

Министерство образования и науки Российской Федерации
Государственное образовательное учреждение
высшего профессионального образования
«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»

Кафедра приборостроения и информационно-измерительных
технологий

ИНТЕРФЕЙСЫ ИЗМЕРИТЕЛЬНЫХ УСТРОЙСТВ

Методические указания к лабораторным работам

Составители
Н.Ю. МАКАРОВА
Д.Д. ПАВЛОВ



Владимир 2011

УДК 681.142
ББК 32.84 216
И73

Рецензент
Кандидат технических наук, профессор
Владимирского государственного университета
Г.П. Колесник

Печатается по решению редакционного совета
Владимирского государственного университета

Интерфейсы измерительных устройств: метод. указания к
И73 лаб. работам / Владим. гос. ун-т.; сост.: Н. Ю. Макарова,
Д. Д. Павлов. – Владимир: Изд-во Владим. гос. ун-та. – 2011. –
40 с.

Приведены методики выполнения лабораторных работ по сопряжению современных приборов с компьютером с использованием стандартных интерфейсов и информационных протоколов по дисциплине «Интерфейсы измерительных устройств».

Предназначены для студентов 4-го курса дневного отделения, обучающихся по специальности 200106 - информационно-измерительная техника и технологии.

Рекомендовано для формирования профессиональных компетенций в соответствии с ФГОС 3-го поколения.

Ил. 14. Табл. 1. Библиогр.: 6 назв.

УДК 681.142
ББК 32.84 216

ВВЕДЕНИЕ

Курс “Интерфейсы измерительных устройств” изучается в 7-м и 8-м семестрах и является важным компонентом для успешного усвоения последующих дисциплин «Измерительные информационные системы», «Интеллектуальные средства измерений», «Компьютерные измерения» и необходим студентам для последующего решения производственных и исследовательских задач в соответствии с квалификационной характеристикой инженера по специальности 200106 «Информационно-измерительная техника и технологии».

Представленные методические указания знакомят студентов со схемными решениями и программными средствами для сопряжения разнообразных датчиков с персональным ЭВМ семейства IBM PC, со стандартными интерфейсами и приборными шинами; формирования представления о внутренней организации, порядке функционирования и режимах работы устройств сопряжения; принципах их взаимодействия с логикой шин и компонентами информационной или управляющей системы.

В методических указаниях рассмотрены параллельный интерфейс CENTRONIX и работа с LPT-портом (лабораторная работа №1), последовательный интерфейс RS-232 и работа с COM-портом (лабораторная работа № 2), особенности чтения и записи данных в COM-порт, протокол MODBUS (лабораторная работа № 3), работа с прибором измерения давления через интерфейс RS-485 (лабораторная работа № 4). В лабораторных работах приведены тексты программ с подробными комментариями для ознакомления и правки.

Лабораторная работа № 1

ПАРАЛЛЕЛЬНЫЙ ИНТЕРФЕЙС РАБОТА С LPT-ПОРТОМ

Цель работы – изучение состава и принципа работы параллельного интерфейса, получение навыков работы с LPT-портом ПК.

Оборудование – ПК с ОС Windows XP и выше, плата, совместимая с ПК для индикации сигналов на контактах LPT-порта.

Введение

LPT-порт – это набор контактов, на которых мы можем установить напряжение 0 или +5 В (логическая 0 и 1) из программы или это может сделать внешнее устройство снаружи. Давайте разберемся, какими контактами мы можем оперировать, а какими нет. В этом нам поможет рис.1.1.

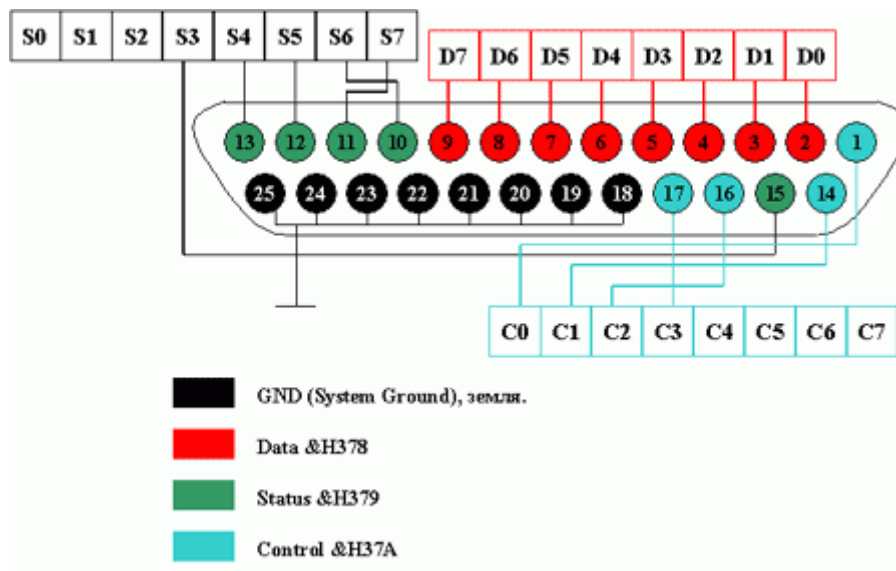


Рис.1.1 Наименования контактов LPT-порта

Из рис.1.1 видно, что выходы порта можно разделить на четыре группы: это “земляные” выходы (8 штук). Они обозначены черным цветом (контакты 18 – 25). Все они соединены между собой, поэтому для своих разработок в качестве земли можно использовать любой из них.

Буквами $D_7 - D_0$ обозначены выходы так называемого регистра **Data** (контакты 2 – 9). Под регистром будем понимать объединение группы контактов LPT-порта. В регистре Data их 8 штук. Этот регистр позволяет нам как из программы, так и из внешнего устройства установить на его контактах логическую 0 или 1, т.е. он двунаправленный.

Регистр **Status** (контакты 10 – 13, 15). Это однонаправленный регистр. Управлять им можно только снаружи через внешнее устройство (имеется в виду изменять данные на нем, читать можно из любого регистра в любую сторону). И регистр **Control** (контакты 1, 14, 16, 17). Он имеет всего 4 контакта и может управляться только программой.

Работа с LPT-портом

Далее при помощи простого устройства (рис. 1.2) можно визуально отследить работу параллельного интерфейса и LPT-порта.

Как видно из рис. 1.2, в устройстве есть 17 светодиодов. Каждый из них соответствует определенному биту LPT-порта. 8 светодиодов – регистр Data, 4 в верхнем левом углу – регистр Control и 5 находящихся справа – регистр Status. В этом устройстве в первые два регистра мы будем записывать информацию из программы, а в последний (Status) – данные снаружи с помощью блока переключателей.



Рис. 1.2. Устройство, демонстрирующее работу параллельного интерфейса и LPT-порта

Запись / чтение данных в LPT-порт

Далее требуется написать программу для проведения чтения и записи данных в порт. Программа предназначена для работы как под ОС Windows 9x, так и под Windows NT. Для обеспечения такой функциональности используется библиотека **inpout32.dll**.

