B. Kainka

SIOS-Programmierung mit Delphi

Wer Windows-Programme schreiben will, aber die langwierige Einarbeitung in C++ vermeiden möchte, für den stellt Delphi eine echte Alternative dar, insbesondere dann, wenn schon Erfahrungen mit Turbo Pascal vorliegen. Bei der Ansteuerung von Interfaces und allgemein der Verwendung von Schnittstellen ist jedoch einiges zu beachten.

Das SIOS-Interface wird über die serielle Schnittstelle mit einzelnen Bytes angesteuert. Ein Hardware-Handshake ist nicht vorgesehen, so dass man nur die Leitungen TXD und RXD berücksichtigen muß. Auf der Software-Seite genügt eine Prozedur SendByte und eine Funktion ReadByte, jeweils zur Übertragung einzelner Bytes. Außerdem muß eine Prozedur zum Öffnen der Schnittstelle mit den vorgegebenen Parametern (19200,N,8,1) vorhanden sein. Prinzipiell kann man in Delphi Windows-API-Aufrufe zur Ansteuerung der Schnittstellen nutzen, dies ist jedoch nicht ganz einfach.

Ein leichterer Weg führt über die Verwendung einer DLL. Hier soll die RSAPI.DLL aus dem Buch „Messen, Steuern, Regeln mit Word und Excel“ von Berndt/Kainka verwendet werden. Die folgenden Ausführungen beziehen sich auf die 32-Bit-Version der DLL und Delphi 4 unter Windows95/98. Die DLL bietet drei verschiedene Initialisierungs-Funktionen für allgemeine Zugriffe auf die RS232, für ein bestimmtes Interface wie CompuLAB, SIOS oder ZELLE und schließlich eine automatische Suchfunktion, die ein bekanntes Interface und die verwendete Schnittstelle findet.

1. Einbinden der DLL

Das erste Programmbeispiel soll das SIOS auf der untersten Ebene mit einzelnen Bytes ansteuern. Man braucht dazu nur OpenCOM, SendByte und ReadByte. Jede Prozedur oder Funktion einer DLL muß zunächst deklariert werden. Die Deklaration erfolgt in der Unit SIOSDLL.DCU und folgt im Aufbau einer normalen Prozedur oder Funktion, wobei mit EXTERNAL auf die DLL verwiesen wird. Die Deklarationen unterscheiden sich etwas zwischen den Programmiersprachen. Deshalb sollen hier die wichtigsten Einzelheiten genannt werden. Das zusätzliche Schlüsselwort STDCALL sorgt für die korrekte Übergabe von Parametern.

```
interface
const THEDLL='RSAPI.DLL';

function INITSIOS (COM: WORD):Integer;
         stdcall external THEDLL;
function INITCOMPULAB (COM: WORD):Integer;
         safecall  external THEDLL;
function FINDHARD( i:Integer):integer; stdcall; external THEDLL;
function AIN(i:Integer):integer;  stdcall; external THEDLL;
procedure DOUT(d:WORD);
         stdcall external THEDLL;
function DIN:Integer;
         stdcall external THEDLL;
function OPENCOM( s:AnsiString):integer; stdcall; external THEDLL;
function CLOSECOM():Integer;
         stdcall external THEDLL;
function READBYTE: integer; stdcall; external THEDLL;
function SENDBYTE(i:integer): integer; stdcall; external THEDLL;
function DELAY(i:integer): integer; stdcall; external THEDLL;
procedure TIMEINIT; stdcall; external THEDLL;
function TIMEREAD: longint; stdcall; external THEDLL;


implementation

end.
```

