BLE HID Hardware-Erweiterungsmodul für Drohnenfernbedienungen

Studienarbeit

des Studiengangs IT-Automotive

an der Dualen Hochschule Baden-Württemberg Stuttgart

von

Fabian Kuffer

11. Dezember 2022

Bearbeitungszeitraum	4. Oktober 2022 - 8. Juni 2023
Matrikelnummer, Kurs	2044882, TINF-20ITA
Betreuer	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, 11. Dezember 2022

Fabian Kuffer

Kurzfassung

TODO: Kurzfassung

Abstract

TODO: Abstract

Inhaltsverzeichnis

Abkürzungsverzeichnis

ADC	Analog-Digital-Wandler
ATT	Attribute Protocol
BBR	Bluetooth Basic Rate
BLE	Bluetooth Low Energy
CID	Kanalidentifizierer
ESP-IDF	Espressif IoT Development Framework
GAP	Generic Access Profile
GATT	Generic Attribute Profile
HCI	Host Controller Interface
HID	Human Interface Device
HOGP	HID over GATT Profile
ISM	Industrial, Scientific and Medical
L2CAP	Logical Link Control and Adaption Protocol
LL	Link Layer
MFi	Made for iPhone/iPad/IPad
PHY	Physical Layer
PPM	Puls-Positions-Modulation
SDP	Service Discovery Protocol
SIG	Special Interest Group
SMP	Security Manager Protocol
UART	Universal Asynchronous Receiver Transmitter
UUID	universal unique identifier

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

1 Einleitung

Multikopter beziehungsweise Quadrokopter haben in den letzten Jahren sowohl im privaten als auch im kommerziellen Sektor ein konstantes Wachstum erzielt. So sind beispielsweise in den Vereinigten Staaten von Amerika zum Stand vom 31. Mai 2022 über 865.000 Multikopter registriert. Davon sind über 500.000 Multikopter 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 Unterstützungsfunktionen an die teils ungeübten Piloten bereitzustellen. Ein Beispiel für einen Quadrokopter dieser Kategorie wäre die Mavic 3 Classic von dem Unternehmen DJI. Diese hat beispielsweise Sichtsensoren welche nach unten, oben, vorne und hinten ausgerichtet sind, um Objekte im Flugfeld zu erkennen und auszuweichen. 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. Dafür wird jedoch auf die Unterstützungsfunktionen von Consumerquadrokoptern verzichtet. Eine weitere Besonderheit ist die Möglichkeit zwischen drei Flugmodi auszuwählen. Zum einen den Flugmodus *Angle Mode*. In diesen Modus wird der Quadrokopter ab einen fest vordefinierten Neigungswinkel automatisch begrenzt, wodurch Loopings und Rollen des Quadrokopters 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 diesen Flugmodus Loopings und Rollen gemacht werden. Der letzte verfügbar Flugmodus ist der *Air beziehungsweise Acro Mode*. In diesen Modus muss der Pilot sich um das Ausrichten der Drohne in alle Drehrichtungen kümmern, dabei loslassen der Steuerknüppel die vorhanden Drehungen des Quadrokopters beibehalten werden. Dieser Modus wird von Freestyle und Rennpiloten meist verwendet. [6]

Neben dem Multikopterfliegen stellt für Renn- und Freestyle-Quadrokopterpiloten das Training ein wichtiger Bestandteil dar, um die Bedienung des Quadrokopters 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, wodurch 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 ist herzustellen. 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 zu 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 es die Fernsteuerung von ausgewählten Herstellern mittels USB zu verbinden. Dadurch wird die Fernsteuerung am Computer als USB-HID-Joystick erkannt [8]. Die zweite Möglichkeit ist es, den Quadrokopter auf dem die Firmware Betaflight vorhanden ist mittels USB anzustecken und diesen als Empfänger für die Fernsteuerung zu verwenden [10]. Da diese zwei Möglichkeiten jeweils USB zur Datenübertragung verwenden, sind diese Varianten nicht für alle Endgeräte geeignet. Die letzte Möglichkeit ist es an Fernsteuerung mit Erweiterungsmodulschacht ein Hardwaremodul zu verwenden, welche die Daten mittels Bluetooth übertragen. Hierbei ist als einziges bekanntes Modul, dass Modul des Unternehmens Orqa zu nennen [11]. Dieses Modul ist jedoch nur für eine Typ von Modulschächten geeignet, nämlich des Typs JR und nicht für Modulschächte des Typs Lite.

