A freely programmable USB-Interface for eCS - focussing on the DLP-USB245M

by **Uwe Hinz**, 2007
uhdrelb@t-online.de

Introduction

- OS/2 or eCS are rarely used in

  Data Acquisition and Control (DAC) or
  Measurement and Control

  although being a very reliable platform.

- Just a small number of DAC-Hardware is or was supported by the manufacturers. Drivers are difficult to get or they do not exist at all. Only RS232 serial interfaces are supported.

- DAC – Software as **LabView** , **VEE** or **SampLin** is far from existence on OS/2 or eCS.

A freely programmable USB-Interface for eCS

Introduction - The devices waiting for eCS

My historic Stepper Motor Controller
would have been much more usable
if it had a USB-Interface

and my digital interface, both
needed a USB-Driver...

OLD & NEW

gave the final push to go !

A freely programmable USB-Interface for eCS

Introduction

The wish to use eCS on my workbench for developing electronic devices spread to my collection consisting of

- a Digital Thermometer **DTM 2010** (Parallel interface , non standard)
- a RCL Meter (Parallel interface, non standard)
- an EPROM Programmer (Parallel interface, non standard)

A freely programmable USB-Interface for eCS

Introduction

A multi purpose interface seemed to be a very simple start...

so the **meM-PIO** [11] was the USB Interface to begin with.

Providing 24 digital I/O lines it appeared to be a good choice... but

Introduction

... but no USB Driver for OS/2 or eCS was available by then!?

Fortunately

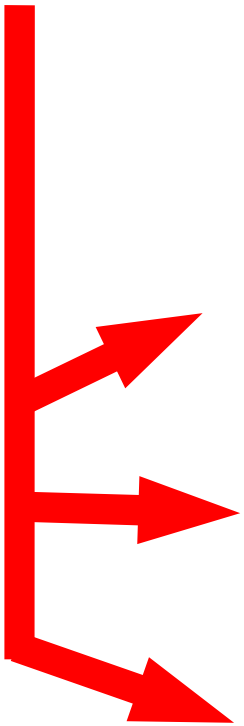
usbecd.sys (written by **Wim Brul** [12])

was mentioned in a 2004 session of the OS/2 User Group Dresden.
It appeared to be necessary for **Cool FM** [13][14], a radio application, able to work with FM radio receiver hardware inluding USB types.

A freely programmable USB-Interface for eCS

Three versions of USBecdXX.sys on hobbes

/pub/os2/system/drivers/misc



usb_uhci_fix01.zip	USB host controller driver (VIA and Intel chipsets) [More info]	1999/11/03	Compressed archive, 15.70Kb
usbecd00.zip	USB 1.1 Expanded Control Driver [More info]	2005/03/18	Compressed archive, 7.32Kb
usbecd10.zip	USB 1.1 Expanded Control Driver [More info]	2005/12/12	Compressed archive, 8.47Kb
usbecd10s.zip	USB 1.1 Expanded Control Driver - source/samples [More info]	2005/12/12	Compressed archive, 63.76Kb

A freely programmable USB-Interface for eCS

List of files in usbecd10.zip

```
The volume label in drive C is VOLUME 3.
The Volume Serial Number is 2933:BC14.
Directory of C:\Desktop\work\USBecdDL\usbecd10

26.05.07 23.18          <DIR>      0 ----  .
26.05.07 23.18          <DIR>      0 ----  ..
11.04.05 10.31           1.875     0 a---  usbecd.sys
 6.12.05 14.25          18.668     0 a---  usbecd.txt
 6.12.05 19.14           2.062     0 a---  usbread.cmd
 6.12.05 19.13           9.635     0 a---  usbwrite.cmd
       6 file(s)      32.240 bytes used
                   1.499.275 K bytes free
```

the driver

the readme

the examples

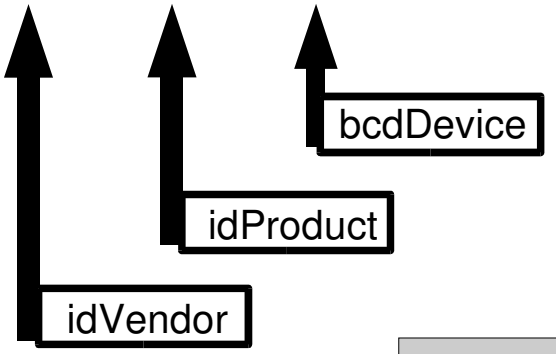
A freely programmable USB-Interface for eCS

Introduction

The way to use the driver goes here:

Put in **config.sys** a set of three numbers complete the D parameter. The three hex numbers make the particular device uniquely identifiable.

```
DEVICE=C:\USBDRVS\USBECD.SYS /D:0000:0000:0000 /N:$$$$$$$$ /S /V
```

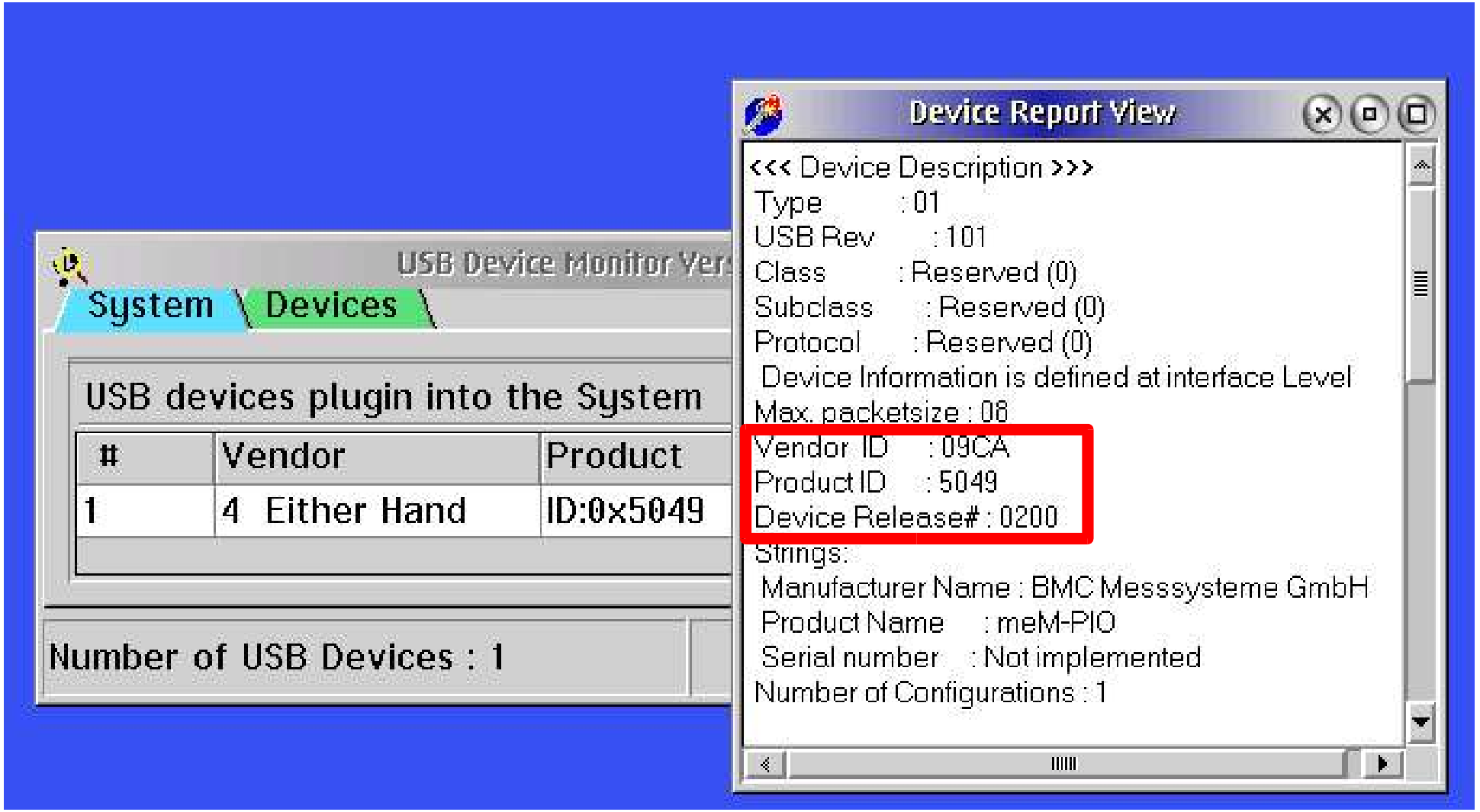


If not found in the documentation of the USB-Device, the three numbers can be checked with the eCS USB monitor easily

All parameters explained in **usbecd.txt**

A freely programmable USB-Interface for eCS

USB Device Monitor screenshot (meM-PIO)



DEVICE=USBECD.SYS /D:09CA:5049:0200 /N:$ /S /V

A freely programmable USB-Interface for eCS

The examples - output of the REXX script usbwrite.cmd

Text Editor-S:\MemPIO\260906\USBECD00\PIOTXT.TXT

File Edit Options Help

bLength	12	Descriptor Length
bDescriptorType	01	Descriptor Type
bcdUSB	0101	USB specification release number
bDeviceClass	00	Device Class code
bDeviceSubClass	00	Device Sub Class code
bDeviceProtocol	00	Device Protocol code
bMaxPacketSize0	08	Maximum Packet Size for endpoint 0
idVendor	09CA	Vendor identification
idProduct	5049	Product identification
bcdDevice	0200	Device release number
iManufacturer	01	BMC Messsysteme GmbH
iProduct	02	meM-PIO
iSerialNumber	00	No String!
bNumConfigurations	01	Number of possible Configurations

Device Driver $ - Device Descriptor

bLength	09	Descriptor Length
bDescriptorType	02	Descriptor Type
wTotalLength	0020	Total Length of data returned
bNumInterfaces	01	Number of Interfaces supported
bConfigurationValue	01	Set Configuration parameter Value

INS Read-only

The USB-Device has got how many interfaces ? How many configurations ? - Listed here !

A freely programmable USB-Interface for eCS

The next level

After Wim Brul's examples work, the next question is:

- How do I know what I have to send to my USB device to make it work ?
- What can I expect, my USB device will send me back ?

If the manufacturer has got a good documentation – well, if not, the field of thorough investigation will be entered !

To do this, an OS different from eCS has to be used for spying !

How to spy on the USB traffic ?

The first try to spy on the traffic of the **meM-PIO** was a DELPHI example, from the driver CD, running on Win98.

With the free Windows tool **usbsniffer** [10] I wanted to see what telegram the USB device **meM-PIO** gets or sends.

Cutting a long and frustrating story short, I saw a lot but I was not able to interpret the swarm of bytes...

I gave up with it !

