Franz-Meyers-Gymnasium

Beschleunigungen messen mit Mikrocontrollern

Facharbeit im Leistungskurs

Physik

von

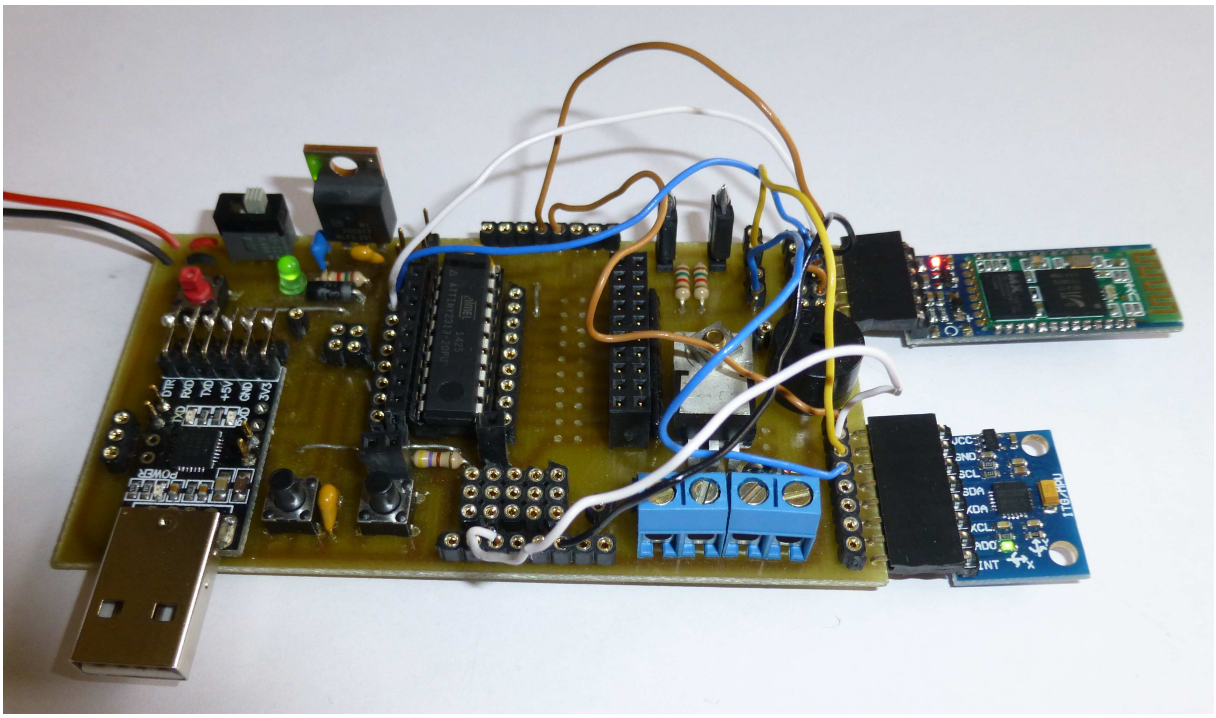
Lukas Jacobs

Schuljahr 2014/15

Inhaltsverzeichnis

1. Einleitung

Ziel dieser Facharbeit ist es, mit einem Mikrokontroller Beschleunigungen zu messen und auszuwerten. An der Attiny-Platine 3.0 schloss ich den 3 Achsen Beschleunigungssensor MPU6050 über I2C an. Die Daten werden mit Hilfe eines Bluetooth-Moduls, welches über RXD und TXD angeschlossen wird, an ein Handy oder einen PC übertragen.

