

Глава 5

Компьютер как источник звука

В компьютерных программах большую роль играет звук. Не говоря уже об игровых и специальных музыкальных программах.

Например, в некоторых программах имеются голосовые подсказки для выполнения последующих действий, голосовое предупреждение о возможной ошибке и т.д. Существуют компьютерные программы, которые человеческим голосом на различных языках могут читать текст, представленный в электронном виде, а также программы, которые заставляют компьютер разговорную речь записывать в виде текста в электронном виде.

Приведенные выше примеры применения звука в компьютерных программах знакомы многим пользователям и ничего необычного не представляют.

В то же время часто появляется необходимость самому создать какое-то необычное применение звука, полученного от компьютера. Особенно часто подобная необходимость возникает у радиолюбителей.

Во многих радиолюбительских программах звуковая карта компьютера может заменять аппаратный модем (служить «эрзац-модемом»), либо изначально ей предназначается быть звуковым модемом (программы PSK31).

При создании подобных любительских программ часто возникает необходимость в нестандартном программировании звука. О некоторых таких нестандартных способах программирования звука рассказывается в этой главе.

Программирование компьютерного динамика

Практически каждый IBM PC компьютер имеет в своем составе небольшой громкоговоритель (динамик). Как правило, он находится на передней панели системного блока и предназначен для выдачи не очень качественных сигналов различной звуковой частоты при каких-то особых случаях в работе компьютера.

Обычные пользователи, как правило, не имеют потребности в работе этого устройства, но для «необычных» пользователей временами возникает необходимость заставить его издавать нужной частоты звуки в нужное время. Поэтому приводимый мною ниже материал о программировании компьютерного динамика может для них представлять определенный интерес.

Рассмотрим такой случай программирования динамика, когда используется звуковой канал находящейся в компьютере микросхемы-таймера.

Программирование звукового канала таймера

В программировании звука, издаваемого компьютерным динамиком, принимают участие две микросхемы компьютера — таймер и программируемый периферий-

ный интерфейс, управляющий клавиатурой и динамиком. На рис. 5.1 приведены упрощенные схематические изображения этих микросхем.

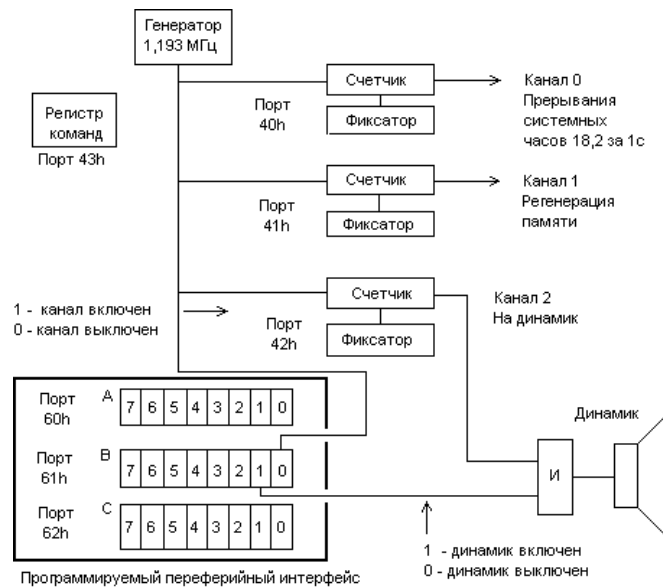


Рис. 5.1. Упрощенная схема генерации звука

Микросхема таймера работает независимо от процессора (и параллельно с ним) от собственного генератора, вырабатывающего сигналы с частотой 1,19318 МГц. Таймер имеет три независимых канала, каждый из которых можно независимо перепрограммировать.

Для управления режимами программирования в микросхеме имеется регистр команд, обращение к которому осуществляется через порт 43h. Каждый канал таймера включает счетчик, пересчитывающий сигналы от генератора, и регистр-фиксатор, в который программно заносится число, определяющее коэффициент пересчета счетчика. Фиксаторы каналов таймера адресуются через порты 40h, 41h и 42h.

При включении компьютера в фиксатор канала 0 заносится максимально возможное число 65535 (FFFFh), в результате чего сигналы на выходе канала 0 имеют частоту 18,2 за 1с. Эти сигналы возбуждают прерывания с вектором 08h, которые обрабатываются программой BIOS, осуществляющей отсчет текущего времени. Прерывания от канала 0 можно использовать для временной синхронизации программы, например, для периодического вывода на экран некоторой информации.

Канал 1 таймера используется схемами регенерации памяти и его в своих программах использовать не следует.

