Einfach und schnell messen, steuern, testen und programmieren – mit einem einfachen Bussystem wie dem I²C-Bus ist dies kein Problem, und die große Vielfalt I²C-kompatibler Bausteine, Prozessoren und Geräte macht zahlreiche Lösungen einfach.
Mit unserem kleinen Interface kann von einem PC aus via USB mit einfachsten Befehlen direkt auf angeschlossene Geräte oder Bausteine mit I²C-Schnittstelle zugegriffen werden. Durch die Makrofunktion ist das Interface z. B. mit nur einem zusätzlichen IC als Datenlogger für analoge/digitale Signale, Temperaturwerte usw. konfigurierbar.

Intelligente Chips an zwei Drähten

Der I²C-Bus genießt unter Elektronik-Entwicklern einen guten Ruf, ermöglicht er es doch als Standard-Industrie-Bus, dass Mikrocontroller und andere intelligente Komponenten, die über eine I²C-Schnittstelle verfügen, auf einfache und ökonomische Weise miteinander kommunizieren können.
Eines der wohl typischsten Beispiele hierfür ist die Anbindung eines seriellen EEPROMs an einen Mikrocontroller per I²C-Bus. Ein großer (ökonomischer) Vorteil dieses Bustyps und seines Busprotokolls ist, dass ein steuernder Mikrocontroller lediglich zwei I/O-Leitungen benötigt, um eine große Anzahl von intelligenten I²C-Komponenten zu erreichen. Derartige Komponenten gibt es in großer Anzahl und Vielfalt, als da u. a. wären: die erwähnten EEPROMS, analoge Sensoren wie z. B. Temperatursensoren, A/D-D/A-Wandler, Echtzeituhren, I/O-Bausteine und LCD-/LED-Treiber.
Natürlich liegt es nahe, solche Bausteine ob ihrer vielseitigen Einsatzmöglichkeiten auch an einen PC anzubinden, nur dem fehlt halt eine dem Anwender zugängliche I²C-Schnittstelle! Dieses Manko auszugleichen, ist die Aufgabe unseres kleinen Interfaces. Es setzt eine USB-Schnittstelle mittels eines USB-UART-Wandlers und eines kleinen Mikrocontrollers in eine I²C-Schnittstelle um und umgekehrt. Die bis zu 128 Geräte am I²C-Bus können nun über einen relativ einfachen Befehlsalgorithmus direkt mit dem PC kommunizieren. Auf diese Weise lassen sich sehr unkompliziert auch größere Mess-, Regel- und Steueraufgaben mit I²C-kompatiblen

Technische Daten: USB-I2C-Interface

Schnittstellen:	3 x I²C-Bus (TWI-Bus) inkl. 5-V-Versorgung
Mögliche Bus-Taktfrequenzen:	245 Hz – 400 kHz (Standard-Speed: 100 kHz, Fast-Speed: 400 kHz)
Versorgung von I²C-Komponenten:	5 V/max. 450 mA
Eigene/gesamte Stromaufnahme:	<50 mA/max. 500 mA (High Power USB)
Anschließbare Komponenten:	bis zu 128 Geräte, ICs oder Sensoren
Anzeigeelement:	LED für I²C-Bus-Aktivität
PC-Anbindung:	USB (Kommunikation über virtuellen COM-Port)
Firmware-Update:	über USB möglich
USB-Treiber für:	Windows 2000/XP/Vista, Linux, Mac-OS X
Mitgeliefertes Zubehör:	USB-Kabel, 3 Anschlusskabel
Abmessung Gehäuse (B x H x T):	39 x 14 x 50 mm

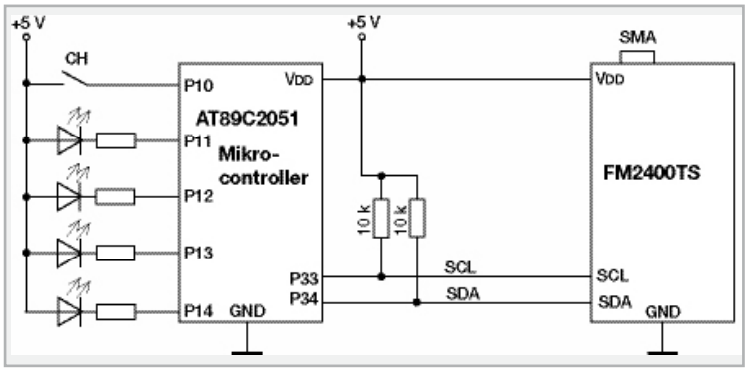


Bild 1: Einsatzbeispiel des I²C-Busses in der Consumer-Elektronik

Komponenten aufbauen.
Bevor wir jedoch zum Interface selbst und dessen Programmierung kommen, wollen wir zunächst die historischen und technischen Grundlagen des I²C-Bussystems betrachten.

Weltstandard I²C

Bereits Anfang der 80er Jahre wurde nach einer einfach beherrschbaren Kommunikationslösung zwischen den immer komplexer werdenden Prozessoren und Peripheriebausteinen in der Radio-und Fernsehtechnik gesucht. Die Philips Semiconductors Division war zu dieser Zeit stark in diesem Markt vertreten und kreierte in den 80er Jahren den seriellen Datenbus I²C (Inter Integrated Circuit Bus) zur Steuerung von Komponenten der Heimelektronik. Seit der ersten Spezifizierung 1992 hat sich dieser Bus als Industriestandard durchgesetzt und dominiert Anwendungen wie die eingangs erwähnten. Vor allem aber ist der Bus das Rückgrat zahlreicher Heimelektronikgeräte, die auf Prozessorlösungen basieren, z. B. Fernsehgeräte, Sat-Receiver, Recorder, aber auch von Steuerlösungen der Haustechnik. Heute sind z. B. nahezu alle Empfangstuner in Fernsehgeräten, Autoradios, Receivern mit einer I²C-Schnittstelle ausgestattet, über die die Frequenzwahl von einem Prozessor aus erfolgt. Selbst in den verbreiteten kleinen 2,4-GHz-ISM-Videosendesystemen ist oft ein I²C-Bus zu finden, nämlich immer dann, wenn für die Auswahl der Kanäle statt eines statischen DIP-Schalters eine einzige Wahltaste benutzt wird. Abbildung 1 zeigt einen Schaltungsausschnitt aus einem solchen System, der die typische I²C-Architektur bereits zeigt: Ein steuernder Controller (der Master), ein Slave-Busgerät (der Tuner), den typischen Zweidraht-Bus und die ebenso typischen Pull-up-Widerstände am Bus. Ganz ähnlich sieht es in den erwähnten kleinen Controllersystemen aus.
Apropos Controller. Hier dominieren in kleinen Embedded-Systemen heute die AVR-Mikrocontroller von Atmel, die kompatible, aber aus lizenzrechtlichen Gründen umbenannte „TWI"-Bus-Schnittstellen (Two-wire Interface) besitzen.

