

BLE HID Hardware-Erweiterungsmodul für Drohnenfernbedienungen

Studienarbeit

des Studiengangs IT-Automotive
an der Dualen Hochschule Baden-Württemberg Stuttgart

von

Fabian Kuffer

15. April 2023

Bearbeitungszeitraum
Matrikelnummer, Kurs
Betreuer

4. Oktober 2022 - 8. Juni 2023
2044882, TINF-20ITA
Prof. Dr. Karl Friedrich Gebhardt

Erklärung

Ich versichere hiermit, dass ich meine Studienarbeit mit dem Thema: *BLE HID Hardware-Erweiterungsmodul für Drohnenfernbedienungen* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Stuttgart, 15. April 2023

Fabian Kuffer

Kurzfassung

TODO: Kurzfassung

Abstract

TODO: Abstract

Inhaltsverzeichnis

Abkürzungsverzeichnis	V
Abbildungsverzeichnis	VII
Tabellenverzeichnis	VIII
Quellcodeverzeichnis	IX
1 Einleitung	1
1.1 Motivation	2
1.2 Stand der Technik	2
2 Aufgabenstellung	3
2.1 Softwareentwicklung	3
2.2 Platinenentwurf	3
2.3 Gehäuseerstellung	4
3 Technische Grundlagen	5
3.1 Human Interface Device (HID)	5
3.1.1 Allgemein	5
3.1.2 Report Deskriptor	5
3.2 Bluetooth	8
3.2.1 Allgemein	8
3.2.2 Benötigte Komponenten eines Bluetooth Low Energy (BLE)-Geräts	9
3.2.3 Sollanforderungen durch Apple	14
3.2.4 HID over GATT Profile (HOGP)	14
3.2.5 Bluetooth-Stacks	17
3.3 Übertragungsprotokolle am Fernbedienungsmodulschacht	18
3.3.1 Puls-Positions-Modulation (PPM)	18
3.3.2 CRSF	19
3.3.3 SBUS	20
3.3.4 MULTI	20
3.4 Mikrocontrollerfamilie ESP32 und Zertifizierungen	21
3.5 FreeRTOS	22
3.5.1 Tasks	22
3.5.2 Software Timer	23
3.5.3 Queues	24
3.5.4 Interrupts	24
3.5.5 Besonderheiten bei der Verwendung auf dem ESP32	24

3.6	Darstellung von Glyphen auf einem Monitor	24
3.7	Eingabeereignisbehandlung unter Linux in Bezug auf Gamepads und Joysticks .	26
4	Umsetzung	28
4.1	Softwareentwicklung	28
4.1.1	Auswahl der Mikrocontrollers	28
4.1.2	Auswahl des Bluetooth-Stacks	28
4.1.3	Kommunikation zwischen dem Mikrocontroller und dem Endgerät . . .	28
4.1.4	Kommunikationsprotokoll zwischen der Multikopterfernsteuerungen und dem Mikrocontroller	30
4.1.5	Statusausgabe des Mikrocontrollers mittels eines OLED -Displays . . .	31
4.1.6	Kombination aller Softwarekomponenten	32
4.1.7	Weiterführende Informationen	33
4.2	Platinenentwurf	33
4.2.1	Teilschaltungen	33
4.2.2	Referenzschaltungen	36
4.2.3	Weiterführende Informationen	37
4.3	Gehäuseerstellung	37
4.3.1	Weiterführende Informationen	39
5	Validierung und Gegenüberstellung	40
5.1	Validierung des Funktionsumfangs	40
5.1.1	Softwareentwicklung	40
5.1.2	Platinenentwurf	41
5.1.3	Gehäuseerstellung	41
5.2	Gegenüberstellung BLE -Modul und USB-Verbindung	42
5.2.1	Versuchsaufbau	42
5.2.2	Auswertung	42
6	Rekapitulation und Ausblick	43
Literatur		44
Anhang		50

Abkürzungsverzeichnis

ADC	Analog-Digital-Wandler
API	application programming interface
ATT	Attribute Protocol
BBR	Bluetooth Basic Rate
BLE	Bluetooth Low Energy
CE	conformité européenne
CID	Kanalidentifizierer
CPU	central processing unit
CRC	Cyclic redundancy check
ESD	Electro-Static-Discharge
ESP-IDF	Espressif IoT Development Framework
evdev	Event Device
FCC	Federal Communications Commission
FDM	Fused Deposition Modeling
FIFO	First In First Out
GAP	Generic Access Profile
GATT	Generic Attribute Profile
GPIO	general purpose input/output
HCI	Host Controller Interface
HID	Human Interface Device
HOGP	HID over GATT Profile
IC	integrated circuit
IDE	integrated development environment
IOCTL	input output Control
ISM	Industrial, Scientific and Medical
I²C	inter integrated circuit
LD0	Low-dropout regulator
LED	light-emitting diode
LL	Link Layer
L2CAP	Logical Link Control and Adaption Protocol
MFi	Made for iPod/iPhone/iPad
OLED	organic light-emitting diode
PHY	Physical Layer
PPM	Puls-Positions-Modulation
RAM	Random-access memory
RED	Radio Equipment Directive
RPA	Resolvable Private Address
SAR	successive approximation
SDP	Service Discovery Protocol
SIG	Special Interest Group
SMP	Security Manager Protocol

UART Universal Asynchronous Receiver Transmitter

UUID universal unique identifier

WLAN Wireless Local Area Network

Abbildungsverzeichnis

1	HID Deskriptorenhierarchie; abgewandelt von [13, S. 4]	5
2	Beispielhafte Verwendung von Elementen, um eine Datenstruktur zu definieren; abgewandelt von [13, S. 24]	6
3	Frequenzband mit Kanälen von BLE; abgewandelt von [17, S. 4]	8
4	Benötigte Komponenten eines BLE-Geräts; abgewandelt von [16, S. 203, S. 1245]	9
5	GATT Hierarchie; abgewandelt von [16, S. 281]	13
6	Hierarchische Verwendung von Profilen; abgewandelt von [16, S. 1468]	13
7	Benötigte Dienste eines HID-Geräts; abgewandelt von [20, S. 11]	15
8	Darstellung einzelner Kanäle im Zeitverlauf bei PPM; abgewandelt von [36] . .	18
9	Taskzustände und deren Übergänge; angepasst von [57, S. 67]	23
10	Speichern der Glyphen U spaltenweise in einem Array aus Bytes; abgewandelt von [71]	25
11	Umwandlung einer Vektor-Schriftart zur Darstellung an einem Monitor; abgewandelt von [72]	26
12	Gegenüberstellung der Schriftartenspeicherverfahren	26
13	Verarbeitung eines Datenstreams für die Darstellung auf dem Erweiterungsmoduldisplay; abgewandelt von [89, S. 37]	32
14	Platinen des Erweiterungsmoduls	33
15	Pinbelegung des Fernsteuerungsmodulschachts; abgewandelt von [90]	34
16	Spannungsregulierung des Erweiterungsmoduls	34
17	ESP32 Programmierlogik	35
18	Hardwaretastenentprellung	36
19	Electro-Static-Discharge (ESD)-Schutzschaltung	36
20	Modell mit Überhängen für den 3D-Druck vorbereiten	37
21	Modell mit Überhängen für den 3D-Druck vorbereiten	38
22	Messing Gewindegarnituren zur Befestigung der einzelnen Modulkomponenten .	38
23	Eingebaute Platinen im Gehäuse	39
24	Tasterschutz für Taster der Platine	39
25	Zusammengebautes Erweiterungsmodul	39
26	Schaltplan der Hauptplatine	51
27	Schaltplan für Ein- und Ausgabekomponenten	52
28	Schaltplan für die Verbindung zwischen Erweiterungsmodul und Multikopterfernsteuerung	53
29	Explosionszeichnung mit allen Modellen des Gehäuses	54

Tabellenverzeichnis

1	Datenstruktur einer Maus mit drei Knöpfen; abgewandelt von [15]	7
2	Liste der verfügbaren Geräteinformationsmerkmale	17
3	Aufbau eines CSRF-Pakets [38]	19
4	Ausschnitt aus den vorhandenen CSRF-Geräteadressen [39]	19
5	Vordefinierte CSRF Datentypen [39]	19
6	Paketaufbau von SBUS [42]	20
7	Paketaufbau von MULTI [44]	21
8	Liste der verfügbaren Geräteinformationsmerkmale	29

Quellcodeverzeichnis

1	Report Deskriptor einer Maus mit 3 Knöpfen [15]	7
2	Minimaler Aufbau eines Tasks	22
3	C-Strukuraufbau eines Eingabeereignisses von evdev [75]	26
4	Report Map Deskriptor des Erweiterungsmoduls	29
5	C-Strukuraufbau der aufbereiteten Kanaldaten	31

1 Einleitung

Multikopter beziehungsweise Quadrokopter haben in den letzten Jahren sowohl im privaten als auch im kommerziellen Sektor ein konstantes Wachstum erreicht. So sind beispielsweise in den Vereinigten Staaten von Amerika mit Stand vom 31. Mai 2022 über 865.000 Multikopter registriert. Davon sind über 500.000 privat und über 300.000 für die kommerzielle Nutzung registriert. [1], [2], [3], [4]

Quadrokopter lassen sich grob in zwei Kategorien einteilen. Zum einen gibt es die Consumerquadrokopter, welche viele Sensoren enthalten, um Unterstützungsfunctionen an die teils ungeübten Piloten bereitzustellen. Ein Beispiel für einen Quadrokopter dieser Kategorie ist die Mavic 3 Classic vom Unternehmen DJI. Diese hat Sichtsensoren, welche nach unten, oben, vorne und hinten ausgerichtet sind, um Objekte im Flugfeld zu erkennen und somit ausweichen zu können. Weitere Hilfsfunktionen, welche der Quadrokopter besitzt, ist der automatische Rückflug an den Startort, sowie die Möglichkeit beide Steuerknüppel loszulassen und dabei stabil die Lage in der Luft beizubehalten. [5]

Die zweite Kategorie von Quadrokoptern sind sogenannte Freestyle- beziehungsweise Rennquadrokopter. Diese sind im Gegensatz zu den Consumerquadrokoptern dazu ausgelegt möglichst leicht zu sein, um möglichst schnelle und beeindruckende Manöver machen zu können. Im Gegenzug wird auf die Unterstützungsfunctionen von Consumerquadrokoptern verzichtet. Eine weitere Besonderheit ist die Möglichkeit zwischen drei Flugmodi auszuwählen. Zum einen den Flugmodus *Angle Mode*. In diesem Modus wird der Quadrokopter bis zu einem fest vordefinierten Neigungswinkel automatisch begrenzt, wodurch Loopings und Rollen des Quadroopters unterbunden werden. Ebenso dreht sich der Quadrokopter wieder in die Ausgangslage zurück, wenn die Steuerknüppel zentriert werden. Der zweite Flugmodus ist der *Horizon Mode*, dieser bietet wie der *Angle Mode* die Funktion, dass sich der Quadrokopter wieder zur Ausgangslage zurückdreht, wenn die Steuerknüppel in die zentrale Stellung zurückgebracht werden. Jedoch können in diesem Flugmodus Loopings und Rollen gemacht werden. Der letzte verfügbar Flugmodus ist der *Air beziehungsweise Acro Mode*. In diesem Modus muss der Pilot sich um das Ausrichten der Drohne in alle Drehrichtungen selbst kümmern, da bei Loslassen der Steuerknüppel die vorhandenen Drehungen des Quadroopters beibehalten werden. Dieser Modus wird meist von Freestyle- und Rennpiloten verwendet. [6]

Neben dem Multikopterfliegen stellt für Renn- und Freestyle-Quadrokopterpiloten das Training einen wichtigen Bestandteil dar, um die Bedienung des Quadroopters im *Air beziehungsweise Acro Mode* zu verbessern. Das Training kann in zwei Varianten durchgeführt werden. Der Quadrokopterpilot trainiert entweder am Flugplatz – hier können aber durch Abstürze hohe Reparaturkosten und lange Reparaturzeiten entstehen –, oder der Quadrokopterpilot trainiert im Simulator am Rechner, wobei keine Reparaturkosten und Reparaturzeiten entstehen.

1.1 Motivation

Da in den letzten Jahren das Unternehmen Apple Tablets mit leistungsstarken Prozessoren, welche ursprünglich für Notebooks und Desktops gedacht waren, entwickelt hat [7], wäre es wünschenswert Quadrokopter-Simulatoren für die immer leistungsfähigeren mobilen Geräte bereitzustellen. Hierfür muss jedoch die Möglichkeit bestehen, die gewohnte Fernsteuerung der Quadrokopter mit Endgeräten zu verbinden, um den Piloten eine gewohnte Umgebung zu bieten. Die Verbindung einer Fernsteuerung mit einem Endgerät bieten einige Hersteller an, indem sich die Fernsteuerung per USB als USB-HID-Joystick identifiziert [8]. Das Problem hierbei ist jedoch, dass die Verbindung mittels USB mit mobilen Geräten nur eingeschränkt beziehungsweise unmöglich herzustellen ist. Beseitigt werden kann dieses Problem bei Fernsteuerung mit Modulschächten [9], mit deren Hilfe die Tasten- und Joysticksignale über andere Kommunikationswege übertragen werden können.

Ziel der Arbeit ist es daher ein Hardware-Erweiterungsmodul für Multikopterfernsteuerungen zu entwickeln, womit eine Fernsteuerung mit einem Endgerät verbunden werden kann, welches nicht USB zur Datenübertragung bereitstellt.

1.2 Stand der Technik

Damit Fernsteuerungen von Quadrokoptern für das Training im Simulator an Computern verwendet werden können, gibt es zurzeit drei Möglichkeiten. Die erste Möglichkeit ist die Verbindung der Fernsteuerung von ausgewählten Herstellern mittels USB. Dadurch wird die Fernsteuerung am Computer als USB-HID-Joystick erkannt [8]. Die zweite Möglichkeit ist den Quadrokopter, auf dem die Firmware Betaflight vorhanden ist, mittels USB anzustecken und diesen als Empfänger für die Fernsteuerung zu benutzen [10]. Da diese zwei Möglichkeiten jeweils USB zur Datenübertragung verwenden, sind diese Varianten nicht für alle Endgeräte geeignet. Die letzte Möglichkeit ist die Anwendung eines Hardwaremoduls an einer Fernsteuerung mit Erweiterungsmodulschacht. Die Datenübertragung erfolgt hier mittels Bluetooth. Das einzige bekannte Modul dieser Art stellt das Unternehmen Orqa her [11]. Dieses Modul ist jedoch nur für Modulschächte des Typs JR geeignet.

2 Aufgabenstellung

Ziel der Arbeit ist es, ein Hardware-Erweiterungsmodul für Multikopterfernsteuerungen zu entwickeln. Vorausgesetzt wird im Rahmen dieser Arbeit, dass die Fernsteuerungen einen Modulschacht aufweisen und die Firmware OpenTX [12] beziehungsweise eine Abspaltung davon verfügbar ist. Das Erweiterungsmodul soll sich dabei durch **BLE** als **HID**-Gerät an Endgeräten authentifizieren, wodurch die Multikopterfernsteuerung als kabelloser Joystick an Endgeräten verwendet werden kann. Die Umsetzung dieser Arbeit lässt sich in nachfolgende drei Teilbereiche aufteilen.

2.1 Softwareentwicklung

Im Aufgabenbereich der Softwareentwicklung soll die Kommunikation zwischen dem ESP32-Entwicklerboard und Windows, Linux, iOS/ iPadOS und Android-Systemen hergestellt werden. MacOS soll kein Teil der unterstützten Betriebssysteme sein, da kein Endgerät mit MacOS zum Testen vorhanden ist. Die Kommunikation soll dabei mittels **BLE** stattfinden und das Entwicklerboard soll sich als **HID**-Gerät authentifizieren. Des Weiteren soll die Kommunikation zwischen dem ESP32-Entwicklerboard und der Fernsteuerung mittels des Modulschachts der Fernsteuerung implementiert werden. Dafür soll auf eines der vorhandenen Protokolle der Fernsteuerung zurückgegriffen werden, damit die Firmware der Fernsteuerung nicht angepasst werden muss. Der letzte Bestandteil dieses Aufgabenbereichs ist die Implementierung weiterer Möglichkeiten der Eingabe und Ausgabe an dem ESP32-Entwicklerboard. Dafür soll zum einen ein 0,91 Zoll großes **OLED**-Display verwendet werden, um kurze Statusnachrichten anzuzeigen. Ebenso sollen Status-**LEDs** die Interaktion mit dem Modul vereinfachen. Die Bestimmung des Akkustandes der Fernsteuerung soll mittels des integrierten Analog-Digital-Wandlers (**ADCs**) des ESP32-Entwicklerboard geschehen, da der Akkustand bei **HID**-Geräten bereitgestellt werden muss.

2.2 Platinenentwurf

In diesem Aufgabenbereich soll der erstellte Steckbrettaufbau, der während der Softwareentwicklung benötigt wurde, in eine Platine umgewandelt werden. Ein Bestandteil dieser Platine soll das ESP32-Modul sein, welches als primärer Mikrocontroller fungiert. Ebenso soll eine Spannungsregulierung für die Komponenten der Platine erstellt werden, da die Elektronik über die Stromversorgung der Fernsteuerung betrieben werden soll. Ein weiterer Bestandteil der Platine ist die Verbindung der Ein- und Ausabeelemente mit dem ESP32-Modul, um die Bedienung des Erweiterungsmoduls zu vereinfachen. Zur Umsetzung soll auf bereits vorhandene Referenzdesigns des ESP32-Entwicklerboards zurückgegriffen werden.

2.3 Gehäuseerstellung

Im letzten Aufgabenbereich soll ein Gehäuse für die erstellte Platine hergestellt werden, damit die Platine in den Modulschacht vom Typ Lite fest verbaut werden kann. Ebenso muss bei der Konstruktion beachtet werden, dass das Gehäuse möglichst ohne Stützstrukturen mittels eines 3D-Druckers gedruckt werden kann. Dadurch soll die Nachbearbeitung des Gehäuses nach dem Druck auf ein Minimum reduziert werden.

3 Technische Grundlagen

3.1 Human Interface Device (HID)

3.1.1 Allgemein

Das USB Protokoll kann Geräte beim Starten beziehungsweise beim Einsticken an ein Computersystem automatisch konfigurieren. Dafür werden Geräte in Klassen eingeteilt. Jede Klasse definiert dabei wie das gewöhnliche Verhalten und die verwendeten Protokolle der Geräte der Klasse sind. Eine dieser Klassen in USB ist die **HID** Klasse. Der primäre Einsatzzweck für Geräte in der **HID** Klasse ist die Bedienung von Computern durch Menschen. Beispiele für solche Geräte sind: Tastaturen, Mäuse, Joysticks, Barcodeleser und auch Simulationsgeräte. [13, S. 1f.]

Informationen eines USB-Geräts für ein Computersystem werden in Segmente, auch Deskriptoren genannt, des ROMs des jeweiligen USB-Geräts abgespeichert. Ein Gerät, welches zur **HID**-Klasse gehört, hat wie in Abbildung 1 zu sehen ist, drei Deskriptoren. Zunächst einmal den **HID** Deskriptor, welcher alle weiteren benötigten Deskriptoren für USB-**HID**-Geräte auflistet. Der zweite optionale Deskriptor ist der physikalische Deskriptor, welcher nicht genauer in dieser Arbeit betrachtet wird. Er stellt Informationen an das System bereit, wie einzelne Teile des **HID**-Geräts von einem Menschen bedient werden sollen. Der letzte Deskriptor ist der Report Deskriptor. Mittels dieses Deskriptors wird die Struktur der übermittelten Daten zwischen dem Rechnersystem und dem **HID**-Gerät beschrieben. [13, S. 4f.]

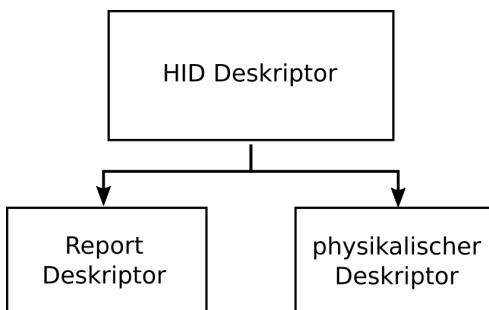


Abbildung 1: **HID** Deskriptorenhierarchie; abgewandelt von [13, S. 4]

3.1.2 Report Deskriptor

Das USB-Protokoll definiert meist in USB-Geräte-Klassen das zu verwendende Protokoll des Geräts durch Subklassen. Dies ist jedoch in der USB-**HID**-Klasse nicht der Fall. USB-**HID**-Geräte stellen nämlich durch den Report Deskriptor einem anderen System den Aufbau und die Datentypen der übermittelten Datenpakete bereit. [13, S. 8] Durch dieses Verfahren ist es möglich, dass Applikationen durch Lesen des Report Deskriptors die modular aufgebauten

Daten verarbeiten können [13, S. 24]. Die Länge des Deskriptors ist dabei für jedes Gerät variabel und hängt von der Menge der übermittelten Daten ab [13, S. 23].