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 unterstützen 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. Auch soll die Kommunikation zwischen dem ESP32-Entwicklerboard und der Fernsteuerung mittels des Modulschachts der Fernsteuerung implementiert werden. Dafür soll auf eins der vorhanden Protokolle der Fernsteuerung zurückgegriffen werden, damit die Firmware der Fernsteuerung nicht angepasst werden muss. Der letzte Bestandteil dieses Aufgabenbereichs ist es weitere Möglichkeiten der Eingabe und Ausgabe an dem ESP32-Entwicklerboard zu implementieren. 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 dem integrierten Analog-Digital-Wandlers (ADCs) des ESP32-Entwicklerboard geschehen, da der Akkustand bei HID-Geräten bereitgestellt werden muss.

2.2 Platinendesign

In diesen Aufgabenbereich soll der erstellte Steckbrettaufbau der während der Softwareentwicklung benötigt wurde in eine Platine umgewandelt werden. Bestandteile dieser Platine soll zum einen 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 Ausgabeelemente mit dem ESP32-Modul, um die Bedienung des Erweiterungsmoduls zu vereinfachen. Zur einfacheren Umsetzung soll auf bereits vorhandene Referenzdesigns des ESP32-Entwicklerboards zurückgegriffen werden.

2.3 Gehäuseerstellung

Im letzten Aufgabenbereich soll ein Gehäuse für die erstellte Platine erstellt 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 Einstecken 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 Segmenten, auch Deskriptoren genannt, des ROMs des jeweiligen USB-Geräts abgespeichert. Ein Gerät, welches sich in der HID-Klasse befindet, hat wie in Abbildung 1 zu sehen ist drei Deskriptoren. Zunächst einmal den HID Deskriptor, welcher alle weiteren benötigte Deskriptoren für USB-HID-Geräte auflistet. Der zweite optionale Deskriptor ist der physikalische Deskriptor. Dieser stellt Informationen an das System bereit, wie einzelne Teil des HID-Geräts von einem Menschen bedient werden sollen. Dieser wird in dieser Arbeit nicht genauer betrachtet. Der letzte Deskriptor ist der Report Deskriptor. Mittels diesem Deskriptor wird die Struktur der übermittelten Daten zwischen dem Rechnersystem und dem HID-Gerät beschrieben. [13, S. 4f.]

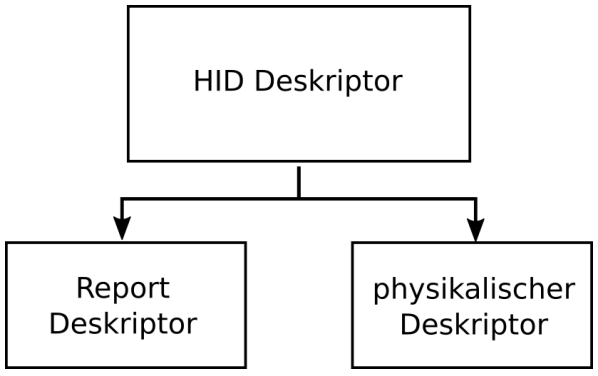

Abbildung 1: HID Deskriptorenhierarchie; abgewandelt von [13, S. 4]

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 dabei 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, 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]

