HAW HAMBURG

Bachelorarbeit

Rüdiger Willing

Softwareentwicklung für eine ETCS-basierte Modelllokomotive

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Computer Science and Engineering
Department of Information and Electrical Engineering

Rüdiger Willing

Softwareentwicklung für eine ETCS-basierte Modelllokomotive

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Elektro- und Informationstechnik*
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Pawel Buczek
Zweitgutachter: Prof. Dr. Annabella Rauscher-Scheibe

Eingereicht am: 01.02.2021

Rüdiger Willing

Thema der Arbeit

Softwareentwicklung für eine ETCS-basierte Modelllokomotive

Stichworte

Softwareentwicklung, ETCS, Modelllokomotive, Mikrocontroller, STM32, CubeIDE, FreeRTOS, TCPHH

Kurzzusammenfassung

Diese Bachelorarbeit bildet die Grundlage für die nachfolgenden Entwicklungen einer Modelleisenbahn, die angelehnt an das European Train Control System (ETCS) arbeiten soll. In diesem Teil des Projekts geht es vorrangig um die Softwareentwicklung und erste Implementierung auf einem Mikrocontroller sowie dem Erstellen einer später als Grundlage nutzbaren Software-Architektur.

Rüdiger Willing

Title of Thesis

Software development for an ETCS-based model locomotive

Keywords

Software development, ETCS, model locomotive, microcontroller, STM32, CubeIDE, FreeRTOS, TCPHH

Abstract

This bachelor thesis forms the basis for subsequent developments of a model railroad, which is inspired by the European Train Control System (ETCS). This part of the project is primarily dealing with software development, initial implementation on a microcontroller and the creation of a software architecture that can later be used as a basis.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

API Application Programming Interface

Balise Eurobalise

BLDC bürstenlosen Gleichstrommotor

BSP Board Support Package

CB Circular Buffer

DC Duty-Cylce

DMA Direct Memory Access

DMI Driver Machine Interface

EoA End of Authority

EOLM End-Of-Loop-Marker

ERTMS European Rail Traffic Management System

ETCS European Train Control System

ETML European Traffic Management Layer

EU Europäische Union

EVC ETCS-Fahrzeugcomputer

GPOS General Purpose Operating System

GSM Global System for Mobile Communications

GSM-R Global System for Mobile Communications – Rail(way)

HAL Hardware Abstraction Layer

HTerm H-Terminal

IDE Integrated Development Environment

Indusi induktive Zugsicherung

LEU Lineside Electronic Unit

LoA Limit of Authority

LSB Least Significant Bit

LZB Linienzugbeeinflussung

MA Fahrterlaubnis

MAL Movement Authority List

MC Mikrocontroller

MSB Most Significant Bit

OS Betriebssystem

PWM Pulsweitenmodulation

PZB punktförmige Zugbeeinflussung

RBC Radio Block Center

RS-232 Recommended Standard 232

RTOS Real-Time Operating System

Sifa Sicherheitsfahrschaltung

SoC System-on-a-Chip

STM Specific Transmission Module

TEN Transeuropäische Netze

UART Universal Asynchronous Receiver Transmitter

UML Unified Modeling Language

USART Universal Synchronous/Asynchronous Receiver Transmitter

WP Waypoint

WPP Waypoint Plan

Listings

1 Einleitung

1.1 Motivation

Mit der steigenden Zahl an zugelassenen Personenkraftwagen (Pkw) steigt auch der Verkehrsdruck innerhalb der großen Städte und immer mehr Städte schließen die Innenstadt für den Privat- und Pendelverkehr. Eine Auslagerung des Personenverkehrs auf die öffentlichen Verkehrsmittel, darunter auch Regional- und S-Bahnen, würde eine Verringerung des Verkehrsdrucks auf den Straßen unterstützen, gleichzeitig aber den auf den Schienenverkehr erhöhen. Diesem Druck muss entgegengewirkt werden, um eine attraktive Alternative zu dem Individualverkehr bieten zu können.

Der Verlauf der pro Jahr gezählten Pkw in Hamburg wird in Abbildung 1.1 dargestellt.

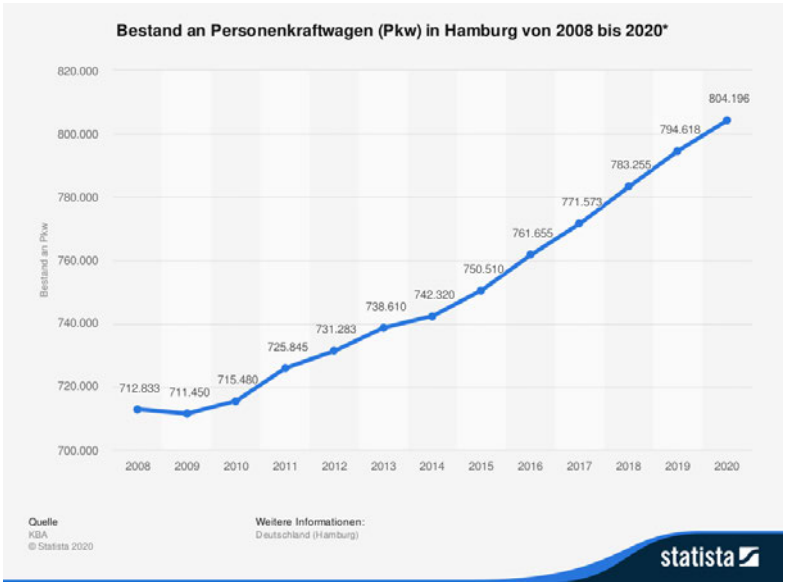

Abb. 1.1: Bestand an Personenkraftwagen in Hamburg von 2008 bis 2020 [33]

Neben dem Nahverkehr wird auch der Fernverkehr durch ein erhöhtes Fahrgastaufkommen gefordert sowie durch weitere Mitbewerber, die dieselben Gleisnetze verwenden müssen.

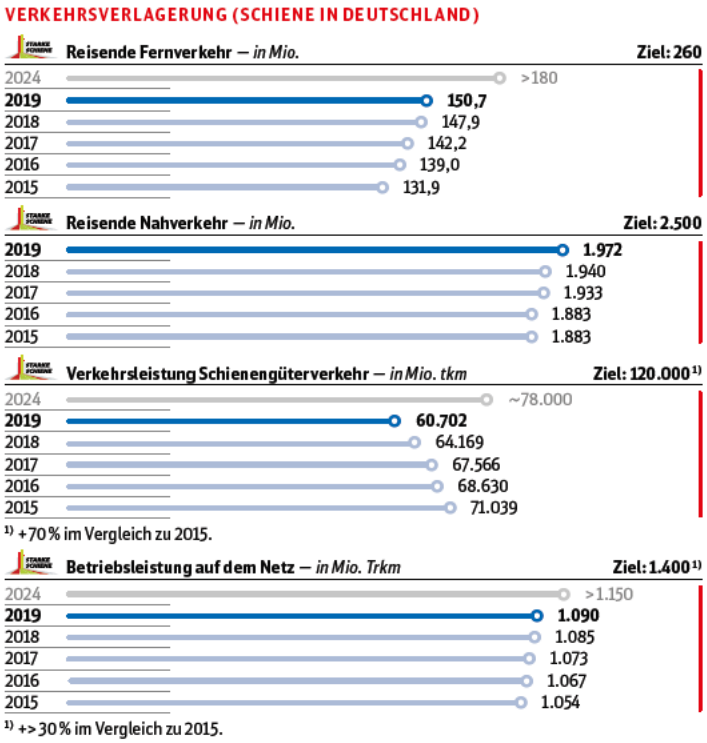

Abb. 1.2: Verkehrsverlagerung (Schiene Deutschland) [22]

Wie den Grafiken aus Abbildung 1.2 entnehmbar ist, wachsen Nah- und Fernverkehr seit 2015 stetig, während der Schienengüterverkehr abnimmt. Insgesamt steigt die Betriebsleistung auf dem Netz immer weiter an.
Um weiterhin in diesen drei Kategorien konkurrenzfähig bleiben zu können, müssen die Abläufe von ihrem starren, vom Stellwerk abhängigen Verkehrsfluss zu einem modernen, vernetzten und vor allem dynamischen Verkehrsfluss migriert werden.
Die Einführung des European Train Control System (ETCS) dient genau diesem Schritt.

Nicht nur dem Schienenverkehr in Deutschland wird damit geholfen. Das ETCS soll EU-weit bestehende Systeme ersetzen und damit dem grenzübergreifenden Verkehr zugutekommen.
Es wird damit ein großflächigeres und leichter befahrbares Schienennetz für die EU ermöglicht.

Auch für den Schutz des Klimas kann das ETCS einen Beitrag leisten. Wenn die Bahn weiterhin konkurrenzfähig bleibt und auch ihr Image der verspäteten Züge ablegen kann, könnte die Entwicklung dazu führen, dass mehr Menschen vom Pkw auf die Bahn wechseln.

Abgesehen von den direkten Auswirkungen auf den Bahnverkehr und den Klimaschutz ist seit der Einführung von Industrie 4.0 die Idee der Vernetzung aller Komponenten eines Systems immer weiter fortgeschritten und ermöglicht neue Wege der Leistungsmessung und, im Falle der Bahn, der Leistungssteigerung.

Durch das Erstellen einer auf ETCS basierenden Modelleisenbahn für die HAW, ist es später möglich, verschiedenste Experimente im Rahmen der Thematik durchzuführen.

1.2 Zielsetzung

Diese Arbeit soll die Grundlage für dieses ETCS-Modell bilden und die ersten Ergebnisse zur Umsetzung des Projekts liefern.

Durch die Analyse des Realsystems und der Vorgabe einiger Randbedingungen soll in der Anforderungsanalyse die erste Zusammenfassung der an das Eisenbahnmodell gestellten Anforderungen erarbeitet werden. Mithilfe dieser Anforderungen wird im Design ein Hardware-Konzept und ein ausführliches Software-Konzept erstellt. Auf diesem Software-Konzept basierend wird der erste Prototyp der Eisenbahnsoftware umgesetzt. Das genutzte Kommunikationsprotokoll wird zusammen mit Software entwickelt und festgelegt.

Der vollständige Ablauf der Entwicklung soll dokumentiert werden und mithilfe der Evaluation die Machbarkeit des entwickelten Design festgestellt werden. In der Schlussbetrachtung wird anschließend das Fazit gezogen und mögliche Optimierungsmöglichkeiten des Designs diskutiert.

Die Dokumentation der Arbeit ist insbesondere deshalb wichtig, da sie späteren Projekten als Grundlage für weitere Umsetzungen dienen soll.

1.3 Aufbau der Arbeit

Der Aufbau der Arbeit ist stark an eine Architekturentwicklung, mit anschließender Umsetzung und Evaluation, angelehnt. Zuerst werden jedoch in den Grundlagen wichtige Komponenten der Arbeit dargestellt. Anschließend wird die Architektur in den Kapiteln Anforderungsanalyse und Design entwickelt und im Anschluss in der Umsetzung implementiert. Zum Schluss wird die Umsetzung in der Evaluation getestet und in der Schlussbetrachtung bewertet.

2 Grundlagen

In den Grundlagen werden die Themen dargestellt, die für das Verständnis der restlichen Arbeit erforderlich sind.

Darunter fallen der Funktionsumfang und der Aufbau des European Train Control System, auf dem die Modelleisenbahn basieren soll, sowie die programmiertechnischen Themen UART, FreeRTOS und Hardware Abstraction Layer, die im Design und der Umsetzung verwendet werden.

2.1 European Train Control System (ETCS)

Dieses Kapitel setzt sich mit dem Grundgedanken und Aufbau des European Train Control System auseinander. Besonderes Augenmerk wird dabei auf die technische Funktion gelegt. Der Großteil der in diesem Kapitel enthaltenen Informationen werden dem Buch „Eine Einführung in das ETCS“ [40] entnommen.

2.1.1 Stand vor ETCS

Mit dem technischen Fortschritt entwickeln sich immer weitere Möglichkeiten für den Eisenbahnverkehr. So werden in Deutschland Systeme genutzt, die den Zugführer bei seiner Arbeit unterstützen oder z. B. bei fehlender Reaktion eingreifen.

Diese Systeme nennen sich Zugbeeinflussungssysteme und existieren parallel zu Beschilderung und Signalen. Eines der ersten Systeme ist die induktive Zugsicherung (Indusi). Sie wird auch heute noch in weiterentwickelter Form verwendet. So existieren in Deutschland derzeit die punktförmige Zugbeeinflussung (PZB), Linienzugbeeinflussung (LZB) und Sicherheitsfahrschaltung (Sifa), welche in den folgenden Abschnitten beschrieben werden. [40]

Die PZB (auch PZB 90 genannt und Nachfolger der Indusi) ist ein Gleismagnet und löst beim Überfahren mit der Fahrzeugantenne eine Routine aus, die vom Lokführer z. B. das Abbremsen auf eine bestimmte Geschwindigkeit sowie das Drücken einer Wachsamkeitstaste verlangt. Mit dieser wird bestätigt, dass das zum Gleismagnet gehörige Signal wahrgenommen wird und eine entsprechende Handlung erfolgt. Während die PZB auch durch die Bauform nur auf einen Punkt beschränkt ist, kann die LZB eine längere Beeinflussung oder Überwachung bewerkstelligen. Statt eines Gleismagneten kommt hier ein Linienleiter zum Einsatz. Dies ist eine längere Leitung im Gleisbett, die als Antenne fungiert. Sie dient unter anderem dazu, Geschwindigkeitsreduzierungen im Voraus anzukündigen, sodass der verlängerte Bremsweg bis zu der Stelle der Geschwindigkeitsreduzierung ohne ein Durchrutschen möglich ist. [40]

Zusätzlich gibt es mit der Sifa ein System, das den Zug zum Stillstand bringt, sollte der Triebfahrzeugführer nicht mehr handlungsfähig sein. Hierzu muss alle 30 Sekunden ein Taster oder Pedal betätigt werden, andernfalls wird erst optisch, dann akustisch davor gewarnt, dass eine automatische Bremsung bevorsteht. [23]

In vielen Ländern der EU gibt es ähnliche Systeme, die aber nicht grundsätzlich untereinander kompatibel und unter Umständen sogar patentrechtlich geschützt sind. [40]

2.1.2 European Rail Traffic Management System

Da sich der Bahnverkehr mit der Gründung der EU und der späteren Entstehung der transeuropäischen Netze (TEN) auch grenzüberschreitend bewegt, wird es notwendig, einen einheitlichen Standard für Steuerung und Management zu entwickeln. Die Entwicklung eines solchen Standards wurde Mitte der 90er Jahre gestartet und das daraus entstehende System wird European Rail Traffic Management System (ERTMS) genannt. Es bestand ursprünglich aus drei Komponenten. [40]

- Global System for Mobile Communications – Rail(way)
- European Train Control System (ETCS)
- European Traffic Management Layer (ETML)
  Früher: Train Management Layer. Seit 2018 keine Erwähnung mehr seitens Europäischer Eisenbahnagentur, impliziert Abbruch. [21]

GSM-R Das Global System for Mobile Communications – Rail(way) (GSM-R) ist eine speziell für den Eisenbahnverkehr weiterentwickelte Version des GSM Standards mit bahnspezifischen Zusatzfunktionen. Dazu gehören z. B. die Möglichkeiten, Broadcast Anrufe zu tätigen, ein Link Assurance Signal abspielen zu lassen und das Implementieren eines Prioritätssystem mit Abstufungen für Notruf bis hin zu einfachen Kommunikationsanrufen. [28, 29, 30]

ETCS Ein Zugbeeinflussungssystem stellt dem Triebwagenfahrer Informationen zum bestimmungsgemäßen Führen des Zuges auf den jeweiligen Streckenabschnitten bereit.

Die Zugbeeinflussungssysteme, die derzeit verwendet werden, sind in Abbildung 2.1 dargestellt.

Abb. 2.1: Zugbeeinflussungssysteme: Schild, Signal und Gleismagnet (eigenes Foto)

Mit dem Ausbau der ETCS-Infrastruktur bleiben vorhandene Systeme wie Schilder und Signale erhalten. Die bestehende Beschilderung wird durch zusätzliche Schilder erweitert (z. B. ETCS-Haltetafeln, die Haltesignale ersetzen, oder ETCS-Standortsignale). Die Beschilderung wird aber je nach Ausrüstungsstufe überflüssig. Die größten Änderungen betreffen die automatischen Systeme (PZB und LZB). Bestimmte, in der EU eingesetzte Systeme fallen durch den Bestandsschutz in die sogenannten Class-B-Systeme. Diese Systeme dürfen parallel zu den durch ETCS eingeführten Komponenten genutzt werden, jedoch teilweise nur für Strecken, die nicht dem TEN angehören oder auf denen keine ETCS-Züge verwendet werden. [24]

Je nach Ausrüstungsstufe können durch die Einführung unterschiedliche Betriebsarten realisiert werden (siehe Ausrüstungsstufe, beschrieben in Unterunterabschnitt 2.1.4).

2.1.3 Motivation und Vorteile

Die grundlegende Motivation hinter ETCS und somit auch ERTMS ist eine verbesserte Wirtschaftlichkeit und Organisation des grenzüberschreitenden Eisenbahnverkehrs. So sind die konventionellen Systeme aus Deutschland nicht mit denen aus Frankreich (z. B. Crocodile, KVB) kompatibel und der Triebwagen muss bei der Grenzüberschreitung unter Einkalkulieren längerer Warte- und Standzeiten an der Grenze gewechselt werden. Eine weitere Möglichkeit ist die Ausstattung mit Komponenten beider Länder. Letzteres führt zu erhöhten Kosten in Produktion und Wartung. [40]

Es werden folgende Erwartungen mit der Einführung des ERTMS verbunden [40]:

- Schaffung eines freien Marktzugangs.
  Eine harmonisierte Normung ermöglicht die europaweite Fertigung von Komponenten durch verschiedene Hersteller. Das Monopol landesspezifischer Hersteller wird aufgelöst.
- Interoperabilität.
  Grenzüberschreitende Verbindungen können einfacher realisiert werden, ohne eine Mehrfachausrüstung oder einen Triebwagenwechsel durchführen zu müssen.
- Sicherer und qualitätsgerechterer Betrieb.
  Nationale Sicherheitseinschränkungen können durch ERTMS behoben werden.
- Erhöhung der Streckenleistungsfähigkeit.
  Die Streckenauslastung kann an die steigende Nachfrage angepasst werden, indem, z. B. durch dynamische Raumabstände, die Streckenkapazität erhöht wird.
- Reduktion der Lebenszykluskosten.
  Investitionskosten können durch Reduzierung oder Verzicht ortsfester technischer Systeme gesenkt werden.

2.1.4 Technische Aspekte des ETCS

Technische Komponenten des ETCS

Die technischen Komponenten des ETCS lassen sich in drei Bereiche eingrenzen:

- An der Strecke angebrachte Komponenten
- Am Fahrzeug angebrachte Komponenten
- Komponenten, die den Datenaustausch beschreiben

Streckenkomponenten Auf oder an der Strecke sind folgende technische Komponenten zu finden:

- Eurobalise
  Die Eurobalise übernimmt die Aufgabe des PZB, jedoch mit einem wichtigen Vorteil. Sie wird über die Fahrzeugantenne mit Energie versorgt und benötigt keine eigene Spannungsversorgung. Balisen werden mittig im Gleisbett auf den Schwellen oder Balisenträgern montiert und können in zwei Ausführungen verbaut werden:

  - Festdatenbalise
  - Transparentdatenbalise

  Die Festdatenbalise ist komplett autark von weiteren Streckenkomponenten. Sie enthält im Gegensatz zur Transparentdatenbalise jedoch nur ein fest definiertes Telegramm. Dieses enthält dabei bahnbetriebliche Informationen wie Balisennummern und Balisenkoordinaten. Darüber kann eine genaue Positionsbestimmung des Zuges durchgeführt werden. Die Transparentdatenbalise (auch schaltbare Balise genannt) kann an eine LEU angeschlossen werden und sendet das an der Kabelschnittstelle anstehende Signal transparent (ohne Zwischenspeicherung) an die Fahrzeugantenne. [40]

- Lineside Electronic Unit (LEU)
  Eine Lineside Electronic Unit wird streckenseitig an vorhandene Signale angeschlossen und dient dort als Signaladapter. Sie kann die Signalstellung (das „Fahrt“ eines grünes Signals, oder das „Stopp“ eines roten Signals) über eine Transparentdatenbalise in das ETCS einspeisen. Die LEU ist kein standardisiertes System und kommt daher mit proprietären Schnittstellen zum Einsatz. Sie ermöglicht es jedoch, mit vergleichsweise geringem Aufwand an verschiedene Stellwerksbauformen angepasst zu werden. [40]

- Euroloop
  Der Euroloop ist vergleichbar mit der LZB. Er besteht aus einer im Gleis liegenden Antenne, die bis zu 1000 m lang sein darf. Im Gleis muss er mit einem End-Of-Loop-Marker (EOLM) angekündigt werden. Die EOLM ist eine Eurobalise, die in Fahrtrichtung vor dem Loop platziert wird. Ein EOLM ist notwendig, um den sogenannten Crosstalk (das Empfangen von Nachrichten von benachbarten Gleisen) zu verhindern. Wie die Eurobalise brauchen auch Euroloops keine gesonderte

Stromversorgung und erhalten die für den Betrieb notwendige Energie mittels Induktion durch die Fahrzeugantenne. [40]

- Radio Block Center (RBC)
  Das Radio Block Center, auch Funkstreckenzentrale genannt, ist die Kernkomponente der streckenseitigen Ausrüstung. Sie stellt die Verbindung zwischen Fahrzeug, Fahrwegsicherung und anderen Funkstreckenzentralen her. [40]

  - Schnittstelle zur Fahrwegsicherung
    Das RBC erhält vom Stellwerk Informationen über den Sicherungszustand des Streckennetzes und übernimmt diese in ihren eigenen Streckenatlas. Außerdem teilt das RBC dem Stellwerk mit, ob es Signale auf der Strecke dunkel schalten muss, um widersprüchliche Aussagen vom ETCS zu ortsfesten Signalen zu vermeiden. [40]
  - Schnittstelle zu den Fahrzeugen
    Neben den Daten zu Position, Geschwindigkeit und Fahrtrichtung, die das RBC von den Fahrzeugen zum Abgleich mit dem Streckenatlas fortlaufend erhält, erzeugt das RBC auch die Fahrterlaubnis für jeden Zug und nimmt deren Anmeldung in seinem Wirkbereich an. [40]
  - Schnittstelle zu anderen Funkstreckenzentralen
    Bevor ein Zug den Wirkbereich einer Funkstreckenzentrale verlässt, kümmert sich diese um die Voranmeldung im nächsten Bereich und übergibt die streckenseitigen Führungsgrößen. Die benachbarte Funkstreckenzentrale kann den Zug bei Bedarf abweisen oder die beiden Zentralen tauschen sich weiter über die fahrwegbezogenen Informationen aus. [40]

Fahrzeugkomponenten Fahrzeugseitig sind folgende Komponenten zu finden:

- ETCS-Fahrzeugcomputer (EVC)
  Der Fahrzeugcomputer ist die Kernkomponente des ETCS im Fahrzeug. Für den Rechner wird eine Sicherheitsarchitektur durch Mehrheitsentscheidung genutzt, die je nach Hersteller als 2oo2 oder 2oo3 ausgelegt wird.

- Odometrie
  Odometrie wird die Einrichtung genannt, die sowohl Geschwindigkeit als auch Weg misst. Sensoren sind z. B. Wegimpulsgeber, Radarsensoren und Beschleunigungssensoren. Die Ausrüstung kann je nach Hersteller variieren.

- Train Interface Unit (TIU)
  Die Schnittstelle dient als Brücke zwischen ETCS-Fahrzeugcomputer und Fahrzeug. Darüber können Informationen über Schalterstellungen und diversen Führungsgrößen an den EVC übergeben werden sowie Befehle an das Fahrzeug, wie zur Auslösung einer Bremsung.

- Driver Machine Interface (DMI)
  Das DMI besteht aus allen Bedien- und Anzeigegeräten, die im Zusammenhang mit ETCS stehen und ist somit der wichtigste Bestandteil für den Triebwagenführer. Darauf werden alle für die Zugführung relevanten Informationen angezeigt und Eingaben vom Zugführer, z. B. zur Betriebsartwahl, getätigt.

- Juridical Recorder Unit (JRU)
  Zur Ausrüstung von ETCS-Triebwagen gehört ein Fahrdaten-Aufzeichnungsgerät, um eine Rekonstruktion eines Unfallherganges zu ermöglichen.

- GSM-R-Datenfunk
  Das GSM-R dient zur Kommunikation zwischen RBC und Fahrzeug. Zur Sicherheit ist das GSM-R-Modul in zweifacher Ausführung in der Ausrüstung des Fahrzeugs vorhanden.

- Fahrzeugantenne
  Die Fahrzeugantenne ist die Schnittstelle zur Eurobalise im Gleisbett. Sie aktiviert diese beim Überfahren und empfängt das Telegramm der Balise.

- Specific Transmission Module (STM)
  Da mit der Class-B-Regelung auch nationale Zugbeeinflussungssysteme verwendet werden dürfen, benötigt es eine Übersetzung dieser Daten in die ETCS-Sprache. So kann das ETCS auch die Führungsgrößen des nationalen Systems überwachen.

Datenkommunikation Für die Datenkommunikation sind auch mehrere Komponenten bzw. Datendienste vorgesehen. Der Aufbau der Komponenten wird im Folgenden dargestellt:

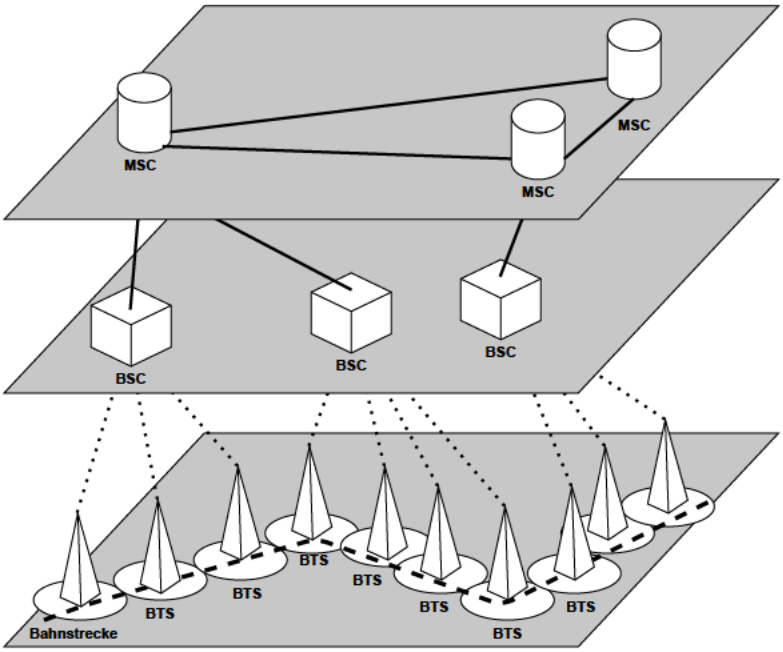

Abb. 2.2: Aufbau des GSM-R Systems (in Anlehnung an [40])

- GSM-R
  - Base Transceiver Station (BTS)
    Das Base Transceiver Station ist die Komponente, welche die streckenseitige Einrichtung über die Funkschnittstelle mit dem Fahrzeug verbindet.
  - Base Station Controller (BSC)
    Einem Base Station Controller sind mehrere BTS untergeordnet. Der BSC übernimmt das „Weiterreichen des Zuges“ von einem BTS zum Nächsten (Handover). Solange die BTS zum selben BSC gehören, führt dieser das Handover selbstständig durch, ansonsten wird die Aufgabe an das MSC weitergereicht. [40]
  - Mobile-service switching centre (MSC)
    Einem Mobile-service switching centre sind mehrere BSC zugeordnet, inklusive der jeweiligen BTS. Sobald ein Zug den Bereich eines BSC verlässt, kümmert sich das MSC um das Handover zur nächsten BSC. Außerdem kommunizieren die MSC untereinander und haben auch Verbindungen zu weiteren Telefonnetzen. [40]
- Circuit Switched Data (CSD)
  Zwischen BTS und ETCS-Triebwagen besteht eine Circuit Switched Data-Verbindung.

Diese Art der Datenübertagung ist sehr ineffizient, weshalb in Räumen mit erhöhter Interaktion (Rangierbahnhöfe und Eisenbahnknoten) das Packet Switched Data-Verfahren eingesetzt wird. Bei einer CSD-Verbindung wird eine feste Verbindung zwischen den Teilnehmern hergestellt. Der genutzte Funkkanal ist dann ausschließlich für diese Kommunikation reserviert und lässt keine weiteren Teilnehmer zu. [41]

- Packet Switched Data (PSD)
  Im Gegensatz zu der CSD-Verbindung können mit dem Packet Switched Data-Verfahren mehrere Teilnehmer eingebunden werden. Statt einer reservierten Verbindung wird ein Paket mit Empfängerdaten und Datentelegramm losgeschickt und zu dem Empfänger durchgestellt. Es ermöglicht eine größere Durchsatzrate, kann aber je nach Telegramm-Route zu längeren Latenzen oder einer falschen Reihenfolge der empfangenen Telegramme führen. [41]

Ausrüstungsstufe

Um die Ausrüstung der Strecke an verschiedene Bedürfnisse und Streckenbegebenheiten anpassen zu können, beinhalten die Spezifikationsdokumente des ETCS verschiedene Level für den Ausbau der Strecke. [40]

Die verschiedenen Ausrüstungsstufen sind:

- ETCS Level 0
  Die unterste Ausbaustufe ist für Strecken vorgesehen, die nicht mit ETCS ausgebaut sind. Es wird der Zug nach Außensignalen gefahren und der ETCS-Fahrzeugcomputer überwacht die maximal zulässige Geschwindigkeit dieser Ausbaustufe. [40]

- ETCS Level STM
  Wenn mit ETCS ausgerüstete Züge auf einer Strecke mit nationalen Zugsteuerungs- und Zugsicherungssystemen eingesetzt werden sollen, ist ein STM-Modul zwingend erforderlich. Was auf dem DMI angezeigt wird, hängt von den nationalen Systemen ab. [40]

- ETCS Level 1
  In dieser Ausrüstungsstufe bleiben nationale Signalisierung und das Stellwerk erhalten. Aber der EVC wird von den Streckenkomponenten gespeist und daraus werden Bremskurven und Höchstgeschwindigkeiten generiert. [40]

- ETCS Level 2
  Level 2 ist das Ziel der DB Netz AG [18, S. 8] und kommt komplett ohne Außensignale aus. Das RBC ist dabei direkt an das Stellwerk angeschlossen. Der EVC berechnet kontinuierlich Geschwindigkeit und Position des Zuges und gibt diese an das RBC weiter. Die Balisen dienen in diesem Level nur noch als Kilometersteine und auf Transparentdatenbalisen kann verzichtet werden. [40]

Betriebsarten

Je nach Zustand des EVC wird zwischen verschiedenen Betriebsarten unterschieden:

- Mit ETCS-Überwachung
  - Full Supervision (FS)
    Sobald alle notwendigen Informationen über den Zug vorliegen (z. B. Geschwindigkeit, Position, Zuglänge und Bremsvermögen), kann der EVC in den vollüberwachten Modus wechseln. In diesem Modus übernimmt das ETCS die Steuerung, solange keine anderen Befehle (z. B. Zwangsstopp) über die Strecke eintreffen. [40]
  - Limited Supervision (LS)
    Während der eingeschränkten Überwachung muss der Triebwagenfahrer weiterhin auf Außensignale achten und diese umsetzen. Eine eingeschränkte Überwachung durch den EVC im Hintergrund wird weiterhin ausgeführt (z. B. Bremswegüberwachung). [40]
  - On Sight (OS)
    Sollten Gleisaußensignale ausgefallen und eine Fahrt auf Sicht notwendig sein, tritt dieser Betriebsmodus in Kraft. Zwar überwacht der EVC noch immer die zulässige Höchstgeschwindigkeit und das Ende der Fahrterlaubnis, jedoch muss der Zugführer jede Änderung der Betriebsmodi quittieren, da er in diesem Zustand einem höheren Maße an Sicherheitsverantwortung unterliegt. [40]
  - Staff Responsible (SR)
    Wenn die Position des Fahrzeuges nicht bekannt ist, wie beim Aufstarten des Zuges, wird dieser Modus verwendet. Der EVC überwacht die national zulässige Höchstgeschwindigkeit, die größtmögliche Distanz, die der Zug in diesem Modus zurücklegen kann und das Ende der Fahrterlaubnis. Der Triebwagenführer

muss sich über den richtigen Verlauf und das Freisein der Strecke versichern sowie auf Außensignale achten. [40]

- Shunting (SH)
  Shunting ist eine Betriebsart extra für das Rangieren, bei dem der EVC auf eine speziell reduzierte Rangiergeschwindigkeit achtet sowie auf das Befahren von zulässigen Rangierbereichen. Auch in diesem Modus übernimmt der Triebwagenführer eine größere Verantwortung und muss daher den Wechsel quittieren. [40]
- Trip (TR)
  Nach dem Ende einer Fahrterlaubnis für einen Streckenabschnitt wechselt der Betriebsmodus automatisch in den Trip-Modus und führt eine Zwangsbremsung bis zum Stillstand durch. Solange der Zug nicht still steht, kann der Zugführer keine weiteren Bedienungen ausführen. [40]
- Post Trip (PT)
  Nach einem Trip (Überfahren der Fahrterlaubnis) muss der Triebwagenführer den Stillstand des Zuges quittieren und wechselt damit in den Betriebsmodus Post Trip. Der EVC überwacht nun die Fahrtrichtung und blockiert ein Weiterfahren. Eine Fahrt in die entgegengesetzte Richtung kann gemäß nationaler Regeln zulässig sein. [40]
- Reversing (RV)
  Die Rückwärtsfahrt ist ein für Gefahrensituationen vorbehaltener Modus. Er ermöglicht es, den Zug auch aus dem vorderen Führerstand heraus rückwärts über einen gesicherten Fahrweg aus der Gefahrensituation zu fahren. Der EVC überwacht hierbei die zulässige Höchstgeschwindigkeit. Den sicheren Fahrweg müssen Stellwerk und RBC gewährleisten. [40]

■ Ohne ETCS-Überwachung

- Sleeping (SL)
  In diesem Modus kann sich der Triebwagen von einem anderen, sich am Anfang vom Zugverband befindlichen Triebwagen fernsteuern lassen. Der EVC hat im ferngesteuerten Fahrzeug keine Sicherheitsfunktion. [40]
- Non Leading (NL)
  Wenn ein Triebwagen nicht an der Spitze eines Verbundes ist, jedoch von einem Triebwagenführer bedient wird, ist der Betriebsmodus Non Leading (Nicht zugführend). Dies ist z. B. beim Schiebe-, Vorspann- oder Zwischendienst der Fall. [40]

- Passive Shunting (PS)
  Wenn der Triebwagen bei Rangierfahrten involviert ist, aber dabei mit einem anderen Fahrzeug gekoppelt ist, überlässt er die Führung dem Fahrzeug mit dem Modus Shunting. [40]
- STM National
  Im Betriebsmodus STM National überwacht der EVC mithilfe des STM-Moduls und den nationalen Streckeneinrichtungen die Fahrt. Der EVC hat zudem Zugriff auf das Bremssystem. [40]
- Unfitted (UN)
  In diesem Betriebsmodus übernimmt der Triebwagenführer die Einhaltung der Außensignale. Die Strecke selbst ist nicht für den ETCS-Betrieb ausgestattet. Der EVC unterstützt jedoch bei der Einhaltung der national zulässigen Höchstgeschwindigkeit der Betriebsart. [40]

■ Bei inaktivem ETCS-Computer

- Isolation (IS)
  Sollte der EVC ständig Zwangsbremsungen durchführen oder eine andere Fehlfunktion zeigen, kann der Triebwagenführer über einen Abtrennschalter den ETCS-Betriebsmodus Isolation einschalten. Der EVC erhält dann keine Informationen der Streckeneinrichtungen und überwacht die Fahrt nicht mehr. In diesem Modus übernimmt der Fahrzeugführer die volle Verantwortung. [40]
- No Power (NP)
  In dieser Betriebsart ist der EVC spannungslos. Normalerweise gilt das auch für den kompletten Triebwagen. [40]
- System Failure (SF)
  Sollte ein sicherheitskritischer Fehler in der ETCS-Ausrüstung festgestellt werden, wechselt der ETCS-Fahrzeugcomputer (EVC) in den Systemfehler-Modus und führt eine Zwangsbremsung bis zum Stillstand durch. Ein sicherheitskritischer Fehler könnte z. B. der Ausfall des GSM-R-Moduls oder der Fahrzeugantenne für die Eurobalise sein. [40]

Funktionsweise

Die Funktionsweise der ETCS-Fahrzeugbeeinflussung besteht im Grunde aus vier Punkten: Der Vergabe und dem Entziehen von Fahrterlaubnissen, der Lokalisierung des Zuges

im Streckennetz, der Überwachung der Geschwindigkeit und der daraus resultierenden Bremskurve sowie der Kommunikation zwischen Fahrzeug und Strecke.

