**DIPLOMARBEIT**

Gesamtprojekt

**FPV-Drohne**

**Elektronik FPV-Drohne**

Marcel Bieder 5AHEL

**Softwareentwicklung FPV-Drohne**

Maximilian Lendl 5AHEL

**CAD-Entwicklung & Datenübertragung**

Ben Heinicke 5AHEL

**Entwicklung einer APP für Smartphone & Videoübertragung**

Sebastian Hinterberger 5AHEL

Betreuer: Dipl.-Ing. Josef Reisinger

Schuljahr 2023/24

Abgabevermerk:

Datum: 02.04.2024 übernommen von:

****

**Höhere Technische Bundeslehranstalt Hollabrunn**

**Höhere Lehranstalt für Elektronik und Technische Informatik**

**EIDESSTATTLICHE ERKLÄRUNG**

**Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.**

**Marcel Bieder**

**Maximilian Lendl**

**Ben Heinicke**

**Sebastian Hinterberger**

Hollabrunn, am 02.04.2024

**HINWEISE**

Die vorliegende Diplomarbeit wurde in Zusammenarbeit mit der Firma **Dronetech Austria** ausgeführt.

oder

Die vorliegende Diplomarbeit wurde für die Abteilung Elektronik und Technische Informatik der HTL Hollabrunn ausgeführt.

Die in dieser Diplomarbeit entwickelten Prototypen und Software-Produkte dürfen ganz oder auch in Teilen von Privatpersonen oder Firmen nur dann in Verkehr gebracht werden, wenn sie diese selbst geprüft und für den vorgesehenen Verwendungszweck für geeignet befunden haben.

Es wird keinerlei Haftung übernommen für irgendwelche Schäden, die aus der Nutzung der hier entwickelten oder beschriebenen Bestandteile des Projekts resultieren.

Für alle Entwicklungen gilt die GNU General Public License [http://www.gnu.org/licenses/gpl.html] der Free Software Foundation, Boston, USA in der Version 3.

Die Diplomarbeit erfüllt die “Standards für Ingenieur- und Technikerprojekte” entsprechend dem Rundschreiben Nr. 60 aus 1999 des BMBWK (GZ.17.600/101-II/2b/99).

[https://www.bmb.gv.at/ministerium/rs/1999\_60.html]

SCHLÜSSELBEGRIFFE

DANKSAGUNGEN

Reisinger, Wihsböck, Kauer, Stoll, Dronetech Austria (Daniel Stoiber),

**DIPLOMARBEIT**

**DOKUMENTATION**

|  |  |
| --- | --- |
| Namen der  Verfasser/innen |  |
| Jahrgang  Schuljahr |  |
| Thema der Diplomarbeit |  |
| Kooperationspartner |  |

|  |  |
| --- | --- |
| Aufgabenstellung |  |

|  |  |
| --- | --- |
| Realisierung |  |

|  |  |
| --- | --- |
| Ergebnisse |  |

|  |  |
| --- | --- |
| Typische Grafik, Foto etc.  (mit Erläuterung) |  |

|  |  |
| --- | --- |
| Teilnahme an Wettbewerben,  Auszeichnungen |  |

|  |  |
| --- | --- |
| Möglichkeiten der Einsichtnahme in die Arbeit | HTL Hollabrunn  Anton Ehrenfriedstraße 10  2020 Hollabrunn |

|  |  |  |
| --- | --- | --- |
| Approbation  (Datum / Unterschrift) | Prüfer/Prüferin | Direktor/Direktorin  Abteilungsvorstand/Abteilungsvorständin |

**DIPLOMA THESIS**

**Documentation**

|  |  |
| --- | --- |
| Author(s) |  |
| Form  Academic year |  |
| Topic |  |
| Co-operation partners |  |

|  |  |
| --- | --- |
| Assignment of tasks |  |

|  |  |
| --- | --- |
| Realisation |  |

|  |  |
| --- | --- |
| Results |  |

|  |  |
| --- | --- |
| Illustrative graph, photo  (incl. explanation) |  |

|  |  |
| --- | --- |
| Participation in competitions  Awards |  |

|  |  |
| --- | --- |
| Accessibility of  final project thesis | HTL Hollabrunn  Anton Ehrenfriedstraße 10  2020 Hollabrunn |

|  |  |  |
| --- | --- | --- |
| Approval  (Date / Signature) | Examiner/s | Head of Department / College |

DA Antrag und unterschriebene Erklärung aus der Diplomarbeitsdatenbank einfügen

Inhaltsverzeichnis

[1 Einleitung 16](#_Toc160296755)

[1.1 Projektziel 16](#_Toc160296756)

[1.2 Gesamtüberblick 16](#_Toc160296757)

[2 Grundlagen Drohnenflug 17](#_Toc160296758)

[2.1 Inertial Measurement Unit (IMU) 17](#_Toc160296759)

[2.1.1 Gyroskop 17](#_Toc160296760)

[2.1.2 Accelerometer 17](#_Toc160296761)

[2.1.3 Magnetometer 17](#_Toc160296762)

[2.1.4 Barometer 17](#_Toc160296763)

[2.2 Lagewinkel 18](#_Toc160296764)

[2.2.1 Throttle 18](#_Toc160296765)

[2.2.2 Pitch 18](#_Toc160296766)

[2.2.3 Roll 18](#_Toc160296767)

[2.2.4 Yaw 18](#_Toc160296768)

[2.2.5 Komplementärfilter 18](#_Toc160296769)

[2.3 PID-Regler 21](#_Toc160296770)

[2.3.1 Proportionalglied (P-Glied) 22](#_Toc160296771)

[2.3.2 Integralglied (I-Glied) 22](#_Toc160296772)

[2.3.3 Differenzialglied (D-Glied) 23](#_Toc160296773)

[2.3.4 PID-Regler 23](#_Toc160296774)

[3 Mechanischer Aufbau 24](#_Toc160296775)

[3.1.1 FPV – Drohne Gesamtaufbau 24](#_Toc160296776)

[3.1.2 3D – Modelle 24](#_Toc160296777)

[3.1.2.1 Propellerschutz 24](#_Toc160296778)

[3.1.2.2 Groundstation 24](#_Toc160296779)

[4 Elektronik FPV-Drohne (BIE) 25](#_Toc160296780)

[4.1 Allgemeines 25](#_Toc160296781)

[4.1.1 Grundlegendes Hardwarekonzept 25](#_Toc160296782)

[4.1.2 Anforderungen 25](#_Toc160296783)

[4.1.3 Komponentenauswahl 25](#_Toc160296784)

[4.2 Flight Controller 25](#_Toc160296785)

[4.2.1 Allgemeines 25](#_Toc160296786)

[4.2.2 Spannungsversorgungskonzept 25](#_Toc160296787)

[4.2.3 Mikrocontroller 25](#_Toc160296788)

[4.2.4 Altium PCB Design 25](#_Toc160296789)

[4.2.5 Pinbelegung 25](#_Toc160296790)

[4.3 Electronic Speed Controller (ESC) 25](#_Toc160296791)

[4.3.1 Übersicht 25](#_Toc160296792)

[4.3.2 Aufbau 25](#_Toc160296793)

[4.3.3 Funktionsweise 25](#_Toc160296794)

[4.3.4 Technische Daten 25](#_Toc160296795)

[4.4 Motoren 25](#_Toc160296796)

[4.4.1 Übersicht 25](#_Toc160296797)

[4.4.2 Aufbau 25](#_Toc160296798)

[4.4.3 Funktionsweise 25](#_Toc160296799)

[4.4.4 Technische Daten 25](#_Toc160296800)

[4.5 Receiver Fernsteuerung 25](#_Toc160296801)

[4.5.1 Übersicht 25](#_Toc160296802)

[4.5.2 Funktionsweise 25](#_Toc160296803)

[4.5.3 Technische Daten 25](#_Toc160296804)

[4.6 Video Transmitter (VTx) 25](#_Toc160296805)

[4.6.1 Übersicht 25](#_Toc160296806)

[4.6.2 Funktionsweise 25](#_Toc160296807)

[4.6.3 Technische Daten 25](#_Toc160296808)

[4.7 Live Kamera 25](#_Toc160296809)

[4.7.1 Übersicht 26](#_Toc160296810)

[4.7.2 Funktionsweise 26](#_Toc160296811)

[4.7.3 Technische Daten 26](#_Toc160296812)

[4.8 VR-Brille 26](#_Toc160296813)

[4.8.1 Übersicht 26](#_Toc160296814)

[4.8.2 Funktionsweise 26](#_Toc160296815)

[5 Steuerungssoftware 27](#_Toc160296816)

[5.1 Softwarearchitektur 27](#_Toc160296817)

[5.1.1 Hauptprogramm 27](#_Toc160296818)

[5.1.2 Real Time System Interrupt 28](#_Toc160296819)

[5.1.2.1 Motoransteuerung 29](#_Toc160296820)

[5.1.2.2 Terminal Übertragung und Status LEDs 30](#_Toc160296821)

[5.1.3 Umgang mit Initialisierungsfehler 31](#_Toc160296822)

[5.1.3.1 Übersicht Fehlercodes 32](#_Toc160296823)

[5.2 Fernsteuerung 34](#_Toc160296824)

[5.2.1 Konfiguration Fernsteuerung 36](#_Toc160296825)

[5.2.2 Unterstützte Protokolle 38](#_Toc160296826)

[5.2.2.1 PPM (Pulse Position Modulation) 38](#_Toc160296827)

[5.2.2.2 S.Bus 38](#_Toc160296828)

[5.2.2.3 I.Bus 40](#_Toc160296829)

[5.2.3 Initialisierung Empfangssoftware 42](#_Toc160296830)

[5.2.4 Empfangssoftware 46](#_Toc160296831)

[5.3 Inertial Measurement Unit (IMU) 54](#_Toc160296832)

[5.3.1 I²C Protokoll 54](#_Toc160296833)

[5.3.1.1 Schreibzyklus IMU 56](#_Toc160296834)

[5.3.1.2 Lesezyklus IMU 57](#_Toc160296835)

[5.3.2 IMU-Verbindungstest 59](#_Toc160296836)

[5.3.3 Gyroskop und Accelerometer - MPU9250 61](#_Toc160296837)

[5.3.3.1 Registerübersicht MPU9250 61](#_Toc160296838)

[5.3.3.2 Initialisierung Accelerometer und Gyroskop 62](#_Toc160296839)

[5.3.3.3 Einlesen der Accelerometer- und Gyroskop-Daten 63](#_Toc160296840)

[5.3.3.4 Komplementärfilter 65](#_Toc160296841)

[5.3.4 Barometer / Luftdrucksensor - BMP280 67](#_Toc160296842)

[5.3.4.1 Registerübersicht BMP280 67](#_Toc160296843)

[5.3.4.2 Initialisierung Barometer 68](#_Toc160296844)

[5.3.4.3 Einlesen der Barometer-Daten 69](#_Toc160296845)

[5.3.4.4 Berechnung der Höhe 71](#_Toc160296846)

[5.4 Smart Battery Monitor - DS2438 73](#_Toc160296847)

[5.4.1 One-Wire Protokoll 73](#_Toc160296848)

[5.4.1.1 One-Wire Schreibzyklen 74](#_Toc160296849)

[5.4.1.2 One-Wire Lesezyklen 77](#_Toc160296850)

[5.4.1.3 Initialisierungssequenz 79](#_Toc160296851)

[5.4.1.4 ROM-Funktionsbefehl 80](#_Toc160296852)

[5.4.1.5 Memory-Funktionsbefehl 80](#_Toc160296853)

[5.4.2 Registerübersicht DS2438 81](#_Toc160296854)

[5.4.3 Initialisierung DS2438 82](#_Toc160296855)

[5.4.4 Spannungsüberwachung 83](#_Toc160296856)

[5.5 Motoransteuerung 85](#_Toc160296857)

[5.5.1 DShot Protokoll 85](#_Toc160296858)

[5.5.2 Initialisierung Motoransteuerung 86](#_Toc160296859)

[5.5.3 Motoransteuerung Software 87](#_Toc160296860)

[5.6 PID-Regler 91](#_Toc160296861)

[5.6.1 Initialisierung PID-Regler 91](#_Toc160296862)

[5.6.2 PID-Algorithmus 93](#_Toc160296863)

[5.7 Debugging 95](#_Toc160296864)

[5.7.1 Terminal Ausgabe 95](#_Toc160296865)

[5.7.2 Status – LEDs 97](#_Toc160296866)

[6 Datenübertragung der Mess- und Videodaten (Ben) 99](#_Toc160296867)

[6.1 Überblick Datenübertragung 99](#_Toc160296868)

[6.2 Kommunikation: Flight Controller und Sender 99](#_Toc160296869)

[*6.2.1 Aufbau Sendermodul (VTx)* 99](#_Toc160296870)

[6.2.2 ASK – Modulation 99](#_Toc160296871)

[6.2.3 Transmit – Programm auf Cortex µC 99](#_Toc160296872)

[6.2.3.1 Gleitkommadarstellung (memcpy) 99](#_Toc160296873)

[6.2.3.2 Code 99](#_Toc160296874)

[*6.2.4* Testen der Übertragung 99](#_Toc160296875)

[6.3 Kommunikation: Sender und Empfänger 99](#_Toc160296876)

[6.3.1 Aufbau Empfängermodul 99](#_Toc160296877)

[6.3.2 Verbindungsaufbau 99](#_Toc160296878)

[6.3.3 Testen der Übertragung 99](#_Toc160296879)

[6.4 Kommunikation: Empfänger und Raspberry Pi 99](#_Toc160296880)

[6.4.1 Empfangsprogramm auf Raspberry Pi 99](#_Toc160296881)

[6.4.1.1 UART auf dem Raspberry Pi 99](#_Toc160296882)

[6.4.1.2 Code 99](#_Toc160296883)

[6.4.1 Senden der Daten auf dem MQTT-Server 99](#_Toc160296884)

[6.5 Testen der Datenübertragungskette 99](#_Toc160296885)

[7 Visualisierungs-App 100](#_Toc160296886)

[7.1 Applikation 100](#_Toc160296887)

[7.1.1 Einführung – Dart / Flutter - Framework 100](#_Toc160296888)

[7.1.1.1 Allgemeines 100](#_Toc160296889)

[7.1.1.2 Pub Dev / Package Installer 100](#_Toc160296890)

[7.1.1.3 Dart Syntax 100](#_Toc160296891)

[7.1.1.4 State Management 100](#_Toc160296892)

[7.1.1.4.1 Stateful vs. Stateless Widgets 100](#_Toc160296893)

[7.1.1.4.2 Provider 100](#_Toc160296894)

[7.1.2 Allgemeines zur App 100](#_Toc160296895)

[7.1.2.1 Usersystem 100](#_Toc160296896)

[7.1.2.2 Datenvisualiserung 100](#_Toc160296897)

[7.1.2.3 3D-Model-Viewer 100](#_Toc160296898)

[7.1.2.4 Livestream-Viewer 100](#_Toc160296899)

[7.1.3 UI-Konzept 100](#_Toc160296900)

[7.1.4 Projektstruktur und -umgebung 100](#_Toc160296901)

[7.1.4.1 Editor – Visual Studio Code 100](#_Toc160296902)

[7.1.4.2 Flutter Installation 100](#_Toc160296903)

[7.1.4.3 Projekterstellung 100](#_Toc160296904)

[7.1.4.4 Projektstruktur 100](#_Toc160296905)

[7.1.4.5 Packages 100](#_Toc160296906)

[7.1.4.5.1 Pubspec.yaml 100](#_Toc160296907)

[7.1.4.5.2 Installieren neuer Packages 100](#_Toc160296908)

[7.1.5 Splash + Willkommensscreen 101](#_Toc160296909)

[7.1.5.1 Native Splash Screen 101](#_Toc160296910)

[7.1.5.2 Willkommensbildschirm 101](#_Toc160296911)

[7.1.5.3 Logo-Design 101](#_Toc160296912)

[7.1.6 Login + Registrierung 101](#_Toc160296913)

[7.1.6.1 E-Mail-Login 101](#_Toc160296914)

[7.1.6.2 Google-Login 101](#_Toc160296915)

[7.1.6.3 Passwort vergessen 101](#_Toc160296916)

[7.1.6.4 Registrierung 101](#_Toc160296917)

[7.1.7 Homepage 101](#_Toc160296918)

[7.1.7.1 Bottom Navigation Bar / GNav-Bar 101](#_Toc160296919)

[7.1.7.2 Serverdatendialog 101](#_Toc160296920)

[7.1.7.3 Flugdatenvisualisierung 101](#_Toc160296921)

[7.1.7.3.1 MQTT-Datenstreams 101](#_Toc160296922)

[7.1.7.3.2 Syncfusion Cartesian Charts 101](#_Toc160296923)

[7.1.7.3.3 Echtzeitdaten + Animierte Diagramme 101](#_Toc160296924)

[7.1.7.3.4 Speicherung der Daten 101](#_Toc160296925)

[7.1.7.4 3D-Model-Viewer 101](#_Toc160296926)

[7.1.7.4.1 Flutter Cube 101](#_Toc160296927)

[7.1.7.4.2 MQTT-Datenstream 101](#_Toc160296928)

[7.1.7.4.3 3D-Dronenmodell 101](#_Toc160296929)

[7.1.7.4.3.1 Optimierungen in Blender 101](#_Toc160296930)

[7.1.7.4.4 Extrabedienungen 101](#_Toc160296931)

[Live-View 101](#_Toc160296932)

[7.1.7.4.5 VLC-Plugin 101](#_Toc160296933)

[7.1.7.4.6 Darstellung eines Netzwerkstreams 101](#_Toc160296934)

[7.1.7.4.7 Video-Overlay 101](#_Toc160296935)

[7.1.8 Sidemenu / Drawer 101](#_Toc160296936)

[7.1.8.1 Userprofil 101](#_Toc160296937)

[7.1.8.1.1 Darstellung der Userdaten 101](#_Toc160296938)

[7.1.8.1.2 Profilbildauswahl 101](#_Toc160296939)

[7.1.8.1.3 Änderung der Userdaten 102](#_Toc160296940)

[7.1.8.2 Credits 102](#_Toc160296941)

[7.1.8.3 Vorherige Flüge 102](#_Toc160296942)

[7.1.8.3.1 Darstellung aller gespeicherten Flüge 102](#_Toc160296943)

[7.1.8.3.2 Sortiermöglichkeiten 102](#_Toc160296944)

[7.1.8.3.3 Einsicht in vorherigen Flug 102](#_Toc160296945)

[7.1.8.4 Einstellungen 102](#_Toc160296946)

[7.1.8.4.1 Dark- / Light-Mode 102](#_Toc160296947)

[7.1.8.4.2 3D-Model Ausrichtung 102](#_Toc160296948)

[7.1.8.5 Logout´ 102](#_Toc160296949)

[7.2 Firebase Backend 102](#_Toc160296950)

[7.2.1 Installation via Firebase CLI 102](#_Toc160296951)

[7.2.1.1.1 Firebase CLI – Setup 102](#_Toc160296952)

[7.2.1.1.2 Firebase für Dart / Flutter aktivieren 102](#_Toc160296953)

[7.2.1.1.3 Flutter App mit Flutterfire konfigurieren 102](#_Toc160296954)

[7.2.2 Einbindung in Flutter 102](#_Toc160296955)

[7.2.2.1.1 Flutterfire Configure 102](#_Toc160296956)

[7.2.2.1.2 Benutzten Firebase Packages hinzufügen 102](#_Toc160296957)

[7.2.3 Nutzen der verschiedenen Datenbanksysteme 102](#_Toc160296958)

[7.2.3.1 Auth 102](#_Toc160296959)

[7.2.3.1.1 E-Mail + Passwort – Login 102](#_Toc160296960)

[7.2.3.1.2 Google Login 102](#_Toc160296961)

[7.2.3.2 Firestore Database 102](#_Toc160296962)

[7.2.3.2.1 Struktur 102](#_Toc160296963)

[7.2.3.2.2 Allgemeine Userdaten 102](#_Toc160296964)

[7.2.3.2.3 Flugdaten 102](#_Toc160296965)

[7.2.3.2.4 Einstellungen 102](#_Toc160296966)

[7.2.3.3 Realtime Database 102](#_Toc160296967)

[7.2.3.3.1 Allgemein 102](#_Toc160296968)

[7.2.3.3.2 Sinn der verschiedenen Flags 103](#_Toc160296969)

[7.2.3.4 Storage 103](#_Toc160296970)

[7.2.3.4.1 Allgemein 103](#_Toc160296971)

[7.2.3.4.2 Referenz zu User in Firestore 103](#_Toc160296972)

[8 Videostreaming 103](#_Toc160296973)

[8.1 Allgemeiner Aufbau 103](#_Toc160296974)

[8.1.1 Anforderungen 103](#_Toc160296975)

[8.1.2 Übertragungskette 103](#_Toc160296976)

[8.2 CADFPX Analog Kamera 103](#_Toc160296977)

[8.2.1 Allgemein 103](#_Toc160296978)

[8.2.2 Produktinformationen 103](#_Toc160296979)

[8.2.3 Verwendungszweck 103](#_Toc160296980)

[8.2.4 Verbindungstest via USB-Camera App 103](#_Toc160296981)

[8.3 USB2.0 VHS Video Grabber 103](#_Toc160296982)

[8.4 RTMP-Server via NGINX aufsetzen 103](#_Toc160296983)

[8.4.1 RTMP Allgemein 103](#_Toc160296984)

[8.4.2 NGINX 103](#_Toc160296985)

[8.4.2.1 Allgemein 103](#_Toc160296986)

[8.4.2.2 Installation 103](#_Toc160296987)

[8.4.2.3 RTMP-Konfiguration 103](#_Toc160296988)

[8.4.2.4 Serverstatus einsehen 103](#_Toc160296989)

[8.5 RTMP-Stream erstellen 103](#_Toc160296990)

[8.5.1 FFMPEG Allgemein 103](#_Toc160296991)

[8.5.2 Aufbau der Konvertierung 103](#_Toc160296992)

[8.5.2.1 Simple Konvertierung 103](#_Toc160296993)

[8.5.2.2 Optimierungen 103](#_Toc160296994)

[8.5.2.2.1 Encodervergleich 103](#_Toc160296995)

[8.5.2.2.2 Optimierungsflags 103](#_Toc160296996)

[8.5.2.2.3 Unoptimiert vs. Optimiert 104](#_Toc160296997)

[9 Videoserver-Port Forwarding 104](#_Toc160296998)

[9.1 Allgemein 104](#_Toc160296999)

[9.2 Routereinstellungen 104](#_Toc160297000)

[9.3 Kontakt mit Routerfirma zur Freischaltung 104](#_Toc160297001)

[9.4 Testen der Ports 104](#_Toc160297002)

[9.4.1 Portchecker.co 104](#_Toc160297003)

[9.5 Testen des Videostreams 104](#_Toc160297004)

[9.5.1 VLC-Player 104](#_Toc160297005)

[9.5.1.1 Devicestream direkt testen 104](#_Toc160297006)

[9.5.1.2 Netzwerkstream im selben Netzwerk 104](#_Toc160297007)

[9.5.1.3 Netzwerkstream via Public IP 104](#_Toc160297008)

[10 Ergebnisse 104](#_Toc160297009)

[11 Anhang 105](#_Toc160297010)

[11.1 Einführung CAD – Software (Fusion 360) (Ben) 105](#_Toc160297011)

[*11.1.1* UI und Projekterstellung 105](#_Toc160297012)

[*11.1.2* Skizze anfertigen 105](#_Toc160297013)

[*11.1.3* Körper erstellen 105](#_Toc160297014)

[*11.1.4* Schrift und Bilder einfügen 105](#_Toc160297015)

[11.2 3D – Druck (Ben) 105](#_Toc160297016)

[*11.2.1* 3D – Drucker 105](#_Toc160297017)

[*11.2.2* Filamente 105](#_Toc160297018)

[11.2.3 3D – Drucker Software (Ultimaker Cura) 105](#_Toc160297019)

[11.3 Inbetriebnahme Anleitung 105](#_Toc160297020)

[11.4 Projektplan 105](#_Toc160297021)

[11.5 Projektkosten 105](#_Toc160297022)

[11.6 Projekttagebuch 105](#_Toc160297023)

[11.7 Einführung Entwicklungsumgebung - Steuerungssoftware 106](#_Toc160297024)

[12 Quellen 112](#_Toc160297025)

[12.1 Gedruckte Medien 112](#_Toc160297026)

[12.2 Online 112](#_Toc160297027)

[13 Verzeichnis der Abbildungen 114](#_Toc160297028)

[14 Begleitprotokoll 116](#_Toc160297029)

[15 Anhang 117](#_Toc160297030)

[15.1 — Projektdokumentation (Kostendarstellung, Besprechungsprotokolle etc.) 117](#_Toc160297031)

[15.2 — Technische Dokumentation (technische Beschreibungen, Berechnungen, 117](#_Toc160297032)

[15.3 Konstruktionszeichnungen, Versuchsberichte, betriebswirtschaftliche Kalkulationen etc.) 117](#_Toc160297033)

[15.4 Schaltungen, Zeichnungssätze, sonstiges 117](#_Toc160297034)

# Einleitung

## Projektziel

## Gesamtüberblick

# Grundlagen Drohnenflug

## Inertial Measurement Unit (IMU)

### Gyroskop

### Accelerometer

### Magnetometer

### Barometer

## Lagewinkel

### Throttle

Throttle beschreibt die Geschwindigkeit aller vier Motoren. Ein anderer Begriff für Throttle ist der Motorschub. Bei der Angabe von zum Beispiel 80% Throttle ist gemeint, dass alle Motoren sich mit 80% Geschwindigkeit drehen. Wenn der Wert hoch ist, fliegt die Drohne nach oben. Bei einem niedrigen Wert fliegt die Drohne nach unten.

Die Motoren und Propellern müssen genug Schub erzeugen, damit das Gesamtgewicht der Drohne angehoben und manövriert werden kann. Die Throttleangabe ist vor allem wichtig bei einem Hover-Modus. Dabei regelt der Quadrokopter die einzelnen Geschwindigkeiten der Motoren, um sich selber aufrecht zu halten.

### Pitch

Bei Pitch handelt es sich um den Neigungswinkel nach vorne und hinten, wenn die Drohne von hinten angeschaut wird. Mit der Angabe vom Pitch kann der Quadrokopter nach vorne und nach hinten gelenkt werden.

Bild Drohne mit Pitch winkel

Vielleicht mehr physikalisch erklären

### Roll

Bei Roll handelt es sich um den Neigungswinkel nach rechts und links, wenn die Drohne von hinten angeschaut wird. Mit der Angabe vom Roll kann der Quadrokopter nach rechts und links gesteuert werden.

Bild Drohne mit Roll winkel

Vielleicht mehr physikalisch erklären

### Yaw

Bei Yaw handelt es sich um die Drehung um die eigene Achse nach links oder rechts. Mit der Angabe vom Yaw kann der Quadrokopter nach links und rechts drehen.

