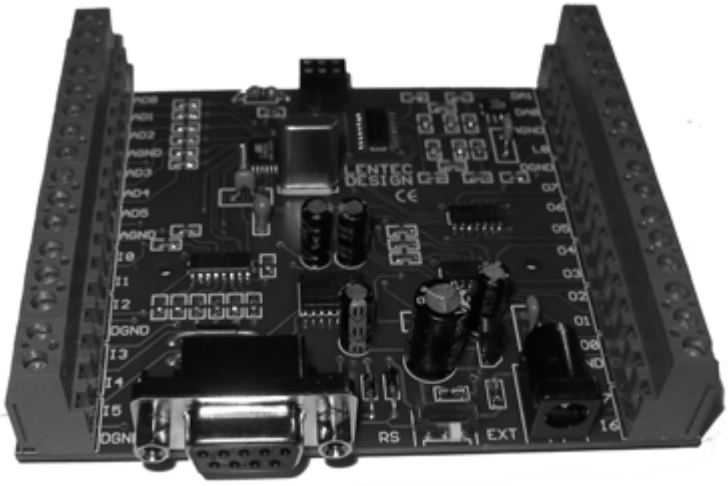
INSTRUKCJA OBSŁUGI URZĄDZENIA

LENDEVICE80RS232

LENTEC DESIGN



WPROWADZENIE

Urządzenie LENDEVICE80RS232 służy do rejestrowania i generowania sygnałów napięciowych. Urządzeniem można sterować dowolnym programem komputerowym, który komunikuje się z urządzeniem za pomocą łącza COM (RS232) komputera.

Urządzenie posiada możliwość:

- **rejestrowania sygnałów analogowych z rozdzielczością 10 bitów w zakresie napięć 0 – 5 V;**
- **generowania sygnałów analogowych z rozdzielczością 10 bitów w zakresie napięć 0 – 5 V;**
- **rejestrowania sygnałów cyfrowych TTL w zakresie napięć 0 – 5 V;**
- **generowania sygnałów cyfrowych TTL w zakresie napięć 0 – 5 V;**
- **zliczania impulsów (funkcja licznika), maksymalna częstotliwość impulsów 900 kHz.**

Ponadto istnieje możliwość zmiany programu urządzenia i w przyszłych aktualizacjach rozszerzy się zakres zastosowań tych urządzeń.

Istnieje możliwość stworzenia specjalistycznego oprogramowania przez producenta dla odbiorców chcących wykorzystywać urządzenie we własnych aplikacjach.

Do zasilania urządzeń LENDEVICE80RS232 można wykorzystać możliwości łącza COM komputera lub zewnętrzne źródło zasilania o stałym napięciu z zakresu 9 – 12 V.

Dziękujemy Państwu za wybranie produktu firmy Lentec Design i życzymy samych udanych projektów.

Zespół Lentec Design

www.lentecdesign.com

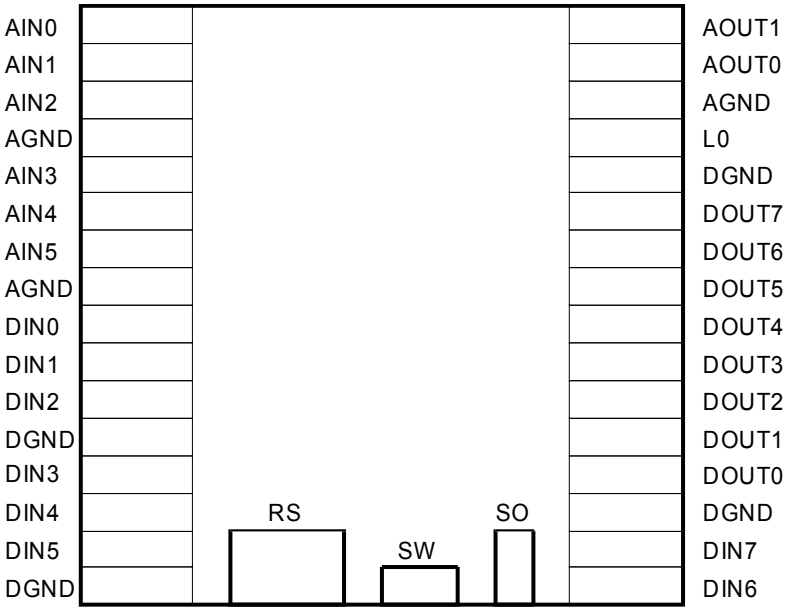
PRAWA AUTORSKIE

Dokument ten był wielokrotnie sprawdzany i Lentec Design wierzy, że informacje w nim zawarte są poprawne. Jeśli jednak w dokumencie będzie błąd, Lentec Design zastrzega sobie prawo do zmiany treści tego dokumentu bez informowania o tym aktualnych użytkowników. Jeśli czytelnik znajdzie błąd, powinien poinformować i skonsultować się z Lentec Design. W żadnym przypadku Lentec Design nie będzie odpowiedzialny za jakiekolwiek zniszczenia wynikające z informacji zawartych w tym dokumencie.

SPIS TREŚCI:

1. UŻYTKOWANIE URZĄDZENIA

Usytuowanie wejść i wyjść urządzenia LENDEVICE80RS232 zostało przedstawione na rysunku 1.



Rys. 1. *Rozmieszczenie wejść oraz wyjść urządzenia LENDEVICE80RS232 (widok od strony zacisków).*

Wejścia AGND oraz DGND są połączone z masą urządzenia 0 V (wszystkie jednostki są podane w układzie SI).

1.1. WEJŚCIA ANALOGOWE

Urządzenie posiada 6 wejść analogowych: AIN0, AIN1, ..., AIN5 służących do pomiaru napięcia w zakresie od 0 V do 5 V. Jeżeli mierzone napięcie nie będzie zawierać się w tym zakresie, urządzenie może ulec uszkodzeniu. Chcąc zmniejszyć wpływ zakłóceń, powinno się korzystać z wejść masy AGND usytuowanych w pobliżu wejść AIN0, ..., AIN5. Wejście analogowe pobiera typowo prąd 10 µA (zob. „Załącznik C: Parametry wejść analogowych”, str. 36).

W celu zmierzenia napięcia na wejściach analogowych należy wykonać jedną z funkcji [metod] opisanych w rozdz. „Pobieranie wartości napięć analogowych”, str. 23 (zob. rozdz. „Interfejsy w C oraz C++”, str. 12):

- `analog_input_len(...)`
  `[analog_input(...)]`
- `AnalogInputLen(...)`
  `[AnalogInput(...)]`

1.2. WYJŚCIA ANALOGOWE

Urządzenie posiada 2 wyjścia analogowe: AOUT0, AOUT1 o impedancji wyjściowej typowo 50 Ω. Na wyjściach analogowych można wytworzyć napięcie w zakresie od 0 V do 5 V. Maksymalny szum na tych wyjściach wynosi 10 mV między szczytowymi wartościami generowanego napięcia ($V_{peak-peak}$) Podłączenie zewnętrznych źródeł napięciowych do wyjść analogowych może zniszczyć urządzenie. Maksymalny prąd, jaki można pobrać z wyjścia wynosi 10 mA (zob. „Załącznik D: Parametry wyjść analogowych", str. 37).

W celu ustawienia wartości napięcia na wyjściu analogowym należy wykonać jedną z poniższych funkcji [metod] opisanych w rozdz. „Sterowanie wyjściami analogowymi", str. 23 (zob. rozdz. „Interfejsy w C oraz C++", str. 12):

- `analog_output_len(...)`
  `[analog_output(...)]`
- `AnalogOutputLen(...)`
  `[AnalogOutput(...)]`

1.3. WEJŚCIA CYFROWE

Urządzenie posiada 8 wejść cyfrowych: DIN0, DIN1, DIN2, ..., DIN7.

„Załącznik E: Parametry wejść cyfrowych oraz licznika" na str. 37 zawiera parametry wejść cyfrowych.

W celu zarejestrowania stanów na wejściach cyfrowych należy wykonać jedną z poniższych funkcji [metod] opisanych w rozdz. „Pobieranie stanów logicznych na wejściach cyfrowych", str. 24 (zob. rozdz. „Interfejsy w C oraz C++", str. 12):

- `digital_input_len(...)`
  `[digital_input(...)]`
- `is_hi_digital_input_len(...)`
  `[is_hi_digital_input(...)]`
- `is_lo_digital_input_len(...)`
  `[is_lo_digital_input(...)]`
- `DigitalInputLen(...)`
  `[DigitalInput(...)]`
- `IsHiDigitalInputLen(...)`
  `[IsHiDigitalInput(...)]`
- `IsLoDigitalInputLen(...)`
  `[IsLoDigitalInput(...)]`

1.4. WYJŚCIA CYFROWE

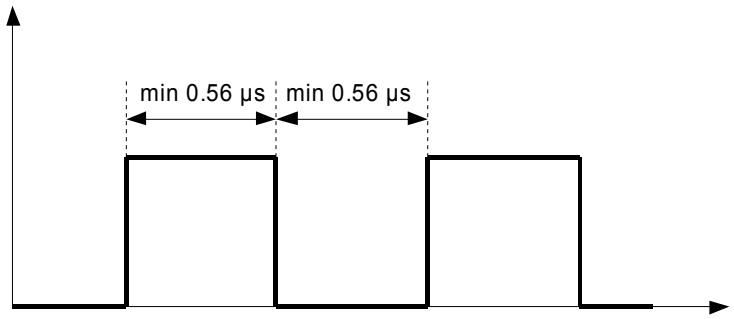
Urządzenie posiada 8 wyjść cyfrowych: DOUT0, DOUT1, ..., DOUT7. Podłączanie zewnętrznych źródeł napięciowych do wyjść cyfrowych może zniszczyć urządzenie. Maksymalny prąd, jaki może przepływać przez wyjścia cyfrowe wynosi 5 mA (zob. „Załącznik F: Parametry wyjść cyfrowych" na str. 37).

W celu ustawienia stanów na wyjściach cyfrowych należy wykonać jedną z poniższych funkcji [metod] opisanych w rozdz. „Sterowanie wyjściami cyfrowymi”, str. 26 (zob. rozdz. „Interfejsy w C oraz C++”, str. 12):

- `digital_output_len(...)`
  `[digital_output(...)]`
- `hi_digital_output_len(...)`
  `[hi_digital_output(...)]`
- `lo_digital_output_len(...)`
  `[lo_digital_output(...)]`
- `DigitalOutputLen(...)`
  `[DigitalOutput(...)]`
- `HiDigitalOutputLen(...)`
  `[HiDigitalOutput(...)]`
- `LoDigitalOutputLen(...)`
  `[LoDigitalOutput(...)]`

1.5. WEJŚCIE LICZNIKA

Urządzenie posiada jedno wejście licznika L0, które może zliczać impulsy. Zliczane impulsy muszą mieć stan wysoki (ponad 3,7 V) przez minimalny okres czasu 0,56 µs oraz stan niski (poniżej 1,3 V) przez minimalny okres czasu 0,56 µs (zob. rysunek 2). Maksymalna częstotliwość rejestrowanych impulsów wynosi 900 kHz (zob. „Załącznik E: Parametry wejść cyfrowych oraz licznika”, str. 37).



Rys. 2. *Charakterystyka impulsów, jakie mogą być rejestrowane przez urządzenie na wejściu licznika LO.*

W celu odczytania liczby zliczonych impulsów należy wykonać jedną z poniższych funkcji [metod] opisanych w rozdz. „Obsługa licznika”, str. 28 (zob. rozdz. „Interfejsy w C oraz C++”, str. 12):

- `DWORD get_counter_len(...)`
  `[unsigned long get_counter(...)]`
- `BOOL GetCounterLen(...)`
  `[bool GetCounter(...)]`

W celu wyzerowania licznika należy wykonać jedną z poniższych funkcji [metod]

opisanych w rozdz. „Obsługa licznika”, str. 28 (zob. rozdz. „Interfejsy w C oraz C++”, str. 12):

- `VOID clear_counter_len(...)`
  `[void clear_counter(...)]`
- `BOOL ClearCounterLen(...)`
  `[bool ClearCounter(...)]`

1.6. ZASILANIE URZĄDZENIA

Urządzenie może być zasilane jednym z dwóch źródeł:

- Z portu COM komputera: umożliwia zasilanie urządzenia oraz wykorzystanie typowo 5 mA na wszystkich jego wyjściach; jeżeli pobór prądu z wyjść urządzenia będzie zbyt duży, to urządzenie może nie działać poprawnie.

  Przed podłączeniem zewnętrznych urządzeń i układów elektrycznych do urządzenia LENDEVICE80RS232 należy się upewnić, że w urządzeniu jest włączone zasilanie. W tym celu należy uruchomić program OpenLenDevice (skrót do programu powinien znajdować się na pulpicie lub w menu START) i uruchomić urządzenie (zob. rozdz. „Program OpenLenDevice”, str. 31).

