Die I²C-Schnittstelle

Inhalt:

I²C mit M051

Das Modul stellt neben dem Scanner- und dem R232-Interface auch zwei unabhängige I²C-Kanäle („Busse“) zur Verfügung. I²C und auch die beiden anderen Komponenten können nicht separat aktiviert werden; es gibt nur ein/aus für das gesamte M051:

`%SWITCH m n` wobei m=Schacht-Nr. und n=1 → Modul ein, n=0 → Modul aus

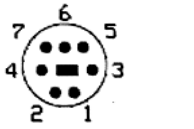
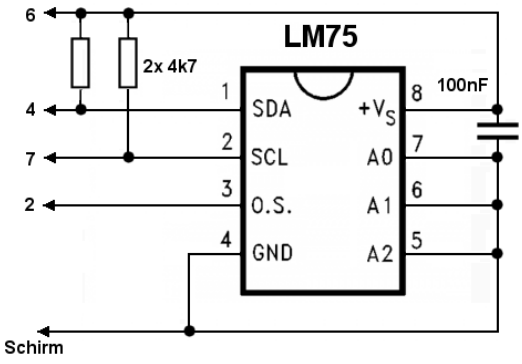
Mit einem „Nur-Scanner-M051“ - ohne bestückten Mikrocontroller, dafür aber mit den * markierten Brücken - ist eine Arbeit mit RS232 und I2C nicht möglich!

Slave-Anschluss

Beide I2C-Busse sind auf eine spezielle 7-polige Mini-DIN-Buchse gelegt, an welcher der Slave per Kabel angeschlossen wird (d.h. I²C dient hier nicht zur geräteinternen Kommunikation). Unmittelbar am Slave (Kabelende) sind zwei Pullup-Widerstände nötig.

Die I²C-Schnittstelle des M051 ist für 5V-Betrieb ausgelegt. Viele I²C-Bausteine lassen sich im Bereich von 3...5V betreiben.

Das M051 kann auch die Betriebsspannung für die Slaves liefern. Sie liegt immer an Anschluss (6) an, auch wenn das Modul M051 per Software abgeschaltet ist (zu beachten bei „power-up“-Vorgängen des Slaves!). Maximal zulässiger entnehmbarer Strom: ca. 200 mA (Sicherung 250mA im M 051 vorhanden).

J3, I²C-BUS Sicht auf Buchse  1 = /INT1 2 = /INT0 3 = SDA1 4 = SDA0 5 = SCL1 6 = + 5 V 7 = SCL0 Schirm = GND	Die Kabellänge zwischen M051 und Slave ist von der Güte des Kabels abhängig. Dieses soll möglichst kapazitätsarm sein. Als Richtwert für sichere Funktion kann ca. 2 m bei Verwendung von USB-Kabel (ca. 90pF/m) gelten. 10m bis zu einem Außenfühler ist vermutl. zu viel...	
Buchsenbelegung	Kabelinfos	beispielhafter Anschluss eines Temperatursensors

Der Datenverkehr zwischen dem Anwenderprogramm mit U880 im KC85/3...5 und dem Slave wird von einem Mikrocontroller im M051 gesteuert. Dieser nimmt uns das serielle „Bitschubsen“ auf den Leitungen SDA und SDL, d.h. sowohl die Start-/Stopp-Sequenzen und Bestätigungen als auch die eigentliche Datenübertragung mit dem nötigen Timing ab. In einem „per Hand“-Modus kann der Nutzer jedoch auch dieses selbst realisieren. Dazu gibt es die Möglichkeit, die Steuersignale bzw. Signalbedingungen *START, STOPP*, *ACK* und *NACK* je nach Erfordernis selbst zu setzen, siehe hier.

Der Mikrocontroller wird vom U880 über zwei Peripherieadressen für I²C angesprochen:

Port #27: I²C-Befehle (Betriebsart des Controllers setzen)
Port #25: I²C-Daten (eigentlicher Datenverkehr mit dem Slave)

Controller-Befehle

Nachstehende Übersicht gilt für Firmwareversion 1.5 (bzw. 1.6, siehe unten). Die Beschreibung weicht daher etwas von der in den KC-News veröffentlichten „Urversion“ ab.

