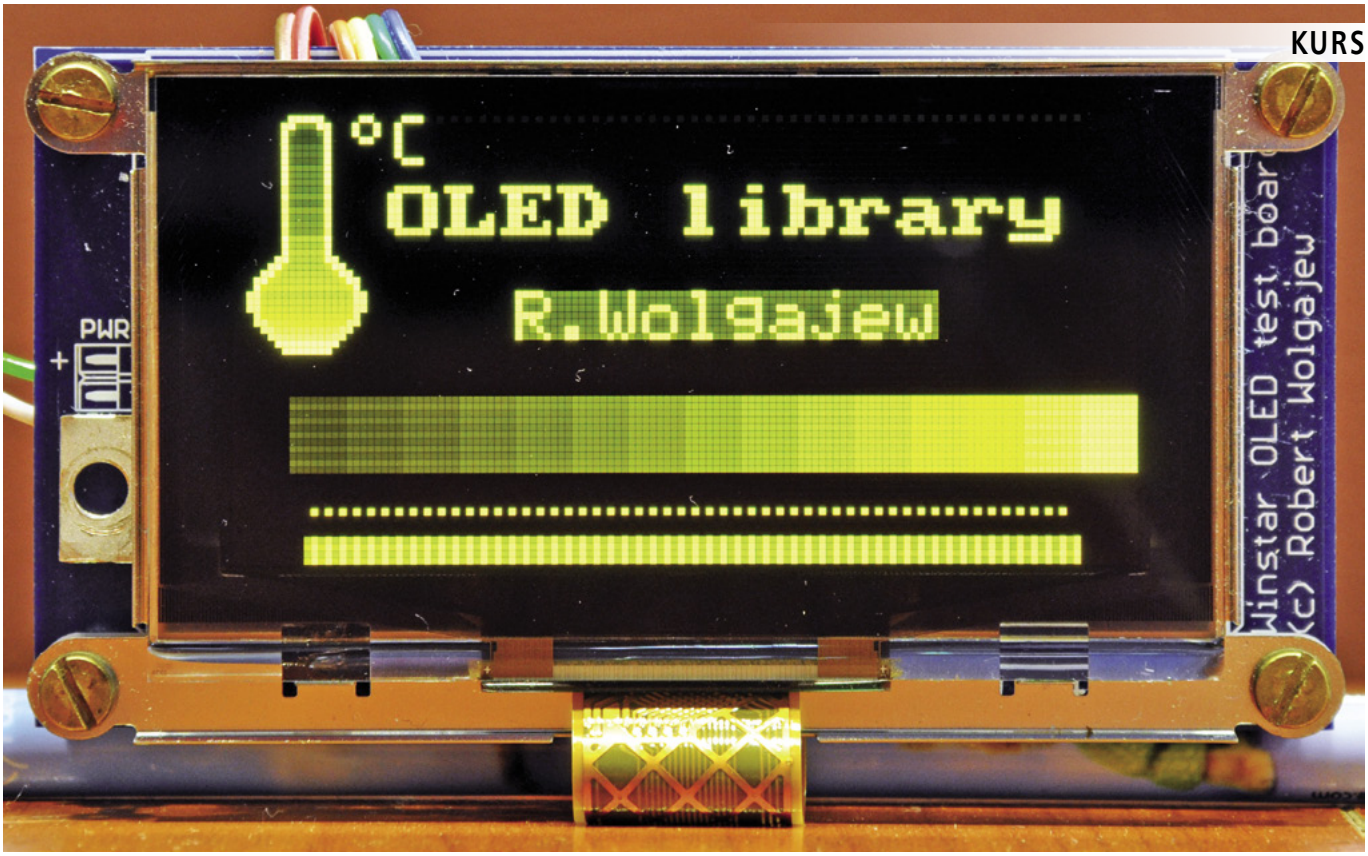


Obsługa monochromatycznych wyświetlaczy OLED ze sterownikiem SSD1325 za pomocą Bascom AVR (2)

Kontynuujemy prezentację obsługi wyświetlacza OLED wyposażonego w sterownik SSD1325 za pomocą języka Bascom dla mikrokontrolerów AVR.

Dodatkowe informacje:
Testy przeprowadzono z wykorzystaniem modułu Winstar WEGC012864AL udostępnionego przez firmę Unisystem.

