htl bildung mit zukunft	HÖHERE TECHNISCHE BUNDES-LEHR- UND VERSUCHSANSTALT SALZBURG
	Abteilung: Elektronik
	Ausbildungsschwerpunkt: Technische Informatik

DIPLOMARBEIT

Flight-Info-System

Ausgeführt im Schuljahr 2012/2013

von:

Johannes Schwöller 5AHELI

Sebastian Modl 5AHELI

Betreuer:

Prof. Dipl.-Ing. Ing. Karl Heinz Steiner

Salzburg, am 07.05.2013

Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Diplomarbeit mit Ausnahme der ausdrücklich als zitiert oder verwendet gekennzeichneten Teile selbständig erarbeitet und verfasst habe.

Datum/Ort: 07.05.2013 Salzburg

Unterschrift(en):

Johannes Schwöller

Sebastian Modl

DIPLOMARBEIT

Zusammenfassung

Namen der Verfasser/innen	Johannes Schwöller Sebastian Modl
Jahrgang Schuljahr	5AHELI
Thema der Diplomarbeit	Entwicklung einer Messeinheit zur Flugdatenerfassung von Ultraleichtflugzeugen
Kooperationspartner	ScaleWings Modelltechnik GmbH Ing. Johann Schwöller
Aufgabenstellung	Eine Messeinheit zeichnet diverse Flugdaten (True Airspeed, Groundspeed, g-Belastung, GPS-Koordinaten und die Flughöhe) auf, zeigt diese auf einem Display an und speichert sie zusätzlich auf einer SD-Karte ab. Bei Stromausfall liefert die Messeinheit noch notwendige Flugdaten.
Realisierung	Ein Mikrocontroller steuert die Sensoren an und nimmt deren Daten auf. Weiters werden das Display und die SD-Karte vom Mikrocontroller angesteuert. Programmiert wird dieser mit der Sprache C. Zur Überlastungsanzeige dienen vier Duo-LEDs (Rot-Grün). Die Computersoftware wird mit der Programmiersprache Java realisiert.
Ergebnisse	Mit Hilfe der aufgezeichneten Daten können Piloten ausfindig gemacht werden, die das Flugzeug überlastet haben. Bei Ausfall der Boardinstrumente liefert das Modul weiterhin notwendige Daten und kann daher als Backup-System dienen. Nach einer Testphase wird die Messeinheit durch die Firma ScaleWings vermarktet.
Typische Grafik, Foto etc. (mit Erläuterung)	FLINOS Flight-Info-System
Teilnahme an Wettbewerben, Auszeichnungen	Jugend-Innovativ
Möglichkeiten der Einsichtnahme in die Arbeit	Schulbibliothek HTBLuVA-Salzburg

DIPLOMA THESIS

Summary

Author(s)	Johannes Schwöller Sebastian Modl
Form Academic year	5AHELI
Topic	Development of a device for flight data aquisition for ultralight planes
Cooperationpartners	ScaleWings Modelltechnik GmbH Ing. Johann Schwöller
Assignmentoftasks	This product will be placed in the cockpit of an ultralight plane. It measures various flight-infos (True Airspeed, Groundspeed, g-Load, GPS-coordinates and altitude). The data is displayed on a little monitor and additionally stored on an SD-Card.
Realization	A microcontroller controls the sensor, the display and the SD-Card. Two-in-one-LEDs (red-green) signalize various overloads. The microcontroller is programmed with the computer language C. The computer software is realized with the programming language Java.
Results	The purpose of this product is the recording of the flight-data. Flying clubs will be able to check the recording and detect overloads (limits). In case of an electrical power blackout the product continues displaying the necessary flight-data for a limited time.
Illustrative graph, photo (incl. explanation)	FLINOS Flight-Info-System
Participation in competitions Awards	Jugend-Innovativ
Accessibilityof diplomathesis	School library HTBLuVA-Salzburg

Vorwort

In der vorliegenden Arbeit beschäftigten wir uns mit der Entwicklung einer Messeinheit zur Flugdatenerfassung von Ultraleichtflugzeugen.

Die Idee zu unserem Projekt entstand bereits im Sommer 2012. Gemeinsam mit Herrn Ing. Johann Schwöller, der selbst Ultraleichtflugzeuge entwickelt, wurden die Grundlagen für unser Projekt geschaffen.

Aufgrund der komplizierten Geschwindigkeitsermittlung der tatsächlichen Fluggeschwindigkeit (True Airspeeds, in Relation zur umgebenden Luft) und der komplexen Bus-Struktur in unserem Projekt, haben wir folgende vertiefenden Grundlagenarbeiten für unsere Diplomarbeit gewählt:

- Messtechnische Erfassung der analogen Drucksensordaten und Entscheidung, ob diese Signale vor der Übertragung digitalisiert werden müssen
- Aufzeichnung und Bewertung der Signalverläufe am vorliegenden I²C-Bus

Wir bedanken uns bei unserer Partnerfirma ScaleWings und unserem Ansprechpartner Ing. Johann Schwöller für die tatkräftige Unterstützung und dem verantwortlichen Leiter des Printlabors der Elektronikabteilung, Herrn Pölzer für die Unterstützung bei der Erstellung der notwendigen Leiterplatten.

Inhaltsverzeichnis

1 Einleitung

Da wir uns sehr für die Luftfahrt, speziell Ultraleichtflieger, interessieren, haben wir nach einem Projekt in diesem Bereich bereits im Jahr 2012 gesucht.
Nach Absprache mit Herrn Ing. Johann Schwöller, der selbst Ultraleichtflugzeuge entwickelt, entstand die Idee zu unserem Projekt.

Grundsätzlich wurde das Modul für den Einbau in Ultraleichtflugzeugen ausgelegt. Ultraleichtflugzeuge, auch ULs genannt, sind Flugzeuge, die klein, sehr leicht und mit Motor angetrieben sind. Diese Flugzeuge sind in Europa für maximal 2 Personen zugelassen.
Der Einsatz ist jedoch auch in allen anderen Privatfliegern möglich.

Ziel war es, ein Gerät zu entwickeln, das sämtliche Flugdaten aufzeichnet, anzeigt und abspeichert. Vereine können die Daten abrufen und mögliche Überlastungen feststellen. Das Produkt kann auch im Hobbybereich eingesetzt werden. Kommt es zu einem Stromausfall, so liefert die Messeinheit über eine bestimmte Zeit noch notwendige Flugdaten.

Zu Beginn erkundigten wir uns ausführlich bei unserem Ansprechpartner Ing. Johann Schwöller. Er erklärte uns das Problem in der privaten Fliegerei, sowie diverse Lösungsansätze, um den Flugverkehr sicherer zu machen. Bei Fragen konnten wir uns jederzeit an unseren Projektbetreuer Herrn Professor Steiner wenden.

2 Aufgabenstellung

Bisher gab es in Privatflugzeugen keine Überlastungsanzeige. Vereine, Vercharterer und die Piloten selbst konnten nicht nachvollziehen, ob, wann und wo ein Limit überschritten wurde und wie stark die Überlastung letztendlich war. Die zurzeit erhältlichen Lösungen am Weltmarkt sind leider für Normalverbraucher unerschwinglich.
Um die Sicherheit für Piloten und Fluggäste zu erhöhen, haben wir uns daher dazu entschlossen, solch ein Modul selbst zu entwickeln. Es soll Messwerte im Cockpit anzeigen, Überlastungen signalisieren und die Daten abspeichern, um eine ausführliche Analyse am Computer möglich zu machen.

Personen die ihre Flugzeuge vermieten (verchartern) erhalten somit sämtliche Daten eines Fluges. Mit Hilfe der aufgezeichneten Daten können nun Piloten ausfindig gemacht werden, die das Flugzeug überlastet haben. Beim Auftreten von Überlastungen kann es durchaus sein, dass das Flugzeug beschädigt wird. Der/die nächste Pilot/in könnte aufgrund des vorigen Benutzers somit ein nicht funktionstüchtiges Flugzeug in Betrieb nehmen, und womöglich kann dies zu einem Absturz führen. So bietet dieses Produkt auch Sicherheit für die Piloten und Fluggäste.

Das Produkt wird im Cockpit des Ultraleichtflugzeuges angebracht. Es handelt sich dabei um eine Messeinheit, die diverse Flugdaten (True Airspeed, Groundspeed, g-Belastung, GPS-Koordinaten und Flughöhe) auf einem Display anzeigt und zusätzlich auf einer internen und externen SD-Karte abspeichert.
Der Sinn und Zweck dieses Produktes ist die Anzeige und Aufzeichnung der Flugdaten. Vereine und Vercharterer können die Daten abrufen und mögliche Überlastungen feststellen. Das Produkt kann auch im Hobbybereich eingesetzt werden. Kommt es zu einem Stromausfall, so liefert die Messeinheit über eine bestimmte Zeit noch notwendige Flugdaten.

Zu Beginn wurde eine Prototypenplatine angefertigt. Die TAS-Messung wurde auf einem Steckbrett aufgebaut und mit der Platine verbunden. Herzstück dieses Produktes ist ein Mikrocontroller. Dieser nimmt Daten von den Sensoren auf, zeigt diese am Display an und speichert sie intern ab. Mittels einer SD-Karte kann man die Daten auslesen und extern auswerten. Am Computer bietet eine Software eine genaue Analyse der Flugdaten.
Realisiert wird die Mikrocontroller-Software in der Programmiersprache C, die Auswertung am Computer erfolgt mit Java.

2.1 Projektorganisation

Unser Projektteam besteht aus zwei Personen, Johannes Schwöller und Sebastian Modl, wobei Johannes die Rolle des Projektleiters übernimmt. Entscheidungen werden immer gemeinsam getroffen.
Aufgrund des eingespielten Teams aus den Projekten der vorigen Jahre, haben wir uns wieder für die selbe Projektorganisation entschieden.

2.1.1 Projektmitglieder

Name: Johannes Schwöller
Adresse: Sonnenweg 5, 5204 Straßwalchen
E-Mail: j.schwoeller@gmx.at

Abb. 1: Johannes Schwöller

Aufgabenbereiche:

- Hardware Entwicklung
  - Auswahl der Bauteile
  - Platinenentwicklung
  - Prototypenaufbau
  - Schaltung für die Ladeelektronik des Akkus
- Software Entwicklung
  - Ansteuerung des Displays
  - Ansteuerung und Auswertung des Beschleunigungssensors
- Sonstiges
  - Gestaltung der Flyer und Plakate

Name: Sebastian Modl
Adresse: Mondseerstraße 47, 5204 Straßwalchen
E-Mail:

Abb. 2: Sebastian Modl

Aufgabenbereiche:

- Hardware Entwicklung
  - Auswahl der Bauteile
  - Prototypenaufbau
- Software Entwicklung
  - Ansteuerung und Auswertung der Drucksensoren
  - Ansteuerung des GPS-Moduls
  - Ansteuerung der SD-Karte
- Sonstiges
  - Dokumentationsorganisation

2.1.2 Projektbetreuer

Name: Prof. Dipl.-Ing. Ing. Karl Heinz Steiner
E-Mail: KarlHeinz.Steiner@htl-salzburg.ac.at

Die Wahl des Projektbetreuers fiel uns nicht schwer, wir entschieden uns für unseren Klassenvorstand Karl Heinz Steiner. Herr Professor Steiner hat uns seit dem 2. Jahrgang an als Klassenvorstand begleitet. Unterrichtet wurde unsere Klasse in den technischen Fächern IKT und EDT von ihm. Er war es, der uns das Programmieren von Mikrocontrollern schon im 3. Jahrgang beibrachte. Hätte er nicht unsere Begeisterung für die Softwareentwicklung im Bereich der Hardware geweckt, wäre unser Projekt nie zustande gekommen.
Im Zuge der Diplomarbeit hat Herr Professor Steiner sich immer bemüht, uns zu motivieren. Besonders bei den Präsentationen stand er immer mit wichtigen Tipps zur Seite und bemühte sich stets darum, dass die Zuschauer unsere Präsentation auch in 20 Jahren nicht vergessen werden.

Wir bedanken uns sehr herzlich für die tatkräftige Unterstützung!

Abb. 3: Prof. Dipl.-Ing. Ing. Karl Heinz Steiner

2.1.3 Projektpartner[1]

Die Firma ScaleWings hat ihren Sitz in Steindorf bei Straßwalchen und entwickelt seit zehn Jahren originalgetreue Großflugmodelle. Die Modelle werden in Glas/Kohlefaser-Sandwich-Konstruktion produziert. Der absolut maßstabgetreue und sehr detailierte Aufbau sind das Einzigartige dieser Großflugmodelle.

Seit Anfang 2011 befasst sich ScaleWings mit dem Nachbau einer Jägermaschine aus dem 2. Weltkrieg der Amerikaner (Mustang P51) als Ultraleichtflieger. Dabei wird jedes kleinste Detail, jede Niete, jeder Deckel usw. berücksichtigt. Die Konstruktion erfolgt mittels 3D-CAD-Systemen. Die Urmodelle werden auf großen CNC-Fräsanlagen hergestellt und bearbeitet. Die präzise Formenentwicklung erfolgt in aufwändigen Handlaminierverfahren.

Weiters bietet ScaleWings seit kurzem Rundflüge mit Ing. Johann Schwöller und seinem Sportdoppeldecker an.

Unser Ansprechpartner ist Ing. Johann Schwöller, der die Firma ScaleWings leitet.
Wir möchten uns sehr für die hervorragende Zusammenarbeit bedanken.

Name: Ing. Johann Schwöller
Adresse: Sonnenweg 5, 5204 Straßwalchen
E-Mail: j.schwoeller@scalewings.com

Abb. 5: Ing. Johann Schwöller[3]

ScaleWings®

Abb. 4: Logo ScaleWings Modelltechnik GmbH[2]

[1] vgl. http://www.scalewings.com/ (31.03.2013)
[2] http://www.scalewings.com/index_htm_files/10.jpg (31.03.2013)
[3] http://www.scalewings.com/chartering_htm_files/48.jpg (31.03.2013)

Abb. 6: Entwicklung des Rumpfes der ScaleWings Mustang FK-51[4]

Abb. 7: ScaleWings Mustang FK-51 auf der Aero 2013

[4] http://www.scalewings.com/mustang_htm_files/145.jpg (31.03.2013)

2.2 IST-Analyse

Bisher gibt es in Privatflugzeugen keine Anzeige in einem Ultraleichtflugzeug, die Überlastungen signalisiert. Die G-Belastung und der Groundspeed (GS) sind dem Piloten während dem Flug nicht bekannt.
Piloten müssen sich daher blind auf das Flugzeug und auf vorige Benutzer verlassen. Das Risiko, ein nicht funktionstüchtiges Flugzeug in Betrieb zu nehmen, ist dadurch nicht ausgeschlossen.
Eine Beschädigung am Flugzeug kann durchaus zu einer gefährlichen Situation im Flugzeug oder zu einem Absturz führen.
Daher haben wir uns überlegt, welche Maßnahmen und Zielsetzungen man treffen könnte. Unser Ziel war es, ein Gerät zu entwickeln, das im Cockpit eines Ultraleichtflugzeuges angebracht wird, um die Sicherheit der Piloten und Fluggäste zu erhöhen.
Gemeinsam mit unserer Partnerfirma ScaleWings haben wir bereits im Sommer 2012 die Grundlagen für unser Projekt geschaffen.

2.3 Zielsetzung

Ziel ist es, ein Gerät zu entwickeln, das sämtliche Flugdaten aufzeichnet, anzeigt und abspeichert. Um sicherzustellen, dass die Messeinheit ordnungsgemäß funktioniert, wird sie von der Firma ScaleWings gründlich getestet.
Die Messeinheit soll True Airspeed (TAS), Groundspeed (GS), g-Belastung, GPS-Koordinaten und Flughöhe von Ultraleichtflugzeugen aufzeichnen.
Die gemessenen Daten werden auf einen permanent vorhandenen Speicher abgelegt und auf Anforderung auf einen Wechseldatenträger übertragen.
Die Momentanwerte sind im Cockpit anzuzeigen und mögliche Überlastungen sind zu signalisieren.
Eine Backupversorgung soll bei Stromausfall vorrübergehend die ordnungsgemäße Funktion ermöglichen.

Wenn möglich kann die geflogene Flugroute mittels Google-Earth angezeigt werden. Weiters kann eine Überspannungsschutzschaltung dimensioniert und aufgebaut werden. Optional kann eine Computersoftware zur Datenauswertung programmiert werden.

2.4 Systemarchitektur

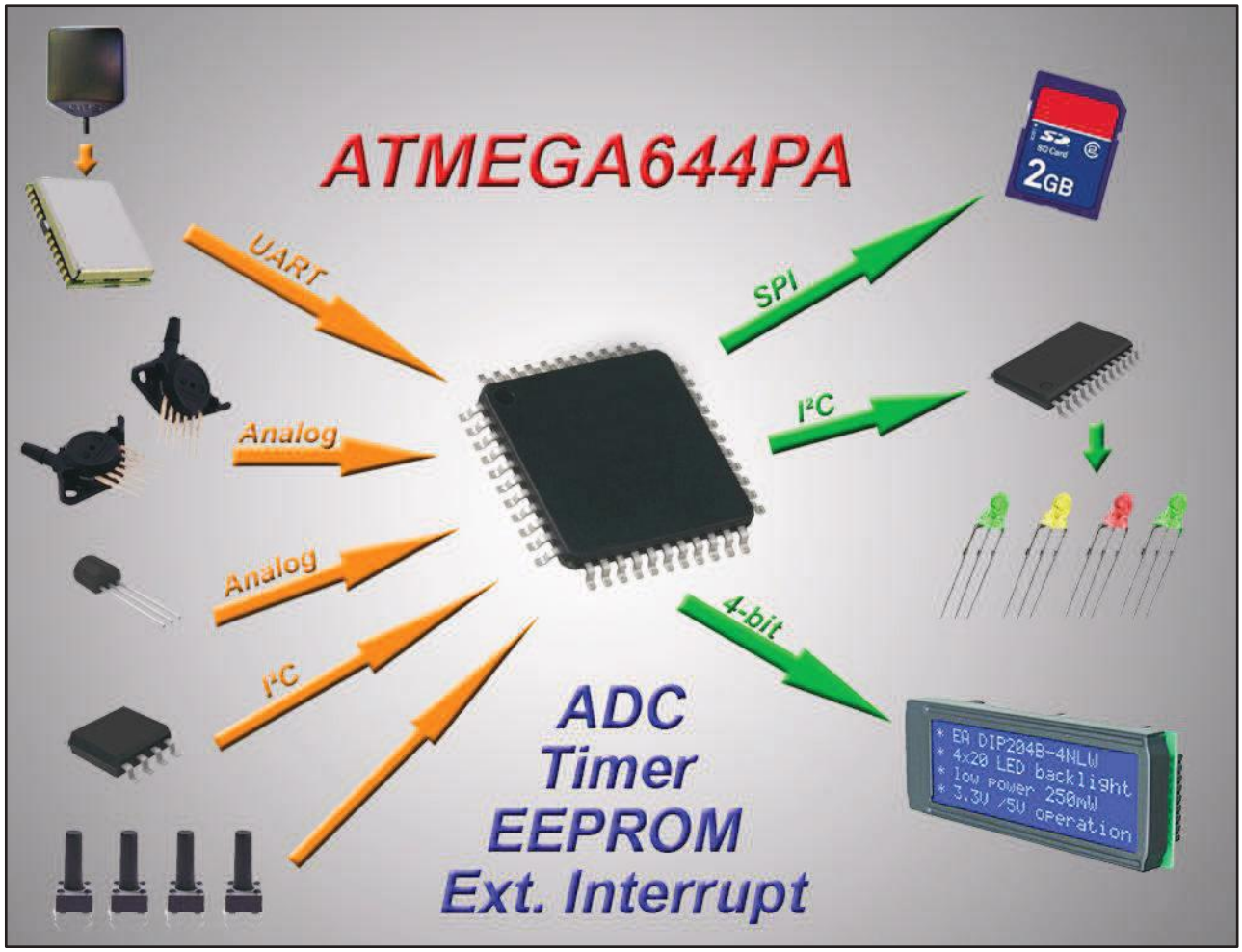


Abb. 8: Systemarchitektur des Projektes

Die Zentrale der Messeinheit ist der Mikrocontroller ATmega644PA. An diesem werden der Digital-Analog-Wandler, EEPROM, Timer und Externe Interrupts programmiert und angesteuert. Weiters dienen zur Übertragung der Signale die Bus-Systeme I²C, SPI und USART. Das Display wird im 4-Bit-Modus angesteuert.

Der Mikrocontroller hat 44 Pins, dennoch benötigen wir mehr I/O-Ports. Um dies zu lösen, wurde ein I/O-Expander verwendet. Dieser wird mit I²C angesteuert und dient zur LED-Ansteuerung.

3 Überlegungen und Lösungsvarianten

3.1 Allgemeine Schaltungsentwicklung

Zu Beginn wurde überlegt, welche Lösungsvarianten für die Schaltungsentwicklung in Frage kämen. Berücksichtigt wurden unter anderem die Kosten, der Zeitaufwand und das Risiko der einzelnen Varianten.

3.1.1 Aufbau am Steckbrett

Auf dem ersten Blick erscheint der Aufbau am Steckbrett als die schnellste und vor allem sicherste Variante. Vorteil dieser Lösungsvariante ist es, dass der Aufbau Schritt für Schritt erfolgen kann. Die Programmierung und der Schaltungsaufbau erfolgen somit beinahe zeitgleich. Systematische Probleme und die daraus entstehenden Kosten können zum frühestmöglichen Zeitpunkt erkannt werden.

Bei näherer Betrachtung ergeben sich aber viele Problemstellen. Für die Prototypenplatine ändern sich die Kosten nicht. Rechnet man aber die Entwicklung des Serienproduktes mit ein, so benötigt man die doppelte Anzahl an Bauteilen. Das bedeutet auch doppelte Kosten. Jeder einzelne Sensor in SMD-Bauform braucht eine Adapterplatine, die die Verbindung zum Steckbrett ermöglicht. Da bei diesem Projekt sehr viele Adapterplatinen nötig wären, würde der Aufbau unübersichtlich werden. Außerdem kann es bei Steckbrettern leicht passieren, dass Kabelverbindungen unabsichtlich getrennt werden. Derartige Fehler können schwer erkannt werden.

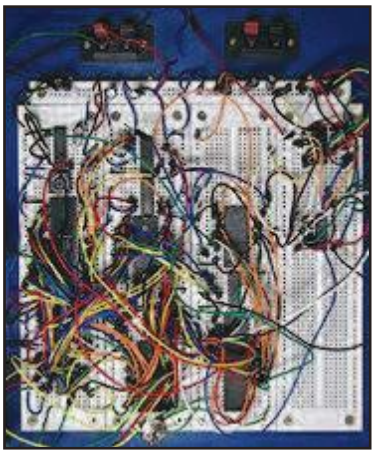


Abb. 9: Unübersichtlicher Aufbau am Steckbrett[5]

Kostenschätzung:

Kostenstelle	*Preis*
Display	*€ 60*
GPS-Modul	*€ 100*
Drucksensoren	*€ 50*
Programmiergerät	*€ 80*
Weitere Bauteile	*€ 100*
Summe	*€ 390*

[5] http://upload.wikimedia.org/wikipedia/commons/thumb/4/47/Breadboard_complex.jpg/220px-Breadboard_complex.jpg (31.03.2013)

3.1.2 Sofortiger Platinenentwurf

Die Idee, sofort mit der Platinenentwicklung zu starten, wäre grundsätzlich gut. Die Kosten reduzieren sich im Vergleich zur ersten Lösungsvariante um die Hälfte, da die Bauteile nur einmal benötigt werden.
Überlegungen für das fertige Serienprodukt sind bereits früh vorhanden. Der Projektfortschritt erfolgt aber zu Beginn sehr langsam. Außerdem kann man erst zum Schluss mit der Programmierung beginnen. Fehler werden erst sehr spät erkannt und können nur schwer korrigiert werden.

Kostenschätzung:

Kostenstelle	*Preis*
Display	*€ 30*
GPS-Modul	*€ 50*
Drucksensoren	*€ 25*
Programmiergerät	*€ 40*
Weitere Bauteile	*€ 500*
Summe	*€ 195*

3.1.3 Kombination aus Steckbrett und Platine

Nach langen Überlegungen haben wir uns für diese Variante entschieden. Es ist die Kombination aus der Lösungsvariante eins und zwei. Der wesentliche Vorteil ist die frühzeitige Erkennung von Problemen oder Gefahren. Auch der Schaltungsaufbau bleibt übersichtlich, da sich nur mehr ein kleiner Teil der Schaltung am Steckbrett befindet. Alle SMD-Bauteile befinden sich auf der Platine. Diese wird mit Kabeln mit dem Steckbrett verbunden. Nach Sicherstellung der einzelnen Funktion, kann man mit der Entwicklung des Serienproduktes beginnen.

Kostenschätzung:

Kostenstelle	*Preis*
Display	*€ 60*
GPS-Modul	*€ 100*
Drucksensoren	*€ 50*
Programmiergerät	*€ 80*
Weitere Bauteile	*€ 100*
Summe	*€ 390*

3.1.4 Risikoanalyse

Nummer	Lösungsvariante	Vorteile	Nachteile	Risiko
1	Aufbau am Steckbrett	Schritt für Schritt, Frühe Programmierung, Schnelle Problemerkennung	Hohe Kosten (doppelt), Doppelte Anzahl von Bauteilen, Unübersichtlicher Aufbau	mittel
2	Sofortiger Platinenentwurf	Frühzeitiger Serienentwurf, Geringe Kosten	Langsamer Projektfortschritt zu Beginn, Programmierung erst am Schluss, Fehler werden erst spät erkannt	hoch
3	Kombination aus Steckbrett und Platine	Frühe Programmierung, Schnelle Problemerkennung, übersichtlich	Hohe Kosten (doppelt), Doppelte Anzahl von Bauteilen	niedrig

Aufgrund des geringen Risikos und der schnellen Entwicklung bzw. Problemerkennung, haben wir uns für die dritte Variante, Kombination aus Steckbrett und Platine, entschieden.

3.2 Speichermedium

3.2.1 USB-Stick

Der große Nachteil dieses Speichermediums ist die Größe. Das Einstecken in einen USB-Slot ist unproblematisch, jedoch steht ein Großteil des Gehäuses beim Produkt heraus. Ein Abbruch des Sticks ist somit nicht ausgeschlossen.
Die direkte Ansteuerung mit dem Mikrocontroller ist sehr schwierig. Um das Problem der Ansteuerung zu umgehen, wird ein USB-Controller, wenn nicht sogar ein Adapterkit, benötigt. Die Kosten und der Aufwand steigen somit enorm.
Andererseits ist von Vorteil, dass ein USB-Stick weitverbreitet und bereits in vielen Anwendungen im Einsatz ist.

3.2.2 SD-Karte

In unserem Projekt wird als Speichermedium eine SD-Karte verwendet. Anders wie beim USB-Stick, verschwindet nahezu die gesamte Karte beim Einstecken in den SD-Slot. Durch Einrasten in den Slot ist vergewissert, dass die Karte nicht versehentlich verloren geht.
Die Ansteuerung erfolgt via SPI. Es ist auch möglich mehrere SD-Karten parallel an diesen Bus anzuschließen.
Die Kosten sind sehr gering und der Arbeitsaufwand hält sich in Grenzen. Genauso wie USB-Sticks, sind SD-Karten für jeden leicht und günstig zu erhalten.

4 Lösung

4.1 Hardware

4.1.1 Mikrocontroller ATmega644PA[6]

Die Anforderungen an unseren Mikrocontroller waren sehr hoch. Wir verwendeten den ATmega644PA, welchen wir im Vorjahr (4. Jahrgang) bereits im Projektunterricht verwendet haben.

Die MCU verfügt über I²C, SPI und USART. Weiters ist die hohe Pinanzahl (32 IO-Pins) ein großer Vorteil. Trotz allem benötigten wir für das komplexe Projekt einen I/O-Expander, der zusätzliche Ein- und Ausgänge zur Verfügung stellt. Durch den eingebauten EEPROM von zwei Kilobyte, ist die Speicherung der wichtigen Daten ohne zusätzliche Hardware möglich.

Der Mikrocontroller muss mit mindestens 1.8V und höchstens 5.5V versorgt werden und zieht im Power-down-Mode einen sehr kleinen Strom von lediglich 600nA.

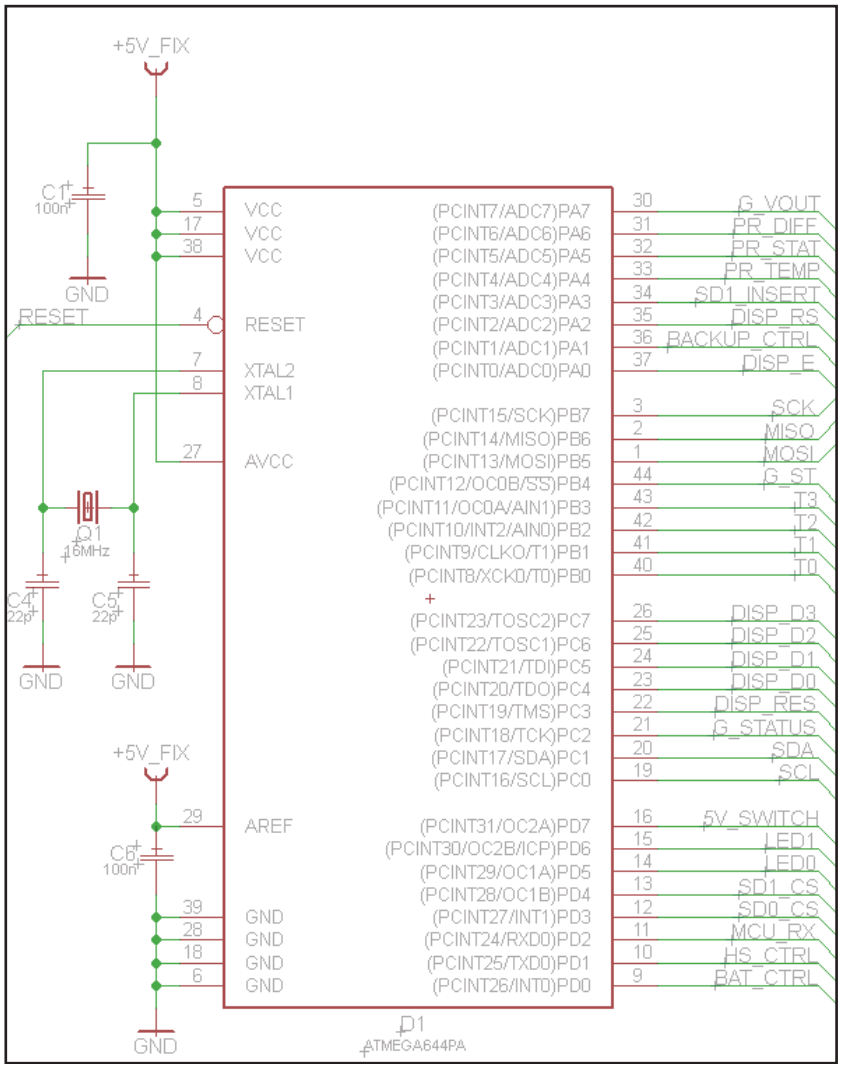


Abb. 10: Mikrocontroller Anschlüsse

[6] vgl. http://docs-europe.electrocomponents.com/webdocs/0dc5/0900766b80dc5089.pdf (10.04.2013)

G_VOUT	…	G-Sensor Ausgangsspannung
PR_DIFF	…	Differenzdruck OPV Ausgangsspannung
PR_TEMP	…	Temperatursensor Ausgangsspannung
SD1_INSERT	…	SD-Karte 1 eingesteckt Ja/Nein
DISP_RS	…	Display Befehl/Daten
BACKUP_CTRL	…	Backupbatterie Spannungsüberwachung
DISP_E	…	Display Enable
SCK	…	SPI Clock
MISO	…	SPI Master in Slave out
MOSI	…	SPI Master out Slave in
G_ST	…	G-Sensor Selftest
T3-T0	…	4 Taster
DISP_D3-D0	…	Display 4 Datenbits
DISP_RES	…	Display Reset
G_STATUS	…	G-Sensor Status (Fehlermeldung)
SDA	…	I²C Daten
SCL	…	I²C Clock
5V_SWITCH	…	Mosfet-Schalter zur Versorgungsabschaltung
LED1-0	…	Warn-LEDs
SD1_CS, SD0_CS	…	Chipselect der internen bzw. externen SD-Karte
MCU_RX	…	GPS-Empfänger USART Receive
HS_CTRL	…	Hauptschalter überprüfen High/Low
BAT_CTRL	…	Flugzeugbatterie überprüfen High/Low

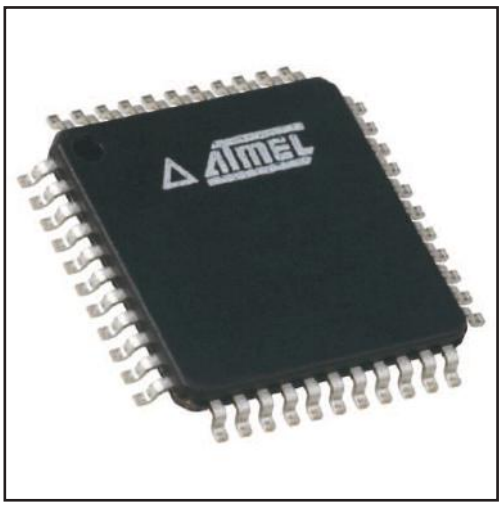


Abb. 11: Mikrocontroller ATmega644PA im TQFP-Gehäuse[7]

[7] http://abusemark.com/store/images/313-44-TQFP.jpg (10.04.2013)

4.1.2 GPS-Empfänger GPS-1513[8]

Wir suchten nach einem GPS-Empfänger, welcher die GPS-Koordinaten (Latitude, Longitude) und die Flughöhe (Altitude) liefert. Wichtig war auch eine entsprechend kleine Bauform. Angesteuert wird das Modul mittels USART, wobei es jede Sekunde über eine Leitung alle Daten sendet. Es können auch Einstellungen an den Empfänger übermittelt werden, aber in unserem Fall wird dies nicht benötigt.
Da das Modul die Geschwindigkeit (Groundspeed) bereitstellt, muss keine zusätzliche Berechnung mit Hilfe der GPS-Koordinaten durchgeführt werden.
Mit 13x15.8mm Größe gehört der Receiver zu den kleinsten Produkten die es auf dem Markt gibt. Weiters verfügt das Modul über einen externen Antennenanschluss. Dieser ist unbedingt notwendig, falls die Flugzeugkarosserie aus abschirmendem Material (z.B. Metall) besteht. In diesem Fall kann eine externe GPS-Antenne außen montiert werden.
Der GPS-Empfänger muss mit einer Spannung von 3.3V versorgt werden.

Beim Layout des GPS-Moduls ist besonderes Augenmerk auf die Verbindung zum Antennenanschluss zu legen. Der Wellenwiderstand dieser möglichst kurzen Leitung muss genau 50Ω betragen (Eingangswiderstand der Antenne). Die Leiterbahn muss 1.27mm breit sein. Weiters ist wichtig, dass auf beiden Platinenseiten, rund um den Antennenanschluss, Masseflächen vorhanden sind. Besonders wichtig ist die gegenüberliegende Massefläche, da sie einen wesentlichen Einfluss auf den Wellenwiderstand der Leiterbahn hat.

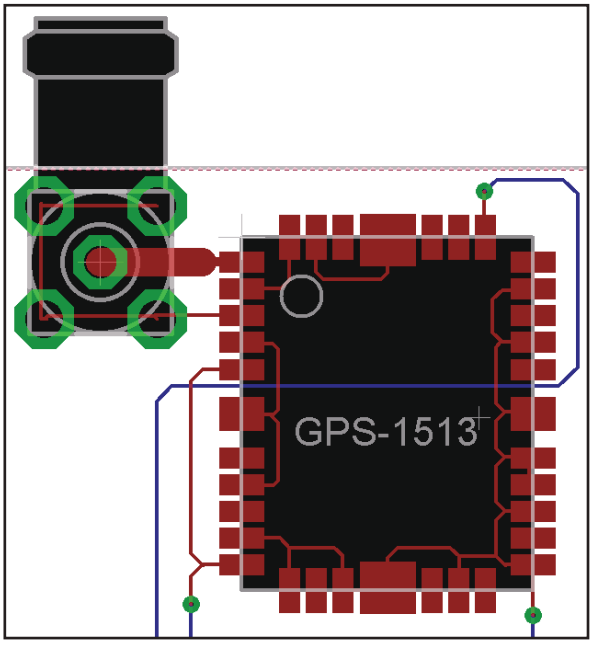


Abb. 12: Layout vom Antennenanschluss

[8] vgl. http://docs-europe.electrocomponents.com/webdocs/0df9/0900766b80df94d1.pdf (10.04.2013)

Für die Übertragung der Daten wird das NMEA (National Marine Electronics Association) Protokoll verwendet. Es werden einzelne Datenpakete übertragen, die unterschiedliche Informationen enthalten. Jede Sekunde werden alle Datenpakete nacheinander gesendet.
Um die nötigen Information zu erhalten, benötigen wir nur zwei dieser Datenpakete:

- GGA - Global Positioning System Fix Data
- RMC - Recommanded Minimum Specific GPS/Transit Data

Da die Positionen in einem Paket immer gleich bleiben, ist es nicht sehr schwierig, die Daten herauszufiltern.

Beispiele für die einzelnen Datenpakete:
*$GPGGA,060932.448,2447.0959,N,12100.5204,E,1,08,1.1,108.7,M,,,,0000*0E<CR><LF>*
*$GPRMC,092204.999,A,4250.5589,S,14718.5084,E,0.00,89.68,211200,,A*25<CR><LF>*

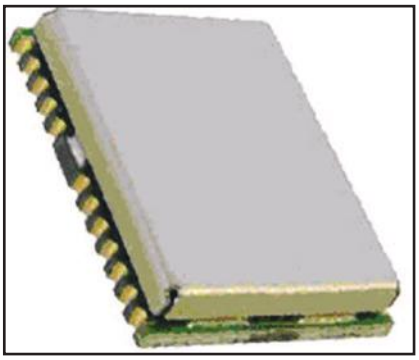


Abb. 13: GPS-Empfänger GPS-1513[9]

Abb. 14: Aktive GPS-Antenne IP67 Waterproof[10]

[9] http://www.rfsolutions.co.uk/acatalog/GPS1513-500.jpg (10.04.2013)

[10] http://p.globalsources.com/IMAGES/PDT/B1032729230/GPS-Active-Antenna.jpg (10.04.2013)

4.1.3 Drucksensor MPX4115AP[11]

Die Anforderungen an unsere Drucksensoren waren:

- Linearität
- Hohe Genauigkeit
- Feine Auflösung
- Geringe Kosten
- Schlauchanschluss

Entschieden haben wir uns für den Drucksensor MPX4115AP. Der Sensor liefert den absoluten Druck in Form einer analogen Spannung. Die Ausgangsspannung ist proportional zum Druck (linear: 46mV/1kPa).

Um die Fluggeschwindigkeit (True Airspeed) zu erhalten, wird das Prinzip der Prandtlsonde verwendet. True Airspeed ist die Geschwindigkeit in Bezug auf die umgebende Luft, sie ist bei Gegenwind also höher als der Groundspeed.
Das Messsystem ermittelt den statischen Druck und den Staudruck. Der daraus resultierende Druckunterschied wird für die Geschwindigkeitsberechnung verwendet.

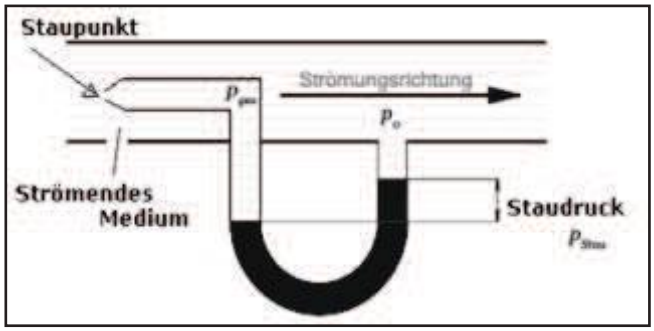


Abb. 15: Prandtlsonde[12]

Ist der Druckunterschied gering, so ist auch der Spannungsunterschied der beiden Drucksensoren sehr klein. Der Analog/Digital-Wandler des Mikrocontrollers kann solch geringe Differenzen nicht erkennen. Die Lösung ist ein Subtrahierverstärker mittels OPV, der den Differenzdruck bildet und diesen mit dem Faktor zehn multipliziert.

[11] vgl. http://docs-europe.electrocomponents.com/webdocs/0ef4/0900766b80ef40fa.pdf (10.04.2013)
[12] http://upload.wikimedia.org/wikipedia/commons/thumb/3/3e/Pitot_principle.png/320px-Pitot_principle.png (10.04.2013)

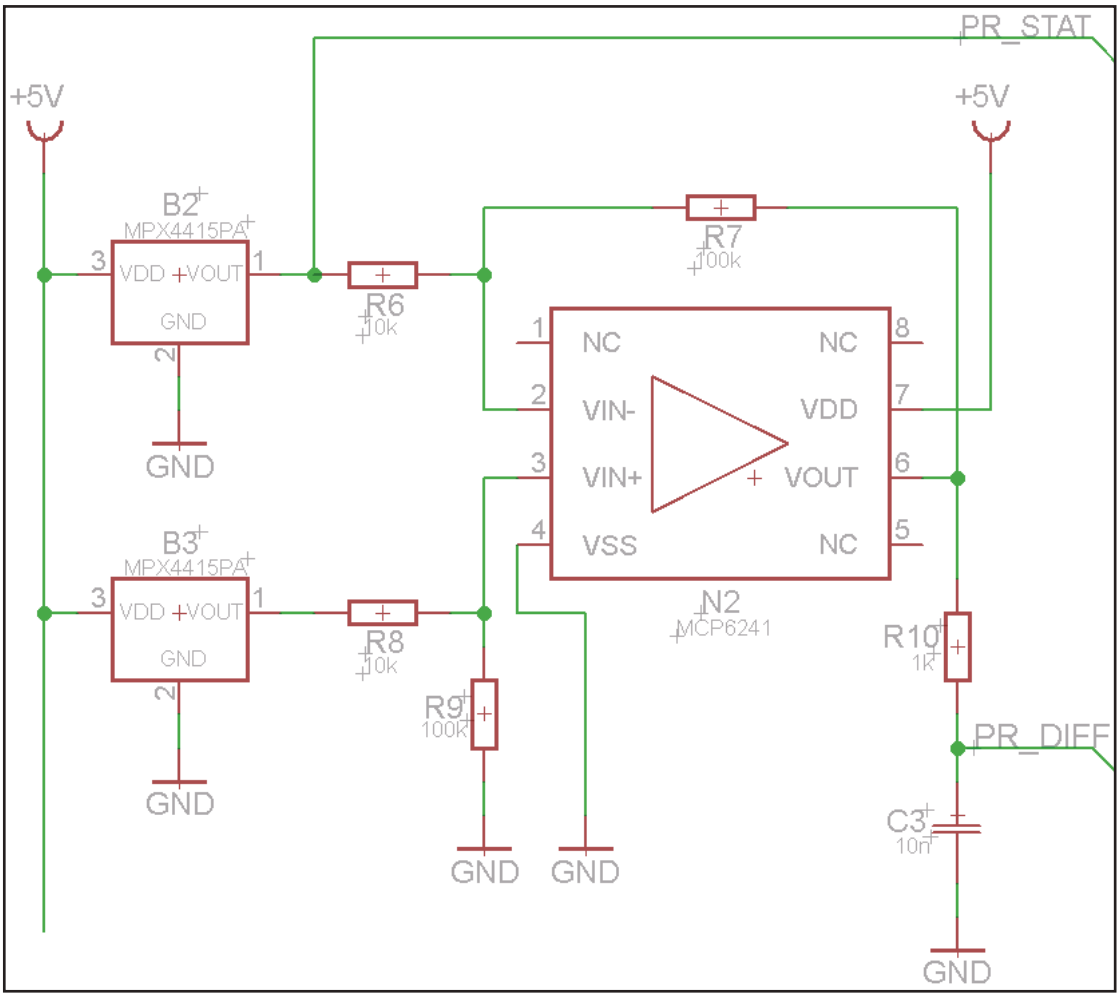


Abb. 16: Beschaltung des Subtrahierverstärkers

Der Ausgang des Staudrucksensors ist mit dem positiven Eingang des OPVs verbunden. Der Ausgang des statischen Drucksensors mit dem negativen Eingang.

Daraus kann der Differenzdruck gebildet werden.

Es konnte noch nicht festgestellt werden, ob der Tiefpass am Ausgang des OPVs das Messergebnis positiv bzw. negativ beeinflusst. Die Grundidee besteht darin, das Signal zu mitteln, um stabile Werte zu erhalten.

