C dla mikrokontrolerów 8051

część 2

Drugą część kursu poświęcimy omówieniu przykładowego programu, który umożliwia wyświetlanie informacji na standardowym alfanumerycznym wyświetlaczu LCD, sterowanym poprzez interfejs 4-bitowy.

Sterowanie wyświetlaczem LCD w trybie 4-bitowym

Zaczynamy od bardzo użytecznego programu. Przyda się on nam w przyszłości, dzięki czemu czas poświęcony na naukę nie będzie zmarnowany. Analizując kod poszczególnych fragmentów programu (kod źródłowy przedstawiono na **list. 1**) rozłożymy go na kawałeczki. To najlepsza droga do poznania całego programu.

Na początku znajdują się deklaracje. Wybieramy model pamięci (*#pragma SMALL*), dołączamy właściwą definicję rejestrów mikroprocesora (*#include <reg51.h>*), następnie określamy, które bity sterują wyświetlaczem (*sbit*), jakie definicje znaków zapiszemy do wyświetlacza itp. Zwróćmy uwagę na trochę odmienny, niż w innych językach programowania, opis bitu portu. Separatorem pomiędzy nazwą portu, a numerem bitu nie jest znak kropki, lecz umowny symbol potęgi. Słowo kluczowe ***sbit*** umieszcza naszą definicję w rejestrze funkcji specjalnych SFR, w obszarze adresowania bitowego.

Przyjrzyjmy się zawartości tablicy z definicjami znaków użytkownika, które będą wprowadzane do pamięci CG-ROM (*Character-Graphics ROM*) sterownika wyświetlacza. Zauważmy, że jej deklaracja zawiera wiele istotnych informacji dla naszego programu, praktycznie w jednej linijce:

- określony zostaje typ elementów tablicy, w tym przypadku ***char*** (czyli element jednobajtowy),
- tablica zostaje umieszczona w pamięci programu mikrokontrolera (słowo ***code***),
- tablica otrzymuje nazwę symboliczną (***CGRom***) i określona zostaje liczba jej elementów (**65**),
- elementom tablicy nadawana jest wartość stała.

Pierwszą z funkcji, którą napotkamy analizując program, jest *Delay*. Jest to zwykła pętla, która absorbuje mikrokontroler na wielokrotność około 1 milisekundy. Tak stanie się tylko wtedy, gdy użyjemy rezonatora kwarcowego 7,3728MHz. Jeśli stosujemy inny kwarc, procedura wymaga zmiany. Zmieni się bowiem czas potrzebny na wykonanie pętli. Słowo ***void*** przed definicją funkcji oznacza, że funkcja nie zwraca żadnych wartości jako rezultatu działania. Tutaj wyjaśnienie: funkcja w języku C może zwracać tylko jedną wartość. Nie może zwrócić ich kilku tak, jak procedura języka Pascal poprzez *var*. Jeśli zachodzi potrzeba, aby funkcja zwracała więcej niż jeden wynik działania, można to zrobić na przykład przekazując, jako argument funkcji wskaźniki do zmiennych. Wówczas obliczenia wykonywane są bezpośrednio na zmiennych źródłowych. Działania wewnątrz funkcji, o ile nie są wykonywane na wskaźnikach, wykonywane są na kopiach argumentów przekazywanych w wywołaniu funkcji i dotyczą wyłącznie zmiennych lokalnych zawartych pomiędzy nawiasami klamrowymi {}, oznaczającymi jej początek i koniec.

Parametr ***unsigned int k*** jest to po prostu liczba milisekund do „odczekania". Słowo ***unsigned*** oznacza, że liczba **k** jest dwubajtową liczbą bez znaku, czyli przyjmuje wartości tylko dodatnie. Ciało funkcji zawiera również deklaracje zmiennych pomocniczych: ***j*** oraz ***k***. Posłużą one do budowy pętli ***for***. Interesujący jest w języku C jej zapis: ***for (j = 0; j < k; j++)***, co oznacza:

- wstaw do zmiennej *j* liczbę *0* (*j = 0;*);
- dopóki *j* jest mniejsze od *k* (*j < k;*) zwiększaj wartość *j* o *1* (*j++*), wykonuj działanie opisane za nawiasem (w tym przypadku jest to następna pętla *for*).

