C dla mikrokontrolerów 8051

część 4

O sposobie, w jaki przerwania są obsługiwane w języku C, pisałem już przy okazji omawiania podstaw programowania. Dziś wykorzystamy programowanie przerwań do budowy prostego licznika. Wykorzystamy w nim przerwanie generowane przez Timer 1 do obsługi wyświetlacza LED oraz przerwanie generowane przez opadające zbocze sygnału na wejściu INT0 do zliczania impulsów zegarowych. W programie wykorzystamy też wskaźniki i ich arytmetykę - będzie okazja co nieco się nauczyć.

Budujemy licznik, czyli język C i przerwania

Licznik prezentowany w artykule jako przykład zbudowałem, korzystając z płytki AVR Starter Kit oraz kawałka płytki uniwersalnej. Oczywiście, jeśli używasz innego zestawu eksperymentalnego - nie jest to żadną przeszkodą. Prawdopodobnie będziesz tylko musiał zbudować sobie wyświetlacz LED.

Płytka wyświetlacza LED

Schemat połączeń wyświetlacza LED pokazano na **rys. 1**. Niestety - jeżeli chcesz eksperymentować z tym przykładem programowania, musisz sobie taki układ zbudować. Moim zdaniem przyda on ci się nie tylko do eksperymentów, ale również można go użyć w dowolnym innym układzie wykorzystującym wyświetlacze LED. Trzeba jednak uważać - obsługa wyświetlacza oparta jest o przerwanie generowane przez Timer 1, dlatego też nie polecam takiego układu na przykład do częstościomierza. Chyba że zgodzisz się na wyłączenie wyświetlania w czasie pomiaru częstotliwości. Zaowocuje to migotaniem przy pomiarze przebiegów o niskiej częstotliwości. Jednak tam, gdzie mikrokontroler nie jest zbyt mocno obciążony i gdzie nie ma ścisłych zależności czasowych - śmiało możesz tego układu użyć. Z powodzeniem na przykład stosuję go w układzie termometru cyfrowego, mimo różnic poglądów na temat zawieszania transmisji z DS1820 na czas obsługi przerwań - nie mam z tym kłopotu w mojej aplikacji.

Do konstrukcji wyświetlacza użyłem rejestrów przesuwających 74HCT595. Zrobiłem tak z dwóch powodów. Po pierwsze, cena tych układów jest bardzo niska, a obciążalność ich wyjść w stanie niskim jest wystarczająca do zasilania typowego wyświetlacza LED. Po drugie zaś, układ nie wyprowadza informacji na wyjścia do momentu pojawienia się osobnego impulsu zegarowego, który ją tam przepisze. Nie ma więc efektu migotania cyfr w czasie wpisywania danych do rejestrów. Połączyłem szeregowo dwa układy 74HCT595, tworząc w ten sposób rejestr 16-bitowy. Jako pierwszy w szeregu znajduje się rejestr segmentów cyfr, jako drugi rejestr załączający poszczególne cyfry. Wejście szeregowe danych taktowane jest sygnałem o częstotliwości 4,8 kHz, natomiast cyfry przełączane są z częstotliwością około 300 Hz.

Użyłem wyświetlaczy LED ze wspólną anodą. Zasilanie anod załączane jest przez tranzystory MOS z kanałem typu P (BS250). Ich stosowanie jest bardzo wygodne, ponieważ nie wymagają żadnych dodatkowych elementów, takich jak na przykład rezystory. Wartości rezystorów podłączonych do poszczególnych segmentów wyświetlacza musisz dobrać sobie do posiadanych cyfr. Numery wyprowadzeń wyświetlacza LED potraktuj jako orientacyjne. Istotne są literowe oznaczenia segmentów. Na wejściach rejestrów szeregowych znajdują się rezystory *pull-up*, tak aby można było wyświetlacz podłączyć do dowolnego z portów mikrokontrolera.

