Bachelorarbeit

im Studiengang Mechatronik

Entwicklung eines GPS-Trackers unter Einsatz eines Low-Cost Mikroconrollers

von **Changlai Bao**

Matr.-Nr.: 6220001

vorgelegt am 29.02.2024

an der Hochschule Hamm-Lippstadt

Erstprüfer: Prof. Dr. Axel Thümmler

Zweitprüfer: Prof. Dr. Jörg Wenz

Kurzzusammenfassung

Im Zuge der rasanten technologischen Entwicklung der letzten Jahrzehnte hat sich der Einsatz von GPS-Technologie in unserem Alltag fest etabliert. Von der Navigation in unbekannten Gebieten bis hin zur Überwachung wertvoller Güter bietet die GPS-Technologie eine unverzichtbare Grundlage für eine Vielzahl moderner Anwendungen. Allerdings sind kommerzielle GPS-Tracker oft teuer und bieten nicht immer die gewünschten Funktionen.

Diese Arbeit beschäftigt sich mit der Entwicklung eines kostengünstigen GPS-Trackers auf Basis eines Low-Cost Mikrocontrollers. Die Forschung und Entwicklung, die in dieser Arbeit vorgestellt wird, basieren auf einer umfassenden Analyse vorhandener Technologien und Methoden sowie auf experimentellen Ansätzen zur Realisierung eines Prototyps. Dabei wurde besonderes Augenmerk auf die Auswahl geeigneter Hardwarekomponenten, die Optimierung der Software zur Datenverarbeitung und die Integration der verschiedenen Komponenten gelegt.

Die Ergebnisse zeigen, dass der entwickelte GPS-Tracker in Bezug auf Genauigkeit, Energieverbrauch und Kosten mit teureren kommerziellen Produkten vergleichbar kann. Darüber hinaus bietet der entwickelte GPS-Tracker eine hohe Flexibilität und Anpassungsfähigkeit, die es ermöglicht, ihn an verschiedene Anwendungsszenarien anzupassen. Die entwickelte PC-Anwendung ermöglicht es dem Benutzer, die aufgezeichneten Daten zu visualisieren und zu analysieren, was eine detaillierte Auswertung der zurückgelegten Wege ermöglicht. Damit leistet die Arbeit einen positiven Beitrag zur Entwicklung von kostengünstigen GPS-Trackern und eröffnet neue Möglichkeiten für die Anwendung dieser Technologie in verschiedenen Bereichen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkrüzungsverzeichnis

API	Application Programming Interface
CAN	Controller Area Network
CPU	Central Processing Unit
DMA	Direct Memory Access
EEPROM	Electrically Erasable Programmable Read-Only Memory
GND	Ground
GNSS	Global Navigation Satellite System
GPIO	General Purpose Input/Output
GPS	Global Positioning System
GPX	GPS Exchange Format
GUI	Graphical User Interface
HSHL	Hochschule Hamm-Lippstadt
I/O	Input/Output
I2C	Inter-Integrated Circuit
IDE	Integrated Development Environment
INT	Interrupt
ISR	Interrupt Service Routine
KML	Keyhole Markup Language
LCD	Liquid Crystal Display
MISO	Master In Slave Out
MOSI	Master Out Slave In

PC	Personal Computer
PCB	Printed Circuit Board
RAM	Random Access Memory
ROM	Read-Only Memory
RST	Reset
RTC	Real-Time Clock
RX	Receive
SCL/SCK	Serial Clock
SD	Secure Digital
SDA	Serial Data
SDHC	Secure Digital High Capacity
SDXC	Secure Digital eXtended Capacity
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
SS	Slave Select
TWI	Two-Wire Interface
TX	Transmit
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus
VCC	Voltage Common Collector
WLAN	Wireless Local Area Network

1 Einführung

1.1 Hintergrund und Relevanz

In der heutigen Zeit spielt das GPS-Tracking eine immer größere Rolle. Die Anwendungsbereiche von GPS-Tracking sind vielfältig. In der Logistikbranche wird GPS-Tracking eingesetzt, um die genaue Position von Waren und Fahrzeugen zu verfolgen. Dies ermöglicht eine effiziente Routenplanung und eine präzise Lieferverfolgung. Im Flottenmanagement wird GPS-Tracking verwendet, um die Position und den Betriebsstatus von Fahrzeugen zu überwachen. Dies ermöglicht eine effiziente Flottenverwaltung und eine präzise Fahrzeugortung. Darüber hinaus wird GPS-Tracking auch im Bereich der Navigation eingesetzt, um die genaue Position und den Weg zu einem Zielort zu bestimmen.

In Szenarien, die einen intensiven Einsatz von Trackern erfordern, oder in schwer zugänglichen Bereichen wie Wäldern oder Bergen, ist es jedoch oft schwierig, die Kosten für meisten kommerziellen GPS-Tracker zu tragen. Die Entwicklung eines kostengünstigen GPS-Trackers, der dennoch zuverlässige und genaue Daten liefert, könnte daher eine attraktive Alternative darstellen. Ein solches System könnte nicht nur die Kosten für die Anschaffung und Wartung von Trackern senken, sondern auch die Effizienz und Genauigkeit der Positionsverfolgung verbessern.

Ein Beispiel dazu ist [1]. Dieses Beispiel erklärt eine Richtung, dass die GPS-Tracking-Technologie im Tierschutz eingesetzt wird. In diesem Fall spielt der GPS-Tracker eine wichtige Rolle, wenn sich der Gesundheitszustand der Wildtiere verschlechtert. Die GPS-Tracker verwendet einen kostengünstigen Mikrocontroller (ATmega88PA), werden an den Tieren befestigt und zeichnen die GPS-Koordinaten in festgelegten Zeitintervallen auf. Dazu kann auch die Position des Tieres an die zuständige Behörde senden, um eine schnelle Rettung zu ermöglichen.

1.2 Zielsetzung

Das Hauptziel dieser Arbeit ist die Entwicklung eines effizienten und benutzerfreundlichen GPS-Tracking-Systems auf Basis eines kostengünstigen Mikrocontrollers. Dieses System soll nicht nur in der Lage sein, GPS-Koordinaten und Zeitstempel in vordefinierten Intervallen präzise aufzuzeichnen, sondern auch die gesammelten Daten sicher und zugänglich auf einer SD-Karte zu speichern. Eine wesentliche Funktion ist die Echtzeitanzeige der GPS-Koordinaten auf einem LCD-Display, was eine unmittelbare Orientierung und Positionsverfolgung ermöglicht. Zusätzlich ist die Steuerung der Messung und das Auslesen der Daten über Taster vorgesehen, um eine einfache Handhabung zu gewährleisten. Ein weiteres zentrales Ziel ist die Entwicklung einer PC-Anwendung, die nicht nur die Visualisierung der empfangenen Daten ermöglicht, sondern auch deren Speicherung in einem nutzerfreundlichen Format. Die Lösung soll dabei nicht nur kosteneffizient und einfach in der Bedienung sein, sondern auch eine hohe Zuverlässigkeit und Genauigkeit bieten.

Um diese Ziele zu erreichen, wurde während der Entwicklungsphase des GPS-Trackers unter Einsatz eines kostengünstigen Mikrocontrollers eng mit Prof. Dr. Axel Thümmler zusammengearbeitet, um die grundlegenden Anforderungen zu definieren. Diese Anforderungen wurden in weiteren Gesprächen und Beratungen mit dem wissenschaftlichen Mitarbeiter Ilya Raza spezifiziert und verfeinert. Die Ergebnisse dieser Diskussionen führten zu den in Tabelle 1.1 zusammengefassten Grundanforderungen für die Produktentwicklung.

Tabelle 1.1: Funktionsanforderung des Tracking-Systems

Nr	Funktion	Beschreibung der Anforderung
1	Taster-Bedienung	Ermöglicht es dem Benutzer, die Messung zu starten, zu pausieren und neu zu initialisieren, was eine flexible Handhabung der Datenerfassung ermöglicht. Das System unterstützt auch das Starten des Lesens der Daten.
2	Anzeigefunktion	Zeigt die aktuelle Position und den Zeitstempel in Echtzeit an. Informiert den Benutzer über den aktuellen Betriebsstatus und ermöglicht eine unmittelbare Rückmeldung während des Betriebs.
3	Speicherfunktion	Speichert die aufgezeichneten GPS-Daten und Zeitstempel sicher auf einer SD-Karte, was eine langfristige Datensicherung und einfache Übertragbarkeit ermöglicht.
4	Lesefunktion	Ermöglicht das Auslesen der auf der SD-Karte gespeicherten Daten, was eine spätere Analyse und Visualisierung der aufgezeichneten Routen unterstützt.
5	Erstellung lesbarer Dateien	Konvertiert die ausgelesenen Daten in ein benutzerfreundliches Format, das es ermöglicht, den Messungspfad visuell auf einer Karte darzustellen und somit eine detaillierte Analyse der zurückgelegten Wege bietet.
6	Benutzerfreundliche PC-Anwendung	Entwickelt eine intuitive PC-Anwendung für die Visualisierung, Bearbeitung und Speicherung der GPS-Daten, die eine einfache Interaktion mit den gesammelten Informationen ermöglicht.

Die oben entwickelten Anforderungen bildeten den Rahmen für die Gestaltung des Tracking-Systems und stellten sicher, dass es die Anforderungen der verschiedenen Stakeholder erfüllte. Die Einbeziehung von Expertenmeinungen hat wesentlich zur Qualität und Effektivität des entwickelten Systems beigetragen.

1.3 Vorgehensweise

Die Entwicklung eines GPS-Tracking-Systems folgt einem strukturierten Vorgehen, das sicherstellt, dass die Anforderungen der Stakeholder erfüllt und technische Herausforderungen effizient bewältigt werden. Dieser Prozess beinhaltet mehrere Kernschritte:

Anforderungsanalyse: Zu Beginn des Projekts steht die Ermittlung und Analyse der Anforderungen an das GPS-Tracking-System. Hier werden die Bedürfnisse und Erwartungen der Stakeholder genau definiert.

Literaturrecherche und Designentwurf: Nach der Anforderungsanalyse folgt die Recherche technischer Grundlagen, die für die Entwicklung des Systems notwendig sind. In diesem Schritt werden passende Hardwarekomponenten ausgewählt und ein vorläufiges Softwaredesign entwickelt.

Umsetzung und Softwareentwicklung: Im nächsten Schritt erfolgt die eigentliche Entwicklung des Systems. Die ausgewählten Hardwarekomponenten werden beschafft, miteinander verbunden und mit der entwickelten Software ausgestattet. Zusätzlich wird eine Anwendung für den PC entwickelt, die die Datenverwaltung und -analyse ermöglicht.

Test und Validierung: Nach der Entwicklung des Systems werden umfangreiche Tests durchgeführt, um die Funktionalität und Zuverlässigkeit des GPS-Tracking-Systems zu überprüfen. Diese Phase beinhaltet sowohl Simulationstests als auch die Bewertung der Datenpräzision und Systemzuverlässigkeit.

Ergebnisse und Diskussion: Abschließend werden die Ergebnisse der Test- und Validierungsphase präsentiert und diskutiert. Dabei wird untersucht, inwieweit das entwickelte System die anfangs definierten Anforderungen erfüllt und den Erwartungen der Stakeholder entspricht. Zusätzlich werden mögliche Verbesserungen und Erweiterungen für das System identifiziert und vorgeschlagen.

2 Grundlagen und Designentwurf

In diesem Abschnitt werden die technischen Grundlagen für die Entwicklung des GPS-Tracking-Systems recherchiert. Dazu gehören die Auswahl der Hardwarekomponenten und die Integration der Komponenten. Die Ergebnisse der Recherche und Analyse bilden die Grundlage für den Designentwurf des GPS-Tracking-Systems.

2.1 Mikrocontroller

2.1.1 Grundlegende Begriffe von Mikrocontrollern

Mikrocontroller sind integrierte Schaltungen, die einen Mikroprozessor, Speicher und Ein-/Ausgabeperipherie auf einem einzigen Chip vereinen [2]. Ein wesentliches Anwendungsfeld von Mikroprozessoren und insbesondere Mikrocontrollern sind die sogenannten eingebetteten Systeme (Embedded Systems) [2].

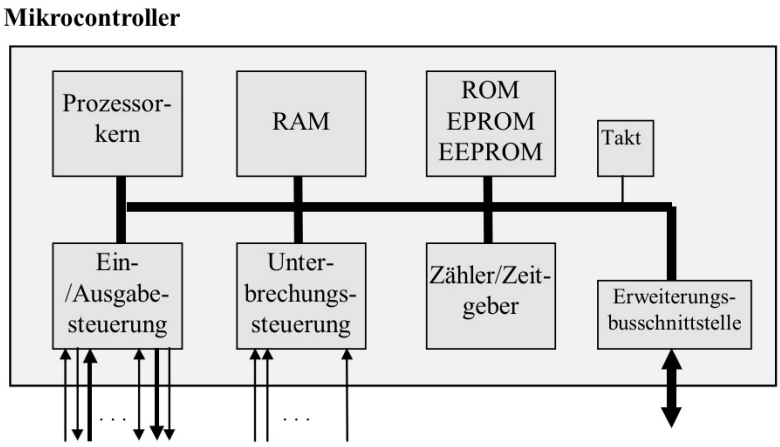

Abbildung 2.1: Aufbau eines typischen Mikrocontrollers [2]

Abbildung 2.1 zeigt den Aufbau eines typischen Mikrocontrollers. Mikrocontroller bestehen aus verschiedenen Komponenten, die auf die Lösung von Steuerungs- und Kommunikationsaufgaben zugeschnitten sind. Die wichtigsten Komponenten sind der Prozessorkern, der Speicher, die Ein-/Ausgabeeinheiten, die zeitgeberbasierten Einheiten, die Unterbrechungssteuerung und die DMA (Direct Memory Access) [2].

Der Prozessorkern, der die Befehle des Programms ausführt und die anderen Komponenten

steuert. Der Prozessorkern kann verschiedene Architekturen aufweisen, wie z.B. CISC, RISC, VLIW oder EPIC. Die Architektur bestimmt die Struktur und den Umfang des Befehlssatzes, die Anzahl und Art der Register, die Adressraumorganisation, die Pipelinetechniken und die Unterstützung von Parallelität und Spekulation [2].

Der Speicher kann aus verschiedenen Typen bestehen, wie z.B. Festwertspeicher (ROM), Schreib-/Lesespeicher (RAM) oder Flash-Speicher. Der Speicher kann intern oder extern zum Mikrocontroller angebunden sein. Die Speichergröße, die Zugriffszeit, die Lebensdauer und die Programmierbarkeit sind wichtige Faktoren für die Speicherauswahl [2].

Die Ein-/Ausgabeeinheiten, die die Schnittstellen zu den externen Geräten und Sensoren bilden. Die Ein-/Ausgabeeinheiten können digital oder analog sein, parallel oder seriell, synchron oder asynchron. Sie können verschiedene Standards und Protokolle unterstützen, wie z.B. UART, SPI, I2C, CAN, USB, Ethernet oder WLAN. Die Ein-/Ausgabeeinheiten können direkt an den Prozessorkern oder über einen Erweiterungsbus angeschlossen sein [2].

Die zeitgeberbasierten Einheiten, die die zeitliche Steuerung und Überwachung der Mikrocontrollerfunktionen ermöglichen. Die zeitgeberbasierten Einheiten können aus Zählern, Zeitgebern, Capture-und-Compare-Einheiten, Pulsweitenmodulatoren, Watchdog-Einheiten oder Echtzeit-Ein-/Ausgabeeinheiten bestehen [2].

Die Unterbrechungssteuerung, die die Reaktion des Mikrocontrollers auf interne oder externe Ereignisse ermöglicht. Die Unterbrechungssteuerung kann aus einem Unterbrechungsvektor, einem Unterbrechungsprioritätsregister, einem Unterbrechungsstatusregister und einem Unterbrechungsmaskenregister bestehen. Sie kann verschiedene Unterbrechungsquellen und -arten verwalten, wie z.B. Hardware- oder Softwareunterbrechungen, maskierbare oder nicht-maskierbare Unterbrechungen, vorrangige oder gleichrangige Unterbrechungen. Sie kann verschiedene Unterbrechungsbehandlungsverfahren durchführen, wie z.B. Polling, Daisy-Chaining oder Vektorisierung [2].

Die DMA (Direct Memory Access), die die direkte Übertragung von Daten zwischen Speicher und Ein-/Ausgabeeinheiten ohne Beteiligung des Prozessorkerns ermöglicht. Die DMA kann aus einem DMA-Controller, einem DMA-Kanalregister, einem DMA-Adressregister und einem DMA-Zählerregister bestehen. Sie kann verschiedene DMA-Modi und -Verfahren unterstützen, wie z.B. Burst- oder Cycle-Stealing-Modus, Single- oder Multi-Transfer-Verfahren, Memory-to-Memory- oder Memory-to-Peripheral-Verfahren [2].

2.1.2 Auswahl des Mikrocontrollers

Für die Entwicklung eines GPS-Trackers wurde ein Vergleich verschiedener auf dem Markt erhältlicher Mikrocontroller durchgeführt, wobei besonderes Augenmerk auf bestimmte Kriterien gelegt wurde. Die Verfügbarkeit des Mikrocontrollers ist entscheidend, da er leicht zu beschaffen sein sollte und eine lange Lebensdauer aufweisen muss. Gleichzeitig spielt der Preis eine wichtige Rolle, um die Gesamtkosten des Trackers zu minimieren. Eine ausführliche und verständliche Dokumentation erleichtert die Programmierung und den Einsatz des Mikrocontrollers erheblich. Die Unterstützung durch den Hersteller sowie eine aktive Community sind ebenfalls von Bedeutung, da sie bei Fragen und Problemen hilfreich sein können.

Die Bauformen, in denen der Mikrocontroller verfügbar ist, müssen vielfältig sein, um eine flexible Anpassung an das Design des Tracker-Gehäuses zu ermöglichen. Der Betrieb des Mikrocontrollers bei niedriger Spannung ist wichtig, um Kompatibilität mit anderen Tracker-Komponenten zu gewährleisten. Ein geringer Stromverbrauch ist entscheidend für die Langlebigkeit des Trackers im Batteriebetrieb. Die Taktgeschwindigkeit des Mikrocontrollers muss hoch genug sein, um GPS-Daten effizient verarbeiten und übertragen zu können. Schließlich ist ausreichender Speicherplatz, sowohl im Flash-Speicher als auch im RAM, notwendig, um das Programm und die Daten speichern zu können.

Nach einem Vergleich verschiedener Mikrocontroller-Familien, wie z.B. AVR, PIC, MSP430, Zilog, NXP und STM32, wurde die AVR-Familie von Microchip (früher Atmel) als geeignete Wahl identifiziert [3]. Die AVR-Mikrocontroller zeichnen sich durch eine klare, moderne und übersichtliche Struktur aus, die die Vermittlung von Grundlagenwissen erleichtert. Sie haben einen gemeinsamen Befehls- und Registersatz, der die Portabilität der Programme ermöglicht. Sie sind weit verbreitet und haben eine große Nutzer- und Entwicklergemeinschaft. Sie sind in verschiedenen Bauformen und Preisklassen erhältlich und bieten eine hohe Leistung bei geringem Stromverbrauch.

Innerhalb der AVR-Familie wurde der ATmega88PA als konkreter Mikrocontroller für den GPS-Tracker ausgewählt. Der ATmega88PA ist ein 8-Bit-Mikrocontroller mit einem RISC-Prozessor, der mit bis zu 20 MHz getaktet werden kann [4]. Er hat 8 KB Flash-Speicher, 512 Byte EEPROM und 1 KB SRAM [4]. Er hat 23 programmierbare I/O-Pins, die in 6 Ports organisiert sind [4]. Der ATmega88PA ist in verschiedenen Bauformen erhältlich, wie z.B. PDIP28, TQFP32, QFN32, MLF32, VQFN32, SOIC28, SOIC32, TSSOP28, TSSOP32, DFN28, DFN32, WLCSP32 [4].

Dazu wird Bauform PDIP28 für die Entwicklung und den Einsatz des GPS-Trackers verwendet und kostet etwa 1,5 bis 3 Euro pro Stück (Stand: 2024 Microchip). Er hat eine serielle UART-Schnittstelle, eine serielle SPI-Schnittstelle, eine serielle TWI-Schnittstelle (I2C) [4]. Er kann mit einer Spannung von 1,8 V bis 5,5 V betrieben werden und hat einen Stromverbrauch von 0,2 mA im Aktivmodus [4]. Er hat eine ausführliche und verständliche Dokumentation und wird vom Microchip Studio (früher Atmel Studio 7) unterstützt. Der ATmega88PA erfüllt die Anforderungen an den Mikrocontroller für den GPS-Tracker und bietet eine gute Grundlage für die Entwicklung und den Einsatz des Trackers. Abbildung 2.2 zeigt die Anschlussbelegung des ATmega88PA in einem 28-poligen PDIP-Gehäuse.

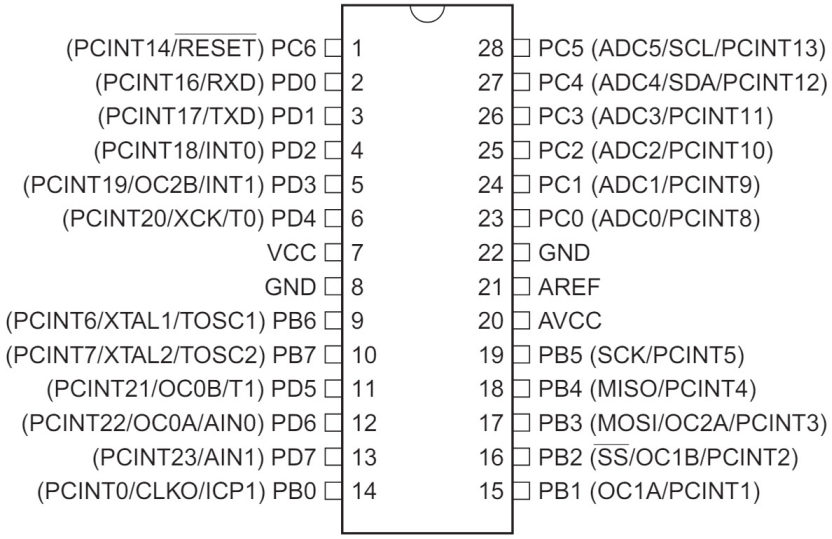

Abbildung 2.2: Die Anschlussbelegung des ATmega88PA in der Bauform PDIP28 [4]

2.1.3 Auswahl des Entwicklungsboards

Um den ATmega88PA zu programmieren und zu testen, wurde das myAVR Board MK2 verwendet. Abbildung 2.3 zeigt das myAVR Board MK2. Das myAVR Board MK2 hat einen ATmega88PA-Mikrocontroller in einem PDIP28-Gehäuse, der mit einem 3686400-MHz-Quarzoszillator getaktet wird. Über Microchip JTAGICE3 Debugger kann der ATmega88PA auf dem myAVR Board MK2 programmiert und debuggt werden. Außerdem bietet das Board verschiedene Schnittstellen und Anschlüsse, wie z.B. USB, UART, SPI, I2C, was geeignet für die Entwicklung und den Einsatz des GPS-Trackers ist. Darüber hinaus bietet das Board zwei Taster, was für die Benutzerinteraktion des GPS-Trackers nützlich ist [5].

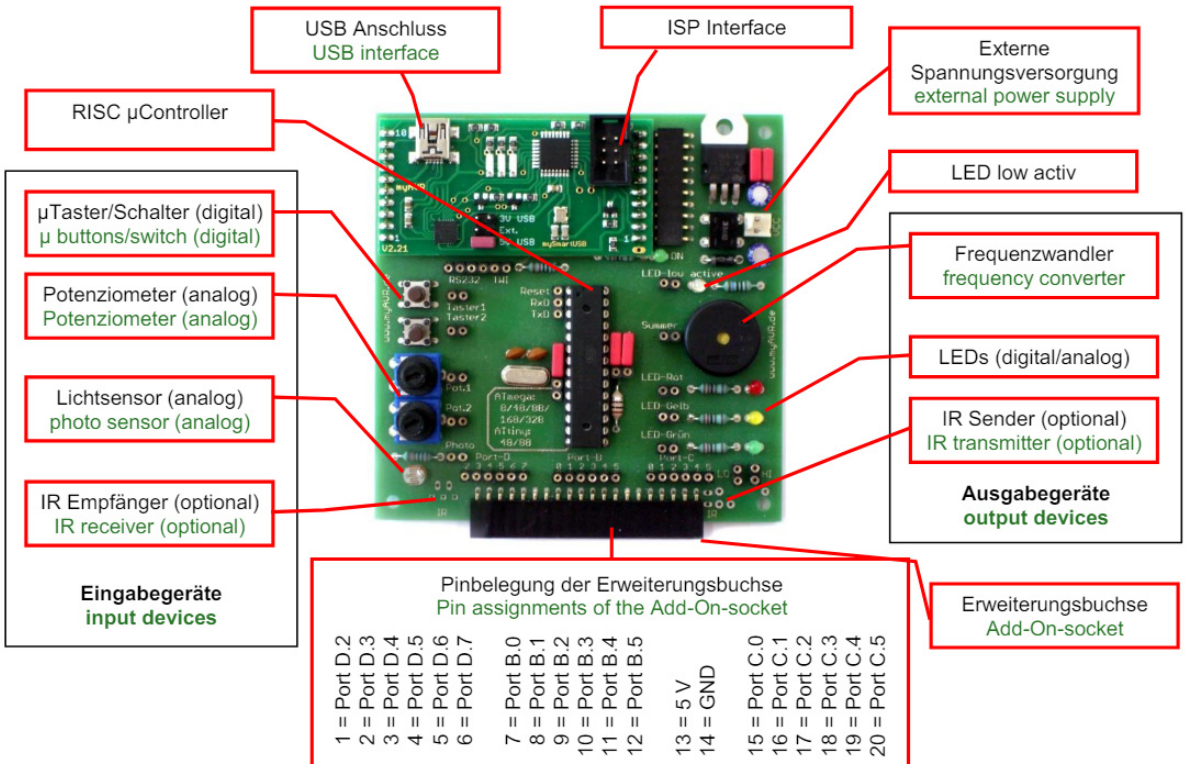

Abbildung 2.3: Das myAVR Board MK2 [5]

2.1.4 Auswahl der Programmiersprache und der Entwicklungsumgebung (IDE)

Die Programmierung von Mikrocontrollern erfordert die Verwendung von geeigneten Programmiersprachen und Entwicklungsumgebungen, die an die spezifischen Anforderungen und Eigenschaften dieser Geräte angepasst sind. Die Wahl der Programmiersprache hängt von verschiedenen Faktoren ab, wie z.B. dem Ziel des Projekts, den verfügbaren Ressourcen, den persönlichen Vorlieben und der Erfahrung des Entwicklers. Daher ist es wichtig, die Anforderungen und Erwartungen des jeweiligen Projekts zu berücksichtigen und die Sprache entsprechend auszuwählen. Neben der Programmiersprache ist auch die Entwicklungsumgebung ein wichtiger Aspekt der Mikrocontroller-Programmierung. Die Entwicklungsumgebung ist die Software, die dem Entwickler Werkzeuge zur Verfügung stellt, um den Code zu schreiben, zu kompilieren, zu übertragen, zu debuggen und zu testen. Es gibt verschiedene Entwicklungsumgebungen für Mikrocontroller, die je nach der verwendeten Programmiersprache, dem verwendeten Mikrocontroller und den gewünschten Funktionen variieren.

In diese Arbeit wird C Sprache verwendet, weil C Sprache direkte Kontrolle über die Hardware und eine hohe Leistung ermöglicht. C Sprache ist auch eine portable Sprache, die auf verschiedenen Plattformen und Architekturen verwendet werden kann. Außerdem wird Microchip Studio (früher

Atmel Studio 7) verwendet, weil Microchip Studio eine integrierte Entwicklungsumgebung (IDE) ist, die für die Programmierung von Atmel-Mikrocontrollern, wie z.B. AVR und ARM, entwickelt wurde [3]. Microchip Studio unterstützt die Programmierung in C/C++ mit einem vollständigen Zugriff auf die Hardware-Register und die Optimierung des Codes. Microchip Studio bietet auch eine leistungsstarke und anpassbare GUI, die einen erweiterten Code-Editor, einen integrierten Debugger, einen Simulator, einen Logikanalysator, einen Leistungsanalysator und einen Geräteprogrammierer umfasst.

2.2 GPS Modul

2.2.1 Grundlegende Begriffe des Global Positioning Systems

Das Global Positioning System (GPS) ist ein satellitengestütztes Navigationssystem, das es Nutzern ermöglicht, ihre Position und Zeit auf der Erde zu bestimmen. GPS nutzt ein Netzwerk von mindestens 24 Satelliten, die in sechs Umlaufbahnen um die Erde kreisen und kontinuierlich Navigationsnachrichten aussenden [6]. Die Satelliten senden ihre Navigationsnachrichten mit einer Frequenz von 1575,42 MHz (L1-Band) und 1227,60 MHz (L2-Band) aus, die von GPS-Empfängern empfangen und verarbeitet werden [7]. Diese Nachrichten enthalten Informationen über die Position und die Zeit des Satelliten sowie Korrekturdaten für die Signalverzögerung und die Umlaufbahnabweichung.

Um die Position zu berechnen, empfängt ein GPS-Empfänger die Navigationsnachrichten von mindestens vier Satelliten und misst die Zeitdifferenz zwischen dem Aussenden und dem Empfangen der Signale. Diese Zeitdifferenz entspricht der Entfernung zwischen dem Empfänger und dem Satelliten, die als Pseudostrecke bezeichnet wird. Durch die Verwendung von mindestens vier Pseudostrecken kann der Empfänger seine dreidimensionale Position (Längen-, Breiten- und Höhengrad) und die Zeit mit Hilfe eines mathematischen Verfahrens namens Trilateration bestimmen [7]. Die Trilateration ist ein geometrisches Verfahren, das die Position eines Punktes in einem Raum durch die Messung seiner Entfernung zu bekannten Punkten bestimmt. Die Trilateration basiert auf der Lösung eines Gleichungssystems, das die Entfernungen zu den Satelliten und die Position des Empfängers enthält. Die Lösung des Gleichungssystems ergibt die genaue Position des Empfängers. Das Gleichungssystem wird im Unterabschnitt 2.2.2 detailliert beschrieben.

GPS ist nicht das einzige System, das Satellitensignale zur Positionsbestimmung nutzt. Es gibt auch andere globale Navigationssatellitensysteme (GNSS), wie z.B. das russische GLONASS, das europäische Galileo und das chinesische Beidou [7]. Diese Systeme sind mit GPS kompatibel und bieten ähnliche oder bessere Genauigkeit und Verfügbarkeit. Darüber hinaus gibt es auch regionale und lokale Systeme, die GPS ergänzen oder erweitern, wie z.B. das europäische EGNOS, das japanische QZSS und das indische IRNSS. Diese Systeme bieten zusätzliche Signale oder Korrekturdaten, um die Genauigkeit, Zuverlässigkeit und Integrität von GPS zu verbessern.

