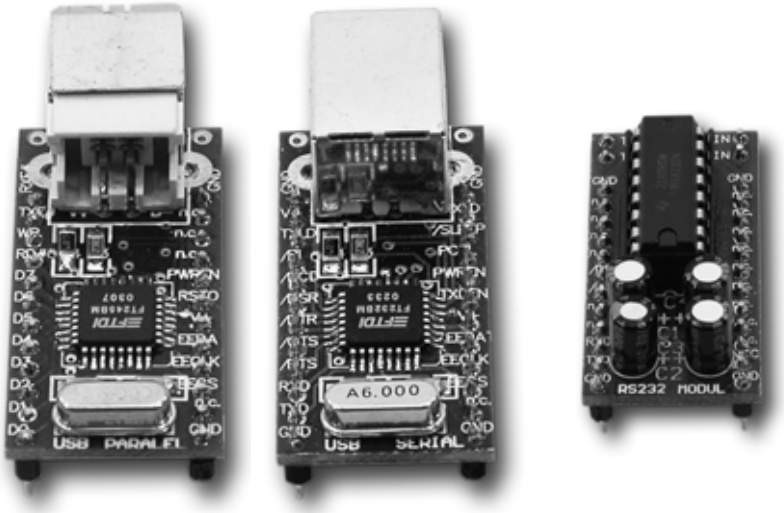
Moduły interfejsów szeregowych

AVT-553/USB232

AVT-553/USB245

AVT-553/RS232



Działanie wielu urządzeń polega na wymianie danych, najczęściej z komputerami. Popularnym i prostym sposobem jest wykorzystanie w tym celu portu szeregowego, który jest dostępny we wszystkich komputerach stacjonarnych lub przenośnych. Do niedawna niepodzielnie panował RS232, teraz coraz częściej jest stosowany interfejs USB. Od strony mikrokontrolera do obsługi takiego połączenia potrzebna jest zarówno część sprzętowa, jak i trochę oprogramowania realizującego transmisję. W artykule pokażemy, jak można to zrobić w urządzeniach opartych na mikrokontrolerach z rodziny '51 i AVR. ***Rekomendacje:*** *informacje przedstawione w artykule przydadzą się konstruktorom samodzielnie budującym urządzenia cyfrowe, które komunikują się z otoczeniem za pomocą połączeń przewodowych.*

Jaki standard wybrać?

Przez długie lata urządzenia zewnętrzne można było podłączyć do komputera na dwa sposoby: albo poprzez równoległy port drukarkowy (Centronics), albo poprzez port szeregowy RS232. Wiele mikrokontrolerów jednoukładowych, w tym prawie wszystkie z rodzin '51 i AVR, posiada mechanizmy ułatwiające wymianę danych w uproszczonym formacie RS232. Oznacza to, że nie ma potrzeby pisania specjalnego oprogramowania przeznaczonego do obsługi transmisji i odbioru ramki pojedynczego znaku. Wystarczy jedynie wpisać właściwe dane do kilku rejestrów i zainicjować odpowiednie przerwania, dalej cała transmisja przebiega w sposób prawie niewidoczny dla głównego programu mikrokontrolera. Niezbędny jest jeszcze konwerter poziomów najczęściej wykorzystujący układ MAX232 lub jego odpowiednik. Ostatnimi laty pojawiła się jednak nowa możliwość - port USB.

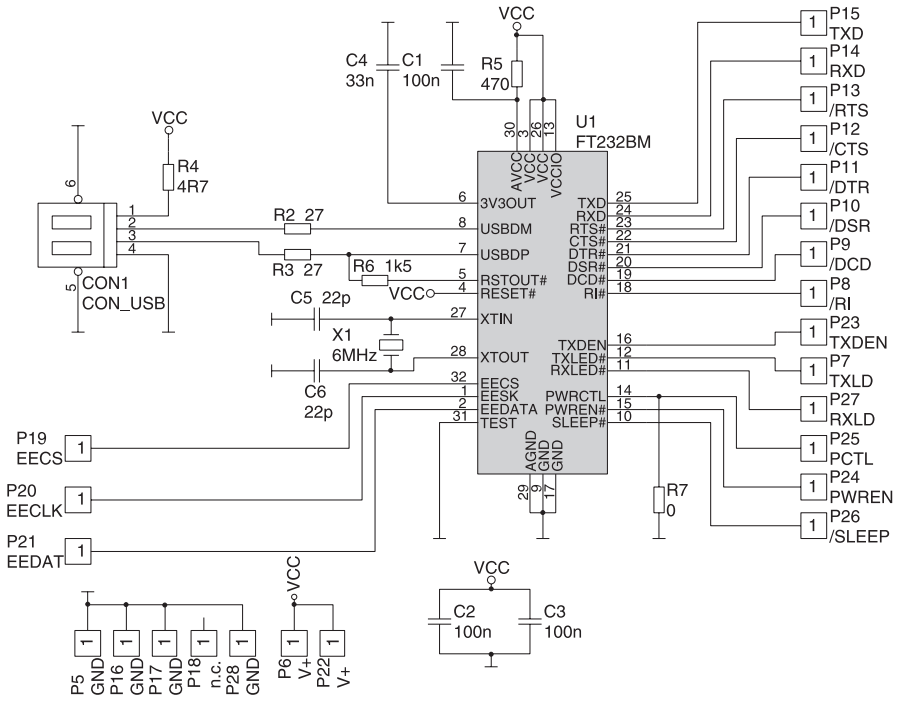
Coraz więcej komputerów PC jest wyposażanych w ten właśnie rodzaj interfejsu. Coraz częściej jest to jedyny, dostępny w komputerze port. USB ma kilka istotnych zalet, np.: dość dużą prędkość transmisji, możliwość podłączania wielu urządzeń peryferyjnych, dostęp do stabilizowanego napięcia +5 V oraz możliwość bezpiecznego dołączania i odłączania urządzenia do/od włączonego komputera.