Итак, создаем пустой проект консольного приложения в VS++, создаем пустой файл *.cpp и записываем туда следующий код:
`#include <iostream.h>`

```

#include <stdio.h>
#include <conio.h>
#include "h.h"
void WriteToDataRegister()
{
    cout<<"Write data to DATA register.\n"<<endl;
    int b=0;
    cout<<"Enter data: "<<endl;
    cin>>b;
    if (b<0||b>255)
        cout<<"Error! You can take data only from diaposon 0-
255."<<endl;
    else
    {
        Out32(888, b);
        cout<<"Write done.\n"<<endl;
    }
    cout<<"Press any key to back to Main menu..."<<endl;
    getch();
    system("cls");
}
void ReadDataRegister()
{
    cout<<"Read data from DATA register.\n"<<endl;
    int data;
    data = Inp32(888);
    cout<<"10: "<<data<<endl;
    char number[20];
    itoa(data,number,2);
    cout<<" 2: "<<number<<endl;
    cout<<"\nPress any key to back to Main menu..."<<endl;
    getch();
    system("cls");
}

```

```

void WriteToControlRegister()
{
    cout<<"Write data to CONTROL register.\n"<<endl;
    int b=0;
    cout<<"Enter data: "<<endl;
    cin>>b;
    if (b<0||b>16)
    cout<<"Error! You can take data only from diaposon 0-16."<<endl;
    else
    {
        Out32(890, b);
        cout<<"Write done.\n"<<endl;
    }
    cout<<"Press any key to back to Main menu..."<<endl;
    getch();
    system("cls");
}

void ReadControlRegister()
{
    cout<<"Read data from CONTROL register.\n"<<endl;
    int data;
    data = Inp32(890);
    cout<<"10: "<<data<<endl;
    char number[20];
    itoa(data,number,2);
    cout<<" 2: "<<number<<endl;
    cout<<"\nPress any key to back to Main menu..."<<endl;
    getch();
    system("cls");
}

void ReadStatusRegister()
{
    cout<<"Read data from STATUS register.\n"<<endl;
    int data;
    data = Inp32(889);
}

```

```

    cout<<"10: "<<data<<endl;
    char number[20];
    itoa(data,number,2);
    cout<<" 2: "<<number<<endl;
    cout<<"\nPress any key to back to Main menu..."<<endl;
    getch();
    system("cls");
}
void Menu()
{
    cout<<"Select operation:"<<endl;
    cout<<"1 - write data to DATA register."<<endl;
    cout<<"2 - read data from DATA register"<<endl;
    cout<<"3 - write data to CONTROL register."<<endl;
    cout<<"4 - read data from CONTROL register"<<endl;
    cout<<"5 - read data from STATUS register\n"<<endl;
    cout<<"0 - exit"<<endl;
}
int main()
{
    Menu();
    while(1){
        switch(getch())
        {
            case '1': system("cls");
                    WriteToDataRegister();
                    Menu();
                    break;

            case '2': system("cls");
                    ReadDataRegister();
                    Menu();
                    break;

            case '3': system("cls");
                    WriteToControlRegister();

```



```

        Menu();
        break;
    case '4': system("cls");
        ReadControlRegister();
        Menu();
        break;
    case '5': system("cls");
        ReadStatusRegister();
        Menu();
        break;
    case '0': exit(0);
        break;
    }
}
return true;
}

```

Не забываем скопировать файлы **h.h**, **inpout32.lib**, **inpout32.dll** в папку проекта, а файл **inpout32.dll** еще и в папку **Debug**, где будет располагаться откомпилированная программа. Далее выбираем **Project -> Add to Project -> Files...** В открывшемся окне открыть папку вашего проекта, выбирать там только что помещенный туда файл **h.h** и нажимать ОК. Файл добавится в проект. Прodelывайте такую же операцию и для файла **inpout32.lib**.

Затем компилируем и запускаем. Вы должны увидеть окно (рис. 1.3). Перед нами так называемое DOS-меню, реализованное на операторе switch-case. Нажатием клавиш от 0 до 5 вызываем соответствующий обработчик case.



Рис. 1.3. DOS-меню, реализованное на операторе switch-case

Нажмем, например, клавишу **1**. Будет вызвана функция `WriteToDataRegister()`, в которой осуществляется ввод данных с клавиатуры и запись их в регистр DATA LPT-порта (рис. 1.4). Перед вызовом этой функции строчкой `system ("cls")`; проводим очистку экрана: так работать удобней. Вводим число в диапазоне от 0 до 255 для записи в порт. Если число выходит за эти пределы, запись не проводим (т.к. регистр DATA 8- битный, и больше в него не поместится). Запись проводим с помощью функции `Out32()` библиотеки `inout32.dll`. Первым параметром этой функции передается адрес порта, куда надо записать данные, вторым параметром - сами данные.

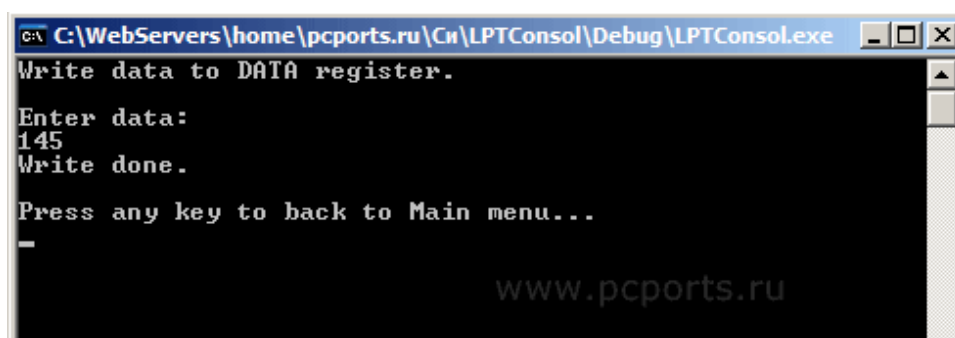
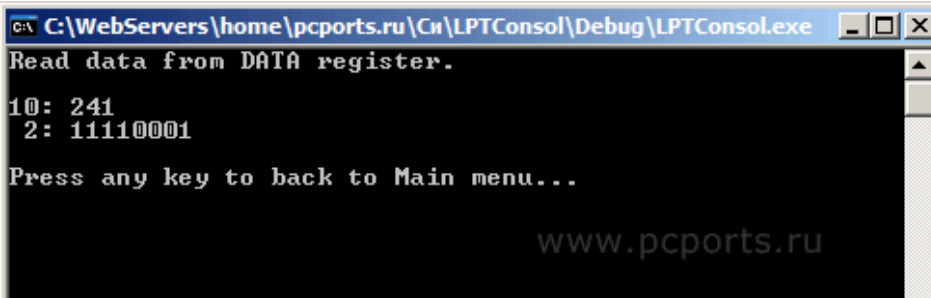


Рис. 1.4. Ввод данных с клавиатуры и запись их в регистр DATA LPT-порта

Нажатием клавиши **3** вызовем аналогичную операцию ввода и записи данных в регистр CONTROL. Следует обратить внимание, что границы вводимых чисел ограничены диапазоном 0 – 16, т.к., хотя CONTROL и 8-битный, но реально в нем используются только первые четыре бита.

Если нажмем **2**, то будем читать данные из регистра DATA (рис. 1.5). Чтение проводим с помощью функции `Inp32()` библиотеки `inout32.dll`. Единственным параметром этой функции передается адрес порта, откуда надо произвести чтение. Функция возвращает прочитанные данные в десятичной форме, которые мы тут же и выводим на экран. Для большего удобства восприятия с помощью стандартной функции `itoa()` проводим преобразование в двоичную систему счисления. Согласитесь, так гораздо удобнее: смотришь на блок светодиодов и сверяешь с результатом чтения, представленным в виде нулей и единиц (горит – не горит).

С помощью клавиш **4** и **5** аналогично проводим чтение данных из регистра CONTROL и STATUS.



```
C:\WebServers\home\pcports.ru\Cm\LPTConsol\Debug\LPTConsol.exe
Read data from DATA register.
10: 241
2: 11110001
Press any key to back to Main menu...
www.pcports.ru
```

Рис. 1.5. Чтение данных из регистра DATA

О регистре STATUS следует поговорить подробнее. В него, как мы знаем, данные можно записать только снаружи. Это мы сейчас и сделаем. Собрав устройство, мы сможем писать данные в регистр. Используя набор ключей (замыкателей), будем устанавливать на соответствующем выводе регистра STATUS логический ноль или единицу. Попробуйте установить какую-нибудь произвольную комбинацию переключателей и прочтите состояние регистра с помощью программы. Потом установите другую комбинацию. При этом, прочитав снова, вы получите другое число. Так программа может узнать об изменении состояния внешнего устройства.

К сожалению, при работе с регистром STATUS может возникнуть неприятная ситуация - "помигивание" других битов порта при первом замыкании на землю.

Вопросы для усвоения

1. Что такое параллельный интерфейс?
2. Как осуществляется чтение и запись данных по параллельному интерфейсу?
3. Назначение контактов LPT-порта.
4. В чем заключаются плюсы и минусы параллельного интерфейса?
5. Где применяется параллельный интерфейс?
6. Как осуществляется управление внешними устройствами через параллельный интерфейс?
7. Как осуществить управление ПК через параллельный интерфейс?

