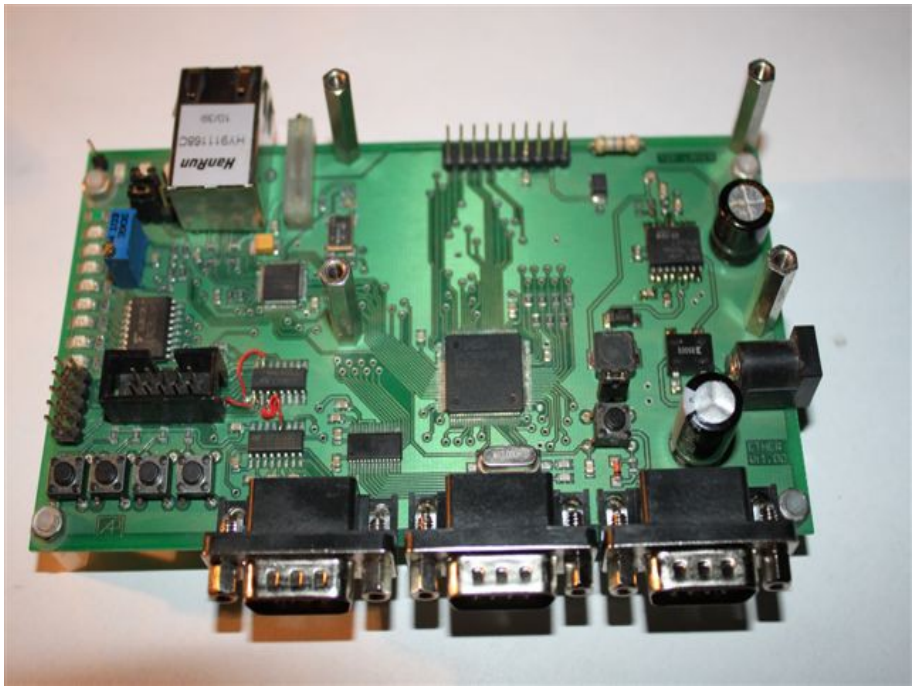
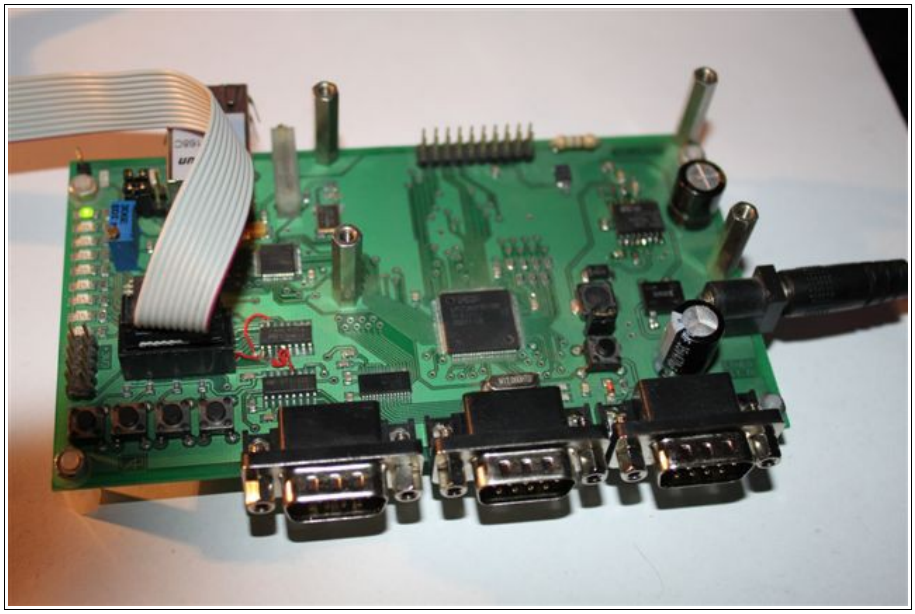
Andrzej Pawluczuk



Filozofia sieci:
Obsługa transmisji szeregowej via UART w przerwaniach

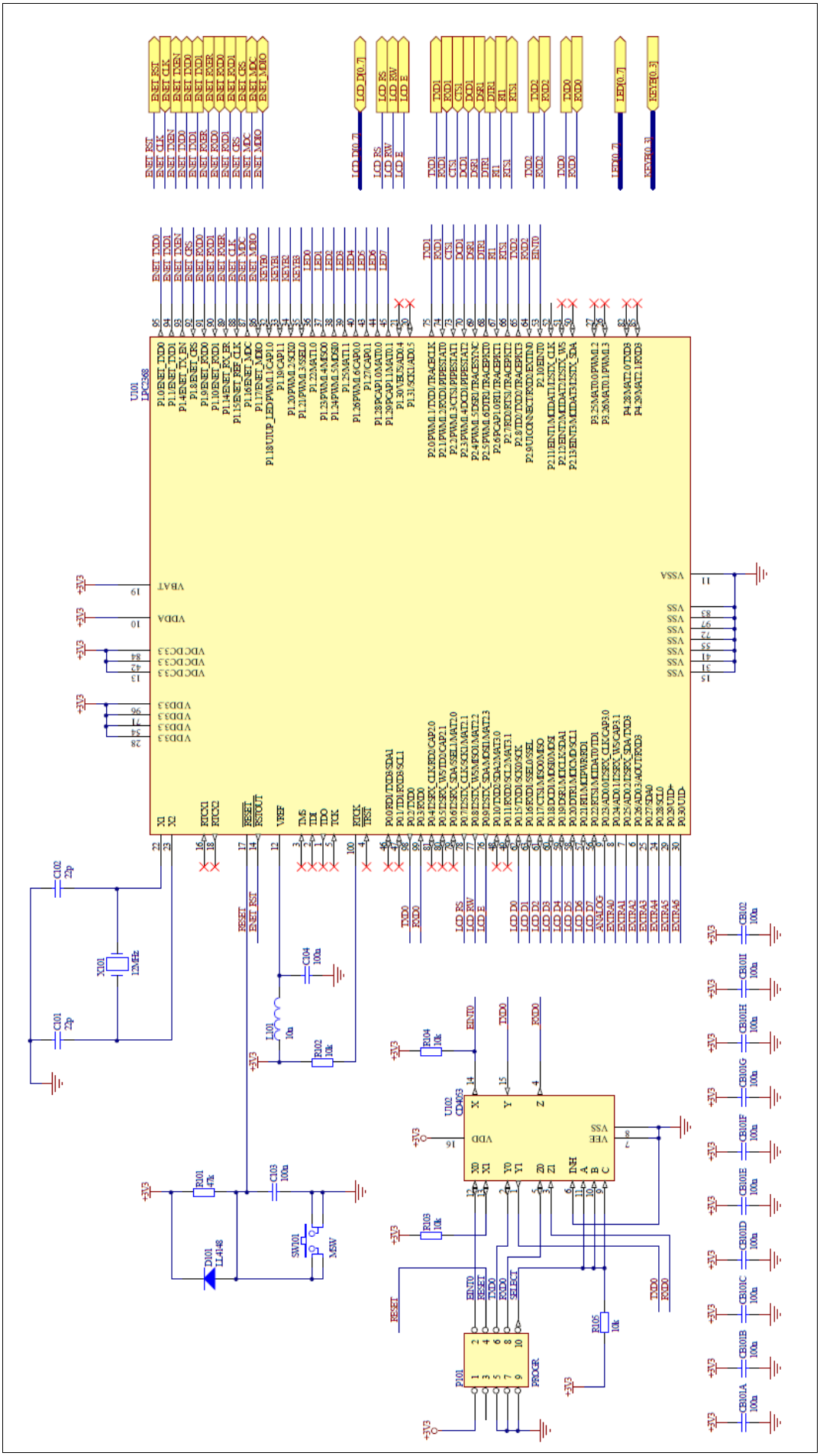
dla Elportal
Białystok, styczeń 2019

Zagadnienie obsługi sieci generuje potrzebę „opanowania” również innych peryferali, które towarzyszą programom ilustrującym problematykę sieciową. Takim wręcz elementarnym podzespołem jest obsługa asynchronicznej transmisji szeregowej via UART (mikrokontroler ma ich 4, dokument opisuje posługiwanie się wszystkimi). Przykładowe programy demonstracyjne mogą być uruchamiane (bez jakichkolwiek zmian) w mikrokontrolerach LPC236x, LPC237x. Do eksperymentów posłuży nam środowisko sprzętowe pokazane na ilustracji 1. Jest to płytka badawczo-rozwojowa zawierająca mikrokontroler LPC2366. Zaprezentowane programy nie wyczerpują wszystkich możliwości oferujących przez zespoły UART zintegrowane w mikrokontrolerze, jedynie są sugestią i wskazówką do indywidualnych eksperymentów i prób. Tylko własne doświadczenie daję prawdziwą wiedzę.

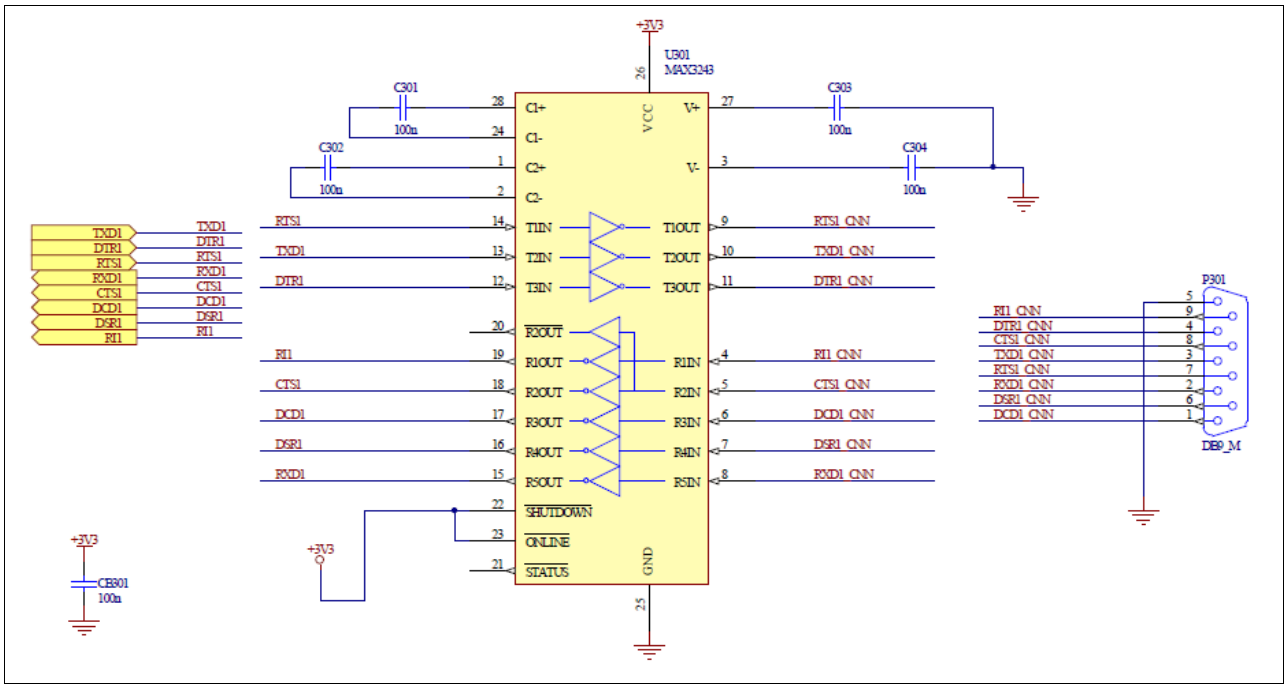


Ilustracja 1: Królik doświadczalny

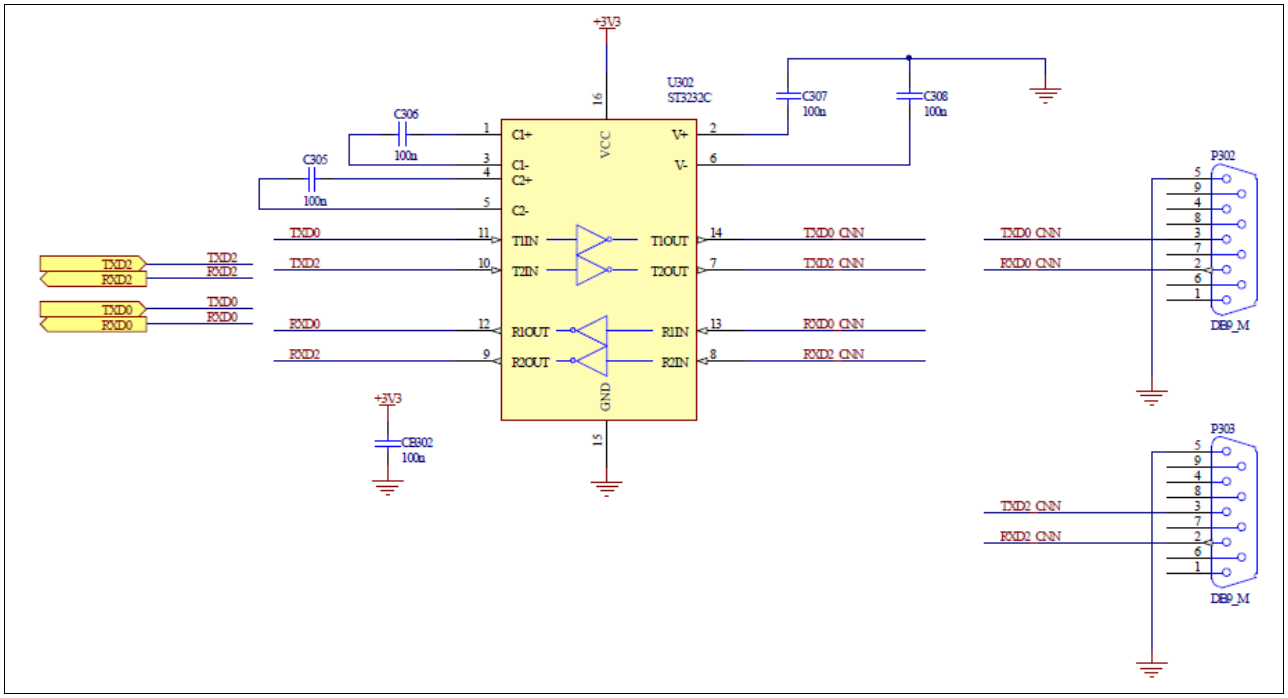
Jej schemat, w zakresie istotnych w tym dokumencie elementów pokazują ilustracje 2 (Jednostka centralna), 3 (Interfejs UART1), 4 (Interfejs UART0 i UART2) i 5 (Stabilizator napięcia).