Zwróćmy uwagę, że o ile w RC-51, przed deklaracją typu *int* musimy użyć słowa *unsigned*, gdy ma to być liczba bez znaku, o tyle słowo to jest zbędne w przypadku typu *char*. Domyślnie kompilator zakłada, że chodzi o liczbę *unsigned char*. Uwaga: jest to cecha kompilatora RC-51 firmy Raisonance. Używając innego można się napotkać odmienne rozwiązania (np. firma Keil stosuje typ *uchar*). Oczywiście, można przed typem *char* dopisać słowo *unsigned*. Wówczas jest to zgodne ze specyfikacją ANSI C i prawdopodobnie będzie działać tak samo, bez względu na typ użytego kompilatora.

Po *Delay* występuje funkcja ***WriteByteToLcd***. Tak jak *Delay*, jest to funkcja typu *void*. Jej zadaniem jest zapisanie jednobajtowej liczby ***X*** do rejestru wyświetlacza. Funkcja dokonuje podziału bajtu na połówki i zapisuje je w bezpieczny sposób - wykorzystując tylko bity *b4..b7* i nie uszkadzając zawartości *b0..b3* - do portu PORT. Prześledźmy ten proces.

Dzięki funkcji sumy bitowej OR cztery najstarsze bity portu PORT, w tym przypadku zadeklarowanego jako P2, są ustawiane na „1". Zapis *PORT |= 0xF0* można bowiem przekształcić na równoważny mu *PORT=PORT | 0xF0*. Właściwy zapis zmiennej do portu następuje dzięki funkcji AND, jednak po uprzednim ustawieniu dolnej połówki zapisywanego bajtu na wartość „1" (X | 0x0F). Zapis:

```
PORT &= (X | 0x0F);
```

można rozłożyć na następującą sekwencję działań:

```
temp = X | 0x0F; //mlodsze 4 bity
     //X przyjmują wartość "1"
PORT = PORT & temp;//mlodsze 4 bity
//portu PORT pozostają niezmienione
//górne przyjmują wartość X
```

Nawias w wyrażeniu `PORT &= (X | 0x0F)` jest konieczny, ponieważ suma bitowa OR musi być wykonana przed iloczynem zapisującym połówkę bajtu do portu mikrokontrolera. Identycznie postępujemy z dolną połówką bajtu z tym, że wykonywane jest przesunięcie w lewo (o 4 pozycje) bitów zmiennej *X* (*X <<= 4*, to znaczy *X = X << 4*).

Funkcja podczas zapisywania danych do LCD nie sprawdza stanu flagi *busy* wyświetlacza zakładając, że po 1 milisekundzie wszystkie operacje wykonywane przez kontroler wyświetlacza zostaną zakończone. Wy-

wołanie *Delay(1)* wprowadza konieczne opóźnienie.

Kolejnymi są funkcje o nieco przydługich nazwach ***WriteToLcdCtrlRegister*** i ***LcdWrite***. Ich zadaniem jest zapisywanie parametru *X* do pamięci sterownika LCD, co wymaga wygenerowania odpowiedniej kombinacji sygnałów sterujących. Obie funkcje korzystają z omówionej wcześniej funkcji *WriteByteToLcd*. Ponieważ określiliśmy w deklaracjach, że zmienne *LcdReg*, *LcdRead* i *LcdEnable* są bitami portu wyjściowego mikrokontrolera, podstawienie wartości „1" lub „0" odpowiada rozkazom asemblera *SETB* i *CLR*. Prawidłowy (i szczególnie użyteczny, gdy mamy do czynienia z dużą liczbą zmiennych) jest również zapis ***LcdReg = LcdRead = 0;***. Kompilator podzieli go na pojedyncze operacje *SETB* i *CLR*. Zapis w takiej postaci nie ma więc wpływu na wielkość generowanego kodu wynikowego, poprawia jednak czytelność programu.

