USB/I²C-INTE

Von Paul Goossens

Seit der Einführung des I²C-Bus von Philips in den achtziger Jahren wird dieser Bus auch gern für die Verbindung selbst entwickelter Hardware mit dem PC eingesetzt. Die Anpassung übernimmt meist eine einfache Interface-Schaltung, die vom PC-Parallelport gesteuert wird. Weil das nicht mehr ganz zeitgemäß ist, haben wir ein I²C-Interface für den USB-Port entwickelt.

RFACE I²C for ever....

Dem I²C-Bus wurde in Elektor schon öfter Aufmerksamkeit zuteil. In der Vergangenheit veröffentlichten wir zum Beispiel eine Reihe von Modulen, die über einen I²C-Bus mit dem PC kommunizieren können. Vervollständigt wurde diese Reihe durch eine ISA-Karte, die als I²C-Interface arbeitet. Eine andere häufig angewandte Lösung besteht darin, den PC-Parallelport mit Hilfe einer einfachen Anpassschaltung und der nötigen PC-Software in einen I²C-Bus zu verwandeln. Leider wird das Steuern des PC-Parallelports durch die modernen PC-Betriebssysteme erheblich erschwert. Es kommt immer häufiger vor, dass die Software von damals auf neuen PCs nicht mehr fehlerfrei läuft. Wir haben uns deshalb ein zeitgemäßes I²C-Interface entwickelt, das an den USB-Port angeschlossen wird.

Wichtige Eigenschaften

- **Kompakter Aufbau**
- **USB 1.0 Full-speed**
- **Kompatibel mit USB 2.0 Hosts**
- **Windows-Versionen: ab 98 SE**
- **Kein zusätzlicher Treiber nötig**
- **Kabellänge zwischen Interface und Modulen: bis 100 m**
- **I²C-Taktfrequenz: 100 kHz**
- **Einfache Anwendung durch DLL**
- **Source-Code der Firmware und Anwendungsbeispiele frei verfügbar**
- **Stromversorgung mit Steckernetzteil**

I²C-Geschichte

Philips entwickelte den I²C-Bus ursprünglich, um integrierte Bausteine in Geräten der Unterhaltungselektronik auf einfache und (Platinen-)Platz sparende Weise miteinander zu verbinden. Der I²C-Bus begnügt sich mit zwei Signalleitungen, wo sonst acht Datenleitungen, diverse Adressleitungen und mehrere weitere Steuersignal-Leitungen nötig sind. Nach Einführung des I²C-Bus wurden sowohl von Philips als auch von anderen Herstellern diverse Bausteine entwickelt, die den I²C-Bus unterstützen. Zunächst waren die meisten Bausteine für Audio- und Video-Systeme bestimmt, später kamen auch digitale I/O-Chips, A/D-Wandler und anderes mehr hinzu. Inzwischen ist die Palette der Bausteine mit I²C-Bus-Anschluss breit gefächert, sie deckt viele Bereiche der digitalen und analogen Schaltungstechnik ab.

I²C-Philosophie

Der I²C-Bus ist ein serieller Bus, der nach dem Master-Slave-Prinzip arbeitet. An einem I²C-Bus arbeitet immer ein Baustein als so genannter Master, während sich die übrigen Bausteine als Slaves verhalten. Dies bedeutet unter anderem, dass der Master das Taktsignal generiert, und ferner muss jede Daten-Transaktion vom Master initiiert werden. Slave-Bausteine dürfen niemals aus eigener Initiative Daten auf den Bus legen.
Jedem Slave, der mit dem I²C-Bus verbunden ist, wird eine eigene Adresse zugewiesen. Alle Transaktionen beginnen mit einer START-Kondition, gefolgt von einer 7-bit-Adresse und einem Read/Write-Bit (R/W). Wenn ein Slave mit dieser Adresse am I²C-Bus angeschlossen ist, sendet er ein Acknowledge-Signal (ACK) zurück zum Zeichen dafür, dass er seine Adresse erkannt hat. Von diesem Zeitpunkt an ignorieren alle anderen Slaves die Vorgänge auf dem I²C-Bus, und zwar so lange, bis sie eine STOP-Kondition erkennen.
Das R/W-Bit gibt an, ob der Master anschließend eine Schreib- oder Lese-Operation ausführt. Wenn R/W logisch 1 ist, handelt es sich um eine Lese-Operation. In diesem Fall setzt der Slave das folgende Byte auf den I²C-Bus. Ist R/W logisch 0 (Schreib-Operation), wird das folgende Byte vom Master auf den Bus gesetzt. In beiden Fällen generiert der Master das Taktsignal. Nach Abschluss der Transaktion folgt eine STOP-Kondition, danach kann die nächste Transaktion beginnen.