2.2.2 Berechnung und Umwandlung der GPS-Koordinaten

Die Berechnung der Position eines GPS-Empfängers basiert auf der Bestimmung der genauen Entfernung zu mehreren GPS-Satelliten und der anschließenden Umrechnung dieser Entfernungen in geographische Koordinaten (Breitengrad, Längengrad und Höhe). Die Grundlage für die Umrechnung bildet das Prinzip der Trilateration, das die Position des Empfängers durch die Schnittpunkte von mindestens drei Kugeloberflächen, die um die Satelliten mit den Radien ihrer Distanzen zum Empfänger gezogen werden, ermittelt [8]. Abbildung 2.4 zeigt das Prinzip der Trilateration. In dieser Darstellung repräsentieren die Kugeln um die Satelliten S_1, S_2, und S_3 die Distanzmessungen vom Empfänger zu diesen Punkten. Der GPS-Empfänger befindet sich dort, wo sich alle drei Kugeloberflächen schneiden, was zu zwei möglichen Punkten führt: P und P'. In der Regel ist einer der Punkte, wie P, nicht plausibel (z.B. liegt er weit außerhalb der Erdoberfläche), und kann daher verworfen werden. Der übrig bleibende Schnittpunkt P' gibt die genaue Position des Empfängers an.

Die Entfernung r zum Satelliten wird durch die Signaltransitzeit t bestimmt, wobei die Lichtgeschwindigkeit c als Konstante dient, wie in der Gleichung 2.1 gezeigt:

$$r = c \cdot t \tag{2.1}$$

Unter Berücksichtigung der Erdatmosphäre und anderer Störfaktoren wird die Formel für die Entfernungsberechnung erweitert, um Korrekturfaktoren einzuschließen. Die Position (x, y, z) des GPS-Empfängers in einem dreidimensionalen kartesischen Koordinatensystem, das im Mittelpunkt der Erde zentriert ist, kann durch das Lösen des Gleichungssystems aus den Entfernungen zu mindestens drei Satelliten bestimmt werden, wie in der Gleichung 2.2 gezeigt:

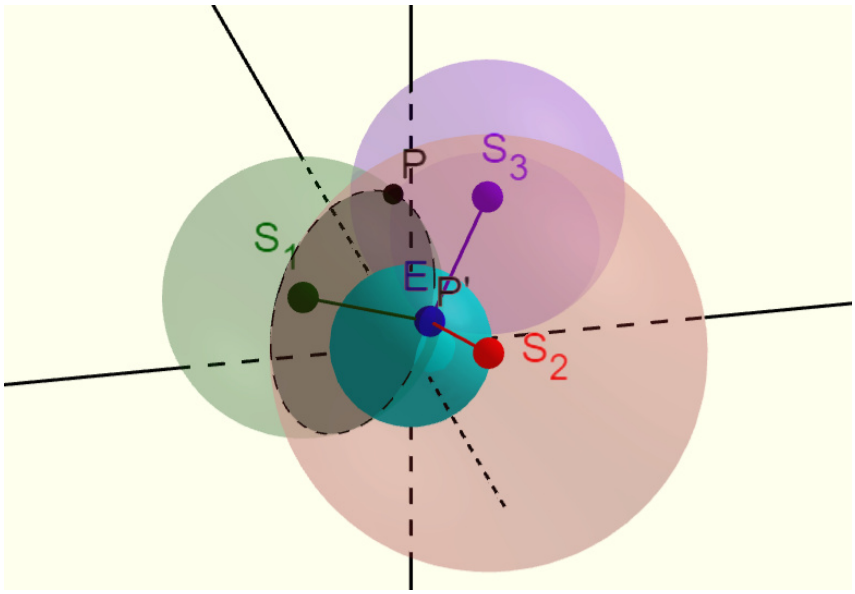

Abbildung 2.4: Das Prinzip der Trilateration [8]

$$
\begin{aligned}
(x - x_{S_1})^2 + (y - y_{S_1})^2 + (z - z_{S_1})^2 &= r_1^2 \\
(x - x_{S_2})^2 + (y - y_{S_2})^2 + (z - z_{S_2})^2 &= r_2^2 \\
(x - x_{S_3})^2 + (y - y_{S_3})^2 + (z - z_{S_3})^2 &= r_3^2
\end{aligned}
\tag{2.2}
$$

Hierbei sind $(x_{S_i}, y_{S_i}, z_{S_i})$ die Koordinaten der Satelliten und r_i die gemessenen Entfernungen zum Empfänger. Nach der Bestimmung der kartesischen Koordinaten des Empfängers können diese in geographische Koordinaten (Längen-, Breiten- und Höhengrad) umgerechnet werden. Die genaue Berechnung der GPS-Position erfordert zusätzlich die Anwendung von Korrekturen für Signallaufzeitverzögerungen, die durch die Ionosphäre und Troposphäre verursacht werden, sowie für die relativistische Zeitdilatation [9]. Durch die Integration dieser Korrekturen und die Nutzung fortgeschrittener Algorithmen können moderne GPS-Empfänger Positionen mit einer Genauigkeit von wenigen Metern ermitteln.

Geographische Koordinaten werden häufig im Grad-Minuten-Sekunden (DMS)-Format angegeben. Um diese in das Dezimalgrad-Format umzurechnen, das in vielen geographischen Informationssystemen und bei der Programmierung genutzt wird, kann folgende Methode angewandt werden. Sei D der Wert in Grad, M der Wert in Minuten und S der Wert in Sekunden der ursprünglichen Koordinaten. Die Umrechnung in Dezimalgrad D_{dez} kann mit der Gleichung 2.3 durchgeführt werden. Dabei ist zu beachten, dass bei südlichen Breitengraden und westlichen Längengraden das Ergebnis negativ ist, um die Richtung zu kennzeichnen. Falls bei einer

Umrechnung negative Werte entstehen, kann das Vorzeichen einfach umgekehrt werden, um die ursprüngliche Richtung zu erhalten, wie in der Gleichung 2.4 gezeigt.

$$D_{\text{dez}} = D + \frac{M}{60} + \frac{S}{3600} \tag{2.3}$$

$$D_{\text{dez}} = -(D + \frac{M}{60} + \frac{S}{3600}) \tag{2.4}$$

Beispiel: Die Umwandlung von 49° 30’ 0” in Dezimalgrad wird wie folgt Gleichung 2.5 durchgeführt:

$$49 + \frac{30}{60} + \frac{0}{3600} = 49.5° \tag{2.5}$$

Diese Methode ermöglicht eine präzise und einfache Umrechnung von Koordinaten im DMS-Format in das Dezimalgrad-Format, welches für weitere Berechnungen und Anwendungen erforderlich ist.

2.2.3 Auswahl der seriellen Schnittstellen (UART)

Die serielle Kommunikation über UART (Universal Asynchronous Receiver/Transmitter) spielt eine wesentliche Rolle in der Datenübertragung zwischen verschiedenen elektronischen Geräten, insbesondere in Anwendungen wie der Kommunikation mit GPS-Modulen. UART ist ein universell einsetzbarer Sender und Empfänger für asynchrone Datenübertragungen. Der Begriff „asynchron“ bedeutet, dass kein Taktsignal zwischen Sender und Empfänger ausgetauscht wird, wodurch eine flexible und einfache Verbindung zwischen verschiedenen Systemen ermöglicht wird [3], [10].

Jedes übertragene Datenpaket beginnt mit einem Startbit, gefolgt von einer vorher festgelegten Anzahl von Datenbits, optional einem Paritätsbit zur Fehlererkennung und einem oder mehreren Stoppbits. Diese Struktur ermöglicht es dem Empfänger, jedes Wort aus dem kontinuierlichen Datenstrom zu extrahieren, ohne dass eine externe Taktquelle erforderlich ist [3], [10]. Die Abbildung 2.5 zeigt den zeitlichen Verlauf der Übertragung eines Bytes bei der Verwendung des UART-Protokolls. Es illustriert die Zusammensetzung eines typischen „Frames“ aus Startbit,

Datenbits, optionalem Paritätsbit und Stoppbits. Die Übertragung beginnt mit einem Startbit, das auf niedrig (logisch 0) gesetzt ist, gefolgt von den Datenbits – beginnend mit dem Least Significant Bit (Bit 0) bis zum Most Significant Bit (Bit 7). Optional kann ein Paritätsbit für die Fehlererkennung eingefügt werden, gefolgt von einem oder mehreren Stoppbits, die auf hoch (logisch 1) gesetzt sind, um das Ende des Frames anzuzeigen. Der gesamte Frame endet in einer Ruhephase, bevor das nächste Byte beginnt.

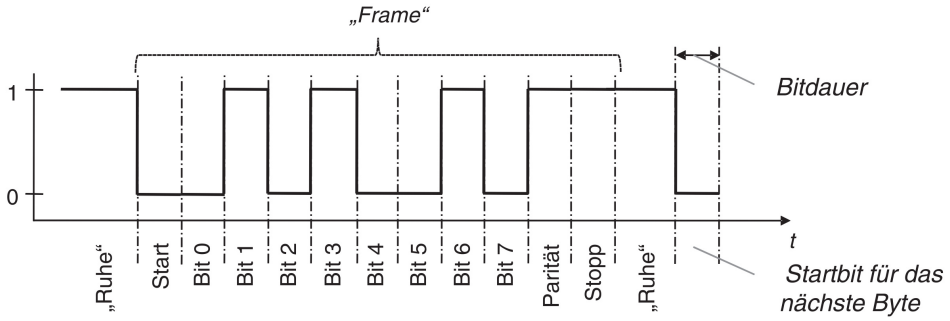

Abbildung 2.5: Zeitlicher Verlauf der Übertragung eines Bytes bei der Verwendung des UART-Protokolls [10]

UART-Schnittstellen sind aufgrund ihrer Einfachheit und der geringen Anzahl benötigter Leitungen (hauptsächlich Senden (TX) und Empfangen (RX)) in vielen Mikrocontrollern und GPS-Modulen integriert [3], [10]. Diese Integration führt zu geringeren Hardwarekosten und einfacherer Implementierung im Vergleich zu komplexeren Kommunikationsprotokollen. Für viele GPS-Anwendungen, wo die Datenrate und Komplexität relativ gering sind, bietet UART eine kosteneffiziente Lösung. Die Tabelle 2.1 gibt einen Überblick über typische Baudraten, die in der Praxis verwendet werden, und die entsprechenden Bitdauern in Mikrosekunden (µs). Die Baudrate definiert, wie viele Bits pro Sekunde übertragen werden, und ist ein wesentlicher Parameter bei der Konfiguration von UART-Schnittstellen. Eine höhere Baudrate ermöglicht eine schnellere Datenübertragung, erfordert jedoch auch eine präzisere Zeitabstimmung zwischen Sender und Empfänger. Standard-Baudraten wie 9600 oder 115200 Bits pro Sekunde sind in der Industrie weit verbreitet und werden häufig für die Kommunikation mit GPS-Modulen eingesetzt, da sie einen guten Kompromiss zwischen Geschwindigkeit und Zuverlässigkeit bieten.

Tabelle 2.1: In der Praxis häufig verwendete Baudraten [10]

Baudrate (in bit/s)	Bitdauer (in µs)
2400	416,67
9600	104,17
19.200	52,08
38.400	26,04
57.600	17,36
115.200	8,68

Dies macht UART besonders geeignet für die Kommunikation mit GPS-Modulen. GPS-Module senden Daten in einem formatierten Textformat, welches leicht durch eine UART-Schnittstelle interpretiert werden kann. Zudem benötigen GPS-Module oft keine hohen Übertragungsraten, was mit der typischen Leistungsfähigkeit von UART-Verbindungen übereinstimmt.

2.2.4 Auswahl des GPS Moduls

Der CD-PA1616D GNSS Patch-Antennenmodul, ausgestattet mit dem MediaTek GNSS-Chipsatz MT3333, bietet eine Reihe von Merkmalen, die ihn für den Einsatz in GPS-Trackern besonders geeignet machen. Abbildung 2.6 zeigt die Pins des CD-PA1616D GNSS Patch-Antennenmoduls.

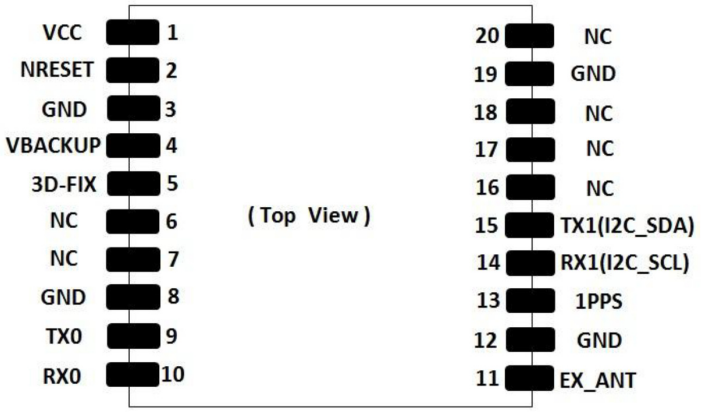

Abbildung 2.6: Die Pins des CD-PA1616D GNSS Patch-Antennenmoduls [11]

Das Modul ist für die Nutzung der L1-Band GPS-Frequenz von 1575,42 MHz ausgelegt, welche von GPS-Satelliten für zivile Zwecke genutzt wird, und kann ebenfalls die L1-Band GLONASS-Frequenz von 1598,0625 bis 1605,375 MHz empfangen, die von GLONASS-Satelliten verwendet

wird [11]. Mit einer hohen Empfindlichkeit von -165 dBm ist das Modul fähig, auch schwache GPS-Signale zu empfangen, was in städtischen oder bewaldeten Gebieten von großer Bedeutung ist, wo Signale leicht durch Gebäude oder Bäume blockiert werden können [11]. Es bietet eine hohe Positionsgenauigkeit von bis zu wenigen Metern, wobei die Positionsgenauigkeit ohne Hilfe bei 3,0 m und mit DGPS-Unterstützung bei 2,5 m liegt [11]. Zudem zeichnet sich das Modul durch eine kurze Startzeit aus, mit 1 s für einen heißen Start, 33 s für einen warmen Start und 35 s für einen kalten Start, was für GPS-Tracker, die eine schnelle Standortbestimmung benötigen, essentiell ist [11]. Der niedrige Stromverbrauch von 34 mA im Erfassungsmodus und 29 mA im Tracking-Modus macht das Modul ideal für batteriebetriebene Geräte wie GPS-Tracker [11]. Mit einem Gewicht von 6 g und den kompakten Abmessungen von 16,0 mm x 16,0 mm x 6,7 mm eignet es sich hervorragend für den Einsatz in tragbaren Geräten [11].

Diese Eigenschaften, insbesondere die hohe Sensitivität, schnelle Startzeiten und geringer Stromverbrauch, machen den CD-PA1616D zu einer idealen Wahl für GPS-Tracking-Anwendungen, bei denen Zuverlässigkeit und Effizienz entscheidend sind. Die kompakte Größe ermöglicht eine einfache Integration in verschiedene Geräteformfaktoren.

2.3 Display mit I2C Schnittstelle Modul

2.3.1 Auswahl der seriellen Schnittstellen (I2C / TWI)

Die Inter-Integrated Circuit (I2C) Schnittstelle, auch als Two-Wire Interface (TWI) bekannt, wurde in den frühen 1980er Jahren von Philips eingeführt und stellt eine wesentliche Komponente in der Kommunikation zwischen verschiedenen integrierten Bausteinen auf einer Leiterplatte dar. In diesem Abschnitt wird die Bedeutung und Funktionsweise dieser seriellen Schnittstelle erläutert und begründet, warum sie insbesondere für die serielle Kommunikation in Anzeigegeräten von Bedeutung ist.

I2C ist ein synchrones, seriell arbeitendes Bussystem, das mit nur zwei Leitungen - SCL (Serial Clock) und SDA (Serial Data) - eine effiziente Kommunikation zwischen Mikrocontrollern, A/D-Umsetzern, D/A-Umsetzern, Speichern und anderen Komponenten ermöglicht [3], [10]. Durch die geringe Anzahl an erforderlichen Leitungen reduziert sich der Verkabelungsaufwand erheblich, was gerade in komplexen Schaltungen wie bei Anzeigegeräten einen erheblichen Vorteil darstellt.

Die I2C-Anschlüsse sind als Open-Collector- bzw. Open-Drain-Ausgänge konzipiert, was bedeutet, dass mehrere Geräte gleichzeitig an den Bus angeschlossen werden können, ohne dass es zu Konflikten kommt. Jedes Gerät kann die Leitung auf einen Low-Pegel ziehen, aber keines kann aktiv einen High-Pegel setzen. Dies wird durch Pull-Up-Widerstände erreicht, die die Leitungen im Ruhezustand auf High-Pegel halten [3], [10].

Bei Anzeigegeräten ist die Übertragung von Daten zwischen verschiedenen Komponenten wie dem Mikrocontroller, Speicherbausteinen und dem Display-Controller von zentraler Bedeutung. I2C/TWI bietet hierfür eine flexible und effiziente Lösung. Die Fähigkeit von I2C, mehrere Geräte über nur zwei Leitungen zu verbinden, ermöglicht eine vereinfachte und kostengünstige Implementierung.

Zusätzlich unterstützt das I2C-Protokoll sowohl Master- als auch Slave-Konfigurationen, was bedeutet, dass ein Gerät (z.B. ein Mikrocontroller) als Master fungieren und die Kommunikation mit mehreren Slaves (z.B. Display) steuern kann. Dies ist besonders nützlich in Anzeigegeräten, wo der Mikrocontroller Informationen an verschiedene Komponenten des Displays senden muss [3], [10].

Abbildung 2.7 stellt die Synchronisierung beim I2C-Protokoll dar. Sie zeigt, wie die Datenleitung (SDA) und die Taktleitung (SCL) zusammenarbeiten, um eine synchrone Übertragung zu ermöglichen. Die Daten auf der SDA-Leitung werden nur geändert, wenn das SCL-Signal auf Low ist, was die Integrität der übertragenen Daten sicherstellt. Dieses Timing ist entscheidend, da eine Änderung der Daten während eines High-Signals auf der SCL-Leitung als Start- oder Stoppbedingung interpretiert werden könnte.

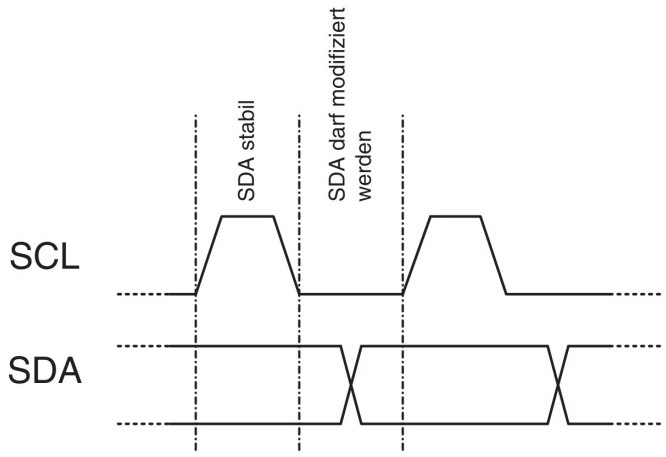

Abbildung 2.7: Synchronisierung beim I2C-Protokoll [10]

Abbildung 2.8 zeigt die Start- und Stoppbedingungen im I2C-Protokoll. Die Startbedingung ist durch einen Übergang von High zu Low auf der SDA-Leitung bei einem High-Signal auf der SCL-Leitung gekennzeichnet. Dies signalisiert allen Geräten auf dem Bus, dass eine Kommunikation beginnt. Die Stoppbedingung ist das Gegenteil, wobei die SDA-Leitung von Low zu High übergeht, während SCL High ist, was das Ende einer Kommunikation anzeigt.

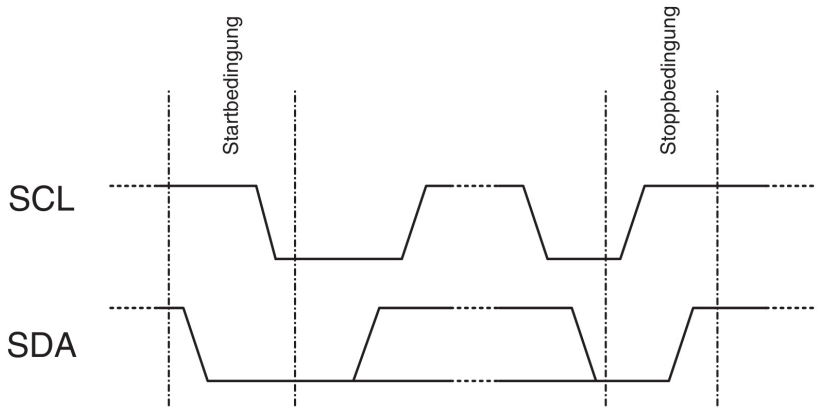

Abbildung 2.8: Start- und Stoppbedingungen im I2C-Protokoll [10]

In der Abbildung 2.9 wird ein vollständiger I2C-Kommunikationszyklus gezeigt, unterteilt in zwei Teile. Im ersten Teil sendet der Master die Bausteinadresse (im Beispiel `0x35`), gefolgt von einem Lese- oder Schreibbit. Nach dem Erhalt der Adresse und des R/W-Bits von den Slaves senden diese eine Bestätigung, das sogenannte Acknowledge, zurück an den Master. Der zweite Teil zeigt die eigentliche Datenübertragung, wobei im Beispiel der Slave den Wert `0xA5` an den Master sendet, gefolgt von einer weiteren Bestätigung durch den Master.

Zusammenfassend bietet I2C/TWI als serielle Schnittstelle in der Kommunikation von Anzeigegeräten eine Reihe von Vorteilen, darunter die Reduzierung des Verkabelungsaufwands, die Möglichkeit, mehrere Geräte über nur zwei Leitungen zu verbinden, und die Unterstützung von Master- und Slave-Konfigurationen. Diese Eigenschaften machen I2C/TWI zu einer idealen Wahl für die effiziente und zuverlässige Kommunikation in Anzeigegeräten.

2.3.2 Auswahl des Displays mit I2C Schnittstelle

Das ausgewählte LCD-Display, das HD44780 1602A mit I2C-Schnittstelle, bietet aufgrund seiner technischen Spezifikationen eine ideale Lösung für die Arbeit. Dieses Display weist eine Auflösung von 16x2 auf, was die Darstellung von 16 Zeichen pro Zeile auf zwei Zeilen ermöglicht, mit einer Sichtfläche von 12 mm x 56 mm, die für die Anzeige von GPS-Positionsdaten

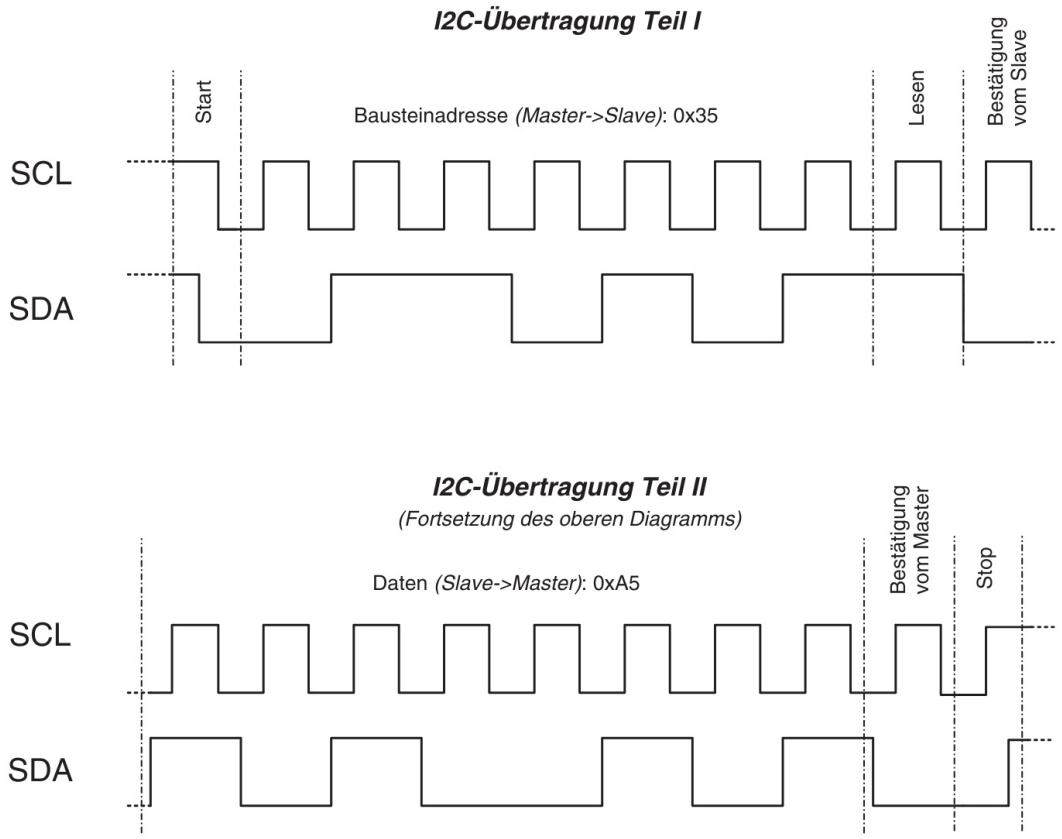

Abbildung 2.9: Ein vollständiger I2C-Kommunikationszyklus (Beispiel) [10]

ausreichend ist [12]. Die physische Größe des Displays beträgt 80 mm x 36 mm x 12,5 mm, was es kompakt und geeignet für Anwendungen mit begrenztem Platz macht [12]. Es kann mit einer Spannung von 3,3 V bis 5 V betrieben werden, was es kompatibel mit dem Mikrocontroller ATmega88PA macht [12]. Das Display bietet eine weiße Hintergrundbeleuchtung und einen Blickwinkel von 180 Grad, ideal für verschiedene Einsatzumgebungen und garantiert gute Lesbarkeit auch bei schlechten Lichtverhältnissen [12]. Der geringe Stromverbrauch, insbesondere der Hintergrundbeleuchtung mit nur 15 mA, macht das Display ideal für batteriebetriebene Geräte wie GPS-Tracker [12]. Zudem verfügt es über eine I2C-Schnittstelle, die eine einfache Verbindung mit dem Mikrocontroller ATmega88PA ermöglicht, wodurch es eine praktische Wahl für viele Anwendungen darstellt. Abbildung 2.10 zeigt das LCD-Display mit I2C-Schnittstelle 1602A HD44780.

Diese Eigenschaften machen das LCD-Display mit I2C-Schnittstelle 1602A HD44780 zu einer ausgezeichneten Wahl für viele Projekte. Seine kompakte Größe, flexible Spannungsversorgung, gute Helligkeit, niedriger Stromverbrauch und einfache Verbindung machen es ideal für den Einsatz in tragbaren Geräten wie GPS-Trackern.

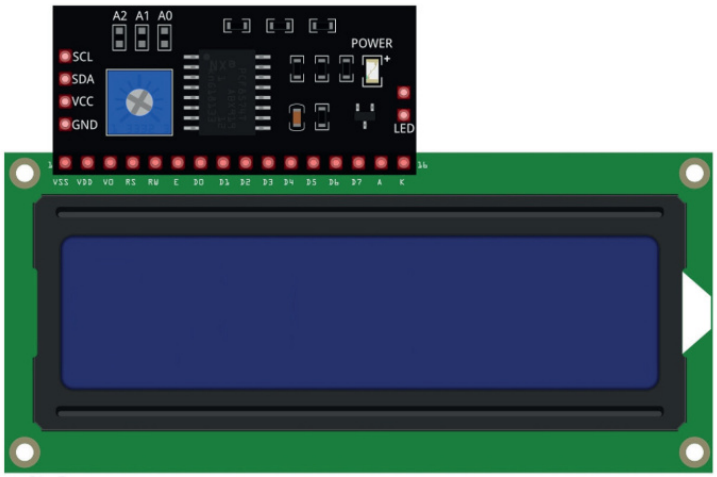

Abbildung 2.10: LCD-Display mit I2C-Schnittstelle 1602A HD44780 [12]

2.4 SD-Karte Modul

2.4.1 Auswahl der seriellen Schnittstellen (SPI)

Das SPI (Serial Peripheral Interface) ist eine weit verbreitete synchrone serielle Datenverbindungstechnologie, die in eingebetteten Systemen zur Kommunikation zwischen einem Mikrocontroller (Master) und einem oder mehreren Peripheriegeräten (Slaves) wie SD-Karten und Sensoren genutzt wird [3], [10]. Die Master-Slave-Architektur des SPI-Netzwerks vereinfacht durch eine klare Rollenverteilung das Design und die Implementierung von Systemen. SPI kennzeichnet sich durch den Einsatz separater Datenleitungen für die Kommunikation: MOSI (Master Out, Slave In) für die Datenübertragung vom Master zum Slave und MISO (Master In, Slave Out) für die Datenübertragung vom Slave zum Master [3], [10]. Zusätzlich wird eine Taktleitung (SCK) zur Synchronisierung und eine /SS-Leitung (Slave Select) zur Auswahl des kommunizierenden Slaves verwendet [3], [10].

Die taktgesteuerte Datenübertragung ermöglicht eine präzise Steuerung des Datenflusses, indem Daten entweder auf der steigenden oder fallenden Taktflanke übernommen werden. Die SPI-Kommunikation zeichnet sich durch konfigurierbare Parameter aus, wie die Auswahl der Datenübertragungsreihenfolge (MSB-first oder LSB-first) und der aktiven Taktflanke, was eine flexible Anpassung an spezifische Anforderungen ermöglicht [3], [10]. Dank der Skalierbarkeit des SPI kann das System leicht um zusätzliche Slaves erweitert werden, entweder durch Kaskadierung oder durch eine Sternverbindung, was die Integration mehrerer Geräte vereinfacht.

Wie in Abbildung 2.11 dargestellt, besteht die SPI-Verbindungsstruktur zwischen einem Master und einem Slave aus vier Hauptleitungen. In der SPI-Verbindungsstruktur, die aus vier Hauptleitungen besteht, überträgt der Master Daten zum Slave über die MOSI-Leitung, während der Slave über die MISO-Leitung Daten zum Master sendet. Das SCK-Signal, generiert vom Master, dient der Synchronisierung der Datenübertragung, und das /SS-Signal ermöglicht die Auswahl des aktiven Slaves für die Kommunikation. Diese klar definierten Leitungen und Signale tragen zur Effizienz und Zuverlässigkeit der SPI-basierten Datenkommunikation bei.

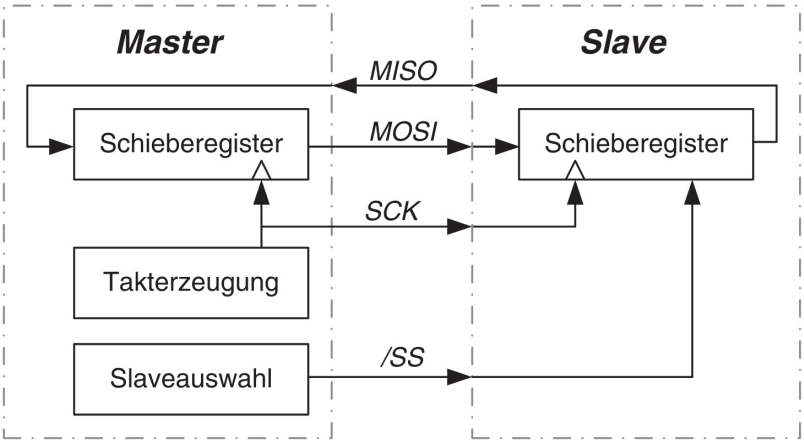

Abbildung 2.11: SPI-Verbindungsstruktur zwischen einem Master und einem Slave [10]

Der Signalverlauf, wie in Abbildung 2.12 gezeigt, verdeutlicht das Timing der Datenübertragung. Der Ruhezustand des Taktsignals (SCK) ist typischerweise auf 0 gesetzt, und die Datenübernahme erfolgt mit der ersten aktiven Flanke des Taktsignals. Die Darstellung zeigt, wie die Datenbits auf MOSI und MISO mit den Taktsignalen synchronisiert werden.