Выход канала 2 связан с динамиком и используется для генерации звука. Изменяя содержимое фиксатора этого канала, можно изменять частоту сигналов, поступающих на динамик, от 18,2 Гц до 1,19 МГц. Реально для возбуждения звука можно использовать частоты не выше 10 кГц.

Управление каналом 2 таймера и его подключение к динамику осуществляется с помощью регистров программируемого периферийного интерфейса. Эта многофункциональная микросхема содержит три регистра, адресуемые через порты A, B

и С с адресами 60h, 61h и 62h. Порт А используется для приема кодов полученных в результате сканирования клавиш клавиатуры, порты В и С — для управления клавиатурой и таймером и для хранения информации о конфигурации компьютера. Наличие микросхемы программируемого периферийного интерфейса характерно для компьютеров типа IBM PC и PC/XT. На компьютерах PC/AT и PS/2 такой микросхемы нет, однако и прием данных с клавиатуры, и управление таймером и динамиком осуществляется через порты с теми же адресами.

Бит 0 порта 61h управляет включением и выключением канала 2 таймера.

Пока бит 0 установлен, на выходе канала действуют периодические сигналы заданной фиксатором частоты, при сбросе бита 0 колебания прекращаются.

Бит 1 того же порта управляет посылкой в динамик тока. Таким образом, при использовании для возбуждения звука таймера, имеются две возможности прекратить звучание тона:

- ♦ запретить работу канала 2 путем сброса бита 0 порта 61h;
- ♦ выключить прохождение через динамик тока путем сброса бита 1 того же порта.

Программирование любого канала таймера осуществляется одинаково. Прежде всего, в регистр команд (порт 43h) засылается управляющее слово, характеризующее режим работы таймера. Формат управляющего слова приведен на рис. 5.2.

Бит 0 определяет способ задания константы пересчета. При нулевом значении бита константа задается в двоичной форме, при единичном — в двоично-десятичной (BCD). Чаще используется двоичный способ задания константы.

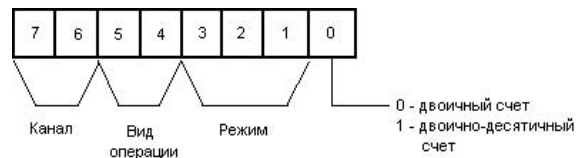


Рис. 5.2. Формат управляющего слова таймера

В биты 1–3 засылается режим работы таймера (разовый или периодический, с различной скважностью и проч.). Для возбуждения звука используется режим 011 — периодическая генерация прямоугольных сигналов со скважностью 2.

В битах 4–5 записывается способ загрузки в фиксатор константы. Обычно используемый код 11 позволяет загрузить в фиксатор 16-разрядную константу пересчета двумя последовательными командами Ассемблера `mov`. Сначала загружается младший байт константы, затем старший.

Наконец, в биты 6–7 засылается номер программируемого канала таймера. Как видно на рис. 5.1, для программирования звука используется канал 2 (код 10). Таким образом, управляющее слово оказывается равным **10110110b**, или **B6h**.

Программа Beep_s1.exe как пример программирования

Для учебных целей освоения программирования компьютерного динамика, мною разработана программа `Beep_s1`, которая использует принципы нестандартного программирования портов, и потому работает только под управлением операционных систем Microsoft Windows 95/98/ME.

На рис. 5.3 показана форма проекта программы, заполненная компонентами.

В листинге 5.1 приведен текст подключаемого заголовочного файла `Beep_s.h`.

Листинг 5.1. Файл `Beep_s.h`

```
//-----
#ifndef Beep_sH
#define Beep_sH
//-----
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
//-----
class TForm1 : public TForm
{
__published: // IDE-managed Components
    TButton *Button1;
    TButton *Button2;
    TButton *Button3;
    TButton *Button4;
    TButton *Button5;
    void __fastcall Button1Click(TObject *Sender);
    void __fastcall Button2Click(TObject *Sender);
    void __fastcall Button3Click(TObject *Sender);
    void __fastcall Button4Click(TObject *Sender);
    void __fastcall Button5Click(TObject *Sender);
private: // User declarations
public: // User declarations
    __fastcall TForm1(TComponent* Owner);
};
//-----
extern PACKAGE TForm1 *Form1;
//-----
#endif
```

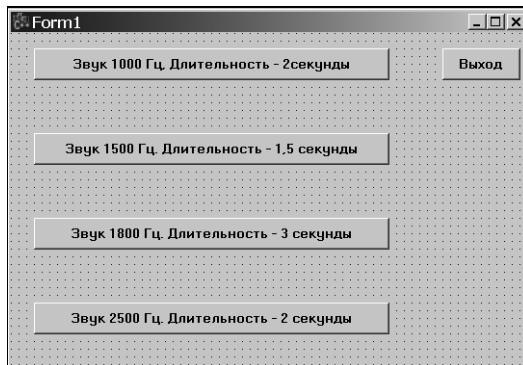


Рис. 5.3. Форма проекта программы `Beep_sl`

В приведенном тексте файла `Beep_s.h` нет ничего особенного и приведен он только для того, чтобы вы могли разобраться с названиями компонентов, размещенных на форме проекта. В листинге 5.2 приведен текст файла `Beep_s.cpp`.

