

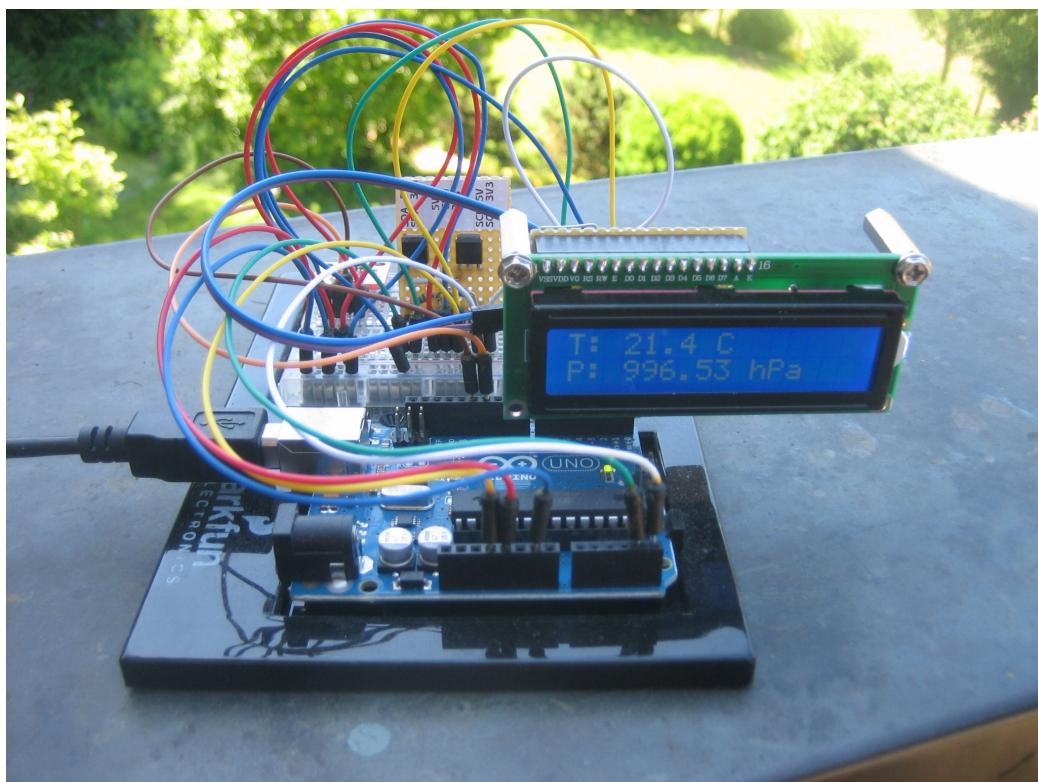
# Ausarbeitung

## Projektarbeit

Philip Priss

Matr. Nr. 15 20 80 13

Projekt: Wetterstation mit dem Arduino



---

# Inhaltsverzeichnis

Überblick.....	3
1. – Aufgabenstellung.....	4
2. – Problemanalyse.....	4
2.1 – Technik des Arduino Uno.....	5
2.1.1 – Was macht den Arduino zum Arduino?.....	6
2.1.2 – Verwendete Hardwareschnittstellen.....	7
2.1.2.1 – U(S)ART.....	7
2.1.2.2 – I <sup>2</sup> C.....	9
2.2 – Bosch BMP085.....	11
2.2.1 – Sensorspezifikationen.....	11
2.2.2. – Messprinzip / Messwertverarbeitung.....	11
2.2.3 – Kommunikationsschnittstelle.....	12
2.3 – Sensirion SHT015.....	14
2.3.1 – Sensorspezifikationen.....	14
2.3.2 – Messprinzip / Messwertverarbeitung.....	14
2.3.3 – Kommunikationsschnittstelle.....	15
2.4 – LCD.....	16
3 – Lösungsentwurf.....	17
3.1 – Schaltplan.....	17
3.2 – Programmablaufplan.....	18
3.3 – SHTPS – Protokoll.....	19
4. – Auswertung Messergebnisse.....	19
5 – Zusammenfassung und Ausblick.....	20
Anhang A – Quelltextauszug.....	21
Anhang B – SHTPS Protokoll Spezifikation.....	23
Anhang C – Abbildungsverzeichnis.....	24
Anhang D – Quellenverzeichnis.....	25

---

# Überblick

Diese Ausarbeitung befasst sich mit der Realisierung einer Physical Computing Plattform am Beispiel einer Wetterstation mit Hilfe der Arduino Plattform. Diese ist im Rahmen der Projektarbeit im Sommersemester 2014 entstanden. In dieser Ausarbeitung werden die Themen Arduino, Sensorik und Kommunikationsbusse behandelt. Im speziellen wird auf die Details der Arduino Hardware sowie der eingesetzten Sensoren eingegangen. Dies ist der Bosch BMP 085 und der Sensirion SHT15. Auch die verwendeten Kommunikationsbusse und -protokolle werden im Detail vorgestellt. Hierbei handelt es sich um den I<sup>2</sup>C Bus für den BMP085, ein proprietäres Protokoll von Sensirion für den SHT15 und das hier für entwickelte SHTPS Protokoll. Im Anschluss gibt es noch einen Ausblick, wie das Projekt in Zukunft weitergeführt werden kann.

---

## **1. – Aufgabenstellung**

Mit dem Mikrocontroller Board Arduino Uno soll eine Wetterstation realisiert werden. Diese soll die Daten auf den PC übermitteln und zusätzlich auch auf einem LC Display anzeigen. Ein Display dient zu einem PC autarken Betrieb. Zur Erfassung der Wetterdaten stehen folgende Sensoren zur Verfügung: Bosch BMP085 für Temperatur und Luftdruck und Sensirion SHT15 für Luftfeuchte. Um die Daten an den PC zu übermitteln enthält der Arduino eine serielle Schnittstelle. Für eine direkte Anzeige der Messdaten wird ein 16x2 LCD benutzt.

Besonders soll dabei ein Verständnis für die Funktionsweise der Hardware erlangt werden. Dies beinhaltet sowohl die physikalische Funktionsweise der Sensorik als auch die elektrischen Signale und Protokolle der verwendeten Bussysteme.

## **2. – Problemanalyse**

Für die Realisierung der Wetterstation gibt es nur die einzelnen Komponenten wie die Sensoren und den Arduino. Es ist noch keine vorgefertigte Hardware vorhanden. Die gesamte Hardware muss also zuerst geplant und zusammengebaut werden, bevor diese programmiert werden kann. Dies bedeutet eine Programmierung auf unterster Hardwareebene. Darin eingeschlossen ist auch die Kommunikation mit den Sensoren über das entsprechende Kommunikationsprotokoll. Zum Teil muss das jeweilige Protokoll erst noch implementiert werden. Aus dem Grund muss zuerst die Ansteuerung der Sensoren über die entsprechenden Datenblätter in Erfahrung gebracht werden. Diese muss dann wiederum in den Arduino implementiert werden. Auch wie die Sensoren anzuschließen sind und welche Versorgungsspannungen diese benötigen kann in diesem Schritt gut geklärt werden.

