I²C-Routinen

(für PIC-Mikrocontroller)

Autor: Buchgeher Stefan

Letzte Bearbeitung: 27. Februar 2004

Inhaltsverzeichnis

1. Grundlegendes zu I²C

Der I²C-Bus[1] wurde ursprünglich von der Firma Philips für die Kommunikation verschiedener Baugruppen in einem Gerät entwickelt.

Der I²C-Bus ist eine so genannte synchrone serielle 2-Drahtverbindung zwischen einem Master und mehreren Slaves. In den meisten Fällen ist der Master ein Mikrocontroller (z.B. ein PIC), während es sich beim Slave z.B. um einen Speicherbaustein handelt, oder um Porterweiterungsbausteine, Temperatursensor, Uhrenbausteine, Analog-Digital-Wandler oder um Spezialbausteine aus dem Bereich der Audio- und Videotechnik.

Der I²C-Bus benötigt neben einer Masseleitung nur zwei Leitungen:

- Eine Taktleitung (diese wird meist mit **SCL** bezeichnet) und dient zur Erzeugung des Taktes. Der Takt wird dabei immer vom Master erzeugt. Die Taktgeschwindigkeit ist von den verwendeten Slave-Bausteinen abhängig. Sie kann maximal 100 kHz oder 400 kHz betragen.
- Eine Datenleitung (diese wird meist mit **SDA** bezeichnet) und dient zur Übertragung der Daten. Die Daten können sowohl vom Master zum Slave (hier spricht man von Daten schreiben) als auch vom Slave zum Master (hier spricht man von Daten lesen) übertragen werden.

In den meisten Fällen wird die Datenübertragung von einem Master ausgelöst. Es ist aber auch möglich, dass zwei oder mehr Master in einen Bus vorkommen. In diesem Fall spricht man vom Multi-Master-Mode. Hier ist durch ein bestimmtes Protokoll gesichert, dass immer nur ein Master den Bus übernehmen kann, wenn der Bus frei ist.
Der Multi-Master-Mode soll hier aber **nicht** näher betrachtet werden. Diese Dokumentation bezieht sich ausschließlich auf die Konfiguration ein Master mit ein oder mehreren Slaves.

Einige Bausteine der PIC16-Familie beinhalten ein Hardwaremodul für die Ansteuerung von I²C-Slave-Bausteinen (z.B. die PIC16F7x und PIC16F87x). Beinhaltet der für ein Projekt ausgewählte PIC kein I²C-Hardwaremodul, so ist man gezwungen diese per Software zu realisieren, wenn man mit I²C-Bausteinen kommunizieren muss. Diese Dokumentation beschreibt die Software für die Implementierung des I²C-Protokolls für PIC-Mikrocontroller ohne I²C-Hardwaremodul. (Die Verwendung bzw. Konfiguration des vorhandenen I²C-Hardwaremoduls der PIC16F7x bzw. PIC16F87x wird hier **nicht** beschrieben. Dies kann in den Datenblättern des verwendeten PIC-Mikrocontrollers nachgelesen werden)

2. I²C-Protokoll

Jede Aktion auf dem I²C-Bus geht vom Master aus. Als Master dient hier ein PIC (z.B. der PIC16F84 oder PIC16F62x). Die erste Aufgabe des Masters einer jeden Datenübertragung ist das erzeugen der Startbedingung. Die **Startbedingung** ist wie folgt definiert: **Auf der Datenleitung (SDA) erfolgt eine High-Low-Flanke, während die Taktleitung (SCL) High ist.** Damit wird gekennzeichnet, dass auf dem I²C-Bus ein Datenverkehr stattfindet. Würden sich auf dem Bus weitere Master befinden (Multi-

[1] Das Kürzel I²C bedeutet Inter-Integrated- Circuit, wird daher auch oft mit IIC abgekürzt

Master-Mode) so dürfen diese jetzt keine Datenübertragung zu einem Slave beginnen, da der Bus jetzt belegt ist.
Nun muss der Master die Adresse des Slaves mit welchem er kommunizieren möchte bekannt geben. Zu diesem Zweck besitzt jeder Slave eine Adresse. Diese Adresse besteht in den meisten Fällen aus einem vom Hersteller festgelegtem Bauteilkode und einer Bausteinadresse, welche durch Beschaltung dafür reservierter Anschlüsse wählbar ist. So ist es möglich mehrere gleichartiger Bausteine an einem Bus anzuschließen. (Z.B. mehrere Speicherbausteine oder mehrere Porterweiterungs-bausteine.)
Nach der Bauteiladresse (diese besteht in den meisten Fällen aus 7 Bit oder aus 10 Bit) erfolgt als nächstes die Bekanntgabe der Übertragungsrichtung. Möchte der Master (also der PIC) von einem Slave Daten lesen, so erfolgt als Übertragungsrichtung ein Low (0). Möchte der Master (PIC) zu einem Slave Daten (oder Befehle) übertragen, so erfolgt als Übertragungsrichtung ein High (1).
Die Adresse und das Bit welches die Übertragungsrichtung festlegt werden auch als **Kontrollbyte** bezeichnet.

Nach dem Kontrollbyte muss der Master auf ein **Bestätigungsbit** vom adressierten Slave warten.

Je nach Slave kann nun entweder **ein Datenbyte oder mehrere Datenbytes** vom Master zum Slave oder vom Slave zum Master übertragen werden.
Werden Daten vom Master zum Slave Übertragen (Schreibvorgang), so muss der Master nach jedem übertragenen Byte auf eine Bestätigung vom Slave warten.
Erfolgt der Datenverkehr in die andere Richtung (also vom Slave zum Master, man spricht hier auch von einem Lesevorgang) so muss der Master jedes empfange Byte bestätigen.

Sind alle Daten übertragen, muss der I²C-Bus wieder freigegeben werden. Diese Freigabe erfolgt mit einer Stoppbedingung. Die **Stoppbedingung** ist wie folgt definiert: **Auf der Datenleitung (SDA) erfolgt eine Low-High-Flanke, während die Taktleitung (SCL) High ist.** Der I²C-Bus ist nun frei und kann von einem anderen Master zur Datenübertragung benutzt werden.

Das folgende Bild zeigt zur besseren Verdeutlichung nochmals den gesamten Vorgang.

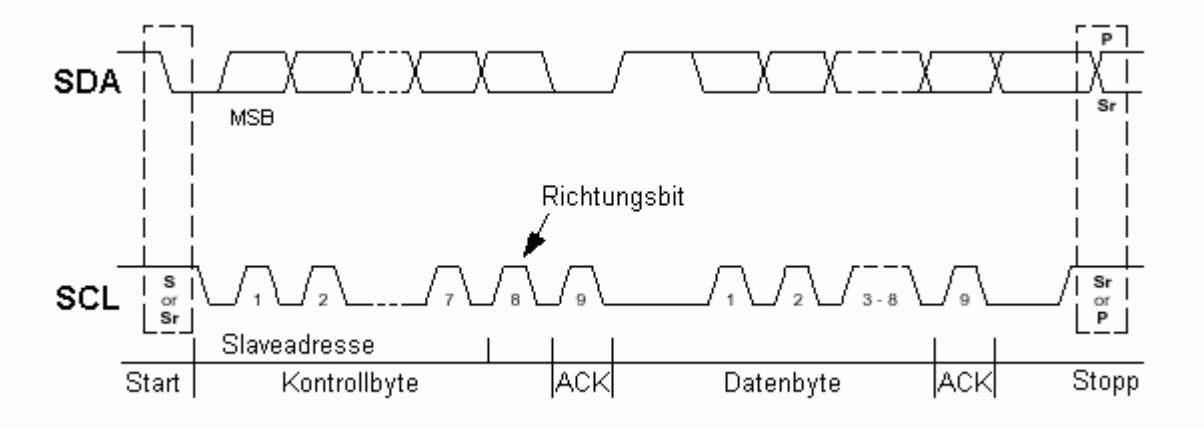

Bild 1: I²C-Protokoll

3. Hardware

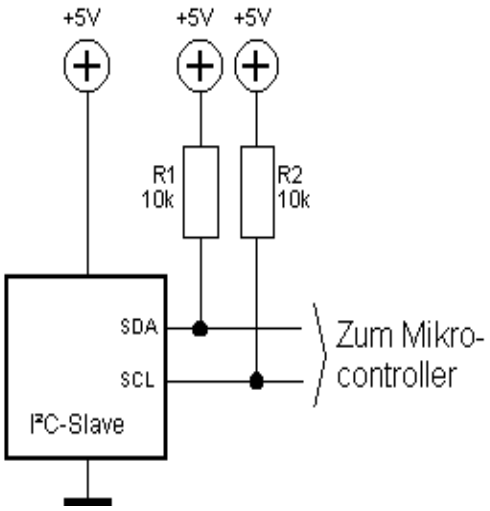

Bild 2: Anschluss eines Slaves am I²C-Bus

Bild 2 zeigt die minimale Beschaltung für die Datenübertragung. Sowohl die Datenleitung (SDA) als auch Taktleitung (SCL) arbeiten bidirektional und müssen mit je einem Pull-Up-Widerstand (R1 und R2) mit der Betriebsspannung verbunden werden.
Wenn der Bus frei ist, führen beide Leitungen einen High-Pegel. An den I²C-Bus angeschlossene Chips müssen Open-Drain- oder Open-Collector-Ausgänge haben. Mit diesen lässt sich die entsprechende Leitung zur Ausgabe eines Low-Pegels auf Masse ziehen. Die Gesamtheit dieser Chips am Bus realisiert somit eine sogenannte "Wired-AND-Schaltung" (Bild 3). Die maximale Taktgeschwindigkeit beträgt 100 kBit/s (im so genannten Fast-Mode sogar 400 kBit/s).
Die Anzahl der Busteilnehmer wird nur durch die maximale Buskapazität von 400 pF begrenzt.

