Linuxで電光掲示板

――Linuxハードウェア工作シリーズ

有限会社ハンブルソフト・成松 宏

幅32ドット×縦16ドットのLEDユニットをLinuxを搭載したパソコンに接続し、テキストファイルを表示させる装置を作成しましたので紹介します。

LEDマトリックスはプリンタポート(パラレルポート)で接続しています。「gd」というグラフィックライブラリを用いてテキストファイル中の各文字をドットに分解し、LED上に表示させています(写真1)。LEDユニットは横幅が32ドットしかありませんが、電光掲示板のように文字を流して表示しますので、長い行でも読むことができます。展示会等での案内板に便利でしょう。

事の始め

出張の合間に秋葉原をうろつき、何か面白そうな部品でも買って帰ろうと物色しているときのことでした。秋葉原のT-Zoneパーツショップ(旧 亜土電子パーツショップ)の向かいの左隣りのパーツ屋で、面白そうなものが目に止まりました(写真2、写真3)。

そこには電光掲示板に使用するような16ドット×16ドット

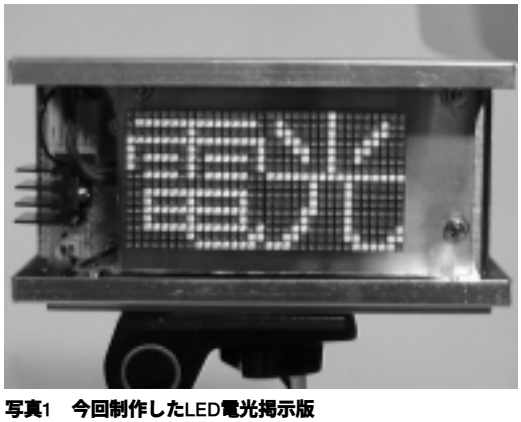


写真1　今回制作したLED電光掲示版

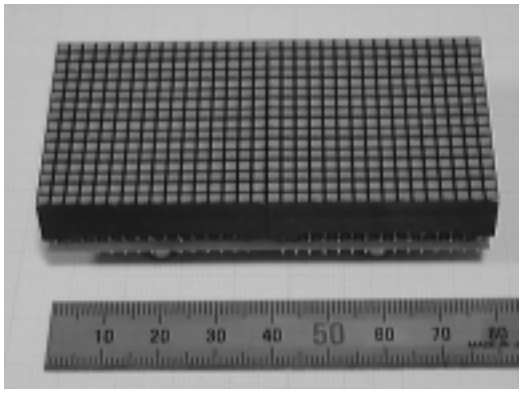


写真2　購入したLEDユニット(LED側)

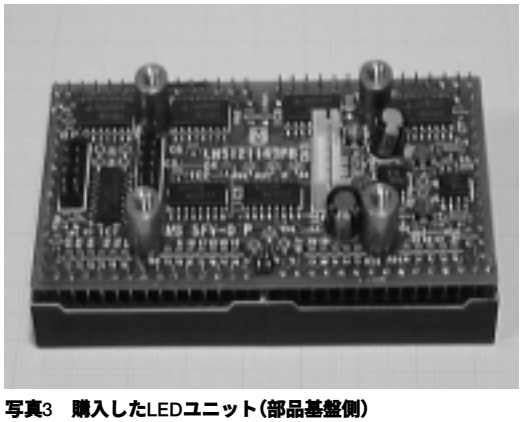


写真3　購入したLEDユニット(部品基盤側)

のLEDユニットが置いてありました。割と大きなサイズで、複数個のユニットを並べて拡張できるような作りになっていました。値段は……、3000円。安い！　裏を見ると部品がいろいろついているので、制御は楽そうな気がしました。値段のところには技術資料付きと書いてあり、その技術資料を見ると、割と簡単に光らせることができそうです。これををLinuxマシンにつないで光らせると面白そうだし、KMLUG(熊本Linuxユーザ会)の定例会で見せびらかせば、ウケそうな気がします。

店頭をよく見ると、もっと小さいLEDユニットや、さらに小さくて32ドット×16ドットで3800円というのもあります。大きいのは置き場に困りそうな気がしましたし、32ドット×16ドットのユニットなら、5インチベイに組み込むこともできそうです。そういうわけで、32ドット×16ドットのLEDユニットを買って帰ったのでした。

