量に流れますが、それには大変周波数の高い信号成分を多く含んでいるためです。そのため、プリント基板上で実装する場合にはグラウンドパターンで、デジタル回路とアナログ回路を分離します。また、電源に十分なコンデンサとコイルでデジタル回路からの電源を経由してのノイズの進入を防ぐようにしますが、今回はユニバーサル基板に入れていますので、あんまり考えてません(;; ただ、

MP3 のアナログ電源に対しては十分な容量のコンデンサと、コイルをいれてノイズに対処しています。

ROM と RAM を動作させる

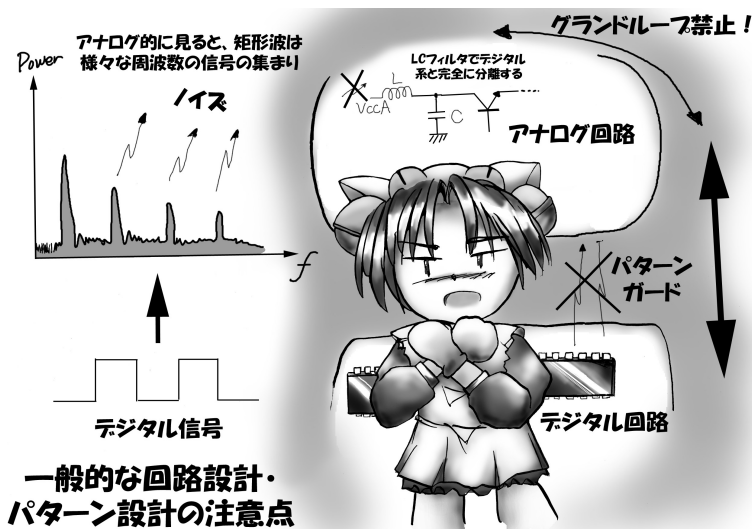
前項のことを踏まえ、回路設計と製作、組み立てを終えた後に電源入れて、燃えることもなく(;;大丈夫だな、と確認したあと、ROM に簡単なプログラムを入れて動作するかをチェックします。まず、リセット番地から無限ループするソフトを書いてみて動かしてみます。ロジックアナライザがあれば、そのときにアドレスバスやデータバスの動きをみて、思った通りにバスが動いているかを確認すれば、プログラムが正常に動作しているとわかります。たとえば、無限ループしているときには、プログラムのあるアドレスを繰り返し CPU が実行する命令を読んでいる様子が見えます。

さらに CPU 内蔵 I/O すべての初期化モジュールを作り、メモリ関係のウェイト設定などをきちんと行ったあと、ROM 全域に ROM ライターでデータを書き、そのチェックサムをとってみたい、RAM 全域にテスト書き込みと読み出しをして、正常に読み書きできるかを確認します。今回は RAM に SRAM を使用しているので関係ありませんが、DRAM を使用する場合にはリフレッシュがきちんと行われているかをみる必要があります。この場合は、全域に特定のデータを書いた後、何十秒〜1分ぐらいは RAM をアクセスしないようにソフトループで待ってからベリファイしてみる、という感じでチェックします。これは DRAM の場合、リフレッシュが正常に動いていなくても、アクセスするだけでリフレッシュがかかるのと同じような効果が得られるため、すぐにベリファイを行うとリフレッシュ動作も含めた RAM の動作チェックにならないからです。

なお、V53A などの I/O をいろいろ内蔵している組み込み用途向け CPU の場合、リセット時の設定内容が大変多いので、念入りにデータシートをみて初期設定は漏れなく行いましょう。

で、念入りにデータシートをみて初期設定は漏れなく行いましょう。

この段階で全くシステムが動かない場合には、それ以降どんなプログラムを書いても無駄ですので、ここで確実に動かしましょう。動かない場合の原因は、ソフトでは起動時の CPU の I/O の初期設定のミス、ハードではバスの結線間違いや端子処理の間違い、設計間違い(;; などいろいろ考えられますが、こういうときは最小限の構成にして (CPU と ROM、RAM、CPLD)、地道に問題点の切り分けを行って、1 つ 1 つ確認して潰して行くしかあり



