Hochschule Bremen

Fakultät 5 Natur und Technik

Bachelorthesis

im Studiengang Elektrische Energietechnik, B. Eng.

Entwicklung eines planaren Miniaturautopiloten

Eingereicht von: Benjamin Forke
Matrikel Nummer: 173548

Eingereicht am: 22. Dezember 2010

1. Prüfer/Betreuer: Prof. Dr.-Ing. Heinrich Warmers

2. Prüfer: Prof. Dr.-Ing. Karsten Dünte

Eidesstattliche Erklärung

Hiermit erkläre ich unter Eides statt, dass ich die vorliegende Bachelorthesis zum Thema

„Entwicklung eines planaren Miniaturautopiloten“

selbstständig verfasst und keine weiteren als die aufgeführten Hilfsmittel verwendet habe, dass Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche gekennzeichnet sind und dass diese Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegt wurde.

Bremen, den 22. Dezember 2010

_____

Benjamin Forke

Danksagung

An dieser Stelle möchte ich mich bei den Personen bedanken, die mich bei der Fertigstellung dieser Arbeit unterstützt haben.

Herrn Prof. Dr.-Ing. Heinrich Warmers für Aufgabenstellung, die Betreuung und die Unterstützung während der Arbeit.

Herrn Prof. Dr.-Ing. Karsten Dünte für die Übernahme der Funktion des Zweitprüfers.

Herrn Oliver Riesener für die Unterstützung während der Arbeit in Hard- und Softwarefragen.

Herrn Roman Szczuka für die Unterstützung und Hilfestellung beim Bestücken und Löten.

Herrn Dimitri Nowik für die vielen aufbauenden Stunden und den Kaffee.

Meinen Eltern für die immerwährende Unterstützung während des Studiums und der vorherigen Schulzeit.

Da ich den Gedanken an freie und für alle kostenfrei nutzbare Software fördern möchte, stellen ich, wie das Paparazzi-Team, alle entwickelten Schaltpläne, Layouts, Programme und Dokumentation unter die GPL.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Diese Arbeit beschreibt die Entwicklung, Aufbau und Inbetriebnahme eines Miniaturautopiloten für den Einsatz auf Schwebe- und Flächenfluggeräten. Eine Aufgabe eines Autopiloten ist es, die Lage des Fluggeräts im Raum jederzeit bestimmen zu können. Hierzu sind verschiedene Sensoren erforderlich, die Winkelgeschwindigkeit und die Beschleunigung messen und als analoge Messwerte an einen Mikrocontroller liefern. Der Mikrocontroller verarbeitet die Messwerte und gibt entsprechende Steuersignale an die Drehstromsteller bei Schwebefluggeräten aus, die die einzelnen Drehzahlen der Motoren stellen. Bei Flächenfluggeräten werden die Steuersignale an die Servos ausgegeben.
Zur Erkennung der Lage im Raum einige Messdaten unbedingt erforderlich. Zu den unbedingt nötigen Messgrößen gehören die Höhe, die Beschleunigungskräfte in allen Richtungen und die Winkelgeschwindigkeit um alle Achsen. Neuartige dreiachsige Sensoren sind heutzutage sehr klein und werden als MEMS (Micro Electro Mechanical System) Sensoren bezeichnet. Sie können platzsparend verbaut werden, was die Entwicklung eines Miniaturautopiloten auf einer Leiterplatte erst ermöglicht. Die Sensoren sind in den Maßen 4x4mm, bei einer Höhe von 1,5mm, erhältlich. Ihre Vorgänger hatten noch Abmessungen von 7x7x4mm und konnten nur eine Achse Messen. Für jede der zu messenden Achsen wurde jeweils eine Leiterplatte benötigt. Zur Bestimmung der Lage im Raum ist es möglich, verschiedene Sensoren zu verwenden. In der Vergangenheit kamen für die Lagebestimmung Infrarotsensoren zum Einsatz, die die Lage über die Temperaturdifferenz ermitteln. Infrarotsensoren arbeiten jedoch sehr unzuverlässig, da sie stark von den Witterungsbedingung in der Umgebung und der Umgebung an sich abhängen. Ein Einsatz in Gebäuden ist ebenso nicht möglich. Daher sind Gyroskope und Beschleunigungssensoren die erste Wahl. Sie funktionieren unabhängig von der Witterung, können innerhalb von Gebäuden eingesetzt werden und sind zudem kleiner und somit platzsparender.
Bei Schwebefluggeräten kann auf den Einsatz von Magnetfeldsensoren verzichtet werden, da Richtungsänderungen meist sehr langsam durchgeführt werden, im Gegensatz zu Flächenfluggeräten. Bei diesen wird zur genaueren Gierwinkelbestimmung ein Magnetfeldsensor benötigt. Da der zu entwickelnde Autopilot auf beiden Fluggeräten einsetzbar sein soll, ist in jedem Fall ein Magnetfeldsensor vorgesehen. Eine Korrektur der Lage ist aber erforderlich. Auch unterscheiden sich die Konfigurationen von Beschleunigungssensor und Gyroskope je nach Art des Fluggerätes. So ist bei einem Schwebefluggerät ein kleiner Messbereich der Beschleunigung für die statische Lagebestimmung (im Schwebeflug) von Nöten, da Schwebefluggeräte träge Systeme sind. Bei einem Einsatz auf Flächenfluggeräten ist ein großer Messbereich erforderlich. Hier können wesentlich höhere Beschleunigungskräfte auftreten, insbesondere im Kurvenflug. Mithilfe neuartiger Sensoren ist die Konfiguration zwischen großem und kleinem Messbereich möglich. Andere Sensoren haben für unterschiedliche Messbereiche verschiedene Ausgänge, die man beide zur gleichen Zeit nutzen kann.
Zum Erfassen der Höhendaten wird ein Luftdrucksensor eingesetzt. Mit Luftdrucksensoren lassen sich über hochauflösende 24 Bit Analog-Digital-Umsetzer die aktuelle Höhe sehr präzise bestimmen. Anders ist es bei GPS Systemen, mit denen ebenfalls die Höhe ermittelt werden kann. Die Toleranz bei Höhenmessungen mit einem GPS System beträgt

sieben bis acht Meter und ist somit zu ungenau für den Einsatz auf Flächen- bzw. Schwebefluggeräten.

2 Aufgabenstellung

Aufgabenstellung Bachelorthesis Herrn Benjamin Forke Matr. Nummer 173548

Entwicklung eines planaren Miniaturautopiloten

An der Hochschule Bremen wurde ein für automatisierte Mikrofluggeräte universaler Autopilot mittels zweier Leiterplatten (Mikroprozessor- und Sensorleiterplatte) realisiert und erfolgreich erprobt. Durch Fortschritte bei der mikromechanischen Drehraten- und Magnetfeldsensoren ist es möglich die Funktionalität auf einer planaren Leiterplatte zu realisieren und dabei Kosten, Gewicht und Bauraum einzusparen. Herr Forke erhält im Rahmen seiner Bachelorthesis die Aufgabe einen planaren Miniaturautopiloten mit folgenden Eigenschaften zu entwickeln und zu erproben.

- Abmessung: Breite 30mm, Länge 58mm, Höhe 10mm
- Externer Anschluss für GPS-Modul
- Magnetfeldsensor HMC5348
- ST Drehratensensoren
- Beschleunigungssensor ADXL335
- Mikroprozessor LPC2148
- Dezimalzähler für 8 Servoausgänge
- Luftdrucksensor, Stromausgang mit 24 Bit ADC (SPI-Interface)
- Stromversorgung mittels Schaltregler
- 3,3V Stromversorgung (800mA)
- Rauscharme 3,3V Analogstromversorgung
- Steckverbindung für SPI- und I^2C Interface, USB, Analogeingänge, Funkmodul, RC-Empfänger, Stromversorgung, Kamera, Schaltausgänge
- 2 LEDs
- Rücksetztaster
- 2 I^2C Schnittstellen (Magnetsensor und Drehstromsteller getrennt)
- Pegelumsetzer von 3,3V auf 5V des I^2C Interface

Als Nachweis für die Funktionsfähigkeit soll der fertige Autopilot in einem bestehenden Quadrokopter eingesetzt und ein Testflug absolviert werden. Die Treiberprogramme für den Magnetsensor sind neu zu entwickeln.

3 Funktionsfestlegung

Zu Beginn der Bachelorthesis wurden mit Prof. Dr. Warmers die Anforderungen und Realisierungsmöglichkeiten festgelegt. Da es sich um einen planaren Autopiloten handelt, steht nur eine Platine zur Verfügung, auf der sämtliche Sensoren und Bauteile untergebracht werden müssen. Aus diesem Grund ist es sinnvoll die Bauteile so klein wie möglich zu wählen.
Für den Entwurf der Platine standen mehrere, verschiedene Sensoren zur Auswahl, deren Anforderungen nachfolgend aufgelistet sind. Für alle Sensoren gilt eine möglichst kleine Bauform.

- Drucksensor
  - Zur Flughöhenmessung ausreichende Auflösung des Drucks
  - Versorgungsspannung soll mit 5V möglich sein
- Kompass
  - Geringe Messabweichung des Magnetfeldes in jeder der drei Achsen
  - I^2C Schnittstelle zur Kommunikation mit dem Mikrocontroller
- Beschleunigungssensor
  - Großer Messbereich für Flächenflug, kleiner Messbereich für Schwebeflug
  - Dreiachsiger Sensor
  - Umschaltbarer Messbereich
- Gyroskope (Drehratensensoren)
  - Wählbarer Messbereich für Langsam- und Schnellflug
  - Geringe Störempfindlichkeit des Messsignals gegen Vibrationen
  - Messbereiche bis $\pm 1200°/s$ und $\pm 300°/s$
- Analog-Digital-Umsetzer
  - Zwei separate ADC für Drucksensor und Beschleunigungssensor, Gyroskope
  - 24 Bit für Luftdruckmessung
  - 16 Bit für Beschleunigungs- und Drehratenmessung
  - Versorgungsspannung soll 5V betragen
  - Ohne 16 Bit ADC soll der Mikrocontroller interne 10 Bit Analog-Digital-Umsetzer nutzbar sein
- Mikrocontroller
  - 2x UART
  - 2x SPI

  - 2x I^2C
  - Freie ADC Eingänge
  - JTAG Interface
  - Kompatibel zu anderen Flugsystemen (Paparazzi Tiny)

- Spannungsversorgung
  - Schaltregler

    Ausgangsstrom 2A

    Eingangsspannungsbereich 6V bis 16V
  - Linearregler

    3,3V am Ausgang

    0,8A bis 1A Ausgangsstrom

    Rauscharm bei Analogspannungsversorgung

4 Sensorauswahl

4.1 Magnetsensor

Der Magnetsensor wurde nach Aufgabenstellung ausgewählt. Dieser dreiachsige Sensor HMC5843 von Honeywell ist ein aktueller MEMS Sensor, der eine planare Leiterplattenentwicklung erlaubt.Er kann liegend montiert werden, da seine drei integrierten Sensoren in einem 90° Winkel versetzt messen Der Sensor weist mit 4x4x1,3mm eine kompakte Bauform auf. Die Daten am Ausgang werden über eine I^2C Schnittstelle an den Mikrocontroller übertragen. Die Auswahl dieses Sensors ist in den oben aufgeführten Punkten begründet. Auch die guten Erfahrungen in früheren Projekten trägt zur Auswahl bei. Ein Nachteil ist, dass laut Datenblatt [D2] keine ferromagnetischen Bauteile in einem 3mm Umkreis um den Sensor liegen dürfen. Da viele Kondensatoren nickelhaltig sind, muss dies beim Positionieren der Bauteile unbedingt beachtet werden.

4.2 Drucksensor

Der Drucksensor soll den absoluten Luftdruck erfassen. Der Luftdruck stellt eine wichtige Größe zur Höhenberechnung im Raum dar. Laut Aufgabenstellung wird der Sensor MPXH6115 von Freescale Semicondutors ausgewählt. Er besitzt einen Messbereich von 15kPa - 115kPa. Mit der Barometrische Höhenformel

$$h = \frac{p_0}{\rho_0 \cdot g} \cdot ln\frac{p_0}{p} \tag{4.1}$$

lässt sich nun mit dem oben genannten Messbereich der messbare Höhenunterschied errechnen. Dieser beträgt 16270m. Wobei p der relative Druck ist, g die Schwerebeschleunigung, p_0 der Luftdruck an der Erdoberfläche und ρ_0 die Dichte der Luft an der Erdoberfläche sind.
Die Höhe soll über einen 24Bit ADC ausgewertet werden, dadurch erhält man eine Auflösung der Höhe von wenigen Zentimetern. Der Luftdrucksensor ist in einem kleinen SSOP Gehäuse, in den Abmessungen 7,62x7,62 mm erhältlich.

4.3 Gyroskope (Drehratensensoren)

Die Drehratensensoren sollen nach Aufgabenstellung vom Hersteller ST Microelectronics sein. Da der Autopilot für Flächen- und Schwebefluggeräte einsetzbar sein soll, ist ein konfigurierbarer Messbereich erforderlich. Er soll von einem großen Messbereich für Flächenfluggeräte auf einen kleinen für einen Schwebeflug umschaltbar sein. Die beiden Sensoren LPR530AL und LY530ALH besitzen Messbereiche von jeweils ±300°/s und ±1200°/s. Dies sind Messbereiche, die für einen erfolgreichen Flug ausreichend sind.
Als zweite Variante des Autopiloten sollen Drehratensensoren des Herstellers InvenSense zum Einsatz kommen. Die Wahl fiel auf die Sensoren IDG500 und ISZ500. Diese Sensoren besitzen einen Messberecih von jeweils ±110°/s und ±500°/s und sind ebenso in ihrem Messbereich umschaltbar.

4.4 Beschleunigungssensor

Zur Messung der Beschleunigung soll der Sensor ADXL335 von Analog Devices verwendet werden. Er misst die Beschleunigung um alle drei Achsen. Mit einem Messbereich von ±3g eignet er sich für Flächen- und Schwebefluggeräte. Ein weiteres Kriterium ist die Temperaturdrift. Sie beträgt ±0,01%/°C. Ebenso ist die Nichtlinearität eine wichtige Aussage zur Qualität des Sensors. Sie beträgt ±0,3%. Der 0g Offset vs. Temperature beträgt ±1mg/°C. Dies ist zwar ein sehr hoher Wert im Gegensatz zu anderen Sensoren, jedoch wurden mit diesem Sensor bereits gute Erfahrungen erzielt.
Im späteren Verlauf der Bachelorthesis wurde die zweite Variante des Autopiloten mit einem anderen Beschleunigungssensor ausgestattet. Hier fiel die Wahl auf den LIS344ALH von ST Microelectronics. Auch dieser ist ein dreiachsiger MEMS Sensor. Er hat den Vorteil, dass zwischen einem ±2g und einem ±6g Messbereich umgeschaltet werden kann. Dies ist bei der Wahl zwischen Flächen- und Schwebeflug vorteilhaft. Kriterien sind auch hier die, die schon beim ADXL335 erwähnt sind. Der LIS344ALH weist eine Nichtlinearität von ±0,5% auf, einen 0g Offset vs. Temperature von ±0,4mg/°C und die Temperaturdrift beträgt ±0,01%/°C.

5 Schaltungsentwurf

In diesem Kapitel werden sämtliche Halbleiter und Sensoren und deren externe Beschaltung beschrieben. Es wird auf die Dimensionierung der Beschaltung und die Verschaltung der Bauteile miteinander eingegangen.
Da es zwei Versionen des Autopiloten gibt, wird an den entsprechenden Stellen darauf hingewiesen, dass es je nach Version unterschiedliche Sensoren und Beschaltungen gibt.
So existiert eine Version, auf der die Gyroskope und der Beschleunigungssensor ausschließlich von dem Hersteller ST Microelectronics verbaut ist. Auf der zweiten Version des Autopiloten sind hingegen Gyroskope von InvenSense und ein Beschleunigungssensor des Herstellers Analog Devices im Einsatz. Die Schaltpläne sind im Anhang auf den Seiten 47 bis 50 aufgeführt.

5.1 Blockschaltbild

Es wurde ein Blockschaltbild erstellt, in dem alle Sensoren, Stecker und externe Module enthalten sind. Das Blockschaltbild des Autopiloten zeigt das Bild 5.1 In der Mitte ist der Mikrocontroller angeordnet. Links von ihm die Sensoren, rechts und unten die Peripheriebausteine.

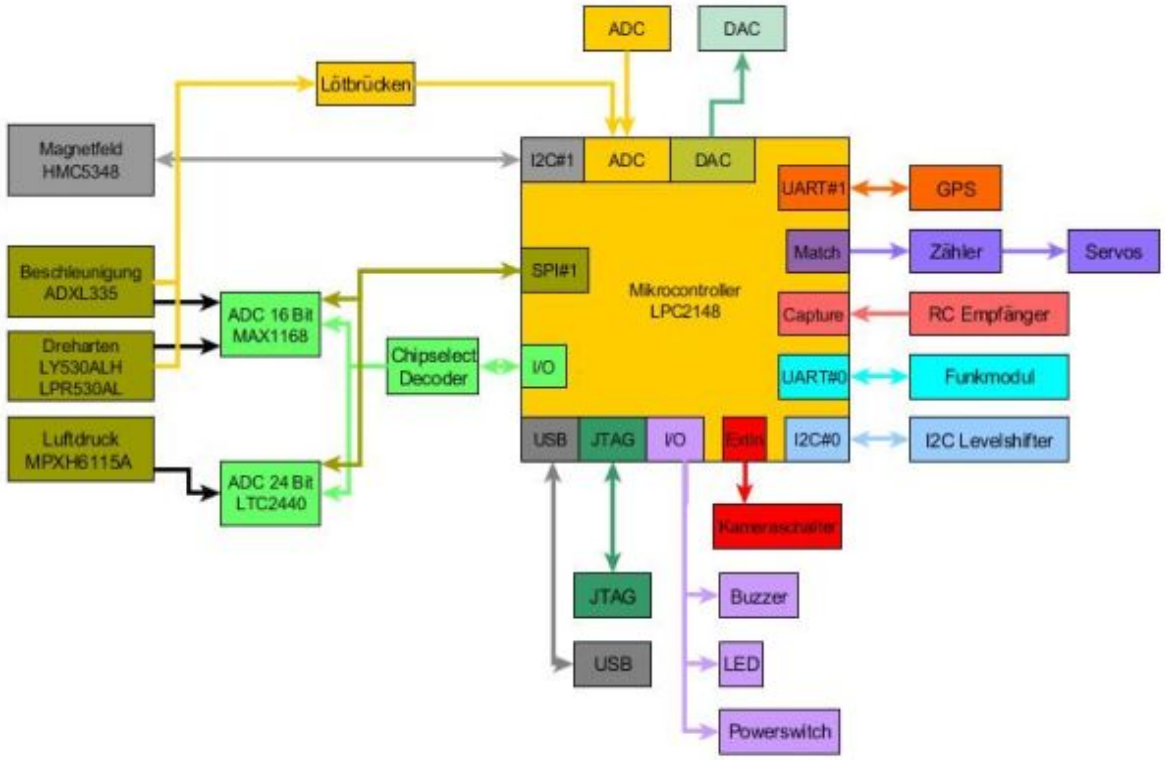


Abbildung 5.1: Blockschaltbild

5.2 Mikrocontroller

Der Mikrocontroller LPC2148 von Philips Semiconductors [D1] ist mit einer Standardbeschaltung versehen. Er wird mit 3,3V versorgt. Alle Versorgungsspannungen sind mit 100nF gegen Masse abgeblockt. Ein 12 MHz Quarz mit den dazugehörigen 22pF Kondensatoren sorgen für die Takterzeugung.

5.3 Magnetsensor

Die Verschaltung und die Dimensionierung der Beschaltung des Magnetfeldsensor erfolgt nach dem Datenblatt [D2]. Der Ladekondensator C56 ist entgegen dem Datenblatt mit 10μF, statt mit 4,7μF ausgeführt. In früheren Versuchen mit einem 4,7μF Kondensator zeigte der Magnetfeldsensor keine Funktion. Der Ladekondensator dient zum Bereitstellen von Energie bei kurzzeitigen Flipimpulsen, die bis zu 1A groß sein können. Ein 100nF Abblockkondensator liegt zwischen AVDD und GND. Die Kommunikation zwischen dem Magnetfeldsensor und dem Mikrocontroller erfolgt mittels I^2C Bus und ist mit jeweils einem Pullup Widerstand versehen.

5.4 Drucksensor

Die Beschaltung des Drucksensors ist an sein Datenblatt [D3] angelehnt. Mit dieser Beschaltung arbeitet der Ausgang als Stromausgang. Es ist ein 47pF Kondensator zwischen dem Ausgang und GND geschaltet. Statt des vorgesehenen 51kΩ Widerstandes, wird über einen Spannungsteiler aus einem 24kΩ und einem 27kΩ Widerstand die Ausgangsspannung geteilt und dann auf einen 24Bit ADC gelegt. Der ADC hat einen Eingangsspannungsbereich von VREF/2, also 0 bis +2,5V, da seine Referenzspannung auf 5V liegt. Die maximale Ausgangsspannung des Luftdrucksensors beträgt 4,8V. Durch den Spannungsteiler ergibt sich somit eine Spannung von ca. 2,4V am Eingang des ADC. Der Ausgang des Drucksensors über ein 10kΩ Widerstand auf den 16 Bit ADC MAX1168 gelegt.

5.5 Analog-Digital-Umsetzer 24 Bit

Der ADC LTC2440 ist nach Empfehlung des Datenblattes [D4] beschaltet. Die Versorgungsspannung bilden digitale 5V, die über einen 100nF Kondensator und einem 10μF Vielschichtkondensator gegen Masse geblockt ist. Die Referenzspannung wird von analogen 5V bereitgestellt. Sie ist zusätzlich mit einer 0,47μH Induktivität und einem 100nF Kondensator gegen Störungen entkoppelt. Die Kommunikation zwischen dem ADC und dem Mikrocontroller erfolgt über einen SPI Bus, dessen Spannung über einen Spannungsteiler auf 3,3V herabgesetzt wird. 3,3V ist auch die Spannung, die den Mikrocontroller versorgt.

5.6 Gyroskope

Die Gyroskope LPR530AL und LY530ALH von ST Microelectronics sind nach Datenblatt[D5 und D6] beschaltet. Der LPR530AL misst die Pitch- und Roll-Drehraten, der LY530ALH die

Yaw-Drehrate.
Die Gyroskope haben zwei Ausgänge für zwei verschiedene Messbereiche, die beide zur gleichen Zeit genutzt werden können. Zum einen ist das ein ±300°/s Messbereich und zum anderen ein vierfach Verstärkter Messbereich von ±1200°/s . Bei einem Einsatz auf Schwebefluggeräten reicht der ±300°/s Messbereich aus. Für Flächenfluggeräte kommt der ±1200°/s Messbereich zum Einsatz, um die Rollbewegungen im Kunstflug messen zu können.
Ein 100nF Abblockkondensator ist zwischen der analogen 3,3V Spannungsversorgung und der analogen Masse vorgesehen. Eine Filterschaltung, bestehend aus einem 10nF Kondensator (C26) und parallel dazu eine Reihenschaltung aus einem 10kΩ Widerstand (R63) mit einem 470nF Kondensator (C23), wurde realisiert. Die Signalausgänge sind mit einem Tiefpassfilter versehen, der eine Eckfrequenz von 159Hz aufweist. Ein optionaler Hochpass, wie im Datenblatt beschrieben, wurde nicht verwendet, da er sich im Algorithmus als nachteilig erwies. Um den Messbereich wählen zu können ist in jeden Ausgangskreis jeweils eine Lötbrücke vorgesehen (J1 - J4). Wird der ±300°/s Messbereich benötigt, so werden J2 und J4 verlötet. J1 und J3 bleiben offen. Im ±1200°/s Messbereich werden entsprechend J1 und J3 verlötet und J2 und J4 bleiben offen. Der vierfach verstärkte Ausgang ist zusätzlich permanent auf einen Eingang des MAX1168 gelegt, um immer den großen Messbereich auswerten zu können. Der Ausgang VREF bleibt in jedem Modus offen. Der Selftest Eingang ist zusammen mit dem des Beschleunigungssensors auf einen I/O Anschluss des Mikrocontroller gelegt. Die Ausgänge PD (Power-down) und HP (High Pass Filter Reset) sind nach Datenblatt auf analoge Masse gelegt.

Für die andere Version des Autopiloten sind die Gyroskope IDG500 und ISZ500 vorgesehen. Der IDG500 erfasst die Pitch- und Roll-Drehraten, der ISZ500 die Yaw-Drehraten. Beide sind nach Datenblatt [D7 und D8] verschaltet. Auch diese Gyroskope haben zwei Ausgänge mit verschiedenen Messbereichen. Dies sind ±110°/s und ±500°/s. Die Spannungsversorgung übernimmt eine separate, analoge 3,3V Versorgung. Das Datenblatt sieht einen 100nF Kondensator zum Abblocken der Versorgungsspannung vor und jeweils einen 220nF Kondensator an den Anschlüssen XAGC und YAGC gegen Masse. Zum Betreiben der Ladungspumpe ist ein 100nF Kondensator mit einer Spannungsfestigkeit von 10V, am CPOUT Ausgang, vorgesehen. Die nicht verstärkten Ausgänge sind mit jeweils einem 750Ω Widerstand und einem 100nF Kondensator versehen, was einen Tiefpassfilter mit einer Eckfrequenz von 2122Hz ergibt. Mittels der Lötbrücken J1 bis J4 kann auch hier zwischen den beiden Messbereichen gewählt werden.

5.7 Beschleunigungssensor

Der Beschleunigungssensor LIS334ALH von ST Microelectronics wird nach Datenblatt [D9] beschaltet. Das Datenblatt sieht einen 100nF Abblockkondensator zwischen der analogen Versorgung und Masse vor. An jedem der drei Signalausgänge ist jeweils ein 100nF Kondensator geschaltet, die eine bestimmte Bandbreite parametrisieren. Aus dem Datenblatt kann nach Gleichung (5.1) die Bandbreite errechnet werden.

$$C_{(x,y,z)} = \frac{1,45\mu F}{f_t} \tag{5.1}$$

Aufgrund guter Erfahrungswerte für die Kondensatoren von 100nF, ergibt sich eine Bandbreite von 14,5Hz.
Die drei Ausgänge sind jeweils auf ein Eingangsanschluss des MAX1168 ADC gelegt. Der Selftest-Eingang ist an einen I/O Anschluss des Mikrocontroller gelegt. Ebenso der Fullscale-Eingang. Mit dem Fullscale Eingang kann der Messbereich von ±2g oder ±6g gewählt werden. Bei einem Einsatz des Autopiloten auf einem Schwebefluggerät kommt der ±2g Messbereich zum Einsatz, da hier nur geringe Beschleunigungsraten auftreten. Anders bei Flächenfluggeräten, bei denen vor allem im Kurvenflug hohe Beschleunigungskräfte herrschen. Es ist weiterhin möglich den Messbereich Situationsabhängig umzuschalten.
Für die zweite Version des Autopiloten ist der Beschleunigungssensor ADXL335 von Analog Devices vorgesehen, der nach Datenblattempfehlung [D10] beschaltet ist. Ein 100nF Kondensator blockt hochfrequente Störspitzen ab. Die Spannungsversorgung übernimmt die 3,3V analog Versorgung. Auch hier kommen jeweils ein 100nF Kondensator in den Ausgangsleitungen zum Einsatz, mit der die Bandbreite eingestellt wird. Das Datenblatt empfiehlt eine Berechnung nach Gleichung (5.2).

$$C_{(x,y,z)} = \frac{5\mu F}{F_{\text{-3dB}}} \tag{5.2}$$

Setzt man für $F_{\text{-3dB}}$ 50Hz ein, ergibt sich eine Kapazität von 100nF. Mit dieser wurden in der Vergangenheit bereits gute Erfahrungen erzielt. Der Selftest Eingang ist an einen I/O Anschluss des Mikrocontroller gelegt. Der ADXL335 hat nur einen ±3g Messbereich. Dieser könnte beim Kunstflug überschritten werden, was diesen Autopiloten vorzugsweise für Schwebefluggeräte einsetzbar macht.

5.8 Analog-Digital-Umsetzer 16 Bit

Die Beschaltung des MAX1168 von Maxim erfolgt nach Datenblattempfehlung [D11]. So sind an allen Versorgungseingängen und REF jeweils ein 100nF Kondensator zum Abblocken der Versorgungsspannung gelegt. Die Eingänge DSEL und DSPR sind über 10kΩ Pullups an 5V angeschlossen. Der digitalen Versorgungseingang ist mit einem 4,7μF Keramik-Vielschicht-Kondensator verschaltet.
Die Analogeingänge sind für die Rohdaten aus dem Beschleunigungssensor und den Gyroskopen mit diesen beschaltet und der Ausgang des Drucksensors ist mit dem MAX1168 verbunden. In der Variante mit dem ST Sensoren ist zusätzlich der 1200°/s Rollausgang des Gyroskops auf einen Eingang des MAX1168 gelegt.

5.9 Spannungsversorgung

5.9.1 PTH08080

Als 5V Spannungsversorgung ist der Schaltregler PTH08080 von Texas Instruments vorgesehen. Er setzt die Batteriespannung auf 5V herab. Seine Beschaltung erfolgt nach Empfehlung des Datenblatts [D12]. Das Datenblatt sieht am Eingang einen 100μF mit einem niedrigen ESR (Equivalent Series Resistance) vor. Die Spannungsfestigkeit von 20V ist aufgrund des Platzbedarfs des Kondensators niedrig gewählt. Laut Datenblatt ist für eine Ausgangsspannung von 5V ein Widerstand von 348Ω zwischen Adj. und Masse zu schalten. Um diesen Wert zu erreichen wird eine Reihenschaltung aus einem 330Ω und einem 18Ω Widerstand realisiert. Zur Minimierung der Restwelligkeit am Ausgang, wird an diesem ein 100μF Kondensator eingesetzt. Alternativ kann statt des PTH08080 ein LT1117 zur 5V Spannungsversorgung eingesetzt werden. Diese Alternative soll zum Einsatz kommen, wenn keine Servos an dem Autopiloten angeschlossen werden und somit ein geringer Strombedarf des Autopiloten besteht.

5.9.2 LT1117

Für die 3,3V digital Spannungsversorgung ist ein LT1117 von Linear Technology ausgewählt. Er wird von den 5V aus dem PTH08080 versorgt und setzt diese Spannung auf 3,3V herab. Seine Beschaltung erfolgt nach Datenblattempfehlung [D13]. So ist am Eingang ein 10μF Kondensator vorgesehen. Der Kondensator hat eine Spannungsfestigkeit von 16V und ist ausreichend dimensioniert. Am Ausgang befindet sich ein 22μF Kondensator mit einer Spannungsfestigkeit von 6,3V. Ein 100nF Kondensator ist parallel zum Ausgang geschaltet um hochfrequente Störspitzen zu unterdrücken.

5.9.3 LT1761

Alle LT1761 dienen zur 3,3V analog- und 3,3V Gyro Spannungsversorgung. Dies sind zum einen eine 3,3V Versorgung und eine weitere 3,3V Versorgung, mit der die Gyroskope versorgt werden. Zum Anderen ist eine 5V Analogversorgung vorgesehen. Bei allen dieser Versorgungen ist die externe Beschaltung des LT1761 nach Datenblatt [D14] gleich. In Batterie betriebenen Schaltungen wird am Eingang ein 2,2μF Kondensator empfohlen. Der ausgewählte Kondensator ist ein Keramik-Vielschicht-Kondensator. Die Spannungsfestigkeit ist aufgrund der hohen möglichen Eingangsspannung mit 25V bemessen. Um Schwingungen zu vermeiden ist am Ausgang ein 10μF Kondensator mit einer Spannungsfestigkeit von 6,3V geschaltet. Ein Bypasskondensator zwischen dem Ausgang und dem Bypassanschluss von 10nF hält das Rauschen unter 20μV_{RMS}. Des weiteren wird am Eingang der Regler, die die analogen Versorgungsspannungen erzeugen, eine Induktivität von 0,47μH zur Entstörung verwendet.

5.10 Chipselect

Zum Erzeugen des Chipselectsignals dient ein Demultiplexer 74HC138 [D15]. Da seine Versorgungsspannung 3,3V beträgt, weisen auch alle Chipselectsignale einen 3,3V Pegel auf. Beim Einschalten kann der Mikrocontroller undefinierte Signale führen. Jeder Eingangsanschluss ist mit einem 10kΩ Pullup versehen, um einen definierten Highpegel beim Einschalten zu erhalten. Ein 100nF Kondensator blockt an der Versorgungsspannung hochfrequente Störspitzen ab. Fünf weitere Ausgänge des Chipselects sind als Reserve auf den SPI#1 Stecker gelegt. Um den Baustein einzuschalten sind die Eingänge G1 auf 3,3V und die Eingänge G2A und G2B auf Masse geschaltet.

5.11 Kameraschalter

Zum Schalten einer externen Kamera ist der Leistungsschalter TPS2051 vorgesehen. Er wird nach Datenblatt [D16] beschaltet. Dies sieht einen 100nF Kondensator am Eingang vor. Am Ausgang liegt ein 22μF Kondensator.

5.12 I^{2}C Levelshifter

Die Beschaltung des I^{2}C Levelshifter PCA9306DP [D17] wurde aus der Diplomarbeit [1] von Herrn Andreas Dei übernommen. Die Beschaltung stimmt nicht mit der des Datenblatts überein. Jedoch wurde bei der Beschaltung nach Herrn Dei eine einwandfreie Funktion des I^{2}C Levelshifter nachgewiesen.

5.13 Servo/Zähler

Der Zählerbaustein CD4017 ist mit einem 100nF Kondensator zum Abblocken der Versorgungsspannung beschaltet. Dies ist die einzig nötige Beschaltung für den CD4017. Er wird mit 5V versorgt. Acht Ausgänge sind auf eine Molexstiftleiste gelegt, so dass handelsübliche Servos eingesteckt werden können. Drei sind für die Kamera vorgesehen. Des weiteren wird eine zweite Versorgung der Servos realisiert, die es ermöglicht die Servos auch mit einer externen Spannung zu versorgen. Dies ist dann der Fall, wenn mittelgroße Servos angesteuert werden, deren Anlaufstrom bis zu 2A betragen kann. Ein Strom von 2A würde die interne Stromversorgung überlasten. Die externe Stromversorgung kann mittel Lötbrücke auf die Servoanschlüsse geschaltet werden.

5.14 EEPROM

Das EEPROM 24LC64 wird nach Datenblatt [D18] verschaltet. Ein 100nF Abblockkondensator ist jedoch das einzige Peripheriebauteil. Versorgt wird er mit digitalen 3,3V.

5.15 Molex Steckerleistenanschluss

Die Signalanordnung auf den Steckerleisten ist meist nach folgendem Muster aufgebaut: Auf dem Anschluss 1 liegt digital bzw. analog Masse. Auf Anschluss 2 folgt die höchste Spannung auf dem jeweiligen Stecker. Also Batteriespannung, 5V oder 3,3V. Darauf folgen die Signale des Steckers.
Die 5V analog vom Analogstecker bildet die Ausnahme, da diese Spannung erst nachträglich auf den Stecker gelegt wurden und sie sich so leichter entflechten lies.

5.15.1 UART0 X1 (Funk)

Ein 7-poliger Stecker übernimmt die Verbindung zu einem Funkmodul, das von der UART0 Schnittstelle angesprochen wird. Zusätzlich sind ein I/O Anschluss des Mikrocontrollers und das Resetsignal auf den Stecker gelegt.

Tabelle 5.1: Anschlussbelegung X1 UART0

Anschluss	**Signal**	**Beschreibung**
1	GND	Digital Masse
2	+5V	Digital 5V
3	+3V3	Digital 3,3V
4	LPC_TXD0	Transmit Data
5	LPC_RXD0	Receive Data
6	FUNK_IO	IO Anschluss
7	LPC_RESET	Reset

5.15.2 USB X2

Der 4-polige USB Stecker ist nach USB Standard verschaltet, somit liegt auf Anschluss 1 das U+_USB (5V) Signal, auf Anschluss 2 D+, auf Anschluss 3 D- und auf Anschluss 4 die digitale Masse. Das USB Signal vom Mikrocontroller ist über die üblichen 33Ω Serienwiderstände und 18pF Kondensatoren gegen Masse geführt. Der Schaltausgang wird über einen Transistor mit einem 1,5kΩ Widerstand geschaltet.

Tabelle 5.2: Anschlussbelegung X2 USB

Anschluss	**Signal**	**Beschreibung**
1	U+_USB	5V vom USB Bus
2	USB+	USB Daten +
3	USB-	USB Daten -
4	GND	Digital Masse

5.15.3 UART1 X3 (GPS)

Die UART1 Schnittstelle ist für ein GPS Modul gedacht, das über dem Autopiloten montiert werden kann. Hierzu ist auch ein Timepuls Signal auf den Stecker gelegt.

Tabelle 5.3: Anschlussbelegung X3 UART1

Anschluss	Signal	Beschreibung
1	GND	Digital Masse
2	+5V	Digital 5V
3	+3V3	Digital 3,3V
4	LPC_TXD1	Transmit Data
5	LPC_RXD1	Receive Data
6	GPS_Timepuls	IO Anschluss
7	LPC_RESET	Reset

5.15.4 CAM X4

Der Cam Stecker dient zum Anschluss einer Kamera an das Fluggerät. Drei Servokanäle sind auf den Stecker gelegt, um den Blickwinkel der Kamera stellen zu können. Des weiteren liegt auf dem Stecker eine 5V Spannung, sowie die Batteriespannung die durch ein LR Glied gefiltert ist. Zusätzlich ist eine geschaltete 5V Versorgung auf den Stecker gelegt.

Tabelle 5.4: Anschlussbelegung X4 CAM

Anschluss	Signal	Beschreibung
1	GND	Digital Masse
2	V+	Batteriespannung
3	+5V	Digital 5V
4	S8	Servosignal 8
5	S9	Servosignal 9
6	S10	Servosignal 10
7	PWR_SW	Power Switch

5.15.5 I²C0 X5

Die I²C0 Schnittstelle liegt auf einem 4-poligen Stecker. Die Signale haben 5V Pegel und sind zum Ansteuern der Motorsteller. Ein Levelshifter passt die Signalpegel zwischen den 5V und den 3,3V vom Mikrocontroller an.

Tabelle 5.5: Anschlussbelegung X5 I²C0

Anschluss	Signal	Beschreibung
1	GND	Digital Masse
2	SDA0_5V	I²C Data 5V
3	SCL0_5V	I²C Clock 5V
4	+5V	Digital 5V

5.15.6 RC X6

RC ist ein 3-poliger Stecker und ist zum Anschluss eines Fernsteuerempfängers gedacht. Der Fernsteuerempfänger benötigt lediglich die Versorgungsspannung von 5V, Masse und das Fernsteuersignal. Ein 100Ω Widerstand in der Fernsteuersignalleitung verhindert die Ausbreitung von Empfangsstörungen die durch den Autopiloten verursacht werden.

Tabelle 5.6: Anschlussbelegung X6 RC

Anschluss	Signal	Beschreibung
1	GND	Digital Masse
2	+5V	Digital 5V
3	PPM_IN	Fernsteuersignal

5.15.7 SPI1 X7

An die SPI1 Schnittstelle können weitere externe Module, wie zum Beispiel eine SD Karte oder ein weiterer Mikrocontroller angeschlossen werden. Fünf freie Chipselectsignale vom Chipselectdecoder sind auf den Stecker herausgeführt.

