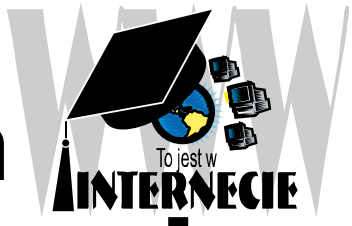
Biblioteki mikroprocesorowych procedur standardowych



Kolejkowana transmisja przez port szeregowy

Przedstawiamy kolejną procedurę, która może być przydatna w samodzielnie tworzonych opracowaniach. Zapewnia ona programową emulację kolejki FIFO dla szeregowej transmisji danych przez port szeregowy UART procesorów pochodnych '51.

Wielu Czytelników stosujących w swoich opracowaniach mikrokontrolery podchodzi z rezerwą do zagadnienia obsługi portu szeregowego w procesorze rodziny '51. Poniżej spróbujemy, przynajmniej częściowo, rozwiać ich obawy.

Niewątpliwą zaletą komunikacji szeregowej jest mała liczba wykorzystywanych linii sygnałowych, zaś jej wadą stosunkowo niska prędkość transmisji. Taki interfejs może więc być idealnym rozwiązaniem we współpracy z powolnymi urządzeniami mechanicznymi (np. drukarki termiczne), czy urządzeniami odległymi, relatywnie rzadko wymagającymi obsługi, np. czujnikami inteligentnymi, które samodzielnie gromadzą i wstępnie obrabiają dane pomiarowe.

Port szeregowy jest w procesorach '51 traktowany jako urządzenie zewnętrzne, czyli podobnie jak timer albo przerwania zewnętrzne. Wszystkie te urządzenia komunikują się z jednostką centralną przy pomocy procedury obsługi przerwań.

Interfejs szeregowy zapewnia transmisję dupleksową, czyli odbiór oraz nadawanie mogą być w pełni niezależe od siebie. Transmisja nie jest kolejkowana, co oznacza, że po odbiorze danej powinna ona być odebrana z interfejsu przed nadejściem kolejnej. Do komunikacji z interfejsem szeregowym służy rejestr specjalny oznaczony symbolicznie *SBUF*. W przypadku nadawania, transmisja jest inicjowana za pomocą dowolnej instrukcji adresującej ten rejestr jako docelowy, natomiast odbiór rozpoczyna się po zajściu określonych warunków zewnętrznych.

Jako linie komunikacyjne ze światem zewnętrznym wykorzystywane są dwie linie portu P3, oznaczone jako RxD (P3.0 - linia przeznaczona do odbioru) oraz TxD (P3.1 - linia przeznaczona do nadawania). Pierwszym bitem przesyłanych danych jest najmłodszy (LSB), ostatnim zaś najstarszy (MSB). Drugim rejestrem specjalnym skojarzonym z interfejsem szeregowym jest rejestr *SCON*, który steruje procesem transmisji.

Mamy do wyboru cztery tryby pracy interfejsu, oznaczone cyframi od 0 do 3. Oto ich skrócony opis działania:

- **Tryb 0** - dane są odbierane i nadawane poprzez RxD, a na TxD panuje sygnał synchronizujący o częstotliwości wynoszącej 1/12 częstotliwości zegara procesora - transmisja jest synchroniczna. Długość słowa danych wynosi 8 bitów.
- **Tryb 1** - dane są odbierane na linii RxD, nadawane poprzez linię TxD. Transmisja jest asynchroniczna, więc dodatkowo pojawiają się: synchronizujący bit startu (zawsze 0) na początku transmisji i na jej końcu bit stopu (zawsze 1). Pomiędzy tymi elementami synchronizującymi występuje 8 bitów danych. Prędkość transmisji jest zależna od ustawienia timera T1, którego sygnały przepełnienia liczników wyznaczają impulsy synchronizujące całą transmisję. Dodatkowo, prędkość transmisji można spowolnić dwukrotnie, zerując bit *SMOD*, który jest najstarszym bitem w rejestrze specjalnym *PCON* (uwaga! rejestr PCON nie jest adresowany bitowo).

- **Tryb 2** - dane są odbierane na linii RxD, nadawane przez linię TxD. Transmisja jest asynchroniczna, ale pomiędzy bitami startu i stopu znajduje się 9 bitów, z czego 8 młodszych bitów to bity danych, bit ostatni jest pobierany z rejestru *SCON* (bit RB8). Bit RB8 może być bitem kontrolnym, np. wcześniej obliczonym bitem parzystości. Prędkość transmisji jest sztywna: 1/32 (SMOD=0) albo 1/64 (SMOD=1) częstotliwości zegara.
- **Tryb 3** - ten tryb różni się od trybu 2 tylko prędkością transmisji, która jest zależna od ustawienia timera T1. Jego sygnały przepełnienia synchronizują transmisję.

