第7章

USB MIDI 键盘

音乐在人们的生活中起着至关重要的作用，难以想象，如果没有了音乐，世界将会变成什么样子。本章将在 USB 学习板上实现一个简易的 USB MIDI 键盘，通过它可以弹奏曲子，它也可以自动弹奏。通过对程序的简单修改，还可以实现 MIDI 接口转 USB 接口的功能。

7.1 MIDI 简介

MIDI(Musical Instrument Digital Interface)是乐器数字接口的缩写，是国际 MIDI 协会开发的一种通用标准。MIDI 传输和存储的并不是音频数据本身，而是演奏信息。这样可以大大节约存储空间，例如：一首 3 分钟的歌曲，如果保存为 CD 音质的 WAV 文件大概需要 30 MB 的存储空间，而 MIDI 文件则只需要几十 KB。播放 MIDI 时，将 MIDI 控制信息交给 MIDI 合成器，由 MIDI 合成器将声音合成出来。

1983 年国际乐器制造商协会 NAMM(National Association of Music Merchants)正式通过了 MIDI 标准 1.0 版本。MIDI 的出现是现代乐器制造史上的一个里程碑，它标志着以电子计算机技术为基础的高性能电子乐器进入市场的开始。很快，许多厂商相继推出了具有MIDI功能的电子乐器和相关外围设备，例如 Roland、YAMAHA 等乐器公司所生产的电子琴、鼓机、音序器等。

从 MIDI 出现到发展至今，具有 MIDI 接口的乐器得到了广泛的使用。在音乐的创作、制作，特别是在流行音乐领域，MIDI 乐器几乎无所不在，如 MIDI 键盘、MIDI 萨克斯管、MIDI 黑管等。

MIDI 音频与传统的音频概念上有着很大的区别。传统的音频录音，不管是模拟的还是数字的，都是直接将声音信息记录在介质上。而 MIDI 记录的并不是音频信息，而是演奏信息，如演奏的力度、音高等。在 MIDI 键盘上按下中央 C 音时，MIDI 键盘就可能会产生 3 字节的 MIDI 消息：90H、3CH、64H。其中，90H 表示在通道 0 上播放；3CH 表示音高，为中央 C；64H 表示力度。MIDI 合成器在收到该 MIDI 消息时，就会按照这些参数，以及选择好的音色来合成一个中央 C 的音。只要 MIDI 合成库的音色库里具有某种乐器，就可以使用 MIDI 键盘来演奏这种乐器，这足见 MIDI 的优点。

电子琴、鼓机等通常都具有 MIDI 接口，可以与其他 MIDI 设备相连。标准的 PC 声卡上也具有 MIDI 接口(通常和游戏控制口在一起，即声卡背后的那个 D 型连接头)，可以将电子琴、鼓机等 MIDI 设备与 PC 相连。利用计算机强大的处理能力，可以对 MIDI 进一步编辑和处理。对于电影、电视声音的后期制作，MIDI 技术更显现出其得天独厚的优势。计算机上的 MIDI 音乐容易制作和编辑，演奏速度可以随意改变，即使不会乐器，也可以使用计算机制作出音乐来。

7.2　MIDI 的工作原理

通常我们所听到的音乐都是乐队现场演奏，然后用录音技术直接录下来的。但是，直接录音需要巨大的数据量。以量化位数为 16 位，采样率为 44.1 kHz 的双声道 CD 来说，每秒产生的数据量为 16×2×44.1/8＝176.4 KB。那么能不能有一种办法，让计算机来“现场演奏”呢？这就是 MIDI 的基本思想，只保存演奏信息，然后由计算机“现场演奏”。对于 USB MIDI 键盘，就是通过 USB 接口发送弹奏信息给计算机，然后由计算机将声音合成出来。

音有几个基本的特征：音高、响度、音色和时值。只要这几个参数决定了，那么音也就确定了，MIDI 信息正是由这些参数组成的。通常，MIDI 合成器有 16 个不同的通道，可以给不同的通道(通道 9(第十通道)是打击乐专用)选择不同的乐器(即音色)。演奏时，不同的设备发送各自的控制信息给不同的通道，控制信息包括通道编号、音高、力度。当需要让某个音停止发声时，就关闭它(通常是发一个力度为 0 的命令)，从而控制了时值。

MIDI 控制命令有很多，这里不详细讲解了，只讲一个程序中用到的，就是让指定通道发声的消息——Note On Message。该命令的格式(十六进制)为 9n，kk，vv。其中，9 表示该消息的 ID；n 表示发送到的通道，可以选择为 0～15，共有 16 个通道；kk 为音符的音高，60(十进制)为中央 C 音；vv 表示演奏的力度，0 为关闭声音，127 为最大音量。

将 MIDI 信息转换为声音由 MIDI 合成器来完成。合成器通常使用波表(对实际乐器声音的波形采样然后建立的表格)来合成，它极大地决定了音色的好坏。相同的 MIDI 音源，在不同的合成器中所表现的效果可能会相差很远。像普通声卡所使用的软件波表合成器，延迟就会很明显，而使用高档一些的硬件波表合成声卡，则几乎感觉不到延迟。

7.3　USB MIDI 设备的数据流模型

一个 USB MIDI 设备可以有很多模块，例如 MIDI 合成器、MIDI 转换器(必须的)、插孔等。这里要设计的 MIDI 键盘没有合成器等模块，仅有转换器和 MIDI 插

