TU Clausthal

Fakultät für Mathematik/Informatik und Maschinenbau
Lehreinheit Maschinenbau/Verfahrenstechnik

Projektarbeit

im Diplomstudiengang Informationstechnik

Entwicklung einer mikrokontrollergesteuerten Wetterstation

Christoph Altmann
Sebastian Bierwirth
Andreas von Daake
Matthias Döll

Institut für Informatik
Arbeitsbereich Hardwareentwurf und Robotik

Betreuer
Prof. Dr. Günter Kemnitz

Inhaltsverzeichnis

Teil I.

Wetterstation

1 Aufgabenstellung

Herstellung einer Mikrokontrollergesteuerten Wetterstation.

1.1. Ansatz

Auswahl eines geeigneten Mikrocontrollers aus der ATmega Serie. Auswahl sowie Beschaffung geeigneter Sensoren zu Messung von:

- Temperatur Innen/Außen
- Luftdruck
- Luftfeuchte
- Helligkeit
- Windgeschwindigkeit/-richtung

Auswahl eines geeigneten Displays zur Darstellung.

1.2. Umsetzung

Als Basis wurde ein ATmega8 ausgewählt, da dieser

- mit einer möglichen Taktfrequenz von 16Mhz zunächst ausreichend für sämtliche Sensorik schien,
- über jeweils eine TWI-, SPI- sowie UART-Schnittstelle verfügt,
- 6 interne ADC's enthält, davon vier 10 Bit und zwei 8 Bit,
- und 23 I/O-Pins für Peripherie bietet.

Weitere Entwicklung zeigte, dass dieser Prozessor für das hier verwendete Windmessungsverfahren nicht schnell genug arbeitet.

An Sensorik kam zum einen 1-Wire-Komponenten zum Einsatz, um die Erfahrungen mit diesem Bussystem zu sammeln. Und um die Vorzüge des geringen Verdrahtungsaufwandes – speziell für den Einsatz am Außenfühler – zu nutzen.

Eingesetzte 1-Wire Sensoren:

- DS2482-100 Master (Schnittstelle zwischen Mikrocontroller und 1-Wire),
- DS28EA00U (Digital Thermometer),
- DS18S20 (Digital Thermometer),
- DS2450 (Quad A/D-Converter).

Desweiteren ein Drucksensor von Freescale

- MPX4115A (Analoger Sensor zur Messung des absoluten Luftdrucks)

und ein Feuchtigkeitssensor von Sensirion

- SHT11 (Digitaler Sensor zur Messung der relativen Luftfeuchte und Temperatur).

Als Display wurde ein 4x Zeichen Dotmatrixdisplay ausgewählt, welches im Institut zur Verfügung stand.

Die ultraschallbasierte Windmessung wurde aufgrund der zu hohen Anforderungen an den Mikrocontroller noch nicht integriert, jedoch separat untersucht und wird im Teil II näher beschrieben.

2 Luftdruckmessung

2.1. Drucksensor

Zur Druckmessung haben wir uns für den MPX4115A, ein integrierter Silikon Drucksensor von Freescale, entschieden. Dieser Sensor ist relativ günstig, vorkalibriert und sehr gut für den Einsatz mit Mikrocontrollern geeignet. Sein Messbereich von $15kPa$ bis $115kPa$ reicht für unsere Anforderungen vollkommen aus, da für den Luftdruck Werte zwischen $90kPa$ und $110kPa$ normal sind.

Der Arbeitsbereich des Sensors liegt zwischen $-40°C$ bis $+125°C$. Bei Temperaturen zwischen $0°C$ und $+85°C$ beträgt der maximale Fehler $1,5\%$. Mit einer Ausgangsspannung von $0,2V$ bis $4,8V$ hat der Sensor eine Auflösung von $4,6\frac{mV}{kPA}$. Die Versorgungsspannung sollte zwischen $4,85V$ und $5,35V$ liegen (typisch $5,1V$). Die empfohlene Schaltung zur Spannungsversorgung und Ausgangsfilterung ist in Abbildung 2.1 dargestellt.

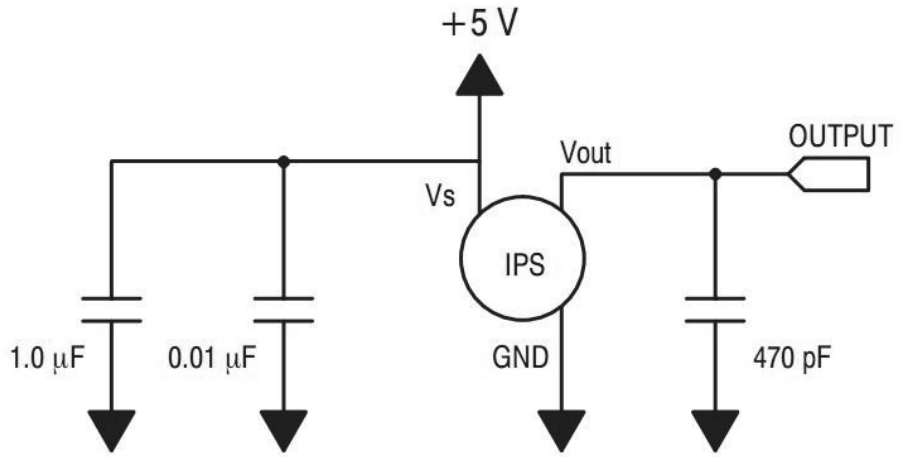

Abbildung 2.1.: Empfohlene Spannungsversorgung und Ausgangsfilterung

2.2. Analog-Digital-Wandlung

Der Drucksensor gibt ein analoges Signal aus. Zur Weiterverarbeitung im Mikrocontroller muss dieses analoge Signal in digitale Werte gewandelt werden. Für diese Aufgabe wird ein Analog- Digital-Umsetzer (ADU/ADC) benötigt. Unser gewählter Mikrocontroller, der ATmega8 hat einen integrierten 10-Bit ADC mit 8 Eingängen. Über die Bibliothek „ADC“ wird eine Funktion bereit gestellt, die für die Initialisierung des ADC vom ATmega8 sorgt, die ADC Messung auf dem gewünschten Eingang aufruft und den empfangenen ADC Wert in eine Spannung wandelt.

2.2.1. „ADC“ Bibliothek:

```
// Initialisierung ADC, Aufruf der ADC Messung, Rückgabe der entsprechenden
    Spannung
double read_ADC(unsigned char chan);
```

2.3. Berechnung Druck

Im Datenblatt des MPX411A ist die Transfer-Funktion des Sensors wie folgt angegeben.

$$V_{out} = V_s \cdot (0{,}009 \cdot P - 0{,}095) \pm Fehler \tag{2.1}$$

Durch umstellen dieser Formel kann man aus dem empfangenen ADC Wert den Druck berechnen.

$$P = \frac{\frac{V_{out}}{V_s} + 0{,}095}{0{,}009} \tag{2.2}$$

Damit hat man den Druck in kPa. Für die Ausgabe in hPa muss noch mit 10 multipliziert werden. Der hier berechnete Druck ist der absolute Luftdruck am Ort der Messung. Da der Luftdruck mit ansteigender Höhe sehr schnell abnimmt, muss, zum Vergleich mit dem Luftdruck in anderen Gebieten, die Standorthöhe aus dem Messwert entfernt werden. Dazu erfolgt eine Umrechnung auf eine gemeinsame Bezugshöhe (Meereshöhe). Die hier verwendete Formel

$$P_{rel} = \frac{P_{abs}}{e^{\left(\frac{g \cdot h}{R_s \cdot \left(T + \frac{a \cdot h}{2}\right)}\right)}} \tag{2.3}$$

wurde aus der Barometrischen Höhenformel abgeleitet und berücksichtigt zur Bestimmung des relativen Luftdrucks neben der Höhe auch die momentane Temperatur. Die Funktion, die obige Formeln implementiert, wird in der Bibliothek „MPX4115A“ bereitgestellt.

2.3.1. „MPX4115A“ Bibliothek

```
// Berechnung des absoluten und des relativen Drucks aus dem ADC-Wert, Höhe und
    Temperatur
void get_pres(double u_val, double temp, double *abs_pres, double *rel_pres);
```

3 Luftfeuchtigkeitsmessung

3.1. Luftfeuchtesensor

Bei der Suche nach einem Sensor zur Messung der Luftfeuchtigkeit sind wir auf den SHT11 von Sensirion gestoßen. Dies ist ein „single chip“ Multisensor für Messungen zur relativen Feuchte und Temperatur mit kalibriertem Digitalausgang. Er ist sehr kompakt, langzeitstabil, sparsam im Verbrauch und somit sehr gut für unsere Wetterstation geeignet. Seine „2-Wire“-Schnittstelle ist sehr vorteilhaft zur Anbindung an einen Mikrocontroller.

Der SHT11 besitzt einen internen 14-Bit-ADC, über diesen sind die zwei Mikrosensoren (Feuchte und Temperatur) mit dem seriellen Interface zur digitalen Signalausgabe gekoppelt.

Der Messbereich des Sensors liegt für die Luftfeuchte zwischen 0% und 100%, für die Temperatur zwischen $-40°C$ und $123,8°C$. Die Genauigkeit für die Feuchte beträgt $\pm 3,0\%$ rF und bei der Temperatur $\pm 0,4°C$, beide Werte beziehen sich auf eine Temperatur von $25°C$.

Die Auflösung von 14 Bit ($0,01°C$) bei der Temperatur- und 12 Bit ($0,03\%$ rF) bei der Feuchtigkeitsmessung kann über ein Statusregister auf 12 Bit ($0.04°C$) und 8 Bit ($0,5\%$ rF) reduziert werden, falls dies für eine schnellere Messung oder geringeren Leistungsverbrauch gewünscht ist.

3.2. Hardware

Abb. 3.1 zeigt die typische Beschaltung des SHT11. Die Versorgungsspannung sollte zwischen $2,4$ und $5,5$ Volt betragen. Der Sensor ist über 2 Leitungen mit dem Mikrocontroller verbunden. Eine ist für den Takt zur Synchronisierung der Übertragung (SCK), über die andere Leitung werden bidirektional Daten ausgetauscht (DATA). Für die Tristate-Datenleitung wird ein Pull-Up- Widerstand benötigt. Zusätzlich wird zwischen GND und VDD ein Kondensator zur Entkopplung empfohlen.

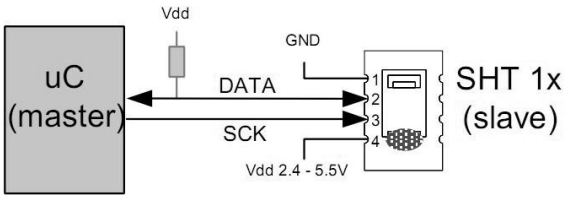

Abbildung 3.1.: Typische Schaltung des SHT11

3.3. Protokoll

Die serielle Schnittstelle ist nicht kompatible zum I^2C-Protokoll, da sie für die Übertragung der Messdaten und geringen Energiebedarf optimiert wurde.

Beim Auslesen der Daten muss darauf geachtet werden, dass nach der fallenden Flanke von SCK die Pegeländerung von DATA erfolgt. Bei steigender Flanke von SCK ist der Status von DATA gültig. Während SCK *high* ist, muss DATA konstant gehalten werden.

3.3.1. Startsequenz

Um dem Sensor einen Kommunikationswunsch mitzuteilen muss zuerst eine Startsequenz gesendet werden. Während die Taktleitung auf high ist, erfolgt eine Absenkung der Datenleitung, dann ein Low-Puls der Taktleitung und schließlich wieder ein Anheben von DATA auf high (Abb. 3.2). Implementiert wurde diese Startsequenz in einer eigenen Funktion in der Bibliothek „SHT11“.

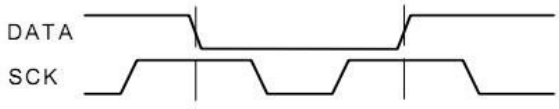

Abbildung 3.2.: Startsequenz

3.3.2. Befehlssequenz

An die Startsequenz knüpft direkt die Befehlssequenz an. Diese besteht aus acht aufeinanderfolgenden Bits. Die ersten drei sind Adressen-Bits, welche immer Null sind, und die restlichen Fünf sind Befehl-Bits. Mit den Befehl-Bits wird dem Sensor mitgeteilt, welche Aktion durchgeführt werden soll, z. B. Temperaturmessung oder Feuchtigkeitsmessung. Hat

der SHT11 den Befehl ordnungsgemäß erhalten, gibt er ein Acknowledge durch Absenkung der Datenleitung.

3.3.3. Mess-Sequenz

Wurde durch die Befehlssequenz eine Messung angestoßen, muss nun gewartet werden bis diese abgeschlossen ist. Bei einer 14 Bit Messung dauert dies etwa 210*ms*. Zieht der SHT11 die Datenleitung auf low, ist die Messung beendet und Daten zum Abholen bereit. Nun wird vom Mikrocontroller der Takt neu gestartet. Anschließend werden erst die Messdaten in zwei Byte, danach noch ein Byte mit der CRC-Checksumme übertragen. Zu beachten ist hier, dass zuerst das MSB kommt und dass jedes empfangene Byte bestätigt werden muss. Außerdem dürfen nicht mehr als 3 Messungen pro Sekunde durchgeführt werden, ansonsten könnte es zu einer Erwärmung des Sensors führen. Dies sollte bei der Wetterstation kein Problem darstellen, da so häufige Messungen nicht notwendig sind. Abb. 3.3 zeigt ein Beispiel für eine Übertragung bei einer Feuchtigkeitsmessung.