Modusauswahl

Der Steuerbefehl umfasst ein bis drei Bytes und ist vom U880 an Port #27 zu senden:

Bit	7	6	5	4	3	2	1	0	
1.Byte	0	0							0 = Bus 0 aktiv 1 = Bus 1 aktiv 00 = Einzelbyte-Übertragung 01 = Blockmodus 11 = „per Hand“ 0 = 7-Bit-Adresse 1 = 10-Bit-Adresse 0 = Interrupt für Bus 0 verboten 1 = Interrupt für Bus 0 erlaubt 0 = Interrupt für Bus 1 verboten 1 = Interrupt für Bus 1 erlaubt
[2. Byte	x	x	x	x	x	x	x	x	I-Vektor Bus0 (oder Bus1, wenn kein Bus0)]
[[3. Byte	x	x	x	x	x	x	x	x	I-Vektor Bus1 (wenn auch Bus0)]]

Auf das 1. (Modus-)Byte folgt nur dann ein zweites oder drittes Byte, wenn ein Interrupt benutzt wird: die Interruptvektoren.

Es kann bei einer Übertragung immer nur ein Bus (0 oder 1) aktiv sein. Sind Geräte auf dem jeweils anderen Bus anzusprechen, so ist dieser erst mit einem neuen Steuerbefehl zu aktivieren.

Wird im Modus-Byte eine 7-Bit-Adresse (Normalfall) vereinbart, so ist pro Adressangabe nur eine Schreiboperation nötig. Bei einer 10-Bit-Adresse sind zwei Byte zu senden. Das gilt sowohl für den Gerätetest (siehe dort) als auch die Slave-Ansprache. Der high-Teil (mit dem 10-Bit-Präfix **11110**...) folgt zuletzt.

Die drei möglichen Übertragungsmodi unterscheiden sich darin, wie der Nutzer Einfluss auf den Datenverkehr nehmen kann.

Einzelbyte-Übertragung

Es wird nach dem Senden der Slave-Adresse immer nur ein Datenbyte gelesen bzw. geschrieben. Für ein neues Datenbyte ist der Slave erneut zu adressieren. Es ist nicht möglich, zwei Bytes hintereinanderweg zu lesen (z.B. einen 16-Bit-Wert). Im Gegensatz zu den anderen beiden Modi muss sich der Programmierer hier aber um nichts weiter kümmern.

Beispiel:

```
LD    A,#00       ;Modus “Einzelbyte“
OUT   (#27),A     ;setzen
LD    A,#48       ;Slave-Adresse #48
OUT   (#25),A     ;senden
IN    A,(#25)     ;ein Byte Daten lesen
...
```

Blockmodus

Sollen z.B. ein 16-Bit-Wert oder noch mehr Bytes hintereinander gelesen oder geschrieben werden, kann der Blockmodus benutzt werden. Nach dem Senden der Slave-Adresse ist es möglich, fortlaufend zu lesen oder zu schreiben, je nach Erfordernis des Slaves.

Die Datenübertragung wird nach der gewünschten Anzahl von Bytes beendet, indem man ein STOPP an den Controller (Port #27) sendet und damit den Bus auch wieder freigibt.

Beispiel:

```
LD    A,#02       ;Blockmodus
OUT   (#27),A     ;setzen
LD    A,#48       ;Slave-Adresse #48
OUT   (#25),A     ;senden
IN    A,(#25)     ;ein Byte Daten lesen
;...              ;merken oder verarbeiten
IN    A,(#25)     ;das nächste Byte Daten lesen
;...
;Ende der Übertragung im Blockmodus:
LD    A,#81       ;STOPP
OUT   (#27),A     ;setzen
```

In der Firmwareversion 1.5 des Controllers gibt es noch einen Fehler, der die Arbeit im Blockmodus verhindert. In der neuen Version 1.6 ist das korrigiert.

Ein Update von 1.5 auf 1.6 ist nur für diejenigen Nutzer nötig, die mit dem Blockmodus des I²C-Interface arbeiten wollen. Andere I²C-Funktionalitäten sowie die Scanner- und RS232-Schnittstelle bleiben im Update unberührt.

„Per Hand“

Im Einzelbytemodus setzt der Controller automatisch jeweils ein NACK (beim Lesen:„ich will nichts mehr lesen...“) gefolgt von STOPP („Ende der Übertragung“) bzw. beim Schreiben nur ein STOPP. Der Programmierer muss sich keine Gedanken dazu machen.