Listing 1 Quelltext der Unit SIOSDLL

Die RSAPI.DLL wurde in C geschrieben. Die Übergabe von Parametern aus Delphi ist nicht ganz einfach. Besondere Probleme macht die Übergabe von Textstrings. Nach einigen Versuchen wurde jedoch der richtige Weg gefunden: Bei der Einbindung der DLL-Funktion OPENCOM wird der Übergabetext als AnsiString deklariert. Beim eigentlichen Aufruf der Funktion muss der übergebene Text mit PCHAR in einen nullterminierten String nach C-Konvention umgewandelt werden.

```
unit SIOS1;
interface
uses  SIOSDLL,
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ExtCtrls, StdCtrls;
type
  TForm1 = class(TForm)
    Edit1: TEdit;
    Timer1: TTimer;
    procedure FormCreate(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
  end;
var
  Form1: TForm1;

implementation
{$R *.DFM}
procedure TForm1.FormCreate(Sender: TObject);
begin
 OpenCom(Pchar('com2:19200,N,8,1'));
  SendByte (16);
  SendByte (85);
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  SendByte (48);
  Edit1.Text:= (FloatToStr (ReadByte));
end;

end.
```

Listing 2 Ein Beispielprogramm mit OpenCom

Im Beispiel nach Listing 2 wird zunächst die allgemeine OPENCOM-Funktion benutzt. Die Schnittstelle COM2 wird beim Start des Programms mit den erforderlichen Parametern initialisiert. Ein erster Test erfolgt mit einem Ausgabekommando an das SIOS-Interaface. Bei der Initialisierung wird ein Bitmuster an die digitalen Ausgänge gesandt. Am SIOS sieht man jede zweit LED angeschaltet und hat so bereits eine Rückmeldung über die erfolgreiche Initialisierung. Über eine Timerprozedur wird dann ein Analogwert vom Eingang A laufend abgefragt und in einem Ausgabefenster angezeigt.

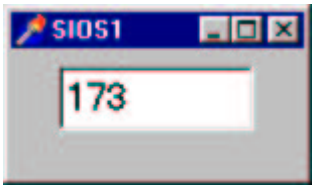

Abb. 1 Messwertausgabe von Analogeingang 1

2. Interface-Spezifische Initialisierung

Ein zweiter Versuch verwendet die spezielle Initialisierungsfunktion INITSIOS. Dabei wird zum einen die Schnittstelle geöffnet und zum anderen ein Satz spezifischer Kommandos für das SIOS aktiviert. In Listing 1

wurden bereits die Funktionen DOUT, DIN, AOUT und AIN deklariert. In Listing 2 wird nun einen digitale Ausgabe mit DOUT durchgeführt und über AIN ein Analogwert gelesen.

```
procedure TForm1.FormCreate(Sender: TObject);
var n: Integer;
begin
  n := InitSIOS(2);  {COM2}
  Dout (15);
  Edit1.Text := FloatToStr (Ain(1));
end;
```

Listing 3 Interface-spezifische Funktionen

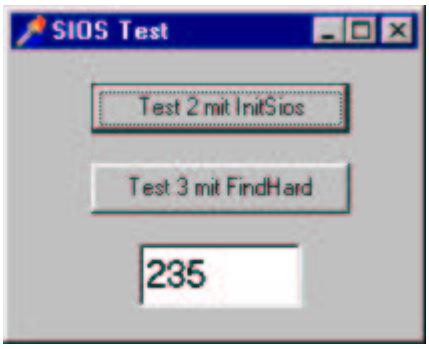

Abb. 3 Programmtest mit InitSios und FindHard

In einem dritten Schritt soll die allgemeine Initialisierung mit FINDHARD erprobt werden. Mit einem Übergabeparameter „1“ zeigt die Funktion das gefundene Interface und die verwendete Schnittstelle in einer Messagebox an. Das Programm funktioniert ohne Änderung für das CompuLAB, die MiniRS-Box, das SIOS und die ZELLE. Hier soll laufend ein Meßwert des ersten analogen Eingangskanals in einem Textfenster angezeigt werden. Die Messung und Ausgabe erfolgt daher in einer Timer-Prozedur.

```
procedure TForm1.FormCreate(Sender: TObject);
var n: Integer;
begin
  n := FindHard (1);
  DOUT (15);
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Edit1.Text := FloatToStr (Ain(1));
end;
```

Listing 4 Verwendung von FINDHARD

Abb. 3 Meldung des erkannten Interfaces

Abbildung 3 zeigt das Meldungsfenster der Funktion FindHard. Mit dieser Funktion kann nun mit gleichem Erfolg auch die ZELLE, das CompuLab oder die MiniRS-Box eingesetzt werden. Die DLL merkt sich intern das gefundene Interface und verwendet automatisch die passenden Komandobytes.