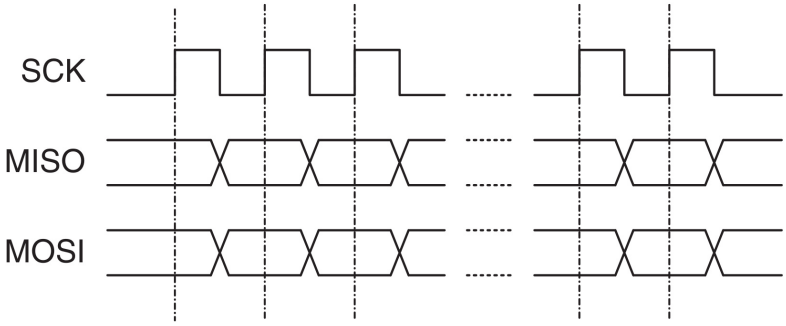

Abbildung 2.12: Signalverlauf der SPI-Datenübertragung [10]

Die Entscheidung, das SPI (Serial Peripheral Interface) für die Kommunikation mit SD-Kartenmodulen zu nutzen, gründet auf einer Reihe von Vorteilen, die dieses Protokoll bietet. Die Geschwindigkeit der Datenübertragung über SPI ist im Vergleich zu anderen seriellen Schnittstellen höher, was für Anwendungen, die eine schnelle Datenübertragung erfordern, entscheidend ist (wie es bei SD-Karten der Fall ist). Die einfache Integration in die Systemarchitektur ist ein weiterer wichtiger Faktor. Die meisten Mikrocontroller, darunter auch die der AVR-Familie, unterstützen das SPI-Protokoll nativ, was den Entwicklungsprozess vereinfacht und die Notwendigkeit zusätzlicher Hardware reduziert.

SPI zeichnet sich zudem durch seine Effizienz aus, da es im Vergleich zu parallelen Schnittstellen weniger Leitungen benötigt. Dies führt zu einer vereinfachten und kostengünstigeren Hardwarekonfiguration, die besonders in Systemen mit begrenztem Platzangebot vorteilhaft ist. Die Flexibilität von SPI, mehrere Geräte über denselben Bus steuern zu können, ermöglicht eine effiziente Nutzung verschiedener Peripheriegeräte innerhalb eines Systems. Diese Fähigkeit zur gleichzeitigen Anbindung mehrerer Geräte ist in eingebetteten Systemen, wo die Integration verschiedenartiger Funktionalitäten auf engem Raum erforderlich ist, besonders wertvoll.

Zusammenfassend lässt sich sagen, dass das SPI-Protokoll aufgrund seiner hohen Übertragungsgeschwindigkeit, der einfachen Implementierung, der Skalierbarkeit und der effizienten Nutzung der Systemressourcen für die Kommunikation mit SD-Kartenmodulen in eingebetteten Systemen eine ideale Wahl darstellt.

2.4.2 Grundlegende Begriffe von SD-Karte

Die SD-Karte (Secure Digital Card) ist ein weit verbreitetes Speichermedium, das in einer Vielzahl von elektronischen Geräten verwendet wird, wie z.B. Digitalkameras, Mobiltelefonen und Computern. Diese Karten nutzen den NAND-Flash-Speicher, eine Art von nichtflüchtigem Speicher, der seine Daten auch ohne Stromversorgung behält [13]. Die Funktionsweise von SD-Karten basiert auf der Speicherung von Daten in Form von elektrischen Ladungen, die in den Speicherzellen des Flash-Chips gespeichert werden. Die Ladungen werden durch das Anlegen einer Spannung an die Speicherzellen erzeugt und können in zwei Zuständen gespeichert werden, die als logische 0 und logische 1 bezeichnet werden. Die Speicherzellen sind in Blöcken organisiert, die wiederum in Sektoren unterteilt sind, um die Daten effizient zu verwalten und zu lesen [14].

Eine der Schlüsselfunktionen von SD-Karten ist ihre Fähigkeit, Daten über die Wear-Leveling-Technik zu speichern und zu verwalten. Diese Technik verteilt die Schreibzugriffe gleichmäßig über den Speicherchip, um die Lebensdauer der Karte zu verlängern [13]. Da jeder Bereich des Flash-Chips nur eine begrenzte Anzahl von Schreibzyklen aushält, ist diese Technik entscheidend, um eine vorzeitige Abnutzung zu vermeiden.

SD-Karten variieren primär in Speicherkapazität und Geschwindigkeit und lassen sich in drei Hauptkategorien einteilen: SD (Secure Digital), SDHC (Secure Digital High Capacity) und SDXC (Secure Digital eXtended Capacity), die jeweils unterschiedliche Speichergrößen und Dateisysteme unterstützen [14].

SD-Karten sind die Grundform dieser Technologie und bieten Speicherkapazitäten von bis zu 2 GB [14]. Sie arbeiten mit den Dateisystemen FAT12 oder FAT16, welche für kleinere Datenvolumen konzipiert wurden. Diese Karten stellen eine gute Wahl für ältere oder weniger anspruchsvolle Geräte dar, bei denen keine großen Datenmengen gespeichert werden müssen.

SDHC-Karten repräsentieren die nächste Generation mit Speicherkapazitäten von 4 GB bis 32 GB [14]. Sie nutzen das FAT32-Dateisystem, das effizienter mit größeren Dateien und Kapazitäten umgeht. SDHC-Karten eignen sich für Nutzer, die mehr Speicherplatz benötigen, etwa für hochauflösende Fotos oder längere Videos.

SDXC-Karten sind die fortschrittlichste Kategorie mit Speicherkapazitäten von über 32 GB bis zu 2 TB [14]. Sie verwenden das exFAT-Dateisystem, das für seine Fähigkeit, sehr große Dateien und Speicher zu verwalten, entwickelt wurde. Diese Karten sind ideal für professionelle Anwendungen, die extrem große Datenmengen erfordern, wie z.B. hochauflösendes Videoaufzeichnen, ausgedehnte Fotosammlungen und umfangreiche Datenspeicherung.

Für die meisten Anwendungen, einschließlich GPS-Tracker, sind SD-Karten mit 2GB Speicherplatz ausreichend. Diese SD-Karte ist mit einem 9-poligen Anschluss ausgestattet, der die Verbindung mit einem Mikrocontroller über die serielle Schnittstelle (SPI) ermöglicht. Die Tabelle 2.2 zeigt die Belegung der Kontakte einer SD-Karte und deren Anschluss an einen Mikrocontroller im SPI-Modus.

Tabelle 2.2: Belegung der Kontakte einer SD-Karte und deren Anschluss an einen Mikrocontroller im SPI-Modus [13]

SD-Karten-Pin	Funktion	Mikrocontroller-Anschluss
DAT2	Nicht verwendet im SPI-Modus	Nicht verbunden
DAT3 / SS	Slave Select im SPI-Modus	CS (Chip Select Pin)
CMD / MOSI	Master Out Slave In	MOSI (Master Out Slave In)
GND	Erdung	GND (Ground)
VCC	Versorgungsspannung	5V Versorgung
CLK / SCK	Taktleitung im SPI-Modus	SCK (Serial Clock)
DAT0 / MISO	Master In Slave Out im SPI-Modus	MISO (Master In Slave Out)
DAT1	Nicht verwendet im SPI-Modus	Nicht verbunden

3 Umsetzung und Softwareentwicklung

In diesem Abschnitt wird die Umsetzung des GPS-Tracking-Systems beschrieben. Zunächst wird die Hardware-Implementierung des Systems erläutert. Anschließend wird die Softwareentwicklung für den Mikrocontroller beschrieben. Softwareentwicklung umfasst Softwaredesign für Mikrocontroller und Softwaredesign für PC-Anwendung. Die Software für Mikrocontroller wird auf dem ATmega88PA Mikrocontroller ausgeführt und die Software für PC-Anwendung wird auf einem Windows 10/11 PC/Laptop ausgeführt. Die Software für Mikrocontroller und die Software für PC-Anwendung kommunizieren über die UART-Schnittstelle (RS232).

3.1 Bauteilverbindung

Die Hardware-Implementierung des GPS-Tracking-Systems umfasst die Verbindung der Hardwarekomponenten. Die Hardwarekomponenten sind der Mikrocontroller, das GPS-Modul, das LCD-Display und die SD-Karte. Die Verbindung der Hardwarekomponenten ist in Abbildung 3.1 dargestellt.

Die Abbildung 3.2 zeigt die detaillierten Verbindungen des GPS-Moduls mit dem Mikrocontroller. Die Verbindung erfolgt über die UART-Schnittstelle. TXD des GPS-Moduls ist mit RXD des Mikrocontrollers verbunden. VCC und GND sind die Kontakte für die Stromversorgung. RXD des GPS-Moduls muss nicht mit dem Mikrocontroller verbunden werden, da der Mikrocontroller keine Daten an das GPS-Modul sendet.

In diesem speziellen Szenario wird angegeben, dass der PC keine Daten an den Mikrocontroller sendet und das GPS-Modul ebenfalls nur Daten an den Mikrocontroller sendet. Dies vermeidet Konflikte, da es keinen Moment gibt, in dem der Mikrocontroller versuchen würde, gleichzeitig Daten von beiden Geräten zu empfangen. Abbildung 3.3 zeigt die Kommunikation zwischen PC, Mikrocontroller und GPS-Modul.

Anschließend wird die Verbindung des LCD-Displays mit dem Mikrocontroller in Abbildung 3.4 dargestellt. Die Verbindung erfolgt über das I2C-Schnittstelle. SDA und SCL sind die Kontakte für die I2C-Kommunikation. VCC und GND sind die Kontakte für die Stromversorgung.

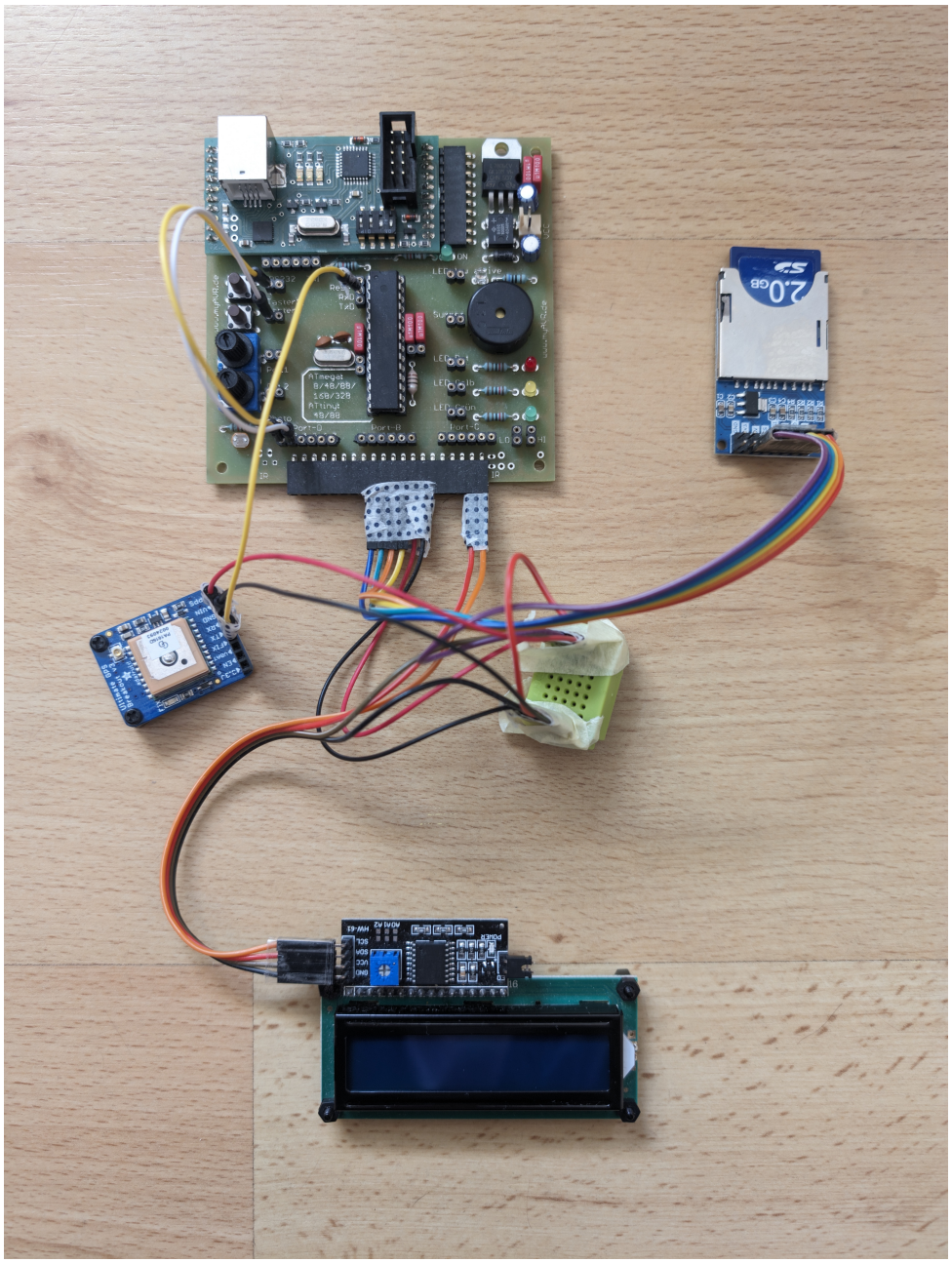

Abbildung 3.1: Verbindung der Hardwarekomponenten

Die Abbildung 3.5 zeigt die detaillierten Verbindungen der SD-Karte mit dem Mikrocontroller. Die Verbindung erfolgt über die SPI-Schnittstelle. MOSI, MISO, SCK und CS sind die Kontakte für die SPI-Kommunikation. VCC und GND sind die Kontakte für die Stromversorgung.

Zuletzt wird die Verbindung der Taster mit dem Mikrocontroller in Abbildung 3.6 dargestellt. Die Taster sind über die Interrupts INT0 und INT1 mit dem Mikrocontroller verbunden. INT0 und INT1 sind die Kontakte für die Interrupts.

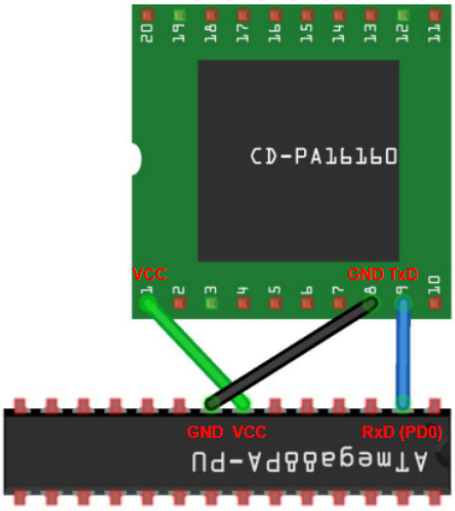

Abbildung 3.2: Verbindung des GPS-Moduls mit dem Mikrocontroller

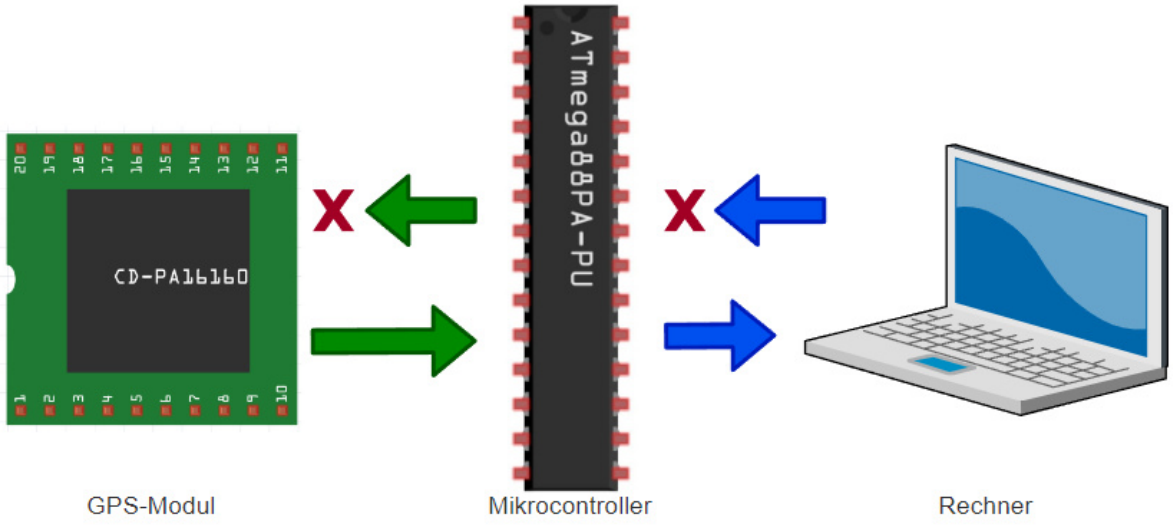

Abbildung 3.3: UART-Kommunikation zwischen PC, Mikrocontroller und GPS-Modul

3.2 Entwicklung der Mikrocontroller-Firmware

Die Software für Mikrocontroller besteht aus zwei Hauptmodi: Messmodus und Lesemodus. Im Messmodus werden GPS-Daten empfangen und verarbeitet. Die Daten werden dann in einem Format auf der SD-Karte gespeichert. Im Lesemodus werden Daten von der SD-Karte gelesen und über UART ausgegeben. Der Lesemodus liest die Daten bis zur zuletzt gespeicherten Adresse. Listing 5.1 in Anhang zeigt den vollständigen Quellcode der Mikrocontroller-Firmware.

Zur Benutzerinteraktion und Feedback nutzt die Firmware externe Interrupts zur Verarbeitung der Benutzereingaben über Tasten. Diese Eingaben ermöglichen es dem Benutzer, zwischen dem Mess- und dem Lesemodus zu wechseln oder die Adresse im EEPROM zurückzusetzen. Ein LCD-

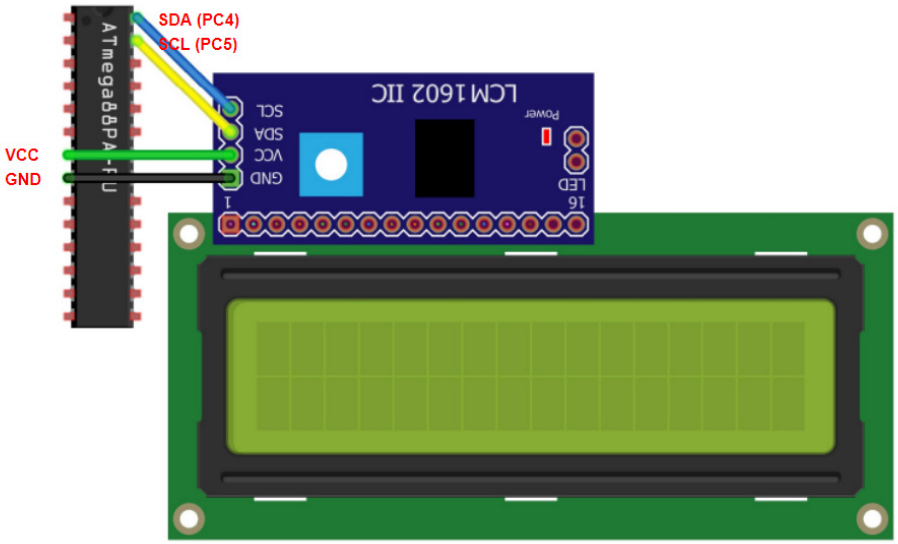

Abbildung 3.4: Verbindung des LCD-Displays mit dem Mikrocontroller

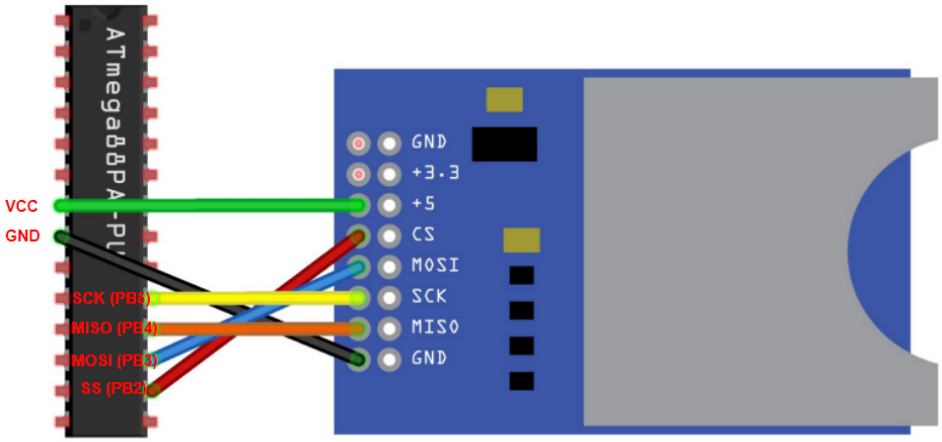

Abbildung 3.5: Belegung der Kontakte einer SD-Karte und deren Anschluss an einen Mikrocontroller im SPI-Modus

Display wird verwendet, um dem Benutzer Feedback zu geben, was die Benutzerfreundlichkeit erhöht.

Eine wichtige Funktion der Firmware ist die Speicherung der letzten Schreibadresse im EEPROM. Dies ermöglicht es dem System, nach einem Neustart nahtlos fortzufahren und gewährleistet eine kontinuierliche Datenaufzeichnung ohne Datenverlust.

Die Firmware beinhaltet außerdem Mechanismen zur Fehlerbehandlung, insbesondere bei der Initialisierung der SD-Karte und beim Empfang der GPS-Daten. Fehlermeldungen werden auf dem LCD-Display angezeigt, um den Benutzer über den Status zu informieren.

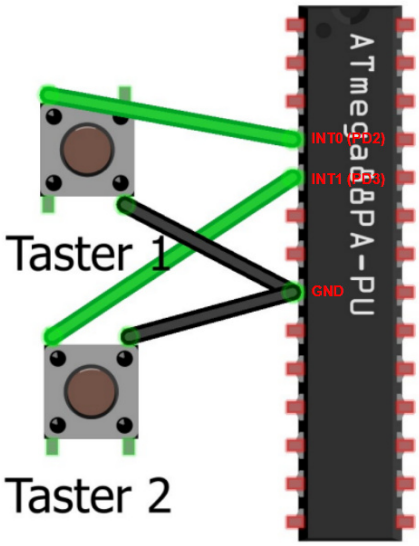

Abbildung 3.6: Verbindung der Taster mit dem Mikrocontroller

Die Interrupt Service Routinen (ISRs) spielen eine entscheidende Rolle bei der Interaktion des Benutzers mit dem System. Durch die Verwendung von Hardware-Interrupts ermöglichen diese Routinen eine sofortige Reaktion auf Benutzereingaben. Die `ISR(INT0_vect)` ist zuständig für die Behandlung von Tastendrücken, die den Messmodus betreffen, und implementiert eine Entprellung, um sicherzustellen, dass Tastendrücke korrekt interpretiert werden. Die `ISR(INT1_vect)` behandelt Tastendrücke, die den Lesemodus steuern, und ermöglicht es dem Benutzer, zwischen dem Lesemodus und dem normalen Betriebsmodus zu wechseln. In beiden Routinen wird eine Verzögerung von 20 Millisekunden eingeführt, um das Prellen der Tasten zu minimieren. Dies ist ein wichtiger Aspekt, da das Prellen zu falschen oder mehrfachen Aktivierungen des Interrupts führen kann, was wiederum die Systemleistung und Benutzererfahrung beeinträchtigen könnte.

Außerdem verwendet das Mikrocontroller-Firmware mehrere extern Bibliotheken, um die Funktionalität zu erweitern und die Entwicklung zu erleichtern. Die `Uart.h` Bibliothek wird verwendet, um die UART-Kommunikation zu ermöglichen. Die `lcd.h` Bibliothek wird verwendet, um die LCD-Operationen zu ermöglichen. Die `spi.h` Bibliothek wird verwendet, um die SPI-Kommunikation zu ermöglichen. Die `sd_card.h` Bibliothek wird verwendet, um die SD-Karten-Operationen zu ermöglichen. Bibliothek aufgrund der übermäßigen Länge des Quellcodes nicht im Anhang aufgeführt. Trotzdem sind einige wichtige Funktionen von Bibliotheken in den folgenden Unterabschnitten beschrieben.

3.2.1 Beschreibung von `initializeSystem()`

Im Folgenden wird die Funktionsweise der `initializeSystem()` Funktion detailliert beschrieben. Diese Funktion spielt eine zentrale Rolle bei der Vorbereitung des Systems, indem sie verschiedene Hardwarekomponenten und Schnittstellen initialisiert und konfiguriert, um deren reibungslosen Betrieb zu gewährleisten. Die meisten Funktionen werden durch Aufrufen von externen Bibliotheken realisiert. Die Struktur und die spezifischen Aktionen, die während der Initialisierung durchgeführt werden, sind in Abbildung 3.7 visualisiert.

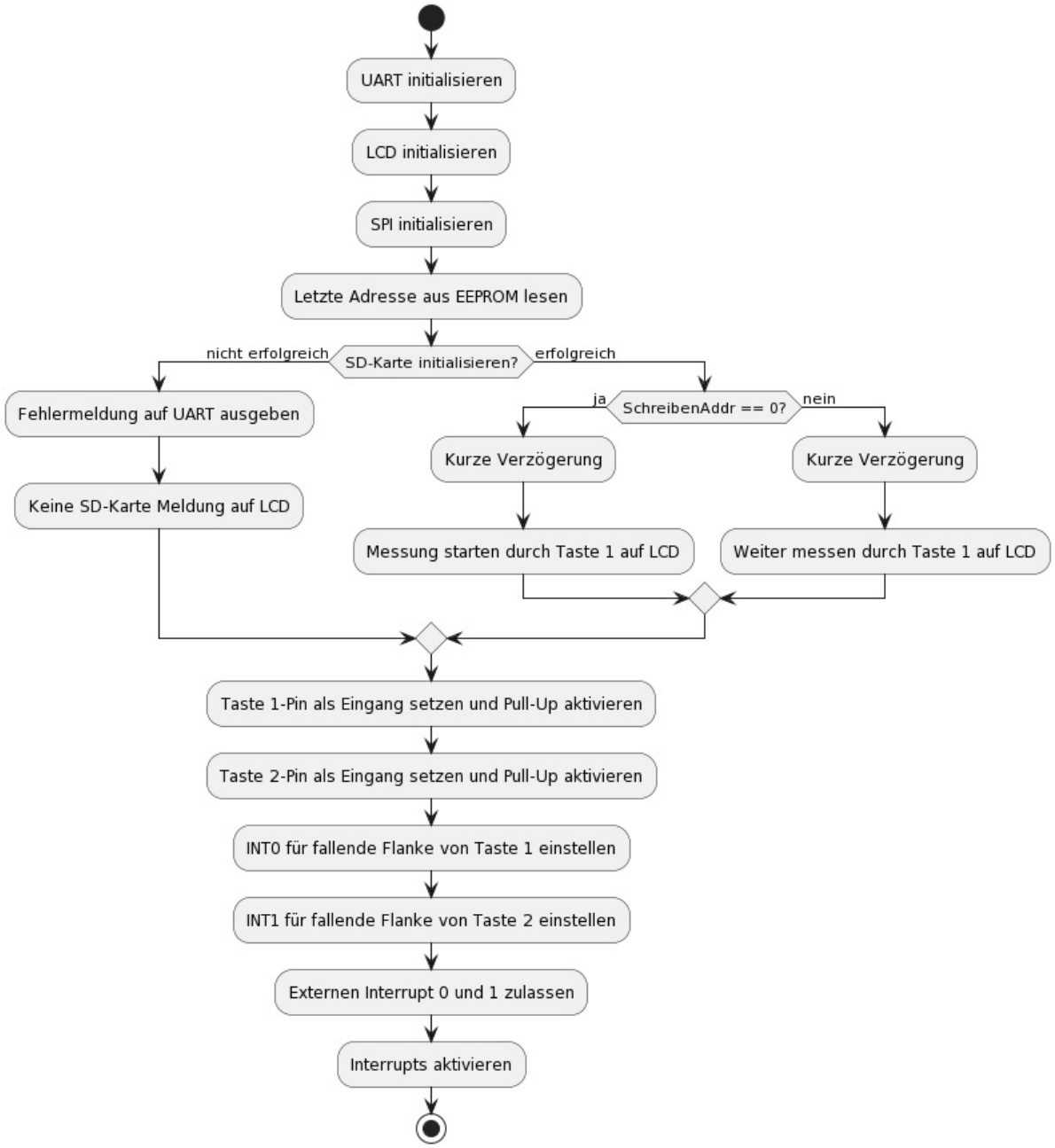

Abbildung 3.7: Struktur und Funktionsweise der `initializeSystem()` Funktion

Zeile 122-177 von Listing 5.1 in Anhang zeigt die Quellcode der `initializeSystem()`. Zunächst wird die UART-Kommunikationsschnittstelle initialisiert, um die serielle Kommunikation zwischen dem Mikrocontroller und externen Geräten wie PCs oder GPS-Modulen zu ermöglichen. Dies erfolgt durch Aufrufen der Funktion `uart_init()` mit spezifischen Parametern für Baudrate und CPU-Frequenz. Die Baudrate wird auf 9600L festgelegt, weil das GPS-Modul mit dieser Baudrate standardisiert. Außerdem wird CPU-Frequenz auf 3686400L festgelegt, weil die CPU-Frequenz des Mikrocontrollers 3.686.400 Hz beträgt.

Anschließend erfolgt die Initialisierung des LCD-Displays, das zur Anzeige von Informationen und Statusmeldungen dient. Die Funktion `lcd_init()` bereitet das Display vor und gewährleistet die korrekte Darstellung der Daten. Hier wird die Anzahl der Zeilen und Spalten des Displays festgelegt. In diesem Fall handelt es sich um ein 16x2-Zeichen-LCD-Display.

Die SPI-Schnittstelle wird konfiguriert, um die Kommunikation mit der SD-Karte zu ermöglichen. Durch den Aufruf von `SPI_init` wird der Mikrocontroller als SPI-Master festgelegt, und die erforderlichen Übertragungseinstellungen werden definiert. Die Geschwindigkeit der SPI-Kommunikation wird auf SPI_FOSC_16 festgelegt, um eine schnelle und zuverlässige Datenübertragung zu gewährleisten. SPI_FOSC_16 bedeutet, dass die SPI-Frequenz gleich 1/16 der CPU-Frequenz ist.

Ein wichtiger Schritt ist der EEPROM-Adresslesevorgang, bei dem die zuletzt im EEPROM gespeicherte Adresse ausgelesen wird. Diese Adresse ist entscheidend für die Fortführung der Datenerfassung nach einem Neustart.

Die Initialisierung der SD-Karte wird überprüft, und im Fehlerfall werden entsprechende Meldungen ausgegeben. Dies stellt sicher, dass die Datenspeicherung korrekt funktionieren kann.

Des Weiteren werden die Tasten und Interrupts konfiguriert. Die Pins für die Tasten werden als Eingänge mit aktivierten internen Pull-Up-Widerständen festgelegt, und die Interrupts INT0 und INT1 werden für die Erkennung von Tastendrücken konfiguriert. Abschließend wird durch den Befehl `sei()` die globale Aktivierung der Interrupts durchgeführt, was für die Reaktionsfähigkeit des Systems auf Benutzereingaben und externe Ereignisse unerlässlich ist.