Dodatkowe materiały na CD i FTP:
ftp://ep.com.pl, user: 17855, pass: 4s406qj2
- poprzednia część artykułu

Po wywołaniu procedury inicjalizacyjnej wyświetlacz jest już gotowy na przyjęcie danych (komend sterujących czy danych dla pamięci obrazu). Poniżej przedstawię kilka użytecznych procedur związanych z oprogramowywaniem podstawowych funkcji panelu. Należy szczególnie podkreślić fakt, iż procedury te nie zostały zoptymalizowane pod kątem użycia pamięci RAM mikrokontrolera celowo. Dzięki temu ich kod jest bardziej przejrzysty, a więc prostszy, jeśli chodzi o możliwość własnej implementacji. Pierwszą z nich będzie procedura pozwalająca zdefiniować aktywny obszar ekranu dla operacji zapisu. Jest to bardzo użyteczna procedura, której użycie upraszcza, a zarazem znacznie przyspiesza wyświetlanie obrazów, których rozmiar jest inny aniżeli całkowity, fizyczny rozmiar ekranu (**listing 4**).

Dla uproszczenia zapisu do pamięci obrazu GDDRAM przyjmuje się, że dopuszczalne wartości argumentów dla osi X muszą być parzyste, co wydaje się niewielkim ograniczeniem przedstawionej procedury i kolejnych. Teraz pora na trzy procedury: rysujące prostokąt oraz poziomą i pionową linię (dwie z nich korzystają z mechanizmu akceleracji sprzętowej sterownika SSD1325) – **listing 5, 6** i **7**.

Kolejna procedura pozwala wyświetlać napisy na ekranie panelu. Sterownik SSD1325, tak jak wiele innych sterowników, nie jest wyposażony we własny generator znaków. Jest to przykładowe, rzekłbym, typowe i zarazem edukacyjne rozwiązanie bazujące na definicji czcionki dostarczanej przez producenta kompilatora Bascom. Mowa o czcionce Color8x8.font, której definicja nie jest niczym innym jak zbiorem bajtów poszczególnych linii każdego znaku uszeregowanych według ich kodów ASCII. Szczegółów dotyczących struktury pliku typu **.font* (w tym przypadku koniecznie w wersji dla wyświetlaczy kolorowych) należy szukać w pliku pomocy pakietu Bascom. Plik z definicją takiej czcionki należy dołączyć do naszego programu następującą deklaracją:

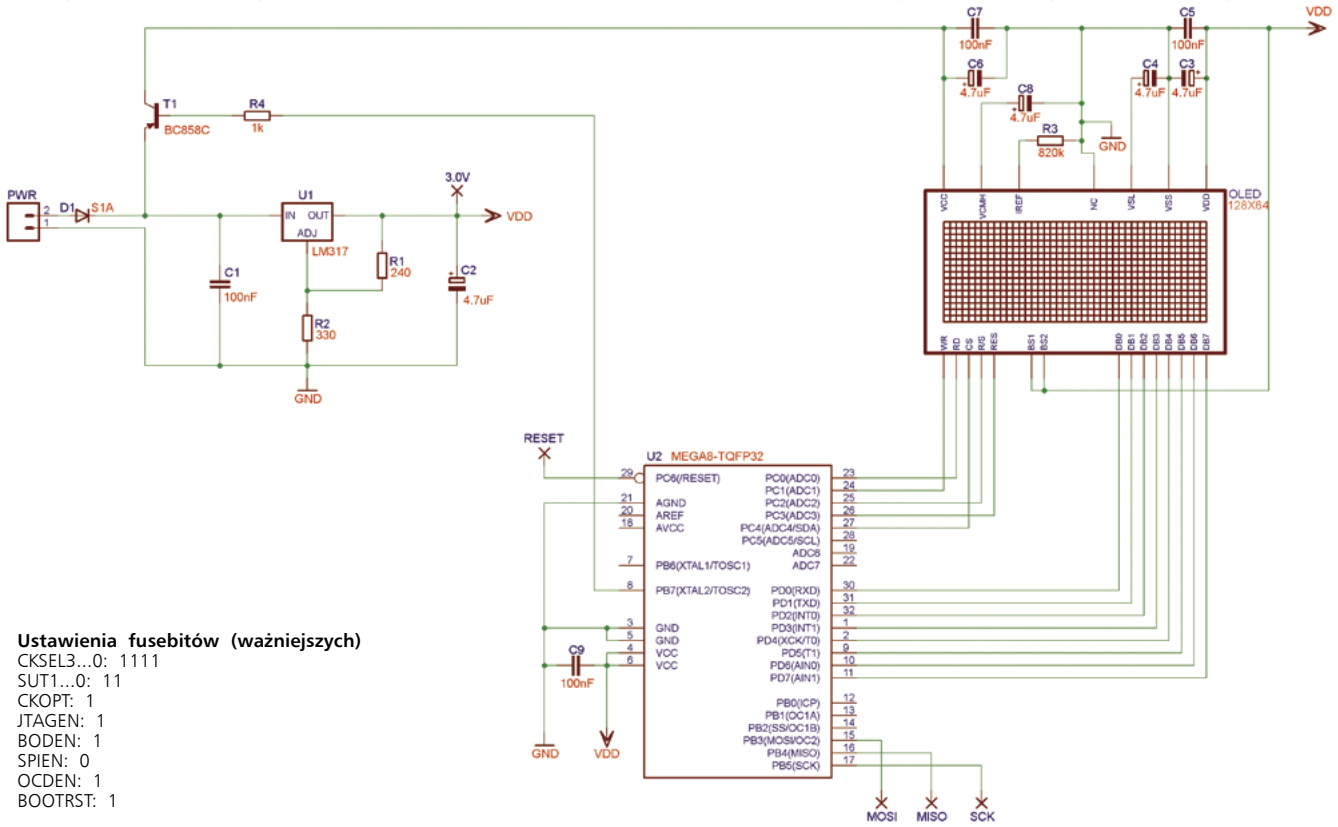
$include „color8x8.font”