Лабораторная работа № 2

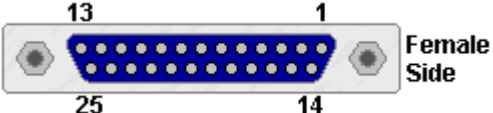
ПОСЛЕДОВАТЕЛЬНЫЙ ИНТЕРФЕЙС РАБОТА С СОМ-ПОРТОМ

Цель работы – изучение состава и принципа работы последовательного интерфейса RS-232, получение навыков работы с СОМ-портом ПК.

Оборудование – два ПК с ОС Windows XP и выше, нуль-модемный кабель.

Введение

RS-232 – популярный протокол, применяемый для связи компьютеров с модемами и другими периферийными устройствами. В данной работе представлены комплект полезной и справочной информации, распиновка стандартных разъемов, описаны, что такое квитирование (HANDSHAKING), и применение микросхем MAX232 фирмы MAXIM. RS-232 – интерфейс передачи информации между двумя устройствами на расстоянии до 20 м. Информация передается по проводам с уровнями сигналов, отличающимися от стандартных 5В, для обеспечения большей устойчивости к помехам. Асинхронная передача данных осуществляется с установленной скоростью при синхронизации уровнем сигнала стартового импульса. Интерфейс RS-232-C был разработан для простого применения, однозначно определяемого по его названию "Интерфейс между терминальным оборудованием и связным оборудованием с обменом по последовательному двоичному коду". Каждое слово в названии значимое, оно определяет интерфейс между терминалом (DTE) и модемом (DCE) по передаче последовательных данных. Устройства для связи по последовательному каналу соединяются кабелями с девятью или двадцатьюпятью контактными разъемами типа D. Обычно они обозначаются DB-9, DB-25, CANNON 9, CANNON 25 и т.д. Разъемы типов розетки и штырей. Каждый вывод обозначен и пронумерован. Расположение выводов представлено ниже (рис 2.1).

| Контакт | Обозначение | Направление | Описание |
|---|-------------|-------------|--|
|  <p>DB25 Розетка (мама)</p> | | | |
| 1 | SHIELD | --- | Shield Ground - защитная земля, соединяется с корпусом устройства и экраном кабеля |
| 2 | TXD | --> | Transmit Data - выход передатчика |
| 3 | RXD | <-- | Receive Data - вход приемника |
| 4 | RTS | --> | Request to Send - выход запроса передачи данных |
| 5 | CTS | <-- | Clear to Send - вход разрешения терминалу передавать данные |
| 6 | DSR | <-- | Data Set Ready - вход сигнала готовности от аппаратуры передачи данных |
| 7 | GND | --- | System Ground - сигнальная (схемная) земля |
| 8 | CD | <-- | Carrier Detect - вход сигнала обнаружения несущей удаленного модема |
| 9-19 | N/C | - | - |
| 20 | DTR | --> | Data Terminal Ready - выход сигнала готовности терминала к обмену данными |
| 21 | N/C | - | - |
| 22 | RI | <-- | Ring Indicator - вход индикатора вызова (звонка) |
| 23-25 | N/C | - | - |

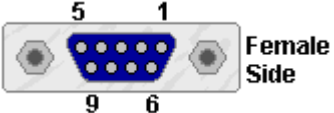
| Контакт | Обозначение | Направление | Описание |
|---|-------------|-------------|---------------------|
|  <p>DB9 Розетка (мама)</p> | | | |
| 1 | CD | <-- | Carrier Detect |
| 2 | RXD | <-- | Receive Data |
| 3 | TXD | --> | Transmit Data |
| 4 | DTR | --> | Data Terminal Ready |
| 5 | GND | --- | System Ground |
| 6 | DSR | <-- | Data Set Ready |
| 7 | RTS | --> | Request to Send |
| 8 | CTS | <-- | Clear to Send |
| 9 | RI | <-- | Ring Indicator |

Рис. 2.1. Контакты разъемов

Нуль-модемные кабели RS-232

Рассмотрим сначала DSR сигнал (конт. 6) (рис. 2.2). Это вход сигнала готовности от аппаратуры передачи данных. В схеме соединений вход замкнут на выход DTR

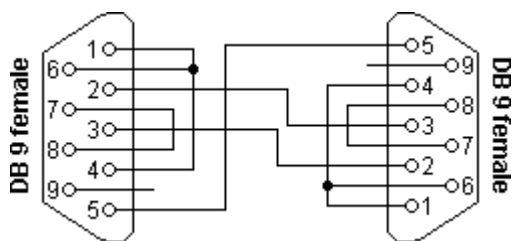


Рис. 2.2. Нуль-модемный кабель
3-проводный минимальный

для контроля возможности соединения будет установлен выходной сигнал DTR.

Это соответствует 99 % коммуникационного программного обеспечения. Под этим подразумевается, что 99 % программного обеспечения с этим нуль-модемным кабелем примут проверку сигнала DSR.

Аналогичный подход применяется для входного сигнала CTS. В оригинале сигнал RTS (конт. 7) устанавливается и затем проверяется CTS (конт. 8).

Соединение этих контактов приводит к невозможности зависания программ по причине отсутствия ответа на запрос RTS.

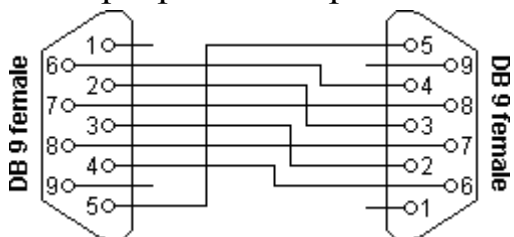


Рис. 2.3. Нуль-модемный кабель
7-проводный полный

Самый дорогой полный нуль-модемный кабель с семью проводами (рис. 2.3). Только сигналы индикатора вызова и определения несущей не подключены.

Этот кабель не разрешает использовать предыдущий метод контроля передачи данных. Основная несовместимость заключается в перекрестном соединении сигналов RTS и CTS. Первоначально эти сигналы использовались для контроля потоком данных по типу запрос/ответ. При использовании полного нуль-модемного кабеля более нет запросов. Эти сигналы применяются для сообщения другой стороне, есть ли возможность соединения.

Контакты 2 и 3 на 9-выводном разъеме D типа противоположны этим же контактам на 25-контактном разъеме. Поэтому, если соединить контакты 2-2 и 3-3 между разъемами D25 и D9, получится коммуникационный кабель. Контакты сигнальной земли Signal Ground (SG) также должны быть подключены между собой (рис. 2.4).

Можно найти или изготовить много типов кабелей для связи по интерфейсу RS-232. В этом нуль-модемном кабеле используется только 5 проводов: сигналы данных TXD, RXD, сигнал GND и управляющие сигналы RTS, CTS для управления потоком.

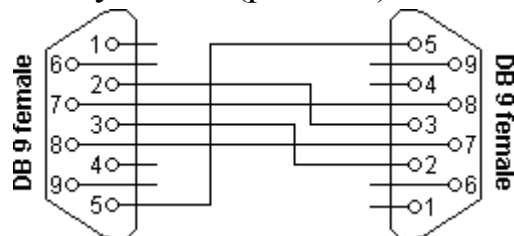


Рис. 2.4. Нуль-модемный кабель 5-проводный с управлением потоком

Обозначение кабелей

Все DTE-DCE кабели прямого соединения, контакты соединяются один к одному. Кабели DTE-DTE и DCE-DCE кросс-кабели.

1. DTE - DCE называется прямым кабелем;
2. DTE - DTE называется нуль-модемным кабелем;
3. DCE - DCE называется Tail Circuit Cable.

