# Kurs programowania mikrokontrolerów PIC (3)

## Interfejs użytkownika

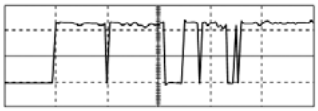

*Interfejs użytkownika jest przeznaczony do komunikacji pomiędzy osobą obsługującą urządzenie a obsługiwanym urządzeniem (w tym przypadku sterownikiem mikroprocesorowym). Interfejs musi dawać użytkownikowi możliwość wprowadzania parametrów i poleceń. Najprostszym urządzeniem wejściowym jest klawiatura. Naciskając klawisze uruchamiany realizację zadania, wprowadzamy parametry np. godzinę i datę, dokonujemy wyboru funkcji z menu itp. Sterownik używa do sygnalizowania swojego stanu różnych elementów: diod LED, wyświetlaczy, miniaturowych głośników itp. W tym odcinku zajmiemy się się głównie obsługą komponentów menu użytkownika.*

**Dodatkowe materiały na CD/FTP:**
ftp://ep.com.pl, user: 10925, pass: 87thc181
- poprzednie części kursu

### Klawiatura

Klawisze ze stykami zwiernymi są jednymi z podstawowych elementów sterowników. Wydawałoby się, że nie ma nic prostszego niż wykrycie zwartego styku. Ale jak to zwykle bywa, diabeł tkwi w szczegółach. W czasie zwierania kontaktów na skutek zjawiska sprężystości odbijają się one od siebie, co jest powodem powstawania krótkotrwałych przerw w obwodzie, w czasie których obwód elektryczny jest zamykany i otwierany. Zależnie od rodzaju przycisku to zjawisko może trwać nawet do 20 ms od pierwszego zwarcia. Jeżeli nasz mikrokontroler jest taktowany sygnałem o częstotliwości 4 MHz, to rdzeń będzie taktowany z częstotliwością 4 MHz/4=1 MHz. Rozkaz testowania styków wykona się w czasie 1/1 MHz= 1 ms. Jak widać mikrokontroler może testować stan wejścia, do którego jest dołączony styk szybciej niż – nawet kilkakrotnie w czasie trwania drgań. Może to powodować (i najczęściej powoduje) błędną interpretację stanu przycisku.

Najczęściej spotykanym sposobem radzenia sobie z tym problemem jest programowa eliminacja zakłóceń. Załóżmy, że program ma określić czy styk jest zwarty, czy nie. Po wykryciu zmiany stanów na którejś z linii klawiatury odmierzamy opóźnienie ok. 20...30 ms i po tym czasie ponownie odczytujemy stan klawiatury. Mamy wtedy pewność, że ten stan jest stabilny.



**Rysunek 1. Przebiegi na linii wejściowej po naciśnięciu przycisku**

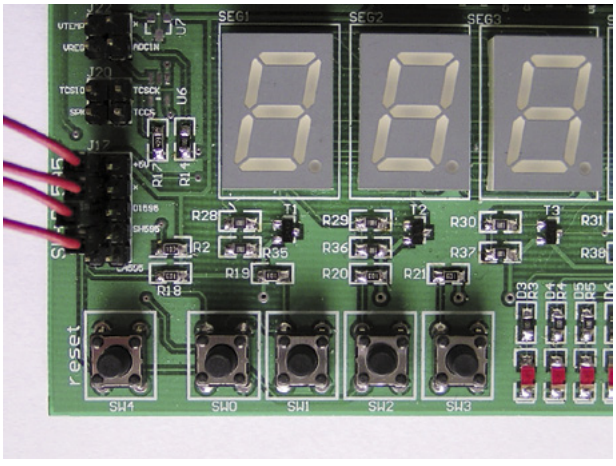
Zwykle styki klawiatury zwierają linie portów mikrokontrolera do masy i jednocześnie te linie są połączone do plusa zasilania przez rezystory podciągające. Jeżeli styk jest rozwarty, to na wejściowej linii portu jest stan wysoki wymuszony przez rezystor. Po zwarciu styku na linii jest stan niski. Taki układ z 4 stykami jest zabudowany w module ewaluacyjnym (**rysunek 2**). Wyprowadzenia SW0...SW3 są połączone z pinami złącza J17 i trzeba je połączyć przewodami z liniami portu mikrokontrolera. Na fotografii 3 pokazano umiejscowienie na płytce styków SW1...SW3 i złącza J17 (z wetkniętymi kablami).

Obsługę klawiatury można wykonywać na kilka sposobów. Procedura czytania może zwracać kod przyciśniętego klawisza po jednokrotnym naciśnięciu i puszczeniu klawisza. Taki sposób jest wygodny w wielu zastosowaniach, ale czasami trzeba aby kod przyciśniętego klawisza był powtarzany co zadany czas tak długo, jak styk jest zwarty (na przykład przy ustawianiu zegara). Na początek zajmijmy się pierwszym przypadkiem.