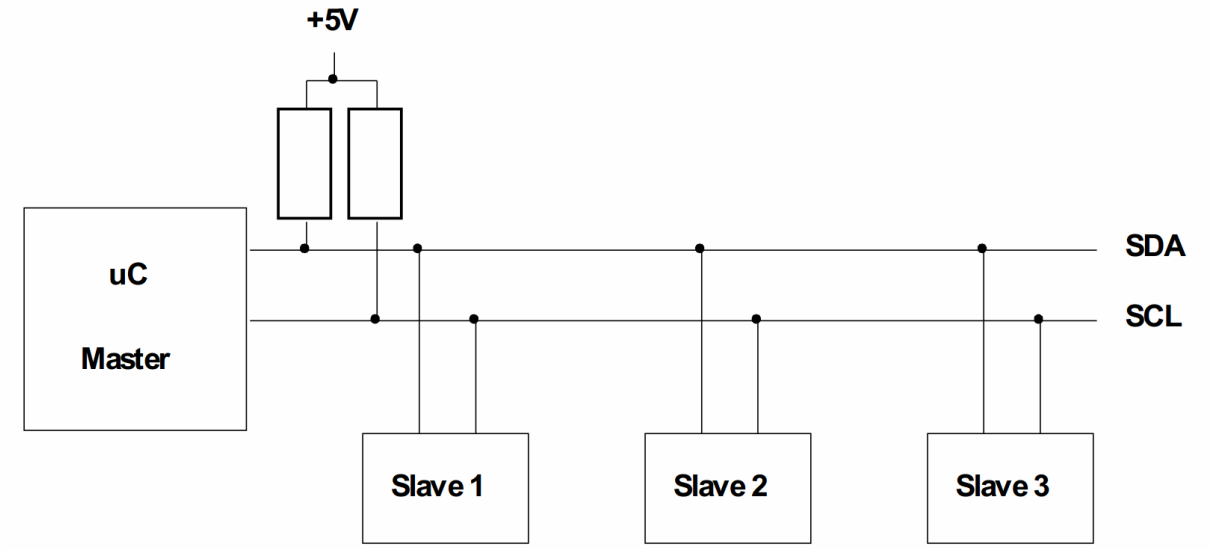


(Klick auf dieses und folgende Bilder, vergrößert das jeweilige Bild)

2. I2C

I2C ist eine Bustechnik zur Datenübertragung. Vorteil an dieser Art der Übertragung ist der geringere Bedarf an Leitungen. Jedoch ist die Bustechnik deutlich störanfälliger, als eine direkte Verbindung. Der Bus besteht aus zwei Leitungen. Diese sind einmal SCL als Taktgeber und SDA als Datenleitung. Beide Leitungen werden über einen Widerstand mit +5V verbunden. Dies führt dazu, dass sobald kein Baustein die Leitungen mit Masse verbunden und damit auf LOW gezogen hat, das Signal automatisch zu HIGH wechselt. Alle Geräte sind an beiden Leitungen angeschlossen. Der Master gibt über SCL den Takt vor. Jedes Gerät verfügt über eine 7 Bit Adresse, wobei sich meist die letzten Bits durch Jumper festlegen lassen. Es sind folglich bis 128 (=2^7) Geräte an einem Bus möglich. Die einzelnen Geräte besitzen mehrere Register, in denen jeweils ein Byte gespeichert ist. Manche davon lassen sich aber nur lesen und nicht beschreiben.

(Quelle: 1)



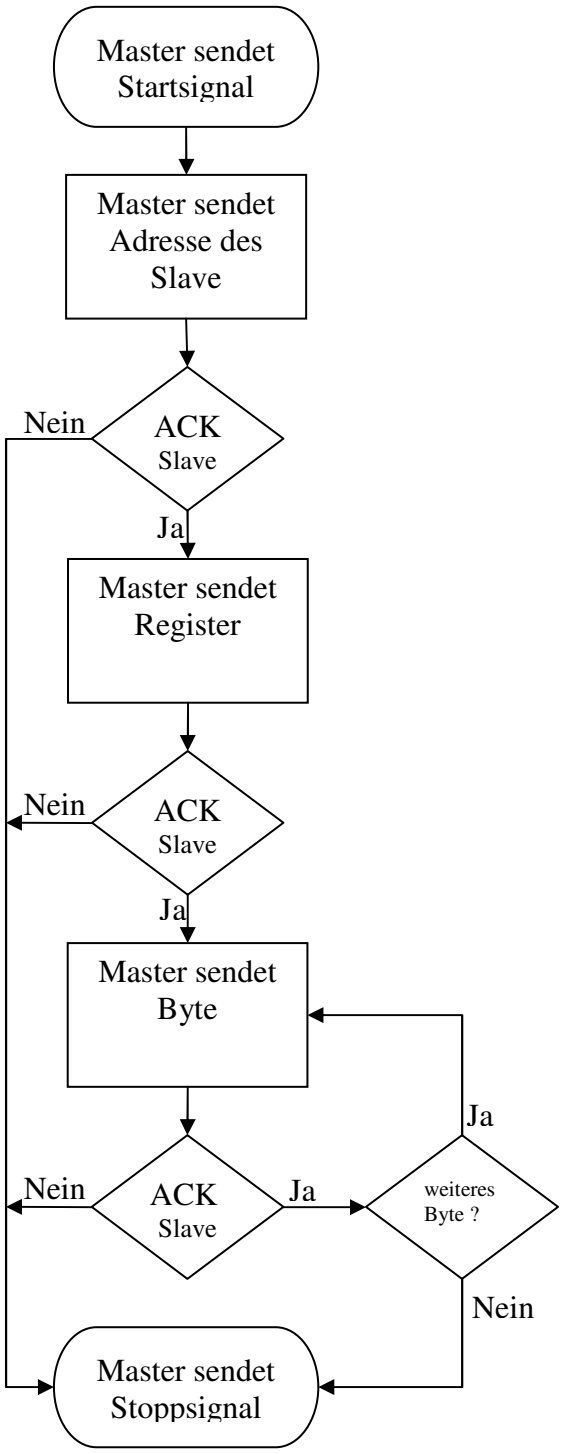
(Quelle: http://staff.ltam.lu/feljc/electronics/bascom/BASCOM_I2C.pdf Seite1)

