

**Bidirektionales
Funkübertragungssystem
zur Steuerung von
Miniaturmödellfahrzeugen**

Masterarbeit von
Philipp Gahtow
1090126

Aufgabenstellung:
Prof. K. Buchenrieder, Ph.D.

Betreuung:
Dipl.-Medieninf. Andreas Attenberger

Abgabedatum: 06.08.2013

Universität der Bundeswehr München
Fakultät für Informatik

Selbstständigkeitserklärung

Ich versichere mit meiner Unterschrift, dass ich die vorliegende Arbeit - abgesehen von der Mitwirkung des genannten Betreuer - selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe. Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken als solche gekennzeichnet habe.

Neubiberg, den 06. August 2013

Unterschrift

Inhaltsverzeichnis

1 Einleitung	1
2 Technische Grundlagen	3
2.1 Arduino Leonardo (Atmega32U4)	3
2.2 Funkmodul RFM12B	5
2.2.1 Technische Daten	5
2.2.2 Arduino Library JeeLib	6
2.3 Verwendete Protokolle	8
2.3.1 P50X-Protokoll	8
2.3.2 LocoNet™	9
2.4 Computeranwendung	11
2.5 Leiterplattenerstellung mit Eagle™	12
3 Konzept der Steuerung	13
3.1 Car System Mechanik	13
3.2 Grundidee der Steuerung	14
4 Aufbau des Steuerungssystems	17
4.1 Zentrale	17
4.1.1 Kommunikation	17
4.1.2 Hardware	21
4.1.3 Software	23
4.2 Fahrzeugdekoder	26
4.2.1 Funktionsprinzip	26
4.2.2 Hardware	27
4.2.3 Software	30
4.2.4 Programmierung der Konfigurationsvariablen	35
4.2.5 Ladestation	35
5 Zusammenfassung	37
Tabellenverzeichnis	39
Abbildungsverzeichnis	41

Literaturverzeichnis	43
A Erläuterung P50X-Protokollbefehle	47
A.1 Systemdatenpakete	47
A.2 Lok Datenpakete	50
A.3 Zubehör Datenpakete	51
A.4 Rückmelde Datenpakete	52
A.5 Programmierung Datenpakete	52
B LocoNet™-Protokoll Befehle	55
B.1 2 Byte Nachrichten	55
B.2 4 Byte Nachrichten	56
B.3 6 Byte Nachrichten	57
B.4 Variable Nachrichtenlänge	57
C RCMC-Funk-Protokoll Befehle	59
C.1 Datenpakete von der Zentrale zu den Fahrzeugdekodern	59
C.2 Datenpakete von Fahrzeugdekomodern zur Zentrale	60
C.3 Datenübertragung zwischen der Zentrale und dem Fahrzeugdekomodern	61
D Schaltungen	63
E Konfigurationsvariablen des Fahrzeugdekomoders	65

Kapitel 1

Einleitung

Auf Modelleisenbahnanlagen befinden sich neben den fahrenden Zügen, meist auch Ortschaften mit Straßen, Parkplätzen und Kreuzungen. Die Straßen sind dabei oft mit Fahrzeugstandmodellen, die teilweise mit Beleuchtung ausgestattet sind, bestückt. Um einen solchen Aufbau realistischer wirken zu lassen, wurden von der Firma Faller analog steuerbare Modellfahrzeuge entwickelt. Die Funktionalität dieses Systems begrenzt sich auf das reine Fahren der Fahrzeuge. Im Laufe der Zeit wurde das Interesse an diesem System größer und Faller stelle 2011 erstmals digitalisierte Fahrzeuge vor. Im Jahr 2013 präsentierte die Firma Faller ihre neuste Generation von Modellfahrzeugen [13]. Aufgrund der hohen Entwicklungskosten ist dieses System sehr kostspielig.

Im Rahmen dieser Masterarbeit wurde ein drahtloses Funksteuerungssystem für Modellfahrzeuge entwickelt, welches darüber hinaus als Open-Source-System erstellt ist. Es basiert auf den Faller-Car-System-Fahrzeugen und orientiert sich dabei an der digitalen Zugsteuerung über eine Computeranwendung für Modelleisenbahnen. Das System stützt sich auf ein ebenfalls für Modellfahrzeuge einsetzbares Infrarotübertragungssystem [12]. Die eingesetzte Funkübertragung bietet gegenüber der Infrarotdatenübertragung eine wesentlich höhere Reichweite, höhere Übertragungsgeschwindigkeit und geringere Anfälligkeit gegen Störungen, die zum Beispiel durch Schatten oder starke Sonneneinstrahlung entstehen können.

Die neu entwickelte Funkübertragung ermöglicht dabei eine bidirektionale Funkkommunikation zwischen Modellfahrzeugen und einer Steuerzentrale. Damit ergeben sich zusätzliche Möglichkeiten der Steuerung. So lässt sich zum Beispiel der Ladezustand des Akkus jedes Fahrzeugs während der Fahrt abfragen. Sollte im Verlauf der Fahrt die Versorgungsspannung unter einen vorgegebenen Minimalwert sinken, lenkt die Computeranwendung das Modellfahrzeug automatisch in die Ladestation. Das Fahrzeug meldet diesen Zustand über die Funkverbindung einer zentralen Komponente. Diese Zentrale steuert, koordiniert und überwacht das Gesamtsystem und stellt die Schnittstelle zur Steuerungsanwendung auf einem Computer dar. Die in der hier vorliegenden Arbeit dargelegte Modellfahrzeugsteuerung wurde als Prototyp aus den beschriebenen Komponenten aufgebaut und in Betrieb genommen. Die Zielstellung war es, für die Kommunikation mit der Computeranwendung das weit verbreitete P50-Protokoll zu nutzen. Das Steuerungssystem, welches aus der Zentrale als Schnittstelle zwischen den

Fahrzeugen, der Fahrbahn und der Computeranwendung besteht, soll dabei ohne weitere Komponenten auskommen. Die Datenübertragung zu den Modellfahrzeugen erfolgt ausschließlich über Funk. Der optimierte und universell einsetzbare Funkfahrzeugdekomodulator stellt viele frei programmierbare Zusatzausgänge zur Verfügung. Durch die Motorsteuerung mit Lastregelung und der Versorgung über einen Lithium-Polymer-Akku bietet der Fahrzeugdekomodulator ein gutes Fahrverhalten auf der Modellfahrstraße.

In den folgenden Kapiteln werden Hardware und Software des erstellten Systems sowie die notwendigen Grundlagen zur Funktionsweise und Datenübertragung beschrieben. Die verwendeten Protokolle wurden speziell für die Steuerung von Modelleisenbahnen entwickelt. Sie werden in die erstellte Open-Source-Software integriert. Damit wird eine hohe Kompatibilität mit vorhandener Software und Hardware erzielt. Die bidirektionale Datenübertragung zwischen Fahrzeugen und Zentrale, wird durch ein eigens entwickeltes RCMC-(Radio Control for Model Cars)-Funk-Protokoll realisiert.

Kapitel 2

Technische Grundlagen

In diesem Kapitel werden die grundlegenden technischen Voraussetzungen zur Realisierung eines bidirektionales Funkübertragungssystem für Modellfahrzeuge erläutert. Dazu werden zu Beginn die verwendete Technik erklärt und die benutzten Protokolle erläutert.

2.1 Arduino Leonardo (Atmega32U4)

Arduino ist der Handelsname für ein Open Source Projekt¹. Mit der frei verfügbaren Software und den vielen preisgünstigen Entwicklungs-Boards können sehr komfortabel Anwendungsprogramme auf Mikrocontrollern realisiert werden. Die verschiedenen Entwicklungs-Boards sind ausnahmslos mit Mikrocontrollern der Firma ATMEL bestückt. Auf den Boards befinden sich Mikrocontroller der AVR-Familie mit unterschiedlicher Leistung und Größe [1]. Die hier als Prototyp aufgebaute Modellfahrzeugsteuerung verwendet ein Arduino Leonardo Entwicklungs-Board (siehe Abbildung 2.1) [2].

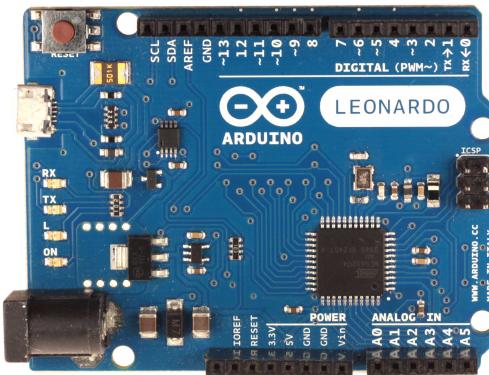


Abbildung 2.1: Arduino Leonardo Entwicklungs-Board [2].

Das Entwicklungs-Board ist mit dem Mikrocontroller Atmega32U4 bestückt [3].

¹Webseite: <http://www.arduino.cc>

Dieser Mikrocontroller ist nur in der SMD²-Bauform mit 44 Pins erhältlich. Er verfügt über 24 programmierbare Ein/Ausgänge. Davon können 12 als analoge Eingänge mit einer Auflösung von 10 Bit genutzt werden. Der Arduino Leonardo bietet im Gegensatz zu anderen Arduino-Entwicklungs-Boards eine im Mikrocontroller integrierte USB³-2.0-Schnittstelle und benötigt somit keinen zusätzlichen Prozessor für die USB-Verbindung [2].

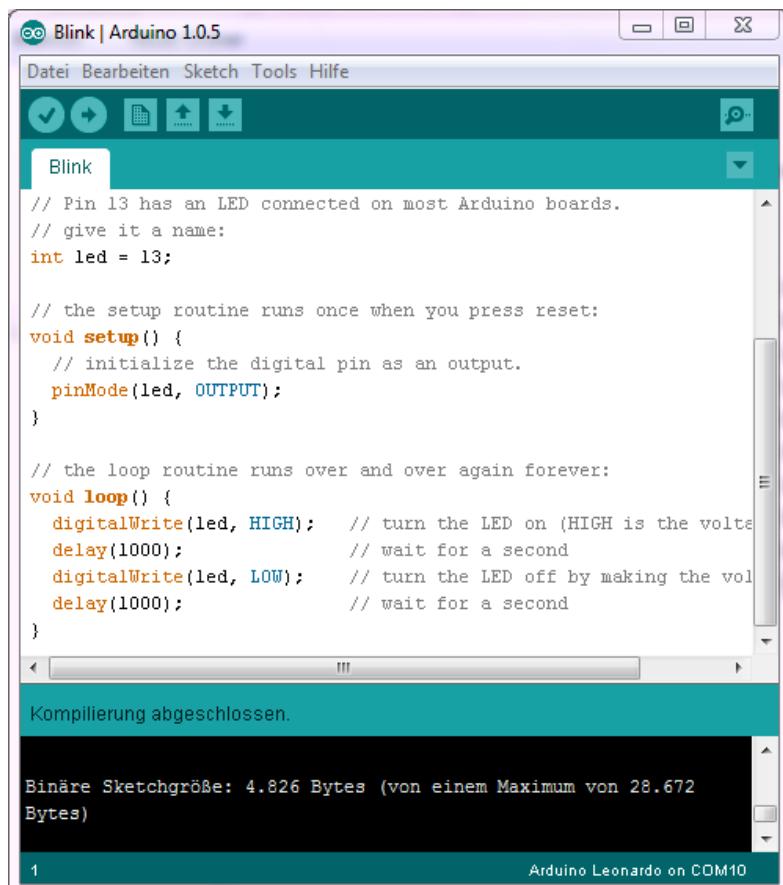


Abbildung 2.2: Arduino IDE Entwicklungsumgebung.

Um den Mikrocontroller auf dem Arduino-Board für unterschiedliche Anwendungsprogramme nutzen zu können, wird ein Bootloader mitgeliefert. Dieses Programm steuert das Schreiben neuer Software über die USB-Schnittstelle in den Speicher des Mikrocontrollers. Er muss bei der Inbetriebnahme eines neuen Mikrocontrollers zuerst über die ICSP-(In Circuit Serial Programming)-Schnittstelle eingespielt werden. Zusätzlich müssen Takt- und Bootloader-Optionen in den Mikrocontroller geschrieben werden. Danach ist die integrierte USB-Schnittstelle aktiv und kann über das Programm *avrdude* für das

²Surface-Mounted-Device (SMD), ist ein Gehäuse ohne Drahtanschlüsse, welches dadurch eine sehr kompakte Bauform besitzt.

³Universal-Serial-Bus (USB) ist ein serielles Bussystem zur Verbindung eines Computers mit externen Geräten.

Einspielen von Anwendungsprogrammen genutzt werden. Die USB-Schnittstelle wird vom Betriebssystem als ein normaler RS-232 COM-Port erkannt.

Arduino-Boards werden über eine Open-Source Entwicklungsumgebung der sogenannten Arduino-IDE⁴ programmiert. Diese bietet eine Vielzahl von Bibliotheken, die einen sehr einfachen Zugriff auf die interne und externe Hardware ermöglichen. So lassen sich Funktionen, wie die digitale Signalausgabe, EEPROM⁵-Zugriff oder das Einlesen analoger Werte, schnell und unkompliziert realisieren (siehe Abbildung 2.2).

2.2 Funkmodul RFM12B

Die Datenübertragung zwischen den Fahrzeugen und der Zentrale wird über eine Funkverbindung hergestellt. Das verwendete Funkmodul RFM12B (siehe Abbildung 2.3) ermöglicht einen bidirektionalen Datenverkehr.

2.2.1 Technische Daten

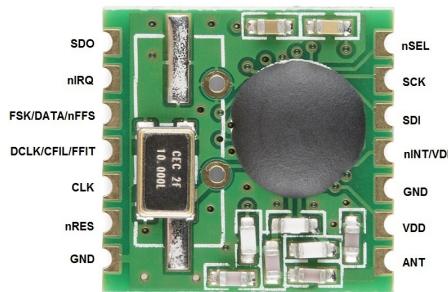


Abbildung 2.3: Funkmodul RFM12B-S2 [16].

Das RFM12B Funkmodul von Hersteller HopeRF ist ein Mehrkanal-TRX (Transceiver) Modul, für die Trägerfrequenzen 433, 868 und 915 MHz und arbeitet mit einer Versorgungsspannung von 2,2 Volt bis 3,8 Volt [16]. Die maximale Übertragungsrate im Digital-Modul beträgt 115,2 kbps⁶ beziehungsweise 256 kbps im Analog-Modus. Alle, für den bidirektionalen Datenverkehr notwendigen Komponenten sind auf dem Chip des Funkmoduls integriert. Somit sind keine zusätzlichen Bauelemente außerhalb des Moduls für den Betrieb erforderlich. Über die serielle Schnittstelle SPI (Serial-Peripheral-Interface) kann das Funkmodul mit dem Mikrocontroller kommunizieren. Das verwendete Funkmodul RFM12B S2 hat SMD-Bauform ausgeführt und hat 14 Anschlusspins. Die Leiterplattenabmessung sind 15,9 x 16,1 x 2,2 mm [16]. Bei der Übertragung von Digitalsignalen über Funk wird die FSK (Frequency-Shift-Keying), Frequenzumtastung angewendet. Diese Modulationstechnik ist der analogen Frequenzmodulation verwandt und zeichnet sich durch hohe Störsicherheit aus [22].

⁴Arduino Integrated Development Environment (IDE) ist eine integrierte Programmierentwicklungsumgebung. Webseite: <http://arduino.cc/en/Main/Software>

⁵Electrically Erasable Programmable Read-Only Memory

⁶Kilobit pro Sekunde

Für den Funkbetrieb wurde eine Trägerfrequenz von 868 MHz ausgewählt. Dies geschah aus den folgenden Gründen: Der Sendebereich geht von 15 bis 240 kHz und der Empfangsbereich liegt bei 67 bis 400 kHz [16]. Somit wird hier ein sehr schmalbandiges Übertragungssystem genutzt. Das lässt wiederum auf hohe Störsicherheit und geringe Frequenzinterferenzen schließen. Weiterhin besteht die Tatsache, dass die hohe Trägerfrequenz mit 868 MHz bisher viel weniger genutzt wird, als die wesentlich niedrigere Frequenz von 433 MHz. Die Trägerfrequenz von 915 MHz ist im Deutschland nicht zulässig [5].

An der SMD Ausführung des RFM12B sollte eine auf die Übertragungsfrequenz abgestimmte Antenne angeschlossen werden, da auf dem Modul keine Leiterbahnantenne vorhanden ist. Eine ideale Länge für 868 MHz sind 82,2 mm, was einem Viertel der Wellenlänge entspricht [16].

2.2.2 Arduino Library JeeLib

Zur einfachen Kommunikation zwischen Funkmodul und Mikrocontroller werden die Programme der JeeLib-Bibliothek verwendet. Diese Bibliothek ist eine Sammlung von Beispielprogrammen und Klassen für die Arduino IDE [19]. Unterstützt wird die Arduino IDE ab der Version 1.0 und folgende. Die Bibliothek wurde im Laufe der Jahre immer weiterentwickelt und soll dem Nutzer eine möglichst einfache Handhabung bieten. Unter anderem enthält die JeeLib-Bibliothek einen kompletten Treiber für die Funkmodule RFM12 und RFM12B. Der Treiber ist in der Programmiersprache C geschrieben und unterstützt die ATMEL Mikrocontroller ATmega und ATTiny. Es werden vier verschiedene Kategorien von Funktionen zur Ansteuerung des Funkmoduls RFM12 angeboten [19]:

- **Hauptfunktionen:**

- **`uint8_t rf12_initialize(uint8_t ID, uint8_t band, uint8_t group)`**
Konfiguration des RFM12 Funkmodul unter Angabe der ID (0...31), Frequenz (RF12_433MHZ = 1, RF12_868MHZ = 2, RF12_915MHZ = 3) und Gruppe (0...212).
- **`uint8_t rf12_recvDone()`**
Durch den Aufruf dieser Funktion wird der Datenempfang aktiv. Bei der Rückgabe von wahr wurde ein Datenpaket erhalten.
- **`uint8_t rf12_canSend()`**
Prüft ob es möglich ist ein Datenpaket zu senden.
- **`void rf12_sendStart(uint8_t header, const void *ptr, uint8_t length)`**
Schaltet in den Datenübertragungsmodus ein und sendet das Datenpaket.
- **`void rf12_sendNow(uint8_t header, const void *ptr, uint8_t length)`**
Wartet bis eine Datenübertragung möglich ist und sendet anschließend das Datenpaket.

- **Einfaches Senden:**

- **void rf12_easyInit(uint8_ secs)**
Startet den einfachen Übertragungsmodus. Übergeben wird dabei nur die zu nutzende Übertragungsfrequenz.
- **char rf12_easyPoll()**
Durch den Aufruf dieser Funktion wird der einfache Übertragungsmodus aktiv gehalten.
- **char rf12_easySend(const void *data, uint8_t size)**
Sende Daten mit dem einfachen Übertragungsmodus.

- **Alternatives Initialisieren:**

- **void rf12_set_cs(uint_8 pin)**
Setzt den Chip-Select (CS or SS) Anschluss für das Funkmodul am Mikrocontroller.
- **void rf12_spiInit()**
Initialisiert den SPI-Bus.
- **uint8_t rf12_config(uint8_t show)**
Holt die Konfiguration des Funkmoduls aus dem EEPROM und initialisiert die Datenübertragung.

- **Andere Funktionen:**

- **uint16_t rf12_control(uint16_t cmd)**
Direktes Schreiben der Register des RFM12 Funkmoduls.
- **void rf12_encrypt(const uint8_t *key)**
Aktiviert die Verschlüsselung der Datenübertragung mit einer Schlüssellänge von 128 Bit.
- **char rf12_lowbat()**
Gibt wahr zurück wenn die Versorgungsspannung unter 3,1 Volt absinkt.
- **void rf12_onOff(uint8_t value)**
Schaltet das Funkmodul ein oder aus.
- **void rf12_sendWait(uint8_t mode)**
Das Funkmodul wartet bis das Senden der Daten abgeschlossen ist. Über den Modus kann während des Warten das Funkmodul in den Energiesparmodus versetzt werden.
- **void rf12_sleep(char n)**
Versetzt das Funkmodul in den Schlafmodus. Bei Übergabe von 1...127 wird das Modul automatisch nach der eingestellten Zeit (n * 32 Millisekunden) aufgeweckt und löst eine Unterbrechungsfunktion aus.

Durch die JeeLib-Bibliothek wird dem Programmierer ein einfacher Zugang zu allen Funktionen der RFM12- und RFM12B-Module gegeben. Sie übernimmt die komplette

Kommunikation mit dem Funkmodul. Die Kommandos und Daten werden als 16-Bit-Pakete über das SPI-Protokoll versendet oder empfangen. Die Datenübertragung erfolgt dabei immer mit dem höherwertigen Bit zuerst. Vor Beginn der Datenübertragung muss das Funkmodul mit dem Befehl `rf12_initialize()` initialisiert werden.

2.3 Verwendete Protokolle

Im folgenden Abschnitt werden die zur Steuerung verwendeten Übertragungsprotokolle beschrieben. Es handelt sich um Protokolle die speziell für Modelleisenbahn-Systeme entwickelt wurden und gegenwärtig dort ihren Einsatz finden. Eine Ausnahme bildet das für den hier vorliegenden speziellen Einsatzfall entwickelte RCMC-Funk-Protokoll. Es realisiert die Funkdatenübertragung zwischen der Zentrale und den Modellfahrzeugen und wird im Kapitel 4.1.1 näher erläutert.

2.3.1 P50X-Protokoll

Das P50-Protokoll wurde von der Firma Märklin™ für die Kommunikation zwischen einem Steuerungscomputer und der Modelleisenbahnzentrale entwickelt. Es ist kompatibel mit der Steuerungssoftware verschiedener Modelleisenbahnsysteme. Die Steuerungsdaten werden über die serielle RS-232-Schnittstelle mit 8 Datenbits, 2 Stopbit (8N2), jedoch ohne Paritybit zur Modelleisenbahnzentrale übertragen. Die Baudrate der Übertragung richtet sich nach der verwendeten Steuerungssoftware und erfolgt mit 2400, 4800, 9600, 19200, 38400, 57600 oder 115200 Baud. Die Erkennung der Baudraten erfolgt über die automatische Baudratenidentifizierung (Break and Automatic Baud-rate Identification, kurz BABI) zu Beginn der Kommunikation. Für eine erfolgreiche Kommunikation mit der Computeranwendung ist zusätzlich ein aktiver CTS⁷-Hardware-Handshake erforderlich.