Das gesamte Projekt unter dem Namen SIOSTEST kann übrigens auch über www.modul-bus.de geladen werden.

3. Zeitprobleme unter Windows

Windows ist nicht echtzeitfähig. Insbesondere seit Windows95 führt das verwendete Multitasking-Verfahren dazu, dass man sich nicht mehr auf einen bestimmten Zeitablauf verlassen kann. Dieses Problem tritt z.B. bei der Steuerung von Schrittmotoren auf. Einige kleine Vorsichtsmaßnahmen führen jedoch trotzdem zu brauchbaren Ergebnissen.

Hier soll ein Schrittmoor in einem älteren Aufzug-Modell gesteuert werden. Modul-Bus hat zwei Varianten eines Aufzugs gebaut. Die ganz alte Version verwendete eine Anschlußbelegung, die in dem Buch „Modellversuche – Teil 1“ von J.Hüvelmeyer verwendet wurde. Danach wurde die Anschlussbelegung des Schrittmotors und der Schalter geändert. Die meisten der noch in den Schulen vorhandenen Aufzugsmodelle dürften mit dem hier vorgestellten Programm korrekt arbeiten. Für Neuanschaffungen wird das Modell nicht mehr empfohlen.

Funktionsmodelle wie der Aufzug wurden in den Schulen häufig für Programmierübungen in Turbo Pascal eingesetzt. Da nun die Zeit der Programmierung unter DOS fast überall abgelaufen ist, sucht man nach neuen Wegen. In vielen Schulen hat sich Delphi bewährt. Daher erscheint es sinnvoll, auch ältere Modelle vom Staub zu befreien und unter Delphi wieder in Dienst zu stellen.

Listing 5 zeigt eine erste Lösung. Die Schnittstelle wird mit FindHard geöffnet, so dass von vornherein das SIOS, das CompuLAB mit der PowerBox und die die MiniRS-Box gleichberechtigt verwendet werden können. Die gültigen Bitmuster zur Steuerung des Schrittmotors werden beim Start des Programms in einem Array abgelegt. Die eigentliche Ausgabe erfolgt in einer Timerprozedur. Der verwendete Timer in Delphi arbeitet bis herab zu Intervallen von 20 ms. Man kann also bis zu 50 Schritte in einer Sekunde erreichen. Die Schrittgeschwindigkeit wird global durch einen Schieberegler vorgenommen.

Ein zweiter Schieberegler gestattet eine Feineinstellung der Schritte zwischen den Stockwerken. In der Praxis hängt der korrekte Wert etwas von der Motorachse und dem verwendeten Seilzug ab. Das Programm wird komplett über die Tastschalter im Modell gesteuert, mit denen man den Aufzug in jedes Stockwerk schicken kann.

