AVRUSB 技术探讨

AVR的速度刚好可以跟得上USB1.0 的通讯速度，所以可以模拟和USB通讯。USBASP就是这么来的（不知道什么是USBASP？就是下载器）。

摘要
本文介绍了一种独特的AVR单片机和计算机进行USB通信的方法：AVRUSB。介绍了AVRUSB技术的基本原理、特点、应用，同时还详细的介绍了AVRUSB系统的单片机软件开发和计算机软件开发的方法。

关键字：
AVRUSB，LibUSB，LibUSB-Win32

正文

1 简介

1.1 AVRUSB是什么

AVRUSB技术是利用高性能的 8 位RISC架构的AVR单片机，使用单片机的IO口来模拟USB的通信端口，由软件来实现USB通信协议，将普通的AVR单片机模拟成一个USB低速设备，从而实现AVR单片机与计算机之间的通信和控制。

AVRUSB 技术的基本原理就是利用AVR单片机的普通IO端口来模拟USB的硬件端口进行通信。因为低速USB设备的速度是 1.5M位/秒，而AVR单片机是单指令周期的，在单片机使用 12MHz的时钟频率时，正好是 1.5MHz的 8 倍。也就是说，单片机每 8 条指令就精确完成一个数据位的采集。采用这种方法时，对单片机的时序要求非常严格，所以软件的核心部分代码完全由汇编语言实现。

1.2 AVRUSB的历史

AVRUSB 技术最早的文档可见于AVR的官方应用笔记《AVR309 Software Universal Serial Bus (USB)》（软件USB）一文中。在这篇应用笔记里，详细的介绍了如何使用AVR单片机的普通IO口来实现USB通信，同时介绍了计算机的驱动程序以及计算机上用户程序的编程方法，并提供了全部源码。但是AVR309 中介绍的单片机程序完全是使用汇编语言编写的，不利于将程序移植到其他应用环境中，也不利于程序的维护（毕竟使用汇编语言的人相对比较少，使用C语言编程是大部分人的选择），这使得其应用受到了很多限制。

后来，OBJECTIVE公司推出了AVRUSB。它以汇编语言实现USB通信的底层接口，用C语言实现用户层的程序接口。用户接口被简化为很简单的几个函数（在最简化的情况下，只需要三个函数，一个初始化函数、一个轮询函数和一个数据处理函数）和一个配置文件，用户可以完全不会使用汇编语言编程。同时它还提供了一个稳定而成熟的用户程序框架，用户可以在此框架的基础上，通过修改和扩展接口函数的功能来实现各种USB通信功能和控制功能，实现各种带USB接口的应用系统。并且，AVRUSB支持目前最流行的AVR GCC编译器和IAR C编译器，因此具有很强的实用性。

1.3 AVRUSB的特点

1.3.1 低成本

传统的单片机与计算机进行USB通信，需要使用专用的接口芯片进行USB协议转换，如CP2101、FT232、CH342、PDIUSBD12、 SL811 等。象CP2101、FT232 这样的芯片使用起来虽然简单，但是功能比较单

一；而PDIUSBD12、SL811 功能较强，但是使用复杂。并且这些专用芯片的价格都相对较高，增加了系统的成本。而AVRUSB简单易用，成本低廉，只需要一个普通的低成本AVR单片机以及很少的几个外部元件，就可以组成一个USB系统。

AVRUSB 的代码为AVR GCC编译器做了高度优化，同时也完全兼容于更专业的IAR C编译器。程序编译后在最小情况下还不到 2KB，因此绝大部分的AVR单片机都可以使用AVRUSB（只要支持外部中断INT0，Flash容量不小于 2KB就可以实现AVRUSB的功能）。这样在很多低成本的小容量AVR单片机上也可以使用AVRUSB，如ATtiny2313、ATmega45、 ATmega48 等，因此AVRUSB技术具有很高的实用价值。

1.3.2 资源丰富，容易开发

AVRUSB 提供了一个完整而又简单易用、成熟稳定的应用程序框架。这个框架包括了底层（单片机部分）和上层（PC部分），单片机可以使用gcc（或者IAR）编程；PC上则可以使用各种通用编程软件，如Windows下使用VC、VB、Delphi、C++ Builder、BDS2006、GCC，Linux下使用GCC等等。用户可以在这个框架基础上添加和扩展各种功能，快速开发出适合于各种需求的单片机控制系统，而且AVRUSB支持Windows、Linux、MacOS等多种操作系统，具有很好的跨平台特性。