Falls die Initialisierung erfolgreich war, wird eine entsprechende Meldung auf dem LCD-Display angezeigt, um den Benutzer über den erfolgreichen Start des Systems zu informieren. Abbildung 3.8 zeigt das LCD-Display mit der Meldung „Messung starten durch Taste 1“ nach einem erfolgreichen Umschalten in den Messmodus. Diese Meldung informiert den Benutzer darüber, dass das System bereit ist, die Datenerfassung zu starten.

Abbildung 3.8: Display mit der Meldung `Messung starten durch Taste 1`

Darüber hinaus zeigt Abbildung 3.9 das LCD-Display mit der Meldung „Weiter messen durch Taste 1“ nach einem erfolgreichen Umschalten in den Messmodus. Diese Meldung informiert den Benutzer darüber, dass das System bereit ist, die Datenerfassung fortzusetzen.

Abbildung 3.9: Display mit der Meldung `Weiter messen durch Taste 1`

Falls bei Initialisierung keine SD-Karte gefunden wurde, wird eine entsprechende Meldung auf dem LCD-Display angezeigt, um den Benutzer über den Fehler zu informieren. Abbildung 3.10 zeigt das LCD-Display mit der Meldung „Keine SD-Karte!“ nach einem erfolglosen Initialisierung der SD-Karte.

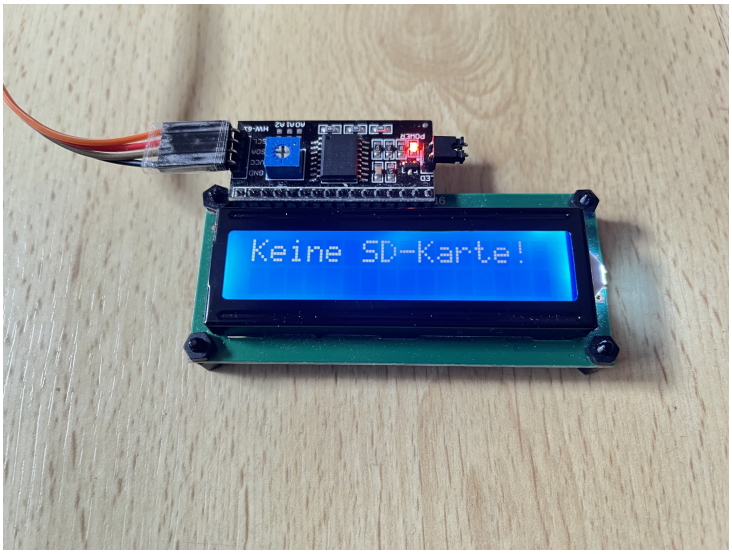

Abbildung 3.10: Display mit der Meldung `Keine SD-Karte!`

3.2.2 Beschreibung von `lesenSDCard()`

Die Funktion `lesenSDCard()` ist speziell dafür konzipiert, Daten von der SD-Karte zu lesen und sie über die UART-Schnittstelle auszugeben. Abbildung 3.11 zeigt die Struktur und Funktionsweise dieser Funktion.

Zeile 179-226 von Listing 5.1 in Anhang zeigt die Quellcode der `lesenSDCard()`. Zu Beginn des Prozesses wird das LCD-Display gelöscht und eine Nachricht („Lesen...“) angezeigt, um den Benutzer über den Beginn des Lesevorgangs zu informieren. Die Nachricht wird durch den Aufruf der Funktion `lcd_clear()` und `lcd_print()` auf dem LCD-Display angezeigt. Listing 3.1 und Listing 3.2 zeigt die Quellcode der `lcd_clear()` und `lcd_print()` Funktionen von Bibliothek `lcd.c`.

```
1 void lcd_clear() {
2     lcd_nibble_out(0x01, 0); // clear display
3     lcd_nibble_out(0x80, 0);
4     char_counter = 0;
5 }
```

Listing 3.1: Quellcode der `lcd_clear()` Funktion von Bibliothek `lcd.c`

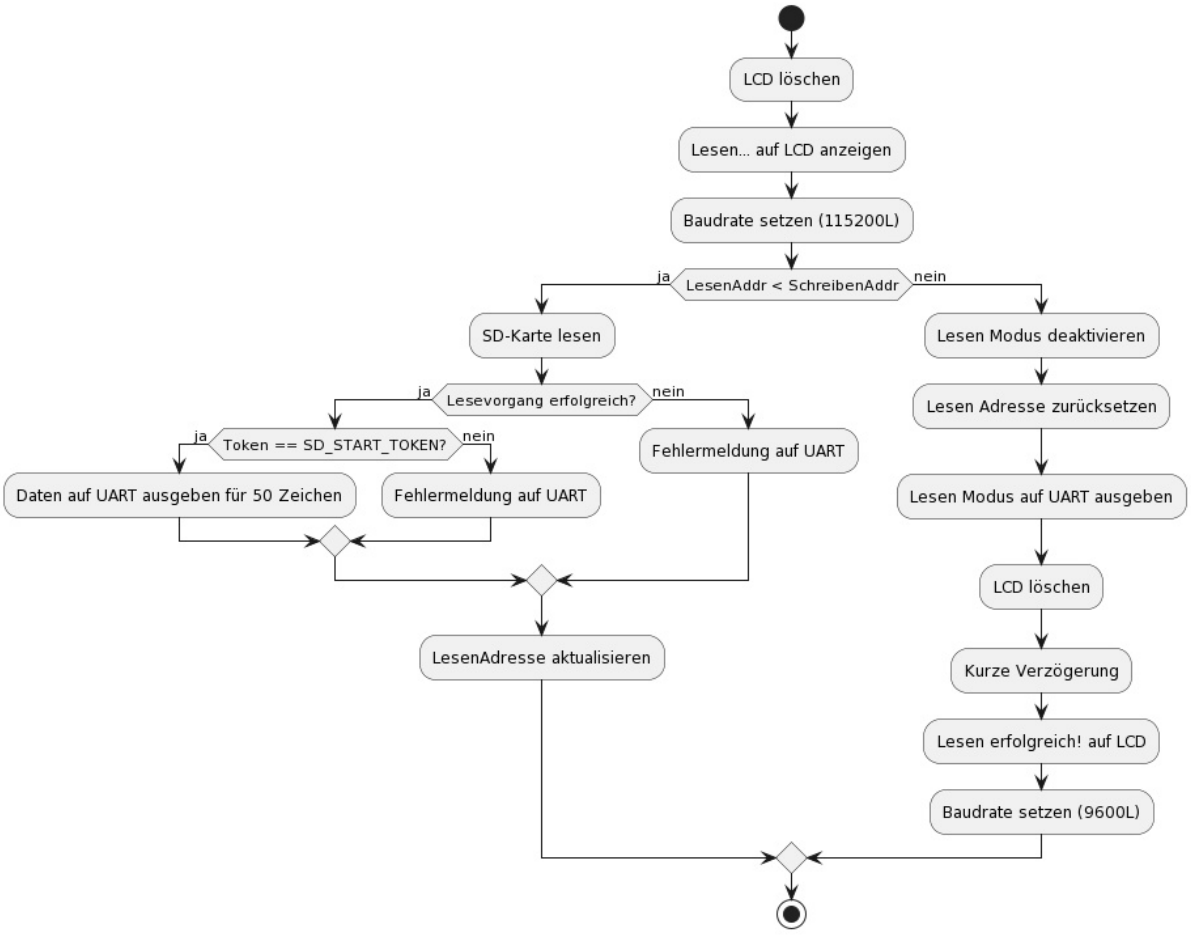

Abbildung 3.11: Struktur und Funktionsweise der `lesenSDCard()` Funktion

```
void lcd_print_str(char *str) {
  while (*str != 0){
    if(char_counter == LCD_WIDTH) lcd_nibble_out(LCD_ADDR_LINE2,0);
    if(char_counter == (LCD_WIDTH*2)){
      lcd_nibble_out(LCD_ADDR_LINE1,0);
      char_counter = 0;
    }
    char_counter++;
    lcd_nibble_out(*str++, 1);
  }
}
```

Listing 3.2: Quellcode der `lcd_print()` Funktion von Bibliothek `lcd.c`

Anschließend wird die Baudrate für die UART-Kommunikation auf 115200L gesetzt, um eine schnellere Datenübertragung zu ermöglichen. Dies erfolgt durch den Aufruf der Funktion `setBaudRate()` mit dem entsprechenden Parameter. Listing 3.3 zeigt die Quellcode der `setBaudRate()` Funktion.

```
void setBaudRate(unsigned long baud) {
  uart_init(UART_BAUD_SELECT(baud, F_CPU));
}
```

Listing 3.3: Quellcode der `setBaudRate()` Funktion

Ein wesentlicher Schritt ist die Überprüfung der Lese- und Schreibadressen, um sicherzustellen, dass nur die Daten gelesen werden, die bereits geschrieben wurden. Die Funktion prüft, ob die Adresse zum Lesen (`LesenAddr`) kleiner als die Adresse zum Schreiben (`SchreibenAddr`) ist. Dies verhindert das Lesen von unbeschriebenen Bereichen auf der SD-Karte.

Der eigentliche Lesevorgang wird durch die Funktion `SD_readSingleBlock()` von Bibliothek `sd_card.c` durchgeführt, die einen Block von der SD-Karte liest und die Daten in einem Puffer speichert. Die Ergebnisse des Lesevorgangs werden in einem Puffer (`buf1`) gespeichert. Die Funktion gibt auch einen Statuscode (`res1[0]`) und einen Token (`token1`) zurück, der den Beginn der Daten markiert. Listing 3.4 zeigt die Quellcode der `SD_readSingleBlock()` Funktion von Bibliothek `sd_card.c`.

```
/*****************************************************************************
 Read single 512 byte block
 token = 0xFE - Successful read
 token = 0x0X - Data error
 token = 0xFF - timeout
*****************************************************************************/
uint8_t SD_writeSingleBlock(uint32_t addr, uint8_t *buf, uint8_t *token)
{
    uint16_t readAttempts;
    uint8_t res1, read;

    // set token to none
    *token = 0xFF;

    // assert chip select
    SPI_transfer(0xFF);
```

```
17      CS_ENABLE();
18      SPI_transfer(0xFF);
19
20      // send CMD24
21      SD_command(CMD24, addr, CMD24_CRC);
22
23      // read response
24      res1 = SD_readRes1();
25
26      // if no error
27      if(res1 == SD_READY)
28      {
29          // send start token
30          SPI_transfer(SD_START_TOKEN);
31
32          // write buffer to card
33          for(uint16_t i = 0; i < SD_BLOCK_LEN; i++) SPI_transfer(buf[i]);
34
35          // wait for a response (timeout = 250ms)
36          readAttempts = 0;
37          while(++readAttempts != SD_MAX_WRITE_ATTEMPTS)
38              if((read = SPI_transfer(0xFF)) != 0xFF) { *token = 0xFF; break;
    }
39
40          // if data accepted
41          if((read & 0x1F) == 0x05)
42          {
43              // set token to data accepted
44              *token = 0x05;
45
46              // wait for write to finish (timeout = 250ms)
47              readAttempts = 0;
48              while(SPI_transfer(0xFF) == 0x00)
49                  if(++readAttempts == SD_MAX_WRITE_ATTEMPTS) { *token = 0x00
   ; break; }
50          }
51      }
52
53      // deassert chip select
```

```
54      SPI_transfer(0xFF);
55      CS_DISABLE();
56      SPI_transfer(0xFF);
57
58      return res1;
59  }
```

Listing 3.4: Quellcode der `SD_readSingleBlock()` Funktion von Bibliothek `sd_card.c`

Wenn der Lesevorgang erfolgreich war (indiziert durch `res1[0] == 0x00`) und der korrekte Start-Token empfangen wurde, werden die gelesenen Daten Zeichen für Zeichen über die UART-Schnittstelle ausgegeben. Nach erfolgreichem Lesen eines Blocks wird die Leseadresse (`LesenAddr`) um die Größe eines Sektors (512 KB) erhöht, um beim nächsten Lesevorgang den nächsten Block zu lesen. Dieser Schritt ist essentiell für die kontinuierliche Datenverarbeitung.

Der Lesevorgang endet, sobald die Leseadresse die Schreibadresse erreicht oder überschreitet, woraufhin der Lesemodus beendet und die Leseadresse zurückgesetzt wird. Abschließend wird eine abschließende Meldung auf dem LCD-Display angezeigt und die Baudrate wird auf den Standardwert (9600L) für das GPS-Modul zurückgesetzt, um die Kompatibilität mit anderen Modulen zu gewährleisten. Abbildung 3.12 zeigt das LCD-Display mit der Meldung „Lesen...“ nach einem erfolgreichen Umschalten in den Lesemodus. Diese Meldung informiert den Benutzer darüber, dass die Lesenvorgang jetzt durchgeführt wird.

Abbildung 3.12: Display mit der Meldung `Lesen...`

Falls alle Daten gelesen wurden, wird eine entsprechende Meldung auf dem LCD-Display angezeigt, um den Benutzer über den Abschluss des Lesevorgangs zu informieren. Abbildung 3.13 zeigt

das LCD-Display mit der Meldung „Lesen erfolgreich!“ nach einem erfolgreichen Lesevorgang.

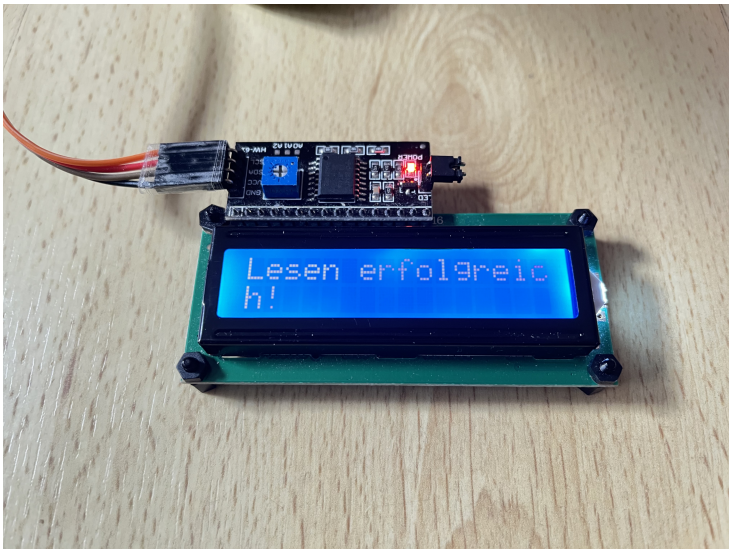

Abbildung 3.13: Display mit der Meldung `Lesen erfolgreich!`

3.2.3 Beschreibung von `abholenGPSDaten()`

Die Funktion `abholenGPSDaten()` ist für das Abholen, Analysieren und Verarbeiten der GPS-Daten verantwortlich, die von einem angeschlossenen GPS-Modul über die UART-Schnittstelle gesendet werden. Abbildung 3.14 zeigt die Struktur und Funktionsweise dieser Funktion.

Zeile 228-297 von Listing 5.1 in Anhang zeigt die Quellcode der `abholenGPSDaten()`. Zu Beginn wird mit der Funktion `uart_available()` geprüft, ob GPS-Daten über die UART-Schnittstelle verfügbar sind. Listing 3.5 zeigt die Quellcode der `uart0_available()` Funktion von Bibliothek `uart.c`.

```
1  /*************************************************************************
2  Function: uart0_available()
3  Purpose:  Determine the number of bytes waiting in the receive buffer
4  Input:    None
5  Returns:  Integer number of bytes in the receive buffer
6  *************************************************************************/
7  uint16_t uart0_available(void)
8  {
9    return (UART_RX0_BUFFER_SIZE + UART_RxHead - UART_RxTail) &
       UART_RX0_BUFFER_MASK;
10 }
```

Listing 3.5: Quellcode der `uart0_available()` Funktion von Bibliothek `uart.c`

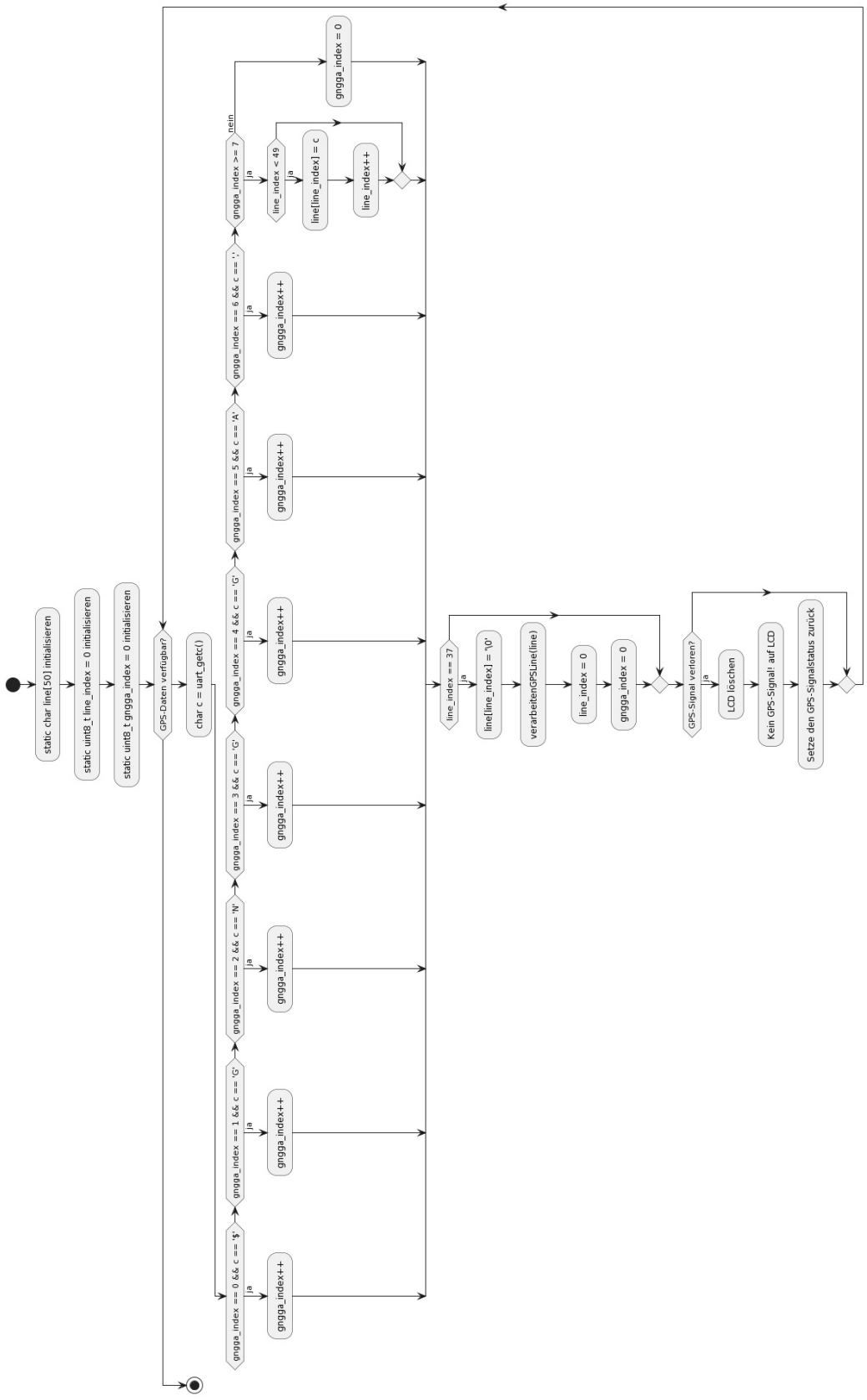

Abbildung 3.14: Struktur und Funktionsweise der `abholenGPSDaten()` *Funktion*

Sind Daten vorhanden, so werden sie sequenziell abgerufen, wobei ein besonderes Augenmerk auf der Identifikation der *GNGGA*- Zeile liegt. Diese Zeilen enthalten wichtige Informationen wie die aktuelle Zeit, Breiten- und Längengrade sowie die Fix-Qualität und sind daher für die Datenerfassung von besonderem Interesse. Nach erfolgreicher Identifikation der *GNGGA*-Zeile werden die nachfolgenden Zeichen in einem Array gespeichert, bis die gesamte Zeile vollständig ist. Dies wird durch eine Schleife realisiert, die die eingehenden Zeichen bis zu einer bestimmten Länge (37 Zeichen) speichert. Sobald eine vollständige *GNGGA*- Zeile empfangen wurde, wird diese an die Funktion `verarbeitenGPSLine()` im Unterabschnitt 3.2.4 übergeben. Diese Funktion ist dafür zuständig, die empfangenen Daten zu analysieren und für die weitere Verwendung im System aufzubereiten.

Nach Abschluss der Verarbeitung oder bei Auftreten eines Fehlers werden der Zeilenindex und der GNGGA-Index zurückgesetzt. Dieser Schritt stellt sicher, dass das System für die Verarbeitung der nächsten Zeile bereit ist, und ermöglicht eine kontinuierliche und effiziente Datenerfassung.

3.2.4 Beschreibung von `verarbeitenGPSLine()`

Die Funktion `verarbeitenGPSLine()` verarbeitet eine *GNGGA*- Zeile, extrahiert daraus wichtige Informationen wie Breiten- und Längengrad, Zeit und Fix-Status, und bereitet diese Daten zur Anzeige und Speicherung vor. Abbildung 3.15 zeigt die Struktur und Funktionsweise dieser Funktion.

Zeile 299-415 von Listing 5.1 in Anhang zeigt die Quellcode der `verarbeitenGPSLine()`. Die Funktion beginnt mit der Analyse der übergebenen Zeichenkette, die eine *GNGGA*- Zeile darstellt. Eine vollständige *GNGGA*- Zeile enthält 15 Segmente, die durch Kommas getrennt sind. Listing 3.6 zeigt ein Beispiel einer *GNGGA*- Zeile. Diese Zeile bedeutet, dass die aktuelle Zeit 16:50:06.000 ist, die Breitengrad 22 Grad 41 Minuten 91.07 Sekunden Nord, der Längengrad 120 Grad 17 Minuten 23.83 Sekunden Ost, der GPS-Fix-Status 1 ist, die Anzahl der Satelliten 14 ist, die HDOP 0.79 ist, die Höhe 22.6 Meter ist und die Geoidenhöhe 18.5 Meter ist.

```
1 $GNGGA,165006.000,2241.9107,N,12017.2383,E,1,14,0.79,22.6,M,18.5,M,,*42
```

Listing 3.6: Beispiel einer *GNGGA*- Zeile

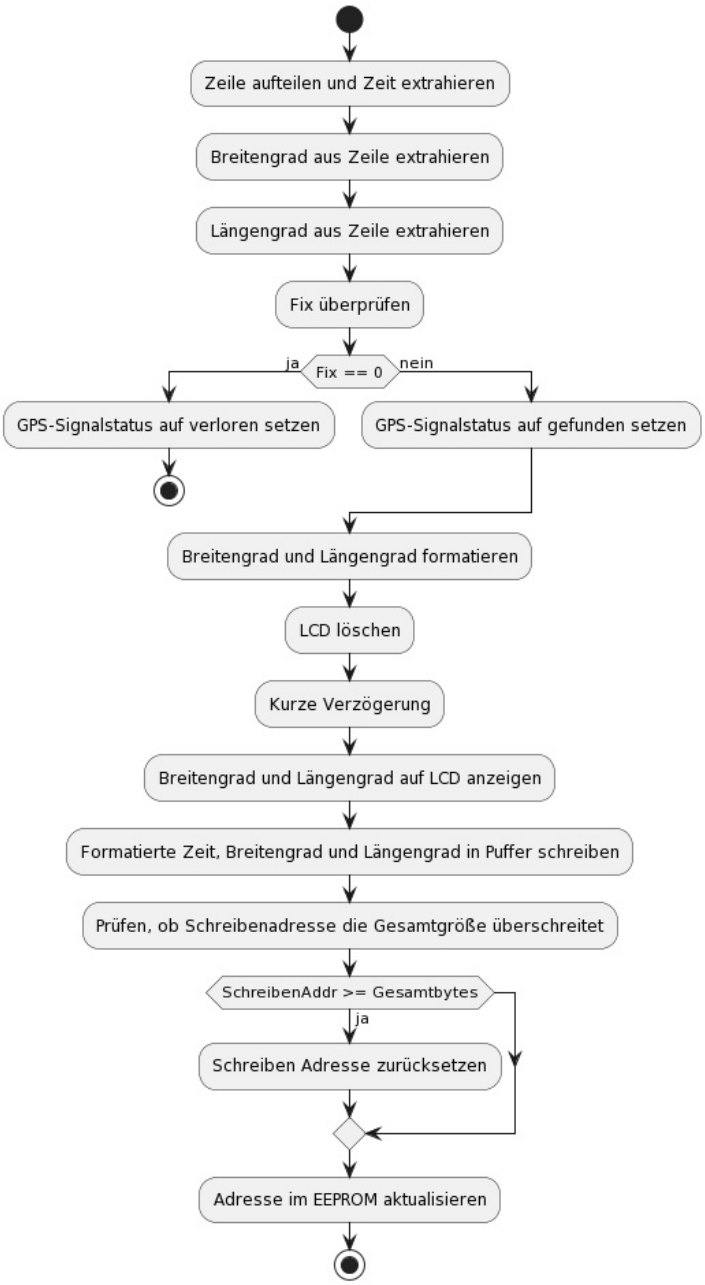

Abbildung 3.15: Struktur und Funktionsweise der `verarbeitenGPSLine()` Funktion

Mit Hilfe der Funktion `strtok()` wird diese Zeile in einzelne Segmente zerlegt, indem sie an Kommas getrennt wird. Anschließend wird die Zeitinformation extrahiert, indem die ersten sechs Zeichen der Zeile verwendet werden, um Stunden, Minuten und Sekunden zu bestimmen und in einem Zeitformat zusammenzufassen.

Für die Berechnung von Breiten- und Längengrad werden die Grad- und Minutenangaben aus der *GNGGA*- Zeile in ein Dezimalformat konvertiert. Dieser Schritt ist entscheidend, um die Daten in einem Format zu speichern, das für die weitere Verarbeitung und Anzeige geeignet ist. Die Funktion `atof()` wird verwendet, um die Grad- und Minutenangaben in Dezimalzahlen

umzuwandeln. Die Umrechnung von Breiten- und Längengrad ist bereits im Unterabschnitt 2.2.2 detailliert beschrieben. Zusätzlich werden die Zeichen für Norden/Süden und Osten/Westen extrahiert, um die geografische Lage genauer zu bestimmen.

Die Funktion prüft auch den GPS-Fix-Status, um sicherzustellen, dass ein gültiges GPS-Signal vorliegt. Bei einem fehlenden GPS-Fix wird der Signalstatus entsprechend gesetzt, und die Funktion endet ohne weitere Datenverarbeitung. Listing 3.7 zeigt die Quellcode zur Überprüfung des GPS-Fix-Status. Wenn der Fix 0 ist, bedeutet dies, dass es keine gültigen GPS-Daten gibt (0 = kein Fix, 1 = Fix).

```
if (fix == 0) {
    gpsSignalLost = true; // GPS-Signalstatus auf verloren setzen
    return; // Zurück
}

gpsSignalLost = false;
```

Listing 3.7: Quellcode zur Überprüfung des GPS-Fix-Status

Sollte das GPS-Signal verloren gehen, wird eine entsprechende Warnmeldung auf dem LCD-Display angezeigt, um den Benutzer über diesen Zustand zu informieren. Abbildung 3.16 zeigt das LCD-Display mit der Meldung „Kein GPS-Signal!“ nach einem Verlust des GPS-Signals.

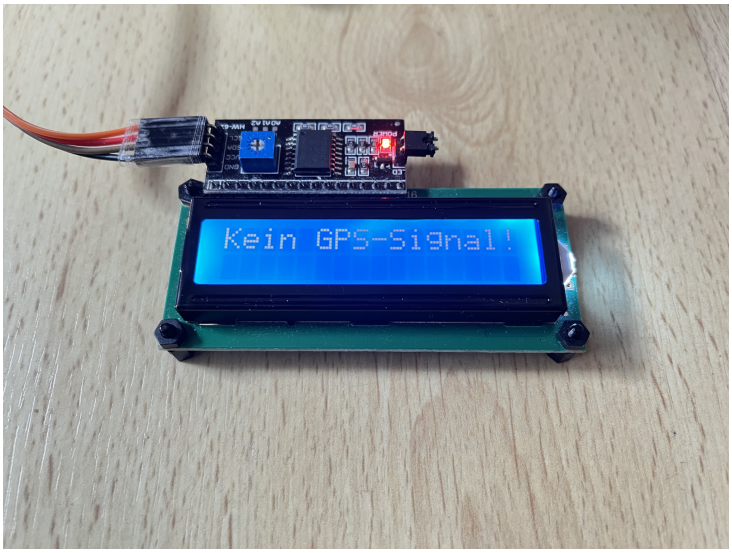

Abbildung 3.16: Display mit der Meldung `Kein GPS-Signal!`

Nach der erfolgreichen Verarbeitung werden die GPS-Daten auf einem LCD-Display dargestellt und in einem Puffer für die spätere Speicherung auf einer SD-Karte vorbereitet. Um die Daten auf dem LCD-Display besser darzustellen, werden die Daten in einem bestimmten Format

zusammengefasst und anschließend mit der Funktion `lcd_setcursor()` benutzt, um die Daten auf dem LCD-Display in zwei Zeilen anzuzeigen. Listing 3.8 zeigt die Quellcode der `lcd_setcursor()` Funktion von Bibliothek `lcd.c`.

```
void lcd_setcursor(uint8_t col, uint8_t row) {
uint8_t address;

/* compute the address according to the LCD layout */
switch (row) {
    case 0: address = 0x00 + col; break;  // first line
    case 1: address = 0x40 + col; break;  // second line
    // add more cases if your LCD has more lines
    default: return;  // invalid row
}

/* set the address counter to this address */
lcd_nibble_out(0x80 | address, 0);
}
```

Listing 3.8: Quellcode der `lcd_setcursor()` Funktion von Bibliothek `lcd.c`

Die aufbereiteten Daten werden schließlich auf das LCD-Display ausgegeben. Abbildung 3.17 zeigt das LCD-Display mit den GPS-Daten nach einer erfolgreichen Verarbeitung. Diese Darstellung informiert den Benutzer über die aktuellen GPS-Daten und ermöglicht eine visuelle Überprüfung der Daten.

