fischertechnik Interface

umFish.DLL

Handbuch zu Version 1.3.5

Ulrich Müller

Inhaltsverzeichnis

Dokumentname : umFish13.odt. / Fisch6.WMF Druckdatum : 24.05.2000

umFish.DLL

Allgemeines

umFish.DLL ist eine in VC++ 6.0 geschriebene 32bit DLL, die Zugriffe auf die fischertechnik-Interfaces 30520 (einschl. CVK fischertechnik Interface von Cornelsen) und 30566 (parallel) und 30402 (seriell und Extension Module 16554) – beide auch im Master-Slave-Betrieb - ermöglicht. Dazu werden eine Reihe von Basisfunktionen angeboten, die aus den Programmiersprachen Visual Basic, C/C++ und Delphi genutzt werden können.

umFish.DLL ist unter den Betriebssystemen Windows95/98/Me sowie Windows NT 4.0 ab SP3 und Windows 2000 einsetzbar. umFish.DLL kann, wie DLL-üblich, im Pfad der Anwendung oder im Win/System-Verzeichnis liegen, eine Registrierung ist nicht erforderlich. Zum Betrieb eines parallelen Interfaces ist ggf. zusätzlich ein WinRT-Treiber erforderlich.

Ulrich Müller, D-33100 Paderborn, Lange Wenne 18, Fon 05251/56873, Fax 05251/55709
eMail : UM@ftcomputing.de

Homepage : www.ftcomputing.de

Hinweise

Refresh

Die fischertechnik Interfaces benötigen ca. alle 300 mSek ein Refresh des Status der (Motor)Ausgänge. Sonst werden die Ausgänge abgeschaltet, das geschieht um das Modell vor in die Irre gelaufenen Programmen zu schützen. Technisch läßt sich ein Refresh am einfachsten durch den umFish-Befehl GetInputs() realisieren. Es kann aber auch durch jeden anderen umFish-Befehl geschehen. Siehe Programmiertechniken.

Interface Parameter

Zur Ansteuerung des Interface benötigt umFish einen Kontollblock (ftDCB), der beim OpenInterface mit aktuellen und Defaultwerten besetzt wird. Bei ”merkwürdigem” Verhalten des Interfaces (das Interfaceverhalten wird durch die Rechnergeschwindigkeit beeinflußt) können vor oder nach dem Open beim parallelen Interface die Werte LPTDelay bzw. LPTAnalog modifiziert werden (siehe ftDCB). Intern werden den Port(LPT)-Namen Port-Adressen zugeordnet, wenn der aktuelle Rechner nicht die Standardwerte verwendet, sollte dem Open die PortID manuell gesetzt werden.

Unterbrechbarkeit

Wird das Interface in einer engen Schleife angesteuert z. B. Loop zur Abfrage eines Digitaleinganges, ist das Programm nicht mehr unterbrechbar. D.h. es reagiert nicht mehr

auf Control-Buttons und macht meist auch keine Bildschirmanzeige mehr. Hier ist programmtechnisch für eine Unterbrechbarkeit durch Abgabe des Zeitscheibe an das Betriebssytem (VB : DoEvents, Delphi : Application.ProcessMessage, C/C++ : weiß nicht).

Abbrechbarkeit

Endlos-Abfrageschleifen oder auch langlaufende Methoden (Wait ..., MoveTo aus umFishEx.DCU oder FishN632.DLL) können von außen nicht einfach beendet werden, sie laufen ggf. sogar noch, wenn die Programm-Form schon entladen wurde. Außerdem sollte für den Fall eines drohenden Crashes ein NotHalt vorhanden sein. Auf der Form sollte ein Button vorhanden sein, der die Eigenschaft NotHalt = True setzt (daran denken vor jedem ProgrammStart NotHalt = False zu setzen). NotHalt kann dann auch in Endlos-Abrageschleifen ausgewertet werden.

