Ein Kapitel aus dem Buch „Messen, Steuern, Regeln mit dem PC“ von Burkhard Kainka, Franzis-Verlag 2000. Für den privaten Gebrauch freigegeben vom Autor. www.b-kainka.de

7 Treiber

Bei den bisherigen Versuchen wurde zunächst der Thermometer-Treiber von Cypress verwendet. Auch die erweiterten Anwendungen liefen als "Thermometer". Ein komplett eigene Lösung benötigt auch einen eigenen Treiber. Speziell für das neu entwickelte CompuLAB-USB wurde ein neuer Treiber benötigt. Diese Aufgabe erfordert einige Vorkenntnisse in C-Programmierung. Als Werkzeuge benötigt man Visual C++ und das Driver Development Kit für Windows98 (DDK98) von Microsoft.

Bei der Entwicklung eines Treibers konnte ich auf Vorarbeiten und Hilfen von Craig Peacock und Hans-Joachim Berndt zurückgreifen. Beiden sei an dieser Stelle herzlich gedankt.

7.1 Umbau eines Beispiels-Treibers

In der DDK89 findet man Beispiele zu USB-Treibern. Am einfachsten ist es, von einem bestehenden Beispiel auszugehen und nur die entscheidenden Stellen abzuändern. Craig Peacock (www.beyondlogic.org) schreib einen eigenen Treiber durch Umbau des ISOUSB-Beispiels zum Intel-Microcontroller 82930 aus der DDK98. Dieser Treiber ist auf der CD enthalten. Er ist noch speziell auf das USB-Thermometer zugeschnitten, erlaubt aber bereits Vendor-Commands mit einer Übertragung von 8 Bits. Der Treiber Beyond.sys lässt sich mit der Datei CypressUSB.INF laden und ersetzt den ursprünglichen Treiber von Cypress.

Die erste veränderte Datei ist das Hauptmodul ISOUSB.C. Hier wird unter anderem ein symbolischer Dateiname für den Treiber aufgebaut, der später mit CreateFile verwendet wird. Wegen der gewünschten Kompatibilität zum Treiber von Cypress wurde der Name "Thermometer_0" verwendet. Hier kann leicht ein anderer Name eingesetzt werden.

```
/*++
Copyright (c) 1997-1998  Microsoft Corporation
Module Name:
    IsoUsb.c

....

{
    NTSTATUS ntStatus;
    UNICODE_STRING deviceLinkUnicodeString;
    PDEVICE_EXTENSION deviceExtension;
    USHORT i;

    UNICODE_STRING pdoUniCodeName;
    WCHAR         pdoName[] = L"\\Device\\Thermometer_0";

    UNICODE_STRING DeviceLinkUniCodeString;
    WCHAR          DeviceLinkName[] =
       L"\\DosDevices\\Thermometer_0";

    RtlInitUnicodeString (&pdoUniCodeName, pdoName); // Creates
                                                     Unicode Name

    ntStatus = IsoUsb_SymbolicLink( PhysicalDeviceObject,
                &deviceLinkUnicodeString );

    ntStatus = IoCreateDevice (DriverObject, // Driver Object
                sizeof (DEVICE_EXTENSION),   // Size
                &pdoUniCodeName,             // Name
```

```
                FILE_DEVICE_UNKNOWN,
                0,               // No Special Characteristics
                FALSE,
                DeviceObject);

    if (NT_SUCCESS(ntStatus))
        {
             deviceExtension = (PDEVICE_EXTENSION)
                ((*DeviceObject)->DeviceExtension);
        }

    if (!NT_SUCCESS(ntStatus))
        {
             return ntStatus;
        }

RtlInitUnicodeString (&DeviceLinkUniCodeString,
           &DeviceLinkName);
ntStatus = IoCreateSymbolicLink(&DeviceLinkUniCodeString,
             &pdoUniCodeName);

...
```