Листинг 5.2. Файл `Beep_s.cpp`.

```
//-----
#include <vcl.h>
#pragma hdrstop
```

```

#pragma inline          // разрешается использовать команды Ассемблера
#include "Beep_s.h"
//-----
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
void Beep1(int, unsigned int); // объявляется функция как глобальная
//-----
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
    Caption = "Звук через динамик (только Windows 95/98/ME)";
}
//-----
// Реакция на нажатие кнопки <Выход>
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    Close();
}
//-----
// Основная подпрограмма для программирования звука
void Beep1(int ton, unsigned int dlitel)
{
    // Установим режим таймера 43
asm {
    mov al, 0x0B6    // вводим управляющее слово B6h
    out 0x43, al    //      в порт 43h
    }
    // Установим частоту канала 2 таймера
if(ton == 1000)      // Назначим частоту 1000 Гц
{
    asm mov ax, 1193    // и введем соответствующий делитель
    goto wh1;
}
if(ton == 1500)      // Если устанавливается частота 1500 Гц, то
{
    asm mov ax, 795    // вводим соответствующий делитель
    goto wh1;
}
if(ton == 1800)      // точно также, как и в предыдущих
{
    asm mov ax, 663    // случаях
    goto wh1;
}
}

```

```
    }
    if(ton == 2500)
    {
        asm mov ax, 477
        goto wh1;
    }
wh1:
asm {
    out 0x42, al
    mov al, ah
    out 0x42, al
}
// Включим динамик и таймер
asm {
    in al, 0x61          // Получаем данные из порта 61h
    or al, 00000001b     // Включаем таймер
    or al, 00000010b     // Включаем динамик
    out 0x61, al         // Записываем новые данные в порт 61h
}
Sleep(dlitel);          // Устанавливаем длительность звучания
// выключим таймер
asm {
    and al, 11111101b    // Выключаем динамик
    out 0x61, al
    and al, 11111110b    // Выключаем таймер
    out 0x61, al
}
return;
}
//-----
```

В конце листинга 5.2 — исходные коды основной подпрограммы для создания звука `void Beep1(int ton,unsigned int dlitel)`.

В листинге 5.3 приведены функции, являющиеся реакциями на нажатие соответствующих кнопок.

Листинг 5.3. Продолжение 1 файла *Beep_s.cpp*.

```
//-----
void __fastcall TForm1::Button2Click(TObject *Sender)
{
    Beep1(1000,2000);
}
//-----
```

```

void __fastcall TForm1::Button3Click(TObject *Sender)
{
    Beep1(1500,1500);
}
//-----
void __fastcall TForm1::Button4Click(TObject *Sender)
{
    Beep1(1800, 3000);
}
//-----
void __fastcall TForm1::Button5Click(TObject *Sender)
{
    Beep1(2500, 2000);
}
//-----

```

Программа `Beep_s1` задает тон нужной частоты и продолжительности. В первых предложениях 1-2 исходного кода (см. листинг 5.2) в регистр команд таймера засылается управляющее слово, описанное выше. После получения управляющего слова микросхема таймера ждет получения двух байтов данных, которые она будет рассматривать, как младший и старший байты константы пересчета. Поэтому после послышки управляющего слова необходимо выполнить две команды `out` в порт `61h`. Константа пересчета, определяющая частоту тона, загружается нами в регистр `AX`, откуда ее младший байт засылается в порт. Затем командой Ассемблера `mov` старший байт константы отправляется в младший байт регистра `AX`, после чего вторая команда `out` загружает старшую половину фиксатора (порт `42h`). Все это время звука нет.

В последующих предложениях текста исходных кодов к содержимому порта `61h` добавляются команды, заданные двумя числами в двоичном исчислении (для большей наглядности). Тем самым включается таймер и включается ток динамика. После окончания заданного функцией `Sleep(dlitel)` интервала времени выполняется выключение таймера и динамика. На рис. 5.4 показано рабочее окно программы `Beep_s1`.

Далее рассмотрим стандартные функции воспроизведения звуков, заложенные непосредственно в операционной системе Microsoft Windows и в средах программирования.

