

Глава 1

Компьютер управляет внешними аппаратами

Предисловие

Слово «робот» пришло к нам из фантастической литературы и стало обозначать чудесный аппарат, наделенный необыкновенными возможностями.

Собственно говоря, сам персональный компьютер уже является подобным чудодейственным аппаратом, способным выполнять такую работу, которая еще несколько лет тому назад считалась невыполнимой в домашних условиях.

В этой книге я не буду рассказывать о том, как сделать робота. Об этом написано уже много различных книг, да и в рамках одной главы невозможно раскрыть тему создания робота, например, на базе микроконтроллера PIC16xxxx фирмы *Microchip*.

Далее будут описаны методы, которыми следует пользоваться при изготовлении очень простых аппаратов, управляемых непосредственно домашним компьютером. Изложенный в этой книге материал должен оказать вам помощь при изучении более сложных вопросов.

Рассмотрим подробнее эти методы.

- Компьютер может управлять каким-то внешним аппаратом только с помощью заложенной в этот компьютер программы. Поэтому первым методом, который должен быть использован при создании управляемого компьютером аппарата, является создание необходимой компьютерной программы. Следовательно, в этой книге должно быть рассказано о том, как можно быстро создать такую программу.
- Компьютер является электронным аппаратом, созданным на базе электро- и радиотехники. Следовательно, в этой книге должны быть показаны примеры создания принципиальных электрических схем аппаратов, управляемых компьютером посредством электрических цепей и компонентов.

Сначала, для учебных целей, рассмотрим пример создания очень простого устройства, которое должно управляться компьютером.

Для работы с подключаемыми к компьютеру (IBM PC) внешними устройствами служат COM-порты (порты для последовательной передачи данных), LPT-порты (порты для параллельной передачи данных), USB-порты (универсальные порты для передачи данных блоками) и звуковая карта компьютера.

Работа с внешними устройствами посредством звуковой карты будет рассмотрена далее, в одной из глав этой книги, работа с USB-портами представляет пока большую сложность даже для профессионалов, и поэтому рассматриваться не будет, но в приложении 3 вы сможете найти очень краткую информацию по USB-портам.

Довольно подробная информация об устройстве COM-портов и LPT-портов изложена в приложении 3. О работе с внешними устройствами через LPT-порт вы найдете информацию в последующих главах книги. В настоящей главе этой книги рассмотрим основные принципы работы с простыми внешними устройствами через COM-порт.

Простое управляемое устройство

Обычно компьютер оснащен одним или двумя портами последовательной передачи данных. Эти порты расположены, как правило, на материнской плате компьютера. Каждому COM-порту соответствует несколько регистров, через которые компьютерная программа может управлять портом, и определенная линия прерываний IRQ для сигнализации компьютеру о состоянии порта.

Последовательная передача данных означает, что данные передаются бит за битом по единственной линии связи. Формат передачи данных для передающего и приемного устройств должен быть один и тот же, иначе передача данных будет невозможна. Формат передачи данных определяют стартовый и стоповый биты, бит четности и скорость передачи.

Основную роль в последовательной передаче данных выполняет асинхронный адаптер, который может управлять сразу несколькими COM-портами. Основу адаптера составляет микросхема универсального асинхронного приемопередатчика UART (Universal Asynchronous Receiver Transmitter). Встречаются несколько разновидностей этой микросхемы — Intel 8250, 16450, 16550, 16550A. При этом часть из них уже безнадежно устарела.

Для каждого COM-порта указанные микросхемы содержат регистры передатчика и приемника данных, а также несколько управляющих регистров, доступных через команды ввода/вывода.

При передаче очередной байт записывается в буферный регистр передатчика, откуда затем переписывается в сдвиговый регистр. Далее байт «выдвигается» из сдвигового регистра по битам. Аналогично работают сдвиговый и буферный регистры приемника.

Программа имеет доступ только к буферным регистрам. Копирование информации в сдвиговые регистры и сдвиг данных выполняется микросхемой UART автоматически. Регистры, управляющие асинхронным последовательным портом, будут описаны ниже.

Внешне каждый COM-порт асинхронного последовательного адаптера представлен собственным разъемом. Существует два стандарта на разъемы COM-порта: DB25 и DB9. Первый разъем имеет 25, а второй 9 выводов. Несмотря на то, что разъем DB25 содержит в два с половиной раза больше выводов, чем DB9, они передают одинаковые сигналы. При необходимости можно приобрести (или сделать самостоятельно) переходник между разъемами DB25 и DB9. Подробности смотрите в приложении 3.

Порт последовательной передачи данных предназначен для широко использования. К нему может быть подключен манипулятор мышь, модем, сканер, графопостроитель, принтер и т.п. Все эти аппараты и устройства через COM-порт проводят обмен данными с компьютером,

используя при этом стандартные принципы последовательной передачи данных, заложенные в конструкции порта.

Но в то же время радиолюбителями для своих конструкций часто применяются *нестандартные* (или *нетрадиционные*) методы использования COM-порта для управления внешними, подключенными к компьютеру, устройствами. Сущность этого метода заключается в том, что программным путем компьютер устанавливает в регистрах порта те или иные биты (ячейки), состояние которых вызывает у подключенного к COM-порту внешнего устройства необходимые действия.

Нетрадиционное использование COM-порта можно успешно осуществлять только на ПК, которые работают под управлением операционных систем Microsoft Windows 95/98/ME. Под управлением операционных систем Microsoft Windows 200/XP этот вариант не работает. В то же время программы с нетрадиционным применением управления COM-портами, созданные в операционной системе MS DOS, работают и под управлением Windows 2000/XP.

В качестве примера нетрадиционного управления внешним устройством рассмотрим конструкцию устройства, принципиальная электрическая схема которой показана на рис. 1.1.

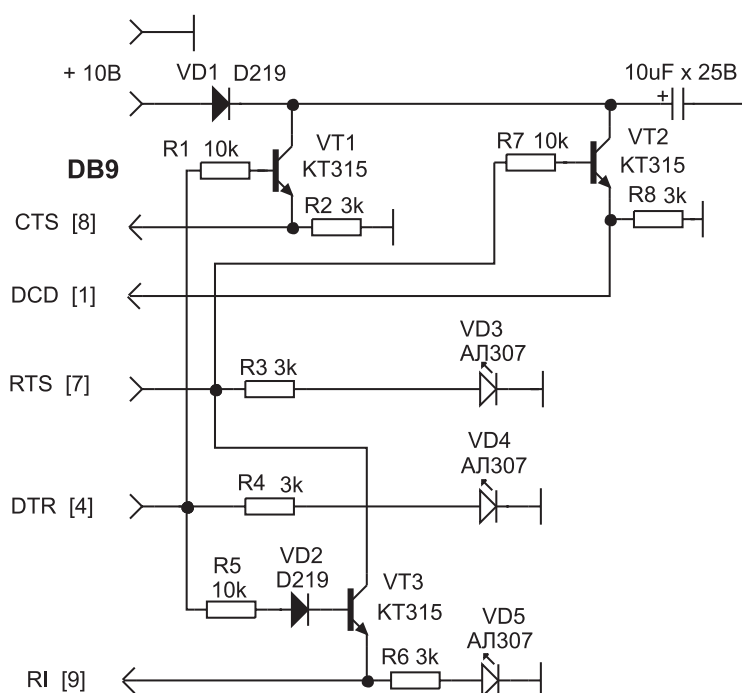


Рис. 1.1. Схема простейшего внешнего устройства

Устройство представляет собой изделие, которое не имеет практического применения и которое создавалось только в учебных целях для иллюстрации принципов нетрадиционного управления внешним устройством через COM-порт.

Слева на схеме рис. 1.1 показаны сокращенные названия соответствующих выводов (линий) COM-порта и в квадратных скобках расположены номера этих выводов для разъема DB9. Подробную информацию о выводах смотрите в приложении 3.

По линиям TD, RTS и DTR на устройство поступают сигналы от СОМ-порта. Это выходные сигналы, направленные от компьютера к внешнему устройству.

Линия RTS при стандартном применении служит для выдачи сигнала «Запрос передачи». При нетрадиционном использовании состояние этой линии устанавливается программно, в не зависимости от работы других линий.

Линия DTR при стандартном применении служит для выдачи сигнала «Готовность выходных данных». При нетрадиционном использовании состояние этой линии устанавливается программно, в не зависимости от работы других линий.

По линиям CTS, DCD и RI сигналы подаются от устройства к СОМ-порту. Это входные сигналы.