Listing 7.1 Erzeugen eines symbolischen Dateinamens für den Treiber

Der symbolische Treibername muss auch noch in die Datei IsoPnp.c eingetragen werden (vgl. Listing 7.2).

```
/*++

Copyright (c) 1997-1998  Microsoft Corporation

Module Name:

    IsoPnp.c

Abstract:

    Isochronous USB device driver for Intel 82930 USB test board
    Plug and Play module

...

--*/
{

    PIO_STACK_LOCATION irpStack;
    PDEVICE_EXTENSION deviceExtension;
    NTSTATUS ntStatus = STATUS_SUCCESS;
    NTSTATUS waitStatus;
    PDEVICE_OBJECT stackDeviceObject;
    KEVENT startDeviceEvent;
    UNICODE_STRING DeviceLinkUniCodeString;
    WCHAR          DeviceLinkName[] =
              L"\\DosDevices\\Thermometer_0";



    Irp->IoStatus.Status = STATUS_SUCCESS;
    Irp->IoStatus.Information = 0;
...
```

Listing 7.2 Änderung in ISOPNP.C

Allgemein werden in den beschriebenen Anwendungen nur DeviceIoControl-Zugriffe über Endpoint 0 durchgeführt. Daher betreffen die entscheidenden Änderungen in der Funktionalität des Treibers nur die Datei IOCTLISO.C. Listing 7.3 zeigt die entscheidenden Anpassungen von Craig Peacock.

```
/*++
Module Name:
```

```
   ioctliso.c
--*/

NTSTATUS
IsoUsb_ProcessIOCTL(
    IN PDEVICE_OBJECT DeviceObject,
    IN PIRP Irp
    )
/*++

Routine Description:

    Dispatch table handler for IRP_MJ_DEVICE_CONTROL;
    Handle DeviceIoControl() calls  from User mode

Return Value:

    NT status code

--*/
{
    PIO_STACK_LOCATION irpStack;
    PVOID ioBuffer;
    ULONG inputBufferLength;
    ULONG outputBufferLength;
    PDEVICE_EXTENSION deviceExtension;
    ULONG ioControlCode;
    NTSTATUS ntStatus;
    ULONG length;
    PUCHAR pch;
    PURB urb;
    USHORT temp;
    USHORT temp2;
    PUSB_CONFIGURATION_DESCRIPTOR configurationDescriptor;

    IsoUsb_IncrementIoCount(DeviceObject);

    deviceExtension = DeviceObject->DeviceExtension;

    if ( !IsoUsb_CanAcceptIoRequests( DeviceObject ) )
     {
        ntStatus = STATUS_DELETE_PENDING;
        Irp->IoStatus.Status = ntStatus;
        Irp->IoStatus.Information = 0;

        IoCompleteRequest( Irp, IO_NO_INCREMENT );

        IsoUsb_DecrementIoCount(DeviceObject);
        return ntStatus;
     }

    irpStack = IoGetCurrentIrpStackLocation (Irp);

    Irp->IoStatus.Status = STATUS_SUCCESS;
    Irp->IoStatus.Information = 0;

    ioBuffer           = Irp->AssociatedIrp.SystemBuffer;
    inputBufferLength  =
      irpStack->Parameters.DeviceIoControl.InputBufferLength;
    outputBufferLength =
      irpStack->Parameters.DeviceIoControl.OutputBufferLength;

    ioControlCode =
      irpStack->Parameters.DeviceIoControl.IoControlCode;

    switch (ioControlCode)
    {
     case 4: //Implemented for Compatability with Cypress
                     Thermometer Driver.
       pch = (PUCHAR) ioBuffer;
       switch(pch[0])
        {
         case 0x0E: // Set LED Brightness
                    temp = pch[1];       // Store Brightness
                    length = VendorCommand(DeviceObject,
                                0x03, // WriteRAM
                                0x2C, // gbLEDBrightness
                                (USHORT)pch[1],
```

```
                    ioBuffer);
             length = VendorCommand(DeviceObject,
                    0x03, // WriteRAM
                    0x2B, // gbLEDBrightnessUpdate
                    0x01, // TRUE
                    ioBuffer);
            pch[0] = 0;        //Status
           pch[1] = temp;     //Brightness
           Irp->IoStatus.Information = 2;
            Irp->IoStatus.Status = STATUS_SUCCESS;

           ntStatus = STATUS_SUCCESS;
           break;

case 0x0B: // Read Thermometer
           length = VendorCommand(DeviceObject,
                    0x02, // ReadRAM
                    0x33, // gbThermTempRead
                    0,
                    ioBuffer);
           temp = pch[1]; // Store Temperature
            length = VendorCommand(DeviceObject,
                    0x02, // ReadRAM
                    0x34, // gbThermTempRead2
                    0,
