Universal USB Device

Michael Odenwald, Michael Keller and Paul Goossens

Designing your own USB device can be an enjoyable task for electronics hobbyists.
The main stumbling block is often the driver for the device.
Writing this piece of software can be a bit too difficult for many people.
A universal USB device driver, which is also open source, presents the solution!

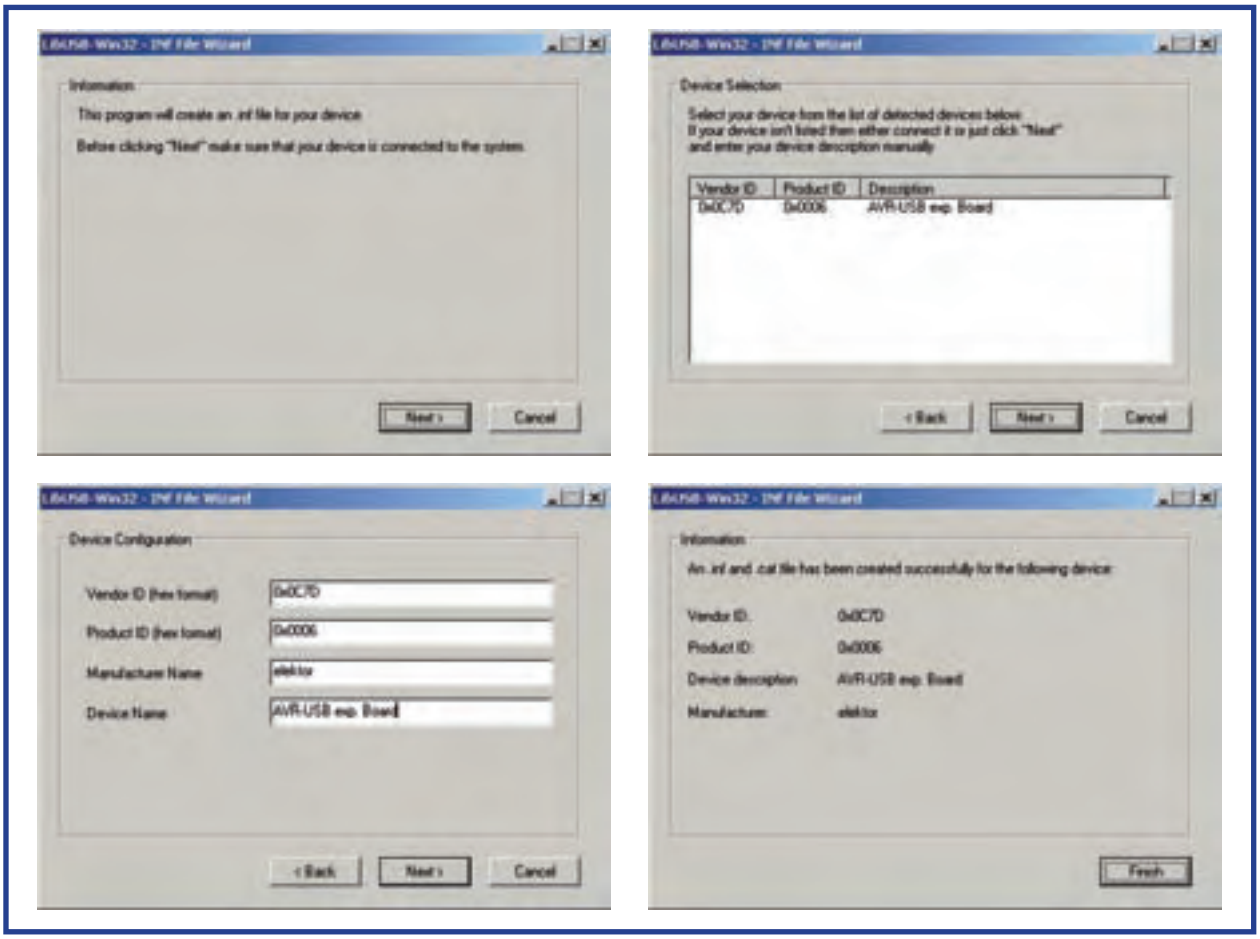


Figure 1.
These are the steps you'll come across when creating the INF file for the universal USB driver.

