# Datenaustausch zwischen zwei (oder mehreren) PIC16Fxx-Mikrocontrollern via $I^2C^{TM}$ in C

(oder: Wie realisiere ich einen $I^2C^{TM}$-Slave mit einem PIC16Fxx-Mikrocontroller)

Stefan Buchgeher

4. August 2014

⌂ www.stefan-buchgeher.info
✉ kontakt@stefan-buchgeher.info
✉ Großambergstraße 106/3, A-4040 Linz (Österreich)
✆ ++43/699/11 29 63 38

# Inhaltsverzeichnis



# Literatur

[1] Microchip. Datenblatt des PIC16F887. `http://ww1.microchip.com/downloads/en/DeviceDoc/41291G.pdf`, 2012.

[2] Microchip. Application Note: AN734 - Using the Mid-Range Enhanced Core PIC16 Devices' MSSP Moduls for slave I2C Communication. `http://ww1.microchip.com/downloads/en/AppNotes/00000734C.pdf`, 2013.

[3] mikroElekronika. mikroElektronika - Tools, Compilers, Books, Magazine. `http://www.mikroe.com/`, 2009.

[4] mikroElekronika. mikroc user's manual. `http://www.mikroe.com/`, 2009.

# 1 Einleitung

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 Zwei-Drahtverbindung zwischen einem Master und einem oder 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. Oder, und darum geht es hier, kann der Slave auch ein PIC-Mikrocontroller sein.

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, 400 kHz oder 1 MHz 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 **nicht** näher betrachtet werden. Diese Dokumentation bezieht sich ausschließlich auf die Konfiguration ein Master mit einem oder mehreren Slaves.

Einige Bausteine der PIC16-Familie beinhalten ein (oder zwei) Hardwaremodul(e) für die Ansteuerung von I²C-Slave-Bausteinen (z.B. PIC16F88x). 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 einen Slave mit einem PIC-Mikrocontroller mit I²C-Hardwaremodul.

**Wichtig:**
Damit ein Mikrocontroller vom Typ PIC16Fxx als Slave für den I²C-Bus verwendet werden kann muss dieser über das **MSSP**-Modul verfügen, und dieser muss den Slave-Mode beinhalten. MSSP steht für **M**aster **S**ynchronous **S**erial **P**ort.

Als Informationsquellen für diese Arbeit dienen das Datenblatt des verwendeten PIC-Mikrocontroller (hier PIC16F887 [1])und die Application-Note **AN734** mit dem Titel:

[1] Das Kürzel I²C bedeutet **I**nter-**I**ntegrated-**C**ircuit und wird daher auch oft mit IIC abgekürzt

*Using the Mid-Range Enhanced Core PIC16 Devices' MSSP Modules for Slave I²C™ Communication* [2].

Als Programmiersprache für die PIC-Mikrocontroller-Software wurde C mit dem mikroC-Compiler[2] verwendet. Für die Demonstrationsbeispiele (ab Seite 10) ist die kostenlose Demoversion[3] ausreichend.

# 2 I²C-Protokoll

Jede Aktion auf dem I²C-Bus geht vom Master aus. Als Master dient hier ein PIC (z.B. PIC16F887). 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-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 Porterweiterungsbausteine.)
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 (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**

[2]Der mikroC-Compiler wurde von der Firma mikroElektronika [3] entwickelt und dient zum Programmieren der PIC16Fxx-Familie in der Hochsprache C

[3]Die freie Demoversion ist nur auf eine Programmspeichergröße von 2k begrenzt, ansonst voll kompatibel zur Vollversion. Die Demoversion ist auf der Firmen-Webseite [3] downloadbar

**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.

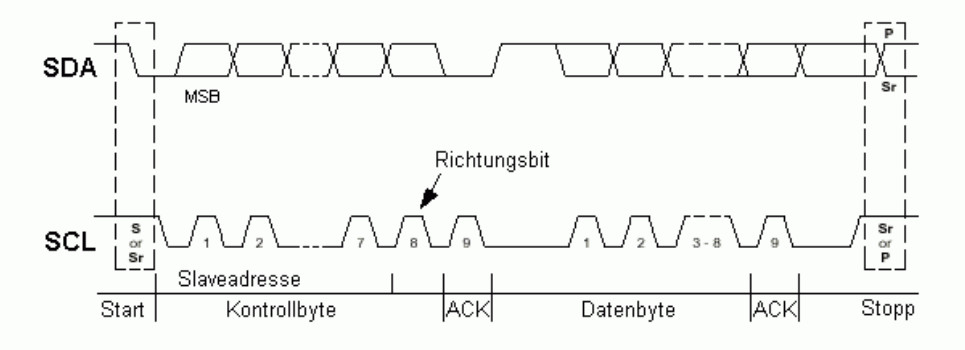


Abbildung 1: I²C-Protokoll

**Achtung:**
Das hier beschriebene Protokoll zum I²C-Bus - also das Setzen und das Löschen der Leitungen SDA und SCL muss in der Software **nicht** nachgebildet werden. Diese Aufgabe übernimmt das Hardwaremodul MSSP des PIC16Fxx-Mikrocontroller. In der Software müssen nur die Steuerbits gesetzt (oder gelöscht) werden und die zu übermittelten Daten bereitgestellt oder abgeholt werden. Mehr dazu in den folgenden Abschnitten.

# 3 PIC16Fxx als I²C-Slave

Dieser Abschnitt beschreibt nun wie ein PIC16Fxx-Mikrocontroller als Slave für den I²C zu konfigurieren ist und welche Software-Routinen notwendig sind.

**Wichtig:**
Damit ein Mikrocontroller vom Typ PIC16Fxx als Slave für den I²C-Bus verwendet werden kann muss dieser über das **MSSP**-Modul verfügen, und dieser muss den Slave-Mode beinhalten. MSSP steht für **M**aster **S**ynchronous **S**erial **P**ort.

## 3.1 Hardware

Zunächst die Hardwarebeschaltung. Abbildung 2 zeigt die Beschaltung des PIC16Fxx-Mikrocontrollers, wobei auch die Beschaltung des Oszillators X1 entfallen kann, wenn der verwendete PIC16Fxx-Mikrocontroller über einen internen Oszillator verfügt.
Der I²C-Bus mit seinen Leitungen (SDA, SCL, Masse, Betriebsspannung und dem optionalen Interrupt) ist im Bild unten zu sehen. Wesentlich ist hier eigentlich nur, dass

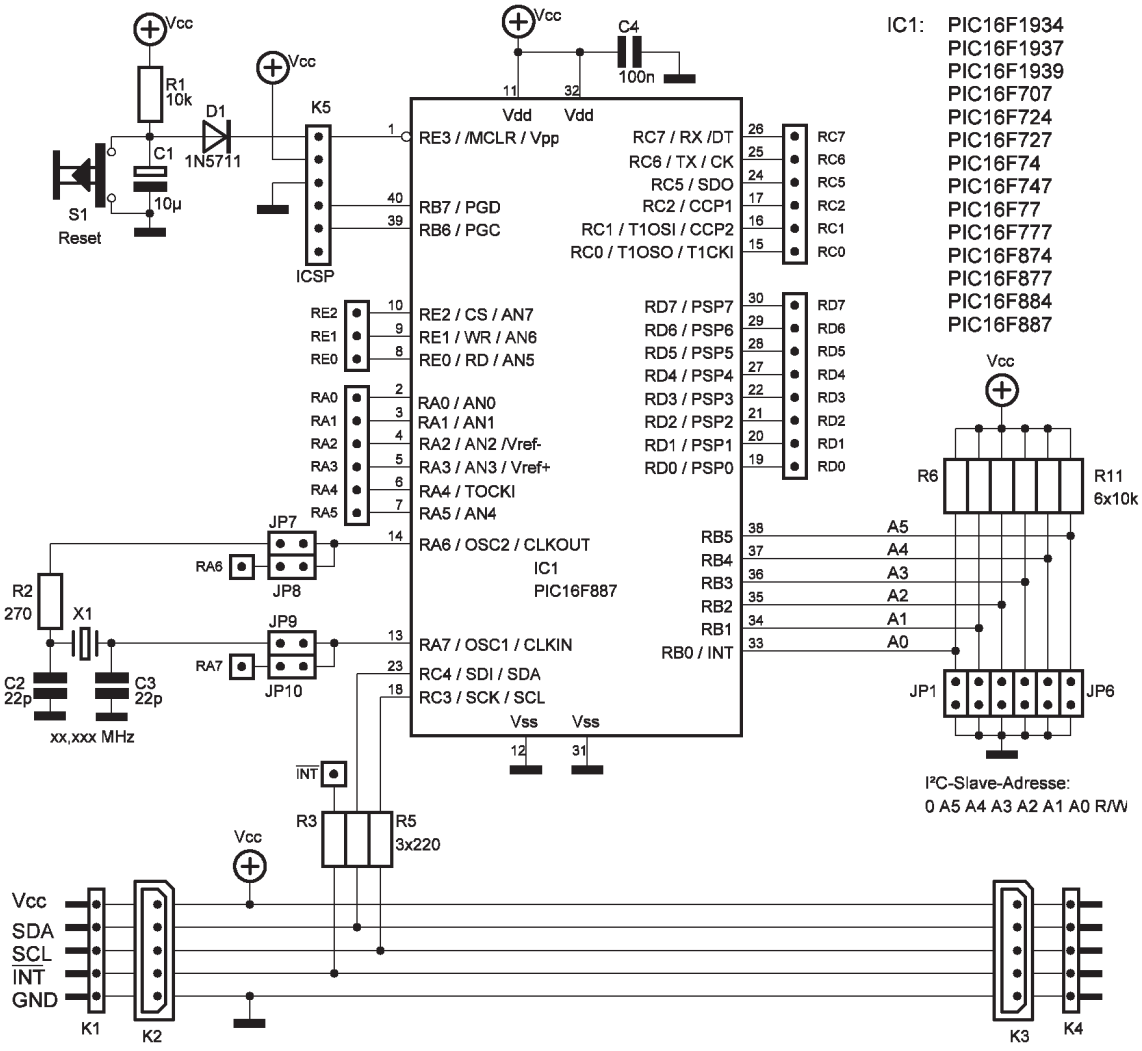


Abbildung 2: PIC16Fxx als I²C-Slave, Schaltung

die beiden PIC-I/O-Pins SDA und SCL über Schutzwiderstände (R3 bis R5) mit dem I²C-Bus verbunden werden.

Die übrigen Schaltungsteile entsprechen den üblichen PIC16Fxx-Applikationen. Die Schaltung rund um den Quarz X1 erzeugt einen stabilen Systemtakt. Dieser Schaltungsteil kann entweder komplett entfallen falls der verwendete PIC16F-Mikrocontroller einen internen Oszillator besitzt und dieser verwendet wird, oder umschaltbar mit Steckbrücken.

Die PIC16Fxx-Familie lässt sich sehr gut in der Schaltung programmieren. Diese Methode wird ICSP[4] genannt. Der Reset wird hier mit einem RC-Glied bestehend aus R1 und C1 erzeugt. Zusätzlich kann ein Reset jederzeit mit dem Taster S1 ausgelöst werden. Da der Reseteingang des Mikrocontrollers ($\overline{\text{MCLR}}$, Pin 1) auch gleichzeitig die Programmierspannung (ca. 13V) bei der ICSP-Proghrammierung ist, darf während einer Programmierung kein Reset ausgelöst werden. Durch die Diode D1 ist ein Reset durch das RC-Glied (R1

[4]ICSP steht für **I**n-**C**ircuit-**S**erial-**P**rogramming

und C1) während einer Programmierung via ICSP nicht möglich.

Die Spannungsversorgung für den Mikrocontroller erfolgt über den I²C-Bus, so dass hier kein extra Schaltungsteil notwendig ist. Ein Koppelkondensator (C4) ist aber empfehlenswert.

Die I²C-Busadresse kann entweder in der Software fix vergeben werden oder über Steckbrücken (Jumper, JP1 bis JP6). Hier wird fast der gesamte Port B dafür verwendet.

Alle übrigen Portpins sind frei verfügbar.

## 3.2 Software

Damit ein PIC16Fxx-Mikrocontroller als Slave für den I²C-Bus verwendet werden kann muss dieser (wie in diesem Dokument schon öfter erwähnt) über ein so genanntes **MSSP**-Modul verfügen. Dieses Modul ist im Mikrocontroller als Hardware integriert und unterstützt neben dem I²C-Bus einen weiteren seriellen Bus, der hier nicht weiter behandelt wird. Dieses Hardware-Modul muss zunächst für den I²C-Slave-Mode konfigueriert werden. Dazu dienen die folgenden Register:

- `SSPCON`
- `SSPCON2`
- `SSPSTAT`
- `SSPADD`

Wie diese konfiguriert werden müssen zeigen die beiden Demonstrationsbeispiel in Abschnitt 5 ab Seite 10.

Wichtig ist auch, dass die Portpins (SDA und SCL) als Eingang konfiguriert werden.

Da der Mikrocontroller über ein geeignetes Hardwaremodul verfügt erfolgt die Umsetzung des I²C-Protokolls automatisch, und der Anwender muss sich darum nicht kümmern. Er hat „nur“ die Aufgabe die empfangenen Daten rechtzeitig zu lesen und die zu sendenden Daten rechtzeitig zur Verfügung zu stellen. Dieses „rechtzeitig“ lässt sich sehr elegant mit einem Interrupt erkennen, die der Mikrocontroller erzeugt. Wichtig ist, dass dieser Interrupt (SSP-Interrupt) freigegeben werden muss! Die Abbildungen 3 und 4 zeigen die Statusbits bei einem Lesebefehl bzw. bei einem Schreibbefehl.

**FIGURE 13-7: I²C™ SLAVE MODE WAVEFORMS FOR RECEPTION (7-BIT ADDRESS)**

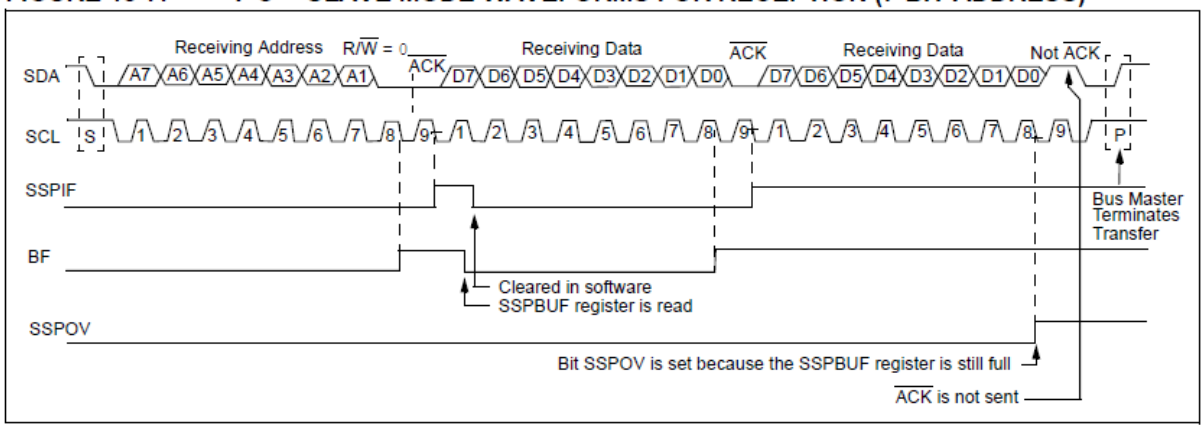


Abbildung 3: PIC16Fxx als I²C-Slave, Statusbits bei einem Lesebefehl (Auszug aus dem Datenblatt zum PIC16F887)

**FIGURE 13-8: I²C™ SLAVE MODE WAVEFORMS FOR TRANSMISSION (7-BIT ADDRESS)**

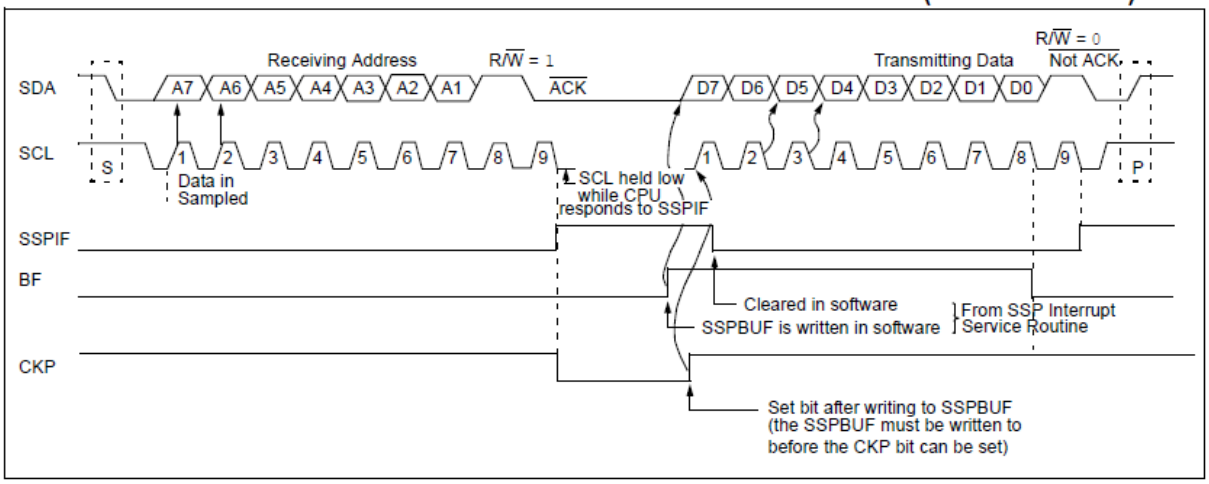


Abbildung 4: PIC16Fxx als I²C-Slave, Statusbits bei einem Schreibebefehl (Auszug aus dem Datenblatt zum PIC16F887)

Für den Anwender (Softwarenetwickler) besteht „nur“ die Aufgabe zum richtigen Zeitpunkt die zu lesenden Daten abzuholen bzw. die zu schreibenden Daten zur Verfügung zu stellen. Dazu dienen die zwei Flags `R_W` und `D_A` im Register `SSPSTAT`. Diese beiden Flags werden automatisch gesetzt oder gelöscht nachdem ein Datenbyte via I²C-Bus empfangen oder gesendet wurde. Gleichzeitig wird auch das Interrupt-Flag für das SSP-Modul (`SSPIF`) gesetzt und ein Interrupt ausgelöst, falls dieser Interrupt freigegeben wurde.
Mit anderen Worten ausgedrückt: Nach jedem empfangene oder gesendeten Byte via I²C-Bus wird der SSP-Interrupt ausgelöst. Dabei gibt das Flag `R_W` an, ob die Daten am Bus vom Slave gelesen werden sollen `R_W = 0` oder ob der Slave Daten auf den Bus Schreiben muss `R_W = 1`. Das Flag `D_A` gibt an, ob das zuletzt empfangene Byte das Adressbyte war `D_A = 0` oder ob es sich um ein Datenbyte handelt `D_A = 1`.