Bardzo wygodne dla konstruktorów rozwiązanie zaproponowała firma FTDI, wprowadzając na rynek układy interfejsowe FT8U232 i FT8U245. Są to scalone konwertery USB2.0 na doskonale znany RS232 lub wyjście równoległe. Oba te układy były już opisywane w EP i nie ma teraz potrzeby dokładnego omawiania sposobu ich działania. Zajmiemy się natomiast możliwością ich zastosowania jako interfejsów układów z mikrokontrolerami oraz sposobem ich oprogramowania.

Trzy moduły interfejsów

Na **rys. 1, 2** i **3** pokazano schematy trzech modułów interfejsowych, w których zastosowano układy: FT8U232 (moduł *USB232*), FT8U245 (moduł *USB245*) oraz konwerter poziomów MAX232 (moduł *RS232*). Wszystkie moduły mają wymiary nie większe niż standardowy układ scalony w obudowie DIP32 i można je umieścić w podstawce o takim rozmiarze.

Każdy moduł jest kompletnym interfejsem sprzętowym pomiędzy mikrokontrolerem a portem USB lub



Rys. 1. Schemat elektryczny modułu *USB232*

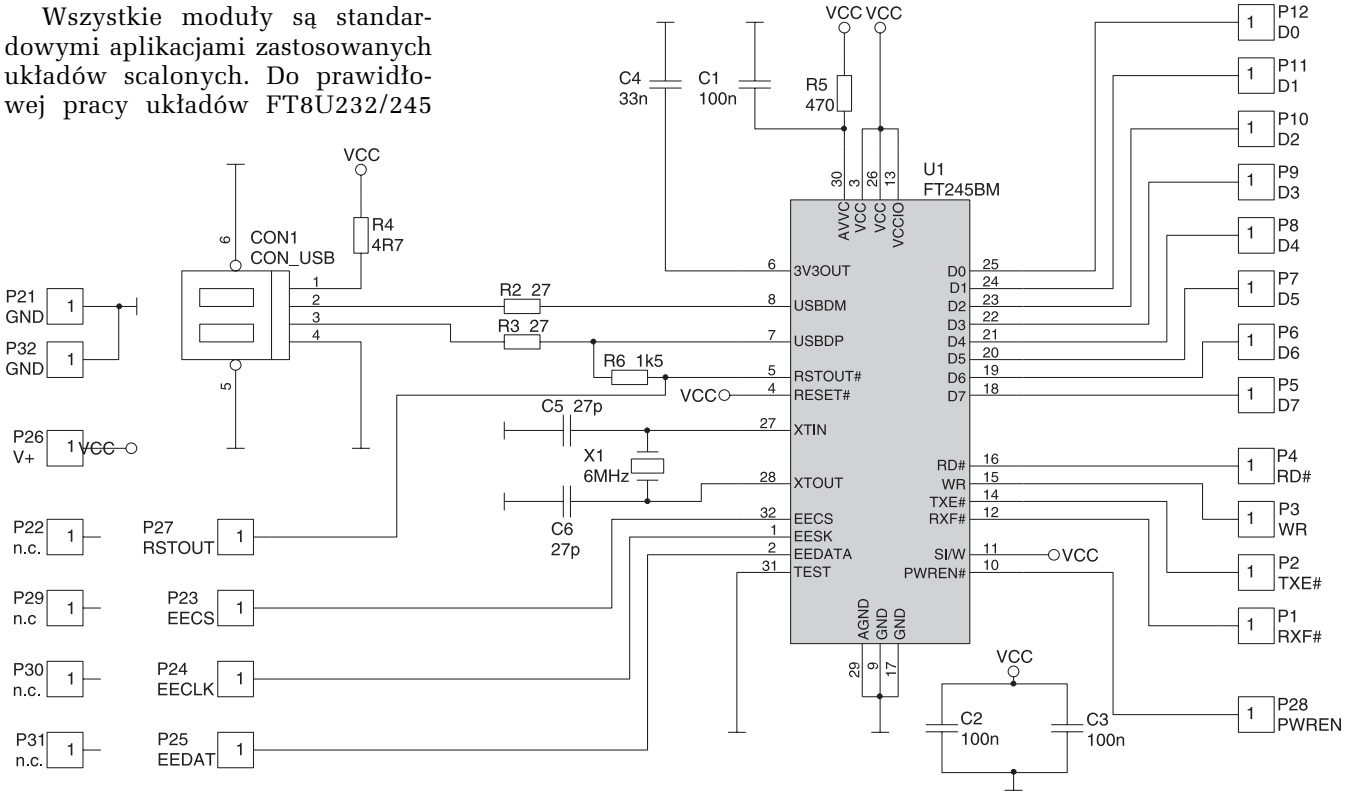
RS232. Moduły były opracowane z myślą o wzajemnej kompatybilności i przykładowo moduł *USB232* można stosować wymiennie z układem *RS232* i w prosty sposób osiągnąć możliwość współpracy mikrokontrolera zarówno z portem USB, jak i RS232 bez konieczności zmian na płytce drukowanej układu mikrokontrolera.

Wszystkie moduły są standardowymi aplikacjami zastosowanych układów scalonych. Do prawidłowej pracy układów FT8U232/245 konieczne jest ich połączenie kablem USB z komputerem, na którym zainstalowane są odpowiednie sterowniki. Moduły zasilane są z portu USB i do ich inicjalizacji nie jest potrzebne włączenie urządzenia, w którym pracują. Co więcej, z wyprowadzeń oznaczonych symbolem *V+* można czerpać niewielki prąd o wartości kilkunastu miliamperów z gniazda USB poprzez opornik zabezpieczający 4,7Ω montowany na płytkach układów *USB245* i *USB232*. Z kolei moduł *RS232* zasilany jest z urządzenia, w którym pracuje. Ponieważ jest to tylko konwerter poziomów TTL/RS232, do jego działania nie jest potrzebna inicjalizacja ani połączenie z komputerem.

