Ansteuerung eines Feuchte-Temperatur-Sensors (SHTxx)

(mit PIC-Mikrocontroller)

Autor: Buchgeher Stefan

Letzte Bearbeitung: 12. Dezember 2004

Inhaltsverzeichnis

1. Der Feuchtigkeitssensor (SHTxx)

1.1. Allgemeines

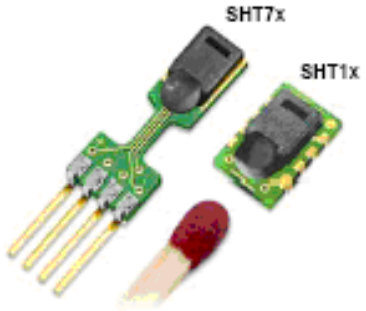

Bild 1.1: Sensor

Der SHTxx[1] (Bild 1.1) ist ein so genannter *single chip* Feuchtigkeits- und Temperatur-Multi-Sensor mit einem werkseitig abgeglichenen Digitalausgang. Dieser Sensor beinhaltet ein kapazitives Polymer-Sensor-Element (für die Feuchtigkeit) und einen Bandgap-Temperatur-Sensor. Diese beiden Teilsensoren sind mit einem 14-Bit-ADC verbunden. Anschließend erfolgt ein serielles Interface.

Jeder SHTxx wurde werksseitig in einer präzisen Feuchtigkeitskammer abgeglichen. Diese Kalibrierwerte sind im Sensor in Form eines speziellen Speichers abgelegt

Das 2-Draht-Interface und die interne Spannungsregelung erlauben eine einfache und schnelle Ansteuerung mit z.B. einem PIC-Mikrocontroller.

Eine weitere Eigenschaft sind die sehr kleinen Gehäuseabmessungen und der geringe Leistungsverbrauch.

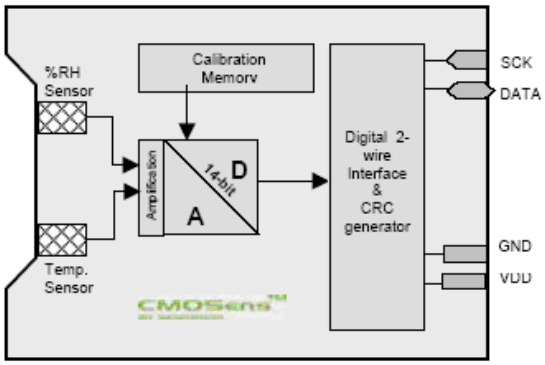

Bild 1.2: Blockschaltbild des Sensors

1.2. Genauigkeit

Für die Sensorgenauigkeit gibt der Hersteller folgende Angaben an:

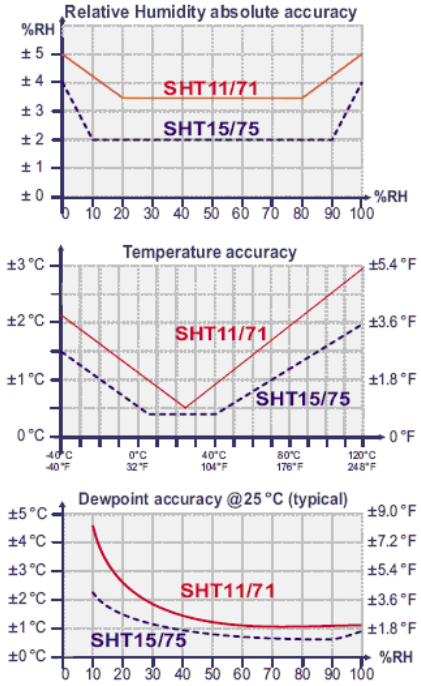

Bild 1.3: Genauigkeit des Sensors

Oder:

SHT11/15:	Feuchtigkeit::	+/- 3%
	Temperatur:	+/- 0.4°C @ 25°C
SHT15/75:	Feuchtigkeit::	+/- 2%
	Temperatur:	+/- 0.3°C @ 25°C

[1] SHT11, SHT15, SHT71, SHT75

2. Hardware

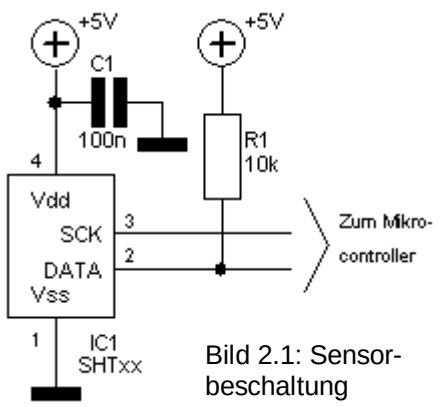

Bild 2.1: Sensor-beschaltung

Bild 2.1 zeigt die minimale Beschaltung des Sensors. Neben dem Sensor selbst ist nur mehr ein Pull-Up-Widerstand (R1) für die bidirektionale Datenleitung (DATA) und ein Stützkondensator C1 notwendig begrenzt.

Stromversorgung:
Der SHTxx benötigt eine Versorgungsspannung zwischen 2.4 und 5.5 Volt. Nach dem Anlegen der Spannung benötigt der Sensor 11ms um von seinem „sleep"-Zustand aufzuwachen. Während dieser Zeit sollten keine Anweisungen an den Sensor gesendet werden. Weiters sollte zwischen den Versorgungspins GND (Vss) und Vdd ein 100nF-Keramikkondensator vorgesehen werden (Im Bild 2.1 ist dies der Kondensator C1).

Serielles Interface:
Das serielle Interface des SHTxx ist für das Auslesen der Sensordaten und für einen geringen Leistungsverbrauch optimiert, und ist ***nicht*** kompatibel zum I²C-Interface!

Die Taktleitung (SCK) ist für die Synchronisierung der Kommunikation mit z.B. einem PIC-Mikrocontroller zuständig.

Die Daten-Tristate-Leitung (DATA) ist für den Datentransfer vom und zum Sensor zuständig. Die Datenleitung darf sich nur nach einer fallenden Flanke der Taktleitung (SCK) ändern und ist gültig bei einer steigenden Flanke der Taktleitung. Während die Taktleitung high ist muss die Datenleitung konstant bleiben.

3. Protokoll

Achtung: Wie schon vorher angemerkt ist das Protokoll zur Kommunikation mit dem Sensor ***nicht*** mit I²C-Protokoll (der Fa. Phillips) identisch.

3.1. Start-Sequenz

Jede Kommunikation mit dem Sensor beginnt mit einer so genannten „Start-Sequenz". Diese Start-Sequenz ist wie folgt definiert:
Diese Startsequenz besteht aus einer Absenkung der Datenleitung (DATA), während die Taktleitung (SCK) high ist. Anschließend muss die Taktleitung (SCK) von high nach low und kurz darauf wieder nach high gehen. Während dieser Zeit muss aber die Datenleitung auf Low-Pegel bleiben. Sie darf erst wieder high werden, wenn auch die Taktleitung (SCK) wieder high ist. Bild 3.1 verbildlicht diese Sequenz.

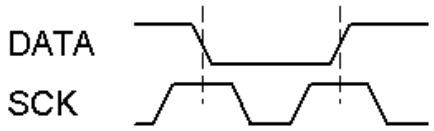

Bild 3.1: Start-Sequenz

Ausnahme: Soll der Sensor zurückgesetzt werden, so erfolgt die Start-Sequenz nach der so genannten „Connection-Reset-Sequenz“ (siehe Abschnitt 3.5).

3.2. Feuchtigkeitsmessung

Um eine Feuchtigkeitsmessung starten zu können, muss zunächst die **Start-Sequenz** (entsprechend Abschnitt 3.1) erfolgen, und anschließend der **Befehlscode** für eine Feuchtigkeitsmessung (Binärcode: 00000101) erfolgen. Ist der Befehlscode gültig, so legt der Feuchtigkeitssensor die Datenleitung zunächst auf Low und anschließend wieder auf high (**Ack**). Nun muss der Mikrocontroller warten bis die Feuchtigkeits-messung beendet ist. Diese Zeit hängt in erster Linie von der gewählten Auflösung auf. (ca. 55ms bei einer 12-Bit-Auflösung bzw. 11ms bei einer 8-Bit-Auflösung). Die Auflösung kann über das Statusregister eingestellt werden (siehe Abschnitt 3.4).
Während der Sensor die Feuchtigkeit misst, bleibt die Datenleitung (DATA) auf High-Pegel und die Taktleitung (SCK) auf Low-Pegel. Ist der Sensor mit der Feuchtigkeits-messung fertig, so legt er die Datenleitung auf Low-Pegel, und signalisiert somit dem Mikrocontroller, dass die Daten für die Feuchtigkeit zur Abholung bereit sind. Dazu wird zunächst das höherwertige Byte vom Mikrocontroller Bit für Bit, beginnend mit dem MSB eingelesen. (Achtung: Bei einer 12-Bit-Auflösung, sind die ersten vier Bits low, bei einer 8-Bit-Auflösung sind alle Bits low). Nun muss der Mikrocontroller den Empfang des höherwertigen Bytes bestätigen, indem er die Datenleitung auf Low-Pegel legt. Anschließend kann der Mikrocontroller das niederwertige Byte Bit für Bit (wieder beginnend mit dem MSB) einlesen und dieses wiederum bestätigen. (Datenleitung auf Low-Pegel). Zum Schluss kann noch eine Checksumme empfangen werden. Das nach der Checksumme folgende Bestätigungsbit (Ack) beendet dann die Kommunikation. Wird die Checksumme nicht benötigt, so kann der Mikrocontroller bereits nach dem letzten Datenbit die Kommunikation mit dem Sensor beenden, indem er die Datenleitung (DATA) nach dem Bestätigungsbit (Ack) auf High legt.
Nachdem die Kommunikation beendet ist kehrt der Sensor automatisch in den „Sleep“-Modus zurück.

Bild 3.2 verbildlicht den gesamten Ablauf einer Feuchtigkeitsmessung beginnend mit der Start-Sequenz bis zur CRC-8-Checksumme

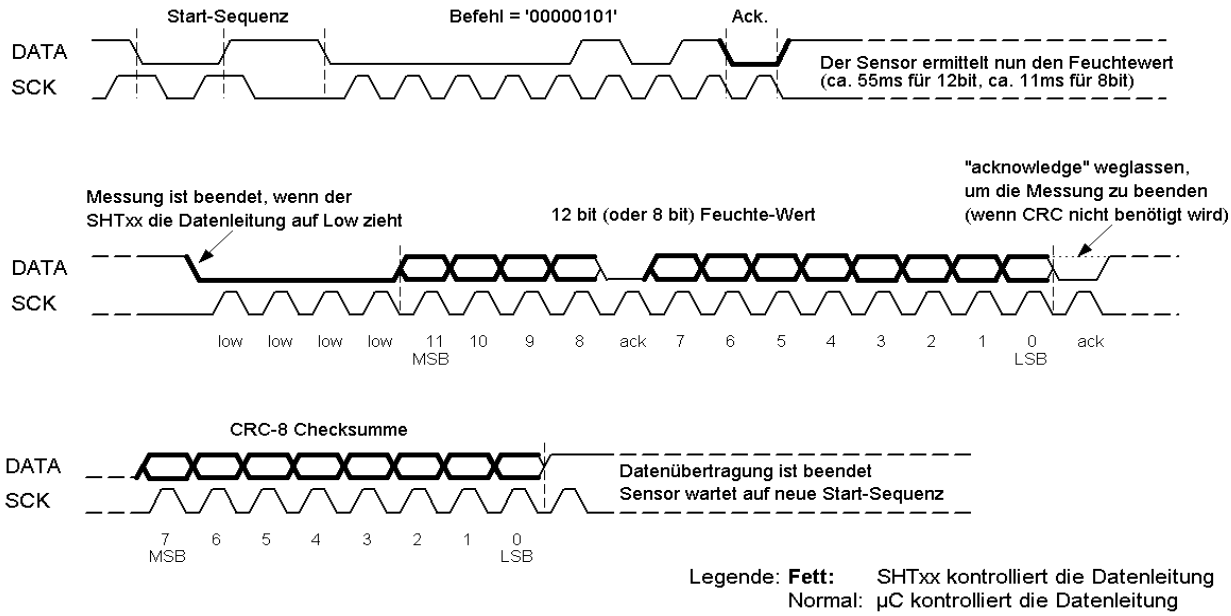

Bild 3.2: Protokoll zur Feuchtigkeitsmessung

Achtung: Damit sich der Sensor nicht um mehr als 0,1°C erwärmt, sollte der Sensor nicht länger als 15% der Zeit aktiv sein. Dies entspricht max. 3 Messungen pro Sekunde bei einer Auflösung von 12 Bit)