Somit ergeben sich die vier Fälle:

| R_W | D_A | Bedeutung |
|---|---|---|
| 0 | 0 | Daten vom Bus lesen, zuletzt empfangenes Byte war die Slaveadresse |
| 0 | 1 | Daten vom Bus lesen, zuletzt empfangenes Byte war ein Datenbyte |
| 1 | 0 | Daten auf den Bus schreiben, zuletzt empfangenes Byte war die Slaveadresse |
| 1 | 1 | Daten auf den Bus schreiben, zuletzt gesendetes Byte war ein Datenbyte |

Das Flussdiagramm in Abbildung 5 zeigt was in diesen vier Fällen zu tun ist.

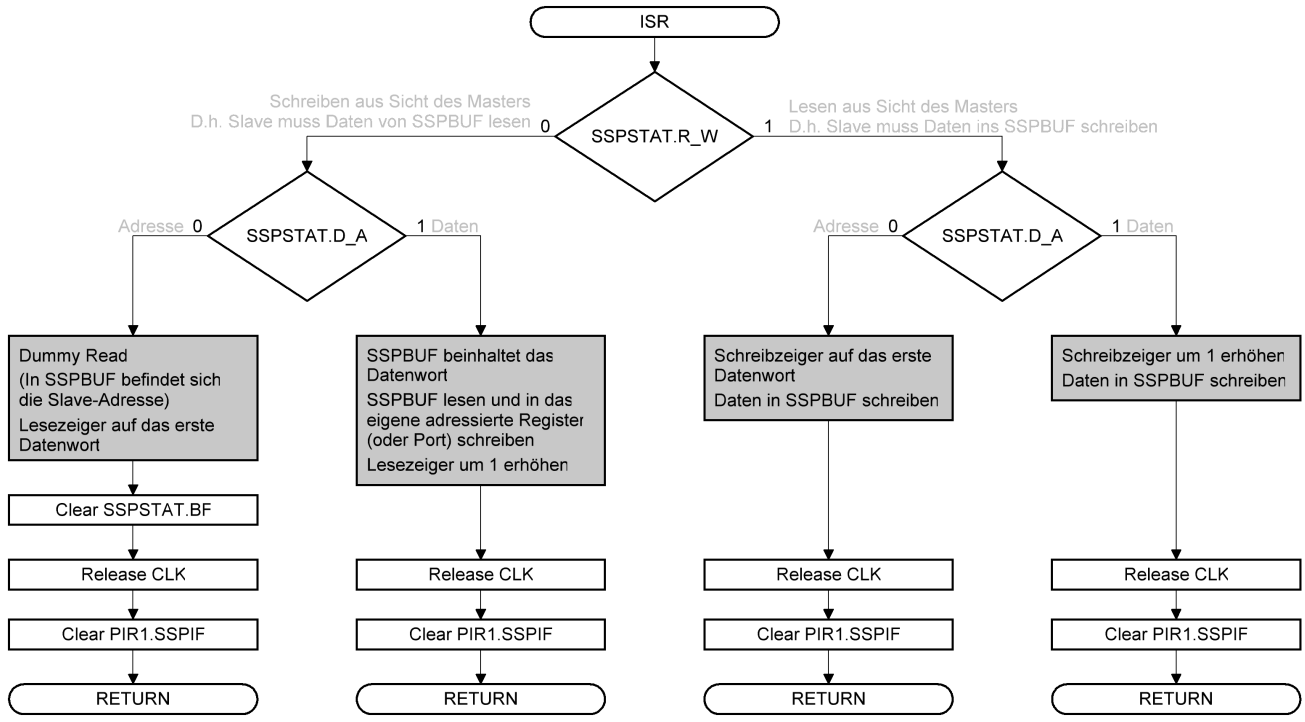


Abbildung 5: PIC16Fxx als I²C-Slave, Flussdiagramm (Interrupt-Service-Routine)

Anmerkungen:

- Die beiden Demonstrationsbeispiel in Abschnitt 5 ab Seite 10 zeigen diese Interrupt-Service-Routine an einem einfachen konkreten Beispiel
- Die Slave-Adresse wird in das Register `SSPADD` geschrieben und muss in der Interrupt-Service-Routine (ISR) nicht beachtet werden
- Das Interrupt-Flag `SSPIF` wird nur dann gesetzt, wenn die empfange Adresse mit dem Wert im Register `SSPADD` übereinstimmt
- Wichtig: Das Interrupt-Flag `SSPIF` muss in der ISR wieder gelöscht werden!

Noch ein Hinweise: Leider hat sich herausgestellt, dass meine Software nicht bei jedem PIC16Fxx funktioniert obwohl dieser gemäß dem Datenblatt als I²C-Slave geeignet ist. Den Grund dafür habe ich leider nicht gefunden bzw. ich hatte noch nicht das Bedürfnis diesen Fehler zu suchen. Die nachfolgende Tabelle zeigt mit welche PIC16Fxx-Typen „meine“ Software funktioniert und mit welchen nicht. Typen die in dieser Tabelle nicht aufscheinen habe ich (noch) nicht als I²C-Slave getestet.

| Typ | Anz.Pins | funktioniert oder funktioniert nicht |
|---|---|---|
| PIC16F887 | 40 | funktioniert :-) |
| PIC16F877 | 40 | funktioniert (leider nicht) :-( |

# 4 PIC16Fxx als I²C-Master

Dieser Abschnitt beschreibt nun wie ein PIC16Fxx-Mikrocontroller als Master für den I²C zu konfigurieren ist und welche Software-Routinen notwendig sind.

**Wichtig:**
Damit ein Mikrocontroller vom Typ PIC16Fxx als Master für den I²C-Bus verwendet werden kann muss dieser über das **MSSP**-Modul verfügen. MSSP steht für **M**aster **S**ynchronous **S**erial **P**ort.

## 4.1 Hardware

Zunächst die Hardwarebeschaltung. Abbildung 6 zeigt die minimal notwendige Beschaltung des PIC16Fxx-Mikrocontrollers.
Auffällig ist, dass hier kein Oszillator vorhanden ist. Dies ist auch nicht notwendig, da die Kommunikation via I²C-Bus nicht zeitkritisch ist, und daher ein interner Oszillator verwendet werden kann.

Wichtig sind hier nur die Pull-Up-Widerstände R2 bis R4 für die beiden I²C-Leitungen (SDA und SCL) sowie für die Interrupt-Leitung ($\overline{\texttt{INT}}$)[5]. An die Buchse K3 bzw. an den Stecker K4 kann ein beliebiges (5-poliges) Kabel angeschlossen werden. Buchsentyp sowie Steckertyp und Pinbelegung sind frei wählbar. Hier wird auch die Betriebsspannung sowie die Masse über das Kabel übertragen, so dass die I²C-Slaves mit dieser Betriebsspannung versorgt werden können.

Wie schon bei der Beschaltung für den PIC16Fxx als Slave wird auch der Master mittels ICSP[6] programmiert. Schaltungsteil um Buchse K5.

[5] Achtung: Dies ist kein Interrupt des Mikrocontroller PIC16Fxx. Es gibt I²C-Bausteine (Slaves) die neben den Leitungen SDA und SCL einen so genannten INT-Ausgang besitzen um den I²C-Master mitzuteilen, dass Daten zum Abholen bereit sind. z.B. siehe PCF8574.

[6] ICSP steht für **I**n-**C**ircuit-**S**erial-**P**rogramming

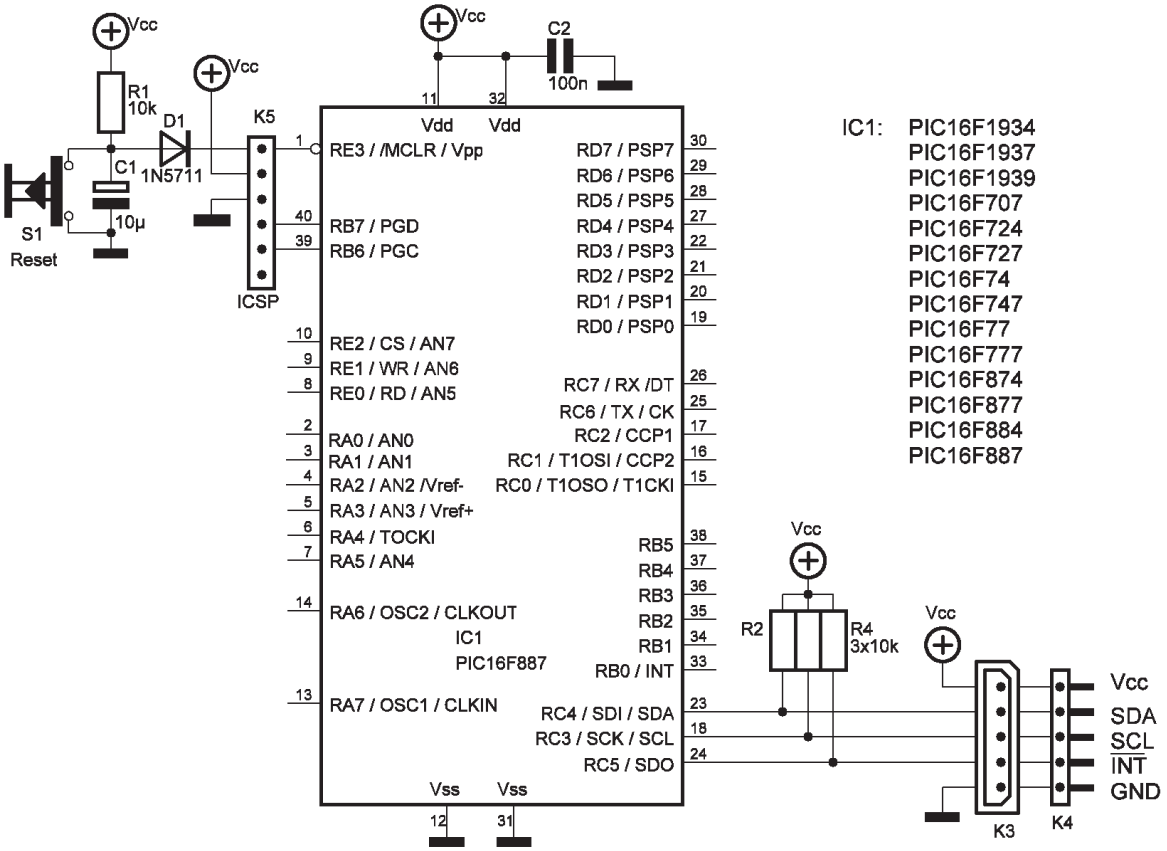


Abbildung 6: PIC16Fxx als I$^2$C-Master, Schaltung

## 4.2 Software

Der hier verwendete C-Compiler verfügt bereits über Unterprogramme zur Kommunikation via I$^2$C für den Master [4].

Wichtig ist dass zu Beginn die Taktfrequenz für den I$^2$C-Bus definiert werden muss. Diese Aufgabe übernimmt das Unterprogramm `I2C_Init()`. Für die Erzeugung der Start- und Stopp-Bedingung stehen die Unterprogramme `I2C_Start()` und `I2C_Stop()` zur Verfügung. Das Schreiben von 8-Bit-Daten erfolgt mit dem Unterpogramm `I2C_Wr()` und das Lesen von 8-Bit-Daten erfolgt mit dem Unterprogramm `I2C_Rd()`.

Wichtig ist dass die Pins SDA und SCL (beim Master) als Ausgang definiert werden müssen. (Vgl.: Bei der Verwendung eines PIC16Fxx als I$^2$C-Slave müssen die Pins SDA und SCL als Eingang definiert werden).

# 5 Demonstrationsbeispiele

Das folgende Beispiel dient nur zur Demonstration. Es zeigt eine mögliche Einbindung der zuvor beschriebenen ISR (Interrupt-Service-Routine) für den I$^2$C-Slave. Neben einem PIC16Fxx als I$^2$C-Slave dient ein weiterer Mikrocontroller vom Typ PIC16Fxx als I$^2$C-

Master. Weitere I²C-Slaves sind ein I/O-Expander vom Typ PCF8574 zur 8-Bit-Eingabe mittels Jumper und ein zweiter I/O-Expander vom gleichen Typ PCF8574 zur 8-Bit-Ausgabe mittels Leuchtdioden (LEDs).

Die Aufgabe des I²C-Masters ist es nun:

1. 8-Bit-Wert vom I²C-Eingabemodul (mit PCF8574) lesen und am PIC-I²C-Slave am Port D mittels 8 Leuchtdioden (LEDs) ausgeben.
2. 8-Bit-Wert vom PIC-I²C-Slave vom Port A einlesen und am I²C-Ausgabemodul (mit PCF8574) anzeigen (mittels 8 Leuchtdioden).

## 5.1 Schaltung (Hardware)

Die Abbildung 7 zeigt die gesamte Schaltung für dieses Demonstrationsbeispiel.

Der schwarz gezeichnete Schaltungsteil kennzeichnet den PIC-I²C-Slave. Dieser Schaltungsteil entspricht der Schaltung aus Abbildung 2 (Seite 5). Zusätzlich sind hier die Jumper und Pull-Up-Widerstände am Port A, sowie die Leuchtdioden und die dazugehörigen Vorwiderstände am Port D (grau dargestellt). Der externe Oszillator X1 mit den Kondensatoren C2 und C3 sowie dem Widerstand R2 werden hier nicht benötigt, da der interne Oszillator verwendet wird. Diese sind daher ebenfalls grau gekennzeichnet. Weiters sind die Jumper JP1 bis JP10 so wie in Abbildung 7 zu setzen. JP1 bis JP6 definieren die I²C-Slave-Adresse. Hier `0011000 R/W`. Als Mikrocontroller für den I²C-Slave wurde hier der Typ PIC16F887 verwendet.

Der zweite Mikrocontroller (IC2) ist ebenfalls vom gleichen Typ (PIC16F887) und dient hier als I²C-Master. Dieser Schaltungsteil ist genau wie in Abschnitt 4.1 (Seite 9) beschrieben. (Wichtig sind hier die Pull-Up-Widerstände an den I/O-Pins SCL, SDA und $\overline{\texttt{INT}}$).

An den I²C-Bus sind neben dem PIC-I²C-Slave (rund um IC1) auch die beiden I/O-Expander vom Typ PCF8574 angeschlossen (IC3 und IC4). Wichtig sind die I²C-Slave-Adressen die hier ebenfalls mit Jumpern ausgewählt werden können. Bei IC3 sind die Jumper für A1 und A2 gesteckt, während bei IC4 nur der Jumper für A2 gesteckt ist. Es versteht sich von selbst, dass beide I/O-Expander unterschiedliche Slaveadressen besitzen müssen, und dass die Slaveadresse des PIC-I²C-Slave sich von denen der I/O-Expander unterscheiden muss!

Die Spannungsversorgung besteht hier aus einem Labornetzgerät und beträgt 5V.

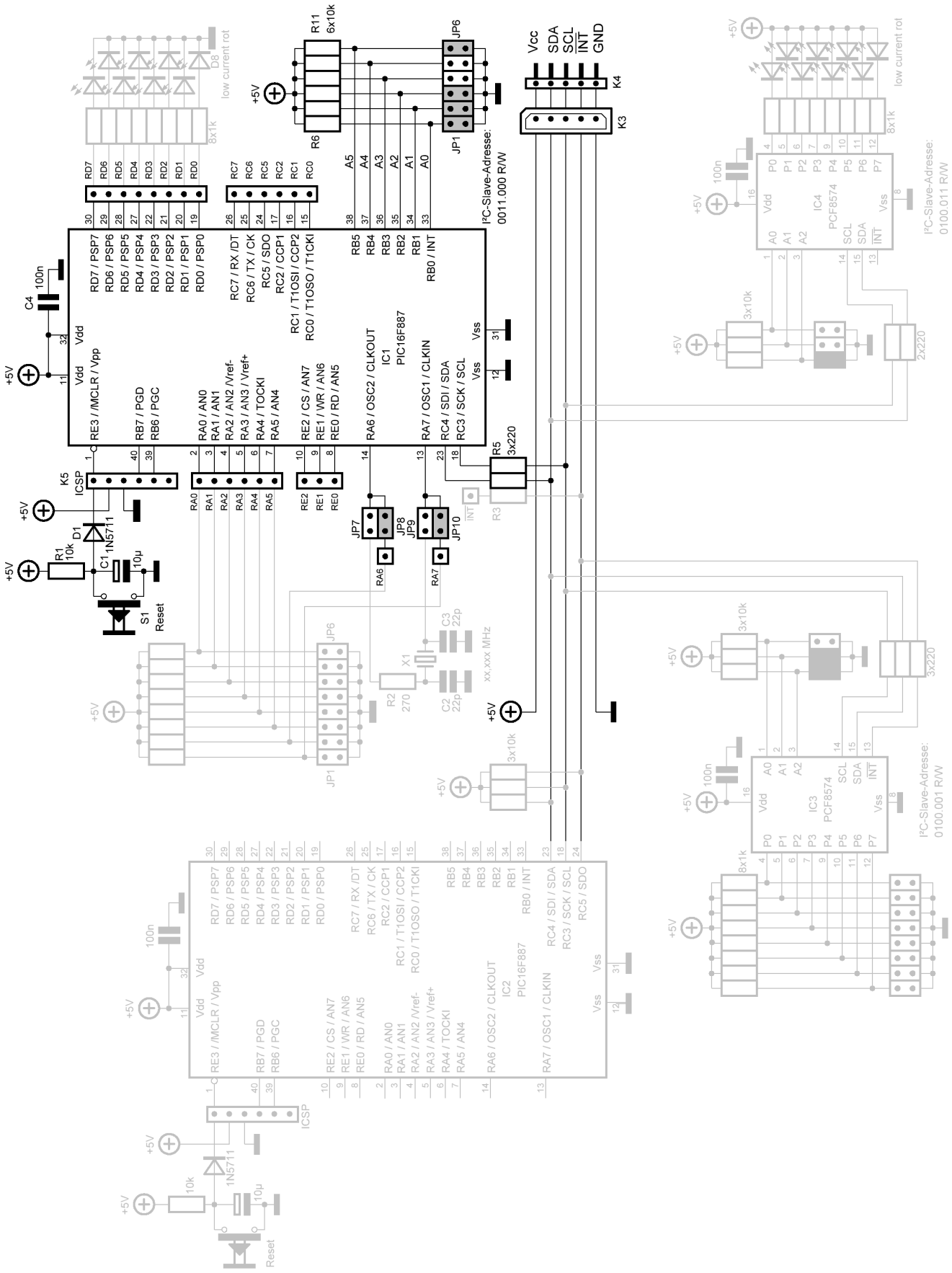


Abbildung 7: Demonstrationsbeispiel (Schaltung)

## 5.2 Software für 1. Demo (Slave)

Beim ersten Demonstrationsbeispiel sendet der I²C-Master nur **ein** Datenbyte zum PIC-I²C-Slave, bzw. liest anschließend auch nur **ein** Datenbyte vom PIC-I²C-Slave.