Vorüberlegungen

An ein I²C-Interface für den PC werden spezifische Anforderungen gestellt, die wir natürlich bei unserem Entwurf berücksichtigt haben. So soll sich zum Beispiel die Installation unter Windows möglichst einfach gestalten, und auch das Steuern aus einer Applikation soll möglichst unkompliziert sein. Auf der Seite der Interface-Hardware kommt der maximalen Bus-Länge eine wichtige Bedeutung zu. Der zulässige Abstand zwischen PC und I²C-Hardware soll möglichst groß sein, so dass die I²C-Schaltung weit entfernt vom PC platziert werden kann.
Um unser Vorhaben zu realisieren, mussten wir auf die Suche nach einem geeigneten Controller mit USB-Anschluss gehen. Die Wahl fiel auf den TUSB3410 von Texas Instruments, weil dieser Controller sowohl über eine USB- als auch I²C-Schnittstelle verfügt. Von seinem Hersteller wurde er unter der etwas missverständlichen Bezeichnung "USB to Serial Port Controller"

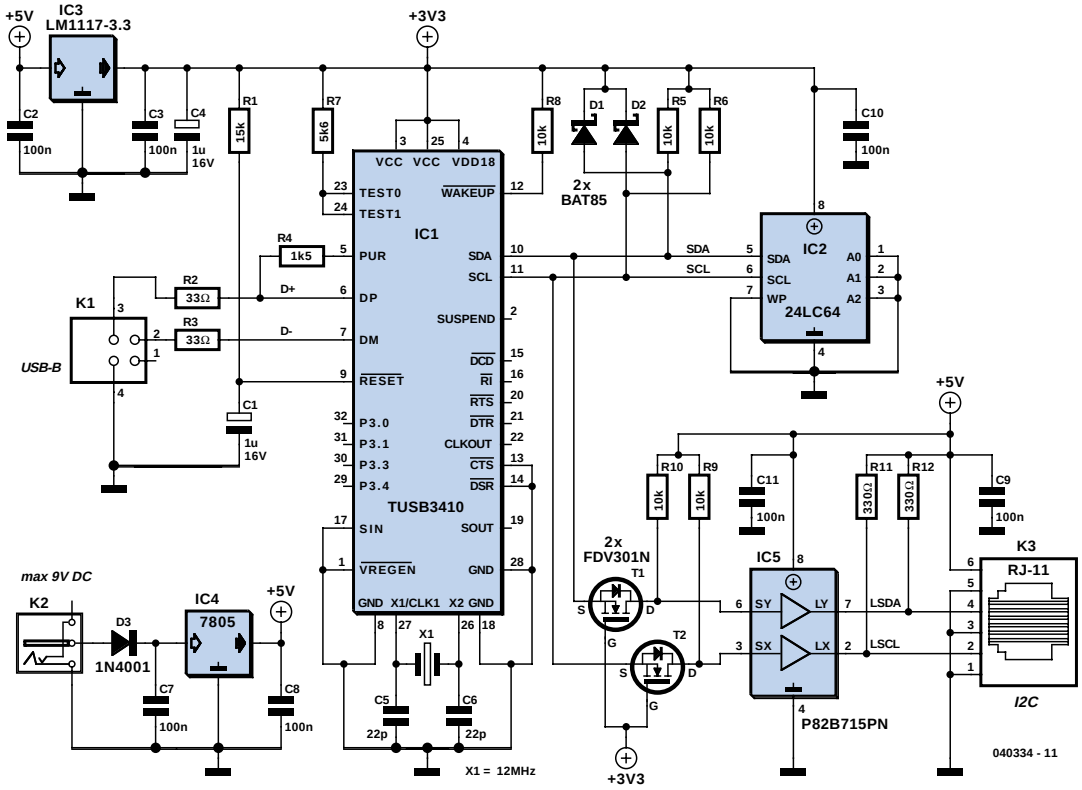


Bild 1. Das USB/I²C-Interface ist mit dem TUSB3410 von Texas Instruments aufgebaut.

auf den Markt gebracht. Genauer betrachtet ist der TUSB3410 ein 8051-kompatibler Controller mit 16 KB Programmspeicher, der außer den Schnittstellen für USB und I²C eine vielseitig verwendbare serielle Schnittstelle bietet. Dies alles und noch einiges mehr ist in einem winzigen 32-Pin-SMD-Gehäuse untergebracht.

Damit das I²C-Interface auch aus größerer Entfernung zum PC genutzt werden kann, haben wir die mit dem TUSB3410 aufgebaute Interface-Schaltung durch einen I²C-Bus-Extender des Typs P82B715PN ergänzt.

Schaltung

Mittelpunkt der Schaltung in **Bild 1** ist der Controller TUSB3410 (IC1). Die Stromversorgung übernimmt ein handelsübliches 9-V-Steckernetzteil, seine Ausgangsspannung wird von Spannungsregler IC4 auf 5 V herabgesetzt. Die Betriebsspannung 3,3 V für den TUSB3410 und das I²C-EEPROM (IC2) liefert ein zweiter Spannungsregler (IC3), der dem ersten nachgeschaltet ist. Aus Sicherheitsgründen wird die Schaltung nicht über den USB-Bus mit Strom versorgt. Da die Betriebsspannung 5 V auch an der RJ11-Anschlussbuchse K3 anliegt, sind Überlastungen und Kurzschlüsse von außen nicht auszuschließen; der USB-Port des PC könnte dadurch beschädigt werden.