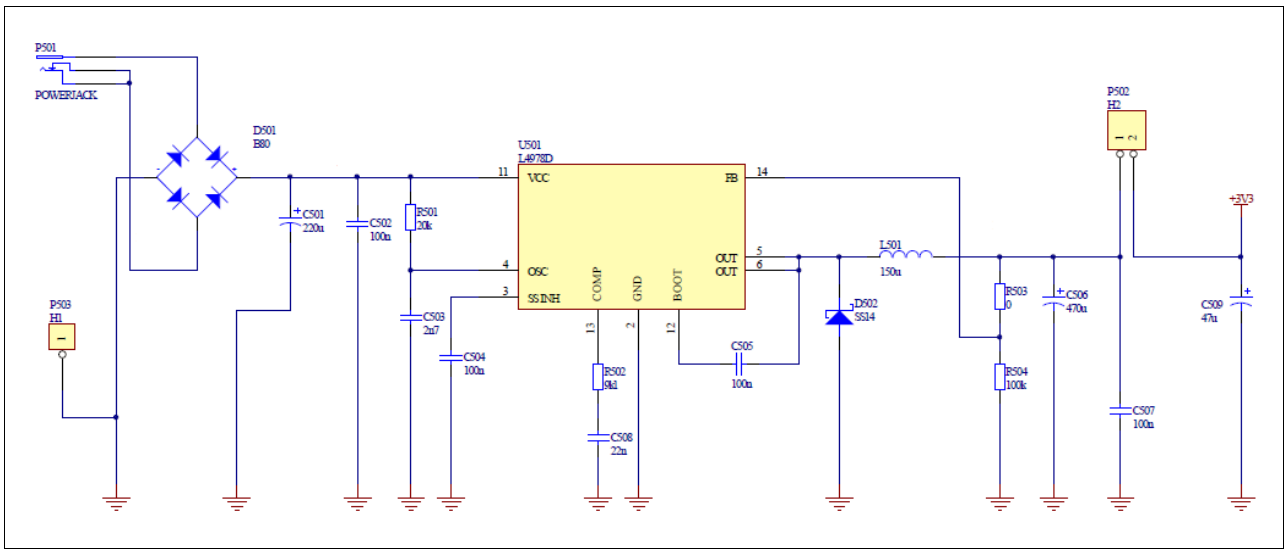
Ilustracja 2: Jednostka centralna



Ilustracja 3: Interfejs UART1

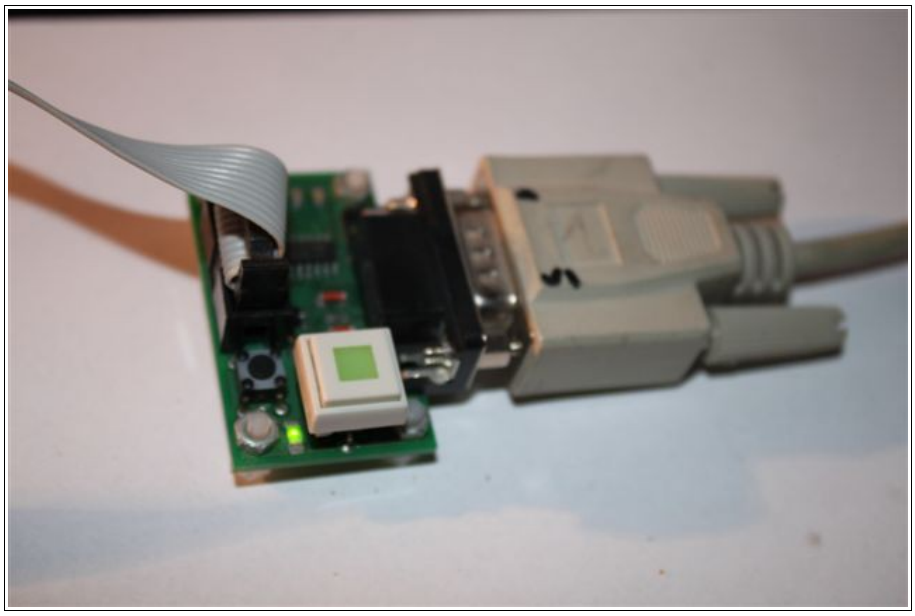


Ilustracja 4: Interfejs UART0 i UART2



Ilustracja 5: Stabilizator napięcia

Całość jest programowana (jako operacja załadowania kodu programu do pamięci FLASH) w trybie IAP poprzez interfejs (ilustracja 6), z przyłączonym interfejsem pokazuje ilustracja 7.



Ilustracja 6: Interfejs do programowania w trybie IAP

Do operacji programowania zawartości pamięci FLASH mikrokontrolera używany jest program FLASHMAGIC.

Ilustracja 7: Przyłączony interfejs programujący

Zespoły UART (wszystkie) są wersją kontrolera stosowanego w komputerach PC. Standard ten, historycznie wywodzi się od układu Intela 8250. Usunięcie z tego układu pewnych problemów związanych z generowaniem przerwań stworzyło układ 16450, który z kolei wyewuluował (po dodaniu obsługi kolejek FIFO) do standardu 16550. Ten ostatni jest powszechnie używany w komputerach PC (no może nie we wszystkich, gdyż klasyczne porty szeregowe wyginęły w komputerach przenośnych, ale w komputerach stosowanych do automatyki przemysłowej nadal są wszechobecne). Pełny standard 16550, oprócz linii do transmisji danych (nadawania i odbioru) zawiera sterowanie liniami modemowymi. Są przewidziane dwie linie wyjściowe (DTR oraz RTS) oraz cztery linie wejściowe (CTS, DCD, DSR i RI). Sam standard 16550 pozwala praktycznie zrealizować każdą potrzebę komunikacyjną. Kontroler ten umożliwia transmisję znaków 5-bitowych, 6-bitowych, 7-bitowych oraz 8-bitowych. Można mniemać, że to trochę dziwne, w dobie powszechnie używanych 8-bitowych transmisji, istnieją transmisje o krótszych znakach, jednak są takie zastosowania. Jako przykład może służyć myszka komputerowa. Obecnie jest ona obsługiwana poprzez złącze USB, ale w minionych czasach, tych trochę wcześniejszych (jeszcze przez standardem PS2), myszki były przyłączane do portu COM komputera, a komunikacja do nich była realizowana jako 7-bitowa. Również standard 16550 umożliwia przesyłanie znaków bez bitu parzystości, z bitem parzystości lub bitem nieparzystości. Dodatkowo znaki mogą być zakończone jednym bitem stopu, dwoma bitami stopu i 1,5 bitu stopu. Można zastanawiać się po co tyle tego, skoro w większości przypadków wbudowane w różnorodne mikrokontrolery układy transmisji szeregowej przeważnie używają jednego bitu stopu. Po prostu uniwersum serwuje wiele możliwości a każdy użytkownik wybiera własny wariant.

Podobnie, jak w innych mikrokontrolerach, standard 16550 umożliwia transmisję z różną prędkością nadawania i odbierania znaków. Oznacza to, że rozwiązanie to zawiera rejestry, do których można wpisać wartości podzielników przewidzianych do

generowania rzeczywistego sygnału taktującego transmisję szeregową.

W operacjach dostępu do rejestrów kontrolera UART (zapisu jak i odczytu) występuje tryb określany jako DLAB (ang. **D**ivisor **L**atch **A**ccess **B**it) – takie rozszerzenie adresowe rejestrów. Obecnie, w mikrokontrolerach z rodziny LPC2000, występowanie takiej „pokraki" może wydawać się zastanawiające, ale jeżeli na to spojrzeć z innej perspektywy, to uzasadnienie tego może okazać się, wbrew pozorom, całkiem sensowne (nie wszystko jest takie na jakie wygląda). Standard 16550 ma do dyspozycji 8 rejestrów. Z biegiem czasu ta przestrzeń adresowa stawała się za mała, więc wymyślono jakąś koncepcję rozszerzenia. Czy ona jest dobra, czy zła, nie jest to istotne, po prostu jest jakaś. Ponieważ ludzkość stworzyła ogrom oprogramowania na PC-ty, gdzie kanały szeregowe obsługuje się w jakiś sposób, to adaptacja ich do nowego środowiska (przykładowo LPC2000), w sytuacji gdy obsługa sprzętowa jest identyczna, zawsze upraszcza modyfikację „starego" oprogramowania. Stan DLAB uzyskuje się w wyniku ustawienia najstarszego bitu (o numerze 7, gdyż wszystkie rejestry kontrolera URTx są ośmiobitowe), w rejestrze UxLCR.

UART0

W interfejsie UART0 nie ma linii modemowych (ten kanał transmisyjny jest okrojony o linie modemowe). Z tego powodu nie zawiera on również w przestrzeni adresowej rejestrów dedykowanych do obsługi linii modemowych (w standardzie 16550, jak i w poprzednich, znajdują się rejestry do sterowania wyjściowymi liniami modemowymi, jak i rejestry pozwalające wczytać wejściowe linie modemowe).

Do obsługi kontrolera UART0 przewidziane są następujące rejestry:

Symbol rejestru	Adres	Dostęp	Znaczenie
U0RBR	0xE000 C000 (DLAB=0)	odczyt	rejestr, z którego odczytywany jest odebrany kod znaku
U0THR	0xE000 C000 (DLAB=0)	zapis	rejestr, do którego wpisywany jest kod znaku do nadania
U0DLL	0xE000 C000 (DLAB=1)	zapis odczyt	rejestr, do którego wpisywana jest młodsza część podzielnika częstotliwości
U0DLM	0xE000 C004 (DLAB=1)	zapis odczyt	rejestr, do którego wpisywany jest starsza część podzielnika częstotliwości
U0IER	0xE000 C004 (DLAB=0)	zapis odczyt	rejestr określający jakie zdarzenia generują przerwanie
U0IIR	0xE000 C008	odczyt	rejestr pozwalający zidentyfikować przyczynę przerwania
U0FCR	0xE000 C008	zapis	rejestr sterujący kolejką FIFO

Symbol rejestru	Adres	Dostęp	Znaczenie
U0LCR	0xE000 C00C	zapis odczyt	rejestr sterujący trybem transmisji
U0LSR	0xE000 C014	odczyt	rejestr statusu odbiornika
U0SCR	0xE000 C01C	zapis odczyt	rejestr bez przydzielonej funkcji (w układach 16C550 był dopełnieniem do ośmiu komórek w przestrzeni adresowej)
U0ACR	0xE000 C020	zapis odczyt	
U0FDR	0xE000 C028	zapis odczyt	rejestr zawierający dodatkowe współczynniki pozwalające doprecyzować prędkość transmisji
U0TER	0xE000 C030	zapis odczyt	rejestr, poprzez który można włączać/wyłączać UART0

Wybrane rejestry (zaznaczone kolorem) mają tą samą alokację w przestrzeni adresowej. Dostęp do poszczególnych rejestrów jest rozróżniany na podstawie typu operacji (zapis/odczyt) oraz wskaźnika DLAB. Przejście do tego trybu następuje po ustawieniu bitu b7 w rejestrze U0LCR (zapisaniu do wymienionego rejestru takiej sekwencji bitowej, gdzie ustawiony jest odpowiedni bit).

Do obsługi zespołu UART0 utworzony jest moduł UART0.C (oraz UART0.H). To oprogramowanie udostępnia następujące funkcje:

- **void UART0InitEnvir (void)** – bezparametrowa funkcja, której zadaniem jest skonfigurowanie pinów portu do potrzeb transmisji szeregowej (wyprowadzenia P0[2] i P0[3] stają się wyprowadzeniami zespołu UART0),
- **void UART0HardwInit (ULONG PeriphClk , UCHAR SerialMode , ULONG SerialSpeed)** – funkcja do zainicjowania trybu i prędkości pracy zespołu UART0, parametr *PeriphClk* jest częstotliwością sygnału zegarowego taktującego mikrokontroler wynikającą sumarycznie z częstotliwości rezonatora kwarcowego i ewentualnego układu PLL zwielokrotniającego tą częstotliwość, parametr *SerialMode* określa tryb pracy kontrolera transmisji szeregowej (jest wartością wprost zapisywaną do rejestru trybu), oraz *SerialSpeed* określa prędkość transmisji szeregowej wyrażoną w bajtach na sekundę (bps),
- **void UART0InputReset (void)** – funkcja zerująca programową kolejkę FIFO związaną z odebranymi w przerwaniach znakami,
- **void UART0Send (UCHAR Data)** – funkcja wysyłająca w kanał szeregowy podany w parametrach znak,
- **void UART0SendString (UCHAR * String)** – funkcja wysyłająca w kanał szeregowy ciąg znaków o dowolnej długości (ciąg musi być zakończony znakiem o kodzie 0),
- **ULONG UART0DataPresent (void)** – funkcja zwracająca informację, czy w kolejce FIFO odbiornika są dane do odczytania,

- **UCHAR UART0GetData (void)** – funkcja odczytująca z programowej kolejki FIFO kolejny znak,
- **UCHAR UART0Working (void)** – funkcja zwracająca status czy UART0 jest w danej chwili zajęty (zajmuje się wysyłaniem danych).

Odbierane przez UART0 dane są gromadzone w programowej kolejce FIFO związanej z odbiornikiem. Każde przerwanie wygenerowane przez odbiornik sprowadza się do odczytania z rejestrów UART0 kodu odebranego znaku oraz dodania jego do wspomnianej kolejki FIFO. Nadawanie znaków prowadzi do realizacji jednego z dwóch wariantów: jeżeli układ UART0 nie jest zajęty, to nadawany znak trafia bezpośrednio w rejestry nadajnika UART0. W przeciwnym wypadku (drugi wariant), kod znaku dodany jest do programowej kolejki FIFO związanej z nadawaniem znaków. Przerwanie wygenerowane przez nadajnik UART0 ewentualnie pobiera z tej kolejki kolejny znak do nadania aż do wyczerpania „zapasu" znaków zgromadzonych w kolejce FIFO. Wielkości buforów cyklicznych (obsługujących określone kolejki FIFO) są następujące:

```
#define InpSerialBuffSize      64
#define OutSerialBuffSize      128
```

Obsługa przerwania (zarówno generowanego przez nadajnik jak i odbiornik) zespołu UART0 trafia do jednej funkcji (obsługi przerwania wygenerowanego przez układ UART0 z jakiegokolwiek powodu). Rozpoznanie przyczyny przerwania polega na odczytaniu odpowiedniego rejestru identyfikującego (U0IIR). Przyczyna przerwania jest określona na odpowiednich bitach. Przykładowa realizacja obsługi przerwania przedstawia się następująco:

```
__irq void UART0IrqHandler ( void )
```

Funkcja obsługi przerwania układu UART0. Kontroler ten generuje jeden sygnał przerwania. Z tego względu w implementacji obsługi przerwania należy rozpoznać przyczynę, w wyniku której został wygenerowany sygnał przerwania.

```
{
  UCHAR Data ;
  UCHAR U0IrqStatus ;
  /*-------------------------------------------------------------------*/
  for ( ; ; )
  {
    U0IrqStatus = ( UCHAR ) U0IIR ;
```

Przyczynę przerwania należy rozpoznać na podstawie informacji odczytanej z rejestru *U0IIR*.

```
    if ( U0IrqStatus & 0x01 )
    {
```

Jeżeli w odczytanych danych ustawiony jest najmłodszy bit, to oznacza, że układ UART nie zgłasza przerwania.

```
      VICVectAddr = 0 ;
      return ;
    } /* if */ ;
    switch ( ( U0IrqStatus >> 1 ) & 0x03 )
    {
```

Jeżeli najmłodszy bit ma wartość '0', to na kolejnych 3 bitach określona jest przyczyna wygenerowania sygnału przerwania. Po przesunięciu zawartości odczytanego statusu o jeden bit w prawo i wyselekcjonowaniu 3 najmłodszych bitów, należy przejść do właściwego wariantu pozwalającego na właściwą obsługę przerwania. Nas interesują dwie możliwości: przerwanie od nadajnika oraz przerwanie od odbiornika.

```
      case 1              :   /* 0 1 - transmitter */
```

Jeżeli jest to przerwanie od nadajnika (zostały zakończone operacje związane z nadaniem ostatniego znaku i UART gotowy jest do nadania kolejnego), ewentualny kolejny znak do nadania jest pobrany z programowej kolejki FIFO nadajnika.

```
        if (UART0OutBuffer.RdDataPointer==UART0OutBuffer.WrDataPointer)
        {
          UART0OutBuffer . BusyStatus = 0 ;
          break ;
        } /* if */ ;
        Data = UART0OutBuffer.Buffer [ UART0OutBuffer.RdDataPointer ] ;
        UART0OutBuffer . RdDataPointer ++ ;
        if ( UART0OutBuffer . RdDataPointer >= OutSerialBuffSize )
          UART0OutBuffer . RdDataPointer = 0 ;
        U0THR = Data ;
        break ;
      case 2              :   /* 1 0 - reciever data */
```