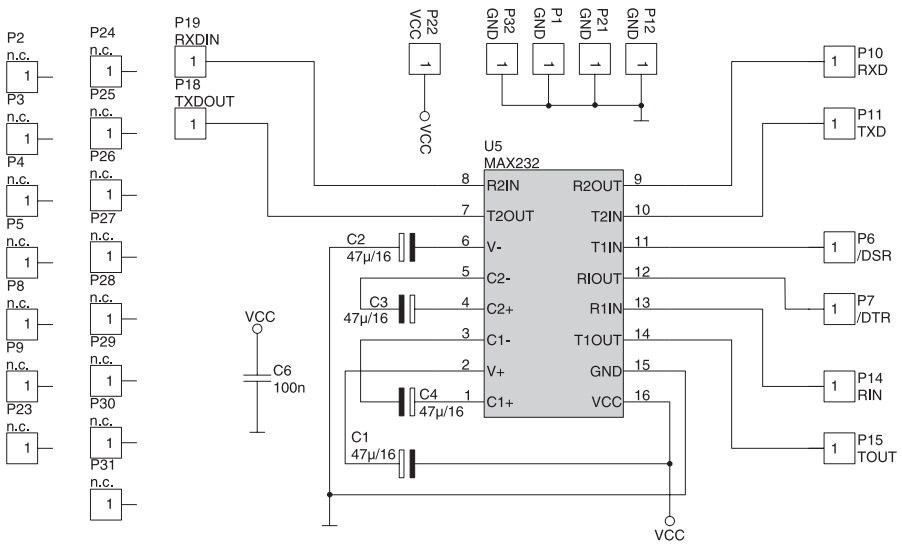
Wyprowadzenia układów *USB232* i *RS232*

Wszystkie układy zostały zaprojektowane tak, aby mogły być umieszczone w typowej podstawce DIP32, co ułatwia ich szybką wymianę. Jednak liczba wyprowadzeń płytki modułu jest mniejsza niż 32. W module *USB232* nie są wyprowadzone styki 13...16 oraz 17...20, natomiast w *RS232* styki 13, 16, 17 i 20.

Moduły *USB232* i *RS232* są ze sobą kompatybilne jeśli chodzi o wyprowadzenia najważniejszych sygnałów. Oznacza to, że można je stosować wymiennie i zależnie od potrzeb wyposażać układ mikrokontrolera w interfejs USB lub RS232. W przypadku modułu



Rys. 2. Schemat elektryczny modułu *USB245*



Rys. 3. Schemat elektryczny modułu *RS232*

USB232 funkcje poszczególnych wyprowadzeń są w zasadzie kopią wyprowadzeń układu FT8U232BM, który stanowi interfejs pomiędzy linią USB a sygnałami interfejsu RS232. Dokładny opis poszczególnych sygnałów można znaleźć w dostępnych opisach standardu RS lub w dokumentacji technicznej układu scalonego do pobrania ze strony producenta *www.ftdichip.com* (materiały tej firmy publikujemy na CD-EP11/2003B).

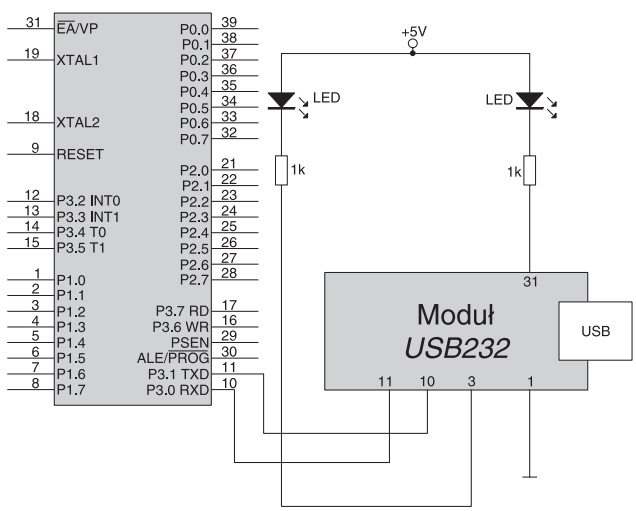
Kompatybilność obydwu modułów istnieje na poziomie najprostszej i najpopularniejszej transmisji portem RS z wykorzystaniem jedynie linii TxD, RxD i masy. Moduł *USB232* posiada wyprowadzenia wszystkich sygnałów RS232 dla złącza DB9. W module *USB232* można wykorzystać dwie linie sterujące (sygnały /DSR i /DTR), co umożliwia sterowanie przepływem danych pomiędzy mikrokontrolerem i komputerem. Na **rys. 4** i **5** pokazano najprostsze warianty połączenia modułów M1 i M3 z mikrokontrolerem typu '51.