Bild Drohne mit Yaw winkel

Vielleicht mehr physikalisch erklären

### Komplementärfilter

Der Komplementärfilter hat die Aufgabe, die Messwerte von einen Beschleunigungssensor und Gyroskop in Lagewinkel umzuwandeln.

**Accelerometer Winkel**:

Durch die Kombination der terrestrischen Gravitation und Trigonometrie ermöglicht es die Accelerometer-Werte in Neigungswinkel umzuwandeln:

accelerometerPitch/Roll … jeweiliger berechneter Winkel in Radiant (rad)

accelerometerX/Y/Z … Accelerometer Messwert der jeweiligen Achse in g (Erdbeschleunigung, ungefähr 9,81m/s²)

Der Yaw-Winkel kann nicht berechnet werden, da die Yaw-Bewegung in der Ebene, um die Z-Achse, stattfindet und daher unabhängig von dem Werten der X- und Y-Achse ist.

Im Gegensatz zum Gyroskop besitzt ein Accelerometer keinen Drift in der Messung und kann daher für längerfristige Messungen verwendet werden.

Für den Filter wird der berechnete Winkel in Grad statt in Radiant benötigt. Daher müssen die Ergebnisse mit einem Faktor von multipliziert werden.

Problem:

Wenn der Accelerometer bewegt wird, kann dieser nicht zwischen den Bewegungsbeschleunigungen und der Erdbeschleunigung unterscheiden und liefert daher verfälschte Werte.

**Gyroskop Winkel**:

Durch die Integration der Winkelbeschleunigungswerte ist es möglich, Lagewinkel zu bestimmen:

gyroscopePitch/Roll/Yaw … jeweiliger berechneter Winkel in Radianten (rad)

gyroscopeX/Y/Z … Gyroskop Messwert der jeweiligen Achse in Grad pro Sekunde (°/s)

dt … Zeitbereich in Sekunden (s)

Problem:

Durch die Integration werden Messungenauigkeit aufsummiert, die zu einen Wertedrift führen. Daher kann das Gyroskop nur für kurzfristige Messungen verwendet werden.

**Sensorfusion**:

Der Komplementärfilter kann als Kombination von zwei Filter gesehen werden: Ein Hochpassfilter für das Gyroskop und einen Tiefpassfilter für den Accelerometer.

Das Accelerometer liefert eine gute Indikation der Orientierung bei konstanten Bedingungen, und das Gyroskop liefert eine gute Indikation bei schnellen Neigungsänderungen.

Winkel … Pitch/Roll-Winkel in Grad (°)

α … Filterkoeffizient

gyroscopeData … Gyroskop Messwerte der jeweiligen Achse in Grad pro Sekunde (°/s)

dt … Abtastzeit in Sekunden (s)

accelerometerWinkel … berechneter Roll/Pitch nur mit Accelerometerdaten in Grad (°)

Der Wert für α ist typischerweise 0,98. Das bedeutet, dass die Gyroskop-Messung zu 98% und die Accelerometer-Messung zu 2% gewichtet wird. Daraus folgt:

Dadurch, dass mit dem Accelerometer kein Yaw-Winkel bestimmt werden kann, wird der Winkel nur mit dem Gyroskop berechnet. Wegen dem Gyroskop-Drift wird dieser Winkel über die Zeit immer ungenauer und muss für einen genauen Wert nach einer gewissen Zeit zurückgesetzt werden.

## PID-Regler

Im Allgemeinen wird zwischen einer Steuerung und einer Regelung unterschieden. Bei einer Steuerung wird der Sollwert direkt in einen Steuerungsalgorithmus geleitet, der immer nach demselben Schema abläuft. Bei einer Regelung wird der Istwert des Systems rückgekoppelt und mit dem Sollwert verglichen. Dieser Vergleich ergibt eine Regelabweichung, die das in den Regelalgorithmus geleitet wird. Das Ziel einer Regelung ist, die Regelabweichung zum Verschwinden zu bringen – Soll- und Istwert auf denselben Wert regeln.

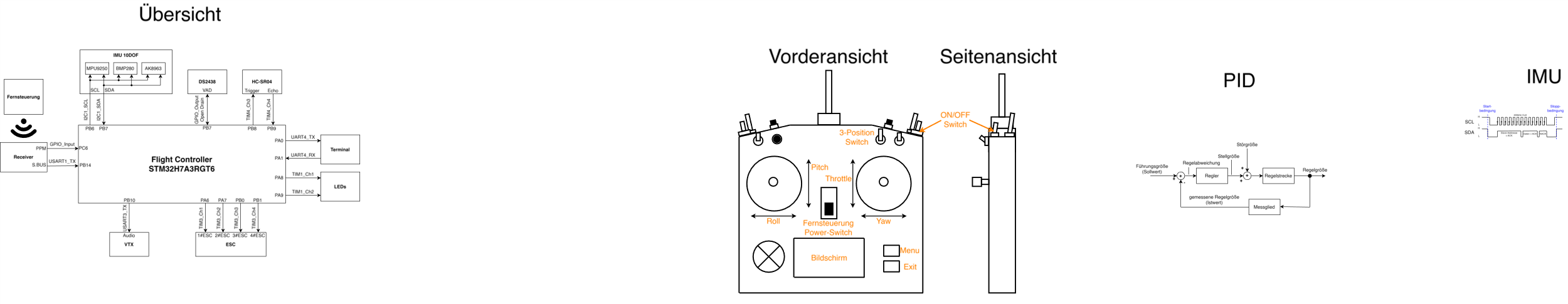


Abbildung 1: Regler-Blockschaltbild

Führungsgröße … eingestellter Sollwert, der vom Regler erreicht werden soll

Regelabweichung … Differenz zwischen Führungsgröße und gemessener Regelgröße

Regler … erstellt Stellgröße proportional von der Regelabweichung

Stellgröße … Reglerausgang

Störgröße … externe Einflüsse, die die Stellgröße beeinflussen

Regelstrecke … wandelt Stellgröße in die Regelgröße um

Regelgröße … gemessener Istwert, meistens durch Sensoren bestimmt

Es gibt verschiedene Arten für Regler, die für unterschiedliche Anwendungen Vor- und Nachteile liefern. Für den Quadrokopter wurde ein PID-Regler verwendet. Dieser hält während des Fluges die Drohne stabil. Bei Störgrößen, wie zum Beispiel Windstößen, soll die Drohne sich automatisch wieder in die richtige Lage ausrichten, um weiter einen stabilen Flug zu gewährleisten.

Ein PID-Regler besteht aus drei verschiedenen Gliedern (Proportional-, Integral-, Differenzialglied), die unterschiedliche Aufgaben erfüllen und Funktionsweisen vorweisen:

### Proportionalglied (P-Glied)

Das Proportionalglied setzt das Ausgangssignal proportional zum Eingangssignal.

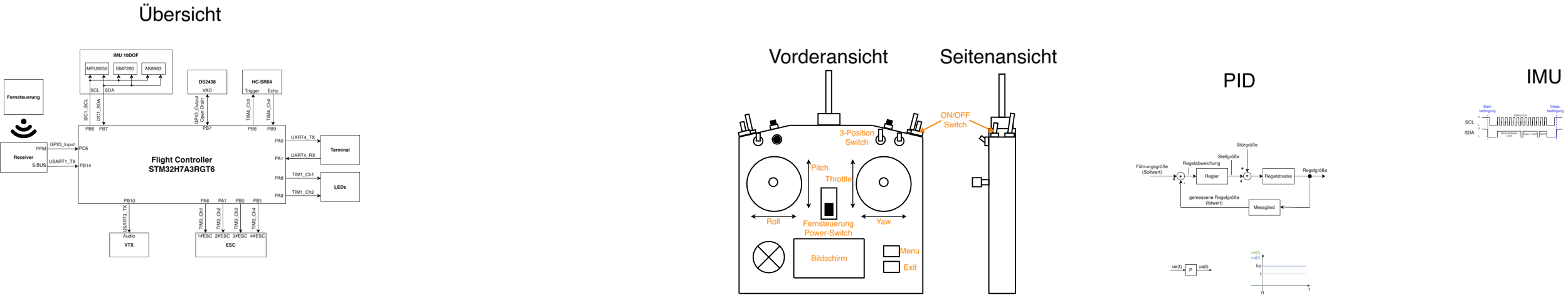


Abbildung 2: P-Glied Sprungantwort

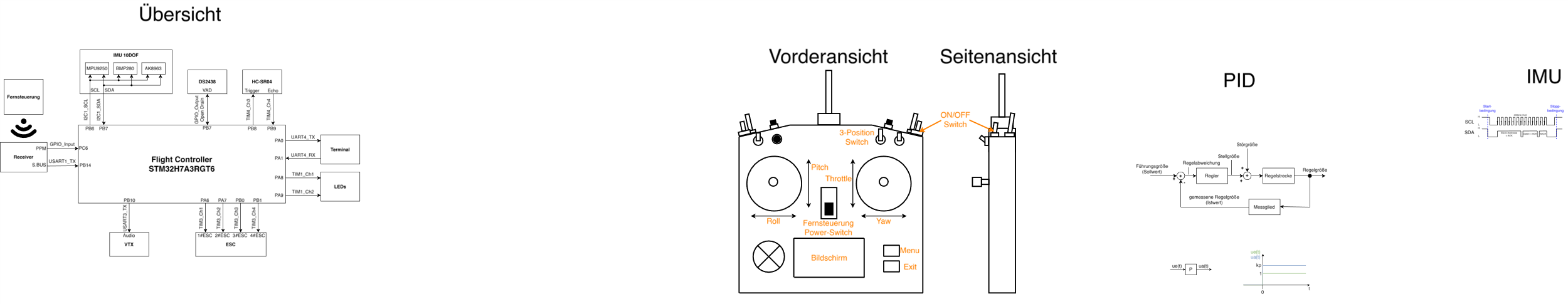


Abbildung 3: P-Glied Schaltsymbol

**Vor- (+) und Nachteile (-):**

+ kann Regelabweichung zum Verschwinden bringen

+ schnelles Anregeln

+ keine Phasenverschiebung

- kann Regelabweichung = 0 nicht halten

### Integralglied (I-Glied)

Das Integralglied setzt das Ausgangssignal zum Integral vom Eingangssignal.

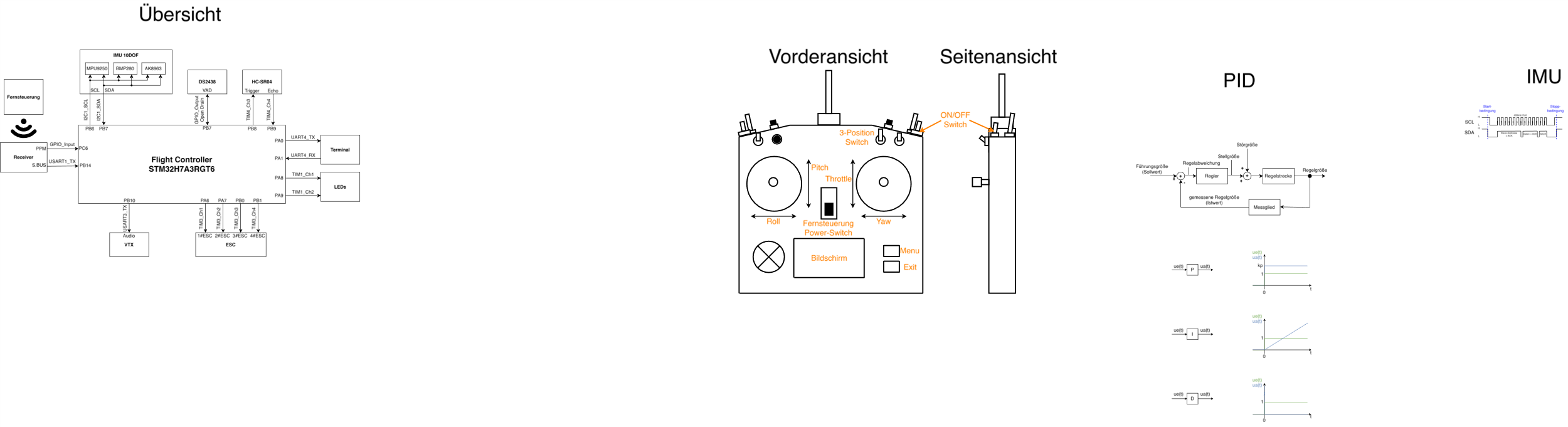


Abbildung 4: I-Glied Schaltsymbol

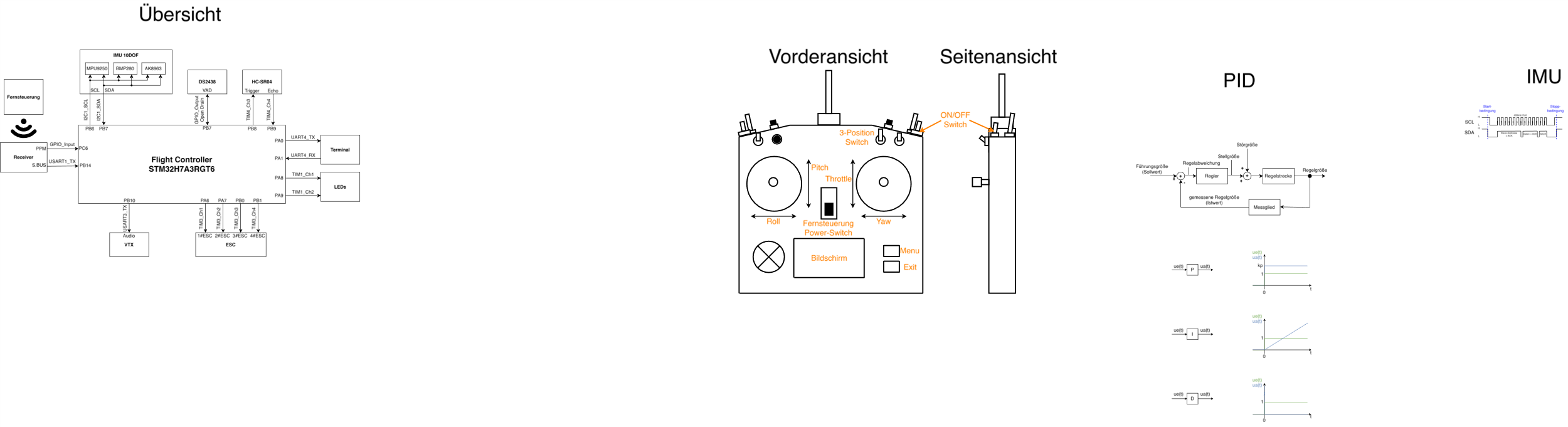


Abbildung 5: I-Glied Sprungantwort

**Vor- (+) und Nachteile (-):**

+ kann Regelabweichung zum Verschwinden bringen

- langsames Anregeln

- verringert die Stabilität (φ = -90°)

- neigt zum Überschwingen

### Differenzialglied (D-Glied)

Das Differenzialglied setzt das Ausgangssignal zur momentanen Änderung des Eingangssignals.

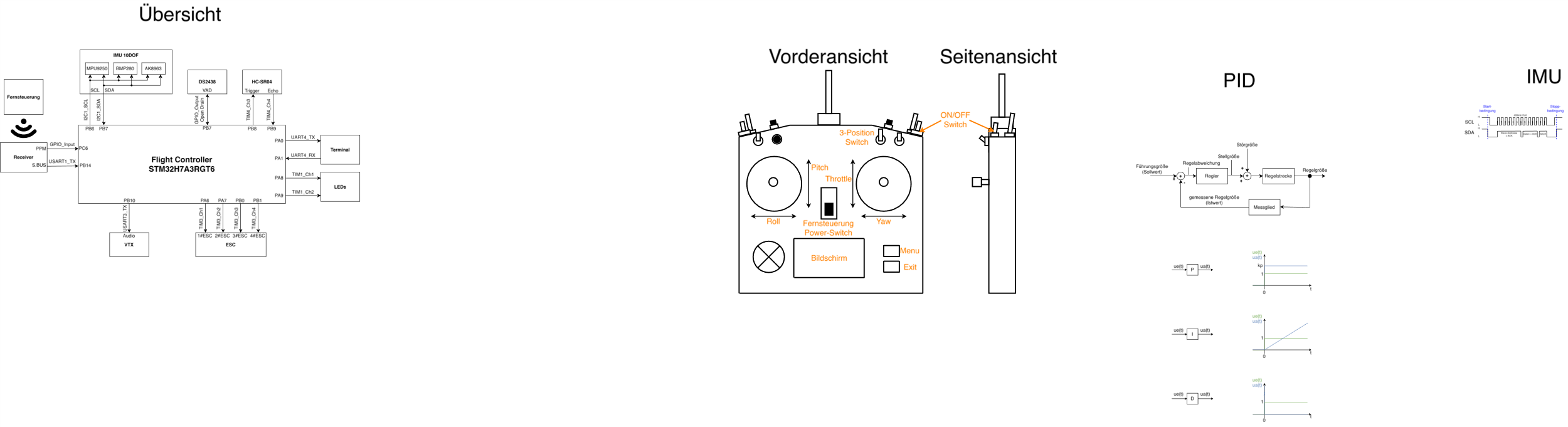


Abbildung 6: D-Glied Schaltsymbol

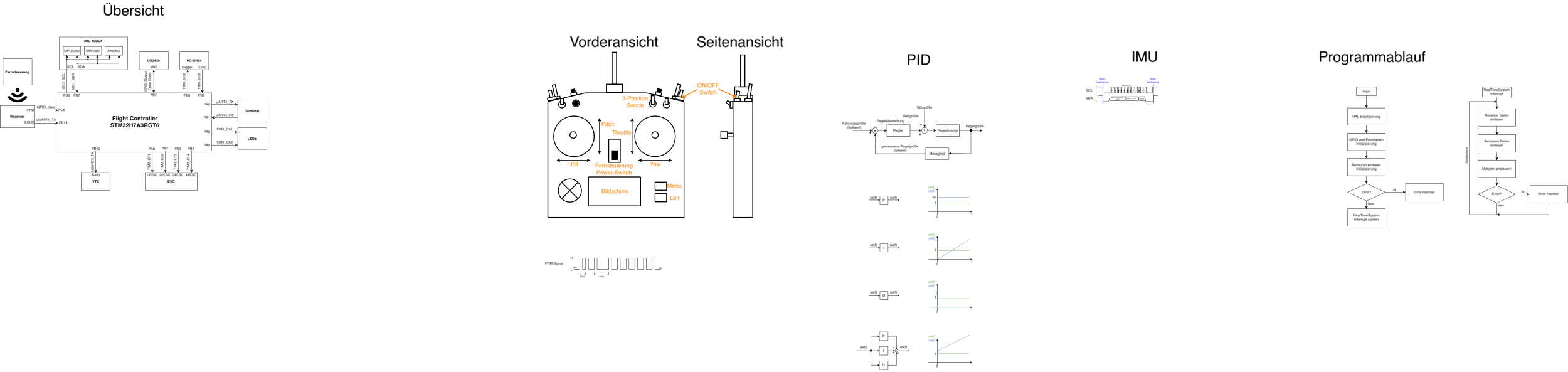


Abbildung 7: D-Glied Sprungantwort

**Vor- (+) und Nachteile (-):**

- kann allein nicht Regeln

In Kombination mit anderen Gliedern:

+ schnelles Anregeln

+ verbessert die Stabilität (φ = +90°)

### PID-Regler

Der PID-Regler kombiniert alle Ausgangssignale der einzelnen Glieder.

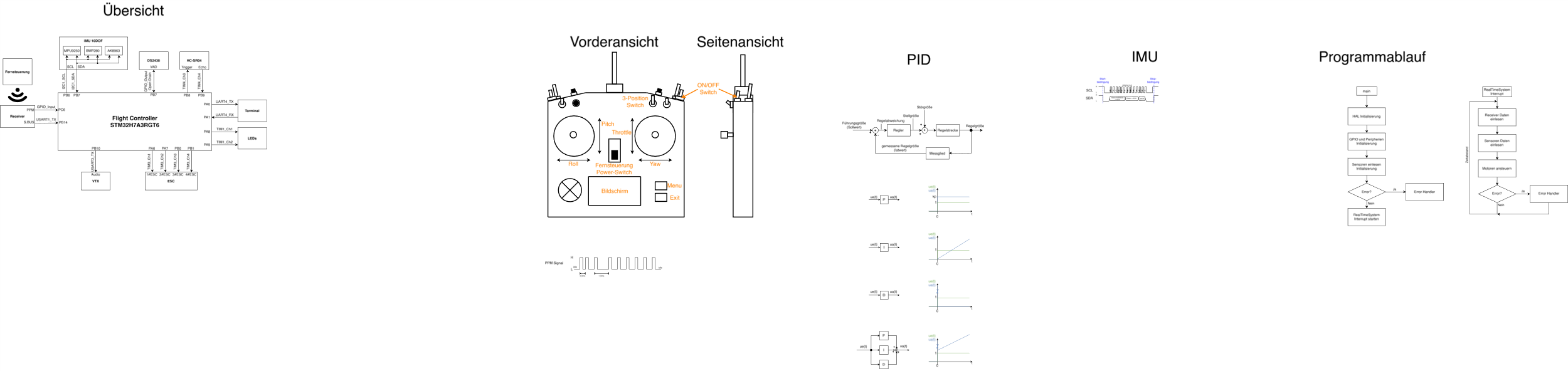


Abbildung 8: PID-Regler Schaltsymbol



Abbildung 9: PID-Regler Sprungantwort

**Vor- (+) und Nachteile (-):**

+ schnelles Anregeln (durch P- und D-Glied)

+ kann Regelabweichung zum Verschwinden bringen (P und I) und halten (I)

+ verbessert die Stabilität (D)

- kann Überschwingen (I)

# Mechanischer Aufbau

### FPV – Drohne Gesamtaufbau

### 3D – Modelle

#### Propellerschutz

#### Groundstation

# Elektronik FPV-Drohne (BIE)

## Allgemeines

### Grundlegendes Hardwarekonzept

### Anforderungen

### Komponentenauswahl

## Flight Controller

### Allgemeines

### Spannungsversorgungskonzept

### Mikrocontroller

### Altium PCB Design

### Pinbelegung

## Electronic Speed Controller (ESC)

### Übersicht

### Aufbau

### Funktionsweise

### Technische Daten

## Motoren

### Übersicht

### Aufbau

### Funktionsweise

### Technische Daten

## Receiver Fernsteuerung

### Übersicht

### Funktionsweise

### Technische Daten

## Video Transmitter (VTx)

### Übersicht

### Funktionsweise

### Technische Daten

## Live Kamera

### Übersicht

### Funktionsweise

### Technische Daten

## VR-Brille

### Übersicht

### Funktionsweise

# Steuerungssoftware

## Softwarearchitektur

Die Software des Flight Controllers hat die Aufgabe, die Daten der Fernsteuerung über einen Receiver und die Messwerte der Sensoren einzulesen und diese umzuwandeln, damit diese an die ESC, VTX, Status-LEDs und Terminal senden zu können.

Die Architektur ist sehr zeitkritisch und benötigt genaue Interrupts, damit ein stabiler und sicherer Flug möglich ist. Daher wird das gesamte System von dem Real Time System Interrupt gesteuert, welcher in einem Zeitbereich von ein paar Millisekunden auslöst und mithilfe der Sensorik die gesamte FPV-Drohne steuert.

### Hauptprogramm

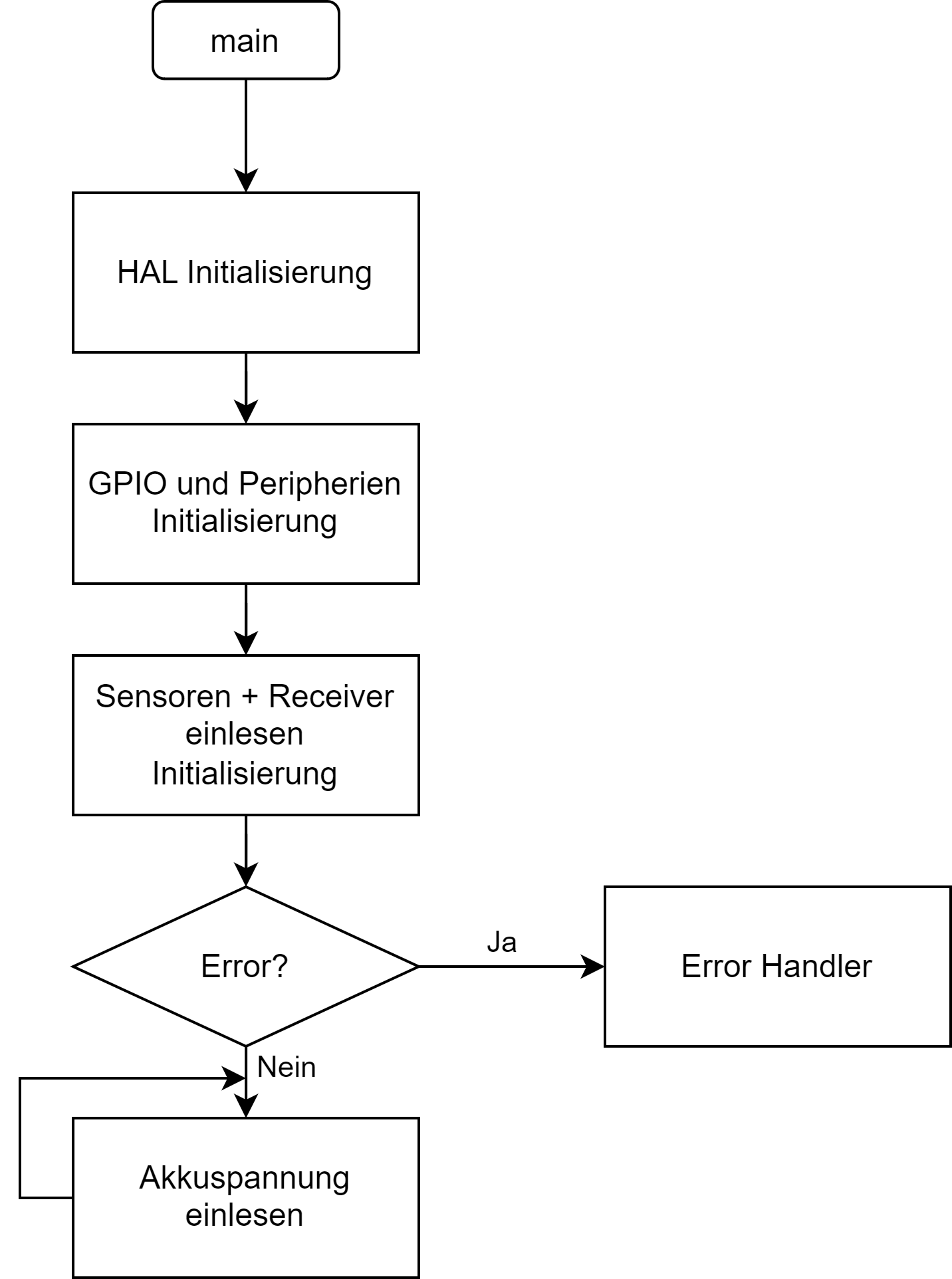


Abbildung 10: Flussdiagramm Programmablauf

Am Beginn des Hauptprogramms (main) wird die gesamte HAL-Struktur initialisiert. Danach werden alle GPIO-Ports und Peripherien mit den Angaben aus STM32CubeMX eingestellt. Weiters werden alle Sensoren mit den gewünschten Einstellungen initialisiert und auf Fehler überprüft [*(siehe: 5.1.3 Umgang mit Initialisierungsfehler)*](#_Umgang_mit_Initialisierungsfehler). Als letzten Schritt geht das Hauptprogramm in eine Endlosschleife über, die durchgehen Akkuspannungswerte einliest.