Jeżeli jest to przerwanie od odbiornika (odebrano poprawny znak), należy odczytać znak i umieścić go w buforze cyklicznym programowej kolejki FIFO odbiornika do późniejszej „konsumpcji”.

```
        Data = U0RBR ;
        UART0InpBuffer.Buffer [ UART0InpBuffer.WrDataPointer ]=Data ;
        UART0InpBuffer . WrDataPointer ++ ;
        if ( UART0InpBuffer . WrDataPointer >= InpSerialBuffSize )
          UART0InpBuffer . WrDataPointer = 0 ;
        break ;
      default             :
        VICVectAddr = 0 ;
        return ;
    } /* switch */ ;
  } /* for */ ;
} /* UART0IrqHandler */
```

Zaprogramowanie kontrolera UART0 do wymaganego trybu pracy i wymaganej prędkości komunikacyjnej sprowadza się do wywołania odpowiedniej funkcji. W jej parametrach wniesiona jest częstotliwość taktująca mikrokontroler, tryb transmisji (liczba bitów znaku, liczba bitów stopu, rodzaj parzystości) oraz prędkość transmisji (podana w bps – bajtach na sekundę). Realizacja funkcji może być następująca:

```
void UART0HardwInit( ULONG PeriphClk ,
                     UCHAR SerialMode ,
                     ULONG SerialSpeed )
{
  ULONG Fdiv ;
  ULONG Fmod ;
  ULONG LDivider ;
  ULONG HDivider ;
```

```
/*-------------------------------------------------------------------*/
UART0DisableInterrupt ( ) ;
```

Konfigurowanie układu UART może w trakcie wygenerować przerwanie, więc wszystkie czynności są robione przy zablokowanych przerwaniach od kontrolera UART0.

```
UART0InputReset ( ) ;
UART0OutputReset ( ) ;
```

Wyzerowanie wszystkich programowych kolejek FIFO (nadajnika i odbiornika).

```
Fdiv = ( PeriphClk / 16 ) / SerialSpeed ;
Fmod = ( PeriphClk / 16 ) % SerialSpeed ;
if ( ( Fmod << 1 ) >= SerialSpeed )
  Fdiv ++ ;
LDivider = Fdiv % 256 ;
HDivider = Fdiv / 256 ;
```

Obliczenie podzielnika do uzyskania oczekiwanej prędkości transmisji. Podzielnik jest rozłożony na dwie części 8-bitowe części. Należy pamiętać, że sam UART0 dzieli częstotliwość przez 16.

```
U0FDR = 0x10 ;
```

Rejestr korekcji podzielnika częstotliwości: zawartość rejestru jest podzielona na na dwie części: ***DivAddVal*** jako bity 0..3 oraz ***MulVal*** jako bity 4..7. Wyrażenie na korekcję jest następujące: $(1+\frac{DivAddVal}{MulVal})$, toteż wpis wartości 0x10 nie wnosi żadnej korekcji, gdyż ***DivAddVal***=*0* oraz ***MulVal***=*1* → finalnie wyrażenie ma wartość 1 – nic nie wnosi.

```
U0LCR = 0x80 ;
```

Przejście do trybu ***DLAB***, jako ustawienie właściwego bitu w rejestrze ***U0LCR***. Od tej chwili dwa kolejne zapisy do rejestru ***U0DLL*** i ***U0DLM*** dotyczą wpisów podzielnika do prędkości transmisji. Bez ustawienia bitu **DLAB** powyższe zapisy dotyczyłyby rejestru danych nadajnika oraz rejestru określającego dopuszczalne przyczyny przerwania. Po dwóch wpisach tryb DLAB „kasuje się”.

```
U0DLL = LDivider ;
U0DLM = HDivider ;
```

Wpis młodszej i starszej części podzielnika częstotliwości do uzyskania
oczekiwanej prędkości transmisji.

```
U0LCR = SerialMode ;
```

Wpis do rejestru determinującego tryb transmisji.

```
U0FCR = 0x07 ;
```

Wpis kasujący ewentualną sprzętową kolejkę FIFO zawartą w samym zespole UART oraz determinujący pracę bez kolejki (w zespole UART – nie mylić z programową kolejką FIFO znaków oczekujących na nadanie).

```
VICVectAddr6 = ( unsigned long ) UART0IrqHandler ;
VICVectCntl6 = 15 ;
```

„Podłączenie” funkcji ***UART0IrqHandler*** do systemu przerwań. Polega na wpisie do rejestru kontrolera przerwań określanego jako

VICVectAddr6 adresu funkcji obsługi przerwania (końcówka nazwy rejestru 6 – oznacza numer przerwania generowanego przez UART).

```
  UART0EnableInterrupt ( ) ;
```

Określenie priorytetu dla przerwania od UART0 – wpis wartości 15 oznacza najniższy priorytet (0 – oznacza priorytet najwyższy).

```
  U0IER = RecvDataInterruptEnable | TransmDataInterruptEnable ;
```

Wpis do rejestru zezwolenia na przerwania. Program dopuszcza (spodziewa się) przerwań od odbiornika oraz nadajnika. Powyższe stałe odpowiadają odpowiednim pozycjom bitowym w rejestrze.

```
} /* UART0HardwInit */
```

Przykładowy program demonstracyjny jest następujący (program odbiera znaki z kanału szeregowego i gromadzi je w buforze aż do napotkania znaku *Enter* [właściwie dowolnego znaku kontrolnego], po czym cały odebrany tekst wysyła zwrotnie w kanał szeregowy):

```
#include <LPC23xx.H>
#include "system.h"
#include "types.h"
#include "irqvectors.h"
#include "serialconst.h"
#include "uart0.h"
#include "strings.h"

#define IRQCallCntLimit                 500
#define BufferSize                      64

static ULONG IRQCallCounter ;
static USHORT Index ;
static UCHAR Buffer [ BufferSize ] ;

static void SoftwareInit ( void )
{
  /*-------------------------------------------------------------------------*/
  IRQCallCounter = 0 ;
  Index = 0 ;
} /* SoftwareInit */


static __irq void T1_IRQHandler ( void )
{
  /*-------------------------------------------------------------------------*/
  IRQCallCounter ++ ;
  if ( IRQCallCounter >= IRQCallCntLimit )
  {
    IRQCallCounter = 0 ;
  } /* if */ ;
  T1IR = 1 ;
  VICVectAddr = 0 ;
} /* T1_IRQHandler */


static void HardwareInit ( void )
{
  /*-------------------------------------------------------------------------*/
  PINSEL10 = 0 ;
  T1MR0 = ( PCLK / 1000 ) - 1 ;
  T1MCR = 3 ;
```

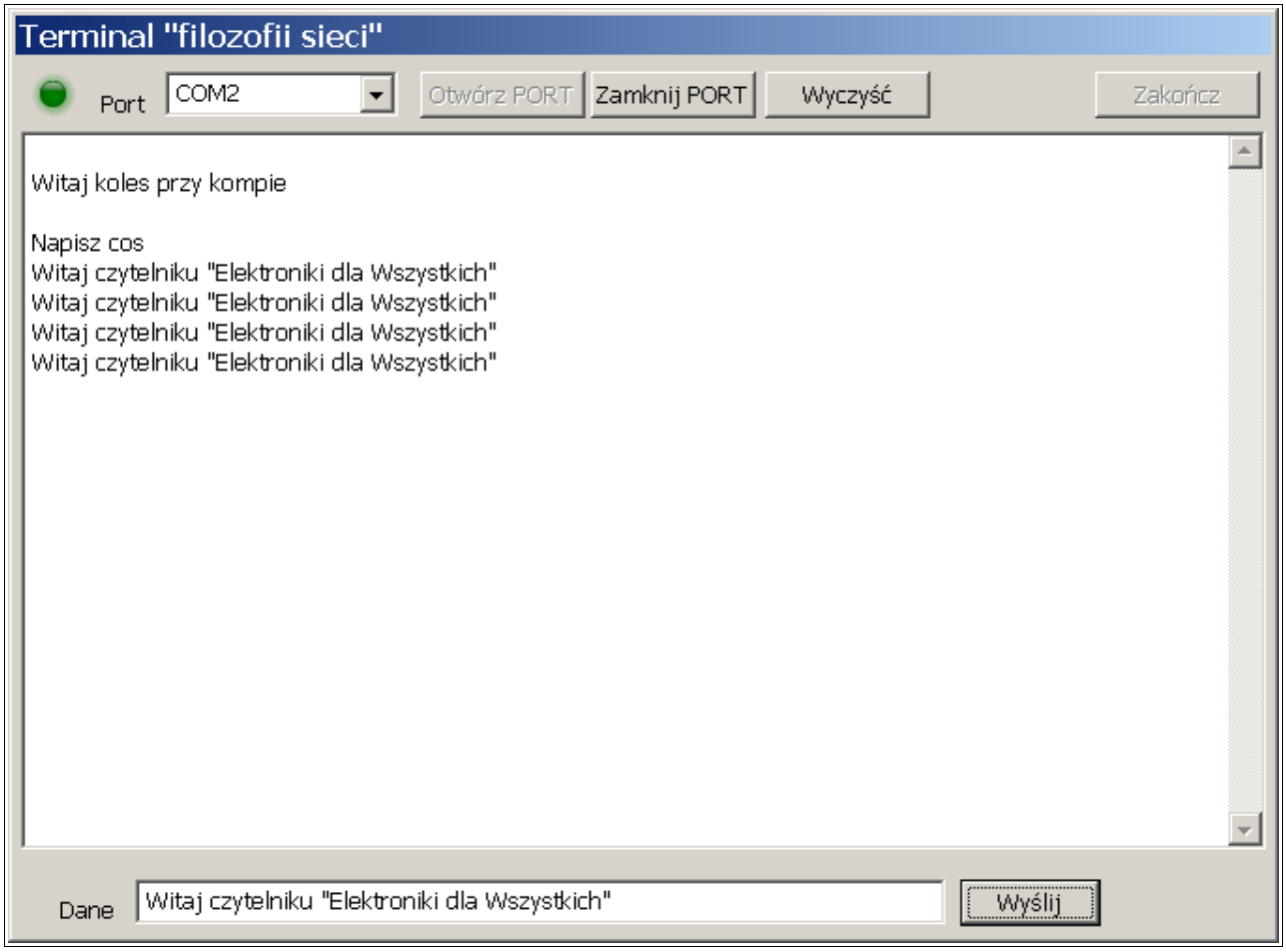
```
  T1TCR = 1 ;
  VICVectAddr5 = ( unsigned long ) T1_IRQHandler ;
  VICVectCntl5 = 15 ;
  VICIntEnable = ( 1 << TIMER1_Interrupt ) ;
  UART0InitEnvir ( ) ;
} /* HardwareInit */


UCHAR Hello1 [ ]      = "\r\nWitaj koles przy kompie\r\n\r\nNapisz cos\r\n" ;
UCHAR Hello2 [ ]      = "\r\nNapisales: " ;


static void HelloMessage ( void )
{
  /*---------------------------------------------------------------------------*/
  UART0SendString ( Hello1 ) ;
} /* HelloMessage */


void ProcessChar ( UCHAR Ch )
{
  USHORT Loop ;
  /*---------------------------------------------------------------------------*/
  if ( Ch < ' ' )
  {
    if ( Index )
    {
      for ( Loop = 0 ; Loop < Index ; Loop ++ )
        UART0Send ( Buffer [ Loop ] ) ;
      UART0Send ( 0x0D ) ;
      UART0Send ( 0x0A ) ;
    } /* if */ ;
    Index = 0 ;
  } /* if * ... (*/
  else
  {
    if ( Index < BufferSize )
    {
      Buffer [ Index ] = Ch ;
    } /* IF */ ;
    Index ++ ;
  } /* if ... else */ ;
} /* ProcessChar */


int main ( void )
{
  /*---------------------------------------------------------------------------*/
  SysInit ( ) ;
  HardwareInit ( )  ;
  SoftwareInit ( ) ;
  UART0HardwInit( FOSC , N81Mode , 9600 ) ;
```

Zainicjowanie kontrolera UART0. Parametr ***FOSC*** jest wyrażeniem (zdefiniowanym jako *#define*) zawierającym w sobie częstotliwość rezonatora kwarcowego i parametry układu PLL. Stała ***N81Mode*** (identyfikuje transmisję 8-bitową z jednym bitem stopu bez parzystości) pochodzi z pliku *serialconst.h*. Trzeci parametr określa oczekiwaną prędkość transmisji.

```
  HelloMessage ( ) ;
  for ( ; ; )
  {
```

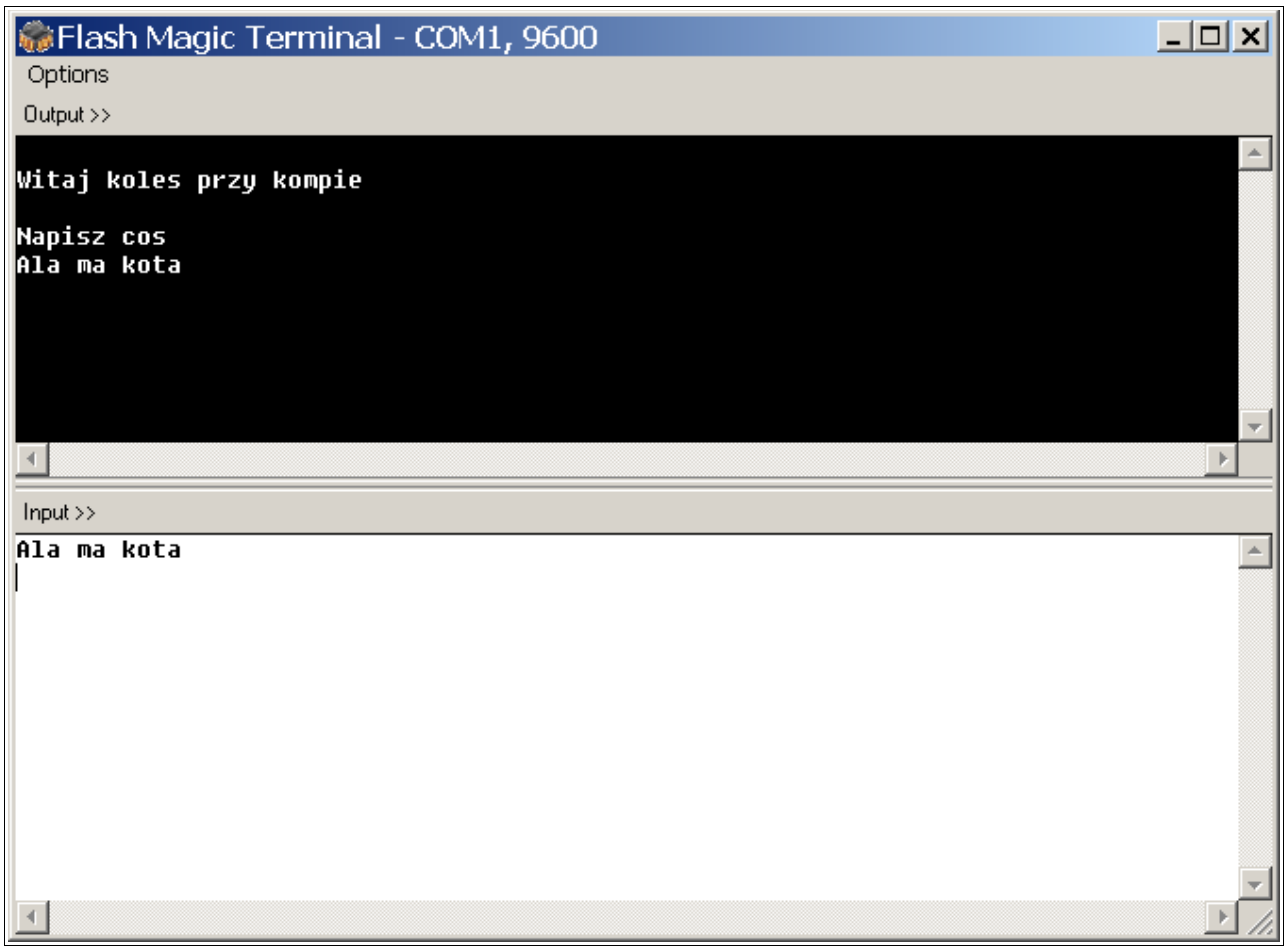
```
    if ( UART0DataPresent ( ) )
      ProcessChar ( UART0GetData ( ) ) ;
  } /* loop */ ;
} /* main */
```

Po skompilowaniu programu i załadowaniu go do pamięci FLASH mikrokontrolera, można cieszyć się nową zabawką (ilustracja 8).