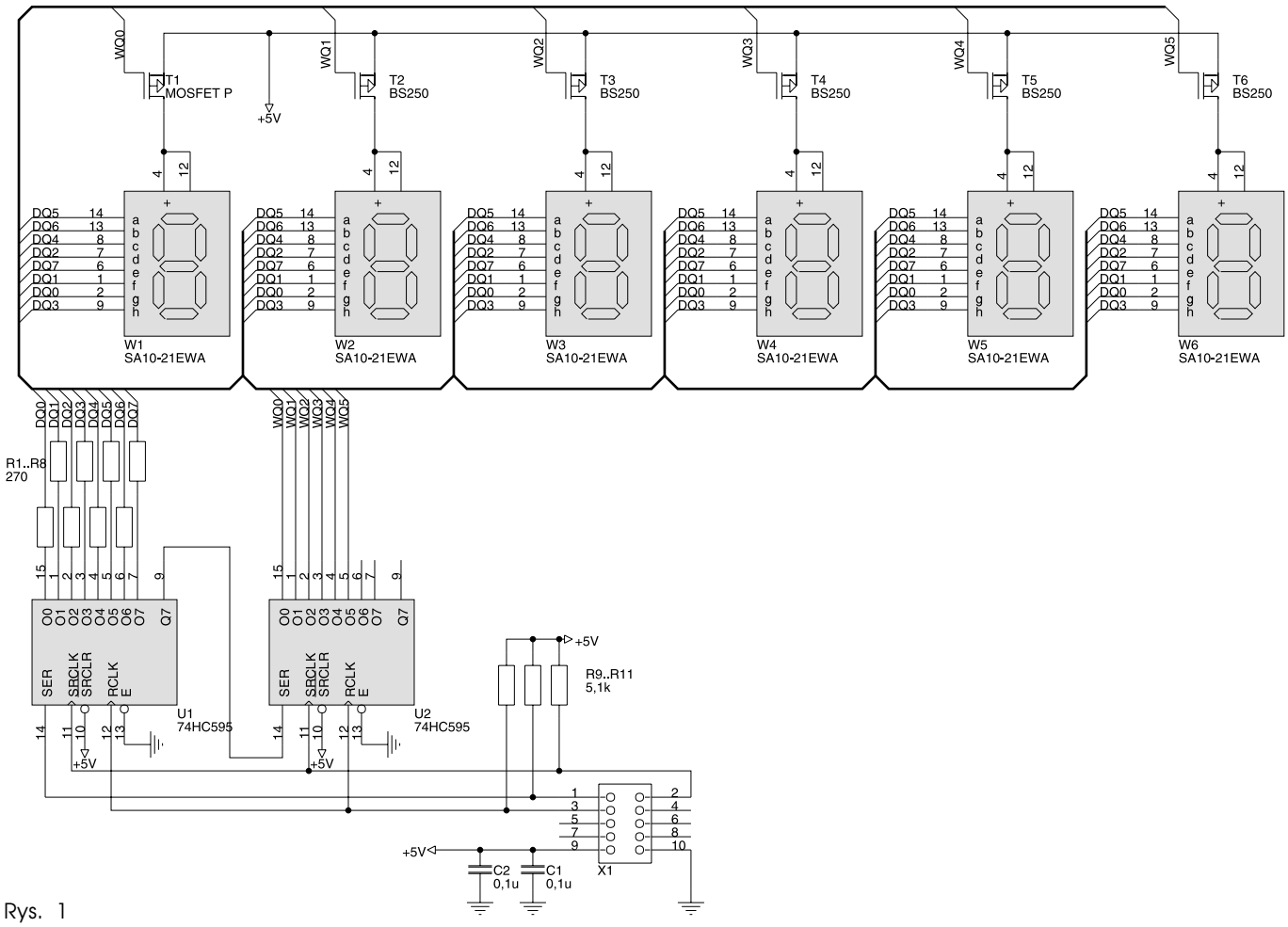
Wyświetlacz do sterowania wymaga trzech linii - jednej danych i dwóch zegarowych. Ja wykorzystałem P1.1, P1.2 i P1.3 Oczywiście, zmieniając program, możesz użyć dowolnych innych. Również wykonując drobne modyfikacje w programie, można podłączyć do 8 wyświetlaczy LED o wspólnej anodzie. W skrócie funkcjonowanie wyświetlacza wygląda następująco: dane za pomocą opadającego zbocza sygnału zegarowego podawanego na wyprowadzenie 11 (SRCLK) wpisywane są z wejścia szeregowego na wyprowadzeniu 14 (SER) do wewnętrznego rejestru. Mikrokontroler przesyła pełne słowo 16-bitowe tak, aby działały oba układy rejestrów. Następnie, po wpisaniu 16 bitów, na wyprowadzenie 12 (RCLK) podawany jest impuls zegarowy, którego opadające zbocze powoduje przepisanie danych z wewnętrznego szeregowo-równoległego rejestru do wyjściowego rejestru typu zatrzask.