Описание полного нуль-модемного кабеля

Соединение D9- D9

| DB9-1 | | DB9-2 | |
|---------------------------------|-----|-------|---------------------------------|
| Receive Data | 2 | 3 | Transmit Data |
| Transmit Data | 3 | 2 | Receive Data |
| Data Terminal Ready | 4 | 6+1 | Data Set Ready + Carrier Detect |
| System Ground | 5 | 5 | System Ground |
| Data Set Ready + Carrier Detect | 6+1 | 4 | Data Terminal Ready |
| Request to Send | 7 | 8 | Clear to Send |
| Clear to Send | 8 | 7 | Request to Send |

Соединение D25-D25

| DB25-1 | | DB25-2 | |
|--------------|---|--------|---------------|
| Receive Data | 3 | 2 | Transmit Data |

| | | | |
|---------------------------------|-----|-----|---------------------------------|
| Transmit Data | 2 | 3 | Receive Data |
| Data Terminal Ready | 20 | 6+8 | Data Set Ready + Carrier Detect |
| System Ground | 7 | 7 | System Ground |
| Data Set Ready + Carrier Detect | 6+8 | 20 | Data Terminal Ready |
| Request to Send | 4 | 5 | Clear to Send |
| Clear to Send | 5 | 4 | Request to Send |

Соединение D9-D25

| DB9 | | DB25 | |
|---------------------------------|-----|------|---------------------------------|
| Receive Data | 2 | 2 | Transmit Data |
| Transmit Data | 3 | 3 | Receive Data |
| Data Terminal Ready | 4 | 6+8 | Data Set Ready + Carrier Detect |
| System Ground | 5 | 7 | System Ground |
| Data Set Ready + Carrier Detect | 6+1 | 20 | Data Terminal Ready |
| Request to Send | 7 | 5 | Clear to Send |
| Clear to Send | 8 | 4 | Request to Send |

В данной лабораторной работе будет использоваться полный нуль-модемный кабель 7-проводный.

Выполнение работы

Сначала требуется создать консольное приложение, которое будет считывать символ с клавиатуры и посылать его в COM-порт. Создаем в Microsoft Visual Studio консольное приложение и называем его, например, **COMServer**. К пустой заготовке необходимо подключить библиотеку **SerialGate.dll**. Последовательность действий при этом полностью аналогична работе с библиотекой **inport32.dll**. Помещаем файлы **SerialGate.lib**, **SerialGate.h** в папку проекта и **SerialGate.dll** в папку к исполняемой программе. Далее добавляем файл **lib** и **h** в самом проекте.

Ниже приведен код программы. Рассмотрим, что в нем происходит. Сначала создается экземпляр класса **SerialGate** - **sg**. Далее пытаемся открыть доступ к COM-порту с помощью метода класса **OpenPort()**. Для этого в функцию нужно передать номер COM-порта

(целое число), т.е., например, если на компьютере установлен COM-порт под названием COM3, и вы хотите его открыть, вам нужно передать первым параметром в функцию число 3. Второй параметр - это скорость, на которой будет работать порт при передаче данных. Скорость для COM-портов измеряется в битах в секунду (бод) и может принимать некоторое значение из определенного списка. По умолчанию в Windows можно использовать следующие скорости:

110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 57600, 115200, 128000 и 256000 бит/с.

В представленном примере открывается порт "COM1" на скорости 57600 бит/с. Далее в "бесконечном цикле" с клавиатуры вводится символ и с помощью метода класса Send() отправляется в COM-порт. Функции передают два параметра: адрес на буфер, в котором находятся данные для отправки, и второй параметр – размер этих данных в байтах. Для завершения ввода достаточно ввести символ '#' - программа выйдет из цикла и, используя метод Close(), закроет порт.

```
#include <iostream>
#include <conio.h>
#include "SerialGate.h"
int main()
{
    int port = 1;
    int rate = 57600;
    SerialGate sg;
    bool res = sg.Open(port, rate);
    if(res == false)
    {
        printf("Open Error..\n");
        getch();
        return 0;
    }
    else
    {
        printf("Open OK!..\n");
```

```

    }
    printf("Press key to send it to COM port.\n");
    printf("Press '#' to exit.\n\n");
    while(true)
    {
        char c = getch();
        printf("%c", c);
        sg.Send(&c, sizeof(c));
        if(c == '#')
            break;
    }
    sg.Close();
    getch();
    return 0;
}

```

Затем необходимо написать вторую программу, которая будет открывать СОМ-порт, соединенный с первым кабелем, и будет считывать оттуда данные.

Итак, опять создаем экземпляр класса SerialGate и открываем порт "СОМ2" на скорости 57600 бит/с. Обратите внимание, что для приема/передачи данных оба порта, участвующих в обмене информацией, должны быть открыты на одной и той же скорости. Далее в "бесконечном" цикле с интервалом в одну секунду считываем данные с СОМ-порта с помощью метода Recv(). Ему необходимо передать два параметра: адрес на буфер, в который будут положены прочтенные данные и размер этого буфера во избежание выходов за пределы массива. Функция возвращает число прочтенных байт. Затем прочтенные данные печатаются побайтно, и если будет обнаружен символ '#', значит COMServer закончил работу и этой программе нужно сделать то же самое.

```

#include <iostream>
#include <conio.h>
#include "SerialGate.h"
int main()

```

```

{
    int port = 2;
    int rate = 57600;
    SerialGate sg;
    bool res = sg.Open(port, rate);
    if(res == false)
    {
        printf("Open Error..\n");
        getch();
        return 0;
    }
    else
    {
        printf("Open OK!..\n");
    }
    printf("Get data from COM port every 1 sec.\n\n");
    char buf[20];
    int dwBytesRead = 0;
    bool terminate = false;
    while(!terminate)
    {
        Sleep(1000);
        dwBytesRead = sg.Recv(buf, sizeof(buf));
        for(int i = 0; i < dwBytesRead; i++)
        {
            printf("%c", buf[i]);
            if(buf[i] == '#')
            {
                terminate = true;
                break;
            }
        }
    }
    sg.Close();
}

```

```
    return 0;
}
```

Проверьте, соединили ли вы порты кабелем. Компилируем и запускаем оба приложения. В программе COMServer набираем какой-либо текст с клавиатуры, с некоторой задержкой он должен появиться в окне программы COMClient (рис. 2.5).