Ein Report Deskriptor ist aus sogenannten Elementen aufgebaut. Ein Element stellt eine Teilinformation über ein USB-HID-Gerät dar. Jedes Element hat zu Beginn jeweils ein 1 Byte großes Präfix. In diesem Präfix befinden sich jeweils ein Element-Marker, ein Elementtyp und eine Elementengröße. Darauf folgend können optional Daten angefügt werden. Es kann dabei zwischen langen und kurzen Elementen unterschieden werden, wodurch die Größe zwischen 0 und 258 Byte groß sein kann. [13, S. 14]

Alle Elemente, die in einem Report Deskriptor enthalten sein müssen, sind in nachfolgender Aufzählung enthalten: Input (Output oder Feature), Usage, Usage_Page, Logical_Minimum, Logical_Maximum, Reprot_Size und Report Count. [13, S. 24]

Alle Elemente eines Report Deskriptors lassen sich in drei Gruppen einordnen. Zunächst gibt es die Hauptelemente. Diese werden verwendet, um Datenfelder zu definieren oder Datenfelder zu gruppieren. Die zweite Gruppe sind die globalen Elemente. Mit diesen werden die zu übermittelten Datenfelder beschrieben. Die letzte Gruppe umfasst die lokalen Elemente. Diese werden verwendet, um Merkmale eines Datenfelds zu beschreiben. [13, S. 16, S. 28, S. 35]

Die Einsatzzwecke der drei Gruppen stehen folgendermaßen in Beziehung. Mittels eines Hauptelements wird ein Datenfeld definiert. Durch globale und lokale Elemente werden erstellten Datenfeldern Definitionen hinzugefügt. Dabei gelten lokale Elemente nur für das nächst kommende Hauptelement. Globale Elemente gelten im Gegensatz dazu so lange, bis das globale Element durch ein anderes globales Element überschrieben wird. Dadurch ist es möglich Datenstrukturen wie in Abbildung 2 zu erstellen. [13, S. 24]

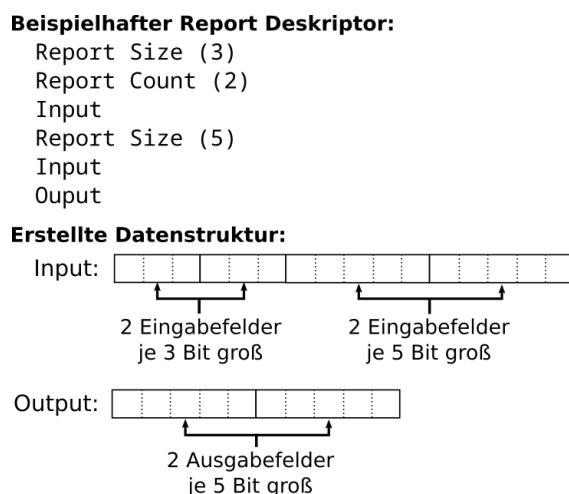


Abbildung 2: Beispielhafte Verwendung von Elementen, um eine Datenstruktur zu definieren; abgewandelt von [13, S. 24]

In der Gruppe der Hauptelemente gibt es fünf Element-Marker. Ein häufig verwendeter Marker ist der Input-Marker, mittels dessen Ausgabedatenfelder definiert werden können. Weitere wichtige Element-Marker für Hauptelemente sind Collection und End Collection, womit Datenfelder gruppiert werden können. [13, S. 23f., S. 30ff.]

In der Gruppe der globalen Elemente gibt es 13 Element-Marker. Wichtige Marker hierbei sind: Usage Page, Logical Minimum, Logical Maximum, Report ID, Report Size und Report Count. Mittels des Markers Report Size wird angegeben wie groß ein Datenfeld in Bits sein soll. Mit dem Marker Report Count wird angegeben wie viele Datenfelder mit den definierten Eigenschaften erstellt werden sollen. Mit dem Marker Report ID ist es möglich mehrere Datenstrukturen innerhalb eines Report Deskriptors zu erstellen und diese eindeutig zu identifizieren [13, S. 17]. [13, S. 35ff.]

Wichtige Element-Marker der Gruppe der lokalen Elemente, welche elf Element-Marker umfasst, sind: Usage, Usage Minimum und Usage Maximum [13, S. 40]. Durch den Element-Marker Usage wird der Einsatzzweck eines Datenfelds definiert. Ebenso können statt einzelnen Datenfeldern auch Kollektionen von Datenfeldern mit Einsatzzwecken markiert werden. Einsatzzwecke sind in Einsatzzweck-Seiten organisiert und werden mit dem Element-Marker Usage Page definiert. Beachtet werden sollte, dass ein Einsatzzweck so spezifisch wie nötig und so allgemein wie möglich gehalten werden sollte, damit das HID-Gerät alle gerätespezifischen Eigenschaften bereitstellen kann. [14, S. 15f.]

Schlussendlich können mittels Padding-Bits die enthaltenen Datenfelder byteorientiert ausgerichtet werden. In Tabelle 1 ist eine beispielhafte Datenstruktur für eine Maus mit dazugehörigen Report Deskriptor in Quellcode 1 zu sehen.

Tabelle 1: Datenstruktur einer Maus mit drei Knöpfen; abgewandelt von [15]

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0	unbenutzt	unbenutzt	unbenutzt	unbenutzt	unbenutzt	linke Taste	mittlere Taste	rechte Taste
Byte 1	relative X-Achsen Bewegung als signed Integer							
Byte 2	relative Y-Achsen Bewegung als signed Integer							

Quellcode 1: Report Deskriptor einer Maus mit 3 Knöpfen [15]

```

0x05, 0x01,      //USAGE_PAGE (Generic Desktop)
0x09, 0x02,      //USAGE (Mouse)
0xa1, 0x01,      //COLLECTION (Application)
0x09, 0x01,      //  USAGE (Pointer)
0xa1, 0x00,      //  COLLECTION (Physical)
0x05, 0x09,      //    USAGE_PAGE (Button)
0x19, 0x01,      //    USAGE_MINIMUM (Button 1)
0x29, 0x03,      //    USAGE_MAXIMUM (Button 3)
0x15, 0x00,      //    LOGICAL_MINIMUM (0)
0x25, 0x01,      //    LOGICAL_MAXIMUM (1)
0x95, 0x03,      //    REPORT_COUNT (3)
0x75, 0x01,      //    REPORT_SIZE (1)
0x81, 0x02,      //    INPUT (Data, Var, Abs)
0x95, 0x01,      //    REPORT_COUNT (1)
0x75, 0x05,      //    REPORT_SIZE (5)
0x81, 0x03,      //    INPUT (Cnst, Var, Abs)
0x05, 0x01,      //    USAGE_PAGE (Generic Desktop)
0x09, 0x30,      //    USAGE (X)
0x09, 0x31,      //    USAGE (Y)
0x15, 0x81,      //    LOGICAL_MINIMUM (-127)

```

```

0x25, 0x7f,      // LOGICAL_MAXIMUM (127)
0x75, 0x08,      // REPORT_SIZE (8)
0x95, 0x02,      // REPORT_COUNT (2)
0x81, 0x06,      // INPUT (Data, Var, Rel)
0xc0,            // END_COLLECTION
0xc0            //END_COLLECTION

```

3.2 Bluetooth

3.2.1 Allgemein

Bluetooth ist ein Kurzstreckenkommunikationssystem, bei welchem die Hauptmerkmale auf Robustheit, einen geringen Stromverbrauch und geringe Kosten gelegt wurde. Bluetooth wird in zwei Kategorien aufgeteilt. Die erste Kategorie ist Bluetooth Basic Rate (**BBR**). Die zweite Kategorie ist **BLE**. Beide Kategorien beinhalten dabei Mechanismen, um Bluetooth-Geräte zu entdecken, einen Verbindungsaufbau durchzuführen sowie eine Verbindung herzustellen. Das Augenmerk bei **BLE** Produkten liegt dabei auf einem niedrigen Stromverbrauch, was durch eine geringere Datenrate und eine geringere Einschaltzeit während des Datenaustauschs als bei **BBR** realisiert wird. Die Übertragungsrate bei **BLE** in der physikalischen Schicht beträgt 2 MB/s. Zu beachten ist, dass ein Bluetooth-Controller entweder **BLE**, **BBR** oder beide Bluetooth-Kategorien unterstützen kann. [16, S. 187]

Die Übertragungsfrequenz von **BLE** liegt im lizenzenfreien 2.4 GHz Industrial, Scientific and Medical (**ISM**)-Band von 2402 MHz bis 2489 MHz [17, S. 4], [16, S. 190]. Das Frequenzband ist in 40 physikalische Kanäle mit jeweils einer Bandbreite von 2 MHz aufgeteilt, wie in Abbildung 3 zu sehen ist [16, S. 190]. Drei dieser 40 physikalischen Kanäle sind für das sogenannte Advertising vorhanden [16, S. 190], welches für die Geräteentdeckung, den Verbindungsaufbau und für das Broadcasting von Nachrichten vorhanden ist [17, S. 4]. Die restlichen Kanäle sind für eine allgemeine Datenübertragung dar [16, S. 190]. Zusätzlich zu der Aufteilung des Frequenzbandes in Kanäle werden Kanäle in Zeiteinheiten aufgeteilt, welche Events genannt werden [16, S. 190]. Daten werden in Paketen innerhalb eines Events übertragen. Zusätzlich wird bei der Übertragung von Daten Frequenzhopping betrieben, welches zu Beginn jedes Events stattfindet [16, S. 190f.].

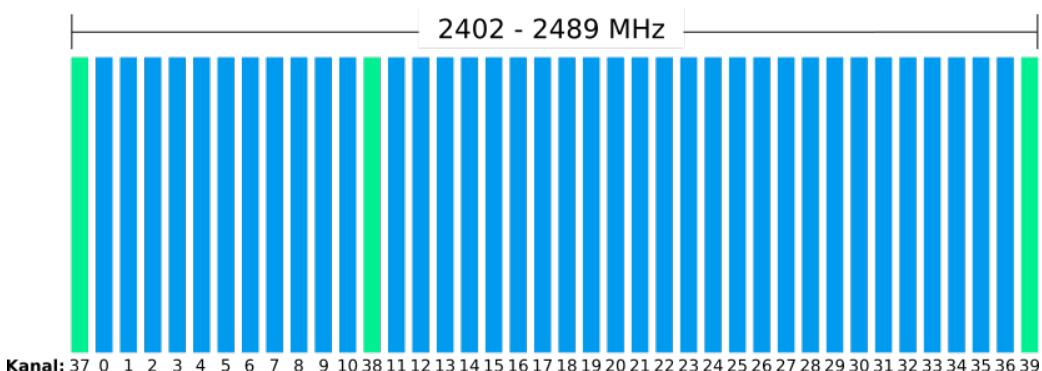


Abbildung 3: Frequenzband mit Kanälen von **BLE**; abgewandelt von [17, S. 4]

Die Kompatibilität zwischen Bluetooth-Geräten wird durch sogenannte Profile sichergestellt. Profile beschreiben Funktionen und Eigenschaften von jeder Schicht im Bluetoothsystem [16, S. 277]. Die benötigten Nachrichten und Prozeduren für die verwendeten Profile werden durch die Bluetooth Special Interest Group (**SIG**) spezifiziert [16, S. 1241].

Bluetooth-Geräten werden unterschiedliche Rollen zugewiesen welche entweder Observer, Broadcaster, Central oder Peripheral sein können. Ein Gerät mit der Rolle Broadcaster versendet Advertising-Pakete und ein Gerät, welches nur Advertising-Pakete empfangen kann, hat die Observer Rolle. So kann eine einseitige Kommunikation zwischen Geräten mittels Advertising-Paketen erfolgen. Eine andere Art der Kommunikation ist mittels einer Verbindung, bei dem das Initiatorgerät eine Verbindungsanfrage eines Broadcastergeräts annimmt. Daraufhin bekommt das Initiatorgerät die Rolle Central und das Gerät, welches ursprünglich in der Rolle Broadcaster war, die Rolle Peripheral. Anzumerken ist, dass ein Gerät zu jeder Zeit mehrere Rollen unterstützen kann, welche jedoch alle der Bluetooth-Controller unterstützen muss. [16, S. 190f., S. 278, S. 1246ff.]

3.2.2 Benötigte Komponenten eines **BLE**-Geräts

Ein **BLE**-Gerät benötigt einen Mindestumfang an Funktionen, damit es laut Bluetooth **SIG BLE** kompatibel ist. In Abbildung 4 sind die benötigten Funktionen und deren Zusammenspiel durch ein Schichtenmodell dargestellt. Die Funktionen können dabei in einen Hostteil und einen Controllerteil aufgeteilt werden. Im Hostteil befinden sich die Funktionen Logical Link Control and Adaption Protocol (**L2CAP**), Generic Access Profile (**GAP**), Attribute Protocol (**ATT**), Generic Attribute Profile (**GATT**), Service Discovery Protocol (**SDP**) und Security Manager Protocol (**SMP**). Im Controllerteil befinden sich die Funktionen Physical Layer (**PHY**) und Link Layer (**LL**). Die Kommunikation zwischen den Hostteil und dem Controllerteil findet mittels des Host Controller Interface (**HCI**) statt [16, S. 1735]. [16, S. 193]

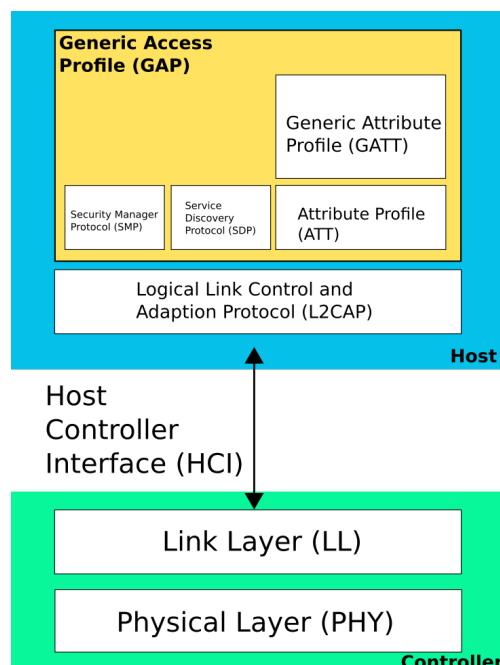


Abbildung 4: Benötigte Komponenten eines **BLE**-Geräts; abgewandelt von [16, S. 203, S. 1245]

In den nachfolgenden Unterkapiteln werden die wichtigsten Informationen jeder benötigten Funktion von **BLE** beschrieben.

Physical Layer (PHY)

Die physikalische Schicht in **BLE** ist zum Verschicken und Erhalten von Paketen über eines der physikalischen Funkkanäle verantwortlich. [16, S. 209]

Link Layer (LL)

Die Verbindungsschicht im **BLE**-System besteht aus mehreren Komponenten. Eine Komponente ist für die Erstellung, Modifizierung und das Freigeben von logischen Verbindungen zuständig. Eine weitere Komponente ist für das Kodieren und Dekodieren von Bluetooth Paketen zuständig. Es gibt eine weitere Komponente, welche für die Datenflusskontrolle, die Datenbestätigung und für die erneute Übertragung von Paketen zuständig ist. Die letzten Komponenten in der Verbindungsschicht sind für den Zugriff auf das Radiomedium zuständig. Dafür gibt es einen Scheduler, welcher Zeitschlüsse des physikalischen Mediums an die höherliegenden Dienste verteilt. [16, S. 207f.]

Host Controller Interface (HCI)

Das Host Controller Interface stellt die Möglichkeit bereit, dass der Hostteil die Funktionen des Controllerteil erreichen kann. Die Übertragung des **HCI** kann dabei wahlweise mittels USB, UART oder anderen Bussystemen stattfinden. [16, S. 1735f.]

Logical Link Control and Adaption Protocol (L2CAP)

Das Logical Link Control and Adaption Protocol ist die Schicht im **BLE**-Stack, welches eine kanalbasierte Abstraktion zu den Applikationen und Diensten der höheren Schichten bereitstellt. Diese Schicht kümmert sich zusätzlich um die Segmentierung, den Zusammenbau, das Multi- und Demultiplexing von Daten auf einer, beziehungsweise mehreren, logischen Verbindungen. [16, S. 195, S. 1013]

Logical Link Control and Adaption Protocol baut dabei auf dem Konzept von logischen Kanälen auf, wobei jeder Endpunkt eines logischen Kanals einen eindeutigen Kanalidentifizierer (**CID**) hat [16, S. 1021]. Die logischen Kanäle werden über logische Verbindungen der **LL**-Schicht übertragen [16, S. 1013].

Generic Access Profile (**GAP**)

Das Generic Access Profile beschreibt die Basisfunktionalitäten, welche ein **BLE**-Gerät benötigt [16, S. 207]. Dabei werden alle in diesem Kapitel vorgestellten Schichten als Mindestanforderung aufgelistet und alle benötigten Fähigkeiten, die eine **BLE**-Rolle enthalten muss [16, S. 277f., S. 1241].

Weitere wichtige Eigenschaften, die in **GAP** definiert sind, sind zum einen die Bluetooth-Geräteadressen. Die Geräteadresse wird verwendet, um ein Bluetooth-Gerät eindeutig zu identifizieren. Eine weitere Eigenschaft, welche in **GAP** definiert wird, ist der Gerätename. Dieser Name ist eine benutzerfreundliche Zeichenfolge die an entfernten Geräten angezeigt wird. Der Gerätename kann bis zu 248 Byte lang sein und sollte in UTF-8 kodiert sein. Es muss davon ausgegangen werden, dass ein Gerät nur die ersten 40 Zeichen auswerten kann. [16, S. 1251ff.]

Damit eine Verfolgung von Geräteadressen minimiert werden kann, gibt es in **BLE** zwei Arten von Geräteadressen. Zum einen eine sich verändernde öffentliche Adresse, welche an alle **BLE**-Geräte verschickt wird. Zum anderen gibt es sich nicht verändernde private Adressen, welche von Geräten, die schon einmal eine Verbindung mit einem bestimmten Gerät aufgebaut hatten, ausgerechnet werden können. Somit kann eine Überprüfung des Gegengeräts stattfindet. [17, S. 19]

Auch wird in **GAP** beschrieben, wie der Bluetooth-Pin für eine Authentifizierung zweier Geräte im Verbindungsmodus aufgebaut sein muss. Die Pin soll sechs Zeichen lang sein und aus Ziffern bestehen. [16, S. 1253]

Als letzten Bestandteil beschreibt **GAP** die verschiedenen Sicherheitsmodi, welche durch die verschiedenen **BLE**-Rollen implementiert sein müssen [16, S. 1337].

Service Discovery Protocol (**SDP**)

SDP ermöglicht es, die verfügbaren Dienste und die zugehörigen Merkmale eines Bluetooth-Geräts für entfernte Geräte sichtbar zu machen [16, S. 1173]. Dabei pflegt das Gerät eine Liste aller Dienste und Merkmale des Geräts [16, S. 1177].

Security Manager Protocol (**SMP**)

SMP definiert Methoden zum Verbindungsauflaufbau und zum Schlüsselaustausch zwischen Bluetooth-Geräten [16, S. 1554]. Die gerätespezifischen Schlüssel werden für die Identifizierung von Geräten und für den verschlüsselten Datenaustausch zwischen Geräten verwendet [16, S. 1556], [17, S. 18].

Der Verbindungsauflaufbau und der dazugehörige Schlüsselaustausch für die Identifizierung der Geräte erfolgt in 3 Phasen. In der ersten Phase wird ein Verbindungsauflaufbau angefragt. Die zweite Phase nach einer erfolgreichen Anfrage ist die Generierung eines Schlüssels mit einer kurzen oder langen Lebenszeit. In der letzten Phase wird der generierte Schlüssel an die Gegenstelle bereitgestellt. [16, S. 1556]

Zu beachten ist, dass es verschiedene Möglichkeiten gibt einen Verbindungsaufbau herzustellen, der abhängig von den Sicherheitsansprüchen der Anwendung definiert werden kann [17, S. 18].