Die Datenleitungen der USB-Anschlussbuchse K1 liegen über die Schutzwiderstände R2 und R3 an den zugehörigen Controller-Anschlüssen. Datenleitung D+ ist außerdem über einen 1,5-k_-Widerstand (R4) mit Ausgang PUR (Pull Up Resistor) von IC1 verbunden. Normalerweise liegt dieser Widerstand an der Betriebsspannung des Controllers. Der USB-Hub erkennt daran, dass am USB-Port ein Full-speed-USB-Device angeschlossen ist. Wenn der Pullup-Widerstand an einem Controller-Ausgang liegt, kann der Controller dem USB-Hub selbst signalisieren, dass ein USB-Device angeschlossen ist. Außerdem kann der Controller dem Hub auch "vorgaukeln", dass kein USB-Device angeschlossen ist, indem er den Ausgang auf 0 setzt. Dies ist beim Booten des Controllers von Bedeutung; weitere Informationen zu diesem Thema enthält die Textbox "Booten des TUSB3410".

Die Firmware befindet sich in einem I²C-EEPROM (IC2), es ist über den I²C-Bus mit dem Controller verbunden. Die Zener-Dioden D1 und D2 schützen den Controller vor möglicherweise gefährlichen Spannungsspitzen auf dem I²C-Bus. Die Pullup-Widerstände R5 und R6 müssen generell bei jedem I²C-Bus vorhanden sein.

Da die Betriebsspannung des Controllers 3,3 V beträgt, arbeitet auch der I²C-Bus des Controllers mit dieser Spannung. Die an seinen I²C-Anschlüssen liegenden Signale können nicht ohne Weiteres Bausteine steuern, die mit 5 V betrieben werden. Leider ist der I²C-Bus-Extender P82B715PN ein 5-V-Chip; die Spannungspegel müssen deshalb von T1, T2, R9 und R10 an die höhere Spannung angepasst werden. Der Bus-Extender sorgt zusammen mit R11 und R12 dafür, dass der I²C-Bus auch größere Distanzen überbrücken kann, indem er die Ströme verstärkt und die Impedanzen herabsetzt. Die in dieser Weise aufbereiteten Signale stehen an RJ11-Buchse K3 zur Verfügung.

Die doppelseitige Platine, deren Layout **Bild 3** zeigt, hat relativ bescheidene Abmessungen; trotzdem sind auf ihr alle notwendigen Steckverbindungen untergebracht. Die Montage der SMD-Bauelemente IC1, T1 und T2 bedarf erhöhter Sorgfalt. Die Transistoren können noch einigermaßen bequem mit einem Lötgerät gelötet werden, das mit einer bleistift-förmigen Spitze ausgerüstet ist. Die IC-Montage erfordert jedoch zusätzliche Maßnahmen. Erfolg versprechend ist folgende Vorgehensweise:

Kleben Sie zunächst den IC-Körper mit einem kleinen Tropfen Alleskleber in der exakten Position auf die Platine.

Verlöten Sie die IC-Pins nicht nur mit den zugehörigen Platinen-Kontaktflächen, sondern auch miteinander. Dies muss so schnell geschehen, dass die zulässige Löttemperatur des IC nicht überschritten wird. Nachdem das IC und die entstandenen Kurzschlussbrücken abgekühlt sind, legen Sie ein Stück Entlötlitze quer über die IC-Pins und saugen das überschüssige Lötzinn (die Brücken) mit der Entlötlitze und dem Lötgerät ab. Verwenden Sie hierzu nicht die Entlötpumpe, und lassen Sie bei der Lötarbeit auf keinen Fall außer Acht, dass das IC nicht zu heiß werden darf! Nach dem Löten kontrollieren Sie unter der Lupenlampe und mit dem Ohmmeter genauestens, ob alle IC-Pins

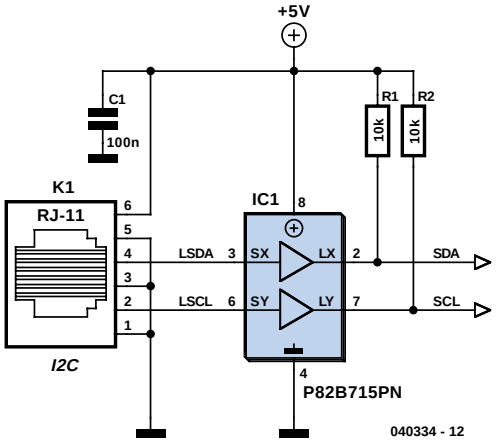


Bild 2. Eingangsschaltung für I²C-Module bei Verwendung des I²C-Extenders (IC5).