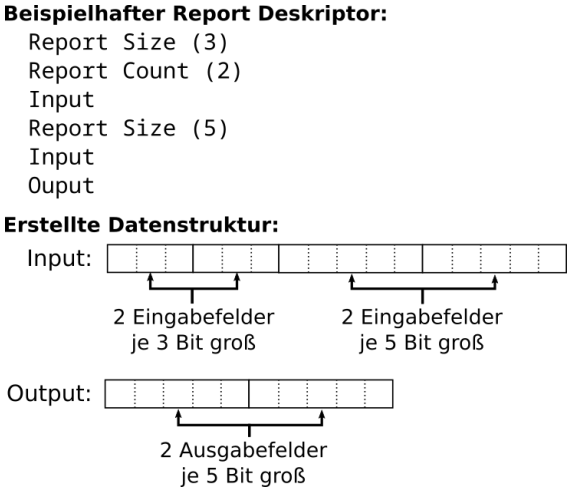

Abbildung 2: Beispielhafte Verwendung von Elementen, um eine Datenstruktur zu definieren; abgewandelt von [13, S. 24]

Unter der Gruppe der Hauptelemente gibt es fünf Element-Marker. Ein häufig verwendeter Marker ist der Input-Marker, mittels diesem können Ausgabedatenfelder definiert werden. 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 Minimun, Logical Maximum, Report ID, Report Size und Report Count. Mittels dem Marker 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 welchen die Hauptmerkmale auf Robustheit, einen geringen Stromverbrauch und geringe Kosten gelegt wurde. Bluetooth wir 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 einen niedrigen Stromverbrauch, was durch eine geringere Datenrate und eine geringere Einschaltdauer während den Datenaustausch 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 lizenzfreien 2.4 GHz Industrial, Scientific and Medical (ISM)-Band von 2402 MHz bis 2480 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 vorhanden [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.].

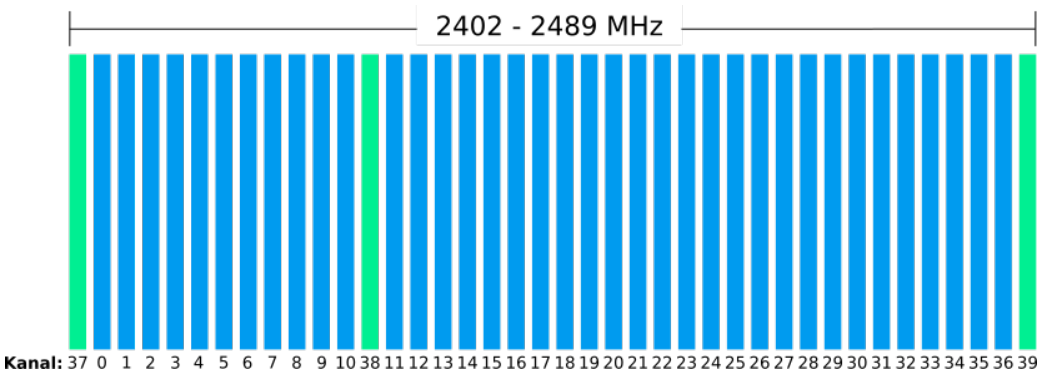

Abbildung 3: Frequenzband mit Kanälen von BLE; abgewandelt von [17, S. 4]

Die Kompatibilität zwischen Bluetooth-Geräten wird durch sogenannte Profile sichergestellt. Profile beschreiben dafür Funktionen und Eigenschaften von jeder Schicht im Bluetoothsystem [16, S. 277]. Ebenso werden die benötigten Nachrichten und Prozeduren für die verwendeten Profile 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 verschickt 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 Peripherial. 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 finden mittels des Host Controller Interface (HCI) statt [16, S. 1735]. [16, S. 193]

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. Auch gibt es eine Komponente welche für die Datenflusskontrolle, die Datenbestätigung und für die erneute Übertragung von Paketen zuständig ist. Die letzten Komponenten in der Verbindungsschicht ist für den Zugriff auf das Radiomedium zuständig. Dafür gibt es einen