Procedura odpowiedzialna za wyświetlanie napisów przedstawia się następująco (Picture_pointer i Offset to zmienne globalne typu *Word*) – **listing 8**.

Na deser przedstawię procedurę pozwalającą na wyświetlanie grafik na ekranie panelu OLED. Oczywiście obraz ten musi

Zestaw testowy

Procedury prezentowane w artykule przetestowano na bazie zestawu z mikrokontrolerem ATmeg8, którego schemat elektryczny pokazano niżej.



Ustawienia fusebitów (ważniejszych)

CKSEL3...0: 1111
SUT1...0: 11
CKOPT: 1
JTAGEN: 1
BODEN: 1
SPIEN: 0
OCDEN: 1
BOOTRST: 1

List. 4. Procedura definiująca aktywny obszar ekranu panelu OLED

```
,X1 i X2: 0...127 (tylko parzyste), Y1 i Y2: 0...63
Sub Set_active_region(byval X1 As Byte , Byval Y1 As Byte , Byval X2 As Byte , Byval Y2 As Byte)

 Call Write_cmnd(&H15)  ,Set Column Address
 Shift X1 , Right , 1
 Call Write_cmnd(x1)    ,Adres startowy aktywnego obszaru pamięci ekranu dla osi X
 Shift X2 , Right , 1
 Call Write_cmnd(x2)    ,Adres końcowy aktywnego obszaru pamięci ekranu dla osi X

 Call Write_cmnd(&H75)  ,Set Row address
 Call Write_cmnd(y1)    ,Adres startowy aktywnego obszaru pamięci ekranu dla osi Y
 Call Write_cmnd(y2)    ,Adres końcowy aktywnego obszaru pamięci ekranu dla osi Y

End Sub
```

List. 5. Procedura rysowania prostokąta

```
,X1 i X2: 0...127 (tylko parzyste), Y1 i Y2: 0...63
Sub Rectangle(byval X1 As Byte , Byval Y1 As Byte , Byval X2 As Byte , Byval Y2 As Byte , Byval Pattern As Byte)

 ,Korzystamy z akceleracji sprzętowej: funkcji rysowania prostokąta
 Call Write_cmnd(&H23)  ,Graphic acceleration command options
 Call Write_cmnd(&H01)  ,Enable filling
 Call Write_cmnd(&H24)  ,Draw Rectangle
 Shift X1 , Right , 1
 Call Write_cmnd(x1)
 Call Write_cmnd(y1)
 Shift X2 , Right , 1
 Call Write_cmnd(x2)
 Call Write_cmnd(y2)
 Call Write_cmnd(pattern)

End Sub
```

