Kurs programowania mikrokontrolerów PIC (5)

Obsługa graficznych wyświetlaczy LCD

Graficzne wyświetlacze LCD powoli stają się jednym z zasadniczych elementów interfejsu użytkownika w systemach mikroprocesorowych. Ich popularność do niedawna była ograniczana dość wysoką ceną, ale aktualnie popularne wyświetlacze monochromatyczne można nabyć za kilkadziesiąt złotych. W tym odcinku kursu pokażemy, w jaki sposób poradzić sobie z wyświetlaniem obrazu – bitmapy za pomocą mikrokontrolera PIC.

Dodatkowe materiały na CD/FTP:
ftp://ep.com.pl, user: 19623, pass: 6c5r20n3
- poprzednie części kursu

Z moich doświadczeń wynika, że nawet 8-bitowy mikrokontroler bez jakiejś olbrzymiej pamięci programu może bez problemu obsłużyć monochromatyczny, a po spełnieniu pewnych warunków, kolorowy wyświetlacz LCD. Może się to wydać paradoksem, ale najczęściej stosuję takie wyświetlacze graficzne do wyświetlania tekstu, ponieważ mieści się go na takim ekranie dużo więcej niż na standardowym, alfanumerycznym wyświetlaczu o rozdzielczości 2×16 lub 2×20 znaków. Poza tym można stosować czcionki różnej wielkości, co ułatwia wyróżnianie pewnych fragmentów tekstu. Nie bez znaczenia jest też możliwość „upiększania" napisów czy to za pomocą jakiegoś wskaźnika, czy też rysowania wokół niego ramki.

Wyświetlanie obrazów w postaci bitmap wymaga sporych zasobów (pamięci) do ich zapisania, ale w wypadku wyświetlaczy monochromatycznych nie są to ogromne pliki, bo jeden piksel jest odwzorowywany przez jeden bit pamięci obrazu. Dużo gorzej z wyświetlaniem bitmap jest w przypadku wyświetlaczy kolorowych, bo w trybie 8-bitowym pojedynczy piksel jest reprezentowany przez 1 bajt lub w innych trybach przez 1,5 czy 2 bajty.

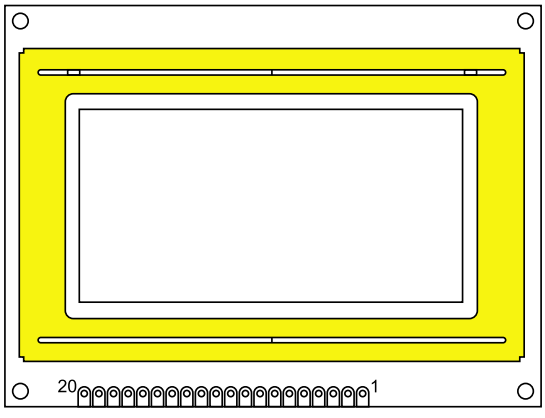
Wyświetlacz monochromatyczny HY12864

Moduł ewaluacyjny AVT5275 został wyposażony w złącze umożliwiające dołączenie monochromatycznego wyświetlacza HY12864 z matrycą LCD o rozdzielczości 128×64 piksele. Steruje nią popularny i dobrze udokumentowany sterownik KS0108, a właściwie 2 takie sterowniki. Dwa, ponieważ KS0108 potrafi sterować tylko matrycą 64×64 piksele i dlatego jeden z układów jest przeznaczony dla lewej połowy matrycy, a drugi dla prawej.

Rozmieszczenie kontaktów obudowy i funkcje wyprowadzeń wyświetlacza HY12864 pokazano na **rysunku 1**. Sterownik KS0108 (**rysunek 2**) ma w swojej strukturze wbudowany równoległy interfejs sterujący, pamięć RAM o pojemności 512 bajtów (4096 bitów) i drivery wyświetlacza LCD o rozdzielczości 64×64 piksele. Układy logiczne są zasilane napięciem VDD=+5 V, a drivery napięciem ujemnym o wartości od –8 V do –17 V mierzonym w stosunku do VDD.