Ilustracja 8: Terminal „filozofii sieci" we współpracy z programem

Możliwości na komunikację jest wiele. Trudno jest wyliczyć choćby część programów spełniających funkcję nadawania i odbierania via kanał asynchronicznej transmisji szeregowej. Taką możliwość, wbrew pozorom, daje program do ładowania kodu do pamięci FLASH mikrokontrolera (FLASHMAGIC). Klikając na odpowiedni przycisk istnieje możliwość przejścia do funkcji terminala oferowanego przez ten program. FLASHMAGIC jest przeznaczony przede wszystkim do programowania pamięci mikrokontrolera a dysponuje możliwościami, jakie odbiegają od głównej jego funkcji. Przykład użycia prezentuje ilustracja 9.



Ilustracja 9: Terminal wbudowany w program FLASHMAGIC

UART1

UART1 jest pełnomodemowym rozwiązaniem interfejsu komunikacyjnego i w maksymalnej konfiguracji zajmuje 8 pinów portu P2.

Do obsługi kontrolera UART1 przewidziane są następujące rejestry:

Symbol rejestru	Adres	Dostęp	Znaczenie
U1RBR	0xE001 0000 (DLAB=0)	odczyt	rejestr, z którego odczytywany jest odebrany kod znaku
U1THR	0xE001 0000 (DLAB=0)	zapis	rejestr, do którego wpisywany jest kod znaku do nadania
U1DLL	0xE001 0000 (DLAB=1)	zapis odczyt	rejestr, do którego wpisywany jest młodsza część podzielnika częstotliwości
U1DLM	0xE001 0004 (DLAB=1)	zapis odczyt	rejestr, do którego wpisywany jest starsza część podzielnika częstotliwości

Symbol rejestru	Adres	Dostęp	Znaczenie
U1IER	0xE001 0004 (DLAB=0)	zapis odczyt	rejestr określający jakie zdarzenia generują przerwanie
U1IIR	0xE001 0008	odczyt	rejestr pozwalający zidentyfikować przyczynę przerwania
U1FCR	0xE001 0008	zapis	rejestr sterujący kolejką FIFO
U1LCR	0xE001 000C	zapis odczyt	rejestr sterujący trybem transmisji
U1MCR	0xE001 0010	zapis odczyt	rejestr sterujący liniami modemowymi
U1LSR	0xE001 0014	odczyt	rejestr statusu odbiornika
U1MSR	0xE001 0018	odczyt	rejestr statusu linii modemowych
U1SCR	0xE001 001C	zapis odczyt	rejestr bez przydzielonej funkcji (w układach 16C550 był dopełnieniem do ośmiu komórek w przestrzeni adresowej)
U1ACR	0xE001 0020	zapis odczyt	
U1FDR	0xE001 0028	zapis odczyt	rejestr zawierający dodatkowe współczynniki pozwalające doprecyzować prędkość transmisji
U1TER	0xE001 0030	zapis odczyt	rejestr, poprzez który można włączać/wyłączać UART1

Istniejące w interfejsie linie modemowe, podzielone są na linie wejściowe (z punktu widzenia mikrokontrolera, może on poprzez te linie odczytać dodatkowe sygnały), oraz linie wyjściowe (z punktu widzenia mikrokontrolera, może on poprzez te linie wysterować jakieś sygnały). Znaczenie linii modemowych wywodzi się ze współpracy z typowymi modemami telekomunikacyjnymi. Linie wyjściowe to:

- DTR – Data Terminal Ready – stan aktywny sygnału oznacza, że UART1 jest gotowy do zestawienia połączenia z zewnętrznym modemem, w złączu DSUB 9 jest to styk numer 4,
- RTS – Request To Send – stan aktywny sygnału oznacza, że UART1 informuje zewnętrzny modem o gotowości do odbioru danych, w złączu DSUB9 jest to styk numer 7.

Wejściowe linie modemowe to:

- CTS – Clear To Send – aktywny stan sygnału oznacza, że zewnętrzny modem sygnalizuje gotowość do odbioru danych, w złączu DSUB 9 jest to styk numer 8,
- DCD – Data Carrier Detect – aktywny stan sygnału oznacza, że zewnętrzny modem jest gotowy do zestawienia połączenia (modem złapał nośną), w złączu DSUB 9 jest to styk numer 1,

- DSR – Data Set Ready – aktywny stan sygnału oznacza, że zewnętrzny modem jest gotowy do komunikacji (ma zestawione stabilne połączenie), w złączu DSUB 9 jest to styk numer 6,
- TI – Ring Indicator – aktywny sygnał oznacza, że zewnętrzny modem odebrał sygnał dzwonienia, w złączu DSUB 9 jest to styk numer 9.

Pozostałe sygnału to:

- RxD – sygnał odbieranych danych szeregowych (z punktu widzenia mikrokontrolera), w złączu DSUB 9 jest to styk numer 2,
- TxD – sygnał nadawanych danych szeregowych (z punktu widzenia mikrokontrolera), w złączu DSUB 9 jest to styk numer 3,
- GND – „masa", potencjał odniesienia dla wszystkich sygnałów, w złączu DSUB 9 jest to styk numer 5.

Do obsługi zespołu UART1 utworzony jest moduł UART1.C (oraz UART1.H). To oprogramowanie udostępnia następujące funkcje:

- **void UART1InitEnvir (void)** – bezparametrowa funkcja, której zadaniem jest skonfigurowanie pinów portu do potrzeb transmisji szeregowej (wyprowadzenia P2[0] .. P2[7] stają się wyprowadzeniami zespołu UART1),
- **void UART1HardwInit (ULONG PeriphClk , UCHAR SerialMode , ULONG SerialSpeed)** – funkcja do zainicjowania trybu i prędkości pracy zespołu UART1, parametr *PeriphClk* jest częstotliwością sygnału zegarowego taktującego mikrokontroler wynikającą sumarycznie z częstotliwości rezonatora kwarcowego i ewentualnego układu PLL zwielokrotniającego tą częstotliwość, parametr *SerialMode* określa tryb pracy kontrolera transmisji szeregowej (jest wartością wprost zapisywaną do rejestru trybu), oraz *SerialSpeed* określa prędkość transmisji szeregowej wyrażoną w bajtach na sekundę (bps),
- **void UART1InputReset (void)** – funkcja zerująca programową kolejkę FIFO związaną z odebranymi w przerwaniach znakami,
- **void UART1Send (UCHAR Data)** – funkcja wysyłająca w kanał szeregowy podany w parametrach znak,
- **void UART1SendString (UCHAR * String)** – funkcja wysyłająca w kanał szeregowy ciąg znaków o dowolnej długości (ciąg musi być zakończony znakiem o kodzie 0),
- **ULONG UART1DataPresent (void)** – funkcja zwracająca informację, czy w kolejce FIFO odbiornika są dane do odczytania,
- **UCHAR UART1GetData (void)** – funkcja odczytująca z programowej kolejki FIFO kolejny znak,
- **void SetOutputModemLine (UCHAR ModemLineControl)** – funkcja ustawiająca (do stanu aktywnego) określone wyjściowe linie modemowe,
- **ResetOutputModemLine (UCHAR ModemLineControl)** – funkcja zerująca określone (ustawiająca do stanu pasywnego) określone wyjściowe linie modemowe,

- **UCHAR GetOutputModemLine (void)** – funkcja zwracająca aktualny stan wyjściowych linii modemowych,
- **UCHAR GetInputModemLine (void)** – funkcja zwracająca aktualny stan wejściowych linii modemowych,
- **UCHAR UART1Working (void)** – funkcja zwracająca status czy UART1 jest w danej chwili zajęty (zajmuje się wysyłaniem danych).

Odbierane przez UART1 dane są gromadzone w programowej kolejce FIFO związanej z odbiornikiem. Każde przerwanie wygenerowane przez odbiornik sprowadza się do odczytania z rejestrów UART1 kodu odebranego znaku oraz dodania jego do wspomnianej kolejki FIFO. Nadawanie znaków prowadzi do realizacji jednego z dwóch wariantów: jeżeli układ UART1 nie jest zajęty, to nadawany znak trafia bezpośrednio w rejestry nadajnika UART1. W przeciwnym wypadku (drugi wariant), kod znaku dodany jest do programowej kolejki FIFO związanej z nadawaniem znaków. Przerwanie wygenerowane przez nadajnik UART1 ewentualnie pobiera z tej kolejki kolejny znak do nadania aż do wyczerpania „zapasu" znaków zgromadzonych w kolejce FIFO. Wielkości buforów cyklicznych (obsługujących określone kolejki FIFO) są następujące:

```
#define InpSerialBuffSize      64
#define OutSerialBuffSize      64
```

Funkcja obsługi przerwania układu UART1. Kontroler ten generuje jeden sygnał przerwania. Z tego względu w implementacji obsługi przerwania należy rozpoznać przyczynę, w wyniku której został wygenerowany sygnał przerwania. Obsługa przerwania jest zbieżna z obsługą przerwania dla UART0 z tym, że dla UART1 dochodzą przerwania dotyczące stanu linii modemowych.

```
__irq void UART1IrqHandler ( void )
  {
    UCHAR U1IrqStatus ;
    UCHAR Data ;
    /*-----------------------------------------*/
    for ( ; ; )
    {
      U1IrqStatus = U1IIR ;
      if ( U1IrqStatus & IIR_PEND )
      {
        VICVectAddr = 0 ;
        return ;
      } /* if */ ;
      switch ( ( U1IrqStatus >> 1 ) & 0x07 )
      {
        case IIR_RLS        :/* 0 1 1 - reciever status */
          Data = U1LSR ;
          break ;
        case IIR_CTI        :/* 1 1 0 - Character Time-out Indicator */
        case IIR_RDA        :/* 0 1 0 - reciever data */
          Data = U1RBR ;
          UART1InpBuffer.Buffer[UART1InpBuffer.WrDataPointer] = Data ;
          UART1InpBuffer . WrDataPointer ++ ;
          if ( UART1InpBuffer . WrDataPointer >= InpSerialBuffSize )
            UART1InpBuffer . WrDataPointer = 0 ;
          break ;
```

```
        case IIR_THRE      :/* 0 0 1 - transmitter */
          while ( U1LSR & LSR_THRE )
          {
            if (UART1OutBuffer.RdDataPointer==UART1OutBuffer.WrDataPointer)
            {
              UART1OutBuffer . BusyStatus = 0 ;
              break ;
            } /* if */ ;
            Data = UART1OutBuffer . Buffer [ UART1OutBuffer . RdDataPointer ] ;
            UART1OutBuffer . RdDataPointer ++ ;
            if ( UART1OutBuffer . RdDataPointer >= OutSerialBuffSize )
              UART1OutBuffer . RdDataPointer = 0 ;
            U1THR = Data ;
          } /* while */ ;
          break ;
        case IIR_MS        :   /* 0 0 0 - Modem status */
          Data = U1MSR ;
```

Odczyt rejestru statusu linii modemowych, odczyt gasi przerwanie zgłaszane przez UART1.

```
          break ;
        default            :  /* Unknown */
          Data = U1LSR ;
          Data = U1RBR ;
          Data = U1MSR ;
          break ;
      } /* switch */ ;
    } /* while */ ;
  } /* UART1IrqHandler */
```

Zaprogramowanie kontrolera UART1 do wymaganego trybu pracy i wymaganej prędkości komunikacyjnej sprowadza się do wywołania odpowiedniej funkcji. W jej parametrach wniesiona jest częstotliwość taktująca mikrokontroler, tryb transmisji (liczba bitów znaku, liczba bitów stopu, rodzaj parzystości) oraz prędkość transmisji (podana w bps – bajtach na sekundę). Realizacja funkcji może być następująca:

```
void UART1HardwInit( ULONG PeriphClk ,
                     UCHAR SerialMode ,
                     ULONG SerialSpeed )
{
  ULONG Fdiv ;
  ULONG Fmod ;
  ULONG LDivider ;
  ULONG HDivider ;
  /*-------------------------------------------------------------------*/
  UART1DisableInterrupt ( ) ;
```

Konfigurowanie układu UART może w trakcie wygenerować przerwanie, więc wszystkie czynności są robione przy zablokowanych przerwaniach od kontrolera UART.

```
  UART1InputReset ( ) ;
  UART1OutputReset ( ) ;
```

Wyzerowanie wszystkich programowych kolejek FIFO (nadajnika i odbiornika).

```
  Fdiv = ( PeriphClk / 16 ) / SerialSpeed ;
  Fmod = ( PeriphClk / 16 ) % SerialSpeed ;
  if ( ( Fmod << 1 ) >= SerialSpeed )
    Fdiv ++ ;
  LDivider = Fdiv % 256 ;
```

```
  HDivider = Fdiv / 256 ;
```

Obliczenie podzielnika do uzyskania oczekiwanej prędkości transmisji. Podzielnik jest rozłożony na dwie części 8-bitowe części. Należy pamiętać, że sam UART dzieli częstotliwość przez 16.

```
  U1FDR = 0x10 ;
```

Rejestr korekcji podzielnika częstotliwości: zawartość rejestru jest podzielona na na dwie części: ***DivAddVal*** jako bity 0..3 oraz ***MulVal*** jako bity 4..7. Wyrażenie na korekcję jest następujące: $(1+\frac{DivAddVal}{MulVal})$, toteż wpis wartości 0x10 nie wnosi żadnej korekcji, gdyż ***DivAddVal***=0 oraz ***MulVal***=1 → finalnie wyrażenie ma wartość 1 – nic nie wnosi.

```
  U1LCR = 0x80 ;
```

Przejście do trybu ***DLAB***, jako ustawienie właściwego bitu w rejestrze ***U0LCR***. Od tej chwili dwa kolejne zapisy do rejestru ***U0DLL*** i ***U0DLM*** dotyczą wpisów podzielnika do prędkości transmisji. Bez ustawienia bitu ***DLAB*** powyższe zapisy dotyczyłyby rejestru danych nadajnika oraz rejestru określającego dopuszczalne przyczyny przerwania. Po dwóch wpisach tryb ***DLAB*** „kasuje się”.

```
  U1DLL = LDivider ;
  U1DLM = HDivider ;
```

Wpis młodszej i starszej części podzielnika częstotliwości do uzyskania oczekiwanej prędkości transmisji.

```
  U1LCR = SerialMode ;
```

Wpis do rejestru determinującego tryb transmisji.

```
  U1MCR = 0 ;
```

Ustawienie linii modemowych do stanu pasywnego.

```
  U1FCR = 0x07 ;
```

Wpis kasujący ewentualną sprzętową kolejkę FIFO zawartą w samym zespole UART oraz determinujący pracę bez kolejki (w zespole UART – nie mylić z programową kolejką FIFO znaków oczekujących na nadanie).

```
  VICVectAddr7  = ( unsigned long ) UART1IrqHandler ;
  VICVectCntl7  = 15 ;
```

„Podłączenie” funkcji ***UART1IrqHandler*** do systemu przerwań. Polega na wpisie do rejestru kontrolera przerwań określanego jako ***VICVectAddr7*** adresu funkcji obsługi przerwania (końcówka nazwy rejestru 7 – oznacza numer przerwania generowanego przez UART1).

```
  UART1EnableInterrupt ( ) ;
```

Określenie priorytetu dla przerwania od UART1 – wpis wartości 15 oznacza najniższy priorytet (0 – oznacza priorytet najwyższy).

```
  U1IER = RecvDataInterruptEnable | TransmDataInterruptEnable ;
```

Wpis do rejestru zezwolenia na przerwania. Program dopuszcza (spodziewa się) przerwań od odbiornika oraz nadajnika. Powyższe stałe odpowiadają odpowiednim pozycjom bitowym w rejestrze.

```
} /* UART1HardwInit */


void UART1InitEnvir ( void )
```