Линия CTS при стандартном применении служит для приема сигнала «Сброс для передачи». При нетрадиционном использовании состояние этой линии устанавливается устройством и поступает на СОМ-порт, в зависимости от принципов работы этого внешнего устройства.

Линия DCD при стандартном применении служит детектором принимаемого с линии сигнала. При нетрадиционном использовании состояние этой линии устанавливается устройством и поступает на СОМ-порт, в зависимости от принципов его работы.

Линия RI при стандартном применении служит индикатором вызова. При нетрадиционном использовании состояние этой линии устанавливается устройством и поступает на СОМ-порт, в зависимости от принципов работы этого устройства.

Питается устройство от маломощного источника постоянного тока напряжением 10–12 В.

Программа для работы простого устройства

Для управления простым устройством создадим компьютерную программу. Программа будет называться `s_port` и создавать ее будем в среде программирования Borland C++ Builder. Краткую информацию об этой среде программирования можно найти в приложении 2, а научиться программированию в среде Borland C++ Builder поможет книга «Быстрое программирование на C++» [3]. Для создания программы `s_port` можно применять любую версию указанной среды программирования.

Если у вас нет возможности самому создавать проект программы, то можете воспользоваться готовым исполняемым вариантом программы, который находится на прилагаемом к книге компакт-диске.

Изготовленное по схеме рис. 1.1 устройство должно быть подключено к соответствующему порту до начала экспериментов с программой `s_port`.

Следует всегда помнить, что подключение внешнего устройства к любому из портов компьютера следует проводить только при полностью выключенном компьютере.

Рабочее окно программы показано на рис. 1.2.

В верхнем левом углу окна располагается кнопка **Проверка СОМ-портов**, которая служит для определения всех СОМ-портов, находящихся в данном компьютере. Результаты проверки выводятся в расположенном ниже многострочном окне редактирования.

Номер COM-порта выбранного для работы следует ввести в поле COM, над которым располагается надпись **Выбор COM-порта**.

Для продолжения работы с программой следует нажать кнопку **Открыть COM-порт**. Если выбранный вами COM-порт будет успешно открыт, то кнопка изменит цвет и название на **COM-порт открыт**. Помните, что открытым этот порт будет постоянно до нажатия кнопки **Выход**.

Теперь можно приступить к экспериментам.

Нажимая кнопки **Установить RTS**, **Установить DTR** и **Установить RTS+DTR** можно наблюдать возникновение и пропадание свечения светодиодов VD3, VD4 и VD5 (см. рис. 1.1). При этом VD5 должен светиться только тогда, когда одновременно установлены сигналы и на линии RTS и на линии DTR. В этом случае срабатывает логический элемент И, выполненный на транзисторе VT3.

Очистить линии RTS и DTR можно с помощью кнопки **Сброс RTS и DTR**. Транзисторы VT1 и VT2 работают как электронные ключи и выдают на COM-порт сигналы CTS и DCD, которые сигнализируют об установке сигналов на линиях RTS и DTR.

Для проверки наличия входных сигналов предназначена кнопка **Проверка входных линий CTS, DCD и RI**. Если на одной из проверяемых линий появился сигнал, то ниже этой кнопки появляется соответствующее сообщение.

Ход выполнения всех операций отображается в многострочном окне редактирования, расположенном в левом нижнем углу рабочего окна программы.

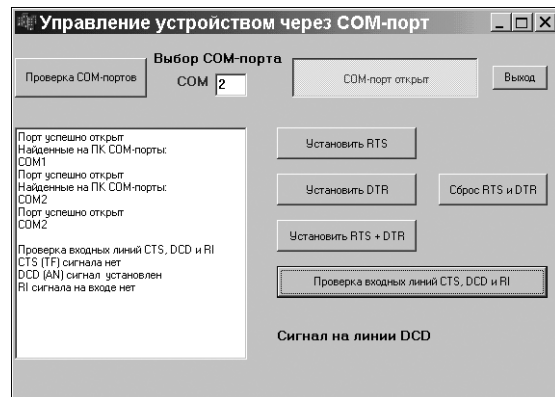


Рис. 1.2. Рабочее окно программы

Создаем проект программы

Учитывая то, что не каждый из читателей знаком с процессом создания проекта Windows-приложения в среде Borland C++ Builder, перечислим все этапы создания нового проекта, которые следует твердо выучить и запомнить.

1. Перед началом создания рабочего проекта приложения нужно создать для этого проекта отдельную директорию (папку). Это может быть заранее созданная директория, например, d:\moi_proekty\S_port\.

Запускаем C++ Builder, выбираем команду **File⇒New⇒Application**. На экране появляется готовая к заполнению форма и все прочие окна, необходимые для нового проекта. Появляется и название проекта — **Proekt1**.

2. Поскольку нас не устраивает название проекта, следует проект сохранить в заранее созданную директорию и под назначенным заранее названием проекта. Для этого выберем **File⇒Save Proekt As**. Основные файлы проекта сохраняются поочередно. Первым C++ Builder сохраняет основной функциональный файл проекта и просит ввести имя этого файла. Введем имя файла «sport» и сохра-

ним файл под этим названием. Затем C++ Builder предлагает ввести имя файла проекта. Еще раз предупреждаю, что имена сохраняемого первым функционального файла и файла проекта должны различаться. В таких случаях используется следующий прием: для имени проекта возьмем имя функционального файла, при этом добавим к этому имени цифру, обозначающую вариант программы. Вы можете выбирать любые удобные для вас имена. Итак, сохраняем файл проекта под именем «s_port11».

3. После того, как проект сохранен в определенной директории, приступаем к заполнению формы проекта. Когда вы будете разрабатывать собственную программу, следует предварительно на листе бумаги сделать эскиз формы с размещенными на ней компонентами. В процессе разработки программы количество и размещение компонентов могут значительно меняться, но предварительный эскиз нужен обязательно.
4. На панели компонентов должна быть установлена страница библиотеки компонентов **Standard** (Стандартная).

Теперь можно, если это кому-то интересно, заглянуть в директорию d:\moi_proekty\s_port\.. Здесь вы увидите созданные программой файлы нашего нового проекта.

- s_port11.bpr — главный файл проекта — текстовый файл, содержащий всю необходимую информацию для компилятора, используемую при трансляции и сборке программных модулей.
- s_port11.cpp — главный программный файл — текстовый файл, содержащий список всех используемых программных модулей, а также стандартную для всех программ на языке C++ функцию WinMain(), обеспечивающую запуск приложения в работу. Этот файл создается и постоянно контролируется средой C++ Builder автоматически.
- s_port11.res — файл ресурсов — в нем хранятся некоторые данные, например, выводимые на панель задач.
- sport_a.cpp — файл программного модуля — в нем будут находиться исходные коды всех функций на языке C++.
- sport_a.h — заголовочный файл — в нем содержатся интерфейсные части формы, выполняется объявление задействованных компонентов и функций класса формы.
- sport_a.dfm — файл описания форм — содержатся описания форм и расположенных на них компонентов, запоминаются начальные значения свойств, установленных в инспекторе объектов.

Пока все эти файлы содержат только самый необходимый для проекта минимум информации. Начать разработку проекта следует с заполнения (разработки) формы.

Предлагаемый мною вариант оформления формы показан на рис. 1.3.

На странице компонентов **Standard** вы найдете все необходимые компоненты, кроме компонентов Speed Button, которые находятся на странице **Additional** (Дополнительно).

В листинге 1.1 приведен текст заголовочного файла `sport_a.h`, который пригодится вам при определении названий компонентов, установленных на форме в предлагаемом автором варианте.

