Kurs programowania mikrokontrolerów PIC (4)

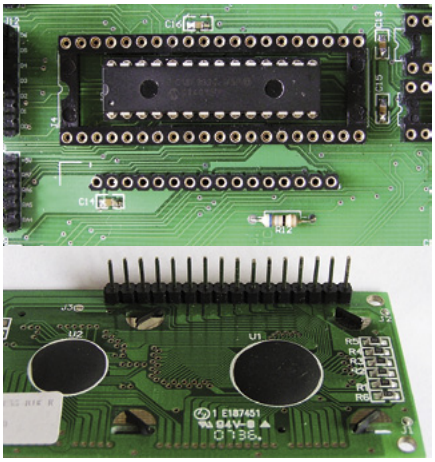
Obsługa modułu wyświetlacza LCD

Dodatkowe materiały na CD/FTP:
ftp://ep.com.pl, user: 16732, pass: 630v2nfb
- poprzednie części kursu

Niestety, jeszcze nie doczekaliśmy się urządzeń, które komunikują się z nami w „ludzki" sposób, tzn. za pomocą słów i gestów. Być może kiedyś w „Elektronice Praktycznej" pojawi się kurs opisujący sposób wykonania takiego interfejsu, ale na to trzeba jeszcze poczekać. A na razie jednymi z najczęściej stosowanych elementów menu użytkownika są klawisze, manipulatory i wyświetlacze. Poprzednio opisywaliśmy sposób obsługi enkodera, klawiatury i wyświetlacza LED. Teraz zajmiemy się tekstowym modułem wyświetlacza LCD, a w kolejnym odcinku – wyświetlaczami graficznymi.

Alfanumeryczny wyświetlacz LCD ze sterownikiem zgodnym z HD4470 jest niekwestionowanym standardem w bardziej rozbudowanych interfejsach użytkownika. Jego zaletą jest dobrze udokumentowany układ sterownika oraz niska cena przy doskonałej jakości wyświetlania i sporych możliwościach. Zwykle moduł nie wymaga dodatkowych napięć zasilających matrycę i jest zasilany napięciem 5 V. To napięcie zasilania zaczyna trochę przeszkadzać w układach, które są coraz częściej zasilane napięciem 3,3 V lub niższym. Jednak w praktyce bez problemu można go sterować poziomami logicznymi CMOS +3,3 V przy założeniu, że mikrokontroler tylko wysyła dane do sterownika, a nic z niego nie odczytuje. Mikrokontroler na płytce ewaluacyjnej jest zasilany napięciem +5 V i ten problem nie występuje. Ponieważ wspomniany wyświetlacz jest tak popularny, na płytce ewaluacyjnej przewidziano dla niego specjalne gniazdo oznaczone LCD1 przystosowane do połączenia wyprowadzeń umieszczonych w jednym rzędzie (**fotografia 1**).

Wszystkie sygnały sterujące są doprowadzone do złącza J12 oznaczonego jako LCD. Do regulacji kontrastu jest przeznaczony potencjometr PR1 opisany jako „Kontrast".



Fotografia 1. Gniazdo i wtyk wyświetlacza

Własności modułu LCD

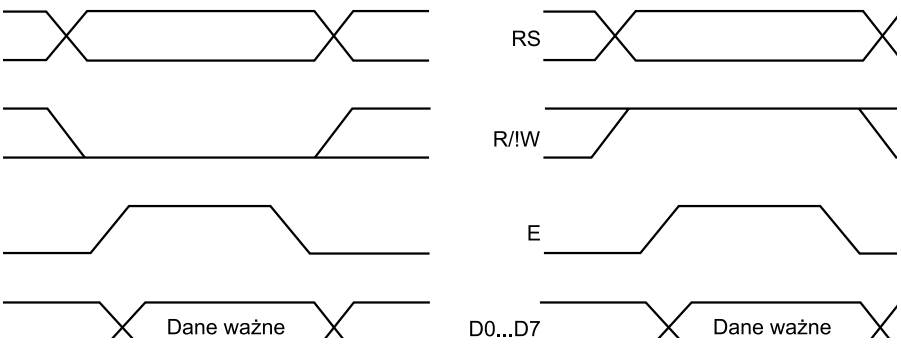
Mikrokontroler komunikuje się ze sterownikiem wyświetlacza za pomocą interfejsu równoległego zbudowanego z 8 linii danych oznaczonych D0...D7 oraz trzech sygnałów sterujących: R/W, RS i E. Interfejs jest zgodny ze standardem stosowanym w 8-bitowych mikroprocesorach produkowanych przez Motorolę.