Fahrterlaubnis Die Fahrterlaubnis (MA), auch Fahrbefehl genannt, definiert einen Wegpunkt, bis zu dem die Fahrterlaubnis gilt. Für diesen Wegpunkt gibt es zwei Auslegungen, den End of Authority (EoA), auch Ende der Fahrterlaubnis, und den Limit of Authority (LoA), auch Einschränkung der Fahrterlaubnis. Die beiden Definitionen legen die zulässige Geschwindigkeit nach Erreichen des Wegpunktes fest. Wenn diese über $0\,\mathrm{km/h}$ angesetzt ist, wird von einer LoA gesprochen, während die Zielgeschwindigkeit von $0\,\mathrm{km/h}$ den Wegpunkt zu einem EoA macht. [40]

Eine vollständige Fahrterlaubnis besteht dabei aus mehreren Teilen. Zum einen aus dem Weg bis zum Ende der Fahrterlaubnis. Dieser kann aus mehreren Abschnitten bestehen (vgl. Section(1) und Section(2) in Abbildung 2.3). Diese Abschnitte können zusätzlich Informationen über eine zeitlich begrenzte Gültigkeit enthalten (Section time-out). Je nach Situation ist das Festlegen von Punkten hinter dem Ende der Fahrterlaubnis notwendig. Möglichkeiten hierfür sind Gefahrenpunkt und Durchrutschpunkt sowie die zusätzliche zeitliche Gültigkeit dieser Punkte (vgl. Danger Point und Overlap in Abbildung 2.3). [2, Subset 026, Abschnitt 3.8.3]

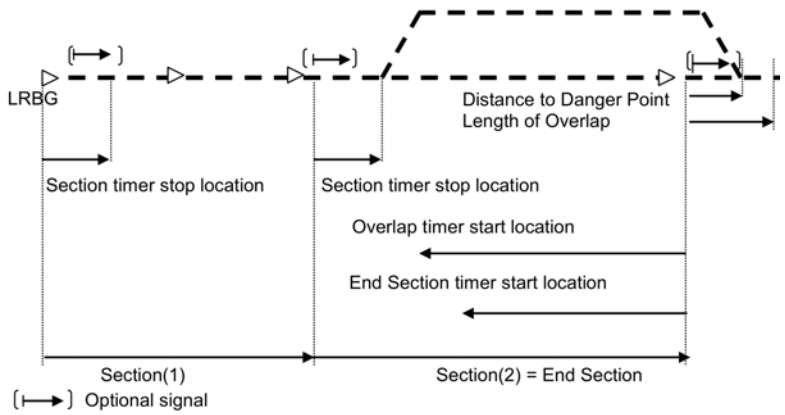

Abb. 2.3: Struktur einer Fahrterlaubnis [2, Subset 026, Abb. 17]

Lokalisierung Zur Lokalisierung einer Bahn werden mehrere Systeme verwendet. Ein Grundlegendes ist das Zählen der Achsenumdrehungen, welche anschließend mit dem Raddurchmesser multipliziert werden. Diese Art der Überwachung der zurückgelegten

Distanz ist jedoch sehr ungenau und fehleranfällig (Durchrutschen mit blockierenden Rädern, beim Beschleunigen durchdrehende Räder). Um die Abweichung von der realen Position so gering wie möglich zu halten, wird diese Methode mit einer genaueren Positionierung kombiniert, um die durch die ETCS-Spezifikation geforderte hohe Genauigkeit zu erreichen. [40]

Die genauere Positionierung wird durch die Synchronisation an Fixpunkten durchgeführt. Als Fixpunkte dienen hier die Eurobalisen. Diese sind üblicherweise in Gruppen angeordnet und bestehen aus bis zu acht einzelnen Balisen. Neben einer eindeutigen Identifikationsnummer enthält das jeweilige Telegramm auch die entsprechende Indexnummer. Durch diese Anordnung kann, neben der Fahrtrichtung, auch die Position des Zuges durch die gegebene Position der Balise bestimmt werden. [40]

Abweichungen können durch sogenannte Linking-Informationen, die die Entfernung zweier Balisengruppen voneinander enthält, korrigiert werden, solange sie sich in einem Toleranzbereich befinden oder der EVC kann eine sicherheitsgerichtete Reaktion durchführen (siehe Abbildung 2.4). [40]

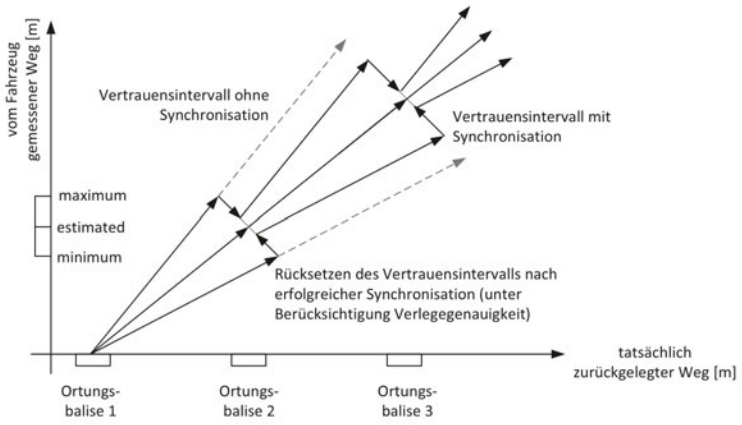

Abb. 2.4: Positionskorrektur durch Linking-Informationen [40, S. 37, Abb. 6.1]

Als weitere Methode gibt es das sogenannte Repositioning. Es erleichtert das Navigieren des Zuges innerhalb von Bahnhöfen, wenn die Position zu Anfang nicht genau genug bestimmt oder überwacht werden kann. Dabei wird dem EVC durch das Überfahren der Repositioning-Balise (A) mitgeteilt, dass die Position unbekannt bleibt und diese Information durch die nächste Balise (B 1-3) nachgeliefert wird (siehe Abbildung 2.5). [40]

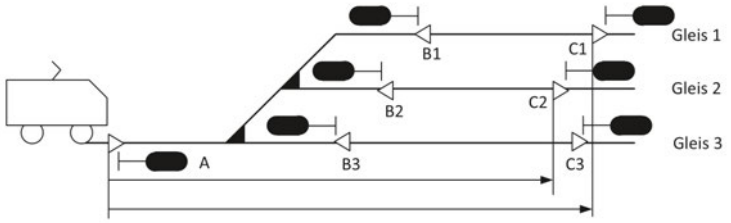

Abb. 2.5: Prinzip des Repositioning [40, S. 37, Abb. 6.2]

Die Balisengruppe B (B 1-3) überträgt im Anschluss je nach Fahrweg neue Entfernungsinformationen zur Balisengruppe C (C 1-3), Geschwindigkeitsprofile und neue Linking-Informationen. [40]

Geschwindigkeitsüberwachung und Bremskurven Durch die Kommunikation mit dem RBC und dadurch mit weiteren streckenseitigen Einrichtungen (z. B. Stellwerk) bekommt der EVC die Informationen zu den Geschwindigkeitsprofilen auf der Strecke mit MA. Es kann weiterhin dazu kommen, dass die statische Höchstgeschwindigkeit der Strecke durch temporäre Beeinflussungen (Baustellen etc.) weiter restriktiert wird. Solche Stellen werden auch Langsamfahrstellen genannt. Der Fahrzeugrechner hat bei der Überwachung der Geschwindigkeit dafür zu sorgen, dass immer die restriktivste Geschwindigkeit eingehalten wird sowie die Höchstgeschwindigkeiten oder andere Einschränkungen des Zuges nicht überschritten werden (vgl. Abbildung 2.6). [40]

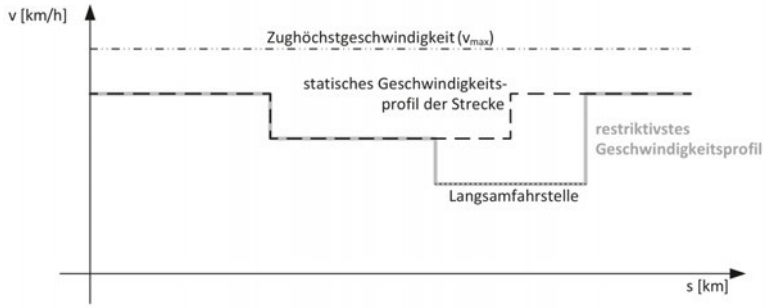

Abb. 2.6: Überwachung des Geschwindigkeitsprofils [40, S. 39, Abb. 6.3]

Passend zur Geschwindigkeit wird auch zu jeder Reduzierung der Geschwindigkeit die Bremskurve berechnet und überwacht, um passend zum Einsetzen der Beschränkung den Zug entweder zum Stehen zu bringen oder auf die gewünschte Geschwindigkeit zu

reduzieren. Für dieses System gibt es verschiedene Eskalationsstufen, die von einer reinen Information bis zu einer Zwangsreaktion des Systems führen können. [40]

Die berechnete **zulässige Geschwindigkeit** wird dem Zugführer auf dem DMI Display angezeigt.

Bei Überschreiten des **Warnlimits** wird eine Warnung abgespielt bis der Fahrer die Geschwindigkeit angepasst oder ein automatisches System dies übernommen hat.

Das **Betriebslimit** beschreibt das Limit, bei dem die erste automatische Bremsung durchgeführt wird, um den Zug bis zum Streckenpunkt der Geschwindigkeitsreduzierung zu verlangsamen oder zum Stillstand zu bringen. Sobald die Geschwindigkeit die Vorgabe unterschreitet, schaltet sich die Bremsung selbstständig wieder ab.

Beim Überschreiten des letzten Limits, dem **Zwangsbremslimit**, wird unter Beachtung diverser physikalischer Grenzen des Bremssystems eine Bremsung zum zuletzt möglichen Zeitpunkt durchgeführt. Dieser Bremsbefehl kann vom Zugführer nicht unterbrochen werden und bringt den Zug zum Stillstand. Je nach nationaler Vorgabe kann sich dieses Verhalten unterscheiden. [40]

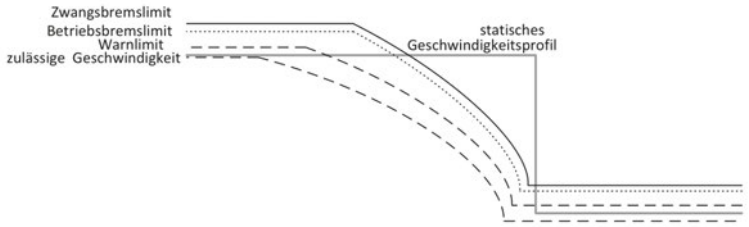

Abb. 2.7: Überwachung der Bremskurven [40, S. 40, Abb. 6.4]

Datenkommunikation Der Austausch der Daten im ETCS wird durch das „System Requirements Specification Subset 026 - Chapter 7 und 8“ [2] definiert. Darin wird spezifiziert, wie die Pakete aufgebaut sind, welcher Präfix zu welcher Variable gehört und wie die Nachrichten aufgebaut sind. Für diese Arbeit wird jedoch ein sehr viel kompakteres Protokoll entwickelt (siehe Anhang B).

2.2 UART

Das Protokoll Universal Asynchronous Receiver Transmitter (UART) ist ein universelles Protokoll zur Datenübertragung zwischen einem Sender und einem Empfänger. Die Datenübertragung findet asynchron statt, was bedeutet, dass kein Taktsignal notwendig ist, um den Start und das Ende eines Frames zu erkennen und anschließend auch verarbeiten zu können. Eine Erweiterung des UART-Protokolls stellt das Universal Synchronous/Asynchronous Receiver Transmitter (USART) dar. Dabei wird vom Sender ein Takt erzeugt, der zusätzlich zu den Datenleitungen übertragen wird. [39]

Im Bereich Mikrocontroller (MC) hat sich UART durchgesetzt, da statt Geschwindigkeit oder Entfernung hier die einfache Implementierbarkeit im Vordergrund steht. So wird UART bei MC meist dafür verwendet, um kleine Mengen an Informationen zwischen Komponenten auszutauschen (z. B. zwischen MC und RFID-Modul) oder das kompilierte Programm auf einen MC zu schreiben. [39]

2.2.1 Hardware Schnittstelle

Es gibt verschiedene Möglichkeiten, einen Anschluss für eine USART-Schnittstelle zu realisieren. Bei der Verwendung mit Mikrocontrollern haben sich meist einfache Pins zum Anschließen oder Anlöten der Leitung durchgesetzt. Wenn beide Teilnehmer auf derselben Platine verbaut sind, wird eine direkte Verbindung realisiert. Alternativ gibt es je nach Einsatzzweck verschiedene Steckverbindungen. Ein Beispiel der Stiftleiste oder einfacher Pins (auch Vias genannt) ist in Abbildung 2.8 gezeigt.

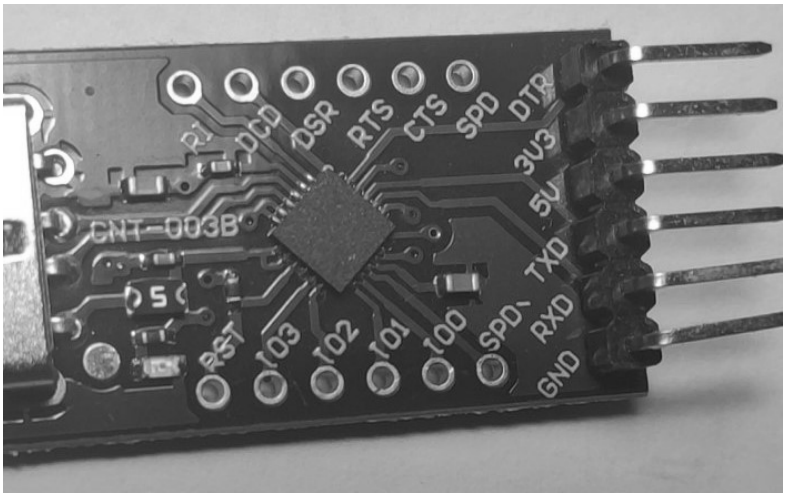

Abb. 2.8: Beispiel einer Hardware-Schnittstelle auf einem USB-UART Adapter (eigenes Foto)

2.2.2 Frame Aufbau

Ein Frame im UART-Protokoll besteht aus einer fest definierten Abfolge und Anzahl von Bits, die aus der am Port anliegenden Spannung interpretiert werden. Ein Frame besteht mindestens aus einem Startbit, einem Stoppbit, einem Paritätsbit und mehreren Datenbits. Das aufgeführte Beispiel-Frame (vgl. Abbildung 2.9) wird folgendermaßen beschrieben werden: **8E1** entspricht **8** Datenbits, ein **E**ven-Paritätsbit und **1** Stoppbit.

Das Startbit ist zwingend erforderlich und wird daher nicht im Frame-Namen erwähnt. Die Anzahl der Datenbits können statt der 8 Bits auch auf 5 bis 9 Bits konfiguriert werden. Die Übertragung erfolgt nach der LSB-Methode. Das Paritätsbit ist optional und wird meist nicht eingefügt, mögliche Optionen sind hier **E**ven (Gerade), **O**dd (Ungerade) oder **N**one (Keine). Das Stoppbit, die folgende „1“ im Beispiel, ist wahlweise 1 Bit, 1,5 Bit oder 2 Bit lang. In speziellen Fällen gibt es auch weitere gebrochene Längen. [39]

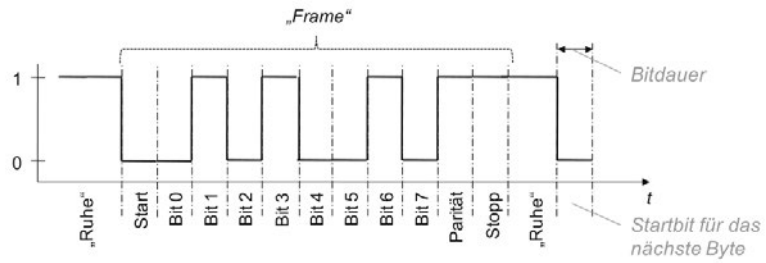

Abb. 2.9: 8E1-UART Frame [39, S. 433, Abb. 9.24]

Die Zeichenkodierung wird nach dem Empfang vom Programmierer bzw. im Voraus durch den Software-Architekten festgelegt. Längere Nachrichten werden in mehrere Frames aufgespalten. [39]

2.2.3 Baudrate und Synchronisation

Um die Daten ordentlich empfangen zu können, müssen Sender und Empfänger synchron arbeiten. Das bedeutet, dass sie die Geschwindigkeit bzw. Taktrate kennen müssen, mit der das Gegenüber die Nachricht schickt. Diese Rate nennt sich in der digitalen Übertragungstechnik Baudrate oder auch Symbolrate. Das Baud gibt dabei die Anzahl der Symbole pro Sekunde an. Sollte statt einem Symbol eine andere Menge an Daten übertragen werden, wird auch von Bit pro Sekunde gesprochen. [39]

Weiterhin muss beiden Parteien bekannt sein, wie die Nachricht selbst aussieht. Abgesehen vom Startbit kann alles der Anwendung entsprechend angepasst werden. So wird in der Konfiguration des Adapters immer angegeben, welches Frame-Layout genutzt wird. [39]

Die Synchronisierung wird mithilfe dieser Daten mit dem Start jedes Frames erneut durchgeführt und kann so Taktschwankungen ausgleichen. [39]

Sobald das Ruhesignal, eine logische 1, am Eingang des Empfängers durch den Sender auf eine logische 0 gezogen wird (vgl. Abbildung 2.9), weiß der Empfänger, dass das Frame mit der vorher definierten Länge anfängt (8E1 in diesem Beispiel). [39]

2.3 Echtzeitbetriebssystem

Im Rahmen dieser Arbeit wird auf dem MC ein Echtzeitbetriebssystem, auch Real-Time Operating System (RTOS) genannt, genutzt. Folgend werden die Eigenschaften und die Funktionsweise des RTOS dargestellt. Als Schwerpunkt dient hierbei das verwendete Software-Paket FreeRTOS.

2.3.1 Grundlegendes zu Betriebssystemen

Ein Betriebssystem ist eine Plattform, die als Schnittstelle zwischen Hard- und Software dient. Dabei regelt das OS den Datenaustausch zwischen den eingebetteten Programmen und der Hardwareperipherie. Daraus ergibt sich bei der Entwicklung der Vorteil einer schnelleren und wartungsfreundlicheren Implementierung durch das Nutzen von bereitgestellten Funktionen (vgl. Abbildung 2.10). Außerdem übernimmt es das Managen von Aufgaben in Tasks und ermöglicht die effiziente Nutzung von Ressourcen. [20]

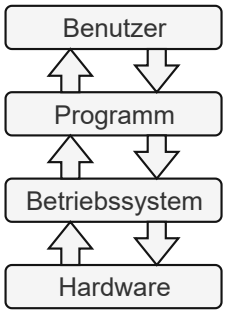

Abb. 2.10: Zusammenhang: Benutzer zu Hardware

2.3.2 Unterschiede zwischen RTOS und GPOS

Ein einfaches Betriebssystem wie z. B. Windows, ein General Purpose Operating System (GPOS), legt bei der Ausführung der Tasks mehr Wert auf Bedienbarkeit als auf eine deterministische Ausführung. So kann das GPOS bei starker Beanspruchung langsamer reagieren und nicht mehr zeitnah auf Ereignisse reagieren, während es versucht, eine möglichst flüssige Nutzererfahrung zu bieten und priorisiert damit verbundene Aufgaben höher als Hintergrundaufgaben. Ein RTOS dagegen ermöglicht die deterministische Ausführung von mehreren Aufgaben. Dies wird durch die Nutzung von Routinen, die immer die gleiche Zeitspanne zur Ausführung benötigen, umgesetzt (vgl. Unterabschnitt 2.3.3).

Die deterministische Ausführung von Tasks, auch auf bestimmte Ereignisse, ermöglicht es, zeitkritische Aufgaben auszuführen und somit echtzeitfähig zu sein. Dies wird auch

dadurch erreicht, dass der Programmierer jedem Task Prioritäten zuordnen kann und so Aufgaben mit einer Deadline zeitnah abarbeiten lassen kann. [20]

Einige der durch FreeRTOS bereitgestellten Funktionen sind besonders hervorzuheben, da sie den Ablauf mehrerer paralleler Funktionen stark verändern können. Zu diesen Funktionen zählt das Multitasking und Scheduling, die Nachrichtenwarteschlange, Semaphoren und Mutexs, Timer und Event Gruppen.

Multitasking und Scheduling

Wie eingangs erwähnt läuft eine Anwendung normalerweise auf einem Prozessor. Multitasking kann durch das Zuweisen von Programmen bzw. Programmabläufen auf mehreren Prozessorkernen erreicht werden. Das setzt aber die Nutzung eines Multicore-Systems voraus. [37]

FreeRTOS ermöglicht es, auch auf einem Singlecore-System Aufgaben parallel abarbeiten zu lassen. Dafür werden die Aufgaben in unabhängige Tasks aufgeteilt, deren zeitgerechte Ausführung der Scheduler übernimmt. [37]

FreeRTOS nutzt für das Scheduling zwei verschiedene Möglichkeiten.
Beim präemptiven Multitasking besitzt der Task eine festgelegte Zeitspanne, die er laufen darf, ansonsten wird er unterbrochen und später fortgesetzt. Alternativ kann die Ausführung auch pausiert bzw. blockiert werden. Nach dem Ablaufen der zugewiesenen Zeit oder nach einer Unterbrechung entscheidet dann der Scheduler, anhand der eingestellten Prioritäten, welcher Task als nächstes gestartet wird.
Bei dem kooperativen Multitasking (auch Co-Routinen genannt) läuft ein Task solange, bis dieser die Kontrolle von sich aus aufgibt oder blockiert wird. Zum freiwilligen Übergeben muss die Weitergabe durch eine „Yield“-Funktion gestartet werden, die innerhalb des Tasks aufgerufen wird. Dann übernimmt erneut der Scheduler und entscheidet, welcher Task als nächstes gestartet bzw. fortgesetzt wird. [25]

Nachrichtenwarteschlange

In einer Anwendung mit vielen Tasks und anderen Unterbrechungsmöglichkeiten (wie z. B. Interrupts) kann es kompliziert werden, Nachrichten zu empfangen und abzuspeichern. Wenn während des Empfangs oder des Abspeicherns eine Unterbrechung auftritt, geht der Rest der Nachricht verloren. Die „Message Queues“ von FreeRTOS stellen sicher, dass nur

vollständig empfangene Nachrichten abgespeichert und weitergegeben werden können. Sie bieten einen konfigurierbaren Nachrichtenspeicher, um sicherzugehen, dass der Speicher nicht durch unvollständige Nachrichten überläuft. Ein Task, der eine Nachricht in eine volle Warteschlange ablegen will, wird solange pausiert, bis wieder Speicher verfügbar ist. [25]

Semaphoren und Mutexe

Mit der Einführung von Semaphoren und Mutexe ist es möglich, Komponenten wie UART-Schnittstellen für einen Task zu blockieren. Damit kann kein anderer Task während einer Übertragung die Kontrolle übernehmen (Mutexe) oder nur eine bestimmte Anzahl von weiteren Zugriffen stattfinden (Semaphore). [36]

Timer

Auch wenn ein MC häufig mehrere Hardware-Timer besitzt, ermöglichen die Timer des FreeRTOS eine unterbrechungsfreie Pausierung von Tasks sowie zeitgesteuertes Starten. Mithilfe der Software-Timer von FreeRTOS ist es besonders einfach, Timer zu nutzen, ohne dass die Ausführung des Codes blockiert wird. [39]

Event Gruppen

Event Gruppen bestehen aus mehreren Event Bits (Flags), die einzelne aufgetretene Ereignisse beschreiben können. Ein solches Flag kann daraufhin bestimmte Funktionen aufrufen. Meist haben die Flags innerhalb einer Gruppe ähnliche Bedeutungen oder sind anderweitig zusammenhängend. [19]

2.3.3 Speichermanagement

Durch die Nutzung eines OS wird das Speichermanagement auf dem MC fundamental verändert. Programmiersprachen wie C und C++ bieten mehrere Methoden zur dynamischen Speicherreservierung und auch zum Freigeben des Speichers. Die Methoden malloc(), calloc() und free() ermöglichen es, während der Laufzeit des Programms, Speicher passend

zur aktuellen Situation zu reservieren und auch wieder freizugeben. Ein möglicherweise auftretendes Problem ist jedoch, dass der Speicher dabei fragmentiert und spätere Speicheranfragen damit nicht mehr arbeiten können. Zusätzlich sind diese Funktionen nicht deterministisch, was mit den Algorithmen zur Ermittlung zusammenhängender Speicherblöcke zusammenhängt. [19]

Ein Echtzeitbetriebssystem löst dieses Problem durch das Erstellen von festen Speicherbereichen für die Tasks. Wird ein solcher Task erstellt, wird der maximal definierte Speicher in Form eines zusammenhängenden Blocks reserviert und beim Beenden des Tasks wieder freigegeben. Der Task kann die maximale Menge an Speicher nicht übersteigen. So ist der Bereich in dem der Task liegt, nach der Freigabe für einen anderen Task wieder nutzbar, ohne dass der Speicher auf mehrere Bereiche aufgeteilt werden muss. Durch dieses Vorgehen ist die Reservierung und Freigabe des Speichers wieder deterministisch und für die Anwendung in einem RTOS nutzbar. [19]

FreeRTOS bietet dafür mehrere Implementierungen, die in der Konfiguration umgeschaltet und die einfach per pvPortMalloc() und pvPortFree(), anstelle von malloc() und free(), genutzt werden können. [19]

2.4 Hardware Abstraction Layer

Die Hardware Abstraction Layer (HAL) soll eine Portabilität des Programms ermöglichen. Außerdem reduziert es mögliche Wiederholungen und bündelt wichtige Hardware-bezogene Methoden in einer einfach zu nutzenden Bibliothek. [31]

Die Implementierung der HAL kann dabei in beide Richtungen geschehen, sowohl eine hardwareabhängige HAL, als auch eine software- bzw. betriebssystemabhängige HAL. Im letzteren Fall wird sie auch oft Board Support Package (BSP) genannt. [31]

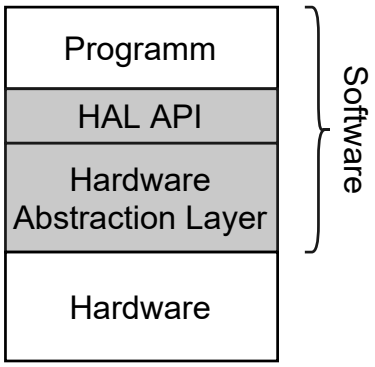

Abb. 2.11: Schichten des Hardware Abstraction Layer

2.4.1 Aufbau

Vereinfacht dargestellt umfasst die Hardwareabstraktionsschicht die Hardware-Ansteuerung durch den MC und eine Programmierschnittstelle (API) (vgl. Abbildung 2.12). Die Anwendung nutzt dabei die API, um einen Motortreiber anzusteuern, indem nur wenige Parameter übergeben werden müssen. Oder indem eine Aktion von außen durch einen Interrupt und dem zugehörigen Handler eine Nutzerfunktion ausführt. [31]

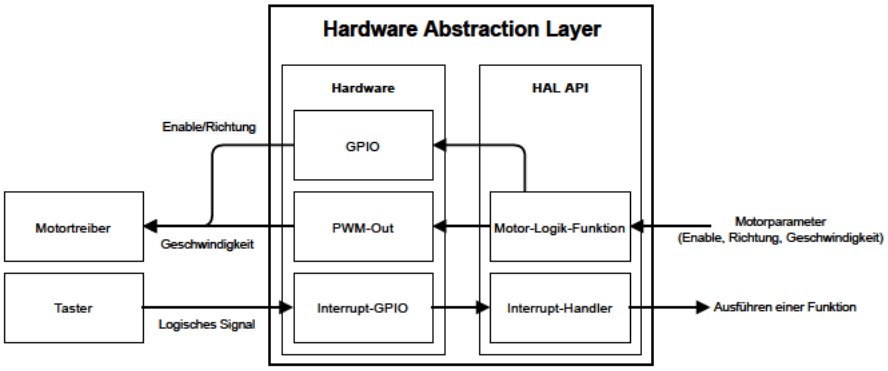

Abb. 2.12: Beispiel einer einfachen Nutzung des HAL

2.4.2 Vorteile

Die Vorteile der Nutzung einer HAL liegen in der Möglichkeit, die Software- und die Hardware-Architektur parallel zu entwickeln, wenn zuvor die Schnittstelle beider Architekturen festgelegt wurde. Sonst müsste die Hardware-Entwicklung die Schnittstelle während der Entwicklung anpassen und die Software-Entwicklung könnte erst mit dem Erstellen der Firmware, nach Beendigung der letzten Anpassung, anfangen. [31]

Für die Entwicklerebene ist vor allem das einfache Konfigurieren und die einfache Steuerung der Hardware von Vorteil. Auch kann es eine mögliche Portabilität der Software auf andere Mikrocontroller vereinfachen. Die Firma STMicroelectronics N.V. nutzt beispielsweise für viele ihrer angebotenen System-on-a-Chip (SoC) dieselbe HAL. [6]

Zusätzlich ermöglicht es eine HAL, eine umfangreichere Toolchain anzubieten. So kann STMicroelectronics N.V. neben der HAL eine All-In-One-Lösung für die Entwicklung mit ihren Mikrocontrollern anbieten, die neben einem auf Eclipse basierenden Integrated Development Environment (IDE) einen Hardware-Konfigurator mit vielen Einstellungsmöglichkeiten bietet. Über den Konfigurator ist es weiterhin möglich, unterstützte

Drittanbieter-Software in das Projekt aufzunehmen. FreeRTOS ist nur eine der angebotenen Erweiterungen (auch Middleware genannt). [31]

3 Anforderungsanalyse

Folgend wird eine Anforderungsanalyse über das vollständige System der ETCS-basierten Modelllokomotive durchgeführt. Diese Analyse umfasst sowohl Hardware als auch Software. Als grobe Vorlage zur Erstellung der Anforderungsanalyse und des Designs dient das frei verfügbare arc42 Architektur-Template [1].

Ziel der Anforderungsanalyse ist es, die Aufgaben und Voraussetzungen des zu entwickelnden Systems zu erkennen, festzuhalten und zu organisieren.

3.1 Aufgabenstellung

Die Aufgaben und Funktionen des Systems ETCS werden im Abschnitt 2.1 bereits ausführlich dargestellt. In Verbindung mit der in Abschnitt 1.2 genannten Zielsetzung, werden die folgenden Aufgaben an die Software definiert:

- Betriebsmodus angelehnt an Full Supervision
- Drahtlose Kommunikation zwischen Zug und RBC
- Auslesen von Balisen im Gleisbett zur Positionsbestimmung
- Geschwindigkeitsüberwachung durch Odometrie
- Positionsbestimmung durch Odometrie
- Fahrtenfreigabe durch MA
- Geschwindigkeitsvorgabe der Strecke durch einen Waypoint Plan (WPP)

Bezüglich der Architektur werden folgende Ziele definiert:

- Ausführliche Dokumentation der Softwareseite
- Hardwareanforderungen und -design als Rahmen für die Software mitentwickeln
- Die Softwarearchitektur soll der Umsetzung als Vorlage dienen

3.2 Qualitätsziele

Folgende Qualitätsziele, oder auch nicht funktionale Anforderungen, sollen dabei beachtet werden (Priorität in Tabelle 3.1 absteigend sortiert):

Tabelle 3.1: Qualitätsziele

Qualitätsziel	Motivation und Erläuterung
Saubere Dokumentation der Softwarearchitektur	Mit der Software als Hauptschwerpunkt wird auf eine saubere Dokumentierung der Software besonders viel Wert gelegt. Die Struktur soll dabei grob an arc42 [1] angelehnt werden. Dies ist besonders wichtig, um eine Weiterentwicklung des Prototyps zu ermöglichen.
Anforderungsanalyse: Hardware	Um eine spätere Hardwareentwicklung zu unterstützen, wird in der Anforderungsanalyse auch die Hardware projektiert. Außerdem ist es auch für die Software-Entwicklung wichtig zu wissen, wie das System außerhalb der Software genutzt werden kann.
Baugröße	Die Baugröße für die Elektronik wird durch die Baugröße einer Modelleisenbahn beschränkt, weshalb bei der Auswahl der Hardware auf deren Größe geachtet werden muss.
Erweiterbarkeit & Modularität	Es ist wichtig, für eine Erweiterbarkeit und Modularität zu sorgen, damit künftige Projekte die Software einfach ihren Bedürfnissen anpassen und erweitern können. Zudem senkt die Modularität auch die technischen Schulden, da alte Funktionen leichter mit den neuen getauscht werden können.

3.3 Randbedingungen

In diesem Kapitel werden die an das Projekt gestellten Randbedingungen in Tabelle 3.2 aufgezeigt.

Die Randbedingungen des Projektes werden in drei Kategorien unterteilt: **ORG**anisatorisch, **TECH**nisch und **KONV**entionen.

Tabelle 3.2: Randbedingungen

ID	Randbedingung	Beschreibung
ORG-1	COVID-19-Pandemie	Wenig Kundenkontakt, keine Hardwareentwicklung (Platinendruck, etc.).
ORG-2	Zeitplan	Start: 12.10.2020, Ende: 09.02.2021
ORG-3	Vorgehensmodell	Experimentelles Prototyping [35, 47]: Mangels Erfahrung mit den verwendeten Komponenten wird versucht, einzelne Programmteile und Schnittstellen miteinander zu verknüpfen, so dass ein lauffähiges Programm entsteht. Ausgehend von diesem Programm kann dann analysiert werden, welche Programmteile machbar sind und welche eine andere Herangehensweise benötigen.
ORG-4	Versionsverwaltung	Git auf Gitlab-Server, bereitgestellt durch die HAW Hamburg.
TECH-1	COVID-19-Pandemie	Kein Zugriff auf eine Teststrecke, Reduzierung der Hardware und Ersetzen durch Simulationen.
TECH-2	Mikrocontroller	Es soll ein Mikrocontroller der Firma STMicroelectronics N.V. verwendet werden. Namentlich der STM32F303K8.
TECH-3	Entwicklungswerkzeuge	Durch die Nutzung des Mikrocontrollers von STMicroelectronics N.V. soll - soweit möglich - die Entwicklungsumgebung CubeIDE von STMicroelectronics N.V. verwendet werden.
TECH-4	Baugröße	Fertiges System soll nicht sehr viel größer sein als das einer Modelleisenbahn, damit andere Modellbaukomponenten genutzt werden können.
TECH-5	Antriebssystem	Für das Antriebssystem sollen Bauteile verwendet werden, die es ermöglichen, einen Motor per PWM zu steuern.
TECH-6	H-Brücke für das Antriebssystem	Für das Antriebssystem wird die H-Brücke TLE 5206-2 von Infineon vorgegeben.

Weiter auf der nächsten Seite

ID	Randbedingung	Beschreibung
TECH-7	Implementierung in C	Die Implementierung der Software auf dem Mikrocontroller soll in C umgesetzt werden.
TECH-8	Fremdsoftware frei verfügbar	Wenn Fremdsoftware verwendet wird, dann nach Möglichkeit frei verfügbare Software, damit die Nutzung durch Lizenzen nicht erschwert wird.
TECH-9	Nutzen von Modulen	In dieser Arbeit werden externe Module verwendet, die über die Kommunikationsschnittstellen des Controllers angesteuert werden müssen.
TECH-10	Kommunikation zw. RBC und Zug	Wie in den Grundlagen zum ETCS beschrieben, findet die Kommunikation zwischen Zug und RBC drahtlos statt.
TECH-11	Kommunikationsschnittstelle	Es wird vorgegeben, als Ersatz für GSM-R eine Bluetooth-Verbindung zu nutzen. Vorgesehen ist das Modul HC-08.
TECH-12	RFID-Modul als Fahrzeugantenne	Als Vorgabe für das RFID-Modul, das die Fahrzeugantenne widerspiegeln soll, wird das Sonmicro SM130 Mifare vorgegeben.
TECH-13	Umsetzung der Balisen	Die Balisen des ETCS werden durch RFID-Tags ersetzt, die zu dem genutzten RFID-Modul kompatibel sind.
TECH-14	Spannungsversorgung	Die Spannungsversorgung erfolgt, wie für Modelleisenbahnen üblich, über das Gleis.
TECH-15	Odometrie	Die Odometrie soll über einen Sensor an der Achse verwirklicht werden.
KONV-1	Sprache (Deutsch / Englisch)	Die Dokumentation wird auf Deutsch erstellt, während der Programmcode (Funktionsnamen, Variablennamen, Kommentare, etc.) auf Englisch gehalten wird.
KONV-2	Kodierkonventionen	Die Kodierkonventionen werden in der Tabelle 4.5 im Kapitel Design und Entwurf festgelegt.

Weiter auf der nächsten Seite

ID	Randbedingung	Beschreibung
KONV-3	Diagramme in Anlehnung an UML	Die für diese Thesis erstellten Diagramme nutzen ein an Unified Modeling Language (UML) angelehntes Format. Eine vollständige Verfolgung des Standards ist durch den Umfang der UML-Spezifikationen keine Priorität.

3.4 Kontextabgrenzung

In diesem Kapitel wird das Umfeld der ETCS-Modelleisenbahn definiert und festgelegt, über welche Kanäle ein Datenaustausch stattfindet. Die Kontextansicht bietet den ersten groben Überblick über das zu erstellende System und wird im Verlauf weiter verfeinert.

3.4.1 Fachlicher Kontext

Eine erste Darstellung des Systems, in einem fachlichen Kontext, wird in Abbildung 3.1 vorgenommen. Es zeigt, welche Systeme existieren und welche Informationen diese untereinander austauschen. Die durch diese Arbeit behandelten Systeme sind dabei in Grau hervorgehoben.

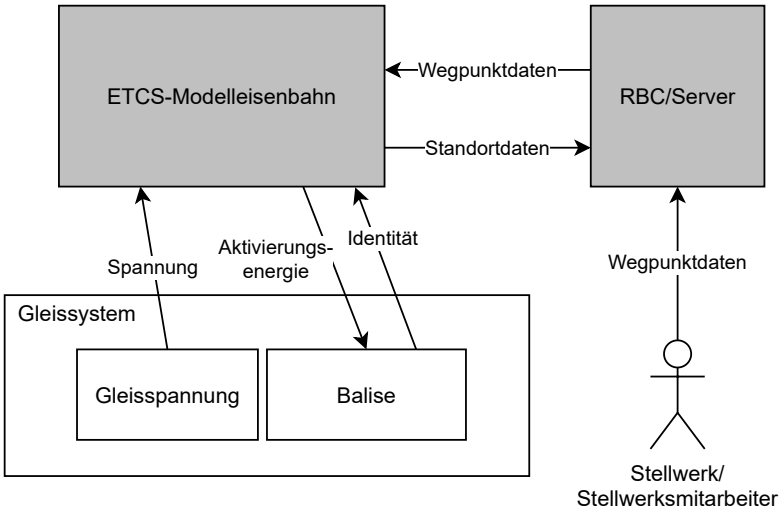

Abb. 3.1: Fachlicher Kontext

Das System der **ETCS-Modelleisenbahn** besteht aus mehreren Komponenten. Neben der Kommunikationseinheit für die Verbindung zum RBC sind auch die Spannungsversorgung, das System zur Interaktion mit den Balisen, die Odometrie, das Antriebssystem und eine Logikeinheit darin zusammengefasst.

Die **RBC-Server-Application** ist die Gegenstelle der ETCS-Modelleisenbahn und soll in der ersten Implementierung nur die wichtigsten Funktionen enthalten.

Das **Gleissystem** besteht, neben den Gleisen, aus den für die ETCS-Modelleisenbahn wichtigen Komponenten Gleisspannung sowie den Balisen. Die vom Gleis abgenommene Spannung wird für das gesamte Modelleisenbahnsystem verwendet (siehe TECH-14, Tabelle 3.2). Die Balisen sind essenzielle Komponenten des ETCS und werden im Gleisbett angebracht. Dort werden sie beim Überfahren durch die Fahrzeugantenne mit Energie versorgt und übermitteln ihre einzigartige Seriennummer.

3.4.2 Technischer Kontext

Der technische Kontext stellt das System nun in einer komplexeren Ansicht dar. Durch die in den Randbedingungen definierten Komponenten und allgemeinen Bedingungen kann hier bereits ein sehr viel detaillierterer Kontext dargestellt werden (vgl. Abbildung 3.2).

Erneut sind die durch diese Thesis behandelten Komponenten in Grau hervorgehoben:

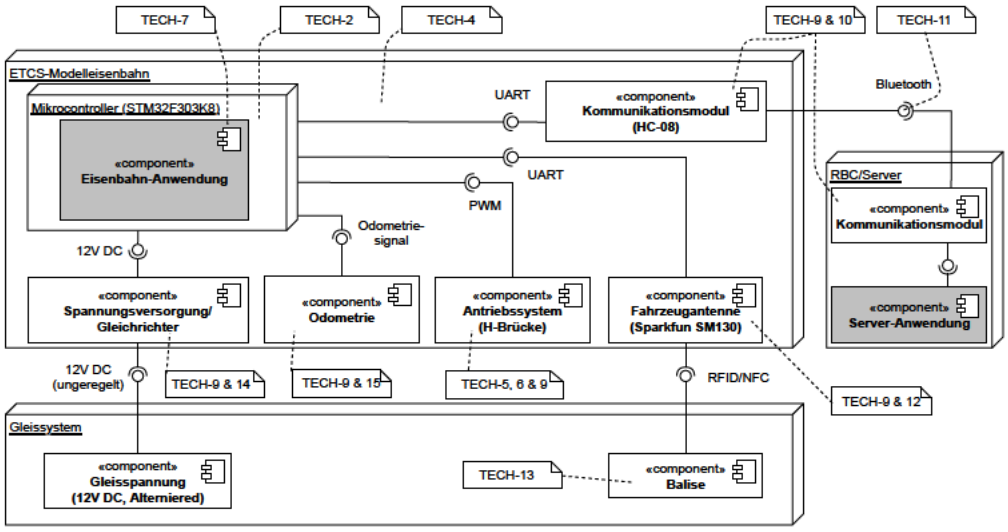

Abb. 3.2: Technischer Kontext

In der Tabelle 3.3 werden die in Abbildung 3.1 abgebildeten Schnittstellen mit denen aus Abbildung 3.2 verbunden.

Tabelle 3.3: Beschreibung der Schnittstellen

Fachliche Schnittstelle	Technische Schnittstelle am MC
Gleisspannung	Versorgungsspannung 12V DC (alternierend)
Wegpunktdaten	UART (über Bluetooth)
Standortdaten	UART (über Bluetooth)
Identität	UART (über RFID)
Aktivierungsenergie	RFID

Außerdem wird die Modelleisenbahn in den einzelnen Systemen untersucht und mit den technischen Randbedingungen verknüpft.

Der **Mikrocontroller** bildet das Herzstück der ETCS-Modelleisenbahn. Er übernimmt die Funktion des EVC. Der MC nutzt dabei die vorhandenen UART-Schnittstellen, um mit dem Kommunikationsmodul sowie der Fahrzeugantenne zu sprechen. Außerdem steuert er das Antriebssystem per PWM an.

Zur Kommunikation mit dem Server (RBC) wird durch TECH-11 das Bluetooth-Modul HC-08 von Wavesen als **Kommunikationsmodul** vorgeschrieben. Das Modul nutzt für die Bluetooth-Verbindung die Spezifikation 4.2 inklusive Low-Energy Funktionen. Es wird vom MC über UART angesprochen und überträgt die so empfangenen Frames über eine bestehende Bluetooth-Verbindung an den Server und empfängt dessen Frames und leitet sie per UART an den MC weiter. Für die Konfiguration des Moduls werden sogenannte AT-Befehle [46] genutzt.

Die **Fahrzeugantenne**, bestehend aus dem durch TECH-12 vorgegebenen Modul SM-130 von Sonmicro, imitiert zusammen mit einer passenden Antenne die ETCS-Fahrzeugantenne. Das RFID-Modul nutzt für das elektromagnetische Feld 13,56 MHz. Dieses Feld dient dazu, RFID-Tags im Gleisbett zu aktivieren und auszulesen. Das Modul wird vom MC über einen UART-Befehlssatz angesprochen und antwortet mit vordefinierten Antworten [42].

Die Funktion der **Balise** übernehmen RFID-Tags, die passend zu dem verwendeten RFID-Modul ausgelegt werden.

Das **Antriebssystem** soll gemäß TECH-5 aus einem Motor bestehen, der durch eine Halbbrücke angesteuert wird. Zum Steuern der Halbbrücke wird ein PWM-Signal benötigt.