LEDユニットの仕組み

図1にコネクタのピン配置、図2に動作波形、図3にブロック図を示します。

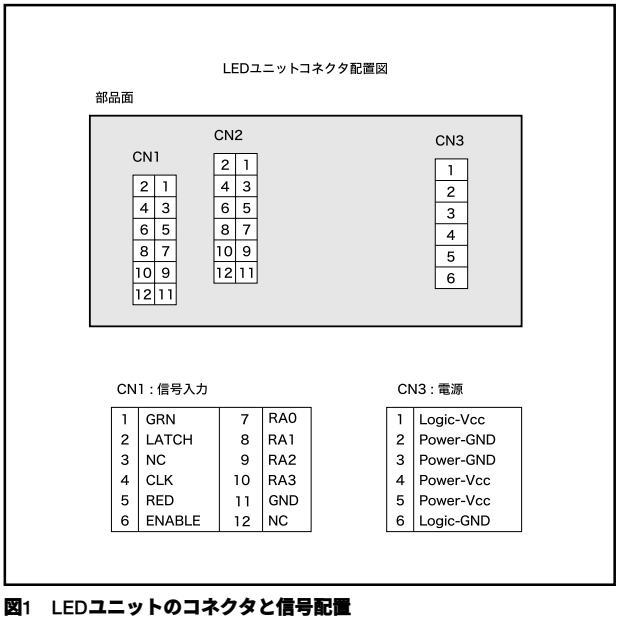
LEDモジュールは、32ビットのシフトレジスタを赤色用と緑色用に2本持ち、このシフトレジスタには、GRN信号とRED信号の値がCLK信号の立ち上がりで読み込まれます。シフトレジスタに読み込まれた値は、LATCH信号の立ち上がりでそれぞれ32ビットレジスタにラッチされ、RA0-3で選択された行の32個のLEDを光らせます。各行のデータを次々にシフトレジスタに送り込んで光らせることで、(縦の)16行全体を光らせることができるわけです。

シフトレジスタの出力はコネクタCN2に出力されています。追加のLEDユニットを用意し、その入力コネクタCN1を元のLEDユニットのCN2と接続すれば、あたかも64ドット×16ドットのLEDユニットであるかのように制御することができます。

LEDユニットには、このCN1とCN2を接続するケーブルと、CN3を接続するケーブルが付属します。おかげで、LEDモジュールに使用されているコネクタに合うプラグを探し回らずに済みました。

パソコンとの接続

LEDモジュールの制御には9つの信号線が必要ですので、プ



1	GRN	7	RA0
2	LATCH	8	RA1
3	NC	9	RA2
4	CLK	10	RA3
5	RED	11	GND
6	ENABLE	12	NC

1	Logic-Vcc
2	Power-GND
3	Power-GND
4	Power-Vcc
5	Power-Vcc
6	Logic-GND

図1　LEDユニットのコネクタと信号配置

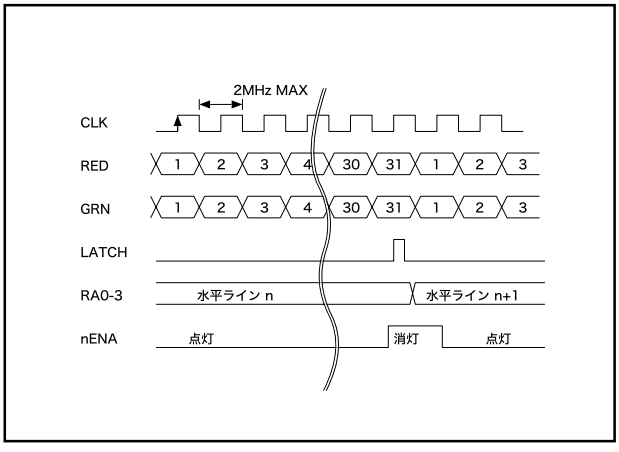


図2　動作波形

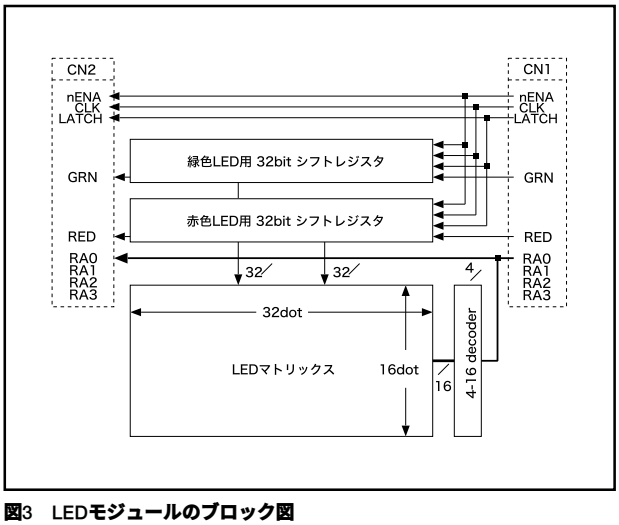


図3　LEDモジュールのブロック図

リンタポートと接続することにしました。回路図を図4に示します。

パソコン内部から5Vの電源を取り出すことも可能ですが、誤ってショートさせるとパソコン本体を壊してしまう可能性もありますし、今回はLEDモジュールをパソコン外部に配置したかったので、安全も考え、ACアダプタを使用しレギュレータで5Vにして使用することにしました。