3.3. Temperaturmessung

Der Zyklus für eine Temperaturmessung ist prinzipiell gleich wie jener für die Feuchtigkeitsmessung: Zuerst erfolgt die **Start-Sequenz** (entsprechend Abschnitt 3.1). Anschließend der **Befehlscode** für eine Temperaturmessung (Binärcode: 00000011). Ist der Befehlscode gültig, so legt der Feuchtigkeitssensor die Datenleitung zunächst auf Low und anschließend wieder auf high (**Ack**). Nun muss der Mikrocontroller warten bis die Temperaturmessung beendet ist. Diese Zeit hängt in erster Linie von der gewählten Auflösung auf. (ca. 210ms bei einer 14-Bit-Auflösung bzw. 55ms bei einer 12-Bit-Auflösung). Auch hier kann die Auflösung über das Statusregister eingestellt werden (siehe Abschnitt 3.4).
Während der Sensor die Temperatur misst, bleibt die Datenleitung (DATA) auf High-Pegel und die Taktleitung (SCK) auf Low-Pegel. Ist der Sensor mit der Temperaturmessung fertig, so legt er die Datenleitung auf Low-Pegel, und signalisiert somit dem Mikrocontroller, dass die Daten für die Temperatur zur Abholung bereit sind. Dazu wird zunächst das höherwertige Byte vom Mikrocontroller Bit für Bit, beginnend mit dem MSB eingelesen. (Achtung: Bei einer 12-Bit-Auflösung, sind die ersten vier Bits low). Nun muss der Mikrocontroller den Empfang des höherwertigen Bytes bestätigen, indem er die Datenleitung auf Low-Pegel legt. Anschließend kann der Mikrocontroller das niederwertige Byte Bit für Bit (wieder beginnend mit dem MSB) einlesen und dieses wiederum bestätigen. (Datenleitung auf Low-Pegel). Zum Schluss kann noch eine Checksumme empfangen werden. Das nach der Checksumme folgende Bestätigungsbit (Ack) beendet dann die Kommunikation. Wird die Checksumme nicht benötigt, so kann der Mikrocontroller bereits nach dem letzten Datenbit die Kommunikation mit dem Sensor beenden, indem er die Datenleitung (DATA) nach dem Bestätigungsbit (Ack) auf High legt.

Nachdem die Kommunikation beendet ist kehrt der Sensor automatisch in den „Sleep“-Modus zurück.

Bild 3.3 verbildlicht den gesamten Ablauf einer Temperaturmessung beginnend mit der Start-Sequenz bis zur CRC-8-Checksumme

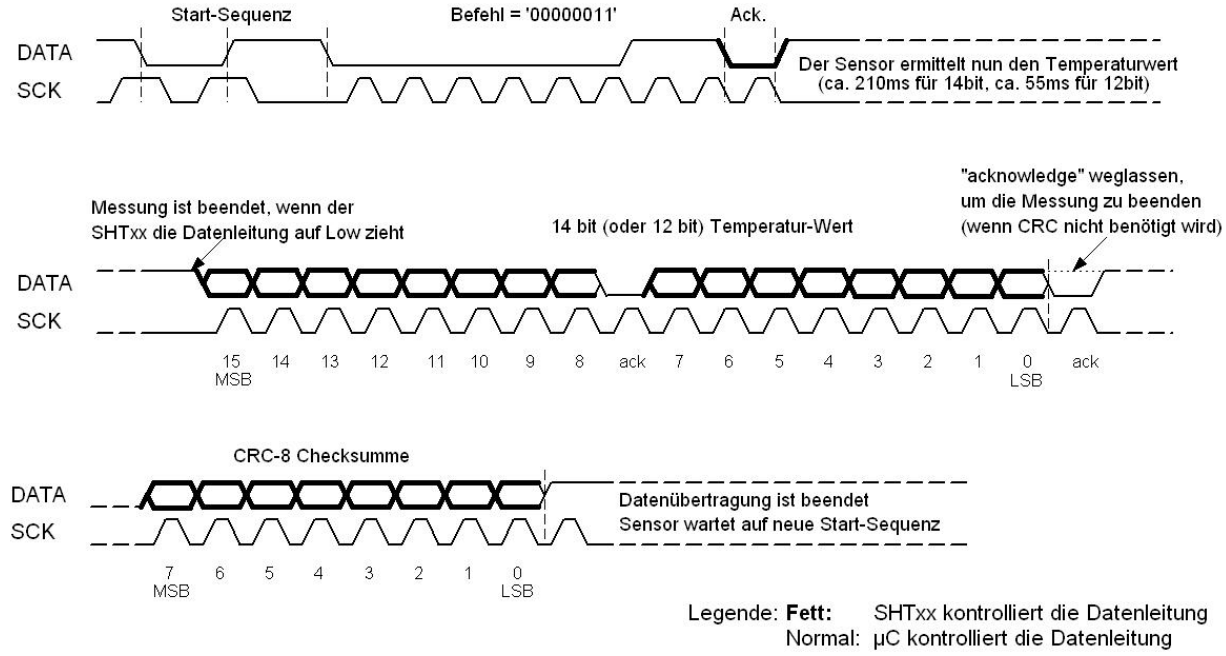

Bild 3.3: Protokoll zur Temperaturmessung

Achtung: Auch bei der Temperaturmessung gilt: Damit sich der Sensor nicht um mehr als 0,1°C erwärmt, sollte der Sensor nicht länger als 15% der Zeit aktiv sein. Dies entspricht max. 3 Messungen pro Sekunde bei einer Auflösung von 12 Bit.

3.4. Sensor-Statusregister lesen bzw. beschreiben

3.4.1. Allgemeines zum Statusregister

Das 8-Bit-Statusregister dient zur Einstellung der zusätzlichen Funktionen und ist laut Tabelle 3.1 aufgebaut.

Bit	Type	Beschreibung	Defaultwert	
7		reserviert	0	
6	R	End of Battery (Geringe Betriebsspannung) ‚0' für Vdd > 2.47V ‚1' für Vdd < 2.47V	x	Kein default, Bit wird nur nach Messung upgedated
5		reserviert	0	
4		reserviert	0	
3		Nicht verwendet, nur für Testzwecke	0	
2	R/W	Heizung	0	aus
1	R/W	Nicht vom OTP laden	0	Reload
0	R/W	‚1' – 8 bit rF / 12 bit Temperatur Auflösung ‚0' – 12 bit rF / 14 bit Temperatur Auflösung	0	12 bit rF 14 bit Temperatur

Tabelle 3.1: Statusregister

Für den Anwender sind nur die Bits 0,1,2 und 6 interessant, wobei die Bits 0,1 und 2 sowohl von der Auswerteelektronik (z.B. ein Mikrocontroller) gelesen als auch beschrieben werden können. Bit 6 gibt an, ob die Versorgungsspannung des Sensors im erlaubten Bereich ist. Dieses Bit wird vom Sensor automatisch gesetzt oder gelöscht, und kann daher von der Auswerteelektronik nur gelesen werden.

Erläuterung zu Bit 0:
Die voreingestellte Mess-Auflösung ist 14Bit (Temperatur) und 12Bit (Feuchtigkeit). Diese Einstellung kann auf 12Bit (Temperatur) und 8Bit (Feuchtigkeit) herabgesetzt werden. Dies ist besonders bei Anwendung nützlich, die schnelle Messungen oder eine niedrige Leistungsaufnahme erfordern.

Erläuterung zu Bit 2:
Auf dem Sensor befindet sich ein Heizelement, welches eingeschaltet (Bit 2 = 1) oder ausgeschaltet (Bit 2 = 0) werden kann. Das eingeschaltet Heizelement kann die Temperatur des Sensorchips um ca. 5°C erhöhen. Die Leistungszunahme beträgt dabei ca. 8mA (bei 5-V-Betriebsspannung).

Anwendung des Heizelements:

- Die Funktion des Sensors kann durch einen Vergleich der Temperatur vor und nach dem Einschalten des Heizelements überprüft werden.
- Die Kondensation des Sensors in sehr feuchter Umgebung kann durch das Einschalten des Heizelements vermieden werden.

Achtung:
Während der Sensor geheizt wird zeigt der Sensor höhere Temperaturen und niedrigere Feuchtigkeitswerte an.

Erläuterung zu Bit 6:
Die „End of Battery“-Funktion prüft auf geringe Versorgungsspannung (<2.47V) bei einer Genauigkeit von +/- 0.05V.

3.4.2. Statusregister des Sensors lesen

Zuerst erfolgt die **Start-Sequenz** (entsprechend Abschnitt 3.1). Anschließend der **Befehlscode** für des Lesen des Statusregisters (Binärcode: 00000111). Ist der Befehlscode gültig, so legt der Feuchtigkeitssensor die Datenleitung auf Low (**Ack**). Nun kann der Mikrocontroller das 8-Bit-Statusregister Bit für Bit, beginnend mit dem MSB einlesen, und muss es anschließend bestätigen, indem er die Datenleitung auf Low-Pegel legt. Zum Schluss kann noch eine Checksumme empfangen werden. Das nach der Checksumme folgende Bestätigungsbit (Ack) beendet dann die Kommunikation. Wird die Checksumme nicht benötigt, so kann der Mikrocontroller bereits nach dem letzten Bit des Statusregisters die Kommunikation mit dem Sensor beenden, indem er die Datenleitung (DATA) nach dem Bestätigungsbit (Ack) auf High legt.
Nachdem die Kommunikation beendet ist kehrt der Sensor automatisch in den „Sleep“-Modus zurück.

Bild 3.4 verbildlicht den gesamten Ablauf beginnend mit der Start-Sequenz bis zur Checksumme.

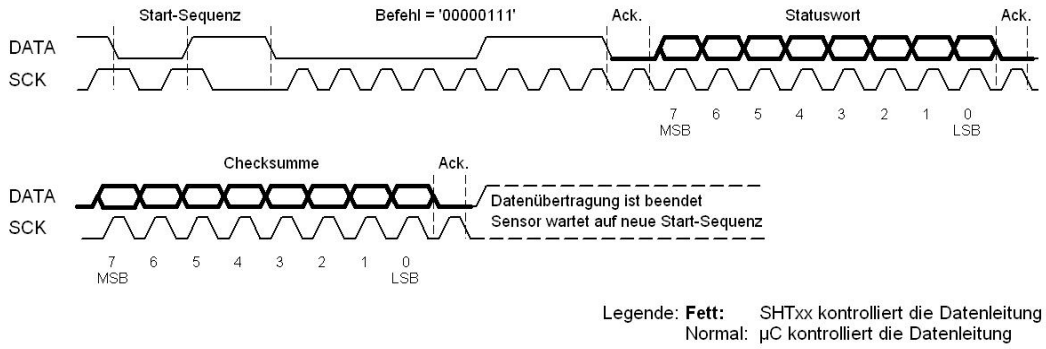

Bild 3.4: Protokoll zum Lesen des Statusregisters

3.4.3. Statusregister des Sensors beschreiben

Zuerst erfolgt die **Start-Sequenz** (entsprechend Abschnitt 3.1). Anschließend der **Befehlscode** für des Schreiben des Statusregisters (Binärcode: 00000110). Ist der Befehlscode gültig, so legt der Feuchtigkeitssensor die Datenleitung auf Low (**Ack**). Nun kann der Mikrocontroller das 8-Bit-Statusregister Bit für Bit beginnend mit dem MSB an den Sensor übertragen. Ist das übertragene Statusregister gültig, so legt der Feuchtigkeitssensor die Datenleitung auf Low (**Ack**). Abschließend kehrt der Sensor automatisch in den „Sleep“-Modus zurück.