Dalej w listingu występują linie z tymi samymi funkcjami. Nie omawianą dotychczas spotkamy dopiero w *GotoXY*. Jej rolą jest obliczenie i ustawienie właściwego dla pozycji kursora *x*, *y* adresu zapisu bajtów do wyświetlacza. Wykorzystuje ona polecenie ***switch*** do rozpatrzenia poszczególnych możliwych wartości *y* i wyboru odpowiedniej akcji:

```
switch (y)
{
 case 0:
  x += 0x80;
  break;
 case 1:
  x += 0xC0;
  break;
 case 2:
  x += 0x94;
  break;
 case 3:
  x += 0xD4;
}
```

Z poleceniem *switch* ściśle są związane słowa kluczowe ***case*** i ***break***. *Case* ma znaczenie takie, jak etykieta danego warunku, a *break* przerywa rozpatrywanie warunków. Jeśli nie użyjemy polecenia *break*, wówczas nastąpi przejście do następnego warunku i jego rozpatrzenie. W naszym przypadku jest to tylko strata czasu. Jedynym zadaniem funkcji *GotoXY* jest bowiem zwiększenie wartości *x* tak, aby wskazywała pożądany adres - nie ma potrzeby dalszego analizowania zmiennej *y*. Jeśli nasz *y* będzie miał wartość 0, wówczas do wartości *x* zostanie dodana liczba 80H. Jeśli 1 - to C0H, jeśli 2 to 94H i tak dalej.