Program napisany został tak, że w danym momencie świeci tylko jedna cyfra. Jeśli przełączanie cyfr będzie wystarczająco szybkie, ludzkie oko tego nie zauważy. Jest to typ wyświetlania zwany multipleksowanym (wyświetlanie dynamiczne). Charakteryzuje się on małym poborem prądu - w danym momencie zasilana jest tylko jedna cyfra. Tyle na temat zasady działania, zajmijmy się teraz programem.

Mikrokontroler jako licznik

Program pokazano na **list. 1**. Licznik wykorzystuje dwa przerwania. Pierwsze, zewnętrzne, powodowane przez opadające zbocze napięcia na wejściu INT0, używane jest do zwiększania wartości licznika. Drugie - wewnętrzne, generowane cyklicznie - pochodzące od Timera 1, przepisuje stan bufora *display* do rejestrów wyświetlacza.

Program rozpoczyna się od deklaracji. Linia danych wyświetlacza zadeklarowana zostaje jako P1^0, linia zegara szeregowego jako P1^1, linia zegara wyjściowego rejestru latch jako P1^2. Oprócz tego inicjujemy zmienną typu *unsigned int* zawierającą zliczane impulsy oraz stałą typu *char* z zawartością inicjującą rejestr TH1 Timera 1. Stała ta to pośrednio częstotliwość, z jaką wywoływane jest przerwanie obsługujące wyświetlacz LED. Dalej znajduje się uporządkowana w kolejności rosnącej (od 0 do 9) tablica określająca wygląd wyświetlanego znaku (*patterns*), tablica z kodami kolejności załączania cyfr (*digits*) oraz tablica - bufor



Rys. 1

wyświetlacza w RAM (*display*). Jak łatwo zauważyć, dwie pierwsze umieszczone są w obszarze pamięci ROM mikrokontrolera (słowo kluczowe *code*) i mają przypisane wartości. Trzecia znajduje się w obszarze RAM. Jest ona odwzorowaniem stanu wyświetlacza, abstrahując od numeru wyświetlanej aktualnie cyfry. Każda z tablic ma przypisany właściwy jej wskaźnik, czyli zmienną, która będzie wskazywać na element tablicy. W momencie zadeklarowania, każdy wskaźnik ustawiany jest na pierwszy element tablicy. Wyrażenie *Wskaźnik = &Tablica* powoduje przypisanie zmiennej *Wskaźnik* adresu, pod którym umieszczona jest *Tablica*. Wskaźniki to fantastyczne narzędzie języka C!