Wyprowadzenia układu *USB245*

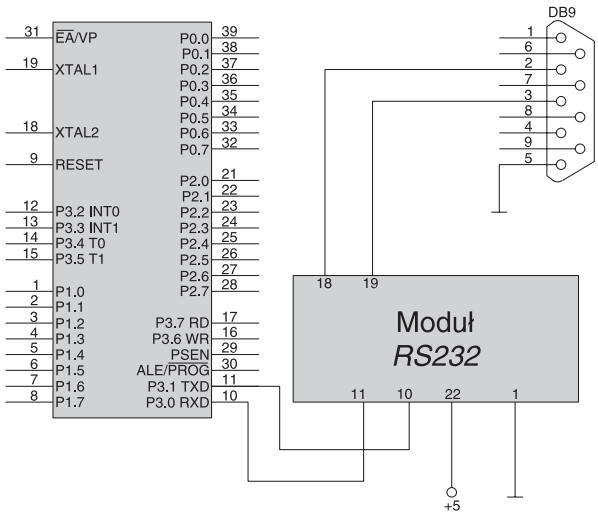
Moduł *USB245* jest interfejsem pomiędzy portem USB a dowolnym 8-bitowym portem mikrokontrolera. Kilka dodatkowych linii steruje przepływem danych pomiędzy modułem a mikrokontrolerem. Funkcje wyprowadzeń modułu przedstawiono w **tab. 1**. Przykładowe połączenie modułu *USB245* z mikrokontrolerem pokazano na **rys. 6**.

Tab. 1. Funkcje wyprowadzeń modułu *USB245*

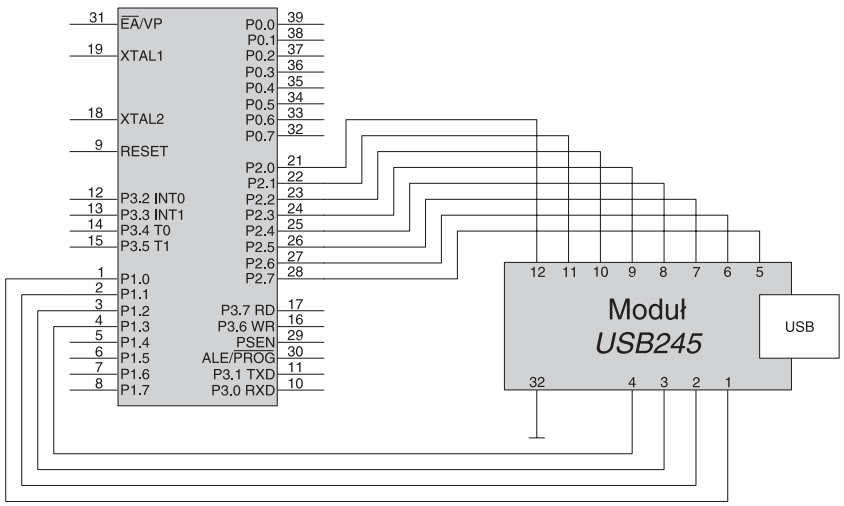
Nr	Nazwa	Kierunek	Opis
1	RXF	OUT	stan niski oznacza, że bajt danych do odczytu z modułu jest dostępny
2	TXE	OUT	stan niski oznacza, że moduł jest gotów do przyjęcia kolejnego bajtu danych
3	WR	IN	zapis do modułu bajtu danych zboczem opadającym
4	RD	IN	odczyt z modułu bajtu danej zboczem narastającym
5	D7	I/O	b.7 bajtu danych
6	D6	I/O	b.6 bajtu danych
7	D5	I/O	b.5 bajtu danych
8	D4	I/O	b.4 bajtu danych
9	D3	I/O	b.3 bajtu danych
10	D2	I/O	b.2 bajtu danych
11	D1	I/O	b.1 bajtu danych
12	D0	I/O	b.0 bajtu danych
21	GND masa		
22	n.c.		
23	EECS	I/O	sygnał selekcji, jeśli stosowany jest zewnętrzny EEPROM
24	EECLK	OUT	zegar transmisji danych, jeśli stosowany jest zewnętrzny EEPROM
25	EEDAT	I/O	sygnał danych, jeśli stosowany jest zewnętrzny EEPROM
26	V+	OUT	zasilanie z gniazda USB
27	RSTOUT	OUT	wyjście wewnętrznego generatora resetu
28	PWREN	OUT	stan niski w czasie konfiguracji modułu
29	n.c.		
30	n.c.		
31	n.c.		
32	GND masa		



Rys. 4. Sposób dołączenia modułu *USB232* do mikrokontrolera



Rys. 5. Sposób dołączenia modułu *RS232* do mikrokontrolera



Rys. 6. Sposób dołączenia modułu *USB245* do mikrokontrolera

Sterowniki

Układy interfejsowe FT8U232BM i FT8U245BM, aby móc prawidłowo pracować, wymagają zainstalowania specjalnych sterowników na komputerze PC, do którego są podłączane. Sterownik zostaje uaktywniony w momencie dołączenia do gniazda USB modułu z wymienionymi układami. Działanie sterownika powoduje, że port USB widziany jest w systemie komputerowym jako kolejny port COM obsługiwany w taki sam sposób jak wszystkie inne porty RS232. Dzięki temu programy potrafiące obsługiwać porty COM będą mogły korzystać z USB bez konieczności jakiejkolwiek przeróbki.

Sterowniki dla różnych systemów operacyjnych nieodpłatnie udostępnia firma FTDI na stronie internetowej *www.ftdichip.com/FTDriver.htm*. Ich instalacja jest bardzo prosta. Po ściągnięciu pliku sterownika odpowiedniego do wersji systemu operacyjnego, w którym będzie pracował, należy plik rozpakować do tymczasowego katalogu na dysku. Następnie za pomocą kabla należy połączyć moduł z komputerem. Po wykryciu dołączonego modułu system operacyjny automatycznie przeprowadza instalację sterownika, należy tylko podać katalog, w którym znajdują się rozpakowane pliki. Każde kolejne dołączanie modułu do komputera będzie już automatycznie uaktywniało odpowiedni sterownik, który już będzie w systemie.

