4.2 Peripherie ansprechen in C/I2C-Bus

4.2.1 Problem

Das Mikrocontroller-Board soll mit Hilfe eines Sensors vom Typ LM 75A (NXP) die Temperatur erfassen und ausgeben.

Der Sensor ist vom Typ LM 75A hat eine digitale Schnittstelle. Er wird über den I²C-Bus angeschlossen.

4.2.2 Hardware

Da der LM 75A ein SMD-Baustein ist, den man nicht so leicht auf einer Lochrasterplatine festlöten kann, wird ein Breakout-Board von der Firma Horter&Kalb (`http://www.horter.de`) erworben (auf der Website unter dem Namen „I²C-Temperatursensor mit LM75“; aufgeführt). Das RN-Board hat bereits einen Anschluss für I²C-Peripherie[1]. Die Tabelle 1 zeigt die Belegung des Flachbandkabels für die Verbindung zwischen den beiden Platinen.

RN	Bedeutung	I²C
1	SCL	ST1.4
2	GND	ST1.2
3	SDA	ST1.3
4	GND	—
5	+5V	ST1.1

Tabelle 1: Verbindung RN-Board – LM75-Breakout-Board

Die Mikrocontroller der Familie ATmega sind speziell für den I²C-Bus eingerichtet (Register, Beschaltung), so dass die Programmierung vereinfacht wird.

4.2.3 Der I²C-Bus

Der I²C-Bus ist ein serielles Bussystem mit nur zwei Leitungen. Er wird daher oft auch TWI (*two-wire-interface*) genannt. Die beiden Leitungen sind die Taktleitung SCL und die Datenleitung SDA. Dazu braucht ein angeschlossener Baustein natürlich noch die beiden Versorgungsleitungen Vcc (1,1V..+5V) und GND (0V).

Bei der Datenleitung wird positive Logik verwendet, niedriges Potential (weniger als 30% von Vcc)entspricht einer Null und hohes Potential (mehr als 70% von Vcc) einer Eins.

Alle Bausteine liegen mit so genannten Open-Collector-Ausgängen am Bussystem. Der Ruhezustand ist High-Potential bzw. Eins. Der Ausgangs-FET schaltet also nur bei einer Null durch, bei einer Eins ist er hochohmig.

Es handelt sich ja um ein serielles Bussystem. Mit den beiden Leitungen werden also viele Bits übertragen:

- 8 Datenbits, in den Abbildungen rot markiert
- 7 Adressbits (mit Erweiterung auch 10 Bit), in den Abbildungen gelb markiert
- 2 Steuerbits, nämlich R/W (Read/Write) und ACK/NACK (Acknowledge/Not Acknowledge), in den Abbildungen blau
- sowie die START- und STOP-Aktionen, in den Abbildungen ebenfalls blau

[1] Auf dem Board muss J6.5-6 gesetzt werden, damit die I²C-Peripherie mit Spannung versorgt wird.

4.2.3.1 Das Protokoll Am Bussystem gibt es zu jeder Zeit immer nur einen Master, dazu einen oder mehrere Slaves[2].

Nur der Master darf START und STOP sagen. Außerdem gibt der Master den Takt vor. Damit ist die SCL-Leitung unidirektional. Eine Ausnahme ist das so genannte Clock-Stretching, bei dem ein Slave die SCL-Leitung auf Low halten kann, um Zeit zu gewinnen. Die Datenleitung darf von Master und Slave bewegt werden, SDA ist damit bidirektional.

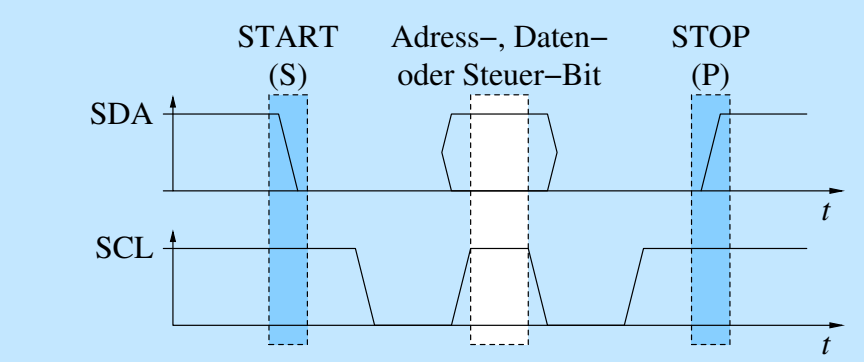

Abbildung 1: Zusammenspiel von SDA und SCL

Eine wichtige Grundregel auf dem I²C-Bus lautet: Wenn SCL High ist, darf niemand SDA bewegen. Es gibt davon nur zwei Ausnahmen, nämlich die START- und die STOP-Aktion:

- Start-Aktion: Wenn SDA von 1 auf 0 (auf aktiv) wechselt, heißt das START.
- Stop-Aktion: Wenn SDA von 0 auf 1 (auf passiv) wechselt, heißt das STOP.

Die Zeiträume, in denen SCL Low ist, dienen dazu, dass die SDA-Leitung auf den richtigen Zustand gebracht wird. Wenn SCL dann auf High schaltet, gilt der Zustand von SDA als ein Bit (Abbildung 1).

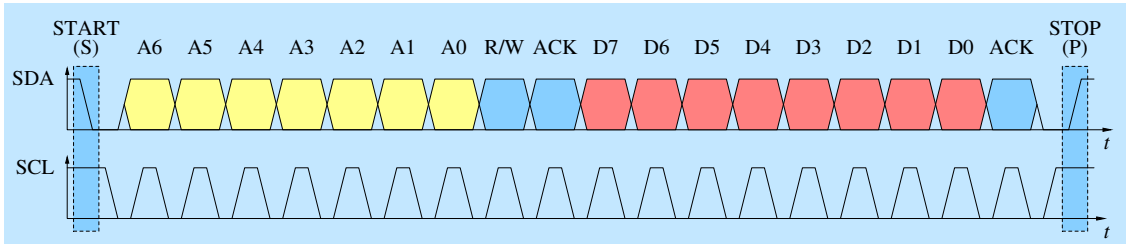

Abbildung 2: Übertragung eines Bytes

Jede Byte-Übertragung (Abbildung 2) beginnt mit der START-Aktion des Masters und endet mit STOP-Aktion des Masters (egal, wann). Zuerst sendet der Master ein START. Dann schickt er die Adresse des Bausteins, der angesprochen wird. Das sind sieben Adressbits, das MSB wird zuerst geschickt. Danach kommt das R/W-Bit. R/W=1 bedeutet, der Master möchte lesen. R/W=0 bedeutet, der Master möchte auf den Baustein schreiben. Daraufhin sendet der Slave ein NACK/ACK (der SCL-Impuls dazu kommt vom Master!). Ein NACK/ACK=1 bedeutet, der Baustein fühlt sich nicht angesprochen oder ist nicht da (passiver Zustand ist High!). Ein NACK/ACK=0 bedeutet, dass der Baustein verstanden hat.

Nun wird das Datenbyte übertragen (je nach Richtung vom Master oder Slave, das MSB zuerst), es folgt dann ein NACK/ACK von der Seite, die das Byte empfangen hat, und zum Schluss kommt ein STOP vom Master.

[2]Es gibt auch Multi-Master-Betrieb, dann darf aber auch immer nur ein Baustein gleichzeitig als Master arbeiten.

4.2.3.2 Beispiel: Master liest ein Byte In Abbildung 3 ist der Verlauf für das Lesen eines Bytes dargestellt. Zur Vereinfachung sind die Zeiträume für die Start-/Stop-Aktionen sowie für die Übertragung der Bits nur noch als Kästchen dargestellt. Die tiefgestellten Kästchen kennzeichnen dabei die Übertragungen, die vom Slave zum Master hin stattfinden.

