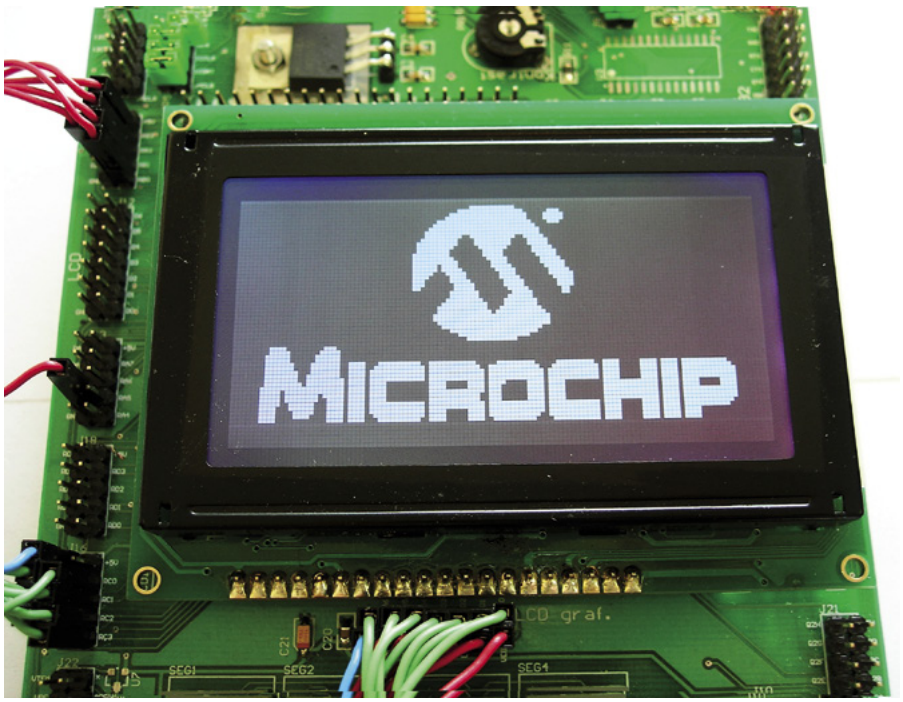
Kurs programowania mikrokontrolerów PIC (6)

Sterowanie kolorowym wyświetlaczem Nokii 3510i

Graficzne wyświetlacze z telefonów komórkowych są chętnie stosowane. Swoją popularność zawdzięczają głównie małemu poborowi prądu, niskiej cenie i dostępności. Oczywiście, nie w każdym urządzeniu można je zastosować. Z racji przeznaczenia ekrany te mają niewielkie wymiary, co może się okazać istotnym ograniczeniem. Stosując kolorowy wyświetlacz od telefonu, można zacząć zdobywać doświadczenia w sterowaniu tego typu elementów. W artykule opisano sposób sterowania modułem wyświetlacza z telefonu Nokia 3510i.



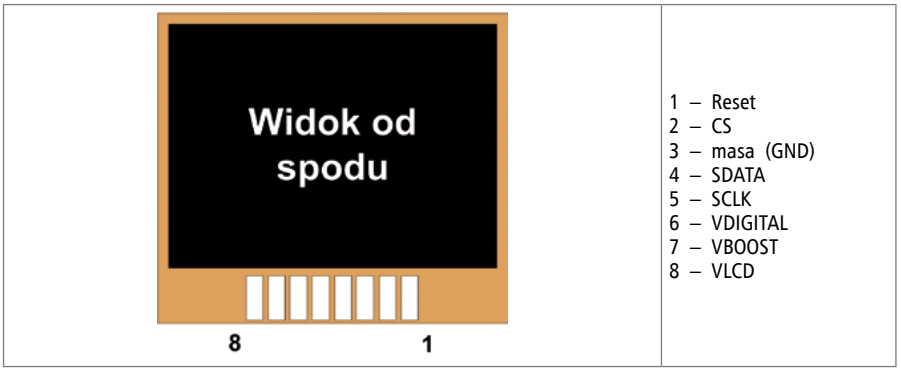
Kolorowa matryca ma rozdzielczość 96×65 pikseli, a informacja o kolorze może być zapisywana na 12 bitach (4096 kolorów) lub na 8 bitach (256 kolorów). Pole wyświetlacza ma wymiary 33×24 mm. Mimo że sam sterownik i matryca pobierają bardzo mało energii, to trzeba pamiętać, że wymagane jest zasilenie podświetlania za pomocą białych diod LED. Bez tego wyświetlania informacja może nie być dobrze widoczna.

Wyświetlacz Nokii 3510i

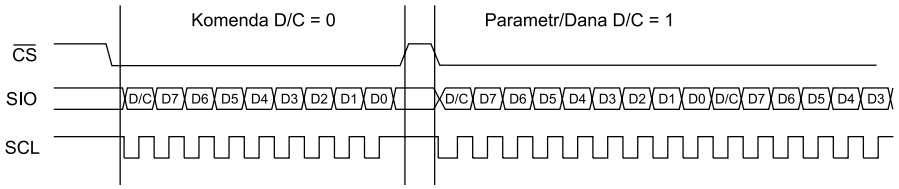
Kolorowa matryca LCD jest nadzorowana przez układ S1D15G14 produkowany przez firmę Epson. Wbudowane w strukturę przetwornika drivery sterują 84 liniami po 312 segmentów. Do komunikacji z mikrokontrolerem przewidziano 2 interfejsy: szeregowy i równoległy, który w tym wyświetlaczu jest niedostępny. Układ wyposażono w kompletny układ zasilania z programowaną przetwornicą zasilającą matrycę LCD oraz wewnętrzny oscylator RC.

