

W bascomowym „kąciku” będziemy się starać w miarę przystępnie przedstawiać rozwiązania problemów napotykanym przez naszych Czytelników podczas pisania programów w Bascomie. Rubryka ta powstała z myślą o rozwiązywaniu problemów, na jakie najczęściej napotykają programiści, zatem zachęcamy wszystkich Czytelników do zgłaszania problemów na jakie się natknęli podczas tworzenia własnych programów.

## Obsługa interfejsu RS232

Na początek parę słów wstępu na temat interfejsu RS232, który jest spotykany w większości typowych aplikacji mikrokontrolerowych. Najczęściej w praktyce są spotykane problemy wynikają z błędnego założenia, że mikrokontroler wystarczy „wprost” połączyć trzema przewodami z RS-em komputera i to wystarczy do wymiany danych. Trzeba jednak wziąć pod uwagę, że w liniach interfejsu RS232 występują inne napięcia niż typowe dla standardu TTL - mikrokontroler jako logiczne „0” traktuje napięcie 0...0,8 V, natomiast logiczną „1” jest napięcie +2,0...5 V. Zgodnie z zaleceniami normy opisującej RS232 napięcia dla logicznych „0” i „1” wynoszą odpowiednio >+3 V oraz <-3 V. Żeby uruchomić komunikację między procesorem a komputerem niezbędny jest konwerter napięć ze standardu TTL na RS232. Najbardziej popularnym scalonym konwerterem jest układ MAX232 i jego odpowiedniki, które mają wbudowaną ładunkową przetwornicę napięcia. Czasami można sobie także poradzić stosując różne sztuczki np. robiąc interfejs na tranzystorach. Na rys. 1 przedstawiono schemat typowego układu

konwertera TTL<->RS232.

Tu mała uwaga - na przedstawionym schemacie, jak i w notach aplikacyjnych układów, podawane są różne wartości kondensatorów wymaganych do poprawnej pracy układu (od 0,1 µF do 10 µF). Przed zmontowaniem układu należy sprawdzić w nocie katalogowej stosowanego układu, kondensatory o jakiej pojemności należy zastosować.

Jeszcze jedna uwaga na temat połączenia mikrokontrolera z interfejsem RS znajdującym się w komputerze: masy łączy się razem, ale styki oznaczone TxD (*Transmit Data* - nadawanie danych) oraz RxD (*Receive Data* - odbiór danych) łączymy „na krzyż”: TxD mikrokontrolera łączymy ze stykiem RxD komputera i na odwrót.

Aby skorzystać z transmisji RS232, na początek należy skonfigurować UART/USART mikrokontrolera (wymaga to napisania odpowiedniego programu) oraz w komputerze - należy ustawić jednakowe parametry transmisji (jednakową prędkość oraz pozostałe parametry, np.: 8 bitów danych, 1 bit stopu, bez parzystości - co często jest zapisywane jako 8n1). W Bascomie do deklaracji prędkości

pracy interfejsu jest używana następująca instrukcja dyrektywy kompilatora:

```
$baud = 9600
```

Należy ją użyć na początku programu (prędkość można także ustawić w menu *Options>Compiler>Communication*, ale lepiej zadeklarować w programie). Bardzo ważną rzeczą jest także poznanie zależności szybkości RS-a od częstotliwości użytego kwarcu. Najłatwiej sprawdzić to w Bascomie ustawiając w menu *Options>Compiler>Communication* żadaną prędkość transmisji oraz częstotliwość kwarcu w dolnej części okna (ramka *ERROR*) będziemy mieli podaną średnią stopę błędów lub napis *Not Possible*, czyli nie możliwe do ustawienia - tak jest np. dla 1200 bodów i kwarcu powyżej 4 MHz. Złe dopasowanie częstotliwości kwarcu do wymaganej prędkości transmisji jest jedną z częstych przyczyn błędów lub wręcz braku transmisji.

W tab. 1 przedstawiono niektóre zależności błędów dla danych częstotliwości kwarców przy różnych prędkościach.