Attribute Protocol (**ATT**)

ATT ist ein Teilnehmer-zu-Teilnehmer Protokoll zwischen zwei Geräten [16, S. 206]. **ATT** definiert dabei zwei Rollen, den Client und den Server [16, S. 1410]. **ATT** erlaubt es Geräten (Clients) kleine Werte (sogenannte Attribute [16, S. 279]), welche sich auf dem Gerät mit der Rolle Server befinden, zu lesen, zu schreiben und zu entdecken [16, S. 1409]. Ein Gerät kann sich simultan in der Rolle Server und Client befinden [16, S. 279].

Ein Attribut besteht jeweils aus drei Eigenschaften. Die erste Eigenschaft ist der Attribut-Typ, welcher durch einen universal unique identifier (**UUID**) definiert wird und in **SDP** definiert ist. Die zweite Eigenschaft ist der Attribut-Handle. Der Attribut-Handle ist ein einzigartiger, eindeutig definierter Identifikator für ein Attribut auf einem Gerät mit der Rolle Server. Die letzte Eigenschaft eines Attributs sind die Berechtigungen, welche durch eine höhere Schicht definiert werden müssen. [16, S. 1410ff.]

Attribut-Handles haben eine Länge von 16 Bit und können durch weitere spezielle Attribute gruppiert werden [16, S. 1412f.]. Die Entdeckung aller vorhandenen Attribute eines Servers durch einen Client erfolgt durch eine höhere Schicht des **BLE**-Stacks [16, S. 1410].

Die hinterlegten Werte eines Attributs bestehen aus einem Oktett-Array mit einer fixen oder variablen Länge [16, S. 1413].

Generic Attribute Profile (**GATT**)

GATT baut auf **ATT** auf und stellt ein Framework für die Daten, welche in **ATT** gespeichert werden, bereit. **GATT** definiert wie **ATT** zwei Rollen, den Server und den Client. Ebenso legt **GATT** das Format der Daten, welche auf dem **GATT**-Server gespeichert werden dürfen, in sogenannten Profilen fest. Attribute werden hierfür in Profile, Dienste und Merkmale untergliedert, wie in Abbildung 5 zu sehen ist. Ein Applikationsprofil besteht aus einen oder mehreren Diensten, um bestimmt definierte Use-Cases abzudecken und definiert darüber hinaus die benötigten Dienste, Merkmale und Attribute [16, S. 207]. Ein Dienst enthält eine Ansammlung von Merkmalen und kann andere Dienste inkludieren. Ein Merkmal enthält einen Wert sowie eine Menge von Deskriptoren. Durch diesen Aufbau kann ein Client die Daten eines bestimmten Profils auslesen, ohne davor den Aufbau der Attribute des Servers kennen zu müssen. [16, S. 280, S. 1480]

Anzumerken ist, dass jedes Attribut, welches in **ATT** vorhanden ist, entweder in einer Dienstdeklarierung oder in einer Dienstdefinition enthalten sein soll. [16, S. 1483]

Das **GATT**-Profil soll von anderen Profilen als Grundstruktur verwendet werden, damit eine reibungslose Kommunikation zwischen einem Client und Server sichergestellt werden kann, wie in Abbildung 6 zu sehen ist. [16, S. 1470]

Ein Dienst stellt unter **GATT** eine Ansammlung von Daten dar, um ein bestimmtes Verhalten des Servers zu erzielen. Ein Dienst kann zur Vereinfachung der Verhaltensdarstellung weitere

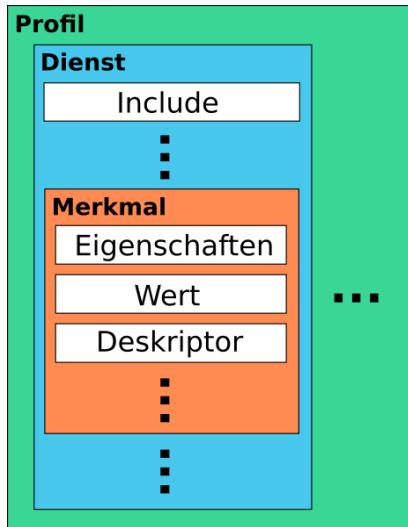


Abbildung 5: GATT Hierarchie; abgewandelt von [16, S. 281]

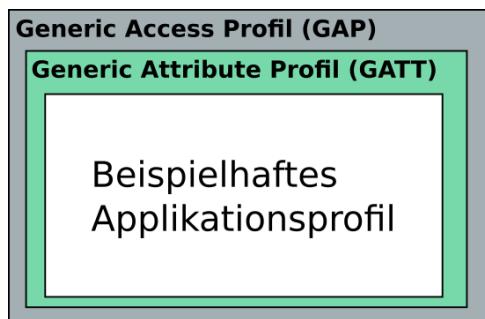


Abbildung 6: Hierarchische Verwendung von Profilen; abgewandelt von [16, S. 1468]

Dienste inkludieren. Dienste können in zwei Gruppen eingeteilt werden. Zunächst einmal in die primären Dienste. Primäre Dienste bieten alleinstehende Funktionalitäten an. Im Gegensatz dazu gibt es sekundäre Dienste, welche optionale Funktionalitäten enthalten und von mindestens einem primären Dienst inkludiert werden müssen. [16, S. 281]

Die Definition eines Dienstes umfasst die inkludierten Dienste sowie die benötigten und optionalen Merkmale [16, S. 1481].

Der Start eines Dienstes in der Liste der ATT-Attribute wird durch ein spezielles Attribut festgelegt, vom Attribut-Typ *primärer Dienst* oder *sekundärer Dienst*. Das Ende eines Dienstes wird durch eine Folgedeklaration eines neuen Dienstes festgelegt. [16, S. 1483]

Merkmale sind Werte eines Dienstes, welche aus mehreren ATT-Attributen besteht. Ein Merkmal besteht aus drei Komponenten. Der Deklaration, den Eigenschaften des Merkmals und dem dazugehörigen Wert. Zusätzlich können noch Deskriptoren in einem Merkmal enthalten sein, um die Berechtigungen des Merkmals zu setzen. [16, S. 281]

Der Start eines Merkmals in der Liste der ATT-Attribute wird durch ein spezielles Attribut festgelegt, vom Attribut-Typ *Merkmal* enthält. Das Ende eines Merkmals stellt eine neue Merkmaldeklaration oder eine neue Dienstdeklaration dar. [16, S. 1484ff.]

3.2.3 Sollanforderungen durch Apple

Im Apple Ökosystem muss Zubehör, welches Made for iPod/iPhone/iPad (**MF**i) lizenzierte Technologie zur Verbindung zu Apple Geräte verwendet, beispielsweise **MF**i Game Controller, von Apple geprüft werden. Eine Ausnahme stellen dabei **BLE**-Geräte dar. [18] Diese Geräte müssen einige Sollanforderungen in Bezug auf **BLE** erfüllen. Eine Anforderung ist, dass alle drei Advertising-Kanäle bei jedem Advertising-Event verwendet werden sollen [19, S. 186]. Dabei muss ein Advertising-Paket mindestens folgende Daten enthalten: TX Power Level, lokaler Name (ohne : und ;), Flags und den Identifikator des primären Dienstes des Geräts [19, S. 186f.]. Eine weitere Anforderung ist, dass die Advertising-Intervalle zunächst 20 ms für die ersten 30 Sekunden lang sind und danach auf andere Intervalle umgeschalten werden sollen, welche in der Tabelle [19, S. 187] stehen. Des Weiteren dürfen keine speziellen Berechtigungen benötigt werden, um Dienste und Merkmale eines Gerätes zu entdecken [19, S. 190]. Auch soll auf den **BLE**-Geräten der Geräteinformationsdienst implementiert sein, damit der Herstellername, die Modellnummer, die Firmwareversion und die Softwareversion ausgelesen werden können [19, S. 191]. Ebenso sollte Zubehör im **GATT**-Profil das Merkmal *Gerätename* implementiert haben und durch Apple-Geräte beschreibbar sein [19, S. 190]. Als weitere Anforderung ist zu nennen, dass die Datenpacketlängenerweiterung vorhanden sein sollte, damit der Datenteil eines Pakets statt 27 Byte 251 Byte lang sein kann [19, S. 189]. Die letzte Anforderung ist die Fähigkeit von **BLE**-Geräte private Gerätadressen auflösen zu können.[19, S. 189].

Apple-Geräte geben nicht alle **BLE**-Dienste an Drittanbieter-Apps weiter, sondern verarbeiten diese intern und geben daraufhin die verarbeiteten Daten an die Drittanbieter-Apps weiter. Die heraus gefilterten Dienste sind: **GAP**, **GATT** sowie **BLE HID**. [19, S. 192]

3.2.4 **HID over GATT Profile (HOGP)**

In diesem Abschnitt der Arbeit wird nur auf die Anforderungen eines **HID**-Geräts (Bereitstellung eines **GATT**-Servers [20, S. 9]) und nicht eines **HID**-Hosts (Bereitstellung eines **GATT**-Clients [20, S. 9]) eingegangen, da eine Implementierung des **HID**-Hosts in diesem Projekt nicht benötigt wird.

Mittels des **HOGP**s werden Prozeduren und Fähigkeiten definiert, welche ein **BLE-HID** fähiges Gerät benötigt, um als **HID**-Gerät von **HID**-Hosts wahrgenommen zu werden. Das Profil baut dabei auf der USB **HID** Spezifikation auf. [20, S. 9]

Das **HID over GATT Profile (HOGP)** benötigt weitere Profile und Dienste, welche auf einem **HID**-Gerät implementiert sein müssen. Dazu zählen das **GATT**, der Batteriedienst, der Geräteinformationsdienst, das Scan Parameters Profil und der **HID** Dienst. Dabei ist zu beachten, dass auf einem **HID**-Gerät ein oder mehrere Instanzen des **HID**-Dienstes und des Batteriedienstes, sowie nur eine Instanz des Geräteinformationsdienstes und optional eine Instanz des Scan Parameters Dienstes vorhanden sein darf. [20, S. 9, S. 11] In Abbildung 7 sind alle benötigten und optionalen Dienste grafisch dargestellt. Optionale Dienste werden dabei durch eine gestrichelte Linie angedeutet.

Im **HID over GATT Profile** werden für alle benötigten Dienste und Profile zusätzliche Bedingungen festgelegt, welche in den folgenden Unterkapiteln bei dem jeweiligen Dienst beziehungsweise Profil dargestellt sind.

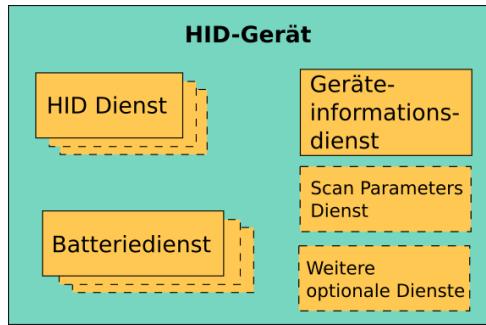


Abbildung 7: Benötigte Dienste eines HID-Geräts; abgewandelt von [20, S. 11]

HID-Dienst

Der **HID-Dienst** ist auf **HID-Geräten** zuständig für die Bereitstellung aller benötigten Daten für einen **HID-Host**. Dabei ist zu beachten, dass alle gespeicherten Merkmale des **GATT-Servers** mit dem niederwertigsten Oktett zuerst übertragen werden müssen. Der Dienst für den standardkonformen Betrieb muss zudem mindestens die Merkmale *Report Map*, *HID Information* und *HID Control Point* implementieren. [21, S. 8ff.]

Report Merkmal

Das Merkmal *Report* ist optional – jedoch ist es ein wichtiges Merkmal, da der Datentransfer zwischen **HID-Gerät** und **HID-Host** hauptsächlich über dieses Merkmal stattfindet. Das Merkmal *Report* kann dabei einen von drei Typen annehmen, nämlich Eingabe, Ausgabe oder Feature. Diese Typen finden sich ebenso in der USB **HID** Spezifikation wieder. [21, S. 11f.]

Da ein **HID-Gerät** mehrere Reports haben kann, muss für jeden Report ein eigenes Merkmal erstellt werden. Zur Unterscheidung der verschiedenen Reports muss jeweils ein Referenz-Merkmalsdeskriptor hinzugefügt werden, welcher eine eindeutige Report-ID und den Report-Typen enthält. Als zusätzliche Bedingung muss in allen Report-Merkmalen vom Typ Eingabe ein Konfigurationsdeskriptor vorhanden sein. Mittels dieses Deskriptors kann konfiguriert werden, ob bei Änderung des Merkmalswerts der **HID-Host** informiert werden soll oder nicht. Der Konfigurationsdeskriptor ist dabei verpflichtend hinzuzufügen. [21, S. 14.f]

Report Map Merkmal

Im Merkmal *Report Map* wird der USB Report Deskriptor abgespeichert (wie in der USB **HID** Spezifikation definiert [20, S. 21]), welcher den Aufbau und die Formatierung der einzelnen Report-Merkmale enthält [21, S. 11]. Pro **HID-Dienst** darf jeweils nur eine Instanz dieses Merkmals vorhanden sein und die maximale Größe ist auf 512 Oktette beschränkt [21, S. 16]. Mittels des zusätzlich benötigten *Report Referenz-Merkmalsdeskriptors* ist es den **HID-Hosts** möglich, die Informationen des *Report Map* Merkmals mit den *Report* Merkmalen zu verknüpfen [21, S. 17].

HID Information Merkmal

Dieses Merkmal enthält eine Ansammlung von Informationen, welche **HID** spezifische Werte sind. Zwei beispielhafte Werte ist zum einen der Wert *bcdHID*. Dieser wird verwendet, um

den **HID**-Host anzuzeigen, welche USB-Spezifikation im **HID**-Gerät implementiert wurde. Zum anderen gibt es den Wert *bCountryCode*. Mit diesem Wert wird angegeben, für welches Land das **HID**-Gerät entwickelt wurde. Da Geräte meist nicht für ein spezielles Land entwickelt werden, steht dieser Wert häufig auf 0x00. Das **HID Information** Merkmal darf pro **HID** Dienst nur einmal vorkommen und die Daten müssen statisch sein. [21, S. 20f.]

HID Control Point Merkmal

Dieses Merkmal wird von **HID**-Hosts verwendet, um dem **HID**-Gerät anzuzeigen, dass sich der **HID**-Host in den Schlafmodus oder in den normalen Betrieb versetzt. Dieses Merkmal darf nur einmal pro **HID** Dienst vorkommen. [20, S. 23], [21, S. 21]

Zusätzliche Bedingungen durch das HID over GATT Profile

Alle Merkmale, die in dem *Report Map* Merkmal beschrieben sind und nicht im **HID** Dienst enthalten sind, sollen mittels eines *Includes* in der **HID** Dienst Definition referenziert werden. Zusätzlich müssen alle referenzierten Merkmale den *Report Referenz* Merkmaldeskriptor enthalten. Alle **HID**-Dienste müssen als primärer Dienst initialisiert werden und während der Entdeckungsphase für einen Verbindungsaufbau als möglicher Dienst angegeben werden. [20, S. 13f.]

Batteriedienst

Mittels dieses Diensts wird dem **GATT**-Host der aktuelle Batteriestatus einer oder mehrerer Batterien des **GATT**-Servers bereitgestellt. Dabei gilt es zu beachten, dass alle bereitgestellten Merkmale des **GATT**-Servers mit dem niederwertigsten Oktett zuerst übertragen werden. [22, S. 6]

Für diesen Dienst muss ein Merkmal mit dem Namen *Battery Level* implementiert werden. Der Batteriestand wird dabei als ein Prozentwert zwischen 0 und 100 angegeben. Wobei 100 % einer voll aufgeladenen Batterie entspricht. Zusätzlich kann das Merkmal so eingerichtet werden, dass der **GATT**-Server den **GATT**-Client informiert, sobald sich der Wert geändert hat. [22, S. 8]

Zusätzliche Bedingungen durch das HID over GATT Profile

Es muss mindestens ein Batteriedienst als primärer Dienst auf dem **HID**-Gerät laufen. Falls ein Batteriestandsmerkmal Bestandteil des *Report Map* Merkmals ist, muss der Dienst mittels eines *Include* in der **HID** Dienst Definition referenziert werden. [20, S. 14]

Geräteinformationsdienst

Dieser Dienst stellt einem **GATT**-Client Informationen über den Hersteller und Anbieter des **GATT**-Servers bereit. Dabei darf jedes verfügbare Merkmal nur einmalig pro Dienst vorkommen. Zu beachten ist, dass alle Merkmale optional sind. [23, S. 6ff.]. In Tabelle 2 sind alle vorhandenen Merkmale mit einer kurzen Beschreibung aufgelistet.

Tabelle 2: Liste der verfügbaren Geräteinformationsmerkmale

Merkmalname	Kurzbeschreibung
Herstellername	Enthält den Namen des Herstellers [23, S. 8]
Modellnummer	Enthält die Modellnummer des Geräteanbieters [23, S. 8]
Seriennummer	Enthält die Seriennummer des Geräts [23, S. 8]
Hardwareversion	Enthält die Hardwareversion [23, S. 9]
Firmwareversion	Enthält die Firmwareversion [23, S. 9]
Softwareversion	Enthält die Softwareversion [23, S. 9]
System-ID	Enthält eine Kombination aus organisatorischer UID und Hersteller definierte ID. Diese ID ist eindeutig für jedes Gerät eines Produkts. [23, S. 9]
IEEE 11073-20601 Regulatory Certification Data Liste	Enthält eine Liste aller Regulations- und Zertifizierungsfromationen des Produkts [23, S. 9]
PNP-ID	Enthält eine eindeutige Geräte-ID. Diese besteht aus der Anbieter-ID-Quelle (Angabe, ob die Anbieter-ID durch Bluetooth SIG oder USB Implementer's Forum festgelegt wurde), der Anbieter-ID, der Produkt-ID (von Anbieter festgelegt) und einer Produktversion. Die Produktversion wird als binär-kodierte Dezimalzahl dargestellt. Zum Beispiel Version 2.13 = 0x0213 [23, S. 10f.]

Zusätzliche Bedingungen durch das **HID over GATT Profile**

Der Dienst muss als primärer Dienst gestartet werden und muss das *PNP-ID* Merkmal enthalten. [20, S. 14f.]

Scan Parameters Profil

Mittels dieses optionalen Profils beziehungsweise Diensts stellt ein **GATT**-Server einem **GATT**-Client Informationen zur Verfügung, die die Geräte unterstützen bei der Verwaltung von Verbindungszeitüberschreitungen und den Advertising-Paketen. Durch diese Informationen kann der Stromverbrauch sowie die Wiederverbindungslatenz optimiert werden. [24, S. 6]

3.2.5 Bluetooth-Stacks

Damit die Verwendung von Bluetooth auf dem verwendeten Mikrocontroller einfacher ist, bietet das Espressif IoT Development Framework (**ESP-IDF**) zwei Bluetooth-Stacks an. Zum einen den Stack Bluedroid, welcher **BBR** und **BLE** unterstützt. Zum anderen wird der Stack NimBLE bereitgestellt, welcher nur **BLE** unterstützt. [25]

Bluedroid ist ein von Broadcom Corporation bis 2012 entwickelter Bluetooth-Stack, welcher unter der Apache Lizenz steht [26]. Dieser Bluetooth-Stack wird von Google seit der Android Version 4.2 als Bluetooth Stack verwendet [27]. Heutzutage wird dieser Bluetooth-Stack weiterhin unter Android verwendet, mit dem Namen Fluoride [28], [29].

Apache NimBLE ist ein Open Source BLE-Stack, welcher vollständig mit der Bluetooth 5 Spezifikation konform ist [30] und Bestandteil des Apache Mynewt project ist [31].

3.3 Übertragungsprotokolle am Fernbedienungsmodulschacht

Die Multikopter-Fernsteuerungssoftware OpenTX bietet eine Vielzahl von verschiedenen Übertragungsmöglichkeiten zwischen der Fernsteuerung und den Modulen im Modulschacht. Die unterstützten Übertragungsmöglichkeiten sind hierbei: PPM, ACCST(D12, D8, LR12)[32], DSM, MULTI, ein Protokoll für R9M-Erweiterungsmodul [33] und SBUS. [34]

Zu beachten bei der Übertragung mittels des PPM-Ausgabepins am Modulschacht ist, dass dieser mittels der Batteriespannung der Fernbedienung betrieben wird, worüber auch das SBUS-Protokoll übertragen wird. Dadurch sollte ein Modul nur betrieben werden, wenn ein Spannungsteiler oder ein Pegelumsetzer vorhanden ist. [35]

Für die vorhandenen Übertragungsprotokolle ACCST, DSM und für das Protokoll der R9M-Erweiterungsmodul konnte keine Dokumentation gefunden werden, weshalb im folgenden Unterabschnitten nur die Übertragungsprotokolle PPM, CRSF, SBUS und MULTI betrachtet werden.