Obsługę klawiatury można podzielić na kilka etapów:

1. Czekanie na zwolnienie przycisku. Właściwa procedura czytania stanu styków może rozpocząć się tylko wtedy, gdy żaden z klawiszy nie jest wciśnięty.
2. Czekanie na wciśnięcie klawisza.
3. Po stwierdzeniu, że jeden z przycisków jest wciśnięty zostaje uruchomione odliczanie opóźnienia eliminującego możliwość odczytania stanu styków podczas ich drgań.
4. Po odliczeniu czasu ponownie jest sprawdzany stan linii portów dołączonych do styków. Jeżeli żaden styk nie jest zwarty, to program wraca do punktu 2 lub kończy działanie z informacją, że żaden ze styków nie został wciśnięty. Jeżeli styk jest zwarty, to określamy jego kod (numer).
5. Opcjonalnie, czekamy na zwolnienie klawisza.

Do prób z klawiaturą połączymy SW0... SW3 z liniami RA0...RA3 portu PORTA mikrokontrolera. Poprawność działania procedury odczytu klawiatury będziemy sygnalizować za pomocą zaświecenia lub zgaszenia diod LED LD0...LD3 połączonych z liniami RB0...RB3 portu PORTB. Do odliczania opóźnień zostanie wykorzystana znana już procedura delay wykorzystująca przerwanie od przepełnienia licznika Timer0. Linie RA0...




**Fotografia 2. Klawiatura modułu ewaluacyjnego**

**Listing 1. Procedura odczytania stanów klawiszy SW0...SW3**

```
unsigned char GetKey(void){
unsigned char key;
  while((PORTA&0x0f)!=0x0f);       //czekaj na puszczenie klawisza
  while((PORTA&0x0f)==0x0f);       //czekaj na przyciśnięcie klawisza
  delay(25);                       //odczekanie drgań styków
  key=~PORTA;                      //aktywne jedynka
  key=key&0x0f;                    //tylko 4 młodsze bity
  while((PORTA&0x0f)!=0x0f);       //czekaj na puszczenie klawisza
  return(key);
}
```

**Listing 2. Testowanie działania procedury GetKey**

```
unsigned char  kl;

  while(1){
  kl=GetKey();    //odczytaj stan klawiatury
  if(kl==1)       //klawisz SW0
  RB0=~RB0;
  if(kl==2)       //klawisz SW2
  RB1=~RB1;
  if(kl==4)       //klawisz SW3
  RB2=~RB2;
  if(kl==8)       //klawisz SW4
  RB3=~RB3;
}
```

RA3 muszą być skonfigurowane jako wejściowe, a linie RB0...RB3 jako wyjściowe (sterowanie diodami LED).

Na **listingu 1** zamieszczono procedurę GetKey działająca według algorytmu opisanego wyżej w punktach. Po wykryciu zwarcia styków i odczekaniu 25 ms do zmiennej key jest wpisywana zanegowana wartość odczytana z portu PORTA. Zrobiona tak dla wygody użytkownika, ponieważ stanem aktywnym jest poziom niski, a jest wygodniej posługiwać się na przykład dla wciśniętego SW1 kodem 0b00000010 niż nie 11111101. Funkcja zwraca następujące kody wciśniętych klawiszy: SW1=0x01, SW2=0x02, SW3=0x04, SW4=0x08. Jeśli po wykryciu zwarcia i doliczeniu 25 ms ponowne odczytanie nie wykryje zwarcia styku, procedura zwróci 0x00 i można to traktować jako kod błędu.

**Listing 3. Testowanie ciągłego czytania klawiatury**

```
unsigned char  kl, port;
  while(1){
  while((PORTA&0x0f)!=0x0f){       //wykonuj pętlę, gdy klawisz wciśnięty
    kl=~PORTA;                     //aktywne jedynka
    kl=kl&0x0f;                    //tylko 4 młodsze bity
    if(kl==1)
    port=port+1;
    if(kl==2)
    port=port-1;
    port=port&0x0f;
    PORTB=port;
    delay(200);                    //opóźnienie 200msek
  }
}
```

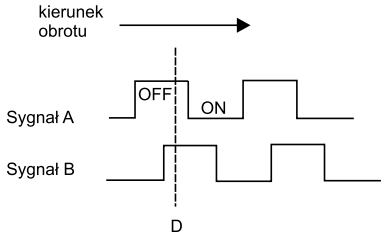
**Listing 4. Zmodyfikowane testowanie ciągłego czytania klawiatury**

```
...
kl=0;
while(1){
  while((PORTA&0x0f)!=0x0f){       //wykonuj pętlę, gdy klawisz wciśnięty
    kl=~PORTA;                     //aktywne jedynka
    kl=kl&0x0f;                    //tylko 4 młodsze bity
    if(kl==1)
    port=port+1;
    if(kl==2)
    port=port-1;
    if(kl==8)                      //kod ESCAPE
    break;
    port=port&0x0f;
    PORTB=port;
    delay(200);
  }
  if(kl==8)
  break;
}
PORTB=0x0f;                        //sygnalizacja zakończenia „regulacji”
while(1);
```