Podstawowym poleceniem w Bascomie służącym do wysłania danych z mikrokontrolera przez RS232 jest instrukcja *print*. Używa się jej w najprostszym z możliwych sposobów, czyli:

```
Print zmienna
```

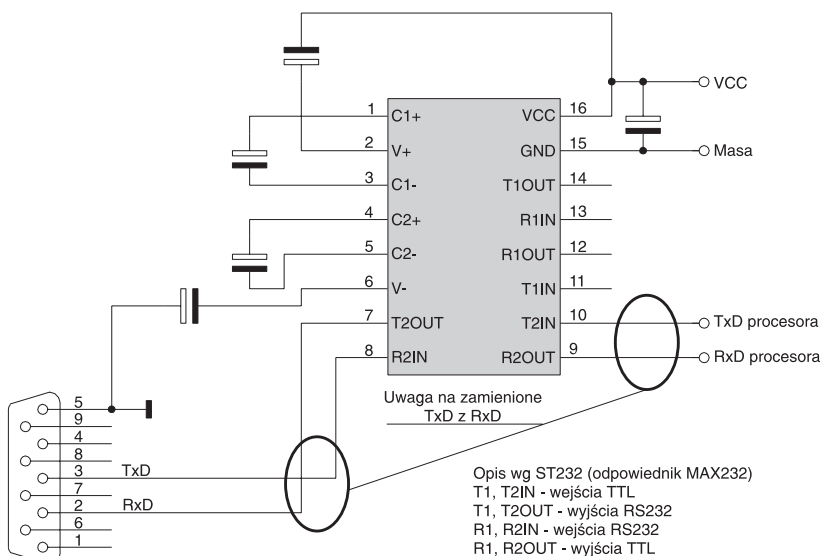
gdzie *zmienna* to dowolna zmienna (lub stała) zadeklarowana w programie. Można podawać wartości stałe bezpośrednio w instrukcji *print*. Np.:

```
Print "To jest tekst testowy"
```

Jak widać, wysyłany tekst zawarto w cudzysłowach. Tu mała dygresja - jeśli mamy zadeklarowaną zmienną typu *byte*, a jej wartość wyniesie 255, to polecenie:

```
Print zmienna
```

„wydrukuje” nam 255 (czyli wartość dziesiętną). Czasem potrzebne są dane w formacie szesnastkowym - tu Bascom pomaga nam polece-



Rys. 1. Schemat typowego konwertera z wykorzystaniem MAX232

**Tab. 1. Przykłady wartości błędów przy różnych wartościach prędkości transmisji i oscylatorów kwarcowych**

Częstotliwość kwarcu/prędkość transmisji	1200	2400	4800	9600	19200	57600	115200
1,000000	0,16%	0,16%	0,16%	7,84%	7,84%	7,84%	45,75%
1,843200	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
2,457600	0,00%	0,00%	0,00%	0,00%	0,00%	25,00%	25,00%
3,276000	0,37%	0,37%	1,54%	1,54%	6,23%		
3,686000				4,16%			
4,000000	0,16%	0,16%	0,16%	0,16%	0,16%	7,84%	7,84%
4,096300				2,51%			
4,194000				1,12%			
4,433618				3,00%			
4,433618				3,00%			
4,915200	0,00%	0,00%	0,00%	0,00%	0,00%	6,25%	6,26%
6,144000	x	0,00%	0,00%	0,00%	0,00%	10,00%	10,00%
6,553600	x	0,39%	0,39%	1,56%	1,56%	1,56%	15,63%
7,372800	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
7,680000	0,00%	0,00%	0,00%	0,00%	0,00%	4,00%	4,00%
8,000000	x						
8,860000	x	0,16%	0,16%	0,16%	0,16%	7,84%	7,84%

x - wartość niemożliwa do otrzymania

niem:

```
Print Hex(zmienna)
```

i takie polecenie zamiast 255 „wydrukuje” nam FF. Czasem potrzebujemy także wydrukować znak ASCII znając kod danej liczby np. 66 czyli „b” - do tego celu możemy użyć instrukcji:

```
Print Chr(66)
```

która to instrukcja „wydrukuje” znak b. Polecenie *print* zakończone średnikiem nie wyśle domyślnie znaku końca wiersza i przejścia do następnej linii (odpowiednik ENTER - „CR” - #13“) natomiast bez średnika terminal zawsze przejdzie do następnej linii.