3.3.1 Puls-Positions-Modulation (PPM)

PPM ist ein Modulationsverfahren, womit Daten – auch Kanäle genannt – als Zeitdauer zwischen zwei steigenden Flanken zweier Impulse dargestellt werden - zu sehen in Abbildung 8. Ein Paket besteht dabei aus $n+1$ Impulsen, wobei n die Anzahl an Kanälen ist. Nach jedem Paket erfolgt eine Pause von 12 ms, damit eine Synchronisation zwischen Sender und Empfänger gegeben ist. Die Anzahl an Kanälen im Modellbau ist bei PPM auf acht Kanäle beschränkt. [36]

Die Gesamtlänge eines Pakets beträgt 22,5 ms mit inkludierter Pause. Die Pulse haben dabei eine Länge zwischen 0,7 ms bis 1,7 ms. [37]

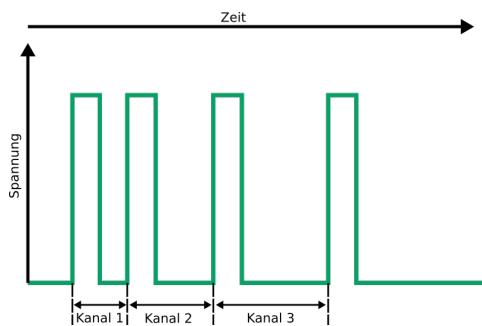


Abbildung 8: Darstellung einzelner Kanäle im Zeitverlauf bei PPM; abgewandelt von [36]

3.3.2 CRSF

Das CRSF-Protokoll verwendet eine Eindrahtleitung für eine halbduplex Universal Asynchronous Receiver Transmitter ([UART](#))-Verbindung. Über diese Verbindung sendet der Master alle 6 ms ein Paket. Zwischen den Paketen des Masters kann der Slave den Master optional antworten. Optional zu Beginn eines Pakets gibt es ein Synchronisationsbyte mit den Daten 0xC8. [38], [39]

Die Symbolrate beträgt 420000 Baud bei einer 8N1 Übertragung, womit die Übertragungsrate 46 KByte/s entspricht. Die Übertragung findet invertiert statt (hohes Spannungsniveau = logisch niedrig [40]) und Daten werden im Big Endian-Format übertragen. [38]

Ein Paket hat eine maximale Größe von 64 Byte und besteht aus fünf Teilen, welche in Tabelle 3 zu sehen sind. [38]

Tabelle 3: Aufbau eines CRSF-Pakets [38]

Feld-Index	Feldtyp	Größe in Byte
0	Geräteadresse	1
1	Paketlänge	1
2	Typenfeld	1
3-62	Daten	maximal 60
63	Cyclic redundancy check (CRC)-Feld	1

Die Geräteadressen sind im CRSF-Protokoll vordefiniert. Ein Ausschnitt davon ist in Tabelle 4 zu sehen.

Tabelle 4: Ausschnitt aus den vorhandenen CRSF-Geräteadressen [39]

Empfängergerät	Adresse
CRSF Fernsteuerung	0xAE
CRSF Empfänger	0xCE
CRSF Sender	0xEE
CRSF Multikopterplatine	0x8C

Im Paketlängenfeld wird die Größe des Pakets in Byte angegeben, wobei das Typenfeld, das Datenfeld und das [CRC](#)-Feld in die Größe des Pakets mit einfließen. [38]

Im CRSF-Protokoll werden ebenso wie die Adressen, die Typen fest definiert. Ein Ausschnitt möglicher Typen ist in Tabelle 5 zu finden.

Tabelle 5: Vordefinierte CRSF Datentypen [39]

Datentyp	Typenwert
Batteriesensor	0x08
Verbindungsstatistiken	0x14
Kanaldaten	0x16
Weitere Datentypen auf der nächsten Seite	

Datentyp	Typenwert
Multikopterflugmodus	0x21

Der Datenteil eines Pakets wird in 16 Kanäle aufgeteilt. Jeder Kanal ist dabei 11 Bit groß. Der Wertebereich pro Kanal liegt zwischen 172 und 1811. [38]

Die CRC-Bildung erfolgt über das Typenfeld und das Datenfeld eines Pakets [38]. Das Generatorenpolynom lautet dafür: 0xD5 [41]

3.3.3 SBUS

Das SBUS-Protokoll ist ein invertiertes serielles Protokoll, welches über eine Leitung mit einer Symbolrate von 100000 Baud übertragen wird. Die Daten werden dabei im 8E2-Format übertragen. Die Länge eines SBUS-Pakets beträgt 25 Byte, mit welchem 18 Kanäle übertragen werden. [42]

Das Intervall zum Versenden von SBUS-Paketen kann in OpenTX zwischen 6 ms und 40 ms frei eingestellt werden. Der Aufbau eines Pakets ist in Tabelle 6 zu sehen.

Tabelle 6: Paketaufbau von SBUS [42]

Byte	Verwendungszweck
0	Kopf des Pakets. Immer 0x0F
1 - 22	16 Kanäle mit jeweils einer Größe von 11 Bit
23 Bit 0	Digitaler ein/aus Kanal 17
23 Bit 1	Digitaler ein/aus Kanal 18
23 Bit 2	Paketverlustanzeige. Anzeige, wenn ein Paket zwischen Sender und Empfänger verloren geht.
23 Bit 3	Paketausfallanzeige. Anzeige, wenn mehrere hintereinander verschickte Pakete zwischen Sender und Empfänger verloren gehen.
24	Paketfluss. Immer 0x00 [40]

Für die Synchronisation zwischen Sender und Empfänger gibt es Lücken innerhalb eines Pakets [40]. Ebenso gibt es eine weitere Version von SBUS, welche den Namen *Fast SBUS* hat und Daten mit einer Symbolrate von 200000 Baud überträgt [42]. Der Wertebereich der Kanäle 1 bis 16 liegt zwischen 172 und 1811 und kann auf den Wertebereich von 0 bis 2047 erweitert werden, um die vollen 11 Bit auszunutzen [42].

3.3.4 MULTI

Das MULTI-Protokoll wird für die Kommunikation zwischen einer Fernsteuerung, auf der OpenTX läuft, und einem 2,4 GHz Erweiterungsmodul verwendet. Mittels dieses Protokolls wird zum einen das Erweiterungsmodul konfiguriert und zum anderen werden die Daten der 16 zu übertragenden Kanäle an das Erweiterungsmodul geschickt. [43], [44]

Die Übertragung findet dabei seriell mittels des 8E2-Formats statt und mit einer Symbolrate von 100000 Baud. In Version 1 ist die Länge eines Pakets 26 Byte. In Version 2 ist die Länge eines Pakets zwischen 27 und 36 Byte groß. [44]. In Tabelle 7 ist der Aufbau eines MULTI-Pakets dargestellt.

Tabelle 7: Paketaufbau von MULTI [44]

Byte	Verwendungszweck
0	Paketkopf mit der Angabe welche Art von Daten übermittelt werden.
1	Das zu verwendende Protokoll, welches das Erweiterungsmodul zum Versenden der Daten verwenden soll.
2	Informationen über den Verbindungszustand zwischen dem Erweiterungsmodul und einem Empfänger sowie das zu verwendende Subprotokoll.
3	Optionale Protokollauswahl, welche nicht vordefiniert ist.
4 - 25	Daten der Kanäle oder die Daten, welche bei einem Paketausfall versendet werden.
26	Weitere Protokollinformationen und Telemetriedaten.
27 - 35	Weitere Möglichkeiten zur Protokolldefinition.

Ab hier überarbeiten

3.4 Mikrocontrollerfamilie ESP32 und Zertifizierungen

Die Mikrocontrollerfamilie ESP32 wird von Espressif Systems entwickelt und ist in mehreren Bauformen erhältlich. Entweder kann die ESP32-Familie als Chip, Platinenmodul oder Entwicklungsplatine erworben werden [45], [46], [47]. Als Besonderheit unterstützt dieser Mikrocontroller 2,4 GHz WLAN und Bluetooth sowie in einigen Modellen die Zweikern-CPU [48], [49, S. 8], [49, S. 24]. Die Kerne des Mikrocontrollers basieren dabei auf den Xtensa 32-Bit LX6 Kernen – mit Harvard Architektur – und haben eine maximale Frequenz von 240 MHz [48], [49, S. 9], [49, S. 8], [49, S. 24]. Der ESP32 verfügt neben der Funkkommunikationseinheit über weitere Hardwarekomponenten. Beispielsweise über eine Fließkommaeinheit, einen 12-Bit successive approximation (SAR) ADC, einige programmierbare general purpose input/outputs (GPIOs) mit konfigurierbaren Pull-Up- und Pull-Down-Widerständen sowie über mehrere weitere kabelgebundene Kommunikationseinheiten [49, S. 10f.], [49, S. 23], [49, S. 34]. Zusätzlich sind viele Platinenmodule von Espressif Systems nach der Federal Communications Commission (FCC) [50] und conformité européenne (CE) regulatorisch zertifiziert [51]. Die Entwicklung des ESP32 Mikrocontrollers kann durch viele verschiedene integrated development environments (IDEs) erfolgen, wobei alle auf dem Entwicklungsframeworks ESP-IDF aufbauen [48], [52].

Die FCC gibt verpflichtende Richtlinien für elektronische Geräte in den Vereinigten Staaten von Amerika heraus, um die Interferenzen zwischen elektronischen Geräten zu minimieren und die Geräte für den Endnutzer sicher zu gestalten. Falls auf einem Endprodukt das FCC-Zeichen vorhanden ist, ist dieses Produkt nach FCC-Vorgaben konform. [50]

Ebenso gibt es für die Europäische Union Richtlinien, welche für Endkundenfunkgeräte gelten [53], [54]. Diese Regularien sind in der Radio Equipment Directive (RED) festgeschrieben und können durch Hersteller des Produktes selbstständig überprüft werden und ermächtigt den Hersteller nach erfolgreichen Prüfungen zur Anbringung des CE-Zeichens [55, S. 14]. Die CE-Zertifizierung ist dabei nur für die alleinstehenden ESP32 Platinenmodule gültig und muss im eingebauten Zustand erneut zertifiziert werden [56].

3.5 FreeRTOS

FreeRTOS ist ein von der Real Time Engineers Ltd. entwickelter Echtzeitkernel für die parallele Ausführung von mehreren Threads (in FreeRTOS Tasks genannt), welcher unter der MIT-Lizenz publiziert wird [57, S. 2], [58], [59]. Der Kernel enthält einen Echtzeitscheduler, Komponenten für die Inter-Task-Kommunikation, Primitive für die zeitliche Koordination und Komponenten zur Synchronisation von Threads [58]. Eine Besonderheit des Kernels ist die geringe Speichergröße, welche normalerweise zwischen 6 KB und 12 KB liegt und in der Grundausführung aus 3 Dateien besteht [60], [61]. Erweiterungen des Kernels können nach Bedarf nachgeladen werden [61].

3.5.1 Tasks

Für das Scheduling von Tasks verwendet FreeRTOS ein preemptives Scheduling mit fixen Prioritäten. Falls mehrere Tasks dieselbe Priorität haben, wird Round-Robin Time-Slicing verwendet. Beim Time-Slicing wechselt der Scheduler bei jedem Tick-Interrupt zwischen den Tasks mit derselben Priorität. Ein Nachteil des Schedulingverfahrens ist, dass Tasks mit einer niedrigen Priorität verhungern können und der Programmierer während der Programmierung der Software darauf Acht geben muss. [62], [63]

Im FreeRTOS-Kernel werden Zeiten in Ticks gemessen. Dafür zählt ein Timer-Interrupt – auch Tick-Interrupt genannt – den sogenannten Tickzähler hoch, weshalb die Kernel-Zeitauflösung abhängig vom Interruptintervall des Tickzählers ist. Nach jedem Tick – auch TimeSlice genannt – überprüft der Kernel, ob ein Task in den blockierenden Zustand gesetzt oder aufgeweckt werden soll und entscheidet darauffolgend, welcher Task als Nächstes ausgeführt werden soll. [57, S. 61], [64]

Tasks müssen normalerweise während der gesamten Laufzeit des Mikrocontrollers laufen. Dies wird durch die Verwendung einer Endlosschleife, wie im Quellcode 2 zu sehen ist, umgesetzt. Vor der Endlosschleife werden alle benötigten Variablen deklariert. Nach der Endlosschleife wird als Sicherheitsmaßnahme der Task aus dem RAM gelöscht, damit keine Rückstände des Tasks im RAM bleiben. [57, S. 46]

Quellcode 2: Minimaler Aufbau eines Tasks; angepasst von [57, S. 47]

```
void ATaskFunction (void *Parameters)
{
    /* Block fuer die Deklaration von Variablen und einmalig
       auzufuehrenden Code */
```

```

for(;;)
{
    /* Code der endlos ausgefuehrt werden soll */
}
/* Speicher leeren in Ausnahmefaelen */
vTaskDelete(NULL);
}

```

Wie in Abbildung 9 zu sehen ist, gibt es vier Taskzustände. Ein Task befindet sich im Running-Zustand, wenn der Task ausgeführt wird. Die restlichen drei Zustände werden unter dem sogenannten Not-Running-Zustand zusammengefasst. Dieser Zustand tritt auf, sobald ein Task nicht für den Scheduling vorhanden ist (Suspended-Zustand), der Task auf ein Event wartet (Blocked-Zustand) oder ein Task auf die Ausführung wartet (Ready-Zustand). [57, S. 55], [57, S. 65f.]

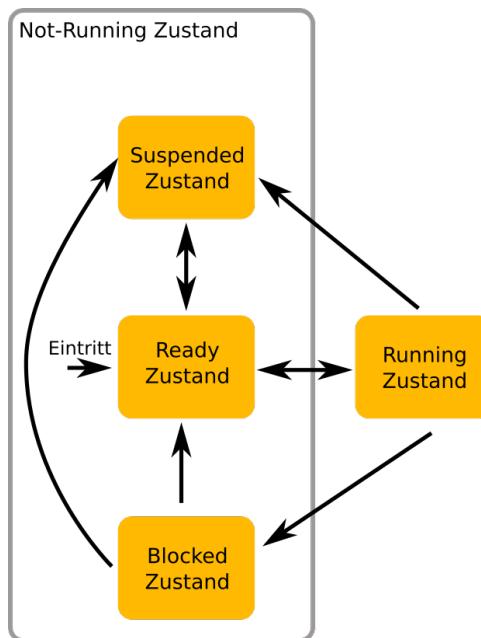


Abbildung 9: Taskzustände und deren Übergänge; angepasst von [57, S. 67]

Der sogenannte Idle-Task ist ein spezieller Task, welcher in FreeRTOS verwendet wird, wenn es keinen Task im Running-Zustand gibt, da in FreeRTOS immer ein Task im Running-Zustand sein muss. Der Idle-Task hat die niedrigst mögliche Priorität, damit der Task von allen anderen Tasks verdrängt werden kann. [57, S. 76]

3.5.2 Software Timer

Mittels Software Timern können Softwarefunktionen zu einem bestimmten Zeitpunkt in der Zukunft (one-shot Timer) oder in periodischen Intervallen (auto-reload Timer) ausgeführt werden. Ausgeführte Funktionen der Software Timer können nicht in den blockierenden Zustand wechseln und das System blockieren, bis die Funktion vollständig durchlaufen ist. [57, S. 149], [57, S. 151], [65]

3.5.3 Queues

Die primäre Form für die Inter-Task-Kommunikation in FreeRTOS findet mittels Queues statt. Queues können eine vordefinierte Anzahl an Elementen mit fixer Größe speichern. Die Größe eines Elements und die maximale Anzahl an Elementen in einer Queue müssen bereits während der Erstellung der Queue durch den Kernel vorgegeben werden. Die Kommunikation mittels Queues kann dabei zwischen Tasks oder zwischen Tasks und Interrupts erfolgen und ist nicht auf eine vordefinierte Anzahl an Teilnehmern begrenzt. Queues werden meist als First In First Out ([FIFO](#)) Buffer verwendet, es können jedoch auch Elemente am Start des Buffers angehängt werden. Elemente werden in einer Queue nicht als eine Referenz gespeichert, sondern das Element wird vollständig kopiert. [57, S. 104ff.], [66]

3.5.4 Interrupts

FreeRTOS unterstützt die Verwendung von Interrupts mit einer Einschränkung. Es dürfen nämlich nur FreeRTOS Funktionen in der Interrupt-Routine verwendet werden, welche die Endung *FromISR* haben. Auch müssen die Interrupt-Routinen zeitlich so kurz wie möglich gehalten werden, damit das Scheduling der Tasks weiter ausgeführt werden kann. Um eine möglichst kurze Ausführungszeit zu erreichen, kann das Verfahren *aufgeschobene Interrupts* verwendet werden. Dabei werden nur die benötigten Daten durch den Interrupt gesammelt und an einen Task weitergeleitet, welcher die Daten auswertet. [57, S. 185], [57, S. 195]

3.5.5 Besonderheiten bei der Verwendung auf dem ESP32

Das [ESP-IDF](#) bietet für den ESP32 Mikrocontroller eine angepasste Version des FreeRTOS an. Eine FreeRTOS-Anpassung ist die Unterstützung für symmetrisches Multiprocessing, da viele ESP32 über eine Zweikern-[CPU](#) mit symmetrischen Multiprocessing verfügen. Dabei wird zwischen dem [CPU](#)-Kern 0 (Protokoll [CPU](#)) und dem [CPU](#)-Kern 1 (Applikation [CPU](#)) unterschieden. Die Namensgebung der [CPU](#)-Kerne weist auf die präferierte Taskaufteilung hin. Um Tasks [CPU](#)-Kernen fest zuzuweisen, gibt es zusätzliche Funktionen in FreeRTOS. Als weitere Anpassungen enthält jeder CPU-Kern einen eigenen Idle-Task und die [CPU](#) 0 ist allein für die Ausführung der Tick-Interrupt-Routine zuständig. [67], [68]

Zudem wird der FreeRTOS-Kernel automatisch mit einem Task für die Inter-Processor-Aufrufe gestartet. Durch Inter-Processor-Aufrufe kann ein aufrufender [CPU](#)-Kern, Funktionen auf einem anderen [CPU](#)-Kern aufrufen. [69], [68]

3.6 Darstellung von Glyphen auf einem Monitor

Für die Darstellung von Zeichen auf einem Monitor gibt es mehrere Verfahren. Im nachfolgenden werden die drei meist verwendeten Verfahren beschrieben.

Als erstes Verfahren sind die Raster- beziehungsweise Bitmap-Schriftarten zu nennen [70]. Rasterschriftarten stellen die erste Generation von digitalen Schriftarten dar, da diese nur

eine geringe Rechenleistung für die Darstellung von Glyphen benötigen und sich dadurch für Rechner mit geringer Rechenleistung gut eignen. Eine Glyphe für ein Zeichen wird dabei durch ein Rechteck von Pixeln definiert und jedem Pixel wird ein fester Farbwert zugewiesen. Das Rechteck an Pixeln kann durch ein Byte-Array zeilen- oder spaltenweise effizient gespeichert werden, wie in Abbildung 10 zu sehen ist [71]. Die Vorteile von Raster-Schriftarten liegen in der geringen Rechenleistung zur Darstellung von Glyphen als auch in der guten Optimierung der Glyphen für eine bestimmte Schriftgröße. Der Nachteil bei Raster-Schriftarten ist jedoch, dass Glyphen für eine bestimmte Schriftgröße und Monitorauflösung definiert sind und dadurch geräteabhängig sind. Werden mehrere Größen und Monitorauflösungen unterstützt, wird viel Speicherplatz pro Schriftart benötigt, da für jede Größe und Monitorauflösung alle Glyphen erneut definiert werden müssen. [72], [70]

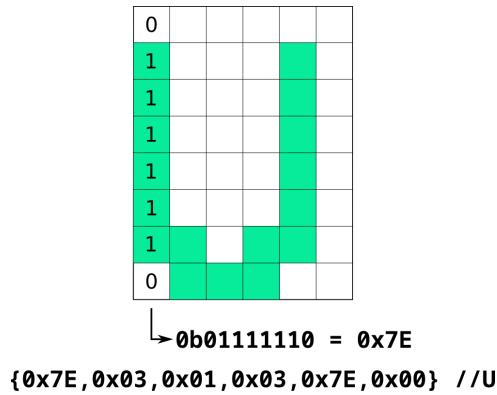


Abbildung 10: Speichern der Glyphe *U* spaltenweise in einem Array aus Bytes; abgewandelt von [71]

Das zweite Verfahren sind die Vektor-Schriftarten, bei denen einzelne Glyphen als Sammlung von Vektoren dargestellt werden [70]. Kurven in einer Glyphe werden durch viele kurze Vektoren approximiert [73, S. 24]. Durch die Verwendung von Vektoren können Glyphen mit einem geringeren Speicherplatz als Raster-Schriftarten geräteunabhängig skaliert werden [72], [70]. Jedoch ist die Darstellung der Glyphen langsamer und rechenintensiver als bei Raster-Schriftarten [70], [74].