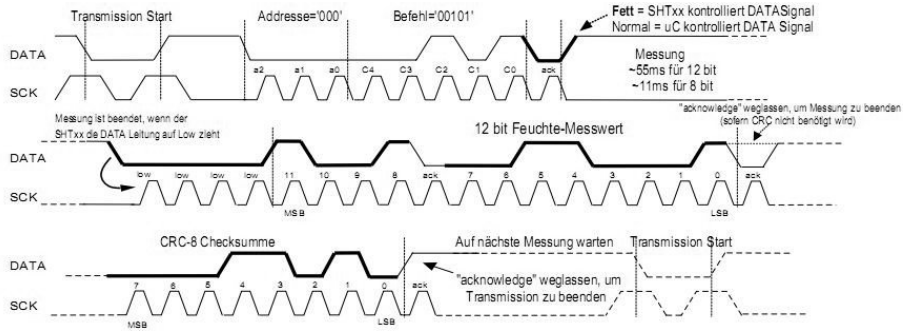

Abbildung 3.3.: Übertragungsbeispiel für eine Feuchtigkeitsmessung

Für die Übertragung von einzelnen Bytes zwischen Mikrocontroller und Sensor, sowie zur Feuchteund Temperaturmessung wurden ebenfalls Funktionen in der „SHT11“ Bibliothek implementiert.

3.3.4. Reset-Sequenz

Falls eine Unterbrechung während der Kommunikation stattfand, muss eine Sequenz zum Zurücksetzen der seriellen Schnittstelle zum Sensor gesendet werden. Diese Sequenz besteht aus neun oder mehr Pulsen der Taktleitung, bei High-Pegel auf der Datenleitung. Anschließend folgt die Startsequenz (siehe Abb. 3.4).

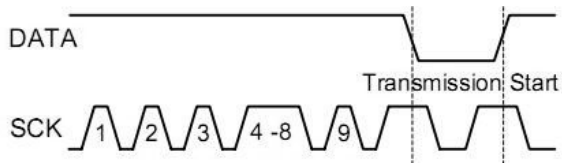

Abbildung 3.4.: Reset-Sequenz

3.4. Status Register

Über das Status Register des SHT11 können verschiedene erweiterte Funktionen eingestellt werden. Von den 8 Bit des Registers sind für die Messeinstellungen 4 Bit relevant, Bit 0, 1 und 2. Über Bit 0 kann wie oben schon erwähnt, die Auflösung der Messung verändert werden. Mit Bit 1 kann der Abgleich der Kalibrierungswerte des Sensors abgeschaltet werden, was die Dauer der Messung verkürzt. Allerdings ist dann die Genauigkeit der Messung nicht gewährleistet. Mit dem 2. Bit des Status Registers kann die eingebaute Heizung des Sensors aktiviert werden. Die Heizung erhöht die Temperatur des Chips um ca. 5°C. Dies kann zur Funktionsüberprüfung verwendet werden. Außerdem verhindert das eingeschaltete Heizelement die Kondensation des SHT11 in sehr feuchter Umgebung.

Weiterhin kann hier das Bit 6 des Status Registers erwähnt werden, welches vom Sensor selbst verändert wird. Wenn die Versorgungsspannung unter das Minimum fällt wird das Bit gesetzt. Da eine Batterieversorgung für unsere Wetterstation im Moment nicht vorgesehen ist, ist dieses Bit für uns weniger interessant.

Bit	Type	Beschreibung	Default	
7		reserviert	0	
6	R	End of Battery (geringe Betriebsspannung '0' für Vdd > 2.47 '1' für Vdd < 2.47	X	Kein default, Bit wird nur nach Messung upgedated.
5		reserviert	0	
4		reserviert	0	
3		Nicht verwenden, nur für Testzwecke	0	
2	R/W	Heizung	0	aus
1	R/W	nicht vom OTP laden	0	reload
0	R/W	'1' = 8bit rF / 12bit Temperatur Auflösung '0' = 12bit rF / 14bit Temperatur Auflösung	0	12bit rF 14bit Temp.

Abbildung 3.5.: Status Register Bits

3.5. Linearisierung und Kompensation

Da die Kennlinien des Sensors für Temperatur und Feuchtigkeit nicht linear sind und die Feuchtigkeit von der Temperatur abhängt, müssen die gemessenen Werte noch korrigiert werden. Formeln dazu finden sich im Datenblatt des SHT11. Die Berechnung hängt von der eingestellten Messauflösung und der Versorgungsspannung ab. Für unsere Einstellungen (14 Bit Feuchte- / 12 Bit Temperaturmessung, $5V$,$^\circ$ C) lauten die Formeln:

Linearisierung der Temperatur

$$\mathrm{T_{lin}} = -40 + 0,01 \cdot \text{Sensorwert(Temperatur)} \tag{3.1}$$

Linearisierung der Feuchtigkeit

$$\mathrm{RH_{lin}} = -4 + 0,0405 \cdot \text{Sensorwert(Feuchte)} - 2,8 \cdot 10^{-6} \cdot \text{Sensorwert(Feuchte)}^2 \tag{3.2}$$

Temperaturkompensation der Feuchtigkeit

$$\mathrm{Rh_{komp.}} = (\mathrm{T_{lin}} - 25) \cdot (0,01 + 0,00008 \cdot \text{Sensorwert(Feuchte)}) + \mathrm{Rh_{lin}} \tag{3.3}$$

Diese Berechnungen sind ebenfalls über eine Funktion in der Bibliothek „SHT11“ implementiert.

3.6. CRC Checksummen Berechnung

Für unsere Zwecke ist eine Überprüfung der Checksumme nicht unbedingt notwendig. Da aber anfänglich Probleme bei der Übertragung auftraten, wurde diese Überprüfung trotzdem implementiert. Dabei verzichten wir allerdings auf die Fehlerkorrektur, es findet ein reiner Vergleich statt.

Das Polynom was im SHT11 für die Checksummenbildung verwendet wird lautet: $x^8 + x^5 + x^4$. Die Berechnung erfolgt über die komplette Übertragung. Es werden gesendete und empfangene Bits berücksichtigt, nur Acknowledges gehen nicht mit in die Checksumme ein.

Der Ablauf zur Bildung der CRC-Checksumme ist wie folgt: zuerst wird das CRC-Register im oberen Byte mit ’0000’ und im unteren Byte mit dem Inhalt des unteren Bytes des Status Registers initialisiert (”0000’$s_3 s_2 s_1 s_0$”). Anschließend wird jedes empfangene Bit mit Bit 7 des CRC-Registers verglichen. Stimmen die Bits überein, findet ein „Shift“ des Registers statt und Bit 0 wird ’0’ gesetzt. Unterscheiden sich das Empfangene und das Bit

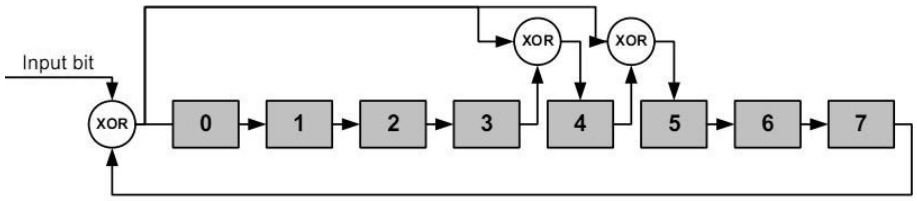

Abbildung 3.6.: CRC-Checksummenbildung

7, werden die Bits des Register ebenfalls verschoben. Allerdings findet im Anschluss eine Konvertierung der Bits 4 und 5 statt und das Bit 0 wird auf '1' gesetzt.

Diese Vorgehensweise wurde in der „SHT11“ Bibliothek durch eine Funktion nachgestellt. Zu beachten ist, dass die Checksumme, die vom Sensor übertragen wird, erst umgekehrt werden muss bevor sie mit der berechneten verglichen werden kann.

3.7. „SHT11“ Bibliothek:

```
// Startsequenz
void s_transstart(void);

// Ein Byte an den Sensor senden
char s_write_byte(unsigned char value);

// Ein Byte vom Sensor empfangen
char s_read_byte(unsigned char ack);

// Reset Sequenz
void s_connectionreset(void);

// Status Register lesen mit Checksumme
char s_read_statusreg(unsigned char *p_value, unsigned char *p_checksum);

// Status Register schreiben
char s_write_statusreg(unsigned char *p_value);

// Sensor zur Messung anstoßen und Messdaten empfangen mit Checksumme
char s_measure(unsigned char *p_value, unsigned char *p_crc_recieved, unsigned
    char *p_crc_calculated, unsigned char mode);

// Linearisierung und Kompensation
void calc_sth11(double *p_humidity ,double *p_temperature);

// Taupunkt Berechnung
double calc_dewpoint(double h,double t);

// CRC Checksummen Berechnung
void calc_checksum(char bit);
```

```
// Bitreihenfolge umkehren - nötig für die empfangene Checksumme
unsigned char retrieve_char(char in);

// Status Register lesen ohne Checksumme - für die Initialisierung des CRC-
    Registers
char s_read_statusreg_crc(unsigned char *p_crc);
```

4 1-Wire

Zur Anbindung vieler Komponenten an den Mikrokontroller der Wetterstation bieten sich das 1-Wire Bussystem an, welches sehr viele Sensoren über eine Datenleitung betreiben kann. Neben dem Vorteil, dass auf diese weise Mikrokontrollerresourcen gespart werden (welche für Displayansteuerung, Windsensor, etc. benötigt werden), ist auch der Verdrahtungsaufwand für den externen Teil der Wetterstation (für Messungen von Außentemperatur etc.) mit einer Datenleitung neben einer gemeinsamen Masse für sämtliche Sensoren sehr gering.

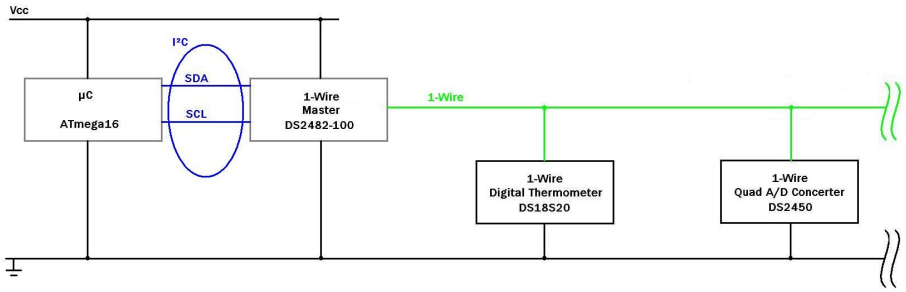

Abbildung 4.1.: 1-Wire Funktionsaufbau

4.1. Grundlagen

4.1.1. Bus

Der 1-Wire Bus ist ein 1-Leiter System zur Halbduplexkommunikation zwischen einem Master und einem oder mehreren Slaves.

Die 1-Wire Slaves (in diesem Projekt Sensoren) werden über eine gemeinsame Datenleitung (1-Wire) im Halbduplexverfahren angesprochen, ausgelesen, sowie mit Energie versorgt.

4.1.2. Energieversorgung

Die zur Kommunikation benötigte Energie speichert jeder 1-Wire Slave in einem internen Kondensator. Dieser wird geladen, wenn die Datenleitung auf „high“ ist, und versorgt den jeweiligen Slave mit Energie wenn die Datenleitung während Datenübertragungen auf „low“ ist.

Ein größerer Energiebedarf, welcher z. B. bei A/D Wandlungen benötigt wird, kann optional ebenfalls über 1-Wire übertragen werden. Dieses geschieht in diesem Projekt sowohl beim Quad A/D Converter (DS2450) als auch beim Digital Thermometer (DS18S20) und ist in den entspr. Abschnitten erläutert.

4.1.3. Adressierung

Die Adressierung der 1-Wire Slaves geschieht über einzigartige 64-Bit Seriennummern, welche bei der Produktion als nicht veränderbar (ROM), auf jeden Slave geschrieben werden. Die ersten 8 Bit sind der „Family Code“, welcher den Typ des Slaves angibt, werden gefolgt von 48 Bit, welche sich innerhalb der Familie niemals wiederholen sowie weiteren 8 Bit CRC.

4.2. Kommunikation

Die Kommunikation geschieht auf dem 1-Wire Bus per „Data-Bit-Level-Communication“. Jedes Bit wird vom Master mit einem „low“ auf der Datenleitung initialisiert. Dieses dient gleichzeitig der Synchronisation.