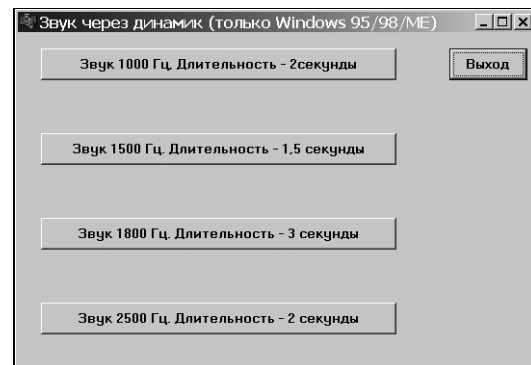


Рис. 5.4. Рабочее окно программы `Beep_s1`

Функции воспроизведения звуков

Среда быстрого программирования Borland C++Builder может работать со всеми возможными вариантами управления звуковыми картами компьютера, заложенными в операционных системах Microsoft Windows. В табл. 5.1 приведены задействованные в этой среде программирования функции воспроизведения звука.

Таблица 5.1. Функции воспроизведения звука

Функция	Синтаксис / описание	Файл
Beep	Extern PACKAGE void Beep(void) Функция C++Builder, воспроизводит стандартный звуковой сигнал	SysUtils.hpp
Beep	BOOL Beep(DWORD dwFreq, DWORD dwDuration); Функция API Windows, только для операционной системы Microsoft Windows NT, воспроизводит звуковой сигнал с частотой dwFreq (в Гц) и длительностью dwDuration (в миллисекундах)	
MessageBeep	BOOL MessageBeep(UINT uType); Функция API Windows, воспроизводит звуковой сигнал типа uType	
PlaySound	BOOL PlaySound(LPCSTR pszSound, HMODULE hmod, DWORD fdwSound); Функция API Windows, воспроизводит звук указанного волнового файла, или звук системного события, или звука из ресурса	mmsystem.hpp

Введем некоторые пояснения. Функция C++ Builder Beep воспроизводит стандартный звуковой сигнал, вызывая функцию MessageBeep API Windows с нулевым параметром. При этом воспроизводится стандартный звуковой сигнал, установленный в операционной системе Microsoft Windows. Если звуковой карты нет или стандартный сигнал не установлен, звук воспроизводится через динамик компьютера.

Воспроизведение асинхронное, т.е. приложение продолжает выполняться во время воспроизведения звука.

Функция Beep API Windows, примененная в операционной системе Microsoft Windows NT, синхронно воспроизводит звук простого тона через динамик и не возвращается до окончания звука. В Windows NT параметр dwFreq задает частоту звука в герцах. Он может иметь значения в диапазоне от 37 до 32 767 (от 0x25 до 0x7FFF). Параметр dwDuration устанавливает длительность звука в миллисекундах.

Воспроизведение синхронное, т.е. функция не возвращается до окончания воспроизведения звука.

Все сказанное относится только к операционной системе Microsoft Windows NT. В операционных системах Microsoft Windows 95/98 параметры игнорируются и функция становится подобной функции Beep C++ Builder. Отличие этих функций остается только в том, что Beep C++ Builder ничего не возвращает, а Beep API

Windows при успешном выполнении возвращает ненулевое значение. При аварийном завершении она возвращает нуль. Тогда более развернутую информацию об ошибке можно получить вызовом функции `GetLastError`.

В качестве примера использования функции `Beep`, предлагаю создать проект простого приложения с использованием этой функции для операционной системе Microsoft Windows XP.

Создаем проект под названием `bepsnd1`, при этом файл с исходными кодами будет иметь название `bepsnd`. Форма приложения показана на рис. 5.5.

На форме установлены только хорошо вам известные компоненты `Label` и `Button`, поэтому никаких описаний не требуется. В листинге 5.4 приводится текст файла `bepsnd.cpp`.

Листинг 5.4. Файл `bepsnd.cpp`

```
//-----
#include <vcl.h>
#pragma hdrstop
#include "bepsnd.h"
//-----
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//-----

__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
    Caption = "Проверка звучания";
    Label1->Caption = "Частота 400 Гц, звучит 2 секунды";
    Label2->Caption = "Частота 800 Гц, звучит 3 секунды";
    Label3->Caption = "Частота 1000 Гц, звучит 4 секунды";
    Label4->Caption = "Частота 1400 Гц, звучит 3 секунды";
    Label5->Caption = "Частота 1600 Гц, звучит 2 секунды";
    Label6->Caption = "Частота 2000 Гц, звучит 2 секунды";
}
//-----

void __fastcall TForm1::Button1Click(TObject *Sender)
{
    Beep(400, 2000);
}
//-----

void __fastcall TForm1::Button2Click(TObject *Sender)
{