---

## 2.1 – Technik des Arduino Uno

Der Arduino besteht im Kern aus einem ATMega328 der Firma Atmel als gesockelter DIP. Es handelt sich dabei um einen 8 Bit Mikrocontroller mit folgenden Kenndaten:

- 8-Bit Controller bis 20 MHz
- 32 KB Programm Flash (min. 10.000 Schreib / Lese Zyklen)
- 2 KB RAM
- 1 KB EEPROM(min. 100.000 Schreib / Lese Zyklen)
- In – System Programable (ISP) und Self – Programable
- 32 programmierbare I/O Pins
- Interner oder externe Oszillator
- TWI (I<sup>2</sup>C), SPI und USART Schnittstelle
- Programmierbare Lock Bits zum Programmschutz
- 6 Channel PWM, 6 Channel ADC

Zusätzlich befindet sich auf dem Board noch ein ATMega8 der die USB Kommunikation übernimmt. Dieser ist für den Benutzer nicht ohne weiteres programmierbar. Der Controller wird benötigt, um den USB zu Seriell Wandler des Arduinos zu emulieren, der zur Programmierung und PC Kommunikation benötigt wird. Um die nötigen Versorgungsspannungen zu erzeugen, gibt es jeweils einen linearen 5V und 3,3V Spannungswandler. Der 5V Regler wird nur benutzt, wenn nicht der USB Anschluss sondern eine externe Spannungsversorgung angeschlossen ist. Normalerweise wird die gesamte Hardware über USB versorgt. Des weiteren gibt es noch Status LED's für die Spannungsversorgung und Serielle Kommunikation. Auch eine LED die direkt über Pin 13 des Arduino gesteuert werden kann ist vorhanden.

---

### **2.1.1 – Was macht den Arduino zum Arduino?**

Das was den Arduino aus macht liegt weniger in der verwendeten Hardware, als in der dazugehörigen Software. Die ATMega328 Controller lassen sich auch in C und Assembler programmieren. Auch bringt das direkte Programmieren sogar noch Vorteile gegenüber der Arduino Umgebung. Ein nicht zu vernachlässigender Vorteil ist das Hardware debugging über JTAG oder DebugWire Schnittstelle. Dies ist beim Arduino allerdings nicht möglich.

Der große Vorteil, wenn ein Arduino benutzt wird ist, dass sehr viele Funktionen bereits vorgefertigt sind und einfach nur noch eingebunden bzw. benutzt werden müssen. Ein gutes Beispiel hierfür ist die Benutzung der USART Schnittstelle. Wenn diese ohne Arduino Umgebung genutzt werden sollte, muss zuerst das Datenblatt des Controllers gelesen werden. Darauf hin können die entsprechenden Register konfiguriert werden. Dies sind alleine bei der USART fünf Stück. Zusätzlich muss die Baud Rate berechnet werden. Alle benötigten Funktionen für z.B. senden und empfangen müssen selber geschrieben werden usw. . Bei dem Arduino genügt der Befehl *Serial.begin(9600);* und die USART ist passend konfiguriert. Daraus ergibt sich eine schnelle und unkomplizierte Entwicklung einer lauffähigen Software.

Eine weitere Eigenschaft ist, dass für das Programmieren des Controllers kein spezielles Programmiergerät, wie ein AVR ISP MKII, benutzt werden muss. Dies ist möglich, da bereits ein kleines Programm, ein sogenannter Bootloader, auf dem ATMega238 installiert ist. Dieser kümmert sich um die Programmierung des Controller. Das sich der Controller selber programmieren kann, liegt an der Eigenschaft des Self – Programmable. Ein nicht zu unterschlagender Nachteil ist allerdings, dass der Bootloader selber Platz auf dem ATMega328 benötigt und somit der effektive Programmspeicher kleiner als die angegebenen 32KB ist.

Sollte es nun aber zu dem Problem kommen, dass das Programm zu groß für den Controller ist, lässt sich der Bootloader löschen und das in der Arduino IDE geschriebene Programm direkt auf den ATMega328 schreiben. Voraussetzung hierfür ist aber die nötige Fachkenntnis im Umgang mit dem Controllern und ein entsprechender In – System Programmer.

---

## 2.1.2 – Verwendete Hardwareschnittstellen

### 2.1.2.1 – U(S)ART

USART ist die Kurzform für **U**niversal **S**ynchronous and **A**synchronous serial **R**eceiver and **T**ransmitter. Es ist die universelle, serielle Schnittstelle des ATMega328. Im UART Modus ist sie Protokoll kompatibel mit der RS-232 Schnittstelle am PC.

- ATMega UART: TTL Pegel
  - Logisch 0: 0V
  - Logisch 1: 5V
- RS-232:
  - Logisch 0: -15V
  - Logisch 1: 15V

Somit darf die UART nur über einen Pegelwandler (z.B. MAX232) an einen PC angeschlossen werden! Diese Gefahr tritt bei dem Arduino nicht auf, da hier ein UART – zu – Seriell – Wandler über USB emuliert wird.

Da die UART / RS-232 verschiedene Einstellungen vorsieht, wird im weiteren von folgenden Bedingungen ausgegangen, die so auch sehr häufig anzutreffen sind und in diesem Projekt verwendet werden:

- Baudrate: 9600, 8 Byte
- 1 Startbit, 1 Stopbit, kein Handshake

Die UART ist eine asynchrone Schnittstelle. Das bedeutet, dass es keine Taktleitung gibt um die eingehenden Daten zu synchronisieren. Aus diesem Grund geschieht die Übertragung immer Byteweise. Im Ruhezustand hat die Datenleitung einen High-Pegel. Um nun eine Übertragung zu starten, wird die Datenleitung für eine Bitzeit auf den Low-Pegel gezogen. Dies ist das sogenannte Startbit, dass vor jedem Byte gesendet wird. Es sorgt für eine Synchronisation des Empfängertaktes auf den Bitstrom. Wenn es diese Synchronisation nicht bei jedem Byte wiederholen würde, würde der Empfängertakt nach kürzester Zeit so unsynchron zu dem Bitstrom sein, dass es zu einer fehlerhaften Dekodierung kommt. Auf das Startbit folgen die 8 Datenbits. Um nun die Übertragung abzuschließen, wird noch ein Stopbit angefügt, dass auch die Länge einer Bitzeit hat. In Abbildung 1 in eine entsprechende Übertragung auf der UART Schnittstelle zu sehen. Das hier übertragene Zeichen ist ein 'A'.



Abbildung 1: Kommunikation Serielle Schnittstelle

Nun müssen noch die Begriffe Baudrate und Bitzeit definiert werden. Um eine einheitliche Übertragung zu gewährleisten, gibt es die Baud Rate. Die Baud Rate oder auch Symbolrate genannt, gibt an, wie viele Symbole pro Sekunde übertragen werden. Ein Symbol besteht hier aus 1 Bit. Womit die Symbolrate hier gleich der Bitrate ist. Dies ist nicht immer gegeben. Hier ist die Symbol- und Bitrate gleichwertig, da jedes Bit genau ein Symbol überträgt.

$$\text{Bitzeit [s]} = \frac{1}{\text{Baudrate}}$$

Bei der vorhandenen Baud Rate von 9600 kommen wir also auf eine Symbolzeit bzw. Bitzeit von  $1/(9600 \text{ 1/s}) = 104 \mu\text{s}$ . Die UART / RS-232 Schnittstelle benutzt eine NRZ (Non Return to Zero) Kodierung. Bei der NRZ Kodierung ändert sich der Pegel bei einem High-Signal in der Bitmitte nicht.



Abbildung 2: Bitzeiten Serielle Schnittstelle

### 2.1.2.2 – I<sup>2</sup>C

Der **I**nter-**I**ntegrated **C**ircuit **B**us, IIC oder auch I<sup>2</sup>C Bus ist eine Entwicklung der Firma Philips von Anfang 1980. Er war ursprünglich für Consumer Anwendungen gedacht, da die damaligen parallelen Busse viele Leitungen benötigten. Einige Hersteller wie Atmel, bezeichnen den I<sup>2</sup>C Bus auch als **T**wo **W**ire **I**nterface, kurz TWI, um Lizenzkosten zu sparen. Insgesamt benötigt der Bus nur 2 Leitungen. Eine Datenleitung (kurz SDA) und eine Takteleitung (kurz SCL). Da es eine Takteleitung gibt, handelt es sich hier um einen synchronen Bus. Dies hat sowohl Vor- als auch Nachteile. Ein Nachteil besteht in der zusätzlichen Takteleitung. Ein großer Vorteil ist allerdings, dass jetzt nicht mehr auf das genaue Timing geachtet werden muss. Auch eine Synchronisation zwischen Sender und Empfänger ist notwendig, da der Takt mit übertragen wird.