孔。这里的插孔是一个抽象的概念，供MIDI信息输入输出使用；转换器的作用就是将各个插孔、模块连接起来。

USB MIDI设备使用批量端点来传输MIDI消息，一个批量端点可以传输多路MIDI消息。在USB批量端点传输的数据并不是原始的MIDI消息，而是由打包为32位的USB-MIDI事件包发送。具体可以参看USB MIDI设备类的文档，对于Note On消息，就是在前面加上0xP9，其中P为某一路MIDI消息的编号，本实例只有一路，设置为0即可。

本实例将在USB键盘的实例上进行修改，将UsbKeyboard复制一份，改名为UsbMidiKeyboard。USB MIDI设备在Windows XP下是自带驱动的，用户无须自行安装驱动，插上即可使用。

7.4 设备描述符

设备描述符只需要修改PID即可，其他内容保持不变。这已经是第八个程序了，将PID改成0x0008。

7.5 配置描述符集合

USB MIDI设备不需要HID描述符和报告描述符，将配置描述符集合当中的HID描述符删除，同时报告描述符、获取报告描述符的代码也一并删除。在USB MIDI设备中使用了MIDI流接口以及类特殊接口描述符和类特殊端点描述符，详细情况请看下面的介绍以及USB MIDI设备类的协议文档。

7.5.1 配置描述符

该USB MIDI键盘设备的配置需要两个接口，修改配置描述符中的bNumInterfaces字段为0x02。虽然在这里并没有用到音频控制接口，但是作为一个音频设备，音频控制接口是强制需要的。

7.5.2 音频控制接口描述符

每个音频设备都有一个音频控制接口AC(Audio Control)，可以有多个音频流接口或者MIDI流接口。在这里，音频控制接口没有实际的用途，仅是协议强制要求而已。所使用的端点数目为0，接口类为音频类(0x01)，子类为音频控制子类(0x01)，没有使用协议，为0值。修改好的音频控制接口描述符如下：

```
/*******************音频控制接口描述符 *********************/
 //bLength字段。接口描述符的长度为9字节
 0x09,
```

```
//bDescriptorType 字段。接口描述符的编号为 0x04
0x04,
//bInterfaceNumber 字段。该接口的编号,第一个接口,编号为 0
0x00,
//bAlternateSetting 字段。该接口的备用编号为 0
0x00,
//bNumEndpoints 字段。非 0 端点的数目。该接口没有端点
0x00,
//bInterfaceClass 字段。该接口所使用的类。音频接口类的代码为 0x01
0x01,
//bInterfaceSubClass 字段。该接口所使用的子类。音频控制接口的子类代码为 0x01
0x01,
//bInterfaceProtocol 字段。没有使用协议
0x00,
//iConfiguration 字段。该接口的字符串索引值。这里没有,为 0
0x00,
```

7.5.3　类特殊音频控制接口描述符

类特殊音频控制接口描述符用来描述该音频控制接口的属性,这里仅有一个头描述,它用来说明从属于该音频控制接口的其他接口有哪些。该描述符的结构如表 7.5.1 所列,其中取值一栏为本实例所使用的值,具体的代码请参看光盘中的源代码。这里只有一个流接口,并且它的编号为 1,所以在这里最后 2 字节的值都为 0x01。

表 7.5.1　类特殊音频控制接口描述符的结构

偏移量	域	大　小	取　值	描　述
0	bLength	1	0x09	该描述符的大小
1	bDescriptorType	1	0x24	描述符类型(CS_INTERFACE)
2	bDescriptorSubtype	1	0x01	描述符子类(MS_HEADER)
3	bcdADC	2	0x0100	该协议的版本号(1.0)
5	wTotalLength	2	0x0009	该描述的总长度
7	bInCollection	1	0x01	流接口的数目(只有一个)
8	baInterfaceNr(1)	1	0x01	哪个 MIDI 流接口属于该音频控制接口(这里为接口 1,即 MIDI 流接口)

7.5.4　MIDI 流接口描述符

MIDI 流接口描述符使用标准的接口描述符类型,它使用两个批量端点来分别输入和输出 MIDI 流数据。将前面的音频控制接口描述符复制一份修改为 MIDI 流

接口描述符。这里它属于第二个接口，所以将 bInterfaceNumber 字段改为 0x01。端点数目 bNumEndpoints 字段修改为 0x02，类代码依旧为音频类，子类代码 bInterfaceSubClass 字段改为 0x03，表示 MIDI 流接口子类。修改好的 MIDI 流接口描述符代码如下：

```
/ ********************MIDI 流接口描述符 ********************/
 //bLength 字段。接口描述符的长度为 9 字节
 0x09,

 //bDescriptorType 字段。接口描述符的编号为 0x04
 0x04,

 //bInterfaceNumber 字段。该接口的编号，第二个接口，编号为 1
 0x01,

 //bAlternateSetting 字段。该接口的备用编号为 0
 0x00,

 //bNumEndpoints 字段。非 0 端点的数目。MIDI 流接口使用一对批量输出/输出端点
 0x02,

 //bInterfaceClass 字段。该接口所使用的类。音频接口类的代码为 0x01
 0x01,

 //bInterfaceSubClass 字段。该接口所使用的子类。MIDI 流接口的子类代码为 0x03
 0x03,

 //bInterfaceProtocol 字段。没有使用协议
 0x00,

 //iConfiguration 字段。该接口的字符串索引值。这里没有，为 0
 0x00,
```

7.5.5 类特殊 MIDI 流接口描述符

类特殊 MIDI 流接口描述符用来描述 USB MIDI 设备内各种模块（包括元件与插孔）间的连接关系。在 USB MIDI 设备中有 4 种插孔：内嵌输入插孔、内嵌输出插孔、外部输入插孔和外部输出插孔。注意：这里的输入和输出是针对设备来说的，与前面所说的端点方向刚好相反。