Da die Anzahl der Befehle des P50-Protokolls nicht ausreicht, wurde das Protokoll erweitert. Daraus entstand die Kommandoerweiterung P50X. Um diese erweiterte Kommandostruktur anzusprechen, muss vor jedem P50X-Befehl als erstes Byte, ein 0X58 (ASCII "X") oder 0X78 (ASCII "x") gesendet werden. Ist das darauf folgende Byte kleiner oder gleich 0X7F, handelt es sich um die ASCII Kommando-Erweiterung P50Xa. Ist das zweite Byte größer gleich 0X80, wird die Binary-Kommando-Erweiterung P50Xb angesprochen [11]. Im Rahmen dieser Arbeit wird ausschließlich die Binary-Kommandostruktur verwendet. Dazu wurde die Dekodierung der Befehle der Binärerweiterung des P50Xb-Protokolls in einer Arduino-Bibliothek umgesetzt.

Das P50X-Protokoll wird ausschließlich zwischen der Computeranwendung (siehe Abschnitt 2.4) und der Zentrale (siehe Abschnitt 4.1) verwendet. Der Aufbau der Datenpakete des P50Xb-Protokolls ist im Anhang A aufgeführt. Dazu wurden diese zur besseren Übersicht in vier Kategorien eingeteilt: Lok-, Zubehör-, Rückmelde- und Programmierbefehle. Im Weiteren wird das Kommunikationsprotokoll zwischen der Zentrale und der Fahrstraße erläutert.

⁷Clear to send, Datenflussteuerung bei der Datenübertragung.

2.3.2 LocoNet™

Mit dem LocoNet™-Protokoll wird die Verbindung zwischen der Zentrale zur Hardware der Fahrstraße hergestellt. Das LocoNet™-Busübertragungssystem funktioniert anähernd wie ein Computernetzwerk. Es wurde von der Firma Digitrax⁸ speziell für die Anwendung auf Modelleisenbahnanlagen konzipiert [9]. Das System besitzt einen Master und arbeitet nach dem Peer-to-Peer Prinzip. Der Master stellt den Hauptknoten dar und ist für die Datenverwaltung zuständig. Er bildet gleichzeitig die Schnittstelle zu den anderen Datenübertragungsprotokollen. Das Netzwerk kann in beliebiger Topologie aufgebaut werden. Eine Terminierung der Enden, ist nicht erforderlich und ein Ringaufbau nicht möglich [7]. Nachfolgend wird die Datenübertragung und der Aufbau der Datenpakete beschrieben.

Datenübertragung

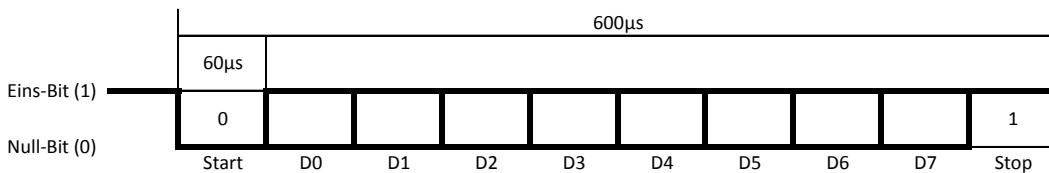


Abbildung 2.4: Aufbau eines LocoNet™-Datenpaketes [4].

Die Datenübertragung über das Bussystem erfolgt seriell mit der im Standard festgelegten 16660 Baud. Für eine sichere Datenübertragung darf die Baudrate eine Abweichung von maximal $\pm 1,5\%$ nicht über- oder unterschreiten. Die Zeit für Bit beträgt so 60 Mikrosekunden (ms). Übertragen werden dabei acht Datenbit gefolgt von einem Stopbit (8N1) (siehe Abbildung 2.4). Da das LocoNet™-Bussystem über nur eine einzige Datenleitung verfügt, wird als Kollisionsschutz für die Übertragung das Carrier Sense Multiple Access with Carrier Detect (kurz: CSMA/CD) verwendet [8]. Dabei wird vor dem Senden eines Datenpaketes geprüft, ob das Netz möglicherweise schon durch eine anderen Knoten im Netzwerk belegt ist. Erst wenn keine Daten übertragen werden, kann eine anderer Knoten sein Datenpaket senden. Während der Übertragung wird ständig das Signal der Datenleitung auf die Übereinstimmung mit dem aktuell gesendeten Pegel des Knoten geprüft. Eine Kollision wird erkannt, wenn ein Eins-Bit übertragen wird und auf der Datenleitung ein Null-Bit anliegt. In diesem Fall wird das Senden abgebrochen und eine Datenpause mit 15 Null-Bit übertragen, bevor erneut Daten gesendet werden können.

Alle Knoten legen beim Senden nur den Low-Pegel (Null-Bit) an die Datenleitung. Eine auf die Datenleitung geschaltete Stromquelle mit 15 bis 20 mA erzeugt den High-Pegel. Das Netzwerk kann außerdem mit der oben geschilderten Zugangskontrolle Datenpakete priorisieren. Der 20 Bit lange Carrier Detect (CD) definiert das Ende eines jeden Datenpaketes. Daraufhin folgen sechs Bit Wartezeit (Master Delay = MD) in welcher es dem Master möglich ist, vor allen anderen Knoten im Netzwerk zu senden.

⁸Webseite: <http://www.digitrax.com/>

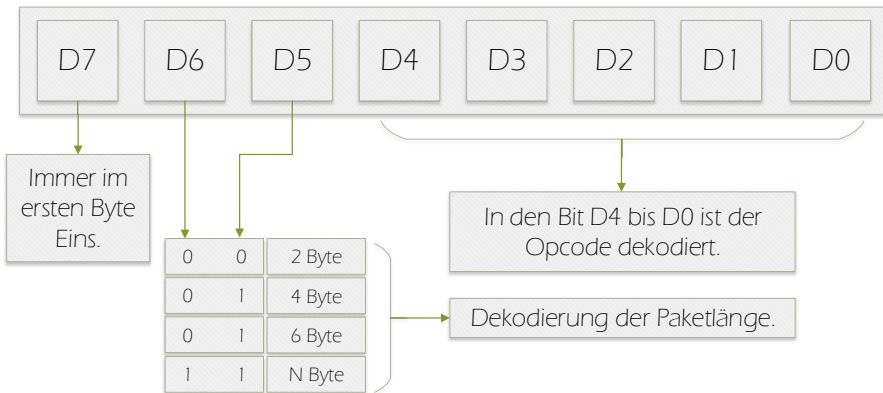


Abbildung 2.5: Kodierung der Nachrichtenlänge und des Opcode im ersten Byte der LocoNet™-Nachricht [4].

Zur Priorisierung einzelner Stationen folgt auf den Master Delay eine weitere Wartezeit (Priority Delay, kurz PD) von 0...20 Bit. Je nach zuvor festgelegter Priorität eines Knoten, muss dieser vor Sendebeginn die jeweilige Anzahl von Bit abwarten. Je höher die Priorität eines Knoten ist, umso schneller kann dieser auf das Netzwerk zugreifen [4].

Datenpakete

Ein LocoNet™-Datenpaket beginnt immer mit einem Byte, dessen höchstwertigstes Bit Eins ist. Bei den folgenden Byte einer Nachricht ist dieses ein Null-Bit. Das erste gesendete Byte wird als Opcode bezeichnet. Dieser gibt an, wie viele und welche Datenpakete folgen. Das erste Bit D7 im Opcode ist somit immer ein Eins-Bit. In den beiden folgenden Bit D6 und D5 ist die Nachrichtenlänge kodiert. So sind Paketlängen von zwei-, vier-, sechs- oder N-Byte möglich. Die restlichen fünf Bit D4 bis D0 ergeben 32 mögliche Opcodes. Ist das Bit D3 ein Eins-Bit, so folgt eine weitere Nachricht oder ein Antwort-Byte [8]. Der Aufbau des ersten Byte einer Loconet™-Nachricht ist in der Abbildung 2.5 dargestellt.

Je nach kodierter Nachrichtenlänge folgt auf den Opcode die eigentlichen Nutzdaten. Den Abschluss eines Datenpaketes bildet die Prüfsumme (siehe Abschnitt 2.3.2). In Abbildung 2.6 ist eine solche LocoNet™-Nachricht dargestellt.



Abbildung 2.6: Aufbau eines LocoNet™-Datenpaket [4].

Ein Datenpaket kann den Weichenstatus, Signalzustand, Rückmeldungen, Programmieranweisungen, Uhrzeit oder Lokdatenänderungen enthalten. Das LocoNet™ ist vor allem darauf ausgelegt, Loksteuerungsdaten zu verwalten. Dazu gibt es ein Slot-System mit bis zu 127 Einträgen. Jeder Slot enthält ein 14-Bit-Adressfeld, den Status des Slots

(free, common, idle, IN_USE), das Lokprotokoll mit der zugehörigen maximalen Anzahl an Geschwindigkeitsstufen (14, 28 oder 128), den Status der Lok mit 7 Bit für die Geschwindigkeit und 9 Bit für die Funktionen F0 bis F8, Bestandsdaten (consist tree) und eine 14 Bit ID des Steuerungsgeräte [4]. Zur besseren Übersicht sind die Datenpakete nach Anzahl der Datenbytes geordnet und im Anhang B aufgeführt.

Prüfsumme

Jede LocoNet™-Nachricht enthält als letztes Byte eine 8-Bit-Prüfsumme. Auch bei der Prüfsumme ist das höchstwertige Bit so wie bei allen anderen Datenbytes außer dem OP-CODE auf Null gesetzt. Die Prüfsumme wird aus dem Einerkomplement eines Exklusiv-Oder (XOR⁹) über alle Bytes einer Nachricht außer der Prüfsumme selbst berechnet. Bei Empfang der Nachricht wird die Korrektheit des Datenpaket mit Hilfe der Prüfsumme bestimmt [26].

2.4 Computeranwendung

Eine herkömmliche analoge Modelleisenbahn wird per Handregler gesteuert. Ein solches System ist leicht aufzubauen und bequem zu betreiben. Sollen mehrere Züge gleichzeitig fahren, wird es durch Abtrennung von Gleisabschnitten und Weichenansteuerung schnell zu einem komplizierten System mit vielen Kabeln kommen. Übersicht bietet dann nur eine digitale Modelleisenbahnsteuerung. Eine Computer-Software stellt ein Abbild aller Komponenten dar und ist in der Lage den Betrieb der kompletten Anlage zu steuern. So können zum Beispiel per Klick mehrere Weichen, die eine Weichenfahrstraße bilden, gleichzeitig geschaltet werden. Über die Rückmeldung der Weichenstellung kontrolliert die Steuerungssoftware die korrekte Position und verhindert damit Zusammenstöße oder Entgleisungen. So lässt sich ein vollautomatischer Betrieb mit Fahrplänen und millimetergenauem Halten realisieren. Auch beliebige Zusatzfunktionen wie zum Beispiel das schalten des Raumlicht oder Geräuscheffekte sind möglich [17].

Zur Steuerung der hier beschriebenen Miniaturmodellfahrzeuge wird die Modelleisenbahn-Software Rocrail¹⁰¹⁰ eingesetzt. Dies ist eine Open-Source-Software für Linux, Mac OS X, Raspberry Pi und Windows Betriebssysteme. Über die Steuerungssoftware Rocrail¹⁰ können dabei Lokomotiven direkt gesteuert. Durch den Erhalt von Rückmeldeereignissen, wie Gleiskontakte, Weichen- und Signalstellung, wird eine vollautomatische Fahrsteuerung über die gesamte Modelleisenbahnlandschaft möglich. Die Rocrail¹⁰-Software selbst besteht dabei aus zwei voneinander unabhängigen und über das TCP/IP¹¹-Protokoll kommunizierenden Programmteilen [27]. Der Rocrail¹⁰-Server repräsentiert die Struktur der angeschlossenen Modelleisenbahn. Er kommuniziert mit der Digital-Zentrale und dem Rocrail¹⁰-Viewer (Rocview).

⁹Ein exklusives ODER (XOR), ist im Ergebnisbit Eins, falls die beiden zu vergleichenden Bit unterschiedlich sind. Das Ergebnisbit liefert Null, falls sie gleich sind.

¹⁰Webseite: <http://www.rocrail.net>

¹¹Transmission Control Protocol/ Internet Protocol

Rocview ist eine Anwendung mit grafischer Benutzeroberfläche. Sie stellt Menüstrukturen und Dialoge zur Konfiguration bereit. Der erzeugbare Gleisplan bietet eine Übersicht über die komplette Modelleisenbahnanlage und stellt zusätzlich alle Ereignisse im Betrieb dar.

Zur Steuerung der Modellfahrzeuge musste die Rocrail®-Software nicht angepasst werden. Es wurde lediglich beim Erstellen der Fahrstraßen darauf geachtet, das die Fahrzeuge sich wie im Straßenverkehr verhalten und nicht kollidieren. Gefahren wird immer von einem Block zu nächsten Block. Allerdings existieren keine Wartezeiten in den Blöcken (Stop and Go) und jeder Block besitzt einen eigenen Rückmelder. Außerdem darf sich in jedem Block nur ein Fahrzeug befinden. So wird das Auffahren auf ein anderes Fahrzeug verhindert. An Abzweigungen, zum Beispiel an einer Ampelkreuzung, wurde zu den jeweiligen Fahrtrichtungsänderungen die Aktion Blinker einschalten hinzugefügt.

Zur Realisierung des prototypisch aufgebauten Fahrzeugsteuerung, wurden verschiedene Leiterplatten entwickelt und hergestellt. Der nächste Abschnitt beschreibt die zur Leiterplattenerstellung verwendete Software, Eagle™.

2.5 Leiterplattenerstellung mit Eagle™

Für die Fahrzeugdekoder in den Miniaturfahrzeugen musste aufgrund der sehr geringen Abmessungen der Modellfahrzeuge eine spezielle kleine Leiterplatte entworfen werden. Das Arduino Leonardo Entwicklungs-Board konnte aus Platzgründen dazu nicht verwendet werden. Das Entwicklungs-Board ist fast doppelt so groß wie ein Modellfahrzeug. Den Fahrzeugabmessungen entsprechend wurde für den Fahrzeugdekoder ein spezielles prototypisches Platinen-Layout mit dem Entwicklungsprogramm Eagle™¹² entworfen. Die Software Eagle™ besteht aus dem Layout-Editor, dem Schaltplan-Editor, dem Autorouter und einer Bauteildatenbank [6]. Die Bauteildatenbank kann mit eigenen Bauteilentwürfen erweitert werden. So lassen sich beliebige Leiterplatten für unterschiedliche Verwendungen herstellen. Zuerst wird der gewünschte Schaltplan in Schaltplan-Editor erstellt. Ist dieser vollständig bestückt und die Komponenten besitzen die korrekte Gehäusegröße wird im Layout-Editor die Leiterplatte konfiguriert. Dazu müssen die Komponenten optimal plaziert werden und mit der Leiterbahnen verbunden werden. Beim Verbinden der Pins mit Leiterzügen hilft der Eagle™ Autorouter. Dieser sucht, nach den eingestellten Kriterien, den besten Verlauf für jeden noch nicht verbundenen Leiterzug. Auf diese Weise lassen sich einfache als auch komplizierte Leiterplatten herstellen.

¹²Webseite: <http://www.cadsoft.de>

Kapitel 3

Konzept der Steuerung

In diesem Kapitel wird die Funktionsweise der Fahrzeuge erläutert. Es werden dabei die erforderlichen Komponenten in einem Modellfahrzeug aufgezeigt und im Weiteren die Funktionsweise der digitalen Datenübertragung beschrieben.

3.1 Car System Mechanik

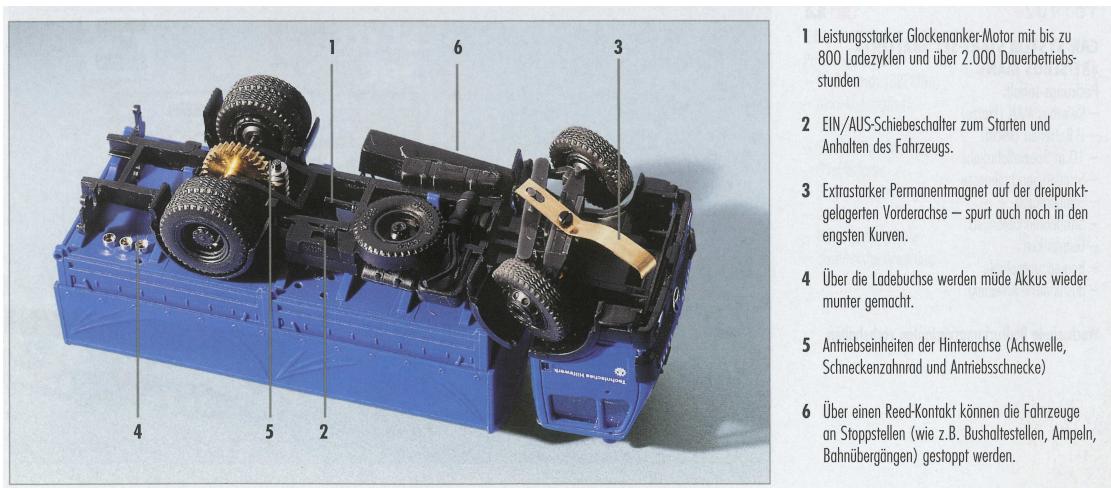


Abbildung 3.1: Komponenten eines originalen Modellfahrzeuges der Firma Faller [14].

Das in dieser Arbeit genutzte Konzept der fahrenden Modellfahrzeuge innerhalb von Modelleisenbahnanlagen wurde ursprünglich von der Firma Faller als Car System entwickelt. Das System basiert dabei auf einer magnetischen Spurführung die in der Fahrstraße verlegt wird. Durch magnetische Lenkschleifer an der Vorderachse der Fahrzeuge werden diese in der Spur gehalten [14]. Damit die Fahrzeuge auch ihre Richtung ändern können, wird die magnetische Spur in der Fahrstraße durch einen Servo-Antrieb, unterhalb der Fahrstraße umgelenkt. So lassen sich Kreuzungen und Parkplätze mit mehreren Abstellflächen realisieren. Die Abbildung 3.1 zeigt ein solches Fahrzeug mit einer

beweglichen Vorderachse und einem Lenkschleifer mit Magnet. An der Hinterachse befindet sich ein Miniaturmotor der von einem kleinen Akku versorgt wird.

Das ursprüngliche System der Firma Faller lässt die Fahrzeuge über einen Reedkontakt¹, der sich an der Unterseite befindet und einer steuerbaren Magnetspule in der Fahrstraße an geeigneten Positionen anhalten. Dabei müssen vorher alle Haltepositionen und Systemabläufe genau geplant und die Fahrstraße speziell darauf angepasst werden. Eine Elektronik innerhalb der Modellfahrzeuge gab es nicht. Die Abbildung 3.2 zeigt den aufgebauten Prototyp der Fahrstraße.

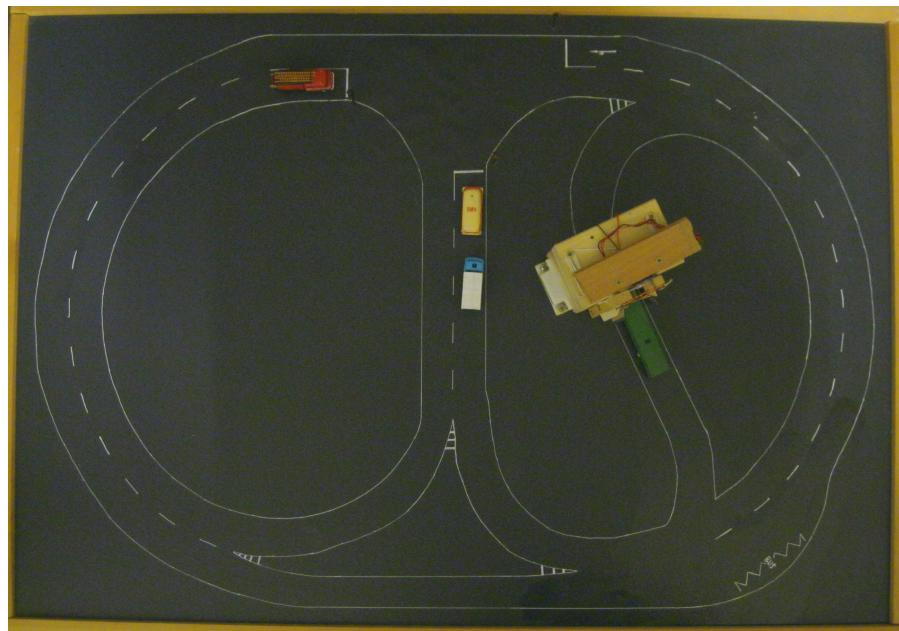


Abbildung 3.2: Prototyp der für Fahrversuche genutzten Fahrstraße.

3.2 Grundidee der Steuerung

Im Gegensatz zur herkömmlichen Technik der Firma Faller, besitzen die hier vorgestellten Fahrzeuge einen Fahrzeugdecoder mit Funkmodul. Über diesen Funkfahrzeugdecoder und dessen Software werden alle Betriebsabläufe des Fahrzeugs gesteuert. Neben der Motorsteuerung lassen sich auch das Licht, Bremslicht, Blinker oder Geräusche schalten. Zur exakten Steuerung mehrerer Fahrzeuge, muss die Kommunikation über den Funkkanal nahezu in Echtzeit ablaufen. Für die Datenübertragung wurde ein eigens zu diesem Zweck entwickeltes RCMC-(Radio Control for Model Cars)-Funk-Protokoll (siehe Abschnitt 4.2) geschrieben. Es deckt den kompletten Funktionsumfang des der Abbildung 3.3 dargestellten Fahrzeugdecoders ab. Das bidirektionale Funkübertragungssystem zur Steuerung von Miniaturmodellfahrzeugen ist eine Erweiterung des in der vorliegenden Bachelorarbeit entwickelten Infrarotübertragungssystems [12].

¹Ein Reedkontakt oder Reedschalter ist ein Glasrohr mit eingeschmolzenen Kontakten die durch ein Magnetfeld betätigt werden können [15].