Berechnung:[13]

$$R_7 = R_9 = 100k\Omega$$

$$R_6 = R_8 = 10k\Omega$$

$$U_a = \frac{R_7}{R_6} \cdot (U_{in+} - U_{in-})$$

$$v = \frac{R_7}{R_6} = \frac{100k\Omega}{10k\Omega} = 10 \quad \ldots \quad Verstärkung$$

[13] vgl. http://www.elektronik-kompendium.de/sites/slt/0210153.htm (10.04.2013)

Geschwindigkeitsberechnung:[14]

$p_{statisch}$ = 100kPa = 1bar (4.169V)

Δp = 10.87Pa – 10870Pa (0.005V – 5.000V)

T = 293K (~20°C)

$$\Delta p = p_{stau} - p_{statisch}$$

Version 1: Ohne Temperatur

$$v = \sqrt{\frac{2 \cdot \Delta p \cdot R \cdot T}{p_{statisch}}} \cdot 3.6$$

v	Geschwindigkeit [km/h]
Δp	Differenzdruck [Pa]
ρ_N	Luftdichte 1.292 [kg/m³]

Version 2: Mit Temperatur

$$v = \sqrt{\frac{2 \cdot \Delta p \cdot R \cdot T}{p_{statisch}}} \cdot 3.6$$

v	Geschwindigkeit [km/h]
T	aktuelle Temperatur [K]
Δp	Differenzdruck [Pa]
$p_{statisch}$	statischer Luftdruck [Pa]
R	Gaskonstante Luft 287,058[J/(kg*K)]

Version 3: Mit Temperatur, Berechnung im Mikrocontroller

$$v = 25{,}967103 \cdot \sqrt{\frac{U_{\Delta p} \cdot T}{U_{p_{statisch}}}}$$

v	Geschwindigkeit [km/h]
$U_{\Delta p}$	Spannung Differenzdruck [V]
T	aktuelle Temperatur [K]
$U_{pstatisch}$	Spannung statischer Luftdruck [V]

[14] vgl. http://www.electro-mation.de/productattachments/index/download?id=47 (10.04.2013)

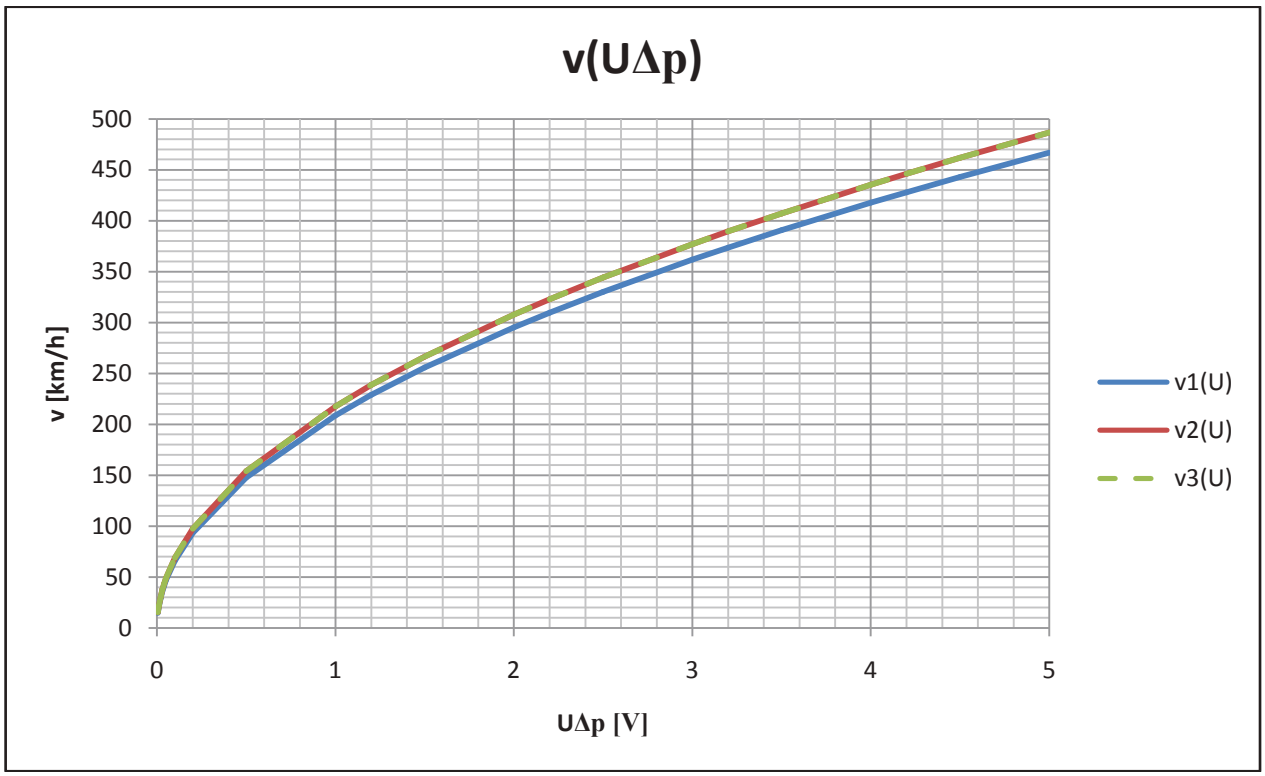


Abb. 17: Diagramm der Geschwindigkeiten

v1(U) … Ohne Temperatur
v2(U) … Mit Temperatur
v3(U) … Mit Temperatur, Berechnung im Mikrocontroller

Der blaue Graph berücksichtigt die Temperatur nicht. Ein kleiner Fehler ist immer vorhanden. Die rote und die grüne Linie decken sich. Bei beiden wird die Temperatur miteinbezogen.

Nachfolgende Diagramme zeigen die Auflösung bei niedrigen bzw. hohen Geschwindigkeiten. Weiters werden Maximum bzw. Minimum der Drucksensoren dargestellt. Die maximale Abweichung der Sensoren beträgt ±1.5kPa = ±69mV. Bei hohen Geschwindigkeiten ist der Fehler sehr gering. Eine Kalibrierung der Drucksensoren würde das Ergebnis nochmals verbessern, ist aber zurzeit noch nicht vorgesehen.
Wie in den Diagrammen weiters zu sehen ist, liegt der erste Wert bei etwa 15km/h. Bis zu einer Geschwindigkeit von 100km/h sind die Schritte noch sehr groß (2-5km/h pro Schritt). Ab dieser Geschwindigkeit geht der Graph in den brauchbaren Teil über und die Auflösung wird feiner.
X-Achse … Eingangsspannung in Volt, Vin [V]
Y-Achse … Geschwindigkeit TAS in Kilometer pro Stunde, Speed [km/h]
Gelb … Maximum
Blau … Normal
Rot … Minimum

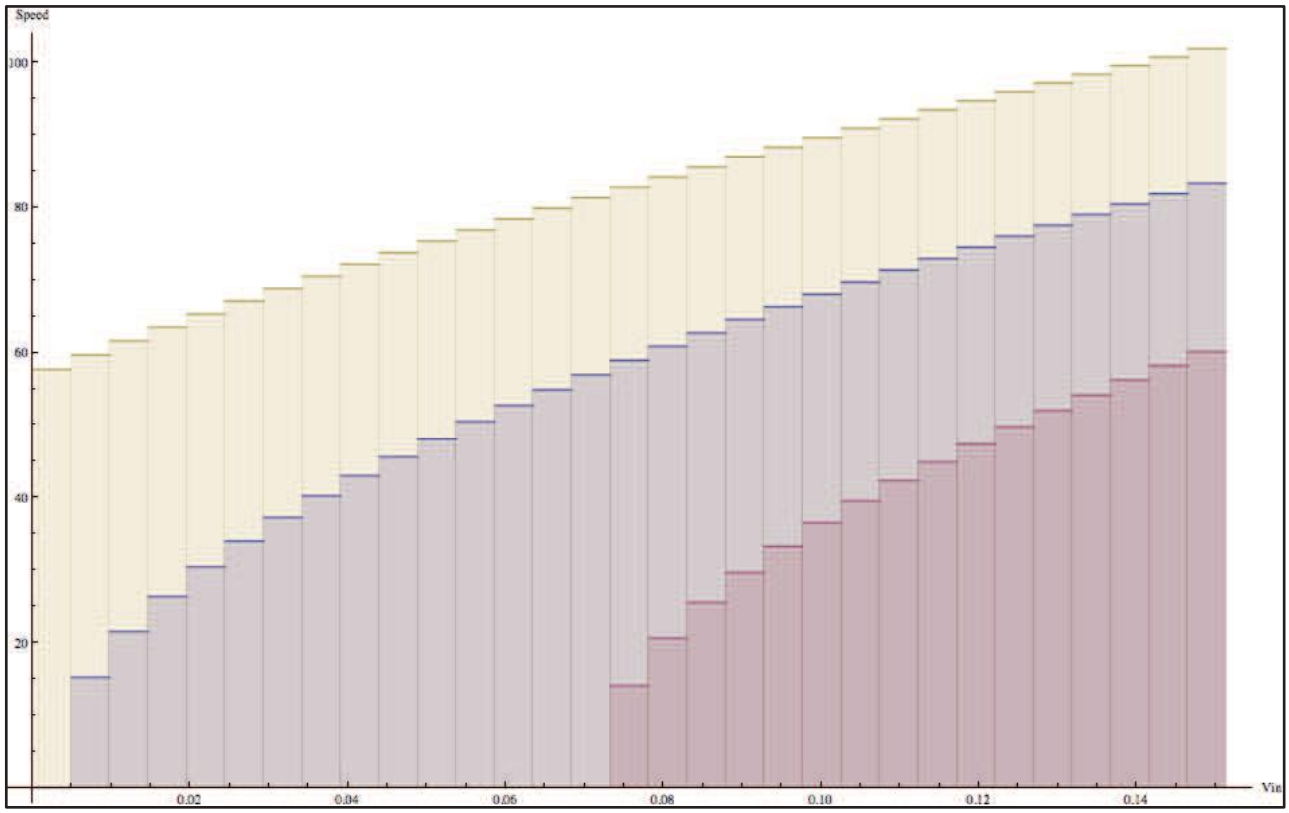


Abb. 18: Auflösung bei niedriger Geschwindigkeit

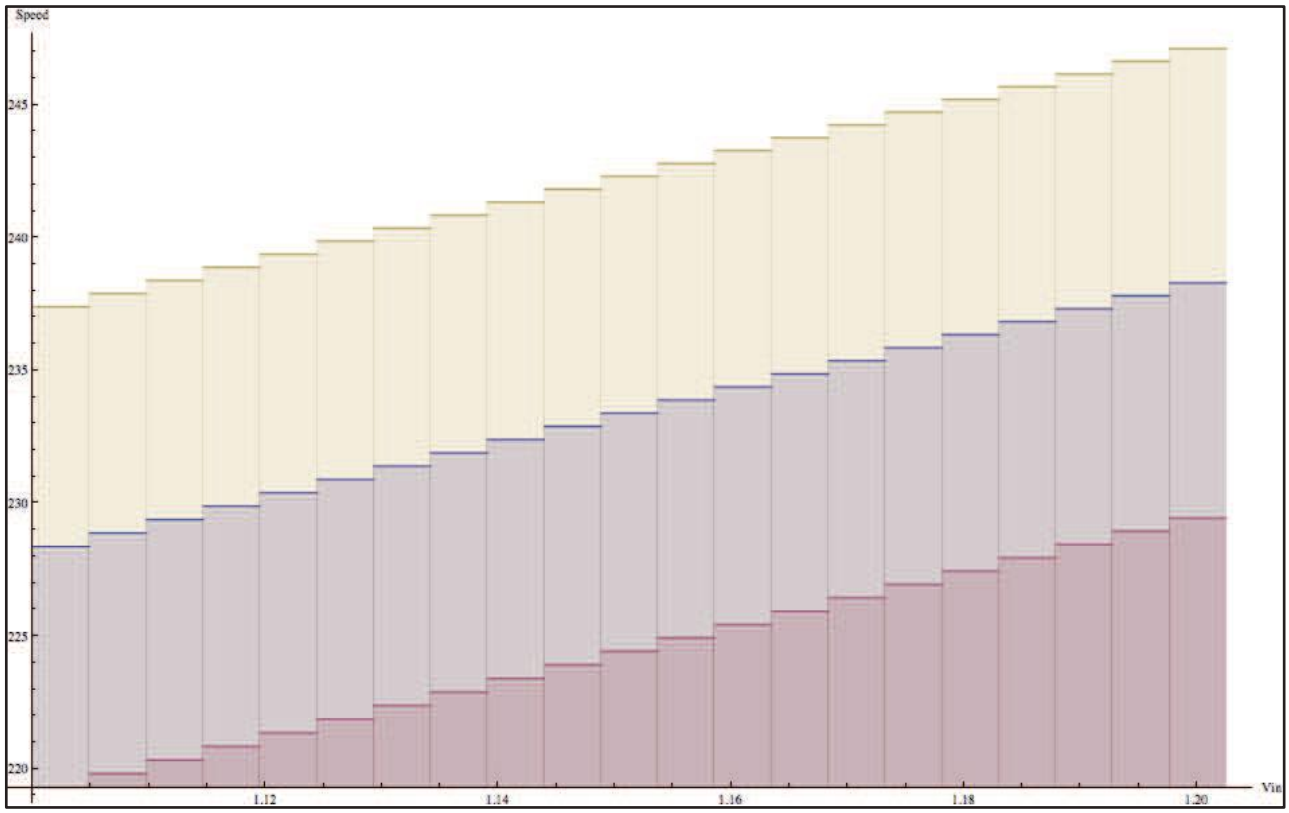


Abb. 19: Auflösung bei hoher Geschwindigkeit

4.1.4 Spannungsversorgung

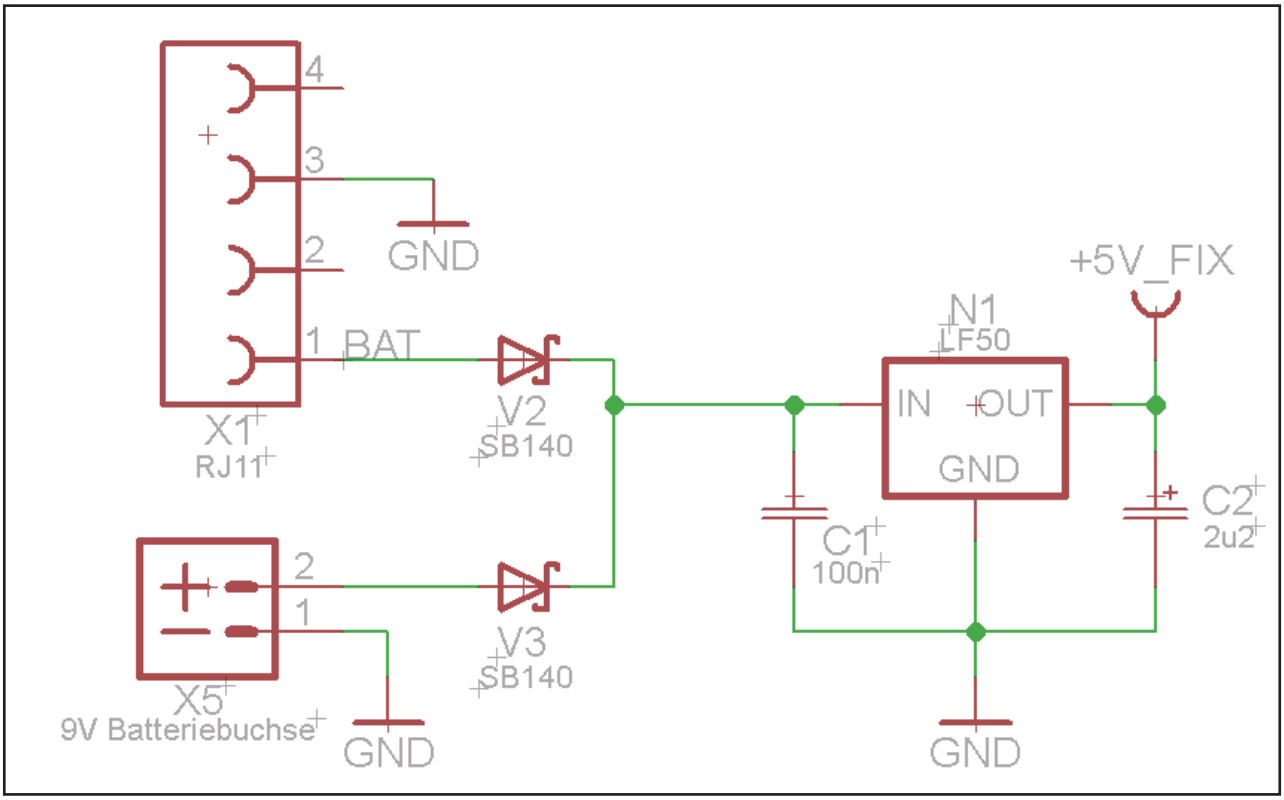


Abb. 20: Schaltung der Spannungsversorgung

Normalerweise wird das Produkt von der Batterie des Fliegers (12-15V) versorgt. Im Falle eines Stromausfalls übernimmt die Backup-Batterie (9V) sofort die Versorgung. Um dies zu bewerkstelligen, werden zwei Schottky-Dioden verwendet. Ist am Eingang einer dieser Dioden ein höheres Potenzial als am Ausgang, wird sie durchlässig. Da Schottky-Dioden einen Spannungsabfall von lediglich 0.2V haben, sind sie bestens für diese Anwendung geeignet.

Um die hohen Spannungen auf 5V zu reduzieren, wird ein Spannungswandler vom Typ LF50 verwendet. Im Gegensatz zu Step-Down-Konvertern treten keine hohen Frequenzen auf, die möglicherweise den Flugfunk stören könnten. Allerdings wird die überflüssige Energie in Wärme umgesetzt. Darunter leidet der Wirkungsgrad, und außerdem ist eine Kühlung unbedingt von Nöten. Wie bei jeder Versorgungsschaltung sind Kondensatoren zur Glättung und Stabilisation notwendig.

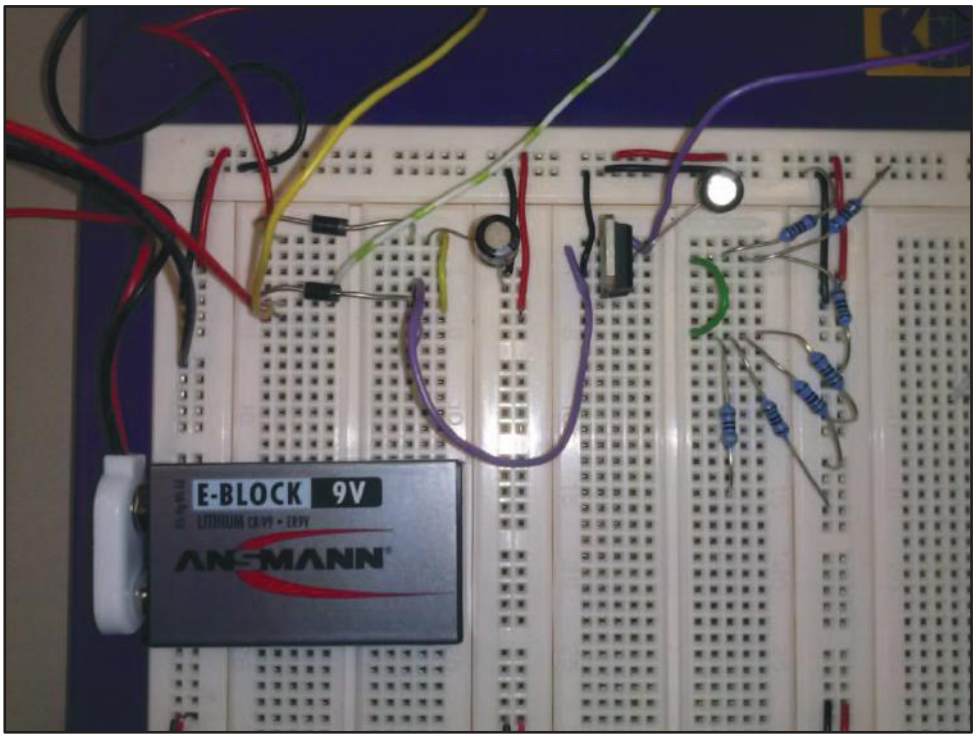


Abb. 21: Aufbau der Spannungsversorgung am Steckbrett

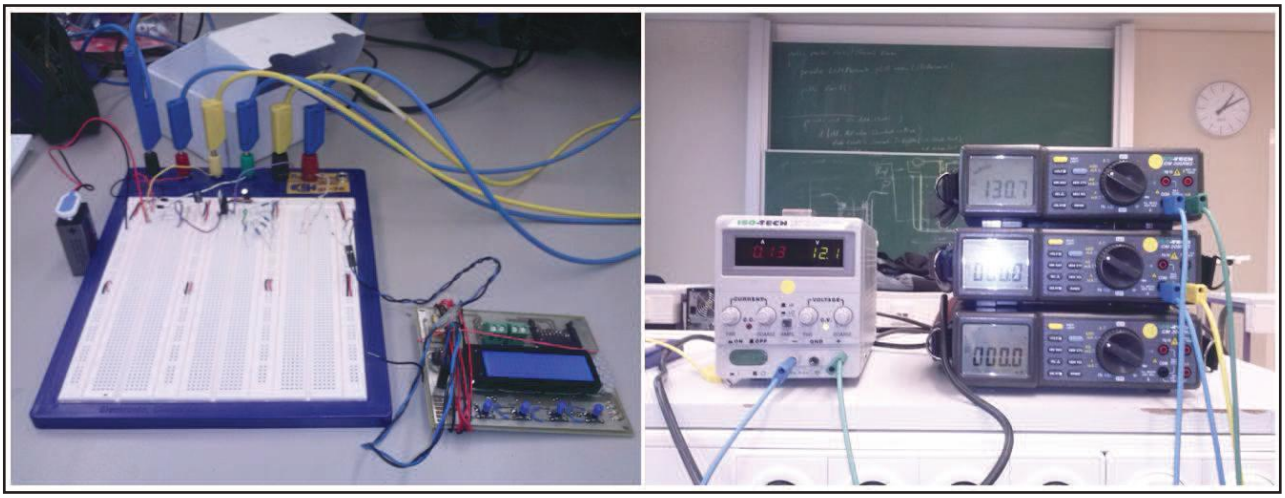


Abb. 22: Messaufbau der Spannungsversorgung

Sleepstrom:

Im Sleep-Modus soll der Strom so klein wie möglich sein. Laut Datenblattangaben des Mikrocontrollers und der Realtime-Clock liegt der Sleepstrom im nA-Bereich.
Der Ruhestrom des Spannungswandlers beträgt aber ca. 1mA. Aus diesem Grund muss er auf beiden Seiten abgeschlossen werden. Als Schalter werden Mosfets verwendet.

Gemessener Sleepstrom ohne LF50: *600±2nA*

4.1.5 MOSFET-Steuerschaltung

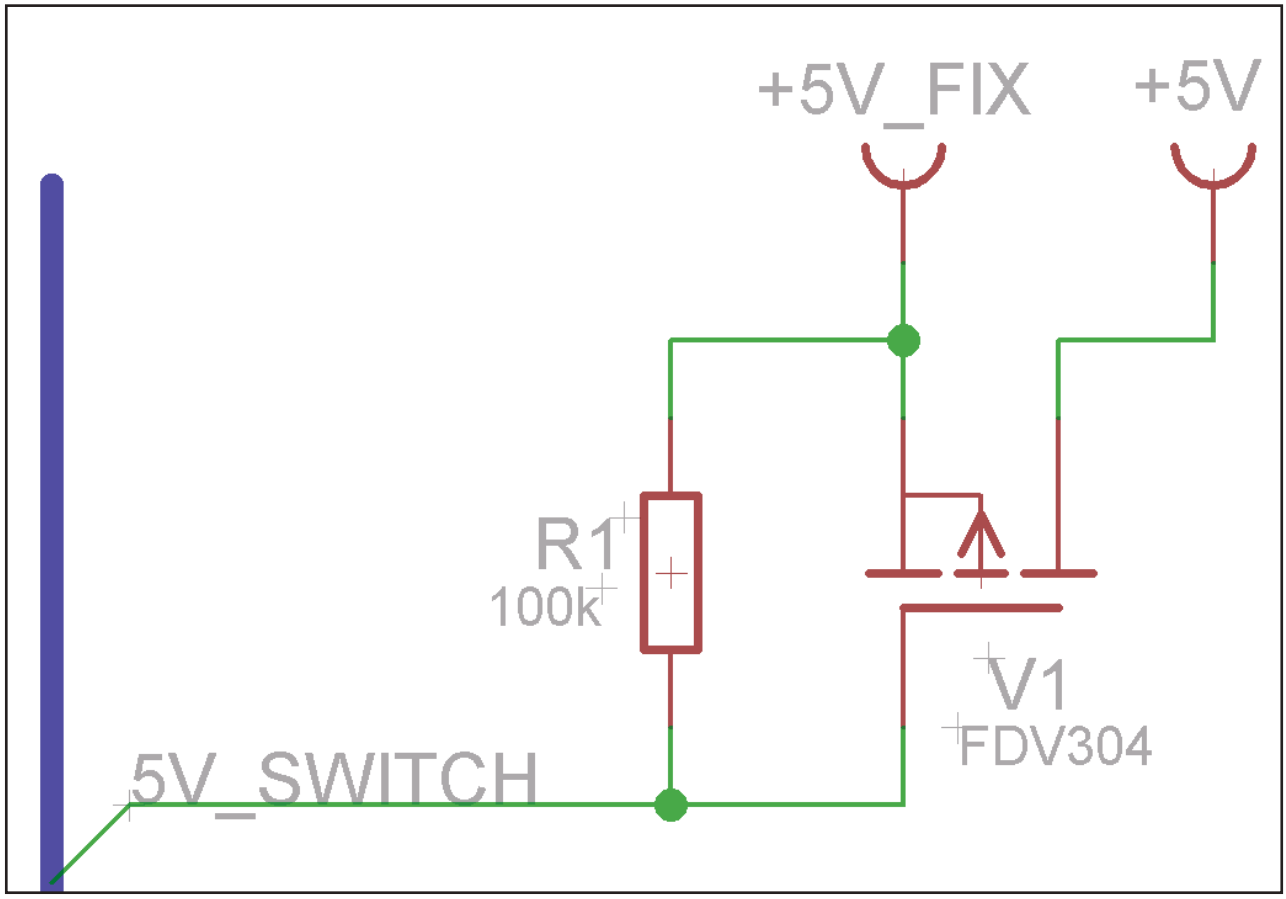


Abb. 23: Schaltung zum Abschalten aller Bauteile

Im Sleep-Modus ist es notwendig, alle nötigen Bauteile von der Spannungsversorgung zu trennen. Nur der Mikrocontroller und die Realtime-Clock hängen an *5V_FIX* und werden somit dauerhaft versorgt. Solange keine Spannungsdifferenz zwischen Gate und Source herrscht, ist der Mosfet nicht durchgeschaltet. Mit einem Low-Signal an *5V_SWITCH* von der MCU, wird der Mosfet reaktiviert und die externen Bauteile wieder versorgt.

Die Backup-Batterie muss, um die Sicherheit der Piloten zu gewährleisten, regelmäßig überprüft werden. Für diese Batterieprüfung ist es notwendig, die Spannung der Batterie unter Belastung zu überprüfen. Bevor die Batterie leer ist, muss eine entsprechende Warn-LED leuchten.
Vorgesehen ist eine Lithium-Ionen-Batterie, die kaum Selbstentladung aufweist.
Allerdings sind am Markt unzählig verschiedene 9V Batterien erhältlich, die unterschiedliche Entladekurven besitzen. Aus diesem Grund ist es äußerst schwierig ein geeignetes Verfahren für die Batterieüberprüfung zu entwickeln.
Grundsätzlich haben wir uns folgende einfache Lösung überlegt. Fällt der Spannungspegel unter einen gewissen Wert, so wird eine Warnung angezeigt.

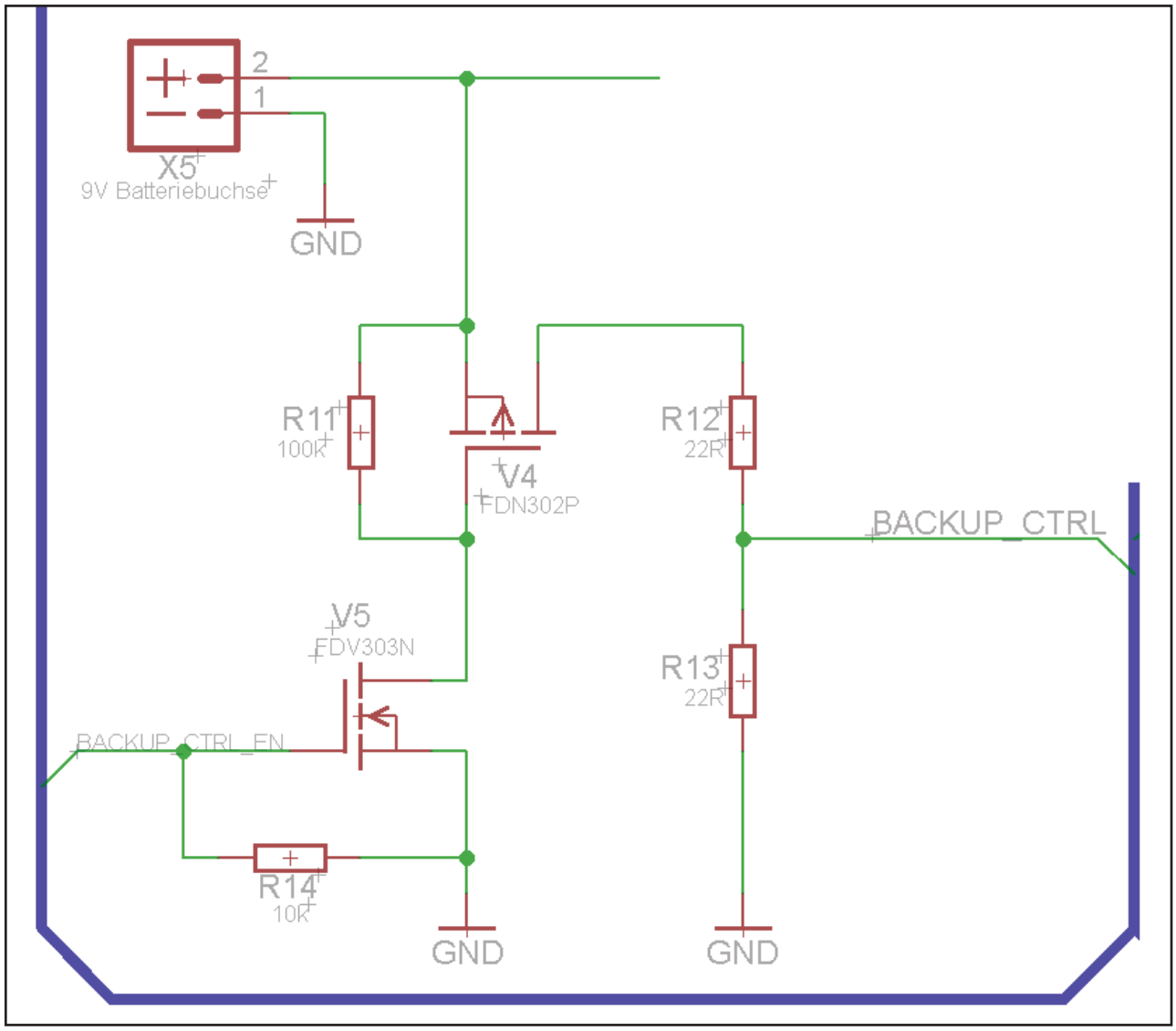


Abb. 24: Schaltung zum Einschalten der Batterieüberprüfung

Die Widerstände R_{12} und R_{13} begrenzen den Strom auf etwa 200mA (Normaler Arbeitsstrom des Produkts). Da die Spannung von 9V vor der Messung halbiert werden muss, stehen die Widerstände im Verhältnis 1:1.

Damit der Mikrocontroller nicht mit den 9V in Berührung kommt, sind zwei Mosfets notwendig. Das Prinzip ist das Selbe wie das der vorhergehenden Schaltung.

$$R_{ges} = \frac{U_{BAT}}{I_{Arbeitsstrom}} = \frac{9V}{0.2A} = 45\Omega$$

$$R_{12} = R_{13} = \frac{R_{ges}}{2} = \frac{45\Omega}{2} = 22.5\Omega$$

$$Gewählt: R_{12} = R_{13} = 22\Omega$$

4.1.6 Prototyp

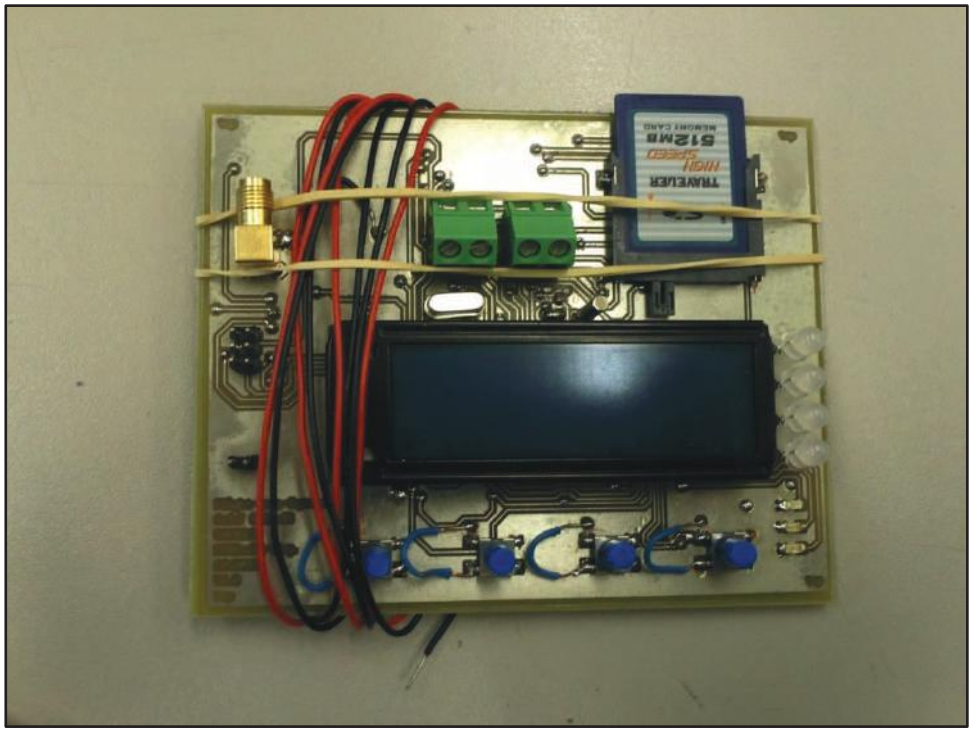


Abb. 25: Prototyp Vorderseite

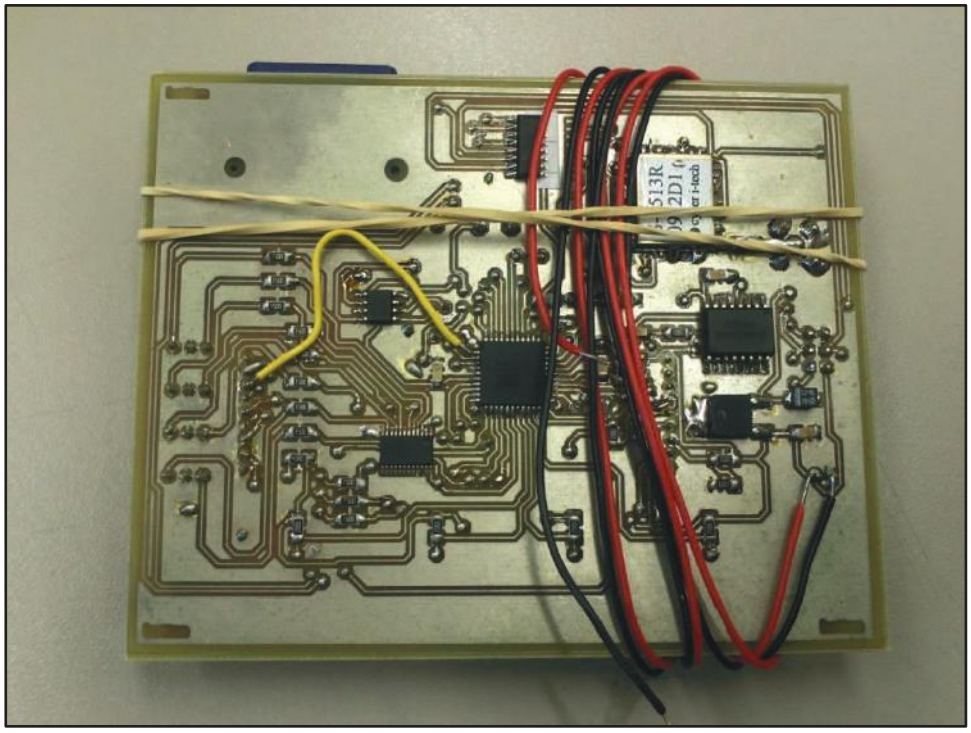


Abb. 26: Prototyp Rückseite

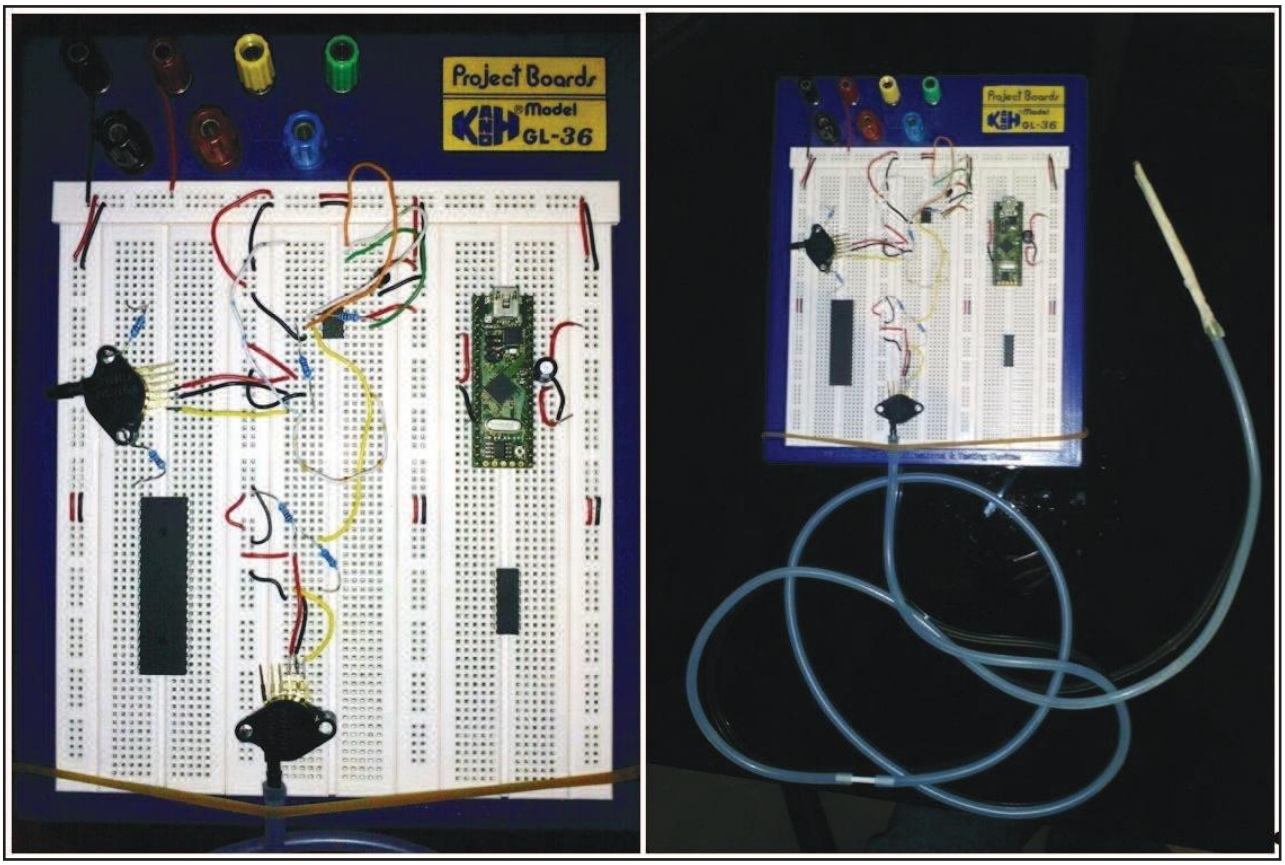


Abb. 27: Aufbau der TAS-Messung am Steckbrett

Ziel der Entwicklung der Prototypenplatine war es, die gesamte Hardware ansteuern zu können. Die Platine war bis auf kleine Fehler, die einfach behoben werden konnten, voll funktionstüchtig.

Da Ungewissheit herrschte, ob die Realisierung der TAS-Messung auf diese Art überhaupt möglich ist, wurde die entsprechende Schaltung zunächst auf einem Steckbrett aufgebaut. Zur Verbindung der Prototypenplatine und des Steckbretts werden vier Kabel verwendet. Jeweils eines für Masse, 5V, statischen Druck und Differenzdruck. Der Drucksensor für den Staudruck wird mit einem Schlauch nach außen verbunden. Dieser Schlauch muss exakt nach vorne ausgerichtet werden, um gute Ergebnisse zu erzielen.