Die **Odometrie** soll dem System, bzw. dem MC, ermöglichen, kontinuierlich die Geschwindigkeit und die zurückgelegte Strecke zu messen. Hierfür soll ein Sensor verwendet werden, der pro Umdrehung ein digitales Signal an den MC sendet.

Die **Spannungsversorgung** muss die kontinuierliche Versorgungsspannung des Mikrocontrollers sicherstellen. Die Spannung soll 12 V betragen, die von den Gleisen geliefert wird und auch vom MC genutzt werden kann. Da die Spannung an Weichenüberfahrten alternieren und kurzzeitig unterbrochen werden könnte, muss die Spannungsversorgung diese Spannung gleichrichten, puffern und glätten.

Der **RBC-Server** soll in diesem System die Funktionen des RBC übernehmen. Hierfür muss er sich mittels Bluetooth mit der Modelleisenbahn verbinden und über UART mit dem MC kommunizieren. Seine Aufgaben bestehen darin, dem Zug die Streckendaten (Waypoint Plan) und die Fahrterlaubnis zukommen zu lassen sowie die Positionsdaten und die Geschwindigkeit des Zuges entgegenzunehmen.

3.5 Use-Cases

In diesem Abschnitt werden die Anwendungsfälle (Use-Cases) der ETCS-Modelleisenbahn grafisch dargestellt und anschließend erklärt. In Abbildung 3.3 ist das System der ETCS-Modelleisenbahn in Use-Cases dargestellt und aufgeteilt.

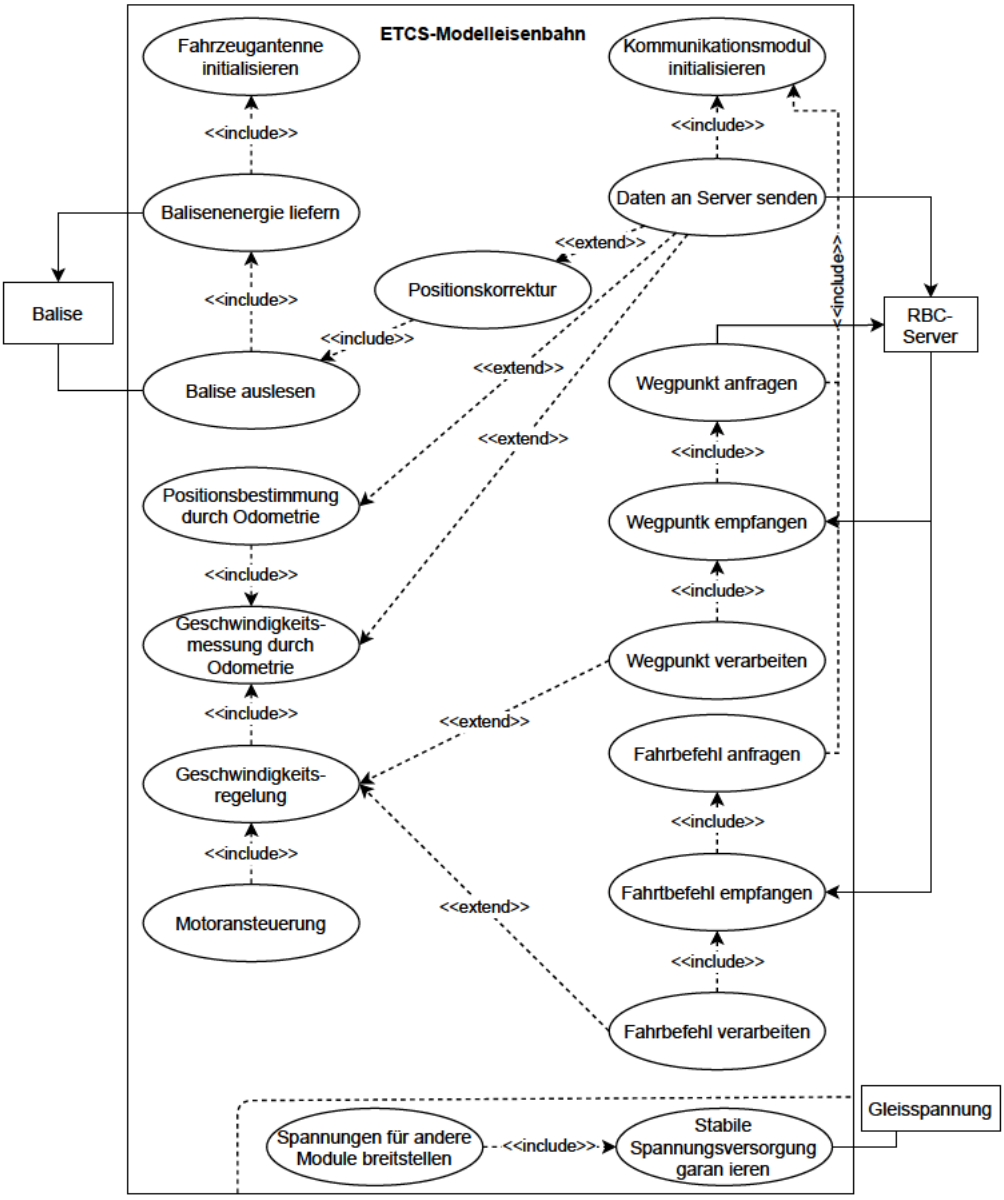

Abb. 3.3: Use-Case Diagramm

Das System der Spannungsversorgung ist in Abbildung 3.3 abgegrenzt von den restlichen Anwendungsfällen gezeichnet, da es sich hier um ein reines Hardware-System handelt (siehe Unterabschnitt 4.3.1).

Fahrzeugantenne initialisieren Dieser Anwendungsfall stellt sicher, dass die Fahrzeugantenne erfolgreich aktiviert wird. Mögliche Aufgaben sind unter anderem das Konfigurieren und Starten des Moduls.

Balisenenergie liefern Die Balisen bzw. RFID-Tags sind passive elektronische Komponenten. Als Grundvoraussetzung benötigen sie eine von außen induzierte Energie, um funktionieren zu können. Es muss sichergestellt werden, dass das der Fall ist.

Balise auslesen Die Fahrzeugantenne muss den Inhalt der Balise lesen können.

Positionskorrektur Die typische Aufgabe in dieser Implementierung des ETCS ist es, die Balisen-ID zu nutzen, um die ungenaue Position des Zuges (berechnet durch die Odometrie) mit der bekannten Position der Balise zu verbessern.

Kommunikationsmodul initialisieren Analog zur Fahrzeugantenne muss das System sichergehen, dass das Kommunikationsmodul richtig konfiguriert und gestartet ist.

Daten an Server senden Laut ETCS-Spezifikation muss der Server fortlaufend über Position und Geschwindigkeit des Zuges informiert werden. Zusätzlich kann der Zug noch weitere Entwicklerdaten zum Server senden.

Wegpunkt anfragen Soweit noch kein Wegpunkt im System gespeichert ist und das Kommunikationsmodul initialisiert ist, muss die Modelleisenbahn nach Wegpunkten fragen.

Wegpunkt empfangen Sobald der RBC einen oder mehrere Wegpunkte schickt, müssen diese von der Modelleisenbahn empfangen werden können.

Wegpunkt verarbeiten Sobald alle Wegpunkte empfangen wurden, müssen diese von der Modelleisenbahn entsprechend verarbeitet (gespeichert) werden.

Fahrbefehl anfragen Soweit noch kein Fahrbefehl im System gespeichert ist und das Kommunikationsmodul initialisiert ist, muss die Modelleisenbahn nach Fahrbefehlen fragen.

Fahrbefehl empfangen Sobald der RBC einen oder mehrere Fahrbefehle schickt, müssen diese von der Modelleisenbahn empfangen werden können.

Fahrbefehl verarbeiten Sobald alle Fahrbefehle empfangen wurden, müssen diese von der Modelleisenbahn entsprechend verarbeitet und gespeichert werden.

Geschwindigkeitsmessung durch Odometrie Die Modelleisenbahn muss in der Lage sein, fortlaufend durch die Nutzung der Odometrie die Geschwindigkeit zu bestimmen.

Positionsbestimmung durch Odometrie Um kontinuierlich die Position aktuell halten und mit der Wegpunktliste abgleichen zu können, muss die Modelleisenbahn fortlaufend durch die Nutzung der Odometrie die Fortbewegung nachverfolgen können.

Geschwindigkeitsregelung Um die Geschwindigkeit der Modelleisenbahn an die vorgegebene Geschwindigkeit der Strecke anpassen zu können, müssen die aktuelle Ist-Geschwindigkeit und die Soll-Geschwindigkeit genutzt werden, um ein Ausgangssignal zu erzeugen.

Motorsteuerung Das Ausgangssignal der Geschwindigkeitsvorgabe muss genutzt werden, um den Motor zu regeln.

Stabile Spannungsversorgung garantieren Die Grundfunktion der Spannungsversorgung ist die Bereitstellung einer stabilen 12 V-Versorgungsspannung. Die Spannung an den Rädern der Modelleisenbahn kann beim Überfahren von Weichen die Polarität wechseln. Eine Umpolung, ein möglicher Einbruch oder Kurzschluss der Spannungen über die Räder muss verhindert werden.

Spannungen für andere Module bereitstellen Um nicht auf die durch den Mikrocontroller bereitgestellten Spannungen angewiesen sein zu müssen, könnte die Spannungsversorgung die stabilisierte 12 V-Spannung zur Erzeugung weiterer Spannungen für die anderen Module genutzt werden.

3.6 Ergebnis der Analyse

Mithilfe der Use-Cases und dem technischen Kontext aus Unterabschnitt 3.4.2 wird nun eine Tabelle der funktionalen Anforderungen erstellt und durch einige optionale Anforderungen ergänzt. Die nicht funktionalen Anforderungen werden bereits im Abschnitt 3.3 aufgestellt.
Die Tabelle mit Anforderungen befindet sich im Anhang in Forum von Tabelle A.1.

4 Design und Entwurf

In diesem Kapitel wird mithilfe der aus Kapitel 3 gewonnenen Anforderungen und Randbedingungen über ein grundlegendes Design und die zu verwendenden Hardware- und Softwarekomponenten entschieden. Dabei wird die durch Tabelle 3.2 vorgegebene Hardware durch einen Vergleich mit alternativen Komponenten bestätigt. Außerdem wird die weitere genutzte Hard- und Software aufgezählt soweit sie erwähnenswert ist.

Um den in Tabelle 3.1 aufgestellten Qualitätszielen gerecht zu werden, wird die Hardware-Architektur bis zu einem grundlegenden Design ebenso detailliert fortgeführt wie die darauffolgende Software-Architektur. Dabei wird auf die Entscheidung von spezifischen Bauteilen verzichtet, außer wenn diese durch die Randbedingungen vorgegeben werden. Andernfalls wird die genutzte Schnittstelle entsprechend festgelegt.

In der folgenden Tabelle 4.1 werden die Systeme des technischen Kontexts mit den sie beschränkenden Randbedingungen aus der Tabelle 3.2 und an sie gestellten Anforderungen aus Tabelle A.1 verknüpft.

Tabelle 4.1: Verknüpfung der Systeme mit Randbedingungen und Anforderungen

System	Randbedingungen	Anforderungen
Spannungsversorgung	TECH-4, 9, 14	REQ-S1 – S5
Odometrie	TECH-4, 9, 15	REQ-O1
Antriebssystem	TECH-4, 5, 6, 9	REQ-A1 – A2
Fahrzeugantenne	TECH-4, 9, 12, 13	REQ-F1 – F2
Kommunikationsmodul	TECH-4, 9, 10, 11	REQ-K1 – K6
Mikrocontroller	TECH-2, 4, 7, 9 KONV-1, 2	REQ-M1 – M6
Server-Anwendung	ORG-3, 4 TECH-1, 3, 8, 10, 11 KONV-1, 2	REQ-R1 – R7
Eisenbahn-Anwendung	ORG-3, 4 TECH-2, 3, 7, 8 KONV-1, 2	REQ-E1 – E15

Die erste Herausforderung an das Design besteht darin, es auf ein vorhandenes System aufzubauen. Durch das System, die Modelleisenbahn, ist die Baugröße ein stark limitierender Faktor (TECH-4). Die Maße einer üblichen H0-Spur-Lokomotive sind beispielsweise 22 cm in der Länge und 4 cm in der Breite (von Modell zu Modell unterschiedlich).

Das Aussehen ist für einen Hardwareprototyp nicht entscheidend, weshalb auf die Nutzung fertiger Module zurückgegriffen wird.

Bezüglich der Randbedingung Baugröße TECH-4 werden die Komponenten aus bereits bekannten Modulen ausgewählt. Die Baugrößen der Komponenten sind wie folgt:

- Mikrocontroller: 51 mm x 19 mm [44]
- Bluetooth Modul: 27 mm x 13 mm [46]
- RFID-Modul: 38 mm x 31 mm [42]
- Antenne: 25 mm x 25 mm [38]
- H-Brücke: 10 mm x 17 mm (Package TO-263) [26]

4.1 Hardwareentscheidungen

In diesem Kapitel wird erklärt, worauf die Entscheidungsfindung zur Nutzung einer Komponente basiert. Bei manchen Komponenten geschieht dies über die Definition von Bewertungskriterien, bei anderen wird das System im Ganzen betrachtet und die möglichen Lösungen gegeneinander abgewogen. Auch wenn durch die Randbedingungen ORG-1 und TECH-1 die Entwicklung eines Hardware-Prototyps entfällt, müssen die nutzbaren Komponenten definiert werden, um deren Schnittstellen festzulegen. An diese Entscheidungen wird später die Software angepasst.

Bei einigen Komponenten wird ebenfalls eine Analyse der Alternativen durchgeführt.

4.1.1 Spannungsversorgung

Eine typische H0-Spur-Modelleisenbahn bezieht ihre Spannung über die Gleise, welche durch eine zentrale Spannungsquelle gespeist werden.
Das Gleis als Spannungsquelle ist dementsprechend die naheliegende Lösung.

Alternativ könnte in der Bahn ein Akku verbaut werden. Dies würde allerdings die verfügbare Baugröße stark beeinflussen. Ein typischer Lithium-Ionen-Akku wie der Typ 18650 besitzt einen Durchmesser von 18 mm und eine Länge von 65 mm. Damit würde er ca. ein Drittel der verfügbaren Länge einer typischen Lokomotive einnehmen (22 cm, vgl. Maße in Tabelle 4).

Die Herausforderung bei der Nutzung der Gleisspannung ist, dass es baubedingt zu Unterbrechungen und Umpolungen der Spannung kommen kann. Diesen Aspekten kann aber durch Gleichrichten und Pufferung entgegengewirkt werden.

4.1.2 Odometrie

Durch die Anlehnung an das reale ETCS wird hier die gängige Methode eines Achsensensors genutzt. Die zu prüfenden Sensoren müssen dafür an der Achse der Modelleisenbahn Platz finden. Einfachheitshalber werden Sensoren geprüft, die über einen digitalen Ausgang verfügen und je nach Situation zwischen `High` und `Low` am Ausgang wechseln.

Die folgenden Kriterien werden bei der Wahl des Sensors berücksichtigt:

- Preis
  Ein niedriger Preis ist zu bevorzugen, aber nicht ausschlaggebend.
- Bauform
  Eine kleine Bauform des Sensors sowie des Betätigers ist wichtig, um die Randbedingung TECH-4 zu erfüllen.
- Wartungsintensität
  Der Sensor soll wenig Wartung (säubern, ölen, etc.) benötigen.
- Genauigkeit
  Der Sensor soll jede Umdrehung erfassen, starke Abweichungen sind zu vermeiden.
- Zuverlässigkeit
  Der Sensor soll eine lange Lebensdauer besitzen und auf mechanische Bauteile verzichten.
- Dokumentation
  Soweit erforderlich soll eine gute Dokumentation des Sensors bereitstehen.

Diese Kriterien werden nun in der Tabelle 4.2 in einer Pugh-Matrix gesammelt und gewichtet bewertet. [32]

Die Auswahl der Sensoren erfolgt unter der Voraussetzung, dass sie durch einen an der Achse angebrachten Betätiger geschaltet werden können. Es werden folgende Sensoren überprüft:

- Hall-Sensor: INFINEON TLE 4905L
- Schnappschalter: MARQUARDT 01006.0901-01
- Reed-Sensor: MEDER MK 1466A
- Reflektionstaster: VISHAY CNY 70

Die Spezifikationen der Sensoren werden dabei aus der jeweiligen Produktseite und den dort hinterlegten Datenblättern des Reichelt Online-Shops entnommen. [3, 7, 8, 17]

In der Tabelle 4.2 werden die Sensoren nun gegeneinander verglichen, wobei der Hall-Sensor als Vorgabe dient. Wenn eine Bewertung besser als die Vorgabe ausfällt, wird der Vergleich durch ein (+) und eine schlechtere durch ein (–) gekennzeichnet. Eine neutrale Bewertung wird duch eine (0) markiert. Anschließend werden die so gewonnenen Punkte anhand der Gewichtung summiert.
Die Bewertung erfolgt dabei durch subjektive Betrachtung der Spezifikationen und durch die bereits gesammelten, subjektiven Erfahrungen mit solchen Sensoren.

Im Vergleich zu dem Hall-Sensor konnten die anderen Sensoren keine Vorteile bieten, weshalb dieser für diese Arbeit genutzt wird.

Tabelle 4.2: Pugh-Matrix: Odometrie

Odometrie-Sensor / **Kriterien**	**Hall-Sensor**	**Taster**	**Reed-Sensor**	**Reflektionstaster**	**Gewichtung**
Bauform	0	-	0	-	5
Wartungsintensität	0	-	0	-	4
Genauigkeit	0	0	0	0	4
Zuverlässigkeit	0	-	-	0	4
Dokumentation	0	-	0	0	2
Preis	0	-	-	0	1
Summe +	0	0	0	0	
Summe -	0	5	2	2	
Summe 0	6	1	4	4	
Gewichtete Summe +	0	0	0	0	
Gewichtete Summe -	0	16	5	9	

4.1.3 Antriebssystem

Zum Ansteuern eines Motors mit einem Mikrocontroller bedarf es externer Elektronik bzw. eines Motortreibers, da ein MC in der Regel nicht mit Leistungselektronik ausgestattet ist.

Für die Nutzung von Gleichstrommotoren stehen zwei Möglichkeiten zur Auswahl: Der bürstenbehaftete Gleichstrommotor und der bürstenlose Gleichstrommotor. Der Bürstenlose wird mithilfe von drei Halbbrücken und einem PWM-Signal gesteuert. Der bürstenbehaftete Gleichstrommotor kann mit einer einzelnen Halbbrücke und einem PWM-Signal angesteuert werden.

Die Verwendung eines Wechselstrommotors ist auf dem Zielsystem Modelleisenbahn schlecht umsetzbar, da das Gleissystem nur eine 12 V Gleichspannung an die Modelleisenbahn liefert. Die Leistungselektronik für die Generierung der Wechselspannung benötigt wesentlich mehr Platz als eine H-Brücke.

Die Auswahl des genutzten Antriebssystems wird dementsprechend durch TECH-4 beschränkt und bevorzugt die Lösung mit dem geringsten Platzanspruch. Das ist der bürstenbehaftete Motor mit einer einzelnen Halbbrücke. Da das System der ETCS-Modelleisenbahn einen Fahrzeugcomputer vorsieht, welcher in diesem Projekt durch einen Mikrocontroller realisiert wird, kann bei der Auslegung darauf geachtet werden, dass dieser mindestens ein PWM-Signal erzeugen kann.

Die Verwendung eines PWM-Signals wird durch die Randbedingung TECH-5 vorgegeben und ist somit in die Anforderungen eingeflossen.

4.1.4 Fahrzeugantenne

Die Fahrzeugantenne wird durch die Randbedingung TECH-12 bereits festgelegt, dennoch stehen weitere Methoden zur Auswahl.

Das Realsystem (siehe Abschnitt 2.1) nutzt für diese Komponente ein System aus Empfänger und Sender. Dabei ist der Sender (die Balise) ein passives Bauteil, das erst durch den Sender (Fahrzeugantenne) mit Energie versorgt wird. Diese Technik nennt sich RFID und kann auch in kleineren Bauformen genutzt werden.

Das durch die Randbedingung TECH-11 definierte Modul ermöglicht bereits die geforderten Funktionen, dennoch wird es mit anderen möglichen Lösungen verglichen. Mithilfe des durch Randbedingung TECH-11 definierten Moduls ist eine, dem realen System entsprechende, Technik ausgewählt worden. Dennoch wird diese Lösung mit anderen möglichen Lösungen verglichen.

Eine ETCS angelehnte Methode benötigt den Informationsfluss aus dem Gleis in den Zug, mit dem der Zug lokalisiert werden kann. Dieses Vorgehen schließt Systeme wie Triangulation über Funkmasten oder die Nutzung von Bluetooth-Beacons aus, da deren Reichweite zu groß ist und bei der Triangulation auch der Informationsfluss über den RBC-Server stattfinden müsste.

Es sind die folgenden Methoden im Gleisbett denkbar:

- RFID-Tag
  Im Gleisbett liegt ein RFID-Tag, der beim Überfahren mit Energie versorgt wird und damit ein Antwortsignal an den Sensor überträgt.
- Barcode
  Es wird ein Barcode im Gleisbett hinterlegt, der beim Überfahren vom Zug-Sensor erfasst wird.
- Aktiver Sender
  Im Gleisbett liegt eine Antenne, die aktiv ihr Signal versendet. Das Signal wird beim Überfahren vom Zug-Sensor empfangen.

Folgende Kriterien werden für die Auswahl der Technik berücksichtigt:

- Preis des Sensor
  Ein niedriger Preis ist zu bevorzugen, ist aber nicht ausschlaggebend.
- Preis vom Gegenstück
  Das im Gleisbett verbaute Gegenstück wird potentiell in wesentlich höheren Stückzahlen benötigt und kann daher nicht vernachlässigt werden.
- Bauform des Zugsensors
  Eine kleine Bauform des Systems ist wichtig, um die Randbedingung TECH-4 zu erfüllen - sowohl als Bauteil am Zug als auch im Gleisbett.
- Wartungsintensität
  Das System soll wartungsarm (säubern, ölen, etc.) sein.
- Genauigkeit
  Das System soll die Position mit geringer Abweichung übertragen.
- Zuverlässigkeit
  Das System soll eine lange Lebensdauer besitzen und auf mechanische Bauteile verzichten.
- Dokumentation
  Soweit erforderlich soll eine gute Dokumentation des Systems bereitstehen.

Erneut werden diese Punkte in eine Pugh-Matrix eingetragen, um eine mögliche Alternative zur RFID-Technik erkennen zu können. Die Spezifikationen der Komponenten werden aus den in den Quellen aufgeführten Online-Shops zusammengesucht.

Für die RFID-Methode wird das bereits definierte Modul SM130 genutzt. Für das Gegenstück wird ein kompatibles Produkt gesucht, um die Kosten abschätzen zu können

[9, 15]. Bei einer späteren Hardware-Entwicklung muss auf Kompatibilität des RFID-Moduls und der Tags geachtet werden sowie auf die Verfügbarkeit. Als Antenne wird die W7001 von Pulse [13] ausgewählt, die auch in den Ergebnissen des Projekts „RFID Positioning Balise In Miniature ETCS System“ [34] genutzt wird.

Für die Methode des Barcodes wird ein Komplett-Modul herausgesucht, um einen Vergleich durchzuführen. Die Möglichkeit der Nutzung mit einem Mikrocontroller ist das einzige Auswahlkriterium [5]. Über die Genauigkeit und Schnelligkeit des Sensors werden, abgesehen von der verwendeten Technik, keine Bewertungen durchgeführt, da sie nicht für einen Test vorliegen. Besonders positiv an dieser Technik ist, dass die Gegenstücke im Gleis mit einem einfachen Drucker erzeugt werden können.

Bei der Gleisantenne wird keine Rücksicht darauf genommen, dass das Gegenstück zum Zug-Sensor ein aktives Bauelement ist und somit im Grunde bereits ausscheidet, da es kaum noch der ETCS-Methode folgt [4]. Die Methode kann mit Mehraufwand umgesetzt werden. Eine mögliche Ungenauigkeit durch erhöhte Reichweite wird unter dem Kriterium Genauigkeit berücksichtigt.

Durch das Aufstellen in der Pugh-Matrix in Tabelle 4.3 wird ersichtlich, dass die RFID-Technik am ehesten den Anforderungen entspricht.

Tabelle 4.3: Pugh-Matrix: Fahrzeugantenne

Fahrzeugantenne / Kriterien	RFID	Barcode	Gleis-Antenne	Gewichtung
Bauform des Zug-Sensor	0	0	+	5
Genauigkeit	0	-	-	5
Zuverlässigkeit	0	-	+	4
Preis Gegenstück	0	+	-	4
Wartungsintensität	0	-	0	4
Größe Gegenstück	0	0	-	3
Dokumentation	0	-	0	2
Preis Zug-Sensor	0	-	+	1
Summe +	0	1	3	
Summe -	0	5	3	
Summe 0	8	2	2	
Gewichtete Summe +	0	4	10	
Gewichtete Summe -	0	16	12	

4.1.5 Kommunikationsmodul

Für die Kommunikation stehen verschiedene Möglichkeiten zur Wahl. Durch die Randbedingung TECH-9 und dem realen ETCS wird bereits eine drahtlose Kommunikation vorgesehen. Eine drahtgebundene Methode über das Gleis wäre möglich (Stichwort Frequenzmodulation), wird aber zugusten der Vergleichbarkeit zum Realsystem ausgeschlossen. Stattdessen soll nach TECH-10 Bluetooth verwendet werden.

Neben Bluetooth gibt es noch weitere Möglichkeiten, eine Verbindung aufzubauen. Um zwei Beispiele zu nennen: WLAN und LoRaWAN (Long Range Wide Area Network). Diese ist auch für die Modelleisenbahn vorstellbar und besonders in späteren Versionen mit

mehreren Teilnehmern besser geeignet, aber hinsichtlich der einfachen Implementierung wird die Nutzung von Bluetooth festgelegt. Durch die Nutzung des in TECH-10 definierten HC-08 Moduls lässt sich eine feste Verbindung zwischen Modelleisenbahn und Server einrichten. So bedarf es keiner Priorisierung zwischen Teilnehmern und keiner Beachtung von Störeinwirkungen durch andere Teilnehmer.

4.1.6 Mikrocontroller

Die an die Mikrocontroller gestellten Anforderungen sind durch REQ-M1–M6 definiert. Für die Ausführung paralleler Aufgaben (REQ-E15) muss der MC die hierfür benötigten Eigenschaften besitzen. Für das Verarbeiten paralleler Aufgaben ist die Verwendung von FreeRTOS vorgesehen, das einen eigenen Timer benötigt. Zusätzlich wird für die Zeitmessung ein Timer benötigt.

Dadurch ergeben sich die benötigten Eigenschaften wie folgt:

- mind. 1 Digital-Input
- mind. 2x UART
- mind. 1x PWM-Output
- 12 V DC-Input
- Möglichkeit, Start zu verzögern (Reset-Pin)
- Möglichkeit, FreeRTOS zu implementieren
- mind. 3 Timer (1x FreeRTOS, 1x PWM, 1x Timer)

Alternativen, zu den in TECH-2 genannten MC, sind die STM32F303K8, STM32F103C8T6 und der Arduino Nano.

In der folgenden Tabelle 4.4 sind die Spezifikationen der einzelnen MC bzw. Entwicklerboards aufgelistet und werden miteinander verglichen. Die Punkte, die mit den Anforderungen übereinstimmen, werden hervorgehoben.

Tabelle 4.4: Spezifikationsvergleich Entwicklerboards

	NUCLEO-F303K8 [10]	„Bluepill“ [16]	Arduino Nano [12]
CPU	STM32F303K8	STM32F103C8T6	ATmega328
Preis (€)	9,11	4,85	19,95
Größe (mm)	51 x 19	54 x 23	45 x 18
Pin Anzahl (GPIO)	32	34	22
Kerne@Frequenz (MHz)	1@72	1@72	1@16
Flash (KB)	64	64	32
EEPROM (KB)	–	–	1
RAM (Bytes)	12	20	2
Timers	11	7	3
Versorgungsspannung (V)	3,3/5/7-12	2-3,6	5/7-12
Interrupt Pins	16	Alle	2
UART-Channel	3	3	1
PWM-Out	4	4	6
Reset-Pin	Verfügbar	Verfügbar	Verfügbar
FreeRTOS	Verfügbar	Verfügbar	Verfügbar

Aus der Tabelle 4.4 wird ersichtlich, dass zwei der drei Entwicklerboards die benötigten Schnittstellen anbieten. Zusätzlich ist es mit dem Nucleo-Board möglich, die 12 V der Spannungsversorgung zu nutzen, ohne diese reduzieren zu müssen. Über die benötigte Taktfrequenz und den Speicherbedarf des fertigen Programms kann hier noch keine Aussage getroffen werden.

4.2 Hardware-Architektur

Durch die Nutzung der Hardware, die durch das vorangehende Thema definiert wird, lässt sich das in Abbildung 4.1 dargestellte Blockdiagramm festlegen. Spätere Projekte können hiermit auf die konzeptionierte Hardware aufbauen.

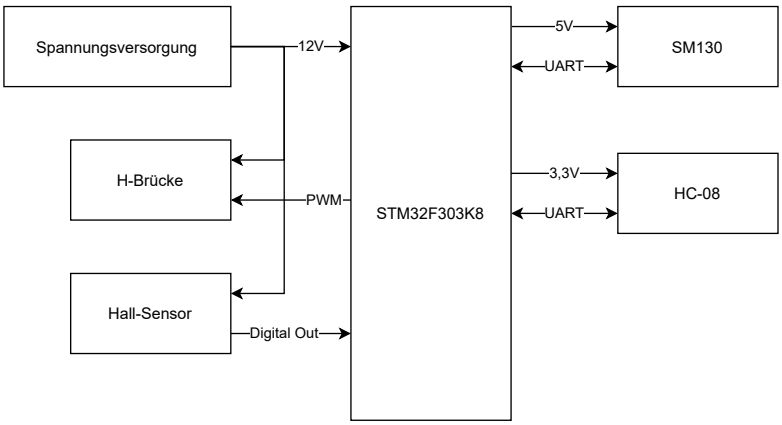

Abb. 4.1: Blockdiagramm der Hardware-Komponenten

4.3 Softwareentscheidungen

Im folgenden Teil der Arbeit werden die Entscheidungen über die Strukturierung der Software sowie die genutzten Tools aufgezeigt.

4.3.1 Reduzierung auf eine Simulation

Wie bereits durch die Randbedingungen ORG-1 und TECH-1 vorgegeben, wird die Hardwareentwicklung innerhalb der vorliegenden Arbeit nicht weiter vorangetrieben. Stattdessen wird aus dem technischen Kontext (Unterabschnitt 3.4.2) eine reduzierte Variante erstellt. In Abbildung 4.2 wird dargestellt, auf welche Komponenten sich die Arbeit durch den Wegfall der Hardware fokussieren wird.

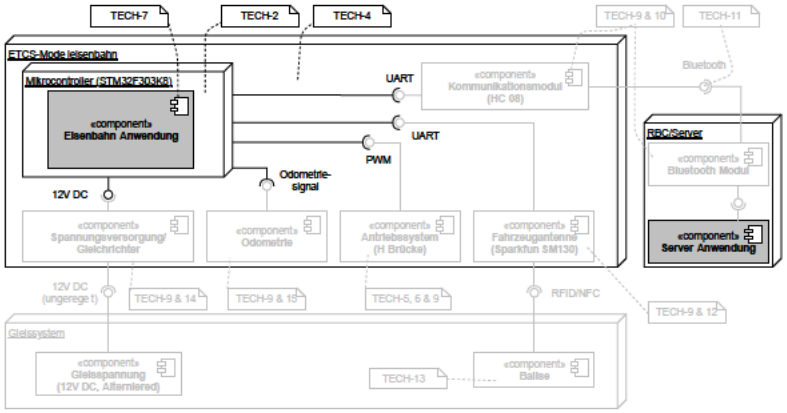

Abb. 4.2: Auf Simulation reduzierter Kontext

Damit die REQ-E1 und REQ-E2 prüfbar sind, werden das Kommunikationsmodul (HC-08) und das RFID-Modul (SM130) innerhalb der Server-Anwendung simuliert. Die Kommunikationsschnittstelle (UART) bleibt für beide Module erhalten. Die Antworten werden nicht vom Modul, sondern vom Server erzeugt. Der Eingang für das Odometrie-Signal und der PWM-Ausgang bleiben unbesetzt. Die Spannungsversorgung wird durch ein 12 V-Netzteil ermöglicht. Der im Folgenden umgesetzte Kontext für die Simulation ist in Abbildung 4.3 ersichtlich.

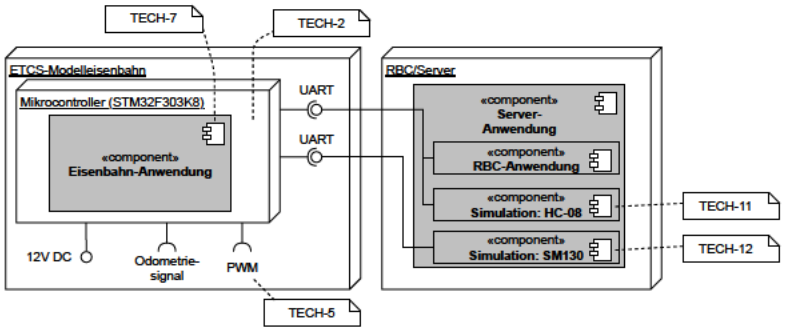

Abb. 4.3: Kontext der Simulation

Um diese Simulation auf dem MC laufen zu lassen, bedarf es zweier UART-Schnittstellen. In Abbildung 4.4 ist dargestellt, wie der MC mit dem Programmiergerät verbunden werden kann.

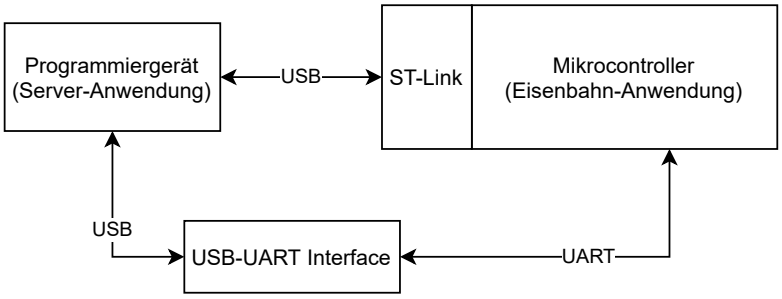

Abb. 4.4: Anschlussdiagramm

4.3.2 Nutzen eines RTOS

Früh in der Entwicklungsphase ist die Entscheidung getroffen worden, ein Real-Time Operating System einzusetzen. Dies vereinfacht das Handling mehrerer Aufgaben auf einem MC mit nur einem physischen Prozessorkern. Da das STM32CubeMX-Konfigurationstool die Option bietet, FreeRTOS in das Projekt zu implementieren, wird dieses als RTOS genutzt.

Neue Task werden, wie in Abbildung 4.5 gezeigt, über einen Assistenten hinzugefügt und parametriert. Der Konfigurator erstellt in der Quellcode-Datei einen neuen Eintrag. In diesen Eintrag wird der für den Task entsprechende Code geschrieben. Wie in den Grundlagen (Abschnitt 2.3) erklärt, übernimmt der FreeRTOS-Kernel das Handling der Tasks.

Abb. 4.5: Konfigurationsassistent für Tasks in CubeMX

4.3.3 Nutzen eines Circular Buffers

Der Umfang der Daten, die in diesem Projekt empfangen und versendet werden, benötigt eine Methode, um die empfangenen Telegramme zwischenzuspeichern und später byteweise zu verarbeiten. Für diesen Zweck wird von Prof. Dr. Buczek die Implementierung eines Circular Buffers zur Verfügung gestellt.

4.3.4 Eisenbahn-Anwendung

Durch die Nutzung des STM32F303K8 bietet es sich an, die von STM zur Verfügung gestellten Softwarepaket zu nutzen. Dieses umfasst eine komplette Toolchain, inklusive des Konfigurators der Hardware, der IDE zum Programmieren, des Compilers und der Debug-Schnittstelle zur späteren Fehlerbehebung. Die Toolchain heißt STM32CubeIDE und besteht unter anderem aus folgenden Komponenten:

- Firmware v.1.11.2: Firmware für den MC
- STM32CubeMX v6.1.0: Konfigurator
- STM32CubeIDE v1.5.0: Auf Eclipse basierende Programmierumgebung
- ST-Link v2.1: Programmier- und Debug-Schnittstelle
- GCC: GNU-C-Compiler

Wie in der Randbedingung TECH-6 gefordert, wird für die Implementierung auf dem MC die Programmiersprache C verwendet. Der Code wird mithilfe des Konfigurators und der IDE entwickelt. Für die Übertragung auf den MC und das Debuggen wird die integrierte Lösung über den ST-Link verwendet.

4.3.5 Server-Anwendung

Für die Entwicklung der Server-Anwendung wird auf dem Programmiergerät innerhalb der Visual Studio Code IDE ein Python-Skript erstellt. Mithilfe des Moduls PySerial können die in Abbildung 4.4 gezeigten USB-Verbindungen für eine serielle Datenübertragung genutzt werden.

4.3.6 Konventionen

Folgende Konventionen werden an das zu erstellende Software-Projekt gestellt:

Sprache Im Code wird entsprechend der gängigen Praxis die englische Sprache benutzt. Zum einen wird dadurch an den durch CubeMX generierten Code angeknüpft, zum anderen wird der Code für ein breiteres Spektrum verständlich gemacht.

Kommentare und Dokumentation Um die Übersichtlichkeit des Codes zu verbessern, soll eine ordentliche Gliederung und Kommentierung erfolgen.

Modulare Software Soweit möglich und praktikabel sollen Funktionen in Header-Dateien ausgelagert werden, um von dort aufgerufen werden zu können.

Codingstyle Die Konventionen zum Codingstyle werden in Tabelle 4.5 aufgelistet. Diese orientieren sich entfernt an den in C und Python üblichen Vorgaben.

Tabelle 4.5: Codingstyle

Sprache:	C	
Kategorie	Notation	Beispiel
Variablen	camelCase	movementAuthority
Funktionen	PascalCase	StartSendTrainReport
Objekte	–	–
Makros	SCREAMING_SNAKE_CASE	INC_ETCS_CONFIG_H_
Aufzählungstyp	PascalCase	BufferSizes_
Datenstrukturen	camelCase + _struct angehängt	movementAuthority_struct
Sprache:	Python	
Variablen	snake_case	checksum_calc
Funktionen	snake_case	module_answer_seek_Tag
Objekte	snake_case	module_simulation
Makros	–	–
Aufzählungstyp	–	–
Datenstrukturen	–	–

4.4 Software-Architektur

Das Kapitel Software-Architektur beschäftigt sich mit der Erstellung der ersten Grundkonzepte der Software. Zuerst wird dabei ein grobes Design erstellt und im Laufe der Analyse konkretisiert und verfeinert. Es werden dabei die Interaktionen der einzelnen Module sowie die dafür genutzten Schnittstellen festgelegt.

Die Hauptaufgaben der Software sind es, die einzelnen Hardware-Module zu initialisieren, deren Funktion zu aktivieren, entsprechende Signale und Telegramme zu empfangen und auszuwerten, die Geschwindigkeitskontrolle basierend auf WPP und MAL durchzuführen, die Geschwindigkeitsmessung und Lokalisierung der Modelleisenbahn durchzuführen sowie regelmäßig Nachrichten an den Server zu senden.

Die folgenden Perspektiven sollen die Funktionsweise der Software verständlicher machen:

Die **Bausteinsicht** soll einzelne Zusammenhänge verdeutlichen. Dafür werden im kleinen Umfang die Module soweit möglich in Klassendiagrammen dargestellt.

Die **Aktivitätssicht** dient dazu, den Programmablauf darzustellen, ohne darauf einzugehen, wie die Programmteile miteinander kommunizieren. Für die Laufzeitsicht wird das System in Ablauf- und Sequenzdiagrammen dargestellt, um den Programmablauf und die Kommunikation mit Server und Balise festzulegen.

Die **Datenverarbeitungssicht** dient zur detaillierten Veranschaulichung der Prozesse in einer Aufgabe.

Im folgenden Kapitel wird, ausgehend von einer Blackbox-Ansicht, die Anwendung immer weiter verfeinert dargestellt.

Dies ist der erste Schritt zu einem Programm. Es dient dazu, die Designkomponenten in der Implementierung durch ein programmiertes Gegenstück zu ersetzen.

4.4.1 Bausteinsicht

UART und Circular Buffer

Für die Kommunikation mit der Fahrzeugantenne und dem Kommunikationsmodul wird jeweils ein UART-Channel benötigt. Die genutzten Telegramme besitzen keine feste Länge. Daher muss die Verarbeitung der Daten byteweise nacheinander erfolgen. Die Telegramme werden direkt nach dem Empfang in dem Circular Buffer gespeichert, der dem Kanal zugehörig ist.