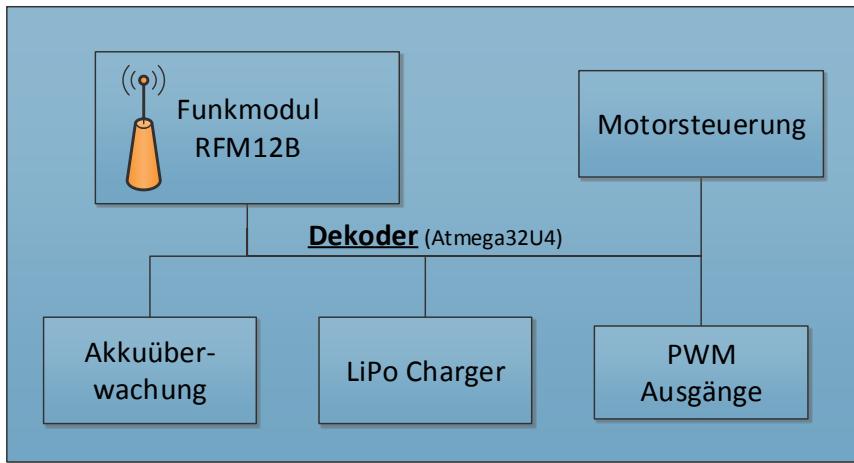


Abbildung 3.3: Blockbild aller Systemkomponenten der Funkfahrzeugsteuerung und deren Kommunikationsprotokollen.

Durch die nun mögliche direkte Rückkommunikation aus den Modellfahrzeugen zur Fahrstraße und der Computeranwendung wird ein zentrales Steuerelement zwischen den eingesetzten Protokollen benötigt. In Abbildung 3.4 sind die einzelnen Komponenten und deren Kommunikationsprotokolle dargestellt. Als zentrales Steuerelement (siehe Abschnitt 4.1) kontrolliert die Zentrale die komplette Kommunikation zwischen allen eingesetzten Komponenten. Sie ist dafür verantwortlich, bei einer Statusänderung die Steuerungsdaten an das Zielkommunikationsprotokoll weiterzuleiten. Dazu müssen die Daten vorher in das Zielprotokoll der Empfängerkomponenten übersetzt werden. Durch die Verwendung des P50X-Protokolls (siehe Kapitel 2.3.1) für die Kommunikation zwischen dem Computer und der Zentrale werden eine Vielzahl an unterschiedlichen Modelleisenbahn-Software unterstützt. In dieser Arbeit wird dabei Open-Source-Software Rocail(c) verwendet.

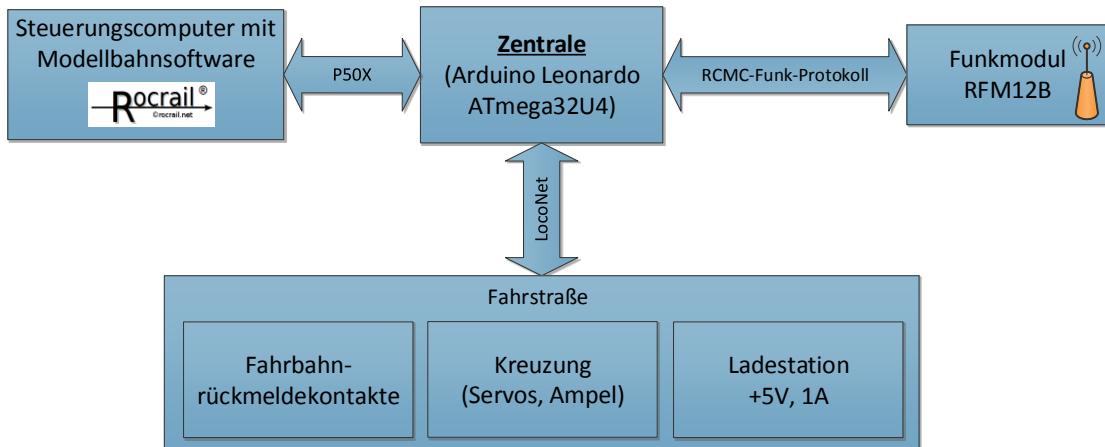


Abbildung 3.4: Blockbild des gestamten Funktionsaufbaus.

Zur Erkennung der Fahrzeuge auf der Fahrstraße durch die Computeranwendung werden zwischen der magnetischen Spurführung Hallsensoren eingelassen. Diese Sensoren bilden durch ihren Abstand einen Streckenabschnitt (Block), in dem sich immer nur ein Fahrzeug befindet. Die Hallsensoren generieren ein elektrisches Signal, wenn ein Magnetfeld sie passiert. Jedes Fahrzeug hat auf dem Lenkschleifer einen Magneten der es in der Fahrspur hält. Dieser Magnet löst beim Überfahren des Hallsensor ein Signal aus, welches über die Zentrale an die Computeranwendung weitergeleitet wird. Die Kommunikation zwischen Fahrstraße und Zentrale geschieht über das LocoNetTM-Protokoll (siehe Kapitel 2.3.2). Über dieses netzwerkähnliche Protokoll werden Rückmeldeereignisse, Steuerungsbefehle für Abzweigungen an Ampelkreuzungen, sowie die Kommunikation mit der Ladestation abgewickelt. Da die Befehle des Protokolls sich stark an der Modelleisenbahnsteuerung orientieren, können keine beliebigen Datenpakete übertragen werden. Das hier eingesetzte LocoNetTM-Protokoll unterstützt nur den in dieser Arbeit zur Kommunikation notwendigen Teil des kompletten LocoNetTM-Befehlsumfangs.

Durch die Reduzierung der Befehle und Befehlslängen rücken Zeitprobleme in den Hintergrund, die sich anderfalls bei der Funkdatenübertragung zu einer Vielzahl von Fahrzeugen auf der Fahrstraße bemerkbar machen würden. So gelingt es mit dem Steuerungssystem von Zentrale, Fahrzeugdekodern und der Computeranwendung einen vollständigen automatischen Betriebsablauf zu realisieren. Im Weitern wird der realisierte Aufbau von Zentrale und Fahrzeugdekoder dargelegt. Dabei wird auf die prototypisch erstellte Hardware und Software eingegangen und deren Funktion ausführlich erläutert.

Kapitel 4

Aufbau des Steuerungssystems

Das Steuerungssystem besteht aus der Softwareanwendung Rocrail©, einer Zentrale, der Fahrstraße und mehreren Fahrzeugen mit Funkfahrzeugdekodern. Alle Daten zur Steuerung gelangen von und zur Computeranwendung über die Zentrale. Die Zentrale übersetzt, verteilt und leitet die erhaltenen Daten an die entsprechenden Komponenten weiter.

4.1 Zentrale

In diesem Abschnitt wird der Aufbau und die notwendigen Komponenten für die Kommunikation und Datenübertragung beschrieben. Die Zentrale ist das Herzstück des gesamten Systems. Sie arbeitet mit verschiedenen Protokollen und leitet die Daten von der Fahrstraße als auch die Daten der Fahrzeuge an die Computeranwendung Rocrail© (siehe Kapitel 2.4) weiter. Die Zentrale arbeitet im Wesentlichen als Protokollübersetzer. Dabei werden keine Daten verändert, diese werden lediglich mit dem entsprechenden Paketformat an das Zielprotokoll weitergeleitet.

4.1.1 Kommunikation

Einleitend werden in diesem Abschnitt die drei verwendeten Protokolle und ihre Funktionen beschrieben. Jedes Protokoll besitzt speziell für seine Aufgabe angepasste Kommandos die bestimmte Funktionen auslösen. Dabei dient das P50x-Protokoll dient zur Kommunikation zwischen der Computeranwendung auf dem Computer mit der Zentrale. Das LocoNet™-Protokoll wird für die Datenverbindung zwischen der Zentrale und der Fahrstraße eingesetzt. Das RCMC-Funk-Protokoll realisiert die Datenübertragung zwischen der Zentrale und den Modelfahrzeugen auf der Fahrstraße. In der Abbildung 4.1 ist die Zentrale mit ihrer Anbindung der verschiedenen Kommunikationsprotokolle an Mikrocontroller dargestellt.

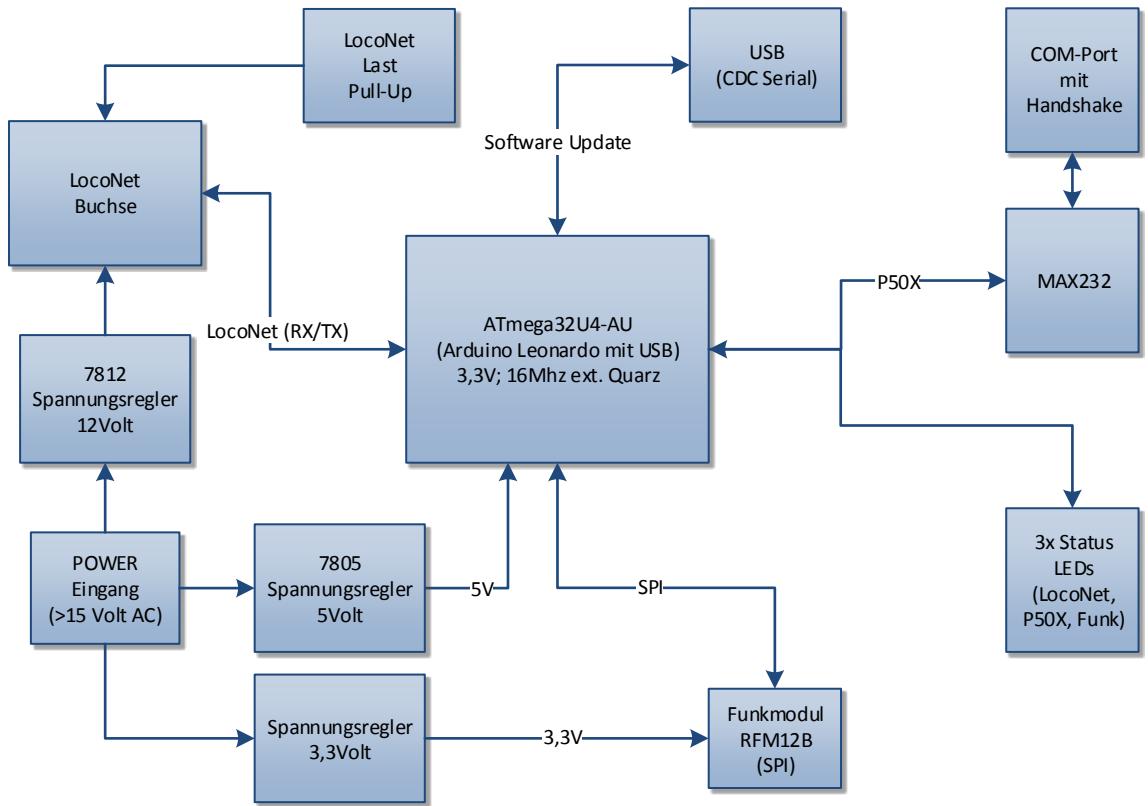


Abbildung 4.1: Blockbild des Funktionsaufbau der Zentrale.

P50X-Protokoll

Da es keine Arduino Bibliothek gibt, die das P50-Protokoll dekodiert, wurde eine eigene Bibliothek für diesen Zweck erstellt. Über das serielle P50X-Protokoll empfängt und sendet die Zentrale Daten zur Computeranwendung des Computer. Die Computeranwendung erhält Rückmeldezustände von den Ampeln, Fahrzeugpositionen aus den Streckenabschnitten, sowie Meldungen von Weichen in der Fahrstraße. Zusätzlich können die Akkuzustände der Fahrzeuge sowie deren Konfigurationsvariablen an den Computer übergeben werden. Die Zentrale empfängt Daten zur Schaltung der Weichen, sowie fahrzeugspezifische Programmier-, Fahr- und Funktionsbefehle. Alle Befehle die darüber hinaus unbekannt für die erstelle Arduino P50X-Bibliothek sind können nicht dekodiert werden. Das kann je nach Implementierung der Computeranwendung zu einem Abbruch der Kommunikation führen. Bei aktiver Kommunikation mit der Computeranwendung leuchtet als optische Rückmeldung die P50X-Status-LED kontinuierlich. Ist keine Kommunikation vorhanden, signalisiert die LED durch rhythmisches Blinken im 750 ms Takt, dass die Versorgungsspannung an der Zentrale anliegt.

LocoNet™

Das implementierte LocoNet™-Protokoll zur Kommunikation zwischen der Zentrale und der Fahrstraße, unterstützt nur zwei Arten von Befehlen. Der Empfang von Rückmeldeereignissen und das Senden neuer Weichenpositionierungen. Diese Befehle reichen um alle Komponenten der Fahrstraße an die Computeranwendung anzubinden. Eine weitere Status-LED auf der Platine der Zentrale zeigt für die Dauer von 500 ms eine aktive Datenübertragung an.

RCMC-Funk-Protokoll

Die Funkübertragung wird für einen reibungslosen und schnellen Datenaustausch zwischen der Zentrale und den Fahrzeugen verwendet. Dazu wurde ein speziell auf diese Bedürfnisse angepasstes Protokoll entwickelt. Eine Status-LED an der Zentrale zeigt für die Dauer von 500 ms eine aktive Datenübertragung auf dem RCMC-Funk-Protokoll an.

Funktionsweise des Funk-Protokolls Die von der Zentrale zu übertragenden Datenpakete sollen alle aktiven Fahrzeugdekoder auf der Fahrstraße möglichst schnell und ohne unnötige Verzögerung erreichen. Bei der Erstellung des Protokolls wurde deshalb speziell darauf geachtet, dass die Datenpakete eine variable Länge besitzen. Kürzere Datenpakete erreichen ihr Ziel schneller und sind bei der Übertragung weniger Störungsfällig durch fremde Funksignale. Um eine einfache Übersetzung zwischen dem P50X- und dem Funk-Protokoll zu ermöglichen, sind die Funkdatenpakete am P50X-Protokoll orientiert. Übertragen werden, von der Zentrale zu den Fahrzeugdekodern, Statusinformationen wie Lichtfunktionen, 28 Zusatzfunktionen, 128 Geschwindigkeitsstufen, Änderungen und Abfrage von Konfigurationsvariablen sowie Rückmeldeinformationen. Mit Hilfe der Rückmeldung können Werte der Konfigurationsvariablen ausgelesen und gesendete Programmierbefehle bestätigt werden. Auch Meldungen über spezielle Ereignisse im Fahrzeug, wie ein geringer Akkuladezustand werden durch das Funk-Protokoll zur Zentrale übermittelt.

RCMC-Funk-Datenpakete Ein RFM12B Funk-Datenpaket setzt sich aus fünf Teilen (siehe Abbildung 4.2) zusammen. Diese sind: Präambel, Synchronisation, Header, Daten-Byte und Prüf-Byte. Die Präambel besteht aus drei Byte, welche jeweils den Wert 0XAA enthalten. Sie wird zu Beginn gesendet und signalisiert dem RFM12B Empfänger das anschließend Daten übertragen werden. Danach folgt eine zwei Byte lange Synchronisation. Sie besteht aus einem Byte mit 0X2D und aus einem Netzwerk-ID-Byte. Der darauf folgende Nachrichten-Header besteht aus drei Byte. Sie enthalten jeweils ein Bit Control-Flag, eine sieben Bit Ziel-ID, ein Bit Acknowledge-Flag, eine sieben Bit Quell-ID und im dritten Byte die Länge des folgenden Datenbyte. Alle Datenpakete enthalten als letztes Byte die Prüfsumme.

Der Grundaufbau der RFM12B Datenpakete wird durch die verwendete JeeLib-Bibliothek für das RFM12-Funkmodul (siehe Abschnitt 2.2.2) erzeugt. Um die gewünschten Steuerungsinformationen für die Modellfahrzeuge zu übertragen, werden diese als

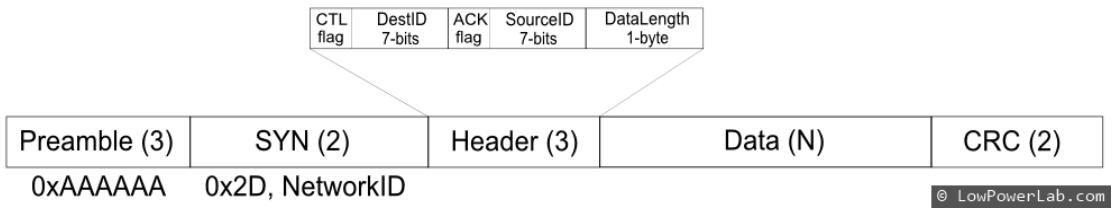


Abbildung 4.2: Struktur der Datenpakete des RFM12 Funkmoduls [23].

Daten-Bytes von der Bibliothek gesendet. Ein solches Steuerungsinformation besteht aus mindestens zwei Byte und ihr Aufbau ist im RCMC-Protokoll definiert. Eine Zusammenstellung aller RCMC-Datenpakete befindet sich im Anhang C.

Da mit der Ziel-ID und der Quell-ID des RFM12 Funkmoduls sind nur maximal 128 Geräte adressierbar. Deshalb wird diese nur als Kennung für den Modelfahrzeugdekker und die Zentrale verwendet. Die eigentliche Adressierung der Modelfahrzeuge erfolgt mit einer separaten Adresse (1...10, welche in den Datenbytes verpackt versendet wird. Dabei dekodieren die Fahrzeugdekkoder nur Daten, die von der ihm bekannten Zentrale an sie gesendet werden. Umgekehrt dekodiert die Zentrale nur Datenpakete, die von der konfigurierten Adresse für Fahrzeugdekkoder gesendet werden. Jedes Steuerungssystem darf deshalb nur eine Zentrale besitzen. Da die Funkmodule sich zusätzlich in maximal 250 Gruppen einteilen lassen, ist es möglich mehrere Systeme unabhängig voneinander im gleichen Raum auf verschiedenen Anlagen einzusetzen. Eine Kommunikation zwischen den Gruppen ist nicht ohne eine Umschaltung möglich.

Auf Broadcast-Befehle die von der Zentrale ohne Angabe einer Adresse an alle Fahrzeugdekkoder gesendeten reagieren alle Fahrzeuge. Die Datenpakete erhalten dabei als Dekoderadresse im Datenbyte eine Null. So können sie nicht mit dem Geschwindigkeits- und F0-Datenpaket verwechselt werden (siehe Anhang C), da Null eine ungültige Fahrzeugdekkoderadresse ist.

Statusänderungen werden über Funk mindestens zweimal wiederholt. Dadurch wird sichergestellt, dass die Daten vom Fahrzeugdekkoder empfangen wurden. Im späteren Ablauf werden die Datenpakete etwa alle 2 bis 10 Sekunden erneut zyklisch wiederholt. Somit wird ein inaktives Fahrzeug, welches erst nachträglich auf die Fahrstraße gesetzt wurde, bei Zuschaltung der Versorgungsspannung nach einer gewissen Zeit mit den zuletzt gesendeten Zustandsinformation versorgt.

Die bidirektionale Kommunikation zwischen der Zentrale und dem Fahrzeugdekkoder ermöglicht es dem Fahrzeug Statusmeldungen an die Zentrale zu senden. Diese müssen mit einem Acknowledge Leerdatenpaket, welches ein Byte 0X00 enthält, von der Zentrale bestätigt werden. Wird keine Bestätigung erhalten, erfolgt seitens des Fahrzeugdekkoders ein wiederholtes Senden mit mindestens zwei weiteren Versuchen. Diese Bestätigung ist nur für Programmierbefehle oder Meldungen die vom Fahrzeugdekkoder kommen notwendig. Eine Bestätigung für den Erhalt von Steuerungsinformationen die von der Zentrale und den Fahrzeugdekkoder gesendet werden ist nicht vorgesehen. Dies würde das System durch die notwendige Wartezeit verlangsamen und zusätzlich beansprucht die Bestätigung der Datenpakete durch das Fahrzeug den Fahrzeugakku unnötig.

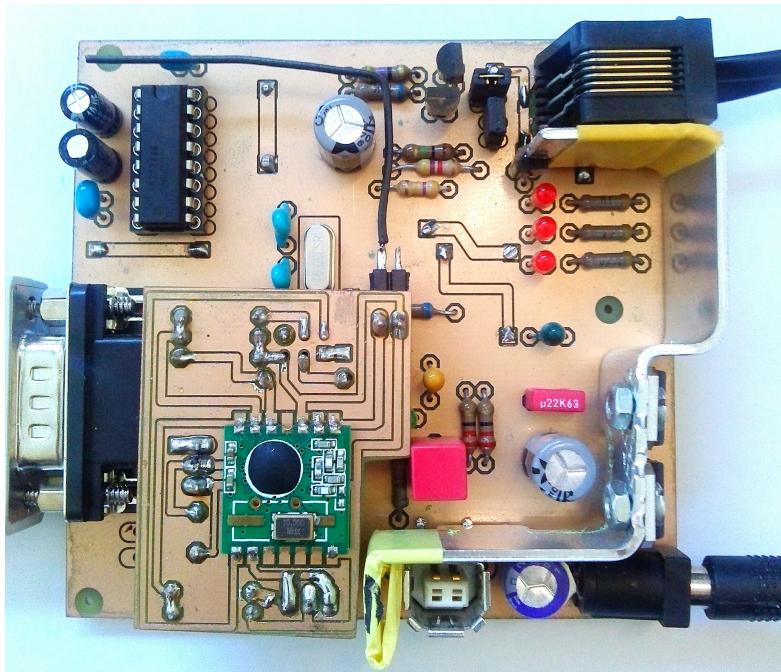


Abbildung 4.3: Draufsicht der prototypisch aufgebauten Zentrale.

4.1.2 Hardware

Die Zentrale benötigt für jede Kommunikationsschnittstelle eine angepasste Hardware-Schnittstelle. Die Basis bildet dabei der Mikrocontroller des Arduino Leonardo. Dieser wurde in SMD-Bauform auf einer angefertigten Leiterplatte aufgelötet. Der verwendete Atmega32U4 ist der gleiche Mikrocontroller der auch im Fahrzeugdekoder (siehe Abschnitt 4.2) eingesetzt wird. Allerdings arbeitet dieser in der Zentrale mit einer Taktgeschwindigkeit von 16 MHz. Abbildung 4.3 zeigt die Leiterplatte des Prototyps. Der Mikrocontroller befindet sich auf der Rückseite der Leiterplatte und ist auf der Abbildung 4.4 dargestellt.

In der Zentrale arbeitet der Atmega32U4 mit einer Versorgungsspannung von 5 Volt. Eine voll ausgebauten serielle Schnittstelle mit dem Bustreiber MAX232 realisiert den notwendigen Hardwarehandshake über CTS und RTS¹ für die Übertragung des P50X-Protokoll zum Computer.

Im LocoNet™-Protokoll sind die Verbinder und die Leitungsbelegung genau spezifiziert. Die Datenübertragung geht dabei über ein sechsadriges Flachbandkabel. Digitrax empfiehlt die Verwendung eines Telefonkabels mit RJ12-Steckverbinder zu verwenden [10]. Diese sind dabei kostengünstig und bieten eine hohe Kontaktzuverlässigkeit. Die sechs LocoNet™-Adern liefern sind auf den Pins 2 und 5 mit Masse belegt. Für die Datenkommunikation läuft über die Pins 3 und 4 ab. Beide Adern übertragen das Empfangs- und Sendesignal. Wenn keine Daten übertragen werden, sollte die Datenleitung auf 12 Volt geschaltet werden. Dies wird mit einem Pull-Up-Widerstand und einer Stromquelle, die 15 bis 20 mA liefert, erreicht. Für die Zuschaltung der Stromquelle in

¹Request to Send signalisiert eine Sendeanforderung.