Wyprowadzenia i interfejs. Wyprowadzenia wykorzystywane do zasilania i sterowania wyświetlaczem pokazano na **rysunku 1**. Napięcie VDIGITAL dołączane do wyprowadzenia 6 zasila obwody cyfrowe sterownika. Producent zaleca, aby miało ono wartość 1,8 V. Napięcie VBOOST zasila układ przetwornicy pojemnościowej. Jej napięcie wyjściowe zasila programowane źródło napięcia zasilającego matrycę LCD. Napięcie VBOOST powinno mieć wartość 2,8 V. Oba napięcia, VDIGITAL i VBOOST, mogą mieć maksymalną wartość 3,6 V. W trakcie prób z wyświetlaczem wyprowadzenia VBOOST i VDIGITAL były zasilane

Dodatkowe materiały na CD/FTP:
ftp://ep.com.pl, user: 15352, pass: 760hp6s5
• poprzednie części kursu



Rysunek 1. Rozmieszczenie wyprowadzeń wyświetlacza Nokii E3510i



Rysunek 2. Przebiegi czasowe podczas transmisji danych do modułu LCD

jednym napięciem +3,3 V. Do wyprowadzenia VLCD dołączono wyjście programowanego regulatora napięcia zasilającego matrycę LCD. VLCD trzeba zablokować do masy ceramicznym kondensatorem o pojemności 100 nF.

W wyświetlaczu Nokii 3510i na stałe dołączono interfejs szeregowy i nie ma możliwości użycia magistrali równoległej. Ten interfejs składa się z 3 linii:

– dwukierunkowej linii danych SDATA,
– linii zegarowej SCLK,
– linii wyboru interfejsu CS.

Dane mogą być przesyłane w trybie 8- lub 9-bitowym. Tryb 8-bitowy wymaga użycia dodatkowej linii A0 sygnalizującej rodzaj przesyłanych danych (dane lub komendy). W trybie 9-bitowym nie ma konieczności użycia dodatkowej linii, a informację o rodzaju przesyłanych danych zawarto w 9. bicie. Zaletą takiego rozwiązania jest brak dodatkowej linii sterującej, ale konieczność przesyłania 9 bitów uniemożliwia wykorzystanie sprzętowego wsparcia transmisji przez wbudowany moduł SPI. Przebiegi czasowe w czasie przesyłania danych do modułu LCD pokazano na **rysunku 2**.

Przesłanie danych rozpoczyna się od wyzerowania przez mikrokontroler linii dołączonej do wejścia CS. Jako pierwszy jest wysyłany bit *C/D*. Kiedy jest on ustawiony, przesyłane dane wpisywane są do pamięci RAM wyświetlacza. Wyzerowanie C/D oznacza przesyłanie komend. Dane są wpisywane w czasie narastającego zbocza zegara SCLK.

Zastosowany protokół transmisyjny umożliwia odczytywanie danych z pamięci RAM oraz rejestru statusowego. W opisywanych dalej przykładach obsługi dane będą tyko wysyłane do wyświetlacza. W większości zastosowań odczytywanie nie jest konieczne. Do sprzętowego restartowania sterownika służy linia RESET.

Organizacja pamięci obrazu. Układ scalony sterownika ma wbudowaną pamięć RAM obrazu. Wpisywanie danych do tej pamięci powoduje zmianę kolorów wyświetlanych pikseli. Każdy piksel wyświetlacza kolorowego jest zbudowany z 3 warstw – segmentów. Każdy z tych trzech segmentów odpowiada za wyświetlanie jednego z kolorów podstawowych: czerwonego R (red), zielonego G (green) i niebieskiego B (blue). Jasność każdego z segmentów jest sterowana przebiegiem PWM o wypełnieniu programowanym wartością zakodowana na 4 bitach. Informacja o kolorze piksela jest zakodowana łącznie na 12 bitach (3 segmenty po 4 bity). W jednej linii jest 312 segmentów, czyli do zapamiętania wszystkich informacji potrzeba 312×4 bity. Ponieważ wszystkich linii jest 84, całkowita pojemność pamięci musi wynosić 312×4×84=104832 bitów. Z praktycznego punktu widzenia jest wygodniej operować pikselami, a nie segmentami. 312 segmentów w linii tworzy 104 piksele (3 segmenty na piksel). Kolor każdego piksela jest zapisany na 12 bitach, więc organizacja pamięci jest następująca: 104 piksele×12 bitów×84 linie. Aby określić kolor piksela, trzeba pod odpowiadającą mu lokację w pamięci RAM wpisać 12-bitową liczbę.

