Wyświetlacze graficzne LCD ze sterownikiem KS0108 – sterowanie w języku C od podstaw, część 2

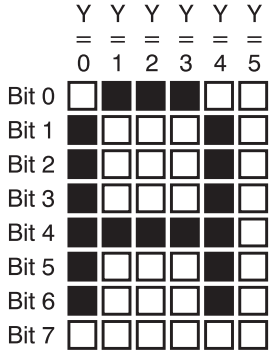
W najróżniejszych urządzeniach zbudowanych w oparciu o mikrokontrolery, do prezentacji danych wyjściowych wykorzystywane są wyświetlacze LCD. Najczęściej są to wyświetlacze alfanumeryczne ze sterownikiem HD44780 ze względu na stosunkowo niską cenę oraz łatwe sterowanie. Możliwości prezentacji danych na wyświetlaczach alfanumerycznych są niewielkie w porównaniu z wyświetlaczami graficznymi. W artykule zajmiemy się obsługą w języku C wyświetlacza graficznego ze sterownikiem KS0108.

W pierwszej części artykułu zapoznaliśmy się z ogólnymi zasadami sterowania wyświetlaczem graficznym i sposobami dołączenia go do mikrokontrolera. Poniżej pokażemy, w jaki sposób na ekranie wyświetlacza graficznego można wyświetlać teksty, a także – co nas może nawet bardziej interesuje – grafikę.

Tryb tekstowy

Wyświetlacze zbudowane w oparciu o kontroler KS0108 nie posiadają generatora znaków, wyświetlanie tekstu wymaga więc zdefiniowania własnej tablicy czcionek, która będzie przechowywana w pamięci programu mikrokontrolera. Tablica ta znajduje się w pliku font.h.

Każdy znak jest określony pięcioma bajtami danych (typowa czcionka 5x8 pikseli). Kolejność bajtów w tablicy odpowiada kolejności poszczególnych pionowych fragmentów znaku. Przykładowy wygląd litery A przedstawiono na **rys. 4**. Do projektowania własnych czcionek można wykorzystać program ze strony http://radzio.dxp.pl/font/. Program ten umożliwia eksport danych bezpośrednio do plików źródłowych języka C oraz asemblera.



Rys. 4. Przykładowy kształt litery A

Wyświetlenie znaku

Wyświetlenie znaku polega na zapisaniu do wyświetlacza 5 kolejnych bajtów tworzących dany znak. Pierwszy znak w tablicy (spacja) ma indeks 0, podczas gdy w tablicy ASCII ten znak posiada kod 32. Przyjęto tak, żeby nie marnować miejsca na niewyświetlane znaki w pamięci programu (pierwsze 32 znaki tablicy ASCII). Należy więc od kodu znaku przekazanego do funkcji jako parametr odjąć liczbę 32.

```
void lcdWriteChar(char x)
{
char i;
x-= 32; // konwersja kodu znaku
for(i = 0; i < 5; i++)
  lcdWriteData(pgm_read_byte(font5x7
+ (5 * x) + i)); // zapis do wyświe-
tlacza
// 5 kolejnych bajtów tworzących
znak
lcdWriteData(0); // odstęp między
znakami
}
```

Wyświetlanie ciągu znaków

Parametrem funkcji jest wskaźnik do typowego dla języka C ciągu znaków zakończonego zerem (znajdującego się w pamięci danych).

```
void lcdWriteStrin-
g(char * s)
{
while(*s) // wy-
konuj dopóki znak
wskazywany przez s
jest różny od zera
  lcdWriteCha-
r(*s++); // zapis
znaku
}
```

