Wyświetlacze graficzne LCD ze sterownikiem KS0108 – sterowanie w języku C od podstaw, część 1

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.

Najtańsze wyświetlacze graficzne zbudowane są w oparciu o sterownik KS0107/KS0108 (HD61202/HD61203) bez generatora znaków. Cena takiego wyświetlacza o rozdzielczości 128x64 pikseli jest w przybliżeniu 3–krotnie większa od ceny zwykłego wyświetlacza alfanumerycznego 16x2, przy o wiele większych możliwościach prezentacji danych. Wyświetlacz o takiej rozdzielczości pozwala na wyświetlenie ośmiu linii po 21 znaków (typowa czcionka 5x7 pikseli), co daje łączną liczbę 168 znaków, oczywiście posiada także możliwość wyświetlenia grafiki. Brak wbudowanego generatora znaków wydaje się być poważną wadą wyświetlacza, jednak w prosty sposób można sobie z tym poradzić przez programową generację znaków przez mikrokontroler. Wszystkie niezbędne funkcje realizujące wyświetlanie tekstu wraz z tablicą znaków zapisaną w pamięci programu zajmują około 1 kB pamięci.

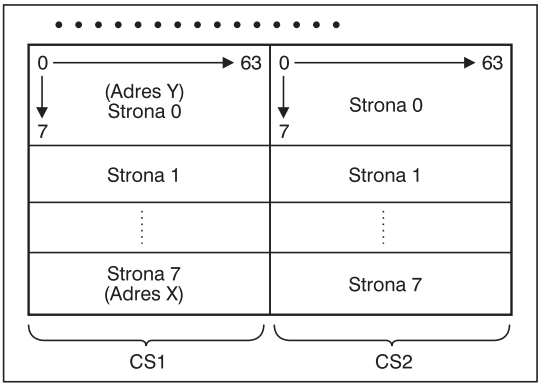
Tab. 1. Opis wyprowadzeń wyświetlacza graficznego JM12864A

Numer	Oznaczenie	Funkcja
1	/CS1	Wybór pierwszego kontrolera
2	/CS2	Wybór drugiego kontrolera
3	V_{ss}	Masa
4	V_{dd}	+5V
5	V_{ee}	Zasilanie LCD (regulowane)
6	RS	Wybór rejestru (0: instrukcje, 1:dane)
7	R/W	Kierunek przesyłu (0: zapis, 1: odczyt)
8	EN	Wykonanie operacji
9	DB0	Bit danych 0
10	DB1	Bit danych 1
11	DB2	Bit danych 2
12	DB3	Bit danych 3
13	DB4	Bit danych 4
14	DB5	Bit danych 5
15	DB6	Bit danych 6
16	DB7	Bit danych 7

„Na warsztat" weźmiemy wyświetlacz JM12864A produkowany przez firmę JHD. Wyświetlacz JM12864A posiada dwa kontrolery, aktywowane sygnałami /CS1 oraz /CS2 (aktywny stan niski). Opis wyprowadzeń tego wyświetlacza przedstawiono w **tab. 1**.

Wyświetlacze innych firm mogą posiadać inny rozkład wyprowadzeń, tak więc zawsze przed podłączeniem ich należy dokładnie zapoznać się z dokumentacją. Do sterowania wyświetlaczem wymaganych jest 13 wyprowadzeń mikrokontrolera (można tę liczbę zmniejszyć stosując ekspandery portów z szeregowym wejściem, np. PCF8574, ale spowolni to wymianę danych pomiędzy mikrokontrolerem, a wyświetlaczem). W przeciwieństwie do wyświetlaczy ze sterownikiem HD44780 nie jest możliwa transmisja w trybie 4–bitowym.

