USB Driver Programming (1)

a DIY developer's guide

By M. Müller and C. Ehmer

There are many advantages to connecting hardware via the Universal Serial Bus (USB), but there are also many obstacles to be overcome. In this series of articles, we provide you with a map to help you find your way along the stony path of developing USB applications.

DIY hardware for the PC is usually connected to the serial RS232 interface. After several decades of use, this interface is very well documented and easy to use. Of course, it also has certain disadvantages One of these is that you can only connect a device to it before booting the PC, and another is that powering an attached circuit from the serial interface is awkward (and only a small amount of power can be drawn this way).

The USB interface is free of these disadvantages. However, most users – including experienced PC hobbyists – know little or nothing about the USB interface. The objective of this series of article is to enable you to connect homemade USB devices to a PC. The key factor here is that almost everything you need can be downloaded from the Internet for free.

There are three elements to the development of a peripheral USB circuit:

1. The hardware and a program that runs in the USB IC.

2. A program that communicates with the USB IC and displays responses on the screen.

3. A device driver (SYS file) that provides communication between the USB port and the program.

All of this will be discussed one step at a time, but first we want to see what's available on the market. Many IC manufacturers offer USB ICs, but only a few companies provide good support. An outstanding example in this regard is Cypress Semiconductor (www.cypress.com).

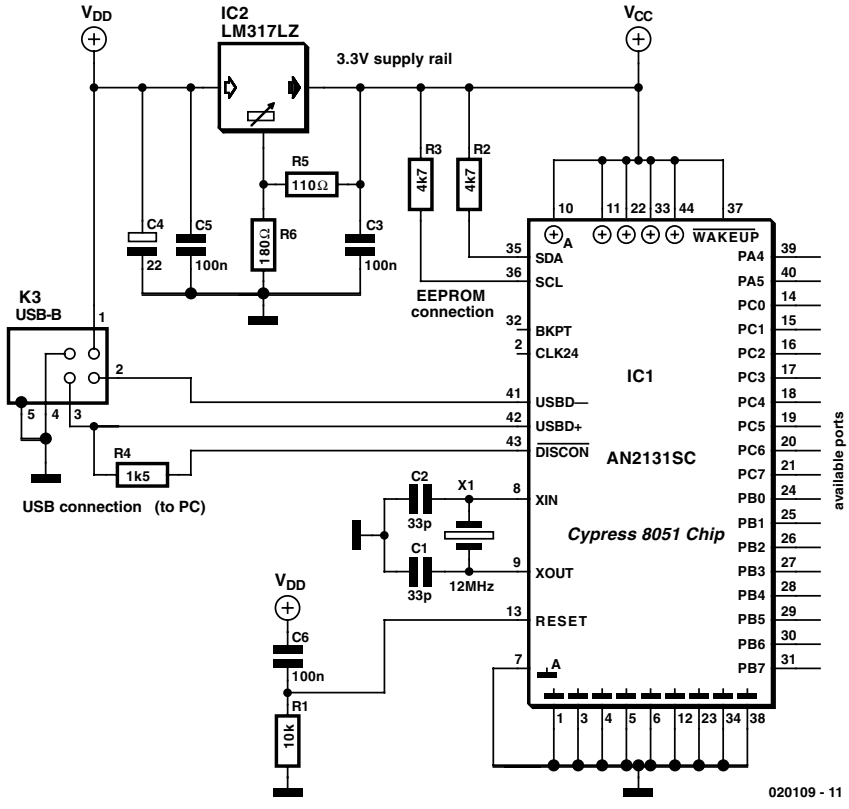


Figure 1. Minimal configuration for using the Cypress AN2131SC IC.