Wyświetlanie ciągu znaków z pamięci programu

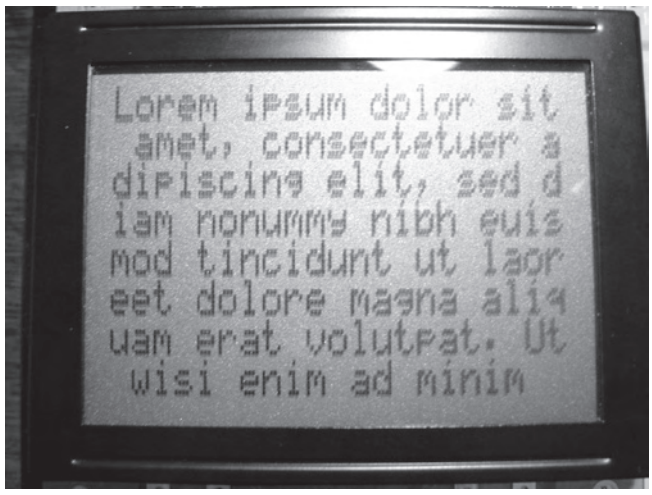
Ponieważ przechowywanie w pamięci RAM stałych napisów powoduje znaczne zużycie stosunkowo niewielkiej pamięci, wygodniej jest niezmienne ciągi znaków przechowywać w pamięci programu. Ze względu na specyficzny odczyt pamięci programu, w kompilatorze avr–gcc do tego celu służy oddzielna funkcja.

```
void lcdWriteStringPgm(prog_char *
s)
{
char c; // pomocnicza zmienna
while(c = pgm_read_byte(s++)) //
wykonuj dopóki znak wskazywany przez
s jest
// rożny od zera
  lcdWriteChar(c);  // zapis znaku
}
```

Ustawienie współrzędnych tekstowych

W trybie tekstowym wygodniejsze jest posługiwanie się współrzędnymi odpowiadającymi położeniu znaków na wyświetlaczu. Jak wiemy każdy znak składa się z sześciu pionowych części, zamiana współrzędnych tekstowych na graficzne sprowadza się więc do pomnożenia przez 6 poziomej współrzędnej tekstowej.

```
void lcdLocate(unsigned char x,
unsigned char y)
{
lcdGoTo(x * 6, y);
}
```



Fot. 5. Przykładowy tekst umieszczony na wyświetlaczu graficznym

Przykładowy tekst umieszczony na wyświetlaczu za pomocą zaprezentowanych funkcji przedstawia **fot. 5**.

Tryb graficzny

Głównym zadaniem wyświetlacza graficznego jest (jak sama nazwa wskazuje) wyświetlanie grafiki. Przedstawię teraz kilka podstawowych procedur graficznych.

Zapalanie piksela

Włączenie piksela polega na ustawieniu odpowiadającego mu bitu w pamięci wyświetlacza. Niestety możemy odczytywać i zapisywać tylko cały bajt, dlatego procedura włączenia piksela przebiega następująco:

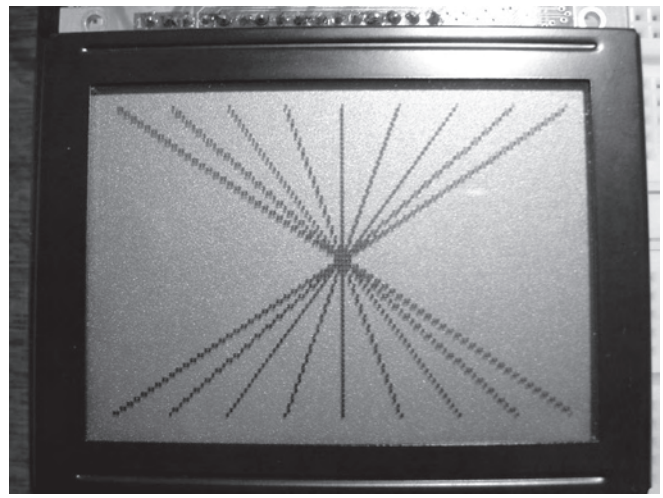
- ustawiamy współrzędne: poziomą oraz podzieloną przez 8 pionową (ponieważ dostęp do pamięci odbywa się całymi bajtami),
- odczytujemy z wyświetlacza aktualny stan pikseli (operację odczytu należy wykonać dwukrotnie),
- modyfikujemy odczytany bajt poprzez wykonanie na nim operacji sumy logicznej z bitem o wartości 1 przesuniętym w lewo o resztę z dzielenia współrzędnej pionowej przez 8 (położenie bitu w obrębie wybranej strony),
- zapisujemy tak zmodyfikowany bajt danych pod odpowiedni adres pamięci wyświetlacza.

```
void lcdSetPixel(unsigned char x,
unsigned char y)
{
unsigned char temp;  // zmienna po-
mocnicza
lcdGoTo(x, y/8); // ustawienie
współrzędnych
temp = lcdReadData(); //  odczyt da-
nych z wyświetlacza
lcdGoTo(x, y/8); // ustawienie
współrzędnych
temp = lcdReadData(); // odczyt ak-
tualnego stanu pikseli
lcdGoTo(x, y/8); // ponowne ustawie-
nie współrzędnych
lcdWriteData(temp | (1 << (y % 8)));
// zapis odpowiedno zmodyfikowanej
wartości
}
```