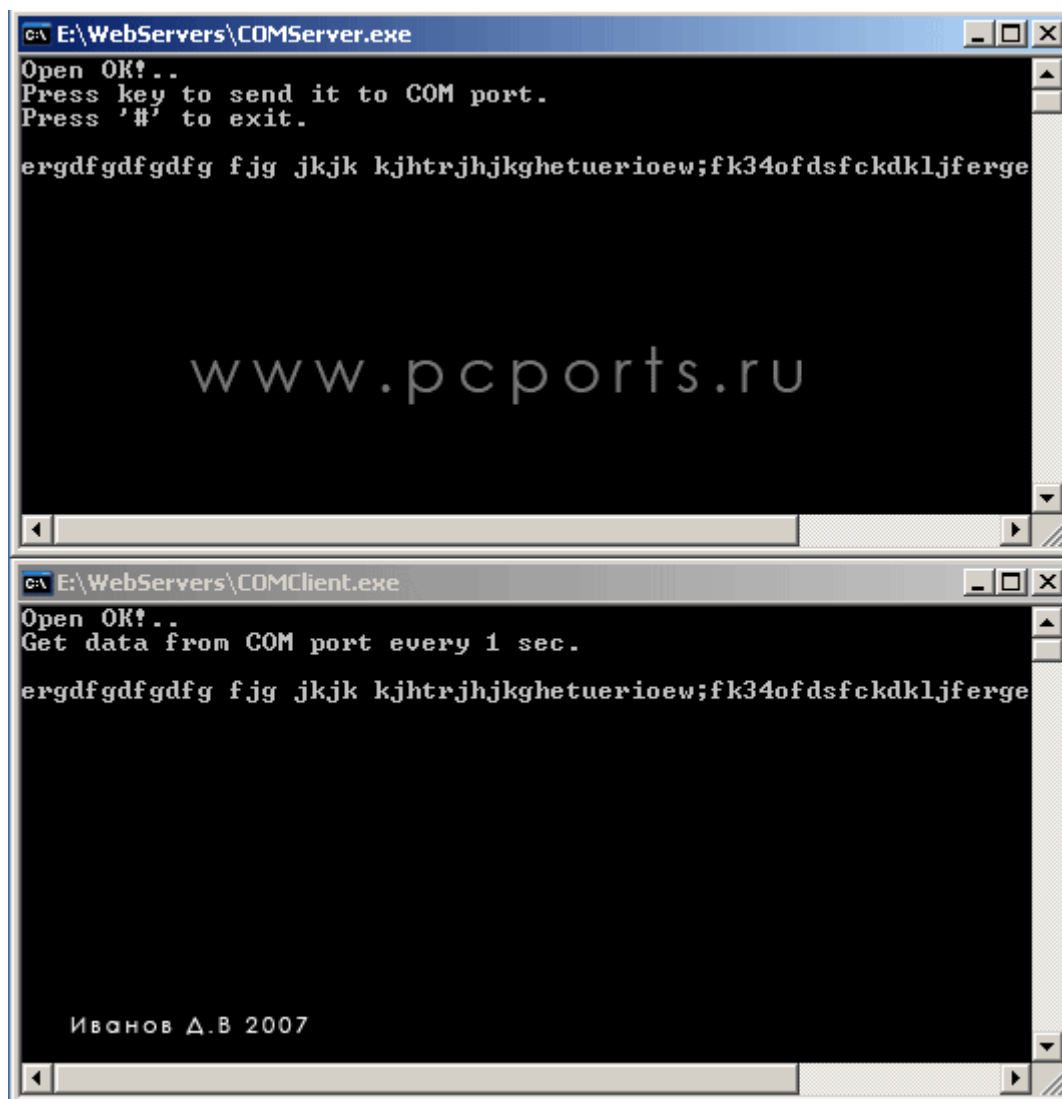


Рис. 2.5. Пример работы созданных программ

Вопросы для усвоения

1. В чем заключаются особенности использования последовательного интерфейса RS-232C?
2. Опишите организацию сопряжения через RS-232C.

3. Каковы внутреннее аппаратное устройство, разъем и кабель порта RS-232C?
4. Объясните порядок обмена по интерфейсу RS-232C.
5. Что такое "квитирование установления связи" и какова его аппаратная реализация?
6. Приведите типы последовательных интерфейсов и опишите их особенности.
7. Поясните функционирование универсального асинхронного приемопередатчика.
8. Какова структура сбора данных при использовании последовательного интерфейса?
9. Опишите особенности программирования параллельного интерфейса RS-232C.

Лабораторная работа № 3

ЧТЕНИЕ / ЗАПИСЬ ДАННЫХ В СОМ-ПОРТ. ПРОТОКОЛ MODBUS

Цель работы – создать приложение под ОС Windows для передачи и приема данных через СОМ-порт посредством интерфейса RS-232, протокол обмена данными Modbus.

Оборудование – два ПК с ОС Windows XP и выше, нуль-модемный кабель.

Введение

Modbus - коммуникационный протокол, основанный на клиент-серверной архитектуре. Разработан фирмой Modicon для использования в контроллерах с программируемой логикой (PLC). Стал стандартом де-факто в промышленности и широко применяется для организации связи промышленного электронного оборудования. Использует для передачи данных последовательные линии связи RS-485, RS-422, RS-232 и другие, а также сети TCP/IP.

Устройства разных производителей, поддерживающие протокол Modbus, легко интегрируются в единую сеть автоматизации. На рын-

ке представлен практически весь спектр необходимого оборудования от простых модулей ввода-вывода до частотных преобразователей. Все универсальные SCADA/HMI системы поддерживают данный протокол.

При использовании последовательных линий связи в одной сети может быть только одно ведущее устройство (master), которое может опрашивать другие подчиненные устройства (slave). Ни одно подчиненное устройство не может самостоятельно запросить или передать данные другому устройству. Ведущее устройство (master) может запросить данные с каждого подчиненного устройства по очереди или инициировать одновременную передачу сообщения на все подчиненные устройства. В одной сети может быть до 247 подчиненных устройств (slave).

Протокол Modbus предусматривает для передачи данных по последовательным линиям связи два режима передачи: RTU и ASCII.

Режим ASCII предназначен для медленных линий связи, где каждый байт пакета передается как два ASCII символа. Новый пакет начинается со специального служебного символа. При этом между передачей символов одного пакета пауза может быть несколько секунд (в зависимости от настроек) без возникновения ошибок при передаче.

Использование режима RTU позволяет приблизительно в 2 раза увеличить количество передаваемых данных по последовательной линии связи. Данные пакета передаются по сети в двоичном виде без изменений. В режиме RTU перед передачей пакета в линии пять секунд выдерживается небольшой интервал тишины. Пакет передается непрерывным потоком данных. Таким образом на скорости 19200 бит/с возможно передать за 1 с до 1400 байт данных (при повторяющемся запросе 126 переменных) или опросить до 40 раз одно или несколько устройств (при запросе одной переменной) RTU.

Пример

В этом примере показана структура передаваемых данных с синхронизирующим тактовым сигналом, используется 8 бит данных,

бит четности и стоп бит. Такая структура также обозначается 8E1 (рис. 3.1).

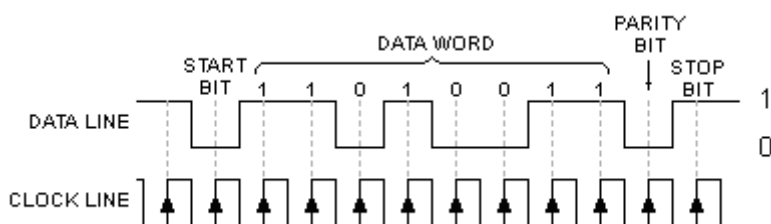


Рис. 3.1. Пример передаваемых данных по Modbus протоколу

Тактовый сигнал (clock line) для асинхронной передачи – это внутренний сигнал.

Старт бит

Сигнальная линия может находиться в двух состояниях: включена и выключена. Линия в состоянии ожидания всегда включена. Когда устройство или компьютер инициируют передачу данных, они переводят линию в состояние выключено - это установка старт бита. Биты сразу после старт бита являются битами данных.

Стоп бит

Стоп бит позволяет устройству или компьютеру произвести синхронизацию при возникновении сбоев. Например, помеха на линии скрывает старт бит. Период между старт и стоп битами постоянен согласно значению скорости обмена, числу бит данных и бита четности. Стоп бит всегда включен. Если приемник определяет выключенное состояние, когда должен присутствовать стоп бит, фиксируется появление ошибки.

Установка стоп бита

Стоп бит не просто один бит минимального интервала времени в конце каждой передачи данных. На компьютерах обычно он эквивалентен 1 или 2 битам, и это должно учитываться в программе драйвера. Хотя 1 стоп бит наиболее общий, выбор 2 бит в худшем случае немного замедлит передачу сообщения. (Есть возможность установки значения стоп бита равным 1,5. Это используется при передаче менее 7 битов данных. В этом случае не могут быть переданы символы ASCII, и поэтому значение 1,5 используется редко).

Выполнение работы

Сначала посмотрим в файл "SerialGateTest.h" (см. лаб. работу № 2). Здесь в описании класса можно увидеть объявление класса SerialGate sg.

```
class CSerialGateTestDlg : public CDialog  
{
```

```
// Construction
```

```
public:
```

```
    CSerialGateTestDlg(CWnd* pParent = NULL); // standard constructor
```

```
    SerialGate sg;
```

Теперь рассмотрим обработчик на нажатие кнопки "Open". Сначала идет минимальная проверка на валидность введенных данных, и, если они в порядке, с помощью метода класса SerialGate Open() пытаемся открыть указанный порт. Если открыть удалось, увидим сообщение об этом. Далее стартует таймер с идентификатором 1 и временем срабатывания 1000 мс. О нем поговорим позже.

```
void CSerialGateTestDlg::OnOpen()
```

```
{
```

```
    // TODO: Add your control notification handler code here
```

```
    UpdateData(true);
```

```
    if(m_port == 0 || m_rate == 0)
```

```
    {
```

```
        MessageBox("Not correct data","Error",  
MB_ICONERROR);
```

```
        return;;
```

```
    }
```

```
    bool b = sg.Open(m_port, m_rate);
```

```
    if(b == false)
```

```
    {
```

```
        MessageBox("Can`t open Port","Error",  
MB_ICONERROR);
```

```
        return;;
```

```
    }
```



```

        else
        {
            MessageBox("Port open OK","Info",
MB_ICONINFORMATION);
        }

        SetTimer(1, 1000, NULL);
    }