LEDモジュールをパソコンの5インチベイに取りつける場合などには、内部から電源を取った方が便利だろうと思います。

部品表を表1に示します。値段は参考程度に考えてください。LEDモジュール以外には、入手に苦労する部品はないと思います。秋葉原で簡単に手に入るものばかりですし、地方の方も、秋月電子の通信販売などを利用すれば簡単に手に入れられるでしょう。

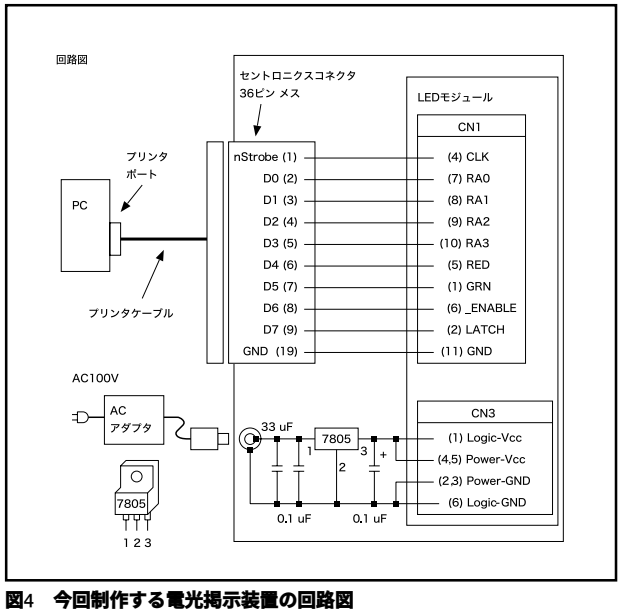


図4　今回制作する電光掲示装置の回路図

表1　今回制作した電光掲示板の部品表

部品名	型名／仕様	数量	単価
LEDユニット	32ドット×16ドット	1	3800円
コネクタ	セントロニクス、36ピン／メス	1	500円
ユニバーサル基板		半分	200円
ACアダプタ	10V 0.5A程度	1	300円
ソケット	ACアダプタ用	1	50円
電源レギュレータ	7805	1	70円
小型放熱板		1	60円
コンデンサ	0.1uF 積層セラミック	2	20円
コンデンサ	33uF 25V電解	2	30円
シャーシ	145mm×70mm×50mm	1	240円
スペーサ	5mm	2	50円
スペーサ	25mm	4	60円
アルミ板	1mm厚、100mm×50mm		
アルミ板	3mm厚、40mm×130mm		

私は、普通のプリンタ用ケーブルを使用したかったので、コネクタに、36ピン／メスのセントロニクスコネクタを使用しました。しかし、このコネクタが手に入りにくい場合には、DSUB 25ピンコネクタを使用するのもいいかもしれません。その場合には、コネクタのピン番号が変わってきますので注意してください。

消費電力は実測で0.1A程度でした。

ACアダプタが10Vですと、電源レギュレータ「7805」の消費電力は、

(10V－5V)×0.1A＝500mW

となります。放熱板をつけない7805の熱抵抗は、約75度C/Wだそうなので、30度近くも熱が出ることになります。小さい放熱器でも熱抵抗は19度C/Wぐらいなので、放熱器をつけてやれば随分温度が下がります。できれば、つけてやった方が安心でしょう(触ったときレギュレーターが熱いと結構びっくりします)。

ケースに収める

配線しただけでケースに収めないと、結線が切れやすいですし、見栄え、収納性も悪いので、ここはやはり、ケースに収めることにしました。

ちょうど良いサイズのケースを探したところ、145mm×70mm×50mmのシャーシが240円で売っていました。シャーシですのでフタがありませんが、今回はかえって便利です。そのうち透明なアクリル板か何かで蓋をしたいと思っています。

このケースをカメラの三脚に取り付けることができれば便利だろうと考えて、取り付け金具を作ることにしました。

三脚の取り付けネジのサイズは1/4インチです。1/4インチのタップでネジを切れば取り付けることができます。1/4インチタップは、ホームセンターで、680円で買うことができました。シャーシのアルミ板は1mmほどで、これにネジを切ってもすぐにつぶれてしまうので、手持ちの厚さ3mm、幅40mmのアルミ板にネジを切ってシャーシに取り付けました(写真4)。