Die Umwandlung von Vektor-Schriftarten findet nämlich in zwei Schritten statt. Der erste Schritt ist die Generierung der Umrandung der Glyphen aus den Schriftartdaten. Der zweite Schritt ist die Konvertierung der generierten Umrandungen in einzelne Pixel, indem berechnet wird, ob ein Pixel innerhalb oder außerhalb der Umrandung liegt und passend dazu eingefärbt wird. [72] Während der Umwandlung können ungewollte Artefakte entstehen, da es bei bestimmten Monitorauflösungen oder Schriftgrößen passieren kann, dass die Umrandung der Glyphen nur teilweise einzelne Pixel einschließen. Zu sehen ist das Problem in Abbildung 10.

Das letzte Verfahren sind Truetype- beziehungsweise Opentype-Schriftarten. Diese bauen auf dem Verfahren der Vektor-Schriftarten auf und erweitern die Daten der Glyphen um Kurven und zusätzliche Hinweise, um weitere Optimierungen während der Umwandlung zu erzielen. [70]

In Abbildung 12 sind die drei vorgestellten Schriftartenspeicherverfahren nebeneinandergestellt.

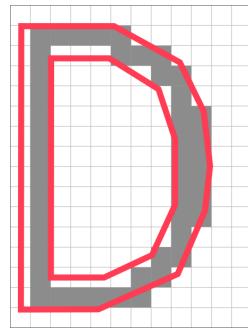


Abbildung 11: Umwandlung einer Vektor-Schriftart zur Darstellung an einem Monitor; abgewandelt von [72]



Abbildung 12: Gegenüberstellung der Schriftartenspeicherverfahren

3.7 Eingabeereignisbehandlung unter Linux in Bezug auf Gamepads und Joysticks

Unter Linuxsystemen gibt es zwei Möglichkeiten die Ereignisse von Gamepads und Joysticks abzufragen. Zum einen die Joystick Treiber application programming interface ([API](#)). Dieses ist im Linuxkernel aus Kompatibilitätsgründen für ältere Software enthalten und sollte nicht mehr für neue Projekte verwendet werden. Zum anderen gibt es das sogenannte Event Device ([evdev](#)) [75], welches die Joystick Treiber [API](#) durch den größeren Funktionsumfang ersetzen und als präferierte Schnittstelle verwendet werden soll. [76], [77]

[evdev](#) ist eine generische Eingabeereignisschnittstelle für Programme im Userspace sowie für Konsumenten im Kernel, welche Ereignisse von Eingabegeräten benötigen. Eingabeereignisse werden dabei als C-Struktur dargestellt und haben den Aufbau wie in Quellcode 3 zu sehen ist. Das erste Strukturelement ist der Zeitstempel, der den Ereigniszeitpunkt im Kernel wiedergibt. Als zweites Element ist der Typ des Ereignisses hinterlegt, welcher beispielsweise für einen Tastendruck den Wert *EV_KEY* hat. Das nächste Element ist der Ereigniscode, welcher hardwareunabhängig ist. Eine vollständige Liste aller Ereigniscodes ist unter Quelle [78] zu finden. Das letzte Element eines Ereignisses ist der Ereigniswert, welcher unterschiedliche Bedeutungen abhängig vom Ereignistyp hat. Beispielsweise liegt der Wertebereich für einen Knopfdruck zwischen 0 und 3. Der Wert 0 stellt dabei ein Loslassen der Taste dar und der Wert 1 das Drücken einer Taste. Der Wert 3 stellt das automatische erneute Drücken bei längerem Tastendruck dar. [75], [77]

Quellcode 3: C-Strukuraufbau eines Eingabeereignisses von evdev [75]

```
struct input_event {
```

```
    struct timeval time;
    unsigned short type;
    unsigned short code;
    unsigned int value;
};
```

Der Umgang mit [evdev](#)-Kernelgeräten in der Programmiersprache C wird durch die Bibliothek *libevdev* vereinfacht. Diese Bibliothek abstrahiert den Zugriff auf die sogenannte [evdev](#) input output Control ([IOCTL](#)) [79] und stellt dadurch ein Bindeglied zwischen dem Linux-Kernel und den Programmen im Userspace dar. [80], [81]

[IOCTL](#) ist unter Linux der meist verwendete Weg, um zwischen Programmen und Gerätetreibern zu interagieren. Die Interaktion findet dabei durch character devices, block devices, sockets oder spezielle Dateideskriptoren statt. [82]

4 Umsetzung

4.1 Softwareentwicklung

In diesem Abschnitt wird das Zusammenspiel der Softwarekomponenten sowie einige Kernkomponenten detaillierter beschrieben und die Auswahl des Mikrocontrollers erklärt.

4.1.1 Auswahl der Mikrocontrollers

Als Mikrocontroller wird das kostengünstige ESP32-WROOM-32E-Modul – zirka 5 € [45] – verwendet, da es eine große Beliebtheit in Hobbyprojekten hat und es daraus folgernd eine große Community gibt die bei Problemen während der Entwicklung von Softwareprojekten helfen kann [83], [84]. Ebenfalls ist die [ESP-IDF](#) gut durch Kommentare im Sourcecode und durch Beispielprogramme dokumentiert. Als weiterer Grund für die Verwendung des ESP32 spricht, die Unterstützung mehrerer Bluetooth-Stacks, welche in Kapitel 3.2.5 beschrieben sind. Zuletzt ist noch zu nennen, dass der Mikrocontroller in kompakten Hardwaremodulen bezogen werden kann, in denen alle benötigten Elektronikkomponenten für den Betrieb des ESP32 enthalten sind, beispielsweise benötigte Kondensatoren, Widerstände und gegebenenfalls eine Antenne für den Betrieb von Bluetooth oder [WLAN](#) [85, S. 14]. Auch besitzen die ESP32-Module meist eine CE-Zertifizierung, wie in Kapitel 3.4 beschrieben, womit das zu entwickelnde Fernsteuerungsmodul vereinfacht zertifiziert werden kann.

4.1.2 Auswahl des Bluetooth-Stacks

Da die Kommunikation zwischen dem Fernsteuerungserweiterungsmodul und den Endgeräten mittels [BLE](#) erfolgen soll, wird der Bluetooth-Stack Apache NimBLE verwendet. Dieser wird bei der ausschließlichen Verwendung von [BLE](#) empfohlen, da dieser eine geringe Codegröße hat und wenig Speicher zu Laufzeit benötigt [25].

4.1.3 Kommunikation zwischen dem Mikrocontroller und dem Endgerät

Für die Übermittlung der Daten zwischen dem Mikrocontroller zu einem Endgerät wird [BLE](#) verwendet, indem die Fernsteuerungsdaten mittels [HOGP](#) verpackt werden. [BLE](#) eignet sich, wie im Kapitel 3.1.1 beschrieben, gut durch den geringen Stromverbrauch, da das Fernsteuerungsmodul durch den Akku der Fernsteuerung mitbetrieben wird. Durch die Verwendung des [HID](#)-Datenformats kann das Erweiterungsmodul, Daten für die Steuerung von Simulatoren an viele Endgeräte ohne zusätzlich benötigte Treiber übertragen [86]. Für die Datenübertragung werden die [BLE](#)-Dienste und Merkmale, welche in Tabelle 8 zu sehen sind, verwendet. Zusätzlich

enthält das Report-Merkmal einen Konfigurationsdeskriptor, womit sich Endgeräte für das automatische versenden von Daten abonnieren können.

Tabelle 8: Liste der verfügbaren Geräteinformationsmerkmale

BLE-Dienst	Merkmale
Geräteinformationsdienst	Herstellername
	Modellnummer
	Firmwareversion
	Softwareversion
Batteriedienst	Akkustand
HID-Dienst	Report-Map
	HID Information
	HID Control Point
	Report, definiert als Eingabe

Der finale Report-Map-Deskriptor ist für die Übertragung von Gamepaddaten konfiguriert. Die Daten bestehen dafür aus acht analogen und acht digitalen Kanaldaten, welche jeweils die absoluten Stellungen der Fernsteuerungseingaben darstellen. Die analogen Kanäle haben eine Größe von 16 Bit und haben einen Wertebereich von 0 bis 2047. Die ersten vier analogen Kanäle werden für die Übertragung der Steuernüppelpositionen verwendet. Die restlichen vier analogen Kanäle werden für die Stellung der Kippschalter mit jeweils drei Positionen verwenden. Die digitalen Kanäle haben jeweils eine Größe von 1 Bit und werden für Knöpfe mit 2 Stellungen verwendet. In Quellcode 4 ist der beschriebene Report Deskriptor zu sehen.

Damit das Erweiterungsmodul von Endgeräten als Gamepad, während des Verbindungsaufbaus, erkannt wird, enthalten die Advertising-Pakete ebenso Informationen über das Erweiterungsmodul.

Quellcode 4: Report Map Deskriptor des Erweiterungsmoduls

```

0x05 , 0x01 ,          // Usage Page (Generic Desktop Ctrl)
0x09 , 0x05 ,          // Usage (Game Pad)
0xA1 , 0x01 ,          // Collection (Application)
0x85 , 0x01 ,          // Report Id (1)
0xA1 , 0x00 ,          // Collection (Physical)
0x05 , 0x01 ,          // Usage Page (Generic Desktop Ctrl)
0x09 , 0x30 ,          // Usage (X)
0x09 , 0x31 ,          // Usage (Y)
0x09 , 0x32 ,          // Usage (Z)
0x09 , 0x33 ,          // Usage (Rx)
0x09 , 0x35 ,          // Usage (Rz)
0x09 , 0x34 ,          // Usage (Ry)
0x09 , 0x36 ,          // Usage (Slider)
0x09 , 0x36 ,          // Usage (Slider)
0x15 , 0x00 ,          // Logical Minimum (0)
0x26 , 0xFF , 0x07 ,  // Logical Maximum (2047)
0x75 , 0x10 ,          // Report Size (16)
0x95 , 0x08 ,          // Report Count (8)
0x81 , 0x02 ,          // Input (Absolute)
0x05 , 0x09 ,          // Usage Page (Button)

```

```

0x19, 0x01,          // Usage Minimum (0x01)
0x29, 0x08,          // Usage Maximum (0x08)
0x15, 0x00,          // Logical Minimum (0)
0x25, 0x01,          // Logical Maximum (1)
0x95, 0x08,          // Report Count (8)
0x75, 0x01,          // Report Size (1)
0x81, 0x02,          // Input (Absolute)
0xC0,                // End Collection
0xC0,                // End Collection

```

Zusätzlich zu den in Kapitel 3.2.3 beschrieben Sollanforderungen durch Apple, haben Geräte von Apple weitere nicht dokumentierte Anforderungen. Eine Anforderung ist, dass das Erweiterungsmodul Resolvable Private Address (**RPA**) auflösen können muss, da sonst kein Verbindungsauftbau zum Datenaustausch zwischen den Geräten stattfindet. Eine weitere nicht dokumentierte Anforderung ist, dass die Datenübertragung zwischen dem Erweiterungsmodul und dem Applegerät verschlüsselt stattfinden muss, wenn es sich um **HID**-Daten handelt. Dafür werden im ersten Schritt die Geräte gekoppelt, was durch verschiedene Verfahren erfolgen kann. Die Kopplung mit dem Erweiterungsmodul findet durch die Anzeige und der Bestätigung eines Pins auf beiden Geräten statt. Im zweiten Schritt erfolgt das Bonding, womit die Daten der Kopplung gespeichert werden, um bei einem erneuten Verbindungsauftbau die Kopplungsphase zu überspringen [87].

Für die Ermittlung des aktuellen Verbindnungszustands mit Endgeräten werden die auftretenden **GAP**-Events am Erweiterungsmodul ausgewertet. Ein verwendetes **GAP**-Event ist für die Kopplung der Geräte zuständig und enthält die Erstellung und Darstellung des Kopplungspin. Ein weiteres **GAP**-Event ist für das Abonnieren von Endgeräten zu **BLE**-Merkmale zuständig. Zuletzt wird das **GAP**-Event für den Verbindungsauftbau verwendet, um eine Anfrage an das Endgerät zu schicken, damit die Übertragungsrate für die Verbindung erhöht wird.

4.1.4 Kommunikationsprotokoll zwischen der Multikopterfernsteuerungen und dem Mikrocontroller

Als Kommunikationsprotokoll zwischen der Multikopterfernsteuerung und dem Mikrocontroller wird CRSF verwendet, da wie in Kapitel 3.3 beschrieben das Protokoll die höchste Übertragungsrate bei möglichst kleinen Datenpaketen hat.

Da die CRSF-Daten mittels einer **UART**-Verbindung übertragen werden, findet das Auslesen der Daten auf dem ESP32 mittels eines **UART**-Treibers statt. Der **UART**-Treiber abstrahiert dafür den vorhanden **UART**-Interrupt und stellt alle Ereignisse und Daten durch eine vereinfachte **API** bereit [88]. Die **UART**-Interrupt sind konfiguriert, das empfange Daten erst verarbeitet werden, wenn entweder der interne **UART**-Buffer voll ist oder ein definierter Timeout zwischen empfangenen Bytes auftritt.

Die Auswertung der empfangenen CRSF-Daten findet in zwei Schritten statt. Im ersten Schritt findet eine Überprüfung der Geräteadresse – 0xEE –, der Länge der übermittelten Daten und des übermittelten Datentyps – 0x16 – statt. Wenn all diese Werte stimmen findet die Überprüfung der **CRC**-Prüfsumme statt, um festzustellen, ob alle Kanaldaten gültig sind. Im zweiten Schritt, werden alle Kanaldaten zunächst ausgelesen und darauffolgend im Wertebereich angepasst,

damit der vollständige Wertebereich des **HID**-Reports verwendet wird. Dafür werden die analogen Kanaldaten auf einen Wertebereich von 0 bis 2047 erweitert und digitale Kanaldaten bis zu einem Wert von 992 als logisch 0 und ab einem Wert von 993 als logisch 1 ausgewertet. Nach Anpassung des Wertebereichs werden die Daten in einer globalen Datenstruktur abgelegt, welche in Quellcode 5 zu sehen ist. Falls dabei eine Änderung des Werts stattfindet, wird veranlasst, dass die Daten per **BLE** an abonnierte Endgeräte versendet werden. Falls ein Problem während der Auswertung stattfindet, wird das komplette Paket verworfen und auf ein neues Paket gewartet.

Quellcode 5: C-Strukuraufbau der aufbereiteten Kanaldaten

```
typedef struct ChannelDataStruct{
    uint16_t roll;           //roll = x
    uint16_t pitch;          //pitch = y
    uint16_t aux3;           //aux3 = z
    uint16_t yaw;            //yaw = rx
    uint16_t aux1;           //aux1 = rz
    uint16_t throttle;       //throttle = ry
    uint16_t aux4;           //aux4 = slide
    uint16_t aux2;           //aux2 = slide
    uint8_t buttons;          //buttons = aux12(b8) .. aux5(b1)
} ChannelDataStruct;
```

4.1.5 Statusausgabe des Mikrocontrollers mittels eines OLED-Displays

Wie in Kapitel 2.1 beschrieben, wird ein **OLED**-Display verwendet, um Statusinformationen für den Benutzer darzustellen. Dabei hat das Display eine Displaydiagonale von 0,91 Zoll, eine Auflösung von 128 zu 32 Pixel und den Displaycontroller SSD1306. Die Datenübertragung zwischen dem ESP32 und dem SSD1306 findet mittels inter integrated circuit (**I²C**) statt. Zur Ansteuerung einzelner Pixel im Display teilt der Displaycontroller das Display in vier Seiten mit jeweils 128 Segmente auf. Jedes Segment hat dabei eine Größe von 8 Bit. Zu sehen ist die Aufteilung des Displays in Seiten und Segmente, sowie die Pixelansteuerung durch einen Übermittelten Datenstream in Abbildung 13.

Die Darstellung von Text auf dem Display erfolgt mittels einer Bitmap-Schriftart, da dadurch mit geringeren Rechenaufwand eine gute Lesbarkeit auf dem kleinen Display erreicht wird. Alle Glyphen der Schriftart werden in einem zweidimensionalen Array gespeichert und haben eine Höhe von 14 und eine Breite von 9 Pixel. Die Umwandlung von Zeichenketten in Glyphen für das Display findet in zwei Schritten statt. Im ersten Schritt werden alle Pixelinformationen für das komplette Displaybild in einen lokalen 492 Byte großen Buffer des ESP32 zwischengespeichert. Jedes Bit dieses Buffers repräsentiert dabei ein Pixel des Displays. Dadurch müssen die Glypheninformation aus dem zweidimensionalen Glyphenarray nur an die richtige Position des Buffers übertragen werden. Im zweiten Schritt wird der lokale Puffer mittels **I²C** an das Display für die Darstellung übertragen. Die Aufteilung der Darstellung in zwei Schritte hat den Vorteil, dass Inhalte auf dem Display ohne Artefakte überlappt werden können, was nicht einfach möglich wäre, wenn nur die zu veränderten Pixel an das Display geschickt werden würden.

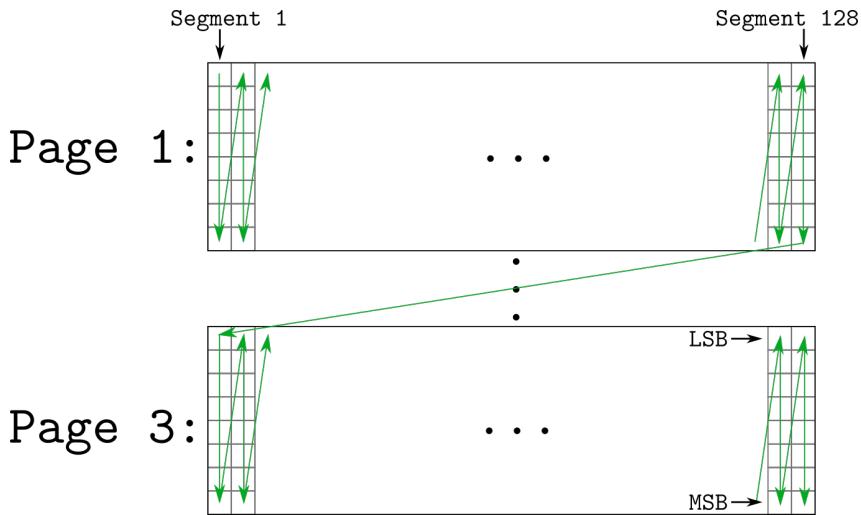


Abbildung 13: Verarbeitung eines Datenstreams für die Darstellung auf dem Erweiterungsmoduldisplay; abgewandelt von [89, S. 37]

4.1.6 Kombination aller Softwarekomponenten

Für die einfachere Integration von eigenen Softwarekomponenten mit bereits vorhandenen Bibliotheken, wird für die Studienarbeit die Programmiersprache C verwendet, da eine Vielzahl an Bibliotheken und Beispielprogrammen in C geschrieben sind [52]. Die finale Software des Mikrocontrollers besteht auf fünf Komponenten, welche mittels dem FreeRTOS-Echtzeitkernell verwaltet werden.

Die erste Komponente für die **BLE**-Kommunikation und dem **BLE**-Verbindungsaufbau läuft in einen FreeRTOS-Task (**BLE**-Task). Die zweite Komponente läuft ebenso in einen FreeRTOS-Task und kümmert sich um das Einlesen und Verarbeiten von CRSF-Daten von der Multikopterfernsteuerung (CRSF-Task). Die dritte Komponente, welche als FreeRTOS Software Timer integriert ist, kümmert sich um die Bestimmung des aktuellen Akkustands. Hierfür wird mittels des integrierten **ADC** des ESP32 die aktuelle Spannung des Multikopterfernsteuerungsakkus bestimmt und in einen Prozentwert zwischen 0 und 100 umgewandelt. Dieser Akkustand wird mittels **BLE** an das Endgerät weitergeschickt. Die vierte Komponente der ESP32-Software ist für die Erkennung von Tasteneingabeevents zuständig. Die Erkennung von Tasteneingaben erfolgt dabei mittels eines FreeRTOS-Interrupts und die Weitergabe der Daten an ein Task mittels FreeRTOS-Queues. Die letzte Softwarekomponente des ESP32, ist für die Steuerung des Displays vorhanden. Da Anzeigeänderungen am Display nur selten erfolgen und schnell abgearbeitet werden können, werden die Displayhilfsfunktionen innerhalb der zwei vorhandenen Tasks ausgeführt und nicht in einem zusätzlichen Task. Damit der ESP32-Mikrocontroller mit den zwei vorhandenen Kernen optimal ausgelastet wird, werden beide Tasks jeweils einem Kern fest zugewiesen, wodurch sich die einzelnen Softwarekomponenten im Betrieb seltener blockieren.