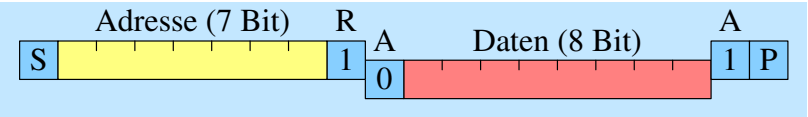

Abbildung 3: Master liest ein Byte

a) START vom Master

b) Master sendet Adresse (7 Bit)

c) Master sendet R/W=1=Read

d) Slave sendet ACK (=0), sonst STOP durch Master

e) Slave sendet Datenbyte (8 Bit)

f) Master sendet NACK (=1), damit der Slave aufhört

g) STOP vom Master

4.2.3.3 Beispiel: Master schreibt ein Byte In Abbildung 4 ist der Verlauf für das Schreiben eines Bytes dargestellt.

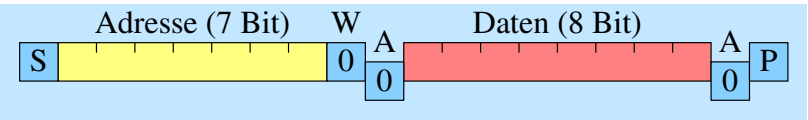

Abbildung 4: Master schreibt ein Byte

a) START vom Master

b) Master sendet Adresse (7 Bit)

c) Master sendet R/W=0=Write

d) Slave sendet ACK (sonst STOP durch Master)

e) Master sendet Datenbyte (8 Bit)

f) Slave sendet ACK (wenn er es verstanden hat)

g) STOP vom Master

4.2.3.4 Beispiel: Master liest zwei Bytes In Abbildung 5 ist der Verlauf für das Lesen mehrerer Bytes dargestellt.

a) START vom Master

b) Master sendet Adresse (7 Bit)

c) Master sendet R/W=1=Read

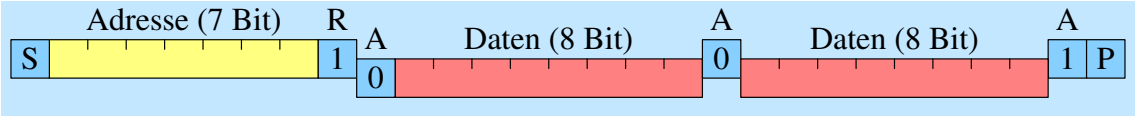

Abbildung 5: Master liest zwei Bytes

d) Slave sendet ACK (sonst STOP durch Master)

e) Slave sendet Datenbyte (8 Bit)

f) Master sendet ACK, damit der Slave weitermacht

g) Slave sendet zweites Datenbyte (8 Bit)

h) Master sendet NACK, damit der Slave aufhört

i) STOP vom Master

4.2.3.5 Beispiel: Master schreibt zwei Bytes In Abbildung 6 ist der Verlauf für das Schreiben mehrerer Bytes dargestellt.

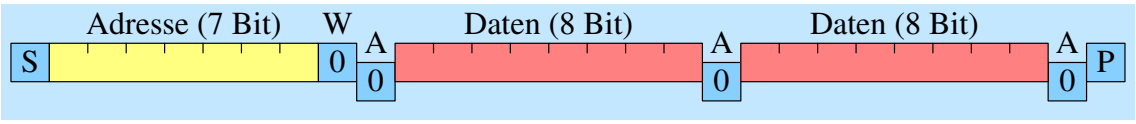

Abbildung 6: Master schreibt zwei Bytes

a) START vom Master

b) Master sendet Adresse (7 Bit)

c) Master sendet R/W=0=Write

d) Slave sendet ACK=0 (sonst STOP durch Master)

e) Master sendet Datenbyte (8 Bit)

f) Slave sendet ACK=0 (wenn er es verstanden hat, sonst STOP)

g) Master sendet Datenbyte (8 Bit)

h) Slave sendet ACK=0 (wenn er es verstanden hat)

i) STOP vom Master

4.2.3.6 Beispiel: Master schreibt erst, liest dann In Abbildung 7 ist dargestellt, dass ein Master zuerst ein Datenbyte schreibt, dann die Lese-Schreib-Richtung umkehrt und aus demselben Baustein ein Byte liest. Man nennt dies den *kombinierten Modus*. In diesem Modus kann der Master bei demselben Baustein in einer Aktion lesen und schreiben. Das ist sinnvoll zum Beispiel bei seriellen Speichern, bei denen zuerst eine interne Speicheradresse geschrieben und anschließend der Inhalt gelesen werden soll. Natürlich ginge das auch mit mehreren Zugriffen, aber mit diesem kombinierten Modus klappt es schneller und zusammenhängend. Für das Umschalten wird die Aktion nicht mit STOP abgebrochen, sonder mit einer wiederholten START-Aktion (REPEATED START, abgekürzt Sr) weitergeführt. Im folgenden Beispiel wird erst ein Byte geschrieben und dann ein Byte gelesen.