シャーシの加工は、ハンドドリル、やすり、ハンドリーマーがあればできますが、最近は電動工具が安いので、これらを使用するともっと簡単に加工ができます。私は、小型ボール板で穴をあけ、電動ジグソーでアルミ板を切断しました(写真5、写真6)。

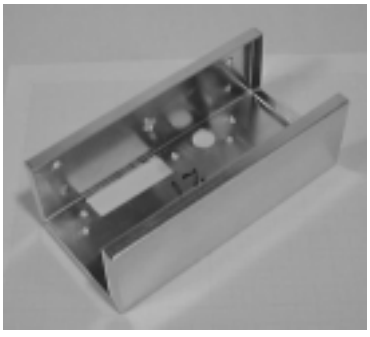


写真4　工作して穴をあけたアルミシャーシ

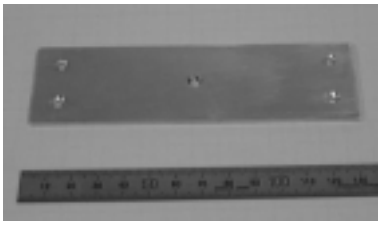


写真5　制作した三脚固定用金具

写真6
三脚に取り付けた
LED電光掲示板

プログラムの構成

このLED電光掲示板を駆動するプログラムについて説明します。今回作成したプログラムは、テキストファイルを読み込み、1行づつLEDユニットに流して表示するという動作をします。テキストファイルは緑色で表示し、行の終りを赤色のマークで表示しています。電光掲示板のプログラムなので、もっと凝った表現がいろいろ考えられますが、今回はとりあえず、簡単で実用性もあるということで、このような仕様にしました。

プログラムの構造図を図5に示します。

プログラムを実現するには、文字をドットのデータに変換してやる必要があります。gdライブラリと、それをPerlから利用する「GDモジュール」を使用し、Perlのプログラムでこれを行ないます。図5の「news.pl」がこれに当たります。

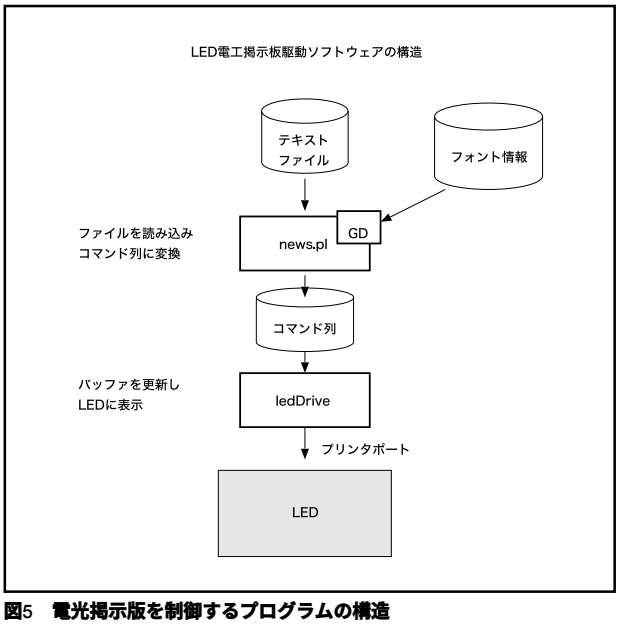


図5　電光掲示版を制御するプログラムの構造

プリンタポートを経由してLEDモジュールをコントロールするプログラムは、C言語で作成しました(図5の「ledDrive」)。

news.plとledDriveとのインターフェイスは、表2に示すコマンド列で行ないます。news.plがテキストファイルを読み込み、それを表2のコマンド列に変換し、ledDriveがそのコマンド列を読み込んでledモジュールを光らせる、という仕組みです。

gd

プログラム本体の説明の前に、gdについて簡単に説明します。

gdとは、線や円弧文字などで書かれたイメージデータを作成し、PNG形式で出力するグラフィックライブラリです(記事末RESOURCE[1]を参照)。PNG形式は、多くのWebブラウザでサポートされるフォーマットですので、Web上で動く、グラフィックを出力するようなアプリケーションの開発にはとても便利です。

gd添付のドキュメント(readme.txt)によりますと、Ver.1.7で、Masahito Yamaga氏によって、TrueType機能に日本語サポートが追加されました。今回のプログラムは、この機能を使用しましたので、Ver.1.7以降のgdが必要です。私は、gd-1.7.3を使用しました(付録CD-ROMにVer.1.8.1を収録しました)。