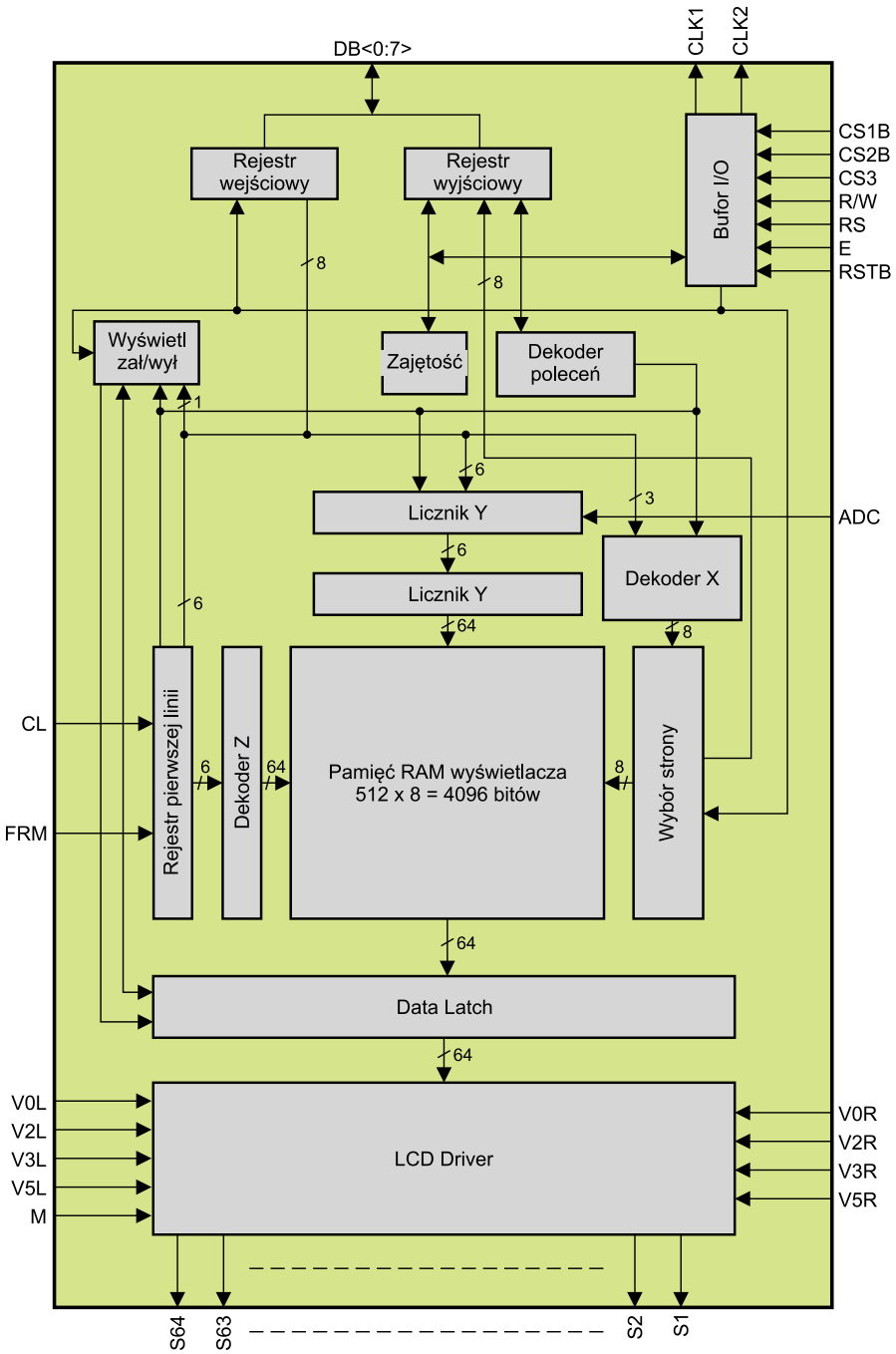
Pamięć obrazu RAM jest adresowana licznikami kolumn i stron. Licznik kolumn zmienia się od 0 do 63, a licznik stron od 0 do 7. Zapisanie jednego bajtu równego 0xFF do pamięci przy wyzerowanych licznikach adresowych powoduje wyświetlenie pionowego paska o długości 8 pikseli w lewym górnym rogu wyświetlacza. Zapisywanie kolejnych 64 bajtów utworzy poziomą linijkę o szerokości 8 pikseli i długości 64 pikseli. W taki sposób jest zapisywana i wyświetlana jedna strona pamięci.

Wyświetlacze z matrycą LCD wymagają do zasilania driverów ujemnego napięcia. Jesteśmy w dobrej sytuacji, bo panel wyświetlacza ma wbudowaną przetwornicę ujemnego napięcia dostępnego na styku numer 18 (VEE). Napięcie VEE podzielone przez potencjometr jest podawane na wyprowadzenie V0 (styk 3). Wartością napięcia na V0 reguluje się kontrast wyświetlacza.



Wyprowadzenie	Symbol	opis
1	VSS	masa
2	VDD	Zasilanie +5 V
3	V0	Kontrast (zasilanie driverów LCD)
4	RS	Wybór rejestrów „H" – dane „L" – instrukcje
5	RW	Wybór „H" – odczyt „L" – zapis
6	E	Wejście zezwolenia
7-14	DB0-DB7	Linie danych
15	CS1	Wybór sterownika lewej strony panelu
16	CS2	Wybór sterownika prawej strony panelu
17	RST	Sygnał zerowania
18	VEE	Wyjście napięcia ujemnego
19	LEDA	Podświetlenie +
20	LEDK	Podświetlenie –