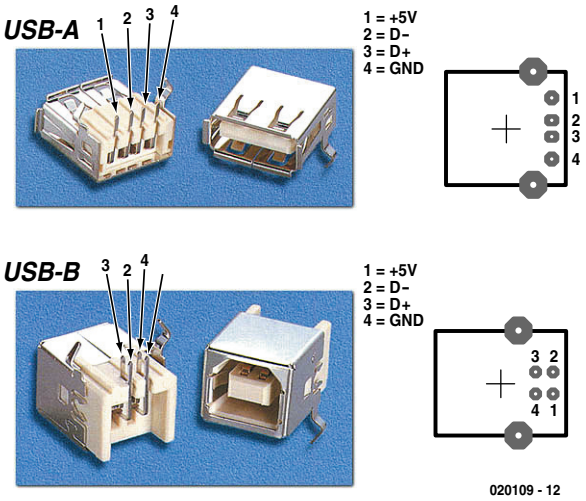


Figure 2. USB connector pin assignments.

Basic circuit for USB developers

Cypress makes our standard USB controller, the AN2131SC. The specifications of this microcontroller are provided in **Table 1**, and its data sheet can be downloaded from the Web at www.cypress.com/cfuploads/img/products/AN2131SC.pdf.

As can be seen from **Figure 1**, besides the USB microcontroller the USB connector is the second special component of the basic circuit. There are two versions, called 'Type A' and 'Type B'. The Type A version is always located on the PC or hub side, while the Type B version is always found in the peripheral device that logs in to the PC. Power can be drawn from a Type A connector, while a Type B connector can receive power. Consequently, we use exclusively Type B connectors (**Figure 2**) in our circuit. A USB cable is always wired 1:1.

A maximum of 500 mA can be drawn via the USB cable, but the Cypress IC must first report this level of current consumption, since if the PC does not have this much reserve power capacity it will automatically switch off the device. The AN2131SC IC automatically reports a value of 100 mA via its internal USB communications software. If the hardware needs more current, you must write your own software for USB communications.

The USB Controller

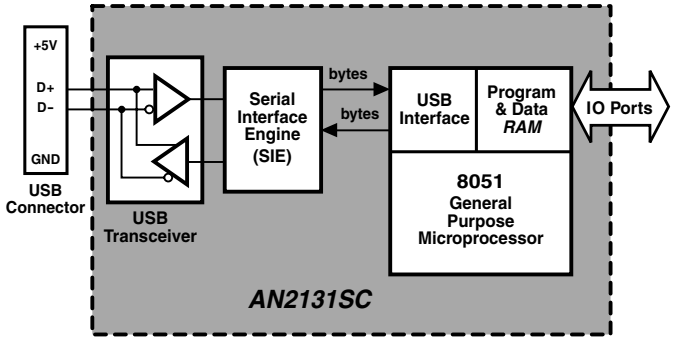
The AN2131SC single-chip microcontroller contains an 8051 processor core and a full-speed USB interface. The microcontroller has 8 KB of RAM, three counter/timers, two serial interfaces and interrupt inputs. A total of 18 freely programmable I/O pins are available.

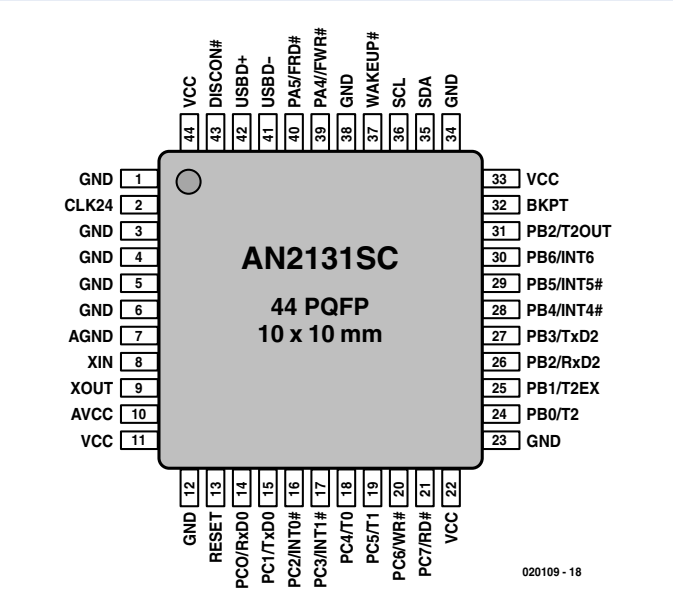
A serial EEPROM can be connected to act as program memory, or the IC can be operated using a program loaded into the RAM via USB. Practically no additional external components are necessary. The AN2131SC runs on 3.3 V, with the USB port clock frequency being derived from the 12-MHz oscillator frequency. A frequency doubler is used to generate a 24-MHz clock for the processor. Thanks to an optimised instruction set, machine instructions are executed at four times the usual rate.