Die Architektur

Zurück zur Technik. Der I²C-Bus ist ein synchroner, serieller Zweidraht-Bus, auf dem mindestens ein Master-Gerät und die adressierten Slave-Geräte mittels eines in Hard- und Software realisierten Protokolls miteinander kommunizieren. Dies erfolgt, je nach Bus-Spezifikation, mit einer maximalen Taktrate von 3,4 MHz, üblich sind auch die Taktraten 100 und 400 kHz. Vorgegeben wird die maximale Taktrate durch den langsamsten Baustein am Bus. Nach unten darf der Takt beliebig reduziert werden, solange der Master dazu in der Lage ist. Ebenso sind mehrere Master (Multimaster) zulässig, wenn diese einen solchen Betrieb unterstützen (mit dem USB-I2C nicht möglich). Die Buslänge darf bei einer maximalen Buskapazität von 400 pF (je Bus-Segment, über Expander erweiterbar) je nach Geschwindigkeit bis zu mehreren Metern betragen. Der Bus besteht aus zwei bidirektional genutzten Leitungen, der Taktleitung SCL (Serial Clock Line) und der Datenleitung SDA (Serial Data Line), die die beteiligten Geräte miteinander verbinden. Abbildung 2 zeigt eine typische I²C-Konfiguration.

So funktioniert's

Der als Master fungierende Baustein, z. B. der in den Abbildungen 1 und 2 gezeigte Mikrocontroller, ist für den definierten Ablauf der Kommunikation zuständig, also Steuerung der Abläufe, Generierung des Taktsignals und Adressierung sowie Datenversand/-empfang. Der Slave empfängt Daten und bestätigt deren Empfang (Acknowledge). Natürlich kann auf Anforderung (Master setzt das Read-Bit in der Adressierung) auch der Slave Daten an den Master senden. Generell ergeben sich daraus vier Betriebsarten am I²C-Bus:
- Senden durch den Master (Master-Transmit)
- Empfangen durch den Slave (Slave-Receive)
- Senden durch den Slave nach Aufforderung durch den Master (Slave-Transmit)
- Empfangen durch den Master (Master-Receive)

Zum Verständnis des Kommunikationsablaufs sind die folgend beschriebenen Buszustände bzw. Abläufe grundlegend. Die Logik-Pegel für die Zustände „high" und „low" sind keine fest definierten Werte, sondern abhängig von der Betriebsspannung des Systems, dabei wird „low" mit <0,3 • VDD und High mit >0,7 • VDD definiert.
Die Grundregel der Datenübertragung ist in Abbildung 3 dargestellt. Hier erkennt man die Funktion der Taktleitung SCL. Ein auf der Datenleitung übertragenes Bit ist nur gültig, solange SCL High-Pegel führt, egal, ob SDA High- oder Low-Pegel führt. Ein Pegelwechsel auf SDA ist nur zulässig, so-