List. 6. Procedura rysowania linii poziomej

```
,X1: 0...127 (tylko parzyste), Y1: 0...63, Colour: określa wzór dla linii tak jak w przypadku rysowania prostokąta
Sub Draw_hline(byval X1 As Byte , Byval Y1 As Byte , Byval Length As Byte , Byval Colour As Byte)
Local X2 As Byte

  X2 = X1 + Length
  Decr X2
  Call Rectangle(x1 , Y1 , X2 , Y1 , Colour)       ,Korzystamy z akceleracji sprzętowej sterownika SSD1325

End Sub
```

spełniać pewne założenia, jeśli chodzi o jego strukturę, co wynika z bieżącej konfiguracji sterownika SSD1325, organizacji pamięci wyświetlacza OLED oraz z chęci uproszczenia kodu programu obsługi w tym zakresie. Plik zawierający obraz przeznaczony do wyświetlenia należy dołączyć do programu obsługi następującą deklaracją:

$inc Obrazek , Nosize , „Obrazek.opf”

Struktura wspomnianego pliku (w przykładzie jest to „Obrazek.opf”) jest bardzo prosta, a przedstawia się następująco:

List. 7. Procedura rysowania linii pionowej

```
,X1: 0...127 (tylko parzyste), Y1: 0...63, Colour: 0...15 (określa jasność linii)
Sub Draw_vline(byval X1 As Byte , Byval Y1 As Byte , Byval Length As Byte , Byval Colour As Byte)
Local Y2 As Byte

  Shift Colour , Left , 4
  Y2 = Y1 + Length
  Decr Y2
  Call Set_active_region(x1 , Y1 , X1 , Y2)       ,Określamy aktywny obszar ekranu w celu uproszczenia operacji zapisu
do pamięci GDDRAM

  For Y2 = 1 To Length
    Call Write_gddram(colour)
  Next Length
End Sub
```

List. 8. Procedura wyświetlania tekstów

```
,X1: 0...127 (tylko parzyste), Y1: 0...63
Sub Draw_text(byval X1 As Byte , Byval Y1 As Byte , Byval Colour As Byte , Byval Background As Byte , Text As String)
  Local Char_index As Byte
  Local Char As String * 1
  Local Index As Byte
  Local A As Byte
  Local B As Byte
  Local C As Byte
  Local 2pixels As Byte

 Colour = Colour And &H0F
 Background = Background And &H0F

 For Char_index = 1 To Len(text)       ,Pętla po kolejnych znakach zmiennej Text

     Char = Mid(text , Char_index , 1)       ,Pobieramy kolejny znak ze zmiennej tekstowej Text
     Offset = Asc(char) - 32       ,Obliczamy wskaźnik do miejsca w tablicy fontów, gdzie znajduje się początek
definicji znaku o odczytanym kodzie ASCII
     Shift Offset , Left , 3       ,Mnożymy otrzymaną wartość przez 8, gdyż każda definicja znaku zajmuje 8 bajtów
     Offset = Offset + 4       ,Pomijamy 4 bajty opisujące specyfikację czcionki, gdyż używamy wyłącznie czcionki 8x8

     ,Obliczmy współrzędne okna dla wyznaczenia aktywnego obszaru wyświetlacza, aby uprościć zapis bajtów
     ,do pamięci sterownika. W ten sposób poruszamy się wyłącznie w zakresie wyznaczonego okna pamięci obrazu.
     A = X1 + 7
     B = Y1 + 7
     Call Set_active_region(x1 , Y1 , A , B)

     Picture_pointer = Loadlabel(color8x8) + Offset       ,Ustawiamy wskaźnik na dane używanej czcionki plus niezbędny
offset wynikający z kodu ascii wyświetlanego znaku
     Lds _DPTRL , {Picture_Pointer}       ,Do bibliotecznych zmiennych Bascoma (symboliczne nazwy wybranych rejestrów)
wczytujemy wskaźnik do tych danych
     Lds _DPTRH , {Picture_Pointer+1}


     For Index = 1 To 8       ,Czytamy 8 kolejnych bajtów przypadających na definicję znaku
        Read C

        For A = 0 To 6 Step 2
          B = A + 1

          If C.a = 0 Then 2pixels = Background Else 2pixels = Colour
          Shift , 2pixels , Left , 4
          If C.b <> 0 Then 2pixels = 2pixels Or Colour Else 2pixels = 2pixels Or Background
          Call Write_gddram(2pixels)       ,Wysłanie danych 2 pikseli do pamięci GDDRAM sterownika SSD1325
        Next A

     Next Index

     X1 = X1 + 8       ,Zwiększamy wartości X1, by ustalić miejsce startowe do wyświetlenia kolejnych znaków
  Next Char_index

End Sub
```