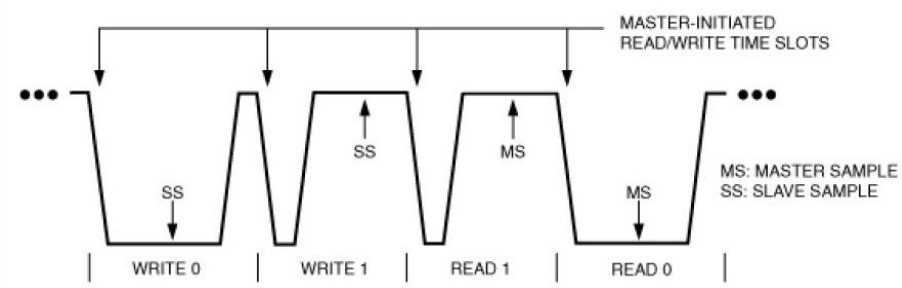

Abbildung 4.2.: 1-Wire Waveform-Beispiel

Um eine „1“ zu schreiben wird die Datenleitung nach Tlow1 ($1\mu s < \text{Tlow1} < 15\mu s$) wieder auf „high“ gesetzt. Fur eine „0“ bleibt sie Tlow0 ($60\mu s < \text{Tlwo0} < 120\mu s$) auf „low“. Die Kommunikation geschieht somit uber die unterschiedliche Lange der jeweiligen „high“ zustande bei gleicher Gesamtlange.

Gleiches gilt fur das Lesen von Daten vom Slave. Hier wird die Datenleitung nach Initialisierung vom Master uber einen internen Pullup auf „high“ gehalten. Der Slave kann die Datenleitung somit fur „0“ auf „low“ ziehen bzw. fur „1“ auf „high“ belassen.

Die 1-Wire Kommunikation setzt sich grundsätzlich aus vier im Folgenden beschriebenen Abschnitten zusammen:

- Initialisierung (4.2.1)
- ROM FUNCTION COMMAND (4.2.2)
- CONTROLL FUNCTION COMMAND (4.2.3)
- Transaction/Data (4.2.4)

4.2.1. Initialisierung

Am Anfang jeder Kommunikation auf dem 1-Wire Bus steht ein Reset. D. h. die Datenleitung muss vom Master fur mindestens $480\mu s$ auf „low“ gehalten werden. Diese Zeit kann von den internen Kondensatoren der Slaves nicht überbrückt werden und fuhrt somit zum Reset. Diese Funktion ist – neben vielen weiteren 1-Wire Funktionen – in der „M1W“ Bibliothek zu finden. (Die Namensgebung der Bibliothek kommt von „Maxim 1-Wire“, da der Name der Bibliothek mit keiner Ziffer beginnen sollte).

„M1W“ Bibliothek (Fur I^2C Master DS2482-100):

```
// Reset und Test ob Slaves vorhanden
void M1W_1WireReset(void);
```

4.2.2. ROM FUNCTION COMMAND

Nach der Initialisierung ist einer von 7 ROM function commands zu senden. ROM function commands sind jeweils 8 Bit lang und dienen z. B. dazu, einzelne Slaves anhand ihrer Seriennummer auszuwählen, oder ihre Seriennummer überhaupt erst zu bestimmen.

Die einfachste Methode einen Slave anzusprechen besteht darin, ihn als einzigen Slave am Bus zu betreiben und ein „SKIP ROM“ (=0xCC) zu senden.

Befinden sich mehrere Slaves am Bus, können diese durch Senden eines „MATCH ROM“ (=0x55) commands, gefolgt von ihrer jeweiligen Seriennummer angesprochen werden. Dein

einfachste Methode die Seriennummer eines Slaves zu bestimmen besteht darin, ihn als einzigen Slave an einen 1-Wire Bus zu hangen und ein „READ ROM“ (=0x33) zu senden (in dem Fall folgt kein Control Function Command, sondern direkt 8x8 Read-Slots zum auslesen der 64 Bit Seriennummer, siehe 4.).

Alle diese ROM function commands sind in der „M1W“ Bibliothek zu finden.

„M1W“ Bibliothek (Fur I^2C Master DS2482-100):

```
// Waehlt Slave aus
void M1W_match_Rom(char Serial[8]);
// Überspringt Slaveauswahl, nur nutzbar, wenn nur ein einiger Slave
// am Bus
// betrieben wird
void M1W_Skip_Rom(void);
// Liest Seriennummer aus, wenn nur ein einziger Slave am Bus
// betrieben wird
void M1W_Read_Rom(void);
```

Des weiteren existieren „SEARCH ROM“ und „CONDITIONAL SEARCH ROM“ zum bestimmen der Seriennummern mehrer gleichzeitig am Bus betriebenen Slaves, sowie „OVERDRIVE SKIP ROM“ und „OVERDRIVE MATCH ROM“ für schnellere Kommunikation (standard speed = 16,3kBit/s overdrive speed = 142kBit/s).

In dieser Projektarbeit war die Nutzung der Standartkommunikationsgeschwindigkeit völlig ausreichend, da Temperatur- und Helligkeitsmessungen keine hohen Dynamik aufweisen. Auch wurden aufgrund des statischen Aufbaus keine Search ROM commands benötigt. Die Wetterstation wird schließlich beim Aufbau mit 1-Wire Slaves bestückt, und diese später nicht mehr ergänzt/ausgewechselt.

4.2.3. CONTROL FUNCTION COMMAND

Nach erfolgreicher Adressierung mittels ROM function command und Seriennummer werden Slave spezifische control function commnands gesendet um die jeweiligen Funktionen des Slaves nutzen zu können. Auch diese commands sind 8 Bit lang.

Im Rahmen der Projektarbeit wurden folgende control function commands in Bibliotheken angelegt:

„DS28EA00U“ Bibliothek (Digital Thermometer):

```
// Wandelt Temperatur
void DS28EA00U_get_Temp(void);
// Liest scratchpad aus und gibt Temperature als double zurück
double DS28EA00U_transmit_Temp(void);
```

„DS18S20“ Bibliothek (Digital Thermometer):

```
// Wandelt Temperatur
void DS18S20_get_Temp(void);
// Liest scratchpad aus und gibt Temperature als double zurück
double DS18S20_transmit_Temp(void);
```

„DS2450“ Bibliothek (Quad A/D Converter):

```
// AD_Wandlung anstoßen
void DS2450_get_AD(void);
// Results aller 4 AD Eingänge lesen
void DS2450_transmit_AD(DS2450* ADC1);
```

4.2.4. Transaction/Data

Nach dem Senden einiger Control function commands bzw. des ROM function commands „Read ROM“ antwortet der Slave über die Datenleitung auf die Anfrage des Masters. Hierzu stellt der Master Read-Timeslots zu Verfügung, in welchen der Slave seine Antwort übertragen kann. 8 Solche Slots stellt die Funktion `M1W_1WireRead` zur Verfügung.

„M1W“ Bibiliothek (Für I^2C Master DS2482-100):

```
// Read 1 Byte from 1-Wire Slave
char M1W_1WireRead(void);
```

4.3. Eingesetzte 1-Wire Komponenten

4.3.1. DS2482-100 Master (Schnittstelle zwischen Mikrokontroller und 1-Wire)

Die Einbindung der 1-Wire Komponenten geschieht, aufgrund des sehr zeitkritischen Protokolls, indirekt über einen Master. Dieser Master (DS2482-100) wird vom ATmega per I^2C angesprochen. Dieses geschieht über zwei Leitungen: Eine Datenleitung (SDA) sowie eine Clock (SCL) welche jeweils über Pullupwiderständen an Vcc angebunden sind.

Hierfür wurde eine I^2C („I2C“) Bibliothek, sowie eine darauf aufbauende 1-Wire Bibliothek („M1W“) geschrieben. Somit lassen sich die 1-Wire Funktionen komfortabel vom ATmega ausführen, als würde dieser selber 1-Wire unterstützen.

Neben Luftdruck-, Wind- und Feuchtigkeitssensoren sollten an der Wetterstation Innen- und Außentemperatur, sowie Helligkeit mittels 1-Wire gemessen werden. Somit viel die Wahl auf die folgenden drei 1-Wire Komponenten.

4.3.2. DS28EA00U (Digital Thermometer)

Das DS28EA00U ist ein digitales Thermometer, welches Temperaturen im Bereich $-40°C$ bis $+85°C$ misst. Im 12 Bit Modus wird dabei eine Auflösung von 16Bit/°C erreicht. Auch die Temperaturwandlung kann im „Parasite Mode“ geschehen, d. h. ohne eigene Versorgungsspannung. In dem Fall wird die Versorgungsspannung für die Dauer der Wandlung vom Master auf dem 1-Wire Bus zu Verfügung gestellt. Hierzu ist am Master ein „Strong Pullup“ zu aktivieren und keine andere Kommunikation auf dem Bus zuzulassen. Die Conversiontime beträgt für den DS28EA00U im parasit mode $< 750ms$. Anschließend kann das Conversionresult aus dem entsprechenden Resultregister des Scratchpads ausgelesen werden.

Weitere Funktionen dieser Sensoren sind manuell festlegbare Temperaturober- und Untergrenzen zur schnellen Bestimmung von Temperatur Über-/Unterschreitungen. Mittels eines speziellen „Alarm Search Command“ lassen sich so sehr schnell alle Sensoren identifizieren, deren Messungen außerhalb des vorgegebenen Bereichs liegen. Des Weiteren bieten diese Sensoren eine „Chain Function“. Mittels dieser Kettenfunktion können mehrere untereinander verbundene Sensoren nacheinander zur Temperaturwandlung angestoßen und ausgelesen werden. Sie geben den Buszugriff über die Kette jeweils an den nächsten Sensor weiter und erleichtern somit dem Master die Adressierung bzw. reduzieren den Kommunikationsoverhead auf dem Bus.

4.3.3. DS18S20 (Digital Thermometer)

Dieser Sensor ist ähnlich dem DS28EA00U. Er weist jedoch eine geringerer Auflösung von nur 2Bit/°C auf und bietet keine Chain Function.

Aufgrund der Verfügbarkeit im Institut – und der Tatsache, dass diese ICs in geringer Menge bei Maxim nicht bestellt, sondern nur als Samples angefordert werden können – wurde dieser Sensor anstelle des besseren DS28EA00U in der Projektarbeit verwendet.

4.3.4. DS2450 (Quad A/D Converter)

Der DS2450 bietet 4 hochohmige Eingänge zur A/D-Wandlung. Der Eingangsbereich lässt sich zwischen 2,56V und 5,12V umstellen und wird mit einer Auflösung von 16 Bit A/D gewandelt. Die Funktionen `DS2450_get_AD` und `DS2450_transmit_AD` dienen dem Wandlungsanstoß, sowie dem Auslesen der gespeicherten Digitalwerte aus dem Scratchpad des A/D Converters. In dieser Projektarbeit wird der DS2450 zunächst zur Ermittlung von Helligkeit

eingesetzt. Eine einfache Analogschaltung erzeugt hierfür mittels Photodiode und Spannungsteiler die Messspannung.

4.4. Geschriebene Bibliotheken

- DS18S20 (Digital Thermometer)
- DS28EA00U (Digital Thermometer)
- DS2450 (Quad A/D Converter)
- M1W (1-Wire)
- I2C (I^2C)

5 Display

Als Anzeige wurde ein im Labor vorhandenes Display mit 16×4 Zeichen und HD44780-kompatiblen Controller gewählt. Es besteht die Möglichkeit das Display mit acht- oder vierbitig anzusteuern. Um Leitungen zu sparen, wurde die etwas langsame 4-Bit-Variante verwendet.

5.1. Ansteuerung

Das Display besitzt eine 16-polige Anschlussleiste. Diese wurde vollständig per Flachbandkabel mit einem Pfostenstecker verbunden. So könnte das Display an anderen Schaltungen auch mit vollen 8-Bit angesteuert werden. In Tabelle 5.1 ist die Pinbelegung aufgeführt. Neben den Datenleitungen werden noch die E- und RS-Leitungen an den Atmega angeschlossen. Der Rest wird mit Versorgungsleitungen auf dem Board verbunden.

Als erstes muss das Display initialisiert werden. Um einen Befehl an das Display zu senden, muss die RS-Leitung auf „low“ gesetzt werden. Danach wird die Enable-Leitung auf „high“ gesetzt, was den Displaycontroller veranlasst die Zustände der Datenbitleitungen abzufragen. Wurde der Befehl gesendet, geht die Enable-Leitung wieder auf „low“. Das Senden von Daten verläuft fast identisch: in diesem Fall hat die RS-Leitung ein „high“-Level.

Da das Display nur mit vier Bit angesteuert wird, müssen die acht Bit Daten in zwei Hälften übertragen werden. Zuerst werden die oberen vier Bits übertragen, danach die anderen. Die Displaybibliothek enthält Funktionen, die das Darstellen von Wörtern, Zeichen oder Symbolen ermöglichen. Außerdem kann der Curser an eine beliebige Stelle gerückt werden oder eigene Zeichen definiert werden. Durch eine Zusatzfunktion ist auch die direkte Eingabe von Sonderzeichen innerhalb von Strings möglich.