Informacje zawarte w rejestrze SCON rozkładają się na jego poszczególne bity w sposób następujący, poczynając od bitu najstarszego.

- **SM0, SM1** - dwa bity kodujące numer jednego z trybów pracy opisanych wyżej.
- **SM2** - zapewnia komunikację wieloprocesorową w systemie.
- **REN** - programowa blokada odbioru.
- **TB8** - dziewiąty bit nadawanego słowa w trybie 2 albo 3.
- **RB8** - dziewiąty bit odebranego słowa w trybie 2 albo 3.
- **TI** - flaga przerwania ustawiana sprzętowo po zakończeniu nadawania, powinna być zerowana w procedurze obsługi przerwania interfejsu szeregowego.
- **RI** - flaga przerwania ustawiana sprzętowo po odebraniu pełnego słowa, powinna być zerowana w procedurze obsługi przerwania interfejsu szeregowego.

Twórcy rodziny '51 uznali, że nie ma istotnej potrzeby oddzielania obsługi przerwania dla danych przychodzących i nadawanych. Układ interfejsu szeregowego ma jedno przerwanie wspólne dla obu kierunków przesyłania danych i tylko na drodze programowej można rozróżnić, co jest przyczyną wywołania przerwania.

Spośród czterech trybów pracy układu transmisji szeregowej wybierzemy do naszych eksperymentów *tryb 1*. Łatwo to uzasadnić. Po pierwsze, interesuje nas osiem bitów danych i nie mamy istotnej potrzeby dokładania dziewiątego bitu jako bitu parzystości. Po drugie - chcemy uzyskać jedną z przyjmowanych najczęściej prędkości transmisji, np. 1200 bitów na sekundę. Po trzecie - dobrze by było, żeby oprócz możliwości poprawnej transmisji szeregowej, nasz system miał jeszcze zdolność pomiaru czasu w jednostkach u nas przyjętych. Zapewni nam to w niezależny sposób kształtowaną częstotliwość synchronizacji portu szeregowego (timer T1) i jednocześnie odliczanie czasu (timer T0).

Argument trzeci doprowadza nas do poważnej decyzji: wyboru częstotliwości zegara. Dobrym wyborem dla odliczania czasu jest typowa częstotliwość 12MHz. Cykl maszynowy wynosi dokładnie 1µs, więc nie ma problemu z przeliczaniem. Gorzej sprawa ma się z doborem parametrów timera T1. Weźmy dla przykładu prędkość 1200 bodów. Przyjmijmy, że timer T1 będzie pracował w trybie 2, czyli z automatycznym przeładowaniem. Wtedy wzór na obliczenie zawartości TH1 jest następujący:

$TH1=256-(2^{SMOD} * f/(32*12*BR))$,

gdzie:

SMOD - stan bitu SMOD,

f - częstotliwość zegara w hercach,

BR - prędkość transmisji w bodach.

Dla 1200 bodów i SMOD=1, TH1 wynosi w przybliżeniu 203,92. Z konieczności musimy przyjąć wartość całkowitą, czyli 204. Jeśli SMOD=0, TH1=229,96, to daje w efekcie TH1=230. Obie wartości TH1 mogą nie zapewnić poprawnego odbioru i należy to sprawdzić. Po przekształceniu powyższego wzoru wartości prędkości transmisji dla SMOD=1 i SMOD=0 wynoszą 1201,92. W porównaniu z prędkością dokładną wartość ta różni się o mniej niż 0,2%. Musimy zwrócić uwagę na charakter transmisji w *trybie 1*: jest to transmisja asynchroniczna, czyli proces synchronizacji zachodzi dla pojedynczego słowa. Ponieważ układ w procesorze '51 próbkuje stan bitów w domniemanym środku impulsu, a bitów mamy 10, dopuszczalna niedokładność odbioru wynosi 5%. Możemy być więc spokojni o poprawność odbioru. Dla większych prędkości transmisji problem doboru wartości w TH1 zaczyna być coraz bardziej uciążliwy.