Wyświetlacz posiada 1 kB pamięci RAM. Organizację pamięci i jej odwzorowanie na ekran wyświetlacza przedstawia **rys. 1**. Adres Y odpowiada współrzędnej x ekranu (oś pozioma), natomiast adres X określa numer strony (czyli jest podzieloną przez 8 współrzędną y ekranu). Adres Z określa, która linia pamięci obrazu będzie odwzorowana w najwyższej linii wyświetlacza. Zapis danych do pamięci odbywa się po 8 bitów, najmłodszy bit odpowiada pikselowi położonemu w linii najwyższej w obrębie danej strony, a najstarszy bit odpowiada pikselowi położonemu w linii najniższej. Pokazano to na **rys. 2** (fragment dwóch pierwszych stron ekranu wyświetlacza).



Rys. 1. Organizacja pamięci i jej odwzorowanie na ekran wyświetlacza

Rozkazy sterownika KS0108

Sterownik KS0108 obsługuje 7 podstawowych rozkazów:

Display On/Off

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	1	1	1	1	D

D=0 – wyświetlanie zawartości pamięci RAM na ekranie wyłączone

D=1 – wyświetlanie zawartości pamięci RAM na ekranie włączone

Instrukcja nie ma wpływu na zawartość pamięci RAM.

Set Address (Y address)

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0

Ustawienie adresu Y wyświetlacza (czyli współrzędnej x ekranu) z zakre-

su 0...63. Adres jest automatycznie inkrementowany po każdej operacji odczytu, bądź zapisu danych.

Set Page (X address)

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	0	1	1	1	AC2	AC1	AC0

Ustawienie adresu X (wybór strony) wyświetlacza z zakresu 0...7. Wszelkie operacje zapisu i odczytu danych wykonywane są na bieżącej stronie do momentu wybrania nowej strony.

Display Start Line (Z address)

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	1	AC5	AC4	AC3	AC2	AC1	AC0

Adres Z określa, od której linii obraz zawarty w pamięci RAM ma zostać wyświetlony na ekranie.

Status read

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	1	BUSY	0	ON/ OFF	RE- SET	0	0	0	0

Status kontrolera:

BUSY=1 – kontroler wykonuje operację i nie przyjmuje kolejnych instrukcji

BUSY=0 – kontroler jest gotowy do przyjęcia kolejnej instrukcji

ON/OFF=1 – wyświetlacz jest włączony

ON/OFF=0 – wyświetlacz jest wyłączony

RESET=1 – trwa inicjalizacja wyświetlacza, żadne instrukcje za wyjątkiem instrukcji odczytu statusu nie są akceptowane

RESET=0 – inicjalizacja zakończona, kontroler w stanie normalnej pracy

Write Display Data

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
1	0	D7	D6	D5	D4	D3	D2	D1	D0

Zapis danej do pamięci RAM wyświetlacza pod aktualny adres Y. Po wykonaniu instrukcji następuje automatyczna inkrementacja adresu Y.

Read Display Data

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
1	1	D7	D6	D5	D4	D3	D2	D1	D0

Odczyt danej z pamięci RAM wyświetlacza spod aktualnego adresu Y. Po wykonaniu instrukcji następuje automatyczna inkrementacja adresu Y.

Komunikacja mikrokontrolera z wyświetlaczem

Szyna danych interfejsu wyświetlacza zajmuje cały dowolnie wybrany port, natomiast sygnały sterujące zajmują 5 wyprowadzeń drugiego dowolnie wybranego portu. Poszczególne sygnały sterujące można przypisać do dowolnych wyprowadzeń w ramach wybranego portu. Schemat połączenia wyświetlacza JM12864A z mikrokontrolerem ATMega16 jest przedstawiony na **rys. 3**. Napięcie ujemne konieczne do zasilania wyświetlacza jest wytwarzane przez układ MAX232 (w przypadku zastosowania wyświetlacza z wbudowanym generatorem napięcia ujemnego można ten układ pominąć).