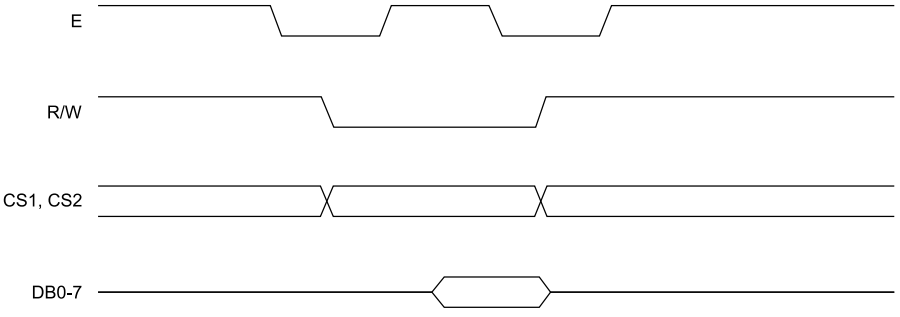
Rysunek 1. Widok i funkcje wyprowadzeń wyświetlacza HY12864



Rysunek 2. Schemat blokowy sterownika KS0108

Sterownik KS0108 komunikuje się z systemem mikroprocesorowym poprzez równoległą magistralę standardu Motoroli zbudowaną z:

- Linii zezwolenia E. Zbocze narastające na tej linii powoduje zapisanie lub odczytanie danych na magistrali danych.
- Linii zapisu lub odczytu R/W. Poziom wysoki na linii oznacza operację odczytywania, a niski zapisywania.
- Linii wybrania układu *Chip Select* (CS). Ponieważ wyświetlacz ma 2 sterowniki dla lewej i prawej połowy, są dwa sygnały CS1 i CS2.



Rysunek 3. Sekwencja zapisu danych do sterownika KS0108

- 8 dwukierunkowych linii danych D0...D7.

Poziom logiczny dodatkowej linii RS określa, gdzie ma być zapisana dana w sterowniku. Jeżeli RS=0, to dane są zapisywane w rejestrze komend lub jest odczytywany rejestr statusu. Pamięć RAM wyświetlacza jest zapisywana lub odczytywana przy RS=1.

Na **rysunku 3** pokazano sekwencje zapisu danych do sterownika KS0108. Na początku są zerowane sygnały E oraz R/W. Musi też zostać wybrany jeden ze sterowników wyświetlacza przez ustawienie sygnału CS1 lub CS2. W kolejnym kroku zewnętrzny mikrokontroler wystawia 8-bitową daną na magistrali i sygnał E zostaje ustawiony. Dane są wpisywane do sterownika opadającym zboczem sygnału E.

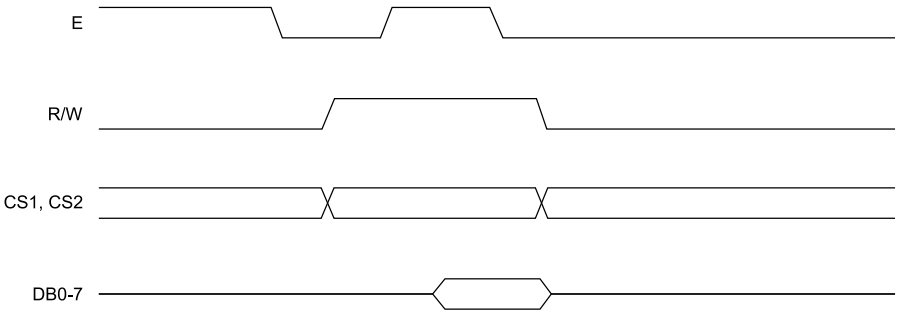
Sekwencja odczytywania danych (**rysunek 4**) ze sterownika KS0108 rozpoczyna się od wyzerowania sygnału E, a następnie ustawienia sygnału R/W i z linii CS1 lub CS2. Potem sygnał E zostaje ustawiony. Opadające zbocze sygnału E powoduje, że sterownik wyświetlacza wystawia dane na magistralę. Sekwencja odczytywania danych kończy się wyzerowanie sygnału CS1 lub CS2.

Sygnały magistrali sterującej modułu wyświetlacza LCD są wyprowadzone na złącze J15 (LCD Graf). Niestety, z powodu drobnych błędów na starszej wersji płytki, aby wszystko połączyć prawidłowo, trzeba będzie ciąć ścieżki i przylutować mostki z przewodów. Na nowszej wystarczy przepiąć zworki.