Der I<sup>2</sup>C Bus kann als Single Master und als Multi Master Bus genutzt werden. Dabei muss sichergestellt werden, dass immer nur ein Master kommuniziert.

Jeder Slave hat eine 7-Bit Adresse Basisadresse, über die er angesprochen werden kann. Aus dieser lässt sich direkt eine Leseadresse (0xXXXX XXX0) und eine Schreibadresse (0xXXXX XXX1) ableiten. Die Anzahl der maximalen Slaves ist nur durch die Adressen selber begrenzt. Oft sind die Adressen von den Herstellern fest vergeben. Bei z.B. EEPROM's findet man aber oft auch eine Einstellmöglichkeit der von 3 LSB Bits. Es ist nicht möglich 2 oder mehr Bauteile mit der gleichen Adresse an einem Bus zu betreiben. Hierzu müssen getrennte Busse z.B. mit Hilfe von Multiplexern erstellt werden.

Die Kommunikation über den I<sup>2</sup>C Bus läuft nach einem festen Schema ab.

Auch ist jede Kommunikation eindeutig. So kann es nicht passieren, dass eine Nachricht 2 Bedeutungen hat und es zu Problemen auf dem Bus führt.

Wie in dem Diagramm rechts zu sehen ist, läuft die Kommunikation auf dem Bus wie folgt ab:

Zuerst wird eine START Bedingung gesendet. Darauf folgt die Adresse des

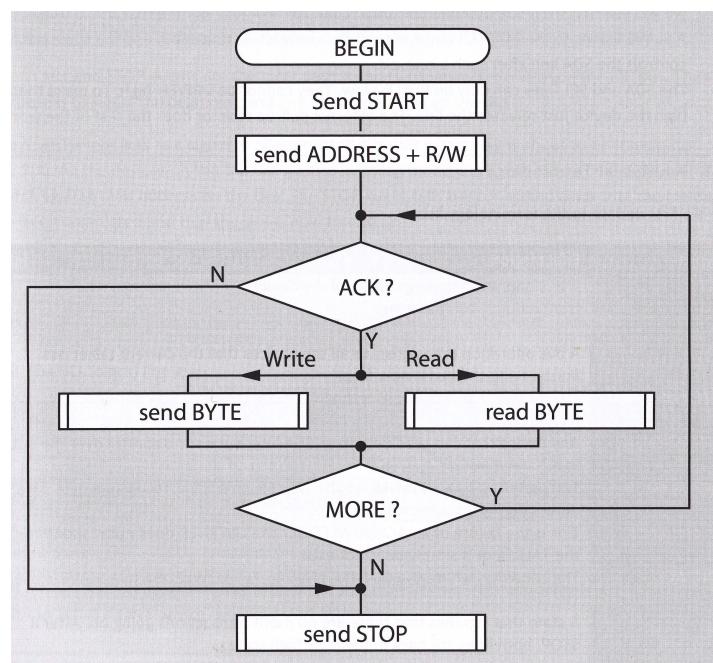


Abbildung 3: I<sup>2</sup>C Kommunikationsablauf

---

Slaves zusammen mit dem Read / Write Bit. Wenn der Slave mit einem ACKnowledge bestätigt, war die Adresse korrekt und die Kommunikation kann fortgesetzt werden. Als nächstes können die Daten Byte weise an den Slave geschrieben bzw. vom Slave gelesen werden. Dabei muss beim schreiben nach jedem Byte, auf den ACK des Slaves gewartet werden, wenn noch weitere Daten geschrieben werden sollen. Beim Lesen muss der Master nach jedem Byte ein ACK senden, wenn er noch Daten lesen möchte. Ist die Kommunikation abgeschlossen, wird der Bus mit dem senden der STOP Bedingung wieder freigegeben.

## 2.2 – Bosch BMP085

Der BMP 085 ist ein von Bosch entwickelter Sensor für Temperatur und Druck. Es handelt sich dabei um einen digitalen Sensor, so das keine weitere Wandlung von analog zu digital erfolgen muss. Der Sensor ist bereits ab Werk kalibriert. Die gemessene Temperatur des BMP085 wird auch auf dem LCD angezeigt und an den PC übertragen.

### 2.2.1 – Sensorspezifikationen

- Messbereich
  - Luftdruck: 300 – 1100 hPa
  - Temperatur: -40 - +85 °C
- Versorgungsspannung: 1,8 – 3,6V
- Kommunikation über I<sup>2</sup>C Bus
- Werksseitig kalibriert
- Typische, absolute Abweichungen
  - Luftdruck: ± 1,0 hPa
  - Temperatur: ± 1,0 °C
- Auflösung:
  - Luftdruck: 0,01 hPa
  - Temperatur: 0,1 °C

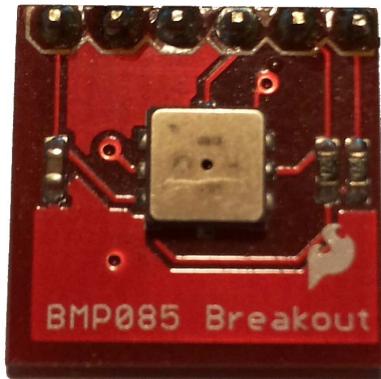


Abbildung 4: Sensor BMP085

### 2.2.2. – Messprinzip / Messwertverarbeitung

Der Luftdruck Sensor funktioniert mit dem Piezoresistiven Effekt. Dabei wird der Widerstand einer dünnen Silizium Membran gemessen. Wenn auf diese Membran ein Druck ausgeübt wird, ändert sich deren Widerstand. Über diesen wird der aktuelle Luftdruck bestimmt.

Das Messprinzip der Temperatur ist im Datenblatt nicht aufgeführt.

Obwohl der Sensor digital arbeitet, sind die ausgelesenen Sensordaten unkomprimierte Rohdaten. Der BMP085 hat insgesamt 11 long integer (16Bit) die die Sensor spezifischen Kalibrierungswerte enthalten. Diese sind von Werk aus für jeden Sensor individuell. Um nun den Luftdruck zu bestimmen, muss im ersten Schritt die kompensierte Temperatur berechnet werden. Mit diesem Wert lässt sich dann im zweiten Schritt der wahre Wert für den Luftdruck berechnen. Die genaue Rechnung findet sich im Datenblatt auf S. 13. Obwohl sich viele Multiplikationen und Divisionen durch Schiebeoperationen vereinfachen lassen, sind Abweichungen bereits an dieser Stelle wahrscheinlich.

## 2.2.3 – Kommunikationsschnittstelle

Der Sensor kommuniziert über den I<sup>2</sup>C Bus. Die genaue Funktionsweise

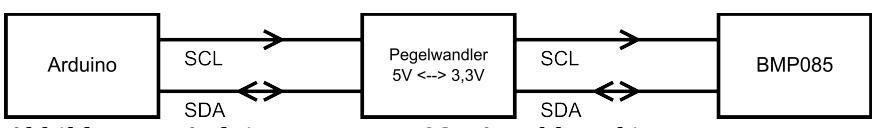


Abbildung 5: Arduino <-> BMP085 Anschlussskizze

findet sich in Kapitel 2.1.2.2 – I<sup>2</sup>C. Die Spannungsversorgung ist über die 3,3V Leitung am Arduino realisiert. Bei der Kommunikation mit diesem Sensor ergibt sich allerdings das Problem der unterschiedlichen Spannungspegel. Der Arduino arbeitet mit 5V wohin gegen der BMP085 für maximal 3,6V ausgelegt ist. Hier würde der Sensor zerstört werden, da die Eingänge (Daten- und Takteleitung) nicht 5V tolerant sind. Bei einer genaueren Betrachtung der Funktionsweise des I<sup>2</sup>C Busses würde im ersten Moment angenommen werden, dass dies kein Problem ist, da der I<sup>2</sup>C Bus Pull-Up Widerstände und Open-Collector Eingänge benötigt. Bei einer genaueren Betrachtung der mitgelieferten 'Wire' Bibliothek stellt sich allerdings heraus, dass die Intern Pull-Up Widerstände des ATMegas benutzt werden. Daraus ergibt sich, dass die Benutzung eine bidirektionalen Pegelwandlers unumgänglich ist. Ein entsprechender Aufbau findet sich in der Application Note AN10441 von NXP. Der darin enthaltene Pegelwandler funktioniert nach folgendem Prinzip:

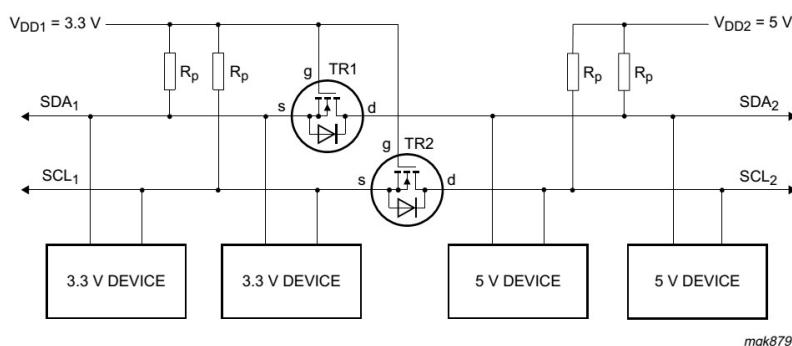


Abbildung 6: Schaltplan I<sup>2</sup>C Pegelwandler

Wenn der Bus im Idle Zustand ist, sperren beide FET's da die Spannung zwischen Gate (g) und Source (s) nicht  $U_{GSTh}$  (Gate – Source threshold) ist. Dadurch werden beide Seiten auf den entsprechenden Pegeln gehalten. Wenn jetzt der Arduino die 5V Leitung auf Masse zieht, wird die parasitäre Diode im FET leitend. Da durch wird die 3,3V Leitung auch in Richtung Masse gezogen. Dies bewirkt dann, dass der FET anfängt zu leiten, da  $U_{GSTh}$  überschritten wird und somit auch die 3,3V Leitung über den FET sicher auf Masse gezogen wird. Der letzte Fall ist, dass ein 3,3V Slave den Bus auf Masse zieht. Hier durch wird der FET leitend, da wie schon zuvor  $U_{GSTh}$  unterschritten wird. Nun wird über den leitenden FET auch die 5V Leitung auf Masse gezogen.

Allerdings ergibt sich bei der Kommunikation ein weiteres Problem. Mit den im Atmel vorhandenen Pull-Up Widerständen lassen sich nicht die benötigten Rise-Time gemäß der I<sup>2</sup>C Spezifikation erreichen. Wie sich in dem nebenstehenden Oszilloskopogramm erkennen lässt, liegt die Rise-Time auf der Takteleitung (Gelb) bei 4,13 µs

und auf der Datenleitung (Grün) sogar bei 6,05µs. Gemäß der I<sup>2</sup>C Spezifikation ist allerdings eine Rise-Time von 1000ns (1µs) erlaubt. Die Internen Pull-Up Widerstände sind im Datenblatt mit 20kΩ – 50kΩ angegeben. Bei I<sup>2</sup>C werden 4,7kΩ empfohlen. Es zeigt sich aber, dass erst mit sehr niederohmigen Widerständen ein Spezifikations gemäßes Ergebnis erzielt ließ. Die hier verwendeten Widerstände haben einen Wert von 560Ω. Auf Grund des Oszilloskopograms gibt es vermutlich noch parasitäre Kondensatoren in der Schaltung. Dies ließ sich abschließen nicht klären.

Wenn nun mit dem Sensor kommuniziert wird um Messwerte zu Empfangen, läuft dies immer nach dem gleichen Schemata ab. Es wird die Adresse gesendet, zusammen mit der Information, ob ein Register gelesen oder geschrieben werden soll. Als nächstes wird der Befehl z.B. 'Wandle Temperatur' gesendet. Dann muss gewartet werden, bis der Sensor die Wandlung abgeschlossen hat. Daraufhin muss die Adresse des Registers gesendet werden, das gelesen werden soll. Bei den Messwerten sind dies 2 Byte. Dementsprechend müssen 2 Register gelesen werden. Dies muss für Temperatur und Luftdruck separat gemacht werden, da der Sensor immer nur einen Messwert wandeln kann. In dem unten stehenden Bild ist beispielhaft eine entsprechende Kommunikation zu sehen.

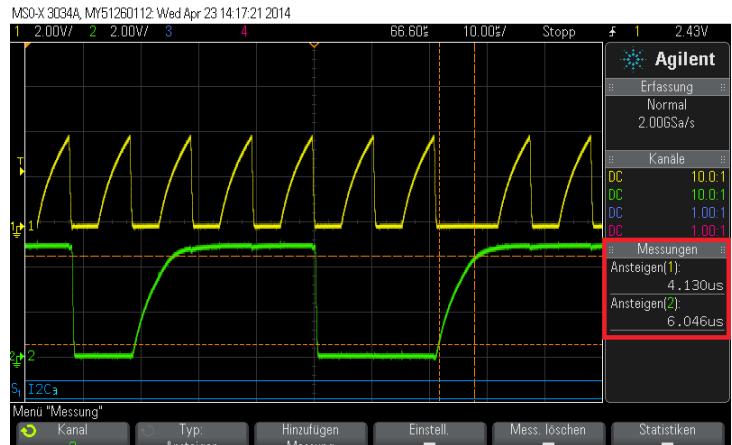


Abbildung 7: I<sup>2</sup>C Oszilloskopogramm Rise-Time

Wenn nun mit dem Sensor kommuniziert wird um Messwerte zu Empfangen, läuft dies immer nach dem gleichen Schemata ab. Es wird die Adresse gesendet, zusammen mit der Information, ob ein Register gelesen oder geschrieben werden soll. Als nächstes wird der Befehl z.B. 'Wandle Temperatur' gesendet. Dann muss gewartet werden, bis der Sensor die Wandlung abgeschlossen hat. Daraufhin muss die Adresse des Registers gesendet werden, das gelesen werden soll. Bei den Messwerten sind dies 2 Byte. Dementsprechend müssen 2 Register gelesen werden. Dies muss für Temperatur und Luftdruck separat gemacht werden, da der Sensor immer nur einen Messwert wandeln kann. In dem unten stehenden Bild ist beispielhaft eine entsprechende Kommunikation zu sehen.

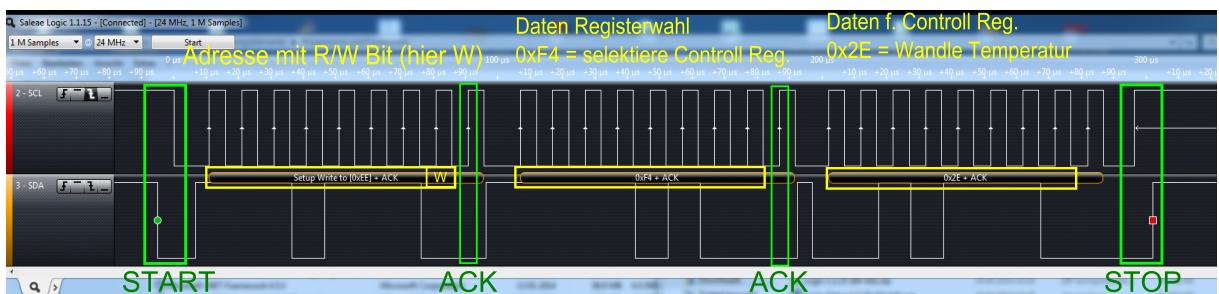


Abbildung 8: I<sup>2</sup>C Kommunikation mit Detailbeschriftung

## 2.3 – Sensirion SHT015

Der SHT15 ist ein von Sensirion entwickelter Sensor für die Messung von Temperatur und Luftfeuchte. Es handelt sich dabei um einen digitalen Sensor, so das keine weitere Wandlung von analog zu digital erfolgen muss. Auch ist der Sensor bereits ab Werk kalibriert.