Kilka słów komentarza. Przechodzenie od jednego przypadku do dru-

List. 1

```
/*********************************************
Obsluga wyswietlacza LCD w trybie 4 bity
RC-51 Raisonance
Przykladowy program w jezyku C dla EP
------------------------
jacek.bogusz@ep.com.pl
*********************************************/
// wybór modelu pamieci
#pragma SMALL
// dolaczenie definicji rejestrów '51
#include <reg51.h>

// definicje znaków specjalnych dla wyswietlacza LCD
char code CGRom[65] = {
    0xAA,0x55,0xAA,0x55,0xAA,0x55,0xAA,0x55,              // "kratka"         0x00
    0xC0,0xC0,0xFF,0xF1,0xF1,0xF1,0xFF,0xC0,              // pusty kwadrat    0x01
    0xC0,0xC0,0xFF,0xFF,0xFF,0xFF,0xFF,0xC0,              // kwadrat zacz.    0x02
    0xE0,0xFF,0xFF,0xFF,0xE0,0xE0,0xE0,0xE0,              // linia            0x03
    0xFF,0xFF,0xF9,0xF3,0xE7,0xF3,0xF9,0xFF,              // znak w lewo      0x04
    0xFF,0xFF,0xF3,0xF9,0xFC,0xF9,0xF3,0xFF,              // znak w prawo     0x05
    0xFF,0xFF,0xFB,0xF1,0xE4,0xEE,0xFF,0xFF,              // znak w góre      0x06
    0xFF,0xFF,0xFF,0xEE,0xE4,0xF1,0xFB,0xFF,              // znak w dól       0x07
    0x00};

// port,do którego podlaczono wyswietlacz LCD
#definePORT    P2
// 4 najstarsze bity, to bity danych, 4 mlodsze moga byc w dowolny sposób skonfigurowane
// jako sterujace
// po zmianie definicji moga to byc równiez 2 rózne porty,jednak w programie przykladowym
// zakladam wykorzystanie tylko jednego z portów mikrokontrolera

// bity sterujace LCD
sbit LcdEnable =  PORT^0;
sbit LcdRead =   PORT^3;
sbit LcdReg =    PORT^1;

// opóznienie okolo k*1 milisekundy (dla kwarcu 7,3728MHz)
void Delay (unsigned int k)
{
 unsigned int i,j;
 for (j = 0; j < k; j++)
   for (i = 0; i <= 296; i++);
}

// zapis bajtu do lcd, osobna procedura dla zmniejszenia objetosci kodu wynikowego
void WriteByteToLcd(char X)
{
 LcdEnable = 1;
 PORT |= 0xF0;          // ustawienie górnej polówki portu PORT na "1"
 PORT &= (X | 0x0F);    // "bezkolizyjny" zapis 1-szej polówki bajtu (przez funkcje logiczna)
 LcdEnable = 0;         // zapis do wyswietlacza (opadajace zbocze sygnalu E)

 LcdEnable = 1;         // zapis 2-giej polówki bajtu
 X <<= 4;               // przesuniecie 4x w lewo
 PORT |= 0xF0;          // ustawienie górnej polówki portu PORT na "1"
 PORT &= (X | 0x0F);    // zapis 2-giej polówki bajtu i maskowanie 4 mlodszych bitów
 LcdEnable = 0;         // opadajace zbocze E - zapis do LCD

 Delay(1);
}

// zapis bajtu do rejestru kontrolnego LCD
void WriteToLcdCtrlRegister(char X)
{
 LcdReg = LcdRead = 0; // ustawienie sygnalów sterujacych
 WriteByteToLcd(X);
}


// zapis bajtu do pamieci obrazu
void LcdWrite(char X)
{
 LcdReg = 1;            // ustawienie sygnalów sterujacych
 LcdRead = 0;
 LcdEnable = 1;
 WriteByteToLcd(X);
}


// czyszczenie ekranu LCD
void LcdClrScr(void)
{
 WriteToLcdCtrlRegister(0x01);
}


// inicjalizacja wyswietlacza LCD w trybie 4 bity
void LcdInitialize(void)
{
 char i;

 Delay(15);
 P0 = 0x0F;             // wyzerowanie linii LcdReg,LcdRead,LcdEnable
```

```
 for (i = 0; i<3; i++)
 {
   LcdEnable = 1;       // impuls na E
   PORT &= 0x3F;        // ustawienie wartosci inicjujacej
   LcdEnable = 0;
   Delay(5);
 }
 LcdEnable = 1;         // wpisanie wartosci 2 do rejestru kontrolnego
 PORT &= 0x2F;          // tylko "górne" 4 bity
 LcdEnable = 0;
 Delay(1);
 WriteToLcdCtrlRegister(0x28);          // interfejs 4 bity,znaki 5x7
 WriteToLcdCtrlRegister(0x08);          // wylaczenie LCD
 WriteToLcdCtrlRegister(0x01);          // kasowanie ekranu, powrót do pozycji home
 WriteToLcdCtrlRegister(0x06);          // przesuwanie kursora z inkrementacja
 WriteToLcdCtrlRegister(0x0C);          // zalaczenie wyswietlacza
}


// ustawia kursor na współrzednych x,y
void GotoXY(char x, char y)
{
 switch (y)        // obliczenie o ile nalezy zwiekszyc x w zaleznosci od wartosci y
 {
   case 0:
     x += 0x80;     // x = x + 0x80
     break;
   case 1:
     x += 0xC0;     // x = x + 0xC0
     break;
   case 2:
     x += 0x94;     // x = x + 0x94
     break;
   case 3:
     x += 0xD4;     // x = x + 0xD4
     break;
 }
 WriteToLcdCtrlRegister(x);
}


// wyswietla tekst na współrzednych x, y
void WriteTextXY(char x, char y, char *S)
{
 while (*S)           // petla dziala dotad, az napotkany zostanie znak konca lancuch (/0)
 {
   GotoXY(x, y);      // wyliczenie adresu dla znaku
   LcdWrite(*S);      // wyswietlenie pojedynczego znaku
   x++;               // nastepna pozycja x na ekranie
   S++;               // nastepna pozycja wskaznika, tzn. nastepny znak napisu
   if (x > 19)        // jesli x>19 to nastepna linia
   {
     x = 0;
     y++;
   }
 }
}
// wyświetla tekst rozpoczynając od aktualnego położenia kursora
void WriteText(char *S)
{
 while (*S)           // petla dziala dotad,az napotkany zostanie znak konca lancucha (/0)
 {
   LcdWrite(*S);      // wyswietlenie pojedynczego znaku
   S++;
 }
}

// definiowanie znaków z tablicy CGRom
void DefineSpecialCharacters(char *ptr)
{
 WriteToLcdCtrlRegister(0x40);          // ustawienie trybu definicji
 while (*ptr != 0)     // petla wykonywana do napotkania znaku konca tablicy
 {
   LcdWrite(*ptr);     // zapis znaku do lcd cgram
   ptr++;              // nastepna pozycja tablicy (wskaznika)
 }
 WriteToLcdCtrlRegister(0x80);          // przelaczenie do trybu wyswietlania
}

// program główny
void main(void)
{
 char ix = 1, iy = 1, x, y, i = 0;

 LcdInitialize();
 DefineSpecialCharacters(&CGRom);
 while (1)
 {
   WriteTextXY(x, y, " ");
   if (ix == 1) x++; else x--;
   if (iy == 1) y++; else y--;
   if (x == 19) ix = 0;
   if (x == 0) ix = 1;
   if (y == 3) iy = 0;
   if (y == 0) iy = 1;
   WriteTextXY(x, y, 0x01);
   Delay(50);
 }
}
```

