C dla mikrokontrolerów 8051

część 10

Większość urządzeń wykonanych na mikrokontrolerach wymaga zastosowania jakiegoś interfejsu zapewniającego komunikację z użytkownikiem. W poprzednich odcinkach kursu zajmowaliśmy się prezentacją wyników przy pomocy wyświetlacza LED czy LCD. Czasami jednak trzeba również wprowadzić pewne parametry - na przykład wartość czasu, napięcia itp. Można to zrobić na wiele sposobów, lecz jeden z nich przyjął się wiele lat temu i króluje do dziś - klawiatura.

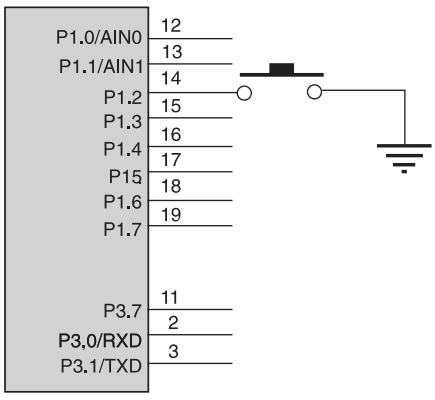
Obsługa klawiatury w C, część 1

Ten odcinek w całości poświęcony będzie obsłudze pojedynczych klawiszy, klawiatur, zawrzemy w nim przegląd najczęściej stosowanych rozwiązań. Zaczniemy od prostych metod wykrywania naciśnięcia pojedynczego klawisza, poprzez proste klawiatury wielostykowe, przedstawimy także zagadnienia obsługi klawiatur matrycowych, a także działających z wykorzystaniem przerwań.

Testowanie stanu pojedynczego przycisku

Na początek zajmiemy się testowaniem stanu pojedynczego przycisku. Wbrew pozorom nie jest to zadanie łatwe, mimo iż tak może się wydawać na początku. Program musi uwzględnić fizyczne właściwości styku, zabezpieczyć się przed przypadkowym jego wciśnięciem lub zewnętrznym zakłóceniem. Na podstawie obsługi pojedynczego przycisku omówimy podstawowe zasady odczytu stanu przełącznika. Funkcje omawiane dalej będą tylko rozwinięciem omawianej tutaj funkcji podstawowej.

Na **rys. 1** pokazano sposób podłączenia klawisza wykorzystywany w przykładzie. Pojedynczy przycisk podłączony został do portu P1.2 mikrokontrolera z rodziny 8051 (AT89C2051). Ty możesz zrobić to tak samo, możesz również wykorzystać inną linię portu. Zwróć tylko uwagę na to, czy nie ma przypadkiem konieczności dołączenia do tej linii rezystora *pullup*. Potrzebne informacje można znaleźć w dokumentacji mikrokontrolera. W przykładzie używałem AT89C2051, który posiada wewnętrzny rezystor *pullup* dołączony do linii P1.2.

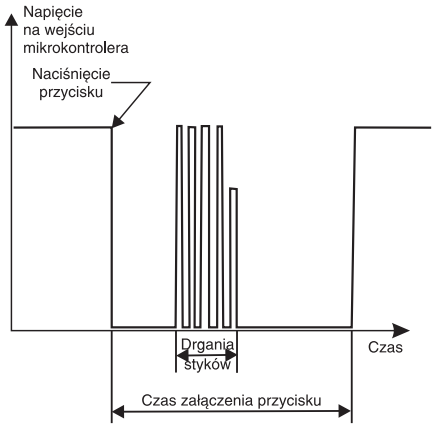


Rys. 1. Podłączenie pojedynczego klawisza do mikrokontrolera wykorzystywane w przykładzie 1

Przycisk podłączyłem w taki sposób, aby w momencie jego naciśnięcia linia portu P1.2 zwierana była do masy. Dlaczego? Otóż w „normalnej" sytuacji, gdy port pracuje jako wejściowy, linia ta jest podłączona do dodatniego napięcia zasilania przez rezystor *pullup*. W konsekwencji mikrokontroler sprawdzając stan bitu P1.2 odczyta stan wysoki. Można go zmienić (pamiętajmy, że linia pracuje jako wejściowa) podając stan niski, tak aby zmienił się potencjał wymuszany przez rezystor. Podanie stanu niskiego odpowiada zwarciu linii P1.2 do masy. To jeden z powodów. Drugi, to mogące pojawić się zakłócenia. W przyjętej konfiguracji znacznie łatwiej je wyeliminować, ponieważ impedancja linii w stanie niskim jest wielokrotnie mniejsza niż w stanie wysokim. Tyle o podłączeniu, teraz warto by wspomnieć o właściwościach fizycznych styków przełącznika.