Die Widerstände R3 bis R8 dienen als Schutz vor Spannungsspitzen.

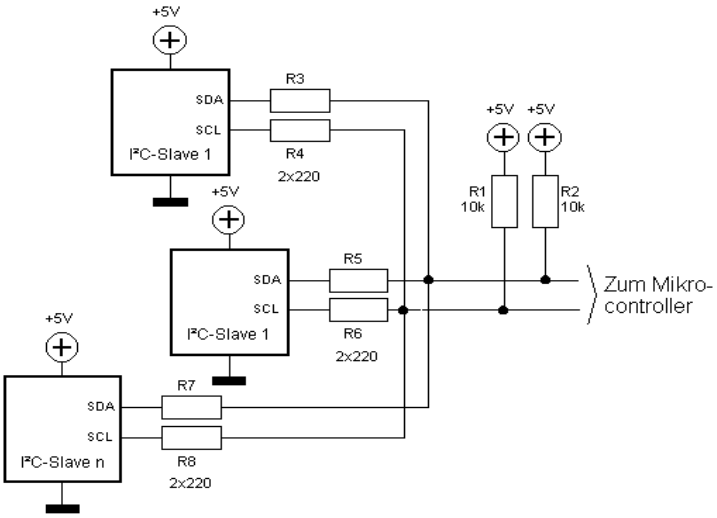

Bild 3: Anschluss mehrere Slaves am I²C-Bus

4. Software

Die Aufgabe der Software ist das im Abschnitt 2 beschriebene I²C-Protokoll umzusetzen. Die hier beschriebenen Unterprogramme sind nur notwendig, wenn der verwendete Mikrocontroller **kein** I²C-Hardwaremodul besitzt. Weiters unterstützt die hier beschrieben Software nur die langsamere Taktgeschwindigkeit (100 kBit/s), und ist für die Standardtaktfrequenz des PIC (4 MHz) ausgelegt.

4.1. Benötigte Register und Portdefinition

Für die Realisierung des Softwareprotokolls werden neben einigen PIC internen Register (SFR, **S**pezielle **F**unktions-**R**egister) noch folgende Register benötigt:

- TEMP1: Übergabeparameter, dieses Register beinhaltet entweder das zum I²C-Slave zu schreibende Byte oder das vom I²C-Slave gelesene Byte.
- TEMP2, TEMP3: Hilfsregister

Anmerkung:
Diese drei Register (TEMP1 bis TEMP3) sind so genannte temporäre Register. D.h. Sie werden nur kurzzeitig verwendet und können daher auch in beliebigen anderen Unterprogrammen verwendet werden.

Wie schon erwähnt benötigt der I²C-Bus eine Takt- und eine Datenleitung. Diese werden im Code wie folgt definiert (siehe auch Abschnitt 5 Demonstrationsbeispiel):

```
#define     SDA   PORTA,3
#define     SCL   PORTA,4
```

Anmerkung:
Hier wird für die Softwareimplementierung des I²C-Bus der Port A verwendet, wobei die Datenleitung am Portpin RA3 angeschlossen ist, während der Taktleitung der Portpin RA4 zugeordnet wird. Diese Zuordnung ist natürlich von Projekt zu Projekt unterschiedlich. Es ist auch möglich, dass diese beiden Pins an unterschiedlichen Ports angeschlossen sind. Z. B SDA (Datenleitung) am Port B (genauer am Portpin RB1) und SCL (Taktleitung) am Port C (genauer am Portpin RC3).

4.2. Initialisierung (Unterprogramm „INIT“)

Dieses Unterprogramm dient zur Initialisierung des Mikrocontrollers. Bei diesem Beispiel ist hier, für die Implementierung des I²C-Protokolls, nur die Definition der der verwendeten Portpins als Ausgang notwendig.

Der folgende Programmausschnitt zeigt eine mögliche Initialisierungsroutine für den PIC16F628. Die I²C-Slaves sind hier am Port A an den Bits 3 und 4 angeschlossen. Die schwarz hervorgehobenen Stellen sind für die Implementierung des I²C-Protokolls notwendig.

```
INIT            clrf  PORTA
                movlw 0x07              ;Alle Comparatoreingaenge
                movwf CMCON             ;  auf digital I/O umschalten

                bsf   STAT,RP0          ;Registerseite 1
                bcf   SDA               ;SDA und
                bcf   SCL               ;  SCL als Ausgang definieren
                bcf   STAT,RP0          ;Registerseite 0

                return
```

4.3. Unterprogramme für die Nachbildung des I²C-Protokolls

Zur Nachbildung des I²C-Protokolls sind 7 Unterprogramme notwendig:

- Je ein Unterprogramm zur Erzeugung der Start- und der Stoppbedingung. (Unterprogramme *I2C_START* bzw. *I2C_STOP*)
- Je ein Unterprogramm welches ein Bit auf den I²C-Bus schreibt (*I2C_BITSCHREIBEN*) bzw. ein Bit vom I²C-Bus einliest (*I2C_BITLESEN*). Diese beiden Unterprogramme dienen als Basis für die Unterprogramme zum Schreiben bzw. zum Lesen eines Bytes. (Unterprogramme *I2C_SCHREIBEN* bzw. *I2C_LESEN*)
- Das letzte Unterprogramm dient „nur“ zur Anpassung der I²C-Taktrate (*DELAY5*)

Nun aber zu den einzelnen Unterprogrammen im Detail:

4.3.1 Unterprogramm I2C_START

Aufgabe:
Dieses Unterprogramm erzeugt die Startbedingung.

Zur Erinnerung, die Startbedingung ist wie folgt definiert:
Auf der Datenleitung (SDA) erfolgt eine High-Low-Flanke, während die Taktleitung (SCL) High ist.

Hier das Unterprogramm:

```
I2C_START       bsf   SCL             ;SCL = 1
                call  DELAY5          ;5us warten
                bsf   SDA             ;SDA = 1
                call  DELAY5          ;5us warten
                bcf   SDA             ;SDA = 0
                call  DELAY5          ;5us warten
                bcf   SCL             ;SCL = 0
                return
```

Anmerkung:
Die Anweisung `call DELAY5` ruft ein Unterprogramm auf, welches eine Verzögerung von 5µs erzeugt. Diese Verzögerungszeit ist notwendig, damit die maximale Taktgeschwindigkeit von 100 kbit/s nicht überschritten wird. (siehe auch Abschnitt 4.3.7. Unterprogramm DELAY5 und Abschnitt 4.4. Änderung bei Verwendung eines höheren PIC-Taktes als 4MHz)

4.3.2 Unterprogramm I2C_STOP

Aufgabe:
Dieses Unterprogramm erzeugt die Stoppbedingung.

Zur Erinnerung, die Stoppbedingung ist wie folgt definiert: Auf der Datenleitung (SDA) erfolgt eine Low-High-Flanke, während die Taktleitung (SCL) High ist.

Hier das Unterprogramm:

```
I2C_STOP        bcf   SDA             ;SDA = 0
                call  DELAY5          ;5us warten
                bsf   SCL             ;SCL = 1
                call  DELAY5          ;5us warten
                bsf   SDA             ;SDA = 1
```

```
                return
```

Anmerkung:
Die Anweisung `call DELAY5` ruft ein Unterprogramm auf, welches eine Verzögerung von 5µs erzeugt. Diese Verzögerungszeit ist notwendig, damit die maximale Taktgeschwindigkeit von 100 kbit/s nicht überschritten wird. (siehe auch Abschnitt 4.3.7. Unterprogramm DELAY5 und Abschnitt 4.4. Änderung bei Verwendung eines höheren PIC-Taktes als 4MHz)

4.3.3 Unterprogramm I2C_BITLESEN

Aufgabe:
Dieses Unterprogramm liest ein Bit vom Slave ein und sichert es im Übergabeflag TEMP3,0. Der für die Datenübertragung erforderliche Takt (Leitung SCL) wird dabei softwaremäßig so erzeugt, dass der Pin SCL von Low (0) auf High (1) gesetzt wird. Nach einer bestimmten Zeit wird SCL wieder auf Low (0) zurückgesetzt. Diese Zeit wird hauptsächlich vom Unterprogramm DELAY5 bestimmt.