Tabelle 5.7: Anschlussbelegung X7 SPI1

Anschluss	Signal	Beschreibung
1	+3V3	Digital 3,3V
2	GND	Digital Masse
3	LPC_SCK1	SPI Clock1
4	LPC_MISO1	SPI MISO1
5	LPC_MOSI1	SPI MOSI1
6	LPC_SSEL	SPI Slave Select
7	CS_RES_0	Chipselect Reserve 0
8	CS_RES_1	Chipselect Reserve 1
9	CS_RES_2	Chipselect Reserve 2
10	CS_RES_3	Chipselect Reserve 3
11	CS_RES_4	Chipselect Reserve 4

5.15.8 Analog X10

Der Analogstecker ist 11-polig und ist mit analogen Eingängen des Mikrocontrollers belegt. Über Lötbrücken sind die Gyroskop- und Beschleunigungssignale verschaltet. Dies macht den Analogstecker kompatibel zum Tiny13 Analogstecker. Bleiben die Lötbrücken unverlötet, so liegen auf dem Stecker die Analogeingänge vom LPC2148. Zunächst wurde nur eine analoge 3,3V Versorgung vorgesehen. Erst später wurde beschlossen, dass auch eine analoge 5V Versorgung auf den Stecker gelegt werden soll. Aus diesem Grund ist bei dem Analogstecker die 5V Versorgung auf dem Anschluss 11 angeordnet. Des weiteren war diese Anordnung einfacher zu entflechten, da Anschluss 11 an den Analogteil der Leiterplatte grenzt.

Tabelle 5.8: Anschlussbelegung X10 Analog

Anschluss	Signal	Beschreibung
1	GNDA	Analog Masse
2	+3V3A	Analog 3,3V
3	AD0.3/ADC_0	AD Kanal
4	AD0.2/ADC_1	AD Kanal
5	AD0.1/ADC_2	AD Kanal
6	AOUT/ADC_3	AD Kanal
7	AD1.7/ADC_4	AD Kanal
8	AD1.3/ADC_5	AD Kanal
9	AD1.4/ADC_6	AD Kanal
10	AD1.5/ADC_7	AD Kanal
11	+5VA	Analog 5V

5.15.9 JTAG X11

Der JTAG Stecker ist 9-polig und ermöglicht ein Testen und Debuggen. Hierzu ist der JTAG Port des Mikrocontrollers auf den Stecker gelegt. Als Spannungsversorgung dienen digitale 3,3V. Auch das Resetsignal ist auf den Stecker geführt.

Tabelle 5.9: Anschlussbelegung X11 JTAG

Anschluss	**Signal**	**Beschreibung**
1	GND	Digital Masse
2	+3V3	Digital 3,3V
3	JTAG_/TRST	Testreset
4	JTAG_TDI	Test Data Input
5	JTAG_TMS	Test Mode Select Input
6	JTAG_TCK	Test Clock
7	JTAG_RTCK	Returned Test Clock Output
8	JTAG_TDO	Test Data Output
9	LPC_RESET	Reset

6 Bauteilpositionierung

Eine wesentliche Vorgabe für das Positionieren der Bauteile ergibt sich aus der Größe der Leiterplatte. Sie ist auf 30mm x 58mm begrenzt. Es sind vier Befestigungsbohrungen zu platzieren, mit denen ein GPS Modul montiert werden kann. Diese Bohrungen sind 2mm im Durchmesser groß und quadratisch anzuordnen. Die Abstände der Löcher zueinander ergibt sich aus dem Datenblatt [D19] des GPS Modul NAVILOCK 507ETTL. Weitere Vorgaben sind die Ausrichtung der Steckleisten, deren Stecker nach unten und die Kabel nach hinten, also gegen die Flugrichtung, weggeführt werden dürfen. Dies soll Platz bei der Montage des Autopiloten im Fluggerät einsparen. Alle Sensoren sollen auf der Oberseite der Leiterplatte angeordnet sein. Aufgrund neuartiger Sensoren, ist es möglich die Leiterplatte planar zu Bestücken. Das heißt, alle Sensoren können flach liegend positioniert werden. Es soll eine räumliche Trennung von digitalen und analogen Signalen erfolgen.

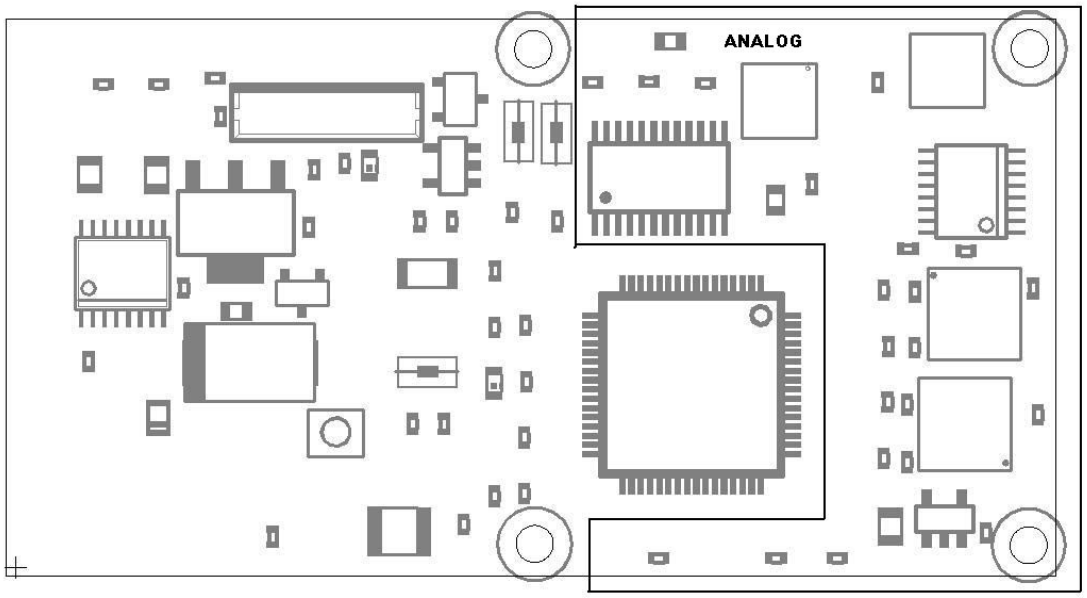


Abbildung 6.1: Bauteilpositionierung auf der Leiterplattenoberseite der ST Variante

Zuerst wurden die Bohrlöcher positioniert. Diese liegen auf der rechten Seite der Leiterplatte. Danach wurden alle Bauteile und ihre externen Bauelemente gruppiert und auf möglichst kleinem Raum zusammengefügt. Die Anordnung der Stecker gestaltete sich als schwierig, da durch sie viel Platz zum Platzieren der anderen Bauteile verloren ging. Sie wurden auf der linken Seite der Leiterplatte untergebracht, zusammen mit dem PTH08080, der ebenfalls viel Platz in Anspruch nimmt. Ein Versuch, die Stecker senkrecht anzuordnen scheiterte, so wurden sie waagerecht, untereinander und nebeneinander platziert. Der Stecker für das GPS Modul sollte von beiden Seiten bestückbar sein. Zudem durften die Stecker nicht dicht an dicht platziert werden, damit man die Stecker sicher aus den Steckleisten entfernen kann. Alle Stecker wurden von der Unterseite bestückt. An der linken Leiterplattenkante wurden die Lötpads für die Servoanschlüsse gelegt. Der Resettaster musste auf der Oberseite platziert werden, jedoch nicht unterhalb des GPS Moduls, um ihn bedie-

nen zu können. Nun wurde der Mikrocontroller und seine externen Bauteile in der Mitte des Autopiloten positioniert. Von den Anschlüssen des Mikrocontrollers führen nahezu alle Leitungen sternförmig zu den Sensoren, Steckern und anderen Bauelementen ab. Die Sensoren wurden oberhalb und rechts des Mikrocontrollers angeordnet. Hier soll auch der analoge Teil der Schaltung verlaufen. Unterhalb des Mikrocontrollers ist der Analogstecker angeordnet. Der JTAG Stecker links vom Mikrocontroller. Diese beiden Stecker können unter dem GPS Modul platziert werden. Der Luftdrucksensor muss er auf der Unterseite der Leiterplatte angeordnet werden. Auf der Oberseite würde das mögliche GPS Modul zu nahe über dem Luftdrucksensor liegen. Bei dem Magnetfeldsensor ist zu beachten, dass sich keine ferromagnetische Gegenstände im Umkreis von 3mm um ihn herum befinden dürfen. Dieser Sensor wird ganz oben rechts auf der Platine angeordnet. Sämtliche externe Beschaltungsbauteile wurden um ihre Hauptbauteile herum verteilt, auf der Ober- sowie Unterseite. Dabei wurden kleinere Bauteile wie Widerstände und Kondensatoren exakt übereinander, auf der Ober- und Unterseite gelegt, um so mehr Platz für Leitungen und Durchkontaktierungen zu bekommen. So mussten alle Bauteile untereinander ebenso viel Platz haben, dass es möglich ist zwischen ihnen mindestens noch eine Durchkontaktierung zu platzieren.

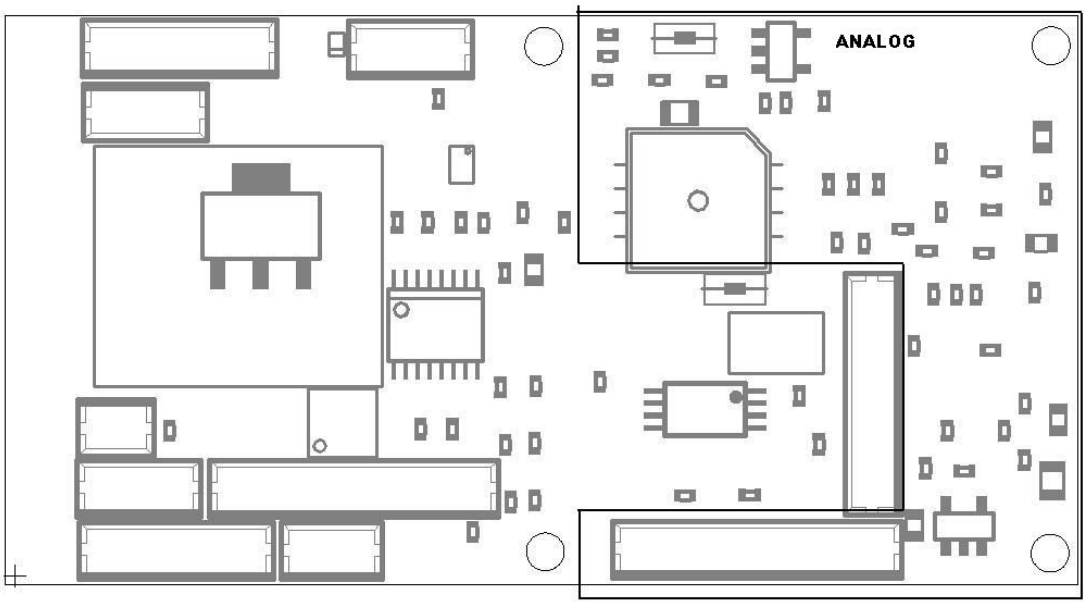


Abbildung 6.2: Bauteilpositionierung auf der Leiterplattenunterseite der ST Variante

Die Vorgaben für die Platzierung gilt für beide Varianten der Autopiloten. Aufgrund anderer Maße der verschiedenen Sensoren kommt es zu Unterschiedlichen Bauteilpositionierungen der beiden Varianten. In den Abbildungen 6.1 und 6.2 sind die Bauteilplatzierungen auf der Ober- und Unterseite der ST Variante und in den Abbildungen 6.3 und 6.4 die der IDG Variante zu sehen.

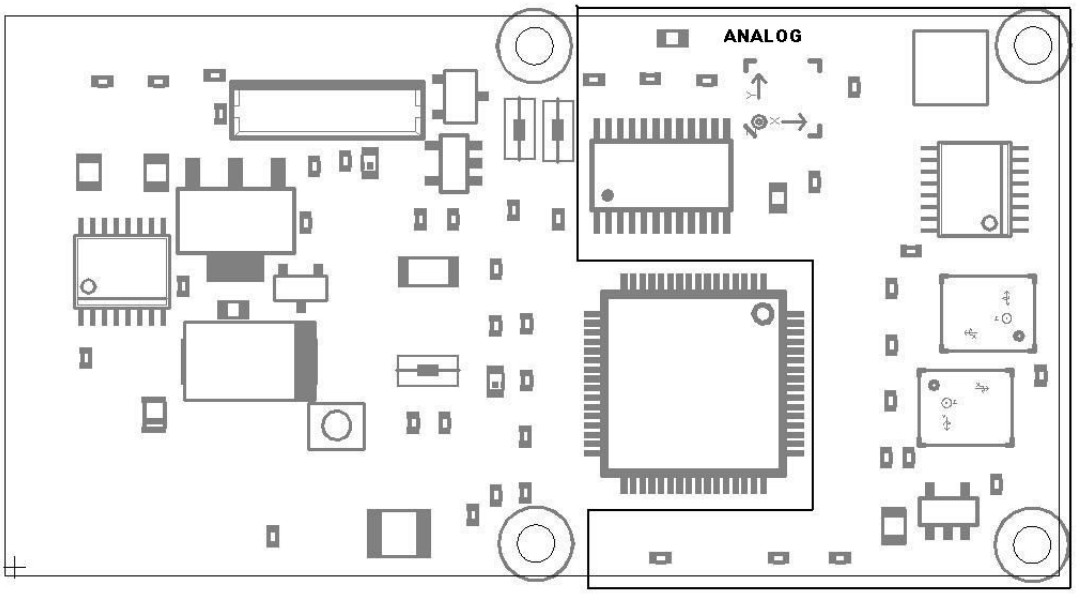


Abbildung 6.3: Bauteilpositionierung auf der Leiterplattenoberseite der IDG Variante

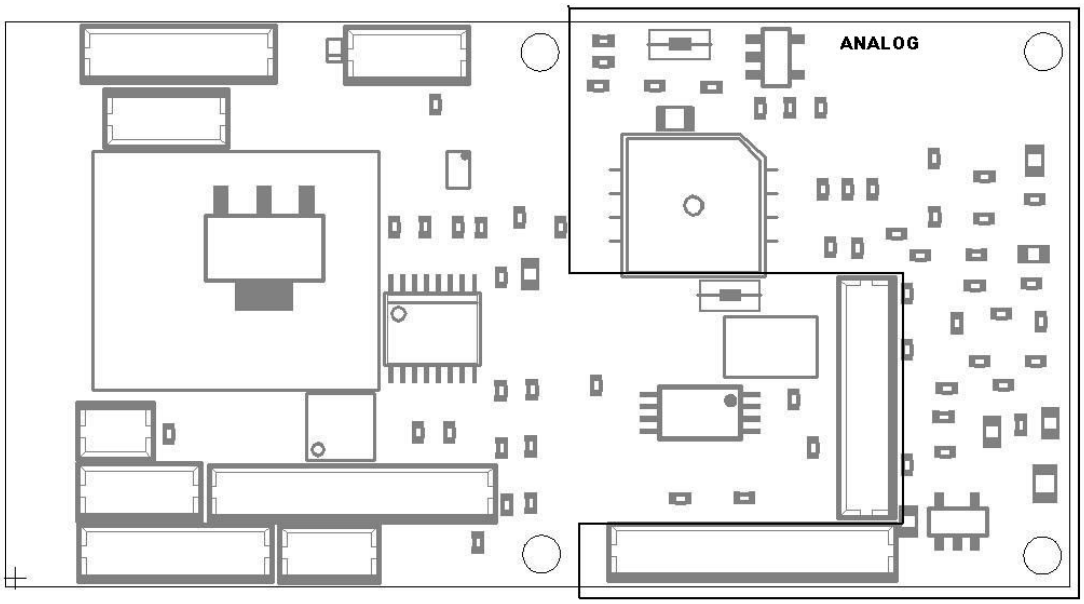


Abbildung 6.4: Bauteilpositionierung auf der Leiterplattenunterseite der IDG Variante

7 Das Entflechten

Nachdem alle Bauteile Positioniert wurden, konnte mit dem Entflechtvorgang begonnen werden. Zuerst mussten jedoch einige Parameter ermittelt und eingestellt werden. Diese betreffen die Leiterbahnbreiten, Maße der Durchkontaktierungen, Abstände der Leiterbahnen zueinander und die Nutzung der vier zur Verfügung stehenden Layer. Es wurde vom Leiterplattenhersteller PCB-POOL eine DRC Datei herunter geladen, die alle der aufgeführten Parameter enthält. So ist sichergestellt, dass die entworfenen Leiterplatten auch von einem Hersteller produzierbar sind. Wichtige Vorgaben sind zum Beispiel die Leiterbahnbreite, die für Signalleitungen eine Breite von 5mil vorsehen. Für Stromversorgungsleitungen zunächst 20mil. Diese Leiterbahnbreite stellte sich in einem ersten Entflechtvorgang allerdings als zu platzverbrauchend heraus. So wurden diese Breiten auf 10mil reduziert. Nur die Stromführenden Leitungen der externen Servostromversorgung wurden mit 20mil entflechtet, da hier Ströme bis zu 2A auftreten können. Durchkontaktierungen haben einen Durchmesser von 8mil und einen Restring von 5mil. Diese stellen die kleinsten produzierbaren Maße des Herstellers dar. Restringe in den Innenlayern haben ebenfalls ein Maß von 5mil. So ist es möglich Leitungen in den Innenlagen auch durch die Kontakte der Steckleisten zu verlegen. Die Leiterbahnstärke beträgt 35μm. Der Abstand zwischen Leiterplattenkante und Kupfermaterial beträgt 20mil. Die Leiterbahnabstände zueinander sollen 5mil betragen. Dann wurden die Einstellungen zu den einzelnen Layern getroffen. Die Einstellungen sehen vor, dass der obere Layer für Signalleitungen gedacht ist. Die Vorzugsrichtung ist waagerecht, das heißt dort wo es möglich ist, werden Leitungen auf dem oberen Layer waagerecht verlegt. Sollte dies nicht möglich sein, wird von Autorouter eine Durchkontaktierung gesetzt und auf einem anderen Layer mit einer anderen Vorzugsrichtung weiter entflechtet.

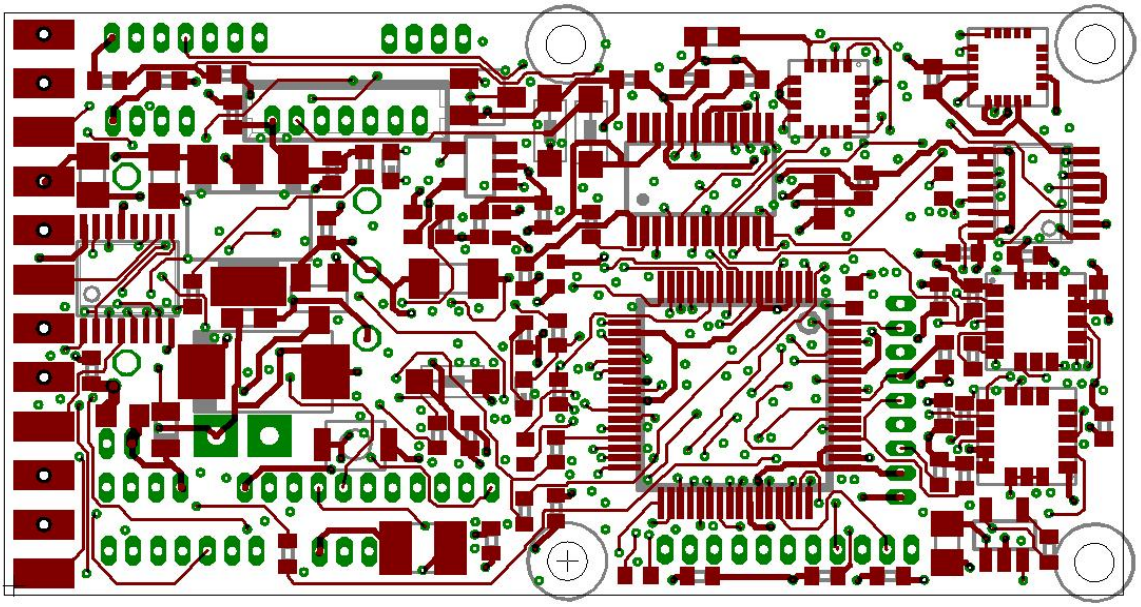


Abbildung 7.1: Entflechtungsansicht der Leiterplattenoberseite der ST Variante

Der untere Layer hat eine senkrechte Vorzugsrichtung und ist ebenfalls für Signalleitungen. Der obere Innenlayer ist für Signalleitungen ohne Vorzugsrichtung. So legt der Autorouter die Leitungen so, dass möglichst kurze Verbindungen entstehen. Der untere Innenlayer ist für digitale und analoge Masse. Hierzu wurden zwei Polygone in die Leiterplatte eingefügt, um zwei Masseflächen zu erhalten. Es gibt ein GND Polygon für die digitale Masse. Es füllt den größten Teil der Platine und liegt über den meisten der digital versorgten Bauteile. Das zweite ist das analoge Massepolygon und liegt über sämtlichen analogen Bauteilen.

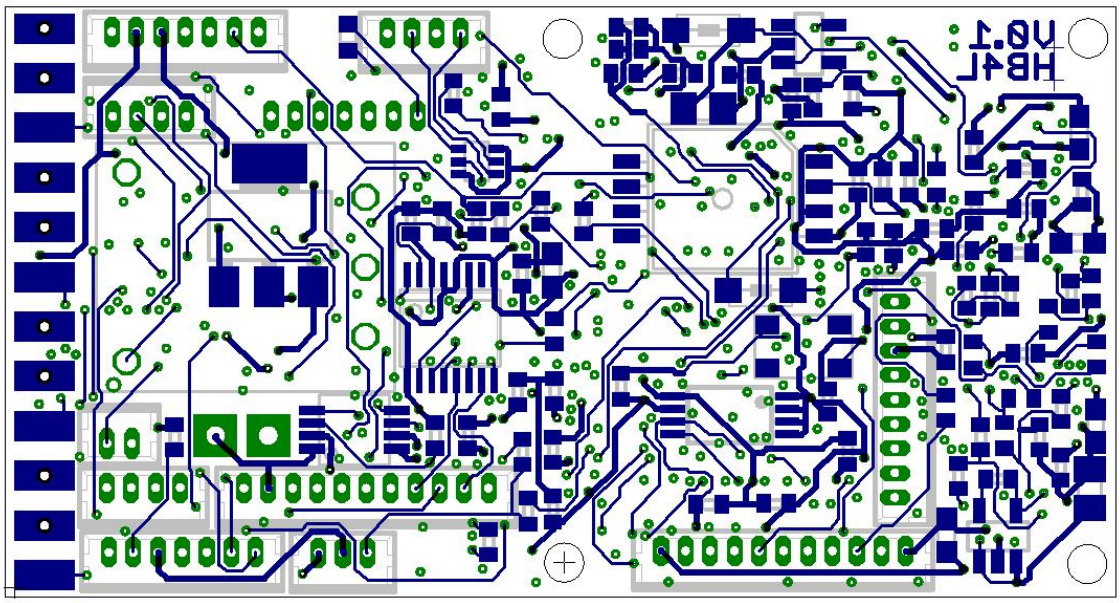


Abbildung 7.2: Entflechtungsansicht der Leiterplattenunterseite der ST Variante

Nachdem alle Abstände und Layereinstellungen getroffen wurden, wurde ein erster Entflechtungsvorgang mit dem EAGLE Autorouter gestartet. Dieser schaffte etwa 60% der insgesamt 474 zu verlegenden Leiterbahnen. Dann wurden an Stellen die der Autorouter die Leitungen nicht verlegen konnte, die Bauteile und bestehende Leitungen ein wenig verschoben, um mehr Platz für Leitungen und Durchkontaktierungen zu schaffen. Diese Methode erwies sich allerdings als Nachteil, da der EAGLE Autorouter von Hand gelegte oder verschobene Leitungen nicht mehr in seinem Entflechtvorgang berücksichtigt. Dennoch wurde ein nächster Entflechtvorgang gestartet. Nun wurden ca. 75% erreicht. Nach einigen Wiederholungen des oben genannten Verfahrens wurde keine nennenswerte Verbesserung des Entflechtungsergebnisses erzielt. Daraufhin wurde mit dem Autorouter von Freeroute [2] weitergearbeitet. Dieser dient ausschließlich zum Entflechten, jedoch nicht zum Editieren des Layouts. Freeroute erzielte ein Ergebnis von 100%. Allerdings gingen beim Exportieren der Datei nach EAGLE einige Verbindungen Verloren, sodass noch etwa 20 Leitungen wieder zu verlegen waren. Diese wurden dann per Hand entflechtet. Es erwies sich als sicherste und schnellste Methode. Da im weiteren Verlauf der Bachelorthesis noch einige Änderungen im Schaltplan und im Layout gemacht wurden, konnten diese so effektivsten per Hand umgesetzt werden. Nachdem alle Leitungen verlegt wurden, mussten alle Verbindungen optimiert werden, in dem Leiterbahnen in einem 45° Winkel verkürzt

wurden und es konnten noch einige Durchkontaktierungen eingespart werden, da der Autorouter teilweise unnötige Durchkontaktierungen gesetzt hat. Ein weiteres Problem stellten verschiedene Leiterbahnlängen des SPI Busses dar. Um Laufzeitenfehler zu vermeiden, sollten diese Leitungen möglichst gleich lang sein. Sie wurden soweit verkürzt, dass sie alle die gleiche Länge hatten.
In der Abbildung 7.1 ist das Ergebnis des Entflechtens auf der Oberseite der ST Variante zu sehen. Abbildung 7.2 zeigt die gleiche Leiterplatte von der Unterseite. Die Abbildungen 7.3 und 7.4 zeigen Ober- und Unterseite der IDG Variante. Die mittleren Lagen sind im Anhang auf den Seiten 55, 55, 58 und 59 zu finden.

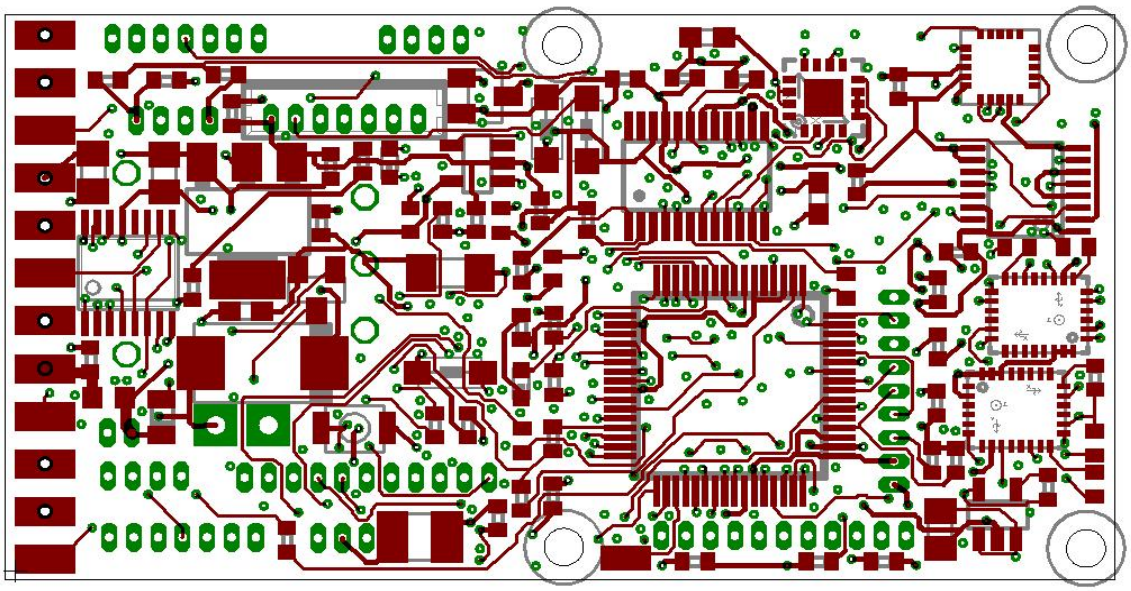


Abbildung 7.3: Entflechtungansicht der Leiterplattenoberseite der IDG Variante

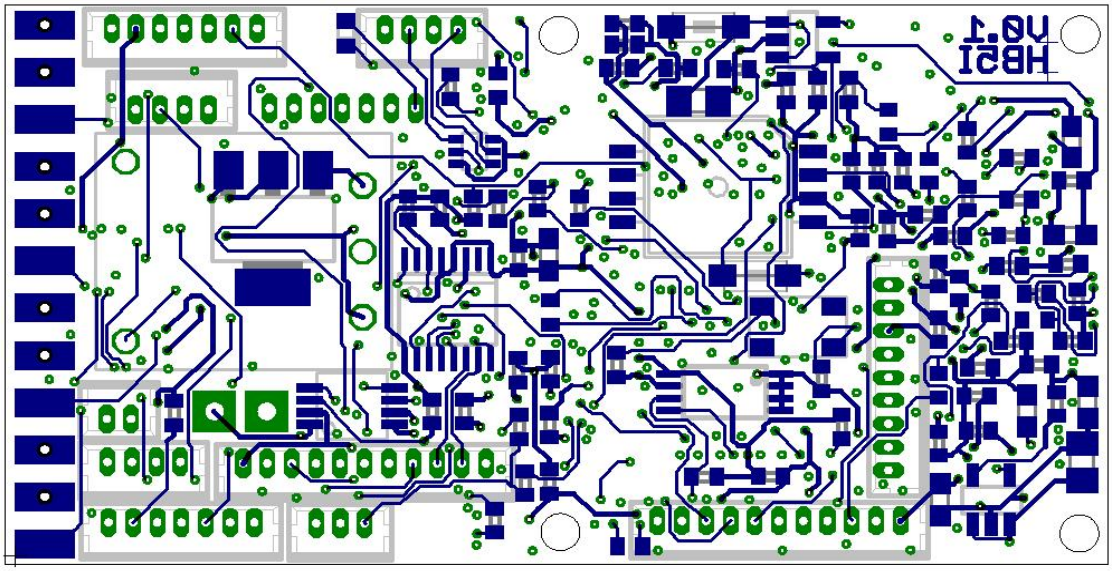


Abbildung 7.4: Entflechtungsansicht der Leiterplattenunterseite der IDG Variante

8 Bestücken und Löten

8.1 Prozess des Bestückens und Lötens

Die in EAGLE entwickelten Leiterplatten wurden beim Platinenhersteller PCB-Pool gefertigt. Der Preis für sechs Leiterplatten war der gleiche Preis, wie für nur eine Leiterplatte. So wurden sechs Platinen bestellt. Jeweils zur Hälfte als LIS und IDG Variante. Davon wurden jeweils zwei bestückt und sind nachfolgend als "Platine 1 LIS, Platine 2 LIS, Platine 1 IDG und Platine 2 IDG"benannt.
Da die Hochschule Bremen nicht über die nötigen Gerätschaften verfügt, um eine Platine mit Bauteilen der Form 0402 zu bestücken, wurde die Bestückung und der Lötvorgang an der Universität Bremen durchgeführt.
Zuerst wurde ein Bestückungsplan mit allen Bauteilenamen ausgedruckt. Dann erfolgte der Erste Schritt der Bestückung. Die Leiterplatten wurden mit einer Schablone, dem sogenannten Stencil, geliefert. Mit dieser Schablone lässt sich die Lötpaste auf alle SMD Lötpads auftragen, ohne dass Leiterbahnen oder andere Elemente der Leiterplatte mit der Lötpaste in Berührung kommen. Die Schablonen wurden in einer Vorrichtung eingespannt und darunter die Platine ausgerichtet. Nun konnte die Lötpaste auf der Leiterplatte aufgetragen werden. Die Lötpaste wurde zuerst auf der Oberseite der Platine aufgetragen und bestückt.

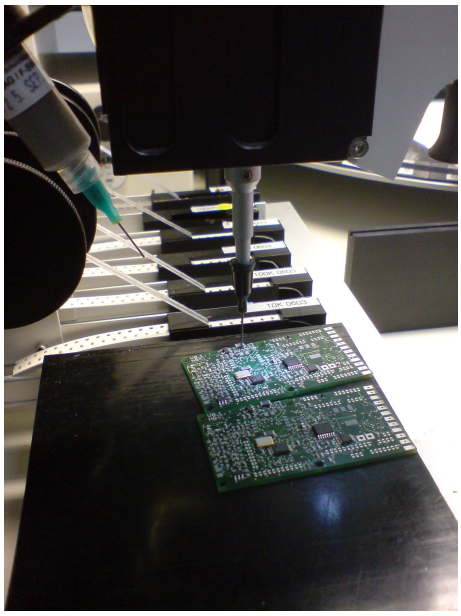


Abbildung 8.1: Unterseite der Leiterplatten währen der Bestückung mit der Saugspitze

Im nächsten Schritt wurden die Platinen in einer halbautomatischen Bestückungsmaschine eingespannt. Diese Maschine verfügt über eine dünne, hohle Spitze, in der ein kleiner Unterdruck herrscht. Mit dem Unterdruck konnten alle Bauteile, bis auf die Sensoren, angesaugt und exakt auf den Lötpads platziert werden, wie in Abbildung 8.1 zu sehen. Durch die Lötpaste hafteten die Bauteile an ihren platzierten Stellen. Dann wurden die Sensoren bestückt. Die Sensorenbestückung erfolgte mit einer weiteren Maschine, die mit einer Kamera ausgerüstet ist. Über einem Monitor wurde die Platine und die Sensoren dargestellt.

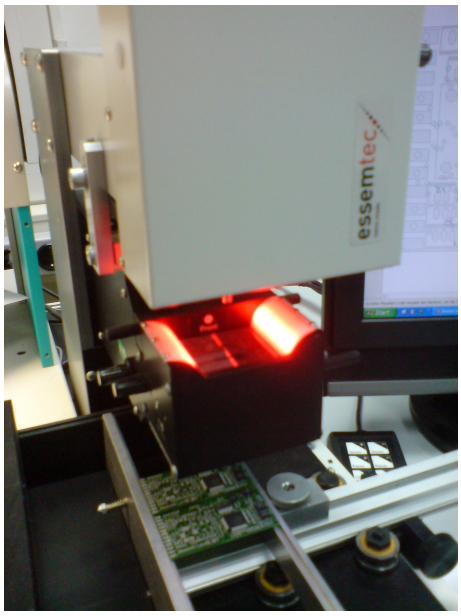


Abbildung 8.2: Bestückung der QFN Gehäuse

Nur mit dieser Maschine können Bauteile mit einem QFN Gehäuse sehr exakt platziert werden. Dies ist in Abbildung 8.2 dargestellt. Nachdem die Platinen komplett bestückt waren, wurde mit einem Mikroskop noch einmal alle Bauteile auf eine genau Positionierung geprüft und eventuell per Hand auf den Lötpads ausgerichtet. Es folgte der Lötvorgang. Die bestückten Platinen wurden in einem Gestell eingespannt, welches auf einem Förderband durch den Reflowofen fährt. Abbildung 8.3 zeigt diesen Vorgang. Zuerst wurde in der ersten Stufe, der Preheat Phase 1, die Platinen auf 170°C erwärmt. In der Preheat Phase 2 auf 180°C. In der dritten und Letzten Stufe folgte der Reflowprozess. Dieser hatte eine Temperatur von 220°C. Da die Lötpaste bleihaltig war und Blei einen niedrigen Schmelzpunkt aufweist, konnte mit dieser Temperatur die Lötpaste zum Schmelzen gebracht werden. So wurden sämtliche Bauteile schonend verlötet. Nach ca. 5 Minuten war der Lötprozess abgeschlossen. Nach dem ersten Lötvorgang wurden die Unterseiten der Leiterplatten in die Vorrichtung zum Auftragen der Lötpaste eingespannt. Da die Oberseiten verlötet waren, konnten die Bauteile nicht mehr herunterfallen. Die Vorgänge des Bestückens und des Lötens waren die gleichen wie bei denen der Unterseite. Da der Reflowofen die erforderliche Lötwärme von oben auf die Leiterplatte übertragt, wurde die bereits Verlötete Unterseite nicht so stark erwärmt. So wurden die vorhandenen Lötstellen nicht auf den Schmelzpunkt des Zinns gebracht und ein Zweiter Lötvorgang der Platine war problemlos durchführbar.

Abbildung 8.3: Reflowofen

8.2 Nacharbeit der Leiterplatten

Nach dem Lötprozess wurde ein Sichtkontrolle an allen Leiterplatten durchgeführt. Da nicht alle Lötstellen ein einwandfreies Ergebnis zeigten, mussten diese Lötstellen nachgebessert werden. Auch fehlten einige Bauteile die nicht bestückt wurden, da sie noch nicht bestellt waren.
Zum Testen wie das manuelle Löten am besten funktioniert, wurde auf einer alte Leiterplatte Widerstände der Bauform 0402 aufgelötet und entlötet. Es zeigte sich, dass dies am besten mit Auftragen von Lötpaste auf den Lötpads durchzuführen war. Zusätzlich war die Lötpaste mit Flussmittel zu versehen. So wurde zunächst die Leiterplatte des Autopiloten korrigiert, die am wenigstens Nacharbeit bedeutete. Hier fehlten Widerstände, die wahrscheinlich noch vor dem Reflowprozess beim Transport der Leiterplatten vom Bestückungsplatz zum Ofen abgerissen wurden. Diese wurden nach oben genannter Methode auf die Leiterplatte gelötet. Ein weiterer Fehler der an mehreren ICs auftrat war, dass einige IC Anschlüssen kurzgeschlossen waren. Hier wurde mit Entlötlitze und Flussmittel das überschüssige Lötzinn entfernt.

9 Inbetriebnahme

9.1 Messung der Versorgungsspannungen

Als alle signifikanten Fehler behoben waren, wurde die erste Leiterplatte der LIS Variante einer Funktionskontrolle unterzogen. Es wurde zunächst eine Spannungsversorgung mit einstellbarer Strombegrenzung an den Autopiloten angeschlossen. Die Strombegrenzung wurde auf 80mA eingestellt und die Spannung langsam auf 5V hochgefahren. Jedoch erreichte der Strom schon bei ca. 3,3V die Strombegrenzung. Die Strombegrenzung wurde auf 200mA erhöht. Auch hier konnte die Spannung 3,3V nicht überschreiten. Nach einigen Untersuchungen wurde ein Kurzschluss an 5 Anschlüssen des Magnetfeldsensors festgestellt. Nachdem der Kurzschluss behoben war, wurde erneut eine Spannung an den Autopiloten angelegt. Die Strombegrenzung wurde wieder auf 100mA eingestellt. Nun wurden die 5V problemlos erreicht. Die Stromaufnahme betrug 43 mA. Es wurden alle Versorgungsspannungen gemessen. Sie sind in Tabelle 9.1 dargestellt und lagen alle in der erlaubten Toleranz.

Tabelle 9.1: Messung der Versorgungsspannungen, Platine 1 LIS Variante

Versorgungsspannung	**U_{soll} in V**	**U_{ist} in V**
5V digital	5	5,11
5V analog	5	5,11
3,3V digital	3,3	3,41
3,3V analog	3,3	3,40
3,3V Gyro	3,3	3,41

Dann wurde versucht, den Mikrocontroller urzuladen. Hierzu wurde der Autopilot über einen Programmieradapter an einen PC angeschlossen. Das Urladen scheiterte allerdings. Es stellte sich heraus, dass der Quarz falsch bestellt wurde und ein Oszillator war. Der Oszillator wurde, da er in einem QFN Gehäuse montiert ist, unter Wärmezufuhr über sein Gehäuse von der Leiterplatte entlötet und der neue Quarz auf gleiche Weise wieder auf die Platine gelötet. Ein Funktionstest zeigte, dass die Stromaufnahme auf der 3,3V Analogebene zu hoch war. Die Strombegrenzung stand bei 200mA. Zudem war die Amplitude des Quarztaktes zu klein. Nachdem kein sichtbarer Fehler festgestellt werden konnte, wurde die 3,3V Analogspannungsversorgung von der Leiterplatte gelötet. Eine Widerstandsmessung zwischen 3,3V analog und analog Masse ergab 18Ω. Erwartet wurden mehrere kΩ. Eine weitere Messung zeigte, dass durch den 1Ω Widerstand, der die digitale und die analoge Masse miteinander verbindet, ein Strom von 180mA floss. Dies ist ein unzulässig hoher Strom. Da jedoch die Fehlerhafte Stelle nicht ermittelt werden konnte, wurde die zweite Leiterplatte der LIS Variante herangezogen.
Auch hier wurde ein neuer Quarz montiert. Mit dieser Leiterplatte wurde ebenso ein Funktionstest durchgeführt und wie mit der ersten Platine verfahren. Hier waren alle Spannungen im erlaubten Bereich. Die Stromaufnahme betrug 30mA. Die Messwerte sind in Tabelle 9.2 wiederzufinden. Die Messung mit einem Oszilloskop zeigte ein einwandfrei funktionierender Quarztakt.