Podczas zwierania styków przełącznika, może pojawić się ich drganie, co zilustrowano na **rys. 2**. Jak łatwo zauważyć, stan niski na wejściu portu pojawi się co najmniej kilkakrotnie i jeśli mikrokontroler będzie wystarczająco szybki, a zadanie klawisza polegać ma np. na zwiększeniu jakiegoś parametru wartości o 1 przy jego naciśnięciu, to może się okazać, że pojedyncze naciśnięcie klawisza owocuje przyrostem wartości zmiennej o kilka czy nawet kilkanaście kroków. We współcześnie produkowanych miniaturowych przełącznikach zjawisko drgania styków zostało prawie całkowicie wyeliminowane, jednak jego występowanie zależeć będzie od konstrukcji mechanicznej przełącznika - ponieważ tej nigdy nie możemy być pewni, lepiej się przed nim zabezpieczyć.

Lekarstwem na to zjawisko jest prosty zabieg: reakcja na wciśnięty klawisz (czy to opadające zbocze, czy poziom niski sygnału), odczekanie przez czas 20...50 ms i ponowny odczyt stanu portu. Jeśli nie zmienił się on, to oznacza, że klawisz nadal jest wciśnięty i użytkownik oczekuje reakcji na wciśnięcie przycisku. Jeśli odczytany stan jest wysoki, to oznaczać może, że albo odebrano zakłócenie, albo przypadkowo naciśnięto klawisz. Tę prostą zasadę działania wykorzystuje program umieszczony na **list. 1**. Stan klawisza sygnalizowany jest przez bit 3 portu P1. Bit ten przyjmuje poziom logiczny niski, gdy klawisz jest wciśnięty.



Rys. 2. Typowy przebieg sygnału na wejściu mikrokontrolera po naciśnięciu przycisku

List. 1. Prosty program odczytujący stan klawisza podłączonego do P1.2

```
/* prosty program demonstracyjny "odczyt pojedynczego klawisza"
   klawisz włączony pomiędzy masę a port P1.2, stan aktywny = L;
   rezonator kwarcowy 8MHz */

#include <reg51.h>          //dołączenie definicji rejestrów mikrokontrolera

sbit PortKey = P1^2;        //definicja bitu portu klawisza
sbit Active = P1^3;         //port przyjmuje stan niski, gdy klawisz
                            //jest wciśnięty

//opóźnienie około 1 milisekundy dla kwarcu 8MHz
void Delay(unsigned int time)
{
   unsigned int j;

   while (time >= 1)        //wykonanie pętli FOR zajmuje około 1 msek.
   {                        //pętla jest powtarzana TIME razy
       for (j=0; j<65; j++);
       time--;
   }
}

//funkcja - reakcja na naciśnięcie klawisza
//tylko ustawienie stanu portu wyjściowego
void KeyPressed(void)
{
   Active = 0;
}

//funkcja - reakcja, gdy klawisz nie jest wciśnięty
//tylko ustawienie stanu portu wyjściowego
void KeyNotPressed(void)
{
   Active = 1;
}

//początek programu głównego
void main(void)
{
   while (1)                //pętla nieskończona
   {
       if (!PortKey)
       {
           Delay(20);       //opóźnienie 20ms
           if (!PortKey)    //ponowny odczyt klawisz i podjęcie akcji,
           {                //jeśli nadal wciśnięty
               KeyPressed();
               while (!PortKey);    //oczekiwanie na zwolnienie klawisza
           }
       }
       KeyNotPressed();     //akcja, gdy klawisz nie jest wciśnięty
   }
}
```