Stückliste

Widerstände:
R1 = 15 k
R2,R3 = 33 Ω
R4 = 1k5
R5,R6,R8...R10 = 10 k
R7 = 5k6
R11,R12 = 330 Ω

Kondensatoren:
C1,C4 = 1 µ/16 V stehend
C2,C3,C7...C11 = 100 n
C5,C6 = 22 p

Halbleiter:
D1,D2 = BAT85
D3 = 1N4001
IC1 = TUSB3410 (z.B. Digikey 296-12699-ND)
IC2 = 24LC64 (programmiert, EPS 040334-21)
IC3 = LM1117-3.3 oder LD1117V33C (z.B. Digikey 497-1492-5-ND)
IC4 =7805
IC5 = P82B715PN (z.B. Farnell 559-258, RS-Components 821-784)
T1,T2 = FDV301N (z.B. Farnell 995-848, RS-Components 354-4907)

Außerdem:
K1 = USB-B-Buchse, gewinkelt, für Platinenmontage (z.B. Farnell 152-754)
K2 = Netzadapter-Buchse für Platinenmontage
K3 = RJ11-Buchse, 6-polig (z.B. Farnell 393-8359)
X1 = Quarz 12 MHz
USB-Kabel
Platine: EPS 040334-1 *
Diskette mit Software: EPS 040334-11*

* Das Platinen-Layout im PDF-Format und die Software stehen auch auf unserer Website www.elektor.de zum kostenlosen Download bereit.

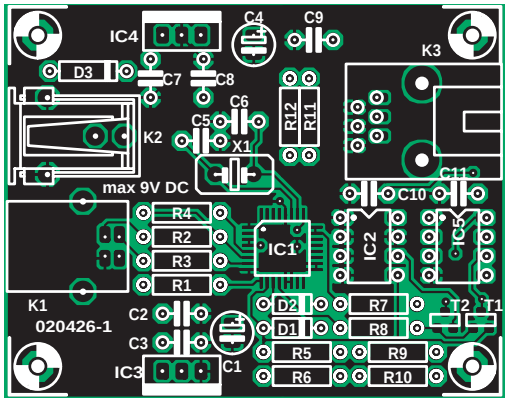
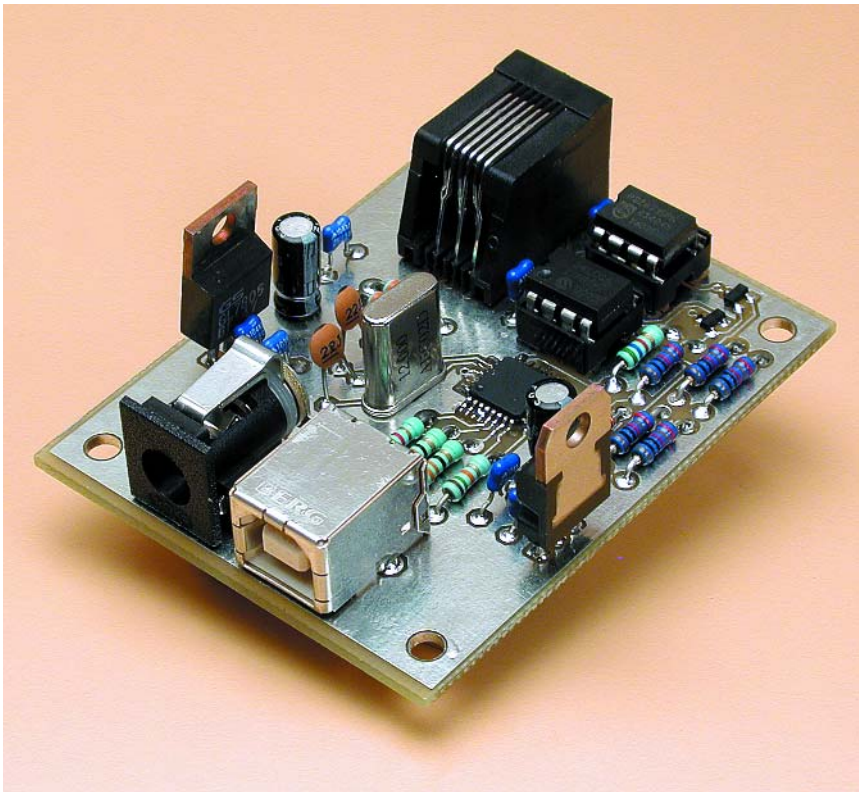


Bild 3. Platine für das USB/I²C-Interface. Die Montage der SMD-Bauelemente IC1, T1 und T2 bedarf erhöhter Sorgfalt.



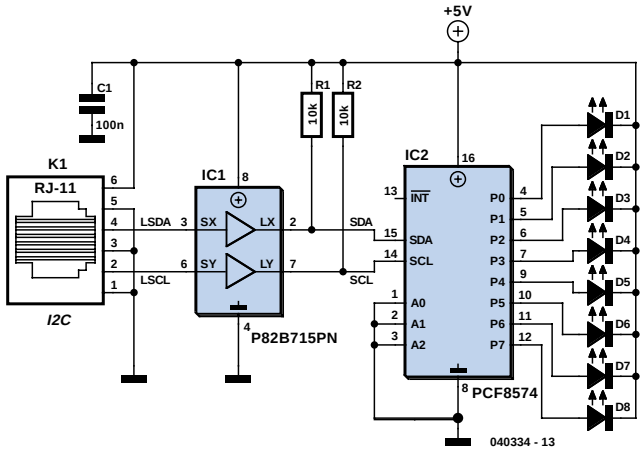


Bild 4. Hilfsschaltung zum Testen des USB/I²C-Interface.