Zur Abspeicherung der Multikopterfernsteuerungsdaten und zum Datenaustausch zwischen CRSF-Task und **BLE**-Task werden globale Variablen verwendet. Die Mitteilung, dass neue Multikopterfernsteuerungsdaten ausgelesen wurden, findet mit zwei Funktionen von NimBLE statt. Zum einen durch die Funktion `ble_hs_mbuf_from_flat`. Mittels dieser Funktion werden die Fernsteuerungsdaten in einen internen **BLE**-Buffer geschrieben. Zum anderen werden durch

die Funktion `ble_gattc_notify_custom` alle gespeicherten Daten im BLE-Buffer automatisch an alle abonnierten Endgeräte versendet.

4.1.7 Weiterführende Informationen

Der vollständige Sourcecode der Studienarbeit ist für weitere Details öffentlich unter nachfolgenden Link auffindbar: <https://github.com/SimLinkModule/ModuleSoftware>

4.2 Platinenentwurf

Wie in Kapitel 2.2 beschrieben, soll eine Platine entwickelt werden, die alle benötigten Elektronikkomponenten des Erweiterungsmoduls enthält. Dadurch wird es möglich das Erweiterungsmodul kompakt, mobil und benutzerfreundlich zu verwenden. Dafür sind alle Elektronikkomponenten auf drei Platinen, wie in Abbildung 14 zu sehen, untergebracht, womit für die Gehäuseerstellung eine größtmögliche Flexibilität realisiert wird. *Platine 1* dient als Verbindungsglied zwischen der Multikoperfernsteuerung und dem ESP32-Mikrocontroller. Hierzu enthält diese Platine zum einen eine Buchsenleiste mit der dargestellten Pinbelegung wie in Abbildung 15 zu sehen ist. Zusätzlich ist eine ESD-Schutzschaltung vorhanden, welche im Kapitel 4.2.1 genauer beschrieben wird. Auf der *Platine 2* befindet sich die Hauptkomponenten des Erweiterungsmoduls. Dazu zählen die Spannungsregulierung, der ESP32-Mikrocontroller und die Logik für das Beschreiben des ESP32 mit Programmdaten. Alle Eingabe- und Ausgabeelemente für die Benutzerinteraktionen befinden sich auf der *Platine 3*.

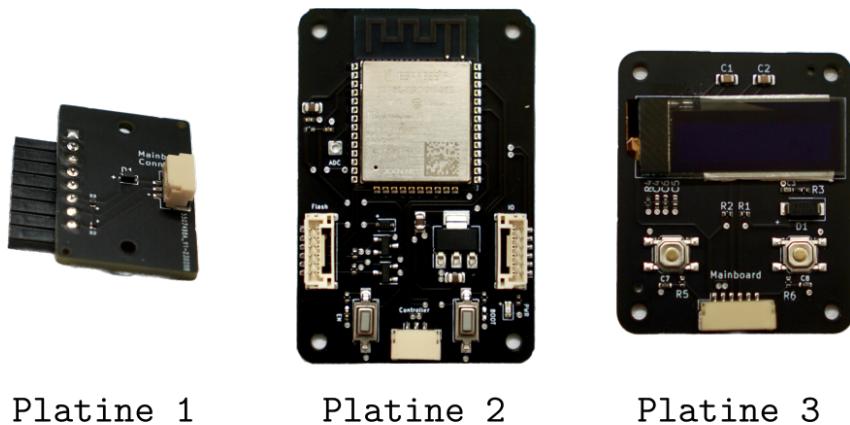


Abbildung 14: Platinen des Erweiterungsmoduls

4.2.1 Teilschaltungen

In nachfolgenden Unterkapiteln werden die wichtigsten Teilschaltungen aller Platinen des Erweiterungsmoduls genauer erklärt.

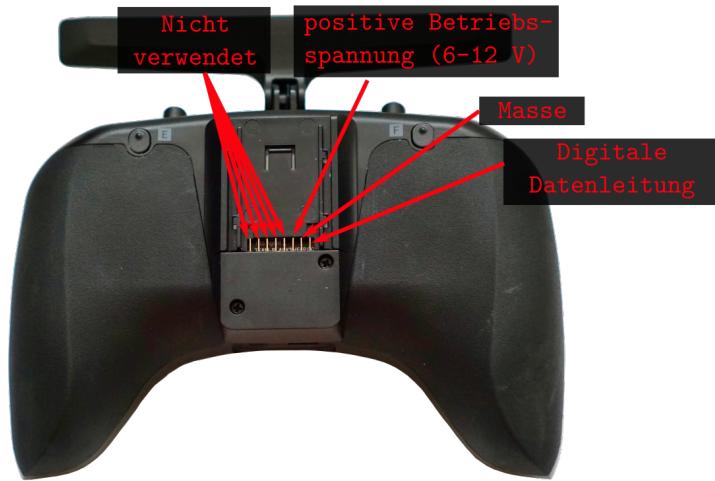


Abbildung 15: Pinbelegung des Fernsteuerungsmodulschachts; abgewandelt von [90]

Spannungsregulierung

Für den Betrieb des Erweiterungsmoduls mit dem ESP32-Mikrocontroller wird eine Spannung von 3,3 V benötigt. Jedoch liegt die verfügbare Spannung des Fernsteuerungsmodulschachts auf einen konstanten Wert zwischen 6 V und 12 V, abhängig vom verwendeten Fernsteuerungsmodell. Ebenso besteht die Möglichkeit die Platine mittels der integrierten Programmierbuchse zu betreiben, wobei hier die Spannung entweder 3,3 V oder 5 V betragen kann. Zur Regulierung der möglichen zu hohen Spannungen wird ein Low-dropout regulator (LDO) integrated circuit (IC) verwendet, welcher eine Spannung die höher als 3,3 V beträgt auf eine konstante Spannung von 3,3 V herunterregelt. Zu sehen ist die benötigte Schaltung für den Betrieb des LDOs in Abbildung 16. Zusätzlich enthält diese Schaltung im geregelten 3,3 V Spannungskreis eine LED, womit bei Fehlbetrieb des Erweiterungsmoduls überprüft werden kann, ob eine Stromzufuhr zum ESP32-Mikrocontroller vorhanden ist.

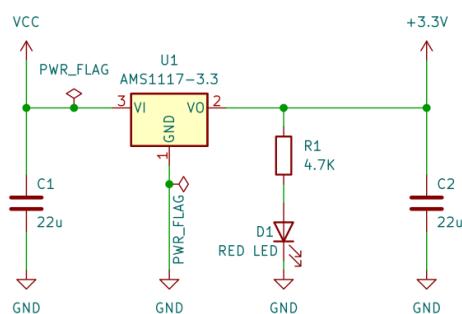


Abbildung 16: Spannungsregulierung des Erweiterungsmoduls

ESP32 Programmierlogik

Mittels einer seriellen Verbindung zwischen dem Entwicklercomputer und dem Mikrocontroller wird der ESP32-Mikrocontroller programmiert. Da jedoch heutzutage wenige Rechner eine serielle Buchse besitzen, gibt es grundlegend zwei Möglichkeiten den ESP32 zu programmieren.

Die erste Möglichkeit ist, ein USB-zu-Seriell-Programmiergerät zu verwenden, womit der Entwicklercomputer mittels USB die Programmierung durchführen kann. Die zweite Möglichkeit ist, auf der eigenentwickelten Platine ein USB-zu-Seriell-Converter IC zu integrieren, womit direkt die Programmierung des ESP32 mittels USB erfolgen kann. Jedoch entstehen bei dieser Methode zusätzliche Kosten und es wird wichtiger Platz auf der Platine verschwendet, da das Erweiterungsmodul nur einmal nach der Bestückung der Platine programmiert werden muss. Aus diesen genannten Einschränkungen wird die erste Möglichkeit zur Programmierung des ESP32 verwendet.

Damit der ESP32 für die Programmierung automatisch in den benötigten Programmiermodus schaltet, hat die verfügbare ESP32-Programmiersoftware des Herstellers einen fest definierten Programmierablauf. Der Programmierablauf verwendet, dafür zwei vorhandene Statusleitungen der seriellen Verbindung. Zum einen die *Data Terminal Ready*-Leitung und zum anderen die *Request To Send*-Leitung. Zusätzlich zu den verwendeten Statusleitungen werden noch zwei Transistoren, angeordnet wie in Abbildung 17 zu sehen, auf der Erweiterungsmodulplatine benötigt. Mittels dieser Schaltung werden die Kontakte *EN* und *Boot* des ESP32 zu Beginn der Programmierung passend beschalten. Der Kontakt *EN* am ESP32 wird verwendet, um den ESP32 lauffähig zu betreiben (Pegel von 3,3 V) oder die Ausführung von Programmen auf dem ESP32 (Pegel von 0 V) zu verhindern. Mit dem Kontakt *Boot* kann während des Starts des ESP32 festgelegt werden, ob der normale Ausführungsmodus (Pegel von 3,3 V) oder der Programmiermodus (Pegel von 0 V) gestartet werden soll. Zusätzlich muss bei der Programmierung beachtet werden, dass der maximale Spannungspegel der seriellen Verbindung bei 3,3 V liegen muss. [91]

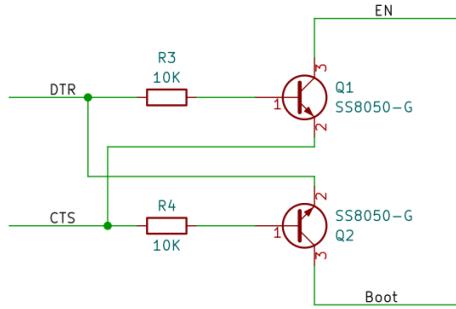


Abbildung 17: ESP32 Programmierlogik

Tastenentprellung

Zu Beginn der Betätigung eines Taster entstehen immer Schwingungen zwischen der Massespannung und der Betriebsspannung, was zur fehlerhaften Auswertung mittels eines Mikrocontrollers führt. Um dieses ungewollte Verhalten zu minimieren, beziehungsweise zu verhindern, sollten Tasten entprellt werden. Eine einfache Methode ist die Entprellung mittels Hardwarekomponenten, wie in Abbildung 18 zu sehen ist. Das Hauptelement zur Entprellung ist dabei der parallel zum Taster platzierte Kondensator [92]. Damit bei Nichtbetätigung des Tasters eine Spannung von 3,3 V am ESP32-Kontakt anliegt, enthält jeder Kontakt des ESP32 einen internen Pull-Up Widerstand (45K Ohm). Eine Betätigung des Tasters zieht folgend die Spannung des ESP32-Kontakts auf eine Spannung von 0 V.

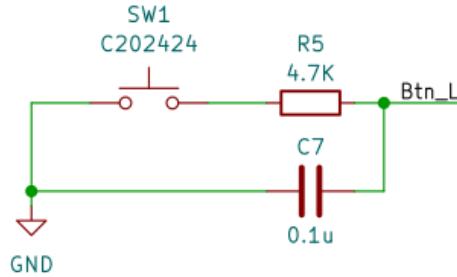


Abbildung 18: Hardwaretastenentprellung

Der Taster, welcher für den manuellen Start des Programmiermodis, auf der Platine des Erweiterungsmoduls vorhanden ist, darf jedoch nicht durch eine Hardwareschaltung entprellt werden. Da während der Aufladungsphase des Kondensators, sonst die Spannung am ESP32-Kontakt *Boot* während des Starts nahe bei 0 V liegt und der ESP32 ungewollt im Programmiermodus gestartet wird.

Electro-Static-Discharge (ESD)-Schutz

Um zu vermeiden, dass durch elektrostatische Aufladung des Körpers das Erweiterungsmodul beschädigt wird, müssen die Kontakte des Erweiterungsmoduls geschützt werden. Der Schutz wird durch Zerner-Dioden, welche in Gegenrichtung mit 0 V verbunden sind, realisiert. Zu sehen ist die Schutzschaltung der Buchsenleiste des Erweiterungsmoduls in Abbildung 19. Die Zerner-Dioden sind dabei so dimensioniert, das Spannungen größer 12 V (Versorgungsleitung) beziehungsweise größer 3,3 V (Datenleitung) über die Zener-Dioden Richtung 0 V geleitet werden.

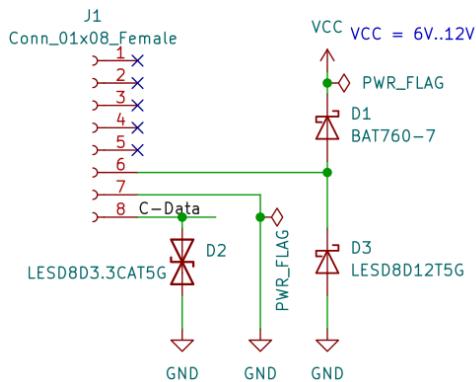


Abbildung 19: ESD-Schutzschaltung

4.2.2 Referenzschaltungen

Zur Vermeidung von Fehlern während des Schaltungsentwurfs und zur Minimierung von Platinenproduktionsdurchgängen, bauen die Platinen des Erweiterungsmoduls auf Referenzdesigns, welche in der nachfolgenden Liste zu sehen sind, auf.

- OLED-Schaltplan von ShenZhen QDtech Co. LTD; Stand: 24. Juli 2019
- ESP32_DevKitc_V4-Schaltplan von Espressif Systems (Shanghai Co., Ltd.); Stand: 7. Juni 2018
- Produktspezifikation des OEL Display Module von Allvision technology Inc.; Version: B

4.2.3 Weiterführende Informationen

Alle Schaltpläne der Erweiterungsmodulplatinen sind im Anhang in den Abbildung 26, 27 und 28 zu finden. Die vollständigen Platinenprojektdaten sind öffentlich unter folgenden Link auffindbar: <https://github.com/SimLinkModule/PCB>

Die Platinen sind dabei mit der kostenlosen Open Source Software KiCad [93] entwickelt worden und die Produktion und die Bestückung der meisten Elektronikkomponenten fand durch das Unternehmen [JLCPCB](#) statt.

4.3 Gehäuseerstellung

Damit die entworfenen Platinen fest und kompakt im Modulschacht (Typ: Lite) der Fernsteuerung befestigt werden können, ist zusätzlich ein Kunststoffgehäuse entwickelt worden, welches mittels eines Fused Deposition Modeling ([FDM](#))-3D-Druckers hergestellt werden kann. Alle benötigten Komponenten, des Gehäuses, sind im Anhang als Explosionszeichnung in Abbildung 29 zu sehen.

Der Fokus im Gehäuseentwurf liegt darin, das Gehäuse zu drucken, dass eine möglichst geringe Nachbearbeitung der Oberfläche nötig ist. Eine Nachbearbeitung, beim 3D-Druck mit dem [FDM](#)-Verfahren, wird hauptsächlich an Oberflächen benötigt, welche mittels Stützstrukturen gedruckt werden. Um Stützstrukturen im Druck zu vermeiden, sollte auf Überhänge im Modell verzichtet werden. Diese Einschränkung geht aus dem [FDM](#)-Verfahren hervor, da nur ausgehend vom Druckbrett des 3D-Druckers gedruckt werden kann. Falls jedoch nicht auf Überhänge verzichtet werden kann, gibt es zwei bekannte Möglichkeiten, um auf Stützstrukturen zu verzichten. Eine Möglichkeit ist das Modell in mehrere Teilmodelle aufzuteilen, welche einzeln ohne Stützstrukturen gedruckt werden können. Zu sehen ist dieses Verfahren in Abbildung 20. Die zweite Möglichkeit ist, Stützstrukturen in das Modell zu integrieren. Diese Stützstrukturen, sollten wie in Abbildung 21 zu sehen ist, einen Winkel zwischen 45 und 90 Grad zum Druckbrett haben.

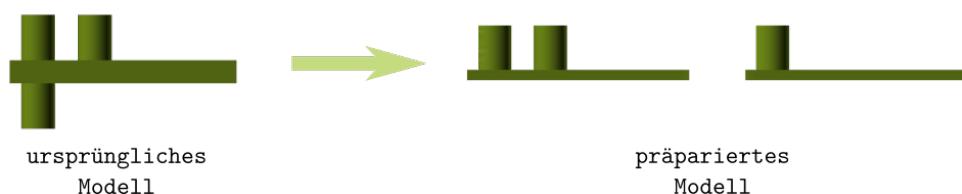


Abbildung 20: Modell mit Überhängen für den 3D-Druck vorbereiten

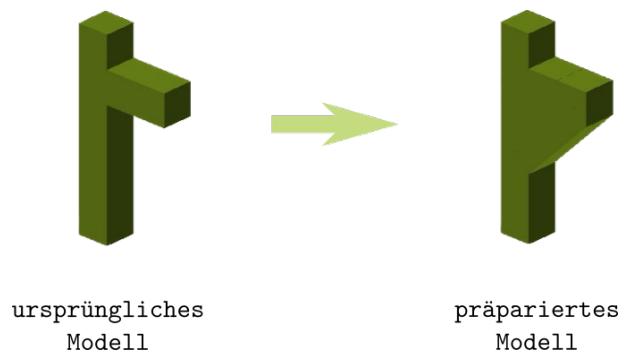


Abbildung 21: Modell mit Überhängen für den 3D-Druck vorbereiten

Zur Befestigung aller Komponenten im Kunststoffgehäuse sind mehrere Gewindegänge aus Messing integriert. Dies bietet den Vorteil, dass die einzelnen Komponenten des Erweiterungsmoduls mehrfach ein und ausgebaut werden können, da vorhandene integrierte Gewindegänge im Kunststoffgehäuse leicht überdreht und dadurch beschädigt werden können. Zu sehen sind alle benötigten Messinggewindegänge des Gehäuses in Abbildung 22.

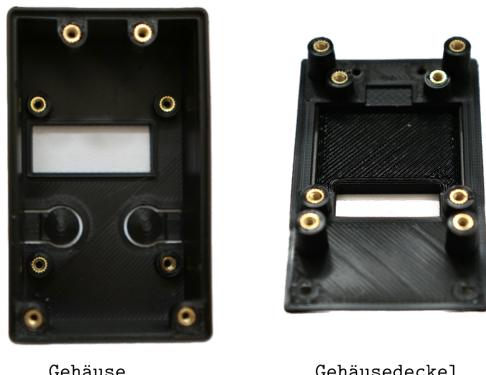


Abbildung 22: Messing Gewindegarnituren zur Befestigung der einzelnen Modulkomponenten

Da der Gehäusedeckel, auf einer Seite Befestigungen für die Platinen hat und sich auf der gegenüberliegenden Seite die Schienen des Modulschachts befinden, ist dieses 3D-Modell in zwei Teilmodelle aufgeteilt. Dies ist nötig, um sowohl ohne Stützstrukturen auszukommen als auch die filigranen Schienen mit einer guten Qualität drucken zu können. Für eine höhere Stabilität des Gehäusedeckels sollten die Teilmodelle nach dem Druck mittels Kleber verbunden werden.

Damit die Platine mit der Modulbuchse im Gehäuse befestigt werden kann, werden zusätzlich gedruckte Kunststoffhalterungen benötigt. Dies ist nötig, da die Platine sehr nah an der Gehäuseaußenkante liegen muss, um eine Verbindung mit dem Stecker der Fernsteuerung herstellen zu können. Zu sehen sind die eingebauten Platinen in Abbildung 23, sowie die Position der Modulbuchse für die Verbindung mit der Multikopterfernsteuerung.

Zum Schutz und zur Vergrößerung der Tasterdruckfläche der Taster der Eingabe- und Ausgabeplatine, ist ein integrierter Tasterschutz im Gehäuse vorhanden. Dieser Schutz und das Erweiterungsmodulgehäuse bilden dabei ein Bauteil, wodurch kein zusätzlicher Zusammenbau benötigt wird. Jedoch muss dafür das Verbindungsstück zwischen Tasterschutz und dem



Abbildung 23: Eingebaute Platinen im Gehäuse

Gehäuse flexibel entworfen werden. Dies ist möglich, indem die Schichtdicke des Verbindungsstücks verringert wird, damit der Kunststoff biegsam wird. Zu sehen ist der Tasterschutz mit Verbindungsstück in Abbildung 24. Zusätzlich ist in Abbildung 24 zu sehen, dass ein konischer Zylinder auf dem Tasterschutz vorhanden ist, womit die Tastendruckfläche vergrößert und die Lücke zwischen der befestigten Platine und dem Erweiterungsmodulgehäuse überbrückt wird.