A freely programmable USB-Interface for eCS

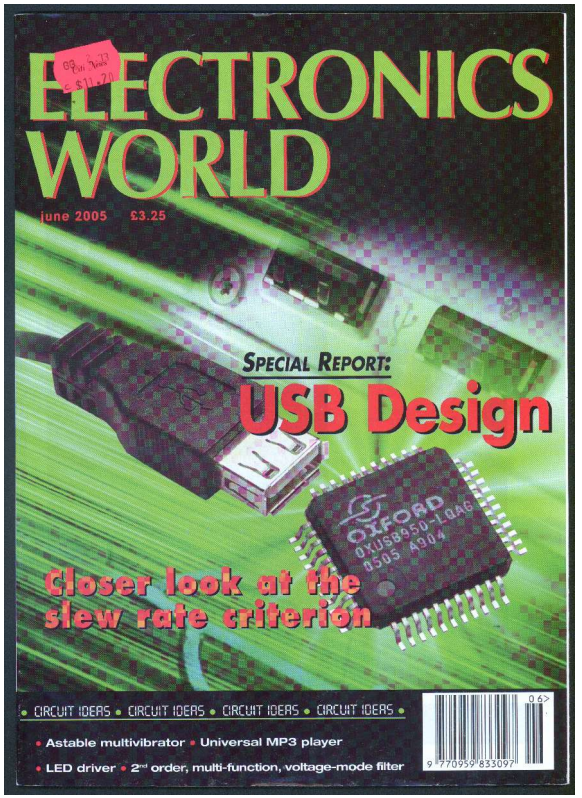
A different approach

The decision to give up upon the **meM-PIO** was made shortly before the Developers Workshop 2005 Dresden.

A few days after, a copy of

' ELECTRONICS WORLD '

popped out of nowhere.



In the article

'Design of a USB interface ...for a data acquisition system'
by **Qian Xie** and **Wuquian Yang**
ELECTRONICS WORLD 111 (2005, June) 1830, p. 18-26 [1]

the USB interface module **DLP-USB245M**

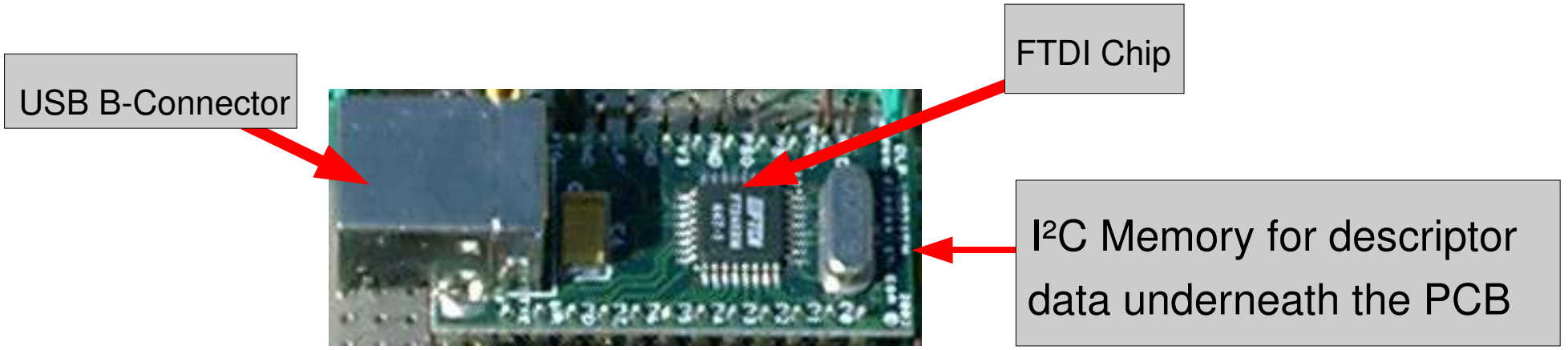
was mentioned...

A freely programmable USB-Interface for eCS

The other form of communication – the DLP-USB245M

As a module for electronic design it offered the opportunity to establish a form of data exchange protocol that **can be freely invented** and would therefore **not suffer from any obscurity** !

The module **DLP-USB245M** not bigger than a DIL28 chip with a USB-B connector already mounted,



appeared to be the ideal solution to overcome the meM-PIO problems.

(Supplier is named in [6]. On p.89 the module is listed.)

A freely programmable USB-Interface for eCS

How to spy on the USB traffic without hardware tools

The standard situation looks like this.

No helping hardware.

Spying is just possible with debugging tools.

PC (any OS)

USB cable

DLP-USB245M

If debugging is to much fuss, the USB sniffer is an option.

But there is still the problem to collect knowledge about USB on eCS – the unknown !

A freely programmable USB-Interface for eCS

The spy hardware

Additionally I ordered an USB analyser. Just to avoid more tinkering.

Market investigation lead to the

Ellisys 110 .

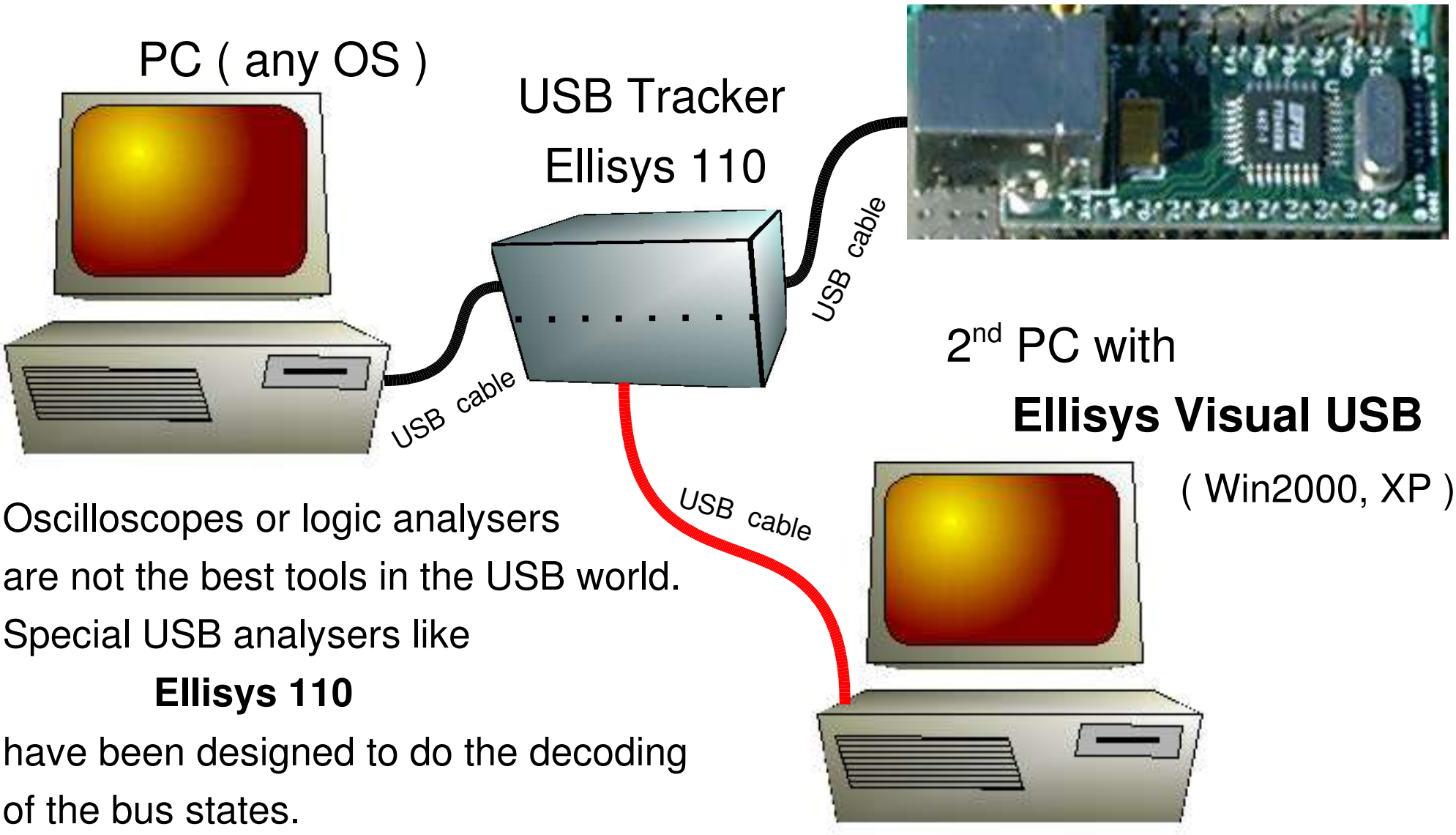
Including the analyser program

Ellisys Visual USB

for Win2000, it promised professional support.

A freely programmable USB-Interface for eCS

How to spy on the USB traffic



Oscilloscopes or logic analysers
are not the best tools in the USB world.
Special USB analysers like