Abbildung 4.6 stellt diesen Zusammenhang zwischen Circular Buffer und UART Kanal dar.

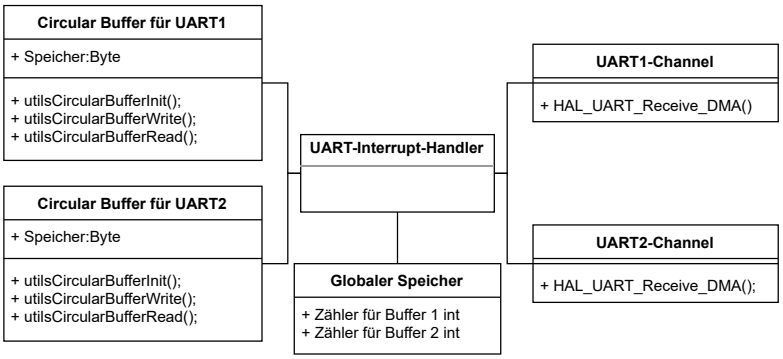

Abb. 4.6: Zusammenhang der Circular Buffer und der UART-Channel

Sobald der UART-Kanal den Empfang einer vollständig übermittelten Nachricht meldet, in diesem Fall eines Bytes, wird der Interrupt-Handler aufgerufen und kümmert sich um das Speichern des Bytes in dem dazugehörigen Buffer. Zusätzlich wird der entsprechende Zähler im globalen Speicher inkrementiert.

Der geschilderte Ablauf findet sich in der Abbildung 4.15 wieder.

Globaler Speicher

Die Klasse Globaler Speicher ist weniger eine Klasse, wie in Abbildung 4.7 dargestellt, als vielmehr eine Übersicht über Variablen und Konstrukte, die global genutzt werden. So beinhaltet der globale Speicher unter anderem die Soll-Geschwindigkeit, an der sich die Geschwindigkeit des Zuges orientieren soll. Damit jede Funktion bzw. Aktivität darauf Zugriff hat, werden solche Informationsquellen global definiert.

Die Abbildung 4.7 zeigt die Struktur des Speichers.

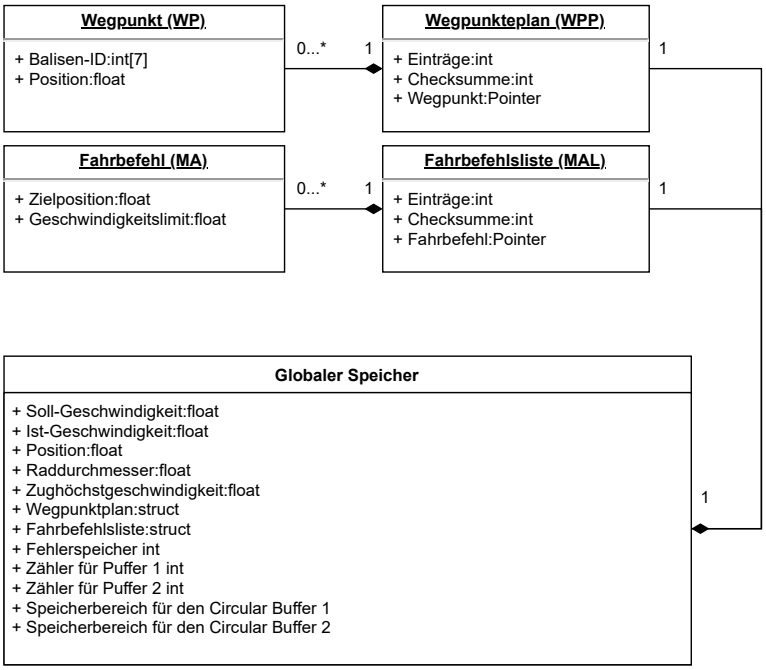

Abb. 4.7: Darstellung des globalen Speichers als Klassendiagramm

Abgesehen von einfachen Variablen enthält der globale Speicher auch den Speicherbereich der Circular Buffer, der durch das Modul des Circular Buffers (`utilsCircularBuffer.h`

und `circularBuffer.c`) verwaltet wird. Auch zeigt die Abbildung den Aufbau des Wegpunkteplans und die Fahrbefehlsliste.

Die beiden Listen selbst bestehen aus den folgenden Komponenten:

- Anzahl der Einträge
  Einem Eintrag über die Menge von Elementen (Wegpunkten/Fahrbefehlen).
- Checksumme
  Die Checksumme der letzten UART-Nachricht zum schnellen Vergleich.
- Pointer auf einen Eintrag
  Der Pointer zeigt auf den ersten Eintrag und kann so später, mithilfe der Anzahl der Einträge, als Array genutzt werden.

Die Einträge, WP und MA, bestehen der Abbildung 4.7 entsprechend aus den Elementen, die für die Nutzung benötigt werden.

4.4.2 Aktivitätssicht

Um den groben Ablauf des Programms zu visualisieren, werden Aktivitätsdiagramme erstellt. Der Ablauf wird in mehrere Stufen unterteilt, um die Funktionen übersichtlicher darstellen zu können. Für die Darstellung der Diagramme wird ein sprachliches Format genutzt, das zum Teil auch bestehende Funktionen widerspiegeln kann.

Initialisierungen und Funktionen, die durch das Konfigurationstool CubeMX durchgeführt und erstellt wurden, werden dabei zwar mit eingezeichnet, jedoch nicht vertiefend analysiert.

Stufe 1

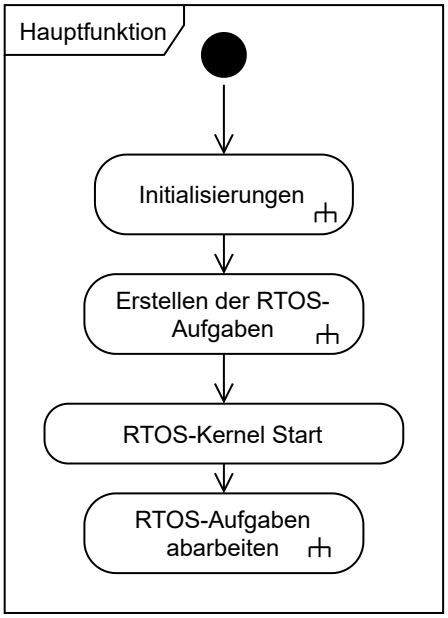

Abb. 4.8: Programmablauf der Hauptfunktion (Aktivitätsdiagramm Stufe 1)

In Abbildung 4.8 ist der Ablauf der Hauptfunktion dargestellt, die direkt nach dem Start des MC abläuft.

Der Ablauf besteht daraus, dass der Controller startet und direkt die genutzten Peripherien initialisiert und Konfigurationen durchführt. Dieser Prozess wird in Abbildung 4.9 dargestellt.
Die Aktivität Erstellen der RTOS-Aufgaben wird durch das Konfigurationstool erstellt und führt die Initialisierung und Erstellung der Aufgaben aus. Der Entwickler selbst nutzt zur Erstellung der Aufgaben die Oberfläche des CubeMX-Tools.
Anschließend wird der RTOS-Kernel inkl. Scheduler gestartet, der dafür sorgt, dass die erstellten Aufgaben anschließend parallel ausgeführt werden (siehe Abschnitt 2.3).
Ein Ende der Hauptfunktion ist im Aktivitätsdiagramm nicht enthalten, da durch die Benutzung des RTOS kein definiertes Ende vorgesehen ist. Auch im Realmodell sollte das System immer zur Kommunikation und Reaktion bereit sein. Somit wäre ein Ende und damit die Abschaltung des Systems ein Fehler, der nicht eintreten darf.

Stufe 2

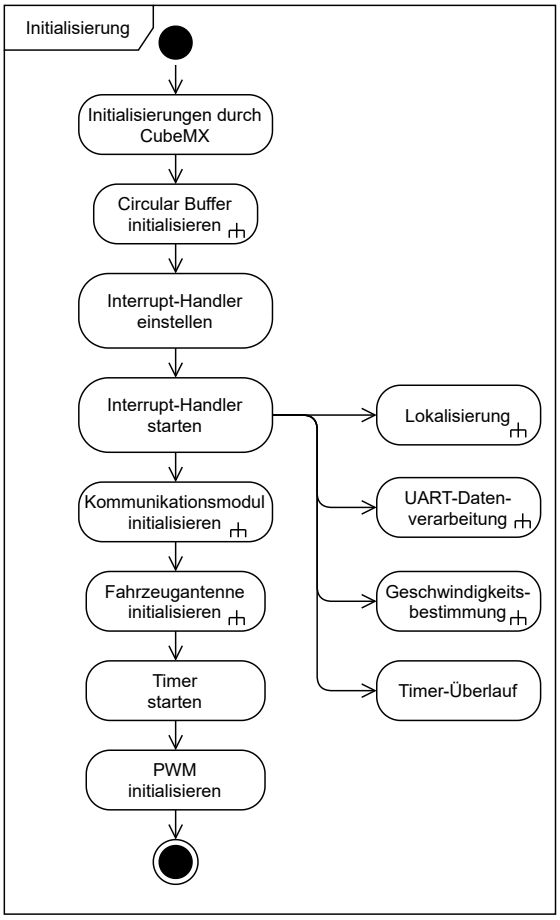

Abb. 4.9: Programmablauf der Initialisierung (Aktivitätsdiagramm Stufe 2)

In der, in Abbildung 4.9 dargestellten Initialisierungsphase werden die von CubeMX erstellten Funktionen zur Konfiguration und Initialisierung der Peripherien ausgeführt. Danach werden die selbst erstellten Initialisierungen abgearbeitet, auf welche in Unterunterabschnitt 4.4.2 genauer eingegangen wird.

Durch die Nutzung der Interrupt-Handler können bestimmte Aufgaben, die durch ein Event (wie einem Interrupt) ausgelöst werden, erst bei Bedarf abgearbeitet werden, ohne durch das Warten auf ein Event den Programmablauf aufzuhalten.
Dazu zählen z. B. die Lokalisierung (vgl. Abbildung 4.21), die UART-Datenverarbeitung (vgl. Abbildung 4.15) oder das einfache Zählen von Odometrie-Signalen und Timer-Überläufen für die Geschwindigkeitsbestimmung (vgl. Abbildung 4.20).

Um einen schnelleren Zugriff auf den Speicher zu erhalten, ohne dass die CPU sich damit befassen muss, kann für die Datenverarbeitung statt der einfachen Interrupt-Methode die Direct Memory Access (DMA)-Funktion für den UART-Empfang genutzt werden. Diese wird neben der Interrupt-Version durch die HAL bereitgestellt.
Die empfangenen Daten müssen byteweise in den dazugehörigen Circular Buffer geschrieben werden. (vgl. Abbildung 4.7 und 4.15)

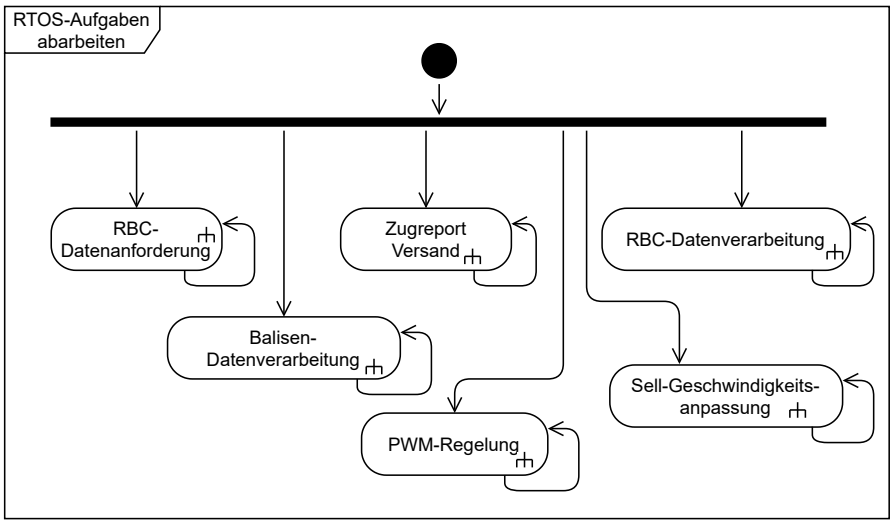

Abb. 4.10: Programmablauf der Aufgaben (Aktivitätsdiagramm Stufe 2)

In der Abbildung 4.10 wird die Aktivität RTOS-Aufgaben abarbeiten dargestellt. Diese Aktivität ist die Sammlung aller mit CubeMX erstellten Tasks, die Aufgaben für den Betrieb der Modelleisenbahn erledigen. Die Ausführung dieser Aufgaben wird unbegrenzt oft wiederholt und die parallele Ausführung wird durch den Scheduler des RTOS-Kernels ermöglicht (siehe Abschnitt 2.3).

Auf den Ablauf dieser Funktionen wird im Unterabschnitt 4.4.3 näher eingegangen.

Stufe 3

Die in Stufe 3 gezeigten Diagramme beruhen auf der Einsicht in die Datenblätter bzw. den Funktionen der dort gezeigten Komponenten. Während der Circular Buffer ein funktionaler Bestandteil der Simulation sein wird, werden die Fahrzeugantenne und das Kommunikationsmodul in dem Programm nicht hardwarenah simuliert. Der Ablauf der Initialisierungen wird, soweit möglich, getrennt von der Modelleisenbahnsoftware

getestet und in das Diagramm übernommen. Die Nutzung der realen Hardware kann erhebliche Unterschiede im Ablauf bewirken und erfordert zu einem passenden Zeitpunkt eine Revision.

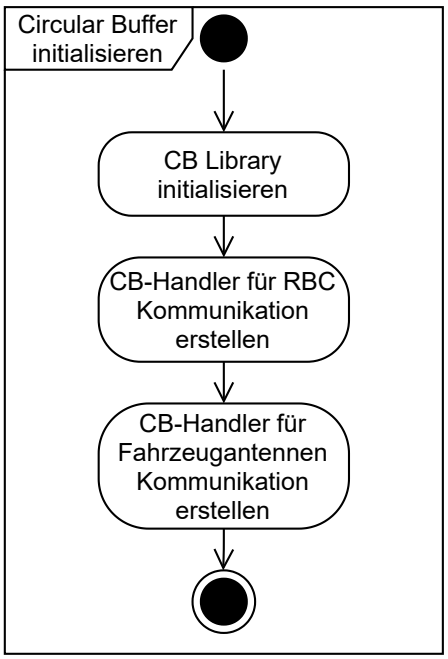

Abb. 4.11: Programmablauf der Circular Buffer Initialisierung (Aktivitätsdiagramm Stufe 3)

Um den Circular Buffer zu nutzen, wird eine Initialisierung der Bibliothek sowie die Erstellung zweier Puffer benötigt (vgl. Abbildung 4.6).

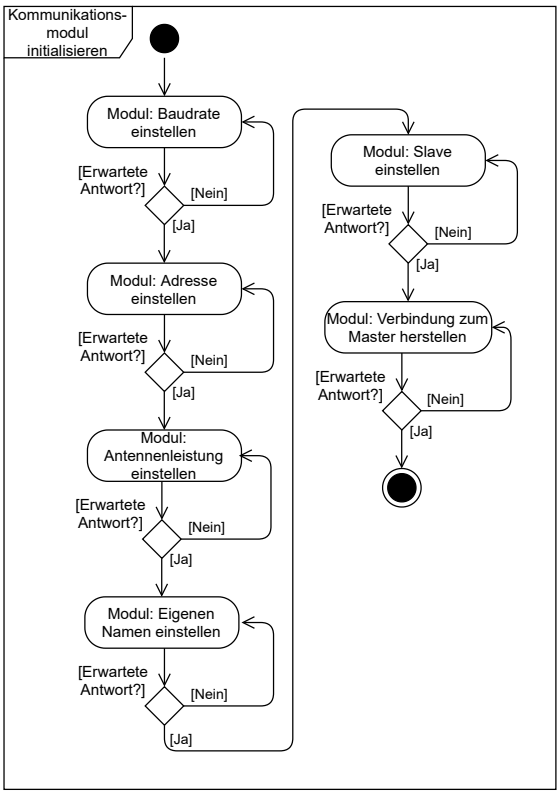

Abb. 4.12: Programmablauf der Initialisierung vom Kommunikationsmodul (Aktivitätsdiagramm Stufe 3)

Das Kommunikationsmodul basiert in dieser Simulation auf dem HC-08 Modul. Konfiguriert wird es mithilfe sogenannter AT-Befehle, welche sich im Datenblatt [46] wiederfinden. Jede Aktivität in dem Diagramm in Abbildung 4.12 entspricht dabei einem dieser AT-Befehle, wird aber zugunsten der sprachlichen Modellierung nicht entsprechend ausgeschrieben.

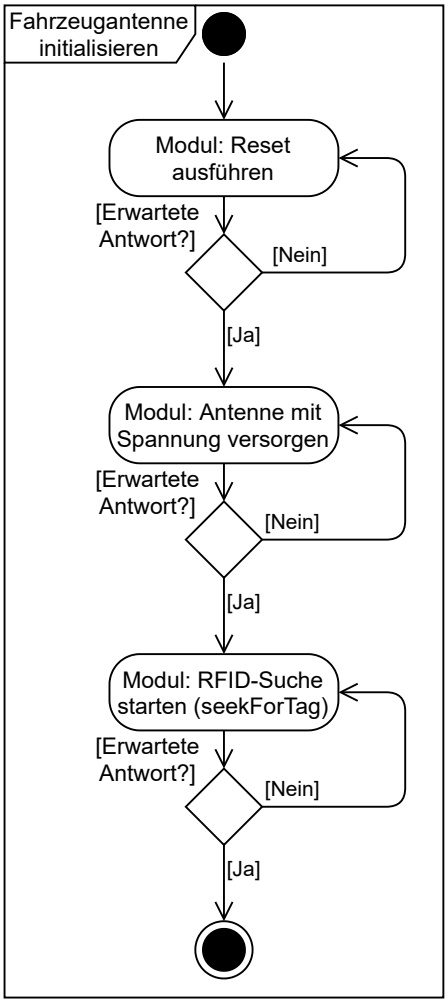

Abb. 4.13: Programmablauf der Initialisierung der Fahrzeugantenne (Aktivitätsdiagramm Stufe 3)

Die Umsetzung der Initialisierung der Fahrzeugantenne ist entstanden in Anlehnung an die Ergebnisse von der Projektgruppe „RFID Positioning Balise In Miniature ETCS System“ um die Autoren Ludwig von Feilitzen, Ikhyel Alama und Militsa Palazova. [34]

Die Schritte zur Initialisierung sind in der Abbildung 4.13 dargestellt. Nach jedem gesendeten Befehl wird auf eine entsprechende Antwort seitens des Modul gewartet. Wenn diese der erwarteten Antwort entspricht, wird der nächste Schritt gestartet, sonst wird derselbe Befehl wiederholt.

4.4.3 Datenverarbeitungssicht

Im folgenden Teil werden einige der Aufgaben als Sequenzdiagramme oder Datenflussdiagramme dargestellt. Diese Diagramme gehören ebenfalls zur Stufe 3 und sind Bestandteil der Abbildung 4.10.

RBC-Datenverarbeitung

Die in Abbildung 4.10 dargestellte Aktivität beinhaltet das Abfragen der Wegpunktdaten vom Server sowie der Fahrbefehle direkt nach dem Start der Eisenbahnsoftware.

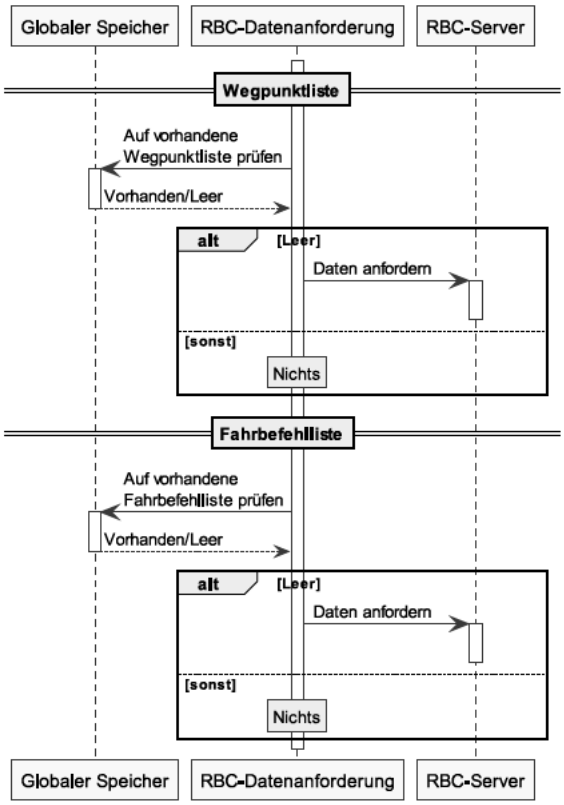

Abb. 4.14: Datenfluss der Anforderung der Daten vom RBC-Server (Sequenzdiagramm Stufe 3)

Solange die Variable für die Einträge gleich Null ist, erfragt diese Aufgabe bei dem Server neue Daten. Sobald ein Eintrag vorhanden ist, wird das Senden von Anfragen

eingestellt. Neue Daten können weiterhin durch den RBC-Server gesendet werden (siehe Abbildung 4.16).

UART-Datenverarbeitung

Die Daten, die über UART vom MC empfangen werden, werden in einem Circular Buffer gespeichert. Das Sequenzdiagramm in Abbildung 4.15 stellt diesen Prozess dar.

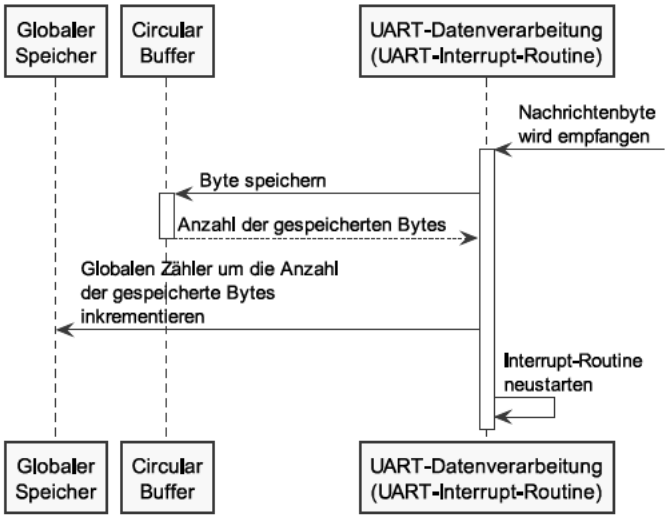

Abb. 4.15: Datenfluss von der Datenverarbeitung der UART-Telegramme (Sequenzdiagramm Stufe 3)

Der Ablauf soll wie folgt aussehen:

1. Auf dem UART-Kanal wird ein Frame einer Nachricht empfangen.
2. Die Interrupt-Routine wird gestartet.
3. Die Routine nutzt die API des Circular Buffers zum Speichern des empfangenen Bytes.
4. Der Circular Buffer gibt die Menge der gespeicherten Element zurück, diese wird genutzt, um den globalen Zähler zu erhöhen.
5. Um das nächste Byte zu empfangen, muss die Interrupt-Routine erneut gestartet werden.

Für jeden UART-Kanal wird ein Circular Buffer verwendet. Der Ablauf ist jedoch für jeden Buffer gleich, solange eine ähnliche Datenstruktur empfangen wird.

Im Gegensatz zu den anderen Aufgaben wird dieser Prozess nicht durch den Scheduler des RTOS ausgeführt, sondern immer wieder durch den Interrupt-Handler des MC angestoßen, sobald ein Byte empfangen wurde. Die Sequenz in Abbildung 4.15 wird für jedes Byte der anstehenden Nachricht ausgeführt und speichert deren Inhalt byteweise in den Circular Buffer.

RBC-Datenanforderung

Nachdem die Daten vom Sender im Circular Buffer gespeichert wurden (vgl. Abbildung 4.15), werden sie in einer weiteren Aufgabe ausgewertet. Die Aufgabe wird dabei vom Scheduler des RTOS organisiert ausgeführt.

In Abbildung 4.16 wird dargestellt, wie die vom RBC-Server in den Circular Buffer geschickten Daten verarbeitet werden.

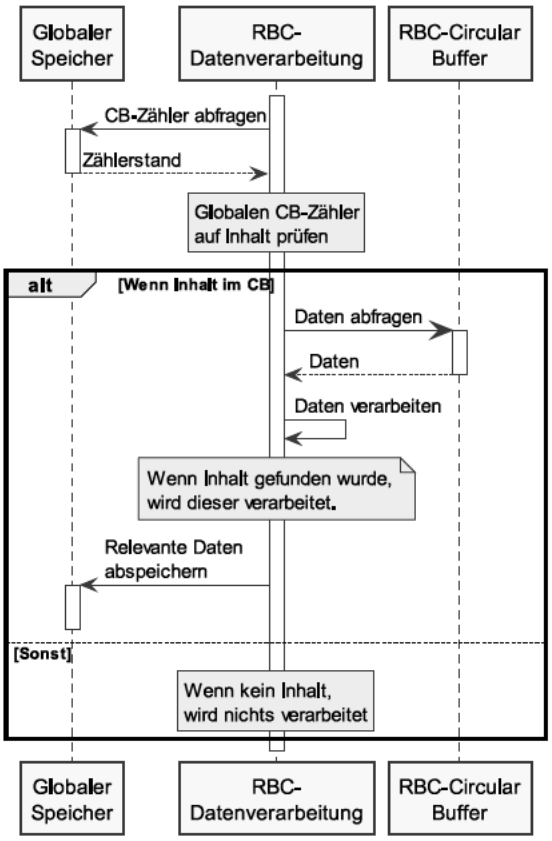

Abb. 4.16: Sequenz zur Verarbeitung der Daten vom RBC-Server (Sequenzdiagramm Stufe 3)

Damit der Prozess nicht ständig den Circular Buffer abfragt, wird mithilfe des im globalen Speicher befindlichen Zählers erkannt, ob neue Daten anstehen. Sobald Daten anstehen, werden diese byteweise verarbeitet, um die genaue Prozedur je nach Inhalt festzulegen.

Balisen-Datenverarbeitung

Analog zu der Datenverarbeitung der RBC-Daten geschieht dasselbe mit den Daten der Balise, wie in Abbildung 4.17 dargestellt.

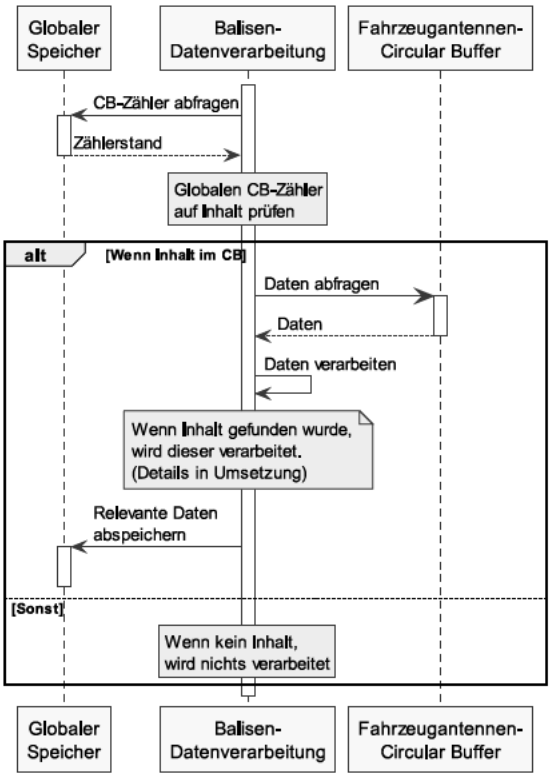

Abb. 4.17: Sequenz der Verarbeitung der Daten von der Fahrzeugantenne (Sequenzdiagramm Stufe 3)

Der einzige Unterschied bei diesem Ablauf ist der genutzte Circular Buffer.

Zugreport Versand

Da die Modelleisenbahn dem Server regelmäßig Geschwindigkeit und Standort mitteilen muss, wird das Versenden eines Zugreports ebenfalls in die durch den Scheduler ausgeführten Aufgaben gestellt.

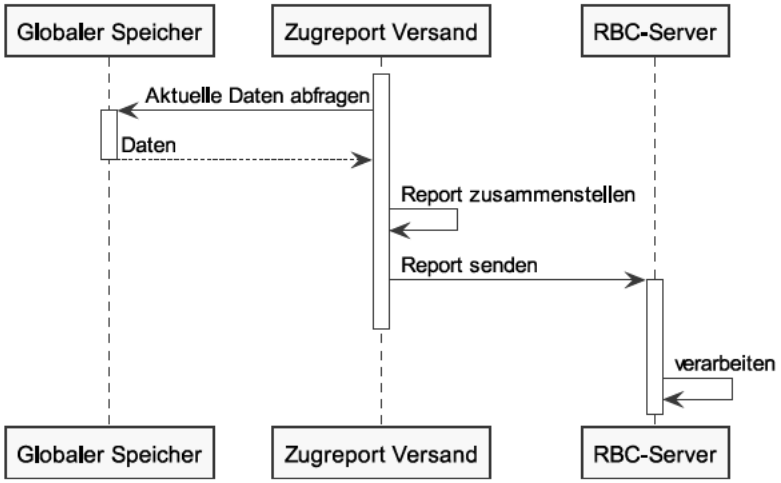

Abb. 4.18: Sequenz des Zugreports an den Server (Sequenzdiagramm Stufe 3)

Sobald die Funktion ausgeführt wird, werden die benötigten Daten aus dem globalen Speicher der Eisenbahnsoftware entnommen. Die Daten werden anschließend in ein Telegramm verpackt, welches gemäß der Definition im Kommunikationsprotokoll (Abschnitt 4.5) aufgebaut ist. Anschließend wird das fertige Telegramm einer nicht-blockierenden HAL-Funktion zum Versenden an den RBC-Server übergeben.
Auf dem Server werden diese Daten dann entsprechend empfangen und verarbeitet.

Optional können hier weitere Betriebsdaten übertragen werden, um neben der aktuellen Geschwindigkeit und Position, die durch ETCS vorgegeben sind, dem Server auch Informationen über die Soll-Geschwindigkeit, den Duty-Cycle oder sonstige für die Leistungsmessung interessante Daten bereitzustellen (siehe Abschnitt 7.3).

Soll-Geschwindigkeitsanpassung

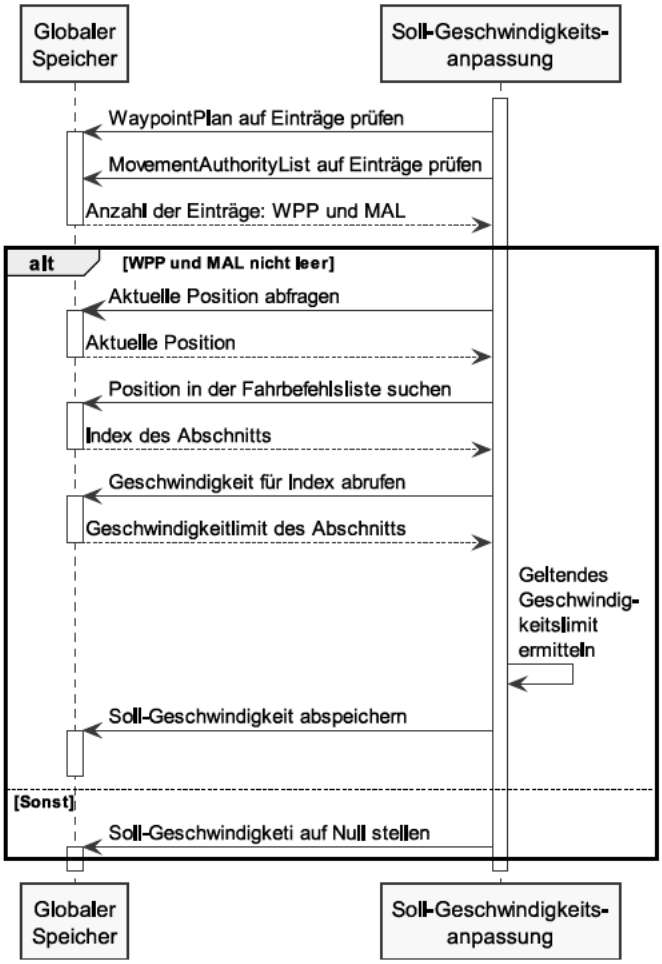

Abb. 4.19: Sequenz der Anpassung der Soll-Geschwindigkeit an MA-Vorgabe (Sequenzdiagramm Stufe 3)

Für die erste Implementierung wird die Soll-Geschwindigkeit zusammen mit dem Fahrbefehl übertragen.
Das Format wird folgendermaßen gestaltet:

- MA1: Fahre mit der Geschwindigkeit x_1 bis zum Kilometer y_1
- MA2: Fahre mit der Geschwindigkeit x_2 bis zum Kilometer y_2
- MA3: ...

Diese Informationen werden bei der Datenverarbeitung der RBC-Nachrichten in der Fahrbefehlsliste gespeichert (siehe Abbildung 4.16).
Mit der aktuellen Position und der daraus resultierenden Indexnummer kann in dieser Liste die Soll-Geschwindigkeit ermittelt werden.
Diese Geschwindigkeit wird in der dazugehörigen globalen Variable gespeichert, die von der PWM-Regelung abgerufen wird (siehe Abbildung 4.23). Ist das Ende der Fahrbefehlsliste erreicht und es existiert kein weiterer Eintrag, wird die Soll-Geschwindigkeit auf 0 gesetzt.

Geschwindigkeitsbestimmung

Um fortlaufend die Geschwindigkeit des Zuges zu ermitteln, wird ein Achsensensor simuliert. Für die erste Implementierung ohne Hardware wird angenommen, dass der Sensor bei einer Umdrehung ein deutliches `High`-Signal übergibt und den Rest der Umdrehung auf einem `Low`-Signal verweilt. Ausgewertet wird dabei nur die steigende Flanke am Sensoreingang.
Im CubeMX-Konfigurator muss der genutzte GPIO-Port auf einen Interrupt bei steigender Flanke eingestellt werden. Ebenfalls wird ein Timer benötigt, der bei einem Überlauf einen Interrupt auslöst. Die Geschwindigkeitsbestimmung findet außerhalb des Aufgabenmanagements des RTOS-Schedulers statt und mit jeder Umdrehung des Rades wird die aktuelle Geschwindigkeit berechnet.
Sobald das Sensorsignal einen Interrupt auslöst, wird die aktuelle Zeit des Timers gespeichert und kann beim nächsten Interrupt mit der dort anstehenden Zeit verglichen werden.
Sollte der Timer zwischen den Interrupts übergelaufen sein, ist es nötig, diese Überläufe zu zählen. Dies kann bei Langsamfahrten geschehen, aber auch im normalen Betrieb, sollte eine Umdrehung kurz vor dem Überlauf des Timers beginnen.
Wenn der Zug in einem Extremfall so langsam fahren sollte, dass die Geschwindigkeit nicht zeitnah ermittelt werden kann, da die Umdrehung mehrere Sekunden braucht, wird die im Speicher liegende Ist-Geschwindigkeit verwendet, bis die Modelleisenbahn 3,14 cm und damit eine vollständige Radumdrehung zurückgelegt hat. Mit dem Maßstab einer H0-Bahn von 1:87 wären das etwas unter 3 Meter ($3,14\,cm \cdot 87 = 273,18\,cm \approx 2,7\,m$) und somit nicht problematisch. Die parallel ablaufende Lokalisierung der Modelleisenbahn nutzt für die Berechnung der zurückgelegten Strecke nicht die Ist-Geschwindigkeit und ist somit auch nicht durch eine alte Geschwindigkeit beeinflusst.

Das Datenflussdiagramm in Abbildung 4.20 zeigt das Vorgehen und die benötigten Datenquellen.

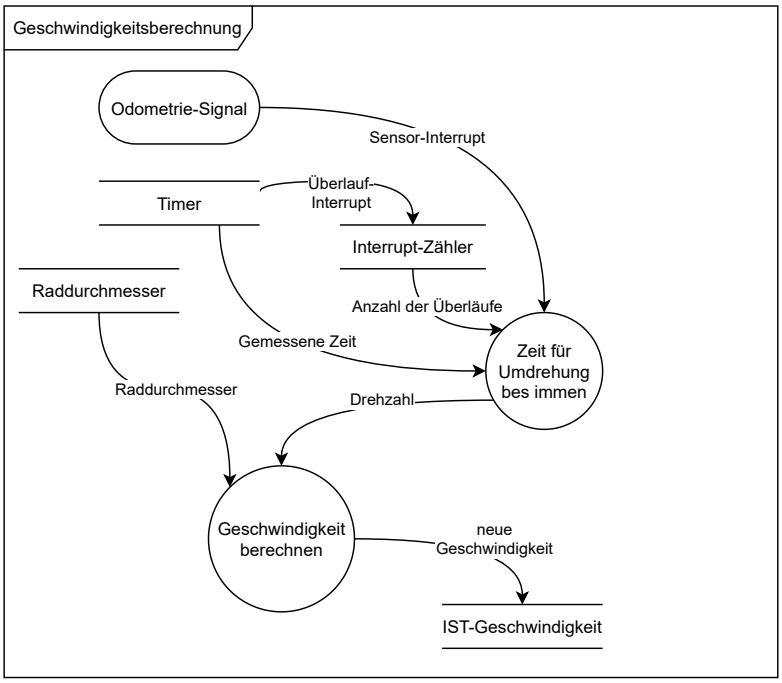

Abb. 4.20: Datenfluss der Geschwindigkeitsbestimmung (Datenflussdiagramm Stufe 3)

Lokalisierung

Die fortlaufende Lokalisierung der Modelleisenbahn basiert auf dem Zählen der Radumdrehungen mithilfe der Odometrie. Mit jedem Interrupt durch die Odometrie wird zusammen mit dem Raddurchmesser die zurückgelegte Strecke berechnet und auf die letzte Position aufgerechnet.

Die Abbildung 4.21 zeigt den Datenfluss dieser Methode.

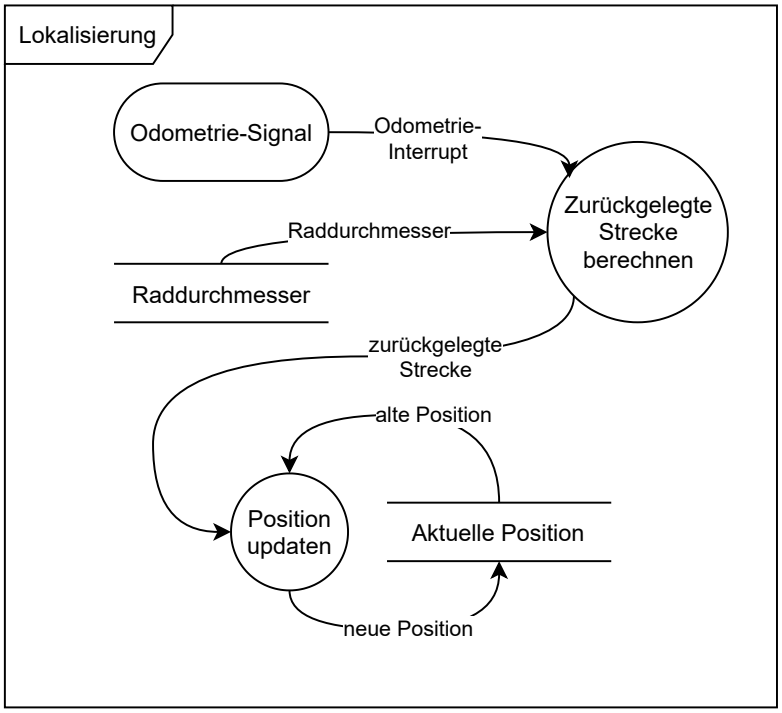

Abb. 4.21: Datenfluss der Lokalisierung durch die Odometrie (Datenflussdiagramm Stufe 3)

PWM-Regelung

Für die erste Implementierung ohne Hardware wird davon ausgegangen, dass die Duty-Cycle/Geschwindigkeits-Kennlinie linear verläuft (vgl. Abbildung 4.22) und daher wird eine Zweipunktregelung verwendet. Die Geschwindigkeitsregelung der Modelleisenbahn wird wie in Unterabschnitt 4.1.3 beschrieben auf einer Halbbrücke basieren, die mithilfe eines Duty-Cycles die Drehzahl des Motors bestimmt.