Funkcja **Translate** zamienia argument *x*, którym jest dwubajtowa liczba całkowita bez znaku, na odpowiadającą tej liczbie zawartość bufora *display*. Jednym słowem, zamienia liczbę na odpowiadający jej wygląd wyświetlacza LED. Metoda jest bardzo prosta, chociaż zapis początkowo może się wydać niezrozumiały. Po wywołaniu funkcji wyłączane są przerwania Timera 1. Zostało to zrobione w celu uniknięcia migotania wyświetlacza. Później, wskaźnikowi *TDisplay* przypisywane jest wskazanie na ostatni element bufora wyświetlacza. Od niego to rozpocznie się translacja na kody LED. Przebiega ona według następującego schematu: do wartości wskaźnika *TPatterns* dodaj resztę z dzielenia argumentu przez 10, a następnie skopiuj wskazywaną w wyniku działania stałą typu *char* z tablicy *patterns* pod adres wskazywany przez *TDisplay*. Podziel liczbę przez 10, przesuń wskazanie na następną pozycję w buforze wyświetlacza i powtórz operację dla następnej cyfry. I tak 6 razy - dla każdej z cyfr LED.

Jako swego rodzaju rozszerzenie funkcjonalności, umieszczono wygaszanie zer nieznaczących na początku cyfry. Pętla - rozpoczynając od początku bufora - sprawdza znak znajdujący się pod wskazanym przez wskaźnik bufora wyświetlacza adresem (*TDisplay)* i porównuje go ze znakiem na początku tablicy *digits*, to znaczy wzorcem „0“. Jeśli są to te same znaki, kod „0“ w buforze wyświetlacza zostaje zamieniony na 0xFF, co odpowiada całkowitemu wygaszeniu cyfry. Tak dzieje się aż do momentu napotkania wzorca różnego od wzorca „0“. Wówczas to instrukcja *break* (spotkaliśmy ją w konstrukcjach warunku *switch*) przerywa działanie pętli *for*. Po zakończeniu pętli załączane jest wyświetlanie - odpowiada mu zezwolenie na przyjmowanie przerwań Timera 1.

Dalej napotkamy procedury obsługi przerwań. Wyróżnia je słowo kluczowe *interrupt* umieszczone w nagłówku funkcji. Funkcja **IncrementCounter** obsługuje przerwanie zewnętrzne INT0. W zasadzie nie robi nic za wyjątkiem zwiększenia stanu zmiennej *counter* i wywołania funkcji *Translate*. Druga z nich to procedura obsługi przerwania Timera 1 - **DisplaySend**, zajmująca się konstrukcją i przesłaniem słowa do wyświetlacza LED. Przyjrzyjmy się dokładniej stosowanym w niej metodom.

Timer 1 pracuje w trybie 16-bitowym. Przerwanie zgłaszane jest przez timer w momencie przepełnienia, to znaczy zmiany stanu z 0xFFFF na 0x0000. Wówczas to wywoływana jest procedura obsługi przerwania. Cykl odliczania rozpoczyna się na

nowo od wartości 0 do 0xFFFF. Jeśli nie zdecydujemy inaczej, to od tego momentu do następnego przerwania upłynie czas 65536 cykli maszynowych (to jest 1/12 częstotliwości oscylatora). W przypadku mojego modelu byłby to czas około 100 milisekund - z całą pewnością byłby on powodem migotania cyfr, ponieważ jest zbyt długi. Można czas do wywołania przerwania skrócić, ustawiając na pożądaną wartość najpierw młodszy a później starszy bajt timera. Ponieważ nie zależy mi na bardzo dokładnym odmierzaniu czasu, zdecydowałem się na ustawienie (odświeżenie) tylko starszego bajtu.

Dalej zwiększane są wartości wskaźników *TDisplay* (wskazanie na cyfrę w buforze wyświetlacza) oraz *TDigits*, to znaczy wskaźnik do tablicy z kodami załączenia cyfr. Oba te wskaźniki zmieniane są synchronicznie co oznacza, że przesunięcie się na następną cyfrę musi powodować również zmianę kodu załączenia wyświetlacza.

Warunek *if (TDisplay == 0)* służy do zbadania, czy napotkano znak końca bufora w RAM. Jeśli tak, to wskaźniki są ustawiane ponownie na początek wskazywanych tablic.