Zapis pamięci RAM. Pamięć RAM wyświetlacza może być zapisywana w 2 trybach. Pierwszy z nich to tryb nazywany 444. Nazwa bierze się od tego, że każdy z kolorów podstawowych jest kodowany i przesyłany na 4 bitach. W ten sposób można zakodować łącznie 4096 kolorów. Tryb 444 jest naturalny dla wyświetlacza mogącego wyświetlić 4096 kolorów i może być niezastąpiony przy wyświetlaniu bitmap, ale jest kłopotliwy w stosowaniu.

Prostszy w obsłudze, ale też dający mniejsze możliwości, jest tryb nazywany 332. Informacja o kolorach R i G jest zakodowana na 3 bitach, a o kolorze B na 2 bitach. Razem daje to 8 bitów na piksel, tzn. 256 kolorów. Przesłanie 8 bitów danych z zakodowanym kolorem musi być w jakiś sposób uzupełnione do 12 bitów piksela w pamięci RAM. Do tego celu jest stosowana programowana tablica konwersji. Trzy bity R i G adresują po 8 lokacji w tej tablicy, a 2 bity B – 4 lokacje. Tablica zawiera słowa 4-bitowe określające kolor. Ten sposób adresowania pamięci RAM wyświetlacza zostanie wyjaśniony przy okazji omawiania wybranych komend sterownika. Kody komend mają długość 8 bitów. W tabeli (plik znajduje się na serwerze FTP) zamieszczono zestawienie wszystkich komend S1D15G14.

Kody komend są wpisywane z bitem D/C=0, a argumenty z bitem C/D=1. Dokładny opis wszystkich komend można znaleźć w dokumentacji sterownika, tu podam opis tylko kilku najważniejszych. Zaczniemy od komend adresowania pamięci: *Column address set* i *Page address set*.

Gdy dane są przesyłane z mikrokontrolera do pamięci wyświetlacza, komenda *Column address set* jest używana do ustawiania obszaru zapisywania. *Column address set* ma dwa parametry. Pierwszy z nich ustala początek obszaru (numer pierwszej kolumny), a drugi koniec wpisywania (numer ostatniej kolumny). Wpisywanie do obszaru określonego przez tę komendę jest uzależnione ustawieniem skanowania adresów komendą *Memory access control*.

Ustawienie za pomocą *Memory access control* skanowania kolumn powoduje, że w czasie wpisywania kolejnych bajtów do pamięci RAM adres jest inkrementowany od numeru kolumny początku do numeru kolumny końca obszaru. Kiedy adres osiągnie numer końca, jest inkrementowany numer strony (następna linia wyświetlacza) i zapisywanie rozpoczyna się od numeru początkowego kolumny w nowej linii. Jest to bardzo użyteczna właściwość pozwalająca w połączeniu z komendami *Memory access control* i *Page address set* na elastyczne definiowanie zakresu automatycznej zmiany adresów przy zapisywaniu bitmapy. Ta właściwość zostanie wykorzystana w procedurach wyświetlania znaków alfanumerycznych o różnych wielkościach czcionki.

Tabela 1. Znaczenie bitów słowa argumentu komendy Memory Access Control

Bit/ wartość	Opis
B7=0 B7=1	Linie są adresowane od góry do dołu wyświetlacza Linie są adresowane od dołu do góry wyświetlacza
B6=0 B6=1	Numer linii jest inkrementowany (od lewej do prawej) Numer linii jest dekrementowany (od prawej do lewej)
B5=0 B5=1	Adresowanie kolumn – tryb normalny. Adres kolumny po każdym wpisie danej zwiększa się o 1, a po osiągnięciu wartości maksymalnej (określonej za pomocą komendy Column address set), jest mu nadawana wartość początkowa zdefiniowana za pomocą Column address set Tryb inwersji. Po każdym wpisaniu danej adres wiersza jest inkrementowany. Kiedy osiągnie wartość maksymalną (określoną za pomocą komendy Page address set), licznik kolumn jest zwiększany o 1, a licznik wierszy przyjmuje wartość początkową określoną za pomocą komendy Page address set
B4=0 B4=1	Skanowanie obrazu od góry do dołu Skanowanie obrazu z dołu do góry
B3=0 B3=1	Sekwencja RGB wpisywania do pamięci RAM Sekwencja BGR wpisywania do pamięci RAM
B2=0	
B1=0	
B0=0 B0=1	Zmiana orientacji obrazu na poziomą (orient. domyślna) Zmiana orientacji obrazu na pionową

Komenda *Page address set* jest używana do określania obszaru pamięci przy wpisywaniu danych do pamięci wyświetlacza. *Page address set* ma dwa parametry. Pierwszy z nich ustawia początek obszaru (numer pierwszej linii), a drugi koniec wpisywania (numer ostatniej linii). Wpisywanie do obszaru określonego przez tę komendę jest uzależnione od ustawienia skanowania adresów komendą *Memory access control*. Ustawienie za jej pomocą skanowania linii (stron pamięci) powoduje, że w czasie wpisywania kolejnych bajtów do pamięci RAM adres jest inkrementowany od numeru kolumny linii do numeru linii końca obszaru. Kiedy adres osiągnie numer końca, jest inkrementowany numer kolumny i zapisywanie rozpoczyna się od numeru początkowego linii w nowej kolumnie.

Komenda *Memory access control* określa sposób dostęp do pamięci RAM. Jej parametry są zawarte w 1-bajtowym argumencie wpisywanym po kodzie komendy. Opis znaczenia poszczególnych bitów podano w **tabeli 1**.