Abbildung 24: Tasterschutz für Taster der Platine

4.3.1 Weiterführende Informationen

In Abbildung 25 ist das vollständig montierte Gehäuse mit eingebauten Platinen zu sehen.



Abbildung 25: Zusammengebautes Erweiterungsmodul

Der Entwurf des Gehäuses fand in der kostenlosen und Open Source Software OpenSCAD [94] statt. Die zugehörigen Projektdateien sind öffentlich unter folgenden Link auffindbar: <https://github.com/SimLinkModule/Shell>

5 Validierung und Gegenüberstellung

5.1 Validierung des Funktionsumfangs

In diesen Abschnitt der Studienarbeit soll der in Kapitel 2 definierte Aufgabenumfang mit dem realisierten Umfang verglichen werden und gegebenenfalls die vorhandenen Einschränkungen erläutert werden.

5.1.1 Softwareentwicklung

Die Anforderungen an die Software des Erweiterungsmoduls sind in Kapitel 2.1 aufgelistet und werden nacheinander in diesem Unterkapitel mit der finalen Umsetzung verglichen.

Die erste Anforderung ist die Unterstützung der Kommunikation des Erweiterungsmoduls mit Endgeräten auf denen das Betriebssystem Android, Windows, Linux oder iOS/iPadOS vorhanden ist. Getestet wurde das Erweiterungsmodul mit allen genannten Betriebssystemen mit jeweils einer Betriebssystemversion und einer Simulatorsoftware auf dem jeweiligen Betriebssystem. Getestet wurde das Erweiterungsmodul unter Android Version 10 und der Simulatorsoftware *FPV.SkyDive*, dabei konnte der vollständige Funktionsumfang nachgewiesen werden. Unter Windows 11 mit der Simulatorsoftware *Velocidrone* konnte ebenso der vollständige Funktionsumfang des Erweiterungsmoduls nachgewiesen werden. Als Linux-Betriebssystem wurde *Pop! OS 22.04 LTS* mit der Simulatorsoftware *Velocidrone* verwendet. Dort konnte gleichermaßen der vollständige Funktionsumfang nachgewiesen werden. Für die Überprüfung des Funktionsumfangs des Erweiterungsmoduls mit iOS/iPadOS 16.3.1 wurde die Simulatorsoftware *FPV.SkyDive* verwendet. Dort konnte keine Funktionsfähigkeit nachgewiesen werden. Jedoch findet unter iOS/iPadOS ein Verbindungsaufbau zwischen dem Erweiterungsmodul und dem iOS/iPadOS-Endgerät statt und ebenfalls werden die Daten vom Erweiterungsmodul iOS/iPadOS-Endgerät empfangen. Nachgewiesen werden kann dies mit der Software *packetLogger* und *libimobiledevice*. Für weitere Untersuchungen wurde ein, mit iOS/iPadOS kompatibler, Nintendo Switch-Controller herangezogen und der Kommunikationsaustausch zwischen dem iOS/iPadOS-Endgerät und dem Nintendo Switch-Controller betrachtet. Dabei stellt sich heraus, dass der Nintendo Switch-Controller mittels *BBR* die Gamecontrollerdaten und nicht mittels *BLE* überträgt. Wie in Quelle [95] beschrieben ist, müssen *BLE*-Gamecontroller durch das *MFi*-Programm zertifiziert werden und ein zusätzliches Hardwaremodul enthalten, entgegen der ursprünglichen Annahme, welche in Kapitel 3.2.3 beschrieben ist.

Die zweite Anforderung an die Software ist, dass die Kommunikation zwischen dem Erweiterungsmodul und den Endgeräten mittels *BLE* erfolgen und sich das Erweiterungsmodul als *HID*-Gerät authentifizieren soll. Die Umsetzung dieser Anforderung ist in Kapitel 4.1.2 und 4.1.3 beschrieben.

Die dritte Anforderung ist, dass die Kommunikation zwischen dem Erweiterungsmodul und der Multikopterfernsteuerung über den Modulschacht der Fernsteuerung erfolgen soll. Auch soll die

Kommunikation mittels eines vorhandenen Protokolls der Fernsteuerungsfirmware OpenTX oder einer Abspaltung davon erfolgen. Als Kommunikationsprotokoll zwischen dem Erweiterungsmodul und der Multikopterfernsteuerung wird CSRF verwendet wie in Kapitel 4.1.4 nachgelesen werden kann. Getestet wurde die Erweiterungsmodulsoftware mit der Multikopterfernsteuerung Tango 2 von dem Unternehmen *Team Blacksheep* mit der Firmware FreedomTX Version TBS-1.3.0. Dabei kann festgestellt werden, dass alle versendeten Kanaldaten mittels dem CSRF-Protokoll empfangen und verarbeitet werden können.

Die vierte Anforderung ist, dass es verschiedene Ein- und Ausgabemöglichkeiten geben soll für die Interaktion mit dem Erweiterungsmodul durch den Endanwender des Erweiterungsmoduls. Dafür wird einerseits für die Ausgabe von Statusnachrichten ein **OLED**-Display verwendet, wie in Kapitel 4.1.5 beschrieben ist. Andererseits gibt es Taster für Eingaben durch den Endanwender, nachzulesen in Kapitel 4.1.6. Ebenfalls war ein Teil dieser Anforderung, dass weitere Statusindikatoren als light-emitting diodes (**LEDs**) integriert werden sollen. Dies ist jedoch nicht umgesetzt worden, da alle wichtigen Statusnachrichten für den Endanwender durch das **OLED**-Display angezeigt werden können. Lediglich eine Status-**LED** ist vorhanden, um anzulegen, ob eine funktionierende Stromzufuhr zum Erweiterungsmodul vorhanden ist. Dies ist jedoch komplett in Hardware realisiert wie in Abbildung 16 zu sehen ist.

Die letzte Anforderung an die Software ist, dass der Akkustand der Multikopterfernsteuerung an das zugehörige Endgerät übermittelt wird. Die Softwarekomponente hierfür ist vollständig vorhanden und funktionsfähig, jedoch bietet der Lite-Modulschachtstecker keinen Pin, an dem die rohe Akkuspannung anliegt (zu sehen in Abbildung 15). Aus diesem Grund wird im **BLE**-Akkustand-Merkmal immer der Wert 0 übertragen. Dadurch wird sichergestellt, dass der Endanwender des Erweiterungsmoduls selbst den aktuellen Akkustand der Multikopterfernsteuerung überprüft, da er davon ausgehen kann, dass es sich um ein Fehlverhalten des Erweiterungsmoduls handelt.

5.1.2 Platinenentwurf

Die Anforderungen an die entworfenen Platinen des Erweiterungsmoduls sind in Kapitel 2.2 aufgelistet. Dabei ist die Hauptanforderung, dass die Platine alle benötigten Elektronikkomponenten für das Erweiterungsmodul, welche im prototypischen Steckbrettaufbau vorhanden sind, enthalten muss. Zusätzlich sollte die Platine möglichst kompakt sein, damit die Platine an der Multikopterfernsteuerung verwendet werden kann, ohne dass diese während der Verwendung der Fernsteuerung stört. Diese Anforderungen wurden in den Platinen, welche in Kapitel 4.2 vorgestellt wurden, umgesetzt. Die Platine enthält dafür den ESP32-Mikrocontroller, die benötigte Spannungsregulierung für alle Elektronikkomponenten sowie Taster, ein Display und eine Status-**LED** für die Interaktion mit dem Endanwender. Zusätzlich ist ein **ESD**-Schutz an der Erweiterungsmodulbuchse integriert, ebenso wie weitere Komponenten für die komfortablere Programmierung des ESP32-Mikrocontrollers.

5.1.3 Gehäuseerstellung

Die Anforderungen an das Erweiterungsmodulgehäuse sind in Kapitel 2.3 aufgelistet und umfassen den Entwurf eines Kunststoffgehäuses, welches für Modulschächte des Typs *Lite*

ist und möglichst ohne Nachbearbeitung verwendet werden kann. All diese Anforderungen sind im entworfenen Gehäuse von Kapitel 4.3 realisiert worden. Das entworfene Gehäuse ist für Erweiterungsmodulschächte des Typs *Lite* und kann mit vier Stützstrukturen gedruckt werden. Da diese an nicht sichtbaren Bereichen des Gehäuses benötigt werden, entfällt die Nachbearbeitung der Oberflächen.

5.2 Gegenüberstellung BLE-Modul und USB-Verbindung

Ein wichtiges Merkmal von Fernsteuerung während der Verwendung im Simulator als auch mit einem reellen Multikopter stellt die Latenz zwischen der Eingabe eines Signals bis zur Verarbeitung an der Gegenstelle dar. Um die Latenz zu bestimmen wird der nachfolgend beschriebene Versuchsaufbau verwendet.

5.2.1 Versuchsaufbau

5.2.2 Auswertung

6 Rekapitulation und Ausblick

TODO: Rekapitulation und Ausblick

Literatur

- [1] *Drones by the Numbers*, https://www.faa.gov/uas/resources/by_the_numbers/, Aufgerufen am: 16. November 2022, Federal Aviation Administration.
- [2] *Commercial Drones are Taking Off*, <https://www.statista.com/chart/17201/commercial-drones-projected-growth/>, Aufgerufen am: 16. November 2022, Katharina Buchholz.
- [3] *Drones: A Tech Growth Market in the United States*, <https://www.statista.com/chart/9525/sales-of-consumer-drones-to-dealers-in-the-us/>, Aufgerufen am: 16. November 2022, Dyfed Loesche.
- [4] *The Economic Impact Of The Commercial Drone Sector*, <https://www.statista.com/chart/3898/the-economic-impact-of-the-commercial-drone-sector/>, Aufgerufen am: 16. November 2022, Niall McCarthy.
- [5] *DJI Mavic 3 Classic*, <https://www.dji.com/de/mavic-3-classic>, Aufgerufen am: 16. November 2022, DJI.
- [6] *Learn the Different FPV Drone Flight Modes & How to Set Up*, <https://academy.wedio.com/fpv-drone-flight-modes/>, Aufgerufen am: 16. November 2022, Wedio.
- [7] *Apple introduces next-generation iPad Pro, supercharged by the M2 chip*, <https://www.apple.com/newsroom/2022/10/apple-introduces-next-generation-ipad-pro-supercharged-by-the-m2-chip/>, Aufgerufen am: 16. November 2022, Apple Inc.
- [8] *Joystick emulation*, https://doc.open-tx.org/manual-for-opentx-2-2/advanced-features/radio_joystick, Aufgerufen am: 16. November 2022, OpenTX.
- [9] *Taranis IO ports*, <https://github.com/opentx/opentx/wiki/Taranis-IO-ports#external-module-bay-pinout>, Aufgerufen am: 16. November 2022, OpenTX.
- [10] *HID Joystick Support*, <https://github.com/betaflight/betaflight/wiki/HID-Joystick-Support>, Aufgerufen am: 16. November 2022, Betaflight.
- [11] *Orqa FPV.JR Bluetooth*, <https://github.com/betaflight/betaflight/wiki/HID-Joystick-Support>, Aufgerufen am: 16. November 2022, Orqa.
- [12] *Welcome to OpenTX*, <https://www.open-tx.org/>, Aufgerufen am: 16. November 2022, OpenTX.
- [13] *Device Class Definition for Human Interface Devices (HID), Firmware Specification*, Version 1.11, USB Implementers' Forum, Mai 2001.
- [14] *HID Usage Tables for Universal Serial Bus (USB)*, Version 1.3, USB Implementers' Forum, 2002.
- [15] F. Zhao, *Tutorial about USB HID Report Descriptors*, <https://eleccelerator.com/tutorial-about-usb-hid-report-descriptors/>, Aufgerufen am: 15. Oktober 2022.

- [16] *Bluetooth Core Specification*, Revision v5.3, Bluetooth SIG, 2021.
- [17] *UG103.14: Bluetooth LE Fundamentals*, Revision 0.7, SILICON LABS.
- [18] *MFi Program, Frequently Asked Questions*, <https://mfi.apple.com/en/faqs.html>, Aufgerufen am: 05. Oktober 2022, Apple Inc.
- [19] *Accessory Design Guidelines for Apple Devices*, Release R18, Apple Inc., 2022.
- [20] *HID OVER GATT PROFILE SPECIFICATION*, Revision v10r00, Bluetooth SIG, Dez. 2011.
- [21] *HID SERVICE SPECIFICATION*, Revision v10r00, Bluetooth SIG, Dez. 2011.
- [22] *BATTERY SERVICE SPECIFICATION*, Revision v10r00, Bluetooth SIG, Dez. 2011.
- [23] *DEVICE INFORMATION SERVICE*, Revision v11r00, Bluetooth SIG, Dez. 2011.
- [24] *SCAN PARAMETERS PROFILE SPECIFICATION*, Revision v10r00, Bluetooth SIG, Dez. 2011.
- [25] *Bluetooth API*, <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/bluetooth/index.html>, Aufgerufen am: 16. November 2022, Espressif Systems (Shanghai) Co., Ltd.
- [26] *esp-idf/components/bt/host/bluedroid/main/bte_main.c*, https://github.com/espressif/esp-idf/blob/ac99c0ad6ba783e99d21fe142fc610001dc93457/components/bt/host/bluedroid/main/bte_main.c, Aufgerufen am: 16. November 2022, Espressif.
- [27] J. Edge, *Returning BlueZ to Android*, <https://lwn.net/Articles/597293/>, Aufgerufen am: 16. November 2022.
- [28] *Fluoride Bluetooth stack*, <https://cs.android.com/android/platform/superproject/+/master:packages/modules/Bluetooth/>, Aufgerufen am: 16. November 2022, Google.
- [29] Anchao, *Fluoride Bluetooth stack*, <https://github.com/anchao/fluoride>, Aufgerufen am: 16. November 2022.
- [30] *BLE User Guide*, <https://mynewt.apache.org/latest/network/>, Aufgerufen am: 16. November 2022, Apache.
- [31] *Apache mynewt*, <https://github.com/apache/mynewt-nimble/blob/b7c1dd7a62dab4de17984f382e8ee101c5361c7d/README.md>, Aufgerufen am: 16. November 2022, Apache.
- [32] O. Liang, *RC Protocols Explained: SBUS, CRSF, PWM, FPort and More*, <https://oscarliang.com/rc-protocols/>, Aufgerufen am: 22. Oktober 2022.
- [33] *R9M*, <https://www.frsky-rc.com/product/r9m/>, Aufgerufen am: 22. Oktober 2022, FrSky.
- [34] *Model Setup*, https://doc.open-tx.org/manual-for-opentx-2-2/software-overview/model_menus/model_setup, Aufgerufen am: 22. Oktober 2022, OpenTX.
- [35] *FAQ*, <https://doc.open-tx.org/manual-for-opentx-2-2/faq>, Aufgerufen am: 22. Oktober 2022, OpenTX.
- [36] U. Horn und H. Schneider, *PPM*, <https://wiki.rc-network.de/wiki/PPM>, Aufgerufen am: 22. Oktober 2022.

- [37] *opentx/radio/src/pulses/ppm.cpp*, <https://github.com/opentx/opentx/blob/d67d1aa0c09d2f485c3ef7ca8c4fb1e9214a2ee7/radio/src/pulses/ppm.cpp>, Aufgerufen am: 22. Oktober 2022, OpenTX.
- [38] *cleanflight/src/main/rx/crsf.c*, <https://github.com/cleanflight/cleanflight/blob/acc56ce09dc4cf67dd6712d7c228352659133ce3/src/main/rx/crsf.c#L74>, Aufgerufen am: 22. Oktober 2022, Cleanflight.
- [39] *cleanflight/src/main/rx/crsf_protocol.h*, https://github.com/cleanflight/cleanflight/blob/acc56ce09dc4cf67dd6712d7c228352659133ce3/src/main/rx/crsf_protocol.h, Aufgerufen am: 22. Oktober 2022, Cleanflight.
- [40] *Protocol decoder:sbus_futaba*, https://sigrok.org/wiki/Protocol_decoder:Sbus_futaba, Aufgerufen am: 22. Oktober 2022, sigrok.
- [41] *cleanflight/src/main/common/crc.c*, <https://github.com/cleanflight/cleanflight/blob/acc56ce09dc4cf67dd6712d7c228352659133ce3/src/main/common/crc.c#L67>, Aufgerufen am: 22. Oktober 2022, Cleanflight.
- [42] *sbus/README.md*, <https://github.com/bolderflight/sbus/blob/61a9d25eb964b9a75cca51ce047715570b14cac8/README.md>, Aufgerufen am: 22. Oktober 2022, Bolder Flight.
- [43] P. Langer, *DIY-Multiprotocol-TX-Module/docs/Transmitters.md*, <https://github.com/pascallanger/DIY-Multiprotocol-TX-Module/blob/75c9fb40a7eeafbd7716ca12373936706017be05/docs/Transmitters.md>, Aufgerufen am: 22. Oktober 2022.
- [44] P. Langer, *DIY-Multiprotocol-TX-Module/Multiprotocol/Multiprotocol.h*, <https://github.com/pascallanger/DIY-Multiprotocol-TX-Module/blob/75c9fb40a7eeafbd7716ca12373936706017be05/Multiprotocol/Multiprotocol.h#L834>, Aufgerufen am: 22. Oktober 2022.
- [45] *Modules*, <https://www.espressif.com/en/products/modules>, Aufgerufen am: 14. Januar 2023, Espressif Systems.
- [46] *Development Boards*, <https://www.espressif.com/en/products/devkits>, Aufgerufen am: 14. Januar 2023, Espressif Systems.
- [47] *SoCs*, <https://www.espressif.com/en/products/socs>, Aufgerufen am: 14. Januar 2023, Espressif Systems.
- [48] *Get Started*, <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/index.html>, Aufgerufen am: 14. Januar 2023, Espressif Systems (Shanghai) Co., Ltd.
- [49] *ESP32, Technical Reference Manual*, Version 4.8, Espressif Systems, 2022.
- [50] *What Needs FCC Approval?* <https://compliancetesting.com/what-needs-fcc-approval/>, Aufgerufen am: 14. Januar 2023, Compliance Testing.
- [51] *Certificates*, <https://www.espressif.com/en/support/documents/certificates>, Aufgerufen am: 14. Januar 2023, Espressif Systems.
- [52] *Espressif IoT Development Framework*, <https://github.com/espressif/esp-idf>, Aufgerufen am: 14. Januar 2023, Espressif Systems.

- [53] *CE marking*, https://europa.eu/youreurope/business/product-requirements/labels-markings/ce-marking/index_en.htm, Aufgerufen am: 14. Januar 2023, European Union.
- [54] *Radio Equipment Directive (RED)*, https://single-market-economy.ec.europa.eu/sectors/electrical-and-electronic-engineering-industries-eei/radio-equipment-directive-red_en, Aufgerufen am: 14. Januar 2023, European Comission.
- [55] *RICHTLINIE 2014/53/EU DES EUROPÄISCHEN PARLAMENTS UND DES RATES vom 16. April 2014 über die Harmonisierung der Rechtsvorschriften der Mitgliedstaaten über die Bereitstellung von Funkanlagen auf dem Markt und zur Aufhebung der Richtlinie 1999/5/EG*, Europäischen Union.
- [56] H. Naumann, *GSM / NB-IoT antenna and radio certification*, <https://www.gsm-modem.de/M2M/m2m-components/gsm-nb-iot-antenna-radio-certification-ce-red-fcc/>, Aufgerufen am: 14. Januar 2023.
- [57] R. Barry, *Mastering the FreeRTOS Real Time Kernel, a Hands-On Tutorial Guide*, 2016.
- [58] *What is An RTOS?* <https://www.freertos.org/about-RTOS.html>, Aufgerufen am: 16. Januar 2023, Real Time Engineers Ltd.
- [59] *License Details*, <https://www.freertos.org/a00114.html>, Aufgerufen am: 16. Januar 2023, Real Time Engineers Ltd.
- [60] *The FreeRTOS Kernel*, <https://www.freertos.org/RTOS.html>, Aufgerufen am: 16. Januar 2023, Real Time Engineers Ltd.
- [61] *Source Organization*, <https://www.freertos.org/a00017.html>, Aufgerufen am: 16. Januar 2023, Real Time Engineers Ltd.
- [62] *FAQ - Scheduling*, <https://www.freertos.org/FAQSched.html>, Aufgerufen am: 16. Januar 2023, Real Time Engineers Ltd.
- [63] *FreeRTOS scheduling*, <https://www.freertos.org/single-core-and-smp-rtos-scheduling.html>, Aufgerufen am: 16. Januar 2023, Real Time Engineers Ltd.
- [64] *The RTOS Tick*, <https://www.freertos.org/implementation/a00011.html>, Aufgerufen am: 16. Januar 2023, Real Time Engineers Ltd.
- [65] *Software Timers*, <https://www.freertos.org/RTOS-software-timer.html>, Aufgerufen am: 16. Januar 2023, Real Time Engineers Ltd.
- [66] *FreeRTOS Queues*, <https://www.freertos.org/Embedded-RTOS-Queues.html>, Aufgerufen am: 16. Januar 2023, Real Time Engineers Ltd.
- [67] *ESP-IDF FreeRTOS (SMP)*, <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/freertos-smp.html>, Aufgerufen am: 16. Januar 2023, Espressif Systems (Shanghai Co., Ltd.)
- [68] *FreeRTOS*, <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/freertos.html>, Aufgerufen am: 16. Januar 2023, Espressif Systems (Shanghai Co., Ltd.)
- [69] *Inter-Processor Call*, <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/ ipc.html>, Aufgerufen am: 16. Januar 2023, Espressif Systems (Shanghai Co., Ltd.)