When designing USB devices you also need to write an associated device driver. For many designers this is a very tricky task and many won't even attempt such a project because of this. This is now all set to change! With the help of our AVR USB board we'll show you how we tackle this driver problem, using an existing universal open source device driver program.

Connections

Everything went smoothly as far as the hardware for our AVR USB board was concerned, until we connected it to the PC. Windows (XP) then politely asks us for a device driver, as expected. And this doesn't exist (yet) for our hardware.
To help us out we made use of 'libusb-win32', a universal open source USB device driver [1]. This comes as a DLL and lets Windows applications communicate directly with a USB device. With this we can deal with the (device dependent) data stream in a Windows application and there is no need for a specially written device driver.
Unlike the way a 'normal' device driver functions, libusb-win32 lets us use an API (Application Programming Interface) instead of having to communicate via a so-called IOCTL (Input/Output ConTroL).

Driver Say goodbye to connection problems

Let's get busy!

First we have to download libusb-win32 and extract all files from this ZIP file in their own folder. Before we can use this driver we first have to create an INF file. This file tells Windows which driver belongs to which device.
The creation of this INF file is made very easy by the inclusion of a program in libusb-win32 ('inf-wizard.exe'). The INF file is created as follows: Power up the AVR USB board and connect it to the PC. Windows will now ask for the driver. Click on 'Cancel'. Start the program 'inf-wizard.exe'. In the first window click on 'Next', after which the second window appears. This contains a list of all the connected USB devices. Select our AVR USB board, with a VendorID of 0x0C7D and a ProductID of 0x0006, and click on 'Next'. In the third window we can give the device a different name, if required. This isn't necessary, so we click on 'Next' again. The program now reports that it has generated an INF and a CAT file. Click on 'Finish' to close the program. (**See Figure 1.**)

Initial test

We now have to disconnect the AVR board from the PC and then reconnect it again. Windows will ask for the driver program once more. This time, select the option 'Specify a location' and browse to the INF file that we created with 'inf-wizard.exe'.
If everything goes to plan, Windows will install the device driver. From the Device Manager in Windows you should now be able to see the AVR USB board.

Applications

We can now use the AVR USB board in our applications. On the Elektor Electronics website are a few downloads for this project, which contain several applications (including the source code) for controlling the AVR USB board. These applications have been written in C, . NET and a few other languages. You can use the source code as the basis for your own applications.
The testing of all functions of the AVR USB board is very easy with the program 'AVR-USB-Windows1.exe'. With this you can test all I/O capabilities of the hardware.

A closer look at the source code

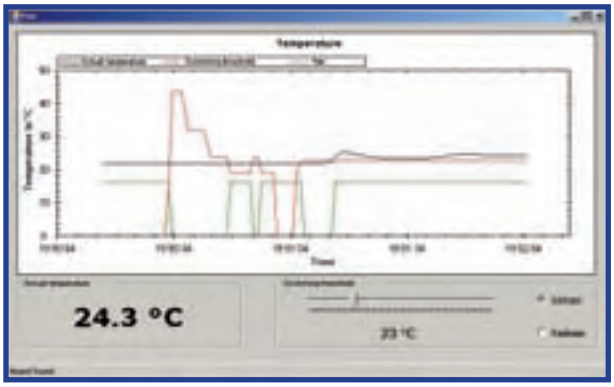
To give you an idea how we can control our USB device from within an application, we'll show you a few parts of the source code. The saying 'a picture is worth a thousand words' also applies to source code!
In **Listing 1** you can see how the application searches for a specific USB device. If it is found, a connection with this device is initiated.
In a for-next loop all of the available USB busses of the computer are interrogated. The VendorID and ProductID of every USB device connected to each bus are requested. When both the VendorID and ProductID have the required values the program tries to 'open' this device. The 'handle' for this device is stored in the global variable 'usb_dev_handle'. The last step is to configure the device according to the USB standard and claim exclusive rights to the use of this device.
When no unexpected errors occur the function will return the value '1' to indicate that everything went well. If anything went wrong the function will return the value '0'.