4.1.7 Serienprodukt

Abb. 28: Serienprodukt Front On

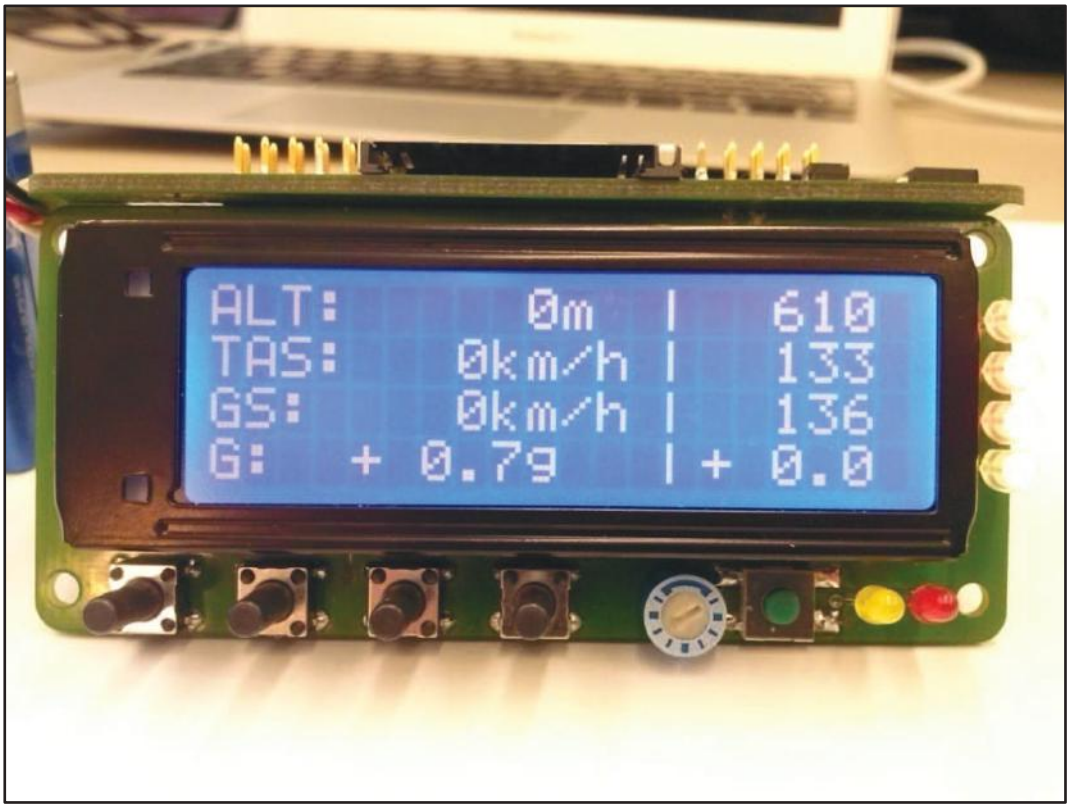


Abb. 29: Serienprodukt Front On 2

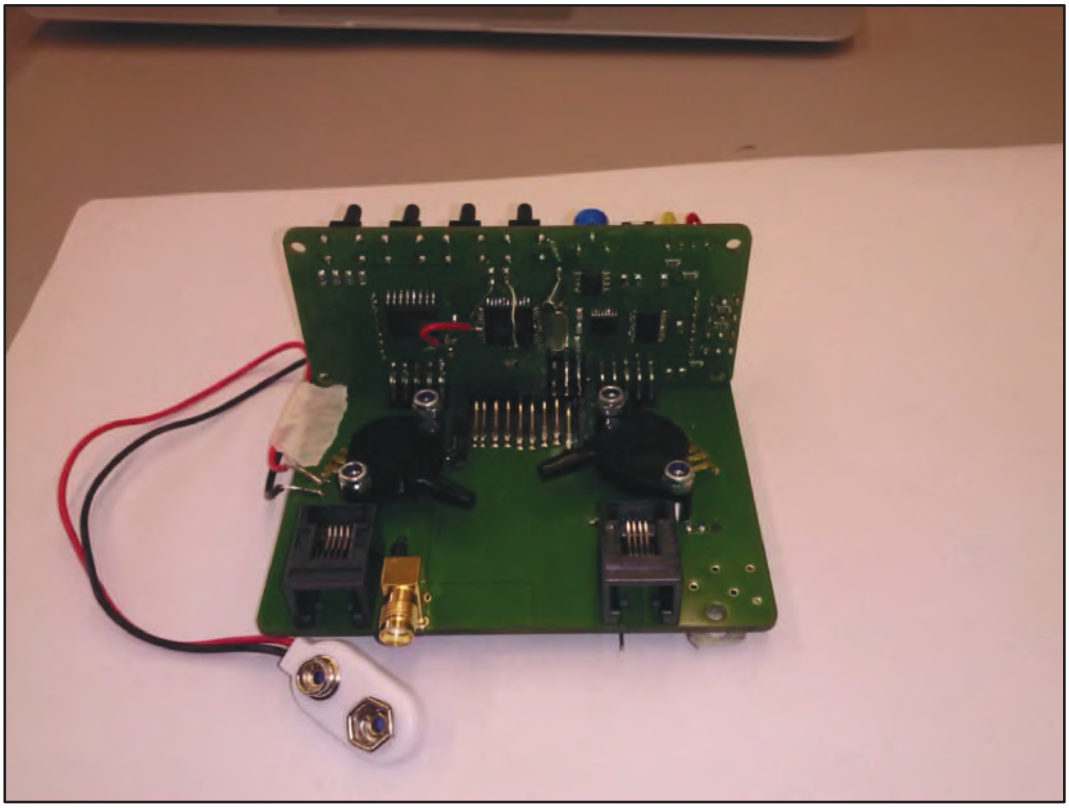


Abb. 30: Serienprodukt Back

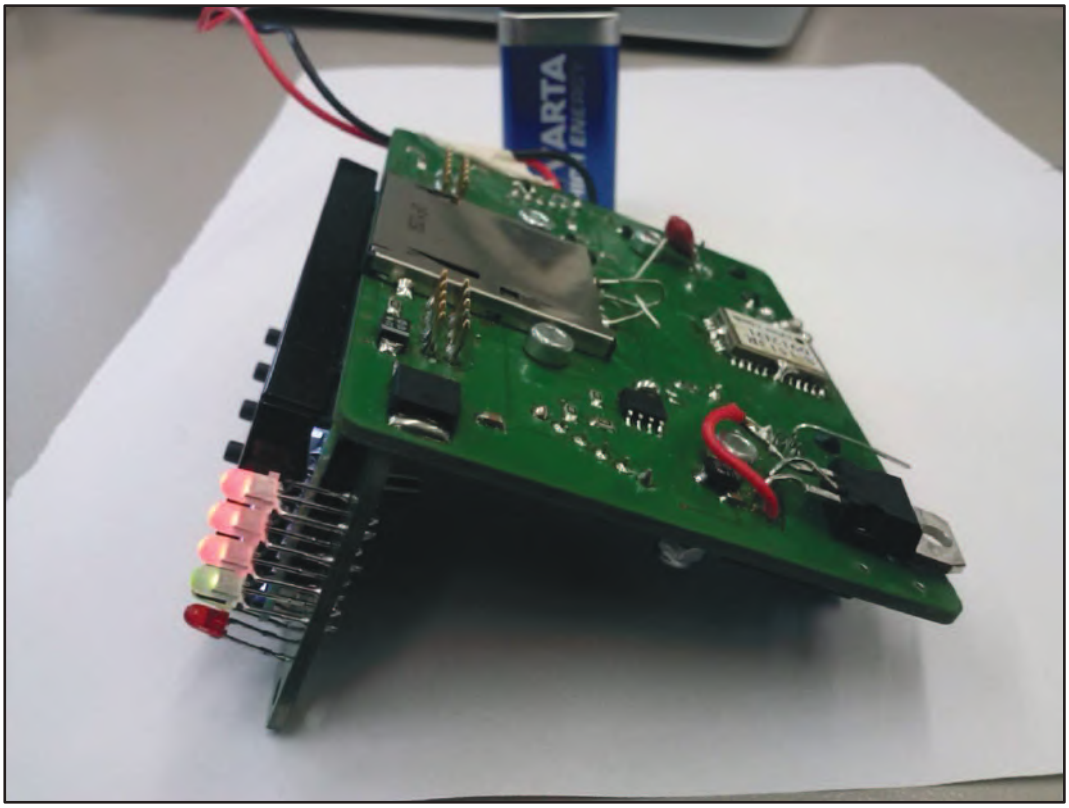


Abb. 31: Serienprodukt Side

Wie in den vorherigen Abbildungen zu sehen ist, besteht das Serienprodukt aus zwei Platinen. Die Verbindung erfolgt mit gewinkelten Steckverbindern (90°). Auf der Mainplatine befinden sich alle Ein- und Ausgabemodule (Display, LEDs, Taster, Trimmer, externer SD-Slot). Auf der Backplatine sind die Drucksensoren und der interne SD-Slot angebracht. Die externen Anschlüsse für den Temperatursensor und der Versorgungsspannung befinden sich hinten auf der Backplatine. Aufgrund der gewinkelten Ausführung braucht das Produkt im Cockpit nur wenig Platz.
Kleine Fehler waren anfangs noch vorhanden, konnten aber sehr schnell korrigiert werden. Diverse Änderungen werden noch notwendig sein, damit ein komplett serienreifes Produkt entsteht.

Abb. 32: Befestigung der Hardware (Tests mit dem Auto)

4.2 Mikrocontroller-Software

4.2.1 Allgemein

Die Mikrocontroller-Software ist modular aufgebaut, um Übersichtlichkeit zu waren und gleichzeitig einfache Erweiterbarkeit zu ermöglichen. Die unterste Ebene der Programmstruktur besteht aus den Interfaces, die zur Kommunikation auf den Bussystemen (I²C, SPI, USART) und außerdem zur AD-Wandlung nötig sind. Die Devices (Display, G-Sensor, GPS-Modul, ...) stellen die nächste Ebene dar, sie greifen auf die Funktionen der Interfaces zu, um mit den Hardwarekomponenten kommunizieren zu können. An oberster Stelle stehen die Implementations, sie sind dafür verantwortlich, dass die Daten von den Devices verarbeitet, angezeigt und ggf. gespeichert werden. Wenige Teile der Software (Sleep-Modus-Verwaltung) weichen aufgrund der Komplexität ihrer Aufgabe von diesem Schema ab, was jedoch kein Problem darstellt.
Folgende Abbildung zeigt schematisch den Aufbau der Software anhand von einigen Modulen:

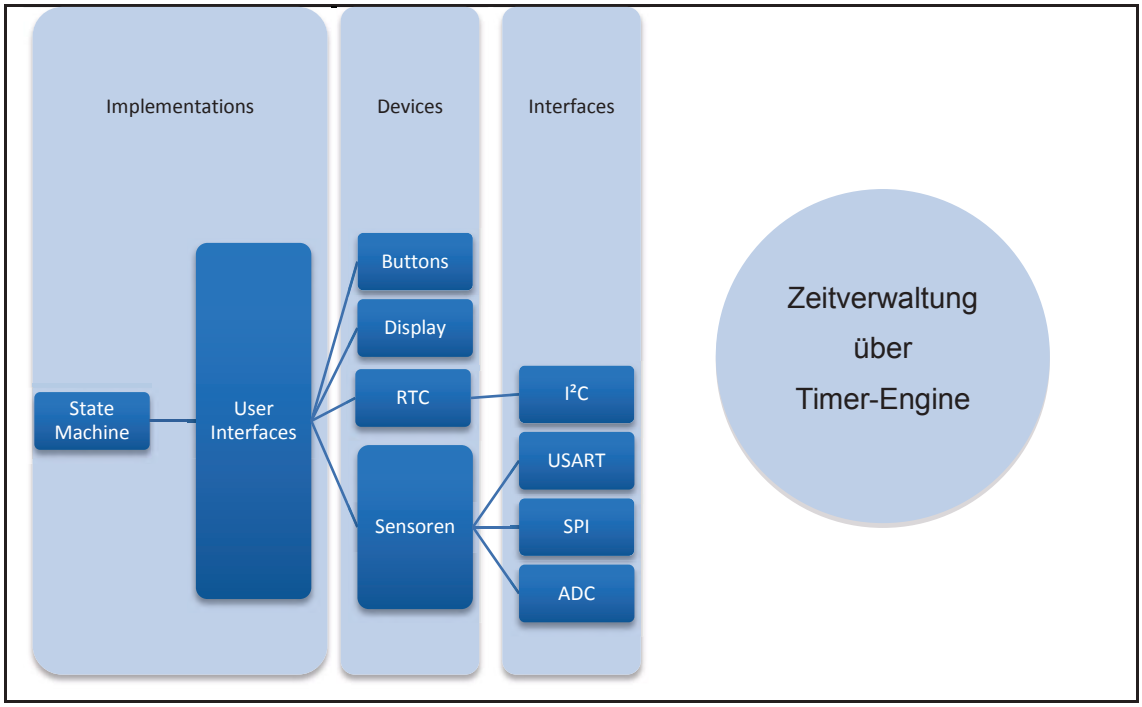


Abb. 33: Schematischer Aufbau der Mikrocontroller-Software

4.2.2 Timer-Engine

Aufgrund des komplexen Timings der Messungen, Ausgaben und Speicherungen, ist es nicht möglich, das Programm einfach in einer Endlosschleife mit Delays o.ä. zu programmieren. Als Lösung dieses Problems wurde ein Modul programmiert, das sich der zeitlichen Verwaltung der Funktionen annimmt. Es muss auch sichergestellt sein, dass alle Funktionen nicht direkt von der Timer Interrupt Routine aufgerufen werden. In diesem Fall können bis zum Ende der Funktion, keine weiteren Interrupts mehr ausgelöst werden.
Jede Funktion muss zuerst bei der Timer-Engine angemeldet werden, dabei muss ein Intervall übergeben werden. Alle angemeldeten Funktionen werden dann immer wieder aufgerufen, wobei der zeitliche Abstand dem übergegebenen Intervall entspricht.

Durch die Anmeldung wird der neue Task zu einem internen Array hinzugefügt:

```
Typedef struct{
        SBOOL enabled;
        TimerEngineFunction function;
        Unsigned long interval;
        Unsigned long current;
}TimerEngineTaskInfo;

TimerEngineTaskInfo timerEngineTaskInfo[MAXTASKS];
```

Zu jedem Task werden ein Funktionspointer, das Intervall, und die bereits verstrichene Zeit (*current*) gespeichert. Tasks können außerdem einzeln ein- und ausgeschaltet werden (*enabled*).

Die Timer-Engine beginnt erst dann mit dem Aufrufen der Funktionen, wenn man *TimerEngine_run()*ausführt. In dieser Funktion wird der Timer des Mikrocontrollers gestartet, und anschließend beginnt das Programm mit der Haupt-Endlosschleife. In dieser Schleife werden alle Tasks immer wieder überprüft und entschieden, ob sie ausgeführt werden müssen. Als Zeitgeber verwendet die Timer-Engine den Timer0 des Mikrocontrollers.

```
ISR(TIMER0_COMPA_vect){
        for(i=0; i<taskCounter; i++){
                if(timerEngineTaskInfo[i].enabled == ETRUE)
                        timerEngineTaskInfo[i].current++;
        }
}
```

Über den Timer des Mikrokontrollers werden alle *current* hochgezählt, aber nur wenn die jeweiligen Tasks auch gestartet sind. Da der Timer so eingestellt ist, dass er exakt jede Millisekunde den Interrupt auslöst, entspricht der Wert in *current* der verstrichenen Zeit in Millisekunden.

```
while(1){
        for(i=0; i<taskCounter; i++){
                if(timerEngineTaskInfo[i].enabled == ETRUE){
                        TIMSK0 &= ~(1<<OCIE1A);
                        if(timerEngineTaskInfo[i].current >= timerEngineTaskInfo[i].interval){
                                timerEngineTaskInfo[i].current %= timerEngineTaskInfo[i].interval;
                                TIMSK0 |= (1<<OCIE1A);
                                timerEngineTaskInfo[i].function();
                        }
                        TIMSK0 |= (1<<OCIE1A);
                }
        }
}
```

In der Endlosschleife wird bei allen Tasks überprüft, ob sie eingeschaltet sind. Wenn dies der Fall ist und wenn der Wert von *current* den Wert von *interval* erreicht, wird *current* wieder rückgesetzt und die Funktion einmal ausgeführt. Da die *current* Variable von der Interruptroutine bearbeitet wird und außerdem vom Typ *long* ist – *long* Variablen benötigen mehr als einen Assembleraufruf zum lesen und schreiben – müssen alle Zugriffe auf diese Variable als kritischer Abschnitt definiert werden. Vor dem Zugriff wird also der Timer mit *TIMSK0&=~(1<<OCIE1A);* ausgeschaltet und anschließend wieder reaktiviert. Er muss auch unbedingt vor der Ausführung der Funktion wieder eingeschaltet werden, da diese meist längere Zeit benötigt und der Timer immer nur sehr kurz deaktiviert werden darf. Der Modulo-Operator „%“ stellt sicher, dass *interval* sooft von *current* subtrahiert wird, bis es kleiner als *interval* ist.

```
TimerEngine_init();
TimerEngine_addTask(function1,1000,ETRUE);
TimerEngine_addTask(function2,100,ETRUE);
TimerEngine_run();
```

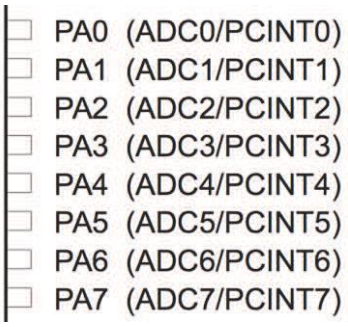
Diese Aufrufe initialisieren die Timer-Engine, fügen 2 Tasks hinzu und starten zum Schluss den Timer und die Endlosschleife. `function1` wird alle 1000ms und `function2` alle 100ms aufgerufen. Alle Tasks müssen vor dem starten der Timer-Engine angemeldet werden, sie können aber anschließend mit `TimerEngine_enableTask(…, bool enable)` gestartet und gestoppt werden. Dadurch ist die Ausfallsicherheit des Programms erhöht, da nur eine boolesche Variable gesetzt werden muss, anstatt einen Eintrag zu einem Array hinzuzufügen bzw. aus diesem einen zu löschen. Zweiteres macht Probleme, weil Manipulationen von Arrays (insbesondere das Löschen von Einträgen) in der Schleife, die ständig durch alle Einträge geht, Fehler verursacht.

Zusätzlich zur Taskverwaltung wurde noch eine Zählerfunktion in der Timer-Engine realisiert. Man kann also einen Pointer auf eine Variable vom Typ long übergeben. Die Variable wird dann automatisch jede Millisekunde inkrementiert.

Es wäre natürlich auch möglich das Ganze mit einer dynamischen Liste, anstatt eines Arrays zu realisieren. Da aber alle Tasks schon zu Beginn angemeldet werden müssen, bleibt die Länge des Arrays ohnehin zur Laufzeit konstant. Da Arrays weniger fehleranfällig sind, wurde auf diese zurückgegriffen.

4.2.3 ADC

Zur einfachen Verwendung vom AD-Wandler ist es von Vorteil, ein Modul zu programmieren, das eigenständig das periodische Einlesen der benötigten Pins übernimmt. Die ADC-Eingänge befinden sich beim Mikrocontroller ATmega644PA am PORTA. Von außen soll das Modul auf folgende Funktionen reduziert sein:




```
void ADC_init(char mask);
int ADC_getValue(char pinADC);
```

Mit dem Übergabeparameter *mask* wird dem ADC-Modul zu Beginn mitgeteilt, an welchen Pins am PORTA überhaupt gemessen werden muss. Dann wird mit Hilfe der Timer-Engine alle 10ms ein anderer Pin eingelesen und intern gespeichert. Bei fünf aktivierten Pins wird somit jeder Pin einmal alle 50ms gemessen. Mit *ADC_getValue(pin)* kann von außen jederzeit der zuletzt eingelesene Wert des Pins abgefragt werden. Da es sich bei der Variable, in der die Werte gespeichert sind, um ein Integer Array handelt, muss – ähnlich zur Timer-Engine – der Interrupt für die Dauer des Auslesens deaktiviert werden. Nach der Initialisierung kann jedes Modul mit nur einer Funktion auf analoge Eingänge zugreifen.

4.2.4 Buttons

Im vorliegenden Projekt müssen 4 Taster (Buttons) von diesem Modul verwaltet werden. Die einfache Verwendung von Außen ist das Wichtigste. Weiters muss darauf geachtet werden, dass beim softwaretechnischen Entprellen der Taster kein Zeitverlust aufgrund von Delays auftritt. Nur durch die Verwendung der Timer-Engine ist dies überhaupt möglich.
Um die nötige Einfachheit zu gewährleisten, wird beim Initialisieren des Moduls lediglich ein Funktionspointer übergeben.

```
typedef void ( * Button_ActionFunction ) (char ButtonNumber, BUTTON_ACTION action, int duration);
```

Die übergebene Funktion wird immer dann aufgerufen, wenn ein Taster gedrückt, losgelassen oder gehalten wird. Wenn ein Taster gehalten wird, wird die Funktion alle 100ms aufgerufen und es wird zusätzlich die bisherige Haltezeit übergeben. Wichtig dabei ist, dass die Funktion nicht direkt von der Interrupt-Routine gestartet, sondern mithilfe der Timer-Engine aufgerufen wird.

4.2.5 State Machine

Mithilfe der State Machine und den User Interfaces werden die einzelnen Anzeigen getrennt. Dadurch wird alles viel übersichtlicher und einfacher. Die erste Aufgabe des Moduls ist es, eine Funktion beim Initialisieren des Buttons-Moduls anzumelden, damit es bei Eingaben benachrichtigt wird. Aufgrund dieser Eingaben muss die State Machine in den richtigen Zustand wechseln. Es gibt für jeden Zustand ein entsprechendes User Interface, diese sind nach außen alle gleich strukturiert.
Wenn zum Beispiel nach dem Einschalten die „prog.“-Taste lange gedrückt wird, muss die State Machine in den Zustand „Passwort Eingabe“ gehen und das entsprechende Interface zum Eingeben des Passworts anzeigen. Zusätzlich hat das Modul zu gewährleisten, dass während es sich im Zustand „Passwort Eingabe“ befindet, alle Tastenereignisse an das entsprechende User Interface weitergeleitet werden.

Es existieren folgende Zustände bzw. User Interfaces:

- Anzeige der derzeitigen Messwerte (Default)
- Passwort Eingabe
- Auswahl der Einstellungen
  - Passwort Änderung
  - Einstellung Zeit/Datum
  - Einstellung Grenzwerte
  - Einstellung Einheiten

4.2.6 User Interfaces

Jedes User Interface ist folgendermaßen aufgebaut:

```
void UI_name_show();
char UI_name_buttonEvent(char ButtonNumber, BUTTON_ACTION action, int duration);
void UI_name_finalize();
```

Wenn die State Machine in einen anderen Zustand wechselt, ruft sie zuerst *finalize()* des derzeitigen User Interfaces auf, danach *show()* des anderen. Außerdem wird *buttonEvent()* des derzeitigen User Interfaces aufgerufen, wenn die State Machine selbst ein Taster Ereignis empfängt.
In der Funktion *show()* muss das UI seine Variablen initialisieren, das Display mit seiner Anzeige überschreiben und ggf. eine repaint Funktion bei der Timer-Engine anmelden. Die repaint Funktion ist dann nötig, wenn sich die Anzeige automatisch aktualisieren soll.

In *buttonEvent()* reagiert das Modul auf Tasteneingaben. Über den Rückgabewert teilt es der State Machine mit, ob z.B. die Eingabe abgeschlossen ist. In *finalize()* müssen nur noch die Variablen genullt und die repaint Funktion ausgeschaltet werden.

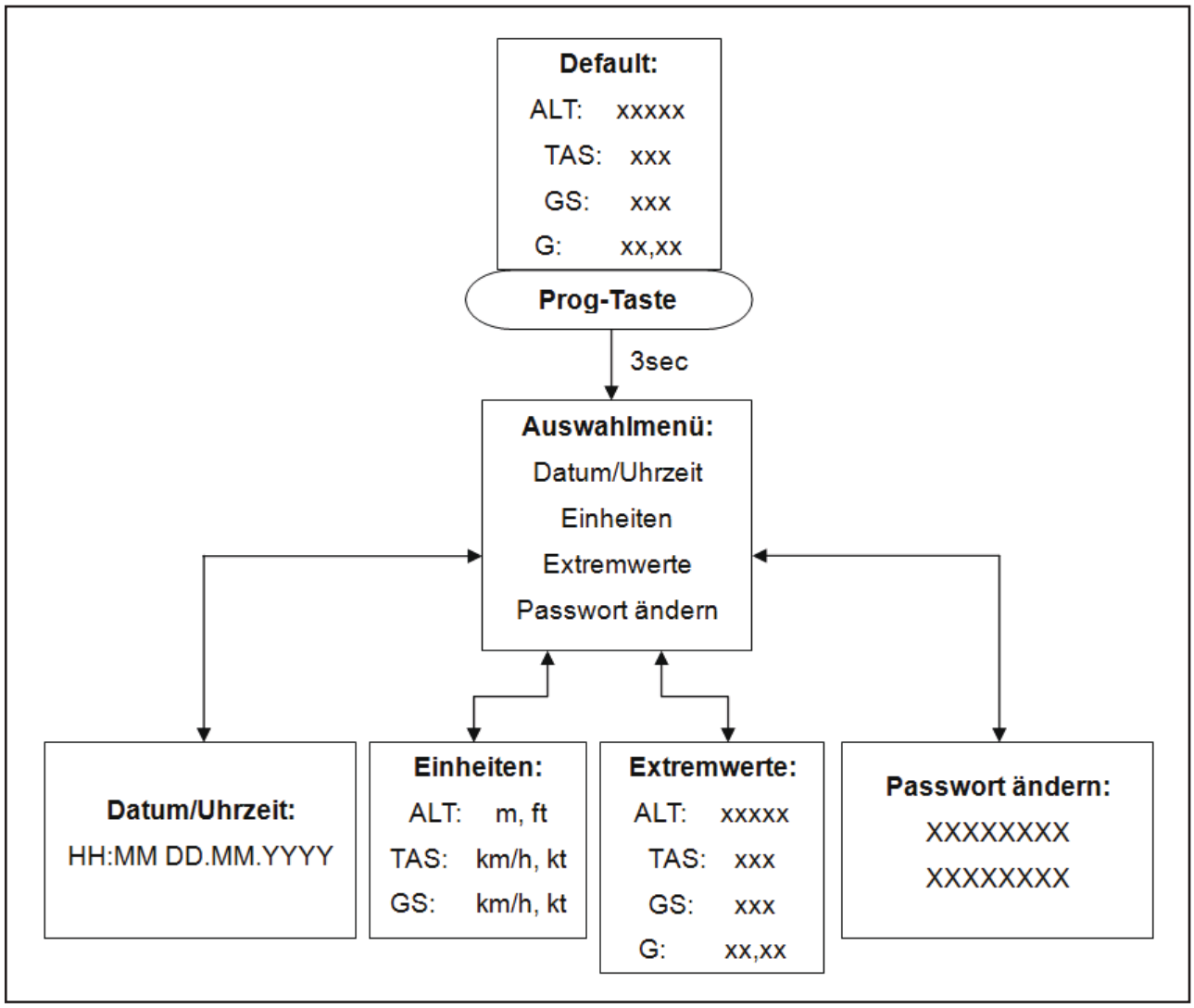


Abb. 34: Zustandsdiagramm des User Interfaces

4.3 PC-Software

4.3.1 Allgemein

Zu Beginn haben wir die Programmierung der PC-Software den Wunsch- bzw. Kann-Kriterien zugeordnet. Es hat sich aber sehr schnell herausgestellt, dass die gemessenen Werte auf Richtigkeit überprüft werden müssen. Die PC-Software wurde somit parallel zur Hardware-Prototypen entwickelt. Dadurch konnten die Messwerte sofort überprüft werden.
In der zweiten und dritten Klasse haben wir uns mit der Programmiersprache Java beschäftigt. Infolge dessen entschieden wir uns für diese Sprache.
Die Software bietet eine genaue Analyse der Flugdaten. Am Anfang wird eine FLX-Datei von der SD-Karte importiert. In einem Liniendiagramm werden die einzelnen Messdaten (True Airpseed, Groundspeed, Flughöhe, g-Belastung) angezeigt. Mittels Zoomfunktion kann man die Verläufe näher betrachten. Die Einheit von Flughöhe und Geschwindigkeit kann zwischen Meter und Fuß bzw. zwischen Kilometer pro Stunde und Knoten umgeschaltet werden. Weiters ist es möglich eine GPX- oder KML-Datei zu exportieren, um den 3D-Flug in Google Earth anzuzeigen. Ist Google Earth auf dem Rechner nicht installiert, so kann Google Maps die exportierte GPX-Datei ebenfalls interpretieren.

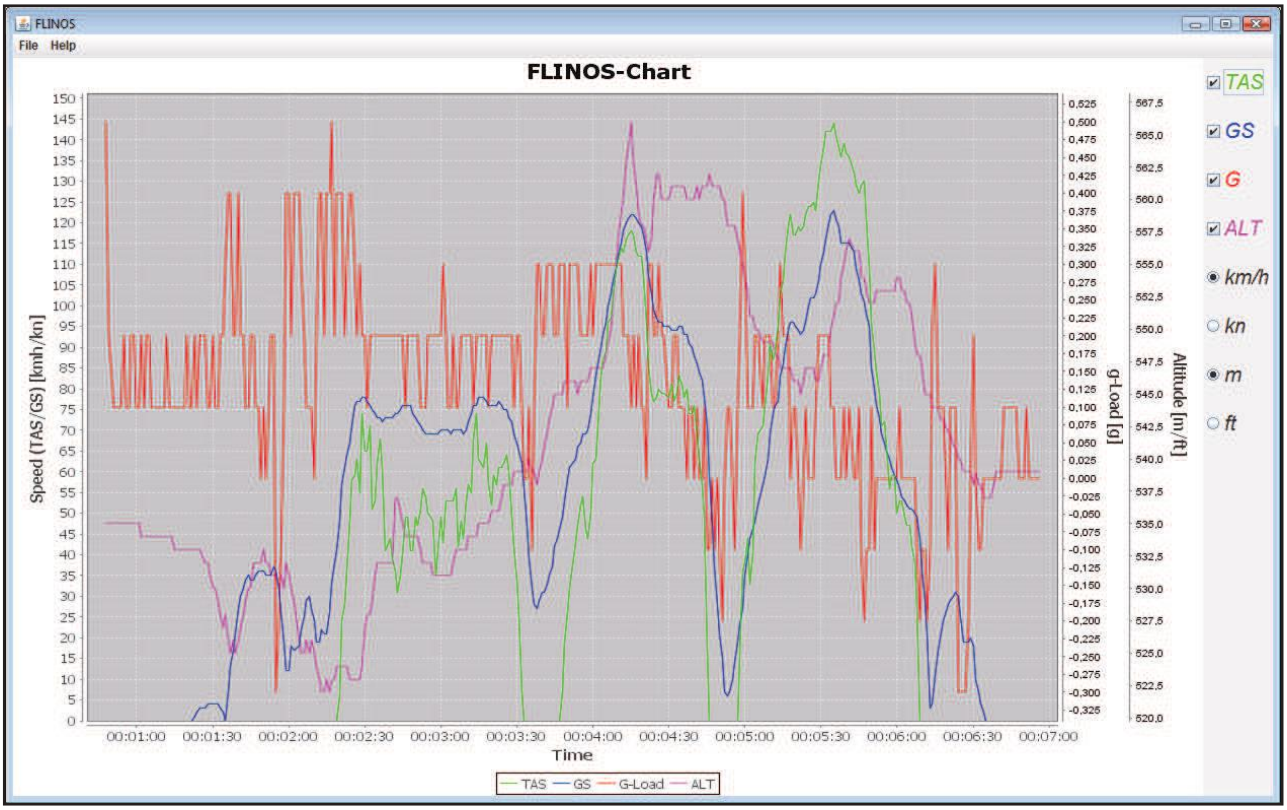


Abb. 35: GUI des Programmes

Das Programm ist in folgende drei Bereiche aufgeteilt:

- Hauptframe
- Einstellmöglichkeiten
- Datenspeicherung

Folgende Grafik gibt einen genaueren Einblick in die Programmstruktur:

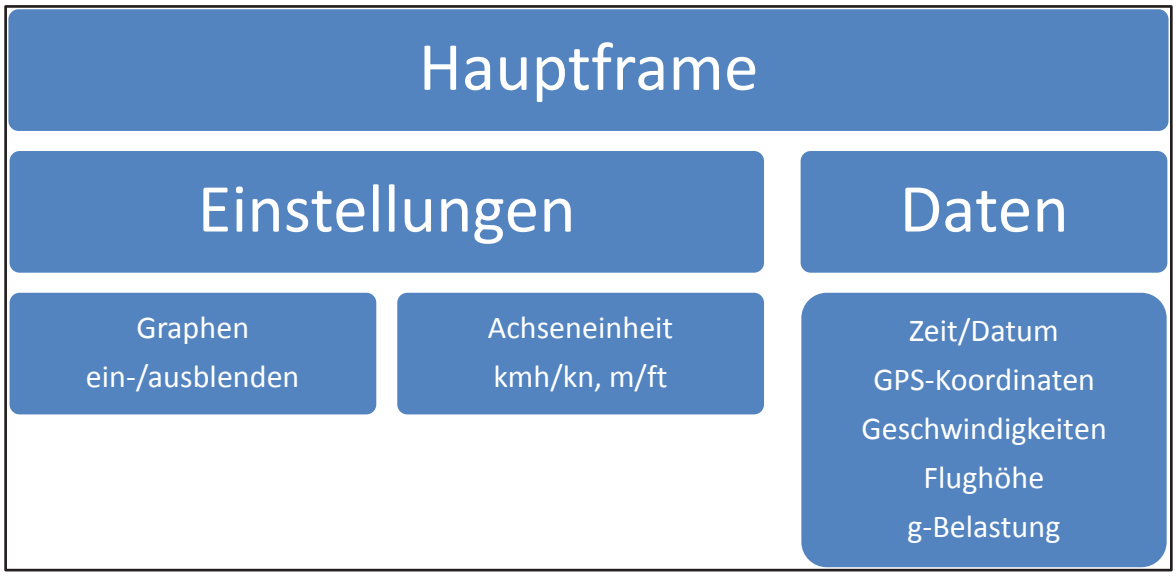


Abb. 36: Programmstruktur der PC-Software

Der Hauptframe besteht aus einem JFreeChart-Diagramm und diversen Action-Listenern. JFreeChart bietet die einfache Möglichkeit Diagramme in Java zu erstellen. Im Internet steht ein kostenloser Download zur Verfügung. Es ist lediglich die Einbindung von zwei Dateien im Projektordner notwendig.
Eine Menübar ermöglicht Aktionen wie „FLX-Datei laden“ oder „KML-Datei exportieren“. Die Informationen werden mittels JOptionPanes angezeigt.
Die Einstellmöglichkeiten des Diagramms sind am rechten Bildschirmrand angebracht. Um eine Linie im Diagramm ein- oder auszublenden, muss einfach ein Häckchen bei der entsprechenden Checkbox gesetzt werden. Die Umstellung der Y-Achse, Meter und Fuß bzw. Kilometer pro Stunde und Knoten, erfolgt mittels Setzen eines Radio-Buttons.
Jede importierte FLX-Datei enthält eine unterschiedliche Anzahl von Datensätzen. Daher muss die interne Datenspeicherung dynamisch erfolgen. Es wurde eine Klasse erstellt, die die nötigen Variablen beinhaltet. Diese Membervariablen können mit Getter- und Setter-Methoden gelesen bzw. gesetzt werden. Eine Arraylist vom Typ dieser Klasse wurde erstellt. Beim Zugriff auf einen Datensatz wird nur ein Index benötigt.

4.3.2 Hauptframe

```
public class LineChartFrame extends JFrame implements ActionListener
{
```

Die Hauptframe-Klasse *LineChartFrame* erbt von der Klasse *JFrame*. Da eine Mehrfachvererbung in Java nicht möglich ist, wird das Interface *ActionListener* implementiert.

```
    public LineChartFrame()
    {
        super("FLINOS");
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
```

Im Konstruktor der Klasse *LineChartFrame* wird mittels ***super()***der Konstruktor der Oberklasse *JFrame* aufgerufen. Als Parameter wird der Programmtitel mitgegeben.

Der zweite Befehl dient zum ordnungsgemäßen Beenden des Programms.

Im Konstruktor werden des Weiteren die vorher definierten Variablen initialisiert.

```
    public void actionPerformed(ActionEvent e) {
            if(e.getActionCommand() == "TAS")
                    rendererSpeed.setSeriesLinesVisible(0, ((JCheckBox)e.getSource()).isSelected());
```

In der Methode *actionPerformed()* werden die Informationen der Menübuttons, Radiobuttons und Checkboxes ausgewertet.

Mit *e.getActionCommand()* == "TAS" wird der Befehl mit dem Button verglichen. Ist das Ergebnis ***true***, so wird der Befehl des jeweiligen Buttons ausgeführt. Im oben angeführten Beispiel wird die Sichtbarkeit des TAS-Graphen aktiviert bzw. deaktiviert.

Abb. 37: Diagrammeinstellungen

```
    class menuItemExportGPX implements ActionListener
    {
    class menuItemExportKML implements ActionListener
    {
    class menuItemLoadFile implements ActionListener
    {
```

Für die Menüitems Export-GPX, Export-KML und Load-File wurde jeweils eine eigene Klasse angelegt. Diese Klassen implementieren den *ActionListener*. Unter den Menüpunkten Export-GPX und Export-KML kann man die Erzeugung von Dateien veranlassen, die Google Earth interpretieren vermag. Zu Beginn wird mit Load-File die Flugdatei (.FLX) eingelesen. Anschließend können die Exporte durchgeführt werden.

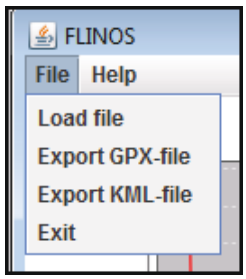


Abb. 38: Menüitems

Die bis jetzt erklärten Programmausschnitte waren für das Grundgerüst des Layouts und deren Funktionen notwendig. Im Nachfolgenden wird die Programmierung des JFreeChart-Diagrammes etwas näher erklärt.

```
    public XYDataset createDatasetGLoad()
    {
        final TimeSeries seriesGLoad = new TimeSeries("G-Load");

        for(FlightDataPoint fdp : flightDataPoints)
        {
        seriesGLoad.addOrUpdate(new Second(     fdp.getSecond(),
                                                fdp.getMinute(),
                                                fdp.getHour(),
                                                fdp.getDay()+1,
                                                fdp.getMonth()+1,
                                                fdp.getYear()),
                                                fdp.getG());
        }

        datasetGLoad = new TimeSeriesCollection();
        datasetGLoad.addSeries(seriesGLoad);

        plot.setDataset(1, datasetGLoad);
        return datasetGLoad;
    }
```

Grundsätzlich besteht ein JFreeChart-Diagramm aus einem Chart. Dieser Chart baut auf einem sogenannten Plot auf. Der Plot dient zum Design des Hintergrundes und zum Hinzufügen von Datenlinien.

In dem vorigen Programmausschnitt wird der Graph für die g-Belastung erzeugt. Um nun Daten hinzufügen zu können, wird eine *TimeSeries* verwendet. In der for-Schleife wird jeder Datensatz von der ArrayList *flightDataPoints* gelesen und in *seriesGLoad* eingetragen.

Die Methode *createDatasetGLoad()* dient also zum Hinzufügen von Datensätzen im Diagramm.

```
        rendererSpeed.setSeriesLinesVisible(0, true);    //TAS
        rendererSpeed.setSeriesPaint(0, Color.GREEN);
        rendererSpeed.setSeriesShapesVisible(0, false);
```

Um die vorher hinzugefügte Linie optisch ändern zu können, ist ein Renderer notwendig.

Mit der ersten Anweisung wird die TAS-Linie sichtbar gemacht. Mit der Zweiten ändert man die Farbe auf grün und mit dem dritten Befehl werden die Shapes (Punkte auf dem Graph) unsichtbar gemacht.

```
private JFreeChart createChart(final XYDataset dataset) {

        final JFreeChart chart = ChartFactory.createTimeSeriesChart(
                "FLINOS-Chart",                 // chart title
                "Time",                     // x axis label
                "Speed (TAS/GS) [kmh/kn]",  // y axis label
                createDatasetSpeed(),                   // data
                true,                           // include legend
                true,                           // tooltips
                false                           // urls
        );

        chart.setBackgroundPaint(Color.white);

        plot = chart.getXYPlot();
        plot.setBackgroundPaint(Color.lightGray);
        plot.setDomainGridlinePaint(Color.white);
        plot.setRangeGridlinePaint(Color.white);

        rendererSpeed.setSeriesLinesVisible(0, true);    //TAS
        rendererSpeed.setSeriesPaint(0, Color.GREEN);
        rendererSpeed.setSeriesShapesVisible(0, false);

        rendererSpeed.setSeriesLinesVisible(1, true);    //GS
        rendererSpeed.setSeriesPaint(1, Color.BLUE);
        rendererSpeed.setSeriesShapesVisible(1, false);

        plot.setRenderer(rendererSpeed);

        final NumberAxis axis2 = new NumberAxis("g-Load [g]");
        axis2.setLabelFont(new Font("Tahoma", Font.BOLD, 14));
        axis2.setLabelPaint(new Color(50,50,50));
        axis2.setAutoRangeIncludesZero(false);
        createDatasetGLoad();
        plot.setRangeAxis(1, axis2);
        plot.mapDatasetToRangeAxis(1, 1);
        rendererGLoad.setSeriesPaint(0, Color.RED);
        rendererGLoad.setSeriesLinesVisible(0, true);
        rendererGLoad.setSeriesShapesVisible(0, false);
        plot.setRenderer(1, rendererGLoad);

        final NumberAxis axis3 = new NumberAxis("Altitude [m/ft]");
        axis3.setLabelFont(new Font("Tahoma", Font.BOLD, 14));
        axis3.setLabelPaint(new Color(50,50,50));
        axis3.setAutoRangeIncludesZero(false);
        createDatasetAltitude();
        plot.setRangeAxis(2,axis3);
        plot.mapDatasetToRangeAxis(2, 2);
        rendererAltitude.setSeriesPaint(0, Color.MAGENTA);
        rendererAltitude.setSeriesLinesVisible(0, true);
        rendererAltitude.setSeriesShapesVisible(0, false);
        plot.setRenderer(2, rendererAltitude);

        final DateAxis axis = (DateAxis) plot.getDomainAxis();
        axis.setDateFormatOverride(new SimpleDateFormat("HH:mm:ss"));
        return chart;
    }
```

Das komplette Diagramm wird mit der Methode `createChart()` erzeugt. Dort werden Datensätze zum Plot hinzugefügt, das Design der Graphen mittels Renderer verändert und zusätzliche Y-Achsen hinzugefügt.

```
	public static void main(String[] args) {
		new LineChartFrame();
	}
```

In der main-Methode des Projektes wird lediglich der Frame erzeugt. Eigenschaften des Frames (Größe des Fensters, Position, Sichtbarkeit, usw.) wurden zuvor im Konstruktor definiert.

4.3.3 Einstellungen

```
public class SettingsPanel extends JComponent
{
	JCheckBox checkBoxTAS;
	JCheckBox checkBoxGS;
	JCheckBox checkBoxG;
	JCheckBox checkBoxALT;

	JRadioButton radioButtonKMH;
	JRadioButton radioButtonKN;
	JRadioButton radioButtonM;
	JRadioButton radioButtonFT;
```

In der Klasse `SettingsPanel` werden die Einstellungen, die im Hauptframe rechts zu sehen sind, ausprogrammiert. Es werden diverse Checkboxen und Radiobuttons als Membervariablen angelegt und anschließend im Konstruktor initialisiert.
Der objektorientierte Aufbau ermöglicht einen einfachen Zugriff auf die Membervariablen vom Hauptframe aus.

4.3.4 Daten

```
public class FlightDataPoint{
        private int second;
        private int minute;
        private int hour;
        private int day;
        private int month;
        private int year;
        private long tasKMH;    //kmh
        private long gsKMH;             //kmh
        private double g;
        private long altitudeM; //m
        private String latitude;
        private String longitude;
```

Die Datenspeicherung erfolgt in der Klasse `FlightDataPoint`. Wie in dem vorigen Programmteil zu sehen ist, werden die Membervariablen als ***private*** definiert. So ist ein direkter Zugriff auf die Members von außerhalb nicht möglich. Die einzige Möglichkeit die Daten zu lesen oder zu setzen erfolgt über die Getter- und Setter-Methoden.

```
        public int getSecond(){
                return second;
        }
        public void setSecond(int second){
                this.second = second;
        }
```

```
        public void setAllInfos(String flxLine){
        setHour(Integer.parseInt(flxLine.substring(0, 2)));
        setMinute(Integer.parseInt(flxLine.substring(2, 4)));
        setSecond(Integer.parseInt(flxLine.substring(4, 6)));
        setDay(Integer.parseInt(flxLine.substring(6, 8)));
        setMonth(Integer.parseInt(flxLine.substring(8, 10)));
        setYear(Integer.parseInt(flxLine.substring(10, 14)));
        setG(Double.parseDouble(flxLine.substring(46, 50))/10);
        setAltitudeM(Integer.parseInt(flxLine.substring(35, 40)));
        setTasKMH(Integer.parseInt(flxLine.substring(40, 43)));
        setGsKMH(Integer.parseInt(flxLine.substring(43, 46)));
        setLatitude(flxLine.substring(14, 24));
        setLongitude(flxLine.substring(24, 35));
        }
```

In der Hauptklasse werden die Daten von der FLX-Datei eingelesen und anschließend in dieser Klasse abgespeichert. Es wäre sehr aufwendig für jedes Datensegment (Sekunde, Minute, Stunde usw.) die entsprechende Setter-Methode aufzurufen. Mit der Funktion `setAllInfos()` ist es möglich, alle Variablen zentral an einer Stelle zu setzen. Übergeben wird der Funktion eine gesamte Zeile der FLX-Datei. Anschließend wird der String zerteilt und die Setter-Methoden aufgerufen.