### Real Time System Interrupt

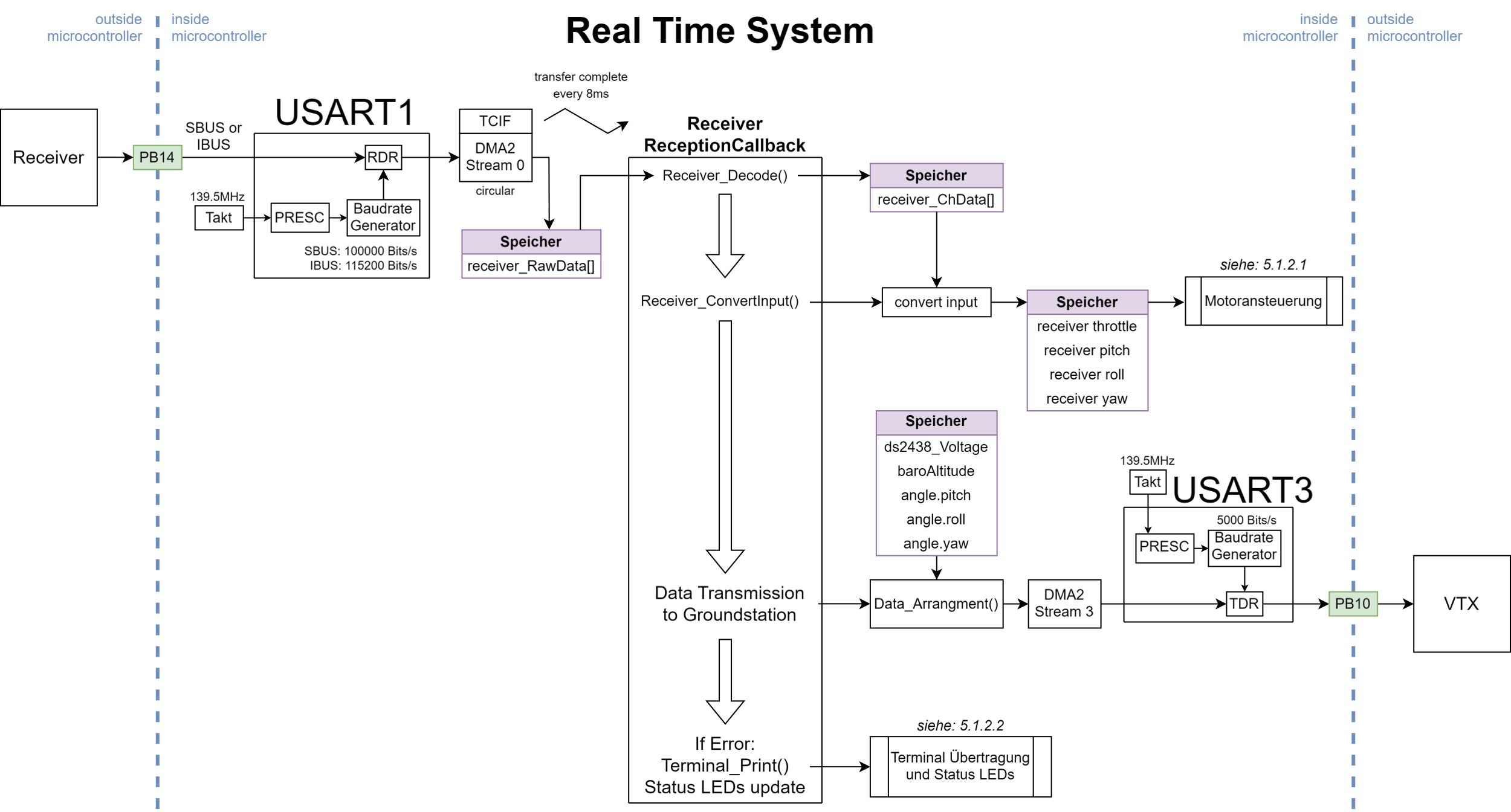


Abbildung 11: Real Time System Interrupt Architektur

#### Motoransteuerung

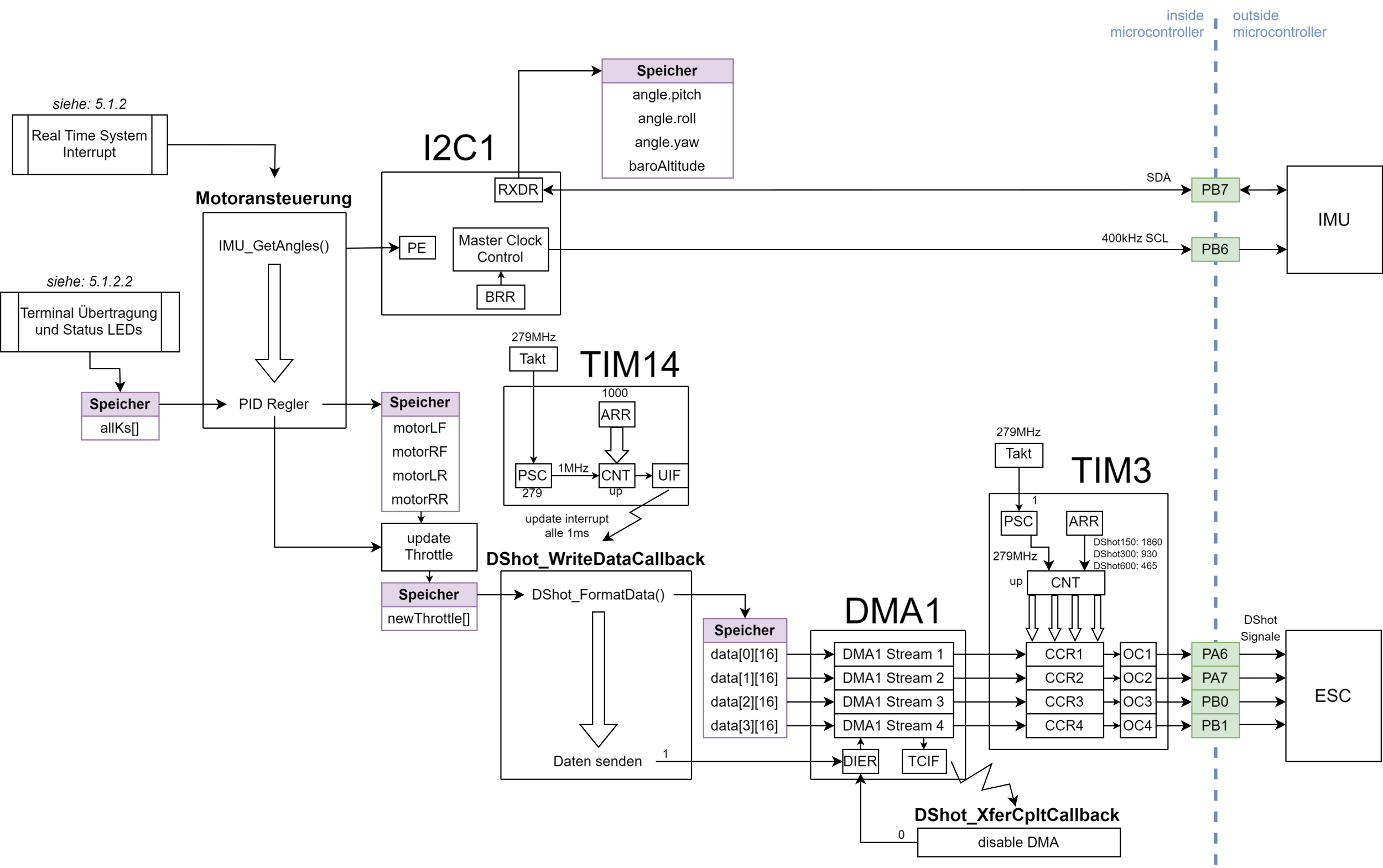


Abbildung 12: Motoransteuerung Architektur

#### Terminal Übertragung und Status LEDs

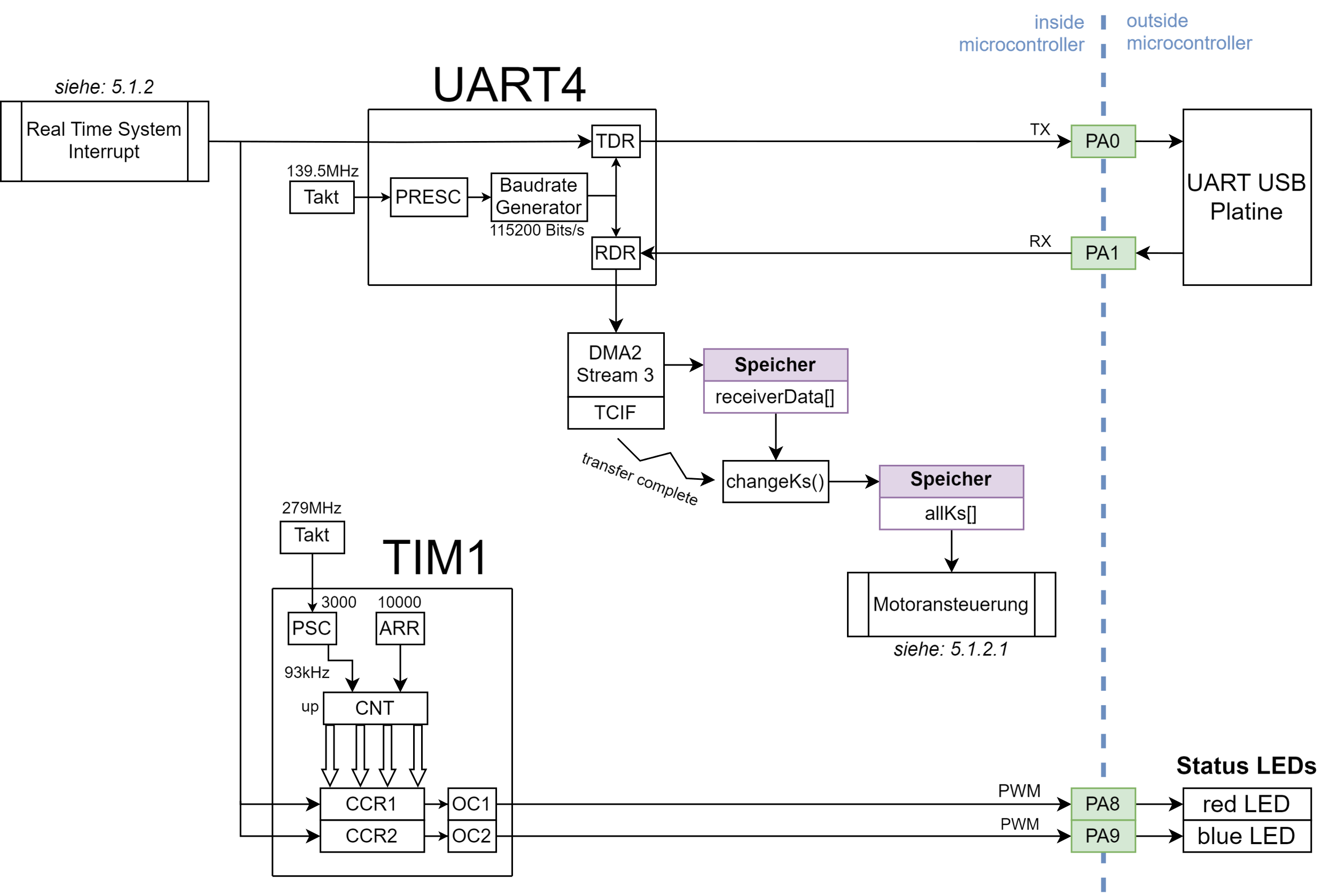


Abbildung 13: Terminal Übertragung und Status LEDs Architektur

### Umgang mit Initialisierungsfehler

Wenn während des Initialisierungsprozesses ein Error auftritt, wird die folgende Funktion aufgerufen:

Dateiname: status\_handling.c

 /\*\*

  \* @brief This function completely stops the program

  \* @param sens what sensor has the error

  \* @param errorCode

  \* @retval None

  \*/

void Sensor\_ErrorHandler(Sensors sens, int8\_t errorCode)

{

  char txt[100];

  // choose error source

  switch(sens)

  {

    case DATA\_TRANSMIT:

      sprintf(txt, "DATA TRANSMIT ERROR | Code: %d\n\r", errorCode);

      break;

    case DS2438:

      sprintf(txt, "DS2438 ERROR | Code: %d\n\r", errorCode);

      break;

    case IMU:

      sprintf(txt, "IMU ERROR | Code: %d\n\r", errorCode);

      break;

    case RECEIVER:

      sprintf(txt, "RECEIVER ERROR | Code: %d\n\r", errorCode);

      break;

    case DSHOT:

      sprintf(txt, "DSHOT ERROR | Code: %d\n\r", errorCode);

      break;

    case PID:

      sprintf(txt, "PID ERROR | Code: %d\n\r", errorCode);

      break;

    default:

      sprintf(txt, "wrong sensor ERROR | Code: %d\n\r", errorCode);

      break;

  }

  // output error message

  Terminal\_Print(txt);

  // turn red LED on and the blue LED off

  \_\_HAL\_TIM\_SET\_COMPARE(LED\_TIM, LED\_RED\_CHANNEL, 10000);

  \_\_HAL\_TIM\_SET\_COMPARE(LED\_TIM, LED\_BLUE\_CHANNEL, 0);

  // disable all interrupts

  \_\_disable\_irq();

  // infinite loop

  while(1);

}

Diese Funktion gibt über das Terminal die Errorquelle mit dem Errorcode aus.

Beispiel: RECEIVER Error | Code 13

Für eine Fehlercodeerklärung – *siehe 5.1.3.1 Übersicht Fehlercodes*

Weiters wird die rote LED durchgehend eingeschalten, und die blaue LED ausgeschalten. Alle Interrupts werden deaktiviert, und das Programm wird durchgehend mit einer Endlosschleife pausiert. Um den Initialisierungsprozess neu zu starten, muss der Mikrocontroller zurückgesetzt werden.

#### Übersicht Fehlercodes

Für jede Verbindung gibt es eigene Fehlercodes, die sich in enum-Objekte befinden. Diese werden in den Header-Dateien der einzelnen Komponente als Datentyp mit den Namen *<Komponente>\_Status* definiert. Zusätzlich hat jedes Objekt einen Wert, der angibt, dass kein Fehler aufgetreten ist: diese sind mit *<Komponente>\_OK* definiert.

**IMU-Fehlercodes:**

Dateiname: IMU\_10DOF.h

typedef enum IMU\_Status

{

    IMU\_OK = 0,

    IMU\_ADDRESS\_ERROR = 1,      // wrong I2C slave address

    IMU\_I2C\_ERROR = 2,          // no I2C typedef set

    IMU\_TIM\_ERROR = 3,          // no TIM typedef set

    IMU\_MPU\_WHOAMI\_ERROR = 10,  // MPU9250 who am i value wrong

    IMU\_MAG\_WHOAMI\_ERROR = 11,  // AK8963 who am i value wrong

    IMU\_BARO\_CHIPID\_ERROR = 12, // BMP280 who am i value wrong

    IMU\_BARO\_INIT\_ERROR = 13    // BMP280 init timeout

} IMU\_Status;

**DS2438 Fehlercodes:**

Dateiname: DS2438.h

typedef enum DS2438\_Status

{

    DS2438\_OK = 0,

    DS2438\_ERROR = 1,           // sensor not found or initialisation error

    DS2438\_VOLTAGE\_ERROR = 2    // battery voltage too low

} DS2438\_Status;

**Receiver Fehlercodes:**

Dateiname: receiver.h

typedef enum Receiver\_Status

{

    RECEIVER\_OK = 0,                // receiver ok

    RECEIVER\_UART\_ERROR = 1,        // uart configuration doesnt match selected protocol

    RECEIVER\_PWM\_ERROR = 2,         // pwm timer not set

    RECEIVER\_PPM\_ERROR = 3,         // IBUS selected: PPM not configured correctly

    RECEIVER\_TIMEOUT = 4,           // no data signal found

    PROTOCOL\_ERROR = 5,             // selected protocol wrong

    IBUS\_ERROR = 6,                 // IBUS UART DMA not starting

    IBUS\_HEADER\_ERROR = 7,          // IBUS header is wrong

    IBUS\_CHECKSUM\_ERROR = 8,        // IBUS checksum is wrong

    IBUS\_SIGNAL\_LOST\_ERROR = 9,     // IBUS signal lost

    SBUS\_ERROR = 10,                // SBUS UART DMA not starting

    SBUS\_HEADER\_ERROR = 11,         // SBUS header is wrong

    SBUS\_FOOTER\_ERROR = 12,         // SBUS footer is wrong

    SBUS\_SIGNAL\_LOST = 13,          // SBUS signal lost flag is set

    SBUS\_SIGNAL\_FAILSAFE = 14       // SBUS signal failsafe flag is set

} Receiver\_Status;

**DShot Fehlercodes:**

Dateiname: dshot\_own.h

typedef enum DShot\_Status

{

    DSHOT\_OK = 0,

    DSHOT\_TIM\_ERROR = 100 // no TIM typedef set

} DShot\_Status;

## Fernsteuerung

Um eine FPV-Drohne steuern zu können, muss mindestens ein 4-Kanal Sender und Empfänger verwendet werden, um die gewünschten Steuerungsdaten, Throttle, Pitch, Roll und Yaw, zu senden. Mit weiteren Kanälen können zusätzliche Funktionen, wie zum Beispiel ON/OFF-Switch, realisiert werden.

Für das Projekt wurde der Turnigy 9X 9Ch Mode 2 Transmitter mit dem TGY-iA6C Receiver ausgewählt. Diese Kombination bietet eine große Auswahl von programmierbaren Schaltern, deren Stellungen mittels eines schnellen, digitalen seriellen Protokolls übertragen wird.



Abbildung 14: Fernsteuerung



Abbildung 15: Receiver

In der Abbildung 3 wird die verwendete Tastenbelegung auf der Fernsteuerung dargestellt:

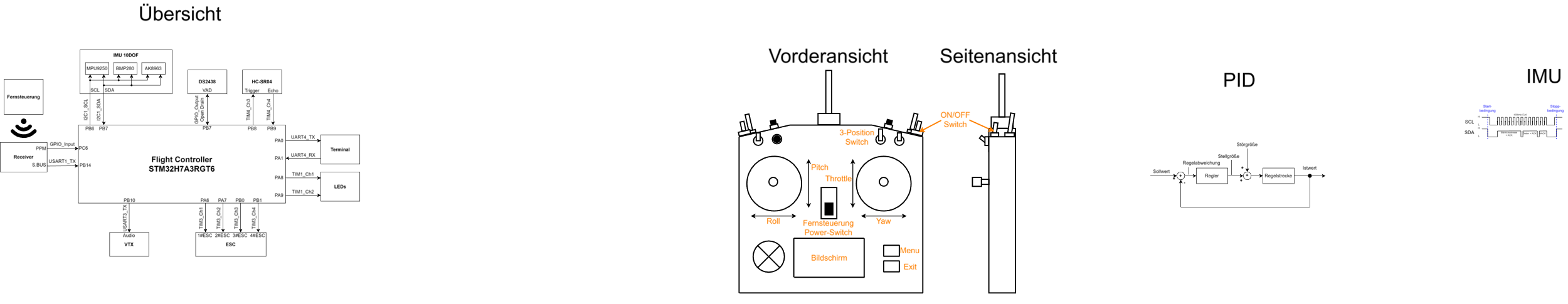


Abbildung 16: Fernsteuerung Tastenbelegung

Die zwei großen Joysticks reichen aus, um die Drohne steuern zu können. Für eine bessere Bedienung werden zwei zusätzlich Schalter verwendet.

Der ON/OFF-Switch ermöglicht das Ein- und Ausschalten der Fernsteuerunsbefehlen.

Mit dem 3-Position-Switch kann zwischen mehreren Flugmodi unterschieden werden.

Um die richtige Funktionalität der Schalter zur gewährleisten - [*siehe: 5.2.1 Konfiguration Fernsteuerung*](#_Konfiguration_Fernsteuerung)

Wenn der ON/OFF-Switch nach unten zeigt, schalten sich die Motoren aus und alle Fernsteuerungsbefehle werden übernommen. Wenn der Schalter nach oben zeigt, wirkt der 3-Position-Switch als Modus Auswahl.

In der oberen Stellung befindet sich die Drohne im „Safe-Mode“. Das bedeutet, dass der maximale Throttle-Wert, der zur ESC geschickt werden kann, begrenzt wird.

In der mittleren Stellung schaltet wird der „Normal-Mode“ eingeschaltet. Dieser Modus verhält sich gleich, wie der „Safe-Mode“, nur erlaubt er, höhere Motordrehzahl.

Die untere Position stellt einen Hover-Mode ein. Die Drohne versucht jetzt, sich selbst gerade in der Ebene mit Sensoren- und Throttlewerten zu halten.

Wichtig: Wenn die Versorgungsspannung der Fernsteuerung weniger als 8,5V beträgt, beginnt ein Buzzer alle 5 Sekunden den Benutzer zu alarmieren. Die aktuelle Spannung kann auf dem Bildschirm der Fernsteuerung überprüft werden.

**Switch Error:**

Beim Starten der Fernsteuerung kann es vorkommen, dass am Bildschirm „switch-error!“ angezeigt wird. Dieser Fehler wird angegeben, wenn die Schalter nicht auf High-Position sind.

Das bedeutet: Beim Starten der Fernsteuerung müssen alle Schalter nach oben zeigen beziehungsweise die Schalter, die auf der Oberseite montiert worden sind, müssen nach hinten zeigen, wie in Abbildung 3 dargestellt ist.

### Konfiguration Fernsteuerung

Damit die ausgewählte Fernsteuerung mit unserem System funktioniert, müssen grundlegende Vorbereitungen vor der Inbetriebnahme getroffen und kontrolliert werden.

**Verbindung mit Empfänger:**

Um die Verbindung zu testen, schaltet man die Fernsteuerung und den Receiver ein. Die eingebaute Status-LED des Receivers fängt schnell zum Blinken an. Wenn die LED nach ein paar Sekunden anfängt durchgehend zu leuchten, ist die Fernsteuerung mit dem Receiver verbunden.

Wenn die LED anfängt langsamer zu blinken, wurde die Verbindung nicht gefunden und die Fernsteuerung muss mit dem Receiver neu gekoppelt werden:

1. Schalte die Fernsteuerung und den Receiver aus
2. Halte den Bind-Knopf auf der Rückseite der Fernsteuerung, während diese angeschaltet wird
3. Schalte den Receiver ein
4. Warte, bis die Status-LED auf dem Receiver durchgehend leuchtet
5. Lasse den Bind-Knopf los und starte die Fernsteuerung neu, um die Koppelung zu testen

**Einstellungen in Fernsteuerung:**

Damit die Signale richtig von der Flugsoftware interpretiert werden können, müssen folgende Einstellungen in der Fernsteuerung getroffen werden:

Beim längeren gedrückt halten der MENU-Taste wird eine Auswahl zwischen „System Setting“ und „Function Setting“ angezeigt.

Die folgenden Tabellen zeigen die Einstellung, die für die richtige Funktionsweise notwendig sind:

|  |  |
| --- | --- |
| **System Setting** | |
| Type Select | ACRO |
| Modeuat | PPM |
| Stick Set | MODEL 1 |
| Output Select | PPM s-BUS |

|  |  |  |
| --- | --- | --- |
| **Function Setting** | | |
| AUX-CH  (für ON/OFF Switch) | CH5 Gear | |
| PROG. MIX  (für 3-Position-Switch) | MIX1 | STATE: ACT  MASTER: GYR  SLAVE: FLP  OFFSET: 000  UPRATE: -100  DNRATE: 100  SW: NOR |
| MIX2 | STATE: ACT  MASTER: GYR  SLAVE: FLP  OFFSET: 000  UPRATE: 000  DNRATE: 000  SW: ID1 |
| MIX3 | STATE: ACT  MASTER: GYR  SLAVE: FLP  OFFSET: 000  UPRATE: 100  DNRATE: -100  SW: ID2 |
| DISPLAY | Kontrolle der Kanal-Ausgänge | |

Diese Einstellungen sind notwendig, damit der ON/OFF Switch und der 3-Position-Switch richtig dekodiert werden, und eine richtige Funktionalität nachweisen können.

Zur Kontrolle werden in Balkendiagrammen die einzelnen Kanalwerte unter DISPLAY angezeigt.

### Unterstützte Protokolle

Der Receiver unterstützt eine gleichzeitige Signalausgabe von einem analogen und digitalen Signal. Der Pin PPM gibt immer ein PPM-Signal aus, während der zweite Pin S.BUS entweder ein S.Bus oder I.Bus Signal ausgibt. Die Ausgabe kann in den Fernsteuerungseinstellungen festgelegt werden *(*[*siehe: 5.2.1 Konfiguration*](#_Konfiguration)*)*.

#### PPM (Pulse Position Modulation)

Das PPM-Signal ist das einzige analoge Signal, das der Receiver ausgibt. Die Daten können mit der Timer-Peripherie eingelesen werden, indem die Länge der Periodendauer bestimmt wird. Die Daten kommen in 9 Kanälen mit einem 8,1ms low-aktiven Ruhezustand. Die Kanalwerte befinden sich in den Bereich 0,5ms (0% Throttle) bis 1,5ms (100% Throttle).