Tabelle 9.2: Messung der Versorgungsspannungen, Platine 2 LIS Variante

Versorgungsspannung	**U_{soll} in V**	**U_{ist} in V**
5V digital	5	4,92
5V analog	5	4,88
3,3V digital	3,3	3,28
3,3V analog	3,3	3,28
3,3V Gyro	3,3	3,32

Nachdem die Funktion der Platine 2 LIS gegeben und kontrolliert war, wurde die Leiterplatte 1 IDG für einen Funktionstest, nach oben genanntem Schema, vorbereitet. Auch hier wiesen alle Versorgungsspannungen die richtigen Pegel auf. Sie sind in Tabelle 9.3 aufgelistet.

Tabelle 9.3: Messung der Versorgungsspannungen, Platine 1 IDG Variante

Versorgungsspannung	**U_{soll} in V**	**U_{ist} in V**
5V digital	5	4,90
5V analog	5	4,92
3,3V digital	3,3	3,35
3,3V analog	3,3	3,29
3,3V Gyro	3,3	3,38

9.2 Fehler während der Inbetriebnahme

9.2.1 Bestellfehler

Der Quarz wurde falsch bestellt. Da der richtige Quarz noch ausreichend vorrätig war, konnten die Quarze auf den Leiterplatten einfach und schnell getauscht werden.
Ebenso ist der Luftdrucksensor MPXA6115A falsch, da die Bezeichnung in der EAGLE Bibliothek falsch eingetragen wurde. Der zu verwendende Sensor soll ein MPXH6115A sein und wurde in der EAGLE Bibliothek geändert. Auch dieser Sensor war noch in der Hochschule Bremen vorrätig und konnte problemlos nachbestückt werden.

9.2.2 Schaltungsfehler

Ein Fehler in der Schaltung ist die nicht normgerechte Anschlussbelegung der USB Steckerleiste. Hier sind die Signale USB+ und USB- vertauscht. Dieser Fehler wird behoben, indem im Kabel die beiden Signalleitungen getauscht werden.
Die Gyrospannungsversorgung wurde auf den LIS Leiterplatten nicht an die Gyroskope angeschlossen, sondern an 3,3V analog. Es wird die Leitung 3,3V analog an den Gyroskopen aufgetrennt und mit einem Draht auf die 3,3V Gyrospannung gelegt. Ein weiterer Fehler ist, dass das Bootloadsignal nicht auf die Steckerleiste X1 gelegt ist. Durch eine kurze Kontaktherstellung mit der Bootloadleitung zu dem Bootloadsignal auf der Leiterplatte ist der Fehler behoben.

9.2.3 Layoutfehler

Da die Package Datei für den Magnetfeldsensor gespiegelt erstellt wurde, muss der Sensor von allen bestehenden Leiterplatten entfernt werden. Der Magnetfeldsensor wird auf einer externen Leiterplatte, dem sogenannten Breakout board von Sparkfun Electronics [3], über den I^{2}C1 Bus an den Autopiloten angeschlossen. Die Leitungen des I^{2}C1 Bus wird auf den Stecker X9 gelegt und ist in Bild 9.1 zu sehen.

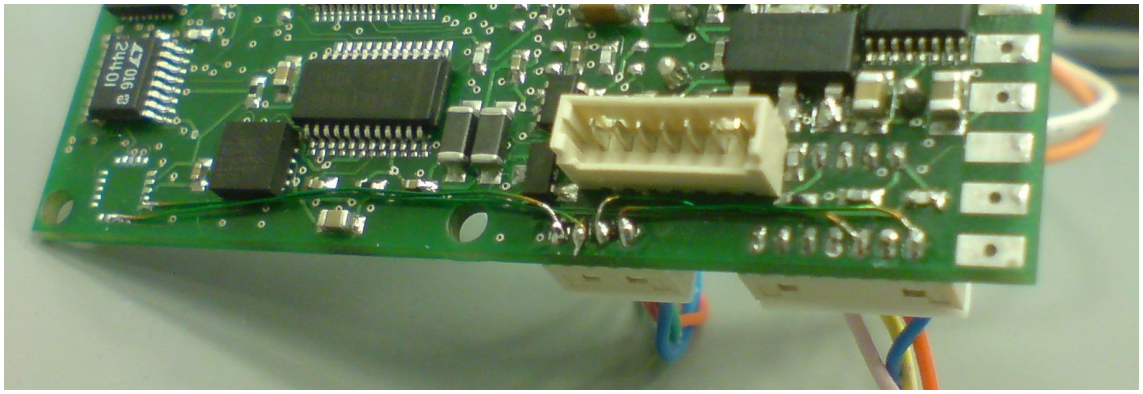


Abbildung 9.1: I^{2}C1 Bus auf Stecker x9 gelegt

Der Enableanschluss des I^{2}C0 Levelshifter wurde auf GND gelegt. Er muss über einen 200kΩ Widerstand an +5V liegen. Zum Beheben des Fehlers wurden die Leiterbahnen aufgetrennt und mit einem 200kΩ Widerstand und einer Drahtbrücke richtig angeschlossen, wie in Bild 9.2 zu sehen ist.

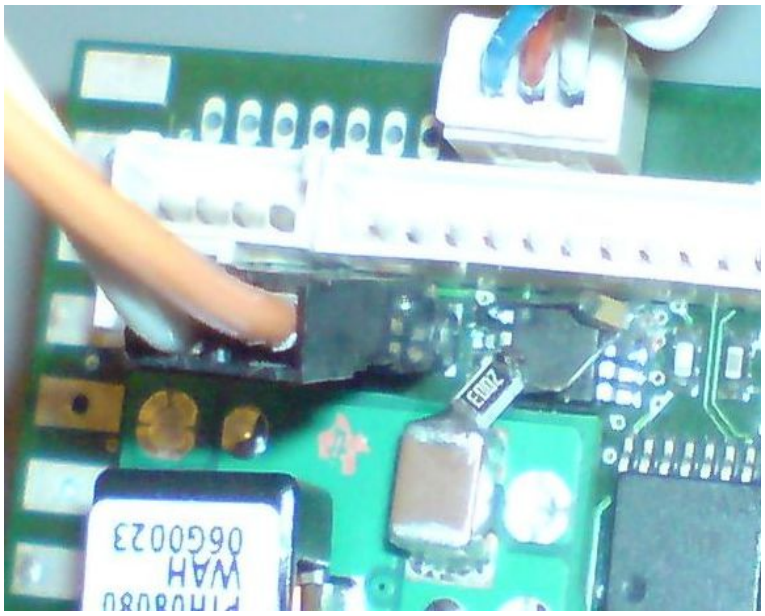


Abbildung 9.2: Korrektur am I2C0 Levelshifter

9.3 Flashen des Mikrocontrollers

Der Mikroprozessor konnte urgeladen werden. Danach wurde ein bereits bestehendes Demoprogramm in den Mikrocontroller geladen, welches die beiden LEDs ansteuert. Es zeigte sich, dass sich beide LEDs wie erwartet verhielten. In einem zweiten Demoprogramm wurde die serielle Schnittstelle UART0 getestet. Es wurden Zeichensätze vom PC an den Autopiloten gesendet. Auch hier waren keine Fehlfunktionen zu beobachten. Ebenso funktionierte die Kommunikation über den USB Bus einwandfrei. Daraufhin wurde der Paparazzi Quellcode [4] für den Miniaturautopilot umgeschrieben und eine Simulation der Gyroskop- und Beschleunigungssensoren im Paparazzi Center durchgeführt. Die Auswertung der Sensordaten erfolgte über die 10Bit Analogeingänge des Mikrocontrollers.

9.4 Test der Sensoren

Der Autopilot wurde einer manuellen Dreh-, Roll- und Yaw-Bewegung ausgesetzt und die Sensordaten grafisch dargestellt. Die Darstellungen zeigen die Abbildungen 9.3 und 9.4. Es zeigte sich, dass eine Funktion gegeben war. Die richtige Zuordnung der Achsen und der Abgleich der Offsets wurde dann in der Software realisiert. Im nächsten Versuch wurde die Datenübertragung per Funk kontrolliert. Das Funkmodul Radiotronix wurde am UART0 Stecker des Autopiloten angeschlossen. Sämtliche Daten wurden einwandfrei an das Empfängermodul übertragen. Dann wurde die I^2C0 Software aus dem HB-Autopiloten Quellcode in den Mikrocontroller geladen. Der I^2C0 Bus dient zum ansteuern der Drehstromsteller, der die vier Motoren steuert.

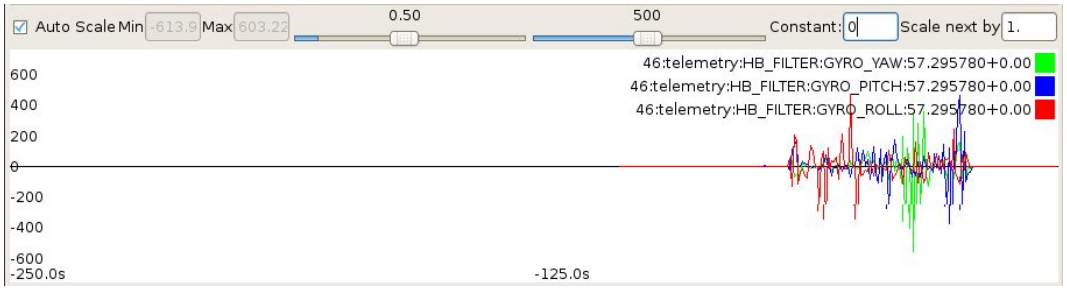


Abbildung 9.3: Grafische Darstellung der Gyrosensorwerte

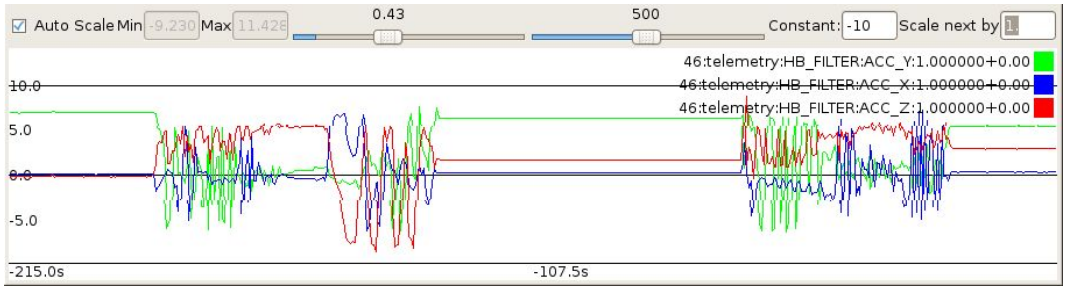


Abbildung 9.4: Grafische Darstellung der Beschleunigungssensorwerte

9.5 Montage des Autopilot auf Quadrokopter

Nachdem die Funktion der Sensoren verifiziert war, wurde der Autopilot in einer flüchtigen Montage auf dem bestehenden Quadrokopter eingesetzt und ein erster Testflug absolviert. Der Aufbau ist in Abbildung 9.5 zu sehen. Der Testflug fand im Raum E402 der Hochschule Bremen statt. Der Quadrokopter war ca. 15 Sekunden in der Luft und schwebte ca. 30cm über dem Fußboden. Das Flugverhalten war stabil in der Luft, driftete jedoch leicht ab.

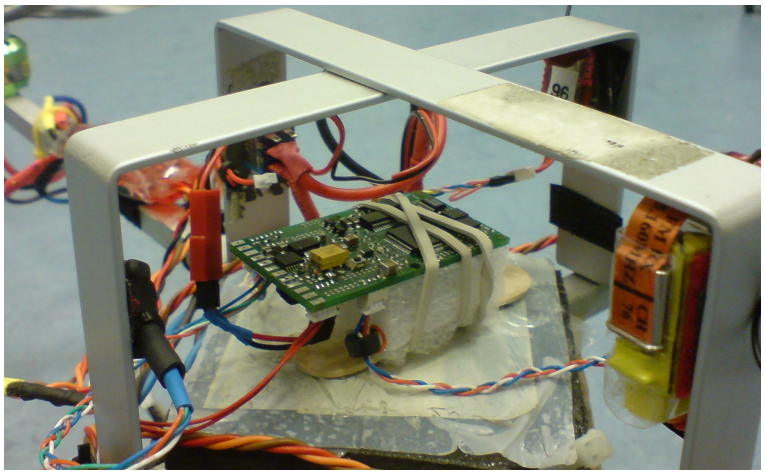


Abbildung 9.5: Autopilot auf Quadrokopter

10 Softwareanpassung

10.1 Anpassen der Software

Bisher wurden alle Sensordaten nur über die Mikrocontroller internen 10 Bit Umsetzer erfasst. Um eine 16 Bit Auflösung zu bekommen, muss die bereits bestehende Software vom HB-Autopiloten an den Miniaturautopiloten angepasst werden. Da bei der Anschlussbelegung des Mikrocontrollers darauf geachtet wurde, dass möglichst die gleichen Anschlüsse für die gleiche Funktion gewählt wurde, konnte die bestehende Software ohne Änderungen verwendet werden. Bei Anschlüssen, die für andere Signale oder andere Schnittstellen verwendet wurden, wird die Software angepasst. Softwareänderungen bzw. Erweiterungen sollen so erfolgen, dass die Software in die bereits bestehende Tiny13 und HBC (HB-Copter Quellcode) Software integriert wird und umschaltbar sein soll. Das Umschalten erfolgt mit dem Befehl *#define Tiny13*, bzw. *#define HBC* oder *#define HBMINI*. Die SPI1 Schnittstelle muss neu programmiert werden. Die Schnittstelle dient zur Kommunikation mit dem 16 Bit- sowie dem 24 Bit ADC. In den vorherigen Projekten wurde die SPI0 Schnittstelle für die Kommunikation verwendet. Die Software für den Buzzer, die Servosteuerung, dem Chip Select und dem Cam power switch mussten nur so geändert werden, dass die richtigen Anschlüsse am Microkontroller konfiguriert werden. Der Quellcode an sich blieb unverändert. Die Quellcodes sind im Anhang zu finden.

10.2 Test der geänderten Software

Nach dem die Software angepasst war, wurde die neue Software getestet. Zum Testen des Buzzers wurde ein Buzzer an die Steckerleiste X8 angeschlossen. Der Buzzer erzeugte wie erwartet einen hörbaren Ton.
Das Testen der Servosteuerung erfolgte mithilfe eines Oszilloskop. Jedes Servosignal wurde mit einer Funkfernbedienung geändert und Oszilloskopiert. Jeder Servokanal lies sich mit der Funkfernbedienung ansteuern.
Die SPI1 Schnittstelle und der Chipselect wurden so getestet, dass die Sensorsignale von Beschleunigungs- und Gyroskopsensoren aufgenommen wurden. Während des Aufnehmens wurde die Leiterplatte einer manuellen Drehbewegung ausgesetzt und die aufgenommenen Werte in einem Gnuplot Diagramm grafisch dargestellt. Die Sensorwerte sind in Abbildung 10.1 zu sehen. Alle Werte sind mit einem Offset behaftet. Der Offsetabgleich erfolgte in einem weiteren Schritt der Softwareanpassung.

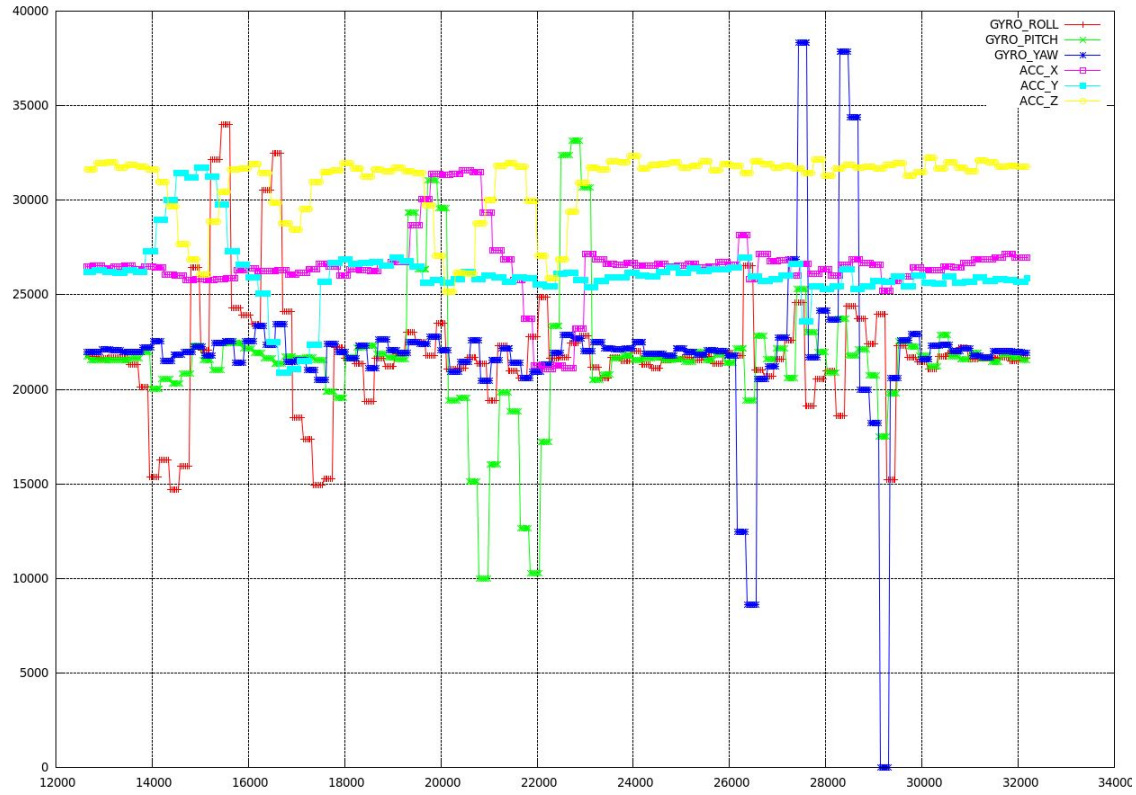


Abbildung 10.1: Grafische Darstellung der Sensorwerte in 16 Bit Auflösung

Die Funktion der Sensoren und die richtige Zuordnung der Sensorkanäle ist somit bewiesen. Es wurde nun die Funktion des Luftdrucksensors und die Software für diesen Sensor getestet. In Abbildung 10.2 ist die grafische Darstellung des Luftdrucksensors (schwarze Kennlinie) sowie alle anderen Sensorwerte mit Offsetabgleich zu sehen. Zum Test des Luftdrucksensors wurde die Leiterplatte einer manuellen Auf und Ab Bewegung ausgesetzt. Der Höhenunterschied betrug ca. 50 cm.

Der I^2C1 Bus, der für die Kommunikation mit dem Magnetfeldsensor zuständig ist, wurde ebenfalls getestet. Der Test zeigte eine Funktion der x- und y-Achse. Die z-Achse blieb ohne Funktion. Eine Messung der Spannung am Ladekondensator ergab 1,8 Volt, so wie im Datenblatt angegeben. Somit kann die Ursache auf die Software reduziert werden. Da die die Fehlersuche im Programm zu zeitaufwendig war, konnte der Fehler nicht gefunden und behoben werden.

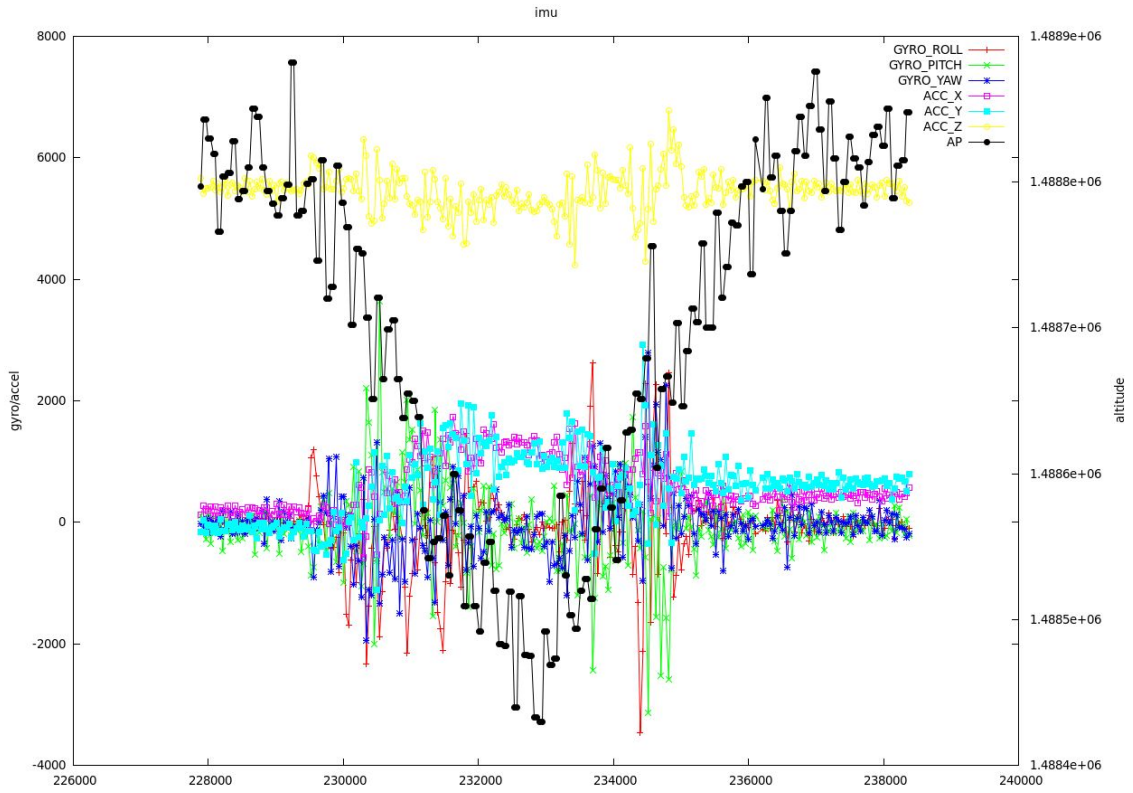


Abbildung 10.2: Test des Luftdrucksensor und Offsetabgleich

11 Flugbetrieb

Nach dem erfolgreichen Test der Software wurde der Autopilot auf einem Quadrokopter montiert. Die Montage erfolgte wie es bereits in Kapitel 9.5 und in der Abbildung 9.5 zu sehen ist. Es folgte ein Flug mit der geänderten Software. Der Flug fand im Foyer des Elektrotechnikgebäudes der Hochschule Bremen statt. Der Quadrokopter flog stabil und kontrolliert per Funkfernbedienung. Der Flug ist in Abbildung 11.1 zu sehen. Somit ist mit dieser Arbeit eine praktische Eignung nachgewiesen.

Abbildung 11.1: Autopilot im Flugbetrieb auf Quadrokopter

12 Preiskalkulation

Um den Preis eines Autopiloten kalkulieren zu können, muss zunächst festgelegt werden, welche Faktoren in die Rechnung mit einbezogen werden sollen.
Es wird davon ausgegangen, dass nur die Bauteilekosten sowie die Herstellung der Leiterplatten zu berücksichtigen sind. Des weiteren kann die maschinelle Bestückung bei einem externen Leiterplattenbestücker erfolgen. Laut Online Preiskalkulator von PCB-POOL [5] vom 09.11.2010, liegt der Preis für eine Leiterplatte bei 134,28 Euro. Da der Hersteller jedoch fünf Leiterplatten aus einem Leiterplattenrohling bekommt, wird kein weiterer Aufpreis für diese fünf Leiterplatten fällig. Somit beläuft sich der Preis pro Leiterplatte auf 26,86 Euro.
In Tabelle 12.1 sind alle Kosten zusammengefasst, zuzüglich Versandkosten. Im Anhang ist eine detaillierte Auflistung der einzelnen Bauelemente aufgeführt.

Tabelle 12.1: Zusammenstellung aller Kosten in Euro

	Leiterplatte LIS	**Leiterplatte IDG**
Leiterplatte	26,86	26,86
Material	144,60	154,29
Gesamt	**171,46**	**181,15**

13 Zusammenfassung

Mit dieser Arbeit konnte ein planarer Miniaturautopilot, für den Einsatz auf Flächen- sowie Schwebefluggeräte, entwickelt und erprobt werden. Alle Forderungen in der Aufgabenstellung wurden realisiert. So befinden sich die Funktionen Messen und Verarbeiten auf nur einer Leiterplatte. In früheren Projekten wurden für die Funktionen noch zwei verschiedene Leiterplatten benötigt. Bei der Entwicklung des Autopiloten wurde eine räumliche Trennung von analogen und digitalen Bauteilen realisiert, um die Gefahr von Störungen der analogen Signale zu minimieren. Ein erfolgreicher Testflug mit einem bereits bestehendem Quadrokopter wurde absolviert. Die Bestückung der Leiterplatten kann manuell, halbautomatisch oder automatisch erfolgen. Die Kosten für einen Autopiloten ohne Bestückung belaufen sich auf 171 Euro, bzw. 181 Euro.

Die Sensoren und Schnittstellen wurden erfolgreich getestet. Mit den vorhandenen Schnittstellen und freien Chipselectanschlüsse auf dem SPI Stecker ist es möglich, weitere Module an den Autopiloten anzuschließen. Ferner ist es möglich, über Lötbrücken den Messbereich der Gyroskope zu wählen. Der Messbereich des Bechleunigungssensors LIS344ALH kann über den Mikrocontroller gewählt werden. Ein GPS Modul lässt sich auf dem Autopiloten montieren, so dass möglichst wenig Platz für Autopilot und GPS Modul in Anspruch genommen wird. Hervorzuheben sind eine geringere Masse von 11,8 g und ein geringerer Herstellungsaufwand des Autopiloten. So soll dieser Autopilot in zukünftigen IMAV (International Micro Air Vehicle conference an flight competition) Wettbewerben, aufgrund seiner geringen Masse, eingesetzt werden.

Die Software wurde im Quellcode des HB-Autopiloten implementiert. So wurde der an der Hochschule Bremen entwickelte Quadrokopterquellcode des HB-Autopilot nicht verändert, sondern um den des Miniaturautopiloten erweitert. Mit dem Quellcode kann zwischen HB-Autopilot und Miniaturautopilot gewählt werden. Auch der in Toulouse für Flächenfluggeräte entwickelte Paparazzi Quellcode, eignet sich für den Flugbetrieb des Autopiloten.

Zusätzlich wurden zwei Varianten von Drehratensensoren eingesetzt. Welche Sensoren besser geeignet sind, müssen zukünftige Projekte klären. Auch für zukünftige Projekte wurde die Version v0.2 in EAGLE bereits entwickelt. In der Version v0.2 sind alle Schaltungsfehler behoben und kann ohne Weiteres bestückt, getestet und in Betrieb genommen werden.
Der Autopilot wurde bereits im August 2010 anlässlich des ES0802 Workshops in Cambridge, Großbritannien veröffentlicht.

14 Anhang

14.1 Literaturverzeichnis

[1] Diplomarbeit, Andreas Dei, SS2009, Hochschule Bremen

[2] http://www.freerouting.net

[3] http://www.sparkfun.com/products/9371

[4] http://paparazzi.enac.fr/wiki/Main_Page

[5] http://www.pcb-pool.com/ppde/order_productconfiguration_js.html

[D1] Datenblatt, Philips Semiconductors, LPC2148, Rev.01

[D2] Datenblatt, Honeywell, HMC5843

[D3] Datenblatt, Freescale Semicondutor, MPXH6115A, Rev.5

[D4] Datenblatt, Linear Technology, LTC2440, Rev.D

[D5] Datenblatt, ST Microelectronics, LPR530AL, Rev.1

[D6] Datenblatt, ST Microelectronics, LY530ALH, Rev.1

[D7] Datenblatt, InvenSense, IDG500, Rev.B

[D8] Datenblatt, InvenSense, ISZ500, Rev.B

[D9] Datenblatt, ST Microelectronics, LIS334ALH,Rev.3

[D10] Datenblatt, Analog Devices, ADXL335, Rev.A

[D11] Datenblatt, MAXIM, MAX1168, Rev.0

[D12] Datenblatt, Texas Instruments, PTH08080W, Rev.C

[D13] Datenblatt, Linear Technology, LT1117, Rev.C

[D14] Datenblatt, Linear Technology, LT1761, Rev.A

[D15] Datenblatt, Philips, 74HC138, Rev.3

[D16] Datenblatt, Texas Instruments, TPS2051B, Rev.K

[D17] Datenblatt, Philips NXP, PCA9306DP, Rev.2

[D18] Datenblatt, Microchip, 24LC64, Rev.R

[D19] Datenblatt, Navilock, NL-507ETTL, Rev.1

14.2 Abkürzungsverzeichnis

GNDA	Analoge Masse
GND	Digitale Masse

PCB	Printed Circuit Board
JTAG	Joint Test Action Group
I/O	Input/Output
ADC	Analog-Digital-Converter
I^2C	Inter integrated Circuit
MEMS	Micro Electro Mechanical System
SPI	Serial Peripheral Interface
USB	Universal Serial Bus
GPS	Global Positioning System
SMD	Surface Mounted Device
IC	Integrated Circuit
QFN	Quad Flat No Leads
SSOP	Super Small Outline Package
ESR	Equivalent Series Resistance
mil	1mil = 1/1000 inch = 25,4 μm

14.3 Fertig aufgebauter Autopilot

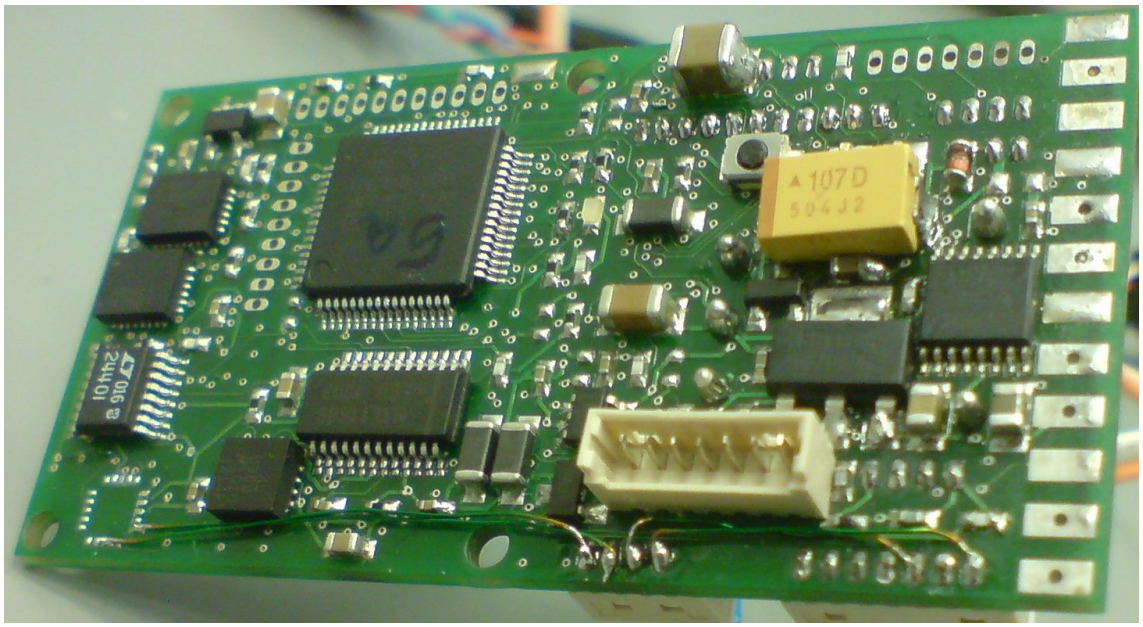


Abbildung 14.1: Oberseite des aufgebauten Autopiloten

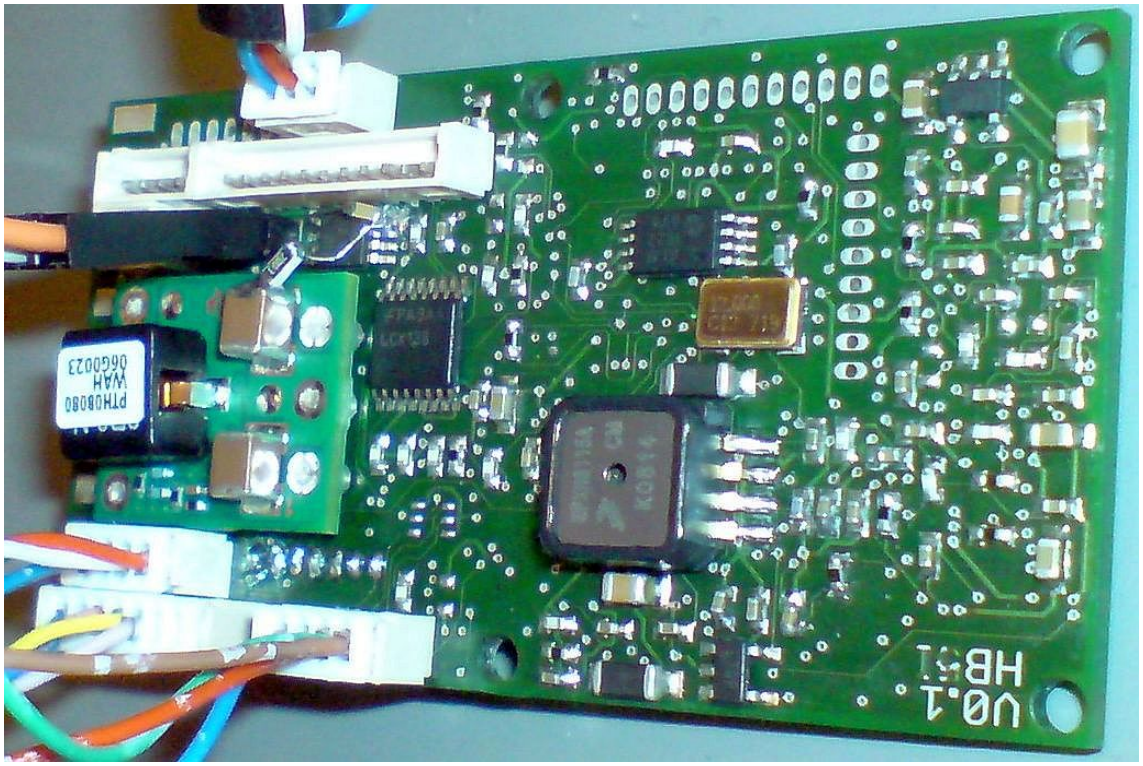
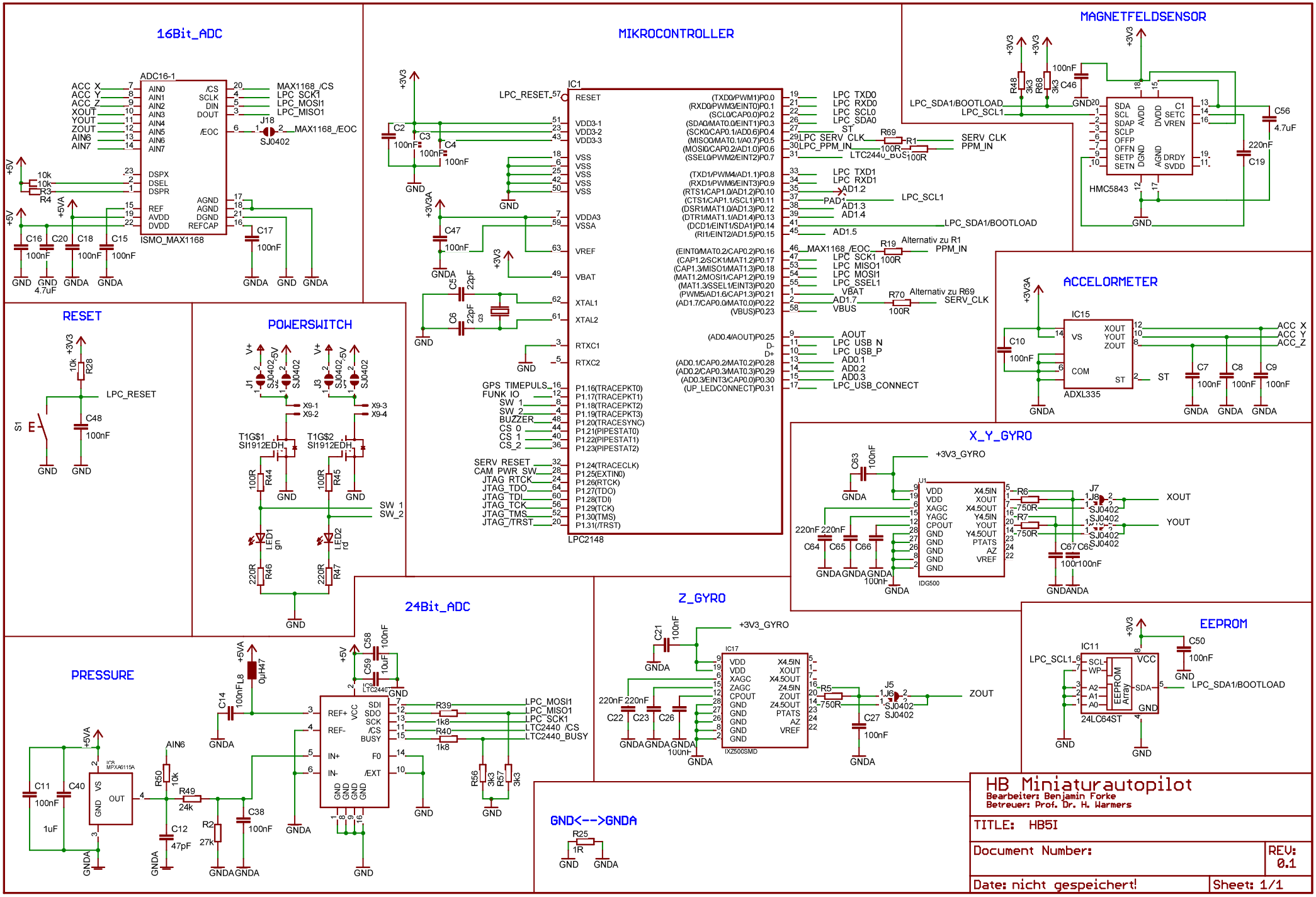
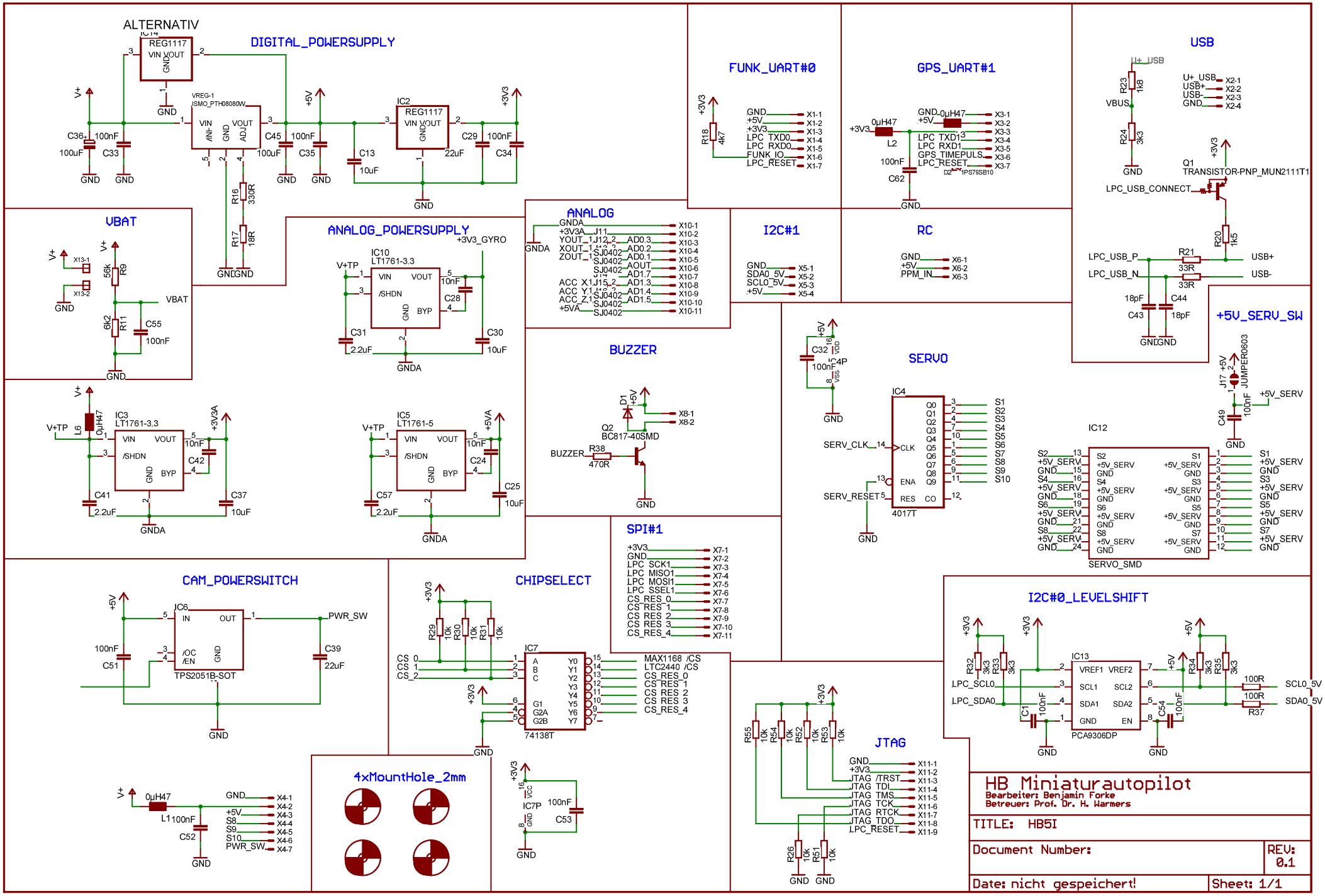
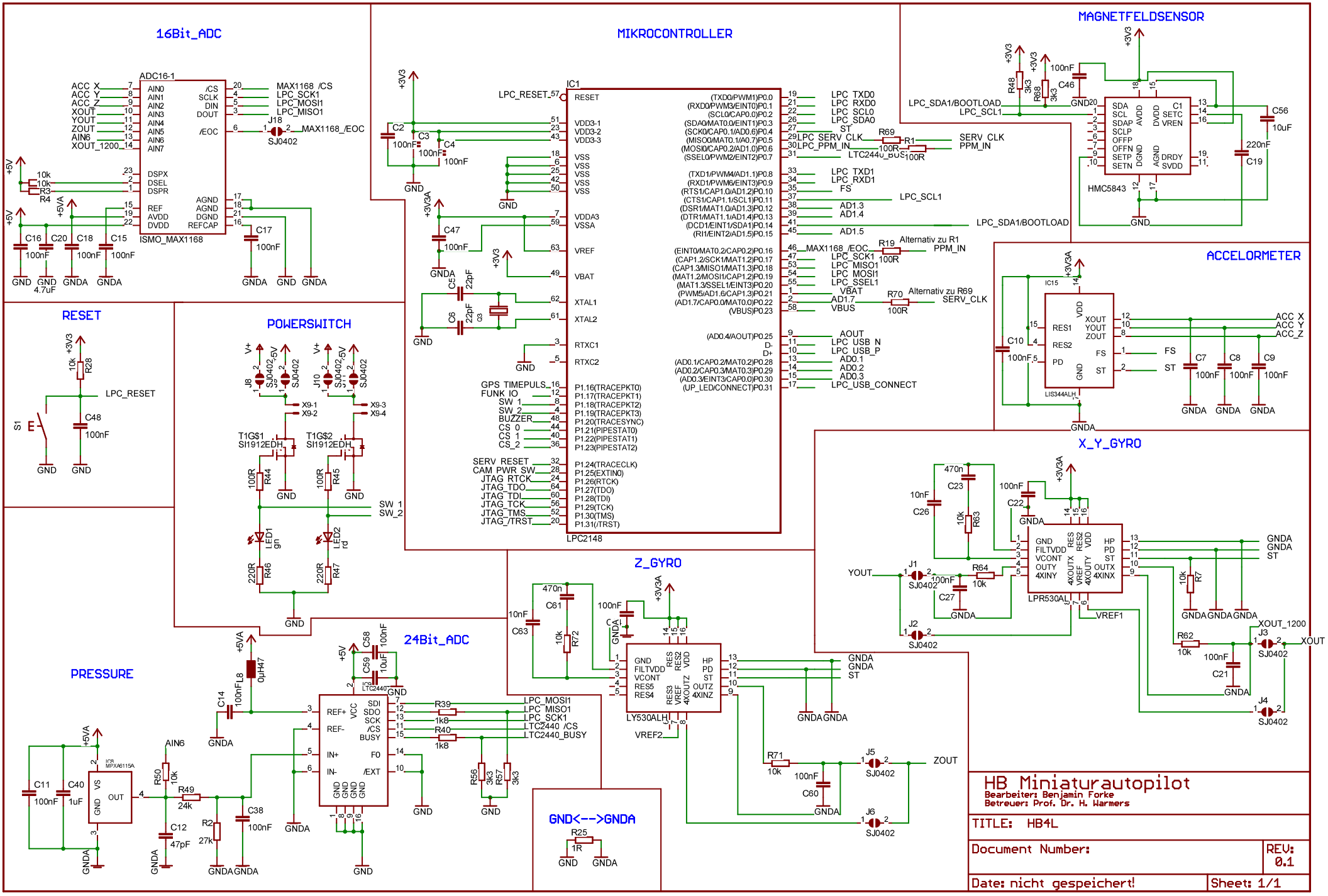