4.4 Kostenaufstellung

4.4.1 Stückliste Mainplatine

1	2	3	4	5	6	7
Lfd. Nr.	Stück-zahl	Ein-heit	Benennung	Lieferant / Bestell-nummer	RoHS	Bemerkung
1	1	STK	IC ATmega 644PA D1	RS Components 719-3979	J	TQFP
2	1	STK	IC CD74HC4050M96G4 D2	RS Components 6627131	J	Pegelwandler, SMD
3	1	STK	IC PCA9555PW D3	RS Components 483-7222	J	IO-Port 16-bit, TSSOP
4	1	STK	IC PCF8563FT4 B1	RS Components 436-8300	J	RTC, SO
5	1	STK	IC MMA2244KEGR2 B2	Reichelt Elektronik 752-2336	J	Accelerometer,x-axis,20g,Vout,SOIC16
6	1	STK	DISP 204BL-4 DIP H1	Reichelt Elektronik LCD 204BL-4 DIP	J	4x20 Zeichen, blau, Modulmaße: 75 x 27 mm
7	4	STK	LED DUO H2-H5	Conrad 187496-62	J	LED Bi-Colour 3mm rot/grün
8	1	STK	LED H6	RS Components 228-5922	J	3mm rot
9	1	STK	LED H7	RS Components 228-5966	J	3mm gelb
10	1	STK	WID 47R R1	HTL-Salzburg -	J	SMD 0805
11	5	STK	WID 10k R2-R6	HTL-Salzburg -	J	SMD 0805
12	2	STK	WID 1k5 R7-R8	HTL-Salzburg -	J	SMD 0805
13	1	STK	WID 1k R10	HTL-Salzburg -	J	SMD 0805
14	10	STK	WID 150R R11-R20	HTL-Salzburg -	J	SMD 0805
15	1	STK	TRIM 5k R9	Reichelt Elektronik 75H 5,0K	J	Cermet-Miniatur-Regler, liegend, 6mm 5,0 K-Ohm
16	3	STK	KERKO 100n C1-C2, C6	HTL-Salzburg	J	SMD 0805
17	1	STK	KERKO 10n C3	HTL-Salzburg	J	SMD 0805
18	2	STK	KERKO 22p C4-C5	HTL-Salzburg	J	SMD 0805
19	1	STK	QU 16.000MHz Q1	Reichelt Elektronik 16,0000-HC49U-S	J	Cl: 32 pF, Toleranz: 30ppm
20	1	STK	QU 32.768kHz Q2	RS Components 478-9268	J	LFXTAL002997, Durch-steckmontage
21	4	STK	TAST PRINT S1-S4	Reichelt Elektronik TASTER 3301D	J	vert. Montage, Höhe 12,5mm
22	1	STK	TAST SMD S5	Reichelt Elektronik TASTER 9313	J	vert. Montage, Höhe 3,1mm reset
23	1	STK	CON Steckerleiste ISP X1	HTL-Salzburg -	J	6x2
24	2	STK	CON Steckerleiste 90° X2-X3	RS Components 360-6588	J	2,54mm 90° 10-polig

4.4.2 Materialkalkulation Mainplatine

1	2	3	4	5	6	
Lfd. Nr.	Stück-zahl	Ein-heit	Benennung	Lieferant / Bestellnummer	Preis Einzel	Preis Gesamt
1	1	STK	IC ATmega 644PA, D1	RS Components / 719-3979	5,480	5,480
2	1	STK	IC CD74HC4050M96G4, D2	RS Components / 6627131	0,160	0,160
3	1	STK	IC PCA9555PW, D3	RS Components / 483-7222	1,930	1,930
4	1	STK	IC PCF8563FT4, B1	RS Components / 436-8300	1,460	1,460
5	1	STK	IC MMA2244KEGR2, B2	Reichelt Elektronik / 752-2336	6,790	6,790
6	1	STK	DISP 204BL-4 DIP, H1	Reichelt Elektronik / LCD 204BL-4 DIP	28,550	28,550
7	4	STK	LED DUO, H2-H5	Conrad / 187496-62	0,560	2,240
8	1	STK	LED, H6	RS Components / 228-5922	0,126	0,126
9	1	STK	LED, H7	RS Components / 228-5966	0,126	0,126
10	1	STK	WID 47R, R1	HTL-Salzburg / -	0,030	0,030
11	5	STK	WID 10k, R2-R6	HTL-Salzburg / -	0,030	0,150
12	2	STK	WID 1k5, R7-R8	HTL-Salzburg / -	0,030	0,060
13	1	STK	WID 1k, R10	HTL-Salzburg / -	0,030	0,030
14	10	STK	WID 150R, R11-R20	HTL-Salzburg / -	0,030	0,300
15	1	STK	TRIM 5k, R9	Reichelt Elektronik / 75H 5,0K	0,780	0,780
16	3	STK	KERKO 100n, C1-C2, C6	HTL-Salzburg /		
17	1	STK	KERKO 10n, C3	HTL-Salzburg /		
18	2	STK	KERKO 22p, C4-C5	HTL-Salzburg /		
19	1	STK	QU 16.000MHz, Q1	Reichelt Elektronik / 16,0000-HC49U-S	0,140	0,140
20	1	STK	QU 32.768kHz, Q2	RS Components / 478-9268	0,578	0,578
21	4	STK	TAST PRINT, S1-S4	Reichelt Elektronik / TASTER 3301D	0,161	0,644
22	1	STK	TAST SMD, S5	Reichelt Elektronik / TASTER 9313	0,350	0,350
23	1	STK	CON Steckerleiste ISP, X1	HTL-Salzburg / -		
24	2	STK	CON Steckerleiste 90°, X2-X3	RS Components / 360-6588	0,274	0,548

Übertrag:	
Materialpreis	**€ 50,472**
Materialgemeinkosten (10%)	**€ 5,047**
Materialverkaufspreis	**€ 55,519**

Zusätzlich anfallende Kosten:

- Platine (ca. 20€)
- Gehäuse
- Frontblende

4.4.3 Stückliste Backplatine

1	2	3	4	5	6	7
Lfd. Nr.	Stück-zahl	Ein-heit	Benennung	Lieferant / Bestell-nummer	RoHS	Bemerkung
1	1	STK	IC GPS-1513 D1	RS Components 704-3307	J	GPS Receiver, SMT
2	1	STK	OP MCP6241-E/P N2	RS Components 403-018	J	PDIP, 110 dB
3	1	STK	VREG LF33CDT N3	RS Components 355-4106	J	Spannungswandler, SMD
4	1	STK	VREG LF50CV N1	RS Components 355-4257	J	Spannungswandler
5	2	STK	SENS MPS4115AP B1-B2	RS Components 717-6527	J	15 - 115 kPa, 46mV/kPa
6	1	STK	SENS TMP36GT9Z -	RS Components 709-2784	J	TO-92
7	1	STK	FET P FDV304 V1	RS Components 354-4913	J	SOT-23
8	1	STK	FET N FDV303 V4	RS Components 354-4890	J	SOT-23
9	1	STK	FET P FDN302 V5	RS Components 671-0400	J	SuperSOT
10	2	STK	ZDIO SB140 V2-V3	RS Components 701-0199	J	Z-Diode
11	1	STK	CON Batterieclip 9V X5	RS Components 489-021	J	für PP3 9V-Block, 250mm, 0.8mm drill
12	1	STK	CON SMA Stecker X3	RS Components 616-3400	J	90°, 50 Ohm, SMA Steckverbin-der
13	1	STK	CON SD-Slot X8	RS Components 685-0779	J	9-polig, Metall, extern
14	1	STK	CON SD-Slot X4	RS Components 237-647	J	9-polig, Kunststoff, intern
15	2	STK	CON RJ11 6/4 X1, X7	RS Components 423-4112	J	Durchsteckmontage
16	1	STK	BAT Lithium 9V -	RS Components 720-9083	J	800mAh
17	2	STK	KAB RJ11 6/4 -	RS Components 303-1287	J	3m, weiß
18	6	STK	WID 100k R1, R7, R9, R10-R12	HTL-Salzburg -	J	SMD 0805
19	2	STK	WID 2M R2, R4	HTL-Salzburg -	J	SMD 0805
20	2	STK	WID 1M R3, R5	HTL-Salzburg -	J	SMD 0805
21	3	STK	WID 10k R6, R8, R13	HTL-Salzburg -	J	SMD 0805
22	2	STK	KERKO 100n C1, C3	HTL-Salzburg -	J	SMD 0805
23	2	STK	ELKO 2u2 C2, C4	RS Components 6843989	J	Tantal SMD 293D 16V

4.4.4 Materialkalkulation Backplatine

1	2	3	4	5	6	
Lfd. Nr.	Stück-zahl	Ein-heit	Benennung	Lieferant / Bestellnummer	Preis Einzel	Preis Gesamt
1	1	STK	IC GPS-1513, D1	RS Components / 704-3307	22,490	22,490
2	1	STK	OP MCP6241-E/P, N2	RS Components / 403-018	0,263	0,263
3	1	STK	VREG LF33CDT, N3	RS Components / 355-4106	1,480	1,480
4	1	STK	VREG LF50CV, N1	RS Components / 355-4257	1,610	1,610
5	2	STK	SENS MPS4115AP, B1-B2	RS Components / 717-6527	12,340	24,680
6	1	STK	SENS TMP36GT9Z, -	RS Components / 709-2784	1,600	1,600
7	1	STK	FET P FDV304, V1	RS Components / 354-4913	0,590	0,590
8	1	STK	FET N FDV303, V4	RS Components / 354-4890	0,540	0,540
9	1	STK	FET P FDN302, V5	RS Components / 671-0400	0,740	0,740
10	2	STK	ZDIO SB140, V2-V3	RS Components / 701-0199	0,065	0,130
11	1	STK	CON Batterieclip 9V, X5	RS Components / 489-021	2,310	2,310
12	1	STK	CON SMA Stecker, X3	RS Components / 616-3400	2,380	2,380
13	1	STK	CON SD-Slot, X8	RS Components / 685-0779	3,520	3,520
14	1	STK	CON SD-Slot, X4	RS Components / 237-647	5,840	5,840
15	2	STK	CON RJ11 6/4, X1, X7	RS Components / 423-4112	1,190	2,380
16	1	STK	BAT Lithium 9V, -	RS Components / 720-9083	10,080	10,080
17	2	STK	KAB RJ11 6/4, -	RS Components / 303-1287	1,940	3,880
18	6	STK	WID 100k, R1, R7, R9, R10-R12	HTL-Salzburg / -	0,030	0,180
19	2	STK	WID 2M, R2, R4	HTL-Salzburg / -	0,030	0,060
20	2	STK	WID 1M, R3, R5	HTL-Salzburg / -	0,030	0,060
21	3	STK	WID 10k, R6, R8, R13	HTL-Salzburg / -	0,030	0,090
22	2	STK	KERKO 100n, C1, C3	HTL-Salzburg / -		
23	2	STK	ELKO 2u2, C2, C4	RS Components / 6843989	0,239	0,478

Übertrag:	
Materialpreis	**€ 85,381**
Materialgemeinkosten (10%)	**€ 8,538**
Materialverkaufspreis	**€ 93,919**

Zusätzlich anfallende Kosten:

- Platine (ca. 30€)
- Gehäuse

4.5 Testpläne, Testfälle, Testergebnisse

4.5.1 Geschwindigkeit TAS

Das Display gibt den True Airspeed in Knoten oder Kilometer pro Stunde aus. Um dies testen zu können wird die Messeinheit im Auto provisorisch eingebaut.
Der True Airspeed wird nun mit der Geschwindigkeit des Autos verglichen und zusätzlich werden die Daten auf der SD-Karte abgespeichert.
Ein genaues Ergebnis wird aber vermutlich nur erzielt, wenn man den Test im Flugzeug durchführt. Durch Gegenwind, Seitenwind und Rückenwind sind TAS und GS nicht gleich. Bei Rückenwind ist TAS niedriger als GS, bei Gegenwind erwartungsgemäß umgekehrt. Bei Seitenwind kann es sein, dass in eine Richtung beide Geschwindigkeiten ident sind, aber in die andere Richtung TAS höher bzw. niedriger ist als GS. Auch Luftverwirbelungen haben einen Einfluss auf das Ergebnis.
Nachfolgende Diagramme zeigen die ersten Ergebnisse der Geschwindigkeitsermittlung.

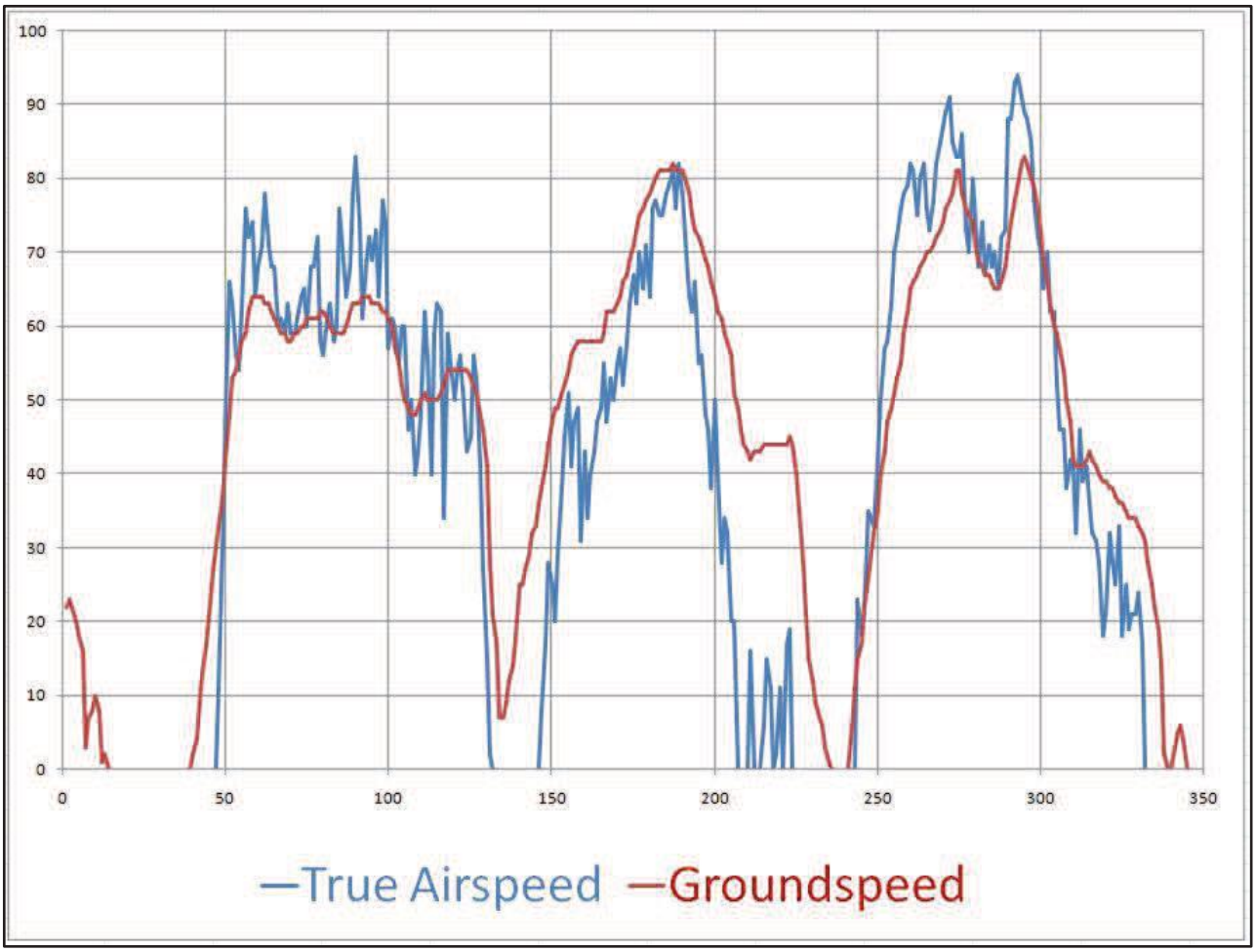


Abb. 39: Diagramm Vergleich TAS mit GS (Excel)

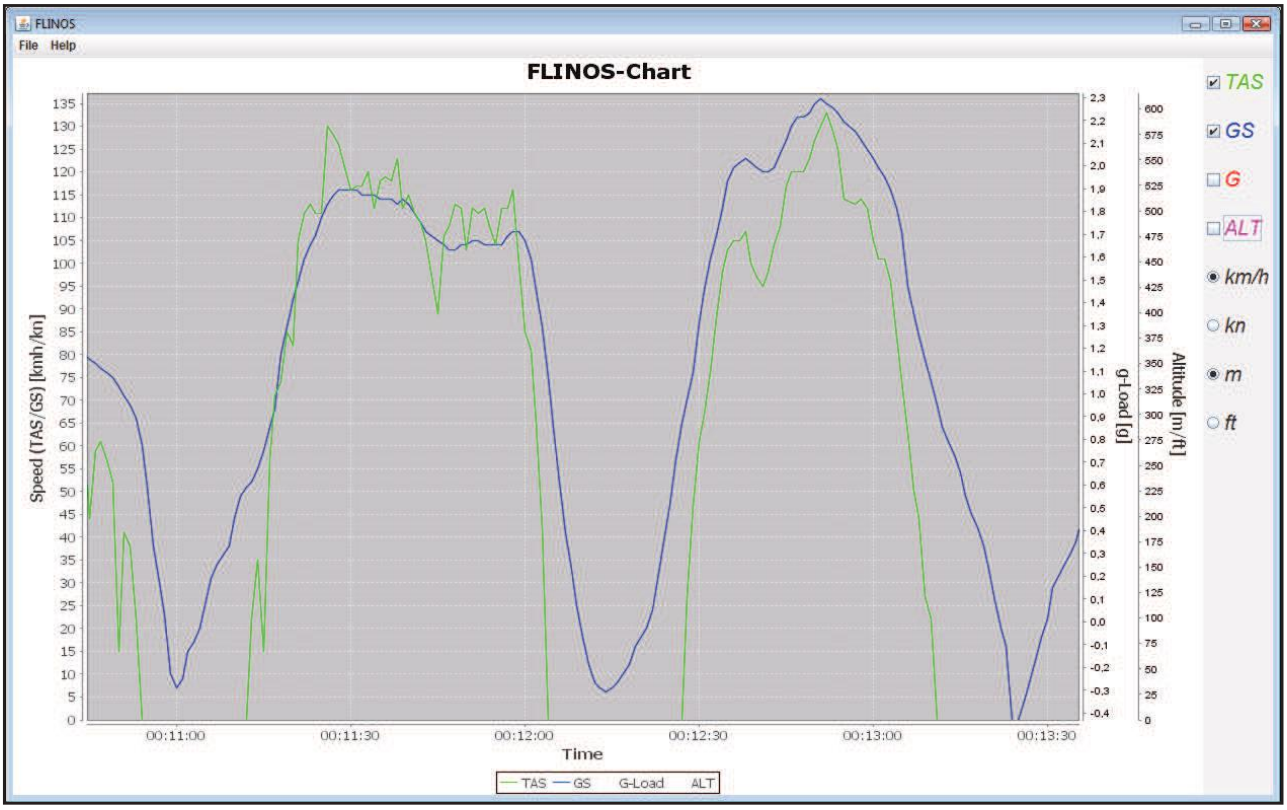


Abb. 40: Diagramm Vergleich TAS mit GS (PC-Software)

4.5.2 GPS Daten

Um die Korrektheit der GPS Daten zu gewährleisten, müssen einige Tests durchgeführt werden. Die Messeinheit samt GPS-Antenne wird im Auto montiert. Das Display gibt die aktuellen Werte (Groundspeed, Longitude, Latitude, Flughöhe) aus.
Longitude und Latitude werden auf der SD-Karte in einer XML-Datei gespeichert. Anschließend wird die Datei in Google-Maps importiert und die aufgezeichnete Route erscheint.
Der Groundspeed wird mit der Tachonadel des Autos verglichen.
Um die Flughöhe testen zu können, kann eine GPX-Datei auf der SD-Karte erstellt werden. In dieser Datei sind sowohl Longitude, Latitude, als auch die Flughöhe (Altitude) gespeichert. Diese Datei wird in Google-Earth importiert und zeigt die Route an. Der Verlauf müsste dann genau auf dem Boden sein (Test mit dem Auto). Will man jedoch ein genaueres Ergebnis der Flughöhe bekommen, muss man die Messeinheit im Flugzeug anbringen und den Test noch ein Mal durchführen.

Abb. 41: Gefahren Strecke mit dem Auto (Google Maps)

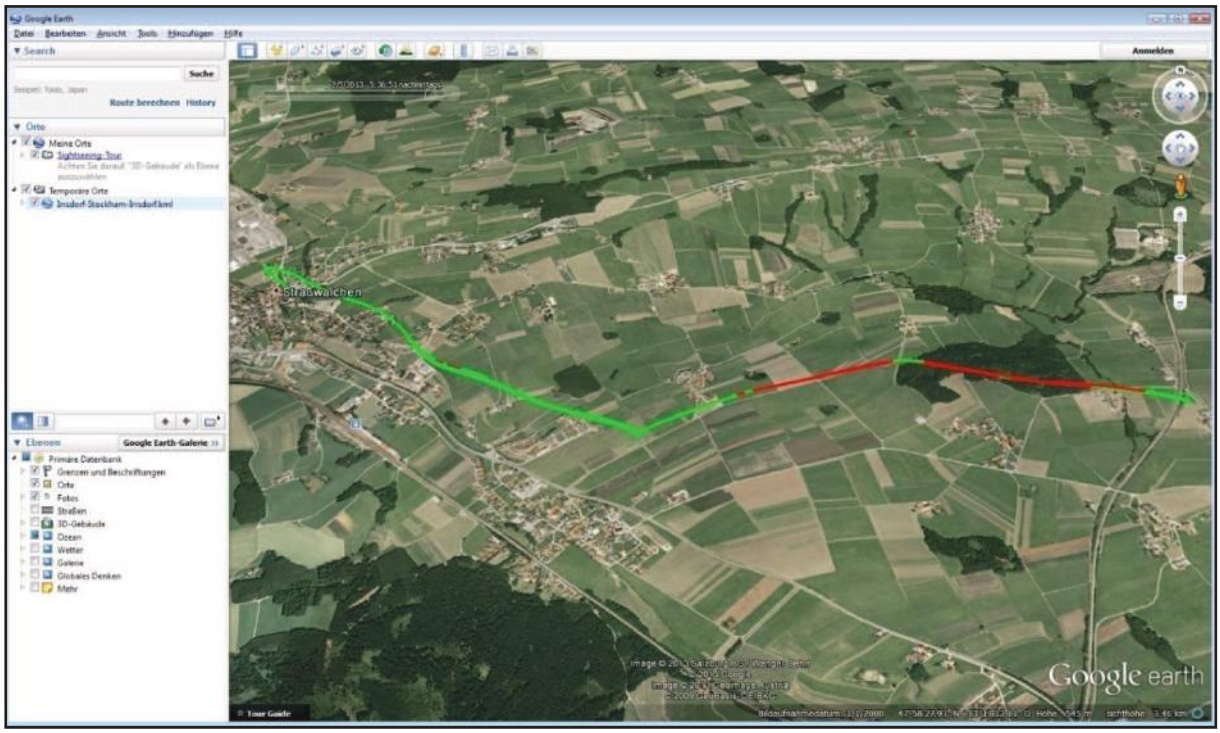


Abb. 42: Gefahrene Strecke mit dem Auto 2 (Google Earth)

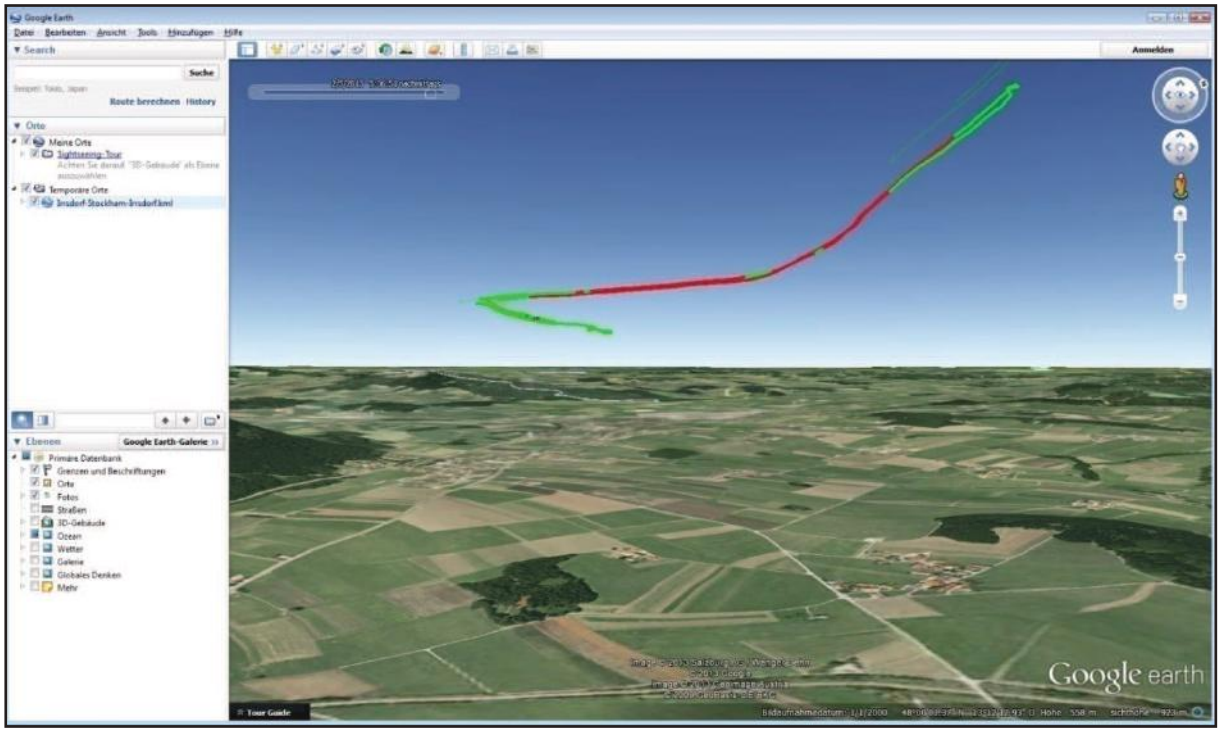


Abb. 43: Gefahrene Strecke mit dem Auto 3 (Google Earth)

Abbildung 42 und 43 zeigen die Route in Google Earth. Der Verlauf ist in rot und grün dargestellt, wobei die rote Farbe die Überlastungen (in diesem Fall eine Geschwindigkeitsüberschreitung mit dem Auto) signalisiert. Bei der zweiten Grafik wurde ein Höhenoffset eingebaut, damit der 3D-Effekt erkennbar ist (Verlauf sonst auf der Straße).

4.5.3 g-Belastung

Wenn sich die Beschleunigung in Z-Richtung ändert, dann muss das Display es anzeigen. Die Platine wird so ausgerichtet, wie sie auch im Flugzeug eingebaut wird. Bei dieser Ausrichtung muss +1.0g angezeigt werden. Wendet man nun die Platine (kopfüber), so gibt das Display den Wert -1.0g aus. Weiters kann man die Platine ruckartig schütteln, wobei das Display darauf reagieren und die Datenänderungen ausgeben muss.

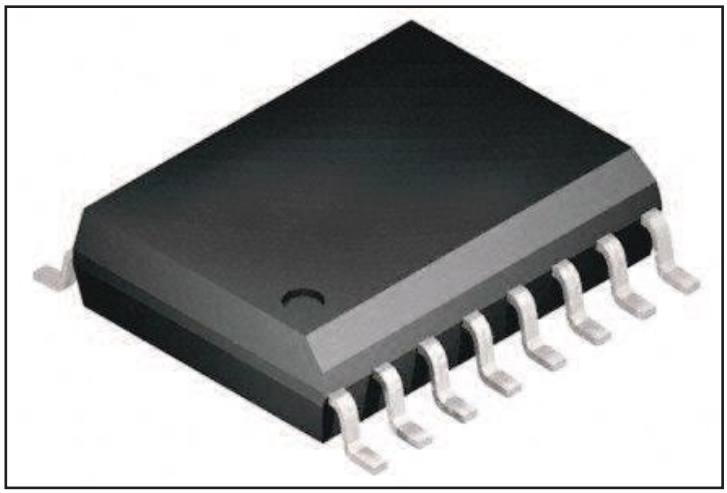


Abb. 44: G-Sensor MMA2244KEG[15]

4.5.4 Datenspeicherung auf der SD-Karte

Um die ordnungsgemäße Speicherung auf der SD-Karte überprüfen zu können, wird eine Textdatei mit vorgegebenem Inhalt erstellt. Anschließend wird die Karte beim Computer eingesteckt und die erstellte Datei auf Korrektheit überprüft. Nachfolgende Abbildungen zeigen die beiden SD-Slots, die wir bei unserem Projekt verwendet haben.

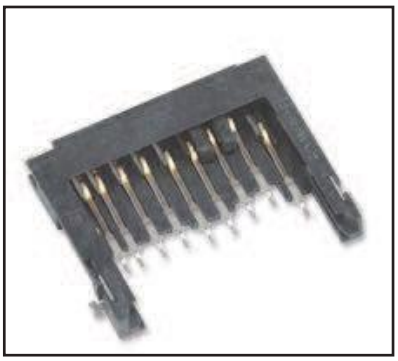


Abb. 45: SD-Slot intern[16]

Abb. 46: SD-Slot extern[17]

[15] http://img-europe.electrocomponents.com/largeimages/SOICW-16.jpg (15.04.2013)
[16] http://de.farnell.com/productimages/farnell/standard/42266788.jpg (15.04.2013)
[17] http://media.digikey.com/photos/Hirose%20Elect%20Photos/DM1AA-SF-PEJ(21).jpg (15.04.2013)

5 Zusammenfassung

Die private Fliegerei, insbesondere der Einsatz von Ultraleichtflugzeugen, gewinnt immer mehr an Bedeutung im Bereich der Luftfahrt. Um die Sicherheit für die Piloten und Fluggäste zu erhöhen, haben wir dieses Produkt entwickelt.

Meilensteine und Ziele wurden erreicht. Darüber hinaus ist auch ein PC-Programm entwickelt worden, welches ursprünglich den Kann- bzw. Wunschkriterien zugeordnet wurde. Mit diesem ist eine genaue Datenauswertung möglich.
Bisher sind Tests nur mit dem Auto durchgeführt worden. Akzeptable Ergebnisse wurden dabei erzielt. Um die Genauigkeit des True Airspeeds ordnungsgemäß überprüfen zu können, sind Messungen im Flugzeug geplant. Alle weiteren Daten wurden auf Richtigkeit erfolgreich überprüft.

Die Helligkeit und der Kontrast des Displays konnten ebenfalls noch nicht im Flugzeug getestet werden. Eine Aussage über die Lesbarkeit der Flugdaten bei Sonneneinstrahlung im Cockpit kann noch nicht gemacht werden.

Das Modul braucht zurzeit im Sleep-Modus (ausgeschalteter Zustand) noch einen Strom von etwa 500µA. Eine Idee, mit der dieses Problem gelöst werden kann, ist bereits vorhanden, aber noch nicht umgesetzt worden. Der Ruhestrom würde sich auf etwa 600nA reduzieren.

Weiters stellt die Backup-Batterie noch ein kleines Problem dar. Jeder 9V-Block (verschiedene Hersteller) weist eine unterschiedliche Entladekurve auf. Daher ist es nicht so einfach, dem Piloten mitzuteilen, wann die Batterie fast leer ist. Vorübergehend wurde das Problem gelöst, doch eine Überarbeitung wird noch notwendig sein.

Für die Zukunft haben wir vor, das Produkt zu erweitern und verbessern. Nach einer bestimmten, erfolgreich abgeschlossenen Testphase, wird das Produkt für private Piloten erhältlich sein.

Besonders bedanken möchten wir uns bei unserem Projektbetreuer Prof. Dipl.-Ing. Ing. Karl Heinz Steiner und bei unserem Projektpartner Ing. Johann Schwöller (Firma ScaleWings).
Weiters danken wir allen Lehrkräften und Personen, die uns bei unserem Projekt tatkräftig unterstützt haben.

6 Literaturverzeichnis

- **Website der Partnerfirma ScaleWings:**
  http://www.scalewings.com/, 31.03.2013

- **Datenblatt des Mikrocontrollers ATmega644PA:**
  http://docs-europe.electrocomponents.com/webdocs/0dc5/0900766b80dc5089.pdf, 10.04.2013

- **Datenblatt des GPS-Empfängers GPS-1513:**
  http://docs-europe.electrocomponents.com/webdocs/0df9/0900766b80df94d1.pdf, 10.04.2013

- **Datenblatt des Drucksensors MPX4115AP:**
  http://docs-europe.electrocomponents.com/webdocs/0ef4/0900766b80ef40fa.pdf, 10.04.2013

- **Artikel zur Berechnung des Differenzverstärkers, Elektronik Kompendium:**
  http://www.elektronik-kompendium.de/sites/slt/0210153.htm, 10.04.2013

- **Artikel zur Geschwindigkeitsberechnung mittels Prandtlsonde:**
  http://www.electro-mation.de/productattachments/index/download?id=47, 10.04.2013

7 Abbildungsverzeichnis

8 Anhang

8.1 Diplomarbeits-Antrag

8.2 Pflichtenheft

8.3 Projekttagebuch

8.4 Fertigungsunterlagen

8.5 Vertiefende Grundlagenarbeiten

htl bildung mit zukunft	HÖHERE TECHNISCHE BUNDES-LEHR- UND VERSUCHSANSTALT SALZBURG
	Abteilung: Elektronik
	Ausbildungsschwerpunkt: Technische Informatik

Anhang 8.1:

Diplomarbeits-Antrag

htl bildung mit zukunft	HÖHERE TECHNISCHE BUNDES-LEHR- UND VERSUCHSANSTALT SALZBURG
	Abteilung: Elektronik Ausbildungsschwerpunkt: Technische Informatik

DIPLOMARBEIT

5AHELI – Reife- und Diplomprüfung 2012/13

Thema	Entwicklung einer Messeinheit zur Flugdatenerfassung von Ultraleichtflugzeugen
Aufgabenstellung (Kurzfassung)	Die Messeinheit muss True Airspeed (TAS), Groundspeed, g-Belastung, GPS-Koordinaten und Flughöhe von Ultraleichtflugzeugen aufzeichnen. Die gemessenen Daten werden auf einen permanent vorhandenen Speicher abgelegt und auf Anforderung auf einen Wechseldatenträger übertragen. Die Momentanwerte sind im Cockpit anzuzeigen. Eine Backupversorgung ermöglicht bei Stromausfall vorrübergehend die ordnungsgemäße Funktion.

Kandidaten	**Betreuer**
Johannes Schwöller	Prof. Dipl.-Ing. Ing. Karl Heinz Steiner
Sebastian Modl	

Externe Kooperationspartner
Firma / Institution: ScaleWings Modelltechnik GmbH
Betreuer / Kontaktperson: Ing. Johann Schwöller
Schriftliche Kooperationsvereinbarung: ist in Ausarbeitung
Budget: Euro 300
Bedeckung durch: ScaleWings Modelltechnik GmbH
Geplante Verwertung der Ergebnisse: Nach einer Testphase wird die Messeinheit durch die Firma ScaleWings vermarktet.

htl bildung mit zukunft	HÖHERE TECHNISCHE BUNDES-LEHR- UND VERSUCHSANSTALT SALZBURG
	Abteilung: Elektronik Ausbildungsschwerpunkt: Technische Informatik

Erklärung

Die unterfertigten Kandidaten / Kandidatinnen haben gemäß § 34 (3) SchUG in Verbindung mit § 22 (1) Zi. 3 lit. b der Verordnung über die abschließenden Prüfungen in den berufsbildenden mittleren und höheren Schulen, BGBl. II Nr. 70 vom 24.02.2000 (Prüfungsordnung BMHS), die Ausarbeitung einer Diplomarbeit mit der umseitig angeführten Aufgabenstellung gewählt.
Die Kandidaten / Kandidatinnen nehmen zur Kenntnis, dass die Diplomarbeit in eigenständiger Weise und außerhalb des Unterrichtes zu bearbeiten und anzufertigen ist, wobei Ergebnisse des Unterrichtes mit einbezogen werden können.
Die Abgabe der vollständigen Diplomarbeit hat bis spätestens

10.05.2013, 17:00 Uhr

beim zuständigen Betreuer zu erfolgen.

Die Kandidaten / Kandidatinnen nehmen weiterhin zur Kenntnis, dass gemäß § 9 (6) der Prüfungsordnung BMHS nur der Schulleiter bis spätestens Ende des vorletzten Semesters den Abbruch einer Diplomarbeit anordnen kann, wenn diese aus nicht beim Prüfungskandidaten (bei den Prüfungskandidaten) gelegenen Gründen nicht fertiggestellt werden kann.

Kandidaten	Unterschrift
Johannes Schwöller	
Sebastian Modl	

Prof. Dipl.-Ing. Ing. Karl Heinz Steiner
Betreuer

AV Dipl.-Ing. Dr. techn. Andreas Magauer
Abteilungsvorstand

DIR Dipl.-Ing. Dr. techn. Herbert Kittl
Direktor

Genehmigung:

Salzburg, am 16/10/2012

LSI HR Dipl.-Ing. Robert Vasak
Landesschulinspektor

Ergänzende Ausführungen

1. Diplomarbeitstitel:

Entwicklung einer Messeinheit zur Flugdatenerfassung von Ultraleichtflugzeugen

2. Arbeitstitel:

FLINOS
Flight-Info-System

3. Firmenvorstellung:

ScaleWings Modelltechnik GmbH

ScaleWings

Sonnenweg 5
A-5204 Straßwalchen
Entwicklung von Scale-Großflugmodellen und manntragende Ultraleichtflugzeuge in Voll-GFK/CFK.
Ing. Johann Schwöller

4. Projektorganisation:

Projektleiter:
- Johannes Schwöller

Projektmitarbeiter:
- Sebastian Modl

Projektbetreuer:
- Prof. Dipl.-Ing. Ing. Karl Heinz Steiner

Externe Kontaktperson:
- Ing. Johann Schwöller

5. Ist-Darstellung:

Vereine und Vercharterer verfügen kaum über eine Aufzeichnung bzw. Kontrolle von überschrittenen Belastungen und geflogenen Flugstrecken, der von ihnen vercharterten Flugzeugen.
Es liegt keine Backup-Information von Geschwindigkeit und Flughöhe bei Ausfall der Boardinstrumente vor.

6. Aufgabenstellung:

Die Messeinheit muss True Airspeed (TAS), Groundspeed, g-Belastung, GPS-Koordinaten und Flughöhe von Ultraleichtflugzeugen aufzeichnen.
Die gemessenen Daten werden auf einen permanent vorhandenen Speicher abgelegt und auf Anforderung auf einen Wechseldatenträger übertragen.
Die Momentanwerte sind im Cockpit anzuzeigen.
Eine Backupversorgung ermöglicht bei Stromausfall vorrübergehend die ordnungsgemäße Funktion.

7. Zielsetzung (Ergebnisse):

Die Zielsetzung ist die Entwicklung eines Moduls zur Aufzeichnung und Anzeige von Flugzeugdaten. Es ist hauptsächlich für die Verwendung in Ultraleichtflugzeugen bestimmt.
Mit Hilfe der aufgezeichneten Daten können Piloten ausfindig gemacht werden, die das Flugzeug überlastet haben. Bei Ausfall der Boardinstrumente liefert das Modul weiterhin notwendige Daten.
Nach einer Testphase wird die Messeinheit durch die Firma ScaleWings vermarktet.

8. Aufgabenbeschreibung:

Johannes Schwöller

- Entwicklungsarbeit
  - Hardware Entwicklung
    - Auswahl der Bauteile
    - Platinenentwicklung
    - Prototypenaufbau
    - Schaltung für die Ladeelektronik des Akkus
  - Software Entwicklung
    - Ansteuerung des Displays
    - Ansteuerung und Auswertung des Beschleunigungssensors
- Vertiefende Grundlagenarbeit
  - Messtechnische Erfassung der analogen Drucksensordaten und Entscheidung, ob diese Signale vor der Übertragung digitalisiert werden müssen

Sebastian Modl

- Entwicklungsarbeit
  - Hardware Entwicklung
    - Auswahl der Bauteile
    - Prototypenaufbau
  - Software Entwicklung
    - Ansteuerung und Auswertung der Drucksensoren
    - Ansteuerung des GPS-Moduls
    - Ansteuerung der SD-Karte
- Vertiefende Grundlagenarbeit
  - Aufzeichnung und Bewertung der Signalverläufe am vorliegenden I²C-Bus

9. Meilensteine:

Meilensteine	Datum
Abgabe des DA-Antrages	12.10.2012
1. Review	17.10.2012
2. Review	15.11.2012
1. Präsentation	19.12.2012
3. Review	16.01.2013
2. Präsentation	18, 19.01.2013
4. Review	16.02.2013
5. Review	21.03.2013
6. Review	11.04.2013
Abgabe der DA	10.05.2013
Schlusspräsentation	14.05.2013

10. Kostenschätzung:

Kostenstelle	Preis
Display	€ 30
GPS-Modul	€ 50
Drucksensoren	€ 100
Programmiergerät	€ 40
Weitere Bauteile	€ 80
Summe	€ 300

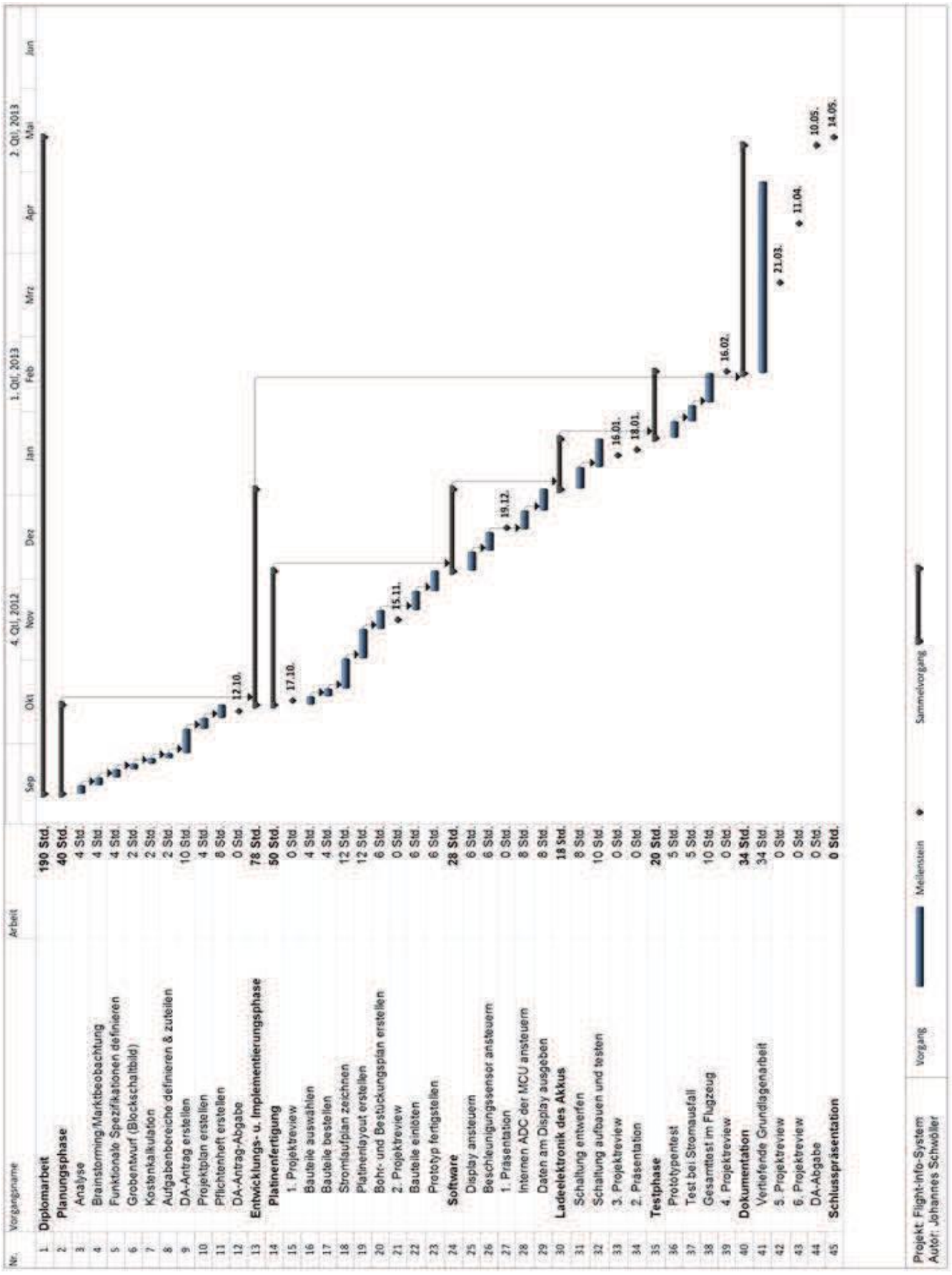
11. Abzugebende Unterlagen:

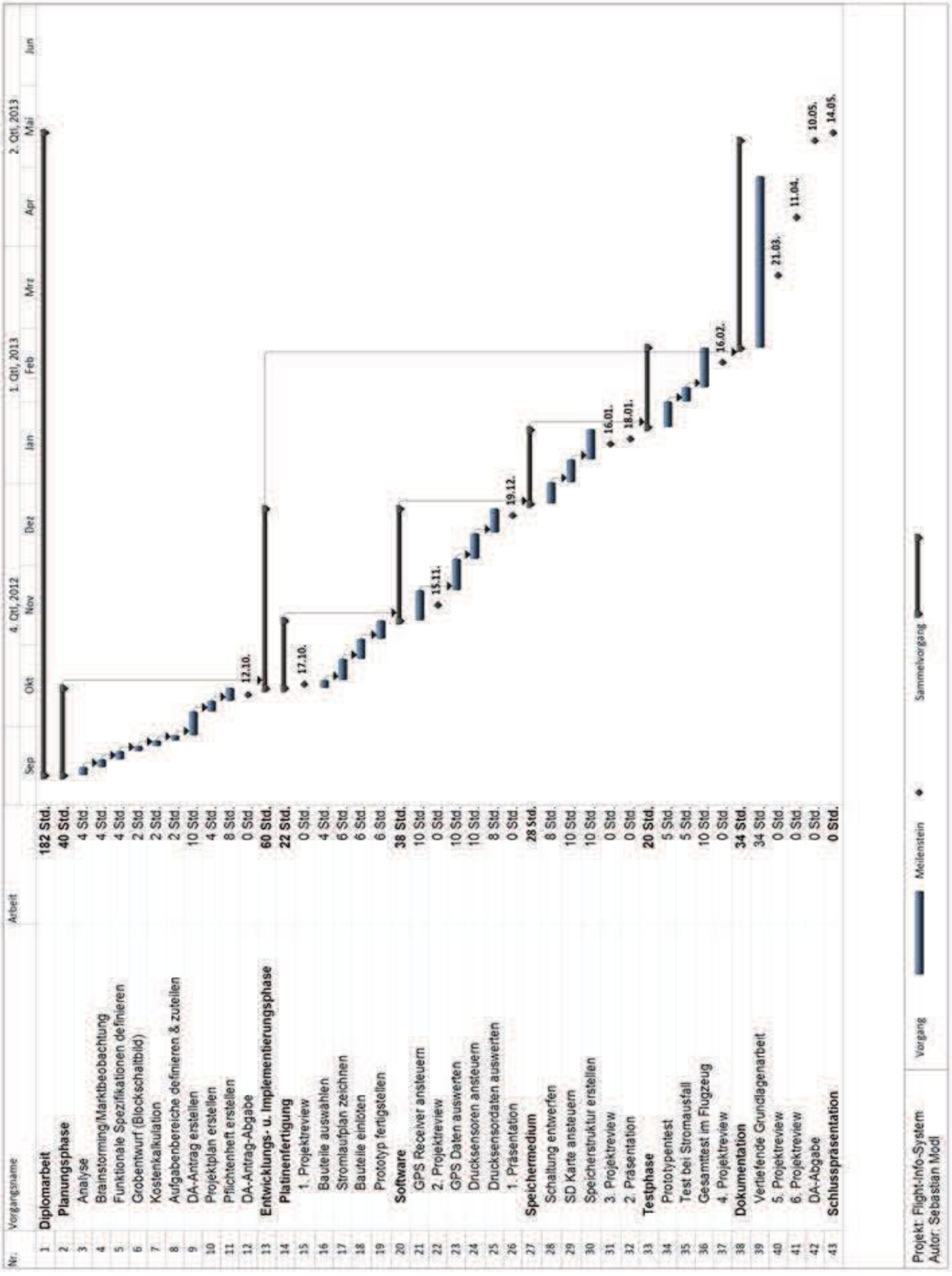
Gebundene Diplomarbeit mit System- und Benutzerdokumentation sowie der Grundlagenarbeit CD od. DVD mit allen Unterlagen und Inhalten in elektronischer Form einschließlich Quell- und Programmcode.