Argument komendy *Interface pixel format* określa liczbę kolorów. Jeżeli jest równy 2, to kolor piksela jest kodowany na 8 bitach (256 kolorów), a jeżeli jest równy 3, to na 12 bitach (4096 kolorów). Pozostałe wartości argumentu nie są używane. Po zerowaniu domyślnie jest włączany tryb wyświetlania w 4096 kolorach.

Do inicjowania zapisywania obszaru pamięci obrazu jest używana komenda *Memory write*. Wszystkie dane po przesłaniu tej komendy będą wpisywane do pamięci RAM pod adresami ustalonymi za pomocą komend *Column address set* i *Page address set*. Zapisywanie jest przerywane po wysłaniu do sterownika dowolnej komendy.

Komenda *Display control* służy do sterowania wyświetlaczem i ma 7. bajtowy parametr, którego strukturę pokazano na **rysunku 3**. Wartość zapisana w pierwszym bajcie (P10…P17), w drugim bajcie (P20….P28) oraz na bicie P37 jest związana z trybami pracy driverów LCD. Działanie driverów i sposób programowania tych bitów jest opisany w dokumentacji sterownika na stronie 29, w rozdziale 8.6. W opisie praktycznego sterowania wyświetlaczem zostaną przedstawione wartości, które trzeba wpisać na te pozycje, żeby wyświetlacz pracował prawidłowo. Na bitach P35…P33 jest zapisana wartość określająca sterowanie układem polaryzacji wyświetlacza: 000=1/9, 001=1/8, 010=1/7, 011=1/6, 100=1/5. Bit P32 określa położenie sterownika względem panelu LCD. Zalecane jest wyzerowanie tego bitu. Bit P31 określa rozmiar obsługiwanego wyświetlacza. Jeżeli jest wyzerowany, to obsługiwane są wyświetlacze o rozdzielczościach 98×67 lub 98×84 piksele. Po ustawieniu tego bitu dostępne są rozdzielczości 104×67 lub 104×84 piksele. Cykl pracy wyświetlacza jest programowany nastawą bitu P30. Jeżeli ten bit jest wyzerowany, to cykl jest równy 1/82, a jeżeli jest ustawiony, to cykl jest równy 1/67.

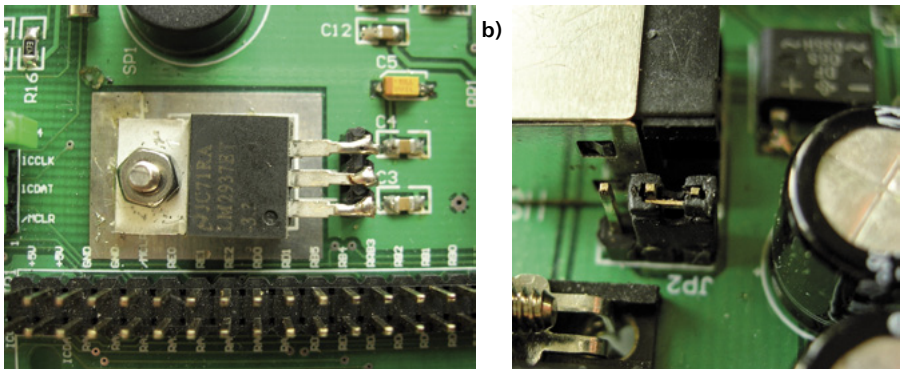
D7	D6	D5	D4	D3	D2	D1	D0
P17	P16	P15	P14	P13	P12	P11	P10
P27	P26	P25	P24	P23	P22	P21	P20
P37	*	P35	P34	P33	P32	P31	P30
*	P46	P45	P44	P43	P42	P41	P40
*	P56	P55	P54	P53	P52	P51	P50
*	P66	P65	P64	P63	P62	P61	P60
P77	P76	P75	P74	P73	P72	P71	P70

Rysunek 3. Struktura parametrów komendy *Display control*

D7	D6	D5	D4	D3	D2	D1	D0
*	P16	P15	P14	P13	P12	P11	P10
*	*	*	*	*	*	1	1

Rysunek 4. Struktura parametrów komendy *Voltage control*

a) b)



Fotografia 5. Zmiana zasilania na +3,3 V: a) wymiana stabilizatora na LDO +3,3 V, b) ustawienie zworki JP2

Po zerowaniu sterownika rejestry komendy mają wartości nieustalone i trzeba je zaprogramować w trakcie inicjalizacji wyświetlacza.

Sterownik jest wyposażony w układ pojemnościowej przetwornicy napięcia i programowany komendą *Voltage control* układ regulacji napięcia zasilającego matrycę LCD. Parametrami komendy są 2 bajty – o strukturze pokazanej na **rysunku 4**. Bity P16….P10 określają napięcie wyjściowe według zasady dokładnie opisanej w dokumentacji sterownika.

Zachęcam do uważniejszego przestudiowania dokumentacji sterownika S1D15G14 i zapoznania się ze wszystkimi komendami oraz sposobami definiowania kolorów w obu dostępnych trybach wyświetlania.