Początek programu zawiera deklaracje bitów portu: bit 2 pracuje jako wejściowy - do niego jest podłączony klawisz, bit 3 pracuje jako wyjściowy - jego stan zmienia się w zależności od stanu klawisza. Funkcja *void KeyPressed()* zawiera polecenia wykonywane w sytuacji, gdy klawisz jest wciśnięty i odwrotnie: funkcja *void KeyNotPressed()* zawiera polecenia wykonywane, gdy klawisz jest zwolniony. Właściwy odczyt klawisza został zdefiniowany w programie głównym i sprowadza się do testowania stanu pojedynczego bitu portu przy pomocy polecenia *if* (jeśli). Wyrażenie *if (!PortKey)* oznacza: jeśli zmienna *PortKey* ma wartość „nie zero“, to wykonaj polecenia zawarte pomiędzy nawiasami klamrowymi. Czyli: odczekaj 20 milisekund, odczytaj ponownie stan bitu portu. Jeśli nadal jest on równy stanowi niskiemu, to wykonaj funkcję odpowiadającą wciśniętemu klawiszowi. Zadaniem pętli *while (!PortKey)* jest oczekiwanie aż klawisz zostanie zwolniony.

```
while (1) //pętla nieskończona,
{         //w niej odczyt klawisza
 if (!PortKey)
 {
  Delay(20);    //opóźnienie 20ms
  if (!PortKey)
  //ponowny odczyt klawisz
  //i podjęcie akcji,
  {
   KeyPressed();
   //jeśli nadal wciśnięty
   while (!PortKey);
   //oczekiwanie na zwolnienie
   //klawisza
  }
 }
 KeyNotPressed();
 //akcja, gdy klawisz nie jest
 //wciśnięty
}
```

Zaznaczam - jest to jedna z najprostszych możliwych funkcji obsługi klawisza. W większości sytuacji jest wystarczy, ma jednak tę wadę, że po jego wciśnięciu oczekuje na zwolnienie blokując tym samym wykonywanie innych zadań, o ile nie są one obsługiwane przez przerwania.

Funkcja „autorepeat“ klawisza

Każdy użytkownik komputera PC zna funkcję „autorepeat“ klawiatury. Polega ona na powtarzaniu kodu wciśniętego klawisza. Zrealizujemy taką funkcję w programie dla mikrokontrolera. Ustalmy najpierw metodę. Wydaje mi się, że spośród wielu sposobów obsługi klawiszy, najbardziej naturalnymi są:

1. Po wciśnięciu jednokrotne wysłanie kodu wciśniętego klawisza, brak reakcji jeśli klawisz nie został zwolniony.

2. Po wciśnięciu wysłanie kodu klawisza i jeśli nie został on zwolniony powtarzanie kodu klawisza co pewien ustalony odstęp czasu.

3. Po wciśnięciu wysłanie kodu klawisza i jeśli nie został on zwolniony - powtarzanie kodu klawisza początkowo wolno a po wykonaniu pewnej liczby powtórzeń - znacznie szybciej.

W tym odcinku kursu wykonamy program, który zrealizuje w praktyce działanie klawiatury w sposób numer 3. Z racji tego, że liczba klawiszy nie ma większego znaczenia dla zrozumienia zasady działania, zanim przejdziemy do bardziej zaawansowanych rozwiązań, posłużymy się (podobnie jak poprzednio) rozwiązaniem wykorzystującym pojedynczy przycisk. Tu jedna uwaga: to tylko propozycja rozwiązania. Pisząc programy często zauważysz, że jeden problem można rozwiązać na kilka możliwych sposobów.

Aby funkcja automatycznego powtarzania naciśniętego klawisza mogła działać, nie wolno zatrzymywać programu i oczekiwać na zmianę stanu przycisku. Mikrokontroler musi nieprzerwanie (nie bierz tego dosłownie) wykonywać program sprawdzając stan klawisza i sprawdzając warunki dla funkcji „autorepeat“. W związku z tym nie wolno nam użyć konstrukcji *while (!PortKey)* z poprzedniego przykładu. Trzeba zastosować zupełnie inną metodę: program przykładowy liczy liczbę cykli odczytu stanu przycisku i w zależności od jej wartości podejmuje odpowiednią akcję.