Ellisys 110

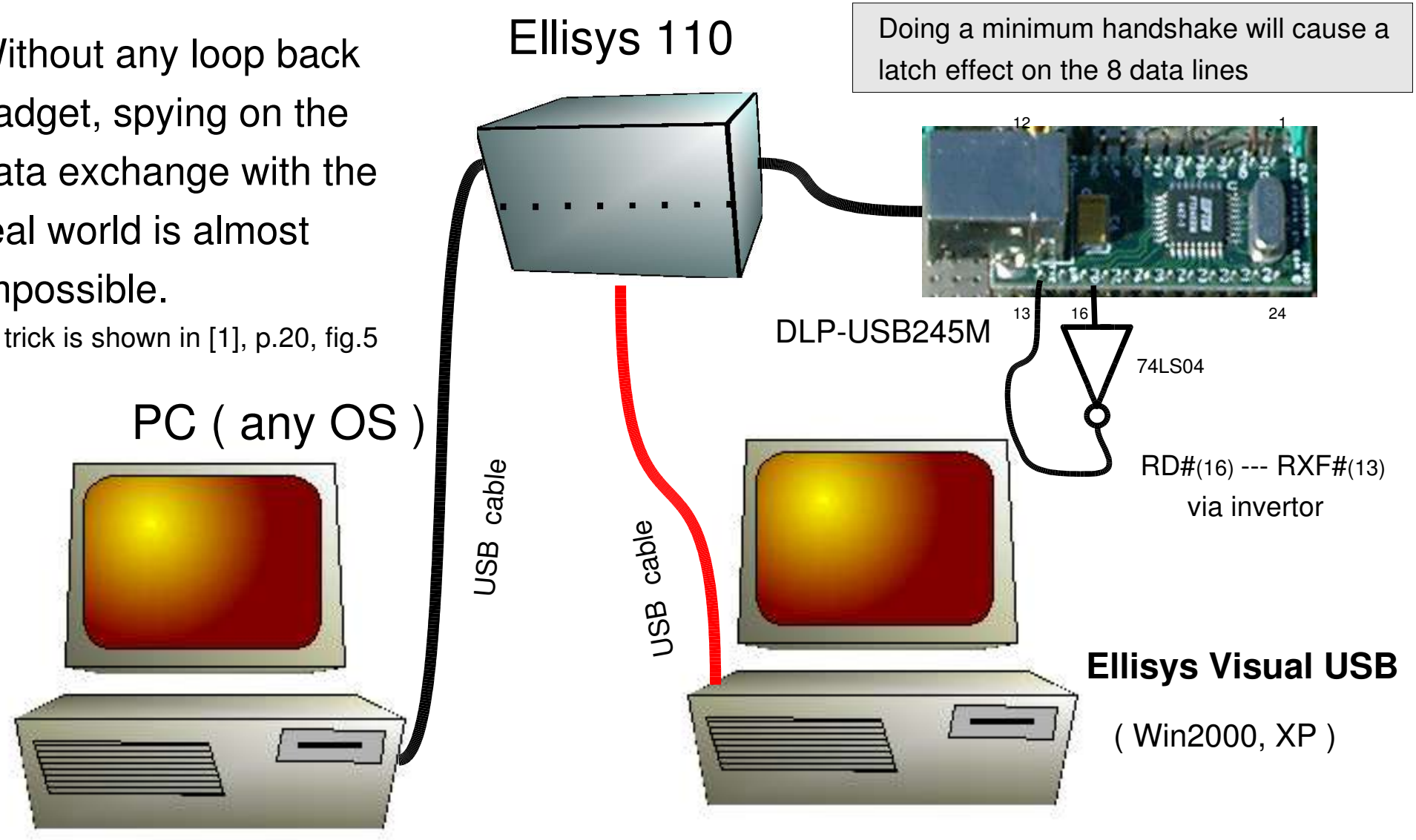
have been designed to do the decoding
of the bus states.

A freely programmable USB-Interface for eCS

Spying on the data exchange

Without any loop back gadget, spying on the data exchange with the real world is almost impossible.

A trick is shown in [1], p.20, fig.5



A freely programmable USB-Interface for eCS

The other form of communication – the DLP-USB245M test application

Shipped with its test application **dlptest 1.0b** the **DLP-USB245M** offers an ideal opportunity for spying on the USB traffic with **Win98**.

The wanted knowledge for eCS !

Connecting the Module to Win98

(two ways to connect to Windows)

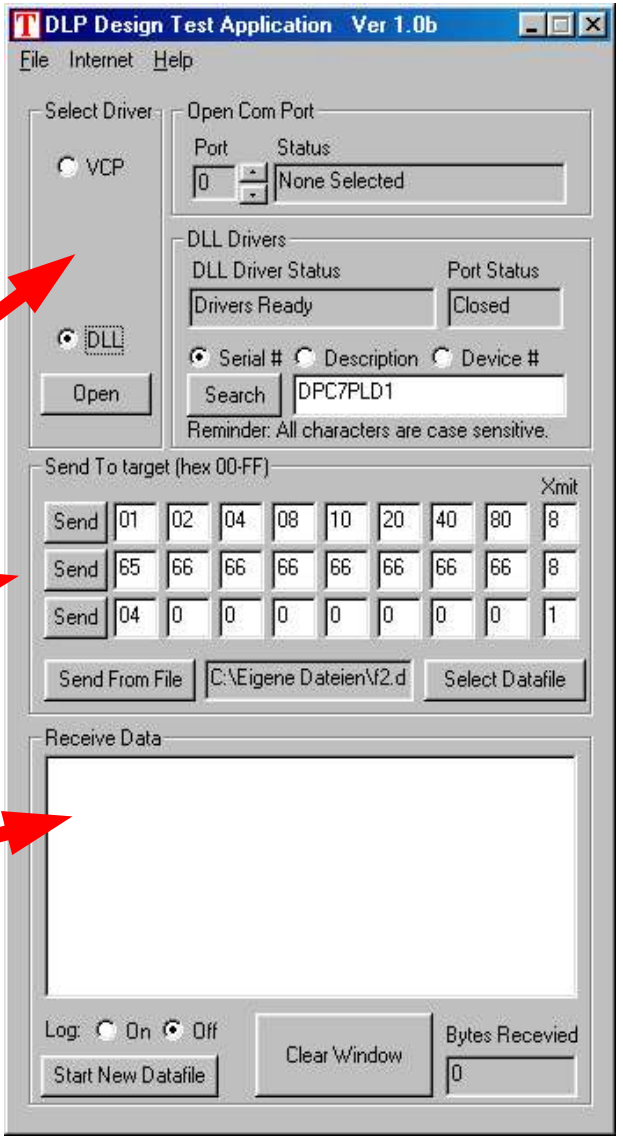
Sending bytes

(3 times 8 bytes can be sent to the Module)

Receiving bytes

(a loop back facility is needed here)

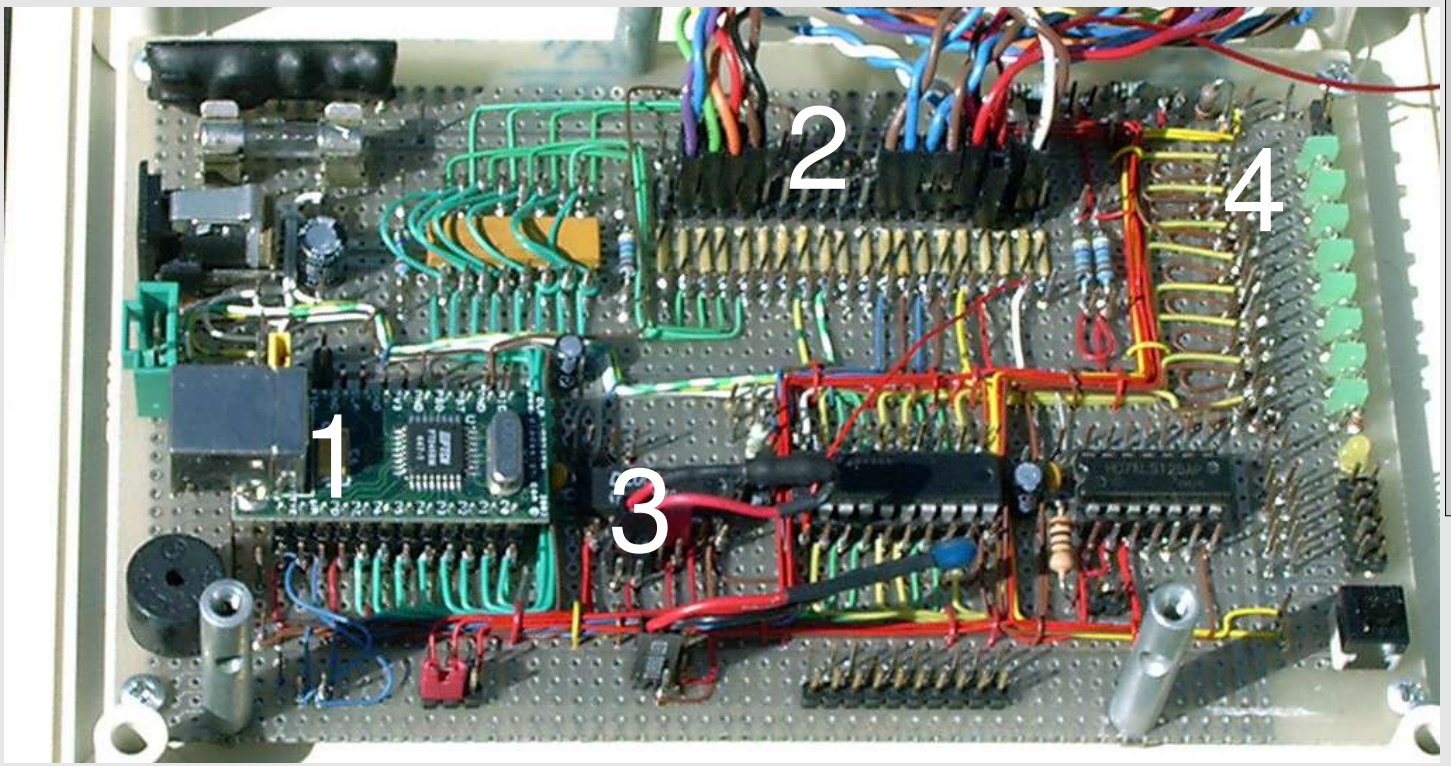
!



A freely programmable USB-Interface for eCS

The other form of communication – the DLP-USB245M and its adaptor

In order to have some place for experimental wires and connectors, this is the adaptor special for the **DLP-USB245M** module. Spying and testing both requires proper conditions...



1 - DLB-USB245M

2 - MCU connector

3 - handshake chips

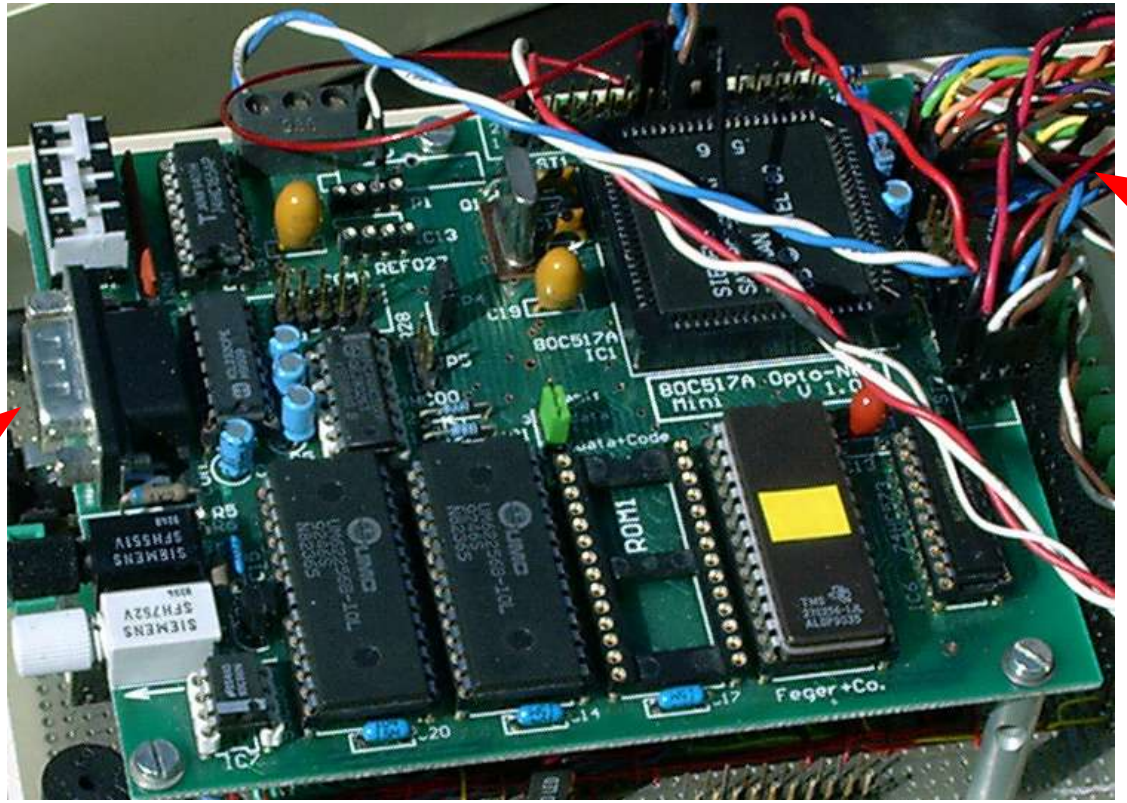
4 - 8 LED indicator

A freely programmable USB-Interface for eCS

The other side of communication – the DLP-USB245M and its MCU

As an MCU, a commercial PCB supplement to the book **MC-Tools 13** [2] was chosen.