  Raz uruchomione urządzenie LENDEVICE80RS232 powinno działać w czasie całej sesji systemu Windows (czyli gdy nie dochodzi do wylogowania użytkownika).

  Drugi sposób na upewnienie się, że urządzenie jest zasilane, to wywołanie jendej z poniższych funkcji [metod] opisanych w rozdz. „Rozpoczęcie pracy z urządzeniem”, str. 14 (zob. rozdz. „Interfejsy w C oraz C++”, str. 12):

  - `open_len(...)`
    `[open(...)]`
  - `open_with_baudrate_len(...)`
    `[open_with_baudrate(...)]`

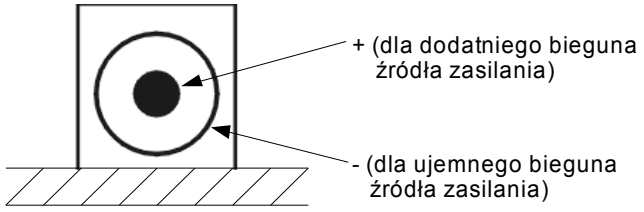
  Jeżeli jedna z powyższych funkcji [metod] otworzy urządzenie, to jest ono zasilane z portu COM komputera w czasie całej sesji systemu Windows (o ile nie zostało odłączone od komputera lub źródła zasilania).

- Z zewnętrznego źródła prądu o stałym napięciu o wartości mieszczącej się w zakresie od 9 V do 12 V podłączonego do gniazda SO: umożliwia zasilanie urządzenia LENDEVICE80RS232 oraz wykorzystanie 60 mA na wszystkich jego wyjściach.

  Schemat wyglądu gniazda SO przedstawiony jest na rysunku 3.

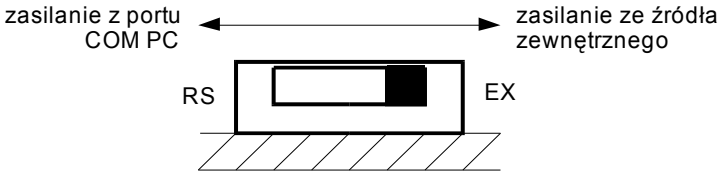
  W komplecie razem z urządzeniem dostarczany jest wtyk umożliwiający podłączenie zewnętrznego źródła zasilania do urządzenia.

  Nieprawidłowe podłączenie zasilania może uszkodzić urządzenie.



Rys. 3. *Schemat wyglądu gniazda zasilającego SO.*

W celu wybrania źródła zasilania urządzenia należy ustawić przełącznik SW w odpowiednim położeniu jak na rysunku 4.

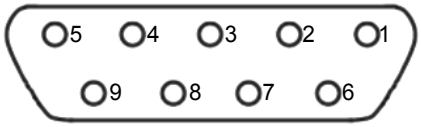


Rys. 4. *Schemat przełącznika SW kontrolującego źródło zasilania urządzenia.*

Jeżeli pobór prądu z wszystkich wyjść urządzenia przekroczy 60 mA, to urządzenie może ulec uszkodzeniu.

1.7. KOMUNIKACJA Z KOMPUTEREM

Urządzenie komunikuje się z komputerem za pomocą kabla podłączonego do portu COM komputera (RS232) i gniazda w urządzeniu LENDEVICE80RS232 oznaczonego symbolem RS, którego schematyczny wygląd i oznaczenia linii transmisyjnych przedstawiono na rysunku 5.



Rys. 5. *Gniazdo RS urządzenia LENDEVICE80RS232 służące do komunikacji z komputerem.*

Tabela 1 przedstawia funkcje poszczególnych linii transmisyjnych (pokazanych na rysunku 5) we wtyczce przyłączanej do gniazda RS urządzenia LENDEVICE80RS232.

Tab. 1. *Funkcje linii transmisyjnych wtyczki przyłączanej do gniazda RS urządzenia.*

Nr linii	Nazwa linii	Funkcja linii
1	DCD	Data Carrier Detected (wykryto nośną danych)
2	RD	Received Data (odbiór danych)
3	TD	Transmitted Data (transmisja danych)
4	DTR	Data Terminal Ready (terminal danych gotowy)
5	SG	Signal Ground (masa sygnałowa)
6	DSR	Data Set Ready (zbiór danych gotowy)
7	RTS	Request To Send (żądanie wysłania)
8	CTS	Clear To Send (zezwolenie na wysłanie)
9	RI	Ring Indicator (wskaźnik dzwonka), nie wykorzystane

Przedstawiony powyżej układ rozmieszczenia linii transmisyjnych jest typowy i bardzo często stosowany w komputerach. Zwykle podłączenie urządzenia wymaga wykorzystania kabla do komunikacji, w którym przewody są połączone „jeden do jednego". Jeżeli jednak w komputerze jest zastosowane inne rozmieszczenie linii transmisyjnych lub wykorzystane jest złącze DB25, to należy wykorzystać przejściówkę. Inne spotykane ułożenia linii transmisyjnych są opisane w „Załączniku B: Parametry wejść i wyjść na złączu RS”, str. 35.

Jeżeli układ UART komputera obsługuje tylko transmisję typu HANDSHAKING, to należy zastosować zasilanie zewnętrzne (zob. rozdz. „Zasilanie urządzenia”, str. 8).

Komunikacja z urządzeniem jest możliwa z prędkościami transmisji (bitów na sekundę [b/s]):

- 115200
- 57600
- 38400
- 19200
- 14400
- 9600
- 4800
- 2400
- 1200
- 600
- 300

Fabrycznie ustawiona prędkość startowa (startup baud rate) wynosi 9600 b/s (tzn. prędkość z jaką urządzenie pracuje zaraz po włączeniu zasilania). Prędkość tę można zmienić wywołując funkcję [metodę] opisaną w rozdz. „Zmiana startowej prędkości transmisji”, str. 19 (zob. rozdz. „Interfejsy w C oraz C++”, str. 12):

- `set_startup_baudrate_len(...)`
  `[set_startup_baudrate(...)]`

Nie jest zalecane częste wywoływanie tej funkcji [metody], ponieważ pamięć EEPROM urządzenia LENDEVICE80RS232 ma możliwość zapisu danych tylko ograniczoną liczbę razy (około 100000 razy).

Aktualną prędkość transmisji można zmienić (np. ze startowej 9600 b/s do szybszej 115200 b/s) wywołując funkcję [metodę] opisaną w rozdz. „Zmiana aktualnej prędkości transmisji”, str. 18 (zob. rozdz. „Interfejsy w C oraz C++”, str. 12):

- `change_baudrate_len(...)`
  `[change_baudrate(...)]`

Tą funkcję [metodę] można wywoływać dowolną ilość razy.

2. STEROWANIE URZĄDZENIEM

Urządzeniem LENDEVICE80RS232 można sterować za pomocą komputera z zainstalowanym systemem Windows 98/NT (wersje 98/Me/NT/2000/XP) wykorzystując dostarczone oprogramowanie. Istnieją trzy interfejsy, które dostarczają niezbędne funkcje do sterowania urządzeniem:

- Interfejs w języku **C** dostarcza funkcje zadeklarowane w pliku „lendev.h". Do projektu należy jeszcze dołączyć bibliotekę statyczną „lendev.lib".
- Interfejs w języku **C++** dostarcza klasę z wieloma metodami sterującymi urządzeniem, zdefiniowanymi w pliku „lendev.hpp". Do projektu należy jeszcze dołączyć bibliotekę statyczną „lendev.lib" oraz plik „lendev.h".
- Interfejs **.NET** dostarcza komponent `Lentec::LenDeviceRS232`, który umożliwia sterowanie urządzeniem oraz ustawianie jego właściwości. Do projektu należy dodać referencję do komponentu `Lentec::LenDeviceRS232` znajdującego się w pliku „lenNET.dll".

Plik „lendev.lib" umożliwia wykorzystanie funkcji eksportowanych z biblioteki „lendev.dll". „Załącznik H: Dodawanie bibliotek statycznych (*.lib) do projektu" na str. 38 przedstawia sposób dodania biblioteki statycznej (*.lib) do projektu w różnych środowiskach programistycznych. „Załącznik G: Dodawanie nowych skrótów do folderów przeszukiwanych w środowiskach programistycznych" na str. 38 przedstawia sposób dodania skrótów do folderów, które są przeszukiwane przez środowisko programistyczne w trakcie wyszukiwania plików, co bardzo ułatwia pisanie programów. W Załączniku I podane zostały dodatkowe informacje dla deweloperów o bibliotece „lendev.dll" i sterowniku „lenR232.exe". Po dołączeniu plików można przystąpić do programowania.

2.1. INFORMACJA O ZAMIESZCZONYCH PRZYKŁADACH

Wszystkie programy zamieszczone w dokumentacji służą pokazaniu zasad korzystania z oprogramowania sterującego urządzeniem LENDEVICE80RS232. Przedstawione zostały najprostsze programy działające w konsoli systemu Windows, jednak możliwe jest wykorzystanie oprogramowania w dowolnych projektach napisanych w języku C/C++. Programy były testowane na kompilatorach: Visual C++ 8.0, Visual C++ 7.0, Visual C++ 6.0, MinGW 3.4.

Opis wykorzystania komponentu .NET sterującego urządzeniem został podany w oddzielnej instrukcji.

2.2. INTERFEJSY W C ORAZ C++

Sterowanie urządzeniem odbywa się za pomocą funkcji zadeklarowanych w pliku „lendev.h" lub metod klasy `len::DeviceRS232` zdefiniowanych w pliku „lendev.hpp".

Przed przystąpieniem do programowania należy zdecydować się na jeden sposób programowania: korzystając z funkcji pobierających uchwyt `LENDEVICE` lub obiektu klasy `len::DeviceRS232`. W obu interfejsach znajdują się funkcje i metody identycznie

sterujące urządzeniem i różnią się nazwami, np. pomiar napięcia analogowego wykonywany jest po wywołaniu funkcji `analog_input_len(...)` lub metody `analog_input(...)` (dokładniej `len::DeviceRS232::analog_input(...)`). Dalej **w tekście metoda klasy `len::DeviceRS232` odpowiadająca funkcji z pliku „lendev.h" będzie podawana w nawiasach kwadratowych**, np.:

- `FLOAT analog_input_len(const LENDEVICE hdevice, const INT channel)` **`[`**`float analog_input(const int channel)`**`]`**

Większość operacji sterujących urządzeniem zostało zdublowanych w celu łatwiejszego ich wykorzystania w programach. Przykładowo w pliku „lendev.h" zadeklarowane zostały dwie funkcje zwracające wartość napięcia zmierzonego na wejściach analogowych:

- `FLOAT analog_input_len(const LENDEVICE hdevice, const INT channel)`
- `BOOL AnalogInputLen(const LENDEVICE hdevice, const INT channel, FLOAT* result_ptr)`

Pierwsza z powyżej wymienionych funkcji zwraca wartość zmierzonego napięcia. Informację na temat, czy operacja się powiodła można uzyskać wywołując funkcję (zob. rozdz. „Pobieranie informacji o ostatnim błędzie", str. 17):

- `INT get_last_error_len(const LENDEVICE hdevice)`

Druga z wymienionych funkcji `AnalogInputLen(...)` zwraca wynik powodzenia operacji:

- `TRUE` – gdy operacja się powiodła;
- `FALSE` – w przeciwnym razie.

Więcej informacji na temat dlaczego operacja się nie powiodła, można uzyskać wywołując funkcję `get_last_error_len(...)`.

Funkcję `analog_input_len(...)` wygodnie jest wykorzystać gdy jesteśmy pewni, że urządzenie jest podłączone do komputera, np.:

Przykład 1.

```
#include "lendev.h"
#include "stdio.h"

int main()
{
  LENDEVICE hdev = open_len(1); // zob. rozdz. 2.3
  if ( hdev != NO_LENDEVICE )   // chwile temu zostalo otwarte
  {                             // urzadzenie, wiec teraz tez jest
    printf("Wartosc napiecia na kanale 0: %.3f V\n",
      analog_input_len(hdev, AIN0));
    close_len(hdev);            // zob. rozdz. 2.4
  }
  else
    printf("Nie mozna polaczyc sie z urzadzeniem\n");
  system("PAUSE");
  return 0;
}
```

Funkcję `AnalogInputLen(...)` wygodnie wywoływać wówczas, gdy bardzo długo

korzystamy z urządzenia i w tym czasie może dojść do awarii (np. uszkodzenia kabla, odłączenia urządzenia itp.). Przykład 2 przedstawia program, w którym wykorzystano funkcję `AnalogInputLen(...)`.