```

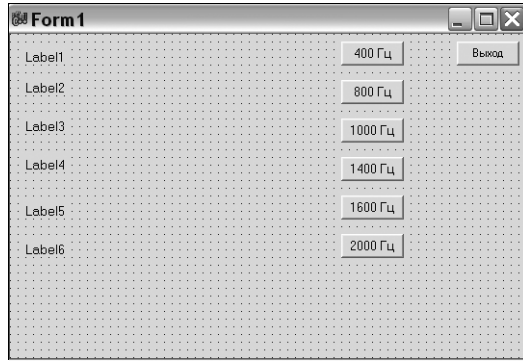


Рис. 5.5. Форма проекта `bepsnd1`

```

    Beep(800,3000);
}
//-----
void __fastcall TForm1::Button3Click(TObject *Sender)
{
    Beep(1000,4000);
}
//-----
void __fastcall TForm1::Button4Click(TObject *Sender)
{
    Beep(1400,3000);
}
//-----
void __fastcall TForm1::Button5Click(TObject *Sender)
{
    Beep(1600,2000);
}
//-----
void __fastcall TForm1::Button6Click(TObject *Sender)
{
    Beep(2000,2000);
}
//-----
void __fastcall TForm1::Button7Click(TObject *Sender)
{
    Close();
}
//-----

```

Я думаю, что нет необходимости комментировать этот простейший проект. На рис. 5.6 показано главное окно работающего приложения.

Компилятор автоматически разбирается, какая именно из функций `Beep` использована в программе, по наличию или отсутствию параметров.

Функция `MessageBeep` воспроизводит звуковой сигнал указанного типа. Звуки, соответствующие различным типам сигналов, хранятся в реестре в разделе `sounds` и устанавливаются пользователем с помощью программы **Панель управления**. Для этого в **Панель**

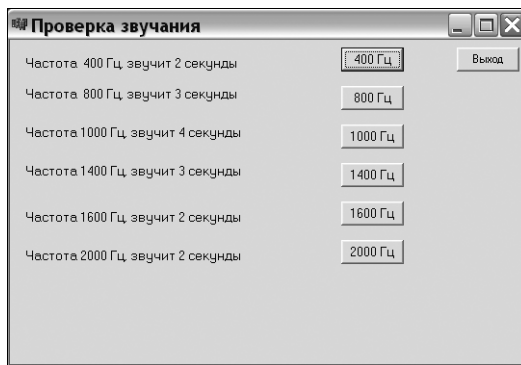


Рис. 5.6. Окно приложения `beepsnd`

управления необходимо дважды щелкнуть на пиктограмме **Звук**.

Целый без знака параметр `uType` функции `MessageBeep` определяет воспроизводимый звук. Для него предопределены следующие константы, приведенные в табл. 5.2.

Таблица 5.2. Параметры аргумента `uType`

Значение	Звук
<code>0xFFFFFFFF</code>	Стандартный звук через динамик
<code>MB_ICONASTERISK</code>	Звездочка
<code>MB_ICONEXCLAMATION</code>	Восклицание
<code>MB_ICONHAND</code>	Критическая ошибка
<code>MB_ICONQUESTION</code>	Вопрос
<code>MB_OK</code>	Стандартный звук

При успешном завершении функция возвращает ненулевое значение (`true`). Если функция вернула нулевое значение, то получить информацию об ошибке можно с помощью вызова `GetLastError`.

После инициализации воспроизведения звука функция `MessageBeep` возвращает управление в точку вызова и воспроизведение звука производится асинхронно.

Если функция `MessageBeep` не нашла указанный тип звука, она пытается воспроизвести стандартный звук. Если и он не установлен или если компьютер не снабжен звуковой картой, то звук воспроизводится через динамик компьютера.

Функция `PlaySound` API Windows воспроизводит звук указанного волнового файла или звук системного события, или звук из ресурса.

Параметр `pszSound` представляет собой строку с нулевым символом в конце и определяет воспроизводимый звук. В зависимости от значений флага `fdwSound` (`SND_FILENAME`, `SND_ALIAS` или `SND_RESOURCE`) параметр `pszSound` может определять имя волнового файла, псевдоним системного события или идентификатор ресурса. Если ни один из этих файлов не указан, функция ищет в реестре операционной системы Microsoft Windows или файле `WIN.INI` указанное имя звука. Если звук найден, то он воспроизводится. Если звук не найден, то параметр `pszSound` интерпретируется как имя файла.