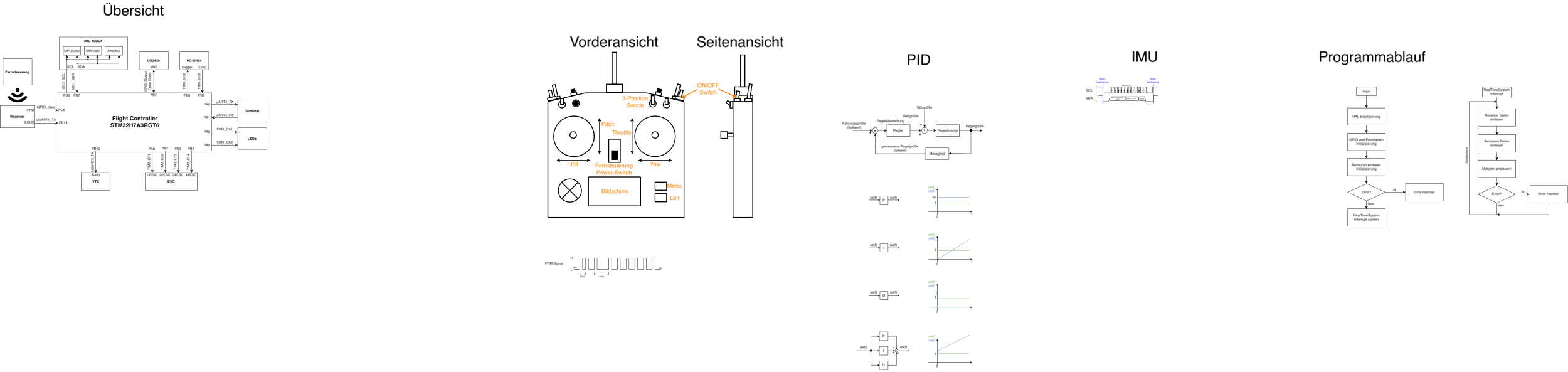


Abbildung 17: Beispiel PPM-Signal

Dadurch, dass die digitalen Protokolle um ein Vielfaches schneller und nicht sehr fehleranfällig sind, unterstützt die Steuerungssoftware das PPM-Protokoll nicht. Das PPM-Signal wird nur bei der Initialisierung für einen Verbindungstest mit GPIO-Input und nicht für das Fliegen der Drohne verwendet.

#### S.Bus

Das S.Bus-Protokoll ist das empfohlene Protokoll für die Benutzung der Drohne. Es handelt sich um ein digitales Protokoll, das mittels der UART-Peripherie eingelesen wird. Das Protokoll ist schneller als das analoge Signal PPM und weniger fehleranfällig als das I.Bus-Protokoll, da das Signal mit invertierten Pegeln übertragen wird. Die Datenpakete werden kontinuierlich gesendet, sind ungefähr 3ms lang und besitzen einen ungefähr 4,7ms langen low-aktiven Ruhezustand. Der digitale Wertebereich befindet sich zwischen 350 und 1680.

**UART-Konfiguration in STM32CubeMX**:

Ein Bild, das Text, Screenshot, Software, Webseite enthält.

Automatisch generierte Beschreibung

Abbildung 18: STM32CubeMX Einstellungen S.Bus

Die wichtigsten UART-Einstellungen sind in der folgenden Tabelle dargestellt:

|  |  |
| --- | --- |
| **Einstellung** | **Wert** |
| Mode | Single Wire (Half-Duplex) |
| Baud Rate | 100000 Bits/s |
| Word Length | 9 Bits (including Parity) |
| Parity | Even |
| Stop Bits | 2 |
| RX Pin Active Level Inversion | Enable |

**Protokolldaten:**

Das S.Bus Protokoll besteht aus 25 Bytes:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Byte[0] | S.Bus header, 0x0F | | | |
| Byte[1…22] | Daten, 16 Kanäle | | | |
| Byte[23] | Bit[0] (0x01)  Kanal 17 | Bit[1] (0x02)  Kanal 18 | Bit[2] (0x04)  Frame Lost Flag | Bit[3] (0x08)  Failsafe Flag |
| Byte[24] | S.Bus footer, 0x00 | | | |

Die Bytes[1…22] beinhalten die Daten der einzelnen Kanäle. Jeder Kanal besteht aus 11 Bits, die nacheinander gesendet werden, wobei die ersten Bits des Kanals den niedrigsten Stellenwert haben.

Beispiel für eine Dekodierung:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Byte** | 0 | 1 | 2 | 3 | … | 23 | 24 |
| **Daten**  header  0b001 0011 0001  Kanal 1  0b001 1001 0001  Kanal 2  flags  footer | 0x0F | 0b00110001 | 0b00110001 | 0b00110001 | … | 0b00000000 | 0x00 |

#### I.Bus

Das I.Bus-Protokoll ist das zweite, digitale Protokoll, welches der Receiver ausgeben kann. Die Daten werden über die UART-Peripherie eingelesen. Die Datenpakete werden kontinuierlich gesendet, sind ungefähr 3ms lang und besitzen einen ungefähr 4,7ms langen high-aktiven Ruhezustand. Im Gegensatz zum S.Bus Protokoll kann mit I.Bus kein Verbindungsverlust festgestellt werden. Der digitale Werteberiech befindet zwischen 1070 und 1920.

**UART-Konfiguration in STM32CubeMX**:

Ein Bild, das Text, Screenshot, Software, Computersymbol enthält.

Automatisch generierte Beschreibung

Abbildung 19: STM32CubeMX Einstellungen I.Bus

Die wichtigsten UART-Einstellungen sind in der folgenden Tabelle dargestellt:

|  |  |
| --- | --- |
| **Einstellung** | **Wert** |
| Mode | Single Wire (Half-Duplex) |
| Baud Rate | 115200 Bits/s |
| Word Length | 8 Bits (including Parity) |
| Parity | None |
| Stop Bits | 1 |
| RX Pin Active Level Inversion | Disable |

**Protokolldaten:**

Das I.Bus Protokoll besteht aus 32 Bytes:

|  |  |
| --- | --- |
| Byte[0] | Protokolllänge, 0x20 |
| Byte[1] | Command Code, 0x40 |
| Byte[2…29] | Daten, 14 Kanäle |
| Byte[30…31] | Checksumme |

Die Bytes[2…29] beinhalten die Daten der einzelnen Kanäle. Jeder Kanal besteht aus 2 Bytes, welche in der little-endian-byte-order geschickt werden. Die Checksumme besteht auch aus 2 Byte, welche in der little-endian-byte-order gesendet werden. Die Checksumme berechnet sich aus 0xFFFF weniger der Summe der ersten 30 Bytes.

Beispiel für die Dekodierung:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Byte** | 0 | 1 | 2 | 3 | … | 30 | 31 |
| **Daten**  Protokolllänge  Command Code  0x05DC  Kanal 1  0x3412  Checksumme | 0x20 | 0x40 | 0xDC | 0x05 | … | 0x12 | 0x34 |

Beispiel Checksumme:

### Initialisierung Empfangssoftware

Dadurch, dass der Receiver die Daten kontinuierlich sendet, wird ein DMA (direct memory access) - Controller zum Einlesen verwendet, der durchgehend auf die Werte wartet und diese einliest.

**DMA-Konfiguration in STM32CubeMX:**

Ein Bild, das Text, Screenshot, Software, Zahl enthält.

Automatisch generierte Beschreibung

Abbildung 20: Einstellungen DMA für Receiver in STM32CubeMX

Die wichtigste Einstellung ist, dass der Modus als *Circular* festgelegt wird. Dadurch wird der DMA-Stream automatisch neu gestartet, wenn dieser den Einlesevorgang abgeschlossen hat. Dies ermöglicht ein durchgehendes Einlesen.

Dateiname: receiver.c

/\*\*

 \* @brief This function calibrates and starts uart receive dma with selected protocol

 \* @param proto protocol to use (SBUS / IBUS)

 \* @param huart pointer to a UART\_HandleTypeDef structure (input u(s)art)

 \* @return Receiver\_Status

 \*/

Receiver\_Status Receiver\_Init(Receiver\_Protocol proto, UART\_HandleTypeDef \*huart)

{

  receiver\_InputUART = huart;         // set input uart

  receiver\_SelectedProtocol = proto;  // set serial protocol

  // set custom reception complete ISR

  HAL\_UART\_RegisterCallback(receiver\_InputUART, HAL\_UART\_RX\_COMPLETE\_CB\_ID, Receiver\_ReceptionCallback);

  switch(receiver\_SelectedProtocol)

  {

    /\*\*

     \* 115200 baud

     \* 8 data bits, 1 stop bit, no parity

     \* LSB first, not inverted

     \* 32 Bytes:

     \*      Byte[0]: protocol length, 0x20

     \*      Byte[1]: command code, 0x40

     \*      Byte[2-29]: channel data, 14 channels, 2 byte each, little endian

     \*      Byte[30-31]: checksum, little endian, 0xFFFF - sum of other 30 bytes = checksum

     \*/

    case IBUS:

    {

      // check if uart is configured via baudrate

      if(receiver\_InputUART->Init.BaudRate != 115200)

        return RECEIVER\_UART\_ERROR;

      // check if trnasmitter is connected (ppm signal reception)

      uint8\_t timeout = 0;

      int8\_t tmp\_PinState = HAL\_GPIO\_ReadPin(RECEIVER\_PPM\_GPIO\_Port, RECEIVER\_PPM\_Pin);

      while(tmp\_PinState == HAL\_GPIO\_ReadPin(RECEIVER\_PPM\_GPIO\_Port, RECEIVER\_PPM\_Pin))

      {

        // if the ppm signal doesn't change in 10ms -> error

        if(timeout++ > 10)

          return RECEIVER\_PPM\_ERROR;

        HAL\_Delay(1);

      }

      uint8\_t tmp[2] = {0};

      timeout = 0;

      // calibrate reception to begin of protocol

      while(!(tmp[0] == 0x20 && tmp[1] == 0x40))

      {

        // if the header is wrong 100x -> error

        if(timeout++ > 100)

          return RECEIVER\_TIMEOUT;

        HAL\_UART\_Receive(receiver\_InputUART, tmp, 2, 3);

      }

      HAL\_Delay(4); // wait to sync to next data packet

      // start DMA read i.bus signal

      if(HAL\_UART\_Receive\_DMA(receiver\_InputUART, receiver\_RawData, 32) != HAL\_OK)

        return IBUS\_ERROR;

      // set min/max values of receiver input data

      receiver\_InputLimits.min = 1070;

      receiver\_InputLimits.max = 1920;

      break;

    }

    /\*\*

     \* 100000 baud

     \* 9 data bits, 2 stop bit, even parity

     \* LSB first, inverted

     \* 25 Bytes:

     \*      Byte[0]: protocol header, 0x0F

     \*      Byte[1-22]: channel data, 16 channels, 11 bits each

     \*      Byte[23]:

     \*          bit[4]: signal failsafe flag

     \*          bit[5]: signal lost flag

     \*          bit[6]: digital channel 18

     \*          bit[7]: digital channel 17

     \*      Byte[24]: protocol footer, 0x00

     \*/

    case SBUS:

    {

      // check if uart is configured via baudrate

      if(receiver\_InputUART->Init.BaudRate != 100000)

        return RECEIVER\_UART\_ERROR;

      // check if transmitter is connected (ppm signal reception)

      uint8\_t timeout = 0;

      int8\_t tmp\_PinState = HAL\_GPIO\_ReadPin(RECEIVER\_PPM\_GPIO\_Port, RECEIVER\_PPM\_Pin);

      while(tmp\_PinState == HAL\_GPIO\_ReadPin(RECEIVER\_PPM\_GPIO\_Port, RECEIVER\_PPM\_Pin))

      {

        // if the ppm signal doesn't change in 10ms -> error

        if(timeout++ > 10)

          return RECEIVER\_PPM\_ERROR;

        HAL\_Delay(1);

      }

      uint8\_t tmp = 0;

      timeout = 0;

      // calibrate reception to begin of protocol

      while(tmp != 0x0F)

      {

        // if the header is wrong 100x -> error

        if(timeout++ > 100)

          return RECEIVER\_TIMEOUT;

        HAL\_UART\_Receive(receiver\_InputUART, &tmp, 1, 4);

      }

      HAL\_Delay(4); // wait to sync to next data packet

      // start DMA read s.bus signal

      if(HAL\_UART\_Receive\_DMA(receiver\_InputUART, receiver\_RawData, 25) != HAL\_OK)

        return SBUS\_ERROR;

      // set min/max values of receiver input data

      receiver\_InputLimits.min = 350;

      receiver\_InputLimits.max = 1680;

      break;

    }

    // wrong or no protocol selected

    case NO\_PROTO:

    default:

      return PROTOCOL\_ERROR;

      break;

  }

  // set value range and half value

  receiver\_InputLimits.delta = receiver\_InputLimits.max - receiver\_InputLimits.min;

  receiver\_InputLimits.half = (receiver\_InputLimits.max + receiver\_InputLimits.min) / 2;

  return RECEIVER\_OK;

}

In den Programm *Receiver\_Init()* wird je nach ausgewählten Empfangsprotokoll (SBUS oder IBUS) die UART-Peripherie mit den Datenpaketen synchronisiert und dann mit dem DMA-Controller durchgehend eingelesen.

### Empfangssoftware

Wenn der DMA-Controller ein Packet fertig eingelesen hat, wird ein Interrupt mit der Funktion *Receiver\_ReceptionCallback()* aufgerufen:

Dateiname: receiver.c

/\*\*

 \* @brief This function is the ISR for DMA receiver reception complete

 \* @details all data gets decoded, the motor speed gets regulated and data gets transmitted

 \* @param huart

 \*/

void Receiver\_ReceptionCallback(UART\_HandleTypeDef \*huart)

{

// dekodieren

  uint8\_t errorCode;

  errorCode = Receiver\_Decode();

  if(receiver\_SelectedProtocol == IBUS && errorCode == RECEIVER\_OK)

    Receiver\_IBusFailsafeCheck();

  // motor control

  errorCode = Receiver\_ConvertInput();

  if(errorCode != RECEIVER\_OK)

  {

    sprintf(txt, "Receiver Error %d\n\r", errorCode);

    Terminal\_Print(txt);

    if(errorCode == IBUS\_SIGNAL\_LOST\_ERROR || errorCode == SBUS\_SIGNAL\_LOST || errorCode == SBUS\_SIGNAL\_FAILSAFE)

      Receiver\_FailsafeHandler();

  }

  // get IMU data

  IMU\_GetAngles();

  IMU\_BARO\_ReadBaro();

// data transfer

  static int8\_t dataTransmitDelay = 0;

  if(dataTransmitDelay++ >= 60)

  {

    static int8\_t packetSelect = 0;

    if(packetSelect == 0)

      DATA\_TRANSMISSION\_1(ds2438\_Voltage, baroAltitude, 0x00);

    else

      DATA\_TRANSMISSION\_2(angle.pitch, angle.roll, angle.yaw);

    packetSelect = packetSelect == 0;

    dataTransmitDelay = 0;

  }

}

**Erster Teil - dekodieren:**

Zu Beginn werden die empfangene Daten mit der Funktion *Receiver\_Decode()* dekodiert.

Dateiname: receiver.c

/\*\*

 \* @brief This function decodes the receiver raw data depending on the protocol

 \* @details

 \* i.bus channel values from 1070 - 1920

 \* s.bus channel values from 350 - 1680

 \*

 \* what channel does what depends on the defines found in receiver.h:

 \*  - RECEIVER\_YAW\_CHANNEL

 \*  - RECEIVER\_PITCH\_CHANNEL

 \*  - RECEIVER\_THROTTLE\_CHANNEL

 \*  - RECEIVER\_ROLL\_CHANNEL

 \*  - RECEIVER\_ONOFF\_SWITCH\_CHANNEL

 \*  - RECEIVER\_MODESEL\_SWTICH\_CHANNEL

 \* @return Receiver\_Status

 \*/

Receiver\_Status Receiver\_Decode(void)

{

  switch(receiver\_SelectedProtocol)

  {

    case IBUS:

    {

    // if reception input start at the last byte -> reorder for correct order (rotate left)

      if(receiver\_RawData[1] == 0x20 && receiver\_RawData[2] == 0x40)

      {

        uint8\_t tmp = receiver\_RawData[0];

        for(int8\_t i = 1; i < 32; i++)

          receiver\_RawData[i - 1] = receiver\_RawData[i];

        receiver\_RawData[31] = tmp;

      }

      // check if protocol header is correct

      if(receiver\_RawData[0] != 0x20 || receiver\_RawData[1] != 0x40)

        return IBUS\_HEADER\_ERROR;

    // check if checksum is correct (0xFFFF - sum of other 30 bytes = checksum)

      uint16\_t sum = 0;

      for(int8\_t i = 0; i < 30; i++)

        sum += receiver\_RawData[i];

      uint16\_t checksum = (receiver\_RawData[31] << 8) | receiver\_RawData[30];

      if((0xFFFF - sum) != checksum)

        return IBUS\_CHECKSUM\_ERROR;

    // decode channel data (14 channels, 2 bytes each, little endian byte order)

      for(int8\_t i = 0, j = 0; i < 14; i++, j += 2)

        receiver\_ChData[i] = (receiver\_RawData[j + 3] << 8) | receiver\_RawData[j + 2];

    // check disconnection

      Receiver\_IBusFailsafeCheck();

      if(receiver\_SameDataCounter > 250)

        return IBUS\_SIGNAL\_LOST\_ERROR;

      break;

    }

    case SBUS:

    {

    // if reception input start at the last byte -> reorder for correct order (rotate left)

      if(receiver\_RawData[0] == 0x00 && receiver\_RawData[1] == 0x0F)

      {

        uint8\_t tmp = receiver\_RawData[0];

        for(int8\_t i = 1; i < 25; i++)

          receiver\_RawData[i - 1] = receiver\_RawData[i];

        receiver\_RawData[24] = tmp;

      }

      // check if protocol header is correct

      if(receiver\_RawData[0] != 0x0F)

        return SBUS\_HEADER\_ERROR;

    // check if protocol footer is correct

      if(receiver\_RawData[24] != 0x00)

        return SBUS\_FOOTER\_ERROR;

    // check signal lost flags

      if(receiver\_RawData[23] & 0x04)

        return SBUS\_SIGNAL\_LOST;

    // check signal failsafe flag

      if(receiver\_RawData[23] & 0x08)

        return SBUS\_SIGNAL\_FAILSAFE;

    // decode channel data (16 channels, 11 bits each, lsb first)

      for(int8\_t i = 0, j = 0; i < 16; i += 8, j += 11)

      {

        receiver\_ChData[i + 0] = ((receiver\_RawData[j + 1] >> 0) | (receiver\_RawData[j + 2] << 8)) & 0x7FF;

        receiver\_ChData[i + 1] = ((receiver\_RawData[j + 2] >> 3) | (receiver\_RawData[j + 3] << 5)) & 0x7FF;

        receiver\_ChData[i + 2] = ((receiver\_RawData[j + 3] >> 6) | (receiver\_RawData[j + 4] << 2) | (receiver\_RawData[5] << 10)) & 0x7FF;

        receiver\_ChData[i + 3] = ((receiver\_RawData[j + 5] >> 1) | (receiver\_RawData[j + 6] << 7)) & 0x7FF;

        receiver\_ChData[i + 4] = ((receiver\_RawData[j + 6] >> 4) | (receiver\_RawData[j + 7] << 4)) & 0x7FF;

        receiver\_ChData[i + 5] = ((receiver\_RawData[j + 7] >> 7) | (receiver\_RawData[j + 8] << 1) | (receiver\_RawData[9] << 9)) & 0x7FF;

        receiver\_ChData[i + 6] = ((receiver\_RawData[j + 9] >> 2) | (receiver\_RawData[j + 10] << 6)) & 0x7FF;

        receiver\_ChData[i + 7] = ((receiver\_RawData[j + 10] >> 5) | (receiver\_RawData[j + 11] << 3)) & 0x7FF;

      }

      break;

    }

    // wrong or no protocol selected

    case NO\_PROTO:

    default:

      return PROTOCOL\_ERROR;

      break;

  }

  return RECEIVER\_OK;

}

Diese Funktion wandelt die empfangenen Daten in S.Bus/I.Bus Kanalwerte um.

Es wird kontrolliert, ob keine Protokollfehler, wie zum Beispiel Header- oder Footer-Error, vorkommen. Im Fall eines Fehlers wird der dazugehörige Fehlercode von der Funktion zurückgegeben – [*siehe 5.1.3.1 Übersicht Fehlercodes*](#_Übersicht_Fehlercodes)

Da das I.Bus-Protokoll keine Failsafe Flags besitzt, wird der Verbindungsstatus mit der Funktion *Receiver\_IBusFailsafeCheck()* überprüft.

Dateiname: receiver.c

/\*\*

 \* @brief This function saves the current channel data and check if its the same as before

 \* @param huart pointer to UART\_HandleTypeDef

 \* @retval None

 \*/

void Receiver\_IBusFailsafeCheck(void)

{

  // only used for IBUS because SBUS does have a signal lost / failsafe flag

  if(receiver\_SelectedProtocol != IBUS)

    return;

  static uint16\_t receiver\_OldChData[16] = {0};   // previous channel data

  // when off -> don't check

  if(receiver\_ChData[RECEIVER\_ONOFF\_SWITCH\_CHANNEL] < receiver\_InputLimits.half)

  {

    receiver\_SameDataCounter = 0; // reset counter

  }

  else

  {

    int8\_t same = 1; // data is same flag

    for(int8\_t i = 0; i < 14 && same == 1; i++)

    {

      // check if the old channel data is not the same as the current

      if(receiver\_OldChData[i] != receiver\_ChData[i])

      {

        receiver\_SameDataCounter = 0; // reset channel data check

        same = 0;

      }

      // increment receiver\_SameDataCounter when data is the same

      else if(i == 14 - 1)

        (receiver\_SameDataCounter == UINT16\_MAX - 10) ? receiver\_SameDataCounter = 260 : receiver\_SameDataCounter++;

    }

  }

  // save current channel data

  for(int8\_t i = 0; i < 14; i++)

    receiver\_OldChData[i] = receiver\_ChData[i];

}

Im Falle des Verbindungsverlust sendet der Receiver durchgehend die exakt selben Kanalwerte. Daher zählt diese Funktion, wie oft das exakt selbe Paket mit denselben Werten gesendet worden ist. Dafür wird der Zähler *receiver\_SameDataCounter* verwendet. Wenn der Wert über 250 ist, liefert das Dekodierungsprogramm *Receiver\_Decode()* einen Failsafe-Error.

**Zweiter Schritt – motor control:**

Danach werden mit den dekodierten Kanaldaten, die Motoren mit der Funktion *Receiver\_ConvertInput()* angesteuert.

Dateiname: receiver.c

/\*\*

 \* @brief This function convert the input from the receiver to throttle percentage

 \* @details

 \* The max throttle values per mode can be changed in receiver.h with:

 \*  - ESC\_SAFEMODE\_THR\_MAX

 \*  - ESC\_NORMALMODE\_THR\_MAX

 \*  - ESC\_OFFMODE\_THR

 \*  - ESC\_TURN\_OFFSET\_MAX

 \* @return Receiver\_Status

 \*/

Receiver\_Status Receiver\_ConvertInput(void)

{

  if(currentStatus != RECEIVER\_OK)

    return currentStatus;

  \_\_HAL\_TIM\_SET\_COMPARE(LED\_TIM, LED\_RED\_CHANNEL, 0);

// \*\*\*\*\*\*\*\*\*\*\*\*\*

// ON/OFF switch

// \*\*\*\*\*\*\*\*\*\*\*\*\*

  float throttle = 0, pitch = 0, roll = 0, yaw = 0;

  droneOffModeFlag = 0;

  // top position (< half) = off (set standard throttle)

  if(receiver\_ChData[RECEIVER\_ONOFF\_SWITCH\_CHANNEL] < receiver\_InputLimits.half)

    droneOffModeFlag = 1;

  if(droneOffModeFlag == 1 && receiver\_ChData[RECEIVER\_THROTTLE\_CHANNEL] > receiver\_InputLimits.min + 10)

    droneOffModeFlag = 1;

// \*\*\*\*\*\*\*\*\*\*\*\*

// mode select

// \*\*\*\*\*\*\*\*\*\*\*\*

  uint16\_t esc\_MaxThr = ESC\_SAFEMODE\_THR\_MAX;

  uint8\_t hoverModeFlag = 0;

  // top position (< half) = safemode

  if(receiver\_ChData[RECEIVER\_MODESEL\_SWTICH\_CHANNEL] < receiver\_InputLimits.half - 10)

    esc\_MaxThr = ESC\_SAFEMODE\_THR\_MAX;

  // middle position (half +- 10) = normalmode

  else if(receiver\_ChData[RECEIVER\_MODESEL\_SWTICH\_CHANNEL] >= receiver\_InputLimits.half - 10 && receiver\_ChData[RECEIVER\_MODESEL\_SWTICH\_CHANNEL] <= receiver\_InputLimits.half + 10)

    esc\_MaxThr = ESC\_NORMALMODE\_THR\_MAX;

  // down position = extra mode hover mode

  else

    hoverModeFlag = 1;

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// calc throttle input

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

  if(droneOffModeFlag == 0)

    throttle = 0;

  else

  {

    throttle = (float)(receiver\_ChData[RECEIVER\_THROTTLE\_CHANNEL] - receiver\_InputLimits.min) / receiver\_InputLimits.delta; // get joystick position

    throttle \*= esc\_MaxThr; // get percent of max duty cycle addition

  }

  // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// calc pitch input

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

  if(droneOffModeFlag == 0)

    pitch = 0;

  else

  {

    pitch = (float)(receiver\_ChData[RECEIVER\_PITCH\_CHANNEL] - receiver\_InputLimits.min) / receiver\_InputLimits.delta; // get joystick position

    pitch -= 0.5f;

    pitch \*= ESC\_TURN\_OFFSET\_MAX; // get percent of max duty cycle addition

  }

  // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// calc roll input

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

  if(droneOffModeFlag == 1)

    roll = 0;

  else

  {

    roll = (float)(receiver\_ChData[RECEIVER\_ROLL\_CHANNEL] - receiver\_InputLimits.min) / receiver\_InputLimits.delta; // get joystick position

    roll -= 0.5f;

    roll \*= ESC\_TURN\_OFFSET\_MAX; // get percent of max duty cycle addition

  }

  // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// calc yaw input

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

  if(droneOffModeFlag == 1)

    yaw = 0;

  else

  {

    yaw = (float)(receiver\_ChData[RECEIVER\_YAW\_CHANNEL] - receiver\_InputLimits.min) / receiver\_InputLimits.delta; // get joystick position

    yaw -= 0.5f;

    yaw \*= ESC\_TURN\_OFFSET\_MAX; // get percent of max duty cycle addition

  }

  // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// output values

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

  if(hoverModeFlag == 1)

    PID\_Hover(throttle);

  else

    PID\_Normal(throttle, pitch, roll, yaw);

  return RECEIVER\_OK;

}

Diese Funktion extrahiert die Throttle, Pitch, Roll und Yaw Werte aus den Kanalwerten und sendet diese zum PID-Regler – [*siehe 5.6 PID-Regler*](#_PID-Regler)

**Dritter Schritt – get IMU data:**

Für die Erklärung und Funktionsweise der Befehle

– [*siehe 5.3 Inertial Measurement Unit (IMU)*](#_Inertial_Measurement_Unit)

**Vierter Schritt – data transfer:**

Für die Erklärung und Funktionsweise der Befehle

– *siehe Kapitel Ben Data Transmission*

## Inertial Measurement Unit (IMU)

Inertial Measurement Unit (IMU) ist ein Sammelbegriff für alle Sensoren, die die Kräfte auf einen Körper messen. Typischerweise besteht ein IMU aus einem Gyroskop und einen Beschleunigungsmesser (Accelerometer). In komplexeren Systemen sind weitere Sensoren, wie ein Kompass (Magnetometer) und ein Luftdrucksensor (Barometer) zusätzlich vorhanden.