Dołączenie wyświetlacza do płytki ewaluacyjnej

Aby wyświetlacz pracował prawidłowo, należy obniżyć napięcie zasilania. Standardowo, wszystkie układy na płytce ewaluacyjnej są zasilane napięciem +5 V ze stabilizatora 7805, natomiast wyświetlacz nie może być zasilony napięciem wyższym niż +3,6 V. Z tego powodu musimy obniżyć wartość napięcia zasilania do +3,3 V. Najłatwiej można to zrobić przez wymianę stabilizatora 7805 na np. LM2937-3,3V lub podobny (**fotografia 5a**). Po tej wymianie płytki przestawiamy zworkę JP2 (**fotografia 5b**) i zasilamy płytkę z zewnętrznego zasilacza przez złącze P2. Nie możemy zasilić układu przez złącze USB nawet po wymianie stabilizatora, bo w tym wypadku układ będzie zasilany bezpośrednio napięciem +5 V z USB bez pośrednictwa stabilizatora na płytce. Po wykonaniu tych czynności i sprawdzeniu napięcia +3,3 V można przystąpić do dołączenia wyświetla-

Listing 1. Definicje linii sterujących

```
#define RES 0x40;
#define CS 0x20;
#define DATA 1
#define CLK 2
```

Listing 2. Przesłanie 9-bitowej danej przez SPI

```
void SendLcd(unsigned char data, unsigned char cmd){
  unsigned char i;
  LATC&=~CS;//CS=0;
  asm(„nop”);
  LATC&=~CLK;
//pierwszy bit określa czy przesyłane są dane czy komendy
  if(cmd==Cmd) LATC&=~DATA;//DATA=0
  else LATC|=DATA;//DATA=1
  asm(„nop”);
  LATC|=CLK;//CLK=1
  for (i=0;i<8;i++) {
    LATC&=~CLK;//CLK=0
    if ((data&0x80)==0) LATC&=~DATA;// DATA=0;
    else LATC|=DATA;//DATA=1
    asm(„nop”);
    LATC|=CLK;//CLK=1
    data<<=1;
  }
  LATC|=CS;//CS=1
}
```

cza. Najłatwiej jest przylutować do kontaktów modułu wyświetlacza cienkie elastyczne przewody. Ten sposób, mimo iż może dać zadawalające efekty, niestety bywa zawodny. Ja wykorzystałem do tego celu płytkę z projektu timera fotograficznego (opisywany w EP 6/2011) przystosowaną do podłączenia takiego właśnie wyświetlacza. Na płytce są umieszczone kontakty, więc po przyłożeniu do niej wyświetlacza i przykręceniu wkrętami ustalającymi powstaje pewny kontakt.

Interfejs do wyświetlacza musimy zaimplementować programowo z powodów podanych wcześniej. Najpierw musimy zdefiniować 4 linie sterujące: danych (DATA), zegarową (CLK), wyboru (CS) i zerowania (RES) (**listing 1**). Definicje te mogą się wydawać na pierwszy rzut oka trochę dziwne. W rzeczywistości są to wagi bitów: 0, 1, 4 i 5. Do obsługi interfejsu szeregowego użyjemy portu PORTC. Ustawienie bitu CS będzie wykonywane jako *LATC|=CS;*. Wówczas w rejestrze LATC zostanie ustawiony bit 4 przez wykonanie operacji *LATC|=0b0010000*. Natomiast wyzerowanie bitu CS jest równie łatwe – *LATC&=~CS;*. W tym wypadku na rejestrze LATC zostanie wykonana operacja *LATC&=0b1101 111* i bit 4 rejestru LATC zostanie wyzerowany. Podobnie nadaje się wartości pozostałym bitom.

Na **listingu 2** zamieszczono procedurę *SendLcd* wysyłająca 9-bitową daną przez interfejs szeregowy. Procedura ma 2 argumenty: 8-bitową daną *data* i argument *cmd* określający wartość bitu C/D. Najpierw jest wysyłany bit C/D, a potem 8-bitowe słowo, od bitu najstarszego do najmłodszego.

SendLcd jest używana do inicjalizacji sterownika. Inicjalizacja jest niezbędna, bo po włączeniu zasilania w istotnych rejestrach sterownika są wartości nieustalone. Żeby wyświetlacz mógł pracować, trzeba zainicjować sterownik przez wpisanie wartości do rejestrów sterujących pracą wyświetlacza. Wszystkie parametry inicjalizacji zostały tak dobrane, aby wyświetlacz pracował prawidłowo. Z szeregiem parametrów można eksperymentować. Jako przykłady mogą posłużyć komendy *Gray scale position* lub *Colour set*.

Inicjalizacja wyświetlacza pokazana na **listingu 3** jest podzielona na kilka etapów: zerowanie sterownika, ustawienie parametrów sterownika, ustawienie układu zasilania matrycy i tablicy kolorów dla trybu 8-bitowego i końcowe ustawienie parametrów wyświetlacza. Bardziej dociekliwi Czytelnicy zapewne spróbują prześledzić działanie komend inicjalizacji.

Sterownik jest zerowany przez podanie trwającego kilka milisekund sygnału o poziomie niskim na doprowadzenie *Reset*. Linie *CS*, *SDATA* i *SCLK* są wtedy na poziomie wysokim. Po zakończeniu zerowania sprzętowego jest wysyłana komenda *Software reset*. Sterownik wychodzi z fazy zerowania po wykonaniu komendy *Initial escape*.