Timer 0 and Timer 1 are 8-bit timers, while Timer 2 is a 16-bit timer. The signal applied to a counter/timer input port must not exceed a frequency of 2 MHz. The serial interface can be operated at 9600 baud using Timer 0 or Timer 1. Up to 38,400 baud is possible using Timer 2. In order to achieve even higher baud rates, a different frequency must be applied to the timer input.

The IC can respond individually to five separate active-low or active-high interrupts. The I^2C pins must be connected to 3.3 V via 4.7-kΩ resistors, and the Reset input must be wired to a POR network. After this, all that is necessary is to connect everything to a USB port so the PC can install a device driver.

Software updates are no longer required with the second version of the microcontroller, since with Active Software the appropriate program always runs in the USB IC. Using the built-in re-numeration feature, the USB interface can log in to the operating system with a new VID and PID.





A serial EEPROM can be connected to the SDA and SCL lines of the Cypress IC. Either the ID or an ID and a program can be stored in this EEPROM. If no program is stored in the EEPROM, the USB portion and the processor operate fully independently. All communications take place via the RAM. The processor can be placed in the Reset mode using USB driver commands. After this, a program can be loaded into the RAM starting at address 0. Furthermore, the USB driver has full access to the entire RAM region, even while the 8051 core is running. It's not necessary to write even a single line of code for this, since the USB portion of the microcontroller provides this capability on its own.

If you want to make a circuit for yourself, this basic circuit can serve as a good starting point. However, you should carefully consider how you use the ports. Although the port pins are freely programmable, they can also be used for timer inputs, interrupts and even two serial interfaces. If you connect ICs that have interrupt outputs, you should definitely use the interrupts. This makes programming significantly easier, even using C.

Tools from the Net

Before you can get started with actual programming, you will need a few programs and tools from the Internet, as follows:

1. The Cypress EZUSB Development Kit contains a C compiler from Kiel along with code templates for generating USB device drivers. In the local search engine, enter the IC designation and then go to the AN2131-DK001 Developer Kit (**Figure 3**). Here you can download the *EZ-USB Family Complete Tools***.** The current version (July 2002) is V2.52.701, which weighs in at 63,153 kB.

2. Also download the *EZ-USB Technical Reference Manual* from the same site.

3. Finally, you will need BinTerm, which is available at no charge from the author's home page (www.mmvisual.de). You can use BinTerm to download the generated C program to the Cypress IC via the USB interface and to control the entire RAM region. The necessary device drivers are included in BinTerm.

Install the development kit, taking care to avoid making any changes to the installation path. Select the 'Custom' installation option, since the Keil C compiler is only installed with this option. When installing the Keil uVision2 C compiler, you must not change any of the installation directories, since otherwise the sample programs (which use absolute path names) will not run properly.

Figure 3. Download page for the Developer's Kit.

You can consult www.keil.com for additional tips on using the Keil program. One important note is that the evaluation demo from the Keil home page does not work correctly with the Cypress IC. Other compilers, such as Rigel51, generate code that is seven times as large as that produced by the Keil compiler.

Many software developers swear by assembly-language programming. A full assembly-language program is very awkward to work with, but it is easily possible to incorporate specific assembly-language routines in the C code. Code written in assembly language can only be used for a particular type of processor, while C code can be used for all other types of processors with only minor modifications.

It's not necessary to install BinTerm; all you have to do is create a new directory (folder) named BinTerm under C:\Programs and simply copy all the files into it. Drag the BinTerm program icon onto the desktop to create a shortcut. Then click on the shortcut icon with the right-hand mouse button and select 'Properties'. Modify the link by adding the startup parameter '/MmVisual' to the 'Target' field. This will cause an additional USB-Test tab to appear in the BinTerm window (see **Figure 4**).