1.4 AVRUSB的应用

AVR单片机低成本、高性能的特性，使得AVRUSB非常适合于应用到USB加密狗、USB接口的系统控制、低速USB数据采集等，这样构成的具有USB通信功能的单片机系统比很多使用专用芯片的系统简单、成本低。

目前，AVRUSB已经成功应用到了很多产品上，比较有名的有：USBasp(USB接口的AVR编程器)、AVRCDC(USB 转 RS232 串口)、USB Bootlader（USB 接口的 Bootloader 软件）等。在http://www.obdev.at/products/avrusb/projects.html中，还专门列举了很多使用AVRUSB的开源项目，这些项目提供了完整的单片机程序和计算机程序的代码和原理图。我们在开发自己的AVRUSB应用时，可以参考这些资源，在这些开源项目的基础上进行修改，快速开发出适合于自己需求的应用来。本文作者也成功的将AVRUSB用到公司的一个项目中，很好的实现了一个USB接口的开关矩阵控制。

1.5 AVRUSB的限制

因为AVRUSB使用普通IO口模拟来USB通信的过程，由软件实现了硬件完成的功能。而USB通信的速率是比较高的。因此，在进行USB通信时单片机的CPU占用率比较高的（大于 90%）。

另外，因为受到单片机的处理能力限制，所以通信的数据处理能力不是很强，最大数据处理速度约为 20k/s，因此AVRUSB不适合用于大数据量通信的应用场合。

2 硬件结构

构成一个AVRUSB系统的硬件结构非常简单，只需要一个普通的AVR单片机（大部分型号都可以），再加上少量的外部元件（晶体、几个电阻以及可选的稳压二极管等），就组成了一个基本的AVRUSB系统，如图 1。

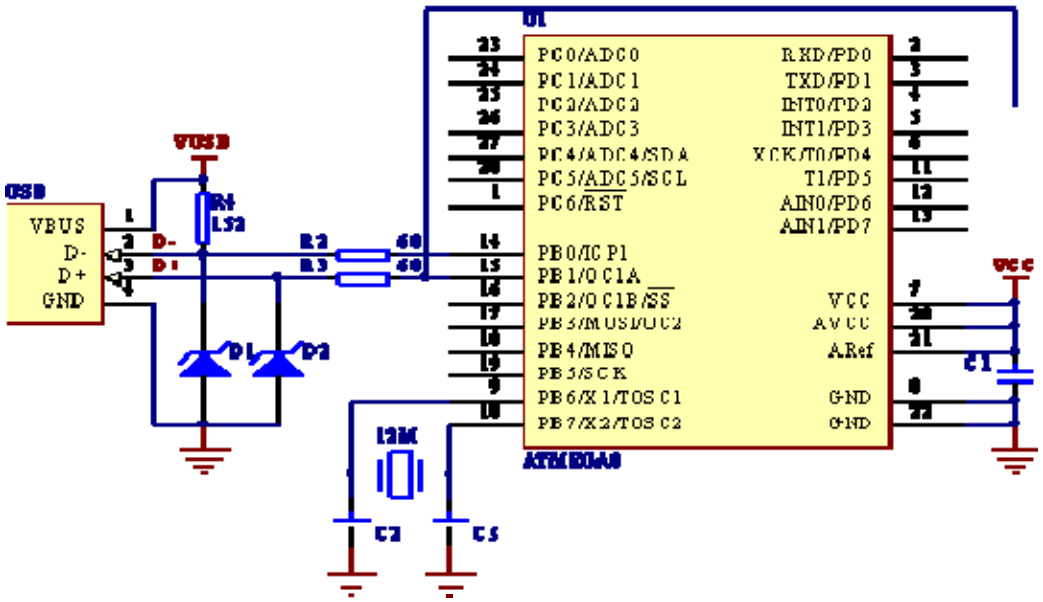

图 1. AVRUSB的基本硬件接口

图中的单片机以ATmega8 为例。数据线D-上的上拉电阻R4 用来通知计算机这是一个低速USB设备（这是在USB规范中定义的，更多内容请参考 AVR309 以及USB的官方文档）。12MHz晶体和两个 22p的电容C2和C3 组成单片机运行所必须的时钟。D+ 和D- 数据线可使用单片机的任意IO端口，但是必须使用相同的IO端口。在这里D+ 连接到PB1，D- 连接到PB0。此外数据线D+还需要连接到INT0 上，这是为了在不同的AVR单片机中使用AVRUSB时有更好的适应性和兼容性，无需修改底层核心部分程序的代码。如果D-连接到端口D上（就是和INT0 同一端口中），同时D+只连接到INT0，还可以节省出一个端口来。