Im Blockmodus ergeht beim Lesen nach einem Byte automatisch ein ACK („... ich möchte noch mehr...“), beim Schreiben sendet der Controller alle Bytes fortlaufend (ohne ein STOPP nach jedem Byte).

Anders ist es in diesem „Experten-Modus“, wo der Programmierer alle Steuersignale selbst beeinflussen kann:

Code	Signal	Bedeutung
#80	START	„Beginn einer Übertragung“ (Belegen des I2C-Busses)
#81	STOPP	„Ende einer Übertragung“ (Freigabe des I2C-Busses)
#82	ACK	„weitere Daten folgen...“
#83	NACK	„keine Daten mehr...“

Diese Codes müssen je nach gewünschter Funktion auf Port #27 geschrieben werden, damit der Controller die entsprechenden Signale generiert. Die Datenbytes selbst werden wie üblich über Port #25 gelesen/geschrieben.

Beispiel:

```
LD    A,#06        ;Modus “Per Hand“
OUT   (#27),A      ;setzen
LD    A,#80        ;START
OUT   (#27),A      ;setzen
LD    A,I00I0001b ;Slaveadresse<<1 + RW-Bit=1: es soll danach gelesen werden!
OUT   (#25),A      ;senden
LD    A,#82        ;ACK
OUT   (#27),A      ;setzen
IN    A,(#25)      ;das 1. Datenbyte lesen
;...               ;merken für Verarbeitung...
LD    A,#82        ;ACK
OUT   (#27),A      ;setzen => es soll weiter gelesen werden
IN    A,(#25)      ;das 2. Datenbyte lesen
;...               ;merken für Verarbeitung...
LD    A,#83        ;NACK
OUT   (#27),A      ;setzen => fertig mit lesen
LD    A,#81        ;STOPP
OUT   (#27),A      ;setzen => Zyklus komplett
;...
```

Wichtig:
Beim Senden der Slave-Adresse ist hier auch das RW-Bit händisch zu setzen, d.h. Slave-Adresse 1x Linksschieben und als Bit0 das RW-Bit benutzen.

Interruptbetrieb:

- Ein I2C-Bauelement kann einen Interrupt auslösten, indem es mit einem entsprechenden Steuerausgang die /INT-Leitung des Masters auf low zieht. Diese Ausgänge sind meist als als „open drain“ bzw. „open collector“ ausgeführt. Mehrere Bausteine können daher auch an der /INT-Leitung einfach parallelgeschaltet werden.
- Sollen Interrupts zugelassen sein, so muss dies bei der Modusauswahl im Steuerwort (s.o.) entsprechend vereinbart werden. Um mit Interrupt zu arbeiten, müssen im Anschluss an das Modus-Auswahlbyte ein oder zwei Interrupt-Vektoren übermittelt werden.
- Zusammen mit dem Inhalt des I-Registers (Standard: #01) ergibt sich wie üblich die Adresse des Unterprogramms (#0100... #01FF), welches bei Eingang eines Interrupts abgearbeitet wird und das mit RETI enden muss.
- In der Interruptroutine muss ggf. (bei mehreren Slaves) geprüft werden, welcher Slave die Unterbrechung angefordert hat.

Gerätetest

Mit diesem Controller-Befehl wird ein I²C-Gerät adressiert und ein Test auf dessen Vorhandensein durchgeführt. Es ist dazu vom U880 eine 2-Byte-Befehlsfolge (bzw. 3 Byte bei 10-Bit-Adressierung) auf Port #27 zu schreiben:

	7	6	5	4	3	2	1	0	
1. Byte:	0	1	0	0	0	0	0	0	„Test“
2. Byte:	x	x	x	x	x	x	x	x	I2C-Adresse low
[3. Byte:	x	x	x	x	x	x	x	x	I2C-Adresse high]*

*) nur wenn vorher der Modus auf „10-Bit-Adresse“ eingestellt wurde

Das RW-Bit ist hier (im Gegensatz zum „per Hand“-Modus) nicht Bestandteil der Adresse, es wird vom Controller beim nachfolgenden Schreiben/Lesen von Daten automatisch erzeugt!

