Entwicklung eines modularen Feldbussystems zur Steuerung einer Modellbahnanlage

Diplomarbeit von Stephan Höhrmann

Christian-Albrechts-Universität zu Kiel
Lehrstuhl für Echtzeitsysteme und Eingebettete Systeme
Prof. Dr. Reinhard von Hanxleden

1. März 2006
Betreuer: Jan Lukoschus

Diplomarbeit von Stephan Höhrmann

Veröffentlicht am 1. März 2006

Mein besonderer Dank gilt Reinhard von Hanxleden für die ungewöhnliche Aufgabenstellung und die Möglichkeit, eigenverantwortlich die Arbeit organisieren zu können. Jan Lukoschus hatte jederzeit ein offenes Ohr für Fragen und konnte einige Ideen zur Entwicklung beisteuern. Jürgen Noss, Stefan Baumgart und Jochen Koberstein haben mir viel dabei geholfen, die alte Anlage zu verstehen und darauf basierend Verbesserungen zu entwickeln. Jan Täubrich und Jan Schröder waren eine wichtige Hilfe beim Verlegen der Kabel. Schließlich bedanke ich mich noch bei Xenia für ihre Geduld, wenn meine Zeit sehr knapp gewesen ist, und ihre Hilfe bei den Korrekturen.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Im Zuge der Automatisierung sind heutzutage größere Anlagen und Maschinen oft mit einer Vielzahl von Sensoren und Aktoren ausgestattet, die von einem oder mehreren Steuergeräten kontrolliert werden. Dazu gehören industrielle Produktionsstraßen und chemische Großanlagen genauso wie Autos. Bis in die 1990er Jahre hinein wurde die Peripherie sternförmig an das Steuergerät angeschlossen, was entsprechend viele Anschlußpunkte und Leitungen erforderte. Kosten und Komplexität stiegen mit dem Automatisierungsgrad aber so stark an, daß eine Alternative dringend erforderlich wurde. Die Lösung sollten Feldbussysteme bringen, bei denen ein einfach gehaltener, meist serieller Bus alle Geräte miteinander verbindet.

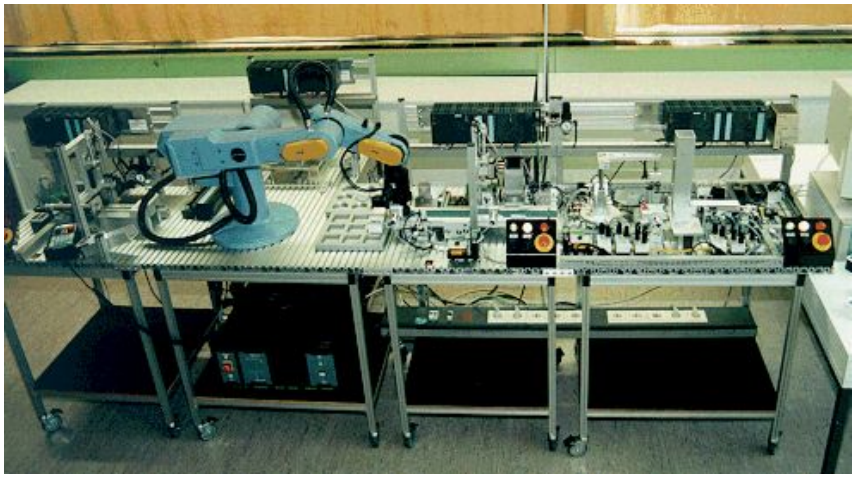


Abbildung 1.1: Feldbuslabor der TU Braunschweig (Foto der Arbeitsgruppe)

Die Vorteile liegen auf der Hand, die reduzierte Verkabelung spart Kosten, neue Komponenten lassen sich jederzeit in das System integrieren und Diagnosedaten der ganzen Anlage können an einem Punkt aufgenommen werden. Dafür erfordert die Kommunikation auf dem Bus eine aufwendige Koordination. Gleichzeitig sendende Knoten erzeugen Kollisionen, die den Bus blockieren können, und das Zeitverhalten ist ohne entsprechende Maßnahmen nicht berechenbar. Daher wurde eine Vielzahl von Feldbussen entwickelt, die verschiedene Lösungsansätze für die geschilderten Probleme bieten und zueinander in Konkurrenz stehen. Sie unterscheiden sich in der Maximallänge, dem Durchsatz, der Anzahl erlaubter Teilnehmer, den tolerierbaren Fehlern und wie weit das Zeitverhalten eines Kommunikationsmusters berechenbar ist (siehe [Santner]).

Je wichtiger die Feldbusse in der Praxis werden, desto mehr finden sie auch ihren Platz in der Forschung und Ausbildung. Oft stehen in den ingenieurwissenschaftlichen Studiengängen Labore zur Verfügung, in denen eine Anlage über ein Feldbussystem gesteuert werden kann. Abbildung 1.1 zeigt ein Labor des Lehrstuhls für Regelungstechnik an der TU Braunschweig. Bei der Planung einer Anlage oder eines Labors ist die Auswahl des Feldbusses oft der erste Schritt, da alle übrigen Komponenten mit dem passenden Interface ausgestattet sein müssen. Diese Entscheidung ist in der Regel endgültig, ein späterer Wechsel auf ein anderes System ist zu teuer und aufwendig. Damit können keine anwendungsspezifischen Vergleiche zwischen den Feldbussen durchgeführt werden, und die Ausbildung beschränkt sich auf diesen einen Bus.