内嵌插孔是一种逻辑插孔，它是由 USB 端点产生的。一个 USB 批量输出端点可以设置一到多个内嵌输入插孔，USB 发出的数据从该插孔流入 USB MIDI 设备中。一个 USB 批量输入端点可以设置一到多个内嵌输出插孔，USB MIDI 设备的输出数据流（例如从外部输入插孔上获取或者本身键盘产生）从该插孔流出到主机中。

外部插孔为实际的（物理上的）MIDI 端子，可以用来连接外部的 MIDI 设备。事实上也可以为逻辑上的虚拟端子，例如，本例中虚拟的 MIDI 键盘输入插孔，但是主机并不知道这些细节，都把它们当作实际的物理插孔来看待。外部输入插孔用来接收其他 MIDI 设备发来的数据，外部输出插孔则将数据发送给其他 MIDI 设备。其实标准的 MIDI 接口所

使用的传输协议很简单，它就是标准的 UART 串口，波特率为 31.25 kb/s。

在 USB MIDI 设备中，每一个插孔都有一个唯一的 ID 号，用来标识它的身份。对于输出插孔，它可以具有多个输入引脚，用来选择哪些输入插孔作为其输入源。例如，要做的 USB MIDI 键盘，可以在内嵌输出插口（数据将输出给计算机）描述符中指定一个外部输入插孔作为其数据源；而如果要将计算机端的 MIDI 数据发送到实际的 MIDI 接口，则可以在一个外部输出插孔描述符中指定内嵌输入插孔作为其数据源。

输入插孔和输出插孔分别由 MIDI 输入插孔描述符（MIDI IN jack descriptor）和 MIID 输出插孔描述符（MIDI OUT jack descriptor）描述。此外，还有一个元件描述符（element descriptor），它用来描述 USB MIDI 设备的元件，例如 MIDI 合成器等（本实例没有用到）。所有的这些类特殊 MIDI 流接口描述符由一个头描述符来引导：类特殊 MIDI 流接口头描述符（class-specific MS interface header descriptor）。

下面将依次讲述这些描述符的结构以及具体的实现类特殊 MIDI 流接口头描述符的结构如表 7.5.2 所列。其中，wTotalLength 为整个类特殊 MIDI 流接口描述符的总长度，视具体的描述符而定，该实例中为 37 字节。

表 7.5.2 类特殊 MIDI 流接口头描述符的结构

偏移量	域	大 小	取 值	描 述
0	bLength	1	0x07	该描述符的字节数
1	bDescriptorType	1	0x24	该描述符的类型（CS_INTERFACE）
2	bDescriptorSubtype	1	0x01	该描述符子类（MS_HEADER）
3	bcdMSC	2	0x0100	MIDI 流子类的协议版本号（1.0）
5	wTotalLength	2	0x0025	整个类特殊 MIDI 流接口描述符的总长度，包括头描述符、各种插孔描述符、元件描述符的长度总和

说明：取值一栏的数据为本实例所使用的数据，实际设计时根据需要设置。

MIDI 输入插孔描述符的结构如表 7.5.3 所列。其中，bJackType 字段为插孔的类型，可以选择内嵌（0x01）或者外部（0x02）。bJackID 字段为插孔的唯一 ID，可以用在输出插孔的输入源选择中。

表 7.5.3 MIDI 输入插孔描述符的结构

偏移量	域	大 小	取 值	描 述
0	bLength	1	0x06	该描述符的字节数
1	bDescriptorType	1	0x24	该描述符的类型（CS_INTERFACE）
2	bDescriptorSubtype	1	0x02	该描述符的子类（MIDI_IN_JACK）
3	bJackType	1	0x01 或 0x02	插孔的类型（EMBEDDED 或 EXTERNAL）
4	bJackID	1	ID	该插孔的唯一 ID
5	iJack	1	0x00	描述该插孔的字符串描述符的索引值

说明：Value 一栏的数据为本实例所使用的数据，实际设计时根据需要设置。

MIDI输出插孔描述符的结构如表7.5.4所列。输出插孔描述符和输入插孔描述符的前5字节内容意义是一样的。bNrInputPins字段为该输出插孔的输入引脚数量，一个输出插孔可以有多个输入引脚，从而可以把多个输入插孔连接到输出插孔，以提供输出数据。baSourceID为连接到该输出插孔的输入插孔的ID号；BaSourcePin为输入源(上述输入插孔)连接在该输出插孔的输入引脚号。中间省略号部分表示可以有多组baSourceID和BaSourcePin。iJack为描述该插孔的字符串描述符的索引，本实例中没有该字符串。

表7.5.4 MIDI输出插孔描述符的结构

偏移量	域	大 小	取 值	描 述
0	bLength	1	长度	该描述符的长度
1	bDescriptorType	1	0x24	该描述符的类型(CS_INTERFACE)
2	bDescriptorSubtype	1	0x03	该描述符的子类(MIDI_OUT_JACK)
3	bJackType	1	0x01或0x02	插孔的类型(EMBEDDED或EXTERNAL)
4	bJackID	1	ID	该插孔的唯一ID
5	bNrInputPins	1	n	该输出插孔的输入引脚数
6	baSourceID(1)	1	ID	连接到该输出插孔的输入引脚的输入插孔的ID号
7	BaSourcePin(1)	1	引脚	连接到该输出插孔的那个输入引脚
⋮	⋮	⋮	⋮	⋮
	iJack	1	0x00	描述该插孔的字符串描述符的索引值