Das anschließende Lesen von Port #27 ergibt, ob ein Gerät auf dieser Adresse existiert:
Rückgabewert : #01 → vorhanden
#FF → fehlt

Info

Mit zwei aufeinanderfolgenden Leseoperationen des U880 auf Port #27 können aktuell eingestellter Übertragungsmodus und Firmwareversion des Mikrocontrollers ausgelesen werden:

1. Byte: aktuelle Moduseinstellung
2. Byte: Versionsnummer: #15 für Version 1.5

Es müssen immer beide Werte ausgelesen werden, auch wenn nur der Modus benötigt wird!

Damit kann auch festgestellt werden, ob überhaupt ein Controller bestückt ist. Ist dieser nicht vorhanden, so wird in beiden Bytes #FF geliefert.

I²C-Datenoperationen

Wurde der Controller in den gewünschten Übertragungsmodus gebracht und ggf. das Vorhandensein des Slaves getestet, so können Daten ausgetauscht werden.

Zum Schreiben oder Lesen von Daten durch den Master ist zunächst der Slave zu adressieren. Im Normalfall (7-Bit-Adressierung) reicht dafür ein Ausgabebefehl mit der Slaveadresse auf Port #25. Außer im „per Hand“-Modus ist das RW-Bit dabei nicht Bestandteil der Adresse!

Anschließend werden auf dem gleichen Port die Daten verschickt oder abgeholt.

```
OUT   (#25),A      => Schreiben von Daten
IN    A,(#25)      => Lesen von Daten
```

Die nötige Reihenfolge der Operationen bestimmt der konkrete Slave bzw. die gewünschte Funktion. Meist folgt auf die Slave-Adresse zunächst die Auswahl eines Registers (Schreiben der Registernummer), anschließend dessen Daten (Lesen oder Schreiben). Dafür ist das Datenblatt des Slaves zu Rate zu ziehen.

Soll nach dem Lesen von Daten aus einem Register nochmals auf das gleiche oder auch auf ein anderes Register zugegriffen werden, so ist im Standard-Einzelbytemodus der Slave erneut mit seiner Adresse (ggf. gefolgt von einer Registerauswahl) zu anzusprechen.

Beispiel:

```
LOOP: LD    A,#48       ;SLAVE
      OUT   (#25),A     ;ADRESSIEREN
      IN    A,(#25)     ;WERT AUSLESEN
                        ;VERARBEITEN...
      JR    LOOP        ;NOCHMAL GLEICHEN WERT...
```

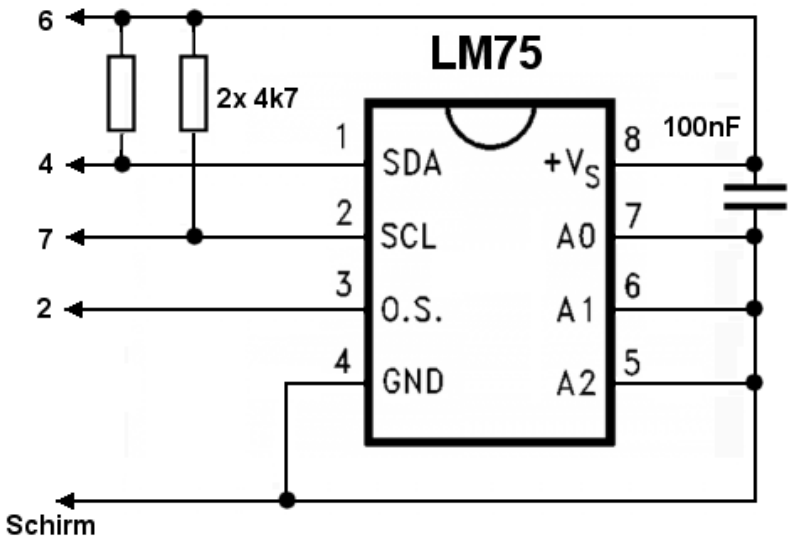
Müssen mehrere Bytes hintereinander verarbeitet werden (z.B. zwei Bytes lesen für einen 16-Bit-Wert eines Registers), so ist bei der Modusauswahl entweder der „Blockmodus“ (`00xxx0Ix`) oder der „per Hand“-Modus (`00xxxIIx`) zu benutzen.