8-bitowe słowa danych są przesyłane do sterownika wyświetlacza i z niego odczytywane (kody znaków, instrukcje sterujące itp.). Dwukierunkowa magistrala danych D0...D7 może pracować w dwóch trybach:

- 8-bitowym, w którym całe słowo jest przesyłane w jednym cyklu (zapisu lub odczytu).
- 4-bitowym, w którym słowo jest przesyłane w dwóch cyklach. Najpierw przesyłana jest bardziej znacząca połowa bajtu, a następnie mniej znacząca. W tym trybie wykorzystywane są tylko linie danych D4...D7 oraz sterujące.

Tryb 4-bitowy jest stosowany częściej, pomimo, że jego obsługa jest bardziej skomplikowana, a transmisja danych przebiega wolniej. Jego zaletą jest to, że pozwala dołączyć wyświetlacz do mikrokontrolera za pomocą zaledwie 6 lub 7 linii I/O. Dzięki temu taki wyświetlacz można dołączyć nawet do mikrokontrolerów z niewielką liczbą portów wejścia/wyjścia, na przykład do popularnych PIC16F628 lub PIC16F88.

Sygnał R/W steruje kierunkiem przepływu danych przez magistralę danych. Jeżeli linia R/W ma poziom niski, to dane są zapisywane do sterownika wyświetlacza, natomiast gdy wysoki, to dane są



Rysunek 2. Przebiegi na magistrali w trakcie zapisywania i odczytywania danych

Nr znaku	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Adres znaku w wierszu 1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Adres znaku w wierszu 2	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

Rysunek 3. Organizacja pamięci dla wyświetlacza 2 linie×16 znaków

Instrukcja	Kod										Funkcja	Czas realizacji
	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0		
Clear Display	0	0	0	0	0	0	0	0	0	0	Wygaszenie ekranu – kursor na pozycję wyjściową (adres 0)	1,64 ms
Cursor Home	0	0	0	0	0	0	0	0	1	*	Kursor na pozycję wyjściową – zawartość DD RAM bez zmian	1,64 ms
Entry Mode	0	0	0	0	0	0	0	1	ID	S	Ustawienie kierunku przesuwu kursora, kiedy dane są zapisywane lub odczytywane z pamięci DDRAM	40 µs
Dispaly ON/OFF	0	0	0	0	0	0	1	D	C	B	Włączenie/wyłączenie ekranu (D), kursora (C) oraz migania kursora (B)	40 µs
Cursor/display shift	0	0	0	0	0	1	S.C.	RL	*	*	Przesunięcie kursora i przesunięcie o jedną poz. Wyświetlanego tekstu bez zmiany zaw. DDRAM	40 µs
Function Set	0	0	0	0	1	DL	N	F	*	*	Ustawienie magistrali 8- lub 4-bitowej, parametru Duty i wzorca 5×7 lub 5×10 pikseli	40 µs
CG RAM addres set	0	0	0	1	Agc						Ustawienie adresu CG RAM	40 µs
DD RAM addres set	0	0	1	Add							Ustawienie adresu DD RAM	40 µs
BF/address read	0	1	BF	AC							Odczytanie znacznika zajętości oraz licznika adresu	0 µs
Write data to DDRAM Or CGRAM	1	0	Zapisywana dana								Zapisywanie do pamięci DDRAM lub CG RAM	40 µs
Data read from DDRAM Or CGRAM	1	1	Odczytywana dana								Odczytywane dane z pamięci DDRAM lub CGRAM	40 µs