Master-Slave-Betrieb

Sowohl die parallelen (Universal Interface) wie auch das serielle(Intelligent Interface) Interface erlauben einen Master-Slave-Betrieb. Das heißt die Kopplung (siehe Handbuch des jeweiligen Interfaces) eines zweiten (Slave) Interfaces an das erste (an den Rechner angeschlossene Inteface, Master). Beim parallelen Interface sind zum Betrieb des Slave keine besonderen Parameter erforderlich (es läuft intern immer mit), beim seriellen wurde der vorhandene Parameter LPTDelay = 2 (des ftDCB Kontrollblocks) dafür zweckentfremdet. Der Defaultwert von LPTDelay ist 1 (kein Slave)

Besonderheiten Windows NT

Windows NT benötigt zum Betrieb des parallelen Interfaces zusätzlich zur FishFace-Software einen Kernel0-Treiber. In diesem Fall WinRT.SYS (zur Installation siehe Readme). Das Einrichten (Festlegen des LPT-Ports, Start des Treibers) des Treibers muß unter Administratorrechten geschehen. Bei privaten Rechnern ist das normalerweise gegeben, im Netzwerk oder auch in Schulen nicht. In diesen Fällen muß der Treiber so eingerichtet werden, daß er beim Booten automatisch gestartet wird. Mit den umFish.DLL-Funktionen Start-/Stop-Driver kann das (mit Administratorrechten) auch im Anwendungsprogramm geschehen.

Lampen am Interface

Die 4 Digitalausgänge des Interfaces sind primär zum Schalten von Motoren in zwei Laufrichtungen vorgesehen. Doch bietet sich zusätzlich die Möglichkeit, Geräte, die nur ein- und ausgeschaltet werden müssen – wie z. B. Lampen an nur einen Pol eines Digitalausganges und die Erde anzuschließen. Auf diese Weise ist es möglich 8 Lampen mit einem Interface zuschalten. Hierfür gibt es die umFish.DLL-Funktion SetLamp.

Funktionen

ftDCB-Kontrollblock

Die Informationen zum Betrieb des Interfaces werden in einem Kontrollblock gehalten, der vor dem ersten Aufruf von OpenInterface anzulegen ist :

- hCom — Handle zum COM- bzw. LPT-Port, wird bei erfolgreichem Open gesetzt und bei Close auf Null gesetzt, sollte nicht verändert werden.
- PortID — nur LPT, I/O-Adresse des LPT-Ports, von OpenInterface gesetzt, sollte nicht geändert werden
- LPTDelay — Ausgabeverzögerung an dem LPT-Port, von OpenInterface auf den Defaultwert = 10 gesetzt, kann ggf. nach dem Open verändert werden. Schnelle Rechner größerer Wert.
  Wird nur bei Portname LPT1-3 ausgewertet
  Bei Einsatz des **Extension Module 16554** (COM) :
  LPTDelay = 2 setzen um dies anzuzeigen.
- LPTAnalog — Skalierung des Analogwertes, von OpenInterface auf den Defaultwert = 5 gesetzt, kann ggf. nach dem Open verändert werden. Der angezeigte Wertebereich sollte zwischen 0 und 1024 liegen, bei zu kleinen Werten ein größeres LPTAnalog wählen.
  Nur Portname= LPT1-3.
- InputStatus — Status aller Eingänge nach dem letzten Zugriff auf das Interface, bit 0 entspricht Eingang 1, bit 7(15) Eingang 8(16).
- MotorStatus — Status aller (Motor)Ausgänge nach dem letzten Zugriff auf das Interface. Jeweils 2 bit pro Ausgang, das niederwertige bit : "Linkslauf" (ftiLinks), das höherwertige bit "Rechtslauf" (ftiRechts). "Aus" (ftiAus) beide 0.
  Die bits dürfen nicht gleichzeitig gesetzt sein.
  bit 0/1 : Motor 1 - bit 6/7(14/15) : Motor 4(8). MotorStatus muß vor Aufruf von SetMotors (Steuerung aller angeschlossenen Motoren)
  mit entsprechenden Werten besetzt werden. Siehe auch Sonderfall Lampen.