Gaszenie piksela

Procedura gaszenia piksela przebiega podobnie jak jego zapalanie. Jedyną różnicą jest sposób modyfikacji odczytanego aktualnego stanu pikseli: zamiast sumy logicznej wykonywany jest iloczyn logiczny z zanegowaną wartością przesunięcia w lewo bitu o wartości 1 o resztę z dzielenia przez 8 współrzędnej pionowej.

```
void lcdClrPixe-
l(unsigned char x,
unsigned char y)
{
unsigned char temp;
// zmienna pomoc-
nicza
lcdGoTo(x, y/8); //
ustawienie współ-
rzędnych
temp = lcdReadDa-
ta(); //
lcdGoTo(x, y/8); //
ustawienie współ-
rzędnych
temp = lcdReadDa-
ta(); // odczyt
aktualnego stanu
pikseli
lcdGoTo(x, y/8); //
ponowne ustawienie
współrzędnych
lcdWriteData(temp &
~(1 << (y % 8)));
// zapis odpowied-
nio zmodyfikowanej
wartości
}
```



Fot. 6. Efekt działania funkcji lcdLine

Rysowanie linii

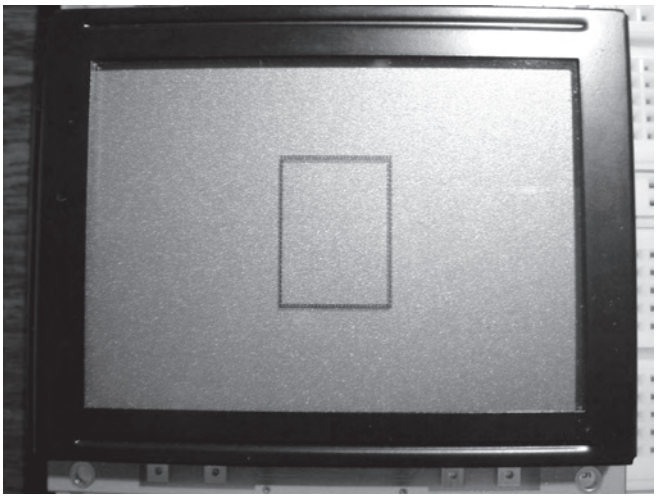
Funkcja przyjmuje 4 argumenty: współrzędne x i y początku oraz współrzędne x i y końca linii. Funkcja rysowania linii bazuje na algorytmie Brensenham'a. Szczegółowy opis algorytmu Brensenham'a do rysowania linii można znaleźć w [1].

```
void lcdLine(int X1, int Y1,int
X2,int Y2)
{
int CurrentX, CurrentY, Xinc, Yinc,
    Dx, Dy, TwoDx, TwoDy,
  TwoDxAccumulatedError, TwoDyAccu-
mulatedError;

Dx = (X2-X1); // obliczenie składo-
wej poziomej
Dy = (Y2-Y1); // obliczenie składo-
wej pionowej

TwoDx = Dx + Dx; // podwojona skła-
dowa pozioma
TwoDy = Dy + Dy; // podwojona skła-
dowa pionowa

CurrentX = X1; // zaczynamy od X1
CurrentY = Y1; // oraz Y1

Xinc = 1; // ustalamy krok zwiększa-
nia pozycji w poziomie
Yinc = 1; // ustalamy krok zwiększa-
nia pozycji w pionie

if(Dx < 0) // jesli składowa pozioma
jest ujemna
  {
  Xinc = -1; // to będziemy się "co-
fać" (krok ujemny)
  Dx = -Dx;  // zmieniamy znak skła-
dowej na dodatni
  TwoDx = -TwoDx; // jak również po-
dwojonej składowej
  }

if (Dy < 0) // jeśli składowa piono-
wa jest ujemna
  {
  Yinc = -1; // to będziemy się "co-
fać" (krok ujemny)
  Dy = -Dy; // zmieniamy znak skła-
dowej na dodatki
  TwoDy = -TwoDy; // jak równiez po-
dwojonej składowej
  }

lcdSetPixel(X1,Y1); // stawiamy
pierwszy krok (zapalamy pierwszy
piksel)

if ((Dx != 0) || (Dy != 0)) //
sprawdzamy czy linia składa się
z więcej niż jednego
 // punktu ;)
  {
  // sprawdzamy czy składowa pionowa
jest mniejsza lub równa składowej
poziomej
  if (Dy <= Dx) // jeśli tak, to
idziemy "po iksach"
    {
    TwoDxAccumulatedError = 0; //
zerujemy zmienną
    do // ruszamy w drogę
    {
      CurrentX += Xinc; // do aktu-
alnej pozycji dodajemy krok
      TwoDxAccumulatedError += Two-
Dy; // a tu dodajemy podwojoną skła-
dową   //pionową
      if(TwoDxAccumulatedError > Dx)
// jeśli TwoDxAccumulatedError jest
większy  //od Dx
        {
        CurrentY += Yinc; // zwięk-
szamy aktualną pozycję w pionie
        TwoDxAccumulatedError -=
TwoDx; // i odejmujemy TwoDx
        }
       lcdSetPixel(CurrentX,Curren-
tY);// stawiamy następny krok (zapa-
lamy piksel)
      }while (CurrentX != X2); //
idziemy tak długo, aż osiągniemy
punkt docelowy
    }
   else // w przeciwnym razie idzie-
my "po igrekach"
      {
      TwoDyAccumulatedError = 0;
      do
      {
        CurrentY += Yinc;
        TwoDyAccumulatedError +=
TwoDx;
        if(TwoDyAccumulatedError>Dy)
          {
          CurrentX += Xinc;
          TwoDyAccumulatedError -=
TwoDy;
          }
         lcdSetPixel(CurrentX,Cur-
rentY);
        }while (CurrentY != Y2);
    }
  }
}
```