Листинг 1.1. Файл `sport_a.h`

```
//-----
#ifndef sport_aH
#define sport_aH
//-----
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Buttons.hpp>
//-----
class TForm1 : public TForm
{
__published: // IDE-managed Components
    TButton *Button1;
    TLabel *Label1;
    TLabel *Label2;
    TEdit *Edit1;
    TSpeedButton *SpeedButton1;
    TButton *Button2;
    TMemo *Memo1;
    TSpeedButton *SpeedButton2;
    TSpeedButton *SpeedButton3;
    TSpeedButton *SpeedButton4;
    TLabel *Label3;
    TLabel *Label4;
    TLabel *Label5;
    TSpeedButton *SpeedButton5;
    TButton *Button3;
    void __fastcall Button1Click(TObject *Sender);
    void __fastcall Edit1Change(TObject *Sender);
    void __fastcall SpeedButton1Click(TObject *Sender);
    void __fastcall Button2Click(TObject *Sender);
    void __fastcall SpeedButton2Click(TObject *Sender);
    void __fastcall SpeedButton3Click(TObject *Sender);
    void __fastcall SpeedButton4Click(TObject *Sender);
    void __fastcall SpeedButton5Click(TObject *Sender);
    void __fastcall Button3Click(TObject *Sender);
};
```

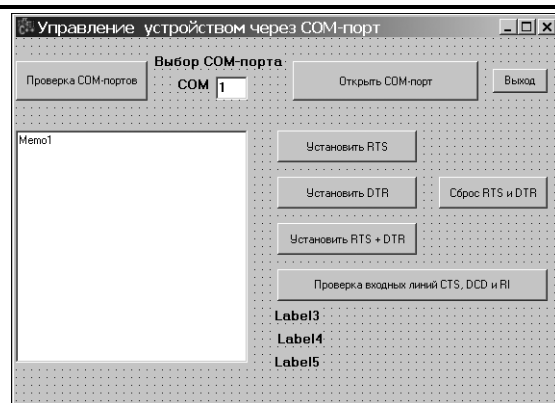


Рис. 1.3. Вариант оформления формы

```
private:      // User declarations
public:       // User declarations
    __fastcall TForm1(TComponent* Owner);
};

extern PACKAGE TForm1 *Form1;

#endif
//-----
```

Рассмотрим подробно содержание этого файла, что может пригодиться начинающим. В дальнейшем этого делать не будем.

Файл `sport_a.h` создается средой программирования автоматически, вмешательство в него должно быть только очень осмысленным.

Первые две строки совместно с последней строкой файла относятся к директивам препроцессора и задают начало и конец подключаемого заголовочного файла `sport_a.h`.

Две наклонных черты `«//»` являются символом начала строки с пояснениями (комментариями). Пояснения могут также ограничиваться символами `/*` и `*/`.

Следующие пять строк, начинающихся с текста `#include` равнозначны понятию «подключение» и дают указание компилятору подключить к работе над программой указанные далее в угловых скобках специальные «подключаемые заголовочные файлы». Эти подключаемые файлы содержат описания задействованных в данной программе функций и находятся в составе C++ Builder в директории INCLUDE.

Если подключаемый файл находится в директории разрабатываемого проекта, то он вместо угловых скобок заключается в двойные кавычки `"..."`.

Словами `class TForm1 : public TForm` начинается «объявление» класса `TForm1`. Все, что далее находится между фигурными скобками, относится к этому классу. Первыми членами этого класса объявляются установленные нами ранее на форму компоненты типа `Label` и `Button`, затем объявляются функции, вызванные событиями этих компонентов.

Смысл понятия «объявление» заключается в том, что этим действием для объявляемого объекта выделяется необходимый участок памяти и организуется адресный указатель на этот участок.

Функции — члены класса `TForm1` — в этом файле только объявляются, а вот описывать эти функции, т.е. наполнять их соответствующими кодами мы будем уже при создании файла `sport_a.cpp`.

Описание функций в файле `sport_a.cpp`

После запуска в работу Borland C++ Builder в составе главного рабочего окна программы будет выведена также и рабочая форма `Form1`. Двойной щелчок на любом, незаполненном компонентами участке формы, вызывает на экран окно редактора текста. При этом курсор будет находиться внутри функции, предназначенной для описания события, совершаемого при выводе на экран `Form1`. В этом, обозна-

ченном курсором участке, необходимо ввести тексты, в которых задается название главного рабочего окна программы и состояние текстов в некоторых компонентах. Эти и другие тексты, соответствующие выбранным компонентам, смотрите в листингах файла `sport_a.cpp`.

Рисунки с изображением окна редактирования можно увидеть в приложении 2 на рис. П2.2 и рис. П2.3.

Если установить курсор в окне редактирования `Edit1`, предназначенном для выбора СОМ-порта, и дважды щелкните мышью, то на экране откроется редактор текста с открытым в нем файлом `sport_a.cpp`. При этом курсор будет находиться внутри функции, предназначенной для описания события, вызванного окном редактирования `Edit1`. Точно таким же образом следует поступать при вводе содержания текста функций, относящихся к событиям всех остальных компонентов.

Рассмотрим файл `sport_a.cpp`, начало которого располагается в листинге 1.1, несколько подробнее.

Начинается он с подключения файла `<vcl.h>`, который является главным файлом в описаниях всех задействованных в C++ Builder библиотек компонентов. Строки, начинающиеся с выражения `#pragma`, являются специфическими для C++ Builder служебными строками и никаким изменениям или удалению подвергаться не должны. Строка `#pragma inline` предлагает компилятору работать со строками исходного текста, написанными на языке Ассемблера. Эта строка должна быть введена вами.

Далее подключается рассмотренный нами ранее файл `"sport_a.h"`, а также подключаемые файлы `<conio.h>` и `<stdio.h>` — это файлы с описаниями функций, необходимых для работы с портами и строками. Вы должны вписать две строки, в которых выполняется подключение двух последних файлов.

Строкой `TForm1 *Form1` начинается объявление задействованных в программе компонентов и переменных.

Листинг 1.2. Файл `sport_a.cpp`

```
//-----
#include <vcl.h>
#pragma hdrstop
#pragma inline
#include "sport_a.h"
#include <conio.h>
#include <stdio.h>
//-----
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
TMemo *Memo1;
TLabel *Label3;
TLabel *Label4;
TLabel *Label5; int i_prt;
```

```
int prt1, prt2, prt3;
//-----
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
    Caption = "Управление устройством через COM-порт";
    Memo1->Clear();
    Label3->Caption = "";
    Label4->Caption = "";
    Label5->Caption = "";
    SpeedButton1->AllowAllUp = true;
    SpeedButton1->GroupIndex = 1;
    SpeedButton2->AllowAllUp = true;
    SpeedButton2->GroupIndex = 2;
    SpeedButton3->AllowAllUp = true;
    SpeedButton3->GroupIndex = 2;
    SpeedButton4->AllowAllUp = true;
    SpeedButton4->GroupIndex = 2;
    SpeedButton5->AllowAllUp = true;
    SpeedButton5->GroupIndex = 2;
}
//-----
```

Строка `__fastcall TForm1::TForm1(TComponent * Owner)` является заголовком функции, которая будет выполняться первой при запуске в работу программы `s_port`. Первая строка в теле этой функции задает название главному рабочему окну программы, вторая строка очищает текстовое окно компонента `Memo1`, а три следующих строки убирают с экрана названия компонентов типа `Label`.

Последние десять строк задают свойства компонентов `SpeedButton`, необходимые для их совместной работы в группе. Подробную информацию по этому вопросу можно найти в [3].

Продолжение файла `sport_a.cpp` смотрите в листинге 1.2.

Строки, начинающиеся с выражения `void __fastcall`, являются функциями, каждая из которых выполняет определенные для нее действия, записанные в «теле» этой функции в виде программных кодов. Для удобства обращения, пронумеруем все функции, описанные в этом файле определенной цифрой в сочетании с символами начала комментариев, например `//№..`. Эти номера будут находиться справа от заголовка каждой из функций.

Для продолжения работы нажимаем клавишу `<F12>` для перехода в режим работы с формой.

Дважды щелкните на кнопке **Выход** и появится курсор ввода команд для функции № 1, которая должна прекращать работу нашего приложения и выполнить выход в окно рабочего стола операционной системы Microsoft Windows.

Команда в данном случае очень простая — пишем `Close()`;

Снова выводим на экран форму. Двойной щелчок в окне редактирования `Edit1` вызовет на экран текстовый редактор, при этом курсор будет установлен между двумя фигурными скобками, предлагая ввести в это место соответствующую команду. Мы так и сделаем, введем тексты команд, записанные в листинге 1.2 в теле функции под № 2, которые начинаются со строки:

```
if(Edit1->Text == "")
ShowMessage("Введите номер COM-порта!");
```

Далее подобным образом вводим текст функции № 3. Эта функция в готовой программе начинает работать после нажатия на кнопку **Открыть COM-порт** и выполняет открытие заданного COM-порта в данном компьютере. Результаты проверки выдаются в многострочном окне редактирования, расположенном ниже кнопки **Проверка COM-портов**.

Итак, мы заполнили кодами описание третьей функции. Нажимаем клавишу <F12> и переходим в режим работы с формой.