Jako że kod załączenia musimy wysłać jako pierwszy, na początek zmiennej *x* przypisywana jest wartość kodu załączenia cyfry. Później przesuwana jest ona w lewo o 8 pozycji, a następnie sumowany jest z nią wzorzec znaku do wyświetlenia z bufora *display*. To są wszystkie operacje, które muszą być wykonane w celu poprawnej budowy słowa do sterowania wyświetlaczem. Teraz docieramy do pętli *for*, która ma za zadanie wysłanie wszystkich 16 bitów słowa do wyświetlacza.

Znajdująca się wewnątrz pętli operacja przesuwania w lewo zmiennej *x* ma na celu przeniesienie pojedynczego bitu słowa do flagi C, która to w następnym poleceniu (*dataline* = CY) wpływa na stan bitu portu - wyjścia danych. Impuls zegarowy, zmiana stanu *shiftline* z wysokiego na niski, kończy proces wysłania bitu. Pętla *for* powtarza operację 16 razy, dla wszystkich bitów zmiennej. Transmisję kończy przepisanie danych z wewnętrznego rejestru do rejestru wyjściowego poprzez zmianę stanu linii *latchline*. I to jest koniec obsługi przerwania Timera 1.

Program główny **main()** zawiera tylko proste ustawienia mikrokontrolera oraz, na samym początku, zapisanie znaku końca bufora wyświetlacza. Ustawiane są:

List. 1.

```
//================================================
// (C) Easy Soft 01/2002
// Raisonance RC-51, version 6.4.16
//================================================

// rezonator 7,3728MHz

#include <reg52.h>

//definicje linii sterujących wyświetaniem
sbit  dataline  = P1^0;
sbit  shiftline = P1^1;
sbit  latchline = P1^2;

//licznik impulsów
int counter = 0;

//wartość rejestru timera TH1, TL1 nie jest modyfikowane liczy zawsze od 0
const char interval = 0xF8;


/* tutaj wzorce cyfr
          d5
         ======
     d1 |      | d6
        | d0   |
         ======
     d7 |      | d4
        |      |
         ======
          d2                  */

char code patterns[10] = { 0x09,0xAF,0x1A,0x8A,0xAC,0xC8,0x48,0x8F,0x08,0x88 };
//przecinek = d3, wyłączenie cyfry = 0xFF
char *TPatterns = &patterns;


//tutaj kolejność załączania
char code digits[6] = { 0xFE,0xFD,0xFB,0xF7,0xEF,0xDF };
char *TDigits = &digits;


//bufor wyświetlacza w RAM
char data display[7];
char data *TDisplay = &display;


//zamiana zmiennej x na zawartość bufora do wyświetlenia
void Translate(unsigned int x)
{
   char temp;
   ET1 = 0;                    //wyłączenie na czas translacji wyświetlania
                               //(migotanie LED)
   TDisplay = &display + 5;    //zaczynamy translację od najmłodszej cyfry
   for (temp=0; temp<6; temp++)
   {
      *TDisplay = *(TPatterns + x % 10); //adres wzorca + (reszta z dzielenia
                                         //przez 10)
      TDisplay-;
      x /= 10;
   }
   TDisplay += 1;              //po poprzedniej pętli TDisplay jest o 1 za małe
   for (temp=0; temp<5; temp++)    //wygaszenie zer nieznaczących od pozycji 1..5
   {
      if (*TDisplay == *TPatterns) *TDisplay = 0xFF; else break;
                                  //jeśli napotkamy znak różny od 0,to koniec
      TDisplay++;
   }
   ET1 = 1;                       //załączenie wyświetlania
}


//procedura obsługi przerwania INT0
//zwiększanie licznika impulsów
void IncrementCounter(void) interrupt 0
{
   counter++;                     //zwiększenie licznika impulsów
   Translate(counter);            //zamiana counter na liczby do wyświetlenia
}
```