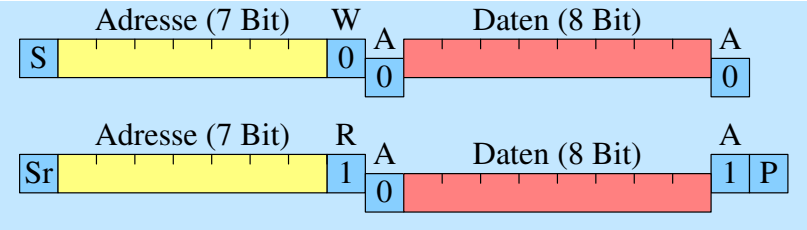

Abbildung 7: Master schreibt erst ein Byte und liest dann ein Byte

a) START vom Master

b) Master sendet Adresse (7 Bit)

c) Master sendet R/W=0=Write

d) Slave sendet ACK=0 (sonst STOP durch Master)

e) Master sendet Datenbyte (8 Bit)

f) Slave sendet ACK=0 (wenn er es verstanden hat, sonst STOP)

g) REPEATED START vom Master

h) Master sendet gleiche Adresse noch einmal (7 Bit)

i) Master sendet R/W=1=Read

j) Slave sendet ACK (sonst STOP durch Master)

k) Slave sendet zweites Datenbyte (8 Bit)

l) Master sendet NACK, damit der Slave aufhört

m) STOP vom Master

Die meisten Bausteine, für die der kombinierte Modus gedacht ist, merken sich die Daten aus dem ersten Zugriff für die Ausführung des zweiten.

4.2.4 Der I²C-Bus am ATmega-Mikrocontroller

4.2.4.1 Schnittstelle Die Mikrocontroller der ATmega-Familie besitzen eine eigene Schnittstelle für den I²C-Bus. In den Unterlagen der Herstellerfirma wird sie (aus Lizenzgründen) als TWI-Schnittstelle bezeichnet.

I²C	ATmega8	ATmega16
SDA	PC4	PC1
SCL	PC5	PC0

4.2.4.2 Taktfrequenz für SCL Zur Einstellung der SCL-Taktfrequenz f_{SCL} im Betrieb als Master dienen das Bitratenregister TWBR und die untersten zwei Bit des Statusregister TWSR. Ihr Wert dient als Vorteiler-Faktor TWPS. Die vollständige Formel für f_{SCL} lautet:

$$f_{SCL} = \frac{f_{CPU}}{16 + 2 \cdot TWBR \cdot 4^{TWPS}} \tag{1}$$

Umgestellt nach TWBR:

$$TWBR = \frac{f_{CPU}/f_{SCL} - 16}{2 \cdot 4^{TWPS}} \tag{2}$$

TWPS dient aber nur dazu, sehr niedrige Taktraten zu realisieren, deshalb kann man mit $TWPS = 0$ die Formel vereinfachen:

$$f_{SCL} = \frac{f_{CPU}}{16 + 2 \cdot TWBR} \tag{3}$$

Und für TWBR:

$$TWBR = \frac{1}{2} \cdot (\frac{f_{CPU}}{f_{SCL}} - 16) \tag{4}$$

Mit $f_{CPU} = 16\,\text{MHz}$, $TWPS = 0$ und $TWBR = 72$ erhält man $f_{SCL} = 100\,\text{kHz}$. Ein kleines Rechenprogramm findet man auf:
`http://www.dieelektronikerseite.de/Tools/TWI-Rechner.htm`.

4.2.4.3 Kontrollregister Das Kontrollregister TWCR ist quasi die Schaltzentrale der TWI-Schnittstelle (Tabelle 2). Die einzelnen Bits haben folgende Bedeutung:

Name									Adresse
	Bit	7	6	5	4	3	2	1	0	
TWCR		TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	SFR 54
	R/W	R/W	R/W	R/W	R/W	R	R/W	R	R/W	
	Init.	0	0	0	0	0	0	0	0	

Tabelle 2: TWCR

a) TWEN=1 (*TWI enable*) schaltet die TWI-Schnittstelle ein, so dass die Anschlüsse nicht mehr normale Ports sind, sondern SDA und SCL.