Der Sensor wird meistens mit dem Begriff „Degrees Of Freedom / DOF“ beschrieben. Dieser Parameter gibt an, wie viele Achsen der Sensor messen kann.

Das heißt, dass zum Beispiel ein 6DOF-IMU, der aus einem Gyroskop und einen Accelerometer besteht, um sechs Achsen messen kann – pro Sensor die x-, y- und z-Achse.

Für die FPV-Drohne wird ein 10DOF-IMU mit einem Gyroskop, Accelerometer, Barometer und Magnetometer verwendet, die auf dem IMU-Breakout vorhanden sind. Mit diesen Sensoren werden die Lagewinkel, die Flughöhe, Temperatur und Regelgrößen für PID-Regler gemessen.

### I²C Protokoll

Das IMU-Breakout auf der FPV-Drohne kann nur mit dem I²C-Protokoll angesprochen werden. Dieses Protokoll besteht aus drei Leitungen: SDA (Datenleitung), SCL (Taktleitung) und GND (Masseleitung) und basiert auf dem Master-Slave-Prinzip.

Um die Datenübertragung zu starten, muss eine Startbedingung erfüllt werden: während die SCL-Leitung im high-Ruhezustand ist, wechselt die SDA-Leitung auf einen low-Zustand.

Nach der Startbedingung schickt der Master im System (Flight Controller) ein 400kHz Rechtecksignal mit einen Duty Cycle von 50% auf die SCL-Leitung. Zuerst wird auf der SDA-Leitung wird die I²C-Addresse des I²C-Slaves geschickt. Danach werden gleichzeitig die Daten gesendet beziehungsweise empfangen und bei einer steigenden Flanke der SCL-Leitung interpretiert. Nach jedem Datenbyte schickt der Empfänger der Daten ein Acknowledge-Bit (ACK / 0), um den Empfang der Daten zu bestätigen. Wenn das Acknowledge-Bit einen high-Pegel hat (not Acknowledge, NACK), trat bei der Übertragung ein Fehler auf.

Am Ende jeder Übertragung wird ein not-Acknowledge-Bit (NACK / 1) empfangen. Darauf folgt eine Stoppbedingung, die eine zeitlich invertierte Startbedingung ist. Während die SCL-Leitung sich in den high-Ruhezustand befindet, wechselt die SDA-Leitung von einen low-Zustand auch in den high-Ruhezustand.

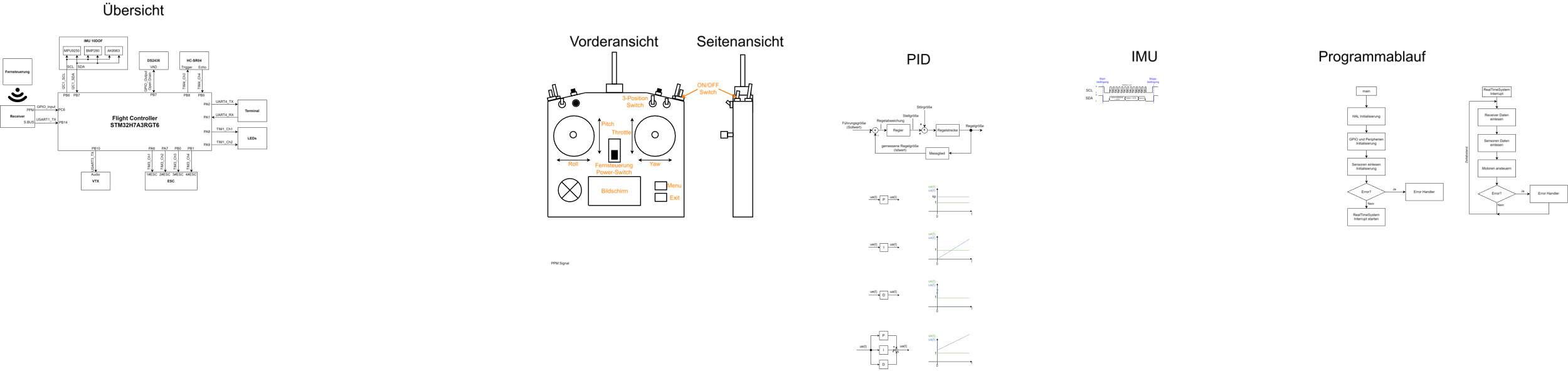


Abbildung 21: I²C Datentransfer

**I²C Einstellung in STM32CubeMX:**

Ein Bild, das Text, Screenshot, Software, Webseite enthält.

Automatisch generierte Beschreibung

Abbildung 22: STM32CubeMX Einstellungen IMU

Um die schnellste Kommunikation mit dem IMU-Breakout festzulegen, muss die Einstellung *I2C Speed Mode* auf *Fast Mode* und *I2C Speed Frequency (KHz)* auf *400* gesetzt werden.

Um zwischen mehreren I²C Geräten zu unterscheiden, hat jedes Gerät eine eigene I²C-Adresse. Die Adresse besteht aus einer 7-Bit-Zahl. Während der Kommunikation muss nach der Sensoradresse ein achtes Bit (R/W Bit) zur Adresse hinzugefügt werden, welches bei Schreibzyklen auf 0 gesetzt und bei Lesezyklen auf 1 gesetzt werden muss.

|  |  |  |
| --- | --- | --- |
| **Sensor** | **Slave Adressen + R/W Bit** | |
| **Schreiben** | **Lesen** |
| MPU9250 | 0xD0 | 0xD1 |
| BMP280 | 0xEE | 0xEF |
| AK8963 | 0x18 | 0x19 |

Damit mit den Sensoren des IMUs kommuniziert werden kann, müssen bestimmte Lese- und Schreibzyklen eingehalten werden:

#### Schreibzyklus IMU

Ein Bild, das Text, Screenshot, Reihe, Schrift enthält.

Automatisch generierte Beschreibung

Abbildung 23: I²C Schreibzyklus

|  |  |
| --- | --- |
| **Signal** | **Beschreibung** |
| S | Startbedingung |
| AD+W | Slave-Adresse + Write Bit |
| ACK | Acknowledge Bit |
| RA | Register Adresse |
| P | Stoppbedingung |

Es können mehrere Byte direkt hintereinander geschickt werden. Dabei können beliebig viele Datenbyte mit Acknowledge-Bit vor der Stoppbedingung gesendet werden. Ein interner Zeiger erhöht die Register Adresse um eine Stelle pro Datenbyte.

**Schreiben zu IMU-Register**:

Dateiname: IMU\_10DOF.c

/\*\*

 \* @brief This function writes an amount of bytes to registers from the IMU

 \* @param sensor MPU9250, AK8963 (MAG), BMP280 (BARO)

 \* @param regAddr register address

 \* @param data data to write

 \* @return IMU\_Status

 \*/

IMU\_Status IMU\_WriteRegister(IMU\_Sensor sensor, uint8\_t regAddr, uint8\_t data)

{

  // determine the I2C device address

  uint16\_t devAddress;

  switch(sensor)

  {

    case MPU9250:

      devAddress = IMU\_MPU\_I2C\_ADDR;

      break;

    case AK8963:

      devAddress = IMU\_MAG\_I2C\_ADDR;

      break;

    case BMP280:

      devAddress = IMU\_BARO\_I2C\_ADDR;

      break;

    default:

      return IMU\_ADDRESS\_ERROR;

  }

  // read register(s)

  HAL\_I2C\_Mem\_Write(imu\_ComI2C, devAddress, regAddr, I2C\_MEMADD\_SIZE\_8BIT, &data, 1, 1000);

  return IMU\_OK;

}

Die Funktion *HAL\_I2C\_Mem\_Write()* sendet automatisch die angegeben Daten und Befehle an den I²C-Peripheriebaustein, die für die Kommunikation notwendig sind. Wenn nach einer Sekunde die Kommunikation noch nicht abgeschlossen ist, liefert die Funktion einen Timeout-Error.

#### Lesezyklus IMU

Ein Bild, das Text, Reihe, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Abbildung 24: I²C Lesezyklus

|  |  |
| --- | --- |
| **Signal** | **Beschreibung** |
| S | Startbedingung |
| AD+W | Slave-Adresse + Write Bit |
| ACK | Acknowledge Bit |
| RA | Register Adresse |
| AD+R | Slave-Adresse + Read Bit |
| NACK | not Acknowledge Bit |
| P | Stoppbedingung |

Wie beim Schreibablauf können auch mehrere Byte direkt hintereinander eingelesen werden. Dabei wird die Kommunikation mit NACK vom Master im System beendet. Ein interner Zeiger erhöht die Register Adresse um eine Stelle pro Datenbyte.

**Lesen von IMU-Register/n:**

Dateiname: IMU\_10DOF.c

/\*\*

 \* @brief This function reads an amount of bytes from registers from the IMU

 \* @param sensor MPU9250, AK8963 (MAG), BMP280 (BARO)

 \* @param regAddr register address

 \* @param data data pointer

 \* @param rxBytes amount of bytes to read

 \* @return IMU\_Status

 \*/

IMU\_Status IMU\_ReadRegister(IMU\_Sensor sensor, uint8\_t regAddr, uint8\_t \*data, uint8\_t rxBytes)

{

  // determine the I2C device address

  uint16\_t devAddress;

  switch(sensor)

  {

    case MPU9250:

      devAddress = IMU\_MPU\_I2C\_ADDR;

      break;

    case AK8963:

      devAddress = IMU\_MAG\_I2C\_ADDR;

      break;

    case BMP280:

      devAddress = IMU\_BARO\_I2C\_ADDR;

      break;

    default:

      return IMU\_ADDRESS\_ERROR;

  }

  // read register(s)

  HAL\_I2C\_Mem\_Read(imu\_ComI2C, devAddress, regAddr, I2C\_MEMADD\_SIZE\_8BIT, data, rxBytes, 1000);

  return IMU\_OK;

}

Die Funktion *HAL\_I2C\_Mem\_Read()* sendet automatisch die Befehle an den I²C-Peripheriebaustein, die für die Kommunikation notwendig sind, und speichert die Daten in der angegeben Variable. Wenn nach einer Sekunde die Kommunikation noch nicht abgeschlossen ist, liefert die Funktion einen Timeout-Fehler.

### IMU-Verbindungstest

Alle drei Sensoren haben ein Who-Am-I-Register. Diese Register haben festgelegte Werte, die nicht geändert werden können. Es werden alle Sensoren überprüft, obwohl der Magnetometer nicht verwendet wird, damit eine vollständige Verbindungsüberprüfung mit dem gesamten IMU-Breakout durchgeführt werden kann.

|  |  |  |
| --- | --- | --- |
| **Sensor** | **Who Am I Register Adresse** | **Registerinhalt** |
| MPU9250 | 0x75 | 0x71 |
| BMP280 | 0xD0 | 0x58 |
| AK8963 | 0x00 | 0x48 |

**Verbindungstest aller IMU-Sensoren:**

Dateiname: IMU\_10DOF.c

/\*\*

 \* @brief This function checks the connection of all sensors on the IMU

 \* @attention This function enables the bypass mode in the MPU9250

 \* @return IMU\_Status

 \*/

IMU\_Status IMU\_CheckConnection(void)

{

  uint8\_t regVal[3] = {0x71, 0x48, 0x58};

  uint8\_t regAddr[3] = {IMU\_MPU\_WHOAMI\_ADDR, IMU\_MAG\_WHOAMI\_ADDR, IMU\_BARO\_CHIPID\_ADDR};

  uint8\_t sensor[3] = {MPU9250, MAG, BARO};

  uint8\_t timeout;

  uint8\_t data = 0x00;

  for(uint8\_t i = 0; i < 3; i++)

  {

    timeout = 0;

    while(data != regVal[i])

    {

      if(IMU\_ReadRegister(sensor[i], regAddr[i], &data, 1) != IMU\_OK)

        return IMU\_ADDRESS\_ERROR;

      if(timeout++ > 100)

        return IMU\_MPU\_WHOAMI\_ERROR + i;

    }

    if(sensor[i] == MPU9250)

    {

      // enable bypass mode

      if(IMU\_WriteRegister(MPU9250, IMU\_MPU\_INT\_PIN\_CFG\_ADDR, 0x02) != IMU\_OK)

        return IMU\_ADDRESS\_ERROR;

      IMU\_DelayUs(10000);

    }

  }

  return IMU\_OK;

}

Damit auf das Magnetometer zugegriffen werden kann, muss im MPU9250 im bypass enable Register das BYPASS\_EN – Bit gesetzt werden. Dadurch werden die I²C-Leitung durch den Sensor durchgeführt, und auf den anderen Sensoren kann zugegriffen werden.

Wenn nach 100 Lesezyklen der Registerwert immer noch falsch ist, liefert die Funktion einen *IMU\_<Sensor>\_WHOAMI\_ERROR*.

### Gyroskop und Accelerometer - MPU9250

#### Registerübersicht MPU9250

In der folgenden Abbildung wird die Registerübersicht, der verwendeten Registern des MPU9250 dargestellt:

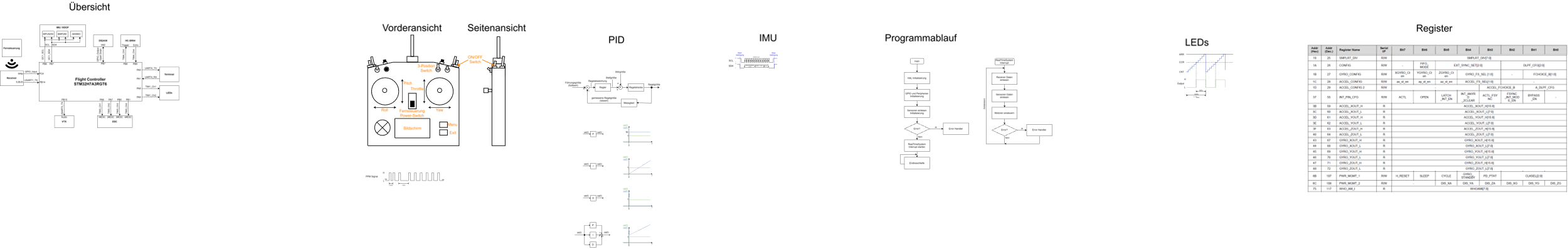


Abbildung 25: Registerübersicht MPU9250

|  |  |
| --- | --- |
| **Register** | **Verwendungszweck** |
| SMPLRT\_DIV | dividiert die interne Abtastrate |
| CONFIG | digitaler Tiefpassfilter für Gyroskop |
| GYRO\_CONFIG | Messbereich für Gyroskop |
| ACCEL\_CONFIG | Messbereich für Accelerometer |
| ACCEL\_CONFIG 2 | digitaler Tiefpassfilter für Accelerometer |
| INT\_PIN\_CFG | direkter I²C-Zugriff auf Magnetometer |
| ACCEL\_<Achse>OUT\_H/L | Messwerte von Accelerometer  <Achse>: X/Y/Z - Messachse  H/L: hoch- oder niederwertiges Byte |
| GYRO\_<Achse>OUT\_H/L | Messwerte von Gyroskop  <Achse>: X/Y/Z - Messachse  H/L: hoch- oder niederwertiges Byte |
| PWR\_MGMT\_1 | Sensor reset und Taktquelle |
| PWR\_MGMT\_2 | Sensoren aktivieren |
| WHO\_AM\_I | Überprüfung der Verbindung |

#### Initialisierung Accelerometer und Gyroskop

Dateiname: IMU\_10DOF.c | Ausschnitt aus IMU\_Init() Funktion

/\*\*

 \* @brief This function initialzes the 10DOF IMU (accel, gyro, mag, baro)

 \* @param imuInit pointer to IMU\_InitTypeDef

 \* @return IMU\_Status

 \*/

IMU\_Status IMU\_Init(IMU\_InitTypeDef \*imuInit)

{

...

// reset MPU

  IMU\_WriteRegister(MPU9250, IMU\_MPU\_PWR\_MGMT\_1\_ADDR, 0x00);

  IMU\_DelayUs(10000);

  // auto select best clk source

  IMU\_WriteRegister(MPU9250, IMU\_MPU\_PWR\_MGMT\_1\_ADDR, 0x01);

  // enable gyro and accel

  IMU\_WriteRegister(MPU9250, IMU\_MPU\_PWR\_MGMT\_2\_ADDR, 0x00);

  // select full scale range for gyro and accel

  IMU\_WriteRegister(MPU9250, IMU\_MPU\_ACCEL\_CONFIG\_ADDR, imuInit->accelFS << 3);

  IMU\_WriteRegister(MPU9250, IMU\_MPU\_GYRO\_CONFIG\_ADDR, imuInit->gyroFS << 3);

  // select digital low pass filter for gyro and accel

  IMU\_WriteRegister(MPU9250, IMU\_MPU\_ACCEL\_CONFIG\_2\_ADDR, imuInit->accelDLPF);

  IMU\_WriteRegister(MPU9250, IMU\_MPU\_CONFIG\_ADDR, imuInit->gyroDLPF);

  // select fastest sample rate

  IMU\_WriteRegister(MPU9250, IMU\_MPU\_SMPLRT\_DIV\_ADDR, 0x00);

  // calculate sensitivity scale factor (LSB/g and LSB/(°/s))

  accelSens = IMU\_ACCEL\_RES\_MAX / (1 << imuInit->accelFS);

  gyroSens = IMU\_GYRO\_RES\_MAX / (1 << imuInit->gyroFS);

  // calibrate gyro

  uint16\_t amount = 2000;

  IMU\_RegCoordinates tempGyro = {0};

  int32\_t tempX = 0, tempY = 0, tempZ = 0;

  for(uint16\_t i = 0; i < amount; i++)

  {

    tempGyro = IMU\_MPU\_ReadGyro();

    tempX += tempGyro.x;

    tempY += tempGyro.y;

    tempZ += tempGyro.z;

    IMU\_DelayUs(3000);

  }

  gyroOffset.x = (float)tempX / (float)amount;

  gyroOffset.y = (float)tempY / (float)amount;

  gyroOffset.z = (float)tempZ / (float)amount;

...

  return IMU\_OK;

}

Bemerkung:

Die Punkte ... bedeuten, dass vor und nach dem Ausschnitt noch weiterer Programmteile sind, die nicht in direkte Verbindung mit der MPU9250 stehen.

Die Messwerte eines Gyroskops driften bei Bewegung in eine Richtung [*(siehe: 2.1.1 Gyroskop)*](#_Gyroskop). Um dem dagegen zu wirken, wird ein Gyroskop Offset bestimmt. Es wird nach 2000 Messung der durchschnittliche Messwert bestimmt, der als Offsetwert verwendet wird. Im Programm wird das Ergebnis in der globalen Variable *gyroOffset* gespeichert.

#### Einlesen der Accelerometer- und Gyroskop-Daten

Die Messdaten des Accelerometers werden in den Registern 59 ACCEL\_XOUT\_H bis zum Register 64 ACCEL\_ZOUT\_L gespeichert. XOUT, YOUT und ZOUT bestimmt die einzelne Messachse und \_H und \_L bestimmt das high- und low-Byte des Messwertes.

**Auslesen der Accelerometer-Daten**:

Dateiname: IMU\_10DOF.c

/\*\*

 \* @brief This function reads accelerometer register data (x,y,z)

 \* @return IMU\_RegCoordinates

 \*/

IMU\_RegCoordinates IMU\_MPU\_ReadAccel(void)

{

  uint8\_t buffer[6] = {0};

  IMU\_ReadRegister(MPU9250, IMU\_MPU\_ACCEL\_XOUT\_H\_ADDR, buffer, 6);

  IMU\_RegCoordinates accelData = {0};

  accelData.x = ((int16\_t)buffer[0] << 8) | buffer[1];

  accelData.y = ((int16\_t)buffer[2] << 8) | buffer[3];

  accelData.z = ((int16\_t)buffer[4] << 8) | buffer[5];

  return accelData;

}

In dem Programm werden die Daten beginnend mit dem Register 59 nacheinander bis zum Register 64 eingelesen und danach zusammengefügt.

Die Messung des Gyroskops funktioniert gleich mit dem Accelerometer. Die Messdaten werden in den Registern 67 GYRO\_XOUT\_H bis zum Register 72 GYRO\_ZOUT\_L gespeichert. XOUT, YOUT und ZOUT bestimmen die einzelne Messachse und \_H und \_L bestimmen das high- und low-Byte des Messwertes.

**Auslesen der Gyroskop-Daten**:

Dateiname: IMU\_10DOF.c

/\*\*

 \* @brief This function reads gyroscope register data (x,y,z)

 \* @return IMU\_RegCoordinates

 \*/

IMU\_RegCoordinates IMU\_MPU\_ReadGyro(void)

{

    uint8\_t buffer[6] = {0};

    IMU\_ReadRegister(MPU9250, IMU\_MPU\_GYRO\_XOUT\_H\_ADDR, buffer, 6);

    IMU\_RegCoordinates gyroData = {0};

    gyroData.x = ((int16\_t)buffer[0] << 8) | (int16\_t)buffer[1];

    gyroData.y = ((int16\_t)buffer[2] << 8) | (int16\_t)buffer[3];

    gyroData.z = ((int16\_t)buffer[4] << 8) | (int16\_t)buffer[5];

    return gyroData;

}

Bemerkung:

Bei beide Programmen werden nur die Registerwerte zurückgeliefert. Um die eigentlichen Messwerte zu bekommen, müssen die Werte mit dem sensitivity scale factoren des Accelerometers und des Gyroskops gerechnet werden. Diese Faktoren werden im Initialisierungsprogramm berechnet und in den globalen Variablen *accelSens* und *gyroSens* gespeichert [*(siehe: 5.3.3.2 Initialisierung Accelerometer und Gyroskop)*](#_Initialisierung_Accelerometer_und).

Um die Drift vom Gyroskop entgegenzuwirken, wird der Registerwert noch mit dem Gyroskop-Offset gerechnet. Diese Werte werden auch im Initialisierungsprogramm bestimmt und in der globalen Variable *gyroOffset* gespeichert [*(siehe: 5.3.3.2 Initialisierung Accelerometer und Gyroskop)*](#_Initialisierung_Accelerometer_und).

**Berechnung der eigentlichen Messwerte**:

IMU\_RegCoordinates gyroData = IMU\_MPU\_ReadGyro();

IMU\_RegCoordinates accelData = IMU\_MPU\_ReadAccel();

gyro.x = (gyroData.x - gyroOffset.x) / gyroSens;

gyro.y = (gyroData.y - gyroOffset.y) / gyroSens;

gyro.z = (gyroData.z - gyroOffset.z) / gyroSens;

accel.x = (accelData.x / accelSens) - 0.01f;

accel.y = (accelData.y / accelSens) - 0.02f;

accel.z = (accelData.z / accelSens) - 0.1f;

Die Offsetwerte bei der Beschleunigungsberechnung (0,01; 0,02 und 0,1) müssen händisch bestimmt werden. Wenn der Sensor auf einer ebenen Fläche gerade liegt, muss der Accelerometer eine Beschleunigung von 1g auf der z-Achse messen. Der Offset ergibt sich aus der eigentlichen Messung auf einer ebenen Fläche von 1,01g. Dieser Vorgang muss mit 90° Drehungen für allen Achsen wiederholt werden, um die restlichen Offsetwerte zu bestimmen.

#### Komplementärfilter

Um die Lagewinkel Pitch, Roll und Yaw zu bestimmen, wird ein Komplementärfilter auf die Messwerte des MPU9250 angewendet [*(siehe: 2.2.5 Komplementärfilter)*](#_Komplementärfilter).

**Anwendung des Komplementärfilters:**

Dateiname: IMU\_10DOF.c

/\*\*

 \* @brief This function calculates pitch,roll and yaw

 \* @details data gets stored in the global variable 'angle'

 \* @retval None

 \*/

void IMU\_GetAngles(void)

{

  // read counter value since last function call

  uint16\_t tmpTime = \_\_HAL\_TIM\_GET\_COUNTER(imu\_DelayTIM);

  // reset timer

  \_\_HAL\_TIM\_SET\_COUNTER(imu\_DelayTIM, 0);

  // start first time call

  static int8\_t firstTimeFlag = 0;

  if(firstTimeFlag == 0)

  {

    firstTimeFlag = 1;

    return;

  }

  // calculate delta time

  imu\_DeltaTime = (float)tmpTime / 1E6f;

  // read sensors

  IMU\_RegCoordinates gyroData = IMU\_MPU\_ReadGyro();

  IMU\_RegCoordinates accelData = IMU\_MPU\_ReadAccel();

  // calculate actual values

  gyro.x = (gyroData.x - gyroOffset.x) / gyroSens;

  gyro.y = (gyroData.y - gyroOffset.y) / gyroSens;

  gyro.z = (gyroData.z - gyroOffset.z) / gyroSens;

  accel.x = (accelData.x / accelSens) - 0.01f;

  accel.y = (accelData.y / accelSens) - 0.02f;

  accel.z = (accelData.z / accelSens) - 0.1f;

  // invert axis because the sensor is upside down

  accel.z = -accel.z;

  // apply complementary filter to calc angles

  float accelPitch = atan2(accel.y, accel.z) \* RAD2DEG;

  float accelRoll = atan2(accel.x, accel.z) \* RAD2DEG;

  angle.roll = 0.98f \* (angle.roll + gyro.y \* imu\_DeltaTime) + 0.02f \* accelRoll;

  angle.pitch = 0.98f \* (angle.pitch - gyro.x \* imu\_DeltaTime) + 0.02f \* accelPitch;

  angle.yaw += gyro.z \* imu\_DeltaTime;

}

Die Funktion liest die Registerwerte vom Accelerometer und Gyroskop ein. Danach werden die eigentlichen Messwerte bestimmt. Mittels Komplementärfilter werden die Lagewinkel ausgerechnet, die in der Variable *angle* gespeichert werden.

Dadurch, dass der Sensor verkehrt auf der Drohne montiert ist, muss der Messwert um die z-Achse des Accelerometer invertiert werden.

### Barometer / Luftdrucksensor - BMP280

#### Registerübersicht BMP280

In der folgenden Abbildung wird die Registerübersicht, der verwendeten Registern des BMP280 dargestellt:

Ein Bild, das Text, Screenshot, Reihe, Zahl enthält.