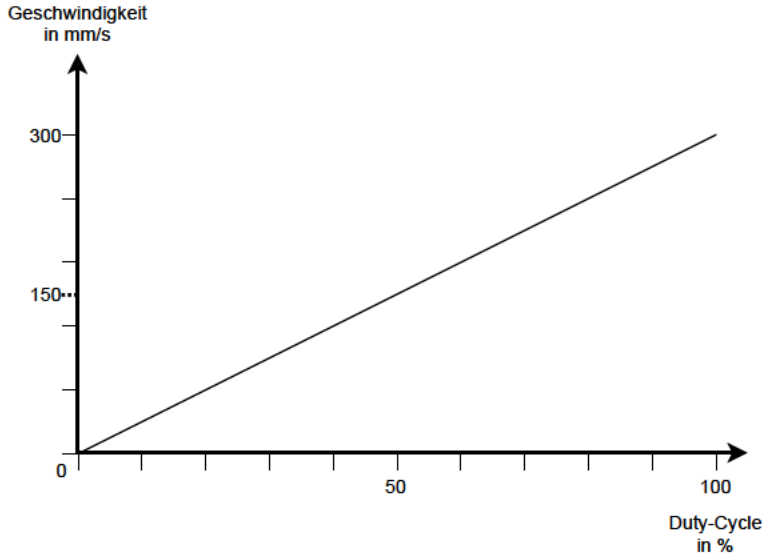

Abb. 4.22: Geschwindigkeit/Duty-Cycle-Diagramm

Für die Reglung des DC wird eine Zweipunktregelung vorgesehen und basiert dementsprechend auf der Soll- und der Maximal-Geschwindigkeit. Aus diesen beiden Werten wird dann der einzustellende Duty-Cycle berechnet, welcher mithilfe der HAL an den PWM-Ausgang weitergegeben wird (vgl. Abbildung 4.23).

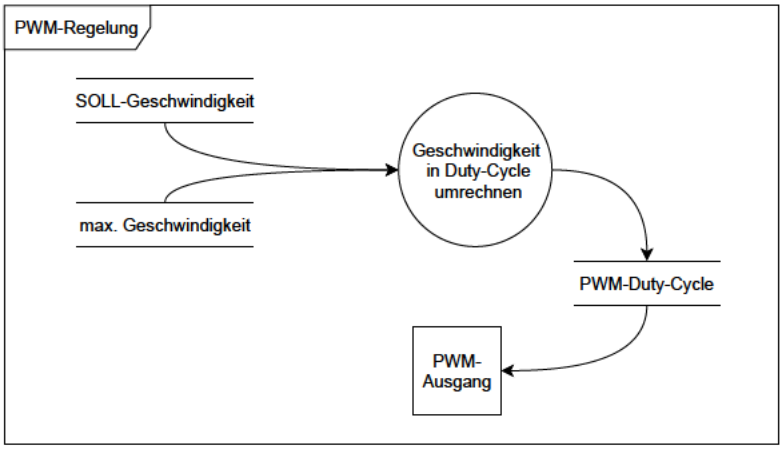

Abb. 4.23: Datenfluss der PWM-Regelung zur Geschwindigkeitsregelung (Datenflussdiagramm Stufe 3)

4.5 Design des Kommunikationsprotokolls

Ein wichtiger Bestandteil des Designs ist das zu verwendende Protokoll für die Kommunikation zwischen Modelleisenbahn und RBC-Server. Um die Flexibilität der Erweiterung beizubehalten sowie den anfallenden Overhead so gering wie möglich zu halten, wird ein eigenes Protokoll entwickelt.

Das hier entwickelte Design wird bereits für das Design der Datenverarbeitung in Unterabschnitt 4.4.3 verwendet.

4.5.1 Aufbau

Der Aufbau einer Nachricht soll einfach gestaltet sein und der benötigten Größe entsprechen, um überflüssigen Overhead zu verhindern. Dafür soll anstelle einer festen Nachrichtengröße und der Aufteilung der Nutzlast auf mehrere Telegramme eine variable Größe der Nutzlast in einer Nachricht verwendet werden.

Der geplante Aufbau ist in Tabelle 4.6 dargestellt.
Start und Ende werden durch feste Zeichen definiert. Außerdem enthält das Telegramm Informationen über die Länge der Nutzlast, die aus dem Befehl und den Daten besteht sowie einer Checksumme zur Fehlerüberprüfung. Ein Byte wird vorerst für spätere Zwecke reserviert, auch wenn das Protokoll noch nicht vollständig festgelegt ist.

Tabelle 4.6: Aufbau des Protokolls

Start	Reserviert	Länge	Befehl	Daten	Checksumme	Ende
1 Byte	1 Byte	1 Byte	1 Byte	n Byte	1 Byte	1 Byte

Die Bestandteile aus Tabelle 4.6 sind im Einzelnen:

- Start: Um den Beginn einer Nachricht anzukündigen, wird hier immer ein `0xFF` stehen.
- Reserviert: Ein noch nicht genutztes Byte. Es sollte daher immer `0x00` enthalten.
- Länge: Die Länge enthält die Information der noch kommenden Bytes, bestehend aus Befehl und Daten. Die Länge ist mindestens 1.
- Befehl: Der Befehl enthält die Information, welchen Zweck das Telegramm hat.

- Daten: Die Daten werden entsprechend der angekündigten Länge, abzüglich des Bytes für den Befehl, gefüllt.
- Checksumme: Enthält die Summe der Bytes von Reserviert bis zum Ende der Daten.
- Ende: Das Ende wird durch ein `0xEE` markiert.

4.5.2 Datenübertragung

Die Baudrate der Datenübertragung wird für die erste Implementierung auf 38400 *Baud* festgelegt.

Da der Aufbau einer Nachricht aus mehreren Bytes besteht, wird als Nutzlast `8 Bit` gewählt. Eine Paritätsprüfung ist durch die Nutzung einer separaten Checksumme nicht notwendig. Die Anzahl der Endbits wird auf der Standardkonfiguration von `1` belassen, da zu diesem Zeitpunkt keine Besonderheiten erwartet werden.
Die Frame-Konfiguration nutzt somit den `8N1` Aufbau.

Für die Datenübertragung müssen mehrere Befehle definiert werden, welche die Use-Cases aus Abschnitt 3.5 abdecken und die Anforderungen durch die Eisenbahnsoftware und RBC-Server erfüllen. Für die erste Implementierung werden folgende Befehle mit den dazugehörigen Byte-Codes definiert:

- `0x50` Request WPP: Wegpunkteplan vom Server anfordern
- `0x51` Waypoint Plan: Enthält den Wegpunkteplan
- `0x52` Request MA: Fahrbefehl vom Server anfordern
- `0x53` Movement Authority List: Enthält die Fahrbefehle
- `0x54` Train Report: Enthält Daten zum Zug Report

4.5.3 Fehlererkennung

Um ein fehlerhaftes Telegramm zu erkennen, werden zwei Methoden genutzt:

1. Das Festlegen von Start und Ende
2. Eine Checksumme

Wird bei der Auswertung nicht das vorgesehene Zeichen gelesen, ist die Nachricht als nicht gültig zu interpretieren.

Mithilfe der Checksumme soll eine Prüfung des Inhalts ermöglicht werden. Ein wahrscheinlich auftretender Überlauf des Bytes ist dabei gewollt und soll die Größe der Prüfsumme klein halten.

5 Umsetzung und Entwicklung

Im folgenden Kapitel werden die Designentscheidungen umgesetzt und in der Software implementiert.

5.1 Train Control Protocol HAW-Hamburg

Bevor die Entwicklung anderer Komponenten beginnen kann, müssen die Protokolle für die UART-Schnittstellen erstellt werden. Durch die Nutzung des SM130 als Fahrzeugantenne besteht hier bereits ein durch den Hersteller vorgegebenes Protokoll [42]. Für die Kommunikation mit dem Server wird das in Abschnitt 4.5 entworfene Protokoll implementiert.

Der Name wird dabei in Anlehnung an die Funktion und an die Einrichtung, die an der Entwicklung beteiligt war, gewählt: TCPHH - Train Control Protocol HAW-Hamburg.

Ein detaillierteres Dokument inklusive Beispiel-Nachrichten befindet sich im Anhang B.

5.2 Konfigurieren des Projekts mit CubeMX

Um ein Projekt mit der CubeMX-Software zu konfigurieren, gibt es die Möglichkeit, CubeMX als alleinstehendes Tool oder die in der STM32CubeIDE integrierte Version zu verwenden. Da die CubeIDE als Entwicklungsumgebung genutzt wird, wird auch darin das Projekt erstellt.
Der Vorteil der Nutzung der alleinstehenden Software liegt in der freien Wahl der Entwicklungsumgebung.

Im Folgenden wird nun erläutert, welche wichtigen Einstellungen innerhalb des Tools getroffen werden. Die vollständige Konfiguration findet sich im angehängten Programmcode bzw. als Projekt auf der beigelegten CD.

5.2.1 Debugging aktivieren

Um die ST-Link Schnittstelle zum Debuggen nutzen zu können, wird unter „SYS“ die Debug-Schnittstelle auf „Serial Wire“ eingestellt.

5.2.2 System Clock

Um den MC mit einer höheren Taktrate zu betreiben, muss im „Clock Configurator“ die Clock `HCLK` verstellt werden. Maximal ist hier 72 MHz möglich, jedoch nur unter Zuhilfenahme eines externen Oszillators und der Aktivierung der „High Speed Clock“ Option. Ohne Modifizierung der Hardware ist eine Taktfrequenz von 64 MHz möglich. Diese wird für die Umsetzung gewählt, da in der Simulation noch keine niedrigen Bearbeitungszeiten notwendig sind.

Die notwendigen Modifikationen an der Hardware, zur Nutzung der maximalen Taktgeschwindigkeit, können dem Kapitel 6.8 des User Manuals [44] des Entwicklungsboards entnommen werden.

5.2.3 Middleware

Eine weitere grundlegende Einstellung ist das Aktivieren der Middleware FreeRTOS. CubeMX bietet hierfür eine Einstellung, die sich um die Implementierung im Projekt inklusive der Erstellung des ersten Tasks kümmert. Mit der Absicht, eine größere Kompatibilität zu bieten, wird das Interface „CMSIS_V1“ gewählt. Zudem bietet „CMSIS_V2“ an dieser Stelle keinen Mehrwert.

Um FreeRTOS zu nutzen, wird die „Timebase Source“ des Systems, entsprechend der Empfehlung von CubeMX, auf „TIM2“ festgelegt.

Heap Size

In den „Config parameters“ wird der Wert der `TOTAL_HEAP_SIZE` geändert. Statt der voreingestellten 3072 Bytes werden 6496 Bytes eingestellt. Da das geschriebene Programm noch nicht optimiert ist und laut Tabelle 4.4 ausreichend RAM zur Verfügung steht, wird etwas mehr als die Hälfte des zur Verfügung stehenden Speichers belegt. Der Rest bleibt

für die Nutzung außerhalb von FreeRTOS frei.
Die Größe des Bereichs muss mindestens der Größe aller Tasks entsprechen.

Tasks

Unter „Tasks and Queue“ können mithilfe eines Assistenten die benötigten Tasks erstellt werden. Neben dem standardmäßig erstellten `defaultTask` werden weitere Aufgaben erstellt, die unter Nutzung der in 3.1 definierten Aufgaben benannt werden.

Folgende Aufgaben sind definiert:

- `defaultTask`
- `sendTrainReport`
- `checkRbcMessage`
- `checkBaliseInfo`
- `setTargetSpeed`
- `changeDutyCycle`

Die Einstellungen werden, bis auf den Funktionsaufruf und die Priorität der Ausführung, vorerst bei den Standardwerten belassen. Die Priorität wird auf „Normal“ gestellt und die Funktionsaufrufe der RTOS-Aufgaben werden von z. B. `StartTask1` in `StartSendTrainReport` umbenannt. Ob genug Speicher bereitgehalten wird, kann unter „FreeRTOS Heap Usage“ nachgeprüft werden.

5.2.4 UART

Für die Datenübertragung zwischen RBC-Server und Modelleisenbahn sowie RFID-Modul und Modelleisenbahn wird jeweils eine UART-Schnittstelle im asynchronen Modus aktiviert. Um den Empfang der Nachrichten zu ermöglichen, wird pro Kanal unter „DMA Settings“ jeweils die RX-Leitung hinzugefügt und der Modus auf „Circular“ gestellt. Anschließend benötigt es noch der Aktivierung des globalen Interrupts in den „NVIC Settings“.

Die Anschlüsse für den UART Kanal 1, im Pinout RX auf D0 und TX auf D1 gelegt, werden für die Kommunikation mit dem RBC-Server verwendet. In der Simulation wird diese Schnittstelle durch den FTDI-Adapter mit dem Programmiergerät verbunden. In CubeMX werden diese Ausgänge außerdem mit den Namen `USART1_RBC_TX` bzw. `_RX`

bezeichnet.
Der Kanal 2 wird für die Kommunikation mit dem RFID-Modul verwendet. In der Simulation läuft dieser über den Virtuellen Kommunikations-Port (VCP) des ST-Link Adapters (USB-Anschluss auf dem Entwicklungsboard). Die TX-Leitung ist jedoch durch die Lötbrücke SB2 an den Pin A7 angeschlossen. Die RX-Leitung besitzt dagegen keinen hardwareseitig verdrahteten Pin und muss später auf den Ausgang D12 parametriert werden (siehe Abschnitt 7.3). Benannt wurden diese Anschlüsse mit `USART_RFID_TX` bzw. `_RX`.

5.2.5 Timer und PWM

Für die Generierung eines PWM-Signals ist die Nutzung eines Timers notwendig. Für dieses Projekt wird der erste Kanal vom „TIM1“ auf die PWM-Erstellung sowie die interne Clock als Zeitquelle eingestellt. In den „Parameter Settings“ werden anschließend „Prescaler“, „Counter Periode“ (Auto Reload Register(ARR)) und „Pulse“ (Duty-Cycle) eingestellt. Der Duty-Cycle wird später im Code mithilfe eines Register-Zugriffes eingestellt.

Der PWM-Ausgang wird mit dem Label `PWM_1_OUT` versehen und befindet sich auf dem Pin P8.

Zur Berechnung der Parameterwerte eignet sich die Formel

$$Frequenz = \frac{Taktrate}{(Prescaler + 1) \cdot (ARR + 1)}, \tag{5.1}$$

angelehnt an die Formel des CTC-Modus aus dem Buch „Digitaltechnik“ [48, S. 417]. Da sowohl Prescaler als auch Auto Reload Register bei 0 anfangen zu zählen, werden die Parameterwerte in der Rechnung im 1 inkrementiert.

Der STM32 erlaubt es, den Zähler-Endwert mit der Parametrierung des Auto Reload Registers einzustellen. Der maximale Einstellungswert entspricht der Auflösung $2^{16} = 65.536$ des Timers [43]. In dieser Anwendung dient der Endwert dazu, möglichst feine Geschwindigkeitsvorgaben zu ermöglichen.

Da das simulierte System kein Antriebssystem besitzt, wird die Frequenz mit $3\,KHz$ angesetzt. Die Taktrate entspricht der eingestellten System Clock mit $64\,MHz$ und für

den (ARR+1) wird der Wert 10.000 vorgegeben, so dass

$$3.000\,Hz = \frac{64.000.000\,Hz}{(Prescaler + 1) \cdot 10.000}$$

gilt. Um einen Wert für den (Prescaler+1) zu ermitteln, lässt sich die Gleichung auf

$$(Prescaler + 1) = \frac{64.000.000\,Hz}{3.000\,Hz \cdot 10.000}$$
$$(Prescaler + 1) = 2,1\overline{3}$$

umstellen und ausrechnen. Da der Prescaler nur mit natürlichen Zahlen parametriert werden kann, wird von $1,1\overline{3}$ auf 1 abgerundet und eingesetzt. Es ergibt sich

$$Frequenz = \frac{64.000.000\,Hz}{(1 + 1) \cdot 10.000}$$
$$Frequenz = 3.200\,Hz$$

für die Berechnung. Die $3,2\,KHz$ liegen nah genug an den gewünschten $3\,KHz$ und werden für die Parametrierung genutzt.

Damit wird der Timer `TIM1` wie folgt eingestellt:

- Clock Source: Internal Clock
- Prescaler: 1
- Counter Period (ARR): 9999

Zur Zeitmessung in der Geschwindigkeitsbestimmung wird der „TIM3“ aktiviert, die Zeitquelle wird ebenfalls auf die interne Clock eingestellt. Zudem werden die Parameter so geändert, dass jede Sekunde ein Überlauf stattfindet und der globale Interrupt aktiviert wird. Um die benötigten Parameter zu erhalten wird die Gleichung 5.1 genutzt.
Ein Überlauf pro Sekunde entspricht der Frequenz 1 Hz, die Taktrate ist die $64\,MHz$ der System Clock, was analog zur Berechnung für `TIM1` genutzt wird. Da der `TIM3` zur Zeitmessung genutzt wird, eignet sich eine Auflösung im Millisekundenbereich. Damit eine Sekunde aus 1000 Einheiten besteht, wird für den (ARR+1) der Parameterwert 1000 vorgegeben. Durch Einsetzen der Werte in die Gleichung

$$(Prescaler + 1) = \frac{64.000.000\,Hz}{1\,Hz \cdot 1.000}$$
$$(Prescaler + 1) = 64.000$$

kann der benötigte (Prescaler+1) berechnet werden.

Daraus folgt, dass der `TIM3` wie folgt eingestellt wird:

- Parameter Settings
  - Clock Source: Internal Clock
  - Prescaler: 63999
  - Counter Period (ARR): 999
- NVIC Settings
  - TIM3 global interrupt: Enabled ✓

5.2.6 Ein- und Ausgänge

Um ein schnelles Entwickler-Feedback beim Debuggen zu ermöglichen, wird der PB3 Pin als „GPIO_Output“ definiert. Auf den Nucleo-Entwicklerboard befindeen sich neben einer USB-Kommunikations-LED (LD1, Rot/Grün) eine Power LED (LD2, Rot) und eine User LED (LD3, Grün). Die User LED kann frei genutzt werden und durch die Lötbrücke SB15 wird diese LED direkt an den Pin D13 angeschlossen. Zusätzlich wird die Bezeichnung auf `LED` geändert.

Ebenfalls wird der Pin PA11 als Interrupt-Input für den Achsensensor gewählt und als `AXIS_SENSOR_IN` bezeichnet.
CubeMX übernimmt zwar den größten Teil der Konfiguration als Interrupt-Eingang, dennoch müssen unter „GPIO“ folgende Einstellungen geprüft werden:

- GPIO → PA11 → GPIO mode → „External Interrupt Mode with Rising edge trigger detection“
- NVIC → EXTI line [15:10] interrupts → Enabled: ✓

Weitere Ein- oder Ausgänge werden derzeit nicht benötigt. In Abbildung 5.1 befindet sich eine vollständige Übersicht des Pinouts.

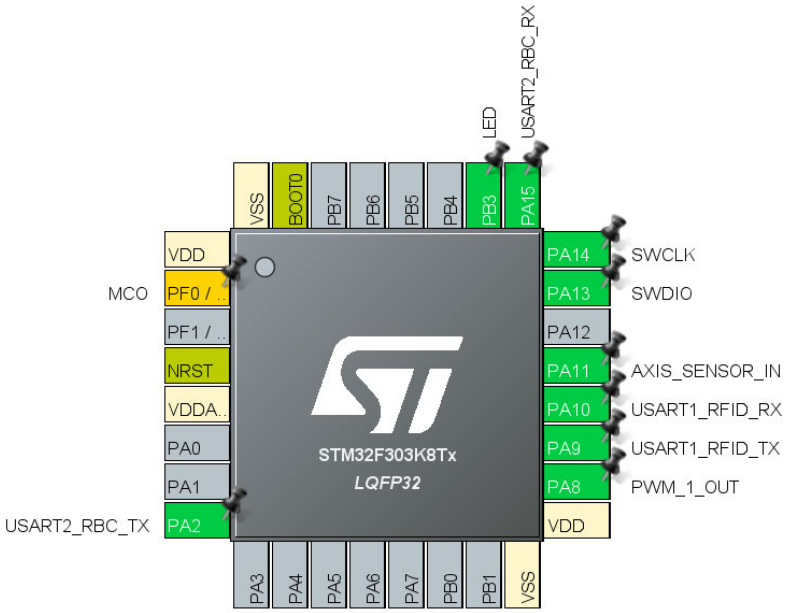

Abb. 5.1: Darstellung der Pinbelegung in CubeMX

Aus der Abbildung C.1 im Anhang wird ersichtlich, wo die in Abbildung 5.1 gewählten Pins am Entwicklerboard abgreifbar sind.

5.3 Entwicklung der Software

Durch die Nutzung von CubeMX erhält der Entwickler direkt ein lauffähiges Projekt, in dem nur die Funktion fehlt. Es werden automatisch die gewählten Konfigurationen in den automatisch generierten Initialisierungen ausgeführt. Für den Code, der später über die Funktion des Programms entscheidet, sind durch Kommentare markierte Bereiche vorgesehen. Untereinander werden diese Bereiche weiter abgegrenzt und mit dem vorgesehenen Nutzen benannt. So steht `Init` für weitere Initialisierungen, `RTOS MUTEX` steht für durch den Nutzer erstellte Mutexe. Außerdem gibt es Bereiche, die nur anhand einer Nummer gekennzeichnet sind, in denen Funktionen ohne bestimmte Kategorie eingeordnet werden können. Die erstellten und genutzten Quellcode-Dateien werden in Tabelle 5.1 aufgeführt.

Tabelle 5.1: Quellcode Dateien

Name	Pfad im Projekt-Ordner	Inhalt
main.c	Core/Src/	Hauptprogramm Quellcode
main.h	Core/Inc/	Hauptprogramm Header
etcs_config.h	Core/Inc/	ETCS Konfiguration
circularBuffer.c	Core/Src/	Circular Buffer Quellcode
utilsCircularBuffer.h	Core/Inc/	Circular Buffer Header

5.3.1 Speicheraufbau

Diverse Konfigurationsparameter befinden sich in der ausgelagerten Headerdatei `etcs_config.h`. Dort enthalten sind die genutzten Puffergrößen, verschiedene Fehlerzustände, die Befehle für die UART-Kommunikation sowie genutzte Wartezeiten.

Die `etcs_config.h` soll es ermöglichen, genutzte Variablen übersichtlich einzustellen und dann global anzuwenden. So ist die Größe des Balisen-Kommunikationsbuffers (`circularBufferSizeBaliseInfo_`) auf 48 Bytes gesetzt, da die längste erwartete Nachricht des RFID-Moduls 22 Byte groß ist. Dieser Puffer kann so zwei Balisen speichern und hat noch 10 % Puffer.

Die Größe des Puffers für die Server-Kommunikation wird auf 510 Byte gesetzt. Derzeit besteht nicht die Möglichkeit, Nachrichten länger als 255 Byte zu verschicken, weshalb der Speicher für zwei Nachrichten der höchsten Länge ausreichend ist.

Für die Fehlerzustände wird hier eine Liste angelegt, die den Wert der Variable `errorState` (8-Bit Signed Integer) einem bekannten Fehler zuordnen lässt. Diese Variable besitzt noch keine Fehlerbehandlung, kann jedoch für Debugzwecke genutzt werden, um den Bereich des Fehlers einzugrenzen.

Die Liste der Befehle enthält ebenfalls nur eine Zuordnung des Befehls zu einem Bytewert. So ist `comCodeRequestWPP` dem Bytewert `0x50` zugeordnet. Sollte sich das Protokoll (siehe Abschnitt 4.5) ändern, kann hier der Bytewert geändert bzw. neue Befehle hinzugefügt werden.

Listing 5.1: Befehlssatz im globalen Speicher

```
enum uartCommunicationCommands {
        comCodeRequestWPP = 0x50,
        comCodeReceiveWPP = 0x51,
        comCodeRequestMA = 0x52,
        comCodeReceiveMA = 0x53,
        comCodeSendTrainReport = 0x54,
};
```

5.3.2 Initialisierungen

Für die Initialisierungen wird der Bereich `USER CODE BEGIN 2` gewählt. Dieser Aufruf befindet sich direkt nach den durch CubeMX erstellten Initialisierungen. Dort werden die Circular Buffer, UART-Interrupt-Handler, PWM, Kommunikationsmodul und Fahrzeugantenne erstellt und initialisiert.

Circular Buffer

Der Code des Circular Buffers wird aus dem Repository des Institute for Theoretical Physics [27] zum Zeitpunkt der Commit-Version `6967a84b` entnommen und dem Projekt-Ordner hinzugefügt.
Die Initialisierung benötigt den einmaligen Aufruf der dazugehörigen Init-Funktion sowie die Erstellung zweier Bufferspeicher.
Der Speicher wird vor der Main-Funktion innerhalb des Bereiches `USER CODE BEGIN PV` (private Variablen) bereits erstellt, inkl. eines temporären Speichers für den UART-Empfang.

UART-Empfang

Um einen zuverlässigen UART-Empfang zu gewährleisten und die eingehenden Telegramme byteweise analysieren zu können, werden diese auch byteweise empfangen und in den Circular Buffer gespeichert.

Hierfür ist es notwendig, den Empfang über eine HAL-Funktion zu starten. Dieser Aufruf enthält dabei UART-Kanal, Speicher und Menge an Bytes, die empfangen werden sollen,

bevor die Callback-Funktion aufgerufen wird. Um einen direkten Speicherzugriff zu ermöglichen und dadurch CPU-Zeit zu sparen, wird hier die DMA-Funktion genutzt.

In der folgenden Abbildung 5.2 werden die benötigten Aufrufe in einem Aktivitätsdiagramm dargestellt.

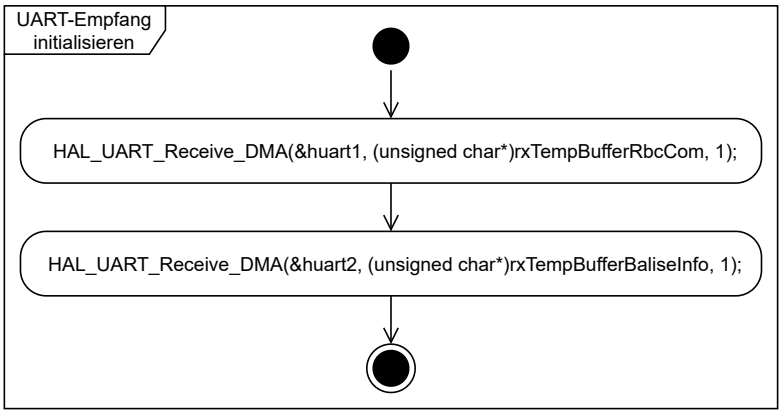

Abb. 5.2: UART-Initialisierung im Aktivitätsdiagramm

Wie der Empfang und das Speichern von Telegrammen abläuft, wird in Unterunterabschnitt 5.3.3 beschrieben.

Fahrzeugantenne

Für die Initialisierung der Fahrzeugantenne, bzw. des RFID-Moduls SM130, werden drei Befehle benötigt:

1. Reset
2. Turn Antenna On
3. Seek for Tag

Diese werden solange nacheinander an das Modul gesendet, bis dies mit der erwarteten Antwort (siehe Datenblatt des Moduls [42]) antwortet. Die Antwort wird durch die UART-Empfangsroutine in den entsprechenden Circular Buffer gespeichert und erhöht damit den Zähler für den Inhalt. Sobald ein Element im Buffer enthalten ist, fängt die Auswertung der Antwort an.

In der folgenden Abbildung 5.3 ist dargestellt, wie das Modul initialisiert wird:

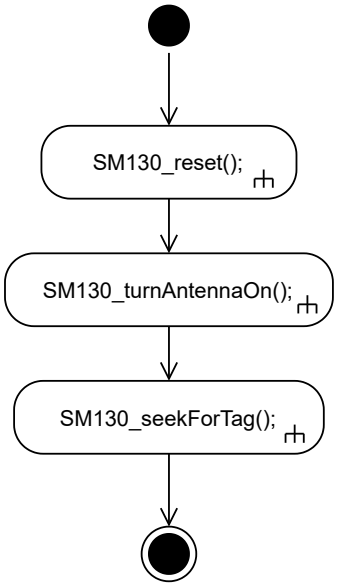

Abb. 5.3: SM130 Initialisierung im Aktivitätsdiagramm

In der Abbildung 5.4 wird der Ablauf des Modul-Resets dargestellt. Die Notiz Byteweise Überprüfung stellt darin eine Wiederholung des darüber durchgeführten Abrufs und Vergleichs dar.
Auf die Darstellung der beiden anderen Befehle wird aufgrund der Ähnlichkeit verzichtet.

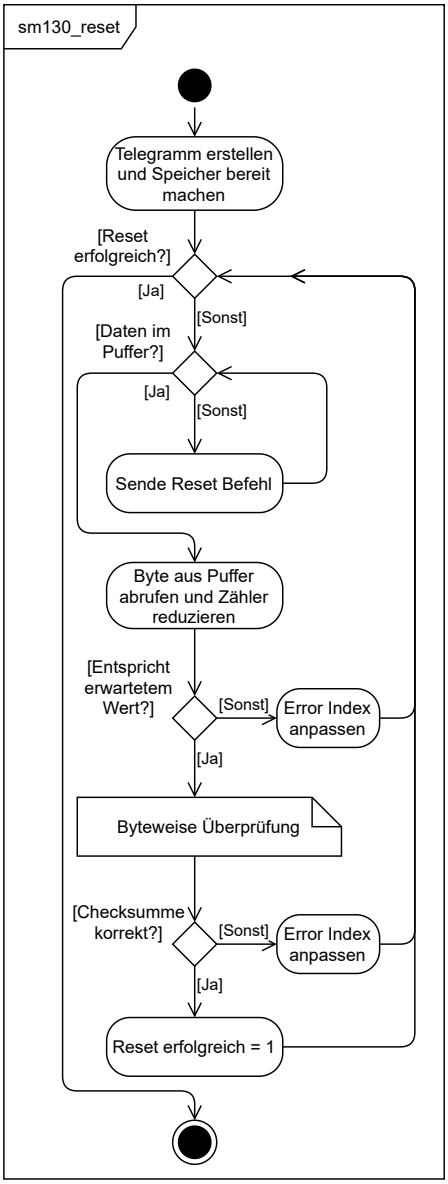

Abb. 5.4: SM130 Reset Ablauf im Aktivitätsdiagramm

PWM

Um den PWM-Ausgang zu starten, wird der Funktionsaufruf `HAL_TIM_PWM_Start_IT(&htim1, TIM_CHANNEL_1);` genutzt.
Dabei wird die PWM-Generierung durch die Nutzung des Interrupt-Modus im Hin-

tergrund gestartet. Durch die Konfiguration in CubeMX wird hier nun eine Frequenz generiert, die den eingestellten Duty-Cycle enthält.

Um den Duty-Cycle zu ändern, wird der Register für den Pulse direkt neu zugewiesen. In Unterunterabschnitt 5.3.4 wird hierauf näher eingegangen.

Timer

Um die Ausführung des Timers zu starten, wird die HAL-Funktion des Timers als Interrupt-Version (`HAL_TIM_Base_Start_IT(&htim3);`) ausgeführt. `HAL_TIM_Base_Start_IT(&htim3);`. Die Parametrierung wird im CubeMX Tool durchgeführt.

5.3.3 Interrupthandler

Einige Funktionen der Modelleisenbahn werden durch die Nutzung der Interrupt-Routinen ermöglicht. Statt eines sequenziellen Aufrufs wie bei den RTOS-Aufgaben, werden hier auftretende Events zum Starten der Funktionen genutzt.

UART-Empfangsverarbeitung

Der Interrupt-Handler `void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)` wird aufgerufen, sobald die eingestellte Menge an Daten auf dem UART-Kanal empfangen wurde. Durch Übergabe des UART-Handlers wird unterschieden, auf welchem Kanal der Empfang vollständig war. Die Funktion enthält die Routinen für alle UART-Empfänge.

In der folgenden Abbildung 5.5 wird die Funktionsweise der Callback-Funktion dargestellt:

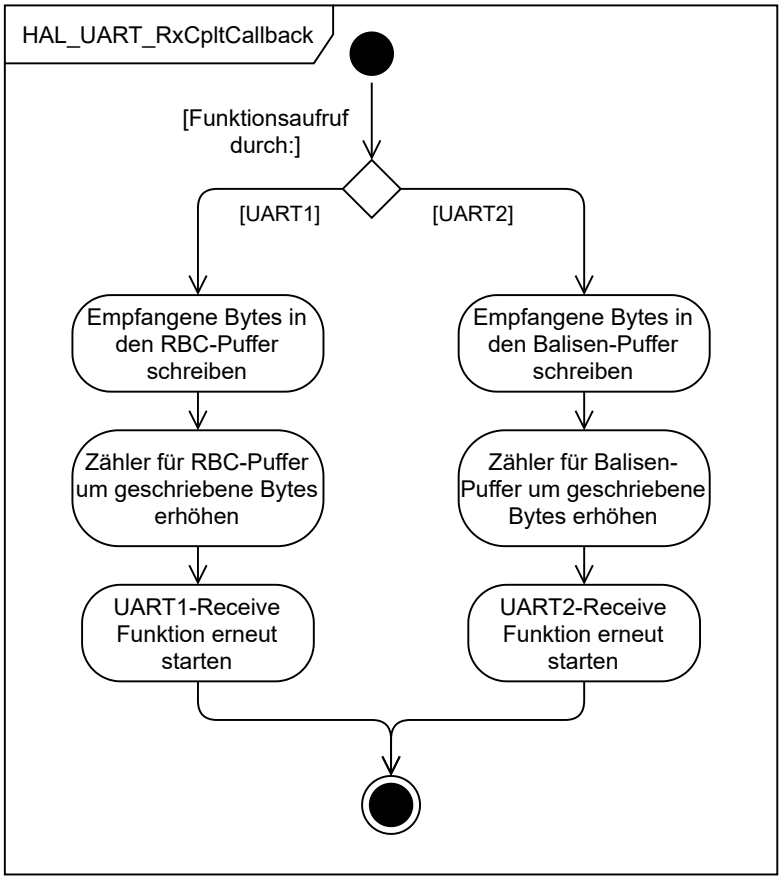

Abb. 5.5: HAL_UART_RxCpltCallback im Aktivitätsdiagramm

Kommunikationsmodul

Für das Kommunikationsmodul wird die Funktion `ComModul_init()` vorbereitet. Da das Kommunikationsmodul in der Simulation durch die direkte serielle Verbindung mit dem FTDI-Adapters ersetzt wird, ist noch nicht bekannt, welche Schritte für die Initialisierung durchgeführt werden müssen.
Dem Datenblatt des HC-08 Moduls kann entnommen werden, dass dafür die AT+-Befehle genutzt werden und der Server als Master mit der Modelleisenbahn als Slave miteinander bekannt gemacht werden müssen, also ein sogenannter Bind durchgeführt werden muss. Für die Simulation sind diese Schritte nicht notwendig und werden deshalb nur in Form eines Platzhalters eingeplant.

Lokalisierung

Für die Lokalisierung wird eine globale Variable in der `etcs_config.h` angelegt, in welcher der Raddurchmesser gespeichert wird. Beim Aufruf des Interrupt-Handlers wird daraus, mithilfe eines festen Teils von Pi, der Umfang des Rades berechnet. Dieser Umfang wird nun bei jeder Umdrehung, beziehungsweise bei jedem Interrupt, auf die aktuelle Position aufaddiert und wieder gespeichert.

Die Funktion selbst wird zusammen mit der Geschwindigkeitsbestimmung in dem Callback-Handler `HAL_GPIO_EXTI_Callback` der Interrupt Routine verwirklicht.

Ein Aktivitätsdiagramm des Interrupts der Lokalisierung wird in Abbildung 5.6 dargestellt.

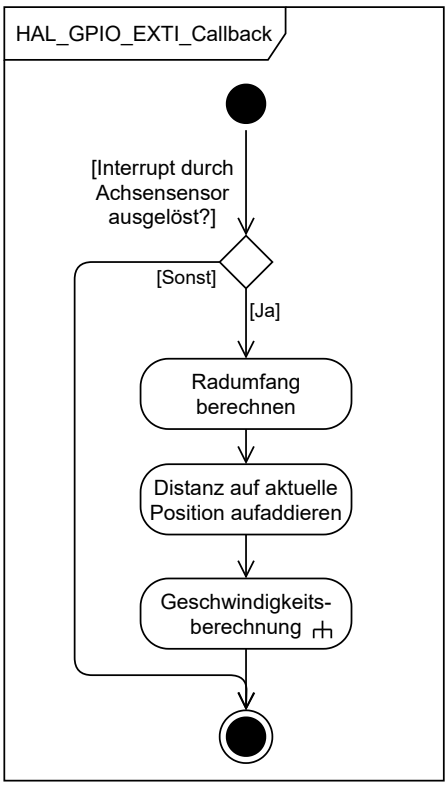

Abb. 5.6: Ablauf der Lokalisierung

Geschwindigkeitsbestimmung

Die Geschwindigkeitsbestimmung wird nach der Lokalisierung in der Callback-Funktion des externen Interrupt-Handlers implementiert. Sie wird direkt nach der Lokalisierung ausgeführt.

Um die Geschwindigkeit berechnen zu können, muss der genutzte Timer `TIM3` auf eine bekannte Dauer eingestellt werden. Diese soll möglichst kurz sein, um eine genaue Bestimmung ermöglichen zu können, jedoch nicht so kurz, sodass die Überlauf-Funktion andere Funktionen blockieren würde. Wie in Unterabschnitt 5.2.5 dargelegt, wird der Überlauf auf eine Sekunde eingestellt.

Die Abbildung 5.7 zeigt den Ablauf der Funktion.

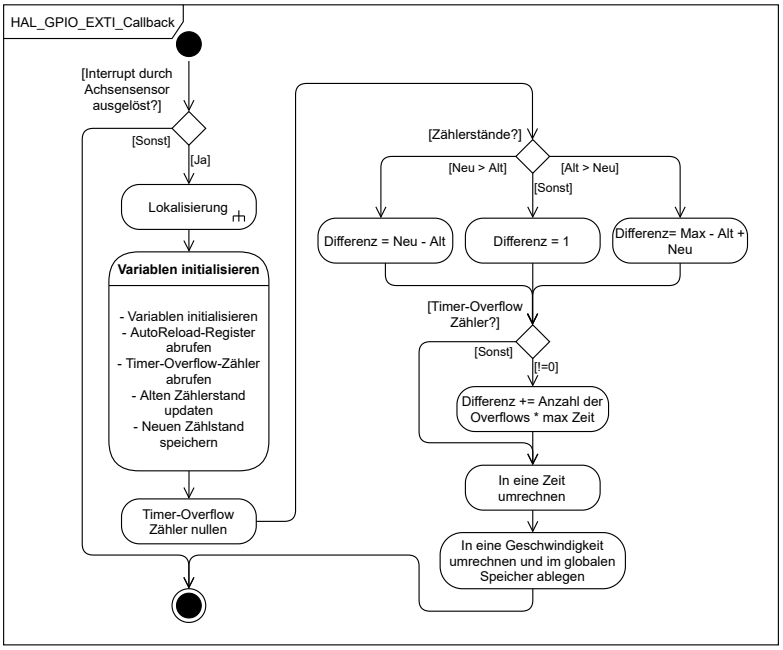

Abb. 5.7: Ablauf der Geschwindigkeitsbestimmung im Interrupt-Handler

Timer-Überlauf

Um das Zählen des Timer-Überlaufs zu implementieren wird die Callback-Funktion `HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)` benötigt. Diese wird jedes Mal aufgerufen, wenn ein Timer überläuft. Bei diesem Aufruf wird der übergelaufene

Timer als Argument mitgeschickt.
Da durch die Nutzung des `TIM2` als Zeitquelle für das FreeRTOS diese Funktion automatisch in den Code eingetragen wird, ist das Hinzufügen des in Listing 5.2 gezeigten Codes ausreichend, um die Überläufe zu zählen.
Dabei wird das übergebene Argument auf `TIM3` geprüft und inkrementiert den globalen Überlauf-Zähler, wenn es zutrifft.

Listing 5.2: Timer-Überlauf

```
if (htim->Instance == TIM3){
        globalTim3OverflowCounter++;
}
```

Zusätzlich wird hier die User-LED, die sich auf dem Entwicklerboard befindet, an- und ausgeschaltet, um den normalen Betrieb anzuzeigen. Dies wird im Design nicht vorgesehen, erleichtert aber die Erkennung des Zustands des MC.

5.3.4 RTOS-Aufgaben

Nachfolgend wird auf die Umsetzung der RTOS-Aufgaben eingegangen.

Daten vom RBC-Server anfordern

Das Anfordern der Daten vom RBC-Server wird in dem Task `defaultTask` bzw. `StartDefaultTask`, wie die Funktion im Source Code genannt wird, implementiert.

Solange keine Daten in den entsprechenden Listen im Speicher liegen, wird hier eine fest definierte Nachricht der `HAL_UASRT_Transmit_IT` Funktion übergeben.

In der folgenden Abbildung 5.8 wird der Ablauf dargestellt.