**Beachte:**
Da bei diesem ersten Demonstrationsbeispiel nur ein Datenbyte gesendet bzw. nur ein Datenbyte gelesen wird, vereinfacht sich das Flussdiagramm (aus Abbildung 5, Seite 8). Es sind nämlich kein Lesezeiger und kein Schreibzeiger notwendig. Siehe auch Abschnitt 5.3, ab Seite 20.

```
1  /*****************************************************************************/
2  /* Demo zur Ansteuerung des PIC16Fxx als I2C-Slave gemaess Application Note AN734      */
3  /*                                                                           */
4  /* Slave                                                                     */
5  /*                                                                           */
6  /* Demo 1: jeweils nur ein Byte vom Port A lesen und an den Master senden bzw. ein     */
7  /*         Byte vom Master lesen und am Port D ausgeben                      */
8  /*                                                                           */
9  /* Mikrocontroller:       PIC16F887                                          */
10 /* Takt:                  interner Takt (4.0 MHz)                            */
11 /*                                                                           */
12 /* Compiler:              mikroC V8.2                                        */
13 /*                                                                           */
14 /* Entwickler:            Stefan Buchgeher                                   */
15 /* Entwicklungsbeginn der Software: 19. Februar 2014                         */
16 /* Funktionsfaehig seit:  2. Maerz 2014                                      */
17 /* Letzte Bearbeitung:    4. Juli 2014                                       */
18 /*****************************************************************************/
19
20 /***************** Include-Dateien ******************************************/
21 /* keine Include-Dateien */
22
23
24 /***************** Strukturen ***********************************************/
25 /* keine Strukturen verwendet */
26
27
28 /***************** Externe Register *****************************************/
29 /* keine externen Register */
30
31
32 /***************** Bits in den externen Registern ***************************/
33 /* keine externen Register */
34
35
36 /***************** Globale Register *****************************************/
37 char                cDummy;
38
39
40 /***************** Bits in den globalen Registern ***************************/
41 /* keine globalen Register */
42
43
44 /***************** Portbelegung *********************************************/
45 /* \INT-Ausgang des PCF8574 */
46 //#define nINTPCF8574              PORTC.F5
47
48
49 /***************** Konstanten ***********************************************/
50 /* kein Konstanten */
51
52
53 /***************** Tabellen *************************************************/
```

```
54  /* keine Tabellen */
55
56
57  /****************** Funktionsprototypen ****************************************/
58  /* Unterprogramm zur Initialisierung des Mikroccntrollers */
59  void Init(void);
60
61
62  /****************** Interrupt-Service-Routine (ISR) ***********************************/
63
64  /*****************************************************************************************/
65  /* Interrupt Service Routine:                                                            */
66  /*                                                                                       */
67  /* Aufruf:                                                                               */
68  /*   SSP Interrupt Flag (SSPIF), nach jedem I2C-Event (wenn der Master eine Adresse    */
69  /*     auf den I2C-Bus schreibt oder wenn sich Daten am I2C-Bus befinden                 */
70  /*                                                                                       */
71  /* Aufgaben:                                                                             */
72  /*   + Je nach Status sind unterschiedliche Aktionen auszufuehren: siehe Kommentare im */
73  /*     Quellcode                                                                         */
74  /*   + Nach jeder Aktion muss das SSP-Interrupt-Flag SSPIF wieder geloeschen werden    */
75  /*                                                                                       */
76  /* Achtung:                                                                              */
77  /*   Der Status Lesen bzw. Schreiben erfolgt immer aus Sicht des Master!!               */
78  /*****************************************************************************************/
79  void interrupt (void)                          // Interruptroutine
80  {
81      if (SSPSTAT.R_W == 0)
82      {
83          // Schreibzugriffe (aus Sicht des Masters, d.h. Slave muss Daten aus dem Sende-
84          // und Empfangsbuffer (SSPBUF) lesen)
85          if (SSPSTAT.D_A == 0)
86          {
87              // State 1: I2C write operation, last byte was an address byte
88              // SSPSTAT bits: S=1, D/A=0, R/W=0, BF=1
89              cDummy = SSPBUF;
90              SSPSTAT.BF = 0;                    // Flag Buffer-Full loeschen
91              SSPCON.CKP = 1;                    // Clockstretching erlauben
92              PIR1.SSPIF = 0;                    // SSP-Interrupt-Flag wieder loeschen
93              return;
94          }
95          else
96          {
97              // State 2: I2C write operation, last byte was a data byte
98              // SSPSTAT bits: S=1, D/A=1, R/W=1, BF=1
99              PORTD = SSPBUF;                    // Datenbyte von SSPBUF lesen und am
100                                                 //   Port D ausgeben
101             SSPCON.CKP = 1;                    // Clockstretching erlauben
102             PIR1.SSPIF = 0;                    // SSP-Interrupt-Flag wieder loeschen
103             return;
104         }
105     }
106     else
107     {
108         // Lesezugriff (aus Sicht des Masters, d.H. Slave muss Daten in Sende- und
109         // Empfangsbuffer (SSPBUF) schreiben)
110         if (SSPSTAT.D_A == 0)
111         {
112             // State 3: I2C read operation, last byte was an address byte
113             // SSPSTAT bits: S=1, D/A=0, R/W=1, BF=0
114             SSPBUF = PORTA;                    // Datenbyte von PortA lesen und in
115                                                //   SSPBUF schreiben
116             SSPCON.CKP = 1;                    // Clockstretching erlauben
117             PIR1.SSPIF = 0;                    // SSP-Interrupt-Flag wieder loeschen
118             return;
119         }
120         else
121         {
122             // State 4: I2C read operation, last byte was a data byte
123             // SSPSTAT bits: S=1, D/A=1, R/W=0, BF=0
124             SSPBUF = cDummy;
```

```
125             SSPCON.CKP = 1;                     // Clockstretching erlauben
126             PIR1.SSPIF = 0;                     // SSP-Interrupt-Flag wieder loeschen
127             return;
128         }
129     }
130 }
131
132
133 /****************** Unterprogramme und Funktionen ************************************/
134
135 /*****************************************************************************************/
136 /* Init:                                                                                 */
137 /*                                                                                       */
138 /* Aufgabe:                                                                              */
139 /*   Initialisierung des Prozessor:                                                      */
140 /*      + internen Oszillator konfigurieren                                              */
141 /*      + Ports auf Digital I/O umschalten und als Ein- oder Ausgang konfigurieren       */
142 /*      + I2C-Hardware-Modul als Slave konfigurieren                                     */
143 /*      + I2C Slave Adresse von Port B lesen und in SSPADD schreiben                     */
144 /*      + Interrupt freigeben                                                            */
145 /*                                                                                       */
146 /* Uebergabeparameter:                                                                   */
147 /*   keiner                                                                              */
148 /*                                                                                       */
149 /* Rueckgabeparameter:                                                                   */
150 /*   keiner                                                                              */
151 /*****************************************************************************************/
152 void Init(void)
153 {
154     char cI2CAdresse;                   //I2C-Slave-Adresse
155
156
157     // (internen) Oszillator konfigurieren
158     OSCCON = 0b01100001;                // Sync Oscillator Control Register
159 /*              +------------------- Bit 7 Reserve
160                  +++----------------- Bit 6-4 (IRCF2:IRCF0): Internal Oscillator Frequency
161                     ||||                    Select bits
162                     ||||                 000 : 31 kHz (LFINTOSC)
163                     ||||                 001 : 125 kHz
164                     ||||                 010 : 250 kHz
165                     ||||                 011 : 500 kHz
166                     ||||                 100 : 1 MHz
167                     ||||                 101 : 2 MHz
168                     ||||               -> 110 : 4 MHz (default)
169                     ||||                 111 : 8 MHz
170                    +------------------- Bit 3 (OSTS): Oscillator Start-up Time-out Status
171                     |||                    (Read Only)
172                     |||                  0 : Device is running from the clock defined
173                     |||                      by FOSC2:FOSC0 of the CONFIG1 register
174                     |||                  1 : Device is running from the internal
175                     |||                      oscillator (HFINTOSC or LFINTOSC)
176                     +-------------------- Bit 2 (HTS): HFINTOSC Status bir
177                      ||                    (High Frequency - 125kHz to 8MHz)
178                      ||                    (Read Only)
179                      ||                  0 : HFINTOSC is not stable
180                      ||                  1 : HFINTOSC is stable
181                      +-------------------- Bit 1 (LTS): LFINTOSC Status bit
182                       |                    (Low Frequency - 31kHz)
183                       |                    (Read Only)
184                       |                  0 : LFINTOSC is not stable
185                       |                  1 : LFINTOSC is stable
186                       +-------------------- Bit 0 (SCS): System Clock Select bit
187                                            0 : Clock source defined by FOSC2:FOSC0
188                                                of the CONFIG1 register
189                                          ->  1 : Internal oscillator is used for system
                                                clock
190 */
191
192     // Alle Ports auf digital I/O konfigurieren
193     ANSEL = 0b00000000;                 // Analog Select Register
194                 ||||||||                //  (0: Digital I/O, 1: Analogeingang)
```

```
195 /*            +------------------------ Bit 7: Port RE2/AN7 ist hier Digital I/O
196                +------------------------ Bit 6: Port RE1/AN6 ist hier Digital I/O
197                 +------------------------ Bit 5: Port RE0/AN5 ist hier Digital I/O
198                  +------------------------ Bit 4: Port RA5/AN4 ist hier Digital I/O
199                   +------------------------ Bit 3: Port RA3/AN3 ist hier Digital I/O
200                    +------------------------ Bit 2: Port RA2/AN2 ist hier Digital I/O
201                     +------------------------ Bit 1: Port RA1/AN1 ist hier Digital I/O
202                      +------------------------ Bit 0: Port RA0/AN0 ist hier Digital I/O
203 */
204
205     ANSELH = 0b00000000;            // Analog Select Register
206                 ||||||||            //  (0: Digital I/O, 1: Analogeingang)
207 /*              ++------------------------ Bit 6,7 Reserve
208                   +------------------------ Bit 5: Port RB5/AN13 ist hier Digital I/O
209                    +------------------------ Bit 4: Port RB0/AN12 ist hier Digital I/O
210                     +------------------------ Bit 3: Port RB4/AN11 ist hier Digital I/O
211                      +------------------------ Bit 2: Port RB1/AN10 ist hier Digital I/O
212                       +------------------------ Bit 1: Port RB9/AN9 ist hier Digital I/O
213                        +------------------------ Bit 0: Port RB2/AN8 ist hier Digital I/O
214 */
215
216     // Ports konfigurieren
217     TRISA = 0b11111111;             // Richtungsregister Port A (0: Ausgang, 1: Eingang)
218 /*            +------------------------ Bit 7 Port RA7: hier Eingang (Jumper Pin 7)
219                +------------------------ Bit 6 Port RA6: hier Eingang (Jumper Pin 6)
220                 +------------------------ Bit 5 Port RA5: hier Eingang (Jumper Pin 5)
221                  +------------------------ Bit 4 Port RA4: hier Eingang (Jumper Pin 4)
222                   +------------------------ Bit 3 Port RA3: hier Eingang (Jumper Pin 3)
223                    +------------------------ Bit 2 Port RA2: hier Eingang (Jumper Pin 2)
224                     +------------------------ Bit 1 Port RA1: hier Eingang (Jumper Pin 1)
225                      +------------------------ Bit 0 Port RA0: hier Eingang (Jumper Pin 0)
226 */
227
228     TRISB = 0b00111111;             // Richtungsregister Port B (0: Ausgang, 1: Eingang)
229 /*            +------------------------ Bit 7 Port RB7: hier unbenutzt
230                +------------------------ Bit 6 Port RB6: hier unbenutzt
231                 +------------------------ Bit 5 Port RB5: hier Eingang (I2C-Slave-Adresse A5)
232                  +------------------------ Bit 4 Port RB4: hier Eingang (I2C-Slave-Adresse A4)
233                   +------------------------ Bit 3 Port RB3: hier Eingang (I2C-Slave-Adresse A3)
234                    +------------------------ Bit 2 Port RB2: hier Eingang (I2C-Slave-Adresse A2)
235                     +------------------------ Bit 1 Port RB1: hier Eingang (I2C-Slave-Adresse A1)
236                      +------------------------ Bit 0 Port RB0: hier Eingang (I2C-Slave-Adresse A0)
237 */
238
239     TRISC = 0b00011000;             // Richtungsregister Port C (0: Ausgang, 1: Eingang)
240 /*            +------------------------ Bit 7 Port RC7: hier unbenutzt
241                +------------------------ Bit 6 Port RC6: hier unbenutzt
242                 +------------------------ Bit 5 Port RC5: hier unbenutzt
243                  +------------------------ Bit 4 Port RC4: hier Eingang (SDA)
244                   +------------------------ Bit 3 Port RC3: hier Eingang (SCL)
245                    +------------------------ Bit 2 Port RC2: hier unbenutzt
246                     +------------------------ Bit 1 Port RC1: hier unbenutzt
247                      +------------------------ Bit 0 Port RC0: hier unbenutzt
248 */
249
250     TRISD = 0b00000000;             // Richtungsregister Port D (0: Ausgang, 1: Eingang)
251 /*            +------------------------ Bit 7 Port RD7: hier Ausgang (LED D7)
252                +------------------------ Bit 6 Port RD6: hier Ausgang (LED D6)
253                 +------------------------ Bit 5 Port RD5: hier Ausgang (LED D5)
254                  +------------------------ Bit 4 Port RD4: hier Ausgang (LED D4)
255                   +------------------------ Bit 3 Port RD3: hier Ausgang (LED D3)
256                    +------------------------ Bit 2 Port RD2: hier Ausgang (LED D2)
257                     +------------------------ Bit 1 Port RD1: hier Ausgang (LED D1)
258                      +------------------------ Bit 0 Port RD0: hier Ausgang (LED D0)
259 */
260
261     TRISE = 0b00000000;             // Richtungsregister Port E (0: Ausgang, 1: Eingang)
262                 ||||||||            //    und Parallel Slave Port Status/Control Bits
263 /*              +------------------------ Bit 7 (IBF): Input Buffer Full Status bit
264                  |||||||                    (Read only)
265                  |||||||                 0 : No word has been received
```

```
266                 |||||||                     1 : A word has been received and is waiting to
267                 |||||||                         be read by the CPU
268                 +-------------------- Bit 6 (OBF): Output Buffer Full Status bit
269                  ||||||                  (Read only)
270                  ||||||                  0 : The output buffer has been read
271                  ||||||                  1 : The output buffer still holds a previously
272                  ||||||                      written word
273                  +-------------------- Bit 5 (IBOV): Input Buffer Overflow Detect bit
274                   |||||                   (in Microprocessor mode)
275                   |||||                  0 : No overflow occurred
276                   |||||                  1 : A write occurred when a previously input
277                   |||||                      word has not been read (must be cleared
278                   |||||                      in Software)
279                   +-------------------- Bit 4 (PSPMODE) : Parallel Slave Port Mode Select
280                    ||||                -> 0 : PORTD functions in general Purpose I/O mode
281                    ||||                   1 : PORTD functions in Parallel Slave Port mode
282                    +-------------------- Bit 3 Reserve
283                     +-------------------- Bit 2 Port RE2: hier unbenutzt
284                      +-------------------- Bit 1 Port RE1: hier unbenutzt
285                       +-------------------- Bit 0 Port RE0: hier unbenutzt
286 */
287
288     PORTA = 0;
289     PORTB = 0;
290     PORTC = 0;
291     PORTD = 0;
292     PORTE = 0;
293
294
295     // I2C-Hardware-Modul als Slave konfigurieren
296     SSPCON = 0b00100110;              // Sync Serial Port Control Register 1
297 /*                  +-------------------- Bit 7 (WCOL): Write Collision Detect bit
298                      |||||||                 Master mode:
299                      |||||||                 0 : No collision
300                      |||||||                 1 : A write to SSPBUF was attempt while the I2C
301                      |||||||                     conditions were not valis
302                      |||||||                 Slave mode:
303                      |||||||                 0 : No collision
304                      |||||||                 1 : SSPBUF register is written while still
305                      |||||||                     transmitting the previous word (must be
306                      |||||||                     cleared in software)
307                     +-------------------- Bit 6 (SSPOV): Receive Overflow Indicator bit
308                      ||||||                 0 : No overflow
309                      ||||||                 1 : A new byte is received while SSPBUF holds
310                      ||||||                     previous data (weiter siehe Datenblatt!!)
311                      ||||||                     (must be cleared in software)
312                     +-------------------- Bit 5 (SSPEN): Synchronous Serial Port Enable bit
313                      |||||                  0 : Disables serial port and configures these
314                      |||||                      Pins as I/O port pins
315                      |||||                ->   1 : Enables the serial port an configures
316                      |||||                     SCK, SDO, SDI, SS / SDA, SCL pins as the
317                      |||||                     source of the serial port pins
318                     +-------------------- Bit 4 (CKP): Clock Polarity Select bit
319                      ||||                   In SPI mode:
320                      ||||                   0 : Idle state for clock is a low level
321                      ||||                   1 : Idle state for clock is a high level
322                      ||||                   In I2C Slave mode: SCK release control
323                      ||||                ->   0 : Holds clock low (clock stretch). (Used to
324                      ||||                       ensure data setup time.)
325                      ||||                   1 : Enable clock
326                      ||||                   In I2C Master mode: Unused in this mode
327                     ++++-------------------- Bit 3-0 (SSPM3:SSPM0): SSP Mode Select bits
328                                              0000 : SPI Master mode, clock = Fosc/4
329                                              0001 : SPI Master mode, clock = Fosc/16
330                                              0010 : SPI Master mode, clock = Fosc/64
331                                              0011 : SPI Master mode, clock = TMR2 output/2
332                                              0100 : SPI Slave mode, clock = SCK pin, nSS pin
333                                                  control enabled
334                                              0101 : SPI Slave mode, clock = SCK pin, nSS pin
335                                                  control disabled. nSS can be used as I/O
336                                             -> 0110 : I2C Slave mode, 7-bit address
```

```
                                        0111 : I2C Slave mode, 10-bit address
                                        1000 : I2C Master mode,
                                               clock = Fosc/(4*(SSPADD+1))
                                        1001 : Reserved
                                        1010 : Reserved
                                        1011 : I2C Firmware Controlled Master mode
                                               (slave idle)
                                        1100 : Reserved
                                        1101 : Reserved
                                        1110 : I2C Firmware Controlled Master mode,
                                               7-bit address with START and STOP bit
                                               interrupts enabled
                                        1111 : I2C Firmware Controlled Master mode,
                                               10-bit address with START and STOP bit
                                               interrupts enabled
*/

    SSPCON2 = 0b00110110;           // Sync Serial Port Control Register 2
/*              +------------------ Bit 7 (GCEN): General Call Enable bit (In I2C Slave
                |||||||                 mode only)
                |||||||         ->      0 : General Call address disabled
                |||||||                 1 : Enable interrupt when a general call
                |||||||                     address (0000h) ist received in the SSPSR
                +------------------ Bit 6 (ACKSTAT) : Acknowledge Status bit (In I2C
                 ||||||                 Master mode only)
                 ||||||                 0 : Acknowledge was received from slave
                 ||||||                 1 : Acknowledge was not received from slave
                 +------------------ Bit 5 (ACKDT): Acknowledge Data bit (In I2C Master
                  |||||                 mode only):
                  |||||                 Value that will be transmitted when the user
                  |||||                 iniziates an Acknowledge sequence at the end
                  |||||                 of a receive
                  |||||                 0 : Acknowledge
                  |||||                 1 : Not Acknowledge
                  +------------------ Bit 4 (ACKEN): Acknowledge Sequence Enable bit
                   ||||                 (In I2C Master mode only):
                   ||||                 0 : Acknowledge sequence idle
                   ||||                 1 : Initiate Acknowledge sequence on SDA and
                   ||||                     SCL pins and transmit ACKDT data bit.
                   ||||                     Automatically cleared by hardware
                   +------------------ Bit 3 (RCEN): Receive Enable bit (In I2C Master
                    |||                 mode only)
                    |||                 0 : Receive idle
                    |||                 1 : Enables Receive mode for I2C
                    +------------------ Bit 2 (PEN): STOP Condition Enable bit (In I2C
                     ||                 Master mode only)
                     ||                 0 : STOP condition idle
                     ||                 1 : Initiate STOP condition on SDA and SCL pins
                     ||                     Automatically cleared by hardware
                     +------------------ Bit 1 (RSEN): Repeated START Condition Enable bit
                      |                 (In I2C Master mode only)
                      |                 0 : Repeated START condition idle
                      |                 1 : Initiate Repeated START condition on SDA
                      |                     and SCL pins
                      |                     Automatically cleared by hardware
                      +------------------ Bit 0 (SEN): START Condition Enable bit (In I2C
                                          Master mode only)
                                          0 : START condition idle
                                          1 : Initiate START cond. on SDA and SCL pins
                                             Automatically cleared by hardware
*/

    SSPSTAT = 0b10000000;

    // I2C Slave Adresse von Port B lesen und in SSPADD schreiben
    // Hier 0 A5 A4 A3 A2 A1 A0 R/nW
    cI2CAdresse = 0b00111111;       // Maske
    cI2CAdresse = cI2CAdresse & PORTB;
    cI2CAdresse = cI2CAdresse << 1;
    cI2CAdresse.F0 = 0;             // Schreibzugriff aus Sicht des Master
    SSPADD = cI2CAdresse;
```

```
408
409
410     // Interrupt freigeben
411     PIE1 = 0b00001000;                  // Peripheral Interrupt Enable Register 1 (Bank 1)
412 /*            +---------------------- Bit 7  Reserve
413              +----------------------- Bit 6 (ADIE): A/D Converter Interrupt Enable bit
414               ||||||                 ->   0 : Disables the ADC interrupt
415               ||||||                      1 : Enables the ADC interrupt
416              +----------------------- Bit 5 (RCIE): EUSART Receive Interrupt Enable bit
417               |||||                  ->   0 : Disables the EUSART Receive Interrupt
418               |||||                       1 : Enables the EUSART Receive Interrupt
419              +----------------------- Bit 4 (TXIE): EUSART Transmit Interrupt Enable bit
420               ||||                   ->   0 : Disables the Transmit Interrupt Enable bit
421               ||||                        1 : Enables the Transmit Interrupt Enable bit
422              +----------------------- Bit 3 (SSPIE): Master Synchronous Serial Port (MSSP)
423               |||                                   Interrupt Enable bit
424               |||                        0 : Disables the MSSP interrupt
425               |||                   ->   1 : Enables the MSSP interrupt
426              +----------------------- Bit 2 (CCP1IF): CCP1 Interrupt Enable bit
427               ||                    ->   0 : Disables the CCP1 interrupt
428               ||                         1 : Enables the CCP1 interrupt
429              +----------------------- Bit 1 (TMR2IE): Timer 2 to PR2 Match Interrupt
                      Enable bit
430               |                     ->   0 : Disables the Timer 2 to PR2 Match Interrupt
431               |                          1 : Enables the Timer 2 to PR2 Match Interrupt
432              +----------------------- Bit 0 (TMR1IE): Timer 1 Overflow Interrupt Enable
                     bit
433                                     ->   0 : Disables Timer 1 Overflow Interrupt
434                                          1 : Enables Timer 1 Overflow Interrupt
435 */
436
437     INTCON = 0b11000000;            // Interrupt-Control-Register (Bank 1)
438 /*            +----------------------- Bit 7 (GIE): Global Interrupt Enable bit
439               |||||||                       0 : Disables all interrupts
440               |||||||                ->   1 : Enables all unmasked interrupts
441              +----------------------- Bit 6 (PEIE): Peripheral Interrupt Enable bit
442               ||||||                       0 : Disables all peripheral interrupts
443               ||||||                 ->   1 : Enables all unmasked peripheral interrupts
444              +----------------------- Bit 5 (T0IE): Timer 0 Overflow Interrupt Enable Bit
445               |||||                  ->   0 : Disabled
446               |||||                       1 : Enabled
447              +----------------------- Bit 4 (INTE): RB0/INT External Interrupt Enable Bit
448               ||||                   ->   0 : Disabled
449               ||||                        1 : Enabled
450              +----------------------- Bit 3 (RPIE): RB Port Change Interrupt Enable Bit
451               |||                    ->   0 : Disabled
452               |||                         1 : Enabled
453              +----------------------- Bit 2 (T0IF): Timer 0 Overflow Interrupt Flag bit
454               ||                     ->   0 : TMR0 register did not overflow
455               ||                          1 : TMR0 register has overflowed
456              +----------------------- Bit 1 (INTF): RB0/INT External Interrupt Flag bit
457               |                      ->   0 : RB0/INT external interrupt did not occur
458               |                           1 : RB0/INT external interrupt occurred
459              +----------------------- Bit 0 (RBIF): RB Port Change Interrupt Flag bit
460                                      ->   0 : None of RB7:RB4 pins have changed state
461                                           1 : One of the RB7:RB4 pins changed state
462 */
463 }
464
465
466 /***************** Hauptprogramm ************************************************/
467
468 /*****************************************************************************/
469 /* Aufgaben des Hauptprogramms:                                              */
470 /*   + Controller initialisieren (Unterprogramm Init)                        */
471 /*   + Alles andere erfolgt in der ISR!!                                     */
472 /*****************************************************************************/
473 void main(void)
474 {
475     // Controller initialisieren
476     Init();
```

```
477
478
479     // Endlosschleife
480     while (1);
481 }
```