12. Hinweise:

Für die Erstellung der Diplomarbeit sind die FTKL- und Software-Dokumentations-Richtlinien der Elektronikabteilung der HTBLuVA-Salzburg einzuhalten.
Alle nicht selbst erarbeiteten Inhalte werden laut Zitierrichtlinien der HTBLuVA –Salzburg zitiert.

13. Ganttdiagramme:


Nr.
Vorgangsname
Arbeit
Sep
Okt
4. Qtl. 2012
Nov
Dez
Jan
1. Qtl. 2013
Feb
Mrz
Apr
2. Qtl. 2013
Mai
Jun
1 Diplomarbeit 190 Std.
2 Planungsphase 40 Std.
3 Analyse 4 Std.
4 Brainstorming/Marktbeobachtung 4 Std.
5 Funktionale Spezifikationen definieren 4 Std.
6 Grobentwurf (Blockschaltbild) 2 Std.
7 Kostenkalkulation 2 Std.
8 Aufgabenbereiche definieren & zuteilen 2 Std.
9 DA-Antrag erstellen 10 Std.
10 Projektplan erstellen 4 Std.
11 Pflichtenheft erstellen 8 Std.
12 DA-Antrag-Abgabe 0 Std.
13 Entwicklungs- u. Implementierungsphase 78 Std.
14 Platinenfertigung 50 Std.
15 1. Projektreview 0 Std.
16 Bauteile auswählen 4 Std.
17 Bauteile bestellen 4 Std.
18 Stromlaufplan zeichnen 12 Std.
19 Platinenlayout erstellen 12 Std.
20 Bohr- und Bestückungsplan erstellen 6 Std.
21 2. Projektreview 0 Std.
22 Bauteile einlöten 6 Std.
23 Prototyp fertigstellen 6 Std.
24 Software 28 Std.
25 Display ansteuern 6 Std.
26 Beschleunigungssensor ansteuern 6 Std.
27 1. Präsentation 0 Std.
28 Internen ADC der MCU ansteuern 8 Std.
29 Daten am Display ausgeben 8 Std.
30 Ladeelektronik des Akkus 18 Std.
31 Schaltung entwerfen 8 Std.
32 Schaltung aufbauen und testen 10 Std.
33 3. Projektreview 0 Std.
34 2. Präsentation 0 Std.
35 Testphase 20 Std.
36 Prototypentest 5 Std.
37 Test bei Stromausfall 5 Std.
38 Gesamttest im Flugzeug 10 Std.
39 4. Projektreview 0 Std.
40 Dokumentation 34 Std.
41 Verliefende Grundlagenarbeit 34 Std.
42 5. Projektreview 0 Std.
43 6. Projektreview 0 Std.
44 DA-Abgabe 0 Std.
45 Schlusspräsentation 0 Std.
19.12.
15.11.
17.10.
12.10.
16.01.
18.01.
16.02.
21.03.
11.04.
10.05.
14.05.
Projekt: Flight-Info-System
Autor: Johannes Schwöller
Vorgang
Meilenstein
Sammelvorgang


Nr.
Vorgangsname
Arbeit
4. Qtl, 2012
1. Qtl, 2013
2. Qtl, 2013
Sep
Okt
Nov
Dez
Jan
Feb
Mrz
Apr
Mai
Jun
1 Diplomarbeit 182 Std.
2 Planungsphase 40 Std.
3 Analyse 4 Std.
4 Brainstorming/Marktbeobachtung 4 Std.
5 Funktionale Spezifikationen definieren 4 Std.
6 Grobentwurf (Blockschaltbild) 2 Std.
7 Kostenkalkulation 2 Std.
8 Aufgabenbereiche definieren & zuteilen 2 Std.
9 DA-Antrag erstellen 10 Std.
10 Projektplan erstellen 4 Std.
11 Pflichtenheft erstellen 8 Std.
12 DA-Antrag-Abgabe 0 Std.
13 Entwicklungs- u. Implementierungsphase 60 Std.
14 Platinenfertigung 22 Std.
15 1. Projektreview 0 Std.
16 Bauteile auswählen 4 Std.
17 Stromlaufplan zeichnen 6 Std.
18 Bauteile einlöten 6 Std.
19 Prototyp fertigstellen 6 Std.
20 Software 38 Std.
21 GPS Receiver ansteuern 10 Std.
22 2. Projektreview 0 Std.
23 GPS Daten auswerten 10 Std.
24 Drucksensoren ansteuern 10 Std.
25 Drucksensordaten auswerten 8 Std.
26 1. Präsentation 0 Std.
27 Speichermedium 28 Std.
28 Schaltung entwerfen 8 Std.
29 SD Karte ansteuern 10 Std.
30 Speicherstruktur erstellen 10 Std.
31 3. Projektreview 0 Std.
32 2. Präsentation 0 Std.
33 Testphase 20 Std.
34 Prototypentest 5 Std.
35 Test bei Stromausfall 5 Std.
36 Gesamttest im Flugzeug 10 Std.
37 4. Projektreview 0 Std.
38 Dokumentation 34 Std.
39 Verliefende Grundlagenarbeit 34 Std.
40 5. Projektreview 0 Std.
41 6. Projektreview 0 Std.
42 DA-Abgabe 0 Std.
43 Schlusspräsentation 0 Std.
15.11.
16.01.
18.01.
16.02.
21.03.
11.04.
10.05.
14.05.
Projekt: Flight-Info-System
Autor: Sebastian Modl
Vorgang
Meilenstein
Sammelvorgang

HÖHERE TECHNISCHE BUNDES-LEHR- UND VERSUCHSANSTALT
SALZBURG

Abteilung: Elektronik

Ausbildungsschwerpunkt: Technische Informatik

Anhang 8.2:

Pflichtenheft

htl bildung mit zukunft	HÖHERE TECHNISCHE BUNDES-LEHR- UND VERSUCHSANSTALT SALZBURG
	Abteilung: Elektronik
	Ausbildungsschwerpunkt: Technische Informatik

PFLICHTENHEFT

Flight-Info-System

Ausgeführt im Schuljahr 2012/2013

Auftragnehmer:

Johannes Schwöller 5AHELI

Sebastian Modl 5AHELI

Auftraggeber/Ansprechpartner:

ScaleWings Modelltechnik GmbH

Ing. Johann Schwöller

Salzburg, am 07.05.2013

Inhaltsverzeichnis

1 Zielbestimmungen

1.1 Musskriterien

Die Messeinheit muss True Airspeed (TAS), Groundspeed (GS), g-Belastung, GPS-Koordinaten und Flughöhe von Ultraleichtflugzeugen aufzeichnen.
Die gemessenen Daten werden auf einen permanent vorhandenen Speicher abgelegt und auf Anforderung auf einen Wechseldatenträger übertragen.
Die Momentanwerte sind im Cockpit anzuzeigen.
Eine Backupversorgung ermöglicht bei Stromausfall vorrübergehend die ordnungsgemäße Funktion.

1.2 Wunschkriterien

Wenn möglich kann die geflogene Flugroute bei Google-Earth angezeigt werden. Weiters kann eine Überspannungsschutzschaltung dimensioniert und aufgebaut werden.

1.3 Abgrenzungskriterien

Die Computersoftware zur Datenauswertung muss nicht programmiert werden.

2 Produkteinsatz

2.1 Anwendungsbereiche

Das Modul dient zur Aufzeichnung bzw. Kontrolle von Flügen für Vereine, Vercharterer, Wettbewerbe und Hobby. Es liegt auch eine Backupinformation bei Stromausfall im Flugzeug für die Flughöhe und True Airspeed (TAS) vor.
Weiters zeigt die Messeinheit die Überlastungen im Cockpit an.

2.2 Zielgruppen

Die geplanten Zielgruppen sind Vereine, Vercharterer und Hobbypiloten. Es ist auch der Einsatz bei Flugwettbewerben zur Aufzeichnung bzw. Kontrolle möglich.

2.3 Betriebsbedingungen

+40°C/-10°C, +/- 20g, Flughöhe maximal 20000ft, TAS max. 500km/h, spritzsicher, GPS-Antenne außerhalb anbringen (wegen der Abschirmung des Flugzeuges), Temperatursensor außen anbringen

3 Produktumgebung

3.1 Software

AVR Studio 5.0, Microsoft Office 2007/10, Scrumy (www.scrumy.com), Dropbox 1.4.17, Java Eclipse Indigo

3.2 Hardware

Mikrocontroller, diverse Sensoren, Display, Leuchtdioden, Taster, Platine, Programmiergerät, Akku, diverse Buchsen

4 Produktfunktionen

/F0010/Ermittlung von True Airspeed (TAS) mittels Drucksensoren

Die Geschwindigkeit TAS wird mit 2 Drucksensoren ermittelt. Die beiden Sensoren stehen im Winkel von 90° zueinander. Einer der beiden Drucksensoren misst den Staudruck, der andere den statischen Druck. Analoge Werte werden zurückgegeben, vom Mikrocontroller digitalisiert, ausgewertet und die Geschwindigkeit daraus berechnet.

/F0020/Daten des GPS-Moduls:

Das GPS-Modul liefert über USART viele GPS bezogene Daten.
Folgende Daten werden herausgefiltert:

- Groundspeed (GS)
- Koordinaten (Lon., Lat.)
- Flughöhe

/F0030/Ermittlung der G-Belastung:

Die Beschleunigung wird mit einem G-Sensor gemessen. Dieser gibt eine analoge Spannung zurück, die proportional zur Beschleunigung in Z-Richtung ist (Einheit: g). Die Spannung wird mit dem internen AD-Wandler des Mikrocontrollers digitalisiert und ausgewertet.

/F0040/Speicherung der Daten auf der SD-Karte

Die aufgenommenen Daten werden auf der internen SD-Karte gespeichert. Die SD-Karte wird via SPI angesprochen. Um die Funktionalität zu gewährleisten, soll die Speicherkarte FAT32 formatiert sein.
Weiters kann eine externe SD-Karte in einen angebrachten Slot eingesteckt werden. Mit einer Tastenkombination wird eine Kopie der internen Karte auf der externe Karte erstellt.

Folgende Daten werden bei jedem Flug in eine Datei gespeichert.

- True Airspeed
- Groundspeed
- g-Belastung
- GPS-Koordinaten
- Flughöhe
- Datum/Uhrzeit

5 Produktdaten

/D0050/Anzeige der Flugdaten im Cockpit:

Folgende Daten werden am Display angezeigt:

- True Airspeed (in km/h oder kn)
- Groundspeed (in km/h oder kn)
- g-Belastung (in g)
- Flughöhe (in m oder ft)

Mittels RGB-Leds werden die Überlastungen signalisiert:

- Rot blinkend: Überlastung
- Gelb leuchtend: nahe am Grenzwert
- Grün leuchtend: alles in Ordnung

/D0060/Speicherung der Flugdaten:

Speicherung folgender Daten eines Fluges:

- True Airspeed
- Groundspeed
- g-Belastung
- GPS-Koordinaten
- Flughöhe
- Datum/Uhrzeit

Wird eine bestimmte Geschwindigkeit ermittelt und überschritten, so wird ein neuer Flug aufgezeichnet und die gemessenen Daten gespeichert. Erst wenn der interne Speicher voll ist werden die Flüge von vorne wieder überschrieben.
Gespeichert werden die Daten intern auf einer SD-Karte. Bei Bedarf kann eine weitere SD-Karte eingesteckt und die Daten auf diese kopiert werden.

6 Produktleistungen

/L0070/Genauigkeit:

Die Beschleunigung wird mit einer Nachkommastelle angezeigt und gespeichert, alle anderen Daten liegen ganzzahlig vor.

/L0080/Anzeige- und Speicherintervall:

Die Daten werden jede Sekunde am Display aktualisiert und angezeigt. Die Speicherung der Daten erfolgt im Sekundenabstand.

7 Benutzungsoberfläche

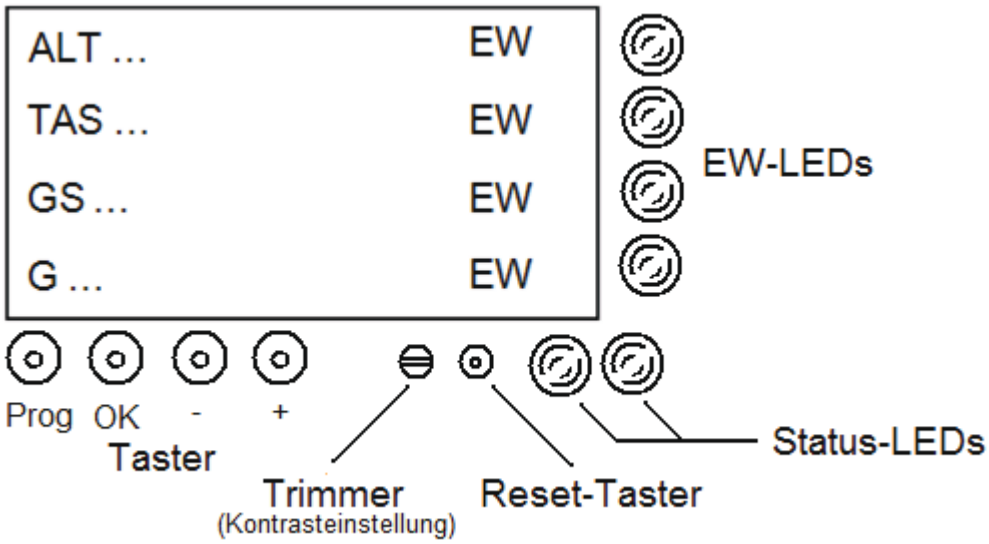
7.1 Anzeige im Cockpit

Die aktuellen Daten werden am Display angezeigt. Wird ein Extremwert überschritten, so signalisiert dies die entsprechende Leuchtdiode.

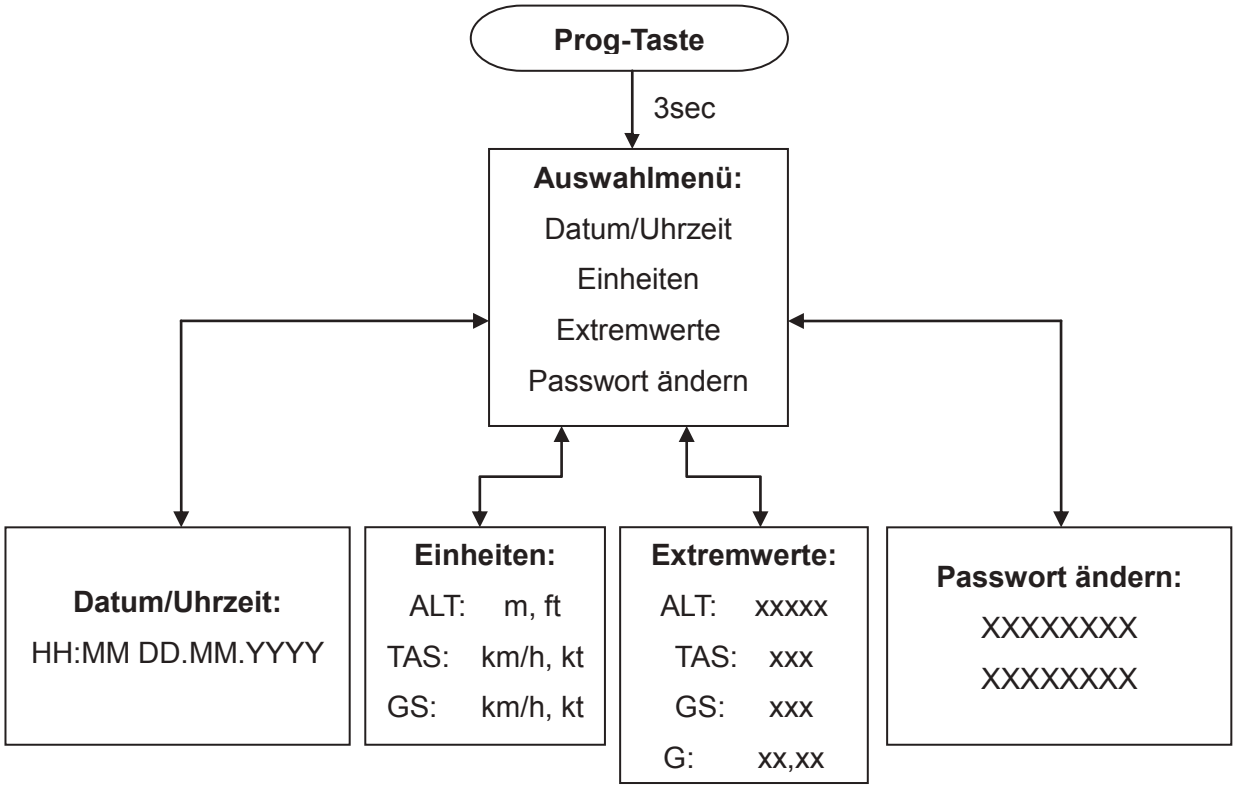
Um in den Programmiermodus einsteigen zu können, muss zuerst die Taste „Prog“ 3 Sekunden lang gedrückt werden. Danach wird ein Passwort (8-stellige Tastenkombination) abgefragt. Im Menü kann man folgende Punkte auswählen:

- Einheiten
- Extremwerte
- Datum/Uhrzeit
- Passwort ändern

Um den Programmiermodus wieder zu verlassen, muss man die Prog-Taste wieder betätigen.



7.2 Einstellungen im Programmiermodus



- Ein-/Ausstieg in/aus dem Programmiermodus mit Prog-Taste (mind. 3 Sekunden)
- Rückkehr zum Auswahlmenü mit Prog-Taste
- Veränderung der Werte (Datum, Einheit, Extremwerte) mit „+“ bzw. „-„ Taste
- 8-stelliges Passwort mit Tastenkombination, neues Passwort 2 mal eingeben
- Bestätigung der einzelnen Einstellungen mit OK-Taste → nächste Einstellung
  (z.B.: Einstellung Extremwert TAS mit +/-, danach OK-Taste, daraus folgt Einstellung GS), Step-to-Step-Modus

8 Qualitätszielbestimmungen

Eigenschaft	Sehr wichtig	Wichtig	Weniger wichtig	Unwichtig
Robustheit	X			
Zuverlässigkeit	X			
Korrektheit	X			
Benutzerfreundlichkeit		X		
Preis		X		

9 Globale Testszenarien und Testfälle

/T0010/Geschwindigkeit TAS:

Das Display gibt den True Airspeed in Knoten oder Kilometer pro Stunde aus. Um dies testen zu können, wird die Messeinheit im Auto provisorisch eingebaut und die Daten auf der SD-Karte aufgezeichnet.

Der True Airspeed wird nun mit der Geschwindigkeit des Flugzeuges oder des Autos verglichen. Ein genaues Ergebnis wird aber vermutlich nur erzielt, wenn man den Test im Flugzeug durchführt.

/T0020/GPS Daten:

Um die Korrektheit der GPS Daten zu gewährleisten, müssen einige Tests durchgeführt werden. Die Messeinheit samt GPS-Antenne wird im Auto montiert. Das Display gibt die aktuellen Werte (Groundspeed, Longitude, Latitude, Flughöhe) aus.

Longitude und Latitude werden auf der SD-Karte in einer XML-Datei gespeichert. Anschließend wird die Datei in Google-Maps importiert und die aufgezeichnete Route erscheint.

Der Groundspeed wird mit der Tachonadel des Autos verglichen.

Um die Flughöhe testen zu können, kann eine GPX- Datei auf der SD-Karte erstellt werden. In dieser Datei sind sowohl Longitude, Latitude, als auch die Flughöhe (Altitude) gespeichert. Diese Datei wird in Google-Earth importiert und zeigt die Route an. Der Verlauf müsste dann genau auf dem Boden sein (Test mit dem Auto). Will man jedoch ein genaueres Ergebnis der Flughöhe bekommen, muss man die Messeinheit im Flugzeug anbringen und den Test noch ein Mal durchführen.

/T0030/G-Belastung:

Wenn sich die Beschleunigung in Z-Richtung ändert, dann muss das Display es anzeigen. Die Platine wird so ausgerichtet, wie sie auch im Flugzeug eingebaut wird. Bei dieser Ausrichtung muss +1.0g angezeigt werden. Wendet man nun die Platine (kopfüber), so gibt das Display den Wert -1.0g aus. Weiters kann man die Platine ruckartig schütteln, wobei das Display darauf reagieren und die Datenänderung ausgeben muss.

/T0040/Datenspeicherung auf der SD-Karte:

Um die ordnungsgemäße Speicherung auf der SD-Karte überprüfen zu können, wird eine Textdatei mit vorgegebenem Inhalt erstellt. Anschließend wird die Karte beim Computer eingesteckt und die erstellte Datei auf Korrektheit überprüft.

10 Entwicklungsumgebung

10.1 Software

AVR Studio V5.0

Java Eclipse Indigo

10.2 Hardware

AVR Programmer und Emulator

11 Glossar

Backupversorgung Bei Stromausfall liefert eine Batterie die nötige Energie. Die Messeinheit arbeitet ordnungsgemäß weiter.

FAT32 englisch „File Allocation Table", ist ein Dateisystem von Microsoft.

g-Belastung ist die Kraft (Beschleunigung), die auf einen Körper ausgesetzt wird. 1g = Erdbeschleunigung = ca. 9,81m/s²

GPS englisch „Global Positioning System", ist ein satellitenunterstütztes Modul zur Positionsbestimmung. (Longitude, Latitude)

Groundspeed ist die Geschwindigkeit relativ zum Boden (Fahrt über Grund). Die Abkürzung ist GS.

True Airspeed ist die Geschwindigkeit relativ zur Luft (Wahre Fluggeschwindigkeit). Die Abkürzung ist TAS.

Ultraleichtflugzeug sind Flugzeuge, die klein und sehr leicht sind. Sie sind mit einem Motor angetrieben und in Europa für maximal 2 Personen zugelassen.

Vercharterer sind Personen, die ihr/e Flugzeug/e kostenpflichtig verleihen („vermieten).

htl bildung mit zukunft	HÖHERE TECHNISCHE BUNDES-LEHR- UND VERSUCHSANSTALT SALZBURG
	Abteilung: Elektronik
	Ausbildungsschwerpunkt: Technische Informatik

Anhang 8.3:

Projekttagebuch

htl bildung mit zukunft	HÖHERE TECHNISCHE BUNDES-LEHR- UND VERSUCHSANSTALT SALZBURG
	Abteilung: Elektronik
	Ausbildungsschwerpunkt: Technische Informatik

PROJEKTTAGEBUCH

Flight-Info-System

Ausgeführt im Schuljahr 2012/13

Auftragnehmer:

Johannes Schwöller 5AHELI

Sebastian Modl 5AHELI

Auftraggeber/Ansprechpartner:

ScaleWings Modelltechnik GmbH

Ing. Johann Schwöller

Salzburg, am 07.05.2013

Inhaltsverzeichnis

1 Projekttagebuch

1.1 Mi. 12.09.2012 / Do. 13.09.2012

1.1.1 Johannes Schwöller

- Arbeitsplatz einrichten
- Ideen sammeln
- Im Internet nach Bauteilen suchen
- Mit DA-Antrag beginnen

1.1.2 Sebastian Modl

- Arbeitsplatz einrichten
- Ideen sammeln
- Im Internet nach Bauteilen suchen

1.2 Mi. 19.09.2012 / Do. 20.09.2012

1.2.1 Johannes Schwöller

- Arbeitsplatz einrichten
- Mit Betreuer sprechen
- Im Internet nach Bauteile suchen
- Stromlaufplan zeichnen
- An DA-Antrag weiterarbeiten

1.2.2 Sebastian Modl

- Arbeitsplatz einrichten
- Mit Betreuer sprechen
- Im Internet nach Bauteile suchen
- Internen ADC vom ATmega644PA testen
- UART und SPI programmieren

1.3 Mi. 26.09.2012 / Do. 27.09.2012

1.3.1 Johannes Schwöller

- Abwesend am 1. Tag
- An DA-Antrag weiterarbeiten
- Stromlaufplan weiterzeichnen

1.3.2 Sebastian Modl

- Abwesend an beiden Tagen

1.4 Mi. 03.10.2012 / Do. 04.10.2012

1.4.1 Johannes Schwöller

- Stromlaufplan weiterzeichnen
- Stromlaufplan fertigstellen
- Leiterplattendesign Anordnung
- Platinenlayout
- An DA-Antrag weiterarbeiten

1.4.2 Sebastian Modl

- An DA-Antrag weiterarbeiten
- Display Drähte anlöten
- Display ansteuern (8-bit)
- Display ansteuern (4-bit)

1.5 Mi. 10.10.2012 / Do. 11.10.2012

1.5.1 Johannes Schwöller

- DA- Antrag fertigstellen
- Tasterprogrammierung
- Auswahl der Drucksensoren
- Auswahl des Temperatursensors
- Über Geschwindigkeitsberechnung informieren
- ADC testen

1.5.2 Sebastian Modl

- DA- Antrag fertigstellen
- Tasterprogrammierung
- Auswahl der Drucksensoren
- Auswahl des Temperatursensors
- Über Geschwindigkeitsberechnung informieren
- ADC testen

1.6 Mi. 17.10.2012 / Do. 18.10.2012

1.6.1 Johannes Schwöller

- Prototypenplatine SMD-Bauteile einlöten
- Display auf der Prototypenplatine ansteuern
- Fehlersuche Display

1.6.2 Sebastian Modl

- ADC-Genauigkeit testen
- Display auf der Prototypenplatine ansteuern
- Fehlersuche Display

1.7 Mi. 24.10.2012 / Do. 25.10.2012

1.7.1 Johannes Schwöller

- IO-Expander und RGB-Leds einlöten
- IO-Expander und RGB-Leds programmieren
- Fehlersuche GPS

1.7.2 Sebastian Modl

- Taster umlöten
- IO-Expander und RGB-Leds programmieren
- Fehlersuche GPS
- SD-Karte programmieren

1.8 Mi. 31.10.2012 / Do. 01.11.2012

Frei

1.9 Mi. 07.11.2012 / Do. 08.11.2012

1.9.1 Johannes Schwöller

- Zusammenfügen funktionierender Programmteile
- Aufbau TAS-Schaltung
- Grober Funktionstest der TAS-Schaltung

1.9.2 Sebastian Modl

- Zusammenfügen funktionierender Programmteile
- Aufbau TAS-Schaltung
- Grober Funktionstest der TAS-Schaltung

1.10 Mi. 14.11.2012 / Do. 15.11.2012

1.10.1 Johannes Schwöller

- Exkursion am 14.11.2012
- TAS-Schaltung überprüfen
- Grober Funktionstest der TAS-Schaltung

1.10.2 Sebastian Modl

- Exkursion am 14.11.2012
- TAS-Schaltung überprüfen
- Grober Funktionstest der TAS-Schaltung

1.11 Mi. 21.11.2012 / Do. 22.11.2012

1.11.1 Johannes Schwöller

- Bauteile auswählen
- Platinenlayout vom Serienprodukt

1.11.2 Sebastian Modl

- Bauteile auswählen
- Programmierung Einheitenauswahl

1.12 Mi. 28.11.2012 / Do. 29.11.2012

1.12.1 Johannes Schwöller

- Platinenlayout vom Serienprodukt
- Abwesend am 2. Tag

1.12.2 Sebastian Modl

- Programmierung Einheitenauswahl

1.13 Mi. 05.12.2012 / Do. 06.12.2012

1.13.1 Johannes Schwöller

- Akku-Ladeschaltung entwerfen
- Präsentation erstellen
- Flyer entwerfen

1.13.2 Sebastian Modl

- Akku-Ladeschaltung entwerfen
- Präsentation erstellen
- Strommessung der Prototypenplatine

1.14 Mi. 12.12.2012 / Do. 13.12.2012

1.14.1 Johannes Schwöller

- Vorbereitungen für die 1. Präsentation

1.14.2 Sebastian Modl

- Vorbereitungen für die 1. Präsentation

1.15 Mi. 19.12.2012 / Do. 20.12.2012

1.15.1 Johannes Schwöller

- 1. Präsentation
- Abwesend am 2. Tag

1.15.2 Sebastian Modl

- 1. Präsentation

1.16 Mi. 26.12.2012 / Do. 27.12.2012

Frei

1.17 Mi. 02.01.2013 / Do. 03.01.2013

Frei

1.18 Mi. 09.01.2013 / Do. 10.01.2013

1.18.1 Johannes Schwöller

- Erweiterung der Computersoftware

1.18.2 Sebastian Modl

- Fehlerbehebung der Mikrocontrollersoftware

1.19 Mi. 16.01.2013 / Do. 17.01.2013

1.19.1 Johannes Schwöller

- Vorbereitungen für den Tag der offenen Tür

1.19.2 Sebastian Modl

- Vorbereitungen für den Tag der offenen Tür

1.20 Mi. 23.01.2013 / Do. 24.01.2013

1.20.1 Johannes Schwöller

- Stromlaufplan und Layout der Serienplatine (Backplatine) erstellen

1.20.2 Sebastian Modl

- Stückliste erstellen

1.21 Mi. 30.01.2013 / Do. 31.01.2013

1.21.1 Johannes Schwöller

- Mikrocontrollersoftware erweitern
- Computersoftware erweitern

1.21.2 Sebastian Modl

- Mikrocontrollersoftware erweitern
- Dokumentation überprüfen

1.22 Mi. 06.02.2013 / Do. 07.02.2013

1.22.1 Johannes Schwöller

- Layout von den Serienplatinen kontrollieren
- Dokumentation erweitern

1.22.2 Sebastian Modl

- Dokumentation erweitern

1.23 Mi. 13.02.2013 / Do. 14.02.2013

Frei

1.24 Mi. 20.02.2013 / Do. 21.02.2013

1.24.1 Johannes Schwöller

- Computersoftware überarbeiten
- Jugend Innovativ Projektbericht erstellen
- Dokumentation erweitern

1.24.2 Sebastian Modl

- Computersoftware überarbeiten
- Jugend Innovativ Anhang erstellen
- Dokumentation erweitern

1.25 Mi. 27.02.2013 / Do. 28.02.2013

1.25.1 Johannes Schwöller

- Jugend Innovativ Projektbericht absenden
- Dokumentation erweitern

1.25.2 Sebastian Modl

- SMD-Bauteile besorgen
- Dokumentation erweitern

1.26 Mi. 06.03.2013 / Do. 07.03.2013

1.26.1 Johannes Schwöller

- Hauptplatine Bauteile einlöten (Serie)
- Fehlersuche

1.26.2 Sebastian Modl

- Backplatine Bauteile einlöten (Serie)
- Fehlersuche

1.27 Mi. 13.03.2013 / Do. 14.03.2013

1.27.1 Johannes Schwöller

- Fehlersuche
- Erweiterung der Computersoftware

1.27.2 Sebastian Modl

- Fehlersuche
- Erweiterung der Mikrocontrollersoftware

1.28 Mi. 20.03.2013 / Do. 21.03.2013

1.28.1 Johannes Schwöller

- Fehlersuche
- Grundlagenarbeit

1.28.2 Sebastian Modl

- Fehlersuche
- Grundlagenarbeit
- Erweiterung der Mikrocontrollersoftware

1.29 Mi. 27.03.2013 / Do. 28.03.2013

- Frei

1.30 Mi. 03.04.2013 / Do. 04.04.2013

1.30.1 Johannes Schwöller

- Frei am 1. Tag
- Grundlagenarbeit
- Arbeiten an der Diplomarbeit

1.30.2 Sebastian Modl

- Frei am 1. Tag
- Grundlagenarbeit
- Arbeiten an der Diplomarbeit

1.31 Mi. 10.04.2013 / Do. 11.04.2013

1.31.1 Johannes Schwöller

- Grundlagenarbeit

1.31.2 Sebastian Modl

- Grundlagenarbeit

1.32 Mi. 17.04.2013 / Do. 18.04.2013

1.32.1 Johannes Schwöller

- Abwesend am 1. Tag
- Gesamttest mit dem Projektbetreuer
    - Rundgang um die Schule
    - Getestet: Extremwerte, Groundspeed, Altitude, G-Belastung, GPS-Koordinaten, Menüeinstellungen, Visualisierung mittels PC-Programm und Google Earth

1.32.2 Sebastian Modl

- Grundlagenarbeit
- Gesamttest mit dem Projektbetreuer
    - Rundgang um die Schule
    - Getestet: Extremwerte, Groundspeed, Altitude, G-Belastung, GPS-Koordinaten, Menüeinstellungen, Visualisierung mittels PC-Programm und Google Earth

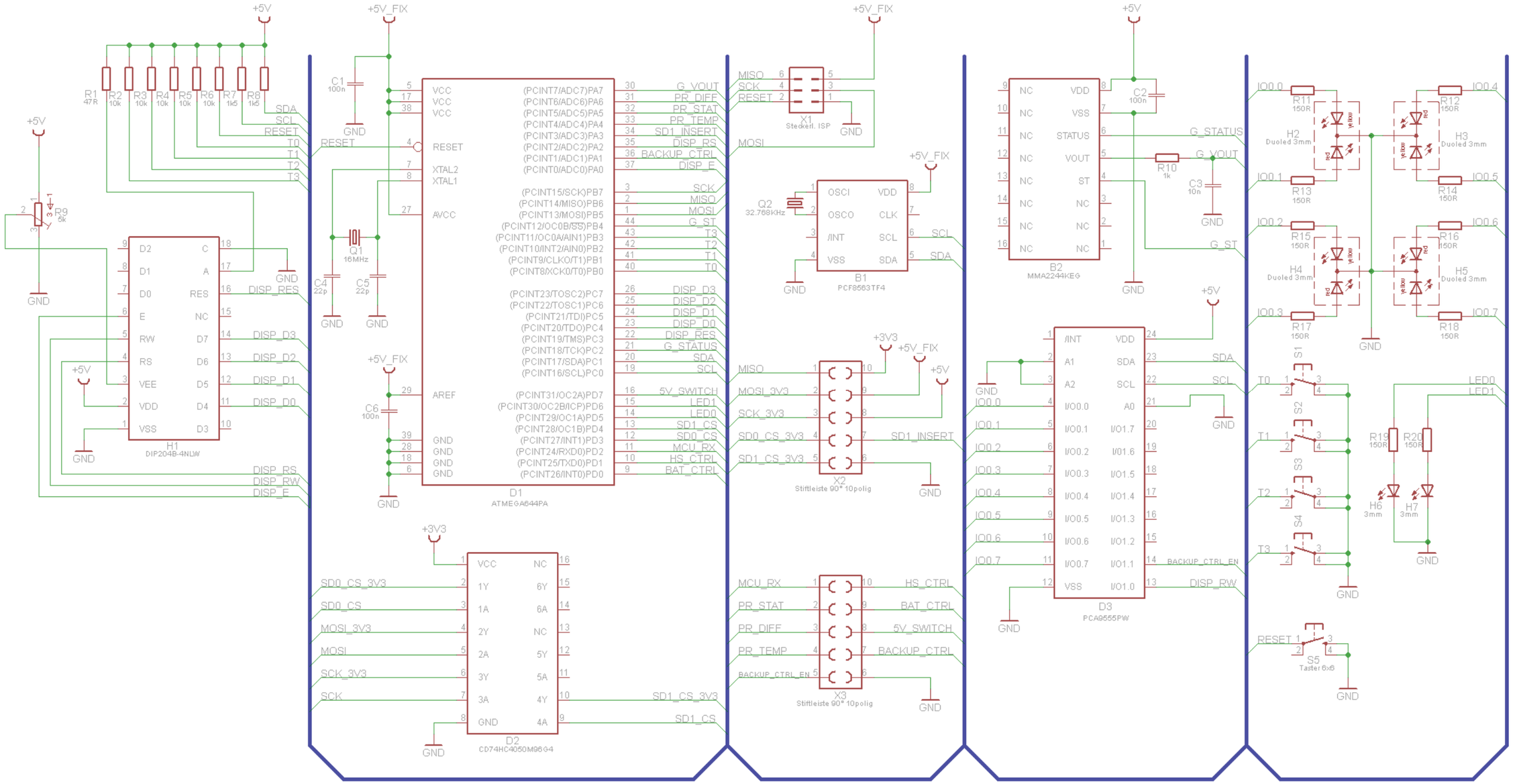
HÖHERE TECHNISCHE BUNDES-LEHR- UND VERSUCHSANSTALT
SALZBURG

Abteilung: Elektronik

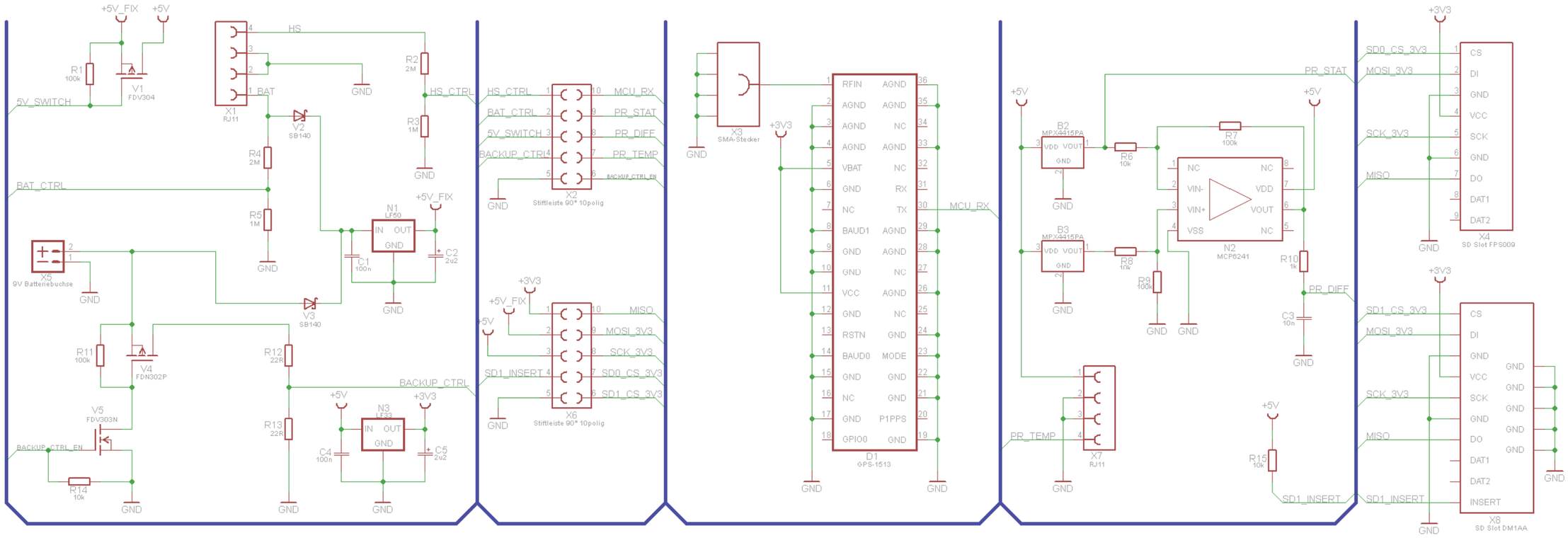
Ausbildungsschwerpunkt: Technische Informatik

Anhang 8.4:

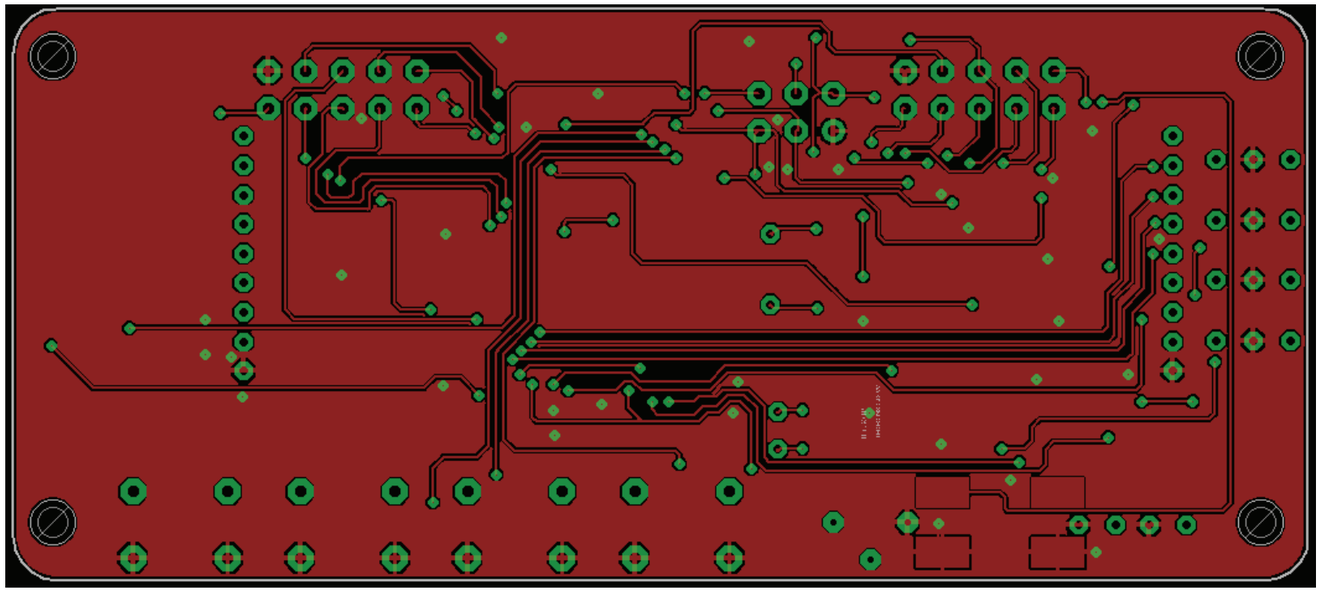
Fertigungsunterlagen

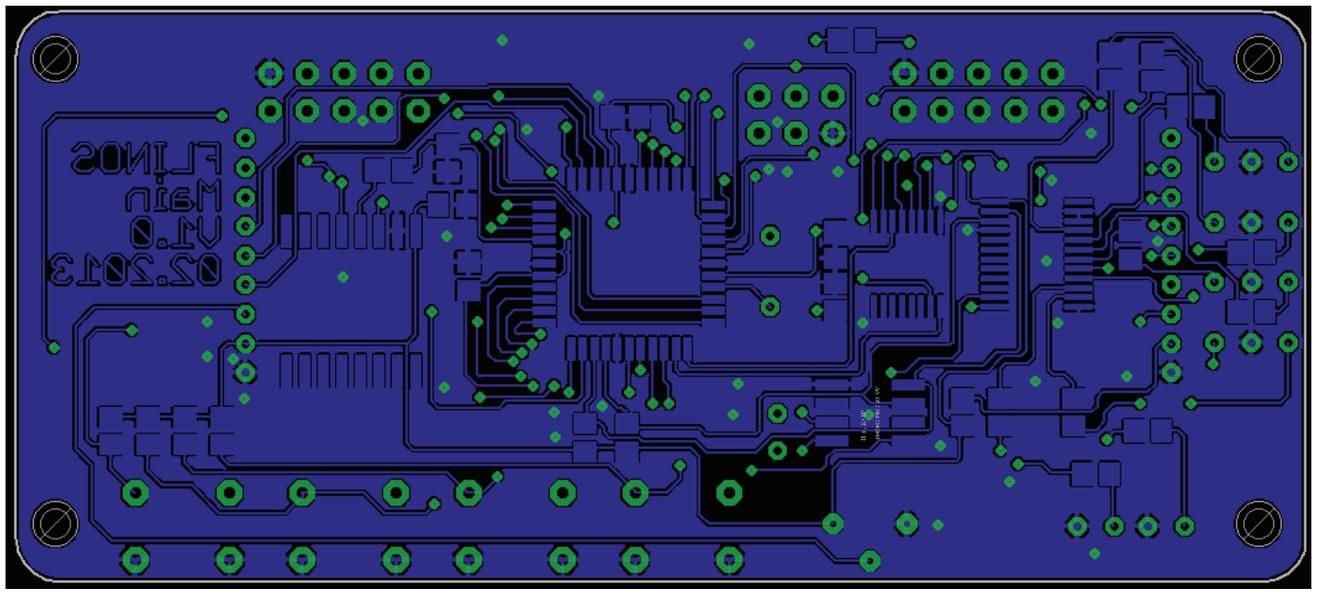


Sachnummer, Dateiname: Stromlaufplan_Main.docx	Name: Johannes Schwöller			Toleranz: ohne	Werkstoff:
HTBLuVA Salzburg, Elektronik	ID-Nr.: 5AHELI, 30	Gez.: SchJ	Geprüft: ModS	Betreuer: SteK	Dokumentart: Stromlaufplan / Freigabe: SchJ
	Benennung: FLINOS Stromlaufplan Main				Version: V1.0 / Revision: R1 / Dokumentstatus: Fertigung
					Maßstab: ohne / Spr.: DE / Datum: 19.04.2013 / Blatt: 1 / Blätter: 1