电阻R2、R3 起到限流和保护作用，防止在意外情况下损坏计算机的USB端口或单片机的端口。单片机所需的电源Vcc可由USB的 5V输出电源直接提供，或者由USB的 5V电源转换得到（如LDO、稳压二极管等），或者通过电池等其他外部电源来供电。

D+ 和D-上的 3.6V稳压二极管D1 和D2 起到限制数据线上的电平的作用。因为在USB规范中规定数据线D+和D-上的电平范围是 3.0V至 3.6V，而 AVR单片机的输出电平是Vcc。如果单片机的Vcc是 5V，在没有D1和D2 的情况下将造成电平不匹配，会造成在很多计算机中无法正确识别出USB设备。如果用户系统的Vcc在 3.0V至 3.6V之间，就可以省略这两个稳压二极管。从这里也可以看出用户系统的Vcc必须高于 3V。

上面硬件就组成了一个最小的AVRUSB系统，它能够和计算机进行USB通信。在上面最小系统的基础上，如果添加一个红外传感器，就可以接收发送红外信号，就是一个USB红外控制器；如果添加一个MAX202，就是一个带缓冲的USB <-> RS232 串口转换器；如果加入ADC转换功能，就是USB的数据采集器；如果加入ADC和电源控制，就能够实现一个简单实用的USB充电器……各种功能的AVRUSB系统都是在这个最小系统的基础上，添加不同功能的外围模块或接口来实现的。

3 单片机程序的开发

要使用AVRUSB，就需要在单片机中开发合适的软件，实现特定的功能。下面将具体介绍开发需要使用的工具软件和开发的步骤。

3.1 开发环境

AVRUSB可使用AVR GCC编译器或IAR C编译器。因为AVRUSB特别为AVR GCC编译器做了优化，并且

AVR GCC还是免费软件，使用非常广泛，是目前AVR单片机主要的开发工具软件之一，所以我们下面以AVR GCC为例来介绍。

在 Windows操作系统下开发时，使用WinAVR中带有的AVR GCC编译器；代码编辑、仿真和调试使用了Atmel公司的AVR Studio，这个IDE的好处是不需要用手工修改makefile配置文件，减少了初学者的使用难度。如果希望使用其他IDE或编辑软件作为开发工具也可以，下面的使用步骤也是类似的。如果是在Linux操作系统下开发，可以使用Linux版本的AVR GCC，以及其他工具软件进行编辑和调试。

3.2 建立项目文件

要将AVRUSB加入到自己的程序中，首先需要在AVR Studio中建立一个新的项目（Project），然后将AVRUSB所需要的文件复制到项目文件的文件夹中（当然也可以不用复制文件，只添加已经存在的AVRUSB的文件路径到项目中也可以，但是将项目的所有文件放在一个文件夹下更容易对整个项目进行维护）。一般来说，凡是使用AVRUSB的项目，在项目文件夹下都会单独存放一个USBDRV的文件夹，里面存放着所有与AVRUSB相关的文件。AVRUSB包含了多个文件，但是我们只需要添加以下几个文件到项目中：

usbconfig.h	用户配置文件
iarcompat.h	为兼容IAR编译器而定义的宏
usbdrv.h	usb驱动接口文件的头文件
usbdrv.c	usb驱动接口文件
usbdrvasm.asm	为兼容IAR编译器而使用的底层接口函数文件的别名
usbdrvasm.S	汇编语言编写的底层接口函数
oddebug.h	调试用函数的头文件（不使用调试功能时可以不添加）
oddebug.c	包含调试用的函数（不使用调试功能时可以不添加）

注意到上面的文件除了usbconfig.h外，都在USBDRV文件夹中。在USBDRV文件夹中一般都有一个usbconfig- prototype.h文件，这个文件是用户配置文件usbconfig.h的原始模板，我们需要将这个文件复制到项目文件夹中并将它改名为 usbconfig.h。复制后，还需要再添加USBDRV文件夹的路径到项目的包含路径中，这样在编译时才可以正确找到上面的文件，如图二（因为 USBDRV文件夹在项目文件夹下，所以显示出的是相对路径）。如果不使用AVR Studio，也可以使用其他的IDE软件或编辑软件，但是可能会需要用户自行手工修改项目配置文件makefile，具体做法这里就不做介绍了。