ません。近道はないので、ハマったら頭を休め、風呂に入るなどリラックスして、じっくりと取り組みましょう。ロジックアナライザやオシロスコープなどがこういふときにあると、バスの動きなどがヒントで問題がわかったりしますので、いろいろな信号をじっくり観測してみることも必要です。

8086 の組み込みソフト作成環境

MPROM のプログラムの作成には、LSIC 86(エル・エス・アイジャパン)というコンパイラと Turbo Assembler(Borland)というアセンブラを組み合わせて使っています。

最近では H8 や PIC、AVR などが流行ですので、86 系の組み込みプログラム作成のためのツールが少なくなっているようですが、上記の開発ツールの組み合わせは、国内では比較的入手しやすく、安価で使いやすい部類のものであります。

LSIC 86 については無料で配布されている試食版を手に入れることができます。このバージョンの制限は、Small モデル(データ、プログラム各 64K バイト)のプログラムしか作れないというもののだけです。ただ、MPROM 程度のものであれば Small モデルを越えるようなプログラムサイズにはなりませんので、これで十分です。x86 系では、Small モデルより上のモデルでセグメントを越えるプログラムを作ろうとすると、ポインタを 32bit の far ポインタで持つ必要があるため、余り効率がよいものではなく、コードサイズも大きくなります。

MPROM では、プログラムのメモリ上の配置は表1のようになっています。MS-DOS などで作動するプログラムを実行する場合、これをメモリ上のどこに配置するかは、OS がメモリの使用状況により決めて再配置します。これをリロケーションといいます。EXE ファイルの中には、セグメント情報のオフセット位置が配列になってかかれており、OS はプログラムを展開するときに、その配列の内容に対応した部分を書き換えて、プログラムの位置を直します。例えば、MOV AX,0000 / MOV DS,AX というコードがあるとき、本来 0000 のところには DATA というセグメントが入るべき場所だとすると、プログラムファイルにおける 0000 の位置が EXE ファイルに書いてあり、OS はそれを見つけて 0000 を OS が配置したセグメントに書き直すという感じです。

しかし MPROM では、ROM にプログラムデータを焼き付けますので、後から書き直すことはできません。そのため、すでに ROM を焼くためのバイナリ、またはインテル HEX ファイルとなった段階で、これらのセグメントのアドレスは確定していないといけません。これを行うのが LSIC 86 では LLD というプログラムです。本来はリンカーなのですが、組み込みプログラムに必要な ROM 化を前提としたロケーション機能(アドレス配置機能)も持った優れたものです。たとえば、セグメント DATA をセグメント 400h に割り当てるとき

は、-TDATA 04000 という風に設定します。設定は物理アドレス指定ですので、4ビットを左シフトした値を入れます。

```
MONSEG = -TTEXT 84000
ASMSEG = -TSYSRAM 00400 -TBOOT FFFF0 -TSETUP
80000 -TBIOS 80800 -TUNICODE_SEG 90000
-TGRAPHICS_SEG E0000
LINKSEG = -Fh -TDATA 04000 -TXSTACK 0B000
-TFAR_BSS 09000 -s 4000 -M
```

makefile では上記のように設定されます。これをリンクに渡すと、表1のようなマッピングになります。バイナリファイルとして実際に作成すると、1M バイトのメモリーイメージがそのまま出力されますが、これでは使いにくいので別のインテル HEX 形式というテキストファイルで普通は使います。

LLD での出力指定で -Fh と設定すると、このインテル HEX ファイルを LLD は出力します。このファイルは、チェックサムとプログラムとその配置アドレスが書かれたテキストファイルで、ROM ライタなどで使用される一般的な形式です。これはバイナリファイルのようにすべてのイメージが出るのではなく、ROM として書かれる場所のみのデータが出力されます。これを ROM ライタに送って書き込むわけですが、実際にハードウェアが CPU にマッピングするアドレスと ROM の内部での物理アドレスがことなることがあります。