Звук, указанный параметром `pszSound`, должен помещаться в доступную память и должен подходить для установленного драйвера устройства воспроизведения волновых файлов. Функция `PlaySound` ищет файл звука в следующих каталогах: текущем, каталоге Windows, системном каталоге Windows, каталогах, перечисленных в переменной среде `PATH`, в списке каталогов, предоставляемых сетью. Более подробно последовательность поиска в каталогах рассмотрена в документации по функции `OpenFile`.

Если указанный звук не находится, функция `PlaySound` воспроизводит системный звук по умолчанию. Если функция не может найти и его, то воспроизведения не будет, а значение `false` вернется.

Если параметр `pszSound` задан равным 0, то воспроизведение любого волнового файла прерывается. Для прерывания воспроизведения звука, не связанного с волновым файлом, надо указывать `SND_PURGE` в параметре `fdwSound`.

Параметр `hmod` используется только при параметре `fdwSound` равном `SND_RESOURCE`. В этом случае `hmod` является дескриптором выполняемого файла, содержащего ресурс, который должен загружаться. В противном случае значение `hmod` задается равным 0.

Параметр `fdwSound` задает флаги воспроизведения звука. Флаги могут комбинироваться друг с другом комбинацией ИЛИ «|». Возможны такие значения флагов, которые показаны в табл. 5.3.

Таблица 5.3. Флаги параметра `fdwSound`

Флаг	Назначение
<code>SND_ALIAS</code>	Параметр <code>pszSound</code> определяет псевдоним системного события в реестре операционной системы Microsoft Windows или в файле <code>WIN.INI</code> . Нельзя использовать совместно с <code>SND_FILENAME</code> и <code>SND_RESOURCE</code>
<code>SND_ALIAS_ID</code>	Параметр <code>szSound</code> является предопределенным идентификатором звука
<code>SND_APPLICATION</code>	Звук воспроизводится с использованием установок приложения
<code>SND_ASYNC</code>	Звук производится асинхронно и функция <code>PlaySound</code> возвращается немедленно после начала воспроизведения. Чтобы прекратить асинхронное воспроизведение волнового файла, надо вызвать <code>PlaySound</code> с параметром <code>pszSound</code> , равным 0
<code>SND_FILENAME</code>	Параметр <code>pszSound</code> является именем файла
<code>SND_LOOP</code>	Воспроизведение звука постоянно повторяется, пока не вызовется <code>PlaySound</code> с параметром <code>pszSound</code> , равным 0. Одновременно надо указать флаг <code>SND_ASYNC</code> асинхронного воспроизведения звука
<code>SND_MEMORY</code>	Файл звука события загружен в память. В этом случае параметр <code>pszSound</code> должен указывать на образ звука в памяти
<code>SND_NODEFAULT</code>	Звук события, кроме звука по умолчанию. Если указанный звук не найден, <code>PlaySound</code> вернется, не воспроизведя звук по умолчанию
<code>SND_NOSTOP</code>	Если заданный звук не может быть воспроизведен, поскольку ресурсы, необходимые для воспроизведения, заняты воспроизведением другого звука, функция <code>PlaySound</code> немедленно вернет <code>false</code> , не воспроизведя заданного звука. Если данный флаг не указан, функция <code>PlaySound</code> пытается остановить воспроизведение другого звука, чтобы устройство могло быть использовано для воспроизведения нового звука
<code>SND_NOWAIT</code>	Если драйвер занят, функция сразу вернется без воспроизведения заданного звука
<code>SND_PURGE</code>	Останавливается воспроизведение любых звуков, вызванных в данной задаче. Если <code>pszSound</code> не 0, то останавливаются все экземпляры указанного звука. Если <code>pszSound</code> равен 0, то останавливаются все звуки, связанные с данной задачей. Отдельно надо указать дескриптор для остановки события <code>SND_RESOURCE</code>
<code>SND_RESOURCE</code>	Параметр <code>pszSound</code> является идентификатором ресурса. Параметр <code>hmod</code> должен указывать на источник ресурса
<code>SND_SYNC</code>	Синхронное воспроизведение звука события. Функция <code>PlaySound</code> возвращается только после окончания воспроизведения

Функция `PlaySound` при успешном выполнении возвращает `true`, в противном случае — `false`.

Примеры использования функции `PlaySound` приведены в главе 4. Функция используется в подключаемой DLL-библиотеки для формирования звуковых сигналов кода Морзе.

Использовать эту функцию можно для создания в работающей программе голосовых подсказок. Для этого с помощью микрофона и программы записи звука от Windows создать (наговорить в микрофон и записать в файл) необходимое количество голосовых файлов. При создании проекта программы в определенных местах включается функция `PlaySound` для выдачи нужного голосового сигнала или сообщения.