Funkcja do konfigurowania wyprowadzeń mikrokontrolera.

```
{
  /*-------------------------------------------------------------------------*/
  PINSEL4 &= ~ 0x0000FFFF ; /* TXD1, RXD1, CTS1, DCD1, DSR1, DTR1, RI1, RTS1 */
  PINSEL4 |= 0x0000AAAA ;
```

Powyższy wpis do rejestru PINSEL4 oznacza, że:

- P2.0 staje się stykiem TXD1
- P2.1 staje się stykiem RXD1
- P2.2 staje się stykiem CTS1
- P2.3 staje się stykiem DCD1
- P2.4 staje się stykiem DSR1
- P2.5 staje się stykiem DTR1
- P2.6 staje się stykiem RI1

```
  UART1InputReset ( ) ;
  UART1OutputReset ( ) ;
} /* UART1InitEnvir */
```

Przykład programu pokazujący standardową obsługę szeregowych danych.

```
#include <LPC23xx.H>
#include "system.h"
#include "types.h"
#include "irqvectors.h"
#include "serialconst.h"
#include "uart1.h"

#define IRQCallCntLimit                 500

static ULONG IRQCallCounter ;

static void SoftwareInit ( void )
{
  /*-------------------------------------------------------------------------*/
  IRQCallCounter = 0 ;
} /* SoftwareInit */


static __irq void T1_IRQHandler ( void )
{
  /*-------------------------------------------------------------------------*/
  IRQCallCounter ++ ;
  if ( IRQCallCounter >= IRQCallCntLimit )
  {
    IRQCallCounter = 0 ;
  } /* if */ ;
  T1IR = 1 ;
  VICVectAddr = 0 ;
} /* T1_IRQHandler */


static void HardwareInit ( void )
{
  /*-------------------------------------------------------------------------*/
  PINSEL10 = 0 ;
  T1MR0 = ( PCLK / 1000 ) - 1 ;
  T1MCR = 3 ;
  T1TCR = 1 ;
  VICVectAddr5 = ( unsigned long ) T1_IRQHandler ;
  VICVectCntl5 = 15 ;
  VICIntEnable = ( 1 << TIMER1_Interrupt ) ;
```

```
  UART1InitEnvir ( ) ;
} /* HardwareInit */


UCHAR Hello1 [ ]      = "\r\nWitaj koles przy kompie\r\n\r\nNapisz cos\r\n" ;
UCHAR Hello2 [ ]      = "\r\nNapisales: " ;


static void HelloMessage ( void )
{
  /*---------------------------------------------------------------------------*/
  UART1SendString ( Hello1 ) ;
} /* HelloMessage */


static void EncodeControlChar ( UCHAR Ch )
{
  /*---------------------------------------------------------------------------*/
  switch ( Ch )
  {
    case 0x00 :
      UART1SendString ( ( UCHAR * ) "null" ) ;
      break ;
    case 0x01 :
      UART1SendString ( ( UCHAR * ) "CTRL A" ) ;
      break ;
    case 0x02 :
      UART1SendString ( ( UCHAR * ) "CTRL B" ) ;
      break ;
    case 0x03 :
      UART1SendString ( ( UCHAR * ) "CTRL C" ) ;
      break ;
    case 0x04 :
      UART1SendString ( ( UCHAR * ) "CTRL D" ) ;
      break ;
    case 0x05 :
      UART1SendString ( ( UCHAR * ) "CTRL E" ) ;
      break ;
    case 0x06 :
      UART1SendString ( ( UCHAR * ) "CTRL F" ) ;
      break ;
    case 0x07 :
      UART1SendString ( ( UCHAR * ) "CTRL G" ) ;
      break ;
    case 0x08 :
      UART1SendString ( ( UCHAR * ) "CTRL H" ) ;
      break ;
    case 0x09 :
      UART1SendString ( ( UCHAR * ) "CTRL I" ) ;
      break ;
    case 0x0A :
      UART1SendString ( ( UCHAR * ) "CTRL J" ) ;
      break ;
    case 0x0B :
      UART1SendString ( ( UCHAR * ) "CTRL K" ) ;
      break ;
    case 0x0C :
      UART1SendString ( ( UCHAR * ) "CTRL L" ) ;
      break ;
    case 0x0D :
      UART1SendString ( ( UCHAR * ) "CTRL M" ) ;
      break ;
    case 0x0E :
      UART1SendString ( ( UCHAR * ) "CTRL N" ) ;
```

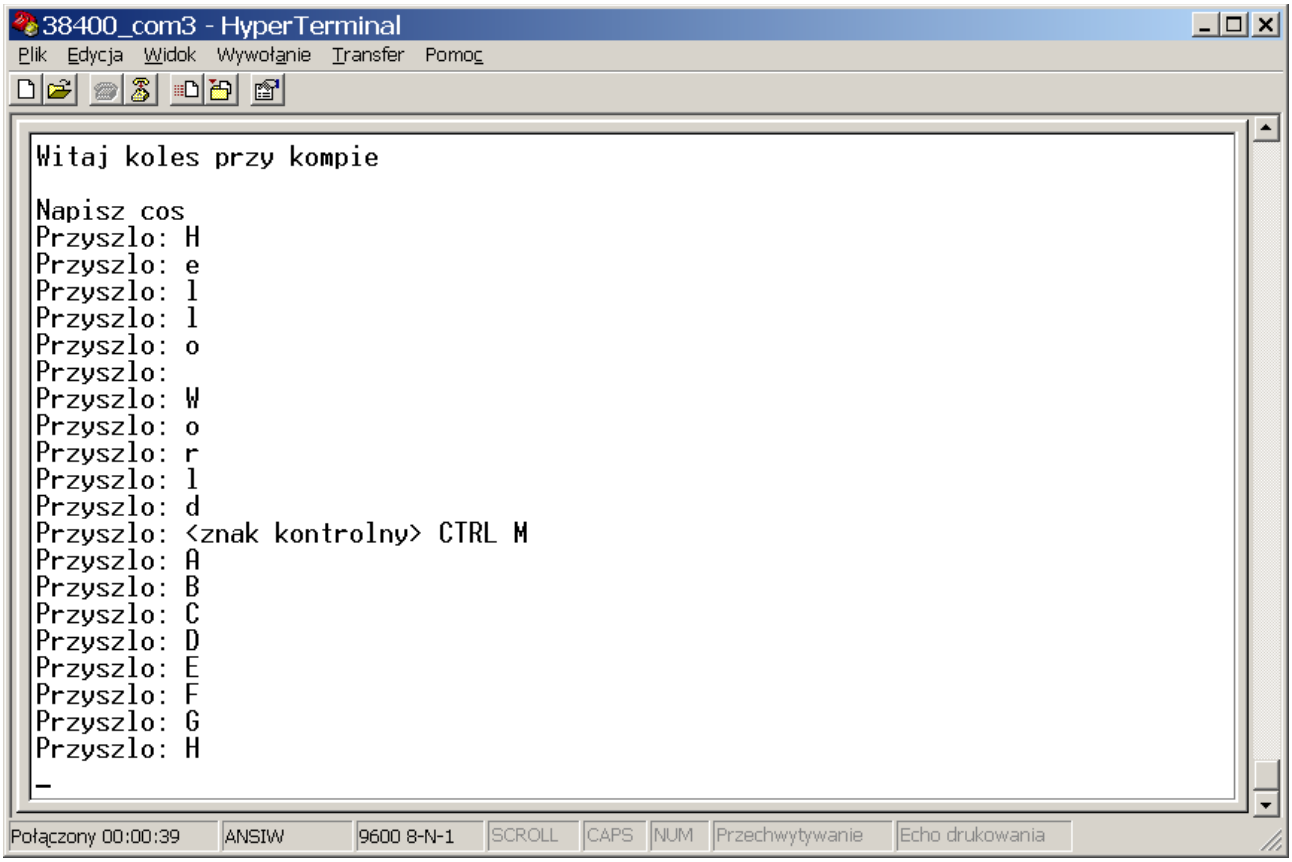
```
      break ;
    case 0x0F :
      UART1SendString ( ( UCHAR * ) "CTRL O" ) ;
      break ;
    case 0x10 :
      UART1SendString ( ( UCHAR * ) "CTRL P" ) ;
      break ;
    case 0x11 :
      UART1SendString ( ( UCHAR * ) "CTRL Q" ) ;
      break ;
    case 0x12 :
      UART1SendString ( ( UCHAR * ) "CTRL R" ) ;
      break ;
    case 0x13 :
      UART1SendString ( ( UCHAR * ) "CTRL S" ) ;
      break ;
    case 0x14 :
      UART1SendString ( ( UCHAR * ) "CTRL T" ) ;
      break ;
    case 0x15 :
      UART1SendString ( ( UCHAR * ) "CTRL U" ) ;
      break ;
    case 0x16 :
      UART1SendString ( ( UCHAR * ) "CTRL V" ) ;
      break ;
    case 0x17 :
      UART1SendString ( ( UCHAR * ) "CTRL W" ) ;
      break ;
    case 0x18 :
      UART1SendString ( ( UCHAR * ) "CTRL X" ) ;
      break ;
    case 0x19 :
      UART1SendString ( ( UCHAR * ) "CTRL Y" ) ;
      break ;
    case 0x1A :
      UART1SendString ( ( UCHAR * ) "CTRL Z" ) ;
      break ;
    case 0x1B :
      UART1SendString ( ( UCHAR * ) "esc" ) ;
      break ;
    case 0x1C :
      UART1SendString ( ( UCHAR * ) "1C hex" ) ;
      break ;
    case 0x1D :
      UART1SendString ( ( UCHAR * ) "1D hex" ) ;
      break ;
    case 0x1E :
      UART1SendString ( ( UCHAR * ) "1E hex" ) ;
      break ;
    case 0x1F :
      UART1SendString ( ( UCHAR * ) "1F hex" ) ;
      break ;
  } /* switch */ ;
} /* EncodeControlChar */


static UCHAR Prompt [ ]        = "Przyszlo: " ;
static UCHAR ControlChar [ ]   = "<znak kontrolny> " ;

void ProcessChar ( UCHAR Ch )
{
  /*-------------------------------------------------------------------------*/
  UART1SendString ( Prompt ) ;
  if ( Ch < ' ' )
```

```
  {
    UART1SendString ( ControlChar ) ;
    EncodeControlChar ( Ch ) ;
  } /* if ... */
  else
  {
    UART1Send ( Ch ) ;
  } /* if ... else */ ;
  UART1Send ( 0x0D ) ;
  UART1Send ( 0x0A ) ;
} /* ProcessChar */


int main ( void )
{
  /*-------------------------------------------------------------------------*/
  SysInit ( ) ;
  HardwareInit ( )  ;
  SoftwareInit ( ) ;
  UART1HardwInit( FOSC , N81Mode , 9600 ) ;
  HelloMessage ( ) ;
  for ( ; ; )
  {
    if ( UART1DataPresent ( ) )
      ProcessChar ( UART1GetData ( ) ) ;
  } /* loop */ ;
} /* main */
```

Ilustracja 10: Obsługa UART1

Po skompilowaniu programu i załadowaniu jego kodu do pamięci FLASH mamy „nową zabawkę”.

Zastosowanie jakiegokolwiek programu pełniącego rolę emulatora terminala (przykładowo windowsowy HYPERTERMINAL) pozwala na sprawdzenie funkcjonalności programu w zakresie podstawowych funkcji nadawania i odbierania znaków (ilustracja 11).



Ilustracja 11: UART1 – transmisja danych

Nadawanie i odbieranie danych szeregowych nie wyczerpuje zakresu funkcjonalności zespołu UART1. Drugi program prezentacyjny dotyczy wyjściowych linii modemowych. Po połączeniu z emulatorem terminala, można „zadać odpowiednie polecenia dotyczące manipulacji liniami modemowymi". Jest pewien problem dotyczący rzeczywistego stwierdzenia stanu linii modemowych. Niby można zestaw badawczy przyłączyć do komputera, ale tu występuje pewien problem. Wyjście UART1 wraz z liniami modemowymi jest identyczne jak w komputerach PC, więc „to się nie styka". Można by ewentualnie zrobić jakiś specjalizowany kabelek do połączenia, przykładowo wyjściowe linie modemowe (zestawu badawczego) połączyć z wejściowymi (innymi) liniami modemowymi i mając wiedzę, co jest z czym połączone sygnalizować w komputerze odpowiednie stany. Trzeba pamiętać, że budowa modemu telekomunikacyjnego jest „lustrzana". To co w złączu zestawu badawczego jest wyjściem, w modemie jest wejściem (pomimo, że sygnał nazywa się identycznie). Analogicznie wejściowe linie modemowe w zestawie odpowiadają wyjściowym liniom w modemie. Powinno to być zrozumiałe (przykładowo sygnał RI: modem odbiera sygnał dzwonienia i sygnalizuje ten fakt przez wyjściową (modemu) linię na wejściową (w module, komputerze) linię modemową. Zmiana stanu pozwala oprogramowaniu w komputerze (module) na wysłanie polecenia do modemu, by odebrał połączenie.

Mając na uwadze powyższe problemy, zastosuję najprostsze z możliwych rozwiązań: zmierzę napięcie na odpowiednich stykach woltomierzem. Stan aktywny (jako wyjście z mikrokontrolera) odpowiada logicznemu zeru (stan pasywny to logiczna jedynka na wyjściu portu). Te stany logiczne są translowane przez interfejs RS232 do napięcia dodatniego dla stanów aktywnych (5V do 12V w zależności od

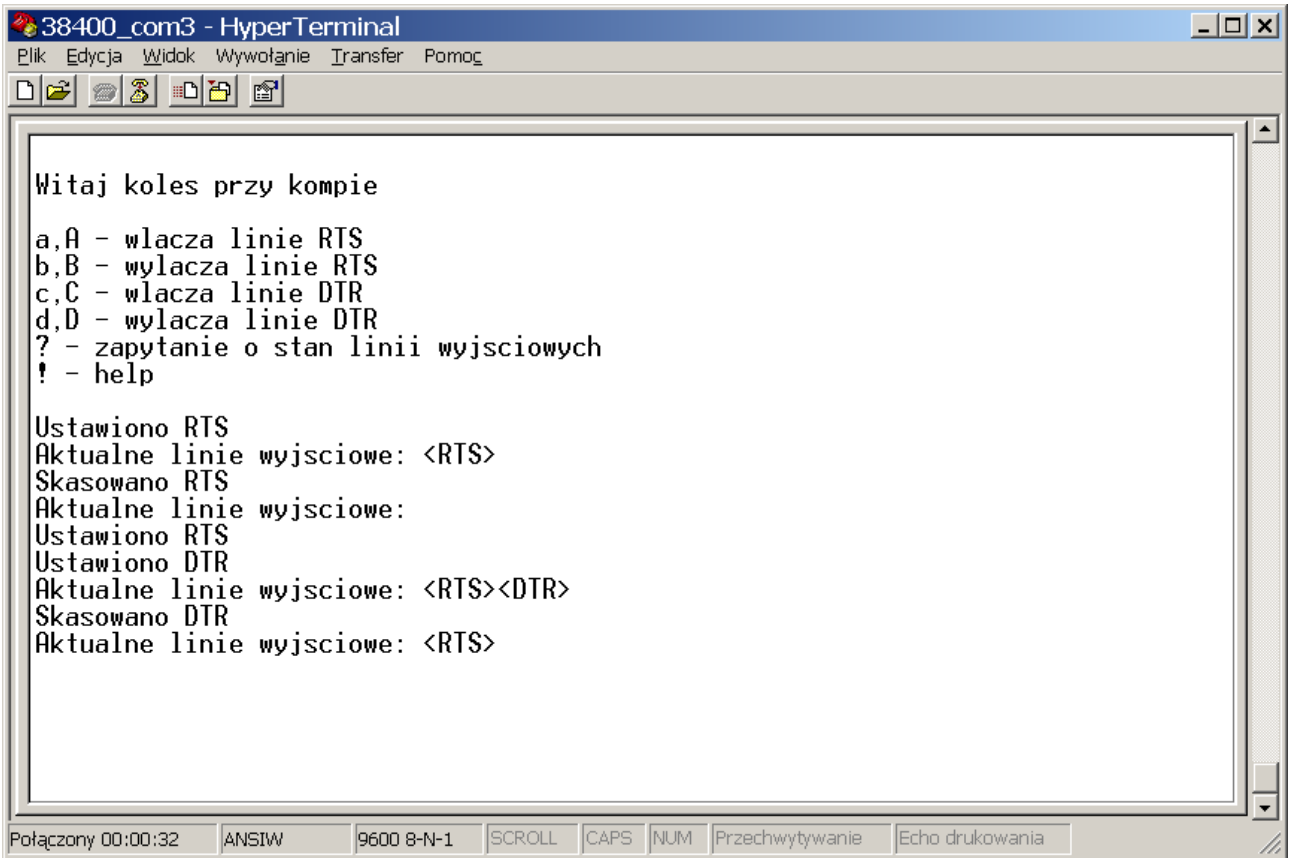
modelu interfejsu) oraz napięcia ujemnego dla stanów pasywnych (-5V do -12V w zależności od modelu interfejsu).