MPROM では、意図的にこのような不整合が起きないように、80000h 以降の ROM を CPU の実アドレスと同じになるようにマップし、80000h 以下は CPU では 100000h 以降に見えるようになっています。このため、このまま HEX ファイルを ROM ライターで焼いても問題のないようにしていますが、もし逆に ROM の下位の 00000h-7FFFFh をプログラムエリアとするならば、ROM ライターに転送したあとに ROM ライターの MOVE 機能で移動するか、HEX ファイルを作成したあとに、HEX ファイルのセグメント指定フィールドの値を書き換えて、下位に移動するようにしないといけません。

このように組み込み系のソフトを作る際には、常に実行アドレスだけでなく、ROM 上の物理アドレスが CPU ではどこに配置されるのかを意識しなければいけません。これは 86 系以外の CPU でも同様です。

C 言語のプログラムを動かす

C 言語で設計したプログラムを PC 以外の環境で動かすには、いくらか前準備が必要です。先にロケーションの所で解説したように、PC 上と異なり MS-DOS のような OS の動作していない組み込みシステムでは、本来 OS がやっていることを自分でやらなければならない場面が多くあります。C 言語のプログラムを動かす、ということもその一つです。

C 言語で作られたプログラムは、普通はいきなり main()関数から動作するものではありません。一般にスタートアップルーチンという部分が動作したあと、その中から main 関数が呼び出される形になっています。

このルーチンは、一般的に次のようなことを行います。

- ・リンクされたライブラリの初期化
- ・BSS 領域(未初期化変数領域)を確保し、0 で埋める
- ・スタックポインタの初期化
- ・main 関数の呼び出し

MS-DOS のプログラムでは、他に argc,argv の引数に渡すパラメータを設定したりもします。

LSIC86 では、スタートアップルーチンは start.p86 というファイルに記述されており、先に説明したことと同様のこと

を行わなければなりません、特に ROM 化する C 言語のプログラムでは、さらに次のようなことを行います

- ・必要であれば、ハードウェアの初期化を行う
- ・DATA 領域(初期化済み変数領域)に必要とされる初期値をすべて書き込んで、初期化を行う。
- ・割り込みを処理する関数にかぶせる関数を設定する。この中から、割り込み発生時に処理を行う C 言語で書かれた関数を呼び出す(LSIC-86 の場合)

これらについて、ソフトを組み込むハードウェアの構成によって、スタートアップルーチンを書き直します。普通の PC のプログラムでは、このようなことをすることはまずありませんが、組み込み系のソフトを作る場合には、この部分の書き換えは避けて通れません。

さらに割り込みを処理する関数では、Turbo C などでは interrupt 修飾子を使って直接関数を割り込みに対応できるようになりましたが LSIC 86 ではそのようなものはありません。どうするかというとスタートアップルーチン内部に割り込みを受ける関数を書き、そこから C 言語で書かれている実際に処理を行う関数を呼び出します。割り込みを受ける関数では、全レジスタの待避と、必要であれば割り込みコントローラへの EOI の発行を行います。割り込み関数で使うベクタの設定も、スタートアップルーチンの中で行います。

設計した MPROM では、必要な 3 つの割り込み関数(DMA,CD-ROM,タイマー)のラップ関数をスタートアップルーチンに仕込んでおり、それぞれ C 言語側の関数 dma_intr() , cdrom_intr() , timer_intr() の呼び出しを行います。

シリアルを動かす

最低限のプログラムが動作するようになったところで、まずはシリアル RS232C 端子を動くようにします。組み込み系に限らず、ソフトを設計する場合にはまずきちとしたデバッグ環境を整える必要