Die Funktionen liefern als Rückgabewert im Erfolgsfall den aktuellen InputStatus, im Fehlerfall den Wert ftiFehler. Ausnahme GetInput : liefert den Status des abgefragten Einganges. ftiFalse(0) : nicht gesetzt, ftiTrue(-1) : gesetzt und GetAnalog : liefert den abgefragten Analogwert.

Allgemein : ftiFehler : allgemeiner Fehler beim Interfacezugriff.

Der Kontrollblock wird im Anwendungsprogramm (sprachspezifisch, siehe Nutzung) als Struktur mit 32bit-Worten dargestellt. Eine simultaner Einsatz mehrerer Interfaces ist möglich, es müssen dann entsprechend viele Kontrollblöcke angelegt und mit OpenInterface besetzt werden.

Alle Funktionen von umFish.DLL erwarten als ersten Parameter die Adresse des Kontrollblocks (Ausnahme Start-/StopDriver).

Sonderfall Lampen.

Geräte, die nur eine Ein-/Ausstatus kennen, können einpolig an einen M-Ausgang (zweiter Pol Mass) angeschlossen werden. Sie können dann über den Befehl SetLamp gesteuert werden. Sollen sie gemeinsam (z. B. bei einer Verkehrsampel) geschaltet werden, können sie auch über MoterStatus und SetMotors geschaltet werden. Dazu sind die entsprechenden einzelnen Lampenbits im MotorStatus vor Aufruf von SetMotors zu setzen.

Es gibt hier Unterschiede zwischen den Interfaces :
Paralleles Interface : Im nicht geschalteten Zustand sind die Lampen aus. M1 vorderer (gelber) Kontakt : Lampe 1, hinterer (orange) Kontakt Lampe 2 ...
Serielles Interface (30402) : im nicht geschalteten Zustand sind die Lampen **an**. M1 vorderer Kontakt : Lampe 2, M1 hinterer Kontakt Lampe 1 ...

Also genau hinsehen. Beim seriellen Interface ergibt sich der etwas irritierende Effekt, dass alle Lampen im Ruhezustand eingeschaltet sind. Ausßerdem ist zu bedenken, dass mit SetMotors stets **alle** M-Ausgänge geschaltet werden.

OpenInterface

[inputstatus =] OpenInterface(ftDCB, PortName)

Öffnen der Verbindung zum Interface, besetzen des Kontrollblocks. PortName (string) : COMx bzw LPT1 - 3. Bei Windows NT / 2000 muß beim parallelen Port LPT angegeben werden. Bei Windows 95/98/Me kann LPT angebeben werden (wenn LPT1-3 nicht "funktioniert"). In beiden Fällen muß dann WinRT.SYS bzw. WRTdev0.VxD installiert sein, siehe Readme.

CloseInterface

[inputstatus =] CloseInterface(ftDCB)

Schließen der Interfaceverbindung, freigeben des Ports.

GetInputs

[inputstatus =] GetInputs(ftDCB)

Abfrage aller (Digital)Eingänge

GetInput

bool = GetInput(ftDCB, InputNr)

Abfrage des ausgewählten Eingangs.

ClearMotors

[inputstatus =] ClearMotors(ftDCB)

Löschen aller Ausgänge.

SetMotors

[inputstatus =] SetMotors(ftDCB)

Setzen aller Ausgänge auf den Wert in ftDCB.MotorStatus. Der MotorStatus muß vor dem Aufruf entsprechend gesetzt werden.

SetMotor

[inputstatus =] SetMotor(ftDCB, MotorNr, Richtung)

Schalten eines Ausganges. Richtung ftiLinks, ftiRechts oder ftiAus.

GetAnalog

analog = GetAnalog(ftDCB, AnalogNr)

Abfrage des ausgewählten Analogeinganges (EX = 0, EY = 1, nicht beim Slave-Interface).

Zur Skalierung der Werte siehe oben : Kontrollblock.