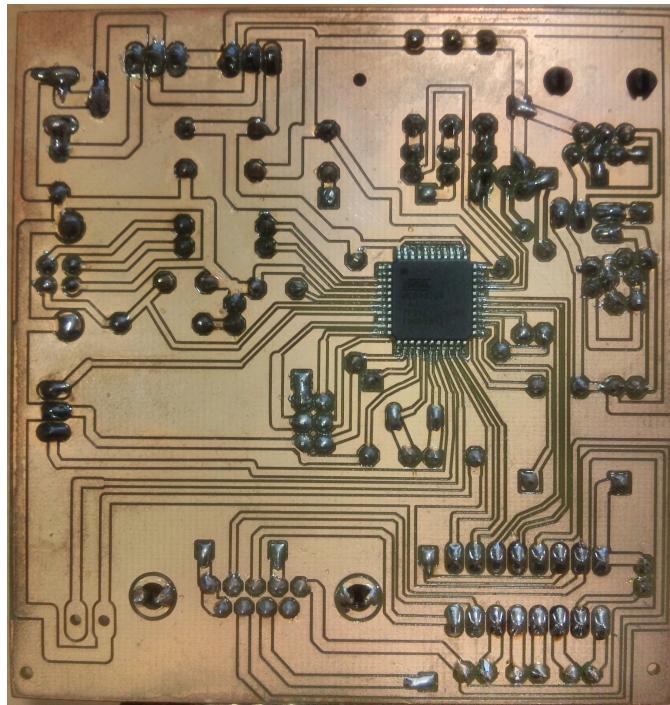


Abbildung 4.4: Platinenrückseite der prototypisch aufgebauten Zentrale.

der Zentrale muss dafür ein Jumper gesetzt werden, wenn kein anderer Knoten diese zur Verfügung stellt. An Pin 1 und 6 liegt das sogenannte Railsync-Signal. Railsync ist ein Low-Power-Signal und überträgt das DCC-Protokoll vom Modelleisenbahngleis [10]. Über diese Leitungen kann auch eine geringe Anzahl von LocoNetTM-Verbraucher seine Versorgungsspannung erhalten. Da die Zentrale kein DCC-Signal besitzt, kann mit einem Jumper die Railsync-Leitung auf 12 Volt Versorgungsspannung gelegt werden. Diese darf mit maximal 1 A belastet werden.

Um jederzeit einen anderen Bootloader in den fest verlöteten Atmega32U4 der Zentrale einspielen zu können, besitzt die Leiterplatte einen ISP²-Verbinder. Auf diesen wird die Zusatzplatine mit dem verbauten RFM12B Funkmodul aufgesteckt. Auf dieser Zusatzplatine befindet sich neben dem Funkmodul ein Pegelwandler und ein 3,3 Volt Festspannungsregler. Der Pegelwandler ist für den Betrieb des Funkmoduls am Atmega32U4 notwendig, da der Mikrocontroller mit einer Versorgungsspannung von 5 Volt arbeitet und das Funkmodul dagegen, darf nur mit maximal 3,8 Volt betrieben werden [16]. Die Beschaltung der verschiedenen Baugruppen ist im Schaltplan (siehe Anhang D.2) dargestellt. In dem Block unten links auf dem Schaltplan ist die Zusatzplatine mit dem Funkmodul dargestellt.

²In-System-Programmierung (ISP) ermöglicht das Programmieren eines Mikrocontroller direkt auf der Schaltung.

4.1.3 Software

Die Software für den Mikrocontroller Atmega32U4 der Zentrale ist in drei Bereiche untergliedert. Jeder Bereich enthält dabei die unterstützten Befehle des dort verwendeten Protokolls.

Dekodierung des LocoNetTM-Protokolls

Für die Implementierung des LocoNetTM-Protokolls, wurde die Bibliothek von Alex Shepherd genutzt [24]. Diese Bibliothek unterstützt leider nicht die notwendigen Funktionen für den Betrieb eines LocoNetTM-Master. Deshalb wurden nur die Befehle verwendet, die für das Stellen der Weichen in der Fahrstraße und für das Empfangen von Rückmeldungen notwendig sind. So ist zum Beispiel der Betrieb eines Handreglers, wie bei Modelleisenbahnen üblich, am LocoNetTM-Bus nicht möglich. Es gibt zwei wichtige Funktionen zur Steuerung der Kommunikation. Die *void notifySensor* ist eine Funktion die nach dem Eintreffen eines LocoNetTM-Rückmelddatenpaket von der Auswertefunktion *LocoNet.processSwitchSensorMessage* aufgerufen wird. Über diese wird der Status des Rückmenders abgefragt und an die P50X-Bibliothek übergeben. Mit der Funktion *LocoNet.requestSwitch* wird ein neuer Status an eine Weiche in der Fahrstraße übertragen. Diese Funktion wird aufgerufen, wenn über die P50X-Bibliothek ein Schaltbefehl für eine Weiche empfangen wird.

RCMC-Funkdatenübertragung

Zur Implementierung der Ansteuerung des RFM12B Funkmoduls über die SPI-Schnittstelle, wurde die Jeelib-Bibliothek genutzt [19]. Diese erleichtert die zu Beginn notwendige Initialisierung des Funkmoduls. Der Quellcode zur Datenübertragung enthält das Erzeugen und Senden der RCMC-Datenpakete. Jeder Fahrbefehl wird dabei über das P50X-Protokoll empfangen. Da zum Zeitpunkt des Empfangs eines solchen Befehles nicht sichergestellt werden kann ob das Funkmodul gerade beschäftigt ist, werden alle eingehenden Nachrichten in das RCMC-Protokoll umgewandelt und in einer Liste chronologisch nacheinander zwischengespeichert. Dabei wird jede Nachricht mit der vorhergehenden Nachricht verglichen und nur ungleiche Nachrichten in die Liste aufgenommen. Dadurch kann beim Senden Zeit eingespart werden. Gesendet wird immer, wenn neue Datenpakete in der Liste vorhanden sind. Jedes Datenpaket wird dabei dreimal im Abstand von zirka 30 ms an die Fahrzeugdekoder gesendet. Wenn keine neuen Befehle in der Liste sind, wird fortlaufend zirka alle 1,5 Sekunden nach dem letzten Befehls, der aktuelle Status jedes aktiven Fahrzeugs erneut einfach versendet. So wird sichergestellt, dass auch abgeschaltete Fahrzeuge beim Zuschalten ihren aktuellen Motor- und Funktionsstatus erhalten.

P50X-Datenverarbeitung

Zur Dekodierung der P50X-Protokolldaten, wurde eine Arduino Bibliothek erstellt. Diese bietet alle notwendigen Funktionen zum Betrieb der Fahrzeugsteuerung und dessen

Rückmeldungen. Über die serielle Schnittstelle des Mikrocontroller werden die Daten Byte-Weise eingelesen. Da ein Befehl aus mehreren Byte besteht, wird jedes eingelesene Byte zuerst in den Speicher geschrieben. In der Bibliothek wird nach jedem gesendeten Byte geprüft, ob ein vollständiger Befehl empfangen wurde. Ist dies nicht der Fall, wird auf das Eintreffen des nächsten Byte gewartet. Damit während dieser Wartezeit die anderen Aufgaben und Protokolle abgefragt werden können, wird nach jedem eingelesenen Byte die P50X-Bibliothek verlassen und zum Hauptprogramm zurückgekehrt. Wurde ein P50X-Befehl korrekt empfangen, werden die Daten dekodiert und der passenden Funktion übergeben. Über diese werden die Daten dann an ihr Zielprotokoll gesendet. Jedes Steuerungsdatenpaket für Fahrgeschwindigkeit und Funktionsstatus wird zusätzlich in einer Liste erfasst. Dies macht es möglich den letzten Zustand eines jeden Fahrzeugs abzurufen und als Wiederholungsdatenpaket erneut zu versenden. Wenn ein Rückmeldeereignis eintrifft, wird dieses in der internen Datenbank für Rückmelderzustände der P50X-Bibliothek gespeichert und außerdem das Änderungsflag gesetzt. Über Ereignisabfragen der Computeranwendung wird vom Protokoll erkannt, dass dieses Flag aktiv ist und neue Daten zum Abruf bereit liegen. Nach Aufforderung werden die Daten in Form einer Liste an die Computeranwendung gesendet. Bei schnell hintereinander auftretenden Ereignissen, wird dabei in der Datenbank der P50X-Bibliothek beim Eintragen eine Vorsortierung durchgeführt. So wird sichergestellt, dass jedes Ereignis auch an die Computeranwendung weitergemeldet wird.

Zeitliche Optimierung des Programmablauf

Damit keine ankommenden Daten während der Verarbeitung eines Protokolls verloren gehen, muss die Software ohne Zeitverzögerungen, ständig die verschiedenen Protokollschnittstellen nach Statusänderungen abfragen. Auch innerhalb der Verarbeitung eines Protokolls dürfen die Funktionen keine Pausen enthalten. Nur so ist es möglich, ein exaktes und kontinuierliches Fahrverhalten zu realisieren. Bereits geringe Verzögerungen bei der Datenübertragung oder gar ein auslassen eines Datenpaket, führen zu einem unerwünschten Verhalten der Modelfahrzeuge. Unter Umständen fahren diese dann aufeinander auf oder halten zu spät in der Kreuzung oder dem Parkplatz an. Auf der Abbildung 4.5 ist der Programmablauf der Zentrale in einem Diagramm abgebildet.

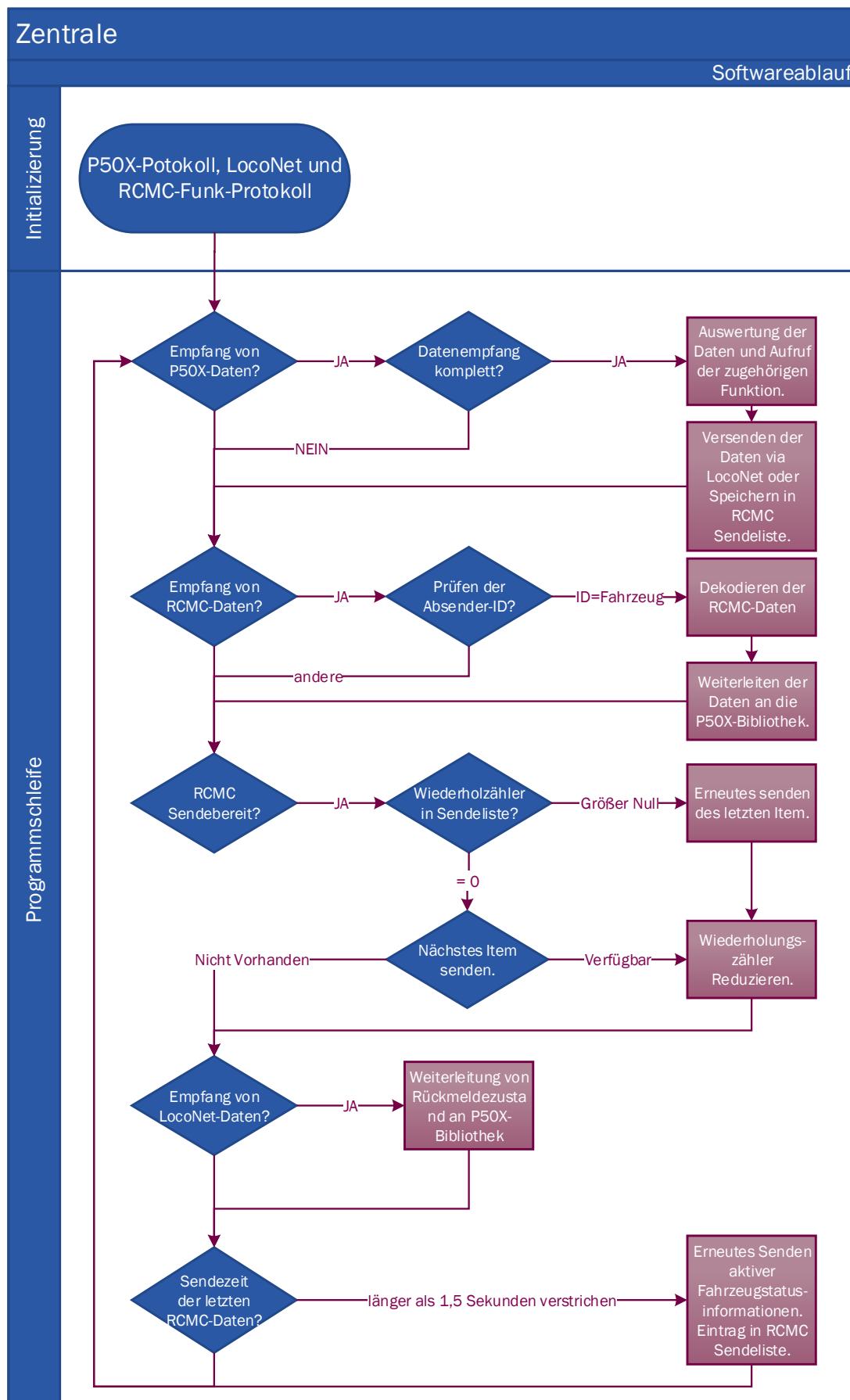


Abbildung 4.5: Softwareablauf der Zentrale.

4.2 Fahrzeugdekoder

Der Fahrzeugdekoder bildet die Schnittstelle zwischen der Zentrale und den verschiedenen Komponenten im Fahrzeug. Er sorgt außerdem dafür, dass jedes Fahrzeug eine eigene Adresse erhält. Durch die Einstellung von Konfigurationsvariablen lässt sich jeder Fahrzeugdekoder individuell anpassen.

4.2.1 Funktionsprinzip

Mit dem Fahrzeugdekoder sollten mindestens alle Grundfunktionen der Fahrzeuge des Faller Car System realisiert werden. Im Wesentlichen sind das die Fahrstufensteuerung des Motor und die Haltefunktion. Um ein realistisches Fahrzeug zu erhalten, wurden auch die Beleuchtungs- und Sound-Steuerung realisiert werden. Bei dem Prototyp des Fahrzeugdekoder kann das Abblendlicht mit integriertem Fernlicht, Rücklicht mit integriertem Bremslicht und die Blinkern als Grundfunktionen angesteuert werden. Für weitere Zusatzfunktionen stehen am Fahrzeugdekoder bis zu zehn, variabel programmierbare, Ausgänge zur Verfügung. Jeder Ausgang kann über Pulsweitenmodulation (PWM) gesteuert werden. Das macht es möglich, die Ausgangshelligkeiten für jeden Ausgang beliebig zu variieren. Zusätzlich ist die Zeit für das Ein- und Ausdimmen pro Ausgang programmierbar. So können die LEDs zum Beispiel, wie bei Glühfäden von Glühbirnen üblich, langsam an und aus glimmen. Bis auf die fest reservierte Grundfunktion F0 für das Abblendlicht und F3 für die Lichtautomatik lassen sich alle anderen Ausgänge auf eine beliebige Funktionen F1 bis F28 programmieren. Für die zehn Zusatzausgänge können sogar mehrere Funktionen mit und ohne Abhängigkeiten konfiguriert werden. Deshalb ist es möglich, eine zweite Helligkeit für diese Ausgänge, über eine separate Funktion zu schalten. Je nach Art der angeschlossenen Hardware, kann der Ausgangspegel jedes Ausgangs invertiert angesteuert werden. Besitzt ein Fahrzeug einen Helligkeitssensor, kann über diesen, je nach Umgebungslicht das Abblendlicht bei aktiver Funktion F3 automatisch eingeschaltet werden. So kann zum Beispiel bei einer Tunneleinfahrt, durch die Erkennung des Helligkeitsunterschied das Licht automatisch eingeschaltet und beim Verlassen nach kurzer Zeit wieder abgeschaltet werden.

Spannungversorgung

In der Abbildung 4.6 sind die einzelnen Komponenten des Fahrzeugdekoders dargestellt. Der verwendete Mikrocontroller Atmega32U4-AU besitzt ein USB-Interface, welches zum Auslesen und Ändern von Konfigurationsvariablen als auch für ein Softwareupdate des Fahrzeugdekoders verwendet werden kann. Versorgt wird der Fahrzeugdekoder gegenüber dem Infrarotübertragungssystem jetzt nicht mehr mit einem herkömmlichen Nickel-Metallhydrid-Akkumulator (NiMH), sondern über eine Zelle eines Lithium-Polymer-Akkus (LiPo-Akku) mit einer Zellspannung von 3,7 Volt. Diese Akkus benötigen weniger Platz, haben ein geringeres Gewicht und besitzen eine höhere Leistungsfähigkeit. Aufgrund des internen gelartigen Aufbaus sind LiPo-Akkus in vielerlei Bauformen erhältlich, was den Einbau in die kleinen Modellfahrzeuge sehr erleichtert [20]. Einen weiteren Vorteil bietet die hohe Zellenspannung von 3,7 Volt,

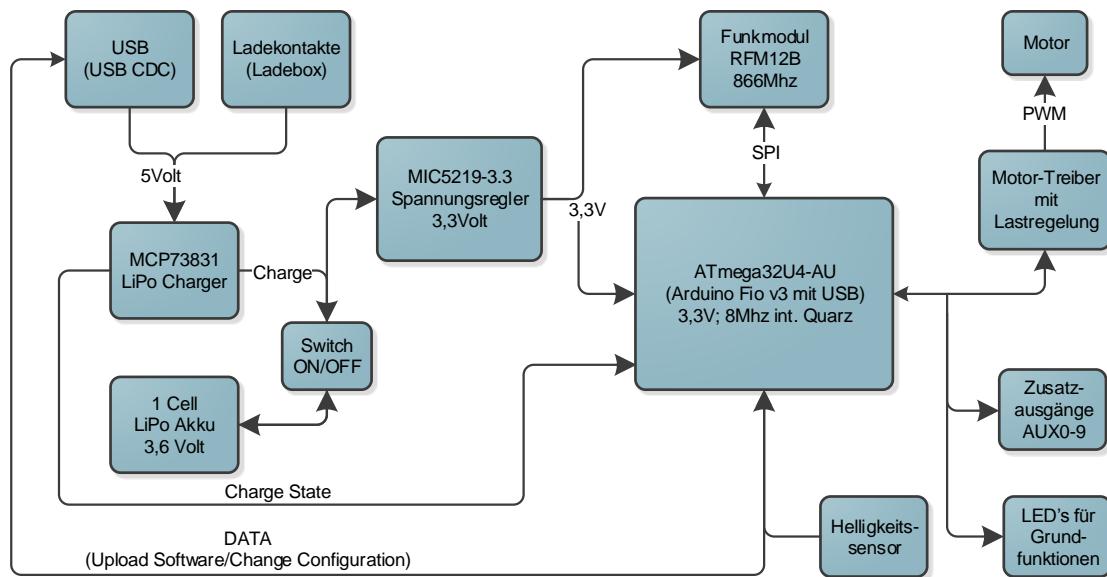


Abbildung 4.6: Komponenten des Fahrzeugdekoders.

welche ideal ist zur Nutzung von Mikrocontrollern. Da für LiPo-Akkus eine spezielle Ladefunktion benötigt wird, wurde ein LiPo-Ladeschaltkreis (siehe Abschnitt 4.2.5) im Fahrzeugdekomodul integriert. Dieser LiPo-Laderegler überwacht und steuert den gesamten Ladeprozess und erspart damit ein externes LiPo-Ladegerät. Die Ladespannung wird aus dem USB-Interface des Fahrzeugdekomoduls erzeugt.

Damit das Fahrzeug auch bei unterschiedlicher Belastung, zum Beispiel durch eine Bergfahrt nicht zu stark von der eingestellten Geschwindigkeit abweicht, kommt für die Motoransteuerung eine Lastregelung zum Einsatz. So ist es möglich eine konstante Fahrgeschwindigkeit, unabhängig von einer abnehmender Akkuspannung oder an Fahrstraßenanstiegen oder Fahrstraßenengeläufen zu realisieren. Die Energie eines Fahrzeug Akku ist nicht unbegrenzt. Ein Fahrzeug mit einer Akkukapazität von 110 mAh kann für zirka eine Stunde betrieben werden. Zur wirtschaftlichen Verwaltung der Akkutypen, besitzt die Fahrzeugdekomodul-Software ein Energiemanagement. Das ermöglicht bei Inaktivität und bei Nutzungspausen außerhalb der Fahrstraße eine automatische Abschaltung des Funkmoduls und aller Fahrzeugdekomodulausgänge des Mikrocontrollers.

4.2.2 Hardware

Der Baugrößenmaßstab, der hier prototypisch eingesetzten Modelfahrzeuge beträgt 1:120. Diese Miniaturisierung verlangt eine größtmäßig passende Platine für die Fahrzeuge. Diese sollte dabei den genannten Anforderungen entsprechen und trotzdem mit einer möglichst geringen Baugröße auskommen. Um die Platzverhältnisse ideal auszunutzen, wurde die zweiseitige Platine so konzipiert, dass sich auf einer Seite der Mikrocontroller und auf der anderen das Funkmodul befindet. So lassen sich die Abmessungen trotz der hohen Zahl von 24 Zusatzbauelementen in der Länge auf 26 mm und Brei-

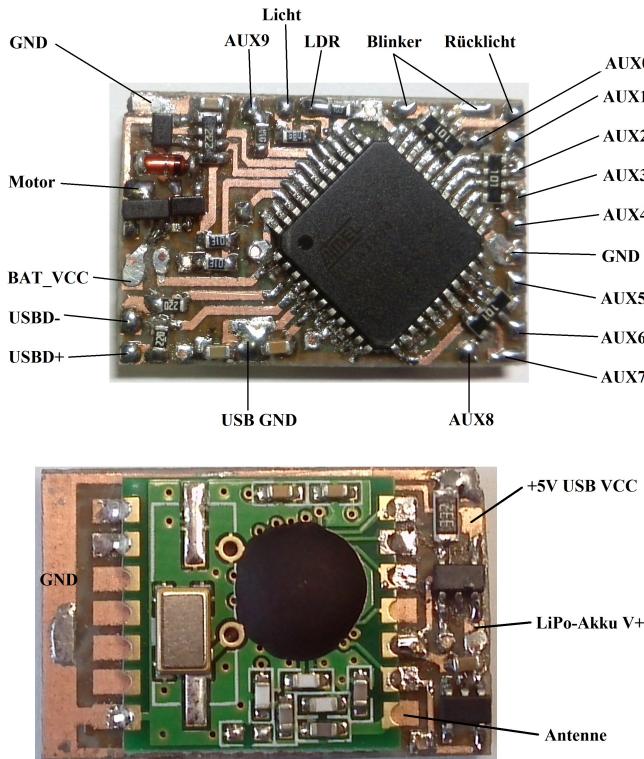


Abbildung 4.7: Fahrzeugdekker mit Atmega32U4 und RFM12B.

te auf 16,5 mm reduzieren. Der Nachteil dieses Aufbaus zeigt sich durch die Vielzahl von 15 notwendigen Durchkontaktierungen um die Daten- und Versorgungsleitungen von der einen auf die andere Leiterplattenseite zu führen. Abbildung 4.7 zeigt den prototypischen Aufbau eines fertig bestückten Fahrzeugdekoders. Auf der Rückseite der Leiterplatte befindet sich neben dem RFM12B-Funkmodul die Spannungsregelung mit LiPo-Laderegler und einem 3,3 Volt Festspannungsregler. Die Vorderseite bietet neben dem 44-Pin-Mikrocontroller Atmega32U4 Anschlüsse für die Funktionsausgänge und die Motorregelung. Alle Komponenten wurden SMD-Bauform verwendet.