表 1 MPROMのプログラムマップ

FFFF:FFFF	リセット番地 BOOT		MBM29F800 (フラッシュROM)	100000
FFFF:0000				
F000:0000	空			
	グラフィックデータ GRAPHICS_SEG			
E000:0000				
D000:F000	ANKフォント			
B000:0000	漢字フォント			
	UNICODE→SJIS 変換テーブル UNICODE_SEG			
9000:0000				
	C言語プログラムエリア CGROUP(TEXT)			
8400:0000				
	アセンブラプログラム (BIOS)エリア BIOS			
8080:0000				
	アセンブラプログラム (初期化)エリア SETUP			
8000:0000				
7000:C000	プレイリストバッファ			
7000:4000	GUI用ウィンドウバッファ (2画面分)			
7000:0000	表示用VRAM			
6000:0000	リードバッファ1			
5000:0000	リードバッファ0			
4000:0000	ローカルリードバッファ			
3800:0000	ディレクトリテーブル			
3000:0000	パステーブル			
	空			
2000:0000	(デバッグモニタ動作時には プログラム実行領域)			
1000:0000	空			
	C言語プログラム データ領域 DGROUP (DATA/FAR_BSS/XSTACK)			
0400:0000				
0040:0000	BIOSワークエリア SYSRAM			
0000:0000	割り込みベクタ領域			

CPU論理アドレスマップ

デバイスごとに対応するアドレス

があります。Windows などでは、PC 上でよいデバッグが動作しますので、それを使えばいいですが、組み込み系のハードではそんなものはないどころか、表示すらないことが多いです。このため `printf()` などの標準表示関数をそのまま使うことすらできないので、何かモニタリングする機構を別に外付けし、それを使った表示、入力関数を追加、改造して、デバッグを行わなければなりません。お金があれば、ICE というデバッグ用の機材を使えばいろいろできますが、個人でそんな機材を持っていることはまれです。大体この手のものは安くても数十万円はするのが相場ですので、そのため手軽な方法として、シリアル端子を使って通信ソフトで組み込みソフト内のデバッグとやりとりをするのが一般的です。こうすれば、変数の内容やメモリの内容をプログラム上からシリアルで送受信することで、接続した通信ターミナル上でそれらを見たり、操作することができるわけです。これに使用するシリアル BIOS は、単純に文字データをやりとりするだけです。簡単なハードフロー制御ができる送受信処理さえ実装できれば問題ありません。このようなシリアル BIOS の実装方法はどんな CPU のものでも基本的には同じですので、一度は自分で作っておくと別の CPU になったときでもすぐにそのシステムに対応したものを作ることができます。実際の設計では、パフォーマンスを重視すれば、送信も受信も割り込みで処理した方がよいのですが、送信は特にポーリングでも問題ありませんので簡単に済ませましょう。受信は割り込みを使わないとデータの取りこぼしが出る場合がありますので、必ず割り込みで実装しなければいけません。

作ったあとは、簡単なループバックプログラム(送信と受信端子をショートして、自分で送信したデータを確実に受信できるかをテストするプログラム)で動作確認をしましょう。

細かい回路とソフトのデバッグ

さて、シリアル端子が動くようになったことで、通信ソフトを

使って組み込んだソフトとやりとりができるようになったところで、先に挙げたデバッグモニタというものを作ります。ホームページにアップロードしてあるファームウェアソースでは `debugmon.c` がこれに当たります。デバッグモニタは作る人によっていれるものは細かくは異なりますが、基本的には以下のようなものは最低限入れるのが一般的です。

- ・指定のメモリ内容をダンプする機能
- ・指定のI/Oポートの読み書き機能
- ・インテル HEX ファイル(モトローラ S ファイル)を受信して、メモリへ書き込む機能
- ・指定の番地へ実行を移す機能

デバッグモニタは、システム完成時でも特定の端子をショートしたり、特定のキー入力をする事で起動するようにして、問題が発生したときにその場で調査がすぐに行えるようにしておくようになっていることが多いです。MPROM では、デバッグ途中ではリセットを行うと、次のようなメッセージがシリアル端子を通して出力されました。