Bei seriellen EEPROMs z.B. ist das gesetzte „Register“ gleich der anfänglichen Speicheradresse (zwei Bytes). Diese wird nach dem Zugriff vom EEPROM immer automatisch inkrementiert, sodass nach einer Adressierung (Slaveadresse + Speicheranfangsadresse) der Speicher (im Blockmodus) fortlaufend gelesen werden kann.

Anwendungsbeispiel: Temperaturmessung

Hardware

Der nachfolgend beispielhaft verwendete Thermosensor ist als Slave mit SDA und SCL am Bus 0 angeschlossen und wird auch über die Buchse mit Spannung versorgt.

I²C-Thermo-Sensor LM75	• Temperaturbereich -55 °C und +125 °C in 0,5 °C Schritten • Genauigkeit ±2 °C • Es gibt zwei Ausführungen, die beide im Bereich 3,3...5V arbeiten. Bei 5V Betriebsspannung ist zweckmäßigerweise die 5V-Ausführung zu benutzen (liefert genauere Ergebnisse). • Ruhestromaufnahme: max. 1 mA • nur im SO-8 Gehäuse • Datenblatt z.B. hier: http://cdn-reichelt.de/documents/datenblatt/A400/lm75.pdf
Anschluss	 (O.S.) = Alarmausgang (für Interrupt) wird hier aktuell nicht verwendet
Pullup-Widerstände	Die beiden Pullup-Widerstände sind immer dann nötig, wenn (längere) Kabel benutzt werden. Deren Wert darf bei 5V minimal ca. 1,5 kOhm betragen. Der Maximalwert hängt von der Kabelkapazität ab (vgl. z.B. hier). Wird der Sensor direkt an/in einem Mini-Din-7-Stecker am Modul 051 eingesetzt, können sie entfallen (interne Pullup-Widerstände vorhanden).
Kabel	Ein im Test benutztes ungeschirmtes 6pol. „Wald- und Wiesen“-Kabel von ca. 1m Länge brachte eine Kapazität von ca. 50pF zwischen zwei Adern. Mit 4,7kOhm Pullup-Widerständen funktionierte das sicher.
Slave-Adresse	0 I 0 0 I A2 A1 A0 A0, A1 und A2 liegen auf Masse, d.h. die Hardwareadresse ist 0. Damit ergibt sich die Slave-Adresse zu #48. Mit den drei Bits Hardwareadresse könnten maximal acht LM75 parallel an einem Bus arbeiten. Die Abfrage der einzelnen Meßstellen würde dann mit den jeweiligen Slaveadressen (#48 ... #4F) erfolgen.

Software

Die softwaremäßige Bedienung hängt immer von den Spezifikationen des jeweiligen I2C-Baulementes ab. Ohne das jeweilige Datenblatt geht gar nichts...

Organisation LM75

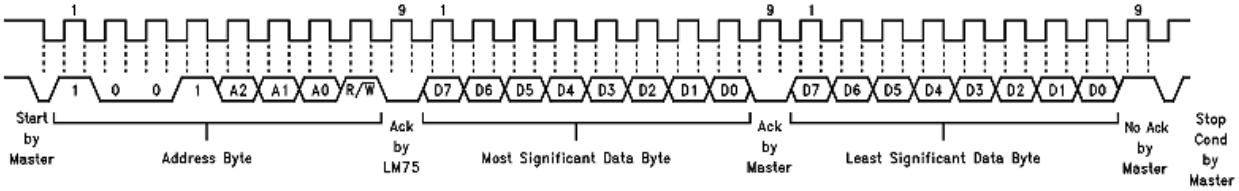
Der Sensor-Baustein LM75 hat vier Register, welche über einen Zeiger (→Registernummer) angesprochen werden:

Nr.	Bedeutung
#00	Temperatur
#01	Konfiguration
#02	unterer Temperaturwert (für Abschalten des Alarms)
#03	oberer Temperaturwert (Alarm)

Gem. Datenblatt erfolgen die einzelnen Zugriffe wie folgt:

Funktion	nötige Operationen		
Zeiger setzen	Schreiben Slaveadresse	Schreiben Zeigerbyte	
Registerinhalt schreiben		Schreiben Zeigerbyte	Schreiben Datenbyte
aktuelles Register lesen		Lesen Datenbyte	