Vorgehensweise:
- Datenleitung (SDA) als Eingang definieren. Dazu muss das zum Portpin zugehörig TRIS-Flag gesetzt werden. Dieses befindet sich jedoch in der Registerbank 1. Es muss daher zuvor zur Registerbank 1 gewechselt werden und danach wieder zurück zur Registerbank 0.
- Damit SDA eingelesen werden kann muss dieses gesetzt werden
- Taktleitung (SCL) setzen (Low-High-Flanke) und 5µs warten
- Den Inhalt von SDA ins Übergabeflag (TEMP3,0) sichern. Dies wird wie folgt realisiert: zunächst das Übergabeflag (TEMP3,0) löschen. Nur wenn SDA gesetzt ist, das Übergabeflag (TEMP3,0) setzen.
- Taktleitung (SCL) wieder auf Low (High-Low-Flanke)

Hier das Unterprogramm:

```
I2C_BITLESEN    bsf   STAT,RP0          ;Registerbank 1
                bsf   SDA               ;SDA als Eingang definieren
                bcf   STAT,RP0          ;Registerbank 0
                bsf   SDA               ;SDA = 1
                bsf   SCL               ;SCL = 1
                call  DELAY5            ;5us warten
                bcf   TEMP3,0           ;Uebergabeflag (TEMP3,0) mit dem
                btfsc SDA               ;  Inhalt von SDA laden.
                bsf   TEMP3,0
                bcf   SCL               ;SCL = 0
                return
```

Anmerkung:
Das Bit 0 im temporären Register TEMP3 beinhaltet das gelesene Bit. Dieses Register dient hier nur als Übergaberegister zum übergeordneten Unterprogramm I2C_LESEN Das Register TEMP3 kann daher auch in anderen Unterprogrammen verwendet werden.

4.3.4 Unterprogramm I2C_BITSCHREIBEN

Aufgabe:
Dieses Unterprogramm schreibt das im Übergabeflag TEMP3,1 stehende Bit zum Slave Der für die Datenübertragung erforderliche Takt (Leitung SCL) wird dabei softwaremäßig so erzeugt, dass der Pin SCL von Low (0) auf High (1) gesetzt wird Nach einer bestimmten Zeit wird SCL wieder auf Low (0) zurückgesetzt. Diese Zeit wird hauptsächlich vom Unterprogramm DELAY5 bestimmt.

Vorgehensweise:
- Datenleitung (SDA) als Ausgang definieren. Dazu muss das zum Portpin zugehörige TRIS-Flag gelöscht werden. Diese befindet sich jedoch in der Registerbank 1. Es muss daher zuvor zur Registerbank 1 gewechselt werden und danach wieder zurück zur Registerbank 0.
- SDA = TEMP3,1. Für diese Anweisung existiert jedoch kein Assemblerbefehl! Es ist daher folgender Umweg notwendig: Zuerst prüfen ob das Übergabeflag (TEMP3,1) gesetzt ist. Ist es gesetzt, die Datenleitung (SDA) ebenfalls setzen. Andernfalls die Datenleitung (SDA) zurücksetzen.
- Takt erzeugen: Dazu zunächst die Taktleitung (SCL) setzen, 5µs warten, und abschließend die Taktleitung (SCL) wieder auf Low.

Hier das Unterprogramm:
```
I2C_BITSCHREIBEN
                bsf   STAT,RP0          ;Registerbank 1
                bcf   SDA               ;SDA als Ausgang definieren
                bcf   STAT,RP0          ;Registerbank 0
                btfss TEMP3,1           ;Uebergabeflag (TEMP3,1) gesetzt?
                goto  I2C_WEITER1
                bsf   SDA               ;ja: SDA = 1
                goto  I2C_WEITER2
I2C_WEITER1     bcf   SDA               ;nein: SDA = 1
I2C_WEITER2     bsf   SCL               ;SCL = 1
                call  DELAY5            ;5us warten
                bcf   SCL               ;SCL = 0
                return
```

Anmerkung:
Das Bit 1 im temporären Register TEMP3 beinhaltet das zu schreibende Bit. Dieses Register dient hier nur als Übergaberegister vom übergeordneten Unterprogramm I2C_SCHREIBEN. Das Register TEMP3 kann daher auch in anderen Unterprogrammen verwendet werden.

4.3.5 Unterprogramm I2C_LESEN

Aufgabe:
Ein Byte vom I²C-Bus bitweise mit Hilfe des Unterprogramms I2C_BITLESEN lesen und im Übergaberegister TEMP1 sichern. Danach ein Bestätigungsbit ausgeben.

Vorgehensweise:
- Das Übergaberegister TEMP1 löschen. Dieses Register wird nun bitweise mit einer Schleife beschrieben und beinhaltet am Ende dieses Unterprogramms das vom I²C-Bus eingelesene Byte.

- Schleifenzähler (TEMP2) mit dem Wert 8 laden. (die folgende Schleife wird demnach achtmal durchlaufen)
- Carry löschen (dies ist notwendig, damit in den folgenden Schiebebefehlen keine ungewollte 1 in das Übergaberegister TEMP1 geschoben wird)
- Schleifenbeginn
  - Alle Bits im Übergaberegister TEMP1 auf die nächst höhere Stelle schieben
  - Ein Bit mit Hilfe des Unterprogramms I2C_BITLESEN vom I²C-Bus lesen
  - Ist das soeben eingelesene Bit (Bit 0 vom Register TEMP3) gesetzt, Bit 0 des Übergaberegister TEMP1 setzen
  - Schleifenzähler (TEMP2) um 1 vermindern. Besitzt der Schleifenzähler danach einen Wert der größer als 0 ist die Schleife wiederholen. Besitzt es nach der Verminderung den Wert 0, so wurde ein ganzes Byte vom I²C-Bus eingelesen und die Schleife wird verlassen
- Schleifenende
- Eine Bestätigung (ACK = High) mit Hilfe des Unterprogramms I2C_BITSCHREIBEN auf den I²C-Bus schreiben

Hier das Unterprogramm:

```
I2C_LESEN   clrf  TEMP1       ;Uergaberegister TEMP1 loeschen. (Dieses
                               ;  Register beinhaltet am Ende dieses
                               ;  Unterprogramms das vom I2C-Bus
                               ;  eingelesene Byte
            movlw 08
            movwf TEMP2        ;Schleifenzaehler (TEMP2) mit dem Wert 8 laden
            bcf   STAT,C       ;Carry loeschen
I2C_SCHL1   rlf   TEMP1,f      ;Bits im Register TEMP1 auf die naechst hoehere
                               ;  Stelle schieben
            call  I2C_BITLESEN ;Ein Bit mit Hilfe des Unterprogramms
                               ;  I2C_BITLESEN vom I2C-Bus lesen
            btfsc TEMP3,0      ;Ist das soeben eingelesene Bit gesetzt, Bit 0
            bsf   TEMP1,0      ;  des Uebergaberegister TEMP1 setzen
            decfsz TEMP2,f     ;Diesen Vorgang 8mal durchfuehren
            goto  I2C_SCHL1
            bsf   TEMP3,1      ;Bestaetigung (ACK = High) auf den I2C-Bus
            call  I2C_BITSCHREIBEN  ;schreiben
            return
```

Anmerkung:
Die temporären Register TEMP1 bis TEMP3 dienen hier als Übergabe- oder als Hilfsregister. Diese Register können daher auch in anderen Unterprogrammen verwendet werden.

4.3.6 Unterprogramm I2C_SCHREIBEN

Aufgabe:
Das zu schreibende Byte (befindet sich im Übergaberegister TEMP1) bitweise mit Hilfe des Unterprogramms I2C_BITSCHREIBEN auf den I²C-Bus schreiben. Danach ein Bestätigungsbit empfangen.