V53A Monitor program by myan

COMMAND>

"COMMAND>"はプロンプトです。ここでコマンドを通信ソフトから入力することで、対話的にテストを行うことができます。たとえば、"OB 50 3a" と入力すれば、50h ポートに 3ah を出力するという感じです。このような I/O ポートの入出力機能を使って、CPLD や外部のインターフェイスの簡単な機能テストを手軽に行うことができます。

また、コンパイラで生成したプログラムのインテル HEX ファイルを通信ソフトで RAM 上にダウンロードし、実行できるようにすることで、フラッシュ ROM を焼くことなしにいろいろなテストプログラムを実行することができるようになっていきます。デバッグを行うときにいちいちフラッシュ ROM を焼くのは手間ですので、これができることでかなりの設計の効

表2 MPROM デバッグモニターコマンド (抜粋)

コマンド名	機能	対応するテスト
IB (IW)	指定のポートからのデータの読み出し(B:Byte,W:Word)	I/O の読み出し汎用テスト
OB (OW)	指定のポートへのデータの書き込み(B:Byte,W:Word)	I/O の書き込み汎用テスト
D	指定のメモリ内容をダンプする	メモリ内容確認
G	指定のアドレスへ実行を移す	RAM にロードしたプログラムの実行
L	インテル HEX ファイルをシリアルからダウンロードする	デバッグターミナルからのプログラムの読み込み
SENDDAC	メモリ上にある MP3 データをデコーダに出力する	MP3 デコーダ関連の CPLD の機能テスト
SENDDSP	MP3 デコーダや DAC に I ² C バス経由でコマンドを出力する	MP3 デコーダ関連の I ² C バスコントロールの機能テスト
PLAYM	CD-ROM 上の MP3 ファイルを演奏する	ファイル読み出し機能と MP3 演奏機能を組み合わせたテスト

率があがります。

このモニタを使って、まずは簡単な I/O の読み書きのテストを行います。例えば、特定のポートに書き込んで、きちんと目的の端子がそのような動きをするか、読み出した値が意図した値となっているかなどです。これにより、基本的なハードウェアの問題はかなり修正することができます。

それが終わった後は、単機能モジュールを組んで、そのプログラムをデバッグモニタを使い RAM にダウンロードし、実行テストをしたり、デバッグモニタに組み込んでテストしてみます。これは、最終的にはプログラムは ROM で動かしますので、SRAM 上で動作するデバッグモニタのプログラムロード機能で実行した場合とは、実行速度などの環境が異なるため、デバッグ過程の中できちんとモニタ側へ組み込んでテストする必要があるからです。

MPROM では、表2後半のようなコマンドを実装してテストを行っています。それらは MPROM では基本的な単機能の骨格となるモジュールです。そのため、これらを確実に動かささえすれば、あとはそれらを拡張してまとめるだけで1つのシステムができあがってしまうわけです。

これらすべてを動かすのに、様々な細かいデバッグやソフト上の試行錯誤を繰り返しました。なにしろ作る機能があまりに多いため時間もかなりかかりましたが、所詮地道に1つ1つの機能を完成させていくことが完成への近道なのです。ただ、これらのデバッグが終わった後はハードウェアのトラブルに見舞われることもなく、実際に MPROM のコアの部分は、あっという間にできました。

GUI の設計

ハードウェアに依存するような機能は、デバッグモニタ上で大体作成させることができました。最後はそれら全部をまとめあげ、GUI のシステムを組み込み、実用に耐えるようにするだけです。設計した MPROM ではビットマップ表示が可能なドットマトリックス LCD を使用していますので、グラ