Element Descriptor描述符在本实例中没有用到，这里就不详述了，感兴趣的读者可以查看USB MIDI设备协议。

本实例设置了4个插孔，分别为内嵌输入插孔、内嵌输出插孔、外部输入插孔和外部输出插孔。键盘的数据通过内嵌输出插孔输出给计算机，但是键盘的数据不能直接发送到内嵌输出插孔，而是假设有一个外部输入插孔(虚拟的)，将该输入插孔连接到内嵌输出插孔的输入引脚上。对于计算机端输出的数据也相类似，数据将到达内嵌输入插孔，将内嵌输入插孔连接到外部输出插孔的输入引脚上，数据就可以发送出去了。这里指定内嵌输入插孔的ID为1，外部输入插孔的ID为2，内嵌输出插孔的ID为3，外部输出插孔的ID为4。在内嵌输出插孔的baSourceID中指定ID为2，即外部输入插孔。在外部输出插孔的baSourceID中指定ID为1，即内嵌输入插孔。最终设置好的类特殊MIDI流接口描述符如下：

```
/**********内嵌输入插孔描述符 *********/
 //bLength 字段。该描述符的长度，为 6 字节
 0x06,

 //bDescriptorType 字段。该描述符的类型为 CS_INTERFACE
```

```
0x24,
//bDescriptorSubtype 字段。描述符子类,为 MIDI_IN_JACK
0x02,
//bJackType 字段。该插孔的类型为内嵌
0x01,
//bJackID 字段。该插孔的唯一 ID,这里取值 1
0x01,
//iJack 字段。该插孔的字符串描述符索引,这里没有,为 0
0x00,
/**********外部输入插孔描述符*********/
//bLength 字段。该描述符的长度,为 6 字节
0x06,
//bDescriptorType 字段。该描述符的类型为 CS_INTERFACE
0x24,
//bDescriptorSubtype 字段。描述符子类,为 MIDI_IN_JACK
0x02,
//bJackType 字段。该插孔的类型为外部
0x02,
//bJackID 字段。该插孔的唯一 ID,这里取值 2
0x02,
//iJack 字段。该插孔的字符串描述符索引,这里没有,为 0
0x00,
/**********内嵌输出插孔描述符*********/
//bLength 字段。该描述符的长度为 9 字节
0x09,
//bDescriptorType 字段。该描述符的类型为 CS_INTERFACE
0x24,
//bDescriptorSubtype 字段。描述符子类为 MIDI_OUT_JACK
0x03,
//bJackType 字段。该插孔的类型为内嵌
0x01,
//bJackID 字段。该插孔的唯一 ID,这里取值 3
0x03,
//bNrInputPins 字段。该输出插孔的输入引脚数。这里仅有一个
0x01,
//baSourceID 字段。连接到该插孔输入引脚的输入插孔的 ID,选择为外部输入插孔
```

```
0x02,
//BaSourcePin 字段。外部输入插孔连接到该插孔的输入引脚 1 上
0x01,
//iJack 字段。该插孔的字符串描述符索引,这里没有,为 0
0x00,
/**********外部输出插孔描述符*********/
//bLength 字段。该描述符的长度,为 9 字节
0x09,
//bDescriptorType 字段。该描述符的类型为 CS_INTERFACE
0x24,
//bDescriptorSubtype 字段。描述符子类,为 MIDI_OUT_JACK
0x03,
//bJackType 字段。该插孔的类型为外部
0x02,
//bJackID 字段。该插孔的唯一 ID,这里取值 4
0x04,
//bNrInputPins 字段。该输出插孔的输入引脚数。这里仅有一个
0x01,
//baSourceID 字段。连接到该插孔输入引脚的输入插孔的 ID,选择为内嵌输入插孔
0x01,
//BaSourcePin 字段。内嵌输入插孔连接到该插孔的输入引脚 1 上
0x01,
//iJack 字段。该插孔的字符串描述符索引,这里没有,为 0
0x00,
```

7.5.6 端点描述符和类特殊端点描述符

在 USB MIDI 设备中,除了有标准的批量数据端点描述符之外,还有类特殊 MIDI 流批量数据端点描述符(class-specific MS bulk data endpoint descriptor),它们用来描述内嵌插孔是如何在端点上组织的。每个标准的批量数据端点描述符后面跟一个类特殊批量数据端点描述符。

类特殊 MIDI 流批量数据端点描述符的结构如表 7.5.5 所列。其中 bNumEmbMIDIJack 字段为分配在该批量端点的内嵌插孔的数量,在本实例中,仅有一个内嵌输入插孔或一个内嵌输出插孔,所以值为 1。baAssocJackID 字段为分配在此批量端点的内嵌插孔的 ID 号,可以有多个,视该端点内嵌插孔的数量而定;对于输入端点,指定为内嵌输出插孔的 ID 号,对于输出端点,指定为内嵌输入插孔的 ID 号。

表 7.5.5　类特殊 MIDI 流批量数据端点描述符的结构

偏移量	域	大　小	取　值	描　述
0	bLength	1	4+n	该描述符的长度
1	bDescriptorType	1	0x25	该描述符的类型(CS_ENDPOINT)
2	bDescriptorSubType	1	0x01	该描述符的子类(MS_GENERAL)
3	bNumEmbMIDIJack	1	n	分配在此端点的内嵌插孔的数量
4	baAssocJackID(1)	1	ID1	分配在此端点的第一个内嵌插孔的 ID 号
⋮	⋮	⋮	⋮	⋮
4+(n−1)	baAssocJackID(n)	1	IDn	分配在此端点的最后一个内嵌插孔的 ID 号