Współpraca mikrokontrolerów z modułami

Moduły *USB232* i *USB245* umożliwiają korzystanie z portu USB na nieco odmienne sposoby. Pierwszy - w przypadku zastosowania modułu *USB245* - jest prostszy. Sprowadza się do badania stanu linii RXF i TXE sygnalizujących stan gotowości modułu do transmisji lub konieczność odczytu z modułu kolejnego bajtu. Niski stan obydwu linii sygnalizuje gotowość do kolejnej transmisji lub zakończenie odbioru bajtu. W przypadku transmisji mikrokontroler powinien wysłać na linie D0...7 poziomy odpowiadające wysyłanemu bajtowi i wygenerować ujemny impuls na linii WR. W przypadku odczytu dana pojawi się na liniach D0...7 po podaniu stanu niskiego na linię RD.

Wszystko na CD-EP11/2003B

Najnowsze wersje sterowników, noty katalogowe oraz oprogramowanie narzędziowe dla układów produkowanych przez firmę FTDI publikujemy na CD-EP11/2003B.

Taka prostota okupiona jest jednak koniecznością zaangażowania aż 12 linii portów. Jeżeli w mikrokontrolerze brakuje wolnych linii, które można przeznaczyć do tego celu, trzeba zastosować szeregową transmisję i moduł *USB232*. W mikrokontrolerach z rodzin '51 lub AVR wydzielono dwie linie sterujące, które oprócz uniwersalnych zastosowań jako wejścia/wyjścia mogą także pełnić funkcję wyprowadzeń sprzętowego interfejsu szeregowego. Linie te oznaczone są symbolami TxD i RxD (lub podobnymi). Pierwsza przeznaczona jest do wysyłania danych w trybie szeregowym, a druga do odbioru danych. Aby je uaktywnić, należy wpisać odpowiednie wartości do kilku rejestrów mikrokontrolera oraz napisać trochę oprogramowania do obsługi transmisji szeregowej przez mikrokontroler.

Ważną decyzją jest określenie sposobu obsługi transmisji. Można zaprogramować mikrokontroler tak, aby sygnalizował jedynie odczyt bajtu lub gotowość do wysłania kolejnego, a program główny będzie musiał sam zadbać o właściwą reakcję na zasygnalizowane w ten sposób zdarzenia. Nie jest to najwygodniejsze rozwiązanie, gdyż zmusza główny program do ciągłego sprawdzania, czy nowe dane już się pojawiły lub czy można już wysłać resztę. W dodatku jeżeli przegapi się odpowiedni moment, kolejny przychodzący bajt może być wpisany na miejsce poprzedniego jeszcze nieodebranego przez program główny zajęty np. długotrwałymi obliczeniami. Znacznie wygodniej zdać się na obsługę transmisji przez przerwania -małe podprogramy działające niezależnie od programu głównego w tych momentach, gdy trzeba coś zrobić z transmisją szeregową. Takie oprogramowanie mikrokontrolera nie jest wcale trudne i pokażemy, jak można to zrobić.

W przykładach zostaną użyte dwa bufory: nadawczy i odbiorczy o rozmiarze 8 bajtów każdy. Poza tym będą wykorzystywane 3 rejestry mikrokontrolera dla liczników bajtów w buforze nadawczym i odbiorczym, wskaźnika aktualnej pozycji w rejestrze nadawczym oraz dwa bity flag sygnalizujących fakt opróżnienia bufora nadawczego i umieszczenia w buforze odbiorczym kolejnego bajtu odebranego portem szeregowym.

Jak to zrobić w programie dla '51?

Przykład oprogramowania nadzorującego transmisję w mikrokontrolerze z grupy '51 zaczniemy

List. 1. Program obsługi przerwania od portu szeregowego

```
;przerwanie portu szeregowego
;--------------------
Rs_przerwanie:
 push acc       ;zachowanie zawartości rejestrów
 push r0        ;użytych w procedurze przerwania
 jb scon.1,rs_t ;przerwanie zostało spowodowane wysłaniem poprzedniego bajtu
                ;obsługa odczytu kolejnego bajtu
 clr scon.0     ;zerowanie flagi odbioru bajtu w rejestrze SCON
 mov a,#bufor_odbiorczy
 add a,licznik_odebranych_bajtów
 mov r0, a      ;obliczenie adresu do zapisu w buforze odbiorczym
 mov @r0,sbuf   ;odebrany bajt z rejestru SBUF do buforu odbiorczego
 inc licznik_odebranych_bajtów      ;zwiększenie licznika bajtów w buforze
 setb rs_rec_flag      ;ustawienie flagi odbioru bajtu
 jmp rs_t1

rs_t:           ;obsługa wysłania kolejnego bajtu
 clr scon.1     ;w rejestrze SCON zerowanie flagi zakończenia transmisji bajtu
 mov a,licznik_wysylanych_bajtow
 jnz rs_t2
                ;wszystkie bajty zostały wysłane
 setb rs_trans_flag       ;ustawienie flagi sygnalizującej wysłanie wszystkich znaków
 jmp rs_t1
rs_t2:
 mov a,wskaznik_pozycji_w_buforze
 add a,#bufor_nadawczy    ;pozycja w buforze kolejnego bajtu do wysłania
 mov r0,a
 mov a,@r0      ;pobranie kolejnego znaku do wysłania
 mov sbuf,a     ;znak do rejestru wyjściowego SBUF
 inc wskaznik_pozycji_w_buforze
 dec licznik_wysylanych_bajtow
rs_t1:
 pop r0         ;odtworzenie stanu rejestrów
 pop acc        ;używanych w procedurze przerwania
 reti
```