フィック周りのインターフェイスなど、アプリケーションレベルの設計が比較的大がかりなものとなります。

この LCD 表示用の VRAM は水平型の VRAM、つまり1ビットが1ドットで MSB 側が原点に対応した形になるように設計されています。MPROM の VRAM の場合、1バイト目の MSB のビットを1にすると、(0,0)のドットが黒で表示されます。このような形式は AT 互換機や PC-9801 などでも同様です。特に今回は同じ x86 系の CPU でするので、PC と MPROM は、単に表示サイズが違うだけのものと見なすことができます。このため、GUI の設計と基本的なデバッグは AT 互換機上で行いました。さらに CD-ROM のアクセスもポート番号やレジスタの位置がちょっと違うだけですので、最下層の制御ルーチンさえ差し替えれば、ほとんど V53A で動かすのと同じプログラムを PC 上でテストすることができます。このようなデバッグをなるべく簡単にする工夫は、開発効率に直接効いてきますから、目先の面倒くささより先(デバッグ中)の楽さを常に考えておくと幸せになれます。

なお AT 互換機でテストする場合は、各 VGA のレジスタを表3のように設定すれば OK です。こうすると、たとえば VRAM の領域に何らかの値を書くと、全プレーンに対して同じ値が書かれますので、MPROM と同様のモノクロ表示が行われます。このような VGA のレジスタの詳細が書かれた AT 互換機のグラフィック系の資料は最近入手が難しいので、見つけた時はすぐに捕獲しておいた方がいいでしょう。

表示系以外のデバッグモニタの入出力やキーについては、PC であればふつうにキーボードでエミュレーションできますので、モニタのキー待ち部分を `getc()` や `putc()` で置き換え、MPROM の PPI のポートをみてキー入力待つ部分も、`getc()` 関数で特定のキーを押したときに、特定のポートのビットがたっているように見えるモジュールに置き換えるだけですみます。

これらの PC 上でのエミュレーション部は、プログラムでは定義シンボルを変えることで、AT 互換機で動作するか、か V53A で動作するかを切り替えられるようにしました。さらに

表3 VGAレジスタの設定

グラフィックコントロール ポート 3FEH レジスタ番号 ポート 3FFH 設定値 シーケンサコントロール ポート 3C4H レジスタ番号 ポート 3C5H 設定値

	名称	レジスタ番号	内容	設定値
Graphics Control Register	Mode	05h	00h	書き込みモード=0、読み込みモード=0
	Set/Reset Enable	01h	00h	ラッチ全プレーンディスエーブル。(Set/Reset(00h)レジスタの内容を無効にし、VRAM へ書き込む値を有効にする
	Bit Mask	08h	FFh	全ビットイネーブル
	Data Rotate/Function Select	03h	00h	右ローテートビット数=0、論理演算=SET
Sequencer Control	Color Plane Write Enable	02h	0Fh	全プレーンイネーブル。0Fh を設定することで、VRAM にデータを書き込むと、全プレーンに対して同じデータが書き込まれる

表4 FONTX2 フォントファイルフォーマット

ヘッダ

オフセット	サイズ	内容
0	6	ID: 'FONTX2'
6	8	ユニークなフォントファイル名
14	1	Width: フォントの幅
15	1	Height: フォントの高さ
16	1	Code Type: 0: ASCII, 1: 漢字コード
17~	—	フォントデータ、 またはコード範囲指定データ

ANK フォント 以下のフォーマットで 00h~7fh 分並ぶ。

左上 MSB

X=7~0,Y=0	X=15~8,Y=0		X=x+7~x,Y=0
X=7~0,Y=1	X=15~8,Y=1		X=x+7~x,Y=1

..... × 高さ

※上のフォントデータフォーマットは、漢字フォントでも同様

漢字フォント

オフセット 17 より、コード範囲指定テーブルが入る。

オフセット	サイズ	内容
17	1	テーブルサイズ(N)
18	2	開始コード 1
20	2	終了コード 1
....		
(N-4)+18	2	開始コード n
(N-2)+18	2	終了コード n

フォントデータは オフセット N+18 より始まり、コード

範囲指定テーブルの順番で入る

AT 互換機で動作するように定義したときは Borland C++コンパイラでコンパイルし、実行ができるようにすることで、Turbo Debugger などの高機能なデバッガを使用した効率的なデバッグができるようにしています。