Listing 1: `I2C_Demo_PIC_Slave.c` (Demo 1)

## 5.3 Anmerkungen zur Software für 1. Demo (Slave)

Der Quellcode für dieses Demonstrationsbeispiel wurde in C mit der freien Demoversion des mikroC-Compilers erstellt[7].

Die Software (für den PIC-I²C-Slave) besteht im Wesentlichen aus den folgenden Programmteilen:

- kurzes Hauptprogramm am Ende des Quellcodes, Zeilen 473 bis 481.
- 1 Unterprogramm zur Initialisierung des PIC16F887 (Unterprogramm `Init`, Zeilen 152 bis 463).
- kurze Interrupt-Service-Routine (kurz: ISR), Zeilen 79 bis 130.

**Hauptprogramm:**
Die Aufgabe des Hauptprogramms (Zeilen 473 bis 481) ist hier „nur“ das Initialisieren des Mikrocontrollers. Dies erfolgt mit dem Unterprogramm `Init` (Zeile 476). Alles andere erfolgt in der Interrupt-Service-Routine (ISR).

**Unterprogramm `Init`:**
Das Unterprogramm `Init` (Zeilen 152 bis 463) dient zur Initialisierung des Mikrocontrollers. Hier werden die benötigten Hardwaremodule konfiguriert. Für den PIC-I²C-Slave sind das zunächst der interne Oszillator (Zeile 158), sämtliche Ports als Ein- oder Ausgang kunfigurieren und als digitale Ein-/Ausgabepins konfigurieren (Zeilen 193 bis 292). Wichtig ist dass die Portpins (SDA und SCL) als Eingang konfiguriert werden.
Ganz wichtig ist hier natürlich das SSP-Modul, denn dieses ist für die Kommunikation via I²C zuständig. Im Register `SSPCON` wird u.a. ausgewählt dass der Mikrocontroller im I²C-Slave-Mode betrieben wird (Zeile 296). Wichtig ist auch dass der I²C-Slave über eine Slave-Adresse verfügt. Dafür ist das Register `SSPADD` zuständig. Da bei diesem Demonstrationsbeispiel die Slave-Adresse am Port B des Mikrocontrollers ausgewählt werden kann muss diese vom Port B gelesen werden und die entsprechenden Bits müssen in das Register `SSPADD` geschrieben werden. Beachte: Bit 0 gibt die Datenflussrichtung aus Sicht des Masters an (also Master schreibt Daten oder liest Daten, Zeilen 403 bis 407). Als letzter Schritt muss noch der SSP-Interrupt freigegeben werden. Bit `SSPIE` im Register `PIE1` (Zeile 411). Weiters die Globale Interruptfreigabe (Bit `GIE` im Register `INTCON`), sowie das Bit `PEIE` ebenfalls im Register `INTCON` (Zeile 437).

[7]Die freie Demoversion ist nur auf eine Programmspeichergröße von 2k begrenzt, ansonst voll kompatibel zur Vollversion. Die Demoversion ist auf der Firmen-Webseite [3] downloadbar

**Interrupt-Service-Routine (kurz: ISR):**
Die SSP-ISR (Zeilen 79 bis 130) wird jedesmal aufgerufen wenn ein gesamtes Datenbyte vom SSP-Hardware-Modul empfangen wurde und dieses für den PIC-Slave mit der Adresse im Register `SSPADD` bestimmt ist. Jedesmal wenn diese SSP-ISR aufgerufen wird muss eine der folgenden vier Fälle ausgeführt werden (vgl. Flussdiagramm in Abbildung 5, Seite 8 jedoch hier ohne Lese- und Schreibezeiger, da nur ein Datenbyte gelesen bzw. geschrieben (gesendet) wird.).

Fall 1: Schreibzugriff aus Sicht des Masters, zuletzt wurde die Slave-Adresse empfangen (Bits: `SSPSTAT.R_W = 0, SSPSTAT.D_A = 0`):
In diesem Fall **muss** der Mikrocontroller (Slave) die empfangenen Daten aus dem SSP-Buffer (Register `SSPBUF`) lesen, kann den Inhalt von `SSPBUF` aber verwerfen, da es nur die Adresse und keine Daten enthält (Zeile 89). Hier wird der Inhalt von `SSPBUF` in ein Dummy-Register geschrieben. Wichtig ist das dass Register `SSPBUF` gelesen wird, weil dadurch Steuerbits für das Hardwaremodul verändert werden. Wichtig sind auch die folgenden Codezeilen 90 bis 93. Das SSP-Interrupt-Flag (`SSPIF`) muss ebenfalls gelöscht werden, da sonst dieser Interrupt sofort wieder ausgelöst wird!

Fall 2: Schreibzugriff aus Sicht des Masters, zuletzt wurde ein Datenbyte empfangen (Bits: `SSPSTAT.R_W = 0, SSPSTAT.D_A = 1`):
Jetzt handelt es sich tatsächlich um die Nutzdaten. Bei diesem Demonstrationsbeispiel wird nur ein 1 Datenbyte übertragen. Daher sind dies die einzigen Nutzdaten und können vom Register `SSPBUF` gelesen und in ein eigenes Register gespeichert werden, oder so wie hier direkt am Port D ausgegeben werden (Zeile 99). Auch hier sind die folgenden Codezeilen 101 bis 103 wichtig und dürfen nicht weggelassen werden!

Fall 3: Lesezugriff aus Sicht des Masters, zuletzt wurde die Slave-Adresse empfangen (Bits: `SSPSTAT.R_W = 1, SSPSTAT.D_A = 0`):
In diesem Fall müssen die zu sendenden Nutzdaten in das Register `SSPBUF` geschrieben werden. Hier, bei diesem Demonstrationsbeispiel ist dies der Inhalt von Port A (Zeile 114). Auch hier sind die folgenden Codezeilen 116 bis 118 wichtig und dürfen nicht weggelassen werden!

Fall 4: Lesezugriff aus Sicht des Masters, zuletzt wurde ein Datenbyte gesendet (Bits: `SSPSTAT.R_W = 1, SSPSTAT.D_A = 1`):
Meiner Ansicht nach dürfte bei diesem Demonstrationsbeispiel dieser Fall gar nicht auftreten, da vom Master nur ein Datenbyte gelesen werden darf. Daher wird in diesem Fall nur der Inhalt einer Dummy-Variable (`cDummy`) in das Register `SSPBUF` geschrieben (Zeile 124). Auch hier sind die folgenden Codezeilen 125 bis 127 wichtig und dürfen nicht weggelassen werden!

Mehr als das Lesen des `SSPBUF`-Registers oder das Schreiben in das `SSPBUF`-Register ist in der Interrupt-Service-Routine (ISR) nicht zu tun. Alles andere übernimmt das Hardware-Modul SSP des Mikrocontrollers.

**Sonstiges:**
Am Beginn des Quellcodes befinden sich die folgenden Definitionen und Einstellungen:

- Definition der globalen Register (hier nur `cDummy`, Zeile 37)
- Die Funktionsprototypen für die hier verwendeten Unterprogramme (hier nur für das Unterprogramm `Init`, Zeile 59)
- Ab Zeile 62 beginnt der Quellcode der ausreichend mit Kommentaren versehen ist

Ein kleiner Nachteil beim mikroC-Compiler ist, dass die Konfigurationsbits für den PIC-Mikrocontroller in der IDE gesetzt werden müssen. Ich persönlich würde diese lieber im Quellcode mit einer geeigneten Anweisung setzen.

Hier sind die notwendigen Einstellungen für den PIC16F887 für dieses Demonstrationsbeispiel (für Slave und Master):

- ✓ _INTOSCIO
- ✓ _WDT_OFF
- ✓ _PWRTE_OFF
- ✓ _MCLRE_ON
- ✓ _CP_OFF
- ✓ _CPD_OFF
- ✓ _BOR_OFF
- ✓ _IESO_OFF
- ✓ _FCMEN_OFF
- ✓ _LVP_OFF
- ✓ _DEBUG_OFF
- ✓ _WRT_OFF
- ✓ _WRT_1FOURTH

## 5.4 Software für 1. Demo (Master)

Für den I²C-Bus-Master wurde hier ebenfalls ein Mikrocontroller vom Typ PIC16F887 verwendet. Dieser hat hier folgende Aufgabe: Jedes mal wenn sich bei IC3 (PCF8574) zumindest ein Port-Pin (P0 bis P7) ändert erzeugt dieser Chip einen Interrupt, indem der Ausgang $\overline{\texttt{INT}}$ (Pin 13) den Pegel von High auf Low ändert. Daraufhin soll der Master diesen (geänderten) Eingang lesen und am PIC-I²C-Slave (am Port D) ausgeben. Danach soll der Master die Daten vom PIC-I²C-Slave (vom Port A) lesen und am IC4 (PCF8574) ausgeben.