Bild 3.5 verbildlicht den gesamten Ablauf beginnend mit der Start-Sequenz bis zur Bestätigung des neuen Statusregisters durch den Sensor.

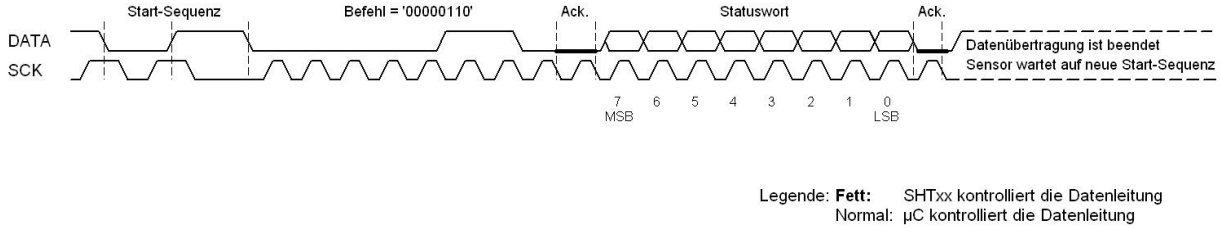

Bild 3.5: Protokoll zum Beschreiben des Statusregisters

3.5. Rücksetzen des Sensors

Wird die Kommunikation mit dem Sensor unterbrochen, so kann die serielle Schnittstelle des Sensors mit der folgenden Sequenz zurückgesetzt werden:

Neun oder mehr Taktimpulse während die Datenleitung (DATA) high ist. Gefolgt von einer Startsequenz entsprechend Abschnitt 3.1. Bild 3.6 verbildlicht diese Sequenz.

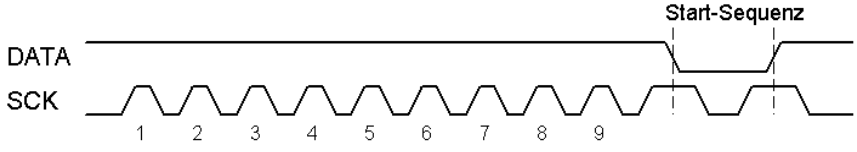

Bild 3.6: „Connection Reset“-Sequenz

3.6. Zeitdiagramm

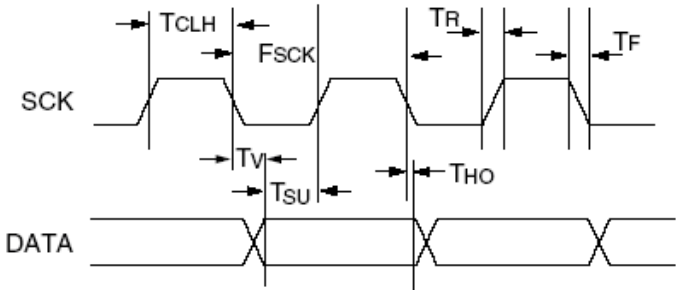

Bild 3.7: Zeitdiagramm

Parameter		Bedingung	Min	Typ	Max	Einheit
SCK-frequency	F_{SCK}	VDD > 4.5 V			10	MHz
		VDD < 4.5 V			1	MHz
DATA fall time	T_{RFO}	Output load 5 pF	3.5	10	20	ns
		Output load 100 pF	30	40	200	ns
SCK high time	T_{CLH}		100			ns
SCK low time	T_{CLL}		100			ns
DATA valid time	T_V			250		ns
Output hold time	T_{HO}			10		ns
DATA setup time	T_{SU}		100			ns
SCK rise/fall time	T_R / T_F				200	ns

Tabelle 3.2: Zeitdiagramm

4. Linearisierung und Temperaturkompensation

Ein Nachteil des Sensors ist, dass sowohl die Temperatur als auch die Feuchtigkeit eine Nichtlineare Kennlinie besitzen. Dazu kommt, dass die Feuchtigkeit von der Temperatur abhängt und daher kompensiert werden muss. Doch glücklicherweise lassen sich sowohl die Nichtlinearität als auch die Kompensation mit den folgenden Formeln korrigieren:

Bei 14- bzw. 12-Bit-Auflösung:
Linearisierung der Temperatur bei 14-Bit-Auflöung:
*Temp(lin) = 0.01 + 0.018 * Sensorwert (Temp)*

Linearisierung der Feuchtigkeit bei 12-Bit-Auflöung:
*Feuchte(lin) = -4 + 0.0405*Sensorwert(Feuchte) - 0.0000028 * Sensorwert(Feuchte)²*

Temperatur-Kompensation der Feuchtigkeit bei 12-Bit-Auflöung:
Feuchte(komp)=(Temp(lin)-25)(0.01+0.00008*Sensorwert(Feuchte))+Feuchte(lin)*

Bei 12- bzw. 8-Bit-Auflösung:
Linearisierung der Temperatur bei 12-Bit-Auflöung:
*Temp(lin) = 0.04 + 0.072 * Sensorwert (Temp)*

Linearisierung der Feuchtigkeit bei 8-Bit-Auflöung:
*Feuchte(lin) = -4 + 0.648*Sensorwert(Feuchte) - 0.00072 * Sensorwert(Feuchte)²*

Temperatur-Kompensation der Feuchtigkeit bei 8-Bit-Auflöung:
Feuchte(komp)=(Temp(lin)-25)(0.01+0.00128*Sensorwert(Feuchte))+Feuchte(lin)*

Anmerkung:
Die Feuchtigkeit kann im 8-Bit-Mode auch mit guter Näherung „einfacher“ ermittelt werden. Der Fehler beträgt dabei max. +/- 0.8%:

Sensorwert(Feuchte) zwischen 0 und 127:
*Feuchte(lin) = (143 * Sensorwert(Feuchte) – 512) / 256*

Sensorwert(Feuchte) zwischen 128 und 255:
*Feuchte(lin) = (111 * Sensorwert(Feuchte) + 2893) / 256*

5. CRC-8 Checksumme-Berechnung

Die CRC-8-Checksumme wird bei meinen Anwendungen nicht benötigt!

6. Software

Die Aufgabe der Software ist das im Abschnitt 3 (Protokoll) beschriebene Protokoll umzusetzen. Weiters die Linearisierung und Temperaturkompensation gemäß Abschnitt 4. Je nach Bedarf kann auch noch die Checksumme gemäß Abschnitt 5 berücksichtigt werden. Dies ist aber hier (noch) nicht ausgeführt!

Als Programmiersprache wurde C, und als Compiler CC5X gewählt.

6.1. Portdefinition, externe Register und Konstanten

Portdefinition:
Im Allgemeinen werden bei jeder Anwendung die Eingangspins für den Feuchtigkeitssensor an einen anderen Portpin verwendet. Damit dies in der Software nur an einer Stelle berücksichtigt werden muss befindet sich in der Software eine Portdefinition. Diese besteht aus den folgenden 3 Parametern:

- DATA: Dieser Parameter gibt den Portpin für die Datenleitung (DATA) an
- TRIS_DATA: Während der Erzeugung des Protokolls muss der DATA-Pin mehrmals zwischen Ein- und Ausgang umgeschaltet werden. Dazu wird der Parameter TRIS_DATA benötigt.
- SCK: Dieser Parameter gibt den Portpin für die Taktleitung (SCK) an

Achtung: Wird für DATA der Port B verwendet, so muss für TRIS_DATA das zum Port B zugehörige TRIS-Register definiert werden. Eine mögliche Portdefinition für einen PIC-Mikrocontroller der PIC16Fxx-Familie ist:

```
bit   DATA              @ PORTB.0;
bit   TRIS_DATA         @ TRISB.0;
bit   SCK               @ PORTB.3;
```

Externe Register:
Die externen Register sind in einer Struktur (struct Sensor) zusammengefasst und beinhalten die Rohdaten vom Sensor und die daraus berechnete Temperatur und Luftfeuchtigkeit

```
struct Sensor
{
      unsigned char     feuchte_lo; // Rohwert vom Sensor (Low-Byte der
                                    // Feuchte)
      unsigned char     feuchte_hi; // Rohwert vom Sensor (High-Byte der
                                    // Feuchte)
      unsigned char     temperatur_lo; // Rohwert vom Sensor (Low-Byte der
                                    // Temperatur)
      unsigned char     temperatur_hi; // Rohwert vom Sensor (High-Byte der
                                    //Temperatur)
      long        temperatur_lin;   // linearisierter Temperaturwert
      long        feuchte_lin;      // linearisierter Feuchtewert
      unsigned long     feuchte_komp; // kompensierten Feuchtewert
};
```

Konstanten für den Feuchtigkeitssensor:

- STATUS_SCHREIBEN: Diese Konstante beinhaltet den Befehlscode für das Beschreiben des Statusregisters des Sensors (siehe Abschnitt 3.4.3.)
- STATUS_LESEN: Diese Konstante beinhaltet den Befehlscode für das Lesen des Statusregisters des Sensors (siehe Abschnitt 3.4.2.)
- MESSUNG_TEMPERATUR: Diese Konstante beinhaltet den Befehlscode für eine Temperaturmessung (siehe Abschnitt 3.3.)
- MESSUNG_FEUCHTE: Diese Konstante beinhaltet den Befehlscode für eine Feuchtigkeitsmessung (siehe Abschnitt 3.2.)
- ACK, noACK: Diese beiden Konstanten geben an, ob beim Telegramm zum Sensor, eine Bestätigung (ACK) oder keine Bestätigung (noACK) erfolgen sollen
- TEMPERATUR, FEUCHTE: Mit diesen beiden Konstanten wird zwischen einer Temperaturmessung (TEMPERATUR) und einer Feuchtigkeitsmessung (FEUCHTE) unterschieden

6.2. Initialisierung (Unterprogramm „INIT“)

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

Der folgende Programmausschnitt zeigt eine mögliche Initialisierungsroutine für den PIC16F628. Der Feuchtigkeitssensor ist hier am Port B angeschlossen. Die schwarz hervorgehobenen Stellen sind für die Implementierung des Protokolls notwendig.

(Anmerkung: die grau markierten Stellen dienen zur Konfiguration eines Timer-Interrupts und zur Konfigurierung einer RS232-Schnittstelle)

```
void INIT(void)
{
      TMR0 = 0;                    // Timer0 auf 0 voreinstellen
      OPTION = 0b.0000.0111;

      TRISB = 0;

      ZAEHLERISR10SEK = KONSTISR10SEK;

      // UART initialisieren
      TXSTA = 0b.0010.0100;        // Sender
      RCSTA = 0b.1001.0000;        // Empfaenger
      SPBRG = 12;                  // Baudrate (19200 Baud -> SPBRG = 12)
}
```

6.3. Unterprogramme zur Kommunikation mit dem Feuchtigkeitssensor

Zur Kommunikation mit dem Feuchtigkeitssensor sind folgende Unterprogramme notwendig:

- Ein Unterprogramm zur Erzeugung der Startbedingung. (*FEUCHTE_START*)
- Ein Unterprogramm welches einen Verbindungsreset des Sensors durchführt (*FEUCHTE_CONNECTIONRESET*)
- Je ein Unterprogramm welches ein Byte auf den Bus schreibt (*FEUCHTE_SCHREIBEN*) bzw. ein Byte vom Bus einliest (*FEUCHTE_LESEN*).
- Je ein Unterprogramm welches das Statusregister vom Sensor beschreibt (*FEUCHTE_STATUSSCHREIBEN*) bzw. den Inhalt vom Statusregister des Sensors einliest (*FEUCHTE_STATUSLESEN*).
- Ein Unterprogramm, welches einen Messzyklus startet, auf das Messergebnis wartet und diesem vom Sensor ausliest (*FEUCHTE_MESSUNG*)
- Ein Unterprogramm, welches aus den Rohdaten des Sensors die Temperatur und Luftfeuchtigkeit berechnet (*FEUCHTE_LINEARISIERUNG*)

Nun aber zu den einzelnen Unterprogrammen im Detail:

6.3.1. Unterprogramm FEUCHTE_START

Aufgabe:
Dieses Unterprogramm erzeugt die Startbedingung für die Kommunikation mit dem Feuchtigkeitssensor.