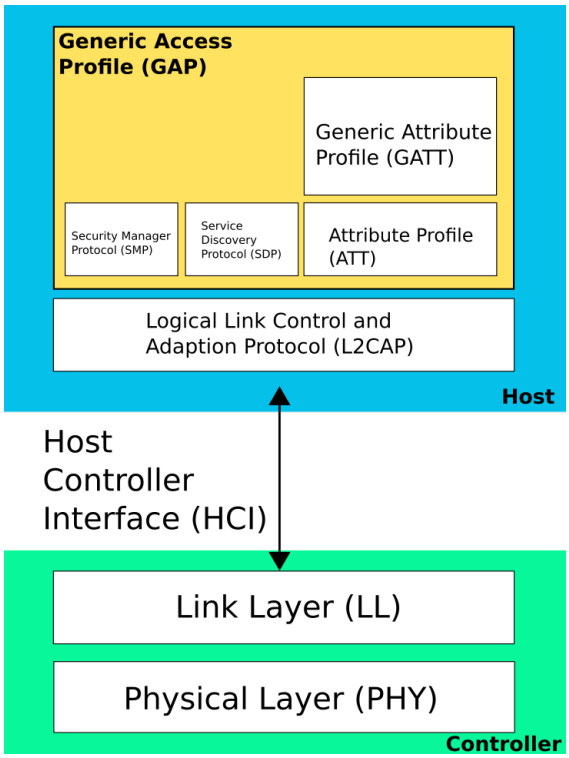

Abbildung 4: Benötigte Komponenten eines BLE-Geräts; abgewandelt von [16, S. 203, S. 1245]

Scheduler, welcher Zeitschlitze 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 bereitgestellt. 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 der 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 allen BLE-Geräte verschickt wird. Zum anderen gibt es sich nicht verändernde private Adressen, welche von Geräten ausgerechnet werden können, welche schon einmal eine Verbindung mit einem bestimmten Gerät aufgebaut hatten. Damit können Geräte, welche schon einmal mit einem anderen Gerät verbunden waren, überprüfen, ob es sich um ein bereits bekanntes Gerät handelt. [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]

Zu guter Letzt, beschreibt GAP noch die verschieden Sicherheitsmodi, welche durch die verschiedenen BLE-Rollen implementiert sein müssen [16, S. 1337].

Service Discovery Protocol (SDP)

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

Security Manager Protocol (SMP)

SMP definiert Methoden zum Verbindungsaufbau 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 Verbindungsaufbau und der dazugehörige Schlüsselaustausch für die Identifizierung der Geräte erfolgt in 3 Phasen. Die erste Phase ist die Anfrage für einen Verbindungsaufbau. Die zweite Phase, welche nach einer erfolgreichen Anfrage erfolgt, ist die Generierung eines Schlüssels mit einer kurzen oder langen Lebenszeit. Die letzte Phase ist die Bereitstellung der generierten Schlüssel an die Gegenstelle. [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] – zu lesen, zu schreiben und zu entdecken, welche sich auf dem Gerät mit der Rolle Server befinden [16, S. 1409]. Ein Gerät kann simultan in der Rolle Server und Client sein [16, S. 279].

Ein Attribut besteht jeweils aus drei Eigenschaften. Die erste Eigenschaft ist der Attribut-Typ, welcher durch eine universal unique identifier (UUID) definiert wird und in SDP definiert sind. Die zweite Eigenschaft ist der Attribut-Handle. Der Attribut-Handle ist ein einzigartiger Identifikator für ein Attribut auf einem Gerät mit der Rolle Server. Durch das Handle ist ein Attribut eindeutig auf einem Gerät definiert. Die letzte Eigenschaft eines Attributs sind die Berechtigungen, welche durch eine höhere Schicht definiert werden muss. [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 stellt wie ATT zwei Rollen – den Server und den Client – bereit. Ebenso definiert GATT das Format der Daten, welche auf dem GATT-Server gespeichert werden dürfen, in sogenannten Profilen. 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 ein Wert, sowie eine Menge von Deskriptoren. Durch diesen Aufbau ist es einen Client möglich die Daten eines bestimmten Profils auszulesen 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 einen Client und Server sichergestellt werden kann, wie in Abbildung 6 zu sehen ist. [16, S. 1470]