- pierwszy bajt określa szerokość obrazka,
- drugi bajt określa wysokość obrazka,
- pozostałe bajty to dane kolejnych pikseli obrazu, przy czym zgodnie z organizacją pamięci sterownika SSD1325, każdy wspomniany bajt przechowuje informację o dwóch, kolejnych punktach obrazu. Organizacja obrazu odpowiada bieżącej konfiguracji sterownika SSD1325, co pokazano na **rysunku 4** (z tą wszakże różnicą, że zakres dla osi Y wynosi: 0x00…0x3F i jest zgodny z treścią procedury inicjalizacyjnej). Ponadto, co należy szczególnie podkreślić, dla bajtów będących danymi kolejnych pikseli obrazu zastosowano prosty algorytm kompresji danych eliminujący powtórzenia tych samych wartości, a co za tym idzie oszczędzający niezbędne miejsce pamięci Flash mikrokontrolera, w której przechowywane są obrazki przeznaczone do wyświetlenia. Ideę działania algorytmu kompresującego najlepiej jest prześledzić, analizując przykładowe ciągi bajtów (założono powtarzanie się bajtów o wartości 0xAA), które przedstawiono w **tabeli 2**.

Jak widać, idea działania wspomnianego algorytmu jest bardzo prosta, a pomimo to, dla typowych obrazków przeznaczonych dla wyświetlaczy OLED, osiągany stopień kompresji jest całkiem spory, co pozytywnie wpływa na użycie pamięci Flash mikrokontrolera. Kod procedury wyświetlającej skompresowany obrazek pokazano na **listingu 9**.

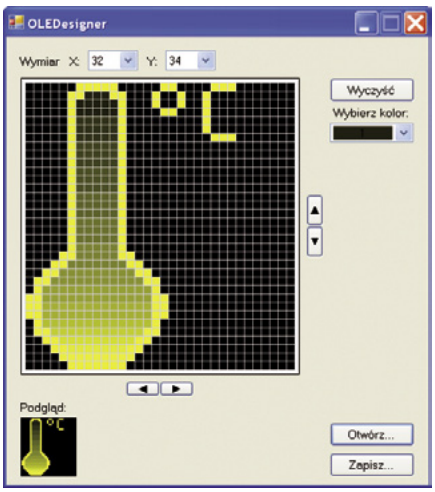
Tab. 2. Idea działania algorytmu kompresującego dane obrazków

Liczba powtórzeń	Dane przed kompresją	Dane po kompresji
1	0xAA	0xAA
2	0xAA 0xAA	0xAA 0xAA 0x00
3	0xAA 0xAA 0xAA	0xAA 0xAA 0x01
4	0xAA 0xAA 0xAA 0xAA	0xAA 0xAA 0x02
5	0xAA 0xAA 0xAA 0xAA 0xAA	0xAA 0xAA 0x03
257	257 x 0xAA	0xAA 0xAA 0xFF