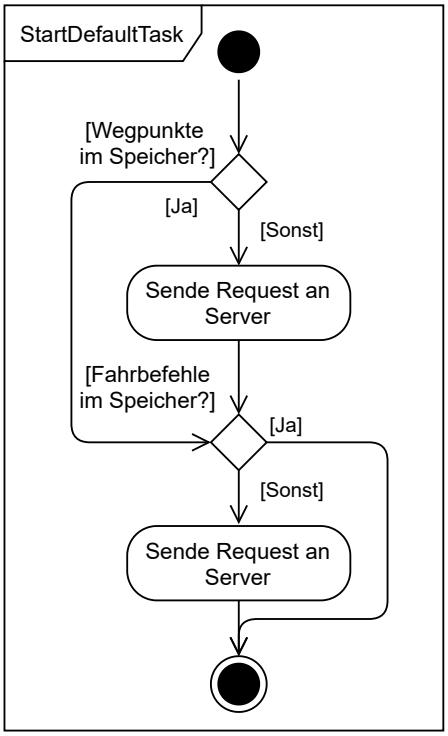

Abb. 5.8: Ablauf der Funktion StartDefaultTask im Aktivitätsdiagramm

Report an den RBC senden

Der Task `sendTrainReport`, bzw. die Funktion `StartSendTrainReport`, kümmert sich darum, regelmäßig Geschwindigkeit und Position an den Server zu schicken. Dies geschieht auch, ohne dass der Zug sich bewegt oder jegliche Fahrdaten (Wegpunkte oder Fahrbefehle) aufweist. Der Sinn ist es, dass der Server trotz stehenden Zuges bereits Kenntnis über dessen Position erhält.

Das Format des zu sendenden Telegramms wird dabei als ein `Struct` (vgl. Listing 5.3) erstellt und wird durch den ersten Aufruf der Aufgabe generiert und gefüllt (vgl. Listing 5.4).
Die variablen Werte Geschwindigkeit, Position und Checksumme werden in der Aufgabenschleife vor dem Versenden abgerufen und in die Nachricht eingefügt.
Die Typendefinition wird mit dem Attribut Packed versehen, um die Größe möglichst gering zu halten und vor allem dafür zu sorgen, dass die Bytes nacheinander liegen und

nicht durch Anweisungen unterbrochen werden. Dies ist vor allem für das Speichern der Fahrbefehle und Wegpunkte sinnvoll.

Listing 5.3: Train Report Struct

```
// Predefined struct for the TR, packed for easy sending
typedef struct __attribute__((__packed__)) {
        uint8_t header;
        uint8_t reserved;
        uint8_t length;
        uint8_t command;
        float speed;
        float position;
        uint8_t checksum;
        uint8_t end;
}trainReport_struct;
```

Listing 5.4: Nachricht generieren

```
// create struct and set fixed parameters
trainReport_struct trainReport;
trainReport.header = 0xFF;
trainReport.reserved = 0x00;
trainReport.length = 0x09; // command + data length
trainReport.command = comCodeSendTrainReport;
trainReport.end = 0xEE;
```

Die Regelmäßigkeit des Reports kann über die Variable `osDelaySendTrainReport` in der `etcs_config.h` eingestellt werden.

Nachrichten von dem RBC-Server verarbeiten

Die Daten vom RBC-Server werden, nachdem sie byteweise im Circular Buffer gespeichert wurden, ebenfalls byteweise aus diesem abgerufen und auf Korrektheit bzw. Inhalt geprüft. Das Vorgehen ähnelt dabei stark dem in Abbildung 5.4 dargestellten. Da diese Funktion jedoch auf verschiedene Längen und Fehler reagieren muss, statt auf eine bereits bekannte und damit erwartete Nachricht, ist diese Überprüfung sehr viel komplexer.

Beim Start dieser Aufgabe wird Speicher für die Bestandteile der Nachricht reserviert sowie Pointer für die späteren Daten erstellt, deren Größe in diesem Moment jedoch noch nicht bekannt ist.

Listing 5.5: Vorbereitung für checkRbcMessage

```
// telegram data
uint8_t reserved = 0;
uint8_t length = 0;
uint8_t entrys = 0;
uint8_t command = 0;
uint8_t checksum_received = 0;
uint8_t checksum_calculated = 0;

// pointer to WPP and MA structs
waypoint_struct *waypointPlanP;
movementAuthority_struct *movementAuthorityP;
```

Der Ablauf der sich wiederholenden Funktion wird in Abbildung D.1 im Anhang D dargestellt.

Nachrichten von der Fahrzeugantenne verarbeiten

Im Gegensatz zu dem stark in der Größe variierenden Dateninhalt, besitzen die Antworten des RFID-Moduls feste Längen. Der benötigte Speicher kann bereits beim Funktionsaufruf reserviert werden. Neben dem Speicher für die einzelnen Komponenten des Telegramms (Länge, Befehl, Prüfsumme) wird daher auch der Speicher für eventuell auftretende Nutzlast initialisiert.

Listing 5.6: Vorbereitung für checkBaliseInfo

```
// used variables
uint8_t i = 0;                  //index
uint8_t equal = 1;              //for use with memcmp (equal==0)

// telegram variables
uint8_t reserved = 0;   // reserved byte for later use
uint8_t length = 0;     // lenght of raw data + command byte
uint8_t command = 0;    // command of received message
```

```
uint8_t payload[BALISE_PAYLOAD_LENGTH]; // storage for tag type
    and serial number
uint8_t tagType = 0;     // tag type of balise
uint8_t serialNumber[BALISE_SERIAL_NUMBER_LENGTH];      // serial
     number of balise
uint8_t checksum_received = 0;  // received message checksum
uint8_t checksum_calculated = 0;// checksum calculated

// module stm130 messages
char seekTag[BALISE_SEEK_TAG_COMMAND_LENGTH] =
    BALISE_SEEK_TAG_COMMAND; //seek for tag cmd

// initialize arrays
for(i = 0; i < BALISE_PAYLOAD_LENGTH;++i)
payload[i] = 0x00;
for(i = 0; i < BALISE_SERIAL_NUMBER_LENGTH; ++i)
serialNumber[i] = 0x00;
```

Neu in dieser Funktion ist es, dass später im Ablauf der gespeicherte Wegpunkteplan nach der neu eingetroffenen Seriennummer durchsucht werden muss, weshalb eine Laufvariable und eine Variable zum Vergleich (`equal`) initialisiert werden. Diese Variable beendet später im Code den Vergleich der empfangenen Seriennummer mit den gespeicherten Seriennummern.
Die dort genutzte Funktion `memcmp` vergleicht zwei Speicheradressen einer definierten Länge miteinander und gibt eine Null zurück, sobald beide Speicherinhalte gleich sind.

Ebenfalls wird die Nachricht erstellt, die das Modul erneut in den Seek For Tag-Modus bringt. Diese Nachricht wird jedes Mal benötigt, wenn das Modul eine Balise erkannt hat oder ein Fehler entstanden ist.

Der komplette Ablauf der sich wiederholenden Funktion wird in Abbildung D.2 im Anhang D dargestellt.

Soll-Geschwindigkeit anpassen

Die Soll-Geschwindigkeitsanpassung (`StartSetTargetSpeed`) wurde entsprechend Abbildung 4.19 erstellt und durch die Abbildung 5.9 dargestellt.
Diese wird jede Sekunde (bzw. je nach Einstellung in `etcs_config.h` nach `osDelaySetTargetSpeed`) ausgeführt.

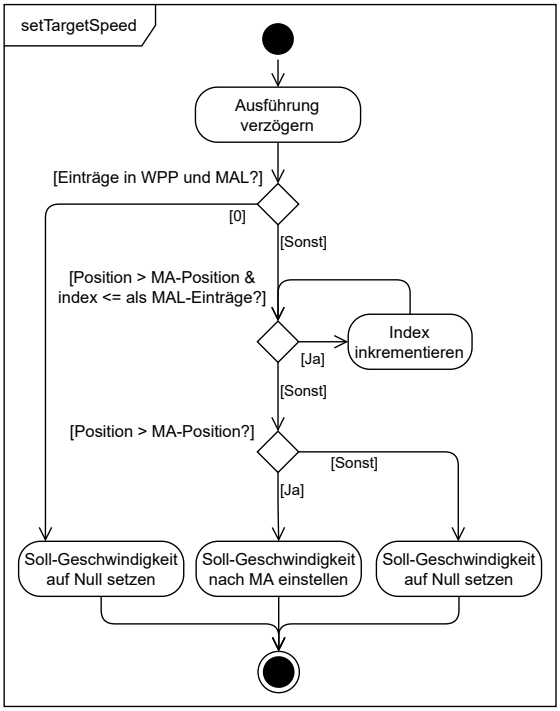

Abb. 5.9: Ablauf der Funktion zum Einstellen der Soll-Geschwindigkeit im Aktivitätsdiagramm

PWM-Regelung

Für die PWM-Regelung wird davon ausgegangen, dass ein Duty-Cycle von 100 % die maximale Endgeschwindigkeit erreichen kann. Die Endgeschwindigkeit kann in der `etcs_config.h` für den Zug parametriert werden. In Abbildung 5.10 wird der Ablauf inkl. der Berechnungen abgebildet.

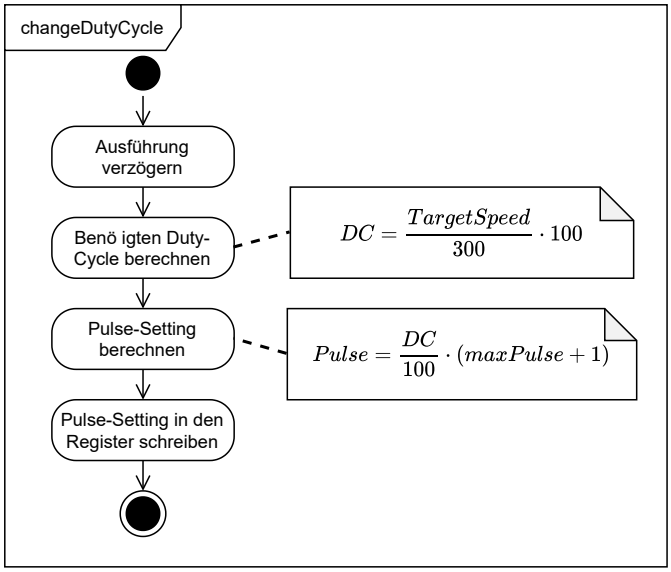

Abb. 5.10: Ablauf der Anpassung des Duty-Cycles nach Geschwindigkeit im Aktivitätsdiagramm

TargetSpeed ist dabei die Soll-Geschwindigkeit, die im globalen Speicher abgelegt wird. DC beschreibt den benötigten Duty-Cycle in Prozent.
Da die Register für den Timer keine Prozentwerte verarbeiten können, wird noch der Pulse berechnet. Dies ist der Zählerwert, an dem die Zustandsänderung durchgeführt wird. maxPulse beschreibt den in CubeMX definierten Wert für das Auto-Reload-Register.

5.4 Git-Ablage

Das komplette Projekt wurde in dem GitLab-Repository auf https://gitlab.tphys.jku.at/haw-hamburg/embedded-c/stm32f303k8/specific-applications/ba-r-diger_willing/tree/devel abgelegt und kann von dort für die Weiterentwicklung abgerufen werden. Zu den versionierten Dateien gehören neben der Eisenbahnsoftware außerdem das Python-Skript (Server-Software), die Konfigurationsdateien für HTerm und die Arbeitsdatei für das Kommunikationsprotokoll.
Eine vollständige Kopie des Gitlab-Repository befindet sich zusätzlich im Anhang auf der CD.

6 Evaluation

Es bedarf einer Prüfstrategie mit Testfällen, um die in Kapitel 3 aufgestellten Anforderungen an der fertigen Software zu testen. In diesem Kapitel werden die genutzten Prüfstrategien dargestellt. Außerdem wird analysiert und dokumentiert, inwieweit die Software auf ihre ordentliche Funktion geprüft werden kann und welche Probleme dabei auftreten.

Ausschlaggebend für die Bewertung der Funktionalität der Software ist die in der Anforderungsanalyse erstellte Tabelle, in der die zu erfüllenden Funktionen detailliert aufgegliedert stehen (siehe Tabelle A.1).

6.1 Prüfstrategie

Für die Tests wird eine sogenannte funktionsorientierte Test-Strategie verwendet. In dieser Strategie werden Testfälle aufgestellt, mit denen systematisch die in der Tabelle festgehaltenen Anforderungen gegengetestet werden. Zum Erstellen der Testfälle werden die Use-Cases aus Abbildung 3.3 und Abschnitt 3.5 verwendet.

Die Tests werden in einem Whitebox-Modell durchgeführt, da teilweise die im Speicher liegenden Variablen als Referenz genutzt werden, um absehen zu können, ob eine Funktion erfolgreich durchgeführt wurde oder fehlgeschlagen ist. Ein Blackbox-Test ist nicht möglich, da Entwicklungsdaten, wie bestimmte Variablen, nicht an den Entwickler übergeben werden können.

Für die Durchführung der Tests wird die manuelle Methode über ein Terminal-Programm genutzt. Eine weitere Methode nutzt ein Python-Skript, um Teile der Anforderungen in einem automatisierten Umfeld zu testen. Wie diese Methoden funktionieren, wird in den folgenden Abschnitten genauer erklärt.

6.1.1 Manuell: HTerm

HTerm ist ein Terminal-Programm, mit dem serielle Übertragungen über die COM-Ports des Programmiergeräts (Computer) durchgeführt werden können. Es liegt derzeit in der Version 0.8.5 vor und kann kostenlos über die URL http://der-hammer.info/pages/terminal.html bezogen werden. Die Software ist lauffähig unter OS Windows und Linux.

Diese Software wird gewählt, um auch die Randbedingung TECH-8 zu erfüllen. Die Lizenz des Programms erlaubt die freie Nutzung unter allen Umgebungen. Verboten ist nur der Verkauf und das Bereitstellen der ausführbaren Datei.

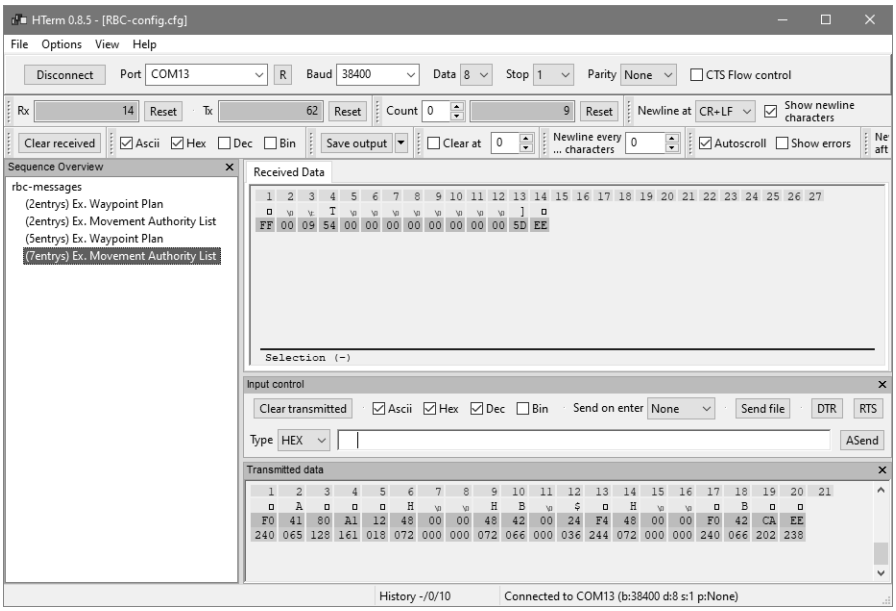

Abb. 6.1: Oberfläche der Software HTerm v0.8.5.0

In der Abbildung 6.1 ist die Oberfläche der konfigurierten Software als RBC-Gegenstelle zu sehen. Auf der linken Seite ist die Liste mit den erstellten Sequenzen zu erkennen, welche per Mausklick versendet werden können. Diese Sequenzen stellen die in einem automatisiertem Betrieb versendeten Nachrichten dar. Nachrichten der Modelleisenbahn werden im mittleren Teil der Softwareoberfläche dargestellt, versendete Nachrichten im unteren.

Für die Prüfung werden zwei Konfigurationen erstellt, welche die UART-Verbindung zum RBC-Server und die UART-Verbindung zu der Fahrzeugantenne widerspiegeln. Eine

Übersicht über die erstellten Sequenzen kann der Abbildung 6.2 entnommen werden. Innerhalb des Programms können auch die Details zu den Sequenzen eingesehen werden. Die Konfigurationsdateien befinden sich auf der angehängten CD unter „Projekt-Ordner/—Tools/hterm/“.

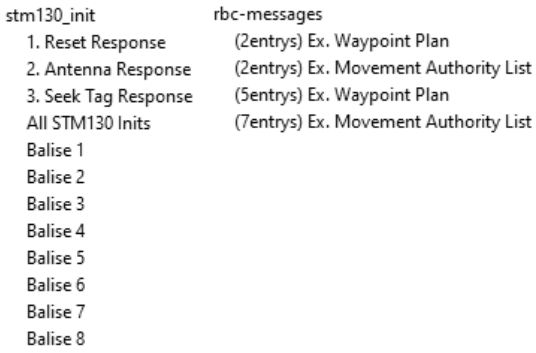

Abb. 6.2: Übersicht über die Sequenzen in HTerm

Die Sequenzen der `stm130_init` Reihe enthalten die Antworten des RFID-Moduls an den Mikrocontroller und ermöglichen die Simulation der Initialisierung. Nach Abbildung 4.13 sind die gespeicherten drei Antworten enthalten und lassen sich auch alle nacheinander mit der `All STM130 Inits` Sequenz verschicken.

Die Sequenzen `Balise 1-8` enthalten die Antworten auf den Seek for Tag-Befehl und stellen acht Balisen aus dem Wegpunkteplan aus der `rbc-messages` dar.

In der `rbc-messages` Konfiguration befindet sich eine Auswahl an Wegpunktlisten und Fahrbefehlslisten. Die Länge dieser Listen ist jeweils vermerkt und in den Details stehen die Informationen zu Position oder Geschwindigkeit.

6.1.2 Automatisch: Python-Skript

Das den RBC-Server repräsentierende Python-Skript wird entwickelt, um die ersten automatisierten Abläufe darzustellen. Auch hier sollen möglichst alle Funktionen getestet werden können, die in Tabelle A.1 aufgezählt werden.

Für die Simulation wird auch die umgesetzte Fahrzeugantennensimulation in den Test eingeschlossen, auch wenn hierfür keine Anforderungen definiert sind. Auf die Simulation

des Kommunikationsmoduls wird verzichtet, da hierzu kein genauer Ablauf bekannt ist.

Das fertige Python-Skript liegt auf der angehängten CD im Ordner „Projekt-Ordner/— Server/“.

6.1.3 Genutzte Hilfsmittel

Die folgenden Hilfsmittel werden in dieser Arbeit zum Testen benutzt.

VisualStudio Code und Python

Der RBC-Server wird als Python-Skript mithilfe von Visual Studio Code, in der Version 1.52.1, entwickelt. Jede andere Python-IDE sollte jedoch ebenfalls kompatibel sein.
Zum Ausführen werden die Dateien aus dem Unterordner „— Server“ des Projekt-Repository benötigt sowie das installierte Python-Modul PySerial und Python in Version 2.7 oder neuer.

Logic Analyzer

Als Logic Analyzer wird ein aus China bezogener Klon verwendet, der auf der obsoleten Hardware eines bekannten Herstellers basiert. Um keine proprietäre Software mit dem Klon zu nutzen, wird die quelloffene Software PulseView [14], in der Version 0.5.0-git-76c2f5b, verwendet.
Der Logic Analyzer wird, wie in Tabelle 6.1 dargestellt, am Mikrocontroller angeschlossen.

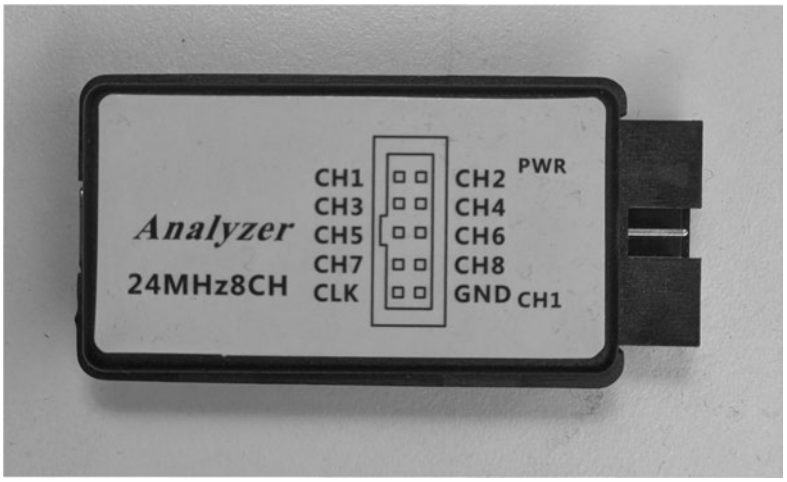

Abb. 6.3: Logic Analyzer (eigenes Foto)

CubeIDE

Für die Beobachtung der Variablen wird der Debug-Modus der CubeIDE genutzt. Im Reiter „Live-Expressions“ werden die zu beobachtenden Variablen eingetragen und können so ohne Programmunterbrechung beobachtet werden.

HTerm

Das Terminal-Programm wird nur für den manuellen Modus genutzt. Da das Python-Skript selbst die Verbindung zu den COM-Ports herstellen muss, ist ein Parallelbetrieb nicht möglich. Im manuellen Modus werden mit dieser Software Telegramme versendet und die von der Eisenbahnsoftware empfangen. Die Interpretation der Telegramme bleibt dabei dem Entwickler überlassen, die Software unterstützt keine Sequenzerkennung.

Taster

Ein Taster wird an dem Achsensensor-Eingang angeschlossen, um die Funktion der Odometrie rudimentär zu simulieren. Da der Taster nicht entprellt ist, kann es zu einer Vielzahl falscher Flankenerkennungen kommen.

6.2 Testaufbau

Für den Test wird der Mikrocontroller wie in Tabelle 6.1 an das Programmiergerät angeschlossen.

Tabelle 6.1: Pinbelegung beim Versuchsaufbau

Komponente	Board-Pin	Bezeichnung	Beschreibung
Taster:1	5 V	5 V	Spannungsversorgung
Taster:2	D10	AXIS_SENSOR_IN	Input-Pin des Achsensensors
Logic Analyzer:CH1	D9	PWM_1_OUT	PWM-Ausgang
Logic Analyzer:CH2	D13	LED	Ausgang der LED LD3
Logic Analyzer:CH3	D1/TX	USART_1_RBC_-TX	Abgriff der RBC-Kommunikation
Logic Analyzer:CH4	D2/RX	USART_1_RBC_-RX	Abgriff der RBC-Kommunikation
FTDI:Rx (ye)	D1/TX	USART_1_RBC_-TX	RBC-Kommunikation über FTDI-Adapter
FTDI:Tx (or)	D2/Rx	USART_1_RBC_-RX	RBC-Kommunikation über FTDI-Adapter
Micro-USB	Micro-USB	VCP (USART_2_-RFID_Tx & _Rx)	Fahrzeugantennenkommunikation über den Virtuellen COM-Port

In Abbildung 6.4 ist dargestellt, wie der reale Versuchsaufbau aussieht.

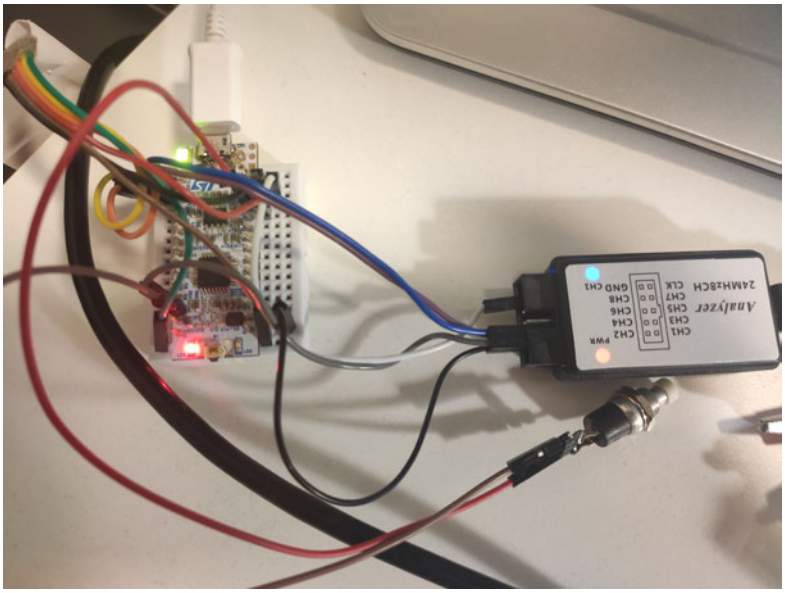

Abb. 6.4: Versuchsaufbau (eigenes Foto)

6.2.1 Testfälle

Ausgehend von den Ergebnissen der Anforderungsanalyse werden nun entsprechende Testfälle entwickelt. Diese Testfälle basieren dabei zum Teil auf dem erstellten Use-Case-Diagramm aus Abbildung 3.3 sowie dem geplanten Ablauf der Software, wie er in Kapitel 4 entworfen wurde.
Die Anforderungen, gegen die getestet wird, werden in der Tabelle A.1 durch „REQ-R“ und „REQ-E“ gekennzeichnet. Diese Anforderungen betreffen die Server- und Eisenbahnsoftware. Die restlichen Anforderungen betreffen die Hardware und sind nicht in dieser Thesis entwickelt worden.

Die erstellten Testfälle sind im Anhang unter Anhang E zu finden.

6.3 Ergebnisse der Prüfung

In der Tabelle 6.2 sind die Ergebnisse der aufgestellten Testfälle (Anhang E) gesammelt und bei Bedarf mit Anmerkungen versehen.

Tabelle 6.2: Testergebnisse

Testfall-ID	Erfolg-reich:	Anmerkung:
S0.1	Ja	Durch RBC-Server und HTerm simuliert.
S0.2	Nein	Funktion wurde nicht implementiert, da der Ablauf nicht hinreichend bekannt war.
S1	Ja	Maximale Einträge: Wegpunkte 23 Stück, Fahrbefehle 31 Stück. Darüber stürzt das Skript ab, da die Variable „Länge“ zu klein ist.
S2	Ja	Der Taster ist nicht entprellt und die Resultate ändern sich unvorhersehbar durch das Auslösen mehrerer Interrupts.
S3	Nein	Funktion wurde nicht implementiert. Die Implementierung der Funktion hatte laut Anforderungstabelle keine hohe Priorität.
E1	Ja	Kommunikationsmodul wird nicht getestet. (Siehe Bemerkung zu S0.2 und Limit des Testfalls (Anhang E))
E2	Ja	Listen mit mehr als 23 Einträgen werden vom Skript nicht unterstützt und können nicht getestet werden. Da im Code die Länge nur mit 1 Byte vorgesehen ist, wird die Eisenbahnsoftware ebenfalls einen Fehler produzieren.
E3	Ja	Listen mit mehr als 23 Einträgen werden vom Skript nicht unterstützt und können nicht getestet werden. Da im Code die Länge nur mit 1 Byte vorgesehen ist, wird die Eisenbahnsoftware ebenfalls einen Fehler produzieren.
E4	Ja	Unbekannte Balisen werden ignoriert.
E5	Ja	
E6	Ja	Leichte Abweichungen des Duty-Cycles (siehe Abbildung 6.6).
E7	Ja	
E8	Ja	Der genutzte Taster ist nicht entprellt und hat keine genauen Ergebnisse pro Betätigung geliefert. Zu schnelles Betätigen hat die restlichen Funktionen blockiert.
E9	Ja	Der genutzte Taster ist nicht entprellt und hat keine genauen Ergebnisse pro Betätigung geliefert. Zu schnelles Betätigen hat die restlichen Funktionen blockiert.

Weiter auf der nächsten Seite

Testfall-ID	Erfolg-reich:	Anmerkung:
E10	Ja	Ausnahmen treten auf, wenn durch die Interrupt-Routinen die Ausführung blockiert wird.
E11	Nein	Funktion wurde nicht implementiert. Die Implementierung der Funktion hat laut Anforderungstabelle keine hohe Priorität.

6.3.1 Anmerkungen zu den Tests

Im Folgenden werden einige der Testergebnisse und deren Resultate genauer analysiert. Dabei können die Pin-Ausgänge visualisiert werden, wie auch die UART-Kommunikation mit dem Server. Die Darstellung der UART-Kommunikation mit der Fahrzeugantenne mit PulseView ist nicht möglich, da diese über die USB-Schnittstelle des Entwicklerboards läuft.

Testfall-ID: E1

In der geprüften Eisenbahnsoftware wird die auf dem Entwicklerboard nutzbare LED in den Timer-Overflow Interrupt eingebaut. Bei jedem Überlauf wird der Pin-Zustand von D13 und damit der Zustand der LED gewechselt. In Abbildung 6.5 ist dieser Zustandswechsel am Verlauf des Signals am LED-Ausgang erkennbar.

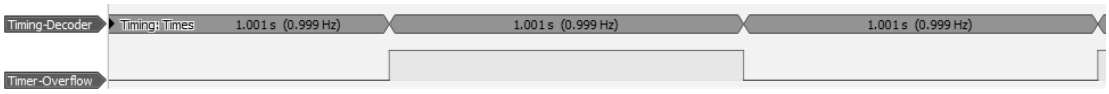

Abb. 6.5: Timer-Overflow in PulseView. Aufgenommen mit einer Abtastrate von 500 kHz.

Der Timing-Decoder von PulseView gibt an, dass der gemessene Wert um 0,1 % von 1 Sekunde abweicht. Diese Abweichung kann daher stammen, dass die HAL-Funktion zum Ändern des Pin-Zustandes innerhalb der Overflow-Routine genutzt wird, da die Berechnung des Timer korrekt ist.

Testfall-ID: E6

In der Abbildung 6.6 wurde der Wechsel des Duty-Cycles von der Geschwindigkeit von 10 $^{mm}/_{s}$ auf die Höchstgeschwindigkeit von 300 $^{mm}/_{s}$ aufgenommen.

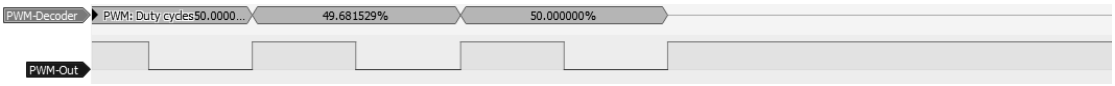

Abb. 6.6: Scope des PWM-Signals während eines Duty-Cycle Wechsels. Aufgenommen in PulseView bei einer Abtastrate von 500 kHz.

Eine geringe Abweichung von den 50 % Duty-Cycle ist jedoch nicht problematisch, da die Geschwindigkeit geregelt werden muss. Bei dem Decodieren des 100 % Duty-Cycles existiert keine fallende Flanke. Daher kann der Decoder keine Duty-Cylce berechnen.

Testfall-ID: E7

Nach der Initialisierung beginnt die Eisenbahnsoftware, den Zugreport über UART an den Server zu senden. Einer dieser Reports wurde in Abbildung 6.7 aufgenommen. Der Inhalt ist durch das Betätigen des Tasters nicht mehr mit Nullen gefüllt und spiegelt den nachfolgenden Inhalt wider. Wird der Inhalt ab dem Wert `0xAB` bis `0x3E` in LSB-Schreibweise abgeschrieben, so resultiert der folgende Hex-Wert `0x3EA102AB`, welcher in einen Float umgewandelt den Wert 0,314473479986 darstellt. Selbiges gilt für die folgenden vier Bytes `0xE4` bis `0x45`. Diese ergeben den Hex-Wert `0x4530B6E4` und damit einen Float-Wert von 2827,43066406. Der erste Float-Wert stellt die Geschwindigkeit aus der Variable `speed_global` dar, der zweite die aktuelle Position aus der Variable `position_global`.

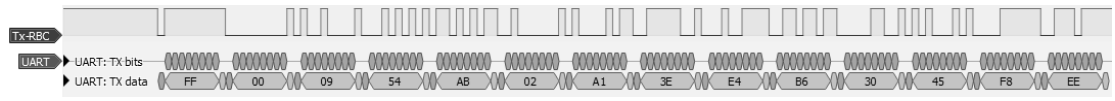

Abb. 6.7: Darstellung des Train Reports in PulseView. Aufgenommen mit einer Abtastrate von 500 kHz

6.3.2 Testfall-ID: E8 und E9

Durch das Testen der fortlaufenden Geschwindigkeits- und Positionsbestimmung ist aufgefallen, dass das Programm beim schnellen Betätigen des Tasters unterbrochen wird. Dies kann daran liegen, dass der Taster nicht entprellt ist und mit einem Druck viele Interrupts auslöst.
Es ist aber wahrscheinlich, dass diese Unterbrechung auch bei hohen Geschwindigkeiten des Zuges auftreten wird.

6.3.3 Testfall-ID: S1, E2 und E3

Durch das Testen verschieden langer Listen wird hier ein Mangel festgestellt, der dazu führt, dass Listen ab einer bestimmten Länge nicht mehr versendet werden können. Durch die Nutzung nur eines Bytes für die Länge des Telegramms beschränkt sich die maximale Anzahl an Einträgen der entsprechenden Listen.

Ausgehend von der Formel

$$Nutzdaten = Größe\ des\ Befehlsfeld + (Einträge \cdot Größe\ eines\ Eintrags),$$

wobei die Menge der Nutzdaten dem Inhalt des Längenfeldes entspricht, kann die Gleichung zu

$$Einträge = \frac{Nutzdaten - Größe\ des\ Befehlsfeld}{Größe\ eines\ Eintrags} \tag{6.1}$$

umgestellt und für die Berechnung der maximalen Anzahl von Einträgen genutzt werden. Durch die für das Längenfeld vorgesehenen 8 Bits ergibt sich durch $2^{8\,Bit} - 1 = 255\,Bytes$ die maximale Größe der Nutzdaten. Für das Befehlsfeld wird 1 Byte eingesetzt und 11 Byte für einen Eintrag der Wegpunktliste (WPP). Die mögliche Zahl der Einträge ergibt sich mit

$$Einträge = \frac{255\,Byte - 1\,Byte}{11\,Byte} \approx 23,09$$

zu maximal 23 Einträgen. Dieses Ergebnis lässt sich durch den Test mit einer 23-Einträge- und 24-Einträge-Liste bestätigen.

Für die Fahrbefehlsliste (MAL), deren Einträge nur 8 Byte groß sind, ergibt dieselbe Rechnung

$$Einträge = \frac{255\,Byte - 1\,Byte}{8\,Byte} \approx 31,75$$

maximal 31 übertragbare Einträge. Auch dieses Ergebnis lässt sich durch die Nutzung einer 31-Einträge- und 32-Einträge-Liste bestätigen.

Falls die Nutzdaten 254 Byte überschreiten, wirft das Python-Skript einen Fehler aus, da die Menge der Nutzdaten nicht mehr im Längenfeld abgebildet werden können.

Die manuelle Erstellung eines Telegramms mit übergelaufener Länge ist möglich und würde in der Eisenbahnsoftware zu einem nicht verarbeitbaren Telegramm führen. Da die Checksumme und das Ende nicht an der vorhergesehenen Stelle stehen würden, würde die Verarbeitung vor dem Erreichen des eigentlichen Endes gestoppt werden. Der Rest der im Circular Buffer liegenden Nachricht würde durch die Suche nach einem neuen Startbyte `0xFF` byteweise verworfen werden.

7 Schlussbetrachtung

7.1 Zusammenfassung

Ziel dieser Arbeit war es, einen Prototypen einer Modelleisenbahnsoftware zu entwickeln, der eine digitale Zugbeeinflussung, angelehnt an das ETCS, realisiert. Die Ergebnisse sollen als Grundlage für die spätere Entwicklung dienen. Zum Teil wurde dabei auf Ergebnisse aus der Vorlesung „Mission Critical Systems“ von Prof. Dr. Buczek zurückgegriffen, welche sich bereits mit Teilen der Hardware auseinandergesetzt hat.

Diese Arbeit fokussierte die Entwicklung und Umsetzung der Software, während die Entwicklung der Hardware bereits mit dem Stand des groben Designentwurfs als beendet galt.

Mithilfe der FreeRTOS Einbindung und den HAL-Bibliotheken, die durch die Toolchain des Mikrocontrollers bereitgestellt werden, konnte der Systementwurf gut umgesetzt werden. Somit konnten die ersten Funktionen des ETCS auf dem Mikrocontroller implementiert werden.

Folgende Eigenschaften sind am Ende der Arbeit funktionsfähig:

- Initialisierung der Fahrzeugantenne
- Auswerten der Balisendaten und Korrektur der Position
- Anfrage und Empfang des Wegpunktplans und der Fahrbefehlsliste
- Vorgabe der Soll-Geschwindigkeit der Position und des entsprechenden Fahrbefehls
- Erzeugen eines PWM-Signals mit variablem Duty-Cylce aus der Soll-Geschwindigkeit
- Bestimmung der aktuellen Geschwindigkeit mithilfe eines Achsensensors
- Bestimmen der fortlaufenden Position mithilfe eines Achsensensors
- Senden eines Zugreport mit Geschwindigkeit und Position an den Server
- Paralleles Abarbeiten der Aufgaben in der Software

Zusätzlich zu der Software wurde die erste Version des TCPHH (Train Control Protocol HAW-Hamburg) entwickelt und zusammen mit der Serversoftware zu dem Projekt-Ordner der Eisenbahnsoftware hinzugefügt.

Bei dem Experimentieren mit dem RFID-Modul ist aufgefallen, dass die Reichweite der Tag-Erkennung sehr eingeschränkt ist. Hier ist es eventuell notwendig, die genutzte Antenne und/oder die genutzten Tags zu wechseln.

7.2 Fazit

Durch das Erstellen einer ersten Hardwarearchitektur konnten die Schnittstellen bereits definiert und so die Software hardwarenah entwickelt werden. Die Lösung ist bei Weitem noch nicht bereit, als fertige Lösung zu gelten.
Stattdessen war es möglich, einige Unzulänglichkeiten zu erkennen, die in späteren Entwicklungen beachtet und verbessert werden müssen.

Mit der umfangreichen Anforderungsanalyse konnten bei der Entwicklung auftauchende Probleme bereits frühzeitig erkannt und umgangen werden. Eines dieser Probleme war das Unvermögen, die über UART übertragenen Telegramme dynamisch auszuwerten. Dies wird dadurch umgangen, dass ein Circular Buffer eingesetzt wird. Durch die Nutzung der Circular Buffer kann die Software sehr viel dynamischer und erweiterbarer gestaltet werden, als anfänglich geplant wurde. Dies reduzierte insgesamt die Komplexität der Verarbeitung und des entwickelten Protokolls.

Die Software selbst funktionierte in der Simulation sehr gut, zeigte in den Tests aber zwei Probleme. Beim Testen der Funktionen ist sichtbar geworden, dass der Speicher für die Telegramm-Länge zu klein angesetzt wurde und so keine realistischen Längen für spätere Fahrten zulässt. Dies hätte verhindert werden können, wenn zu dieser Eigenschaft bereits Anforderungen definiert worden wären.

Die zweite Herausforderung betraf die fortlaufende Geschwindigkeits- und Positionsbestimmung, welche blockierend wirken kann. Die genutzte Lösung kann durch das Auslösen zu vieler Interrupts die Ausführung der FreeRTOS Tasks unterbrechen. In der Simulation mag das an dem nicht entprellten Taster liegen, der pro Betätigung mehr als einen Interrupt auslöst. Später könnte es aber auch zu Problemen führen, wenn ein Hall-Sensor verwendet und die Geschwindigkeit des Zuges zu hoch wird. Derzeit soll der Interrupt und damit die Berechnung bei jeder Umdrehung der Achse ausgeführt werden. Besser wäre es

hier gewesen, die DMA-Funktion des Mikrocontrollers zu benutzen, um die Umdrehungen zu zählen, während die Berechnung selbst in einem RTOS-Task ausgeführt wird. Mangels Erfahrung mit der DMA-Funktion und durch die fortschreitende Projektdauer wurde bei der Entwicklung auf den Einsatz verzichtet.

Ähnlich verlief es bei der Nutzung von FreeRTOS. Erst mit der Bearbeitung des Grundlagenthemas wurde das Modul so tief durchdrungen, dass ersichtlich wurde, weshalb die bekannten Funktionen `malloc()` und `free()` nicht mit FreeRTOS kompatibel sind.
FreeRTOS selbst bietet auch noch weitere Optimierungsmöglichkeiten, die in dieser Arbeit noch keine Anwendung finden. So könnte die Eigenschaft der Queues dabei helfen, die erzwungene Wartezeit nach UART-Übertragungen obsolet zu machen.

7.3 Ausblick

Mit der Bearbeitung dieses Projekts wurden einige Details festgestellt, die in einem Folgeprojekt anders oder besser umgesetzt werden können.