C code for the AN2131SC

Now you're ready for the real work. Start Keil uVision2, which is the C environment for the 8051. Create a

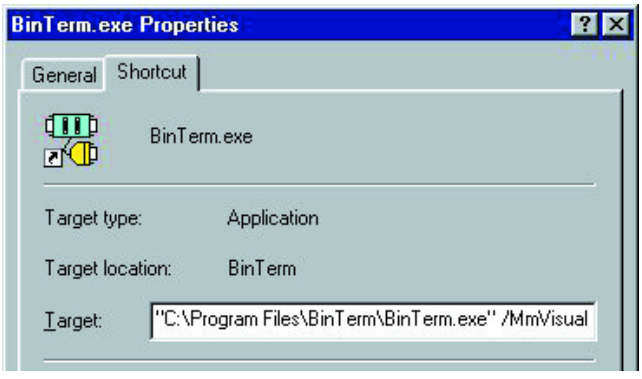


Figure 4. Entering the additional startup parameter.

new project under Project → New Project. Enter the file name 'RamTest' and confirm it with 'OK'. The next step is to select the processor (Select Device for Target): select Cypress Semiconductor → EZUSB AN21XX → OK. In the Project Window (the left-hand window), you will see that the program has created 'Target 1' with Source Group 1.

In Target 1, open the popup menu by clicking with the right-hand mouse button and then click on 'Options for Target Target 1'. A convenient entry dialog for setting all of the compiler and linker options will appear. Select the 'Output' tab and switch on the 'Create Hex File' checkbox, then close the dialogue using 'OK'.

In order to be able to write code, you must first create a new, empty file using the menu command File → New. Next, use File → Save As to store the file in the current project directory with the name 'eMain.c'. Then use File → Close to close the file.

The file is now in the project, but it still has to be appended. To do this, select the entry 'Source Group 1' in 'Project Window', press the right-hand mouse button and select 'Add Files to Group Source Group 1'. The file eMain.c will be proposed, and you can add it to the project with 'Add'. You can now close the dialogue with 'Close'.

A new entry named 'eMain.c' will now appear under Source Group 1. A double click opens the editor, which now knows that it is dealing with a C source code and enables syntax highlighting. A major disadvantage of the C language is that the compiler distinguishes between upper- and lower-case characters.

Now you can generate a small program for the Cypress IC by using the editor to enter the code lines shown in **Listing 1**.

Press F1 to generate the project. The compiler should report '0 Errors, 0 Warnings' if the code has been correctly typed in.

How the program works

The first two lines generate two variables with fixed memory assignments. The processor starts the program using the function *main*. The endless loop *while(1)* { } is executed forever (until the next Reset). The 'meaningful task' of this routine is to copy the content of memory address 0x100 to address 0x101.

When the project is generated using F1, several files are created in the project directory. The file with the extension '.hex' can be loaded into the Cypress IC using BinTerm and then started.

Start BinTerm with the option '/MmVisual', using the shortcut icon that you have previously set up on your desktop. Select the USB-Test tab. You can control the entire Cypress IC using this program function. Plug in a USB device using the Cypress IC. The device name for the hardware must be selected according to whether you use a USB data spy or a Cypress IC with no EEPROM.

At this point, BinTerm can generate a driver diskette containing the device driver for the USB device. When the new USB device is plugged in, the operating system will attempt to install a device driver that enables the BinTerm software to communicate with the Cypress USB IC.

Built-in communications between BinTerm and the USB device must be disabled by switching on the 'stop query' checkbox located under 'controlling BinTerm'. You will also see a button with three dots next to the button labelled 'send new program.'. Click on this button to open the file 'RamTxt.hex' generated by the compiler and download it to the USB IC. Finally, click on the 'start USB program' button to cause the Cypress IC to start running.

Testing the 8051 processor

The processor can be tested by checking the memory region. This is done by entering and displaying numerical values in hexadecimal form, as shown in **Figure 5**. The option 'write in the address' can be used to write a byte directly to the RAM while the 8051 processor is running. In the example shown, the byte 0x02 will be written to address 0x100. The byte is written when the 'Send' button is activated. The option 'read from address' can be used to specify the start address for reading a number of bytes (in this example, three bytes starting at address 0x100). The 'auto-read every second' option causes the displayed value to be automatically updated.

