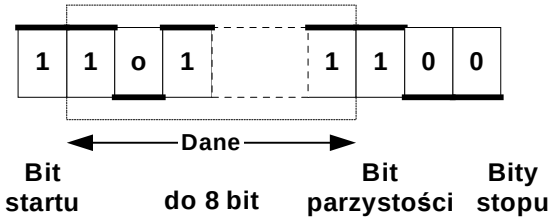
1 Obsługa interfejsu transmisji szeregowej i protokół MODBUS

1.1 Transmisja asynchroniczna – standard RS232C

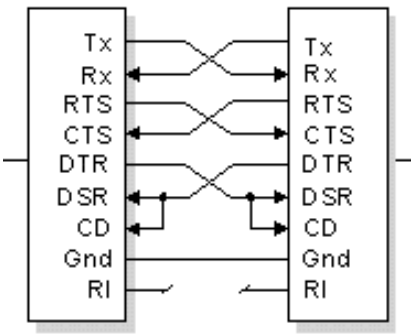


Rys. 1-1 Struktura znaku w transmisji szeregowej

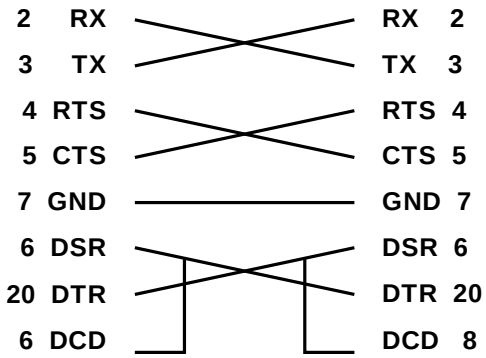
Oznaczenie	Typ	Opis	Opis angielski
RX	We	Dane odbierane	*Received data*
TX	Wy	Dane nadawane	*Transmitted data*
DTR	Wy	Urządzenie gotowe	*Data terminal ready*
DSR	We	Test czy urządzenie gotowe	*Data set ready*
RTS	Wy	Znaki mogą być nadawane	*Request to send*
CTS	We	Test czy znaki mogą być nadawane	*Clear to send*
DCD	Wy	Jest nośna modemu	*Data carrier dedected*
GND	-	Potencjał ziemi	*Ground*

Tab. 1-1 Linie interfejsu szeregowego

1.2 Łączenie komputera z urządzeniem pomiarowym

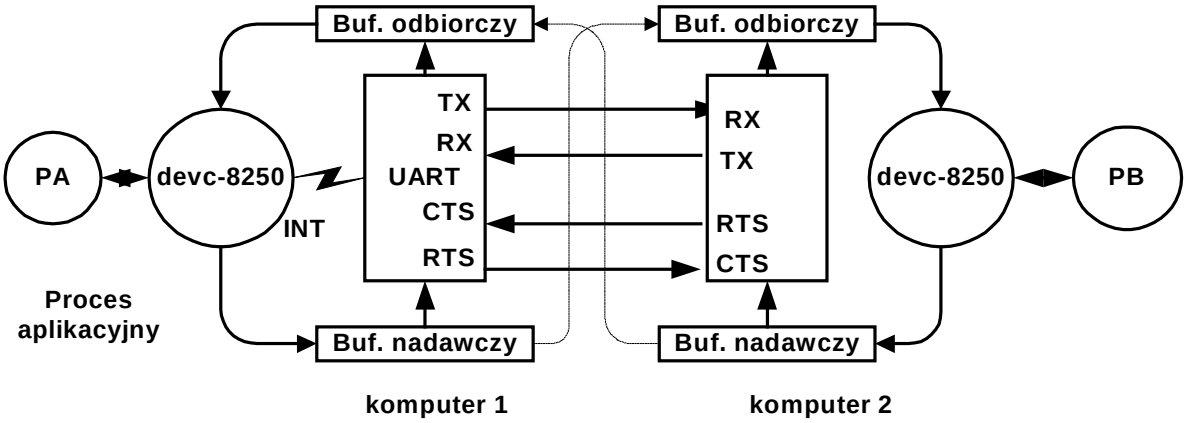


Rys. 1-2 Połączenie typu NULL Modem dla 2 urządzeń DCE



Rys. 1-3 Połączenie typu NULL MODEM dla złącza DB25

1.3 Struktura portu szeregowego i kontrola przepływu



Rys. 1-4 Współpraca procesów aplikacyjnych PA i PB komunikujących się przez port szeregowy

Kontrola przepływu
Wstrzymywanie nadawcy gdy odbiorca nie jest gotowy na przyjęcie wiadomości:

- Sprzętowa - (*ang. hardware flow control*)
- Programowa - (*ang. software flow control)*

Sprzętowa
Powstrzymanie nadawania - RTS -> LOW
Dozwolenie nadawania - RTS -> HIGH

Testowanie czy można nadawać:
Gdy CTS = HIGH – można nadawać
Gdy CTS = LOW – nie można nadawać

Programowa
Sygnalizacja:
Powstrzymanie nadawania - wysłanie XOFF
Dozwolenie nadawania - wysłanie XON

Testowanie czy można nadawać:
Gdy odebrano XOFF - nie można nadawać
Gdy odebrano XON - można nadawać

1.4 Konfigurowanie i użycie portu szeregowego

Ustalanie parametrów portu szeregowego z konsoli

Testowanie ustawienia portu szeregowego:

```
stty < /dev/ser1
```

Ustawianie parametrów portu szeregowego

```
stty [operandy] > /dev/ser1
```

Ważniejsze operandy:

Operand	Znaczenie	Wartości parametrów
baud	Szybkość transmisji	1 do 115200
par	Parzystość	o - nieparzystość e - parzystość n – brak bitu parzystości
bits	Liczba bitów w znaku	5,6,7,8
stopb	Liczba bitów stopu	1 lub 2

Przykład:

```
$ stty baud=2400 par=e  bits=8 stopb=1 > /dev/ser1
```

1.5 Podstawowe funkcje obsługi portu szeregowego

Otwarcie urządzenia – funkcja open

```
int open(char *path,int oflag,…)
```

path Scieżka dostępu do urządzenia /dev/ser1, /dev/ser2, ...
oflag Tryb dostępu do urządzenia (opis w pliku nagłówkowym <fcntl.h>)

Funkcja powoduje otwarcie o nazwie wyspecyfikowanej w parametrze path. Otwarcie następuje zgodnie z trybem oflag.

Funkcja zwraca:
> 0 – uchwyt do pliku (*ang. File handle*) – mała liczba typu int.
- 1 – gdy wystąpił błąd.

Odczyt z urządzenia – funkcja read

```
int read(int fdes, void *bufor, int nbytes)
```

fdes Uchwyt do pliku zwracany przez funkcję open
bufor Bufor w którym umieszczane są przeczytane bajty
nbytes Liczba bajtów którą chcemy przeczytać.

Funkcja powoduje odczyt z urządzenia identyfikowanego przez **fdes**, **nbytes** bajtów i umieszczenie ich w buforze.
Funkcja zwraca:
> 0 – liczbę rzeczywiście przeczytanych bajtów,
- 1 – gdy błąd.

Zapis do urządzenia – funkcja write

```
int write(int fdes, void *bufor, int nbytes)
```

fdes Uchwyt do pliku zwracany przez funkcję open
bufor Bufor w którym umieszczane są bajty przeznaczone do zapisu
nbytes Liczba bajtów którą chcemy zapisać

Funkcja powoduje zapis do urządzenia identyfikowanego przez fdes nbytes bajtów znajdujących buforze.
Funkcja zwraca:
> 0 – liczbę rzeczywiście zapisanych bajtów,
- 1 – gdy błąd.