2.1 Datenübertragung

Am Anfang einer Übertragung befinden sich alle Bausteine in einem Ruhemodus, SCL und SDA sind auf HIGH. Der Master startet nun die Übertragung durch ein Startsignal, dazu zieht der Master SDA auf LOW. Alle Bausteine beobachten nun den weiteren Datenverkehr. Nun überträgt der Master ein Byte. Die ersten 7 Bits des Bytes ist die Adresse des Slave, mit dem kommuniziert werden soll, das letzte Bit gibt an, ob der Slave ausgelesen oder beschrieben wird. 1 bedeutet der Master wird lesen, 0 bedeutet das geschrieben wird.
Bei der Übertragung ist SCL auf LOW. Sobald der Master SCL auf HIGH setzt, liest der Slave das Bit auf SDA ein. Auf diese Weise werden alle 8 Bits eingelesen. Den Empfang eines Bytes bestätigt der Slave durch ein ACK (Acknowledge), indem der Slave zum Takt des Masters SDA auf LOW setzt. (Sendet der Slave, so bestätigt natürlich der Master durch ein ACK.) Empfängt der Master bzw. der Slave ein NAC (No ACK), also kein ACK (SDA bleibt beim Takt auf HIGH), so ist je nach Situation ein Fehler aufgetreten oder der Slave bzw. Master möchte keine Bytes mehr empfangen.
Als nächstes wird das Register, welches ausgelesen oder beschrieben werden soll, gesendet. Bei diesen ersten beiden Bytes muss immer ein ACK zurückkommen, ansonsten ist ein Fehler aufgetreten. Wenn Daten an den Slave gesendet werden sollen, so überträgt der Master jetzt das Byte mit den eigentlichen Daten an den Slave. Sollen mehrere Bytes übertragen werden, so kann man diese einfach hintereinander übertragen. Kann der Slave keine weiteren Bytes mehr empfangen, so sendet dieser ein NAC. Nun wird die Datenübertragung durch ein Stoppsignal beendet, indem der Master SDA und SCL wieder dauerhaft auf HIGH setzt.

Sollen Daten aber vom Slave gelesen werden, so muss man jetzt erneut das Startsignal senden. Ebenfalls wird nun die Adresse erneut gesendet, jedoch mit einer 1 anstatt einer 0 als letztes Bit, um den Lesevorgang einzuleiten. Nun wird das zu lesende Register übertragen.
Der Slave sendet dann das angeforderte Byte. Durch Senden von ACK macht der Master deutlich, dass er bereit ist, das Byte des nächsten Registers zu empfangen, durch NAC, dass er keines mehr empfangen möchte. Auch hier wir die Datenübertragung durch das Stoppsignal beendet. (Quelle: 1, 5)