Przykład 2.

```
#include "lendev.h"
#include "stdio.h"

int main()
{
  LENDEVICE hdev = open_len(1); // zob. rozdz. 2.3
  if ( hdev != NO_LENDEVICE )
  {
    FLOAT wartosc_napiecia;
    for (int nr_testu = 0; nr_testu < 10; ++nr_testu)
    {
      if ( AnalogInputLen(hdev, 0, &wartosc_napiecia) == TRUE )
      {
        // operacja zakonczona sukcesem
        printf("Wartosc napiecia na kanale 0: %.3f V.\n",
          wartosc_napiecia);
      }
      else
        printf("Nie mozna zmierzyc napiecia.\n");
      Sleep(2000);  // czekanie 2 sekundy
    }
    close_len(hdev); // zob. rozdz. 2.4
  }
  else
    printf("Nie mozna polaczyc sie z urzadzeniem.\n");
  system("PAUSE");
  return 0;
}
```

Podobny interfejs dostarczony został w klasie `len::DeviceRS232`. Przykładowo metoda:

- `float analog_input(const int channel)`

zwraca wartość zmierzonego napięcia, a metoda:

- `bool AnalogInput(const int channel, float* result_ptr)`

zwraca wynik powodzenia operacji.

2.3. ROZPOCZĘCIE PRACY Z URZĄDZENIEM

Przed pierwszym użyciem urządzenia należy otworzyć urządzenie oraz uzyskać uchwyt `LENDEVICE`. W tym celu należy wywołać jedną z funkcji:

- `LENDEVICE open_len(const INT port)`

  Parametr `port` oznacza numer portu COM, do którego podłączone jest urządzenie: `port` = 1 dla COM1, `port` = 2 dla COM2 itd., musi zachodzić: $1 \leq$ `port` ≤ 256.

  Funkcja otwiera połączenie z urządzeniem rozpoczynając transmisję danych z

prędkością 9600 b/s i jeżeli nie udało się połączyć z tą prędkością, próbuje się skomunikować z urządzeniem wykorzystując inne prędkości transmisji (jeżeli układ UART komputera umożliwia przesyłać i odbierać dane) w podanej kolejności: 115200, 57600, 38400, 19200, 14400, 9600, 4800, 2400, 1200, 600, 300 b/s.

Funkcja zwraca uchwyt typu `LENDEVICE`: gdy zwracana wartość jest różna od `NO_LENDEVICE`, oznacza to, że połączenie z urządzeniem zakończyło się pomyślnie.

Nowe urządzenia mają ustawioną startową prędkość transmisji równą 9600 b/s, ale można ją zmienić za pomocą funkcji `set_startup_baudrate_len(...)` (zob. rozdz. „Zmiana startowej prędkości transmisji", str. 19) lub zmienić aktualną prędkość transmisji za pomocą funkcji `change_baudrate_len(...)` (zob. rozdz. „Zmiana aktualnej prędkości transmisji", str. 18).

Jeżeli ustalona zostanie startowa prędkość transmisji 300 b/s wywołanie funkcji `open_len(...)` będzie trwać kilkanaście sekund. Dlatego warto skorzystać z następnej wymienionej funkcji.

- `LENDEVICE open_with_baudrate_len(const INT port, const BaudRate baudrate)`

  Parametr `port` oznacza numer portu COM, do którego podłączone jest urządzenie: `port` = 1 dla COM1, `port` = 2 dla COM2 itd., musi zachodzić: $1 \leq$ `port` ≤ 256.

  Funkcja otwiera połączenie z urządzeniem rozpoczynając transmisję danych z prędkością równą `baudrate`. W rozdziale „Zmiana aktualnej prędkości transmisji" na str. 18 podane zostały prędkości, z jakimi urządzenie może transmitować dane.

  Funkcja zwraca uchwyt typu `LENDEVICE`: gdy zwracana wartość jest różna od `NO_LENDEVICE`, oznacza to, że połączenie z urządzeniem zakończyło się pomyślnie.

Każde otwarte urządzenie musi zostać zamknięte (zob. rozdz. „Kończenie pracy z urządzeniem", str. 16).

W klasie `len::DeviceRS232` do otwierania urządzenia służą metody:

- `bool open(const int port)` – odpowiednik funkcji `open_len(...)`;
- `bool open_with_baudrate(const int port, const BaudRate baudrate)` – odpowiednik funkcji `open_with_baudrate_len(...)`.

Obie metody zwracają wynik powodzenia otwarcia urządzenia.

Przykłady 3 i 4 przedstawiają sposoby otwarcia i zamknięcia urządzenia.

Urządzenie można także otworzyć za pomocą konstruktorów zdefiniowanych w klasie `len::DeviceRS232`:

- `len::DeviceRS232:DeviceRS232(const int port)` – wywołuje metodę `open(...)`;
- `len::DeviceRS232:DeviceRS232(const int port, const BaudRate baudrate)` – wywołuje metodę `open_with_baudrate(...)`.

Przykład 3.

```
#include "lendev.h"
#include "stdio.h"

int main()
{
  LENDEVICE hdev = open_with_baudrate_len(1, BR_9600);
    // otwarcie  urzadzenia
  if ( hdev != NO_LENDEVICE )  // sprawdzenie, czy zostalo otwarte
  {
    printf("Urzadzenie zostalo otwarte\n");
    // wykonujemy prace
    // wywolujemy funkcje zadeklarowane w pliku "lendev.h"
    // i zawsze podajemy parametr hdev
    close_len(hdev);           // obowiazkowe zamkniecie urzadzenia
    // uchwyt hdev jest juz nieaktualny i nie mozna z niego korzystac
  }
  else
    printf("Nie mozna polaczyc sie z urzadzeniem\n");
  system("PAUSE");
  return 0;
}
```

Przykład 4.

```
#include "lendev.hpp"
#include <iostream>

int main()
{
  len::DeviceRS232 dev;  // obiekt umozliwiajacy sterowanie urzadzeniem
  if ( dev.open_with_baudrate(1, BR_115200) ) // startowa predkosc
                                              // transmisji = 115200 b/s
  {
    std::cout << "Otwarte urzadzenie\n";
    // wykonujemy prace
    // wywolujemy metody klasy len::DeviceRS232 zdefiniowane
    // w pliku "lendev.hpp"
    dev.close();  // opcjonalne zamkniecie urzadzenia
    // jesli nie zamkniemy urzadzenia, to zrobi to destruktor
    // klasy len::DeviceRS232::~DeviceRS232()
  }
  else
    std::cout << "Nie udalo sie polaczyc z urzadzeniem.\n";
  system("PAUSE");
  return 0;
}
```

2.4. KOŃCZENIE PRACY Z URZĄDZENIEM

Przed zakończeniem pracy z urządzeniem należy koniecznie zamknąć uchwyt typu `LENDEVICE` do urządzenia. W tym celu należy wywołać funkcję [metodę]:

- `BOOL close_len(const LENDEVICE hdevice)`
  `[bool close()]`

Funkcja [metoda] zwraca wynik powodzenia operacji. Metoda klasy `close()` nie musi być wywoływana jawnie, ponieważ robi to także destruktor klasy.

2.5. TESTOWANIE PODŁĄCZONEGO URZĄDZENIA

Do sprawdzenia, czy urządzenie jest podłączone do komputera służy funkcja [metoda]:

- `BOOL is_open_len(const LENDEVICE hdevice,`
  `const BOOL check_obligatorily)`
  `[bool is_open(const bool check_obligatorily = false)]`

Jeśli parametr `check_obligatorily` równa się `TRUE` [`true`], wywoływana jest funkcja [metoda] `check_device_len(...)` [`check_device(...)`]. Jeśli parametr `check_obligatorily` równa się `FALSE` [`false`], to funkcja [metoda] `check_device_len(...)` [`check_device(...)`] jest wywoływana tylko wówczas, gdy poprzednia instrukcja sterująca urządzeniem zakończyła się niepomyślnie. Jeśli ostatnia instrukcja sterująca urządzeniem zakończyła się pomyślnie, zwracana jest wartość `TRUE` [`true`]. Rozwiązanie takie umożliwia szybsze działanie programów w sytuacji gdy nie występują problemy z komunikacją pomiędzy PC a urządzeniem LENDEVICE80RS232.

Funkcja [metoda] `is_open_len(...)` [`is_open(...)`] zwraca wartość `TRUE` [`true`], gdy funkcja [metoda] `check_device_len(...)` [`check_device(...)`] zwróci wartość różną od `DEVICE_OFF`, w przeciwnym razie `FALSE` [`false`] (zob. rozdz. „Sprawdzanie urządzenia”, str. 21).

Jeśli nie można połączyć się z urządzeniem, to można wykonać operację resetowania urządzenia wywołując funkcję [metodę] opisaną w rozdz. „Resetowanie urządzenia”, str. 21:

- `reset_device_len(...)`
  `[bool reset_device()]`

2.6. POBIERANIE INFORMACJI O OSTATNIM BŁĘDZIE

Informację o ostatnim błędzie można otrzymać wywołując funkcję [metodę]:

- `INT get_last_error_len(const LENDEVICE hdevice)`
  `[int get_last_error()]`

Funkcja zwraca jedną z poniższych wartości zdefiniowanych w pliku „lendev.h”:

- `ERROR_SUCCESS_LEN` – nie wystąpił błąd;
- `ERROR_SYSTEM_PROBLEM_LEN` – system nie udostępnia operacji: występuje np. gdy próbuje się zmienić prędkość transmisji na taką, której nie udostępnia układ UART komputera (zob. rozdz. „Pobieranie informacji o porcie COM komputera”, str. 29);
- `ERROR_INVALID_PARAMETERS_LEN` – zła wartość parametru podanego do funkcji, np. gdy od urządzenia zażądamy podania wartości napięcia analogowego na 100 kanale, a taki kanał fizycznie nie istnieje w urządzeniu;
- `ERROR_COMMUNICATION_LEN` – błąd w komunikacji pomiędzy urządzeniem a

komputerem, może być spowodowany:

- niepodłączonym urządzeniem do odpowiedniego portu COM komputera;
- źle skonfigurowanym sposobem zasilania (zob. rozdz. „Zasilanie urządzenia”, str. 8);
- różnie ustawionymi prędkościami transmisji w urządzeniu oraz komputerze;

- `ERROR_INVALID_INSTRUCTION_LEN` – błąd wynikający z wywołania instrukcji niemożliwej do wykonania przez urządzenie (zob. rozdz. „Programowanie urządzenia”, str. 22);
- `ERROR_BAD_HANDLE_LEN` – niepoprawny uchwyt typu `LENDEVICE` do urządzenia.

2.7. ZMIANA AKTUALNEJ PRĘDKOŚCI TRANSMISJI

W celu zmiany aktualnej prędkości transmisji danych między komputerem a urządzeniem LENDEVICE80RS232 należy wywołać funkcję [metodę]:

◆
```
BOOL change_baudrate_len(const LENDEVICE hdevice,
    const BaudRate new_baudrate)
[bool change_baudrate(const BaudRate new_baudrate)]
```

Urządzenie umożliwia komunikację z prędkościami transmisji (b/s):

300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 57600, 115200

Parametr `new_baudrate` może mieć jedną z wartości:

- `BR_300`
- `BR_600`
- `BR_1200`
- `BR_2400`
- `BR_4800`
- `BR_9600`
- `BR_14400`
- `BR_19200`
- `BR_38400`
- `BR_57600`
- `BR_115200`

Funkcja [metoda] zwraca wynik powodzenia operacji.

Przed zmianą prędkości transmisji danych należy się upewnić, że układ UART komputera umożliwia transmisję z wybraną prędkością (zob. rozdz. „Pobieranie informacji o porcie COM komputera”, str. 29). Po zamknięciu urządzenia i ponownym otwarciu urządzenia aktualna prędkość transmisji równa jest startowej prędkości transmisji.

W celu zmiany prędkości transmisji zaleca się korzystać z funkcji [metody] `change_baudrate_len(...)` [`change_baudrate(...)`] niż z `set_startup_baudrate_len(...)` [`set_startup_baudrate(...)`].

Jeśli występują problemy ze zmianą aktualnej prędkości transmisji, należy zmienić ją za pomocą funkcji `set_startup_baudrate_len(...)` [`set_startup_baudrate(...)`]

(zob. rozdz. „Zmiana startowej prędkości transmisji”, str. 19), następnie odłączyć urządzenie od zasilania i ponownie włączyć urządzenie.

2.8. ZMIANA STARTOWEJ PRĘDKOŚCI TRANSMISJI

Startowa prędkość transmisji danych to prędkość, z jaką rozpoczyna komunikację urządzenie po jego włączeniu. Aby możliwe było rozpoznanie podłączonego urządzenia, komputer musi także transmitować dane z tą prędkością.

Fabrycznie nowe urządzenie ma ustawioną startową prędkość transmisji danych równą 9600 b/s, ale można ją zmienić wywołując funkcję [metodę]:

- `BOOL set_startup_baudrate_len(const LENDEVICE hdevice, const BaudRate new_baudrate)`
  `[bool set_startup_baudrate(const BaudRate new_baudrate)]`

Parametr `new_baudrate` może przyjmować takie same wartości jak w funkcji `change_baudrate_len(...)` (zob. rozdz. „Zmiana aktualnej prędkości transmisji”, str. 18).