```

Обработчик таймера. Пытаемся прочесть данные из порта с помощью функции Recv(). Если данные были прочтены (их число в байтах > 0), помещаем прочтенную информацию в соответствующее окошко ввода.

```

void CSerialGateTestDlg::OnTimer(UINT nIDEvent)
{
    // TODO: Add your message handler code here and/or call default
    char buff[128];
    int rcv = sg.Recv(buff, sizeof(buff));
    if(rcv > 0)
    {
        for(int i = 0; i < rcv; i++)
            this->m_recieve += buff[i];
        UpdateData(false);
    }

    CDialog::OnTimer(nIDEvent);
}

```

Последняя функция нашего приложения. Обработчик на кнопку "Send". Проверяем, не нулевой ли длины сообщение, которое хотим отправить. Если это не так, то получаем адрес на буфер строки, запрятанной в MFC классе CString, и обычный шаровский буфер с его длиной отправляем в функцию Send(), которая все это запишет в COM-порт.

```

void CSerialGateTestDlg::OnSend()
{

```

```

// TODO: Add your control notification handler code here
UpdateData(true);
int len = this->m_send.GetLength();
if(len > 0)
{
    char* LocBuf = m_send.GetBuffer(128);
    sg.Send(LocBuf, len);
    m_send.ReleaseBuffer();
}
}

```

Проверяем программу в действии. Соединяем два порта кабелем и запускаем программы. Открываем нужные порты и ведем переписку через COM-порт (рис. 3.2).

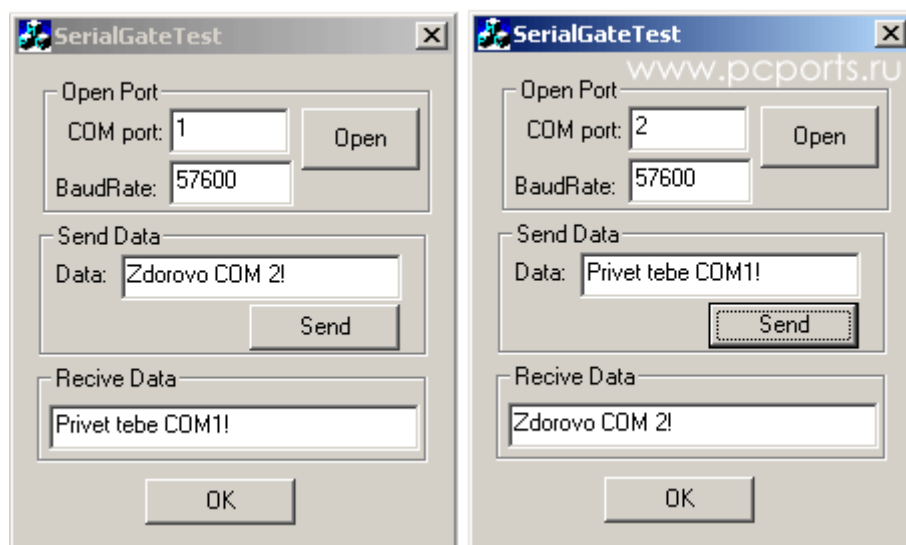


Рис. 3.2. Настройка COM-порта

Вопросы для усвоения

1. Что такое протокол Modbus, в каких линиях связи он используется?
2. Как осуществляется обмен данными при использовании Modbus протокола?

3. Какие режимы передачи данных предусмотрены в исследуемом протоколе, в чем их различие?
4. Назначение стартового и стоп бита.
5. Какие скорости передачи данных используются при работе с протоколом передачи данных Modbus?

Лабораторная работа № 4

РАБОТА С ПРИБОРОМ ИЗМЕРЕНИЯ ДАВЛЕНИЯ ЧЕРЕЗ ИНТЕРФЕЙС RS-485

Цель работы – приобретение практических навыков создания программ для работы с измерительными приборами посредством интерфейса RS-485.

Оборудование – ПК с ОС LINUX, преобразователь интерфейсов RS-485 / RS-232 ICP CON, блок питания БП-24-120-И, прибор контроля давления с интерфейсом RS-485 ПКД-1115 (с руководством по эксплуатации и применению коммуникационного интерфейса).

Введение

RS-485 – это номер стандарта, впервые принятого Ассоциацией электронной промышленности (EIA). Сейчас этот стандарт называется *TIA/EIA-485 Electrical Characteristics of Generators and Receivers for Use in Balanced Digital Multipoint Systems* (электрические характеристики передатчиков и приемников, используемых в балансных цифровых многоточечных системах).

Общепризнано, что RS-485 – это название популярного интерфейса, используемого в промышленных АСУТП для соединения контроллеров и другого оборудования. Главное отличие RS-485 от также широко распространенного RS-232 — возможность объединения нескольких устройств.

Интерфейс RS-485 обеспечивает обмен данными между несколькими устройствами по одной двухпроводной линии связи в полудуплексном режиме. Широко используется в промышленности при создании АСУ ТП.

RS-485 обеспечивает передачу данных со скоростью до 10

Мбит/с. Максимальная дальность зависит от скорости: при скорости 10 Мбит/с максимальная длина линии — 120 м, при скорости 100 кбит/с — 1200 м.

Количество устройств, подключаемых к одной линии интерфейса, зависит от типа примененных в устройстве приемопередатчиков. Один передатчик рассчитан на управление 32 стандартными приемниками. Выпускаются приемники с входным сопротивлением 1/2, 1/4, 1/8 от стандартного. При использовании таких приемников общее число устройств может быть увеличено соответственно: 64, 128 или 256.

Стандарт не нормирует формат информационных кадров и протокол обмена. Наиболее часто для передачи байтов данных используются те же фреймы, что и в интерфейсе RS-232: стартовый бит, биты данных, бит паритета (если нужно), стоповый бит.

Протоколы обмена в большинстве систем работают по принципу "ведущий"- "ведомый". Одно устройство на магистрали считается ведущим (master) и инициирует обмен посылкой запросов подчиненным устройствам (slave), которые различаются логическими адресами. Одним из популярных протоколов является протокол Modbus RTU.

Тип соединителей и распайка также не оговариваются стандартом. Встречаются соединители DB9, клеммные соединители и т.д.

На рис. 4.1 изображена локальная сеть на основе интерфейса RS-485, объединяющая несколько приемопередатчиков.

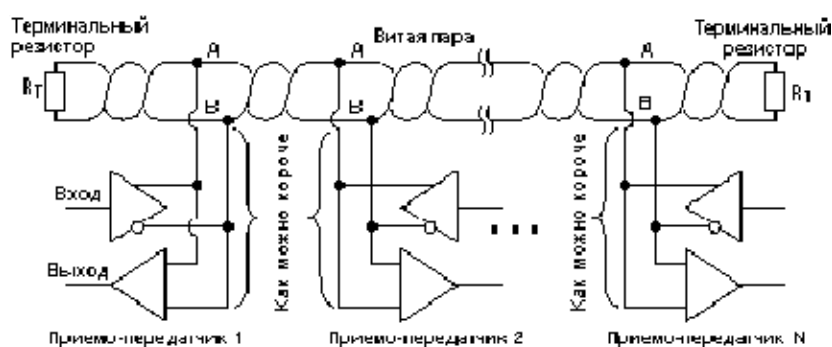


Рис. 4.1. Схема подключения по RS-485

При подключении следует правильно присоединить сигнальные цепи, обычно называемые А и В. Переполюсовка не страшна, но устройство работать не будет.

Общие рекомендации

- Лучшей средой передачи сигнала является кабель на основе *витой пары*.
- Концы кабеля должны быть заглушены *терминальными резисторами* (обычно 120 Ом).
- Сеть должна быть проложена по топологии шины *без ответвлений*.
- Устройства следует подключать к кабелю проводами *минимальной длины*.

Витая пара – оптимальное решение для прокладки сети, поскольку обладает наименьшим паразитным излучением сигнала и хорошо защищена от наводок. В условиях повышенных внешних помех применяют кабели с экранированной витой парой, при этом экран кабеля соединяют с защитной "землей" устройства.