Benötigt wird für diese Demo eigentlich nur Register #00, das den Temperaturwert enthält. Nach dem Einschalten steht der Zeiger schon auf #00, es kann nach der Adressierung des Slaves also sofort die Temperatur ausgelesen werden:



(a) Typical 2−Byte Read From Preset Pointer Location Such as Temp, T_{OS}, T_{HYST}

Das Temperatur-Register #00 enthält 16 Bit:

- Im zuerst gelesenen Byte (MSDB) steht der Vorkomma-Wert der Temperatur als HEX-Zahl. Ist die Temperatur negativ, so erfolgt eine Darstellung im 2er Komplement (d.h. z.B. #FF = -1°C)
- Im zweiten Byte (LSDB) stehen die 10tel Grade, entweder #00 für 0,0 °C oder #80 für 0,5 °C. Ob die Zehntelgrade bei einer Genauigkeit von ±2 °C unbedingt ausgelesen und angezeigt werden müssen, kann jeder selbst entscheiden... ☺
- Sollen auch die Zehntelgrad ausgelesen werden, so kann das nicht im Einzelbyte-Modus erfolgen. Für zwei aufeinander folgende Bytes ist der Blockmodus oder der „per Hand“-Modus zu benutzen. Wegen des o.a. Problems in der Controllersoftware 1.5 (Fehler im Blockmodus) habe ich zunächst den „per Hand“-Modus erfolgreich getestet.

Anwendungsprogramm

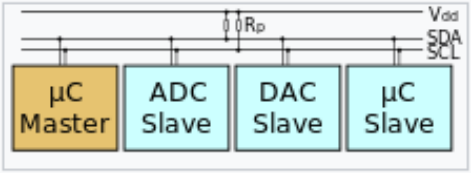
Nachstehende Grund-Befehlsfolgen wurden mit Controller-Firmwareversion 1.5 erfolgreich erprobt.

```
;ÜBERTRAGUNGSMODUS AUSWÄHLEN
LD    A,00000000B       ;BUS 0, EINZELBYTEÜBERTRAGUNG, 7BIT, KEIN INTERRUPT
OUT   (#27),A           ;CONTROLLERBEFEHL SENDEN

;TESTEN, OB SENSOR VORHANDEN:
LD    A,#40             ;CONTROLLERBEFEHL FÜR TEST
OUT   (#27),A           ;SENDEN
LD    A,#48             ;SLAVE-ADRESSE LOW
OUT   (#27),A           ;SENDEN
IN    A,(#27)           ;RÜCKMELDUNG LESEN
CP    #01               ;01: VORHANDEN, #FF: FEHLT
JP    NZ,SFEHLER        ;KEIN SENSOR GEFUNDEN!

;DATENVERKEHR MIT DEM SLAVE (PORT #25)
LOOP:
LD    A,#48             ;SLAVE ADRESSIEREN
OUT   (#25),A           ;

;REGISTER #00 (DEN TEMPERATURWERT) AUSLESEN:
;BEIM EINSCHALTEN IST BEREITS TEMP.-REGISTER 00 AUSGEWÄHLT!
IN    A,(#25)           ;NUR VORKOMMAWERT AUSLESEN (NACHKOMMA NICHT BENUTZT)

;WERT AUFBEREITEN + AUSGEBEN
CALL  #F003
DEFB  #1C               ;AUSGABE TEMPERATUR ALS HEX-BYTE AUF DEM SCHIRM

;ANZEIGEPAUSE MIT TASTENABFRAGE
LD    BC,#2000          ;VERZÖGERUNGSDAUER CA. 1 SEKUNDE
WARTE:
CALL  #F003             ;PV1
DEFB  #0C               ;TASTATURSTATUS
JP    C,ENDE            ;TASTE GEDRÜCKT, PROGRAMM-ENDE
DEC   BC
LD    A,B
OR    C
JR    Z,LOOP            ;SCHLEIFE MIT ERNEUTER SENSOR-ABFRAGE UND AUSGABE
JR    WARTE

;FEHLERTEXT AUSGEBEN
SFEHLER:
;...
ENDE:
;...
RET                     ;RÜCKKEHR ZU CAOS
```

Anlagen

I²C Grundlagen

I²C (inter-integrated-circuit) oder auch **TWI** (two-wire-Interface) genannt, ist ein Interface zur seriellen Kommunikation eines „Masters“ (Rechner, Mikrocontroller) mit einem oder mehreren „Slaves“ (den I²C-Bauelementen). Zwei Leitungen verbinden alle Slaves mit dem Master:
SDA = Daten und SCL = Takt.