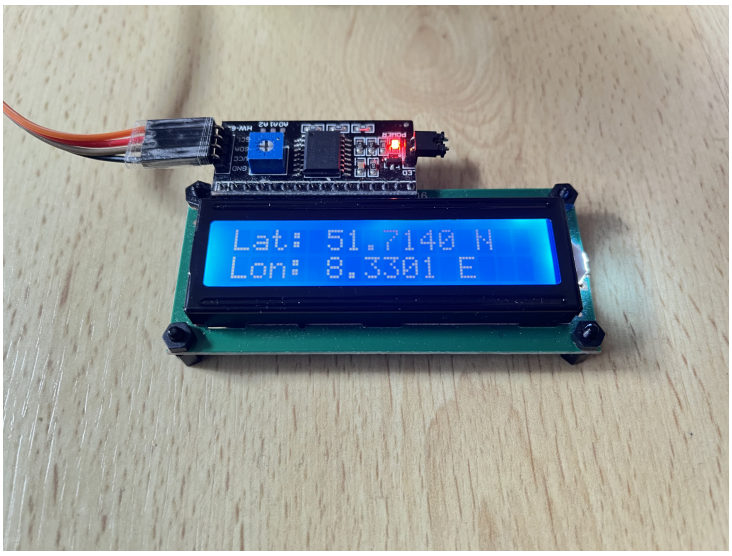

Abbildung 3.17: Display mit den GPS-Daten

Die gespeicherten Daten umfassen die aktuelle Zeit, den Breiten- und Längengrad sowie die entsprechenden Himmelsrichtungen. Der eigentliche Speichervorgang wird durch die Funktion `speichernSDCard()` von Bibliothek `sd_card.c` durchgeführt. Listing 3.9 zeigt die Quellcode der `speichernSDCard()` Funktion von Bibliothek `sd_card.c`.

```
/*************************************************************************
Write data to SD card
Write single 512 byte block
token = 0x00 - busy timeout
token = 0x05 - data accepted
token = 0xFF - response timeout
*************************************************************************/
uint8_t SD_writeSingleBlock(uint32_t addr, uint8_t *buf, uint8_t *token)
{
uint16_t readAttempts;
uint8_t res1, read;

// set token to none
*token = 0xFF;

// assert chip select
SPI_transfer(0xFF);
CS_ENABLE();
SPI_transfer(0xFF);

// send CMD24
SD_command(CMD24, addr, CMD24_CRC);

// read response
res1 = SD_readRes1();

// if no error
if(res1 == SD_READY)
{
    // send start token
    SPI_transfer(SD_START_TOKEN);

    // write buffer to card
    for(uint16_t i = 0; i < SD_BLOCK_LEN; i++) SPI_transfer(buf[i]);
```

```
35
36      // wait for a response (timeout = 250ms)
37      readAttempts = 0;
38      while(++readAttempts != SD_MAX_WRITE_ATTEMPTS)
39          if((read = SPI_transfer(0xFF)) != 0xFF) { *token = 0xFF; break; }
40
41      // if data accepted
42      if((read & 0x1F) == 0x05)
43      {
44          // set token to data accepted
45          *token = 0x05;
46
47          // wait for write to finish (timeout = 250ms)
48          readAttempts = 0;
49          while(SPI_transfer(0xFF) == 0x00)
50              if(++readAttempts == SD_MAX_WRITE_ATTEMPTS) { *token = 0x00;
    break; }
51      }
52  }
53
54  // deassert chip select
55  SPI_transfer(0xFF);
56  CS_DISABLE();
57  SPI_transfer(0xFF);
58
59  return res1;
60  }
```

Listing 3.9: Quellcode der `speichernSDCard()` Funktion von Bibliothek `sd_card.c`

Die aufbereiteten Daten werden schließlich auf die SD-Karte geschrieben. Dabei wird die Schreibadresse für den nächsten Schreibvorgang aktualisiert und im EEPROM gespeichert, um die Kontinuität der Datenerfassung zu sichern. Zusätzlich überwacht die Funktion die Speicherkapazität und setzt die Schreibadresse zurück, sollte das Ende des Speicherbereichs erreicht werden, um einen kontinuierlichen Betrieb des Systems zu gewährleisten. Durch Software HxD-Editor kann die gespeicherte Daten auf der SD-Karte überprüft werden. Abbildung 3.18 und Abbildung 3.19 zeigt zwei Sektor-Daten als Beispiel auf der SD-Karte gespeichert sind.

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Dekodierter Text
00000000   5A 65 69 74 3A 20 30 31 3A 31 38 3A 30 34 2C 20  Zeit: 01:18:04,      Sektor 0
00000010   4C 61 74 3A 20 33 30 2E 30 35 39 34 20 4E 2C 20  Lat: 30.0594 N,
00000020   4C 6F 6E 3A 20 31 32 30 2E 35 37 34 32 20 45 0D  Lon: 120.5742 E.
```

Abbildung 3.18: Sektor 0 auf der SD-Karte

```
00000200   5A 65 69 74 3A 20 30 31 3A 31 38 3A 30 36 2C 20  Zeit: 01:18:06,      Sektor 1
00000210   4C 61 74 3A 20 33 30 2E 30 35 39 31 20 4E 2C 20  Lat: 30.0591 N,
00000220   4C 6F 6E 3A 20 31 32 30 2E 35 37 34 30 20 45 0D  Lon: 120.5740 E.
```

Abbildung 3.19: Sektor 1 auf der SD-Karte

3.2.5 Beschreibung von `EEPROM_speicherAddress()`

Die Funktion `EEPROM_speicherAddress()` speichert eine Adresse im EEPROM des Mikrocontrollers und ermöglicht es, den Fortschritt der Datenspeicherung auf der SD-Karte auch nach einem Neustart des Systems nahtlos fortzusetzen.

Listing 3.10 zeigt die Quellcode der `EEPROM_speicherAddress()`. Zu Beginn der Funktion wird mittels der Funktion `eeprom_busy_wait()` sichergestellt, dass das EEPROM nicht durch andere Prozesse belegt ist.

```
1 void EEPROM_speicherAddress(uint32_t Addr) { // Adresse im EEPROM speichern
2 eeprom_busy_wait(); // Warten, dass EEPROM nicht besetzt ist
3 eeprom_update_block((const void*)&Addr, &speicher_Addr, sizeof(Addr)); //
     Adresse im EEPROM speichern
4 }
```

Listing 3.10: Quellcode der `EEPROM_speicherAddress()` Funktion

Sobald sichergestellt ist, dass das EEPROM verfügbar ist, wird die übergebene Adresse (`Addr`) an einer spezifischen Stelle im EEPROM gespeichert. Diese Aktion wird durch die Funktion `eeprom_update_block()` durchgeführt, die im Gegensatz zu `eeprom_write_block()` die vorhandenen Daten mit den neuen Daten vergleicht und nur schreibt, wenn ein Unterschied festgestellt wird. Diese Vorgehensweise reduziert den Verschleiß des EEPROMs, da unnötige Schreibvorgänge vermieden werden.

3.2.6 Beschreibung von `EEPROM_lesenAddress()`

Die Funktion `EEPROM_lesenAddress()` dient dazu, eine gespeicherte Adresse aus dem EEPROM des Mikrocontrollers auszulesen. Diese Adresse wird verwendet, um die Position zu bestimmen, an der die Datenspeicherung auf einem externen SD-Karte, fortgesetzt werden soll.

Listing 3.11 zeigt die Quellcode der `EEPROM_lesenAddress()`. Analog zur Funktion `EEPROM_speicherAddress()` wird zu Beginn der Funktion `eeprom_busy_wait()` aufgerufen, um sicherzustellen, dass das EEPROM zum Lesen bereit und nicht durch andere Prozesse belegt ist.

```
1 uint32_t EEPROM_lesenAddress(void) { // Adresse im EEPROM lesen
2 uint32_t Addr; // Adresse
3 eeprom_busy_wait(); // Warten, dass EEPROM nicht besetzt ist
4 eeprom_read_block((void*)&Addr, &speicher_Addr, sizeof(Addr)); // Adresse
     im EEPROM lesen
5 return Addr; // Adresse zurückgeben
6 }
```

Listing 3.11: Quellcode der `EEPROM_lesenAddress()` Funktion

Sobald das EEPROM als verfügbar bestätigt wurde, nutzt die Funktion `eeprom_read_block()`, um die am Speicherort `speicher_Addr` hinterlegte Adresse auszulesen. Diese ausgelesene Adresse wird dann in die Variable `Addr` übertragen. Nach dem erfolgreichen Auslesen der Adresse wird dieser Wert von der Funktion zurückgegeben. Diese Adresse wird im System dann genutzt, um den nächsten Schreibvorgang auf der SD-Karte an der korrekten Stelle fortzusetzen.

3.2.7 Beschreibung von `ISR(INT0_vect)`

Die Funktion `ISR(INT0_vect)`, speziell konzipiert für die Handhabung von Aktionen, die durch das Drücken des `BUTTON1_PIN` ausgelöst werden. Diese Funktion ermöglicht diese Funktion die Interaktion des Benutzers mit dem System, insbesondere das Umschalten zwischen dem Mess- und dem Lesemodus sowie die manuelle Initialisierung des Systems. Abbildung 3.20 zeigt die Struktur und Funktionsweise dieser Funktion.

Listing 3.12 zeigt die Quellcode der `ISR(INT0_vect)`. Zunächst wird durch eine kurze Verzögerung die Entprellung des Tasters sichergestellt. Diese Verzögerung (`_delay_ms(20)`) hilft, unerwünschte Signale durch das mechanische Prellen des Tasters zu eliminieren, was für die Zuverlässigkeit der Tastererkennung entscheidend ist.

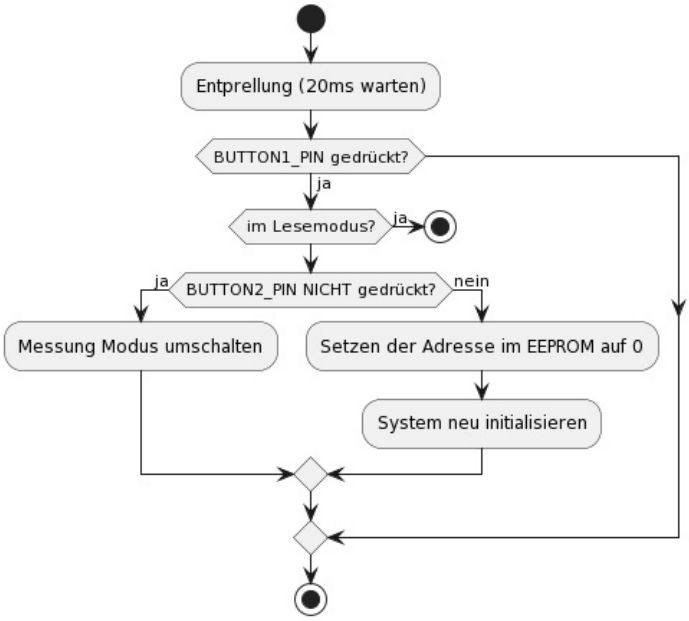

Abbildung 3.20: Struktur und Funktionsweise der `ISR(INT0_vect)` Funktion

```
ISR(INT0_vect) // INT0
{
_delay_ms(20); // Entprellung (20ms)
if (!(PIND & (1 << BUTTON1_PIN))) { // Prüfen, ob BUTTON1_PIN gedrückt ist (0 = gedrückt, 1 = nicht gedrückt)
    if (Lesen_modus) { // Wenn im Lesemodus
        return; // Zurück
    }
    if (PIND & (1 << BUTTON2_PIN)) { // Prüfen, ob BUTTON2_PIN NICHT gedrückt ist (0 = nicht gedrückt, 1 = gedrückt)
        Messung_modus = !Messung_modus; // Messung Modus umschalten (0 = AUS, 1 = AN)
        } else { // Wenn BUTTON2_PIN auch gedrückt ist (0 = gedrückt, 1 = nicht gedrückt)
        EEPROM_speicherAddress(0x00000000); // Setzen der zuletzt gespeicherten Adresse auf 0x00000000 während der manuellen Initialisierung
        initializeSystem(); // System neu initialisieren
    }
}
}
```

Listing 3.12: Quellcode der `ISR(INT0_vect)` Funktion

Im Anschluss prüft die ISR den Zustand des Tasters. Sollte der Taster, der den Interrupt ausgelöst hat, aktiv sein, wird untersucht, ob sich das System im Lesemodus befindet. Um Interferenzen während des Leseprozesses zu vermeiden, wird in diesem Modus keine weitere Aktion durchgeführt, und die ISR wird beendet.

Falls `BUTTON1_PIN` gedrückt und das System befindet sich nicht im Lesemodus, wird überprüft, ob `BUTTON2_PIN` nicht gedrückt ist. In diesem Fall wird der Messmodus (`Messung_modus`) umgeschaltet. Dies erlaubt es, den Messvorgang zu starten oder zu stoppen.

Ein besonderes Feature ist die Möglichkeit zur Systeminitialisierung: Wenn zusätzlich zum `BUTTON1_PIN` auch der `BUTTON2_PIN` gedrückt, so führt dies zur Initialisierung des Systems. Dabei wird zuerst die Adresse im EEPROM auf `0x00000000` gesetzt, was für eine manuelle Initialisierung steht. Anschließend wird das System durch Aufruf von `initializeSystem()` in Unterabschnitt 3.2.1 neu initialisiert.

3.2.8 Beschreibung von `ISR(INT1_vect)`

Die Funktion `ISR(INT1_vect)`, speziell konzipiert für die Handhabung von Aktionen, die durch das Drücken des `BUTTON2_PIN` ausgelöst werden. Diese Funktion ermöglicht diese Funktion die Interaktion des Benutzers mit dem System, insbesondere das Umschalten zwischen dem Mess- und dem Lesemodus sowie die manuelle Initialisierung des Systems. Abbildung 3.21 zeigt die Struktur und Funktionsweise dieser Funktion.

Listing 3.13 zeigt die Quellcode der `ISR(INT1_vect)`. Ähnlich wie bei der zuvor beschriebenen ISR in Unterabschnitt 3.2.7 beginnt auch diese Routine mit einer Entprellungsphase `_delay_ms(20)`, um zuverlässige Signale zu gewährleisten und Fehlauslösungen durch das mechanische Prellen des Tasters zu verhindern.

```
1 ISR(INT1_vect)
2 {
3 _delay_ms(20); // Entprellung (20ms)
4 if (!(PIND & (1 << BUTTON2_PIN))) { // Prüfen, ob BUTTON2_PIN gedrückt ist
    (0 = gedrückt, 1 = nicht gedrückt)
5     if (Messung_modus) { // Wenn im Messmodus
6         return; // Zurück
7     }
8     if (Lesen_modus) { // Wenn im Lesemodus, verhindere Reinitialisierung
```

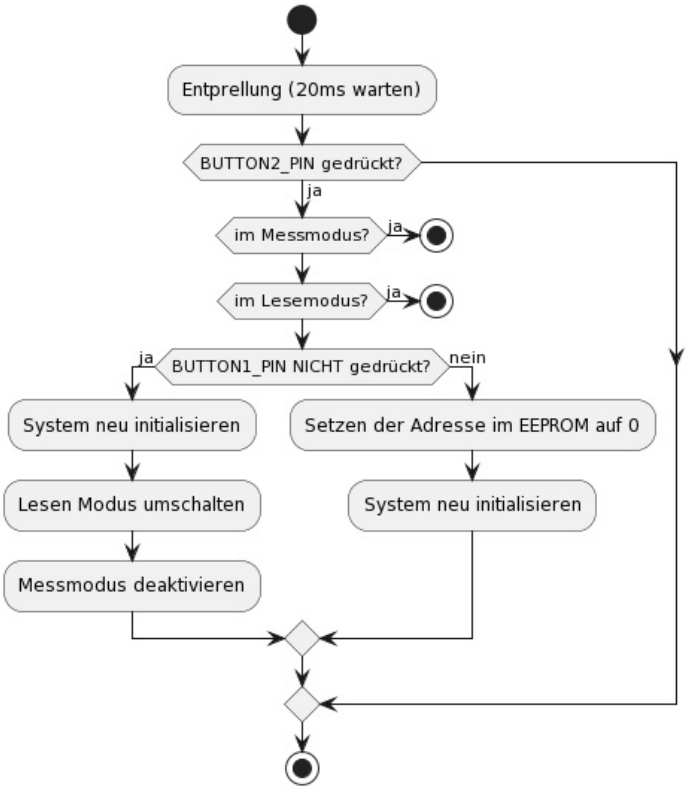

Abbildung 3.21: Struktur und Funktionsweise der `ISR(INT1_vect)` Funktion

```
9           return; // Zurück
10      }
11      if (PIND & (1 << BUTTON1_PIN)) { // Prüfen, ob BUTTON1_PIN NICHT gedrü
    ckt ist (0 = nicht gedrückt, 1 = gedrückt)
12          initializeSystem(); // System neu initialisieren
13          Lesen_modus = !Lesen_modus; // Lesen Modus umschalten (0 = AUS, 1 =
     AN)
14          Messung_modus = 0; // Wenn im Lesemodus, deaktivieren des Messmodus
     (0 = AUS, 1 = AN)
15          } else { // Wenn BUTTON1_PIN auch gedrückt ist (0 = gedrückt, 1 =
    nicht gedrückt)
16          EEPROM_speicherAddress(0x00000000); // Setzen der zuletzt
    gespeicherten Adresse auf 0x00000000 während der manuellen
    Initialisierung
17          initializeSystem(); // System neu initialisieren
18      }
19  }
20  }
```

Listing 3.13: Quellcode der `ISR(INT1_vect)` Funktion

Im Anschluss wird der Zustand von `BUTTON2_PIN` überprüft. Falls dieser gedrückt ist, evaluiert die ISR, ob sich das System in einem speziellen Modus befindet, wie dem Mess- oder Lesemodus. Sollte eine dieser Bedingungen erfüllt sein, wird die ISR ohne Ausführung weiterer Aktionen beendet, um die Integrität dieser Betriebsmodi zu wahren.

Falls der `BUTTON1_PIN` nicht aktiv ist, leitet die ISR eine Neuinitialisierung des Systems ein, gefolgt von einem Umschalten in den Lesemodus, sofern das System sich nicht bereits in einem spezifischen Modus befindet. Diese Aktion stellt sicher, dass der Lesemodus aktiviert wird, während gleichzeitig der Messmodus deaktiviert bleibt, um eine klare Trennung der Funktionalitäten zu gewährleisten.

Eine manuelle Initialisierung des Systems wird durchgeführt, sowohl `BUTTON1_PIN` als auch `BUTTON2_PIN` gedrückt sind, wird das System manuell initialisiert. Dabei wird die im EEPROM gespeicherte Adresse auf `0x00000000` zurückgesetzt und das System neu gestartet.

Bis hierhin wurden die wichtigsten Funktionen und Routinen des Mikrocontroller-Programms auf einem ATmega88PA Mikrocontroller entwickelt. Diese Funktionen ermöglichen die effiziente Handhabung von GPS-Daten, die serielle Kommunikation mit einem GPS-Modul, die Speicherung von Daten auf einer SD-Karte und die Interaktion mit dem Benutzer über Tasten. Im nächsten Abschnitt wird die Entwicklung der PC-Anwendung zur Verarbeitung und Speicherung von GPS-Daten in einer GPX-Datei beschrieben.

3.3 Entwicklung der PC-Anwendung

Die Entwicklung der PC-Anwendung für die Verarbeitung und Speicherung von GPS-Daten in einer GPX-Datei erfolgte in der Entwicklungsumgebung Visual Studio Community 2022. Das Hauptziel der Anwendung ist die effiziente Handhabung von Daten, die über einen COM-Port von einem GPS-Empfänger empfangen werden. Die Software ist in der Lage, die empfangenen Daten in ein spezifisches Format zu konvertieren und sie in einer GPX-Datei zu speichern, welche für die weitere Verwendung in Karte geeignet ist. Die Anwendung wurde in C++ entwickelt und nutzt die Windows-API für die serielle Kommunikation und die Dateiverwaltung.

Im Folgenden wird die Softwareentwicklung und Funktionalität der PC-Anwendung eingehend erläutert. Listing 5.2 in Anhang zeigt den vollständigen Quellcode der PC-Anwendung, der in Visual Studio Community 2022 entwickelt wurde. Die Anwendung beginnt mit der Einbindung

der erforderlichen Header-Dateien, die für die serielle Kommunikation und die Dateiverwaltung benötigt werden. Dazu gehören `windows.h`, `iostream`, `fstream`, `string`, `ctime` und `Shlobj.h`. Die Windows-API wird für die serielle Kommunikation und die Dateiverwaltung genutzt, während die anderen Header-Dateien für die Verarbeitung und Speicherung von GPS-Daten in einer GPX-Datei benötigt werden.

Zunächst setzt die Anwendung die Locale-Einstellungen durch den Aufruf der Funktion `setlocale(LC_ALL, )`, um eine korrekte Darstellung von Umlauten und Sonderzeichen wie „ä“, „ö“, „ü“ und „ß“ zu gewährleisten. Der Benutzer wird daraufhin aufgefordert, die Nummer des zu verwendenden COM-Ports anzugeben, was für die Kommunikation mit dem GPS-Empfänger entscheidend ist. Nach Eingabe der COM-Port-Nummer durch den Benutzer öffnet und konfiguriert das Programm diesen Port mit spezifischen Parametern wie Baudrate, Bytegröße, Stopbits und Parität, um eine stabile und korrekte Datenübertragung zu gewährleisten. Zeile 52-89 von Listing 5.2 zeigt die Quellcode zur Konfiguration des COM-Ports. Darin wird BaudRate auf 115200 gesetzt, um die Datenübertragungsgeschwindigkeit mit Mikrocontroller bei Lesenmodus zu synchronisieren.

Anschließend ermittelt die Software den Pfad zum Benutzerdokumente-Ordner und erstellt dort einen speziellen Ordner für die Speicherung der GPX-Dateien. Daraufhin generiert das Programm einen Dateinamen für die GPX-Datei, basierend auf dem aktuellen Datum und der Uhrzeit, um Eindeutigkeit zu gewährleisten. Das Herzstück der Anwendung ist die Datenerfassung vom COM-Port und deren anschließende Verarbeitung. Die empfangenen Daten werden kontinuierlich gelesen und relevante Informationen wie GPS-Positionen und Zeitstempel extrahiert und verarbeitet. Unterabschnitt 3.3.5 wird detailliert beschreiben, wie die GPS-Daten verarbeitet und in eine GPX-Datei gespeichert werden.

Die verarbeiteten GPS-Daten werden in das GPX-Format konvertiert und in der vorbereiteten Datei gespeichert, was für die spätere Nutzung der Daten in Karte entscheidend ist. Nach erfolgreicher Speicherung der Daten schließt das Programm die GPX-Datei und gibt den COM-Port frei, um keine Ressourcen unnötig zu belegen und den Port für andere Anwendungen verfügbar zu machen. Zum Abschluss informiert die Anwendung den Benutzer über den Speicherort der GPX-Datei, was ein wichtiger Schritt zur Erleichterung der Lokalisierung und anschließenden Verwendung der Datei ist. Unterabschnitt 3.3.6 wird detailliert beschreiben, wie die GPX-Datei zum Schluss bearbeitet werden.

3.3.1 Beschreibung von `COMDatenLesen()`

Die Funktion `COMDatenLesen(HANDLE hCom, std::string& daten)` ermöglicht das Lesen von Daten, die über einen COM-Port empfangen werden, und ist somit entscheidend für die Interaktion zwischen dem GPS-Empfänger und der PC-Anwendung. Abbildung 3.22 zeigt die Struktur und Funktionsweise dieser Funktion.

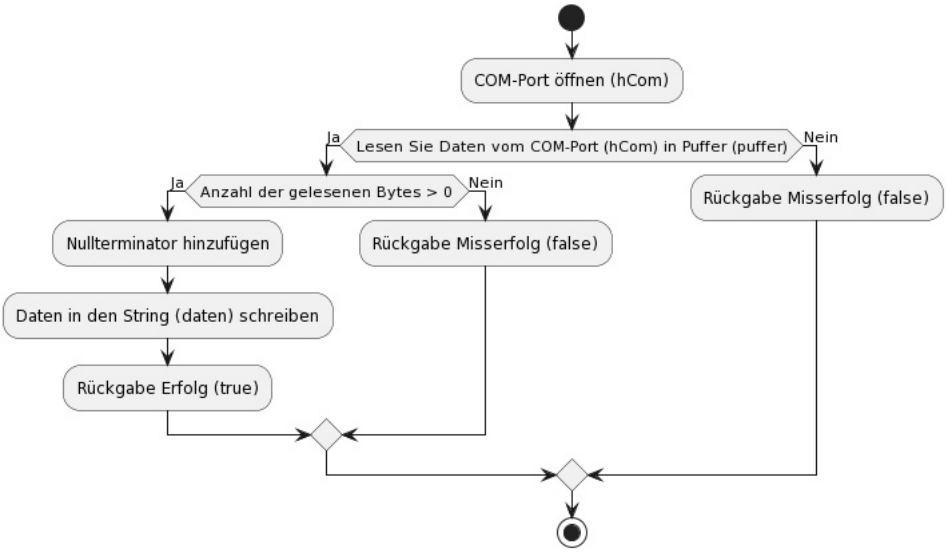

Abbildung 3.22: Struktur und Funktionsweise der `COMDatenLesen()` Funktion

Listing 3.14 zeigt den Quellcode der `COMDatenLesen()` Funktion. Zunächst prüft die Funktion die Gültigkeit des übergebenen COM-Port-Handles (`hCom`), um sicherzustellen, dass eine Verbindung zum COM-Port besteht und dieser bereit ist, Daten zu empfangen. Die Hauptaufgabe der Funktion ist es, Daten vom COM-Port unter Verwendung der Win32 API-Funktion `ReadFile()` zu lesen. Diese liest die Daten, die über den COM-Port gesendet werden, und speichert sie zunächst in einem temporären Puffer. Anschließend werden die Daten aus dem Puffer in den übergebenen String `daten` übertragen.

```
1  bool COMDatenLesen(HANDLE hCom, std::string& daten) {
2  char puffer[64]; // Puffer für die empfangenen Daten (64 Bytes)
3  DWORD geleseneBytes; // Anzahl der gelesenen Bytes
4  if (ReadFile(hCom, puffer, sizeof(puffer) - 1, &geleseneBytes, nullptr)
   && geleseneBytes > 0) { // Lesen Sie die Daten vom COM-Port
5      puffer[geleseneBytes] = '\0'; // Nullterminator hinzufügen
6      daten = puffer; // Daten in den String schreiben
7      return true; // Erfolg
8  }
9  return false; // Misserfolg
```

```
10      }
```

Listing 3.14: Quellcode der `COMDatenLesen` Funktion

Während des Leseprozesses wird kontinuierlich überprüft, ob Daten erfolgreich empfangen wurden, indem die Anzahl der gelesenen Bytes (`geleseneBytes`) überwacht wird. Bei erfolgreichem Empfang von Daten (d.h., die Anzahl der gelesenen Bytes ist größer als Null) wird der Inhalt des Puffers in den String `daten` kopiert. Die Funktion gibt einen booleschen Wert zurück, der den Erfolg des Lesevorgangs anzeigt. Dies erlaubt es der aufrufenden Funktion, entsprechend auf erfolgreiche oder fehlgeschlagene Lesevorgänge zu reagieren. Im Falle eines Fehlschlags wird ein Fehlerstatus zurückgegeben, der signalisiert, dass keine Daten gelesen wurden.

Abbildung 3.23 zeigt die Benutzeroberfläche der PC-Anwendung zur Konfiguration des COM-Ports. Der Benutzer kann hier die Nummer des zu verwendenden COM-Ports eingeben, um die Verbindung zum GPS-Empfänger herzustellen.

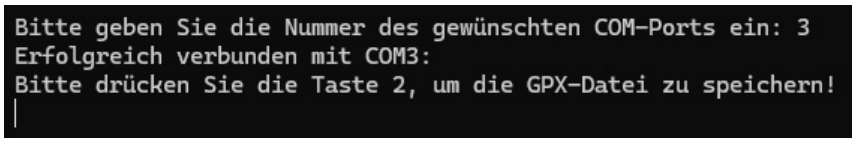

Abbildung 3.23: Benutzeroberfläche zur Konfiguration des COM-Ports

3.3.2 Beschreibung von `aktuellesDatumHolen()`

Die Funktion `aktuellesDatumHolen()` zielt darauf ab, das aktuelle Datum zu ermitteln und in einem standardisierten Format zurückzugeben. Ihre Rolle ist entscheidend für die Erstellung präziser Zeitstempel in den GPX-Daten. Abbildung 3.24 zeigt die Struktur und Funktionsweise dieser Funktion.

Der Prozess beginnt mit dem Abruf der aktuellen Systemzeit, wobei die C++ Standardbibliothek und die Funktion `time(nullptr)` zum Einsatz kommen, um die Zeit als `time_t`-Objekt zu erhalten. Dieses Objekt wird anschließend in eine `tm`-Struktur umgewandelt, die detaillierte Informationen über das Jahr, den Monat, den Tag und weitere Zeitkomponenten enthält. Diese Umwandlung erfolgt durch die Funktion `localtime_s()`.

Nach der Umwandlung in eine strukturierte Form wird das Datum mit der Funktion `strftime()` in ein standardisiertes Format gebracht, üblicherweise „JJJJ-MM-TT“ (Jahr-Monat-Tag). Das formatierte Datum wird dann als String zurückgegeben und kann in der Anwendung verwendet

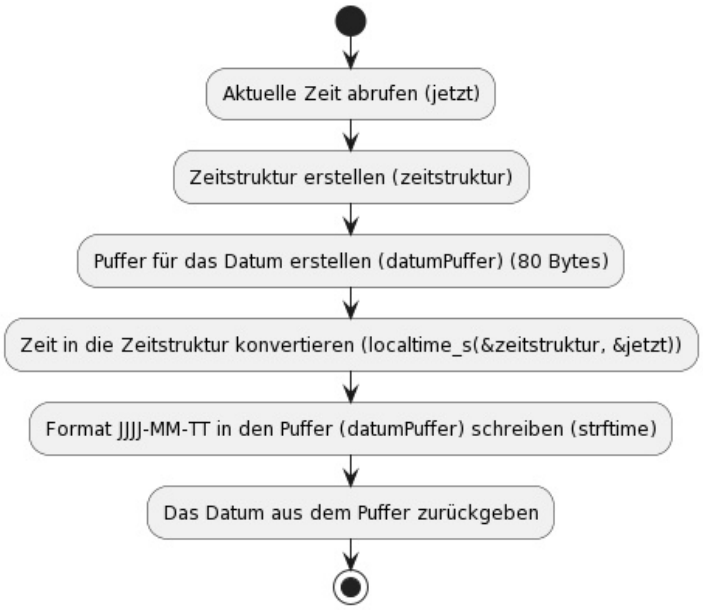

Abbildung 3.24: Struktur und Funktionsweise der `aktuellesDatumHolen()` Funktion

werden, um die Zeitstempel in den GPX-Dateien zu hinzufügen. Listing 3.15 zeigt den Quellcode der `aktuellesDatumHolen()` Funktion.