## Automatyczne powtarzanie wciśniętego klawisza

Jest wiele sytuacji, w których „impulsowe” przyciskanie klawisza jest uciążliwe. Na przykład, jeżeli chcemy regulować siłę głosy za pomocą potencjometru cyfrowego to zmiana tłumienia o 10 dB z krokiem co 0,5 dB wymagałaby 20-krotnego naciśnięcia klawisza obsługiwanego przez procedurę GetKey. Bardziej naturalne wydaje się naciśnięcie i przytrzymanie przycisku aż poziom sygnału będzie odpowiedni. Do tego celu czytanie klawiatury można zorganizować inaczej. W pętli nieskończonej jest sprawdzany warunek przyciśnięcia klawisza i gdy jest on wciśnięty, to jest identyfikowany jego kod. Następnie zależnie od kodu wciśniętego klawisza program podejmuje jakąś akcję. Pomiędzy kolejnymi odczytami stanu styków upływa czas na wykonanie czynności zależnej od wciśniętego klawisza i dodatkowe opóźnienie. To dodatkowe opóźnienie określa, jak często jest powtarzany kod wciśniętego klawisza.

Na **listingu 3** zamieszczono fragment programu demonstrujący ideę takiego czytania klawiatury. Po każdym naciśnięciu SW1 jest zwiększana zmienna port, a po naciśnięciu SW2 – zmniejszana. Potem ta wartość jest przepisywana do rejestru portu PORTB. Stan czterech młodszych linii PORTB jest wyświetlany przez LED0...LED3. Opóźnienie 200 ms określa, jak często jest czytana klawiatura i modyfikowany stan linii PORTB.



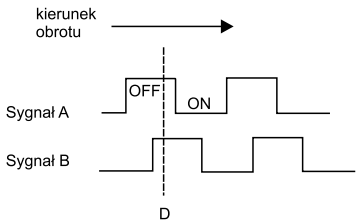

**Rysunek 3. Klawisze SW0...SW3 i złącze J17**

**Fotografia 4. Przykłady enkoderów firmy Bourns**

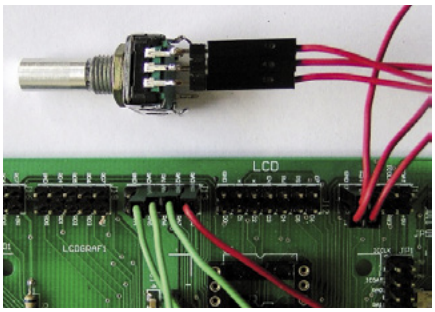
Procedura z **listingu 3** ma pewną wadę – program wykonuje się w niekończonej pętli i jeżeli w nią wejdzie, to nie ma możliwości jej opuszczenia. Możemy wyznaczyć jeden z przycisków odpowiadający klawiszowi ESC (escape). Po jego przyciśnięciu program opuści pętlę i zasygnalizuje to zaświecając wszystkie diody LED (**listing 4**).

Sterowanie przyciskami to jeden z najprostszych i najbardziej przydatnych elementów interfejsu użytkownika. Jest jednak inny element, który w wielu zastosowaniach jest równie użyteczny, a przy tym ma wiele zalet. Myślę tu o obrotowym enkoderze zwanym inaczej impulsatorem. Jest to mechaniczny element z trzema wyprowadzeniami: dwoma skrajnymi oznaczanymi A i B oraz jednym środkowym. Wyglądem nieco przypomina potencjometr obrotowy (**fotografia 4**).

W czasie obrotu ośki enkodera wyprowadzenia A i B są cyklicznie zwierane do wyprowadzenia środkowego. Jeżeli wyprowadzenia A i B podłączymy do plusa zasilania przez rezystory, to w czasie obracania enkoderem na tych wyprowadzeniach pojawią się przesunięte w fazie impulsy ujemne. Z zależności czasowych pomiędzy nimi można określić kierunek obrotu enkodera i moment generowania kodu równoważnego zwieraniu styku przycisku. Inaczej mówiąc, w czasie kręcenia ośką występuje efekt cyklicznego zwierania dwóch styków umownych: jednego dla obrotu w lewo, drugiego dla obrotu w prawo.




**Rysunek 5. Przebiegi na wyprowadzeniu enkodera**




**Fotografia 6. Połączenie modułu z enkoderem PEC11**

**Listing 5. Fragment procedury obsługi przerwania o Timer0 obsługujący enkoder**

```
if(f2==0){                  //jeszcze nie wykryto A=B=0 , lub A=B=1
  if(SA==0&&SB==0){
    f0=1;f1=0;f2=1;         //wykryto A=B=0
    impu=0;impd=0;
  }
  if(SA==1&&SB==1){
    f0=0;f1=1;f2=1;         //wykryto A=B=1
    impu=0;impd=0;
  }
}

if(f1==1){                  //wykrywanie obrotu dla stanu początkowego A=B=1
  if(SA!=1||SB!=1){
    if(SA==0&&SB==1){
      impu=1; f1=0;f3=1;
    }
    if(SB==0&&SA==1) {
      impd=1;f1=0;f3=1;
    }
  }
}
if(f0==1){                  //wykrywanie obrotu dla stanu początkowego A=B=0
  if(SA!=0||SB!=0){
    if(SA==0&&SB==1){
      impd=1;f0=0;f3=1;
    }
  if(SB==0&&SA==1){
    impu=1;f0=0;f3=1;}
  }
}
```