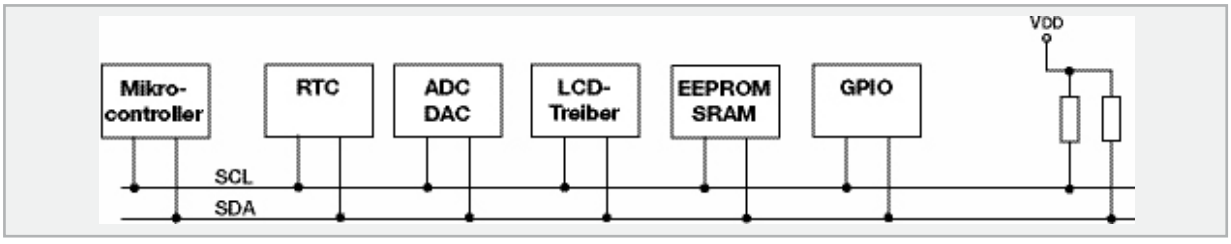


Bild 2: Eine Beispielkonfiguration für verschiedene Komponenten am I²C-Bus

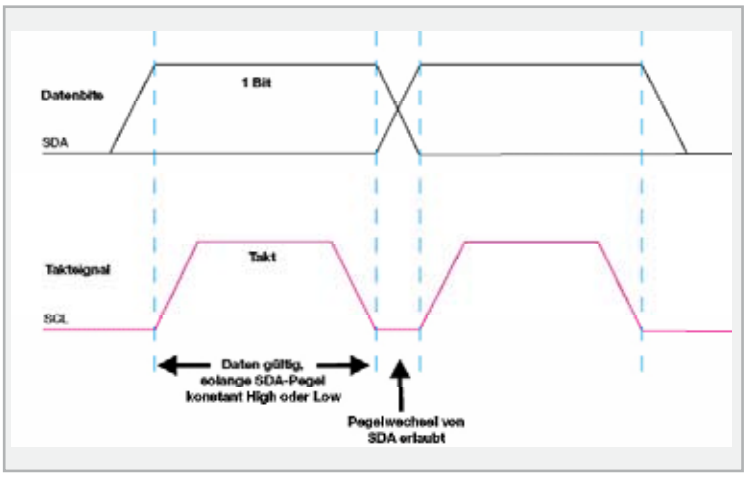


Bild 3: Das grundlegende Zusammenspiel zwischen Daten- und Taktleitung

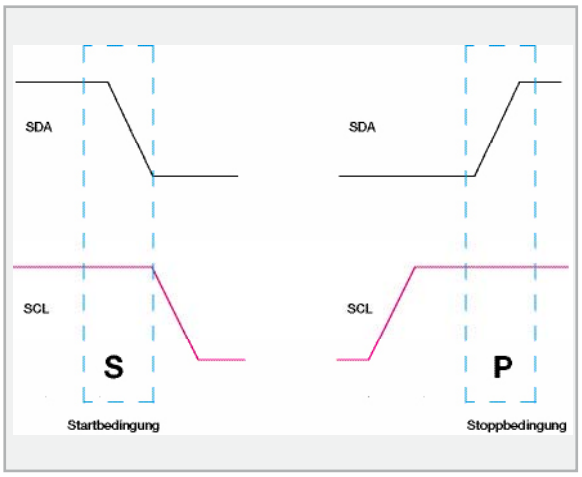


Bild 4: Die Start-/Stopp-Bedingungen am I²C-Bus

lange sich SCL auf Low-Pegel befindet.
Bevor jedoch ein Bit übertragen werden kann, muss den am Bus angeschlossenen Geräten mitgeteilt werden, dass nun gültige Daten folgen. Dazu ist eine Start-Bedingung zu erzeugen (Abbildung 4): SDA muss von „high" zu „low" wechseln, während SCL auf „high" bleibt. Auch für die Kennzeichnung des Abschlusses der Datenübertragung gibt es eine Definition, die Stopp-Bedingung: Wiederum muss SCL auf High-Pegel liegen, während nun aber SDA von „low" auf „high" wechselt.
Eine komplette Datenübertragung (Abbildung 5) erfolgt byteweise und beginnt mit der Adresse des angesprochenen Slave-Gerätes. Das höchstwertigste Bit eines Bytes wird dabei zuerst übertragen (MSB first). Danach quittiert der Slave den Erhalt mit dem Acknowledge-Signal (ACK), indem er, während das nächste Taktsignal vom Master kommt, die SDA-Leitung auf „low" zieht. Er signalisiert damit auch, dass er bereit ist, ein weiteres Byte zu empfangen. Ist der Slave dazu nicht in der Lage oder ist die übertragene Adresse fehlerhaft, so wird die Leitung nicht auf „low" gelegt und somit keine Bestätigung (Not Acknowledge = NACK) versendet. Der Master kann nach der Datenübertragung entweder eine Stopp-Bedingung zum Abschluss des Datentransfers generieren oder eine sogenannte Repeated Start-Bedingung für einen neuen Datentransfer. Dies ist nichts weiter als das Setzen einer erneuten Start-Bedingung ohne vorangegangene Stopp-Bedingung. Damit wissen alle anderen Master am Bus, dass dieser noch nicht für sie frei ist, sie müssen weiter auf das Stopp-Signal warten.
Für den Betrieb mehrerer Master am Bus und den Betrieb unterschiedlich schneller Komponenten gibt es weitere Bedingungen. Näheres hierüber findet der Interessierte unter [1]. Auch unter [2] gibt es eine sehr gute Beschreibung.
Zum Abschluss werfen wir noch einen Blick auf die angewandten Adressierungsmöglichkeiten zur Ansprache der Komponenten. Man unterscheidet hier zwischen der 7-Bit-Adressierung (diese erlaubt bis zu 128 Komponenten am Bus) und der 10-Bit-Adressierung. Letztere ist bei größerer Komponentenanzahl am Bus erforderlich, hiermit sind bis zu 1024 Komponenten am Bus betreibbar. Der Betrieb von Komponenten mit 7- und 10-Bit-Adressierung ist gemischt möglich. Abbildung 6 zeigt den Aufbau der 7-Bit-Adresse mit zwei Übertragungsbeispielen.
Am 10-Bit-Datentransfer-Beispiel in Abbildung 7 kann man erkennen, wie die gemeinsame Existenz von 7- und 10-Bit-Geräten gesichert wird – durch den fest reservierten Adressteil 11110xx (xx sind die ersten beiden Bits der 10-Bit-Adresse) und des zugehörigen R/W-Bits. Erst nach der Slave-Bestätigung (ACK) folgen die weiteren 8 Bit der 10-Bit-Adresse, gefolgt von der nächsten Bestätigung durch die nun vollständig angesprochene 10-Bit-Komponente. Detaillierte Ausführungen zur Adressierung sind ebenfalls unter [1] und [2] nachzulesen.
Die individuelle Adressierung jeder I²C-Komponente ist ab Werk teilweise vorgegeben und ist, falls mehrere gleiche Komponenten am Bus betrieben werden sollen, über eine Hardware-Codierung variierbar. Abbildung 8 zeigt dies am Beispiel des I²C-Temperatursensors LM75.
Dabei wird die Adresse aus einer festen Geräte-ID und einer variablen Adressierung (A0 bis A2) gebildet. Diese wird

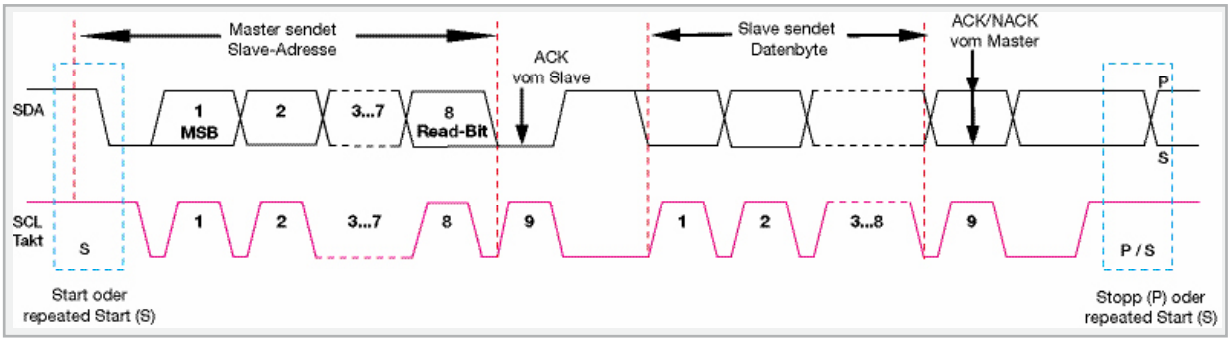


Bild 5: Ein kompletter Datenübertragungszyklus, bei dem der Master vom Slave 1 Datenbyte anfordert und erhält.

Fester Adressteil (Geräte-ID)				Variabler Adressteil			R/W
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	0	1	A0	A1	A2	1/0

Aktivität auf SDA-Leitung:
vom Master
vom Slave

Beispiel: 7-Bit-Adresse und R/$\overline{W}$-Bit von Temperatursensor LM75 und A/D-D/A-Konverter PCF8591

Übertragungsbeispiele 7-Bit-Adressierung:
Master sendet an Slave

START	SLAVE-ADRESSE 1001000	WRITE 0	ACK	DATEN-BYTE	ACK	DATEN-BYTE	ACK/ NACK	STOPP
	Adressbyte = 0x90							

Master liest Daten aus dem Slave

START	SLAVE-ADRESSE 1001000	READ 1	ACK	DATEN-BYTE	ACK	DATEN-BYTE	ACK/ NACK	STOPP
	Adressbyte = 0x91							

Bild 6: Das Prinzip der 7-Bit-Adressierung

START	Fester 10-Bit-Adressteil: 11110	SLAVE-ADRESSE (erste 2 Bit) xx	READ/ WRITE 1/0	ACK 1	SLAVE-ADRESSE (restliche 8 Bit) xxxxxxxx	ACK 2	•••
	1. Adressbyte				2. Adressbyte		

Bild 7: Das Prinzip der 10-Bit-Adressierung

entweder fest verdrahtet oder per Jumper festgelegt. Ausgenommen sind Komponenten wie z. B. Echtzeituhren (RTC), von denen logischerweise nur eine am Bus betreibbar ist. Sie haben eine feste Adresse.
Wollen wir uns nun wieder unserem USB-I2C-Interface zuwenden!