giego budzi trochę wątpliwości. Niewątpliwie zaletą jest to, że można definiować wiele różnych warunków dla jednej akcji. Jednak taka konstrukcja programu, która umożliwia przechodzenie od warunku do warunku, jest bardzo podatna na „rozsypanie" się podczas jej modyfikacji. Z wyjątkiem wielu etykiet dla pojedynczej akcji, przechodzenie przez przypadki, powinno być stosowane bardzo oszczędnie i zawsze opatrzone komentarzem. Do dobrego stylu programowania należy wstawianie *break* po ostatniej instrukcji ostatniego przypadku, mimo że nie jest to konieczne.

Funkcja ***WriteTextXY*** wykorzystuje wskaźnik. Jest to wskaźnik do elementu typu *char*. Jego definicję możemy bardzo łatwo odróżnić po symbolu *. Zauważmy, że do funkcji jako parametr nie jest przekazywany tekst do wyświetlenia, a jedynie wskazanie (adres) do miejsca w pamięci RAM (lub ROM), gdzie ten tekst został umieszczony. Wskaźnik ma rozmiar tylko dwóch bajtów, nie tak jak tekst, który może zająć znacznie więcej komórek pamięci. Operacja *S++;* przesuwa wskazanie na następny znak w łańcuchu. Kompilator sam dba o to, aby zwiększanie wskaźnika powodowało wskazanie na następny znak. Nie musisz przejmować się liczbą bajtów inkrementacji jeżeli wskaźnik ma przypisane wskazanie do określonego typu elementu. Pętla ***while*** kończy się, gdy wskaźnik *S* pokaże znak o kodzie 0. Znak ten umieszczany jest przez kompilator zawsze na końcu tekstu. Równocześnie z *S* zwiększana jest współrzędna *x*. Jeśli wartość *x* przekroczy maksymalną liczbę znaków w wierszu, następuje zwiększenie *y* i przejście do następnej linii na wyświetlaczu LCD.

Podobnie jest realizowana funkcja ***DefineSpecialCharacters***, ale jest ona nieco prostsza, bo nie oblicza żadnych wartości x i y, a jedynie przesuwa wskazania na następny bajt tablicy. Również i tutaj w pętli *while* napisać można warunek *while (*ptr*), ponieważ pojawienie się znaku o kodzie 0, *while* traktuje jako warunek końca (0 jest równoważne *false*). Jednak dla większej czytelności programu i wyraźnego zaakcentowania w jaki sposób kończy się tablica definicji, został użyty zapis *while (*ptr != 0)*.

W funkcji ***DefineSpecialCharacters*** znaleźć możemy jeszcze dwie nowe konstrukcje, których nie używaliśmy wcześniej. W wywołaniu funkcji ***DefineSpecialCharacters*** znajdującym się w programie

głównym, znaleźć możemy pewną konstrukcję, której nie używaliśmy wcześniej. Jest to przypisanie wskaźnikowi ptr adresu tablicy *CGRom*. Jednoargumentowy operator **&** tym razem nie oznacza iloczynu - podaje adres tablicy *CGRom* w pamięci mikrokontrolera

Jak już wspomniałem przy okazji omawiania przerwań, program główny w C jest to funkcja o nazwie ***main***. W programach pisanych dla mikrokontrolerów najczęściej jest ona typu ***void*** z pustą listą parametrów (również *void*). Należy jednak pamiętać, że wykonywanie programu napisanego dla mikrokontrolera nigdy nie może się skończyć. Nawet jeśli zrobił on już swoje i nie ma żadnych dalszych zadań do realizacji, to funkcję *main* należy zakończyć pętlą nieskończoną taką, jak: *while(1)* albo *for(;;)*. Mikrokontroler nie posiada bowiem żadnego systemu operacyjnego takiego jak DOS, który po zakończeniu pracy programu przejmie kontrolę.