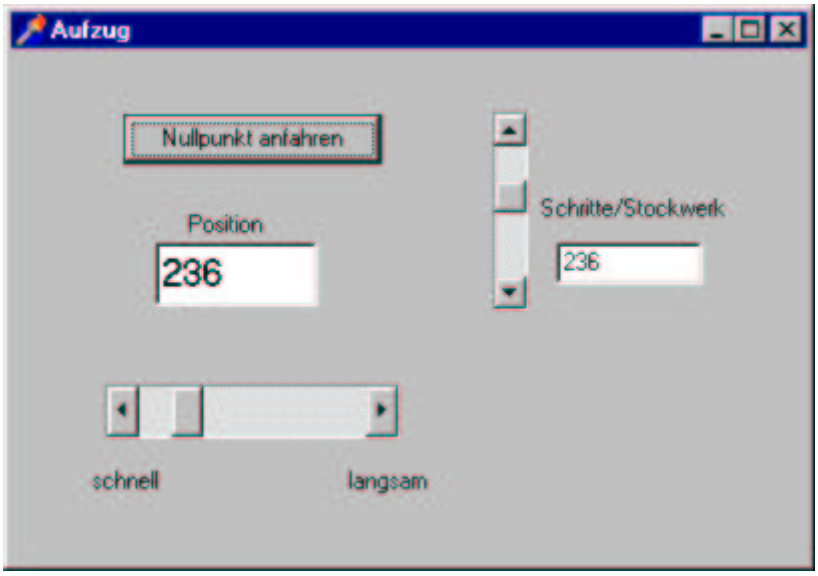

Abb. 3 Das Aufzug-Steuerprogramm

```
unit Aufzug;

interface

uses SIOSDLL,
```

```
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ExtCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    Edit1: TEdit;
    Timer1: TTimer;
    Position: TLabel;
    ScrollBar1: TScrollBar;
    Label1: TLabel;
    Label2: TLabel;
    ScrollBar2: TScrollBar;
    Edit2: TEdit;
    Label3: TLabel;
    procedure Button1Click(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure ScrollBar1Change(Sender: TObject);
    procedure ScrollBar2Change(Sender: TObject);
  end;

var
  Form1: TForm1;
  StepSoll, StepIst, Muster, Anzeige: Integer;
  Phase, Stockwerk: Integer;
  Step : Array  [0..3] of Integer;

implementation

{$R *.DFM}


procedure TForm1.Button1Click(Sender: TObject);
var n: Integer;
begin
     StepSoll := -300;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
var Inp: Integer;
begin
  Inp := Din;
  If (Inp and 1) = 1 then begin
     StepIst := 0;
     StepSoll := 0
  end;
  If (Inp and 2) =2 then begin    {T1}
     StepSoll := 0;
     Anzeige := 128;
  end;
  If (Inp and 4) =4 then begin    {T2}
     StepSoll := Stockwerk;
     Anzeige := 128;
  end;
  If (Inp and 8) =8 then begin    {T3}
     StepSoll := Stockwerk * 2;
     Anzeige := 128;
  end;
  If (Inp and 16) =16 then begin  {T1, Fahrkorb}
     StepSoll := 0;
     Anzeige := 16 + 128;
  end;
```

```
  If (Inp and 32) =32 then begin  {T2, Fahrkorb}
     StepSoll := Stockwerk;
     Anzeige := 32 + 128;
  end;
  If (Inp and 64) =64 then begin  {T2, Fahrkorb}
     StepSoll := Stockwerk;
     Anzeige := 32 + 128;
  end;
  If (Inp and 128) =128 then begin  {T3, Fahrkorb}
     StepSoll := Stockwerk * 2;
     Anzeige := 64 + 128;
   end;

  if StepSoll > StepIst then begin;
    StepIst := StepIst + 1;
    Phase := (StepIst mod 4) and 3;
    Dout (Step[Phase] + Anzeige);
  end;
  if StepSoll < StepIst then begin;
    StepIst := StepIst - 1;
    Phase := (StepIst mod 4) and 3;
    Dout (Step[Phase] + Anzeige);
  end;
  if StepIst > -1 then Edit1.Text := FloatToStr (StepIst);
  if StepIst = StepSoll  then begin
     Anzeige :=  0;
     Dout (0);
   end;
end;

procedure TForm1.FormCreate(Sender: TObject);
var n: Integer;
begin
  n := FINDHARD (1); {Ergebnisanzeige im Fenster}
  Step[0] := 1;
  Step[1] := 2;
  Step[2] := 4;
  Step[3] := 8;
  Timer1.Enabled := true;
  StepSoll := 0;
  StepIst := 0;
  Phase := 0;
  Anzeige := 0;
  Stockwerk := 256;
end;

procedure TForm1.ScrollBar1Change(Sender: TObject);
begin
   Timer1.Interval := ScrollBar1.Position;
end;

procedure TForm1.ScrollBar2Change(Sender: TObject);
begin
  Stockwerk := Scrollbar2.Position;
  Edit2.Text := FloatToStr (Stockwerk);
end;

end.
```