Oczywiście nic nie stoi na przeszkodzie w doborze takiej częstotliwości, która zapewni dokładną prędkość transmisji szeregowej i jednocześnie będą spełnione dogodne warunki do odliczania czasu. Bardzo dogodna z punktu widzenia układu transmisji szeregowej, proponowana przez producenta procesora, częstotliwość 11,05962MHz tylko z pozoru wydaje się niewdzięczną do pomiaru czasu. Rzeczywiście, czas cyklu nie jest liczbą całkowitą i wynosi około 1,085µs. Tak naprawdę, w procesie odliczania czasu zliczamy cykle procesora. Dla tej częstotliwości w ciągu sekundy zostanie zliczonych dokładnie 11059620/12 = 921635 impulsów. Rozkładając tę liczbę na czynniki dostajemy:

921635 = 5*11*13*1289 = 143*6445

Wpisując do TH0=256-143=113, w każdej procedurze obsługi przerwania zwiększamy parę komórek pamięci modulo 6445 i po każdym przejściu z 6444 na 0 odliczyliśmy 1 sekundę.

Jeśli używamy timera T1, nie możemy zapomnieć o zablokowaniu przerwania od niego. Dobrą zasadą jest zdefiniowanie wektora przerwania, a w procedurze obsługi przerwania powinien być tylko rozkaz powrotu z tej obsługi *RETI*.

Tak oto dobrnęliśmy do zasadniczej części artykułu, po drodze rozwiązując problem doboru parametrów interfejsu szeregowego i zegara, który go synchronizuje.

Kiedy uważniej przyjrzymy się specjalizowanym układom transmisji szeregowej, szybko zauważymy, że wspólną ich cechą jest posiadanie systemu kolejkowania odbieranych i nadawanych znaków. Wprawdzie nie są to stosunkowo długie kolejki (kilka..kilkanaście bajtów), jednak transmisja staje się dużo bardziej wygodna: procesor nie musi intensywnie sprawdzać stanu jednobajtowego bufora komunikacyjnego, a wystarczy, że będzie czynił to rzadziej i z równą skutecznością, bowiem przesłanie ich z/do pamięci operacyjnej może trwać krócej niż transmisja jednego bitu w interfejsie szeregowym. Trzeba bowiem wiedzieć, że nie zabrane z bufora w porę dane ulegną zatarciu przez

kolejne przysyłane bajty.

Doceniając dobrodziejstwa płynące z faktu kolejkowania danych w transmisji szeregowej, udostępnimy procedury tworzące i wykorzystujące taką kolejkę. Rozważmy kolejkę dla nadawania, przez analogię można zbudować kolejkę dla danych odbieranych.

Procesor nie posiada wbudowanych sprzętowo mechanizmów kolejkowania, zatem takie musimy stworzyć programowo. Przyjęta tu idea kolejkowania jest bardzo prosta:

1. Wydzielamy obszar pamięci danych, który będzie adresowany tylko pośrednio, przeznaczony na bufor kolejki i niezbędne wskaźniki.
2. Budujemy procedurę zapisu danej do kolejki w oparciu o wskaźnik początku kolejki i liczbę zapisanych tam znaków.
3. Budujemy procedurę odczytu z kolejki w oparciu o wskaźnik początku kolejki i liczbę zapisanych tam znaków.
4. W obsłudze przerwania portu szeregowego, w części dotyczącej nadawania, umieszczamy procedurę odczytu z kolejki zdefiniowaną w p. 3, z jednoczesnym zapisem do *SBUF*.
5. Za pomocą procedury z p. 2 zapisujemy dane do wysłania.
6. Po zapełnieniu kolejki lub wyczerpaniu wszystkich znaków przeznaczonych do wysłania, uruchamiamy tylko jedną procedurę odczytu z kolejki i te dane wysyłamy do rejestru *SBUF*.
7. Jeśli tylko w kolejce będzie co najmniej jeden znak do wysłania, sprzężenie zwrotne utworzone w p. 4 zapewni ciągłą transmisję, aż do ich wyczerpania.

Przykład takiego rozwiązania przedstawiony został na **list. 1**.

Mamy dwie podstawowe procedury: *DoKolejki* i *Zkolejki*. Rejestrem pośredniczącym z kolejką jest rejestr *R3*. Oprócz tego są dwie zmienne reprezentujące adres komórki zawierającej pierwszy, najwcześniej wprowadzony do kolejki znak (*PoczatekKolejki*) oraz liczbę znaków Kolejki (*LZnakowKolejki*). Te dwie zmienne muszą być wstępnie ustawione przed pierwszym użyciem kolejki. W obie procedury wbudowano prosty mechanizm ochrony kolejki