Создаем синусоидальный звук программированием

Очень часто, особенно в программах для любительской радиосвязи, возникает необходимость создавать *гармонические* (синусоидальные) звуки определенных частот и длительностей.

Сейчас мы рассмотрим проект приложения, в котором будет показан один из простейших вариантов создания гармонического звукового сигнала.

Созданный для учебных целей проект приложения может служить примером при создании вами собственных вариантов программирования звука. Файл с кодами (.cpp) имеет название `sinwave.cpp`, а файл проекта называется `sinwave1.bpr`.

Один из возможных вариантов заполнения компонентами формы проекта представлен на рис. 5.7.

Все установленные на форме компоненты вам хорошо известны — это `Label` и `Button`. Никаких изменений в файле `sinwave.h` делать нет необходимости, поэтому приводить текст этого файла не нужно.

Текст файла `sinwave.cpp` представлен в листинге 5.5.

Листинг 5.5. Файл `sinwave.cpp`

```
//-----
#include <vcl.h>
#pragma hdrstop
#include "sinwave.h"
#include <math.h>
#include <mmsystem.h>
//-----
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
```

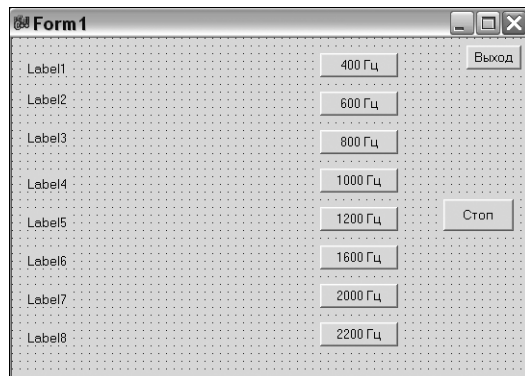


Рис. 5.7. Форма проекта

```
    HWND m_hWnd;
    HWAVEIN hWaveIn;
    HWAVEOUT hWaveOut;
    int ton = 1600; //440;
    const nwh = 3;
    const BUFLen = 44100; WAVEHDR wh[nwh];
    WAVEHDR whout[nwh];
    char buf[BUFLen*nwh];
    char bufout[BUFLen*nwh];
    int OutCnt = 0;
    DWORD SoundLevel[BUFLen];
    BYTE interbuf[BUFLen];
    bool bWaveInIsStarted = 0;
    bool bWaveOutIsStarted = 0;

    void __fastcall TestOut(int);
//-----
// Функция описания формы
__fastcall TForm1::TForm1(TComponent* Owner)           // 1
    : TForm(Owner)
{
    Caption = "Гармонический сигнал";
    Label1->Caption = "Частота звука  400 Герц";
    Label2->Caption = "Частота звука  600 Герц";
    Label3->Caption = "Частота звука  800 Герц";
    Label4->Caption = "Частота звука 1000 Герц";
    Label5->Caption = "Частота звука 1200 Герц";
    Label6->Caption = "Частота звука 1600 Герц";
    Label7->Caption = "Частота звука 2000 Герц";
    Label8->Caption = "Частота звука 2200 Герц";
}
//-----
// Функция реагирует на нажатие кнопки Выход
void __fastcall TForm1::Button1Click(TObject *Sender)    // 2
{
    Close();
}
//-----
// Функция реагирует на нажатие кнопки Стоп — прекращает звук
void __fastcall TForm1::StopOutClick(TObject *Sender)    // 3
{
    if (!bWaveOutIsStarted)
        return;
```

```

    bWaveInIsStarted = 0;

    MMRESULT mmresult;
    mmresult= waveOutPause( hWaveOut );
    if (mmresult!=MMSYSERR_NOERROR)
        return;

    mmresult = waveOutReset( hWaveOut );
    if (mmresult!=MMSYSERR_NOERROR)
        return;

    for (int i = 0; i < nwh; i++)
    {
        waveOutUnprepareHeader(hWaveOut, whout+i, sizeof(WAVEHDR));
    }
    mmresult = waveOutClose( hWaveOut );
    if (mmresult!=MMSYSERR_NOERROR)
        return;
}
//-----
// Функция проигрывает звук, записанный в буфере
void __fastcall TestOut(int) // 4
{
    int NumInputDevices = waveOutGetNumDevs();
    UINT uDeviceID = 0;
    WAVEOUTCAPS woc;
    MMRESULT mmresult;
    mmresult = waveOutGetDevCaps( uDeviceID, &woc,
                                   sizeof(woc) );

    if(mmresult!=MMSYSERR_NOERROR)
        return;
    WAVEFORMATEX wfx;
    wfx.wFormatTag = WAVE_FORMAT_PCM;
    wfx.nChannels = 1;
    wfx.nSamplesPerSec = 44100;
    wfx.nAvgBytesPerSec = 11025;
    wfx.nBlockAlign = 1;
    wfx.wBitsPerSample = 8;
    wfx.cbSize = 0;

    DWORD fdwOpen = /* WAVE_FORMAT_DIRECT; |*/ CALLBACK_WINDOW;