また、GD.pm([2])を使用しますと、gdの機能をPerlから使用できます。今回は、gd-1.7.3に対応したGD-1.23を使用しま

表2　ledDriveが受け付けるコマンド列

コマンド	説明
C	バッファのクリア
L [n]	バッファの内容でLEDをn回点灯させる (指定しなければ1回)
X r ddd……d	バッファのr行目に、ddd……dのデータを書き込む (dは、1ピクセル1文字、R、G、O、.のいずれかの文字)
S ddd……d	バッファを1ピクセル左にシフトし、 最も右の列にデータを書き込む
空白行	無視される
#で始まる行	コメント

した(付録CD-ROMには、gd-1.8.1に対応したGD-1.27を収録しました)。

GD.pmの使い方を示すために、GD.pmを使用した簡単なプログラム「helloGD.pl」を作ってみました(リスト1)。実行結果は実行例1のようになります。

このhellpGD.plは、「Hello!」という文字を大きく出力するプログラムです。

以下、helloGD.plの説明を行ないます。

・1行目

このプログラムがPerlのスクリプトであることの宣言です。perlコマンドのパスが/usr/bin以外の場合は、それに合わせて「#!/usr/local/bin/perl」などと直してください。

・2行目

GDモジュール(GD.pm)を使用すると宣言しています。

・3行目

GDモジュールからLargeフォントを取り出し、変数「$font」にセットしています。

・4行目〜6行目

文字列をセットし、文字列を$fontで描画したときの幅と高さを、$wと$hにセットしています。

・7行目

文字列を描画するためのイメージ領域を生成し、$imにセットしています。

リスト1　GDモジュールのサンプルプログラム「helloGD.pl」

```
 1  #!/usr/bin/perl
 2  use GD;
 3  $font=GD::Font->Large;
 4  $str="Hello!";
 5  $h=$font->height;
 6  $w=$font->width * length($str);
 7  $im = new GD::Image($w,$h);
 8  $black = $im->colorAllocate(0,0,0);
 9  $green = $im->colorAllocate(0,255,0);
10  $im->filledRectangle(0,0,$w-1,$h-1,$black);
11  $im->string($font, 0, 0, $str,$green);
12  for $y (0 ... $h-1){
13      for $x (0 ...$w-1){
14          if($im->getPixel($x,$y)==$black){
15              print " ";
16          } else {
17              print "*";
18          }
19      }
20      print "\n";
21  }
```

・8行目、9行目

黒色と緑色を確保し、変数$blackと$greenにセットしています。

・10行目

イメージの領域全体を黒色で塗り潰し、クリアしています。

・11行目

変数$strにセットされた「Hello!」という文字列を緑色でイメージ領域に描画しています。

・12行目〜21行目

イメージ領域を1ピクセルごとに読み出し、黒色であればスペース(" ")、そうでなければ(緑色であれば)星印("*")を出力します。

このように、GD.pmを使うと、フォントの情報やグラフィック(線や円弧)を用いたプログラムを簡単に作ることができます。

news.pl

news.plプログラムは、第1引数、あるいは標準入力から入力されたテキストファイルを、1行ずつLEDモジュールに表示するプログラムです。例えば、

```
$ perl news.pl example.txt
```

とすれば、example.txtの内容が1行ずつ表示されます。すべての行が表示されたら、また最初から繰り返して表示します。終了するには、Ctrl-Cなどを押してください。suidされたプログラム「ledDrive」を動かしているせいで、1度Ctrl-Cを押しただけでは止まらないこともあります(というか止まらないことがほとんどです)ので、何度か連続してCtrl-Cを押してください。

実行例1　helloGD.plの実行例

```
$ perl -w helloGD.pl

*    *          **      **             *
*    *           *       *             *
*    *           *       *             *
*    *   ****    *       *      ****   *
******  *    *   *       *     *    *  *
*    *  *    *   *       *     *    *  *
*    *  ******   *       *     *    *  *
*    *  *        *       *     *    *
*    *  *        *       *     *    *  *
*    *   ****   *****   *****    ****   *
```

日本語も表示できます。EUCで書いてください。

news.plは、読み込んだテキストファイルを行ごとに分解し、ledDriveプログラムの入力となるコマンド列を作成した後、ledDriveプログラムを呼び出して表示を行ないます。表示は、ledDriveのSコマンド(表1)を使用し、1ピクセルごと横にシフトしながら表示するので、長い行でも表示できます。

news.plのリストをリスト2に示します。

では、このプログラムを説明していきましょう。

・2行目

ledDriveをフルパスで指定しています。

・4行目

文字の描画用に800×16ドットの領域を確保しています。

・5行目〜8行目

4種類の色を定義しています。

・9行目

変数$marginに値" 32 "を設定しています。$marginの値は、各行の右に取る空白の領域の幅を示します。この値を32以上とれば、各行がすべて左にシフトされ消えた後に、次の行が表示されることになります。