Zur Erinnerung, die Startbedingung ist wie folgt definiert (vgl. Abschnitt 3.1.):

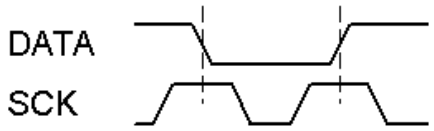

Bild 6.1: Start-Sequenz

Hier das Unterprogramm:

```
void FEUCHTE_START(void)
{
      DATA = 1;
      SCK  = 0;
      SCK  = 1;
      DATA  = 0;
      SCK  = 0;

      #asm
      nop
      #endasm

      SCK  = 1;
      DATA  = 1;
      SCK  = 0;
}
```

6.3.2. Unterprogramm FEUCHTE_CONNECTIONRESET

Aufgabe:
9 SCK-Zyklen, während DATA = 1, anschließend eine Startbedingung (vgl. Abschnitt 3.6.)

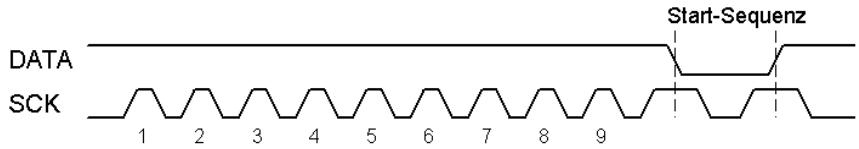

Bild 6.2: „Connection Reset“-Sequenz

Hier das Unterprogramm:

```
void FEUCHTE_CONNECTIONRESET(void)
{
      unsigned char i;

      DATA = 1;                     // Initialisieren
      SCK = 1;
      for(i=0;i<9;i++)              // 9 Takt-Zyklen
      {
            SCK = 1;
            SCK = 0;
      }
      FEUCHTE_START();              // Startbedingung
}
```

6.3.3. Unterprogramm FEUCHTE_SCHREIBEN

Aufgabe:
Ein Byte (also eine Anweisung) an den Sensor übergeben und die Bestätigung vom Sensor empfangen.

Vorgehensweise:
- Register fehler löschen
- Mit einer Schleife jedes einzelne Bit des zu übergebenden Bytes maskieren und das Ergebnis auf die DATA-Leitung geben. Anschließend einen Taktimpuls erzeugen
- Nun die Bestätigung (Acknowledge) empfangen:
  - DATA = 1
  - SCK = 1 (steigende Flanke der Takt-Leitung)
  - Das TRIS_DATA-Flag muss nun gesetzt werden, da der DATA-Pin auf Eingang umgeschaltet werden muss.
  - Nun die DATA-Leitung einlesen und in fehler sichern
  - SCK = 0 (fallende Flanke der Takt-Leitung)
  - Das TRIS_DATA-Flag muss wieder gelöscht werden, da der DATA-Pin nun wieder als Ausgang dienen soll.

Hier das Unterprogramm:
```
char FEUCHTE_SCHREIBEN(unsigned char value)
{
      unsigned char i;
      unsigned char fehler;

      fehler = 0;                   // Register fehler loeschen

      for(i=0x80;i>0;i/=2)          // Mit einer Schleife jedes einzelne Bit
      {
            if (i & value) DATA = 1; // ... des zu uebergebenden Bytes
                                    // maskieren und das
            else DATA = 0;          // Ergebnis auf die DATA-Leitung geben.
                                    // Anschliessend einen Taktimpuls erzeugen
            SCK = 1;                // Taktimpuls

            #asm
            nop
            #endasm

            SCK = 0;
      }
      DATA = 1;
      SCK = 1;                      // steigende Flanke der Takt-Leitung
      TRIS_DATA = 1;                // DATA-Pin auf Eingang umschalten
      fehler = DATA;                // DATA einlesen und in fehler sichern
      SCK = 0;                      // fallende Flanke der Takt-Leitung
      TRIS_DATA = 0;                // DATA-Pin auf Ausgang zurueckschalten
      return(fehler);
}
```

6.3.4. Unterprogramm FEUCHTE_LESEN

Aufgabe:
Ein Byte vom Sensor lesen und eine Bestätigung zurückgeben, wenn ack=1

Vorgehensweise:
- DATA = 1
- Das TRIS_DATA-Flag muss nun gesetzt werden, da der DATA-Pin auf Eingang umgeschaltet werden muss.
- Mit einer Schleife das 8-Bit Datenwort bitweise einlesen
- Das TRIS_DATA-Flag muss wieder gelöscht werden, da der DATA-Pin nun wieder als Ausgang dienen soll.
- Nun die Bestätigung (Acknowledge) senden
- DATA = 1

Hier das Unterprogramm:
```
char FEUCHTE_LESEN(unsigned char ack)
{
      unsigned char i;
      unsigned char val = 0;

      DATA = 1;
      TRIS_DATA = 1;                // DATA-Pin auf Eingang umschalten
      for(i=0x80;i>0;i/=2)          // Mit einer Schleife das 8-Bit Datenwort
                                    // bitweise einlesen
      {
            SCK = 1;
            if (DATA) val = (val | i);
            SCK = 0;
      }
      TRIS_DATA = 0;                // DATA-Pin auf Ausgang zurueckschalten
      DATA = !ack;                  // Bestaetigung (Acknowledge) senden
      SCK = 1;

      #asm
      nop
      #endasm

      SCK = 0;
      DATA = 1;
      return (val);
}
```

6.3.5. Unterprogramm FEUCHTE_STATUSSCHREIBEN

Aufgabe:
Statusregister des Sensors beschreiben

Vorgehensweise:
- Startbedingung mit dem Unterprogramm FEUCHTE_START erzeugen
- Anweisung zum Beschreiben des Sensor-Statusregister (mit dem Unterprogramm FEUCHTE_SCHREIBEN mit dem Parameter STATUS_SCHREIBEN (=0x06))
- Den Wert p_value mit dem Unterprogramm FEUCHTE_SCHREIBEN in das Statusregister des Sensors schreiben

Hier das Unterprogramm:

```
char FEUCHTE_STATUSSCHREIBEN(unsigned char p_value)
{
      unsigned char fehler = 0;

      FEUCHTE_START();

      fehler += FEUCHTE_SCHREIBEN(STATUS_SCHREIBEN);
      fehler += FEUCHTE_SCHREIBEN(p_value);

      return (fehler);
}
```

6.3.6. Unterprogramm FEUCHTE_STATUSLESEN

Aufgabe:
Statusregister des Sensors und Checksumme auslesen

Vorgehensweise:

- Startbedingung mit dem Unterprogramm FEUCHTE_START erzeugen
- Anweisung zum Auslesen des Sensor-Statusregister (mit dem Unterprogramm FEUCHTE_SCHREIBEN mit dem Parameter STATUS_LESEN (=0x07))
- Status-Register mit dem Unterprogramm FEUCHTE_LESEN einlesen und im Übergabeparameter *p_value sichern
- Checksumme mit dem Unterprogramm FEUCHTE_LESEN einlesen und im Übergabeparameter *p_checksum sichern.

Hier das Unterprogramm:

```
char FEUCHTE_STATUSLESEN(unsigned char *p_value, unsigned *p_checksum)
{
      unsigned char fehler = 0;
      char temp;

      FEUCHTE_START();              // Startbedingung

      fehler = FEUCHTE_SCHREIBEN(STATUS_LESEN); //Anweisung zum Auslesen des
                                    // Sensor-Statusregister

      temp = FEUCHTE_LESEN(ACK);    // Status-Register einlesen und in
      *p_value = temp;              // *p_value sichern

      Temp = FEUCHTE_LESEN(noACK);  // Checksumme einlesen und in
      *p_checksum = temp;           // *p_checksum sichern

      return (fehler);
}
```

6.3.7. Unterprogramm FEUCHTE_MESSUNG

Aufgabe:
Einen Feuchtigkeits- oder Temperatur-Messzyklus starten

Vorgehensweise:

- Startbedingung erzeugen

- Je nach Mode (Temperaturmessung oder Feuchtigkeitsmessung) den Wert 03h (für eine Temperaturmessung) oder 05h (für eine Feuchtigkeitsmessung) mit dem Unterprogramm FEUCHTE_SCHREIBEN dem Sensor übergeben, und so diese Messung starten.
- Die DATA-Leitung auf Eingang umschalten und warten bis der Sensor diese DATA-Leitung auf Low legt, als Zeichen, dass er mit der Messung fertig ist und die Messwerte abholbereit sind.
- Nun das High-Byte und das Low-Byte mit Hilfe des Unterprogramms FEUCHTE_LESEN einlesen.

Hier das Unterprogramm:

```
char FEUCHTE_MESSUNG(unsigned char mode)
{
      unsigned char fehler;
      unsigned int i;
      unsigned char temp;


      fehler = 0;

      FEUCHTE_START();              // Startbedingung erzeugen

      switch(mode)                  // je nach Mode entweder eine Temperatur
      {                             // oder eine Feuchtemessung starten
            case TEMPERATUR:  fehler += FEUCHTE_SCHREIBEN(MESSUNG_TEMPERATUR);
                              break;
            case FEUCHTE:     fehler += FEUCHTE_SCHREIBEN(MESSUNG_FEUCHTE);
                              break;
            default:          break;
      }

      TRIS_DATA = 1;                // DATA-Leitung auf Eingang umschalten

      for (i=0;i<65535;i++) if (DATA == 0) break;
      if (DATA) fehler += 1;        // Time-out! -> Fehlerflag setzen

      temp = FEUCHTE_LESEN(ACK);    // High-Byte einlesen und je nach mode
      switch(mode)                  // in temperatur_hi oder feuchte_hi laden
      {
            case TEMPERATUR:        Sensordaten.temperatur_hi = temp; break;
            case FEUCHTE:           Sensordaten.feuchte_hi = temp; break;
            default:    break;
      }

      temp = FEUCHTE_LESEN(noACK);  // Low-Byte einlesen und in
      switch(mode)                  // in temperatur_lo oder feuchte_lo laden
      {
            case TEMPERATUR:        Sensordaten.temperatur_lo = temp; break;
            case FEUCHTE:           Sensordaten.feuchte_lo = temp; break;
            default:    break;
      }

      return (fehler);
}
```

6.3.8. Unterprogramm FEUCHTE_LINEARISIERUNG

Aufgabe:
Aus den vom Sensor empfangenen Roh-Daten die Temperatur und die Feuchtigkeit berechnen. Anschließend muss der Feuchtigkeitswert linearisiert und mit der Temperatur kompensiert werden.

Formeln: (8/12-Bit-Mode):
- Temperatur = 0.04 * (256*temperatur_hi + temperatur_lo) - 40
- Feuchte:
  0 <= feuchte_lo <= 107: Feuchte = (143 * feuchte_lo - 512) / 256
  108 <= feuchte_lo <= 255: Feuchte = (111 * feuchte_lo + 2893) / 256
  Feuchte(komp) = (Temperatur-25)*(0.01 + 0.00128*rH) + Feuchte

Formeln: (12/14-Bit-Mode)
- Temperatur = 0.01 * (256*temperatur_hi + temperatur_lo) – 40
- Feuchte:
  rF = 256*feuchte_hi + feuchte_lo
  Feuchte = -4 + 0.0405 * rH - 0.0000028 * rH * rH
  Feuchte(komp) = (Temperatur-25)*(0.01 + 0.00008*rH) + Feuchte