```
std::string aktuellesDatumHolen() {  // Aktuelles Datum abrufen
time_t jetzt = time(nullptr); // Aktuelle Zeit abrufen
struct tm zeitstruktur; // Zeitstruktur erstellen
char datumPuffer[80]; // Puffer für das Datum (80 Bytes)
localtime_s(&zeitstruktur, &jetzt); // Zeit in die Zeitstruktur
    konvertieren
strftime(datumPuffer, sizeof(datumPuffer), "%Y-%m-%d", &zeitstruktur); //
    Format: JJJJ-MM-TT
return datumPuffer; // Datum zurückgeben
}
```

Listing 3.15: Quellcode der `aktuellesDatumHolen` Funktion

3.3.3 Beschreibung von `aktuellesDatumUndUhrzeitHolen()`

Die Funktion `aktuellesDatumUndUhrzeitHolen()` hat die Aufgabe, sowohl das aktuelle Datum als auch die genaue Uhrzeit zu ermitteln und in einem spezifischen Format zurückzugeben. Die Bedeutung dieser Funktion erstreckt sich Benennung der GPX-Dateien. Abbildung 3.25 zeigt die Struktur und Funktionsweise dieser Funktion.

Der Prozess beginnt mit der Erfassung der aktuellen Systemzeit, wobei ein `time_t`-Objekt

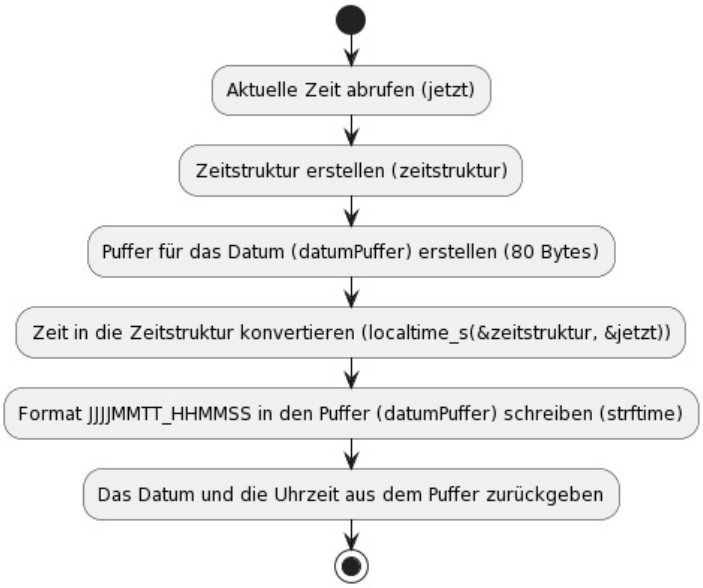

Abbildung 3.25: Struktur und Funktionsweise der `aktuellesDatumUndUhrzeitHolen()` Funktion

verwendet wird, um die gegenwärtige Zeit in Sekunden seit dem Unix-Epoch zu erhalten. Die erfasste Zeit wird dann in eine strukturierte, für Menschen lesbare Form umgewandelt, indem die Funktion `localtime_s()` genutzt wird, um die `time_t`-Zeit in eine `tm`-Struktur zu konvertieren. Diese Struktur beinhaltet detaillierte Informationen über das Datum und die Uhrzeit.

Anschließend wird das Datum und die Uhrzeit in ein spezifisches Format gebracht, typischerweise „JJJJMMTT_HHMMSS“ (JahrMonatTag_StundeMinuteSekunde), durch den Einsatz der Funktion `strftime()`. Nach der Formatierung gibt die Funktion das Datum und die Uhrzeit als String zurück, der für die Benennung der GPX-Dateien verwendet werden kann. Listing 3.16 zeigt den Quellcode der `aktuellesDatumUndUhrzeitHolen()` Funktion.

```
1 std::string aktuellesDatumUndUhrzeitHolen() { // Aktuelles Datum und
     Uhrzeit abrufen
2 time_t jetzt = time(nullptr); // Aktuelle Zeit abrufen
3 struct tm zeitstruktur; // Zeitstruktur erstellen
4 char datumPuffer[80];  // Puffer für das Datum (80 Bytes)
5 localtime_s(&zeitstruktur, &jetzt); // Zeit in die Zeitstruktur
     konvertieren
6 strftime(datumPuffer, sizeof(datumPuffer), "%Y%m%d_%H%M%S", &zeitstruktur);
      // Format: JJJJMMTT_HHMMSS
7 return datumPuffer; // Datum und Uhrzeit zurückgeben
8 }
```

Listing 3.16: Quellcode der `aktuellesDatumUndUhrzeitHolen` Funktion

3.3.4 Beschreibung von `BreitenLängengradKonvertieren()`

Die Funktion `BreitenLängengradKonvertieren()` besteht darin, die Rohdaten der GPS-Koordinaten, die in Form von Strings vorliegen, in ein standardisiertes Dezimalformat zu konvertieren. Diese Konvertierung ist entscheidend für die korrekte Darstellung und Weiterverarbeitung der geografischen Positionen. Abbildung 3.26 zeigt die Struktur und Funktionsweise dieser Funktion.

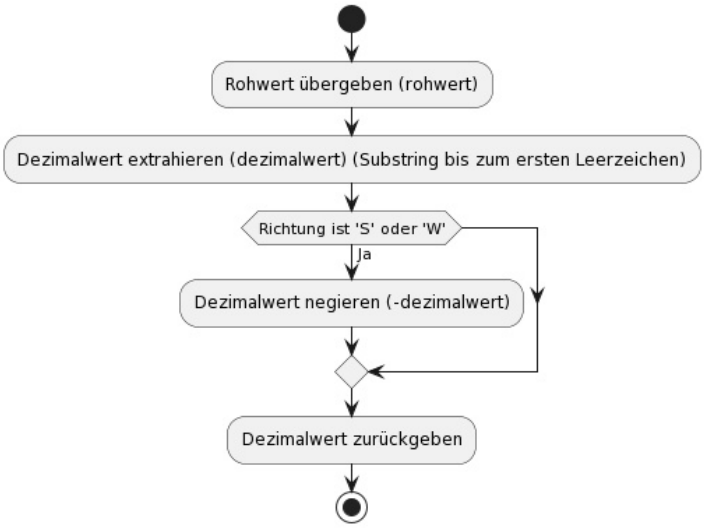

Abbildung 3.26: Struktur und Funktionsweise der `BreitenLängengradKonvertieren()` Funktion

Der Prozess beginnt mit der Entgegennahme von zwei Parametern: dem Rohwert der Koordinaten (`rohwert`) in einem nicht-standardisierten Format und der geografischen Richtung (`richtung`), die durch einen Charakter wie „N“ für Norden, „S“ für Süden, „E“ für Osten und „W“ für Westen angegeben wird. Die Funktion extrahiert zunächst den relevanten Teil des Rohwertstrings und bereitet ihn auf die Konvertierung vor. Die geografische Richtung bestimmt das Vorzeichen der konvertierten Koordinate, wobei für Breiten im Süden und Längen im Westen das Vorzeichen negativ wird. Anschließend konvertiert die Funktion die Koordinaten in das Dezimalgradformat, ein weit verbreitetes Format, das von den meisten geografischen Informationssystemen und GPS-Geräten genutzt wird. Die konvertierten Koordinaten werden als String im Dezimalgradformat zurückgegeben, was eine einfache und effiziente Weiterverarbeitung und Speicherung der GPS-Daten ermöglicht.

Besonders hervorzuheben ist hier die Anpassung die Anzahl der Ziffern für Breiten- und Längengrade. Dies bedeutet, dass die Anzahl der Ziffern für Breiten- und Längengrade für

verschiedene Orte auf der Welt unterschiedlich sind. Z.B ist die Längengrade in Shaoxing, China gegen `120.XXXX E`, während die Längengrade in Lippstadt, Deutschland gegen `8.XXXX E`. Listing 3.17 und Listing 3.18 zeigt, wie automatisch die Anzahl der Ziffern für Breiten- und Längengrade bestimmt wird, um immer 4 Dezimalstellen zu speichern. Die Anpassung der Anzahl der Ziffern für Breiten- und Längengrade ist entscheidend für die Genauigkeit der GPS-Daten. Wenn die Anzahl der Ziffern für Breiten- und Längengrade nicht richtig eingestellt ist, kann es zu einer falschen Positionierung auf der Karte führen.

```
size_t breiteEndePos = nachricht.find(' ', breitePos + 5); // Position des
    Leerzeichens nach der Breite
if (breiteEndePos == std::string::npos || breiteEndePos > laengePos) { //
    Wenn das Leerzeichen nicht gefunden wird oder die Position größer als
    die Länge ist
    breiteEndePos = laengePos; // Position der Länge
}
std::string rohBreite = nachricht.substr(breitePos + 5, breiteEndePos -
    breitePos - 5); // Rohwert der Breite extrahieren
```

Listing 3.17: Quellcode der `BreitengradKonvertieren` Funktion

```
size_t laengeEndePos = nachricht.find(' ', laengePos + 5); // Position des
    Leerzeichens nach der Länge
if (laengeEndePos == std::string::npos) { // Wenn das Leerzeichen nicht
    gefunden wird
    laengeEndePos = nachricht.length(); // Länge des Strings
}
std::string rohLaenge = nachricht.substr(laengePos + 5, laengeEndePos -
    laengePos - 5); // Rohwert der Länge extrahieren
```

Listing 3.18: Quellcode der `LängengradKonvertieren` Funktion

Zuerst wird die Position des Leerzeichens nach der Breite bzw. Länge gefunden, um die Anzahl der Ziffern für Breitengrade und Längengrade zu bestimmen. Wenn das Leerzeichen nicht gefunden wird, wird die Position der Länge des Strings verwendet. Anschließend wird der Rohwert der Breite bzw. Länge extrahiert. Die Anzahl der Ziffern für Breiten- und Längengrade wird dann automatisch bestimmt, um immer 4 Dezimalstellen zu speichern.

3.3.5 Erstellung von GPX-Datei

GPX, kurz für GPS Exchange Format, ist ein XML-Schema, das für den Austausch geografischer Informationen zwischen verschiedenen Systemen konzipiert wurde. Eine GPX-Datei besteht hauptsächlich aus Wegpunkten, Routen und Tracks, die geografische Standorte und Routeninformationen speichern [9]. Die folgenden Elemente sind typischerweise in einer GPX-Datei enthalten:

1. **GPX-Header:** Die GPX-Datei beginnt mit einem Header, der die Version des GPX-Schemas und die XML-Definitionen enthält. Der Header definiert auch die Struktur der Datei und die Art der enthaltenen Daten.

2. **Wegpunkte:** Ein Wegpunkt ist ein geografischer Punkt, der durch seine geografischen Koordinaten (Breiten- und Längengrad) definiert ist. Jeder Wegpunkt kann zusätzliche Informationen wie Name, Beschreibung, Höhe und Zeitstempel enthalten.

3. **Routen:** Eine Route ist eine geplante Reise oder ein Weg, der aus einer Reihe von Wegpunkten besteht. Jeder Wegpunkt in einer Route ist mit dem vorherigen und dem nächsten Wegpunkt verbunden, um eine zusammenhängende Route zu bilden.

4. **Tracks:** Ein Track ist eine aufgezeichnete Spur oder ein Pfad, der aus einer Reihe von Wegpunkten besteht. Jeder Wegpunkt in einem Track ist mit dem vorherigen und dem nächsten Wegpunkt verbunden, um eine kontinuierliche Spur zu bilden.

GPX wird hauptsächlich von GPS-Geräten und Softwareanwendungen zur Speicherung und zum Austausch von geografischen Informationen verwendet. Es ist wegen seiner Einfachheit und Effizienz bei der Speicherung von GPS-spezifischen Informationen weit verbreitet. Im Gegensatz zu KML (Keyhole Markup Language), einem anderen geografischen Dateiformat, das von Google Earth und Google Maps verwendet wird [15], ist GPX ein offenes Format, das von einer Vielzahl von Anwendungen und Geräten unterstützt wird.

Ein Beispiel für eine GPX-Datei ist in Listing 3.19 dargestellt. Diese Datei enthält einen Header und einen Track mit vier Wegpunkten. Darin wird <trkpt lat="51.6894" lon="8.3422"> als ein Wegpunkt definiert, der die geografischen Koordinaten 51.6894 Breitengrad und 8.3422 Längengrad enthält. Der <time> Tag enthält den Zeitstempel des Wegpunkts. Die <trkseg> und <trk> Tags definieren die Struktur des Tracks, der aus einer Reihe von Wegpunkten besteht.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<gpx xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.topografix.com/GPX/1/1" xsi:schemaLocation="http://www.topografix.com/GPX/1/1 http://www.topografix.com/GPX/1/1/gpx.xsd" version="1.1" creator="https://gpx.studio">
    <trk>
        <trkseg>
            <trkpt lat="51.6894" lon="8.3422">
                <time>2024-02-16T14:09:40Z</time>
            </trkpt>
            <trkpt lat="51.6894" lon="8.3422">
                <time>2024-02-16T14:10:01Z</time>
            </trkpt>
            <trkpt lat="51.6894" lon="8.3422">
                <time>2024-02-16T14:10:02Z</time>
            </trkpt>
            <trkpt lat="51.6894" lon="8.3422">
                <time>2024-02-16T14:10:03Z</time>
            </trkpt>
        </trkseg>
    </trk>
</gpx>
```

Listing 3.19: Beispiel einer GPX-Datei

Um die GPX-Datei zu speichern, wird zunächst einen Ordner mit dem Name „GXP_Datei“ im Benutzerdokumente-Ordner erstellt. Listing 3.20 zeigt den Quellcode, wie der Ordner erstellt werden. Dann wird eine GPX-Datei im erstellten Ordner erstellt. Das Name der GPX-Datei wird aus dem aktuellen Datum und der Uhrzeit generiert, um Eindeutigkeit zu gewährleisten. Z.B wird die GPX-Datei `Output_20240216_140940.gpx` genannt, wenn die Datei am 16. Februar 2024 um 14:09:40 Uhr erstellt wird. Listing 3.21 zeigt den Quellcode, wie die GPX-Datei erstellt werden.

```
char dokumentePfad[MAX_PATH]; // Puffer für den Dokumentenpfad (MAX_PATH Bytes)
HRESULT result = SHGetFolderPathA(NULL, CSIDL_PERSONAL, NULL, SHGFP_TYPE_CURRENT, dokumentePfad); // Benutzerdokumente-Ordner abrufen

if (!SUCCEEDED(result)) { // Wenn der Dokumentenpfad nicht abgerufen werden
```

```
    kann, wird eine Fehlermeldung ausgegeben
    std::cerr << "Fehler beim Abrufen des Dokumentenpfads" << std::endl; //
    Fehlermeldung
    return 1; // Beendet das Programm, wenn der Dokumentenpfad nicht
    abgerufen werden kann
}

std::string gxpOrdnerPfad = std::string(dokumentePfad) + "\\GXP_Datei"; //
    GXP_Datei-Ordnerpfad
```

Listing 3.20: Quellcode der `GPXOrdnerErstellen` Funktion

```
std::string datumUndUhrzeit = aktuellesDatumUndUhrzeitHolen(); // Aktuelles
    Datum und Uhrzeit abrufen
std::string dateiName = gxpOrdnerPfad + "\\Output_" + datumUndUhrzeit + ".
    gpx"; // Dateiname
```

Listing 3.21: Quellcode der `GPXDateiErstellen` Funktion

Die empfangenen Daten werden dann in die generierte GPX-Datei geschrieben. Um die empfangenen Daten besser lesen zu können, wird die empfangenen Daten in Terminal ausgegeben. Listing 3.22 zeigt den Quellcode, wie die empfangenen Daten in Terminal ausgegeben werden. Dann wird die empfangenen Daten mit dem Format von Listing 3.19 in die GPX-Datei geschrieben. Zeile 162-183 von Listing 5.2 zeigt den Quellcode, wie die empfangenen Daten in die GPX-Datei geschrieben werden.

```
std::cout << "Empfangene Daten: " << nachricht << std::endl; // Empfangene
    Daten im Terminal anzeigen
```

Listing 3.22: Quellcode der `TerminalAusgabe` Funktion

Schließlich, wenn die empfangenen Daten „Lesen: AUS“ sind, wird die GPX-Datei mit einem Befehl zum Schließen der Dateistreams ordnungsgemäß beendet. Das Unterabschnitt 3.3.6 wird die Abschlussfunktionen des Programms beschreiben.

3.3.6 Beschreibung von `abschluss`

Dieser Teil des Programms ist dafür verantwortlich, alle offenen Prozesse und Ressourcen korrekt zu schließen und sicherzustellen, dass die gesammelten Daten vollständig und korrekt gespeichert werden. Abbildung 3.27 zeigt die Struktur und Funktionsweise dieses Abschnitts.

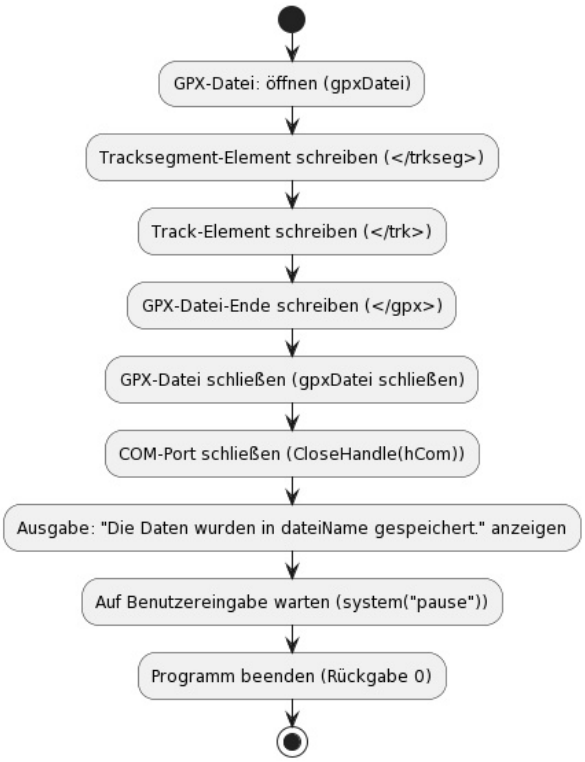

Abbildung 3.27: Struktur und Funktionsweise des **`abschluss`** *Abschnitts*

Der wichtigste Schritt ist das korrekte Schließen der GPX-Datei. Dies umfasst das Einfügen der abschließenden XML-Tags, die für das Format einer GPX-Datei erforderlich sind. Diese Schritte sind unerlässlich, um eine gültige und standardkonforme GPX-Datei zu gewährleisten, die von anderen Anwendungen und Geräten gelesen werden kann. Listing 3.23 zeigt den Quellcode des Abschnitts `abschluss`.

```
1  abschluss:
2  // Abschluss der GPX-Datei schreiben
3  gpxDatei << "        </trkseg>\n"; // Tracksegment-Element
4  gpxDatei << "    </trk>\n"; // Track-Element
5  gpxDatei << "</gpx>"; // GPX-Datei-Ende
6  gpxDatei.close(); // GPX-Datei schließen
7  CloseHandle(hCom); // COM-Port schließen
8
9  std::cout << "Die Daten wurden in " << dateiName << " gespeichert." << std
       ::endl; // Erfolgsmeldung
10 system("pause"); // Programm anhalten
11
12 return 0; // Programm beenden
```

Listing 3.23: Quellcode des Abschnitts **`abschluss`**

Nachdem alle notwendigen Daten in die GPX-Datei geschrieben wurden, wird die Datei mit einem Befehl zum Schließen der Dateistreams ordnungsgemäß beendet. Diese Aktion stellt sicher, dass alle geschriebenen Daten gespeichert und die Datei korrekt abgeschlossen wird, was einen eventuellen Datenverlust verhindert. Ein weiterer wesentlicher Schritt ist die Freigabe des COM-Ports. Dies wird erreicht, indem der COM-Port mit der Funktion `CloseHandle()` geschlossen wird, um sicherzustellen, dass der Port für andere Anwendungen verfügbar ist und keine Ressourcen unnötig belegt werden. Schließlich wird dem Benutzer eine Erfolgsmeldung angezeigt, die den Speicherort der GPX-Datei enthält. Das Programm wird angehalten, um dem Benutzer Zeit zu geben, die Meldung zu lesen, bevor es beendet wird. Abbildung 3.28 zeigt die Benutzeroberfläche der PC-Anwendung nach dem Abschluss des Datenverarbeitungsprozesses. Die Abbildung 3.29 zeigt den Inhalt der GPX-Datei „Output_20240216_143811.gpx“ als Beispiel.

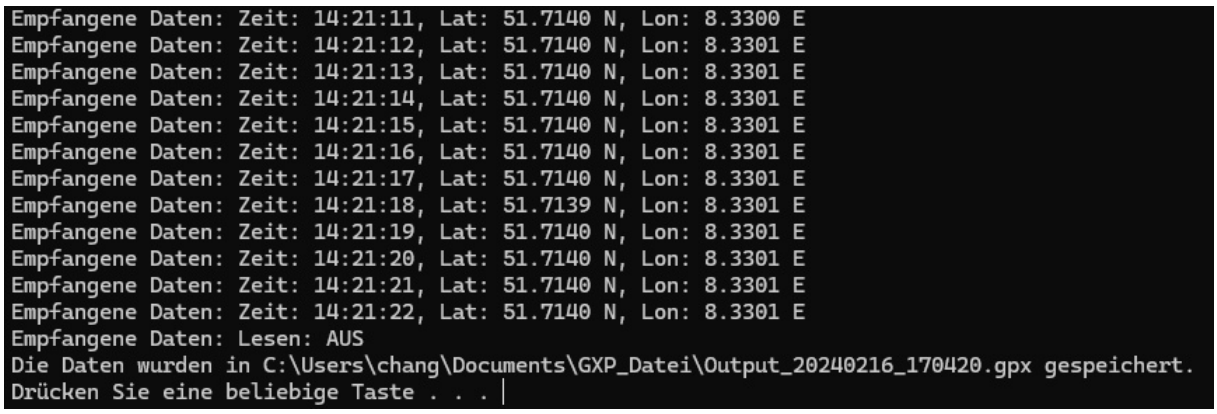

Abbildung 3.28: Benutzeroberfläche der PC-Anwendung nach Abschluss des Datenverarbeitungsprozesses

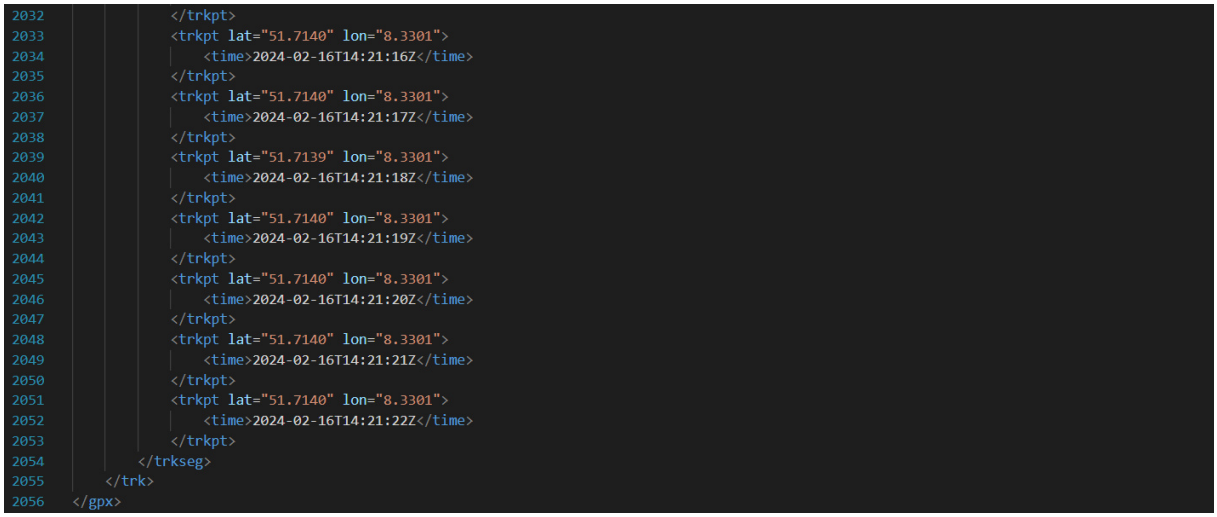

Abbildung 3.29: Inhalt der GPX-Datei „Output_20240216_143811.gpx“

Bis hierhin wurden die wichtigsten Aspekte der Softwareentwicklung und die Funktionalitäten der PC-Anwendung zur Verarbeitung und Speicherung von GPS-Daten entwickelt. Die beschriebenen Funktionen und Abläufe bilden das Fundament der Anwendung und ermöglichen eine effiziente Handhabung von GPS-Daten, die von UART-Schnittstelle des Mikrocontrollers gesendet werden.

4 Test und Ergebnisse

In diesem Abschnitt werden zuerst die Testmethoden und die Testergebnisse des GPS-Trackers beschrieben. Anschließend wird die Analyse der GPS-Daten in verschiedenen Situationen durchgeführt. Schließlich werden die Ergebnisse der Tests und Validierungen zusammengefasst.

4.1 Funktionstests

4.1.1 Vorgehensweise von Funktionstest

Um die Funktionalität des GPS-Trackers zu testen, werden verschiedene Tests durchgeführt, die auf spezifischen Anforderungen basieren. Zuerst wird jede Komponente des GPS-Trackers einzeln getestet, um sicherzustellen, dass sie ordnungsgemäß funktioniert.

Mikrocontroller-Funktionalitätstest: Ein Basisprogramm wird auf den ATmega88PA Mikrocontroller hochgeladen, um seine Funktionalität zu überprüfen. Dieses Programm könnte einfache Aufgaben ausführen, wie z.B. das Blinken einer LED.

Datenübertragungstest: Die Datenübertragung an einen PC wird getestet, indem einige Testdaten über die UART-Schnittstelle (RS232) an den PC gesendet werden. Die Korrektheit der übertragenen Daten wird durch das Terminalprogramm wie PuTTY überprüft.

Display-Test: Das Display wird getestet, indem einige Sätze oder Zahlen auf dem Display angezeigt werden. Die Korrektheit der Anzeige wird überprüft.

SD-Karten-Speicherfunktionstest: Daten werden auf der SD-Karte gespeichert und anschließend ausgelesen, um die Speicherfunktion zu überprüfen. Die Korrekigkeit der gespeicherten Daten wird durch das Software HxD überprüft.

GPS-Modul-Empfangstest: Das GPS-Modul wird getestet, indem es in einer offenen Umgebung platziert wird, um die GPS-Daten zu empfangen. Prüfen, ob das GPS-Modul die GPS-Koordinaten korrekt empfängt und per UART-Schnittstelle an den Mikrocontroller sendet. Die Korrekigkeit von GPS-Koordinaten wird durch Vergleich mit den tatsächlichen Koordinaten überprüft.

Anschließend wird die Gesamtfunktionalität des GPS-Trackers getestet, um sicherzustellen, dass alle Komponenten ordnungsgemäß miteinander interagieren. Die Tests umfassen die

Überprüfung der Tracking-Funktion mit Zeitintervallen, die Speicherfunktion auf der SD-Karte und die Datenübertragung an einen PC. Die PC-Anwendung für den Empfang und die Speicherung der Daten wird ebenfalls getestet.

4.1.2 Ergebnisse von Funktionstest

Das Hauptaugenmerk liegt hier auf den Funktionstests. Die geforderten Funktionstests stammen aus der Anforderungstabelle in Tabelle 1.1. Die Ergebnisse der Tests sind in Tabelle 4.1 aufgeführt. Zusammenfassend kann gesagt werden, dass alle Funktionstests erfolgreich durchgeführt wurden und die Anforderungen erfüllt sind.

Tabelle 4.1: GPS Tracker Testergebnisse

Test-Nr	Testbeschreibung	Ergebnis
1	Überprüfung der Funktionalität des ATmega88PA Mikrocontrollers.	Erfolgreich abgeschlossen
2	Test der Genauigkeit und Zuverlässigkeit des GPS-Moduls.	Genauigkeit und Zuverlässigkeit bestätigt
3	Überprüfung der Anzeige der GPS-Koordinaten auf dem Display.	GPS-Koordinaten werden ordnungsgemäß angezeigt
4	Test der Tracking-Funktion mit Zeitintervallen.	Koordinaten werden in den festgelegten Intervallen (1 Sekunde) aufgezeichnet und gespeichert
5	Test der Speicherfunktion auf der SD-Karte.	Daten können erfolgreich auf der SD-Karte gespeichert und ausgelesen werden
6	Überprüfung der Datenübertragung an einen PC.	Datenübertragung über UART-Schnittstelle erfolgreich
7	Test der PC-Anwendung für den Empfang und die Speicherung der Daten.	Software empfängt und speichert Daten im GPX-Format ordnungsgemäß

4.2 Demonstrationsbeispiel

Nachdem die Funktionalität des GPS-Trackers erfolgreich getestet wurde, wird der GPS-Tracker in der Praxis getestet, um die Genauigkeit und Zuverlässigkeit des GPS-Tracking-Systems zu überprüfen. Die folgenden Schritte beschreiben das Testverfahren und die Validierung der Funktionalitäten des GPS-Tracking-Systems in der Praxis:

Der GPS-Tracker wird an einem Fahrrad befestigt. Dann wird das Fahrrad in Freien (z.B. auf einer Straße) platziert. Nachdem das GPS-Modul die GPS-Koordinaten empfangen hat (Dies kann 30 Sekunden bis mehrere Minuten dauern, abhängig von der Umgebung). Zunächst das LCD Display wird geprüft, ob die GPS-Koordinaten korrekt angezeigt werden. Dann wird das Fahrrad für eine bestimmte Zeit bewegt. Während des Tests wird das GPS-Modul die GPS-Koordinaten in den festgelegten Zeitintervallen aufzeichnen.

Später werden die gespeicherten GPS-Koordinaten über die UART-Schnittstelle per PC-Anwendung empfangen und im GPX-Format gespeichert. Die GPX-Datei wird in Kartendiensten wie gpx.studio importiert, um die zurückgelegte Strecke anzuzeigen. Die aufgezeichneten GPS-Koordinaten werden mit einem Handy verglichen, um die Genauigkeit und Zuverlässigkeit der GPS-Daten zu bewerten.

4.2.1 Testverfahren und Validierung der Funktionalitäten

Abbildung 4.1 und Abbildung 4.2 zeigt die aufgezeichnete Strecke mit orange Farbe im gpx.studio. Das Fahrrad wurde für ca. 10 Minuten bewegt. Die Übertragung der GPS-Daten an den PC und die Speicherung der Daten im GPX-Datei dauerte ca. 1 Minute. Abbildung 4.3 zeigt die aufgezeichnete Strecke mit einem Handy. Vergleich der aufgezeichneten Strecke im gpx.studio und mit dem Handy zeigt, dass die GPS-Koordinaten fast exakt der tatsächlich zurückgelegten Strecke entsprechen.

