Na prawach rękopisu

INSTYTUT CYBERNETYKI TECHNICZNEJ
POLITECHNIKI WROCŁAWSKIEJ
Raport serii SPR nr 19/2002

Komunikacja radiowa z dwukołowym robotem mobilnym

Marek Wnuk

Słowa kluczowe: komunikacja radiowa, Modbus, robot mobilny, sterownik, oprogramowanie

Wrocław 2002

Spis treści

Spis rysunków

Spis tablic

1 Układ do komunikacji radiowej z robotem

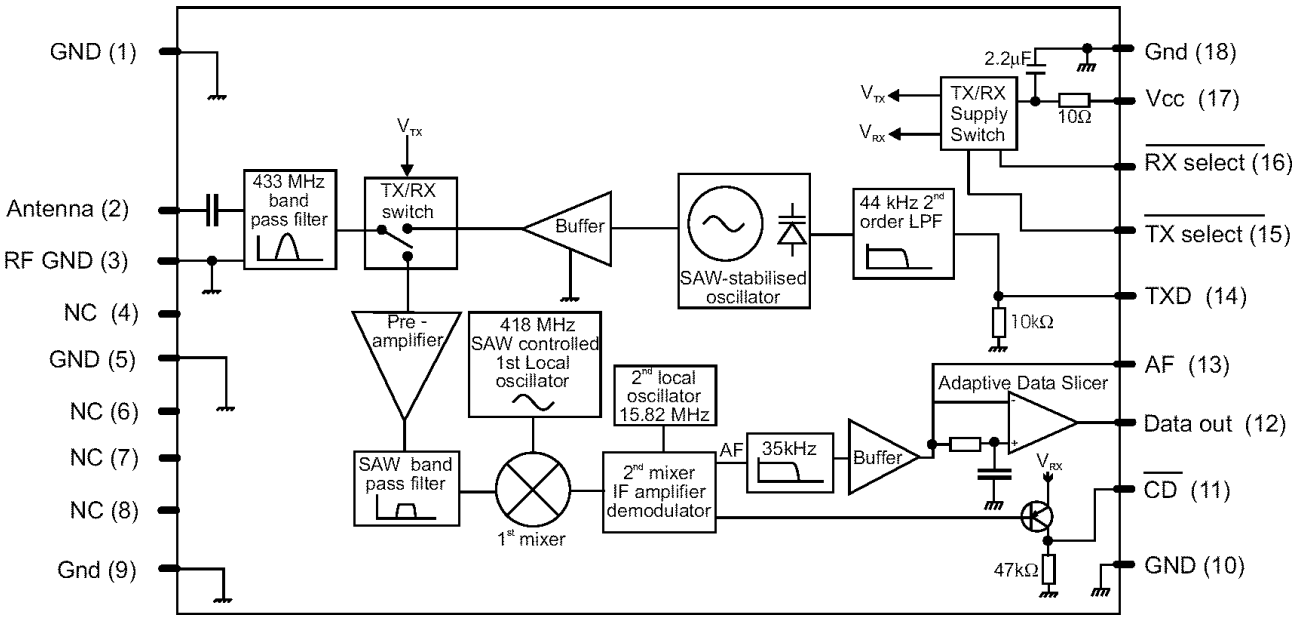
W laboratorium robotyki Instytutu Cybernetyki Technicznej skonstruowano stanowisko do badania algorytmów sterowania obiektem nieholonomicznym. Obiektem tym jest dwukołowy robot mobilny z napędem odniesionym do korpusu stanowiącego wahadło zawieszone na osi kół [1], [2], [3].
W celu umożliwienia w pełni autonomicznej pracy robota zaprojektowano i wykonano dwa moduły zawierające radiowe układy nadawczo-odbiorcze i układy logiczne umożliwiające połączenie ich z komputerem nadrzędnym i sterownikiem robota. Zaproponowano również protokół komunikacyjny (MODBUS) i dokonano jego modyfikacji, niezbędnych przy wykorzystywaniu transmisji radiowej.
Do budowy układu wybrano *tranceiver* BiM2-433-64S firmy Radiometrix [4], a prototypy zmontowano na płytkach uniwersalnych.

1.1 Moduł nadawczo–odbiorczy (*transceiver*) BiM2

BiM2 jest półdupleksowym modułem nadawczo–odbiorczym przeznaczonym do szybkiej dwukierunkowej transmisji danych na niewielkie odległości.
Podstawowe własności:

- certyfikat CE,
- zgodność ze standardem radiowym: ETSI EN 300-220-3,
- zgodność ze standardem EMC: ETSI EN 301-489-3,
- użyteczny zasięg: do 200m (w budynkach do 50m),
- prędkość transmisji danych: do 64kb/s,
- nadajnik FM o mocy 10mW,
- odbiornik superheterodynowy FM z podwójnym przetwarzaniem,
- filtracja sygnału wejściowego i pełne ekranowanie,
- zasilanie: 5V/20mA,
- wymiary: 23x33x4mm.

Schemat blokowy BiM2 przedstawiono na rys. 1.
Linia $\overline{TXselect}$ włącza nadajnik (pełna moc 10mW jest osiągana po 100μs). Sygnał modulujący jest doprowadzany do wejścia TXD (można stosować modulację cyfrową lub analogową). Wyjście stopnia mocy nadajnika jest podłączane do wyprowadzenia antenowego przez szybki przełącznik i filtr zapewniający zgodność z normą EN 300-220-3.
Część odbiorcza BiM2 pracuje jako superheterodyna FM z podwójnym przetwarzaniem (16MHz i 150kHz). Odbiornik jest aktywowany linią $\overline{RXselect}$ z opóźnieniem mniejszym od 1ms. Adaptacyjny dyskryminator danych zapewnia dyskretyzację sygnału odbieranego na poziomie jego średniej wartości (ze stałą czasową 1,2ms, a w wersji BiM2-433-64S – 150μs). Wyjście sygnału binarnego (*Data out*) jest odtwarzane poprawnie tylko w przypadku przebiegów o wypełnieniu zbliżonym do 50% (np. kodowanie typu Manchester lub bi-fazowe). Dostępne jest również bezpośrednie wyjście sygnału analogowego (AF). Obecność fali nośnej jest sygnalizowana na wyjściu $\overline{CD}$.
Producent zaleca stosowanie ćwierćfalowej anteny prętowej (o długości 155mm dla 433MHz).



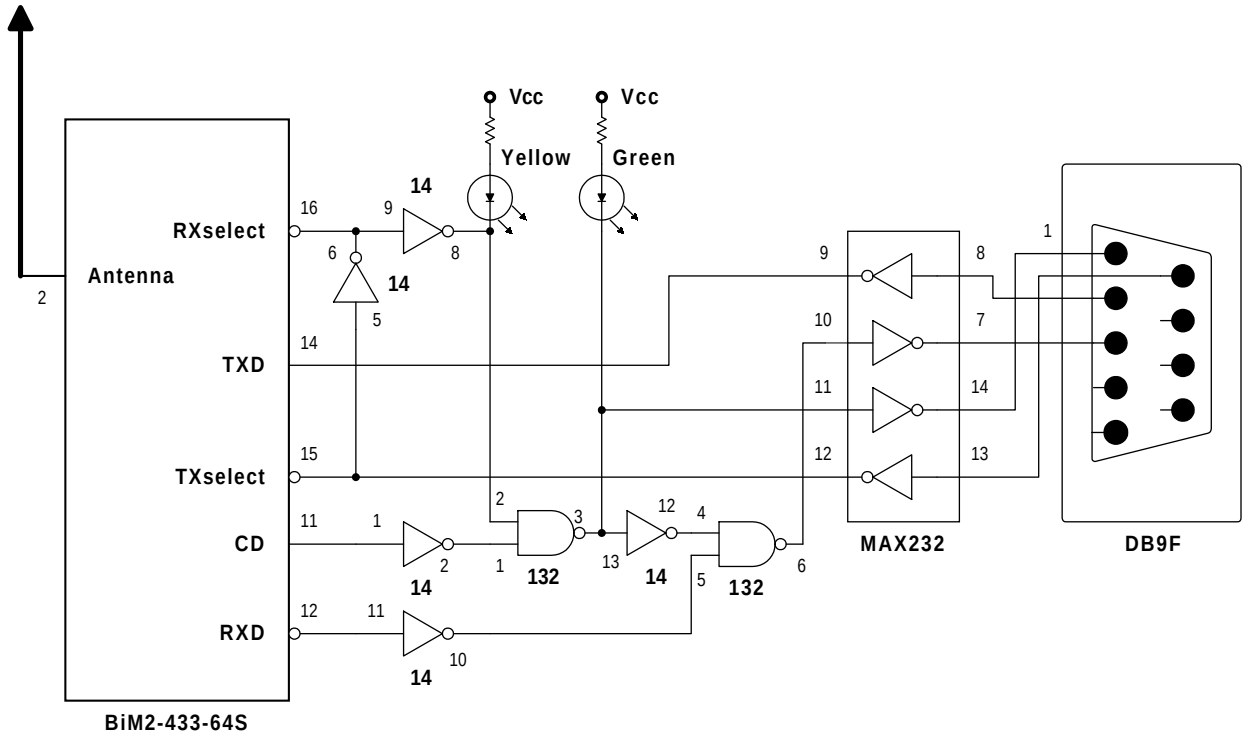
Rysunek 1: Schemat blokowy modułu radiowego BiM2 firmy Radiometrix