图 2. 添加USBDRV路径到项目中

3.3 参数配置

在编译项目之前，除了需要对项目本身的参数进行配置外（如AVR单片机的型号、系统时钟频率、代码优化等级等），还需要对AVRUSB的参数进行配置，这样才能产生出正确的代码。AVRUSB中包含的参数虽然看起来很多，其实配置起来很容易。在用户配置文件usbconfig.h中存放了所有与USB相关的配置参数，配置参数都是以宏定义的方式给出，配置参数的过程就是修改相关的宏定义。只要修改这个文件中的参数就可以实现不同的功能，其它的文件不用修改。

为了获得尽可能高的效率，AVRUSB中使用了大量的宏定义和条件编译，用以获得最小的代码大小和最快的运行速度。配置文件usbconfig.h中的参数非常多，表 1 列出了主要需要修改的参数。一般的情况下只要修改这几个参数就可以了，其它的参数可以根据用户的实际需求做适当的修改。

表 1：usbconfig.h中的主要参数

USB_CFG_IOPORTNAME

定义USB数据线使用的端口。只要是通用的IO都可以，没有特殊的要求。

USB_CFG_DMINUS_BIT

USB数据线D-使用的引脚。

USB_CFG_DPLUS_BIT

USB数据线D+使用的引脚。因为D+要求同时连接到INT0 上，所以一般情况下需要使用 3 个IO口。如果D+使用的引脚就是INT0，那么可以少使用一个IO端口。

USB_CFG_VENDOR_ID

设备生产商的ID号

USB_CFG_DEVICE_ID

设备的产品ID号。这两个参数就是Windows识别USB设备的主要参数。需要注意的是，这两个参数都是低字节在前，高字节在后。

USB_CFG_DEVICE_VERSION

设备的版本号次版本号在前，主版本号在后。在Windows的设备管理中可以看到这个版本号

USB_CFG_VENDOR_NAME

设备生产商的名称，它在Windows的设备管理中可以看到。这里一般写入的是网址。

USB_CFG_VENDOR_NAME_LEN

设备生产商名称的长度。

USB_CFG_DEVICE_NAME

设备的名称，它在Windows的设备管理中可以看到。设备名称和生产商的名称都是以字符的方式定义的，它们目前不支持中文。

USB_CFG_DEVICE_NAME_LEN

设备名称的长度。

3.4 主要接口函数

配置好参数，就可以开始编写用户程序了。用户程序和计算机之间的USB通信是通过AVRUSB提供的接口函数完成的，接口函数有 6 个，下面将分别介绍。

3.4.1 初始化函数

在使用AVRUSB前，需要进行必要的初始化，这通过调用初始化函数usbInit()完成。一般是在程序其他部分初始化完成后再调用函数usbInit()，最后再调用sei()函数允许中断。

```
void main()
{
…
    usbInit();
    sei();
    …
    while(1)        //主循环
    {
      …
    }
}
```

3.4.2 USB事件处理函数

在用户程序的主循环中需要定期调用USB事件处理函数usbPoll()。USB事件处理函数usbPoll()在没有USB事件需要处理时将直接返回，否则将调用内部函数进行相应的事件处理，最后再将数据通过传递到后面介绍的用户接口函数中。通常的用法是：

```
while(1)        //主循环
{
   usbPoll(); //USB事件处理
  ……        //其他用户事件
}
```

一次USB通信的超时时间是 50ms。所以在编程时注意其他事件不要占用太长的时间，使得usbPoll()函数不能及时执行。

3.4.3 用户事件接口函数

在用户程序中需要编写USB用户事件接口函数，完成USB通信。AVRUSB将用户接口简化为以下 3 个函数，这三个函数需要用户进行编程处理，它们将完成USB通信的数据处理。

usbFunctionWrite　主机向单片机写入数据

usbFnctionRead　主机从单片机中读取数据

usbFunctionSetup　一般功能设置

函数usbFunctionSetup负责传递USB请求，参数存放在一个 8 字节的数组中(uchar data[8])，其的含义是：

uchar requestType; //[0]　请求类型

uchar request; //[1]　请求的内容

unsigned value;　//[2], [3] 参数

unsigned index;　//[4], [5] 序号

unsigned length;　//[6], [7] 长度

在一般情况下，除了data[0]和data[1]用于存放USB事件的请求参数外，data[2]至data[7]都可以传递用户参数。这样在数据量非常小的时候，通过usbFunctionSetup就可以完成全部参数传递了，不需要使用函数usbFnctionRead 和 usbFunctionWrite 。这时可以在 usbconfig.h 中将宏USB_CFG_IMPLEMENT_FN_READ和 USB_CFG_IMPLEMENT_FN_WRITE设置为 0，禁止使用函数usbFnctionRead和usbFunctionWrite，这样可以节省出不少代码空间。