Zamknięcie urządzenia – funkcja close

```
int close(int fdes)
```

fdes Uchwyt do pliku zwracany przez funkcję open

Funkcja powoduje zamknięcie urządzenia identyfikowanego przez **fdes**.

```
#include <stdlib.h>
#include <fcntl.h>

main(void) {
  char *outmsg="Ten tekst zostanie wyprowadzony na
                port 1";
  char c;
  int fd;
  fd=open("/dev/ser1", O_RDWR);
  /* Pisanie na port szeregowy */
  write(fd, outmsg, strlen(outmsg);
  /* Odczyt z portu szeregowego */
  do {
     read(fd, &c, 1);
     putch(c);
 } while(1);
 close(fd);
}
```

Przykład – prosty program zapisu i odczytu

1.6 Ustawianie parametrów terminala

Testowanie parametrów terminala

Parametry terminala zawarte są w strukturze termios zdefiniowanej w **<termios.h>.**

```
struct termios {
       tcflag_t  i_iflag;   // Tryb wejściowy
       tcflag_t  o_iflag;   // Tryb wyjściowy
       tcflag_t  c_iflag;   // Tryb sterowania
       tcflag_t  l_iflag;   // Tryb lokalny
           cc_t  c_cc{NCCS] // Znaki sterujące
}
```

`int tcgetattr(int fdes, struct termios *term)`

`term` Struktura opisu terminala
`fdes` Uchwyt do pliku specjalnego zwracany przez funkcję open

Funkcja powoduje nadanie wartości strukturze term która opisuje bieżący status terminala.

Modyfikacja struktury termios

Parametry terminala ustawia się poprzez modyfikację struktury termios.

```
struct termios ts;

tcgetattr(fd, &ts);                 // Odczyt parametrów
ts.c_flag &= ~(ECHO);               // Skasowanie echa
ts.c_flag |= (IHFLOW | OHFLOW);     // Ustawienie handshakingu
tcsetattr(fd,TCSANOW, &ts);         // Zapis nowych parametrow
```

Ustawianie parametrów terminala

`int tcsetattr(int fdes, int action, struct termios *term)`

`term`	Struktura opisu terminala
`action`	TCSANOW, TCSADRAIN, TCSAFLUSH
`fdes`	Uchwyt do pliku specjalnego zwracany przez funkcję open

Funkcja powoduje ustawienie trybu pracy terminala zgodnie z parametrami opisanymi w strukturze term.

Ustawienie szybkości wejściowej i wyjściowej

Szybkości transmisji także ustawia się poprzez modyfikację struktury termios. Jednak nie bezpośrednio ale poprzez zastosowanie funkcji.

```
int cfsetispeed(struct termios *term, speed_t speed)
int cfsetospeed(struct termios *term, speed_t speed)
```

`term`	Struktura opisu terminala
`speed`	Szybkość bodowa (np. 2400, 4800, itd.)

`cfsetispeed` – ustalenie szybkości wejściowej
`cfsetospeed` – ustalenie szybkości wyjściowej

```
#include <stdlib.h>
#include <sys/dev.h>
#include <fcntl.h>
#include <termios.h>

main(void) {
  int fd;
  struct termios termio;
  int res, baud;
  baud = 9600;

  fd = open("/dev/ser2",O_RDWR);
  // Pobranie parametrów terminala do termio
  res = tcgetattr(fd, &termio);

  termio.c_cflag &= ~(PARENB|PARODD|PARSTK|CSIZE|CSTOPB);
  termio.c_cflag |= PARENB;    // Parzystosc
  termio.c_cflag |= CS8;       // Znak 8 bit
  termio.c_cflag |= CSTOPB;    // 2 bity stopu

  // Ustawienie szybkości bodowej
  cfsetispeed(&termio,baud);
  cfsetospeed(&termio,baud);

  // Ustawienie parametrow terminala
  res = tcsetattr(fd, TCSANOW, &termio);
  if(res  == -1) {
     printf("Blad ust. atryb. portu: %d \n",errno);  exit(1);
  }
}
```

Przykład 1-1 Ustawianie parametrów portu szeregowego

1.7 Rozszerzone funkcje obsługi portu szeregowego

Testowanie czy w buforze odbiorczym są jakieś znaki

`int tcischars(int fdes)`

`fdes` **Uchwyt do pliku zwracany przez funkcję open**

Funkcja zwraca liczbę znaków dostępnych w buforze odbiorczym urządzenia identyfikowanego przez **`fdes`**.

```
#include <stdlib.h>
#include <fcntl.h>
#include <stdio.h>
#define SIZE 80

main(int argc, char *argv[]) {
  char c;
  char buf[SIZE];
  int fd,res;
  if(argc < 2) {
      printf("uzycie: terminal /dev/serx \n");
      exit(0);
  }
  printf("Port: %s\n",argv[1]);
  fd = open(argv[1],O_RDWR);
  if(fd < 0) { perror(“open”); exit(0); }
  do {
    if(tcischars(fd) > 0) {
       res = read(fd,&c,1);
       printf("%c",c);
       flushall();
    }
   if(tcischars(0) > 0) {
      gets(buf);
      res = write(fd,buf,strlen(buf));
    }
  } while(1);
  close(fd);
}
```

Przykład 1-2 Mini terminal – program terminal.c

Skasowanie buforów

```
int tcflush(int fdes, int what)
```

fdes Uchwyt do pliku zwracany przez funkcję open
what TCIFLUSH – wejściowy
TCOFLUSH – wyjściowy
TCIOFLUSH – wejściowy i wyjściowy

Funkcja kasuje bufor wejściowy, wyjściowy lub obydwa.

Oczekiwanie aż bufor zostanie wytransmitowany

```
int tcdrain(int fdes)
```

fdes Uchwyt do pliku zwracany przez funkcję open

Funkcja blokuje proces do czasu gdy bufor zostanie wytransmitowany.

Zaawansowana funkcja odczytu znaków

Funkcji **readcond** używa się w trybie RAW i nadpisuje ustawienia zdefiniowane w strukturze termios.

```
int readcond(int fdes, void *buf, int n, int min, int
time, int timeout)
```

fdes Uchwyt do pliku zwracany przez funkcję open
buf Adres bufora na znaki
n Nominalna liczba znaków do odczytu
min Minimalna liczba znaków do odczytu
time Czas pomiędzy znakami w 1/10 sek
timeout Limit czasowy w 1/10 sek

W strukturze **termios** zdefiniowany może być znak separatora ramki (ang. *forward*). Gdy znak ten wystąpi funkcja odblokowuje proces bieżący.

```
#include <termios.h>
…
int fd;
struct termios term;
…
tcgetattr(fd, &term);
fwd_char = 0x0C;
term.c_cc[VFWD] = fwd_char
tcsetattr(fd,TCSANOW,&term)
```

Przykład 1-3 Ustawienie znaku 0x01 jako separatora ramki

min	time	timeout	
M	0	0	Funkcja kończy się gdy odczytanych M znaków.
M	0	T	Funkcja kończy się gdy odczytanych M znaków lub upłynęło t* 0.1 sekundy.
M	T	0	Funkcja kończy się gdy odczytanych M znaków lub czas między kolejnymi znakami był większy od T* 0.1 sekundy.
M	T	t	Funkcja kończy się gdy odczytanych M znaków lub czas między kolejnymi znakami był większy od T* 0.1 sekundy lub upłynęło t* 0.1 sekundy.