Терминальные резисторы обеспечивают согласование "открытого" конца кабеля с остальной линией, устраняя отражение сигнала.

Номинальное сопротивление резисторов соответствует волновому сопротивлению кабеля и для кабелей на основе витой пары обычно составляет 100 – 120 Ом. Например, широко распространённый кабель UTP-5, используемый для прокладки Ethernet, имеет импеданс 100 Ом. Специальные кабели для RS-485 марки Belden 9841 ... 9844 – 120 Ом. Для другого типа кабеля может потребоваться другой номинал.

Резисторы могут быть запаяны на контакты кабельных разъемов у конечных устройств. Иногда резисторы бывают смонтированы в самом устройстве и для подключения резистора нужно установить перемычку. В этом случае при отсоединении устройства линия рассогласовывается, и для нормальной работы остальной системы требуется подключение согласующей заглушки.

Интерфейс RS-485 использует балансную (дифференциальную) схему передачи сигнала. Это означает, что уровни напряжений на сигнальных цепях А и В меняются в противофазе, как показано на рис. 4.2.

Передатчик должен обеспечивать уровень сигнала 1,5 В при максимальной нагрузке (32 стандартных входа и 2 терминальных резистора) и не более 6 В на холостом ходу. Уровни напряжений измеряют дифференциально: один сигнальный провод относительно другого.

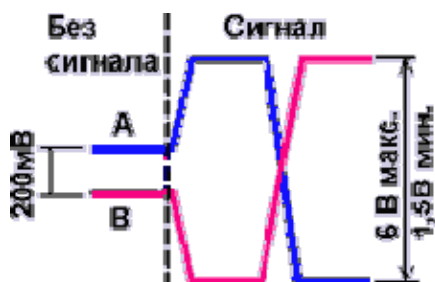


Рис. 4.2. Уровни сигналов в RS-485

Уровни напряжений измеряют дифференциально: один сигнальный провод относительно другого.

На стороне приемника RS-485 минимальный уровень принимаемого сигнала должен быть не менее 200 мВ.

Выполнение работы

Подключить прибор ПКД-1115 к ПК согласно схеме, приведенной в руководстве на данный прибор. Показать результат преподавателю.

Далее создать в терминале пустой файл, в котором будет создаваться программа для программирования прибора и считывания результатов измерения.

Получить у преподавателя задание по работе с прибором (перечень действий по записи/ считыванию данных из прибора, настройке режимов работы прибора и т.д.).

Строка обращения к прибору для считывания данных выглядит следующим образом:

```
modbusc -r32f --com /dev/ttyS0,115200,, --node 6 --reg d0h
```

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|

где: 1. Вызов программы modbusc, которая передает данные в COM-порт (пишется всегда).

2. `-r32f` (или `-r16h`) означает `-r` (read прочитать) 32 (16) бит данных в формате `f` – FLOAT (формат данных с плавающей точкой) `h` – HEX (шестнадцатеричный формат данных). Причем если читается или записывается FLOAT, то число бит всегда равно 32, так как число в этом формате занимает 2 регистра (значение до точки и после).

3. Обращение к СОМ-порту с номером 1 (СОМ1 соответствует адрес ...S0) на скорости 115200 (такая же должна быть установлена на приборе согласно РЭ) без контроля четности «,,» и без стоповых бит.
4. Адрес устройства в сети 6 (устанавливается на приборе в настройках «rs» по РЭ).
5. Номер требуемого регистра по регистровой карте прибора (см. описание коммуникационного интерфейса).

В случае записи данных в прибор в конце строки добавляются данные, которые необходимо передать в следующем виде: --data 4 (HEX) или -data 20.06 (FLOAT).

Примеры программ

1. Программа мониторинга результата измерения давления и значения токового выхода (0-20) мА:

```
#!/bin/sh
sl="sleep"
x="1"
clear
printf "Введите период регистрации (с)"
read p
while [ "$x" == "1" ]; do
# Проверка регистров 00D0h 000D1h
eval `modbusc -c -r32f --com /dev/ttyS0,115200,, --node 6 --reg d0h`;
printf "\r Измеряемое давление: \033[32;1m %s кПа \033[33;0m " $data
# Проверка регистров 00D4h 000D5h
eval `modbusc -c -r32f --com /dev/ttyS0,115200,, --node 6 --reg d4h`;
printf "Выходной аналоговый сигнал:\033[33;1m %s мА\033[33;0m"
$data
$sl $p
done
```

В программе используется бесконечный цикл:

```
x="1"
while [ "$x" == "1" ]; do
```

```
done
```

Работа осуществляется с заданной частотой:

```
printf "Введите период регистрации (с)"
```

```
read p
```

```
$sl $p
```

До прерывания пользователем (нажатием CTRL+C).

2. Программа отображения состояния дискретных выходов (реле):

```
#!/bin/sh
```

```
adr_dev="6";
```

```
sl="sleep"
```

```
echo "Состояние дискретных выходов:"
```

```
x="1"
```

```
while [ "$x" == "1" ]; do
```

```
eval `modbusc -c -r16h --com /dev/ttyS0,115200,, --node $adr_dev --reg  
E0h`; printf "%s " $data;
```

```
case $data in
```

```
0000) printf "\033[32;1m P1\033[33;0m * \033[32;1m P2\033[33;0m  
*\n";; # оба выключены
```

```
0001) printf "\033[32;1m P1\033[31;1m * \033[32;1m P2\033[33;0m  
*\n";; # Реле N1 ВКЛЮЧЕНО Реле N2 ВЫКЛЮЧЕНО
```

```
0002) printf "\033[32;1m P1\033[33;0m * \033[32;1m P2\033[31;1m  
*\n";; # Реле N1 ВЫКЛЮЧЕНО Реле N2 ВКЛЮЧЕНО
```

```
0003) printf "\033[32;1m P1\033[31;1m * \033[32;1m P2\033[31;1m  
*\n";; # Реле N1 и N2 ВКЛЮЧЕНЫ
```

```
*) printf "ОШИБКА!!! Неизвестное значение!!! \n";;
```

```
esac
```

```
$sl 1;
```

```
done
```

3. Тест регистров прибора:

```
#!/bin/sh
```

```
#12h
```

```
adr_dev="6";
```



```

povt="modbusc --com /dev/ttyS0,115200,, -d hex -c -w raw --data";
povtFLw="modbusc -c -w32f --com /dev/ttyS0,115200,, --node 6";
povtFLr="modbusc -c -r32f --com /dev/ttyS0,115200,, --node 6";
sl="sleep"
# проверка регистра 0005h #
#*****#
*****#
for a in {0..4}; do
$pv $adr_dev,6,0,5,0,$a;
$sl 1;
done
#*****#
*****#
# проверка регистра 0007h #
#*****#
*****#
echo "Изменение функции срабатывания реле N1 (регр 0007h)"
for b in {0..3}; do
$pv $adr_dev,6,0,7,0,$b;
#$pv $adr_dev,3,0,7,0,1
eval `modbusc -c -r16h --com /dev/ttyS0,115200,, --node $adr_dev --reg
7`; printf "%s " $data;
case $data in
0000) printf "реле N1 - выключено \n";;
0001) printf "реле N1 - ВКЛЮЧЕНИЕ по превышению уставки \n";;
0002) printf "реле N1 - ВЫКЛЮЧЕНИЕ по превышению уставки \n";;
0003) printf "реле N1 - срабатывание по ошибке измерения \n";;
*) printf "ОШИБКА!!!";;
esac
$sl 1;
done
$sl 1;
#*****#
*****#