The processor on the PCB is a **SAB80C537** (8051). Development software [3] is available all over the Web. On the diskette in the book the user can find some examples.



MCU to DLP-USB245M wiring

Serial Terminal Connector (RS232, SubD-9)

A freely programmable USB-Interface for eCS

Developing for SAB80C537 under OS/2 and eCS

Otmar Feger and **Jürgen Ortmann** provide software in [1]. For their PCB they have included a monitor designed for the SAB80C537. It has to be put on a 32KByte EPROM (27C256) before development can start.

This monitor is able to communicate via almost any terminal emulator, but most important, it is to be said, that uploading a user programm to the onboard RAM of the PCB can only be done with Feger/Ortmann's special terminal program

mon517.exe !

It may look like a restriction, but build for DOS, it **works with OS/2 and eCS** smoothly !

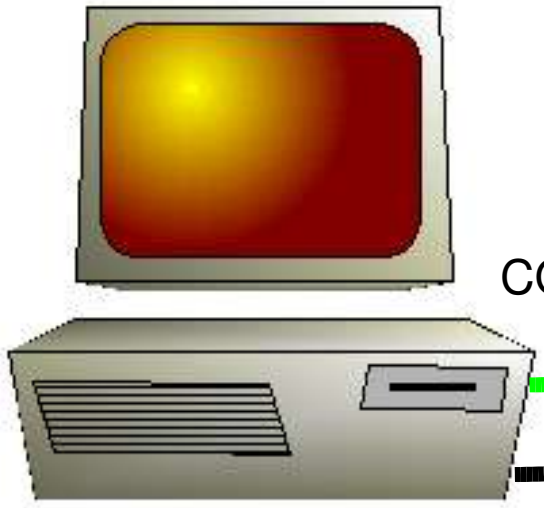
The assembler **ASM51** and the monitor **mon517.exe** work on OS/2 alltogether. They make it possible to have a complete development environment for OS/2 and eCS.

A freely programmable USB-Interface for eCS

Developing the MCU program with eCS

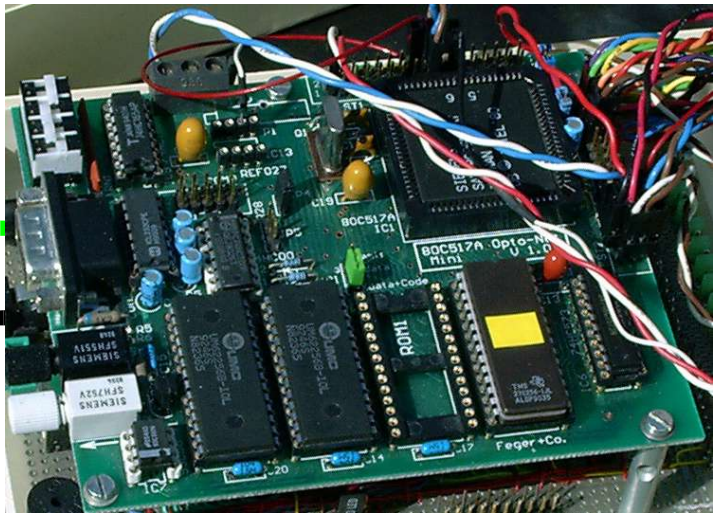
The basics of the development workplace are not so complicated.
The Wim Brul driver

usbecd.sys (v.10)
and **ASM51** plus **mon517.exe** are needed



RS232 cable for **mon517.exe**

USB cable



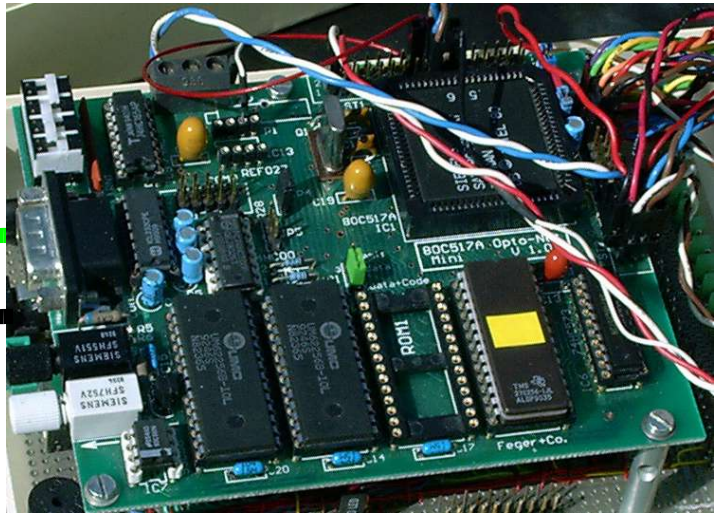
DLP-USB245M, adaptor and MCU

A freely programmable USB-Interface for eCS

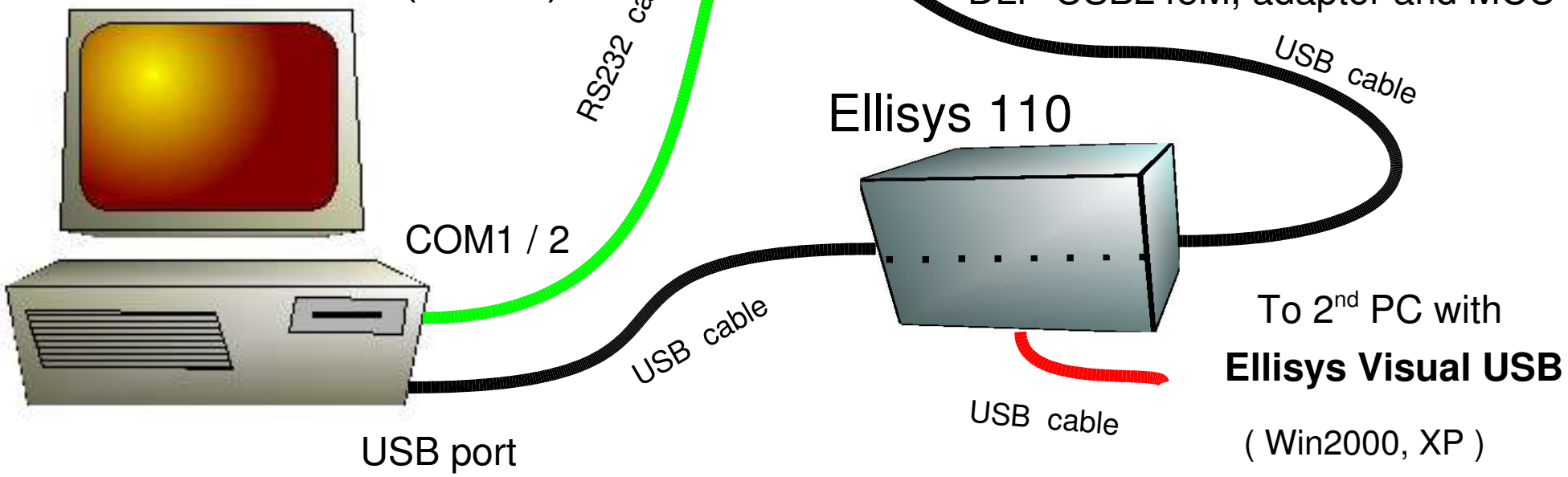
Developing the MCU software with eCS

From time to time the developer has got to spy on its own work.

USB traffic with **Endpoint 0** and the data exchange can be analysed and interpreted in a very detailed manner.



DLP-USB245M, adaptor and MCU



The examples

After a successful installation the focus of the problem changes. This is based upon the unpleasant fact that the provided examples **do not grant** any look onto the exchanged data between eCS and the USB-Device.

Wim Brul's examples show Endpoint 0 communication primarily !

With **usbwrite.cmd** (REXX) the user can request a nice list of descriptors concerning the particular USB-Device.

A freely programmable USB-Interface for eCS

The Wim Brul example 'usbwrite.cmd'

All kind of USB funtionality in 8 byte frames

```
OutputDeviceDescription:
oiBuffer = substr(x2c(80 06 00 01 00 00 12 00),1,26,x2c(00))
call WriteSetupAndReadDescriptor
...

WriteSetupAndReadDescriptor:
rc=charout(ddName,oiBuffer)
/* check completion code */
rc=stream(ddName,'description')
if rc \= 'READY:'
then do
  /* obtain and issue error message */
 ...
```

26 bytes with 0x00, 8 bytes for the frame -> 18 bytes remaining for an empty appendix (lorry for payload).