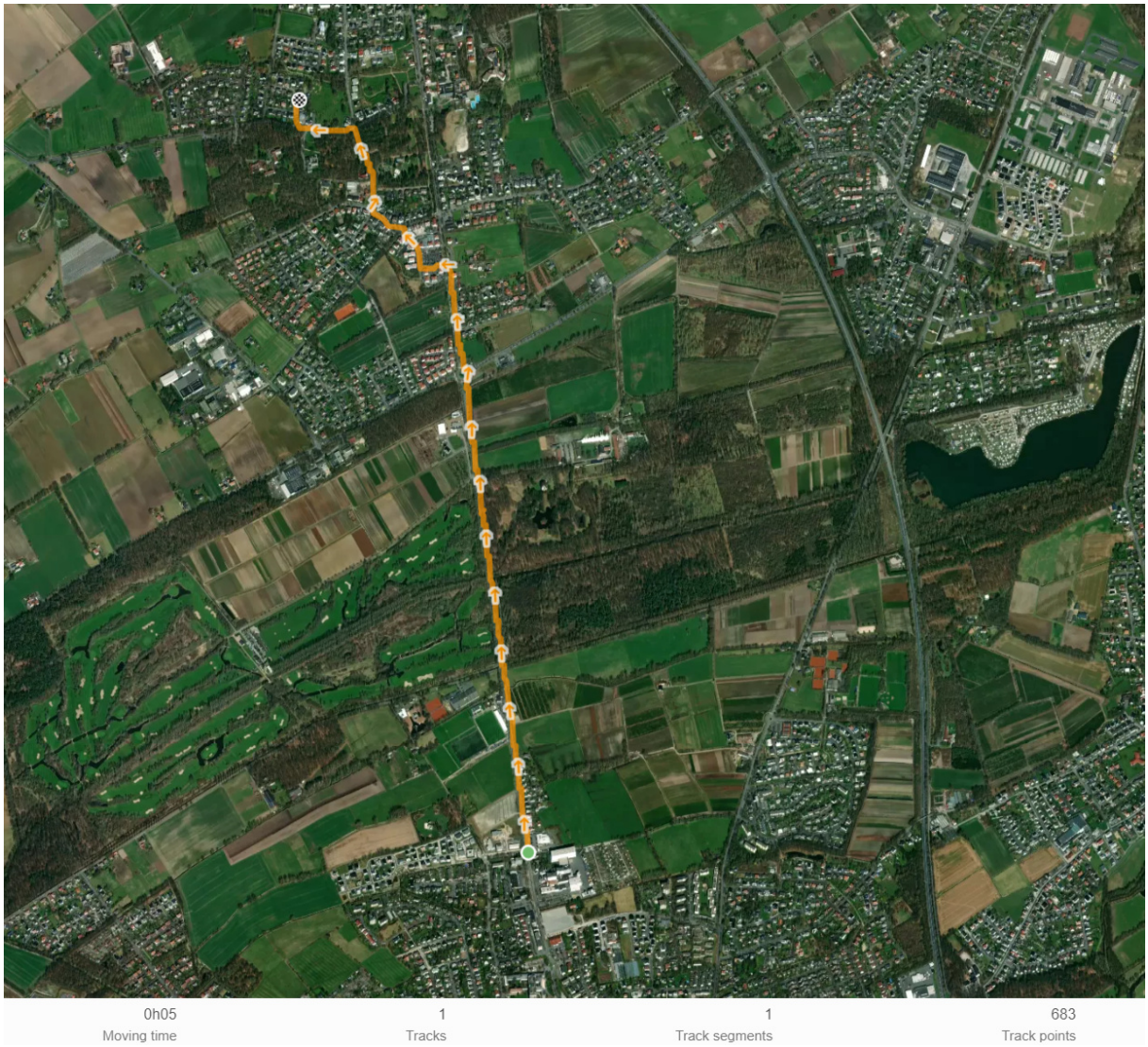

Abbildung 4.1: aufgezeichnete Strecke im 1. Test in gpx.studio

Abbildung 4.2: aufgezeichnete Strecke im 1. Test in gpx.studio (vergrößert)

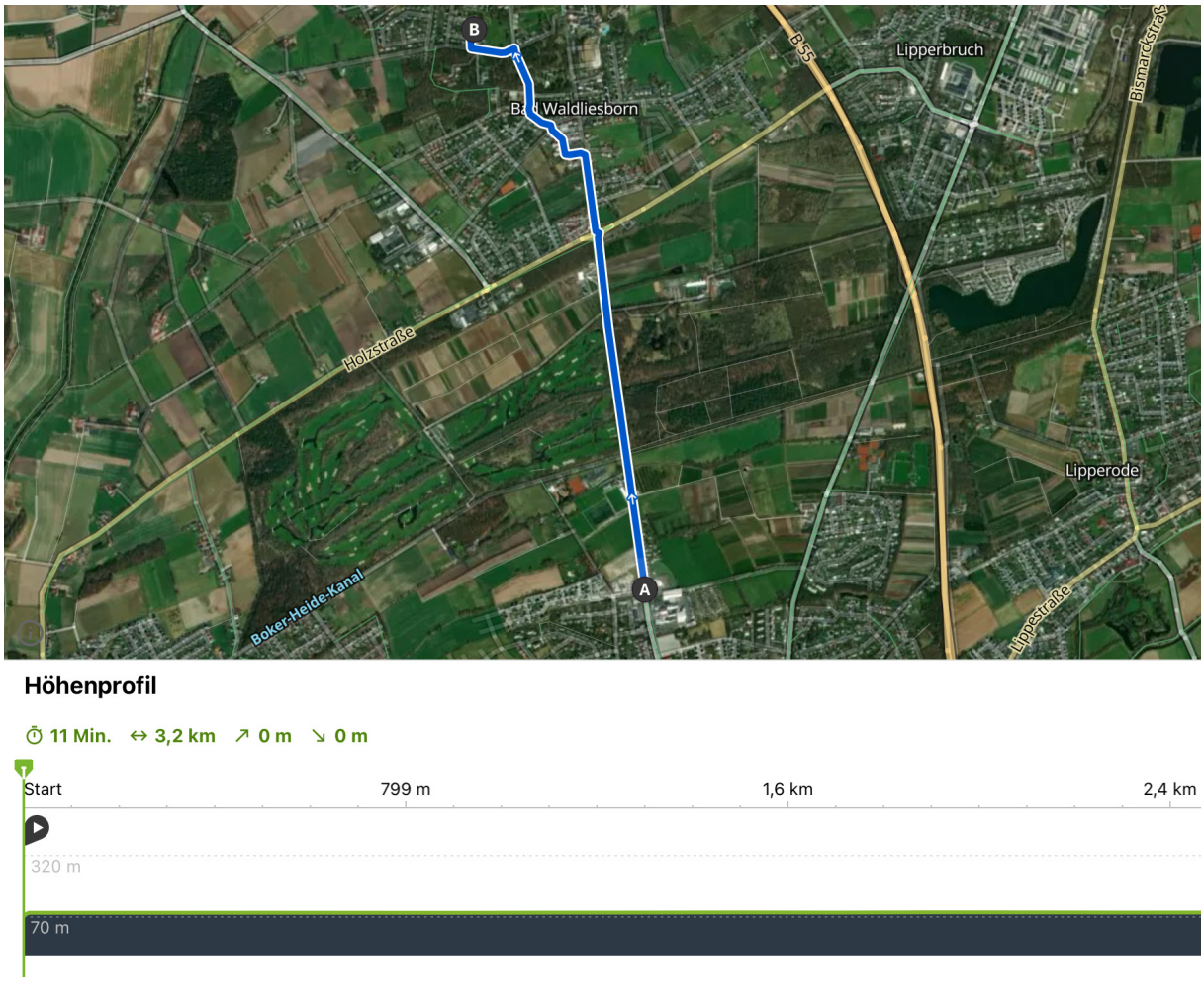

Abbildung 4.3: aufgezeichnete Strecke im 1. Test mit Handy

4.2.2 Testverfahren und Validierung der Zuverlässigkeit

Um die Zuverlässigkeit der GPS-Daten zu bewerten, werden mehrere Tests durchgeführt. Das zweiten Test wird in Abbildung 4.4 und Abbildung 4.5 mit blaue Farbe gezeigt. Das Fahrrad wurde für ca. 30 Minuten bewegt. Wie im ersten Test, die aufgezeichnete Strecke im gpx.studio zeigt, dass die GPS-Koordinaten fast exakt der tatsächlich zurückgelegten Strecke entsprechen.

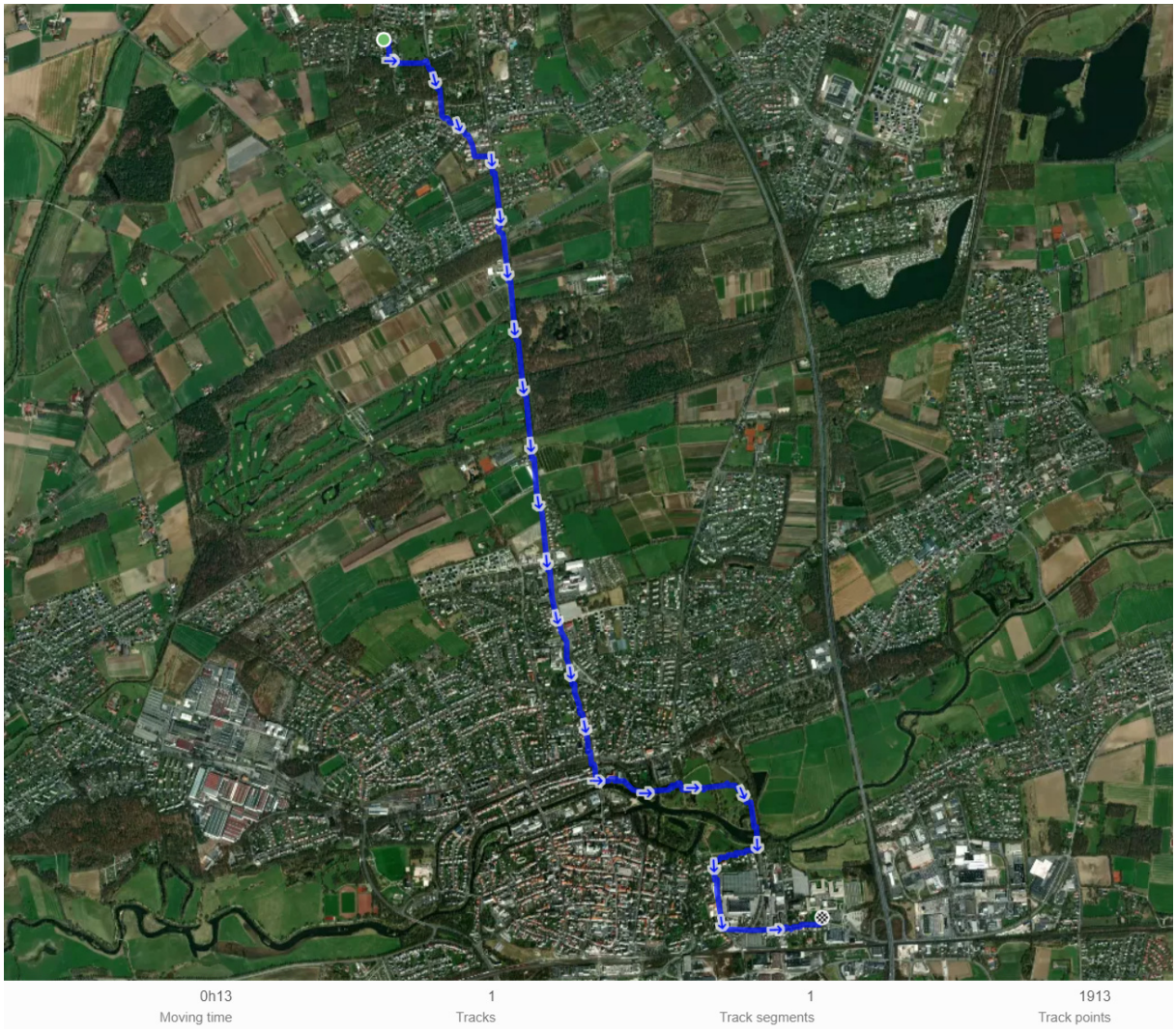

Abbildung 4.4: aufgezeichnete Strecke im 2. Test in gpx.studio

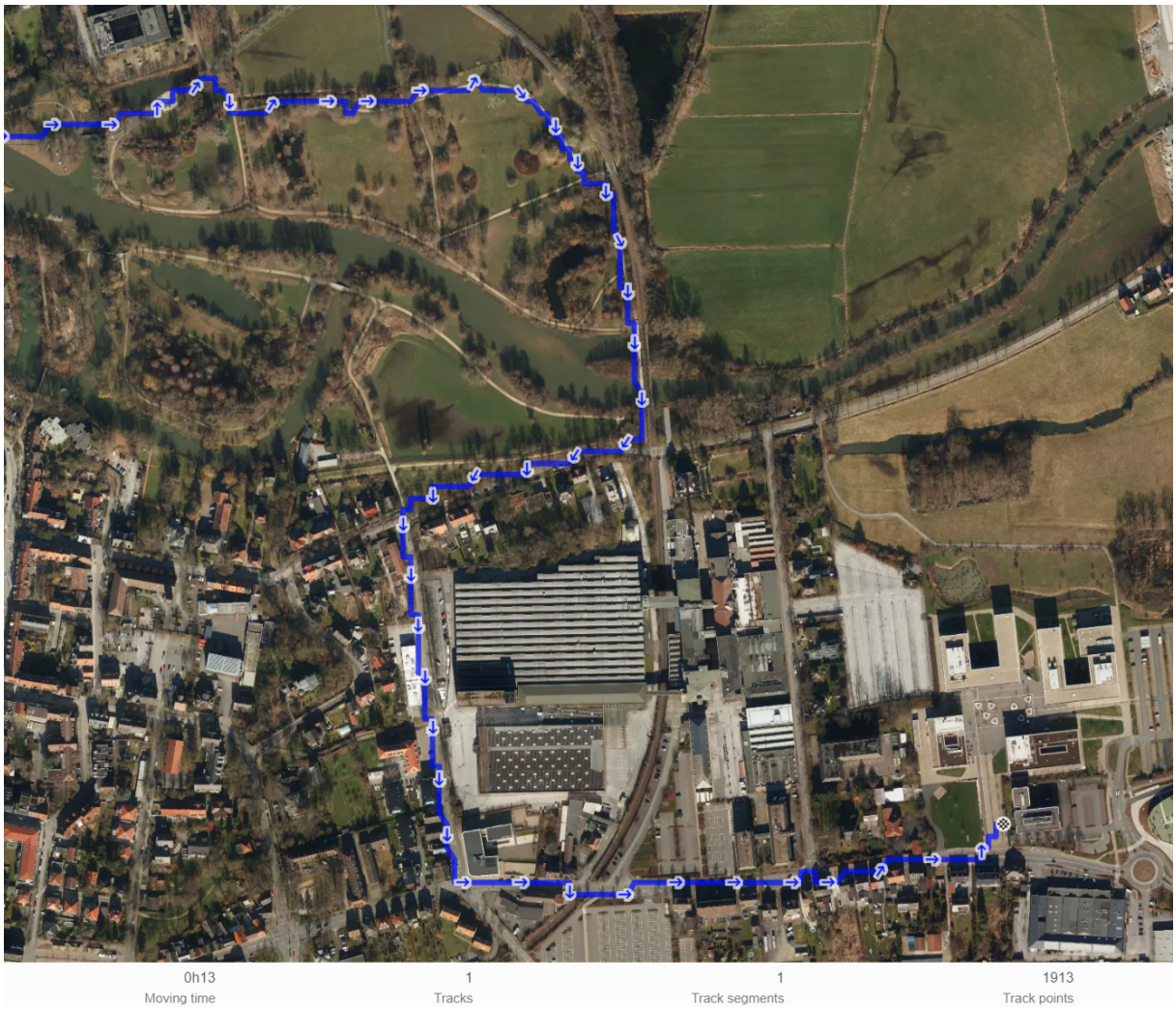

Abbildung 4.5: aufgezeichnete Strecke im 2. Test in gpx.studio (vergrößert)

4.2.3 Testverfahren und Validierung in höhe Geschwindigkeit

Um die Genauigkeit und Zuverlässigkeit der GPS-Daten in höheren Geschwindigkeiten zu bewerten, wird ein Auto-Test durchgeführt. Das Auto wurde für ca. 3 Minuten auf einer Stadtstraße gefahren. Die Geschwindigkeit des Autos betrug ca. 50 km/h. Die aufgezeichnete Strecke wird in Abbildung 4.6 und Abbildung 4.7 mit rote Farbe gezeigt.

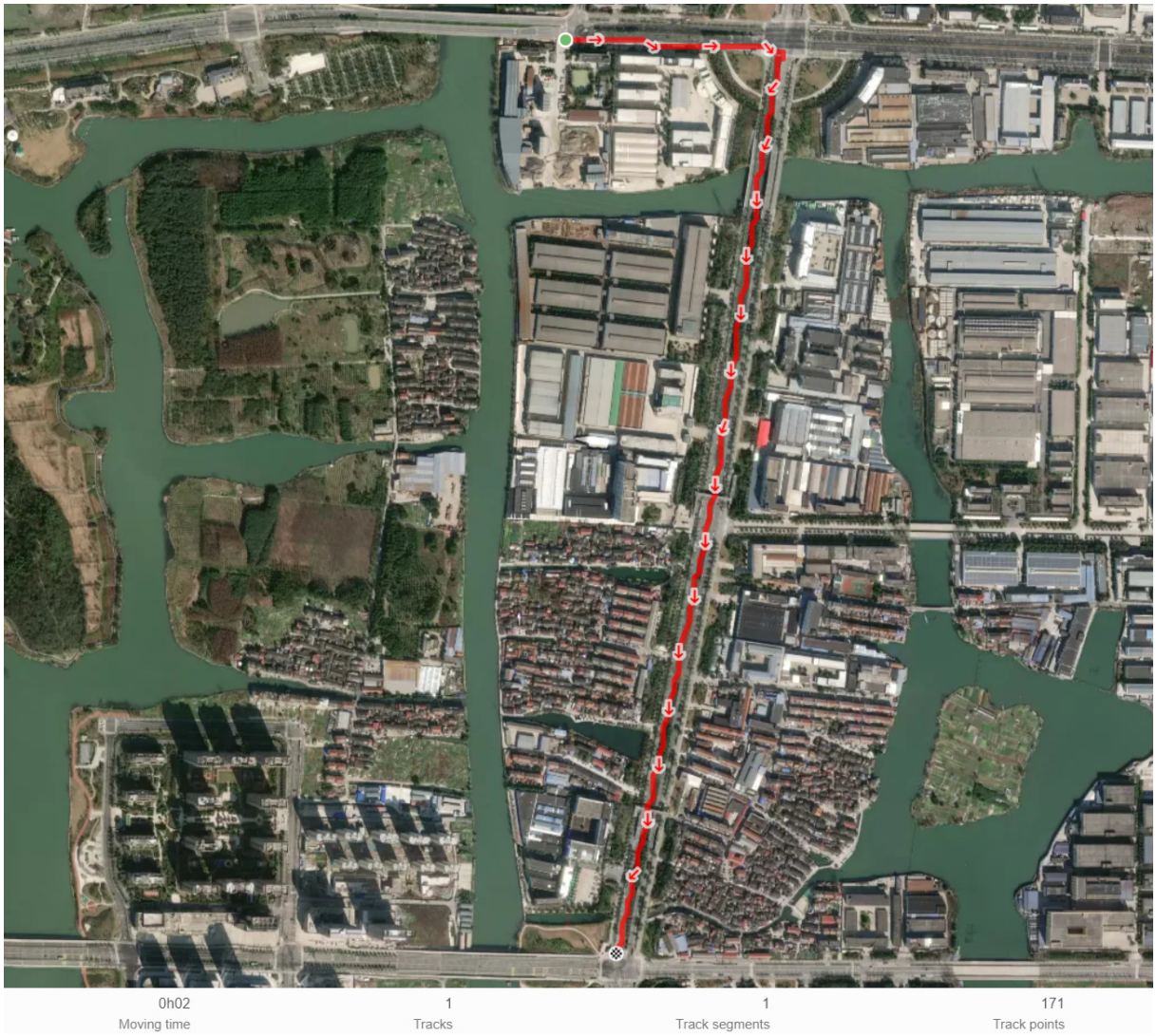

Abbildung 4.6: aufgezeichnete Strecke im Auto-Test in gpx.studio

Abbildung 4.7: aufgezeichnete Strecke im Auto-Test in gpx.studio (vergrößert)

4.3 Ergebnisse der Test- und Validierungsphase

Nach der Analyse der GPS-Daten unter verschiedenen Bedingungen lassen sich die Ergebnisse der Test- und Validierungsphase zusammenfassen. Die Genauigkeit der GPS-Daten ist sehr hoch, was bedeutet, dass die aufgezeichnete Strecke fast exakt der tatsächlich zurückgelegten Strecke entspricht. Ebenso zeichnet sich die Zuverlässigkeit der GPS-Daten aus, da die GPS-Koordinaten in den festgelegten Zeitintervallen korrekt aufgezeichnet werden. Die Übertragung der GPS-Daten an den PC erfolgt zuverlässig, und die Integrität der übertragenen Daten wird gewährleistet, was die Funktionalität der PC-Anwendung bestätigt. Diese arbeitet wie erwartet, und die gespeicherten Daten im GPX-Format sind korrekt und können problemlos in Kartendienste wie gpx.studio importiert werden, um die zurückgelegte Strecke visuell darzustellen. Auch bei höheren Geschwindigkeiten, wie im Auto-Test, zeigt der GPS-Tracker eine hohe Genauigkeit und Zuverlässigkeit. Die aufgezeichnete Strecke entspricht fast exakt der tatsächlich zurückgelegten Strecke, was die Eignung des GPS-Trackers für verschiedene Anwendungsbereiche bestätigt.

Gemäß dem Konstruktionsprinzip des GPS-Trackers ist bekannt, dass das Zeitintervall jeder Aufzeichnung eine Sekunde beträgt. Die Analyse der GPX-Dateien zeigt, dass die Größe der GPX-Datei von der Bewegungsgeschwindigkeit abhängt. Bei höheren Geschwindigkeiten, wie im Auto-Test, ist die Größe der GPX-Datei in eine gleiche Strecke kleiner als bei niedrigeren Geschwindigkeiten, wie im Fahrrad-Test. Allerdings ist die Genauigkeit bei höheren Geschwindigkeiten schlechter als bei niedrigeren Geschwindigkeiten. Dies ist darauf zurückzuführen, dass der Abstand zwischen den Aufzeichnungen umso größer ist, je höher die Geschwindigkeit ist. Außerdem dauert die Übertragung der GPS-Daten auf den PC und die Speicherung der Daten im GPX-Datei dauerte etwa 1 Minute pro 10 Minuten aufgezeichneter Strecke.

5 Zusammenfassung und Ausblick

5.1 Zusammenfassung der Arbeitsergebnisse

In dieser Arbeit wurde die Entwicklung eines GPS-Trackers vorgestellt, der durch den Einsatz eines Low-Cost Mikrocontrollers realisiert wurde. Das primäre Ziel dieses Projektes war es, eine kosteneffiziente, dennoch zuverlässige und benutzerfreundliche Lösung für die Positionsbestimmung zu entwickeln, die flexibel in verschiedenen Anwendungsbereichen eingesetzt werden kann. Besondere Aufmerksamkeit galt der sorgfältigen Auswahl der Hardware-Komponenten und der Optimierung der Software, um die Kosten zu minimieren, ohne dabei Kompromisse bei der Leistungsfähigkeit und Zuverlässigkeit einzugehen.

Durch innovative Ansätze in der Softwareentwicklung und Hardwarekonfiguration ist es gelungen, einen GPS-Tracker zu entwickeln, der sich durch geringe Herstellungskosten, einfache Bedienbarkeit, hohe Zuverlässigkeit und Genauigkeit auszeichnet. Die Ergebnisse der durchgeführten Tests bestätigen, dass der Tracker in der Lage ist, die Position mit einer beeindruckenden Genauigkeit zu bestimmen und die Daten effizient und sicher an ein Endgerät zu übermitteln. Darüber hinaus wurde im Rahmen des Projekts eine intuitive Benutzeroberfläche entwickelt, die es den Nutzern ermöglicht, die aufgezeichneten Daten nicht nur zu visualisieren, sondern auch in einem nutzerfreundlichen Format zu übertragen und zu speichern. Diese Funktionalität erweitert deutlich den Nutzen des GPS-Trackers, indem sie eine einfache Analyse und Verwendung der gesammelten Daten für verschiedene Zwecke ermöglicht.

Durch die Kombination aus niedrigen Produktionskosten, hoher Genauigkeit, Benutzerfreundlichkeit und der Fähigkeit, Daten effektiv zu visualisieren und zu speichern, bietet der entwickelte GPS-Tracker eine attraktive Option für zahlreiche Anwendungsfälle. Dazu zählen unter anderem die Logistik, das Flottenmanagement, persönliche Sicherheitsanwendungen, Outdoor-Aktivitäten und wissenschaftliche Forschung. Die Ergebnisse dieser Arbeit legen nahe, dass der entwickelte GPS-Tracker eine kostengünstige und dennoch leistungsstarke Alternative zu teureren kommerziellen Produkten darstellt und neue Möglichkeiten für den Einsatz von GPS-Technologie in verschiedenen Anwendungsbereichen eröffnet.

5.2 Ausblick auf zukünftige Entwicklungen und Anwendungen

Der GPS-Tracker basiert derzeit auf einem Entwicklungsboard und ist daher noch relativ groß und sperrig. Eines der Hauptziele für die Zukunft ist die weitere Miniaturisierung des Geräts, um es noch vielseitiger einsetzbar zu machen. Durch die Entwicklung eines eigenen PCBs und die Integration der Komponenten in einem kompakten Gehäuse könnte das Gerät weiter optimiert werden.

Ein weiterer wichtiger Forschungsschwerpunkt liegt in der Verbesserung der Energieeffizienz des Geräts, um die Laufzeit des Akkus zu verlängern und den Einsatz in abgelegenen oder schwer zugänglichen Gebieten zu erleichtern. Bei einige Anwendungen, die eine hohe Genauigkeit nicht erfordern müssen, muss das GPS Modul nicht ständig 1 Hz arbeiten, sondern nur in festgelegten Zeitintervallen (Z.B 5 Sekunden), um die Energie zu sparen und die Speichernutzung zu optimieren.

Darüber hinaus wird an der Integration zusätzlicher Sensoren eine wichtige Rolle spielen. Durch die Kombination des GPS-Trackers mit weiteren Sensoren wie Beschleunigungssensoren, Temperatursensoren oder Feuchtigkeitssensoren könnten neue Anwendungsmöglichkeiten erschlossen werden. Dies würde den Anwendungsbereich des Trackers erheblich erweitern und ihn für Aufgaben in der Umweltüberwachung, in der Logistik oder im Bereich der persönlichen Sicherheit noch attraktiver machen.

Zusammenfassend lässt sich sagen, dass die Entwicklung dieses GPS-Trackers ein vielversprechender erster Schritt in Richtung einer kostengünstigen und dennoch leistungsstarken Lösung für die Positionsbestimmung ist. Die vorgestellte Lösung eröffnet neue Möglichkeiten für den Einsatz von GPS-Technologie in verschiedenen Anwendungsbereichen und bietet eine solide Grundlage für zukünftige Entwicklungen und Anwendungen.

Literaturverzeichnis

[1] R.B. Gaikwad, K.R. Pawar, R.P. Gaikwad, S.B. Gaikwad, "Animal Health Monitoring System Using GPS & GSM Modem," S. 314–317, 2019.

[2] U. Brinkschulte und T. Ungerer, *Mikrocontroller und Mikroprozessoren*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, ISBN: 978-3-642-05397-9.

[3] G. Schmitt und A. Riedenauer, Hrsg., *Mikrocontrollertechnik mit AVR*. De Gruyter, 2019, ISBN: 9783110403886.

[4] Microchip Technology, "ATmega48/V/88/V/168/V Data Sheet," 2018.

[5] SiSy Solutions, "Technische Beschreibung myAVR Board Version 2.20," 2019.

[6] J. G. McNeff, "The global positioning system," *IEEE Transactions on Microwave Theory and Techniques*, Jg. 50, Nr. 3, S. 645–652, 2002.

[7] P. J. Teunissen und O. Montenbruck, *Springer Handbook of Global Navigation Satellite Systems*. Cham: Springer International Publishing, 2017, ISBN: 978-3-319-42926-7.

[8] C. Wolfseher. "Wie funktioniert ein Navi? | © C. Wolfseher." (9.11.2021), Adresse: https://www.katharinengymnasium.de/wolf/web/gps/gps1Trilateration.html.

[9] Wikipedia, Hrsg. "GPS Exchange Format." (2024), Adresse: https://de.wikipedia.org/w/index.php?title=GPS_Exchange_Format&oldid=241884354.

[10] W. Gehrke, M. Winzker, K. Urbanski und R. Woitowitz, *Digitaltechnik*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, ISBN: 978-3-662-49730-2.

[11] CDtop Technology, "PA1616D datasheet," 2024.

[12] AZ-Delivery, "HD44780 16x02 Blaues Display HD44780 16x02 Blaues Display mit Serielle Schnittstelle Datenblatt," 2024.

[13] Simeon Maxein, "Benutzen einer SD-Speicherkarte mit dem ATmega-Microcontroller," 2008.

[14] SD Association | The SD Association, Hrsg. "SD Standard Overview | SD Association." (2023), Adresse: https://www.sdcard.org/developers/sd-standard-overview/.

[15] Wikipedia, Hrsg. “Keyhole Markup Language.” (2023), Adresse: https://de.wikipedia.org/w/index.php?title=Keyhole_Markup_Language&oldid=236173780.