在本实例中，需要一个批量输入端点和一个批量输出端点，这里使用 D12 的端点 2 来实现。在每个标准端点描述符下增加一个类特殊端点描述符，最终实现的代码如下：


```
/**************标准批量数据输入端点描述符*****************/
//bLength 字段。端点描述符长度为 7 字节
0x07,

//bDescriptorType 字段。端点描述符的编号为 0x05
0x05,

//bEndpointAddress 字段。端点的地址，这里使用 D12 的输入端点 2
//D7 位表示数据方向，输入端点 D7 为 1，所以输入端点 2 的地址为 0x82
0x82,

//bmAttributes 字段。D1～D0 为端点传输类型选择。该端点为批端点
//批量端点的编号为 2，其他位保留为 0
0x02,

//wMaxPacketSize 字段。该端点的最大包长；端点 2 的最大包长为 64 字节；注意低字节在先
0x40,
0x00,

//bInterval 字段。端点查询的时间，此处无意义
0x00,

/***********类特殊 MIDI 流批量数据端点描述符*****************/
//bLength 字段，该描述符的长度为 5 字节
0x05,

//bDescriptorType 字段，该描述符的类型为类特殊端点描述符(CS_ENDPOINT)
0x25,

//bDescriptorSubType 字段，该描述符的子类型为 MS_GENERAL
0x01,

//bNumEmbMIDIJack 字段，该端点的内嵌输出插孔的数量，这里只有 1 个
0x01,
```

```
    //baAssocJackID 字段,该端点的内嵌输出插孔的 ID 号
    //在前面定义了一个内嵌输出插孔,ID 号为 3
    0x03,

    /**************标准批量数据输出端点描述符*****************/
    //bLength 字段。端点描述符长度为 7 字节
    0x07,

    //bDescriptorType 字段。端点描述符的编号为 0x05
    0x05,

    //bEndpointAddress 字段。端点的地址,这里使用 D12 的输出端点 2
    //D7 位表示数据方向,输出端点 D7 为 0,所以输出端点 2 的地址为 0x02
    0x02,

    //bmAttributes 字段。D1～D0 为端点传输类型选择。该端点为批端点
    //批量端点的编号为 2,其他位保留为 0
    0x02,

    //wMaxPacketSize 字段。该端点的最大包长:端点 2 的最大包长为 64 字节;注意低字节在先
    0x40,
    0x00,

    //bInterval 字段。端点查询的时间,此处无意义
    0x00,

    /**********类特殊 MIDI 流批量数据端点描述符*********/
    //bLength 字段,该描述符的长度为 5 字节
    0x05,

    //bDescriptorType 字段,该描述符的类型为类特殊端点描述符(CS_ENDPOINT)
    0x25,

    //bDescriptorSubType 字段,该描述符的子类型为 MS_GENERAL
    0x01,

    //bNumEmbMIDIJack 字段,该端点的内嵌输入插孔的数量,这里只有 1 个
    0x01,

    //baAssocJackID 字段,该端点的内嵌输入插孔的 ID 号。在前面定义了一个内嵌输入插孔,
ID 号为 1
    0x01
```

7.5.7 字符串描述符

字符串描述符可以按照自己的需要和喜好来修改,这里将产品字符串改为“《圈圈教你玩 USB》之 USB MIDI 键盘”,设备序列号改为“2008 - 08 - 08”(2008 北京奥运会开幕式时间)。

7.6 修改好描述符后的测试

实际上,在 USB MIDI 协议中还规定了一些类特殊请求,但是在该设备中并不

会使用到，因而也就没去实现它们。目前我们的程序仅完成了描述符的修改，对具体的 MIDI 数据处理还没有实现，但是可以先测试一下，描述符是否修改正确了。如果正确的话，就会显示出一个 USB 音频设备。

将修改好的程序编译，并下载到学习板中运行。可以看到弹出发现新硬件的对话框，如图 7.6.1所示。等设备的驱动程序安装完成后，在设备管理器中可以看到一个 USB Composite Device(USB 复合设备)和一个 USB Audio Device(USB 音频设备)，如图 7.6.2 所示。剩下的工作就是对两个批量端点的处理了。