Hier das Unterprogramm:
```
void FEUCHTE_LINEARISIERUNG(void)
{
      unsigned long     feuchte;
      unsigned long     feuchte_lin1;
      long              feuchte_komp;
      long              feuchte_komp1;
      unsigned long     feuchte_komp2;


      // Berechnung der Temperatur
      Sensordaten.temperatur_lin = (long)(256 * Sensordaten.temperatur_hi);
      Sensordaten.temperatur_lin = Sensordaten.temperatur_lin +
Sensordaten.temperatur_lo;

      // Temperaturwert linearisieren
      Sensordaten.temperatur_lin = (long)(Sensordaten.temperatur_lin / 10);
      Sensordaten.temperatur_lin = Sensordaten.temperatur_lin - 400;

      // Feuchtewert berechnen
      feuchte = (long)(256 * Sensordaten.feuchte_hi);
      feuchte = feuchte + Sensordaten.feuchte_lo;

      // Feuchtewert linearisieren
      Sensordaten.feuchte_lin = feuchte / 10;
      Sensordaten.feuchte_lin = Sensordaten.feuchte_lin * 4;
      Sensordaten.feuchte_lin = Sensordaten.feuchte_lin / 10;
      Sensordaten.feuchte_lin = Sensordaten.feuchte_lin - 4;

      feuchte_lin1 = Sensordaten.feuchte_lin / 100;
      feuchte_lin1 = feuchte_lin1 * feuchte_lin1;
      feuchte_lin1 = feuchte_lin1 * 28;
      feuchte_lin1 = feuchte_lin1 / 1000;

      Sensordaten.feuchte_lin = Sensordaten.feuchte_lin - feuchte_lin1;

      // Feuchtewert kompensieren
```

```
        feuchte_komp1 = (long)(Sensordaten.temperatur_lin / 10);
        feuchte_komp1 = feuchte_komp1 - 25;

        feuchte_komp2 = (long)(feuchte / 100);
        feuchte_komp2 = feuchte_komp2 * 8;
        feuchte_komp2 = feuchte_komp2 + 10;

        feuchte_komp = feuchte_komp1 * feuchte_komp2;
        feuchte_komp = feuchte_komp / 1000;
        Sensordaten.feuchte_komp = feuchte_komp + Sensordaten.feuchte_lin;

        if (Sensordaten.feuchte_komp > 100)
        {
              Sensordaten.feuchte_komp = 100;
        }
}
```

7. Demonstrationsbeispiel

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

7.1. Hardware

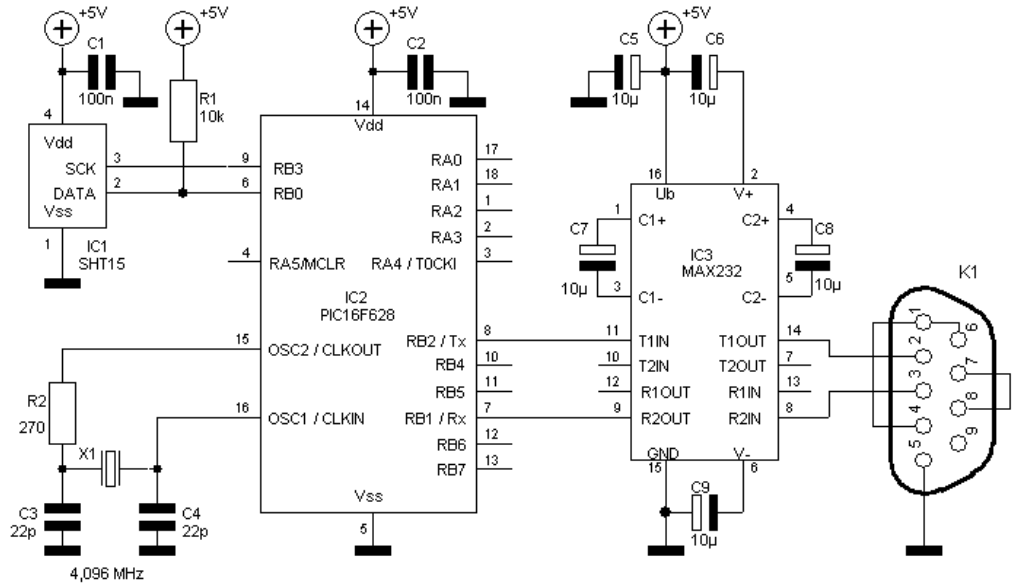

Bild 7.1: Schaltung zur Demonstration

Bei diesem Demonstrationsbeispiel misst der Sensor (IC1) zyklisch (ca. alle 10 Sekunden) die Feuchtigkeit und die Temperatur und übergibt diese dem steuernden Mikrocontroller (IC2) vom Typ PIC16F628. Dieser berechnet aus den vom Sensor empfangenen Daten die Temperatur und die relative Luftfeuchtigkeit. Diese Werte können nun via RS232-Schnittstelle von einem PC angezeigt werden. Dazu ist ein Schnittstellentreiber (IC3) vom Typ MAX232 mit einigen Kondensatoren (C5 bis C9) notwendig.
Die Beschaltung des Sensors erfolgt gemäß Abschnitt 2 mit einem Pull-Up-Widerstand (R1) für die DATA-Leitung und dem Stützkondensator C1

Für die Takterzeugung dient eine Standardbeschaltung bestehend aus einem 4,096-MHz-Quarz (X1), zwei Kondensatoren (C3, C4) und einem Widerstand (R2).