Pliki nagłówkowe (*.h)

Rozważmy teraz pewną możliwość. Mamy już napisany program w języku C, mamy w nim pewne funkcje, o których wiadomo, że przydadzą się nam również w wielu innych przypadkach. Chociażby nasz pierwszy program sterujący wyświetlaczem. Z całą pewnością przyda się na również w innych aplikacjach.

Pojawia się więc pytanie: czy można z takich programów stworzyć bibliotekę, którą będzie można dołączyć do własnego programu i używać zawsze wtedy, gdy jest to potrzebne? A co z modyfikacją pewnych parametrów procedur? Czy zawsze trzeba mieć dostęp do kodu źródłowego programu i szukać, nierzadko wśród kilkuset linii programu tego parametru, który ma być zmieniony? Odpowiedzią na tak postawione pytania są tak zwane pliki nagłówkowe (*header files*), które umożliwiają podzielenie programu na mniejsze fragmenty i mogą zawierać definicje stałych oraz zmiennych używanych zarówno przez program główny, jak i przez biblioteki funkcji. W języku C, zbiory nagłówkowe, wyróżnia rozszerzenie *.h* nazwy (na przykład *lcd4bit.h*). Tak może wyglądać plik nagłówkowy utworzony dla biblioteki funkcji wyświetlacza LCD z poprzedniego przykładu.

```
// port, do którego podłączono
// wyświetlacz LCD
#define PORT   P2
// bity sterujące LCD
sbit LcdEnable =  PORT^0;
sbit LcdRead =   PORT^3;
sbit LcdReg =   PORT^2;

// opóźnienie około k*1 milisekundy
// dla kwarcu 7,3728MHz
void Delay (unsigned int k);
// zapis bajtu do lcd
void WriteByteToLcd(char X);
// zapis bajtu do rejestru
// kontrolnego LCD
void WriteToLcdCtrlRegister(char X);
// zapis bajtu do pamięci obrazu
void LcdWrite(char X);
// czyszczenie ekranu LCD
void LcdClrScr(void);
// inicjalizacja wyświetlacza LCD
// w trybie 4 bity
void LcdInitialize(void);
// ustawia kursor na współrzędnych
// x,y
void GotoXY(char x, char y);
// wyświetla tekst na współrzędnych
// x, y
void WriteTextXY(char x, char y,
char *S)
// wyświetla tekst od miejsca,
// w którym znajduje się kursor
void WriteText(char *S);
// definiowanie znaków z tablicy
// wskazywanej przez ptr
void DefineSpecialCharacters(char
*ptr);
```

Jak widać, jest to bardzo krótki zbiór tekstowy zawierający podstawowe definicje zmiennych i funkcji. Zbiór ten dołączamy do programu głównego za pomocą dyrektywy ***#include***, podobnie jak postępowaliśmy z definicją rejestrów mikrokontrolera (notabene też jest to zbiór nagłówkowy). W takiej sytuacji, program źródłowy naszej biblioteki nie może zawierać funkcji *main()*. Zostanie ona zdefiniowana w programie głównym.

Tworząc pliki nagłówkowe trzeba zachować ostrożność. Można bowiem samemu stworzyć pewien „bałagan" polegający na tym, że maleńki nawet program będzie miał dostęp do wielu danych, których w praktyce nie potrzebuje. Będzie to rzutować również na rozmiar kodu wynikowego oraz utrudni zachowanie porządku w deklaracjach zmiennych i funkcji. Więcej informacji o plikach nagłówkowych przedstawimy w następnym przykładzie programu w języku C (za miesiąc w EP). Aby wykorzystać bowiem mechanizm tworzenia plików nagłówkowych, musimy się nauczyć jak tworzyć *project files* i do czego one służą.

Jacek Bogusz, AVT
jacek.bogusz@ep.com.pl

Dodatkowe informacje

Ewaluacyjną wersję pakietu firmy Raisonance oraz źródo programu prezentowanego w artykule zamieściliśmy na CD-EP7/2002B.