Na **rysunku 5** pokazano przebiegi czasowe w czasie obracania ośki enkodera. Przebieg na wyprowadzeniu A wyprzedza przebieg na wyprowadzeniu B. Przy zmianie kierunku obrotu przebieg na wyprowadzeniu B będzie wyprzedzał przebieg na wyprowadzeniu A. Na pierwszy rzut oka nie powinno być większych problemów z detekcją kierunku obrotów i momentem generowania kodu „zwarcia styku". Jednak ten element sprawia sporo kłopotów początkującym programistom.

Spróbujemy zastanowić się dlaczego tak się dzieje na przykładzie impulsatora z serii PEC11 firmy Bourns. Na początek zapoznamy się z jego danymi technicznymi i najbardziej będzie nas interesował parametr Contact Bounce, równy 5 ms przy 15 obrotach na minutę. Przy obracaniu ośki z tą prędkością stany na wyprowadzeniu A i B zmieniają się co 5 ms. Program obsługi musi być tak napisany, aby wykrył zmiany stanów. Ale to nie wszystko – w czasie zmian stanów występuje tez zjawisko drgań styków, które trzeba wyeliminować sprzętowo lub programowo.

Sposób dołączenia enkodera do płytki ewaluacyjnej pokazano na **fotografii 6**. Na początek spróbujmy wykonać do podprogram obsługi enkodera w sposób tradycyjny tj. testując w pętli stany wejść A i B. Sam kiedyś napisałem kilka procedur działających w taki sposób, ale nie polecam tego sposobu. Zwykle po ruchu ośki i wygenerowaniu kodu program ma coś do wykonania. Może to być np. zmiana wartości i jej wyświetlenie. Jeżeli te czynności trwają bardzo krótko, to wszystko działa, ale kiedy trwają dłużej niż ok. 1...2 ms, to zaczyna się „gubienie" kodów. Zastanawiałem się dlaczego, skoro czas trwania impulsu wynosi ok. 5 ms przy stosunkowo szybkim kręceniu ośką. Otóż krytyczne jest zarejestrowanie pewnych zmian stanów na wyprowadzeniach A i B. Jeżeli program nie zarejestruje ich wtedy, gdy wystąpią, to procedura obsługi gubi jeden a czasami dwa kody lub generuje błędny kod (dla obrotu w drugim kierunku). Posługiwanie się enkoderem w takim przypadku jest bardzo niewygodne.

Jednym ze sposobów na testowanie stanów wyprowadzeń niezależnie od czynności wykonywanych przez program główny jest wykorzystanie przerwań timera. Po kilku próbach okazało się, że bardzo dobrze działają procedury obsługi czytające stan enkodera co 1 ms.

Większość enkoderów ma wbudowane zapadki ustawiające go w ustalonym położeniu. Dobrze jest na początek zmierzyć w jakim stanie są styki w położeniu wymuszonym przez zapadkę. Enkoder PEC11 ma w takich położeniach kolejno oba styki zwarte z środkowym i oba styki rozwarte. Będzie to punkt odniesienia dla automatu obsługującego enkoder. Na początek zdefiniujmy cztery znaczniki bitowe przypisane do trzech faz enkodera:

- f2 – ustawienie tego bitu oznacza wykrycie dwóch „1" na A i B (oba styki rozwarte) lub dwóch „0" na A i B (oba styki zwarte). Do momentu kiedy f2=0 procedura obsługi enkodera nie testuje kierunku obracania się ośki.
- f1 – ustawienie tego bitu oznacza że wykryto „1" na A i B.

**Listing 6. Odczytanie stanu enkodera i generowanie kodu**

```
unsigned char GetEncoder(void){
unsigned char code;
  if(SW==0){                        //czytanie stanu dodatkowego styku
    delay_ms(20);
    while(SW==0);
    return(KOD_IMP_SW);
  }
  if(f3==1){                        //zidentyfikowano obrót ośki
    f3=0;
    if(impu==1){
    f2=0;
    return(KOD_IMP_UP);             //generowanie kodu obrotu
  } else
  if(impd==1){
    f2=0;
    return(KOD_IMP_DWN);}           //generowanie kodu obrotu
  }
  f2=0;
  return(0xff);
}
```

**Listing 7. Testowanie działania obsługi enkodera**

```
//definicje kodów
#define KOD_IMP_SW 0x33
#define KOD_IMP_UP 0x34
#define KOD_IMP_DWN 0x35

while(1){
  kl=GetEncoder();
  if(kl==KOD_IMP_UP) RA0=1;delay(60);RA0=0;
  if(kl==KOD_IMP_DWN) RA1=1;delay(60);RA1=0;
}
```

– f0 – ustawienie tego bitu oznacza, że wykryto „0” na A i B.
– f3 – ustawienie tego bitu oznacza zakończenie detekcji obrotu i wygenerowanie kodu.