Anhang: Code-Listings der Mikrocontroller-Firmware

```
#define F_CPU 3686400L // CPU-Frequenz
#endif
#define BAUD 9600L // Baudrate (GPS-Modul Standard)

#define BUTTON1_PIN PD2 // Taste 1-Pin
#define BUTTON2_PIN PD3 // Taste 2-Pin

#include <avr/io.h> // AVR-IO
#include <avr/interrupt.h> // AVR-Interrupt
#include <util/delay.h> // AVR-Delay
#include <stdint.h> // Standarddatentypen
#include <string.h> // String
#include <stdlib.h> // Standardbibliothek
#include <stdio.h> // Standard-Ein- und Ausgabe
#include <stdbool.h> // Boolesche Werte
#include <avr/eeprom.h> // AVR-EEPROM
#include "Uart.h" // UART
#include "lcd.h" // LCD
#include "spi.h" // SPI
#include "sdcard.h" // SD-Karte

volatile uint8_t Messung_modus = 0; // Flag für Messung Modus (0 = AUS, 1 = AN)
volatile uint8_t Lesen_modus = 0; // Flag für Lesen Modus (0 = AUS, 1 = AN)
static bool gpsSignalLost; // GPS-Signalstatus

uint32_t EEMEM speicher_Addr; // Speicherort der Adresse im EEPROM (0x00000000)
uint32_t SchreibenAddr = 0x00000000; // Addresse zu schreiben (0x00000000)
uint32_t LesenAddr = 0x00000000; // Addresse zu lesen (0x00000000)
uint32_t Gesamtbytes = 1977188352; // Gesamtgröße als KB der 2GB SD-Karte (1977188352)
uint32_t Sektorbytes = 0x00000200; // 512kb pro Sektor (0x00000200)
```

```
uint8_t res[5], buf[50];  // Für SD-Karte schreiben
uint8_t res1[5], buf1[50]; // Für SD-Karte lesen
uint8_t token0, token1; // Für SD-Karte lesen und schreiben
char* token; // Für die GPS-Datensatz verarbeiten

void initializeSystem(void); // System initialisieren
void lesenSDCard(void); // SD-Karte lesen
void abholenGPSDaten(void); // GPS-Daten abholen
void verarbeitenGPSLine(char *line); // GPS-Daten verarbeiten

void setBaudRate(unsigned long baud) {
    uart_init(UART_BAUD_SELECT(baud, F_CPU));
}

void EEPROM_speicherAddress(uint32_t Addr) { // Adresse im EEPROM speichern
    eeprom_busy_wait(); // Warten, dass EEPROM nicht besetzt ist
    eeprom_update_block((const void*)&Addr, &speicher_Addr, sizeof(Addr));
    // Adresse im EEPROM speichern
}

uint32_t EEPROM_lesenAddress(void) { // Adresse im EEPROM lesen
    uint32_t Addr; // Adresse
    eeprom_busy_wait(); // Warten, dass EEPROM nicht besetzt ist
    eeprom_read_block((void*)&Addr, &speicher_Addr, sizeof(Addr)); //
    Adresse im EEPROM lesen
    return Addr; // Adresse zurückgeben
}

// Behandlung von Tastenunterbrechungen mit der ISR (Interrupt Service
    Routine)
ISR(INT0_vect) // INT0
{
    _delay_ms(20); // Entprellung (20ms)
    if (!(PIND & (1 << BUTTON1_PIN))) { // Prüfen, ob BUTTON1_PIN gedrückt
    ist (0 = gedrückt, 1 = nicht gedrückt)
        if (Lesen_modus) { // Wenn im Lesemodus
            return; // Zurück
        }
        if (PIND & (1 << BUTTON2_PIN)) { // Prüfen, ob BUTTON2_PIN NICHT
```

```
    gedrückt ist (0 = nicht gedrückt, 1 = gedrückt)
66              Messung_modus = !Messung_modus; // Messung Modus umschalten (0
    = AUS, 1 = AN)
67              //uart_puts("Messung: "); // Ausgabe auf UART
68              //uart_puts(Messung_modus ? "AN" : "AUS"); // Ausgabe auf UART
69              //uart_puts("\r\n"); // Ausgabe auf UART
70              } else { // Wenn BUTTON2_PIN auch gedrückt ist (0 = gedrückt, 1
     = nicht gedrückt)
71              EEPROM_speicherAddress(0x00000000); // Setzen der zuletzt
    gespeicherten Adresse auf 0x00000000 während der manuellen
    Initialisierung
72              initializeSystem(); // System neu initialisieren
73          }
74      }
75  }
76
77  ISR(INT1_vect)
78  {
79      _delay_ms(20); // Entprellung (20ms)
80      if (!(PIND & (1 << BUTTON2_PIN))) { // Prüfen, ob BUTTON2_PIN gedrückt
    ist (0 = gedrückt, 1 = nicht gedrückt)
81          if (Messung_modus) { // Wenn im Messmodus
82              return; // Zurück
83          }
84          if (Lesen_modus) { // Wenn im Lesemodus, verhindere
    Reinitialisierung
85              return; // Zurück
86          }
87          if (PIND & (1 << BUTTON1_PIN)) { // Prüfen, ob BUTTON1_PIN NICHT
    gedrückt ist (0 = nicht gedrückt, 1 = gedrückt)
88              initializeSystem(); // System neu initialisieren
89              Lesen_modus = !Lesen_modus; // Lesen Modus umschalten (0 = AUS,
     1 = AN)
90              //uart_puts("Lesen: "); // Ausgabe auf UART
91              //uart_puts(Lesen_modus ? "AN" : "AUS"); // Ausgabe auf UART
92              //uart_puts("\r\n"); // Ausgabe auf UART
93              Messung_modus = 0; // Wenn im Lesemodus, deaktivieren des
    Messmodus (0 = AUS, 1 = AN)
94              } else { // Wenn BUTTON1_PIN auch gedrückt ist (0 = gedrückt, 1
```

```
    = nicht gedrückt)
95              EEPROM_speicherAddress(0x00000000); // Setzen der zuletzt
    gespeicherten Adresse auf 0x00000000 während der manuellen
    Initialisierung
96              initializeSystem(); // System neu initialisieren
97          }
98      }
99  }
100
101 int main(void)
102 {
103     // Initialisierung des Systems
104     initializeSystem();
105
106     while(1) // Endlosschleife
107     {
108         if (Lesen_modus) // Lesen Modus (0 = AUS, 1 = AN)
109         {
110             lesenSDCard(); // SD-Karte lesen
111         }
112
113         if (Messung_modus) // Messung Modus (0 = AUS, 1 = AN)
114         {
115             abholenGPSDaten(); // GPS-Daten abholen
116         }
117     }
118     return 0; // Zurück
119 }
120
121 void initializeSystem(void) // System initialisieren
122 {
123     // UART initialisieren
124     uart_init(UART_BAUD_SELECT(BAUD, F_CPU));
125
126     // LCD initialisieren
127     lcd_init();
128
129     // SPI initialisieren
130     SPI_init(SPI_MASTER | SPI_FOSC_16 | SPI_MODE_0);
```

```
131
132     // Letzte Adresse aus EEPROM lesen
133     SchreibenAddr = EEPROM_lesenAddress();
134
135     // SD-Karte initialisieren
136     if(SD_init() != SD_SUCCESS) // Wenn SD-Karte nicht initialisiert werden
         kann
137     {
138         uart_puts("Fehler bei der Initialisierung der SD-Karte!\r\n"); //
    Ausgabe auf UART
139         lcd_print_str("Keine SD-Karte!"); // Ausgabe auf LCD
140     }
141     else
142     {
143         if (SchreibenAddr == 0) // Wenn SchreibenAddr gleich 0 ist
144         {
145             _delay_ms(100); // Kurze Verzögerung
146             lcd_print_str("Messung starten durch Taste 1"); // Ausgabe auf
    LCD
147         }
148         else // Wenn SchreibenAddr nicht gleich 0 ist
149         {
150             _delay_ms(100); // Kurze Verzögerung
151             lcd_print_str("Weiter messen durch Taste 1"); // Ausgabe auf
    LCD
152         }
153     }
154
155     // Taste 1-Pin als Eingang setzen und den internen Pull-Up-Widerstand
    aktivieren
156     DDRD &= ~(1 << BUTTON1_PIN);  // Setze als Eingang
157     PORTD |= (1 << BUTTON1_PIN);  // Aktiviere internen Pull-Up-Widerstand
158
159     // Taste 2-Pin als Eingang setzen und den internen Pull-Up-Widerstand
    aktivieren
160     DDRD &= ~(1 << BUTTON2_PIN);  // Setze als Eingang
161     PORTD |= (1 << BUTTON2_PIN);  // Aktiviere internen Pull-Up-Widerstand
162
163     // INT0 einstellen, um die fallende Flanke von Taste 1 zu erkennen
```

```
164     EICRA |= (1 << ISC01);
165     EICRA &= ~(1 << ISC00); // INT0
166 
167     // INT1 einstellen, um die fallende Flanke von Taste 2 zu erkennen
168     EICRA |= (1 << ISC11);
169     EICRA &= ~(1 << ISC10); // INT1
170 
171     // Externen Interrupt 0 und externen Interrupt 1 zulassen
172     EIMSK |= (1 << INT0) | (1 << INT1); // INT0 und INT1
173 
174     // Interrupts aktivieren
175     sei();
176 }
177 
178 void lesenSDCard(void) // SD-Karte lesen
179 {
180     lcd_clear(); // LCD löschen
181     lcd_print_str("Lesen..."); // Ausgabe auf LCD
182     setBaudRate(115200L); // Baudrate setzen (Daten schneller übertragen)
183     if (LesenAddr < SchreibenAddr) // Wenn die Adresse zum Lesen kleiner
    als die Adresse zum Schreiben ist
184     {
185         res1[0] = SD_readSingleBlock(LesenAddr, buf1, &token1); // SD-Karte
     lesen
186         if (res1[0] == 0x00) // Wenn SD-Karte erfolgreich gelesen werden
    kann
187         {
188             if (token1 == SD_START_TOKEN) // Wenn Token1 gleich dem Start-
    Token ist
189             {
190                 for (uint8_t i = 0; i < 50; i++) // Für 50 Zeichen
191                 {
192                     if (buf1[i] != 0) // Wenn das Zeichen nicht gleich 0
    ist
193                     {
194                         uart_putc(buf1[i]); // Ausgabe auf UART
195                     }
196                     else
197                     {
```

```
198                         break; // Zurück
199                     }
200                 }
201             }
202             else
203             {
204                 uart_puts("Fehler!\r\n"); // Ausgabe auf UART
205             }
206         }
207         else
208         {
209             uart_puts("Fehler!\r\n"); // Ausgabe auf UART
210         }
211         LesenAddr += Sektorbytes * 1; // Jede 1 Sektor einmal lesen (512kb)
212     }
213     else
214     {
215         Lesen_modus = 0; // Lesen Modus deaktivieren (0 = AUS, 1 = AN)
216         LesenAddr = 0x00000000; // Lesen Adresse auf 0x00000000 setzen
217         uart_puts("Lesen: "); // Ausgabe auf UART
218         uart_puts(Lesen_modus ? "AN" : "AUS"); // Ausgabe auf UART
219         uart_puts("\r\n"); // Ausgabe auf UART
220         lcd_clear(); // LCD löschen
221         _delay_ms(100); // Kurze Verzögerung
222         lcd_print_str("Lesen erfolgreich!"); // Ausgabe auf LCD
223         setBaudRate(9600L); // Baudrate setzen (GPS-Modul Standard)
224     }
225 }
226
227 void abholenGPSDaten(void) // GPS-Daten abholen
228 {
229     static char line[50]; // Zeile
230     static uint8_t line_index = 0; // Zeilenindex
231     static uint8_t gngga_index = 0; // GNGGA-Index
232
233     if(uart_available() > 0) // Sobald die GPS-Daten verfügbar sind (0 =
    nicht verfügbar, 1 = verfügbar)
234     {
235         for(uint8_t i = 0; i < uart_available(); i++) // Für die Anzahl der
```

```
 verfügbaren GPS-Daten
236     {
237         char c = uart_getc(); // GPS-Daten abholen
238 
239         if (gngga_index == 0 && c == '$')
240         {
241             gngga_index++;
242         }
243         else if (gngga_index == 1 && c == 'G')
244         {
245             gngga_index++;
246         }
247         else if (gngga_index == 2 && c == 'N')
248         {
249             gngga_index++;
250         }
251         else if (gngga_index == 3 && c == 'G')
252         {
253             gngga_index++;
254         }
255         else if (gngga_index == 4 && c == 'G')
256         {
257             gngga_index++;
258         }
259         else if (gngga_index == 5 && c == 'A')
260         {
261             gngga_index++;
262         }
263         else if (gngga_index == 6 && c == ',')
264         {
265             gngga_index++;
266         }
267         else if (gngga_index >= 7)  // Ohne "$GNGGA,", vom 7. Zeichen
    an zu zählen
268         {
269             if (line_index < sizeof(line) - 1) // Wenn der Zeilenindex
    kleiner als die Zeilengröße - 1 ist
270             {
271                 line[line_index] = c; // Zeile setzen
```

```
272                     line_index++; // Zeilenindex erhöhen
273                 }
274             }
275             else
276             {
277                 gngga_index = 0;
278             }
279 
280             if (line_index == 37) // Bis das Zeichen Fix -> Z.B
    (165006.000,2241.9107,N,12017.2383,E,1)
281             {
282                 line[line_index] = '\0'; // Zeile setzen
283                 verarbeitenGPSLine(line); // GPS-Daten verarbeiten
284                 line_index = 0; // Zeilenindex zurücksetzen
285                 gngga_index = 0; // GNGGA-Index zurücksetzen
286             }
287 
288             // Überprüfen, ob GPS-Signal verloren gegangen ist
289             if (gpsSignalLost) {
290                 lcd_clear(); // LCD löschen
291                 lcd_print_str("Kein GPS-Signal!"); // Ausgabe auf LCD
292                 gpsSignalLost = false; // Setze den GPS-Signalstatus zurück
293             }
294         }
295     }
296 }
297 
298 void verarbeitenGPSLine(char* line) // GPS-Daten verarbeiten
299 {
300     float latitude = 0.0, longitude = 0.0; // Breitengrad und Längengrad
301     char NS, EW; // Norden und Osten
302 
303     token = strtok(line, ","); // Zeile aufteilen
304     uint8_t hour = (token[0] - '0') * 10 + (token[1] - '0'); // Zeitzone
    UTC
305     uint8_t min = (token[2] - '0') * 10 + (token[3] - '0');
306     uint8_t sec = (token[4] - '0') * 10 + (token[5] - '0');
307     char time_data[16]; // Zeit
308 
```

```
309     // Zeit setzen (Zeit: 10:07:53)
310     strcpy(time_data, "Zeit: ");
311     itoa(hour, buf, 10);
312     if (hour < 10) strcat(time_data, "0"); // Wenn die Stunde kleiner als
        10 ist, dann 0 hinzufügen
313     strcat(time_data, buf);
314     strcat(time_data, ":");
315     itoa(min, buf, 10);
316     if (min < 10) strcat(time_data, "0"); // Wenn die Minute kleiner als 10
        ist, dann 0 hinzufügen
317     strcat(time_data, buf);
318     strcat(time_data, ":");
319     itoa(sec, buf, 10);
320     if (sec < 10) strcat(time_data, "0"); // Wenn die Sekunde kleiner als
        10 ist, dann 0 hinzufügen
321     strcat(time_data, buf);
322
323     token = strtok(NULL, ","); // Zeile aufteilen
324     int degree = atoi(token) / 100; // Grad
325     float minute = atof(token) - degree * 100; // Minute
326     latitude = degree + minute / 60.0; // Breitengrad
327
328     token = strtok(NULL, ","); // Zeile aufteilen
329     NS = token[0]; // Norden oder Süden
330
331     token = strtok(NULL, ","); // Zeile aufteilen
332     degree = atoi(token) / 100; // Grad
333     minute = atof(token) - degree * 100; // Minute
334     longitude = degree + minute / 60.0; // Längengrad
335
336     token = strtok(NULL, ","); // Zeile aufteilen
337     EW = token[0]; // Osten oder Westen
338
339     token = strtok(NULL, ","); // Zeile aufteilen
340     int fix = atoi(token); // Fix (0 = kein Fix, 1 = Fix)
341
342     // Falls der Fix 0 ist, bedeutet dies, dass es keine gültigen GPS-Daten
        gibt (0 = kein Fix, 1 = Fix)
343     if (fix == 0) {
```

```
344         gpsSignalLost = true; // GPS-Signalstatus auf verloren setzen
345         return; // Zurück
346     }
347 
348     // GPS-Signalstatus auf gefunden setzen, da gültige Daten vorhanden
    sind
349     gpsSignalLost = false;
350 
351     char lat_data[16], lon_data[16]; // Breitengrad und Längengrad
352     int lat_int = (int)latitude; // Breitengrad
353     int lat_frac = (int)((latitude - lat_int) * 10000); // Längengrad
354 
355     // Breitengrad setzen (Lat: 51.7141 N)
356     strcpy(lat_data, "Lat: ");
357     itoa(lat_int, buf, 10);
358     strcat(lat_data, buf);
359     strcat(lat_data, ".");
360     itoa(lat_frac, buf, 10);
361     if (lat_frac < 1000) strcat(lat_data, "0");
362     if (lat_frac < 100) strcat(lat_data, "0");
363     if (lat_frac < 10) strcat(lat_data, "0");
364     strcat(lat_data, buf);
365     strcat(lat_data, " ");
366     strncat(lat_data, &NS, 1);
367 
368     int lon_int = (int)longitude; // Längengrad
369     int lon_frac = (int)((longitude - lon_int) * 10000); // Längengrad
370 
371     // Längengrad setzen (Lon: 8.3302 E)
372     strcpy(lon_data, "Lon: ");
373     itoa(lon_int, buf, 10);
374     strcat(lon_data, buf);
375     strcat(lon_data, ".");
376     itoa(lon_frac, buf, 10);
377     if (lon_frac < 1000) strcat(lon_data, "0");
378     if (lon_frac < 100) strcat(lon_data, "0");
379     if (lon_frac < 10) strcat(lon_data, "0");
380     strcat(lon_data, buf);
381     strcat(lon_data, " ");
```

```
382     strncat(lon_data, &EW, 1);
383
384     lcd_clear(); // LCD löschen
385     _delay_ms(100); // Kurze Verzögerung
386     lcd_print_str(lat_data); // Ausgabe auf LCD
387     lcd_setcursor(0,1);
388     lcd_print_str("  ");
389     lcd_print_str(lon_data); // Ausgabe auf LCD
390
391     // Die Daten in den Puffer schreiben
392     memset(buf, 0, sizeof(buf));  // Den Puffer mit Nullen initialisieren
393
394     // -> Z.B (Zeit: 10:07:53, Lat: 51.7141 N, Lon: 8.3302 E)
395     strcpy((char *)buf, time_data); // Zeit setzen
396     strcat((char *)buf, ", ");
397     strcat((char *)buf, lat_data); // Breitengrad setzen
398     strcat((char *)buf, ", ");
399     strcat((char *)buf, lon_data); // Längengrad setzen
400     strcat((char *)buf, "\r\n");
401
402     // Daten in den Sektor schreiben (512kb)
403     res[0] = SD_writeSingleBlock(SchreibenAddr, buf, &token0); // SD-Karte
    schreiben
404
405     // Aktualisieren der Adresse für das nächste Schreiben
406     SchreibenAddr += Sektorbytes; // Jede 1 Sektor einmal schreiben (512kb)
407
408     if(SchreibenAddr >= Gesamtbytes) { // Wenn die Adresse zum Schreiben gr
    ößer als die Gesamtgröße ist
409         SchreibenAddr = 0x00000000; // Schreiben Adresse auf 0x00000000
    setzen
410     }
411
412     // Adresse im EEPROM aktualisieren
413         EEPROM_speicherAddress(SchreibenAddr); // Adresse im EEPROM
    speichern
414 }
```

Listing 5.1: vollständiger Quellcode des Mikrocontroller-Programms

Anhang: Code-Listings der PC-Anwendung

```cpp
#include <iostream> // Für Ein- und Ausgabe
#include <fstream> // Für Dateioperationen
#include <string> // Für std::string
#include <windows.h> // Für COM-Port
#include <ctime> // Für aktuelles Datum und Uhrzeit
#include <Shlobj.h>  // Für SHGetFolderPath (Benutzerdokumente-Ordner abrufen)

// Daten von COM lesen
bool COMDatenLesen(HANDLE hCom, std::string& daten) {
    char puffer[64]; // Puffer für die empfangenen Daten (64 Bytes)
    DWORD geleseneBytes; // Anzahl der gelesenen Bytes
    if (ReadFile(hCom, puffer, sizeof(puffer) - 1, &geleseneBytes, nullptr) && geleseneBytes > 0) { // Lesen Sie die Daten vom COM-Port
        puffer[geleseneBytes] = '\0'; // Nullterminator hinzufügen
        daten = puffer; // Daten in den String schreiben
        return true; // Erfolg
    }
    return false; // Misserfolg
}

// Aktuelles Datum abrufen
std::string aktuellesDatumHolen() {  // Aktuelles Datum abrufen
    time_t jetzt = time(nullptr); // Aktuelle Zeit abrufen
    struct tm zeitstruktur; // Zeitstruktur erstellen
    char datumPuffer[80]; // Puffer für das Datum (80 Bytes)
    localtime_s(&zeitstruktur, &jetzt); // Zeit in die Zeitstruktur konvertieren
    strftime(datumPuffer, sizeof(datumPuffer), "%Y-%m-%d", &zeitstruktur); // Format: JJJJ-MM-TT
    return datumPuffer; // Datum zurückgeben
}

// Aktuelles Datum und Uhrzeit abrufen für den Dateinamen
std::string aktuellesDatumUndUhrzeitHolen() { // Aktuelles Datum und Uhrzeit abrufen
```

```
32      time_t jetzt = time(nullptr); // Aktuelle Zeit abrufen
33      struct tm zeitstruktur; // Zeitstruktur erstellen
34      char datumPuffer[80];  // Puffer für das Datum (80 Bytes)
35      localtime_s(&zeitstruktur, &jetzt); // Zeit in die Zeitstruktur
        konvertieren
36      strftime(datumPuffer, sizeof(datumPuffer), "%Y%m%d_%H%M%S", &
        zeitstruktur); // Format: JJJJMMTT_HHMMSS
37      return datumPuffer; // Datum und Uhrzeit zurückgeben
38  }
39
40  // Extrahieren und Konvertieren von Breiten- und Längengraden
41  std::string BreitenLängengradKonvertieren(const std::string& rohwert, char
        richtung) { // Extrahieren und Konvertieren von Breiten- und Lä
        ngengraden
42      std::string dezimalwert = rohwert.substr(0, rohwert.find(' ')); //
        Extrahieren Sie den Dezimalwert aus dem Rohwert
43      if (richtung == 'S' || richtung == 'W') { // Wenn die Richtung Süden
        oder Westen ist, ist der Dezimalwert negativ
44          dezimalwert = "-" + dezimalwert; // Dezimalwert negativ
45      }
46      return dezimalwert; // Dezimalwert zurückgeben
47  }
48
49  int main() {
50      setlocale(LC_ALL, ""); // Locale setzen, um Umlaute zu unterstützen (ä,
        ö, ü, ß)
51
52      // COM-Port vom Benutzer erfragen (Beispiel: 5)
53      int comNummer; // COM-Port-Nummer
54      std::cout << "Bitte geben Sie die Nummer des gewünschten COM-Ports ein:
        "; // Aufforderung zur Eingabe der COM-Port-Nummer
55      std::cin >> comNummer; // COM-Port-Nummer vom Benutzer eingeben
56
57      std::string comName = "COM" + std::to_string(comNummer) + ":"; // COM-
        Port-Name (Beispiel: COM5:)
58
59      HANDLE hCom; // COM-Port-Handle
60      DCB dcbStruktur; // DCB-Struktur
61      BOOL erfolg; // Erfolg
```

```
62
63      hCom = CreateFileA(comName.c_str(), GENERIC_READ | GENERIC_WRITE, 0,
    nullptr, OPEN_EXISTING, 0, nullptr); // Öffnen Sie den COM-Port
64      if (hCom == INVALID_HANDLE_VALUE) { // Wenn der COM-Port nicht geöffnet
        werden kann, wird eine Fehlermeldung ausgegeben
65          std::cerr << "Fehler beim Öffnen von " << comName << std::endl; //
    Fehlermeldung
66          return 1; // Beendet das Programm, wenn der COM-Port nicht geöffnet
        werden kann
67      }
68
69      std::wcout << L"Erfolgreich verbunden mit " << comName.c_str() << std::
    endl; // Erfolgsmeldung
70      std::wcout << L"Bitte drücken Sie die Taste 2, um die GPX-Datei zu
    speichern!" << std::endl; // Aufforderung zum Speichern der GPX-Datei
71
72      erfolg = GetCommState(hCom, &dcbStruktur); // COM-Port-Status abrufen
73      if (!erfolg) { // Wenn der COM-Port-Status nicht abgerufen werden kann,
        wird eine Fehlermeldung ausgegeben
74          std::cerr << "GetCommState fehlgeschlagen" << std::endl; //
    Fehlermeldung
75          CloseHandle(hCom); // COM-Port schließen
76          return 1; // Beendet das Programm, wenn der COM-Port-Status nicht
    abgerufen werden kann
77      }
78
79      dcbStruktur.BaudRate = CBR_115200; // Baudrate
80      dcbStruktur.ByteSize = 8; // Bytegröße
81      dcbStruktur.StopBits = ONESTOPBIT; // Stopbit
82      dcbStruktur.Parity = NOPARITY; // Parität
83
84      erfolg = SetCommState(hCom, &dcbStruktur); // COM-Port-Status setzen
85      if (!erfolg) { // Wenn der COM-Port-Status nicht gesetzt werden kann,
    wird eine Fehlermeldung ausgegeben
86          std::cerr << "SetCommState fehlgeschlagen" << std::endl; //
    Fehlermeldung
87          CloseHandle(hCom); // COM-Port schließen
88          return 1; // Beendet das Programm, wenn der COM-Port-Status nicht
    gesetzt werden kann
```

```
89      }
90
91      // Benutzerdokumente-Ordner abrufen
92      char dokumentePfad[MAX_PATH]; // Puffer für den Dokumentenpfad (
        MAX_PATH Bytes)
93      HRESULT result = SHGetFolderPathA(NULL, CSIDL_PERSONAL, NULL,
        SHGFP_TYPE_CURRENT, dokumentePfad); // Benutzerdokumente-Ordner abrufen
94
95      if (!SUCCEEDED(result)) { // Wenn der Dokumentenpfad nicht abgerufen
        werden kann, wird eine Fehlermeldung ausgegeben
96          std::cerr << "Fehler beim Abrufen des Dokumentenpfads" << std::endl
        ; // Fehlermeldung
97          return 1; // Beendet das Programm, wenn der Dokumentenpfad nicht
        abgerufen werden kann
98      }
99
100     std::string gxpOrdnerPfad = std::string(dokumentePfad) + "\\GXP_Datei";
        // GXP_Datei-Ordnerpfad
101
102     // Überprüfen, ob der Ordner existiert, und ggf. erstellen
103     DWORD ftyp = GetFileAttributesA(gxpOrdnerPfad.c_str()); //
        Dateiattribut abrufen
104     if (ftyp == INVALID_FILE_ATTRIBUTES) { // Wenn das Dateiattribut nicht
        abgerufen werden kann, wird eine Fehlermeldung ausgegeben
105         if (!CreateDirectoryA(gxpOrdnerPfad.c_str(), NULL)) { // Wenn der
        Ordner nicht erstellt werden kann, wird eine Fehlermeldung ausgegeben
106             std::cerr << "Fehler beim Erstellen des Ordners GXP_Datei" <<
        std::endl; // Fehlermeldung
107             return 1; // Beendet das Programm, wenn der Ordner nicht
        erstellt werden kann
108         }
109     }
110     else if (!(ftyp & FILE_ATTRIBUTE_DIRECTORY)) { // Wenn das
        Dateiattribut nicht abgerufen werden kann, wird eine Fehlermeldung
        ausgegeben
111         std::cerr << "GXP_Datei ist kein Ordner" << std::endl; //
        Fehlermeldung
112         return 1; // Beendet das Programm, wenn GXP_Datei existiert, aber
        kein Ordner ist
```

```
113     }
114
115     // Erzeugen Sie einen Dateinamen basierend auf dem aktuellen Datum und
        der Uhrzeit
116     std::string datumUndUhrzeit = aktuellesDatumUndUhrzeitHolen(); //
        Aktuelles Datum und Uhrzeit abrufen
117     std::string dateiName = gxpOrdnerPfad + "\\Output_" + datumUndUhrzeit +
         ".gpx"; // Dateiname
118
119     std::ofstream gpxDatei(dateiName, std::ios::out); // GPX-Datei öffnen
120     if (!gpxDatei.is_open()) { // Wenn die GPX-Datei nicht geöffnet werden
        kann, wird eine Fehlermeldung ausgegeben
121         std::cerr << "Datei konnte nicht zum Schreiben geöffnet werden" <<
        std::endl; // Fehlermeldung
122         CloseHandle(hCom); // COM-Port schließen
123         return 1; // Beendet das Programm, wenn die GPX-Datei nicht geö
        ffnet werden kann
124     }
125
126     // Beginn der GPX-Datei schreiben
127     gpxDatei << "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n"; // XML-
        Deklaration
128     gpxDatei << "<gpx xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance
        \" xmlns=\"http://www.topografix.com/GPX/1/1\" xsi:schemaLocation=\"http
        ://www.topografix.com/GPX/1/1 http://www.topografix.com/GPX/1/1/gpx.xsd
        \" version=\"1.1\" creator=\"https://gpx.studio\">\n"; // GPX-Datei-
        Header
129     gpxDatei << "    <trk>\n"; // Track-Element
130     gpxDatei << "        <trkseg>\n"; // Tracksegment-Element
131
132     std::string aktuellesDatum = aktuellesDatumHolen(); // Aktuelles Datum
        abrufen
133     std::string daten; // Daten
134     std::string angesammelteDaten; // Angesammelte Daten
135
136     while (true) { // Endlosschleife
137         if (COMDatenLesen(hCom, daten)) { // Daten vom COM-Port lesen
138             angesammelteDaten += daten; // Daten anhängen
139
```

```
140             size_t endeDerNachricht; // Ende der Nachricht
141             while ((endeDerNachricht = angesammelteDaten.find("\r\n")) !=
    std::string::npos) { // Wenn das Ende der Nachricht gefunden wird
142                 std::string nachricht = angesammelteDaten.substr(0,
    endeDerNachricht); // Nachricht extrahieren
143                 angesammelteDaten.erase(0, endeDerNachricht + 2); //
    Nachricht löschen
144 
145                 // Datenanzeige im Terminal
146                 std::cout << "Empfangene Daten: " << nachricht << std::endl
    ; // Empfangene Daten im Terminal anzeigen
147 
148                 if (nachricht == "Lesen: AUS") { // Wenn die Nachricht "
    Lesen: AUS" ist, wird die Endlosschleife beendet
149                     goto abschluss;  // Sprung zum Dateiende
150                 }
151 
152                 // Die Positionen der relevanten Datenpunkte bestimmen
153                 size_t zeitPos = nachricht.find("Zeit: "); // Position der
    Zeit
154                 size_t breitePos = nachricht.find("Lat: "); // Position der
     Breite
155                 size_t laengePos = nachricht.find("Lon: "); // Position der
     Länge
156 
157                 if (zeitPos != std::string::npos && breitePos != std::
    string::npos && laengePos != std::string::npos) { // Wenn die Positionen
     der relevanten Datenpunkte gefunden werden
158                     std::string zeit = aktuellesDatum + "T" + nachricht.
    substr(zeitPos + 6, 8) + "Z"; // Zeit extrahieren
159 
160                     size_t breiteEndePos = nachricht.find(' ', breitePos +
    5); // Position des Leerzeichens nach der Breite
161                     if (breiteEndePos == std::string::npos || breiteEndePos
     > laengePos) { // Wenn das Leerzeichen nicht gefunden wird oder die
    Position größer als die Länge ist
162                         breiteEndePos = laengePos; // Position der Länge
163                     }
164                     std::string rohBreite = nachricht.substr(breitePos + 5,
```

```
    breiteEndePos - breitePos - 5); // Rohwert der Breite extrahieren
165                 char breitenrichtung = rohBreite.back(); // Richtung
    der Breite extrahieren
166                 std::string breite = BreitenLängengradKonvertieren(
    rohBreite, breitenrichtung); // Breite konvertieren
167
168                 size_t laengeEndePos = nachricht.find(' ', laengePos +
    5); // Position des Leerzeichens nach der Länge
169                 if (laengeEndePos == std::string::npos) { // Wenn das
    Leerzeichen nicht gefunden wird
170                     laengeEndePos = nachricht.length(); // Länge des
    Strings
171                 }
172                 std::string rohLaenge = nachricht.substr(laengePos + 5,
     laengeEndePos - laengePos - 5); // Rohwert der Länge extrahieren
173                 char laengerichtung = rohLaenge.back(); // Richtung der
     Länge extrahieren
174                 std::string laenge = BreitenLängengradKonvertieren(
    rohLaenge, laengerichtung); // Länge konvertieren
175
176                 gpxDatei << "            <trkpt lat=\"" << breite << "
    \" lon=\"" << laenge << "\">\n"; // Trackpunkt-Element
177                 gpxDatei << "                <time>" << zeit << "</time
    >\n"; // Zeit-Element
178                 gpxDatei << "            </trkpt>\n"; // Trackpunkt-
    Element
179             }
180         }
181     }
182     Sleep(100); // 100 Millisekunden warten
183 }
184
185 abschluss:
186     // Abschluss der GPX-Datei schreiben
187     gpxDatei << "        </trkseg>\n"; // Tracksegment-Element
188     gpxDatei << "    </trk>\n"; // Track-Element
189     gpxDatei << "</gpx>"; // GPX-Datei-Ende
190     gpxDatei.close(); // GPX-Datei schließen
191     CloseHandle(hCom); // COM-Port schließen
```

```
192
193     std::cout << "Die Daten wurden in " << dateiName << " gespeichert." <<
        std::endl; // Erfolgsmeldung
194     system("pause"); // Programm anhalten
195
196     return 0; // Programm beenden
197 }
```

Listing 5.2: vollständiger Quellcode der PC-Anwendung

Eidesstattliche Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig und nur unter Benutzung der angegebenen Literatur und Hilfsmittel angefertigt habe. Wörtlich übernommene Sätze und Satzteile sind als Zitate belegt, andere Anlehnungen hinsichtlich Aussage und Umfang unter Quellenangabe kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen und ist auch noch nicht veröffentlicht.

Lippstadt, den 29.02.2024

(Unterschrift des Verfassers)