SetLamp

[inputstatus =] SetLamp(ftDCB, LampNr, OnOff)

Schalten eines ”halben” Digitalausganges. Das Gerät (Lampe, Magnet) wird mit einem Pol des Ausganges und der Erde verbunden. LampNr 1 – 8(16) entsprechend M1-M4(8), M1 (rot) ist L1. OnOff = ftiEin bzw. ftiAus

---------- nur Windows NT, paralleles Interface ----------

StartDriver

[res =] StartDriver()

Starten des WinRT.SYS Treibers (nur Windows NT/2000). Die Funktion entspricht der Menü-Funktion
Start | Einstellungen | Systemsteuerung | Geräte | WinRT. WinRT.SYS muß für manuellen Start installiert sein. Es sind Administratorrechte erforderlich.

StopDriver

[res =] StopDriver()

Anhalten des WinRT.SYS Treiber (nur Windows NT/2000). Sonst wie StartDriver.

Defaultmäßig wird WinRT.SYS mit ”Allow Conflicts” installiert. Ein sonst erforderliches Anhalten der normalen (ParPort) Treiber ist dann nicht erforderlich. Die Funktionen sind sinnvoll, wenn man nur gelegentlich mit dem Interface arbeitet und nicht ständig den zusätzlichen Treiber mitlaufenlassen will.

Programmiertechniken

Hier werden Programmiertechniken zur Lösung typischer Aufgabenstellungen weitgehend unabhängig von einer Programmiersprache (aber doch in der Nähe von Visual Basic) skizziert.

Programmrahmen

umFish benötigt pro Interface jeweils einen statischen Kontrollblock (der parallele Betrieb von Interfaces – also nicht Master/Slave – ist problemlos möglich. Das kann bei seriellen Interfaces z. B. über eine Multiportkarte geschehen).

Initialisieren des Interface durch OpenInterface, danach ggf. Anpassung der LPT-Parameter an den Rechner.

Beenden des Interface-Betriebs und Freigabe der Schnittstelle durch CloseInterface.

Der Modellbetrieb selber wird häufig in einem Programmzyklus erfolgen. Er kann aber genausogut über einen Timer gesteuert (Intervalle < 300 mSek) werden. In beiden Fällen ist das erforderliche Refresh (s.o.) gegeben. Man sollte zusätzlich eine NotAus-Funktion vorsehen (z. B. die Abfrage eines Endtaster verbunden mit einem Exit aus dem Zyklus).

Anzeige des Interface-Status

```
Do
  DoEvents
Loop Display GetInputs(ftDCB)
```

Anzeige von Analogwerten

```
Do
  DoEvents
Loop Display GetAnalog(ftDCB,0) // --- Abfrage EX
```

NotHalt/Reset

```
SetMotors(ftDCB)
Do
  ....
  DoEvents
Loop Until GetInput(ftDCB, 8)
ClearMotors(ftDCB)
```

Fahren zum Endtaster

```
SetMotor(ftDCB, m, ftiLinks)
Do
  DoEvents
Loop Until GetInput(ftDCB, d)
SetMotor(ftDCB, m, ftiAus)
```

Anfahren einer Position

Visual Basic Programmierer sollten hier auf FishFace.DLL umsteigen, die weniger glücklichen sollten sich die entsprechende Visual Basic Source (C/C++ -Programmierer vorher kurz Kopfschüttteln, das hilft) ansehen : Päckchen FishFacX.ZIP, Source FishFace.CLS, Routinen : WaitForTime (Warten mit Refresh), WaitForLow(Warten auf

Pegelwechsel), WaitForChange(Warten auf eine vorgegebene Impulszahl an einem Eingang) und MoveTo/MoveDelta (simultaner Betriebe mehrerer Motoren mit Positionshaltung).