2.1.1 Schreibvorgang

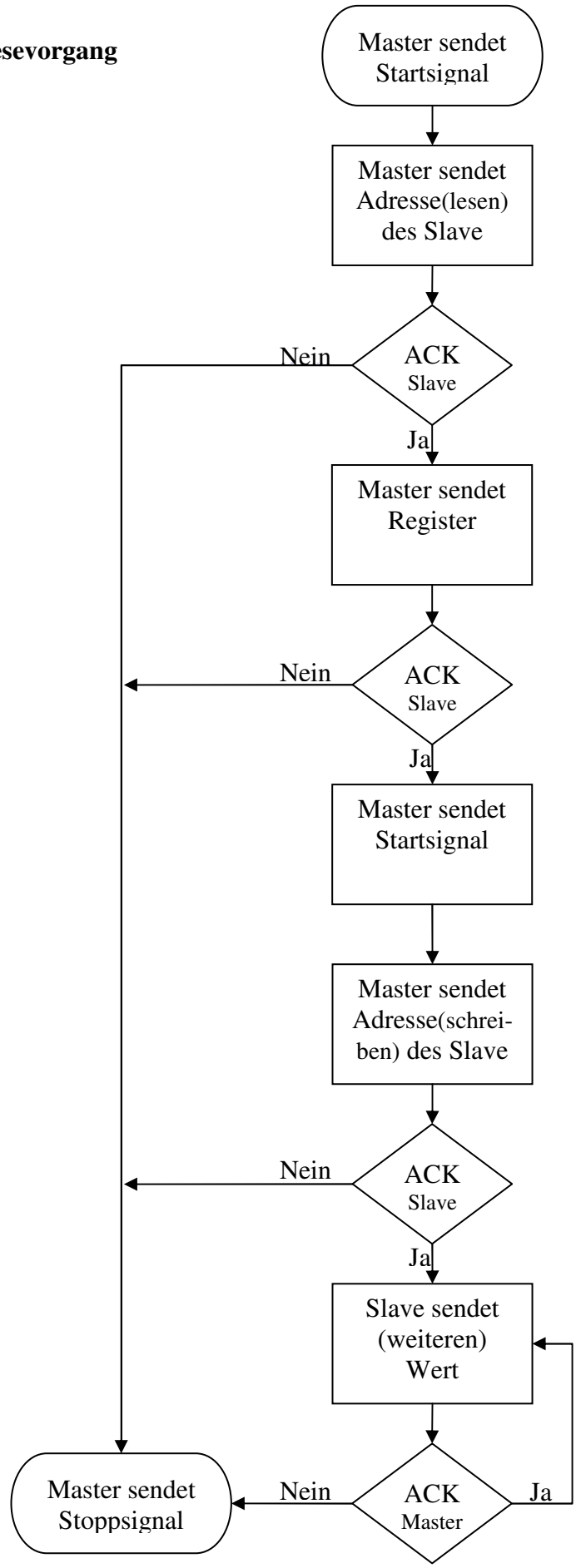


In Bascom sieht das Ganze dann so aus:

```
Sub Writei2c(adresse As Byte , Register As Byte , Byval Wert As Byte)
    I2cstart
    I2cwbyte adresse
    If Err = 0 Then                'überprüft ob Slave ACK gesendet hat
       I2cwbyte Register
       I2cwbyte Wert
     Else
       Print "Fehler Schreiben" ; adresse ; Register ; Wert
    End If
    I2cstop
  End Sub
```

(Quelle: 1)

2.1.2 Lesevorgang



In Bascom sieht das Ganze dann so aus:

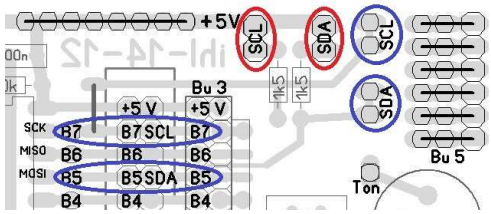
```
Function Readi2c(byval Register As Byte , Byval Adresse As Byte ) As Byte
    I2cstart
    I2cwbyte Adresse
    If Err = 0 Then
       I2cwbyte Register
       Adresse = Adresse + 1            'Adresse +1, wegen Lesen +1
       I2cstart
       I2cwbyte Adresse
       I2crbyte Readi2c , Nack          'keine weiteren Bytes sollen
     Else                                 'gesendet werden
        Print "Fehler Lesen " ;  Adresse ; Register
    End If
    I2cstop
  End Function
```

(Quelle: 1)

2.2 I2c an der Attiny Platine

Um mit der Attiny Platine 3.0 I2C nutzen zu können, müssen zunächst die beiden Jumper (rot markiert) gesetzt werden, damit die Pullwiderstände verwendet werden können. Danach können die Busleitungen an SCL und SDA (blau markiert) angeschlossen werden.

(Quelle: 4)



(Quelle: http://www.g-heinrichs.de/attiny/lageplan3.jpg)

3. Bluetooth

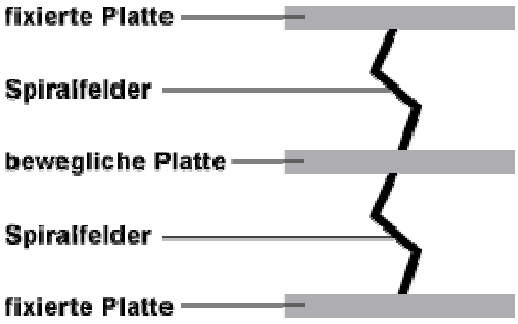
Da sich der Mikrocontroller bewegt, ist das Ablesen der Daten mit Hilfe eines LCD-Displays ungünstig. Ich entschlossen mich deshalb die Daten über Bluetooth zu übertragen. Dazu verwendete ich das Bluetooth-Modul HC-05 RS232. Vorteil an diesem Modul ist, dass es über die serielle Schnittstelle an den Mikrocontroller angeschlossen wird. Dies ermöglich eine einfache Programmierung, denn nach dem Anschließen wird dieser wie das PC-Terminal

mit den Befehlen `print` und `input` gesteuert. Da es sich vom PC-Terminal nicht unterscheiden lässt, sollte es nicht mit diesem gleichzeitig verwendet werden. Bei der Suche nach Bluetooth-Geräten wird das Modul als HC-06 angezeigt. Das Passwort für das Pairing lautet standartgemäß 1234.

Um die Daten auf meinem Android-Handy empfangen zu können, benutze ich den App Bluetooth Terminal. Der Vorteil an diesem Terminal ist, dass es auch den Zeitpunkt festhält und so das Auswerten der Daten erleichtert. Natürlich kann man hier auch eine andere App oder den PC verwenden.