Der eingesetzte Mikrocontroller Atmega32U4 ist in SMD-Bauform 10 mal 10 Millimeter groß und um 45 Grad gedreht auf der Fahrzeugdekker-Platine aufgelötet [3]. Die Drehung bietet einen Zugang zu allen Pins des Mikrocontrollers und macht somit die geringe Breite der Leiterplatte erst möglich. Der Hardwareaufbau des Fahrzeugdekoder ähnelt der des Arduino Leonardo Entwicklungs-Boards. Um weiteren Platz einzusparen, wurde auf einen externen Quarz für den Atmega32U4 verzichtet. Aus diesem Grund wird der Mikrocontroller über den internen Quarz mit 8 MHz getaktet. Für den Einsatz des internen Quarzes sind ein anderer Arduino-Bootloader und angepasste AVR-Fuses beim erstmaligen Programmieren des Mikrocontrollers erforderlich.

Spannungsversorgung des Fahrzeugdekoder

Mit der Verwendung eines LiPo-Akku entstehen zusätzliche Probleme. So steigt die Zellenspannung eines voll geladenen Akkus auf bis zu 4,3 Volt an. Da das zu hoch für die Versorgung des Funkmoduls ist, wird ein Festspannungsregler für 3,3 Volt, auf dem Fahrzeugdekoder verbaut. Er versorgt neben dem Funkmodul auch den Mikrocontroller und dessen Ausgänge. Dadurch bleibt die Helligkeit, der an den Ausgängen angeschlossenen LEDs, bei absinkender Akkuspannung unverändert. Der Motor wird, aufgrund des höheren Stromverbrauchs direkt vom LiPo-Akku gespeist.

Auf dem Fahrzeugdekoder befindet sich eine vollwertige USB-2.0-Schnittstelle. Diese stellt über das USB Communications Device Class (USB CDC) eine virtuelle serielle Schnittstelle zur Verfügung. Da eine USB-Buchse durch die erforderliche Bautiefe zuviel Platz in einem Modellfahrzeug benötigten würde, wurde ein Adapter für die USB-Verbindung entworfen. Dieser USB-Adapter (siehe Abbildung 4.8) wird in eine kleine vier polige Buchse, die sich an der Unterseite des Fahrzeugs befindet, eingesteckt. Eine kleine Sicherung auf dem Mini-USB-Adapter schützt das System vor einem Kurzschluss durch Verpolung.



Abbildung 4.8: Mini USB Adapter für den Fahrzeugdekoder.

Aufbau der Lastregelung

Für die Realisierung einer Lastregelung gibt es verschiedene Möglichkeiten. Die hier eingesetzte Lastregelung basiert auf der Überwachung der Drehzahl der Motorachse. Die Ansteuerung des Fahrzeugmotors über ein PWM-Signal, bietet die Möglichkeit in den Pausen des PWM-Signal (Austastlücken) die Generatorenspannung des leerlaufenden Fahrzeugmotors zu messen. Da diese Spannung direkt proportional mit der Drehzahl des Motors ist, lässt sich bei Veränderung dieser Spannung das PWM-Signal des

Fahrzeugmotor entsprechend korrigieren. Bei der Messung der Generatorspannung ergab sich folgendes Problem: Da der Mikrocontroller im Fahrzeugdekker nur positive Spannungen am Eingang messen kann, musste der Motor über die Plusleitung mit dem PWM-Signal versorgt werden. Aufgrund der hohen Strombelastung des Fahrzeugmotors, kam zum Schalten nur ein P-Kanal-Mosfet infrage. Dieser benötigt, zur vollen Aussteuerung eine negative Gate-Spannung. Zur Erzeugung dieser negativen Gate-Spannung wurden zwei NPN-Transistoren eingesetzt. Über einen Spannungsteiler, direkt am Motoranschluss wird Softwareseitig abgestimmt zum PWM-Signal des Motors, die Generatorspannung in der Austastlücke gemessen.

Ein weiterer Spannungsteiler ist direkt am LiPo-Akku Eingang verbaut. Über diesen ist es möglich die exakte Akkuspannung mit dem Mikrocontroller zu messen. Außerdem befinden sich auf der Leiterplatte an jedem Ausgang bereits die notwendigen Vorwiderstände für den Anschluss von LEDs. Es sind deshalb keine weiteren externen Bauelemente für den Betrieb erforderlich.

4.2.3 Software

Ein Kriterium beim Entwurf der Software für den Fahrzeugdekker, war die Zielsetzung, diese so universell wie möglich zu gestalten. Das heißt sie sollte unabhängig vom verwendeten Fahrzeugtyp universell einsetzbar sein. Da es aber teilweise größere Unterschiede zwischen den Fahrzeugen gibt, wurde durch die Nutzung von Konfigurationsvariablen³ (siehe Anhang E), eine Möglichkeit geschaffen, den Fahrzeugdekker mit Hilfe dieser variablen, speziell auf den jeweiligen Fahrzeugtyp abzustimmen. Über die Konfigurationsvariablen ist es möglich, zum Beispiel im laufenden Betrieb Ausgangs- und andere Dekodereigenschaften zu verändern. Um der Software ein solches Verhalten zu vermitteln, musste ein variabler Quellcode geschrieben werden. Alle Konfigurationen des Fahrzeugdekoders werden im EEPROM des Mikrocontrollers gespeichert. Der EEPROM ist ein nicht flüchtiger Speicher. Er behält seine Daten auch ohne anliegende Versorgungsspannung.

Um die verschiedenen Zeitvorgaben, Abfragen und Hardwaresteuerungen in Software zu realisieren, erfordert es die Software gut zu strukturieren. Der zugehörige Quellcode mit zirka 1600 Zeilen (Lines of Code) ist wie nachfolgend aufgezeigt, gegliedert. Die verschiedenen unten dargelegten Phasen werden nacheinander durchlaufen:

1. **Initialisierung** der Software. Hier werden alle Einstellungen, die während des Betriebs notwendig sind, vorgenommen.
 - **Setzen der Referenzspannung:** Zuerst wird die Referenzspannung eingestellt. Die Software benötigt zur Bestimmung der Akkuspannung eine konstante Referenzspannung von 2,56 Volt, die der Mikrocontroller intern bereitstellt.
 - **PWM Timer Konfiguration:** Der dritte 16 Bit Timer (Timer3) des Mikrocontrollers wird zur Erzeugung des PWM-Signal für die Motoransteuerung

³Configuration Variables (CV)

konfiguriert. Der Vorteiler (Prescaler) wird dazu auf den Faktor eins gesetzt. Das PWM-Signal wird von der Überlauf-Timer-Funktion (Overflow Interrupt) gesteuert.

- **Einlesen der Registerwerte:** Hier werden Konfigurationsregisterwerte in die vorhandenen globalen Variablen geladen. Die Werte sind dafür im EEPROM hinterlegt. Sie können über eine Programmierung im PoM⁴-Mode oder über die serielle Schnittstelle des Fahrzeugdekoders verändert werden.
- **Port-Initialisierung:** Alle aktiven Pins des Mikrocontrollers werden dabei in den jeweiligen Arbeitsmodus gesetzt. Auf diese Weise wird festgelegt, ob ein Pin als Ein- oder Ausgang verwendet wird.
- **Funk-Initialisierung:** Hier wird das Funkmodul für die Datenübertragung konfiguriert. Dafür werden die Knoten- und Gruppen-ID, sowie die verwendete Funkfrequenz von 868 MHz gesetzt. Die Datenübertragungsgeschwindigkeit wird auf 57,471 kbps (0XC605⁵) und die Akkuwarnung auf 3,3 Volt (0XC0EB⁵) konfiguriert.
- **Sleep-Einstellungen:** In dieser Phase werden der Watchdog-Timer und der Sleep-Modus des Mikrocontrollers konfiguriert. Dabei erhält der Watchdog-Timer eine größtmögliche Unterbrechungszeit von acht Sekunden. Dieser Timer ist dafür verantwortlich den Controller aus dem Sleep-Modus aufzuwecken. Dazu wird regelmäßig geprüft ob Funkdaten über das Funkmodul empfangen werden.

2. Programmschleife:

Hier erfolgt die Betriebssteuerung aller angeschlossenen Komponenten.

- **Setzen der Ausgänge:** In dieser Funktion, welche zirka alle 10 ms durchlaufen wird, werden alle Ausgänge des Mikrocontrollers je nach deren Programmierung gesetzt. Zusätzlich wird bei jedem Durchlauf die Akkuspannung gemessen. Mit Hilfe eines Zählers, der bei jeder Ausführung erhöht wird, werden Wartezeiten und Intervalle aller zeitabhängigen Funktionen und Ausgänge gesteuert. Diese Abläufe sind im Abschnitt 4.2.3 genauer erklärt.
- **Serielle Kommunikation:** Diese Funktion wird alle 10 ms aufgerufen und prüft, ob die serielle Schnittstelle aktiv ist. Über serielle Befehle können so zum Beispiel der aktuelle Helligkeitswert, die Akkuspannung oder die Adresse des Fahrzeugdekoders ausgelesen werden. Darüber hinaus ist auch ein Schreiben und Lesen der Konfigurationsvariablen möglich.
- **Funkdekodierung:** Dekodierung der empfangenden Funkdatenpakete und Setzen der zugehörigen Variablen. Falls Rückmelddaten zur Übertragung zur Zentrale bereit sind, werden diese gesendet und auf eine Empfangsbestätigung gewartet. Erfolgt keine Bestätigung innerhalb von 60 ms (siehe

⁴Als Programming on the Main (PoM) wird bei der Modelleisenbahn der Programmievorgang auf dem Hauptgleis bezeichnet. Dabei werden die neuen Registerwerte nur an die festgelegte Dekoderadresse übertragen.

⁵Jeelab RFM12B Konfigurationstool: <http://tools.jeelabs.org/rfm12b.html>

CV#15), werden die Daten bis zu zweimal (siehe CV#14) wiederholt gesendet.

3. **ISR⁶ Zeitgeber:** Der Mikrocontroller Atmega32U4 besitzt vier Hardware-Timer. Timer0 wird für die Verzögerungen im Programmablauf verwendet. Timer3 erzeugt das PWM-Signal für die Motorsteuerung. Über den Hochgeschwindigkeits-Timer4, werden alle weiteren PWM Ausgänge angesteuert.

Dekodierung von Funksignalen

Empfangen werden alle Funksignale, egal für welches Fahrzeug diese bestimmt sind. Zuerst wird geprüft, ob die Prüfsumme des Datenpaketes korrekt ist. Danach werden die Sende-ID und Fahrzeugadresse überprüft. Bei jedem empfangenen Funkdatenpaket, dass eine korrekte Prüfsumme hat, wird die Zeit des Eintreffens gespeichert. Werden für eine bestimmte Zeitspanne (siehe CV#63) keine Funkdatenpakete registriert wird ein aktiver Motor, um eine Kollision mit einem anderen Fahrzeug oder ein unkontrolliertes Fahren zu verhindern, automatisch abgeschaltet. Ein bewegen der Fahrzeuge ist nur möglich, wenn durch die Computeranwendung die Datenübertragung aktiviert wurde. Beim Abschalten der Datenübertragung, kann das System einen kritischen Zustand einnehmen. Daher wird über diesen Befehl ein Notstop an alle Fahrzeuge ausgelöst.

Zur Dekodierung wird ein Datenpaket anhand seines Kommando und der zugehörigen Datenbytelänge gefiltert. Über das Kommando wird bestimmt, welche Art von Befehl gesendet wurde. Bei Geschwindigkeits- und Funktionsdatenpaketen, werden die übertragenden Byte in lokale Variablen übernommen. Dadurch wird sichergestellt, dass immer die zuletzt gesendeten Zustände für die Erzeugung der Ausgangssignale im Fahrzeugdekker zur Verfügung stehen. Wird ein Programmierdatenpaket erhalten, werden die entsprechenden Daten aus dem EEPROM-Speicher gelesen und an die Zentrale zurückgemeldet. Für die Dekodierung des Kommando im Datenpaket wurde eine eigene Funktion erstellt. Diese ermittelt die zwei Bit für das Kommando aus der gesendeten Dekoderadresse. Ein Befehl wird nur durch die Kodierung seines Kommando und die passende Datenbytelänge dekodiert. Mit der Prüfsumme für jedes RFM12-Datenpaket wird sichergestellt, dass keine verfälschten Daten dekodiert werden. Diese könnten dann zu einem unerwünschten Verhalten führen können.

Ansteuerung der Hardware

Die Ansteuerung der Hardware erfolgt entsprechend den am Mikrocontroller angeschlossenen Komponenten. Je nach Konfiguration im EEPROM (siehe Anhang E) verändert sich das Verhalten der Ausgangsansteuerung. Für die Erzeugung von Blinksequenzen der Zusatzausgänge wurde eine eigene Funktion im Quellcode implementiert. Die Zusatzausgänge werden in einer Liste verwaltet und ihr Zustand wird in einer Schleife alle zirka 10 ms Abgefragt und Aktualisiert. Da nur die Ausgänge für die Lichtfunktion (F0) und Lichtautomatik (F3) einer festen Funktionen zugeordnet sind, gibt es eine weitere Funktion die anhand der Konfiguration im EEPROM den Zustandsstatus der gewählten

⁶Interrupt Service Routine (ISR) ist eine Unterbrechungsroutine die durch ein Ereignis ausgelöst wird.

Funktion (F1 bis F28) des Ausgangs zurückgibt. Für die Hardwareansteuerung ist die Software wieder in mehrere, periodisch abzuarbeitende Bereiche unterteilt:

1. Funkdaten Prüfung:

- Dekodieren der übertragenen Funkdaten und ermitteln der Zustände. Falls ein Notstop Datenpaket empfangen wurde, wird ein laufender Motor abgeschaltet und die Warnblinkanlage aktiviert.
- Ermitteln, ob regelmäßig Funkdaten empfangen werden. Wenn eine gewisse Zeit (siehe CV#63) keine Daten über das Funkmodul erkannt wurden, wird der Mikrocontroller automatisch in den Sleep-Modus versetzt.

2. Basisfunktion Ansteuerung:

- Der Blinker hat mehrere Funktionen. Falls der Akkuwert unterhalb des Schwellwert von CV#29 fällt wird das Warnblinklicht eingeschaltet. Dieses überschreibt die normale Funktion des Blinker, welche über zwei frei konfigurierbare Funktionen CV#32 und CV#33 gesetzt werden kann. Bei einem gleichzeitigen einschalten beider Funktionen werden die Ausgänge synchronisiert geschaltet (Warnblinker). Um eine schnelle Reaktion des Blinkers beim Einschalten zu erhalten, wird der Intervallzähler bei Inaktivität dauerhaft zurückgesetzt. Dadurch wird beim Aktivieren einer der beiden Funktionen der Blinker sofort ohne Verzögerung ausgelöst.
- Für die Ansteuerung des Abblendlichts ist die Funktion F0 vorgesehen. Das Abblendlicht kann in Abhängigkeit des Helligkeitssensors aktiviert werden. Dazu lässt sich der Helligkeitssensor mittels der Funktion (F3) einschalten. Wenn dieser eingeschaltet ist, wird anhand von programmierbaren Schwellwerten (CV#52 ein und CV#54 aus) und Verzögerungszeiten (CV#53 ein und CV#55 aus), dass Licht automatisch ein- und ausgeschaltet.
- Für die Motoransteuerung können wie bei der Modelleisenbahn Verzögerungen für das Anfahren und Bremsen festgelegt werden. Über die Lastregulierung, welche im Timer-Interrupt berechnet wird, kann die gewünschte Motordrehzahl immer konstant gehalten werden. Das macht sich vor allem bei Langsamfahrten bemerkbar. Zusätzlich wird über die Motordrehzahländerung und die von der Digitalzentrale gesendete Fahrstufe das Ein- und Ausschalten des Bremslichtes berechnet.
- Die Zustände der Lichtausgänge mit den oben ermittelten Daten aus Punkt 2 werden erst zum nach der Motoransteuerung gesetzt. Es wird außerdem geprüft ob das Fahrzeug sich in der Ladestation befindet. Dann zeigen die Rücklichter über langsames Blinken einen aktiven Ladevorgang an. Ein schnelles Blinken signalisiert das der Akku voll ist. Um Hardware einzusparen wird das Rücklicht über ein gedimmtes Bremslicht realisiert.

3. Programmierbare Ausgänge:

- Für zehn Ausgänge ist deren Funktion nicht definiert. Sie können je nach Verwendung frei programmiert werden. Über die CV-Programmierung des Fahrzeugdekoders, kann das Ausgangsverhalten dieser Ausgänge bestimmt werden. Zusätzlich stehen jedem Ausgang zehn Pausezeiten für frei definierbare Blinkfrequenzen zur Verfügung.
- Alle Zusatzausgänge können außerdem auf verschiedene Funktionen (F1 bis F28) reagieren (siehe Konfigurationsvariablen im Anhang E.1).

Software Lastregelung des Fahrzeugdekoders

Um den Motor dynamisch je nach Belastung zu steuern, ist eine spezielle Hardware notwendig. Es wird dabei während der Fahrt die Drehzahl der Motorachse bestimmt. In Abhängigkeit der eingestellten Geschwindigkeit wird während der Puls-Weiten-Modulation (PWM) die aktuelle Motordrehzahl mit der Solldrehzahl verglichen. Durch kontinuierliches Anpassen der Pulslängen des Motorausgangs wird immer eine nahezu konstante Fahrgeschwindigkeit erreicht. Für die PWM-Steuerung des Motors wird der Timer3 des Atmega32U4 verwendet [25].

Angesteuert wird der Motor über mit 127 Fahrstufen von der Computeranwendung. Jede Fahrstufe wird vom Fahrzeugdekkoder auf die zugehörige Drehzahl der Motorachse umgesetzt. Um Regelspielraum für die Ansteuerung des Motors zu schaffen, wurde ein 16-Bit-Timer verwendet. Da die Auflösung von 16 Bit viel zu genau ist für die Motorregelung und zusätzlich die Reaktion zu stark verzögern würde, arbeitet die Lastregelung intern mit einer Auflösung von maximal 10 Bit. Je nach Konfiguration der minimalen und maximalen Fahrgeschwindigkeit. So verschiebt sich bei Belastung oder absinkender Akkuspannung der Regelbereich von 127 Fahrstufen innerhalb der 1024 möglichen Stufen dynamisch.

Die Berechnung der Motordrehzahl erfolgt über eine analog Messung der Generatorenspannung in den PWM-Pausen. Die Messung erfolgt dabei direkt im der Timer-Interrupt-Routine. Damit das Fahrzeug nicht ruckartig fährt, wird ein Mittelwert aus mindestens drei Messwerten errechnet. Erst danach wird das PWM-Signal neu berechnet und angepasst.

Energieeffizienz

Um den Akku nicht unnötig zu beladen, kann sich der Dekoder selbstständig in einen Ruhemodus versetzen. Dies geschieht bei einer inaktiven Datenkommunikation. Der Fahrzeugdekkoder überwacht dazu dauerhaft die Zeit zwischen dem Eintreffen einzelner Datenpakete. Im Leerlaufbetrieb ohne aktive Ausgänge und bei abgeschalteten Motor, hat der Fahrzeugdekkoder eine Stromaufnahme von zirka 21,8 mA. Wenn die Zentrale keine Daten über Funk sendet, beginnt nach der eingestellten Leerlaufzeit⁷ die Abschaltung aller Ausgänge. In dieser ersten Ruhephase bleibt das Funkmodul aktiv und kann den Dekoder über die Unterbrechungsroutine sofort aufwecken. Sofort nach dem Empfang von Daten wird der Dekoder aktiv und versetzt alle Ausgänge in den letzten Zustand

⁷einstellbar über CV#64

zurück. In dieser Phase verbraucht der Dekoder 90% weniger Energie als im Normalbetrieb. Der Strom von noch etwa 12,1 mA wird größtenteils vom Funkmodul verbraucht. Werden weiterhin keine Daten erfasst, wird auch das Funkmodul nach Ablauf der eingestellten Zeit⁸ in den Ruhemodus versetzt. Um ein Aufwecken des Mikrocontrollers durch ankommende Funkdaten zu ermöglichen, läuft auf dem Controller ein Watchdog-Timer. Der Watchdog-Timer ist ein spezieller Hardware-Zeitgeber. Durch ihn wird eine Unterbrechung nach maximal acht Sekunden ausgelöst. Der Mikrocontroller wird dadurch aufgeweckt und bleibt für eine Zeit von etwa 40 ms aktiv. In dieser Zeit wird auch das Funkmodul aktiviert und geprüft, ob dieses Daten empfängt. So dauert ein Wiedereinschalten des Dekoder aus der zweiten Phase bis zu 10 Sekunden. In dieser Phase liegt die Akkubelastung bei nur noch 1,8 mA. Falls die Fahrzeuge nicht länger als eine Woche abgestellt werden, könnte ein Schalter zum Ein-/Ausschalten entfallen. Davon ist jedoch abzuraten, da über den Schalter der Mikrocontroller zum Beispiel resettet werden kann.

4.2.4 Programmierung der Konfigurationsvariablen

Viele Verhaltenseinstellungen des Fahrzeugdekoders lassen sich über eine Programmierung der Konfiguration beeinflussen. Dabei werden in die EEPROM-Register des Fahrzeugdekoder, benutzerdefiniert Einstellungen abgelegt. Über diese Registerwerte lassen sich die Fahrzeugdekoder speziell auf das jeweilige Fahrzeug anpassen. So kann zum Beispiel über das Register CV#1 die Dekoderadresse festgelegt werden. Über die Registeradressen CV#11 bis CV#16 lässt sich das Funkmodul konfigurieren. Um zum Beispiel alle Registerwerte wieder zurückzusetzen, muss ein beliebiger Wert in das Register CV#8 geschrieben werden.