1.2 Konstrukcja modułu transmisji dla robota mobilnego

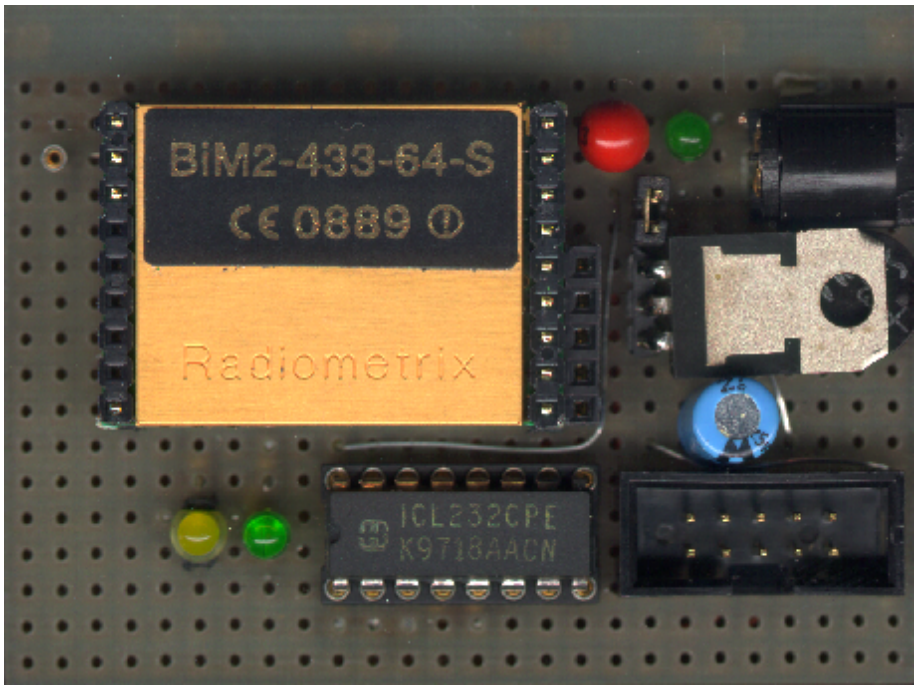
W zaprojektowanym, wykonanym i przetestowanym układzie wykorzystano opisany wcześniej moduł BiM2-433-64S. Ze względu na to, że zachodzi potrzeba użycia pary modułów radiowych, z których jeden współpracuje z komputerem klasy PC, a drugi – ze sterownikiem robota [2], układ opracowano tak, by był uniwersalny. Złącze transmisji szeregowej PC (np. port COM2) pracuje z sygnałami w standardzie RS232C, co wymaga zastosowania po stronie modułu radiowego odpowiedniego konwertera poziomów elektrycznych. Współpraca se sterownikiem robota odbywa się na poziomie standardowej logiki TTL, co pozwala na bezpośrednie połączenie. Układ konwersji TTL–RS232C (MAX232) został zamontowany na podstawce, która równocześnie może pełnić rolę gniazda połączeniowego dla portu szeregowego sterownika.
W przypadku PC do wysterowania sygnałów $\overline{TXselect}$ i $\overline{RXselect}$ wykorzystano sygnał *RTS* portu szeregowego, a sygnał $\overline{CD}$ doprowadzono na wejście *DCD*. Zapewnia to naturalną obsługę sygnałów portu szeregowego w PC, zgodna z większością protokołów.
W przypadku sterownika robota do wysterowania sygnałów $\overline{TXselect}$ i $\overline{RXselect}$ wykorzystano wyjście PE.5, a sygnał $\overline{CD}$ doprowadzono na wejście PF.1 mikrokontrolera MC68332.
Układ logiczny, zbudowany z bramek serii 74HCT, zapewnia właściwe działanie modułu przy wykorzystaniu dwóch, opisanych wyżej, sygnałów:

- naprzemienne włączanie nadajnika i odbiornika,
- blokowanie sygnału $\overline{CD}$ przy włączonym nadajniku,
- blokowanie danych odbieranych przy braku fali nośnej z odbiornika.

Dodatkowo, włączenie nadajnika ($\overline{TXselect}$) jest sygnalizowane diodą czerwoną, a pojawienie się fali nośnej z odbiornika – diodą zieloną.
Schemat ideowy układu przedstawiono na rys. 2. Ukąldy zmontowano na płytce uniwersalnej (rys. 3).



Rysunek 2: Schemat ideowy układu transmisyjnego



Rysunek 3: Układ zmontowany na płytce uniwersalnej

W celu zapewnienia autonomicznej pracy układu współpracującego z PC, wyposażono go w zasilacz stabilizowany zasilany z zewnętrznego transformatora zamontowanego we wtyku (tzw. *wall-transformer*).

1.3 Połączenie układu ze sterownikiem robota

W sterowniku dwukołowego robota mobilnego [2]do komunikacji z otoczeniem przeznaczono układ szeregowej transmisji asynchronicznej (SCI - *Serial Communicztion Interface* [6]), dostępny w 68332. Pozwala on przesyłać dane i komendy pomiędzy sterownikiem autonomicznego robota a komputerem nadrzędnym.

Tablica 1: Sygnały złącza komunikacyjnego sterownika robota

DB15	sygnał	MAX232	przeznaczenie
1	$+5V$		zasilanie interfejsu i joystick-a
2			
3	$U\,joyX$		joystick, oś x
4	GND		masa
5	GND	15	masa
6	$U\,joyY$		joystick, oś y
7			
8			
9	$+5V$	16	zasilanie interfejsu i joystick-a
10			
11			
12	TxD	9	dane nadawane SCI
13	$PE.5$	12	wyjście do włączania nadawania (RTS)
14	$PF.1$	11	wejście do detekcji fali nośnej (CD)
15	RxD	10	dane odbierane SCI

Sygnały danych nadawanych (TxD) i odbieranych (RxD) oraz weejście ($PF.1$) i wyjście ($PE.5$) wyprowadzono na złącze komunikacyjne Z2 typu DB15F (tablica 1) bez buforowania. Zasilanie odpowiednich interfejsów jest dostarczane przez złącze Z2. Sygnały logiczne i zasilające są doprowadzone ze złącza sterownika robota przez wtyk włączony w jeden rząd postawki układu MAX232.

2 Protokół komunikacyjny MODBUS (*Modicon*)

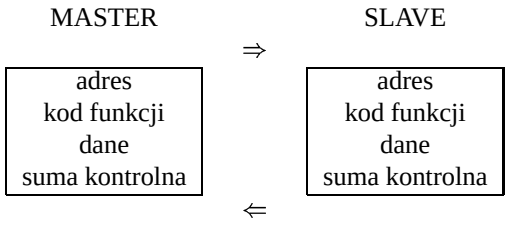
Protokół MODBUS [5] jest rozpowszechnionym standardem komunikacji szeregowej w systemach ontrolno–pomiarowych. Ze względu na liczne zalety:

- dostęp do nośnika typu *master – slave*
- wykrywanie i sygnalizacja błędów
- potwierdzanie wykonania komend
- zabezpieczenie przed blokadą
- możliwość transmisji znakowej RS232C, RS485 itp.

został on również zastosowany w sterowniku dwukołowego robota mobilnego [2].

2.1 Komunikacja w protokole MODBUS

Komunikacja w protokole MODBUS [5] odbywa się w trybie *master – slave*. Transakcję inicjuje *master* przez wysłanie ramki żądania zawierającej adres, kod funkcji, dane i sumę kontrolną (rys. 4).



Rysunek 4: Transakcja w protokole MODBUS

Jednostka podporządkowana (*slave*) odsyła ramkę odpowiedzi (skonstruowaną podobnie). Czas oczekiwania na odpowiedź jest kontrolowany i w przypadku przekroczenia dopuszczalnej wartości (zależnej od trybu protokołu) następuje przeterminowanie transakcji. Master może w tej sytuacji powtórzyć pytanie lub zasygnalizować błąd (*timeout*).