Tab. 1-2 Warunki zakończenia funkcji `readcond`

Funkcja zwraca:

> 0 Liczba odczytanych znaków
= 0 Przeterminowanie
< 0 Błąd

```
#include <unistd.h>
#include <termios.h>
#include <fcntl.h>
#define PORT_COM1 "/dev/ser1"
#define SIZE 64
#define CR 13

int main(int argc, char * argv[]) {
  int fd,speed, timeout,rd;
  struct termios term;
  char buf[SIZE];
  fd = open(PORT_COM1,O_RDWR);
  /* ustawienia predkosci transmitowania */
  speed=9600;
  timeout = 20; // 2 sek
  tcgetattr(fd,&term);
  term.c_cc[VFWD] = LF;
  tcsetattr( fd, TCSANOW, &term);
  do {
    rd = readcond(fd,buf,SIZE,SIZE,0,timeout);
    if(rd >0) printf("rd: %d buf: %s\n",rd,buf);
    else printf("rd: %d\n",rd);
  } while(1);
  close(fd);
  return 0;
}
```

Przykład 1-4 Odbiór ramek zakończonych znakiem LF (program `serial3.c`)

1.8 Testowanie aplikacji używających portu szeregowego

1.8.1 Metody programowe

Testowanie portu szeregowego może być wykonywane metodami programowymi.

Testowanie obecności portów szeregowych

```
$ls /dev/ser*
/dev/ser1  /dev/ser2
```

Testowanie ustawienia portu szeregowego

```
stty < /dev/ser1
Name:  /dev/ser1
Type:  serial
Opens: 1
+raw +echoke +echoctl +imaxbel
+ihflow +ohflow
 intr=^C  quit=^\ erase=^?  kill=^U   eof=^D start=^Q  stop=^S
susp=^Z
lnext=^V   min=01  time=00   pr1=^[   pr2=5B  left=44 right=43
up=41
 down=42   ins=40   del=50  home=48   end=59
par=none bits=8 stopb=1 baud=57600 rows=0,0
```

Ustawianie parametrów transmisji
Parametry transmisji ustawia się poleceniem stty

Wysyłanie i odbiór znaków
Wysyłanie pliku na port /dev/ser1:

```
$cat plik.txt > /dev/ser1
```

Wyświetlanie zawartości portu /dev/ser1 na konsoli:

```
$cat /dev/ser1
```

Program terminalowy

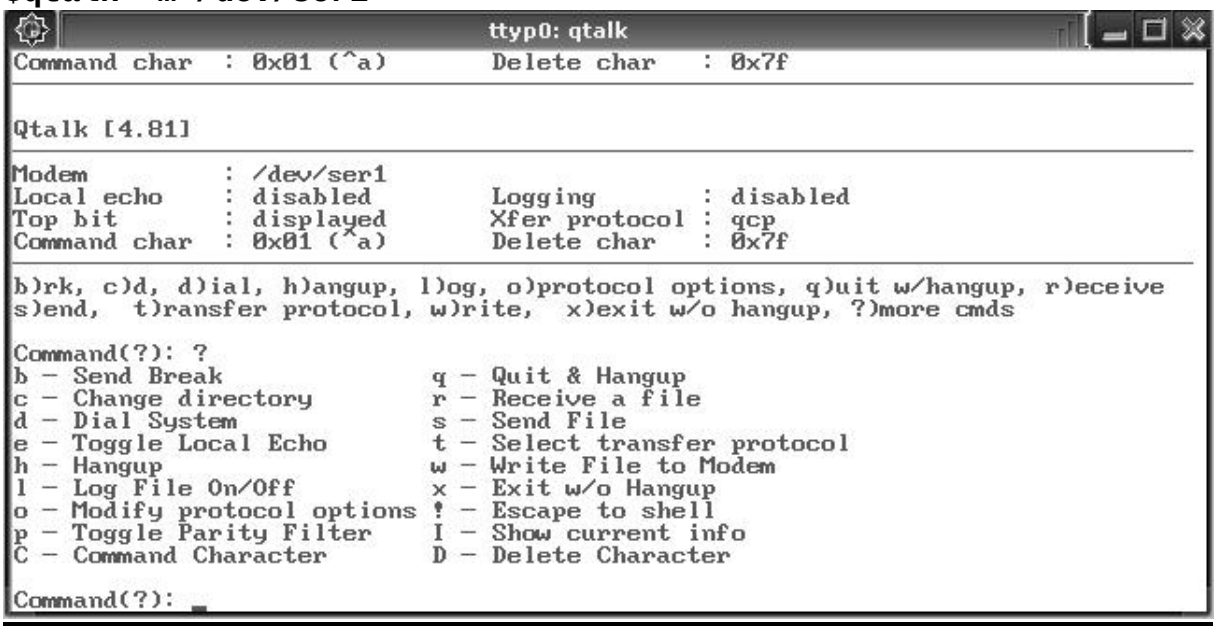
Program terminalowy ułatwia testowanie aplikacji używających portu szeregowego. Program terminalowy umożliwia:

- Wysyłanie i odbiór znaków
- Wysyłanie i odbiór plików według różnych protokołów: qcp, zmodem
- Rejestrację przychodzących znaków w pliku

W systemach operacyjnych dostępne są rozmaite programy terminalowe.

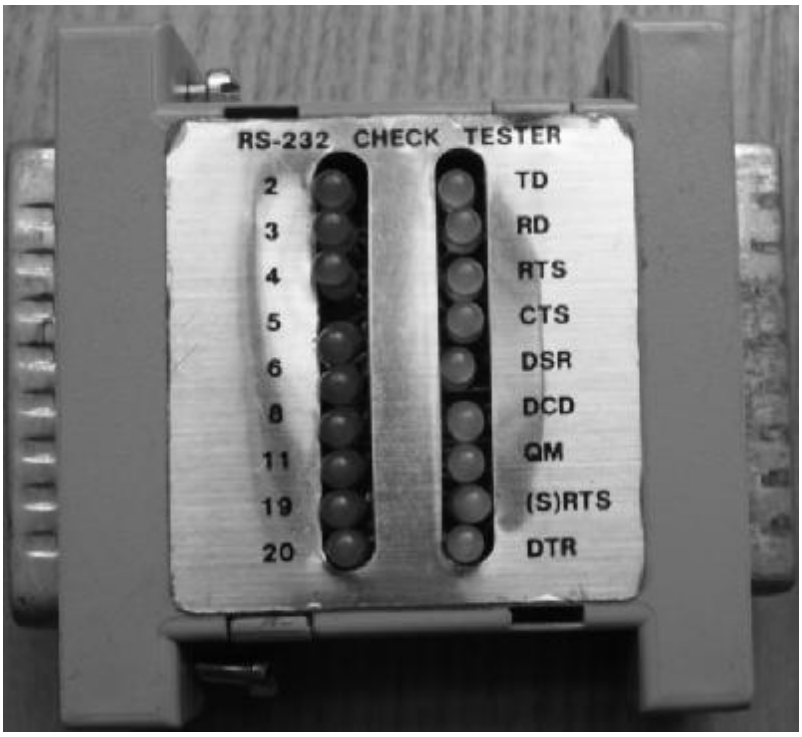
System operacyjny	Program terminalowy
QNX6	qtalk
Windows	Hyperterm
Linux	Putty

$qtalk -m /dev/ser1



Rys. 1-5 Polecenia programu terminalowego qtalk

Sprzętowy tester linii RS232C



Rys. 1-6 Tester łącza RS232

1.9 Komunikacja według protokołu MODBUS

Protokół Modbus został opracowany w firmie Modicon w 1979r. Jest Szeroko stosowany w aplikacjach automatyki przemysłowej o niskich wymaganiach dotyczących szybkości i częstości transmisji danych. Jest standardem przyjętym przez większość producentów sterowników przemysłowych dla asynchronicznej komunikacji.