Installation und Bedienung

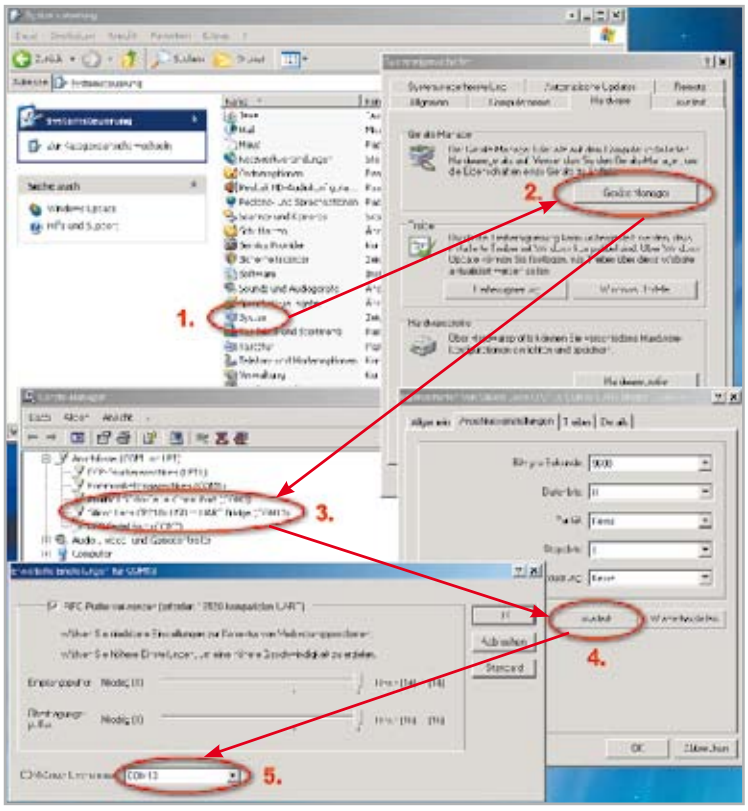


Bild 9: So erfolgt die Kontrolle und ggf. Änderung des zugewiesenen virtuellen COM-Ports unter Windows XP.

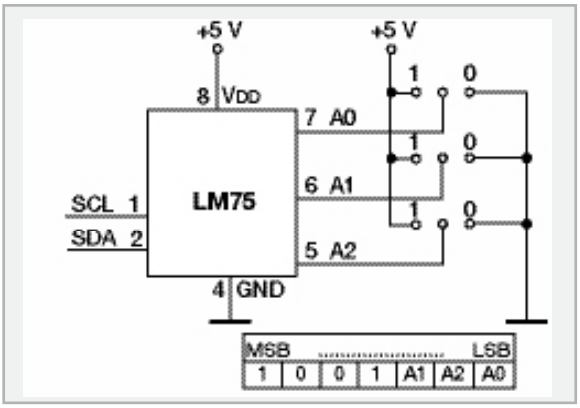


Bild 8: Adressaufbau des LM75 und die Schaltung dazu

Vor dem Anschluss des USB-I2C-Interfaces ist der über einen Download auf der ELV-Produktseite verfügbare Treiber wie im Folgenden beschrieben zu installieren:

1. Silabs-VCP-Treiber (Virtual-COM-Port) installieren
2. USB-I2C-Interface über das beiliegende USB-Kabel an den PC anschließen (vorerst ohne angeschlossene I²C-Hardware)
3. Das Interface wird vom Betriebssystem als neues Gerät erkannt, es öffnet sich der Installationsassistent, dessen Anweisungen zu befolgen sind.
4. Nun ist im Windows-Gerätemanager zu prüfen, welcher COM-Port dem Gerät zugewiesen wurde. Dieser lässt sich im Gerätemanager über: „Eigenschaften"-> „Erweitert..." ändern, siehe dazu Abbildung 9.
5. Schließlich ist ein beliebiges Terminalprogramm (z. B. HTerm oder Realterm, siehe [3] und [4]) zu starten, der zugewiesene COM-Port auszuwählen und mit folgenden Einstellungen zu öffnen:
   115.200 bit/s, 8 Datenbits, 1 Stoppbit, keine Parität, keine Flusssteuerung (Handshake)
6. Nun kann man eigene I²C-Slave-Geräte anschließen und die Kommunikation starten.

Die Kommunikation mit den I²C-Geräten

Wie im I²C-Grundlagen-Teil bereits beschrieben, wird jeder Schreib-/Lesezugriff auf dem I²C-Bus durch eine Start-Bedingung eingeleitet. Dies geschieht beim USB-I2C-Interface durch das ASCII-Zeichen **S**.

Wie in Tabelle 1 dargestellt, folgt danach die Adresse des angesprochenen Slave-Gerätes. Das Adressbyte wird um das Lese/Schreib-Bit ergänzt. Es folgen Daten und/oder weitere Konfigurationsbytes. Abgeschlossen wird die I²C-Kommunikation durch das mit dem Zeichen **P** ausgeführte Stopp-Ereignis.

Mit dem in Abbildung 10 gezeigten Testaufbau wollen wir diesen Ablauf anhand der Ansteuerung des 8-Bit-A/D-D/A-Konverters PCF8591 demonstrieren. Die zugehörige Beispielschaltung ist in Abbildung 11 dargestellt. Der PCF8591 kann direkt am USB-I²C-Interface angeschlossen und zur analogen Datenaufnahme über 4 Kanäle und zur Ausgabe einer analogen Spannung genutzt werden. Die zur Kommunikation nötige Geräteadresse des PCF8591 ist identisch mit der in Abbildung 6 und Abbildung 8 beschriebenen Adresse des LM 75.

Im Beispiel sind A2, A1 und A0 mit GND verbunden, wodurch die jeweiligen Bits auf 0 gesetzt sind.