80 06 00 01 00 00 12 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

80 06 00 01 00 00 12 00 12 01 10 01 00 00 00 08 03 04 01 60 00 04 01 02 03 01

What is in the 8 bytes

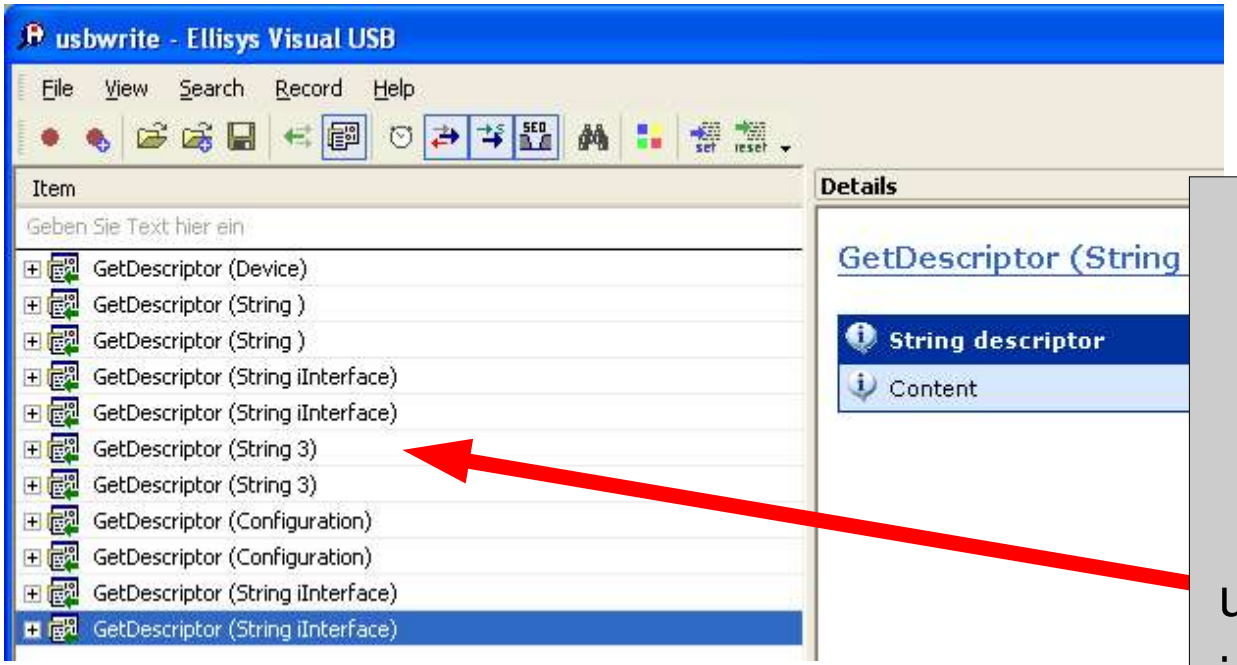
The meaning of every bit and every byte determines the variety of communication that is going on on the universal serial bus.

One line of REXX code in Wim Brul's examples. The exact USB significance [9] added as comments.

```
  oiBuffer = x2c(80 06 00 01 00 00 12 00)
/*                |  |  |     |     ^^-^^------------------ wLength   */
/*                |  |  |     |     0012H(18):  Maximum Block Size.   */
/*                |  |  |     |        p. 189, 9.4.3 Get Descriptor   */
/*                |  |  |     ^^-^^------------------------- wIndex   */
/*                |  |  |     0000H:  Zero by definition.             */
/*                |  |  |        p. 189, 9.4.3 Get Descriptor         */
/*                |  |  ^^-^^------------------------------- wValue   */
/*                |  |  0100H = DEVICE:  Descriptor type             */
/*                |  |     p. 187, Table 9-5, 1st row                */
/*                |  ^^---------------------------------- bRequest   */
/*                |  06 = GETDESCRIPTOR                             */
/*                |   p. 187, Table 9-4, 7th row                    */
/*               ^^------------------------------- bmRequestType   */
/*               80h -> 1  00  00000b                               */
/*                      |  |   ^                                    */
/*                      |  ^   0 = Device                           */
/*                      ^  0 = Standard                             */
/*                      1 = Device-to-host                          */
/*                      p. 183, Table 9-2, 1st row                  */
```

Descriptors only

Wim Brul's example **usbwrite.cmd** has got an unavoidable disadvantage - it deals with descriptors only! Any other form of USB communication is the developer's task.



Spying on

usbwrite.cmd

uncovers desciptors in variouse forms.

```
80 06 xx xx xx xx xx xx
```

The big question remains. What is the **DLP-USB245M** to be told for working.

A freely programmable USB-Interface for eCS

DLP-USB245M traffic recorded with a test application on Win98

32 seconds

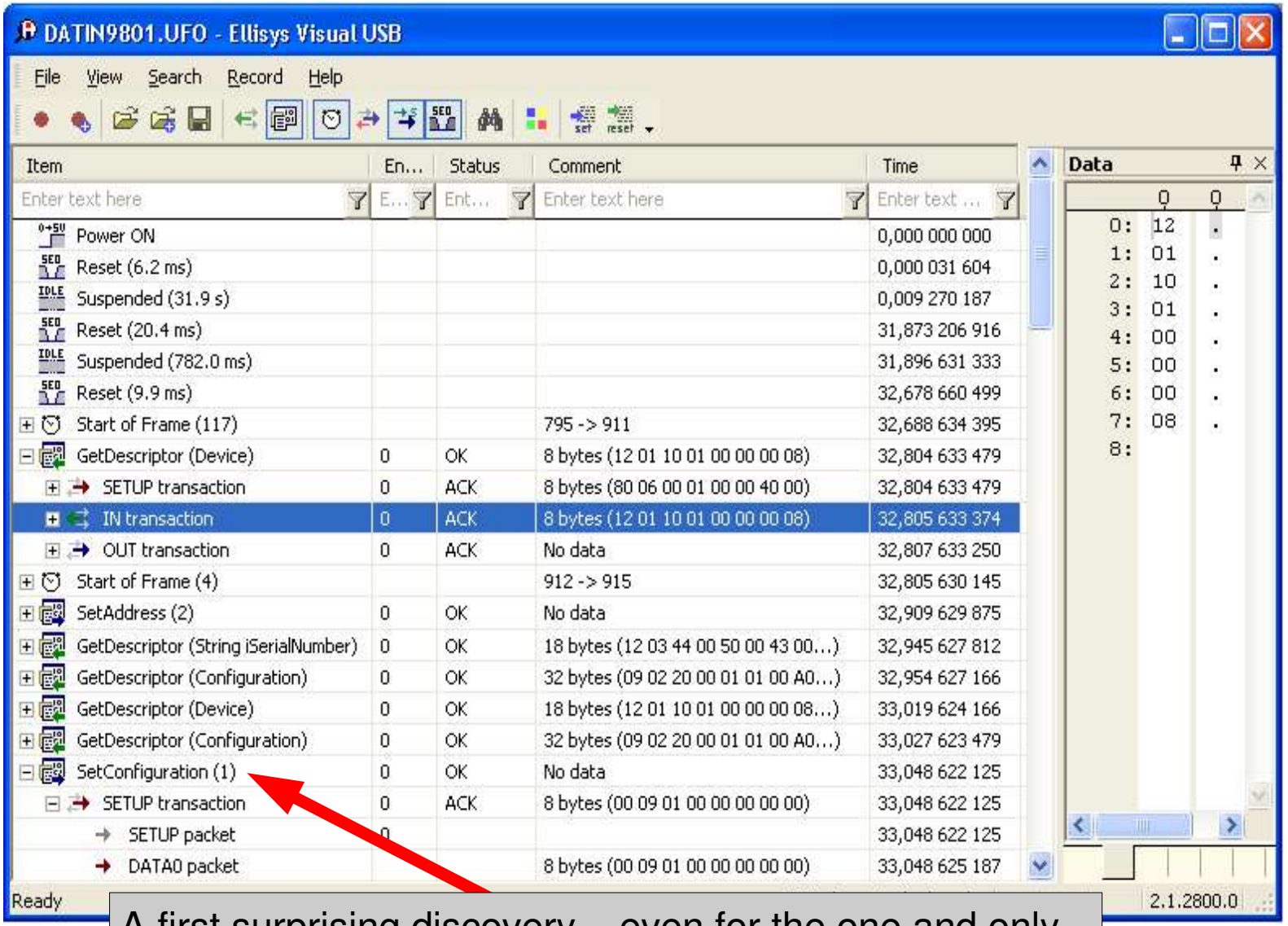
from that moment on when the device was connected with the USB-Hub

to the important event of setting the (first) configuration.

The USB Tracker

Ellisys 110

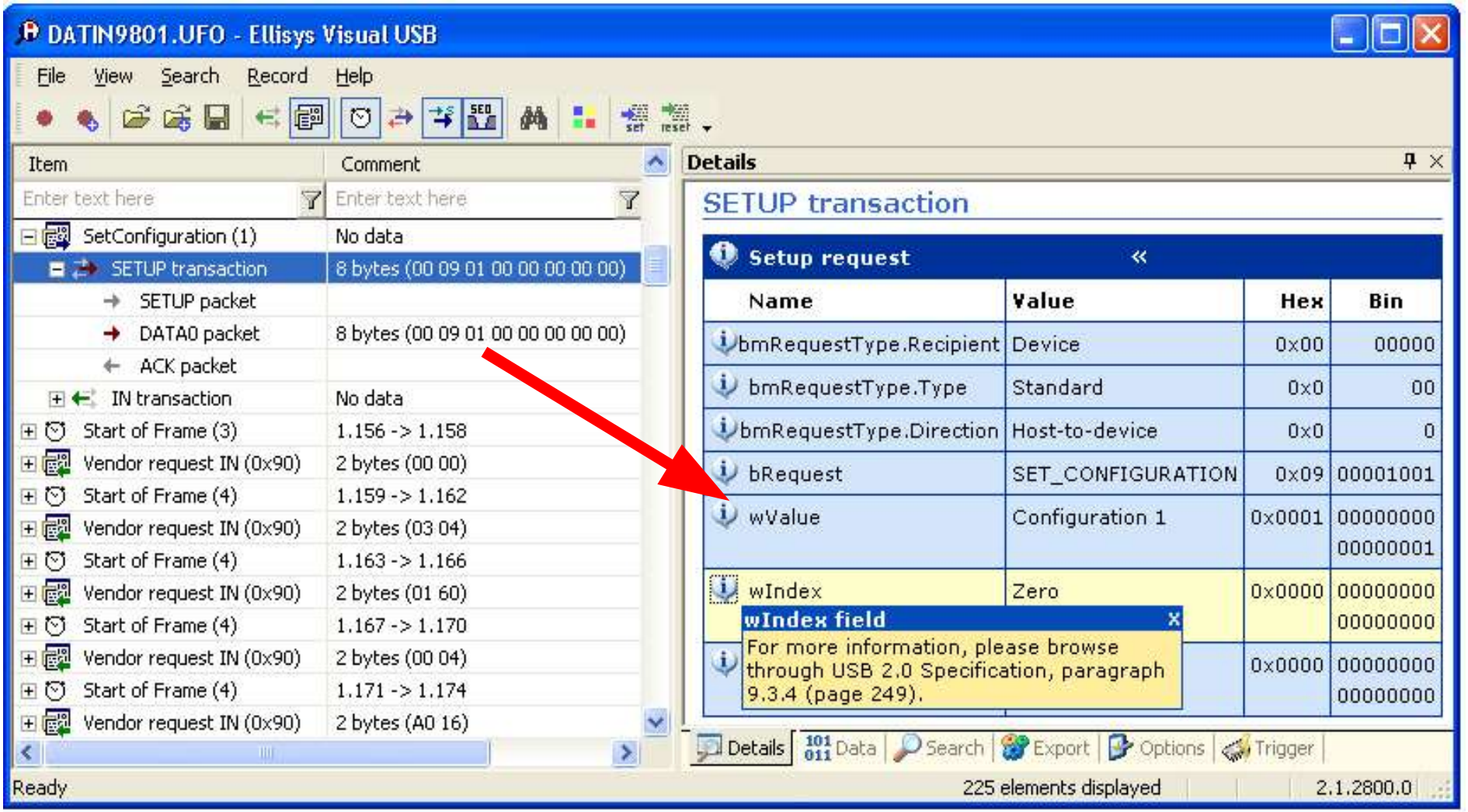
and its super front end provide an excellent unveiling of details.