### 2.3.1 – Sensorspezifikationen

- Messbereich
  - Luftfeuchte: 0 – 100%
  - Temperatur: -40 - +123 °C
- Versorgungsspannung: 2,4 – 5,5V
- Kommunikation über proprietäres Protokoll
- Werksseitig kalibriert
- Typische, Abweichungen
  - Luftfeuchte: ± 2,0 %
  - Temperatur: ± 0,3 °C
- Auflösung:
  - Luftfeuchte: 0,05 %
  - Temperatur: 0,01 °C

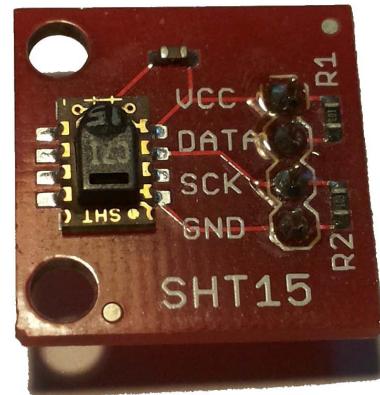


Abbildung 9: Sensor SHT15

### 2.3.2 – Messprinzip / Messwertverarbeitung

Für die Feuchtigkeitsmessung benutzt der Sensor eine Technologie namens „CMOSens“. CMOSens arbeitet auf kapazitiver Basis. Dabei wird die Änderung der Kapazität gemessen. Das Dielektrikum, also die Trennschicht zwischen den beiden Kondensatorpolen, ist aus einem Material das Wasser aufnehmen und abgeben kann. Dabei ändert sich die Kapazität. Dies lässt einen direkten Rückschluss auf die aktuelle Luftfeuchtigkeit zu. Bei diesem Sensor ist zusätzlich zu beachten, dass nicht öfter als mit 1Hz (1 mal pro Sekunde) gemessen werden kann. Sonst gibt es eine Verfälschung der Messwerte aufgrund der Eigenerwärmung.

Der eingebaute Temperatursensor arbeitet mit dem Band-Gap Verfahren. Dabei wird die Temperatur über die Änderung der Bandabstandsspannung des Siliziums bestimmt.

Bei dem SHT15 gibt vorgegeben Kalibrierungskonstanten. Diese können dem Datenblatt entnommen werden. Wie diese gewählt werden, hängt von dem verwendeten Sensor (SHT1x) und der Betriebsspannung ab. Dementsprechend muss auch bei dem SHT15 erst die richtige Temperatur und dann daraus die kompensierte Feuchtigkeit berechnet werden.

### 2.3.3 – Kommunikationsschnittstelle

Die Kommunikation mit dem SHT15 läuft über eine zwei – Draht Schnittstelle (Two - Wire Interface). Es gibt eine Takt- und eine Datenleitung womit auch der SHT15 über eine synchrone Kommunikationsschnittstelle verfügt. Ist zum Teil an das I<sup>2</sup>C Protokoll angelehnt. Da aber signifikante Änderungen vorgenommen wurden, sind die beiden Protokolle zu einander nicht kompatibel. Ein Beispiel ist die verwendete Startbedingung. Wie in Abbildung 11 zu sehen ist, muss beim SHT15 vorher die Clock Leitung auf Masse gezogen werden, bevor die Data Leitung auf Masse gezogen werden kann. Bei dem I<sup>2</sup>C Protokoll entfällt dieser erste toggle. Ein weiterer Punkt der zu einer Inkompatibilität führt ist, dass der Sensor nach der erfolgreichen Wandlung die Datenleitung auf Masse zieht. Bei einem Master gesteuerten Protokoll wie dem I<sup>2</sup>C, wäre dies nicht möglich. Der Umstand, dass es sich hier um ein Proprietäres Protokoll handelt sorgt dafür, dass es komplett selber in den Programmcode implementiert werden muss.

Es gibt aber auch viele Gemeinsamkeiten zwischen I<sup>2</sup>C und dem Sensirion Protokoll. Zum Beispiel sind beide Protokolle Byte orientiert. Eine weitere Gemeinsamkeit ist die Benutzung eines Acknowledge Bit, um korrekten Empfang der Daten zu bestätigen. Auch benutzen beide Bussysteme Pull – Up Widerstände für einen definierten Pegel auf dem Bus.

Im Gegensatz zu dem BMP085 ist bei dem SHT15 keine Anpassung des Buspegels nötig. Der Sensor arbeitet genau wie der Arduino mit 5V.

Auch hier läuft die Kommunikation nach einem ähnlichen Ablauf wie bei dem BMP085. Es wird zuerst ein Kommando Byte gesendet, dass bei dem Sensor beispielsweise die Wandlung der Luftfeuchte auslöst. Nach einiger Zeit zieht der Sensor dann die Datenleitung auf Masse als Zeichen, dass die Wandlung abgeschlossen ist. Im nächsten Schritt gibt der Sensor dann mit jedem Clockimpuls die Daten aus. Es ist keine Adresse im Sensor vorhanden. Daraus ergibt sich auch, dass das Protokoll kein Bus ist, da mehrere Sensoren nicht unterschieden werden könnten

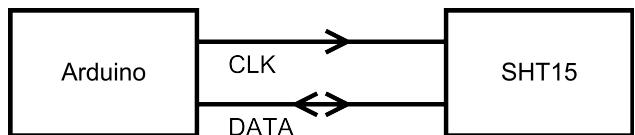


Abbildung 10: Arduino <-> SHT15 Anschlusskizze

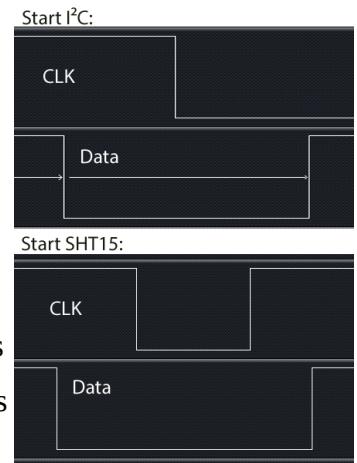


Abbildung 11: Start I<sup>2</sup>C vs. SHT15

## 2.4 – LCD

Bei dem LC – Display handelt es sich um ein GDM1602. Dies ist ein Character Display mit 2 Zeilen die je 16 Zeichen darstellen können. Das Display verfügt bereits über den entsprechenden Zeichensatz. Intern benutzt das LCD einen HD44780 kompatiblen Controller. Dieser Typ wird bei fast jedem Text LCD eingesetzt. Angesteuert wird das LCD über ein paralleles Interface mit Daten- und Steuerleitungen. Dieses verfügt über 8 Datenleitungen (D0 – D7) und 3 Steuerleitungen. Es gibt eine Enable (E) Leitung und eine Register Select (RS) Leitung. Zusätzlich kann noch eine Read / Write (RW) Leitung genutzt werden. Diese wird hier aber nicht genutzt. Mit den Datenleitungen werden die benötigten Daten zwischen LCD und Mikrocontroller ausgetauscht. Über die Enable Leitung wird dem Display mitgeteilt, dass die nun anliegenden Daten gültig und bereit zur Übernahme sind. Die Register Select Leitung gibt an, ob es sich bei den anliegenden Daten um Konfigurationsdaten oder Displayinhalt handelt. Über die nicht genutzte Read / Write Leitung lassen sich Daten aus dem Display auslesen. Die kann zum Beispiel der aktuelle Displayinhalt sein.

Ein weiterer Anschluss ist für die Einstellung des Kontrastes zuständig. Dabei wird über ein Poti eine Spannung zwischen Masse und Versorgungsspannung eingestellt. Der Anschluss erfolgt wie in dem nebenstehenden Bild zu sehen ist.