4. Beschleunigungssensor

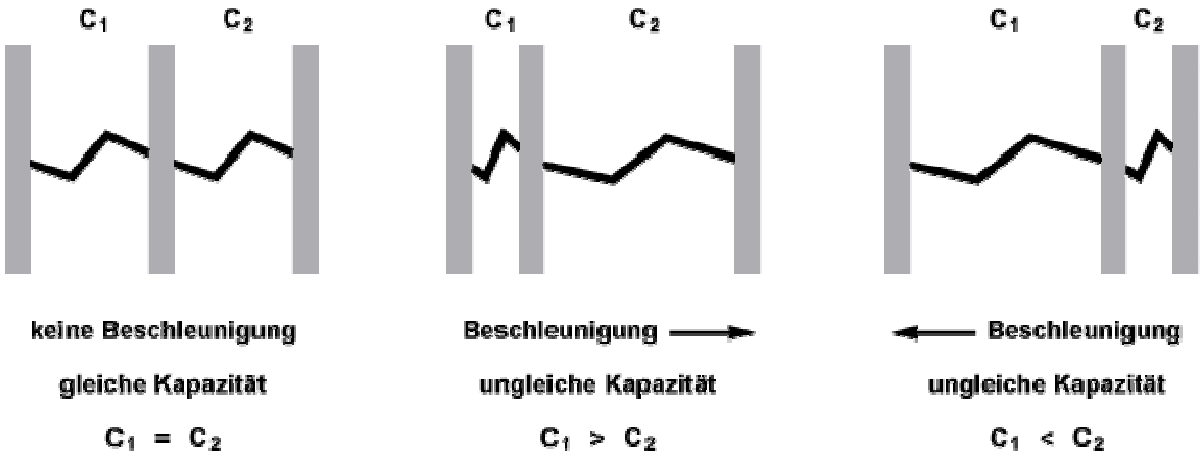
Es stellt sich natürlich auch die Frage, wie so ein Beschleunigungssensor eigentlich funktioniert. Es gibt verschiedene Systeme zur Bestimmung von Beschleunigungen.



(Quelle: http://www.elektronik-kompendium.de/sites/bau/bilder/15030411.gif)

Hier möchte ich jetzt das Micro-Electro-Mechanical System (MEMS) genauer erklären. Die Sensoren bestehen aus zwei festen Eisenplatten. Zwischen den Beiden befindet sich eine weitere Platte. Diese ist mit je einer Feder mit den umgebenden Platten verbunden und folglich freibeweglich.

Die bewegliche Platte bildet mit den anderen Beiden je einen Kondensator. Da die Platte beweglich ist, ändern sich die Abstände und folglich auch die Kapazitäten. Wird der Sensor zum Beispiel nach rechts beschleunigt (siehe untere Grafik Zustand C1 > C2), so bewegt sich die mittlere Platte, wegen der Trägheit der Masse, im Bezug zum Rest des Sensors, nach links. Folglich wird die Kapazität von C1 größer und die Kapazität von C2 kleiner, denn $C = \varepsilon * A/d$.



(Quelle:http://www.elektronik-kompendium.de/sites/bau/bilder/15030412.gif)

Erfolgt keine Beschleunigung mehr, so bewegt sich die Platte wieder zurück in die Mitte (C1=C2). Dies bedeutet nicht zwangsweise, dass sich der Sensor nicht mehr bewegt! Aus dieser Kapazitätsänderung und der Federstärke lässt sich dann die Beschleunigung ermitteln.
(Quelle: 3)

5. Die Programmierung

Um die Werte auslesen zu können entwickelte ich dieses Programm.
Auffällig dürften zwei Stellen sein:

```
Tempi = 256 * Xh
Tempi = Tempi + Xl
```

Diese sind wie folgt zu erklären:
Dem Manual des Beschleunigungssensors kann man entnehmen, dass `Xh` ein HIGH Byte und `Xl` ein LOW Byte ist. Dies bedeutet, es werden alle Bits, welche nach dem 8 Bit folgen in das HIGH anstatt in das LOW Byte geschrieben, da der Speicherplatz eines Bytes nicht für den Wert ausreicht. Um mit dem Wert rechnen zu können, muss aus den beiden Werten wieder einer gemacht werden. Dies macht man, indem man das HIGH Byte mit 256 (=2^8) multipliziert und dann zum LOW Byte addiert.
Die zweite Stelle ist:

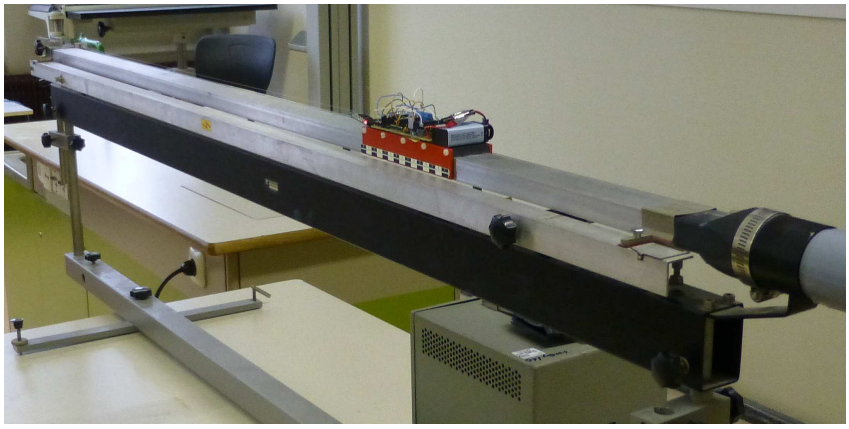
```
Tempsi = Tempi * 0.00059875
```

Dieses erklärt sich wie folgt:
Laut dem Manual entspricht der Wert 16384 (=2^14) standartgemäß 1g (16384 LSB/g (LSB = Least Significant Bit)). Folglich muss man um die Beschleunigung pro Bit zu ermitteln 9,81 durch 16384 dividieren. Das Ergebnis ist dann gleich 0.00059875. Dementsprechend muss der ermittelte Wert mit 0.00059875 multipliziert werden.
(Quelle: 2)

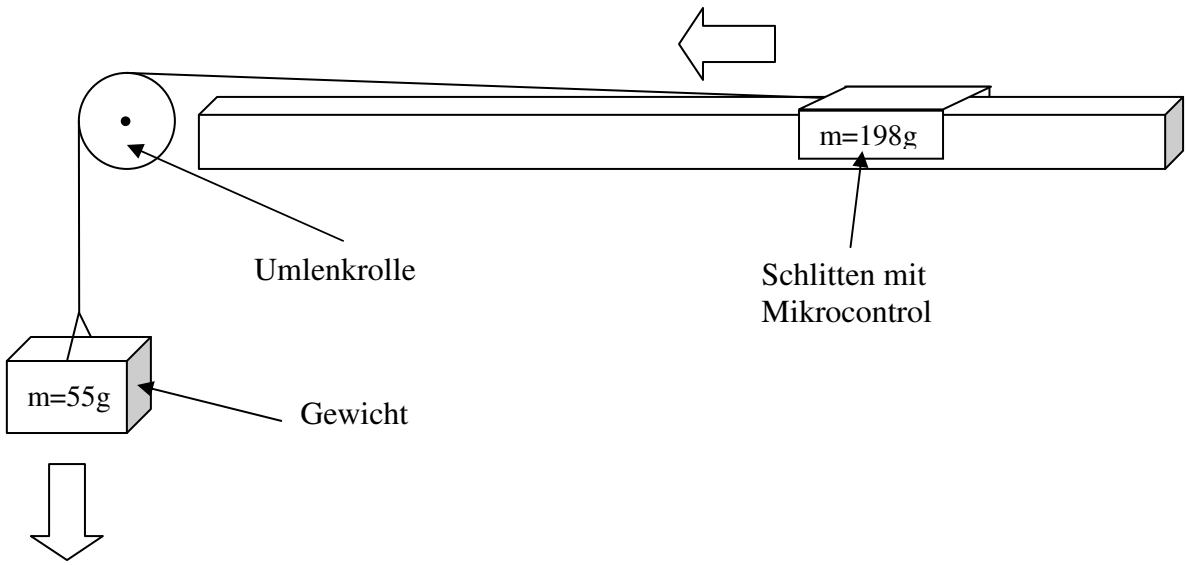
6. Messergebnisse im Stillstand

Nachdem ich das Programm auf den Attiny übertragen hatte, wurden auch schon die ersten Ergebnisse gesendet. Die Werte waren aber nicht gleich 0, wie ich es bei einem ruhenden Körper erwartet hatte. So nahm ich zunächst an, dass es sich um einen Fehler in meinem Programm oder um eine Fehlfunktion im Beschleunigungssensor handelt. Jedoch zeigte sich schnell, dass ich einen Teil der Erdbeschleunigung, wegen leichter Schieflage des Sensors, gemessen hatte. Hier ist aber schon das erste Problem zuerkennen. Denn es stellte sich heraus, dass es sehr schwer ist den Sensor exakt waagerecht auszurichten. Die Messwerte zeigten außerdem eine starke Schwankung von ca. ±0.03m/s^2.

7. Messergebnisse auf der Luftkissenbahn



Als nächstes versuchte ich die Werte mit einer bekannten Beschleunigung zu testen. Dazu verwendete ich eine Luftkissenbahn. Den Mikrocontroller befestigte ich auf einen Schlitten (mit Platine 198g), welcher durch ein Seil und eine Umlenkrolle mit einem Gewicht (55g) verbunden war. Durch dieses Gewicht konnte der Mikrocontroller beschleunigt werden.