2.2 Ramki protokołu MODBUS

Ramki mają budowę zależną od trybu protokołu. W trybie znakowym (ASCII) przez łącze szeregowe są przesyłane znaki ASCII. Adres, kod funkcji, dane i suma kontrolna są przesyłane w postaci szesnastkowej. Znacznikiem początku ramki jest znak dwukropka, a koniec jest sygnalizowany znakami końca linii.
W trybie binarnym (RTU) przesyłane są bajty ośmiobitowe. Adres, kod funkcji, dane i cykliczna suma kontrolna (CRC) są przesyłane binarnie. znacznikiem początku i końca ramki jest odstęp czasowy (cisza na linii transmisyjnej o odpowiednim czasie trwania).

Ramka w trybie znakowym (ASCII)

:	adres		kod f.		dane			suma		CR	LF
						...					

- bajty są wysyłane szesnastkowo (po dwa znaki ASCII)
- odstępy pomiędzy kolejnymi znakami ramki $< 1s$

Ramka w trybie binarnym (RTU)

adres	kod f.	dane			suma	
			...			

- bajty są wysyłane binarnie jako znaki ośmiobitowe
- każda ramka jest poprzedzona odstępem (cisza na linii) $> 3.5T$ (gdzie T oznacza czas transmisji jednego znaku)
- odstępy pomiędzy kolejnymi znakami ramki $< 1.5T$

2.3 Pola ramki MODBUS

adres

0	–	adres rozgłaszania (*broadcast*)
1 – 247	–	adres jednostki *slave*

kod funkcji

1	$01	odczyt wyjść bitowych
2	$02	odczyt wejść bitowych
3	**$03**	**odczyt *n* rejestrów**
4	$04	odczyt *n* rejestrów wejściowych
5	$05	zapis 1 bitu
6	$06	zapis 1 rejestru
7	$07	odczyt statusu
8	$08	test diagnostyczny
15	$0F	zapis *n* bitów
16	**$10**	**zapis *n* rejestrów**
17	**$11**	**identyfikacja urządzenia *slave***
128 – 255	$80–$FF	zarezerwowane na odpowiedzi błędne

rejestry i zmienne
Urządzenie jest widziane jako 16-bitowe rejestry W_n. Typy zmiennych umieszczanych w rejestrach:

bitowe	–	bity rejestrów $W_0 - W_{4095}$
2-bajtowe	–	całe rejestry W_n
4-bajtowe	–	sąsiednie rejestry $W_n : W_{n+1}$

zalecenie
W celu ułatwienia przesyłania danych przy pomocy ramek z funkcją "**odczyt (zapis) *n* rejestrów**" rejestry powinny zajmować spójny obszar adresowany od 0 do rej_{max}.

2.4 Przykłady ramek MODBUS

żądanie (*master*): odczyt 2 rejestrów od W_{30} do W_{31}

adres *slave*	kod fun.	dane				suma LRC
		RA_H	RA_L	RN_H	RN_L	
12	03	00	1E	00	02	CB

odpowiedź (*slave*): dane z rejestrów od W_{30} do W_{31}

adres *slave*	kod fun.	dane					suma LRC
		NB	W_{30H}	W_{30L}	W_{31H}	W_{31L}	
12	03	04	01	23	02	34	8D

odpowiedź (*slave*): błąd – niedozwolony adres danych

adres *slave*	kod fun.	dane kod	suma LRC
12	83	02	69

2.5 Kontrola poprawności transmisji

Wykrywanie błędów transmisji następuje dzięki kontroli parzystości poprzecznej (bit parzystości znaku) i wzdłużnej (LRC, CRC).
Wykrywanie i diagnozowanie błędów komunikacji następuje przez:

- odesłanie przez *slave* ramki z kodem błędu:

  01 – niedozwolona funkcja
  02 – niedozwolony zakres danych
  03 – niedozwolona wartość danej
  04 – uszkodzenie w przyłączonym urządzeniu
  05 – potwierdzenie pozytywne
  06 – brak gotowości, komunikat usunięty
  07 – potwierdzenie negatywne
  08 – błąd parzystości pamięci

- przekroczenie czasu oczekiwania na odpowiedź (*timeout* w jednostce *master*) – *slave* nie odsyła odpowiedzi przy błędach w ramce żądania

2.6 Przystosowanie protokołu do potrzeb komunikacji radiowej

Jednostką nadrzędną (*master*) jest komputer PC, a podporządkowaną (*slave*) – sterownik robota. Przyjęto komunikację asynchroniczną w trybie znakowym (ASCII). Zarówno *master* jak i *slave* obsługują minimalny podzbiór funkcji protokołu:

- 3 - Read N Registers,
- 16 - Write N Registers,
- 17 - Identify Slave.

Zaimplementowano wyłącznie rejestry 16-bitowe reprezentowane przez zmienne całkowite.
Komunikacja znakowa wykorzystuje niewielki podzbiór znaków kodu ASCII:

- dwukropek - ":" - znacznik początku ramki,
- cyfry szesnastkowe - "0 .. 9, A .. F" - w polach numerycznych ramki,
- znaki końca linii - $< CR >$ i $< LF >$ - znacznik końca ramki.

Ta własność protokołu MODBUS ASCII została wykorzystana w celu uniknięcia stosowania wspomnianego wcześniej modulatora/demodulatora (np. Manchester) zapewniającego wypełnienie sygnału modulującego nadajnik bliskie 50%. Zaproponowano tablicę konwersji znaków protokołu na ośmiobitowe kody binarne. Mają one tę własność, że w obszarze jednego znaku (uwzględniając bity startu i stopu) nie występują ciągi jednakowych bitów o długości przekraczającej 2 (tab. 2). Pozwala to zachować warunki poprawnej pracy adaptacyjnego dyskryminatora danych po stronie odbiorczej BiM2.
W celu zapewnienia synchronizacji odbiornika ze strumieniem bitów odbieranych, każda ramka jest poprzedzana preambułą zawierającą kody spoza protokołu (`0x55`) - ignorowane przez odbiornik ramek i zapewniające synchronizację bitową), oraz znacznik synchronizacji bajtów (`0xff`) - pozwalający jednoznacznie ustalić płożenie bitu startu). Ze względu na konieczność transmitowania

Tablica 2: Tablica kodowa do transmisji radiowej w protokole MODBUS

znak	kod binarny	kod szesnastkowy	zastosowanie w protokole
:	01101010	0x6a	znacznik początku ramki
0	00110011	0x33	cyfra ”0”
1	00110101	0x35	cyfra ”1”
2	00110110	0x36	cyfra ”2”
3	01010011	0x53	cyfra ”3”
4	01010110	0x56	cyfra ”4”
5	01011001	0x59	cyfra ”5”
6	01011010	0x5a	cyfra ”6”
7	01100101	0x65	cyfra ”7”
8	01100110	0x66	cyfra ”8”
9	01101001	0x69	cyfra ”9”
A	10010011	0x93	cyfra ”A”
B	10010101	0x95	cyfra ”B”
C	10010110	0x96	cyfra ”C”
D	10011001	0x99	cyfra ”D”
E	10011010	0x9a	cyfra ”E”
F	10100101	0xa5	cyfra ”F”
$< CR >$	10101001	0xa9	1-szy znak końca ramki
$< LF >$	10100110	0xa6	1-szy znak końca ramki
U	01010101	0x55	ignorowany (synchronizacja bitowa)

kodów ośmiobitowych odstąpiono od kontrolowania parzystości poprzecznej (tryb portu szeregowego: 19200, 8N1).
Jednostka nadrzędna (PC) po wysłaniu ramki (poprzedzonej preambułą) wyłącza nadajnik przez deaktywację sygnału RTS na złączu portu szeregowego (co powoduje deaktywację $\overline{TXselect}$ i aktywację $\overline{RXselect}$ w module radiowym) i rozpoczyna oczekiwanie na odpowiedź.
Jednostka podrzędna (sterownik robota) oczekuje na ramki od PC przy wlączonym odbiorniku (PE.5 w stanie niskim: $\overline{RXselect}$ - aktywny, $\overline{TXselect}$ - nieaktyny). Po odebraniu poprawnej ramki adresowanej do siebie, *slave* ustawia stan wysoki na wyjściu PE.5, co powoduje wyłączenie nadajnika i włączenie odbiornika, a następnie wysyła ramkę odpowiedzi (również poprzedzoną preambułą).
Taki sposób komunikacji zmniejsza energię fali nośnej emitowanej przez nadajniki, ograniczając do niezbędnego minimum czas ich włączenia. Jest to również nie bez znaczenia dla poboru mocy z akumulatorów robota mobilnego.

3 Oprogramowanie

W dodatkach załączono oprogramowanie realizujące komunikację z wykorzystaniem zmodyfikowanego protokołu MODBUS, które zostało wykorzystane do zbadania poprawności pracy opracowanego systemu komunikacjo bezprzewodowej z robotem mobilnym. Zostało ono zrealizowane w języku C na dwóch platformach:

- MC68332 - strona sterownika robota (*slave*),
- Linux/PC - strona komputera nadrzędnego (*master*).