Die Datenschnittstelle kann in 2 Modi betrieben werden. Einmal mit allen 8 Bit und einmal mit 4 Bit. Da alle Instruktionen ein Byte lang sind, muss im 4 Bit Modus jede Instruktion geteilt werden. Das sorgt für eine langsame Datenübertragung. Der Vorteil hierbei ist allerdings, dass nur die Hälfte der Anschlüsse am Mikrocontroller belegt werden. Gerade bei wenig Anschlüssen ist dies ein unschlagbarer Vorteil. Die Ansteuerung des LCD ist sehr einfach. Im ersten Schritt wird die Konfiguration des Displays geschrieben. Das ist beispielsweise ob ein Cursor angezeigt werden soll oder nicht. Zum Anzeigen der Daten auf dem Display müssen diese auch nicht umkonvertiert werden. Der Zeichensatz des Displays ist so gestaltet, dass dieser dem ASCII Standard folgt.



Abbildung 12: Verwendetes LCD GDM1602

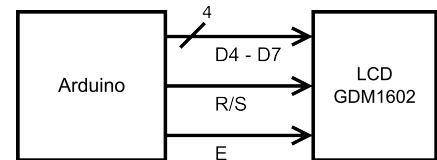


Abbildung 13: Arduino <->  
LCD Anschlusskizze

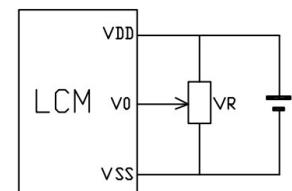


Abbildung 14: Schaltplan  
LCD Kontrastregelung

## 3 – Lösungsentwurf

### 3.1 – Schaltplan

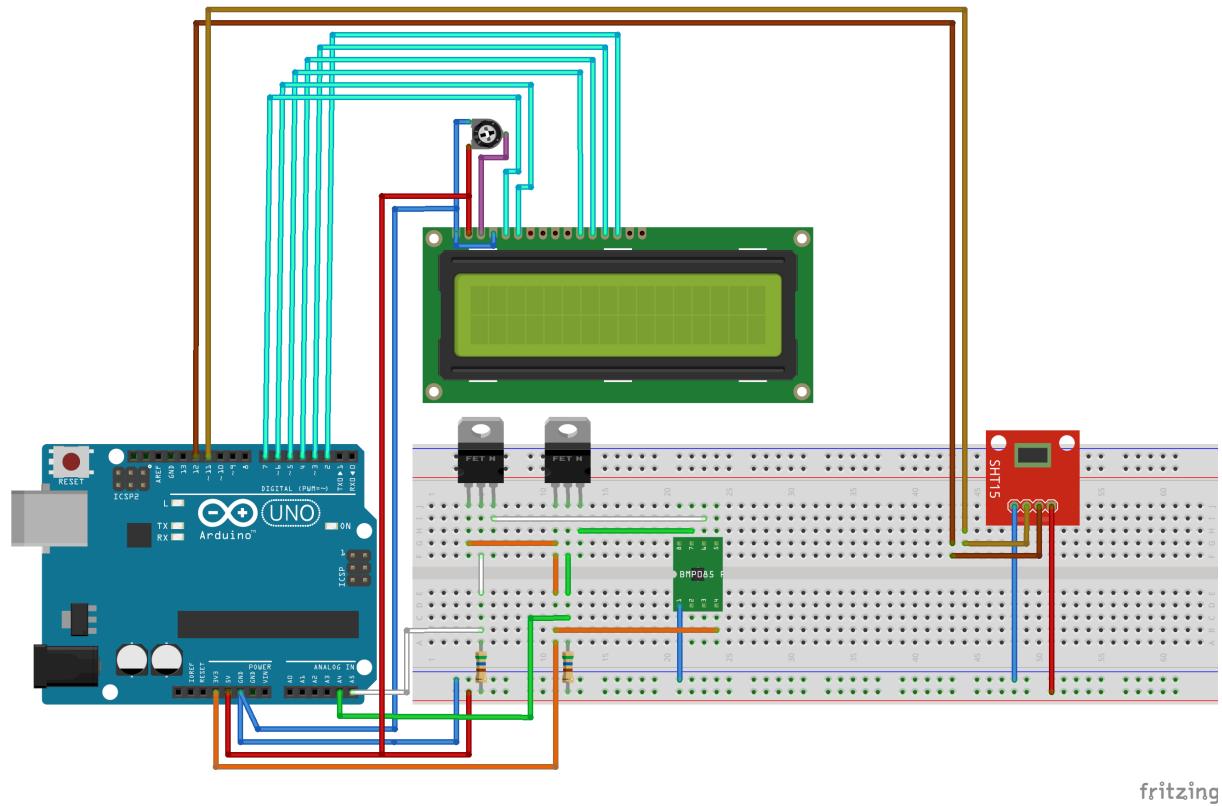
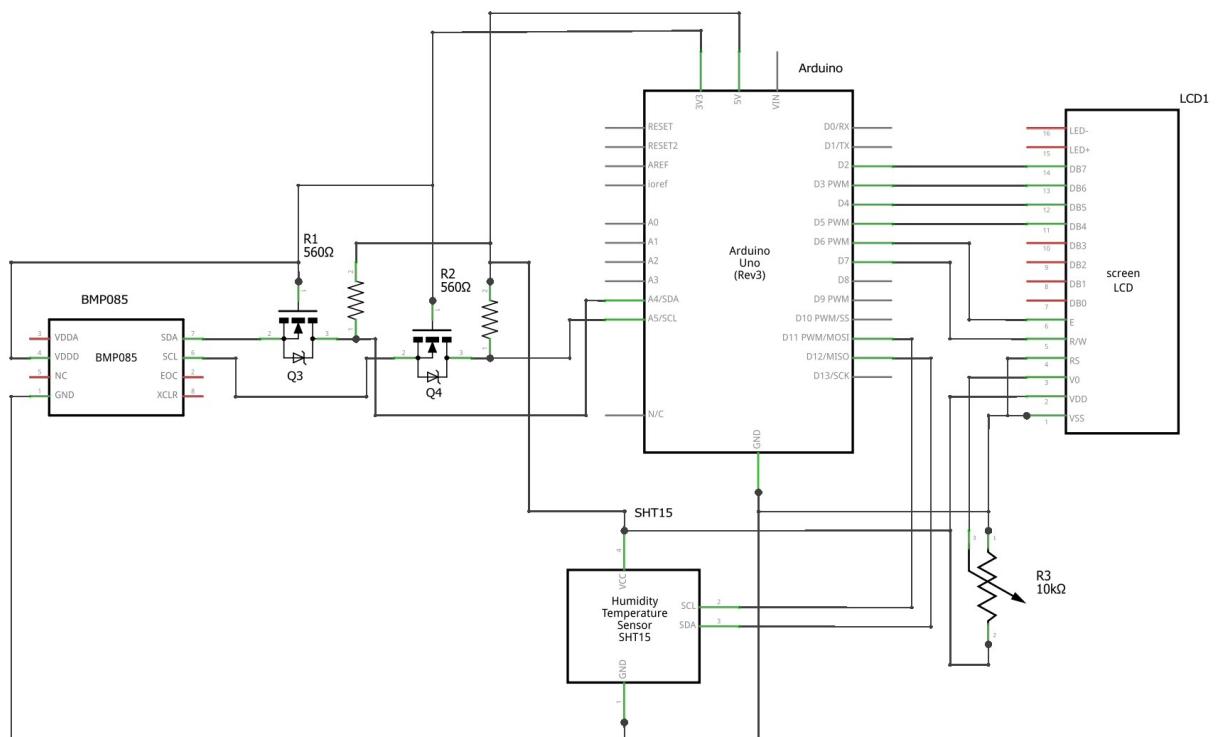