・10行目

変数$tmpFileは、ledDrive用のコマンドファイルを書き込むテンポラリファイルの名前です。

・11行目、12行目

TrueTypeフォントのフォント名を設定しています。日本語を使用できるように、日本語のフォントを指定します。残念なことに、使用できるのは、「.ttf」ファイルだけで、「.ttc」のTrueTypeフォントは使用できないようです。

・14行目

引数に指定されたファイルから、あるいは、指定されたファイルがないときには標準入力から、入力のすべての行を読み込み、配列linesに代入しています。

・15行目

無限ループで、以下のブロックの処理を永遠に続けます。

・16行目

配列linesの要素を1つずつ変数$_に代入しながら、以下のブロックの処理を行ないます。

・18行目

変数$strの最後の文字、つまり改行コードを削除します。

リスト2 news.pl

```
 1  #!/usr/bin/perl
 2  $ledDrive=/usr/local/bin/ledDrive
 3  use GD 1.20;
 4  $im = new GD::Image(800,16);
 5  $black = $im->colorAllocate(0,0,0);
 6  $red    = $im->colorAllocate(255,0,0);
 7  $green  = $im->colorAllocate(0,255,0);
 8  $orange = $im->colorAllocate(255,255,0);
 9  $margin = 32;
10  $tmpFile = ",tmp.led";
11  $font ="/usr/share/fonts/" .
12         "TrueType/watanabe-mincho.ttf";
13  -f $font || die "$font not found.\n";
14  @lines = <>;
15  while(1){
16      foreach (@lines) {
17          $str = $_;
18          chop($str);
19          $fs = 14;
20          @b = $im->stringTTF($green,$font,$fs,
21                              0.0,0, 14, $str);
22          $x1 = $b[2];
23          @b = $im->stringTTF($red,$font, $fs,
24                  0.0,$x1+16, 14, "\$");
25          $x1 = $b[2] + $margin;
26
27          open(OFP,">$tmpFile") ||
28                  die "open $tmpFile failed.\n";
29          print OFP "C\n";
30          print OFP "L 1\n";
31          for($x=0;$x<$x1;$x++){
32              print OFP &ledShift($im,$x),"\n";
33              print OFP "L 1\n";
34          }
35          close(OFP);
36          $im->filledRectangle(0,0,$x1,16,
37                               $black);
38          system("$ledDrive $tmpFile");
39      }
40  }
41  exit(0);
42
43  sub ledShift {
44      local($im, $x) = @_;
45      local($y,$pixel,$ret,$r,$g,$b);
46      $ret = "S ";
47      for($y=0;$y<16;$y++){
48          $pixel =  $im->getPixel($x,$y);
49          ($r,$g,$b) = $im->rgb($pixel);
50          if($r > 0 && $g > 0){ $ret .= "O"; }
51          elsif($r > 0)       { $ret .= "R"; }
52          elsif($g > 0)       { $ret .= "G"; }
53          else                { $ret .= "."; }
54      }
55      $ret;
56  }
```

・19行目
変巣$fsをセットしています。$fsはフォントのサイズを示します。

・20行目
TrueTypeフォントを使用して、変数$strの内容、すなわちファイルの行の内容をイメージ$imに緑色で描画します。配列@bには、8個の数値が代入されます。それらの数値は、以下のように、文字列が描画された領域の4つの座標を示します。

```
@b[0,1]    左下の(x,y)座標
@b[2,3]    右下の(x,y)座標
@b[4,5]    右上の(x,y)座標
@b[6,7]    左上の(x,y)座標
```

・22行目
描画された文字列の右端の座標を変数$x1に代入しています。

・23行目、24行目
文字列の右端の座標から16ドット右に、行末を示す$マークを赤色で描画しています。

・25行目
行末の$マークの右端の座標$marginの値を変数$x1にセットしています。この$x1の値までシフトするコマンドを生成させます。

・27行目
ledDriveプログラムの入力となるコマンドファイルを書き込むために、オープンしています。

・29行目
行の最初にクリアコマンド「C」を書き込んでいます。

・30行目
次に、1回、LEDに表示コマンド「L 1」を書き込んでいます。

・31行目〜34行目
x座標で0から$x1-1まで、イメージを読み出し、ledShift関数で生成したシフトコマンドと、表示コマンド「L 1」を書き込んでいます。