Der Quellcode für den I²C-Bus-Master (mit dem PIC16F887, IC2) ist zur besseren Übersicht in mehrere Dateien aufgeteilt:

- `I2C_Demo_PIC_Master.c` beinhaltet das Hauptprogramm und das Unterprogramm zur Initialisierung des Mikrocontrollers (PIC16F887).
- `PCF8574.C` beinhaltet die Unterprogramme zur Datenübertragung mit dem PCF8574.
- `PCF8574.H` ist die Headerdatei zu `PCF8574.C`

Zuerst die Datei `I2C_Demo_PIC_Master.c`:

```
1  /*****************************************************************************/
2  /* Demo zur Ansteuerung des PIC16Fxx als I2C-Slave gemaess Application Note AN734      */
3  /*                                                                           */
4  /* Master                                                                    */
5  /*                                                                           */
6  /* Demo 1: jeweils nur ein Byte vom Port B lesen und an den Master senden bzw. ein     */
7  /*         Byte vom Master lesen und am Port D ausgeben                      */
8  /*                                                                           */
9  /* Mikrocontroller:       PIC16F887                                          */
10 /* Takt:                  interner Takt (4,0 MHz)                            */
11 /*                                                                           */
12 /* Compiler:              mikroC V8.2                                        */
13 /*                                                                           */
14 /* Entwickler:            Stefan Buchgeher                                   */
15 /* Entwicklungsbeginn der Software: 19. Februar 2014                         */
16 /* Funktionsfaehig seit:  2. Maerz 2014                                      */
17 /* Letzte Bearbeitung:    4. Juli 2014                                       */
18 /*****************************************************************************/
19
20 /***************** Include-Dateien ******************************************/
21 #include "PCF8574.H"
22 #include "PCF8574A.H"
23
24
25 /***************** Strukturen ***********************************************/
26 /* keine Strukturen verwendet */
27
28
29 /***************** Externe Register *****************************************/
30 /* keine externen Register */
31
32
33 /***************** Bits in den externen Registern ***************************/
34 /* keine externen Register */
35
36
37 /***************** Globale Register *****************************************/
38 /* keine globalen Register */
```

```
39
40
41 /****************** Bits in den globalen Registern ***********************************/
42 /* keine globalen Register */
43
44
45 /****************** Portbelegung *****************************************************/
46 /* \INT-Ausgang des PCF8574 */
47 #define nINTPCF8574                PORTC.F5
48
49
50 /****************** Konstanten *******************************************************/
51 /* Konstanten fuer I2C */
52 #define noACK                      0
53 #define ACK                        1
54
55
56 /****************** Tabellen *********************************************************/
57 /* keine Tabellen */
58
59
60 /****************** Funktionsprototypen **********************************************/
61 /* Unterprogramm zur Initialisierung des Mikroccntrollers */
62 void Init(void);
63
64
65 /****************** Unterprogramme und Funktionen ************************************/
66
67 /*************************************************************************************/
68 /* Init:                                                                             */
69 /*                                                                                   */
70 /* Aufgabe:                                                                          */
71 /*   Initialisierung des Prozessor:                                                  */
72 /*      + internen Oszillator konfigurieren                                          */
73 /*      + Ports konfigurieren                                                        */
74 /*                                                                                   */
75 /* Uebergabeparameter:                                                               */
76 /*   keiner                                                                          */
77 /*                                                                                   */
78 /* Rueckgabeparameter:                                                               */
79 /*   keiner                                                                          */
80 /*************************************************************************************/
81 void Init(void)
82 {
83     // (internen) Oszillator konfigurieren
84     OSCCON = 0b01100001;              // Sync Oscillator Control Register
85 /*             +---------------------- Bit 7 Reserve
86               +++---------------------- Bit 6-4 (IRCF2:IRCF0): Internal Oscillator Frequency
87                  ||||                        Select bits
88                  ||||                   000 : 31 kHz (LFINTOSC)
89                  ||||                   001 : 125 kHz
90                  ||||                   010 : 250 kHz
91                  ||||                   011 : 500 kHz
92                  ||||                   100 : 1 MHz
93                  ||||                   101 : 2 MHz
94                  ||||                -> 110 : 4 MHz (default)
95                  ||||                   111 : 8 MHz
96                 +----------------------- Bit 3 (OSTS): Oscillator Start-up Time-out Status
97                  |||                       (Read Only)
98                  |||                    0 : Device is running from the clock defined
99                  |||                       by FOSC2:FOSC0 of the CONFIG1 register
100                 |||                    1 : Device is running from the internal
101                 |||                       oscillator  (HFINTOSC or LFINTOSC)
102                +------------------------ Bit 2 (HTS): HFINTOSC Status bir
103                 ||                        (High Frequency - 125kHz to 8MHz)
104                 ||                        (Read Only)
105                 ||                     0 : HFINTOSC is not stable
106                 ||                     1 : HFINTOSC is stable
107                +------------------------- Bit 1 (LTS): LFINTOSC Status bit
108                 |                         (Low Frequency - 31kHz)
109                 |                         (Read Only)
```

```
110                       |                    0 : LFINTOSC is not stable
111                       |                    1 : LFINTOSC is stable
112                       +------------- Bit 0 (SCS): System Clock Select bit
113                                            0 : Clock source defined by FOSC2:FOSC0
114                                                of the CONFIG1 register
115                                      ->    1 : Internal oscillator is used for system
                                                 clock
116 */
117
118
119     // Ports konfigurieren
120     TRISC = 0b00100000;             // Richtungsregister Port C (0: Ausgang, 1: Eingang)
121 /*            +------------------- Bit 7 Port RC7: hier unbenutzt
122               +------------------- Bit 6 Port RC6: hier unbenutzt
123                +------------------ Bit 5 Port RC5: hier Eingang (/INT)
124                 +----------------- Bit 4 Port RC4: hier Ausgang (SDA)
125                  +---------------- Bit 3 Port RC3: hier Ausgang (SCL)
126                   +--------------- Bit 2 Port RC2: hier unbenutzt
127                    +-------------- Bit 1 Port RC1: hier unbenutzt
128                     +------------- Bit 0 Port RC0: hier unbenutzt
129 */
130 }
131
132
133 /****************** Hauptprogramm ************************************************/
134
135 /*****************************************************************************/
136 /* Aufgaben des Hauptprogramms:                                              */
137 /*   + Controller initialisieren (Unterprogramm Init)                        */
138 /*   + Takt fuer I2C-Bus festlegen (hier 100kHz mit dem Unterprogramm I2C_Init)  */
139 /*   + 8-Bit-Wert vom Eingabemodul (mit dem PCF8574) lesen und am I2C-PIC-Slave  */
140 /*       ausgeben                                                            */
141 /*   + Daten vom I2C-PIC-Slave lesen und am I2C-LED-Modul (mit einem weiteren PCF8574) */
142 /*     ausgeben                                                              */
143 /*   + Taetigkeiten/Unterprogramme, die alle zyklisch durchgefuehrt werden muessen:  */
144 /*      + bei jedem Interrupt des PCF8574(A) die Daten vom Eingabemodul (mit dem  */
145 /*        PCF8574) lesen und am I2C-PIC-Slave ausgeben. Anschliessend Daten vom I2C-  */
146 /*        PIC-Slave lesen und am LED-Ausgabemodul ausgeben.                   */
147 /*****************************************************************************/
148 void main(void)
149 {
150     char cTemp;
151
152     // Controller initialisieren
153     Init();
154
155     // I2C initialisieren
156     I2C_Init(100000);                           // I2C-Taktfrequenz: 100kHz
157
158     // Demo 1: 8-Bit-Wert vom I2C-Eingabemodul lesen ...
159     cTemp = I2C_PCF8574_ReadData(1);            // Moduladresse: hier 1
160     // ... und am I2C-PIC-Slave ausgeben
161     I2C_Start();                                // Startbedingung
162     I2C_Wr(0b00110000);                         // Bausteinadresse + Schreibzugriff
163     I2C_Wr(cTemp);                              // Daten an den PIC-Slave schreiben
164     I2C_Stop();                                 // Stopbedingung
165
166
167     // Demo 2: Daten vom I2C-PIC-Slave lesen ...
168     I2C_Start();                                // Startbedingung
169     I2C_Wr(0b00110001);                         // Bausteinadresse + Lesezugriff
170     cTemp = I2C_Rd(noACK);                      // Daten vom PIC-Slave lesen
171     I2C_Stop();                                 // Stopbedingung
172     // ... und am I2C-LED-Modul ausgeben
173     cTemp = ~cTemp;
174     I2C_PCF8574_WriteData(3, cTemp);            // Moduladresse: hier 3
175
176
177     // Endlosschleife
178     while(1)
179     {
```

```
180         // bei jedem Interrupt des PCF8574(A)
181          if (nINTPCF8574 == 0)
182          {
183              // Demo 1: 8-Bit-Wert vom I2C-Eingabemodul lesen ...
184              cTemp = I2C_PCF8574_ReadData(1);      // Moduladresse: hier 1
185              // ... und am I2C-PIC-Slave ausgeben
186              I2C_Start();                          // Startbedingung
187              I2C_Wr(0b00110000);                   // Bausteinadresse + Schreibzugriff
188              I2C_Wr(cTemp);                        // Daten an den PIC-Slave schreiben
189              I2C_Stop();                           // Stopbedingung
190
191
192              // Demo 2: Daten vom I2C-PIC-Slave lesen ...
193              I2C_Start();                          // Startbedingung
194              I2C_Wr(0b00110001);                   // Bausteinadresse + Lesezugriff
195              cTemp = I2C_Rd(noACK);                // Daten vom PIC-Slave lesen
196              I2C_Stop();                           // Stopbedingung
197
198              // ... und am I2C-LED-Modul ausgeben
199              cTemp = ~cTemp;
200              I2C_PCF8574_WriteData(3, cTemp);     // Moduladresse: hier 3
201          }
202      }
203  }
```

Listing 2: `I2C_Demo_PIC_Master.c` (Demo 1)

Datei `PCF8574.H`:

```
204  /*****************************************************************************/
205  /* Header fuer die Unterprogramme zur Ansteuerung des I/O-Port-Expander PCF8574 */
206  /*                                                                           */
207  /* Compiler: mikroC V8.2                                                     */
208  /*                                                                           */
209  /* Entwickler: Stefan Buchgeher                                              */
210  /* Entwicklungsbeginn der Software: 17. Februar 2013                         */
211  /* Funktionsfaehig seit: 17. Februar 2013                                    */
212  /* Letzte Bearbeitung: 17. Februar 2013                                      */
213  /*****************************************************************************/
214
215
216  /***************** Globale Register ****************************************/
217  /* keine globalen Register */
218
219
220  /***************** Portbelegung ********************************************/
221  /* keine Portdefinitionen */
222
223
224  /***************** Konstanten **********************************************/
225  /* keine Konstanten */
226
227
228  /***************** Funktionsprototypen *************************************/
229  /* Daten zum I/O-Port-Expander PCF8574 schreiben */
230  void I2C_PCF8574_WriteData(char cDeviceAdresse, char cWriteData);
231
232  /* Daten vom I/O-Port-Expander PCF8574 lesen */
233  unsigned char I2C_PCF8574_ReadData(char cDeviceAdresse);
```

Listing 3: `PCF8574.H`

Datei `PCF8574.C`:

```
234  /*****************************************************************************/
235  /* Unterprogramme zur Ansteuerung des I/O-Port-Expander PCF8574              */
236  /*                                                                           */
237  /* Compiler: mikroC V8.2                                                     */
```

```
238 /*                                                                              */
239 /* Entwickler: Stefan Buchgeher                                                 */
240 /* Entwicklungsbeginn der Software: 17. Februar 2013                            */
241 /* Funktionsfaehig seit: 17. Februar 2013                                       */
242 /* Letzte Bearbeitung: 17. Februar 2013                                         */
243 /*******************************************************************************/
244
245 /***************** Include-Dateien *********************************************/
246 #include "PCF8574.H"
247 #include "PCF8574A.H"
248
249 /***************** Strukturen  *************************************************/
250 /* keine Strukturen */
251
252
253 /***************** Globale Register ********************************************/
254 /* keine globalen Register */
255
256
257 /***************** Bits in den globale Register ********************************/
258 /* keine globalen Register */
259
260
261 /***************** Konstanten **************************************************/
262 #define noACK                   0
263 #define ACK                     1
264
265
266 /***************** Tabellen ****************************************************/
267 /* keine Tabellen */
268
269
270 /***************** Funktionsprototypen (fuer interne Hilfsunterprogramme) *************/
271 /* keine internen Hilfsunterprogramme */
272
273
274 /***************** Unterprogramme **********************************************/
275
276 /*******************************************************************************/
277 /* I2C_PCF8574_WriteData:                                                       */
278 /*                                                                              */
279 /* Aufgabe:                                                                     */
280 /*   Daten (8-Bit-Wert) an den PCF8574 schreiben                                */
281 /*                                                                              */
282 /* Uebergabeparameter:                                                          */
283 /*   cDeviceAdresse       Bausteinadresse gemaess den gesetzten Jumper A0 bis A2 */
284 /*   cWriteData           Daten (8-Bit-Wert), welche an den PCF8574 geschrieben werden */
285 /*                                                                              */
286 /* Rueckgabeparameter:                                                          */
287 /*   keiner                                                                     */
288 /*                                                                              */
289 /* Vorgehensweise:                                                              */
290 /*   Schritt 1: Die Device-Adressse ermitteln                                   */
291 /*   Schritt 2: Daten (8-Bit-Wert) an den PCF8574 schreiben                     */
292 /*******************************************************************************/
293 void I2C_PCF8574_WriteData(char cDeviceAdresse, char cWriteData)
294 {
295     char cHelpAdresse = cDeviceAdresse;
296
297
298     // Schritt 1: Device-Adresse zusammensetzen
299     cHelpAdresse = 0b00000111 & cHelpAdresse;
300     cHelpAdresse = cHelpAdresse << 1;
301     cHelpAdresse = 0b01000000 | cHelpAdresse;     // Bausteinadresse + Schreibzugriff
302
303     // Schritt 2: Daten (8-Bit-Wert) an den PCF8574 schreiben
304     I2C_Start();                // Startbedingung
305     I2C_Wr(cHelpAdresse);       // Bausteinadresse + Schreibzugriff
306     I2C_Wr(cWriteData);         // Daten an den PCF8574 schreiben
307     I2C_Stop();                 // Stopbedingung
308 }
```

```
309
310
311 /*****************************************************************************/
312 /* I2C_PCF8574_ReadData:                                                     */
313 /*                                                                           */
314 /* Aufgabe:                                                                  */
315 /*   Daten (8-Bit-Wert) vom PCF8574 lesen                                    */
316 /*                                                                           */
317 /* Uebergabeparameter:                                                       */
318 /*   cDeviceAdresse      Bausteinadresse gemaess den gesetzten Jumper A0 bis A2 */
319 /*                                                                           */
320 /* Rueckgabeparameter:                                                       */
321 /*   cReadData           Daten (8-Bit-Wert), welche vom PCF8574 empfangen wurden */
322 /*                                                                           */
323 /* Vorgehensweise:                                                           */
324 /*   Schritt 1: Die Device-Adressse ermitteln                                */
325 /*   Schritt 2: Daten (8-Bit-Wert) vom PCF8574 lesen                         */
326 /*   Schritt 3: Empfangenen Wert an das aufrufende Programm uebergeben       */
327 /*****************************************************************************/
328 unsigned char I2C_PCF8574_ReadData(char cDeviceAdresse)
329 {
330     char cHelpAdresse = cDeviceAdresse;
331     char cReadData;
332
333
334     // Schritt 1: Device-Adresse zusammensetzen
335     cHelpAdresse = 0b00000111 & cHelpAdresse;
336     cHelpAdresse = cHelpAdresse << 1;
337     cHelpAdresse = 0b01000001 | cHelpAdresse;     // Bausteinadresse + Lesezugriff
338
339     // Schritt 2: Daten (8-Bit-Wert) vom PCF8574 lesen
340     I2C_Start();                  // Startbedingung
341     I2C_Wr(cHelpAdresse);         // Bausteinadresse + Lesezugriff
342     cReadData = I2C_Rd(noACK);
343     I2C_Stop();                   // Stopbedingung
344
345     // Schritt 3: Empfangenen Wert an das aufrufende Programm uebergeben
346     return(cReadData);
347 }
```

Listing 4: `PCF8574.C`

## 5.5 Anmerkungen zur Software für 1. Demo (Master)

Auch dieser Quellcode wurde in C mit der freien Demoversion des mikroC-Compilers erstellt.

Die Software (für den PIC-I²C-Master) besteht im Wesentlichen aus den folgenden Programmteilen:

- Hauptprogramm am Ende des Quellcodes von `I2C_Demo_PIC_Master.c`, Zeilen 148 bis 203.
- 1 Unterprogramm zur Initialisierung des PIC16F887 (Unterprogramm `Init` in der Datei `I2C_Demo_PIC_Master.c`, Zeilen 81 bis 130).
- 2 Unterprogramme zur Kommunikation mit dem PCF8574 (`I2C_PCF8574_WriteData`, Zeilen 293 bis 308 und `I2C_PCF8574_ReadData`, Zeilen 330 bis 347 in der Datei `PCF8574.C`.

**Hauptprogramm:**
Zuerst muss der Mikrocontroller initialisiert werden. Diese Aufgabe übernehmen die Unterprogramme `Init()`, Zeile 153 und `I2C_Init()`, Zeile 156.

Danach werden zum ersten Mal die Portpins vom I²C-Eingabemodul mit dem PCF8574 (IC3) gelesen und am I²C-PIC-Slave ausgegeben. Das Lesen vom I²C-Eingabemodul geschieht mit dem Unterprogramm `I2C_PCF8574_ReadData` (Zeile 159). Übergabeparameter ist hier die Moduladresse (3-Bit entsprechend den Pins A2 bis A0). Hier der Wert 1, da A0 offen ist - also den Pegel High besitzt, während A1 und A2 via Jumper auf low-Pegel liegen (vgl. Schaltung in Abbildung 7, Seite 12).
Für das Ausgeben der Daten am I²C-PIC-Slave muss zunächst die I²C-Startbedingung erzeugt werden (Zeile 161) anschließend die Adresse des I²C-PIC-Slave inkl. Schreibzugriff (Zeile 162, hier `0b00110000` da die Jumper JP1, JP2, JP3 und JP6 gesetzt sind, siehe Abbildung 7 Seite 12. Das letzte niederwertigste Bit gibt die Zugriffsrichtung an. Hier 0 da es sich um einen Schreibzugriff aus Sicht des Masters handelt.) Anschließend werden die Daten an den I²C-PIC-Slave geschrieben (Zeile 163) und zum Schluss die Stopbedingung (Zeile 164).