List. 2. Program odczytuje stan klawisza podłączonego do P1.2 wyposażony w rozbudowaną funkcję "autorepeat"

```
/* prosty program demonstracyjny "odczyt pojedynczego klawisza"
   z funkcją automatycznego powtarzania.
   klawisz włączony pomiędzy masę a port P1.2, stan aktywny = L;
   rezonator kwarcowy 8MHz */

#include <reg51.h>              //dołączenie definicji rejestrów mikrokontrolera

#define Time_1   900            //time 1 = około 15 ms
#define Time_2   9000           //około 1,5 sekundy
#define Time_3   9020           //po 20 wykonaniach pętli dla Time_2

unsigned int Counter = 0;       //deklaracja zmiennej licznika
sbit PortKey = P1^2;            //definicja bitu portu klawisza
sbit Active = P1^3;             //port przyjmuje stan niski, gdy klawisz
                                //jest wciśnięty

//opóźnienie około 1 milisekundy dla kwarcu 8MHz
void Delay(unsigned int time)
{
   unsigned int j;

   while (time >= 1)            //wykonanie pętli FOR zajmuje około 1 msek.
   {                            //pętla jest powtarzana TIME razy
       for (j=0; j<65; j++);
       time-;
   }
}

//funkcja - reakcja na naciśnięcie klawisza
//tylko ustawienie stanu portu wyjściowego
void KeyPressed(void)
{
   Active = Ćounter;
}

//funkcja - reakcja, gdy klawisz nie jest wciśnięty
//tylko ustawienie stanu portu wyjściowego
void KeyNotPressed(void)
{
   Active = 1;
}
//początek programu głównego
void main(void)
{
   while (1)                    //pętla nieskończona
   {
       if (!PortKey)
       {
           Counter++;           //tak, zwiększ licznik
           if (Counter > Time_3)    //czy licznik większy od warunku dla
           {                    //szybszego autorepeat?
               KeyPressed();    //tak - wykonaj fragment kodu
               Counter-;
               Delay(100);      //krótsze opóźnienie
           } else;

           if (Counter > Time_2)    //czy licznik większy od warunku dla
           {                    //wolniejszego autorepeat?
               KeyPressed();    //tak - wykonaj fragment kodu
               Delay(500);      //dłuższe opóźnienie
           } else;

           if (Counter == Time_1)   //czy licznik osiągnął wartość, gdy
           {                    //uznajemy klawisz za wciśnięty?
               KeyPressed();    //tak - wykonaj fragment kodu
           }
       }
       else;
       {
           Counter = 0;         //licznik jest zerowanym, gdy klawisz
           KeyNotPressed();     //zostaje zwolniony
       }
   }
}
```

Oto część programu napisanego w języku C, w którym pozostawiłem tylko naprawdę niezbędne fragmenty kodu źródłowego:

```
if (!PortKey)
//czy naciśnięto klawisz?
{
 Counter++;   //tak, zwiększ licznik
 if (Counter > Time_3)
 //czy licznik jest większy
 //od warunku dla
 { //szybszego autorepeat?
  KeyPressed();  //tak - wykonaj
                 //fragment kodu
  Counter-;
  Delay(100);  //krótsze opóźnienie
 }
 else
 if (Counter > Time_2)
 //czy licznik jest większy
 //od warunku dla
 {    //wolniejszego autorepeat?
  KeyPressed();  //tak - wykonaj
                 //fragment kodu
  Delay(500);  //dłuższe opóźnienie
 }
 else
 if (Counter == Time_1)
 //czy licznik osiągnął wartość,
 //gdy uznajemy klawisz
 {    //za wciśnięty?
  KeyPressed();  //tak - wykonaj
                 //fragment kodu
 }
}
else
 Counter = 0;
//zeruj licznik, jeśli zwolniono
//klawisz
```

Na początku program sprawdza stan klawisza. Jeśli jest on wciśnięty, to wartość zmiennej *Counter* zwiększana jest o 1 i rozpatrywana przez konstrukcję *if*. W przeciwnym przypadku zmiennej nadawana jest wartość 0. Zadaniem *if* jest zdecydowanie jaka akcja zostanie podjęta w zależności od wartości zmiennej *Counter*. Stała *Time_1* to ilość pojedynczych operacji odczytu bitu przycisku, po której uznaje go za wciśnięty. Jej wartość w głównej mierze zależy od szybkości mikrokontrolera i musi być dobrana indywidualnie dla budowanego układu. Stała *Time_2*, to ilość przebiegów cykli odczytu, dla której uznaję warunek dla „wolniejszego" powtarzania za spełniony. Wykonywany jest wówczas fragment programu zawierający instrukcję *Delay(500)* powodującą przerwę ok. 0,5 sekundy pomiędzy powtórzeniami odczytu klawisza. Jeśli wartość zmiennej *Counter* osiągnie wartość stałej *Time_3*, to wykonywany jest fragment programu z instrukcją *Delay(100)*: czas pomiędzy powtórzeniami akcji właściwej dla wciśniętego przycisku jest pięciokrotnie krótszy. Instrukcja *Counter-* służy do zabezpieczenia przed przekroczeniem wartości dopuszczalnej dla danego typu zmiennej *Counter*.

Do automatycznego powtarzania kodu klawisza wrócimy jeszcze przy okazji budowy menu. W tym momencie, proponuję wykorzystać program z **list. 2** aby zobaczyć jak nastawy wartości *Time_1*, *Time_2* i *Time_3* wpływają na pracę programu wykonywanego przez mikrokontroler.

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