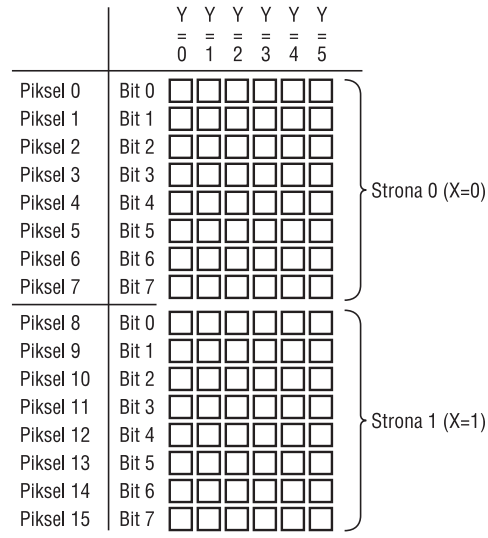
Przedstawione w artykule procedury zostały napisane w języku C i skompilowane kompilatorem avr–gcc. „Sterownik" wyświetlacza znajduje się w trzech plikach: JM12864A.c (kod procedur), JM12864A.h (definicje stałych, konfiguracja portów itp.) oraz font.h (tablica czcionek). Pliki te należy dołączyć do projektu korzystającego z procedur obsługi wyświetlacza.

W pliku JM12864A.h znajdują się definicje poszczególnych sygnałów sterujących:

```
// port szyny danych
// można ustawić dowolny
#define LCD_DATA_PORT PORTA
#define LCD_DATA_PIN PINA
#define LCD_DATA_DDR DDRA
// port sygnałów sterujących
// można ustawić dowolny
#define LCD_CTRL_PORT PORTC
#define LCD_CTRL_PIN PINC
#define LCD_CTRL_DDR DDRC
// sygnały sterujące
// wszystkie sygnały należą do portu
CTRL, kolejność dowolna
#define LCD_CS1P PC6
#define LCD_CS2P PC5
#define LCD_EN PC0
#define LCD_RW PC1
#define LCD_RS PC2
```

Następnie zdefiniowanych jest kilka podstawowych makroinstrukcji kontrolujących stan sygnałów sterujących:

```
// makroinstrukcje ustawienia stanu
na linii CS1
#define SET_CS1() (LCD_CTRL_PORT |=
(1 << LCD_CS1P))
#define CLR_CS1() (LCD_CTRL_PORT &=
~(1 << LCD_CS1P))
// makroinstrukcje ustawienia stanu
na linii CS2
#define SET_CS2() (LCD_CTRL_PORT |=
(1 << LCD_CS2P))
#define CLR_CS2() (LCD_CTRL_PORT &=
~(1 << LCD_CS2P))
```



Rys. 2. Strona pamięci w powiększeniu

```
// makroinstrukcje ustawienia stanu
na linii EN
#define SET_EN() (LCD_CTRL_PORT |= (1
<< LCD_EN))
#define CLR_EN() (LCD_CTRL_PORT &=
~(1 << LCD_EN))
// makroinstrukcje ustawienia stanu
na linii RW
#define SET_RW() (LCD_CTRL_PORT |= (1
<< LCD_RW))
#define CLR_RW() (LCD_CTRL_PORT &=
~(1 << LCD_RW))
// makroinstrukcje ustawienia stanu
na linii RS
#define SET_RS() (LCD_CTRL_PORT |= (1
<< LCD_RS))
#define CLR_RS() (LCD_CTRL_PORT &=
~(1 << LCD_RS))
// makroinstrukcje ustawiające od-
powiednią kombinację sygnałów CS1
i CS2
#define LCD_CS0() CLR_CS1();SET_
CS2();
#define LCD_CS1() SET_CS1();CLR_
CS2();
#define LCD_NOCS() SET_CS1();SET_
CS2();
```

Oraz definicje rozkazów sterownika KS0108:

```
#define DISPLAY_SET_Y 0x40
#define DISPLAY_SET_X 0xB8
#define DISPLAY_START_LINE 0xC0
#define DISPLAY_ON_CMD 0x3E
#define ON 0x01
#define OFF 0x00
#define DISPLAY_STATUS_BUSY0x80
```