Abbildung 15: Aufbau der Schaltung auf dem Steckbrett - Fritzing



fritzing

Abbildung 16: Aufbau als Schaltplan - Fritzing

### 3.2 – Programmablaufplan

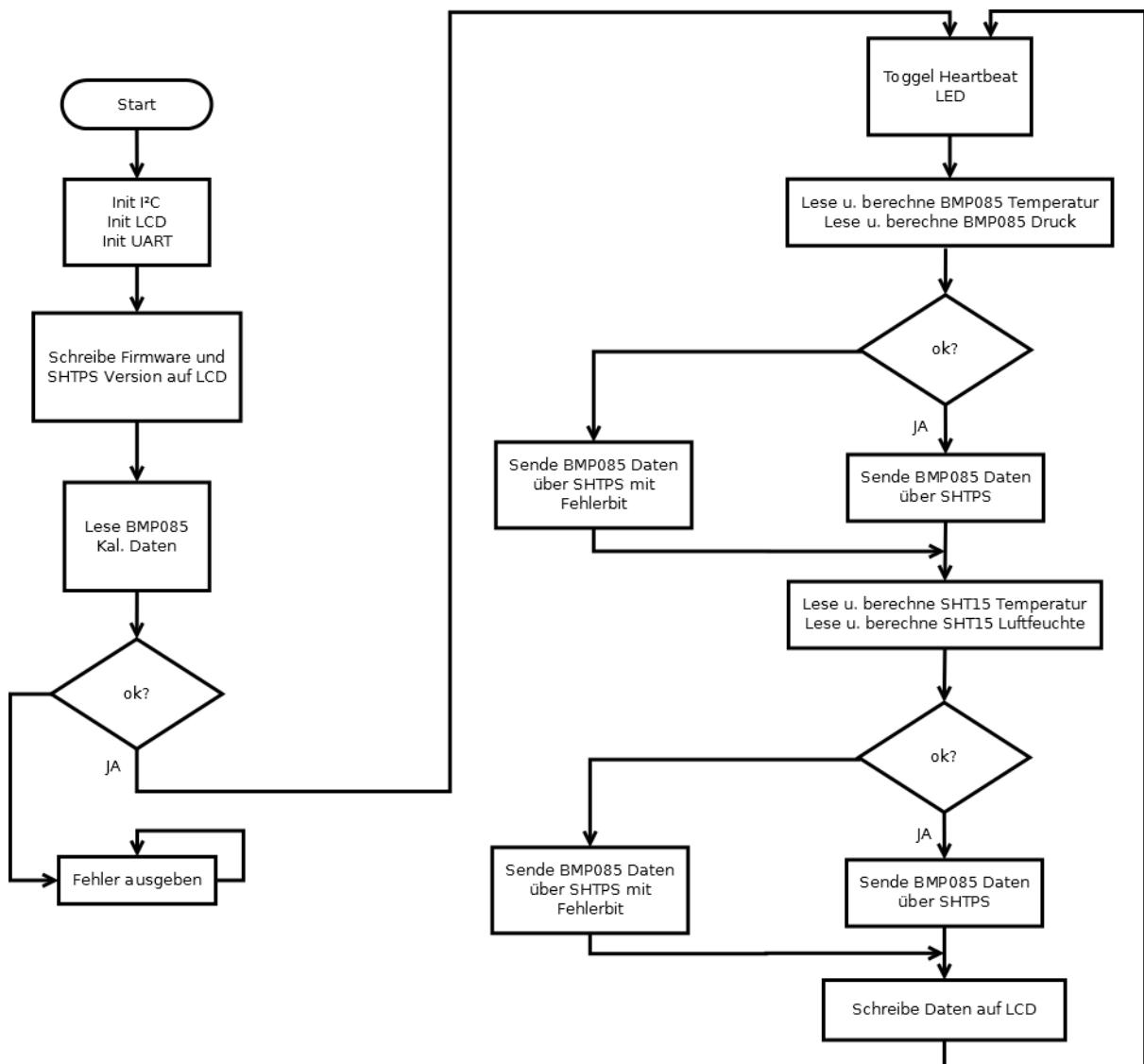


Abbildung 17: PAP der Wetterstation

### 3.3 – SHTPS – Protokoll

Das **S**erial **H**umidity **T**emperature **P**ressure **S**tatus (Kurz SHTPS) Protokoll dient der Übertragung der Daten vom Arduino zum PC. Es wurde eigens für diese Anwendung entwickelt. Aktuell ist es implementiert aber es existiert kein Programm für den PC um dieses auszuwerten. Das Protokoll ist sehr einfach gehalten um auch die Implementierung nicht unnötig zu verkomplizieren. Es basiert auf reinen ASCII Zeichen. Jeder übertragene Frame ist nach folgendem Muster aufgebaut: <Startbyte> <Identifier> <Daten> <Fehlerbyte> <Endbyte>. Über den wird spezifiziert, welche Daten übertragen werden. Beispielsweise 0x01 für Temperatur. Hier ein Beispiel für die Übertragung von **Temperatur**, **Luftdruck** und **Luftfeuchte**:

RSSOH22.5ACKEOTRS|STX0987.85ACKEOTRSETX049ACKEOT

Abbildung 18: Ein SHTPS Frame wie er übertragen wird

Die genaue Spezifikation kann der Quelle X entnommen werden.

## 4. – Auswertung Messergebnisse

Stichprobenartig wurden folgende Werte gemessen:

	Temperatur	Luftfeuchte	Luftdruck
BMP 085	24,7 °C	-	995,43 hPa
SHT 15	24,96 °C	55 %	-
Ref. Messgerät	24,3 °C	56 %	997hPa

Tabelle 1: Vergleich der Messwerte

Obwohl die stichprobenartige Messung nicht repräsentativ ist, lässt sich doch erkennen, ob die Wetterstation richtig arbeitet.

Bei der Temperatur kann man sagen, dass alle 3 Thermometer korrekt arbeiten. Unter der Berücksichtigung der möglichen Abweichungen aller 3 Messgeräte halte ich das Ergebnis für absolut zufriedenstellend.

Auch bei der Luftfeuchte sehen die Ergebnisse passen aus. Die Abweichung von 1% halte ich für doch schon ganz gut.

Als letztes bleibt noch die Betrachtung des Luftdrucks. Auch dieser sieht auf den ersten Blick vollkommen passend aus.

Da auch die Referenzmessung mit einem ungeeichten Messgerät durchgeführt wurden, kann ich hier natürlich nicht für die absolute Korrektheit garantieren. Aber als eine erste Funktionsüberprüfung sollte es vollkommen ausreichend sein.

---

## 5 – Zusammenfassung und Ausblick

Das Projekt verlief weitestgehend ohne Zwischenfälle. Es gab einen sehr guten Einblick in Mikrocontroller Programmierung. Auch wurde das Verständnis von Bussystemen und Sensoren gut ausgebaut. Besonders interessant wurde es bei der Suche nach einer einfachen Lösung für das I<sup>2</sup>C Pegel Problem des BMP085. Aber auch die komplette Implementierung des Kommunikationsprotokolls des SHT15 brachte viele neue Erkenntnisse.

Das Projekt kann aber noch lange nicht als abgeschlossen betrachtet werden. Dafür ist es noch zu ausbaufähig. Auch der Speicher des Mikrocontrollers bietet dafür noch genügend Platz. Aktuell werden erst 11.394 Bytes von 32.256 Bytes für das Programm benötigt.

Der nächste logische Schritt wäre die Implementierung des SHTPS Protokolls in ein Programm für den PC. So ließen sich die Messdaten weiter aufbereiten. Denkbare Features sind beispielsweise das Darstellen als Diagramm, um zeitliche Änderungen erkennen zu können. Auch ein Logging Funktionalität kann hier sehr hilfreich sein.