在传递的数据比较多时，就需要使用usbFunctionWrite和usbFnctionRead函数了。这两个函数具有相同的参数(uchar *data, uchar len)，len表示参数的数量，*data指向数据缓冲区。在编程时，一般是先在usbFunctionSetup函数中根据计算机发送过来的请求类型和参数来设置一个全局标志，然后在usbFunctionWrite和usbFnctionRead函数中根据相应全局标志进行不同的功能处理。

3.4.4 数据校验

为了保证USB数据通信的可靠，避免传输中出现误码，还应当对数据进行校验。USB通信使用了CRC校验来保证通信的可靠性，数据校验的方法是调用usbCrc16()函数，函数的使用方法是：

crc = usbCrc16((uchar *)(unsigned)(usbAppBuf + 1), usbRxLen - 3);

USB数据缓冲区中存放的校验结果在

((uchar *)(unsigned)(usbAppBuf + 1))[usbRxLen - 3] 和

((uchar *)(unsigned)(usbAppBuf + 1))[usbRxLen - 2] 两个字节中。

将计算结果和缓冲区的数据进行比较就可以知道通信缓冲区中的数据是否正确了。需要注意的是usbCrc16函数并不是在用户程序中调用的，而是在 usbdrv.c文件中的usbPoll函数中。在usbPoll函数的开始部分有一段注释，解释了可以在此调用usbCrc16 函数。

在很多 AVRUSB项目中（如USBasp）并没有处理CRC校验，这是因为在USB通信中对时间要求比较严格，AVRUSB在底层处理时来不及进行数据校验，所以底层函数在接收到数据后就直接将ACK信号发送回去了。这样即使在数据校验中发现了错误，也不能由底层函数报告给计算机，只能在用户程序中向计算机发出错误报告。在一般要求不高的应用中可以不进行数据校验，以简化程序结构。但是在要求比较高的控制系统中应当进行CRC校验，使得整个系统具有更高的可靠性。当然，用户也可以不使用比较耗时的CRC校验而用其他方法建立起自己的数据错误处理机制。

4 计算机程序编程

计算机程序的编程主要包括了驱动程序的设计和用户程序的开发，每个部分又包含了很多内容。

4.1 PID和VID

每种USB设备都有一个PID和VID。VID是生产商的代号，PID是产品的代号，每个代号都是一个双字节的整数。PID和VID不能随意设置，它是由 USB标准协会进行分配的，就像IP地址的分配一样。一个USB的PID/VID许可需要花费1500美元，在很多情况下，特别对小公司是一个很大的费用。针对这种情况，AVRUSB

提供了 3 个免费的PID/VID对，分别适用于HID类、CDC类和通用类设备，使用AVRUSB的用户可以免费使用它们，这样对于大多数应用来说就不用再自己去申请PID/VID了。

表二：免费的PID/VID对

VID
PID
控制类
0x16C0
0x05DC
CDC类
0x16C0
0x05DF
HID类
0x16C0
0x05E1

PID和VID在驱动程序和用户程序中都将用到，它是windows识别USB设备的关键参数，用户程序也需要通过PID和VID来查找相应的USB设备。

4.2 USB设备类型

AVRUSB 有多个版本，早期的版本只支持控制类的USB设备和最多两个节点，而最新的版本还可以支持CDC类（串口类）和HID类（人体工学设备）以及最多 4 个节点。CDC类和HID类都无需自己编写Windows的驱动程序（Windows自带）。AVRUSB的作者不推荐使用CDC类的方式（据说是因为在有些计算机上兼容性不好），而推荐使用HID类，但是使用HID类的应用目前还不是很多，支持的程序也比较少。本文将只介绍应用最广泛的控制类设备的编程方法。

4.3 驱动程序

对于每种 USB 设备，都需要有相应的驱动程序。只有安装了驱动程序后，USB 设备才能被操作系统所识别，设备才能正常工作。每个驱动程序都包含了底层驱动程序、设备描述文件等几部分。

