ГЛАВА 8

Системный динамик

Системный динамик является таким же древним устройством, как и сам X86-совместимый компьютер. Он представляет собой маленький динамик, расположенный в системном корпусе, и несколько дополнительных электронных компонентов на материнской плате. В прошлом веке, когда не было звуковых плат, именно системный динамик или, как его еще называют, спикер выполнял основные функции по извлечению звуков. Большинство старых игрушек под DOS использовали спикер для озвучивания различных игровых ситуаций: от стрельбы до звуков летящего самолета. К сожалению, качество звука оставляло желать лучшего. Потом появились отдельные устройства для воспроизведения звука, и спикер перестали применять в игровых и мультимедийных программах. Однако и по сей день на подавляющем большинстве компьютерных систем установлен маленький круглый динамик. При каждой загрузке компьютера раздается одиночный звуковой сигнал, подтверждающий успешное тестирование оборудования и выполнение операции начальной загрузки (POST). Для этого используется именно спикер, поскольку ни звуковая карта, ни любое другое устройство еще не может полноценно функционировать. Разработана целая комбинация различных звуковых кодов, позволяющих выявить сбои в подключенном оборудовании. Именно поэтому системный динамик установлен как в 486-м простеньком компьютере, так и в современном мультимедийном "монстре" на базе Pentium 4. Вы можете спросить, а зачем нужен спикер в операционных системах Windows, если там есть нормальная звуковая плата. Ответ очень прост: системный динамик гарантированно установлен на большинстве компьютеров и его можно применять для вывода системных сообщений и простых звуковых эффектов. Это, несомненно, даст вашей программе преимущество по сравнению с другими. В любом случае, вам самим решать, стоит ли использовать возможности системного динамика или нет, а я просто познакомлю вас с основными методами программирования спикера.

8.1. Программирование системного динамика

Для доступа к системному динамику используется порт с номером 61h. Он имеет размер 8 бит, но для управления спикером применяются только два младших бита (0 и 1): нулевой бит управляет включением канала 2 системного таймера, а первый бит включает динамик. Программирование системного таймера рассматривается в *гл. 10*. Установка этих битов в 1 позволяет включить динамик, а сброс — выключить. Рассмотрим пример для включения спикера (листинг 8.1).

Листинг 8.1. Включение системного динамика

```
Speaker_On proc near
push AX       ; сохраняем значение AX
in  AL, 61h   ; читаем состояние порта
or  AL, 00000011b; устанавливаем биты 0 и 1 в 1
out 61h, AL   ; включаем системный динамик
pop AX        ; восстанавливаем AX
ret
Speaker_On endp
```

Для выключения спикера можно применить код, представленный в листинге 8.2.

Листинг 8.2. Выключение системного динамика

```
Speaker_Off proc near
push AX       ; сохраняем значение AX
in  AL, 61h   ; читаем состояние порта
and AL, 11111100b; устанавливаем биты 0 и 1 в 0
out 61h, AL   ; включаем системный динамик
pop AX        ; восстанавливаем AX
ret
Speaker_Off endp
; реализуем работу спикера
call Speaker_On; включаем динамик
; выводим звук примерно 5 секунд
mov CX, 0050h ; старшее слово паузы
mov CX, 0B350h; младшее слово паузы
mov AH, 86h   ; функция BIOS паузы
int 15h       ; вызываем прерывание
call Speaker_On; включаем динамик
```

Аналогичный пример для C++ показан в листинге 8.3.

Листинг 8.3. Выключение системного динамика в C++

```
// пишем функцию управления динамиком
void Speaker ( bool bOn)
{
   DWORD dwResult = 0;
   // читаем состояние порта
   inPort ( 0x61, &dwResult, 1);
   if ( bOn) // включить динамик
   {
    dwResult |= 0x03;
    // записываем значение в порт
    outPort ( 0x61, dwResult, 1);
   }
   else // выключить динамик
   {
    dwResult &= 0xFC;
    outPort ( 0x61, dwResult, 1);
   }
}
// пишем реализацию работы спикера
Speaker ( true); // включаем динамик
delay ( 3000); // пауза 3 секунды, можно использовать функцию Sleep
Speaker ( false); // выключаем динамик
```

Рассмотренные функции управляют только включением и выключением динамика, сам же выводимый звуковой сигнал не изменяется по частоте. Чтобы немного преобразить наш звук, можно сделать так, как показано в листинге 8.4.