3.1 Strona sterownika (*slave*)

Koncepcja swobodnie programowalnego sterownika, która zapewnia użytkownikowi łatwe implementowanie własnych algorytmów sterowania w sterowniku autonomicznego robota mobilnego obejmuje obsługę komunikacji z otoczeniem w standardzie MODBUS.
Wykorzystywanie funkcji jądra do komunikacji nie wymaga od użytkownika znajomości technicznych szczegółów budowy sterownika. Użytkownik (eksperymentator), chcąc zapewnić komunikację z własnym algorytmem sterowania musi ustalić przyporządkowanie własnych zmiennych i parametŕow odpowiednim rejestrom MODBUS.
Jeżeli zachodzi potrzeba reagowania sterownika w sposób asynchroniczny (tzn. poza cyklem realizacji algorytmu sterowania) na zmianę zawartości rejestrów MODBUS, to do użytkownika należy napisanie procedury obsługującej takie zdarzenia. Procedura ta będzie wywoływana każdorazowo po wykonaniu obsługi funkcji **Write N Registers** (#16) z argumentami przekazującymi adres pierwszego z zapisywanych rejestrów i ilość zapisanych rejestrów. W ten sposób użytkownik może zapewnić np. obsługę komend wpisywanych jako umowne kody do ustalonego rejestru MODBUS.
Zamieszczone źródła zawierają:

- obsługę protokołu MODBUS włączoną do jądra sterownika (pliki *Modbus.h i Modbus.c*),
- przykładowy program użytkownika (plik *Radio_s.c*).

3.2 Strona komputera nadrzędnego (*master*)

Oprogramowanie po stronie komputera nadrzędnego zostało zrealizowane w celu przetestowania własności opracowanych modemów radiowych.
Składa się ono z dwóch części:

- obsługa protokołu MODBUS (pliki *modbus.h i modbus.c*),
- program testowy (plik *mod.c*).

Część użytkowa zawiera jedynie wizualizację wyników testu, pozwala jednak zapoznać się ze sposobem wywoływania procedur obsługi protokołu.
W pełni funkcjonalna jest natomiast część dotycząca obsługi protokołu MODBUS z zaimplementowanym kodowaniem znaków według tab. 2.

4 Uwagi końcowe

Opisane układy transmisji radiowej wraz ze zmodyfikowanym protokołem MODBUS pozwalają na komunikowanie się oprogramowania użytkowego zaimplementowanego na komputerze nadrzędnym (PC) ze sterownikiem robota mobilnego.
Próby przeprowadzono z użyciem ćwierćfalowych anteny prętowej o długości 155mm, w laboratorium robotyki Instytutu Cybernetyki Technicznej. Uzyskane wyniki są zadawalające. Stopa błędów (retransmisji) jest poniżej 0.002 przy odległości do 10m wewnątrz budynku. Dalszego zbadania wymaga zasięg transmisji przy zastosowaniu innych typów anten oraz na zewnątrz budynku.
Bezprzewodowa łączność na niewielkie odległości pozwala przekazywać parametry i rozkazy zadawania trajektorii ruchu i parametrów algorytmu sterowania oraz odbierać wyniki pomiarów dokonanych podczas eksperymentów. Niezbyt wielka prędkość transmisji (19.2kB) nie pozwala wprawdzie na bezpośrednie sterowanie robotem w ruchu przy pomocy algorytmu zaimplementowanego

na koputerze nadrzzędnym, ale nie jest to potrzebne, gdyż koncepcja swobodnie programowalnego sterownika [2] pozwala eksperymentatorowi implementować algorytmy sterowania bezpośredni w sterowniku robota i przeprowadzać badania zachowując autonomiczny charakter obiektu.

Bibliografia

[1] Kabała M., Tchoń K., Wnuk M., *Robot mobilny napędzany w układzie wewnętrznym*, VII KKR, Lądek Zdrój, 2001, Prace Naukowe ICT PWr., Konferencje 46, t.1, ss. 149-158.

[2] Kabała M., Wnuk M., *Dwukołowy robot mobilny napędzany w układzie wewnętrznym*, Raport SPR 21/2001, Inst. Cyb. Techn. PWr, 2001.

[3] Kabała M., Tchoń K., Wnuk M., *Dwukołowy nieholonomiczny robot mobilny*, Materiały Konferencji Automation 2002, Warszawa, marzec 2002, ss. 269-280.

[4] *BiM-2 User's Manual*, Radiometrix Inc., 2000.

[5] *Modicon Modbus Protocol Reference Guide*, PI-MBUS-300 Rev. J, Modicon Inc., 1996.

[6] *Queued Serial Module Reference Manual*, QSMRM/AD, Motorola Inc., 1991.