Tabelle 5.1.: Pinbelegung des Displaytech 164A

Pin	*Symbol*	*Level*	*Description*
1	VSS	0.0V	Ground
2	VDD	5.0V	Supply voltage for logic
3	VO	—	Input voltage for LCD
4	RS	H/L	H: Data signal, L: Instruction signal
5	R/W	H/L	H: Read mode, L: Write mode
6	E	H, H→L	Enable signal for KS0076
7	DB0	H/L	Data bit 0
8	DB1	H/L	Data bit 1
9	DB2	H/L	Data bit 2
10	DB3	H/L	Data bit 3
11	DB4	H/L	Data bit 4
12	DB5	H/L	Data bit 5
13	DB6	H/L	Data bit 6
14	DB7	H/L	Data bit 7
15	BLA	4.2V	Back light anode
16	BLK	0.0V	Back light cathode

6 Platinendesign

Beim Entwurf der Platine wurde die Software Eagle von CadSoft verwendet. Im ersten Schritt wurde ein Schaltplan (Abbildung 6.1) entworfen, in dem alle Komponenten miteinandern verbunden wurden. Zum Anschluss von Display, Programmieradapter und Sensoren wurden Pfostenbuchsen gewählt. Für den Windsensor eine Sub9-Buchse, da sie aus dem Gehäuse geführt werden sollte. Für Spannungsversorgung sorgt ein Spannungswandler sowie zwei Glättungskondensatoren.

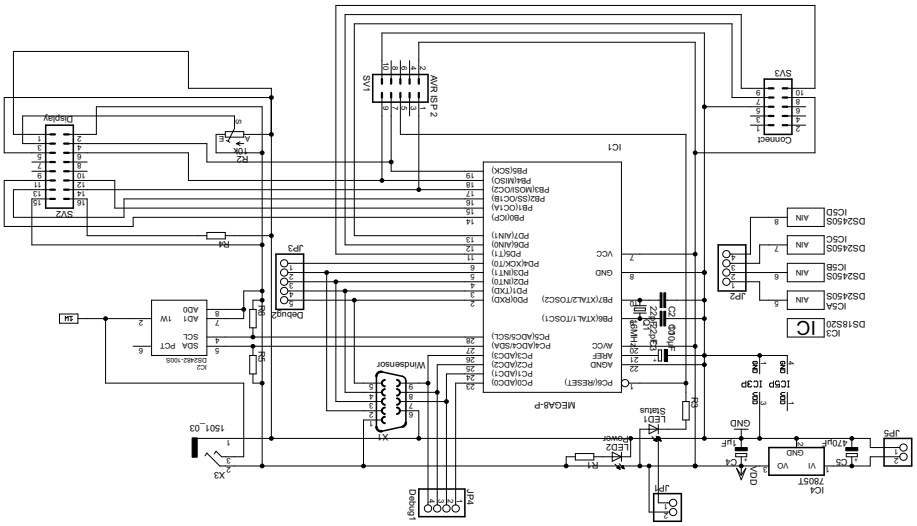

Abbildung 6.1.: Schaltplan

Im zweiten Schritt wurden die Bauteile optimal auf der Platine angeordnet und dann manuell mit Leiterbahnen verbunden. Dabei wurde versucht, möglichst nur eine Platinen-

seite zu benutzen. Trotz aller Mühen waren am Ende neun Kabelbrücken für die andere Seite nötig.

Die Leiterbahn wurden auf Folie ausgedruckt, um im Labor eine Leiterplatte zu belichten. Nach dem Ätzvorgang mussten noch die zahlreichen Löcher gebohrt werden. Erst nach dem fertigen Aufbau (Abbildung 6.2) konnten kleinere Fehler auf der Platine entdeckt werden. So gab es ein paar Pinvertauschungen, aber auch mechanische Probleme.

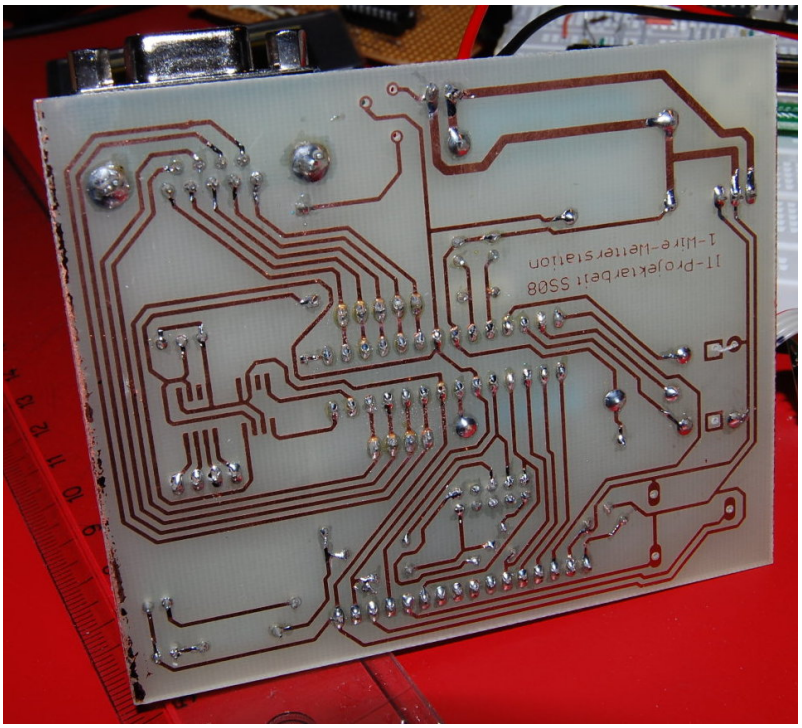

Abbildung 6.2.: erste Version der Platine

Einerseits lösten sich die Leiterbahnen recht schnell, anderseits waren die anfangs benutzten Stiftleisten zu instabil. Daher wurden in einer überarbeiteten Platinenversion die oben genannten Pfostenwannenstecker genutzt, sowie breitere Leiterbahnen verwendet (Abbildung 6.3)

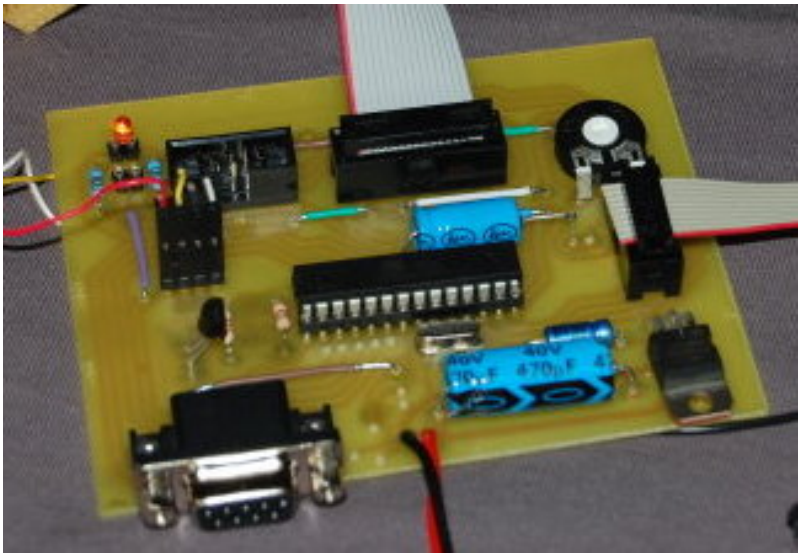

Abbildung 6.3.: zweite Version der Platine

7 Programmierung

7.1. Programmieradapter

Um den Atmega8 zu programmieren ist ein Programmieradapter notwendig. Um ihn auch an neuen Laptops benutzen zu können, wurde nach einem USB-Modell gesucht. Dabei viel die Wahl auf den „USBprog“ von http://embedded-projects.net/ Diese Open-Source-Hardware wurde bestellt, zusammengebaut und programmiert. Er ist bisher unter Linux und Windows lauffähig, so dass sowohl im Labor, als auch daheim ein Programmieren möglich war.

7.2. Hauptprogramm

Das Hauptprogramm steuert die Wetterstation. Es bindet die Bibliotheken ein oder wartet auf Benutzereingaben.

Gleich nach den defines beginnt die Hauptschleife. Hier werden einige Ein- und Ausgänge aktiviert, sowie der Splashscreen angezeigt. Außerdem werden Display und Sensoren initialisiert.

```
int main(void)
{
  DDRC = 0xFF;// all output debug
  PORTC = 0x00;
  DDRD=0x20;  // SCK SHT11
  PORTD=0x80; // Pullup Taster

  sbi(TIMSK,TOIE0);         // enable T0 overflow0 interrupt
  TCCR0 = 0x03;             // 1-1/2-8/3-64/4-256/5-1024
  TCNT0 = 0x00;

  //Display initialisieren
  lcd_init();
  set_cursor(1,1);
  lcd_string(" Projektarbeit  ");
  set_cursor(1,2);
  lcd_string("   TUC SS-08    ");
  set_cursor(1,3);
  lcd_string("");
  set_cursor(1,4);
```

```
  lcd_string("Erstellt durch: ");
  for (char count=0; count<100 ;count++)
  {
    _delay_ms(20);
  }
  lcd_clear();

  set_cursor(1,1);
  lcd_string("Christ. Altmann");
  set_cursor(1,2);
  lcd_string("Seb. Bierwirth");
  set_cursor(1,3);
  lcd_string("Matthias Döll");
  set_cursor(1,4);
  lcd_string("Andreas v. Daake");

  //Sensoren initialisieren
  M1W_Init();
  M1W_DeviceReset();
  M1W_1WireReset();
  DS2450_Init();
  s_connectionreset();

  sei();

  //Hauptschleife
  while (1);
}
```

Wie am Ende des Abschnittes zu sehen war, läuft danach das Programm im Kreis – solange nicht ein Timer-Event auftritt und die folgende Schleife abzuarbeiten ist. In dieser Schleife wird geprüft, ob eine Taste gedrückt wurde, es werden die Messwerte abgerufen und das Display wird beschrieben. Jedoch alles zu unterschiedlich häufigen Zeitpunkten.

Als Beispiel sei hier das Schreiben der Innentemperatur aufgeführt:

```
lcd_clear();
set_cursor(1,1);
lcd_string("Temp_In:  "); LCD_WriteDbl(Temperature1, 1); lcd_string("°C");
set_cursor(1,2);
lcd_string("Max:      "); LCD_WriteDbl(Tmax_In, 1); lcd_string("°C");
set_cursor(1,3);
lcd_string("Min:      "); LCD_WriteDbl(Tmin_In, 1); lcd_string("°C");
```

Teil II.

Windsensor

8 Einleitung

Diese Arbeit ist Teil einer Projektarbeit, in der eine Wetterstation entwickelt werden soll. Für diese Wetterstation soll eine Windmessung mittels Ultraschall-Anemometer erfolgen. Dieser Teil der Projektarbeit beschreibt erste Untersuchungen, wie eine Windmessung mittels Ultraschall realisiert werden kann. Verwendet wird dabei als Mikrocontroller ein Atmega8 von Atmel, da dieser ebenfalls für die Wetterstation zum Einsatz kommt. Es soll untersucht werden, wie sich dieser Mikrocontroller für das Projekt eignet und verschiedene Messergebnisse zusammengetragen werden.

9 Grundlagen

9.1. Ultraschall[1]

Schall ist physikalisch gesehen eine mechanische Schwingung von Materie (Gasen, Flüssigkeiten, Festkörpern), die sich wellenartig ausbreitet. Schall ist also an Materie gebunden und kann sich somit auch nicht im Vakuum ausbreiten. Die Wellenlänge λ lässt sich ausdrücken als

$$\lambda = \frac{c}{f} \tag{9.1}$$

wobei c die Schallgeschwindigkeit in $\frac{m}{s}$ und f die Frequenz in Hz ist. Als Ultraschall wird Schall im Frequenzbereich zwischen $16kHz$ bis $1GHz$ bezeichnet. Die verschiedenen Frequenzbereiche können folgender Tabelle entnommen werden:

Tabelle 9.1.: Frequenzbereiche

Infraschall	0Hz bis 16Hz
Hörschall	16Hz bis 16kHz
Ultraschall	16kHz bis 1GHz
Hyperschall	Über 1 Ghz

Die Schallgeschwindigkeit ist temperaturabhängig und beträgt allgemein:

$$c = \sqrt{\frac{\gamma \cdot p_0}{\rho_0}} \tag{9.2}$$

p_0 ist dabei der statische Druck, ρ_0 die mittlere Dichte und γ das Verhältnis der spezifischen Wärme. In der Luft lässt sich die Schallgeschwindigkeit nach folgender Formel berechnen:

[1] aus [3]

$$c = 331,31 \cdot \sqrt{\frac{T}{273,16}} \cdot \frac{m}{s} \tag{9.3}$$

Damit ergibt sich bei 20°C eine Schallgeschwindigkeit von ungefähr $343\frac{m}{s}$. Erzeugt werden kann Ultraschall z. B. durch Schwingquarze, die durch die Umkehrung des Piezzo-Effektes in Schwingungen versetzt werden können.