7.2. Software

```
/*****************************************************************************/
/* Demonstrationsbeispiel zum Feuchtigkeitssensor in C (CC5X)                */
/*                                                                           */
/* Entwickler: Buchgeher Stefan                                              */
/* Entwicklungsbeginn der Software: 1. Juli 2004                             */
/* Funktionsfähig seit: 2. Juli 2004                                         */
/* Letzte Bearbeitung: 2. Dezember 2004                                      */
/*****************************************************************************/


/****************** Include-Dateien ******************************************/
#include <int16CXX.H>               // Ist fuer die Interrupts notwendig
#include <INLINE.H>


/****************** Pragma-Anweisungen ***************************************/
#pragma library 1                   //.. library functions that are deleted if unused


/****************** Strukturen  **********************************************/
struct Sensor
{
       unsigned char  feuchte_lo;    // Rohwert vom Sensor (Low-Byte der Feuchte)
       unsigned char  feuchte_hi;    // Rohwert vom Sensor (High-Byte der Feuchte)
       unsigned char  temperatur_lo; // Rohwert vom Sensor (Low-Byte der Temperatur)
       unsigned char  temperatur_hi; // Rohwert vom Sensor (High-Byte der Temperatur)
       long           temperatur_lin; // linearisierter Temperaturwert
       long           feuchte_lin;   // linearisierter Feuchtewert
       unsigned long  feuchte_komp;  // kompensierten Feuchtewert
};


/****************** Externe Register  ****************************************/
char FLAGISRHP;                     // beinhaltet Botschaftsflags ISR -> HP
char ZAEHLERISR10SEK;               // Zaehlregister fuer 10-Sekunden-Zeitbasis

struct Sensor Sensordaten;          // beinhaltet die Rohdaten vom Sensor und die daraus
                                    // berechnete Temperatur und Luftfeuchtigkeit


/****************** Bits in den externen Registern ***************************/
/* Register FLAGSISRHP */
Bit FLAG10SEK                       @ FLAGISRHP.0;


/****************** Portbelegung *********************************************/
/* Port A */
/* Port B */
Bit DATA                            @ PORTB.0;
bit TRIS_DATA                       @ TRISB.0;
bit SCK                             @ PORTB.3;


/****************** Konstanten ***********************************************/
/* Konstante fuer Zeitbasis */
#define        KONSTISR10SEK         156

/* Konstanten fuer den Feuchte-Sensor */
#define        STATUS_SCHREIBEN      0x06
#define        STATUS_LESEN          0x07
#define        MESSUNG_TEMPERATUR    0x03
#define        MESSUNG_FEUCHTE       0x05

#define        ACK                   1
#define        noACK                 0

#define        TEMPERATUR            0
#define        FEUCHTE               1
```

```
/****************** Konfigurations ***************************************************/
#pragma config |= 0b.11110100100010
/*                   ++++------------ Bit 13-10: Code Protection
                     |||||||||     CP1:CP0 (Bit 13=Bit 11=CP1, Bit 12=Bit 10=CP0)
                     |||||||||         2k programm memory
                     |||||||||       0 0 : 0000h-07FFh code protected
                     |||||||||       0 1 : 0200h-07FFhcode protected
                     |||||||||       1 0 ; 0400h-07FFh code protected
                     |||||||||    -> 1 1 : code protection off
                     |||||||||         1k programm memory
                     |||||||||       0 0 : 0000h-03FFh code protected
                     |||||||||       0 1 : 0200h-03FFhcode protected
                     |||||||||       1 0 ; code protection off
                     |||||||||       1 1 : code protection off
                     +------------ Bit 11-9: Reserve
                      +----------- Bit  8 (CPD): Data Code Protection
                       ||||||||         0 : CPD on    (= enabled)
                       ||||||||    ->   1 : CPD off   (= disabled)
                       +---------- Bit  7 (LVP): Low Voltage Programming
                        |||||||     ->  0 : RB4/PGM = I/O (LVP disabled)
                        |||||||         1 : RB4/PGM = PGM (LVP enabled)
                        +--------- Bit 6 (BODEN): Brown-Out Detection
                         ||||||    ->   0 : BODEN on  (= disabled)
                         ||||||         1 : BODEN off (= enabled)
                         +-------- Bit 5 (MCLRE)
                          |||||         0 : RA5 = I/O
                          |||||     ->   1 : RA5 = MCLR
                          |+------ Bit 3 (PWRTE): Power Up Timer
                          | |||    ->   0 : PWRT on   (= enabled)
                          | |||         1 : PWRT off  (= disabled)
                          | +----- Bit 2 (WDTE): Watchdog Timer
                          |  ||    ->   0 : WDT off   (= disabled)
                          |  ||         1 : WDT on    (= enabled))
                          +--++--- Bit 4,1-0 (FOSC2:FOSC0): Oszillator Selectio
                                     000 : LP Oszillator    (RA6=CLKOUT, RA7=CLKIN)
                                     001 : XT Oszillator    (RA6=CLKOUT, RA7=CLKIN)
                                  -> 010 : HS Oszillator    (RA6=CLKOUT, RA6=CLKIN)
                                     011 : EC Oszillator    (RA6=I/O, RA7=CLKIN)
                                     100 : INTRC Oszillator (RA6=I/O, RA7=I/O)
                                     101 : INTRC Oszillator (RA6=CLKOUT, RA7=I/O)
                                     110 : ER Oszillator    (RA6=I/O, RA7=CLKIN)
                                     111 : ER Oszillator    (RA6=CLKOUT, RA7=CLKIN)
*/


/****************** Funktionsprototypen **********************************************/
/* Initialisierung des Mikrocontroller */
void INIT(void);

/* Unterprogramme zur Datenausgabe */
void AUSGABE(void);
void AUSGABEBCD(unsigned long Wert, unsigned char mode);

/* Unterprogramme fuer den Feuchtesensor */
void FEUCHTE_START(void);
void FEUCHTE_CONNECTIONRESET(void);
char FEUCHTE_SCHREIBEN(unsigned char value);
char FEUCHTE_LESEN(unsigned char ack);
char FEUCHTE_STATUSLESEN(unsigned char *p_value, unsigned *p_checksum);
char FEUCHTE_STATUSSCHREIBEN(unsigned char p_value);
char FEUCHTE_MESSUNG(unsigned char mode);
void FEUCHTE_LINEARISIERUNG(void);

/* Unterprogramme fuer RS232 */
void RS232_SENDE_ZEICHEN(unsigned char value);


/****************** ISR - Timer0 *****************************************************/

/*********************************************************************************/
/* Interrupt Service Routine:                                                    */
/*                                                                               */
/* Aufruf:                                                                       */
/*   alle 64 ms                                                                  */
/*                                                                               */
/* Aufgaben:                                                                     */
/*   + w-Register (=Arbeitsregister) und Status-Register zwischenspeichern       */
/*    (int_save_registers)                                                       */
```

```
/*   + Zeitbasis fuer 10 Sekunden erzeugen                                      */
/*   + Das Timer-Interrupt-Flag T0IF wieder loeschen                            */
/*   + w-Register (=Arbeitsregister) und Statusregister wiederherstellen        */
/*     (int_restore_registers)                                                  */
/*****************************************************************************/
#pragma origin 4

interrupt InterruptRutine(void)             // Interruptroutine
{
       int_save_registers                   // W, STATUS (und  PCLATH) retten

       // Beginn der eigentlichen ISR-Routine
       ZAEHLERISR10SEK--;
       if (ZAEHLERISR10SEK == 0)
       {
              FLAG10SEK = 1;                // Botschaftsflag setzen
              ZAEHLERISR10SEK = KONSTISR10SEK; // Zaehlregister fuer den 10-Sekundentakt mit
                                            // der Konstanten KONSTISR10SEK neu laden
       }
       T0IF = 0;
       // Ende der eigentlichen ISR-Routine

       int_restore_registers                // W, STATUS (und PCLATH) Wiederherstellen
}


/***************** Hauptprogramm ************************************************/

/*****************************************************************************/
/* Aufgaben des Hauptprogramms:                                              */
/*   + Controller initialisieren (Unterprogramm INIT)                        */
/*   + Feuchtesensor initialisieren (Unterprogramm FEUCHTE_CONNECTIONRESET)   */
/*   + Interrupt freigeben                                                   */
/*   + Taetigkeiten/Unterprogramme, die alle 10 Sekunden durchgefuehrt werden muessen:  */
/*     + Temperatur- und Feuchtigkeitsmessung starten                        */
/*     + Bei einer fehlerhaften Messung (fehler ist ungleich 0 ) Feuchtesensor neu  */
/*       initialisieren                                                      */
/*     + Bei einer erfolgreichen Messung (fehler = 0) aus dem vom Sensor ermittelten  */
/*       Daten die Temperatur und die relative Luftfeuchtigkeit berechnen    */
/*       (Unterprogramm FEUCHTE_LINEARISIERUNG) und am PC ausgeben           */
/*       (Unterprogramm AUSGABE)                                             */
/*     + Botschaftsflag fuer den 10-Sekunden-Takt loeschen                   */
/*****************************************************************************/
void main(void)
{
       unsigned char fehler;
       unsigned char checksum;

       INIT();                              // Controller initialisieren
       FEUCHTE_CONNECTIONRESET();           // Feuchtesensor initialisieren

       INTCON = 0b.1010.0000;               // Timer0 freigeben durch Setzen von
                                            // GIE und T0IE im Register INTCON

       if (FLAG10SEK)                       // Alle 10 Sekunden
       {
              fehler = 0;
              fehler += FEUCHTE_MESSUNG(FEUCHTE);  // eine Feuchtemessung und eine
              fehler += FEUCHTE_MESSUNG(TEMPERATUR); // Temperaturmessung ausführen

              if (fehler != 0)              // Bei einem Fehler den
              {
                     FEUCHTE_CONNECTIONRESET();  // Feuchtesensor neu initialisieren
              }
              else                          // andernfalls
              {
                     FEUCHTE_LINEARISIERUNG();  // Messwerte liniearisieren und kompensieren
                     AUSGABE();             // und am PC ausgeben
              }

              FLAG10SEK = 0;                // Botschaftsflag fuer den 10-Sekunden-Takt
              }                             // loeschen
       }
}


/***************** Unterprogramme und Funktionen ********************************/

/***************** Mathematik-Routinen ******************************************/
```

```
#include <MATH16.H>


/*****************************************************************************/
/* Initialisierung des Prozessor:                                            */
/*   + Timer 0 (TMR0) loeschen                                               */
/*   + TMR0-ISR soll alle 64ms aufgerufen werden, daher Vorteiler mit 256 laden (Bei    */
/*     einem 4,096-MHz-Quarz)                                                */
/*   + Zaehlregister fuer den 10-Sekundentakt vorbelegen                     */
/*   + UART initialisieren                                                   */
/*     + Sender: Register TXSTA                                              */
/*     + Empfaenger: Register RCSTA                                          */
/*     + Baudrate: Register SPBRG                                            */
/*       Diese Berechnet sich nach folgender Formel:                         */
/*                                                                           */
/*                            Fosc [MHz]                                     */
/*                   SPBRG = ------------- - 1                               */
/*                           16 * Baudrate                                   */
/*                                                                           */
/*       Ist z.B. eine Baudrate von 19200 Baud erwuenscht, und als Takt wird ein       */
/*       4,096MHz-Quarz verwendet, so ergibt sich fuer SPBRG folgender Wert:  */
/*                                                                           */
/*                            Fosc [MHz]          4096000                    */
/*                   SPBRG = ------------- - 1 = ---------- - 1 = 12,333     */
/*                           16 * Baudrate       16 * 19200                  */
/*                                                                           */
/*       Das Register SPBRG muss also mit dem Wert 12 geladen werden.        */
/*                                                                           */
/* Anmerkung:                                                                */
/*   Die RS232-Ein- und Ausgaenge (TX bzw. Port RB2 und RX bzw. Port RB1) muessen    */
/*   nicht explizit (mit dem TRISB-Register) als Ein- bzw. Ausgang definiert werden!  */
/*****************************************************************************/
void INIT(void)
{
       TMR0 = 0;                           // Timer0 auf 0 voreinstellen
       OPTION = 0b.0000.0111;

       TRISB = 0;

       ZAEHLERISR10SEK = KONSTISR10SEK;

       // UART initialisieren
       TXSTA = 0b.0010.0100;               // Sender
       RCSTA = 0b.1001.0000;               // Empfaenger
       SPBRG = 12;                         // Baudrate (19200 Baud -> SPBRG = 12)
}


/*****************************************************************************/
/* AUSGABE:                                                                  */
/*                                                                           */
/* Aufgabe und Vorgehensweise:                                               */
/*   Die berechnete Temperatur und Luftfeuchtigkeit via RS232 ausgeben.      */
/*****************************************************************************/
void AUSGABE(void)
{
       RS232_SENDE_ZEICHEN(13);            // CR (Carrige Return)
       RS232_SENDE_ZEICHEN(10);            // LF (Line Feed)
       RS232_SENDE_ZEICHEN(10);            // LF (Line Feed)

       // Ausgabe von "Temperatur: "
       RS232_SENDE_ZEICHEN('T');
       RS232_SENDE_ZEICHEN(':');
       RS232_SENDE_ZEICHEN(' ');

       if (Sensordaten.temperatur_lin < 0)
       {
              // neg. Temperatur
              RS232_SENDE_ZEICHEN('-');
              Sensordaten.temperatur_lin = -Sensordaten.temperatur_lin;
       }
       else
       {
              // pos. Temperatur
              RS232_SENDE_ZEICHEN(' ');
       }

       // Ausgabe des Temperaturwertes
       AUSGABEBCD(Sensordaten.temperatur_lin,TEMPERATUR);
```

```
        RS232_SENDE_ZEICHEN(' ');
        RS232_SENDE_ZEICHEN('C');

        RS232_SENDE_ZEICHEN(13);            // CR (Carrige Return)
        RS232_SENDE_ZEICHEN(10);            // LF (Line Feed)

        // Ausgabe der Feuchte
        RS232_SENDE_ZEICHEN('F');
        RS232_SENDE_ZEICHEN(':');
        RS232_SENDE_ZEICHEN(' ');

        // Ausgabe des unkompensierten Feuchtewertes
        AUSGABEBCD(Sensordaten.feuchte_lin,FEUCHTE);
        RS232_SENDE_ZEICHEN(' ');
        RS232_SENDE_ZEICHEN('%');

        // Ausgabe des kompensierten Feuchtewertes
        RS232_SENDE_ZEICHEN(13);            // CR (Carrige Return)
        RS232_SENDE_ZEICHEN(10);            // LF (Line Feed)
        RS232_SENDE_ZEICHEN('F');
        RS232_SENDE_ZEICHEN('(');
        RS232_SENDE_ZEICHEN('k');
        RS232_SENDE_ZEICHEN(')');
        RS232_SENDE_ZEICHEN(':');
        RS232_SENDE_ZEICHEN(' ');

        AUSGABEBCD(Sensordaten.feuchte_komp,FEUCHTE);
        RS232_SENDE_ZEICHEN(' ');
        RS232_SENDE_ZEICHEN('%');
}


/*****************************************************************************/
/* AUSGABEBCD:                                                               */
/*                                                                           */
/* Aufgabe:                                                                  */
/*   Den uebergeben Wert in BCD-Form umwandeln und via RS232 ausgeben        */
/*****************************************************************************/
void AUSGABEBCD(unsigned long Wert, unsigned char mode)
{
        unsigned char  help;
        unsigned long  help_modulo;


        help_modulo = Wert % 10000;
        help_modulo = help_modulo % 1000;
                                            // "Hunderterstelle" ermitteln
        if ((mode == TEMPERATUR) || (Wert == 100))
        {
                help = (char)(help_modulo/100);     // in eine ASCII-Zeichen umwandeln  (um 30h
                help += 0x30;                       // erhoehen) und via RS232 ausgeben, wenn
                RS232_SENDE_ZEICHEN(help);          // es sich um einen Temperaturwert handelt
        }                                           // oder bei einer Feuchte von 100 %
        help_modulo = help_modulo % 100;

        help = (char)(help_modulo/10);              // "Zehnerstelle" ermitteln
        help += 0x30;                               // in eine ASCII-Zeichen umwandeln  (um 30h
        RS232_SENDE_ZEICHEN(help);                  // erhoehen) und via RS232 ausgeben
        help_modulo = help_modulo % 10;
        help_modulo += 0x30;

        if (mode == TEMPERATUR) RS232_SENDE_ZEICHEN('.'); // Einen Dezimalpunkt ausgeben, wenn
                                                    // es sich um einen Temperaturwert handelt
        RS232_SENDE_ZEICHEN(help_modulo);           // Als letztes die "Einerstelle" ausgeben
}


/***************** Routinen fuer den Feuchtigkeitssensor ******************************/

/*****************************************************************************/
/* FEUCHTE_START:                                                            */
/*                                                                           */
/* Aufgabe:                                                                  */
/*   Startbedingung fuer die Kommunikation mit dem Feuchtigkeitssensor erzeugen */
/*                                                                           */
/*   Die Startbedingung ist wie folgt definiert:                             */
/*                  ____         _______                                     */