Der umgekehrte Weg, also das Lesen von Daten vom I²C-PIC-Slave und Ausgeben am I²C-Ausgabemodul mit dem PCF8574 (IC4) ist ähnlich: Für das Lesen vom I²C-PIC-Slave muss zunächst (wieder) die I²C-Startbedingung erzeugt werden (Zeile 168) anschließend die Adresse des I²C-PIC-Slave hier inkl. Lesezugriff (Zeile 169). Das letzte niederwertigste Bit ist hier 1 da es sich um einen Lesezugriff aus Sicht des Masters handelt. Gefolgt vom Lesen der Daten vom I²C-PIC-Slave (Zeile 170) und zum Schluss wieder die Stopbedingung (Zeile 171). Der Übergabeparameter `noACK` bedeutet, dass keine weiteren Daten gelesen werden.
Das Schreiben der gelesenen Daten (diese befinden sich im Register `cTemp`) am I²C-Ausgabemodul erfolgt mit dem Unterprogramm `I2C_PCF8574_WriteData` (Zeile 174). Übergabeparameter sind hier die Moduladresse (auch hier 3-Bit, entsprechend den Pins A2 bis A0. Hier der Wert 3, da A0 und A1 offen sind - also den Pegel High besitzen, während A2 via Jumper auf low-Pegel liegen) und die zu schreibenden Daten. Beachte. Die Leuchtdioden beim Ausgabemodul sind gegen +5V geschaltet. D.h. damit eine Leuchtdiode leuchtet muss am Portpin Px low anliegen. Daher muss das Register `cTemp` invertiert werden bevor es am I²C-Ausgabemodul ausgegeben werden kann (Zeile 199).

Nun befindet sich das Hauptprogramm in einer Endlosschleife. Jedesmal wenn am I²C-Eingabemodul ein Portpin verändert wird erzeugt der PCF8574 einen so genannten Interrupt, indem der Pin 13 ($\overline{\texttt{INT}}$) von High nach Low übergeht. Dieser Interrupt-Pin ist sozusagen die fünfte Leitung des I²C-Buses und kann mit einem beliebigen I/O-Pin des Mikrocontrollers verbunden werden. Hier ist es der Portpin RC5 (vgl. Schaltbild nach Abbildung 7, Seite 12). In meiner Software erhält dieser Portpin die Bezeichnung `nINTPCF8574` (Zeile 47). Immer wenn dieser Portpin low ist, also wenn am I²C-Eingabemodul zumindest ein Portpin verändert wurde soll der geänderten Eingang gelesen und am PIC-I²C-Slave ausgeben werden (Zeilen 184 bis 189). Danach sollen die Daten vom PIC-I²C-Slave gelesen und am I²C-Ausgabemodul ausgeben werden (Zeilen 193 bis 200).

Hinweise:

- Der Interrupt-Pin ($\overline{\texttt{INT}}$, Pin 13) des PCF8574 geht in seinen Ruhepegel (High) über, nachdem sein Portzustand gelesen wurde.
- Die Zeilen 184 bis 189 sind die gleichen wie 161 bis 164, und die Zeilen 193 bis 200 sind gleich mit den Zeilen 168 bis 174.

**Beachte**:
Die Unterprogramme und Funktionen `I2c_Init()`, `I2c_Start()`, `I2c_Stop()`, `I2c_Wr()` und `I2c_Rd()` stellt der mikroC-Compiler zur Verfügung. Diese müssen hier **nicht** mittels `include`-Anweisung eingebunden werden. Diese Unterprogramme und Funktionen steuern dass SSP-Hardware-Modul im $I^2C$-Master-Mode.

**Unterprogramm `Init`:**
Das Unterprogramm `Init` (Datei: `I2C_Demo_PIC_Master.c`, Zeilen 81 bis 130) dient zur Initialisierung des Mikrocontrollers. Hier wird der interne Oszillator konfiguriert (also auf 4MHz eingestellt, Zeile 84) und die Pins des Port C für den $I^2C$-Bus als Ein- oder Ausgang definiert (Zeile 120). Die anderen Ports (A, B, D und E) werden hier nicht benötigt, und müssen daher auch nicht konfiguriert werden.

Im Gegensatz zum Slave muss hier das SSP-Hardware-Module für den Mastermode **nicht** initialisiert werden. Diese Aufgabe übernimmt das Unterprogramm `I2c_Init()`, welches vom microC-Compiler zur Verfügung gestellt wird.

**Unterprogramme `I2C_PCF8574_WriteData` und `I2C_PCF8574_ReadData`:**
Das Unterprogramm `I2C_PCF8574_WriteData` (Datei: `PCF8574.C`, Zeilen 293 bis 308) dient zum Schreiben eines 8-Bit-Wertes (Adresse oder Daten) zum PCF8574, während das Unterprogramm `I2C_PCF8574_ReadData` (Datei: `PCF8574.C`, Zeilen 228 bis 347) einen 8-Bit-Wert vom PCF8574 liest. Aufgrund der Einfachheit und der ausführlichen Kommentare ist hier eine weitere Erklärung glaube ich nicht notwendig.

**Sonstiges:**
Am Beginn des Quellcodes (in der Datei `I2C_Demo_PIC_Master.c`) befinden sich die folgenden Definitionen und Einstellungen:

- Einbinden externer Codeteile (hier die Unterprogramme für den PCF8574, Zeile 21, oder alternativ die Unterprogramme für den PCF8574A, Zeile 22)
- Portdefinition (hier für den $\overline{\texttt{INT}}$-Ausgang des PCF8574, Zeilen 47)
- Konstanten für $I^2C$ (Zeilen 52 und 53)
- Funktionsprototyp für das hier verwendete Unterprogramm (Zeile 62)
- Ab Zeile 81 beginnt der Quellcode der ausreichend mit Kommentaren versehen ist

Ein kleiner Nachteil beim mikroC-Compiler ist, dass die Konfigurationsbits für den PIC-Mikrocontroller in der IDE gesetzt werden müssen. Ich persönlich würde diese lieber im Quellcode mit einer geeigneten Anweisung setzen.

Hier sind die notwendigen Einstellungen für den PIC16F887 für dieses Demonstrationsbeispiel (für Master und Slave):

- ✓ _INTOSCIO
- ✓ _WDT_OFF
- ✓ _PWRTE_OFF
- ✓ _MCLRE_ON
- ✓ _CP_OFF
- ✓ _CPD_OFF
- ✓ _BOR_OFF
- ✓ _IESO_OFF
- ✓ _FCMEN_OFF
- ✓ _LVP_OFF
- ✓ _DEBUG_OFF
- ✓ _WRT_OFF
- ✓ _WRT_1FOURTH