Cechy:
•Dostęp do łącza na zasadzie „Master - Slave”
•Zwykle protokół pracuje z niewielkimi prędkościami transmisji danych (typowe: 9.6 Kb/s, 19.2 Kb/s) ale możliwe też większe prędkości
•Dwa różne tryby transmisji ASCII (znakowy) lub RTU (binarny)
•Komunikaty zawierające polecenia i odpowiedzi mają identyczną strukturę
•Maksymalna długość komunikatów wynosi 256 bajtów

Zalety:
•Prostota zastosowanych w nim rozwiązań
•Jawność specyfikacji protokołu
•Zabezpieczenie przesyłanych komunikatów przed błędami
•Potwierdzanie wykonania rozkazów zdalnych i sygnalizacja błędów
•Stały format ramki i zestaw standardowych funkcji służących wymianie danych
•Mechanizmy zabezpieczające przed zawieszeniem systemu

W modelu ISO/OSI protokół Modbus zajmuje trzy warstwy:
•1- Warstwę fizyczną - (ang. *physical*)
•2- Warstwę łącza danych - (ang. *data link*)
•7- Warstwę aplikacji - (ang. *application*)

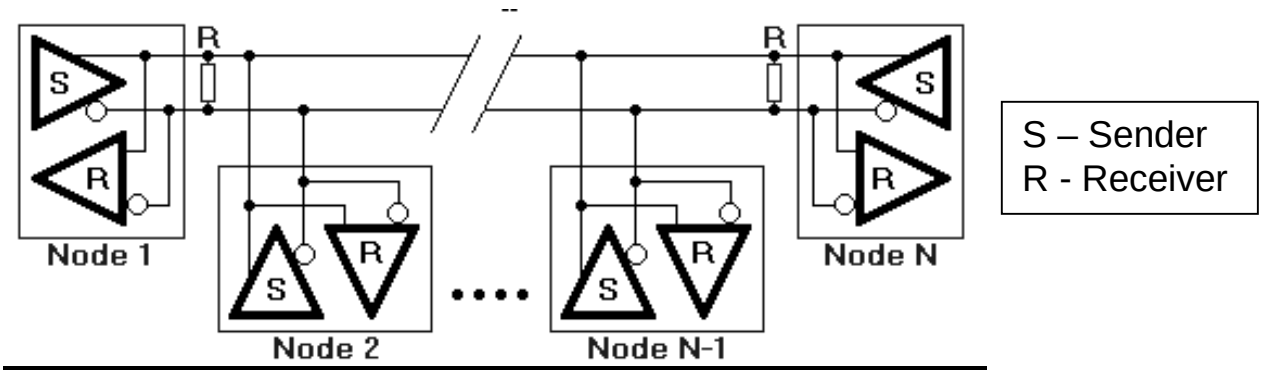
1.9.1 Warstwa fizyczna i łącza

W warstwach 1 i 2 stosuje się:

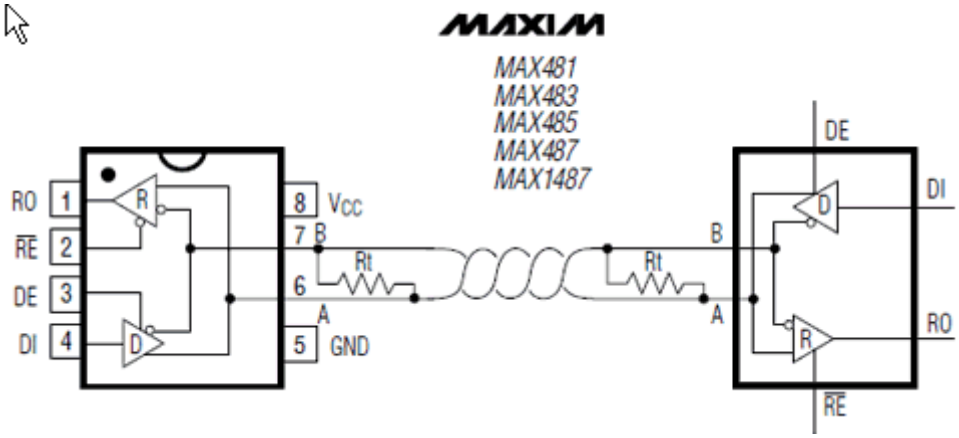
- Standard RS232C, RS485
- Standard TCP/IP

Standard RS485

- Wyjścia nadajników mają trzy stany: wysoki, niski i wysokiej impedancji. W stanie normalnym stacji (odbiór) nadajnik ma wysoka impedancję. Gdy coś nadaje przechodzi w stan wysoki lub niski.
- Urządzenia podłączone do magistrali dwu lub trójprzewodowej.
- Wymagane prędkości przesyłowe to 9600 bps i 19.2 Kbps, Inne prędkości jakie można zaimplementować to: 1200, 2400, 4800, … 38400 bps, 56 Kbps, 115 Kbps, …
-



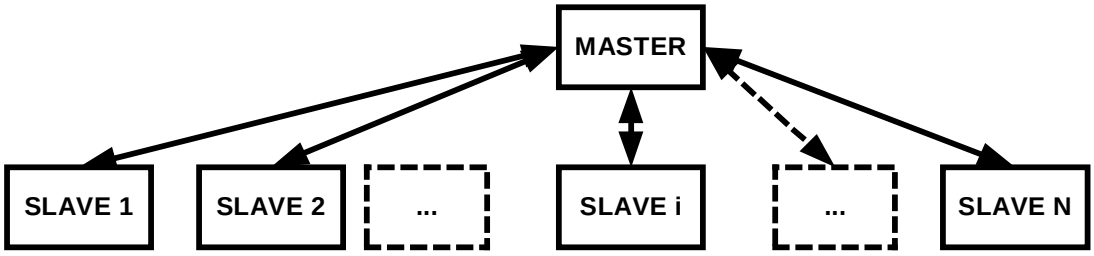
Rys. 1-7 Struktura magistrali RS485



Rys. 1-8 Konwerter RS485 (z katalogu firmy MAXIM)

1.9.2 Struktura komunikacji

W protokole MODBUS przyjęta jest komunikacja typu Master – Slave. Wyróżniona jest jedna stacja nadrzędna (ang. *Master*) i N stacji podrzędnych (ang. *Slave*). Stacja Master wysyła do stacji Slave zapytania/polecenia i uzyskuje odpowiedzi. Stacje Slave nie mogą się ze sobą komunikować ani niczego wysyłać bez zapytania.

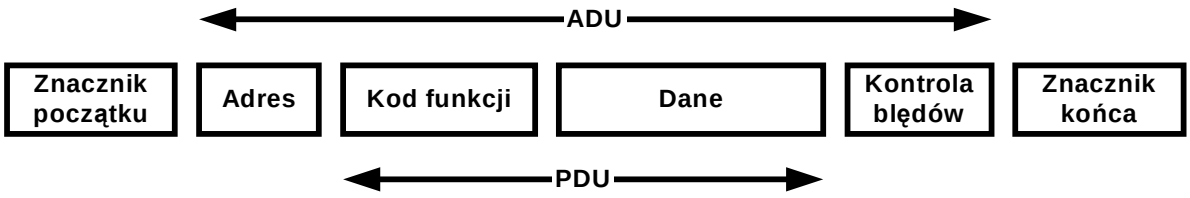


Rys. 1-9 Struktura komunikacji typu MASTER – SLAVE

1.9.3 Format ramki

Informacja pomiędzy stacjami przesyłana jest w postaci jednostek protokołu ADU (ang. *Apllication Data Unit*) zawierające adres docelowy, kod funkcji, dane i pole kontrolne. Kod funkcji i dane tworzą tak zwane PDU (ang. *Protocol Data Unit*).
Adresy jednostek są liczbami z zakresu od 1 do 255.

- Kod funkcji – liczba z zakresu 1-255 która specyfikuje żądanie lub odpowiedź
- Dane – informacja przesyłana do/z jednostki
- Pole kontrolne – zawiera dodatkowe dane umożliwiające sprawdzenie poprawności pozostałych pól.