Automatisch generierte Beschreibung

Abbildung 26: Registerübersicht BMP280

|  |  |
| --- | --- |
| **Register** | **Verwendungszweck** |
| temp\_xlsb | Temperatur Messwerte |
| temp\_lsb/msb |
| press\_xlsb | Luftdruck Messwerte |
| press\_lsb/msb |
| config | Zeitkonstanten |
| ctrl\_meas | Oversampling- und Modus-Auswahl |
| status | Status überprüfen |
| reset | Sensor zurücksetzen |
| id | Überprüfung der Verbindung |
| calib25…calib00 | Kompensationsparameter auslesen |

#### Initialisierung Barometer

Dateiname: IMU\_10DOF.c | Ausschnitt aus IMU\_Init() Funktion

/\*\*

 \* @brief This function initialzes the 10DOF IMU (accel, gyro, mag, baro)

 \* @param imuInit pointer to IMU\_InitTypeDef

 \* @return IMU\_Status

 \*/

IMU\_Status IMU\_Init(IMU\_InitTypeDef \*imuInit)

{

...

  IMU\_WriteRegister(BMP280, IMU\_BARO\_RESET\_ADDR, 0xB6); // reset barometer

  // check status of device

  uint8\_t timeout = 0, status = 1;

  while(status != 0x00)

  {

    IMU\_ReadRegister(BMP280, IMU\_BARO\_STATUS\_ADDR, &status, 1);

    if(timeout++ > 100)

      return IMU\_BARO\_INIT\_ERROR;

  }

  IMU\_BARO\_ReadCompensationValues();

  // set standby time and time constant of IIR filter

  uint8\_t config = ((imuInit->baroSBT << 5) | (imuInit->baroCoeff << 2));

  IMU\_WriteRegister(BMP280, IMU\_BARO\_CONFIG\_ADDR, config);

  // set oversampling settings for temperature and pressure measurement and set normal mode

  uint8\_t ctrl = (imuInit->baroTempOS << 5) | (imuInit->baroPressOS << 2) | 0x03;

  IMU\_WriteRegister(BMP280, IMU\_BARO\_CTRL\_MEAS\_ADDR, ctrl);

  // get current altitude level

  IMU\_DelayUs(UINT16\_MAX - 1);

  float baroSum = 0;

  for(uint16\_t i = 0; i < amount; i++)

  {

    IMU\_BARO\_ReadBaro();

    baroSum += baroAltitude;

    IMU\_DelayUs(1000);

  }

  baroAltitudeOffset = baroSum / amount;

  return IMU\_OK;

}

Bemerkung:

Die Punkte ... bedeuten, dass vor und nach dem Ausschnitt noch weiterer Programmteile sind, die nicht in direkte Verbindung mit dem BMP280 stehen.

Das Barometer wird immer im Normal-Mode betrieben.

Für die richtige Luftdruck- und Höhenbestimmung müssen die Registerwerte kompensiert werden. Diese Kompensationswerte werden vom Sensor zur Verfügung gestellt und müssen vor Beginn der Werteberechnung eingelesen werden. Das Einlesen wird mit der Funktion *IMU\_Baro\_ReadCompensationValues()* realisiert.

**Kompensationswerte für Barometermessung auslesen:**

Dateiname: IMU\_10DOF.c

/\*\*

 \* @brief This function reads the temperature and pressure compensation values

 \* @details values get stored in variable "baroCompensation"

 \* @retval None

 \*/

void IMU\_BARO\_ReadCompensationValues(void)

{

  uint8\_t buffer[24];

  IMU\_ReadRegister(BMP280, IMU\_BARO\_DIG\_T1\_L\_ADDR, buffer, 24);

  baroCompensation.T1 = (buffer[1] << 8) | buffer[0];

  baroCompensation.T2 = (buffer[3] << 8) | buffer[2];

  baroCompensation.T3 = (buffer[5] << 8) | buffer[4];

  baroCompensation.P1 = (buffer[7] << 8) | buffer[6];

  baroCompensation.P2 = (buffer[9] << 8) | buffer[8];

  baroCompensation.P3 = (buffer[11] << 8) | buffer[10];

  baroCompensation.P4 = (buffer[13] << 8) | buffer[12];

  baroCompensation.P5 = (buffer[15] << 8) | buffer[14];

  baroCompensation.P6 = (buffer[17] << 8) | buffer[16];

  baroCompensation.P7 = (buffer[19] << 8) | buffer[18];

  baroCompensation.P8 = (buffer[21] << 8) | buffer[20];

  baroCompensation.P9 = (buffer[23] << 8) | buffer[22];

}

Die Daten werden vom Hersteller angegeben und sind konstant in den Registern gespeichert. Die zusammengefügten Werte werden in der globalen Variable *baroCompensation* gespeichert.

#### Einlesen der Barometer-Daten

Die Messdaten vom BMP280 werden in den Registern 0xF7 press\_msb bis 0xFC temp\_xlsb gespeichert.

**Einlesen der Barometerwerte:**

Dateiname: IMU\_10DOF.c | Ausschnitt aus IMU\_BARO\_ReadBaro() Funktion

/\*\*

 \* @brief This function reads the barometer values and calculates temperature, pressure and altitude

 \* @details values gets stored in global variables 'baroTemperature', 'baroPressure' and 'baroAltitude'

 \* @retval None

 \*/

void IMU\_BARO\_ReadBaro(void)

{

  uint8\_t buffer[6] = {0};

  IMU\_ReadRegister(BMP280, IMU\_BARO\_PRESS\_ADDR, buffer, 6);

  int32\_t adcPress = ((int32\_t)buffer[0] << 12) | ((int32\_t)buffer[1] << 4) | ((int32\_t)buffer[2] >> 4);

  int32\_t adcTemp = ((int32\_t)buffer[3] << 12) | ((int32\_t)buffer[4] << 4) | ((int32\_t)buffer[5] >> 4);

  int32\_t fineTemp;

  int32\_t temp = IMU\_BARO\_CompensateTemp(adcTemp, &fineTemp);

  uint32\_t press = IMU\_BARO\_CompensatePress(adcPress, fineTemp);

  baroTemperature = (float)temp / 100.0;

  baroPressure = (float)press / 256.0;

...

}

Bemerkung:

Die Punkte ... bedeuten, dass vor und nach dem Ausschnitt noch weiterer Programmteile sind, die nicht in direkte Verbindung mit dem Barometer stehen.

Für die Berechnung müssen die Temperatur und der Luftdruck kompensiert werden. Dafür werden die Funktionen *IMU\_BARO\_CompensateTemp()* und *IMU\_BARO\_CompensatePress()* verwendet. Diese Funktionen sind im Datenblatt des Sensors vom Hersteller vorgegeben und wurden für die Nutzung in der Steuerungssoftware angepasst und übernommen.

**Barometer Temperatur- und Luftdruckwerte kompensieren:**

Dateiname: IMU\_10DOF.c

/\*\*

 \* @brief This function compensates the temperature according to the datasheet

 \* @details

 \* Returns temperature in DegC, resolution is 0.01 DegC. Output value of “5123” equals 51.23 DegC.

 \* @param adcTemp measured temperature

 \* @param fineTemp

 \* @return int32\_t (temperature)

 \*/

int32\_t IMU\_BARO\_CompensateTemp(int32\_t adcTemp, int32\_t \*fineTemp)

{

  int32\_t var1, var2, T;

  var1 = ((((adcTemp >> 3) - ((int32\_t)baroCompensation.T1 << 1))) \* ((int32\_t)baroCompensation.T2)) >> 11;

  var2 = (((((adcTemp >> 4) - ((int32\_t)baroCompensation.T1)) \* ((adcTemp >> 4) - ((int32\_t)baroCompensation.T1))) >> 12) \* ((int32\_t)baroCompensation.T3)) >> 14;

  \*fineTemp = var1 + var2;

  T = (\*fineTemp \* 5 + 128) >> 8;

  return T;

}

/\*\*

 \* @brief This function compensates the pressure according to the datasheet

 \* @details

 \* Returns pressure in Pa as unsigned 32 bit integer in Q24.8 format (24 integer bits and 8 fractional bits).

 \* Output value of “24674867” represents 24674867/256 = 96386.2 Pa = 963.862 hPa

 \* @param adcPress measured pressure

 \* @param fineTemp

 \* @return uint32\_t (pressure)

 \*/

uint32\_t IMU\_BARO\_CompensatePress(int32\_t adcPress, int32\_t fineTemp)

{

  int64\_t var1, var2, p;

  var1 = ((int64\_t)fineTemp) - 128000;

  var2 = var1 \* var1 \* (int64\_t)baroCompensation.P6;

  var2 = var2 + ((var1 \* (int64\_t)baroCompensation.P5) << 17);

  var2 = var2 + (((int64\_t)baroCompensation.P4) << 35);

  var1 = ((var1 \* var1 \* (int64\_t)baroCompensation.P3) >> 8) + ((var1 \* (int64\_t)baroCompensation.P2) << 12);

  var1 = (((((int64\_t)1) << 47) + var1)) \* ((int64\_t)baroCompensation.P1) >> 33;

  if(var1 == 0)

  {

    return 0; // avoid exception caused by division by zero

  }

  p = 1048576 - adcPress;

  p = (((p << 31) - var2) \* 3125) / var1;

  var1 = (((int64\_t)baroCompensation.P9) \* (p >> 13) \* (p >> 13)) >> 25;

  var2 = (((int64\_t)baroCompensation.P8) \* p) >> 19;

  p = ((p + var1 + var2) >> 8) + (((int64\_t)baroCompensation.P7) << 4);

  return (uint32\_t)p;

}

#### Berechnung der Höhe

Mit dem gemessenen Luftdruck kann die aktuelle Höhe über dem Meeresspiegel ausgerechnet werden. Dafür wird eine Höhenberechnungsformel aus dem Datenblatt übernommen:

Ein Bild, das Text, Schrift, Diagramm, Reihe enthält.

Automatisch generierte Beschreibung

Abbildung 27: Formel Berechnung Höhe über Meeresspiegel

altitude … Höhe über den Meeresspiegel in Meter (m)

p … gemessener Luftdruck in Hektopascal (hPa)

p0 … typischer Luftdruck am Meeresspiegel, normalerweise 1013,25hPa

**Höhe über Meeresspiegel bestimmen:**

IMU\_10DOF.c | Ausschnitt aus IMU\_BARO\_ReadBaro() Funktion

void IMU\_BARO\_ReadBaro(void)

{

...

  // convert pressure to altitude according to datasheet

  float presshPa = baroPressure / 100;

  baroAltitude = (44330.0 \* (1.0 - pow(presshPa / 1013.25, 1.0 / 5.255))) - baroAltitudeOffset;

}

Bemerkung:

Die Punkte ... bedeuten, dass vor und nach dem Ausschnitt noch weiterer Programmteile sind, die nicht in direkte Verbindung mit der Höhenbestimmung des Barometers stehen.

Dadurch, dass in der Berechnung die Variable *baroAltitudeOffset* verwendet wird, berechnet sich die relative Höhe der Drohne zum Boden. Wenn dieser Teil der Berechnung entfernt wird, kann die absolute Höhe bestimmt werden.

## Smart Battery Monitor - DS2438

Für die Diplomarbeit wird der DS2438 für die Spannungsüberwachung des Akkus verwendet, um zu verhindert, dass die Akkuspannung unter einen kritischen Grenzwert fällt und die Drohne abstürzt. Die Kommunikation zwischen dem Flight-Controller und dem DS2438 findet mit dem One-Wire Protokoll statt.

Als Vorlage für die Programmierung wurde ein in der 4. Klasse erstelltes Projekt verwendet. Nach Absprache mit den Autoren (Lukas Lindmayr und Marcel Bieder) wurde die Erlaubnis für die Verwendung und Anpassung an die FPV-Drohne Diplomarbeit erteilt.

### One-Wire Protokoll

Der Flight Controller kommuniziert mit dem DS2438 über das One-Wire Protokoll. Dieses besteht aus einer einzelnen halbduplexen, bidirektionalen Leitung. Der Cortex-M7 hat keine One-Wire Peripherie, deswegen muss das Protokoll selbständig implementiert werden. Die Kommunikation mit dem DS2438 beginnt immer aus einer Initialisierungssequenz. Als nächstes wird ein ROM-Funktionsbefehl ausgeführt. Danach wird ein Memory-Funktions-Befehl gesendet, und am Ende werden die Daten ausgetauscht.

**GPIO-Einstellungen in STM32CubeMX**:

Ein Bild, das Text, Screenshot, Software, Zahl enthält.

Automatisch generierte Beschreibung

Abbildung 28: STM32CubeMX Einstellungen DS2438

Damit die Kommunikation funktionieren kann, müssen in STM32CubeMX der *GPIO mode* auf *Output Open Drain* und der *GPIO Pull-up/Pull-down* auf *No pull-up and no pull-down* gesetzt werden.

Der Pin PC0 wird am Flight-Controller für DS2438\_DQ (Datenleitung) verwendet. Dieser Pin kann bei dem Funktionsaufruf von *DS2438\_Init()* bestimmt werden. [*(siehe: 5.4.3 Initialisierung DS2438)*](#_Initialisierung_DS2438)

#### One-Wire Schreibzyklen

Bei der Kommunikation wird das niederwertigste Bit immer zuerst gesendet.

**Um das Bit 1 zu senden, müssen folgende Zeitbedingungen eingehalten werden:**

Ein Bild, das Text, Diagramm, Reihe, Schrift enthält.

Automatisch generierte Beschreibung

Abbildung 29: One-Wire Bit 1 senden

**Um das Bit 0 zu senden, müssen andere Zeitbedingungen eingehalten werden**:

Ein Bild, das Text, Reihe, Diagramm, Quittung enthält.

Automatisch generierte Beschreibung

Abbildung 30: One-Wire Bit 0 senden

**Ein Byte zum DS2438 senden:**

Dateiname: DS2438.c

/\*\*

 \* @brief This function writes one byte to the DS2438

 \* @param byte byte to write

 \* @retval None

 \*/

void DS2438\_WriteByte(uint8\_t byte)

{

  for(int8\_t i = 0; i < 8; i++)

  {

    DS2438\_WriteBit(byte & 0x01);

    byte >>= 1;

  }

}

/\*\*

 \* @brief This function writes one bit to the DS2438

 \* @param bit bit to write

 \* @retval None

 \*/

void DS2438\_WriteBit(int8\_t bit)

{

  if(bit == 1)

  {

    HAL\_GPIO\_WritePin(ds2438\_GPIOPort, ds2438\_GPIOPin, GPIO\_PIN\_RESET);

    DS2438\_DelayUs(10);

    HAL\_GPIO\_WritePin(ds2438\_GPIOPort, ds2438\_GPIOPin, GPIO\_PIN\_SET);

    DS2438\_DelayUs(70);

  }

  else

  {

    HAL\_GPIO\_WritePin(ds2438\_GPIOPort, ds2438\_GPIOPin, GPIO\_PIN\_RESET);

    DS2438\_DelayUs(60);

    HAL\_GPIO\_WritePin(ds2438\_GPIOPort, ds2438\_GPIOPin, GPIO\_PIN\_SET);

    DS2438\_DelayUs(10);

  }

}

Um auf eine bestimmte Page Daten schreiben zu können, müssen diese zuerst auf das Scratchpad geschrieben werden. Danach muss das Scratchpad in den ROM oder RAM kopiert werden.

Zuerst wird die Initialisierungssequenz ausgeführt, und danach wird der Skip-ROM- (0xCC) und der Write-Scratchpad-Befehl (0x4E) ausgeführt. Anschließend muss die gewünschte Page-Nummer übertragen werden. Darauf folgen die Daten und eine zweite Initialisierungssequenz. Damit die Daten vom Scratchpad auch in den Speicher des Sensors übertragen werden, muss noch ein Skip-ROM- (0xCC) und ein Copy-Scratchpad-Befehl (0x48), gefolgt von der Page-Nummer, ausgeführt werden.

**Zu einer Page Daten senden:**

Dateiname: DS2438.c

/\*\*

 \* @brief This function writes the data to one page of the DS2438

 \* @param page page number (0 - 7)

 \* @param pageData data of page

 \* @return DS2438\_Status

 \*/

DS2438\_Status DS2438\_WritePage(uint8\_t page, int16\_t \*pageData)

{

  // reset + presence pulse

  if(DS2438\_Reset() == DS2438\_ERROR)

    return DS2438\_ERROR;

// copy current data to scratchpad

  DS2438\_WriteByte(DS2438\_SKIP\_ROM);

  DS2438\_WriteByte(DS2438\_WRITE\_SP);

  DS2438\_WriteByte(page);

  for(uint8\_t i = 0; i < 9; i++)

    DS2438\_WriteByte(pageData[i]);

// reset + presence pulse

  if(DS2438\_Reset() == DS2438\_ERROR)

    return DS2438\_ERROR;

  DS2438\_WriteByte(DS2438\_SKIP\_ROM);

  DS2438\_WriteByte(DS2438\_COPY\_SP);

  DS2438\_WriteByte(page);

  return DS2438\_OK;

}

#### One-Wire Lesezyklen

Genau wie bei den Schreibzyklen gibt es auch spezielle Zeitkriterien, die bei den Lesezyklen eingehalten werden müssen. Es wird auch das niederwertigste Byte zuerst empfangen.

Ein Bild, das Text, Diagramm, Reihe, Schrift enthält.

Automatisch generierte Beschreibung

Abbildung 31: One-Wire Bit empfangen

**Ein Byte vom DS2438 auslesen:**

Dateiname: DS2438.c

/\*\*

 \* @brief This function reads one byte from the DS2438 (LSB first)

 \* @return uint8\_t

 \*/

uint8\_t DS2438\_ReadByte(void)

{

  uint8\_t byte = 0;

  for(int8\_t i = 0; i < 8; i++)

    byte |= (DS2438\_ReadBit() << i);

  return byte;

}

/\*\*

 \* @brief This function reads one bit from the DS2438

 \* @return int8\_t

 \*/

int8\_t DS2438\_ReadBit(void)

{

  int8\_t bit = 0;

  HAL\_GPIO\_WritePin(ds2438\_GPIOPort, ds2438\_GPIOPin, GPIO\_PIN\_RESET);

  DS2438\_DelayUs(10);

  HAL\_GPIO\_WritePin(ds2438\_GPIOPort, ds2438\_GPIOPin, GPIO\_PIN\_SET);

  DS2438\_DelayUs(10);

  // read current pin level

  bit = HAL\_GPIO\_ReadPin(ds2438\_GPIOPort, ds2438\_GPIOPin) == GPIO\_PIN\_SET;

  DS2438\_DelayUs(60);

  return bit;

}

Um von einer bestimmen Page Daten lesen zu können, müssen diese zuerst vom Sensor auf das Scratchpad geschrieben werden. Danach müssen die Daten vom Scratchpad eingelesen werden.

Zuerst wird die Initialisierungssequenz ausgeführt und danach wird der Skip-ROM- (0xCC) und der Recall-Memory-Befehl (0xB8) ausgeführt. Anschließend muss die gewünschte Page-Nummer übertragen werden. Darauf folgt eine zweite Initialisierungssequenz. Um jetzt die Daten vom Scratchpad zu lesen, muss ein Skip-ROM- (0xCC) und ein Read-Scratchpad-Befehl (0xBE), gefolgt von der Page-Nummer, ausgeführt werden. Danach können die Daten auf der One-Wire-Leitung empfangen werden.

**Von einer Page Daten auslesen:**

Dateiname: DS2438.c

/\*\*

 \* @brief This function reads the data from one page of the DS2438

 \* @param page page number (0 - 7)

 \* @param pageData data of page

 \* @return DS2438\_Status

 \*/

DS2438\_Status DS2438\_ReadPage(uint8\_t page, int16\_t \*pageData)

{

    // reset + presence pulse

  if(DS2438\_Reset() == DS2438\_ERROR)

    return DS2438\_ERROR;

// copy current data to scratchpad

  DS2438\_WriteByte(DS2438\_SKIP\_ROM);

  DS2438\_WriteByte(DS2438\_RECALL\_MEM);

  DS2438\_WriteByte(page);

  // reset + presence pulse

  if(DS2438\_Reset() == DS2438\_ERROR)

    return DS2438\_ERROR;

// read scratchpad data

  DS2438\_WriteByte(DS2438\_SKIP\_ROM);

  DS2438\_WriteByte(DS2438\_READ\_SP);

  DS2438\_WriteByte(page);

  for(int8\_t i = 0; i < 9; i++)

    pageData[i] = DS2438\_ReadByte();

  return DS2438\_OK;

}

#### Initialisierungssequenz

Die Initialisierungssequenz besteht immer aus einen Reset- und einen Presence-Puls, die die folgenden zeitlichen Anforderungen einhalten müssen:

Ein Bild, das Text, Diagramm, Reihe, parallel enthält.

Automatisch generierte Beschreibung

Abbildung 32: One-Wire Initialisierungssequenz

Wenn der One-Wire Slave bei dem Presence-Puls nicht nach 60µs bis 240µs die Leitung in den high-Zustand setzt, besteht ein Problem bei der Kommunikation zwischen Master und Slave.

**Initialisierungssequenz DS2438:**

Dateiname: DS2438.c

/\*\*

 \* @brief This function resets / checks device presence

 \* @return DS2438\_Status

 \*/

DS2438\_Status DS2438\_Reset(void)

{

  // reset DS2438

  // send reset pulse (min 480us)

  HAL\_GPIO\_WritePin(ds2438\_GPIOPort, ds2438\_GPIOPin, GPIO\_PIN\_RESET);

  DS2438\_DelayUs(480);

  // release line -> change to receive mode

  HAL\_GPIO\_WritePin(ds2438\_GPIOPort, ds2438\_GPIOPin, GPIO\_PIN\_SET);

  // wait until slave sends presence pulse

  DS2438\_DelayUs(70);

  // read current pin state

  int8\_t pin = HAL\_GPIO\_ReadPin(ds2438\_GPIOPort, ds2438\_GPIOPin);

  DS2438\_DelayUs(410);

  // check pin state (0 -> found, 1 -> not found)

  if(pin == GPIO\_PIN\_SET)

    return DS2438\_ERROR;

  return DS2438\_OK;

}

#### ROM-Funktionsbefehl

Es gibt viele verschiedene ROM-Funktionsbefehle. Für das Spannungsmonitoring ist nur der SKIP-ROM-Befehl (0xCC) wichtig. Dieser Befehl erlaubt es dem Master direkt einen Memory-Funktionsbefehl zu senden, ohne einen ROM-Code senden zu müssen. Dadurch wird Zeit gespart.

#### Memory-Funktionsbefehl

Der einzige Memory-Funktionsbefehl, der wichtig für das Spannungsmonitoring wichtig ist, ist der CONVERT-V (0xB4) Befehl. Diese Anweisung startet die Analog-Digital-Spannungswandlung des Sensors. Wenn die Umwandlung beendet ist, wird das ADB-Flag (A/D Conversion Busy Flag) vom Sensor auf 0 gesetzt.

### Registerübersicht DS2438

Der DS2438 ist unterschiedliche Register (Pages) unterteilt, die alle notwendigen Einstellungen und Messdaten des Sensors beinhalten.

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

Abbildung 33: Registerübersicht DS2438

Für die Spannungsüberwachung ist nur die Page 0 wichtig, da hier alle Einstellungen und Messwerte gesetzt beziehungsweise ausgelesen werden.

|  |  |
| --- | --- |
| **Byte** | **Verwendungszweck** |
| 0 | Einstellungen setzen |
| 3 und 4 | Nieder- und hochwertiges Byte des Spannung-Messwertes |

### Initialisierung DS2438

Dateiname: DS2438.c

/\*\*

 \* @brief This function initializes the DS2438

 \* @param htim pointer to TIM\_HandleTypeDef (timer for us delay)

 \* @param gpio\_Port GPIOx (port of DQ Pin)

 \* @param gpio\_Pin GPIO\_PIN\_x (pin of DQ Pin)

 \* @return DS2438\_Status

 \*/

DS2438\_Status DS2438\_Init(TIM\_HandleTypeDef \*htim, GPIO\_TypeDef \*gpio\_Port, uint16\_t gpio\_Pin)

{

  if(htim == NULL)

    return DS2438\_ERROR;

  DS2438\_DelayTimer = htim;

  HAL\_TIM\_Base\_Start(DS2438\_DelayTimer); // start timer for DS2438\_DelayUs

  ds2438\_GPIOPort = gpio\_Port;

  ds2438\_GPIOPin = gpio\_Pin;

  if(DS2438\_Reset() == DS2438\_ERROR)

    return DS2438\_ERROR;

// set Vad as A/D converter input

  int16\_t pageData[9] = {0x00};

  if(DS2438\_ReadPage(0x00, pageData) == DS2438\_ERROR)

    return DS2438\_ERROR;

// pageData[0] |= 0x08; // supply voltage

  pageData[0] &= 0xF7;    // external input

  if(DS2438\_WritePage(0x00, pageData) == DS2438\_ERROR)

    return DS2438\_ERROR;

// check current voltage

  int8\_t errorCode = DS2438\_ReadVoltage();

  if(errorCode != DS2438\_OK)

    return errorCode;

  return DS2438\_OK;

}

Der/Die BenutzerIn kann sich aussuchen, ob die Versorgungsspannung oder die Spannung am Eingang vom A/D-Wandler gemessen werden soll. Dies funktioniert mit dem AD-Bit in der Page 0 im Byte[0]. Wenn das AD-Bit auf 1 gesetzt wird, wird die Versorgungsspannung gemessen. Wenn das Bit auf 0 gesetzt wird, misst der Sensor die Eingangsspannung vom A/D-Wandler.

**Beispiel Aufruf vom Programm**:

errorCode = DS2438\_Init(&htim16, GPIOC, GPIO\_PIN\_0);

if(errorCode != DS2438\_OK)

  Sensor\_ErrorHandler(DS2438, errorCode);

### Spannungsüberwachung

Die eingelesen Spannungswerte werden in folgendem Format gespeichert:

Ein Bild, das Text, Schrift, Zahl, Reihe enthält.

Automatisch generierte Beschreibung

Abbildung 34: DS2438 Spannungsregister Format

Um mit den eingelesenen Werten einen Spannungswert zu bilden, müssen MSB und LSB zusammengefügt werden. Da die Werte im Spannungsregisters eine Auflösung von 10mV pro LSB haben, muss der Registerwert mit 10mV multipliziert oder durch 100 gerechnet werden, um den eigentlichen Spannungswert zu bestimmen.