Листинг 8.4. Изменение частоты звука

```
mov BX, 2328h    ; определяем значение частоты
@set_tone:
call Speaker_On ; включаем динамик
mov  CX, 1388h  ; определяем значение для счетчика
@pause:
loop @pause      ; делаем паузу
call Speaker_Off; выключаем динамик
mov CX, 2328h
```

```
@pause2:
loop @pause2     ; делаем еще одну паузу
dec BX           ; уменьшаем значение в BX
jnz @set_tone    ; повторяем
```

Такой способ изменения частоты имеет серьезный недостаток — большая загрузка процессорного времени. Чтобы решить эту проблему, придется дополнительно программировать порты системного таймера. Я не буду здесь рассказывать о работе с таймером *(см. гл. 10)*, а просто приведу пример для настройки частоты выводимого звука в герцах (листинг 8.5).

Листинг 8.5. Управление частотой звука посредством системного таймера

```
SetFrequence proc near
; настраиваем регистр управления системного таймера
mov AL, 6Bh  ; канал 2, запись двух байтов, прямоугольные импульсы
out 43h, AL  ; записываем значение в управляющий порт таймера
; определяем значение частоты в 440 Гц ( 1 193 180 / 440)
mov AL, 98h  ; младший байт
out 42h, AL  ; записываем значение в порт динамика
mov AL, 0Ah  ; старший байт
out 42h, AL  ; записываем значение в порт динамика
call Speaker_On; включаем динамик
; устанавливаем паузу
mov  CX, 0C350h; определяем значение для счетчика
@pause:
loop @pause  ; делаем паузу
call Speaker_Off; выключаем динамик
ret
SetFrequence endp
```

Аналогичный пример для C++ показан в листинге 8.6.

Листинг 8.6. Управление частотой звука посредством системного таймера в C++

```
// реализуем функцию управления частотой
void SetFrequence ( unsigned int uFrequence)
{
   DWORD dwResult = 0;
   if ( uFrequence <= 0) return;
   int TimerClock = 1193180; // внутренняя частота таймера
   int iValue = 0; // значение делителя частоты
```

```
    // определяем значение делителя
    iValue = TimerClock / uFrequence;
    // настраиваем регистр управления системного таймера
    outPort ( 0x43, 0xB6, 1); // канал 2, 2 байта, прямоугольные импульсы
    dwResult = nDivider % 256; // младший байт делителя частоты
    outPort ( 0x42, dwResult, 2); // записываем значение в порт динамика
    dwResult = nDivider / 256; // старший байт делителя частоты
    outPort ( 0x42, dwResult, 2); // записываем значение в порт динамика
}
// попробуем вывести звуковой сигнал частотой примерно 1000 Гц
SetFrequence ( 1193); // устанавливаем желаемую частоту
Speaker ( true); // включаем динамик
Sleep ( 2000); // устанавливаем длительность сигнала равным 2 секунды
Speaker ( false); // выключаем динамик
```

Используя порты таймера для задания частоты, можно воспроизводить любые мелодии. Для тех, кто захочет написать полноценное музыкальное произведение, приведу значения частот всех нот в табл. 8.1.

Таблица 8.1. *Список частот*

Нота	Значение частоты, Гц
До	4186
До-диез	4435
Ре	4699
Ре-диез	4978
Ми	5274
Фа	5588
Фа-диез	5920
Соль	6272
Соль-диез	6645
Ля	7040
Ля-диез	7459
Си	7902

Как видите, нет ничего сложного в программировании системного динамика.