Vorgehensweise:
- Schleifenzähler (TEMP2) mit dem Wert 8 laden. (die folgende Schleife wird demnach achtmal durchlaufen)
- Schleifenbeginn

- TEMP3,1 = TEMP1,7. Für diese Anweisung existiert jedoch kein Assemblerbefehl! Es ist daher folgender Umweg notwendig: Zuerst das Flag TEMP3,1 löschen. Danach prüfen, ob das Flag TEMP1,7 gesetzt ist. Nur wenn dieses Flag gesetzt ist, auch Flag TEMP3,1 setzen.
- Den Inhalt vom Flag TEMP3,1 mit Hilfe des Unterprogramms I2C_BITSCHREIBEN auf den I²C-Bus schreiben
- Bits im Übergaberegister TEMP1 auf die nächst höhere Stelle schieben
- Schleifenzähler (TEMP2) um 1 vermindern. Besitzt der Schleifenzähler danach einen Wert der größer als 0 ist die Schleife wiederholen. Besitzt es nach der Verminderung den Wert 0, so wurde ein ganzes Byte auf den I²C-Bus geschrieben und die Schleife wird verlassen

- Schleifenende
- Eine Bestätigung (ACK = High) mit Hilfe des Unterprogramms I2C_BITLESEN vom I²C-Bus lesen

Hier das Unterprogramm:

```
I2C_SCHREIBEN
            movlw 08
            movwf TEMP2       ;Schleifenzaehler (TEMP2) mit dem Wert 8 laden
I2C_SCHL2   bcf   TEMP3,1     ;TEMP3,1 = TEMP1,7
            btfsc TEMP1,7
            bsf   TEMP3,1
            call  I2C_BITSCHREIBEN  ;Den Inhalt vom Flag TEMP3,1 mit Hilfe des
                                    ;  Unterprogramms I2C_BITSCHREIBEN auf den
                                    ;  I2C-Bus schreiben
            rlf   TEMP1,f     ;Bits im Uebergabergister TEMP1 auf die naechst
                              ;  hoehere Stelle schieben
            decfsz TEMP2,f    ;Diesen Vorgang 8mal durchfuehren
            goto  I2C_SCHL2   ;Bestaetigung (ACK) vom I2C-Slave lesen
            call  I2C_BITLESEN
            return
```

Anmerkung:
Die temporären Register TEMP1 bis TEMP3 dienen hier als Übergabe- oder als Hilfsregister. Diese Register können daher auch in anderen Unterprogrammen verwendet werden.

4.3.7 Unterprogramm DELAY5

Aufgabe:
Dieses Unterprogramm erzeugt eine Zeitverzögerung von 5 us. Diese Verzögerungszeit ist notwendig, damit die maximale Taktgeschwindigkeit von 100 kbit/s nicht überschritten wird.

Hier das Unterprogramm:

```
DELAY5      nop
            return
```

Anmerkung:
Bei einer PIC-Taktfrequenz von 4 MHz betragen die hier benötigten Befehle die in den Klammern angegebenen Abarbeitungszeiten (Zykluszeiten)

call DELAY5 (2us)

nop (1us)
return (2us)

Bei Verwendung einer höheren Taktfrequenz müssen zusätzlich nop-Befehle eingefügt werden!

4.4. Änderung bei Verwendung eines höheren PIC-Taktes als 4MHz

Die hier beschriebene Software wurde für die PIC-Standard-Taktfrequenz von 4 MHz beschrieben. eine geringere Taktfrequenz kann problemlos verwendet werde. Nicht aber eine Höhere. Wird eine höhere Taktfrequenz verwendet so muss aber nur das Unterprogramm DELAY5 geändert werden. Dieses Unterprogramm hat ja die Aufgabe die im I²C-Protokoll spezifizierte Taktrate (von 100 kHz) zu erzeugen. Also eine Verzögerung von 5 µs.

Zu beachten ist, dass die Befehle „call“ und „return“ jeweils zwei Taktzyklen benötigen, während der Befehl „nop“ nur einen Taktzyklus benötigt. Als Taktzyklus ist bei der PIC-Familie folgende Formel definiert:

$$Taktzyklus\,[\mu s] = \frac{4}{Taktfreqenz\,[MHz]}$$

Bei der Standard-Taktfrequenz von 4 MHz gilt also: Die Befehle „call“ und „return“ benötigen je 2 µs. Dies ergibt zusammen also 4 µs. Für die benötigte Verzögerungszeit von 5 µs sind also noch Befehle notwendig, die eine µs benötigen. Dies entspricht einem „nop“-Befehl, da dieser nur 1 Taktzyklus benötigt, also genau 1µs.

Bei Verwendung eines 10 MHz Taktes gilt:

$$Taktzyklus\,[\mu s] = \frac{4}{10\,MHz} = 0.4\,\mu s$$

Die Befehle „call“ und „return“ benötigen je 2 Taktzyklen, also 0,8 µs, der Befehl „nop“ benötigt 0,4 µs. Es stellt sich also die Frage wie viele „nop“-Befehle benötigt man für eine Verzögerung von 5 µs. Zieht man von den 5 µs die Zeit für die beiden Befehle „call“ und „return“ ab, bleiben 3,4 µs übrig. (5 µs – 0,8 µs – 0,8 µs = 3,4 µs). Ein „nop“ – Befehl benötigt 0,4 µs. Daraus ergibt sich, dass theoretisch 8,5 „nop“-Befehle notwendig sind (3,4 µs / 0,4 µs = 8,5). Ein halber „nop“-Befehl existiert natürlich nicht! Deshalb müssen 9 „nop“-Befehle verwendet werden.
Das Unterprogramm DELAY5 sieht also bei Verwendung eines 10 MHz-Quarzes wie folgt aus:

```
DELAY5      nop
            nop
            nop
            nop
            nop
            nop
            nop
            nop
            nop
            return
```

Eine bessere, „schönere“ und vor allem codesparendere Möglichkeit ist die Verwendung einer Schleife anstelle dieser vielen „nop“-Befehle. Würde man für den PIC einen 20-MHz-Takt verwenden, so wären 21 (!) „nop“-Befehle notwendig.

Mit einer Schleife würde das Unterprogramm DELAY5 so aussehen:

Für 10 MHz:

```
DELAY5      movlw  .3
            movwf  TEMP3
DELAY5SCHL1 decfsz TEMP3,f
            goto   DELAY5SCHL1
            return
```

è Verzögerungszeit: 5,6 µs

Für 20 MHz:

```
DELAY5      movlw  .5
            movwf  TEMP3
DELAY5SCHL1 nop
            decfsz TEMP3,f
            goto   DELAY5SCHL1
            nop
            return
```

è Verzögerungszeit: 5 µs

5. Demonstrationsbeispiel

Das folgende Beispiel dient nur zur Demonstration. Es zeigt eine mögliche Einbindung der oben beschriebenen Unterprogramme

5.1. Hardware

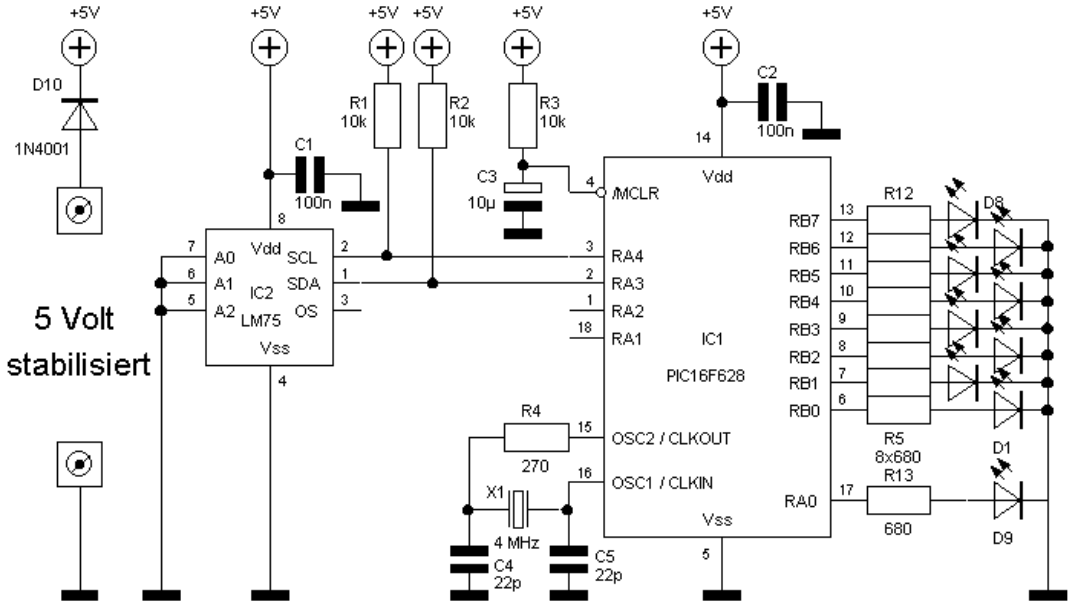

Bild 4: Schaltung zur Demonstration der I²C-Routinen (Ansteuerung des Temperatursensors LM75)

Bei diesem Demonstrationsbeispiel soll ein Mikrocontroller (PIC16F628) zyklisch (ca. alle 65ms) die Temperatur von einem Temperatursensor (IC2; mit I²C-Schnittstelle) auslesen und in BCD kodierter Form am Port B ausgeben. Wie schon im Abschnitt 3 (Hardware) beschrieben sind sowohl bei der Datenleitung (SDA), als auch bei der Taktleitung (SCL) Pull-Up-Widerstände notwendig. (Hier R1 bzw. R2). Die Temperaturanzeige erfolgt der Einfachheit halber nur mit Leuchtdioden (D1 bis D8). Eine

zusätzliche Leuchtdiode (D9) am Portpin RA0 soll das Vorzeichen anzeigen, wobei eine leuchtende LED eine negative Temperatur kennzeichnet.