b) TWINT ist der Startknopf für die TWI-Schnittstelle. Bevor man TWINT setzt, muss man vorher mit den anderen Bits festlegen, welche Aktion es sein soll[3]. Sobald man TWINT gesetzt hat, kann man durch Lesen dieses Bits herausfinden, ob die Aktion fertig ist (0: nicht fertig, 1: fertig)[4].

c) TWSTA=1 sorgt für die Startaktion.

d) TWSTO=1 sorgt für die Stopaktion.

e) TWEA=1 sorgt dafür, dass nach Empfang eines Datenbytes ein ACK ausgegeben wird.

f) Mit TWIE kann man die TWI-Schnittstelle im Interrupt-Betrieb laufen lassen.

g) Mit TWWC kann man Kollisionen erkennen. Es wird hier nicht benötigt.

4.2.4.4 Weitere Register Weiterhin gibt es noch die folgenden Register:

- Das Datenregister TWDR zum Schreiben der Sendedaten und zum Lesen der Empfangsdaten. Sendedaten können zum Beispiel eine Adresse sein oder ein Datenwort.
- Das Statusregister TWSR; die oben genannten Bits TWPS0 und TWPS1 sind mit null initialisiert und brauchen nicht geändert werden. Die anderen Bits geben den Übertragungsstatus an.
- Das Slave-Adress-Register TWAR wird nur gebraucht, wenn der Mikrocontroller als I^2C-Slave arbeitet und dazu eine eigene Adresse braucht oder beim Multi-Master-Betrieb. Es hat nichts mit der Adresse zu tun, die beim Betrieb als Master ausgegeben wird, um einen Slave zu adressieren.

[3] Eventuell müssen auch noch die anderen Register wie TWSR, TWDR richtig gesetzt worden sein.
[4] Im Interrupt-Betrieb ist die Vorgehensweise anders.

4.2.4.5 Aktionen In den folgenden Beispielen soll TWEN=1 sein, dazu TWIE=0.

- START-Aktion ausführen (S oder Sr): TWINT→1, TWSTA→1, Warten auf TWINT=1
- STOP-Aktion ausführen (P): TWINT1, TWSTO→1, *kein* Warten auf TWINT=1 (!)
- Schreiben: TWDR setzen, TWINT→1, Warten auf TWINT=1
- Lesen mit ACK (NACK/ACK=0, weiteres Byte anfordern): TWINT→1, Warten auf TWINT=1
- Lesen mit NAK (NACK/ACK=1, kein weiteres Byte): TWINT→1, TWEA→1, Warten auf TWINT=1

4.2.4.6 Beispiel Hier wird einmal ein Byte von der Adresse 100100_2 gelesen. Das Byte aus Adresse und R/W-Bit ergibt 10010001_2, das ist 0x91.

```
1  int main(void)
2  {
3      DDRA=255;
4      TWCR=255;                                    /* INIT. */
5      TWSR=3;
6
7      TWCR=_BV(TWINT)|_BV(TWSTA)|_BV(TWEN);  /* START */
8      loop_until_bit_is_set(TWCR, TWINT);
9
10     TWDR=0x90+1;                                 /* Lese-Adr. setzen, R/W=1 */
11     TWCR=_BV(TWINT)|_BV(TWEN);                   /* SCHREIBEN */
12     loop_until_bit_is_set(TWCR,TWINT);
13
14     TWCR=_BV(TWINT)|_BV(TWEN);                   /* LESEN MIT NAK */
15     loop_until_bit_is_set(TWCR,TWINT);
16     PORTA=TWDR;                                  /* Ausgeben */
17
18     TWCR=_BV(TWINT)|_BV(TWSTO)|_BV(TWEN);  /* STOP */
19
20     while(1){}
21 }
```

In diesem Beispiel wird nicht abgefragt, ob der Slave ein ACK gesendet hat. Dazu müsste man den Inhalt von TWSR ansehen.

4.2.5 Der I²C-Bus am Temperatursensor LM 75

Der Temperatursensor LM 75 kann Temperaturen zwischen −55° und 125° mit einer Aufösung von 0,5° und einer Genauigkeit von ±3° erfassen. Der erfasste Wert wird in einem integrierten ADC in eine 9-Bit-Dualzahl im Zweierkomplement umgewandelt und linksbündig in einem 16-Bit-Register (Temperaturregister) gespeichert.