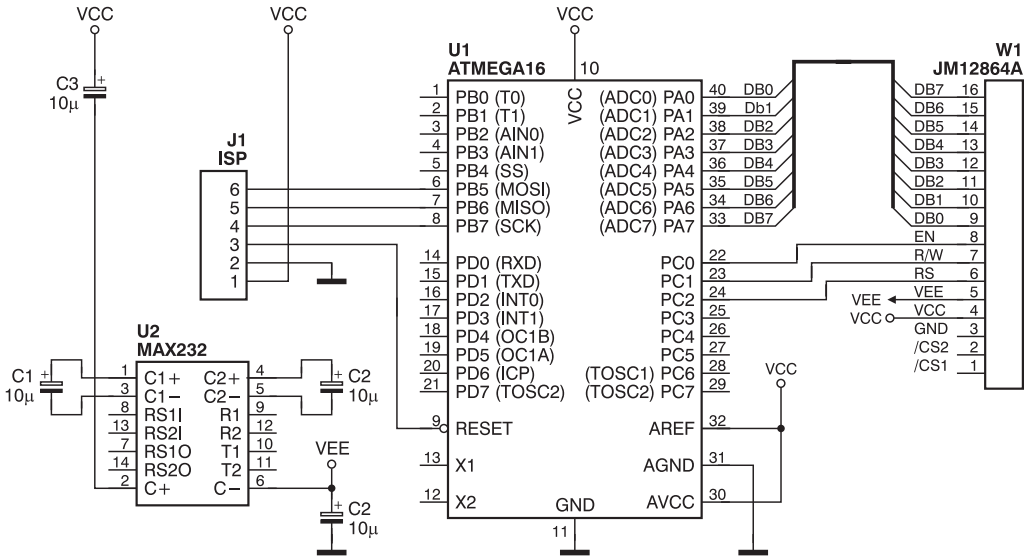
W programie (plik JM12864A.c) należy zadeklarować globalne zmienne przechowujące współrzędne kursora, gdyż nie jest możliwy odczyt z wyświetlacza aktualnej pozycji kursora w:

```
unsigned char lcd_x, lcd_y;
```

Funkcje pomocnicze

Pierwszą funkcją będzie funkcja konfigurująca porty wykorzystywane do komunikacji z wyświetlaczem w tryb wyjściowy. Wyświetlacz JM12864A nie wymaga dodatkowej inicjalizacji (poza wyczyszczeniem ekranu) jak to miało miejsce w wyświetlaczu alfanumerycznym.

```
void lcdInit(void)
{
LCD_DATA_DDR = 0xFF;
```



Rys. 3. Schemat dołączenia wyświetlacza JM12864 do mikrokontrolera

```
LCD_CTRL_DDR |= ((1 << LCD_CS1P) |
(1 << LCD_CS2P) |
   (1 << LCD_RS) | (1 << LCD_RW) | (1
<< LCD_EN));
}
```

Kolejna funkcja wprowadza opóźnienie wymagane przez sterownik wyświetlacza podczas operacji odczytu oraz zapisu danych bądź instrukcji:

```
void delay(void)
{
asm("nop");asm("nop");
}
```

Dokumentacja sterownika określa minimalny czas trwania stanu na linii EN na 450 ns. Przeprowadzone przeze mnie eksperymenty wykazały, że w rzeczywistości dodawanie opóźnień jest zbędne. Wyświetlacz poprawnie współpracował z mikrokontrolerem taktowanym sygnałem o częstotliwości 16 MHz bez żadnych dodatkowych opóźnień. Niemniej jednak, aby mieć pewność poprawnego działania należy spełnić wymogi czasowe transmisji zalecane przez producenta kontrolera.

Sprawdzanie zajętości kontrolera

Przed zapisaniem instrukcji do wyświetlacza należy sprawdzić flagę zajętości kontrolera. Do czasu zakończenia wykonywania poprzedniej instrukcji kontroler ignoruje kolejne instrukcje. W tym celu zdefiniujmy następującą funkcję:

```
void lcdWait(void)
{
LCD_DATA_DDR = 0x00; // ustawienie
portu danych w tryb wejściowy
CLR_RS(); // niski stan na linii RS
-> odczyt rejestru statusu
SET_RW(); // wysoki stan na linii RW
-> odczyt z wyświetlacza
do { //pętla
   delay(); // opóźnienie
   SET_EN();  // ustaw linię EN
   delay(); // opóźnienie
    CLR_EN();  // wyzeruj linię EN
  } while((LCD_DATA_PIN & DISPLAY_
STATUS_BUSY)); // powtarzaj do
// wyzerowania flagi BUSY
}
```