グラフィックや文字の表示

グラフィック表示に関する BIOS は、組み込みのシステムの場合1からすべて書かねばいけません。Windows などのように線を引く API もなにもありませんし、そもそもそのままだでは文字すら表示できませんので、フォントデータもすべて ROM に組み込んで1から表示するモジュールを設計する必要があります。AT 互換機は海外規格である以上実はそれと同様で、ハードウェア自身が日本語フォントデータを持っていません。このため、DOS 起動時にフォントドラ

イバがフォントデータをメモリ上に読み込み、それを使って画面にディスプレイドライバが日本語を表示するような構造となっています。このような仕組みは IBM が IBM DOS J4.0/V で V-TEXT 規格として実現し、さらにユーザーの手でそれを拡張したフリーのドライバ FONTX.SYS が作成されました。このドライバの ver.2.0 以降が使用するのが FONTX2 形式のフォントファイルです。この形式でフリーのフォントも公開されているため、自作に使用するには便利なものです。

MPROM のフォントデータは、このファイルからヘッダを抜いて、インテル HEXファイルへ変換するプログラムにより ROM データを作成しています。この FONTX2 形式のフォントファイル構造は表4のようになっています。そして、この形でデータへのアドレスを返す関数(drawbios.asm getfont 関数)を作成し、それを日本語フォント文字表示 BIOS(drawbios.asm DrawPutKanji 関数)が呼び出すことで、文字の表示を行います。

あとは、グラフィック表示になります。基本的に大きいフォント(時間表示など)は PC 上でモノクロの単純なビットマップデータに落として、それを BIOS により表示するようにします。グラフィック表示 BIOS では、それぞれのグラフィックには番号を振って、その番号を使ってグラフィックデータのあるアドレスを取得し、それを PutImage BIOS を使って表示するようにしています(drawlib.c,drawbios.asm)。

この他に、ウィンドウ表示などを行うため、ライン描画、ボックス描画の BIOS などを作成しますが、これらを描くための描画アルゴリズムは Bresenham アルゴリズムといわれるものが簡単で高速なため、普通はこれが使われます。ちなみにこのアルゴリズムの拡張を行うと円も描くこともできたりします。実際に実装したライン描画 BIOS が drawbios.asm の DrawLine に当たります。なるべく高速にするため、徹底的にレジスタのみを使っているものです。

完成!

GUI 系が動き始めたら、デバッグモニターで作った機能モジュールと統合して、全体で動作する形にし、細かい調整をすれば完成です。既にこの段階では、今までの手順を踏めばハードもソフトもほぼ問題なく動くレベルまでデバッグされていますので、どんどんコードを書いていくだけです。

そうしてできあがったのが、第一号で解説した MPROM のシステムです。規模の比較的大きなものでも、きちんと設計やデバッグのポイントを押さえ、一步一步を着実に動くようにし、さまざまな工夫を行えば、十分に自作を行うことができることがわかるのではないかと思います。みなさんも、ちょっと一つ大きなものの自作に挑戦してみるのもよいのではないのでしょうか。

変更履歴

2003年8月17日

コミケット 64 配布版（東京ビッグサイト）

表紙フルカラー、44P オフセット誌、頒布価格 350 円
【完売しました】

2007年1月28日

インターネット配布版 Revision 0

著者あとがき

ROM男さん

またもや参加してしまいました。一度コミケで何か出品すると次も何か出しくなってしまう。今回の“マクロクールライトみたいなもの”は技術的にはたいした事はないけど、見た目がすごく?て作りたいものを作った、といったところです。この世知辛い時代、だんだん自分のための時間がなくなってきましたけどまた何か作ったら本にしたいと思っています。

Kimさん

K i mですー、おひさしぶりです。

今回はデジタルアンプキット製作とデジタルアンプの紹介みたいな内容を考えていたら、トラ技で特集を組んでいたの、改造法紹介みたいな内容にしました。全て自分で実践した内容です。