Po inicjalizacji wszystkie te bity są wyzerowane. W obsłudze przerwania, gdy f2=0 jest testowane wykrywanie A=B=0 lub A=B=1. Kiedy takie stany są wykryte, to f2=1, f1=1 lub f0=1. Ustawienie f2 powoduje, że program już nie będzie próbował wykrywać stanów początkowych, ale zajmie się określaniem kierunku obrotów i generowaniem kodu. Fragment obsługi umieszczony w procedurze przerwania zgłaszanego co 1 ms pokazano na **listingu 5**. Użyta metoda daje pewność, że kiedy po wykryciu stanów początkowych rozpocznie się określanie kierunku obrotów, to na pewno się wykona. Po wykryciu obrotu są:

– ustawiony bit f2,
– wyzerowane bity f1 i f0,
– ustawiony bit f3,
– ustawiany jest bit impd lub impu.

Jeżeli popatrzymy na listingu 5, to w tym momencie nie jest wykonywane żadne sprawdzanie poziomów na wyprowadzeniach A i B, bo f2=1, f1=0 i f0=1. Procedura umieszczona w programie głównym może w dowolnym momencie odczytać stan bitu f3 (kod gotowy do odczytania) i na podstawie impd lub impu wygenerować kod (**listing 6**).

Po odczytaniu stanu jest zerowany bit f2 pozwalający na odczytanie nowego kodu i bit f3 sygnalizujący odczytany poziom. Automat jest gotowy do odczytania nowego stanu enkodera. Procedura sprawdza też czy nie został zwarty dodatkowy styk enkodera. Jest on zwierany po przyciśnięciu ośki PEC11.

Na **listingu 7** przedstawiono program testujący działanie obsługi enkodera Obrót ośki w jedna stronę powoduje zaświecenie się diody D10, a w drugą stronę diody D9. Wyprowadzenia A i B zostały odpowiednio dołączone do linii RB0 i RB1 portu PORTB. Wyzerowanie bitu RBPU powoduje, że wszystkie linie PORTB ustawione jako wejściowe są podciągane do plusa zasilania przez wewnętrzne rezystory. Procedurę przetestowano z enkoderami typu PEC11 i ECW1J-B24 (rysunek 6). Enkodery można też obsługiwać używając do tego celu wejść przerwań zewnętrznych (INT). Jednak w tym przypadku trzeba zadbać o programową eliminację drgań styków rozbudowując procedurę obsługi i łącząc ją z odliczaniem opóźnień lub zastosować zewnętrzny układ RC eliminujący to zjawisko sprzętowo.

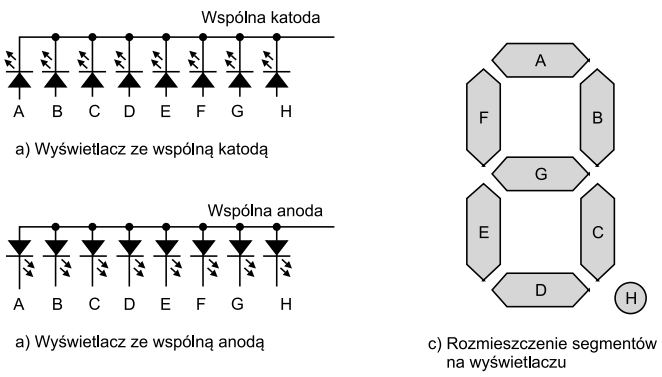
## 7-segmentowy wyświetlacz LED

Przez wiele lat podstawowym elementem stosowanym w sterownikach mikroprocesorach były 7-segmentowe wyświetlacze LED. Mimo, iż teraz mamy do dyspozycji wyświetlacze LCD mające większe możliwości obrazowania informacji, to wyświetlacze LED nadal są chętnie stosowane w zegarach cyfrowych, termometrach, częstościomierzach, woltomierzach itp. 7-segmentowy wyświetlacz LED może być produkowany w wariancie ze wspólną katoda lub ze wspólną anodą. Segmenty są standardowo oznaczane literami A...G , a kropka dziesiętna literą H lub symbolem skrótem DP (**rysunek 7**).

Wyświetlacze steruje się statyczne lub dynamiczne. Sterowanie statyczne jest łatwe. Na liniach portu połączonych z wyprowadzeniami segmentów wystawia się kombinację stanów zaświecających odpowiednie segmenty. W takiej sytuacji dla każdego wyświetlacza jest potrzebny jeden port 8-bitowy. W większości przypadków jest potrzebne sterowanie kilkoma (minimum czterema) wyświetlaczami i takie rozwiązanie ze względu na liczbę potrzebnych linii portów może nie być możliwe. Poza tym sterowanie statyczne charakteryzuje się sporym poborem prądu, ponieważ każdy świecący się segment potrzebuje wymaga prądu o natężeniu ok. 5 mA. Z tych powodów sterowanie statyczne jest rzadko stosowane i zwykle ogranicza się do jednego lub dwóch wyświetlaczy.