Листинг 1.3. Продолжение 1 файла *sport_a.cpp*

```
//-----
void __fastcall TForm1::Button1Click(TObject *Sender)      // 1
{
    Close();
}
//-----
void __fastcall TForm1::Edit1Change(TObject *Sender)      // 2
{
    if(Edit1->Text == "")
        ShowMessage("Введите номер COM-порта!");
    else
        i_prt = StrToFloat(Edit1->Text);    // заполняем переменную i_prt
}
//-----
void __fastcall TForm1::SpeedButton1Click(TObject *Sender) // 3
{
    int i_prt = StrToInt(Edit1->Text);    // Заполняем переменную i_prt
    HANDLE hPort;                        // Объявляем идентификатор порта
    char sPort[10];                      // Буфер для наименования порта
    if(Edit1->Text == "")                // Если окно редактирования пустое,
        ShowMessage("Введите номер COM-порта"); // выводится сообщение
    else
    {
        if(SpeedButton1->Down)          // Если нажата кнопка проверки, то
        {
```

```
        char sPort[10];
        // Создаем имя для выбранного порта
        sprintf(sPort, "COM%d", i_prt);
        // Пытаемся открыть порт с новым названием
        HANDLE hPort = ::CreateFile(sPort, GENERIC_READ | GENERIC_WRITE,
0, 0, OPEN_EXISTING, 0, 0);
        if(hPort == INVALID_HANDLE_VALUE) // если не открывается
        {
            DWORD dwError = GetLastError();
            // Смотрим, что получилось при открытии
            if(dwError == ERROR_ACCESS_DENIED || dwError == ER-
ROR_GEN_FAILURE)
{Mem01->Lines->Add("Порт не может быть открыт"); // Вывод сообщения
        }
        }
        else
        {
            // Порт открыт успешно
            Mem01->Lines->Add("Порт успешно открыт"); // Сообщение
            Mem01->Lines->Add(sPort); // Имя порта
            SpeedButton1->Font->Color = clBlue; // Цвет надписи
            SpeedButton1->Caption = "COM-порт открыт"; // Сообщение
        }
    }
}
//-----
```

Когда на форме имеется окно редактирования, в котором при работе программы нужно вводить различные величины, возникают моменты, когда при замене одной величины другой величиной окно редактирования на какое-то время оказывается пустым. С++ Builder этого не любит и выдает сообщение об ошибке. Чтобы предотвратить подобные сообщения, можно «узаконить» появление пустого пространства в окнах редактирования. Для этих целей используется оператор `if ... else`, начало которого записано в первой строке функции №3.

Следующая строка выводит на экран окно сообщения с текстом «Введите номер COM-порта».

Расположенное в третьей строке слово `else` (Иначе) обозначает, что если условия, описанные в первой строке не выполнены (т.е. в окне редактирования есть тексты), то будут выполняться все программные строки, начиная с четвертой строки, т.е. все строки, находящиеся между фигурными скобками, следующими за `else`.

В листинге 1.3 приведен текст функции № 4, которая выполняется после нажатия кнопки **Проверка COM-портов** и которая поочередно проверяет наличие всех COM-портов, задействованных в данном компьютере.

Листинг 1.4. Продолжение 2 файла *sport_a.cpp*

```

//-----
void __fastcall TForm1::Button2Click(TObject *Sender)           // 4
{
    Mem1->Clear();                // Стираем все старые записи
    Mem1->Lines->Add("Поиск имеющихся на ПК COM-портов.");
    for (UINT i=1; i<256; i++) // Поиск возможных 256 портов
    {
        // Формируем предполагаемое имя устройства
        char sPort[10];
        sprintf(sPort, "COM%d", i); // Создаем имя порта с цифрой
        // Пытаемся открыть каждый из 256 возможных портов
        BOOL bSuccess = FALSE;      // Переменная состояния порта
        HANDLE hPort = ::CreateFile(sPort, GENERIC_READ | GE-
        NERIC_WRITE, 0, 0, OPEN_EXISTING, 0, 0); // Открываем очередной порт
        if(hPort == INVALID_HANDLE_VALUE) // Если ошибка
        {
            DWORD dwError = GetLastError();
            // Смотрим что получилось при открытии
            if(dwError == ERROR_ACCESS_DENIED || dwError ==
            ERROR_GEN_FAILURE)
            {
                Mem1->Lines->Add("Порт не может быть открыт");
                bSuccess = FALSE;
            }
        }
        else // Если все нормально и ошибок нет
        {
            // Порт открыт успешно
            bSuccess = TRUE;          // Состояние порта
            // Не забываем закрывать каждый открытый порт,
            // так как мы не собираемся с ним работать...
            CloseHandle(hPort);
        }
        // Выводим на экран название порта
        if(bSuccess)
        {
            Mem1->Lines->Add("Найденные на ПК COM-порты:");
            Mem1->Lines->Add(sPort);
        }
    }
}
//-----

```

Текст исходных кодов функции № 4 очень похож на текст исходных кодов функции № 3. Так что каких-то дополнительных пояснений, по моему мнению, не требуется. Информация о всех обнаруженных на данном ПК COM-портах будет выведено в многострочном окне редактирования, расположенном ниже кнопки **Проверка COM-портов**.

В листинге 1.5 приведены тексты исходных кодов четырех функций. Выполнение функции № 5 начинается после нажатия кнопки **Установить RTS** и служит для на линии RTS сигнала с определенной величиной положительного постоянного напряжения. Обычно говорят, что на линии RTS установлена 1(единица).

Во второй и третьей строках исходного текста функции №5 программируется изменение цвета и надписи на кнопке **Установить RTS**.

В следующих шести строках программируется восстановление первоначального состояния остальных трех кнопок типа `SpeedButton`, задействованных в группе.

Далее идут строки, написанные на языке Ассемблера, в которых сначала выбирается необходимый адрес для заданного COM-порта, а затем в ячейку по этому адресу записывается соответствующее число. Для установки RTS таким числом является 2.

Листинг 1.5. Продолжение 3 файла *sport_a.cpp*

```
//-----
// Нажатие кнопки <Установить RTS>
void __fastcall TForm1::SpeedButton2Click(TObject *Sender)    // 5
{
    if(SpeedButton2->Down)                                    // Если кнопка нажата,
    {
        SpeedButton2->Font->Color = clBlue;                  // изменяется цвет и
        SpeedButton2->Caption = "RTS установлен";           // изменяется надпись
        SpeedButton5->Font->Color = clWindowText;
        SpeedButton5->Caption = "Сброс RTS и DTR";
        SpeedButton3->Font->Color = clWindowText;
        SpeedButton3->Caption = "Установить DTR";
        SpeedButton4->Font->Color = clWindowText;
        SpeedButton4->Caption = "Установить RTS + DTR";
        if(i_prt == 1)                                        // Если выбран COM1,
        { asm mov dx, 0x3fc;    // далее три строки на языке Ассемблера,
          asm mov al, 2         // по адресу записывается число 2
          asm out dx, al
        }
    }
    else                                                      // Если выбран другой порт, то
    {                                                          // все равно считается, что это
        asm mov dx, 0x2fc                                         // выбран COM2
        asm mov al, 2
        asm out dx, al
    }
    Mem01->Lines->Add("RTS установлен");
```

```

    }
    else          // Если кнопка отжата (в начальном состоянии) ,
    {
        // восстанавливается цвет
        SpeedButton2->Font->Color = clWindowText;
        // и восстанавливается текст
        SpeedButton2->Caption = "Установить RTS";
    }
}
//-----
// Нажатие кнопки <Установить DTR>
void __fastcall TForm1::SpeedButton3Click(TObject *Sender)      // 6
{
    if (SpeedButton3->Down)
    {
        SpeedButton3->Font->Color = clBlue;
        SpeedButton3->Caption = "DTR установлен";
        SpeedButton5->Font->Color = clWindowText;
        SpeedButton5->Caption = "Сброс RTS и DTR";
        SpeedButton4->Font->Color = clWindowText;
        SpeedButton4->Caption = "Установить RTS + DTR";
        SpeedButton2->Font->Color = clWindowText;
        SpeedButton2->Caption = "Установить RTS";
        if(i_prt == 1) asm mov dx, 0x3fc;
        else          asm mov dx, 0x2fc;
        asm {
            mov al, 1          // В ячейку по адресу записывается число 1
            out dx, al
        }
        Mem01->Lines->Add("DTR установлен ");
    }
    else
    {
        SpeedButton3->Font->Color = clWindowText;
        SpeedButton3->Caption = "Установить DTR";
    }
}
//-----
// Нажатие кнопки <Установить RTS + DTR>
void __fastcall TForm1::SpeedButton4Click(TObject *Sender)      // 7
{
    if (SpeedButton4->Down)
    {