Für die Takterzeugung dient eine Standardbeschaltung bestehend aus einem 4-MHz-Quarz (X1), zwei Kondensatoren (C4, C5) und einem Widerstand (R4).
Das RC-Glied (R3, C3) erzeugt einen definierten Reset beim Anlegen der Betriebsspannung.

Die Diode D10 verhindert eine Zerstörung des Mikrocontrollers bei einer Falschpolung der Betriebsspannung. Als Betriebsspannung muss eine stabile 5-V-Spannungsquelle verwendet werden.

5.2. Software

```
;*************************************************************************************************
;** Demonstrationsbeispiel zur Implementierung des I2C-Protokolls bei PIC-Mikrocontrollern   **
;** die nicht ueber dieses Hardwaremodul verfuegen.                                           **
;**                                                                                           **
;** Als I2C-Slave dient hier ein Temperatursensor (LM75). Dieser wird zyklisch (ca. alle 65ms**
;** via I2C-Protokoll von einem PIC16F628 ausgelesen. Der gelesene Temperaturwert wird am     **
;** Port B in BCD-kodierter Form ausgegeben. Zur Anzeige dienen der Einfachheit halber LEDs. **
;** Diese LEDs koennten aber durch zwei 7-Segment-Anzeigen mit einem passenden Treiber er-    **
;** setzt werden. An der Software wuerde sich dadurch nichts aendern. Eine weiter Leuchtdiode**
;** am Port RA0 gibt die Polaritaet der Temperatur an. Bei negativen Temperaturen leuchtet   **
;** diese LED. Sie entspricht somit einem negativen Vorzeichen.                               **
;** Die Hauptaufgabe, also die Kommunikation mit dem Sensor und die Aufbereitung und Anzeige **
;** der Temperatur, erfolgt in der Interrupt Service Routine (ISR). Das I2C-Protokoll wird   **
;** aber mit mehreren Unterprogrammen erzeugt.                                                **
;**                                                                                           **
;** Entwickler: Buchgeher Stefan                                                              **
;** Entwicklungsbeginn der Software: 13. Oktober 2003                                         **
;** Funktionsfaehig seit: 13. Oktober 2003                                                    **
;** Letzte Bearbeitung: 27. Februar 2004                                                      **
;*************************************************************************************************

               List p=PIC16F628


;******************************** Register (in Registerseite 0) ******************************
TMR0           equ    1             ;Timer0-Register
STAT           equ    3             ;Statusregister
PORTA          equ    5             ;PortA-Register
PORTB          equ    6             ;PortB-Register
INTCON         equ    0B            ;Interrupt-Register
CMCON          equ    1F            ;Komparator-Register


;******************************** Register (in Registerseite 1) ******************************
OPTREG         equ    1             ;OPTION-Register
TRISA          equ    5             ;Richtungsregister PortA
TRISB          equ    6             ;Richtungsregister PortB


;******************************** Eigene Register (in Registerbank 0) ************************
ISR_STAT_TEMP  equ    20            ;Zwischenspeicher des Statusregister der ISR
ISR_w_TEMP     equ    21            ;Zwischenspeicher des Arbeitsregister der ISR

TEMPERATURHIGH equ    22            ;High-Byte der Temperatur
TEMPERATURLOW  equ    23            ;Low-Byte der Temperatur

TEMP1          equ    24            ;allgemeines Hilfsregister 1
TEMP2          equ    25            ;allgemeines Hilfsregister 2
TEMP3          equ    26            ;allgemeines Hilfsregister 3


;******************************** Bits in Registern der Registerbank 0 ***********************
;Register STAT
C              equ    0             ;Carrybit im Statuswort-Register
RP0            equ    5             ;Seitenauswahlbit im Statuswort-Register
```

```
;Register INTCON
T0IF            equ     2               ;TMR0-Interruptflag im INTCON-Register


;******************************** Portbelegung ************************************************
#define         SDA     PORTA,3
#define         SCL     PORTA,4
#define         NEGLED  PORTA,0


;******************************** Ziele der Registeroperationen ******************************
w               equ     0
f               equ     1


;******************************** Konfigurations-Bits ****************************************
_BODEN_ON                       EQU     H'3FFF'
_BODEN_OFF                      EQU     H'3FBF'

_CP_ALL                         EQU     H'03FF'
_CP_75                          EQU     H'17FF'
_CP_50                          EQU     H'2BFF'
_CP_OFF                         EQU     H'3FFF'

_DATA_CP_ON                     EQU     H'3EFF'
_DATA_CP_OFF                    EQU     H'3FFF'

_PWRTE_OFF                      EQU     H'3FFF'
_PWRTE_ON                       EQU     H'3FF7'

_WDT_ON                         EQU     H'3FFF'
_WDT_OFF                        EQU     H'3FFB'

_LVP_ON                         EQU     H'3FFF'
_LVP_OFF                        EQU     H'3F7F'

_MCLRE_ON                       EQU     H'3FFF'
_MCLRE_OFF                      EQU     H'3FDF'

_ER_OSC_CLKOUT                  EQU     H'3FFF'
_ER_OSC_NOCLKOUT                EQU     H'3FFE'

_INTRC_OSC_CLKOUT               EQU     H'3FFD'
_INTRC_OSC_NOCLKOUT             EQU     H'3FFC'

_EXTCLK_OSC                     EQU     H'3FEF'
_LP_OSC                         EQU     H'3FEC'
_XT_OSC                         EQU     H'3FED'
_HS_OSC                         EQU     H'3FEE'

        __config        _MCLRE_ON & _PWRTE_OFF & _WDT_OFF & _HS_OSC & _BODEN_OFF & _LVP_OFF


                ORG     0x000
                goto    Beginn
                ORG     0x004
                goto    ISR


;******************************** ISR - Timer0 ***********************************************

;*************************************************************************************************
;** Interrupt Service Routine:                                                                  **
;**                                                                                             **
;** Aufruf: ca. alle 65 ms                                                                      **
;**                                                                                             **
;** Aufgaben:                                                                                   **
;**     + w-Register (=Arbeitsregister) und Status-Register zwischenspeichern (PUSH)           **
;**     + Temperatur (High- und Lowbyte) des Temperatursensors via I2C-Protokoll auslesen.     **
;**       Dazu sind folgende Schritte notwendig:                                                **
;**             + Startbedingung erzeugen (Unterprogramm I2C_START)                             **
;**             + Die Adresse des anzusprechenden I2C-Slaves und die gewuenschte Daten-         **
;**               richtung auf den I2C-Bus legen. (Hier, beim Temperatursensor lautet die       **
;**               Adresse 1001000 und eine 1 fuer die Datenrichtung, da wir ja die Temperatur   **
;**               vom Sensor lesen wollen). Diese Adresse wird nun mit dem Unterprogramm        **
;**               I2C_SCHREIBEN gemaess dem I2C-Protokoll erzeugt                               **
;**             + Highbyte der Temperatur mit dem Unterprogramm I2C_LESEN einlesen, und an-     **
;**               schliessend den im Uebergabeparameter TEMP1 enthaltenen Wert ins Register     **
;**               TEMPERATURHIGH schreiben. Die Bestaetigung (gemaess dem I2C-Protokoll wird    **
```

```
;**             dabei im Unterprogramm I2C_LESEN erzeugt)                                  **
;**           + Lowbyte der Temperatur mit dem Unterprogramm I2C_LESEN einlesen, und an-   **
;**             schliessend den im Uebergabeparameter TEMP1 enthaltenen Wert ins Register  **
;**             TEMPERATURLOW schreiben. Die Bestaetigung (gemaess dem I2C-Protokoll wird  **
;**             dabei im Unterprogramm I2C_LESEN erzeugt)                                  **
;**           + Stoppbedingung erzeugen (Unterprogramm I2C_STOP)                           **
;**     + Temperatur (Highbyte) ueberpruefen. Ist die Temperatur positiv (hoechstwertiges Bit **
;**       des Temperaturwerts gesetzt, dies entspricht einer Zweierkomplementdarstellung), so **
;**       bleibt der Temperaturwert unveraendert und das Vorzeichen wird geloescht. Ist die  **
;**       Temperatur negativ, so werden alle Bits des Temperaturwerts invertiert. Das        **
;**       Ergebnis wird anschliessend um 1 erhoeht und das Vorzeichen wird gesetzt.          **
;**       Der so eventuell korrigierte binaere Temperaturwert muss fuer die Ausgabe noch in  **
;**       eine BCD-Form gebracht werden. Dazu ist das Unterprogramm BINBCD2 zustaendig.Diese **
;**       2-stellige BCD-Zahl muss jetzt nur mehr am Port B ausgegeben werden.               **
;**     + Timer-Interrupt-Flag T0IF wieder loeschen                                        **
;**     + Arbeits- und Statusregister wiederherstellen (POP).                              **
;*************************************************************************************************
ISR
PUSH            movwf   ISR_w_TEMP              ;w-Register retten
                swapf   STAT,w                  ;Statusregister
                bcf     STAT,RP0                ;  in Registerseite 0
                movwf   ISR_STAT_TEMP           ;  retten

                ;Beginn der eigentlichen ISR-Routine
                ;Schritt 1: Temperatur von Temperatursensor einlesen
                call    I2C_START               ;Startbedingung generieren

                movlw   b'10010001'             ;I2C-Adresse des Temperatursensors
                movwf   TEMP1                   ;  und Anweisung fuer einen Lesezugriff
                call    I2C_SCHREIBEN

                call    I2C_LESEN               ;Highbyte des Temperatursensors
                movf    TEMP1,w                 ;  auslesen und im Register
                movwf   TEMPERATURHIGH          ;  TEMPERATURHIGH sichern

                call    I2C_LESEN               ;Lowbyte des Temperatursensors
                movf    TEMP1,w                 ;  auslesen und im Register
                movwf   TEMPERATURLOW           ;  TEMPERATURLOW sichern

                call    I2C_STOP                ;Stoppbedingung generieren (den Bus somit
                                                ;  wieder freigeben)

                ;Schritt 2: Temperaturwert ueberpruefen (Es wird nur das Highbyte verwendet)
                btfsc   TEMPERATURHIGH,7        ;Handelt es sich bei der Temperatur um einen
                                                ;  negativen Wert?
                goto    TEMPERATURNEG
TEMPERATURPOS   bcf     NEGLED                  ;nein: Temperaturwert bleibt unveraendert,
                                                ;  Vorzeichen loeschen
                goto    ISRWEITER1

TEMPERATURNEG   comf    TEMPERATURHIGH,f        ;ja: Die Bits des Temperaturwerts
                incf    TEMPERATURHIGH,f        ;  invertieren und um 1 erhoehen
                bsf     NEGLED                  ;  (Zweierkomplement) Vorzeichen setzen
                goto    ISRWEITER1

ISRWEITER1      movf    TEMPERATURHIGH,w        ;Den binaeren Temperaturwert in
                call    BINBCD2                 ;  BCD umwandeln und
                movwf   PORTB                   ;  am Port B ausgeben
                ;Ende der eigentlichen ISR-Routine

ISRFERTIG       bcf     INTCON,T0IF             ;T0-Interruptflag löschen

POP             swapf   ISR_STAT_TEMP,w         ;Status-Register
                movwf   STAT                    ;  und
                swapf   ISR_w_TEMP,f            ;  w-Register
                swapf   ISR_w_TEMP,w            ;  wieder herstellen

                retfie


;******************************** Unterprogramme ***********************************************

;*************************************************************************************************
;** Initialisierung des Prozessor:                                                              **
;**     + Timer0 loeschen                                                                       **
;**     + Comparatoreingaenge (Port A) deaktivieren (Port A auf digital I/O umschalten)         **