/*     DATA:           |_______|                                            */
/*                   ___     ___                                             */
```

```
/*     SCK:        __|   |___|   |_____                                                */
/*****************************************************************************************/
void FEUCHTE_START(void)
{
       DATA = 1;
       SCK  = 0;
       SCK  = 1;
       DATA  = 0;
       SCK  = 0;

       #asm
       nop
       #endasm

       SCK  = 1;
       DATA  = 1;
       SCK  = 0;
}


/*****************************************************************************************/
/* FEUCHTE_CONNECTIONRESET:                                                              */
/*                                                                                       */
/* Aufgabe:                                                                              */
/*   9 SCK-Zyklen, waehrend DATA = 1, anschliessend eine Startbedingung                  */
/*                                                                                       */
/*             _____________________________________________      ______                 */
/*     DATA:                                                |_____|                     */
/*               _   _   _   _   _   _   _   _   _   _        __    __                  */
/*     SCK:    _| |_| |_| |_| |_| |_| |_| |_| |_| |_| |____|  |__|  |_____             */
/*****************************************************************************************/
void FEUCHTE_CONNECTIONRESET(void)
{
       unsigned char i;

       DATA = 1;                           // Initialisieren
       SCK = 1;
       for(i=0;i<9;i++)                    // 9 Takt-Zyklen
       {
              SCK = 1;
              SCK = 0;
       }
       FEUCHTE_START();                    // Startbedingung
}


/*****************************************************************************************/
/* FEUCHTE_SCHREIBEN:                                                                    */
/*                                                                                       */
/* Aufgabe:                                                                              */
/*   Ein Byte (also eine Anweisung) an den Sensor uebergeben und die Bestaetigung vom    */
/*   Sensor empfangen.                                                                   */
/*                                                                                       */
/* Vorgehensweise:                                                                       */
/*   + Register fehler loeschen                                                          */
/*   + Mit einer Schleife jedes einzelne Bit des zu uebergebenden Bytes maskieren und    */
/*     das Ergebnis auf die DATA-Leitung geben. Anschliessend einen Taktimpuls erzeugen  */
/*   + Nun die Bestaetigung (Acknowledge) empfangen:                                     */
/*     + DATA = 1                                                                        */
/*     + SCK  = 1 (steigende Flanke der Takt-Leitung)                                    */
/*     + Das TRIS_DATA-Flag muss nun gesetzt werden, da der DATA-Pin auf Eingang umge-   */
/*       schaltet werden muss.                                                           */
/*     + Nun die DATA-Leitung einlesen und in fehler sichern                             */
/*     + SCK  = 0 (fallende Flanke der Takt-Leitung)                                     */
/*     + Das TRIS_DATA-Flag muss wieder geloescht werden, da der DATA-Pin nun wieder     */
/*       als Ausgang dienen soll.                                                        */
/*****************************************************************************************/
char FEUCHTE_SCHREIBEN(unsigned char value)
{
       unsigned char i;
       unsigned char fehler;

       fehler = 0;                         // Register fehler loeschen

       for(i=0x80;i>0;i/=2)                // Mit einer Schleife jedes einzelne Bit des
       {
              if (i & value) DATA = 1;     // ...zu uebergebenden Bytes maskieren und das
              else DATA = 0;               // Ergebnis auf die DATA-Leitung geben.
                                           // Anschliessend einen Taktimpuls erzeugen
```

```
                SCK = 1;                        // Taktimpuls

                #asm
                nop
                #endasm

                SCK = 0;
        }
        DATA = 1;
        SCK = 1;                                // steigende Flanke der Takt-Leitung
        TRIS_DATA = 1;                          // DATA-Pin auf Eingang umschalten
        fehler = DATA;                          // DATA einlesen und in fehler sichern
        SCK = 0;                                // fallende Flanke der Takt-Leitung
        TRIS_DATA = 0;                          // DATA-Pin auf Ausgang zurueckschalten
        return(fehler);
}


/*****************************************************************************************/
/* FEUCHTE_LESEN:                                                                        */
/*                                                                                       */
/* Aufgabe:                                                                              */
/*   Ein Byte vom Sensor lesen und eine Bestaetigung zurueckgeben, wenn ack=1            */
/*                                                                                       */
/* Vorgehensweise:                                                                       */
/*   + DATA = 1                                                                          */
/*   + Das TRIS_DATA-Flag muss nun gesetzt werden, da der DATA-Pin auf Eingang umge-     */
/*     schaltet werden muss.                                                             */
/*   + Mit einer Schleife das 8-Bit Datenwort bitweise einlesen                          */
/*   + Das TRIS_DATA-Flag muss wieder geloescht werden, da der DATA-Pin nun wieder als   */
/*     Ausgang dienen soll.                                                              */
/*   + Nun die Bestaetigung (Acknowledge) senden                                         */
/*   + DATA = 1                                                                          */
/*****************************************************************************************/
char FEUCHTE_LESEN(unsigned char ack)
{
        unsigned char i;
        unsigned char val = 0;

        DATA = 1;
        TRIS_DATA = 1;                          // DATA-Pin auf Eingang umschalten
        for(i=0x80;i>0;i/=2)                    // Mit einer Schleife das 8-Bit Datenwort
                                                // bitweise einlesen
        {
                SCK = 1;
                if (DATA) val = (val | i);
                SCK = 0;
        }
        TRIS_DATA = 0;                          // DATA-Pin auf Ausgang zurueckschalten
        DATA = !ack;                            // Bestaetigung (Acknowledge) senden
        SCK = 1;

        #asm
        nop
        #endasm

        SCK = 0;
        DATA = 1;
        return (val);
}


/*****************************************************************************************/
/* FEUCHTE_STATUSLESEN:                                                                  */
/*                                                                                       */
/* Aufgabe:                                                                              */
/*   Statusregister des Sensors und Checksumme auslesen                                  */
/*                                                                                       */
/* Vorgehensweise:                                                                       */
/*   + Startbedingung mit dem Unterprogramm FEUCHTE_START() erzeugen                     */
/*   + Anweisung zum Auslesen des Sensor-Statusregister (mit dem Unterprogramm           */
/*     FEUCHTE_SCHREIBEN() mit dem Parameter STATUS_LESEN (=0x07))                       */
/*   + Status-Register mit dem Unterprogramm FEUCHTE_LESEN() einlesen und im Uebergabe-  */
/*     parameter *p_value sichern                                                        */
/*   + Checksumme mit dem Unterprogramm FEUCHTE_LESEN() einlesen und im Uebergabe-       */
/*     parameter *p_checksum sichern.                                                    */
/*****************************************************************************************/
char FEUCHTE_STATUSLESEN(unsigned char *p_value, unsigned *p_checksum)
{
```

```
        unsigned char fehler = 0;
        char temp;

        FEUCHTE_START();                    // Startbedingung

        fehler = FEUCHTE_SCHREIBEN(STATUS_LESEN); //Anweisung zum Auslesen des Sensor-
                                            // Statusregister

        temp = FEUCHTE_LESEN(ACK);          // Status-Register einlesen und in
        *p_value = temp;                    // *p_value sichern

        temp = FEUCHTE_LESEN(noACK);        // Checksumme einlesen und in
        *p_checksum = temp;                 // *p_checksum sichern

        return (fehler);
}


/*****************************************************************************************/
/* FEUCHTE_STATUSSCHREIBEN:                                                              */
/*                                                                                       */
/* Aufgabe:                                                                              */
/*   Statusregister des Sensors beschreiben                                              */
/*                                                                                       */
/* Vorgehensweise:                                                                       */
/*   + Startbedingung mit dem Unterprogramm FEUCHTE_START() erzeugen                     */
/*   + Anweisung zum Beschreiben des Sensor-Statusregister (mit dem Unterprogramm        */
/*     FEUCHTE_SCHREIBEN() mit dem Parameter STATUS_SCHREIBEN (=0x06))                   */
/*   + Den Wert p_value mit dem Unterprogramm FEUCHTE_SCHREIBEN() in das Statusregister  */
/*     des Sensors schreiben                                                             */
/*****************************************************************************************/
char FEUCHTE_STATUSSCHREIBEN(unsigned char p_value)
{
        unsigned char fehler = 0;

        FEUCHTE_START();

        fehler += FEUCHTE_SCHREIBEN(STATUS_SCHREIBEN);
        fehler += FEUCHTE_SCHREIBEN(p_value);

        return (fehler);
}


/*****************************************************************************************/
/* FEUCHTE_MESSUNG:                                                                      */
/*                                                                                       */
/* Aufgabe:                                                                              */
/*   Einen Feuchtigkeits- oder Temperatur-Messzyklus starten                             */
/*                                                                                       */
/* Vorgehensweise:                                                                       */
/*   + Startbedingung erzeugen                                                           */
/*   + Je nach Mode (Temperaturmessung oder Feuchtemessung) den Wert 03h (fuer eine      */
/*     Temperaturmessung) oder 05h (fuer eine Feuchtemessung) mit dem Unterprogramm      */
/*     FEUCHTE_SCHREIBEN dem Sensor uebergeben, und so diese Messung starten.            */
/*   + Die DATA-Leitung auf Eingang umschalten und warten bis der Sensor diese DATA-     */
/*     Leitung auf Low legt, als Zeichen, dass er mit der Messung fertig ist und die     */
/*     Messwerte abholbereit sind.                                                       */
/*   + Nun das High-Byte, Low-Byte und die Checksumme mit Hilfe des Unterprogramms       */
/*     FEUCHTE_LESEN einlesen.                                                           */
/*****************************************************************************************/
char FEUCHTE_MESSUNG(unsigned char mode)
{
        unsigned char fehler;
        unsigned int i;
        unsigned char temp;


        fehler = 0;

        FEUCHTE_START();                    // Startbedingung erzeugen

        switch(mode)                        // je nach Mode entweder eine Temperatur
        {                                   // oder eine Feuchtemessung starten
                case TEMPERATUR:      fehler += FEUCHTE_SCHREIBEN(MESSUNG_TEMPERATUR); break;
                case FEUCHTE:         fehler += FEUCHTE_SCHREIBEN(MESSUNG_FEUCHTE); break;
                default:      break;
        }
```

```
        TRIS_DATA = 1;                      // DATA-Leitung auf Eingang umschalten

        for (i=0;i<65535;i++) if (DATA == 0) break;
        if (DATA) fehler += 1;              // Time-out! -> Fehlerflag setzen

        TRIS_DATA = 0;                      // DATA-Leitung auf Ausgang umschalten

        temp = FEUCHTE_LESEN(ACK);          // High-Byte einlesen und je nach mode
        switch(mode)                        // in temperatur_hi oder feuchte_hi laden
        {
               case TEMPERATUR:      Sensordaten.temperatur_hi = temp; break;
               case FEUCHTE:         Sensordaten.feuchte_hi = temp; break;
               default:       break;
        }

        temp = FEUCHTE_LESEN(noACK);        // Low-Byte einlesen und in
        switch(mode)                        // in temperatur_lo oder feuchte_lo laden
        {
               case TEMPERATUR:      Sensordaten.temperatur_lo = temp; break;
               case FEUCHTE:         Sensordaten.feuchte_lo = temp; break;
               default:       break;
        }

        return (fehler);
}


/*****************************************************************************************/
/* FEUCHTE_LINEARISIERUNG:                                                               */
/*                                                                                       */
/* Aufgabe:                                                                              */
/*   Aus den vom Sensor empfangenen Roh-Daten die Temperatur und die Feuchtigkeit        */
/*   berechnen. Anschliessend muss der Feuchtewert linearisiert und mit der Temperatur   */
/*   kompensiert werden                                                                  */
/*                                                                                       */
/* Formeln: (8/12-Bit-Mode)                                                              */
/*   +  Temperatur = 0.04 * (256*temperatur_hi + temperatur_lo) - 40                     */
/*   +  Feuchte:                                                                         */
/*       0 <= feuchte_lo <= 107:     Feuchte = (143 * feuchte_lo - 512) / 256            */
/*     108 <= feuchte_lo <= 255:     Feuchte = (111 * feuchte_lo + 2893) / 256           */
/*             Feuchte(komp) = (Temperatur-25)*(0.01 + 0.00128*rH) + Feuchte             */
/*                                                                                       */
/* Formeln: (12/14-Bit-Mode)                                                             */
/*   +  Temperatur = 0.01 * (temperatur_hi + 256*temperatur_lo) - 40                     */
/*   +  Feuchte:                                                                         */
/*     rF = 256*feuchte_hi + feuchte_lo                                                  */
/*     Feuchte = -4 + 0.0405 * rH - 0.0000028 * rH * rH                                  */
/*     Feuchte(komp) = (Temperatur-25)*(0.01 + 0.00008*rH) + Feuchte                     */
/*****************************************************************************************/
void FEUCHTE_LINEARISIERUNG(void)
{
        unsigned long  feuchte;
        unsigned long  feuchte_lin1;
        long                   feuchte_komp;
        long                   feuchte_komp1;
        unsigned long  feuchte_komp2;


        // Berechnung der Temperatur
        Sensordaten.temperatur_lin = (long)(256 * Sensordaten.temperatur_hi);
        Sensordaten.temperatur_lin = Sensordaten.temperatur_lin + Sensordaten.temperatur_lo;

        // Temperaturwert linearisieren
        Sensordaten.temperatur_lin = (long)(Sensordaten.temperatur_lin / 10);
        Sensordaten.temperatur_lin = Sensordaten.temperatur_lin - 400;

        // Feuchtewert berechnen
        feuchte = (long)(256 * Sensordaten.feuchte_hi);
        feuchte = feuchte + Sensordaten.feuchte_lo;

        // Feuchtewert linearisieren
        Sensordaten.feuchte_lin = feuchte / 10;
        Sensordaten.feuchte_lin = Sensordaten.feuchte_lin * 4;
        Sensordaten.feuchte_lin = Sensordaten.feuchte_lin / 10;
        Sensordaten.feuchte_lin = Sensordaten.feuchte_lin - 4;

        feuchte_lin1 = Sensordaten.feuchte_lin / 100;
        feuchte_lin1 = feuchte_lin1 * feuchte_lin1;
        feuchte_lin1 = feuchte_lin1 * 28;
```

```
        feuchte_lin1 = feuchte_lin1 / 1000;

        Sensordaten.feuchte_lin = Sensordaten.feuchte_lin - feuchte_lin1;

        // Feuchtewert kompensieren
        feuchte_komp1 = (long)(Sensordaten.temperatur_lin / 10);
        feuchte_komp1 = feuchte_komp1 - 25;

        feuchte_komp2 = (long)(feuchte / 100);
        feuchte_komp2 = feuchte_komp2 * 8;
        feuchte_komp2 = feuchte_komp2 + 10;

        feuchte_komp = feuchte_komp1 * feuchte_komp2;
        feuchte_komp = feuchte_komp / 1000;
        Sensordaten.feuchte_komp = feuchte_komp + Sensordaten.feuchte_lin;

        if (Sensordaten.feuchte_komp > 100)
        {
                Sensordaten.feuchte_komp = 100;
        }
}


/***************** RS232 Routinen *****************************************************/

/*******************************************************************************************/
/* RS232_SENDE_ZEICHEN:                                                                    */
/*                                                                                         */
/* Aufgabe und Vorgehensweise:                                                             */
/*   + Warten, bis der Sendebuffer des UART gelehrt ist (Dies zeigt das Flag TRMT im       */
/*     Register TXSTA an)                                                                  */
/*   + Das zu sendende Zeichen (value) in das Register TXREG schreiben                     */
/*   + Fertig! - Den Rest uebernimmt die Hardware des UART-Moduls                          */
/*******************************************************************************************/
void RS232_SENDE_ZEICHEN(unsigned char value)
{
                while(!TRMT);

                TXREG = value;
}
```