A first surprising discovery – even for the one and only configuration a SETUP is needed.

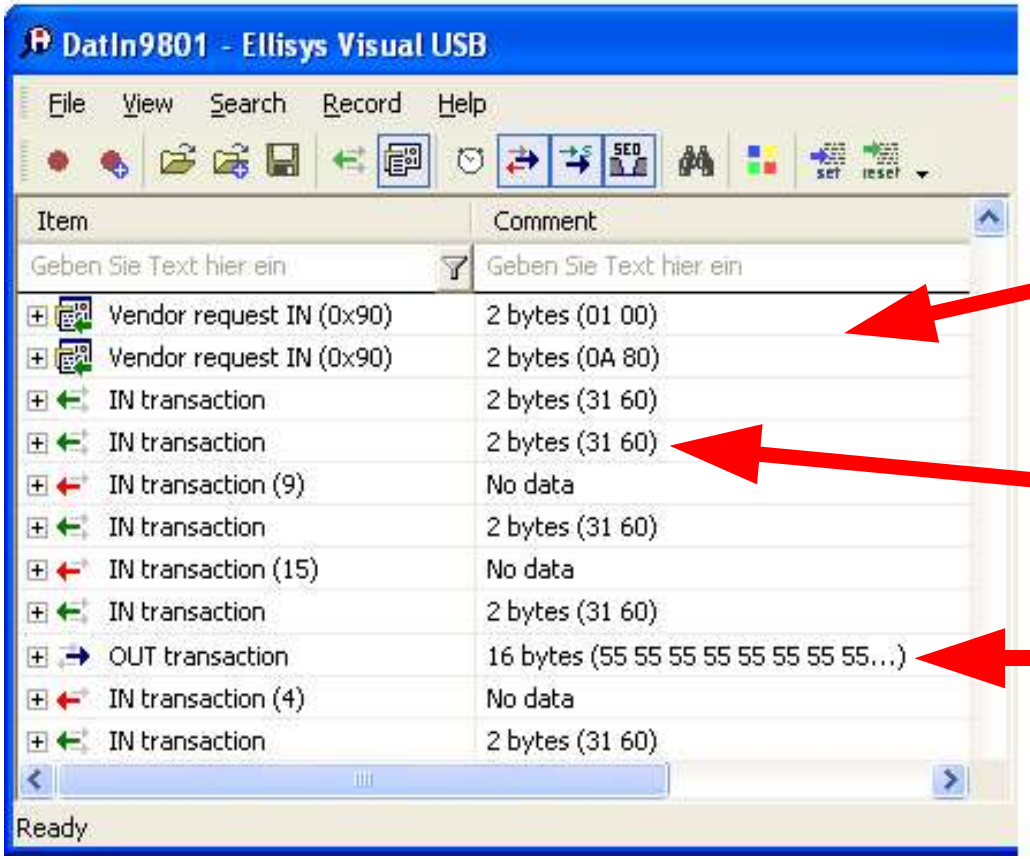
A freely programmable USB-Interface for eCS

DLP-USB245M traffic with detailed USB references



Bits and bytes can be traced by their official names. Even exact pages of the USB Specification are mentioned. Some fields in a data packet can be cracked to atoms of information !

DLP-USB245M traffic with Vendor Requests and Heartbeat



Vendor Requests – the obscure. After 64 Vendor Requests the **DLP-USB245M** is set to work

A kind of heartbeat indicates the working **DLP-USB245M**.

The first set of data visible with the **Ellisys 110** ...

The Vendor Requests do not seem to have an important significance. They have just to be done but cannot be omitted. **After them, the DLP-USB245M is up** and running**.**

A freely programmable USB-Interface for eCS

DLP-USB245M traffic for data

The USB driver wants **0xEC** in the request type byte to do data exchange instead of setup.

Endpoints, transfer types and data length may vary in data exchange...

C:\USBECD10S\usbecd.txt

File Edit Options Help

bmRequestType	{0xEC} Parameter
bmToggle	{0/8} Data Toggle
wValue0	{0} Not Used
wValue1	{0} Not Used
bEndpointAddress	Endpoint Address
bmAttributes	Transfer Type
wLength	Data Length