Efekt działania funkcji lcdLine przedstawiono na **fot. 6**.

Rysowanie prostokąta

Funkcja przyjmuje 4 argumenty: współrzędne x oraz y lewego górnego rogu prostokąta oraz wysokość i szerokość prostokąta. Ponieważ prostokąt może się składać tylko z linii pionowych i poziomych do jego narysowania nie wykorzy-



Fot. 7. Efekt działania funkcji lcdRectangle

stamy funkcji rysowania linii przedstawionej powyżej, tylko zrobimy to w znacznie prostszy sposób.

```
void lcdRectangle(unsigned char x,
unsigned char y, unsigned char b,
unsigned char a)
{
  unsigned char j; // zmienna pomoc-
nicza
  // rysowanie linii pionowych
(boki)
  for (j = 0; j < a; j++) {
     lcdSetPixel(x, y + j); // linia
lewa
     lcdSetPixel(x + b - 1, y + j);
// linia prawa
   }
  // rysowanie linii poziomych (pod-
stawy)
  for (j = 0; j < b; j++) {
     lcdSetPixel(x + j, y); // linia
górna
     lcdSetPixel(x + j, y + a - 1);
// linia dolna
   }
}
```

Efekt działania funkcji lcdRectangle przedstawiono na **fot. 7**. Obraz jest rozciągnięty w pionie, ponieważ wyświetlacz JM12864A posiada prostokątne piksele.

Rysowanie wypełnionego prostokąta

Funkcja przyjmuje 4 parametry podobnie jak funkcja poprzednia. Rysowany jest wypełniony prostokąt. W celu przyśpieszenia funkcji prostokąt nie jest rysowany piksel po pikselu. Całkowicie zajęte strony są wypełniane bajtami o wartości 255, natomiast pozostałe linie, które nie zajmują pełnej strony są dorysowywane w tradycyjny sposób.

```
void lcdFillRectangle(unsigned char
x, unsigned char y, unsigned char b,
unsigned char a)
{
unsigned char i, j;
// zmienna przechowująca ilość stron
zajętych przez prostokąt
unsigned char div8 = a / 8;
// zmienna przechowująca ilość pozo-
stałych linii (nie tworzących pełnej
strony)
unsigned char mod8 = a % 8;
// pętla wykona się tyle razy, ile
stron zajmuje prostokąt
for(j = 0; j < div8; j++)
   {
   lcdGoTo(x,j); //
ustawienie współ-
rzędnych
   // pętla wykona
się tyle razy,
ile punktów jest
w linii
   for(i = 0; i <=
b; i++)
     {
 lcdWriteData-
(0xFF); // zapis
strony
 }
   }
// dorysowujemy
pozostałe linie
for(i = 0; i <=
mod8; i++)
   {
   // pętla wykona
się tyle razy,
ile punktów jest
w linii
   for(j = 0; j <=
b; j++)
     lcdSetPixel(x +
j, y + i + (a - mod8));
   }
}
```