Sterowanie dynamiczne ma trochę wspólnego ze sterowaniem jasnością diody LED za pomocą przebiegu PWM. Segmenty wyświetlacza nie świecą światłem ciągłym, tylko są okresowo zaświecane i gaszone przez co jest redukowany pobór prądu. Poza tym dla większej liczby cyfr znacząco mniejsza jest liczba potrzebnych do sterowania nimi linii portów. Niestety, dzieje się to kosztem skomplikowania programu obsługi.

Na rysunku 8 pokazano przykład połączenia dwóch wyświetlaczy sterowanych dynamicznie. Wyprowadzenia segmentów wszystkich wyświetlaczy są połączone równolegle i tworzą 8 linii. Te linie są połączone do wyprowadzeń portu mikrokontrolera. We współczesnych mikrokontrolerach wydajność prądowa linii portu wynosi zwykle co najmniej 20 mA (dla PIC18F2320 – 25 mA) i do sterowania segmentów nie jest potrzebny dodatkowy wzmacniacz prądowy. Wyprowadzenie wspólnej anody każdego z wyświetlaczy osobno jest połączone przez klucz sterowany liniami portu do plusa zasilania. Klucz jest niezbędny, bo przez wyprowadzenie wspólnej anody płynie sumaryczny prąd wszystkich segmentów zazwyczaj znacznie przekraczający wydajność prądową linii portów mikrokontrolera. W czasie wyświetlania pierwszej cyfry zamykany jest klucz zasilający anodę pierwszego wyświetlacza, a na linie portu wystawiana jest kombinacja zapalająca wymagane segmenty. Taki stan trwa jakiś czas po czym otwierany jest klucz zasilający anodę pierwszego wyświetlacza, a zamyka się na taki sam czas klucz zasilający anodę drugiego wyświetlacza. Jednocześnie na liniach poru mikrokontrolera wystawiana jest kombinacja zapalająca segmenty drugiej cyfry. Jeżeli te czynności będą powtarzane odpowiednio szybko, to przełączanie nie bę-



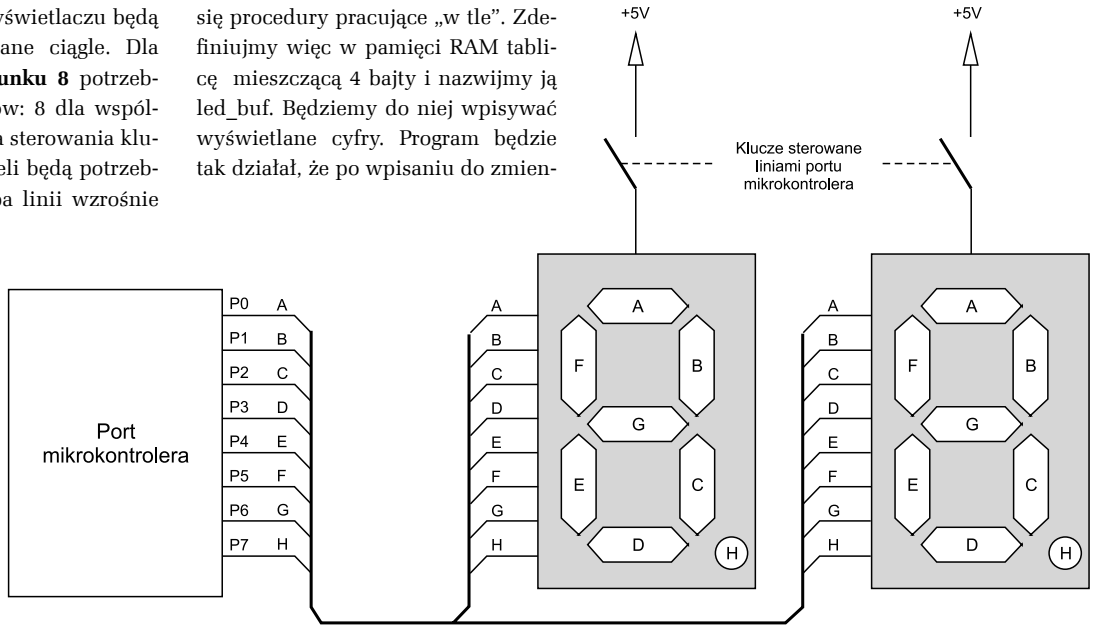

**Rysunek 7. Połączenia diod LED i rozmieszczenie segmentów**

dzie widoczne i cyfry na wyświetlaczu będą postrzegane jako wyświetlane ciągle. Dla dwóch wyświetlaczy z **rysunku 8** potrzebnych będzie 10 linii portów: 8 dla wspólnych linii segmentów i 2 dla sterowania kluczami wspólnych anod. Jeżeli będą potrzebne 4 wyświetlacze, to liczba linii wzrośnie tylko o dwie.

W module ewaluacyjnym zastosowano 4 wyświetlacze połączone do wyświetlania dynamicznego (**rysunek 9**, **rysunek 10**).