Listing 3. Inicjalizacja sterownika

```
void InicLcd(){
   //zerowanie
    LATC|=CS;
    LATC|=CLK;
    LATC|=DATA;
    LATC&=~RES;//RES=0
    Delay();
    LATC|=RES; //RES=1
    Delay();
    SendLcd(1, Cmd);
    Delay();
    SendLcd(0xc6, Cmd);
    Delay();
// display setup 1
    SendLcd(0xB9, Cmd);         // Refresh set
    SendLcd(0x00, Data);
    SendLcd(0xB6, Cmd);         // Display Control
    SendLcd(128, Data);
    SendLcd(128, Data);         //no N line inversion
    SendLcd(0x81, Data);         //frame freq; bias rate of LCD drive voltage;
98x67; diplay duty=1/67;
    SendLcd(84, Data);
    SendLcd(69, Data);
    SendLcd(82, Data);
    SendLcd(67, Data);
    SendLcd(0xB3, Cmd);         // Gray Scale Position
    SendLcd(0x11, Data);
    SendLcd(0x22, Data);
    SendLcd(0x33, Data);
    SendLcd(0x44, Data);
    SendLcd(0x55, Data);
    SendLcd(0x66, Data);
    SendLcd(0x77, Data);
    SendLcd(0x88, Data);
    SendLcd(0x99, Data);
    SendLcd(0xAA, Data);
    SendLcd(0xBB, Data);
    SendLcd(0xCC, Data);
    SendLcd(0xDD, Data);
    SendLcd(0xEE, Data);
    SendLcd(0xFF, Data);
    SendLcd(0xB5, Cmd);         // Gamma Curve Set
    SendLcd(3, Data);
    SendLcd(0xBD, Cmd);         // Common driver setup
    SendLcd(3, Data);
// power supply
    SendLcd(0xBE, Cmd);         // Power Control
    SendLcd(1, Data);
    SendLcd(0x11, Cmd);         // Sleep out
    SendLcd(0xBA, Cmd);         // Voltage control
    SendLcd(63, Data);
    SendLcd(3, Data);
    SendLcd(0x25, Cmd);         // Write contrast
    SendLcd(70, Data);              //69
    SendLcd(0x2D, Cmd);         // Colour set (only for 8-bit/pixel)
    SendLcd(0x00, Data);        // RED
    SendLcd(0x02, Data);
    SendLcd(0x04, Data);
    SendLcd(0x06, Data);
    SendLcd(0x09, Data);
    SendLcd(0x0b, Data);
    SendLcd(0x0d, Data);
    SendLcd(0x0f, Data);
    SendLcd(0x00, Data);        // GREEN
    SendLcd(0x02, Data);
    SendLcd(0x04, Data);
    SendLcd(0x06, Data);
    SendLcd(0x09, Data);
    SendLcd(0x0b, Data);
    SendLcd(0x0d, Data);
    SendLcd(0x0f, Data);
    SendLcd(0x00, Data);        // BLUE
    SendLcd(0x05, Data);
    SendLcd(0x0a, Data);
    SendLcd(0x0f, Data);
    SendLcd(0xB7, Cmd);         // Temperature gradient set
    SendLcd(0, Data);
    SendLcd(0, Data);
    SendLcd(0, Data);
    SendLcd(0, Data);
    SendLcd(0, Data);
    SendLcd(0, Data);
    SendLcd(0, Data);
    SendLcd(0, Data);
    SendLcd(0, Data);
    SendLcd(0, Data);
    SendLcd(0, Data);
    SendLcd(0, Data);
    SendLcd(0, Data);
    SendLcd(0, Data);
    SendLcd(0, Data);
    SendLcd(0, Data);
    SendLcd(0, Data);
    SendLcd(0, Data);
    SendLcd(0x03, Cmd);         // Booster Voltage ON
    Delay();
    Delay();
    Delay();
    Delay();
// display setup 2
    SendLcd(0x21, Cmd);         // Inversion control (OFF)
//display setup 3
    SendLcd(0x3A, Cmd);         // 8bpp
    SendLcd(0x03, Data);
    SendLcd(0x29, Cmd);         // Display On
}
```

Etap ustawiania parametrów wyświetlacza rozpoczyna komenda *Refresh set*. Parametr tej komendy określa częstotliwość odświeżania. Komenda *Display control* ustala „*No N line inversion*”, częstotliwość ramek, poziom polaryzacji, cykl pracy driverow i wielkość matrycy wyświetlacza. Również rejestry sterujące skalą szarości pikseli mają wartości nieustalone i muszą być zaprogramowane komendą *Gray scale position*, a zaprogramowany zestaw rejestrów musi być wybrany komendą *Gamma curve set*. Ostatnia komenda tego etapu *Common driver setup* określa, w jakiej kolejności będą wyświetlane linie wyświetlacza.

Wbudowany w sterownik układ zasilania matrycy również wymaga zaprogramowania. W parametrze komendy *Power control* są umieszczone informacje o częstotliwości pracy wbudowanej przetwornicy (26,3 kHz). Do układu regulatora napięcia wyjściowego może być dołączony zewnętrzny lub wewnętrzny rezystor określający zakres regulacji (tutaj wewnętrzny).