Listing 1.

```
r0reg                   equ         0
r1reg                   equ         1
r2reg                   equ         2
r3reg                   equ         3
; czesc bitowa (20H-2fH)
            :
            :
PoczNadawania           equ         13h; ustawiana
podczas przeslania pierwszego
    ; bajtu do kolejki portu szeregowego,
zerowana podczas zapisu
    ; pierwszego bajtu
; czesc bajtowa adresowana posrednio (80H-FFH)
WielkoscKolejki         equ         40h
PoczatekKolejki         equ         80h; adres
pierwszego znaku w KolejkaSzer, tuz
                        ; przed wyslaniem
LZnakowKolejki          equ
PoczatekKolejki+1; wskazuje ostatni znak w
                        ; KolejkaSzer
KolejkaSzer  equ        LZnakowKolejki+1; wektor
kolejki portu
                        ; szeregowego

    ORG     0000H
    SJMP    RESTART
    ORG     000BH
    JMP     T0SERVIS
    ORG     23H
    JMP     SerialServis
RESTART:
    :
    :
    :
    ; ustawienie parametrow systemu, w tym
parametrow dla
    ; portu szeregowego
    :
    :
    :
    mov     r0,#PoczatekKolejki
    mov     @r0,#KolejkaSzer
    mov     r0,#LZnakowKolejki
    mov     @r0,#0
    :
    :
;przyklad wyslania do kolejki
    mov     r3,'A'
    call    DoKolejki
    jc      $-5
    mov     r3,'1'
    call    DoKolejki
    jc      $-5
    mov     r3,'a'
    call    DoKolejki
    jc      $-5
    mov     r3,' '
    call    DoKolejki
    jc      $-5
    mov     r3,'m'
    call    DoKolejki
    jc      $-5
    mov     r3,'a'
    call    DoKolejki
    jc      $-5
    mov     r3,' '
    call    DoKolejki
    jc      $-5
    mov     r3,'k'
    call    DoKolejki
    jc      $-5
    mov     r3,'o'
    call    DoKolejki
    jc      $-5
    mov     r3,'t'
    call    DoKolejki
    jc      $-5
    mov     r3,'a'
    call    DoKolejki
    jc      $-5
    mov     r3,'.'
    call    DoKolejki
    jc      $-5
    :
    :
DoKolejki:
; podprogram zapisu bajtu do kolejki portu
szeregowego
; wejscie:
; r3 - bajt do zapisu w kolejce
; wyjscie:
; flaga CY
; CY=1 - blad, wektor kolejki zapelniony, zapis
nie zostal dokonany,
;           r3 nie uległ zmianie
; CY=0 - zapis jest poprawny
; na czas zapisu do wektora KolejkaSzer jest
blokowany dostep
; do procedury obslugi przerwania od T0 i portu
szeregowego
; modyfikacja wektora KolejkaSzer,
; modyfikacja zmiennej LZnakowKolejki
; ewentualna modyfikacja flagi PoczNadawania,
jesli jest to pierwszy
; znak w kolejce
; poza tym uzywa zasobow:
; ACC, r0, r1
    clr     ES
    clr     ET0
    mov     r0,#PoczatekKolejki
    mov     r1,#LZnakowKolejki
    mov     a,@r1
    jnz     DoKol3
    setb    PoczNadawania
DoKol3:
    cjne    a,#WielkoscKolejki,DoKol1
DoKol2:
    setb    c
    sjmp    DoKol6
DoKol4:
    jnc     DoKol5
    sjmp    DoKol7
DoKol1:
    jnc     DoKol2
    mov     a,@r0
    add     a,@r1
    cjne
a,#KolejkaSzer+WielkoscKolejki,DoKol4
DoKol5:
    subb    a,#WielkoscKolejki
DoKol7:
    mov     r0,a
    mov     @r0,r3reg
    inc     @r1
    clr     c
DoKol6:
    setb    ES
    setb    ET0
    ret
SerialServis:
; procedura obslugi przerwania od portu
szeregowego
; zablokowana obsluga odbioru z portu
szeregowego
    clr     RI
    jbc     TI,SS1
    reti
SS1:
    push    acc
    push    psw
    push    r0reg
    push    r1reg
    push    r3reg
    call    ZKolejki
    jc      SS2
    mov     SBUF,r3
SS2:
    pop     r3reg
    pop     r1reg
    pop     r0reg
    pop     psw
    pop     acc
    reti
T0Servis:
; OBSLUGA PRZERWANIA T0
; CHRONI UZYWANE REJESTRY
; PRZERWANIE CO 0.25ms
    PUSH    ACC
    PUSH    PSW
    push    r0reg
    push    dp1
    push    dph
    :
    :
    :
            ; odliczono czas 50ms
    jnb     PoczNadawania,T0S16
    push    r3reg
    call    ZKolejki
    mov     SBUF,r3
    pop     r3reg
    clr     PoczNadawania
T0S16:
    :
    :
    :
    :
    pop     dph
    pop     dp1
    pop     r0reg
    POP     PSW
    POP     ACC
    RETI

ZKolejki:
; podprogram odczytania pierwszego bajtu z
wektora KolejkaSzer
; wejscie: nic
; wyjscie:
; r3 - odczytany znak
; flaga CY:
; CY=1 - blad,zerowa liczba znakow w kolejce,
wartosc r3 nieokreslona
; CY=0 - znak odczytany poprawnie
; modyfikacja wektora KolejkaSzer,
; modyfikacja zmiennej PoczatekKolejki -
zwiekszenie o 1
; modyfikacja zmiennej LZnakowKolejki -
zmniejszenie o 1
; poza tym uzywa zasobow:
; acc, r0, r1
    mov     r0,#PoczatekKolejki
    mov     r1,#LZnakowKolejki
    mov     a,@r1
    jnz     ZKol3
    setb    c
    ret
ZKol3:
    mov     r0reg,@r0
    mov     r3reg,@r0
    dec     @r1
    mov     r0,#PoczatekKolejki
    inc     @r0
    mov     a,@r0
    cjne
a,#(KolejkaSzer+WielkoscKolejki),ZKol1
ZKol4:
    mov     @r0,#KolejkaSzer
    ret
ZKol1:
    jnc     ZKol4
    clr     c
    ret
```