## 5.6 Software für 2. Demo (Slave)

Beim zweiten Demonstrationsbeispiel sendet der I²C-Master **mehrere** Datenbytes (hier vier) zum PIC-I²C-Slave, bzw. liest anschließend auch **mehrere** Datenbytes (auch vier) vom PIC-I²C-Slave.

```
1  /*****************************************************************************/
2  /* Demo zur Ansteuerung des PIC16Fxx als I2C-Slave gemaess Application Note AN734      */
3  /*                                                                           */
4  /* Slave                                                                     */
5  /*                                                                           */
6  /* Demo 2: ein Byte vom Port A lesen und in einem Array speichern. Dieses Array mit    */
7  /*         weiteren beliebigen Werten auffuellen, und das gesamte Array via I2C an den */
8  /*         Slave senden. Ebenso ein Array via I2C empfangen und das zuletzt empfangene */
9  /*         Byte am Port D ausgeben.                                          */
10 /*                                                                           */
11 /* Mikrocontroller:       PIC16F887                                          */
12 /* Takt:                  interner Takt (4.0 MHz)                            */
13 /*                                                                           */
14 /* Compiler:              mikroC V8.2                                        */
15 /*                                                                           */
16 /* Entwickler:            Stefan Buchgeher                                   */
17 /* Entwicklungsbeginn der Software: 2. Maerz 2014                            */
18 /* Funktionsfaehig seit:  2. Maerz 2014                                      */
```

```
19 /* Letzte Bearbeitung:    4. Juli 2014                                          */
20 /*******************************************************************************/
21
22 /***************** Include-Dateien ********************************************/
23 /* keine Include-Dateien */
24
25
26 /***************** Strukturen *************************************************/
27 /* keine Strukturen verwendet */
28
29
30 /***************** Externe Register *******************************************/
31 /* keine externen Register */
32
33
34 /***************** Bits in den externen Registern *****************************/
35 /* keine externen Register */
36
37
38 /***************** Globale Register *******************************************/
39 char               cDummy;
40
41 char               cI2CReadBuffer[] = {0, 0, 0, 0};
42 char               cI2CWriteBuffer[] = {0, 0, 0, 0};
43 char               cI2CReadBufferIndex = 0;
44 char               cI2CWriteBufferIndex = 0;
45
46
47 /***************** Bits in den globalen Registern *****************************/
48 /* keine globalen Register */
49
50
51 /***************** Portbelegung ***********************************************/
52 /* \INT-Ausgang des PCF8574 */
53 //#define nINTPCF8574              PORTC.F5
54
55
56 /***************** Konstanten *************************************************/
57 /* Konstanten I2C-Schreib und Lesebuffer */
58 //#define           READBUFFERMAX          4
59 //#define           WRITEBUFFERMAX         4
60
61
62 /***************** Tabellen ***************************************************/
63 /* keine Tabellen */
64
65
66 /***************** Funktionsprototypen ****************************************/
67 /* Unterprogramm zur Initialisierung des Mikroccntrollers */
68 void Init(void);
69
70
71 /***************** Interrupt-Service-Routine (ISR) ****************************/
72
73 /*******************************************************************************/
74 /* Interrupt Service Routine:                                                  */
75 /*                                                                             */
76 /* Aufruf:                                                                     */
77 /*   SSP Interrupt Flag (SSPIF), nach jedem I2C-Event (wenn der Master eine Adresse    */
78 /*     auf den I2C-Bus schreibt oder wenn sich Daten am I2C-Bus befinden       */
79 /*                                                                             */
80 /* Aufgaben:                                                                   */
81 /*   + Je nach Status sind unterschiedliche Aktionen auszufuehren: siehe Kommentare im */
82 /*     Quellcode                                                               */
83 /*   + Nach jeder Aktion muss das SSP-Interrupt-Flag SSPIF wieder geloeschen werden    */
84 /*                                                                             */
85 /* Achtung:                                                                    */
86 /*   Der Status Lesen bzw. Schreiben erfolgt immer aus Sicht des Master!!      */
87 /*******************************************************************************/
88 void interrupt (void)                          // Interruptroutine
89 {
```

```
90      if (SSPSTAT.R_W == 0)
91      {
92          // Schreibzugriffe (aus Sicht des Masters, d.h. Slave muss Daten aus dem Sende-
93          // und Empfangsbuffer (SSPBUF) lesen)
94          if (SSPSTAT.D_A == 0)
95          {
96              // State 1: I2C write operation, last byte was an address byte
97              // SSPSTAT bits: S=1, D/A=0, R/W=0, BF=1
98              cDummy = SSPBUF;
99              SSPSTAT.BF = 0;
100             SSPCON.CKP = 1;                 // Clockstretching erlauben
101             cI2CWriteBufferIndex = 0;
102             PIR1.SSPIF = 0;                 // SSP-Interrupt-Flag wieder loeschen
103             return;
104         }
105         else
106         {
107             // State 2: I2C write operation, last byte was a data byte
108             // SSPSTAT bits: S=1, D/A=1, R/W=1, BF=1
109             cI2CWriteBuffer[cI2CWriteBufferIndex] = SSPBUF; // Datenbyte von SSPBUF
110                                             // lesen und ins Array schreiben
111             cI2CWriteBufferIndex = cI2CWriteBufferIndex + 1;
112             SSPCON.CKP = 1;                 // Clockstretching erlauben
113             PIR1.SSPIF = 0;                 // SSP-Interrupt-Flag wieder loeschen
114             return;
115         }
116     }
117     else
118     {
119         // Lesezugriff (aus Sicht des Masters, d.h. Slave muss Daten in Sende- und
120         // Empfangsbuffer (SSPBUF) schreiben)
121         if (SSPSTAT.D_A == 0)
122         {
123             // State 3: I2C read operation, last byte was an address byte
124             // SSPSTAT bits: S=1, D/A=0, R/W=1, BF=0
125             SSPBUF = cI2CReadBuffer[0];     // Datenbyte von Array lesen und in SSPBUF
126             cI2CReadBufferIndex = 0;
127             SSPCON.CKP = 1;                 // Clockstretching erlauben
128             PIR1.SSPIF = 0;                 // SSP-Interrupt-Flag wieder loeschen
129             return;
130         }
131         else
132         {
133             // State 4: I2C read operation, last byte was a data byte
134             // SSPSTAT bits: S=1, D/A=1, R/W=0, BF=0
135             cI2CReadBufferIndex = cI2CReadBufferIndex + 1;
136             SSPBUF = cI2CReadBuffer[cI2CReadBufferIndex];      // Datenbyte von Array
137                                             // lesen und in SSPBUF
138             SSPCON.CKP = 1;                 // Clockstretching erlauben
139             PIR1.SSPIF = 0;                 // SSP-Interrupt-Flag wieder loeschen
140             return;
141         }
142     }
143 }
144
145
146 /***************** Unterprogramme und Funktionen ************************************/
147
148 /*******************************************************************************/
149 /* Init:                                                                       */
150 /*                                                                             */
151 /* Aufgabe:                                                                    */
152 /*   Initialisierung des Prozessor:                                            */
153 /*      + internen Oszillator konfigurieren                                    */
154 /*      + Ports auf Digital I/O umschalten und als Ein- oder Ausgang konfigurieren */
155 /*      + I2C-Hardware-Modul als Slave konfigurieren                           */
156 /*      + I2C Slave Adresse von Port B lesen und in SSPADD schreiben           */
157 /*      + Interrupt freigeben                                                  */
158 /*                                                                             */
159 /* Uebergabeparameter:                                                         */
160 /*   keiner                                                                    */
```

```
161 /*                                                                              */
162 /* Rueckgabeparameter:                                                          */
163 /*   keiner                                                                     */
164 /******************************************************************************/
165 void Init(void)
166 {
167     char cI2CAdresse;                   // I2C-Slave-Adresse
168
169
170     // (internen) Oszillator konfigurieren
171     OSCCON = 0b01100001;                // Sync Oscillator Control Register
172 /*              +--------------------- Bit 7 Reserve
173                  +++------------------- Bit 6-4 (IRCF2:IRCF0): Internal Oscillator Frequency
174                     ||||                    Select bits
175                     ||||                 000 : 31 kHz (LFINTOSC)
176                     ||||                 001 : 125 kHz
177                     ||||                 010 : 250 kHz
178                     ||||                 011 : 500 kHz
179                     ||||                 100 : 1 MHz
180                     ||||                 101 : 2 MHz
181                     ||||              -> 110 : 4 MHz (default)
182                     ||||                 111 : 8 MHz
183                    +--------------------- Bit 3 (OSTS): Oscillator Start-up Time-out Status
184                     |||                    (Read Only)
185                     |||                  0 : Device is running from the clock defined
186                     |||                     by FOSC2:FOSC0 of the CONFIG1 register
187                     |||                  1 : Device is running from the internal
188                     |||                     oscillator (HFINTOSC or LFINTOSC)
189                     +---------------------- Bit 2 (HTS): HFINTOSC Status bir
190                      ||                    (High Frequency - 125kHz to 8MHz)
191                      ||                    (Read Only)
192                      ||                  0 : HFINTOSC is not stable
193                      ||                  1 : HFINTOSC is stable
194                      +---------------------- Bit 1 (LTS): LFINTOSC Status bit
195                       |                    (Low Frequency - 31kHz)
196                       |                    (Read Only)
197                       |                  0 : LFINTOSC is not stable
198                       |                  1 : LFINTOSC is stable
199                       +---------------------- Bit 0 (SCS): System Clock Select bit
200                                            0 : Clock source defined by FOSC2:FOSC0
201                                               of the CONFIG1 register
202                                         ->  1 : Internal oscillator is used for system
                                                clock
203 */
204
205     // Alle Ports auf digital I/O konfigurieren
206     ANSEL = 0b00000000;                 // Analog Select Register
207                 ||||||||                //  (0: Digital I/O, 1: Analogeingang)
208 /*              +--------------------- Bit 7: Port RE2/AN7 ist hier Digital I/O
209                  +-------------------- Bit 6: Port RE1/AN6 ist hier Digital I/O
210                   +------------------- Bit 5: Port RE0/AN5 ist hier Digital I/O
211                    +------------------ Bit 4: Port RA5/AN4 ist hier Digital I/O
212                     +----------------- Bit 3: Port RA3/AN3 ist hier Digital I/O
213                      +---------------- Bit 2: Port RA2/AN2 ist hier Digital I/O
214                       +--------------- Bit 1: Port RA1/AN1 ist hier Digital I/O
215                        +-------------- Bit 0: Port RA0/AN0 ist hier Digital I/O
216 */
217
218     ANSELH = 0b00000000;                // Analog Select Register
219                 ||||||||                //  (0: Digital I/O, 1: Analogeingang)
220 /*              ++-------------------- Bit 6,7 Reserve
221                   +------------------- Bit 5: Port RB5/AN13 ist hier Digital I/O
222                    +------------------ Bit 4: Port RB0/AN12 ist hier Digital I/O
223                     +----------------- Bit 3: Port RB4/AN11 ist hier Digital I/O
224                      +---------------- Bit 2: Port RB1/AN10 ist hier Digital I/O
225                       +--------------- Bit 1: Port RB9/AN9 ist hier Digital I/O
226                        +-------------- Bit 0: Port RB2/AN8 ist hier Digital I/O
227 */
228
229     // Ports konfigurieren
230     TRISA = 0b11111111;                 // Richtungsregister Port A (0: Ausgang, 1: Eingang)
```

```
231 /*              +------------------------ Bit 7 Port RA7: hier Eingang (Jumper Pin 7)
232                  +----------------------- Bit 6 Port RA6: hier Eingang (Jumper Pin 6)
233                   +---------------------- Bit 5 Port RA5: hier Eingang (Jumper Pin 5)
234                    +--------------------- Bit 4 Port RA4: hier Eingang (Jumper Pin 4)
235                     +-------------------- Bit 3 Port RA3: hier Eingang (Jumper Pin 3)
236                      +------------------- Bit 2 Port RA2: hier Eingang (Jumper Pin 2)
237                       +------------------ Bit 1 Port RA1: hier Eingang (Jumper Pin 1)
238                        +----------------- Bit 0 Port RA0: hier Eingang (Jumper Pin 0)
239 */
240
241     TRISB = 0b00111111;             // Richtungsregister Port B (0: Ausgang, 1: Eingang)
242 /*              +------------------------ Bit 7 Port RB7: hier unbenutzt
243                  +----------------------- Bit 6 Port RB6: hier unbenutzt
244                   +---------------------- Bit 5 Port RB5: hier Eingang (I2C-Slave-Adresse A5)
245                    +--------------------- Bit 4 Port RB4: hier Eingang (I2C-Slave-Adresse A4)
246                     +-------------------- Bit 3 Port RB3: hier Eingang (I2C-Slave-Adresse A3)
247                      +------------------- Bit 2 Port RB2: hier Eingang (I2C-Slave-Adresse A2)
248                       +------------------ Bit 1 Port RB1: hier Eingang (I2C-Slave-Adresse A1)
249                        +----------------- Bit 0 Port RB0: hier Eingang (I2C-Slave-Adresse A0)
250 */
251
252     TRISC = 0b00011000;             // Richtungsregister Port C (0: Ausgang, 1: Eingang)
253 /*              +------------------------ Bit 7 Port RC7: hier unbenutzt
254                  +----------------------- Bit 6 Port RC6: hier unbenutzt
255                   +---------------------- Bit 5 Port RC5: hier unbenutzt
256                    +--------------------- Bit 4 Port RC4: hier Eingang (SDA)
257                     +-------------------- Bit 3 Port RC3: hier Eingang (SCL)
258                      +------------------- Bit 2 Port RC2: hier unbenutzt
259                       +------------------ Bit 1 Port RC1: hier unbenutzt
260                        +----------------- Bit 0 Port RC0: hier unbenutzt
261 */
262
263     TRISD = 0b00000000;             // Richtungsregister Port D (0: Ausgang, 1: Eingang)
264 /*              +------------------------ Bit 7 Port RD7: hier Ausgang (LED D7)
265                  +----------------------- Bit 6 Port RD6: hier Ausgang (LED D6)
266                   +---------------------- Bit 5 Port RD5: hier Ausgang (LED D5)
267                    +--------------------- Bit 4 Port RD4: hier Ausgang (LED D4)
268                     +-------------------- Bit 3 Port RD3: hier Ausgang (LED D3)
269                      +------------------- Bit 2 Port RD2: hier Ausgang (LED D2)
270                       +------------------ Bit 1 Port RD1: hier Ausgang (LED D1)
271                        +----------------- Bit 0 Port RD0: hier Ausgang (LED D0)
272 */
273
274     TRISE = 0b00000000;             // Richtungsregister Port E (0: Ausgang, 1: Eingang)
275               ||||||||              //    und Parallel Slave Port Status/Control Bits
276 /*              +------------------------ Bit 7 (IBF): Input Buffer Full Status bit
277                  |||||||                     (Read only)
278                  |||||||                   0 : No word has been received
279                  |||||||                   1 : A word has been received and is waiting to
280                  |||||||                       be read by the CPU
281                  +------------------------ Bit 6 (OBF): Output Buffer Full Status bit
282                   ||||||                     (Read only)
283                   ||||||                   0 : The output buffer has been read
284                   ||||||                   1 : The output buffer still holds a previously
285                   ||||||                       written word
286                   +------------------------ Bit 5 (IBOV): Input Buffer Overflow Detect bit
287                    |||||                      (in Microprocessor mode)
288                    |||||                   0 : No overflow occurred
289                    |||||                   1 : A write occurred when a previously input
290                    |||||                       word has not been read (must be cleared
291                    |||||                       in Software)
292                    +------------------------ Bit 4 (PSPMODE) : Parallel Slave Port Mode Select
293                     ||||                   -> 0 : PORTD functions in general Purpose I/O mode
294                     ||||                      1 : PORTD functions in Parallel Slave Port mode
295                     +------------------------ Bit 3 Reserve
296                      +----------------------- Bit 2 Port RE2: hier unbenutzt
297                       +---------------------- Bit 1 Port RE1: hier unbenutzt
298                        +--------------------- Bit 0 Port RE0: hier unbenutzt
299 */
300
301     PORTA = 0;
```

```
302     PORTB = 0;
303     PORTC = 0;
304     PORTD = 0;
305     PORTE = 0;
306
307
308     // I2C-Hardware-Modul als Slave konfigurieren
309     SSPCON = 0b00100110;              // Sync Serial Port Control Register 1
310 /*              +-------------------- Bit 7 (WCOL): Write Collision Detect bit
311                  |||||||                   Master mode:
312                  |||||||                   0 : No collision
313                  |||||||                   1 : A write to SSPBUF was attempt while the I2C
314                  |||||||                       conditions were not valis
315                  |||||||                   Slave mode:
316                  |||||||                   0 : No collision
317                  |||||||                   1 : SSPBUF register is written while still
318                  |||||||                       transmitting the previous word (must be
319                  |||||||                       cleared in software)
320                  +------------------- Bit 6 (SSPOV): Receive Overflow Indicator bit
321                   ||||||                   0 : No overflow
322                   ||||||                   1 : A new byte is received while SSPBUF holds
323                   ||||||                       previous data (weiter siehe Datenblatt!!)
324                   ||||||                       (must be cleared in software)
325                   +------------------- Bit 5 (SSPEN): Synchronous Serial Port Enable bit
326                    |||||                   0 : Disables serial port and configures these
327                    |||||                       Pins as I/O port pins
328                    |||||                ->   1 : Enables the serial port an configures
329                    |||||                     SCK, SDO, SDI, SS / SDA, SCL pins as the
330                    |||||                     source of the serial port pins
331                    +-------------------- Bit 4 (CKP): Clock Polarity Select bit
332                     ||||                   In SPI mode:
333                     ||||                   0 : Idle state for clock is a low level
334                     ||||                   1 : Idle state for clock is a high level
335                     ||||                   In I2C Slave mode: SCK release control
336                     ||||                ->   0 : Holds clock low (clock stretch). (Used to
337                     ||||                       ensure data setup time.)
338                     ||||                   1 : Enable clock
339                     ||||                   In I2C Master mode: Unused in this mode
340                     ++++------------------ Bit 3-0 (SSPM3:SSPM0): SSP Mode Select bits
341                                               0000 : SPI Master mode, clock = Fosc/4
342                                               0001 : SPI Master mode, clock = Fosc/16
343                                               0010 : SPI Master mode, clock = Fosc/64
344                                               0011 : SPI Master mode, clock = TMR2 output/2
345                                               0100 : SPI Slave mode, clock = SCK pin, nSS pin
346                                                   control enabled
347                                               0101 : SPI Slave mode, clock = SCK pin, nSS pin
348                                                   control disabled. nSS can be used as I/O
349                                            -> 0110 : I2C Slave mode, 7-bit address
350                                               0111 : I2C Slave mode, 10-bit address
351                                               1000 : I2C Master mode,
352                                                   clock = Fosc/(4*(SSPADD+1))
353                                               1001 : Reserved
354                                               1010 : Reserved
355                                               1011 : I2C Firmware Controlled Master mode
356                                                   (slave idle)
357                                               1100 : Reserved
358                                               1101 : Reserved
359                                               1110 : I2C Firmware Controlled Master mode,
360                                                   7-bit address with START and STOP bit
361                                                   interrupts enabled
362                                               1111 : I2C Firmware Controlled Master mode,
363                                                   10-bit address with START and STOP bit
364                                                   interrupts enabled
365 */
366
367     SSPCON2 = 0b00110110;             // Sync Serial Port Control Register 2
368 /*              +-------------------- Bit 7 (GCEN): General Call Enable bit (In I2C Slave
369                  |||||||                   mode only)
370                  |||||||               ->   0 : General Call address disabled
371                  |||||||                   1 : Enable interrupt when a general call
372                  |||||||                       address (0000h) ist received in the SSPSR
```

```
373                 +---------------------- Bit 6 (ACKSTAT) : Acknowledge Status bit (In I2C
374                  ||||||                      Master mode only)
375                  ||||||                      0 : Acknowledge was received from slave
376                  ||||||                      1 : Acknowledge was not received from slave
377                  +---------------------- Bit 5 (ACKDT): Acknowledge Data bit (In I2C Master
378                   |||||                      mode only):
379                   |||||                      Value that will be transmitted when the user
380                   |||||                      iniziates an Acknowledge sequence at the end
381                   |||||                      of a receive
382                   |||||                      0 : Acknowledge
383                   |||||                      1 : Not Acknowledge
384                   +---------------------- Bit 4 (ACKEN): Acknowledge Sequence Enable bit
385                    ||||                       (In I2C Master mode only):
386                    ||||                       0 : Acknowledge sequence idle
387                    ||||                       1 : Initiate Acknowledge sequence on SDA and
388                    ||||                           SCL pins and transmit ACKDT data bit.
389                    ||||                           Automatically cleared by hardware
390                    +---------------------- Bit 3 (RCEN): Receive Enable bit (In I2C Master
391                     |||                       mode only)
392                     |||                       0 : Receive idle
393                     |||                       1 : Enables Receive mode for I2C
394                     +---------------------- Bit 2 (PEN): STOP Condition Enable bit (In I2C
395                      ||                        Master mode only)
396                      ||                        0 : STOP condition idle
397                      ||                        1 : Initiate STOP condition on SDA and SCL pins
398                      ||                            Automatically cleared by hardware
399                      +---------------------- Bit 1 (RSEN): Repeated START Condition Enable bit
400                       |                        (In I2C Master mode only)
401                       |                        0 : Repeated START condition idle
402                       |                        1 : Initiate Repeated START condition on SDA
403                       |                            and SCL pins
404                       |                            Automatically cleared by hardware
405                       +---------------------- Bit 0 (SEN): START Condition Enable bit (In I2C
406                                                    Master mode only)
407                                                    0 : START condition idle
408                                                    1 : Initiate START cond. on SDA and SCL pins
409                                                       Automatically cleared by hardware
410 */
411
412     SSPSTAT = 0b10000000;
413
414     // I2C Slave Adresse von Port B lesen und in SSPADD schreiben
415     // Hier 0 A5 A4 A3 A2 A1 A0 R/nW
416     cI2CAdresse = 0b00111111;              // Maske
417     cI2CAdresse = cI2CAdresse & PORTB;
418     cI2CAdresse = cI2CAdresse << 1;
419     cI2CAdresse.F0 = 0;                    // Schreibzugriff aus Sicht des Master
420     SSPADD = cI2CAdresse;
421
422
423     // Diverse Register initialisieren
424
425
426     // Interrupt freigeben
427     PIE1 = 0b00001000;                     // Peripheral Interrupt Enable Register 1 (Bank 1)
428 /*          +---------------------- Bit 7  Reserve
429              +---------------------- Bit 6 (ADIE): A/D Converter Interrupt Enable bit
430               ||||||                 ->   0 : Disables the ADC interrupt
431               ||||||                      1 : Enables the ADC interrupt
432               +---------------------- Bit 5 (RCIE): EUSART Receive Interrupt Enable bit
433                |||||                 ->   0 : Disables the EUSART Receive Interrupt
434                |||||                      1 : Enables the EUSART Receive Interrupt
435                +---------------------- Bit 4 (TXIE): EUSART Transmit Interrupt Enable bit
436                 ||||                  ->   0 : Disables the Transmit Interrupt Enable bit
437                 ||||                       1 : Enables the Transmit Interrupt Enable bit
438                 +---------------------- Bit 3 (SSPIE): Master Synchronous Serial Port (MSSP)
439                  |||                                    Interrupt Enable bit
440                  |||                       0 : Disables the MSSP interrupt
441                  |||                  ->   1 : Enables the MSSP interrupt
442                  +---------------------- Bit 2 (CCP1IF): CCP1 Interrupt Enable bit
443                   ||                  ->   0 : Disables the CCP1 interrupt
```

```
444                     ||                      1 : Enables the CCP1 interrupt
445                     +------------------ Bit 1 (TMR2IE): Timer 2 to PR2 Match Interrupt
                            Enable bit
446                      |                  ->  0 : Disables the Timer 2 to PR2 Match Interrupt
447                      |                      1 : Enables the Timer 2 to PR2 Match Interrupt
448                      +------------------ Bit 0 (TMR1IE): Timer 1 Overflow Interrupt Enable
                             bit
449                                         ->  0 : Disables Timer 1 Overflow Interrupt
450                                             1 : Enables Timer 1 Overflow Interrupt
451 */
452
453     INTCON = 0b11000000;          // Interrupt-Control-Register (Bank 1)
454 /*              +------------------------ Bit 7 (GIE): Global Interrupt Enable bit
455                  |||||||                  0 : Disables all interrupts
456                  |||||||                ->  1 : Enables all unmasked interrupts
457                  +------------------------ Bit 6 (PEIE): Peripheral Interrupt Enable bit
458                   ||||||                  0 : Disables all peripheral interrupts
459                   ||||||                ->  1 : Enables all unmasked peripheral interrupts
460                   +----------------------- Bit 5 (T0IE): Timer 0 Overflow Interrupt Enable Bit
461                    |||||                ->  0 : Disabled
462                    |||||                    1 : Enabled
463                    +----------------------- Bit 4 (INTE): RB0/INT External Interrupt Enable Bit
464                     ||||                ->  0 : Disabled
465                     ||||                    1 : Enabled
466                     +----------------------- Bit 3 (RPIE): RB Port Change Interrupt Enable Bit
467                      |||                ->  0 : Disabled
468                      |||                    1 : Enabled
469                      +----------------------- Bit 2 (T0IF): Timer 0 Overflow Interrupt Flag bit
470                       ||                ->  0 : TMR0 register did not overflow
471                       ||                    1 : TMR0 register has overflowed
472                       +----------------------- Bit 1 (INTF): RB0/INT External Interrupt Flag bit
473                        |                ->  0 : RB0/INT external interrupt did not occur
474                        |                    1 : RB0/INT external interrupt occurred
475                        +----------------------- Bit 0 (RBIF): RB Port Change Interrupt Flag bit
476                                         ->  0 : None of RB7:RB4 pins have changed state
477                                             1 : One of the RB7:RB4 pins changed state
478 */
479 }
480
481
482 /****************** Hauptprogramm ************************************************/
483
484 /*********************************************************************************/
485 /* Aufgaben des Hauptprogramms:                                                  */
486 /*   + Controller initialisieren (Unterprogramm Init)                            */
487 /*   + Aufgaben in der Endlosschleife:                                           */
488 /*       + Daten vom emfangenen Array am Port D ausggeben (hier nur das letzte Daten- */
489 /*         byte                                                                  */
490 /*       + Daten vom Port A einlesen und in einem Demo-Array speichern (dient hier */
491 /*         nur zur Demonstration!)                                               */
492 /*   + Alles andere erfolgt in der ISR!!                                         */
493 /*********************************************************************************/
494 void main(void)
495 {
496     // Controller initialisieren
497     Init();
498
499
500     // Endlosschleife
501     while (1)
502     {
503         PORTD = cI2CWriteBuffer[3];
504
505         cI2CReadBuffer[0] = PORTA;
506         cI2CReadBuffer[1] = 0b00111100;
507         cI2CReadBuffer[2] = 0b01010011;
508         cI2CReadBuffer[3] = PORTA;
509     }
510 }
```

Listing 5: `I2C_Demo_PIC_Slave.c` (Demo 2)

## 5.7 Anmerkungen zur Software für 2. Demo (Slave)

Auch bei diesem zweiten Demonstrationsbeispiel wurde der Quellcode in C mit der freien Demoversion des mikroC-Compilers erstellt, und die Software für den PIC-I²C-Slave besteht aus den gleichen, aber angepassten, Programmteilen wie beim ersten Demonstrationsbeispiel. Nämlich aus:

- kurzes Hauptprogramm am Ende des Quellcodes, Zeilen 494 bis 510.
- 1 Unterprogramm zur Initialisierung des PIC16F887 (Unterprogramm `Init`, Zeilen 165 bis 479).
- kurze Interrupt-Service-Routine (kurz: ISR), Zeilen 88 bis 143.

Wichtig zu erwähnen ist, dass es bei diesem zweiten Demonstrationsbeispiel zweckmäßig ist die empfangenen Daten und die zu sendenden Daten in je einem Array zu speichern (hier: `cI2CReadBuffer[]` und `cI2CWriteBuffer[]`).

**Hauptprogramm:**
Die Aufgabe des Hauptprogramms (Zeilen 494 bis 510) ist hier zunächst das Initialisieren des Mikrocontrollers. Dies erfolgt mit dem Unterprogramm `Init` (Zeile 497). Das Bereitstellen der zu übertragenden Daten und die Weiterverarbeitung der empfangenen Daten erfolgt in der Endlosschleife. Bei diesem Demonstrationsbeispiel soll nur das vierte empfange Byte am Port D ausgegeben werden (Zeile 503), während der Zustand von Port A zweimal zum I²C-Master übertragen werden soll. Das zweite und dritte Datenbyte sind hier beliebige konstante Werte (Zeilen 505 bis 508).
Alles andere erfolgt in der Interrupt-Service-Routine (ISR).

Beachte:
Die Bezeichnungen der Variablen `cI2CReadBuffer[]` und `cI2CWriteBuffer[]`) erfolgen aus Sicht des Masters!

**Unterprogramm `Init`:**
Das Unterprogramm `Init` (Zeilen 165 bis 479) dient auch hier zur Initialisierung des Mikrocontrollers. Es ist gleich wie beim ersten Demonstrationsbeispiel (siehe Abschnitt 5.3 ab Seite 20).