Funkcja zwraca wynik powodzenia operacji.

Należy dodać, że aktualna prędkość transmisji się nie zmienia po wywołaniu funkcji [metody] `set_startup_baudrate_len(...)` [`set_startup_baudrate(...)`].

Inna startowa prędkość transmisji od 9600 b/s jest przydatna, gdy będzie wykorzystany długi przewód służący do komunikacji komputera z urządzeniem. Przykładowo, można zastosować przewód o maksymalnej długości 15 metrów i komunikować się z maksymalną prędkością transmisji 19200 b/s (typowo, ponieważ dokładnie standard tego nie określa). Wykorzystując pętlę prądową można się komunikować z urządzeniem z prędkością 9600 b/s na odległości 4000 metrów lub z prędkością 38400 b/s na odległości 500 metrów.

Nie jest zalecane częste wywoływanie funkcji [metody] `set_startup_baudrate_len` [`set_startup_baudrate`], ponieważ pamięć EEPROM urządzenia ma ograniczoną liczbę zapisu danych (około 100000 razy).

2.9. TRYB KOMUNIKACJI Z URZĄDZENIEM

Komputer może komunikować się z urządzeniem wykorzystując jeden z dwóch trybów komunikacji:

- `COMM_FAST_MODE` – szybki tryb komunikacji polega na przesyłaniu danych bezpośrednio do urządzenia bez sprawdzania ich poprawności; dzięki temu trybowi można prowadzić szybsze pomiary, ale nie mając gwarancji, że w trakcie przesyłania danych od urządzenia do komputera oraz od komputera do urządzenia nie pojawiły się błędy;
- `COMM_CONTROLLED_MODE` – tryb komunikacji z kontrolą przesyłu danych; polega na przesyłaniu każdej instrukcji z nadajnika, następnie przesłaniu otrzymanej instrukcji przez odbiornik ponownie do nadajnika w celu jej sprawdzenia i wysłaniu potwierdzenia poprawności przez nadajnik; nadajnikiem może być urządzenie LENDEVICE80RS232 lub komputer (podobnie odbiornik); jeśli wystąpił błąd w trakcie transmisji danych, to operacja jest ponawiana.

Rysunek 6 przedstawia porównanie przesyłania danych między nadajnikiem i odbiornikiem w dwóch trybach.

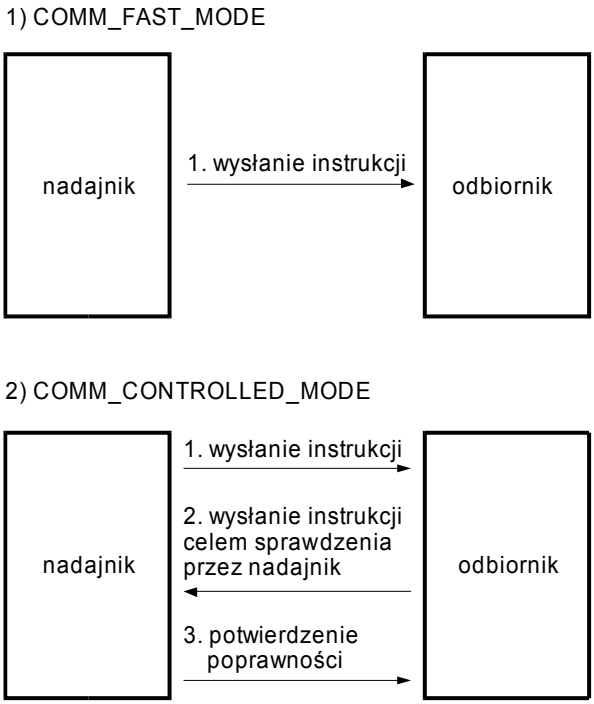
W celu zmiany trybu komunikacji należy wywołać funkcję [metodę]:

- `BOOL change_communication_mode_len(const LENDEVICE hdevice, const CommunicationMode new_mode)`
  `[bool change_communication_mode(const CommunicationMode new_mode)]`

Parametr `new_mode` jest nowym żądanym trybem komunikacji i może mieć jedną z dwu wartości:

- `COMM_FAST_MODE`
- `COMM_CONTROLLED_MODE`

Funkcja zwraca wynik powodzenia operacji.



Rys. 6. *Przedstawienie zasad funkcjonowania trybów przesyłania danych.*

Po włączeniu urządzenia LENDEVICE80RS232 włączany jest tryb `COMM_CONTROLLED_MODE`. Przykład 5 prezentuje wykorzystanie metody `change_communication_mode(...)`.

Przykład 5.

```
#include "lendev.hpp"
#include <iostream>

int main()
{
```

Przykład 5.

```
  len::DeviceRS232 dev;
  if ( dev.open(1) )
  {
    std::cout << "Otwarte urzadzenie\n";
    std::cout << "Nazwa urzadzenia: " << dev.info() << '\n';
      // zob. rozdz. 2.12
    std::cout << "Zmiana aktualnej predkosci transmisji na 115200 b/s:"
      << " powodzenie = " << std::boolalpha
      << dev.change_baudrate(BR_115200) << '\n';
    std::cout << "Zmiana trybu komunikacji na COMM_FAST_MODE: "
      << "powodzenie = " << std::boolalpha
      << dev.change_communication_mode(COMM_FAST_MODE) << '\n';
    std::cout << "Mozna wykonywac dalsze operacje...\n";
  }
  else
    std::cout << "Nie udalo sie polaczyc z urzadzeniem.\n";
  // destruktor len::DeviceRS232::~DeviceRS232() zamknie urzadzenie
  system("PAUSE");
  return 0;
}
```

2.10. SPRAWDZANIE URZĄDZENIA

W celu sprawdzenia urządzenia należy wywołać funkcję [metodę]:

- `DeviceMode check_device_len(const LENDEVICE hdevice, const INT how_many)`
- `[DeviceMode check_device(const int how_many = 10)]`

zwracającą jedną z wartości:

- `DEVICE_OFF` – nie można skomunikować się z urządzeniem;
- `DEVICE_ON` – urządzenie funkcjonuje poprawnie;
- `DEVICE_BAD_TRANSMISSION` – urządzenie jest podłączone ale wystąpiły błędy w trakcie transmisji danych;
- `DEVICE_WITHOUT_PROGRAM` – urządzenie jest podłączone, ale nie ma w nim załadowanego programu (zob. rozdz. „Programowanie urządzenia”, str. 22).

Parametr `how_many` decyduje, ile razy (maksymalnie) ma być wykonane testowanie urządzenia.

2.11. RESETOWANIE URZĄDZENIA

Urządzenie można zresetować. Resetowanie urządzenia polega na ustawieniu na wszystkich wyjściach cyfrowych stanu niskiego (LO), na wyjściach analogowych napięcie 0 V, wyzerowaniu licznika oraz przełączeniu się w tryb komunikacji `COMM_CONTROLLED_MODE`. Dodatkowo, jeżeli występuje problem z komunikacją między komputerem i urządzeniem, to komputer próbuje połączyć się z urządzeniem wykorzystując każdą z dostępnych prędkości transmisji i wówczas operacja resetowania

może potrwać kilkanaście sekund (maksymalnie).

W celu zresetowania urządzenia należy wywołać funkcję [metodę]:

- `BOOL reset_device_len(const LENDEVICE hdevice)`
  `[bool reset_device()]`

Funkcja zwraca wartość `TRUE` [`true`] jeżeli możliwa jest komunikacja z urządzeniem lub `FALSE` [`false`] w przeciwnym razie.

2.12. SPRAWDZANIE TYPU URZĄDZENIA

Oprogramowanie umożliwia korzystanie jednocześnie z kilku różnych urządzeń firmy Lentec Design wykorzystujących do transmisji złącze COM komputera. W celu sprawdzenia typu podłączonego urządzenia można wywołać jedną z dwóch funkcji [metod].

- `INT type_of_device_len(const LENDEVICE hdevice)`
  `[int type_of_device()]`
- `VOID info_len(const LENDEVICE hdevice, BYTE information[40])`
  `[std::string info()]`

Funkcja [metoda] `type_of_device_len(...)` [`type_of_device()`] zwraca jedną z wartości:

- `LENTECDEVICE_NOOPENED` = 0 – nie można się skomunikować z urządzeniem;
- `LENTECDEVICE80RS232` = 10 – urządzenie serii LENDEVICE80RS232.

Producent zastrzega sobie prawo dodania nowych zwracanych wartości przez tą funkcję [metodę] w nowych wersjach oprogramowania, jednak wartości liczbowe nie ulegną zmianie.

Druga funkcja [metoda] `info_len(...)` [`info()`] zwraca informacje o urządzeniu w formie tekstowej. Parametr `information` jest adresem do bufora na znaki ANSI. Zwracana informacja tekstowa w buforze zakończona jest znakiem `'\0'`.

2.13. PROGRAMOWANIE URZĄDZENIA

Producent może w przyszłości dostarczyć nowe oprogramowanie do urządzenia LENDEVICE80RS232 w celu zwiększenia liczby dostarczanych funkcji.

Każde nowe załadowane oprogramowanie będzie wymagało zainstalowania nowych sterowników w komputerze.

Programowanie urządzeń może się nie powieść i wówczas należy operację programowania powtórzyć. Bliższe informacje będą dostarczane wraz z nowym oprogramowaniem dostępnym na stronach internetowych producenta www.lentecdesign.com.

W celu załadowania programu należy wywołać funkcję [metodę]:

- `BOOL program_device_len(const LENDEVICE hdevice,`
  `const BYTE* file_ansi)`
  `[bool program_device(const std::string& file_ansi)]`

Parametr `file_ansi` jest adresem do znaków ANSI z zapisaną ścieżką dostępu do pliku

typu „*.lhx” dostarczonego przez producenta (na końcu łańcucha musi być znak '\0'). Funkcja [metoda] zwraca wynik powodzenia operacji.

2.14. POBIERANIE WARTOŚCI NAPIĘĆ ANALOGOWYCH

W celu zmierzenia wartości napięć analogowych na wejściach analogowych urządzenia (zob. rozdz. „Wejścia cyfrowe”, str. 6) należy wywołać jedną z funkcji [metod]:

- `FLOAT analog_input_len(const LENDEVICE hdevice, const INT channel)`
  `[float analog_input(const int channel)]`
- `BOOL AnalogInputLen(const LENDEVICE hdevice, const INT channel, FLOAT* result_ptr)`
  `[bool AnalogInput(const int channel, float* result_ptr)]`

Funkcja [metoda] `analog_input_len(...)` [`analog_input(...)`] zwraca wartość zmierzonego napięcia lub – 1, gdy wystąpił błąd. Funkcja [metoda] `AnalogInputLen(...)` [`AnalogInput(...)`] zwraca wartość powodzenia operacji odczytu, a wartość zmierzonego napięcia przypisuje zmiennej, której adres to `result_ptr`. Do wyboru kanału należy wykorzystać stałe: `AIN0`, `AIN1`, ..., `AIN6`.

Przykład 6 prezentuje wykonanie pomiarów napięć na wejściach analogowych.

2.15. STEROWANIE WYJŚCIAMI ANALOGOWYMI

W celu ustawienia napięcia na jednym z wyjść analogowych urządzenia (zob. rozdz. „Wyjścia analogowe”, str. 6) należy wywołać jedną z funkcji [metod]:

- `VOID analog_output_len(const LENDEVICE hdevice, const INT channel, const FLOAT value)`
  `[void analog_output(const int channel, const float value)]`
- `BOOL AnalogOutputLen(const LENDEVICE hdevice, const INT channel, const FLOAT value)`
  `[bool AnalogOutput(const int channel, const float value)]`

Funkcje [metody] ustawiają wartość napięcia `value` na wyjściu analogowym `channel` (`channel = AOUT0` lub `AOUT1`). Funkcja [metoda] `AnalogOutputLen(...)` [`AnalogOutput(...)`] zwraca wynik powodzenia operacji.

Przykład 6 prezentuje wykonanie ustawień napięć na wyjściach analogowych.