Nutzung

Visual Basic

Der CodeModul umFish.BAS mit folgenden Deklarationen sollte in das Projekt eingebunden werden :

```
Public Declare Function OpenInterface Lib "umFish.dll" _
  (ft As ftDCB, ByVal ComName$) As Long
Public Declare Function CloseInterface Lib "umFish.dll" _
  (ft As ftDCB) As Long
Public Declare Function GetInput Lib "umFish.dll" _
  (ft As ftDCB, ByVal InputNr&) As Boolean
Public Declare Function GetInputs Lib "umFish.dll" (ft As ftDCB)
As Long
Public Declare Function ClearMotors Lib "umFish.dll" (ft As
ftDCB) As Long
Public Declare Function SetMotors Lib "umFish.dll" _
  (ft As ftDCB, ByVal MotorStatus&) As Long
Public Declare Function SetMotor Lib "umFish.dll" _
  (ft As ftDCB, ByVal MotorNr&, ByVal Richtung&) As Long
Public Declare Function SetLamp Lib "umFish.dll" _
  (ft As ftDCB, ByVal LampNr&, ByVal OnOff&) As Long
Public Declare Function GetAnalog Lib "umFish.dll" _
  (ft As ftDCB, ByVal AnalogNr&) As Long

Public Declare Function StartDriver Lib "umFish.dll" () As Long
Public Declare Function StopDriver Lib "umFish.dll" () As Long

Public Declare Sub Sleep Lib "kernel32" (ByVal dwMsec&)

Public Type ftDCB
  comI As Long
  PortID As Long
  LPTDelay As Long
  LPTAnalog As Long
  InputStatus As Long
  MotorStatus As Long
End Type

Public Const ftiFehler = &H1FFFF
Public Const ftiEin = 1
Public Const ftiAus = 0
Public Const ftiLinks = 1
Public Const ftiRechts = 2
```

Als Programmierbeispiel siehe umFish.FRM

```
DoEvents
```

sollte angeeigneten Stellen in den Code eingefügt werden um die Unterbrechbarkeit der Anwendung in engen Abfrageschleifen (z.B. warten auf Endtaster) zu gewährleisten.

Die True bzw False Werte von GetInputs entsprechen den VisualBasic Konventionen.

Visual C++

In den Workspace des Projektes sollte die Source umFish.h und das Link-File umFish.lib eingefügt werden. umFish.lib muß auch bei den Linker-Optionen des Projektes eingetragen werden :

```
typedef struct {
  HANDLE hCom;
  DWORD PortID;
  DWORD LPTDelay;
  DWORD LPTAnalog;
  DWORD InputStatus;
  DWORD MotorStatus;
} ftDCB;

DWORD __stdcall OpenInterface(ftDCB *Interface, LPCSTR ComName);
DWORD __stdcall CloseInterface(ftDCB *Interface);
BOOL  __stdcall GetInput(ftDCB *Interface, DWORD InputNr);
DWORD __stdcall GetInputs(ftDCB *Interface);
DWORD __stdcall ClearMotors(ftDCB *Interface);
DWORD __stdcall SetMotors(ftDCB *Interface, DWORD MotorStatus);
DWORD __stdcall SetLamp(ftDCB *Interface,
           DWORD LampNr, DWORD OnOff);
DWORD __stdcall SetMotor(ftDCB *Interface,
      DWORD MotorNr, DWORD Direction);
DWORD __stdcall GetAnalog(ftDCB *Interface, DWORD AnalogNr);

DWORD __stdcall StartDriver();
DWORD __stdcall StopDriver();

const DWORD ftiAus    = 0;
const DWORD ftiEin    = 1;
const DWORD ftiLinks  = 1;
const DWORD ftiRechts = 2;

const DWORD ftiFehler = 0x0001FFFF; ftiTrue = 0xFFFFFFFF;
ftiAus=0;
```

Anwendungsbeispiel : umFishVC.cpp als einfache Console-Application.

ACHTUNG : umFish.DLL ist unabhängig von einer Version von VC++, umFish.LIB dagegen hat bei verschiedenen Versionen ein unterschiedliches Format. Mitgeliefert werden :

- umFish.H und umFish.LIB im VC++6.0-Format
- umFish.H und umFish50.LIB im VC++5.0-Format
- umFishLoad.H, das umFish.DLL dynamisch lädt und auf ein *.LIB verzichtten kann. Für chronische Fälle und für Fans.