Komenda *Sleep out* wyprowadza sterownik ze stanu uśpienia, a komenda *Voltage control* reguluje napięcie wyjściowe zasilające układ regulacji napięcia matrycy LCD. Komenda *Write contrast* reguluje napięcie zasilające matrycę LCD. Na tym etapie inicjalizacji jest też zapisywana tablica konwersji kolorów z trybu 8- do 12-bitowego i współczynniki temperaturowe. Po zaprogramowaniu parametrów układu zasilania włącza się go komendą *Booster voltage on* (po włączeniu zasilania jest domyślnie wyłączony). Na zakończenie pozostaje już tylko włączenie trybu 8- lub 12-bitowego koloru (komenda *Interface pixel control*), wyłączenie trybu inwersji i włączenie wyświetlania.

Po prawidłowym zainicjowaniu wyświetlacza na ekranie pojawi się kolorowa mozaika pikseli odzwierciedlająca przypadkowe wartości pamięci RAM po włączeniu zasilania. Dlatego proces inicjalizacji trzeba zakończyć procedurą wpisującą do pamięci RAM wyświetlacza wartość wyświetlania jednego koloru. Dla 12-bitowej głębi kolorów to zadanie jest realizowane przez procedurę *ClsLcd* pokazaną na **listingu 4**.

Jako pierwsza jest wysyłana komenda *Memory access control* z parametrem 0x20. Po wykonaniu tej komendy wykonywany jest tryb adresowania *Invert mode*. Każde kolejne wpisanie bajtu do pamięci RAM wyświetlacza powoduje zwiększenie licznika wierszy. Komenda *Column address set* ustala zakres zmian numerów kolumn od kolumny początkowej o numerze 0 do kolumny końcowej o numerze 97. Podobnie komenda *Page address* ustala zakres numerów linii (stron pamięci) od 0 do 67. Przy takim zdefiniowaniu obszarów zapisywania zostanie zapisana cała pamięć RAM wyświetlacza. Zapisywanie rozpoczyna się komendą *Memory write*, a kończy po wysłaniu komendy *Memory access control*. Argumentem *ClsLcd* jest 16-bitowa stała określająca 12-bitową składową koloru, którą należy wypełnić pamięć obrazu.

Kontroler wyświetlacza nie ma wbudowanego generatora znaków i dlatego trzeba wzorce znaków stworzyć we własnym zakresie. W kolorowych wyświetlaczach graficznych sprawa jest nieco bardziej skomplikowana. Po pierwsze, każdemu pikselowi odpowiada jeden bajt dla trybu 8-bitowego lub 1,5 bajta dla trybu 12-bitowego. Po drugie, dla każdego ze znaków trzeba zdefiniować kolor czcionki i kolor tła znaku. Jednak to, że każdy piksel jest zapisany przynajmniej na 1 bajcie, daje możliwość definiowania znaków o niemal dowolnej wielkości. Poza tym przy ich wyświetlaniu bardzo pomagają komendy *Column address set* i *Page address set*. Po ustaleniu zakresów zmian adresów stron i kolumn wystarczy tylko wpisywać dane z tablicy generatora znaków.

Przygotowanie tablicy z generatorem znaków jest bardzo żmudnym zadaniem. Ja skorzystałem z dostępnej w Internecie tablicy przygotowanej przez D. Lyncha. Tablica składa się z 3 części: znaków małych o wymiarach 6×8 pikseli, znaków średnich 8×8 pikseli i znaków dużych 16×8 pikseli. Procedura wyświetlania *LCDPutChar* znaku jest również autorstwa D. Lyncha. Jej argument *c* zawiera kod wyświetlanego znaku. Jest to kod ASCII, ale tablica wzorców zaczyna się od znaku SPACJA (kod ASCII =0x20). Dlatego LCDPutChar wykonuje konwersję, odejmując od kodu zawartego w argumencie *c* wartość 0x1F: pChar = pFont + (nBytes * (c – 0x1F)) + nBytes – 1;. Pozostałe argumenty określają pozycję początku tekstu na ekranie (współrzędne *x* i *y*) oraz kolor znaku *fColor* i kolor tła *bColor*.

Listing 4. Inicjalizacja pamięci obrazu

```
//czyszczenie pamieci wyświetlacza - kolor 12-bitowy
void ClsLcd(short color) {
int i; // loop counter
        //SendLcd(0x36, Cmd);         // x,y=0
        //SendLcd(0x20, Data);        // Page direction in writing
// ustawienie zkresu liczników stron pamieci
        SendLcd(0x2b,Cmd);
        SendLcd(0,Data);
        SendLcd(67,Data);
//ustawienie zkresów liczników kolumn
        SendLcd(0x2a,Cmd);
        SendLcd(0,Data);
        SendLcd(97,Data);
// wpisanie do pamieci
        SendLcd(0x2c,Cmd);  //komenda Memory Write
for(i = 0; i < ((67 * 98) / 2); i++) {
SendLcd((color >> 4) & 0xFF,Data);
SendLcd(((color & 0xF) << 4) | ((color >> 8) & 0xF),Data);
SendLcd(color & 0xFF,Data);
}
        SendLcd(0x36, Cmd);         // Memory Access Control
        SendLcd(0x80, Data);        // Page direction in writing
        SendLcd(0xBD, Cmd);         // x,y=0
        SendLcd(4, Data);        // Page direction in writing
}
```