・36行目
領域をクリアしています(描画前にクリアした方がいいかも)。

・38行目
system関数で、ledDriveコマンドを呼び出し、LEDマトリックスを点灯させています。

続いて、43行目以降の関数「ledShift」の説明です。この関数は、引数でイメージ$imとX座標$xを渡し、$imのx=$xカラムのピクセル値を読み出し、シフトコマンドを生成しています。生成されるコマンドは、例えば、次のような文字列になります。

```
S ....GGGG..RR.GGG
```

48行目で、ピクセルの色を呼び出した後、その色のRGB成分を49行目で求めています。

本来であれば、このようなことをしなくてもピクセル値の比較だけでいけると思うのですが、stringTTFで描画すると、なぜか色が混じってしまうため(アンチエイリアス処理?)、ピクセル値の比較ではうまくいかず、このようにしました。

ledDrive.c

最後に、コマンドファイルを読み込み、LEDユニットをプリンタポート経由で駆動するプログラム「ledDrive」の説明をします。ここでは、プログラムの中心となるLEDをドライブする部分のみを説明します。news.plを含めたプログラム全体は、筆者のホームページで公開しています([3])。

リスト3に、ヘッダファイル「ledDrive.h」の中身を示します。ヘッダファイルでは、共通のインクルードファイルの読み込み、定数の定義、構造体の定義と関数のプロトタイプ宣言を行なっていますが、プロトタイプ宣言はスペースの関係で省略してあります。

・1行目〜6行目
インクルードファイルをまとめて読み込んでいます。

・8行目〜10行目
プリンタポートのアドレスを宣言しています。プリンタポートのアドレスを自動で取得する方法が分からなかったので、ここで宣言しています。各自の環境に合わせてアドレスを設定してください。

・11行目
ioperm関数で確保するI/Oのアドレスの幅です。

・13行目〜19行目
LEDモジュールの信号線とプリンタポートの接続をここで宣言しています。

・21行目〜25行目
プログラム全体の動作をコントロールする構造体「LedCont」

を定義しています。

・27行目～33行目

Ledモジュールに表示するイメージを保持する構造体「LedBuf」を定義しています。

リスト4にledDrive関数のリストを示します。ledDrive関数は、LedCont構造体とLedBuf構造体を受け取り、LedBufの内容をLedモジュールに表示します。

・51行目

リスト3に示したledDrive.hファイルを読み込んでいます。

・52行目

outbマクロを使用するために/usr/include/asm/io.hファイルを読み込んでいます。

・62行目

ioperm関数で、プリンタポートのアクセス許可を取っています。IOポートはrootでないと使用できませんので、このプログラムはrootで実行するか、rootにsuidしてやる必要があります。

・66行目

プリンタポートのデータポートの出力を一度クリアしています(必要ないかもしれません)。

リスト3　ledDrive.h(一部抜粋)

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3  #include <string.h>
 4  #include <stdarg.h>
 5  #include <ctype.h>
 6  #include <unistd.h>
 7
 8  #ifndef BASEPORT
 9  #  define  BASEPORT 0x378
10  #endif
11  #define  PORTSIZE 8
12
13  #define LED_RA_SHIFT      0
14  #define LED_RA_MASK       0x0f
15  #define LED_RED_MASK      0x10
16  #define LED_GREEN_MASK    0x20
17  #define LED_NENABLE_MASK  0x40   /* 負論理*/
18  #define LED_LATCH_MASK    0x80
19  #define LED_STB_MASK      0x01
20
21  typedef struct _ledCont {
22    int portAdr;
23    int lineDelay;
24    struct _ledBuf *buf;
25  } LedCont;
26
27  typedef struct _ledBuf {
28    int width;
29    int height;
30    int redMask;
31    int greenMask;
32    unsigned char *data;
33  } LedBuf;
```

リスト4　ledDrive.c(LED駆動部を抜粋)