**Spannung einlesen:**

Dateiname: DS2438.c

/\*\*

 \* @brief This function reads the current voltage value of the DS2438

 \* @attention the voltage gets stored in the global variable 'ds2438\_Voltage'

 \* @return DS2438\_Status

 \*/

DS2438\_Status DS2438\_ReadVoltage(void)

{

  // start measurement (send CONVERT T command)

  if(DS2438\_StartVoltageMeasurement() == DS2438\_ERROR)

    return DS2438\_ERROR;

// wait for measurement to be complete (ADB flag: 1 = busy, 0 = ready)

  while(DS2438\_ReadControlVoltageFlag());

  int16\_t pageData[9] = {0x00};

  // read data

  if(DS2438\_ReadPage(0x00, pageData) == DS2438\_ERROR)

    return DS2438\_ERROR;

// extracting voltage bytes

  int16\_t voltageLSB = pageData[3];

  int16\_t voltageMSB = pageData[4];

  ds2438\_Voltage = (((voltageMSB & 0x3) << 8) | (voltageLSB)) / 100.0;

  ds2438\_Voltage \*= 3; // times 3 because of resistor voltage divider

  if(ds2438\_Voltage <= DS2438\_MIN\_VOLTAGE)

      return DS2438\_VOLTAGE\_ERROR;

  return DS2438\_OK;

}

Da der A/D-Wandler des DS2438 nur einen Spannungsbereich von 0-10V hat, muss die Eingangsspannung vor dem Sensor geteilt werden. Auf der Platine des Flight-Controllers ist vor dem Eingang des A/D-Wandlers des DS2438 ein 3:1 Spannungsteiler eingebaut. Daher wird der Spannungswert mit einem Faktor von drei multipliziert, um den eigentlichen Messwert zu bestimmen.

## Motoransteuerung

### DShot Protokoll

Zum Ansteuern der Motoren müssen Befehle an die ESC geschickt werden. Dafür wird das DShot-Protokoll verwendet. Im Vergleich zur analogen PWM-Ansteuerung der ESC, werden die DShot-Signale um ein Vielfaches schneller gesendet und dadurch, dass es sich um ein digitales Protokoll handelt, wird keine Kalibrierungssequenz der Throttle Minimal- und Maximalwerte benötigt.

Das Protokoll besteht aus 16 Bits. Um zwischen 0 und 1 zu unterscheiden, wird der PWM Duty-Cycle des Signals verändert. Das Protokoll gibt es mehreren Geschwindigkeiten (DShot150, DShot300, DShot600, DShot1200), wobei die Zahl angibt, in welcher Frequenz (fDShot in Kilohertz) ein Bit gesendet wird. Der/Die BenutzerIn kann aussuchen mit welcher Geschwindigkeit die Bits gesendet werden soll – *siehe 5.5.2 Initialisierung Motoransteuerung*

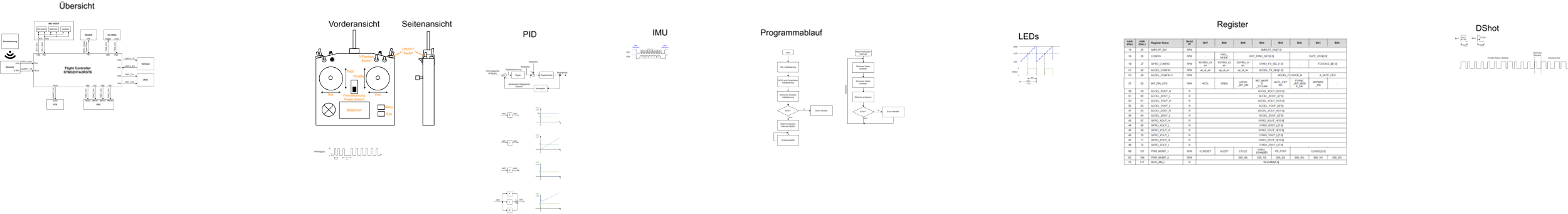


Abbildung 35: DShot Bit 0/1 Duty Cycle + Geschwindigkeit

Die ersten 11 Bits beinhalten den Throttle-Wert in Prozent, wobei 48 = 0% Throttle und 2048 = 100% Throttle. Die Zahlenwerte 0 bis 47 sind für spezielle Befehle reserviert.

Mit dem nächsten Bit kann eine Telemetrie-Anfrage gesendet werden, die bei einer erweiterten Version des Protokolls (bidirektionalen DShot) verwendet werden kann. Die ESC, die für die Diplomarbeit verwendet wird, unterstützt diese Variante nicht.

Die letzten 4 Bits bilden eine Checksumme, die aus der XOR-Verknüpfung der ersten 12 Bits in jeweils Viererblöcken gebildet wird. Die Ruhezeit zwischen Paketen ist im low-Zustand.

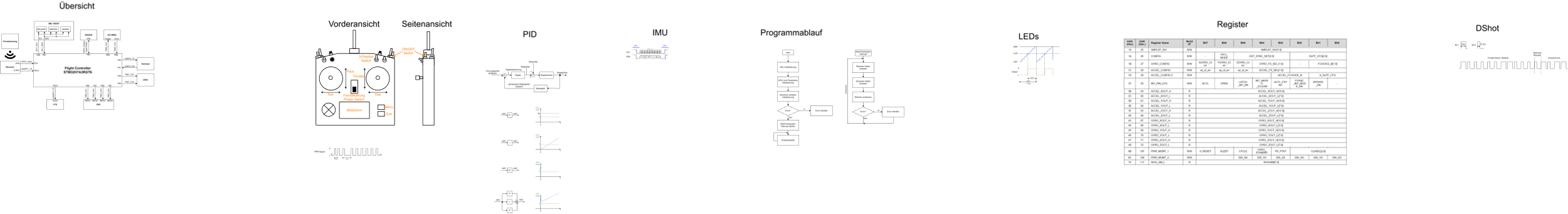


Abbildung 36: DShot Beispiel Übertragung

In der Beispielübertragung wird der Zahlenwert 0b10000011000 (1048) geschickt. Dieser entspricht einen Throttle-Wert von 50%. Die Telemetry Request ist ausgeschaltet und die Berechnung der Checksumme ergibt dem Wert 0b1011, der am Ende des Pakets ausgegeben wird.

### Initialisierung Motoransteuerung

Dateiname: dshot\_own.c

/\*\*

 \* @brief This function initializes the output ESC DShot signal

 \* @param htim pointer to TIM\_HandleTypeDef (output timer)

 \* @param protocol DSHOT150, DSHOT300, DSHOT600

 \* @param updateTim pointer to TIM\_HandleTypeDef (executes 1ms interrupt)

 \* @return DShot\_Status

 \*/

DShot\_Status DShot\_Init(TIM\_HandleTypeDef \*htim, ESC\_OutputProtocol protocol, TIM\_HandleTypeDef \*updateTim)

{

  // set and check timer pointer

  DShot\_OutputTim = htim;

  if(DShot\_OutputTim == NULL)

    return DSHOT\_TIM\_ERROR;

  // set the right timer frequency, 279MHz / (prescaler \* autoreload)

  \_\_HAL\_TIM\_SET\_PRESCALER(DShot\_OutputTim, 1 - 1);

  \_\_HAL\_TIM\_SET\_AUTORELOAD(DShot\_OutputTim, protocol - 1);

  oneDC = ceil((float)protocol \* 0.75);   // calculate duty cycle for sending bit 1

  zeroDC = ceil((float)protocol \* 0.375); // calculate duty cycle for sending bit 0

  // define custom transfer complete ISR

  DShot\_OutputTim->hdma[ESC\_LF\_DMA\_ID]->XferCpltCallback = DShot\_DMA\_XferCpltCallback;

  DShot\_OutputTim->hdma[ESC\_RF\_DMA\_ID]->XferCpltCallback = DShot\_DMA\_XferCpltCallback;

  DShot\_OutputTim->hdma[ESC\_LR\_DMA\_ID]->XferCpltCallback = DShot\_DMA\_XferCpltCallback;

  DShot\_OutputTim->hdma[ESC\_RR\_DMA\_ID]->XferCpltCallback = DShot\_DMA\_XferCpltCallback;

  // set output low

  \_\_HAL\_TIM\_SET\_COMPARE(DShot\_OutputTim, ESC\_LF\_TIM\_CH, 0);

  \_\_HAL\_TIM\_SET\_COMPARE(DShot\_OutputTim, ESC\_RF\_TIM\_CH, 0);

  \_\_HAL\_TIM\_SET\_COMPARE(DShot\_OutputTim, ESC\_LR\_TIM\_CH, 0);

  \_\_HAL\_TIM\_SET\_COMPARE(DShot\_OutputTim, ESC\_RR\_TIM\_CH, 0);

  // start all timers in pwm output mode

  HAL\_TIM\_PWM\_Start(DShot\_OutputTim, ESC\_LF\_TIM\_CH);

  HAL\_TIM\_PWM\_Start(DShot\_OutputTim, ESC\_RF\_TIM\_CH);

  HAL\_TIM\_PWM\_Start(DShot\_OutputTim, ESC\_LR\_TIM\_CH);

  HAL\_TIM\_PWM\_Start(DShot\_OutputTim, ESC\_RR\_TIM\_CH);

  // set custom ISR for 1ms interrupt + start timer

  HAL\_TIM\_RegisterCallback(updateTim, HAL\_TIM\_PERIOD\_ELAPSED\_CB\_ID, DShot\_WriteDataCallback);

  HAL\_TIM\_Base\_Start\_IT(updateTim);

  if(DShot\_SendThrottle(0, 0, 0, 0) != DSHOT\_OK)

    return DSHOT\_TIM\_ERROR;

  HAL\_Delay(5000);

  return DSHOT\_OK;

}

Das Protokoll wird mit der Timer-Peripherie gesendet, welche ein PWM-Signal ausgibt. Damit das DShot-Protokoll erstellt wird, schickt der DMA-Controller die richtigen Werte für die Änderung des Duty-Cycle der Signale.

### Motoransteuerung Software

Dadurch, dass sich Motoren nicht automatisch ausschalten, wird ein DShot-Packet jede Millisekunde gesendet. Dafür wird der updateTim verwendet. Wenn dieser Timer einen Überlauf ergibt, wird ein Interrupt ausgelöst.

Dateiname: dshot\_own.c

/\*\*

 \* @brief This function is the call for a 1ms interrupt, to send every 1ms

 \* @retval None

 \*/

static void DShot\_WriteDataCallback(TIM\_HandleTypeDef \*htim)

{

  static uint16\_t prevThrottle[4] = {1};

  static uint16\_t data[4][18] = {0};

  if(newThrottle[0] != prevThrottle[0] || newThrottle[1] != prevThrottle[1] || newThrottle[2] != prevThrottle[2] || newThrottle[3] != prevThrottle[3])

  {

    DShot\_FormatData(newThrottle, 0, data);

    for(int8\_t i = 0; i < 4; i++)

      prevThrottle[i] = newThrottle[i];

  }

  // start dma transfer to the capture compare register

  HAL\_DMA\_Start\_IT(DShot\_OutputTim->hdma[ESC\_LF\_DMA\_ID], (uint32\_t)&data[0][0], ESC\_TIM\_GET\_CCR\_ADDR(ESC\_LF\_TIM\_CH), 18);

  HAL\_DMA\_Start\_IT(DShot\_OutputTim->hdma[ESC\_RF\_DMA\_ID], (uint32\_t)&data[1][0], ESC\_TIM\_GET\_CCR\_ADDR(ESC\_RF\_TIM\_CH), 18);

  HAL\_DMA\_Start\_IT(DShot\_OutputTim->hdma[ESC\_LR\_DMA\_ID], (uint32\_t)&data[2][0], ESC\_TIM\_GET\_CCR\_ADDR(ESC\_LR\_TIM\_CH), 18);

  HAL\_DMA\_Start\_IT(DShot\_OutputTim->hdma[ESC\_RR\_DMA\_ID], (uint32\_t)&data[3][0], ESC\_TIM\_GET\_CCR\_ADDR(ESC\_RR\_TIM\_CH), 18);

  // reset counter to get rid of delay between channels

  \_\_HAL\_TIM\_SET\_COUNTER(DShot\_OutputTim, 0);

  // enable dma

  \_\_HAL\_TIM\_ENABLE\_DMA(DShot\_OutputTim, TIM\_DMA\_CC1);

  \_\_HAL\_TIM\_ENABLE\_DMA(DShot\_OutputTim, TIM\_DMA\_CC2);

  \_\_HAL\_TIM\_ENABLE\_DMA(DShot\_OutputTim, TIM\_DMA\_CC3);

  \_\_HAL\_TIM\_ENABLE\_DMA(DShot\_OutputTim, TIM\_DMA\_CC4);

}

Diese Funktion bereitet die Daten für die Übertragung vor und startet den DMA-Controller.

Wenn die Datenübertragung des DMA-Controllers abgeschlossen hat, wird ein Interrupt ausgelöst.

Dateiname: dshot\_own.c

/\*\*

 \* @brief This function is the ISR for DMA transmit complete

 \* @param hdma

 \* @retval None

 \*/

static void DShot\_DMA\_XferCpltCallback(DMA\_HandleTypeDef \*hdma)

{

  // diable DMA to get rid of the delay between channels

  if(hdma == DShot\_OutputTim->hdma[TIM\_DMA\_ID\_CC1])

  {

    \_\_HAL\_TIM\_DISABLE\_DMA(DShot\_OutputTim, TIM\_DMA\_CC1);

  }

  else if(hdma == DShot\_OutputTim->hdma[TIM\_DMA\_ID\_CC2])

  {

    \_\_HAL\_TIM\_DISABLE\_DMA(DShot\_OutputTim, TIM\_DMA\_CC2);

  }

  else if(hdma == DShot\_OutputTim->hdma[TIM\_DMA\_ID\_CC3])

  {

    \_\_HAL\_TIM\_DISABLE\_DMA(DShot\_OutputTim, TIM\_DMA\_CC3);

  }

  else if(hdma == DShot\_OutputTim->hdma[TIM\_DMA\_ID\_CC4])

  {

    \_\_HAL\_TIM\_DISABLE\_DMA(DShot\_OutputTim, TIM\_DMA\_CC4);

  }

}

Diese Funktion schaltet den DMA-Controller für alle Kanäle ab. Dies ermöglicht, dass alle Pakete zur selbe Zeit gesendet werden.

Um den Throttlewert der Motoren zu ändern, muss folgende Funktion ausgeführt werden:

Dateiname: dshot\_own.c

/\*\*

 \* @brief This function formats the motor data for the DShot protocol

 \* @param motorLF percent of throttle value of left front motor (0-100)

 \* @param motorRF percent of throttle value of right front motor (0-100)

 \* @param motorLR percent of throttle value of left rear motor (0-100)

 \* @param motorRR percent of throttle value of right rear motor (0-100)

 \* @retval DShot\_Status

 \*/

DShot\_Status DShot\_SendThrottle(double motorLF, double motorRF, double motorLR, double motorRR)

{

  if(DShot\_OutputTim == NULL)

    return DSHOT\_TIM\_ERROR;

  /\*\*

   \* timer channel 1 = left front motor

   \* timer channel 2 = right front motor

   \* timer channel 3 = left rear motor

   \* timer channel 4 = right rear motor

   \*/

   // convert to dshot throttle format (48 = 0% throttle, 2047 = 100% throttle)

  newThrottle[0] = 48 + 20 \* motorLF;

  newThrottle[1] = 48 + 20 \* motorRF;

  newThrottle[2] = 48 + 20 \* motorLR;

  newThrottle[3] = 48 + 20 \* motorRR;

  return DSHOT\_OK;

}

Diese Funktion speichert die neuen Throttlewerte in der Variable *newThrottle*, die in dem *DShot\_WriteDataCallback()* auf Änderung überprüft wird.

Um diese Daten in das richtige Format zu ändern, wird die Funktion *DShot\_FormatData()* ausgeführt.

Dateiname: dshot\_own.c

/\*\*

 \* @brief This function formats and sends DShot data via PWM DMA

 \* @param throttle Throttle values (0-2047)

 \* @param telemetry telemetry request bit

 \* @param data formatted data by the function

 \* @retval None

 \*/

void DShot\_FormatData(uint16\_t \*throttle, int8\_t telemetry, uint16\_t data[4][18])

{

  uint16\_t withoutCS, complete, div;

  // format the data to packets

  for(int8\_t i = 0; i < 4; i++)

  {

    // first 12 bits (without Checksum)

    withoutCS = (throttle[i] << 1) | telemetry;

    // format whole data frame

    complete = withoutCS << 4 | ((withoutCS ^ (withoutCS >> 4) ^ (withoutCS >> 8)) & 0x0F);

    // convert each bit to the specific duty cycle length

    div = 0x8000;

    for(int8\_t j = 0; j < 16; j++)

    {

      data[i][j] = (complete & div) ? oneDC : zeroDC;

      div >>= 1;

    }

  }

}

Die Datenpakete werden zusammengestellt und danach in den richtigen Duty-Cycle-Wert umgewandelt.

## PID-Regler

### Initialisierung PID-Regler

Damit die Filterkoeffizienten während der Programmlaufzeit geändert werden können, wird der DMA-Controller verwendet. Dafür wird die UART4-Peripherie, die auch für die für Terminal Ausgabe [*(siehe: 5.7.1 Terminal Ausgabe)*](#_Terminal_Ausgabe) verwendet wird.

Ein Bild, das Text, Screenshot, Software, Zahl enthält.

Automatisch generierte Beschreibung

Abbildung 37: STM32CubeMX Einstellung PID DMA Empfang

Zum Initialisieren wird die Funktion *PID\_Init()* ausgeführt.

Dateiname: PID.c

/\*\*

 \* @brief This function initializes the change Ks system

 \* @param huart pointer to UART\_HandleTypeDef (input uart)

 \* @retval None

 \*/

void PID\_Init(UART\_HandleTypeDef \*huart)

{

  // start reception with uart

  HAL\_UART\_RegisterCallback(huart, HAL\_UART\_RX\_COMPLETE\_CB\_ID, changeKs);

  HAL\_UART\_Receive\_DMA(huart, (uint8\_t \*)&receiveData, 7);

}

Diese Funktion startet den DMA-Controller, der auf 7 Bytes von der UART-Peripherie wartet und in dem Array *receiveData[]* speichert. Dadurch können Werte im Format „0 0.123“ gesendet werden. Die erste Zahl gibt den Array-Index von dem globalen Array *allKs[9]* und die restliche Zahl den Wert der Stelle an.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Index** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| **Wert** | KP Roll | KI Roll | KD Roll | KP Pitch | KI Pitch | KD Pitch | KP Yaw | KI Yaw | KD Yaw |

Nachdem ein Datensatz eingelesen worden ist, wird ein Interrupt mit der Funktion *changeKs()* ausgeführt.

Dateiname: PID.c

/\*\*

 \* @brief This function changes a PID controller coefficients via uart

 \* @param huart

 \*/

void changeKs(UART\_HandleTypeDef \*huart)

{

  int index = atoi(&receiveData[0]);

  char value[6];

  sprintf(value, "%c%c%c%c%c", receiveData[2], receiveData[3], receiveData[4], receiveData[5], receiveData[6]);

  float tmp = atof(value);

  if(tmp > 5 || tmp < 0)

  {

    Terminal\_Print("value out of range\n\r");

    return;

  }

  if(index < 0 || index > 8)

  {

    Terminal\_Print("index out of range\n\r");

    return;

  }

  allKs[index] = tmp;

  sprintf(txt, "[%d] = %f\n\r", index, allKs[index]);

  Terminal\_Print(txt);

}

Diese Funktion wandelt die Daten in Zahlen um und schreibt die Wert in den richtigen Index im Array *allKs[]*.

Danach wird ein Bestätigungstext mit dem Arrayindex und Zahlenwert ausgegeben. Wenn dieser Text nicht zurückgesendet wird, gab es ein Problem bei der Datenübertragung.

Wenn der Index oder die Zahl außerhalb des verwendetet Zahlenbereichs ist, wird der Text „out of range“ ausgegeben.

### PID-Algorithmus

Dateiname: PID.c

PID\_Status PID\_Normal(float inputThrottle, float inputPitch, float inputRoll, float inputYaw)

{

  // get current angles + delta time

  IMU\_GetAngles();

  // calculate PID roll output

  double errorRoll = inputRoll - angle.roll;

  static double I\_Roll = 0, errorRollPrev = 0;

  I\_Roll += (errorRoll \* imu\_DeltaTime) \* allKs[1];

  if(I\_Roll > PID\_MAX\_TURN) I\_Roll = PID\_MAX\_TURN;

  else if(I\_Roll < -PID\_MAX\_TURN) I\_Roll = -PID\_MAX\_TURN;

  double rollOutput = allKs[0] \* errorRoll + I\_Roll + allKs[2] \* ((errorRoll - errorRollPrev) / imu\_DeltaTime);

  if(rollOutput > PID\_MAX\_TURN) rollOutput = PID\_MAX\_TURN;

  else if(rollOutput < -PID\_MAX\_TURN) rollOutput = -PID\_MAX\_TURN;

  errorRollPrev = errorRoll;

  // calculate PID pitch output

  double errorPitch = inputPitch - angle.pitch;

  static double I\_Pitch = 0, errorPitchPrev = 0;

  I\_Pitch += (errorPitch \* imu\_DeltaTime) \* allKs[4];

  if(I\_Pitch > PID\_MAX\_TURN) I\_Pitch = PID\_MAX\_TURN;

  else if(I\_Pitch < -PID\_MAX\_TURN) I\_Pitch = -PID\_MAX\_TURN;

  double pitchOutput = allKs[3] \* errorPitch + I\_Pitch + allKs[5] \* ((errorPitch - errorPitchPrev) / imu\_DeltaTime);

  if(pitchOutput > PID\_MAX\_TURN) pitchOutput = PID\_MAX\_TURN;

  else if(pitchOutput < -PID\_MAX\_TURN) pitchOutput = -PID\_MAX\_TURN;

  errorPitchPrev = errorPitch;

  // calculate PID yaw output

  double errorYaw = inputYaw - angle.yaw;

  static double I\_Yaw = 0, errorYawPrev = 0;

  I\_Yaw += (errorYaw \* imu\_DeltaTime) \* allKs[7];

  if(I\_Yaw > PID\_MAX\_TURN) I\_Yaw = PID\_MAX\_TURN;

  else if(I\_Yaw < -PID\_MAX\_TURN) I\_Yaw = -PID\_MAX\_TURN;

  double yawOutput = allKs[6] \* errorYaw + I\_Yaw + allKs[8] \* ((errorYaw - errorYawPrev) / imu\_DeltaTime);

  if(yawOutput > PID\_MAX\_TURN) yawOutput = PID\_MAX\_TURN;

  else if(yawOutput < -PID\_MAX\_TURN) yawOutput = -PID\_MAX\_TURN;

  errorYawPrev = errorYaw;

  // check max/min values + output

  float motorLF = (inputThrottle < 5) ? 0 : inputThrottle - pitchOutput - rollOutput + yawOutput;

  float motorRF = (inputThrottle < 5) ? 0 : inputThrottle - pitchOutput + rollOutput - yawOutput;

  float motorLR = (inputThrottle < 5) ? 0 : inputThrottle + pitchOutput - rollOutput - yawOutput;

  float motorRR = (inputThrottle < 5) ? 0 : inputThrottle + pitchOutput + rollOutput + yawOutput;

  if(motorLF < 5) motorLF = 0;

  if(motorRF < 5) motorRF = 0;

  if(motorLR < 5) motorLR = 0;

  if(motorRR < 5) motorRR = 0;

  if(motorLF > inputThrottle + PID\_MAX\_TURN) motorLF = inputThrottle + PID\_MAX\_TURN;

  if(motorRF > inputThrottle + PID\_MAX\_TURN) motorRF = inputThrottle + PID\_MAX\_TURN;

  if(motorLR > inputThrottle + PID\_MAX\_TURN) motorLR = inputThrottle + PID\_MAX\_TURN;

  if(motorRR > inputThrottle + PID\_MAX\_TURN) motorRR = inputThrottle + PID\_MAX\_TURN;

  DShot\_SendThrottle(motorLF, motorRF, motorLR, motorRR);

  return PID\_OK;

}

Der gesamte PID-Regler besteht aus drei PID-Reglern. Jeweils einer für die Pitch-, Roll- und Yaw-Achse. Es wird mit dem IMU-Lagewinkel und den übergebenen Variablen die Regelabweichungen gebildet, die dann weiter zu Throttlewerten umgerechnet werden.

## Debugging

### Terminal Ausgabe

Die Terminal Ausgabe ist essenziell für die Programmkontrolle, da der aktuelle Inhalt von Variablen direkt ausgegeben werden kann. Im Vergleich zum Debuggen wird das Programm dabei nur kurz unterbrochen, was bei Interrupt gesteuerten System zu Problemen führen kann.

Die Kommunikation findet mithilfe der UART-Peripherie statt. Diese ist mit einer Zusatzplatine verbunden, um den Datenaustausch mit einem externen System zu ermöglichen *(siehe Kapitel Marcel)*.

**UART-Einstellungen in STM32CubeMX:**

Ein Bild, das Text, Screenshot, Software, Website enthält.

Automatisch generierte Beschreibung

Abbildung 38: STM32CubeMX Einstellungen Terminal

Für den Empfang der Daten wird die Erweiterung „Serial Monitor“ [SERM] für Visual Studio Code verwendet.

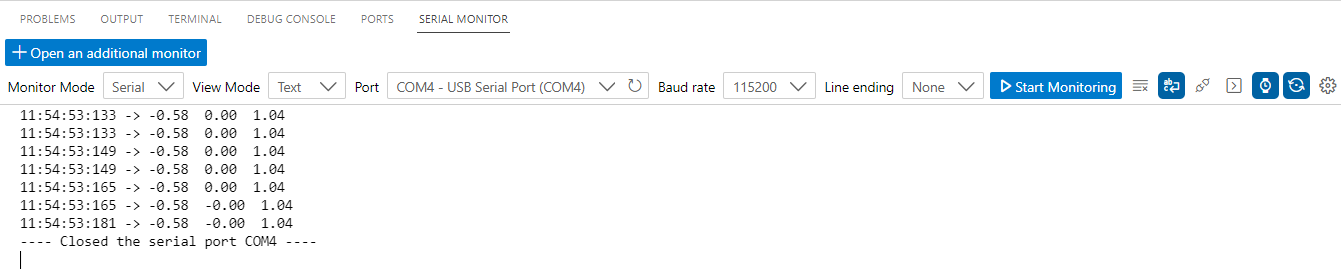


Abbildung 39: Terminal Bild

Wichtig:

Die Baudrate, die für die Datenübertragung verwendet wird, ist 115200 Bits/s.