```

```

SpeedButton4->Font->Color = clBlue;
SpeedButton4->Caption = "RTS+DTR установлены";
SpeedButton5->Font->Color = clWindowText;
SpeedButton5->Caption = "Сброс RTS и DTR";
SpeedButton3->Font->Color = clWindowText;
SpeedButton3->Caption = "Установить DTR";
SpeedButton2->Font->Color = clWindowText;
SpeedButton2->Caption = "Установить RTS";
if(i_prt == 1) asm mov dx, 0x3fc;
else          asm mov dx, 0x2fc;
asm {
    mov al, 3          // В ячейку по адресу записывается число 3
    out dx, al
}
Memol->Lines->Add("RTS + DTR установлены");
}
else
{
    SpeedButton4->Font->Color = clWindowText;
    SpeedButton4->Caption = "Установить RTS+DTR";
}
}
//-----
// Нажатие кнопки <Сброс RTS и DTR>
void __fastcall TForm1::SpeedButton5Click(TObject *Sender)    // 8
{
    if(SpeedButton5->Down)    // Если кнопка нажата,
    {
        SpeedButton5->Font->Color = clBlue;    // изменяется цвет и
        // изменяется надпись
        SpeedButton5->Caption = "RTS и DTR очищены";
        SpeedButton4->Font->Color = clWindowText;
        SpeedButton4->Caption = "Установить RTS+DTR";
        SpeedButton3->Font->Color = clWindowText;
        SpeedButton3->Caption = "Установить DTR";
        SpeedButton2->Font->Color = clWindowText;
        SpeedButton2->Caption = "Установить RTS";
        if(i_prt == 1) asm mov dx, 0x3fc;
        else          asm mov dx, 0x2fc;
        asm {
            mov al, 0          // В ячейку по адресу записывается число 0
            out dx, al
        }
    }
}

```



```

    Memol->Lines->Add("RTS + DTR очищены");
}
else
{
    SpeedButton5->Font->Color = clWindowText;
    SpeedButton5->Caption = "Сброс RTS и DTR";
}
}
//-----

```

Исходные тексты функций № 6 и № 7 полностью совпадают с текстом функции № 5, за исключением несколько измененной формы написания строк на языке Ассемблера и специфических названий задействованных линий.

Функция № 8 предназначена для очистки линий RTS и DTR (установить на линиях 0 вместо 1) и выполняется после нажатия кнопки **Сброс RTS и DTR**.

В первых двух строках устанавливаются новый цвет и новый текст на этой кнопке.

Начиная со третьей строки текста выполняется восстановление первоначального состояния цвета и надписи сначала кнопки **Установить RTS и DTR**, а затем и всех других кнопок типа *SpeedButton*, задействованных в группе.

В листинге 1.6 приведены исходные тексты функции № 9, которая служит для проверки наличия поступающих на COM-порт сигналов по линиям CTS, DCD и RI.

В первых двух строках текста программируется запись в информационное окно *Memol* сначала пустой строки, а затем строки "Проверка входных линий CTS, DCD и RI".

Далее идут строки с текстом на языке Ассемблера, сначала в них выбирается адрес регистра, в котором будет контролироваться появление сигнала, а затем полученный из регистра сигнал сравнивается с контрольным числом, написанным в двоичной форме. Контрольное число 00010000b соответствует той ячейке регистра, которая указывает на состояние сигнала в линии CTS.

Затем точно таким же способом проверяется наличие сигнала в ячейке, указывающее на состояние сигнала в линии DCD, при этом контрольное число будет 10000000b. Последним проверяется точно так же наличие сигнала в линии RI, при этом контрольное число 01000000b.

Листинг 1.6. Продолжение 4 файла *sport_a.cpp*

```

//-----
// Проверка наличия поступающих сигналов на линиях CTS, DCD и RI.
void __fastcall TForm1::Button3Click(TObject *Sender)
{
    Memol->Lines->Add(" ");
    Memol->Lines->Add("Проверка входных линий CTS, DCD и RI");
    if(i_prt == 1) asm mov dx, 0x3fe;
    else          asm mov dx, 0x2fe;
}

```

```
asm {
    in al, dx
    and al, 00010000b
    jz wyh1
    jnz wyh2
}
wyh1: prt1 = 0;
    asm jmp wyh3
wyh2: prt1 = 1;
wyh3: if(prt1 == 1) {Memol->Lines->Add("CTS (TF) сигнал установлен");
        Label3->Caption = "Сигнал на линии CTS";
    }
    else {Memol->Lines->Add("CTS (TF) сигнала нет");
        Label3->Caption = "";
    } if(i_prt == 1) asm mov dx, 0x3fe;
else
    asm mov dx, 0x2fe;
asm {
    in al, dx
    and al, 10000000b
    jz wyh12
    jnz wyh22
}
wyh12: prt2 = 0;
    asm jmp wyh32
wyh22: prt2 = 1;
wyh32:
    if(prt2 == 1) {Memol->Lines->Add("DCD (AN) сигнал установлен");
        Label4->Caption = "Сигнал на линии DCD";
    }
    else {Memol->Lines->Add("DCD (AN) сигнала на входе нет");
        Label4->Caption = "";
    } if(i_prt == 1) asm mov dx, 0x3fe;
else
    asm mov dx, 0x2fe;
asm {
    in al, dx
    and al, 01000000b
    jz wyh13
    jnz wyh23
}
wyh13: prt3 = 0;
    asm jmp wyh33
wyh23: prt3 = 1;
wyh33:
```

```

if(prt3 == 1){ Mem01->Lines->Add("RI сигнал установлен");
    Label5->Caption = "Сигнал на линии RI";
}

else {
    Mem01->Lines->Add("RI сигнала на входе нет");
    Label5->Caption = "";
}}
//-----

```

Если в какой то из линий имеется сигнал, то соответствующее сообщение записывается в информационное окно и в отдельной строке на главной форме, ниже кнопки, вызвавшей эту проверку.

На этом программирование закончено. Для сохранения всех файлов выбираем команду **File⇒Save All**.

Чтобы выявить возможные ошибки воспользуемся командой **Proekt⇒Make s_port1**. Если ошибок в текстах исходных кодов программы нет, то выбираем **Proekt⇒Build s_port1**. В результате выполнения этой команды создается исполняемый файл `decibell1.exe`, который можно запустить на выполнение с помощью команды **Run⇒Run**.

Чтобы избежать одновременного исправления большого числа ошибок советуем на промежуточных этапах программирования почаще выполнять компиляцию программы — выбрать **Proekt⇒Make <name>**.

Доработка проекта

Дело в том, что созданный командой **Proekt⇒Build s_port1** исполняемый файл будет запускаться и работать только на компьютере, на котором установлена среда программирования C++ Builder.

Для того чтобы созданная вами программа могла запускаться в работу на любом компьютере с операционной системой Microsoft Windows 95/98/ME, необходимо перед компиляцией проекта выполнить следующие действия.

1. Выполните команду **Project⇒Options**.
2. В раскрывшемся окне опций проекта **Project Options** (Опции проекта) выберите страницу **Linker** (рис. 1.4) и сбросьте в ней флажок **Use dynamic RTL** (Использовать динамическое связывание).
3. Перейдите на страницу **Packages** (Пакеты) и сбросьте в ней флажок **Built with runtime packages** (Строить с пакетами времени выполнения).
4. Перейдите на страницу **Compiler** (Компилятор) и нажмите кнопку **Release**, обеспечивающую создание файла без отладочной информации.

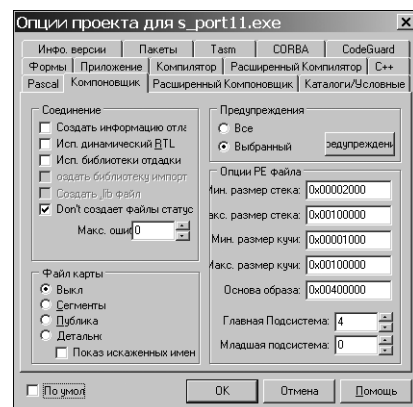


Рис. 1.4. Окно Project Options

После этих действий выполните команду **Project⇒Build**. В результате будет создано выполняемое Windows-приложение без поддержки пакетов, которое будет запускаться, и работать на других компьютерах.

И еще...

Разумеется, что приведенные мною в этой главе исходные коды простой программы с применением нетрадиционного программирования COM-порта, можно несколько упростить. В своих программах вы можете использовать только отдельные функции из этой программы. Желаящим предлагаю сделать программу для управления простым звуковым генератором, схема которого показана на рис. 1.5.