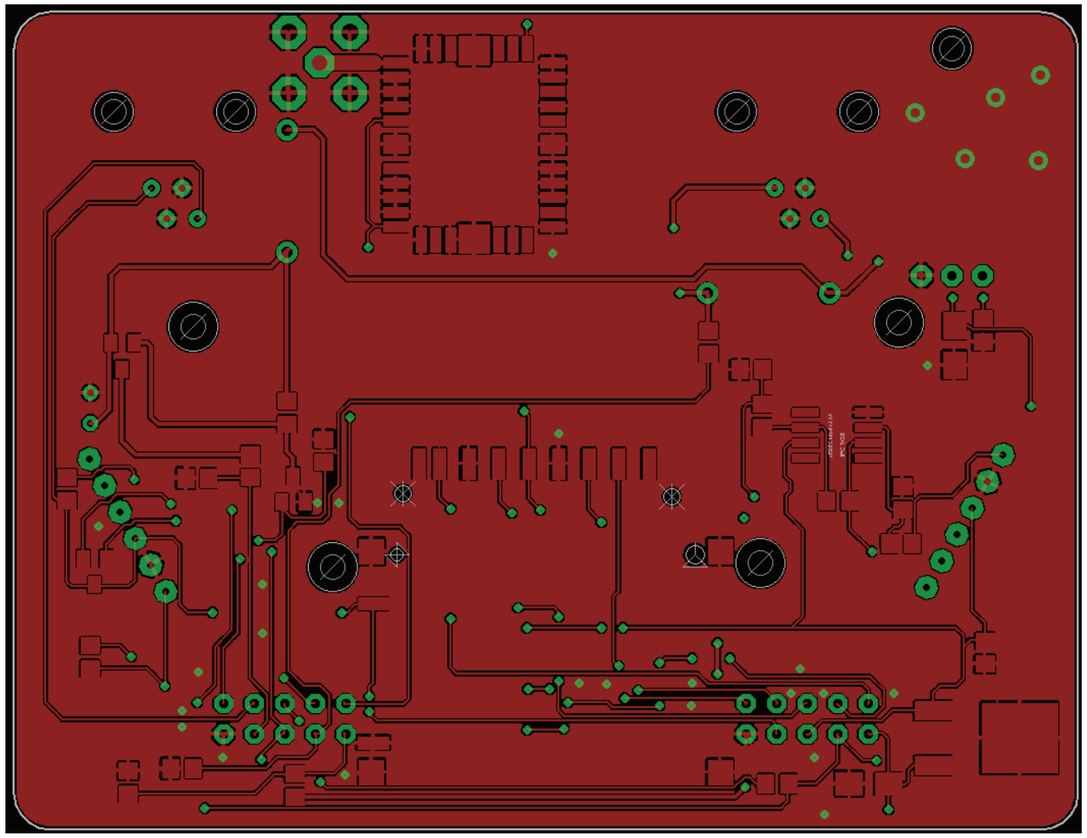


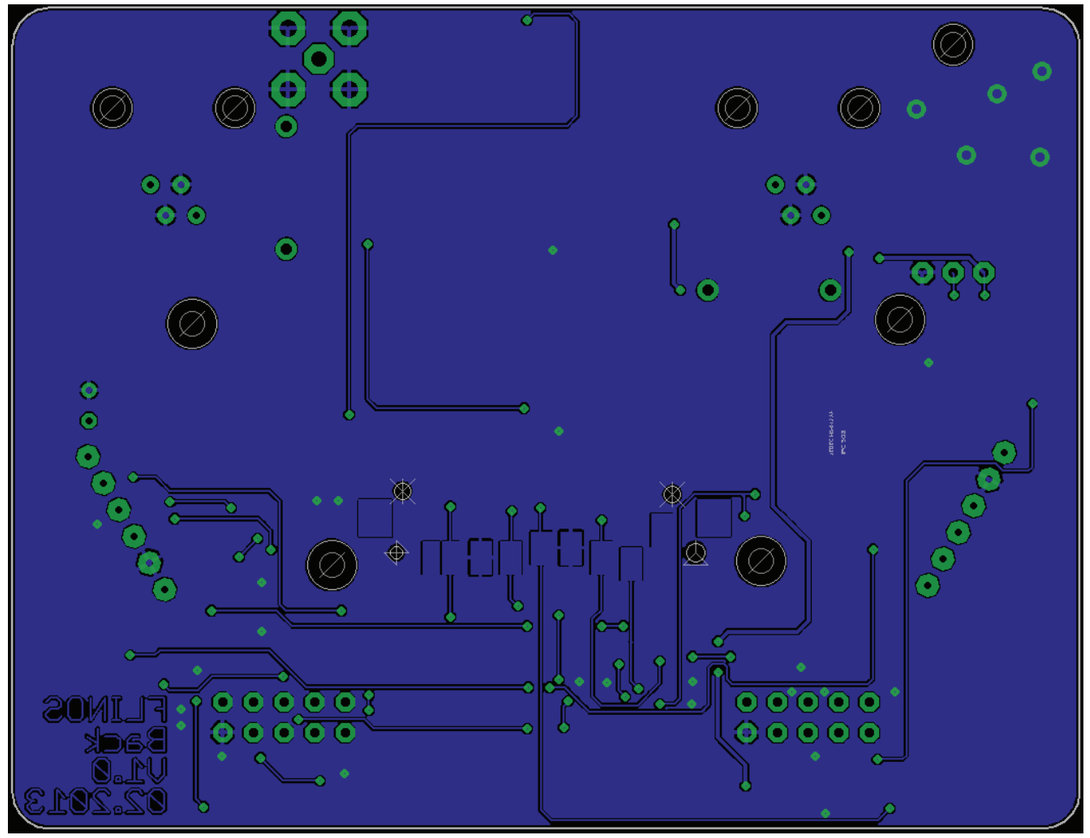
Sachnummer, Dateiname: Stromlaufplan_Back.docx				Name: Johannes Schwöller		Toleranz: ohne	Werkstoff:
htl bildung mit zukunft **HTBLuVA Salzburg** **Elektronik**	ID-Nr.: 5AHELI, 30	Gez.: SchJ	Geprüft: ModS	Betreuer: SteK	Dokumentart: Stromlaufplan		Freigabe: SchJ
	Benennung: FLINOS Stromlaufplan Back				Version: V1.0	Revision: R1	Dokumentstatus: Fertigung
					Maßstab: ohne	Spr.: DE	Datum: 19.04.2013	Blatt: 1	Blätter: 1



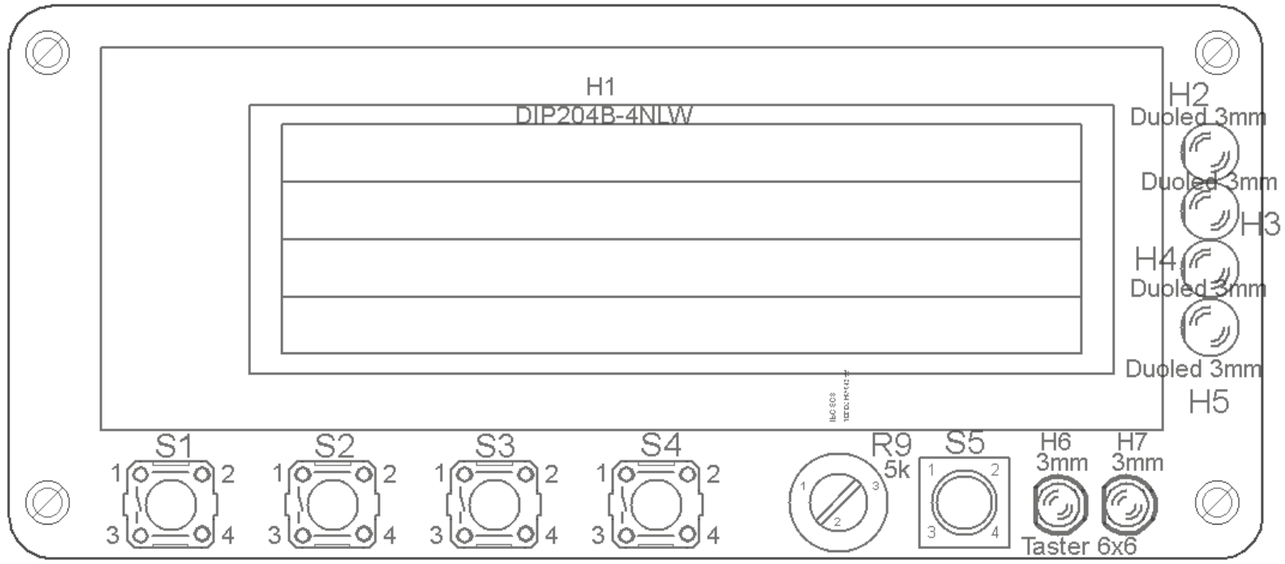


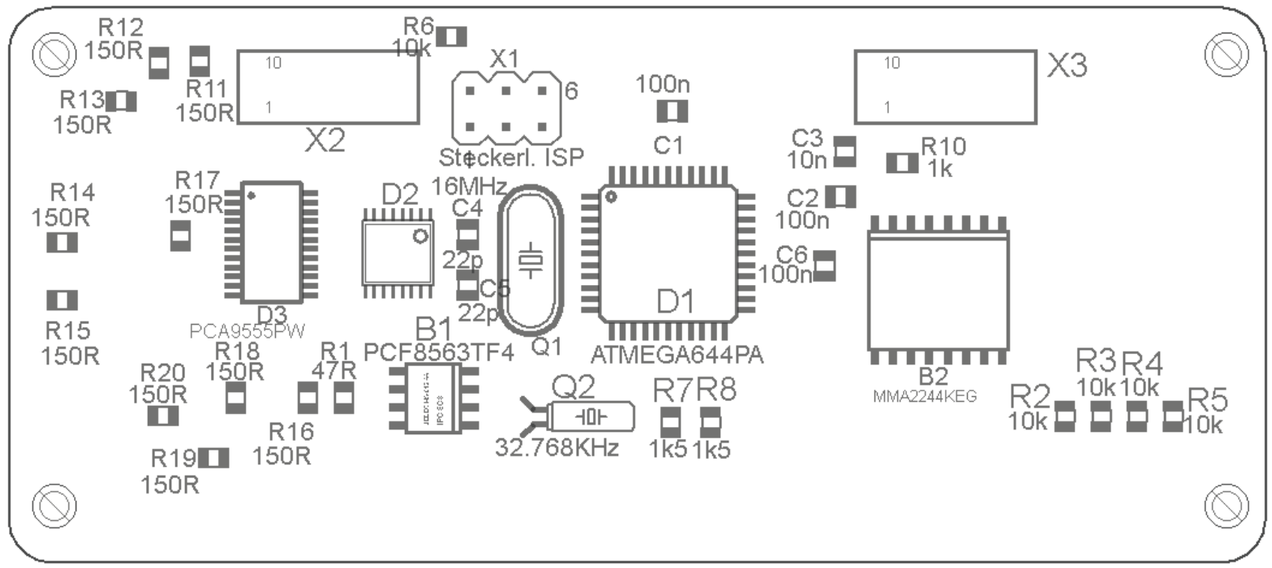
Sachnummer, Dateiname: Layout_Main.docx			Name: Johannes Schwöller			Toleranz: ohne	Werkstoff:
htl bildung mit zukunft **HTBLuVA Salzburg** **Elektronik**	ID-Nr.: 5AHELI, 30	Gez.: SchJ	Geprüft: ModS	Betreuer: SteK	Dokumentart: Layout		Freigabe: SchJ
	Benennung: FLINOS Layout Main Top&Bot				Version: V1.0	Revision: R1	Dokumentstatus: Fertigung
					Maßstab: ohne	Spr.: DE	Datum: 19.04.2013	Blatt: 1	Blätter: 1





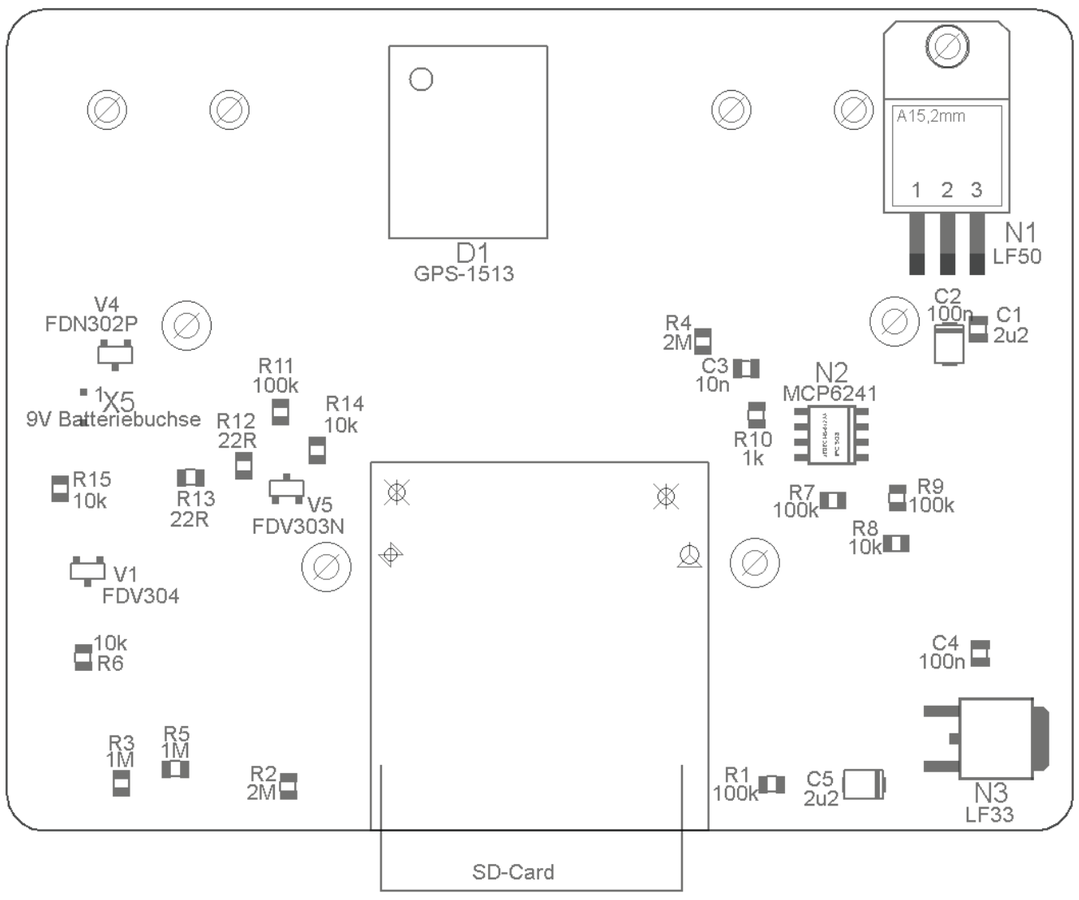
Sachnummer, Dateiname: Layout_Back.docx				Name: Johannes Schwöller		Toleranz: ohne	Werkstoff:
htl bildung mit zukunft **HTBLuVA Salzburg** **Elektronik**				ID-Nr.: 5AHELI, 30	Gez.: SchJ	Geprüft: ModS	Betreuer: SteK
				Benennung: FLINOS Layout Back Top&Bot		Dokumentart: Layout	Freigabe: SchJ
						Version: V1.0	Revision: R1
						Dokumentstatus: Fertigung	
						Maßstab: ohne	Spr.: DE
						Datum: 19.04.2013	Blatt: 1
							Blätter: 1

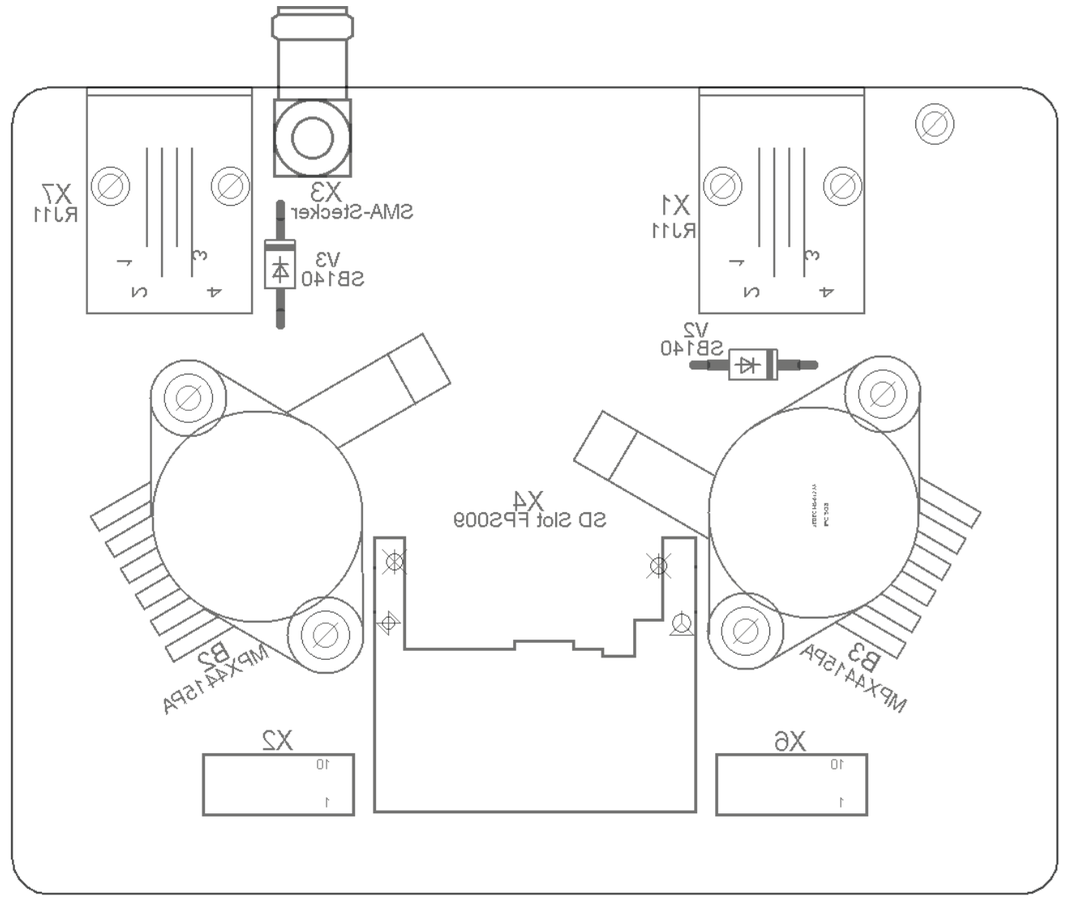




Maße: 88.5x39mm

Sachnummer, Dateiname: Bestueckungsplan_Main.docx	Name: Johannes Schwöller		Toleranz: ohne	Werkstoff:
htl bildung mit zukunft **HTBLuVA Salzburg** **Elektronik**	ID-Nr.: 5AHELI, 30 / Gez.: SchJ / Geprüft: ModS / Betreuer: SteK		Dokumentart: Bestückungsplan	Freigabe: SchJ
	Benennung: FLINOS Bestückungsplan Main Top&Bot		Version: V1.0 / Revision: R1 / Dokumentstatus: Fertigung	
			Maßstab: ohne / Spr.: DE / Datum: 19.04.2013	Blatt: 1 / Blätter: 1





Maße: 88.5x68mm

Sachnummer, Dateiname: Bestueckungsplan_Back.docx	Name: Johannes Schwöller				Toleranz: ohne	Werkstoff:	
htl bildung mit zukunft **HTBLuVA Salzburg** **Elektronik**	ID-Nr.: 5AHELI, 30	Gez.: SchJ	Geprüft: ModS	Betreuer: SteK	Dokumentart: Bestückungsplan		Freigabe: SchJ
	Benennung: FLINOS Bestückungsplan Back Top&Bot				Version: V1.0	Revision: R1	Dokumentstatus: Fertigung
					Maßstab: ohne	Spr.: DE	Datum: 19.04.2013	Blatt: 1	Blätter: 1

HÖHERE TECHNISCHE BUNDES-LEHR- UND VERSUCHSANSTALT
SALZBURG

Abteilung: Elektronik

Ausbildungsschwerpunkt: Technische Informatik

Anhang 8.5:

Vertiefende Grundlagenarbeiten

Vertiefende Grundlagenarbeit

Messtechnische Erfassung der analogen Drucksensordaten und Entscheidung, ob diese Signale vor der Übertragung digitalisiert werden müssen

FLINOS
Flight-Info-System

Ausgeführt im Schuljahr 2012/2013

von:

Johannes Schwöller 5AHELI

Betreuer:

Prof. Dipl.-Ing. Ing. Karl Heinz Steiner

Salzburg, am 07.05.2013

Inhaltsverzeichnis

1 Einleitung

Eine analoge Übertragung kann durch elektromagnetische Einstrahlung sehr leicht gestört werden, wodurch das Sende- und Empfangssignal unter Umständen voneinander abweichen kann. Dadurch, dass im Flugzeug eine elektromagnetische Einstrahlung nicht ausgeschlossen werden kann, ist eine gründliche Untersuchung der analogen Übertragung der Drucksensoren notwendig.
In dieser Arbeit werden die Drucksensordaten ermittelt. Weiters wird untersucht ob elektromagnetische Einflüsse die analoge Übertragung unzulässig stören könnten. Ist das Empfangssignal zu stark verfälscht, müssen Maßnahmen getroffen werden. Eine EMV-Störschaltung oder eine digitale Übertragung der Daten wären alternative Möglichkeiten.

Die digitale Übertragung würde den Aufwand um einiges erhöhen. Eine zusätzliche Platine, auf der ein Drucksensor und ein Analog-Digital-Wandler platziert werden, ist erforderlich. Weiters sind mehr Mikrocontrollerpins (I/O-Pins) für die Übertragung notwendig.

Einstrahlungen auf eine analoge Übertragung können das Ergebnis verfälschen. Schon bei geringen Störungen ist das Auftreten von Informationsverlusten nicht unwahrscheinlich. Anders wie bei der analogen Übertragung, ist die digitale weniger störanfällig. Erst bei einer Überschreitung des Low-Pegels (ca. 0.8V), oder einer Unterschreitung des High-Pegels (ca. 2.0V), kippen die Bits um und es treten Fehler auf. Die genauen Grenzen von High- und Low-Pegel hängen vom jeweiligen Mikrocontroller ab.

2 Drucksensor MPX4115AP[1]

Bei unserem Projekt haben wir uns für den Drucksensor MPX4115AP entschieden. Dieser kann Drücke von 15kPa (0.15bar) bis 115kPa (1.15bar) messen. Der Drucksensor liefert eine analoge Spannung, die proportional dem Druck ist. Da es sich um einen analogen Sensor handelt, ist die Ansteuerung sehr einfach. Der Sensor hat lediglich einen V_{in} (+5V), GND und einen V_{out} (0.204V – 4.794V) Anschluss.

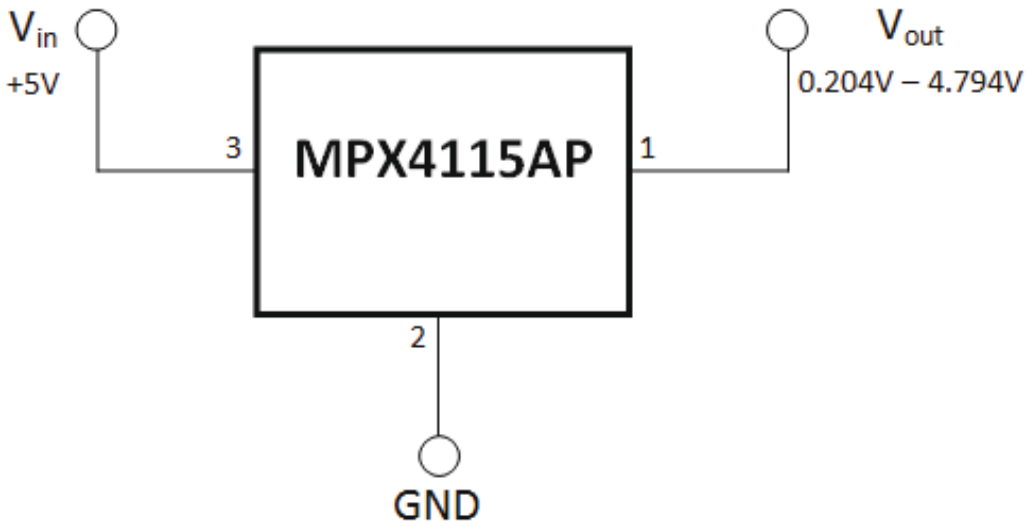


Abb. 1: Pinanschlüsse des Drucksensors

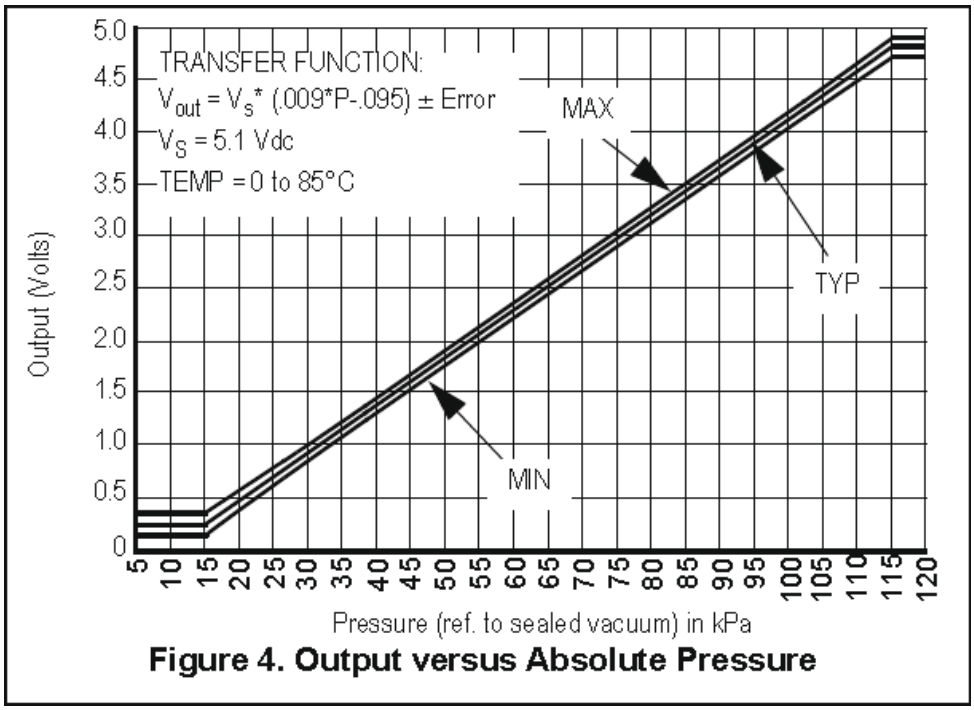


Abb. 2: Diagramm Ausgangsspannung in Abhängigkeit des Druckes[2]

[1] vgl. http://docs-europe.electrocomponents.com/webdocs/0ef4/0900766b80ef40fa.pdf (02.04.2013)
[2] http://docs-europe.electrocomponents.com/webdocs/0ef4/0900766b80ef40fa.pdf (02.04.2013)

Wie in Abbildung 2 zu sehen ist, sind die Ausgangsspannung und der physikalische Druck proportional zueinander. Die Empfindlichkeit beträgt 46mV pro 1kPa. Eine Offsetspannung von 0.204V und eine maximale Ausgangsspannung von 4.794V sind im Diagramm als Waagrechte dargestellt. Neben der typischen Kennlinie sind Graphen mit Minimum und Maximum ebenfalls aufgetragen.

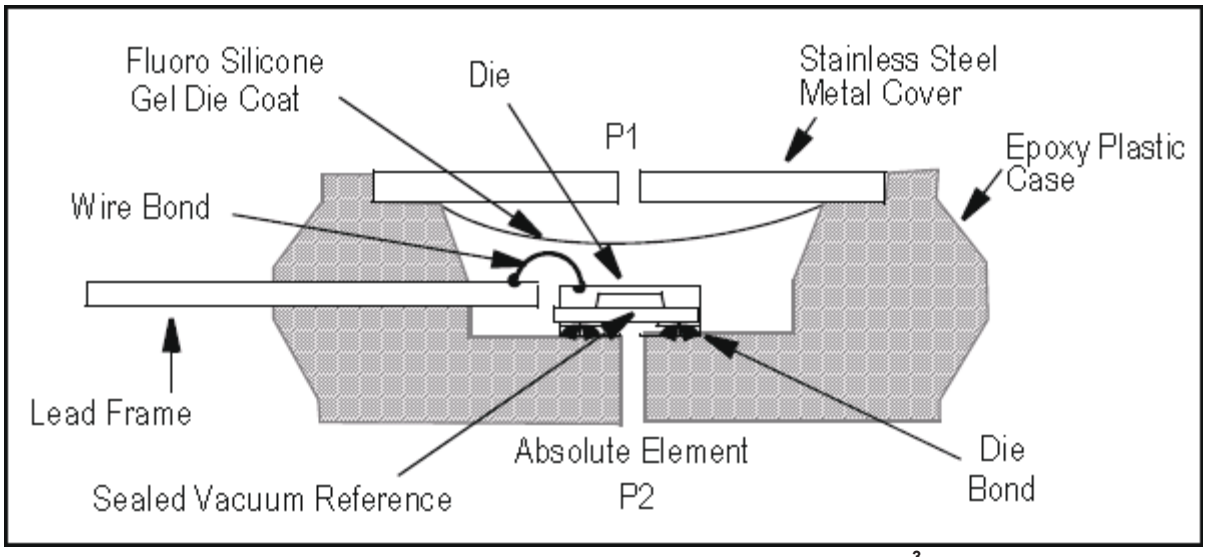


Abb. 3: Physikalischer Aufbau des Drucksensors[3]

Englisch	**Deutsch**
Fluoro Silicone	Fluorsilikon
Die	Aufnahme
Stainless Steel Metal Cover	Rostfreie Stahlbedeckung
Epoxy Plastic Case	Epoxid-Kunststoffrahmen
Wire Bond	Drahtanschluss
Lead Frame	Leitungsanschluss
Sealed Vaccum Reference	Vakuumdichte Referenzpunkt
Absolut Element	Absolutes, ganzes Bauteil
Die Bond	Aufnahmeverbindung

[3] http://docs-europe.electrocomponents.com/webdocs/0ef4/0900766b80ef40fa.pdf (02.04.2013)

Die Gelschicht aus Fluorsilikon trennt die Oberfläche und die Drahtanschlüsse von der Umwelt, aber erlaubt die Übertragung des Druckes an die Sensormembran. Aufgrund der vakuumdichten Referenz im Inneren, wird der analoge Spannungswert am Drahtanschluss angelegt.
Dieser Drucksensor ist ausgelegt für Messungen in der Trockenluft. Bei anderen Umgebungsmedien können die Betriebseigenschaften und die gemessenen Daten verändert oder verfälscht werden.

Abb. 4: Drucksensor MPX4115AP[4]

[4] http://t1.gstatic.com/images?q=tbn:ANd9GcTK-yzUzl6IaxdAcrcrekRd7LikqwfUukHdVhysUtP3eNCT_HqDQ (16.04.2013)

3 EMV[5]

Die Funktion elektrischer Bauteile, Geräte, Schaltungen usw. kann durch elektromagnetische Einstrahlung störend beeinträchtigt oder beeinflusst werden.
Die Umgebung, bei der das Produkt eingesetzt wird, muss schon sehr früh auf EMV untersucht werden. Es ist aber sehr schwer, diese Umgebung im Labor für die Prototypenentwicklung nachzubilden.
Der elektrische Aufbau und die Verkabelung sind meist von Flugzeug zu Flugzeug unterschiedlich. Verschiedene Boardinstrumente und weitere elektronische Verbraucher verursachen elektromagnetische Störungen. Da in jedem Ultraleichtflugzeug dadurch eine andere elektromagnetische Umgebung herrscht, ist es sehr schwer, solch eine Umgebung realistisch nachzubilden.
Um dennoch brauchbare Ergebnisse erzielen zu können, wurden konkrete Fälle in Betracht gezogen.
Um eine rasche Entwicklung, unter Berücksichtigung der EMV, zu gewährleisten, sind folgende Phasen durchzuarbeiten:

- Definitionsphase: Umgebung auf EMV untersuchen
- Projektierungsphase: Auswirkungen auf Bauteile und Baugruppen
- Entwicklungsphase: Auswirkungen auf den Schaltplan
- Layoutentwurf: Auswirkungen auf das Layout, gute Zusammenarbeit mit Schaltplanzeichner erforderlich
- Testphase: Analyse der EMV-Fehler

Folgende Darstellung soll die Problemstellung etwas näher bringen. Die EMV-Störung tritt bei der analogen Übertragung auf.

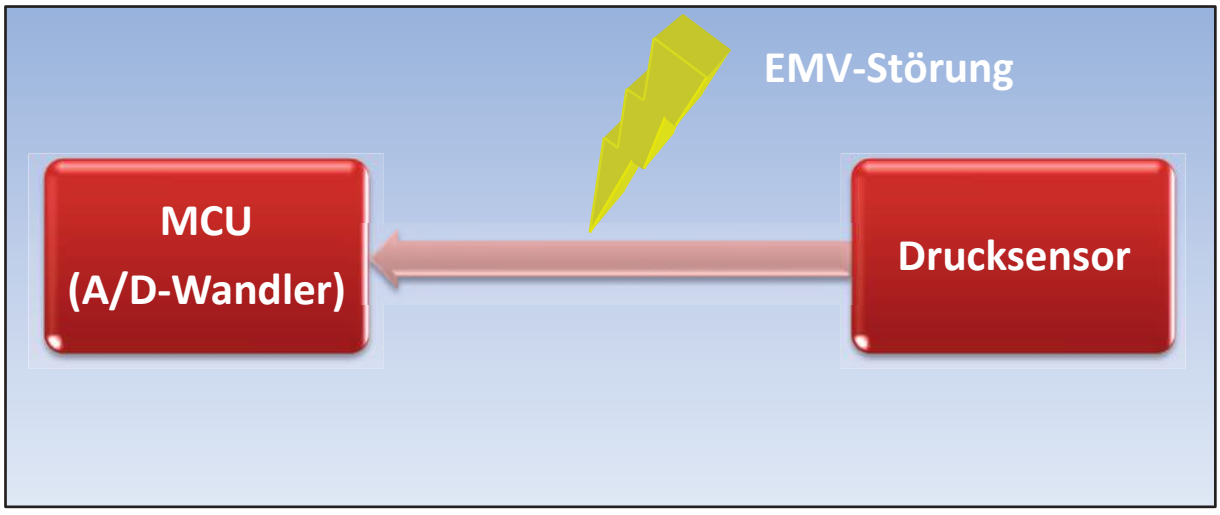


Abb. 5: Grundprinzip der Schaltung

[5] vgl. Franz, 2010, S. 18-23

4 Geschwindigkeitsermittlung

Die Drücke werden für die Geschwindigkeitsermittlung verwendet. Das bedeutet, dass der Drucksensor für geringe Druckänderungen ausgelegt sein muss. Wir verwenden den Mikrocontroller ATmega644PA, welcher in der SMD-Bauform verfügbar ist. Für diese Grundlagenarbeit jedoch verwende ich das Crumb-Modul. Auf dem Crumb-Modul ist der Mikrocontroller ATmega644P angebracht, der sich nur geringfügig zum vorher Erwähnten unterscheidet. Die MCU wird mittels Bootloader programmiert. Hierfür sind lediglich eine USB-Verbindung und ein spezielles Programm (chip45boot2 GUI) nötig.
Eine wichtige Rolle spielt auch die Auflösung des Analog-Digital-Konverters des Mikrocontrollers. Die analoge Spannung wird in einen 10-bit Wert umgewandelt. Eine Spannungsänderung muss mindestens 5mV betragen um erkannt zu werden.

Im nachfolgenden Diagramm ist die Geschwindigkeit in Abhängigkeit von der Spannung des Differenzdruckes abgebildet. Im Bereich von 0-200km/h treten Störungen gravierender auf, als bei höheren Geschwindigkeiten. Die Schrittweite ist zu Beginn sehr groß. Ab 100km/h ist die Messung schon sehr genau. Außerdem werden geringe Geschwindigkeiten (unter 100km/h) mit dem Flugzeug nur selten erreicht (Nur für kurze Zeit bei Start und Landung).

4.1 Berechnung[6]

Annahmen für das Diagramm (Abbildung 6):

$p_{statisch}$ = 693.5Pa (3.19V)

Δp = 1.087Pa – 1087Pa (0.005V – 5.000V)

T = 293K (~20°C)

$$v = \sqrt{\frac{2 \cdot \Delta p \cdot R \cdot T}{p_{statisch}}} \cdot 3.6$$

$$\Delta p = p_{stau} - p_{statisch}$$

v	Geschwindigkeit [km/h]
Δp	Differenzdruck [Pa]
R	Gaskonstante Luft 287,058[J/(kg*K)]
T	aktuelle Temperatur [K]
$p_{statisch}$	statischer Luftdruck [Pa]
p_{stau}	Staudruck [Pa]

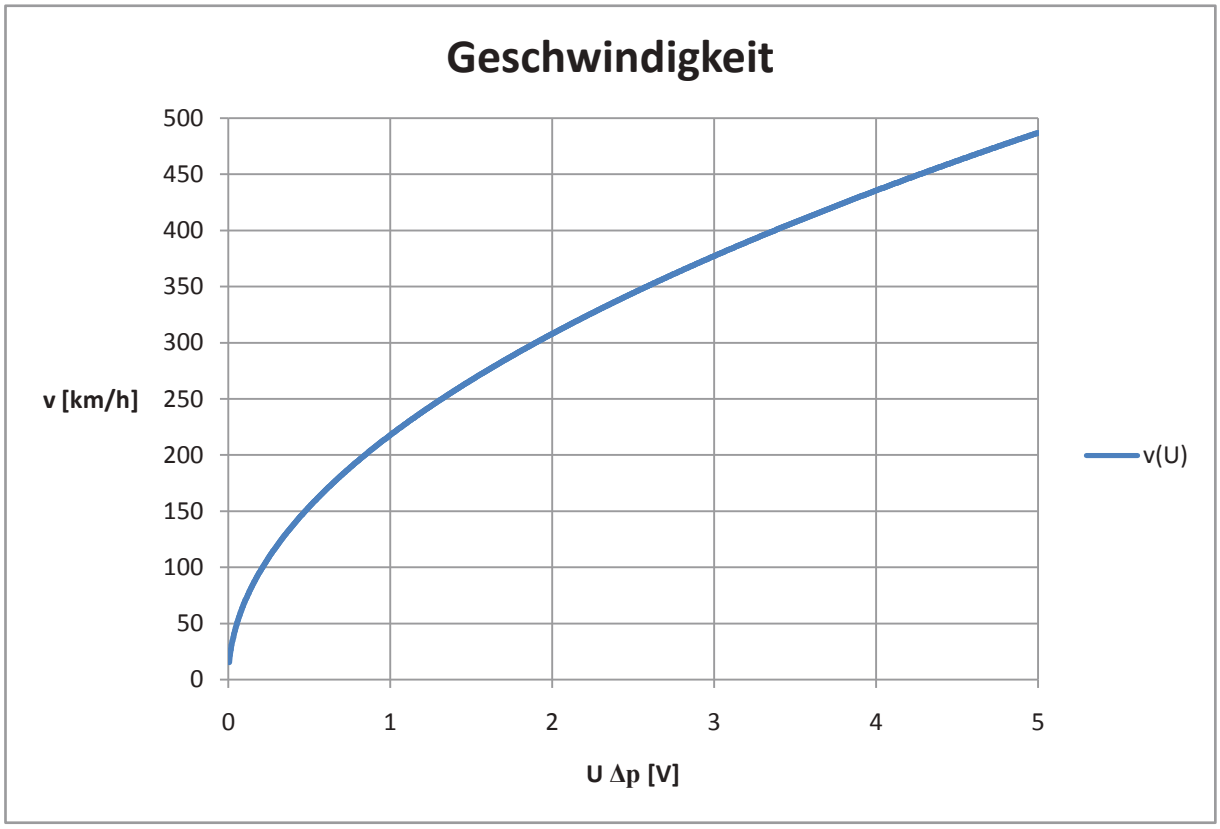


Abb. 6: Geschwindigkeit in Abhängigkeit von der Spannung des Differenzdruckes

[6] vgl. http://www.electro-mation.de/productattachments/index/download?id=47 (02.04.2013)

5 Sinusförmige Störspannung bei der analogen Übertragung

Aufgrund der Auflösung des internen A/D-Wandlers (5mV), wird eine sinusförmige Störspannung mit einer Amplitude von 10mV bei der Übertragung eingespeist. Bei 10mV Amplitudenstörspannung geht man davon aus, dass die untersten 2 Bits möglicherweise kippen werden. Bei niedrigen Geschwindigkeiten (bis 100km/h) wirkt sich eine Störung enorm aus. Bei 10mV Störspannung beträgt der Fehler der Geschwindigkeit etwa 5-10km/h. Bei höherer Geschwindigkeit reduziert sich der Fehler auf etwa 2-3km/h, was toleriert werden kann.

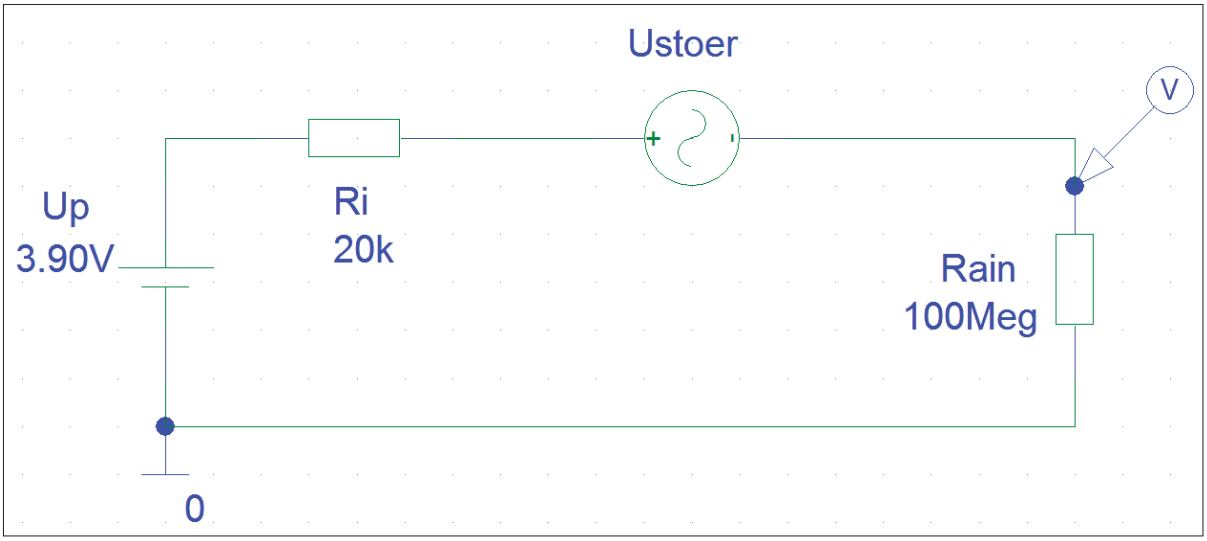


Abb. 7: Messschaltung mit sinusförmiger Störspannung

U_p	3.90V (Drucksensorausgang, Umgebungsdruck ca. 1bar)
R_i	20k (Innenwiderstand Drucksensors)
U_{stoer}	Û = 10mV, f = 1Hz-3.5MHz (sinusförmige Störspannung)
R_{ain}	100MΩ (Eingangswiderstand A/D-Wandler der MCU

Der Innenwiderstand des Drucksensors wurde mit einem Digitalmultimeter gemessen. Der Eingangswiderstand des A/D-Wandlers des Mikrocontrollers ist im Datenblatt angegeben. Die Spannungsquelle stellt den Ausgang des Drucksensors dar und wird mit 3.90V angenommen (Umgebungsdruck). Der Eingangswiderstand der MCU ist um ein Vielfaches größer als der Innenwiderstand des Sensors. Dadurch fällt beinahe die gesamte Spannung am R_{ain} ab (1mV Fehler).
Mittels Spannungsteilerregel wird die Spannung U_{Rain}, die am R_{ain} abfällt, berechnet.

$$U_{R_{ain}} = U_p \cdot \frac{R_{ain}}{R_{ain} \cdot R_i} = 3.90V \cdot \frac{100 \cdot 10^6\Omega}{100 \cdot 10^6\Omega + 20 \cdot 10^3\Omega} = 3.899V$$

Aufgrund der sinusförmigen Störspannung kommt es nun zu einer Überlagerung. Die Amplitude der Störspannung ist ausschlaggebend für die Abweichung der Spannung am R_{ain}.

$$U_{R_{ain,max}} = (U_p + \hat{U}_{stoer}) \cdot \frac{R_{ain}}{R_{ain} \cdot R_i} = (3.90V + 0.01V) \cdot \frac{100 \cdot 10^6\Omega}{100 \cdot 10^6\Omega + 20 \cdot 10^3\Omega} = 3.909V$$

$$U_{R_{ain,min}} = (U_p - \hat{U}_{stoer}) \cdot \frac{R_{ain}}{R_{ain} \cdot R_i} = (3.90V - 0.01V) \cdot \frac{100 \cdot 10^6\Omega}{100 \cdot 10^6\Omega + 20 \cdot 10^3\Omega} = 3.889V$$

$U_{Rain,max}$ und $U_{Rain,min}$ weichen jeweils um 10mV vom ursprünglichen Signal (U_{Rain}, ohne Störung) ab. Wie vorher schon erwähnt stellt dies ein Problem bei der Berechnung von geringen Geschwindigkeiten dar.