Program „użytkowy” jest następujący:

```
#include <LPC23xx.H>
#include "system.h"
#include "types.h"
#include "irqvectors.h"
#include "serialconst.h"
#include "uart1.h"

#define IRQCallCntLimit                 500

static ULONG IRQCallCounter ;

static void SoftwareInit ( void )
{
  /*-------------------------------------------------------------------------*/
  IRQCallCounter = 0 ;
} /* SoftwareInit */


static __irq void T1_IRQHandler ( void )
{
  /*-------------------------------------------------------------------------*/
  IRQCallCounter ++ ;
  if ( IRQCallCounter >= IRQCallCntLimit )
  {
    IRQCallCounter = 0 ;
  } /* if */ ;
  T1IR = 1 ;
  VICVectAddr = 0 ;
} /* T1_IRQHandler */


static void HardwareInit ( void )
{
  /*-------------------------------------------------------------------------*/
  PINSEL10 = 0 ;
  T1MR0 = ( PCLK / 1000 ) - 1 ;
  T1MCR = 3 ;
  T1TCR = 1 ;
  VICVectAddr5 = ( unsigned long ) T1_IRQHandler ;
  VICVectCntl5 = 15 ;
  VICIntEnable = ( 1 << TIMER1_Interrupt ) ;
  UART1InitEnvir ( ) ;
} /* HardwareInit */


UCHAR Hello1 [ ]      = "\r\nWitaj koles przy kompie\r\n\r\n" ;
UCHAR Hello2 [ ]      = "a,A - wlacza linie RTS\r\nb,B - wylacza linie
RTS\r\nc,C - wlacza linie DTR\r\nd,D - wylacza linie DTR\r\n? - zapytanie o stan
linii wyjsciowych\r\n! - help\r\n\r\n" ;


static void HelloMessage ( void )
{
  /*-------------------------------------------------------------------------*/
  UART1SendString ( Hello1 ) ;
  UART1SendString ( Hello2 ) ;
} /* HelloMessage */
```

```
static UCHAR Prompt [ ]    = "Aktualne linie wyjsciowe: " ;
static UCHAR RTSTxt [ ]    = "<RTS>" ;
static UCHAR DTRTxt [ ]    = "<DTR>" ;
static UCHAR Info01 [ ]    = "Ustawiono RTS\r\n" ;
static UCHAR Info02 [ ]    = "Skasowano RTS\r\n" ;
static UCHAR Info03 [ ]    = "Ustawiono DTR\r\n" ;
static UCHAR Info04 [ ]    = "Skasowano DTR\r\n" ;

void ProcessChar ( UCHAR Ch )
{
  UCHAR ModemStatus ;
  /*-------------------------------------------------------------------------*/
  if ( ( Ch == 'a' ) || ( Ch == 'A' ) )
  {
    UART1SendString ( Info01 ) ;
    SetOutputModemLine ( RTSControlUARTCode ) ;
  } /* if */ ;
  if ( ( Ch == 'b' ) || ( Ch == 'B' ) )
  {
    UART1SendString ( Info02 ) ;
    ResetOutputModemLine ( RTSControlUARTCode ) ;
  } /* if */ ;
  if ( ( Ch == 'c' ) || ( Ch == 'C' ) )
  {
    UART1SendString ( Info03 ) ;
    SetOutputModemLine ( DTRControlUARTCode ) ;
  } /* if */ ;
  if ( ( Ch == 'd' ) || ( Ch == 'D' ) )
  {
    UART1SendString ( Info04 ) ;
    ResetOutputModemLine ( DTRControlUARTCode ) ;
  } /* if */ ;
  if ( Ch == '!' )
    UART1SendString ( Hello2 ) ;
  if ( Ch == '?' )
  {
    UART1SendString ( Prompt ) ;
    ModemStatus = GetOutputModemLine ( ) ;
    if ( ModemStatus & RTSControlUARTCode )
      UART1SendString ( RTSTxt ) ;
    if ( ModemStatus & DTRControlUARTCode )
      UART1SendString ( DTRTxt ) ;
    UART1Send ( 0x0D ) ;
    UART1Send ( 0x0A ) ;
  } /* if */ ;
} /* ProcessChar */


int main ( void )
{
  /*-------------------------------------------------------------------------*/
  SysInit ( ) ;
  HardwareInit ( )  ;
  SoftwareInit ( ) ;
  UART1HardwInit( FOSC , N81Mode , 9600 ) ;
  HelloMessage ( ) ;
  for ( ; ; )
  {
    if ( UART1DataPresent ( ) )
      ProcessChar ( UART1GetData ( ) ) ;
  } /* loop */ ;
} /* main */
```

Po skompilowaniu i wgraniu programu w proca, mamy kolejną zabawkę (ilustracja 12).



Ilustracja 12: UART1 – manipulacja wyjściowymi liniami modemowymi

Pomiary napięcia na złączu szeregowym związanych z UART1 zgadają się ze stanem wyświetlanym na ekranie emulatora terminala.

Kolejny program dotyczy wejściowych linii modemowych. Tu również istnieje pewien problem „badawczy”. Nie chcąc budować jakiegoś specjalizowanego układu pomiarowego pozostaje wymuszać odpowiednie stany drucikiem, którego drugi koniec jest przyłączony do napięcia dodatniego (może być +5V do +12V). Wejście interfejsu RS232 pozostawione samopas odpowiada stanowi pasywnemu. Podanie napięcia (ręcznie drucikiem) wymusza zmianę stanu na aktywny. Przy „ręcznym” grzebaniu po stykach daje się zauważyć zjawisko „dzwonienia styków”. Wydawać by się mogło, że raz dotknięty kabelkiem styk powinien zareagować pojedynczą zmianą stanu. W rzeczywistości generuje serię zmian, co daje się zauważyć na ekranie emulatora terminala.

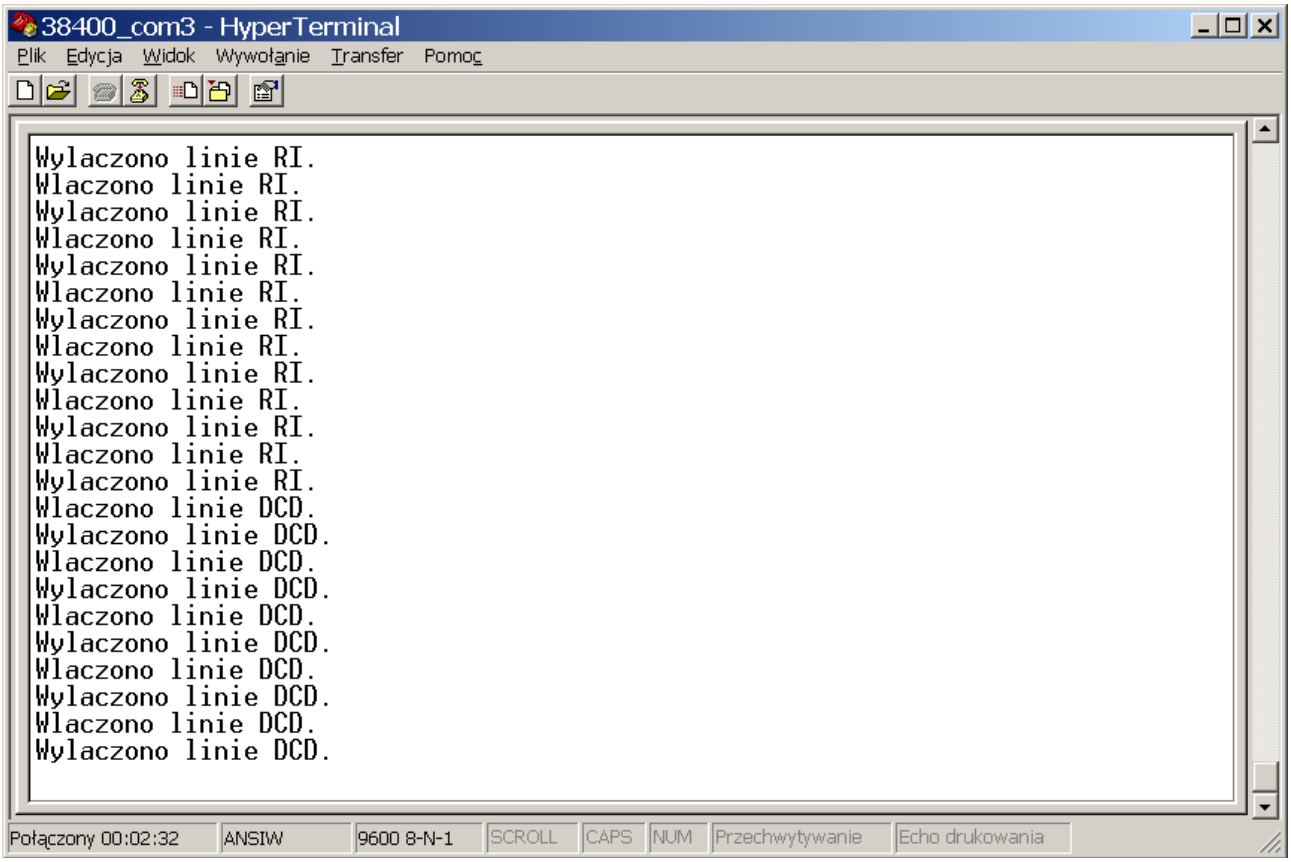
Program badawczy jest następujący:

```
#include <LPC23xx.H>
#include "system.h"
#include "types.h"
#include "irqvectors.h"
#include "serialconst.h"
#include "uart1.h"
```

```
#define IRQCallCntLimit                 500

static ULONG IRQCallCounter ;
static UCHAR LastDCDModemLine ;
static UCHAR LastRIModemLine ;
static UCHAR LastDSRModemLine ;
static UCHAR LastCTSModemLine ;

static void SoftwareInit ( void )
{
  /*-------------------------------------------------------------------------*/
  IRQCallCounter = 0 ;
  LastDCDModemLine = 0 ;
  LastRIModemLine = 0 ;
  LastDSRModemLine = 0 ;
  LastCTSModemLine = 0 ;
} /* SoftwareInit */


static __irq void T1_IRQHandler ( void )
{
  /*-------------------------------------------------------------------------*/
  IRQCallCounter ++ ;
  if ( IRQCallCounter >= IRQCallCntLimit )
  {
    IRQCallCounter = 0 ;
  } /* if */ ;
  T1IR = 1 ;
  VICVectAddr = 0 ;
} /* T1_IRQHandler */


static void HardwareInit ( void )
{
  /*-------------------------------------------------------------------------*/
  PINSEL10 = 0 ;
  T1MR0 = ( PCLK / 1000 ) - 1 ;
  T1MCR = 3 ;
  T1TCR = 1 ;
  VICVectAddr5 = ( unsigned long ) T1_IRQHandler ;
  VICVectCntl5 = 15 ;
  VICIntEnable = ( 1 << TIMER1_Interrupt ) ;
  UART1InitEnvir ( ) ;
} /* HardwareInit */


UCHAR Hello1 [ ]      = "\r\nWitaj koles przy kompie\r\n\r\n" ;

static void HelloMessage ( void )
{
  /*-------------------------------------------------------------------------*/
  UART1SendString ( Hello1 ) ;
} /* HelloMessage */


void ProcessChar ( UCHAR Ch )
{
  /*-------------------------------------------------------------------------*/
  UART1Send ( Ch ) ;
  UART1Send ( 0x0D ) ;
  UART1Send ( 0x0A ) ;
} /* ProcessChar */
```

```
static UCHAR Message01 [ ]   = "Wlaczono linie DCD.\r\n" ;
static UCHAR Message02 [ ]   = "Wylaczono linie DCD.\r\n" ;
static UCHAR Message03 [ ]   = "Wlaczono linie RI.\r\n" ;
static UCHAR Message04 [ ]   = "Wylaczono linie RI.\r\n" ;
static UCHAR Message05 [ ]   = "Wlaczono linie DSR.\r\n" ;
static UCHAR Message06 [ ]   = "Wylaczono linie DSR.\r\n" ;
static UCHAR Message07 [ ]   = "Wlaczono linie CTS.\r\n" ;
static UCHAR Message08 [ ]   = "Wylaczono linie CTS.\r\n" ;

int main ( void )
{
  UCHAR ModemStatus ;
  /*-------------------------------------------------------------------------*/
  SysInit ( ) ;
  HardwareInit ( )  ;
  SoftwareInit ( ) ;
  UART1HardwInit( FOSC , N81Mode , 9600 ) ;
  HelloMessage ( ) ;
  for ( ; ; )
  {
    if ( UART1DataPresent ( ) )
      ProcessChar ( UART1GetData ( ) ) ;
    ModemStatus = GetInputModemLine ( ) ;
    if ( LastDCDModemLine )
    {
      if ( ! ( ModemStatus & DCDControlUARTCode ) )
      {
        UART1SendString ( Message02 ) ;
        LastDCDModemLine = 0 ;
      } /* if */ ;
    } /* if ... */
    else
    {
      if ( ModemStatus & DCDControlUARTCode )
      {
        UART1SendString ( Message01 ) ;
        LastDCDModemLine = 1 ;
      } /* if */ ;
    } /* if ... else */ ;
    if ( LastRIModemLine )
    {
      if ( ! ( ModemStatus & RIControlUARTCode ) )
      {
        UART1SendString ( Message04 ) ;
        LastRIModemLine = 0 ;
      } /* if */ ;
    } /* if ... */
    else
    {
      if ( ModemStatus & RIControlUARTCode )
      {
        UART1SendString ( Message03 ) ;
        LastRIModemLine = 1 ;
      } /* if */ ;
    } /* if ... else */ ;
    if ( LastDSRModemLine )
    {
      if ( ! ( ModemStatus & DSRControlUARTCode ) )
      {
        UART1SendString ( Message06 ) ;
        LastDSRModemLine = 0 ;
      } /* if */ ;
    } /* if ... */
    else
```

```
    {
      if ( ModemStatus & DSRControlUARTCode )
      {
        UART1SendString ( Message05 ) ;
        LastDSRModemLine = 1 ;
      } /* if */ ;
    } /* if ... else */ ;
    if ( LastCTSModemLine )
    {
      if ( ! ( ModemStatus & CTSControlUARTCode ) )
      {
        UART1SendString ( Message08 ) ;
        LastCTSModemLine = 0 ;
      } /* if */ ;
    } /* if ... */
    else
    {
      if ( ModemStatus & CTSControlUARTCode )
      {
        UART1SendString ( Message07 ) ;
        LastCTSModemLine = 1 ;
      } /* if */ ;
    } /* if ... else */ ;
  } /* loop */ ;
} /* main */
```



Ilustracja 13: UART1 – eksperymenty z wejściowymi liniami modemowymi

UART2

W interfejsie UART2, podobnie jak w UART0, nie ma linii modemowych (ten kanał transmisyjny jest okrojony). Z tego powodu nie zawiera on również w przestrzeni adresowej rejestrów dedykowanych do obsługi linii modemowych.