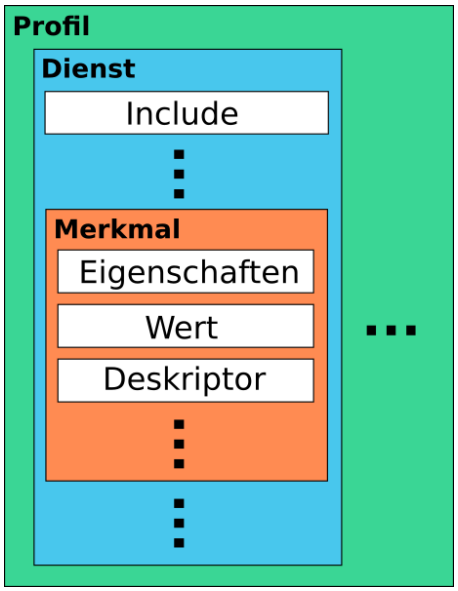

Abbildung 5: GATT Hierarchie; abgewandelt von [16, S. 281]

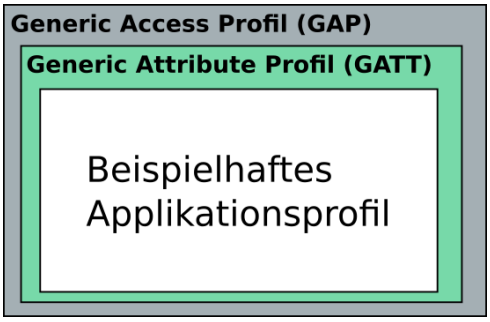

Abbildung 6: Hierarchische Verwendung von Profilen; abgewandelt von [16, S. 1468]

Ein Dienst stellt unter GATT eine Ansammlung von Daten dar, um ein bestimmtes Verhalten durch das vorhandene Gerät darzustellen. Ein Dienst kann zur Vereinfachung der Verhaltensdarstellung weitere 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 einen 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, mit dem 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, welche den 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 iPhone/iPad/IPad (MFi) lizenzierte Technologie, zur Verbindung zu Apple Geräte, verwendet – beispielsweise MFi Game Controller – von Apple geprüft werden. Eine Ausnahme stellen dabei BLE-Geräte dar. [18] Jedoch müssen diese Geräte einige Sollanforderungen in Bezug auf BLE erfüllen. Eine Anforderung ist, dass alle drei Advertising-Kanäle bei jeden 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 soll, welche in der Tabelle [19, S. 187] stehen. Eine weitere Anforderung ist, dass 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 kann [19, S. 191]. Ebenso sollte Zubehör im GATT-Profil das Merkmal mit dem Namen *Gerätename* implementiert haben und durch das Apple-Gerät 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 durch Apple ist, dass BLE-Geräte die Fähigkeit besitzen müssen private Geräteadressen auflösen zu können.[19, S. 189].

Auch geben Apple-Geräte 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 gefilteren Dienste sind: GAP, GATT sowie BLE HID. [19, S. 192]

3.2.4 HID over GATT Profile (HOGP)

In diesen Abschnitt der Arbeit, wird nur auf die Anforderungen eines HID-Geräts – stellt einen GATT-Server bereit [20, S. 9] – und nicht eines HID-Hosts – stellt einen GATT-Client bereit [20, S. 9] – eingegangen, da eine Implementierung des HID-Hosts nicht in diesem Projekt benötigt werden.

Mittels dem HID over GATT Profile werden Prozeduren und Fähigkeiten definiert, welches ein BLE-HID fähiges Gerät benötigt, um als HID-Gerät von HID-Host 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, ein oder mehrere Instanzen des Batteriedienstes, sowie nur eine Instanz des Gerätinformationsdienstes 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.