Aber auch die Hardware an sich kann noch weiter verbessert werden. Unter dem Aspekt Energieeffizienz und autarker Betrieb kann der Energieverbrauch der einzelnen Komponenten analysiert und wenn möglich, sogar optimiert werden. Wenn die Station nun zu Logging Zwecken eingesetzt werden soll, dann muss natürlich auch gewährleistet sein, dass die Daten auch gespeichert werden können. Auch hier gäbe es zwei Ansätze, die verfolgt werden könnten. Der eine ist eine SD Karte, auf die die Daten direkt abgelegt werden können. Eine zweite Möglichkeit kann der Einsatz von WLAN sein. Dann können die Daten direkt auf einen Server übertragen werden und es muss der Messbetrieb zur Auswertung nicht unterbrochen werden. Diese Lösung ist im Gegensatz zu der ersten allerdings deutlich weniger Energieeffizient und somit vermutlich nicht für einen autarken Einsatz geeignet.

---

## Anhang A – Quelltextauszug

```
//Daten entsprechend des SHT15 Protokolles senden
//Rückgabewert gibt an, ob die Übertragung ok war
boolean sht15WriteData(int byteCount, byte data)
{
    digitalWrite(sht15_clk, LOW);
    sht15DataHigh();
    boolean ackReceived = true;
    int i, j;
    //byte auseinandernehmen
    for (j = 0; j < 8; j++) //senden der Bytes
    {
        //maskieren und auf 1 prüfen. Wenn true, 1 übertragen. sonst 0
        if ((data & 0x80) == 0x80)
        {
            digitalWrite(sht15_clk, LOW);
            delayMicroseconds(10);
            sht15DataHigh();
            delayMicroseconds(10);
            digitalWrite(sht15_clk, HIGH);
            delayMicroseconds(10);
            digitalWrite(sht15_clk, LOW);
        }
        else //0 senden
        {
            digitalWrite(sht15_clk, LOW);
            delayMicroseconds(10);
            sht15DataLow();
            delayMicroseconds(10);
            digitalWrite(sht15_clk, HIGH);
            delayMicroseconds(10);
            digitalWrite(sht15_clk, LOW);
        }
        data <<= 1; //nächstes Bit auswählen
    }
    sht15DataHigh(); //auf idle (und input)
    //Bestätigungn durch ACK?
    delayMicroseconds(10);
    digitalWrite(sht15_clk, HIGH);
    delayMicroseconds(10);
    if (digitalRead(sht15_data))
    {
        ackReceived = false;
    }
    digitalWrite(sht15_clk, LOW);
    sht15DataHigh(); //auf idle (und input)

    //}
    return ackReceived;
}
```

---

```

//Ausgabe über UART im SHTPS Protokoll

//SHTPS Protokoll
//Tritt ein Fehler auf, wird ein Errorframe gesendet. Fehler siehe SHTPS
Doku
void sendSHTPSError(byte errorCode)
{
    Serial.write(0x1E);
    Serial.write(0xFF);
    Serial.write(errorCode);
    Serial.write(0x04);
}

//Frame zum Übertragen der Temperatur
void sendTempOverSHTPS(long temp, byte sensorError)
{
    Serial.write(0x1E);
    Serial.write(0x01);
    Serial.print((temp/10), DEC);
    Serial.print(".");
    Serial.print((temp % 10), DEC);
    Serial.write(sensorError);
    Serial.write(0x04);
}

//Frame zum Übertragen des Druckes
void sendPresOverSHTPS(long pres, byte sensorError)
{
    Serial.write(0x1E);
    Serial.write(0x02);
    //Wenn Druck < 1000, vorne mit 0 füllen
    if ((pres/100) < 1000)
        Serial.print("0");
    Serial.print((pres/100), DEC);
    Serial.print(".");
    Serial.print((pres % 100), DEC);
    Serial.write(sensorError);
    Serial.write(0x04);
}

//Frame zum Übertragen der Luftfeuchtigkeit
void sendHumiOverSHTPS(long humi, byte sensorError)
{
    Serial.write(0x1E);
    Serial.write(0x03);
    if (humi <= 0) //Unter bestimmten Voraussetzungen kann es passieren, dass
der Messwert negativ wird
        Serial.print("000");
    if (humi >= 99) //Ab 99% ist der Sensor gesättigt
        Serial.print("100");
    if ((humi > 0) && (humi < 99))
    {
        Serial.print("0");
        Serial.print(humi);
    }
    Serial.write(sensorError);
    Serial.write(0x04);
}

```

# Anhang B – SHTPS Protokoll Spezifikation

The SHTPS Protocol (v0.6)

(S)erial  
(H)umidity  
(T)emperature  
(P)ressure  
(S)tatus

## Error Messages:

RS	0	1	EOT
0x1E	0xFF	Code	0x04

## Code | Message

0x01	Fehler BMP085 Cal. Bytes
0x11	NACK bei BMT085 Temperatur
0x12	NACK bei BMT085 Pressure
0x21	NACK bei SHT15 Temperatur
0x22	NACK bei SHT15 Humidity

## Temperatur Message:

	RS		0		1		2		3		4		5		EOT	
	0x1E		0x01		z		e		'.		d		err		0x04	

BSP:

RS|0x01||'2'||'5'||'.'||'7'||ACK|EOT => 25.7°C Alles ok  
RS|0x01||'2'||'5'||'.'||'7'||NAK|EOT => 25.7°C Busfehler! Daten der  
letzten gültigen Messung

## Pressure Message

### Humidity Message:

	RS	0	1	2	3	EOT
	0x1E	0x03	h	z	err	0x04

BSP:

RS|0x03|'5'|'4'|0x00|ACK|EOT => 54%  
RS|0x03|'5'|'4'|0x01|NAK|EOT => Busfehler! Daten der letzten  
gültigen Messung

RS → Asciizeichen für 'Datensatz'  
EOT → Asciizeichen für 'END\_OF\_TRANSMISSION'  
err → Zeigt, ob es einen Fehler gab  
    ACK → Asciizeichen für Acknowledgement  
    NAK → Asciizeichen für Not Acknowledgement

---

## Anhang C – Abbildungsverzeichnis

Abbildung 1: Eigene Arbeit

Abbildung 2: Eigene Arbeit

Abbildung 3: Quelle 1 S. 25 Kap. 4.1

Abbildung 5: Eigene Arbeit

Abbildung 4: Eigene Arbeit

Abbildung 6: Quelle 2

Abbildung 7: Eigene Arbeit

Abbildung 8: Eigene Arbeit

Abbildung 9: Eigene Arbeit

Abbildung 10: Eigene Arbeit

Abbildung 11: Eigene Arbeit

Abbildung 12: Eigene Arbeit

Abbildung 13: Eigene Arbeit

Abbildung 14: Quelle 6

Abbildung 15: Eigene Arbeit

Abbildung 17: Eigene Arbeit

Abbildung 16: Eigene Arbeit

Abbildung 18: Eigene Arbeit

---

## Anhang D – Quellenverzeichnis

- Quelle 1: Vincent Himpe – Mastering the I<sup>2</sup>C Bus. LabWorx 1  
Elektor Verlag ISBN: 978-0-905705-98-9
- Quelle 2: NXP AppNote AN10441 und I<sup>2</sup>C Spec ([www.nxp.com](http://www.nxp.com))
- Quelle 3: Atmel ATMega238 Datasheet ([www.atmel.de](http://www.atmel.de))
- Quelle 4: Bosch Datasheet BMP085 ([www.bosch-sensortec.com](http://www.bosch-sensortec.com))
- Quelle 5: Sensirion Datasheet SHT15 und CMOSens ([www.sensirion.com](http://www.sensirion.com))
- Quelle 6: LCD GDM1602 Datasheet
- Quelle 7: [www.arduino.cc](http://www.arduino.cc)
- Quelle 8: [www.sparkfun.com](http://www.sparkfun.com)
- Quelle 9: [www.rn-wissen.de](http://www.rn-wissen.de)
- Quelle 10: [learn.adafruit.com](http://learn.adafruit.com)