Beispiele für den Aufbau einer Consol-Application aus den mitgelieferten Sources (auf Projektdateien habe ich verzichtet, weil das noch komplizierter ist).

VC++6.0 deutsch :

- Datei | Neu | Projekte | Konsolanwendung : Name und Verzeichnis umFishVC, leeres Projekt, Fertigstellen
- Manuelles kopieren der nochfolgenden Dateien in das neu angelegte Projektverzeichnis
- Projekt | Dem Projekt hinzufügen | Dateien : umFish.h, umFish.lib, umFishVC.cpp
- Erstellen : umFishVC.EXE (F7)
- Erstellen | Ausführen von umFishVC.EXE (CTL+F5)

VC++5.0 englisch :

- File | New | Projects | Win32 Console Application
- Project | Add to Project | Files : umFish.h, umFish50.lib, umFishVC.cpp
- F7
- F5

Anmerkung (für mich) : umFish50.LIB wird aus umFish.LIB (VC++6.0) wie folgt (unter DOS / Konsole) erstellt :

LIB /CONVERT /LINK50COMPAT /OUT:umFish50.LIB umFish.LIB

dabei sollten alle Beteiligten am besten im gleichen Verzeichnis liegen (das spart Pfadangaben): Lib.EXE, Link.EXE, MSPDB60.DLL (aus C:\Programme\Visual Studio ...) und umFish.LIB

VC++4.2 englisch, dyn. Laden :

- New Workspace : Console
- Insert umFishLoad.h, umFishVCLoad.cpp
- Kompilieren (F7)
- in den Projektpfad \Debug bzw. \Release umFish.dll kopieren, alternativ nach \Win\System.

Delphi

Einbinden folgender Deklarationen z.B. im Interfaceteil der zu einer Form gehörenden Unit :

```
const
  ftiAus=0; ftiEin=1; ftiLinks=1; ftiRechts=2;
  ftiFehler=$0001FFFF; ftiTrue=-1; ftiFalse=0;

type TftiDCB = record
  hCom: LongInt;
  PortID: LongInt;
  LPTDelay: LongInt;
  LPTAnalog: LongInt;
  InputStatus: LongInt;
  MotorStatus: LongInt;
  end;
type TpftiDCB = ^TftiDCB;

var
  ftiDCB: TftiDCB; pftiDCB: TpftiDCB = @ftiDCB;

function OpenInterface(ft: TpftiDCB; PortName: string):LongInt;
       stdcall; external 'umFish.dll';
function CloseInterface(ft: TpftiDCB):LongInt;
       stdcall; external 'umFish.dll';
function GetInput(ft: TpftiDCB; InputNr:LongInt):LongInt;
       stdcall; external 'umFish.dll';
function GetInputs(ft: TpftiDCB):LongInt;
       stdcall; external 'umFish.dll';
function ClearMotors(ft: TpftiDCB):LongInt;
       stdcall; external 'umFish.dll';
function SetMotors(ft: TpftiDCB; MotorStatus:LongInt):LongInt;
       stdcall; external 'umFish.dll';
function SetMotor(ft: TpftiDCB; MotorNr,
Direction:LongInt):LongInt;
       stdcall; external 'umFish.dll';
function SetLamp(ft: TpftiDCB; LampNr, OnOff:LongInt):LongInt;
       stdcall; external 'umFish.dll';
function GetAnalog(ft: TpftiDCB; AnalogNr:LongInt):LongInt;
       stdcall; external 'umFish.dll';
function StartDriver():LongInt; stdcall; external 'umFish.dll';
function StopDriver():LongInt; stdcall; external 'umFish.dll';
```

Siehe auch die Source umFish.PAS.

```
Application.ProcessMessage;
```

sollte angeeigneten Stellen in den Code eingefügt werden um die Unterbrechbarkeit der Anwendung in engen Abfrageschleifen (z.B. warten auf Endtaster) zu gewährleisten.