- [70] *Raster, Vector, TrueType, and OpenType Fonts*, <https://learn.microsoft.com/en-us/windows/win32/gdi/raster--vector--truetype--and--opentype-fonts>, Aufgerufen am: 15. Januar 2023, Microsoft.
- [71] J. Sanson, *Custom Fonts for Microcontrollers*, <https://jared.geek.nz/2014/jan/custom-fonts-for-microcontrollers>, Aufgerufen am: 15. Januar 2023.
- [72] I. Strizver, *A Brief History of Digital Type*, <https://www.fonts.com/content/learning/fyti/using-type-tools/digital-format>, Aufgerufen am: 15. Januar 2023.
- [73] A. Hershey, *Calligraphy for Computers*, U.S. Naval Weapons Laboratory, Dahlgren, Virginia, Aug. 1967.
- [74] *TrueType Fundamentals*, <https://learn.microsoft.com/en-us/typography/opentype/spec/ttch01>, Aufgerufen am: 15. Januar 2023, Microsoft.
- [75] V. Pavlik, *Linux Input drivers v1.0*, <https://www.kernel.org/doc/Documentation/input/input.txt>, Aufgerufen am: 15. Januar 2023.
- [76] R. H. Espinosa, *The Linux Input Documentation »Linux Input Subsystem userspace API »6. Linux Joystick support »6.6. Programming Interface*, <https://www.kernel.org/doc/html/v4.17/input/joydev/joystick-api.html?highlight=joystick%20api>, Aufgerufen am: 15. Januar 2023.
- [77] V. Pavlik, *Docs »The Linux Input Documentation »Linux Input Subsystem userspace API »1. Introduction*, <https://www.kernel.org/doc/html/v4.17/input/input.html>, Aufgerufen am: 15. Januar 2023.
- [78] *linux/include/uapi/linux/input-event-codes.h*, <https://github.com/torvalds/linux/blob/master/include/uapi/linux/input-event-codes.h>, Aufgerufen am: 15. Januar 2023, Linux.
- [79] *7.1 Talking to Device Files (writes and IOCTLs)*, <https://tldp.org/LDP/lkmpg/2.4/html/x856.html>, Aufgerufen am: 15. Januar 2023, The Linux Documentation Project.
- [80] *libevdev Documentation*, <https://www.freedesktop.org/software/libevdev/doc/latest/index.html>, Aufgerufen am: 15. Januar 2023, freedesktop.org.
- [81] *libevdev*, <https://www.freedesktop.org/wiki/Software/libevdev/>, Aufgerufen am: 15. Januar 2023, freedesktop.org.
- [82] *ioctl based interface*, <https://docs.kernel.org/driver-api/ioctl.html>, Aufgerufen am: 15. Januar 2023, The Linux Kernel.
- [83] *ESP32 - WiFi-enabled microcontrollers from Espressif*, <https://www.reddit.com/r/esp32/>, Aufgerufen am: 15. Februar 2023, Reddit Inc.
- [84] *ESPRESSIF ESP32*, <https://www.esp32.com/>, Aufgerufen am: 15. Februar 2023, Espressif Systems (Shanghai Co., Ltd.)
- [85] *ESP32, Hardware Design Guidelines*, Version 3.3, Espressif Systems, 2022.
- [86] *Introduction to Human Interface Devices (HID)*, <https://learn.microsoft.com/en-us/windows-hardware/drivers/hid/>, Aufgerufen am: 24. Februar 2023, Microsoft.
- [87] *BLE Pairing and Bonding*, https://technotes.kynetics.com/2018/BLE_Pairing_and_bonding/, Aufgerufen am: 11. März 2023, Kynetics.

- [88] *Universal Asynchronous Receiver/Transmitter (UART)*, <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/uart.html>, Aufgerufen am: 24. Februar 2023, Espressif Systems (Shanghai Co., Ltd.)
- [89] *SSD1306, Advance Information, 128 x 64 Dot Matrix, OLED/PLED Segment/Common Driver with Controller*, Rev 1.5, Solomon Systech Limited, 2010.
- [90] P. Seidel, *FrSky Taranis X-Lite Fernsteuerung*, <https://blog.seidel-philipp.de/frsky-taranis-x-lite-fernsteuerung/>, Aufgerufen am: 20. März 2023.
- [91] *Boot Mode Selection*, <https://docs.espressif.com/projects/esptool/en/latest/esp32/advanced-topics/boot-mode-selection.html>, Aufgerufen am: 20. März 2023, Espressif Systems (Shanghai Co., Ltd.)
- [92] *EMBED WITH ELLIOT: DEBOUNCE YOUR NOISY BUTTONS, PART I*, <https://hackaday.com/2015/12/09/embed-with-elliot-debounce-your-noisy-buttons-part-i/>, Aufgerufen am: 13. März 2023, Elliot Williams.
- [93] *About KiCad*, <https://www.kicad.org/about/kicad/>, Aufgerufen am: 11. März 2023, KiCad.
- [94] *About OpenSCAD*, <https://openscad.org/about.html>, Aufgerufen am: 11. März 2023, OpenSCAD.
- [95] societyofrobots, *clarification for iOS compatibility note #50*, <https://github.com/mmingDev/ESP32-BLE-Gamepad/issues/50#issuecomment-862770124>, Aufgerufen am: 10. April 2023.

Anhang

- A. Schaltpläne
- B. Explosionszeichnung des Gehäuses

A. Schaltpläne

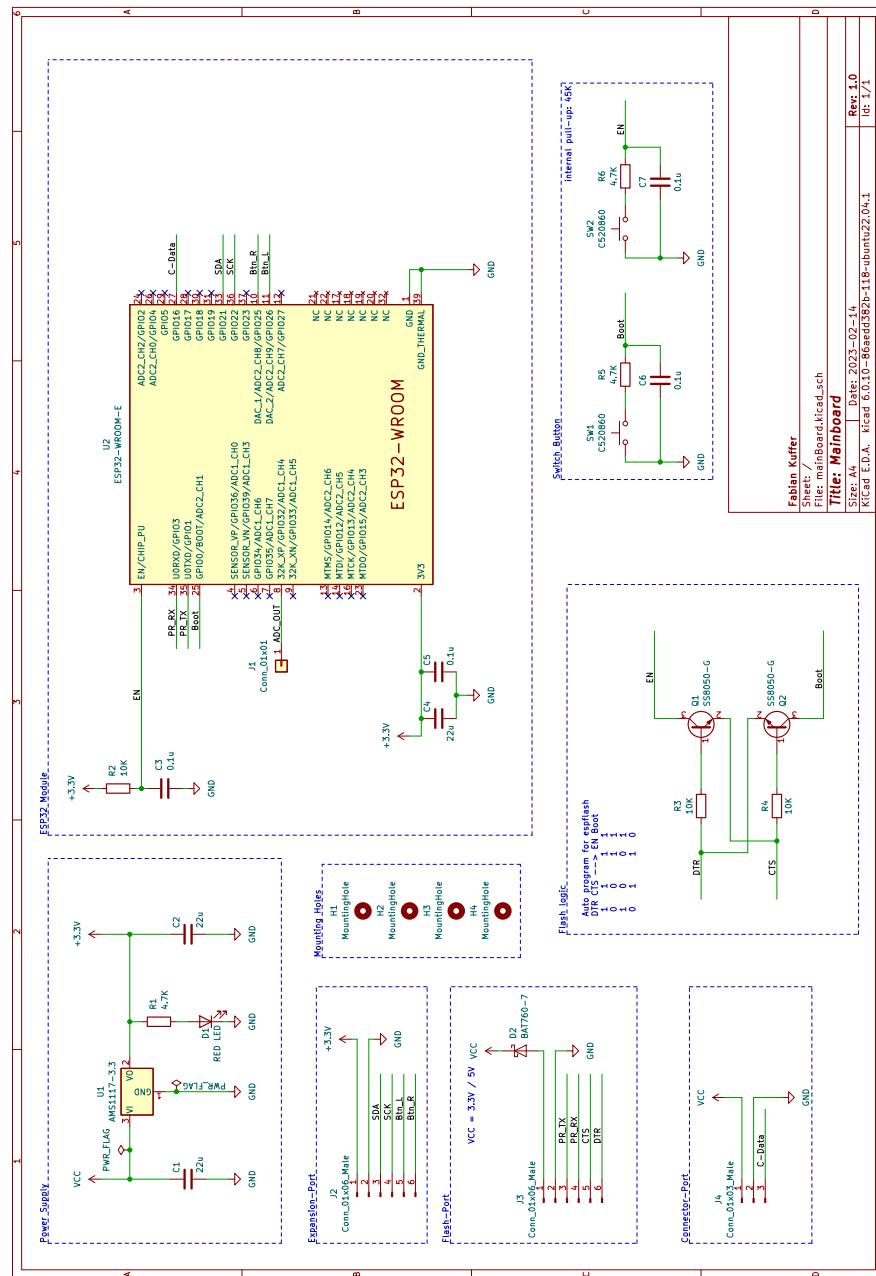


Abbildung 26: Schaltplan der Hauptplatine

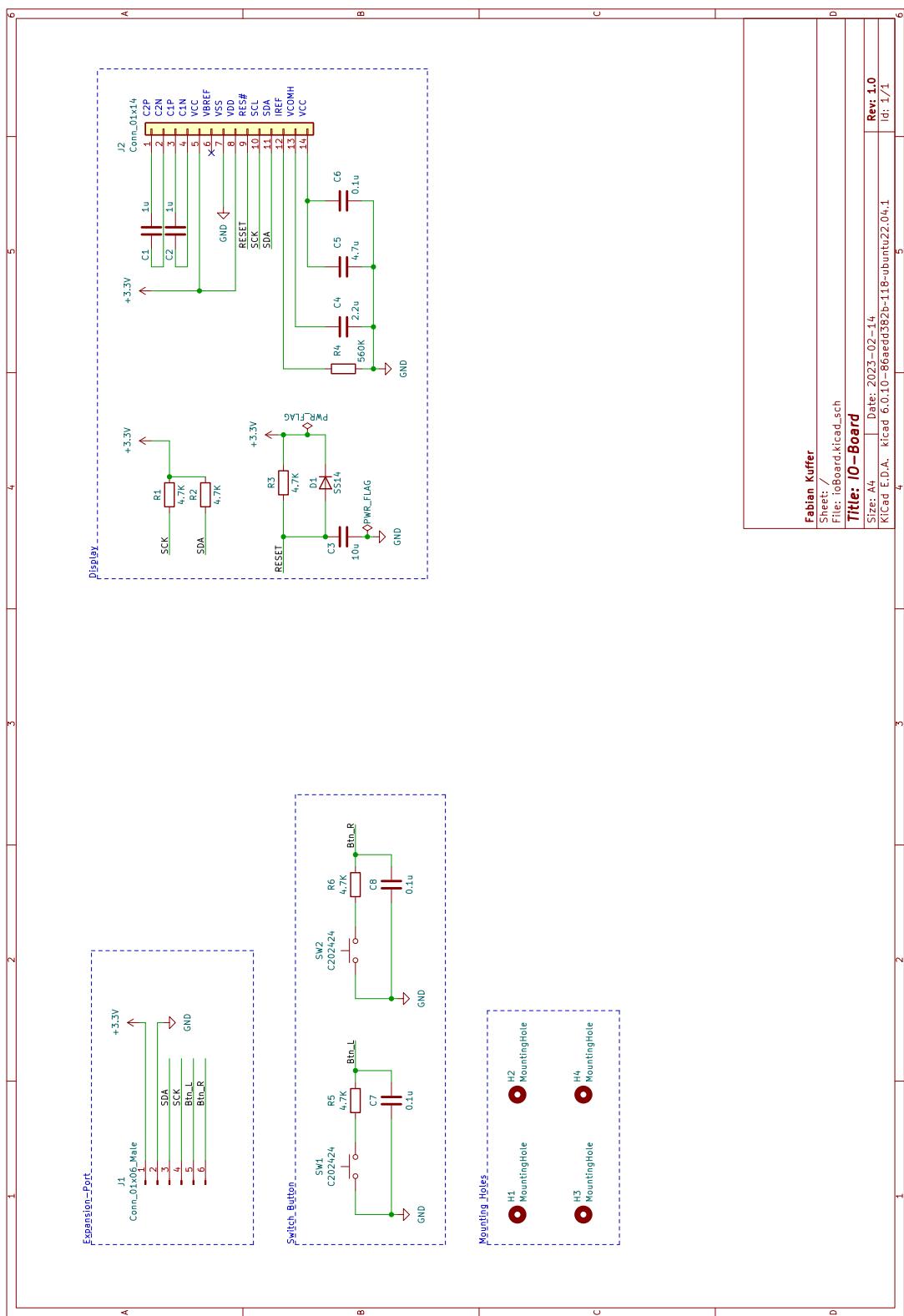


Abbildung 27: Schaltplan für Ein- und Ausgabekomponenten

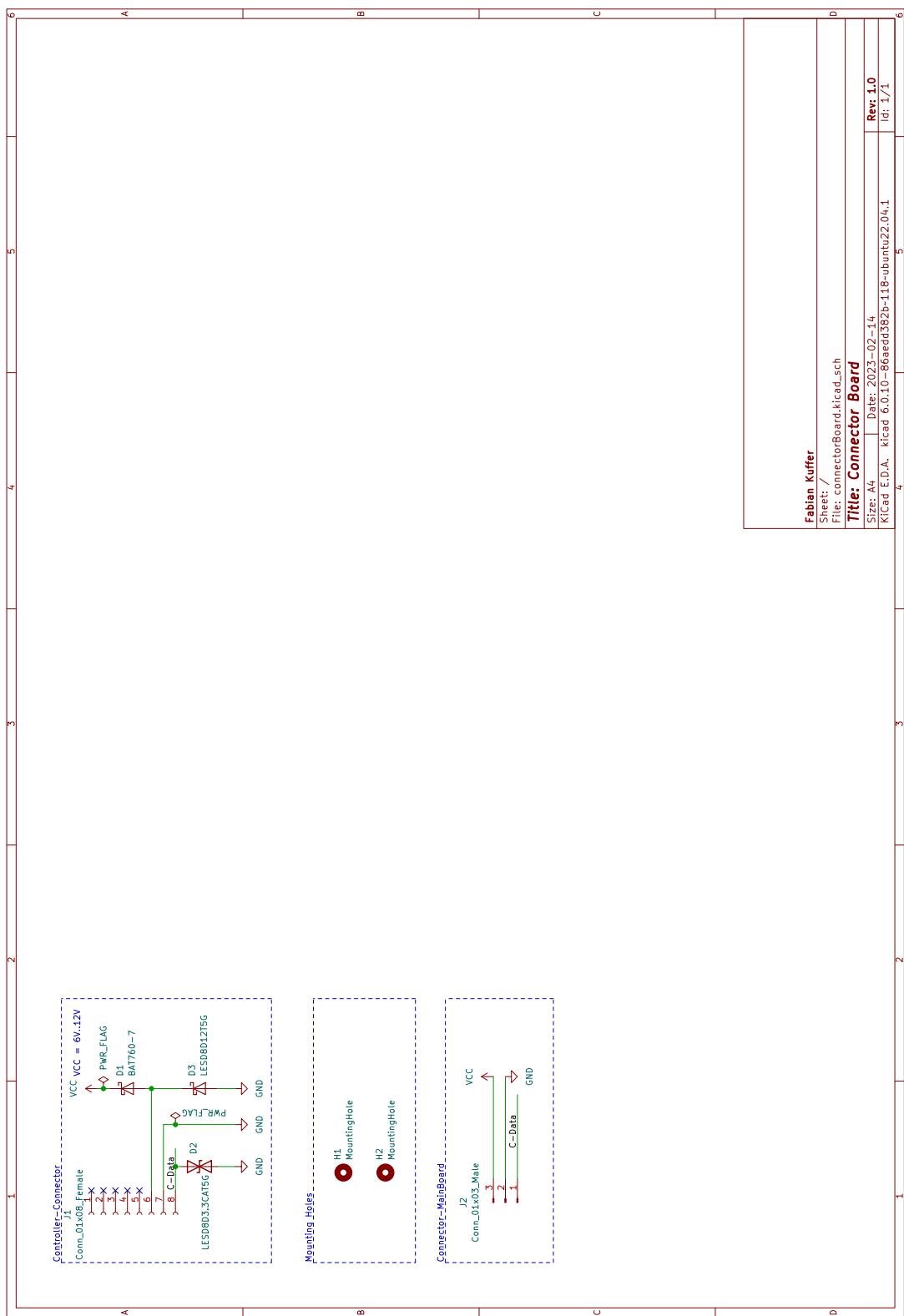


Abbildung 28: Schaltplan für die Verbindung zwischen Erweiterungsmodul und Multikopterfernsteuerung

B. Explosionszeichnung des Gehäuses

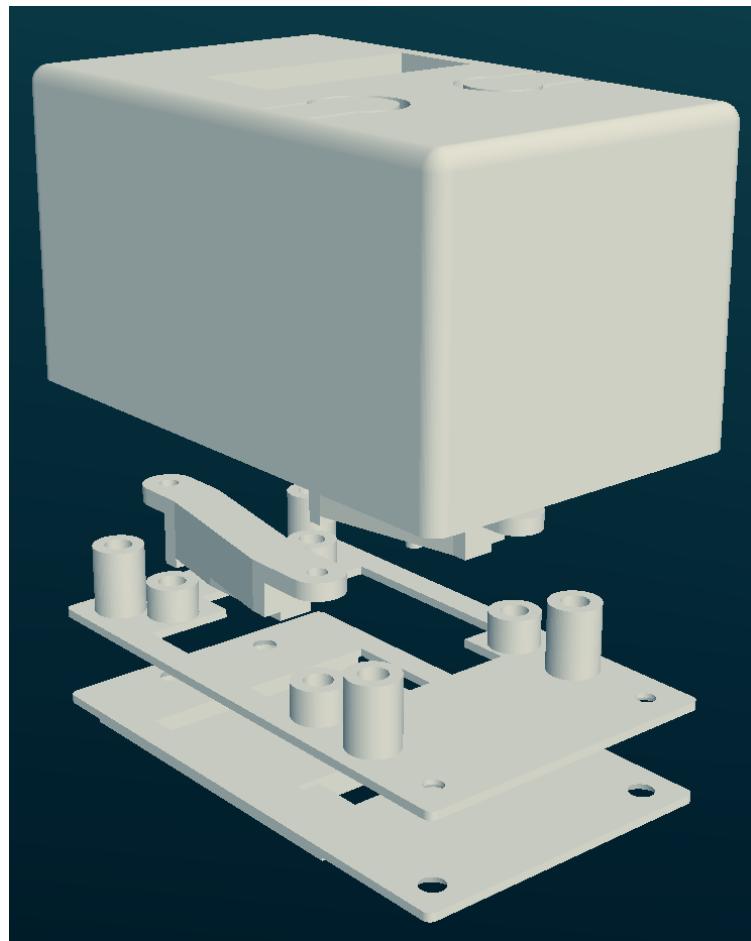


Abbildung 29: Explosionszeichnung mit allen Modellen des Gehäuses