for setup

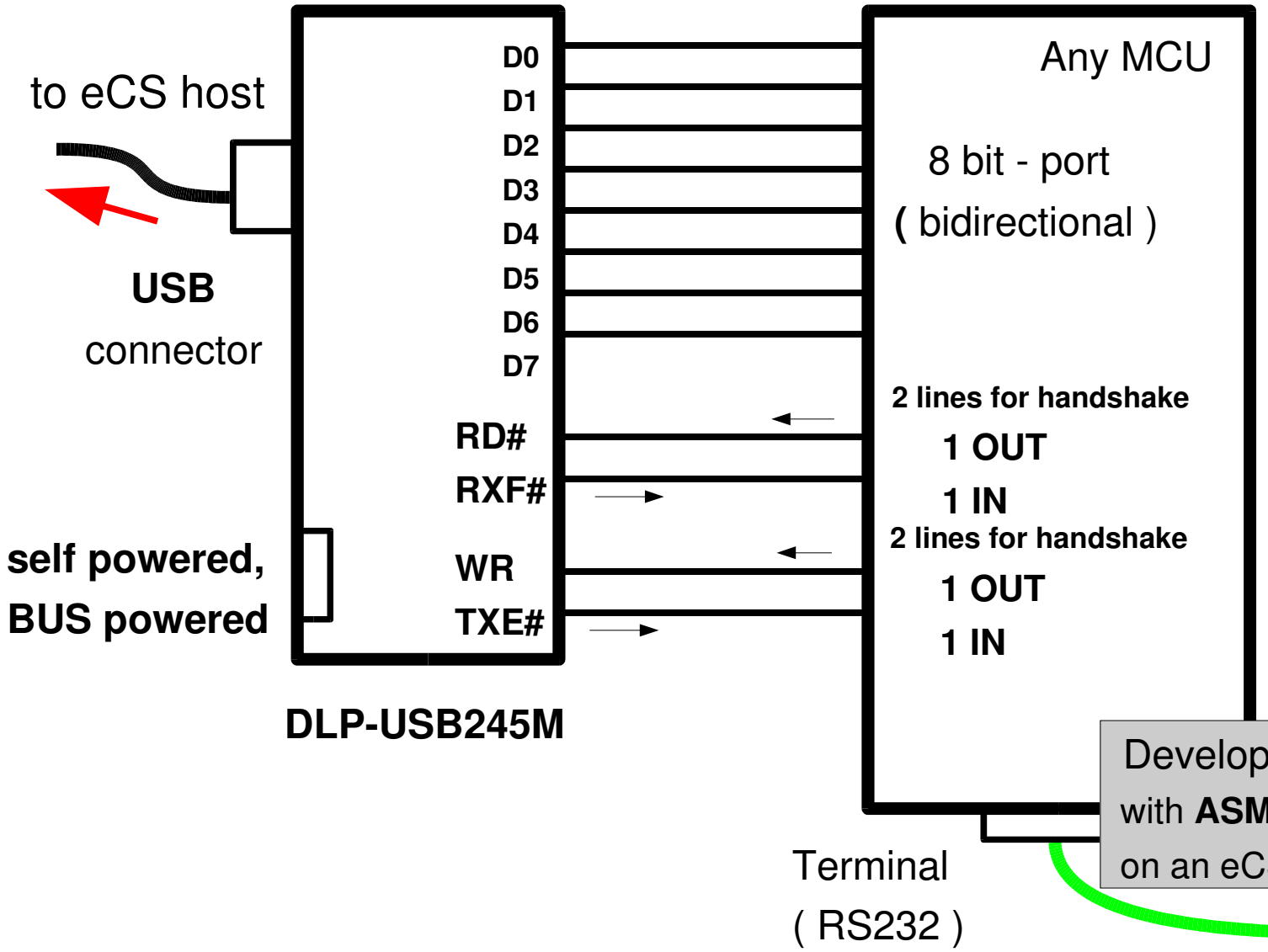
```
80 06 xx xx xx xx xx xx
```

for data exchange

```
EC 0x 00 00 xx xx xx xx
```

A freely programmable USB-Interface for eCS

DLP-USB245M with MCU

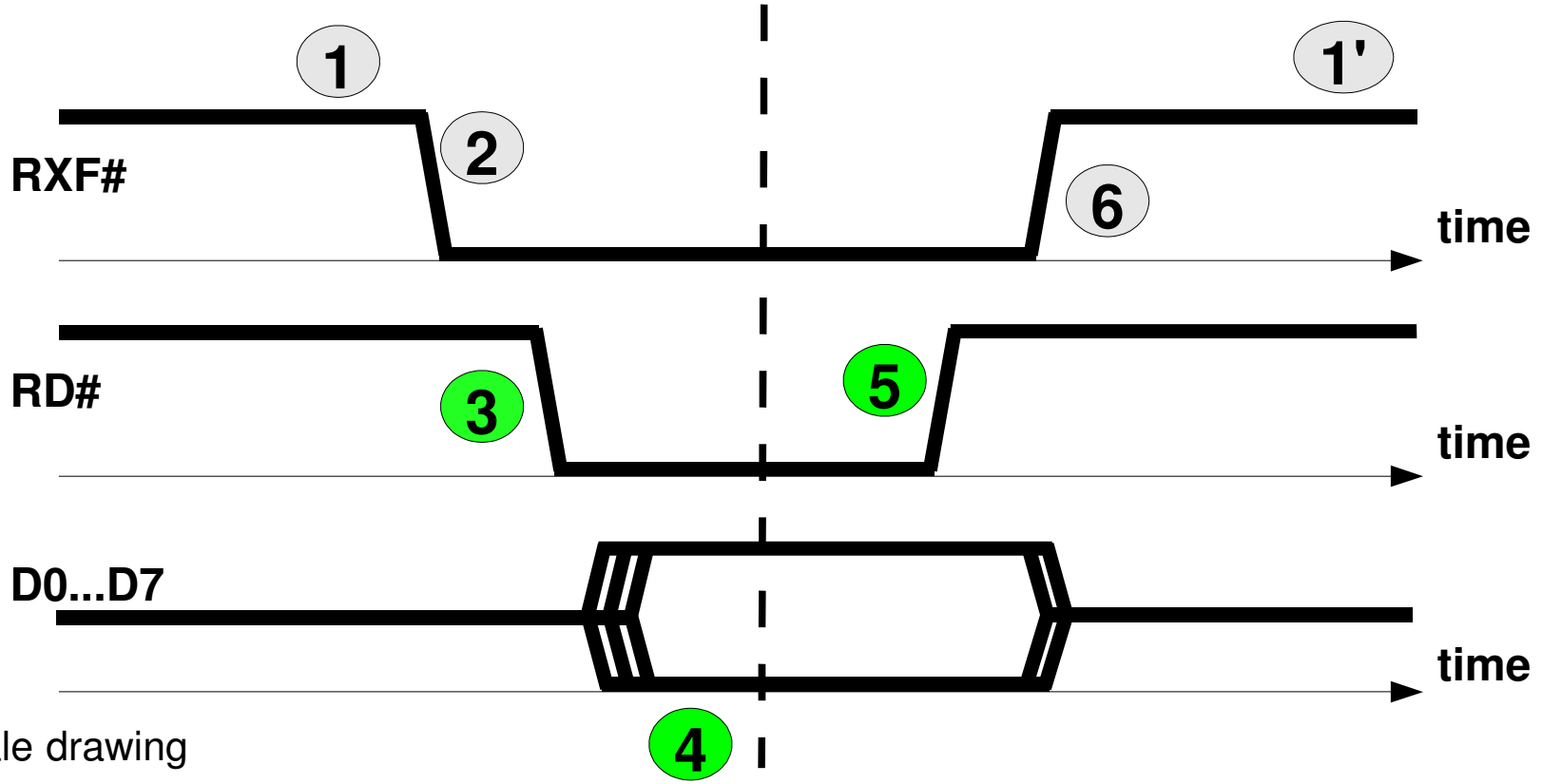


As an MCU almost any type is suited.

Z80 core,
i8051,
PIC,
AVR [15]...
Compatibility with OS/2 is good but not needed.

Development for **SAB80c537** with **ASM51** and **mon517.exe** on an eCS/DOS host via COMx

DLP-USB245M timing to read one byte by the MCU



free scale drawing

1. Waiting for H/L edge (FIFO still empty**?**)
2. Identified H/L edge (byte(s) in FIFO **!**)
3. Setting RD to LOW (active)
4. Waiting for stable levels and reading D0...D7 (time of dashed line)

5. Setting RD to HIGH (passive)
6. Waiting for L/H edge (FIFO empty again**?**)

1'. „The same procedure as...“

A freely programmable USB-Interface for eCS

DLP-USB245M with MCU

Messages from eCS to the data lines of the DLP-USB245M may have almost any format.
It is more important to say how the program on the MCU can deal with the bytes that arrive.
This is the decision point for the simplicity of the MCU program.

The messages should be structured as simple as possible.

1. A message has a fixed length. 16 characters plus 2 delimittes (CRLF)
2. A message must start with a letter and must end with a delimiter
3. Characters in a message shall be printable ASCII only. (20h...7Fh)
4. Substructures in a message are separated by commas.
5. Lower case characters go from the eCS host via DLP-USB245M to the MCU.
6. Upper case characters go from the MCU via DLP-USB245M to the eCS host.

```
c=a0,..........,
```