Do odbierania danych przez RS232, Bascom ma dedykowanych kilka instrukcji. Pierwsza z nich to instrukcja *input*. Składnia instrukcji *input* jest następująca:

```
INPUT "tekst", zmienna1, zmienna2...
```

Jako „tekst” podajemy dowolny tekst, który ma się ukazać jako komunikat o oczekiwaniu na podanie danych (może go nie być) np.:

```
Dim A As Byte
```

```
Dim Ciag As String * 36
```

```
INPUT "Podaj A:", a
```

```
'oczekiwać będzie na wpisanie
```

```
'liczby typu byte (0-255)
```

```
INPUT "A teraz wpisz tekst (max. 36
```

```
znaków):", ciag
```

```
'oczekiwać będzie na wpisanie
```

```
'tekstu o długości max. 36 'znaków
```

Oczywiście można stosować kilka zmiennych na raz np.:

```
INPUT "Podaj A oraz tekst:", a
```

```
, ciag
```

```
'oczekiwać będzie na wpisanie
```

```
'liczby typu byte (0-255) oraz
```

```
'na wpisanie tekstu o długości 36
```

```
'znaków
```

Aby zakończyć odbiór danych poprzez polecenie *input* należy na zakończenie zawsze wysłać znak CR (#13 - ENTER) - co w przypadku podawania danych ręcznie np. z klawiatury PC poprzez terminal nie jest z reguły problemem i jest wygodne, natomiast może się zdarzyć że chcemy odbierać dane z jakiegoś podłączonego urządzenia które np. nie wysyła znaków końca wiersza. Tutaj twórca Bascoma przyszedł z pomocą i stworzył wiele instrukcji. Jedną z nich jest polecenie *inputbin*, które odbiera każdy znak niezależnie od tego czy jest to litera czy kod sterujący (np. #13 którego nie interpretuje w żaden sposób, czyli po prostu zapisuje go do zmiennej). Składnia tego polecenia jest następująca:

```
INPUTBIN zmienna1, zmienna2...
```

Polecenie to nie posiada „tekstu zachęty” i oczekuje na tyle znaków ile jest w definicji zmiennych. Na przykład:

```
Dim Ciag As String * 36
```

```
INPUTBIN ciag
```

W tym przypadku instrukcja *inputbin* będzie czekać na odebranie 36 znaków i zakończy odbiór dopiero po odebraniu wszystkich

znaków. Jest to bardzo pomocne, jeśli zawsze odbieramy z urządzenia zewnętrznego dane o takim samym rozmiarze. W przypadku gdy liczba ta jest zmienna, to lepiej użyć innej konstrukcji procedury odbioru. Polecenie *inputbin* może także służyć do odbioru znaków bezpośrednio do tablicy - należy wówczas z góry określić ile bajtów należy odebrać oraz od którego miejsca w tablicy należy zacząć zapisywać dane:

```
Inputbin tablica(3), 6
```

```
'do tablicy trafi 6 bajtów, jako
```

```
'pierwsza zostanie zapisana komórka
```

```
'nr 3
```

Kolejną instrukcją z rodziny *input* godną wspomnienia jest instrukcja *inputhex*. Nie różni się ona składnią od polecenia *input*, różni się natomiast jedną zasadniczą cechą: odbiera znaki w formacie szesnastkowym wprowadzanym jako zwykły tekst, czyli jeśli podamy jako odpowiedź na poniższy przykład:

```
Dim A As Byte
```

```
INPUTHEX "Podaj wartość szesnastkowa", a
```

```
znaki „FU”, to wtedy w zmiennej
```

```
a znajdzie się wartość 255 czyli
```

```
„FU” szesnastkowo. Polecenie to
```

```
umożliwia wprowadzanie znaków
```

```
w formie szesnastkowej. Jedna
```

```
uwaga, każde spośród dotąd wymienionych z rodziny input zatrzymuje
```

```
działanie programu i oczekuje na podanie znaku. Właśnie ta
```

```
cecha powoduje, że polecenia te
```

```
nie nadają się do wykorzystania
```

```
przy odbiorze danych gdzieś np.
```

```
w głównej pętli programu, który
```

```
nie może być wstrzymywany. Natomiast
```

```
znakomicie nadają się do konkretnych zastosowań takich jak
```

```
np.: programowanie pamięci danych
```

```
programu za pomocą dedykowanej aplikacji, odbioru danych
```

```
wywoływanych zewnętrznym impulsem (np. int0 lub int1, czy
```

```
podanym na jakiegokolwiek wejście).
```

```
Poniżej przedstawiono przykład
```

```
odbioru danych przez RS232, które
```

```
są przeznaczone do zapisania z pamięci EEPROM pod przesłany
```

```
wcześniej adres:
```

```
$regfile = "m8def.dat"
```

```
'powiadomienie o wykorzystywanym
```

```
'procesorze atmega8
```

```
$crystal = 8000000
```

```
'częstotliwość oscylatora
```

```
$baud = 9600 'prędkość transmisji
```

```
dim dana as byte
```

```

dim adres as byte

Do
  Input "Adres:", Adres
  'czeka na adres
  INPUTHEX "Dana:", Dana
  'czeka na dana
  Print Adres; " : "; Dana
  'wysła odebrany adres oraz dana
  'do komputera
  Writeeeprom Dana, Adres
  'zapisuje do eeprom otrzymana
  'dana pod otrzymanym 'adresem
Loop
End