Przykład 6.

```
#include "lendev.hpp"
#include <iostream>

int main()
{
   len::DeviceRS232 dev;
   if ( dev.open(1) )
   {
      std::cout << "Otwarte urzadzenie\n";
      std::cout << "Nazwa urzadzenia: " << dev.info() << '\n';
      std::cout << "Zmiana aktualnej predkosci transmisji na 115200 b/s:"
```

Przykład 6.

```
      << " powodzenie = " << std::boolalpha
      << dev.change_baudrate(BR_115200) << '\n';
    std::cout << "Zmiana trybu komunikacji na COMM_FAST_MODE: "
      << "powodzenie = " << std::boolalpha
      << dev.change_communication_mode(COMM_FAST_MODE) << '\n';
    std::cout << "Pomiar napiec na wejsciach analogowych\n";
    for (int i = 0; i < 100; ++i)
    {
      printf("Pomiar %3d:  %.3f  %.3f  %.3f [V]\n", i+1,
        dev.analog_input(AIN0), dev.analog_input(AIN1),
        dev.analog_input(AIN2));
    }
    dev.analog_output(AOUT0, 2.5f);
    dev.analog_output(AOUT1, 1.0f);
  }
  else
    std::cout << "Nie udalo sie polaczyc z urzadzeniem.\n";
  system("PAUSE");
  return 0;
}
```

2.16. POBIERANIE STANÓW LOGICZNYCH NA WEJŚCIACH CYFROWYCH

W celu odczytania stanów logicznych na wejściach cyfrowych urządzenia (zob. rozdz. „Wyjścia cyfrowe”, str. 6) należy wywołać funkcję [metodę]:

- `DWORD digital_input_len(const LENDEVICE hdevice)`
  `[unsigned long digital_input()]`

która zwraca stan na poszczególnych wejściach cyfrowych. Na wejściu DIN0 jest stan wysoki, gdy operacja bitowej koniunkcji zwróconej wartości przez funkcję [metodę] `digital_input_len(...)` [`digital_input()`] i wartości `DIN0` jest różna od zera (0), tzn.:

```
if ( (digital_input_len(hdev) & DIN0) != 0 )
{
  // jest stan wysoki (HI) na wejsciu DIN0
}
else
{
  // jest stan niski (LO) na wejsciu DIN0
}
```

Można sprawdzić stany na kilku wejśiach cyfrowych jednocześnie, np.:

```
if ( (digital_input_len(hdev) & (DIN0 | DIN1 | DIN2) ) != 0 )
{
  // jest stan wysoki (HI) na wejsciach DIN0, DIN1 i DIN2
}
else
{
  // jest stan niski (LO) na jednym lub kilku wejsciach cyfrowych
```

```
  // DIN0, DIN1 lub DIN2
}
```

Inny sposób wykonania powyższej operacji, to wykorzystanie funkcji [metod]:

- `BOOL is_hi_digital_input_len(const LENDEVICE hdevice, const DWORD inputs)`
  `[bool is_hi_digital_input(const unsigned long inputs)]`
  `[template<int N> bool is_hi_digital_input(const std::bitset<N>& inputs)]`

np.:

```
if ( is_hi_digital_input_len(hdev, DIN0 | DIN1 | DIN2) != FALSE )
{
  // jest stan wysoki (HI) na wejsciu DIN0 i DIN1 i DIN2
}
else
{
  // jest stan niski (LO) na jednym lub kilku z wejsc DIN0
  // lub DIN1 lub DIN2
}
```

Funkcja [metoda] `is_hi_digital_input_len(...)` [`is_hi_digital_input(...)`] zwraca wartość różną od `FALSE` [`false`], gdy na wszystkich wejściach cyfrowych wymienionych w parametrze `inputs` występuje stan wysoki (HI), w przeciwnym razie zwraca wartość `FALSE` [`false`].

Podobnie działa funkcja [metody]:

- `BOOL is_lo_digital_input_len(const LENDEVICE hdevice, const DWORD inputs)`
  `[bool is_lo_digital_input(const unsigned long inputs)]`
  `[template<int N> bool is_lo_digital_input(const std::bitset<N>& inputs)]`

zwracając wartość różną od `FALSE` [`false`], gdy na wszystkich wejściach cyfrowych wymienionych w parametrze `inputs` występuje stan niski (LO), w przeciwnym razie zwraca wartość `FALSE` [`false`]

Podobnie działają funkcje [metody]:

- `BOOL DigitalInputLen(const LENDEVICE hdevice, DWORD* result_ptr)`
  `[bool DigitalInput(unsigned long* result_ptr)]`
- `BOOL IsHiDigitalInputLen(const LENDEVICE hdevice, const DWORD inputs, BOOL* result_ptr)`
  `[bool IsHiDigitalInput(const unsigned long inputs, bool* result_ptr)]`
  `[template<int N> bool IsHiDigitalInput(const std::bitset<N>& inputs, bool* result_ptr)]`
- `BOOL IsLoDigitalInputLen(const LENDEVICE hdevice, const DWORD inputs, BOOL* result_ptr)`
  `[bool IsLoDigitalInput(const unsigned long inputs, bool* result_ptr)]`
  `[template<int N> bool IsLoDigitalInput(const std::bitset<N>& inputs, bool* result_ptr)]`

zwracając wynik powodzenia operacji, a informację o stanach logicznych na wejściach

cyfrowych urządzenia przypisują pod zmienną, której adres to `result_ptr`.

Przykład 7 prezentuje odczytanie stanów logicznych na wejściach cyfrowych.

2.17. STEROWANIE WYJŚCIAMI CYFROWYMI

W celu ustawienia wyjść cyfrowych urządzenia (zob. rozdz. „Wyjścia cyfrowe”, str. 6) należy wywołać funkcję [metody]:

- `VOID digital_output_len(const LENDEVICE hdevice, const DWORD outputs)`
  `[void digital_output(const unsigned long outputs)]`
  `[template<int N> void digital_output(const std::bitset<N>& outputs)]`

Parametr `outputs` steruje stanami wyjść cyfrowych. Wartość `DOUT0` ustawia stan wysoki (HI) na wyjściu cyfrowym urządzenia DOUT0, `DOUT1` na wyjściu cyfrowym urządzenia DOUT1 itd. Na niewymienionych wyjściach cyfrowych ustawiany jest stan niski (LO). W celu ustawienia stanu wysokiego na wyjściach DOUT0 i DOUT1, a na pozostałych wyjściach cyfrowych urządzenia stan niski, należy wykonać operację:

```
digital_output_len(hdev, DOUT0 | DOUT1)
```

W celu ustawienia wybranych wyjść cyfrowych w stanie wysokim (HI) należy wywołać funkcję [metody]:

- `VOID hi_digital_output_len(const LENDEVICE hdevice, const DWORD outputs)`
  `[void hi_digital_output(const unsigned long outputs)]`
  `[template<int N> void hi_digital_output(const std::bitset<N>& outputs)]`

i żądane numery wyjść podać w parametrze `outputs`.

W celu ustawienia wybranych wyjść cyfrowych w stanie niskim (LO) należy wywołać funkcję [metody]:

- `VOID lo_digital_output_len(const LENDEVICE hdevice, const DWORD outputs)`
  `[void lo_digital_output(const unsigned long outputs)]`
  `[template<int N> void lo_digital_output(const std::bitset<N>& outputs)]`

i żądane numery wyjść podać w parametrze `outputs`.

W klasie `len::DeviceRS232` dodano metody pobierające parametr typu `std::bitset<int>`, gdzie numer bitu obiektu typu `std::bitset<int>` odpowiada numerowi wejścia / wyjścia cyfrowego urządzenia.

W plikach „lendev.h” oraz „lendev.hpp” zostały zadeklarowane [zdefiniowane] funkcje [metody] zwracające wynik powodzenia operacji.

- `BOOL DigitalOutputLen(const LENDEVICE hdevice, const DWORD outputs)`
  `[bool DigitalOutput(const unsigned long outputs)]`
  `[template<int N> bool DigitalOutput(const std::bitset<N>& outputs)]`

- BOOL HiDigitalOutputLen(const LENDEVICE hdevice,
  const DWORD outputs)
  [bool HiDigitalOutput(const unsigned long outputs)]
  [template<int N> bool HiDigitalOutput(
  const std::bitset<N>& outputs)]
- BOOL LoDigitalOutputLen(const LENDEVICE hdevice,
  const DWORD outputs)
  [bool LoDigitalOutput(const unsigned long outputs)]
  [template<int N> bool LoDigitalOutput(
  const std::bitset<N>& outputs)]

Przykład 7 prezentuje sterowanie wyjściami cyfrowymi urządzenia.

Przykład 7.

```
#include "lendev.hpp"
#include <iostream>
#include <iomanip>
#include <sstream>

len::DeviceRS232 dev;
using namespace std;

int main()
{
  if ( dev.open(1) )
  {
    cout << "Otwarte urzadzenie\n";
    // testowanie wejsc cyfrowych
    unsigned long inputs = dev.digital_input();
    cout << "Wejscia cyfrowe: wartosc zwracana = " << inputs << '\n';
    cout << "Stany na poszczegolnych wejsciach: "
      << bitset<8>(inputs) << '\n';
    cout << "Poszczegolne stany na wejsciach cyfrowych:\n";
    for (int nr_wejscia_cyfrowego = 0; nr_wejscia_cyfrowego < 8;
      ++nr_wejscia_cyfrowego)
    {
      cout << '\t' << "DIN" << nr_wejscia_cyfrowego << " = "
        << ( (inputs & (1ul << nr_wejscia_cyfrowego)) ? "HI" : "LO")
        << '\n';
    }
    // sprawdzenie, czy na wejsciach cyfrowych DIN3 i DIN7 wystepuje
    // stan niski (LO)
    if ( dev.is_lo_digital_input(DIN3 | DIN7) )
    {
      cout << "Na wejsciach DIN3 i DIN7 wystepuje stan niski\n";
    }

    // sterowanie wyjsciami cyfrowymi
    // ustawienie stanu wysokiego (HI) na wyjsciach DOUT7, DOUT3, DOUT1
    // a na pozostalych (czyli na DOUT6, DOUT5, DOUT4, DOUT2, DOUT0)
    // stan niski (LO)
    dev.digital_output(bitset<8>(string("10001010")));

    // ustawia w stanie niskim (LO) wyjscie DOUT7 i DOUT1 urzadzenia
    // a pozostale wyjscia pozostaja bez zmian
    dev.lo_digital_output(DOUT7 | DOUT1);
```

Przykład 7.

```
    // ustawia stan wysoki (HI) na wyjsciach DOUT7 i DOUT3 urzadzenia
    // a stany pozostalych wyjsc pozostaja bez zmian
    dev.hi_digital_output(bitset<8>(string("10001000")));
    // zastosowanie metod zwracajacych wynik powodzenia operacji
    for (int proba = 0; proba < 20; ++proba)
    {
      bool wynik;
      if( dev.IsHiDigitalInput(bitset<8>(string("00100000")),&wynik) )
      {
        ostringstream s;
        s << "Proba nr " << proba + 1 << ": ";
        cout << setiosflags(ios_base::left) << setw(13)
          << s.str() << resetiosflags(ios_base::adjustfield)
          << "Na wejsciu cyfrowym DOUT5 jest stan "
          << (wynik ? "HI" : "LO") << '\n';
      }
      else
        cout << "Operacja polaczenia z urzadzeniem zakonczyla sie"
          " niepowodzeniem. "
          << "Blad numer " << dev.get_last_error() << '\n';
      Sleep(1000);
    }
  }
  else
    cout << "Nie udalo sie polaczyc z urzadzeniem.\n";
  system("PAUSE");
  return 0;
}
```

2.18. OBSŁUGA LICZNIKA

W celu odczytania liczby impulsów zarejestrowanych przez licznik urządzenia (zob. rozdz. „Wejście licznika”, str. 7) należy wywołać funkcję [metodę]:

- `DWORD get_counter_len(const LENDEVICE hdevice, const INT number)`
  `[unsigned long get_counter(const int number)]`

Funkcja pobiera parametr `number` będący numerem licznika (stałą `L0`).

W celu wyzerowania licznika o numerze `number` należy wywołać funkcję [metodę]:

- `VOID clear_counter_len(const LENDEVICE hdevice, const INT number)`
  `[void len::DeviceRS232::clear_counter(const int number)]`

Funkcja [metoda]:

- `BOOL GetCounterLen(const LENDEVICE hdevice, const INT number,`
  `DWORD* result_ptr)`
  `[bool GetCounter(const int number, unsigned long* result_ptr)]`

zwraca wynik powodzenia operacji, a liczbę zarejestrowanych impulsów zapisuje w zmiennej o adresie `result_ptr`.

Funkcja [metoda]:

- `BOOL ClearCounterLen(const LENDEVICE hdevice, const INT number)`
  `[bool ClearCounter(const int number)]`

zeruje licznik i zwraca wynik powodzenia operacji.

2.19. POBIERANIE NUMERU SERYJNEGO URZĄDZENIA

Każde urządzenie ma przypisany unikalny numer seryjny będący wartością typu `DWORD` (zajmuje 4 bajty).

W celu odczytania numeru seryjnego urządzenia należy wywołać funkcję [metodę]:

- `DWORD get_serial_number_len(const LENDEVICE hdevice, const INT number)`
  `[unsigned long get_serial_number()]`

zwracającą numer seryjny urządzenia. Można wykorzystać także funkcję [metodę]:

- `BOOL GetSerialNumberLen(const LENDEVICE hdevice, DWORD* result_ptr)`
  `[bool GetSerialNumber(unsigned long* result_ptr)]`

zwracającą wynik powodzenia operacji, a wartość numeru seryjnego zapisywana jest w zmiennej o adresie `result_ptr`.