Rys. 1-10 Struktura ramki protokołu Modbus

Adres	Znaczenie
0	Adres rozgłaszania
1 - 247	Indywidualne adresy węzłów podrzędnych
248 - 255	Zarezerwowane

Tab. 1-3 Struktura adresowania protokołu MODBUS

Protokół Modbus definiuje trzy rodzaje ramek PDU protokołu

- Żądanie – (ang. *Request PDU*)
- Odpowiedź – (ang. *Response PDU*)
- Wyjątek – (ang. *Exception Response PDU*)

Ramka żądanie zawiera:

Kod funkcji	1 bajt	Specyfikacja co ma zrobić Slave
Dane	N bajtów	Dane zależą od funkcji, pole zawiera adresy, liczby, kody podfunkcji itd

Ramka odpowiedzi zawiera:

Kod odpowiedzi	1 bajt	Powtórzenie kodu żądania od Master
Dane	bajtów	Dane zależą od funkcji, pole zawiera adresy, liczby, kody podfunkcji itd

Ramka wyjątku zawiera:

Kod błędu	1 bajt	Powtórzenie kodu żądania od Master + 0x80
Kod wyjątku	1 bajt	Kod wyjątku

1.9.4 Typy tansmisji

Wyróżnia się dwa typy transmisji:

- ASCII (znakowy)
- RTU (binarny)

-

Tryb ASCII

- Znacznikiem początku ramki jest znak dwukropka :
- Znacznikiem końca ramki są znaki CR i LF
- Jako pole kontrolne stosuje się LCR
- Pola adresu, funkcji i kodu kodowane są jako liczby HEX – każdy bajt kodowany jako 2 znaki 0 ... 9, A ... F.

Znacznik początku	Adres	Funkcja	Dane	LRC	Znacznik końca
1	2	2	2*N <= 252	2	2
:	adres	funkcja	N bajtów w postaci HEX	suma kontrolna	CR LF

Tab. 1-4 Struktura ramki protokołu MODBUS ASCII

Tryb binarny

- Wiadomości rozpoczynają się odstępem czasowym minimum 3,5 * Tz (Tz oznacza czas trwania pojedynczego znaku).
- Przesył w postaci ciągłej, odstępy między kolejnymi znakami max 1,5 * Tz.
- System kodowania: 8-bitowy binarny

Znacznik początku	Adres	Funkcja	Dane	CRC	Znacznik końca
	1	1	N <= 252	1	
Przerwa 3.5 Tz	adres	funkcja	N bajtów w postaci binarnej	Pole kontrolne	Przerwa 3.5 Tz

Tab. 1-5 Struktura ramki protokołu MODBUS RTU

Pola kontrolne

- Dla trybu ASCII stosuje się sumę kontrolną typu LRC (ang. *Longitudinal Redundancy Check*)
- Dla trybu RTU stosuje się sumę kontrolną typu CRC (ang*. Cyclical Redundancy Check*).

Wartość LRC (8-bitowa) jest dołączana na końcu ramki w postaci dwóch znaków ASCII.

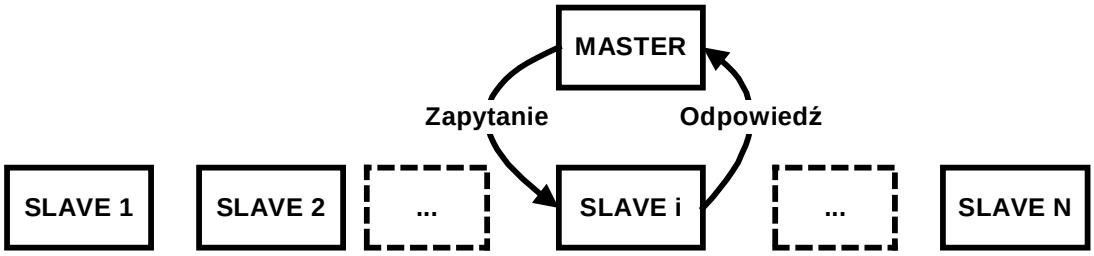
Obliczanie LRC polega na:

- Sumowaniu kolejnych 8-bitowych bajtów wiadomości
- Wydzielenie młodszego bajtu poprzez wyznaczenie reszty z dzielenia przez 256
- Wyznaczeniu uzupełnienia dwójkowego wyniku (dokonanie negacji bajtu i dodanie do niego 1).

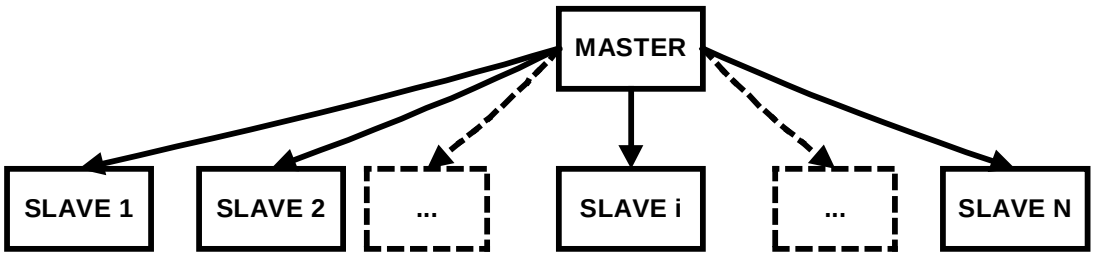
Transakcje

W protokole MODBUS definiuje się dwa rodzaje transakcji:

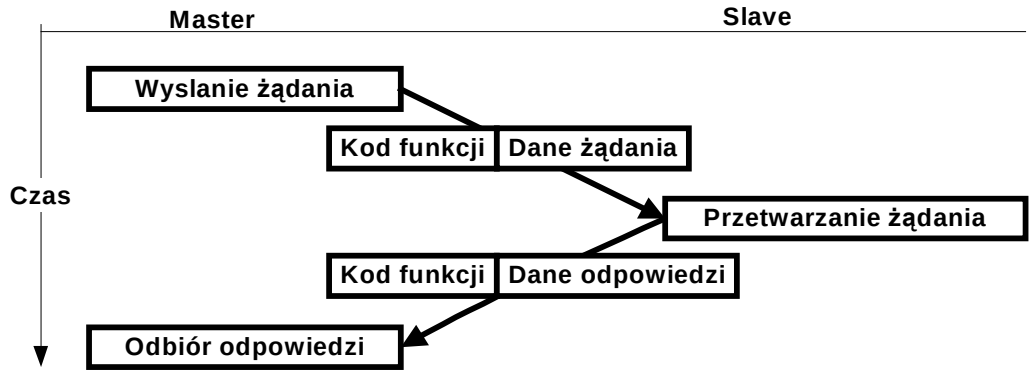
- Transakcja typu zapytanie – odpowiedź (ang. *Unicast*)
- Transakcja typu rozgłoszenie – (ang. *Broadcast*)



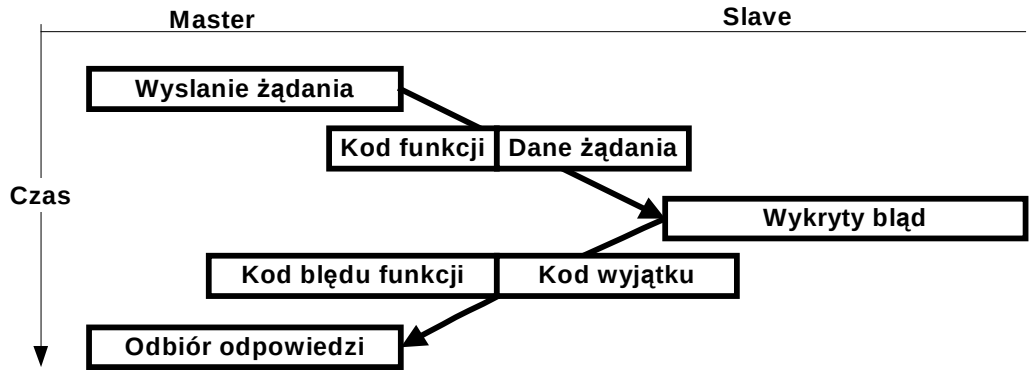
Rys. 1-11 Transakcja typu zapytanie – odpowiedź