```

W przedstawionym przykładzie mikrokontroler czeka na odebranie adresu, pod którym ma być zapisana dana, oraz na zapisywana daną. Odebrana dana następnie zostaje zapisana w pamięci EEPROM pod otrzymanym wcześniej adresem. W przedstawionym przykładzie do odbioru danych została wykorzystana instrukcja *inputhex*, która odbiera dane szesnastkowo - można je od razu zapisać do pamięci EEPROM. W tym przykładzie dla kontroli poprawności odebranych danych wysyłany jest adres oraz odebrana dana poprzez instrukcję *print*. Bascom udostępnia także funkcje umożliwiające odbieranie pojedynczych znaków przez RS232. Do tych funkcji można zaliczyć *waitkey()*, która czeka w na odebranie znaku (wstrzymuje działanie programu do czasu pojawienia się w buforze transmisji szeregowej odebranego znaku) oraz funkcję *inkey()*, która zwraca kod ASCII pierwszego znaku znajdującego się w buforze transmisji szeregowej bez wstrzymywania dzia-

łania programu. Na list. 1 przedstawiono przykład programu ilustrującego różnice w działaniu wymienionych funkcji.

Jeżeli zostanie przez mikrokontroler odebrana „jedynka“ potwierdzona *enterem*, wykonana zostanie funkcja *waitkey()*, która czeka na znak, czyli wstrzymuje działanie programu. Po odebraniu znaku przez funkcję *waitkey()* odebrany znak jest wysyłany z powrotem do PC (w kodzie ASCII). Jeśli zostanie odebrana z portu RS232 wartość 2, to zostanie zaprezentowane działanie funkcji *inkey()*. Jest ona wykonywana aż odebrany zostanie znak o kodzie 27 (ESC). Jeżeli odebrany znak ma kod większy od 0, to do PC zostaje wysłany kod ASCII odebranego znaku oraz sam znak. Tak więc, do PC będą wysyłane odebrane znaki ASCII od 1 do 255. Ponieważ funkcja *inkey()* pobiera znak z bufora sprzętowego układu UART, może ona zwrócić 0, gdy odebrany znakiem będzie znak o kodzie „0“. Powoduje to, że nie nadaje się ona do odbierania danych binarnych, które mogą przecież zawierać bajty zerowe. Można temu zaradzić stosując dodatkowo funkcję *ischarwaiting*, która sprawdzi czy znak rzeczywiście został odebrany. Jeśli funkcja ta zwróci „1“, można odebrać znak przez *inkey()* lub *waitkey()*, który może mieć także wartość 0. Bascom prócz transmisji buforową z wykorzystaniem przerw od wbudowanego UART-u/USART-u w mikrokontrolerze.