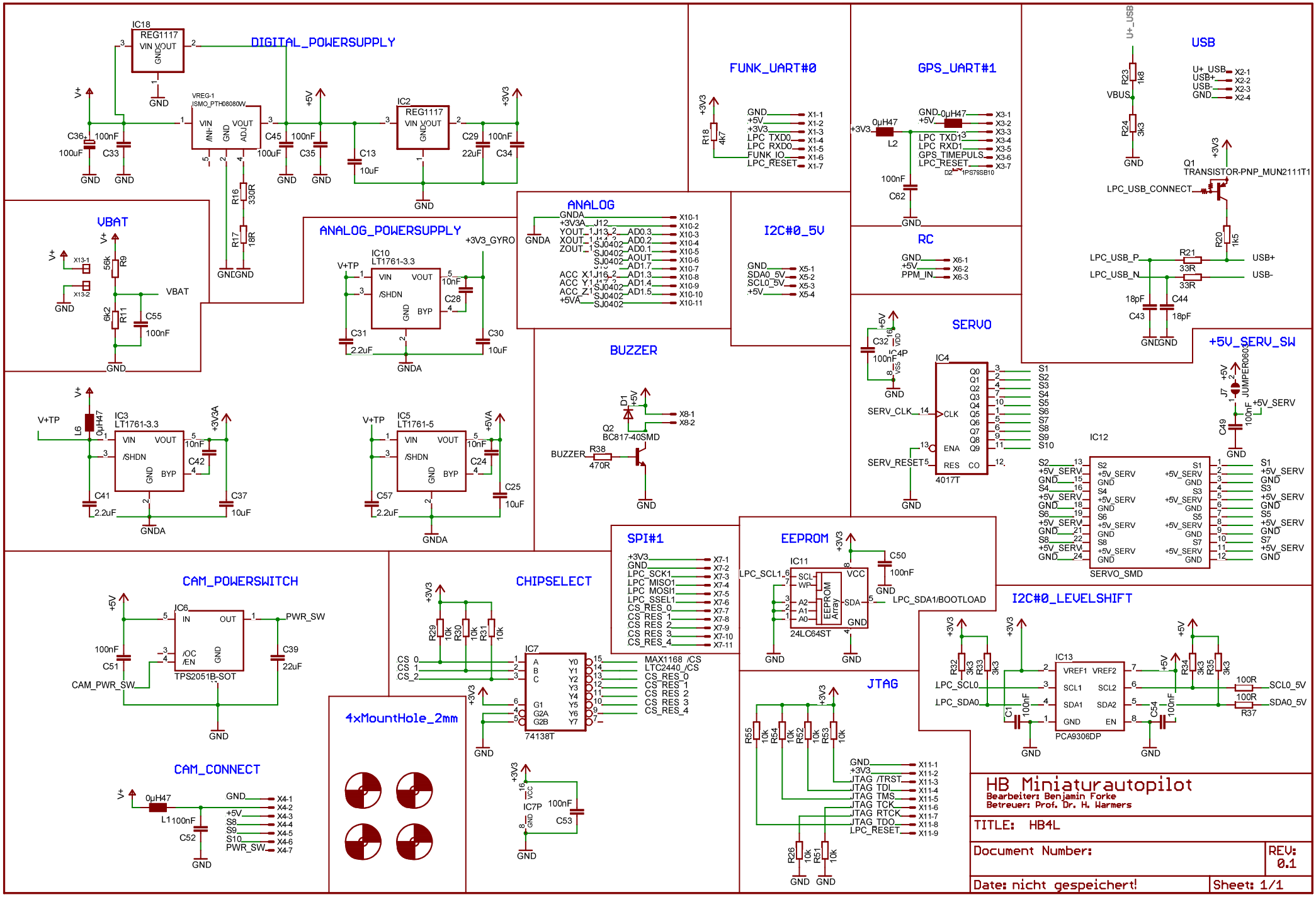
Abbildung 14.2: Unterseite des aufgebauten Autopiloten

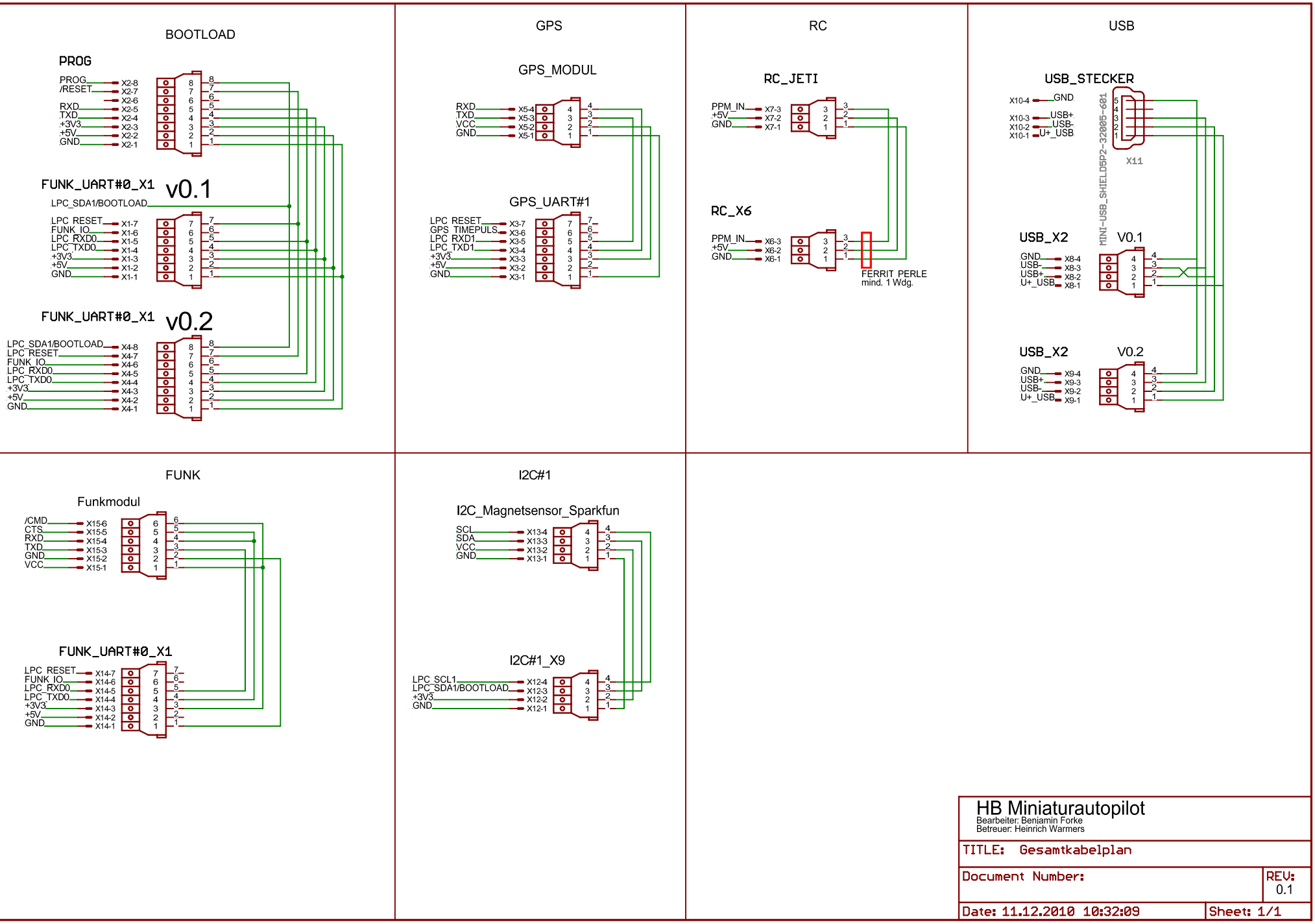
14.4 Schaltpläne v0.1


16Bit_ADC
MIKROCONTROLLER
MAGNETFELDSENSOR
RESET
POWERSWITCH
ACCELORMETER
X_Y_GYRO
Z_GYRO
24Bit_ADC
EEPROM
PRESSURE
GND<-->GNDA
HB Miniaturautopilot
Bearbeiter: Benjamin Forke
Betreuer: Prof. Dr. H. Warmers
TITLE: HB5I
Document Number:
REV: 0.1
Date: nicht gespeichert!
Sheet: 1/1


ALTERNATIV
DIGITAL_POWERSUPPLY
REG1117
VREG-1
ISMO_PTH08080W
FUNK_UART#0
GPS_UART#1
USB
VBAT
ANALOG_POWERSUPPLY
IC10
LT1761-3.3
ANALOG
I2C#1
RC
BUZZER
SERVO
IC4
4017T
+5V_SERV_SW
IC12
SERVO_SMD
IC3
IC5
LT1761-5
SPI#1
CAM_POWERSWITCH
TPS2051B-SOT
CHIPSELECT
IC7
74138T
MAX1168 /CS
LTC2440 /CS
I2C#0_LEVELSHIFT
PCA9306DP
JTAG
4xMountHole_2mm
HB Miniaturautopilot
Bearbeiter: Benjamin Forke
Betreuer: Prof. Dr. H. Warmers
TITLE: HB5I
Document Number:
REV: 0.1
Date: nicht gespeichert!
Sheet: 1/1


16Bit_ADC
MIKROCONTROLLER
MAGNETFELDSENSOR
ACCELORMETER
RESET
POWERSWITCH
X_Y_GYRO
Z_GYRO
24Bit_ADC
PRESSURE
GND<-->GNDA
HB Miniaturautopilot
Bearbeiter: Benjamin Forke
Betreuer: Prof. Dr. H. Warmers
TITLE: HB4L
Document Number:
REV: 0.1
Date: nicht gespeichert!
Sheet: 1/1


DIGITAL_POWERSUPPLY
FUNK_UART#0
GPS_UART#1
USB
VBAT
ANALOG_POWERSUPPLY
ANALOG
I2C#0_5V
RC
SERVO
+5V_SERV_SW
BUZZER
SPI#1
EEPROM
CHIPSELECT
CAM_POWERSWITCH
I2C#0_LEVELSHIFT
JTAG
CAM_CONNECT
4xMountHole_2mm
HB Miniaturautopilot
Bearbeiter: Benjamin Forke
Betreuer: Prof. Dr. H. Warmers
TITLE: HB4L
Document Number:
REV: 0.1
Date: nicht gespeichert!
Sheet: 1/1


BOOTLOAD
PROG
PROG
/RESET
RXD
TXD
+3V3
+5V
GND
X2-8
X2-7
X2-6
X2-5
X2-4
X2-3
X2-2
X2-1
FUNK_UART#0_X1
v0.1
LPC_SDA1/BOOTLOAD
LPC_RESET
FUNK_IO
LPC_RXD0
LPC_TXD0
+3V3
+5V
GND
X1-7
X1-6
X1-5
X1-4
X1-3
X1-2
X1-1
FUNK_UART#0_X1
v0.2
LPC_SDA1/BOOTLOAD
LPC_RESET
FUNK_IO
LPC_RXD0
LPC_TXD0
+3V3
+5V
GND
X4-8
X4-7
X4-6
X4-5
X4-4
X4-3
X4-2
X4-1
GPS
GPS_MODUL
RXD
TXD
VCC
GND
X5-4
X5-3
X5-2
X5-1
GPS_UART#1
LPC_RESET
GPS_TIMEPULS
LPC_RXD1
LPC_TXD1
+3V3
+5V
GND
X3-7
X3-6
X3-5
X3-4
X3-3
X3-2
X3-1
RC
RC_JETI
PPM_IN
+5V
GND
X7-3
X7-2
X7-1
RC_X6
PPM_IN
+5V
GND
X6-3
X6-2
X6-1
FERRIT_PERLE
mind. 1 Wdg.
USB
USB_STECKER
X10-4
X10-3
X10-2
X10-1
GND
USB+
USB-
U+_USB
X11
MINI-USB_SHIELDSP2-32005-601
USB_X2
V0.1
GND
USB-
USB+
U+_USB
X8-4
X8-3
X8-2
X8-1
USB_X2
V0.2
GND
USB+
USB-
U+_USB
X9-4
X9-3
X9-2
X9-1
FUNK
Funkmodul
/CMD
CTS
RXD
TXD
GND
VCC
X15-6
X15-5
X15-4
X15-3
X15-2
X15-1
FUNK_UART#0_X1
LPC_RESET
FUNK_IO
LPC_RXD0
LPC_TXD0
+3V3
+5V
GND
X14-7
X14-6
X14-5
X14-4
X14-3
X14-2
X14-1
I2C#1
I2C_Magnetsensor_Sparkfun
SCL
SDA
VCC
GND
X13-4
X13-3
X13-2
X13-1
I2C#1_X9
LPC_SCL1
LPC_SDA1/BOOTLOAD
+3V3
GND
X12-4
X12-3
X12-2
X12-1
HB Miniaturautopilot
Bearbeiter: Benjamin Forke
Betreuer: Heinrich Warmers
TITLE: Gesamtkabelplan
Document Number:
REV: 0.1
Date: 11.12.2010 10:32:09
Sheet: 1/1