Rys. 1-12 Transakcja typu rozgłoszenie



Rys. 1-13 Przebieg czasowy bezbłędnej transakcji typu zapytanie – odpowiedź



Rys. 1-14 Przebieg czasowy błędnej transakcji typu zapytanie – odpowiedź

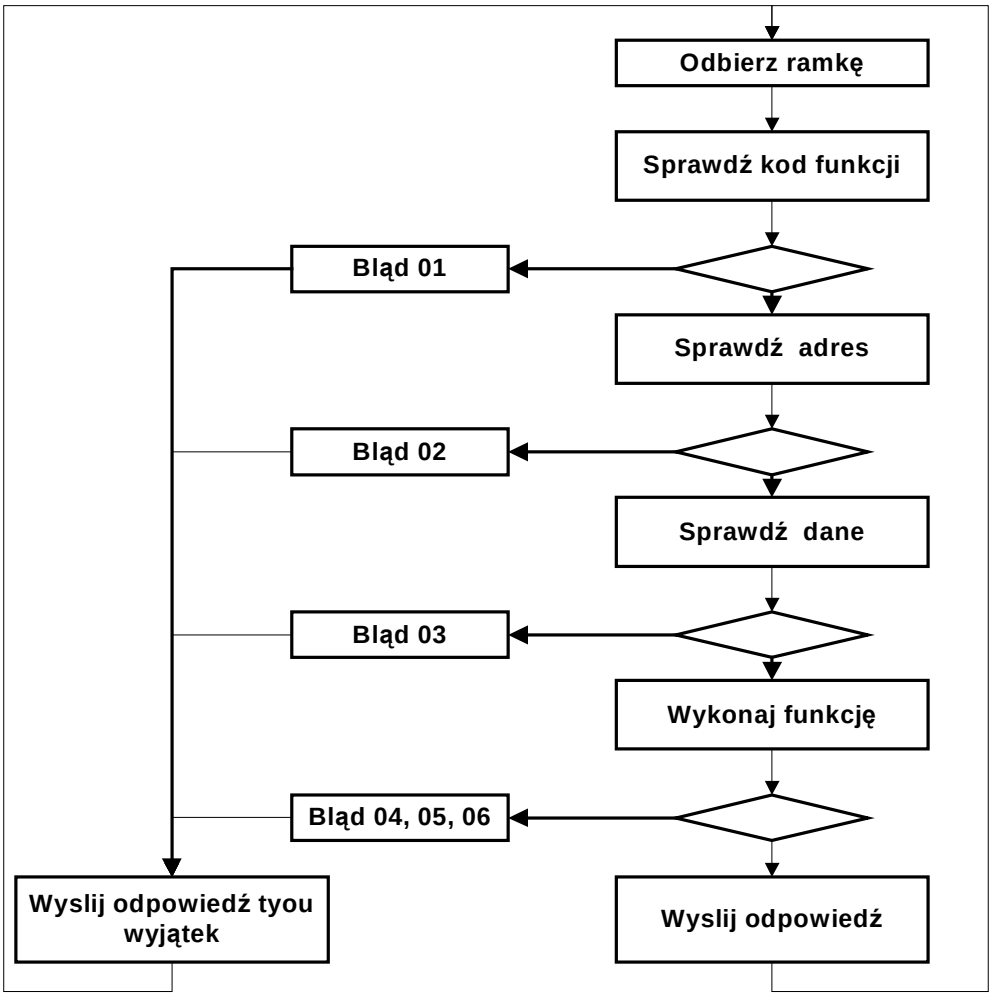
1.9.5 Obsługa błędów

W protokole Modbus prowadzona jest kontrola błędów na dwóch poziomach:

- Suma kontrolna ramki (CRC, LRC)
- Poprawność adresu
- Poprawność kodu funkcji
- Spójność danych dla danej funkcji

Reakcja na błędy:

- Gdy ramka ma złą sumę kontrolną nie należy na nią odpowiadać
- Gdy ramka nie jest przeznaczona dla danej stacji nie należy na nią odpowiadać
- Gdy ramka posiada nieprawidłowe dane lub nastąpi błąd wykonania to odpowiadamy ramką typu wyjątek



Rys. 1-15 Obsługa błędów w stacji Slave

Standardowe kody wyjątków:
•01 – niedozwolona funkcja
•02 – niedozwolony zakres danych (adres)
•03 – niedozwolona wartość danej
•04 – błąd urządzenia Slave
•05 – potwierdzenie pozytywne
•06 – brak gotowości urządzenia Slave
•07 – potwierdzenie negatywne
•08 – błąd parzystości pamięci

1.9.6 Realizacja funkcji

Definicja protokołu wyróżnia trzy rodzaje funkcji:

- Publiczne – dobrze zdefiniowane, unikalne, zdokumentowane, istnieją procedury walidacji
- Zdefiniowane przez użytkownika – zakresy 65 – 72 i 100 - 110
- Zarezerwowane

Od	Do	Przeznaczenie funkcji
1	64	Publiczne
65	72	Zdefiniowane przez użytkownika
73	99	Publiczne
100	110	Zdefiniowane przez użytkownika
111	127	Publiczne

Tab. 1-6 Przeznaczenie kodów funkcji Modbus

Typ	Długość	Dostęp	Opis
Wejścia binarne *discrete inputs*	1 bit	Odczyt	Dane 0/1 ustawiane przez urządzenie
Przekaźniki *coils*	1 bit	Odczyt – zapis	Dane 0/1 które mogą być zmienione przez zewnętrzną aplikację
Rejestry wejściowe *input registers*	16 bit słowo	Odczyt	Dane ustawiane przez urządzenie
Rejestry wyjściowe *Holding registers*	16 bit słowo	Odczyt – zapis	Dane mogą być zmienione przez zewnętrzną aplikację

Tab. 1-7 Typy danych w protokole MODBUS

Funkcja (dec)	Opis	Grupa
01	Czytaj status przekaźnika	Dostęp do danych binarnych
02	Czytaj wejście binarne	
05	Ustaw pojedynczy przekaźnik	
15	Ustaw wiele przekaźników	
03	Czytaj rejestr wyjściowy	Dostęp do danych 16 bitowych
04	Czytaj rejestr wejściowy	
06	Ustaw pojedynczy rejestr wyjściowy	
07	Czytaj status wyjątku	
16	Ustaw wiele rejestrów wyjściowych	
23	Czytaj / pisz wiele rejestrów	
24	Czytaj kolejkę FIFO	
20	Czytaj rekord z pliku	Dostęp do plików
21	Pisz rekord do pliku	
08	Diagnostyka	Diagnostyka
11	Podaj licznik zdarzeń komunikacyjnych	
12	Podaj dziennik zdarzeń komunikacyjnych	
17	Podaj identyfikację stacji Slave	
43	Czytaj identyfikację urządzenia	

Tab. 1-8 Ważniejsze polecenia protokołu MODBUS

Przykład – funkcja 4 czytanie rejestru wejściowego
Funkcja służy do odczytu od 1 do 125 rejestrów zdalnego urządzenia.
Żądanie specyfikuje:
Adres rejestru (zaczynając od zera)
Liczbę rejestrów

Polecenie:

	Długość	Wartość
Kod funkcji	1 bajt	0x04
Adres startowy	2 bajty	0x0000 do 0xFFFF
Liczba rejestrów	2 bajty	0x0001 do 0x007D

Odpowiedź:

	Długość	Wartość
Kod funkcji	1 bajt	0x04
Liczba bajtów	1 bajt	2 * N
Zawartość rejestrów	2 * N bajtów	