9.2. Messprinzip

Um mit Hilfe von Ultraschallsignalen Windgeschwindigkeiten messen zu können, werden zwei gegenüberstehende Ultraschallsensoren verwendet[2]. Einer der Sensoren sendet einen Ultraschallimpuls aus (Sender), der andere empfängt diesen (Empfänger). Die Idee der Ultraschall-Windmessung besteht nun darin, dass sich die Windgeschwindigkeit mit der Schallgeschwindigkeit überlagert und so zu einer kürzeren oder längeren (je nach Windrichtung) Laufzeit führt. Bei völliger Windstille und einer Temperatur von 20°C bewegt sich der Schall mit $343\frac{m}{s}$. Weht nun ein Wind, so beeinflusst dieser die Geschwindigkeit und damit die Laufzeit des Signals.

Um nun Rückschlüsse auf die Windgeschwindigkeit ziehen zu können, muss eine Laufzeitmessung des Signals erfolgen. Abbildung 9.1 verdeutlicht das Messprinzip nochmals.

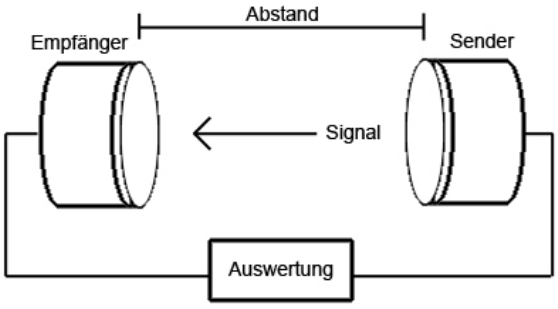

Abbildung 9.1.: Messprinzip

Die wesentlichen Vorteile bei der Windgeschwindigkeitsmessung mit Ultraschall liegen

[2] Zur Vereinfachung werden nur zwei Sender genommen. Damit kann nur die Geschwindigkeit parallel zur Messanordnung gemessen werden

darin, dass es keine beweglichen Teile wie bei der klassischen Windmessung mehr gibt und dass eine höhere Genauigkeit erzielt werden kann.

10 Hardware

10.1. Ultraschallsender und -empfänger[1]

Als Ultraschallsender kommt der „UST-40T“, als Empfänger das entsprechende Gegenstück „UST-40R“ von Reichelt zum Einsatz. Diese haben eine Resonanzfrequenz von $40kHz$ und eine Bandbreite von $2kHz$ (Sender) bzw. $2,5kHz$ (Empfänger). Der Schalldruck beträgt maximal $120dB$ und die dauerhafte maximal zulässige Betriebsspannung $20V$.

Auf Abbildung 10.1 ist das Richtdiagramm der Ultraschallsensoren zu sehen. Darauf lässt sich erkennen, dass die Sensoren einen starken Richtfaktor aufweisen.

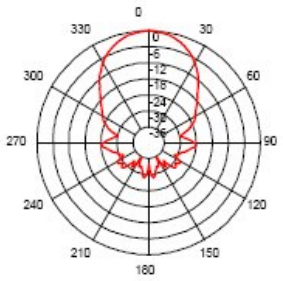

Abbildung 10.1.: Richtdiagramm

10.1.1. Schaltungen

Um das Signal ausreichend zu verstärken, sind auf Seiten des Senders und des Empfängers analoge Verstärkerschaltungen nötig. Diese werden im Folgenden beschrieben.

[1] aus [2]

10.1.2. Senderverstärkung

Da das Signal zur Ansteuerung des Senders durch den Mikrocontroller erzeugt wird[2], hat es nur eine Amplitude zwischen $0V$ und $5V$. Dies hat sich im Laufe einiger Test allerdings als zu gering herausgestellt. Daher muss das Signal auf Seiten des Senders verstärkt werden. Dies geschieht über eine Emitterschaltung.

Durch die in Abbildung 10.2 dargestellte Emitterschaltung findet eine Pegelwandlung von $0V/5V$ auf $0V/12V$ statt. R1 und R2 sind je $1k\Omega$ groß. Als Transistor wird ein BC547B verwendet.

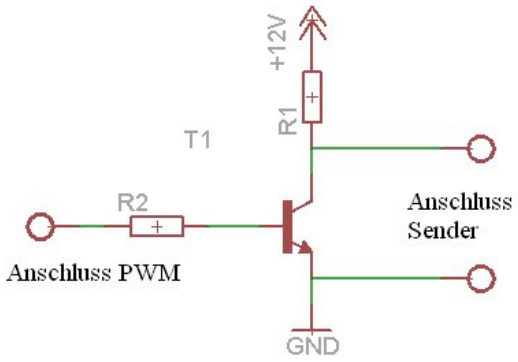

Abbildung 10.2.: Senderverstärkung

10.1.3. Empfängerverstärkung

Zur Verstärkung des Empfängersignals wurden im Laufe des Projekts zwei verschiedene Verstärkerschaltungen aufgebaut. Die erste arbeitet mit Operationsverstärkern, die zweite mit einem Transistor.

Empfängerschaltung mit Operationsverstärkern

Diese Variante arbeitet mit insgesamt drei Verstärkerstufen. Die erste Verstärkerstufe besteht aus einem Operationsverstärker, zwei Kondensatoren und zwei Widerständen. Diese bilden einen aktiven Bandpass (Abbildung 10.3) mit einer Verstärkung von ungefähr 9.

Als zweite Verstärkerstufe wird wiederum ein aktiver Bandpass eingesetzt mit einer Gesamtverstärkung von ungefähr 19 (Abbildung 10.4).

[2]Siehe Kapitel 12.2.1

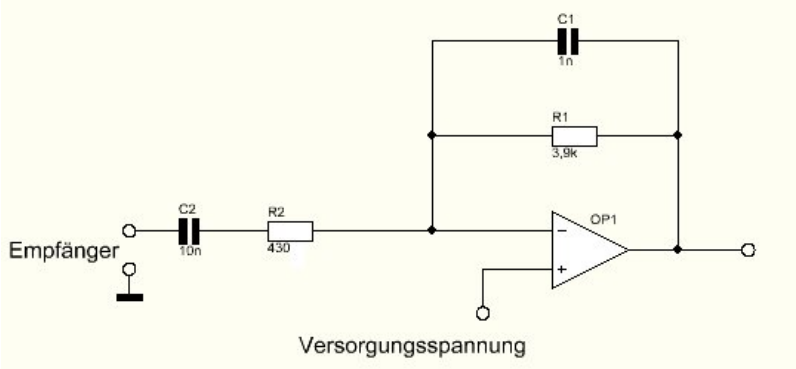

Abbildung 10.3.: 1. Verstärkerstufe

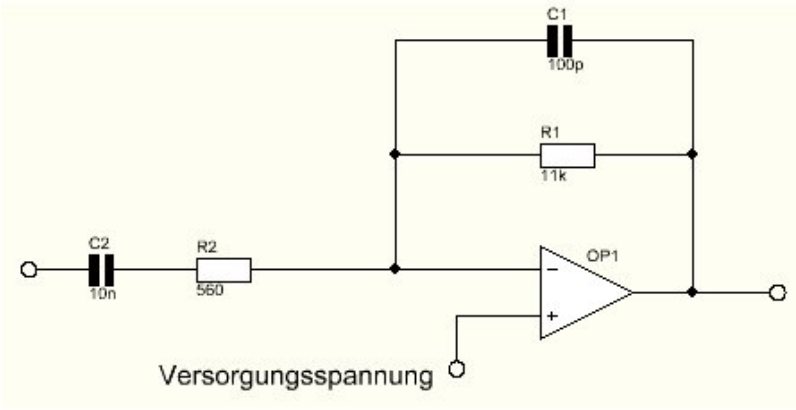

Abbildung 10.4.: 2. Verstärkerstufe

Als drittes wird noch ein Impedanzwandler eingesetzt (Abbildung 10.5).

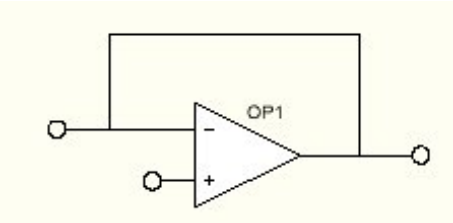

Abbildung 10.5.: Impedanzwandler

Empfängerschaltung mit einem Transistor

Diese Empfängerschaltung ist sehr leicht aufzubauen und bietet ebenfalls eine sehr gute Verstärkung des Signals. Auch die Signalqualität ist gut.

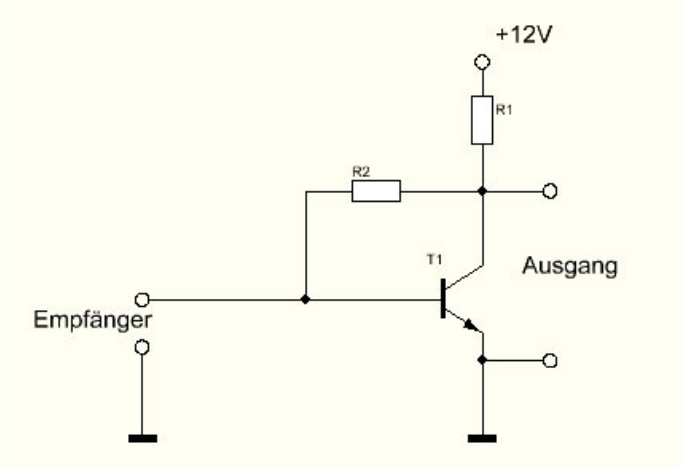

Abbildung 10.6.: Transistor Empfängerschaltung

10.2. Versuchsaufbau

Der Versuchsaufbau ist auf Abbildung 10.7 zu sehen.

Im Hintergrund ist die Elektronik inklusive Mikrocontroller auf einem Bread Board zu sehen. Im Vordergrund sind die Ultraschallsensoren zu erkennen. Diese sind fest auf einem

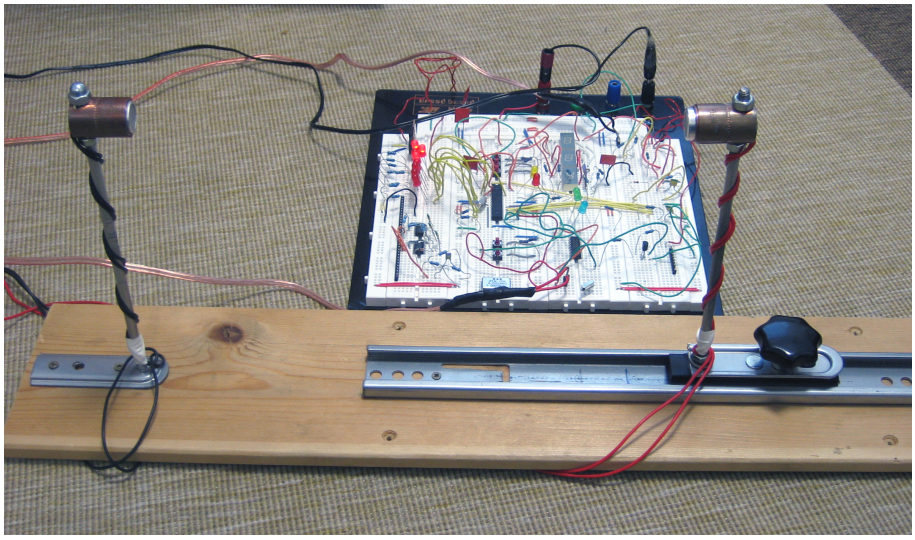

Abbildung 10.7.: Versuchsaufbau

Brett montiert. Der Sender ist zudem auf einer beweglichen Schiene angebracht, so dass die Entfernung zwischen Sender und Empfänger zwischen 11*cm* und 54*cm* variiert werden kann (Abbildung 10.8).

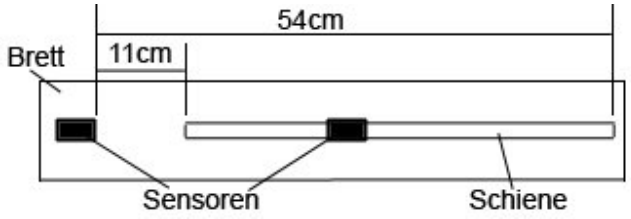

Abbildung 10.8.: Messvorrichtung

11 Mögliche Realisierungen

Zur Realisierung der Laufzeitmessung wurden zwei unterschiedliche Verfahren getestet. Zum einen wird ein Impulsverfahren beschrieben, zum anderen wird auf die Laufzeitmessung mittels Phasendifferenz eingegangen.

11.1. Impulsverfahren

Bei diesem Verfahren wird ein Ultraschallimpuls von einigen Millisekunden Länge ausgesendet. Gleichzeitig wird im Mikrocontroller ein Timer gestartet. Sobald am Empfänger ein Signal detektiert wird, wird der Timer gestoppt. Der gemessene Wert gibt Aufschluss über die Laufzeit des Signals. Der Wert wird auf acht Leuchtdioden als Binärwert ausgegeben. Probleme dieser Messmethode werden in Kapitel 11.1.2 erläutert.

11.1.1. Versuchsdurchführungen

In diesem Kapitel werden verschiedene Versuchsdurchführungen beschrieben, die als Funktionstest für die Laufzeitmessung dienen.