an der Platine angelötet sind und alle Kurzschlüsse beseitigt wurden.
Ein typisches Modul, das an das I²C-Interface angeschlossen werden kann, wird im Beitrag "I²C-Home-Bus" an anderer Stelle in dieser Elektor-Ausgabe beschrieben.
Falls auf den I²C-Bus-Extender verzichtet werden soll, müssen folgende Modifikationen vorgenommen werden: IC5, R11 und R12 entfallen, die Anschlüsse 2 und 3 von IC5 auf der Platine werden miteinander verbunden, und auch zwischen den Anschlüssen 6 und 7 von IC5 wird eine Brücke eingesetzt. In diesem Fall dürfen in den angeschlossenen Modulen die SCL- und SDA-Leitungen unmittelbar mit den I²C-Anschlüssen der dort befindlichen ICs verbunden sein. Dagegen ist bei vorhandenem I²C-Bus-Extender für jedes angeschlossene Modul eine zusätzliche Eingangsschaltung notwendig, sie ist in **Bild 2** dargestellt.

Installation und DLL

Das Installieren des USB/I²C-Interface in Windows (ab Version 98 SE) ist problemlos. Zuerst wird das Steckernetzteil an die Schaltung angeschlossen, danach kann USB-Buchse K1 über ein USB-Kabel mit dem PC verbunden werden. Windows erkennt die Schaltung als "HID-Device" und installiert automatisch den erforderlichen Treiber. Zum Abschluss meldet Windows, dass die Hardware erfolgreich installiert wurde.
Mit der Installation im Betriebssystem ist es natürlich noch nicht getan. Es muss ein Programm geschrieben werden, das auf das USB/I²C-Interface zugreift und das daran angeschlossene I²C-Modul steuert. Natürlich ist dieses Programm auch von der Art des I²C-Moduls und seinem Verwendungszweck abhängig. Da das Steuern eines "HID-Device" unter Windows nicht jedem Programmierer geläufig sein dürfte, haben wir für das USB/I²C-Interface eine DLL geschrieben, die diese Aufgabe übernimmt. Wir empfehlen, die DLL in den Ordner *C:\Windows\System32* zu kopieren, so dass Windows die DLL immer findet. Die DLL muss dann nicht im gleichen Ordner wie die Applikation stehen, die mit der DLL arbeitet.
Die DLL ist zusammen mit jeder Programm-Entwicklungsumgebung verwendbar, die das Einbinden von DLLs unterstützt. Dazu gehören unter anderem Visual Basic, Visual C++, Delphi sowie der Borland C++ Builder.

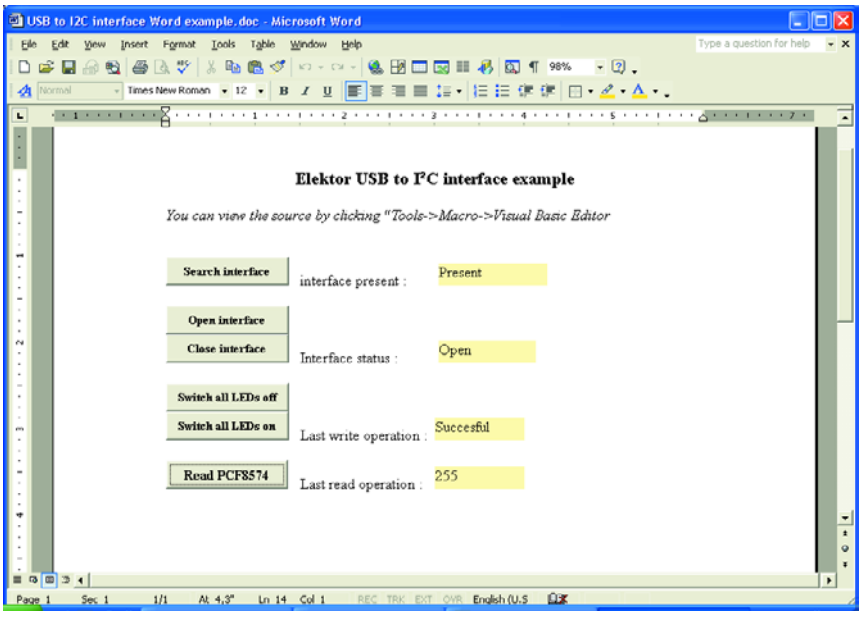


Bild 5. Programm-Beispiel in Microsoft Word.

Zwei Wege

Für den Einbau der DLL in eine Applikation wurden zwei Möglichkeiten geschaffen. Bei der Konzeption der DLL erschien es sinnvoll, dass die Applikation ein Byte-Array bereitstellt, in das die DLL die empfangenen Byte schreibt oder aus dem die DLL die zu sendenden Byte liest. Für die meisten Programmiersprachen stellen Arrays als Argumente von DLL-Funktionen kein Problem dar. Bei Visual Basic verhält sich dies etwas anders, obwohl auch diese Programmiersprache mit Arrays zurechtkommt. Der Umgang mit Arrays gestaltet sich jedoch deutlich mühsamer, insbesondere wenn wenig Programmier-Erfahrung vorhanden ist. Wir haben deshalb verschiedene zusätzliche Funktionen hinzugefügt, die das Programmieren des USB/I²C-Interface unter Visual Basic erleichtern.
Als Starthilfe für das Programmieren haben wir zwei sehr einfache Beispiel-Applikationen in Visual Basic und in Delphi geschrieben, die einen Standard-I²C-I/O-Extender steuern. Diese Beispiele demonstrieren anschaulich die beiden Einsatzmöglichkeiten der DLL. Die Schaltung des I²C-Testmoduls, das damit gesteuert werden kann, ist in **Bild 4** dargestellt. Auf einer kleinen Lötpunktraster-Platine ist diese Testschaltung schnell aufgebaut.