5.1 Simulation

Die Messung und Simulation wurde mit einer sinusförmigen Störspannungsamplitude von 10mV und einer sehr niedrigen (1Hz) bzw. sehr hohen (4MHz) Frequenz durchgeführt.
Die Simulation zeigt eine Gleichspannung von 3.90V, die mit einer sinusförmigen Störspannung überlagert ist. Die Simulation mit der höheren Frequenz unterscheidet sich zur Niedrigen nur indem die Zeitachse (X-Achse) anders skaliert ist. Erst aufgrund der Messung kann man Rückschlüsse und Konsequenzen daraus ziehen.

5.1.1 Sinusförmige Störspannung, 10mV 1Hz

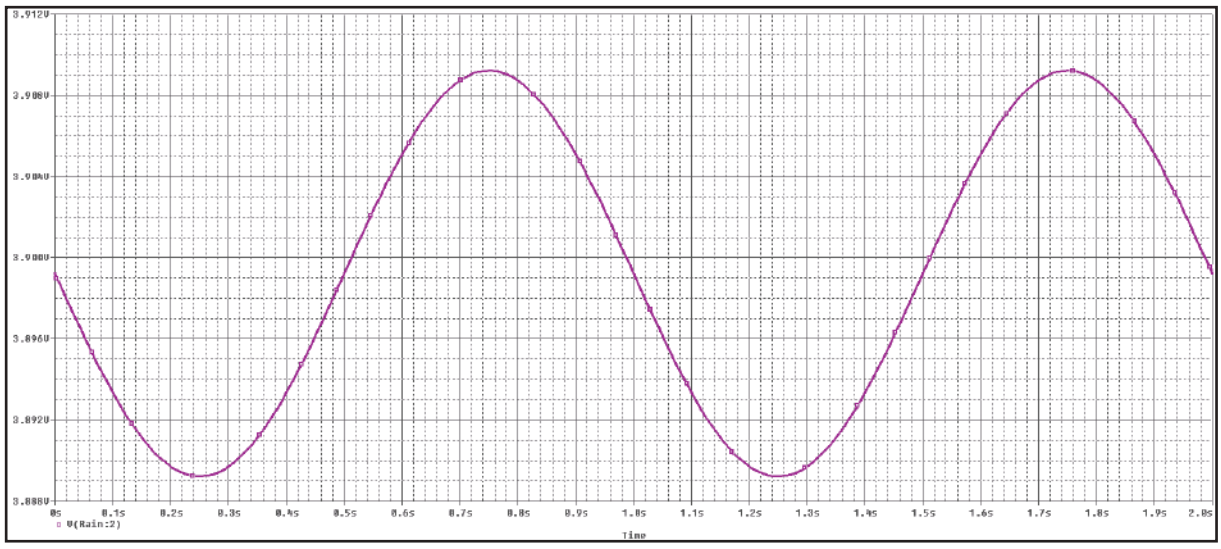


Abb. 8: Sinusförmige Störspannung, 10mV 1Hz

5.1.2 Sinusförmige Störspannung, 10mV 4MHz

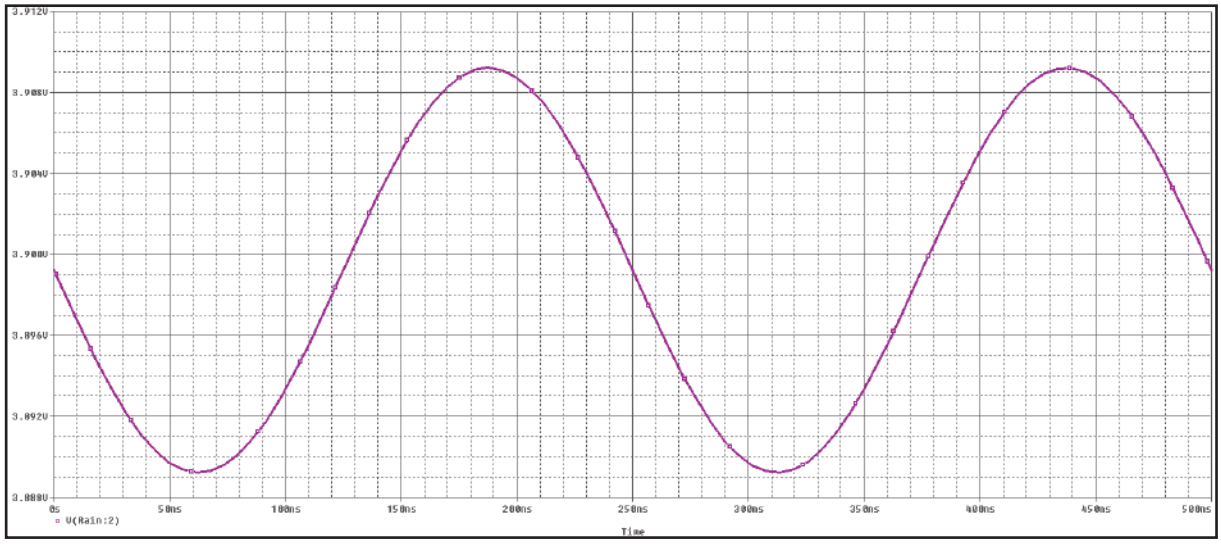


Abb. 9: Sinusförmige Störspannung, 10mV 4MHz

5.2 Messung

5.2.1 Messaufbau

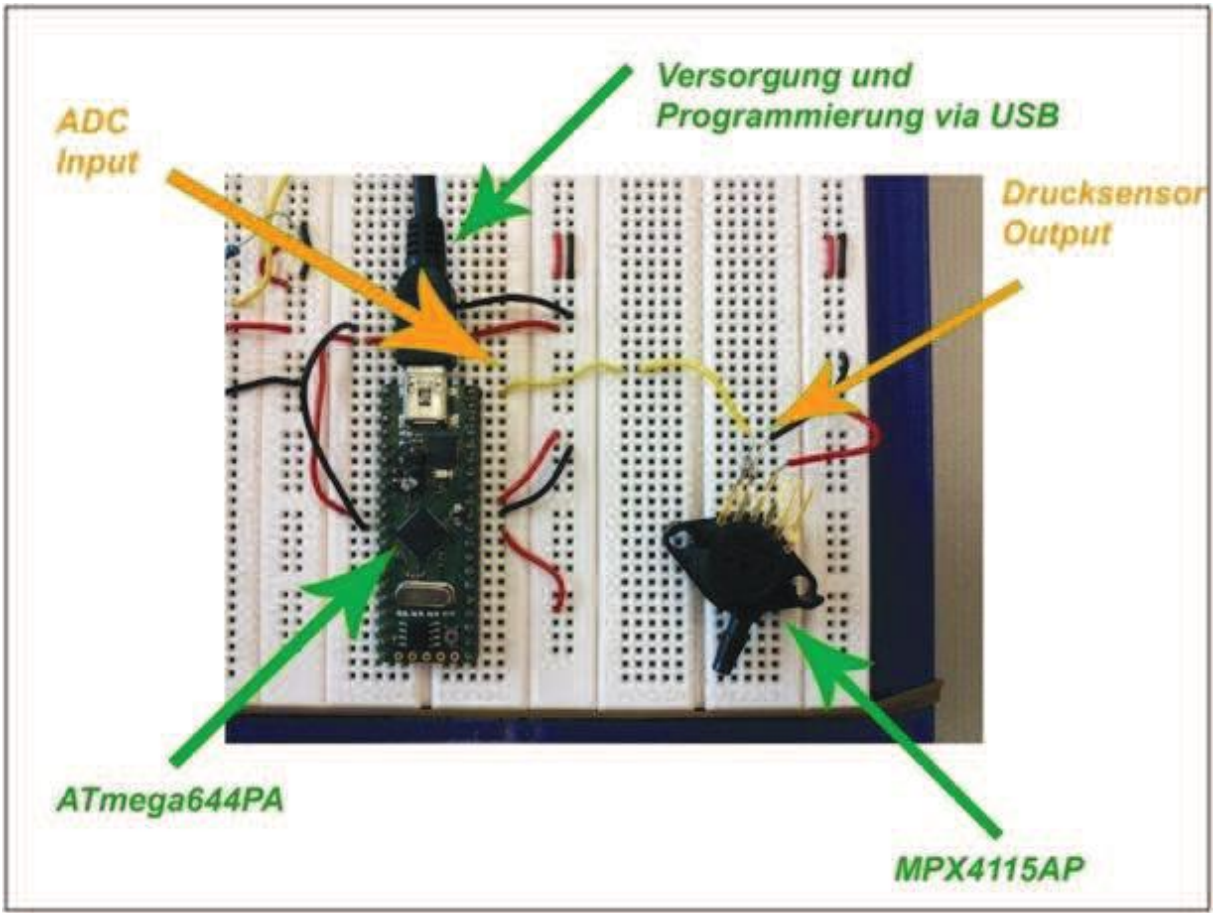


Abb. 10: Schaltungsaufbau ohne Störspannung

Die Spannungsversorgung und Programmierung erfolgt via USB. Auf dem Crumb-Modul befindet sich der Mikrocontroller. Dieser wird mittels Bootloader programmiert. Die Referenzspannung des Mikrocontrollers, welche für die Analog/Digital-Wandlung notwendig ist, ist mit +5V verbunden. Der Ausgang des Drucksensors ist mit dem PINA0 der MCU verbunden. Die Störspannung wird nun zwischen dem Drucksensorausgang und dem analogen Eingang des Mikrocontrollers (PINA0) seriell dazugeschalten (siehe Abbildung 11).

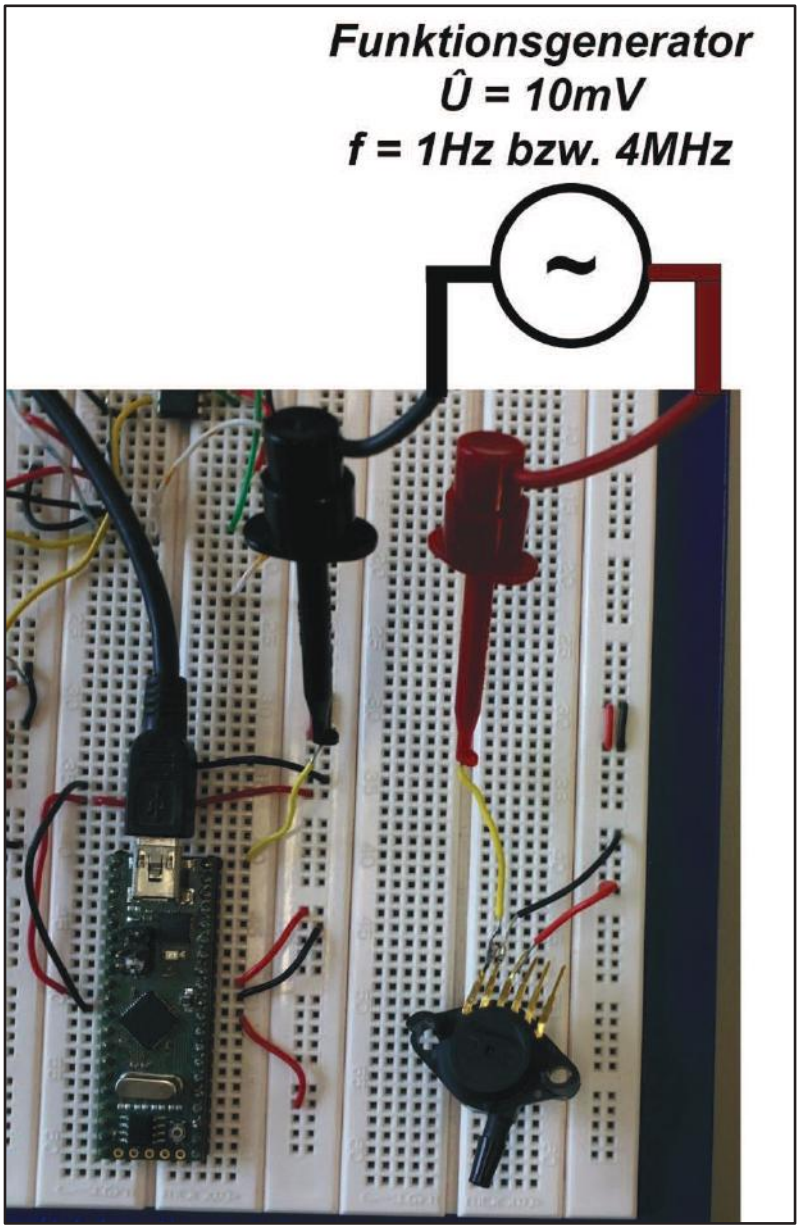


Abb. 11: Messung mit sinusförmiger Störspannung

Wichtig bei der Messung war es, dass die Versorgung des Laptops abgesteckt und auf Akkuverbrauch umgeschaltet wurde. Dies ist nötig, weil die Spannungsversorgung der Schaltung per USB funktioniert. Die Masse (Ground) des Mikrocontrollers und dem Funktionsgenerators darf nicht ident sein. Der Wert, der eingelesenen Spannung, wäre ansonsten immer 0.0V.

5.2.2 Ermittlung der Spannung mittels Digitalmultimeter

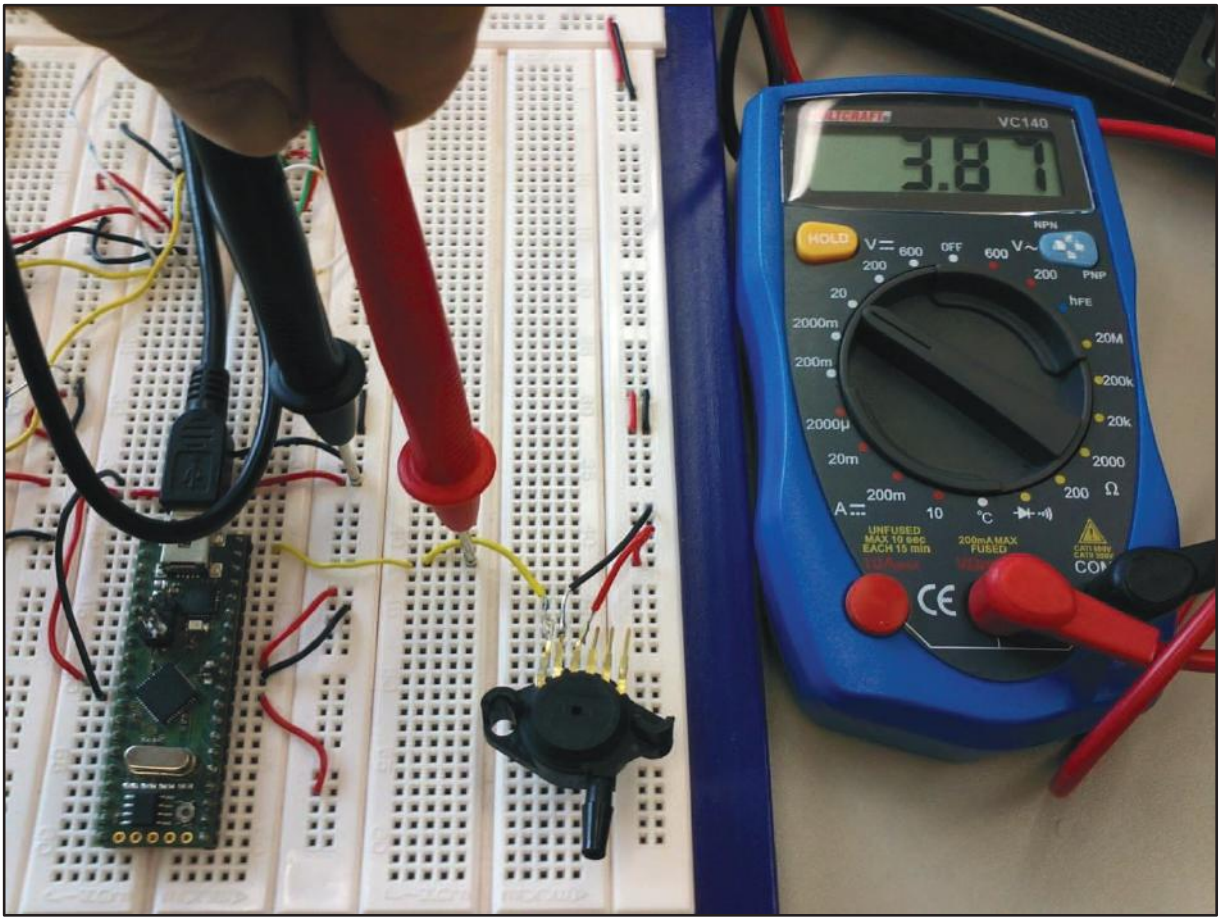


Abb. 12: Ausgangspannung des Drucksensors

In der obigen Abbildung ist die gemessene Ausgangsspannung mittels Digitalmultimeter zu sehen. Die Spannung beträgt 3.87V.
Nachfolgende Berechnung zeigt die Umrechnung der Spannung in einen Druck:

$$p = \frac{p_{max} \cdot U_p}{U_{max}} = \frac{115000Pa \cdot 3.87V}{4.794V} = 92835Pa = 0.92835bar$$

p	Aktueller Druck [Pa oder bar]
p_{max}	Maximaler Druck des Sensors 115kPa
U_p	Aktuelle Spannung des Sensors
U_{max}	Maximale Ausgangsspannung des Drucksensors 4.794V

Der Druck beträgt in diesem Fall 0.92853bar (Umgebungsdruck). Aufgrund dessen, dass wir etwa 400-500m über dem Meeresspiegel liegen (Stadt Salzburg), ist der Druck etwas geringer als 1.0bar.

5.2.3 Ermittlung der Spannung mittels A/D-Wandler der MCU[7]

Die eingelesene Spannung wir mittels A/D-Wandler in einen 10bit-Wert umgewandelt. Im Programm wird der digitale Wert wieder in einen Spannungswert umgeformt, um einen direkten Vergleich der Spannungen zu erhalten.

Die Ausgabe der Daten erfolgt mit dem Programm HTerm mittels USART.

Programmcode:

```
#include <avr/io.h>
#include <avr/interrupt.h>
#define F_CPU 16000000
#include "ADC.h"
#include "Timer0.h"
#include "USART.h"

int main(void)
{
        USART_Init(9600, init);
        USART_TransmitString("start");
        ADC_Init();
        Timer0_Init();

        sei();

    while(1)
    {
    }
}
```

Zu Beginn werden sämtliche Headerfiles inkludiert. Die interne Oszillatorfrequenz der MCU wird auf 16MHz eingestellt.

In der main-Funktion werden USART, der ADC und der Timer0 initialisiert. Mit dem Befehl *`sei();`* werden die Interrupts freigeschaltet. In der Endlosschleife werden keine weiteren Befehle angeführt.

Folgende Funktionen werden für die Übertragung zum PC benötigt (USART):

- `void USART_Init(unsigned int baud, USARTreceiveFunction receiveFunction)`
  - Einstellung der Bitrate und Aufruf einer Funktion bei jedem Interrupt
- `void USART_Transmit(unsigned char data)`
  - Daten zum PC senden
- `unsigned char USART_Receive(void)`
  - Daten vom PC erhalten (wird bei diesem Projekt nicht benötigt)
- `void USART_TransmitString(char * data)`
  - Einen String zum PC senden
- `void USART_TransmitNumber(char maxLength, int val)`
  - Einen Integer mit variabler Länge zum PC senden

Genauer wird auf die angeführten Funktionen nicht eingegangen, da sie nur für die Übertragung und Anzeige der Daten dienen.

[7] http://docs-europe.electrocomponents.com/webdocs/0dc5/0900766b80dc5089.pdf (10.04.2013)

```
void ADC_Init()
{
        DDRA &= ~(1<<PIN_AI0);  //Analog Input PINA0
        PORTA &= ~(1<<PIN_AI0);

        /* Analog Input on PINA0 */
        ADMUX &= ~(1<<MUX0);
        ADMUX &= ~(1<<MUX1);
        ADMUX &= ~(1<<MUX2);
        ADMUX &= ~(1<<MUX3);
        ADMUX &= ~(1<<MUX4);

        ADMUX |= (1<<REFS0);    //external ref voltage
        ADMUX &= ~(1<<REFS1);   //external ref voltage
        ADCSRA |= (1<<ADATE);    //free running
        ADCSRA = (1<<ADPS2) | (1<<ADPS1) | (1<<ADPS0);  //Prescale, 128
        ADCSRA |= (1<<ADIE);    //interruppt enable

        ADCSRA |= (1<<ADEN);    //adc enable

        sei();

        ADCSRA |= (1<<ADSC);// eine ADC-Wandlung
        while (ADCSRA & (1<<ADSC)) {}
}
```

In der Funktion *ADC_Init()* wird zu Beginn der PINA0 als Input definiert. Danach wird der ADC am PINA0 aktiviert. Als Referenz dient die externe Referenzspannung mit zusätzlichem Kondensator. Weiters wurde der Freerunning-Mode eingestellt. So wird nach jeder abgeschlossenen Messung wieder ein neuer Wert eingelesen.
Der Prescaler wurde auf 128 eingestellt. Nachfolgende Berechnung zeigt die Frequenz, mit der der ADC arbeitet.

$$f_{ADC} = \frac{f_{MCU}}{Prescaler} = \frac{16MHz}{128} = 125kH$$

Am Ende der ADC-Initialisierung ist es notwendig, eine ADC-Wandlung durchzuführen. Ansonsten wird das Ergebnis der nächsten Wandlung nicht übernommen.

```
int ADC_GetValue()
{
        return ADC_result;
}

ISR(ADC_vect)
{
        ADC_result = ADCW;
        ADCSRA &= ~(1<<ADIF);
}
```

ADC_GetValue() liefert den aktuell eingelesenen Wert. In der Interruptroutine wird der Variable *ADC_result* der Wert aus dem ADC-Datenregister zugewiesen. Weiters wird das ADC-Flag wieder rückgesetzt, damit eine neue Messung wieder erkannt wird.

```
void Timer0_Init()
{
        TCNT0 = 0;
        TCCR0B |= (1<<CS02);
        TCCR0B &= ~(1<<CS01);
        TCCR0B &= ~(1<<CS00);
        OCR0A = 255;
        TIMSK0 |= (1<<OCIE1A);

        // Global Interrupts activated
        sei();
}
```

Ein Timer ist notwendig, um die Daten in bestimmten Zeitintervallen zu erhalten. Bei dieser Arbeit wurde der Timer0 (8bit-Timer) im CTC Modus (Clear Timer on Compare Match) verwendet. Zu Beginn wird wie beim ADC ein Prescaler verwendet, um die Frequenz des Timers einstellen zu können.

$$f_{Timer\,0} = \frac{f_{MCU}}{256} = \frac{16MHz}{256} = 62.5kHz$$

Das *OCR0A* Register wird auf den Wert 255 gesetzt. Erreicht nun der Timer den Wert 255, so wird ein Interrupt ausgelöst.

```
void ADC_ReceiveData()
{
        ADCSRA |= (1<<ADSC);// eine ADC-Wandlung
        while (ADCSRA & (1<<ADSC)) {}

        counterTimer++;
        voltage = ADC_GetValue() / 1023.0 * 5.0 * 1000;

        if(counterTimer >= 245)
        {
                USART_TransmitNumber(4, (int)voltage);
                USART_TransmitString(" ");
                counterTimer = 0;
                voltage = 0;
        }
}
```

In der Timer-Interruptroutine wird die oben abgebildete Funktion aufgerufen. Diese führt eine ADC-Wandlung durch und rechnet den eingelesenen 10-bit Wert in eine Spannung um.

$$U_{ADC} = \frac{Value_{ADC} \cdot 5V \cdot 1000}{2^{10} - 1}$$

$$2^{10} - 1 = 1023$$

Ergebnis in mV

Bei jedem ausgelösten Interrupt wird eine Countervariable erhöht. Erreicht sie den Wert 245, dann wird der Spannungswert an den Computer gesendet. Jede Sekunde werden somit die Daten gesendet.

$$Value_{Counter} = \frac{f_{Timer\ 0}}{OCR0A \cdot t_{Delay}} = \frac{62.5kHz}{255 \cdot 1s} = 245.1$$

Bei unserem Projekt wird der Wert gemittelt. Um dies auch bei dieser Arbeit testen zu können, werden folgende Zeilen Code geändert:

```
counterTimer++;
voltage += ADC_GetValue() / 1023.0 * 5.0 * 1000;

        if(counterTimer >= 245)
        {
                voltage /= 245;
```

Die Spannung wird bei jedem Timerinterrupt aufaddiert. Erreicht der Counter den Wert 245 (eine Sekunde verstrichen), so wird der aufaddierte Spannungswert durch 245 dividiert und dem Computer gesendet.

Ergebnisse:

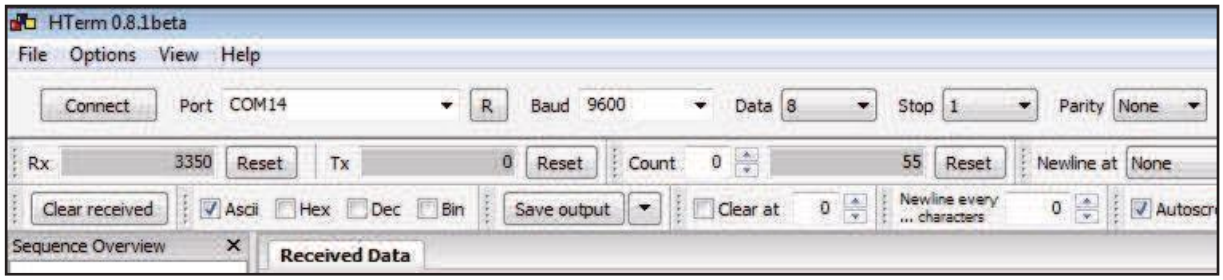


Abb. 13: Einstellungen im Programm HTerm

Das Programm HTerm findet beim Anstecken des Crumb-Moduls den Port selbst (COM-Port 14). Die Bitrate wird auf 9600bits/s eingestellt.
Es werden 8 Datenbits, 1 Stoppbit und kein Paritybit ausgewählt.

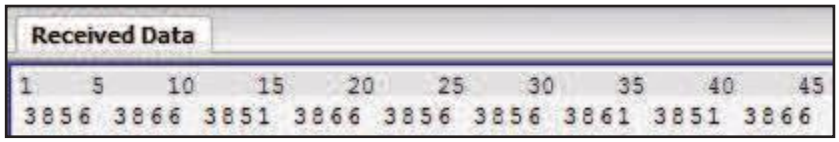


Abb. 14: Störspannung 10mV 1Hz

Der eingelesene Wert ohne Störung betrug 3.861V. Die sinusförmige Störspannung, mit einer Amplitude von 10mV und einer Frequenz von 1Hz, bewirkt eine maximale Abweichung von 10mV (→ 3.851V).

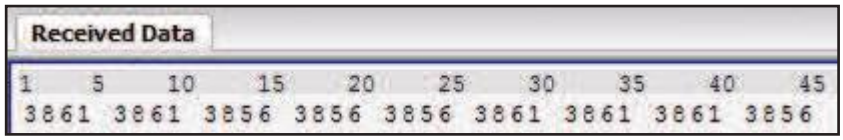


Abb. 15: Störspannung 10mV 4MHz

Bei sehr hochfrequenten Störungen, sprich Hundertkiloherz- bzw. Megaherzbereich, ist die maximale Abweichung geringer. Vermutlich glätten und mitteln parasitäre Kapazitäten/Spulen das Signal. Die eingelesene Spannung ist ab und zu 5mV unter dem richtigen Wert. Dieser minimale Fehler kann sowohl vom Funktionsgenerator, als auch vom ADC des Mikrocontrollers entstehen.

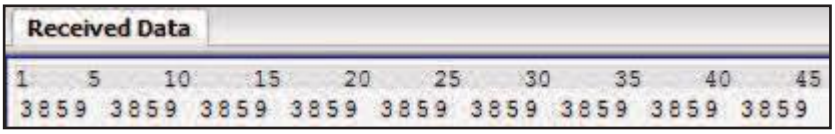


Abb. 16: Störspannung 10mV 1Hz Mittelwertbildung

Aufgrund der Mittelwertbildung entstehen Rundungsfehler bei der Rückrechnung auf den Spannungswert (vorher 3.861V, jetzt 3.859V). Wie aber in der Abbildung 16 zu sehen ist, treten keine Abweichungen mehr auf. Aus diesem Grund wird auch eine Mittelwertbildung in unserem Projekt verwendet. Dort wird der Mittelwert sogar aus 1000 und nicht aus 245 Werten berechnet. Dies wird das Ergebnis nochmals verbessern.

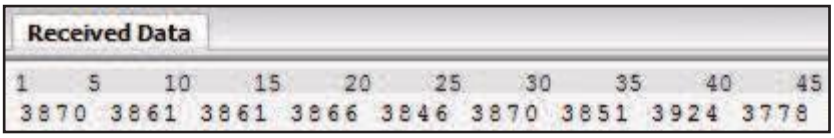


Abb. 17: Störspannung 10-100mV 1Hz

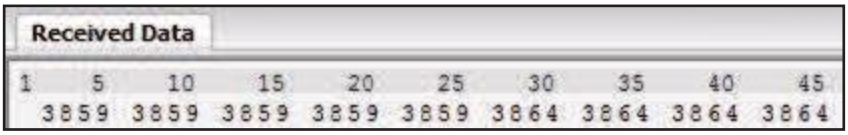


Abb. 18: Störspannung 10-100mV 1Hz Mittelwertbildung

Die Mittelwertbildung ist nicht nur ein Vorteil bei niederfrequenten Signalen, sondern auch bei höheren Amplituden der Störspannung.
Die Amplitude wurde nach etwa 5 Sekunden von 10mV auf 100mV erhöht. In Abbildung 6 ist zu sehen, dass die letzten Messwerte eine enorme Abweichung aufweisen (ca. 100mV).
Bei der Mittelwertbildung (Abbildung 18) treten diese großen Fehler nicht auf. Die eingelesene Spannung unterscheidet sich zum ursprünglichen Signal nur um 5mV.

In nachfolgender Messung wurde der Fehler in Abhängigkeit der Störspannungsamplitude ermittelt.

Die Amplitude wurde zwischen 10mV und 1V variiert. Die Frequenz wurde auf 1Hz eingestellt. Bei einer Messung wurden 20 Messwerte aufgenommen. So konnte sicher gestellt werden, dass Maximum und Minimum jeweils gemessen wurden. Die maximale Abweichung der Spannung wurde ermittelt und der prozentuelle Fehler berechnet. Danach wurde die Störspannungsamplitude wieder verändert.

Es besteht ein linearer Zusammenhang zwischen dem maximalen prozentuellen Fehler und der Amplitude der Störspannung.

U_{Stoer} [V]	$U_{Abweichung}$ [V]	Fehler [%]
0,01	0,01	0,26
0,02	0,02	0,49
0,05	0,05	1,27
0,1	0,10	2,54
0,2	0,20	5,18
0,3	0,30	7,85
0,4	0,38	9,87
0,5	0,44	11,50
0,6	0,60	15,57
0,7	0,70	18,23
0,8	0,81	21,00
0,9	0,88	22,77
1	0,98	25,41

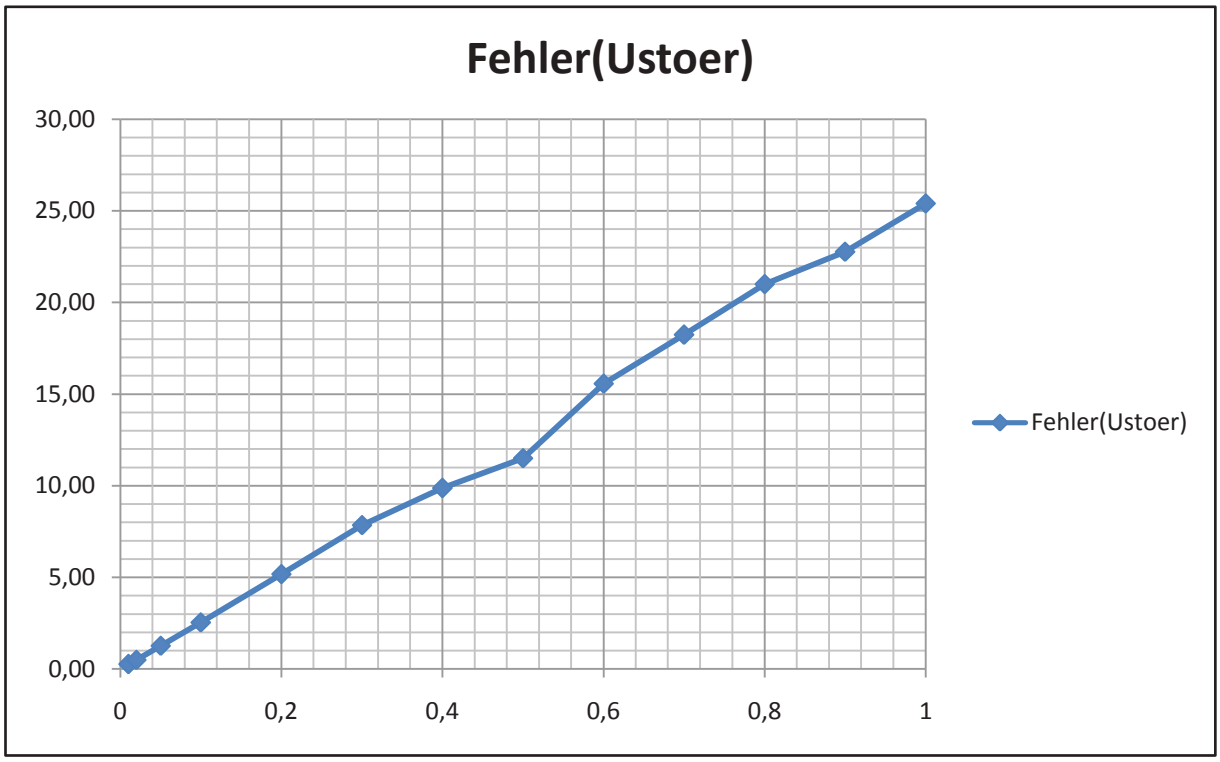


Abb. 19: Diagramm Fehler(Ustoer)

6 Ergebnisse und Konsequenzen

Wie die Messungen zeigen, wirken sich niederfrequente Einstrahlungen enorm auf das Ergebnis aus. Hochfrequente Störungen hingegen bewirken nur einen geringen Fehler, der toleriert werden kann.
Für die Geschwindigkeitsberechnung muss bzw. soll die eingelesene Spannung maximal einen Fehler von 10mV besitzen. Bei geringen Geschwindigkeiten ist somit der Fehler größer, aber ab einer Geschwindigkeit von etwa 100km/h liegt die Abweichung bei wenigen Kilometern pro Stunde.
Weiters zeigen die Messungen, dass die Mittelwertbildung das Ergebnis um einiges verbessert. Auch bei niederfrequenten Störungen ist die Abweichung sehr gering. Ebenfalls ist bei Erhöhung der Störspannungsamplitude die negative Auswirkung auf das Ergebnis nicht so hoch.

Um die bestmöglichen Ergebnisse der TAS-Messung bei unserem Produkt zu erzielen, habe ich mir folgende Lösungen überlegt:

- Abschirmung des gesamten Produktes mittels metallischem Gehäuse
- Mittelwertbildung der eingelesen Spannung
- Drucksensoren direkt auf der Platine platzieren
    - Falls dies nicht möglich ist und sie extern verlegt werden müssen, dann abgeschirmte Kabel zur Übertragung verwenden. Ist der Fehler weiterhin zu groß, ist eine digitale Übertragung die Lösung
      (→ externer A/D-Wandler notwendig).

7 Abbildungsverzeichnis

8 Literaturverzeichnis

- **Datenblatt des Drucksensors:**
  http://docs-europe.electrocomponents.com/webdocs/0ef4/0900766b80ef40fa.pdf, 02.04.2013

- **E-Book über EMV:**
  Franz, Joachim (2010). EMV Störungssicherer Aufbau elektronischer Schaltungen. Vieweg+Teubner Verlag Wiesbaden

- **Artikel zur Geschwindigkeitsberechnung mittels Prandtlsonde:**
  http://www.electro-mation.de/productattachments/index/download?id=47, 10.04.2013

- **Datenblatt des Mikrocontrollers ATmega644PA:**
  http://docs-europe.electrocomponents.com/webdocs/0dc5/0900766b80dc5089.pdf, 10.04.2013

Vertiefende Grundlagenarbeit

Aufzeichnung und Bewertung der Signalverläufe am vorliegenden I²C-Bus

FLINOS
Flight-Info-System

Ausgeführt im Schuljahr 2012/2013

von:

Sebastian Modl 5AHELI

Betreuer:

Prof. Dipl.-Ing. Ing. Karl Heinz Steiner

Salzburg, am 07.05.2013

Inhaltsverzeichnis

1 I2C-Bus Allgemein[1]

Der I2C-Bus wurde in den frühen 80er-Jahren von Philips entwickelt. Er wird benutzt, um eine einfache Kommunikation zwischen ICs, die sich auf derselben Platine befinden, zu ermöglichen. Der Name I2C kommt von „Inter IC“, was so viel bedeutet wie „(Kommunikation) zwischen ICs“. Ein großer Vorteil dieses Bussystems ist, dass es, wie zum Beispiel auch UART, nur 2 Leitungen benötigt. Im Gegensatz zu UART arbeitet I2C jedoch synchron und ist damit nicht an normierte Übertragungsraten gebunden.

Zur Übertragung der Daten benötigt der Bus nur 2 Leitungen, sie werden SDA und SCL genannt. An SCL gibt der Master (steuernder IC) den Takt für die Übertragung an. Es ist auch möglich mehrere Master an einem Bus zu betreiben. SDA wird zum Übertragen der eigentlichen Daten benutzt. Die Leitungen müssen mit einem Pull-Up Widerstand versehen werden, dadurch müssen die angeschlossenen Bausteine die Leitungen nur noch auf GND ziehen oder freigeben.

1.1 Geschichte[2]

Das ursprüngliche System wurde von Philips in den frühen 1980er Jahren entwickelt, um verschiedene Chips in Fernsehgeräten einfach steuern zu können. Seit Mitte der 1990er Jahre wird I²C auch von einigen Wettbewerbern zur Bezeichnung von Philips-kompatiblen I²C-Systemen verwendet, darunter Siemens AG (später Infineon Technologies AG), NEC, STMicroelectronics, Motorola (später Freescale), Intersil, etc.

Atmel führte aus lizenzrechtlichen Gründen die heute auch von einigen anderen Herstellern verwendete Bezeichnung TWI (Two-Wire-Interface) ein, technisch sind TWI und I²C identisch. Allerdings ist das ursprüngliche Patent am 1. Oktober 2006 ausgelaufen, so dass keine Lizenzgebühren für die Benutzung von I²C mehr anfallen. I²C ist auch kein eingetragenes Markenzeichen von NXP Semiconductors, Markenschutz besteht lediglich für das Logo.

Abb. 1: I2C Logo

[1] vgl. http://www.i2c-bus.org (2013)
[2] zitiert nach http://de.wikipedia.org/wiki/I²C (2013),
I2C Logo vgl. http://upload.wikimedia.org/wikipedia/de/9/95/I2c_logo.svg

1992 wurde die erste Spezifikation 1.0 veröffentlicht. Diese ergänzte den ursprünglichen Standard mit 100 kbit/s um einen neuen „schnellen“ Modus mit 400 kbit/s und erweiterte den Adressraum um einen 10-Bit-Modus, so dass statt der ursprünglichen 112 Knoten seitdem bis zu 1136 unterstützt werden. Mit Version 2.0 aus dem Jahr 1998 kam ein „Hochgeschwindigkeitsmodus“ mit max. 3,4 Mbit/s dazu, wobei die Strom- und Spannungsanforderungen in diesem Modus gesenkt wurden. Version 2.1 von 2000 enthält nur kleinere Aktualisierungen gegenüber 2.0. Aktuell gültig ist Version 3.0 von 2007, die einen „extra schnellen“ Modus („Fast-mode Plus“) mit bis zu 1 Mbit/s einführt, der im Gegensatz zum „Hochgeschwindigkeitsmodus“ dasselbe Protokoll verwendet wie die 100-kbit/s- und 400-kbit/s-Modi.

1.2 Hardware[3]

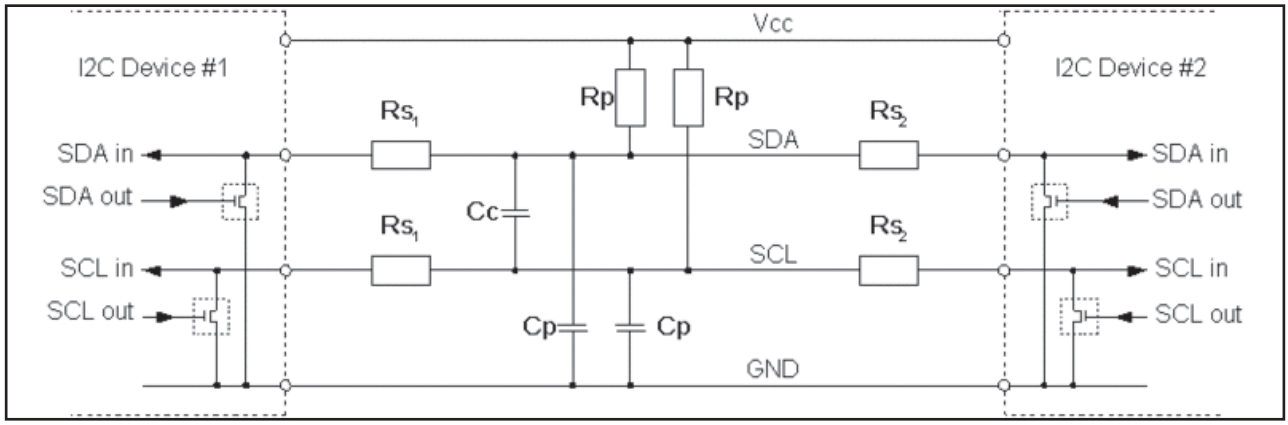


Abb. 2: Ersatzschaltbild I2C-Bus

Die Grafik zeigt eine schematische Darstellung eines I2C-Busses mit 2 angeschlossenen ICs.

Die Widerstände Rs repräsentieren die Leitungswiderstände des Busses. Die parasitären Kapazitäten Cp und Cc sind unerwünscht und sollten layouttechnisch so gering wie möglich gehalten werden, wenn eine schnelle Übertragung gewünscht ist. Weiters sind auch noch Pull-Up Widerstände an SCL und SDA angeschlossen, damit die Übertragung funktionieren kann. Im Normalfall verwendet man Pull-Up Widerstände zwischen 1k und 100k. Da sie in Verbindung mit den Leitungskapazitäten einen Tiefpass bilden, muss darauf geachtet werden, sie nicht zu hoch zu dimensionieren, da die langsamen Anstiegszeiten besonders bei hohen Datenraten zu Problemen führen.

[3] vgl. http://www.i2c-bus.org/typical-i2c-bus-setup/ (2013)

1.3 Übertragung[4]

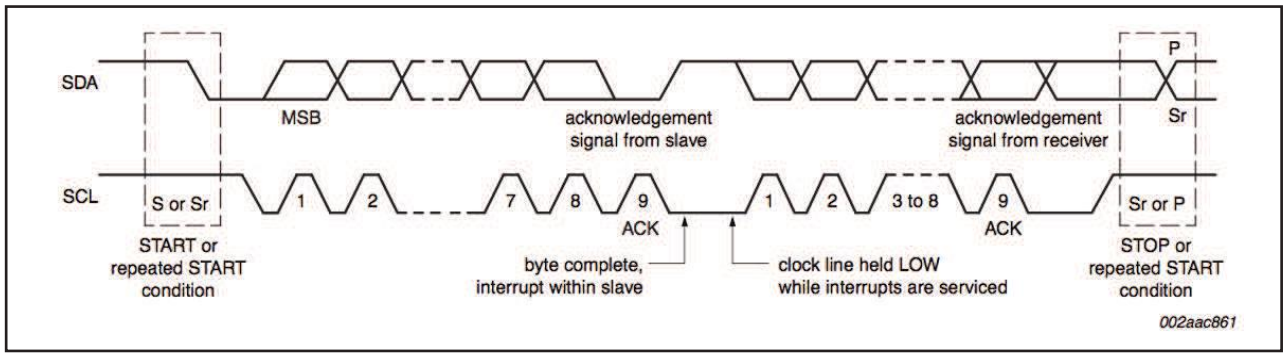


Abb. 3: I2C Übertragung AVR (FIG 6 UM10204)

Vor dem Betrieb des Busses liegt, wegen der Pull-Ups an SDA und SCL, ein HIGH-Pegel an. Vor der Übertragung muss dann der Master eine START-Bedingung ausführen. Dies geschieht indem SDA und erst danach SCL auf LOW gezogen werden. Danach beginnt der Master einen Takt an SCL anzulegen. Wenn der Slave zeitlich nicht mehr mit kommt, hat er die Möglichkeit, SCL aktiv auf LOW zu halten. Dies veranlasst den Master zu warten, bis die Leitung wieder freigegeben wird. Nun werden die Datenbits beginnend mit dem MSB über die SDA-Leitung übertragen. Dabei muss darauf geachtet werden, dass sich SDA nur während den LOW Perioden der Clock-Leitung ändern darf. Nach den 8 übertragenen Datenbits folgt ein Acknowledge-Bit (ACK). Hierbei wird die SDA Leitung vom Master freigegeben und der Slave muss sie zur Bestätigung auf LOW ziehen. Beim Lesen muss der Master ACK selbst anlegen, um weitere Daten vom Slave anzufordern. Nach der Übertragung wird der Bus mit einer STOP-Bedingung wieder freigegeben. Dafür muss zuerst SCL und erst danach SDA auf HIGH gesetzt werden. Wenn man stattdessen sofort wieder eine START-Bedingung schickt (Repeated START), wird der Bus nicht freigegeben und es können keine anderen Master den Bus beanspruchen. Bei einem Fehler hat man mit der STOP-Bedingung auch die Möglichkeit die Übertragung jederzeit abzubrechen.