Entfernungsmessungen

Die Entfernungsmessungen werden bei völliger Windstille und gleicher Umgebungstemperatur durchgeführt, um gleich bleibende Testbedingungen zu erhalten. Die Umgebungstemperatur beträgt ungefähr 22°C. Nach Formel 9.1 ergibt sich damit eine Schallgeschwindigkeit von ca. $343\frac{m}{s}$. Der Abstand zwischen Sender und Empfänger wird in $2cm$ Schritten von $12cm$ bis $34cm$ variiert. Tabelle 11.1 zeigt einen Vergleich zwischen den errechneten und den tatsächlich gemessenen Werten. Bei den gemessenen Werten handelt es sich um Mittelwerte aus jeweils 50 Messungen.

In Abbildung 11.1 ist der Vergleich nochmals grafisch dargestellt. Es lässt sich ein annähernd linearer Verlauf der gemessenen Kurve (rot) erkennen. Die gelbe Kurve stellt den errechneten Wert dar. Im Durchschnitt liegen die gemessenen Werte um $293,14\mu s$ über den berechneten Werten.

Tabelle 11.1.: Entfernungsmessung

Entfernung	*gemessener Wert*	*berechneter Wert*
$12cm$	$608\mu s$	$348,45\mu s$
$14cm$	$670\mu s$	$406,52\mu s$
$16cm$	$765\mu s$	$464,60\mu s$
$18cm$	$814\mu s$	$522,67\mu s$
$20cm$	$880\mu s$	$580,75\mu s$
$22cm$	$947\mu s$	$638,82\mu s$
$24cm$	$979\mu s$	$696,90\mu s$
$26cm$	$1060\mu s$	$754,97\mu s$
$28cm$	$1107\mu s$	$813,05\mu s$
$30cm$	$1163\mu s$	$871,12\mu s$
$32cm$	$1242\mu s$	$929,20\mu s$
$34cm$	$1297\mu s$	$987,27\mu s$

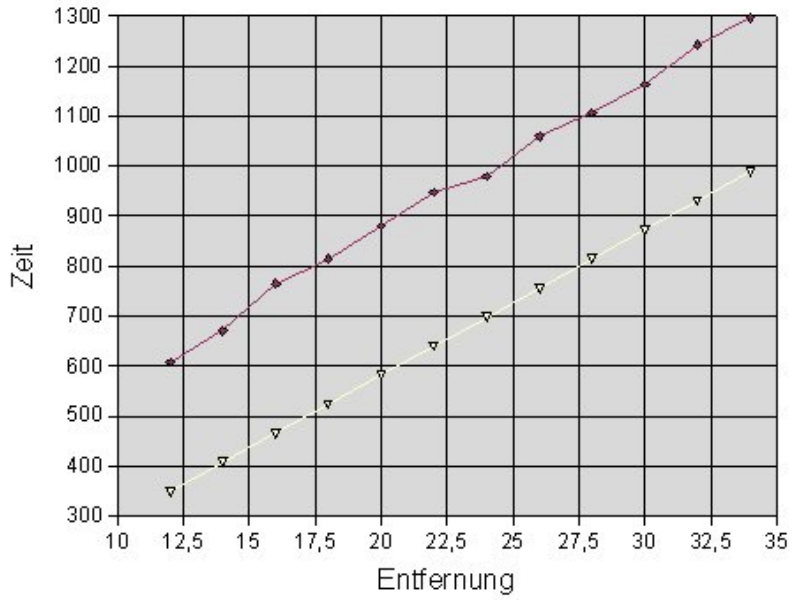

Abbildung 11.1.: Entfernungsmessung

Rechnet man den Offset runter, so stimmen der gemessene und der berechnete Verlauf schon sehr gut überein. Mit dem Verfahren ließe sich also mit einer ungefähren Genauigkeit im Zentimeterbereich eine Entfernungsmessung durchführen.

Der Temperatureinfluss ist da schon deutlich ungenauer. Tabelle 11.2 zeigt einen Vergleich zwischen berechneten und gemessenen Laufzeiten bei Temperaturen zwischen 20°C und 60°C in einem Intervall von $5K$, bei einer Entfernung von Sender zu Empfänger von $46cm$.

Tabelle 11.2.: Temperaturmessung

Temperatur	*Gemessene Laufzeit*	*Berechnete Laufzeit*
20°C	$1692\mu s$	1340,27
25°C	$1644\mu s$	1328,99
30°C	$1660\mu s$	1317,98
35°C	$1580\mu s$	1307,24
40°C	$1484\mu s$	1296,77
45°C	$1516\mu s$	1286,54
50°C	$1452\mu s$	1276,55
55°C	$1516\mu s$	1266,78
60°C	$1436\mu s$	1257,24

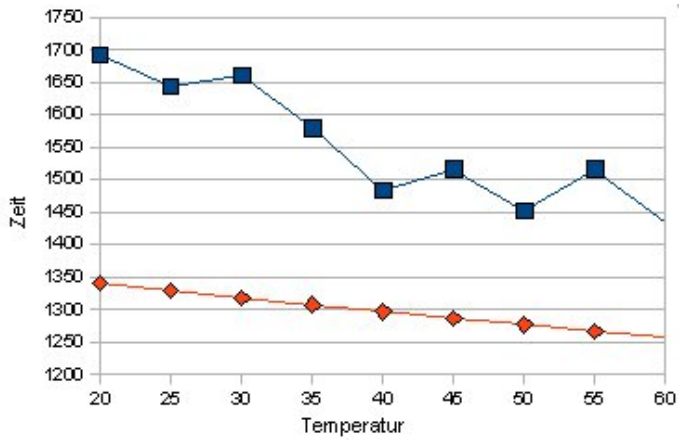

Abbildung 11.2.: Temperaturmessung

Auf Abbildung 11.2 lässt sich deutlich erkennen, dass die Gerade der gemessenen Laufzeit (blau) zwar fällt, aber zwischendurch gibt es immer wieder lokale Maxima. Daher

wäre die Messmethode für eine Temperaturmessung nicht geeignet. Die rote Gerade stellt die berechneten Werte dar. Auch hier ist ein Offset zu erkennen.

Windgeschwindigkeitsmessung

Um eine gleichbleibende Windgeschwindigkeit zu erhalten, wird der Versuchsaufbau auf einem Rollbrett aufgebracht. Das Rollbrett wird nun mit den Sensoren mit konstanter Geschwindigkeit einen Flur entlang gezogen, so dass die Sensoren, einem gleichmäßigem, langsamem Windzug ausgesetzt sind. Nach mehreren Versuchen zeigt sich jedoch, dass eine geringe Geschwindigkeit von wenigen m/s keinen erheblichen Einfluss auf die Messwerte hat. Aus diesem Grund sollen nun höhere Windgeschwindigkeiten getestet werden.

Dazu werden die Sensoren auf einem Autodach montiert. Bei nahezu Windstille und einer Temperatur von 12°C werden verschiedene Geschwindigkeiten getestet.

Es wurden bei verschiedenen Geschwindigkeiten Messungen durchgeführt. Die erste Messung fand im Stand statt. Die nächste bei einer Geschwindigkeit von ungefähr $10\frac{km}{h}$. Dann bei $20\frac{km}{h}$, $30\frac{km}{h}$, $50\frac{km}{h}$ und $70\frac{km}{h}$. In der Tabelle 11.3 sind die jeweils gemessenen Laufzeiten (Geschwindigkeit umgerechnet in $\frac{m}{s}$) angegeben.

Tabelle 11.3.: Windgeschwindigkeitseinfluss

Geschwindigkeit	*Gemessene Laufzeit*
$0,00\frac{m}{s}$	$1440\mu s$
$2,78\frac{m}{s}$	$1472\mu s$
$5,56\frac{m}{s}$	$1440\mu s$
$8,33\frac{m}{s}$	$1392\mu s$
$13,89\frac{m}{s}$	$1376\mu s$
$19,44\frac{m}{s}$	$1312\mu s$

In Abbildung 11.3 sind die Messwerte nochmals grafisch dargestellt. Deutlich zu erkennen ist, dass die Laufzeit erst ein wenig ansteigt und erst mit weiterem Anstieg der Geschwindigkeit abfällt. Die Kurve ist nicht linear.

Bei dieser Versuchsanordnung ist allerdings zu beachten, dass sie von vielen Fehlerfaktoren beeinflusst wird. Zum Beispiel kann es im Freien selbst bei Windstille zwischendurch kleinere Winde geben. Diese wirken ebenso auf die Messung, wie der gewünschte Fahrtwind des Autos. Es kann also nicht hundertprozentig von absoluter Windstille ausgegangen werden. Des weiteren entstehen am Auto Wirbel, die die gemessene Windgeschwindigkeit ebenso beeinflussen.

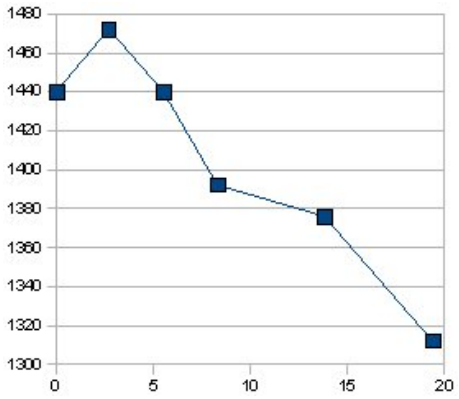

Abbildung 11.3.: Wingeschwindigkeitseinfluss

Nichtsdestotrotz kann zusammenfassend gesagt werden, dass aufgrund des Anstiegs bei kleinen Geschwindigkeiten und des ungeradlinigen Abfalls der Laufzeiten die Messergebnisse zeigen, dass dieses Verfahren viel zu ungenau und daher unbrauchbar für die Windgeschwindigkeitsmessung ist.

11.1.2. Probleme

Die Versuche zeigen deutlich, dass das verwendete Verfahren eine viel zu geringe Auflösung aufweist und somit für die gewünschte Geschwindigkeitsmessung unbrauchbar ist. Ein wesentliches Problem liegt in der Bandbreite des eingesetzten Empfängers. Laut Datenblatt liegt diese bei $2,5kHz$. Wird ein Impuls empfangen, so schwingt sich der Empfänger erst ein. Es dauert also etwas, bis der Empfänger seine volle Amplitude erreicht hat.

Abbildung 11.4 zeigt das gesendete Ultraschallsignal (gelb) und das zeitversetzt empfangene Signal (blau). Es lässt sich erkennen, dass auf der linken Seite das Ultraschallsignal beginnt. Nach einer gewissen Laufzeit wird daraufhin am Empfänger das Signal empfangen. Sehr gut zu sehen ist bei dem empfangenen Signal der Einschwingvorgang von mehreren Perioden.

Dieser Einschwingvorgang ist auf Abbildung 11.5 nochmals verdeutlicht dargestellt. Es werden ungefähr 12 Schwingungen benötigt, bis der Empfänger seine volle Amplitude erreicht. Dies hat bei der Detektion des Signals am Mikrocontroller teilweise erhebliche Messfehler zur Folge. Der Mikrocontroller erkennt ein anliegendes Signal, wenn dieses einen gewissen Spannungswert überschreitet. Durch das Einschwingen kann es nun allerdings vorkommen, dass ein ankommender Impuls einmal schon einige Schwingungen früher, bei

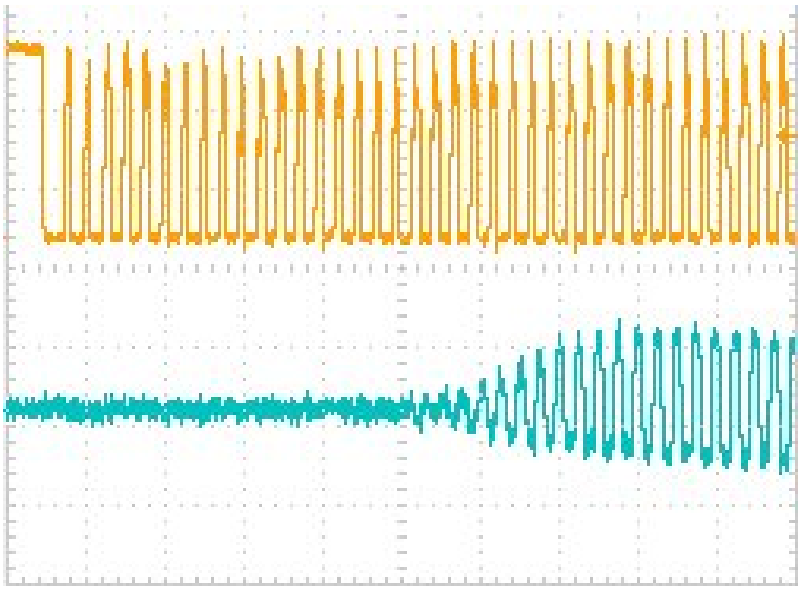

Abbildung 11.4.: Sendensignal und Empfangssignal

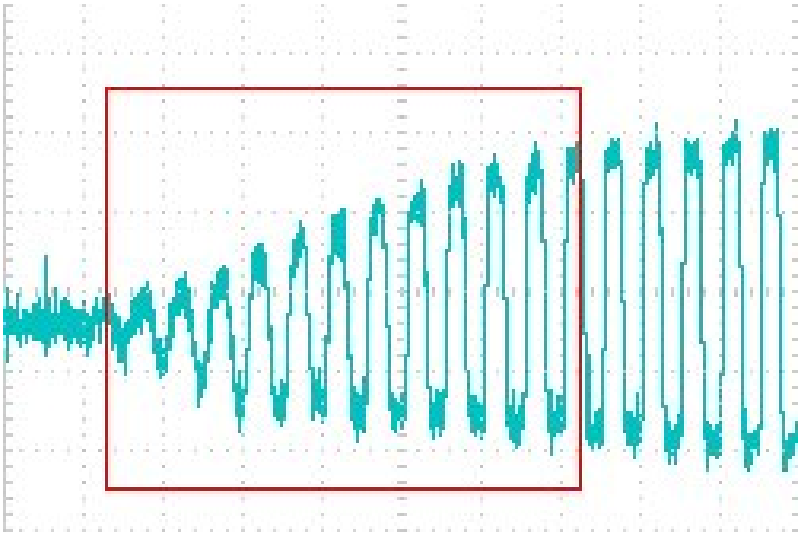

Abbildung 11.5.: Einschwingen

einer weiteren Messung allerdings erst einige Schwingungen später erkannt wird. Bei einer Frequenz von 40kHz beträgt die Schwingungsdauer $25\mu s$. Das bedeutet, dass es schon bei einer einzigen Schwingung, die das Signal später erkannt wird, zu einem Messfehler von $25\mu s$ kommt. Geringe Windgeschwindigkeiten rufen jedoch Veränderungen im μs-Bereich hervor. Es ist ersichtlich, dass auf diese Weise keine Windgeschwindigkeitsmessung vorgenommen werden kann.