N – liczba rejestrów

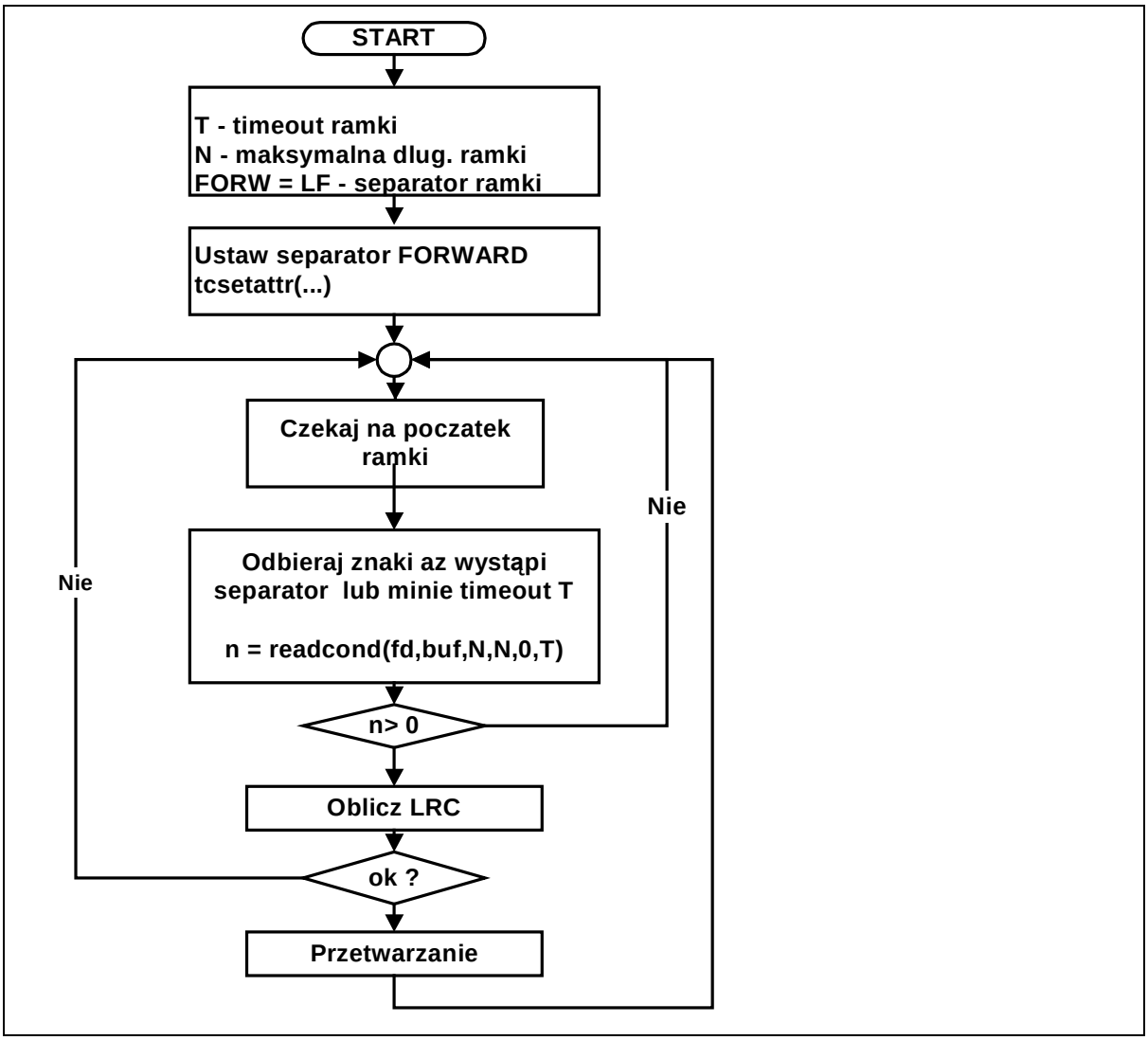
Odpowiedź zawiera po 2 bajty na każdy odczytywany rejestr. Zawartość rejestru zakodowana jako big-Endian , najpierw starsza a potem młodsza część.

Błędy:

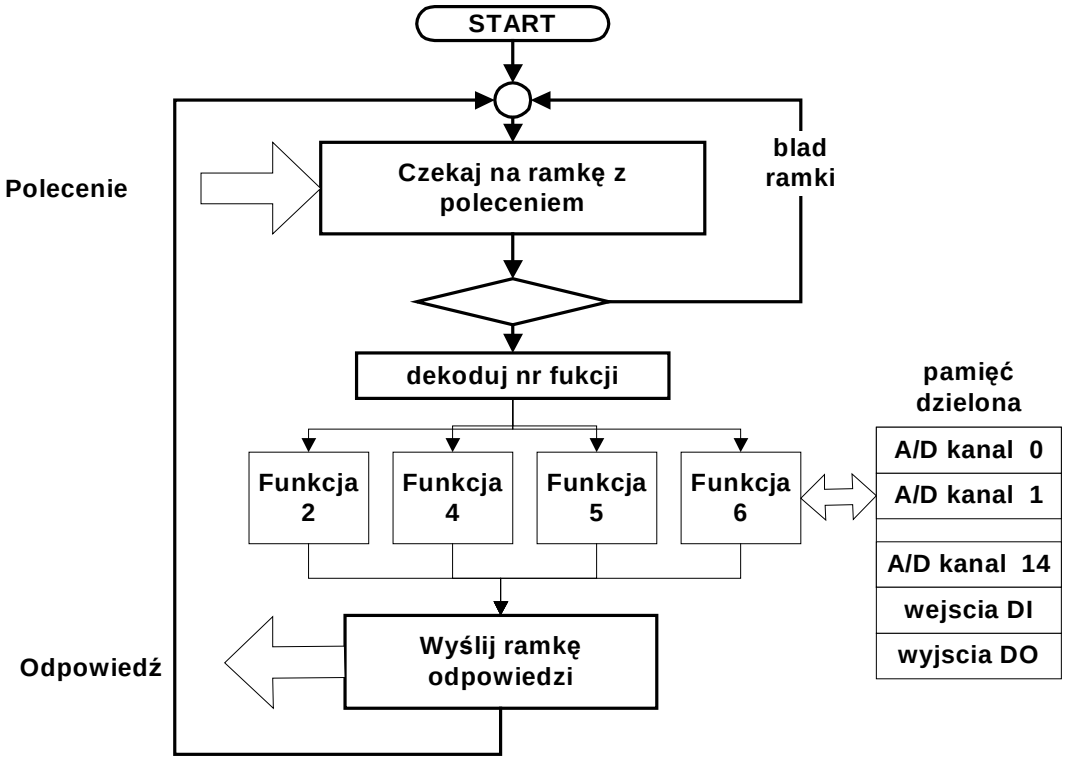
Kod błędu	1 bajt	0x84
Kod wyjątku	1 bajt	01, 02, 03, 04

Żądanie		Odpowiedź	
Nazwa pola	Zawartość	Nazwa pola	Zawartość
Funkcja	0x04	Funkcja	0x04
Adres – Hi	0x00	Liczba bajtów – Hi	0x02
Adres – Lo	0x00	Zawartość rej. – Hi	0x00
Liczba rejestrów - Hi	0x00	Zawartość rej. – Lo	0x0A
Liczba rejestrów - Lo	0x01		