Es ergeben sich die Leseadresse 0x91 (10010001) und die Schreibadresse 0x90 (10010000). Zur Konfiguration des PCF8591 wird erst die Schreibadresse 0x90 und danach das „Control-Byte" 0x05 übertragen. Dessen genaue Bedeutung kann im Datenblatt [5] nachgelesen werden.

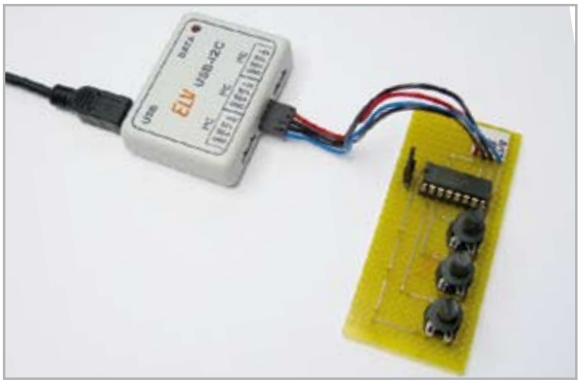


Bild 10: Verwendung des USB-I2C-Interface mit der Beispielschaltung aus Bild 11.

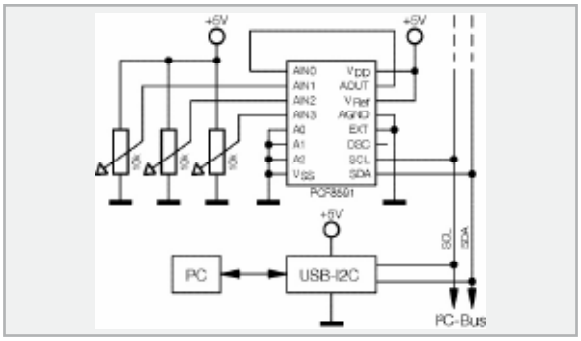


Bild 11: Beispielschaltung für die Anbindung des A/D-D/A-Konverters PCF8591 an das USB-I2C-Interface

Beispiel: A/D-Wandler mit PCF8591

Zum Auslesen der vier A/D-Wandler-Kanäle ergeben sich damit die folgenden Befehlszeichen, die nacheinander als ASCII-Symbole über das Terminal-Programm an das USB-I2C-Interface gesendet werden:

S90 05 R04 P

Die Bedeutung der Befehle:

S 90	05	R 04	P
I²C-Start, Geräte-Adresse + Write-Bit (0)	Control-Byte schreiben	Lese vom zuvor adressierten Gerät 4 Byte	I²C-Stopp
Konfiguration des PCF8591		Lese 4 Byte (4 A/D-Wandlungen)	

Der PCF8591 erhält diese Messanweisung, konvertiert die jeweils anliegende Spannung und gibt diese zurück an das USB-I2C-Interface, das die Messwerte zum PC weiterreicht.

Die Antwort besteht z. B. aus den folgenden ASCII-Zeichen:

A2 EA F5 FF

Die Bedeutung dieser Antwort:

A2	EA	F5	FF
Messwert AIN0	Messwert AIN1	Messwert AIN2	Messwert AIN3

Sehr schön ergänzen lassen sich diese nicht gerade selbsterklärenden Rückgabewerte durch die in Tabelle 2 aufgelisteten Kommentar-Funktionen.

Tabelle 1: Befehle für die I²C-Kommunikation

ASCII-Zeichen	Folgebyte(s)*	Funktions-Beschreibung
S		initiiert Start-Ereignis auf I²C-Bus
	7-Bit-Adresse + Write-Bit **(0)** + Datenbyte(s)	nachfolgende Daten ins adressierte I²C-Gerät schreiben (geringstwertiges Bit [LSB] des Adressbytes muss 0 sein)
	7-Bit-Adresse + Read-Bit **(1)** + Byteanzahl**	Daten aus dem adressierten I²C-Gerät lesen, dem Adressbyte folgt die Anzahl der zu lesenden Bytes (geringstwertiges Bit [LSB] des Adressbytes muss 1 sein)
P		initiiert Stopp-Ereignis auf I²C-Bus (Bus im Leerlauf = idle)
W	Byte1 Byte2 Byte3...	schreibt „Byte1, Byte2, Byte3..." ins zuletzt adressierte I²C-Gerät
R	Byteanzahl**	liest Datenbytes (1...255) aus dem zuletzt adressierten I²C-Gerät
:		wartet mit der Ausführung der nachfolgenden Befehle bis zum nächsten Zeilenumbruch (0x0D oder 0x0A); ist sinnvoll, wenn das Terminal-Programm jedes Zeichen sofort nach der Eingabe überträgt; Ausführung erst nach Abschluss mit Eingabetaste
L	Byte1 Byte2	fügt eine Wartepause von 1 bis 65.535 ms (0001...FFFF) in Hex-Schreibweise (16 Bit) in die Befehlsausführung ein (z. B. innerhalb von Makros)
N		lässt Master nach letztem gelesenen Byte mit NACK antworten, wenn die automatische NACK-Antwort mit Y21 deaktiviert wurde

* Jedes Byte (Hexadezimal) wird mit 2 ASCII-Zeichen geschrieben, z. B.: 0x1F = 1F.

** Wird beim Lesen als Byteanzahl 00 angegeben, erwartet das USB-I2C-Interface einen Pascal-String. Dieser macht es möglich, eine Zeichenkette mit variabler Länge vom Slave auszulesen. Definiert ist die Länge im ersten gelesenen Byte.

Um Kommentare auch zwischen die 4 Datenwerte aus dem vorherigen Beispiel einfügen zu können, muss die Leseroutine in 4 Teilschritte aufgeteilt werden. Das kann z. B. folgendermaßen aussehen:
S90 05 [Wert 0:]R01,. [Wert 1:]R01,. [Wert 2:]R01,. [Wert 3:]R01;P

Die besser verständliche Rückgabe sieht in diesen Fall folgendermaßen aus:
Wert 0: A2,
Wert 1: EA,
Wert 2: F5,
Wert 3: FF;

Beispiel: D/A-Wandler mit PCF8591

Soll mit der vorgestellten Beispielschaltung eine analoge Spannung auf den D/A-Wandler-Ausgang (A_{OUT}) ausgegeben werden, ist eine I²C-Schreibroutine notwendig. Möchte man beispielsweise die halbe Referenzspannung ($V_{ref}/2$ = 2,5 V) ausgeben, ergibt sich die folgende Befehlsfolge:
S90 45 7E P

Die Bedeutung der Befehle:

S 90	45	7E	P
I²C-Start, Geräte-Adresse + Write-Bit (0)	Control-Byte schreiben	$V_{ref}/2$ = 0xFF/2=0x7F	I²C-Stopp

Dabei gibt das USB-I2C-Interface keine Rückmeldung an den PC. Da aber in der Beispielschaltung in Bild 11 die Ausgangsspannung von A_{OUT} gleichzeitig am A/D-Wandler-Eingang AIN0 anliegt, kann sie dort mit der zuvor beschriebenen Mess-Befehlsfolge überprüft werden.