Verbesserungsansätze

Ein möglicher Lösungsansatz, um dieses Problem in den Griff zu bekommen, liegt in der Verwendung von Sensoren mit höhere Bandbreite. Ob dies jedoch zum gewünschten Erfolg führt, bleibt auszutesten.

Ein weiterer Ansatz liegt in der Verwendung eines Hüllkurvedemodulators. Dieser wird hinter der Empfängerschaltung eingesetzt und dient dazu, die Einhüllende der Schwingung zu bilden. Eingesetzt wird ein einfacher Hüllkurvendemodulator, bestehend aus einer Diode, einem Kondensator und einem Widerstand (Abbildung 11.6).

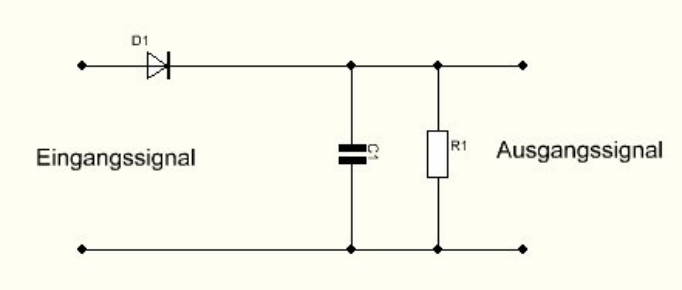

Abbildung 11.6.: Hüllkurvendemodulator

Abbildung 11.7 zeigt das Empfangssignal (blau) bei der Verwendung eines Hüllkurvendemodulators.

Weitere Laufzeitmessungen zeigen zwar nun eine leichte Änderung des Ergebnisses, jedoch ist diese nur sehr schwach. Die Verwendung eines Hüllkurvendemodulators führt somit auch nicht zu den gewünschten Ergebnissen.

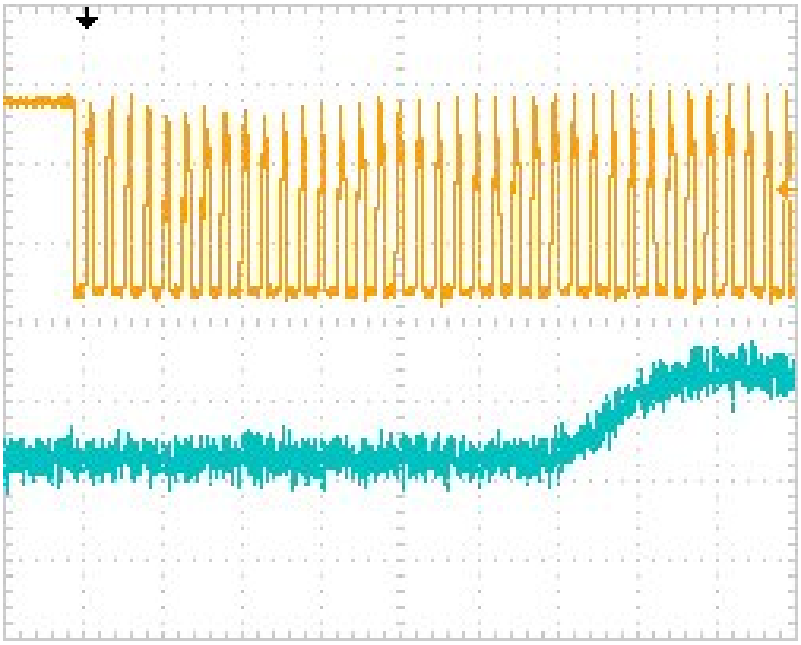

Abbildung 11.7.: Hüllkurvendemodoliertes Empfangssignal

11.2. Phasendifferenz

Da das Impulsverfahren nicht zu den gewünschten Ergebnissen führt, muss ein anderes Messprinzip herangezogen werden. Dieses Kapitel befasst sich mit dem Verfahren der Phasendifferenzmessung.

11.2.1. Grundlagen

Bei diesem Verfahren wird ein kontinuierliches Ultraschallsignal ausgesendet und am Empfänger die Phasendifferenz gemessen. Es wird also das ausgesendete Signal mit dem empfangenen Signal kontinuierlich verglichen, um eine Phasenverschiebung festzustellen.

Abbildung 11.8 zeigt jeweils das vom Mikrocontroller erzeugt Rechtecksignal (gelb) und das empfangene Signal (blau). Dabei sind die beiden Signale in b) zu denen in a) um ungefähr 90° Phasenverschoben. Die Phasenverschiebung wurde hier durch Veränderung der Entfernung von Sender und Empfänger hervorgerufen.

Gemessen werden kann die Phasenverschiebung, indem bei dem ausgesendeten Signal ein Timer bei einem Nulldurchgang beispielsweise einer fallenden Flanke angestoßen wird.

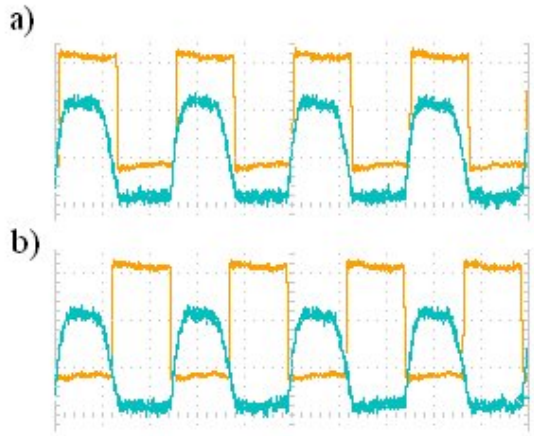

Abbildung 11.8.: 90° Phasenverschiebung

Der Timer wird gestoppt, sobald beim empfangenen Signal eben dieser Nulldurchgang detektiert wird. Über die gemessene Zeit kann auf die Phasendifferenz geschlossen werden. Allerdings hat die Temperatur auch bei diesem Messprinzip Einfluss auf die Messung. Wie der Temperatureinfluss eliminiert werden kann, wird in Kapitel 13.1 beschrieben.

11.2.2. Zeitanforderungen

Um die Zeitanforderungen für die Phasendifferenzmessungen abschätzen zu können, wird mit Hilfe eines Oszilloskops die Phasendifferenz bei einem leichtem Windzug sichtbar gemacht. Der Windzug wird durch eine gleichmäßige Bewegung der Messanordnung hervorgerufen.

Bei genauerer Betrachtung von Abbildung 11.9 fällt die Phasenverschiebung zwischen dem Signal in a) und dem in b) (rot eingezeichnet) auf. Die Messanordnung wurde mit geringer Geschwindigkeit (wenige $fracms$) bewegt. Bei einer Schwingungsdauer von $25\mu s$, lässt sich ein Zeitunterschied von weniger als $2\mu s$ erkennen. Die geforderte Genauigkeit ist mit dem Atmega8 alleine nicht zu realisieren, da die verwendbaren Zähler für diesen Zweck zu langsam arbeiten.

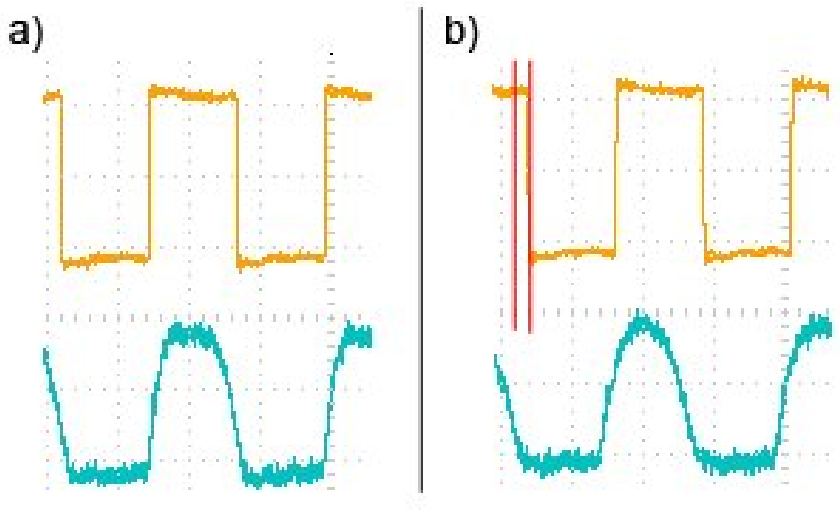

Abbildung 11.9.: Phasendifferenz bei leichtem Windzug

11.2.3. Realisierung (Ausblick)

Um eine Phasenverschiebung mit einer so hohen Genauigkeit zu messen, wie es die Windmessung erfordert, gibt es mehrere Lösungsansätze.

Ein möglicher Lösungsansatz besteht in einer speziellen Schaltung zur Phasendetektion. Dazu werden die Signale des Senders und Empfängers jeweils auf Komparatoren gegeben, mit denen die Nulldurchgänge detektiert werden können. Abbildung 11.10 verdeutlicht die Funktionsweise der Komparatoren.

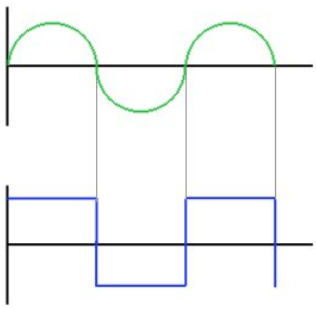

Abbildung 11.10.: Funktionsweise Komparator

Bei einem Nulldurchgang wird der Komparatorausgang auf „High“ geschaltet. Bei ei-

nem erneuten Nulldurchgang, schaltet er auf „Low“ zurück. Das Problem hierbei ist jedoch die genaue Erkennung von Nulldurchgängen. Ist das Signal verrauscht, so kann ein mehrmaliges Schalten erfolgen. Gelöst werden kann dieses Problem durch eine Hysterese. Ein Komparator mit Hysterese ist ein Schmidt-Trigger. [1]

Nach den Komparatoren erfolgt eine Flankenauswertung. Beispielsweise wird durch eine positive Flanke des Komparators des Sendesignals ein RS-FlipFlop gesetzt. Eine positive Flanke, die vom empfangenen Signal herrührt, führt dagegen zu einem Rücksetzen des FlipFlops. Die Zeit, die der Ausgangs des FlipFlops auf „High“ steht, ist ein direktes Maß für die Phasenverschiebung. Diese Zeit kann durch einen entsprechend schnellen Zähler gemessen werden. Ein solcher Zähler ist z. B. der 74LV4020, der mit bis zu $100MHz$ zählen kann. Der Wert des Zählers kann dann vom Mikrocontroller ausgelesen werden.

Ein weiterer Ansatz liegt darin, den bisherigen Versuchsaufbau durch ein FPGA zu ersetzen. Da diese eine viel höhere Geschwindigkeit bieten, stellt eine Phasenauswertung von der geforderten Genauigkeit kein Problem dar.

12 Quellcode

Beschrieben wird in diesem Kapitel nur der Quellcode zu dem Impulsverfahren, da die Phasendifferenzmessung mit dem Atmega8 bisher nicht brauchbar funktionierte.

Die ersten Versuche, das Empfangssignal zu detektieren wurden noch über den Analog-Digital-Converter des Atmega8 ausgeführt. Das Aussenden des Signal erfolgte in gleicher Weise wie bei dem hier beschriebenen Verfahren. Nach dem Aussenden wird ein Timer gestartet, der die Laufzeit misst. Am ADC wird nun gemessen, ob ein bestimmter Schwellwert überschritten wird. Ist dies der Fall, so wird der Timer gestoppt. Das Problem bei diesem Verfahren ist jedoch, dass der ADC für jede Wandlung auch eine gewisse Zeit benötigt. Dies führt zu weiteren Messfehlern.