Listing 5. Procedura wyświetlająca napisy

```
void LCDPutStr(const char *pString, int x, int y, int Size, int fColor, int
bColor) {
//w pętli do znalezienia końca ciągu
  while (*pString != 0x00) {
//wyświetl znak alfanumeryczny
  LCDPutChar(*pString++, x, y, Size, fColor, bColor);
//korekcja pozycji y w zależności od wielkości znaku
  if (Size == SMALL)
  y = y + 6;
  else if (Size == MEDIUM)
  y = y + 8;
  else y = y + 8;
    // czy znak się zmieści
    if (y > 97) break;
  }
}
```

Listing 6. Testowanie trybu tekstowego.

```
    InicLcd();
    ClsLcd(WHITE);
    LCDPutStr(„   Duza      „, 50, 0, LARGE,RED,YELLOW);
    LCDPutStr(„  Srednia     „, 30, 0, MEDIUM,BLUE,GREEN);
    LCDPutStr(„  Mala czcionka  „, 10, 0, SMALL,WHITE,BLACK);
```

Listing 7. Wyświetlanie bitmapy

```
extern const unsigned short pic_bmp[];
void BmpLcd(void) {
int j,k,w0,w1; // loop counter
// ustawienie zkresu liczników stron pamieci
  k=0;
  SendLcd(0x2b,Cmd);
  SendLcd(0,Data);
  SendLcd(67,Data);
//ustawienie zakresów liczników kolumn
  SendLcd(0x2a,Cmd);
  SendLcd(0,Data);
  SendLcd(95,Data);
// wpisanie do pamieci
  SendLcd(0x2c,Cmd);
  for(j = 0; j < ((67 * 95) / 2); j++) {
    w0=pic_bmp[k++];
    w1=pic_bmp[k++];
    SendLcd((w0 >> 4) & 0xFF, Data);  //konwersja na tryb 12-bitowy
    SendLcd(((w0 & 0xF) << 4) | ((w1 >> 8) & 0xF), Data);
    SendLcd(w1 & 0xFF, Data);
  }
  SendLcd(0x36, Cmd);         // x,y=0
  SendLcd(0x80, Data);        // Page direction in writing
  SendLcd(0xBD, Cmd);         // x,y=0
  SendLcd(4, Data);        // Page direction in writing
}
```

Mając procedurę wyświetlania jednego znaku, bez problemu napiszemy procedurę wyświetlającą napisy na ekranie (**listing 5**). Jej argumentami są: wskaźnik do obszaru pamięci zawierającego początek łańcucha z kodami ASCII, współrzędne początku wyświetlania, wielkość znaku oraz kolor znaku i kolor tła. W zależności od wielkości znaku jest przeprowadzana korekcja współrzędnej *y* (kolumny) następnego znaku. Fragment programu wyświetlający testowy tekst pokazano na listingu 6, a wynik jego działania na **fotografii 6**.

Wyświetlanie kolorowych bitmap jest elementem, który potrafi znacznie uatrakcyjnić projektowany interfejs użytkownika. Trzeba jednak pamiętać o tym, że istotnym ograniczeniem jest pojemność pamięci, w której jest zapisywana bitmapa. Tak jak w wypadku wyświetlacza monochromatycznego, trzeba kolorową bitmapę przekonwertować do tablicy w języku C. W „Elektronice Praktycznej” opisywałem już sposób konwersji kolorowych bitmap za pomocą programu *bmp2c* autorstwa Sebestiena Riou. W wyniku konwersji kolorowej bitmapy na tryb 12-bitowy na dwu słowach 16-bitowych w tablicy bitmapy są zapisywane informacje o kolorze 2 kolejnych pikseli. Procedura wyświetlająca bitmapę musi pobierać po 2 słowa i zapisywać je do pamięci obrazu. Na **listingu 6** zamieszczono procedurę wyświetlającą wcześniej przygotowaną i przekonwertowaną bitmapę, a na **fotografii 7** jej wynik działania.

Fotografia 6. Wyświetlacz w trybie tekstowym

Fotografia 7. Wyświetlanie bitmapy w trybie graficznym

Podsumowanie

Monochromatyczne wyświetlacze graficzne nie są trudne w obsłudze, a bitmapy nie zajmują dużo miejsca. Są oferowane w różnych rozdzielczościach z wbudowanym podświetleniem i z przetwornicami zasilającymi drivery matrycy LCD. Jeżeli dodamy do tego umiarkowana cenę, to jest to element, który może z powodzeniem konkurować z klasycznymi wyświetlaczami alfanumerycznymi ze sterownikiem HD44780.

Z kolorowymi wyświetlaczami graficznymi jest trudniej. Duże matryce są jeszcze stosunkowo drogie, ale można stosować tanie i dostępne wyświetlacze od telefonów komórkowych. Pewnym problemem może być dostępność informacji o typie sterownika i wyprowadzeniach, a także bardziej skomplikowana obsługa. Jednak w wielu wypadkach potrzebne informacje są dostępne, a kolorowy interfejs może wynagrodzić większy wysiłek przy jego projektowaniu.

Tomasz Jabłoński, EP