;**     + Timer0-Vorteiler: 256 (-> Aufruf er Timer0-ISR ca. alle 65 ms)                        **
;**     + Ports: Port A: Bit 0..4,6,7: Ausgaenge                                                **
;**                      Bit 5: Eingang (MCLR)                                                  **
```

```
;**             Port B: Ausgaenge                                                    **
;**     + diverse Register vorbelegen                                                **
;*******************************************************************************
INIT            clrf    TMR0                ;Timer0 auf 0 voreinstellen

                clrf    PORTA
                movlw   0x07                ;Alle Comparatoreingaenge
                movwf   CMCON               ;  auf digital I/O umschalten

                bsf     STAT,RP0            ;Registerseite 1
                movlw   b'00000111'         ;interner Takt, Vorteiler = 256 an TMR0
                movwf   OPTREG
                movlw   b'00100000'         ;Port A: Bit 0..4,6,7: Ausgaenge
                movwf   TRISA               ;        Bit 5: Eingang (MCLR)
                movlw   b'00000000'         ;Port B als Ausgang definieren
                movwf   TRISB
                bcf     STAT,RP0            ;Registerseite 0

                return


;****************************** I2C Routinen ************************************

;*******************************************************************************
;** I2C_START                                                                  **
;**                                                                            **
;** Aufgabe:                                                                   **
;**     Dieses Unterprogramm erzeugt die Startbedingung entsprechend dem I2C-Protokoll.  **
;**                                                                            **
;** Anmerkungen:                                                               **
;**     + Die Startbedingung ist wie folgt definiert: Auf der Datenleitung (SDA) erfolgt  **
;**       eine High-Low-Flanke, waehrend die Taktleitung (SCL) High ist.       **
;**     + Die Anweisung call DELAY5 ruft ein Unterprogramm auf, welches eine Verzoegerung  **
;**       von 5us erzeugt. Diese Verzoegerungszeit ist notwendig, damit die maximale Takt-  **
;**       geschwindigkeit von 100 kbit/s nicht ueberschritten wird. (siehe auch Unter-  **
;**       programm DELAY5)                                                     **
;*******************************************************************************
I2C_START       bsf     SCL                 ;SCL = 1
                call    DELAY5              ;5us warten
                bsf     SDA                 ;SDA = 1
                call    DELAY5              ;5us warten
                bcf     SDA                 ;SDA = 0
                call    DELAY5              ;5us warten
                bcf     SCL                 ;SCL = 0
                return


;*******************************************************************************
;** I2C_Stop                                                                   **
;**                                                                            **
;** Aufgabe:                                                                   **
;**     Dieses Unterprogramm erzeugt die Stoppbedingung entsprechend dem I2C-Protokoll.  **
;**                                                                            **
;** Anmerkungen:                                                               **
;**     + Die Stoppbedingung ist wie folgt definiert: Auf der Datenleitung (SDA) erfolgt  **
;**       eine Low-High-Flanke, waehrend die Taktleitung (SCL) High ist.       **
;**      + Die Anweisung call DELAY5 ruft ein Unterprogramm auf, welches eine Verzoegerung  **
;**       von 5us erzeugt. Diese Verzoegerungszeit ist notwendig, damit die maximale Takt-  **
;**       geschwindigkeit von 100 kbit/s nicht ueberschritten wird. (siehe auch Unter-  **
;**       programm DELAY5)                                                     **
;*******************************************************************************
I2C_STOP        bcf     SDA                 ;SDA = 0
                call    DELAY5              ;5us warten
                bsf     SCL                 ;SCL = 1
                call    DELAY5              ;5us warten
                bsf     SDA                 ;SDA = 1
                return


;*******************************************************************************
;** I2C_BITLESEN                                                               **
;**                                                                            **
;** Aufgabe:                                                                   **
;**     Dieses Unterprogramm liest ein Bit vom Slave ein und sichert es im Uebergabeflag  **
;**     TEMP3,0. Der fuer die Datenuebertragung erforderliche Takt (Leitung SCL) wird dabei  **
;**     softwaremaessig so erzeugt, dass der Pin SCL von Low (0) auf High (1) gesetzt wird.  **
;**     Nach einer bestimmten Zeit wird SCL wieder auf Low (0) zurückgesetzt. Diese Zeit  **
;**     wird hauptsächlich vom Unterprogramm DELAY5 bestimmt.                  **
;**                                                                            **
```

```
;** Vorgehensweise:                                                                          **
;**     + Datenleitung (SDA) als Eingang definieren. Dazu muss das zum Portpin zugehoerige   **
;**       TRIS-Flag gesetzt werden. Dieses befindet sich jedoch in der Registerbank 1. Es    **
;**       muss daher zuvor zur Registerbank 1 gewechselt werden und danach wieder zurueck    **
;**       zur Registerbank 0.                                                                **
;**     + Damit SDA eingelesen werden kann muss dieses gesetzt werden                        **
;**     + Taktleitung (SCL) setzen (Low-High-Flanke) und 5us warten                          **
;**     + Den Inhalt von SDA ins Uebergabeflag (TEMP3,0) sichern. Dies wird wie folgt real-  **
;**       isiert: zunaechst das Uebergabeflag (TEMP3,0) loeschen. Nur wenn SDA gesetzt ist,  **
;**       das Uebergabeflag (TEMP3,0) setzen.                                                **
;**     + Taktleitung (SCL) wieder auf Low (High-Low-Flanke)                                 **
;**                                                                                          **
;** Anmerkung:                                                                               **
;**     Das Bit 0 im temporaeren Register TEMP3 beinhaltet das gelesene Bit. Dieses Register **
;**     dient hier nur als Uebergaberegister zum uebergeordneten Unterprogramm I2C_LESEN.    **
;**     Das Register TEMP3 kann daher auch in anderen Unterprogrammen verwendet werden.      **
;******************************************************************************************
I2C_BITLESEN   bsf     STAT,RP0              ;Registerbank 1
               bsf     SDA                   ;SDA als Eingang definieren
               bcf     STAT,RP0              ;Registerbank 0
               bsf     SDA                   ;SDA = 1
               bsf     SCL                   ;SCL = 1
               call    DELAY5                ;5us warten
               bcf     TEMP3,0               ;Uebergabeflag (TEMP3,0) mit dem Inhalt von
               btfsc   SDA                   ;  SDA laden
               bsf     TEMP3,0
               bcf     SCL                   ;SCL = 0
               return


;******************************************************************************************
;** I2C_BITSCHREIBEN                                                                       **
;**                                                                                        **
;** Aufgabe:                                                                               **
;**     Dieses Unterprogramm schreibt das im Uebergabeflag TEMP3,1 stehende Bit zum Slave  **
;**     Der fuer die Datenübertragung erforderliche Takt (Leitung SCL) wird dabei software-**
;**     maessig so erzeugt, dass der Pin SCL von Low (0) auf High (1) gesetzt wird.        **
;**     Nach einer bestimmten Zeit wird SCL wieder auf Low (0) zurueckgesetzt. Diese Zeit  **
;**     wird hauptsaechlich vom Unterprogramm DELAY5 bestimmt.                             **
;**                                                                                        **
;** Vorgehensweise:                                                                        **
;**     + Datenleitung (SDA) als Ausgang definieren. Dazu muss das zum Portpin zugehoerige **
;**       TRIS-Flag geloescht werden. Diese befindet sich jedoch in der Registerbank 1. Es **
;**       muss daher zuvor zur Registerbank 1 gewechselt werden und danach wieder zurueck  **
;**        zur Registerbank 0.                                                             **
;**     + SDA = TEMP3,1. Fuer diese Anweisung existiert jedoch kein Assemblerbefehl! Es ist**
;**       daher folgender Umweg notwendig: Zuerst pruefen ob das Uebergabeflag (TEMP3,1) ge-**
;**       setzt ist. Ist es gesetzt, die Datenleitung (SDA) ebenfalls setzen. Andernfalls  **
;**       die Datenleitung (SDA) zuruecksetzen.                                            **
;**     + Takt erzeugen: Dazu zunaechst die Taktleitung (SCL) setzen, 5us warten, und ab-  **
;**       schliessend die Taktleitung (SCL) wieder auf Low.                                **
;**                                                                                        **
;** Anmerkung:                                                                             **
;**     Das Bit 1 im temporaeren Register TEMP3 beinhaltet das zu schreibende Bit. Dieses  **
;**     Register dient hier nur als Uebergaberegister vom uebergeordneten Unterprogramm    **
;**     I2C_SCHREIBEN. Das Register TEMP3 kann daher auch in anderen Unterprogrammen ver-  **
;**     wendet werden.                                                                     **
;******************************************************************************************
I2C_BITSCHREIBEN
               bsf     STAT,RP0              ;Registerbank 1
               bcf     SDA                   ;SDA als Ausgang definieren
               bcf     STAT,RP0              ;Registerbank 0
               btfss   TEMP3,1               ;Uebergabeflag (TEMP3,1) gesetzt?
               Goto    I2C_WEITER1
               bsf     SDA                   ;ja: SDA = 1
               goto    I2C_WEITER2
I2C_WEITER1    bcf     SDA                   ;nein: SDA = 1
I2C_WEITER2    bsf     SCL                   ;SCL = 1
               call    DELAY5                ;5us warten
               bcf     SCL                   ;SCL = 0
               return


;******************************************************************************************
;** I2C_LESEN                                                                              **
;**                                                                                        **
;** Aufgabe:                                                                               **
;**     Ein Byte vom I2C-Bus bitweise mit Hilfe des Unterprogramms I2C_BITLESEN lesen und im **
;**     Uebergaberegister TEMP1 sichern. Danach ein Bestaetigungsbit ausgeben              **
```

```
;**                                                                                          **
;** Vorgehensweise:                                                                          **
;**     + Das Uebergaberegister TEMP1 loeschen. Dieses Register wird nun bitweise mit einer  **
;**       Schleife beschrieben und beinhaltet am Ende dieses Unterprogramms das vom I2C-Bus  **
;**       eingelesene Byte                                                                   **
;**     + Schleifenzaehler (TEMP2) mit dem Wert 8 laden. (die folgende Schleife wird demnach **
;**       achtmal durchlaufen)                                                               **
;**     + Carry loeschen (dies ist notwendig, damit in den folgenden Schiebebefehlen keine   **
;**       ungewollte 1 in das Uebergaberegister TEMP1 geschoben wird)                        **
;**     + Schleifenbeginn                                                                    **
;**             + Alle Bits im Uebergaberegister TEMP1 auf die naechst hoehere Stelle        **