Der Fahrzeugdekomodulator kann über PoM-Programming von der Steuerungssoftware oder USB-Serial-Interface konfiguriert werden. Über die USB-Schnittstelle kann mit Hilfe von Text-Befehlen der Programmervorgang gesteuert werden. Zusätzlich lassen sich Übersichten über den Akkuzustand, die konfigurierten RFM12 Funkmoduldaten und die eingestellte Adresse ermitteln. Falls der Fahrzeugdekomodulator nicht mehr auf die Funkübertragung reagiert, kann so die interne Konfiguration auf Fehler überprüft werden. Eine Auflistung aller programmierbaren Register des Fahrzeugdekoder ist im Anhang E.1 aufgeführt.

4.2.5 Ladestation

In diesem Kapitel wird die Ladung der Fahrzeuge beschrieben. Alle Fahrzeuge werden mit einem Lithium-Polymer-Akku (LiPo) betrieben. Für diesen Akkutyp sind spezielle Ladegeräte notwendig.

Funktionsweise

Der verbaute Mikrocontroller Atmega32U4 arbeitet bei einer Frequenz von 8 MHz und ab einer Spannung von 2,7 Volt [3]. Der eingesetzte, 1-Zellen-LiPo-Akku besitzt ei-

⁸einstellbar über CV#65

ne maximale Zellenspannung von 3,7 Volt, welche auf minimal 3,1 Volt absinken darf, bevor der Akku durch Tiefentladung beschädigt wird. Der LiPo-Akku gewährleistet damit den Betrieb des Mikrocontroller zu jedem Zeitpunkt, welcher deshalb Rückmeldungen über den aktuellen Lade und Entladezustand geben kann. Für die Ladung der Fahrzeuge ist kein spezielles externes Ladegerät erforderlich. Die Ladung erfolgt über eine geregelte 5,0 Volt Spannung. Diese kann über die USB-Versorgung oder über Kontakte an den Außenspiegeln des Fahrzeuges angelegt werden. Die Ladung startet danach automatisch je nach Ladezustand des LiPo-Akkus. Mit der Möglichkeit, die Fahrzeuge über die Außenspiegel zu laden können diese während des Betrieb ohne Eingriff voll automatisch in eine spezielle Ladestation gelenkt werden. Nach einer erfolgreichen Ladung werden die Autos automatisch wieder in den Verkehr eingegliedert. So ist ein reibungsloser Betrieb über längere Zeit möglich.

Hardware

Die Steuerung des Ladevorgangs übernimmt ein zusätzlicher Charge-Management-Controller. Dieser ist auf dem Fahrzeugdekoder untergebracht. Der MCP73831 von Microchip ist speziell als LiPo-Laderegler für Ladespannungen von 15 bis 500 Milliamper (mA) geeignet [21]. Es ist notwendig den Fahrzeugdekoder je nach verwendeter LiPo-Akku Leistung anzupassen. Dazu muss der Programmierwiderstand des MCP73831 wie folgt berechnet werden:

$$mAh = \frac{1000 \text{ Volt}}{kOhms} \quad (4.1)$$

Die Ladung sollte optimal auf 1C eingestellt werden. Dies ist eine schonende Ladung für die Akkus und die Wärmeentwicklung ist relativ gering. Ein kompletter Ladevorgang dauert dann zirka eine Stunde.

Gesteuert wird der Ladevorgang komplett automatisch durch den Laderegler MCP73831 im Fahrzeugdekoder. Um den Ladevorgang automatisch zu Starten übernimmt die Steuerungssoftware Rocrail (siehe Kapitel 2.4) mit Hilfe von Aktionen die Steuerung des Fahrzeugs zur Ladestation. Dazu hat jeder Fahrzeugdekoder zwei 11 Bit Rückmeldeadressen, die über Konfigurationsvariablen CV#20 und CV#21 gesetzt werden können. Die erste Adresse wird als aktiv gemeldet, wenn der Akku den eingestellten Wert von CV#29 unterschreitet. Die zweite Adresse ist die chronologisch folgende (+1) und wird aktiv wenn der Ladevorgang abgeschlossen ist. Nach dem Eintreffen des Fahrzeugs in der Ladestation, wird nach Ablauf von fünf Sekunden Standzeit des Fahrzeugs, die Ladestation geschlossen. Mit dem schließen der Kontakte startet der Ladevorgang sofort. Das Fahrzeug wird für die Zeit der Ladung daran gehindert, eine neue Fahrstraße zu reservieren um dadurch die Ladestation zu verlassen. Für die Zeit der Ladung wird außerdem die Beleuchtungsfunktion F0 abgeschaltet. Durch die aktive Rückmeldung des Fahrzeugdekoders beim Beenden des Ladevorgangs, wird die Ladestation über die Steuerungssoftware geöffnet und die Fahrstraße wieder freigegeben. Das Fahrzeug kann nun den Fahrbetrieb wieder aufnehmen.

Kapitel 5

Zusammenfassung

Im Rahmen der Masterarbeit wurde ein System zur Steuerung von Miniaturmodellfahrzeugen mit der notwendigen Hard- und Software als Prototyp erstellt. Dabei wurden handelsübliche Protokolle für die Datenübertragung zwischen der Computeranwendung und der Zentrale verwendet. Für die Kommunikation mit den Fahrzeugen wurde ein neues RCMC-Funk-Protokoll entwickelt, das sich auch für zukünftige Erweiterungen ausgelegt ist. Im kompletten Steuerungssystem wird mit dem Atmega32U4 einheitlich der gleiche SMD-Mikrocontroller verwendet. Dieser bietet eine integrierte USB-Schnittstelle. Über diese können neue Software oder Konfigurationen in den Mikrocontroller übertragen werden. Das verwendete RFM12B Funkmodul ist flexibel programmierbar und dank seiner geringen Baugröße auf der Rückseite des Fahrzeugdekoder einfach zu installieren. Die Versorgungsspannung des Fahrzeugdekoder erfolgt über einen LiPo-Akku, der mehr Leistung bietet und dabei weniger Platz als herkömmliche Nickel-Metallhydrid-Akkus benötigt. Für eine einfache Ladung des 1-Zellen-LiPo-Akku wurde ein Laderegler auf dem Fahrzeugdekoder installiert. Dieser steuert den Ladevorgang automatisch.

Die erstellte Software für die Zentrale und den Fahrzeugdekoder wurde ausführlich auf einer speziell angepassten und miniaturisierten Fahrstraße im Zusammenspiel mit der Computeranwendung getestet. Anhand von Fahrversuchen mit den umgebauten Modellfahrzeugen im Maßstab 1:120 wurden Softwareverbesserungen erstellt. Dazu wurde zum Beispiel die Funktion der Lastregelung bei Berg- und Talfahrt erprobt, die Leistungsfähigkeit des LiPo-Akkus getestet und das automatische Einfahren sowie Laden der Fahrzeuge durchgeführt. Die vorliegende Arbeit befasst sich nur mit einem Teil der Komponenten, die für ein automatisiertes Steuerungssystem benötigt werden. Zur vollständigen Automatisierung werden eine Vielzahl an Knoten im Loconet™-Bussystem benötigt. Diese sind zur Rückmeldung der Fahrzeugpositionen, als auch zur Steuerung der Abzweigungen (Weichen) in der Fahrstraße notwendig. Durch vorher festgelegte und in der Computeranwendung programmierte Abläufe können auf diese Weise die Modellfahrzeuge bestimmte Ziele anfahren. Dazu muss zuvor die Fahrstraße mit ihren Eigenschaften und Einschränkungen für bestimmte Fahrzeuge in der Computeranwendung erfasst werden.

Während des Umbaus der Modellfahrzeuge von Infrarotsteuerung auf die Funkda-

tenübertragung zeigte sich, dass der Funkfahrzeugdekoder mit LiPo-Akku weniger Platz benötigt. Das entwickelte Prototypsystem bietet bereits eine Vielzahl von Funktionen. Eine mögliche Erweiterung ist die Anbindung einer Handsteuerung für Modellfahrzeuge. Diese kann über das LocoNet™-Protokoll erfolgen, da für diese Schnittstelle bereits Handsteuerungen für Modelleisenbahnen verfügbar sind. Außerdem könnte mit einer kleinen Computer-Anwendung zur Konfiguration und Programmierung der Fahrzeugdekoder, deren Einstellungen benutzerfreundlicher geändert werden. Die Programmierung neuer Software in den Fahrzeugdekoder kann dann direkt über die Funkkommunikation stattfinden. Dadurch wäre es nicht mehr notwendig, die Fahrzeuge von der Fahrstraße zu nehmen. Mit einer eigenen Computeranwendung für den Automatikbetrieb kann das Fahrverhalten der Fahrzeuge weiter optimiert werden. Dann könnten sich mehrere Fahrzeuge gleichzeitig in einem Block befinden und dichter aufeinander auffahren. Dadurch wäre es auch möglich, eine Vielzahl an Rückmeldern in der Fahrstraße einzusparen.

Tabellenverzeichnis

A.1	Aufbau von Systemstatusnachrichten des P50-Protokolls.	47
A.2	Aufbau von Systemstatusnachrichten.	49
A.3	Aufbau von Lokstatusnachrichten.	50
A.4	Aufbau von Zubehördekodernachrichten.	51
A.5	Nachrichtenaufbau von Rückmeldeinformationen.	52
A.6	Programmernachrichten für verschiedene Programmiermodi.	54
B.1	LocoNet 2 Byte Nachrichten.	55
B.2	LocoNet 4 Byte Nachrichten.	56
B.3	LocoNet 6 Byte Nachrichten.	57
B.4	LocoNet N Byte Nachrichten.	58
B.5	LocoNet Verschiebe Byte von Quelle nach Ziel.	58
B.6	LocoNet Verschiedene Nachrichten mit N Byte.	58
E.1	Werte und Beschreibungen der CV-Konfigurationsvariablen im Fahrzeug-dekoder.	70

Abbildungsverzeichnis

2.1	Arduino Leonardo Entwicklungs-Board [2].	3
2.2	Arduino IDE Entwicklungsumgebung.	4
2.3	Funkmodul RFM12B-S2 [16].	5
2.4	Aufbau eines LocoNet TM -Datenpakets [4].	9
2.5	Kodierung der Nachrichtenlänge und des Opcode im ersten Byte der LocoNet TM -Nachricht [4].	10
2.6	Aufbau eines LocoNet TM -Datenpaket [4].	10
3.1	Komponenten eines originalen Modellfahrzeug der Firma Faller [14].	13
3.2	Prototyp der für Fahrversuche genutzten Fahrstraße.	14
3.3	Blockbild aller Systemkomponenten der Funkfahrzeugsteuerung und deren Kommunikationsprotokollen.	15
3.4	Blockbild des gestamten Funktionsaufbaus.	15
4.1	Blockbild des Funktionsaufbau der Zentrale.	18
4.2	Struktur der Datenpakete des RFM12 Funkmoduls [23].	20
4.3	Draufsicht der prototypisch aufgebauten Zentrale.	21
4.4	Platinenrückseite der prototypisch aufgebauten Zentrale.	22
4.5	Softwareablauf der Zentrale.	25
4.6	Komponenten des Fahrzeugdekoders.	27
4.7	Fahrzeugdekker mit Atmega32U4 und RFM12B.	28
4.8	Mini USB Adapter für den Fahrzeugdekker.	29
C.1	Ablauf des Sendevorgangs für Geschwindigkeit- oder Funktionsstatus im RCMC-Protokoll.	61
C.2	Ablauf der Datenübertragung bei der CV-Programmierung über das RCMC-Protokoll.	62
C.3	Ablauf zum Setzen eines Rückmelders vom Fahrzeugdekker über das RCMC-Protokoll.	62
D.1	Schaltung Funk Fahrzeugdekker.	63
D.2	Schaltung der Zentrale.	64

Literaturverzeichnis

- [1] ARDUINO TEAM: *Arduino*. <http://arduino.cc/en/Main/HomePage>, Stand: 24.05. 2013.
- [2] ARDUINO TEAM: *Arduino*. <http://arduino.cc/de/Main/ArduinoBoardLeonardo>, Stand: 24.05. 2013.
- [3] ATMEL: *8-bit AVR Microcontroller with 16/32K Bytes of ISP Flash and USB Controller*. <http://www.atmel.com/Images/doc7766.pdf>, Stand: 07.06. 2013.
- [4] BORMANN, STEFAN: *LcoNet*. http://fremodcc.sourceforge.net/diy/loconet_e.html, Stand: 21.06. 2013.
- [5] BUNDESNETZAGENTUR FÜR ELEKTRIZITÄT, GAS, TELEKOMMUNIKATION, POST UND EISENBAHNEN: *Non-specific Short Range Devices (SRD)*. http://www.bundesnetzagentur.de/SharedDocs/Downloads/DE/Sachgebiete/Telekommunikation/Unternehmen_Institutionen/Frequenzen/Allgemeinzuteilungen/SRD_Vfg432012.pdf?__blob=publicationFile&v=3, Stand: 30.07. 2013.
- [6] CADSOFT: *Eagle PCB-Software*. <http://www.cadsoft.de/?lang=de>, Stand: 14.07. 2013.
- [7] DCCWIKI: *LcoNet*. <http://www.dccwiki.com/LcoNet>, Stand: 24.05. 2013.
- [8] DIGITRAX, INC.: *LcoNet Personal Use Edition 1.0 SPECIFICATION*. <http://www.digitrax.com/static/apps/cms/media/documents/loconet/loconetpersonaledition.pdf>, Stand: 24.05. 2013.
- [9] DIGITRAX, INC.: *LcoNet System Architecture-Complete Train Control*. <http://www.digitrax.com/support/loconet/home>, Stand: 24.05. 2013.
- [10] DIGITRAX, INC.: *LcoNet, the Digitrax Difference*. <http://www.digitrax.com/static/apps/cms/media/documents/documentation/LocoNet%20Overview%20App%20Note.pdf>, Stand: 16.07. 2013.

- [11] DIPL. ING. KERSTEN TAMS: *Beschreibung der unterstützten Interface-Commands EasyControl.* <http://www.tams-online.de/htmls/download/EasyControl/interface.txt>, Stand: 08.05. 2013.
- [12] GAHTOW, PHILIPP: *Infrarot Übertragungssystem zur Steuerung von Modellfahrzeugen.* Bachelorarbeit, Universität der Bundeswehr München, 2011.
- [13] GEBR. FALLER GMBH: *Car System Digital 3.0 - Fahrspaß der neuen Generation.* http://www.faller.de/xs_db/DOKUMENT_DB/www/allgemeines/FALLER_Car_System_Digital_2013.pdf, Stand: 04.08. 2013.
- [14] GEBR. FALLER GMBH: *Faller - Car System.* <http://faller.de/App/WebObjects/XSeMIPS.woa/cms/page/pid.14.17.109/ecm.p/Car-System.html>, Stand: 14.03. 2013.
- [15] GUREVICH, VLADIMIR: *Electric Relays - Principles and Applications.* CRC Press Inc, Bosa Roca, 2005.
- [16] HOPE MICROELECTRONICS CO.,LTD: *RFM12 Universal ISM Band FSK Transceiver.* <http://www.hoperf.com/upload/rf/RFM12b.pdf>, Stand: 24.05. 2013.
- [17] J. GRÄBNER, U. JOHANN, J. BARTELS A. HÜBSCH G. SCHOLZ J. DOLZE W. FALKENBACH C. LINDECKE O. SAENGER UND L. MOOSHAMMER: *Modellbahnsteuerung via PC.* http://www.der-moba.de/index.php/Modellbahnsteuerung_via_PC, Stand: 14.07. 2013.
- [18] KUFER, WOLFGANG: *OpenDCC Z1 - Zentrale für DCC - Befehle im Intellibox-Mode.* http://www.opendcc.de/elektronik/opendcc/opendcc_doc_ib.html, Stand: 08.05. 2013.
- [19] LANG, JEAN-PHILIPPE: *RF12 wireless driver.* http://jeelabs.net/pub/docs/jeelib/md_intro_rf12.html, Stand: 24.05. 2013.
- [20] LENZ, BJÖRN: *Lithium Polymer Akku - Lipo Akku.* <http://www.akku-abc.de/lipo-akku.php>, Stand: 18.07. 2013.
- [21] MICROCHIP TECHNOLOGY INC.: *MCP73831 Miniature Single-Cell, Fully Integrated Li-Ion, Li-Polymer Charge Management Controllers.* <http://ww1.microchip.com/downloads/en/devicedoc/21984e.pdf>, Stand: 07.06. 2013.
- [22] RUDOLF MÄUSL UND JÜRGEN GÖBEL: *Analoge und digitale Modulationsverfahren. Basisband und Trägermodulation.* Hüthig, 2002.
- [23] RUSU, FELIX: *RFM12B library.* <http://lowpowerlab.com/blog/2012/12/28/rfm12b-arduino-library>, Stand: 20.06. 2013.

- [24] SHEPHERD, ALEX: *Embedded LocoNet*. <http://sourceforge.net/projects/embeddedloconet>, Stand: 04.06. 2013.
- [25] SHIRRIFF, KEN: *Secrets of Arduino PWM*. <http://www.righto.com/2009/07/secrets-of-arduino-pwm.html>, Stand: 08.06. 2013.
- [26] VERSLUIS, ROB: *LocoNet Protokoll*. <http://wiki.rocrail.net/doku.php?id=loconet:lnpe-de>, Stand: 28.05. 2013.
- [27] VERSLUIS, ROB: *Rocrail Project in a Nutshell*. <http://wiki.rocrail.net/doku.php?id=start>, Stand: 26.05. 2013.

Anhang A

Erläuterung P50X-Protokollbefehle

Nachfolgend ist die Kommandoübersicht der implementierten P50 und P50X Befehle aufgelistet [18, 11].

A.1 Systemdatenpakete

Diese Datenpakete enthalten Informationen über den Systemzustand und offene Ereignisse, die abzufragen sind. In der Tabelle A.1 sind die drei implementierten Befehle aus dem P50-Protokoll aufgelistet.

P50 Nachricht	Antwort
(0XC4) Idle	Leeres Datenpaket mit 0X00 als Antwort.
(0X60) Pwr on	Gleisspannung ein.
(0X61) Pwr off	Gleisspannung aus.

Tabelle A.1: Aufbau von Systemstatusnachrichten des P50-Protokolls.

Im Weiteren sind alle Befehle aus der P50X-Protokollerweiterung aufgelistet, die implementiert wurden. Die Tabelle A.2 beeinhaltet Systembefehle, die Auskunft über den Systemstatus geben.

OpCode	Antwort
(0XA0) XVer Versionsabfrage	Antwort: Liste, jeder Eintrag ist wie folgt aufgebaut: Byte0: Anzahl der Bytes für dieses Listenelement. 0x00 bedeutet Listenende. Byte1: Niederwertiges Byte der Version. Byte2: Höherwertigere Bytes, sofern vorhanden.
(0XA2) XStatus Statusabfrage für Spannung, Zustand usw.	Antwort: Bit-Feld: Bit 7: Erweiterungsbit, 0x00. Bit 6: VREG Spannungsregelung. Bit 5: externes I2C Gerät vorhanden Bit 4: Halt, Loks angehalten, aber Gleisspannung aktiv. Bit 3: Power Bit 2: Überhitzung Bit 1: Go Bit 0: Stop

(0XC8)	Statusabfrage für Zustandsänderungen.
XEvent	Antwort ist eine Liste von Bit-Feldern:
	Byte 1:
	Bit 7: Erweiterungsbit, 0x01 wenn ein Byte folgt.
	Bit 6: LSY: Meldung eines Lissy-Ereignis.
	Bit 5: TRNT: Meldung einer Weichenänderung.
	Bit 4: Tres: Mindestens ein Zugriff auf eine reservierte Weiche.
	Bit 3: PWR: Versorgungsspannung aus Ereignis.
	Bit 2: S88: Mindestens ein S88 Ereignis.
	Bit 1: GO: Mindestens ein Start Ereignis.
	Bit 0: LOK: Mindestens ein Zugriff auf eine Lok.
	Byte 2:
	Bit 7: Erweiterungsbit, 0x01 wenn ein Byte folgt.
	Bit 6: Sts: Änderung im Status.
	Bit 5: Hot: Überhitzung.
	Bit 4: PTsh: Kurzschluss Programmier- zum Hauptgleis.
	Bit 3: RSsh: Kurzschluss auf Programmier- oder Hauptgleis.
	Bit 2: IntSh: Kurzschluss intern.
	Bit 1: LMSh: Kurzschluss auf dem Lokmausanschluss.
	Bit 0: ExtSh: Kurzschluss eines externen Boosters.
	Byte 3:
	Bit 7: Erweiterungsbit, 0x01 wenn ein Byte folgt.
	Bit 6: EvBiDi: BiDi Orts- oder Geschwindigkeitsrückmeldung erhalten. Abfrage über XEvtBiDi.
	Bit 5: BiDiCV: BiDi CV-Meldung erhalten. Abfrage über XEvtPT.
	Bit 4: ExVlt: Fremdspannung vorhanden.
	Bit 3: TkRel: Übernahme einer Lok durch einen Fremdregler.
	Bit 2: Mem: Speicher Ereignis.
	Bit 1: RSOF: RS232 Überlauf.
	Bit 0: PT: Programmiergleis Ereignis.
(0XC4) XNop	Leerlaufkommando, wird genutzt zur Autoumschaltung der Bau-drate (BABl). Antwort ist das 0X00 Okay Kommando.
(0XA5) XHalt	Stoppen aller Loks bei aktiver Gleisspannung. Antwort ist das 0X00 Okay Kommando.
(0XA6) XPwrOff	Hauptgleisspannung abschalten. Antwort ist das 0X00 Okay Kommando.
(0XA7) XPwrOn	Hauptgleisspannung einschalten. Antwort ist das 0X00 Okay Kommando.

Tabelle A.2: Aufbau von Systemstatusnachrichten.

A.2 Lok Datenpakete

Die Tabelle A.3 beinhaltet die Befehle zur Steuerung von Lokomotiven. Diese werden hier in der Arbeit für den Fahrzeugdekoder verwendet.