14.5 Layouts v0.1

14.5.1 LIS Variante

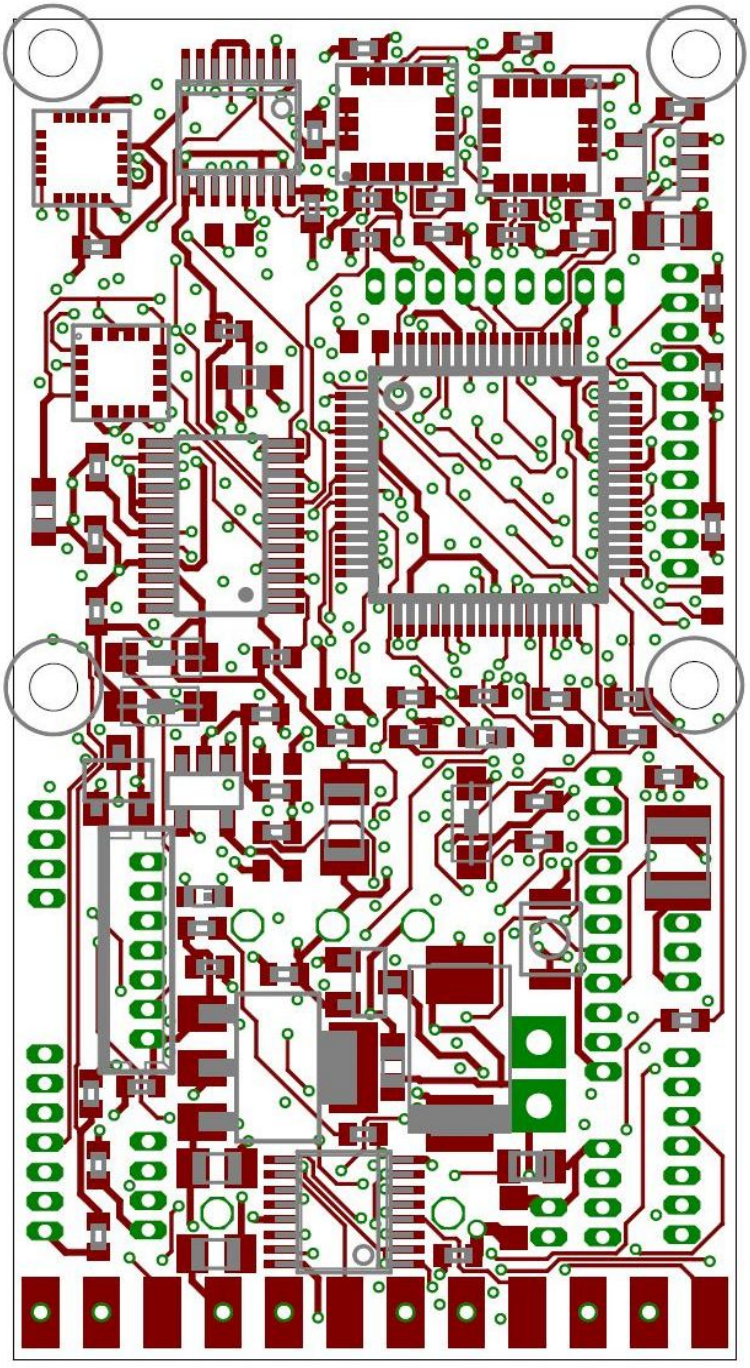


Abbildung 14.3: Layout Oberseite der LIS Variante

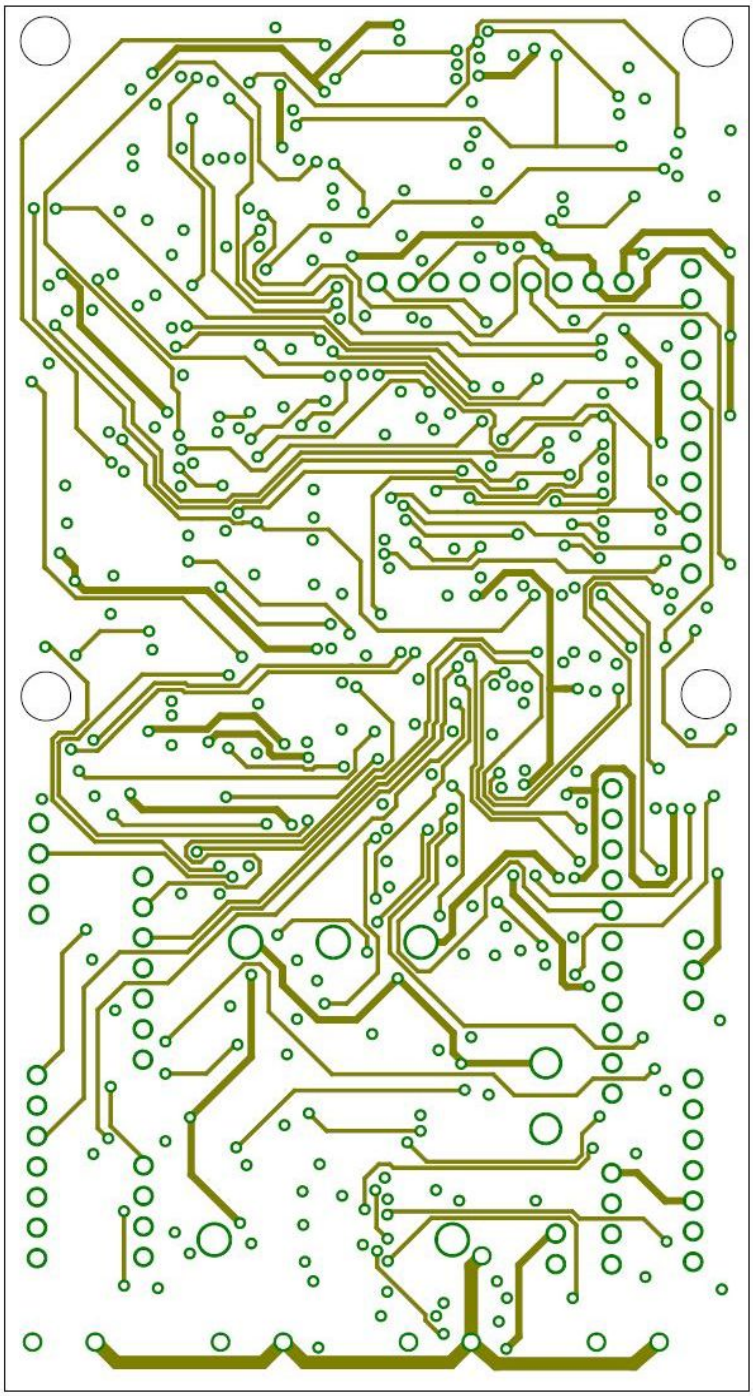


Abbildung 14.4: Layout obere Innenlage der LIS Variante

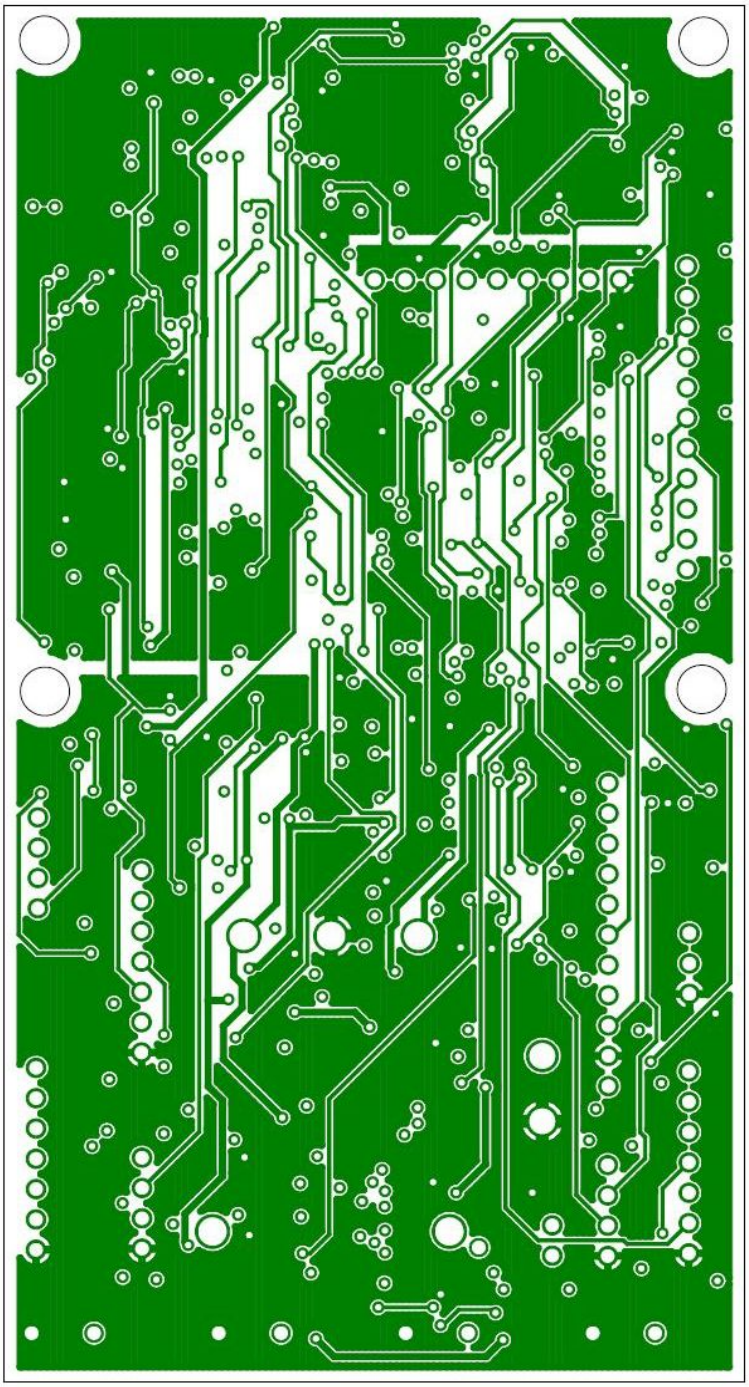


Abbildung 14.5: Layout untere Innenlage (GND Layer) der LIS Variante

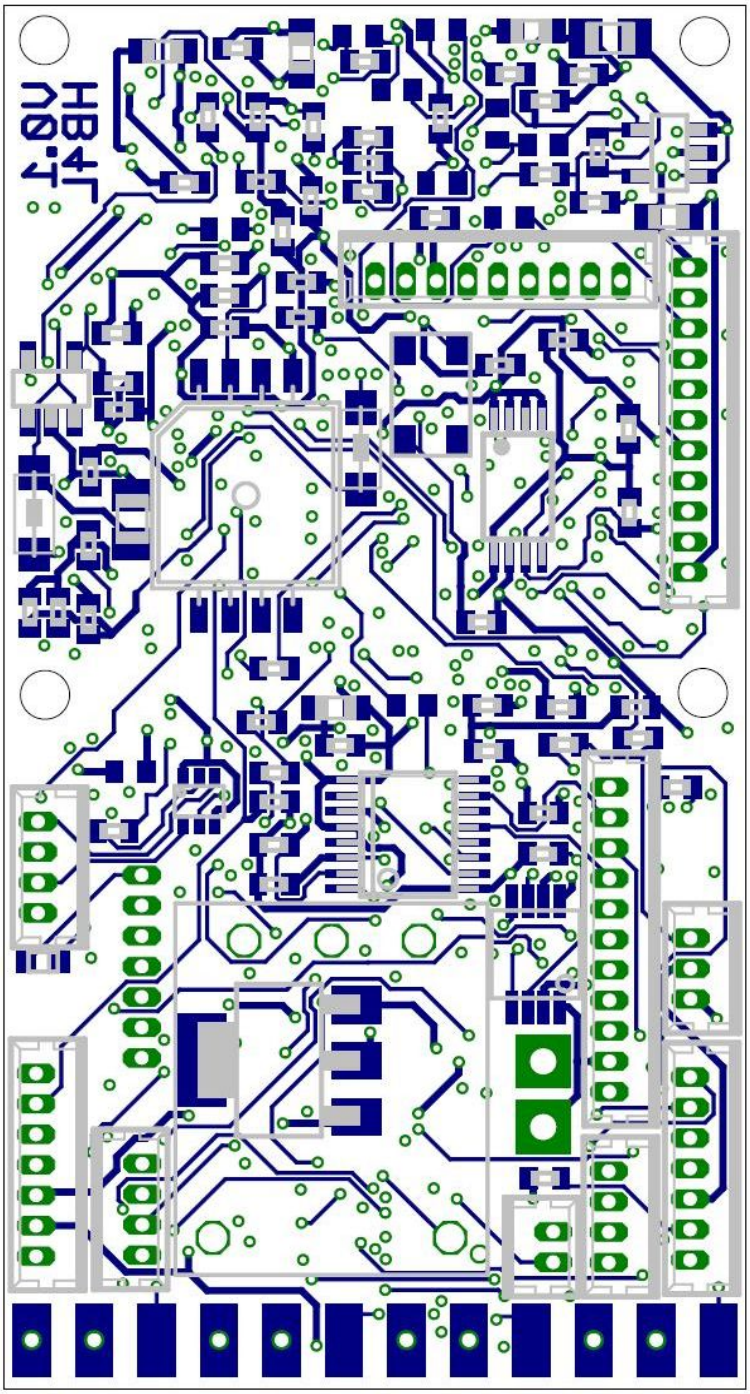


Abbildung 14.6: Layout Unterseite der LIS Variante

14.5.2 IDG Variante

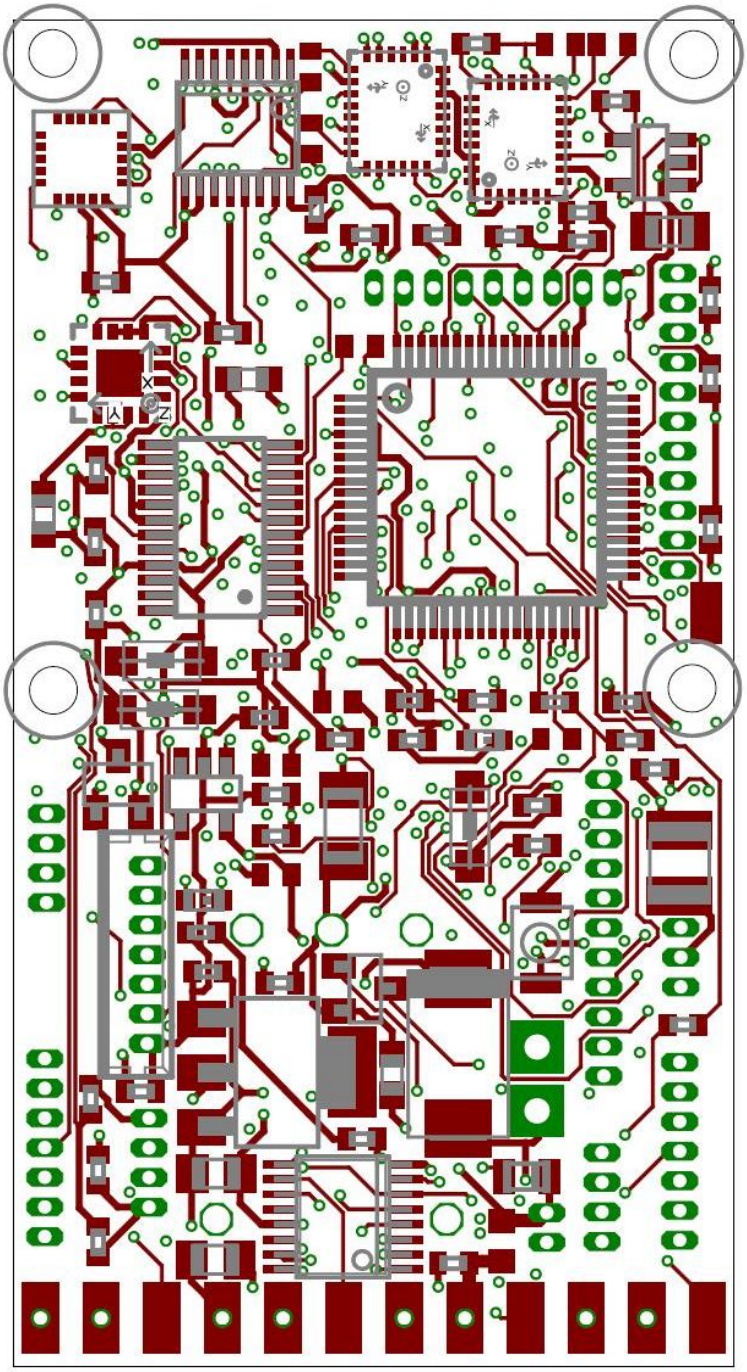


Abbildung 14.7: Layout Oberseite der IDG Variante

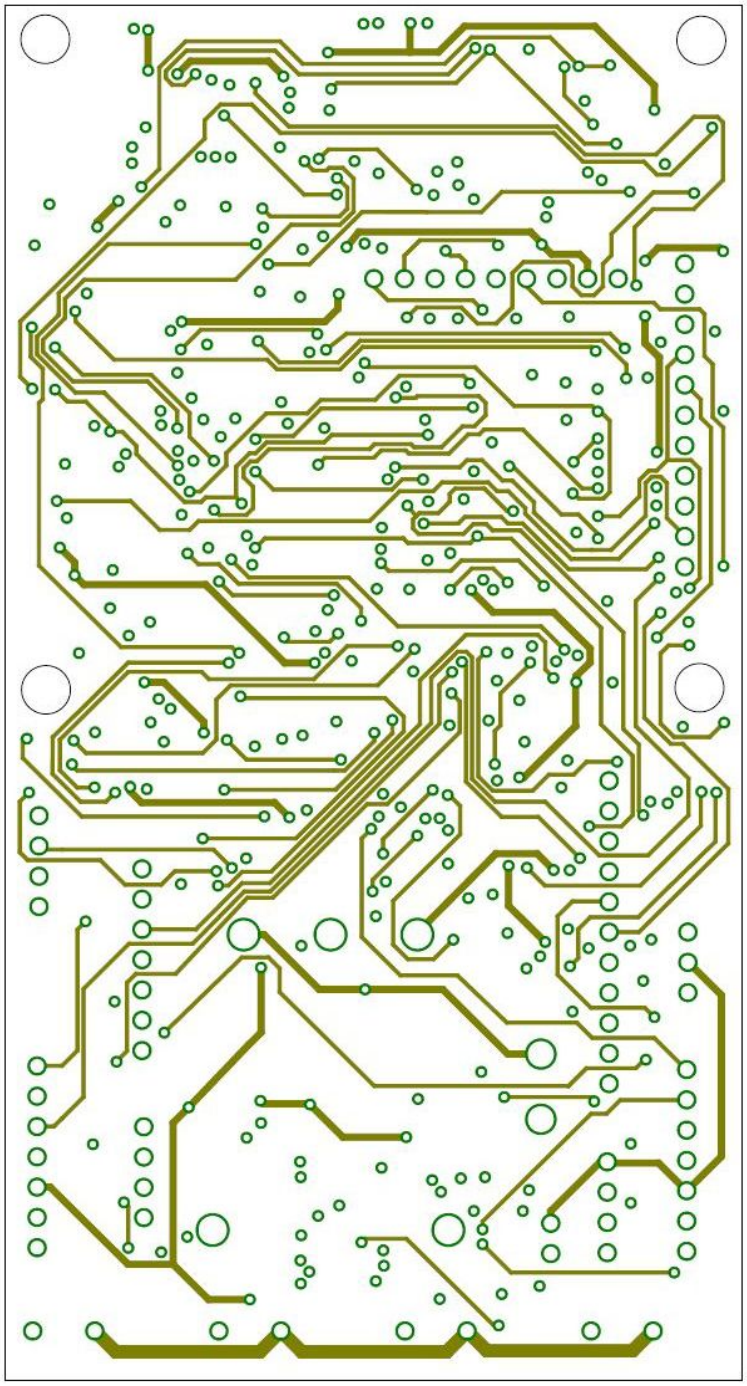


Abbildung 14.8: Layout obere Innenlage der IDG Variante

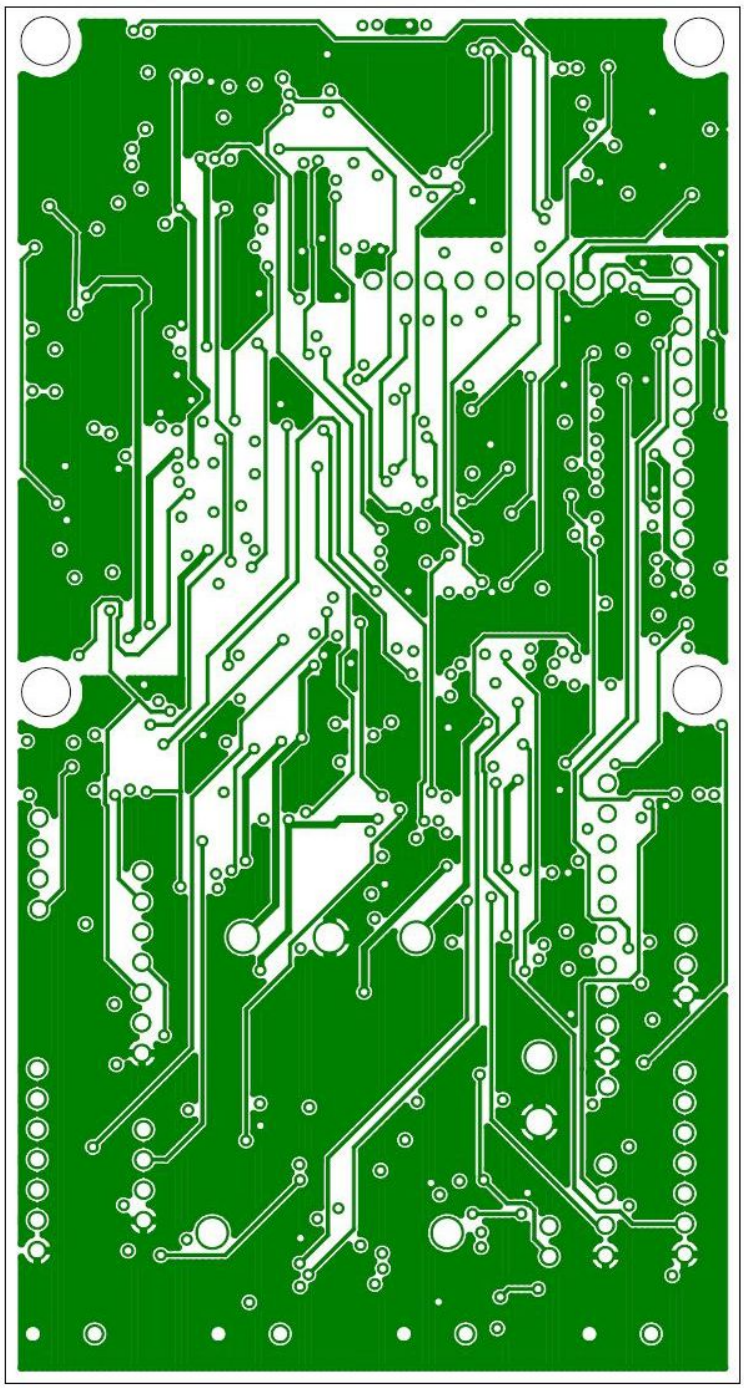


Abbildung 14.9: Layout untere Innenlage (GND Layer) der IDG Variante

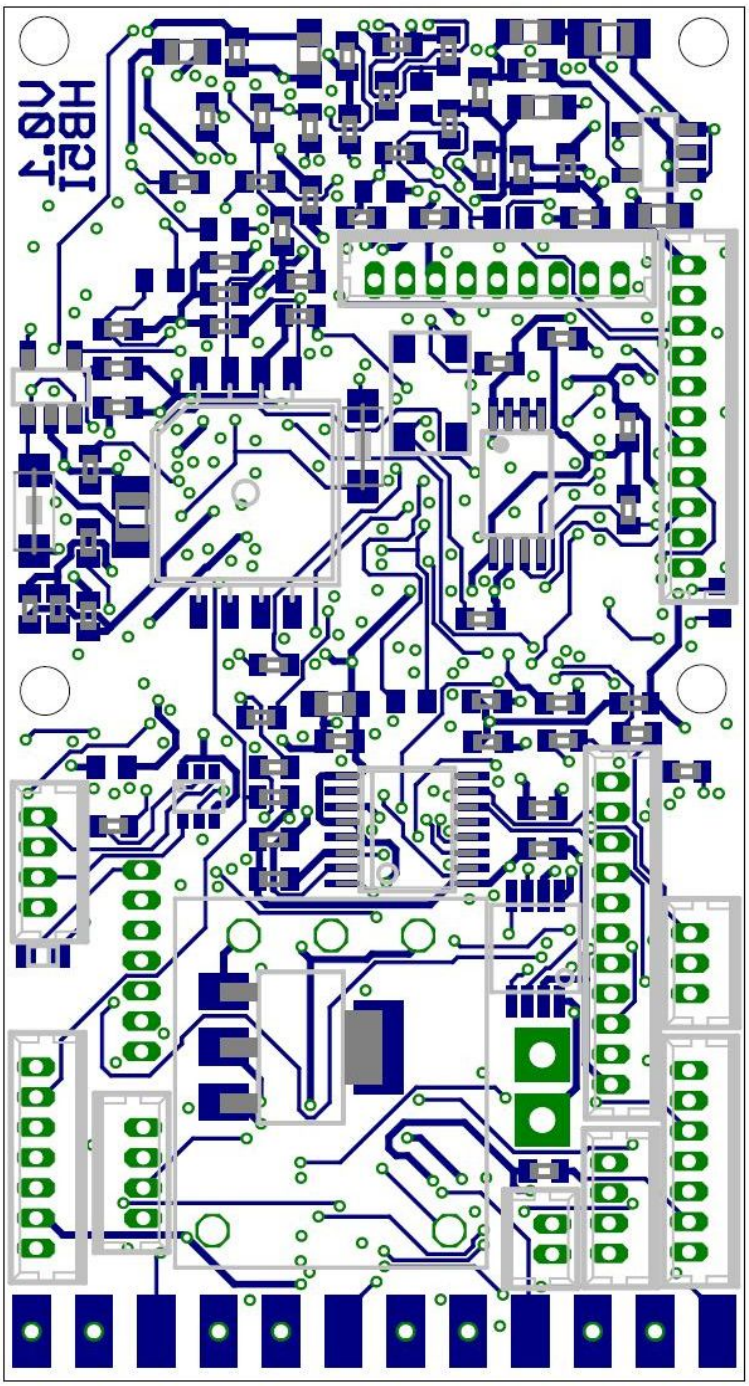
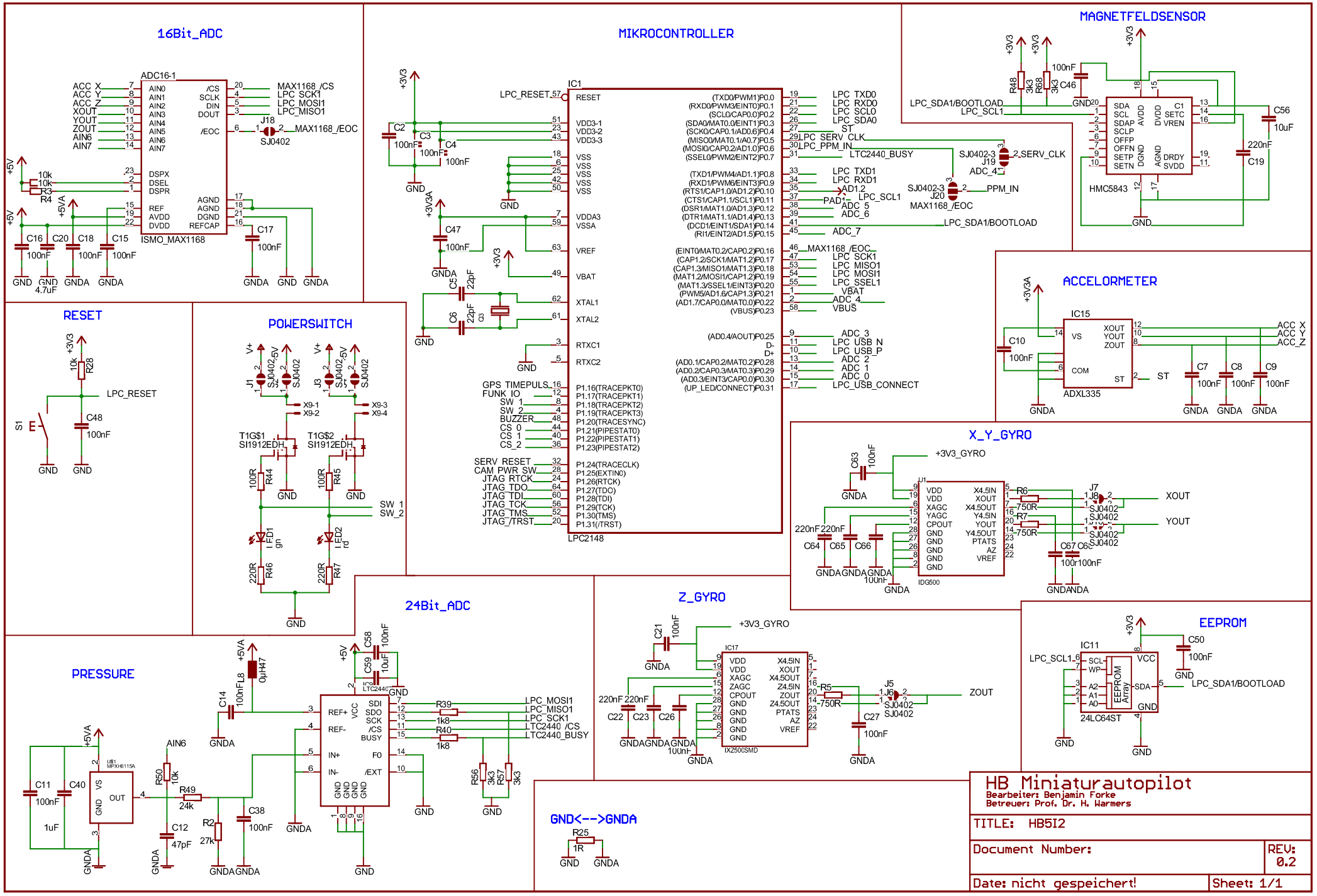
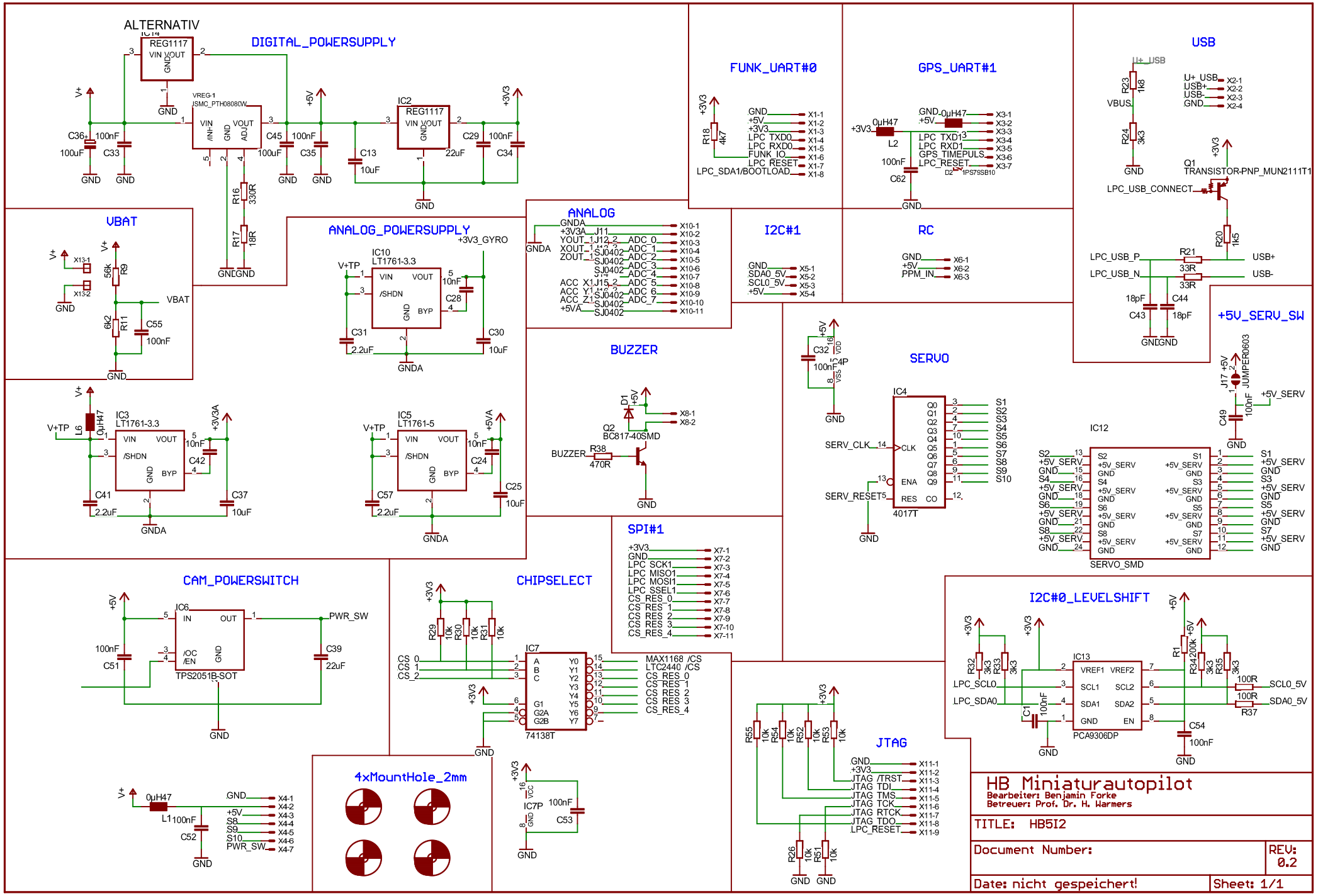
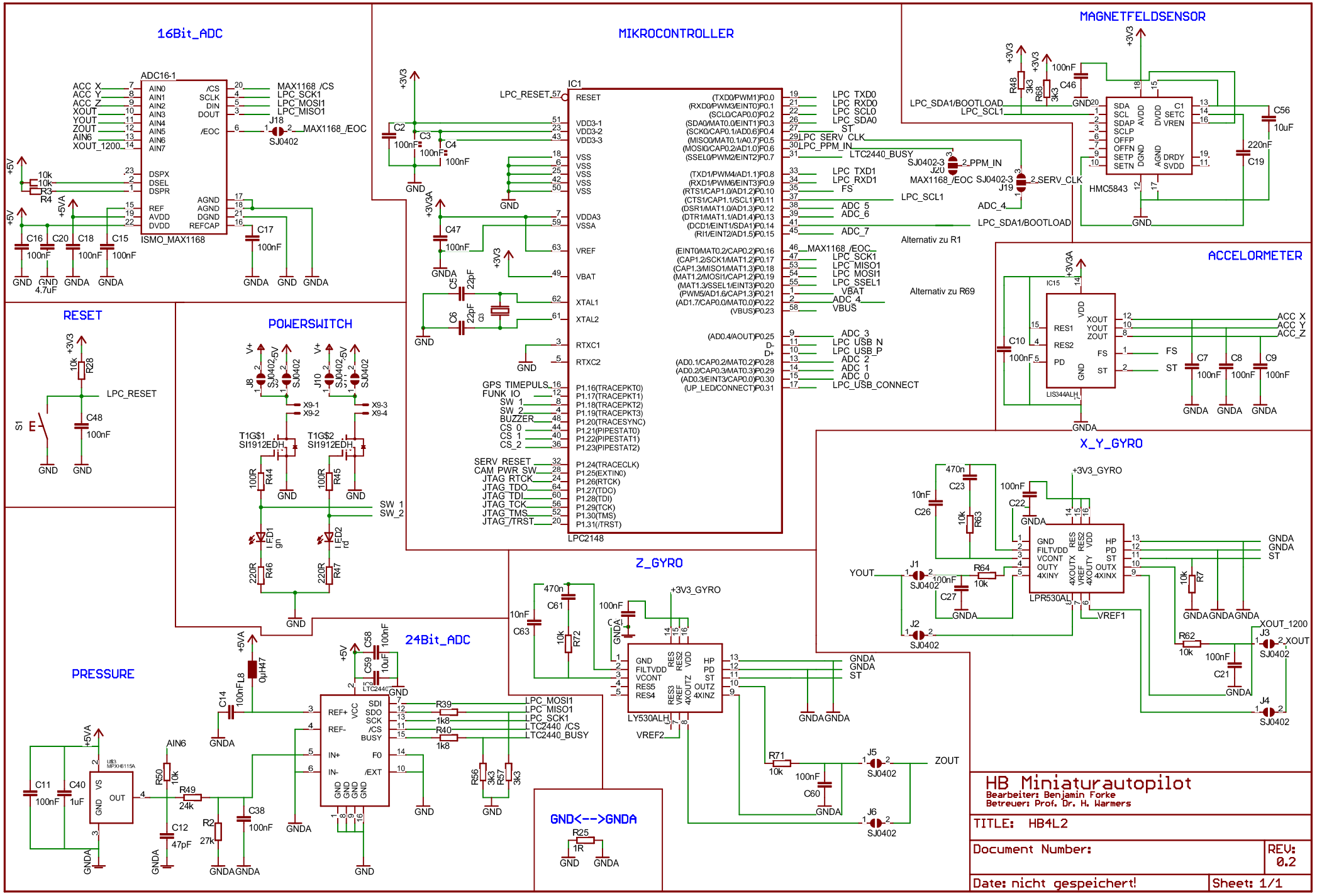


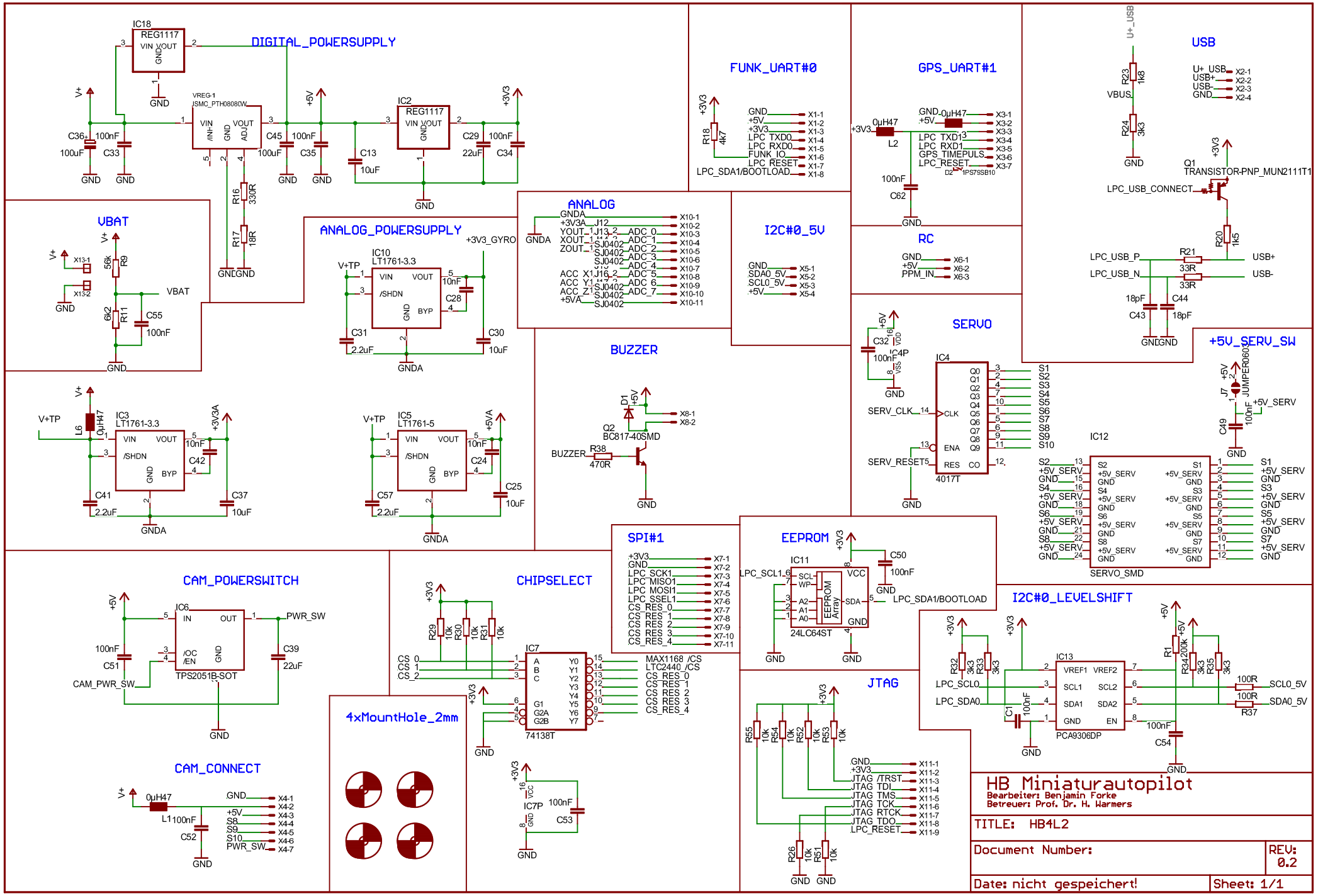
Abbildung 14.10: Layout Unterseite der IDG Variante

14.6 Schaltpläne v0.2


16Bit_ADC
MIKROCONTROLLER
MAGNETFELDSENSOR
ACCELORMETER
RESET
POWERSWITCH
X_Y_GYRO
24Bit_ADC
Z_GYRO
EEPROM
PRESSURE
GND<-->GNDA
ADC16-1
ISMO_MAX1168
IC1
LPC2148
IC15
ADXL335
IC17
IDG500
IXZ500SMD
IC11
24LC64ST
HB Miniaturautopilot
Bearbeiter: Benjamin Forke
Betreuer: Prof. Dr. H. Warmers
TITLE: HB5I2
Document Number:
REV: 0.2
Date: nicht gespeichert!
Sheet: 1/1


ALTERNATIV
DIGITAL_POWERSUPPLY
REG1117
VIN VOUT
GND
VREG-1
ISMC_PTH08080W
VIN VOUT
/INH GND ADJ
C36 100uF
C33 100nF
C45 100uF
C35 100nF
C13 10uF
C29 22uF
C34 100nF
R16 330R
R17 18R
+5V
+3V3
GND
FUNK_UART#0
R18 4k7
GND
+5V
+3V3
LPC_TXD0
LPC_RXD0
FUNK_IO
LPC_RESET
LPC_SDA1/BOOTLOADER
X1-1
X1-2
X1-3
X1-4
X1-5
X1-6
X1-7
X1-8
GPS_UART#1
0uH47
L2
100nF
C62
LPC_TXD1
LPC_RXD1
GPS_TIMEPULS
LPC_RESET
D2 1PS79SB10
X3-1
X3-2
X3-3
X3-4
X3-5
X3-6
X3-7
USB
U+_USB
R23 1k8
R24 3k3
VBUS
USB+
USB-
X2-1
X2-2
X2-3
X2-4
Q1 TRANSISTOR-PNP_MUN2111T1
LPC_USB_CONNECT
R20 1k5
LPC_USB_P
LPC_USB_N
R21 33R
R22 33R
C43 18pF
C44 18pF
VBAT
R9 56k
R11 6k2
C55 100nF
X13-1
X13-2
ANALOG_POWERSUPPLY
IC10
LT1761-3.3
+3V3_GYRO
C28 10nF
C31 2.2uF
C30 10uF
ANALOG
GNDA
+3V3A
YOUT_1
XOUT_1
ZOUT_1
ACC_X1
ACC_Y1
ACC_Z1
+5VA
ADC_0
ADC_1
ADC_2
ADC_3
ADC_4
ADC_5
ADC_6
ADC_7
X10-1
X10-2
X10-3
X10-4
X10-5
X10-6
X10-7
X10-8
X10-9
X10-10
X10-11
I2C#1
SDA0_5V
SCL0_5V
X5-1
X5-2
X5-3
X5-4
RC
PPM_IN
X6-1
X6-2
X6-3
+5V_SERV_SW
J17 JUMPER0603
C49 100nF
+5V_SERV
IC3
LT1761-3.3
L6 10uH47
C42 10nF
C41 2.2uF
C37 10uF
IC5
LT1761-5
C24 10nF
C57 2.2uF
C25 10uF
BUZZER
D1
Q2 BC817-40SMD
R38 470R
SERVO
IC4
C32 100nF
C34
SERV_CLK
SERV_RESET
4017T
S1
S2
S3
S4
S5
S6
S7
S8
S9
S10
IC12
SERVO_SMD
SPI#1
LPC_SCK1
LPC_MISO1
LPC_MOSI1
LPC_SSEL1
CS_RES_0
CS_RES_1
CS_RES_2
CS_RES_3
CS_RES_4
X7-1
X7-2
X7-3
X7-4
X7-5
X7-6
X7-7
X7-8
X7-9
X7-10
X7-11
CAM_POWERSWITCH
IC6
TPS2051B-SOT
PWR_SW
C51 100nF
C39 22uF
CHIPSELECT
IC7
74138T
R29 10k
R30 10k
R31 10k
MAX1168 /CS
LTC2440 /CS
I2C#0_LEVELSHIFT
IC13
PCA9306DP
R32 3k3
R33 3k3
R34 200k
R35 3k3
R37 100R
C1 100nF
C54 100nF
SCL0_5V
SDA0_5V
JTAG
R55 10k
R54 10k
R52 10k
R53 10k
R26 10k
R51 10k
JTAG /TRST
JTAG TDI
JTAG TMS
JTAG TCK
JTAG RTCK
JTAG TDO
4xMountHole_2mm
IC7P
C53 100nF
L1 0uH47
C52 100nF
X4-1
X4-2
X4-3
X4-4
X4-5
X4-6
X4-7
HB Miniaturautopilot
Bearbeiter: Benjamin Forke
Betreuer: Prof. Dr. H. Warmers
TITLE: HB5I2
Document Number:
REV: 0.2
Date: nicht gespeichert!
Sheet: 1/1


16Bit_ADC
MIKROCONTROLLER
MAGNETFELDSENSOR
ACCELORMETER
RESET
POWERSWITCH
X_Y_GYRO
Z_GYRO
24Bit_ADC
PRESSURE
GND<-->GNDA
ADC16-1
ISMO_MAX1168
IC1
LPC2148
LTC2440_BUSY
Alternativ zu R1
Alternativ zu R69
HB Miniaturautopilot
Bearbeiter: Benjamin Forke
Betreuer: Prof. Dr. H. Warmers
TITLE: HB4L2
Document Number:
REV: 0.2
Date: nicht gespeichert!
Sheet: 1/1


DIGITAL_POWERSUPPLY
IC18
REG1117
VIN VOUT GND
VREG-1
ISMC_PTH08080W
IC2
REG1117
FUNK_UART#0
GPS_UART#1
USB
U+_USB
USB+
USB-
GND
VBUS
LPC_USB_CONNECT
LPC_USB_P
LPC_USB_N
TRANSISTOR-PNP_MUN2111T1
VBAT
ANALOG_POWERSUPPLY
IC10
LT1761-3.3
ANALOG
I2C#0_5V
RC
SERVO
IC4
4017T
SERV_CLK
SERV_RESET
+5V_SERV_SW
JUMPER060
IC12
SERVO_SMD
BUZZER
D1
Q2
BC817-40SMD
IC3
LT1761-3.3
IC5
LT1761-5
SPI#1
EEPROM
IC11
24LC64ST
I2C#0_LEVELSHIFT
IC13
PCA9306DP
CAM_POWERSWITCH
IC6
TPS2051B-SOT
PWR_SW
CAM_PWR_SW
CHIPSELECT
IC7
74138T
MAX1168 /CS
LTC2440 /CS
JTAG
CAM_CONNECT
4xMountHole_2mm
IC7P
HB Miniaturautopilot
Bearbeiter: Benjamin Forke
Betreuer: Prof. Dr. H. Warmers
TITLE: HB4L2
Document Number:
REV: 0.2
Date: nicht gespeichert!
Sheet: 1/1