Rysowanie okręgu

Funkcja przyjmuje 3 argumenty: współrzędne x i y środka okręgu oraz jego promień. Funkcja bazuje na algorytmie Brensenham'a. Szczegółowy opis algorytmu zawarty jest w [2].

```
void lcdCircle(unsigned char cx,
unsigned char cy ,unsigned char ra-
dius)
{
int x, y, xchange, ychange, radiu-
sError;
x = radius;
y = 0;
xchange = 1 - 2*radius;
ychange = 1;
radiusError = 0;
while(x >= y)
   {
   lcdSetPixel(cx+x, cy+y);
   lcdSetPixel(cx-x, cy+y);
   lcdSetPixel(cx-x, cy-y);
   lcdSetPixel(cx+x, cy-y);
   lcdSetPixel(cx+y, cy+x);
   lcdSetPixel(cx-y, cy+x);
   lcdSetPixel(cx-y, cy-x);
   lcdSetPixel(cx+y, cy-x);
   y++;
   radiusError += ychange;
   ychange += 2;
   if (2*radiusError + xchange > 0)
     {
     x--;
    radiusError += xchange;
    xchange += 2;
    }
   }
}
```

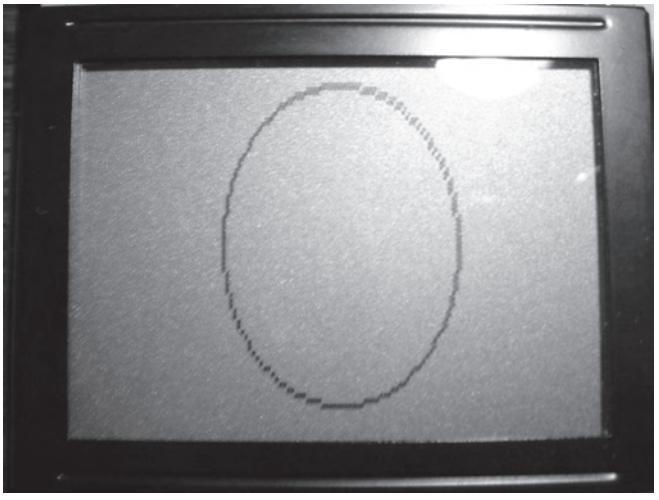
Efekt działania funkcji lcdCircle przedstawiono na **fot. 8**.

Wyświetlanie bitmapy

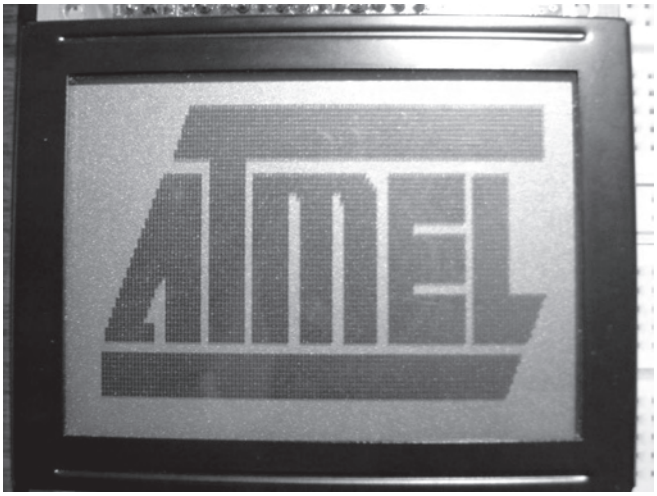
Rysowanie większych obrazków piksel po pikselu byłoby zadaniem bardzo czasochłonnym i skomplikowanym, dlatego też zajmiemy się teraz funkcją umożliwiającą wyświetlenie na wyświetlaczu obrazu w postaci bitmapy. Pozwoli to na przygotowanie obrazu w dowolnym edytorze graficznym na komputerze i jego łatwe wyświetlenie na wyświetlaczu. Bitmapa w pamięci mikrokontrolera musi zajmować kolejne komórki pamięci, zgodnie z kolejnością ich wyświetlania na wyświetlaczu. Format komputerowego pliku BMP posiada nieco inną organizację danych, utrudniającą proste wyświetlenie obrazu na wyświetlaczu LCD. Przygotowanie bitmapy do wyświetlenia będzie wymagało wykorzystania prostego program dla komputera PC, dokonującego konwersji pliku BMP na postać źródłową języka C (wygenerowana zostanie tablica zawierająca kolejne bajty przeznaczone do wyświetlenia). Program ten można pobrać ze strony http://radzio.dxp.pl/asystentlcd/. Dodatkowo w programie tym można zaprojektować ikony o rozmiarze 8x8 oraz 16x16 pikseli (oczywiście program generuje tablicę z danymi gotową do wklejenia do programu dla mikrokontrolera).