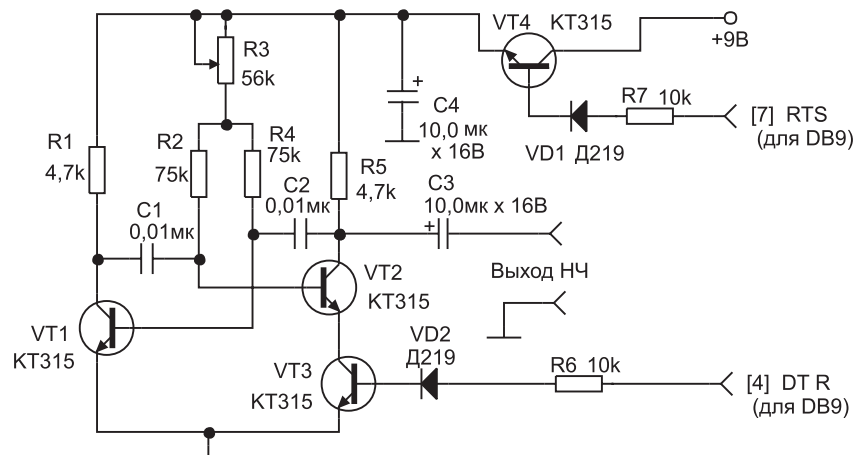


Рис. 1.5. Схема звукового генератора

В заключении необходимо еще раз заметить, что нетрадиционное программирование COM-портов в Windows-приложениях не поддерживается операционными системами Microsoft Windows 2000/XP.

В то же время в этих операционных системах нормально работают программы с нетрадиционным программированием COM-портов, но разработанные в среде программирования Borland Turbo C/C++ под управлением операционной системы MS DOS. В этом вы можете убедиться, если скачаете с сайта r3xb.nm.ru, программы для цифровых видов связи. Там же вы найдете и исходные коды некоторых из этих или подобных программ.

На Internet-сайтах отдельных радиолюбителей, также можно найти программы-драйверы, предназначенные специально для нетрадиционного программирования COM-портов.

В этой главе, а также в некоторых последующих главах этой книги, приведены исходные коды компьютерных программ, разработанных в среде программирования Borland C++ Builder. Если у вас не окажется какой-либо версии C++ Builder, то на сайте этой фирмы www.borland.com в разделе [/products/downloads/](http://www.borland.com/products/downloads/) можно скачать *trial-версию* (пробную версию) этой среды разработки программ. На рис. 1.6 показано окно сайта, предназначенное для скачивания пробной версии C++ Builder.

Если возникнут трудности с поиском этой версии, то в окне поиска введите слово `trial` и нажмите кнопку **Поиск**. В результате вы получите целый перечень различных адресов, откуда можно скачать интересующую вас версию программы. Окно с результатами подобного поиска показано на рис. 1.7.

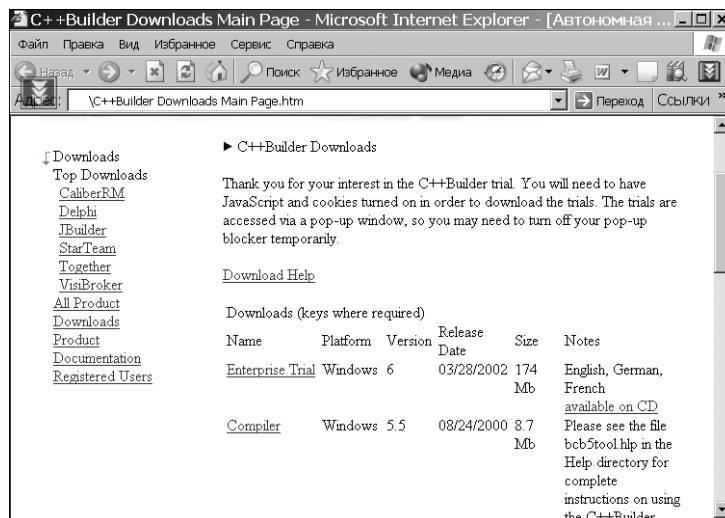


Рис. 1.6. Окно для скачивания C++ Builder

Следует учитывать, что перед тем как будет позволено скачать интересующую вас программу, придется зарегистрироваться, т.е. ответить на целый ряд вопросов. Также нужно сообщить адрес своей электронной почты и придумать пароль. На ваш адрес впоследствии будет выслан ключ, который позволит запустить полученную с сайта программу в работу.

Разумеется, в своих ответах при регистрации вы должны представиться как студент и сообщить о том, что программа необходима для обучения.

Файл с *trial-версией* программы имеет размер порядка 180 Мбайт, поэтому при скачивании придется воспользоваться вспомогательной программой, например, FlashGet, скачать которую можно с сайта amazesoft.com. Рабочее окно этой незарегистрированной программы показано на рис. 1.8. Программа FlashGet автоматически включается в тот момент, когда вы дадите команду для начала скачивания нужных файлов или программ. Если по каким-то причинам скачивание будет прервано, то продолжить эту работу можно будет в последующие дни. Программа запоминает адрес сайта и то место в программе, на котором произошла остановка.

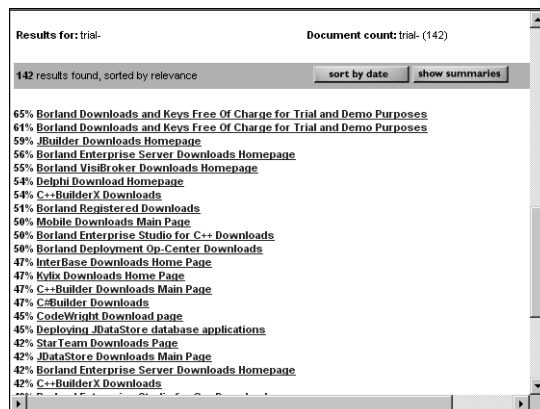


Рис. 1.7. Окно с результатами поиска

Порт LPT в нестандартном режиме

Порт параллельного вывода данных LPT тоже может работать в нестандартном (необычном) режиме. Один из возможных нестандартных режимов работы LPT можно задействовать, манипулируя битами вводимых (выводимых) данных. Подробности смотрите в приложении 3.



Рис. 1.8. Рабочее окно программы FlashGet

Многочисленные опыты показали, что вывод команд со стороны компьютера очень хорошо выполняется посредством сигналов Data1–Data8 (выводы 2–8) порта, а ввод сигналов от внешнего аппаратного устройства на компьютер выполняется через выводы 10, 11, 12 и 15.

Мною разработан в учебных целях проект, исполняемый файл которого `lpt_port2.exe` и исходные коды проекта можете найти на прилагаемом к книге компакт-диске. На рис. 1.9 показана рабочая форма проекта с установленными на ней компонентами, а в листинге 1.7 представлены исходные коды заголовочного файла `sport_a.h`. В этом файле объявлены все задействованные в проекте компоненты и функции.

Листинг 1.7. Файл `sport_a.h`

```
//-----
#ifndef sport_aH
#define sport_aH

#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Buttons.hpp>

class TForm1 : public TForm
{
__published:
    // IDE-managed Components
    TButton *Button1;
```

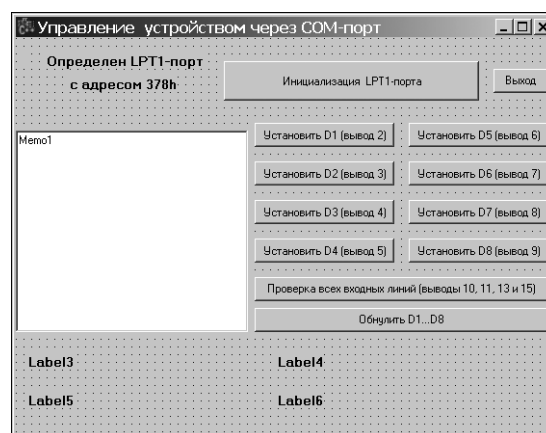


Рис. 1.9. Рабочая форма проекта

```

TLabel *Label1;
TLabel *Label2;
TSpeedButton *SpeedButton1;
TMemo *Memo1;
TSpeedButton *SpeedButton2;
TSpeedButton *SpeedButton3;
TSpeedButton *SpeedButton4;
TLabel *Label3;
TLabel *Label4;
TLabel *Label5;
TSpeedButton *SpeedButton5;
TButton *Button3;
TSpeedButton *SpeedButton6;
TSpeedButton *SpeedButton7;
TSpeedButton *SpeedButton8;
TSpeedButton *SpeedButton9;
TLabel *Label6;
TButton *Button2;
void __fastcall Button1Click(TObject *Sender);
void __fastcall SpeedButton1Click(TObject *Sender);
void __fastcall SpeedButton2Click(TObject *Sender);
void __fastcall SpeedButton3Click(TObject *Sender);
void __fastcall SpeedButton4Click(TObject *Sender);
void __fastcall SpeedButton5Click(TObject *Sender);
void __fastcall Button3Click(TObject *Sender);
void __fastcall SpeedButton6Click(TObject *Sender);
void __fastcall SpeedButton7Click(TObject *Sender);
void __fastcall SpeedButton8Click(TObject *Sender);
void __fastcall SpeedButton9Click(TObject *Sender);
void __fastcall Button2Click(TObject *Sender);
private:      // User declarations
public:       // User declarations
    __fastcall TForm1(TComponent* Owner);
};

extern PACKAGE TForm1 *Form1;

#endif
//-----

```

Файл создается автоматически в процессе работы над проектом и никаким изменениям, без веских на то обстоятельств, подвергаться не должен.