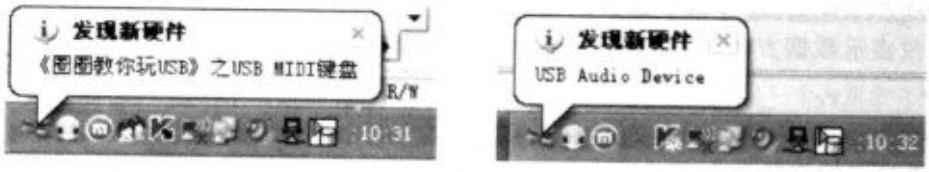

图 7.6.1　发现新硬件对话框

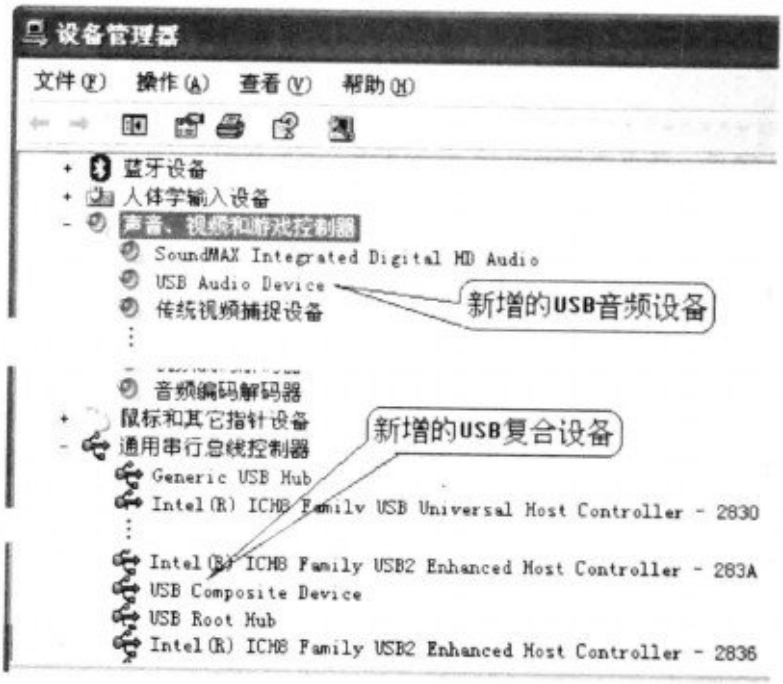

图 7.6.2　设备管理器

7.7　USB MIDI 键盘的数据返回

对于学习板上按键的按下，如何转换为 MIDI 消息发送到 USB 输入端点中去呢？在前面有提到过，要打包为 32 位的 MIDI 事件包发送。本 USB MIDI 键盘功能比较简单，仅能产生一条 Note On(音符开)消息，它的 32 位 MIDI 事件包格式(十六进制)为：P9，9n，kk，vv。其中，第一字节 P9 为 USB MIDI 协议中增加的包头，P 为某一路 MIDI 消息的编号(这里仅有一个内嵌输出插孔，为 0)，9 为该包的 ID 标识。后面的 3 字节为实际的 MIDI 消息，9n 表示在通道 n 上发送 Note On 消息，kk 表示

音符的音高，vv 表示音符的力度(响度，127 为最大声)。

通道和力度容易理解，就这个音符的音高的值稍微麻烦点。在钢琴上，有个中央 C 键，而这里的音高值为 60 时，对应的就是钢琴上的中央 C 键。MIDI 消息中音高的每个值对应着钢琴上的一个键(包括黑键)，例如 59 就对应着中央 C 左边的白键，而 61 就对应着中央 C 右边的黑键。在音乐中，规定了一个标准音高 A，它的频率为 440 Hz。当一个音的频率升高一倍时，就叫做升高了一个八度。例如将 440 Hz 的 A 音升高到 880 Hz(还叫做 A 音，但用后缀来区分)，就升高了一个八度。在 440～880 Hz 这个八度之内，按照指数的关系平均分成 12 等份(每个音的频率比它前一个音的频率高 2 的 1/12 次方，即 12 平均律)，就得到了 12 个频率的音，分别记做 A、#A、B、C、#C、D、#D、E、F、#F、G、#G。其中带有 # 号的就是钢琴中的黑键，没有 # 号的就是钢琴上的白键。相差一个八度的音，所使用的字母是一样的，但使用大小写以及带后缀来区分。钢琴上的键就是按照上述的关系排布的，MIDI 中的音高值也是。像这样的两个相邻的音之间的频率差距，叫做一个半音，也叫做一个小二度。例如 B、C 之间相差一个半音，C、#C 之间相差一个半音。距离为两个半音叫做一个全音(或者叫大二度)，例如 C、#A 之间相差一个全音，C、D 之间相差一个全音。

但是在音乐中常使用简谱来记谱，那么简谱跟刚刚所说的这些音之间有什么关系呢？简谱使用 1～7 这 7 个阿拉伯数字来表示，其中，3 和 4、7 和 1 之间相差一个半音，其他相邻音之间相差为一个全音。为了表示全音中间的半音，可以增加 # 号来表示。这样在一个八度中，简谱的 12 个音就是：1、#1、2、#2、3、4、#4、5、#5、6、#6、7。那么 1 的频率是多少呢？1 的频率是不固定的，可以在前面用字母表示的音符中任选一个作为 1。这样的表示叫做相对音高，而用字母表示的则是绝对音高。在简谱中，不带 # 号的 7 个音叫做自然音阶，大部分歌曲中只使用自然音阶(另外像五声调式中，只有五个音，没有 4 和 7)。如果一个曲以 1 为主音，那么它就叫做大调，如果以 6 为主音，那么就叫做小调。大调给人明亮、欢快的感觉，而小调则给人压抑、忧伤的感觉。如果一个大调的歌曲选择 C 作为 1，就叫做 C 大调，选择 G 作为 1，就叫做 G 大调。通过对照可以发现，如果选择 C 作为 1，那么所有的自然音阶都刚好落在钢琴的白键上，这就是为什么很多曲子是 C 大调(或者 A 小调，选择 A 为 6)的原因。

本 USB 学习板仅有 8 个键，为了能够演奏一些简单的乐曲，将其按照五声调式来排布，选择为 C 调。KEY1～KEY8 分别为简谱的 5、6、1、2、3、5、6、1，第一个 1 为中央 C，按照前面介绍的关系，可以计算出它们在 MIDI 消息中的音高值分别为 55、57、60、62、64、67、69、72。

将原来返回报告的函数 SendReport(改为 SendNoteOnMsg)，在该函数中根据不同按键的按下和弹起来发送 Note On 消息。当某个键按下时，就发送力度值为最大的 Note On 消息(由于本学习板的按键没有力度检测，只好发送最大力度了，在高档的电子琴中按键有力度检测，可以根据具体的弹奏力度来设置该消息)，这将让某