List. 9. Procedura wyświetlająca obraz

```
,X1: 0...127 (tylko parzyste), Y1: 0...63
Sub Show_picture(byval X1 As Byte , Byval Y1 As Byte , Bylabel Mylabel As Word)
 Local A As Byte , B As Byte , Skip_a As Byte
 Local Bytes As Word


  Picture_pointer = Mylabel       ,Ustawienie wskaźnika do danych obrazka, którego nazwa jest argumentem wywołania
procedury
  Lds _DPTRL , {Picture_Pointer}       ,Do bibliotecznych zmiennych Bascoma (symboliczne nazwy wybranych rejestrów)
wczytujemy wskaźnik do tych danych
  Lds _DPTRH , {Picture_Pointer+1}

  Read A       ,Wczytujemy szerokość obrazka
  Read B       ,Wczytujemy wysokość obrazka
  Bytes = A / 2       ,Obliczamy liczbę bajtów opisujących obrazek
  Bytes = Bytes * B

  ,Ustalamy współrzędne prawego, dolnego rogu obrazka
  A = A + X1
  Decr A
  B = B + Y1
  Decr B

  ,Ustawiamy aktywny obszar ekranu, by uprościć zapis do pamięci GDDRAM
  Call Set_active_region(x1 , Y1 , A , B)

  Skip_a = 0
  Do
    If Skip_a = 0 Then Read A
    Read B

    If A <> B Then
         Call Write_gddram(a)       ,Wysłanie odczytanego bajta (dane 2 pikseli) do pamięci GDDRAM
         Decr Bytes
         A = B
         Skip_a = 1
         ,Jeśli to ostatni jeszcze niewysłany bajt to wysyłamy go w tym miejscu, gdyż inaczej zostałby pominięty
         If Bytes = 1 Then
           Call Write_gddram(b)       ,Wysłanie odczytanego bajta (dane 2 pikseli) do pamięci GDDRAM
           Bytes = 0
         End If
    Else
         ,Dwa bajty o tej samej wartości świadczą o kompresji danych w tym miejscu
         Call Write_gddram(b)       ,Wysłanie odczytanego bajta (dane 2 pikseli) do pamięci GDDRAM
         Call Write_gddram(b)       ,Wysłanie odczytanego bajta (dane 2 pikseli) do pamięci GDDRAM
         Bytes = Bytes - 2
         Read A       ,Odczytujemy liczbę powtórzeń - zmienna A użyta jako tymczasowa
         While A <> 0
           Call Write_gddram(b)       ,Wysłanie odczytanego bajta (dane 2 pikseli) do pamięci GDDRAM
           Decr Bytes
           Decr A
         Wend
         Skip_a = 0
    End If
  Loop Until Bytes = 0
  ,Przywracamy ustawienia ekranu
  Call Set_active_region(0 , 0 , 127 , 63)
End Sub
```

Uważny Czytelnik zada sobie z pewnością pytanie: w jaki sposób pozyskać obrazy do wyświetlenia, zważywszy na to, że struktura stosownych plików jest nietypowa, biorąc pod uwagę powszechne rozwiązania? Odpowiedź jest prosta! Do tego celu przewidziano specjalny konwerter pod postacią aplikacji *OLEDesigner*, której autorem jest **Marcin Popławski**. Jest to, co oczywiste, jedno z możliwych rozwiązań tego problemu, lecz moim zdaniem na tyle uniwersalne, komfortowe i proste w użyciu, że może być podstawowym narzędziem dla monochromatycznych wyświetlaczy OLED. Wygląd okna programu OLED Designer przedstawiono na **rysunku 5**.



Rys. 5. Wygląd okna programu OLEDesigner

Jak widać, za pomocą tego prostego, ale jakże użytecznego narzędzia jesteśmy w stanie zaprojektować dowolny element przyszłego interfejsu użytkownika tworzonego urządzenia. Co więcej, oprócz naszego specjalnego formatu pliku nazwanego OPF (od *Oled Picture File*) możemy otworzyć dowolny obraz w formacie BMP, który zostanie automatycznie przekształcony na tryb monochromatyczny. Maksymalny rozmiar takiego obrazu, ze względu na ograniczenia wyświetlacza, to 128 × 64 piksele. W przypadku, gdy wczytany obraz będzie większy, zostanie on automatycznie przycięty do tego rozmiaru. Otrzymany w ten sposób obraz możemy dowolnie edytować. Sama edycja jest bardzo intuicyjna. Po wyborze odpowiedniego koloru z rozwijanej listy i wciśnięciu lewego przycisku myszy nanosimy wybrany kolor na obraz. W celu łatwiejszej identyfikacji kolory w palecie są ponumerowane. Pod prawym przyciskiem myszy kryje się gumka, za pomocą której usuwamy niepotrzebny kolor. Na koniec efekty naszej pracy możemy zapisać do pliku OPF lub jako mapę bitową. Co oczywiste, plik OPF korzysta z opisanego wcześniej algorytmu kompresji danych.

Na tym zakończę opis podstawowych procedur związanych z obsługą paneli tego typu, mając jednocześnie nadzieję, że ten krótki kurs zachęci Was, drodzy Czytelnicy, do własnych eksperymentów z peryferiami tego typu, tym bardziej że ich cena ciągle spada, a własności użytkowe są nie do przecenienia. Wystarczy zuważyć, że coraz więcej producentów telefonów komórkowych zaczyna stosować w swoich urządzeniach aktywne martyce wykonane w tej technologii oraz coraz częściej mówi się o telewizorach OLED z matrycami o wielkich rozmiarach, które połączyłyby zalety technologii LCD z technologią plazmową. Cóż, to niezbyt odległa przyszłość!

Robert Wołgajew, EP