2.20. POBIERANIE INFORMACJI O PORCIE COM KOMPUTERA

Przed wykorzystaniem urządzenia można sprawdzić parametry portów COM komputera w celu upewnienia się, czy możliwa jest komunikacja z urządzeniem oraz jakie parametry transmisji są dozwolone (np. maksymalna prędkość transmisji).

W celu pobrania informacji o porcie COM komputera należy wywołać funkcję zadeklarowaną w pliku „lendev.h”:

- `BOOL check_accessibility_com_len(const INT port, INT* problems_ptr, BaudRate* max_baudrate_ptr, INT* number_of_available_baudrate_ptr, BOOL table_of_available_baudrate[11])`

lub w pliku „lendev.hpp”:

- `bool check_accessibility_com(const int port, int *const problems_ptr, BaudRate *const max_baud_rate_ptr, int *const number_of_available_baud_rate_ptr, bool table_of_available_baud_rate[11])`

Parametr `port` oznacza numer portu COM: `port` = 1 oznacza COM1, `port` = 2 oznacza COM2 itd., musi zachodzić: $1 \leq$ `port` ≤ 256.

Do zmiennej znajdującej się pod adresem `problems_ptr` zapisywana jest informacja o problemach:

- `HARDWARE_NO_OPERATED_9600_BAUDRATE`: oznacza, że układ UART komputera nie obsługuje fabrycznie ustawionej startowej prędkości transmisji 9600 b/s i w celu skomunikowania się z urządzeniem należy zmienić jego prędkość startową (np. wykorzystując drugi komputer umożliwiający komunikację z tą prędkością);
- `HARDWARE_ONLY_WITH_EXTERNAL_SOURCE`: oznacza, że układ UART komputera

umożliwia tylko komunikację typu handshaking i nie można wykorzystać linii RTS i DTR do zasilania urządzenia LENDEVICE80RS232 z portu COM komputera; urządzenie może wówczas pracować tylko wykorzystując zasilanie zewnętrzne (zob. rozdz. „Zasilanie urządzenia”, str. 8);

- `HARDWARE_CRITICAL_ERROR`: oznacza, że układ UART uniemożliwia współpracę z urządzeniem (np. nie umożliwia wysyłania 8 bitów danych); ten błąd występuje bardzo rzadko.

Jeśli nie występują żadne problemy, wówczas zmienna pod adresem `problems_ptr` ma wartość `HARDWARE_NO_PROBLEM` równą 0. Jeśli występują problemy, to można je zidentyfikować wykonując operację bitowej koniunkcji (zob. przykład 8).

W zmiennej pod adresem `max_baud_rate_ptr` zapisywana jest wartość maksymalnej prędkości z jaką może komunikować się układ UART komputera.

W zmiennej pod adresem `number_of_available_baud_rate_ptr` zapisywana jest liczba dostępnych prędkości transmisji, z jakimi może komunikować się układ UART komputera i urządzenie. Maksymalna wartość tej wielkości wynosi 11.

W tablicy `table_of_available_baudrate` podawane są prędkości transmisji udostępniane przez układ UART komputera i które mogą zostać wykorzystane do komunikacji z urządzeniem LENDEVICE80RS232. W celu sprawdzenia, czy przykładowo jest udostępniona prędkość transmisji 19200 b/s należy wykonać operację (dane zostały otrzymane po wywołaniu funkcji `check_accessibility_com_len(...)`):

```
if ( table_of_available_baudrate[BR_19200] != FALSE )
{
  // jest udostępniona prędkość  19200 b/s
}
```

Funkcje `check_accessibility_com_len` oraz `len::check_accessibility_com` zwracają wynik powodzenia operacji: jeśli zwrócona zostanie wartość `FALSE` lub odpowiednio `false`, oznacza to, że system nie udzielił informacji (np. z powodu niemożliwości pobrania uchwytu portu COM komputera) i przekazane dane nie są poprawne.

Przykład 8.

```
#include "lendev.hpp"
#include <iostream>

using namespace std;

int main()
{
  int port = 1;
  cout << "Podaj numer portu COM, ktory ma zostac sprawdzony: ";
  cin >> port;

  int problemy;
  BaudRate maksymalna_dostepna_predkosc;
  int liczba_dostepnych_predkosci;
  bool tablica_dostepnych_predkosci[BR_MAX_AVAILABLE];
```

Przykład 8.

```
  if ( len::check_accessibility_com(port, &problemy,
    &maksymalna_dostepna_predkosc, &liczba_dostepnych_predkosci,
    &tablica_dostepnych_predkosci[0]) )
  {
    if ( (problemy & HARDWARE_CRITICAL_ERROR) != 0 )
      cout << "Uklad UART komputera nie umozliwia wspolpracy z”
        << “urzadzeniem.\n";
    else
    {
      if ( problemy != HARDWARE_NO_PROBLEM )
      {
        cout << "Problemy:\n";
        if ( (problemy & HARDWARE_NO_OPERATED_9600_BAUDRATE) != 0 )
          cout << "Uklad UART komputera nie obsluguje predkosci "
            << "9600 bitow / sekunde.\n";
        if ( (problemy & HARDWARE_ONLY_WITH_EXTERNAL_SOURCE) != 0 )
          cout << "Uklad UART komputera nie umozliwia zasilania "
            << "urzadzenia z portu COM komputera.\n";
      }
      const long predkosci[11] = { 300, 600, 1200, 2400, 4800, 9600,
        14400, 19200, 38400, 57600, 115200 };
      cout << "Liczba dostepnych predkosci transmisji = "
        << liczba_dostepnych_predkosci << " z 11-tu.\n";
      cout << "Maksymalna dostepna predkosc transmisji = "
        << predkosci[maksymalna_dostepna_predkosc]
        << " bitow / sekunde.\n";
      cout << "Ponadto uklad UART komputera umozliwia polaczenia z "
        << "predkosciami transmisji:\n";
      for (int i = 0; i < maksymalna_dostepna_predkosc; ++i)
      {
        if ( tablica_dostepnych_predkosci[i] )
          cout << "\t - " << predkosci[i] << " bitow / sekunde\n";
      }
    }
  }
  else
    cout << "Nie mozna sprawdzic wlasciwosci portu COM" << port
      << ".\n";
  system("PAUSE");
  return 0;
}
```

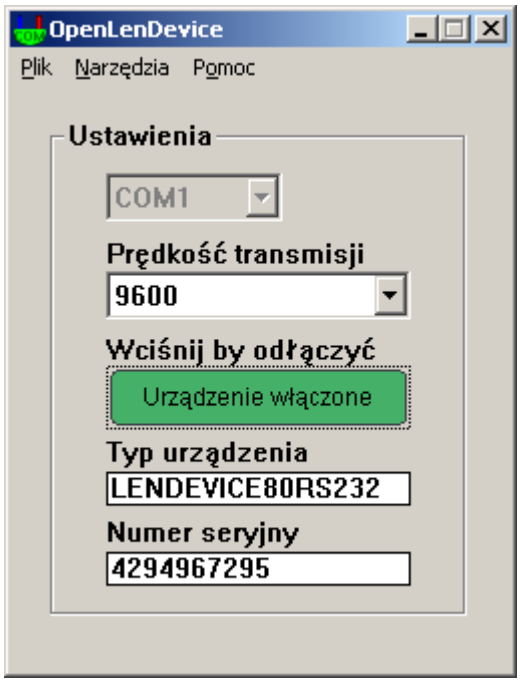
2.21. PROGRAM OPENLENDEVICE

Program OpenLenDevice dostarczany wraz z oprogramowaniem na płycie CD służy do włączania zasilania dla urządzenia LENDEVICE80RS232 i pokazany jest na rysunku 7.

Procedura podłączania urządzenia do komputera:

1. Uruchomić program OpenLenDevice.
2. Dokonać ustawień dotyczących numeru portu COM komputera, do którego podłączone jest urządzenie oraz startowej prędkości transmisji (fabrycznie ustawiona prędkość wynosi 9600 bitów / sekundę).

3. Podłączyć urządzenie LENDEVICE80RS232 do wybranego portu COM komputera za pomocą kabla.
4. Nacisnąć przycisk w programie OpenLenDevice łączący urządzenie z komputerem.
5. Jeśli polecenia 1 – 4 zostały wykonane poprawnie, przycisk powinien mieć kolor zielony (jak na rysunku 7) i w urządzeniu powinna zapalić się dioda LED.
6. Zamknąć program OpenLenDevice. Od tej chwili urządzenie będzie zasilane w czasie trwania całej sesji Windows (zob. rozdz. „Zasilanie urządzenia”, str. 8).



***Rys. 7. P**rogram OpenLenDevice.*

Zapalona dioda LED w urządzeniu sygnalizuje, że urządzenie jest zasilane.

Do urządzenia można podłączać inne urządzenia tylko wtedy, gdy urządzenie jest zasilane (czyli świeci się dioda LED).

Jeśli urządzenie wykorzystuje zasilanie zewnętrzne, to nie jest konieczne korzystanie z tego programu. Wówczas można podłączyć do urządzenia LENDEVICE80RS232 zewnętrzne układy elektryczne nawet wtedy, gdy nie ma połączenia z komputerem (zob. rozdz. „Zasilanie urządzenia”, str. 8).

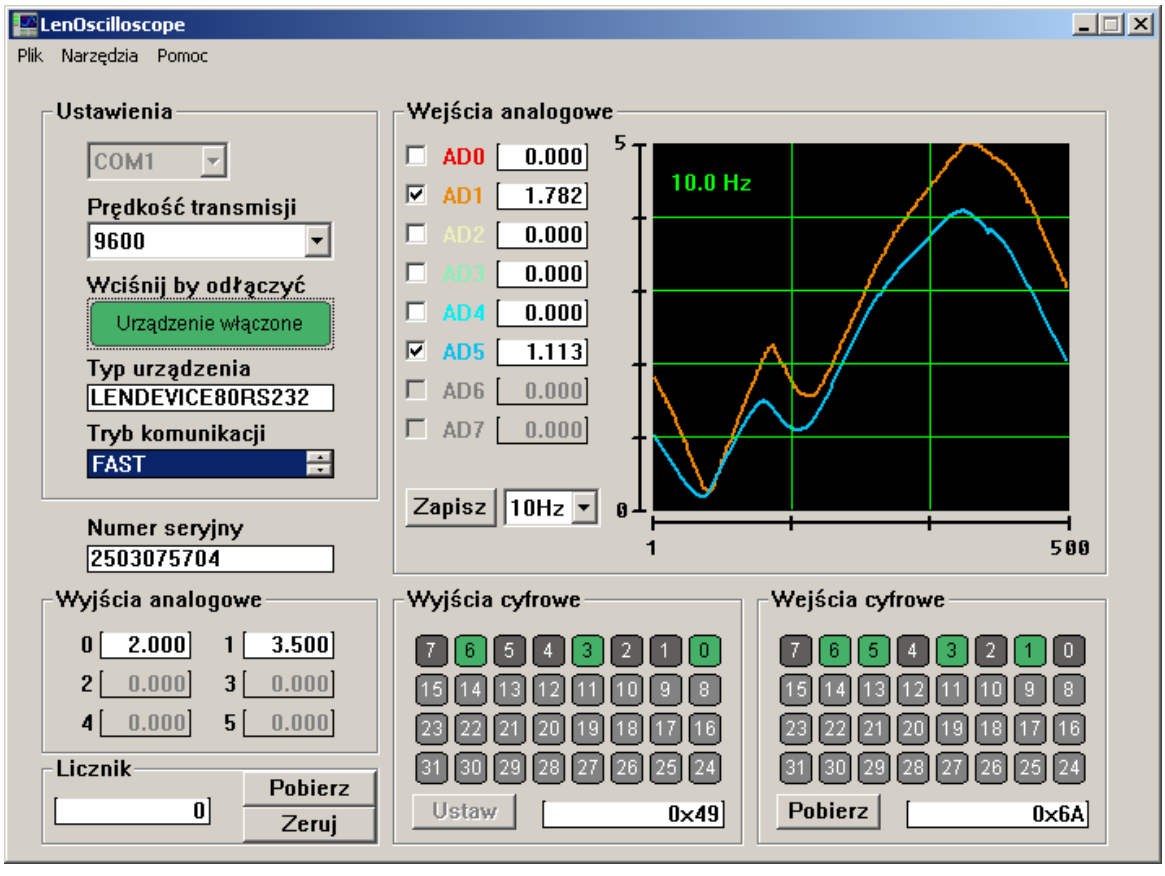
W przypadku problemów z włączaniem zasilania zob. rozdz. „Zasilanie urządzenia”, str. 8.