Beispiel Datenlogger mit PCF8591

Ergänzt man die zum Messen geeignete A/D-Wandler-Befehlsfolge um ein paar Befehle und speichert man diese im Makrospeicher des USB-I2C, so wird aus der Kombination USB-I2C-Interface plus PCF8591 ein kompletter analoger Datenlogger, der selbstständig alle 512 ms die Spannungen an den vier A/D-Wandler-Eingängen ermittelt und zum PC überträgt. Wird das Gerät vom USB-Port getrennt und später neu verbunden, so nimmt der Datenlogger selbstständig erneut die Arbeit auf. Für die Datenaufnahme benötigt man lediglich ein einfaches Terminal-Programm, wie z. B. Realterm, das die Daten entgegennimmt und abspeichert. Anschließend können die Daten mit MS Excel oder ähnlichen Programmen ausgewertet und visualisiert werden.
Ein einfaches und sehr kurzes Makro für solch eine Datenloggerfunktion lautet z. B. folgendermaßen:
S90 05 R04 L0200 >00

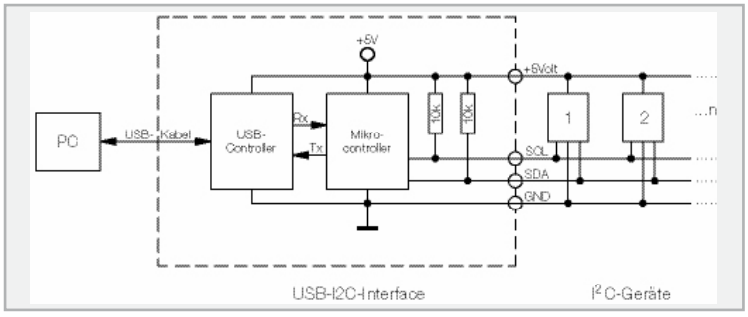


Bild 12: Blockschaltbild des USB-I2C-Interfaces

Die Erläuterung hierzu:

S 90	05	R 04	L 02 00	> 00
I²C-Start, Geräte-Adresse + Write -Bit	Control-Byte schreiben	Lese vom zuvor adressierten Gerät 4 Byte	Warte 512 ms (0x200= 512)	Starte die Ausführung des Makros an Adresse 00

Jetzt muss dieses Makro noch in den Makrospeicher ab Adresse 00 geschrieben werden, was mit dem aus Tabelle 3 entnommenen Befehl V00{...} erfolgt:
V00{ S90 05 R04 L0200 >00 }

Sollen die Messergebnisse als semikolongetrennte Werte direkt in Excel eingelesen werden, so ist zuerst mit dem Konfigurationsbefehl Y01 aus Tabelle 3 der automatische Zeilenumbruch nach jedem Datenbyte abzuschalten. Anschließend müssen Semikolons zwischen die Datenbytes eingefügt und jede Messperiode mit einem manuell eingefügten Zeilenumbruch (Punkt) abgeschlossen werden. Das Ergebnis ist:
Y01
V00{ S90 05 R01;R01;R01;R01. L0200 >00 }

Die Textausgabe sieht dann (mit konstanten Messergebnissen) folgendermaßen aus:
A2; EA; F5; FF
A2; EA; F5; FF
A2; EA; F5; FF
A2; EA; F5; FF

Im Downloadbereich zum USB-I2C-Interface befinden sich weitere Anwendungsbeispiele, die die Verwendung der USB-I2C-Befehle mit I²C-Komponenten wie Echtzeituhr (DS1307), EEPROM (24C02), 8-Bit-I/O-Interface (PCF8574), Thermometer (DS75, LM75, TMP101) demonstrieren.

Tabelle 2: Kommentarbefehle für die Befehls- und Rückgabewerte

ASCII-Zeichen	Funktions-Beschreibung
.	ein Punkt bewirkt einen Zeilenumbruch (0x0D 0x0A) in der Rückgabe
,	fügt ein Komma in die Rückgabe ein (für kommagetrennte Daten für Excel o. Ä.)
;	fügt ein Semikolon in die Rückgabe ein (für semikolongetrennte Daten für Excel)
[...]	ASCII-Zeichen zwischen eckigen Klammern werden zum PC zurückgegeben –> zum Kommentieren von Rückgabewerten Mögliche Steuerzeichen: \r (Carriage Return), \n (Line Feed), \t (Horizontal Tab)
(...)	ASCII-Zeichen zwischen runden Klammern werden ignoriert –> zum Kommentieren von Befehlsanweisungen
Leerstelle	Leerstellen in den Anweisungen werden ignoriert (ausgenommen innerhalb von eckigen Klammern)