The small example program has now been loaded into the Cypress IC and started. As you can see, the IC works perfectly. The byte in address 0x100 is copied to address 0x101, just as desired. But how is it possible for BinTerm to communicate with the Cypress IC, even though you haven't written a single line of code to drive the USB interface and haven't even set the parameters for any sort of special register?

The answer is that the USB portion of the Cypress IC works completely independently, together with the Cypress device driver, just as though a second microcontroller were managing the USB interface. Furthermore, this independent portion has full access to the RAM region of the 8051 at all times. The processor can even be placed in a Reset state and restarted via register 0x7F92. All RAM addresses can be written and read at all times, and we have taken advantage of this property. Even if you don't know anything at all about C and microcontrollers, you have just programmed one!

The Keil home page has a large number of sample programs for downloading. You can also search the Internet, where you are sure to find something you can use.

Listing 1.

```
xdata unsigned char BYTE_IN _at_   0x0100; // Byte arriving (RAM address)
xdata unsigned char BYTE_OUT       _at_   0x0101; // Byte to be written (RAM address)
void main()
{ while (1)   // endless loop
  {    BYTE_OUT = BYTE_IN;  // Copy byte from BYTE_IN to BYTE_OUT
  }
}
```

Can my own program also be so simple?

The sample programs have been generated using Delphi, which is by far the most efficient programming environment for professional use and guarantees the easy use of Visual Basic, classes and C type testing. Delphi can even be downloaded for free via the Internet from www.borland.com.

Now we come to the practical part. First you have to understand the hierarchy from the USB device to your program. You have a device that you connect to the USB interface. Based on the coding of the resistors attached to the D+ and D– data leads, the PC recognises the speed of the USB device. The Windows device driver then asks the device for its Vendor ID and Product ID. If the device is already known to Windows, the manufacturer-specific device driver is loaded. Otherwise, Windows searches its driver database for a suitable device driver and offers to install it. The manufacturer-specific device driver never communicates directly with the actual USB device, but only with the Windows driver. For each USB interface, Windows can manage one hub and several devices connected to this hub. The application program can only create a link to the USB device via the manufacturer-specific device driver. Windows provides the commands 'CreateFile' and 'DeviceIoControl' in Kernel32.dll for this purpose.

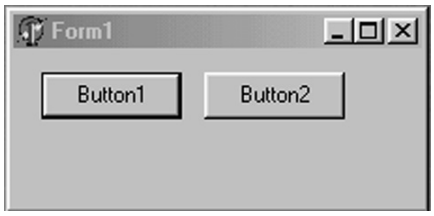


Figure 6. A form with two command buttons.

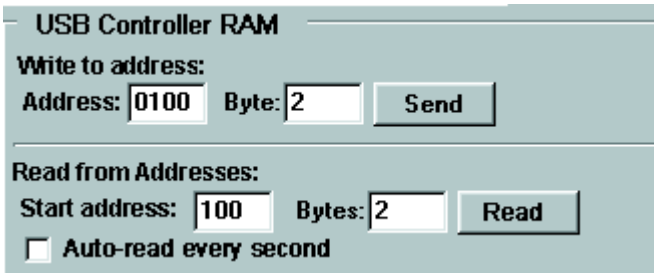


Figure 5. Checking the RAM region.

To produce a program using Delphi, you first generate a new application using File → New → Application. An empty form will appear. Place two 'TButton' buttons in this form. Add the code shown in **Listing 2** between the end of the first User section and the beginning of the first Type section in the program.

The USB device is operated using the program RamTest.hex. If the device is unplugged, the program must be reloaded using BinTerm before the device can again be used. BinTerm can run in the background using RAM query to check the operation of the device by monitoring its RAM. BinTerm will not block the USB device.