Ponieważ transmisja buforowa

używa przerw od UART-u/USART-u, tak więc nie jest możliwe jej wykorzystanie przy programowej realizacji interfejsu RS232 tylko sprzętowej. Transmisja buforowa z wykorzystaniem przerw ma wiele zalet. Odbieranie oraz nadawanie danych może przebiegać w obsłudze przerwania, czyli w tle programu głównego. Może być wykonywany program główny, którego kolejne instrukcje nie muszą czekać aż dany ciąg znaków zostanie odebrany lub wysłany. Dzięki buforowi odbieranie danych czy nadawanie może być przeprowadzone w programie głównym, kiedy będzie do tego sposobność. Jeżeli nie zostanie przekroczona zadana pojemność bufora żaden odebrany czy nadany ciąg znaków nie zostanie zgubiony, co miało by miejsce przy wykonywaniu jakiś długo czasowych instrukcji w programie głównym. W przypadku transmisji buforowej odbierane czy nadawane znaki są ładowane i brane z bufora w przerwaniu. Czyli niezależnie od działania programu głównego. Po prostu transmisji buforowej można używać, gdy potrzebne będzie wysyłanie oraz odbieranie znaków w tle działania programu głównego (która wykorzystuje przerwanie od UART-u/USART-u). Będzie używana zwłaszcza wtedy, gdy zadania wykonywane w programie głównym będą czasochłonne a nie będzie można przeoczyć żadnych odebranych przez RS232 znaków. Do konfiguracji sprzętowego układu UART/USART by używał bufora wejściowego służy polecenie konfiguracyjne *config serialin*. Gdy użyte zostanie to polecenie, automatycznie zostaną stworzone trzy zmienne, których znaczenie jest następujące:

- *\_RS\_HEAD\_PTR0* bajt - wskaźnik, określający gdzie znajduje się pierwszy jeszcze nie odebrany bajt,
- *\_RS\_TAIL\_PTR0* bajt - wskaźnik określający gdzie znajduje się ostatni jeszcze nie odebrany bajt,
- *\_RS232INBUF0* tablica bajtów pełniąca rolę wejściowego bufora kołowego.

Przy transmisji buforowej (odbiorczej) po każdym odebrany znaku wywoływana jest procedura

List. 1. Przykład programu pokazującego działanie funkcji *waitkey()* oraz *inkey()*

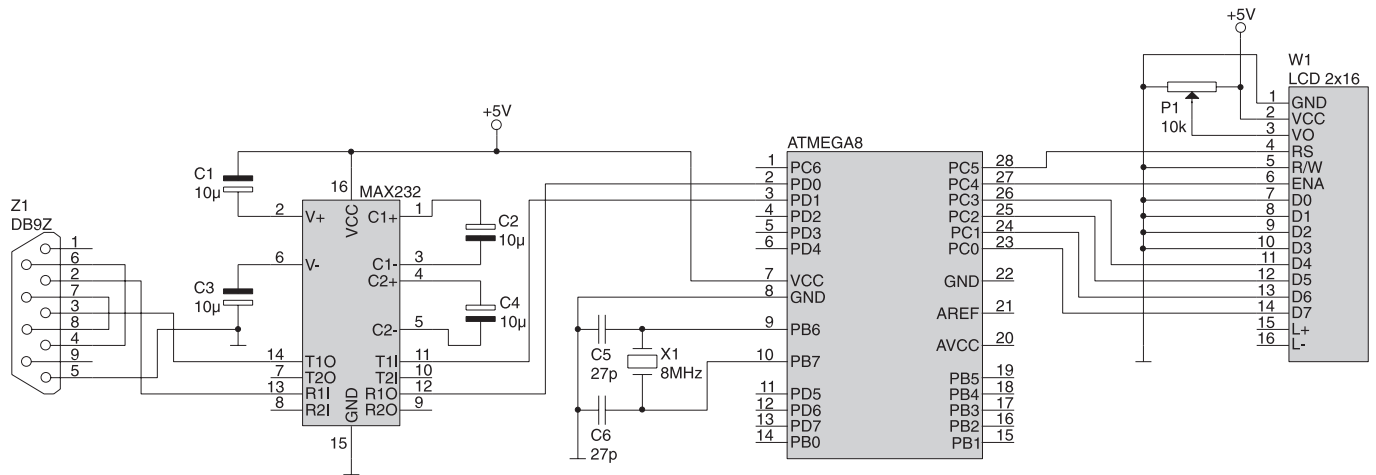
```

$regfile = „m8def.dat“
$crystal = 8000000
$baud = 9600 'predkosc transmisji

Dim A As Byte 'definicje zmiennych

Do 'petla glowna programu
  Print „Program testowy“
  Print „1. Instrukcja waitkey()“
  Print „2. Instrukcja inkey()“
  Input „Wybierz opcje i naciśnij enter:“, A
  If A = 1 Then 'jesli odebrano 1 to
    A = Waitkey() 'czekaj na znak
    Print A; „ - „; Chr(a) 'wyslij do terminala odebrany znak oraz jego kod ascii
  End If
  If A = 2 Then 'jesli odebrano 2 to
    Do 'początek warunkowej petli do-loop
      A = Inkey() 'odebranie znaku
      If A > 0 Then 'jesli odebrany znak wiekszy od 0 (kod ascii) to
        Print „Nacisniete (kod ASCII):“; A; „ „ czyli znak „; Chr(a)
        'wysli do terminala kod ascii odebranego znaku oraz znak
      End If
    Loop Until A = 27 'jesli odebrany znak ma kod 27 (esc) to zakoncz petle do-loop
  End If
Loop
End