На рис. 1.10 показано рабочее окно программы `lpt_port2.exe` в процессе работы. Эта программа предназначена только для работы с портом LPT1.

Вначале следует провести инициализацию порта, в процессе которой в рабочие биты D1–D8 будут установлены положительные напряжения — единицы. Ниже располагаются кнопки, посредством которых в тот или иной бит устанавливается значение NULL. Получается так, что работа выполняется с инверсными величинами.

Это может потребовать установки на вводную линию каждого байта отдельное инвертирующее устройство в виде микросхемы обычной ТТЛ логики. Следует помнить, что LPT-порт работает с сигналами напряжением не более +5В.

В листинге 1.8 приведены исходные коды файла `sport_a.cpp`. В первой секции листинга 1.8 выполнено подключение заголовочных файлов, в следующей секции — объявление задействованных компонентов и глобальных переменных величин.

Листинг 1.8. Файл `sport_a.cpp`

```
//-----
#include <vcl.h>
#pragma hdrstop
#pragma inline           // Разрешение работать с Ассемблером
#include "sport_a.h"
#include <conio.h>
#include <stdio.h>
//-----

#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
TMemo *Memo1;
TLabel *Label3;
TLabel *Label4;
TLabel *Label5;
TLabel *Label6;
//Глобальные переменные
int prt1, prt2, prt3, prt4, prt5, prt6, prt7, prt8;
//-----
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
    Caption = "Управление устройством через LPT-порт"; // Заголовок
    Memo1->Clear(); // Очистка многострочное окно
    Label3->Caption = ""; // Очистка метки Label
```

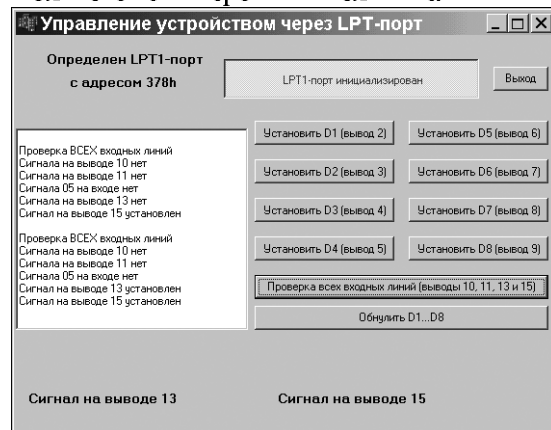


Рис. 1.10. Рабочее окно экспериментальной программы


```

Label4->Caption = "";
Label5->Caption = "";
Label6->Caption = "";
SpeedButton1->AllowAllUp = true; // Показывает нажатую кнопку
SpeedButton1->GroupIndex = 1;    // Принадлежность к группе кнопок
SpeedButton2->AllowAllUp = true; // То же
SpeedButton2->GroupIndex = 2;    // То же
SpeedButton3->AllowAllUp = true;
SpeedButton3->GroupIndex = 2;
SpeedButton4->AllowAllUp = true;
SpeedButton4->GroupIndex = 2;
SpeedButton5->AllowAllUp = true;
SpeedButton5->GroupIndex = 2;
SpeedButton6->AllowAllUp = true;
SpeedButton6->GroupIndex = 2;
SpeedButton7->AllowAllUp = true;
SpeedButton7->GroupIndex = 2;
SpeedButton8->AllowAllUp = true;
SpeedButton8->GroupIndex = 2;
SpeedButton9->AllowAllUp = true;
SpeedButton9->GroupIndex = 2;
}
//-----
// Реакция на нажатие кнопки <Выход>, прекращается работа программы
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    Close();
}
//-----
// Инициализация порта
void __fastcall TForm1::SpeedButton1Click(TObject *Sender)
{
    if(SpeedButton1->Down)
    {asm mov dx, 0x379
      asm mov al, 10011000b
      asm out dx, al
      asm mov dx, 0x37A
      asm mov al, 00001100b
      asm out dx, al
      asm mov dx, 0x378
      asm mov al, 255      // 1111'1111b - во всех битах - 1
      asm out dx, al
    }
}

```

```
        sleep(100);
        SpeedButton1->Font->Color = clBlue;
        SpeedButton1->Caption = "LPT1-порт инициализирован";
        Mem01->Lines->Add("Порт LPT1 инициализирован");
    }
    //-----
    // Обнуляем БИТ0
    void __fastcall TForm1::SpeedButton2Click(TObject *Sender)
    {
        if(SpeedButton2->Down)
        {
            asm mov dx, 0x378
            asm mov al, 254 // 1111'1110b
            asm out dx, al
            Mem01->Lines->Add("Вывод 2 (D1) установлен"); // Это сигнал
        }
        else
        {
            asm mov dx, 0x378
            asm mov al, 255 // Возвращаем 1 в БИТ0
            asm out dx, al
            Mem01->Lines->Add("(Вывод 2 (D1) очищен"); // То есть возвращена 1
            Mem01->Lines->Add("");
        }
    }
    //-----
    // Обнуляем БИТ1
    void __fastcall TForm1::SpeedButton3Click(TObject *Sender)
    {
        if(SpeedButton3->Down)
        {
            asm mov dx, 0x378
            asm mov al, 253 //1111'1101b
            asm out dx, al
            Mem01->Lines->Add("Вывод 3 (D2) установлен ");
        }
        else
        {
            asm mov dx, 0x378
            asm mov al, 255 // 1111'1111b
            asm out dx, al
            Mem01->Lines->Add("Вывод 3 (D2) очищен");
            Mem01->Lines->Add("");
        }
    }
}
```

```

    }
}
//-----
// Обнуляем БИТ2
void __fastcall TForm1::SpeedButton4Click(TObject *Sender)
{
    if(SpeedButton4->Down)
    {
        asm mov dx, 0x378
        asm mov al, 251 //1111'1011b
        asm out dx, al
        Mem01->Lines->Add("Вывод 4 (D3) установлен ");
    }
    else
    {
        asm mov dx, 0x378 //+2
        asm mov al, 255 // FF или 1111'1111
        asm out dx, al
        Mem01->Lines->Add("Вывод 4 (D3) очищен");
        Mem01->Lines->Add("");
    }
}
//-----
void __fastcall TForm1::SpeedButton5Click(TObject *Sender)
{
    if(SpeedButton5->Down)
    {
        asm mov dx, 0x378
        asm mov al, 223 //1101'1111 -
        asm out dx, al
        Mem01->Lines->Add("Вывод 6 (D5) установлен");
    }
    else
    {
        asm mov dx, 0x378
        asm mov al, 255
        asm out dx, al
        Mem01->Lines->Add("Вывод 6 (D5) очищен");
        Mem01->Lines->Add("");
    }
}
//-----

```

В листинге 1.9 продолжается описание функций, которые являются реакциями на нажатие кнопок, устанавливающих сигналы NULL в битах 3–7. Практически все они имеют одинаковую структуру.