Sygnały sterujące segmentami i wzmacniaczami (kluczami) wspólnej anody są doprowadzone do złącza J10. Do sterowania będziemy potrzebować 12 linii portów: 8 dla sterowania segmentami i 4 do sterowania kluczami włączającymi zasilanie anod. Segmenty będą sterowane za pomocą PORTC, a klucze przez 4 młodsze bity portu PORTA. Zastosowane wyświetlacze mają wspólną anodę. Segment jest zaświecany, gdy na linii sterującej SINx jest wymuszony stan niski. Rezystory R23... R27 i R32...R34 ograniczają prąd segmentów zasilanych z napięcia +5 V. Klucze załączające

się procedury pracujące „w tle". Zdefiniujmy więc w pamięci RAM tablicę mieszczącą 4 bajty i nazwijmy ją led_buf. Będziemy do niej wpisywać wyświetlane cyfry. Program będzie tak działał, że po wpisaniu do zmien-

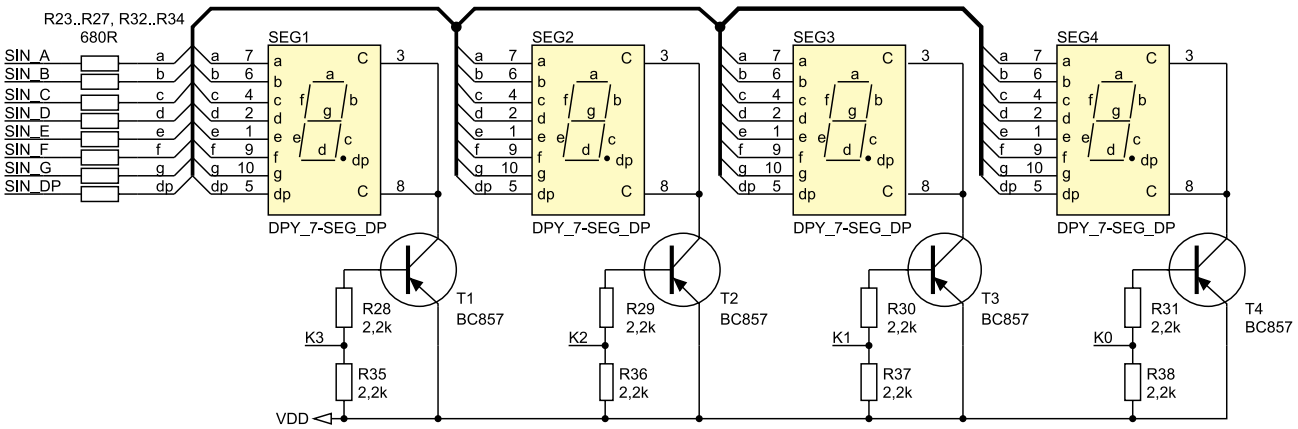



**Rysunek 8. Połączenie wyświetlaczy sterowanych dynamicznie**

nej np. liczby 2, na wyświetlaczu ma się wyświetlić cyfra 2.

Kolejne bajty tablicy led_buf będą „przypisane" do kolejnych cyfr wyświetlacza. Kiedy jest zamykany klucz podający napięcie zasilające pierwszego wyświetlacza, to będzie na nim wyświetlana cyfra o kodzie zapisanym w elemencie tablicy o indeksie 0 (led_buf[0]). Przy początkowej deklaracji tablicy unsigned char led_buf[4]={1,9,6,2}; na wyświetlaczu zostanie wyświetlona liczba „1962", tak jak pokazano na rysunku 10.

Kod cyfry z tablicy led_buf musi uaktywnić kombinację stanów na liniach portów sterujących zapalaniem segmentów, tak aby na przykład liczba 8 była wyświetlona jako

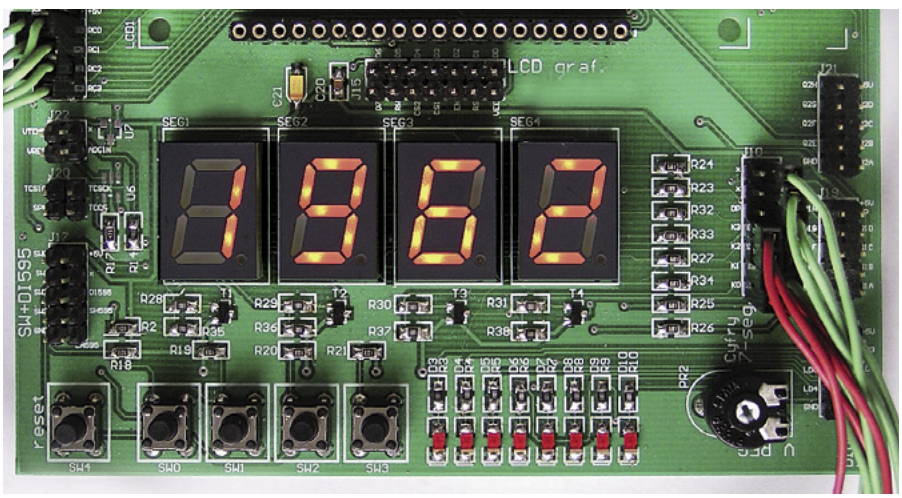



**Rysunek 9. Schemat połączeń 4-cyfrowego wyświetlacza LED**

napięcie +5 V są zbudowane z tranzystorów PNP. Podanie poziomu niskiego (masy) na wyprowadzenie K1...K3 powoduje wejście tranzystora w stan przewodzenia i podanie napięcia na wyprowadzenia C wyświetlacza.