Program jest dwujęzyczny (zmiana języka: Narzędzia → Opcje). Program umożliwia zmianę startowej prędkości transmisji urządzenia LENDEVICE80RS232 (Narzędzia → Ustaw startową prędkość transmisji) oraz załadowanie nowego programu do mikrokontrolera urządzenia (zob. rozdz. „Programowanie urządzenia”, str. 22).

2.22. PROGRAM LENOSCILLOSCOPE

Program LenOscilloscope (rysunek 8) dostarczany wraz z oprogramowaniem na płycie CD służy do sterowania urządzeniem. Program posiada funkcje sterowania wyjściami analogowymi oraz cyfrowymi, rejestrowania napięcia na wejściach analogowych i cyfrowych , pobierania liczby zliczonych impulsów przez licznik. Zmierzone napięcia analogowe są prezentowane na wykresie i ich wartości mogą zostać zapisane do pliku typu „*.txt”.

W programie można zmieniać: liczbę punktów wyświetlanych na wykresie, sposób sterowania wyjściami cyfrowymi, sposób wyświetlania wartości w oknach edycyjnych (format dziesiętny i heksadecymalny) oraz zmienić język (Narzędzia → Opcje). Program umożliwia zmianę startowej prędkości transmisji urządzenia LENDEVICE80RS232 (Narzędzia → Ustaw startową prędkość transmisji) oraz załadowanie nowego programu do mikrokontrolera urządzenia (zob. rozdz. „Programowanie urządzenia”, str. 22).



Rys. 8. *Program LenOscilloscope.*

ZAŁĄCZNIKI

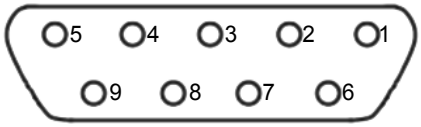
ZAŁĄCZNIK A: STOSOWANE OZNACZENIA

GND – masa urządzenia

VCC – napięcie zasilające

VDD – typowo 4,997 V ≈ 5 V

ZAŁĄCZNIK B: PARAMETRY WEJŚĆ I WYJŚĆ NA ZŁĄCZU RS



Rys. B1. *Gniazdo urządzenia LENDEVICE80RS232 służące do komunikacji z komputerem.*

Tab. B1. *Parametry elektryczne wejść i wyjść na złączu RS urządzenia LENDEVICE80RS232.*

Nr linii	Nazwa linii w porcie COM komputera	Funkcja linii w urządzeniu	Parametry elektryczne: zasilanie ze złącza RS	Parametry elektryczne: zasilanie ze złącza SO
1	DCD	wyjście	nie używane	napięcie jak na DTR
2	RD	wyjście	Max ±14 V (1)	Max ±14 V (3)
3	TD	wejście	Max ±14 V	Max ±14 V
4	DTR	wejście	Max ±14 V (2)	Max ±14 V
5	SG	poziom odniesienia	GND, 0 V	GND, 0 V
6	DSR	wyjście	nie używane	napięcie jak na DTR
7	RTS	wejście	Max ±14 V (2)	nie używane
8	CTS	wyjście	nie używane	napięcie jak na DTR
9	RI	nie używane	–	–

(1) Napięcie wyjściowe nie większe od napięcia na wejściach DTR i RTS
(2) Napięcia na wejściach DTR i RTS nie mogą się różnic o więcej niż 0.1 V. Przepływający prąd nie może być większy niż 50 mA.
(3) Nie większe niż napięcie zasilania.

Przekroczenie maksymalnych wartości napięć może zniszczyć urządzenie. Zwieranie wyjść do masy urządzenia lub innych źródeł napięciowych może zniszczyć urządzenie.

Inne spotykane rozmieszczenia linii w gniazdach DB9 i DB25 portów COM przedstawia tabela B2. W celu umożliwienia komunikacji urządzenia z komputerem należy wykorzystać urządzenie zamieniające kolejność linii transmisyjnych.

Tab. B2. *Stosowane rozmieszczenia linii w gniazdach DB9 i DB25 portów COM komputerów.*

Nazwa linii w porcie COM komputera	DB9	DB9	DB25	DB25
DCD	1	5	8	15
RD	3	4	3	5
TD	5	3	2	3
DTR	7	2	20	14
SG	9	1	7	13
DSR	2	9	6	11
RTS	4	8	4	7
CTS	6	7	5	9
RI	8	6	22	18

ZAŁĄCZNIK C: PARAMETRY WEJŚĆ ANALOGOWYCH

Tab. C1. *Parametry wejść analogowych oraz przetwornika analogowo – cyfrowego.*

Parametr	Wartość (zakres)
maksymalne napięcie wejściowe	od GND – 0.5 V do VDD + 0.5 V
rozdzielczość	10 bitów
błąd	typowo 1 LSB
zakres mierzonych napięć wejściowych	od GND do (1023/1024) VDD
maksymalna rezystancja wyjściowa źródła mierzonego napięcia	10 kΩ

ZAŁĄCZNIK D: PARAMETRY WYJŚĆ ANALOGOWYCH

Tab. D1. *Parametry elektryczne wyjść analogowych.*

Parametr	Wartość (zakres)
napięcie wyjściowe	typowo od GND + 5 mV do VDD – 5 mV
rozdzielczość	10 bitów
błąd	typowo 1 LSB
typowa rezystancja wyjściowa	50 Ω
maksymalny pobór prądu z wyjścia	±10 mA
typowy czas ustalania się napięcia	50 ms

ZAŁĄCZNIK E: PARAMETRY WEJŚĆ CYFROWYCH ORAZ LICZNIKA

Tab. E1. *Parametry elektryczne wejść cyfrowych oraz licznika.*	
Parametr	Wartość (zakres)
maksymalne napięcie wejściowe	od GND – 0.5 V do VDD + 0.5 V
maksymalne napięcie wejściowe dla stanu niskiego $V_{IN\ LOW}$	1.3 V
minimalne napięcie wejściowe dla stanu wysokiego $V_{IN\ HIGH}$	3.7 V
rezystancja wejściowa	typowo 1 MΩ

ZAŁĄCZNIK F: PARAMETRY WYJŚĆ CYFROWYCH

Tab. F1. *Parametry elektryczne wyjść cyfrowych*

Parametr	Wartość (zakres)
maksymalny przepływ prądu przez wyjście	±5 mA
wartość napięcia na wyjściu dla stanu wysokiego	od 3.8 V do VDD
wartość napięcia na wyjściu dla stanu niskiego	od GND do 0.5 V

ZAŁĄCZNIK G: DODAWANIE NOWYCH SKRÓTÓW DO FOLDERÓW PRZESZUKIWANYCH W ŚRODOWISKACH PROGRAMISTYCZNYCH

Visual C++ 2005	Options → Projects and solutions → VC++ directories: pliki typu *.lib → Library files pliki typu *.h, *.hpp → Include files
Visual C++ 6.0	Options → Directiories: pliki typu *.lib → Library files pliki typu *.h, *.hpp → Include files
Dev – C++ 4.9.2.2	Opcje kompilatora → Katalogi: pliki typu *.lib → Biblioteki pliki typu *.h, *.hpp → pliki nagłówkowe C++
Code::Blocks v1.0	Settings → Compiler Settings → Directories: pliki typu *.lib → Linker pliki typu *.h, *.hpp → Compiler

Po poprawnym utworzeniu skrótów do przeszukiwanych folderów, wszystkie pliki znajdujące się w tych katalogach można dołączyć bez podawania pełnej ścieżki np.:

```
// zamiast #include ”C:\Program Files\Lentec\lendev.hpp”
// można napisać
#include ”lendev.hpp”
```

ZAŁĄCZNIK H: DODAWANIE BIBLIOTEK STATYCZNYCH (*.LIB) DO PROJEKTU

Visual C++ 2005	● Project → Properties → Linker → Input → Additional Dependencies → wpisać nazwę pliku (najlepiej z pełną ścieżką dostępu między znakami cudzysłowu ” ”) ● W oknie „Solution Explorer” → Add → Existing Item... → wybrać plik *.lib (może być konieczne ustawienie „All Files”)
Visual C++ 6.0	● Project → Settings → Link → Zakładka General → Object/Library modules → wpisać nazwę pliku (najlepiej z pełną ścieżką dostępu między znakami cudzysłowu ” ”) ● W oknie „FileView” → Add File to Project → ustawić typ pliku: Library Files → wybrać plik *.lib
Dev – C++ 4.9.2.2	● Projekt → Opcje projektu → Parametry → Konsolidator → wciskamy klawisz „Dodaj plik” i wyszukujemy plik
Code::Blocks v1.0	● Project → „Project's Build Options → Zakładka „Linker” → Nacisnąć przycisk „Add” i wybrać bibliotekę.

ZAŁĄCZNIK I: INFORMACJE DLA DEWELOPERÓW

Chcąc połączyć się z urządzeniem niezbędna jest biblioteka „lendev.dll" oraz sterownik „lenR232.exe". Oba pliki muszą znajdować się w tym samym katalogu. Typowo pliki te instalowane są w katalogu systemowym (funkcja API GetSystemDirectory() zwraca adres katalogu systemowego).

Biblioteka „lendev.dll" eksportuje wymienione poniżej funkcje z konwencją `__stdcall`:

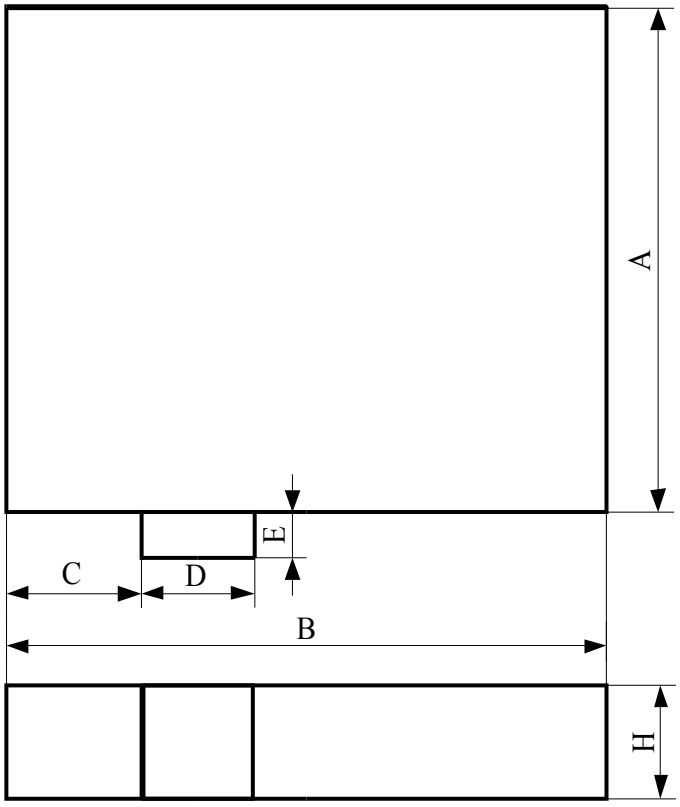
- `open_len`
- `open_with_baudrate_len`
- `close_len`
- `is_open_len`
- `get_last_error_len`
- `change_baudrate_len`
- `set_startup_baudrate_len`
- `change_communication_mode_len`
- `reset_device_len`
- `info_len`
- `check_device_len`
- `program_device_len`
- `digital_output_len`
- `hi_digital_output_len`
- `lo_digital_output_len`
- `digital_input_len`
- `is_hi_digital_input_len`
- `is_lo_digital_input_len`
- `analog_input_len`
- `analog_output_len`
- `get_counter_len`
- `clear_counter_len`
- `get_serial_number_len`
- `check_accessibility_com_len`
- `DigitalOutputLen`
- `HiDigitalOutputLen`
- `LoDigitalOutputLen`
- `DigitalInputLen`
- `IsHiDigitalInputLen`
- `IsLoDigitalInputLen`
- `AnalogInputLen`
- `AnalogOutputLen`
- `GetCounterLen`

- `ClearCounterLen`
- `GetSerialNumberLen`
- `type_of_device_len`

Biblioteka „lendev.dll” korzysta z biblioteki „kernel32.dll” a sterownik „lenR232.exe” z „kernel32.dll” oraz „lendev.dll”.

ZAŁĄCZNIK J: WYMIARY URZĄDZENIA

Urządzenie ma wymiary zewnętrzne jak na rysunku J1.



Rys. J1. *Zewnętrzne wymiary urządzenia LENDEVICE80RS232.*

Tab. J1. Zewnętrzne wymiary urządzenia [mm].