ID=1 inkrementacja
ID=0 dekrementacja
SC=1 przesuwanie wyświetlania
SC=0 przesuwanie kursora
C=1 kursor włączony
C=0 kursor wyłączony
S=1 przesuwanie wyświetlania
S=0 przesuwanie wyłączone
N=1 1/16 duty
N=0 1/8 lub 1/16 duty
BF=1 moduł zajęty
BF=0 wysyłane dane akceptowane
B=1 miganie kursora
B=0 miganie wyłączone
F=1 znaki 5×10
F=0 znaki 5×7
DL=1 interfejs 8-bitowy
DL=0 interfejs 4-bitowy
RL=1 przesuwanie w prawo
RL=0 przesuwanie w lewo
D=1 ekran włączony
D=0 ekran wyłączony

Rysunek 3. Zestawienie komend sterownika HD44780

odczytywane. Sygnał RS odpowiada za wybór jednego z dwóch rejestrów wewnętrznych IR (rejestr instrukcji) i DR (rejestr danych) sterownika LCD. Sygnał E (enable) służy do uaktywnienia wymiany danych pomiędzy mikrokontrolerem i sterownikiem wyświetlacza. Przebiegi czasowe charakterystyczne dla zapisu i odczytu sterownika wyświetlacza pokazano na **rysunku 2**.

Sterowniki zgodne z HD44780 mają trzy rozdzielne obszary wewnętrznej pamięci:

– Pamięć CG ROM (*Character Generator ROM*), w której są umieszczone wzorce znaków do wyświetlenia.
– Pamięć DD RAM (*Display Data RAM*), w której jest przechowywany wyświetlany obraz.
– Pamięć CG RAM (*Character Generator RAM*) ma pojemność używaną do definiowania własnych znaków.

W pamięci CG ROM są umieszczane wzorce znaków, najczęściej o rozdzielczości 5×7 lub 5×10 pikseli. 8-bitowy kod wyświetlanego znaku adresuje obszar w pamięci w CG ROM, w którym znajduje się blok bajtów definiujących jego kształt. Kody podstawowych znaków zdefiniowanych w tej pamięci pokrywają się z kodami ASCII, co znacznie ułatwia obsługę wyświetlacza.

Kody z zakresu 128...255 odpowiadają znakom spoza alfabetu łacińskiego. Zależnie od wykonania, mogą tam być umieszczone znaki charakterystyczne innych alfabetów. Rodzaj wyświetlanego wzorca jest wybierany instrukcją sterującą *Function Set*.

W pamięci DD RAM o pojemności 80 bajtów są przechowywane kody znaków przeznaczonych do wyświetlenia. Większość popularnych wyświetlaczy nie wykorzystuje całej przestrzeni adresowej DD RAM. Przykład użycia pamięci DD RAM dla wyświetlacza o organizacji 2 wiersze po 16 znaków pokazano na **rysunku 3**.