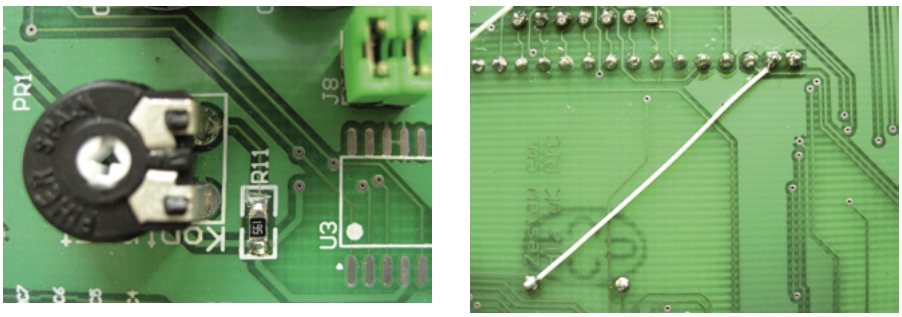
Pierwsza operacja zmiany połączeń będzie polegała na odcięciu zasilania +5 V od potencjometru PR1 (na nowej wersji płytki wystarczy przełożyć zworę w położenie 3–4). Ja przeciąłem ścieżkę na górnej warstwie prowadzącej od R11 do wyprowadzenia potencjometru (**fotografia 5a**). Teraz to wolne wyprowadzenie trzeba połączyć przewodem z wyprowadzeniem 18 wyświetlacza (ujemne napięcie VEE). Po tych operacjach na wyprowadzenie 3 wyświetlacza (V0) podawane jest regulowane napięcie ujemne z suwaka potencjometru PR1 i można regulować kontrast. Kolejny błąd na płytce to brak napięcia zasilającego +5 V na wyprowadzeniu 2 (**fotografia 5b**). Trzeba je połączyć kawałkiem przewodu (niepotrzebne na nowej płytce). Trzecia poprawka powinna odciąć wyprowadzenie 17 (RESET) od linii zerowania mikrokontrolera (również na nowej płytce, ewentualnie w czasie programowania można wyjmować wyświetlacz ze złącza). To połączenie powoduje, że w czasie programowania ISP na to wyprowadzenie jest podawane napięcie +12 V, które może uszkodzić wyświetlacz. Wyprowadzenie to trzeba będzie w jakiś sposób połączyć z wybraną linią portu mikrokontrolera.

Poprawki nie są trudne do wykonania, ale trzeba pamiętać, że kiedy zechcemy testować ponownie wyświetlacz alfanumeryczny LCD, to połączenia potencjometru PR1 powinny być przywrócone do pierwotnej postaci.

Po wykonaniu poprawek można połączyć linie równoległego interfejsu wyświetlacza z portami mikrokontrolera zgodnie z opisem w **tabeli 1**.



Rysunek 4. Sekwencja odczytu danych ze sterownika KS0108



Fotografia 5. Poprawki do wykonania na płytce: a) przecięcie ścieżki z napięciem +5 V do potencjometru kontrastu, b) dołączenie napięcia +5 V do pinu VDD

Definicję linii sterujących wyświetlaczem zamieszczono na **listingu 1**.

Opis sterowania wyświetlaczem rozpoczniemy od procedur przesyłania danych magistralą równoległą. Na **listingu 2** pokazano procedurę zapisu wartości do wyświetlacza *WriteLcd*. Argument *reg* określa, czy dane będą zapisywane do rejestru sterującego, czy do pamięci obrazu. Jeżeli reg=0, to linia RS jest zerowana i dane trafiają do rejestru sterującego. Gdy reg=1, to powoduje ustawienie linii RS i dane są zapisywane do wcześniej zaadresowanej pamięci obrazu.

W argumencie *dana* jest umieszczona wartość przeznaczona do przesłania. Ostatni argument *ctrl* wybiera jeden z dwóch sterowników KS0108 panelu wyświetlacza. Jeżeli ctrl=0, to linia CS1 jest ustawiana i zostaje wybrany lewy sterownik. Dla ctrl=1 CS=1 i jest wybierany prawy sterownik. Na końcu procedury obie linie wyboru sterowników są zerowane.

Każde zapisanie danej do jednego z rejestrów powoduje, że sterownik wyświetlacza jest przez jakiś czas zajęty. W czasie przetwarzania danych mikrokontroler host może tylko odczytywać rejestr statusowy, w którym jest zawarta między innymi informacja o zajętości (najstarszy bit rejestru – flaga *BUSY*). Dopiero kiedy flaga *BUSY* jest wyzerowana, to staje się możliwe zapisywanie lub odczytywanie danych przez magistralę. Sposób Odczytywania danych z rejestru statusowego wyświetlacza pokazano na **listingu 3**.

Podobnie jak w poprzedniej procedurze, argument *ctrl* wskazuje sterownik, którego rejestr będzie odczytywany. Przed odczytaniem danych z magistrali do rejestru *TRISC* trzeba wpisać same 1, tak aby port PORTC był portem wejściowym. Procedura zwraca wartość odczytaną z portu i zapisaną do zmiennej *data*. Funkcję z listingu 3 można łatwo zmodyfikować do odczytywania zawartości pamięci obrazu RAM sterownika. Wystarczy zamiast RS=0 użyć makra RS=1.

Sterowanie pracą wyświetlacza odbywa się przez wysyłanie do jego sterowników komend i zapisywania wyświetlanych danych do pamięci RAM. Zestawienie wszystkich komend akceptowanych przez KS0108 zamieszczono w **tabeli 2**.

Pierwsza z komend włącza i wyłącza wyświetlanie, ale nie modyfikuje w żaden sposób zawartości pamięci obrazu RAM sterownika. Po włączeniu zasilania wyświetlacz jest wyłączony i trzeba go włączyć komendą *Display ON*. Można to zrobić po wyzerowaniu zwartości pamięci RAM, bo po włączeniu zasilania są tam wartości przypadkowe (sterownik nie zeruje pamięci po włączeniu zasilania).