虽然现在已经有很多优秀的开发底层驱动程序的工具，但是对于一般用户来说开发驱动程序还是一件比较复杂的事情。AVRUSB 的底层驱动程序使用了 LibUSB，这是一个很有名的开源 USB 驱动程序，支持多种操作系统平台，在 Windows 操作系统下对应的版本是 LibUSB-Win32。也就是说，计算机上的 USB 底层驱动程序不需要用户自己再编写了，直接使用 LibUSB 就行了，用户只需要编写设备描述文件。设备描述文件是一个文本文件，用于说明设备的参数，主要就是设备的 PID、VID、设备名称、使用的底层驱动程序名称等。

4.3.1 定制自己的 USB 设备驱动程序

要想让 LibUsb-Win32 成为自己开发的 USB 设备的驱动程序，只需对 LibUsb-Win32 提供的 libusb.inf 设备信息文件（INF）的内容按照其中的注释给出的说明进行适当的修改即可。其中最重要的修改是对 VID/PID 的修改，将文件中的 VID/PID 值替换为自己开发的 USB 设备的

VID/PID 值（也就是上面表二给出的值）。其它可以修改的内容还有设备制造商名称，设备描述字符串等等。

以 LibUSB-Win32 的 0.1.10.1 版本为例，一个使用 LibUSB 的 USB 设备驱动程序包至少要包含以下的 3 个文件：libusb0.sys、libusb0.dll 和 libusb.inf。在 LibUsb-Win32 的 0.1.10.1 版本中，还提供了一个用于自动生成 INF 文件的向导工具。该工具位于 libusb-win32-device-bin-0.1.10.1\bin 目录下，文件名为 inf-wizard.exe。通过使用该向导工具，可以方便地生成与自己开发的 USB 设备相关的 INF 文件。需要注意的是，该向导工具不仅生成了一个 INF 文件，同时也生成了一个安全目录文件（*.cat），该文件包含的由 Microsoft 提供的对将要安装到 Windows 系统上的驱动程序文件的数字签名。如果没有这样的安全目录文件，需要从生成的 INF 文件中删除掉“Version“部分中的“CatalogFile“内容。

4.3.2 预安装 USB 设备的驱动程序

通过“找到新硬件向导“的方式来安装设备的驱动程序作为 Windows 系统的一种标准方式已经使用很多年了，每当我们加入一个新的设备到计算机上时，Windows 会提示我们发现一个新的设备，然后进一步提示我们去查找并安装相应的驱动程序。这个步骤对于终端用户来说还是不太方便，对他们而言，希望只要将新的硬件装（插）到 Windows 系统上就可以使用了，而无需再做其它额外的工作。

Microsoft 已经认识到了这个问题，并已制作了一套能够在 Win2000 及更高版本的操作系统上使用的驱动程序安装工具——Driver Install Frameworks (DIFx)。这些工具能够进行驱动程序的预安装，即新硬件在系统上可用之前安装驱动程序。这允许当一个匹配的设备首次被连接到电脑上时，相应的驱动程序能够被自动装载。

Driver Package Installer（DPInst）是 DIFx 工具集中的工具之一，该工具提供了进行驱动程序预安装的最简单也是最常用的方法。DPInst 由 DPInst.exe 和 DPInst.xml 这两个文件组成，其中 DPInst.xml 是一个可以对安装过程进行配置的文件。将驱动程序包放在与 DPInst.exe 和 DPInst.xml 一样的目录下（或者将驱动程序包放在 DPInst.exe 所在目录的子目录下），然后运行 DPInst.exe，就可以将驱动程序包进行安装。以下是 DPInst.xml 的一个例子：

```
<?xml version="1.0"?>
<dpInst>
    <search>
      <subDirectory>DriverPackage</subDirectory>
    </search>
    <language code="0x804">
        <dpinstTitle>XXX 控制器 USB 设备驱动程序安装向导</dpinstTitle>
        <welcomeTitle>
欢迎使用 XXX 控制器 USB 设备驱动程序安装向导！
</welcomeTitle>
        <welcomeIntro>
此向导将帮助您安装 XXX 控制器 USB 设备驱动程序。没有该驱动程序，XXX 控制器 USB 设备将无法运行。
```

```
</welcomeIntro>
        <eulaHeaderTitle>最终用户许可协议</eulaHeaderTitle>
        <eulaYesButton>我同意此协议(&amp;amp;A)</eulaYesButton>
        <eulaNoButton>我不同意此协议(&amp;amp;D)</eulaNoButton>
        <installHeaderTitle>
正在为您的设备安装该驱动程序，请稍候...
</installHeaderTitle>
        <finishTitle>祝贺您！该驱动程序已成功安装到了您的电脑上。</finishTitle>
        <finishText>您现在可以使用 XXX 控制器 USB 设备了。</finishText>
        <eula type="txt" path="CustomData\eula.txt" />
    </language>
    <legacyMode/>
```