Ans Werk mit Word

In der November-Ausgabe von Elektor (Titel: "VB unter Word") hatten wir gezeigt, dass das bekannte Textprogramm Word von Microsoft auch zum Schreiben von Visual-Basic-Programmen dienen kann. Da die meisten Windows-Anwender auch von Word

Gebrauch machen, schien es uns sinnvoll, auch ein einfaches Beispiel mit Hilfe von Word zu schreiben. Natürlich haben wir den bequemen Weg gewählt und für die Kommunikation mit der DLL keine Arrays verwendet.
Das Beispiel ist Bestandteil der zu diesem Projekt gehörenden Software (EPS 040334-11), es ist dort im Ordner "Word example" zu finden. Abhängig von den Sicherheitseinstellungen kann Word beim Öffnen des Dokuments anfragen, ob in dem Dokument Makros verwendet werden dürfen. Diese Frage muss mit "Ja" beantwortet werden, anderenfalls ist das Beispiel-Programm nicht funktionsfähig.
Nach dem Öffnen erscheint auf dem Bildschirm das in **Bild 5** dargestellte Dokument. Nach Anklicken des oberen Button sucht das Programm nach dem USB/I²C-Interface; das Feld rechts daneben gibt Auskunft über den Sucherfolg. Bevor das USB/I²C-Interface genutzt werden kann, muss es durch einen Mausklick auf den Button "Open interface" geöffnet werden. Nach Gebrauch wird es durch einen Mausklick auf den Button "Close interface" geschlossen. Die Schaltzustände dieser beiden Button werden in dem zugehörigen gelben Feld angezeigt.
Mit den beiden nächsten Button können die LEDs der Testschaltung aus Bild 4 geschaltet werden (sie ist hoffentlich aufgebaut und angeschlossen!). Der Erfolg ist natürlich unmittelbar an den LEDs sichtbar, die Schaltzustände werden aber auch auf dem Bildschirm im Feld rechts neben den Button angezeigt.
Ferner ist ein Button vorhanden, mit dem die Eingangszustände des PCF8574 sichtbar gemacht werden können. Die Werte erscheinen in dem Feld rechts neben diesem Button.
Weitere Informationen enthält die Textbox "DLL-Anwendung - Ohne Arrays". Zusammen mit dem Quellcode des Word-Dokuments erklären die dort stehenden Informationen die Arbeitsweise dieses Programm-Beispiels.

Delphi

Das zweite Beispiel ist eine ebenfalls sehr einfache Applikation, diesmal geschrieben in der Sprache Delphi. Wir werden öfter gefragt, weshalb wir viele Beispiele in Delphi schreiben. Der Hauptgrund ist die leichte Lesbarkeit dieser auf Pascal aufbauenden Programmiersprache. Das hat zur Folge, dass auch C-Programmierer diese Programme leicht verstehen können. Umgekehrt haben Pascal-Programmie-

DLL-Funktionen

Standard:

```
type TReport = array[0..200] of Byte;

function I2C_USB_Present : Boolean; stdcall;
function I2C_USB_Opened : Boolean; stdcall;
function I2C_USB_Open : Boolean; stdcall;
procedure I2C_USB_Close; stdcall;
```

Kommunikation mit eigenem Buffer:

```
function I2C_USB_Write (adr : Byte; length : Byte; data : array of Byte) :
        Boolean; stdcall;
function I2C_USB_Read (adr : Byte; length : Byte; var data : array of Byte) :
        Boolean; stdcall;
```

Kommunikation mit Buffer der DLL:

```
procedure I2C_USB_ClearWriteBuffer; stdcall;
procedure I2C_USB_ClearReadPointer; stdcall;
procedure I2C_USB_FillBuffer (data: Byte); stdcall;
function I2C_USB_GetBuffer : Byte; stdcall;
function I2C_USB_WriteWithBuffer (adr : Byte) : Boolean; stdcall;
function I2C_USB_ReadWithBuffer (adr:Byte; length:Byte) : Boolean; stdcall;
```

Booten des TUSB3410

Der "USB to Serial Port Controller" TUSB3410 verhält sich beim Starten anders als gewohnt. Im Gegensatz zu vielen anderen modernen Controllern befindet sich kein Flash-Speicher für die dauerhafte Programm-Speicherung auf dem Chip. Stattdessen ist der TUSB3410 mit einem flüchtigen RAM ausgestattet, in dem Programme nur temporär gespeichert werden können; nach dem Starten ist dieses RAM stets leer. Hersteller Texas Instruments hat dem TUSB3410 jedoch auch noch ein kleines ROM mit auf den Weg gegeben, in dem ein kurzes Lade-Programm permanent gespeichert ist. Dieses Programm lädt beim Start automatisch ein externes Programm in das RAM. Es stehen mehrere Möglichkeiten zur Wahl:

Zuerst prüft das Lade-Programm, ob ein I²C-EEPROM am I²C-Bus angeschlossen ist. Wenn die EEPROM-Suche erfolglos verlief, meldet sich der TUSB3410 auf dem USB-Bus mit einer bestimmten Vendor-ID und Product-ID an. Vorausgesetzt, dass der richtige Treiber unter Windows installiert ist (kostenlos herunterladbar von der Texas Instruments Website http://www.ti.com), erkennt Windows den Controller als "TUSB3410 boot device". Danach sendet Windows die vom Benutzer ausgewählte Firmware zum Controller. Über Einzelheiten dieser Vorgänge gibt die Dokumentation von Texas Instruments Auskunft. Der TUSB3410 speichert die empfangene Software in seinem Programm-RAM und meldet sich danach vom USB-Bus ab.

Die erste Methode, bei der die Firmware in einem I²C-EEPROM gespeichert ist, wird auch beim hier beschriebenen USB/I²C-Interface angewendet. Das Lade-Programm des TUSB3410 erkennt am Header der im EEPROM stehenden Daten, ob es sich um eine zu ladende Firmware handelt. Wenn dies zutrifft, wird die Firmware aus dem EEPROM in das Programm-RAM des TUSB3410 kopiert. Anschließend schaltet der TUSB3410 in den "normalen" Mode und führt die in das RAM geladene Firmware aus. Vorteil dieser Methode ist die Unabhängigkeit von Windows und dem dort zu installierenden Treiber.

Übrig bleibt nur noch ein Problem: Was geschieht, wenn zwei mit dem I²C-Bus verbundene Geräte mit einem TUSB3410 arbeiten, die Firmware jedoch unterschiedlich ist? Woran erkennt Windows, welche Firmware-Dateien zu den Controllern übertragen werden muss? Die Lösung besteht darin, jedem Controller eine eigene Kombination von Vendor-ID und Product-ID zuzuweisen. Die IDs können in den EEPROMs abgelegt werden; der Controller meldet sich dann mit diesen Werten am USB-Bus an. In der Dokumentation von Texas Instruments ist beschrieben, wie die "inf"-Datei des TUSB3410 angepasst werden muss, damit durch unterschiedliche VID-PID-Kombinationen mehrere Firmware-Dateien selektiert werden können.

Weitere Informationen zum Booten des TUSB3410 hält Texas Instruments auf seiner Website bereit.

rer Schwierigkeiten, in C geschriebene Programme nachzuvollziehen, sofern sie nicht auch Kenntnisse in C haben. Die zu diesem Beispiel gehörenden Dateien sind im Ordner "Delphi example" gespeichert. Damit der Quellcode möglichst kurz ist, wird die Kommunikation mit dem USB/I²C-Interface von einer einzigen Prozedur abgewickelt; sie heißt *Timer1Timer*. Diese Prozedur ist auch für das Suchen und Öffnen des USB/I²C-Interface zuständig. Unmittelbar danach folgen Lese- und Schreib-Operationen vom bzw. zum PCF8574, der natürlich am I²C-Bus angeschlossen sein muss.

Das Öffnen des USB/I²C-Interface nach Starten eines Programms und das Schließen beim Verlassen ist problemlos möglich. Im Programm-Beispiel geschieht dies innerhalb einer Timer-Routine, die in jeder Sekunde zwei Mal durchlaufen wird. Dadurch kann das USB/I²C-Interface beliebig mit dem PC-USB-Port verbunden und wieder getrennt werden, während das Programm läuft.

Nächster Schritt

In dieser Elektor-Ausgabe wird an anderer Stelle eine Anwendung für das USB/I²C-Interface beschrieben, der Beitrag hat den Titel "I²C-Home-Bus". Dem Entwerfen eigener Module steht natürlich nichts im Weg, es setzt allerdings etwas Programmier-Erfahrung unter Windows voraus. (040334)gd

DLL-Anwendung

Ohne Arrays

Allgemein:

Durch Aufruf von **I2C_USB_Open** wird das Interface geöffnet. Die Funktion gibt den Wert "Wahr" zurück, wenn der Zugriff erfolgreich war.

Lesen aus einem I²C-Baustein:

Bevor eine neue Lese-Operation gestartet wird, muss **I2C_USB_ClearReadPointer** aufgerufen werden.

Danach werden mit **I2C_USB_ReadWithBuffer** die Byte auf dem I²C-Bus gelesen. Diese Funktion erwartet zwei Argumente: Erstes Argument ist die I²C-Adresse des Bausteins, aus dem gelesen werden soll. Zweites Argument ist die Anzahl der Byte (maximal 255), die gelesen werden sollen. Die Funktion gibt den Wert "Wahr" zurück, wenn die Lese-Operation erfolgreich verlief. Anschließend können die gelesenen Byte mit der Funktion **I2C_USB_GetBuffer** der Reihe nach aus dem Buffer geholt werden.