5 Dodatek A: Oprogramowanie sterownika

```
/*
 * Modbus.c
 * Implementacja minimalnej wersji
 * protokolu MODBUS dla 68332
 *
 * Tryb:        ASCII
 * Serial:      19200 8 n 1
 * Funkcje:     03, 16, 17
 * Rejestry:    16-bit
 *
 * MW 2000
 *
 * Dla Radiometrix BiM2-433-64-S
 * tablice konwersji mw2asc i asc2mw
 * (c) MW 2002
 */

#include śim.h"
#include "qsm.h"

#define _DEFINE_MW_
#include "modbus.h"
#undef  _DEFINE_MW_

#define Baud        19200
#define SCI_VECT    112
#define SCI_LEVEL   6
#define CLOCK       16777216            /* czestotliwosc zegara CPU */
#define MAX_BUF     256

#define BitSet(b,n) (b)|=(1<<(n))
#define BitClr(b,n) (b)&=~(1<<(n))
#define BitTst(b,n) (b)&(1<<(n))
#define DirIn       BitSet(PORTE,5)     /* wylaczenie bufora nadajnika */
#define DirOut      BitClr(PORTE,5)     /* wlaczenie bufora nadajnika */
#define TxEnable    SCCR1 |= 0x0008     /* uruchomienie nadajnika */
#define TxDisable   SCCR1 &= 0xfff7     /* zatrzymanie nadajnika */
#define TxIEnable   SCCR1 |= 0x0080     /* dblokowanie przerwan Tx */
#define TxIDisable  SCCR1 &= 0xff7f     /* zablokowanie przerwan Tx */
#define RxIEnable   SCCR1 |= 0x0020     /* dblokowanie przerwan Rx */
#define RxIDisable  SCCR1 &= 0xffdf     /* zablokowanie przerwan Rx */
#define TxCIEnable  SCCR1 |= 0x0040     /* odblokowanie przerwan Tx */
#define TxCIDisable SCCR1 &= 0xffbf     /* zablokowanie przerwan Tx */
```

```
/* kody bledow MODBUS */

#define FuncErr         1               /* niedozwolona funkcja */
#define AddrErr         2               /* niedozwolony zakres danych */
#define DataErr         3               /* niedozwolona wartosc danych */
#define ExtErr          4               /* uszkodzenie w przylaczonym urzadzeniu */
#define AckErr          5               /* potwierdzenie pozytywne */
#define BusyErr         6               /* brak gotowosci - komunikat usuniety */
#define NackErr         7               /* potwierdzenie negatywne */
#define ParErr          8               /* blad parzystosci pamieci */


/* deklaracje stalych globalnych */

const char      hex[16]="0123456789ABCDEF";

const char      preamble[8]={0x55,0x55,0x55,0xc3,0x3c,0x55,0x00,0x55};

/* deklracje zmiennych globalnych */

int             MsgRdy;
BYTE            MbErr;
BYTE            MbID;
BYTE            addr;
BYTE            fc;
BYTE            LRC;
WORD            index;
WORD            number;
int             i;
int             cntP;
int             cnt;
int             cntB;
int             stat;
int             ch;
WORD            timeout;
BYTE            *TxPtr;
BYTE            TxBuf[MAX_BUF];

#define ScBr(x) (CLOCK/32 + ((x)/2))/(x)   /* dzielnik predkosci SCI */

/* deklaracje procedur obslugi */

void (*Action)();
void (*WriteSvc)();
void NoAction();
```

```
void NoSvc();
void Await();
void Adata();
void Afinal();
void Apre();
void Asend();
void Asend1();
void AsendLRCH();
void AsendLRCL();
void AsendCR();
void AsendLF();
void AsendEnd();
int  Getint();

/* deklaracje funkcji protokolu MODBUS */

void Read03 (WORD, WORD);
void Write16(WORD, WORD);
void Ident17();
void SendEnd(WORD, WORD);

void initmw2asc()
{
  int i;
  for(i=0;i<256;i++) mw2asc[i] = 0;
  for(i=0;i<128;i++)
    {
      if(asc2mw[i]) mw2asc[asc2mw[i]] = i;
    }
}

/* Obsluga SCI */
/* procedura obslugi przerwania */
/* #pragma TRAP_PROC powoduje zamiane RTS na RTE i zachowanie rejestrow */

#pragma TRAP_PROC
void SciSvc()
{
  stat = SCSR;
  if(0 == (ch = mw2asc[SCDR & 0xff])) {Action=&Await; return;}
  if(ch != 0x55) (*Action)();
}



/* Inicjalizacja SCI */
```

```
void SciBaud(int baudrate)
{
  SCCR0 = ScBr(baudrate);    /* dzielnik zegara SCI */
}

void SciMode()
{
  DirIn;
  TxCIDisable;
  RxIDisable;
  MsgRdy=0;
  timeout=0;
  SCCR1 = 0x000c;             /* 8N1, TxEn RxEn ASCII*/
  Action=&Await;
  RxIEnable;
}

void MbInit(BYTE slaveaddr, BYTE slaveid)
{
  int pom;

  /* Ustawienie wektora przerwania od SCI */

  *((void (**) ())(4*SCI_VECT)) = SciSvc;

  QMCR          = 0x0083;     /* SUPV, IARB=0x3 */

  SCCR1         = 0x000c;
  QILR          = SCI_LEVEL;  /* poziom przerwania */
  QIVR          = SCI_VECT;   /* wektor przerwania */

  DirIn;                      /* wylaczenie bufora nadajnika RS232 */
  pom           = SCSR;
  pom           += SCDR;
  Action        = &NoAction;
  WriteSvc      = &NoSvc;
  ExtFail       = 0;
  addr          = slaveaddr;
  MbID          = slaveid;
  RunStat       = 0xff;
  MsgRdy                = 0;
  SciBaud(Baud);
  SciMode();
}
```

```
void NoAction() {}

void NoSvc(WORD ind, WORD nbr) {}

/* Obsluga trybu MODBUS ASCII  */

void Await()
{
  ch &= 0x007f;
  if(stat & 0x0080) TxCIDisable;  // zblokuj przerwanie od nadajnika
  if((stat & 0x004f)==0x0040)     // odebrano znak
    {
      if(ch == ':')               // poczatek ramki
        {
          TxPtr=TxBuf;
          cntB=0;
          LRC=0;
          Action=&Adata;
        }
    }
}

void Adata()
{
  ch &= 0x007f;
  if(stat & 0x0080) TxCIDisable;  /* zblokuj przerwanie od nadajnika */
  if((stat & 0x004f)==0x0040)     /* odebrano znak */
    {
      if(ch == 0x0d)              /* znak konca ramki <CR> */
        {
          Action=&Afinal;
          return;
        }
      if(ch < '0' || ch > '9')
        if(ch < 'A' || ch>'F') {Action=&Await; return;}
      if(ch < 'A') ch-='0';
      else ch-=55;                /* 'A'-10 */
      if(cntB & 0x0001) {
        *TxPtr|=ch;
        LRC+=*TxPtr++;
      }
      else *TxPtr=ch<<4;
      cntB++;
      if(cntB > 2*MAX_BUF) Action=&Await;
    }
}
```

```
void Afinal()
{
  ch &= 0x007f;
  if(stat & 0x0080) TxCIDisable;  // zblokuj przerwanie od nadajnika
  if((stat & 0x004f)==0x0040)     // odebrano znak
    {
      if(ch == 0x0a) {            // drugi znak konca ramki <LF>
        if(LRC==0) {
          if(addr == TxBuf[0] || TxBuf[0]==0) {
            MsgRdy=1;
            Action=&Apre;
            cntP=8;
            RxIDisable;
            return;
          }
        }
      }
      Action=&Await;
    }
}

void Apre()
{
  if(MsgRdy) {TxCIDisable; return;}
  if(stat & 0x0080)               /* nadajnik? */
    {
      if(TxBuf[0]==0) {
        TxCIDisable;
        RxIEnable;
        Action = &Await;
      }
      else {
        DirOut;
        if(--cntP) SCDR = preamble[cntP];        /* wyslij */
        else Action = &Asend;
      }
    }
}

void Asend()
{
  if(stat & 0x0080)               /* nadajnik? */
    {
      TxPtr = TxBuf;
      cntB*=2;
```

```
      LRC = 0;
      SCDR = asc2mw[':'];         /* wyslij */
      Action = &Asend1;