```

```

mmresult = waveOutOpen(&hWaveOut, uDeviceID, &wfx,
                      (ULONG)m_hWnd, 0, fdwOpen);
if (mmresult!=MMSYSERR_NOERROR) return;

for (int i=0; i< nwh; i++)
{
    whout[i].lpData = bufout+i*BUFLen;
    whout[i].dwBufferLength = BUFLen;
    whout[i].dwFlags = 0;

    mmresult = waveOutPrepareHeader(hWaveOut,
                                    whout+i, sizeof(WAVEHDR));
    if (mmresult!=MMSYSERR_NOERROR) return;

    double scale = 2.0 * M_PI / BUFLen;
    int amplitude = 3; // Выбранная величина
    for(int j=0;j<BUFLen;j++)
    {
        int time = j; // Характеризует долю периода колебаний
        whout[i].lpData[j] = int(amplitude * sin(time * ton * scale));
    }

    // Проиграть buffer
    mmresult = waveOutWrite(hWaveOut, whout+i, sizeof(WAVEHDR));
    if (mmresult!=MMSYSERR_NOERROR)
        return;
}
bWaveOutIsStarted = 1;
}

//-----
// Функция реагирует на нажатие кнопки
void __fastcall TForm1::Button2Click(TObject *Sender) // 5
{
    ton = 440;
    TestOut(ton);
}

//-----
void __fastcall TForm1::Button3Click(TObject *Sender) // 6
{
    ton = 6000;
    TestOut(ton);
}

//-----
void __fastcall TForm1::Button4Click(TObject *Sender) // 7

```

```

{
    ton = 800;
    TestOut(ton);
}
//-----
void __fastcall TForm1::Button5Click(TObject *Sender)           // 8
{
    ton = 1000;
    TestOut(ton);
}
//-----
void __fastcall TForm1::Button6Click(TObject *Sender)           // 9
{
    ton = 1200;
    TestOut(ton);
}
//-----
void __fastcall TForm1::Button7Click(TObject *Sender)           // 10
{
    ton = 1600;
    TestOut(ton);
}
//-----
void __fastcall TForm1::Button8Click(TObject *Sender)           // 11
{
    ton = 2000;
    TestOut(ton);
}
//-----
void __fastcall TForm1::Button9Click(TObject *Sender)           // 12
{
    ton = 2200;
    TestOut(ton);
}
//-----

```

Обратите внимание, что в текст файла `sinwave.cpp` добавлены два подключаемых заголовочных файла: `<math.h>` и `<mmsystem.h>`, а также добавлены объявления большого числа переменных величин, задействованных в функциях этого проекта.

В начале файла следует также объявить функцию `void __fastcall TestOut(int)`, которая объявляется здесь только по той причине, что не является членом класса `TForm`, функции которого объявляются в файле `sinwave.h`.

Все задействованные в проекте программы функции пронумерованы для удобства их нахождения в листинге. Номера проставлены как комментарии.

Функция 1 представляет собой описание формы и в ее теле задано название формы и названия всех компонентов `Label`.

Функция 2 прекращает работу программы и закрывает ее.

Функция 3 служит для прекращения проигрывания звука. Функция 4 выполняет проигрывание звука с заданной частотой звука.

Следует заметить, что указанные величины частот могут не соответствовать истинным величинам, измеренным при помощи частотомера. Целью этого примера было только показать один из возможных вариантов получения синусоидального звука. Для тех читателей, которые хотели бы познакомиться с различными методами получения звука цифровыми методами рекомендую прочитать книгу Тима Кинтцеля «Руководство программиста по работе со звуком».

Функции 5–12 являются реакциями на нажатие соответствующих кнопок и выполняют проигрывание звука, записанного в цифровом виде в буфере.

На рис. 5.8 показано главное окно работающего приложения. При нажатии на любую из кнопок, на которых указана величина частоты, начинается звучание гармонического сигнала с примерно указанной частотой. Звучание длится около 3–4 с. С помощью кнопки **Стоп** можно прервать звучание в любой момент.



Рис. 5.8. Проверка гармонического звука

Подобного типа программу можно использовать при разработке проекта программы предназначенной для передачи телеграфного сигнала. Результаты будут значительно лучшими, чем при использовании функции `PlaySound`. Можете сами такое сделать.