Schreiben in einen I²C-Baustein:

Vor dem einer Schreib-Operation wird **I2C_USB_ClearWriteBuffer** aufgerufen. Danach können die zu schreibenden Byte der Reihe nach mit **I2C_USB_FillBuffer** der DLL übergeben werden. Argument ist hier das einzelne zu schreibende Byte. Nach Übergabe aller zu schreibenden Byte an die DLL genügt ein Aufruf von **I2C_USB_WriteWithBuffer**, um die Byte in den gewünschten Baustein zu schreiben. Einziges Argument dieser Funktion ist die I²C-Adresse des Ziel-Bausteins. Die Funktion gibt den Wert "Wahr" zurück, wenn die Schreib-Operation erfolgreich verlief.

Mit Arrays

Allgemein:

Siehe DLL-Anwendung ohne Arrays.

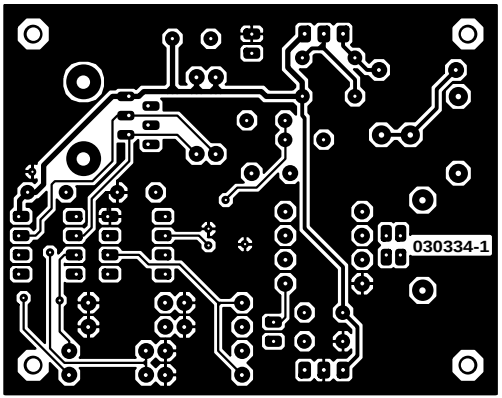
Lesen aus einem I²C-Baustein:

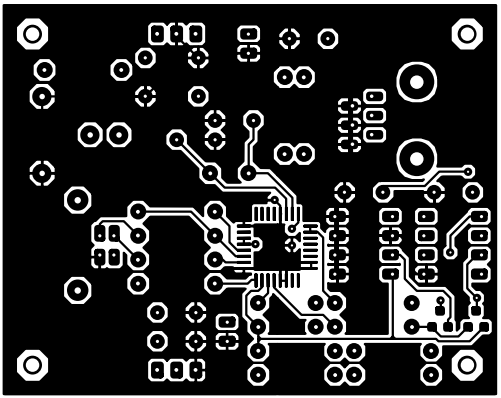
Zum Lesen von Daten aus einem I²C-Baustein wird nur **I2C_USB_Read** benötigt. Diese Funktion erwartet drei Argumente: Erstes Argument ist die I²C-Adresse des Bausteins, aus dem gelesen werden soll. Zweites Argument ist die Anzahl der Byte (maximal 255), die gelesen werden sollen. Als drittes Argument benötigt die Funktion einen Buffer, in dem die gelesenen Byte abgelegt werden sollen. Die Funktion gibt den Wert "Wahr" zurück, wenn die Lese-Operation erfolgreich verlief.

Schreiben in einen I²C-Baustein:

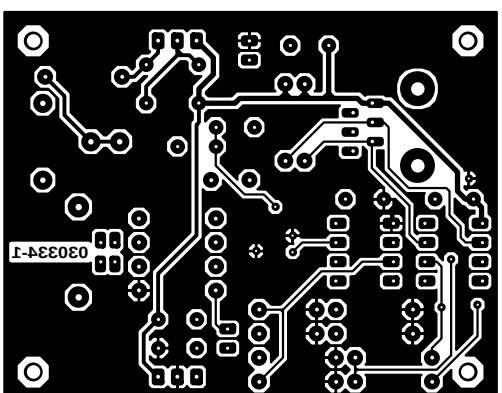
Auch diese Operation ist mit einer einzigen Funktion möglich, sie heißt **I2C_USB_Write**. Diese Funktion erwartet drei Argumente: Erstes Argument ist die I²C-Adresse des Bausteins, in den geschrieben werden soll. Zweites Argument ist die Anzahl der Byte (maximal 255), die geschrieben werden sollen. Als drittes Argument benötigt die Funktion den Buffer, in dem die zu schreibenden Byte stehen. Die Funktion gibt den Wert "Wahr" zurück, wenn die Schreib-Operation erfolgreich verlief.

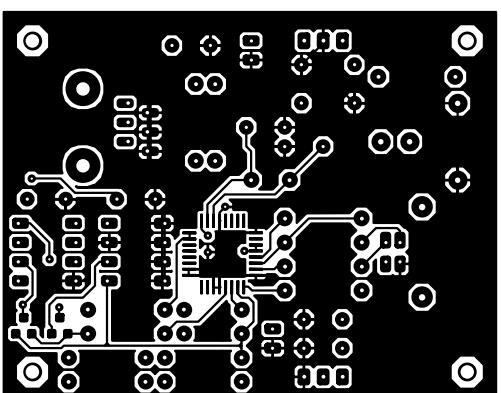
Hinweis: Innerhalb des Programms muss sicher gestellt sein, dass der Buffer groß genug ist, um die zu lesenden bzw. zu schreibenden Byte aufzunehmen!





non reflected





reflected