    }
}

void Asend1()
{
  if(stat & 0x0080)               /* nadajnik? */
    {
      ch = *TxPtr;
      if(cntB & 0x0001) { LRC+=*TxPtr++; ch&=0x0f;}
      else ch>>=4;
      ch = hex[ch];
      SCDR = asc2mw[ch];          /* wyslij */
      cntB--;
      if(!cntB) Action = &AsendLRCH;
    }
}

void AsendLRCH()
{
  if(stat & 0x0080)               /* nadajnik? */
    {
      LRC=~LRC+1;
      ch = hex[LRC>>4];
      SCDR = asc2mw[ch];          /* wyslij */
      Action = &AsendLRCL;
    }
}

void AsendLRCL()
{
  if(stat & 0x0080)               /* nadajnik? */
    {
      ch = hex[LRC&0x0f];
      SCDR = asc2mw[ch];          /* wyslij */
      Action = &AsendCR;
    }
}

void AsendCR()
{
  if(stat & 0x0080)               /* nadajnik? */
    {
      SCDR = asc2mw[0x0d];        /* wyslij CR */
```

```
      Action = &AsendLF;
    }
}

void AsendLF()
{
  if(stat & 0x0080)              /* nadajnik? */
    {
      SCDR = asc2mw[0x0a];       /* wyslij LF */
      Action = &AsendEnd;
    }
}

void AsendEnd()
{
  if(stat & 0x0080)              /* nadajnik? */
    {
      TxCIDisable;
      RxIEnable;
      DirIn;
      Action = &Await;
    }
}

/* Funkcje obslugi bufora transmisji */

char Getchar()
{
  return TxBuf[cnt++];
}

int Getint()
{
  int xx;
  xx=TxBuf[cnt++]*256;
  xx+=TxBuf[cnt++];
  return xx;
}

void Setint(int w)
{
  char* wsk;
  wsk=(char*) &w;
  TxBuf[cntB++]=*wsk++;
  TxBuf[cntB++]=*wsk++;
}
```

```
void Setchar(char w)
{
  TxBuf[cntB++]=w;
}

void MbService()
{
  if(MsgRdy)  {
    fc = TxBuf[1];
    cnt=2;
    MbErr=0;
    if(fc == 0x03 ) {  // odczyt n-rejestrow
      cntB=3;
      index= Getint();
      number= Getint();
      Read03(index,number);
      TxBuf[2]=cntB-3;
    }
    else if(fc == 0x10) { // zapis rejestrow
      index= Getint();
      number= Getint();
      Write16(index,number);
      cntB=6;
    }
    else if(fc == 0x11 ) { // identyfikacja
      cntB=3;
      Ident17();
      TxBuf[2]=cntB-3;
    }
    else MbErr = FuncErr;

    if(ExtFail) MbErr = ExtErr;
    if(MbErr) {
      fc |= 0x80;
      TxBuf[1] = fc;
      TxBuf[2] = MbErr;
      cntB= 3;
    }
    MsgRdy=0;
    TxCIEnable;
    if( fc==16 && MbErr==0 ) (* WriteSvc)(index, number);
  }
}

void Read03(WORD ind, WORD num)
```

```
{
  int     i,k;
  int     wi;
  double  w;
  if(num == 0 || num > FC03_MAX) {MbErr=AddrErr; return;}
  if(ind>=INT_BASE && (ind+num)<=INT_MAX) {
    ind-=INT_BASE;
    for(i=ind; i<ind+num;i++) {
      Setint(MbRegs[i]);
    }
  }
  else MbErr=AddrErr;
}

void Write16(WORD ind, WORD num)
{
  int     k;
  k=Getchar();
  if(k != num*2) {MbErr=FuncErr; return;}
  if(ind>=INT_BASE && (ind+num)<=INT_MAX) {
    ind-=INT_BASE;
    for(i=ind;i<ind+num;i++) {
      MbRegs[i] = Getint();
    }
  }
  else MbErr=AddrErr;
}

void Ident17()
{
  Setchar(MbID);
  Setchar(RunStat);
}

void MbStart()
{
  Action = &Await;
}

void MbStop()
{
  Action = &Await;
}

/* koniec pliku modbus.c */
```

```
/*
 * Modbus.h
 * definicje dla protokolu MODBUS
 *
 * MW 2000
 *
 * Dla Radiometrix BiM2-433-64-S
 * tablice konwersji mw2asc i asc2mw
 * (c) MW 2002
 */

#ifndef WORD
#define WORD unsigned int
#endif

#ifndef BYTE
#define BYTE unsigned char
#endif

#ifndef _modbus_MW_
#define _modbus_MW_

/* definicje rejestrow */

#define INT_NBR     20      /* liczba rejestrow 16-bitowych */
#define INT_BASE    1000    /* adres bazowy rejestrow 16-bitowych */
#define INT_MAX     INT_BASE+INT_NBR    /* maks. adres rejestru */

#define FC03_MAX    10      /* maks. liczba odczytywanych rejestrow */
#define FC16_MAX    10      /* maks. liczba zapisywanych rejestrow */

#ifdef _DEFINE_MW_
#define EXTERN
const unsigned char asc2mw[128] = {0,0,0,0,0,0,0,0,0,0,0xa6,0,0,0xa9,0,0,
                                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

                                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                                   0x33,0x35,0x36,0x53,0x56,0x59,0x5a,0x65,
                                   0x66,0x69,0x6a,0,0,0,0,0,

                                   0,0x93,0x95,0x96,0x99,0x9a,0xa5,0,0,0,0,
                                   0,0,0,0,0,
                                   0,0,0,0,0,0,0x55,0,0,0,0,0,0,0,0,0,0,

                                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0  };
```

```
#else
#define EXTERN extern
extern const unsigned char asc2mw[];

#endif


EXTERN unsigned char mw2asc[256];

EXTERN int  RunStat;              /* flaga uruchomienia urzadzenia */
EXTERN int  ExtFail;              /* flaga uszkodzenia urzadzenia */
EXTERN int  MbRegs[INT_NBR];      /* rejestry MODBUS */
EXTERN void (*WriteSvc)();        /* obsluga komend wpisywanych do rejestrow */

EXTERN void MbInit(BYTE,BYTE);    /* inicjalizacja transmisji szeregowej */
EXTERN void MbService();          /* obsluga protokolu w petli glownej */
EXTERN void MbStart();            /* uruchomienie MODBUS */
EXTERN void MbStop();             /* zatrzymanie MODBUS */

#endif /*_modbus_MW_*/

/* koniec pliku modbus.h */
```

```
/*
 * radio_s.c
 * Program testowy minimalnej wersji
 * protokolu MODBUS dla 68332
 *
 * Tryb:        ASCII, slave #1
 * Serial:      19200 8 n 1
 * Funkcje:     03, 16, 17
 * Rejestry:    16-bit
 *
 * komunikacja przez Radiometrix BiM2-433-64-S
 *
 * (c) MW 2002
 */

#include "tpu.h"
#include "qsm.h"
#include śim.h"
#include "modbus.h"
#include "flash.h"

#ifndef byte
#define byte unsigned char
#endif

#define RBASE   0x800000

extern const byte tpumska;
extern const long EXCEPT;

void komendy(WORD ind, WORD cnt)
{
  if(ind == 1002) MbRegs[0] += 1;
}

int main()
{
  BYTE saddr, sid;

  int  i;
  const byte * tmaptr;
  byte * ramptr;
  volatile long Tblco,ii;
```

```
  /* Programowanie pamieci FlashROM */

  FlashProgFlag = 0;            /* zakaz programowania */
  ProgMode = FlashProg(FlashProgFlag);
  Tblco=EXCEPT;

  /* Emulacja maski A TPU w RAM */

  *((word *)0xfffb04) = RBASE>>8;
  tmaptr = (const byte *)(&tpumska);
  ramptr = (byte *)RBASE;
  for(i=0;i<2048;i++) *ramptr++ = *tmaptr++;
  ramptr = (byte *)(& tpumska);

  initmw2asc();

  saddr = 1;                    /* adres wezla */
  sid = 0;                      /* slave ID */

  MbInit(saddr, sid);

  asm { ORI  #0x0700,SR
        ANDI #0xfdff,SR
        }                       /* ustawienie poziomu przerwan na 5 */

  for(i=0;i<20;i++) MbRegs[i] = i+1;

  WriteSvc = &komendy;

  MbStart();

  while(1)
    {
      MbService();
      ExtFail = 0;
      MbRegs[6] += 1;
    }
}

/* koniec pliku radio_s.c */
```