**Interrupt-Service-Routine (kurz: ISR):**
Die SSP-ISR (Zeilen 88 bis 143) wird auch hier jedesmal aufgerufen wenn ein gesamtes Datenbyte vom SSP-Hardware-Modul empfangen wurde und dieses für den PIC-Slave mit der Adresse im Register `SSPADD` bestimmt ist. Jedes mal wenn diese SSP-ISR aufgerufen wird muss eine der folgenden vier Fälle ausgeführt werden (vgl. Flussdiagramm in Abbildung 5, Seite 8). Im Gegensatz zum ersten Demonstrationsbeispiel müssen hier auch die Lese- und Schreibezeiger richtig behandelt werden, da hier mehr als ein Datenbyte gelesen bzw. geschrieben (gesendet) werden).

Fall 1: Schreibzugriff aus Sicht des Masters, zuletzt wurde die Slave-Adresse empfangen (Bits: `SSPSTAT.R_W = 0, SSPSTAT.D_A = 0`):
In diesem Fall **muss** der Mikrocontroller (Slave) die empfangenen Daten aus dem SSP-Buffer (Register `SSPBUF`) lesen kann den Inhalt von `SSPBUF` aber verwerfen, da es nur die Adresse und keine Daten enthält (Zeile 98). Hier wird der Inhalt von `SSPBUF` in ein Dummy-Register geschrieben. Wichtig ist das dass Register `SSPBUF` gelesen wird, weil dadurch Steuerbits für das Hardwaremodul verändert werden. Neu ist bei diesem Demonstrationsbeispiel das Initialisieren des Schreib-Zeiger `cI2CWriteBufferIndex` (aus Sicht des Masters, Zeile 101). Wichtig sind auch hier die folgenden Codezeilen 99 und 100. Das SSP-Interrupt-Flag `SSPIF` muss ebenfalls gelöscht werden, da sonst dieser Interrupt sofort wieder ausgelöst wird (Zeile 102)!

Fall 2: Schreibzugriff aus Sicht des Masters, zuletzt wurde ein Datenbyte empfangen (Bits: `SSPSTAT.R_W = 0, SSPSTAT.D_A = 1`):
Jetzt handelt es sich tatsächlich um Nutzdaten. Diese befinden sich im SSP-Buffer-Register `SSPBUF`. Daher den Inhalt dieses Registers in das eigene Array `cI2CWriteBuffer[]` an der Stelle auf den der Index-Zeiger `cI2CWriteBufferIndex` zeigt schreiben (Zeile 109). Diesen Index-Zeiger anschließend um eins erhöhen (Zeile 111), damit beim nächsten Aufruf dieser ISR der nächste empfangene Wert an die nächste Stelle im Array `cI2CWriteBuffer[]` geschrieben wird.
Auch hier sind die folgenden Codezeilen 112 bis 114 wichtig und dürfen nicht weggelassen werden!

Fall 3: Lesezugriff aus Sicht des Masters, zuletzt wurde die Slave-Adresse empfangen (Bits: `SSPSTAT.R_W = 1, SSPSTAT.D_A = 0`):
In diesem Fall muss das erste zu sendende Datenbyte in das Register `SSPBUF` geschrieben werden. Daher das erste Datenbyte (mit dem Index 0) vom eigene Array `cI2CReadBuffer[]` lesen und in das Register `SSPBUF` schreiben (Zeile 125). Als nächstes den Lese-Zeiger (aus Sicht des Masters) `cI2CReadBufferIndex` initialisieren (Zeile 126).
Auch hier sind die folgenden Codezeilen 127 bis 129 wichtig und dürfen nicht weggelassen werden!

Fall 4: Lesezugriff aus Sicht des Masters, zuletzt wurde ein Datenbyte gesendet (Bits: `SSPSTAT.R_W = 1, SSPSTAT.D_A = 1`):
Im Gegensatz zum ersten Demonstrationsbeispiel tritt hier auch dieser Fall auf. Hier muss zunächst der Lese-Index-Zeiger `cI2CReadBufferIndex` um eins erhöht werden (Zeile 135). Anschließend den Inhalt des Lesearray, auf den der Lesezeiger zeigt lesen und in das Register `SSPBUF` schreiben (Zeile 136).
Auch hier sind die folgenden Codezeilen 138 bis 140 wichtig und dürfen nicht weggelassen werden!

Mehr als das Lesen des `SSPBUF`-Registers oder das Schreiben in das `SSPBUF`-Register ist auch bei diesem zweiten Demonstrationsbeispiel in der Interrupt-Service-Routine (ISR) nicht zu tun. Alles andere übernimmt auch hier das Hardware-Modul SSP des Mikrocontrollers.

**Sonstiges:**
Am Beginn des Quellcodes befinden sich die folgenden Definitionen und Einstellungen:

- Definition der globalen Register (hier `cDummy`, Zeile 37, sowie die Arrays zum speichern der gesendeten und empfangenen Daten (Zeilen 41 und 42) und die Index-Zeiger (Zeilen 43 und 44)
- Funktionsprototyp für das hier verwendete Unterprogramm (`Init`, Zeile 68)
- Ab Zeile 88 beginnt der Quellcode der ausreichend mit Kommentaren versehen ist

Die Konfigurationsbits für den PIC16F887-Mikrocontroller sind hier gleich wie beim ersten Demonstrationsbeispiel.

## 5.8 Software für 2. Demo (Master)

Natürlich müssen auch beim Master Änderungen vorgenommen werden.

Auch bei diesem zweiten Demonstrationsbeispiel ist der Quellcode für den $I^2C$-Bus-Master (mit dem PIC16F887, IC2) zur besseren Übersicht in mehrere Dateien aufgeteilt:

- `I2C_Demo_PIC_Master.c` beinhaltet das Hauptprogramm und das Unterprogramm zur Initialisierung des Mikrocontrollers (PIC16F887).
- `PCF8574.C` beinhaltet die Unterprogramme zur Datenübertragung mit dem PCF8574.
- `PCF8574.H` ist die Headerdatei zu `PCF8574.C`

Die Dateien `PCF8574.C` und `PCF8574.H` unterscheiden sich nicht zum ersten Demonstrationsbeispiel und werden hier daher auch nicht mehr behandelt.

Hier die Datei `I2C_Demo_PIC_Master.c` (für das zweite Demonstrationsbeispiel):

```
1  /*****************************************************************************/
2  /* Demo zur Ansteuerung des PIC16Fxx als I2C-Slave gemaess Application Note AN734      */
3  /*                                                                           */
4  /* Master                                                                    */
5  /*                                                                           */
6  /* Demo 2: ein Byte vom Port B lesen und in einem Array speichern. Diese Array mit     */
7  /*         weiteren beliebigen Werten auffuellen, und da gesamte Array via I2C an den  */
8  /*         Slave senden. Ebenso ein Array via I2C empfangen und das zuletzt empfangene */
9  /*         Byte am Port D ausgeben.                                          */
10 /*                                                                           */
11 /* Mikrocontroller:       PIC16F887                                          */
12 /* Takt:                  interner Takt (4.0 MHz)                            */
13 /*                                                                           */
14 /* Compiler:              mikroC V8.2                                        */
15 /*                                                                           */
16 /* Entwickler:            Stefan Buchgeher                                   */
17 /* Entwicklungsbeginn der Software: 2. Maerz 2014                            */
18 /* Funktionsfaehig seit:  2. Maerz 2014                                      */
19 /* Letzte Bearbeitung:    4. Juli 2014                                       */
20 /*****************************************************************************/
21
22 /***************** Include-Dateien *******************************************/
```

```
23 #include "PCF8574.H"
24 #include "PCF8574A.H"
25
26
27 /***************** Strukturen ************************************************/
28 /* keine Strukturen verwendet */
29
30
31 /***************** Externe Register ******************************************/
32 /* keine externen Register */
33
34
35 /***************** Bits in den externen Registern ****************************/
36 /* keine externen Register */
37
38
39 /***************** Globale Register ******************************************/
40 /* keine globalen Register */
41
42
43 /***************** Bits in den globalen Registern ****************************/
44 /* keine globalen Register */
45
46
47 /***************** Portbelegung **********************************************/
48 /* \INT-Ausgang des PCF8574 */
49 #define nINTPCF8574                PORTC.F5
50
51
52 /***************** Konstanten ************************************************/
53 /* Konstanten fuer I2C */
54 #define noACK                      0
55 #define ACK                        1
56
57
58 /***************** Tabellen **************************************************/
59 /* keine Tabellen */
60
61
62 /***************** Funktionsprototypen ***************************************/
63 /* Unterprogramm zur Initialisierung des Mikroccntrollers */
64 void Init(void);
65
66
67 /***************** Unterprogramme und Funktionen *****************************/
68
69 /*****************************************************************************/
70 /* Init:                                                                     */
71 /*                                                                           */
72 /* Aufgabe:                                                                  */
73 /*   Initialisierung des Prozessor:                                          */
74 /*      + internen Oszillator konfigurieren                                  */
75 /*      + Ports konfigurieren                                                */
76 /*                                                                           */
77 /* Uebergabeparameter:                                                       */
78 /*   keiner                                                                  */
79 /*                                                                           */
80 /* Rueckgabeparameter:                                                       */
81 /*   keiner                                                                  */
82 /*****************************************************************************/
83 void Init(void)
84 {
85     // (internen) Oszillator konfigurieren
86     OSCCON = 0b01100001;             // Sync Oscillator Control Register
87 /*              +------------------- Bit 7 Reserve
88                  +++----------------- Bit 6-4 (IRCF2:IRCF0): Internal Oscillator Frequency
89                     ||||                    Select bits
90                     ||||                 000 : 31 kHz (LFINTOSC)
91                     ||||                 001 : 125 kHz
92                     ||||                 010 : 250 kHz
93                     ||||                 011 : 500 kHz
```

```
94                      ||||                  100 : 1 MHz
95                      ||||                  101 : 2 MHz
96                      ||||                -> 110 : 4 MHz (default)
97                      ||||                  111 : 8 MHz
98                      +------------------ Bit 3 (OSTS): Oscillator Start-up Time-out Status
99                       |||                    (Read Only)
100                      |||                  0 : Device is running from the clock defined
101                      |||                      by FOSC2:FOSC0 of the CONFIG1 register
102                      |||                  1 : Device is running from the internal
103                      |||                      oscillator (HFINTOSC or LFINTOSC)
104                      +------------------ Bit 2 (HTS): HFINTOSC Status bir
105                       ||                    (High Frequency - 125kHz to 8MHz)
106                       ||                    (Read Only)
107                       ||                  0 : HFINTOSC is not stable
108                       ||                  1 : HFINTOSC is stable
109                       +------------------ Bit 1 (LTS): LFINTOSC Status bit
110                        |                    (Low Frequency - 31kHz)
111                        |                    (Read Only)
112                        |                  0 : LFINTOSC is not stable
113                        |                  1 : LFINTOSC is stable
114                        +------------------ Bit 0 (SCS): System Clock Select bit
115                                            0 : Clock source defined by FOSC2:FOSC0
116                                                of the CONFIG1 register
117                                          ->  1 : Internal oscillator is used for system
                                                clock
118 */
119
120     // Ports konfigurieren
121     TRISC = 0b00100000;                    // Richtungsregister Port C (0: Ausgang, 1: Eingang)
122 /*             +---------------------- Bit 7 Port RC7: hier unbenutzt
123                +----------------------- Bit 6 Port RC6: hier unbenutzt
124               +------------------------ Bit 5 Port RC5: hier Eingang (/INT)
125              +------------------------- Bit 4 Port RC4: hier Ausgang (SDA)
126             +-------------------------- Bit 3 Port RC3: hier Ausgang (SCL)
127            +--------------------------- Bit 2 Port RC2: hier unbenutzt
128           +---------------------------- Bit 1 Port RC1: hier unbenutzt
129          +----------------------------- Bit 0 Port RC0: hier unbenutzt
130 */
131 }
132
133
134 /****************** Hauptprogramm ************************************************************/
135
136 /*****************************************************************************************/
137 /* Aufgaben des Hauptprogramms:                                                          */
138 /*   + Controller initialisieren (Unterprogramm Init)                                    */
139 /*   + Takt fuer I2C-Bus festlegen (hier 100kHz mit dem Unterprogramm I2C_Init)          */
140 /*   + 8-Bit-Wert vom Eingabemodul (mit dem PCF8574) lesen und insgesammt vier Byte an   */
141 /*     den I2C-PIC-Slave senden                                                          */
142 /*   + Daten (4 Byte) vom I2C-PIC-Slave lesen und das vierte Byte am I2C-LED-Modul       */
143 /*     (mit einem weiteren PCF8574) ausgeben                                             */
144 /*   + Taetigkeiten/Unterprogramme, die alle zyklisch durchgefuehrt werden muessen:      */
145 /*      + bei jedem Interrupt des PCF8574(A) die Daten vom Eingabemodul (mit dem         */
146 /*        PCF8574) lesen und (wieder) vier Byte am I2C-PIC-Slave ausgeben. Anschlies-    */
147 /*        send Daten (4 Byte) vom I2C-PIC-Slave lesen und das vierte Byte am I2C-LED-    */
148 /*        Modul ausgeben.                                                               */
149 /*****************************************************************************************/
150 void main(void)
151 {
152     char cTemp;
153     char cTemp1;
154     char cTemp2;
155     char cTemp3;
156     char cTemp4;
157
158
159     // Controller initialisieren
160     Init();
161
162     // I2C initialisieren
163     I2C_Init(100000);                      // I2C-Taktfrequenz: 100kHz
```

```
164
165     // Demo 1: 8-Bit-Wert vom I2C-Eingabemodul lesen ...
166     cTemp = I2C_PCF8574_ReadData(1);              // Moduladresse: hier 1
167     // ... und am I2C-PIC-Slave ausgeben
168     I2C_Start();                                  // Startbedingung
169     I2C_Wr(0b00110000);                           // Bausteinadresse + Schreibzugriff
170     I2C_Wr(cTemp);                                // 4 Datenbyte
171     I2C_Wr(0b00001111);                           //   an den
172     I2C_Wr(0b11001100);                           //   PIC-Slave
173     I2C_Wr(cTemp);                                //   schreiben
174     I2C_Stop();                                   // Stopbedingung
175
176
177     // Demo 2: Daten vom I2C-PIC-Slave lesen ...
178     I2C_Start();                                  // Startbedingung
179     I2C_Wr(0b00110001);                           // Bausteinadresse + Lesezugriff
180     cTemp1 = I2C_Rd(ACK);                         // Daten
181     cTemp2 = I2C_Rd(ACK);                         //   vom
182     cTemp3 = I2C_Rd(ACK);                         //   PIC-Slave
183     cTemp4 = I2C_Rd(noACK);                       //   lesen
184     I2C_Stop();                                   // Stopbedingung
185     // ... und am I2C-LED-Modul ausgeben
186     cTemp4 = ~cTemp4;
187     I2C_PCF8574_WriteData(3, cTemp4);             // Moduladresse: hier 3
188
189
190     // Endlosschleife
191     while(1)
192     {
193        // bei jedem Interrupt des PCF8574(A)
194         if (nINTPCF8574 == 0)
195         {
196             // Demo 1: 8-Bit-Wert vom I2C-Eingabemodul lesen ...
197             cTemp = I2C_PCF8574_ReadData(1);      // Moduladresse: hier 1
198             // ... und am I2C-PIC-Slave ausgeben
199             I2C_Start();                          // Startbedingung
200             I2C_Wr(0b00110000);                   // Bausteinadresse + Schreibzugriff
201             I2C_Wr(cTemp);                        // 4 Datenbyte
202             I2C_Wr(0b00001111);                   //   an den
203             I2C_Wr(0b11001100);                   //   PIC-Slave
204             I2C_Wr(cTemp);                        //   schreiben
205             I2C_Stop();                           // Stopbedingung
206
207
208             // Demo 2: Daten vom I2C-PIC-Slave lesen ...
209             I2C_Start();                          // Startbedingung
210             I2C_Wr(0b00110001);                   // Bausteinadresse + Lesezugriff
211             cTemp1 = I2C_Rd(ACK);                 // Daten
212             cTemp2 = I2C_Rd(ACK);                 //   vom
213             cTemp3 = I2C_Rd(ACK);                 //   PIC-Slave
214             cTemp4 = I2C_Rd(noACK);               //   lesen
215             I2C_Stop();                           // Stopbedingung
216
217             // ... und am I2C-LED-Modul ausgeben
218             cTemp4 = ~cTemp4;
219             I2C_PCF8574_WriteData(3, cTemp4);     // Moduladresse: hier 3
220         }
221     }
222 }
```

Listing 6: `I2C_Demo_PIC_Master.c` (Demo 2)

## 5.9 Anmerkungen zur Software für 2. Demo (Master)

Auch dieser Quellcode wurde in C mit der freien Demoversion des mikroC-Compilers erstellt.

Die Software (für den PIC-I²C-Master) besteht auch hier im Wesentlichen aus den folgenden Programmteilen:

- Hauptprogramm am Ende des Quellcodes von `I2C_Demo_PIC_Master.c`, Zeilen 150 bis 222.
- 1 Unterprogramm zur Initialisierung des PIC16F887 (Unterprogramm `Init` in der Datei `I2C_Demo_PIC_Master.c`, Zeilen 83 bis 131).
- 2 Unterprogramme zur Kommunikation mit dem PCF8574 (`I2C_PCF8574_WriteData` und `I2C_PCF8574_ReadData` in der Datei `PCF8574.C`, siehe erstes Demonstrationsbeispiel.

Hier nur kurz die Änderungen im Hauptprogramm. Das Unterprogramm `Init` bleibt hier unverändert:
Das Hauptprogramm ist - verglichen mit dem ersten Demonstrationsbeispiel, Master - etwas länger. Grund dafür ist nur dass jetzt jeweils vier Datenbytes (anstelle von einer, beim ersten Demonstrationsbeispiel) gesendet bzw. gelesen werden müssen. Dies ist, glaube ich, im Quellcode sehr gut erkennbar. Dies ist auch der einzige Unterschied. Anstelle der temporären Register `cTemp1` bis `cTemp4` könnte auch ein Array verwendet werden, doch dies ist hier nicht unbedingt notwendig.

Zum Abschluss noch ein Wort zu den Konfigurationsbits. Auch diese sind hier gleich wie beim ersten Demonstrationsbeispiel.