Листинг 1.9. Продолжение 1 файла *sport_a.cpp*

```
//-----
void __fastcall TForm1::SpeedButton6Click(TObject *Sender)
{
    if(SpeedButton6->Down)
    {
        asm mov dx, 0x378
        asm mov al, 247 //1111'0111
        asm out dx, al
        Memol->Lines->Add("Вывод 4 (D3) установлен ");
    }
    else
    {
        asm mov dx, 0x378
        asm mov al, 255
        asm out dx, al
        Memol->Lines->Add("Вывод 4 (D3) очищен");
        Memol->Lines->Add("");
    }
}
//-----
void __fastcall TForm1::SpeedButton7Click(TObject *Sender)
{
    if(SpeedButton7->Down)
    {
        asm mov dx, 0x378
        asm mov al, 239 //1110'1111 - DF
        asm out dx, al
        Memol->Lines->Add("Вывод 7 (D6) установлен ");
    }
    else
    {
        asm mov dx, 0x378
        asm mov al, 255
        asm out dx, al
        Memol->Lines->Add("Вывод 7 (D6) очищен");
        Memol->Lines->Add("");
    }
}
```

```

//-----
void __fastcall TForm1::SpeedButton8Click(TObject *Sender)
{
    if (SpeedButton8->Down)
    {
        asm mov dx, 0x378
        asm mov al, 191 //1011'1111 -
        asm out dx, al
        Mem1->Lines->Add("Вывод 8 (D7) установлен");
    }
    else
    {
        asm mov dx, 0x378
        asm mov al, 255
        asm out dx, al
        Mem1->Lines->Add("Вывод 8 (D7) очищен");
        Mem1->Lines->Add("");
    }
}
//-----
void __fastcall TForm1::SpeedButton9Click(TObject *Sender)
{
    if (SpeedButton9->Down)
    {
        asm mov dx, 0x378
        asm mov al, 127 //0111'1111 -
        asm out dx, al
        Mem1->Lines->Add("Вывод 9 (D8) установлен");
    }
    else
    {
        asm mov dx, 0x378
        asm mov al, 255
        asm out dx, al
        Mem1->Lines->Add("Вывод 9 (D8) очищен");
        Mem1->Lines->Add("");
    }
}
//-----

```

Следует помнить, что эта программа написана только для учебных целей, чтобы показать один из возможных вариантов нетрадиционного программирования LPT-порта.

По образцу этой программы вы можете сделать свою программу, предназначенную для управления каким-то аппаратом. В материалах компакт-диска, относящихся к главе 3, например, приведены схемы адаптеров, которые используют LPT-порт. Советую просмотреть этот материал тем читателям, которые заинтересовались возможностями нестандартного программирования портов ввода / вывода.

В листинге 1.10 приведены исходные коды функции, предназначенной для проверки всех битов регистра 379h на наличие в них сигналов типа NULL. Результаты проверки выводятся в окне многострочного редактирования Memo1, а в случае появления сигнала — соответствующая запись появляется в нижней части рабочего окна программы.

Листинг 1.10. Продолжение 2 файла *sport_a.cpp*

```
//-----
// Выполняется поиск битов со значением 0
void __fastcall TForm1::Button3Click(TObject *Sender)
{
    Memo1->Lines->Add(" ");
    Memo1->Lines->Add("Проверка ВСЕХ входных выводов LPT-порта");
    // проверка вывода 15 порта
    asm {mov dx, 0x379
        in al, dx
        and al, 00001000b
        jz wyh14
        jnz wyh24
    }
    wyh14: prt4 = 0;
    asm jmp wyh34
    wyh24: prt4 = 1;
    wyh34:
        if(prt4 == 0){ Memo1->Lines->Add("Сигнал на выводе 15 установлен");
                                Label3->Caption = "Сигнал на выводе 15";
                                }
        else {
            Memo1->Lines->Add("Сигнала на выводе 15 нет");
            Label3->Caption = "";
        }
    // Проверка вывода 13 порта
    asm {mov dx, 0x379
        in al, dx
        and al, 00010000b
        jz wyh15
        jnz wyh25
    }
```

```

wyh15: prt5 = 0;
      asm jmp wyh35
wyh25: prt5 = 1;
wyh35:
      if(prt5 == 0){ Mem01->Lines->Add("Сигнал на выводе 13 установлен");
                                Label4->Caption = "Сигнал на выводе 13";
                                }
      else {
                                Mem01->Lines->Add("Сигнала на выводе 13 нет");
                                Label4->Caption = "";
                                }
// Проверка вывода порта
/*
asm {mov dx, 0x379      //+1
      in al, dx
      and al, 00100000b
      jz wyh16
      jnz wyh26
      }
wyh16: prt6 = 0;
      asm jmp wyh36
wyh26: prt6 = 1;
wyh36:
      if(prt6 == 0){ Mem01->Lines->Add("Сигнал 05 установлен");
                                Label6->Caption = "Сигнал на линии 05";
                                }
      else {
                                Mem01->Lines->Add("Сигнала 05 на входе нет");
                                Label6->Caption = "";
                                }
*/
// Проверка вывода 10 порта
asm {mov dx, 0x379      //+1
      in al, dx
      and al, 01000000b
      jz wyh17
      jnz wyh27
      }
wyh17: prt7 = 0;
      asm jmp wyh37
wyh27: prt7 = 1;
wyh37:

```

```
    if(prt7 == 0){ Mem01->Lines->Add("Сигнал на выводе 10 установлен");
        Label5->Caption = "Сигнал на выводе 10";
    }
    else {
        Mem01->Lines->Add("Сигнала на выводе 10 нет");
        Label5->Caption = "";
    }
// Проверка вывода 11 порта
asm {mov dx, 0x379    //+1
    in al, dx
    and al, 10000000b
    jz wyh18
    jnz wyh28
}
wyh18: prt8 = 0;
asm jmp wyh38
wyh28: prt8 = 1;
wyh38:
    if(prt8 == 1){ Mem01->Lines->Add("Сигнал на выводе 11 установлен");
        Label6->Caption = "Сигнал на выводе 11";
    }
    else {
        Mem01->Lines->Add("Сигнала на выводе 11 нет");
        Label6->Caption = "";
    }
}
//-----
// Обнулить D1-D8
void __fastcall TForm1::Button2Click(TObject *Sender)
{
    asm mov dx, 0x378
    asm mov al, 0        //255
    asm out dx, al
    Mem01->Lines->Add("");
    Mem01->Lines->Add("Все выводы (10, 11, 13 и 15) очищены");
}
//-----
```

Проверка всех битов на наличие в них сигнала закончена. Вместо установки в битах сигналов программным путем можно применить электрический способ, т.е. вывод заданного бита подсоединять на землю (экран) через резистор 10–75 Ом. В

реальных конструкциях к выводу заданного бита следует подключать инвертор с ТТЛ уровнями напряжений. При этом входные сигналы следует подавать через этот инвертор.

Проверяем работоспособность LPT-порта

Для проверки работоспособности LPT-порта при нестандартном его программировании, предлагается простейшее устройство, схема которого показана на рис. 1.11.

Номера выводов

LPT-порта

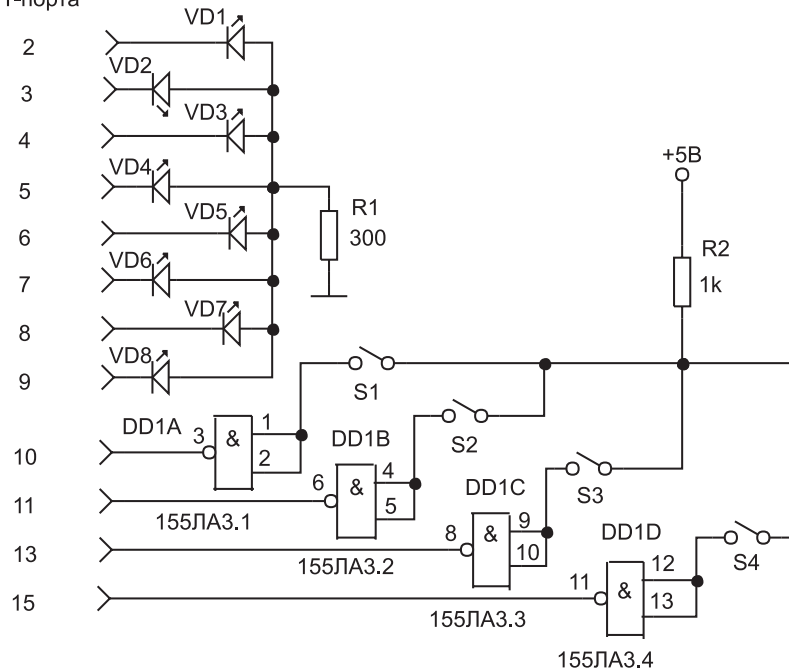


Рис. 1.11. Схема устройства для проверки LPR-порта

Устройство подключается к выводам 2–9, 10, 11, 13 и 15 LPT-порта. При этом с выводов 2–9 поступают сигналы непосредственно на светодиоды VD1–VD8, а на выводы 10, 11, 13 и 15 посредством ключей S1–S4 подаются сигналы с внешнего устройства на компьютер через инверторы DD1A–DD1D.

Предлагаемое устройство создано только для учебных целей и практического применения не имеет.