6 Dodatek B: Oprogramowanie PC

```
/*
 * modbus.c
 *
 * Procedury do obslugi MODBUS-Master
 * Wersja minimalna:
 *
 * Tryb:        ASCII
 * Serial:      19200 8 n 1
 * Funkcje:     03, 16, 17
 * Rejestry:    16-bit
 *
 * (c) MW 2000-2002
 *-------------------------------------------------------------
 * 2000/05/23   pierwsza wersja                         mw
 * 2002/06/17   kodowanie MODBUS dla BiM2               mw
 *-------------------------------------------------------------
 */

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <signal.h>
#include <errno.h>
#include <string.h>
#include <asm/termios.h>

#define VERBF           0       /* 1 dla testow */
#define _DEFINE_MW_
#include "modbus.h"

#define BUFSIZE 100

int verb = VERBF;

int rerr = 0;
int werr = 0;
int ierr = 0;

FILE *fpt, *fpr;
const int rts = TIOCM_RTS;
```

```
void initmw2asc()
{
  int i;
  for(i=0;i<256;i++) mw2asc[i] = 0;
  for(i=0;i<128;i++)
    {
      if(asc2mw[i]) mw2asc[asc2mw[i]] = i;
    }
}

/* Inicjalizacja portu szeregowego do komunikacji */

int SetCom(int fd)
{
  struct termios tds;
  int  i;

  for (i=0; i < NCCS; i++) tds.c_cc[i] = 0;

  tds.c_iflag = (IGNBRK);
  tds.c_iflag &= ~(INPCK | ISTRIP | INLCR | IGNCR | ICRNL | IUCLC |
                   IXON | IXANY | IXOFF | IMAXBEL);
  tds.c_oflag =0;

  tds.c_cflag  = 0;
  tds.c_cflag |=  CREAD | CLOCAL | CS8;

  tds.c_lflag = NOFLSH;

  /*....... Ustawienie predkosci odbioru i nadawania. ........*/
  if (cfsetispeed(&tds,B19200)) { printf("*** Blad 2\n");  exit(2);}
  if (cfsetospeed(&tds,B19200)) { printf("*** Blad 3\n");  exit(3);}
  /*.......... Ustawienie trybu pracy .......................*/
  return tcsetattr(fd,TCSANOW,&tds);
}

/* nadawanie ramki MODBUS */

SendFrame(char *txb)
{
  char tc, rc;
  unsigned char rr;
  char *ptr = txb;
  int i, t0;
  int ilosc, ilosc_nad;
```

```
  ioctl(fd, TIOCMBIS, &rts);      // ustawienie linii RTS w porcie

  while(*ptr){
    *ptr=asc2mw[*ptr];
    ptr++;
  }

  write(fd,"\125\125\125\125\125\125\125\125\125\125\125\125\125\125\
\125\125\125\125\125\125\125\125\377\125",24);

  ptr=txb;
  t0 = time(NULL);
  ilosc_nad=strlen(txb);

  do {
    if (time(NULL) - t0 > 2) {
      fprintf(stderr,"*** Przekroczony dozwolony czas na zapis.\n");
      exit(5);
    }
    /*... Przesylanie tekstu ..............................*/
    if ((ilosc = write(fd,ptr,ilosc_nad)) == 0) continue;
    if (ilosc < 0) { fprintf(stderr,"*** Blad nadawania\n");  exit(4);}
    ilosc_nad -= ilosc;
    ptr += ilosc;
    t0 = time(NULL); /*.. Jesli cos nadane to ponownie odczytany czas */
  } while (ilosc_nad > 0);

  tcdrain (fd);
  ioctl(fd, TIOCMBIC, &rts);      // wyzerowanie linii RTS w porcie
}

/* Funkcje do obslugi MODBUS */

/* Funkcja 03 - read n-registers */

int ReadRegs(unsigned char  slave,
             unsigned short index,
             unsigned short count,
             unsigned short * buffer)
{
  char txbuf[30];
  char rxbuf[256];
  char tc, rc;
  unsigned char rr;
  char *ptr = txbuf;
  char *eptr, ebuf[80];
```

```
unsigned char sum = 3;
unsigned int slrx, fcrx, nbrx, sumx;

int err = 0, flag =0, fin = 0, ecnt=0;
int i, hibuf, lobuf, stat;
int t0, ilosc, ilosc_nad;

sum += slave;
sum += (index-1)%256 + (index-1)/256;
sum += count%256 + count/256;
sum = 256 - sum;

/* Zestawianie ramki dla funkcji 03 */
sprintf(txbuf, ":%02X03%04X%04X%02X\015\012", slave, index-1, count, sum);
if(verb) fprintf(stderr, "%s", txbuf);

for(i=0;i<256;i++) rxbuf[i]=0;

SendFrame(txbuf);

/* Odbieranie odpowiedzi */

ptr = rxbuf;
t0 = time(NULL);

while(!err && !fin)
  {
    if(1 == (stat = read(fd, &rr, 1)))
      {
        rc=mw2asc[rr];
        if(verb) fputc(rc, stderr);
        if(rc == ':') flag = 1;
        if(flag)
          {
            *ptr++ = rc;
            if(rc == 0xa) fin = 1;
            *ptr = 0;
          }
      }
    else
      {
        if(stat==0){
          if(time(NULL)-t0< TOUT_SEC) continue;
          else{
            err ++; rerr++;
            if(verb) fprintf(stderr, "\n!!! timeout: %d, %s \n", err, rxbuf);
```

```
        fprintf(stderr,"\nrerr #%d\n",rerr);

        return ERR_TOUT;
      }
    }
    else{
      err ++;rerr++;
      if(verb) fprintf(stderr, "\n!!! blad przy odpowiedzi na 03\n");
      fprintf(stderr,"\nrurr #%d\n",rerr);

      return ERR_UNKN;
    }
  }

if(fin)
  {
    fprintf(stderr,"\nfin #r:%d w:%d i:%d\n",rerr,werr,ierr);

    if(verb) fprintf(stderr, "%s", rxbuf);
    ptr = rxbuf;
    if(3 != sscanf(ptr, ":%02X%02X%02X", &slrx, &fcrx, &nbrx))
      return ERR_FORM;
    if(verb) fprintf(stderr, "\nslave: %d\nfunct: %d\nnumbr: %d\n",
                             slrx, fcrx, nbrx);
    sum = slrx + fcrx + nbrx;

    if(fcrx & 0x80)
      {
        if(1 != sscanf(rxbuf + 5, "%02X", &sumx))
          return ERR_FORM;
        if(0 != sum + sumx)
          return ERR_CSUM;
        if(slrx != slave) return ERR_SLVN;
        if(fcrx == 0x83)
          {
            ModErrno = nbrx;
            if(verb) fprintf(stderr, " \nerror: %d\n", nbrx);
            return ERR_MBUS;
          }
        else
          return ERR_FUNC;
      }

    if(slrx != slave) return ERR_SLVN;
    if(fcrx != 3) return ERR_FUNC;
```

```
          else
            {
              ptr = rxbuf + 7;
              for(i=0; i<nbrx/2; i++)
                {
                  ptr = rxbuf + 7 + 4*i;
                  if(1 != sscanf(ptr, "%02x", &hibuf)) return ERR_FORM;
                  sum += hibuf;
                  if(1 != sscanf(ptr+2, "%02x", &lobuf)) return ERR_FORM;
                  sum += lobuf;
                  buffer[i] = 256*hibuf+lobuf;
                }

              ptr = rxbuf + 7 + 2*nbrx;
              if(1 != sscanf(ptr, "%02x", &sumx)) return ERR_FORM;
              sum += sumx;

              if(verb) fprintf(stderr, "\nsuma : %02X (%02X)\n",
                                       sum%256, sumx);

              if(sum) return ERR_CSUM;
              if(verb)
                {
                  for(i=0; i<nbrx/2; i++)
                    {
                      fprintf(stderr, "(%d) = %04X\n", index+i, buffer[i]);
                    }
                }
              return 0;
            }
        }
    }
}

/* Funkcja 16 - write n-registers */

int WriteRegs(unsigned char  slave,
              unsigned short index,
              unsigned short count,
              unsigned short * buffer)
{
  char txbuf[256];
  char rxbuf[300];
  char tc, rc;
  unsigned char rr;
  char *ptr = txbuf;
```

```
unsigned char sum = 16;
unsigned int slrx, fcrx, nbrx, sumx, firstx, numberx;
char *eptr, ebuf[80];
int err = 0, flag =0, fin = 0, ecnt=0;
int i, hibuf, lobuf, stat, t0;

sum += slave;
sum += (index-1)%256 + (index-1)/256;
sum += count%256 + count/256;
sum += 2*count;

/* Zestawianie ramki dla funkcji 16 */
sprintf(txbuf, ":%02X10%04X%04X%02X", slave, index-1, count, 2*count);

for(i=0; i<count; i++)
  {
    ptr = txbuf + 15 + 4*i;
    sprintf(ptr, "%04X", buffer[i]);
    sum += buffer[i]%256 + buffer[i]/256;
  }
sprintf( txbuf + 15 + 4*count, "%02X\r\012", 256 - sum);

if(verb) fprintf(stderr, "%s", txbuf);

for(i=0;i<256;i++) rxbuf[i]=0;

SendFrame(txbuf);

/* Odbieranie odpowiedzi */

ptr = rxbuf;

t0 = time(NULL);

//  eptr=ebuf; ebuf[0]=0; ecnt = 0;

while(!err && !fin)
  {
    if(1 == (stat = read(fd, &rr, 1)))
      {
        rc=mw2asc[rr];
        if(verb) fputc(rc, stderr);
        if(rc == ':') flag = 1;
        if(flag)
          {
            *ptr++ = rc;
```

```
        if(rc == 0xa) fin = 1;
        *ptr = 0;
      }
  }
else
  {
    if(stat==0){
      if(time(NULL)-t0< TOUT_SEC) continue;
      else{
        err ++; werr++;
        if(verb) fprintf(stderr, "\n!!! timeout: %d, %s \n",
                                 err, rxbuf);

        fprintf(stderr,"\nwerr #%d\n",werr);

        return ERR_TOUT;
      }
    }
    else{
      err ++;werr++;
      if(verb) fprintf(stderr, "\n!!! blad przy odpowiedzi na 16\n");
      fprintf(stderr,"\neurr #%d\n",werr);

      return ERR_UNKN;
    }
  }

if(fin)
  {
    fprintf(stderr,"\nfin #r:%d w:%d i:%d\n",rerr,werr,ierr);

    if(verb) fprintf(stderr, "%s", rxbuf);
    ptr = rxbuf;
    if(2 != sscanf(ptr, ":%02X%02X", &slrx, &fcrx))
      return ERR_FORM;
    if(verb) fprintf(stderr, "\nslave: %d\nfunct: %d\n", slrx, fcrx);
    sum = slrx + fcrx;
    if(fcrx & 0x80)
      {
        if(2 != sscanf(rxbuf + 5, "%02X%02X", &nbrx, &sumx))
          return ERR_FORM;
        if(0 != sum + nbrx + sumx)
          return ERR_CSUM;
        if(slrx != slave) return ERR_SLVN;
        if(fcrx == 0x90)
          {
```

```
                ModErrno = nbrx;
                if(verb) fprintf(stderr, " \nerror: %d\n", nbrx);
                return ERR_MBUS;
              }
            else
              return ERR_FUNC;
          }

        if(3 != sscanf( rxbuf + 5, "%04X%04X%02X", &firstx, &numberx, &sumx))
          return ERR_FORM;
        if(verb) fprintf(stderr, "\nfirst: %d\nnumbr: %d\n",
                         firstx, numberx);
        sum += firstx%256 + firstx/256 + numberx%256 + numberx/256 + sumx;
        if(verb) fprintf(stderr, "\nsuma : %02X (%02X)\n", sum%256, sumx);

        if(0 != (sum%256))