                    ioBuffer);
           temp2 = pch[1]; // Store Sign
            length = VendorCommand(DeviceObject,
                    0x02, // ReadRAM
                    0x7A, // gbButtonPushed
                    0,
                    ioBuffer);
           pch[3] = pch[1]; //Move Button
           pch[0] = 0;      //Status
           pch[1] = temp;   //Temperature
           pch[2] = temp2;  //Sign
           Irp->IoStatus.Information = 4;
            Irp->IoStatus.Status = STATUS_SUCCESS;
           ntStatus = STATUS_SUCCESS;
           break;

case 0x14: // Read Port
            length = VendorCommand(DeviceObject,
                    0x04,   // ReadPort
                    pch[1], // Address
                    0,
                    ioBuffer);
           pch[0] = 0;      //Status
           Irp->IoStatus.Information = 2;
            Irp->IoStatus.Status = STATUS_SUCCESS;
           ntStatus = STATUS_SUCCESS;
           break;

case 0x15: // Write Port
            length = VendorCommand(DeviceObject,
                       0x05,   // WritePort
                       pch[1], // Address
                       ioBuffer);
           Irp->IoStatus.Information = 1;
           Irp->IoStatus.Status = STATUS_SUCCESS;
           ntStatus = STATUS_SUCCESS;
           break;

case 0x16: // Read RAM
           length = VendorCommand(DeviceObject,
                    0x02,   // ReadRAM
                    pch[1], // Address
                    0,
                    ioBuffer);
           pch[0] = 0;      //Status
           Irp->IoStatus.Information = 2;
           Irp->IoStatus.Status = STATUS_SUCCESS;
           ntStatus = STATUS_SUCCESS;
           break;

case 0x17: // Write RAM

            length = VendorCommand(DeviceObject,
```

```
                                0x03,   // WriteRAM
                                pch[1], // Address
                                pch[2], // Value
                                ioBuffer);
                    Irp->IoStatus.Information = 1;
                    Irp->IoStatus.Status = STATUS_SUCCESS;
                    ntStatus = STATUS_SUCCESS;
                    break;

          case 0x18: // Read ROM
                     length = VendorCommand(DeviceObject,
                                0x01,   // ReadROM
                                pch[1], // Address
                                0,
                                ioBuffer);
                    pch[0] = 0;      //Status
                    Irp->IoStatus.Information = 2;
                    Irp->IoStatus.Status = STATUS_SUCCESS;
                    ntStatus = STATUS_SUCCESS;
                    break;

          default  :
                    break;
          ntStatus = STATUS_SUCCESS;
          }
         break;

      case 8: //Reads a Vendor Command

        pch = (PUCHAR) ioBuffer;
        length = VendorCommand(DeviceObject, (UCHAR)pch[0],
                (USHORT)pch[1], (USHORT)pch[2], ioBuffer);
         Irp->IoStatus.Information = length;
         Irp->IoStatus.Status = STATUS_SUCCESS;
         ntStatus = STATUS_SUCCESS;
        break;

      case 12: // GetDescriptor(s)
         pch = (PUCHAR) ioBuffer;
         length = GetDescriptor(DeviceObject, (UCHAR)pch[0],
          (UCHAR)pch[1], (USHORT)pch[2], ioBuffer);
        Irp->IoStatus.Information = length;
         Irp->IoStatus.Status = STATUS_SUCCESS;
         ntStatus = STATUS_SUCCESS;
        break;

      case 16: // Get Status
         pch = (PUCHAR) ioBuffer;
        length = GetStatus(DeviceObject, (USHORT)pch[0],
               (USHORT)pch[1], ioBuffer);
        Irp->IoStatus.Information = length;
         Irp->IoStatus.Status = STATUS_SUCCESS;
         ntStatus = STATUS_SUCCESS;
        break;

....
....

         default:
         ntStatus = STATUS_INVALID_PARAMETER;
         Irp->IoStatus.Status = STATUS_INVALID_PARAMETER;
    }

    IoCompleteRequest (Irp,
                       IO_NO_INCREMENT
                       );

    IsoUsb_DecrementIoCount(DeviceObject);

    return ntStatus;

}
...
```