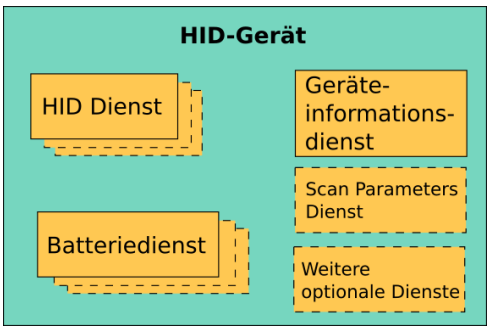

Abbildung 7: Benötigte Dienste eines HID-Geräts; abgewandelt von [20, S. 11]

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

HID-Dienst

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

Report Merkmal

Das Merkmal *Report* ist optional. Dieses stellt jedoch ein wichtiges Merkmal dar, 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. Zu Unterscheidung der verschiedenen Reports muss jeweils ein Referenz-Merkmalsdeskriptor hinzugefügt werden, welche eine eindeutige Report-ID und den Report-Typen enthält. Als zusätzliche Bedingung müssen in allen Report-Merkmalen vom Typ Eingabe ein Konfigurationsdeskriptor vorhanden sein. Mittels diesen Deskiptor 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*, wir 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 dem 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, welche in diesem Merkmal enthalten sind, 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. Auch müssen alle HID-Dienste als primärer Dienst initialisiert werden und während der Entdeckungsphase für einen Verbindungsaufbau als möglicher Dienst angegeben werden. [20, S. 13f.]

Batteriedienst

Mittels diesem Dienst 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 den 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 einen 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 vorhanden 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 Zertifizierungsinfromationen 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 diesem optionalen Profil beziehungsweise Dienst, stellt ein GATT-Server einen 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, aber ist nun unter den Namen Fluoride zu finden [28], [29].

Apache NimBLE ist ein Open Source BLE-Stack, welcher vollständig 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-Erweiterungsmodule [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-Erweiterungsmodule 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]

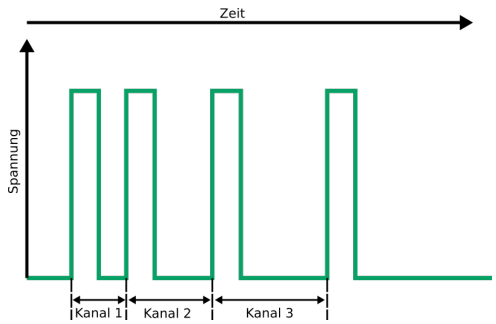

Abbildung 8: Darstellung einzelner Kanäle im Zeitverlauf bei PPM; abgewandelt von [36]

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. Bits werden in einer nicht invertierten Weise 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	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 und das Datenfeld 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
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 Gernerator-polynom lautet dafür: 0xD5 [40]

3.3.3 SBUS

Das SBUS-Protokoll ist ein invertiertes (hohes Spannungsniveau = logisch niedrig [41]) serielles Protokoll, über eine Leitung mit einer Symbolrate von 100000 Baud. 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 [41]

Für die Synchronisation zwischen Sender und Empfänger gibt es Lücken innerhalb eines Pakets [41]. 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-Prokoll wird für die Kommunikation zwischen einer Fernsteuerung auf der OpenTX läuft und einem 2,4 GHz Erweiterungsmodul verwendet. Mittels diesem Protokoll 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.

3.4 Mikrocontroller ESP

TODO: Mikrocontroller ESP

3.5 FreeRTOS

TODO: FreeRTOS

3.6 BITMAP Schriftarten

TODO: BITMAP Schriftarten

3.7 libevdev

TODO: libevdev

4 Umsetzung

TODO: Umsetzung

5 Validierung und Gegenüberstellung

TODO: Validierung und Gegenüberstellung

5.1 Validierung des Funktionsumfangs

5.2 Gegenüberstellung BLE-Modul und USB-Verbindung

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/commecial-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] *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.

[41] *Protocol decoder:sbus_futaba*, https://sigrok.org/wiki/Protocol_decoder:Sbus_futaba, Aufgerufen am: 22. Oktober 2022, sigrok.

[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.

Anhang

TODO: Anhang