OpCode	Antwort
(0X80) XLoc	Byte1: LSB der Lokadresse.
(0X81) XLocX	Byte2: MSB der Lokadresse (1... 10239). Byte3: Speed unabhängig von Lokformat (0... 127). Byte4: Funktion- und Statusbits: Bit 7: Gültigkeit Funktionsbits Bit 6: erzwinge Senden. Bit 5: Bit Fahrtrichtung, Bit 4: Lichtfunktion, Bit 3-0: 4 Bit für Funktion F4 bis F1. Antwort: 1 Byte OK (0x00) oder Fehlercode.
(0X88) XFunc	Byte1: LSB der Lokadresse. Byte2: MSB der Lokadresse (1... 10239). Byte3: Statusbits für F8 (Bit 7) bis F1 (Bit 0). Antwort: 1 Byte OK (0x00) oder Fehlercode.
(0X89) XFunc2	Byte1: LSB der Lokadresse. Byte2: MSB der Lokadresse (1... 10239). Byte3: Statusbits für F16 (Bit 7) bis F9 (Bit 0). Antwort: 1 Byte OK (0x00) oder Fehlercode.
(0X8A) XFunc34	Byte1: LSB der Lokadresse. Byte2: MSB der Lokadresse (1... 10239). Byte3: Statusbits für F24 (Bit 7) bis F17 (Bit 0). Byte4: Statusbits für F28 (Bit 3) bis F25 (Bit 0). Antwort: 1 Byte OK (0x00) oder Fehlercode.
(0X84) XlokSts	Byte1: LSB der Lokadresse. Byte2: MSB der Lokadresse (1... 10239). Antwort: 1 Byte OK (0x00) gefolgt von 1 Byte Status (F8 bis F1) oder Fehlercode.
(0XC9) XEvtLok	Rückmeldung aller Änderungen von Lokdaten seit der letzten Abfrage. (zum Beispiel durch Handregler) Antwort: Byte1: Liste mit Geschwindigkeiten (0X00 bis 0X7F) oder 0X80 keine Lok zu melden. Byte2: Statusbits für F8 (Bit 7) bis F1 (Bit 0). Byte3: LSB der Lokadresse. Byte4: MSB der Lokadresse, Funktion F0 und Fahrtrichtung. Byte5: Aktuelle Fahrstufe.

Tabelle A.3: Aufbau von Lokstatusnachrichten.

A.3 Zubehör Datenpakete

Die Tabelle A.4 stellt die implementierten Weichenbefehle dar.

OpCode	Antwort
(0X90) XTrnt	Weichenschaltbefehl: Byte1: LSB der Weichenadresse. Byte2: MSB der Weichenadresse (0...2043) und Statusbits: Bit 0-3: MSB der Weichenadresse Bit 4: NoCmd: Kommando wird ignoriert. Bit 5: Res: Verriegelung der Weiche. Bit 6: Sts: Spulenzustand (0 = aus, 1 = ein). Bit 7: Farbe: 0 = rot (abzweigend), 1 = grün (gerade). Antwort ist das 0X00 Okay Kommando oder Fehlercode.
(0X93) XTrntFree	Weichenreservierung wird nicht unterstützt. Antwort ist das 0X00 Okay Kommando.
(0X94) XTrntSts	Statusabfrage des Schaltzustands: Byte1: LSB der Weichenadresse. Byte2: MSB der Weichenadresse (0...2043) und Statusbits: Antwort: Byte1: 0X00 Kommando wurde akzeptiert oder Fehlercode. Byte2: Weichenstatus: (Bit 3: Conf1, Bit 2: Color, Bit 1: Res, Bit 0: Conf0) Conf0/1 beinhalten der Protokoll: 00: Motorola (Märklin) 01: SX (Selectrix-System) 10: DCC (Digital Command Control) 11: FMZ (Fleischmann)
(0X95) XTrntGrp	Gruppenweise Statusabfrage Schaltbefehl. Byte1: Adresse der Gruppe = $((T\text{Adr} - 1)/8) + 1$. Byte2: MSB der Weichenadresse. Antwort: Byte1: 0X00 Kommando wurde akzeptiert. Byte2: Bit-Feld mit Positionen der acht Weichen (rot/grün). Byte3: Reservierungsflags für jede Weiche.
(0XCA) XEvtTrnt	Rückmeldung von Weichenzuständen. Antwort: Byte1: Anzahl der zu meldenden Weichen (maximal 64). Für jede Weiche X, werden zwei Byte gesendet: ByteX1: LSB der Weichenadresse. ByteX2: MSB der Weichenadresse und Farbe.

Tabelle A.4: Aufbau von Zubehördekodernachrichten.

A.4 Rückmelde Datenpakete

In der Tabelle A.5 sind die unterstützten Befehle für Rückmeldeereignisse aufgelistet.

OpCode	Antwort
(0X98) XSensor	Auslesen der Änderungen eines Rückmeldemoduls. Byte1: S88 16-fach Rückmeldemodulnummer (1..128) Antwort: Byte1: 0X00 Kommando wurde akzeptiert oder Fehlercode. Byte2: Kontakte eins bis acht des Moduls (Bit 7...0). Byte3: Kontakte neun bis 16 des Moduls (Bit 7...0).
(0X99) XSensOff	Rücksetzen aller gespeicherten S88 Zustände. Antwort: 0X00 Kommando wurde akzeptiert.
(0X9D) X88PSet	Einstellung der S88 Modulanzahl (2 Byte pro Modul). Byte1: Zahl der automatisch gelesenen Byte. Byte2: Zahl der Byte auf Port 1. Byte2: Zahl der Byte auf Port 2. Byte2: Zahl der Byte auf Port 3. Antwort: 0X00 Kommando wurde akzeptiert.
(0xCB) XEvtSen	Meldung geänderter S88 Zustände. Antwort: Byte1: Modulnummer (1...128) oder Null für Listenende. Byte2: Kontakte eins bis acht des Moduls (Bit 7...0). Byte3: Kontakte neun bis 16 des Moduls (Bit 7...0).

Tabelle A.5: Nachrichtenaufbau von Rückmeldeinformationen.

A.5 Programmierung Datenpakete

Bei Modelleisenbahnen werden die Konfigurationsvariablen (CV) im Dekoder über Programmierung verändert. Die Daten werden im EEPROM¹ des Mikrocontroller erfasst. Hierzu sind verschiedene Modi vorhanden um diese Programmierung durchzuführen. Das hier in der Arbeit genutzte Verfahren “Programming on the Main” (kurz PoM) arbeitet im laufenden Betrieb und eignet sich hervorragend für Änderungen der Konfigurationsvariablen der Modellfahrzeuge. In der folgenden Tabelle A.6 sind die Inhalte der Programmierdatenpakete dargestellt.

OpCode	Antwort
(0XE1)	Aktivieren des Programmiermodus.
XPT_On	Antwort ist das 0X00 Okay Kommando.
(0XE2)	Programmiermodus Beenden.
XPT_Off	Antwort ist das 0X00 Okay Kommando.

¹Electrically Erasable Programmable Read-Only Memory

(0XF0)	Byte1: LSB der CV Adresse.
XPT_DCCRD	Byte2: MSB der CV Adresse (0... 1023).
CV Lesen im Direct-Mode.	Antwort: 1 Byte OK (0x00) Kommando.
(0XF1)	Byte1: LSB der CV Adresse.
XPT_DCCWD	Byte2: MSB der CV Adresse (0... 1023).
CV Schreiben im Direct-Mode.	Byte3: Inhalt der CV Adresse (0... 255). Antwort: 1 Byte OK (0x00) Kommando.
(0XF2)	Byte1: LSB der CV Adresse.
XPT_DCCRB	Byte2: MSB der CV Adresse (0... 1023).
Liest CV mit DCC Bit-Read.	Antwort: 1 Byte OK (0x00) Kommando.
(0XF3)	Byte1: LSB der CV Adresse.
XPT_DCCWB	Byte2: MSB der CV Adresse (0... 1023).
Einzelbit Schreiben.	Byte3: Bitposition (0... 7). Byte4: Inhalt des Bit (0... 1). Antwort: 1 Byte OK (0x00) Kommando.
(0XEE)	Byte1: LSB der CV Adresse.
XPT_DCCRP	Byte2: MSB der CV Adresse (0... 1023).
Lesen im Page-Mode.	Antwort: 1 Byte OK (0x00) Kommando.
(0XEF)	Byte1: LSB der CV Adresse.
XPT_DCCWP	Byte2: MSB der CV Adresse (0... 1023).
Schreiben im Page-Mode.	Byte3: Inhalt der CV Adresse (0... 255). Antwort: 1 Byte OK (0x00) Kommando.
(0XDA)	Byte1: LSB der Lokadresse.
XDCC_PDR	Byte2: MSB der Lokadresse (1... 10239).
CV Lesen über POM auf dem Hauptgleis.	Byte2: LSB der CV Adresse. Byte3: MSB der CV Adresse (0... 1023). Antwort: 1 Byte OK (0x00) Kommando oder 0X80 wenn beschäftigt.
(0XDE)	Byte1: LSB der Lokadresse.
XDCC_PDR	Byte2: MSB der Lokadresse (1... 10239).
CV Schreiben über POM auf dem Hauptgleis.	Byte2: LSB der CV Adresse. Byte3: MSB der CV Adresse (0... 1023). Byte4: Inhalt der CV Adresse (0... 255). Antwort: 1 Byte OK (0x00) Kommando oder 0X80 wenn beschäftigt.
(0XFE)	Beenden der aktuellen Programmieraufgabe (Abbruch).
XPT_Term	Antwort: 1 Byte OK (0x00) Kommando oder 0XF3 keine aktive Programmierung.

(0XCE) XEvtPT	Abfrage Programmierergebnisse. Antwort: Byte1: 0XF5 Programmierung noch nicht beendet oder Anzahl Folgebyte. Byte2: Status der Programmieraufgabe: 0X00: Programmierung erfolgreich, kein Fehler. 0XFF: Zeitüberschreitung. 0XFE: Kein Acknowledge vom Dekoder. 0XFD: Kurzschluss auf dem Programmiergleis. 0XFC: Kein Dekoder gefunden. 0XFB: Allgemeiner Fehler. 0XFA: Fehler beim Schreiben im Direct-Mode. 0XF9: Kein Acknowledge im Paged-Modus. 0XF8: Fehler beim Selectrix lesen. 0XF7: XPT_DCCQD: OK, Direct-Bit-Read wird unterstützt. 0XF6: XPT_DCCQD: Fehler, Direct-Bit-Read nicht möglich. 0XF4: Abbruch der Programmierung. 0XF3: Keine Programmierung zum Abbruch aktiv. 0XF2: Kann Programmierung nicht Abbrechen. Byte3: (optional) Erstes Ergebnis: Inhalt der CV oder LSB der DCC-Adresse. Byte4: (optional) Zweites Ergebnis: MSB der DCC-Adresse.
(0XA4) XSoGet	SO EEPROM der Zentrale lesen. Byte1: LSB der SO Adresse. Byte2: MSB der SO Adresse (0...1023). Antwort: Byte1: OK (0x00) Kommando und Folgebyte oder Fehlercode. Byte2: Inhalt der Spezialoption.
(0XA3) XSOSet	SO EEPROM der Zentrale schreiben. Byte1: LSB der SO Adresse. Byte2: MSB der SO Adresse (0...1023). Byte3: Inhalt der SO Adresse (0...255). Antwort: 1 Byte OK (0x00) Kommando oder Fehlercode.

Tabelle A.6: Programmiernachrichten für verschiedene Programmiermodi.

Anhang B

LocoNetTM-Protokoll Befehle

In diesem Abschnitt werden die Datenpakete des LocoNetTM-Protokoll erläutert. Dazu wurden die Befehle nach ihrer Länge gegliedert aufgelistet.

B.1 2 Byte Nachrichten

Nachrichten welche aus 1 Byte Opcode und 1 Byte Prüfsumme bestehen sind in der Tabelle B.1 aufgelistet [8].

OpCode	Beschreibung
OPC_IDLE (0X85)	Broadcast Stopbefehl für alle Dekoder.
OPC_GPON (0X83)	Globale Versorgungsspannung ein.
OPC_GPOFF (0X82)	Globale Versorgungsspannung aus.
OPC_BUSY (0X81)	Master ist beschäftigt.

Tabelle B.1: LocoNet 2 Byte Nachrichten.

B.2 4 Byte Nachrichten

Nachrichten bestehend aus 1 Byte Opcode, 2 Byte Daten und 1 Byte Prüfsumme sind in der Tabelle B.2 aufgelistet [8].

OpCode (OPC)	Beschreibung	Byte 1	Byte 2	Antwort
LOCO_ADR (0XBF)	Lokanfrage: Falls Lok nicht in einem Slot wird vom Master ein Slot vergeben.	0	ADR	OK: OPC_SL_RD_DATA, Fehler: ACK1=0X00.
SW_ACK (0XBD)	Weichen Funktion bestätigen.	SW1	SW2	OK: ACK1=0X7F, Fehler: ACK1=0X00.
SW_STATE (0XBC)	Weichenstellbefehl	SW1	SW2	OK: ACK1=0X7F, Fehler: ACK1=0X00.
RQ_SL_DATA (0XBB)	Anfrage für Slot-Daten.	SLOT	0	OK: OPC_SL_RD_DATA.
MOVE_SLOTS (0XBA)	Verschiebe von Slot nach Slot.	Quelle	Ziel	OK: OPC_SL_RD_DATA.
LINK_SLOTS (0XB9)	Verknüpfe Slot ARG1 mit Slot ARG2.	Slot 1	Slot 2	OK: OPC_SL_RD_DATA.
UNLINK_SLOTS (0XB8)	Verknüpfung aufheben Slot ARG1 nach Slot ARG2.	Slot 1	Slot 2	OK: OPC_SL_RD_DATA.
CONSIST_FUNC (0XB6)	Schreibe Funktions-Bits im CONSIST-Element.	Slot	DIRF	-
SLOT_STAT1 (0XB5)	Schreibe Slot	Slot.	Daten	-
LONG_ACK (0XB4)	Lange Bestätigung (ACK). Schreibe Slot	Slot	Daten	-
INPUT REP (0XB2)	Rückmeldesensor	IN1	IN2	-
SWREP (0XB1)	Weichenstatusrückmeldung	SN1	SN2	-
SW_REQ (0XB0)	Aufruf Schaltfunktion.	SW1	SW2	-
LOCO SND (0XA2)	Schreibe Slot-Funktion F5 bis F8.	Slot	SND	-
LOCO_DIRF (0XA1)	Schreibe Slot-Fahrtrichtung und Funktion F0 bis F4.	Slot	DIRF	-
LOCO_SPD (0XA0)	Schreibe Geschwindigkeit.	Slot-	Slot	SPD -

Tabelle B.2: LocoNet 4 Byte Nachrichten.

B.3 6 Byte Nachrichten

LocoNetTMNachrichten bestehend aus 1 Byte Opcode, 4 Byte Daten und 1 Byte Prüfsumme sind in der Tabelle B.3 dargelegt [8].

OpCode (OPC)	Beschreibung	Byte 1	Byte 2	Byte 3	Byte 4
MULTI_SENSE (0XD0)	Power und Übertragung.	Type	Region	Adresse	Adresse
UHLI_FUN (0XD4)	Funktion F9 bis F28.	0X20	0X01	Funktions Gruppe	Funktion

Tabelle B.3: LocoNet 6 Byte Nachrichten.

B.4 Variable Nachrichtenlänge

Tabelle B.4 enthält LocoNetTMNachrichten bestehend aus 1 Byte Opcode, Anzahl Bytes Zähler, Zähler minus 3 Byte Daten und 1 Byte Prüfsumme [8].

OpCode (OPC)	Beschreibung	Count	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9	Byte 10	Byte 11
WR_SL_DATA (0XEF)	Schreibe Slot-Daten	0X0E	Slot	STAT1	Adr	SPD	DIRF	TRK	SS2	ADR2	SND	ID1	ID2
WR_SL_DATA (0XEF)	Schreibe Slot-Daten	0X0E	0X7C	PCMD	0X00	HOPSA	LOPSA	TRK	CVH	CVL	DATA7	0X00	0X00
WR_SL_DATA (0XEF)	Schreibe Slot-Daten	0X0E	0X7B	CLK_RATE	FRAC_MINSL	FRAC_MINSH	256-MINSH	TRK	256-HRS	DAYs	CLK_CNTR	ID1	ID2
SL_RD_DATA (0XE7)	Schreibe Slot-Daten	0X0E	Slot	STAT1	ADR	SPD	DIRF	TRK	SS2	ADR2	SND	ID1	ID2
0XE6	Programmier Abbruch	0X10/0X15											

Tabelle B.4: LocoNet N Byte Nachrichten.

OpCode (OPC)	Count	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12	B13
OPC_PEER_XFER (0XE5)	0X10	SRC	DSTL	DSTH	PXCT1	D1	D2	D3	D4	PXCT2	D5	D6	D7	D8

Tabelle B.5: LocoNet Verschiebe Byte von Quelle nach Ziel.

OpCode (OPC)	Beschreibung	Count	B1	B2	B3	B4	B5	B6	B7	B8
OPC_LISSY REP (0XE4)	Lissy IR report	0X08	0X00	high unit, direction	low unit	high addr	low addr	—	—	—
OPC_WHEELCNT REP (0XE4)	Radzähler report	0X08	0X40	high unit, direction	low unit	high count	low count	—	—	—
OPC_IMM_PACKET (0XED)	Send n-Byte Paket	0X0B	0X7F	REPS	DH1	IM1	IM2	IM3	IM4	IM5

Tabelle B.6: LocoNet Verschiedene Nachrichten mit N Byte.

Anhang C

RCMC-Funk-Protokoll Befehle

C.1 Datenpakete von der Zentrale zu den Fahrzeugde-kodern

- Geschwindigkeit und F0 (3 Byte)
 - Byte 0: 2 Bit 0X00 Kommando und 6 Bit MSB der Lokadresse (1 - 10239)
 - Byte 1: 8 Bit LSB der Lokadresse
 - Byte 2: F = Funktion F0; 7 Bit S = Geschwindigkeit (0 - 127)
- F1 bis F8 (3 Byte)
 - Byte 0: 2 Bit 0X01 Kommando und 6 Bit MSB der Lokadresse (1 - 10239)
 - Byte 1: 8 Bit LSB der Lokadresse
 - Byte 2: Status F1 (bit #0) .. F8 (bit #7)
- F9 bis F16 (3 Byte)
 - Byte 0: 2 Bit 0X02 Kommando und 6 Bit MSB der Lokadresse (1 - 10239)
 - Byte 1: 8 Bit LSB der Lokadresse
 - Byte 2: Status F9 (bit #0) .. F16 (bit #7)
- F17 bis F28 (3 - 4 Byte)
 - Byte 0: 2 Bit 0X03 Kommando und 6 Bit MSB der Lokadresse (1 - 10239)
 - Byte 1: 8 Bit LSB der Lokadresse
 - Byte 2: Status F17 (bit #0) .. F24 (bit #7)
 - Byte 3: Status F25 (bit #0) .. F28 (bit #7), optional nur wenn Byte 3 nicht 0x00 ist.
- CV Lesen (4 Byte)

- Byte 0: 2 Bit 0X00 Kommando und 6 Bit MSB der Lokadresse (1 - 10239)
- Byte 1: 8 Bit LSB der Lokadresse
- Byte 2: 8 Bit MSB CV Adresse (0 - 1023)
- Byte 3: 8 Bit LSB CV Adresse
- CV Schreiben (5 Byte)
 - Byte 0: 2 Bit 0X00 Kommando und 6 Bit MSB der Lokadresse (1 - 10239)
 - Byte 1: 8 Bit LSB der Lokadresse
 - Byte 2: 8 Bit MSB CV Adresse (0 - 1023)
 - Byte 3: 8 Bit LSB CV Adresse
 - Byte 4: 8 Bit CV Wert
- Broadcast Acknowledge Bestätigung (1 Byte)
 - Byte 0: 0X00 leeres Kommando und Adresse
- Broadcast Not Halt (3 Byte)
 - Byte 0: 0X00 leeres Kommando und Adresse
 - Byte 1: 0X00 keine Daten
 - Byte 2: 0X01 Notstop für alle Fahrzeugdekoder. Bei Fahrt sofort anhalten und die Warnblinkanlage aktivieren.
- Broadcast Go (3 Byte)
 - Byte 0: 0X00 leeres Kommando und Adresse
 - Byte 1: 0X00 keine Daten
 - Byte 2: 0X02 Fahrzeugdekoder reaktivieren und Warnblinkanlage abschalten.

C.2 Datenpakete von Fahrzeugdekoder zur Zentrale

Rückmeldungen vom Fahrzeugdekoder zur Zentrale.

- CV Bytewert Rückgabe (2 Byte) Dieses Paket ist die Antwort auf ein CV Lesen oder CV Schreiben Datenpaket der Zentrale.
 - Byte 0: 0X00 Kommando
 - Byte 1: Aktueller CV Wert.
- Rückmeldestatus setzen (2 Byte)
 - Byte 0: 2 Bit 0x01 Kommando, 2 Bit 0x00, 1 Bit Status und 3 Bit MSB Rückmeldeadresse (1 - 2048)
 - Byte 1: 8 Bit LSB der Rückmeldeadresse.

C.3 Datenübertragung zwischen der Zentrale und dem Fahrzeugdekoder

In diesem Abschnitt wird veranschaulicht wie die Datenübertragung zwischen der Zentrale und dem Fahrzeugdekoder über das RCMC-Funk-Protokoll abläuft.

Geschwindigkeits- und Funktionsdaten

In der Abbildung C.1 ist das Senden von Statusinformationen zu den Fahrzeugdekodern veranschaulicht. Dabei werden diese Informationen bei der Änderung des Zustands jeweils dreimal wiederholt an die Fahrzeuge gesendet.

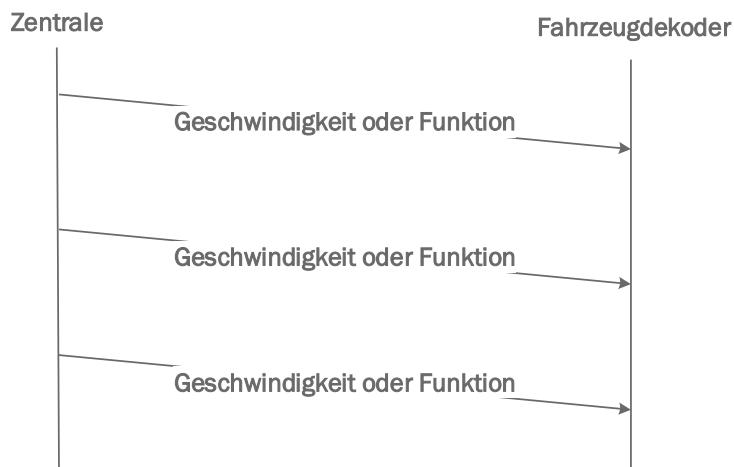


Abbildung C.1: Ablauf des Sendevorgangs für Geschwindigkeit- oder Funktionsstatus im RCMC-Protokoll.