Listing 7.3 Die Bearbeitung von DeviceIoControl-Requests

Unter DeviceIoControlCode=4 werden alle Funktionen des ursprünglichen Treibers von Cypress nachgebildet. Man kann also weiterhin mit der ursprünglichen Firmware die Temperatur auslesen, die LED-Helligkeit verstellen und Schreib- und Lesezugriffe auf das RAM und die Ports ausführen. Die Wahl des Control-Code 4 entspricht nicht den empfohlenen Konventionen (vgl. Kap 2.9), die Funktionsnummern unter 800h für Microsoft reservieren. Sie war aber hier aus Gründen der Kompatibilität zur Thermometer-Applikation erforderlich.

Zusätzlich hat Craig Peacock mit DeviceIoControlCode=8...28 weitere Funktionen eingebaut, mit denen neue Zugriffe möglich werden. Dazu gehört z.B. die Abfrage von Deskriptoren, was bei der Entwicklung hilfreich sein kann. Besonders wichtig ist aber hier die Treiber-Funktion mit DeviceIoControlCode=8 für allgemeine und beliebige Vendor-Requests. Hier werden alle acht übertragenen Bytes unverändert zurückgegeben, so dass man die maximal mögliche Datenmenge in einem Zugriff übertragen kann. Diese Möglichkeit wird im CompuLAB-USB verwendet, um digitale und analoge Eingänge zusammen abzufragen.

Der vorläufig endgültige CompuLAB-USB-Treiber wurde auf der Basis des veränderten ISOUSB-Treibers nach dem Beispiel von Craig Peacock realisiert. Die entscheidende Änderung betraf nur ISOUSB.C und ISOPNP.C. Hier wurde der veränderte Treiber-Name "CompuLABusb-0" eingefügt. Der kompilierte Treiber ISOUSB.SYS wurde dann in COMPULAB.SYS umbenannt. Damit er in dieser Form auch geladen werden kann, benötigt man die Date CompuLAB.INF. Beide Dateien sind mit den entscheidenden Quelltexten auf der CD vorhanden.

Der neue Treiber wurde erfolgreich unter Windows98 getestet. Bisher liegen noch keine Erfahrungen mit Windows 2000 vor. Auch besteht bisher noch nicht die Möglichkeit, mehr als ein Gerät gleichzeitig am Bus zu betreiben. Auch hier wird die Entwicklung noch weiter gehen.

7.2 Anpassung der Firmware

Ein Firmen-eigener Treiber kann nur geladen werden, wenn das entwickelte USB-Gerät über eine entsprechende INF-Datei mit dem Treiber verknüpft wird. Die Zuordnung erfolgt über die Vendor-ID und die Product-ID des Geräts. Jede Firma kann eine VID bei der USB-Organisation beantragen. Die PID wird Firmen-intern vergeben. Für das CompuLAB-USB der Firma Modul-Bus lauten die beiden ID-Numern:

VID = $0A2C = 2604 (AK-Modul-Bus)
PID = $0002 = 2 (CompuLAB-USB)