14.7 Layouts v0.2

14.7.1 LIS Variante

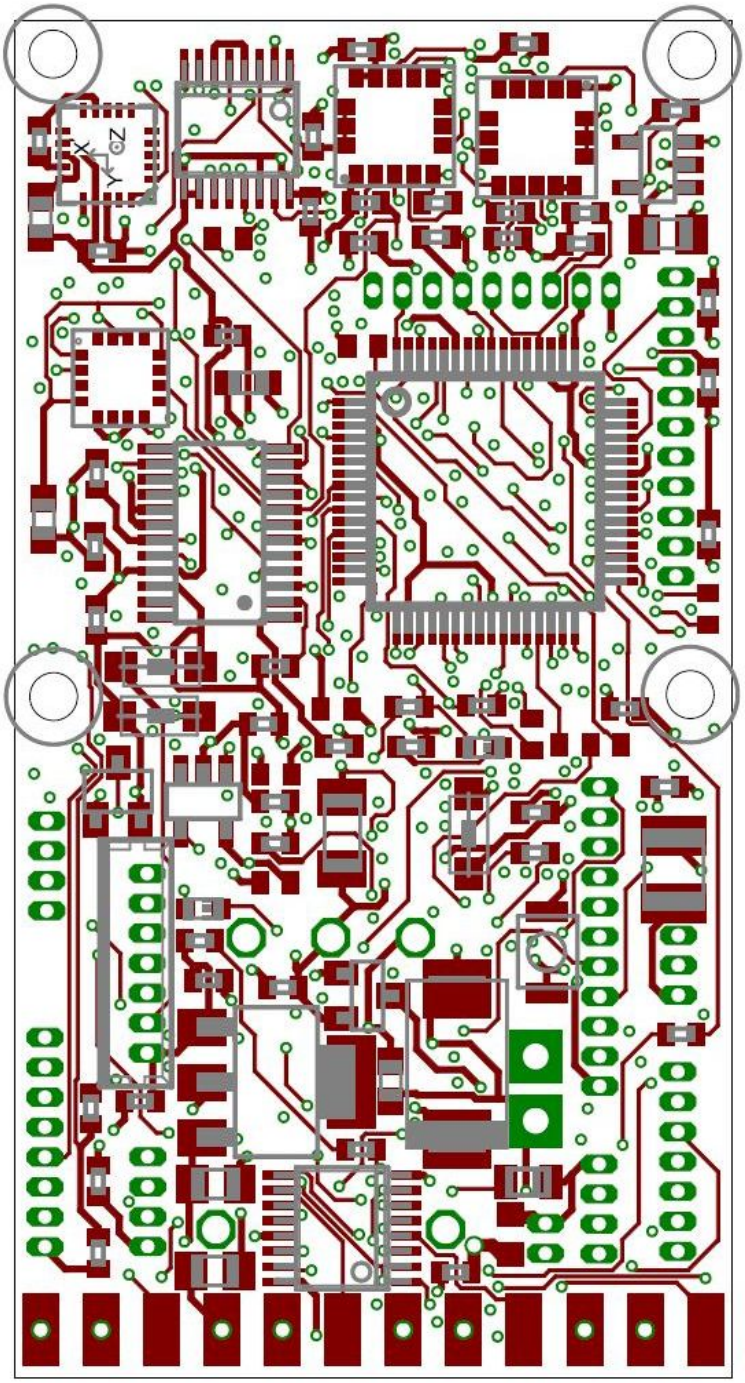


Abbildung 14.11: Layout Oberseite der LIS Variante

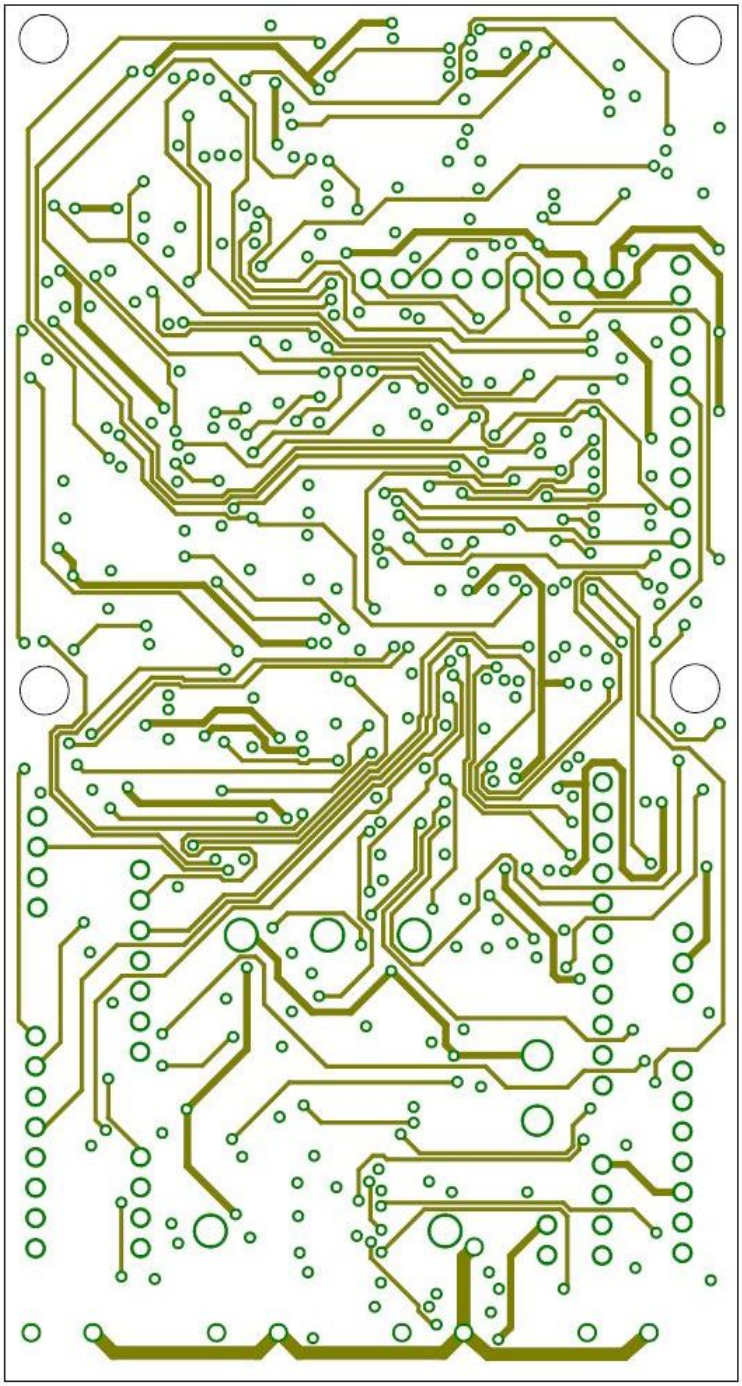


Abbildung 14.12: Layout obere Innenlage der LIS Variante

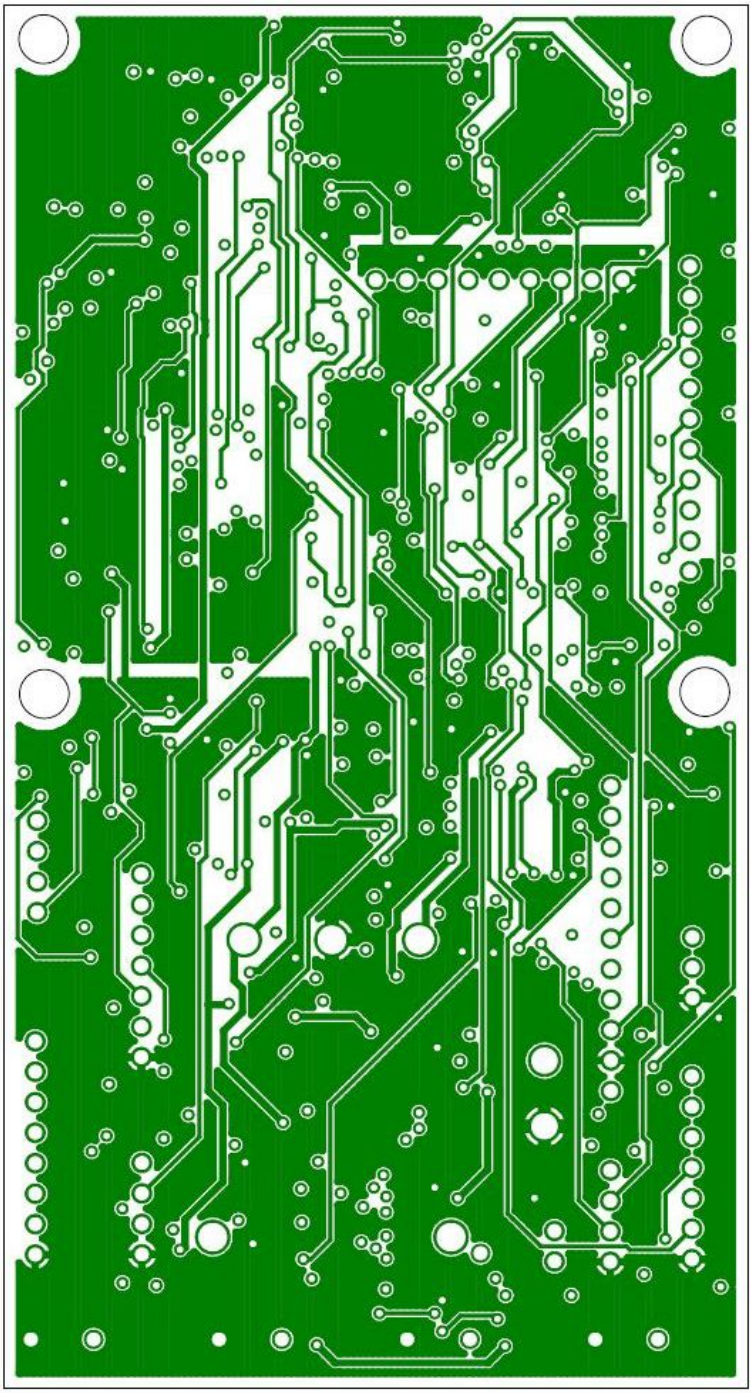


Abbildung 14.13: Layout untere Innenlage (GND Layer) der LIS Variante

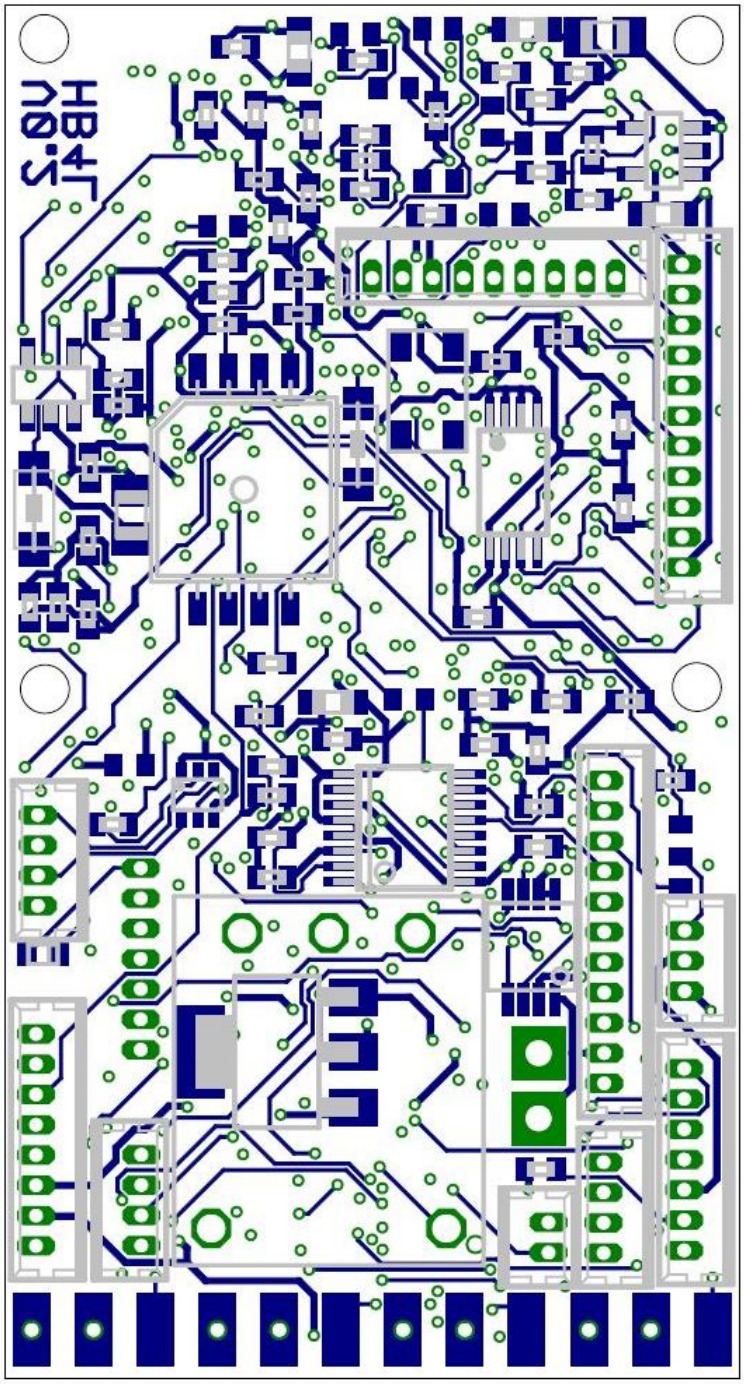


Abbildung 14.14: Layout Unterseite der LIS Variante

14.7.2 IDG Variante

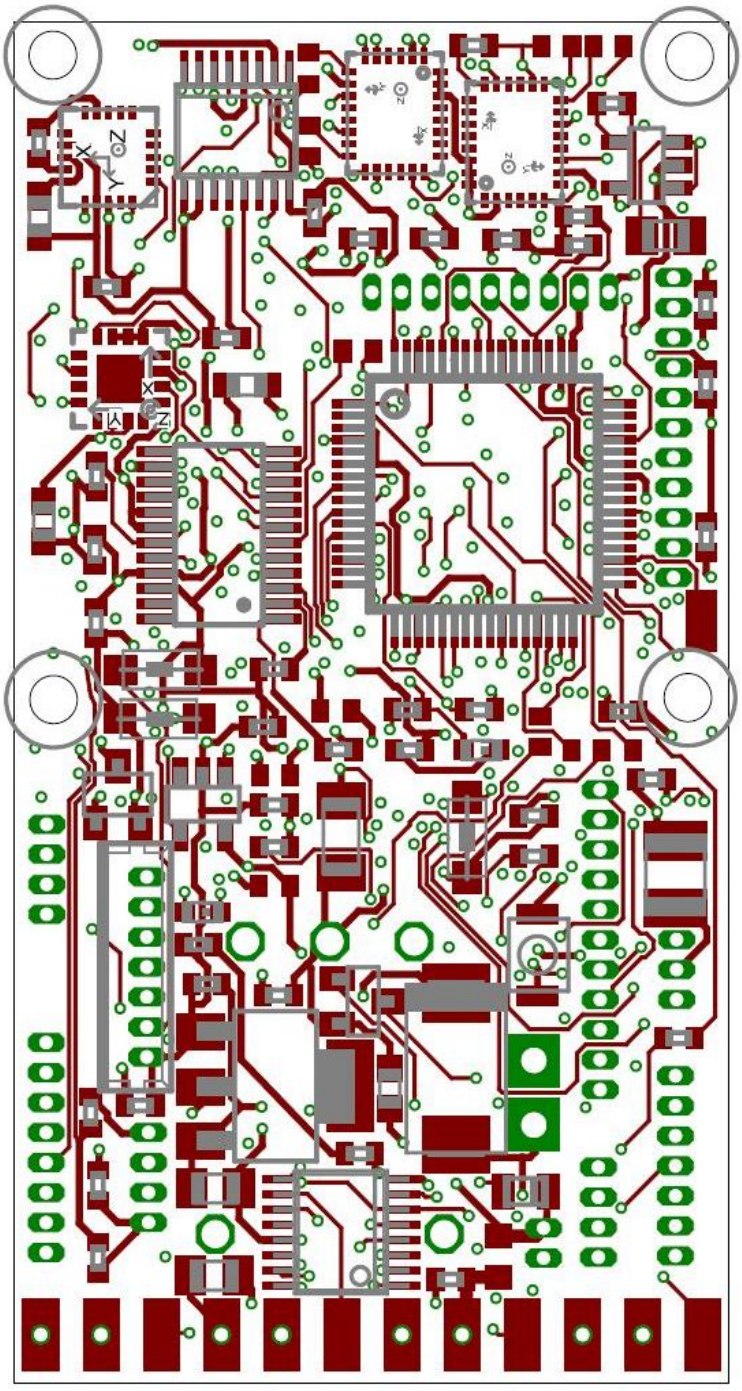


Abbildung 14.15: Layout Oberseite der IDG Variante

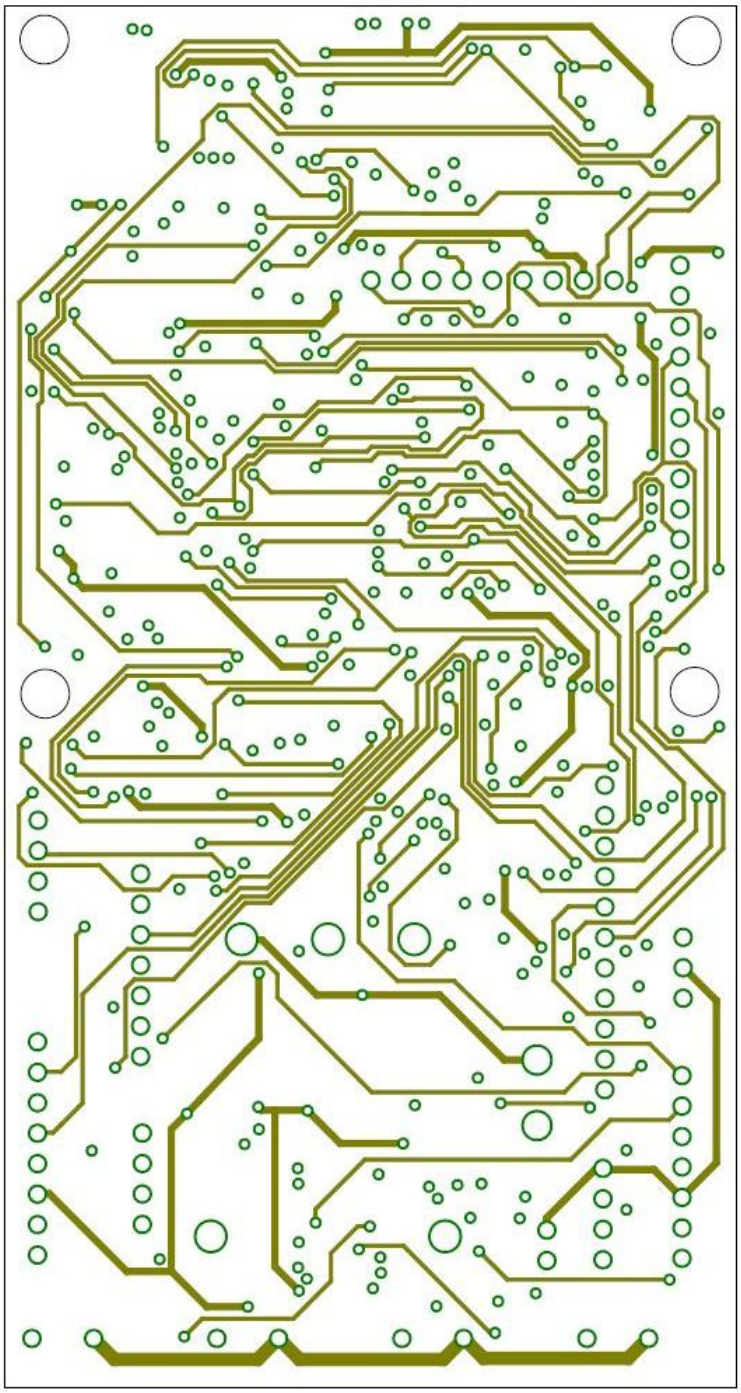


Abbildung 14.16: Layout obere Innenlage der IDG Variante

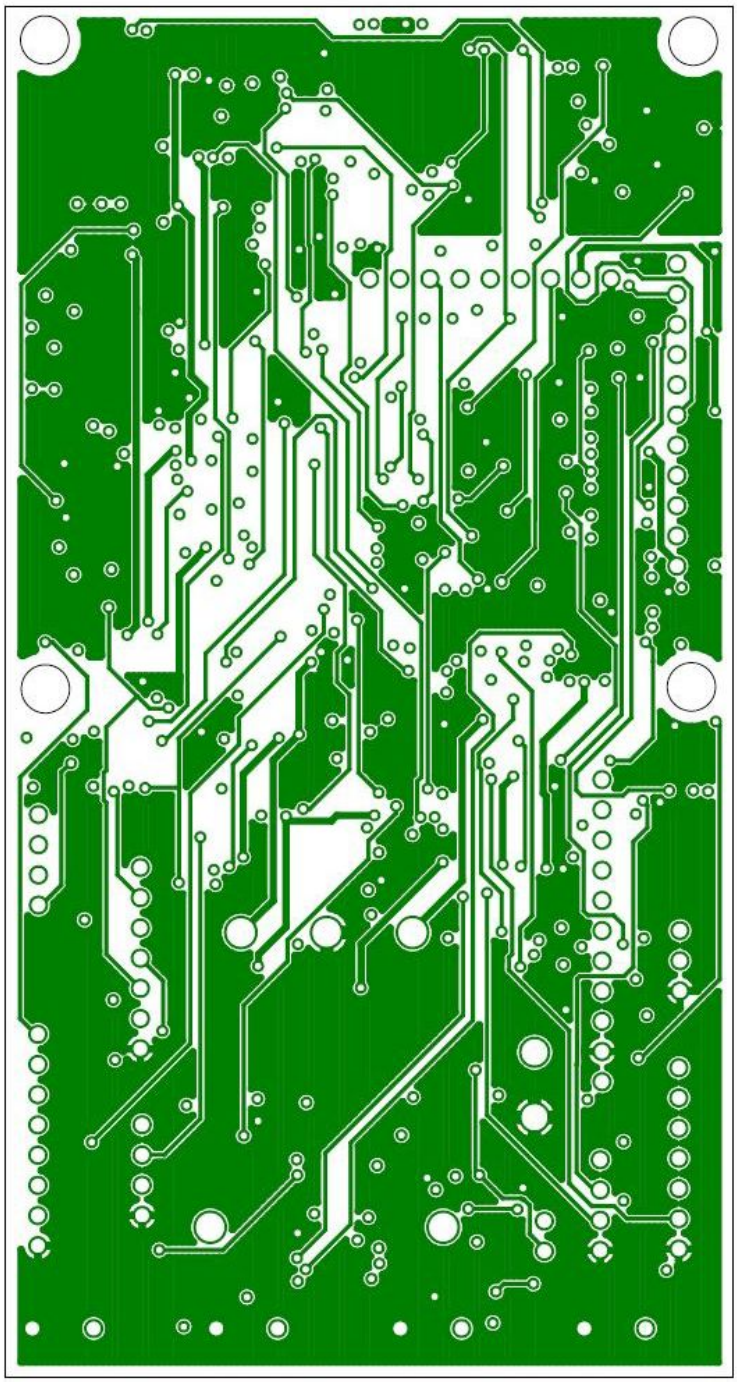


Abbildung 14.17: Layout untere Innenlage (GND Layer) der IDG Variante

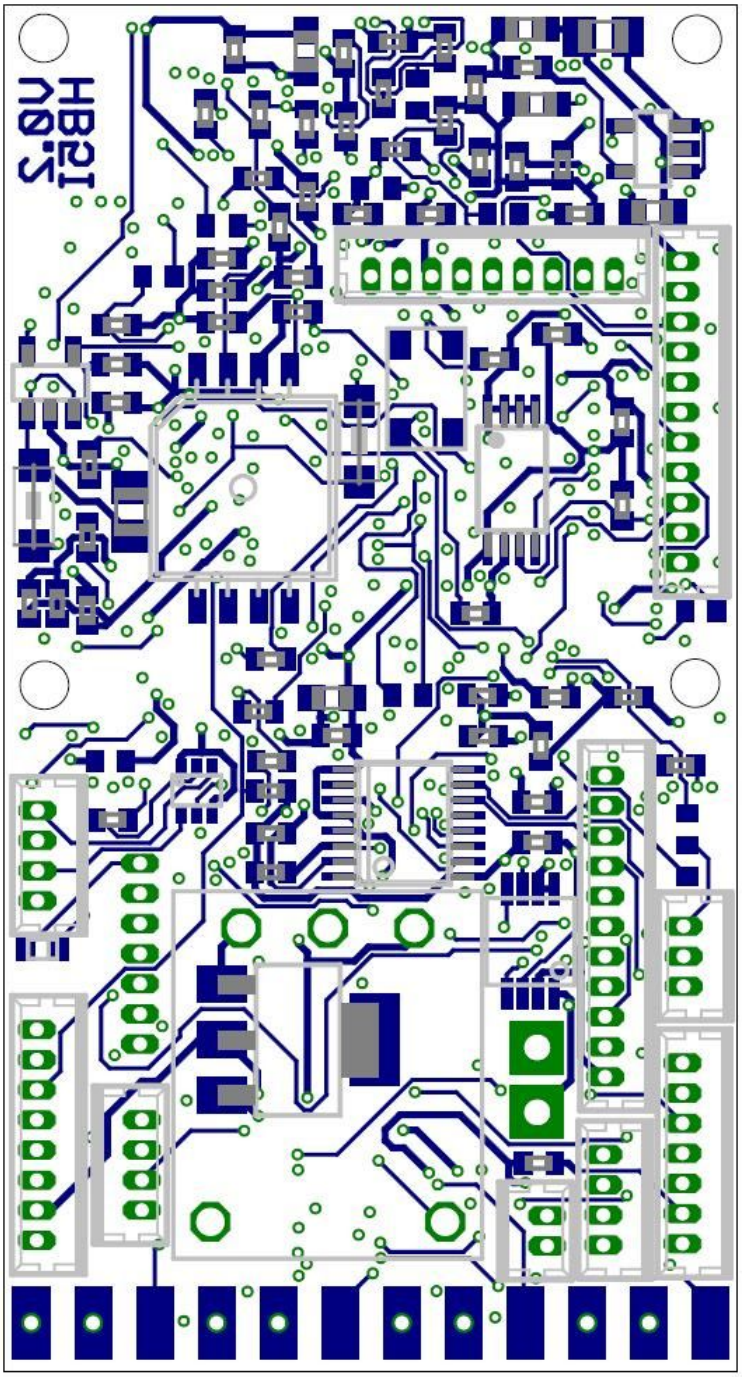


Abbildung 14.18: Layout Unterseite der IDG Variante

14.8 Bestellliste

Menge	Wert	Device	Bauform	Hersteller	Lieferant	Bestellnummer	VPE	Preis/VPE	LIS_LIS.bom	Preis/Stck.	Preis/AP_LIS	Preis/AP_IDG
2	24LC64ST	24LC64ST	TSSOP8	Microchip	Digikey	24LC64-I/ST-ND	1	0,48	IDG_100; LIS_LIS	0,480	0,480	0,480
2	4017T	4017T	TSSOP16	Texas Instruments	Digikey	296-9423-1-ND	1	0,64	IDG_100; LIS_LIS	0,460	0,460	0,460
2		51021-02	Housing	Molex	Digikey	WM1720-ND	1	0,3	IDG_100; LIS_LIS	0,300	0,300	0,300
2		51021-03	Housing	Molex	Digikey	WM1721-ND	1	0,3	IDG_100; LIS_LIS	0,300	0,300	0,300
6		51021-04	Housing	Molex	Digikey	WM1722-ND	1	0,3	IDG_100; LIS_LIS	0,300	0,900	0,900
6		51021-07	Housing	Molex	Digikey	WM1725-ND	1	0,3	IDG_100; LIS_LIS	0,300	0,900	0,900
2		51021-09	Housing	Molex	Digikey	WM1727-ND	1	0,3	IDG_100; LIS_LIS	0,300	0,300	0,300
2		51021-10	Housing	Molex	Digikey	WM1728-ND	1	0,3	IDG_100; LIS_LIS	0,300	0,300	0,300
2		51021-11	Housing	Molex	Digikey	WM1729-ND	1	0,3	IDG_100; LIS_LIS	0,300	0,300	0,300
2		53047-02	gerade	Molex	Digikey	WM1731-ND	1	0,25	IDG_100; LIS_LIS	0,250	0,250	0,250
2		53047-03	gerade	Molex	Digikey	WM1732-ND	1	0,26	IDG_100; LIS_LIS	0,260	0,260	0,260
6		53047-04	gerade	Molex	Digikey	WM1733-ND	1	0,31	IDG_100; LIS_LIS	0,310	0,930	0,930
6		53047-07	gerade	Molex	Digikey	WM1736-ND	1	0,51	IDG_100; LIS_LIS	0,510	1,530	1,530
2		53047-09	gerade	Molex	Digikey	WM1738-ND	1	0,61	IDG_100; LIS_LIS	0,610	0,610	0,610
2		53047-10	gerade	Molex	Digikey	WM1739-ND	1	0,65	IDG_100; LIS_LIS	0,650	0,650	0,650
2		53047-11	gerade	Molex	Digikey	WM1740-ND	1	1,03	IDG_100; LIS_LIS	1,030	1,030	1,030
71	100nF	C-EUC0402	402	AVX	Digikey	478-1129-1-ND	10	0,81	IDG_100; LIS_LIS	0,081	2,835	2,916
5	10n	C-EUC0402	402	AVX	Digikey	478-1114-1-ND	10	0,22	LIS_LIS	0,022	0,110	
3	10nF	C-EUC0402	402	AVX	Digikey	478-1114-1-ND	10	0,81	IDG_100	0,081		0,243
4	18pF	C-EUC0402	402	AVX	Digikey	478-1073-1-ND	10	0,81	IDG_100; LIS_LIS	0,081	0,162	0,162
4	22pF	C-EUC0402	402	AVX	Digikey	478-1074-1-ND	10	0,81	IDG_100; LIS_LIS	0,081	0,162	0,162
2	47pF	C-EUC0402	402	AVX	Digikey	478-1078-1-ND	10	0,88	IDG_100; LIS_LIS	0,088	0,088	0,088
3	100nF	C-EUC0603	C0603C104K5RACTU	CAP CERAMIC .100uF 50V X7R 0603 Kemet	Digikey	399-5089-1-ND	10	0,12	IDG_100; LIS_LIS	0,012	0,012	0,024
9	10uF	C-EUC0603	603	AVX	Digikey	478-5318-1-ND	10	6,18	IDG_100; LIS_LIS	0,618	3,090	2,472
2	1uF	C-EUC0603	603	AVX	Digikey	478-5010-1-ND	10	2,83	IDG_100; LIS_LIS	0,283	0,283	0,283
3	4.7uF	C-EUC0603	603	AVX	Digikey	478-2582-1-ND	10	4,86	IDG_100; LIS_LIS	0,486	0,486	0,972
2	22uF	C-EUC0805K	805	AVX	Digikey	478-3647-1-ND	10	5,3	IDG_100; LIS_LIS	0,530	0,530	0,530
2	22uF	C-EUC1206K	1206	AVX	Digikey	478-5332-1-ND	10	15,9	IDG_100; LIS_LIS	1,590	1,590	1,590
1	100uF	CPOL-EUSMCC	MLCC	AVX	Digikey	478-6116-ND	1	11,98	IDG_100	11,980		11,980
1	100uF	CPOL-EUSMCD	SMC_D	AVX	Digikey	478-3097-1-ND	1	3,16	LIS_LIS	3,160	3,160	
2	HMC5843	HMC5843	SMD	Honeywell	Digikey	342-1071-ND	1	15,84	IDG_100; LIS_LIS	15,840	15,840	15,840
2	ISMO_PTH08080W	ISMO_PTH08080W	5-DIP	Texas Instruments	Digikey	296-20432-ND	1	8,87	IDG_100; LIS_LIS	8,870	8,870	8,870
1	LIS344ALH	LIS344ALH	LGA16	ST	Digikey	497-6345-1ND	1	7,71	LIS_LIS	7,710	7,710	
2	LPC2148	LPC2148	64-LQFP	NXP	Digikey	568-1765-ND	1	9,73	IDG_100; LIS_LIS	9,730	9,730	9,730
4	LT1761-3.3	LT1761	SOT23-5	LT	Digikey	LT1761ES5-3.3#TRMPBFCT-ND	1	2,08	IDG_100; LIS_LIS	2,080	4,160	4,160
2	LT1761-5	LT1761	SOT23-5	LT	Digikey	LT1761ES5-5#TRMPBFCT-ND	1	2,08	IDG_100; LIS_LIS	2,080	2,080	2,080
2	LTC2440T	LTC2440T	SSOP16	Linear Technology	Digikey	LTC2440IGN#PBF-ND	1	11,05	IDG_100; LIS_LIS	11,050	11,050	11,050
1	LY530ALH	LY530ALH	16-LGA	ST	Digikey	497-8938-ND	1	7,28	LIS_LIS	7,280	7,280	
2	PCA9306DP	PCA9306DP	8-TSSOP	NXP	Digikey	568-4214-1-ND	1	0,68	IDG_100; LIS_LIS	0,680	0,680	0,680
16	100R	R-EU_R0402	402	YAGEO	Digikey	311-100JRCT-ND	10	0,59	IDG_100; LIS_LIS	0,059	0,472	0,472
34	10k	R-EU_R0402	402	YAGEO	Digikey	311-10kJRCT-ND	10	0,59	IDG_100; LIS_LIS	0,059	1,000	1,000
2	18R	R-EU_R0402	402	YAGEO	Digikey	311-18JRCT-ND	10	0,59	IDG_100; LIS_LIS	0,059	0,059	0,059
2	1k5	R-EU_R0402	402	YAGEO	Digikey	311-1.5kJRCT-ND	10	0,59	IDG_100; LIS_LIS	0,059	0,059	0,059
6	1k8	R-EU_R0402	402	YAGEO	Digikey	311-1.8kJRCT-ND	10	0,59	IDG_100; LIS_LIS	0,059	0,177	0,177
2	1R	R-EU_R0402	402	YAGEO	Digikey	311-1.0JCT-ND	10	0,64	IDG_100; LIS_LIS	0,064	0,064	0,064
4	220R	R-EU_R0402	402	YAGEO	Digikey	311-220JRCT-ND	10	0,59	IDG_100; LIS_LIS	0,059	0,118	0,118
2	24k	R-EU_R0402	402	YAGEO	Digikey	311-24kJRCT-ND	10	0,54	IDG_100; LIS_LIS	0,054	0,054	0,054
2	27k	R-EU_R0402	402	YAGEO	Digikey	311-24kJRCT-ND	10	0,54	IDG_100; LIS_LIS	0,054	0,054	0,054
2	330R	R-EU_R0402	402	YAGEO	Digikey	311-330JRCT-ND	10	0,59	IDG_100; LIS_LIS	0,059	0,059	0,059
4	33R	R-EU_R0402	402	YAGEO	Digikey	311-33JRCT-ND	10	0,59	IDG_100; LIS_LIS	0,059	0,118	0,118
18	3k3	R-EU_R0402	402	YAGEO	Digikey	311-3.3kJRCT-ND	10	0,59	IDG_100; LIS_LIS	0,059	0,531	0,531
2	470R	R-EU_R0402	402	YAGEO	Digikey	311-470JRCT-ND	10	0,59	IDG_100; LIS_LIS	0,059	0,059	0,059
2	4k7	R-EU_R0402	402	YAGEO	Digikey	311-4.7kJRCT-ND	10	0,59	IDG_100; LIS_LIS	0,059	0,059	0,059
2	56k	R-EU_R0402	402	YAGEO	Digikey	311-56kJRCT-ND	10	0,59	IDG_100; LIS_LIS	0,059	0,059	0,059
2	6k2	R-EU_R0402	402	YAGEO	Digikey	311-6.2kJRCT-ND	10	0,59	IDG_100; LIS_LIS	0,059	0,059	0,059
3	750R	R-EU_R0402	402	VISHAY	Digikey	541-750JCT-ND	10	0,54	IDG_100	0,054		0,162
2	100uF	RCL_C-EUC1210	1210	Kemet	Digikey	399-4697-1-ND	5	7,5	IDG_100; LIS_LIS	1,500	1,500	1,500
2	REG1117	REG1117	SOT223	Texas Instruments	Digikey	296-21321-1-ND	1	2,19	IDG_100; LIS_LIS	2,190	2,190	2,190
2	SERVO_SMD	SERVO_SMD	0.100"	Molex	Digikey	WM8130-ND	1	1,83	IDG_100; LIS_LIS	1,830	1,830	1,830
2	SI1912EDH	SI1912EDH	SC70-6	Vishay	Digikey	SI1912EDH-T1-E3CT-ND	1	1,29	IDG_100; LIS_LIS	1,290	1,290	1,290
2	TASTER_OMRON	TASTER_OMRON	SMD	Omron	Digikey	SW1020CT-ND	1	0,74	IDG_100; LIS_LIS	0,740	0,740	0,740
2	TPS2051B-SOT	TPS2051B-SOT	SOT23-5	Texas Instruments	Digikey	296-21265-1-ND	1	1,49	IDG_100; LIS_LIS	1,490	1,490	1,490
2	74138T	74138T	TSSOP-16	FAIRCHILD	Farnell	74LCX138MTCX	1	0,44	IDG_100; LIS_LIS	0,440	0,440	0,440
2		DIODE-MICROMELF-W	SOT23	Vishay	Farnell	9550089RL	10	0,73	IDG_100; LIS_LIS	0,073	0,073	0,073
2	MPXA6115A	MPXA6115A	8-SSOP	Freescale	Farnell	1457181	1	14,47	IDG_100; LIS_LIS	14,470	14,470	14,470
1	IDG500	IDG500	IDG500	InvenSense	InvenSense	IDG-500	1	9	IDG_100	9,000		9,000
1	IXZ500SMD	IXZ500SMD	IXZ500SMD	InvenSense	InvenSense	ISZ-500	1	6,5	IDG_100	6,500		6,500
1	ADXL335	ADXL335		Analog Devices	LiPoly	ADXL335	1	5,85	IDG_100	5,850		5,850
1	LPR530AL	LPR530AL	LPR530L	ST	LiPoly	LPR530L	1	5,85	LIS_LIS	5,850	5,850	
2	JUMPER0603	JUMPER0603			PCB-POOL				IDG_100; LIS_LIS			
4	MORITZ_ANSCHLUSS_SOLDER_PAD	MORITZ_ANSCHLUSS_SOLDER_PAD			PCB-POOL				IDG_100; LIS_LIS			
8	MOUNTHOLE	MOUNTHOLE			PCB-POOL				IDG_100; LIS_LIS			
32	SJ0402	SJ0402			PCB-POOL				IDG_100; LIS_LIS			
2	10uF	C-EUC0805K	805	Murata	Reichelt	X5R-G0805 10/16	1	0,05	IDG_100; LIS_LIS	0,050	0,050	0,050
6	2.2uF	C-EUC0805K	805	Murata	Reichelt	X7R-G0805 2,2/25	1	0,05	IDG_100; LIS_LIS	0,050	0,150	0,150
2	1PS79SB10	1PS79SB10	SMD	NXP	RS	1PS79SB10	50	4,85	IDG_100; LIS_LIS	0,097	0,097	0,097
2	BC817-40SMD	BC817-40SMD	SOT-23	ON	RS	464-243	200	3,41	IDG_100; LIS_LIS	0,017	0,017	0,017
6	220nF	C-EUC0402	402	AVX	RS	04026C223JAT2A	100	0,07	IDG_100; LIS_LIS	0,070	0,210	0,210
2	470n	C-EUC0402	402	Murata	RS	GRM155R60J474KE19D	1	0,1	LIS_LIS	0,100	0,100	
2		CTS_CHRYSTAL	SMT5x3.2	Euroquartz	RS	693-9200	1	3,28	IDG_100; LIS_LIS	3,280	3,280	3,280
11	0□H47	ISMO_L.-FERRIT_1206	1206	W□rth	RS	327-5566	10	3,15	IDG_100; LIS_LIS	0,032	0,160	0,160
2	gn	LEDCHIP-LED0603	603	Osram	RS	654-5767	50	3,5	IDG_100; LIS_LIS	0,070	0,070	0,070
2	rd	LEDCHIP-LED0603	603	Osram	RS	654-5414	50	6	IDG_100; LIS_LIS	0,120	0,120	0,120
2	TRANSISTOR-PNP_MUN2111T1	TRANSISTOR-PNP_MUN2111T1	SC-59	ON	RS	690-0158	25	1,15	IDG_100; LIS_LIS	0,046	0,046	0,046
2	ISMO_MAX1168	ISMO_MAX1168		Maxim	Sample	MAX1168			IDG_100; LIS_LIS		18,220	18,220
							Summe:	220,26		147,22	144,80	154,29