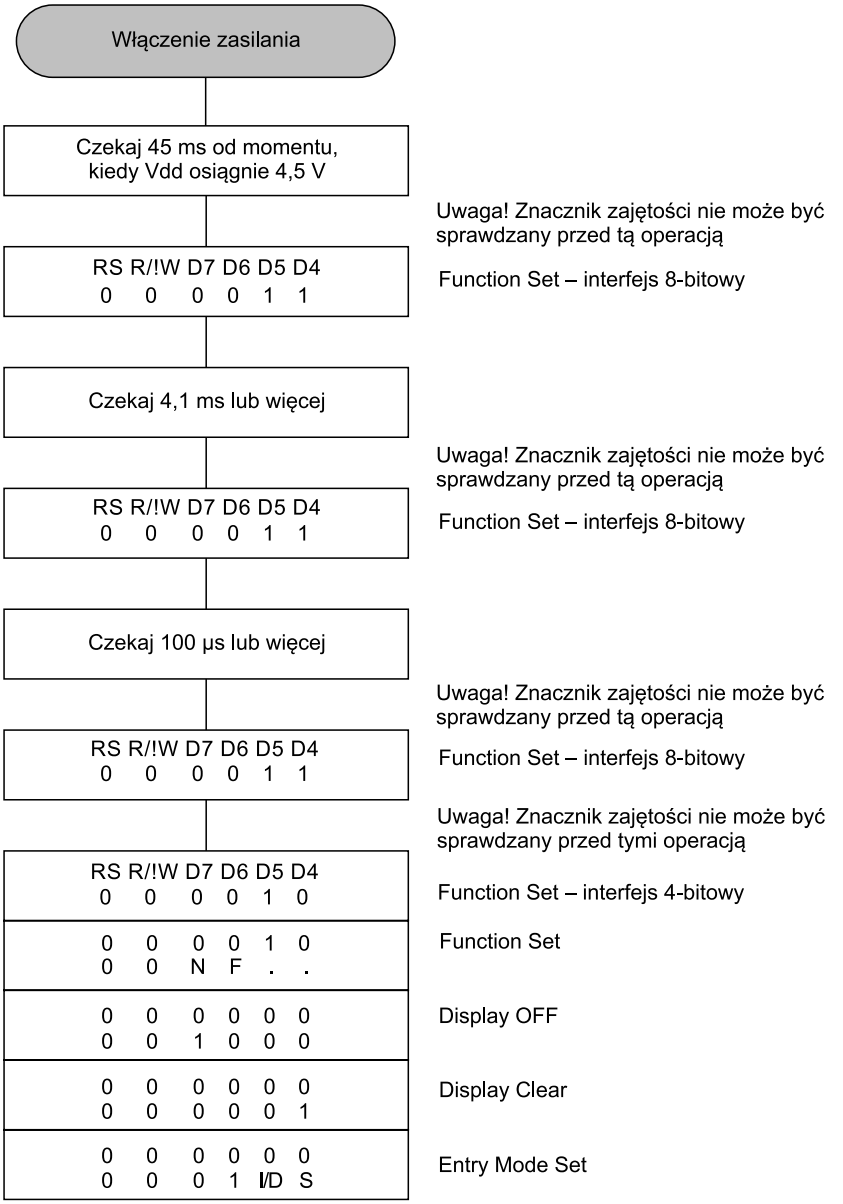
Pamięć CG RAM (*Character Generator RAM*) ma pojemność 64 bajtów i jest używana do definiowania własnych znaków. Funkcjonalnie odpowiada ona pamięci CG ROM z tą różnicą, że można do niej zapisywać dowolne wartości, dzięki czemu można samodzielnie zdefiniować kształty (mieszczące się w matrycy7×5 punktów) 8 znaków. Definiowane przez użytkownika znaki mają kody od 0 do 7. Pierwsze 8 bajtów wpisanych do pamięci CG RAM definiuje znak o kodzie 0, następne 1 i tak dalej. W każdym bajcie pamięci CG RAM wykorzystywanych jest tylko 5 mniej znaczących bitów.

Z punktu widzenia mikrokontrolera kontroler wyświetlacza LCD to dwa 8-bitowe rejestry: instrukcji IR i danych DR. Wybór (adresowanie) tych rejestrów jest dokonywany za pomocą linii sterującej RS. Jeżeli linia RS jest ustawiona (ma poziom wysoki), to transfer danych dotyczy rejestru danych DR, natomiast jeżeli wyzerowana (ma poziom niski), to zapisywany lub odczytywany jest rejestr instrukcji IR.

Po zapisaniu danej do rejestru IR, zależnie od wartości bitów DB6 i DB7, sterownik wyświetlacza interpretuje ją jako polecenie lub adres w pamięci CG RAM albo DD RAM. Na **rysunku 5** umieszczono przegląd poleceń sterujących pracą modułu znakowego wyświetlacza LCD.