Tabelle 3: Befehlsübersicht zur Konfiguration des USB-I2C-Interfaces

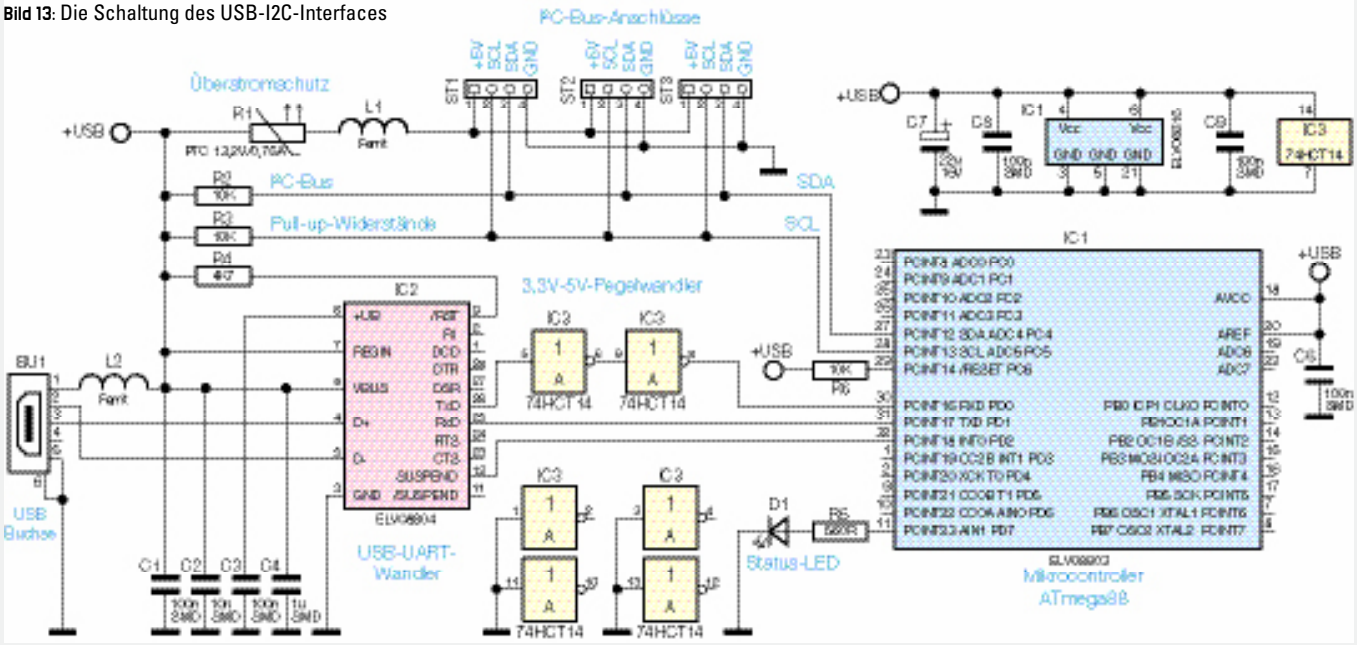
ASCII-Zeichen	Folgebyte(s)*/Zeichen		Funktions-Beschreibung
>	Makroadresse (1 Byte)		startet Makro-Ausführung an der Makro-Speicheradresse
<			beendet Makro-Ausführung und wartet auf neue Anweisungen vom PC
V	Makroadresse { Zeichen1 Zeichen2... }		schreibt die ASCII-Zeichen (Befehle und Daten) zwischen den geschweiften Klammern in den Makro-Speicher ab der übergebenen Makroadresse; Makrospeicher löschen mit: **V00{}** (Speicher wird mit Leerzeichen (Hex:0x20) überschrieben)
U			Inhalt des Makrospeichers (256 ASCII-Zeichen) wird vollständig ausgegeben (zum PC)
?			Systemstatus und Einstellungen werden ausgegeben (zum PC)
T	I²C-Taktrate (6 Zeichen)		I²C-Bustakt von min. 226 Hz bis max. 409,6 kHz (000226...409600), z. B.: 400000 = 400 kHz (Fast-Mode), **100000 = 100 kHz # (Standard-Mode)**, 000226 = 226 Hz (geringst mögliche Frequenz)
X	Baudrate (4 Zeichen)		COM-Port Baudrate: 0048 = 4800 bit/s, 0096 = 9600 bit/s, 0192 = 19.200 bit/s, 0384 = 38.400 bit/s, 0576 = 57.600 bit/s, 0768 = 76.800 bit/s, **1152 = 115.200 bit/s #**, 2304 = 230.400 bit/s (diese Einstellung wird erst nach einem Neustart wirksam)
Y	0	0 #	dem letzten Datenbyte, das der Master aus dem Slave liest, folgt ein Zeilenumbruch (0x0D 0x0A) in der Rückgabe zum PC
		1	Daten, die der Master aus dem Slave ausliest, folgt **kein Zeilenumbruch** in der Rückgabe zum PC
	1	0 #	Master **ignoriert** beim Schreiben die **NACK**-Antwort des Slaves (die bedeutet, dass der Slave keine weiteren Daten akzeptiert) und schreibt weiter
		1	Master stoppt Schreiben, wenn Slave mit NACK antwortet
	2	0 #	nachdem der Master das letzte Byte vom Slave gelesen hat, antwortet er mit NACK (dadurch weiß der Slave, dass keine weiteren Daten folgen)
		1	beim Lesen sendet Master **kein NACK** nach letztem Byte (in diesem Fall ist mit der N-Anweisung manuell NACK zu senden)
	3	0 #	ACK/NACK-Antworten des Slaves werden nicht zum PC übertragen
		1	beim Schreiben wird für jedes erhaltene **ACK** ein **K** zum PC übertragen und für ein **NACK** ein **N** (gut zum Debuggen)
	4	0 #	jeder Daten-Rückgabe (2 Zeichen) folgt ein Leerzeichen (0x20)
		1	einer Daten-Rückgabe zum PC folgt kein Leerzeichen
	5	0 #	nach einem Reset das Makro ausführen, wenn eines im Makrospeicher steht
		1	nach einem Reset kein Makro ausführen
	6	0 #	die Makroadresse (nach > und V) wird mit 2- und die Wartepause (nach L) wird mit 4-ASCII-Zeichen in Hexadezimalschreibweise angegeben
		1	die Makroadresse (nach > und V) wird mit 3- und die Wartepause (nach L) wird mit 5-ASCII-Zeichen in Dezimalschreibweise angegeben
	7	0 #	(reserviert für zukünftige Erweiterungen)
		1	(reserviert für zukünftige Erweiterungen)
Z	4B		startet das USB-I2C-Interface neu (Reset); der Inhalt des Makrospeichers bleibt erhalten
	AA		Konfiguration, Baudrate und Bustakt auf Auslieferungszustand zurücksetzen, Makrospeicher löschen und USB-I2C-Interface neu starten

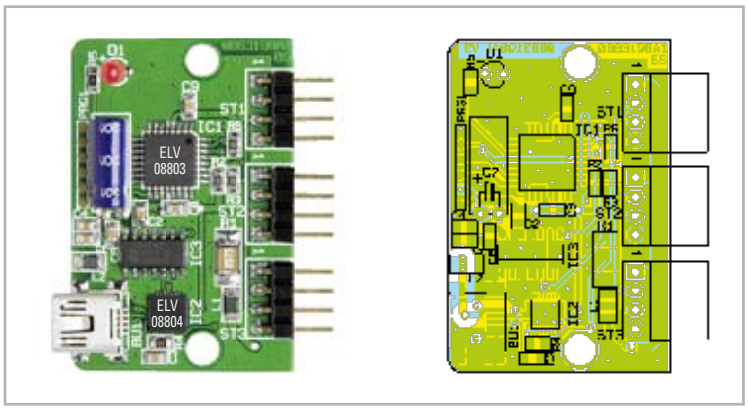
* Jedes Byte (Hexadezimal) wird mit 2 ASCII-Zeichen geschrieben, z. B.: 0x1F = 1F # Standardwert im Auslieferungszustand