;**               schieben                                                                   **
;**             + Ein Bit mit Hilfe des Unterprogramms I2C_BITLESEN vom I2C-Bus lesen       **
;**             + Ist das soeben eingelesene Bit (Bit 0 vom Register TEMP3) gesetzt, Bit 0   **
;**               des Uebergaberegister TEMP1 setzen                                         **
;**             + Schleifenzaehler (TEMP2) um 1 vermindern. Besitzt der Schleifenzaehler     **
;**               danach einen Wert der groesser als 0 ist die Schleife wiederholen. Besitzt **
;**               es nach der Verminderung den Wert 0, so wurde ein ganzes Byte vom I2C-Bus  **
;**               eingelesen und die Schleife wird verlassen                                 **
;**     + Schleifenende                                                                      **
;**     + Eine Bestaetigung (ACK = High) mit Hilfe des Unterprogramms I2C_BITSCHREIBEN auf   **
;**       den I2C-Bus schreiben                                                              **
;**                                                                                          **
;** Anmerkung:                                                                               **
;**     Die temporaeren Register TEMP1 bis TEMP3 dienen hier als Uebergabe- oder als Hilfs-  **
;**     register. Diese Register koennen daher auch in anderen Unterprogrammen verwendet     **
;**     werden.                                                                              **
;******************************************************************************************
I2C_LESEN       clrf    TEMP1                   ;Uergaberegister TEMP1 loeschen. (Dieses
                                                ;  Register beinhaltet am Ende dieses Unter-
                                                ;  programms das vom I2C-Bus eingelesene Byte
                movlw   08
                movwf   TEMP2                   ;Schleifenzaehler (TEMP2) mit dem Wert 8 laden
                bcf     STAT,C                  ;Carry loeschen
I2C_SCHL1       rlf     TEMP1,f                 ;Bits im Register TEMP1 auf die naechst hoehere
                                                ;  Stelle schieben
                call    I2C_BITLESEN            ;Ein Bit mit Hilfe des Unterprogramms
                                                ;  I2C_BITLESEN vom I2C-Bus lesen
                btfsc   TEMP3,0                 ;Ist das soeben eingelesene Bit gesetzt, Bit 0
                bsf     TEMP1,0                 ;  des Uebergaberegister TEMP1 setzen
                decfsz  TEMP2,f                 ;Diesen Vorgang 8mal durchfuehren
                goto    I2C_SCHL1
                bsf     TEMP3,1                 ;Bestaetigung (ACK = High) auf den I2C-Bus
                call    I2C_BITSCHREIBEN        ;schreiben
                return


;******************************************************************************************
;** I2C_SCHREIBEN                                                                          **
;**                                                                                        **
;** Aufgabe:                                                                               **
;**     Das zu schreibende Byte (befindet sich im Uebergaberegister TEMP1) bitweise mit Hilfe **
;**     des Unterprogramms I2C_BITSCHREIBEN auf den I2C-Bus schreiben. Danach ein          **
;**     Bestaetigungsbit empfangen.                                                        **
;**                                                                                        **
;** Vorgehensweise:                                                                        **
;**     + Schleifenzaehler (TEMP2) mit dem Wert 8 laden. (die folgende Schleife wird demnach **
;**       achtmal durchlaufen)                                                             **
;**     + Schleifenbeginn                                                                  **
;**             + TEMP3,1 = TEMP1,7. Fuer diese Anweisung existiert jedoch kein Assembler-  **
;**               befehl! Es ist daher folgender Umweg notwendig: Zuerst das Flag TEMP3,1  **
;**               loeschen. Danach pruefen, ob das Flag TEMP1,7 gesetzt ist. Nur wenn dieses **
;**               Flag gesetzt ist, auch Flag TEMP3,1 setzen                               **
;**             + Den Inhalt vom Flag TEMP3,1 mit Hilfe des Unterprogramms I2C_BITSCHREIBEN **
;**               auf den I2C-Bus schreiben                                                **
;**             + Bits im Uebergabergister TEMP1 auf die naechst hoehere Stelle schieben   **
;**             + Schleifenzaehler (TEMP2) um 1 vermindern. Besitzt der Schleifenzaehler   **
;**               danach einen Wert der groesser als 0 ist die Schleife wiederholen. Besitzt **
;**               es nach der Verminderung den Wert 0, so wurde ein ganzes Byte auf den I2C- **
;**               Bus geschrieben und die Schleife wird verlassen                          **
;**      + Schleifenende                                                                   **
;**     + Eine Bestaetigung (ACK = High) mit Hilfe des Unterprogramms I2C_BITLESEN vom I2C- **
;**       Bus lesen                                                                        **
;**                                                                                        **
;** Anmerkung:                                                                             **
;**     Die temporaeren Register TEMP1 bis TEMP3 dienen hier als Uebergabe- oder als Hilfs- **
;**     register. Diese Register koennen daher auch in anderen Unterprogrammen verwendet   **
;**     werden.                                                                            **
;******************************************************************************************
```

```
I2C_SCHREIBEN  movlw   08
               movwf   TEMP2                ;Schleifenzaehler (TEMP2) mit dem Wert 8 laden
I2C_SCHL2      bcf     TEMP3,1              ;TEMP3,1 = TEMP1,7
               btfsc   TEMP1,7
               bsf     TEMP3,1
               call    I2C_BITSCHREIBEN     ;Den Inhalt vom Flag TEMP3,1 mit Hilfe des
                                            ;  Unterprogramms I2C_BITSCHREIBEN auf den
                                            ;  I2C-Bus schreiben
               rlf     TEMP1,f              ;Bits im Uebergabergister TEMP1 auf die naechst
                                            ;  hoehere Stelle schieben
               decfsz  TEMP2,f              ;Diesen Vorgang 8mal durchfuehren
               goto    I2C_SCHL2            ;Bestaetigung (ACK) vom I2C-Slave lesen
               call    I2C_BITLESEN
               return


;*************************************************************************************************
;** DELAY5                                                                                      **
;**                                                                                             **
;** Aufgabe:                                                                                    **
;**     Dieses Unterprogramm erzeugt eine Zeitverzoegerung von 5 us. Diese Verzoegerungszeit    **
;**     ist notwendig, damit die maximale Taktgeschwindigkeit von 100 kbit/s nicht ueber-       **
;**     schritten wird.                                                                         **
;**                                                                                             **
;** Anmerkung:                                                                                  **
;**     Bei einer PIC-Taktfrequenz von 4 MHz betragen die hier benoetigten Befehle die in       **
;**     den Klammern angegebenen Abarbeitungszeiten (Zykluszeiten)                              **
;**             call    DELAY5 (2us)                                                            **
;**             nop            (1us)                                                            **
;**             return         (2us)                                                            **
;**     Bei Verwendung einer hoeheren Taktfrequenz muessen zusaetzlich nop-Befehle einge        **
;**     fuegt werden!                                                                           **
;*************************************************************************************************
DELAY5         nop
               return


;****************************** Weitere Unterprogramme ***********************************


;*************************************************************************************************
;** Umwandlung von Binaer nach BCD                                                              **
;** Aufgabe:                                                                                    **
;**     + Die im w-Register stehende Binaer-Zahl (0-255) nach BCD umwandeln. Die einzelnen      **
;**       Ziffern werden zunaechst in temporaere register (TEMP1 bis TEMP3) gespeichert und     **
;**       zu Programmende werden die Einer- und Zehnerstelle wieder in das w-Register           **
;**       kopiert (Zehnerstelle im H-Nibble, Einerstelle im L-Nibble). Wird die Hunderter-      **
;**       stelle benoetigt, so muss das temporaere Register TEMP3 in ein entsprechendes         **
;**       Register kopiert werden.                                                              **
;**                                                                                             **
;** Vorgehensweise:                                                                             **
;**     + Von der Zahl wird 10 so oft abgezogen, bis der Rest kleiner 10 ist, bei jeder         **
;**       Subtraktion wird TEMP2 (Zehnerstelle) um 1 erhoeht. Ist Temp2 (Zehnerstelle) = 10     **
;**       wird TEMP3 (Hunderterstelle) um 1 erhoeht und TEMP2 (Zehnerstelle) zu 0 gemacht.      **
;**     + Ergebnis in w                                                                         **
;**                                                                                             **
;** Anmerkung: Die temporaeren Register TEMP1 bis TEMP3 werden hier nur zum Zwischenspeichern**
;**     benoetigt. Sie koennen daher auch woanders verwendet werden.                            **
;*************************************************************************************************
BINBCD2        clrf    TEMP3                ;TEMP3 loeschen (Hunderter)
               clrf    TEMP2                ;TEMP2 loeschen (Zehner)
               movwf   TEMP1                ;w in TEMP1 (Einer)
BinBCDWdh      movlw   .10
               subwf   TEMP1,w              ;TEMP1 - 10 in w
               btfss   STAT,C               ;TEMP1 < 10 ?
               goto    BinBCDfertig         ;  ja: Ruecksprung
               movwf   TEMP1                ;  nein: (TEMP1 - 10) in TEMP1
               incf    TEMP2,f              ;TEMP2 + 1
               movlw   .10
               subwf   TEMP2,w
               btfss   STAT,C               ;TEMP2 = 10 ?
               goto    BinBCDWdh            ;  nein: wiederholen
               clrf    TEMP2                ;  ja: TEMP2 = 0
               incf    TEMP3,f              ;    und TEMP3 + 1
               goto    BinBCDWdh

BinBCDfertig   swapf   TEMP2,w              ;TEMP2 in Hi-Nibble
               iorwf   TEMP1,w              ;TEMP1 in Lo-Nibble
               return
```

```
;******************************** Hauptprogramm ************************************************

;***********************************************************************************************
;** Aufgaben des Hauptprogramms:                                                              **
;** + Controller initialisieren (Unterprogramm INIT)                                          **
;** + Timer0-Interrupt freigeben                                                              **
;** + Endlosschleife (die Hauptaufgaben werden in der ISR erledigt)                           **
;***********************************************************************************************
Beginn         call   INIT                ;Controller initialisieren
               movlw  b'10100000'         ;Timer0 freigeben durch Setzen von
               movwf  INTCON              ;  GIE und T0IE im Register INTCON

HPSCHLEIFE     goto   HPSCHLEIFE          ;Endlosschleife

               end
```