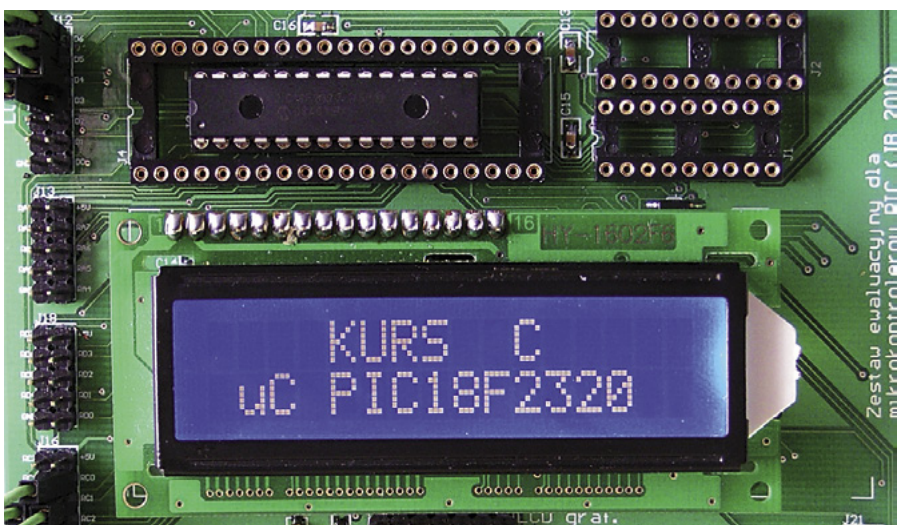
Po włączeniu zasilania sterownik jest inicjalizowany do ustawień domyślnych, jeżeli napięcie zasilania narasta od 0,2 V do 4,5 V w czasie 0,1...10 ms. W praktyce jednak zawsze wykonuje się programową inicjalizację do interfejsu 8- lub 4-bitowego. Należy zauważyć, że interfejs 4-bitowy zawsze musi być zainicjalizowany programowo, bo domyślnie sterownik przyjmuje słowa o długości 8-bitów. Schematycznie przebieg takiej inicjalizacji pokazano na **rysunku 5**.

Po wysłaniu instrukcji lub kodu znaku do modułu wyświetlacza jest wymagany pewien czas na wykonanie operacji związanej z przesłaną daną. Zajętość modułu jest sygnalizowana za pomocą znacznika zajętości (*Busy Flag*). Stan tego znacznika można odczytać, gdy sygnał RS=0 i R/W=1. W odczytanym bajcie bit D7 jest wskaźnikiem zajętości BF. Jeżeli BF=1, to sterownik jest zajęty i wysyłana do niego dana zostanie zignorowana. Gotowość modułu do przyjęcia nowej danej sygnalizowana jest wyzerowaniem flagi BF.



Rysunek 4. Programowa inicjalizacja modułu wyświetlacza LCD dla interfejsu 4-bitowego

Przed wysłaniem nowej danej do modułu trzeba się upewnić, że BF=0. Można również odczekać dłużej niż przez czas po-



Fotografia 5. Zamontowany i działający wyświetlacz

Listing 1. Procedura inicjalizacji wyświetlacza

```
Procedura inicjalizacyjna musi być wywołana przed pierwszym użyciem
wyświetlacza  - listing 10
void InicLcd(void)                  //inicjalizacja modułu LCD {
  W=0;                              //zapisywanie do sterownika
  delay_ms(100);                    //ustabilizowanie się napięcia zasilania
  RS=0;
  asm(„nop”);
  EN=0;
  WriteRiInit(3);
  WriteRiInit(3);
  WriteRiInit(3);
  WriteRiInit(2);
  WriteRi(0x28);
  WriteRi(8);                       //display off
  WriteRi(1);                       //display clear
  WriteRi(6);                       //entry mode
  WriteRi(0x0c);                    //display on
}
```