```



Rys. 2. Schemat przykładowego układu testowego

przerwania, w której ładowany jest odebrany znak do bufora. Gdy nie ma miejsca w buforze, odebrany znak nie jest umieszczony w buforze. Dlatego bufor musi być opróżniany cyklicznie, podczas odczytywania instrukcjami *inkey()* lub *input*. Ponieważ transmisja odbiorcza - buforowa wykorzystuje przerwanie URXC, nie jest możliwe używanie tego źródła przerwania w programie użytkownika. Aby pokazać działanie transmisji buforowej przedstawiony zostanie przykład programu, który odbiera znaki z portu RS232 oraz wyświetla je na LCD. Jeżeli przekroczona zostanie pojemność pierwszej linii wyświetlacza LCD wyświetlacz jest czyszczony i kolejne odebrane znaki są wyświetlane od początku linii. Na rys. 2 przedstawiono schemat ideowy systemu, w którym zastosowano wyświetlacz LCD oraz konwerter poziomów napięć TTL-<->RS232. Wykorzystany został mikrokontroler ATmega8.

Przykładowy program (list. 2) odbiera znaki przez RS232 oraz wyświetla je na wyświetlaczu z opóźnieniem 1 sekundy. Wprowadzone opóźnienie 1 sekundy ma na celu pokazanie poprawności działania transmisji buforowej.

Ja widać na list. 2, sprzętowy UART/USART jest skonfigurowany w taki sposób, aby używał bufora wejściowego (odbiorczego). Wykorzystano do tego celu instrukcja *config serialin* ustalająca rozmiar bufora odbiorczego o wielkości 50 bajtów. Aby było możliwe poprawne działanie transmisji buforowej konieczne należy odblokować globalny system przerwań, co w przykładowym programie zosta-

ło uczynione. Należy o tym pamiętać, bo w przeciwnym wypadku transmisja buforowa nie będzie działać. W nieskończonej pętli *do-loop* odbierany jest znak z bufora transmisji funkcją *inkey()* i jeżeli kod ASCII odebranego znaku jest większy od 31 to odebrany znak zostaje wyświetlony na LCD oraz zwiększona zostanie o jeden wartość zmiennej *poz*. Jeżeli zmienna *poz* osiągnie wartość 17 czyli wypełniony zostanie znakami cały wiersz wyświetlacza 2\*16 znaków to następuje czyszczenie LCD oraz wpisanie do zmiennej *poz* wartości początkowej 1. Opóźnienie 1 sekundy zostało wprowadzone w celu spowolnienia działania pro-