个音符发声；当某个键弹起时，就发送力度为 0 的消息，这将停止该音符的发声。由于 8 个按键的处理非常类似，这里只贴出 KEY1 的按下和弹起处理，其他部分省略。

```
/*************************************************************
函数功能：根据按键情况返回 Note On 消息的函数
入口参数：无
返　　回：无
备　　注：无
*************************************************************/
void SendNoteOnMsg(void)
{
 //4 字节的缓冲区
 uint8 Buf[4];

 //Note On 消息第一字节固定为 0x09，第二字节为 0x9n(n 为通道号)，第三字节为 0xKK(K 为音高)，第四字节为 0xVV(V 为力度)

 Buf[0] = 0x09;                    //Note On 消息的包头
 Buf[1] = 0x90;                    //在通道 0 上发送 Note On 消息
 Buf[3] = 0x7F;                    //音量设置为最大

 if(KeyDown&KEY1)
 {
  Buf[2] = 55;                     //C 调的 5(绝对音高为 G 音)
  //通过端点 2 返回 4 字节 MIDI 事件包
  D12WriteEndpointBuffer(5,4,Buf);
  Ep2InIsBusy = 1;                 //设置端点忙标志
  KeyDown& = ~KEY1;                //清除对应的按键
  return;                          //发送一个音符后就返回
 }
 ⋮                                 //此处省略部分代码
 //如果有按键弹起，则关闭对应的音
 Buf[3] = 0x00;                    //音量设置为 0

 if(KeyUp&KEY1)
 {
  Buf[2] = 55;                     //C 调的 5(绝对音高为 G 音)
  //通过端点 2 返回 4 字节 MIDI 事件包
  D12WriteEndpointBuffer(5,4,Buf);
  Ep2InIsBusy = 1;                 //设置端点忙标志
  KeyUp& = ~KEY1;                  //清除对应的按键
  return;                          //发送一个音符后就返回
 }
 ⋮                                 //此处省略部分代码
```

对于输出数据，由于本实验板无法设置为 31.25 kb/s 的波特率，所以对于端点 2 输出的数据就直接丢弃了，仅在端点 2 输出中断处理中清除中断标志和清空缓冲区。需要注意的是，对于输出数据的处理速度一定要够，否则可能会导致应用软件停止响应甚至整个操作系统崩溃。如果设备用不到 MIDI 输出，则干脆在外部输出插孔描述符中修改 baSourceID 字段为 0x02，选择输入源为外部输入插孔。这样内嵌输入插孔就没有被使用，从而 Windows 就不会增加 MIDI 输出设备。正常使用时，将 config.h 中定义的调试宏删除，避免多余的消耗。

7.8　USB MIDI 键盘的使用

这个 USB MIDI 键盘怎么使用呢？有很多软件是支持 MIDI 输入/输出设备的，比较有名的如 Cakewalk、Guitar Pro 等。在这里，介绍一款比较小巧的软件：HappyEO 电子琴软件，该软件可以在网上或者圈圈的 USB 小组中下载。

该 USB MIDI 键盘连接到计算机后，会产生一个 MIDI 输出设备，Windows 操作系统有时会自动将它选择为“MIDI 音乐播放”的默认设备。如果此时播放一个 MIDI 文件，数据将发送到 USB MIDI 设备去，从而计算机声卡无声音输出。可以进入控制面板的“声音和音频设备”中选择需要的 MIDI 设备，如图 7.8.1 所示。对于一般的声卡，都是使用微软的 GS 波表软件合成器，选择它就可以了。如果声卡比较高档，有硬件合成器，当然也可以选它们。如果想观察播放一个 MIDI 文件时到底发了些什么数据到 USB MIDI 设备中，可以选择下面的 USB 音频设备，然后打开 bushound，选择捕捉该设备的数据。这些数据都是一些 MIDI 消息，需要了解 MIDI 协议才能看懂它的含义，其中最多的就是 Note On 消息和 Note Off 消息了。如果不想改这个 MIDI 音乐播放的默认设备，可以参看 7.7 节的最后一段。

要在 HappyEO 电子琴软件中使用该设备，先需要对软件做一些设置。运行 HappyEO，单击 Option 按钮，在“常用”选项卡中，选择一个 MIDI 输出设备，这里要选择一个能够发声的 MIDI 设备，所以不能选择 USB Audio Device，如图 7.8.2 所示。

然后，再切换到 MIDI 输入标签，将“启用 MIDI 输入设备”勾选上，并在下面选择 USB Audio Device，如图 7.8.3 所示。

设置好之后，再按动学习板上的按键，是不是听到声音了？在 Option 按钮下面的一排数字按钮可以选择不同的乐器，还可以右击，对不同的按钮分配不同的乐器。不过这个键盘使用起来有难度，音域也很窄，要弹奏一个曲子可不容易，这里仅为演示用。既然人工演示起来麻烦，能不能叫单片机帮我们演奏一曲呢？只要按照事先编排好的曲子，一次次发送这些 MIDI 消息即可。下面一节就介绍如何实现这个单片机自动播放曲子，它已经跟 USB 没有多少关系了，放在这里的目的是为了让大家轻松一下，同时也可以学习一些编程的方法。