Listing 2. Funkcja pomocnicza wysyłająca na magistralę słowo 4-bitowe

```
//funkcja zapisu 4 bitów na
//magistrali w czasie
//inicjalizacji interfejsu
//4-bitowego
//zapisywana dana w zmiennej regi
void WriteRiInit(unsigned char regi)
{
  asm(„nop”);
  EN=1;
  asm(„nop”);
  EN=1;
  RS=0;
  PORTC=regi;
  asm(“nop”);
  EN=0;
  delay_ms(5);
}
```

Listing 3. Zapisanie 8-bitowej danej przez magistralę 4-bitową

```
//funkcja zapisania danej na magistralę
//zapisywana dana w zmiennej data
//przed wywołaniem musi być ustalony stan linii RS
//RS=1 dane RS=0 instrukcje
void WriteLcd(unsigned char data) {
  EN=1;
  PORTC=PORTC&0xf0;
  PORTC=PORTC|(data>>4); //zapisanie 4 starszych bitów
  EN=0;
  asm(„nop”);
  EN=1;
  PORTC=PORTC&0xf0;
  PORTC=PORTC|(data&0x0f); //zapisanie 4 młodszych bitów
  asm(„nop”);
  EN=0;
  delay_ms(2);
}
```

Listing 4. Funkcje zapisywania rejestrów DR i IR

```
//funkcja zapisania rejestru danych
//zapisywana dana w zmiennej data
void WriteRd (unsigned char data) {
  RS=1;
  WriteLcd(data);
}

//funkcja zapisania rejestru
//instrukcji
//zapisywana dana w zmiennej regi
void WriteRi(unsigned char regi) {
  RS=0;
   WriteLcd(regi);
}
```

trzebny na wykonanie poprzedniego polecenia. Ten ostatni sposób jest często stosowany, bo pozwala na zmniejszenie liczby linii sterujących o jedną, dołączoną do R/W. W tym rozwiązaniu linię R/W na stałe łączy się z masą, a zewnętrzny sterownik tylko zapisuje dane do modułu wyświetlacza.

Oprogramowanie sterujące

W zademonstrowanych procedurach obsługi wyświetlacza zainicjujemy go do komunikacji przez interfejs 4-bitowy z sygnałem R/W zwartym na stałe do masy i nie będziemy używać testowania BF. W zamian, po przesłaniu danych, zostanie odmierzone stosowane opóźnienie. W trybie 4-bitowym używa się 4 starszych linii magistrali D4...D7. Połączymy je z liniami RC0...RC3 portu PORTA. Linia RS zostanie połączona z RC4, a EN z RC5.

Na **listingu 1** zamieszczono procedurę inicjowania wyświetlacza LCD w trybie interfejsu 4-bitowego. Po jej wykonaniu interfejs sterownika wyświetlacza przyjmował bajty w dwóch porcjach po 4 bity, znaki będą wyświetlane w matrycy 5×7 pikseli, a po każdym wpisaniu kodu znaku do pamięci DD RAM adres będzie automatycznie inkrementowany. Wyłączone będzie wyświetlanie kursora i miganie znaku wskazywanego przez kursor. Instrukcja *Clear Display* czyści pole wyświetlacza, czyli wpisuje do całej pamięci DD RAM znaki spacji (0x20).

Ponieważ w trakcie inicjowania LCD na magistralę są wysyłane tylko słowa 4-bitowe, na potrzeby funkcji inicjalizacji napisano procedurę pomocniczą WriteRiInit (**listing 2**).

Jak wspomniano, po prawidłowo wykonanej inicjalizacji sterownik modułu wyświetlacza LCD jest gotowy do odbierania danych 8-bitowych w porcjach po 4 bity: najpierw 4 starsze, potem 4 młodsze. Odpowiednią funkcję, która realizuje tak rozumianą transmisję danych, pokazano na **listingu 3**.