Na **listingu 4** pokazano procedurę *InicLcd* inicjalizująca, linie sterujące interfejsu równoległego i zerującą sterownik wyświetlacza. Ponadto, wysyłane są komendy włączenia wyświetlania i ustawienia od pierwszej linii (komenda *Display Start Line*). Zerowanie pamięci obrazu obu sterowników KS0108 wykonuje procedura ClrLcd z **listingu 5**. Na początku jest zerowany adres kolumny Y=0, a potem w pętli są zerowane wszystkie bajty na stronie. Po ich wyzerowaniu jest ustawiany komendą Set Page Address kolejny numer strony i zerowanie się powtarza. Cykl zerowania jest wykonywany dla wszystkich 8 stron pamięci.

Zainicjowany wyświetlacz z wyzerowaną pamięcią RAM jest gotowy do pracy. Wyświetlacz graficzny jest elementem wymagającym do obsługi stosunkowo sporych zasobów mi-

Tabela 1. Połączenie wyświetlacza z mikrokontrolerem

Złącze wyświetlacza Con12	Linie portów PORTC J16 PORTB J14
D0	RC0
D1	RC1
D2	RC2
D3	RC3
D4	RD4
D5	RC5
D6	RC6
D7	RC7
RS	RB0
E	RB1
RW	RB2
CS1	RB4
CS2	RB3
RES	RB5

Listing 2. Zapisanie danej do wyświetlacza

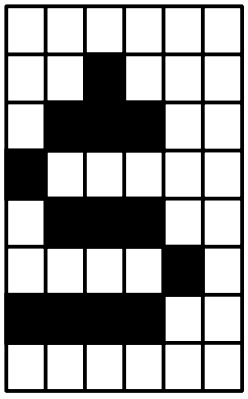
```
//*****************************************************************
//funkcja zapisania danych przez ,magistralę  wyświetlacza
//dana jest w argumencie dana,  typ rejestru: sterujący/dana RAM,
//wybrany sterownik KS0108 w ctrl
//*****************************************************************
void WriteLcd(unsigned char reg, unsigned char dana, char ctrl){
  if(reg==REGS) RS=0;  //zapisanie rejestru sterującego
  else RS=1;  //zapisanie pamięci RAM wyświetlacza
  delay();
  E=0; //E=0
  if(ctrl==0) CS1=1; else CS2=1;
  delay();
  RW=0;  //RW=0
  delay();
  delay();
  TRISC=0;  //PORTC wyjściowy
  LATC=dana;
  E=1;  //E=1
  delay();
  E=0;  //E=0
  CS1=0;  //CS1=CS2=0
  delay();
  CS2=0;
//czekaj na wyzerowanie flagi BUSY
  while((ReadStatus(ctrl)&0x80)==0x80);
}
//pomocnicza funkcja opóźnienia
void delay(void){
  asm(„nop”);asm(„nop”);asm(„nop”);
  asm(„nop”);asm(„nop”);asm(„nop”);
  asm(„nop”);asm(„nop”);asm(„nop”);
  asm(„nop”);asm(„nop”);asm(„nop”);
  asm(„nop”);asm(„nop”);asm(„nop”);
  asm(„nop”);asm(„nop”);asm(„nop”);
}
```

Listing 1. Definicja linii sterujących

```
#define RS RB0
#define E RB1
#define RW RB2
#define CS2 RB3
#define CS1 RB4
#define RES RB5
```

Tabela 2. Zestawienie komend sterownika KS0108

Komenda	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0	Funkcje
Display ON/OFF	0	0	0	0	1	1	1	1	1	1/0	D0=0 wyświetlacz wyłącz D0=1 wyświetlacz załącz RAM obrazu bez zmian
Set colum address Y address	0	0	0	1	Y adres (0-63)						Ustawia adres kolumny Y
Set page Address X address	0	0	1	0	1	1	1	X adres (0-7)			Ustawia adres strony
Display start Line Z address	0	0	1	1	Display start Line (0-63)						Ustawia linię początkową wyświetlacza
Status read	0	1	BUSY	0	ON/OFF	Reset	0	0	0	0	BUSY =0 gotowy BUSY =1 zajęty ON/OFF=0 wyśw. załączony ON/OFF=1 wyśw. wyłączony Reset=0 praca normalna Reset=1 cykl zerowania
Write Display Data	1	0	Write data								Zapisanie danej do pamięci Obrazu – adres Y jest inkrementowany
Read Display Data	1	1	Read data								Odczytanie danej z pamięci obrazu



Rysunek 6. Matryca tworząca znak „ś"

krokontrolera. Szczególnie dotyczy to pamięci stałej, w której są przechowywane bitmapy przeznaczone do wyświetlania. Jednak jak już wspomniałem wyświetlacz graficzny można z powodzeniem wykorzystywać do pracy w trybie tekstowym i dlatego najpierw zajmiemy się wyświetlaniem znaków alfanumerycznych. KS0108 nie ma wbudowanego generatora znaków (tablicy ze zdefiniowanymi wzorcami). Większość sterowników wyświetlaczy graficznych nie ma generatora, bo użytkownik może sobie zdefiniować, zależnie od potrzeb, własne znaki.