14.9 Quellcode

```
/*********************************************************************
 *
 * Hochschule Bremen
 * Forschungsprojekt: Autopilot fuer Schwebefluggeraete
 *                        -- HB-Quadrokopter --
 *                    Prof. Dr.-Ing. Heinrich Warmers
 *
 *  Copyright (C) 2009  Christoph Niemann
 *
 *  This program is free software: you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation, either version 3 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program.  If not, see <http://www.gnu.org/licenses/>.
 *
 *********************************************************************/

/** \addtogroup ALTITUDE GPL-QC Altitude
 * @{
 */

/**
 * \file
 *
 * Altitude implementation
 *
 * This module implement the store of the analog value
 * generated form the air pressure at startpoint.
 * The DAC is use to generate a offset to fit multiple
 * ranges of air pressures.
 *
 * The following features are supported:
 * - Initialisize the alt_p0 value with ground pressure
 * - Implementation variable alt_mode
 *
 * \author Christoph Niemann
 */

#include "lpc214x.h"

#include "types.h"
#include "board.h"
#include "analog.h"
#include "dbg.h"
#include "led.h"
#include "timer.h"
#include "convert.h"

#include "altitude.h"

/**
 * Global value to store the mode of altitude operation
 */
alt_status_t alt_mode = ALT_MANUAL;

#ifdef TINY13
/**
 * Initialise alt_p0 with adc_average[ADC_ALT] value.\n
 * Use the DAC to generate variable offset of different
 * pressure environments.
 */
void alt_init(void) {
    // begin init the altimeter
```

```
    int dac_alti = 800;
    adc_offset[ADC_ALT] = 0;
    while (1) {
        dac_alti--;
        //if (dac_alti == 700) {
        //    dac_alti = 800;
        //}
//#undef DAC_ALTI
#define DAC_ALTI
#ifdef DAC_ALTI
        DACR = (dac_alti << 6);
#endif
        buzzer = ON;//nervt beim einschalten. LED blinken reicht.
        for (int tmp = 0; tmp < 100000; tmp++) {
            asm("nop");
        }
        buzzer = OFF;

        // debug output
        dbg_adc(3);
        dbg_adc_offset(4);

        //
        LED_GREEN_FLASH;
        if (adc_average[ADC_ALT] > 650) {
            break;
        }
        /**
         * @todo beep bei fehler, abbruch ?
         */
    }

    // store altitude at start position
    alt_p0 = adc_average[ADC_ALT];
}
#endif // TINY13

#ifdef HBC
/**
 * to be done, need to be implemented for HBC !
 *
 * \todo blocker: function not implemented for HBC
 */
void alt_init(void) {
/**
 * \todo blocker: function not implemented for HBC
 */

}
#endif // HBC

#ifdef HBMINI
/**
 * to be done, need to be implemented for HBMINI !
 *
 * \todo blocker: function not implemented for HBMINI
 */
void alt_init(void) {
/**
 * \todo blocker: function not implemented for HBMINI
 */

}
#endif // HBMINI
/**
 * @}
 */
```

```
/************************************************************************
 *
 * Hochschule Bremen
 * Forschungsprojekt: Autopilot fuer Schwebefluggeraete
 *                         -- HB-Quadrokopter --
 *                     Prof. Dr.-Ing. Heinrich Warmers
 *
 *  Copyright (C) 2009  Oliver Riesener
 *
 *  This program is free software: you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation, either version 3 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program.  If not, see <http://www.gnu.org/licenses/>.
 *
 ************************************************************************/

/** \addtogroup ADC LPC21xx ADC support
 * @{
 */

/** \file
 *
 * ADC implementation
 *
 * \author Oliver Riesener
 *
 * This module implement access to internal ADC's
 *
 * The following features are supported:
 * - Initialisize the analog ports
 * - registers irq routines for adc0 and adc1
 * - implements a average function
 * - get offsets and substrate then every permanently
 */

#include <lpc214x.h>

#include "types.h"
#include "board.h"
#include "vic.h"
#include "analog.h"
#include "convert.h"
#include "dac.h"

#ifdef TINY13
// globals
/**
 * storage and export of average values from adc0, adc1
 *
 *                            adc0 : adc1     // 2 * 8 values\n
 *               0,1,2,3,4,5,6,7 : 0,1,2,3,4,5,6,7\n
 * 0,IR1,IR2,IR3,DAC,0,KADC,KADC : KADC,KADC,ADC2,ADC3,ADC4,ADC5,VBAT,KADC\n
 *
 * \todo switch to adc1 and adc0 presentation
 */
volatile int adc_average[16] = { 0 };
/**
 * storage of offset values from adc0, adc1
 *
 * this values are permanetly substracted from actual measure value
 * they could by set by adc_offset_set()
 */
```

```
adc_offset_t adc_offset[16] = { 0 };

// locals
/**
 * internal storage variable for building average values
 */
static int adc_value_sum[16] = { 0 };
/**
 * internal storage variable for building average values
 */
static int adc_count[16] = { 0 };

// function prototypes
void __attribute__ ((interrupt("IRQ"))) adc0_int(void);
void __attribute__ ((interrupt("IRQ"))) adc1_int(void);

// functions
/**
 * initialise adc hardware, register irq routines adc0, adc1
 */
void adc_init(void) {
    // switch pins for ADC0: ad0.1,ad0.2,ad0.3,ad0.4
    PINSEL1 |= (1<<24)|(1<<26)|(1<<28)|(1<<18);
    PINSEL1 &= ~((1<<25)|(1<<27)|(1<<29)|(1<<19));

    // switch pins for ADC1: ad1.3,ad1.4,ad1.5
    PINSEL0 |= (3<<24)|(3<<26)|(3<<30);

    // switch pins for ADC1 an Pin: ad1.6,ad1.7;
    PINSEL1 |= (1<<11)|(1<<12);
    PINSEL1 &= ~((1<<10)|(1<<13));

    // ADC0 channel interrupts
    AD0CR |= 0x1E; // channel 7 6 5 (4) (3 2 1) 0
    AD0CR |= (0xFF<<8); // div :256
    AD0CR |= (1<<21); // power up

    // ADC1 channel interrupts
    AD1CR |= 0xF8; // channel (7 6 5 4) (3) 2 1 0
    AD1CR |= (0xFF<<8); // div :256
    AD1CR |= (1<<21); // power up

    // set global burst bit
    ADGSR |= (1<<16);

    // set interrupt vector adc0
    VIC_ADDR_ADC0 = (unsigned long) adc0_int;
    // use it for ADC0 Interrupt
    VIC_CTRL_ADC0 = VICVectCntl_ENABLE | VIC_Slot_ADC0;
    // enable ADC0 Interrupt
    VICIntEnable = (1<<VIC_Slot_ADC0);

    // set interrupt vector adc1
    VIC_ADDR_ADC1 = (unsigned long) adc1_int;
    // use it for ADC1 interrupt
    VIC_CTRL_ADC1 = VICVectCntl_ENABLE | VIC_Slot_ADC1;
    // enable ADC1 interrupt
    VICIntEnable = (1<<VIC_Slot_ADC1);
}

/**
 * interrupt routine for adc0_int
 */
void adc0_int(void) {
    unsigned int adc_value = AD0GDR;
    unsigned char channel = ((adc_value >> 24) & 7);

    adc_value_sum[channel] += ((adc_value >> 6) & 0x3FF) - adc_offset[channel];
    adc_count[channel]++;
    // calculate average
```

```
    if (adc_count[channel] == AVERAGE_COUNT_NB) {
        adc_average[channel] = adc_value_sum[channel] / adc_count[channel];
        adc_count[channel] = 0;
        adc_value_sum[channel] = 0;
    }
    // clear interrupt
    VICVectAddr = 0x00000000;
}

/**
 * interrupt routine for adc1_int
 */
void adc1_int(void) {
    unsigned int adc_value = AD1GDR;
    unsigned char channel = ((adc_value>>24)&7) + 8; // offset+8=chan for adc1

    adc_value_sum[channel] += ((adc_value >> 6) & 0x3FF) - adc_offset[channel];
    adc_count[channel]++;
    // calculate average
    if (adc_count[channel] == AVERAGE_COUNT_NB) {
        adc_average[channel] = adc_value_sum[channel] / adc_count[channel];
        adc_count[channel] = 0;
        adc_value_sum[channel] = 0;
    }
    // clear interrupt
    VICVectAddr = 0x00000000;
}

/**
 * load adc_offset[] with set adc_average[] values
 */
void adc_offset_set(void) {
    unsigned int tmp, i;
    // set neutral offset values to zero
    for (i = 0; i < (sizeof(adc_offset) / sizeof(adc_offset_t)); i++) {
        adc_offset[i] = (adc_offset_t) 0;
    }
    // wait for new adc_average values with of neutral offset
    for (tmp = 0; tmp < 100000; tmp++) {
        asm("nop");
    }
    for (tmp = 0; tmp < 100000; tmp++) {
        asm("nop");
    }
    // set GYRO neutral values
    adc_offset[ADC_ROLL] = adc_average[ADC_ROLL];
    adc_offset[ADC_PITCH] = adc_average[ADC_PITCH];
    adc_offset[ADC_YAW] = adc_average[ADC_YAW];
    // set ACC neutral values
    adc_offset[ADC_ACCX] = adc_average[ADC_ACCX];
    adc_offset[ADC_ACCY] = adc_average[ADC_ACCY];
    /** \todo earth-accel is sensor/quadcopter depend, store in eeprom */
    adc_offset[ADC_ACCZ] = adc_average[ADC_ACCZ] - 256; // -256  earth-accel
    // set ALT and OFF neutral values
    adc_offset[ADC_ALT] = adc_average[ADC_ALT];
    adc_offset[ADC_OFF] = (adc_offset_t) 0;
}
#endif //TINY13

/**
 * \todo low: which analog values HBC really need ?, and which are the names ?
 * These is old stuff from TINY13, where are the differences
 */

#ifdef HBC

// globals
/**
 * storage and export of average values from adc0, adc1
 *
```

```
 *                           adc0 : adc1     // 2 * 8 values\n
 *               0,1,2,3,4,5,6,7 : 0,1,2,3,4,5,6,7\n
 * 0,IR1,IR2,IR3,DAC,0,KADC,KADC : KADC,KADC,ADC2,ADC3,ADC4,ADC5,VBAT,KADC\n
 *
 * \todo switch to adc1 and adc0 presentation
 */
volatile int adc_average[16] = { 0 };
/**
 * storage of offset values from adc0, adc1
 *
 * this values are permanetly substracted from actual measure value
 * they could by set by adc_offset_set()
 */
adc_offset_t adc_offset[16] = { 0 };

// locals
/**
 * internal storage variable for building average values
 */
static int adc_value_sum[16] = { 0 };
/**
 * internal storage variable for building average values
 */
static int adc_count[16] = { 0 };

// function prototypes
void __attribute__ ((interrupt("IRQ"))) adc0_int(void);
void __attribute__ ((interrupt("IRQ"))) adc1_int(void);

// functions
/**
 * initialise adc hardware and enable irq routines for adc0 and adc1
 */
void adc_init(void) {
    // switch pins for ADC0: ad0.1,ad0.2,ad0.3,ad0.4
    PINSEL1 |= (1<<24)|(1<<26)|(1<<28)|(1<<18);
    PINSEL1 &= ~((1<<25)|(1<<27)|(1<<29)|(1<<19));

    // switch pins for ADC1: ad1.2, ad1.3,ad1.4,ad1.5
    PINSEL0 |= (3<<20)|(3<<24)|(3<<26)|(3<<30);

    // switch pins for ADC1 an Pin: ad1.6;
    PINSEL1 |= (1<<11);
    PINSEL1 &= ~(1<<10);

    // ADC0 channel interrupts
    AD0CR |= 0x1E; // channel 7 6 5 (4) (3 2 1) 0
    AD0CR |= (0xFF<<8); // div :256
    AD0CR |= (1<<21); // power up

    // ADC1 channel interrupts
    AD1CR |= 0x7C; // channel 7 (6 5 4) (3 2) 1 0
    AD1CR |= (0xFF<<8); // div :256
    AD1CR |= (1<<21); // power up

    // set global burst bit
    ADGSR |= (1<<16);

    // set interrupt vector adc0
    VIC_ADDR_ADC0 = (unsigned long) adc0_int;
    // use it for ADC0 interrupt
    VIC_CTRL_ADC0 = VICVectCntl_ENABLE | VIC_Slot_ADC0;
    // enable ADC0 interrupt
    VICIntEnable = (1<<VIC_Slot_ADC0);

    // set interrupt vector adc1
    VIC_ADDR_ADC1 = (unsigned long) adc1_int;
    // use it for ADC1 interrupt
    VIC_CTRL_ADC1 = VICVectCntl_ENABLE | VIC_Slot_ADC1;
    // enable ADC1 interrupt
```

```
    VICIntEnable = (1<<VIC_Slot_ADC1);
}

/**
 * interrupt routine for adc0_int
 */
void adc0_int( void ) {
    unsigned int adc_value = AD0GDR;
    unsigned char channel = ((adc_value>>24)&7);

    adc_value_sum[channel] += ((adc_value>>6)&0x3FF) - adc_offset[channel];
    adc_count[channel]++;
    // calculate average
    if( adc_count[channel] == AVERAGE_COUNT_NB) {
        adc_average[channel] = adc_value_sum[channel] / adc_count[channel];
        adc_count[channel] = 0;
        adc_value_sum[channel] = 0;
    }
    // clear interrupt
    VICVectAddr = 0x00000000;
}

/**
 * interrupt routine for adc1_int
 */
void adc1_int( void ) {
    unsigned int adc_value = AD1GDR;
    unsigned char channel = ((adc_value>>24)&7) + 8; // offset+8=chan for adc1

    adc_value_sum[channel] += ((adc_value>>6)&0x3FF) - adc_offset[channel];
    adc_count[channel]++;
    // calculate average
    if( adc_count[channel] == AVERAGE_COUNT_NB) {
        adc_average[channel] = adc_value_sum[channel] / adc_count[channel];
        adc_count[channel] = 0;
        adc_value_sum[channel] = 0;
    }
    // clear interrupt
    VICVectAddr = 0x00000000;
}


/**
 * set adc_average[] values to adc_offset[] values
 */
void adc_offset_set(void) {
    unsigned int tmp, i;
    // set neutral offset values to zero
    for ( i=0; i<(sizeof(adc_offset)/sizeof(adc_offset_t)); i++) {
        adc_offset[i] = (adc_offset_t) 0;
    }
    // wait for new adc_average values with of neutral offset
    for(tmp = 0;tmp<100000;tmp++){
        asm("nop");
    }
    for(tmp = 0;tmp<100000;tmp++){
        asm("nop");
    }
    // set GYRO neutral values
    adc_offset[ADC_ROLL] = adc_average[ADC_ROLL];
    adc_offset[ADC_PITCH] = adc_average[ADC_PITCH];
    adc_offset[ADC_YAW] = adc_average[ADC_YAW];
    // set ACC neutral values
    adc_offset[ADC_ACCX] = adc_average[ADC_ACCX];
    adc_offset[ADC_ACCY] = adc_average[ADC_ACCY];
    /** \todo earth-accel is sensor/quadcopter depend, store in eeprom */
    adc_offset[ADC_ACCZ] = adc_average[ADC_ACCZ]-256; // -256  earth-accel
    // set ALT and OFF neutral values
    adc_offset[ADC_ALT] = adc_average[ADC_ALT];
    adc_offset[ADC_OFF] = (adc_offset_t) 0;
```

```
}
#endif //HBC

/** @} */
/**
 * \todo low: which analog values HBMINI really need ?, and which are the names
?
 * These is old stuff from TINY13, where are the differences
 */

#ifdef HBMINI

// globals
/**
 * storage and export of average values from adc0, adc1
 *
 *                            adc0 : adc1     // 2 * 8 values\n
 *                0,1,2,3,4,5,6,7 : 0,1,2,3,4,5,6,7\n
 * 0,IR1,IR2,IR3,DAC,0,KADC,KADC : KADC,KADC,ADC2,ADC3,ADC4,ADC5,VBAT,KADC\n
 *
 * \todo switch to adc1 and adc0 presentation
 */
volatile int adc_average[16] = { 0 };
/**
 * storage of offset values from adc0, adc1
 *
 * this values are permanetly substracted from actual measure value
 * they could by set by adc_offset_set()
 */
adc_offset_t adc_offset[16] = { 0 };

// locals
/**
 * internal storage variable for building average values
 */
static int adc_value_sum[16] = { 0 };
/**
 * internal storage variable for building average values
 */
static int adc_count[16] = { 0 };

// function prototypes
void __attribute__ ((interrupt("IRQ"))) adc0_int(void);
void __attribute__ ((interrupt("IRQ"))) adc1_int(void);

// functions
/**
 * initialise adc hardware and enable irq routines for adc0 and adc1
 */
void adc_init(void) {
    // switch pins for ADC0: ad0.1,ad0.2,ad0.3,ad0.4
    PINSEL1 |= (1<<24)|(1<<26)|(1<<28)|(1<<18);
    PINSEL1 &= ~((1<<25)|(1<<27)|(1<<29)|(1<<19));

    // switch pins for ADC1: ad1.2, ad1.3,ad1.4,ad1.5
    PINSEL0 |= (3<<20)|(3<<24)|(3<<26)|(3<<30);

    // switch pins for ADC1 an Pin: ad1.6;
    PINSEL1 |= (1<<11);
    PINSEL1 &= ~(1<<10);

    // ADC0 channel interrupts
    AD0CR |= 0x1E; // channel 7 6 5 (4) (3 2 1) 0
    AD0CR |= (0xFF<<8); // div :256
    AD0CR |= (1<<21); // power up

    // ADC1 channel interrupts
    AD1CR |= 0x7C; // channel 7 (6 5 4) (3 2) 1 0
    AD1CR |= (0xFF<<8); // div :256
    AD1CR |= (1<<21); // power up
```

```
    // set global burst bit
    ADGSR |= (1<<16);

    // set interrupt vector adc0
    VIC_ADDR_ADC0 = (unsigned long) adc0_int;
    // use it for ADC0 interrupt
    VIC_CTRL_ADC0 = VICVectCntl_ENABLE | VIC_Slot_ADC0;
    // enable ADC0 interrupt
    VICIntEnable = (1<<VIC_Slot_ADC0);

    // set interrupt vector adc1
    VIC_ADDR_ADC1 = (unsigned long) adc1_int;
    // use it for ADC1 interrupt
    VIC_CTRL_ADC1 = VICVectCntl_ENABLE | VIC_Slot_ADC1;
    // enable ADC1 interrupt
    VICIntEnable = (1<<VIC_Slot_ADC1);
}

/**
 * interrupt routine for adc0_int
 */
void adc0_int( void ) {
    unsigned int adc_value = AD0GDR;
    unsigned char channel = ((adc_value>>24)&7);

    adc_value_sum[channel] += ((adc_value>>6)&0x3FF) - adc_offset[channel];
    adc_count[channel]++;
    // calculate average
    if( adc_count[channel] == AVERAGE_COUNT_NB) {
        adc_average[channel] = adc_value_sum[channel] / adc_count[channel];
        adc_count[channel] = 0;
        adc_value_sum[channel] = 0;
    }
    // clear interrupt
    VICVectAddr = 0x00000000;
}

/**
 * interrupt routine for adc1_int
 */
void adc1_int( void ) {
    unsigned int adc_value = AD1GDR;
    unsigned char channel = ((adc_value>>24)&7) + 8; // offset+8=chan for adc1

    adc_value_sum[channel] += ((adc_value>>6)&0x3FF) - adc_offset[channel];
    adc_count[channel]++;
    // calculate average
    if( adc_count[channel] == AVERAGE_COUNT_NB) {
        adc_average[channel] = adc_value_sum[channel] / adc_count[channel];
        adc_count[channel] = 0;
        adc_value_sum[channel] = 0;
    }
    // clear interrupt
    VICVectAddr = 0x00000000;
}


/**
 * set adc_average[] values to adc_offset[] values
 */
void adc_offset_set(void) {
    unsigned int tmp, i;
    // set neutral offset values to zero
    for ( i=0; i<(sizeof(adc_offset)/sizeof(adc_offset_t)); i++) {
        adc_offset[i] = (adc_offset_t) 0;
    }
    // wait for new adc_average values with of neutral offset
    for(tmp = 0;tmp<100000;tmp++){
        asm("nop");
```

```
    }
    for(tmp = 0;tmp<100000;tmp++){
        asm("nop");
    }
    // set GYRO neutral values
    adc_offset[ADC_ROLL] = adc_average[ADC_ROLL];
    adc_offset[ADC_PITCH] = adc_average[ADC_PITCH];
    adc_offset[ADC_YAW] = adc_average[ADC_YAW];
    // set ACC neutral values
    adc_offset[ADC_ACCX] = adc_average[ADC_ACCX];
    adc_offset[ADC_ACCY] = adc_average[ADC_ACCY];
    /** \todo earth-accel is sensor/quadcopter depend, store in eeprom */
    adc_offset[ADC_ACCZ] = adc_average[ADC_ACCZ]-128+23; // -256  earth-accel
    // set ALT and OFF neutral values
    adc_offset[ADC_ALT] = adc_average[ADC_ALT];
    adc_offset[ADC_OFF] = (adc_offset_t) 0;
}
#endif //HBMINI
```

```
/*******************************************************************
 *
 * Hochschule Bremen
 * Forschungsprojekt: Autopilot fuer Schwebefluggeraete
 *                         -- HB-Quadrokopter --
 *                     Prof. Dr.-Ing. Heinrich Warmers
 *
 *  Copyright (C) 2009  Oliver Riesener
 *
 *  This program is free software: you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation, either version 3 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program.  If not, see <http://www.gnu.org/licenses/>.
 *
 *******************************************************************/

/** \addtogroup CONFIG GPL-QC Configuration
 * @{
 */

/**
 * \file
 *
 * Implements global defines
 *
 * \author Oliver Riesener
 */

#ifndef _CONFIG_H_
#define _CONFIG_H_

#include "board.h" // user specific board selection

/** Define for Photocopter in X Formation, unless defined in + Formation */
#define PHOTOCOPTER
#undef PHOTOCOPTER

/**
 * enables kalman_hb(), else accel2euler()
 * \todo accel2euler() needs to by tested
 */
#define KALMAN_HB
#undef KALMAN_HB
/**
 * enables kompl_quat()
 */
#define KOMPL_QUAT
//#undef KOMPL_QUAT

/** enables automatical hovering
 * \warning enable only if you know what you are doing !
 */
#define HOVER_CTRL
#undef HOVER_CTRL

/** enables automatical waypoint navigation
 * \warning enable only if you know what you are doing !
 */
#define WAYPOINT
#undef WAYPOINT

/** enables servo outputs
```

```
 * \warning enable only if you know what you are doing !
 */
#define SERVOS
#undef SERVOS

/** enables use max1168 values in hb_mini
 *
 */
#define USE_MAX1168
// #undef USE_MAX1168

/** limits the minimal drive speed if drive_mode is on */
#define DRIVE_LIMIT_MIN 10
/** limits the maximal drive speed if drive_mode is on */
#define DRIVE_LIMIT_MAX 255 // valid from 10 to 255


/** battery factor 11.1V 3 cell LiPo */
#ifdef TINY13
/** \note TINY13 version */
#define BAT_MUL .1778 // *10 for Dezivolt
#endif //TINY13
#ifdef HBC
/** \note HBC version */
#ifndef PHOTOCOPTER
#define BAT_MUL .3236 // *10 for Dezivolt
#endif
/** \note PHOTOCOPTER version */
#ifdef PHOTOCOPTER
#define BAT_MUL .2158 // *10 for Dezivolt
/** \todo BAT_MUL Wert stimmt noch nicht */
#endif
#endif //HBC

#ifdef HBMINI
/** \note HBMINI version */
#ifndef PHOTOCOPTER
#define BAT_MUL .3236 // *10 for Dezivolt
#endif
/** \note PHOTOCOPTER version */
#ifdef PHOTOCOPTER
#define BAT_MUL .2158 // *10 for Dezivolt
/** \todo BAT_MUL Wert stimmt noch nicht */
#endif
#endif //HBMINI

/** enable IRQ handling for uarts */
#define USE_UART_IRQ
/** use UART0 */
#define USE_UART0
/** use UART1 */
#define USE_UART1
/** select 38400 baud at UART0 */
#define UART0_BAUD B38400
/** select 38400 baud at UART1 */
#define UART1_BAUD B19200
/** define PPRZ for IRQ Handling a'la PPRZ */
#define PPRZ
#undef PPRZ

#endif //_CONFIG_H_

/**
 * @}
 */
```

```
/*************************************************************************
 *
 * Hochschule Bremen
 * Forschungsprojekt: Autopilot fuer Schwebefluggeraete
 *                         -- HB-Quadrokopter --
 *                     Prof. Dr.-Ing. Heinrich Warmers
 *
 *  Copyright (C) 2009  Christoph Niemann, Andreas Dei
 *
 *  This program is free software: you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation, either version 3 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program.  If not, see <http://www.gnu.org/licenses/>.
 *
 *************************************************************************/

/** \addtogroup CONVERT GPL-QC Convert raw data to physical data
 * @{
 */

/** \file
 *
 * Implements routines to convert raw data to physial data
 *
 * All convert functions are implemented here.
 * The should be speaking and explain then convertion
 *
 * \author Christoph Niemann, Andreas Dei
 *
 * \todo cleanup, remove Y for Kalman
 */

#include <math.h>

#include "lpc214x.h"
#include "types.h"
#include "analog.h"
#include "vic.h"
#include "printf_P.h"
#include "rc.h"
#include "drive.h"
#include "mmag3.h"
#include "altitude.h"
#include "mm3.h"
#include "dac.h"
#include "dbg.h"
#include "timer.h"
#include "nav.h"
#include "mygps.h"
#include "board.h"
#include "imu.h"
#include "mymag.h"
#include "config.h"

#include "convert.h"

// variables

/** adc_converted \todo stores only ADC_ALT */
volatile float adc_converted[] =
{0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.};
```

```
/** acceleration in ms2 */
volatile float accel[ACC_LAST] = {0.};

/** acceleration normaliased */
vector3f_t accel_norm_v = { 0., 0., 0.};

/** gyro rate in rad */
volatile float gyro[G_LAST] = {0.};

/** angle in rad */
volatile float angle[ANG_LAST] = {0.};

/** stick values converted, filtered, scaled, biased */
volatile int stick[STICK_LAST] = {0};

/** Y-Matrix for Kalman who needs this ? \todo remove this */
volatile float Y[] = {0.,0.,0.,0.,0.,0.,0.,0.,0.}; // Y-Matrix for Kalman

/** magnet heading \todo heading ? */
volatile float heading = 0.;

/** target heading \todo targetheading ?*/
volatile float targetheading = 0.;

/** debug switch for dbg_mag() */
volatile int dbgmag = 0;

/** stored air pressure at ground */
volatile int alt_p0 = 0;
/** vertical (Z) acceleration from baro in m/sec */
float vzbaro = 0;

/** \todo accel_converted struct */
/** aktuelle Erdbeschleunigung während des Fluges */
volatile float g = 0.;

/** Erdbeschleunigung im Stand am boden */
volatile float g0 = 0.;

/** vertical (Z) acceleration from accel in m/sec */
float vzaccel = 0.;

/** correcting factor for vzaccel */
float corr_vzaccel = 1;

/** \todo refactoring, build gps_converted struct */
/** horizontal (E) acceleration from GPS in m/sec */
float ve = 0.;
/** horizontal (N) acceleration from GPS in m/sec */
float vn = 0.;
/** vertical (Z) acceleration from GPS in m/sec */
float vz = 0.;
/** actual position east from GPS in \todo unit missing */
int posx = 0;
/** actual position north from GPS in \todo unit missing */
int posy = 0;
/**
 * actual position to ground from GPS in \todo unit missing
 * \bug why is this variable not set or used, olri 17.12.2009
 */
int posz = 0;

//float filterbaro[] =
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
//
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
//
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
//                      0,0,0,0,0,0,0,0,0};
int barotmp = 0;
```

```
int baro_cnt;
float summe = 0.;

// local function prototypes
int rc_clipper(int in);

// functions

/**
 * rc_clipper(): expands the zero aera from thrust to +/- th
 * \return thrust
 */
int rc_clipper(int in) {

    int th = 32; // threshold [bit] (Schwelle)

    // *** Example for <th> = 10: ***
    if (in >= th) {      // in     = +11
        return in - th;  // 11-10  = 1
    }

    if (in < -th) {      // in     = -11
        return in + th;  // -11+10 = -1
    }

    return 0;            // in = -10..10 = 0
}

/**
 * timer2sec():
 *
 * \todo implement
 */
void timer2sec( void ){
//Woher Millisekunden bekommen?
}

/**
 * rc2stick(): convert raw RC data to stick values
 *
 * Implemented features:
 * - Scaling with RC_MUL
 * - low pass filter for thrust
 * - rc_clipper for thrust if HOVER_CTRL defined
 *
 * \return stick[]
 */
void rc2stick( void ) {
    /** \todo RC_MUL aus fc.c gehoert HIER her !  */

    // convert rc_rx to stick values
    stick[STICK_ROLL] = -rc_rx[RC_ROLL] * RC_MUL;
    stick[STICK_PITCH] = -rc_rx[RC_PITCH] * RC_MUL;
    stick[STICK_YAW] = rc_rx[RC_YAW];

    // low pass filter for thrust
    static signed char thrust = -128;
    thrust = (rc_rx[RC_THR]*10+thrust*0)/10;

#ifdef HOVER_CTRL
    stick[STICK_THRUST] = rc_clipper( thrust ); // -128 - +127 for Thr-Stick
#else
    stick[STICK_THRUST] = thrust+RC_OFFSET; // 0 - 255 for Thr-Stick.
#endif
}


/**
 * gps2meter():
 *
```

```
 * \return posx, posy, Y[0-2]
 * \todo cleanup
 */
void gps2meter( void ) {
    if(actualPos.state==3){
        posy=actualPos.north;//-startingpoint[1];
        posx=actualPos.east;//-startingpoint[0];
        posz=actualPos.altitude;
    } else {
        /** \todo 0 ist hoch gefaehrlich ! */
        posy=0;
        posx=0;
        posz=0;
    }
    /** \todo cleanup Y[]*/
    Y[0] = 0; //PosX Measure for Kalman
    Y[1] = 0; //PosY Measure for Kalman
    Y[2] = 0; //PosZ Measure for Kalman (better from Baro?)
}

/**
 * gps2ms():
 *
 * \return vn, ve
 */
void gps2ms( void ) {
    if(actualPos.state==3){
        vn=actualPos.velNorth;
        ve=actualPos.velEast;
    } else {
        vn=0;
        ve=0;
    }
}

/**
 * veloms():
 *
 * \return Y[3-5]
 */
void veloms( void ) {
    /** \todo cleanup Y[]*/
    Y[3] = 0; // Velocity X in m/s for Kalman
    Y[4] = 1; // Velocity Y in m/s for Kalman
    Y[5] = 0; // Velocity Z in m/s for Kalman
}

#ifdef TINY13
/**
 * accel2ms2():
 *
 * \note TINY13 version
 * \return accel[ACC_X], accel[ACC_Y], accel[ACC_Z]
 */
void accel2ms2( void ) {
    accel[ACC_X] = (float)(-adc_average[ADC_ACCX])/256*9.81;
    accel[ACC_Y] = (float)(-adc_average[ADC_ACCY])/256*9.81;
    accel[ACC_Z] = (float)(adc_average[ADC_ACCZ])/256*9.81;
}
#endif //TINY13

#ifdef HBC
/**
 * accel2ms2():
 *
 * \note HBC version
 * \todo cleanup
 * \return accel[ACC_X], accel[ACC_Y], accel[ACC_Z]
 */
void accel2ms2( void ) {
```

```
//    accel[ACC_X] = (float)(imu_average[IMU_ACCX])/535;
//    accel[ACC_Y] = -(float)(imu_average[IMU_ACCY])/535;
    accel[ACC_X] = -(float)(imu_average[IMU_ACCX])/535;
    accel[ACC_Y] = (float)(imu_average[IMU_ACCY])/535;
    accel[ACC_Z] = (float)(imu_average[IMU_ACCZ])/560;
}
#endif //HBC

#ifdef HBMINI
/**
 * accel2ms2():
 *
 * \note HBMINI version
 * \todo cleanup
 * \return accel[ACC_X], accel[ACC_Y], accel[ACC_Z]
 */
void accel2ms2( void ) {
#ifdef USE_MAX1168
#warning use of MAX1168 values for attitude
    accel[ACC_X] = -(float)(imu_average[IMU_ACCX])/535.;
    accel[ACC_Y] = (float)(imu_average[IMU_ACCY])/535.;
    accel[ACC_Z] = (float)(imu_average[IMU_ACCZ])/560.;
#else
#warning use of ADC values for attitude
    accel[ACC_X] = -(float)(adc_average[ADC_ACCX])/256*9.81;
    accel[ACC_Y] = (float)(adc_average[ADC_ACCY])/256*9.81;
    accel[ACC_Z] = (float)(adc_average[ADC_ACCZ])/256*9.81;
#endif
}
#endif //HBMINI

/**
 * accel2accel_norm_v():
 *
 * \return set globel accel_norm_v
 */
void accel2accel_norm_v( void ) {
    accel_norm_v.x = accel[ACC_X];
    accel_norm_v.y = accel[ACC_Y];
    accel_norm_v.z = accel[ACC_Z];
    norm_v( &accel_norm_v );
}

/**
 * gyro2degs():
 */
void gyro2degs( void ) {
    /** \todo empty function, remove */
}

#ifdef TINY13
/**
 * gyro2rads():
 *
 * \return gyro[G_ROLL], gyro[G_PITCH], gyro[G_YAW]
 * \note TINY13 version
 */
void gyro2rads( void ) {
    /** 150 grad/sec 10Bit, 3,3Volt, 1rad = 2Pi/1024 => Pi/512 */
    gyro[G_ROLL]  = (float) (adc_average[ADC_ROLL]) * PI / 512;
    gyro[G_PITCH] = -(float)(adc_average[ADC_PITCH]) * PI / 512;
    gyro[G_YAW]   = (float) (adc_average[ADC_YAW]) * PI / 512;
}
#endif //TINY13

#ifdef HBC
/**
 * gyro2rads():
 *
 * \return gyro[G_ROLL], gyro[G_PITCH], gyro[G_YAW]
```

```
 * \note HBC version
 */
void gyro2rads( void ) {
    /** 300 grad/sec 16 Bit, 4Volt, 1rad = 2Pi, 2Pi/32768 => Pi/17520 */
    gyro[G_ROLL]  = -(float)imu_average[IMU_ROLL]  * PI / 17520;
    gyro[G_PITCH] = -(float)imu_average[IMU_PITCH] * PI / 17520;
    gyro[G_YAW]   =  (float)imu_average[IMU_YAW]   * PI / 17520;
}
#endif //HBC

#ifdef HBMINI
/**
 * gyro2rads():
 *
 * \return gyro[G_ROLL], gyro[G_PITCH], gyro[G_YAW]
 * \note HBMINI version
 */
void gyro2rads( void ) {
#ifdef USE_MAX1168
    /** \todo GRAD? grad/sec 16 Bit, 4Volt, 1rad = 2Pi, 2Pi/32768 => Pi/17520 */
    gyro[G_ROLL]  =  -(float)imu_average[IMU_ROLL]  * PI / 17520;
    gyro[G_PITCH] =  (float)imu_average[IMU_PITCH] * PI / 17520;
    gyro[G_YAW]   =  -(float)imu_average[IMU_YAW]   * PI / 17520;
#else
    /** \todo GRAD? grad/sec 10Bit, 3,3Volt, 1rad = 2Pi/1024 => Pi/512 */
    gyro[G_ROLL]  = -(float) (adc_average[ADC_ROLL]) * PI / 512;
    gyro[G_PITCH] =  (float) (adc_average[ADC_PITCH]) * PI / 512;
    gyro[G_YAW]   = -(float) (adc_average[ADC_YAW]) * PI / 512;
#endif
}
#endif //HBMINI

/**
 * Minimalistic version to get angles from acceleration
 *
 * \todo why has this function 3 callers ?
 * \return g, angle[ANG_ROLL], angle[ANG_PITCH]
 */
void accel2euler( void ) {
    // Calculate g ( ||g_vec|| )
    g = sqrtf(accel[ACC_X] * accel[ACC_X] +
              accel[ACC_Y] * accel[ACC_Y] +
              accel[ACC_Z] * accel[ACC_Z]);

    //values in radians
#define OLD
#ifdef OLD
    angle[ANG_PITCH] = -asinf( accel[ACC_X] / g );
    angle[ANG_ROLL] = asinf( accel[ACC_Y] / g );
#endif

#ifdef NEW
    angle[ANG_PITCH] = -asinf( accel[ACC_X] / g );
    angle[ANG_ROLL] = atan2f( accel[ACC_Y], accel[ACC_Z] );
#endif
}

#ifdef TINY13
/**
 * baro2m():
 * converts from the units given by the Barometer to Meters and
 * writes it to the Variable for the application.
 *
 * \note TINY13 version note
 * \return adc_converted[ADC_ALT]
 * \todo cleanup, alt_p0 better baro.p0
 */
void baro2m( void ) {
    adc_converted[ADC_ALT] = (adc_average[ADC_ALT]-alt_p0)/-22.4745;
```

```
    /*barotmp+=adc_converted[ADC_ALT];
      baro_cnt++;
      if(barocnt>5){

      }*/

    //barotmp = (adc_average[ADC_ALT]-alt_p0)/-10.952380952380953;

    /*for(int i=5; i > 2; i--){
      filterbaro[i+1] = filterbaro[i];
      }

      filterbaro[0] = (adc_average[ADC_ALT]-alt_p0)/-10.952380952380953;

      float summe = 0;

      //filterbaro[0]+filterbaro[1]+filterbaro[2]+filterbaro[3]+filterbaro[4];
      for (int j=0;j<5;j++){
      summe+=filterbaro[j];
      }*/

    /*if(baro_cnt == 50){
      adc_converted[ADC_ALT] = summe/50;
      baro_cnt = 0;
      summe = 0;
      }else{
      summe+=(adc_average[ADC_ALT]-alt_p0)/-10.952380952380953;
      baro_cnt++;
      }*/

    //6.OG 424
    //5.OG 472
    //4.OG 515
    //3.OG 557
    //2.OG 601
    //1.OG 651
    //E 700
    //K 730
    // 1 OG = 4,2m

}
#endif //TINY13


#ifdef HBC
/**
 * baro2m():
 * converts from the units given by the Barometer to Meters and
 * writes it to the Variable for the application.
 *
 * \note HBC version note
 * \return adc_converted[ADC_ALT]
 */
void baro2m( void ) {
    adc_converted[ADC_ALT] = -(imu_average[IMU_ALT]-0);
}
#endif //HBC

#ifdef HBMINI
/**
 * baro2m():
 * converts from the units given by the Barometer to Meters and
 * writes it to the Variable for the application.
 *
 * \note HBMINI version note
 * \return adc_converted[ADC_ALT]
 */
void baro2m( void ) {
    adc_converted[ADC_ALT] = -(imu_average[IMU_ALT]-0);
}
```

```
#endif //HBMINI

/**
 * needs to be called every 250ms
 *
 * \return vzbaro
 */
void baro2ms( void ) {
    /** holds old altitude value for calculating vz */
    static float old_alti = 0;
    /** calcs the current speed in z-dir */
    float tmp = adc_converted[ADC_ALT];
    /** to be called every 250ms */
    vzbaro = (tmp-old_alti)*1/4;
    /** local store old altitude value */
    old_alti = tmp;
    /** \todo cleanup */
    //corr_vzaccel = vzbaro/vzaccel;
    //vzaccel = 0;
}

/**
 * accel2ms():
 *
 * \todo remove, useless
 */
void accel2ms( void ) {
    //vzaccel+=(g-g0)*1/100;//is every 10ms called
}

/**
 * getzspeed():
 *
 * \return vz
 */
void getzspeed( void ) {
    /** \todo cleanup */
    vz = vzbaro;//+(vzaccel*corr_vzaccel);
}

/**
 * mmag2rad(): calls mmag3_read(), MM3_Heading()
 *
 * \return heading = (MM3_Heading()*d2r
 * \todo cleanup
 */
void mmag2rad( void ) {
    /*if(MM3_Heading()>180){
      heading = (float)(-(360-MM3_Heading()));
      } else {
      heading = (float)(MM3_Heading());
      }*/

    /*if(MM3_Heading() < 25){
      heading = -1;
      } else if (335 < MM3_Heading()){
      heading = 1;
      }*/
#ifdef TINY
    mmag3_read();
#endif
    heading = ((float)MM3_Heading())*d2r;
    //heading = (float) MM3_Heading();
}

/**
 * getstuffforkalman(): calls gps2meter(), veloms(), sets Y[6-8]
 * \see gps2meter() , veloms , Y
 */
void getstuffforkalman( void ) {
```

```
    gps2meter();

    /** \todo cleanup Y[] */
    veloms(); // sets Y[3-5]
    Y[6]=angle[ANG_PITCH];
    Y[7]=angle[ANG_ROLL];
    Y[8]=heading;
}

/**
 * analogconversion(): calls baro2m(), accel2ms2(), gyro2rads()
 * \see baro2m(), accel2ms2(), gyro2rads()
 */
void analogconversion( void ) {
    baro2m();//20us
    accel2ms2();//80us
    gyro2rads();//40us
//    accel2accel_norm_v();
}

/**
 * @}
 */
```

```
/************************************************************************
 *
 * Hochschule Bremen
 * Forschungsprojekt: Autopilot fuer Schwebefluggeraete
 *                         -- HB-Quadrokopter --
 *                     Prof. Dr.-Ing. Heinrich Warmers
 *
 *  Copyright (C) 2009  Andreas Dei
 *
 *  This program is free software: you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation, either version 3 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program.  If not, see <http://www.gnu.org/licenses/>.
 *
 ************************************************************************/

/** \addtogroup IMU GPL-QC HB-Autopilot IMU routines
 * @{
 */

/** \file
 *
 * SPI Implementation for HB-IMU
 *
 * \author Andreas Dei 2009
 */

#ifndef SIMULATE
#include <lpc214x.h>
#else
#include <fake_lpc214x.h>
#endif

#include "types.h"
#include "config.h"
// #include "printf_P.h"
#include "board.h"
// #include "dbg.h"
#include "vic.h"
#include "timer.h"
#include "convert.h"

#include "imu.h"

#define SpiSend(a) { S0SPDR = a; }
/** defines for spi_int states
 * \todo, fill with next states
 */
enum spi_states { SPI_WRITE };

// variables

/** only for test */
unsigned short imu_valuetest[8] = {0};
/**  only for test */
unsigned short imu_valuetest_2[8] = {0};

// globals

/** export value for ltc value */
unsigned int ltc_value = 0xAAAAAAAA;
```

```
/** export array for imu values */
volatile int imu_average[IMU_LAST] = {0};

/** offset values which are subtract permanent */
imu_offset_t imu_offset[IMU_LAST] = {0};

/** count of raw values which build a average value */
unsigned int imu_average_nb[IMU_LAST] = {
    4,4,4,
    4,4,4,
    8,8,
    20,20,20,20,20,20,
    8,8,8,8,8,1 };

// prototypes
void __attribute__ ((interrupt("IRQ"))) spi_int(void);
void spi_start( void );

#ifdef HBC
/**
 * Initialises hardware unit spi0
 */
void spi_init(void) {

    // CS
    IO1DIR |= (1<<21)|(1<<22)|(1<<23); //CS pins as output

    // PIN for MAG_FLIP as GPIO / Direction = Output
    PINSEL0 &= ~((1<<14)|(1<<15));
    IO0DIR |= (1<<7);

    // Pins for SPI0
    PINSEL0 |= (1<<8)|(1<<10)|(1<<12);
    PINSEL0 &= ~((1<<9)|(1<<11)|(1<<13));

    S0SPCCR = 0x000E; //SPI SCK divider (60MHz/14=4,28MHz)

    unsigned short int s0spcr_var = S0SPCR;
    //            DataSizeSelBit, MasterMode. IRQ On
    s0spcr_var |= (1<<2)|(1<<5)|(1<<7);
    //               CPHA=0, CPOL=0, MSB, 16bitDataSize
    s0spcr_var &= ~((1<<3)|(1<<4)|(1<<6)|
                    ((1<<8)|(1<<9)|(1<<10)|(1<<11)) );
    S0SPCR = s0spcr_var;

    //set interrupt vector
    VIC_ADDR_SPI0 = (unsigned long) spi_int;
    // use it for SPI0 interrupt
    VIC_CTRL_SPI0 = VICVectCntl_ENABLE | VIC_Slot_SPI0;
    //enable SPI0 interrupt
    VICIntEnable = (1<<VIC_Slot_SPI0);

    // start spi interrupt
    spi_start();

    // enable SPI -- HF 3.12.09 enable SPI fehlte
    /** \bug sorry HF, aber das ist die falsche spi Schnittstelle */
    //olri SSPCR1 |= (1<<1); // SSE

}
#endif // HBC

#ifdef HBMINI
/**
 * cable switch, max1168 (channel) wiring to imu [index] lookup table
 */
unsigned char hbmini[] = {IMU_ACCX, IMU_ACCY, IMU_ACCZ, IMU_ROLL, IMU_PITCH,
IMU_YAW};

/**
```

```
 * Initialises hardware unit spi1
 */

void spi_init(void) {

    // CS
    IO1DIR |= (1<<21)|(1<<22)|(1<<23); //CS pins as output

    // initialize SSP in SPI mode
    SSPCPSR = 0x0E;  // SSP clock prescaler 0x0E => 4.35 Mhz, 0x4E => 768KHz
    SSPCR0 = 0x0F;  // SSP 16bit, SPI, CPOL=0, CPHA=0, max speed
    SSPCR1 = 0x02;  // SSP master mode and enable

    // SCK1, MISO1, MOSI1, SSEL1, P0.17 .. P0.20
    PINSEL1 &= ~((1<<2) | (1<<4) | (1<<6) | (1<<8));
    PINSEL1 |=  ((1<<3) | (1<<5) | (1<<7) | (1<<9));

    // now read until receive fifo is not empty
    while((SSPSR & (1<<2))) {
        SSPDR;
    }

    // read some garbage 3 times
    spi1_test_max1168();
    spi1_test_ltc2440();

#ifdef TODO
    //set interrupt vector
    VIC_ADDR_SPI1 = (unsigned long) spi1_int;
    // use it for SPI0 interrupt
    VIC_CTRL_SPI1 = VICVectCntl_ENABLE | VIC_Slot_SPI1;
    // enable SPI0 interrupt
    VICIntEnable = (1<<VIC_Slot_SPI1);

    // start spi interrupt
    spi1_start();
#endif
}

/**
 * Testcode spi1 with max1168
 *
 * Function polls the channels from max1168.
 * Code is not interrupt driven.
 *
 * \author Oliver Riesener
 */
void spi1_test_max1168(void) {
    unsigned char channel;
    unsigned short cmd_MAXA = (0<<8) | (0<<9) | (0<<10) | (0<<11) | (0<<12);
    // read all
    for( channel = 0; channel<(sizeof(hbmini)/sizeof(unsigned char)); channel++
) {
        //CS MAX1168A
        CSSEL(CS_MAX1168A);
        // send cmd + channel
        SSPDR = (unsigned short) (cmd_MAXA | (channel<<13));
        // send zero bytes for pulling the result
        SSPDR = (unsigned short) 0x0000;
        // now wait until all frames are sent
//          while(!(SSPSR & (1<<0)));
        // wait while RNE
        while(!(SSPSR & (1<<2)));
        // read garbage answer form cmd + channel
        SSPDR;
        // wait while RNE
        while(!(SSPSR & (1<<2)));
        // read data from fifo
        unsigned short imu_value = SSPDR;
        // store data into imu_average array
```

```
            imu_average[hbmini[channel]] = imu_value - imu_offset[hbmini[channel]];
        }
        // CS UNSEL
        CSSEL(CS_UNSEL);
}
/**
 * Testcode spi1 with LTC2440
 *
 * Function polls the LTC2440 via spi1 interface.
 * It's not interrupt driven.
 * This code is self rejecting, if measure period time (39ms) not done.
 *