Schaltungsbeschreibung:

Das USB-I2C-Interface ist in Abbildung 12 als Blockschaltbild zu sehen. In Abbildung 13 ist die detallierte Schaltung dargestellt. Ihr Herzstück ist der Mikrocontroller IC 1, ein ATmega88 mit integrierter TWI-Schnittstelle (wie erwähnt, die Atmel-Entsprechung für I²C). Der I²C-Bus ist mit je einem Pull-up-Widerstand für die SDA- und SCL-Leitungen di-

Bild 13: Die Schaltung des USB-I2C-Interfaces





Ansicht der fertig bestückten Platine des USB-I2C-Interfaces mit zugehörigem Bestückungsplan

rekt am IC 1 angeschlossen. Die Spannungsversorgung der Schaltung erfolgt über den USB-Bus (USB-powered). Dabei sind bis zu 500 mA verfügbar, da sich das USB-I2C-Interface als High-Power-Gerät am PC anmeldet. So können hierüber auch die angeschlossenen I²C-Geräte mit bis zu 450 mA versorgt werden.

Der UART-USB-Treiberbaustein IC 2 von Silabs (CP2102) beinhaltet alle für die USB-Schnittstelle wichtigen Komponenten. Er kommuniziert per UART mit IC 1.

Der TxD-Ausgang von IC 2 zum Mikrocontroller führt einen 3,3-V-High-Pegel, was unter Umständen für den mit 5 V betriebenen ATmega88 zu gering sein kann, weshalb als Pegelwandler zwei CMOS-Inverter (IC 3) dazwischengeschaltet sind. Die 5-V-Eingangsspannung von IC 1 ist für den CP2102 (IC 2) dagegen kein Problem.

Stückliste: USB-I2C-Interface

Widerstände:

560 Ω/SMD/0805	R5
4,7 kΩ/SMD/0805	R4
10 kΩ/SMD/0805	R2, R3
Polyswitch, 13,2 V, 0,75 A, SMD, 1812	R1

Kondensatoren:

10 nF/SMD/0805	C2
100 nF/SMD/0805	C1, C3, C6, C8, C9
1 µF/SMD/1206	C4
22 µF/16 V	C7

Halbleiter:

ELV08803/SMD/Hauptkontroller	IC1
ELV08804/SMD/USB-Controller	IC2
74HCT14/SMD	IC3
LED, 3 mm, Rot	D1

Sonstiges:

Chip-Ferrit, 1206, 80 Ω bei 100 MHz	L1, L2
USB-B-Buchse, mini, 5-polig, winkelprint, liegend, SMD	BU1
Stiftleiste, 1 x 4-polig, winkelprint	ST1–ST3
3 konfektionierte Kabel mit Crimp-Buchse, 4-polig, 20 cm	
1 USB-Kabel (Typ A auf Typ B mini), 2 m	
1 Kunststoffgehäuse, Grau, komplett, bearbeitet und bedruckt	

Der I²C-Bus ist parallel an 3 Stiftleisten ST 1, ST 2 und ST 3 geführt. Diese sind auch mit der 5-V-USB-Spannung verbunden. Als Sicherheitselement zur Absicherung des USB-Ports gegen eine zu hohe Stromaufnahme dient R 1 – ein PTC-Sicherheitselement, das im Fehlerfall den Strom an ST 1 bis ST 3 begrenzt.

Die Ferrite L 1 und L 2 dienen zur Unterdrückung von Störungen, die auf der Spannungsversorgungsleitung auftreten könnten.

Nachbau

Der Aufbau des Interfaces ist schnell erledigt, da alle SMD-Bauteile bereits ab Werk bestückt sind.

Es bleibt noch die Bestückung weniger bedrahteter Bauteile. Dabei beginnen wir mit ST 1/2/3, die wie im Platinenfoto zu sehen, zu bestücken und auf der Platinen-Lötseite zu verlöten sind. Dabei sind die Lötstellen sorgfältig auszuführen und es ist darauf zu achten, dass die Kunststoffträger der Stiftleisten plan auf der Platine aufliegen.

Es folgt der Elko C 7, der polrichtig zu bestücken ist, nachdem man zuvor seine Anschlüsse abgewinkelt hat. Die Polungsmarkierung des Elkos erfolgt üblicherweise am Minuspol.

Abschließend ist D 1 zu bestücken. Auch hier ist auf die richtige Polung zu achten, der längere Anschluss ist die Anode (+). Die LED ist so einzulöten, dass sich eine Einbauhöhe von 11 mm von der Platinenoberfläche bis zur LED-Spitze ergibt.

Nach einer abschließenden Kontrolle auf Löt- und Bestückungsfehler erfolgt der Einbau in das Gehäuse. Dazu wird die Platine in die Gehäuseunterschale eingelegt, so dass die Buchsen exakt in den vorbereiteten Ausschnitten liegen. Nun ist die Gehäuseoberschale aufzulegen und mit den beiliegenden zwei Schrauben zu verschrauben.

Für den Anschluss eigener I²C-Slave-Komponenten an das Interface liegen dem Bausatz drei Buchsen mit fertig konfektionierten Kabeln bei. Diese haben auf der Oberseite einen Verpolungsschutz, der genau in die Gehäuseöffnungen passt. Die Belegung der Adern ist direkt auf das Gehäuse aufgedruckt (rotes Kabel = +5 V, blaues Kabel = GND).

Hierüber kann ein einfacher Anschluss an das Interface erfolgen und der Kommunikation zwischen PC und I²C-Geräten steht nichts mehr im Wege! ELV

Internet:
[1] http://www.nxp.com/acrobat_download/usermanuals/UM10204_3.pdf
[2] http://www.roboternetz.de/wissen/index.php/I²C
[3] http://realterm.sourceforge.net/#I²C%20Bus
[4] http://www.der-hammer.info/terminal/index.htm
[5] http://www.nxp.com/acrobat/datasheets/PCF8591_6.pdf

Entsorgungshinweis

Gerät nicht im Hausmüll entsorgen!
Elektronische Geräte sind entsprechend der Richtlinie über Elektro- und Elektronik-Altgeräte über die örtlichen Sammelstellen für Elektronik-Altgeräte zu entsorgen!