**Text zum Terminal senden:**

Dateiname: status\_handling.c

/\*\*

 \* @brief This function prints a string to the terminal

 \* @param string

 \* @retval none

 \*/

void Terminal\_Print(char \*string)

{

  HAL\_UART\_Transmit(&huart4, (uint8\_t \*)string, strlen(string), HAL\_MAX\_DELAY);

}

**Beispiel Programmaufruf:**

// with only text

Terminal\_Print("wichtige Daten\n\r");

// with variable

char text[] = "wichtige Daten\n\r";

Terminal\_Print(text);

### Status – LEDs

Auf der Flight-Controller Platine befinden sich eine rote und eine blaue LED. Die blaue wird zum Initialisierungsstatus und die rote LED wird für zur Anzeige von Errors verwendet.

|  |  |  |
| --- | --- | --- |
| **Blaue LED** | leuchtet nicht | Problem bei Programmstart |
| blinkt alle 0,5 Sekunden | DS2438 initialisiert |
| blinkt alle 0,4 Sekunden | IMU initialisiert |
| blinkt alle 0,3 Sekunden | PID initialisiert |
| blinkt alle 0,2 Sekunden | DShot initialisiert |
| blinkt alle 0,1 Sekunden | Receiver initialisiert |
| leuchtet durchgehend | Initialisierungen abgeschlossen |
| **Rote LED** | leuchtet nicht | kein Error |
| leuchtet durchgehend | Error während Initialisierungen |
| blinkt alle 0,1 Sekunden | Failsafe von Receiver oder DS2438 ausgelöst |

**Einstellungen STM32CubeMX:**

**Ein Bild, das Text, Screenshot, Software, Computersymbol enthält.

Automatisch generierte Beschreibung**

Abbildung 40: STM32CubeMX Einstellungen LEDs

Für die Änderung der Frequenz des PWM-Signals wird folgende Formel verwendet:

fBlink … Ausgangsfrequenz

fTimer … Timer Eingangsfrequenz, für Diplomarbeit 279MHz

PSC … prescaler Registerwert

ARR … auto-reload Registerwert, für Diplomarbeit fixiert auf 10000-1

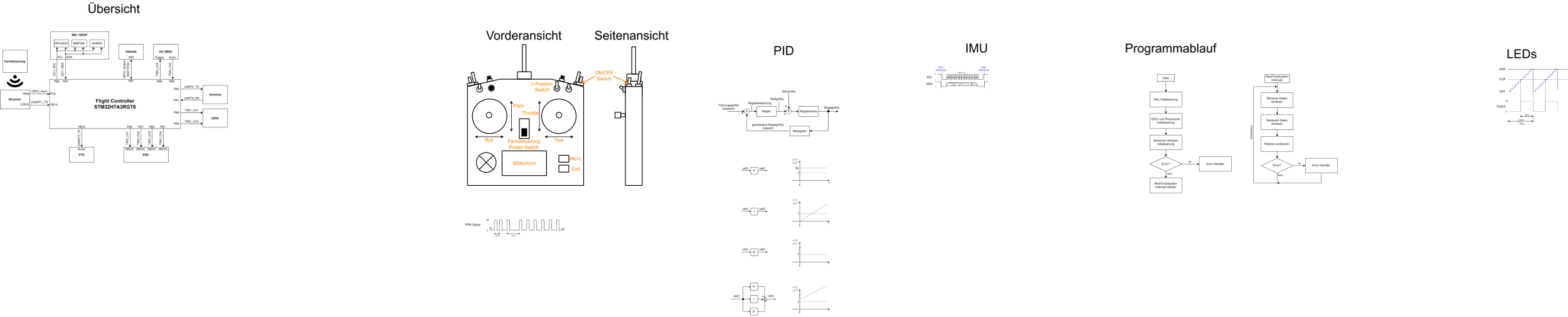


Abbildung 41: Beispiel PWM-Signalverlauf

Wenn der Wert im Counter-Register (CNT) unter dem Wert im capture/compare Register (CCR) ist, wird ein low-Zustand ausgegeben. Wenn der CNT-Wert größer ist als der CCR-Wert ist, dann wird ein high-Zustand ausgegeben. Durch die Änderung des CCR-Wertes kann der Duty Cycle verändert werden.

Für die Ansteuerung der LEDs wird ein Duty-Cycle von 50% verwendet. Dafür muss der Wert im CCR-Register die Hälfte des Wertes im ARR-Register gesetzt werden – *siehe Abbildung 46*

Um die Werte im während der Programmlaufzeit ändern zu können, können folgende Funktionen verwendet werden:

// set duty cycle for LEDs

\_\_HAL\_TIM\_SET\_COMPARE(&htim1, TIM\_CHANNEL\_1, 0);

\_\_HAL\_TIM\_SET\_COMPARE(&htim1, TIM\_CHANNEL\_2, 5000);

// set frequency ~2Hz

\_\_HAL\_TIM\_SET\_PRESCALER(&htim1, 14000 - 1);

// start timer for LEDs

HAL\_TIM\_PWM\_Start(&htim1, TIM\_CHANNEL\_1);

HAL\_TIM\_PWM\_Start(&htim1, TIM\_CHANNEL\_2);

Hinweis:

Die Funktionen erlauben es auch die Registerwerte zu ändern, ohne die Signal-Ausgabe unterbrechen zu müssen.

# Datenübertragung der Mess- und Videodaten (Ben)

## Überblick Datenübertragung

Blockschaltbild und schaltung

## Kommunikation: Flight Controller und Sender

### *Aufbau Sendermodul (VTx)*

### ASK – Modulation

Überblick und wieso nicht verwendet

### Transmit – Programm auf Cortex µC

#### Gleitkommadarstellung (memcpy)

#### Code

### Testen der Übertragung

+ probleme

## Kommunikation: Sender und Empfänger

### Aufbau Empfängermodul

### Verbindungsaufbau

### Testen der Übertragung

## Kommunikation: Empfänger und Raspberry Pi

### Empfangsprogramm auf Raspberry Pi

#### UART auf dem Raspberry Pi

#### Code

### Senden der Daten auf dem MQTT-Server

## Testen der Datenübertragungskette

# Visualisierungs-App

## Applikation

### Einführung – Dart / Flutter - Framework

#### Allgemeines

#### Pub Dev / Package Installer

#### Dart Syntax

#### State Management

##### Stateful vs. Stateless Widgets

##### Provider

### Allgemeines zur App

#### Usersystem

#### Datenvisualiserung

#### 3D-Model-Viewer

#### Livestream-Viewer

### UI-Konzept

### Projektstruktur und -umgebung

#### Editor – Visual Studio Code

#### Flutter Installation

#### Projekterstellung

#### Projektstruktur

#### Packages

##### Pubspec.yaml

##### Installieren neuer Packages

### Splash + Willkommensscreen

#### Native Splash Screen

#### Willkommensbildschirm

#### Logo-Design

### Login + Registrierung

#### E-Mail-Login

#### Google-Login

#### Passwort vergessen

#### Registrierung

### Homepage

#### Bottom Navigation Bar / GNav-Bar

#### Serverdatendialog

#### Flugdatenvisualisierung

##### MQTT-Datenstreams

##### Syncfusion Cartesian Charts

##### Echtzeitdaten + Animierte Diagramme

##### Speicherung der Daten

#### 3D-Model-Viewer

##### Flutter Cube

##### MQTT-Datenstream

##### 3D-Dronenmodell

###### Optimierungen in Blender

##### Extrabedienungen

#### Live-View

##### VLC-Plugin

##### Darstellung eines Netzwerkstreams

##### Video-Overlay

### Sidemenu / Drawer

#### Userprofil

##### Darstellung der Userdaten

##### Profilbildauswahl

##### Änderung der Userdaten

#### Credits

#### Vorherige Flüge

##### Darstellung aller gespeicherten Flüge

##### Sortiermöglichkeiten

##### Einsicht in vorherigen Flug

#### Einstellungen

##### Dark- / Light-Mode

##### 3D-Model Ausrichtung

#### Logout´

## Firebase Backend

### Installation via Firebase CLI

##### Firebase CLI – Setup

##### Firebase für Dart / Flutter aktivieren

##### Flutter App mit Flutterfire konfigurieren

### Einbindung in Flutter

##### Flutterfire Configure

##### Benutzten Firebase Packages hinzufügen

### Nutzen der verschiedenen Datenbanksysteme

#### Auth

##### E-Mail + Passwort – Login

##### Google Login

#### Firestore Database

##### Struktur

##### Allgemeine Userdaten

##### Flugdaten

##### Einstellungen

#### Realtime Database

##### Allgemein

##### Sinn der verschiedenen Flags

#### Storage

##### Allgemein

##### Referenz zu User in Firestore

# Videostreaming

## Allgemeiner Aufbau

### Anforderungen

### Übertragungskette

## CADFPX Analog Kamera

### Allgemein

### Produktinformationen

### Verwendungszweck

### Verbindungstest via USB-Camera App

## USB2.0 VHS Video Grabber

## RTMP-Server via NGINX aufsetzen

### RTMP Allgemein

### NGINX

#### Allgemein

#### Installation

#### RTMP-Konfiguration

#### Serverstatus einsehen

## RTMP-Stream erstellen

### FFMPEG Allgemein

### Aufbau der Konvertierung

#### Simple Konvertierung

#### Optimierungen

##### Encodervergleich

##### Optimierungsflags

##### Unoptimiert vs. Optimiert

# Videoserver-Port Forwarding

## Allgemein

## Routereinstellungen

## Kontakt mit Routerfirma zur Freischaltung

## Testen der Ports

### Portchecker.co

## Testen des Videostreams

### VLC-Player

#### Devicestream direkt testen

#### Netzwerkstream im selben Netzwerk

#### Netzwerkstream via Public IP

# Ergebnisse

Entweder Gesamt System irgendwelche Bilder

Oder wenn nicht, dann einzelne Teile die funktionieren zeigen / beschreiben

Das Kapitel ist, um die Ergebnisse zu verkaufen, basically alles was geht

Name könnte sich noch ändern

# Anhang

## Einführung CAD – Software (Fusion 360) (Ben)

### UI und Projekterstellung

### Skizze anfertigen

### Körper erstellen

### Schrift und Bilder einfügen

## 3D – Druck (Ben)

### 3D – Drucker

### Filamente

### 3D – Drucker Software (Ultimaker Cura)

## Inbetriebnahme Anleitung

## Projektplan

## Projektkosten

## Projekttagebuch

## Einführung Entwicklungsumgebung - Steuerungssoftware

Um den Mikroprozessor STM32H7A3RGT6 auf der Flight-Controller Platine zu programmieren, wird die Entwicklungsumgebung Keil µVision5 in der Version V5.38.0.0 in Kombination mit STM32CubeMX in der Version 6.10.0 und Vision Studio Code verwendet. Programmiert wird dieser mithilfe eines DAPLink (CMSIS-DAP) – Interface über die SWD-Schnittstelle.

Mit STM32CubeMX können die Grundeinstellungen des Mikrocontrollers, wie die Peripherie- und Taktversorgungseinstellungen, mit einer grafischen Oberfläche einfach getätigt und automatisch eine Initialisierungssoftware in der Programmiersprache C generiert werden. Diese Software wird mithilfe von HAL (hardware abstraction layer) erstellt. Dieses System bietet eine Menge APIs, die es ermöglichen mit einfachen Funktionen komplexe Befehle und Einstellungen in einen STM32 Mikrocontroller zu tätigen.

Keil µVision5 bietet eine integrierte Entwicklungsumgebung mit eingebautem Assembler, Compiler und Debugger. Es gibt eine beschränkte kostenlose Version im Internet, die aber nicht für die Programmgröße der Diplomarbeit ausreicht. Daher muss eine kostenlose Community-Lizenz von Keil aktiviert werden [KLIZ].

Für die Programmierung des C-Sourcecodes wird Visual Studio Code verwendet. Mit der Erweiterung „Keil Assistent“ [KASS] kann Keil µVision5 direkt von einer Benutzeroberfläche gesteuert werden. Visual Studio Code bietet eine Vielfalt von Funktionen, die den Arbeitsablauf produktiver und einfacher gestalten.

**STM32CubeMX Projekt erstellen:**

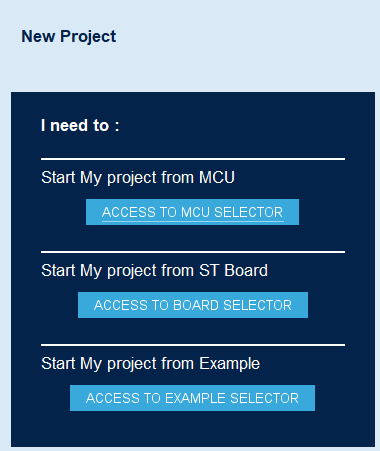


Abbildung 42: STM32CubeMX access to MCU selector

Nach dem Starten von der Software muss der zu programmierende Mikrocontroller ausgewählt werden.

Bei vorgefertigten Entwicklungsplatinen, kann „access to board selector“ oder „access to example selector“ verwendet werden.

Für die Diplomarbeit wird eine eigenerstellte Platine mit Mikrocontroller verwendet, daher wird die Option „access to MCU selector“ verwendet.

Nach der Auswahl werden die aktuellen Informationen der Mikrocontroller und Entwicklungsplatinen von der STM32 Datenbank runtergeladen.

Ein Bild, das Text, Screenshot, Zahl, Schrift enthält.

Automatisch generierte Beschreibung

Abbildung 43: STM32CubeMX Liste von Mikrocontroller

Mit einer Menge von Filtern und Suchoptionen kann der gewünschte Mikrocontroller ausgewählt werden. Bevor das Projekt erstellt wird, können die Eigenschaften der Auswahl kontrolliert. Um das Projekt zu erstellen, muss der Knopf „Start Project“ gedrückt werden.

Ein Bild, das Text, Screenshot, Schrift, Software enthält.

Automatisch generierte Beschreibung

Abbildung 44: STM32CubeMX Auswahl Mikrocontroller

Mit der grafischen Oberfläche können die gewünschten Peripherie-, Pin-, Takt- und Projekteinstellungen getätigt werden.

Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

Automatisch generierte Beschreibung

Abbildung 45: STM32CubeMX grafische Oberfläche

Um die effizienteste Programmstruktur für die Diplomarbeit zu erstellen, müssen wichtige Eistellungen getätigt werden.

Als Erstes muss die Taktquelle ausgewählt werden. Da externe Quarzoszillatoren verwendet werden, muss in der Kategorie RCC für „High Speed Clock (HSE)“ und „Low Speed Clock (LSE)“ die Einstellung „Crystal/Ceramic Resonator“ ausgewählt werden.

Ein Bild, das Text, Software, Computersymbol, Webseite enthält.

Automatisch generierte Beschreibung

Abbildung 46: STM32CubeMX HSE/LSE Einstellung

Mit der „Clock Configuration“ kann die gesamte Taktstruktur nach Anforderungen verändert werden:

Ein Bild, das Diagramm, Text, Reihe, Plan enthält.

Automatisch generierte Beschreibung

Abbildung 47: STM32CubeMX clock configuration

Wichtig: Wenn externe Oszillatoren verwendet werden, muss auf der linken Seite, die Einstellung „Input frequency“ auf die vorhandene Frequenz gesetzt werden.

Für die Diplomarbeit wird ein 8MHz Quarz Oszillator, der auf eine Systemfrequenz von 279MHz erhöht wird, verwendet.

Um den genierten Code mit der Keil µVision5 Entwicklungsumgebung verwenden zu können, müssen unter Project Manager gewisse Einstellungen getätigt werden:

Ein Bild, das Text, Screenshot, Reihe, Zahl enthält.

Automatisch generierte Beschreibung

Abbildung 48: STM32CubeMX project settings

Eine für die Drohne notwendige Einstellung ist unter Project Manager 🡪 Advanced Settings 🡪 Register Callback. In diesen Bereich können für einzelne Peripherien eigenerstellte Callback-Funktionen aktiviert werden.

Für die FPV-Drohne ist die Einstellung für die TIM-, UART- und USART-Peripherie aktiviert werden.



Abbildung 49: STM32CubeMX register callback

Eine Callback-Funktion wird beim Auslösen eines Interrupts von der ISR (Interrupt Service Routine) ausgelöst.

Zum Beispiel: *void HAL\_TIM\_PeriodElapsedCallback (TIM\_HandleTypeDef \* htim)* wird jedes Mal ausgeführt, wenn eine Timer-Peripherie einen Überlauf-Interrupt auslöst. Diese Funktion ist in der Datei *stm32h7xx\_hal\_tim.c* mit dem Präfix *\_\_weak* definiert. Das bedeutet, wenn die Funktion in einer anderen Datei ausprogrammiert wird, wird diese mit der neuen Version ersetzt.

Die Einstellung von Register Callback erlaubt es die Funktion mit einer eigenen zu ersetzen. Dafür muss die Funktion *HAL\_TIM\_RegisterCallback()* aufgerufen werden.

Zum Beispiel:

|  |
| --- |
| HAL\_TIM\_RegisterCallback(&htim15, HAL\_TIM\_PERIOD\_ELAPSED\_CB\_ID, RealTimeSystemCallback); |

Von der Timer15-Peripherie wird das Überlauf-Interrupt-Callback mit der Funktion *RealTimeSystemCallback()* ersetzt.

Nach der fertigen Einstellung des Projekts, können die Programmdateien mit dem Knopf „Generate Code“ erstellt werden. Daraufhin wird folgende Ordnerstruktur erstellt:

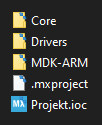


Abbildung 50: STM32CubeMX Ordnerstruktur

* Core: alle Programm- und Headerdateien
* Drivers: alle HAL-Dateien, die für das Projekt notwendig sind
* MDK-ARM: Keil µVision5 Projektdateien

**Keil µVision5 Einstellungen:**

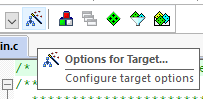


Abbildung 51: µVision Zauberstab

Die Einstellungen des Projekts können mit dem Zauberstab „Options for Target…“ eingestellt werden.

Die wichtigsten Einstellungen sind die Target- und Debug-Einstellungen:

Ein Bild, das Text, Screenshot, Display, Zahl enthält.

Automatisch generierte Beschreibung

Abbildung 52: µVision Target Einstellungen

Ein Bild, das Text, Screenshot, Zahl, Display enthält.

Automatisch generierte Beschreibung

Abbildung 53: µVision Debug Einstellungen

Für die Diplomarbeit wird der default compiler version 6 und der CMSUS-Dap Debugger verwendet.

Das von STM32CubeMX erstellte Programm, muss nicht geändert werden und eigener Code soll zwischen den Kommentaren „USER CODE BEGIN“ und „USER CODE END“ eingefügt werden, damit wenn neuer Code generiert wird, der eigene nicht überschrieben wird.

Um das Programm auf den Mikrocontroller zu spielen, müssen folgende Knöpfe verwendet werden:



Abbildung 54: µVision Build/Flash Knöpfe

# Quellen

## Gedruckte Medien

Für die Quellenangabe bei Fachbüchern ist, wie nachfolgend dargestellt, vorzugehen (löschen):

Vorname Nachname: Titel. Untertitel. - Verlagsort: Verlag, Jahr.

Vorname Nachname: Titel. Untertitel. Auflage - Verlagsort: Verlag, Jahr.

z.B.:

[REI13] Jürgen Reichhardt: Lehrbuch Digitaltechnik - Eine Einführung mit VHDL  
 3. Auflage, München: Oldenbourg Verlag, 2013  
 ISBN: 978-3-486-72765-4

## Online

|  |  |
| --- | --- |
| [KLIZ] | ARM Keil: Community Lizenz Anleitung  <https://www.keil.arm.com/mdk-community/>  (letzter Aufruf: 28.02.2024) |
| [KASS] | Visual Studio Marketplace: Keil Assistant Erweiterung  <https://marketplace.visualstudio.com/items?itemName=CL.keil-assistant>  (letzter Aufruf: 28.02.2024) |
| [SERM] | Visual Studio Marketplace: Serial Monitor Erweiterung  <https://marketplace.visualstudio.com/items?itemName=ms-vscode.vscode-serial-monitor>  (letzter Aufruf: 28.02.2024) |
| [SBDW] | digitalwire: Futaba S-Bus Protokoll  <https://digitalwire.ch/de/projekte/futaba-sbus/>  (letzter Aufruf: 28.02.2024) |
| [IBDSP] | The FlySky iBus protocol  <https://blog.dsp.id.au/posts/2017/10/22/flysky-ibus-protocol/>  (letzter Aufruf: 28.02.2024) |
| [IBGH] | STM32 HAL iBUS  <https://github.com/mokhwasomssi/stm32_hal_ibus>  (letzter Aufruf: 28.02.2024) |
| [SBGH] | STM32 RadioLink SBUS DMA  <https://github.com/osos11-Git/STM32_RadioLink_SBUS_DMA/tree/main>  (letzter Aufruf: 02.03.2024) |
| [DSBW] | Brushless Whoop: DSHOT – the missing handbook  <https://brushlesswhoop.com/dshot-and-bidirectional-dshot/>  (letzter Aufruf: 28.02.2024) |
| [DSBF] | Betaflight: DSHOT  <https://betaflight.com/docs/development/Dshot>  (letzter Aufruf: 02.03.2024) |
| [DSGH] | stm32 hal dshot  <https://github.com/mokhwasomssi/stm32_hal_dshot/tree/main>  (letzter Aufruf: 02.03.2024) |
| [IMUWS] | Waveshare: 10 DOF IMU Sensor  <https://www.waveshare.com/wiki/10_DOF_IMU_Sensor_(C)>  (letzter Aufruf: 28.02.2024) |
| [M7] | STM32H7A3RGT6 Datenblätter  <https://www.st.com/en/microcontrollers-microprocessors/stm32h7a3rg.html>  (letzter Aufruf: 28.02.2024) |
| [M7HAL] | STM32H7 HAL Datenblätter  <https://www.st.com/en/embedded-software/stm32cubeh7.html#documentation>  (letzter Aufruf: 28.02.2024) |
| [DS2438] | DS2438 Datenblatt  <https://www.analog.com/media/en/technical-documentation/data-sheets/DS2438.pdf>  (letzter Aufruf: 28.02.2024) |
| [COFIL] | Complementary filter and relative orientation with MPU9250  <https://www.hackster.io/hibit/complementary-filter-and-relative-orientation-with-mpu9250-d4f79d>  (letzter Aufruf: 28.02.2024) |

# Verzeichnis der Abbildungen

[Abbildung 1: Regler-Blockschaltbild 21](#_Toc160296701)

[Abbildung 2: P-Glied Sprungantwort 22](#_Toc160296702)

[Abbildung 3: P-Glied Schaltsymbol 22](#_Toc160296703)

[Abbildung 4: I-Glied Schaltsymbol 22](#_Toc160296704)

[Abbildung 5: I-Glied Sprungantwort 22](#_Toc160296705)

[Abbildung 6: D-Glied Schaltsymbol 23](#_Toc160296706)

[Abbildung 7: D-Glied Sprungantwort 23](#_Toc160296707)

[Abbildung 8: PID-Regler Schaltsymbol 23](#_Toc160296708)

[Abbildung 9: PID-Regler Sprungantwort 23](#_Toc160296709)

[Abbildung 10: Flussdiagramm Programmablauf 27](#_Toc160296710)

[Abbildung 11: Real Time System Interrupt Architektur 28](#_Toc160296711)

[Abbildung 12: Motoransteuerung Architektur 29](#_Toc160296712)

[Abbildung 13: Terminal Übertragung und Status LEDs Architektur 30](#_Toc160296713)

[Abbildung 14: Fernsteuerung 34](#_Toc160296714)

[Abbildung 15: Receiver 34](#_Toc160296715)

[Abbildung 16: Fernsteuerung Tastenbelegung 35](#_Toc160296716)

[Abbildung 17: Beispiel PPM-Signal 38](#_Toc160296717)

[Abbildung 18: STM32CubeMX Einstellungen S.Bus 39](#_Toc160296718)

[Abbildung 19: STM32CubeMX Einstellungen I.Bus 40](#_Toc160296719)

[Abbildung 20: Einstellungen DMA für Receiver in STM32CubeMX 42](#_Toc160296720)

[Abbildung 21: I²C Datentransfer 55](#_Toc160296721)

[Abbildung 22: STM32CubeMX Einstellungen IMU 55](#_Toc160296722)

[Abbildung 23: I²C Schreibzyklus 56](#_Toc160296723)

[Abbildung 24: I²C Lesezyklus 57](#_Toc160296724)

[Abbildung 25: Registerübersicht MPU9250 61](#_Toc160296725)

[Abbildung 26: Registerübersicht BMP280 67](#_Toc160296726)

[Abbildung 27: Formel Berechnung Höhe über Meeresspiegel 71](#_Toc160296727)

[Abbildung 28: STM32CubeMX Einstellungen DS2438 73](#_Toc160296728)

[Abbildung 29: One-Wire Bit 1 senden 74](#_Toc160296729)

[Abbildung 30: One-Wire Bit 0 senden 74](#_Toc160296730)

[Abbildung 31: One-Wire Bit empfangen 77](#_Toc160296731)

[Abbildung 32: One-Wire Initialisierungssequenz 79](#_Toc160296732)

[Abbildung 33: Registerübersicht DS2438 81](#_Toc160296733)

[Abbildung 34: DS2438 Spannungsregister Format 83](#_Toc160296734)

[Abbildung 35: DShot Bit 0/1 Duty Cycle + Geschwindigkeit 85](#_Toc160296735)

[Abbildung 36: DShot Beispiel Übertragung 85](#_Toc160296736)

[Abbildung 37: STM32CubeMX Einstellung PID DMA Empfang 91](#_Toc160296737)

[Abbildung 38: STM32CubeMX Einstellungen Terminal 95](#_Toc160296738)

[Abbildung 39: Terminal Bild 95](#_Toc160296739)

[Abbildung 40: STM32CubeMX Einstellungen LEDs 97](#_Toc160296740)

[Abbildung 41: Beispiel PWM-Signalverlauf 98](#_Toc160296741)

[Abbildung 42: STM32CubeMX access to MCU selector 106](#_Toc160296742)

[Abbildung 43: STM32CubeMX Liste von Mikrocontroller 106](#_Toc160296743)

[Abbildung 44: STM32CubeMX Auswahl Mikrocontroller 107](#_Toc160296744)

[Abbildung 45: STM32CubeMX grafische Oberfläche 108](#_Toc160296745)

[Abbildung 46: STM32CubeMX HSE/LSE Einstellung 108](#_Toc160296746)

[Abbildung 47: STM32CubeMX clock configuration 109](#_Toc160296747)

[Abbildung 48: STM32CubeMX project settings 109](#_Toc160296748)

[Abbildung 49: STM32CubeMX register callback 109](#_Toc160296749)

[Abbildung 50: STM32CubeMX Ordnerstruktur 110](#_Toc160296750)

[Abbildung 51: µVision Zauberstab 110](#_Toc160296751)

[Abbildung 52: µVision Target Einstellungen 111](#_Toc160296752)

[Abbildung 53: µVision Debug Einstellungen 111](#_Toc160296753)

[Abbildung 54: µVision Build/Flash Knöpfe 111](#_Toc160296754)

# Begleitprotokoll

# Anhang

## — Projektdokumentation (Kostendarstellung, Besprechungsprotokolle etc.)

## — Technische Dokumentation (technische Beschreibungen, Berechnungen,

## Konstruktionszeichnungen, Versuchsberichte, betriebswirtschaftliche Kalkulationen etc.)

## Schaltungen, Zeichnungssätze, sonstiges