Listing 1

```
Globale variabele:
usb_dev_handle *usbIODevice; /* The usb device handle */

Funktie:

int searchUSBDevice(int vendorID, int productID)
{
  struct usb_bus *busses;
  struct usb_bus *bus;
  struct usb_device *dev;

  if (usbIODevice != NULL)
  {
    usb_close(usbIODevice);
  }

  usbIODevice=NULL;

  /* Find the device for the given vendorID and
productID*/
  usb_init();
  usb_find_busses();
  usb_find_devices();
  busses = usb_get_busses();

  for (bus=busses; bus; bus=bus->next) {
    for (dev=bus->devices; dev; dev=dev->next) {
      if (dev->descriptor.idVendor == vendorID &&
      dev->descriptor.idProduct == productID) {
        usbIODevice=usb_open(dev);
        if(usb_set_configuration(usbIODevice, 1)) return 0;
        if(usb_claim_interface(usbIODevice, 0)) return 0;
        return 1;
      }
    }
  }
  return 0;
}
```

Hardware control

Controlling the hardware is also very simple. In **Listing 2** you can see how the status of the digital inputs is read. This assumes that the function 'searchUSBDevice' (from listing 1) has previously been called to assign a valid



handle to 'usb_dev_handle'.
With the help of 'sendUSBVendorCmdIn' we send the command 'AVR_USB_READ_DIGITAL'. This command can also be found in the source code of the firmware. With this we also specify from which port we want to read (the second parameter). The result is put into the array 'iodata'. The application can then use these returned values to determine what the status of the relevant digital input is.

Experimenting

The best way to familiarise yourself with this software is to go through the examples. The examples avr-usb1 to avr-usb4 (which can be downloaded from [2]) can all be compiled using GCC [3]. (GCC stands for GNU Compiler Collection, a free ANSI-C compiler with support for K&R C, C++, Objective C, Java, and Fortran.)
The other examples have been written in C#.

With the many examples available, you should be able to write applications for the AVR USB board in other development environments under Windows as well.

And finally ...

For ideas or help when you get stuck you should take a look at the forum for the open source driver [4]. In here you'll find many enthusiastic programmers who pass on useful information, and you can also ask questions.
You can of course also use our own forum on the *Elektor Electronics* website to share your experiences with other readers. For those of you with a grasp of foreign languages it would also be useful to visit the forums of our French, German or Dutch websites. Here you'll undoubtedly find other electronics hobbyists who'll make worthwhile contributions to this project and who in turn would be interested in your experiences.

(060226-I)

Weblinks

[1] http://libusb-win32.sourceforge.net/

[2] http://www.elektor-electronics.co.uk/

[3] gcc.gnu.org

[4] http://sourceforge.net/forum/?group_id=78138

Listing 2

```
int digiportTest(void)
{
  int i;
  int result;
  unsigned char iodata[8];

  printf("digiport: AVR-USB get value from digital input Port:\n\r");

  for (i=0; i<=1000; i++)
  {
    if((result = sendUSBVendorCmdIn( AVR_USB_READ_DIGITAL, AVR_USB_DIGITAL_P1, 0, iodata, 8)) < 0)
printf(„Error sendUSBVendorCmd %d", result);
    printf(„digital in status P1 is %s\n\r", (iodata[0]) ? „close" : „open");
    if((result = sendUSBVendorCmdIn( AVR_USB_READ_DIGITAL, AVR_USB_DIGITAL_P2, 0, iodata, 8)) < 0)
printf(„Error sendUSBVendorCmd %d", result);
    printf(„digital in status P2 is %s\n\r", (iodata[0]) ? „close" : „open");
    if((result = sendUSBVendorCmdIn( AVR_USB_READ_DIGITAL, AVR_USB_DIGITAL_P3, 0, iodata, 8)) < 0)
printf(„Error sendUSBVendorCmd %d", result);
    printf(„digital in status P3 is %s\n\r", (iodata[0]) ? „close" : „open");
    if((result = sendUSBVendorCmdIn( AVR_USB_READ_DIGITAL, AVR_USB_DIGITAL_P4, 0, iodata, 8)) < 0)
printf(„Error sendUSBVendorCmd %d", result);
    printf(„digital in status P4 is %s\n\r", (iodata[0]) ? „close" : „open");
    Sleep(500);
  }
  return 0;
}
```