Start the Delphi program from the Delphi compiler by pressing F9. When command button 'Button1' is activated, Byte 1 is written to RAM address 0x100 in the USB device. The value in address 0x101 can be read and displayed using 'Button 2' (see **Figure 6**).

Data in the USB device

The function 'WrRAM' can write data to the RAM of the Cypress IC. The function 'RdRAM' reads a byte from the specified address. The device name (which is correct when operating without EEPROM) is stored in the variable 'pGeraet' ('pDevice'). If a BinTerm USB data spy adapter is being used, you must use the name '\\.\binterm-0' instead of '\\.\ezusb-0'. The command 'CreateFile' generates a virtual file that creates a link to the device driver. The command 'DeviceIoControl' enables direct communication with the device driver. A record of type *_VENDOR_REQUEST_IN* is used for run-time monitoring. The command 'CloseHandle' terminates the link to the device driver. Since the link is closed after each request, it is possible for multiple programs to apparently access the hardware concurrently, without the device being aware of this.

By the way, if you rename the above-mentioned functions, all of this will also work with Visual Basic or C++.

(020109-1)

The second article in this series will deal with producing USB device drivers, which does not require a detailed understanding of C!

Listing 2.

```
type _VENDOR_REQUEST_IN = record
   bRequest : Byte;
   wValue : Word;
   wIndex : Word;
   wLength: Word;
   direction : Byte;
   bData : Byte;
end;

Insert the following code snippet at the line containing "implementation"

Function WrRAM(Adr: Word; Dat: Byte): Boolean; // Write byte to RAM address
```

```
var USBDeviceHandle: THandle;
  USBTemplateHandle: THandle;
  nBytes: DWord;
  MyRequest: _VENDOR_REQUEST_IN;
  USBError: Boolean;
  pGeraet: PChar;
begin
  pGeraet := PChar('\\.\ezusb-0'); // using BinTerm: '\\.\binterm-0'
  USBError := False;
  myRequest.bRequest := $A0;
  myRequest.wValue := Adr;
  myRequest.wIndex := 0;
  myRequest.wLength := 1;
  myRequest.direction := 0;
  myRequest.bData := Dat;
  USBTemplateHandle := 0;
  USBDeviceHandle := CreateFile (pGeraet, Generic_write, File_Share_write, nil, open_existing,
                                 0, USBTemplateHandle);
  If USBDeviceHandle = INVALID_HANDLE_VALUE Then
       USBError := True;
  Result := DeviceIoControl(USBDeviceHandle, $00222014, @myRequest, 10, nil, 0, nBytes, nil);
  CloseHandle (USBDeviceHandle);
end;

function RdRAM(Adr: Word): Byte; // Read byte from RAM address
var USBDeviceHandle: THandle;
  USBTemplateHandle: THandle;
  nBytes: DWord;
  MyRequest: _VENDOR_REQUEST_IN;
  Buffer: Array [1..2] of Byte;
  USBError: Boolean;
  pGeraet: PChar;
begin
  pGeraet := PChar('\\.\ezusb-0'); // using BinTerm: '\\.\binterm-0'
  USBError := False;
  USBTemplateHandle := 0;
  myRequest.bRequest := $A0;
  myRequest.wValue := Adr;
  myRequest.wIndex := 0;
  myRequest.wLength := 1;
  myRequest.direction := 1;     // Read
  myRequest.bData := $00;
  USBDeviceHandle := CreateFile (pGeraet, Generic_write, File_Share_write, nil, open_existing,
                                 0, USBTemplateHandle);
  If USBDeviceHandle = INVALID_HANDLE_VALUE Then
       USBError := True;
  DeviceIoControl(USBDeviceHandle,$00222014,@myRequest,10,@Buffer,SizeOf(Buffer),nBytes,nil);
  CloseHandle (USBDeviceHandle);
  RdRAM := Buffer[1];
end;

Double-click on the 1st key and complete the code with this:

procedure TForm1.Button1Click(Sender: TObject);
begin
  WrRAM($100, 1);
end;

The same for the 2nd key:

procedure TForm1.Button2Click(Sender: TObject);
begin
  Button2.Caption := IntToHex(RdRAM($101), 2);
end;
```