Najwygodniej jest realizować obsługę z użyciem cyklicznie zgłaszanego przerwania od przepełnienia licznika. Ponieważ mamy już skonfigurowane i działające przerwania od Timer0 zgłaszane co 1 ms, to załóżmy na początek, że klucze tranzystorowe będą załączane z częstotliwością 1 kHz.

Prawidłowo skonstruowane procedury wyświetlania nie są widoczne dla programu użytkownika. Kod wyświetlanej cyfry jest wpisywany do bufora w pamięci RAM mikrokontrolera, a wyświetlaniem zajmują




**Rysunek 10. Rozmieszczenie wyświetlaczy LED na płytce**

**Listing 8. Definicja tablicy konwersji**

```
Unsigned char const led_konw[10]={0x40, 0x79, 0x24, 0x30, 0x19, 0x12, 0x02,
0x78, 0x00, 0x10};
```

**Listing 9. Procedura obsługi wyświetlacza umieszczona w obsłudze przerwania od TIMER0**

```
++led_l;
led_l=(led_l&3);                        //kolejna cyfra
led_c=led_buf[led_l];                   //kod cyfry do led_c
PORTA=an_konw[led_l];                   //sterowanie kluczami
if(led_l==led_kr)
PORTC=dig_konw[led_c];                  //sterowanie segmentami z kropką
else
PORTC=dig_konw[led_c]|0x80;             //sterowanie segmentami bez kropki
```

ósemka LED. Najprościej do takiego przekodowania użyć dodatkowej tablicy led_konw umieszczonej w pamięci programu mikrokontrolera (**listing 8**).

W trakcie konwersji kod cyfry jest indeksem tablicy led_konw. Na przykład liczby 9 element led_konw[9] będzie miał wartość 0x10 i taką wartość trzeba przesłać na linie portu aby na wyświetlaczu zapaliła się dziewiątka. Warto tutaj dodać, że poziomem aktywnym jest „0”, to znaczy, jeżeli linia jest wyzerowana, to połączony z nią segment świeci się.

W procedurze sterowania wyświetlaczem umieszczonej w obsłudze przerwania musimy spowodować, by przy każdym zgłoszonym przerwaniu zaświecała się kolejna cyfra. Wprowadzimy kolejna zmienną led_l zawierającą numer aktywnej (świecącej się) cyfry. Ponieważ cyfr jest 4, to zmienna led_l zmienia swoja wartość od 0 do 3. Numer aktywnej cyfry adresuje bufor led_buf i program pobiera z niego odpowiedni, wyświetlany kod. Na przykład kiedy led_l=2, to w buforze led_buf[2] jest cyfra 6. Ta cyfra adresuje tablicę led_konw, procedura wyświetlająca pobiera z niej kod 0x02 i przesyła na port PORTC, tak aby na trzecim wyświetlaczu od lewej (SEG3) wyświetliła się cyfra 6. Jednocześnie trzeba by w tym samym momencie został otwarty klucz z tranzystorem T3 podający plus zasilania na anodę SEG3. Linia sterująca kluczem T3 powinno być wyzerowana, a na pozostałych liniach sterujących kluczami powinien być poziom wysoki. Najwygodniej jest sterować liniami RA0...RA3 za pomocą konwersji wykorzystującej kolejną tablicę an_konw[4]={0x07,0x0b,0x0d,0x0e};. Indeksem tej tablicy jest zmienna led_l a pobrana wartość jest przesyłana do rejestru portu PORTA. Na **listingu 9** pokazano fragment procedury obsługi przerwania sterujący wyświetlaczem LED.

Aby umożliwić sterowanie kropką dziesiętną wprowadzono kolejna zmienną led_kr. Wpisujemy do niej numer wyświetlacza, w którym kropka ma być zaświecona. Jeżeli led_kr jest równa numerowi aktywnego wyświetlacza i kropka ma być wyświetlana, to najstarszy bit bajtu wysyłanego na PORTC jest zerowany i kropka świeci. W przeciwnym wypadku ten bit jest ustawiany i kropka jest gaszona.

Mając do dyspozycji tak działający mechanizm możemy wyświetlać dowolną liczbę 4-cyfrową zapisując tylko bufor led_buf i ewentualnie zmienną led_kr. Wyświetlanie odbywa się automatycznie w tle programu głównego i potrzebuje niewielkich zasobów mikrokontrolera.

Odświeżanie wyświetlania z częstotliwością 1 kHz jest zbyt częste jak dla 4 cyfr. Wystarczyłoby, gdyby cyfry były przełączane z częstotliwością około 200 Hz.

**Tomasz Jabłoński, EP**