Listing 5 Der erste Versuch zum Aufzug

Der erste Versuch zur Steuerung des Aufzugs ist eher enttäuschend: Der Motor rattert und ruckelt so vor sich hin. Mit diesem Aufzug möchte wirklich niemand fahren. Was ist hier falsch gelaufen? Es liegt am Timer! Windows kann einfach keinen gleichmäßigen Aufruf der Timerprozedur garantieren. Wenn irgendwelche wichtigen Prozesse vorhanden sind, dauert es einfach mal etwas länger. Man kann solche Prozesse auslösen, wenn man schnelle Bewegungen mit der Maus ausführt. Windows gerät dann in Zeitnöte und das Geruckel wird noch schlimmer.

Natürlich gibt es auch eine Lösung für das Problem. Solange eine Delphi-Prozedur nämlich noch läuft, hat sie eine sehr hohe Priorität und wird nur noch durch wenige, sehr schnelle Vorgänge unterbrochen. Die Bewegung des Mauszeigers wird weiterhin garantiert, ein Mausklick dagegen bleibt wirkungslos. Daher kann ein fehlerhaftes Programm auch zu einem „Hängen“ des Systems führen, das nur noch durch den „Affengriff“ Alt-Strg-Entf beendet werden kann.

Man muss also die vollständige Bewegung des Schrittmotors durchführen ohne die Prozedur zu verlassen. Listing 6 zeigt die entscheidende Änderung. Sobald sich der Aufzug auf den Weg zu einem neuen Stockwerk macht, werden alle erforderlichen Schritte in einer Schleife ausgeführt. Die Zeit zwischen den einzelnen Schritten wird nicht mehr durch die Timerprozedur festgelegt, sondern mit der DELAY-Prozedur aus der RSAPI.DLL. Alte Pascal-Programmierer werden sich freuen, dass Delay nun auch in Delphi wieder verwendbar ist. Außerdem lassen sich nun Verzögerungen bis herunter zu einer Millisekunde verwenden, so dass theoretisch bis zu 1000 Schritte pro Sekunde möglich werden. Der Aufzug wird wesentlich schneller.

```
if StepSoll > StepIst then begin;
    repeat
       StepIst := StepIst + 1;
       Phase := (StepIst mod 4) and 3;
       Dout (Step[Phase] + Anzeige);
       Delay (DelayZeit);
    until StepSoll= StepIst;
  end;
  if StepSoll < StepIst then begin;
    repeat
       StepIst := StepIst - 1;
       Phase := (StepIst mod 4) and 3;
       Dout (Step[Phase] + Anzeige);
       Delay (DelayZeit);
       Inp := Din;
       If (Inp and 1) = 1 then begin
         StepIst := 0;
         StepSoll := 0
       end;
    until StepSoll= StepIst;
   end;
```

Listing 6 Die veränderte Motorsteuerung

Das ganze Aufzug-Projekt mit beiden Varianten kann über das Internet geladen werden. Es wäre schön, wenn weitere Beispielprogramme von anderer Seite zu einer kleinen Sammlung führen würde. Alle Schulen sollten daran gemeinsam mitarbeiten, um sich insgesamt gegenseitig zu helfen.

Es gibt übrigens Vorgänge, die noch wesentlich schneller ablaufen müssen als die Steuerung eines Schrittmotors. Man kann z.B. in Delphi einen Software-Frequenzzähler für reale Messungen bei Frequenzen bis zu 50 kHz realisieren. Solche Lösungen werden in dem Buch „PC-Schnittstellen unter Windows“ im Elektor-Verlag beschrieben. Die Priorität eines eigenen Prozesses kann durch eine zusätzliche DLL-Prozedur „RealTime“ vorübergehend weiter erhöht werden, so dass man zeitliche Konstanz bis in den Bereich von Mikrosekunden erzielen kann.