List. 2. Program konfigurujący UART

```
;podprogram inicjacji transmisji szeregowej
;9600 bodów, 8 bitów danych, 1 bit stopu
;dla kwarcu mikrokontrolera 11,059MHz
;---------------------
Inicjacja:
 mov SCON, #50h          ;ustawienie w rejestrze SCON trybu1: 8 bitów danych 1 bit stopu
 mov TMOD, #20h          ;transmisja będzie taktowana zegarem T1
 mov TH1, #0FDh          ;wartości początkowe wpisywane do rejestrów
 mov TL1, #0FDh          ;zegara T1 dla szybkości transmisji 9600 bodów
 anl PCON, #07Fh         ;zerowanie bitu SMOD w rejestrze PCON
 setb TR1                ;włączenie zegara T1
 setb ES                 ;zezwolenie na przerwanie portu szeregowego
 setb EA                 ;globalne zezwolenie na przerwania
 clr rs_rec_flag         ;kasowanie flagi sygnalizującej odbiór bajtu
 mov licznik_odebranych_bajtów, #0     ;zerowanie licznika odebranych bajtów
 ret
```

omawiać od końca, czyli od podprogramu obsługi przerwania.

W '51 istnieje tylko jedno przerwanie związane z transmisją szeregową. Na początku trzeba więc ustalić, czy przerwanie zostało wywołane odbiorem kolejnego znaku, czy zakończeniem wysyłania poprzedniego. I zależnie od tego albo odczytany bajt „wyląduje" w buforze odbiorczym, albo kolejny bajt zostanie pobrany z bufora nadawczego i wysłany w świat. Przykładowy program realizujący to zadanie pokazano na **list. 1**.

Zanim przerwanie zacznie funkcjonować, należy najpierw ustawić parametry transmisji szeregowej: szybkość, liczbę bitów startu i stopu, zainicjalizować rejestry i flagi związane z transmisją oraz włączyć obsługę przerwań. Dla wygody najlepiej to zrobić w osobnym podprogramie, na przykład takim jak pokazano na **list. 2**.

Szybkość transmisji portu szeregowego zależy od 3 czynników: częstotliwości kwarcu użytego w oscylatorze mikrokontrolera, wartości początkowej wpisanej do licznika T1 i ustawienia bitu SMOD w rejestrze PCON. W praktyce oznacza to, że niektórych szybkości transmisji nie można osiągnąć, jeśli częstotliwość kwarcu będzie zbyt mała. Standardowe szybkości transmisji w zakresie 1200...19200 bodów można osiągnąć stosując kwarc o częstotliwości 11,059 MHz. W **tab. 2** zestawiono wartości początkowe wpisywane do zegara T1 i ustawienie bitu SMOD dla poszczególnych szybkości.

W przypadku zastosowania innego kwarcu skazani jesteśmy na eksperymentalne dobieranie wartości początkowej wpisywanej do zegara T1. Program należy jeszcze uzupełnić o deklarację użytych rejestrów przeznaczonych do obsługi transmisji, a także wpisać pod odpowiednim adresem wektor przerwania portu szeregowego, co pokazano na **list. 3**.

Na **list. 4** pokazano kilka linii kodu, które powinny znaleźć się w programie głównym, aby mieć dostęp do danych odebranych

Tab. 2. Wartości wpisywane do T1 dla różnych prędkości transmisji dla częstotliwości kwarcu 11,059 MHz

Szybkość	SMOD	Wartość wpisywana do T1
19200	1	#FDh
9600	0	#FDh
4800	0	#Fah
2400	0	#F4h
1200	0	#E8h

List. 3. Deklaracja rejestrów przeznaczonych do obsługi transmisji oraz ustalenie wektora przerwania portu szeregowego

```
licznik_odebranych_bajtów    EQU 30H      ;licznik bajtów w buforze odbiorczym
licznik_wysylanych_bajtow    EQU 31H      ;licznik bajtów w buforze nadawczym
wskaznik_pozycji_w_buforze   EQU 32H      ;wskaźnik pozycji w buforze nadawczym
flagi                        EQU 20h      ;adres bajtu flag
rs_rec_flag                  BIT flagi.0  ;flaga sygnalizacji odbioru bajtu
rs_trans_flag                BIT flagi.1  ;flaga zakończenia transmisji bajtów z buforu nadawczego
ROZMIAR_BUFOROW              EQU 8        ;deklaracja rozmiaru buforów
bufor_odbiorczy:   DS ROZMIAR_BUFOROW
bufor_nadawczy:    DS ROZMIAR_BUFOROW
ORG 23H
jmp Rs_przerwanie                         ;wektor przerwania portu szeregowego
```

List. 4.

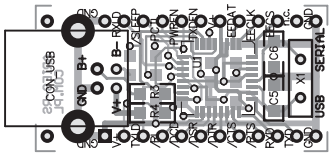
```
mov c, rs_rec_flag
jnc bufor_pusty          ;w buforze odbiorczym nie ma nowych bajtów
clr ES                   ;zablokowanie przerwania portu szeregowego
mov r0,# bufor_odbiorczy
petla_odczytu:
mov a, @r0               ;w rejestrze akumulatora bajt odczytany z bufora
...                      ;tutaj kod programu związany z wykorzystaniem bajtów
...                      ;przesłanych portem szeregowym z PC-ta
 inc r0
 djnz licznik_odebranych_bajtów, petla_odczytu
 clr rs_rec_flag         ;kasowanie flagi sygnalizującej odbiór danych
 setb ES                 ;ponowne włączenie przerwania
bufor_pusty:
Inicjacja transmisji portem szeregowym może wyglądać następująco:
...                      ;kod programu zapisujący do buforu nadawczego bajty danych
...                      ;do wysłania portem szeregowym
 mov licznik_wysylanych_bajtow, #ile_bajtów   ;wpisanie do licznika ilości
;wysyłanych bajtów
 mov wskaznik_pozycji_w_buforze, #0
 mov r0,#bufor_nadawczy
 mov a,@r0               ;w akumulatorze pierwszy bajt do wysłania
 inc wskaznik_pozycji_w_buforze
 dec licznik_wysylanych_bajtow
 mov SBUF, a             ;zapis do SBUF pierwszego bajtu inicjuje transmisję
                         ;zawartości bufora nadawczego
 clr rs_trans_flag
```