みなさんも独自の工夫がありましたらぜひ教えてください。

ではまたー。(m_kobayashi@mvc.biglobe.ne.jp)

朱 -Aka-

にゃう〜ん☆
(似ない(´▽`))



朱に交われれば-Aka-くなる…ってことで、今やりのゲーム「朱」より、チュチュさんです。やっぱり触角付きの女の子って良いですね(うん)。でも、登場人物の中では際だってエロい?衣装デザインのような気が...(汗)。まあ、それはそれとして、まだやっていない人には激しくおススメです。何と言ってもサウンドが素晴らしい! (映画風で、生楽器に声楽曲多い)。おまけのサントラ目当てでも元取れると思いますよ(お。

ほかに並行してやっているのが「Arcturus」で、これもサントラが3枚組でボリュームたっぷり。秋には「英伝6」や「Ys6」が相次いで出るので、今年はゲーム三昧じゃ(^_^)。

おっと、話が後先になりましたが、今回の「どすぶいだによ☆3」はいかがだったでしょうか。私の分は2本とも光を応用した製作・実験ネタです。趣味の電子工作には、やっぱり光りモノが断然面白いですね(単に光りモノ好きなだけです(笑)。LEDファンについては、ページの都合でハードウェアの解説まで、ソフトウェアの部分が切れてしまいました。2P単位の割り当てでぴったりに納めるのは結構難しいです。あと 0.3P あれば全部入ったのですが、わかる人にはあれだけで十分でしょうから、まあいいでしょう(^_^;)。

ChaN

著者あとがき

おゆうさん

今回二度目の参加です。本稿のほうは失敗で終わっていますが、間もないのに後日談ができました(汗; 事前確認が不足していました。データシートの斜め読みだけで、肝心なところが抜け落ちていました..

今回試作で使用した USB-Serial 変換の IC, 原稿終わってから知ったのですが、どうやら FT232A の latency timer は 16 ミリ秒固定のようです..

数ミリ秒短縮したところで、1byte 毎に応答を待つようなプロトコルでは、全部こいつの周期でまたされてしまい、高速化が図れない模様です。

ライタからの応答が確実にタイマ周期に合致していても 33 秒程度 (2KB x 16 ミリ秒) かかってしまいます。二回に一度だけ二周期要したとすれば、今回の結果と同じ 49 秒程度となります。おそらくこれですね..

デバイスを FT232B にすれば、inf ファイル書き換えて 1mSec 周期にできるそうです。こちらで済みたいところもありますが... 普通の COM ポートで試してみれば速度 UP ができたかわかりますね。(*)

方針としては F/W 書き換えとしましょう。あまり互換に集中する必要も有りませんし..

おゆう

*平日にはんだごてを握る余裕がありません..

締め切り到来のため、紙面には間に合いませんでした。

もろほし☆らむ

再利用。ネタ切れ・・・鬱死 (배터리 RF をネタにしようとしたら、自分の実力がついてきませんでした (マル RF は奥が深いです。次こそは何かしたいですが、もうちょっと目標レベルを下げることにします (弱気

おまけに、締め切り直前にページ数え間違い発覚 (泣 表紙を入れるのを忘れてました・・・強引に自分のところで 3P も減らしたので、えらく見にくくなってます。申し訳ありません。3 度目の入校だったから、ちょっと気を抜いてしまった結果ですね。今回の 3 号は反省することばかりです (泣

次の機会があれば挽回したいところです。

奥付

発行

どすぶいだにょ☆

発行年月日

2003 年 8 月 17 日 [コミケット 64 配布版]

2007 年 1 月 28 日 [インターネット配布版]

印刷

(株) POPLS [コミケット 64 配布版]

著者

ELM BBS メンバー s

編集・発行責任者

もろほし☆らむ

連絡先

<http://elfin.sakuratan.com/>

kerberos@elfin.sakuratan.com

収録されている記事の無断転載等を禁止します。