Programmierung der Fahrzeugdekoder über Funk

Um eine sichere Programmierung der Fahrzeugdekoder über das RCMC-Funk-Protokoll zu gewährleisten, wird der Datenfluss über eine Software-Flusssteuerung realisiert. Dabei werden die Daten wie in der Abbildung C.2 dargestellt übertragen. Durch die Antwort der Gegenstelle wird davon ausgegangen, dass diese die Daten korrekt empfangen wurden.

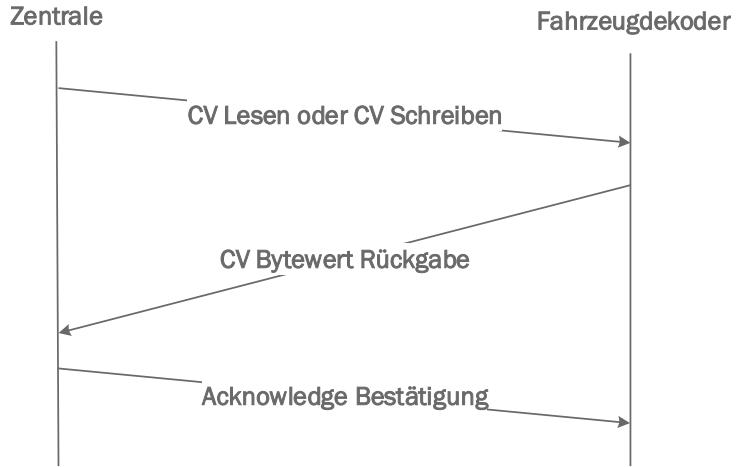


Abbildung C.2: Ablauf der Datenübertragung bei der CV-Programmierung über das RCMC-Protokoll.

Rückmeldung von Zuständen

Bei dem Setzen von Rückmeldungen die vom Fahrzeugdekoder erzeugt wurden, muss die Datenübertragung in den Datenpausen der Zentrale stattfinden. Da es trotzdem zu Problemen beim Empfang kommen kann, wird seitens des Fahrzeugdekoder eine Bestätigung gefordert. Wie in der Abbildung C.3 erkennbar, bestätigt die Zentrale den Empfang von Daten mit einem leeren Datenpaket an alle Fahrzeugdekoder. Wird keine Bestätigung erhalten wiederholt der Fahrzeugdekoder das Senden.

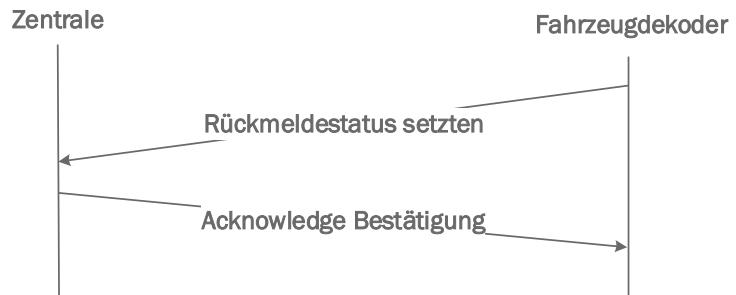


Abbildung C.3: Ablauf zum Setzen eines Rückmelders vom Fahrzeugdekoder über das RCMC-Protokoll.

Anhang D

Schaltungen

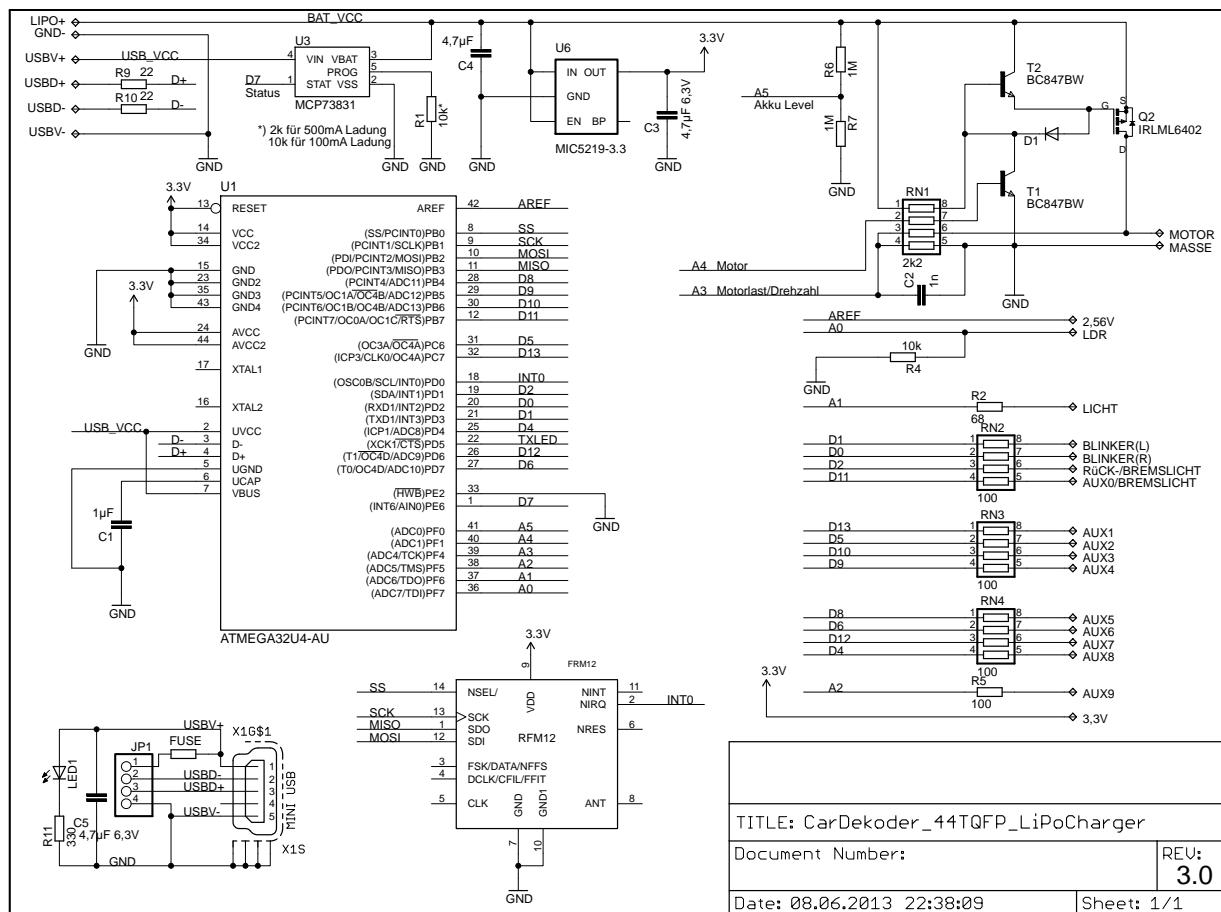


Abbildung D.1: Schaltung Funk Fahrzeugdekomodator.

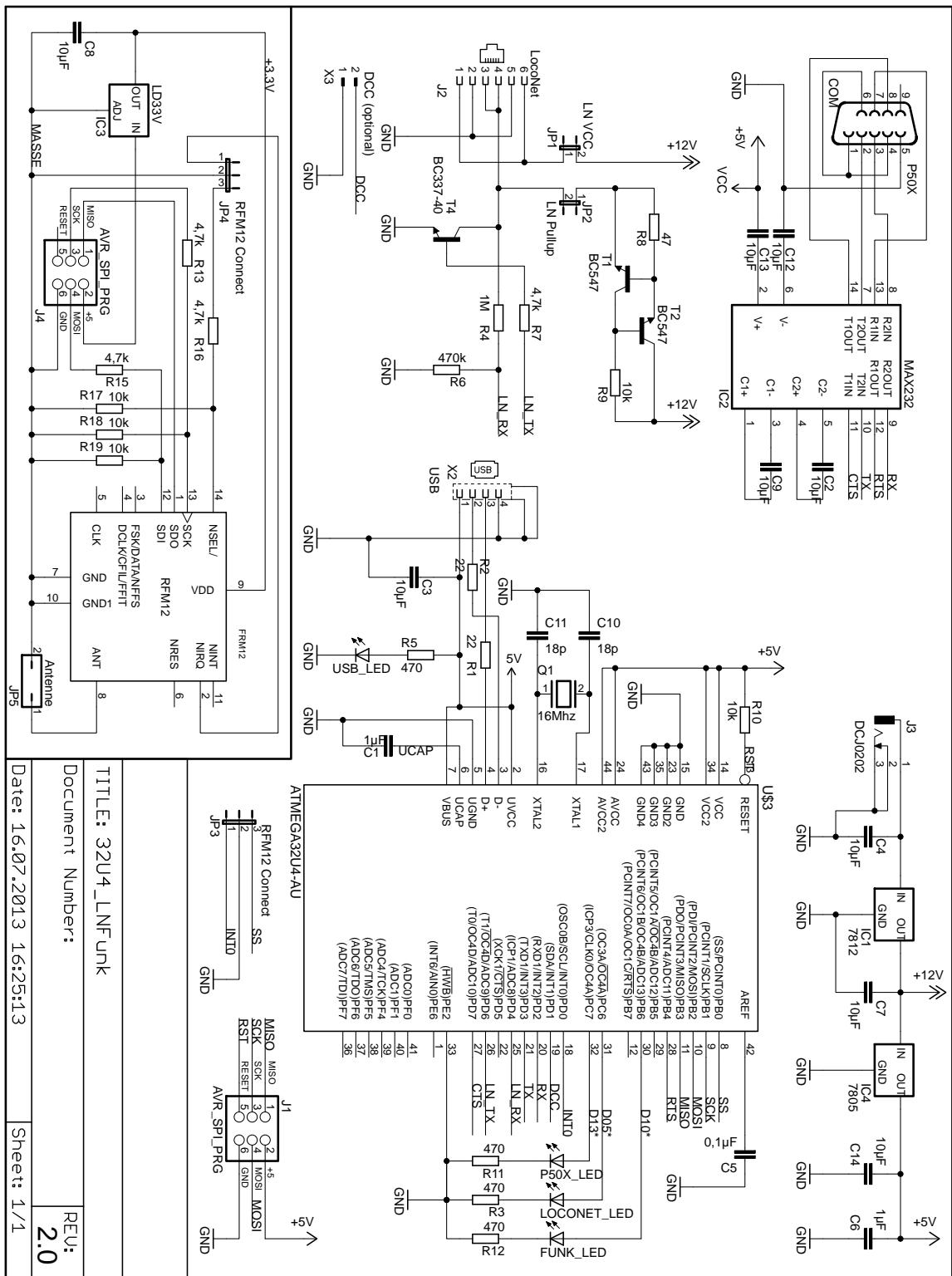


Abbildung D.2: Schaltung der Zentrale.

Anhang E

Konfigurationsvariablen des Fahrzeugdekoders

In der Tabelle E.1 sind alle Konfigurationsvariablen aufgeführt. Die Spalte “CV.Nr.” enthält die Nummer der Konfigurationsvariable. Die Default-Werte sind die Werte, die nach einem Reset oder für die allererste Inbetriebnahme voreingestellt sind.

Name der CV	CV-Nr.	Eingabewert (Defaultwert)	Erläuterungen und Hinweise
Basisadresse	1	1...111 (3)	Legt die Adresse fest, auf welche der Dekoder reagiert. Für größere Adressen muss eine lange Adresse mit CV#17 und CV#18 programmiert werden.
Startgeschwindigkeit	2	0...255 (5)	Geschwindigkeit, die bei Fahrstufe 1 an den Motor ausgegeben wird. Ein Wert “0“ entspricht keine Drehzahl, der Wert “255“ der maximalen Geschwindigkeit.
Beschleunigungsrate	3	0...255 (18)	Länge der Wartezeit, die beim Beschleunigen des Fahrzeug jeweils vor dem Hochschalten zur nächst höheren Fahrstufe vergeht. Die Wartezeit wird wie folgt berechnet: $(CV\#3) \cdot 0,01sec$.
Bremsrate	4	0...255 (5)	Länge der Wartezeit, die beim Abbremsen des Fahrzeug jeweils vor dem Herunterschalten zur nächst niedrigeren Fahrstufe vergeht. Die Wartezeit wird wie unter CV#3 beschrieben berechnet.

66 ANHANG E. KONFIGURATIONSVARIABLEN DES FAHRZEUGDEKODERS

Maximale Geschwindigkeit	5	0...255 (255)	Geschwindigkeit, die bei der höchsten Fahrstufe an den Motor ausgegeben wird. Der Wert “255“ entspricht bei Volllast 100% der maximalen Spannung.
Mittlere Geschwindigkeit	6	0...255 (128)	Ist die Geschwindigkeit die das Fahrzeug bei mittlerer Fahrstufe fährt.
Version	7	—	Nur lesbar.
Hersteller	8	—	Nur lesbar.
Reset	8	0...255	Beim Schreiben eines beliebigen Wertes werden die Einstellungen im Auslieferungszustand wieder hergestellt.
Akkuspannung	9	—	Nur lesbar und entspricht geteilt durch zehn der aktuellen Akkuspannung.
Ladezustand	10	—	Das gelesene Byte ist wie folgt zusammengesetzt: 1 = Ladung beendet 2 = (keine) Ladung 4 = Akkuspannung gering 8 = Akkuspannung kritisch
RFM12 ID	11	1...32 (1)	ID mit welcher die Funksignale akzeptiert werden.
RFM12 Gruppen-ID	12	1...250 (33)	Gruppe des Funkkanals. Diese ist für alle Dekoder und die Zentrale gleich.
RFM12 Master-ID	13	1...32 (15)	ID der Zentrale zu welcher die Rückmelddaten gesendet werden.
RFM12 Frequenz	14	1...3 (2)	Frequenz des verwendeten Funkmodul: 1 = 433MHz 2 = 868MHz 3 = 915MHz
RFM12 Sendeintervall	15	1...255 (3)	Anzahl die ein Rückmelddatenpaket maximal wiederholt an die Zentrale gemeldet wird.
RFM12 Wiederholungsrate	16	1...255 (60)	Zeit in Millisekunden, die zwischen dem erneuten Senden des Datenpaketes maximal verstreichen darf.

Erweiterte Adresse	17 18	0...39 (0) 0...255 (0)	Die erweiterte Adresse berechnet sich aus diesen zwei Registern. Beim Schreiben einer der beiden Register wird die Adresse in CV#1 gelöscht.
Akkuleerwarnung	20 21	0...7 (0) 0...255 (40)	Rückmeldeadresse berechnet aus diesen zwei Registern, welche bei Unterschreitung der (CV#29) auf aktiv gesetzt wird.
Akkuwarnung	29	0...255 (160)	Steuert die Einschaltung des Warnblinker unterhalb der eingestellten Spannung. Zur Berechnung der Spannung: $Volt = \frac{(CV\#28)\cdot 2}{100}$
BlinkerONZeit	30	0...255 (40)	Länge die der Blinker ein geschaltet ist. Die Zeit wird wie folgt berechnet: $(CV\#29) \cdot 0,01sec.$
BlinkerOFFZeit	31	0...255 (30)	Länge die der Blinker aus geschaltet ist. Die Zeit wird für CV#30 angegeben.
Funktion: Blinker links Blinker rechts	32 33	0...28 (1) (2)	Funktion F0 bis F28 mit welcher der Blinkerausgang links geschaltet wird.
Dimmen: Blinker links Blinker rechts	34 35	0...255 (255)	Helligkeit des Blinkers links.
Invertierung: Blinker links Blinker rechts	36 37	0...1 (0)	Ausgangspegel bei inaktivem Ausgang.
Glühlampeneffekt Blinker	38	0...255 (10)	Millisekunden bis der Ausgang die volle eingestellte Helligkeit erreicht.
Komfortblinker	39	0...10 (3)	Mindesblinkanzahl bei kurzer Aktivierung des Blinkers.
Dimmen Rücklicht	40	0...255 (70)	Helligkeit des Rücklichtausgangs bei aktivem Rücklicht.
Invertierung Rücklicht	41	0...1 (0)	Potential des Ausgangs bei Inaktivität.
Dimmen Bremslicht	42	0...255 (255)	Helligkeit des Rücklichtausgangs bei aktivem Bremslicht.
Bremslicht ein	43	0...255 (30)	Gibt die Fahrstufe an, unterhalb welcher das Bremslicht beim Herunterschalten sofort eingeschaltet wird.

68 ANHANG E. KONFIGURATIONSVARIABLEN DES FAHRZEUGDEKODERS

Bremslicht ein stark	44	0...255 (4)	Bei einer starken Reduzierung der Fahrgeschwindigkeit wird das Bremslicht automatisch zugeschaltet. Die stärke der Reduzierung berechnet sich wie folgt: $Fahrstufe + \frac{Fahrstufe}{(CV\#44)} < aktueller_Fahrstufe$.
Bremslicht Stand	45	0...255 (255)	Länge der Zeit, die das Bremslicht im Stand nachleuchtet. Die Zeit berechnet sich wie folgt: $(CV\#45) \cdot 0,01sec$.
Bremslicht Fahrt	46	0...255 (80)	Länge der Zeit, die das Bremslicht nach dem Herunterschalten von Fahrstufen während der Fahrt angeschaltet bleibt. Die Zeit wird dabei wie unter CV#45 beschrieben berechnet.
Zusatzfunktion Abblendlicht	47	0...28 (4)	Funktion zur Aktivschaltung des Abblendlicht vorn.
Dimmen Abblendlicht Funktion F0	48	0...255 (90)	Helligkeit des Abblendlicht vorn.
Dimmen Abblendlicht Zusatzfunktion	49	0...255 (255)	Helligkeit des Abblendlicht vorn bei aktiver Zusatzfunktion CV#47.
Invertierung Abblendlicht	50	0...1 (0)	Ausgangspegel bei inaktivem Ausgang.
Funktion Automatikabblendlicht	51	0...28 (3)	Funktion zur Aktivierung der Lichtaustomatik.
Licht ein	52	0...255 (105)	Umgebungshelligkeit die vom Dekoder gemessen wurde. Ein Wert von "255" entspricht hell und "0" dunkel.
Licht ein Verzögerung	53	0...255 (100)	Länge der Wartezeit bis das Licht beim unterschreiten von CV#52 eingeschaltet wird. Die Zeit wird wie folgt berechnet: $(CV\#53) \cdot 0,01sec$.
Licht aus	54	0...255 (145)	Umgebungshelligkeit die vom Dekoder gemessen wurde. Ein Wert von "255" entspricht hell und "0" dunkel.

Licht aus Verzögerung	55	0...255 (200)	Länge der Wartezeit bis das Licht beim überschreiten von CV#54 abgeschaltet wird. Die Wartezeit wird wie unter CV#53 berechnet.
Motor Notstop	63	0...255 (100)	Länge der Wartezeit ohne erkannte RCMC-Daten bis der Motor gestoppt wird. Die Wartezeit wird wie folgt berechnet: $(CV\#63) \cdot 2 \cdot 0,01sec.$
Power Save	64	0...255 (30)	Wartezeit ohne RCMC-Daten bis der Dekoder alle Ausgänge abschaltet. Ein Aufwecken ist sofort wieder möglich. Die Wartezeit wird für CV#64 in Sekunden angegeben.
Power Down	65	0...255 (6)	Wartezeit nach CV#54 bis der Dekoder die Hardware komplett abschaltet. Ein Aufwecken aus dem "Full Sleep" kann bis zu 8sec. dauern. Die Wartezeit berechnet sich wie folgt: $(CV\#65) \cdot 8sec.$
Eigenschaften für variable Ausgang: AUX0 AUX1 AUX2 AUX3 AUX4 AUX5 AUX6 AUX7 AUX8 AUX9	ab Adr.: 100 120 140 160 180 200 220 240 260 280	Siehe zutreffende Konfiguration der jeweiligen Eigenschaft.	Startadressen für die Konfiguration der zehn Zusatzausgänge des Fahrzeugdekoders. Im folgenden sind für jede Funktion 19 Einstellungen aufgelistet. Dadurch ist es möglich für jedem Ausgang ein individuelles Verhaltensmuster zu Programmieren.

70 ANHANG E. KONFIGURATIONSVARIABLEN DES FAHRZEUGDEKODERS

Pausenzeiten:	AUX? ¹	0...255 (0)	Zu jedem Ausgang kann hier eine individuelle Blinkfrequenz (Zeit zwischen dem Umschalten des Ausgangs) programmiert werden. Dazu besitzt jeder Ausgang zehn Zeit, die nacheinander durchlaufen werden. Nach dem Ablauf der Zeit wird der Status des Ausgangs gewechselt. Wenn kein Blinken des Ausgangs gewünscht ist, sondern nur eine einfache Schaltfunktion, müssen alle Pausenzeiten bis auf die erste auf Null gesetzt werden.
Pause0	+0		
Pause1	+1		
Pause2	+2		
Pause3	+3		
Pause4	+4		
Pause5	+5		
Pause6	+6		
Pause7	+7		
Pause8	+8		
Pause9	+9		
Dimmen des Ausgangs	AUX? +10	0...255 (255)	Helligkeit bei aktiver Funktion 1 bis Funktion 3.
Funktion 1	AUX? +11	0...28 (255)	Hauptfunktion zum Schaltung des Ausgangs.
Funktion 2	AUX? +12	0...28 (255)	Zweite Funktion um den Ausgang zu schalten.
Funktion 3	AUX? +13	0...28 (255)	Bedingungsfunktion: Der Ausgang wird nur aktiv, wenn Funktion 1 oder Funktion 2 und Funktion 3 aktiv sind.
Dimmen Funktion vier	AUX? +14	0...255 (255)	Helligkeit bei aktiver Funktion 4.
Funktion 4	AUX? +15	0...28 (255)	Extrafunktion zum unabhängigen Schalten des Ausgangs.
Ausgangsinvertierung	AUX? +16	0...1 (0)	Ausgangspegel bei inaktivem Ausgang.
Glühlampeneffekt beim Einschalten	AUX? +17	0...255 (0)	Millisekunden bis der Ausgang die volle eingestellte Helligkeit beim Einschalten erreicht.
Glühlampeneffekt beim Abschalten	AUX? +18	0...255 (0)	Millisekunden bis der Ausgang völlig abgeschaltet wird.

Tabelle E.1: Werte und Beschreibungen der CV-Konfigurationsvariablen im Fahrzeugdekomodulator.

¹AUX? steht für das jeweilige Startregister des Zusatzausgangs. Dabei besitzt jeder dieser Ausgänge 19 Registerwerte zur Konfiguration.