Mit einer kleinen Änderung der Firmware werden diese IDs in den Device-Deskriptor eingetragen. Bei der Enumeration des Geräts werden die IDs von System gelesen, um den passenden Treiber zu laden.

```
USBDeviceDescription:
    db 12h          ; Length
    db 01h          ; Type (1=device)
    db 00h,01h      ; Complies to USB Spec. v1.00
    db 00h          ; Class code (0=??)
    db 00h          ; SubClass code (0=??)
    db 00h          ; Protocol (0=none)(9.6.1)
    db 08h          ; Max. packet size for port0
    db 2Ch,0Ah      ; Vendor ID: (0x0A2C=Modul-Bus)
```

```
    db 02h,00h      ; Product ID (0x02=USB CompuLAB)
    db 02h,00h      ; Device release v0.20
    db 01h          ; Manufacturer string descriptor index
    db 02h          ; Product string descriptor index
    db 00h          ; Serial number string descriptor index
    db 01h          ; Number of possible configurations
USBDeviceDescriptionEnd:
```

Listing 7.4 Der veränderte USB-Device-Deskriptor

Die veränderte Firmware befindet sich in der Datei CLUSB02.asm auf der CD. Auch hier ist die Entwicklung noch nicht abgeschlossen. Insbesondere befinden sich noch String-Deskriptoren der ursprünglichen Thermometer-Applikation im Programm. Auch gibt es Abschnitte in der Software, die nicht mehr wirklich verwendet werden. Die Firmware ist jedoch brauchbar, um ein eigenes CompuLAB-USB aufzubauen. Die Entwicklung der Firmware wie auch des Treibers wird nach Fertigstellung des Buchs fortgesetzt. Das mittelfristige Ziel ist die Verwendung auch mit Windows 2000 und der gemeinsame Betrieb mehrerer Geräte.

7.3 Die INF-Datei

Die erforderliche INF-Datei muss unter anderem der Treibernamen COMPULAB.SYS und die verwendeten IDs enthalten. Der genaue Aufbau kann stark variieren. Beispiele für INF-Dateien findet man in der DDK98. Die hier verwendete Datei ist von der Datei ISOUSB.INF abgeleitet. Informationen zum genauen Aufbau einer INF-Datei findet man auch im Internet auf der Microsoft-Seite.

Die Datei enthält einige obligatorische und verschiedene optionale Abschnitte. Der Abschnitt [Version] enthält einige Angaben zum Betriebssystem. [Manufacturer] gibt den Hersteller der Hardware an. In jedem Abschnitt kann ein neuer Name definiert werden, unter dem ein neuer Abschnitt geführt wird. Im der Datei CompuLAB.INF wird entsprechend unter [MODULBUS] der Manufacturer-Eintrag zur Vendor-und Product-ID geführt. Der zugeordnete Treiber ist COMPULAB.SYS.

```
[Version]
Signature="$CHICAGO$"
Class=USB
provider=%MB%
LayoutFile=layout.inf

[Manufacturer]
%MfgName%=MODULBUS

[MODULBUS]
%USB\VID_0A2C&PID_0002.DeviceDesc%=COMPULAB.Dev, USB\VID_0A2C&PID_0002

[PreCopySection]
HKR,,NoSetupUI,,1

[DestinationDirs]
COMPULAB.Files.Ext = 10,System32\Drivers
COMPULAB.Files.Inf = 10,INF

[COMPULAB.Dev]
CopyFiles=COMPULAB.Files.Ext, COMPULAB.Files.Inf
AddReg=COMPULAB.AddReg

[COMPULAB.Dev.NT]
CopyFiles=COMPULAB.Files.Ext, COMPULAB.Files.Inf
AddReg=COMPULAB.AddReg

[COMPULAB.Dev.NT.Services]
Addservice = COMPULAB, 0x00000002, COMPULAB.AddService

[COMPULAB.AddService]
DisplayName    = %COMPULAB.SvcDesc%
```

```
ServiceType    = 1                  ; SERVICE_KERNEL_DRIVER
StartType      = 2                  ; SERVICE_AUTO_START
ErrorControl   = 1                  ; SERVICE_ERROR_NORMAL
ServiceBinary  = %10%\System32\Drivers\COMPULAB.sys
LoadOrderGroup = Base

[COMPULAB.AddReg]
HKR,,DevLoader,,*ntkern
HKR,,NTMPDriver,,COMPULAB.sys
HKLM,"System\Currentcontrolset\Services\COMPULAB\Parameters",
   "MaximumTransferSize",0x10001,256
HKLM,"System\Currentcontrolset\Services\COMPULAB\Parameters",
   "DebugLevel",0x10001,2


[COMPULAB.Files.Ext]
COMPULAB.sys

[COMPULAB.Files.Inf]
COMPULAB.Inf


;------------------------------------------------------------;

[Strings]
MB="AK MODUL-BUS GmbH"
MfgName="MODULBUS"
USB\VID_0A2C&PID_0002.DeviceDesc="Modul-Bus GmbH, CompuLAB-USB"
COMPULAB.SvcDesc="CompuLAB-USB Driver"
```