4.4　用户程序的开发

完成了驱动程序的开发，我们还需要完成更复杂的用户程序开发。开发用户应用程序同样需要使用到 LibUSB，开发使用的编程语言可以使用任何常用的编程语言，在 Linux 下一般是 GCC、Kylix 等；在 Windows 下选择就比较多一些，GCC、VC、VB、Delphi/CBC 等都可以。

因为我们平时使用 Windows 操作系统比较多，所以下面介绍在 Windows 操作系统下的 LibUSB 使用方法和编程，LibUSB 在 Win32 平台下对应的版本是 LibUSB-Win32。

4.4.1　LibUSB-Win32 简介

LibUSB-Win32 是一个用于 Windows 操作系统（Win98SE、WinME、Win2k 和 WinXP）上的通用 USB 设备驱动程序。该驱动程序允许使用者在不写任何一行核心驱动程序代码的情况下，可以访问 Windows 系统上的任意一个 USB 设备。该驱动程序具有以下特点：

- 能够与任意一个已安装的 USB 设备进行通信
- 可被用作自己开发的 USB 设备的驱动程序
- 支持批量和中断传输
- 支持 USB 规范中定义的所有标准设备请求
- 支持 USB 设备制造商的自定义请求

LibUsb-Win32 是由http://libusb-win32.sourceforge.net发布的，遵守GNU Lesser General Public License（LGPL）和GNU General Public License（GPL）许可协议。这些协议明确规定：允许LibUsb-Win32 用于商业软件，而不只是开源软件。

4.4.2　使用 LibUSB-Win32

LibUSB-Win32 为 C/C++程序员提供了用于开发的头文件和 Lib 文件，其中 Lib 文件还提供了 BCC、GCC 和 MSVC 这三个版本。C/C++程序员在自己的程序中要使用 LibUSB-Win32 时，只需包含提供的头文件，并链接合适的 Lib 文件即可。

对于Delphi程序员来说，LibUsb-Win32 没有提供现成的Import Unit（导入单元文件），不过Internet上已经有程序员提供了对LibUSB-Win32 的Delphi Pascal转换文件。Delphi程序员可以从http://www.xs4all.nl/~ynlmns/ 上下载Delphi版的LibUsb-Win32 文件。值得注意的是，该版LibUSB-Win32 不仅提供了Import Unit（LibUSB.pas），还提供了一个DLL文件

（USBLibExport.dll）。在对LibUSB-Win32 编程时，这个DLL必须作为程序的一部分，通过调用这个DLL中的函数来访问LibUSB-Win32，这样的效率比较低。本文的作者在LibUSB.pas文件的基础上进行了分析和研究，对原LibUSB.pas做出了如下一些修改：

1) 在 interface 部分中，将所有导出函数声明中的调用约定“stdcall”改为了“cdecl”；
2) 在 implementation 部分中，将所有导出函数中 external 指示符后面所跟的字符串“USBLibExport.dll”改为了“libusb0.dll”。
3) 进一步封装了 LibUSB 中的函数，使之和 AVRUSB 单片机底层函数相对应，简化了 LibUSB 函数的使用，方便了用户编程调用。

这样可以不再需要使用USBLibExport.dll而直接访问LibUSB-Win32。Delphi程序员只需将修改后的LibUSB.pas包含到自己的程序中，就可让自己的程序直接使用LibUsb-Win32 了。这个修改后的LibUSB.pas文件可以在本文作者的博客上下载：
http://shaoziyang.bloger.com.cn/user2/88141/archives/2006/323777.shtml

4.4.3 常用的函数

LibUsb-Win32 为开发人员提供了一套丰富的 API 函数，包括核心函数、设备操作、控制传输、批量传输和中断传输等几类。下面就几个常用的函数进行一下简单介绍：

1) void usb_init(void);
该函数用于初始化 LibUSB-Win32，它必须第一个被调用。

2) int usb_find_busses(void);
查找系统上所有的 USB 总线。

3) int usb_find_devices(void);
查找每一根 USB 总线上的所有 USB 设备，该函数应该在 usb_find_busses 函数之后被调用。