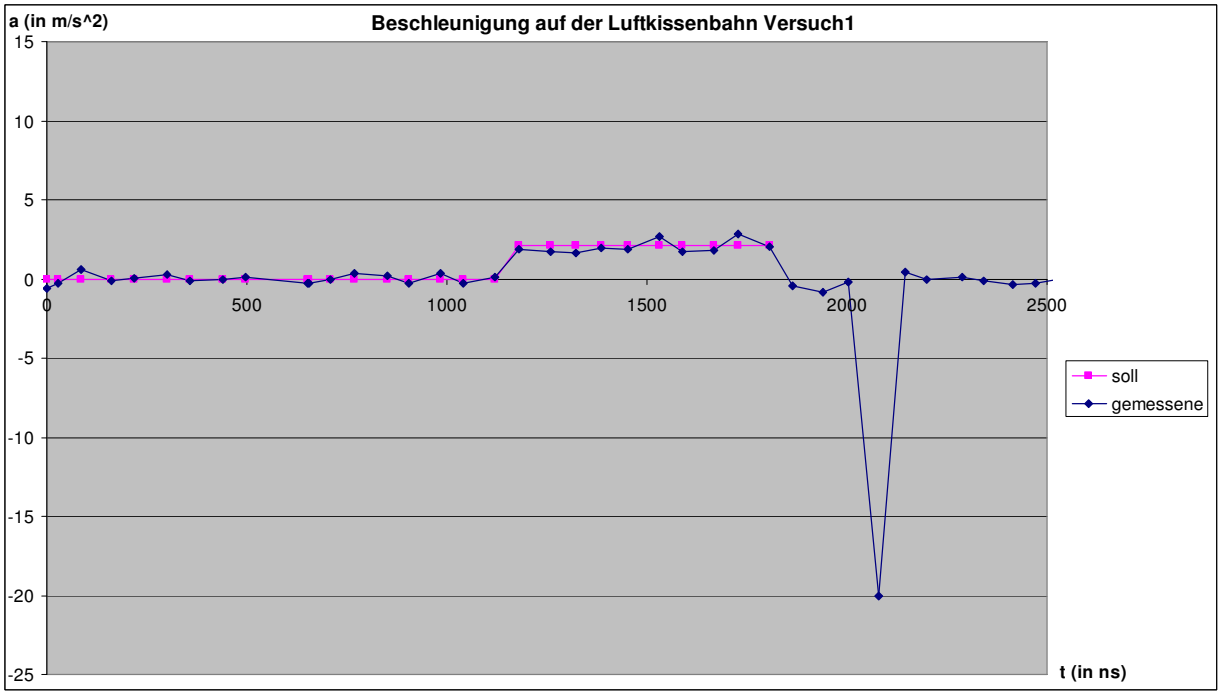
7.1 Auswertung

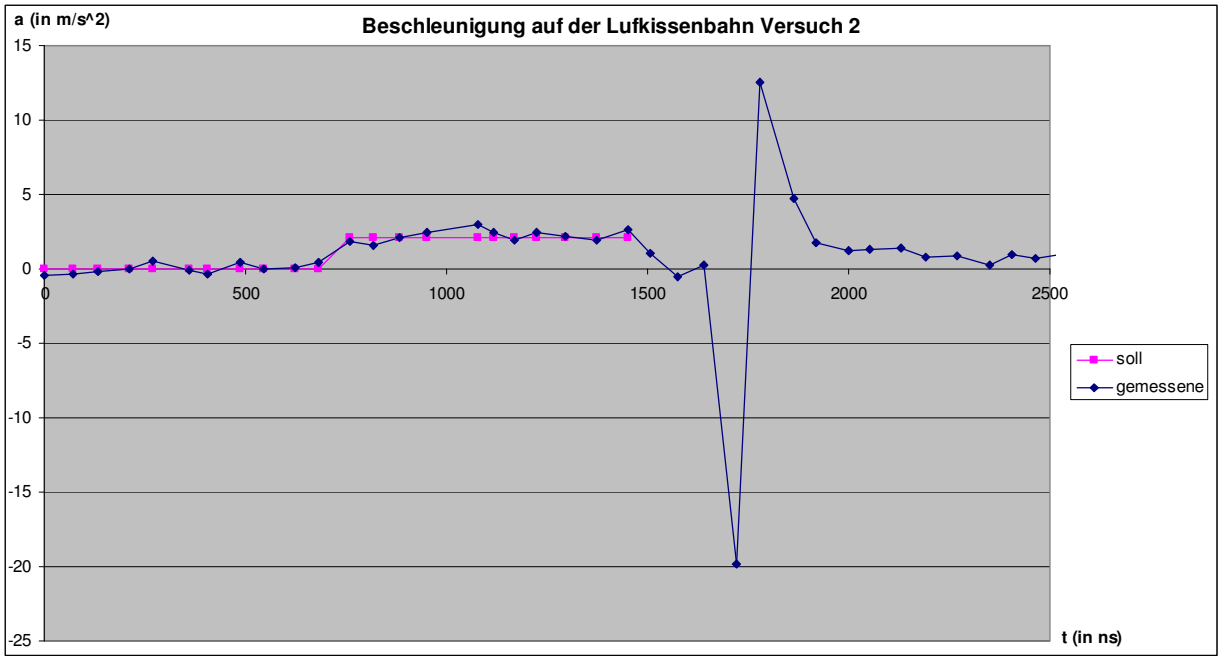
Diesen Versuch wiederholte ich zweimal. Aus den Messreihen erstellte ich mit Hilfe von Excel jeweils ein Diagramm. Die Beschleunigung, welche durch leichte Schieflage des Sensors entstand, habe ich dabei bereits rausgerechnet. Die erste Versuchsdurchführung ergab eine Beschleunigung von durchschnittlich 2,03 m/s^2 in der Beschleunigungsphase, im zweiten Versuch wurde der Schlitten durchschnittlich mit 2.14m/s^2 beschleunigt. Diese beide Werte haben mit -0,1 m/s^2 bzw. 0,11 m/s^2 nur eine "geringe" Abweichungen vom Sollwert 2,13 m/s^2, welcher sich wie folgt berechnet:

Der Index "g" entspricht dem Gewicht, "S" dem Schlitten inklusive Mikrocontroller

$$F = m * a$$
$$F_s = F_g$$
$$(m_S + m_g) * a_S = m_g * a_g$$
$$a_s = \frac{m_g * a_g}{(m_s + m_g)}$$
$$a_s = \frac{0{,}055kg * 9{,}81\frac{m}{s^2}}{0{,}253kg} = 2{,}13\frac{m}{s^2}$$

Diese Abweichungen sind jedoch auch deutlich größer, wie die Schwankungen des Sensors im Stand (±0,3 m/s^2). Die erhöhte Abweichung kann durch Elektrosmog, Reibung, leicht schiefe oder defekte Luftkissenbahn, schief angebrachten Sensor (dann wird nämlich immer nur ein Vektor der Geschwindigkeit erfasst) und vieles mehr verursacht werden.
Der Große negative Ausschlag, welcher bei beiden Versuchen nach der Beschleunigungsphase folgt, lässt sich durch das Anstoßen des Schlittens am Ende der Luftkissenschwebebahn erklären. Denn dabei wird dieser stark abgebremst. Die danach folgende Beschleunigung im zweiten Versuch entsteht durch das Abbremsen des Rückschlages des Mikrocontrollers.





7.2 Fehlversuche

Bei vorherigen Durchführungen dieses Versuches sind jedoch einige ungünstige Bedingungen gewählt worden, welche ich beheben musste. So stand zum Beispiel die starke Luftpumpe, welche für den Betrieb der Luftkissenbahn nötigt ist, auf dem gleichen Tisch, wie die Luftkissenbahn selbst und erzeugte so nicht nur Elektrosmog, sondern brachte auch den Tisch zum Vibrieren. Dies verfälscht die Messergebnisse. Das Problem mit der Vibration ließ sich jedoch leicht lösen, indem ich die Pumpe auf einem anderen Tisch stellte. Die Platine vor dem Elektrosmog abzuschirmen, welcher den Sensor beeinflusst, wäre jedoch kontraproduktiv, da dies auch die Bluetooth-Verbindung beeinflussen würde. Des Weiteren verwendete ich ursprünglich nur ein Gewicht von 5 Gramm. Die daraus resultierende Beschleunigung von nur 0,19 m/s^2 ist deutlich geringer, als die oben beschriebene Abweichung von ±0,3 m/s^2. Folglich war das Arbeiten mit diesem 5 Gramm Gewicht nicht sinnvoll.

8. Nur eine Richtung

Grundsätzlich kann der Beschleunigungssensor auch die Beschleunigung in die anderen beiden Richtungen messen. Nachdem ich jedoch dafür mein Programm erweitert hatte, musste ich feststellen, dass der Speicherplatz des Attinys nicht ausreicht. Meine Versuche, die viel Speicherplatz benötigende Singlevariable durch eine Andere zu ersetzen oder das Programm über SPI zu übertragen, um den Speicherplatzes des Uploaders zu sparen, erwiesen sich alle als erfolglos. So blieb mir nichts anderes übrig, als mich mit einer der drei Richtungen zufrieden zu geben.

9.Quellen:

1. BASCOM I2C

http://staff.ltam.lu/feljc/electronics/bascom/BASCOM_I2C.pdf

2. MPU-6000 Register Map and Description

http://invensense.com/mems/gyro/documents/RM-MPU-6000A.pdf

3. MEMS - Micro-Electro-Mechanical Systems

http://www.elektronik-kompendium.de/sites/bau/1503041.htm

4. I2C Grundlagen Attiny

http://www.g-heinrichs.de/attiny/I2C_Grundlagen_Attiny.pdf

5. I2C Bus Hintergrundwissen – ComputerClub2 WIKI

http://www.cc-zwei.de/wiki/index.php?title=I2C_Bus_Hintergrundwissen