1.4 Adressierung

Da an einem Bus mehrere Slaves angeschlossen sind, muss mithilfe einer Adressierung zuerst ein Slave zur Kommunikation ausgewählt werden. Beim I2C-Bus handelt es sich um 7-bit Adressen, die bei gekauften ICs manchmal fix vorgegeben sind. Oft ist es aber auch möglich einzelne Bits mithilfe von Pins am Baustein zu setzen.
Von den 128 möglichen Adressen können 112 verwendet werden, die anderen sind für spezielle Zwecke reserviert. Dies reicht für die meisten kleineren Anwendung völlig aus. Es wird nur problematisch, wenn man 2 ICs mit fixen Adressen vom gleichen Typ verwendet.

[4] vgl. I2C-Bus Specification UM10204

1.5 Protokoll[5]

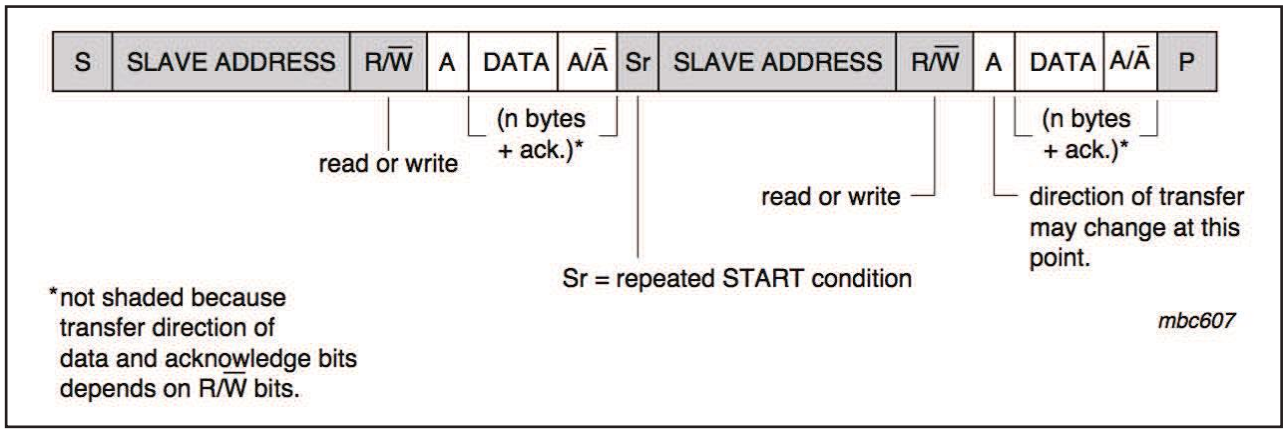


Abb. 4: I2C Protokoll AVR (FIG 13 UM10204)

Nach senden der Start Condition muss zuerst die Adresse des anzusteuernden ICs übertragen werden (7 Bit). Anschließend wird noch ein Bit gesendet, es teilt dem Slave mit, ob vom Master gelesen oder geschrieben wird. Wenn dieses Bit HIGH ist, wird geschrieben, ansonsten gelesen. Am Ende dieser Sequenz meldet sich der angesprochene Slave mit einem Acknowledge-Bit. Nun ist der Slave bereit, und es kann mit der Übertragung der Daten begonnen werden.

1.5.1 Schreiben

Beim Schreiben von Daten werden, nachdem sich der Slave gemeldet hat, nacheinander Datenbytes übertragen. Am Ende jedes Bytes antwortet der Slave mit ACK, sofern kein Fehler aufgetreten ist. Danach hat er die Möglichkeit die Clock-Leitung aktiv auf LOW zu ziehen, bis er bereit für das nächste Bit ist.

1.5.2 Lesen

Zum Lesen gibt der Master SDA frei und gibt wie immer den Takt vor. Der Slave überträgt Byteweise die Daten und der Master antwortet mit ACK, solange er weitere Daten will. Da vor dem Lesen meist eine Speicheradresse des ICs vorgegeben werden muss, ist es wichtig, dass im Betrieb mit mehreren Mastern keine STOP-Condition dazwischen ist (repeated START). Andernfalls könnte ein anderer Master die Speicheradresse dazwischen ändern.

[5] vgl. I2C-Bus Specification UM10204

2 AVR-Software

Der Mikrocontroller ATmega644PA verfügt über eine Hardware I2C Implementierung. Es muss sich nicht mehr um die Übertragung an sich, sondern um die Einstellung der speziellen Register gekümmert werden. Dies erleichtert den softwaretechnischen Umgang mit der Schnittstelle enorm. Außerdem wird wertvolle Rechenleistung gespart und somit das Programm wesentlich beschleunigt.

Um die Hardwareimplementierung nutzen zu können, muss zuerst die TWI Bibliothek eingebunden werden.

```
#include <util/twi.h>
```

2.1 Taktrate[6]

Die Taktrate hängt ab von

- CPU-Frequenz
- TWBR Register
- TWPS Bits (2 Prescaler Bits)

Sie berechnet sich aus dieser Formel:

$$f_{SCL} = \frac{f_{CPU}}{16 + 2^{1+2 \cdot TWPS} TWBR}$$

Somit ergibt sich als maximale Taktfrequenz: $\frac{f_{CPU}}{16}$.

Nun folgt ein Bespiel bei $f_{CPU} = 16MHz,\ TWPS = 1$

fClock entspricht der gewünschten Taktfrequenz des I2C Busses.

```
if(fClock == F400K)
        TWBR = 3;
else if(fClock == F100K)
        TWBR = 18;
else if(fClock == F1M)
         TWBR = 0;
```

[6] vgl. http://www.mikrocontroller.net/articles/AVR_TWI (2013)

2.2 Slave ansprechen

```
SBOOL IIC_start(char address, SBOOL read){
	TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
	while (!(TWCR & (1<<TWINT)));

	if ((TWSR & 0xF8) != TW_START && (TWSR & 0xF8) != TW_REP_START )
	  return EFALSE;
	if(read)
	  address += 1;
	TWDR = address;
	TWCR = (1<<TWINT) | (1<<TWEN);

	while (!(TWCR & (1<<TWINT)));
	if(read){
	  if ((TWSR & 0xF8) != TW_MR_SLA_ACK)
		return EFALSE;
	}else{
	  if ((TWSR & 0xF8) != TW_MT_SLA_ACK)
		return EFALSE;
	}

	return ETRUE;
}
```

2.3 Daten senden

```
SBOOL IIC_sendByte(char byte){
	TWDR = byte;
	TWCR = (1<<TWINT) | (1<<TWEN);

	while (!(TWCR & (1<<TWINT)));
	if ((TWSR & 0xF8) != TW_MT_DATA_ACK)
	  return EFALSE;

	return ETRUE;
}
```

2.4 Daten lesen

```
SBOOL IIC_readByte(char * byte, SBOOL ack){
        if(ack)
          TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);
        else
          TWCR = (1<<TWINT) | (1<<TWEN);
        while (!(TWCR & (1<<TWINT)));
        *byte = TWDR;

        return ETRUE;
}
```

2.5 Bus freigeben

```
SBOOL IIC_stop(){
        TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWSTO);
        TWCR &= ~(1<<TWINT);
        return ETRUE;
}
```

2.6 Realtime-Clock

Address	Register name	BCD format tens nibble				BCD format units nibble			
		Bit 7 2^3	Bit 6 2^2	Bit 5 2^1	Bit 4 2^0	Bit 3 2^3	Bit 2 2^2	Bit 1 2^1	Bit 0 2^0
02H	Seconds	VL	<seconds 00 to 59 coded in BCD>						
03H	Minutes	–	<minutes 00 to 59 coded in BCD>						
04H	Hours	–	–	<hours 00 to 23 coded in BCD>					
05H	Days	–	–	<days 01 to 31 coded in BCD>					
06H	Weekdays	–	–	–	–	–	<weekdays 0 to 6 > [1]		
07H	Months/Century	C	–	–	<months 01 to 12 coded in BCD>				
08H	Years	<years 00 to 99 coded in BCD>							

Abb. 5: Register der Realtime-Clock (Table 5 im Datenblatt)[7]

Da in den Registern für die Zeit auch Bits für andere Zwecke enthalten sind, ist es nicht ganz einfach, ein fehlersicheres Programm zu entwickeln. Viele Tests sind nötig, bis man sich auf die Realtime-Clock Software verlassen kann. Es ist jedoch um Welten einfacher und genauer einen solchen Baustein fertig zu kaufen, als die Zeit am Mikrocontroller selbst zu verwalten.

Die Software wurde für die Realtime-Clock PCF8563 entwickelt.

[7] vgl. http://www.datasheetcatalog.org/datasheet/philips/PCF8563-02.pdf

```
/*
 * RTC.c
 *
 *  Created: 13.09.2012 18:15:54
 *  Author: Sebastian Modl
 */

#include <avr/io.h>
#include "RTC.h"
#include "IIC.h"
#include "SStddef.h"

#define RTC_ADRESS 0xA2

typedef enum
{
	RTC_21TH = 0,
	RTC_20TH = 1
}RTCE_CENTURY;

typedef enum
{
	SECONDS = 2,
	MINUTES = 3,
	HOURS   = 4,
	DAYS    = 5,
	MONTHS  = 7,
	YEARS   = 8
}RTCE_TIMEUNIT;

RTCE_CENTURY RTCcentury;

SBOOL RTCcheckTime(RTCE_TIMEUNIT timeUnit, char val);

SBOOL RTC_init(RTCE_CENTURY century){
	RTCcentury = century;
	return ETRUE;
}

SBOOL RTC_setTime(RTCE_TIMEUNIT timeUnit, char val){
	if(!RTCcheckTime(timeUnit, val))
		return EFALSE;

	val = ((val/10)<<4)|(val-(val/10)*10);

	if(timeUnit == MONTHS)
		val |= (RTCcentury<<7);

	IIC_start(RTC_ADRESS, EFALSE);
	IIC_sendByte(timeUnit);
	IIC_sendByte(val);
	IIC_stop();
	return ETRUE;
}
```

```
SBOOL RTC_getTimeChars(RTCE_TIMEUNIT timeUnit, char* val10, char* val){
	IIC_start(RTC_ADRESS, EFALSE);
	IIC_sendByte(timeUnit);

	IIC_start(RTC_ADRESS, ETRUE);
	IIC_readByte(val, EFALSE);
	IIC_stop();


	*val &= ~(1<<7);
	if(timeUnit >= HOURS && timeUnit < YEARS)
		*val &= ~(1<<6);
	if(timeUnit == MONTHS)
		*val &= ~(1<<5);

	*val10 = (*val)>>4;
	*val &= 0x0F;
	*val10 += 0x30;
	*val += 0x30;

	return ETRUE;
}



SBOOL RTC_getTime(RTCE_TIMEUNIT timeUnit, char* val){
	IIC_start(RTC_ADRESS, EFALSE);
	IIC_sendByte(timeUnit);

	IIC_start(RTC_ADRESS, ETRUE);
	IIC_readByte(val, EFALSE);
	IIC_stop();

	*val &= ~(1<<7);
	if(timeUnit >= HOURS && timeUnit < YEARS)
		*val &= ~(1<<6);
	if(timeUnit == MONTHS)
		*val &= ~(1<<5);

	*val = (((*val)>>4)*10) + ((*val) & 0x0F);

	if(!RTCcheckTime(timeUnit, *val))
		return EFALSE;

	return ETRUE;
}


SBOOL RTCcheckTime(RTCE_TIMEUNIT timeUnit, char val){
		switch (timeUnit){
		case SECONDS:
		case MINUTES:
			if(val < 0 || val > 59)
				return EFALSE;
			break;
```

```
		case HOURS:
			if(val < 0 || val > 23)
				return EFALSE;
			break;
		case DAYS:
			if(val < 1 || val > 31)
				return EFALSE;
			break;
		case MONTHS:
			if(val < 1 || val > 12)
				return EFALSE;
			val |= (RTCcentury<<7);
			break;
		case YEARS:
			if(val < 0 || val > 99)
				return EFALSE;
	}
	return ETRUE;
}
```

3 Hardware

3.1 Layout

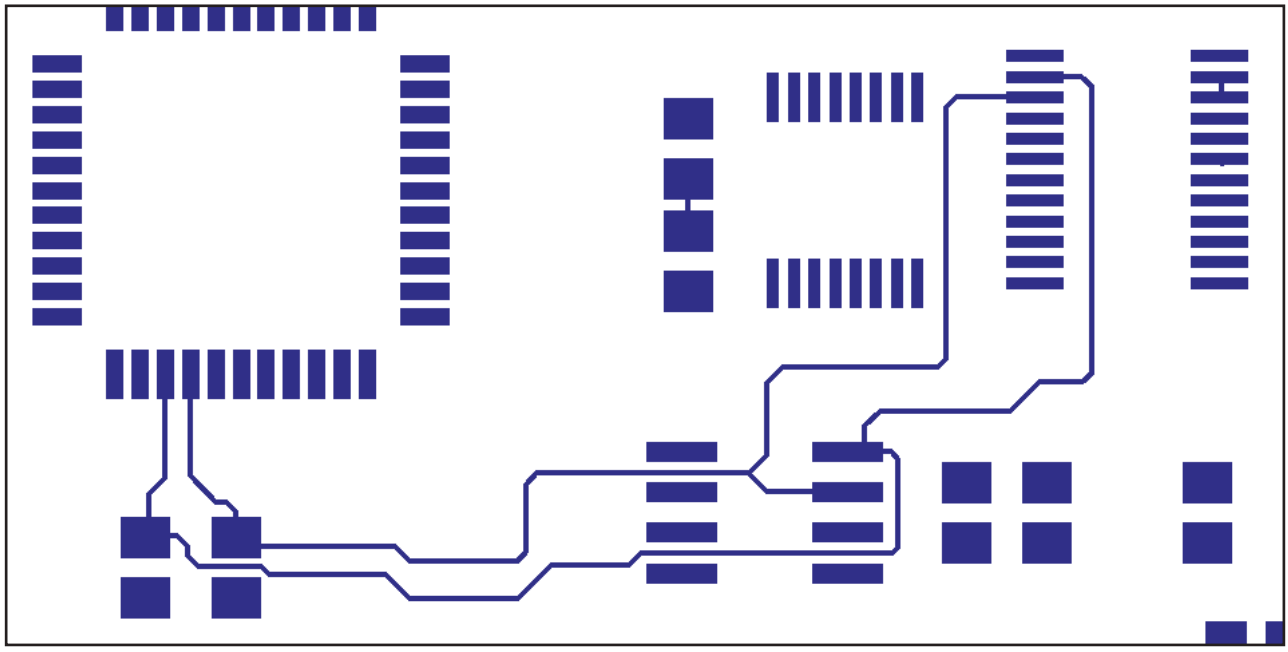


Abb. 6: Layout I2C-Bus ohne Polygon (FLINOS)

Beim Layout der I2C-Verbindung ist darauf zu achten, dass diese so kurz wie möglich ist. Außerdem müssen rechte Winkel vermieden werden, da diese schlechte Eigenschaften bei hohen Frequenzen aufweisen. Je besser das Layout ist, desto höhere Geschwindigkeiten können erreicht werden.

3.2 Pull-Up Widerstände[8]

Die geeignete Wahl der Widerstände ist gerade bei Applikationen, die wenig Strom verbrauchen sollen, wichtig. Für die genaue Berechnung ist es nötig, die Kapazität der Leitungen zu kennen. Dies geschieht am besten experimentell. Es wird ein Widerstand gewählt und über die Anstiegszeit eines Rechtecksignals die Zeitkonstante der Leitung ermittelt. Die Zeit, nach der 63% des Maximalwertes erreicht ist, ist Tau.

$$t_{63\%} = \tau = R \cdot C \rightarrow C = \frac{\tau}{R}$$

Für das obige Layout wurde mit 1k5 Widerständen eine Zeitkonstante von 45ns gemessen. Daraus berechnet sich eine Leitungskapazität von ca. 30pF.

Die Widerstände müssen mindestens so groß sein, dass der Strom nicht höher als 3mA ist. Somit errechnet sich, laut Datenblatt der minimale Widerstandswert folgendermaßen.

$$R_{min} = \frac{Vcc - 0.4V}{3mA}$$

Im Falle von 5V müssen die Widerstände mindestens 1500 Ohm haben.

Da in den meisten Fällen ein Strom von 3mA völlig unnötig ist, werden die Pull-Ups in der Regel so hoch wie möglich gewählt. Der Stromverbrauch wird somit gesenkt. Wenn die Widerstände zu hoch gewählt werden, können die Bausteine die Signale nicht mehr erkennen.

Im Datenblatt des Mikrocontrollers wird der maximale Widerstandswert wie folgt angegeben.

$\boldsymbol{f_{CLK} \leq 100kHz}$	$\boldsymbol{f_{CLK} > 100kHz}$
$R_{max} = \frac{1000ns}{C_P}$	$R_{max} = \frac{300ns}{C_P}$

[8] vgl. Datenblatt ATmega644PA

3.3 Simulation

Sobald die Zeitkonstante der Leitung bekannt ist, kann die Übertragung simuliert werden. Es ist somit möglich, verschiedene Frequenzen bzw. Widerstände zu testen.
Es reicht, die Leitungen als Tiefpass 1. Ordnung anzunehmen. Mit dem Programm BORIS ist es einfach ein solches System zu simulieren.

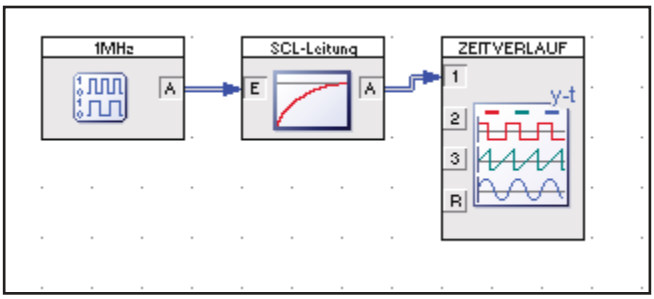


Abb. 7: Schaltung in BORIS

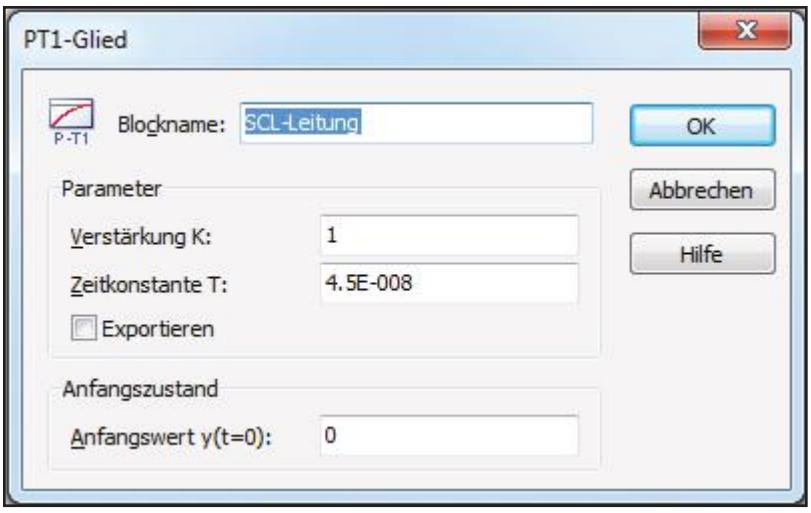


Abb. 8: Einstellung des Tiefpasses in BORIS

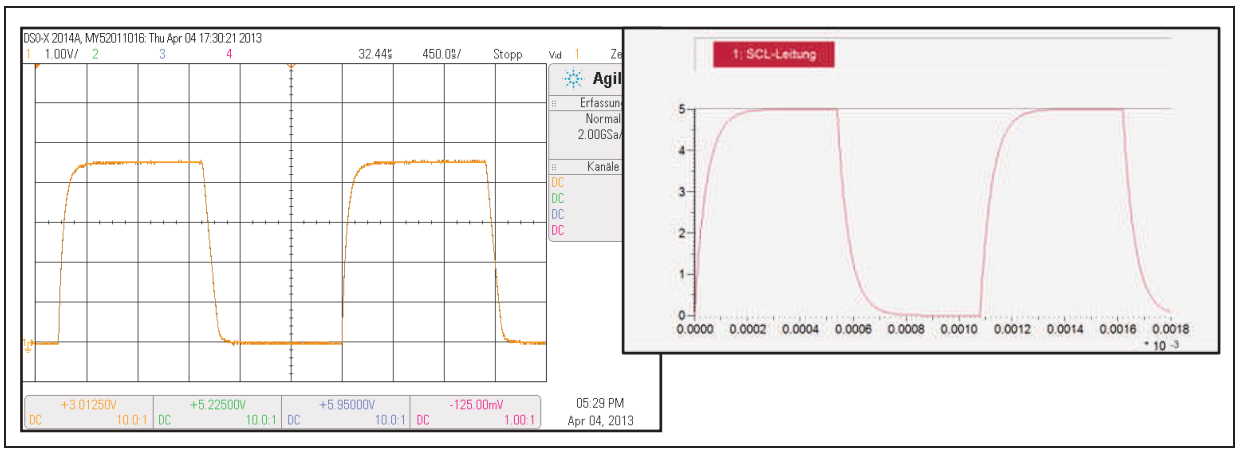


Abb. 9: Vergleich Simulation und Messung bei 400kHz

Folgende Grafiken zeigten, wie sich unterschiedliche Widerstandswerte auf die Signalverläufe auswirken (bei 1MHz und 400kHz). Man erkennt deutlich, dass der 10k Widerstand bei 1Mhz nicht mehr geeignet ist.

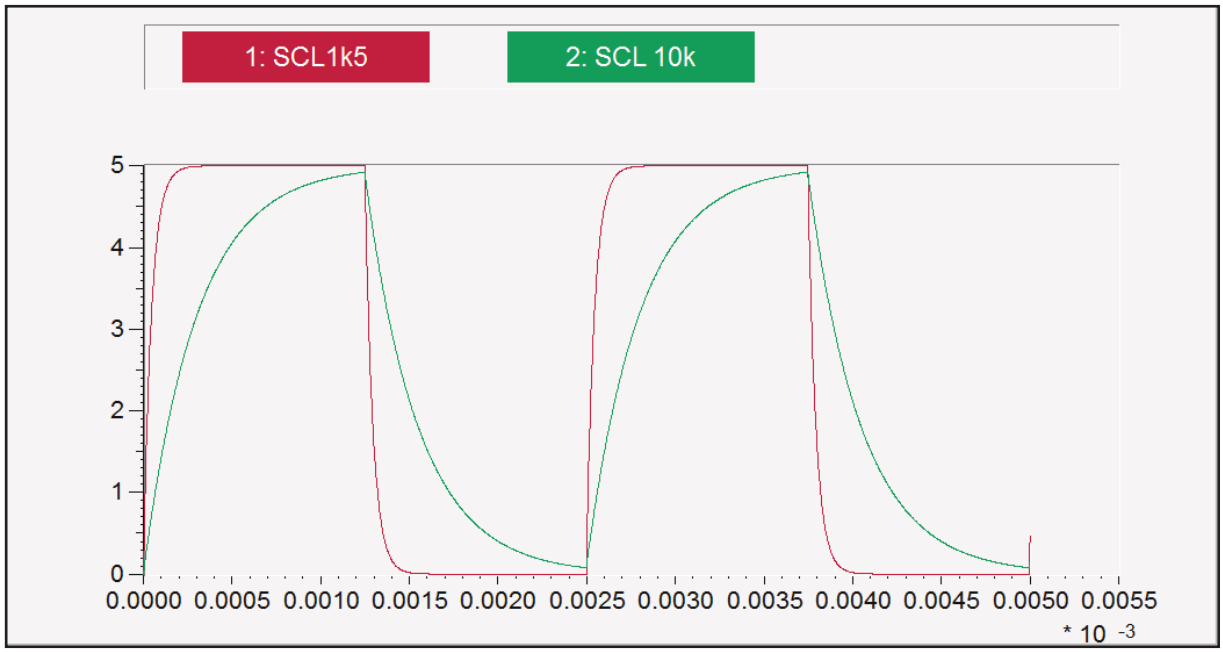


Abb. 10: Simulation 400kHz BORIS

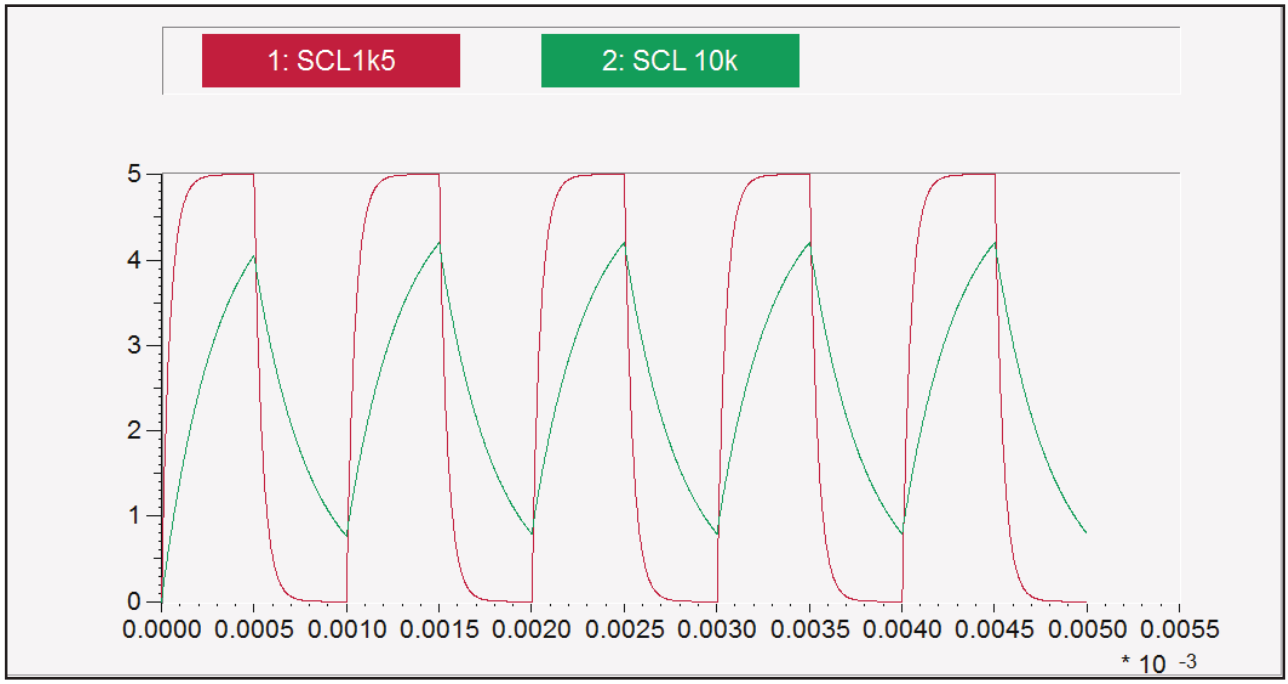


Abb. 11: Simulation 1MHz BORIS

4 Analyse

4.1 Signale

Bei der Analyse der Signale am Oszilloskop fällt auf, dass die Signalverläufe bei 400kHz noch sehr gut sind. Dies liegt an den kleinen Pull-Up Widerständen von 1500 Ohm. Es ist ratsam Tastköpfe zu verwenden, da ansonsten die Messungen verfälscht werden.

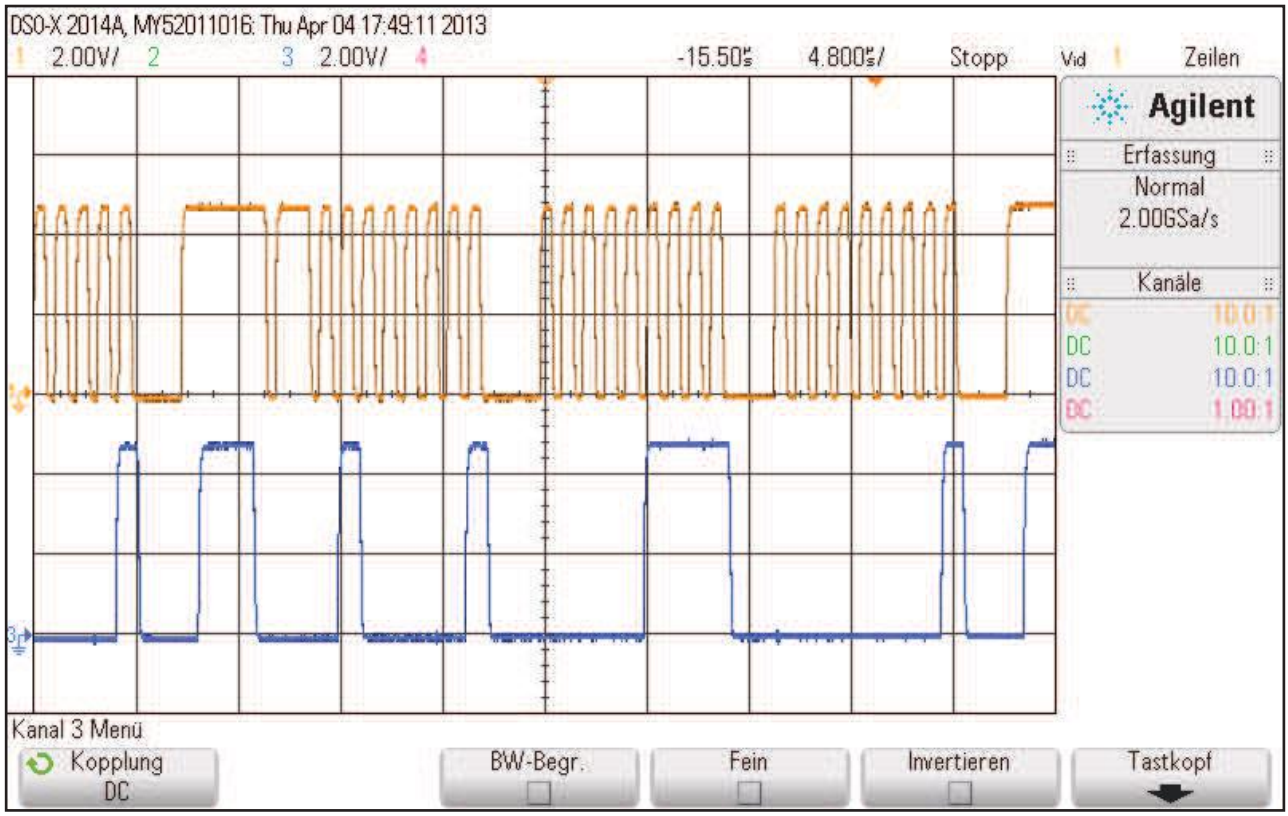


Abb. 12: I2C Verläufe am Oszilloskop

4.2 Acknowledge

Beim Acknowledge gibt der Mikrocontroller SDA frei und wartet, bis der Slave die Leitung wieder auf GND zieht.

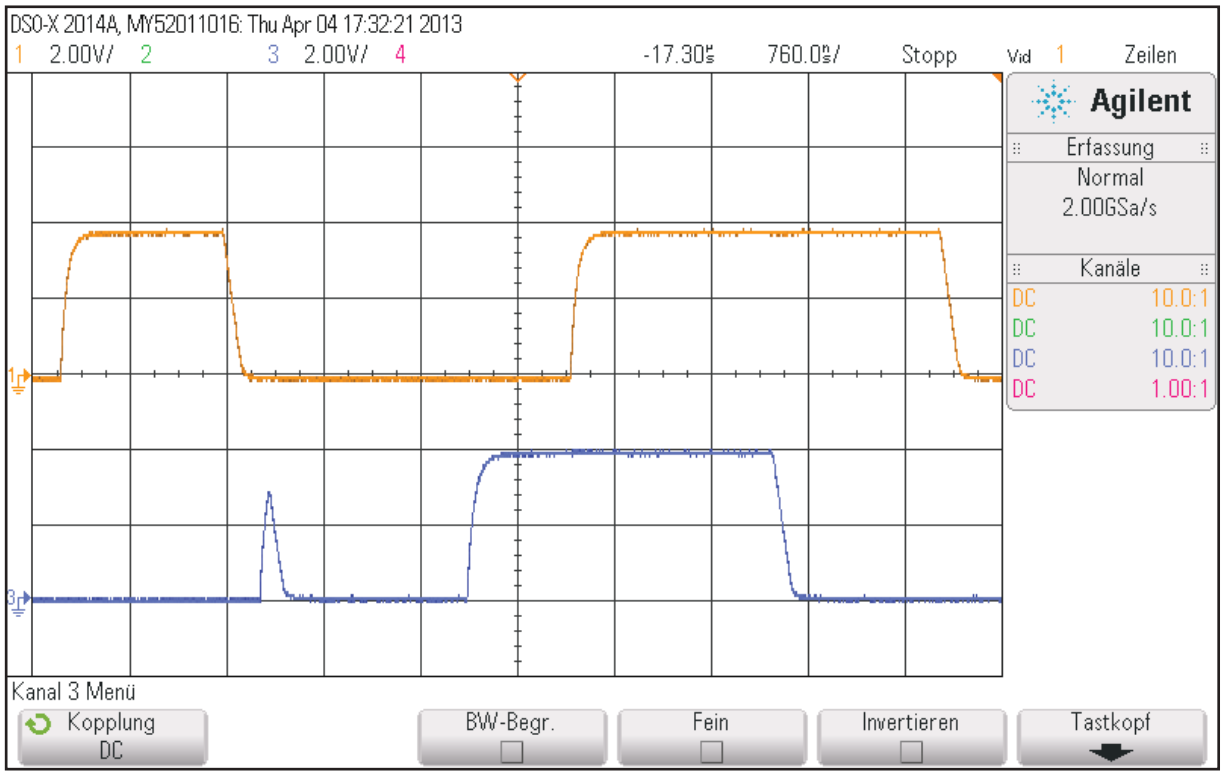


Abb. 13: Acknowledge bei 400kHz

Man erkennt sehr deutlich, dass der Slave schon antwortet, noch bevor das Potenzial der Leitung 5V erreicht hat. Schon bei ca. 3V liegt die Leitung wieder auf Masse.

4.3 Taktfrequenzen

Die drei Bilder auf der nächsten Seite zeigen das Verhalten der Clock-Leitung bei 100, 400 und 1000 kHz. Es wurde ein Pull-Up von 1500Ohm verwendet und die Leitungskapazität beträgt 30pF. Die Leitung kann problemlos für alle diese Frequenzen verwendet werden, bei höheren Frequenzen würde es schon kritisch werden.

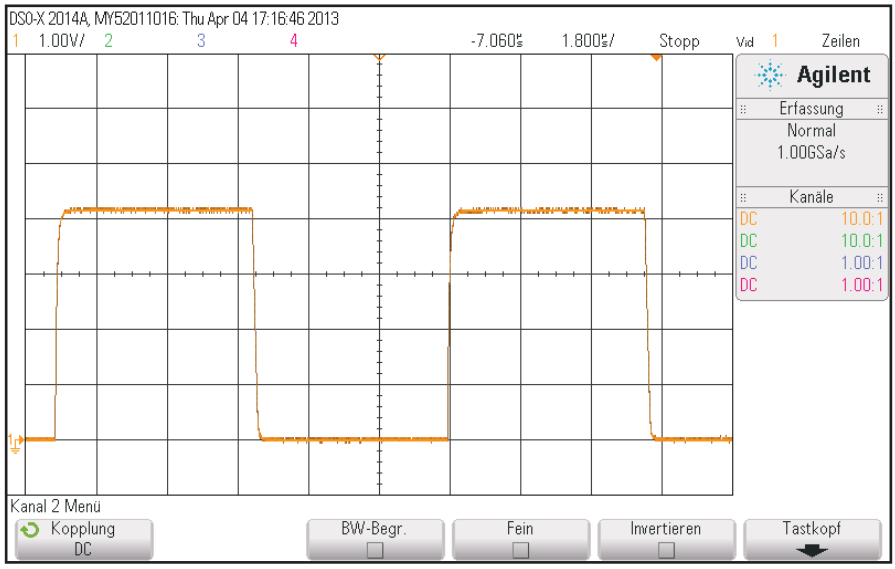


Abb. 14: SCL bei 100kHz

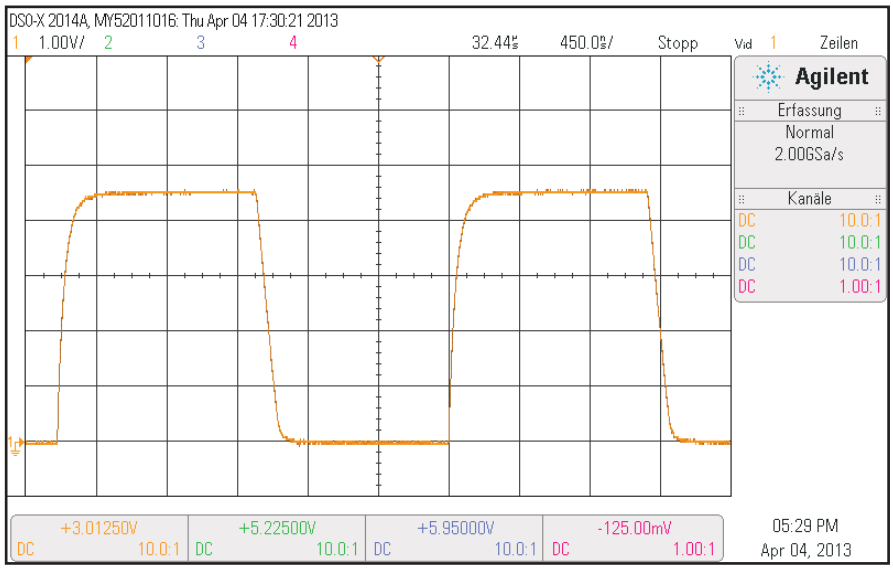


Abb. 15: SCL bei 400kHz

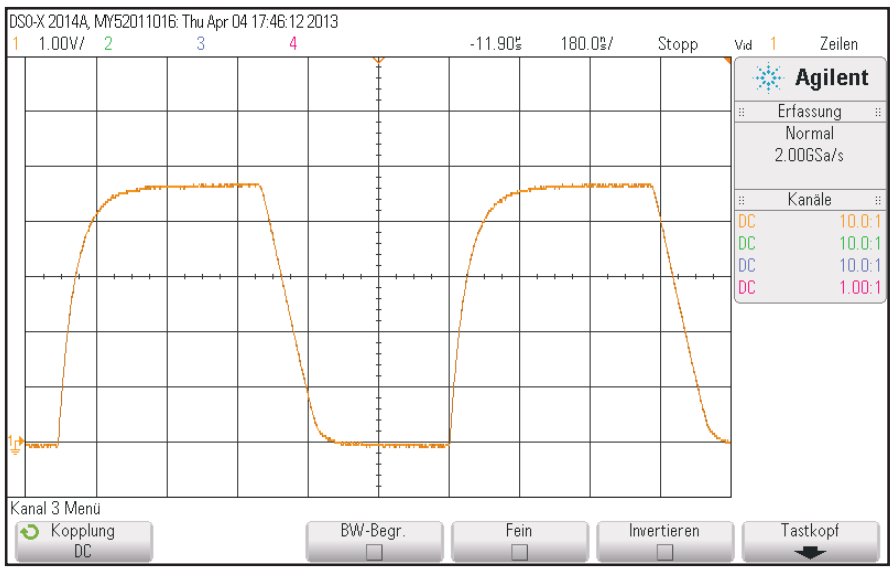


Abb. 16: SCL bei 1MHz

4.4 Zu hoher Pull-Up bei SCL

Wenn ein zu hoher Pull-Up Widerstand bei der SCL-Leitung verwendet wird, ist eine schnelle Übertragung nicht mehr möglich. Der Mikrocontroller erkennt jedoch, dass die Spannung an der Leitung nur sehr langsam steigt und wartet, bis ein gewisser Wert erreicht ist. Wird der Pull-Up an SDA zu hoch, kann nicht darauf reagiert werden und die Übertragung funktioniert nicht mehr. Im folgenden Bild ist zu sehen, wie die Frequenz auf ca. 30kHz gedrosselt wird. Es wurde ein Widerstand von ca. 500 Kiloohm verwendet.

Blau: SCL

Gelb: SDA

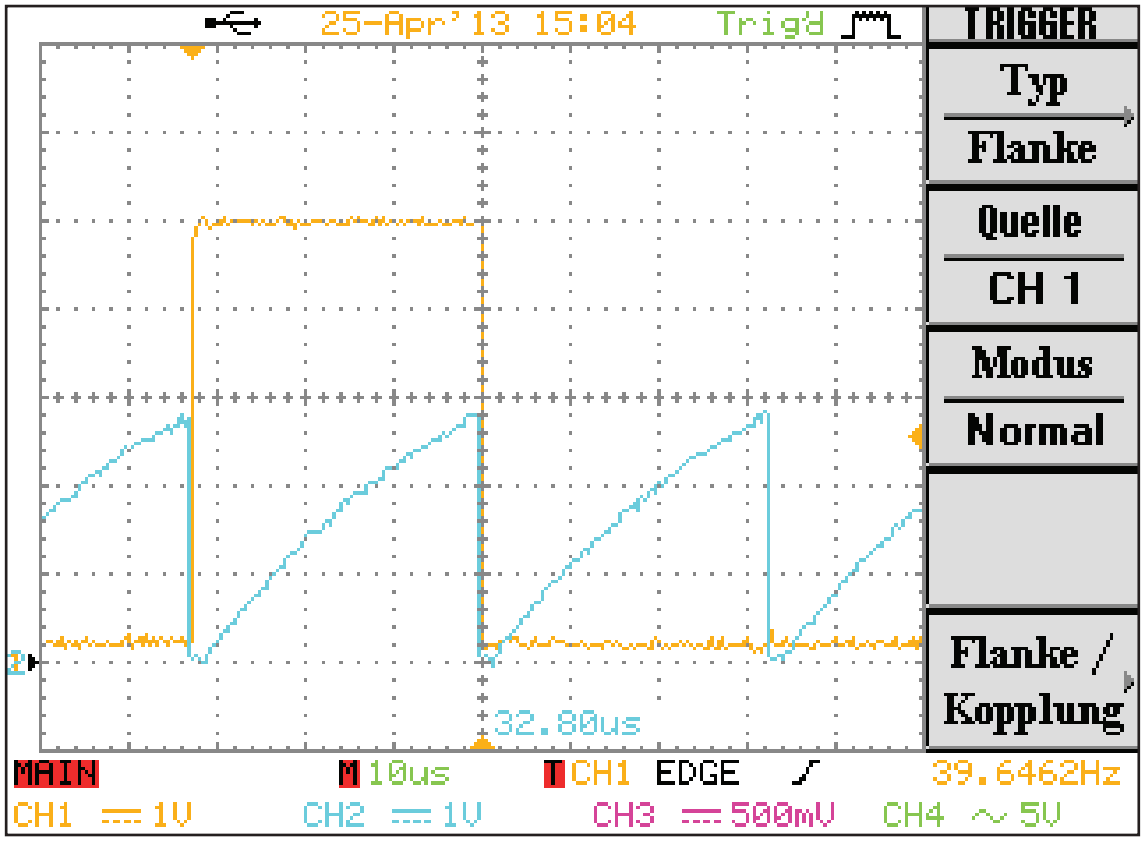


Abb. 17: Verlangsamte Clock-Frequenz

5 Ergebnisse und Konsequenzen

5.1 Pull-Ups

Da für unsere Anwendung 3mA keine Belastung darstellen (über 200mA Betriebsstrom gesamt), können problemlos 1k5 Pull-Up Widerstände verwendet werden. Dies kommt der Störsicherheit zugute.

5.2 Taktrate

1MHz ist ohnehin zu schnell für eines der Bauteile, kommt also nicht in Frage.

400kHz wurde noch nicht ausreichend im Betrieb getestet, um Aussagen über die Fehleranfälligkeit zu machen. Solange es nicht unausweichlich wird, werden wir auf diese Geschwindigkeit verzichten.

Wie die Messungen zeigen, funktioniert die Übertragung bei 100kHz perfekt. Das Rechtecksignal wird überhaupt nicht verfälscht.

Die wichtigste Anforderung im Flugzeugt ist Ausfallsicherheit. Aus diesem Grund haben wir uns dazu entschlossen so sicher wie möglich zu entwickeln. Der Bus wird also bei einer Frequenz von 100kHz betrieben.

6 Abbildungsverzeichnis

7 Literaturverzeichnis

Datenblatt ATmega644PA:

http://www.atmel.com/Images/doc2593.pdf, 05.04.2013

- **Datenblatt Realtime-Clock PCF8563:**
  http://www.datasheetcatalog.org/datasheet/philips/PCF8563-02.pdf, 05.04.2013

- **I2C-bus Specification UM10204:**
  http://www.nxp.com/documents/user_manual/UM10204.pdf, 01.05.2013

- **Website über I2C:**
  http://www.i2c-bus.org, 10.04.2013

- **Artikel AVR TWI:**
  Alexander, Starke
  http://www.mikrocontroller.net/articles/AVR_TWI, 05.04.2013