Listing 7.5 Die Datei COMPULAB.INF

Anders als in der ursprünglichen INF-Datei von Cypress wird hier keine eigene Geräte-Klasse (Thermometer) gebildet, sondern das neue Gerät wird unter der bestehenden Klasse USB eingeordnet.

Beim ersten Verbinden des USB-CompuLAB erscheint eine PlugAndPlay-Meldung "Neues Gerät gefunden". Der Anwender wird aufgefordert, eine Diskette mit Treiber-Informationen einzulegen. Das System findet dann die INF-Datei und die zugehörige Treiberdatei COMPULAB.SYS. Im Geräte-Manager kann man das neue Gerät finden.

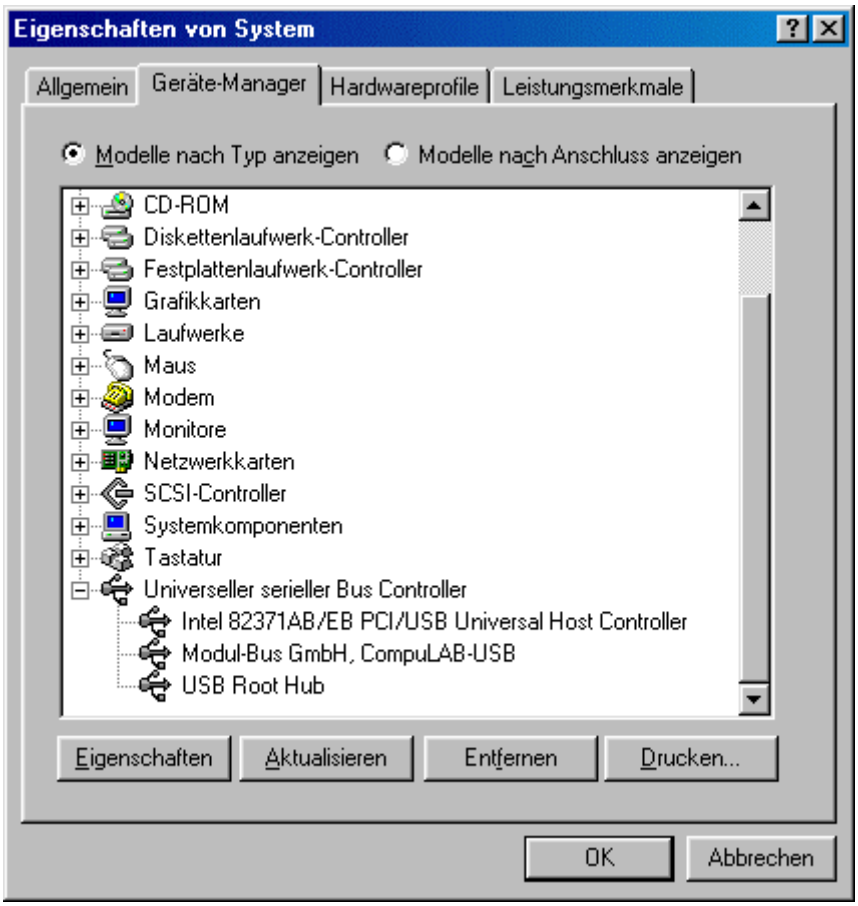

Abb. 7.1 Das enumerierte Gerät im Geräte-Manager