Na końcu procedury jest odmierzane opóźnienie 2 ms. Zgodnie z tabelą z rysunku 5 jest to zbyt dużo dla większości operacji zapisów do pamięci danych i poleceń dla sterownika. Dłuższy czas nie przeszkadza jednak w prawidłowym wykonywani rozkazów, a przy tym nie trzeba różnicować opóźnień. Jeżeli dłuższy czas obsługi jest problemem dla danej aplikacji, to można tę procedurę zmodernizować, wprowadzając dodatkowy argument określający rodzaj zapisywanego rejestru i na podstawie tego argumentu dla komend odliczać 2 ms, a dla danych 40 µs.

Funkcja *WriteLcd* jest używana zarówno przy zapisywaniu rejestru danych DR, jak i rejestru instrukcji IR. O tym, do którego rejestru

trafi dana przesyłana magistralą, decyduje poziom bitu RS (**listing 4**).

Niezbędna do obsługi wyświetlacza jest funkcja pozycjonująca wyświetlane znaki na ekranie. Przykład jej implementacji pokazano na **listingu 5**. Położenie znaku na ekranie LCD określają argumenty: x – numer kolumny, y – numer wiersza. Na początku wykonywania procedury jest sprawdzany zakres współrzędnych określanych przez argumenty dla wyświetlacza o organizacji 2×16 znaków. Na podstawie argumentu *y* jest wyliczany adres w pamięci RAM wyświetlacza. Jeżeli znaki mają być wyświetlane w dolnej linii, to muszą być zapisywane do pamięci wyświetlacza od adresu 0x40. Adres, od którego będą zapisywane dane, jest ustalany za pomocą komendy *CG RAM Address Set* (**rysunek 6**).

Wyświetlacz alfanumeryczny jest często używany do wyświetlania komunikatów tekstowych. Przydatna do tego celu będzie funkcja *DispLcd* (w połączeniu z funkcją *PosLcd*) pokazana na **listingu 7**. Jej argumentem jest ciąg znaków umieszczonych w pamięci Flash mikrokontrolera. Wysyłanie funkcji może wyglądać jak na **listingu 8**. Przy wyświetlaniu pojedynczego znaku ASCII zamiast funkcji *DispLcd* można posłużyć się następującymi poleceniami:

```
PosLcd(1,2);
WriteRd (‚T’);
```

Kompilator wpisze jako argument funkcji *WriteRd* kod znaku „T”, czyli *WriteRd* jest równoważne *WriteRd*(0x54).

Definiowanie własnych znaków jest trudne. Trzeba do sterownika wpisać komendę ustawienia adresu początkowego w pamięci CGRAM i po niej wpisać do rejestru danych 8 bajtów z określoną zawartością. Na **listingu 9** pokazano funkcję definiującą znak stopnia stosowany przed literką „C” dla temperatury wyświetlanej w stopniach Celsjusza. Ponieważ bajty są zapisywane od adresu 0 w tablicy CGRAM, to kod zdefiniowanego znaku będzie równy 0, a jego wyświetlanie będzie wyglądało tak – *WriteRd(0);*.

Na zakończenie jeszcze trzy przydatne procedury. Pierwszą zamieszczono na **listingu 10**. Wyświetla ona zawartość 8-bitowej liczby w postaci dziesiętnej na trzech pozycjach LCD. Jeżeli liczba setek wynosi 0, to procedura wyświetla spację. Można też dodać taki warunek dla pozycji dziesiątek. Do liczby jednostek, dziesiątek i setek jest dodawane 0x30, żeby wyświetlany znak był znakiem ASCII. Jeżeli istnieje potrzeba wyświetlenia liczby z zakresu 0...99, to procedurę można znacznie uprościć. Pokazano to na **listingu 11**.

W wielu wypadkach, głównie w czasie testowania programów, jest wygodnie wyświetlać zawartość 8-bitowej zmiennej jako liczbę szesnastkową. Realizuje to procedura z **listingu 12**.