          return ERR_CSUM;
        else return 0;
      }
  }
}

/* Funkcja 17 - identify */

int Identify(unsigned char slave,
             unsigned char * buffer)
{
  char txbuf[30];
  char rxbuf[256];
  char tc, rc;
  unsigned char rr;
  char *ptr = txbuf;
  unsigned char sum = 17;
  unsigned int slrx, fcrx, nbrx, sumx;

  int err = 0, flag =0, fin = 0, ecnt=0;
  char *eptr, ebuf[80];
  int i, hibuf, lobuf, stat, t0;

  sum += slave;
  sum = 256 - sum;

  /* Zestawianie ramki dla funkcji 17 */
  sprintf(txbuf, ":%02X11%02X\r\012", slave, sum);
  if(verb) fprintf(stderr, "%s", txbuf);
```

```
for(i=0;i<256;i++) rxbuf[i]=0;

SendFrame(txbuf);

/* Odbieranie odpowiedzi */

ptr = rxbuf;
t0 = time(NULL);

while(!err && !fin)
  {
    if(1 == (stat = read(fd, &rr, 1)))
      {
        rc=mw2asc[rr];
        if(verb) fputc(rc, stderr);
        if(rc == ':') flag = 1;
        if(flag)
          {
            *ptr++ = rc;
            if(rc == 0xa) fin = 1;
            *ptr = 0;
          }
      }
    else
      {
        if(stat==0){
          if(time(NULL)-t0< TOUT_SEC) continue;
          else{
            err ++; ierr++;
            if(verb) fprintf(stderr, "\n!!! timeout: %d, %s \n", err, rxbuf);

            fprintf(stderr,"\nierr #%d\n",ierr);

            return ERR_TOUT;
          }
        }
        else{
          err ++;ierr++;
          if(verb) fprintf(stderr, "\n!!! blad przy odpowiedzi na 17\n");
          fprintf(stderr,"\niurr #%d\n",ierr);

          return ERR_UNKN;
        }
      }

    if(fin)
```

```
{
  fprintf(stderr,"\nfin #r:%d w:%d i:%d\n",rerr,werr,ierr);

  if(verb) fprintf(stderr, "%s\n", rxbuf);
  ptr = rxbuf;
  if(3 != sscanf(ptr, ":%02X%02X%02X", &slrx, &fcrx, &nbrx))
    return ERR_FORM;
  if(verb) fprintf(stderr, "\nslave: %d\nfunct: %d\nnumbr: %d\n",
                           slrx, fcrx, nbrx);
  sum = slrx + fcrx + nbrx;

  if(fcrx & 0x80)
    {
      if(1 != sscanf(rxbuf + 5, "%02X", &sumx))
        return ERR_FORM;
      if(0 != sum + sumx)
        return ERR_CSUM;
      if(slrx != slave) return ERR_SLVN;
      if(fcrx == 0x91)
        {
          ModErrno = nbrx;
          if(verb) fprintf(stderr, " \nerror: %d\n", nbrx);
          return ERR_MBUS;
        }
      else
        return ERR_FUNC;
    }

  if(slrx != slave) return ERR_SLVN;
  if(fcrx != 17) return ERR_FUNC;
  else
    {
      buffer[0] = nbrx;
      for(i=0; i<nbrx; i++)
        {
          ptr = rxbuf + 7 + 2*i;
          if(1 != sscanf(ptr, "%02x", &buffer[i+1])) return ERR_FORM;
          sum += buffer[i+1];
        }

      ptr = rxbuf + 7 + 2*nbrx;
      if(1 != sscanf(ptr, "%02x", &sumx)) return ERR_FORM;
      sum += sumx;

      if(verb) fprintf(stderr, "\nsuma : %02X (%02X)\n",
                               sum%256, sumx);
```

```
              if(sum) return ERR_CSUM;
              if(verb)
                {
                  for(i=1; i<=nbrx; i++)
                    {
                      fprintf(stderr, " %02X ",  buffer[i]);
                    }
                  fprintf(stderr, "\n");
                }
              return 0;
            }
        }
    }
}

/* koniec pliku modbus.c */
```

```
/*
 * modbus.h
 *
 * Definicje do obslugi MODBUS-Master
 *
 * MW 2000
 *-------------------------------------------------------------
 * 2000/05/23   pierwsza wersja
 *-------------------------------------------------------------
 */

#ifndef _MODBUS_MW_
#define _MODBUS_MW_

#define TOUT_SEC        2

#define ERR_UNKN        -1      /* nieznany blad przy transmisji */
#define ERR_MBUS        -2      /* odpowiedz slave z sygnalizacja bledu */
#define ERR_FORM        -3      /* blad formatu ramki */
#define ERR_CSUM        -4      /* blad sumy kontrolnej odpowiedzi */
#define ERR_SLVN        -5      /* nieprawidlowy numer wezla */
#define ERR_TOUT        -6      /* przekroczenie czasu odpowiedzi */
#define ERR_FUNC        -7      /* nieprawidlowy kod funkcji */
#define ERR_PARM        -8      /* nieprawidlowe parametry odpowiedzi */

#ifdef  _DEFINE_MW_
#define EXTERN
const unsigned char asc2mw[128] = {0,0,0,0,0,0,0,0,0,0,0xa6,0,0,0xa9,0,0,
                                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

                                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                                   0x33,0x35,0x36,0x53,0x56,0x59,0x5a,
                                   0x65,0x66,0x69,0x6a,0,0,0,0,0,

                                   0,0x93,0x95,0x96,0x99,0x9a,0xa5,0,0,0,
                                   0,0,0,0,0,0,
                                   0,0,0,0,0,0x55,0,0,0,0,0,0,0,0,0,0,

                                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0  };
#else
#define EXTERN extern
        extern const unsigned char asc2mw[];
#endif
```

```
/* zmienne globalne */

EXTERN int ModErrno;    /* kod bledu odpowiedzi przy ERR_MBUS */
EXTERN unsigned char mw2asc[256];
EXTERN int fd;

/* Funkcje do obslugi MODBUS */

EXTERN void initmw2asc();

/* Inicjalizacja portu */

EXTERN int SetCom(int fd);
EXTERN int InitPort(char * port);

/* Funkcja 03 - read n-registers */

EXTERN int ReadRegs(unsigned char  slave,
                    unsigned short index,
                    unsigned short count,
                    unsigned short * buffer);

/* Funkcja 16 - write n-registers */

EXTERN int WriteRegs(unsigned char  slave,
                     unsigned short index,
                     unsigned short count,
                     unsigned short * buffer);

/* Funkcja 17 - identify slave */

EXTERN int Identify(unsigned char slave,
                    unsigned char * buffer);

#endif

/* koniec pliku modbus.h */
```

```
/*
 * mod.c
 * Przykladowy program do komunikacji ze sterownikiem
 * wyposazonym w oprogramowanie MODBUS-slave
 *
 * Korzysta z funkcji w modbus.c
 *
 * (c) MW 2000-2002
 *-------------------------------------------------------------
 * 2000/05/23   pierwsza wersja                          mw
 * 2002/06/17   kodowanie MODBUS dla BiM2                mw
 *-------------------------------------------------------------
 */

#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>

#include "modbus.h"

#define DEFPORT         "/dev/ttyS1"    /* COM2 */


int main(int argc, char *argv[]){

  unsigned char         slvaddr;
  unsigned short        first;
  unsigned short        number;
  unsigned short        value;
  unsigned short        recbuf[256];
  unsigned char         idbuf[256];
  int                   i, result;

  initmw2asc();

  fprintf(stderr, "\n*** Test komunikacji MODBUS:\n");

  /*....... Otwarcie COM2 do odczytu i zapisu bez blokowania .*/

  if ((fd = open("/dev/ttyS1",O_RDWR | O_NONBLOCK)) < 0)
    {
```

```
    switch (fd)
      {
      case EACCES: fprintf(stderr,"*** brak dostepu\n");break;
      case ENOENT: fprintf(stderr,"*** objekt nie istnieje \n");break;
      case ETXTBSY: fprintf(stderr,"*** zajete :) \n");
      default: fprintf(stderr,"Blad otwarcia portu, kod: %d\n",errno);
      }
    exit(1);
  };

SetCom(fd);
fprintf(stderr,"Port otwarty\n");

slvaddr = 1;
first   = 1001;
number  = 5;
value   = 0;
while(1)
  {
    if(0 != (result = Identify(slvaddr, idbuf)))
      {
        switch(result)
          {
          case ERR_TOUT:
            fprintf(stderr, "!ID TIMEOUT\n");
            break;
          case ERR_MBUS:
            fprintf(stderr, "!ID BLAD MODBUS: %d\n", ModErrno);
            break;
          default:
            fprintf(stderr, "!ID INNY BLAD: %d\n", result);
            break;
          }
      }
    else
      {
        for(i=1; i<=idbuf[0]; i++)
          {
            fprintf(stderr, " %02X ", idbuf[i]);
          }
        fprintf(stderr, "\n");
      }

    value ++;

    if(0 != (result = WriteRegs(slvaddr, 1003, 1, &value)))
```

```
        {
          switch(result)
            {
            case ERR_TOUT:
              fprintf(stderr, "!TX TIMEOUT\n");
              break;

            case ERR_MBUS:
              fprintf(stderr, "!TX BLAD MODBUS: %d\n", ModErrno);
              break;

            default:
              fprintf(stderr, "!TX INNY BLAD: %d\n", result);
              break;
            }
        }

      if(0 != (result = ReadRegs(slvaddr, first, number, recbuf)))
        {
          switch(result)
            {
            case ERR_TOUT:
              fprintf(stderr, "!RX TIMEOUT\n");
              break;

            case ERR_MBUS:
              fprintf(stderr, "!RX BLAD MODBUS: %d\n", ModErrno);
              break;

            default:
              fprintf(stderr, "!RX INNY BLAD: %d\n", result);
              break;
            }
        }
      else
        {
          for(i=0; i<number; i++)
            {
              fprintf(stderr, "(%d) = %04X\n", first+i, recbuf[i]);
            }
          fprintf(stderr, "\n");
        }
    }
}

/* koniec pliku mod.c */
```

dr inż. Marek Wnuk
Instytut Cybernetyki Technicznej
Politechniki Wrocławskiej
ul. Janiszewskiego 11/17
50-372 Wrocław

Niniejszy raport otrzymują:

1. OINT ..... - 1 egz.

2. Zleceniodawca ..... - 2 egz.

3. Autor ..... - 2 egz.

Razem : 5 egz.

Raport wpłynął do redakcji I-6
we wrześniu 2002 roku.