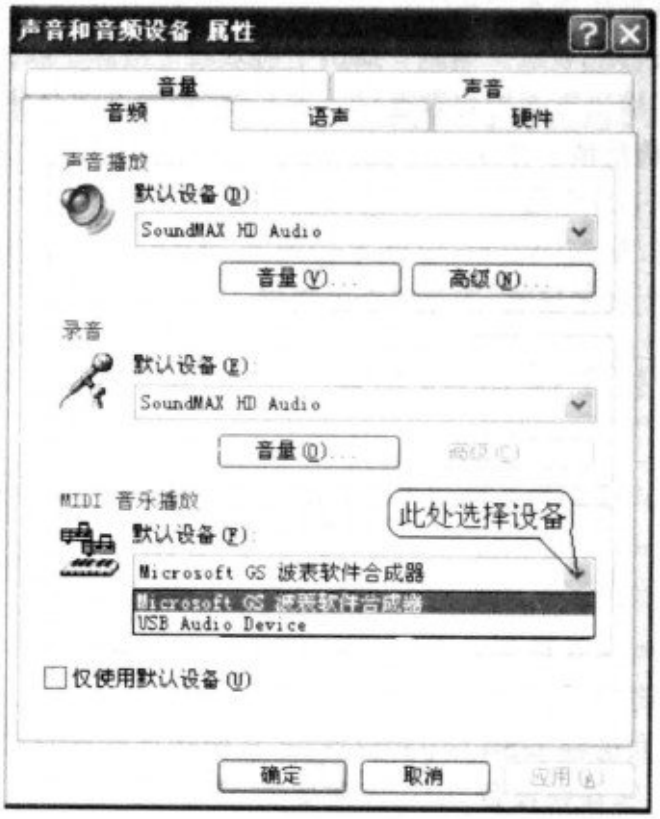

图 7.8.1　选择 MIDI 音乐播放的默认设备

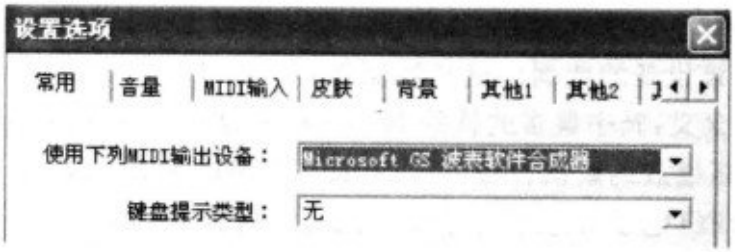

图 7.8.2　选择 MIDI 输出设备

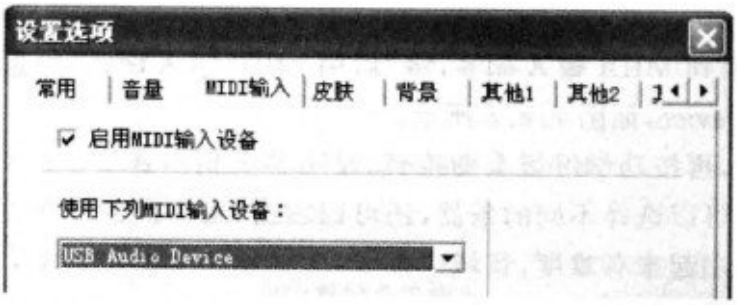

图 7.8.3　选择 MIDI 输入设备

7.9 单片机自动弹奏的实现

要单片机自动弹奏，当然要先给它一个曲子。这里挑一个比较简单、又比较好听的曲子：王菲的《容易受伤的女人》。找到曲子后，把每个音符翻译为MIDI消息，然后按照谱子中给定的时间间隔发送就行了。不过，如果我们只弹奏主旋律，将显得比较单调，应该再加上和弦和打击乐伴奏才好听。这样一个时刻就需要发送多个音符，如果把全部音符的MIDI消息都记录下来，数据量将会比较大。为了减少数据存储量，这里定义了一种结构：曲子以行的格式保存，每行在同一个时刻（实际上是比较短的时间内分多个包发送的）发声。每行的第一字节为该行中需要发声的个数，接下来的两字节分别是音符的音高和力度，一行有多个音时，音高和力度如此重复下去。最后两字节为该行音符发送后停留的时间。但是旋律音和打击乐是在不同的通道上的，那怎么区分通道呢？注意到数据中不会用到0xFF，为此利用0xFF作为通道切换指示。这里仅使用了两个通道，当遇到0xFF时，就切换到另一个通道。整个曲子保存在一个数组中，数组的前两字节为整个曲子的行数，接下来就是一行行的MIDI消息数据。具体的数据格式可以参看源代码中的song.c文件，里面包括了曲子的数据和播放曲子的函数。感兴趣的读者可以自己找个谱子，然后按照这个格式编一下。不过这个过程有点烦琐，还比较费时间。同时要注意别选太长的曲子，否则ROM会不够的。

怎么启动播放呢？这里使用KEY1和KEY8组合来播放，同时按下KEY1和KEY8后，将开始自动播放。你也可以自行决定使用哪些组合键，或者使用串口接收命令来播放等。在光盘中有该自动弹奏时的录音文件，读者可以播放来听听，由于没有使用一些音效，所以显得有点呆板。该录音是圈圈计算机上的MIDI合成器合成出来的声音，也许跟你在计算机上实验时的声音差别比较大，这是正常现象，也是MIDI的特性之一。

7.10 本章小结

本章简单介绍了USB MIDI键盘的实现，由于涉及一些MIDI以及音乐方面的知识，这里不方便详述，仅作了一点简单介绍。如果要制作一个完整、符合规范的USB MIDI键盘，还有很多工作要做。本章旨在介绍USB MIDI设备的基本功能实现，给读者有一个概念上的认识。