Überarbeiten der Geschwindigkeitslimits Derzeit wird die Geschwindigkeitsvorgabe über den Fahrbefehl der Eisenbahn mitgeteilt. Um jedoch ein Geschwindigkeitsprofil wie in Abbildung 2.6 nutzen zu können, bedarf es auch der Übertragung der Streckenhöchstgeschwindigkeit. Diese müsste mithilfe des Wegpunkteplans dem Zug mitgeteilt werden. Der Fahrbefehl dagegen müsste ein optionales Geschwindigkeitslimit transportieren, welches, wenn vorhanden und geringer als das Streckenlimit, die Geschwindigkeitsvorgabe beeinflusst.
Zusätzlich muss der Zug seine eigene Höchstgeschwindigkeit selbst überwachen und bei überhöhten Geschwindigkeitsvorgaben entsprechend drosseln.

Versenden von Debugdaten Es ist problematisch, Debugdaten aus der Eisenbahnsoftware abzurufen, sobald die Hardware entwickelt ist und auf einer Teststrecke fährt. Hierfür wäre eine mögliche Lösung, parallel zum Zugreport auch eine Funktion zu implementieren, die die Debugdaten an den Server schickt, wo sie dann ausgewertet werden können. Wenn Daten wie Streckengeschwindigkeitslimit, Zughöchstgeschwindigkeit und Fahrbefehlslimit neben der aktuellen Geschwindigkeit und Position übermittelt werden würden, wäre die Erstellung von Geschwindigkeitsprofilen für die Fahrten möglich.

Aber auch andere Daten könnten übertragen werden, wie die Anzahl der Einträge in Wegpunkteplan oder Fahrbefehlsliste, Duty-Cycle in Prozent und relativ zum Register-Wert und viele andere. Dies kann dem Entwickler helfen, den Zug zu überwachen, ohne mit dem Programmiergerät verbunden sein zu müssen.

Positions- und Geschwindigkeitsbestimmung mit DMA implementieren Wie bei den Tests aufgefallen ist, könnte ein zu schnelles Auslösen von Interrupts dazu führen, dass der Rest der Software angehalten wird. Da dieses Verhalten durch die Nutzung eines Tasters als Ersatz für den Achsensensor aufgefallen ist, könnte es zum Teil auch daran gelegen haben, dass dieser nicht entprellt war und so viele Interrupts ausgelöst hat. Möglich wäre diese Blockierung bereits im normalen Betrieb bei höheren Geschwindigkeiten. Die Nutzung der DMA-Funktion bietet sich schon deshalb an, da es dann keine Interrupts mehr geben würde, die zu einer Unterbrechung des restlichen Programms führen könnten. Stattdessen würde die DMA-Funktion die Timer-Überläufe und die Achsensensor-Umdrehungen zählen und weiteren FreeRTOS-Tasks zur Verfügung stellen. Diese Tasks würden dann zyklisch Position und Geschwindigkeit berechnen.
Für spätere Projekte wäre es ratsam, entprellte Taster oder einen digitalen Signalgeber zu benutzen. Ein Signalgenerator mit einer frequenzgesteuerten Rechteckspannung wäre für die Simulation ideal.
Ebenfalls wäre es zu testen, ob der geplante Hall-Sensor ein sauberes Signal übermitteln kann oder ebenfalls entprellt werden muss.

Kommunikationsprotokoll Bei dem Entwickeln der Testfälle ist aufgefallen, dass die Länge der versendbaren Listen ein Limit besitzt. Zukünftig wäre es ratsam, zwei Bytes für die Länge zu nutzen.

Aus der Vergrößerung des Längenfelds von 1 Byte auf 2 Byte würde sich basierend auf der Gleichung 6.1

$$Einträge = \frac{Nutzdaten - Befehlsfeld}{Größe\ eines\ Eintrags}$$

durch Einsetzten der maximalen Nutzdatenmenge von 16 Bit

$$\text{WPP: } \textit{Einträge} = \frac{2^{16}-1}{11} \approx \underline{\underline{5957,73}}$$
$$\text{MAL: } \textit{Einträge} = \frac{2^{16}-1}{8} \approx \underline{\underline{8191,86}}$$

die Anzahl der möglichen Einträge vervielfachen.

Durch Verdoppelung des Speichers würden mehr als 200mal so viele Einträge übermittelt werden können. Für den Wegpunkteplan sind dies 5957 Einträge und für die Fahrbefehlsliste 8191 Einträge.
Alternativ kann überlegt werden, lange Listen auf mehrere Telegramme aufzuteilen und durch neu ins Kommunikationsprotokoll eingearbeitete Befehle, das Ergänzen der vorhandenen Listen zu ermöglichen.

Speicher- und RAM-Optimierung Während der Entwicklung der Software wurden noch keine Versuche unternommen, eine Optimierung des Speicherbedarfs durchzuführen. Die Stack-Sizes der RTOS-Aufgaben wurden einheitlich festgelegt und nicht an den tatsächlich benötigten Speicher der Funktion angepasst.
Bei der Einführung von größeren Listen für Wegpunkte und Fahrbefehle wäre es ebenfalls nötig, diese Listen in dem zwar langsameren aber größeren Flashspeicher abzulegen.

Prioritätenmanagement Derzeit laufen alle RTOS-Aufgaben, bis auf eine Ausnahme, auf derselben Priorität. Sobald eine ordentliche Simulation oder ein Hardware-Prototyp getestet wird, muss hier angepasst werden, welche Funktionen andere Funktionen unterbrechen dürfen.
Dies wird auch wichtig, sollte die Positions- und Geschwindigkeitsbestimmung auf die Nutzung von DMA und Tasks ausgelagert werden.

PWM-Signal Bei dem Testen der Duty-Cycle-Regelung wurde nachträglich festgestellt, dass die Halbbrücke aus der Anforderung anders angesteuert wird. Laut Datenblatt der Halbbrücke ist neben einem PWM mindestens ein weiterer Digital-Ausgang notwendig, um den zweiten Eingang dauerhaft auf `HIGH` zu setzen. Die Nutzung von zwei PWM-Signalen ermöglicht das Umschalten zwischen Vorwärts- und Rückwärtsfahrt.
Dieser Umstand könnte dazu führen, dass die Timer neu verteilt werden müssen. Die elf

verfügbaren Timer unterscheiden sich in ihrem Funktionsumfang erheblich. Außerdem ist ein DC von 100 % nicht die Höchstgeschwindigkeit, sondern entspricht der Geschwindigkeit von 0 $^{\mathrm{mm}}/_{\mathrm{s}}$.

USART 1 oder 2 vom USB-Port auf Pins umlegen Um später beide UART-Teilnehmer nutzen zu können, wird es nötig, den virtuellen COM-Port vom USB auf die Pins D12 und D13 umzulegen. Dazu sieht das Entwicklerboard vor, die zwei Lötbrücken SB2 und SB3 zu entfernen. Dazu gibt es mehr Informationen im User Manual [44] des Mikrocontrollers.

Weitere Modularisierung Auch wenn bereits einiges, was das ETCS betrifft, in eine extra Header-Datei ausgelagert wurde, so ist es noch nicht konsequent beendet. Des Weiteren wurde zum Ende der Entwicklung eine Option im CubeMX gefunden, welche die komplette Peripherie-Initialisierung und FreeRTOS-Tasks in extra Dateien auslagert (Projekt → Code Generierung). Diese Einstellung würde die Übersichtlichkeit steigern, verwirft jedoch bereits geschriebenen Code und wird somit nicht in dieser Arbeit genutzt. Weiterhin wäre das Erstellen einer `etcs_config_init.c`-Quelldatei sinnvoll, um auch die zum ETCS gehörenden Initialisierungen (z. B. Circular Buffer und die Berechnung des Radumfangs) aus der `main.c` bzw. aus den zyklisch aufgerufenen Funktionen zu entfernen.

Fehlererkennung, -darstellung und -umgehung Für die Fehlererkennung wurden bereits einige Fehler definiert und mithilfe einer Index-Variable beim Auftreten gespeichert. Bisher wurde diese Variable aber nur beim Debuggen einzelner Funktionen genutzt und verliert ihren Zustand bei jedem weiteren Fehler. Um später einen reibungslosen Ablauf auf der Stecke zu garantieren, darf die Software auch beim Auftreten eines Fehlers nicht anhalten und muss in einem sicheren Zustand weiter agieren.
Um dem Entwickler Fehlerzustände mitzuteilen, könnte auch die LD3-LED des Entwicklerboards genutzt werden.

Mögliche Hilfsmittel Ein Hilfsmittel, welches nach den Tests entdeckt wurde, könnte die Nutzung von der CubeIDE zur Variablenüberwachung überflüssig machen und lässt es auch zu, Variablenverläufe zu plotten. Das Tool nennt sich STM-Studio oder auch STM32CubeMonitor und wird außerhalb der Toolchain zum Download angeboten. [45]

Kommunikationsmodul Initialisierung und Simulation Mangels einer belastbaren Quelle zur Initialisierung des Kommunikationsmoduls HC-08 wurde auf die Implementierung einer Simulation verzichtet und nur als Platzhalter übernommen. Für eine spätere Simulation muss sich mit der Nutzung der AT+-Befehle auseinandergesetzt werden.

UART-Transmits mit anschließender Wartezeit Derzeit ist nach jedem HAL-UART-Transmit eine kurze Wartezeit eingestellt, damit die Nachrichten vollständig übertragen werden können. Die Übertragung bricht ansonsten beim Verlassen der Funktion ab. Die Nutzung von DMA könnte hier ebenfalls eine Lösung darstellen. Eine weitere potentielle Lösung liegt bei der Nutzung der Queue-Funktion von FreeRTOS.

Custom-PCB Sobald ein Custom-PCB entworfen wird, kann darüber nachgedacht werden, einen externen Oszillator zu verwenden. Dieser lässt sich mithilfe von CubeMX als High Speed Clock-Quelle nutzen und ermöglicht damit das Erreichen der vollständigen 72 MHz Taktfrequenz des SoC. Außerdem lässt sich der Prototyp weiter den Maßen und dem Aussehen einer echten Modelleisenbahn annähern und in dem bestehenden H0-Spur-System an Streckenteilen nutzen.

Literaturverzeichnis

[1] Dr. Peter Hruschka, Dr. Gernot Starke (Hrsg.): arc42. – URL https://arc42.org/. – Zugriffsdatum: 23.12.2020

[2] European Union Agency for Railways (Hrsg.): CCS TSI Annex A - Mandatory specifications. – URL https://www.era.europa.eu/content/ccs-tsi-annex-mandatory-specifications. – Zugriffsdatum: 6.11.2020

[3] reichelt elektronik GmbH & Co. KG (Hrsg.): CNY 70 Reflektierender optischer Sensor mit Transistorausgang. – URL https://www.reichelt.de/reflektierender-optischer-sensor-mit-transistorausgang-cny-70-p6683.html?&trstct=pos_1&nbc=1. – Zugriffsdatum: 10.12.2020

[4] reichelt elektronik GmbH & Co. KG (Hrsg.): DEBO 433 RX/TX Entwicklerboards - 433 MHz RX/TX Modul. – URL https://www.reichelt.de/entwicklerboards-433-mhz-rx-tx-modul-debo-433-rx-tx-p224219.html. – Zugriffsdatum: 10.12.2020

[5] reichelt elektronik GmbH & Co. KG (Hrsg.): DEBO BARC SCAN Entwicklerboards - Barcode-Scanner-Modul für 1D / 2D Codes. – URL https://www.reichelt.de/entwicklerboards-barcode-scanner-modul-fuer-1d-2d-codes-debo-barc-scan-p266062.html. – Zugriffsdatum: 10.12.2020

[6] STMicroelectronics N.V. (Hrsg.): Description of STM32F3 HAL and low-layer drivers. – URL https://www.st.com/content/ccc/resource/technical/document/user_manual/a6/79/73/ae/6e/1c/44/14/DM00122016.pdf/files/DM00122016.pdf/jcr:content/translations/en.DM00122016.pdf. – Zugriffsdatum: 20.11.2020

[7] REICHELT ELEKTRONIK GMBH & CO. KG (Hrsg.): MAR 1006.0901 Schnappschalter 1xUM 10A-400VAC Flachhebel. – URL https://www.reichelt.de/schnappschalter-1xum-10a-400vac-flachhebel-mar-1006-0901-p166849.html?search=1006.0. – Zugriffsdatum: 10.12.2020

[8] REICHELT ELEKTRONIK GMBH & CO. KG (Hrsg.): MK 1466A Reed-Sensor, 180 V, 0,5 A, Schließer. – URL https://www.reichelt.de/reed-sensor-180-v-0-5-a-schliesser-mk-1466a-p151843.html?&trstct=pos_7&nbc=1. – Zugriffsdatum: 10.12.2020

[9] SHOP NFC (SINFOTECH.IT S.R.L.S.) (Hrsg.): NFC Klebetags NTAG213 8,5x17mm. – URL https://www.shopnfc.com/de/nfc-stickers/379-nfc-klebetags-ntag213-8x17mm.html. – Zugriffsdatum: 21.01.2021

[10] MOUSER ELECTRONICS, INC. (Hrsg.): NUCLEO-F303K8. – URL https://www.mouser.de/ProductDetail/STMicroelectronics/NUCLEO-F303K8?qs=kWQV1gtkNndPxYr6NNfTBw%3D%3D. – Zugriffsdatum: 10.12.2020

[11] ARM LIMITED (Hrsg.): NUCLEO-F303K8. – URL https://os.mbed.com/platforms/ST-Nucleo-F303K8/#board-pinout. – Zugriffsdatum: 23.12.2020

[12] ECKSTEIN GMBH (Hrsg.): Original Arduino Nano. – URL https://eckstein-shop.de/Original-Arduino-Nano. – Zugriffsdatum: 10.12.2020

[13] MOUSER ELECTRONICS, INC. (Hrsg.): Pulse W7001. – URL https://www.mouser.de/ProductDetail/PulseLarsen-Antennas/W7001?qs=6JAMBR%252BxeVOE6blnIgNNjw==. – Zugriffsdatum: 10.01.2021

[14] SIGROK (Hrsg.): PulseView. – URL https://sigrok.org/wiki/PulseView. – Zugriffsdatum: 20.01.2021

[15] ROBOTSHOP INC. (Hrsg.): RFID-Modul - SM130 Mifare (13,56 MHz). – URL https://www.robotshop.com/de/de/rfid-modul-sm130-mifare-1356-mhz.html. – Zugriffsdatum: 10.12.2020

[16] ECKSTEIN GMBH (Hrsg.): STM32F103C8T6 Cortex-M3 3,3V Board ARM STM32 ARM32 Arduino IDE kompatible. – URL https://eckstein-shop.de/STM32F103C8T6-Cortex-M3-33V-Board-ARM-STM32-ARM32-Arduino-IDE-kompatible. – Zugriffsdatum: 10.12.2020

[17] REICHELT ELEKTRONIK GMBH & CO. KG (Hrsg.): TLE 4905L Hallsensor, digital, uni-/bipolar, 3,8 - 24 V . – URL https://www.reichelt.de/hallsensor-digital-uni-bipolar-3-8-24-v-tle-4905l-p25717.html?search=tle+4905. – Zugriffsdatum: 10.12.2020

[18] DB NETZ AG (Hrsg.): European Train Control System(ETCS) bei der DB Netz AG. 04.2014. – URL https://www.deutschebahn.com/resource/blob/1303328/d9556ec0c860abb53cf07bfcb693f79d/Anhang_Themendienst_ETCS-data.pdf. – Zugriffsdatum: 6.11.2020

[19] AMAZON WEB SERVICES: FreeRTOS Kernel Developer Docs. – URL https://www.freertos.org/features.html. – Zugriffsdatum: 16.11.2020

[20] AMAZON WEB SERVICES: What is An RTOS?. – URL https://www.freertos.org/about-RTOS.html. – Zugriffsdatum: 16.11.2020

[21] BIGBUG21 ; WIKIMEDIA FOUNDATION INC. (Hrsg.): European Traffic Management Layer. Juli 2020. – URL http://de.wikipedia.org/w/index.php?title=European%20Traffic%20Management%20Layer&oldid=201891948. – Zugriffsdatum: 30.10.2020

[22] DEUTSCHE BAHN: Daten und Fakten 2019. 2019. – URL https://ir.deutschebahn.com/fileadmin/Deutsch/2019/Berichte/DuF_d_web_02.pdf. – Zugriffsdatum: 05.1.2021

[23] DEUTSCHE BAHN AG: Zugbeeinflussungssysteme bei der Bahn. – URL https://www.deutschebahn.com/pr-hamburg-de/hintergrund/themendienste/Zugbeeinflussungssystem-1311016. – Zugriffsdatum: 28.10.2020

[24] EUROPEAN UNION AGENCY FOR RAILWAYS: List of CCS Class B systems. – URL https://www.era.europa.eu/sites/default/files/activities/docs/ertms_ccs_class_b_systems_en.pdf. – Zugriffsdatum: 2.11.2020

[25] GAY, Warren: FreeRTOS. S. 59–72. In: Beginning STM32: Developing with FreeRTOS, libopencm3 and GCC. Berkeley, CA : Apress, 2018. – URL https://doi.org/10.1007/978-1-4842-3624-6_5. – ISBN 978-1-4842-3624-6

[26] INFINEON TECHNOLOGIES AG: TLE 5206-2 Datasheet. – URL https://www.mouser.de/datasheet/2/196/Infineon-TLE5205_2-DS-v01_01-en-785506.pdf. – Zugriffsdatum: 08.12.2020

[27] Institute for Theoretical Physics: Gitlab Repository Server. – URL https://gitlab.tphys.jku.at/haw-hamburg/embedded-c/common/repos/utils/circularBuffer/commit/6967a84b5bdcd67b3d00aba4267f047d2d4db98b. – Zugriffsdatum: 23.10.2020

[28] International Union of Railways: EUROPEAN INTEGRATED RAILWAY RADIO ENHANCED NETWORK: Functional Requirements Specification, Version 8.0.0. URL https://uic.org/IMG/pdf/frs-8.0.0_uic_950_0.0.2_final.pdf. – Zugriffsdatum: 27.10.2020, 12.12.2015. – ISBN 2-7461-1831-7

[29] International Union of Railways: GSM-R: Procurment & Implementation Guide. URL https://uic.org/IMG/pdf/2009gsm-r_guide.pdf. – Zugriffsdatum: 27.10.2020, 15.03.2009. – ISBN 978-2-7461-1631-3

[30] International Union of Railways: EUROPEAN INTEGRATED RAILWAY RADIO ENHANCED NETWORK: System Requirements Specification, Version 16.0.0. URL https://uic.org/IMG/pdf/srs-16.0.0_uic_951-0.0.2_final.pdf. – Zugriffsdatum: 27.10.2020, 21.12.2015. – ISBN 2-7461-1832-4

[31] Jerraya, Ahmed A. ; Yoo, Sungjoo ; Verkest, Diederik ; Wehn, Norbert: Embedded software for SoC. chapter 14, Springer, 2003. – ISBN 978-0-306-48709-5

[32] John, Alexander ; Meran, Renata ; Roenpage, Olin ; Staudter, Christian: Pugh-Matrix. S. 318–320. In: Six $Sigma^{+Lean}$ Toolset: Mindset zur erfolgreichen Umsetzung von Verbesserungsprojekten, Springer, 2014. – ISBN 9783662446133

[33] KBA ; GmbH, Statista (Hrsg.): Bestand an Personenkraftwagen (Pkw) in Hamburg von 2008 bis 2020. 2020. – URL https://de.statista.com/statistik/daten/studie/255132/umfrage/bestand-an-pkw-in-hamburg/. – Zugriffsdatum: 20.01.2021

[34] Ludwig von Feilitzen, Ikhyel Alama, Militsa Palazova: RFID Positioning Balise In Miniature ETCS System. Projektprotokoll

[35] Mayhew, Pam J. ; Dearnley, P. A.: An Alternative Prototyping Classification. In: Comput. J. 30 (1987), Nr. 6, S. 481–484. – URL https://doi.org/10.1093/comjnl/30.6.481

[36] Prof. Dr.-Ing. Ulfert Meiners: Steuerungstechnik. Vorlesungsunterlagen

[37] Prof. Dr. Markus Riemenschneider: Mikroprozessortechnik. Vorlesungsunterlagen

[38] Pulse Elevtronics: W7001 Datasheet. – URL https://pulselarsenantennas.com/wp-content/uploads/2017/01/L152.pdf. – Zugriffsdatum: 08.12.2020

[39] Roland Woitowitz, Klaus Urbanski, Winfried Gehrke: Digitaltechnik: Ein Lehr- und Übungsbuch. 6. Auflage. Springer-Verlag Berlin Heidelberg, 2012 (Springer-Lehrbuch). – ISBN 978-3-642-20871-3,978-3-642-20872-0

[40] Schnieder, Lars: Eine Einführung in das European Train Control System (ETCS): Das einheitliche europäische Zugsteuerungs-und Zugsicherungssystem. Springer Fachmedien Wiesbaden, 2019. – URL https://doi.org/10.1007/978-3-658-26885-5_1. – ISBN 978-3-658-26885-5

[41] Schumny, Harald ; Ohl, Rainer: Vermittlungstechnik. S. 186–193. In: Handbuch Digitaler Schnittstellen. Wiesbaden : Vieweg+Teubner Verlag, 1994. – URL https://doi.org/10.1007/978-3-322-84916-8_10. – ISBN 978-3-322-84916-8

[42] SONMicro: SM130 Datasheet. – URL http://www.sonmicro.com/en/downloads/Mifare/ds_SM130.pdf. – Zugriffsdatum: 08.12.2020

[43] STMicroelectronics N.V.: RM0316 - Reference Manual Rev. 8. – URL https://www.st.com/resource/en/reference_manual/dm00043574-stm32f303xbcde-stm32f303x68-stm32f328x8-stm32f358xc-stm32f398xe-advanced-armbased-mcus-stmicroelectronics.pdf. – Zugriffsdatum: 08.12.2020

[44] STMicroelectronics N.V.: UM1956 - User manual (MB1180) Rev. 5. – URL https://www.st.com/resource/en/user_manual/dm00231744-stm32-nucleo32-boards-mb1180-stmicroelectronics.pdf. – Zugriffsdatum: 08.12.2020

[45] STMicroelectronics N.v.: STM Studio run-time variables monitoring and visualization tool for STM32 microcontrollers. 2019. – URL https://www.st.com/en/development-tools/stm-studio-stm32.html#overview. – Zugriffsdatum: 16.01.2021

[46] Tech, HC: HC-08 BLUETOOTH UART COMMUNICATION MODULE V2.4 USER MANUAL. – URL https://ecksteinimg.de/Datasheet/CP06014/HC-08%20Datasheet.pdf. – Zugriffsdatum: 08.12.2020

[47] Wieczorrek, H.W. ; Mertens, P.: Management von IT-Projekten: Von der Planung zur Realisierung. Springer Berlin Heidelberg, 2008 (Xpert.press). – URL https://books.google.de/books?id=FBnZVpUMJzMC. – ISBN 9783540852902

[48] Woitowitz, Roland ; Urbanski, Klaus ; Gehrke, Winfried: Mikroprozessoren und Mikrocontroller. S. 345–476. In: Digitaltechnik: Ein Lehr- und Übungsbuch. Berlin, Heidelberg : Springer Berlin Heidelberg, 2012. – URL https://doi.org/10.1007/978-3-642-20872-0_9. – ISBN 978-3-642-20872-0

A Ergebnis der Anforderungsanalyse

Auf den folgenden Seiten ist das Ergebnis der Anforderungsanalyse in tabellarischer Form angehängt.

Tabelle A.1: Anforderungen

ID	Anforderungstitel	System	Priorität	Art der Funktionalität	Objekt	Funktionalität	Parameterwert/ Hinweis
REQ-S1	Spannungsversorgung	ETCS-Modelleisenbahn	muss	fähig sein	alternierende Versorgungsspannungen	gleichzurichten	Beim Überfahren von Weichen kann Polaritätswechsel stattfinden
REQ-S2	Spannungsversorgung	ETCS-Modelleisenbahn	muss	fähig sein	schwankende Versorgungsspannungen	zu glätten	Um Schwankungen auszugleichen
REQ-S3	Spannungsversorgung	ETCS-Modelleisenbahn	könnte	fähig sein	verschiedene Spannungen	zu bieten	Um alternative Spannungen für weitere Module zu bieten
REQ-S4	Spannungsversorgung	ETCS-Modelleisenbahn	muss	fähig sein	ein Kurzschließen der Gleisspannung	zu verhindern	Spannungsabnahme darf nicht ungehindert über Vorder- und Hinterräder der Modelleisenbahn geschehen
REQ-S5	Spannungsversorgung	ETCS-Modelleisenbahn	könnte	fähig sein	eine kurze Unterbrechung der Gleisspannung	auszugleichen	Zeitspanne: Überfahren einer Weiche

Weiter auf der nächsten Seite

Tabelle A.1: Anforderungen

ID	Anforderungstitel	System	Priorität	Art der Funktionalität	Objekt	Funktionalität	Parameterwert/ Hinweis
REQ-O1	Odometrie	ETCS-Modelleisenbahn	muss	fähig sein	Achsenrotationen	zu erkennen	Aus Achsenrotation kann Geschwindigkeit und Strecke berechnet werden
REQ-A1	Antriebssystem	ETCS-Modelleisenbahn	muss	fähig sein	per PWM gesteuert werden	zu können	Technische Randbedingung 4: H-Brücke. Variabler Taktgrad
REQ-A2	Antriebssystem	ETCS-Modelleisenbahn	muss	fähig sein	einen Motor	anzusteuern	Beschleunigen, Halten und Bremsen
REQ-F1	Fahrzeugantenne	ETCS-Modelleisenbahn	muss	fähig sein	Balisen (RFID-Tags) mit Energie	zu versorgen	Balisen sind passive Komponenten und werden durch die Energie aktiviert
REQ-F2	Fahrzeugantenne	ETCS-Modelleisenbahn	muss	fähig sein	Balisendaten	auszulesen	Enthält Balisen-ID
REQ-K1	Kommunikationsmodul	ETCS-Modelleisenbahn	muss	fähig sein	mit dem RBC-Server	zu kommunizieren	
REQ-K2	Kommunikationsmodul	ETCS-Modelleisenbahn	muss	fähig sein	eine drahtlose Verbindung	herzustellen	

Weiter auf der nächsten Seite

Tabelle A.1: Anforderungen

ID	Anforderungstitel	System	Priorität	Art der Funktionalität	Objekt	Funktionalität	Parameterwert/ Hinweis
REQ-K3	Kommunikationsmodul	ETCS-Modelleisenbahn	wird	fähig sein	eine feste Gegenstelle	zu aktzeptieren	Ein fester Kommunikationsparter ist für ein „Ein-Raum“ Eisenbahnmodell vorläufig ausreichend. Größere Modelle müssten aber mehrere Server aktzeptieren und automatisch wechseln.
REQ-K4	Kommunikationsmodul	RBC-Server	muss	fähig sein	mit der ETCS-Modelleisenbahn	zu kommunizieren	
REQ-K5	Kommunikationsmodul	RBC-Server	muss	fähig sein	eine drahtlose Verbindung	herzustellen	
REQ-K6	Kommunikationsmodul	RBC-Server	wird	die Möglichkeit bieten	mehrere Teilnehmer	zu unterstützen	Könnte später die Möglichkeit bieten, mehrere Züge zu managen. Siehe ETCS-Spezifikation
REQ-R1	Server-Anwendung	RBC-Server	muss	fähig sein	Wegpunktlisten (WPP)	zu verarbeiten	

Weiter auf der nächsten Seite

Tabelle A.1: Anforderungen

ID	Anforderungstitel	System	Priorität	Art der Funktionalität	Objekt	Funktionalität	Parameterwert/ Hinweis
REQ-R2	Server-Anwendung	RBC-Server	muss	fähig sein	Wegpunktlisten (WPP)	zu übermitteln	
REQ-R3	Server-Anwendung	RBC-Server	muss	fähig sein	Fahrbefehle (MA)	zu verarbeiten	
REQ-R4	Server-Anwendung	RBC-Server	muss	fähig sein	Fahrbefehle (MA)	zu übermitteln	
REQ-R5	Server-Anwendung	RBC-Server	muss	fähig sein	Geschwindigkeitsdaten des Zuges	zu empfangen	
REQ-R6	Server-Anwendung	RBC-Server	muss	fähig sein	Positionsdaten des Zuges	zu empfangen	
REQ-R7	Server-Anwendung	RBC-Server	könnte	fähig sein	zusätzliche Daten des Zuges	zu empfangen	z. B. Soll--Geschwindigkeit, Reglungsdaten, Verbindungsinformationen
REQ-M1	Mikrocontroller	ETCS-Modelleisenbahn	muss	fähig sein	das Signal der Odometrie	zu empfangen	
REQ-M2	Mikrocontroller	ETCS-Modelleisenbahn	muss	fähig sein	ein PWM Signal	zu erzeugen	
REQ-M3	Mikrocontroller	ETCS-Modelleisenbahn	muss	fähig sein	eine UART-Schnittstelle für das Kommunikationsmodul	bereitzustellen	

Weiter auf der nächsten Seite

Tabelle A.1: Anforderungen

ID	Anforderungstitel	System	Priorität	Art der Funktionalität	Objekt	Funktionalität	Parameterwert/ Hinweis
REQ-M4	Mikrocontroller	ETCS-Modelleisenbahn	muss	fähig sein	eine UART-Schnittstelle für die Fahrzeugantenne	bereitzustellen	
REQ-M5	Mikrocontroller	ETCS-Modelleisenbahn	muss	fähig sein	mit den 12V der Spannungsversorgung	zu funktionieren	
REQ-M6	Mikrocontroller	ETCS-Modelleisenbahn	wird	fähig sein	später als das Kommunikationsmodul	zu starten	Boot-Up nach den anderen Modulen, um Fehler zu reduzieren. Realisierbar durch Hardware
REQ-E1	Eisenbahn-Anwendung	ETCS-Modelleisenbahn	muss	fähig sein	die Fahrzeugantenne	zu initialisieren	
REQ-E2	Eisenbahn-Anwendung	ETCS-Modelleisenbahn	wird	fähig sein	das Kommunikationsmodul	zu initialisieren	
REQ-E3	Eisenbahn-Anwendung	ETCS-Modelleisenbahn	muss	fähig sein	fortlaufend die Position	zu bestimmen	
REQ-E4	Eisenbahn-Anwendung	ETCS-Modelleisenbahn	muss	fähig sein	fortlaufend die Geschwindigkeit	zu bestimmen	
REQ-E5	Eisenbahn-Anwendung	ETCS-Modelleisenbahn	muss	fähig sein	die Geschwindigkeit des Motors	zu regeln	
REQ-E6	Eisenbahn-Anwendung	ETCS-Modelleisenbahn	muss	fähig sein	Balisendaten	zu empfangen	

Weiter auf der nächsten Seite

Tabelle A.1: Anforderungen

ID	Anforderungstitel	System	Priorität	Art der Funktionalität	Objekt	Funktionalität	Parameterwert/ Hinweis
REQ-E7	Eisenbahn-Anwendung	ETCS-Modelleisenbahn	muss	fähig sein	Balisendaten	auszuwerten	Positionskorrektur durchführen
REQ-E8	Eisenbahn-Anwendung	ETCS-Modelleisenbahn	muss	fähig sein	Wegpunktdaten	zu empfangen	
REQ-E9	Eisenbahn-Anwendung	ETCS-Modelleisenbahn	muss	fähig sein	Wegpunktdaten	auszuwerten	
REQ-E10	Eisenbahn-Anwendung	ETCS-Modelleisenbahn	muss	fähig sein	Fahrbefehle	zu empfangen	
REQ-E11	Eisenbahn-Anwendung	ETCS-Modelleisenbahn	muss	fähig sein	Fahrbefehle	auszuwerten	
REQ-E12	Eisenbahn-Anwendung	ETCS-Modelleisenbahn	muss	fähig sein	Geschwindigkeit an Fahrbefehl	anzupassen	
REQ-E13	Eisenbahn-Anwendung	ETCS-Modelleisenbahn	muss	fähig sein	Position fortlaufend an den Server	zu melden	Alle 5 Sekunden
REQ-E14	Eisenbahn-Anwendung	ETCS-Modelleisenbahn	muss	fähig sein	Geschwindigkeit fortlaufend an den Server	zu melden	Alle 5 Sekunden
REQ-E15	Eisenbahn-Anwendung	ETCS-Modelleisenbahn	muss	fähig sein	parallele Aufgaben	abzuarbeiten	Ohne durch eine Aufgabe eine andere zu blockieren.
REQ-E16	Eisenbahn-Anwendung	ETCS-Modelleisenbahn	könnte	fähig sein	Entwicklerdaten	zu versenden	

B Kommunikationsprotokoll

Das für die Arbeit entwickelte Kommunikationsprotokoll befindet sich als schreibgeschützte PDF-Datei im Ordner „Kommunikationsprotokoll“ der dieser Arbeit beigelegten CD. Eine Arbeitsversion im DOCX-Format befindet sich im Projekt-Ordner und auf dem Gitlab-Server.
Zusätzlich folgt auf den nächsten Seiten die nach dieser Arbeit abgeschlossene Version 0.1.

Train Control Protocol HAW-Hamburg (TCPHH)

1. Version: 0.1 – Final

Versionsnotizen:

Die Version des hier niedergeschriebenen Protokolls ist die für die erste Implementierung genutzte. Das Protokoll ist bei weitem noch nicht ausgereift oder fertig, im Kapitel 8 gibt es eine Tabelle mit Punkten, die noch eingebaut oder erledigt werden müssen.

Gestrichene Features/Befehle/Kapitel sind während der Entwicklung der Version 0.1 angedacht aber nicht implementiert worden. Diese müssen vor Version 1.0 implementiert oder entfernt werden.

2. Aufbau des Telegrams

Genutzte UART-Übertragungseinstellungen:
Baudrate: 38400
Frameeinstellung: 8N1

Die Telegramme für die ETCS-Simulation sind alle nach dem gleichen Muster aufgebaut.

Start	Reserviert	Länge	Befehl	Daten	Checksumme	Ende
1 Byte	1 Byte	1 Byte	1 Byte	n Byte	1 Byte	1 Byte

Start: Immer 0xFF, kündigt den Beginn eines Telegrams an.
Reserviert: Immer 0x00, für späteren Gebrauch reserviert.
Länge: Enthält die Länge der Payload (Befehl + Länge) in Byte.
Daten: Die Daten entsprechend der Aufgabe, n=Länge-1.
Checksumme: Enthält eine Checksumme über die Bestandteile „Reserved“ bis zum Ende der „Daten“, bitweise addiert, auf ein Byte reduziert.
Ende: Immer 0xEE, zeigt das Ende der Nachricht an.

3. Übersicht über die Befehle

Code	Richtung Zug --- RBC	Befehl	Beschreibung
0x50	→	Request WPP	Fragt den Server nach einen Waypoint Plan
0x51	←	Waypoint Plan	Übermitteln des Streckenplans
0x52	→	Request MA	Fragt den Server nach einem Fahrbefehl
0x53	←	Movement Authority	Übermitteln der MA
0x54	→	Train Report	Geschwindigkeit und Position übermitteln
~~**0x55**~~	→	~~Debug Data~~	~~Enthält diverse Entwicklungsdaten~~
~~**0x61**~~	→	~~OK~~	~~Nachricht vollständig empfangen (0xFF bis 0xEE) & Checksumme der letzten Nachricht richtig~~
~~**0x62**~~	→	~~Fehler~~	~~Falsches Ende (nicht 0xEE) oder Checksumme der letzten Nachricht falsch~~

4. Waypoint Plan (WPP)

Der Zug startet die Übermittlung des Waypoint Plans durch eine entsprechende Anfrage beim Server.

4.1. Request WPP

Befehl:

Länge	0x01
Befehl/Antwort	0x50
Daten	0 Byte (Länge-1)

Antwort: Keine, die Liste mit den Wegpunkten wird direkt im Anschluss geschickt.

Beispiel Anfrage:
FF 00 01 50 51 EE --- Frage nach einer Wegpunktliste

4.2. Waypoint Plan

Befehl:

Länge	n+1 Bytes – Entsprechend der Länge der Liste
Befehl/Antwort	0x51
Daten n	Einträge * (7 Byte + 4 Byte) = Größe des WPP

Beispiel Liste:

Eintrag Index	Balisen ID (7 Byte)	Position (laufender Millimeter) (4 Byte, float)	Länge in Bytes
1	04 08 7A 04 04 04 7E	0 mm == 00 00 00 00	11
2	04 08 7A 04 04 04 7F	1000 mm == 44 7A 00 00	22
3	04 08 7A 04 04 04 80	5000 mm == 45 9C 40 00	33
4	04 08 7A 04 04 04 81	10.000 mm == 46 1C 40 00	44
5	04 08 7A 04 04 04 82	50.000 mm == 47 43 50 00	55
6	04 08 7A 04 04 04 83	100.000 mm == 47 C3 50 00	66
7	04 08 7A 04 04 04 84	500.000 mm == 48 F4 24 00	77
8	04 08 7A 04 04 04 85	1.000.000 mm == 49 74 24 00	88

Notiz: Position in mm
Die Übertragung findet im LSB-Modus statt. (z.B. 1000mm = 00 00 7A 44)

Beispiel Nachricht (2 Einträge):

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
FF	00	17	51	04 08 7A 04 04 04 7E	00 00 00 00	04 08 7A 04 04 04 7F	00 00 7A 44	47	EE

Legende:
Waypoint Plan aus 2 Einträgen, Länge 23 (1 Byte Befehl + 22 Byte Daten = 23 = 0x17)

1	Start	**6**	1. Position (0 mm)
2	Reserved	**7**	2. Balisen ID
3	Länge	**8**	2. Position (1000 mm)
4	Befehl	**9**	Checksumme
5	1. Balsisen ID	**10**	Ende

5. Movement Authority

5.1. Request MA

Befehl:

Länge	0x01
Befehl/Antwort	0x52
Daten	Keine

Antwort: Keine, die Liste mit den Wegpunkten wird direkt im Anschluss geschickt.