```

```

# проверка регистра 0008h
#*****
*****#
echo "Функция срабатывания реле N1 при ошибке измерения (регистр
0008h)"
for c in {0..2}; do
$pvvt $adr_dev,6,0,8,0,$c;
eval `modbusc -c -r16h --com /dev/ttyS0,115200,, --node $adr_dev --reg
8`; printf "%s " $data;
case $data in
0000) printf "ВЫКЛЮЧИТЬ (off) \n";;
0001) printf "ВКЛЮЧИТЬ (on) \n";;
0002) printf "Не изменять состояние (Hold) \n";;
*) printf "ОШИБКА!!!";;
esac
$sl 1;
done
#*****
*****#
# проверка регистра 0009h
#*****
*****#
echo "Задержка срабатывания реле N1"
for c in {255..0}; do
$pvvt $adr_dev,6,0,9,0,$c
eval `modbusc -c -r16h --com /dev/ttyS0,115200,, --node $adr_dev --reg
8`;
case $er in
0) printf "Установлена задержка %s секунд\n" $c ;;
*) printf "ОШИБКА!!!";;
esac
done
$sl 1;
#*****

```

```

*****#
# проверка регистра 000Ah, 000Bh
#*****
*****#
echo "Порог срабатывания реле N1 (регистры 000Ah 000Bh)"
echo "Установка порога срабатывания реле N1 = 2,000"
$pvvtFLw --reg 0Ah --data 2.0
echo "Проверка настройки"
$pvvtFLr --reg 0Ah; printf "%s"
$sl 1;
#*****
*****#
# Проверка регистров 000Ch, 000Dh
#*****
*****#
echo "Гистерезис срабатывания реле N1"
echo "Установленный гистерезис"
$pvvtFLr --reg 0Ch; printf "%s"
echo "Установка гистерезиса реле N1 = 0.010"
$pvvtFLw --reg 0Ch --data 0.010
echo "Проверка настройки гистерезиса"
$pvvtFLr --reg 0Ch; printf "%s"
$sl 1
#*****
*****#
# Проверка регистра 000Eh
#*****
*****#
echo "Функция срабатывания реле N2"
for f in {0..3}; do
$pvvt $adr_dev,6,0,Eh,0,$f
eval `modbusc -c -r16h --com /dev/ttyS0,115200,, --node $adr_dev --reg
Eh`; printf "%s " $data;
case $data in

```

```

0000) printf "реле N2 - выключено \n";;
0001) printf "реле N2 - ВКЛЮчение по превышению уставки \n";;
0002) printf "реле N2 - ВЫКЛЮчение по превышению уставки \n";;
0003) printf "реле N2 - срабатывание по ошибке измерения \n";;
*) printf "ОШИБКА!!!";;
esac
$sl 1;
done
$sl 1
#*****
*****#
# проверка регистра 000Fh
#*****
*****#
echo "Функция срабатывания реле N2 при ошибке измерения (регр
000Fh)"
for c in {0..2}; do
$svt $adr_dev,6,0,Fh,0,$c;
eval `modbusc -c -r16h --com /dev/ttyS0,115200,, --node $adr_dev --reg
Fh`; printf "%s " $data;
case $data in
0000) printf "ВЫКЛЮчить (off) \n";;
0001) printf "ВКЛЮчить (on) \n";;
0002) printf "Не изменять состояние (Hold) \n";;
*) printf "ОШИБКА!!!";;
esac
$sl 1;
done
#*****
*****#
# проверка регистра 0010h
#*****
*****#
echo "Задержка срабатывания реле N2"

```

```

for c in {255..0}; do
$pvot $adr_dev,6,0,10h,0,$c
eval `modbusc -c -r16h --com /dev/ttyS0,115200,, --node $adr_dev --reg
10h`;
case $er in
0) printf "Установлена задержка %s секунд\n" $c ;;
*) printf "ОШИБКА!!!";;
esac
done
$sl 1;
#####
#####
# проверка регистров 0011h, 0012h
#####
#####
echo "Порог срабатывания реле N2 (регистры 000Ah 000Bh)"
echo "Текущая уставка"
$pvotFLr --reg 11h; printf "%s"
echo "Установка порога срабатывания реле N2 = 8.000"
$pvotFLw --reg 11h --data 8.0
echo "Проверка настройки"
$pvotFLr --reg 11h; printf "%s"
$sl 1;
#####
#####
# Проверка регистров 0013h, 0014h
#####
#####
echo "Гистерезис срабатывания реле N2"
echo "Установленный гистерезис"
$pvotFLr --reg 13h; printf "%s"
echo "Установка гистерезиса реле N2 = 0.010"
$pvotFLw --reg 13h --data 0.010
echo "Проверка настройки гистерезиса"

```

```

$pvotFLr --reg 13h; printf "%s"
$sl 1;
#*****
*****#
# Проверка регистра 0070h
#*****
*****#
echo "Число усредняемых измерений"
for c in {30..0}; do
$pvot $adr_dev,6,0,70h,0,$c
eval `modbusc -c -r16h --com /dev/ttyS0,115200,, --node $adr_dev --reg
70h`;
case $er in
0) printf "Установлено усреднение %s \n" $c ;;
*) printf "ОШИБКА!!!";;
esac
done
$sl 1;
#*****
*****#
# Проверка регистра 0071h
#*****
*****#
echo "Ускоритель фильтра"
for c in {0..1}; do
$pvot $adr_dev,6,0,71h,0,$c;
eval `modbusc -c -r16h --com /dev/ttyS0,115200,, --node $adr_dev --reg
71h`; printf "%s " $data;
case $data in
0000) printf "Выключен (off) \n";;
0001) printf "Включен (on) \n";;
*) printf "ОШИБКА!!!";;
esac
$sl 1;
Done

```

Вопросы для усвоения

1. В чем отличия стандартов RS-485 и RS-232, каково максимальное расстояние передачи при использовании данных стандартов?
2. Что такое волновое сопротивление, назначение терминального резистора?
3. Какое количество устройств можно объединить в измерительную сеть при использовании стандарта передачи RS-485, что для этого требуется?
4. Что такое дифференциальная схема передачи сигнала?
5. Что такое FLOAT формат данных?
6. Принцип записи строки обращения к прибору (выбор порта, адрес устройства, скорость передачи данных и т.д.).

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Сопряжение датчиков и устройств ввода данных с компьютерами IBM PC / под ред. У.Томпкина, Дж. Уэбстера; пер. с англ. – М.: Мир, 1992. – 592 с.
2. Новиков, Ю. В. Разработка устройств сопряжения для персонального компьютера типа IBM PC: практ. пособие / Ю. В. Новиков, О. А. Калашников, С. Э. Гуляев ; ред. Ю. В. Новикова. – М.: ЭКОМ., 1997. – 224 с. – ISBN 5-7163-0009-х.
3. Ан, П. Сопряжение ПК с внешними устройствами / П. Ан; пер. с англ. – М.: ДМК Пресс, 2003. – 320 с. – ISBN 5-94074-076-6.
4. Бычков, Е. А. Архитектуры и интерфейсы персональных компьютеров / Е. А. Бычков. – М.: Центр "СКС", 1993. – 152 с.
5. Интерфейсы систем обработки данных: справочник / А. А. Мячев, В. Н. Степанов, В. К. Щербо; под ред. А. А. Мячева. – М.: Радио и связь, 1989. – 416 с.
6. Гук. М. Аппаратные интерфейсы ПК: энцикл. / М. Гук. – СПб.: Питер, 2002. — 528 с. – ISBN 5-94723-180-8.

Оглавление

| | |
|---|----|
| Введение..... | 3 |
| Лабораторная работа № 1. Параллельный интерфейс. | |
| Работа с LPT-портом..... | 4 |
| Лабораторная работа № 2. Последовательный интерфейс. | |
| Работа с COM-портом | 12 |
| Лабораторная работа № 3. Чтение / запись данных в COM-порт. Протокол MODBUS | 21 |
| Лабораторная работа № 4. Работа с прибором измерения давления через интерфейс RS-485 | 27 |
| Список рекомендуемой литературы..... | 39 |

ИНТЕРФЕЙСЫ ИЗМЕРИТЕЛЬНЫХ УСТРОЙСТВ

Методические указания к лабораторным работам

Составители

Макарова Наталья Юрьевна
Павлов Дмитрий Дмитриевич

Ответственный за выпуск – зав. кафедрой профессор В. П. Легаев

Подписано в печать 24.10.11.

Формат 60x84/16. Усл. печ. л. 2,32. Тираж 60 экз.

Заказ

Издательство

Владимирского государственного университета
имени Александра Григорьевича и Николая Григорьевича Столетовых
600000, Владимир, ул. Горького, 87.