4.2.5.1 Register Das Temperaturregister hat die interne Nummer 0. Aus diesem Register kann man nur lesen.

Es gibt außerdem noch ein 8 Bit breites Konfigurationsregister an der Adresse 1[5]. Zwei weitere Register dienen zur Ansteuerung eines Schaltausgangs: Die untere Temperatur wird im Register T_{HYST} an der Nummer 2 eingestellt, die obere Temperatur im Register T_{OS} an der Nummer 3.

[5] Genaue Spezifikation im Datenblatt

4.2.5.2 Ansteuerung Der LM 75 besitzt eine I^2C-Schnittstelle. Die Adresse lautet $1001xxx_(2)$, wobei die drei unteren Bits per Jumper einstellbar sind.

Bei der Ansteuerung des LM 75 muss man in der Regel zuerst die Nummer des Registers nennen, das man beschreiben oder von dem man lesen will. Erst dann kann man das gewünschte Register beschreiben oder lesen. Lässt man die Nennung der Nummer des Registers weg und liest sofort ein Register, so wird immer das zuletzt benutzte Register gelesen. Nach dem Start ist es das Register mit der Nummer 0.

4.2.5.3 A 8 Bit in das Konfigurationsregister schreiben Dies ist der einfachste Fall. Folgende Schritte sind nötig: START, Baustein-Adresse+0 schreiben (R/W=0), Registernummer 1 schreiben, Konfigurationsbyte schreiben, STOP.

4.2.5.4 B Lesen des Konfigurationsregisters (8 Bit) Hier muss der kombinierte Modus eingesetzt werden, weil erst geschrieben und dann gelesen werden soll: START, Baustein-Adresse+0 schreiben (R/W=0), Registernummer 1 schreiben, START-REPEATED, Baustein-Adresse+1 schreiben (R/W=1), Konfigurationsbyte lesen, STOP.

4.2.5.5 C 8 Bit Lesen ohne Registernummer Der LM 75 merkt sich, auf welches Register man zuletzt zugegriffen hat. Man muss dann die Registernummer beim Lesen nicht noch einmal angeben. In diesem Fall erlauben das Konfigurations- und das Temperaturregister einen verkürzten 8-Bit-Zugriff: START, Baustein-Adresse+1 schreiben (R/W=1), Konfigurationsbyte lesen, STOP.

4.2.5.6 D 16 Bit in Register T_{HYST} oder T_{OS} schreiben Auch dieser Fall ist einfach: START, Baustein-Adresse+0 schreiben (R/W=0), Registernummer 2 oder 3 schreiben, High-Byte schreiben, Low-Byte schreiben, STOP.

4.2.5.7 E Lesen eines 16-Bit-Registers Hier muss wieder der kombinierte Modus eingesetzt werden, weil erst geschrieben und dann gelesen wird: START, Baustein-Adresse+0 schreiben (R/W=0), Registernummer schreiben, START-REPEATED, Baustein-Adresse+1 schreiben (R/W=1), High-Byte lesen, Low-Byte lesen, STOP.

4.2.5.8 F Lesen eines 16-Bit-Registers ohne Registernummer Auch beim 16-Bit-Lesen gibt es wieder einen verkürzten Zugriff (falls der letzte Zugriff auch schon auf dasselbe Register erfolgt ist: START, Baustein-Adresse+1 schreiben (R/W=1), High-Byte lesen, Low-Byte lesen, STOP.

Nr.	Register	R/W	Breite	A	B	C	D	E	F
0	Temperaturregister	R	16 Bit	-	-	+	-	+	+
1	Konfigurationsregister	R+W	8 Bit	+	+	+	-	-	-
2	T_{HYST}	R+W	16 Bit	-	-	-	+	+	+
3	T_{OS}	R+W	16 Bit	-	-	-	+	-	+

Tabelle 3: Register und Abläufe

4.2.5.9 Zusammenfassung Trotz seiner einfachen Struktur erfordert der Temperatursensor LM 75 eine Reihe verschiedener I^2C-Kommunikationsabläufe. Tabelle 3 fasst zusammen, welcher Ablauf für welches Register erlaubt ist.