Do obsługi kontrolera UART2 przewidziane są następujące rejestry:

Symbol rejestru	Adres	Dostęp	Znaczenie
U2RBR	0xE007 8000 (DLAB=0)	odczyt	rejestr, z którego odczytywany jest odebrany kod znaku
U2THR	0xE007 8000 (DLAB=0)	zapis	rejestr, do którego wpisywany jest kod znaku do nadania
U2DLL	0xE007 8000 (DLAB=1)	zapis odczyt	rejestr, do którego wpisywana jest młodsza część podzielnika częstotliwości
U2DLM	0xE007 8004 (DLAB=1)	zapis odczyt	rejestr, do którego wpisywany jest starsza część podzielnika częstotliwości
U2IER	0xE007 8004 (DLAB=0)	zapis odczyt	rejestr określający jakie zdarzenia generują przerwanie
U2IIR	0xE007 8008	odczyt	rejestr pozwalający zidentyfikować przyczynę przerwania
U2FCR	0xE007 8008	zapis	rejestr sterujący kolejką FIFO
U2LCR	0xE007 800C	zapis odczyt	rejestr sterujący trybem transmisji
U2LSR	0xE007 8014	odczyt	rejestr statusu odbiornika
U2SCR	0xE007 801C	zapis odczyt	rejestr bez przydzielonej funkcji (w układach 16C550 był dopełnieniem do ośmiu komórek w przestrzeni adresowej)
U2ACR	0xE007 8020	zapis odczyt	
U2FDR	0xE007 8028	zapis odczyt	rejestr zawierający dodatkowe współczynniki pozwalające doprecyzować prędkość transmisji
U2TER	0xE007 8030	zapis odczyt	rejestr, poprzez który można włączać/wyłączać UART2

Wybrane rejestry (zaznaczone kolorem) mają tą samą alokację w przestrzeni adresowej. Dostęp do poszczególnych rejestrów jest rozróżniany na podstawie typu operacji (zapis/odczyt) oraz wskaźnika DLAB. Przejście do tego trybu następuje po ustawieniu bitu b7 w rejestrze U2LCR (zapisaniu do wymienionego rejestru takiej sekwencji bitowej, gdzie ustawiony jest odpowiedni bit).

Do obsługi zespołu UART2 utworzony jest moduł UART2.C (oraz UART2.H). To oprogramowanie udostępnia następujące funkcje:

- **void UART2InitEnvir (void)** – bezparametrowa funkcja, której zadaniem jest skonfigurowanie pinów portu do potrzeb transmisji szeregowej

(wyprowadzenia P2[8] i P2[9] stają się wyprowadzeniami zespołu UART2),

- **void UART2HardwInit (ULONG PeriphClk , UCHAR SerialMode , ULONG SerialSpeed)** – funkcja do zainicjowania trybu i prędkości pracy zespołu UART2, parametr *PeriphClk* jest częstotliwością sygnału zegarowego taktującego mikrokontroler wynikającą sumarycznie z częstotliwości rezonatora kwarcowego i ewentualnego układu PLL zwielokrotniającego tą częstotliwość, parametr *SerialMode* określa tryb pracy kontrolera transmisji szeregowej (jest wartością wprost zapisywaną do rejestru trybu), oraz *SerialSpeed* określa prędkość transmisji szeregowej wyrażoną w bajtach na sekundę (bps),
- **void UART2InputReset (void)** – funkcja zerująca programową kolejkę FIFO związaną z odebranymi w przerwaniach znakami,
- **void UART2Send (UCHAR Data)** – funkcja wysyłająca w kanał szeregowy podany w parametrach znak,
- **void UART2SendString (UCHAR * String)** – funkcja wysyłająca w kanał szeregowy ciąg znaków o dowolnej długości (ciąg musi być zakończony znakiem o kodzie 0),
- **ULONG UART2DataPresent (void)** – funkcja zwracająca informację, czy w kolejce FIFO odbiornika są dane do odczytania,
- **UCHAR UART2GetData (void)** – funkcja odczytująca z programowej kolejki FIFO kolejny znak,
- **UCHAR UART2Working (void)** – funkcja zwracająca status czy UART2 jest w danej chwili zajęty (zajmuje się wysyłaniem danych).

Odbierane przez UART2 dane są gromadzone w programowej kolejce FIFO związanej z odbiornikiem. Każde przerwanie wygenerowane przez odbiornik sprowadza się do odczytania z rejestrów UART2 kodu odebranego znaku oraz dodania jego do wspomnianej kolejki FIFO. Nadawanie znaków prowadzi do realizacji jednego z dwóch wariantów: jeżeli układ UART2 nie jest zajęty, to nadawany znak trafia bezpośrednio w rejestry nadajnika UART2. W przeciwnym wypadku (drugi wariant), kod znaku dodany jest do programowej kolejki FIFO związanej z nadawaniem znaków. Przerwanie wygenerowane przez nadajnik UART0 ewentualnie pobiera z tej kolejki kolejny znak do nadania aż do wyczerpania „zapasu” znaków zgromadzonych w kolejce FIFO. Obsługa praktycznie jest kopią rozwiązania dla UART0. Jedyne różnice dotyczą dostępów do rejestrów związanych z UART2 (zamiast UART0), funkcja obsługi przerwań *UART2IrqHandler* jest związana z wektorem przerwań o numerze 28 (ma to swoje odbicie w zainstalowaniu wektora przerwań), inne są maski w funkcjach blokady przerwań od zespołu UART i odmiennie jest skonfigurowany port (rejestr typu PINSEL), by wyprowadzenia P2.8 i P2.9 były skojarzone z sygnałami TXD2 i RXD2. Istotnym szczegółem, który nie występował w obsłudze UART0 i UART1, jest konieczność „włączenia zasilania” dla układu UART2. Po sygnale reset, zespół ten nie jest zasilony, toteż wymaga „ręcznego” włączenia zasilania (wpis do rejestru ***PCONP***), jak pokazuje poniższa funkcja:

```
void UART2InitEnvir ( void )
```

```
{
  PCONP |= PCUART2 ;
```

Włączenie zasilania.

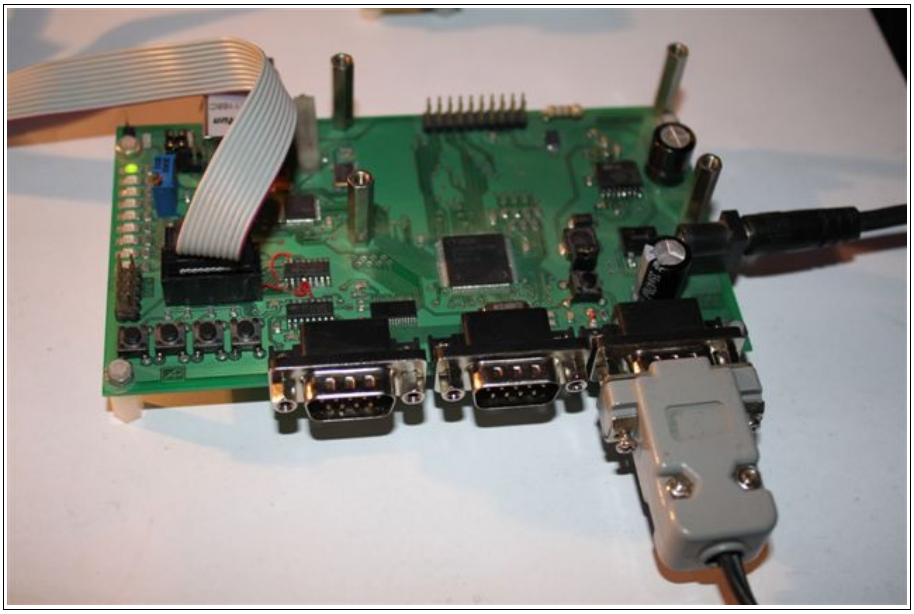
```
  PINSEL4 &= ~ 0x000F0000  ;
  PINSEL4 |= 0x000A0000 ;  // P2.8 -> TXD2  P2.9 -> RXD2
  UART2InputReset ( ) ;
  UART2OutputReset ( ) ;
} /* UART2InitEnvir */
```

Program użytkowy wygląda następująco:

```
#include <LPC23xx.H>
#include "system.h"
#include "types.h"
#include "irqvectors.h"
#include "serialconst.h"
#include "uart2.h"

#define IRQCallCntLimit                 500
#define BufferSize                      64

static ULONG IRQCallCounter ;
static USHORT Index ;
static UCHAR Buffer [ BufferSize ] ;

static void SoftwareInit ( void )
{
  /*-------------------------------------------------------------------------*/
  IRQCallCounter = 0 ;
  Index = 0 ;
} /* SoftwareInit */


static __irq void T1_IRQHandler ( void )
{
  /*-------------------------------------------------------------------------*/
  IRQCallCounter ++ ;
  if ( IRQCallCounter >= IRQCallCntLimit )
  {
    IRQCallCounter = 0 ;
  } /* if */ ;
  T1IR = 1 ;
  VICVectAddr = 0 ;
} /* T1_IRQHandler */


static void HardwareInit ( void )
{
  /*-------------------------------------------------------------------------*/
  PINSEL10 = 0 ;
  T1MR0 = ( PCLK / 1000 ) - 1 ;
  T1MCR = 3 ;
  T1TCR = 1 ;
  VICVectAddr5 = ( unsigned long ) T1_IRQHandler ;
  VICVectCntl5 = 15 ;
  VICIntEnable = ( 1 << TIMER1_Interrupt ) ;
  UART2InitEnvir ( ) ;
} /* HardwareInit */


UCHAR Hello1 [ ]      = "\r\nWitaj koles przy kompie\r\n\r\nNapisz cos\r\n" ;
UCHAR Hello2 [ ]      = "\r\nNapisales: " ;
```

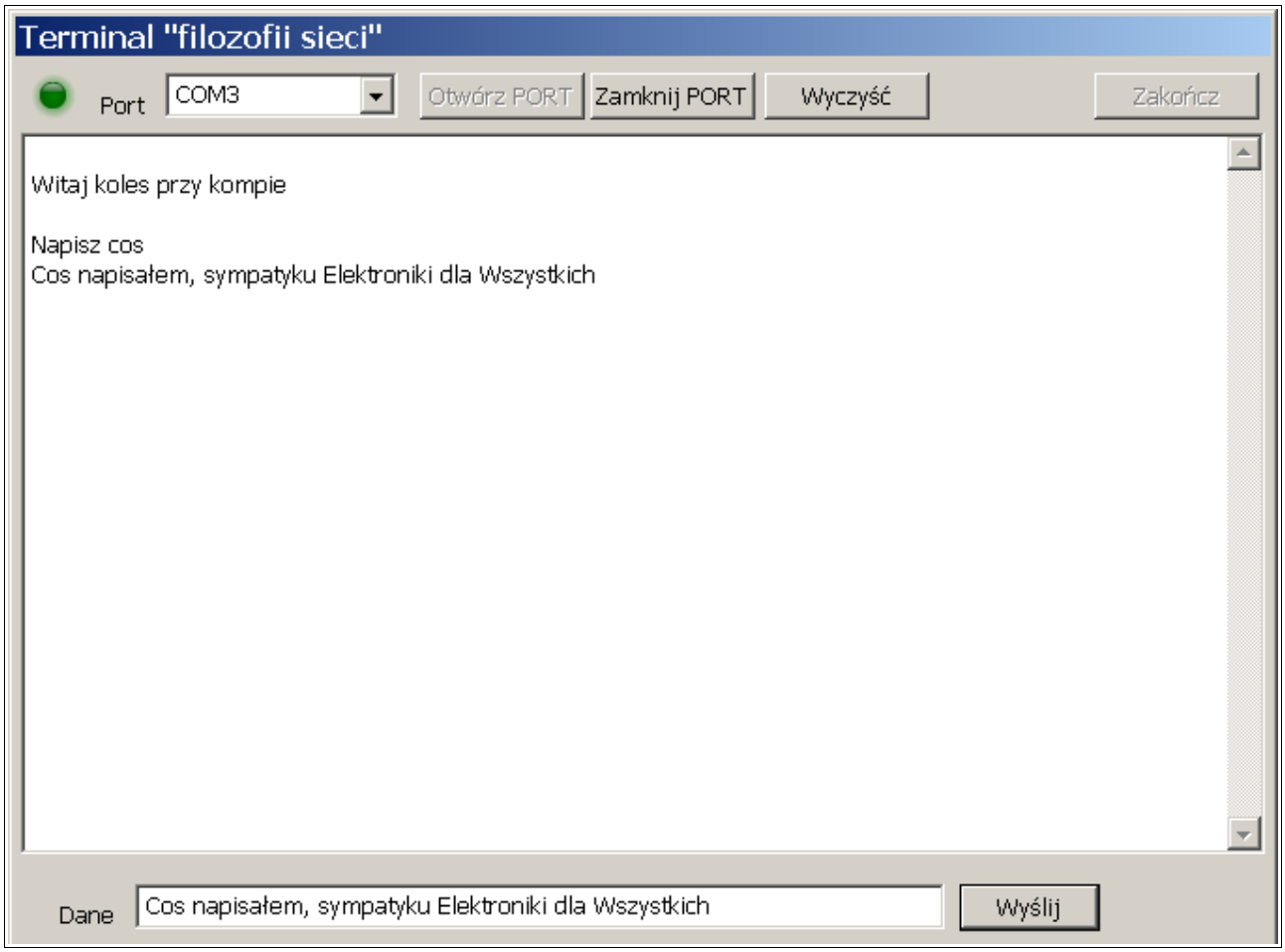
```
static void HelloMessage ( void )
{
  /*---------------------------------------------------------------------------*/
  UART2SendString ( Hello1 ) ;
} /* HelloMessage */


void ProcessChar ( UCHAR Ch )
{
  USHORT Loop ;
  /*---------------------------------------------------------------------------*/
  if ( Ch < ' ' )
  {
    if ( Index )
    {
      for ( Loop = 0 ; Loop < Index ; Loop ++ )
        UART2Send ( Buffer [ Loop ] ) ;
      UART2Send ( 0x0D ) ;
      UART2Send ( 0x0A ) ;
    } /* if */ ;
    Index = 0 ;
  } /* if ... */
  else
  {
    if ( Index < BufferSize )
    {
      Buffer [ Index ] = Ch ;
    } /* IF */ ;
    Index ++ ;
  } /* if ... else */ ;
} /* ProcessChar */


int main ( void )
{
  /*---------------------------------------------------------------------------*/
  SysInit ( ) ;
  HardwareInit ( )  ;
  SoftwareInit ( ) ;
  UART2HardwInit( FOSC , N81Mode , 9600 ) ;
  HelloMessage ( ) ;
  for ( ; ; )
  {
    if ( UART2DataPresent ( ) )
      ProcessChar ( UART2GetData ( ) ) ;
  } /* loop */ ;
} /* main */
```

Po skompilowaniu programu i załadowania go do pamięci FLASH mikrokontrolera mamy kolejną „zabawkę” (ilustracja 14).



Ilustracja 14: Obsługa UART2

Po połączeniu modułu z emulatorem terminala, mamy „maszynę do pisania”, program odsyła na ekran każdy napisany wiersz (ilustracja 15).



Ilustracja 15: "Maszyna do pisania"

UART3

Kolejny kontroler transmisji szeregowej UART3 podobnie jak UART0 oraz UART2, nie zawiera obsługi linii modemowych. Z tego powodu nie zawiera on również w przestrzeni adresowej rejestrów dedykowanych do obsługi linii modemowych.