Funkcja wyświetlająca bitmapę przyjmuje 5 argumentów: wskaźnik do tablicy z bitmapą znajdującą się w pamięci Flash, współrzędne pod którymi bitmapa ma zostać wyświetlona oraz jej rozmiar. Współrzędna pionowa przyjmuje wartości 0...7 (numer strony). Wysokość bitmapy może przyjmować wartości będące wielokrotnością liczby 8. Natomiast współrzędna pozioma i szerokość bitmapy mogą przyjmować dowolne wartości (mieszczące się oczywiście na wyświetlaczu).

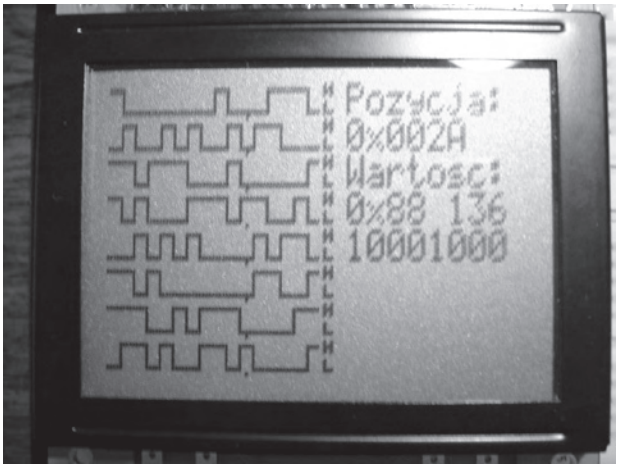
```
void lcdBitmap(char * bmp, unsigned
char x, unsigned char y, unsigned
char dx, unsigned char dy)
```



Fot. 8. Efekt działania funkcji lcdCircle



Fot. 9. Efekt działania funkcji lcdBitmap



Fot. 10. Przykład praktycznego wykorzystania wyświetlacza graficznego do obsługi analizatora stanów logicznych

```
{
unsigned char i, j;
// pętlawykona się tyle razy, ile
stron zajmuje bitmapa
for(j = 0; j < dy / 8; j++)
  {
  // ustawienie współrzędnych
  lcdGoTo(x,y + j);
  // pętla wykona się tyle razy, ile
pikseli szerokości ma bitmapa
  for(i = 0; i < dx; i++)
    {
    lcdWriteData(pgm_read_byte(bm-
p++)); // zapis danych
    }
  }
}
```

Efekt działania funkcji lcdBitmap przedstawiono na **fot. 9**.

Rysowanie kółek i linii jest mało efektownym przykładem zastosowania graficznego wyświetlacza LCD, spójrzmy więc na **fot. 10** przedstawiającą praktyczne zastosowanie wyświetlacza w prostym analizatorze stanów logicznych.

Mam nadzieję, że artykuł przybliżył Czytelnikom sposób obsługi wyświetlacza graficznego ze sterownikiem KS0108. Dzięki stosunkowo dużym możliwościom prezentacji danych takiego wyświetlacza oraz niskiej cenie może on znaleźć szerokie zastosowanie w najróżniejszych urządzeniach elektronicznych, gdzie wymagane jest duże pole odczytowe. Zaadaptowanie zaprezentowanych tu procedur na inny mikrokontroler nie powinno sprawić większych trudności, jako że użyto najbardziej uniwersalnego języka programowania mikrokontrolerów (język C).

Radosław Kwiecień
radoslaw.kwiecien@ep.com.pl

Literatura

1. Bresenham's Integer Only Line Drawing Algorithm, John Kennedy, http://homepage.smc.edu/kennedy_john/BRESENL.PDF
2. A Fast Bresenham Type Algorithm For Drawing Circles, John Kennedy, http://homepage.smc.edu/kennedy_john/BCIRCLE.PDF
3. Driver for graphic LCD display 128x64, Gregor Horvat