Skizze: Wikipedia

Als Slaves können z.B. Thermofühler, Analog-Digital-Wandler u.v.a.m. angeschlossen werden. Welcher der parallelgeschalteten Slaves sich angesprochen fühlen muss, hängt von seiner Adresse ab. Dabei gibt es verschiedene Formen, von der hier nur die 7-Bit-Adressierung beschrieben werden soll. Damit können theoretisch (mit Einschränkungen) bis zu 127 I²C-Geräte parallel an einem Bus angeschlossen sein.

Die konkrete Grund-Adresse bestimmt meist der Hersteller des I²C-Bauelements. Sie wird durch eine Hardwareadresse (Lage des Slaves im Anwender-System, bis zu 3 Bit) ergänzt.

Die Slave-Adresse wird als erstes auf dem Bus verschickt. Es folgt anschließend ein Zugriffsbit. Dieses entscheidet darüber, ob nachfolgend eine Lese- oder Schreiboperation für das I²C-Bauelement durchgeführt werden soll.

Beispiele:

PCF 8583 (Uhr)	I	0	I	0	0	0	A0	RW
LM75 (Temperatursensor):	I	0	0	I	A2	A1	A0	RW

Welche Bytes im Anschluss an die Slaveadresse + Zugriffsbit folgen, das hängt von der Art des I²C-Bauelements und der gewünschten Funktion ab. Um z.B. ein Slaveregister zu beschreiben, muss allgemein folgendes passieren:
1. Senden einer Startsequenz
2. Senden der I²C-Addresse des Slaves
3. Senden des Zugriffsbit „0“ (es soll nachfolgend geschrieben werden)
4. Senden der zu beschreibenden Registernummer
5. Senden der zu schreibenden Registerdaten
6. Senden der Stoppsequenz.

Für diese Aktivitäten muss man die Leitungen SDA und SDL in bestimmter Weise schalten. Der Anwender muss sich um diese Abläufe nur selten kümmern. Es gibt dafür vorgefertigte Bibliotheken (z.B. auch in der Mikrocontroller-Programmierung mit BASCOM), die durch einfache Aufrufe die gewünschten Abläufe mit den nötigen Parametern realisieren.

Obwohl I²C vor allem als *geräteinterne Schnittstelle* und nicht als „Langdraht-Bus“ gedacht ist, kann der Slave auch über eine etwas längere Leitung angeschlossen werden. Die maximal mögliche Leitungslänge zwischen Master und Slave ist vor allem von der Kabelkapazität abhängig.

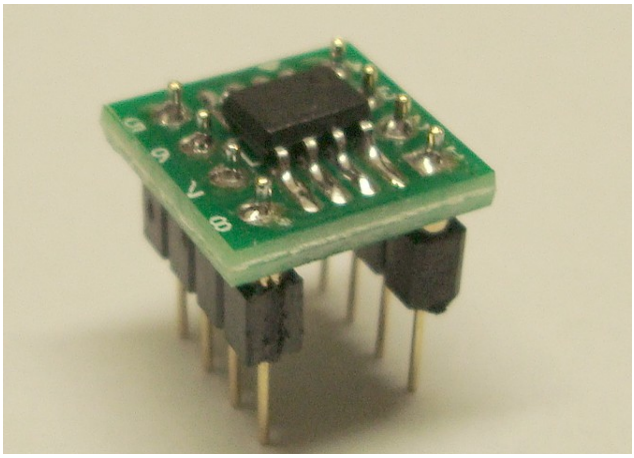
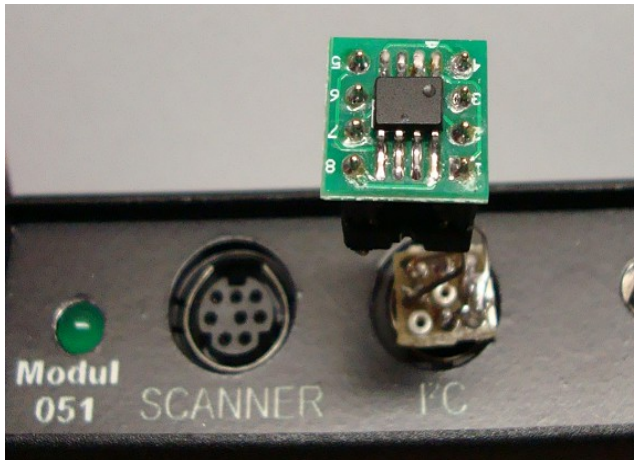
Weitergehende Erklärungen findet man z.B. hier:

http://www.elektronik-magazin.de/page/der-i2c-bus-was-ist-das-21
http://rn-wissen.de/wiki/index.php/I2C

Hinweise

1. Hat man beim Experimentieren mal versehentlich ein falsches Register des LM75 ausgewählt und das aktuelle Programm enthält keine Registerwahl #00 vor dem Auslesen der aktuellen Temperatur, so muss die Betriebsspannung zum Slave kurz unterbrochen werden. Erst dann gilt „Nach dem Einschalten ist Register #00 aktiv“. Ein Neustart des Thermometerprogramms hilft nicht, ebenso kein RESET!
   Dies ist auch von Bedeutung, wenn bei anderen I^2C-Bauelementen eine „power-up“ Funktion existiert/genutzt wird.

2. Da der LM75 als SMD kommt, kann für das Experimentieren (z.B. auch auf dem Steckbrett) ein kleiner Adapter hilfreich sein:

	
Solche Adapterplatinchen gibt es für wenige Cent. Das Auflöten der Pinleisten und des IC (1,27mm Pitch) ist auch kein Thema...	Mangels Mini-Din-7-Stecker wurde der LM75 auf Adapterplatine in eine Fassung gesteckt und diese dann per Drähte (fixiert durch ein Stück Lochrasterplatte) in die I2C-Buchse...

Demo-Programm „I2C-Thermometer“

Screenshot des Demoprogramms

Das komplette Demo-Programm „THERMO.KCC“ belegt aktuell den Adressbereich 0200h...0400h und enthält:

- CAOS-Prolog: Start aus Menü mit „THERMO“
- Prüfung auf Vorhandensein des M051, Ermittlung und Anzeige seines Steckplatzes sowie
- automatisches Ein-/Ausschalten,
- Prüfung ob Controller vorhanden, wenn ja, Ausschrift akt. I2C-Modus und Version,
- Prüfung ob Sensor LM 75 angeschlossen,
- Direktausgabe der aktuellen und der Alarm-Temperaturen als Dezimalzahlen (auch negative Temperaturwerte möglich)
- Ausführung in Endlosschleife, Abbruch mit beliebiger Taste
- Aktualisierung Temperatur ca. jede Sekunde
- Die Programmausführung wird in folgenden Fällen mit Fehlerausschrift abgebrochen:
  - ➔ kein M051 gefunden,
  - ➔ kein Controller eingebaut,
  - ➔ kein Sensor LM75 angeschlossen oder nicht auf Hardwareadresse 0

Quelltext: THERMO.ASM

- ist bewusst nicht optimiert, um nötige Schritte besser zu dokumentieren
- ist reichlich kommentiert
- benutzt Syntax von *ASIde*[1] (Listing lässt sich jedoch leicht an andere Assembler anpassen)

Noch nicht realisiert:

- Setzen der Alarmtemperatur(en)
- Interruptbetrieb. Bei Erreichen der Alarmtemperatur liefert der LM75 an Pin3 ein Signal, welches als Interruptquelle (/INT0) dienen kann. Die Hardware ist bereits dafür vorgesehen.
- Die Software kann vom interessierten Nutzer leicht ergänzt werden...☺

Die Zusammenstellung dieser Doku erfolgte nach „bestem Wissen und Gewissen“ auf der Grundlage der bislang vorliegenden Informationen sowie eigenen Experimenten, ohne Gewähr auf Vollständigkeit und 100%ige Richtigkeit.

Danke für Infos an Enrico Grämer, Mario Müller, René Nitzsche

gefertigt: WeRo
Stand: 01.01.2017

1 siehe z.B. hier: http://www.theeg.de/aside/