przed wykonaniem operacji błędnych, jakimi niewątpliwie jest zapis do kolejki zapełnionej i odczyt z kolejki pustej. W obydwu przypadkach operacje te nie dochodzą do skutku i jest to sygnalizowane ustawieniem flagi przepełnienia *CY*.

W tym rozwiązaniu zaproponowano zainicjowanie nadawania z kolejki poprzez przerwanie od T0. Co 50ms jest sprawdzany stan flagi *PoczNadawania* i w razie jej ustawienia zostanie wysłany pierwszy znak, uruchamiając proces wysyłania pozostałych znaków. Flaga *PoczNadawania* jest ustawiana przed zapisem do kolejki pustej. Ważnym jest, aby po wysłaniu pierwszego znaku z kolejki zerować tę flagę.

Dla programisty powyższe rozwiązanie jest wygodne: wystarczy wysłać daną do kolejki za pomocą procedury *DoKolejki*, a reszta będzie wykonana „w tle" programu. Ponieważ bufor kolejki ma skończoną długość (w naszym przykładzie przesadnie dużą, wystarczy 8..16bajtów), dobrze jest po wywołaniu procedury *DoKolejki* sprawdzić stan flagi *CY*.

Można oczywiście zrezygnować z przerwania od T0, przesuwając obowiązek inicjacji transmisji na program główny, co może wyglądać następująco (wysyłamy ciąg znaków 'Ala ma kota.'):

```
  mov  r3,'A'
  call DoKolejki
  jc   $-5
  mov  r3,'l'
  call DoKolejki
  jc   $-5
  mov  r3,'a'
  call DoKolejki
  jc   $-5
  mov  r3,' '
  call DoKolejki
  jc   $-5
  mov  r3,'m'
  call DoKolejki
  jc   $-5
  mov  r3,'a'
  call DoKolejki
  jc   $-5
  mov  r3,' '
  call DoKolejki
  jc   $-5
  mov  r3,'k'
  call DoKolejki
  jc   $-5
  mov  r3,'o'
  call DoKolejki
  jc   $-5
  mov  r3,'t'
  call DoKolejki
  jc   $-5
  mov  r3,'a'
  call DoKolejki
  jc   $-5
  mov  r3,'.'
  call DoKolejki
  jc   $-5
  jnb  PoczNadawania,ET1
  call Zkolejki
  clr  PoczNadawania
  mov  SBUF,r3
ET1:
```

Sprawdzenie flagi *PoczNadawania* ma na celu wyeliminowanie zbędnej inicjalizacji nadawania w czasie, kiedy trwał zapis do kolejki ze znakami, czyli nie zostało rozerwane sprzężenie zwrotne poprzez obsługę przerwania od portu szeregowego.

Mirosław Lach, AVT
mlach@polbox.com

Kod źródłowy z list. 1 jest dostępny na internetowej stronie EP, pod adresem: www.avt.com.pl/avt/ep/ftp.