Do obsługi kontrolera UART3 przewidziane są następujące rejestry:

Symbol rejestru	Adres	Dostęp	Znaczenie
U3RBR	0xE007 C000 (DLAB=0)	odczyt	rejestr, z którego odczytywany jest odebrany kod znaku
U3THR	0xE007 C000 (DLAB=0)	zapis	rejestr, do którego wpisywany jest kod znaku do nadania
U3DLL	0xE007 C000 (DLAB=1)	zapis odczyt	rejestr, do którego wpisywana jest młodsza część podzielnika częstotliwości
U3DLM	0xE007 C004 (DLAB=1)	zapis odczyt	rejestr, do którego wpisywany jest starsza część podzielnika częstotliwości
U3IER	0xE007 C004 (DLAB=0)	zapis odczyt	rejestr określający jakie zdarzenia generują przerwanie
U3IIR	0xE007 C008	odczyt	rejestr pozwalający zidentyfikować przyczynę przerwania
U3FCR	0xE007 C008	zapis	rejestr sterujący kolejką FIFO
U3LCR	0xE007 C00C	zapis odczyt	rejestr sterujący trybem transmisji
U3LSR	0xE007 C014	odczyt	rejestr statusu odbiornika
U3SCR	0xE007 C01C	zapis odczyt	rejestr bez przydzielonej funkcji (w układach 16C550 był dopełnieniem do ośmiu komórek w przestrzeni adresowej)
U3ACR	0xE007 C020	zapis odczyt	
U3FDR	0xE007 C028	zapis odczyt	rejestr zawierający dodatkowe współczynniki pozwalające doprecyzować prędkość transmisji
U3TER	0xE007 C030	zapis odczyt	rejestr, poprzez który można włączać/wyłączać UART0

Wybrane rejestry (zaznaczone kolorem) mają tą samą alokację w przestrzeni adresowej. Dostęp do poszczególnych rejestrów jest rozróżniany na podstawie typu operacji (zapis/odczyt) oraz wskaźnika DLAB. Przejście do tego trybu następuje po ustawieniu bitu b7 w rejestrze U0LCR (zapisaniu do wymienionego rejestru takiej sekwencji bitowej, gdzie ustawiony jest odpowiedni bit).

Do obsługi zespołu UART3 utworzony jest moduł UART3.C (oraz UART3.H).

To oprogramowanie udostępnia następujące funkcje:

- **void UART3InitEnvir (void)** – bezparametrowa funkcja, której zadaniem jest skonfigurowanie pinów portu do potrzeb transmisji szeregowej (wyprowadzenia P2.25 → TXD3 oraz P2.26 → RXD3 stają się wyprowadzeniami zespołu UART3),
- **void UART3HardwInit (ULONG PeriphClk , UCHAR SerialMode , ULONG SerialSpeed)** – funkcja do zainicjowania trybu i prędkości pracy zespołu UART3, parametr *PeriphClk* jest częstotliwością sygnału zegarowego taktującego mikrokontroler wynikającą sumarycznie z częstotliwości rezonatora kwarcowego i ewentualnego układu PLL zwielokrotniającego tą częstotliwość, parametr *SerialMode* określa tryb pracy kontrolera transmisji szeregowej (jest wartością wprost zapisywaną do rejestru trybu), oraz *SerialSpeed* określa prędkość transmisji szeregowej wyrażoną w bajtach na sekundę (bps),
- **void UART3InputReset (void)** – funkcja zerująca programową kolejkę FIFO związaną z odebranymi w przerwaniach znakami,
- **void UART3Send (UCHAR Data)** – funkcja wysyłająca w kanał szeregowy podany w parametrach znak,
- **void UART3SendString (UCHAR * String)** – funkcja wysyłająca w kanał szeregowy ciąg znaków o dowolnej długości (ciąg musi być zakończony znakiem o kodzie 0),
- **ULONG UART3DataPresent (void)** – funkcja zwracająca informację, czy w kolejce FIFO odbiornika są dane do odczytania,
- **UCHAR UART3GetData (void)** – funkcja odczytująca z programowej kolejki FIFO kolejny znak,
- **UCHAR UART3Working (void)** – funkcja zwracająca status czy UART3 jest w danej chwili zajęty (zajmuje się wysyłaniem danych).

Odbierane przez UART3 dane są gromadzone w programowej kolejce FIFO związanej z odbiornikiem. Każde przerwanie wygenerowane przez odbiornik sprowadza się do odczytania z rejestrów UART3 kodu odebranego znaku oraz dodania jego do wspomnianej kolejki FIFO. Nadawanie znaków prowadzi do realizacji jednego z dwóch wariantów: jeżeli układ UART3 nie jest zajęty, to nadawany znak trafia bezpośrednio w rejestry nadajnika UART3. W przeciwnym wypadku (drugi wariant), kod znaku dodany jest do programowej kolejki FIFO związanej z nadawaniem znaków. Przerwanie wygenerowane przez nadajnik UART3 ewentualnie pobiera z tej kolejki kolejny znak do nadania aż do wyczerpania „zapasu” znaków zgromadzonych w kolejce FIFO.

Różnice (w stosunku do obsługi UART0 oraz UART2) są następujące (wyróżnione kolorem):

```
#include <LPC23xx.h>
#include "uart3.h"
#include "types.h"
#include "system.h"
#include "irqvectors.h"
```

```
#include "serialconst.h"

#define UART3InterruptMask ( 1 << UART3_Interrupt )

#define InpSerialBuffSize      64
#define OutSerialBuffSize      64

#define PCUART3                ( ULONG ) 1 << 25

typedef struct {
                    USHORT      RdDataPointer ;
                    USHORT      WrDataPointer ;
                    UCHAR       Buffer [ InpSerialBuffSize ] ;
                  } InpSerialBufferRecT ;

typedef struct {
                    USHORT      RdDataPointer ;
                    USHORT      WrDataPointer ;
                    USHORT      BusyStatus ;
                    UCHAR       Buffer [ OutSerialBuffSize ] ;
                  } OutSerialBufferRecT ;


static volatile InpSerialBufferRecT UART3InpBuffer ;
static volatile OutSerialBufferRecT UART3OutBuffer ;


__irq void UART3IrqHandler ( void )
{
  UCHAR Data ;
  UCHAR IrqStatus ;
  /*-----------------------------------------*/
  for ( ; ; )
  {
    IrqStatus = ( UCHAR ) U3IIR ;
    if ( IrqStatus & 0x01 )
    {
      VICVectAddr = 0 ;
      return ;
    } /* if */ ;
    switch ( ( IrqStatus >> 1 ) & 0x03 )
    {
      case 1              :   /* 0 1 - transmitter */
        if ( UART3OutBuffer . RdDataPointer == UART3OutBuffer . WrDataPointer )
        {
          UART3OutBuffer . BusyStatus = 0 ;
          break ;
        } /* if */ ;
        Data = UART3OutBuffer . Buffer [ UART3OutBuffer . RdDataPointer ] ;
        UART3OutBuffer . RdDataPointer ++ ;
        if ( UART3OutBuffer . RdDataPointer >= OutSerialBuffSize )
          UART3OutBuffer . RdDataPointer = 0 ;
        U3THR = Data ;
        break ;
      case 2              :   /* 1 0 - reciever data */
        Data = U3RBR ;
        UART3InpBuffer . Buffer [ UART3InpBuffer . WrDataPointer ] = Data ;
        UART3InpBuffer . WrDataPointer ++ ;
        if ( UART3InpBuffer . WrDataPointer >= InpSerialBuffSize )
          UART3InpBuffer . WrDataPointer = 0 ;
        break ;
      default             :
        VICVectAddr = 0 ;
        return ;
```

```
    } /* switch */ ;
  } /* for */ ;
} /* UART3IrqHandler */


void UART3InputReset ( void )
{
  UART3InpBuffer . RdDataPointer = 0 ;
  UART3InpBuffer . WrDataPointer = 0 ;
} /* UART3InputReset */


static void UART3OutputReset ( void )
{
  UART3OutBuffer . RdDataPointer = 0 ;
  UART3OutBuffer . WrDataPointer = 0 ;
  UART3OutBuffer . BusyStatus = 0 ;
} /* UART3OutputReset */


static void UART3DisableInterrupt ( void )
{
  VICIntEnClr = UART3InterruptMask ;
} /* UART3DisableInterrupt */


static void UART3EnableInterrupt ( void )
{
  VICIntEnable = UART3InterruptMask ;
} /* UART3EnableInterrupt */


void UART3Send ( UCHAR Data )
{
  USHORT WStop ;
  /*-------------------------------------------*/
  UART3DisableInterrupt ( ) ;
  if ( ! UART3OutBuffer . BusyStatus )
  {
    U3THR = Data ;
    UART3OutBuffer . BusyStatus = 1 ;
    UART3EnableInterrupt ( ) ;
  } /* if ... */
  else
  {
    for ( ; ; )
    {
      WStop = UART3OutBuffer . WrDataPointer + 1 ;
      if ( WStop >= OutSerialBuffSize )
        WStop = 0 ;
      if ( WStop != UART3OutBuffer . RdDataPointer )
      {
        UART3OutBuffer . Buffer [ UART3OutBuffer . WrDataPointer ] = Data ;
        UART3OutBuffer . WrDataPointer = WStop ;
        UART3EnableInterrupt ( ) ;
        break ;
      } /* if ... */
      UART3EnableInterrupt ( ) ;
      NOP ( ) ;
      NOP ( ) ;
      NOP ( ) ;
      NOP ( ) ;
      NOP ( ) ;
      UART3DisableInterrupt ( ) ;
```

```
    } /* for */ ;
  } /* if ... else */ ;
} /* UART3Send */


void UART3SendString ( UCHAR * String )
{
  while ( * String )
  {
    UART3Send ( * String ) ;
    String ++ ;
  } /* while */ ;
} /* UART3SendString */


ULONG UART3DataPresent ( void )
{
  ULONG Result ;
  /*-----------------------------------------*/
  Result = FALSE ;
  UART3DisableInterrupt ( ) ;
  if ( UART3InpBuffer . RdDataPointer != UART3InpBuffer . WrDataPointer )
  {
    Result = TRUE ;
  } /* if */ ;
  UART3EnableInterrupt ( ) ;
  return ( Result ) ;
} /* UART3DataPresent */


UCHAR UART3GetData ( void )
{
  UCHAR Data ;
  /*-----------------------------------------*/
  UART3DisableInterrupt ( ) ;
  if ( UART3InpBuffer . RdDataPointer == UART3InpBuffer . WrDataPointer )
  {
    UART3EnableInterrupt ( ) ;
    return ( 0 ) ;
  } /* if */ ;
  Data = UART3InpBuffer . Buffer [ UART3InpBuffer . RdDataPointer ] ;
  UART3InpBuffer . RdDataPointer ++ ;
  if ( UART3InpBuffer . RdDataPointer >= InpSerialBuffSize )
    UART3InpBuffer . RdDataPointer = 0 ;
  UART3EnableInterrupt ( ) ;
  return ( Data ) ;
} /* UART3GetData */


UCHAR UART3Working ( void )
{
  UCHAR Data ;
  /*-------------------------*/
  if ( UART3OutBuffer . BusyStatus )
    return FALSE ;
  Data = U3LSR ;
  if ( Data  & 0x40 )
    return ( TRUE ) ;
  else
    return ( FALSE ) ;
} /* UART3Working */


void UART3HardwInit( ULONG PeriphClk ,
```

```
                    UCHAR SerialMode ,
                    ULONG SerialSpeed )
{
  ULONG Fdiv ;
  ULONG Fmod ;
  ULONG LDivider ;
  ULONG HDivider ;
  /*-----------------------------------------*/
  UART3DisableInterrupt ( ) ;
  UART3InputReset ( ) ;
  UART3OutputReset ( ) ;
  Fdiv = ( PeriphClk / 16 ) / SerialSpeed ;
  Fmod = ( PeriphClk / 16 ) % SerialSpeed ;
  if ( ( Fmod << 1 ) >= SerialSpeed )
    Fdiv ++ ;
  LDivider = Fdiv % 256 ;
  HDivider = Fdiv / 256 ;
  U3FDR = 0x10 ;
  U3LCR = 0x80 ;
  U3DLL = LDivider ;
  U3DLM = HDivider ;
  U3LCR = SerialMode ;
  U3FCR = 0x07 ;
  VICVectAddr29  = ( unsigned long ) UART3IrqHandler ;
  VICVectCntl29  = 15 ;
  UART3EnableInterrupt ( ) ;
  U3IER = RecvDataInterruptEnable | TransmDataInterruptEnable ;
} /* UART3HardwInit */


void UART3InitEnvir ( void )
{
  PCONP |= PCUART3 ;
  PINSEL1 |= 0x003C0000 ;  // P2.25 -> TXD3  P2.26 -> RXD3
  UART3InputReset ( ) ;
  UART3OutputReset ( ) ;
} /* UART3InitEnvir */
```

Program demostracyjny jest adaptacją programu pokazanego przy opisie obsługi UART0.

```
#include <LPC23xx.H>
#include "system.h"
#include "types.h"
#include "irqvectors.h"
#include "serialconst.h"
#include "uart3.h"

#define IRQCallCntLimit                 500
#define BufferSize                      64

static ULONG IRQCallCounter ;
static USHORT Index ;
static UCHAR Buffer [ BufferSize ] ;

static void SoftwareInit ( void )
{
  /*-------------------------------------------------------------------------*/
  IRQCallCounter = 0 ;
  Index = 0 ;
} /* SoftwareInit */
```

```
static __irq void T1_IRQHandler ( void )
{
  /*-------------------------------------------------------------------------*/
  IRQCallCounter ++ ;
  if ( IRQCallCounter >= IRQCallCntLimit )
  {
    IRQCallCounter = 0 ;
  } /* if */ ;
  T1IR = 1 ;
  VICVectAddr = 0 ;
} /* T1_IRQHandler */


static void HardwareInit ( void )
{
  /*-------------------------------------------------------------------------*/
  PINSEL10 = 0 ;
  T1MR0 = ( PCLK / 1000 ) - 1 ;
  T1MCR = 3 ;
  T1TCR = 1 ;
  VICVectAddr5 = ( unsigned long ) T1_IRQHandler ;
  VICVectCntl5 = 15 ;
  VICIntEnable = ( 1 << TIMER1_Interrupt ) ;
  UART3InitEnvir ( ) ;
} /* HardwareInit */


UCHAR Hello1 [ ]      = "\r\nWitaj koles przy kompie\r\n\r\nNapisz cos\r\n" ;
UCHAR Hello2 [ ]      = "\r\nNapisales: " ;


static void HelloMessage ( void )
{
  /*-------------------------------------------------------------------------*/
  UART3SendString ( Hello1 ) ;
} /* HelloMessage */


void ProcessChar ( UCHAR Ch )
{
  USHORT Loop ;
  /*-------------------------------------------------------------------------*/
  if ( Ch < ' ' )
  {
    if ( Index )
    {
      for ( Loop = 0 ; Loop < Index ; Loop ++ )
        UART3Send ( Buffer [ Loop ] ) ;
      UART3Send ( 0x0D ) ;
      UART3Send ( 0x0A ) ;
    } /* if */ ;
    Index = 0 ;
  } /* if ... */
  else
  {
    if ( Index < BufferSize )
    {
      Buffer [ Index ] = Ch ;
    } /* IF */ ;
    Index ++ ;
  } /* if ... else */ ;
} /* ProcessChar */
```

```
int main ( void )
{
  /*-------------------------------------------------------------------------*/
  SysInit ( ) ;
  HardwareInit ( )  ;
  SoftwareInit ( ) ;
  UART3HardwInit( FOSC , N81Mode , 9600 ) ;
  HelloMessage ( ) ;
  for ( ; ; )
  {
    if ( UART3DataPresent ( ) )
      ProcessChar ( UART3GetData ( ) ) ;
  } /* loop */ ;
} /* main */
```

Program został również sprawdzony w działaniu (w oparciu o inny haedware).

Indeks ilustracji

Zalecana dodatkowa lektura:

1. *„Bufory cykliczne i ich zastosowanie do organizacji kolejek typu FIFO"*
2. *„Posługiwanie się środowiskiem µVision (Keil)"*
3. *„Posługiwanie się programem FLASHMAGIC"*