Beispiel Anfrage:
FF 00 01 52 53 EE --- Frage nach einer Fahrbefehlliste

5.2. Movement Authority

Befehl:

Länge	n + 1 Byte
Befehl/Antwort	0x53
Daten n	Einträge * (4 Byte + 4 Byte) = Größe der MA

Beispiel Liste:

Eintrag Index	Position/Laufender mm (4 Byte, float)	Geschwindigkeit (in mm/s) (4 Byte, float)	Ges. Bytes
1	50.000 mm == 47 43 50 00	70 mm/s == 42 8C 00 00	8
2	100.000 mm == 47 C3 50 00	120 mm/s == 42 F0 00 00	16
3	150.000 mm == 48 12 7C 00	70 mm/s == 42 8C 00 00	24
4	200.000 mm == 48 43 50 00	30 mm/s == 41 F0 00 00	32
5	350.000 mm == 48 AA E6 00	50 mm/s == 42 48 00 00	40
6	500.000 mm == 48 F4 24 00	120 mm/s == 42 F0 00 00	48

Leseweise: Die nächsten 50.000 Millimeter fahre 70 mm/s, dann fahre weitere 50.000 Millimeter mit 120mm/s (Nach ETCS-Spezifikation, Subset 026, Abschnitt 3.8.1 https://www.era.europa.eu/content/set-specifications-3-etcs-b3-r2-gsm-r-b1)

Beispiel Nachricht:

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
FF	00	11	53	00 50 43 47	00 00 8C 42	00 50 C3 47	00 00 F0 42	98	EE

Legende:
Movement Authority aus 2 Einträgen, Länge 17 (1 Byte Befehl + 16 Byte Daten = 17 = 0x11)

1	Start	**6**	1. Geschwindigkeit (70 mms/s)
2	Reserved	**7**	2. Wegpunkt (100.000 mm)
3	Länge	**8**	2. Geschwindigkeit (120 mm/s)
4	Befehl	**9**	Checksumme
5	1. Wegpunkt (50.000 mm)	**10**	Ende

6. Train Report

Befehl:

Länge	0x09
Befehl/Antwort	0x54
Daten	8 Byte – 4 Byte(float) Geschwindigkeit, 4 Byte(float) Position

Beispiel Nachricht:

1	2	3	4	5	6	7	8
FF	00	09	54	00 00 C8 42	00 00 48 43	98	EE

Legende:
Train Report mit 2 Einträgen, Länge 9 (1 Byte Befehl + 2 * 4 Byte = 9 = 0x09)

1	Start	**6**	Geschwindigkeit (100 mms/s)
2	Reserved	**7**	Position (200 mm)
3	Länge	**8**	Checksumme
4	Befehl	**9**	Ende

~~7. OK/Fehler~~

~~Für den Fall, dass eine Bestätigung der letzten Nachricht nötig ist (siehe Sequenzdiagramm), gibt es zwei extra Befehle. Diese enthalten keine Daten (Länge = 1) und teilen nur das Ergebnis der letzten Checksummenberechnung mit.~~

8. ToDo

Debugdata	Enthält diverse Daten, die beim Entwickeln und Debuggen wichtig sein könnten: global_position, global_speed, global_speed_target, waypintPlan.entrys, etc.
OK / Fehler	Könnte das erneute Senden eines misslungenen Transfers auslösen. (Siehe Kapitel 6)
Vergrößern der Länge auf 2 oder 3 Byte	Um längere Listen verschicken zu können.
Überarbeiten Movement Authority und Waypoint	Ist: MA enthält Wegpunkt und Geschwindigkeitslimit, WP nur Position auf der Strecke Muss: MA enthält Wegpunkt und optionales Geschwindigkeitslimit, WP enthält Position auf der Stecke und Streckengeschwindigkeitslimit

9. Versionshistorie

Version	Datum	Änderungen	Autor
0.1-RC1	12.12.2020	Erste Version	R.Willing
0.1-RC2	13.01.2021	Hübsch machen für die Übergabe	R.Willing
0.1-RC3	24.01.2021	Baudrate und Frame-Charakteristik eingefügt	R.Willing
0.1-Final	30.01.2021	Finale 0.1 Version	R.Willing

C NUCLEO-F303K8-Pinout

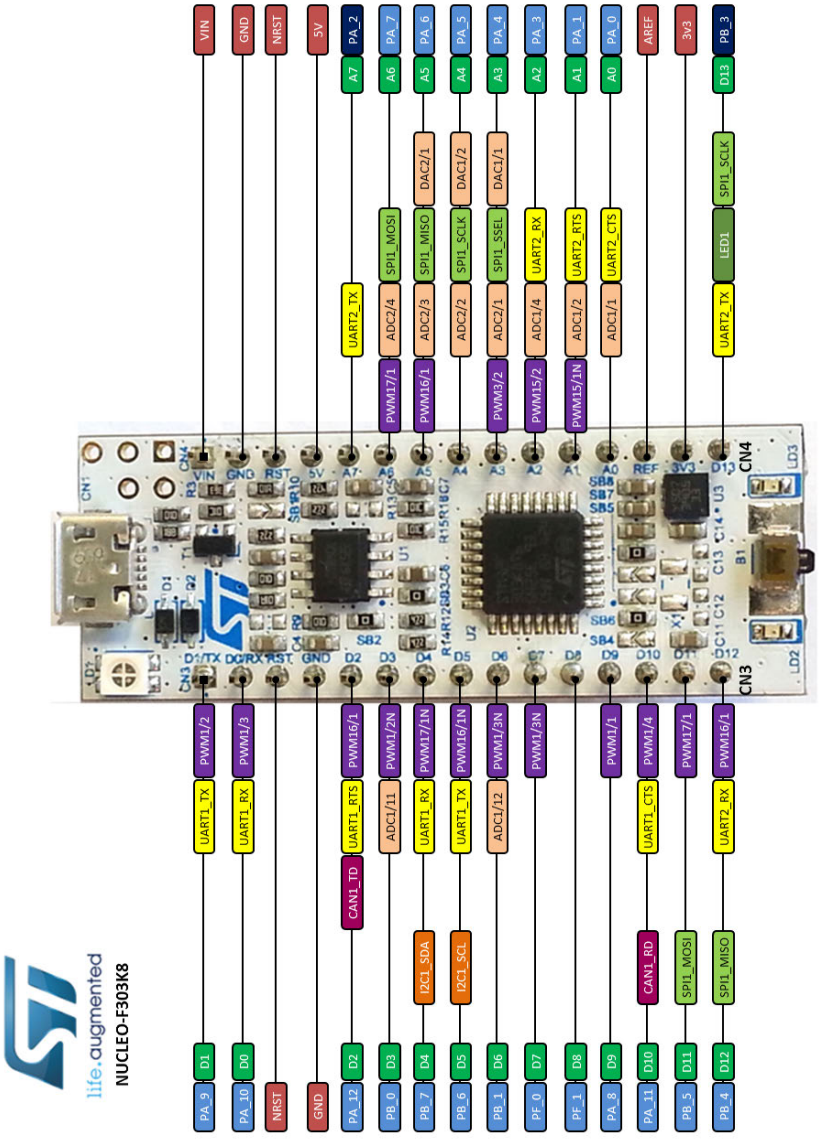

Abb. C.1: Pinout des genutzten Entwicklerboards NUCLEO-F303K8 [11]

D Funktionsabläufe

Auf den folgenden Seiten werden die Funktionsabläufe großer Funktionen angehängt.

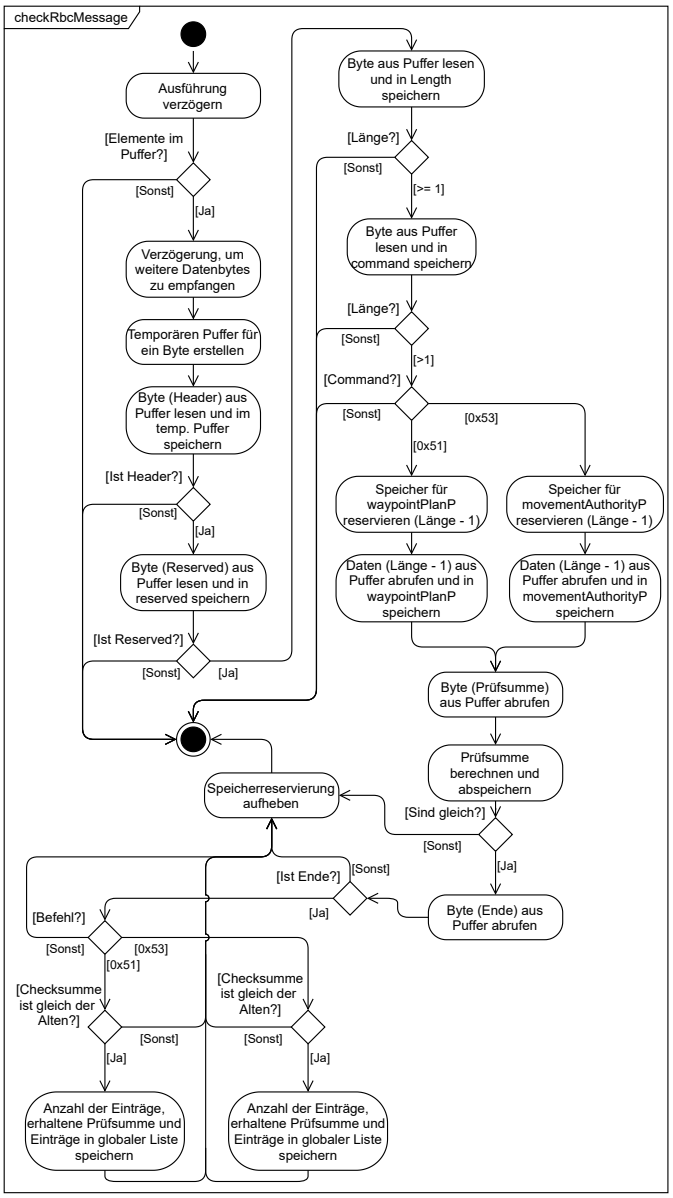

Abb. D.1: Ablauf der Aufgabe checkRbcMessage

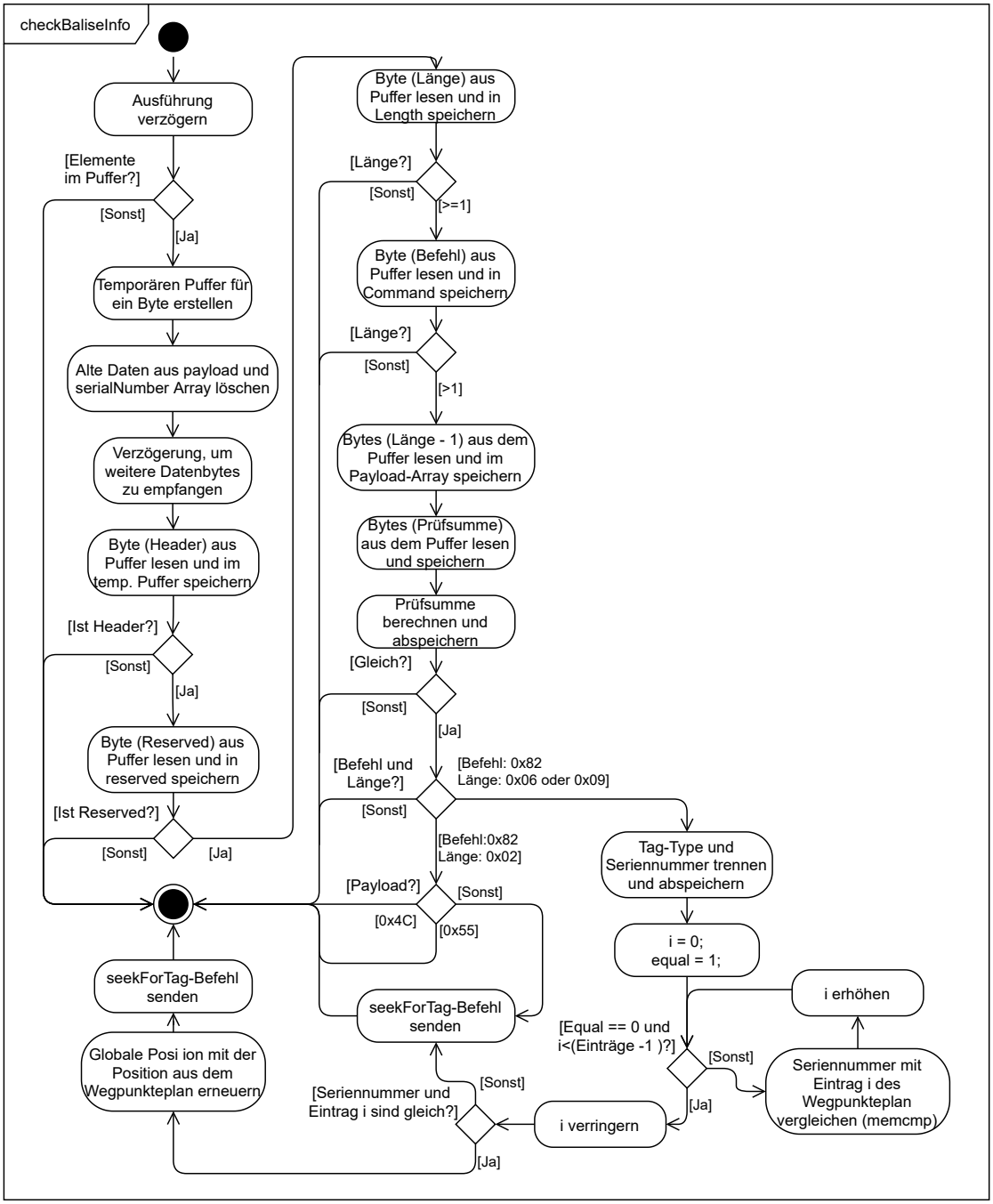

Abb. D.2: Ablauf der Aufgabe checkBaliseInfo

E Testfälle

E.1 RBC-Server

Die an den RBC-Server gerichteten Testfälle sind die folgenden:

Tabelle E.1: Testfall S0.1: Fahrzeugantennen Simulation

Testfall-ID:	S0.1	Name:	Fahrzeugantennen Simulation
Vorbedingung:	Keine.		
Prüfmethode:	Automatisch.		
Ablauf:	Die Simulation antwortet auf vordefinierte Telegramme mit einer festen Antwort: 1. „Reset wird“ empfangen und mit „Versionsnummer“ beantwortet. 2. „Set Antenna“ Power On wird empfangen und mit „RF-Field Switched On“ beantwortet. 3. „Seek For Tag“ wird empfangen und mit „Command in Progress“ beantwortet.		
Resultat:	Wenn keine Fehler auftreten, wird der Code weiter ausgeführt. In der Terminal-Ausgabe werden die abgearbeiteten Schritte aufgeführt und die Initialisierung ist beendet, sobald `-- RFID Initialiation End --` erscheint.		
Varianten:	–		
Limit:	Es werden in der Initialisierung nur diese drei Befehle interpretiert.		
Abgedeckte Anforderungen:	REQ-E1		
Bemerkung:	Testfall-ID ist S0.1, da es keine Anforderung laut Anforderungsanalyse ist.		

Tabelle E.2: Testfall S0.2: Kommunikationsmodul Simulation

Testfall-ID:	S0.2	Name:	Kommunikationsmodul Simulation
Vorbedingung:	Fahrzeugantenne initialisiert.		
Prüfmethode:	Automatisch.		
Ablauf:	Der Ablauf würde sich an der Abbildung 4.12 orientieren.		
Resultat:	Wenn keine Fehler auftreten, wird der Code weiter ausgeführt.		
Varianten:	–		
Limit:	Nicht implementiert, daher nicht funktionsfähig.		
Abgedeckte Anforderungen:	REQ-E1		
Bemerkung:	Testfall-ID ist S0.2, da es keine Anforderung laut Anforderungsanalyse ist.		

Tabelle E.3: Testfall S1: Übertragen von WPP und MAL

Testfall-ID:	S1	Name:	Übertragen von WPP und MAL
Vorbedingung:	Initialisierung durchlaufen.		
Prüfmethode:	Automatisch.		
Ablauf:	Der Server unterscheidet zwischen WPP- und MAL-Anfrage und handelt wie folgt: WPP: 1. Wegpunkteplan einlesen 2. Wegpunkteplan versenden MAL: 1. Fahrbefehlsliste einlesen 2. Fahrbefehlsliste versenden		
Resultat:	Im Terminal erscheint `-- MAL send, XX Entrys` oder `-- WPP send, XX Entrys`, wobei `XX` der Anzahl der versendeten Einträge entspricht.		
Varianten:	Verschieden lange Listen: WPP: 8, 17, 20, 23, 24 MAL: 7, 12, 25, 30, 31, 32		
Limit:	Die Länge der versendbaren Liste ist derzeit limitiert. Limit WPP: 23 Einträge Limit MAL: 31 Einträge		
Abgedeckte Anforderungen:	REQ-R1–R4		
Bemerkung:	Durch die Speicherbegrenzung von einem Byte für die Länge im Telegramm ist die Gesamtlänge begrenzt.		

Tabelle E.4: Testfall S2: Empfang und Parsen des Train Reports

Testfall-ID:	S2	Name:	Empfang und Parsen des Train Reports
Vorbedingung:	Initialisierung durchlaufen.		
Prüfmethode:	Automatisch.		
Ablauf:	Wenn ein Zugreport eintrifft, wird dieser in seine Bestandteile aufgespalten und im Terminal ausgegeben.		

Resultat:	In dem Terminal erscheint:
	`-- Parse Train Report --`
	`-- Start Train Report Parsing --`
	`-- Parsing Train Report --`
	`Result: Speed: 0.0, Location: 0.0`		

Varianten:	Durch den Taster kann der Inhalt des Train-Reports verändert werden.		
Limit:	Die Änderung des Inhalts funktioniert nur, solange die Eisenbahnsoftware läuft.		
Abgedeckte Anforderungen:	REQ-R5–R6		
Bemerkung:	Der Taster ist nicht entprellt und die Resultate ändern sich durch das Auslösen mehrerer Interrupts nicht vorhersehbar.		

Tabelle E.5: Testfall S3: Empfang von Debugdaten

Testfall-ID:	S3	Name:	Empfang von Debugdaten
Vorbedingung:	Initialisierung durchlaufen.		
Prüfmethode:	Automatisch.		
Ablauf:	Parallel zum Empfang des Train Reports können weitere Debugdaten empfangen werden.		
Resultat:	Im Terminal werden die Daten angezeigt.		
Varianten:	–		
Limit:	Nicht implementiert, daher nicht funktionsfähig.		
Abgedeckte Anforderungen:	REQ-R7		
Bemerkung:	Debugdaten haben in dieser Arbeit keine hohe Priorität und wurden deshalb nicht implementiert.		

E.2 Eisenbahnsoftware

Folgende Testfälle werden für die Eisenbahnsoftware definiert:

Tabelle E.6: Testfall E1: Initialisierung

Testfall-ID:	E1	Name:	Initialisierung
Vorbedingung:	Keine.		
Prüfmethode:	Manuell mit HTerm und automatisch mit dem Python-Skript.		
Ablauf:	1. „Reset“ wird empfangen und mit „Versionsnummer“ (HTerm: 1. Response) beantwortet. 2. „Set Antenna Power On“ wird empfangen und mit „RF-Field Switched On“ (HTerm: 2. Response) beantwortet. 3. „Seek For Tag“ wird empfangen und mit „Command In Progress“ (HTerm: 3. Response) beantwortet.		
Resultat:	Nach jeder Sequenz ändert sich das empfangene Telegramm. Sobald die Initialisierung erfolgreich ist, blinkt die LED auf dem Board im 1 Hz Takt. Das Skript zeigt die Schritte der Initialisierung an und teilt mit, ob diese erfolgreich waren.		
Varianten:	HTerm: Einzelne Sequenzen oder gebündelte Sequenzen.		
Limit:	In HTerm wird nur der Bytecode der empfangenen Sequenz angezeigt. Das Kommunikationsmodul wird von diesem Testfall nicht erfasst, da dieses noch nicht implementiert ist.		
Abgedeckte Anforderungen:	REQ-E1–E2		
Bemerkung:	Mit dem Datenblatt des SM130 [42] können die empfangenen Sequenzen identifiziert werden. Die Initialisierung des Kommunikationsmoduls ist nicht implementiert, siehe Tabelle E.2.		

Tabelle E.7: Testfall E2: Empfange WPP

Testfall-ID:	E2	Name:	Empfange WPP
Vorbedingung:	Initialisierung durchlaufen.		
Prüfmethode:	Manuell mit HTerm und automatisch mit dem Python-Skript.		
Ablauf:	1. Eisenbahnsoftware fragt Server oder HTerm nach Wegpunktliste. 2. Server oder HTerm schickt Wegpunktliste. 3. Eisenbahnsoftware verarbeitet und speichert Daten. 4. Eisenbahnsoftware hört auf, nach Wegpunktliste zu fragen.		
Resultat:	Die Variable `waypointPlan.entrys` enthält eine Zahl größer 0. Die Eisenbahnsoftware fragt nicht weiter nach der Wegpunkteplan und weder HTerm noch das Skript empfangen weitere Anfragen.		
Varianten:	Listen mit unterschiedlich vielen Einträgen (8, 17, 20, 23 und 24). Unterschiedliche Reihenfolge: WPP und MAL. Mit bereits vorhandener WPP.		
Limit:	Die Länge der versendbaren Liste ist derzeit limitiert. Limit: 23 Einträge		
Abgedeckte Anforderungen:	REQ-E8–E9		
Bemerkung:	Durch die Speicherbegrenzung von einem Byte für die Länge im Telegramm ist die Gesamtlänge begrenzt.		

Tabelle E.8: Testfall E3: Empfange MAL

Testfall-ID:	E3	Name:	Empfange MAL
Vorbedingung:	Initialisierung durchlaufen.		
Prüfmethode:	Manuell mit HTerm und automatisch mit dem Python-Skript.		
Ablauf:	1. Eisenbahnsoftware fragt Server oder HTerm nach Fahrbefehlsliste. 2. Server oder HTerm schickt Fahrbefehlsliste. 3. Eisenbahnsoftware verarbeitet und speichert Daten. 4. Eisenbahnsoftware hört auf, nach Fahrbefehlsliste zu fragen.		
Resultat:	Die Variable `movementAuthorityList.entrys` enthält eine Zahl größer 0. Die Eisenbahnsoftware fragt nicht weiter nach der Fahrbefehlsliste und weder HTerm noch das Skript empfangen weitere Anfragen.		
Varianten:	Listen mit unterschiedlich vielen Einträgen (7, 12, 25, 30, 31 und 32). Unterschiedliche Reihenfolge: WPP und MAL. Mit bereits bestehender MAL.		
Limit:	Die Länge der versendbaren Liste ist derzeit limitiert. Limit: 31 Einträge		
Abgedeckte Anforderungen:	REQ-E10–E11		
Bemerkung:	Durch die Speicherbegrenzung von einem Byte für die Länge im Telegramm ist die Gesamtlänge begrenzt.		

Tabelle E.9: Testfall E4: Balisendaten empfangen und verarbeiten

Testfall-ID:	E4	Name:	Balisendaten empfangen und verarbeiten
Vorbedingung:	Initialisierung durchlaufen, WPP und MAL empfangen.		
Prüfmethode:	Manuell mit HTerm unde CubeIDE.		
Ablauf:	1. Die Fahrzeugantenne oder HTerm übermitteln eine Balisen-ID. 2. Eisenbahnsoftware verarbeitet die Nachricht. 3. Eisenbahnsoftware schlägt die Balisen-ID in dem WPP nach. 4. Wenn vorhanden wird Position angepasst. 5. Eisenbahnsoftware sendet „Seek For Tag“ an Server oder HTerm.		
Resultat:	Die Variable `position_glonal` ändert sich auf den durch die Balisen-ID und WPP definierten Wert. Ein Telegramm „Seek For Tag“ wird von HTerm empfangen.		
Varianten:	Verschiedene Positionen: Balise 1 Position = 0, Balise 2 = 1000, Balise 3 = 5000, Balise 4 = 10000		
Limit:	–		
Abgedeckte Anforderungen:	REQ-E6–E7		
Bemerkung:	Ohne WPP und MAL wird keine Bewegung ausgeführt, wodurch auch keine Balise empfangen werden kann. Wenn doch eine Balise empfangen wird, enthält der WPP keinen Eintrag zur ID.		

Tabelle E.10: Testfall E5: Soll-Geschwindigkeit anpassen

Testfall-ID:	E5	Name:	Soll-Geschwindigkeit anpassen
Vorbedingung:	Initialisiert durchlaufen.		
Prüfmethode:	Manuell mit CubeIDE und HTerm.		
Ablauf:	1. Wenn sowohl WPP als auch MAL Einträge enthalten, wird die Soll-Geschwindigkeit auf den Wert des aktuell geltenden Fahrbefehls gesetzt. 2. Wenn eine Liste leer ist, wird die Soll-Geschwindigkeit auf Null gesetzt. 3. Wenn die Position sich ändert und die Modelleisenbahn in den nächsten Fahrbefehl fährt, wird die Geschwindigkeit entsprechend angepasst. 4. Wenn die Position größer ist als der höchste Fahrbefehl, wird die Geschwindigkeit auf Null gesetzt. Die Position kann durch das Versenden der Balisen mit HTerm verändert werden oder durch das Ändern der Variable `position_global` direkt in CubeIDE.		
Resultat:	Die Variable `speed_global_target` nimmt den Wert des aktuell geltenden Fahrbefehls an.		
Varianten:	Senden verschiedener Balisen-IDs mit HTerm und damit Ändern des Fahrbefehls und der Geschwindigkeitsvorgabe.		
Limit:	–		
Abgedeckte Anforderungen:	REQ-E12		
Bemerkung:	Nutzen der Balisen-Sequenzen in HTerm nur möglich, wenn Testfall E4 funktional.		

Tabelle E.11: Testfall E6: Duty-Cycle anpassen

Testfall-ID:	E6	Name:	Duty-Cycle anpassen
Vorbedingung:	Initialisierung durchlaufen, WPP und MAL empfangen, Soll-Geschwindigkeit nicht 0.		
Prüfmethode:	Manuell mit HTerm, CubeIDE und PulseView und automatisch mit Python-Skript und PulseView.		
Ablauf:	1. Sobald die Bahn WPP und MAL empfangen hat, ändert sich die Soll-Geschwindigkeit auf den ersten Fahrbefehl. 2. Sobald sich der zuständige Fahrbefehl ändert, muss der Duty-Cycle entsprechend angepasst werden. Der zuständige Fahrbefehl wird durch das Ändern der Postion bewerkstelligt. Entweder durch das Versenden von Balisen-IDs mit HTerm oder durch das direkte Ändern der Variable `position_global` mithilfe von CubeIDE.		
Resultat:	Registerwert `htim1.Instance->CCR1` ändert sich und dadurch ändert sich der Duty-Cycle des PWM-Signals.		
Varianten:	Je nach Position der Eisenbahn ändert sich die Geschwindigkeitsvorgabe und damit der Duty-Cycle.		
Limit:	Automatisch lässt sich nur der Duty-Cycle des ersten Fahrbefehls mit PulseView erfassen, da keine Positionsänderung auftritt.		
Abgedeckte Anforderungen:	REQ-E5		
Bemerkung:	Das erzeugte PWM inkl. Duty-Cycle kann am Pin D9 des Entwicklerboards abgegriffen werden.		

Tabelle E.12: Testfall E7: Train Report senden

Testfall-ID:	E7	Name:	Train Report senden
Vorbedingung:	Initialisierung durchlaufen.		
Prüfmethode:	Manuell mit HTerm und automatisch mit dem Python-Skript.		
Ablauf:	Der Train Report wird automatisch alle 5 Sekunden versendet, sobald alle Vorbedingungen erfüllt sind.		
Resultat:	Es wird eine Sequenz versendet, die dem Befehl „Train Report“ aus dem Protokoll entspricht. Im Skript werden die Werte geparst und angezeigt.		
Varianten:	–		
Limit:	–		
Abgedeckte Anforderungen:	REQ-E13–E14		
Bemerkung:	–		

Tabelle E.13: Testfall E8: Fortlaufende Positionsbestimmung

Testfall-ID:	E8	Name:	Fortlaufende Positionsbestimmung
Vorbedingung:	Initialisierung durchlaufen, Taster an 5 V und D10 angeschlossen.		
Prüfmethode:	Manuell mit HTerm und CubeIDE oder automatisch mit dem Python-Skript.		
Ablauf:	Um eine Achsenrotation zu simulieren, wird der Taster gedrückt.		
Resultat:	Pro Druck des Tasters sollte der Wert des Achsenumfangs (≈314) auf die Position aufaddiert werden. Diese kann mit CubeIDE aus der Variable `position_global` ausgelesen werden oder durch den Zugreport im Skript. In HTerm ändern sich die Bytes 9–12 im Zugreport-Telegramm.		
Varianten:	Variieren im Tempo des Drückens vom Tasters.		
Limit:	Der Taster ist nicht entprellt und lässt keinen einzelnen Druck zu. Ein zu schnelles Drücken kann zu dem Blockieren des Programms führen.		
Abgedeckte Anforderungen:	REQ-E3		
Bemerkung:	–		

Tabelle E.14: Testfall E9: Fortlaufende Geschwindigkeitsbestimmung

Testfall-ID:	E9	Name:	Fortlaufende Geschwindigkeitsbestimmung
Vorbedingung:	Initialisierung durchlaufen, Taster an 5 V und D10 angeschlossen.		
Prüfmethode:	Manuell mit CubeIDE und HTerm oder automatisch mit dem Python-Skript.		
Ablauf:	Um eine Achsenrotation zu simulieren, wird der Taster gedrückt.		
Resultat:	Durch Betätigen des Tasters wird die Geschwindigkeitsberechnung ausgelöst und die Variable `speed_glonbal` ändert sich. Diese kann mit CubeIDE aus der Variable `position_global` ausgelesen werden oder durch den Zugreport im Skript. In HTerm ändern sich die Bytes 5–8 im Zugreport Telegramm.		
Varianten:	Variieren im Tempo des Drückens vom Tasters.		
Limit:	Der Taster ist nicht entprellt und lässt keinen einzelnen Druck zu. Ein zu schnelles Drücken kann zu dem Blockieren des Programms führen.		
Abgedeckte Anforderungen:	REQ-E4		
Bemerkung:	–		

Tabelle E.15: Testfall E10: Paralleles Abarbeiten der Aufgaben

Testfall-ID:	E10	Name:	Paralleles Abarbeiten der Aufgaben
Vorbedingung:	Initialisierung durchlaufen.		
Prüfmethode:	Manuell mit HTerm und automatisch mit dem Python-Skript.		
Ablauf:	Sobald die Initialisierung durchlaufen ist, fängt das parallele, durch FreeRTOS gesteuerte Arbeiten an.		
Resultat:	Es wird eine Sequenz versendet, die dem Befehl „Train Report“ aus dem Protokoll entspricht. Im Skript werden die Werte geparst und angezeigt.		
Varianten:	–		
Limit:	Durch die Interrupt-Routinen kann es zu einem Unterbrechen des parallelen Arbeitens kommen. Interrupts laufen außerhalb der Kontrolle von FreeRTOS.		
Abgedeckte Anforderungen:	REQ-E15		
Bemerkung:	–		

Tabelle E.16: Testfall E11: Erstellen und Versenden von Debugdaten

Testfall-ID:	E11	Name:	Erstellen und Versenden von Debugdaten
Vorbedingung:	Initialisierung durchlaufen.		
Prüfmethode:	Manuell mit HTerm und automatisch mit dem Python-Skript.		
Ablauf:	Sobald die Initialisierung durchlaufen ist, sendet die Eisenbahnsoftware Entwicklerdaten parallel zum Zugreport.		
Resultat:	Es wird eine Sequenz versendet, die dem Befehl „Debug Data“ aus dem Protokoll entspricht. Im Skript werden die Werte geparst und angezeigt.		
Varianten:	–		
Limit:	Das Versenden von Debugdaten wurde nicht implementiert.		
Abgedeckte Anforderungen:	REQ-E16		
Bemerkung:	–		

F Projektordner

Der vollständige Projektordner befindet sich im Ordner „Projekt-Ordner“ auf der beigelegten CD, inklusive vollständigem Python-Skript, HTerm-Konfigurationen und der Arbeitsversion des Kommunikationsprotokolls als DOCX-Datei. Ebenfalls sind zwei Internetverknüpfungen zu den „master“- und „devel“-Repositories des Projekts enthalten.

G Dokumentation

Die Datenblätter, Referenz- und andere Handbücher befinden sich auf der beigelegten CD in dem Ordner „Dokumentation“. In Tabelle G.1 befindet sich eine Auflistung der enthaltenen Dokumente.

Tabelle G.1: Liste aller Datenblätter

Name	Beschreibung	Hersteller
HC-08 User Manual (v2.0, v2.4, v3.1)	Bluetooth Modul (3 Versionen)	HC Tech
TLE5205-INF H-Bridge	H-Brücke	Infineon Technologies AG
TTL-232R	FTDI Adapter	Future Technology Devices International Ltd.
ds_SM130	RFID Modul Datenblatt	SonMicro Electronics Ltd.
um_sm130_a2	RFID Modul Anleitung	SonMicro Electronics Ltd.
W7001	W7001 Antennen Datenblatt	Pulse Electronic
Nucleo-F303K8 - Programming Manual	Entwicklerboard Programmieranleitung	STMicroelectronics N.V.
Nucleo-F303K8 - User Manual	Entwicklerboard Anleitung	STMicroelectronics N.V.
STM32F303K8 - Chip Manual	Mikrocontroller Datenblatt	STMicroelectronics N.V.
STM32F303K8 - Reference Manual	Mikrocontroller Referenzhandbuch	STMicroelectronics N.V.

Weiter auf der nächsten Seite

Name	Beschreibung	Hersteller
STM32F3 HAL Doku	Mikrocontroller HAL-Dokumentation	STMicroelectronics N.V.

Glossar

2oo2 Auch: Zwei aus Zwei. Ein Sicherheitssystem, bei dem die zweifach ausgelegte Komponente das gleiche Signal melden muss um eine Reaktion zu ermöglichen, bzw. keine Warnung auszulösen. Als Beispiel: Zwei Sensoren müssen dieselbe Geschwindigkeit melden, damit eine fehlerhafte Messung aufgrund eines Defekts vermieden werden kann. (Analog 2oo3: Zwei aus Drei)

Application Programming Interface Deutsch: Programmierschnittstelle. Ermöglicht den einfachen Zugriff auf vorgefertigte Funtkionen.

arc42 Ein offenes und frei nutzbares Architektur-Template für die Erstellung von Software-Architekturen.

Board Support Package Bestandteil von plattformunabhängigen Betriebssystemen. Dient als Schnittstelle der OS-Funktionen zur Hardware des spezifischen Entwicklungsboards.

Bremskurve Die Bremskurve soll den Geschwindigkeitsverlust bei einer Bremsung über die Zeit darstellen. Sie wird verwendet, um den Punkt zu bestimmen, an dem der Stillstand eintritt.

Browser-Engine Ein Programm, dass das Ausführen und Anzeigen von Webseiten übernimmt.

Crocodile Zugbeeinflussungssystem aus Frankreich.

Crosstalk Wenn eine Antenne die Nachrichten des Nachbargleises empfängt.

Driver Machine Interface Bedien- und Anzeigeelemente für den Zugführer im ETCS-System.

Durchrutschen Im Eisenbahnbetrieb wird das Überfahren eines Haltesignals beim Bremsforgang durchrutschen bezeichnet. Deshalb befinden sich hinter Signalen auch der sogenannte Durchrutschweg, ein Sicherheitsbereich, in dem das Befahren trotz Haltesignal möglich ist.

Durchrutschpunkt Definiert einen Punkt bis zu dem ein Zug auch nach einem Stop-Signal durchrutschen darf.

Eclipse Open-Source IDE (Integrated Development Environment).

End of Authority Beschreibt das Enden einer Fahrterlaubnis im ETCS genannt.

End-Of-Loop-Marker Eine Balise die den Start eines Euroloops markiert.

ETCS Level Bezeichnet die Ausrüstungsstufe der Strecke im ETCS-System.

ETCS-Fahrzeugcomputer Auch European Vital Computer (EVC), ETCS-Fahrzeugcomputer, ETCS-Computer oder ETCS-Fahrzeugrechner. Kernkomponente des ETCS-Systems im Zug, wertet jegliche Informationen aus und bereitet Sensordaten zur Weitergabe auf. Berechnet urven und überwacht den Zug.

Eurobalise Teil des ETCS-Systems. Es ist ein Transponder im Gleisbett, der durch die Fahrzeugantenne ausgelesen wird.

Euroloop Teil des ETCS-Systems. Es ist eine Antenne, die bis zu 1 km im Gleisbett liegen kann und ähnlich, wie die Balise, Daten an die Fahrzeugantenne überträgt.

European Rail Traffic Management System Überbegriff von zwei (ürsprünglich drei) Komponenten zur Harmonisierung der Sicherheitssysteme im Eisenbahnverkehr.

European Traffic Management Layer Früherer Bestandteil des ERTMS. Sollte den Fahrkartenkauf und die Fahrgastinformation im Zug standarisieren.

European Train Control System Europäisches Zug-Kontroll-System. Ein System zum Reduzieren der Signale und Erweitern der Automatisierung der Bahnfahrt.

Fahrterlaubnis Auch Fahrbefehl. Englisch: Movement Authority. Die Information für den ETCS-Fahrzeugrechner, bis wohin der Zug fahren darf.

Flag Durch Ereignisse gesetzte Werte, die von anderen Funktionen abgefragt werden können.

Fragmentierung Beschreibt das Zerteilen von zusammenhängenen Daten auf mehrere Blöcke, damit diese im Speicher unzusammenhängend abgelegt werden können..

Frame Bezeichnet die übertragung einer UART-Nachricht.

FreeRTOS Ein Echtzeitbetriebssystem für den Einsatz auf einem Mikrocontroller.

Gefahrenpunkt Definiert in der Zugfahrt einen Punkt, den ein Zug nicht erreichen darf.

General Purpose Operating System Ein Betriebssystem, dass auf die Nutzung durch menschliche Anwender ausgelegt ist. z. B. Linux oder Windows.

Global System for Mobile Communications Ein System zur mobilen Kommunikation. Findet z. B. im Handy Anwendung.

Global System for Mobile Communications – Rail(way) Ein auf GSM basierendes System mit Modifikationen für die Anwendung im Eisenbahnbereich.

H-Terminal Ein kostenloses Terminal Programm für die serielle Kommunikation über COM-Ports.

H0-Spur Eine Gleisspurbreite von Modelleisenbahnen.

Handler Handler wird in der Informatik eine Funktion genannt, die eine weitere Funtkion als Parameter enthält und diese unter bestimmten Bedingungen ausführt. So führt der Interrupt-Handler die übergebene Interrupt-Routine nach dem Auftreten eines Interrupt-Signals am definierten Pin aus.

Hardware Abstraction Layer Deutsch: Hardwareabstraktionsschicht. Eine Methode den Zugriff auf die Hardwarefunktionen eines Mikrocontrollers durch das implementieren einer API zu vereinfachen.

induktive Zugsicherung Ein 1934 eingeführtes Zugbeeinflussungssystem, dass die heute noch verwendeten Gleismagneten eingeführt hat.

Integrated Development Environment Bestandteil einer Toolchain, bietet die Oberfläche und Hilfsmittel zum Schreiben von Code.

Konfigurator Bestandteil einer Toolchain. Ermöglicht das automatische Erstellen von Code, dass den SoC konfiguriert.

KVB Zugbeeinflussungssystem aus Frankreich.

Limit of Authority Beschreibt das Einschränken einer Fahrterlaubnis im ETCS genannt.

Lineside Electronic Unit Eine Einrichtung im ETCS, dass an der Bahntrasse stehende Signale in ein ETCS-verständliches Format umwandelt und an eine Transparentdatenbalise weiterreicht. Die Elektronik ist passiv und wird durch die Fahrzeugantenne mit Strom versorgt.

Linienzugbeeinflussung Eine Sicherungseinrichtung, die frühzeitig über Signalstellungen informiert, um z. B. Bremswege zu kompensieren. Resultiert auch in einer dauerhaften Geschwindigkeitsüberwachung und dem Einleiten einer Bremsung im Falle der Überschreitung.

Link Assurance Eine Methode um die bestehende Verbindung zu visualisieren und einen Abbruch der Verbindung möglichst schnell erkennbar zu machen.

Middleware Eine zusätzliche Schnittstelle zwischen Betriebssystem und Anwendung. z. B. die STMicroelectronic Implementierung von FreeRTOS in den Konfigurator.

Movement Authority Deutsch: Fahrterlaubnis. Beschreibt im ETCS-System eine für den Zug freigegebene Strecke, inkl. Streckeninformationen.

Movement Authority List Eine Liste aus Fahrbefehlen.

Mutex Bestandteil einer Technik zum Ressourcenmanagement in der Informatik, durch die eine Ressource nur einmal zeitgleich ausgeführt werden kann.

Odometrie Einrichtungen zur Geschwindigkeits- und Wegmessung.

Pugh-Matrix Eine nach Pugh benannte Konzeptauswahlanalyse. Die Matrix ist eine Entscheidungshilfe zur Bewertung verschiedener Konzepte.

Pulsweitenmodulation Englisch: Puls width modulation. Beschreibt ein Verfahren um ein Signal auf einer Frequenz zu transportieren. Die Pulsweite ist dabei veränderbar und wird auch Taktgrad genannt. Dieser Taktgrad wird in $0\,\% - 100\,\%$ angegeben.

punktförmige Zugbeeinflussung Sicherungseinrichtungen, wie Sensoren im Gleis, die beim Überfahren eines roten Signals eine Bremsung im Zug einleiten (Lokführer hat Signal übersehen). Aktuelle Version ist PZB 90 und basiert auf dem Indusi-System von 1934.

Radio Block Center Streckeneinrichtung zur Übertragung der Fahrwegsinformationen des Stellwerks über GSM-R an den Triebwagen.

Schiebe-, Vorspann- oder Zwischendienst Beschreibt wo die Lokomotive in einem Zugverbund eingesetzt wird.

Sicherheitsfahrschaltung Auch bekannt als Totmannschalter. Sorgt dafür, dass der Zug gebremst wird, sollte der Triebfahrzeugführer länger als 30 Sekunden keine Interaktion an der Bedienung durchgeführt haben.

Specific Transmission Module Bestandteil der ETCS-Einrichtung im Zug, dass die nationalen Zugbeeinflussungssysteme ins ETCS-System übersetzt.

System-on-a-Chip Deutsch: Ein-Chip-System. Als Ein-Chip-Systeme werden Systeme genannt, die mehrere Funktionen in einem Chip kombinieren. Ein Beispiel wäre der Hauptchip in einem Smartphone, dieser enthält neben der CPU und Arbeitsspeicher auch Elektronik für das entsprechende Funknetz.

Toolchain Eine Reihe an Programmen, die zum Erstellen einer SW-Lösung benötigt werden. z. B. Konfigurator + IDE + Kompilierer.

Transeuropäische Netze Ein europäisches Eisenbahnnetz.

Transparentdatenbalise Eine Eurobalise, die im Gegensatz zu der ursprünglichen Eurobalise, ein veränderbares Telegramm verschicken kann.

Unified Modeling Language Eine Modellierungssprache um Software grafisch visualisieren zu können.

Waypoint Ein Wegpunkt, markiert durch eine Balise, besteht aus einer Seriennummer und der Geschwindigkeitsvorgabe für den kommenden Streckenabschnitt.

Waypoint Plan Auch Wegpunkteplan. Eine Liste an Wegpunkten für die kommende Strecke.

Windows CE Ein von Microsoft bereitgestelltes Betriebssystem für eingebettete Systeme.

Yield Eine im Bereich des Multithreading auftauchende Aktion, die eine Ausführung des laufenden Prozesses unterbricht und am Ende der Warteschlange des Prozess-Scheduler einreiht.

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original