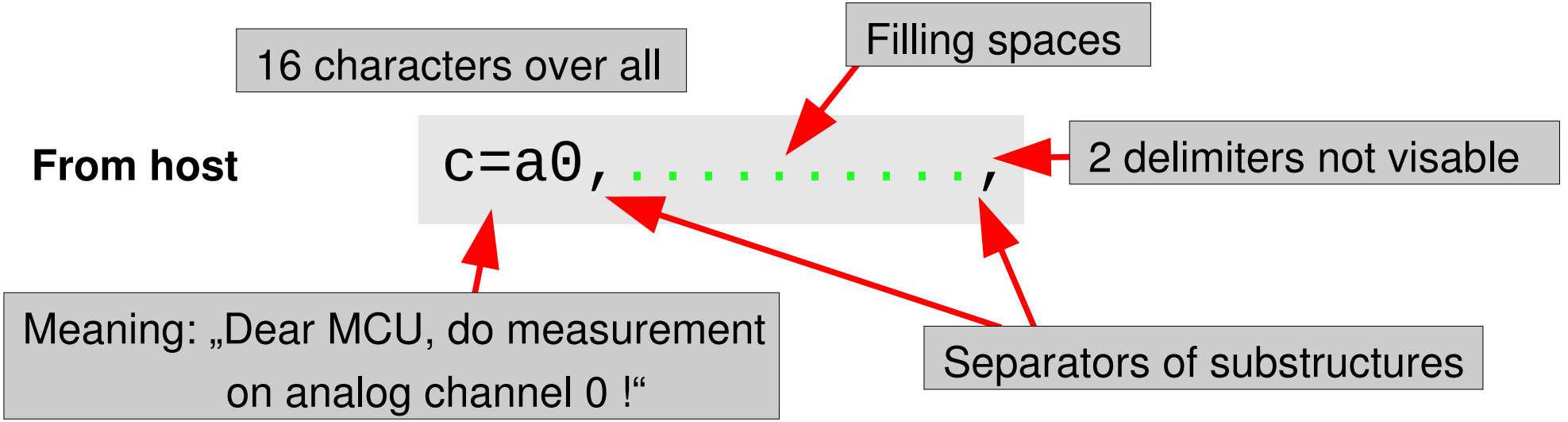
eCS host via DLP-USB245M to MCU

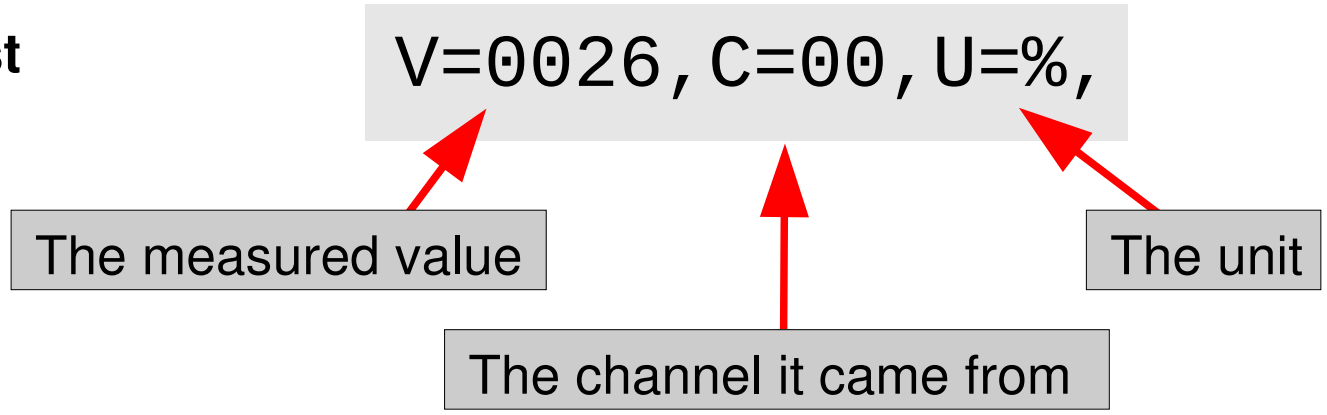
```
V=0026,C=00,U=%,
```

MCU via DLP-USB245M to eCS host

A freely programmable USB-Interface for eCS

Messages explained

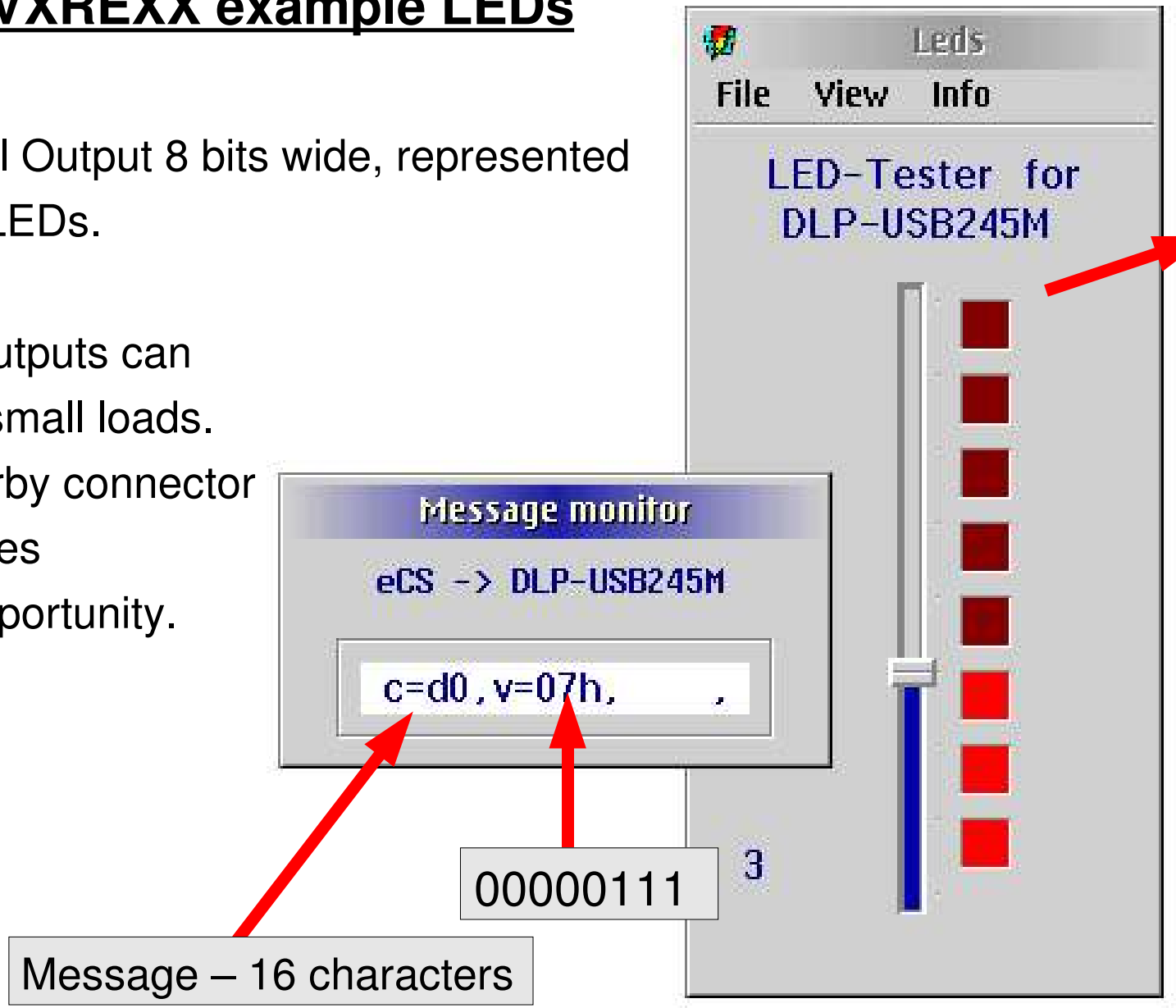




The VXREXX example LEDs

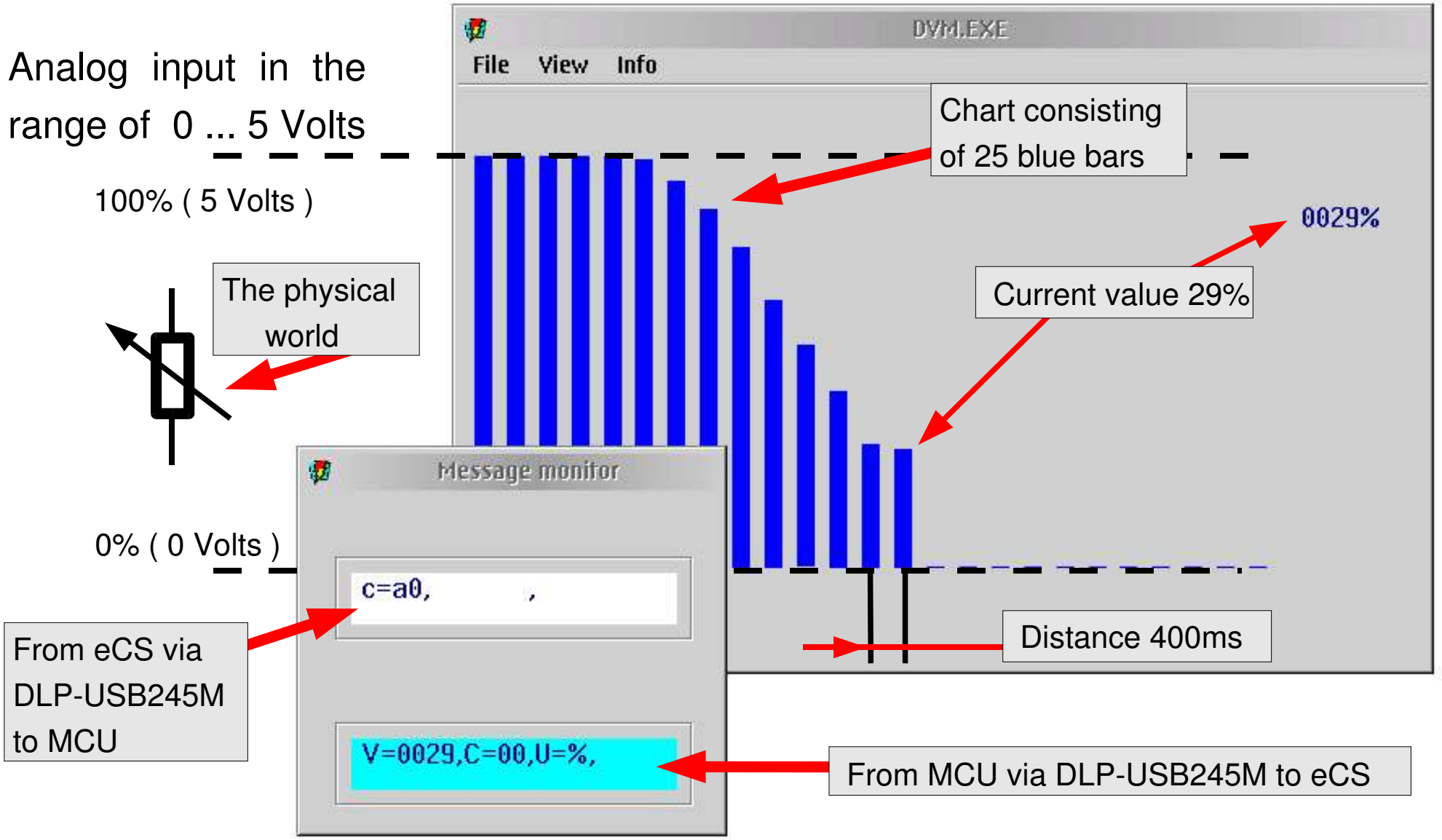
Digital Output 8 bits wide, represented by 8 LEDs.

TTL outputs can drive small loads.
A nearby connector provides the opportunity.



A freely programmable USB-Interface for eCS

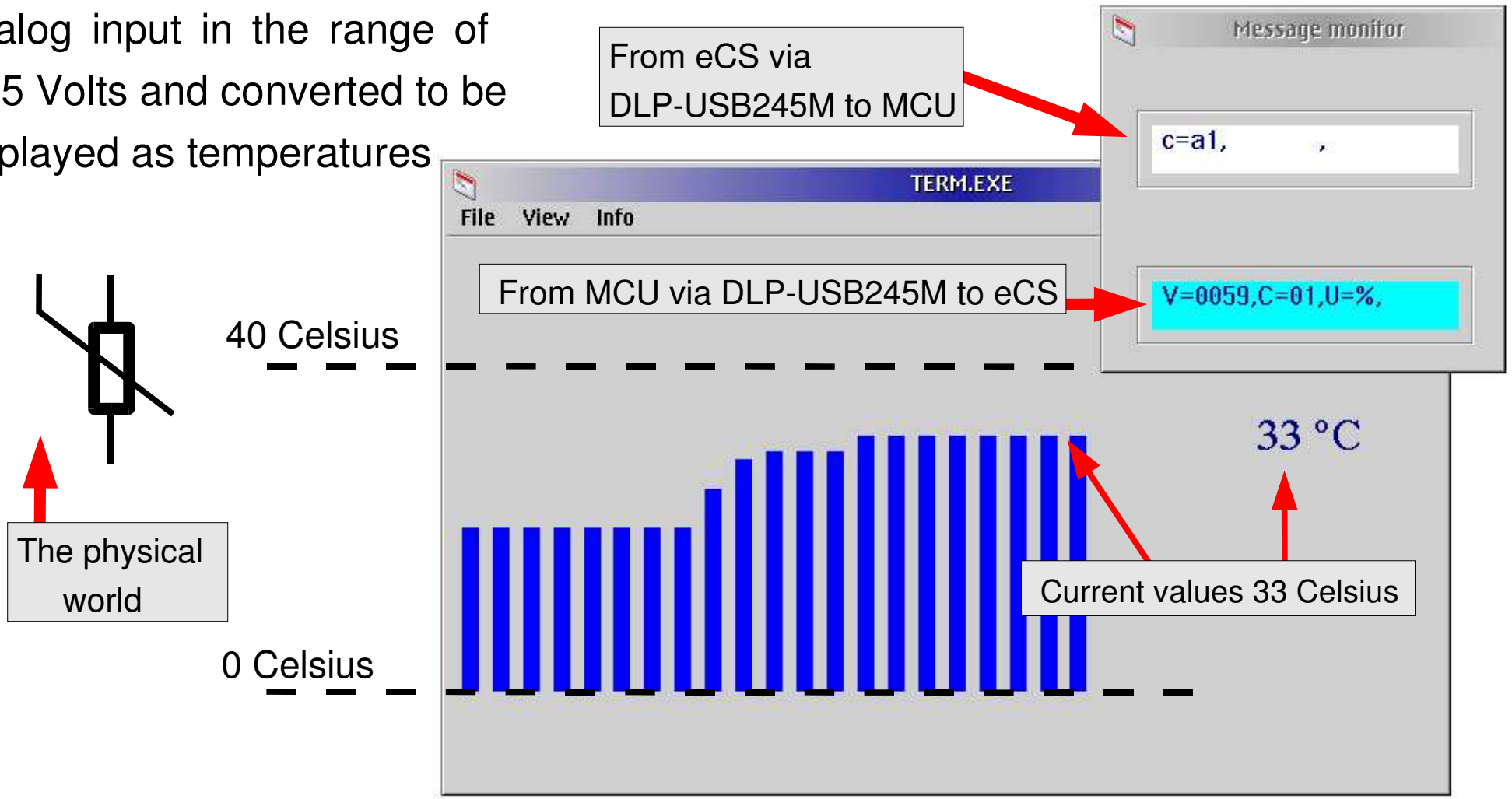
The VXREXX example DVM



A freely programmable USB-Interface for eCS

The VXREXX example TERM

Analog input in the range of 0...5 Volts and converted to be displayed as temperatures



References 1

[1] **Qian Xie, Wuquian Yang** 'Design of a USB interface ...for a data acquisition system' ELECTRONICS WORLD 111 (2005, June) 1830, p. 18-26, ISSN 0959-8332

[2] **Feger, Otmar; Ortmann, Jürgen** 'MC-Tools 13' Feger-und-Co.-Hardware-und-Software-Verl.-OHG. 1993, ISBN 3-928434-17-9

[3] http://www.metaice.com/ASM51/Files/ASM51.zip

[4] http://www.metaice.com/ASM51/Files/ASM51MAN.pdf

[5] **Arnold, Alfred** 'Assembler' Elektor-Verlag, Aachen 1995, ISBN 3-89576-007-2

[6] **Elektronikladen ELMICRO** 'Von EMUFs und EPACs' Preisliste vom 13. Februar 2007, 'ee06.pdf'

[7] http://www.elektronikladen.de, http://elmicro.com

References 2

[8] http://www.dlpdesign.com

[9] **Compaq, Intel, Microsoft, NEC** 'Universal Serial Bus Specification' Revision 1.1, September 23, 1998

[10] http://benoit.papillault.free.fr/usbsnoop/

[11] http://www.bmc-messsysteme.de/ger/sitemap.html

[12] http://home.hccnet.nl/w.m.brul/usbprobe/index.html

[13] http://en.ecomstation.ru/projects/coolfm/

[14] http://www.os2site.com/sw/mmedia/radio/index.html

[15] ' /pub/os2/dev/asm ' on hobbes

Acknowledgement

I thank

my physics teacher **Mr. Hugo**, who allowed me to do experiments in the class room after school,

my electronics teacher **Mr. Duda**, who sold me an oscilloscope, a very heavy one with valves, my first professional device,

Stefan Günther, my friend, who made me an OS/2 user,

Karl-Heinz Sommer, my colleague, who listened to me so many times,

All members of the OS/2 User Group Dresden,

Charlotte Koebel, my step granddaughter, who did the calibration of the temperature sensor for the TERM example

and
Heidi Fritzlar, my wife, for being so appreciative of my time consuming hobby...