 * \author Oliver Riesener
 */
void spi1_test_ltc2440(void) {
    static unsigned int ltc_cnt = 0;

    if (timer1_cnt > ltc_cnt) {
        // retrigger only every 50ms}
        ltc_cnt = timer1_cnt+49;
        //CS CS_LTC2440
        CSSEL(CS_LTC2440);
        // send LTC command, first 16 Bits
        SSPDR = (unsigned short) 0x4000;
        // send CLK as last 16 Bits
        SSPDR = (unsigned short) 0x0000;
        // now wait until all frames are sent
//          while(!(SSPSR & (1<<0)));
        // wait while RNE
        while(!(SSPSR & (1<<2)));
        // read MSB word
        unsigned int ltc_value_tmp = SSPDR;
        // wait while RNE
        while(!(SSPSR & (1<<2)));
        // read data from fifo
        ltc_value_tmp = (ltc_value_tmp << 16) | SSPDR;
        // test of valid data
        if ( (ltc_value_tmp & (1<<31)) == 1 ) {
            // error signaling
            imu_average[IMU_ALT] = 0;
        } else {
            //ignore first 3 bits, shift 5 pos. right make bit 5 to LSB
            //(delete bit 0-4)
            ltc_value = (ltc_value_tmp & 0x1FFFFFFF) >> 5;
            // store data into imu_average array
            imu_average[IMU_ALT] = ltc_value - imu_offset[IMU_ALT];
        }
        // CS UNSEL
        CSSEL(CS_UNSEL);
    }
}
#endif // HBMINI

/**
 * plain tests for max1168(A)
 * \note only test routine !!
 * \return imu_valuetest[]
 */
void spi_test(void) {
    int i;
    for (i=0; i<=7; i++){

        //CS MAX1168A
        CSSEL(CS_MAX1168A);

        // define control byte page 174
        unsigned short control_byte = (0<<8) | (0<<9) | (0<<10) | (0<<11) |
            (0<<12) | (i<<13);

        // send control_byte
```

```
        SpiSend (control_byte);
        while (!(S0SPSR & 0x80));

        // send clocks
        SpiSend (0x0000);
        while (!(S0SPSR & 0x80));

        // read values
        unsigned short tmp;
        tmp = S0SPDR;
        imu_valuetest[i]=tmp;

        // MAX1168A unselect
        CSSEL(CS_UNSEL);
    }
}

/**
 * plain test for max1168(B)
 * \note only test routine !!
 * \return imu_valuetest_2[]
 */
#ifdef HBC
void spi_test_2(void) {
    int i;
    for (i=0; i<=7; i++){

        //CS MAX1168B
        CSSEL(CS_MAX1168B);

        // define control byte page 174
        unsigned short control_byte = (0<<8) | (0<<9) | (0<<10) | (0<<11) |
            (0<<12) | (i<<13);

        // send control_byte
        SpiSend (control_byte);
        while (!(S0SPSR & 0x80));

        // send clocks
        SpiSend (0x0000);
        while (!(S0SPSR & 0x80));

        // read values
        unsigned short tmp;
        tmp = S0SPDR;
        imu_valuetest_2[i]=tmp;

        // MAX1168B unselect
        CSSEL(CS_UNSEL);
    }
}
#endif // HBC
#ifdef HBMINI
#endif // HBMINI

/**
 * plain test routine from ltc sensor
 * \note only test routine !!
 * \return ltc_value
 */
void spi_read_ltc( void ) {

    //control_byte = 27,5Hz = 36,36ms conversion time, 375nV noise, 24bit (p14)
    unsigned short control_byte = 0x4000;
    unsigned int ltc_value_tmp = 0;

    //CS LTC2440
    CSSEL(CS_LTC2440);

    SpiSend (control_byte);
```

```
    while (!(S0SPSR & 0x80));

    ltc_value_tmp = S0SPDR<<16;

    SpiSend (0x0000);
    while (!(S0SPSR & 0x80));
    ltc_value_tmp |= S0SPDR;

    //LTC2440 Unselect
    CSSEL(CS_UNSEL);

    //ignore first 3 bits, shift 5 pos. right make bit 5 to LSB
    //(delete bit 0-4)
    ltc_value = (ltc_value_tmp & 0x1FFFFFFF)>>5;
}

/**
 * select none and start spi0 interrupt routine by sending some chars
 */
void spi_start( void ) {
    // unselect all spi devices
    CSSEL(CS_UNSEL);

    // send dummy bytes to start spi interrupt
    SpiSend(0x050A);
}

/**
 * inner controlling state machine
 *
 * \param[out] *cs pointer to outer next control state
 * \param[out] *cmd_imu pointer to outer next imu state
 * \param[out] *stor_loc pointer to outer next storage location
 * \param[out] *mode_p pointer to outer next mode 8/16 byte access
 */
void sm_ctrl(unsigned int *cs, unsigned int *cmd_imu, unsigned int *stor_loc,
unsigned int *mode_p) {

    static unsigned int state = 0;
    static unsigned int channel = 0;
    static unsigned int sl = 0;
#ifdef HBC
    static unsigned int flip_h_l = 0;
    static unsigned int magnet_cnt = 0;
#endif // HBC
    static unsigned int ltc_cnt = 0;

    // ext clk | int. Ref. allways on | no scan
    unsigned short cmd_MAXA = (0<<8) | (0<<9) | (0<<10) | (0<<11) | (0<<12);

    // ext clk | int. Ref. allways on | no scan
#ifdef HBC
    unsigned short cmd_MAXB = (0<<8) | (0<<9) | (0<<10) | (0<<11) | (0<<12);
#endif

    //27,5Hz = 36,36ms conversion time, 375nV noise, 24bit (DS p14)
    unsigned short cmd_LTC= 0x4000;
    unsigned int redo = 0;

    do {
        switch( state ) {
        case 0:    //Gyro X,Y,Z, ACC X,Y
#ifdef IMU_DEBUG_HF
//      DACR=(1000 << 6); //-- DA-Wandler out
#endif
            redo = 0;
#ifdef HBMINI // sollte auch auf HBC funktionieren
            channel = 0;
            sl=0;
#endif // HBMINI
```

```
            /* kein break; */
        case 1:
        case 2:
        case 3:
        case 4:
#ifdef IMU_DEBUG_HF
            DACR=(state << 11); //-- DA-Wandler out
#endif
            *cmd_imu = cmd_MAXA | (channel<<13);
            *cs = CS_MAX1168A;
            *mode_p = 16;
            *stor_loc = sl;
            channel++;
            sl++;
            state++;
            break;
        case 5:   //ACC Z
#ifdef IMU_DEBUG_HF
            DACR=(state << 11); //-- DA-Wandler out
#endif
            *cmd_imu = cmd_MAXA | (channel<<13);
            *cs = CS_MAX1168A;
            *mode_p = 16;
            *stor_loc = sl;
            channel++;
            sl++;
            state++;
            break;
        case 6:   //ACC2 X
#ifdef IMU_DEBUG_HF
            DACR=(state << 11); //-- DA-Wandler out
#endif
            *cmd_imu = cmd_MAXA | (channel<<13);
            *cs = CS_MAX1168A;
            *mode_p = 16;
            *stor_loc = sl;
            channel++;
            sl++;
            state++;
            break;
        case 7: //ACCZ2
#ifdef IMU_DEBUG_HF
            DACR=(state << 11); //-- DA-Wandler out
#endif
            *cmd_imu = cmd_MAXA | (channel<<13);
            *cs = CS_MAX1168A;
            *mode_p = 16;
            *stor_loc = sl;
            channel = 0;
            sl++;

#ifdef HBC
            state++;
#endif // HBC
#ifdef HBMINI
            state = 19;
#endif // !HBMINI
            break;
#ifdef HBC

        case 8: //MAG X,Y,Z
#ifdef IMU_DEBUG_HF
            DACR=(state << 11); //-- DA-Wandler out
#endif
            if ( (timer1_cnt > magnet_cnt) && (flip_h_l == 1) ) {
// HF--
//      DACR=(state << 11); //-- DA-Wandler out
                magnet_cnt = timer1_cnt+1;
                *cmd_imu = cmd_MAXB | (channel<<13);
                *cs = CS_MAX1168B;
```

```
                *mode_p = 16;
                *stor_loc = sl;
                channel++;
                sl++;
                state++;
            } else {
                channel = 0;
                state = 11;
                sl = 11;
                redo = 1;
            }
            break;
        case 9:
#ifdef IMU_DEBUG_HF
            DACR=(state << 11); //-- DA-Wandler out
#endif
            *cmd_imu = cmd_MAXB | (channel<<13);
            *cs = CS_MAX1168B;
            *mode_p = 16;
            *stor_loc = sl;
            channel++;
            sl++;
            state++;
            break;
        case 10:
#ifdef IMU_DEBUG_HF
            DACR=(state << 11); //-- DA-Wandler out
#endif
            *cmd_imu = cmd_MAXB | (channel<<13);
            *cs = CS_MAX1168B;
            *mode_p = 16;
            *stor_loc = sl;
            channel = 0;
            sl++;
            state++;
            flip_h_l = 0;
            FLIP_L;
            break;
        case 11: //MAG X,Y,Z
#ifdef IMU_DEBUG_HF
            DACR=(state << 11); //-- DA-Wandler out
#endif
            if ( (timer1_cnt > magnet_cnt) && (flip_h_l == 0) ) {
// HF--
//      DACR=(state << 11); //-- DA-Wandler out
                magnet_cnt = timer1_cnt+1;
                *cmd_imu = cmd_MAXB | (channel<<13);
                *cs = CS_MAX1168B;
                *mode_p = 16;
                *stor_loc = sl;
                channel++;
                sl++;
                state++;
                redo = 0;
            } else {
                channel = 3;
                state = 14; // gleich zu 14 ist sinnvoller....
                sl = 14;
                redo = 1;
            }
            break;
        case 12:
#ifdef IMU_DEBUG_HF
            DACR=(state << 11); //-- DA-Wandler out
#endif
            *cmd_imu = cmd_MAXB | (channel<<13);
            *cs = CS_MAX1168B;
            *mode_p = 16;
            *stor_loc = sl;
            channel++;
```

```
            sl++;
            state++;
            break;
         case 13:
#ifdef IMU_DEBUG_HF
            DACR=(state << 11); //-- DA-Wandler out
#endif
            *cmd_imu = cmd_MAXB | (channel<<13);
            *cs = CS_MAX1168B;
            *mode_p = 16;
            *stor_loc = sl;
            channel++;
            sl++;
            state++;
            flip_h_l = 1;
            FLIP_H;
            break;
         case 14:  //TEMP,RP,TEMP2
#ifdef IMU_DEBUG_HF
            DACR=(state << 11); //-- DA-Wandler out
#endif
            redo = 0;
            /* kein break; */
         case 15:
         case 16:
#ifdef IMU_DEBUG_HF
            DACR=(state << 11); //-- DA-Wandler out
#endif
            *cmd_imu = cmd_MAXB | (channel<<13);
            *cs = CS_MAX1168B;
            *mode_p = 16;
            *stor_loc = sl;
            channel++;
            sl++;
            state++;
            break;
         case 17:  //FREE
#ifdef IMU_DEBUG_HF
            DACR=(state << 11); //-- DA-Wandler out
#endif
            *cmd_imu = cmd_MAXB | (channel<<13);
            *cs = CS_MAX1168B;
            *mode_p = 16;
            *stor_loc = sl;
            channel++;
            sl++;
            state++;
            break;
         case 18:
#ifdef IMU_DEBUG_HF
            DACR=(state << 11); //-- DA-Wandler out
#endif
            *cmd_imu = cmd_MAXB | (channel<<13);
            *cs = CS_MAX1168B;
            *mode_p = 16;
            *stor_loc = sl;
            channel = 0;
            sl++;
            state++;
            break;
#endif // HBC
         case 19:  //AP
#ifdef IMU_DEBUG_HF
            DACR=(state << 11); //-- DA-Wandler out
#endif
            if (timer1_cnt > ltc_cnt) {
                ltc_cnt = timer1_cnt+49; //only every 50ms
                *cmd_imu = cmd_LTC;
                *cs = CS_LTC2440;
                *mode_p = 32;
```

```
                *stor_loc = sl;
                sl = 0;
                state = 0;
                break;
            } else {
                state = 0; //wieder besser gleich bei 0 weitermachen
                redo = 1;
                sl = 0;
            }
            break;
        default:
#ifdef IMU_DEBUG_HF
            DACR=(state << 11); //-- DA-Wandler out
#endif
            state = 0;
            sl = 0;
            break;
        } // switch( state )
    } while ( redo );
}

/**
 * interrupt routine spi0, outer state machine
 *
 * Manages the communication to all SPI clients
 * features 8 and 16 Bit mode.
 * Calls inner state machine sm_ctrl() to know
 * whats to do.
 *
 * \todo states are not named
 * \return imu_average[]
 */
void spi_int(void) {

    /** command to send */
    static unsigned int cmd_word=0;
    /** chip select value on spi bus */
    static unsigned int chip_select=0;
    /** index to storage location in imu_* */
    static unsigned int stor_lo=0;
    /** defined actual spi mode 16 or 32 bit */
    static unsigned int mode=0;
    /** buffer to assemble the 32 bit ltc value */
    static unsigned int ltc_tmp=0;

    /** buffer for average calculation, counter */
    static int imu_value_cnt[IMU_LAST] = {0};
    /** state for statemachine */
    static unsigned int h_state = 0;

    /** tmp data storage */
    unsigned int imu_value;

    // clear SPIF bit
    S0SPSR &= ~(0x80);

    // State machine
    switch (h_state) {
    case 1:
        if( mode == 32 ) {
            // mode 32 read first data
            ltc_tmp = S0SPDR<<16;
            //send 2nd dummy word
            SpiSend (0x0000);
        } else {
            //mode 16 send dummy word
            SpiSend (0x0000);
        }
        h_state = 2;
        break;
```

```
    case 2:
        if( mode == 32 ) {
            // mode 32 read second data
            ltc_tmp |= S0SPDR;
            if ( (ltc_tmp & (1<<31)) == 1 ) {
                /** \bug this was not correct here, imu_average is a
                    exported variable and the result is a average
                    calculated value.
                */
                // set error
                imu_average[stor_lo] = 0;
                /** \note Workaround, Needs to be tested in hardware */
                // fake imu value to get 0
                imu_value = imu_offset[stor_lo];
                // trigger the average calculation
                imu_value_cnt[stor_lo] = imu_average_nb[stor_lo]-1;
            } else {
                //ignore first 4 bits, !!! sign bit canceled !!!
                // shift 5 pos. right make bit 5 to LSB
                imu_value = ((ltc_tmp & 0x0FFFFFFF) >> 5);
            }
        } else {
            //mode 16 read data
            imu_value = S0SPDR;
        }

        // --HF export value
        imu_average[stor_lo] = imu_value - imu_offset[stor_lo];

        h_state = 0;

        if( mode == 32 ) {
            //mode 32 send cmd word
            SpiSend (cmd_word);
            break;
        }
        /* if mode = 16 no break; */
    case 0:
        sm_ctrl (&chip_select, &cmd_word, &stor_lo, &mode);
        CSSEL(chip_select);
        if( mode == 32 ) {
            //Send Dummy Word
            SpiSend (0x0000);
        } else {
//--HF
            /** state machine stops here, is triggerd again form timer1_int() */
            if (stor_lo != 6) {
                //Send CMD Word
                SpiSend (cmd_word);
            }
        }
        h_state = 1;
        break;
    default:
        h_state = 0;
        //send dummy word
        SpiSend (0x0000);
        break;
    } // switch h_state

    // clear spi0 interrupt bit
    S0SPINT = 0x01;

    // clear interrupt vector
    VICVectAddr = 0x00000000;
}

/**
 * clear imu_offset[], get actual imu_average[] values, store into imu_offset[]
 */
```

```
void imu_offset_set(void) {
    unsigned int i;
    // set neutral offset values to zero
    for ( i=0; i<(sizeof(imu_offset)/sizeof(imu_offset_t)); i++) {
        imu_offset[i] = (imu_offset_t)0;
    }
    // wait for new imu_average values with of neutral offset
    /** \todo low, better implement a delay function here */
#ifdef HBMINI
    // get values
    spi1_test_max1168();
    spi1_test_ltc2440();
#else
#ifndef SIMULATE
    unsigned int tmp;
    for(tmp = 0;tmp<1000000;tmp++){asm("nop");};
    for(tmp = 0;tmp<1000000;tmp++){asm("nop");};
#endif // ! SIMULATE
#endif // HBMINI
    // set GYRO neutral values
    imu_offset[IMU_ROLL] = imu_average[IMU_ROLL];
    imu_offset[IMU_PITCH] = imu_average[IMU_PITCH];
    imu_offset[IMU_YAW] = imu_average[IMU_YAW];
    // set ACC neutral values

    imu_offset[IMU_ACCX] = imu_average[IMU_ACCX];
    imu_offset[IMU_ACCY] = imu_average[IMU_ACCY];
    /** \todo medium, earth-accel is sensor/quadcopter depend, should stored in
parameters */
    //s04_ande    imu_offset[IMU_ACCZ] = imu_average[IMU_ACCZ]-4800; // -4800
earth-accel
    /** \todo low: ACCZ is sensor depend */
    imu_offset[IMU_ACCZ] = imu_average[IMU_ACCZ]-5500; // -5500  earth-accel
// for HF    imu_offset[IMU_ACCZ] = imu_average[IMU_ACCZ] - 4125;  // -4125
earth-accel (Dei 5500)
    // set ACC neutral values
    imu_offset[IMU_ACCX2] = imu_average[IMU_ACCX2];
    /** \todo medium, earth-accel is sensor/quadcopter depend, should stored in
parameters */
    imu_offset[IMU_ACCZ2] = imu_average[IMU_ACCZ2]-2784; // -2784 earth-accel
    // set ALT and RP neutral values
    imu_offset[IMU_ALT] = imu_average[IMU_ALT];
    imu_offset[IMU_RP] = imu_average[IMU_RP];
}

/**
 * @}
 */
```

```
/*********************************************************************
 *
 * Hochschule Bremen
 * Forschungsprojekt: Autopilot fuer Schwebefluggeraete
 *                         -- HB-Quadrokopter --
 *                     Prof. Dr.-Ing. Heinrich Warmers
 *
 *  Copyright (C) 2009  Oliver Riesener
 *
 *  This program is free software: you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation, either version 3 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program.  If not, see <http://www.gnu.org/licenses/>.
 *
 *********************************************************************/

/** \addtogroup LED GPL-QC Led / Buzzer routines
 * @{
 */

/**
 * \file
 *
 * Implements routines for led and buzzer
 *
 * \author Oliver Riesener
 */

#include <lpc214x.h>

#include "config.h"

#include "led.h"
/**
 * Initialises ports for leds
 */
void led_init(void) {
    // *** LED first steps
#ifdef TINY13
    PINSEL2 &= ~((1 << 3) | (1 << 2)); // TRACE / DEBUG port as GPIO
    /** \todo - LOW - buzzer need's to be separated */
    IODIR0 |= (1 << 11); // buzzer output
    IODIR1 |= (1 << 28) | (1 << 19); // led rot and led green output
#endif //TINY13
#ifdef HBC
    PINSEL2 |= (1<<2); //DBUG port as DBUG/JTAG
    PINSEL2 &= ~(1<<3); // TRACE port as GPIO
    /** todo - LOW - buzzer need's to be separated */
    IODIR1 |= (1 << 20); // buzzer output
    IODIR1 |= (1 << 18) | (1 << 19); // led rot and led green output
#endif //HBC
#ifdef OLIMEX
    PINSEL2 &= ~((1 << 3) | (1 << 2)); // TRACE / DEBUG port as GPIO
    /** todo - LOW - buzzer need's to be separated */
    IODIR0 |= (1 << 12) | (1 <<13); // buzzer output
    IO0CLR = ( 1 << 13); // buzzer gnd
    IODIR0 |= (1 << 11) | (1 << 10); // led rot and led green output
#endif //OLIMEX
#ifdef HBMINI
    // PINSEL2 |= (1<<2); //DBUG port as DBUG/JTAG
    PINSEL2 &= ~(1<<2); // DBUG port as GPIO
    PINSEL2 &= ~(1<<3); // TRACE port as GPIO
```

```
    /** todo - LOW - buzzer need's to be separated */
    IODIR1 |= (1 << 20); // buzzer output
    IODIR1 |= (1 << 19) | (1 << 18); // led rot and led green output
    // power output
    IODIR1 |= (1 << 25); // cam_pwr_sw output
#endif //HBMINI
}

/**
 * runs pattern on led ports
 *
 * \param [in] count n times
 */
void led_test( unsigned int count) {
    volatile unsigned int tmp;
    int i;
    for (i = 0; i < 5; i++) {
        LED_RED_ON;
        for (tmp = 0; tmp < count; tmp++) {
            asm("nop");
        }
        LED_RED_OFF;
        LED_GREEN_ON;
        for (tmp = 0; tmp < count; tmp++) {
            asm("nop");
        }
        LED_GREEN_OFF;
    }
    // buzzer test, toggle pin
    BUZZER_FLASH;
}

/**
 * toggle the buffer port n times
 *
 * \param [in] count n times
 */
void buzzer_beep( unsigned int count ) {
    volatile unsigned int tmp;
    int i;
    for (i = 0; i < count; i++) {
        BUZZER_ON;
        for (tmp = 0; tmp < 50; tmp++) {
            asm("nop");
        }
        BUZZER_OFF;
        for (tmp = 0; tmp < 50; tmp++) {
            asm("nop");
        }
    }
}

/**
 * set cam power on / off
 *
 * \param [in] onoff
 */
void cam_switch( unsigned char onoff ) {
    if ( onoff ) {
        IOCLR2 = (1<<25);
    } else {
        IOSET1 = (1<<25);
    }
}
/**
 * @}
 */
```

```
/************************************************************************
 *
 * Hochschule Bremen
 * Forschungsprojekt: Autopilot fuer Schwebefluggeraete
 *                         -- HB-Quadrokopter --
 *                     Prof. Dr.-Ing. Heinrich Warmers
 *
 *  Copyright (C) 2009  Oliver Riesener, Heinrich Warmers,
 *                      Christoph Niemann, Andreas Dei
 *
 *  This program is free software: you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation, either version 3 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program.  If not, see <http://www.gnu.org/licenses/>.
 *
 ************************************************************************/

/** \addtogroup MAIN GPL-QC main routines
 * @{
 */

/** \file
    This file includes then main routine.
    The main routine initialises the hardware registers, timers and irqs by
    calling the *_init() routines. After that it end in a while(1) endless
    loop which generates dbg_* outputs.

    \mainpage

    \section main_intro Introduction

    This documentation describes the \b "HB-Autopilot" onboard software
system:\n
    \n
    <pre>
    Hochschule Bremen
    Forschungsprojekt: Autopilot fuer Schwebefluggeraete
    -- HB-Quadrokopter --
    Prof. Dr.-Ing. Heinrich Warmers

    Copyright (C) 2009  see authors.h

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.

    </pre>
*/
#include <lpc214x.h>
#include <math.h>

#include "types.h"
#include "config.h"
```

```
#include "dac.h"
#include "analog.h"

#ifdef USE_UART_IRQ
#include "uart_hw.h"
#else
#include "uart.h"
#endif // USE_UART_IRQ

#include "i2c.h"
#include "printf_P.h"
#include "viclowlevel.h"
#include "timer.h"
#include "rc.h"
#include "fc.h"
#include "drive.h"
#include "mmag3.h"
#include "altitude.h"
#include "dbg.h"
#include "mcu.h"
#include "mm3.h"
#include "convert.h"
#include "led.h"
#include "cmd.h"
#include "mygps.h"
#include "nav.h"
#include "board.h"
#include "imu.h"
#include "kalman_hb.h"
#include "kompl_quat.h"
#include "hover_thrust.h"
#include "hover_control.h"
#include "rtwtypes.h"
#include "myway.h"
#include "servos_4017_hb.h"


#define DAC_MAIN

/** \note
 * Wichtiger Hinweis,
 * die im Programmierhandbuch LPC214x UserManual UM10139 angegebenen
 * DEFAULT Registerinhalte sind durch den verwendeten
 * Bootloader bereits geaendert. D.h. auch defaultmässige 0 Werte müssen
 * gesetzt bzw. geloescht werden.
 */

/**
 * main routine called at startup
 */
int main(void) {

#define OK_NO_WAYPOINT_TEST

    volatile unsigned int tmp;

    // *** LED, BUZZER first steps
    led_init();

    // *** interrupt init, disable all irqs
    VICIntEnClr = 0xFFFFFFFF;
    // select all irqs as irq / not fiq
    /*
     * @todo - LOW - implement fast irq, with firq.h by olri
     */
    VICIntSelect = 0x00000000;

    // *** init mcu
    // set PLL, MAM
    mcu_init();
```

```
    // test new clk
    led_test(5 * 100000);
    buzzer_beep(100);

    // *** IRQ enable
    enableIRQ();

#ifdef OK_NO_WAYPOINT_TEST
    // *** initialize internal analog converters adc0, adc1
//#ifdef TINY13
    adc_init();
//#endif

    // *** configure DAC output, REMOVE jumper to luftdruck4
#ifdef DAC_MAIN
    dac_init(); // after ADC_Init, overwrites PINSEL register
#endif

    // *** init i2c
    i2c0_init();
#endif // OK_NO_WAYPOINT_TEST

    // *** setup the UART
#ifdef USE_UART_IRQ
    // UART0
    uart0_init_tx();
    uart0_init_rx();
    // UART1
    uart1_init_tx();
    uart1_init_rx();
#else
    uart0Init( B38400, UART_8N1, UART_FIFO_8 );
    uart1Init( B38400, UART_8N1, UART_FIFO_8 );
#endif

    // Send a sign
    dbg_clean(1);

    // init cmd
    cmd_init();
/*********************
#ifndef OK_NO_WAYPOINT_TEST
    // wait for ever
    while(1) {
        // *** commands ?
        cmd_read();
#ifdef WAYPOINT
        // scan for gps bytes
        GPSdbg();
        // call gps 250ms
        static unsigned int gps_cnt = 0;
//        if (!(timer1_cnt <= gps_cnt)) {
        gps_cnt = timer1_cnt + 249;
        gps2meter();
        // waypoint
        myway_gps2m();
        myway();
        dbg_myway(10);
//      }

        // Flash both LEDs if there is a Fix
        if( actualPos.state == 3 ) {
            LED_GREEN_FLASH;
            LED_RED_FLASH;
        } else {
            LED_RED_OFF;
        }
#warning "flight with waypoint navigation"
#endif // WAYPOINT
```

```
    }
#endif // ! OK_WAYPOINT_TEST
*********************/
    // init timer1
    timer1_init();
    // spi 0/1 init hardware depend

#if defined HBC || defined HBMINI
    spi_init();
#endif

    // Animation
    while (0) {
        LED_RED_ON;
        LED_GREEN_OFF;
        for (tmp = 0; tmp < 1000000; tmp++) {
            asm("nop");
        }
        LED_RED_OFF;
        LED_GREEN_ON;
        for (tmp = 0; tmp < 1000000; tmp++) {
            asm("nop");
        }
    }

#ifdef SERVOS
    // init servos before rc_init()
    servos_init();
#endif // SERVOS

    // init rc
    rc_init();

#ifdef TINY13
    // init mmag3
    mmag3_init();
#endif

    // init MM3
#ifdef HBC
#define MM3_WEG
#ifdef MM3
    MM3_Init();
#endif
#endif

#ifdef HBMINI
//    HMC5843_init();
#endif // HBMINI

#ifdef TINY13
    // get zero attitude
    adc_offset_set();
#endif //TINY13

#ifdef HBC
    // get zero attitude
    imu_offset_set();
#endif //HBC
#ifdef HBMINI
    // get zero attitude
    adc_offset_set();
    // get zero attitude
    imu_offset_set();
#endif //HBMINI

    LED_RED_ON;
    LED_GREEN_OFF;

#ifdef MM3
```

```
    // calibrate MMAG3 if throttle at startup
    if (rc_rx[RC_THR]> 0) {
        MM3_Calibrate();
    }
#endif // MM3

#ifdef TINY13
    // altitude init
    alt_init();
#endif

    // Initialize Kalman-Filter
#ifdef KALMAN_HB
    kalman_hb_init(10);
#endif

    // Initialize Kompl_quat
#ifdef KOMPL_QUAT
    mmag2rad();
    accel2accel_norm_v();
    kompl_quat_init( 10 );
#endif

    // store heading at startpos
    /** \bug heading ist hier nicht initialisiert ! es fehlen euler winkel und
mmag2rad() ! */
    targetangle = heading;


    // All off ready to fly
    LED_RED_OFF;

    // init myway
#ifdef WAYPOINT
    myway_init();
#endif // WAYPOINT

    // wait for stable system, e.g. 4 sec after start
    while ( timer1_cnt < 4000 ) {
        print_f(".");
    }
    print_f("\r\n");


    // endless loop
    while (1) {

        // dac_sync impulse
//        dac_sync();

        // drive switch off, stick lower left
        if (rc_rx[RC_THR] < -RC_STICK_LIMIT && rc_rx[RC_YAW] > RC_STICK_LIMIT) {
            drive_off();
        }
        // drive switch on, if stick lower_right and rc_status == RC_OK
        //                  and left and right switches down
        if (rc_rx[RC_THR] < -RC_STICK_LIMIT && rc_rx[RC_YAW] < -RC_STICK_LIMIT
            && (rc_status == RC_OK) ) {
            if ((rc_rx[RC_SWL] > 0) && (rc_rx[RC_SWR] > 0)) {
                // SWITCH ON
                drive_on();
            } else {
                // SWITCH ERROR
                buzzer_beep(25);
            }
        }
        // sensors calibrate offsets
        if (drive_mode == DRIVE_OFF && rc_rx[RC_THR] > RC_STICK_LIMIT
            && rc_rx[RC_YAW] > RC_STICK_LIMIT) {
            int i;
```

```
            for (i = 0; i < 2; i++) {
//                buzzer_beep(1000);

/** \bug causes the pitch and roll-control not to work */
/*
  #ifdef TINY13
  // initialise offsets
  adc_offset_set();
  #endif //TINY13*/

                //
                //#ifdef HBC
                //                  // initialise offsets
                //                  imu_offset_set();
                //#endif //HBC
                //
                //                  if(actualPos.state==3){
                //                      startingpoint[0]=actualPos.east;
                //                      startingpoint[1]=actualPos.north;
                //                  }
            }

#ifdef TINY13
            // store altitude at start position
            alt_p0 = adc_average[ADC_ALT];
#endif // TINY13
#ifdef HBC
            // store altitude at start position
            alt_p0 = imu_average[IMU_ALT];
#endif // HBC
#ifdef HBMINI
            // test code for spi1 max1168 and ltc2440
            /** \todo need to be tested */
            spi1_test_max1168();
            spi1_test_ltc2440();

            // store altitude at start position
            alt_p0 = imu_average[IMU_ALT];
#endif // HBMINI

            // initialise new heading
            targetheading = heading;
        }

        // altitude hold on if left switch up
        if ( rc_rx[RC_SWL] < 0) {
            if ( !altiswitch ) {
                altiswitch = 1;
                LED_GREEN_ON;
            }
        } else {
            if ( altiswitch ) {
                // don't hold if switch down
                altiswitch = 0;
                LED_GREEN_OFF;
            }
        }

        // yaw correct if right switch up
        if ( rc_rx[RC_SWR] < 0) {
            if ( !yawswitch ) {
                yawswitch = 1;
                LED_RED_ON;
            }
        } else {
            // don't correct if down
            if ( yawswitch ) {
                yawswitch = 0;
                LED_RED_OFF;
            }
```

```
        }

#ifdef WAYPOINT
        // scan for gps bytes
        GPSdbg();

        // call gps 250ms
        static unsigned int gps_cnt = 0;
        if (!(timer1_cnt <= gps_cnt)) {
            gps_cnt = timer1_cnt + 249;
            gps2meter();
            dbg_gps_stat(1);
            dbg_gps(0);
            // waypoint
            myway_gps2m();
#ifdef MYWAY
            myway();
            dbg_myway(0);
#endif // MYWAY
        }

        // Flash both LEDs if there is a Fix
        if( actualPos.state == 3 ) {
            LED_GREEN_FLASH;
            LED_RED_FLASH;
        } else {
            LED_RED_OFF;
        }
#warning "flight with waypoint navigation"
#endif // WAYPOINT

        //call Kalman, Kalman Hover if kalman_run = 1
        if (kalman_run) {
            int mydt = timer1_cnt - timer1_cnt_old;
            timer1_cnt_old = timer1_cnt;
#ifdef KALMAN_HB
            // call kalman
            dbg_kalman_hb(0);
            kalman_hb_run( mydt );
            // copy euler[ARRAY] to euler_v.[xyz]
            euler_v.x = euler[EULER_ROLL];
            euler_v.y = euler[EULER_PITCH];
            //euler_v.z = euler[EULER_YAW];
#warning "flight with kalman"
#else
#ifdef KOMPL_QUAT
            mmag2rad();
            accel2accel_norm_v();
//          dbg_mymag( 0 );
//          print_f("heading(GRAD)=%3d\r\n", MM3_Heading());
            // call kompl_quat
            kompl_quat_run( mydt );
            // copy values to euler array
            euler[EULER_ROLL] = euler_v.x;
            euler[EULER_PITCH] = euler_v.y;
            // euler[EULER_YAW] = euler_v.z;
#warning "flight with komplementaer"
#else
            // set actual angle from accel
            accel2euler();
#warning "flight with accel2euler"
#endif
#endif
            // reset run flag
            kalman_run=0;
        }
//      print_f("\r\nEULER_V.[xyz]: %i %i %i\r\n", euler_v.x, euler_v.y,
euler_v.z);

#ifdef HOVER_CTRL
```

```
        // call kalman_hover + hover_control
        s_hov = kalman_hover(); // + hover_control( accel[ACC_Z] );
#endif

        // call kalman 10ms (and the other stuff down here (dbg, etc))
        static unsigned int kalman_cnt;
        if (!(timer1_cnt <= kalman_cnt)) {
            kalman_cnt = timer1_cnt + 9;

//          led_test(5 * 10000);

            getzspeed(); // setzt vz aus dem vzbaro wert in vz/sec

            // clear screen every sec, give values, if we didn't fly
            if (rc_rx[RC_THR] + RC_STICK_OFFSET < 45) {
                static unsigned int main_timer = 0;
                if (main_timer++ > 10) {
                    // dbg_clean(1);
                    // dbg_rc(2);
                    // dbg_adc(3);
                    // dbg_adc_offset(4);
                    // dbg_capture(1);
                    // dbg_sensors(5);
                    // dbg_kalman_hb(6);
                    // dbg_drive(7);
                    // dbg_mag_capture(1);
                    // dbg_mmag3(8);
                    // dbg_mm3(10);
                    // dbg_fc(9);
                    // dbg_dbg(11);
                    // dbg_mag(2);
                    // dbg_kal_ang(3);
                    // dbg_alti(15);
                    // dbg_pos(16);
                    // dbg_imu(12);
                    // dbg_sensors_conv_angle(6);
                    main_timer = 0;
                }
            }

#ifdef TINY
            mmag2rad();
            do_navigation();
#endif
        } // if 10 ms

#ifdef HOVER_CTRL
        // Hover thrust debugging
        //    dbg_hover_thrust(0);
        if ( stick[STICK_THRUST] == 0) {
            LED_RED_ON;
        } else {
            LED_RED_OFF;
        }
#endif

        // *** commands ?
        cmd_read();

        // *** i2c_wdt testing
        if( i2c0_status == I2C_WDT_ERROR ) {
            LED_RED_ON;
            if( drive_mode == DRIVE_ON ) {
                i2c0_stop();
                i2c0_start();
            }
            LED_RED_OFF;
            i2c0_wdt_error_cnt++;
            dbg_i2c0(0);
        }
```

```
        // *** high speed capture values
        // dbg_adc(3); // alter kram, use adc_capture
        // dbg_adc_offset(4);
  //  dbg_adc_capture(0);
        // dbg_sensors(5);
        // dbg_mag_capture(1);
        // dbg_kalman_hb(0);
        // dbg_drive(7);
        // dbg__mmag3(8);
        // dbg_fc(9);
        // dbg_mm3(10);
        // dbg_dbg(11);
        // dbg_mag(2);
        // dbg_kal_ang(3);
        // dbg_alti(15);
        // dbg_mag_capture(10);
        dbg_imu_full(0);
        // dbg_imu(0);
        // dbg_rc(4);
        // dbg_sensors_conv(0);
        // dbg_adc_capture(1);
        // dbg_hover_thrust(0);
        // dbg_control(0);
        // dbg_rc(0);
//      dbg_stick(0);
        // dbg_drive(0);

        // dbg_mymag( 0 );
        // dbg_model( 0 );
        // dbg_servo( 0 );
    } // while(1)

    // not reached
    return 0;
}

/**
 * @}
 */
```

```
/************************************************************************
 *
 * Hochschule Bremen
 * Forschungsprojekt: Autopilot fuer Schwebefluggeraete
 *                         -- HB-Quadrokopter --
 *                     Prof. Dr.-Ing. Heinrich Warmers
 *
 *  Copyright (C) 2009  Oliver Riesener
 *
 *  This program is free software: you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation, either version 3 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program.  If not, see <http://www.gnu.org/licenses/>.
 *
 ************************************************************************/

/** \addtogroup TIMER LPC21xx timer1 routines
 * @{
 */

/**
 * \file
 *
 * Implementation of timer1 initialisation and interrupt routines
 *
 * \author Oliver Riesener
 *
 * \todo missing doxytags on variables
 */

#include <lpc214x.h>

#include "types.h"
#include "config.h"
#include "analog.h"
#include "drive.h"
#include "i2c.h"
#include "rc.h"
#include "fc.h"
#include "led.h"
#include "convert.h"
#include "vic.h"
#include "mcu.h"
#include "config.h"
#if defined HBMINI && defined USE_MAX1168
#include "imu.h"
#endif

#include "timer.h"

#define SpiSend(a) { S0SPDR = a; } // --HF for trigger spi_int()

volatile unsigned char buzzer = 0;
volatile unsigned char buzzer_on_int = 0;
volatile unsigned int timer1_cnt = 0;
volatile unsigned int timer1_cnt_old = 0;
volatile int kalman_run = 0;
volatile int kalman_period = 0;

// function prototypes
void __attribute__ ((interrupt("IRQ"))) timer1_int(void);
```

```
/**
 * Interrupt routine called 1kHz = 1ms timer
 *
 * Features:
 * - increments timer_cnt every 1ms
 * - watches irq for rc is working
 * - watches for enough battery power
 * - buzzer on/off
 * - implements a raw safty function
 * - triggers the kalman_run flag
 * - calls analogconversion()
 * - calls flight_control()
 * - calls baro2ms()
 * - calls accel2ms()
 * - watches irq for i2c is working
 *
 * \note function of the year !
 * \todo make a light weight function of me
 */
void timer1_int(void) {

#undef DAC_TMR1
#ifdef DAC_TMR1
    unsigned long dacr_val = DACR;
    DACR=(500<<6);
#endif // DAC_TMR1

    // timer1_cnt increase
    timer1_cnt++;

    // test rc_int working, watchdog, no impulses from rc
    if( rc_watchdog++ > 26 ) { // n ms no clearing
        rc_status = RC_LOST;
    }

#ifdef TINY13
    // Vbat watch
    /** \bug VBAT watch here, or in config.h configured ? */
    if( (adc_average[ADC_VBAT]<527) || (rc_status == RC_LOST) ) {
        buzzer_on_int = ON;
    } else {
        buzzer_on_int = OFF;
    }
#endif //TINY13

#ifdef HBC
    // Vbat watch
    /** \bug VBAT watch here, or in config.h configured ? */
    if( (adc_average[ADC_VBAT]<239) || (rc_status == RC_LOST) ) {
        buzzer_on_int = ON;
    } else {
        buzzer_on_int = OFF;
    }
#endif //HBC

#ifdef HBMINI
    // Vbat watch
    /** \bug VBAT watch here, or in config.h configured ? */
    if( (adc_average[ADC_VBAT]<239) || (rc_status == RC_LOST) ) {
        buzzer_on_int = ON;
    } else {
        buzzer_on_int = OFF;
    }
#endif //HBMINI

    // buzzer
    if( (buzzer > OFF) || (buzzer_on_int > OFF) ) {
        if( !(timer1_cnt % BUZZER_MODULO) ) {
            BUZZER_FLASH;
        }
```

```
    } else {
        BUZZER_OFF;
    }

    // safty options
    switch( rc_status ) {
    case RC_OK:
        break;
    case RC_ERROR:
        break;
    case RC_LOST:
        /** set SAFETY_FUNCTION on or off */
#define SAFETY_FUNCTION
//#undef SAFETY_FUNCTION
        /** \todo safty function is turned off at the moment ! */
#ifdef SAFETY_FUNCTION
        /** \todo implement a reliable safty function */
        drive_off();
#else
#warning "SAFTY FUNCTION is turned off at the moment !"
#endif
        break;
    default:
        break;
    }

    //trigger the kalman to run (or not if its not the time)
    if( kalman_period && (timer1_cnt % kalman_period) == 0) {
        kalman_run = 1;
    }

    // olri
#ifdef IMU_DEBUG_HF
    DACR=(900 << 6); // DA-Wandler out
#endif
//--HF--
    // Hier IMU anwerfen
    if( (timer1_cnt % 5) == 0) {
#if defined HBMINI && defined USE_MAX1168
        // test code for spi1 max1168 and ltc2440
        spi1_test_max1168();
        spi1_test_ltc2440();
#endif
        analogconversion(); // moved from analog.c to here
#ifdef HBC
        SpiSend( 0x0000 );
#endif
    }
#ifdef IMU_DEBUG_HF
    DACR=(0); //-- DA-Wandler out
#endif

    /// get stick values
    rc2stick();

    // call control
    if( drive_mode == DRIVE_ON ) {
//--    DACR=(750 << 6); // DA-Wandler out
        flight_control();
    }

    // call convert after flight_control
    if( (timer1_cnt % 250) == 0) {
//--    DACR=(500 << 6); // DA-Wandler out
        baro2ms();
//--    DACR=(0); // DA-Wandler out

    }
    if( (timer1_cnt % 10) == 0) {
        accel2ms();
```

```
    }

    // test i2c0_wdt
    i2c0_wdt++;
    if( i2c0_wdt > I2C0_WDT_TIMEOUT ) {
        i2c0_status = I2C_WDT_ERROR;
    } else {
        i2c0_status = I2C_WDT_OK;
    }

    // timer irq handling
    T1IR = TxIR_MR0; // Clear interrupt flag by writing 1 to Bit 0

#ifdef DAC_TMR1
    DACR=dacr_val;
#endif // DAC_TMR1

    // clear interrupt vector
    VICVectAddr = 0x00000000;
}

/**
 * Initialise the timer1 of the LPC21xx NXP processor
 * - timer runs at cclk = FOSC*PLL_M
 */
void timer1_init(void) {
    // Compare-hit (1Khz)
    T1MR0 = (FOSC*3/TIMER1_DIV)-1;
    // Interrupt and Reset on MR0
    T1MCR = TxMCR_MR0I | TxMCR_MR0R;
    // Timer1 Enable
    T1TCR = TxTCR_COUNTER_ENABLE;

    // set interrupt vector
    VIC_ADDR_TIMER1 = (unsigned long)timer1_int;
    // use it for timer1 interrupt
    VIC_CTRL_TIMER1 = VICVectCntl_ENABLE | VIC_Slot_Timer1;
    // enable timer1 interrupt
    VICIntEnable = (1<<VIC_Slot_Timer1);
}

/**
 * @}
 */
```