Die hier gezeigte Variante funktioniert über einen externen Interrupt, der von dem empfangenen Signal ausgelöst wird. Der Quellcode ist mit Kommentaren versehen, die die einzelnen Abschnitte ausreichend beschreiben. In einigen Fällen wird der Quellcode nochmals ausführlich beschrieben.

12.1. Interruptaufrufe

Hier werden kurz die verwendetetn Interrupt-Service-Routinen (ISR) beschrieben. Je nachdem, um welche ISR es sich handelt, führt eine andere Aktion zum Aufruf dieser Routinen.

```
ISR(TIMER0_OVF_vect)   //ISR für Überlauf des Timer0
{
  ueberlauf_timer0=1;  //bei Timer0-Überlauf wird diese Variable gesetzt
}

ISR(TIMER2_OVF_vect)   //ISR für Überlauf Timer2
{
  senden_zaehler++;    //eine Zähl Variable, die bei jedem Überlauf um
                       //1 inkrementiert wird
}

ISR (INT0_vect)        //ISR wird dur Interrupt-Pin 0 ausgelöst
{
  aufruf_interrupt0=1; //Variable, die beim Interrupt auf "1" gesetzt wird
}
```

12.2. Initialisierungsteil

```
void Initialisierung (void)
{
TCCR0 |= (1<<CS00);                 //Timer0 wird mit CPU Takt betrieben
TCCR2 |= (1<<CS02) | (1<<CS00) ;    //Timer 2 mit 1/1024 CPU Takt
TIMSK |= (1<<TOIE0) | (1<<TOIE2);   //Aktivierung des Timer Overflow
                                    //Interrupts für Timer 0 und Timer 2

DDRD = (1<<DDD0) | (1<<DDD1) | (1<<DDD3) | (1<<DDD4) | (1<<DDD5) | (1<<DDD6) |
    (1<<DDD7);
DDRC = (1<<DDC2);       //Setzen der Ausgänge für LED Ausgabe

/*PWM siehe Kapitel 12.2.1*/
TCCR1A = (1<<COM1A1) | (1<<COM1A0) | (1<<WGM11) ;
TCCR1B = (1<<WGM12) | (1<<WGM13) | (1<<CS11);
OCR1A = 25;
ICR1 = 50;

MCUCR |= (1<<ISC01);    //fallende Flanke am Interruptpin erzeugt Interrupt
GICR |= (1<<INT0);      //Interrupt für Interuptpin 0 aktivieren
```

12.2.1. Erzeugung eines 40kHz Signals

Das benötigte $40kHz$-Signal für die Ansteuerung des Ultraschallsenders, wird durch Pulsweitenmodulation (PWM) des Atmega8 erzeugt. Bei der PWM wird ein Zähler in einer besonderen Betriebsweise verwendet. Er zählt bis zu einem definierten (Vergleichs-)Wert und setzt dann einen Ausgang auf „High“. Nun zählt er weiter bis ein vorgegebener Endwert erreicht wird. Von da an zählt er wieder runter bis „0“. Beim Überschreiten des Vergleichswerts wird der Ausgang wieder auf „Low“ gesetzt. Die verwendete PWM arbeitet allerdings noch ein bißchen anders. Sie zählt nicht vom oberen Wert wieder runter, sondern setzt bei erreichen des Endwerts den Zähler sofort wieder auf „0“. Zudem wird der Ausgang beim Erreichen des Endwertes wieder auf „Low“ gesetzt. Hierbei handelt es sich um eine „Fast-PWM“. Nicht alle Timer des Atmega8 können für eine PWM verwendet werden. Daher wird der „Timer1“ verwendet.

```
TCCR1A = (1<<COM1A1) | (1<<COM1A0) | (1<<WGM11);
TCCR1B = (1<<WGM12) | (1<<WGM13) | (1<<CS11);
OCR1A = 25;
ICR1 = 50;
```

Durch das Setzen von COM1A1 und COM1A0 im Register TCCR1A wird die PWM Betriebsart ausgewählt, so dass der Ausgang beim Hochzählen auf „high“ und beim runterzählen auf „low“ gesetzt wird. WGM11, WGM12 und WGM13 im Register TCCR1A und TCCR1B bewirken, dass zum Vergleichswert noch ein Endwert (ICR1), bis wohin der Zähler also laufen soll, angegeben werden kann. Das setzen des Bits CS11 hat zur Folge, dass der Zähler mit $\frac{1}{8}$ des CPU-Taktes (also $2Mhz$) läuft. OCR1A ist der Vergleichswert und ICR1 der Endwert. Der Zähler zählt

also immer von 0 bis 50, setzt bei 25 den Ausgang auf „high“ und bei 50 auf „low“ (siehe Abbildung 12.1).

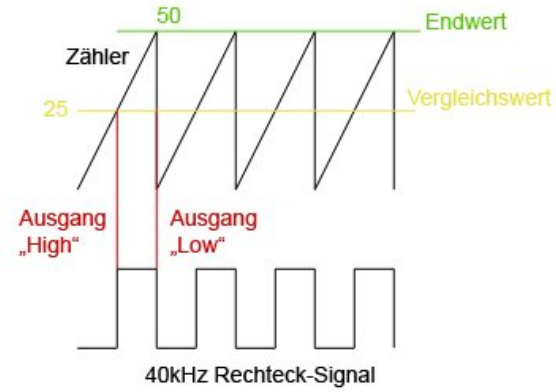

Abbildung 12.1.: Pulsweitenmoduliertes 40kHz Signal

12.3. Senderoutine

```
void sende_us_signal (void)
{
  TCNT0 = 0;
  PORTD=0x00;
  PORTC=0x00;
  cli();          // Interrupts deaktivieren
    TCNT1H=0x00;
    TCNT1L=0x00;
  sei();          // Interrupts wieder aktivieren
}
```

In der Senderoutine wird der Zählerstand von Timer0 auf „0“ zurückgesetzt, weil dieser sonst immer von einem anderen Wert aus startet, da er im Hintergrund weiter läuft. Ebenso werden die Zählerstände von Timer1, der für die PWM zuständig ist, auf „0“ zurückgesetzt. Dabei muss die Reihenfolge unbedingt beibehalten werden. Außerdem darf beim Schreiben auf die Register kein Interrupt ausgelöst werden, daher werden die Interrupts deaktiviert.

12.4. Hauptprogrammteil

```
int main (void)
{
        Initialisierung ();     //Aufruf der Initialisierungsfunktion
```

```
while(1)
{
  if (senden_zaehler==20)
  {
    senden_zaehler=0;
    pulsdauer_zaehler = 120;
    sende_signale = 1;
    sende_us_signal ();       //ruft das Unterprogramm auf
  }
```

Oben stehender Abschnitt dient der Erzeugung eines Ultraschallimpulses. Die Variable „senden_zaehler“ wird durch den Timer2 inkrementiert. Wenn diese den Wert 20 erreicht hat, wird die Routine zum Senden des Ultraschallsignal gestartet. Je höher der Wert dieser Variable für die bedingte Anweisung gewählt wird, desto mehr Zeit liegt zwischen den einzelnen Ultraschallimpulsen. Über die Variable „pulsdauer_zaehler“ kann die Länge eines jeden Ultraschallimpulses angegeben werden. „sende_signale“ dient dazu, in einer Variable den aktuellen Status zu haben, ob gerade ein Signal gesendet wird und dieses noch nicht detektiert wurde (zur Berechnung der Laufzeit des Signals). Diese Variable wird nach dem Senden wieder zurück gesetzt.

```
  if (aufruf_interrupt0==1)        //wird bei Auslösen eines Interupts an
  {                               //Interrupt Pin 0 aufgerufen
    sende_signale = 0;
    empfang_beendet = 1;
    aufruf_interrupt0=0;
  }
  if (ueberlauf_timer0==1)        //bei timer0 interrupt
  {
    if (pulsdauer_zaehler >= 0)  //Dekremtierung für die Pulsdauer
    {
      pulsdauer_zaehler--;
    }
    if (sende_signale==1)         //Messung der Laufzeit
    {
      laufzeit++;
    }
    ueberlauf_timer0=0;
  }
  if (empfang_beendet)           //wird entweder durch eine erfolgreiche
  {                              //Detektion des Signals oder nach
                                 //bestimmter Zeitdauer aufgerufen
    anzeige=laufzeit;            //zur Darstellung der Laufzeit auf den
                                 //LEDs
    empfang_beendet = 0;
    laufzeit=0;
  }

  if (pulsdauer_zaehler == 120) //Schalten des Ausgangs für die PWM
  {
    DDRB |= (1<<PB1);
  }
  if (pulsdauer_zaehler == 0)   //Rücksetzen des Ausgangs
  {
```

```
    DDRB &= ~(1<<PB1);
  }
  if (laufzeit>=512)            //bei Zeitüberschreitung
  {
    sende_signale = 0;
    empfang_beendet = 1;
  }
```

13 Weitere Überlegungen

13.1. Temperatureinfluss eliminieren

Ein wichtiger Punkt, um eine Windmessung durchführen zu können, ist, den Temperatureinfluss zu eliminieren. Wie bereits erwähnt hat die Temperatur einen deutlichen Einfluss auf die Laufzeit des Signals (Formel 9.1). Dieser Einfluss kann mit relativ geringem Aufwand beseitigt werden:

Wird ein Signal in gleicher Richtung, wie der Wind weht, ausgesendet, so verkürzt sich dessen Laufzeit. Wird es entgegen der Windrichtung ausgesendet, so verlängert sich die Laufzeit um den gleichen Betrag. Der Temperatureinfluss hingegen führt in beiden Fällen zu einer Verkürzung der Laufzeit. Nun kann also eine Messung in beide Richtungen durchgeführt werden und die Differenz aus den beiden Laufzeiten gebildet werden. Die Differenz ist dann ein Maß für die Windgeschwindigkeit ohne Temperatureinfluss.

13.2. 2 Dimensionale Messung

Bisher wurde nur eine 1D-Messung betrachtet. Um aber auch die Windrichtung feststellen zu können, ist eine 2D-Messung erforderlich.

Für diese sind vier Sensoren notwendig, die wie in Abbildung 13.1 zu sehen, angeordnet sind. Zunächst ist es wichtig, die Sensoren der Himmelsrichtung nach auszurichten. Im Beispiel zeigt die Y-Achse nach Norden. Weht nun ein Wind, so werden dessen x-Komponente (rot) und dessen y-Komponente (grün) von den Sensoren erfasst. Die resultierende Windstärke w kann demnach einfach nach Pythagoras zu

$$w = \sqrt{x^2 + y^2} \tag{13.1}$$

berechnet werden. Zur Berechnung der Windrichtung muss zuerst der Winkel φ bekannt

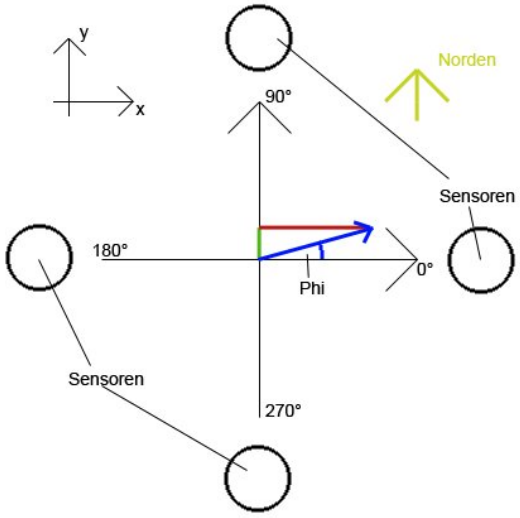

Abbildung 13.1.: 2D-Messung

sein. Um diesen zu berechnen, sind einige Fallunterscheidungen nötig:

$$\varphi = \arctan\left(\frac{y}{x}\right) \qquad \text{für } x > 0 \tag{13.2}$$

$$\varphi = \arctan\left(\frac{y}{x}\right) + 180^\circ \qquad \text{für } x < 0, y \geq 0 \tag{13.3}$$

$$\varphi = \arctan\left(\frac{y}{x}\right) - 180^\circ \qquad \text{für } x < 0, y < 0 \tag{13.4}$$

$$\varphi = +90^\circ \qquad \text{für } x = 0, y > 0 \tag{13.5}$$

$$\varphi = -90^\circ \qquad \text{für } x = 0, y < 0 \tag{13.6}$$

$$\tag{13.7}$$

Die so berechneten Winkel können nun in die jeweilige Himmelsrichtung umgerechnet werden. Dazu ist die geforderte Kenntnis der Ausrichtung der Sensoren zur Himmelsrichtung nötig.

Literatur

[1] http://www.elektronik-kompendium.de/sites/slt/0311261.htm, Abruf: 20.09.2008 57

[2] *Datenblatt Ulraschallsensoren.* http://www.reichelt.de/?;ACTION=7;LA=6;OPEN=1;INDEX=0;FILENAME=B400%252FUST%2523PRT.pdf, Abruf: 15.09.2008 41

[3] Vossiek, Prof. Dr.-Ing.: *Skript „Ultraschallsensorik“ zur Vorlesung Messtechnik II.* Institut für Elektrische Invormationstechnik, 37