A	B	C	D	E	H
80	100	18	30	6	24

ZAŁĄCZNIK K: PLIKI LENDEV.H ORAZ LENDEV.HPP

```
// lendev.h:
// (najważniejsze fragmenty)

#include <windows.h>

#ifndef DLL_IMPORT_DEVICERS232MOD1_LEN_H
#define DLL_IMPORT_DEVICERS232MOD1_LEN_H extern "C" __declspec(dllimport)
#endif

#define I_L(TYPE) DLL_IMPORT_DEVICERS232MOD1_LEN_H TYPE __stdcall

typedef LPVOID LENDEVICE;
#define NO_LENDEVICE ((LENDEVICE) 0)

enum DeviceMode { DEVICE_OFF = 0, DEVICE_ON = 1, DEVICE_BAD_TRANSMISSION = 2,
  DEVICE_WITHOUT_PROGRAM = 3 };

enum CommunicationMode { COMM_FAST_MODE = 0, COMM_CONTROLLED_MODE = 1 };

enum BaudRate { BR_300 = 0, BR_600 = 1, BR_1200 = 2, BR_2400 = 3, BR_4800 = 4,
  BR_9600 = 5, BR_14400 = 6, BR_19200 = 7, BR_38400 = 8, BR_57600 = 9, BR_115200 = 10,
  BR_MAX_AVAILABLE = 11 };

I_L( LENDEVICE )  open_len(const INT port);

I_L( LENDEVICE )  open_with_baudrate_len(const INT port, const BaudRate baudrate);

I_L( BOOL )       close_len(const LENDEVICE hdevice);

I_L( BOOL )       is_open_len(const LENDEVICE hdevice, const BOOL check_obligatorily);

I_L( INT )        get_last_error_len(const LENDEVICE hdevice);

I_L( BOOL )       change_baudrate_len(const LENDEVICE hdevice,
                    const BaudRate new_baudrate);

I_L( BOOL )       set_startup_baudrate_len(const LENDEVICE hdevice,
                    const BaudRate new_baudrate);

I_L( BOOL )       change_communication_mode_len(const LENDEVICE hdevice,
                    const CommunicationMode new_mode);

I_L( BOOL )       reset_device_len(const LENDEVICE hdevice);

I_L( INT )        type_of_device_len(const LENDEVICE hdevice);

I_L( VOID )       info_len(const LENDEVICE hdevice, BYTE information[40]);

I_L( DeviceMode ) check_device_len(const LENDEVICE hdevice, const INT how_many);

I_L( BOOL )       program_device_len(const LENDEVICE hdevice, const BYTE* file_ansi);

I_L( VOID )       digital_output_len(const LENDEVICE hdevice, const DWORD outputs);

I_L( VOID )       hi_digital_output_len(const LENDEVICE hdevice, const DWORD outputs);
```

```
I_L( VOID )       lo_digital_output_len(const LENDEVICE hdevice, const DWORD outputs);

I_L( DWORD )      digital_input_len(const LENDEVICE hdevice);

I_L( BOOL )       is_hi_digital_input_len(const LENDEVICE hdevice, const DWORD inputs);

I_L( BOOL )       is_lo_digital_input_len(const LENDEVICE hdevice, const DWORD inputs);

I_L( VOID )       analog_output_len(const LENDEVICE hdevice, const INT channel,
                    const FLOAT value);

I_L( FLOAT )      analog_input_len(const LENDEVICE hdevice, const INT channel);

I_L( DWORD )      get_counter_len(const LENDEVICE hdevice, const INT number);

I_L( VOID )       clear_counter_len(const LENDEVICE hdevice, const INT number);

I_L( DWORD )      get_serial_number_len(const LENDEVICE hdevice);

I_L( BOOL )       DigitalOutputLen(const LENDEVICE hdevice, const DWORD outputs);

I_L( BOOL )       HiDigitalOutputLen(const LENDEVICE hdevice, const DWORD outputs);

I_L( BOOL )       LoDigitalOutputLen(const LENDEVICE hdevice, const DWORD outputs);

I_L( BOOL )       DigitalInputLen(const LENDEVICE hdevice, DWORD* inputs_ptr);

I_L( BOOL )       IsHiDigitalInputLen(const LENDEVICE hdevice, const DWORD inputs,
                    BOOL* result_ptr);

I_L( BOOL )       IsLoDigitalInputLen(const LENDEVICE hdevice, const DWORD inputs,
                    BOOL* result_ptr);

I_L( BOOL )       AnalogOutputLen(const LENDEVICE hdevice, const INT channel,
                                  const FLOAT value);

I_L( BOOL )       AnalogInputLen(const LENDEVICE hdevice, const INT channel,
                    FLOAT* inputs_ptr);

I_L( BOOL )       GetCounterLen(const LENDEVICE hdevice, const INT number,
                    DWORD* result_ptr);

I_L( BOOL )       ClearCounterLen(const LENDEVICE hdevice, const INT number);

I_L( BOOL )       GetSerialNumberLen(const LENDEVICE hdevice, DWORD* result_ptr);

I_L( BOOL )       check_accessibility_com_len(const INT port, INT* problems_ptr,
                    BaudRate* max_baudrate_ptr, INT* number_of_available_baudrate_ptr,
                    BOOL table_of_available_baudrate[11]);

/* for function: check_accessibility_com_len(...), see documentation */
#define HARDWARE_NO_PROBLEM                          0x000
#define HARDWARE_NO_OPERATED_9600_BAUDRATE           0x001
#define HARDWARE_ONLY_WITH_EXTERNAL_SOURCE           0x002
#define HARDWARE_CRITICAL_ERROR                      0x004

/* returns by get_last_error_len */
#define ERROR_SUCCESS_LEN             0
#define ERROR_SYSTEM_PROBLEM_LEN      1
```

```
#define ERROR_INVALID_PARAMETERS_LEN  2
#define ERROR_COMMUNICATION_LEN       3
#define ERROR_INVALID_INSTRUCTION_LEN 4
#define ERROR_BAD_HANDLE_LEN          -1

/* for function: type_of_device_len(...) */
#define LENTECDEVICE_NOOPENED    0
#define LENTECDEVICE80RS232     10
#define LENTECDEVICE320RS232    20
#define LENTECDEVICE1280RS232   30


/* names of digital outputs, for functions:
  digital_output_len(...), hi_digital_output_len(...), lo_digital_output_len(...)
  DigitalOutputLen(...), HiDigitalOutputLen(...), LoDigitalOutputLen(...) */
#define DOUT0   (1ul << 0)
#define DOUT1   (1ul << 1)
#define DOUT2   (1ul << 2)
#define DOUT3   (1ul << 3)
#define DOUT4   (1ul << 4)
#define DOUT5   (1ul << 5)
#define DOUT6   (1ul << 6)
#define DOUT7   (1ul << 7)
#define DOUT8   (1ul << 8)
#define DOUT9   (1ul << 9)
#define DOUT10  (1ul << 10)
#define DOUT11  (1ul << 11)
#define DOUT12  (1ul << 12)
#define DOUT13  (1ul << 13)
#define DOUT14  (1ul << 14)
#define DOUT15  (1ul << 15)
#define DOUT16  (1ul << 16)
#define DOUT17  (1ul << 17)
#define DOUT18  (1ul << 18)
#define DOUT19  (1ul << 19)
#define DOUT20  (1ul << 20)
#define DOUT21  (1ul << 21)
#define DOUT22  (1ul << 22)
#define DOUT23  (1ul << 23)
#define DOUT24  (1ul << 24)
#define DOUT25  (1ul << 25)
#define DOUT26  (1ul << 26)
#define DOUT27  (1ul << 27)
#define DOUT28  (1ul << 28)
#define DOUT29  (1ul << 29)
#define DOUT30  (1ul << 30)
#define DOUT31  (1ul << 31)

/* names of digital inputs, for functions:
  digital_input_len(...), is_hi_digital_input_len(...), is_lo_digital_input_len(...)
  DigitalInputLen(...), IsHiDigitalInputLen(...), IsLoDigitalInputLen(...) */
#define DIN0  (1ul << 0)
#define DIN1  (1ul << 1)
#define DIN2  (1ul << 2)
#define DIN3  (1ul << 3)
#define DIN4  (1ul << 4)
#define DIN5  (1ul << 5)
#define DIN6  (1ul << 6)
#define DIN7  (1ul << 7)
#define DIN8  (1ul << 8)
```

```
#define DIN9  (1ul << 9)
#define DIN10 (1ul << 10)
#define DIN11 (1ul << 11)
#define DIN12 (1ul << 12)
#define DIN13 (1ul << 13)
#define DIN14 (1ul << 14)
#define DIN15 (1ul << 15)
#define DIN16 (1ul << 16)
#define DIN17 (1ul << 17)
#define DIN18 (1ul << 18)
#define DIN19 (1ul << 19)
#define DIN20 (1ul << 20)
#define DIN21 (1ul << 21)
#define DIN22 (1ul << 22)
#define DIN23 (1ul << 23)
#define DIN24 (1ul << 24)
#define DIN25 (1ul << 25)
#define DIN26 (1ul << 26)
#define DIN27 (1ul << 27)
#define DIN28 (1ul << 28)
#define DIN29 (1ul << 29)
#define DIN30 (1ul << 30)
#define DIN31 (1ul << 31)

/* names of analog outputs, for functions:
   analog_output_len(...), AnalogOutputLen(...) */
#define AOUT0  0
#define AOUT1  1
#define AOUT2  2
#define AOUT3  3
#define AOUT4  4
#define AOUT5  5

/* names of analog inputs, for functions:
   analog_input_len(...), AnalogInputLen(...) */
#define AIN0  0
#define AIN1  1
#define AIN2  2
#define AIN3  3
#define AIN4  4
#define AIN5  5
#define AIN6  6
#define AIN7  7

/* name of counter, for functions:
  get_counter_len(...), clear_counter_len(...),
  GetCounterLen(...), ClearCounterLen(...) */
#define L0    0

#undef I_L
```

```
// lendev.hpp
// najważniejsze fragmenty

#include <string>

namespace len
{

class DeviceRS232
{
public:
DeviceRS232();

DeviceRS232(const int port);

DeviceRS232(const int port, const BaudRate baudrate);

~DeviceRS232();

bool open(const int port);

bool          open_with_baudrate(const int port, const BaudRate baudrate);

bool          close();

bool          is_open(const bool check_obligatorily = false);

bool          change_baudrate(const BaudRate new_baudrate);

bool          set_startup_baudrate(const BaudRate new_baudrate);

bool          change_communication_mode(const CommunicationMode new_mode);

std::string   info();

int           type_of_device();

DeviceMode    check_device(const int how_many = 10);

bool          reset_device();

bool          program_device(const std::string& file_ansi);

unsigned long get_serial_number();

int           get_last_error();

void          digital_output(const unsigned long outputs);

template<int N> void digital_output(const std::bitset<N>& outputs);

void          hi_digital_output(const unsigned long outputs);

template<int N> void hi_digital_output(const std::bitset<N>& outputs);

void          lo_digital_output(const unsigned long outputs);

template<int N> void lo_digital_output(const std::bitset<N>& outputs);

unsigned long digital_input();
```

```
bool          is_hi_digital_input(const unsigned long  inputs);

template<int N> bool is_hi_digital_input(const std::bitset<N>& inputs);

bool          is_lo_digital_input(const unsigned long inputs);

template<int N> bool is_lo_digital_input(const std::bitset<N>& inputs);

float         analog_input(const int channel);

void          analog_output(const int channel, const float value);

unsigned long get_counter(const int number);

void          clear_counter(const int number);

bool          DigitalOutput(const unsigned long outputs);

template<int N> bool DigitalOutput(const std::bitset<N>& outputs);

bool          HiDigitalOutput(const unsigned long outputs);

template<int N> bool HiDigitalOutput(const std::bitset<N>& outputs);

bool          LoDigitalOutput(const unsigned long outputs);

template<int N> bool LoDigitalOutput(const std::bitset<N>& outputs);

bool          DigitalInput(unsigned long* inputs_ptr);

bool          IsHiDigitalInput(const unsigned long  inputs, bool* result_ptr);

template<int N> bool IsHiDigitalInput(const std::bitset<N>& inputs, bool* result_ptr);

bool          IsLoDigitalInput(const unsigned long  inputs, bool* result_ptr);

template<int N> bool IsLoDigitalInput(const std::bitset<N>& inputs, bool* result_ptr);

bool          AnalogInput(const int channel, float* result_ptr);

bool          AnalogOutput(const int channel, const float value);

bool          GetCounter(const int number, unsigned long* result_ptr);

bool          ClearCounter(const int number);

bool          GetSerialNumber(unsigned long* result_ptr);

private:
  LENDEVICE lendevice;
};  // end of class "DeviceRS232"

bool        check_accessibility_com(const int port, int *const problems_ptr,
              BaudRate *const max_baud_rate_ptr,
              int *const number_of_available_baud_rate_ptr,
              bool table_of_available_baud_rate[11]);

} // end of namespace "len"
```

Lentec Design
Polska
tel. 505-69-11-17
www.lentecdesign.com
Biuro: office@lentecdesign.com
Pomoc techniczna: support@lentecdesign.com