List. 5. Program odpowiadający za obsługę przerwania

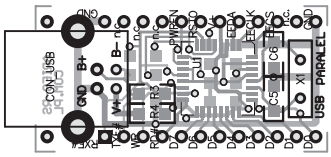
```
;przerwanie odebrania kolejnego znaku z portu RS
;------------------------
UART_RXC:
 push temp                ;zachowanie zawartości rejestrów
 push ZH                  ;użytych w procedurze przerwania
 push ZL
 in temp, SREG
 push temp

 ldi ZH,HIGH(bufor_odbiorczy)
 ldi ZL,LOW(bufor_odbiorczy)
 add ZL, licznik_odebranych_bajtów
 clr temp                 ;obliczenie pozycji w buforze odbiorczym
 adc ZH,temp              ;do zapisu odebranego bajtu
 in temp, UDR             ;w rejestrze temp odebrany bajt
 st Z,temp                ;zapis bajtu w buforze odbiorczym
 inc licznik_odebranych_bajtów   ;zwiększenie licznika bajtów w buforze
 sbr flagi, 1<< rs_rec_flag      ;ustawianie flagi sygnalizującej odbiór bajtu

 pop temp                 ;odtworzenie stanu rejestrów
 out SREG, temp           ;używanych w procedurze przerwania
 pop ZL
 pop ZH
 pop temp
 reti
;przerwanie zakończenia transmisji kolejnego znaku
;-------------------------
UART_TXC:
 push temp                ;zachowanie zawartości rejestrów
 push ZH
 push ZL
 in temp, SREG
 push temp

 cpi licznik_wysylanych_bajtow, 0
 brne uat1
                          ;wszystkie znaki zostały wysłane
 sbr flagi, 1<< rs_trans_flag       ;ustawienie flagi sygnalizującej wysłanie
;wszystkich znaków
 rjmp uat2
uat1:                     ;transmisja kolejnego znaku
 ldi ZH, HIGH(bufor_nadawczy)
 ldi ZL, LOW(bufor_nadawczy)
 add ZL, wskaznik_pozycji_w_buforze
 clr temp
 adc ZH, temp             ;pozycja w buforze kolejnego bajtu do wysłania
 ld temp, Z
 out UDR, temp            ;wysyłany znak do rejestru transmisji UART
 inc wskaznik_pozycji_w_buforze
 dec licznik_wysylanych_bajtow
uat2:
 pop temp
 out SREG, temp
 pop ZL
 pop ZH
 pop temp
 reti
```

List. 6. Podprogram inicjacji parametrów transmisji szeregowej

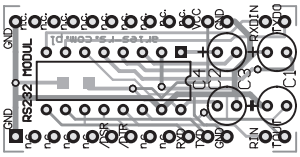
```
;podprogram inicjacji transmisji szeregowej
;19200 bodów, 8 bitów danych, 1 bit stopu
;dla kwarcu mikrokontrolera 10MHz
;----------------------
Inicjacja:

 ldi temp, 32             ;wartość podzielnika dla szybkości 19200
 out UBRR, temp
 sbi UCR, TXEN            ;włączenie części nadawczej
 sbi UCR, RXEN            ;włączenie części odbiorczej
 sbi UCR, RXCIE           ;zezwolenie na przerwanie po odbiorze znaku
 sbi UCR, TXCIE           ;zezwolenie na przerwanie po zakończeniu nadawania znaku
 cbr flagi, 1<< rs_rec_flag  ;kasowanie flagi sygnalizującej odbiór znaku
 sei                      ;ogólne zezwolenie na przerwania
 ret
```



Rys. 7. Schemat montażowy modułu *USB232*



Rys. 8. Schemat montażowy modułu *USB245*



Rys. 9. Schemat montażowy modułu *RS232*

Ustawienie parametrów transmisji, tak jak i w poprzednim przykładzie, odbywa się za pomocą podprogramu inicjalizacji, którego listing pokazano na **list. 6**.

Ze względu na większą szybkość działania mikrokontrolerów AVR, łatwiej jest osiągnąć standardowe szybkości transmisji bez względu na rodzaj zastosowanego kwarcu w generatorze. Wzór pozwalający wyliczyć szybkość transmisji w zależności od częstotliwości kwarcu i wartości wpisanej do rejestru UBRR ma postać:

BAUD =fclk/16*(UBRR-1),

gdzie:

BAUD - szybkość transmisji w bitach/sekundę,

fclk - częstotliwość oscylatora mikrokontrolera,

UBRR - wartość wpisana do rejestru.

portem szeregowym oraz instrukcje inicjujące transmisję portem szeregowym.

Ustawienie w stanie wysokim flagi *rs_trans_flag* będzie sygnalizowało zakończenie transmisji zawartości bufora nadawczego.

Jak to zrobić w programie dla mikrokontrolera AVR?

Podobny program dla mikrokontrolera z grupy AVR będzie wyglądał trochę inaczej. Powodem jest zarówno inna składnia asemblera, jak i różnice konstrukcyjne obydwu mikrokontrolerów. Na przykład mikrokontrolery AVR oferują więcej przerwań, w tym osobne dla odczytu bajtu i osobne dla zakończenia transmisji bajtu. Tak jak w przypadku '51 zacznijmy od obsługi przerwania (**list. 5**).