4) struct usb_bus *usb_get_busses(void);
获取已找到的 USB 总线序列，返回值是一个指向 USB 总线序列首地址的指针。

5) usb_dev_handle *usb_open(struct *usb_device dev);
打开一个指定的 USB 设备。在对某个 USB 设备执行任何操作之前，首先要打开该 USB 设备，就像打开一个文件、打开一个句柄一样。函数的返回值是一个设备句柄，该句柄在后面与设备进行通信时会被用到。

6) int usb_close(usb_dev_handle *dev);
关闭一个指定的 USB 设备。在使用完某个 USB 设备后，应该关闭该 USB 设备，这和在使用完一个句柄后需要关闭句柄是一样的。

7) int usb_control_msg(usb_dev_handle *dev, int requesttype, int request, int value, int index, char *bytes, int size, int timeout);
向 USB 设备发送一个请求。函数各参数的含义如下：
dev：USB 设备句柄，该参数指定了向哪个 USB 设备发送请求；

requesttype：请求的类型，说明了是读请求，还是写请求。可以是标准请求、类请求或自定义请求；

request：表示具体请求的值；

value：与请求相关的参数；

index：与请求相关的参数；

bytes：表示在数据阶段时传输的数据；

size：表示在数据阶段时传输的数据量，以字节为单位；

timeout：请求的超时设置，最大值为 5000，以毫秒为单位。

通过使用以上七个函数，就可以与 USB 设备进行简单通信了，通信的主要流程可分为以下四步：

1) 调用 usb_init 函数，进行初始化。

2) 打开要进行通信的 USB 设备的句柄。首先依次调用 usb_find_busses、usb_find_devices 和 usb_get_busses 这三个函数，获得已找到的 USB 总线序列；然后通过链表遍历所有的 USB 设备，根据已知的要打开 USB 设备的 ID（VID/PID），找到相应的 USB 设备；最后调用 usb_open 函数打开该 USB 设备（在这里假设总线上没有相同 VID 和 PID 的 USB 设备。如果总线上存在着相同 VID 和 PID 的设备，还需要进行其他条件判断，比如设备名称，以保证是打开的是期望的 USB 设备）。

3) 与 USB 设备进行通信。使用 usb_control_msg 函数，向 USB 设备读取数据或写入数据。

4) 关闭 USB 设备。完成所有操作后，调用 usb_close 函数关闭已经打开的 USB 设备。

5 其他要注意的问题

在使用 AVRUSB 时，需要注意以下问题：

1. 晶体一定要使用 12MHz 的，误差范围在±0.2%之内。如果晶体频率误差太大，对 USB 通信会有一定的影响，容易造成通信失败。

2. 一个型号的 AVR 单片机往往分为 8M／10M／16M 等多个频率等级，不同的电压也有不同的频率限制（参考数据手册上相应的说明）。很多 8M／10M 频率等级的单片机在 12M 频率下也可以使用（超频），在一般情况下没有太大的问题。不过在可能情况下应当尽量使用 16M 频率等级的单片机，以提高系统的稳定性。

3. 使用 AVRUSB 的 CDC 类做 USB 转串口时，因为只能使用 12M 的晶体，所以有很多波特率的误差比较大，超过了串口通信允许的±2%误差范围。一般最好使用 2400、9600、19200 等误差较小的波特率。

4. AVRUSB 定义的缓冲区最大是 254。通信时，一次往缓冲区写入数据不要超过缓冲区的大小，否则很容易在通信时造成数据丢失。

5. AVRUSB 中使用了大量的宏定义和条件编译进行系统配置，这样造成了程序的可读性比较差，特别是刚开始时很难弄明白。而且 AVRUSB 没有专门的说明文档，只有通过研究源代码以及代码中的注释来学习。但是这也是一种编程技巧和编程风格，可以有效的提高程序的代码效率。

参考资料：

1. ATMEL公司的应用笔记《AVR309 Software Universal Serial Bus (USB)》：http://www.atmel.com/
2. AVR309 的中文翻译《AVR309 软件USB》:http://shaoziyang.bloger.com.cn/
3. AVRUSB的官方开源项目PowerSwitch：http://www.obdev.at/products/avrusb/
4. 使用AVRUSB的编程器USBasp：http://www.fischl.de/usbasp/
5. LibUSB：http://libusb.sourceforge.net
6. LibUSB-Win32：http://libusb-win32.sourceforge.net