List. 1. (ciąg dalszy)

```
//procedura obsługi przerwania od timer'a 1
//wysłanie zmiennej 2-bajtowej do wyświetlacza - 1 znak z bufora display
void DisplaySend(void) interrupt 3
{
   char temp;
   unsigned int x;

   TH1 = interval;                  //odświeżenie zawartości timera 1
   TDisplay++;                      //następna pozycja do wyświetlenia
   TDigits++;
   if (*TDisplay == 0)              //jeśli osiągnięto koniec bufora,
   {                                //to wróć do początku
       TDisplay = &display;
       TDigits = &digits;
   }
   x = *TDigits;                    //x przyjmuje wartość liczby do wysłania
   x <<= 8;                         //składa się ona z bajtu wzorca cyfry
                                    //i bajtu kolejności załączenia
   x |= *TDisplay;

   for (temp = 0; temp<16; temp++)     //wysłanie cyfry poprzez przypisanie flagi C
                                       //do wyjścia danych
   {
       x <<= 1;
       dataline = CY;
       shiftline = 1;               //impuls na wyjściu zegara przesuwającego
       shiftline = 0;
   }
   latchline = 1;                   //przepisanie danych do wyjść rejestrów
   latchline = 0;
}

//program główny
void main(void)
{
   *(TDisplay + 6) = 0x00;          //tutaj kod końca danych
   Translate(counter);              //wyświetlenia 0.
   TMOD = 0x11;                     //oba timery jako 16 bitowe, kontrolowane
                                    //wewnętrznie
   TH1 = interval;                  //przerwanie wywoływane z częstotliw.około 75Hz
   ET1 = 1;                         //zezwolenie na przerwanie od timer'a 1
   TR1 = 1;                         //uruchomienie timer'a 1
   IT0 = 1;                         //opadające zbocze na INT0 wyzwala przerwanie
   EX0 = 1;                         //załączenie przerwania INT0
   EA = 1;                          //zezwolenie na przyjmowanie przerwań
   while (1);                       //oczekiwanie na przerwania
}
```

- rejestr TMOD na wartość 0x11, to znaczy oba Timery jako 16-bitowe, a impulsy pobierane są z wewnętrznego zegara,
- ustawiany jest starszy bajt licznika Timera 1,
- włączane są przerwania.

Program główny kończy pętla *while(1)*, w której mikrokontroler oczekuje na impulsy przychodzące na INT0 oraz zajmuje się obsługą wyświetlania.

Pewnym zaskoczeniem był dla mnie drobny fakt napotkany podczas testowania programu. Jego pierwowzorem był identycznie funkcjonujący program w języku asembler. Faktycznie nie przejmowałem się mocno jego optymalizacją, ale gdy napisałem program w C, mina mi zrzedła. Programy - mniej więcej równoważne w funkcjach - ten napisany w asemblerze zajmował 169 bajtów, a ten napisany w C - 119 bajtów. Stało się tak chyba z jednego powodu. Po pierwsze, w asemblerze dosyć trudno pisze się programy operujące na adresach. Takie programy są po prostu mało czytelne. W C nie ma z tym większego problemu. Można używać pewnych dróg na skróty. I co na to wszyscy twierdzący, że programy napisane w językach wysokiego poziomu zajmują dużo pamięci? Oczywiście, możesz też powiedzieć, że jestem kiepskim programistą...

Na koniec mam jeszcze małą sugestię. A może by tak dołożyć prostą procedurę komunikacji, chociażby przez port RS232, i nawet bez translacji poziomów napięć zbudować alternatywę dla układów sterowników wyświetlaczy LED, których cena detaliczna jest - lekko mówiąc - przerażająca? Można by było wykorzystać tani mikrokontroler, na przykład AT89C2051. A może AVR? Wówczas nie potrzeba rezonatora kwarcowego. Myślę, że wraz z cyframi będzie on tańszy niż jeden układ sterownika LED.

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

Dodatkowe informacje

Ewaluacyjną wersję pakietu firmy Raisonance prezentowanego w artykule zamieściliśmy na CD-EP8/2002B.