5.3. Anmerkungen zur Software

Die Software besteht im Wesentlichen aus einem kurzen Hauptprogramm, einer Interrupt-Service-Routine (kurz ISR), die die im Abschnitt 4.3 besprochenen Unterprogramme, zur Erzeugung des I²C-Protokolls, und einem Unterprogramm, welches ein Byte von binär nach BCD umwandelt, aufruft.

Die ISR wird ca. alle 65ms aufgerufen und besitzt folgende Aufgaben:

- Die Temperatur des Temperatursensor entsprechend dem I²C-Protokoll auslesen. Dazu ist folgende Bitfolge notwendig:

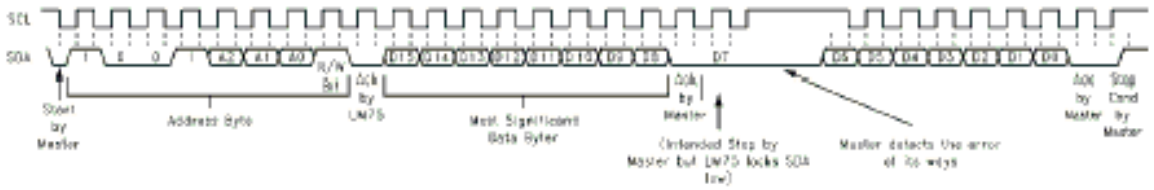

Bild 5: Protokoll zum Auslesen des Temperatursensor LM75

- Temperatur (Highbyte) überprüfen. Ist die Temperatur positiv (höchstwertiges Bit des Temperaturwerts gesetzt, dies entspricht einer Zweierkomplementdarstellung), so bleibt der Temperaturwert unverändert und das Vorzeichen wird gelöscht. Ist die Temperatur negativ, so werden alle Bits des Temperaturwerts invertiert. Das Ergebnis wird anschließend um 1 erhöht und das Vorzeichen wird gesetzt. Der so eventuell korrigierte binäre Temperaturwert muss für die Ausgabe noch in eine BCD-Form gebracht werden. Dazu ist das Unterprogramm BINBCD2 zuständig. Diese 2-stellige BCD-Zahl muss jetzt nur mehr am Port B ausgegeben werden.

Das Hauptprogramm besteht nach der Initialisierung (Unterprogramm INIT) und der Interruptfreigabe nur mehr aus einer Endlosschleife.

Das Unterprogramm INIT dient zur Initialisierung des Controller. Hier werden u.a. die Ports konfiguriert (Port dient als Eingang oder als Ausgang), der oder die Timer eingestellt usw. Dieses Unterprogramm ist vom Controllertyp abhängig und je nach Anwendung mehr oder weniger umfangreich. Siehe auch Abschnitt 4.2 (Initialisierung)

6. Quellen

- Dokument „The I²C-Bus spezification Version 2.1 January 2000“
- Buch „I²C-Bus angewandt“ (ISBN: 3-928051-71-7)
- Datenblatt des Temperatursensor LM75