7.3. Anmerkungen zur Software

Die Software besteht im Wesentlichen aus folgenden Programmteilen:

- kurzes Hauptprogramm
- eine kurze Interrupt-Service-Routine (kurz ISR)
- Unterprogramm INIT (zur Initialisierung des Mikrocontrollers (siehe auch Abschnitt 6.2)
- 8 Unterprogramme zur Kommunikation mit dem Sensor (entsprechende Abschnitt 6.3)
- 2 Unterprogramme zur Ausgabe der ermittelten Temperatur und Feuchtigkeit
- 1 Unterprogramm zur Datenübertragung an einem PC via RS232

ISR:
Die ISR (`interrupt InterruptRutine(void)`) wird alle 64ms aufgerufen und besitzt „nur“ die Aufgabe eine 10-Sekunden Zeitbasis zu erzeugen, indem sie ein Botschaftsflags für das Hauptprogramm erzeugt. Damit eine Zeit von einer Sekunde entsteht, muss die ISR 156-mal aufgerufen werden (156 x 64ms = 9984ms also ca. 10 Sekunde). Bei jedem ISR-Aufruf muss also ein Zählregister um 1 vermindert werden. Besitzt es danach den Wert 0, so sind 10 Sekunde (genau 9,984 Sekunden) vergangen. Nun wird das Botschaftsflag FLAG10SEK im Register FLAGSISRHP gesetzt, und das Zählregister muss mit dem Wert 156 neu geladen werden. Der Wert 156 wird hier durch die Konstante KONSTISR10SEK ersetzt.

Die ISR wird, wie schon mehrmals erwähnt, alle 64ms aufgerufen Diese 64ms ergeben sich folgendermaßen: TMR0 wird mit dem Wert 0 geladen – es dauert also 256 Taktzyklen bis das Register wieder den Wert 0 besitzt, der Vorteiler (VT) besitzt hier ebenfalls den Wert 256 (vgl. Unterprogramm INIT, Abschnitt. 6.2). Der Taktzyklus ergibt sich aus dem verwendeten Quarz (X1). Dieser ist bei der PIC-Familie wie folgt definiert:

$$\frac{4}{f_{Quarz}}$$

Daraus ergibt sich folgender Zusammenhang:

$$ISRAUFRUF \quad [\mu s] = \frac{4 \bullet 256 \bullet VT}{f_{Quarz}\ [MHz]} = \frac{4 \bullet 256 \bullet 256}{4{,}096} = 6\,4000$$

Also ein ISR-Aufruf alle 64000µs, was gleichbedeutend mit 64ms ist.

Achtung:
Die ISR muss nach dem Kompelieren an einer ganz bestimmten Stelle im Programmspeicher des PIC-Mikrocontroller stehen. Nämlich an der Adresse 0004h. Dies wird durch die Präprozessoranweisung `#pragma origin 4` erreicht.

Hauptprogramm:
Das Hauptprogramm `void main(void)` befindet sich nach der Initialisierung (mit dem Unterprogramm `void INIT(void)` und der Interrupt-Freigabe in einer Endlosschleife. Diese Schleife besitzt die Aufgabe ständig dass Botschaftsflag (FLAG10SEK) abzufragen. Ist dieses Botschaftsflag gesetzt, so muss das Hauptprogramm zunächst eine Feuchtigkeitsmessung und anschließend eine Temperaturmessung mit dem Unterprogramm `char FEUCHTE_MESSUNG()` gemäß Abschnitt 6.3.7. durchführen. Waren beide Messungen erfolgreich (fehler = 0) aus dem vom Sensor erhaltenen Daten die Temperatur und die Feuchtigkeit mit dem Unterprogramm `void FEUCHTE_LINEARISIERUNG(void)` gemäß Abschnitt 6.3.8 berechnen, bzw. linearisieren und kompensieren () und anschließend mit dem Unterprogramm `void AUSGABE(void)` am PC ausgeben.
Bei einer fehlerhaften Messung (fehler ist ungleich 0) den Feuchtigkeitssensor mit dem Unterprogramm `void FEUCHTE_CONNECTIONRESET(void)` gemäß Abschnitt 6.3.2. neu initialisieren.
Das Botschaftsflag (FLAG10SEK) wird von der Interrupt Service Routine (ISR) alle 10 Sekunden (siehe oben).

Unterprogramm INIT:
Das Unterprogramm `void INIT(void)` dient zur Initialisierung des PIC-Mikrocontroller. Hier wird unter anderem der Timer eingestellt. Weiters die RS-232-Schnittstelle für die Kommunikation mit dem PC, und das Zählregister für den 10-Sekunden-Takt.

Unterprogramme AUSGABE, AUSGABEBCD und RS232_SENDE_ZEICHEN:
Das Unterprogramm `void AUSGABE()` dient zur Ausgabe der Temperatur und Luftfeuchtigkeit am PC. Dazu müssen diese Werte in die so genannte BCD-Form gebracht werden. Diese Aufgabe übernimmt das Unterprogramm `void AUSGABEBCD()`. Für das eigentliche Übertragen zum PC via RS232 ist das Unterprogramm `void RS232_SENDE_ZEICHEN()` zuständig.

Weitere Anmerkungen:

- Für die Berechnung der Feuchtigkeit bzw. Temperatur ist die Mathematik-Bibliothek (math16.h) notwendig. Diese wird mit der Anweisung `#include <MATH16.H>` eingebunden. Normalerweise werden solche so genannten Include-Dateien am Programmbeginn eingebunden. Diese war aber hier, aus welchem Grund auch immer, nicht möglich. So dass es zu Beginn der Unterprogramme eingefügt wurde.
- Die Präprozessoranweisung `#pragma library 1` sorgt dafür, dass alle Unterprogramme die nicht benötigt werden, durch den Kompiler nicht übersetzt werden und daher auch nicht im Programmspeicher des PIC-Mikrocontroller stehen. Dies ist gerade bei Verwendung von math16.h sinnvoll. Da in der Regel nicht alle von math16.h zur Verfügung gestellten Unterprogramme benötigt werden.

7.4. PC-Programm: HyperTerminal

Zur Demonstration wird das PC-Programm „HyperTerminal“ verwendet. Dieses Programm sollte bei jedem modernen Windows-PC installiert sein und wird bei Windows XP wie folgt aufgerufen: Taste „Start“ in der Taskleiste -> Alle Programme -> Zubehör -> Kommunikation -> HyperTerminal.

Nach dem Start von HyperTerminal müssen Sie zuerst eine neue Verbindung einrichten.

Dazu „Datei“ -> „Neu“ anklicken. Es öffnet sich folgendes Fenster:

Bild 7.2.: HyperTerminal (Schritt 1)

Einen Namen für diese neue Verbindung eingeben: z.B. Feuchtesensor SHTxx.
Optional kann ein Symbol ausgewählt werden.

Taste „OK“

Bild 7.3.: HyperTerminal (Schritt 2)

Den seriellen Port auswählen. Z.B. COM2

Taste „OK“

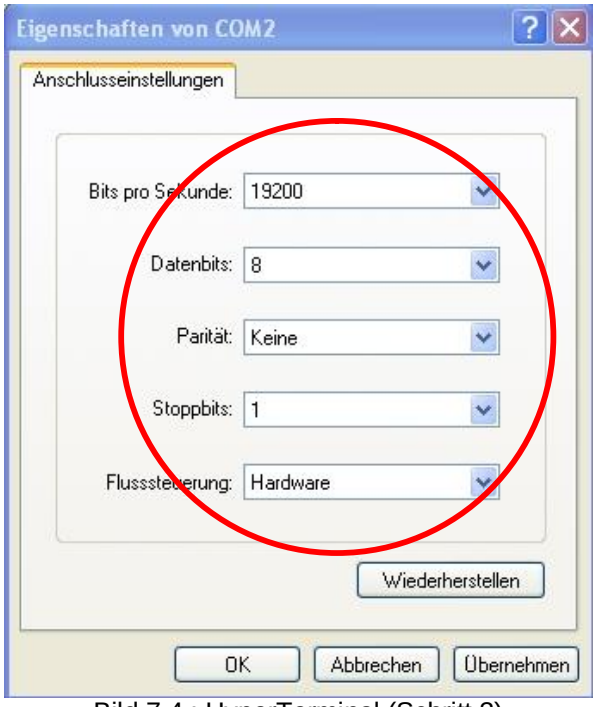

Bild 7.4.: HyperTerminal (Schritt 3)

Den ausgewählten COM-Port (hier COM2) nach Bild 7.3 konfigurieren

Taste „OK“

Wenn alles richtig eingestellt und angeschlossen wurde sollten innerhalb von 10 Sekunden die Temperatur und die Feuchtigkeit so wie in Bild 7.5 erscheinen.

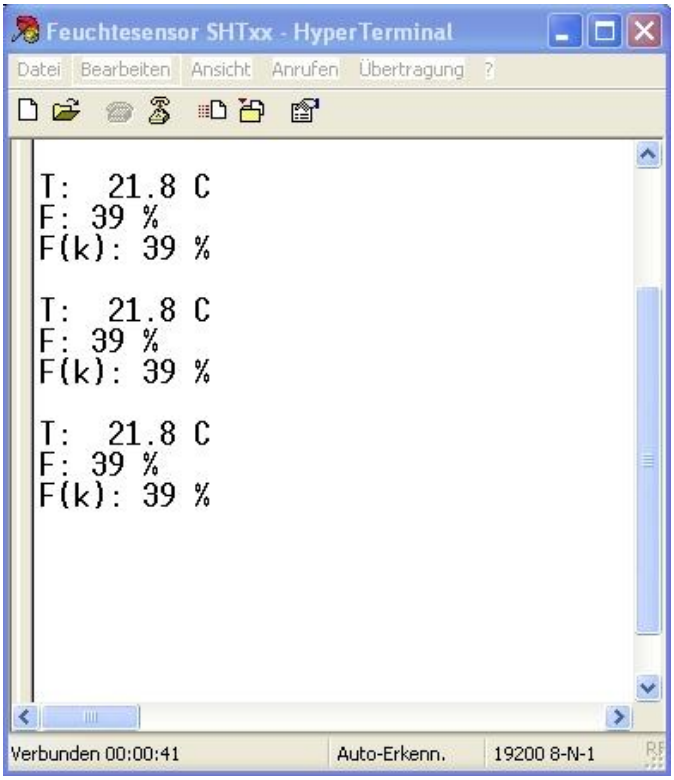

Bild 7.5 HyperTerminal (Schritt 4)

8. Quellen

- Datenblatt und Anwenderhinweise des Feuchtigkeitssensors (www.sensirion.com)

Anhang A: „Wohlfühlzusammenhang“ zwischen Temperatur und Luftfeuchtigkeit

Temperatur	Feuchte									
	20 %	30 %	35 %	40 %	45 %	50 %	55 %	60 %	65 %	70 %
< 18 °	L	L	L	L	L	L	L	L	L	L
18 – 19,9°	L	L	L	K	K	K	K	K	K	L
20 – 21,9 °	L	L	L	K	J	J	J	J	K	L
22 – 23,9 °	L	L	K	J	J	J	J	K	L	L
24 – 25,9 °	L	K	J	J	J	J	K	L	L	L
26 – 27,9 °	L	K	K	K	K	K	K	L	L	L
> 28 °	L	L	L	L	L	L	L	L	L	L