Zapis instrukcji

Zapis instrukcji odbywa się przy niskim stanie linii RS i RW. Opadające zbocze na linii E zatrzaskuje dane w rejestrze wejściowym. Funkcja zapisuje instrukcję do aktywnego w danej chwili kontrolera. Jeśli instrukcja ma być zapisana do obydwu kontrolerów należy ją wywołać dwukrotnie ustawiając przed każdym wywołaniem odpowiedni stan na liniach CS1 i CS2.

```
void lcdWriteCmd(unsigned char cmd)
{
lcdWait(); // oczekiwanie na goto-
wość kontrolera
CLR_RS(); // niski stan na linii RS
-> rozkaz
CLR_RW(); // niski stan na linii RW
-> zapis
LCD_DATA_DDR = 0xFF; // port danych
-> wyjście
LCD_DATA_PORT = cmd; // wystawienie
na port instrucji
SET_EN(); // ustawienie linii EN
delay(); // opóźnienie
CLR_EN(); // wyzerowanie linii EN
}
```

Zapis danych

Dane są zapisywane przy wysokim stanie linii RS i niskim stanie linii RW. Opadające zbocze na linii E zatrzaskuje dane w rejestrze wejściowym. Poniższa funkcja dokonuje sprawdzenia aktualnej pozycji i w zależności od niej odpowiednio ustawiana jest kombinacja sygnałów CS1 i CS2.

```
void lcdWriteData(unsigned char
data)
{
if(lcd_x < 64) // jeśli współrzędna
x wyświetlacza < 64
  {LCD_CS0()}  // to zapisujemy do
pierwszego kontrolera
else    // w przeciwnym ra-
zie
  {LCD_CS1()}  // zapisuje-
my do drugiego kontrolera
lcdWait();  // oczekiwanie
na gotowość kontrolera
SET_RS();   // wysoki stan
na linii RS -> dane
CLR_RW();   // niski stan
na linii RW -> zapis
LCD_DATA_DDR = 0xFF; //
port danych -> wyjście
LCD_DATA_PORT = data; //
wystawienie na port danej
SET_EN(); // wysoki stan na
linii EN
delay(); // opóźnienie
CLR_EN(); // niski stan na
linii EN
lcd_x++; // zwiększenie
współrzędnej x wyświetlacza
(pomocniczej)
if(lcd_x > 127) // jeśli
koniec ekranu
  lcd_x = 0; // to wyzeruj
współrzędną x
LCD_NOCS();
}
```

Odczyt danych

Dane odczytywane są przy wysokim stanie linii RS i RW.

```
unsigned char lcdReadData(void)
{
unsigned char data;
if(lcd_x < 64)  // jeśli współrzędna
x wyświetlacza < 64
  {LCD_CS0()}   // to odczytujemy
z pierwszego kontrolera
else // w przeciwnym razie
  {LCD_CS1()}   // odczytujemy
z drugiego kontrolera
lcdWait(); // oczekiwanie na goto-
wość kontrolera
SET_RS();  // wysoki stan na linii
RS -> dane
SET_RW();  // wysoki stan na linii
RW -> odczyt
SET_EN();  // wysoki stan na linii
EN
delay();   // opóźnienie
LCD_DATA_DDR = 0x00; // ustawienie
portu danych w tryb wejsciowy
data = LCD_DATA_PIN; // odczyt da-
nych z portu
CLR_EN();  // niski stan na linii EN
lcd_x++; // zwiększenie współrzędnej
x wyświetlacza
if(lcd_x > 127) // jesli koniec
ekranu
  lcd_x = 0; // to wyzeruj współ-
rzędną x
LCD_NOCS();
return data;
}
```