gramu głównego, co ma na celu pokazanie zalet transmisji buforowej - odbiorczej. Po uruchomieniu tego programu wysyłane poprzez terminal znaki np. przez ten w jaki został wyposażony Bascom zostaną wszystkie odebrane (nie zostanie żaden zgubiony) i wyświetlone na LCD odebrane znaki oczywiście jeśli nie zostanie przekroczona pojemność zadeklarowanego bufora. Aby zobaczyć działanie tego programu bez użycia transmisji buforowej należy wykasować z programu instrukcję konfiguracyjną *config serialin*. Po tym zabiegu i po załadowaniu programu wysyłane znaki przez terminal częściej niż co sekundę będą po

```
List. 2. Realizacja odbierania znaków przez RS232 z wykorzystaniem
transmisji buforowej
'Przykład buforowego odbioru danych przez RS232 (w przerwaniu)
'predkosc transmisji 9600 bodow

$regfile = „m8def.dat”           'informuje kompilator o pliku dyrektyw
                                'wykorzystywanego mikrokontrolera
$crystal = 8000000               'informuje kompilator o czestotliwosci rezonatora
                                'kwarcowego
$baud = 9600                     'informuje kompilator o predkosci transmisji

Config Lcd = 16 * 2              'konfiguracja organizacji znakow wyswietlacza LCD
Config Lcdpin = Pin, Db4 = Portc.3, Db5 = Portc.2, Db6 = Portc.1, Db7 = Portc.0, E
= Portc.4, Rs = Portc.5
'konfiguracja pinow mikrokontrolera do ktorych dolaczone zostaly linie wyswietlacza
Config Serialin = Buffered, Size = 50 'konfiguracja bufora wejsciowego

Dim Znak As String * 1          'definicja zmiennej znak typu string
Dim Poz As Byte                 'definicja zmiennej poz typu byte

Enable Interrupts               'odblokowanie przerwan globalnych

Cls                              'czyszc lcd
Poz = 1                          'wartosc poczatkowa zmiennej poz

Do                               'nieskonczona petla do-loop
  Znak = Inkey()                 'odbierz znak z bufora wejsciowego
  If Asc(Znak) > 31 Then         'jezeli wartosc ascii odebranego znaku wieksza niz 0 to
    If Poz = 17 Then             'jesli zmienna poz=17 to
      Cls                         'czyszc lcd
      Poz = 1                     'zapisz do poz wartosc 1
    End If
    Lcd Znak                       'wyswietl na lcd odczytany z bufora znak
    Incr Poz                       'zwiększ o jeden wartosc zmiennej poz
  End If
  Wait 1                           'czekaj 1 sekunde
Loop                               'koniec nieskonczonej petli
End                               'koniec programu
```

prostu tracone i nie do odzyskania. Aby móc np. odbierać znaki ASCII od 0 do 255 (oczywiście w przypadku buforowego odbierania znaków) można posłużyć się dodatkowymi zmiennymi `_RS_HEAD_PTR0` oraz `_RS_TAIL_PTR0`, które są tworzone automatycznie podczas używania odbiorczej transmisji buforowej. Jeżeli wartości tych zmiennych są różne to wiadomo, że w buforze odbiorczym znajduje się nieodebrany znak/i. Natomiast, gdy są równe znaczy że w buforze odbiorczym nie ma żadnych nieodebranych znaków. Realizacja nadawania przez UART/USART z wykorzystaniem bufora przebiega podobnie jak odbiór buforowy. Tylko zamiast instrukcji `config serialin` należy wykorzystać instrukcję `config serialout`. W przypadku bufora nadawczego także należy obowiązkowo odblokować globalny system przerwań. Oczywiście jest możliwe jednoczesne używanie bufora wejściowego jak i wyjściowego. Bascom prócz obsługi sprzętowej RS232 umożliwia także realizację programową interfejsu RS232. Programowy UART będzie wykorzystywany zwłaszcza, gdy wykorzystywany mikrokontroler nie będzie zawierał sprzętowego UART-u/USART-u a będzie potrzebny lub gdy będzie potrzebny drugi port RS232. Bascom posiada także instrukcje realizujące dynamiczny programowy UART. Dzięki czemu możliwe jest dowolne wykorzystanie w każdej chwili końcówek mikrokontrolera do nadawania lub odbioru danych przez RS232. Dynamicznie można zmieniać używane do transmisji końcówki mikrokontrolera oraz parametry transmisji. Jeśli będzie zainteresowanie zostanie poświęcona część miejsca w tej rubryce na zagadnienia związane z programowym UART-em. Oczywiście RS232 można nie tylko wykorzystywać do komunikacji z komputerem ale także z innymi mikrokontrolerami czy urządzeniami wyposażonymi w taki port.

**Marcin Wiązania**

**marcin.wiazania@ep.com.pl**

**Artur Starz**

**artur.starz@ep.com.pl**