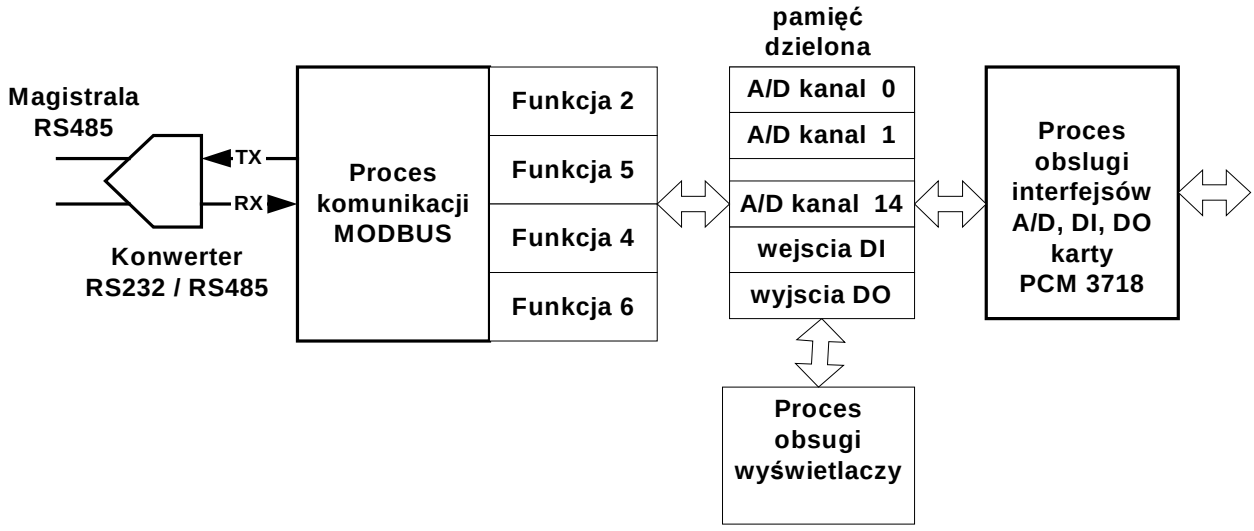
Przykład 1-5 Odczytu jednego rejestru o adresie 0. Zawartość rejestru 0x000A



Rys. 1-16 Procedura odbioru ramki protokołu MODBUS ASCII

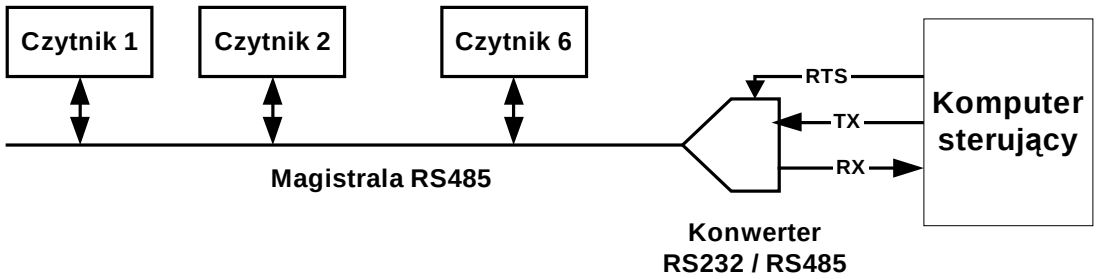


Rys. 1-17 Struktura procesu obsługi protokołu komunikacji Modbus



Rys. 1-18 Struktura oprogramowania stacji akwizycji danych komunikującej się za pomocą protokołu Modbus

1.10 Przykład aplikacji – system kontroli czasu pracy



Rys. 1-19 Struktura urządzeniowa

Dane od czytnika:

1. Numer czytnika
2. Numer karty
3. Data i czas przejścia

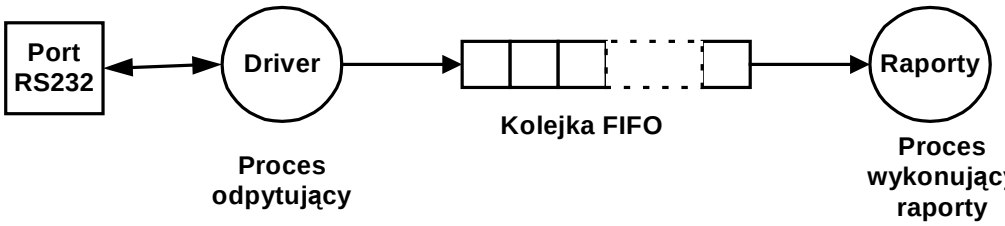
Polecenia do czytnika:

1. Podaj dane
2. Wykonaj restart
3. Ustaw datę i czas

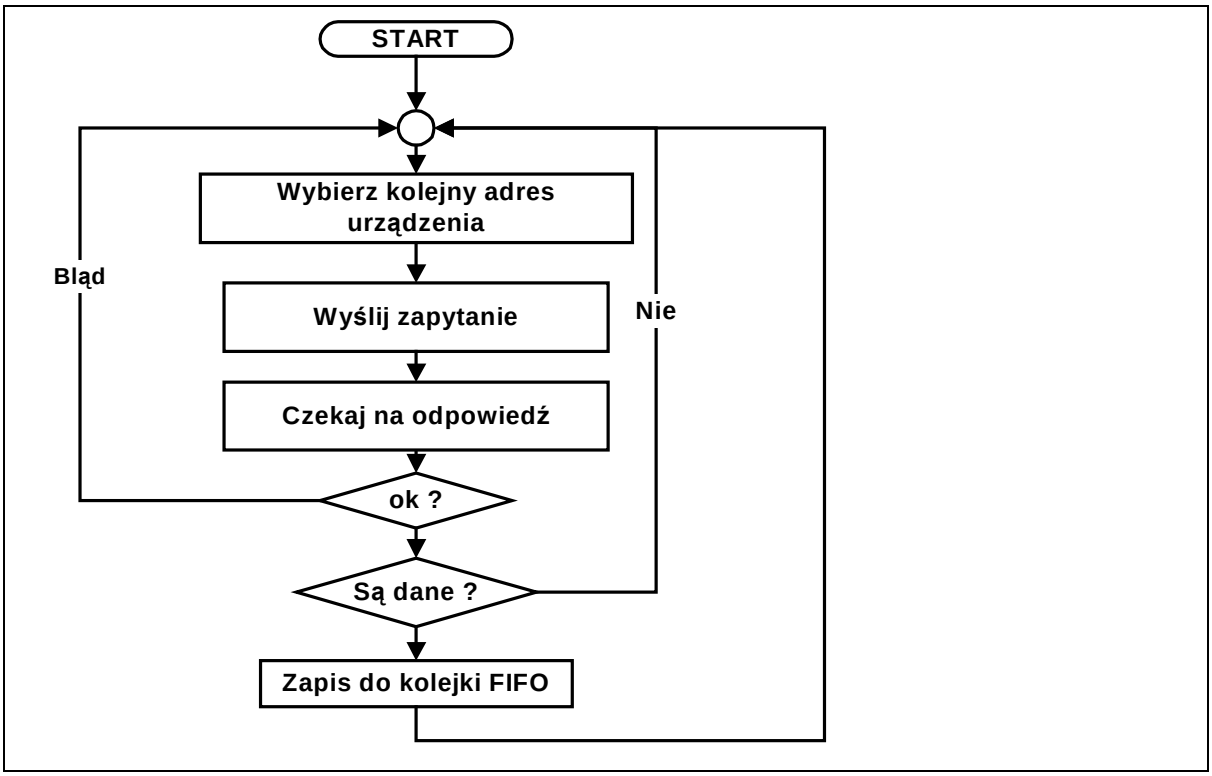
Odpowiedzi czytnika:

1. Rekord danych – numer czytnika, numer karty, data i czas przejścia
2. Brak danych

Tryb pracy: odpytywanie kolejnych urządzeń przez komputer centralny
Protokół transmisji: MODBUS



Rys. 1-20 Struktura procesów po stronie komputera centralnego



Rys. 1-21 Struktura procesu odpytywania po stronie komputera centralnego

1.11 Literatura

[1] MODBUS APPLICATION PROTOCOL SPECIFICATION V1.1b
http://www.Modbus-IDA.org
[2] Mielczarek Wojciech, Szeregowe interfejsy cyfrowe, Helion, Gliwice 1993