Ustawienie współrzędnych wyświetlania

Funkcja ustawia współrzędne wyświetlacza, na których wykonana zostanie następna operacja (odczyt, bądź zapis). Ze względu na organizację pamięci wyświetlacza współrzędna pionowa może przyjmować wartości z zakresu 0...7 (numer strony). Współrzędna pozioma może przyjmować wartości z zakresu 0...127.

```
void lcdGoTo(unsigned char x, unsi-
gned char y)
{
lcd_x = x; // przypisanie współrzę-
dym globalnym nowych wartości
lcd_y = y;
if(lcd_x > 63) // jeśli współrzędna
pozioma jest większa od 64 to
  {
  LCD_CS1(); // uaktywnienie drugie-
go kontrolera
  lcdWriteCmd(DISPLAY_SET_X | lcd_
y); // zapis współrzędnej pionowej
  lcdWriteCmd(DISPLAY_SET_Y | (lcd_x
```

```
- 64)); // zapis współrzędnej poziomej
  }
else // w przeciwnym razie
  {
  LCD_CS0(); // uatywnienie pierwszego kontrolera
  lcdWriteCmd(DISPLAY_SET_X | lcd_y); // zapis współrzędnej pionowej
  lcdWriteCmd(DISPLAY_SET_Y | lcd_x); // zapis współrzędnej poziomej
  LCD_CS1(); // uaktywnienie drugiego kontrolera
  lcdWriteCmd(DISPLAY_SET_X | lcd_y); // zapis współrzędnej pionowej
  lcdWriteCmd(DISPLAY_SET_Y | 0); // wyzerowanie współrzędnej poziomej
  }
LCD_CS0(); // uaktywnienie pierwszego kontrolera
lcdWriteCmd(DISPLAY_START_LINE | 0);
//
LCD_CS1(); // uaktywnienie drugiego kontrolera
lcdWriteCmd(DISPLAY_START_LINE | 0);
LCD_NOCS();
}
```

Włączenie wyświetlacza

Włączenie wyświetlania zawartości pamięci RAM na ekranie następuje po wykonaniu instrukcji *Display On*. Instrukcję należy wysłać do każdego kontrolera oddzielnie:

```
void lcdOn(void)
{
LCD_CS0(); // aktywny pierwszy kontroler
lcdWriteCmd(DISPLAY_ON_CMD | ON); // włączenie wyświetlacza
LCD_CS1(); // aktywny drugi kontroler
lcdWriteCmd(DISPLAY_ON_CMD | ON); // włączenie wyświetlacza
LCD_NOCS();
}
```

Wyłączenie wyświetlacza

Wyłączenie wyświetlania zawartości pamięci RAM na ekranie następuje po wykonaniu instrukcji *Display Off*

```
void lcdOff(void)
{
LCD_CS0(); // aktywny pierwszy kontroler
lcdWriteCmd(DISPLAY_ON_CMD | OFF);
// wyłączenie wyświetlacza
LCD_CS1(); // aktywny drugi kontroler
lcdWriteCmd(DISPLAY_ON_CMD | OFF);
// włączenie wyświetlacza
LCD_NOCS();
}
```

Czyszczenie wyświetlacza

Czyszczenie zawartości wyświetlacza polega na wypełnieniu pamięci danych odpowiednim wzorcem. Jeśli wyświetlacz ma wyświetlać obraz pozytywowy pamięć należy wypełnić zerami, natomiast gdy wyświetlacz ma wyświetlać obraz negatywowy pamięć należy wypełnić bajtami o wartości 255.

```
void lcdCls(void)
{
unsigned char x, y; // pomocnicze zmienne
for (y = 0; y < 8; y++) // 8-krotne powtórzenie petli
  {
  lcdGoTo(0,y); // ustawienie współrzędnej y wyświetlacza
  for (x = 0; x < 128; x++) // 128--krotne powtórzenie pętli
    lcdWriteData(0); // zapis do pamięci wyświetlacza wzorca
  }
lcdGoTo(0,0); // ustawienie początkowych współrzędnych
}
```

Rozkazy związane z trybem tekstowym oraz kilka przykładów funkcji graficznych przedstawimy w kolejnym odcinku.

Radosław Kwiecień, EP
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