Znak alfanumeryczny może być zbudowany w oparciu o matrycę 8×6 pikseli. Same znaki mają wysokości 7 pikseli i szerokość 5 pikseli, ale konieczne jest dodanie wolnych miejsc na odstępy pomiędzy znakami i pomiędzy wierszami. Na **rysunku 6** pokazano matrycę tworzącą polską literę „ś". Jeżeli przyjmiemy konwencję, że piksel jest zapalony, gdy odpowiadający mu bit w pamięci obrazu jest ustawiony, to znak z **rysunku 6** jest definiowany przez 6 bajtów: 48hex, 54hex, 56hex, 20hex i 00hex. Na **listingu 5** zamieszczono fragment przykładowej tablicy z generatorem polskich znaków.

Listing 3. Odczytanie rejestru statusowego

```
//*********************************************************************
//funkcja odczytywania rejestru statusowego
//*********************************************************************
unsigned char ReadStatus(char ctrl){
unsigned char data;
  E=0;  //E=0
  delay();
  RS=0;  //odczytanie rejestru statusowego
  //RS=1 dla odczytania danych z pamięci obrazu
  delay();
  RW=1;  //RW=1 - odczytanie
  delay();
  if(ctrl==0) CS1=1; else CS2=1;
  delay();
  //E=0;
  TRISC=0xff;  //dane wejściowe
  E=1;  //E=1
  delay();
  TRISC=0xff;
  data=PORTC;//zapisanie danych z magistrali do data
  delay();
  E=0;
  return(data);
}
```

Listing 4. Inicjalizacja wyświetlacza

```
void InicLcd(){
  CS1=0;
  CS2=0;
  RES=0;
  delay_ms(5);
  RES=1;
  delay();
  WriteLcd(REGS,0x3f,0);       //lewy LCD zał.
  WriteLcd(REGS,0xc0,0);       //pierwsza linia
  delay();
  WriteLcd(REGS,0x3f,1);       //prawy LCD zał.
  WriteLcd(REGS,0xc0,1);       //pierwsza linia
}
```

Każdej grupie 6 bajtów w tablicy przypisuje się kod znaku. Dla tablicy *rom_gen* z listingu 5 znak „ś" ma kod 0, znak „ą" kod 1, znak „ć" kod 2. Dobrym zwyczajem jest takie zaprojektowanie tablicy generatora znaków, aby kody znaków pokrywały się z kodami ASCII. Bardzo ułatwia to późniejsze tworzenie funkcji wyświetlających ciągi znaków. Wyświetlanie znaków alfanumerycznych będzie polegało na wpisaniu 6 kolejnych bajtów pobranych z tablicy generatora znaków pod określoną lokalizację (pozycję) w pamięci RAM. Pozycja ta jest określana dwoma współrzędnymi *y* (pozycja w wierszu) i *x* (wiersz). W nomenklaturze używanej w sterowniku KS0108 współrzędna *y* nazywana jest kolumną, a współrzędna *x* stroną. Jeden sterownik steruje 64 kolumnami i 8 wierszami. Współrzędna *column* (y) może zmieniać się dla całego wyświetlacza w zakresie 0…127, a współrzędna *page* (x) w zakresie 0…7.

Pokazana na **listingu 6** procedura *LcdWriteData* zapisuje bajt z argumentu *data* pod lokację pamięci RAM wyświetlacza określoną przez współrzędne *column* i *page*. Na podstawie zakresu wartości argumentu column wybierany jest

Listing 4. Zerowanie zawartości pamięci RAM wyświetlacza

```
#define DAT 1
#define REGS 0
//*****************************************************************
// funkcja zerowania zawartości RAM obu sterowników KS0108
//*****************************************************************
void ClrLcd(void){
unsigned char i,j;
  WriteLcd(REGS,0x40,0);  //adres Y=0
  for(i=0;i<8;i++)  //sterownik lewej połowy wyświetlacza
  {
    WriteLcd(REGS,0xb8|i,0);  // komenda ustawiania adresu strony
    for(j=0;j<64;j++)
    WriteLcd(DAT,0x00,0);  //zerowanie pamięci
  }
  delay();
  WriteLcd(REGS,0x40,1);  //adres Y=0
  for(i=0;i<8;i++)  //sterownik prawej części wyświetlacza
  {
    WriteLcd(REGS,0x00|i,1);  // komenda ustawiania adresu strony
    for(j=0;j<64;j++)
    WriteLcd(DAT,0,1);
  }
}
```

Listing 5 fragment tablicy generatora znaków

```
const char rom_gen[]={
0x48,0x54,0x56,0x54,0x20,0,//kod ś
0x20,0x54,0x54,0xd4,0x78,0,//kod ą
0x38,0x44,0x46,0x44,0x20,0,//kod ć
```

automatycznie lewy (dla column <64) lub prawy (dla column >=64) sterownik KS0108.