```
51  #include "ledDrive.h"
52  #include <asm/io.h>
53
54  void ledDrive(LedCont *cont,LedBuf *buf)
55  {
56    int x,y,d;
57    int dv;      /* データポートに出力する値 */
58    int cv;/* コントロールポートに出力する値 */
59
60    dv = 0;
61    cv = 0;
62    if(ioperm(cont->portAdr, PORTSIZE, 1)){
63      perror("ioperm");
64      exit(1);
65    }
66    outb(dv, cont->portAdr);
67    for(y=0;y<buf->height;y++){
68      for(x=0;x<buf->width;x++){
69        d = buf->data[x+y*buf->width];
70        if(d & buf->redMask)dv |=  LED_RED_MASK;
71        else                dv &= ~LED_RED_MASK;
72        if(d & buf->greenMask)
73              dv |= LED_GREEN_MASK;
74        else  dv &= ~LED_GREEN_MASK;
75        outb(dv, cont->portAdr);
76        cv &= ~LED_STB_MASK;
77        outb(cv, cont->portAdr+2);
78        cv |= LED_STB_MASK;
79        outb(cv, cont->portAdr+2);
80      }
81      outb(dv, cont->portAdr);
82      dv |= LED_LATCH_MASK;
83      outb(dv, cont->portAdr);
84      dv &= ~LED_LATCH_MASK;
85      outb(dv, cont->portAdr);
86      dv &= ~(LED_RA_MASK << LED_RA_SHIFT);
87      dv |= (y & LED_RA_MASK) << LED_RA_SHIFT;
88      dv &= ~LED_NENABLE_MASK;
89      outb(dv, cont->portAdr);
90      for(x=0;x<cont->lineDelay;x++)
91        outb(dv,cont->portAdr); /* ディレイ */
92      dv |= LED_NENABLE_MASK;
93      outb(dv, cont->portAdr);
94    }
95    if(ioperm(cont->portAdr, PORTSIZE, 0)){
96      perror("ioperm");
97      exit(1);
98    }
99  }
```

・69行目
座標(x, y)のピクセルの値を読み出しています。1ピクセル当たり1byte使用しています。必要なのは2bitなので、4倍ほど贅沢しております。その代わりにプログラムは楽ができています。

・70行目〜75行目
ピクセル値の赤色と緑色の状態を調べて、対応する値を作りデータポートに出力しています。

・76行目〜79行目
コントロールポートのSTB信号(LEDモジュールのCLKに接続されている)に値"0"のパルスを出力しています。STB信号の立ち上がりで赤色および緑色LEDのデータがLEDモジュールのシフトレジスタに読み込まれます。

・82行目〜85行目
データポートに接続されているLATCH信号に値"1"のパルスを出力しています。このパルスによって、LEDモジュールのシフトレジスタに書き込まれた値が、ドライブ用のレジスタに出力されます。

・86行目〜89行目
y座標の値をLEDモジュールのRA0-3に出力しています。また、ENABLE信号の値を"0"にしています。このことにより、y座標で指定された行のLEDだけが光ります。

・90行目〜91行目
LEDを光らせた状態でしばらく待つために、無駄なoutbを繰り返し実行しています。ここでは2ミリ秒程度待ちたいのですが、usleep()等では、そのように短いディレイを作り出すことができず、仕方なくこのような方法でディレイを作っております。

・92行目〜93行目
ENABLE信号を"1"にし、LEDを消灯します。

・95行目
以上の操作を行数分(通常16回)繰り返した後、IOポートの使用許可を返上しております。

●時分割の問題

RT-LinuxやART-Linuxなら取れるのでしょうが、通常のLinuxでは、短い正確なディレイを取れません。

1秒間に16ラインを30回ドライブするとすると、1ライン当たりの周期は

1秒÷30÷16　2ミリ秒

となります。

ラインのディレイが長くなると、そのラインだけ明るく表示されて、表示がちらついて見えます。usleepで1マイクロ秒単位でディレイを指定できますが、コンテキストスイッチが伴い、実際には10ミリ秒単位のディレイになってしまいます。

ledDriveでは、こうした問題を回避するために、busy-loopでディレイを作り、CPUを無駄に消費しています。並列に動作するプログラムがあると、表示が乱れてしまいます。

このため、実は、

```
generator.pl  | ledDrive
```

というような方法は 使えません。news.plでは、コマンド列を一旦ファイルに生成し、system関数でledDriveを呼び出し、ledDrive実行中にnews.plが動かないようにしています。

今後の課題

部品のLEDマトリックスを使うと、部品の入手がもっと簡単で安価になり、大きいものも作れるようになるのではないかと考えています。但し、ドライブ回路も作らなければなりません。

RT-LinuxやART-Linuxを使うと、マルチタスクでもちゃんと動いて気持がよさそうです。それらを使ってみる例題としても面白そうです。しかし、LED表示装置のためだけにRT-Linuxなどを組み込むというのも大げさなので、PICマイコンや秋月電子のH8ボードで制御するというのが、実用品としては適しているように思います。また、貴重なプリンタポートを使用してしまうのが心苦しいし、別途、電源も必要なのも面倒なので、なんとかUSB化も計りたいと考えています。

R E S O U R C E

[1] グラフィックスライブラリ「gd」
http://www.boutell.com/gd/

[2] Perlからgdを利用するためのモジュール「GD」
http://stein.cshl.org/WWW.software/GD

[3] 筆者のWebサイト
今回作成したプログラムを公開。
http://www.humblesoft.com/nari/