List. 7. Deklaracja zastosowanych rejestrów AVR i ustawienie wektorów skoków

```
.DEF licznik_odebranych_bajtów  =r16   ;licznik bajtów w buforze odbiorczym
.DEF licznik_wysylanych_bajtow  =r17   ;licznik bajtów w buforze nadawczym
.DEF wskaznik_pozycji_w_buforze =r18   ;wskaźnik pozycji w buforze nadawczym
.DEF flagi                      =r19   ;adres bajtu flag
.DEF temp                       =r20   ;rejestr pomocniczy
.EQU rs_rec_flag                =0     ;flaga sygnalizacji odbioru bajtu
.EQU rs_trans_flag              =1     ;flaga zakończenia transmisji bajtów z buforu nadawczego
.EQU ROZMIAR_BUFOROW            =8     ;deklaracja rozmiaru buforów
.DSEG
bufor_odbiorczy:    .BYTE ROZMIAR_BUFOROW
bufor_nadawczy:     .BYTE ROZMIAR_BUFOROW

.ORG 0x09
 rjmp UART_RXC                         ;wektor przerwania odbioru kolejnego znaku z RS-a
.ORG 0x0b
 rjmp UART_TXC                         ;wektor przerwania ukończenia transmisji kolejnego znaku
```

List. 8. Program odpowiadający za dostęp do przesłanych danych

```
 sbrs flagi, rs_rec_flag
 rjmp bufor_pusty            ;w buforze odbiorczym nie ma nowych bajtów
 ldi ZH,HIGH(bufor_odbiorczy)
 ldi ZL,LOW(bufor_odbiorczy)
 cbi UCR, RXCIE              ;wyłączenie zezwolenia na przerwanie po odbiorze znaku
petla_odczytu:
 ld temp,Z+                  ;w temp bajt odczytany z buforu odbiorczego
...                          ;tutaj kod programu związany z wykorzystaniem bajtów
...                          ;przesłanych portem szeregowym z PC-ta
 dec licznik_odebranych_bajtów
 brne petla_odczytu
 cbr flagi, 1<< rs_rec_flag  ;kasowanie flagi sygnalizującej odbiór bajtu
 sbi UCR, RXCIE              ;włączenie zezwolenia na przerwanie po odbiorze znaku

bufor_pusty:

Inicjacja transmisji:
...                          ;kod programu zapisujący do buforu nadawczego bajty danych
...                          ;do wysłania portem szeregowym
 ldi licznik_wysylanych_bajtow, #ile_bajtów   ;wpisanie do licznika ilości
;wysyłanych bajtów
 clr wskaznik_pozycji_w_buforze
 ldi ZH,HIGH(bufor_nadawczy)
 ldi ZL,LOW(bufor_nadawczy)
 ld temp,Z                   ;w temp pierwszy wysyłany bajt z bufora nadawczego
 dec licznik_wysylanych_bajtow
 inc wskaznik_pozycji_w_buforze
 cbr flagi, 1<< rs_trans_flag    ;kasowanie flagi sygnalizującej wysłanie
;wszystkich znaków
  out UDR, temp              ;zapis do SBUF pierwszego bajtu inicjuje transmisję
                             ;zawartości bufora nadawczego
```

Deklarację zastosowanych rejestrów i ustawienie wektorów skoków pokazano na **list. 7**.

Z kolei dostęp do przesłanych poprzez RS danych i inicjacja transmisji w programie głównym mogą wyglądać tak, jak pokazano na **list. 8**. Ustawienie w stanie wysokim flagi *rs_trans_flag* będzie sygnalizowało zakończenie transmisji zawartości bufora nadawczego.

Montaż modułów

Schematy montażowe modułów pokazano na **rys. 7, 8** i **9**.Użyte do budowy modułów rezystory i kondensatory unipolarne mają obudowy przeznaczone do montażu powierzchniowego typu 1206. W module *USB232* rezystor R7 ma wartość 0Ω. Oczywiście zamiast niego można zastosować zwykłą zworę. Pojemność elektrolitów w module *RS232* nie jest krytyczna i może być zmniejszona nawet do 1µF. Może to tylko skutkować nieco mniejszą maksymalną długością kabla, jakiego można będzie użyć do połączenia modułu z portem COM komputera. W modułach *USB232* i *USB245* najlepiej zastosować kwarce w niskich obudowach. Jeżeli moduły nie będą przewidziane do mocowania w podstawkach, można zrezygnować ze stosowania specjalnych pinów, używając w to miejsce srebrzanki do wlutowania na stałe modułu do głównej płytki układu, w którym będą pracować.

Ryszard Szymaniak, EP
ryszard.szymaniak@ep.com.pl

Wzory płytek drukowanych w formacie PDF są dostępne w Internecie pod adresem: ***pcb.ep.com.pl*** *oraz na płycie CD-EP11/2003B w katalogu* ***PCB****.*

WYKAZ ELEMENTÓW

Elementy modułu USB232

Rezystory
R2, R3: 27Ω
R4: 4,7Ω
R5: 470Ω
R6: 1,5kΩ
R7: 0
Kondensatory
C1...C3: 100nF
C4: 33nF
C5, C6: 22pF
Półprzewodniki
U1: FT232BM
Różne
CON_USB: gniazdo USB-A
X1: 6MHz

Elementy modułu USB235

Rezystory
R2, R3: 27Ω
R4: 4,7Ω
R5: 470Ω
R6: 1,5kΩ
Kondensatory
C1...C3: 100nF
C4: 33nF
C5, C6: 27pF
Półprzewodniki
U1: FT245BM
Różne
CON_USB: gniazdo USB-A
X1: 6MHz

Elementy modułu RS232

Kondensatory
C1...C4: 47µF/16V
C6: 100nF
Półprzewodniki
U5: MAX232