Dana z argumentu *data* może być zapisywana bez modyfikacji lub w postaci zanegowanej. Decyduje o tym wartość zmiennej globalnej *atr*. Jeżeli atr=0, to dane są wpisywane bez modyfikacji. Kiedy atr=1, wtedy wpisywane dane są negowane. Umożliwia to wyświetlanie tekstu w postaci „negatywu" wyróżniającego się z pozostałego tekstu. Jest to prosty i efektowny sposób na wyświetlanie na przykład wybranego fragmentu menu.

Jeżeli przyjmiemy, że będą wyświetlane tylko znaki alfanumeryczne, to można wprowadzić dwie współrzędne określające położenie znaku na ekranie: numer wiersza (0…7) i pozycję znaku w wierszu (0…20). Znaków w wierszu może być 21, bo każdy znak ma szerokość 6 pikseli, a pikseli w wierszu jest 128 (21×6=126). Posługiwanie się takimi współrzędnymi ułatwia pozycjonowanie napisów na ekranie wyświetlacza.

Funkcję wyświetlającą znak pokazano na **listingu 7**. Kod znaku jest umieszczany w argumencie *code*, a współrzędne w argumentach *col* i *page*. Argument *col* musi mieć taką wartość, aby cały znak mógł się wyświetlić na ekranie. Jeżeli tak nie jest, to funkcja nic nie wyświetli i zakończy działanie. Współrzędna określająca kolumnę sterownika jest wyliczana przez pomnożenie argumentu *col* przez 7. Indeks *code_point* początku pobierania 6 bajtów z tablicy wzorca *rom_gen* jest wyliczany przez pomnożenie kodu znaku przez 6. Potem z tablicy jest pobieranych 6 kolejnych bajtów. Bajty te są wpisywane do pamięci RAM wyświetlacza pod współrzędne określone przez *col* (argument *col* pomnożony przez 7) i *page*.

Listing 6. Zapisanie 8-bitowej danej pod lokację określoną przez *column* i *page*

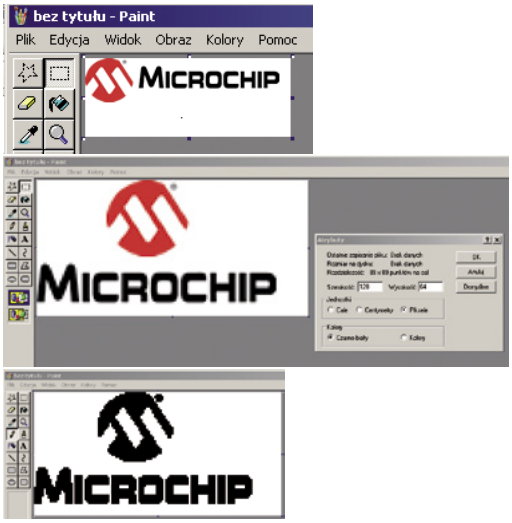
```
/*************************************************************
zapisanie danej do sterownika wyświetlacza po lokację określoną
przez współrzędne y(piksele w wierszu) i x (wiersz)
*************************************************************/

void LcdWriteData(unsigned char data, unsigned char column, unsigned char
page){
  page&=7;
  if(column>=64)    //sterownik prawej części wyswietlacza
  {
    WriteLcd(REGS,page|0xb8,1);//adres strony - wiersz
    column=column-64;
    WriteLcd(REGS,column|0x40,1);//kolumna - piksel w wierszu
    if(atr==0) WriteLcd(DAT,data,1); else WriteLcd(DAT,~data,1);
  } else  //sterownik lewej części wyświetlacza
  {
    WriteLcd(REGS,page|0xb8,0);//adres strony - wiersz
    WriteLcd(REGS,column|0x40,0);//kolumna - piksel w wierszu
    if(atr==0) WriteLcd(DAT,data,0); else WriteLcd(DAT,~data,0);
  }
}
```

Listing 7. Wyświetlenie jednego znaku

```
char WriteChar(char code,unsigned char col,unsigned char page)
{
  unsigned int code_point=0;
  col=col*7;
  if((col+6)>122) return(0);//ten znak sie nie zmieści
  PosLcd(col,page);
  code_point=(unsigned int)(code*6);
  LcdWriteData(rom_gen[code_point++],col++,page);
  LcdWriteData(rom_gen[code_point++],col++,page);
  LcdWriteData(rom_gen[code_point++],col++,page);
  LcdWriteData(rom_gen[code_point++],col++,page);
  LcdWriteData(rom_gen[code_point++],col++,page);
  LcdWriteData(rom_gen[code_point],col,page);
  return(1);
}
```

Fotografia 7. Wyświetlacz w trybie tekstowym



Rysunek 8. Etapy edycji obrazka: a) wklejenie banera ze strony *www.microchip.com*, b) edycja i okno atrybutów, c) obrazek o wymiarach 128×64 piksele po konwersji (powiększenie)

Rysunek 9. Okno programu bmp2c z wczytana bitmapą

Użycie funkcji *WriteChar* jest wygodne, kiedy trzeba wyświetlić pojedynczy znak na określonych współrzędnych. W praktyce często konieczne jest wyświetlenie napisu (ciągu znaków) od określonej pozycji. Do tego celu lepsza będzie funkcja, w której pozycję znaku określi się raz na początku przez zewnętrzną funkcję i zapisze w zmiennych globalnych *col* i *page* (**listing 8**).