Listing 5. Pozycjonowanie początku tekstu na ekranie
```
//funkcja ustalania początku wyświetlania
//x - położenie w wierszu od 1 do 16
//y - wiersz górny 1, dolny 2
void PosLcd(char x, char y) {
  if(x<1||x>16)           //zabezpieczenie przed
  return;                 //błędnymi współrzędnymi
if(y!=1&&y!=2) return;
  --x;                    //bo adresowanie od 0
  if(y==2)
  x=x+0x40;               //adres RAM drugiego wiersza
  RS=0;                   //zapisanie instrukcji
  WriteLcd(x|0x80);       //zapisz adres CG RAM Address set
}
```

Listing 6. Wyświetlanie napisów na ekranie wyświetlacza
```
void DispLcd(const char * s) {
  while(*s)               //czekaj na znak końca ciągu (bajt 0x00)
  WriteRd(*s++);          //zapisz kolejny znak i zwiększ adres w tablicy
}
```

Listing 7. Wyświetlanie napisów w 2 kolumnach
```
  PosLcd(1,1)                       //pozycja na wyświetlaczu
  DispLcd(„      Kurs C      „);    //wyświetlenie napisu
  PosLcd(1,2);
  DispLcd(„ uC PIC18F2320  „);
```

Listing 8. Definiowanie wzorca własnego znaku
```
//zdefiniowanie znaku stopnia
void DefLcdChar(void) {
  WriteRi(0x40);                    //adres 00 w pamięci CG RAM
  WriteRd(0x06);
  WriteRd(0x09);
  WriteRd(0x09);
  WriteRd(0x06);
  WriteRd(0x00);
  WriteRd(0x00);
  WriteRd(0x00);
  WriteRd(0x00);
}
```

Listing 9. Wyświetlenie 8-bitowej liczby w postaci 3 cyfr dziesiętnych
```
void DispDec(unsigned char dana){
  char lj,ld,ls;
  ls=dana/100;                      //liczna setek
  ld=(dana-(ls*100))/10;            //liczba dziesiątek
  lj=(dana-(ls*100)-(ld*10));       //liczba jedności
  if(ls>0)                          //jeżeli liczba setek jest zerowa, to
                                    //wyświetl spację
  WriteRd(ls+0x30);                 //dodanie 0x30 – korekcja do znaku ASCII
  else WriteRd(0x20);
  WriteRd(ld+0x30);
  WriteRd(lj+0x30);
}
```

Listing 10. Wyświetlenie liczby dziesiętnej w postaci 2 cyfr dziesiętnych
```
void DispDec(unsigned char dana) {
  WriteRd((val/10)+0x30);           //liczba dziesiątek
  Write Rd((val%10)+0x30);          //liczba jednostek
}
```

Listing 11. Wyświetlanie liczby 8-bitowej w postaci hexadecymalnej
```
void DispHex(unsigned char dana,char x, char y) {
  unsigned char hex;
  PosLcd(x,y);                      //pozycja
  hex=dana&0xf0;                    //starsza czesc
  hex=hex>>4;
  if(hex<10)
  hex=hex+0x30;
  else
  hex=hex+0x37;
  WriteRd(hex);
  hex=dana&0x0f;
  if(hex<10) hex=hex+0x30;
  else hex=hex+0x37;
  WriteRd(hex);
  return;
}
```

Listing 12. Wyświetlenie liczby 16-bitowej heksadecymalnie
```
Void DispHex16(unsigned int dana) {
  DispHex(dana>>16);
  DispHex(dana);
}
```

Podsumowanie

W poprzednim i tym odcinku kursu poznaliśmy obsługę podstawowych elementów interfejsu użytkownika, to jest klawiszy, enkodera, wyświetlaczy LED i LCD. W następnej kolejności zajmiemy się elementem coraz częściej wykorzystywanym do komunikacji z użytkownikiem, to jest graficznym wyświetlaczem LCD.

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