Funkcji z listingu 8 można użyć do wyświetlania tekstu (ciągu znaków ASCII) umieszczonego w argumencie funkcji (**listing 9**).

Na **fotografii 7** i **listingu 10** są pokazane przykładowe możliwości wyświetlania tekstu na testowanym wyświetlaczu graficznym. Po wpisaniu do zmiennej globalnej *atr* wartości 0x01 program wyświetla tekst w negatywie.

Oprócz wyświetlania tekstu wyświetlacz graficzny może wyświetlać bitmapy (rysunki).

Listing 8. Określanie pozycji i wyświetlanie jednego znaku

```
void PosLcd(unsigned char x, unsigned char y)
{
  page=y;
  col=x*7;
}

void WrChar(char code)
{
  unsigned int code_point;
  if((col+6)>122)
  return;//ten znak sie nie zmiesci
  code_point=(code*6);
  LcdWriteData(rom_gen[code_point++],col++,page);
  LcdWriteData(rom_gen[code_point++],col++,page);
  LcdWriteData(rom_gen[code_point++],col++,page);
  LcdWriteData(rom_gen[code_point++],col++,page);
  LcdWriteData(rom_gen[code_point++],col++,page);
  LcdWriteData(rom_gen[code_point],col++,page);
}
```

Listing 9. Procedura wyświetlania tekstu

```
void LcdTxt(const char * s, char x, char y)
{
  PosLcd(x,y);
  while(*s)
  WrChar(*s++);
}
```

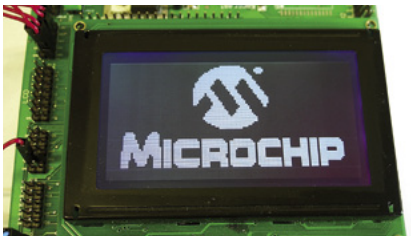
Listing 10. Fragment programu wyświetlający tekst jak na fotografii 7

```
  atr=0;
  LcdTxt(„ KURS PROGRAMOWANIA”,0,0);
  atr=1;LcdTxt(„ Mikrokontrolerow „,1,2);
  LcdTxt(„ firmy  Microchip „ ,1,3);atr=0;
  LcdTxt(„    PIC18F2320  „,0,5);
  LcdTxt(„  Tomasz Jablonski”,0,7);
```

Listing 11. Wyświetlanie pełnoekranowej bitmapy

```
void LcdBmp(const char *bmp){
  char x,y;
  int i=0;
  for(x=0;x<8;x++)
    for(y=0;y<128;y++) LcdWriteData(*bmp++,y,x);
}
```

Rysunek przeznaczony do wyświetlenia musi być przekonwertowany do formatu bitmapy monochromatycznej o wymiarach 128×64 piksele. Doskonale do tego celu nadaje się program graficzny *Paint* dystrybuowany wraz z systemem Windows. Przykładowe etapy edycji obrazka w tym programie pokazano na rysunku 10. Teraz zapisaną bitmapę trzeba przekonwertować na tablicę w języku C. Ja do tego celu używam programu *bmp2c.exe* autorstwa Jerzego Szczesiula (**rysunek 9**). Po ustaleniu rozmiaru bitmapy trzeba kliknąć na przycisk BMP, wczytać zapisaną wcześniej bitmapę i zaznaczyć opcję V8. Plik z tablicą jest generowany po kliknięciu na C i umieszczany w schowku systemowym tak, że można go wkleić do pliku projektu. Na potrzeby kursu utworzyłem nowy plik *bmpc.c* i tam umieściłem wygenerowaną *t* tablicę o wielkości 1024 bajtów (8 banków po 128 wierszy każdy, czyli 128×8=1024). W tym momencie mamy za sobą większość pracy potrzebnej do wykonania, by wyświetlić pełnoekranową bitmapę, jak pokazano na **rysunku 10**.



Rysunek 10. Wyświetlanie obrazu bitmapy z przykładu

Samo wyświetlanie będzie polegało na pobieraniu ośmiu 128 bajtowych „porcji” bajtów z tablicy. Najpierw adresujemy *bank 0* pamięci i wpisujemy 128 bajtów, adresując licznik kolumn od 0 do 127. Potem adresujemy *bank 1* i powtarzamy wpisanie 128 bajtów i tak dalej, aż do zapisania *banku 7*. Jeżeli wykorzystamy do tego celu procedurę *LcdWriteData* z listingu 6, to wyświetlanie pełnoekranowej bitmapy będzie zadziwiająco proste – **listing 11**.

Tomasz Jabłoński, EP
tomasz.jablonski@ep.com.pl