1.1 Aufgabenstellung

Ein außergewöhnliches und komplexes Labor stellt die Modellbahnanlage der Informatik an der Universität Kiel dar. Mit ihren Gleisen, Signalen, Weichen, Gleiskontakten und weiterer Peripherie verfügt sie über insgesamt 245 Sensoren und Aktoren, die von einem verteilten System von 24 Knoten angesteuert werden. Die einzelnen Knoten kommunizieren über einen Feldbus untereinander sowie mit externen Rechnern und steuern so den Fahrbetrieb.

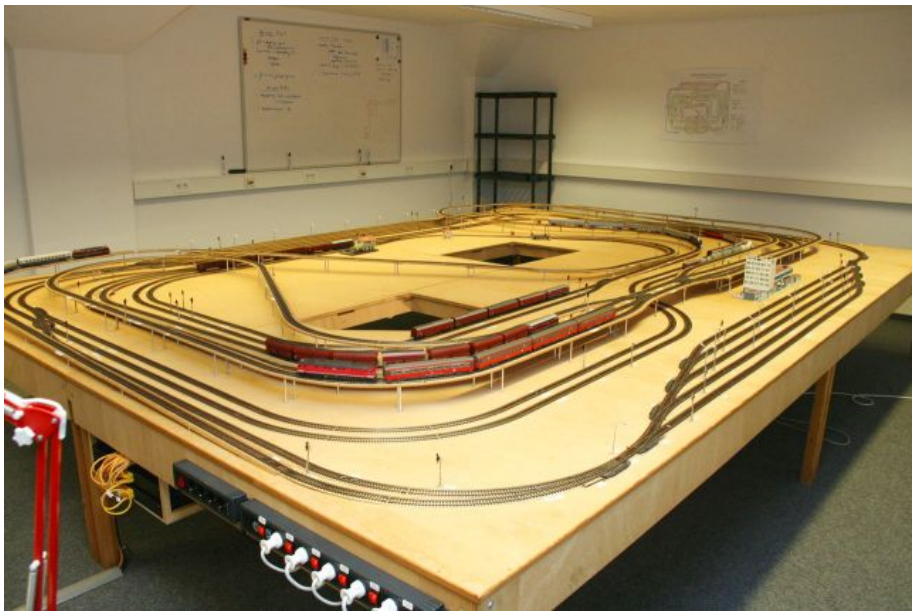


Abbildung 1.2: Die Modellbahnanlage der Informatik

Auf der Anlage kann eine ganze Reihe von Aufgaben anschaulich realisiert werden, zu denen Ressourcenverwaltung, Organisation komplexer Abläufe und auch die Kommunikation über das Feldbussystem gehören. Bis 2004 basierte die Modellbahnanlage auf dem Interbus und ließ sich auch nur über diesen betreiben. Im Rahmen dieser Arbeit sollten die Peripherie und der Feldbus entkoppelt werden, so daß mehrere Bussysteme angeschlossen und unabhängig voneinander zur Ansteuerung benutzt werden können.

Mit diesem Ziel wurde das Steuersystem der Modellbahn bestehend aus Hard- und Software von Grund auf neu zu entwickelt. Der Benutzer sollte jederzeit und ohne technische Veränderungen zwischen den eingebauten Feldbussen wechseln können. Damit bestünde die Möglichkeit, die

Eigenschaften verschiedener Entwicklungswerkzeuge und Bussysteme in einem großen Szenario unter realistischen Bedingungen zu vergleichen. Der modulare Ansatz sollte dafür sorgen, daß nahezu beliebige Steuerrechner und Feldbusse eingesetzt und miteinander kombiniert werden können. In der ersten Aufbaustufe waren PCs im PC104-Format mit Schnittstellen für Ethernet und CAN sowie TTP Powernodes mit TTP-Bus und eigenem CAN-Netzwerk vorgesehen.

1.2 Gliederung dieser Arbeit

Diese Arbeit besteht aus drei Teilen, welche die Modellbahnanlage von der zugrundeliegenden Hardware bis hin zur Organisation des Fahrbetriebs vorstellt.

Der erste Teil beschäftigt sich zunächst mit der Hardware, Kapitel 2 beschreibt die Modellbahn und die auf der Oberseite der Platte vorhandene Elektronik. Dazu gehören die Weichen, Signale, Gleiskontakte, der Fahrstrom und weitere Peripherie. In Kapitel 3 werden die Platinen zur Ansteuerung der Peripherie vorgestellt und die hier eingesetzten Techniken dokumentiert. Die Versorgungssysteme, Steuerrechner und ihre Vernetzung mit den verschiedenen Feldbussen bilden den Schwerpunkt von Kapitel 4.

Der Programmierung und den verschiedenen Schnittstellen widmet sich der zweite Teil. Zunächst erklärt Kapitel 5 das serielle Protokoll, welches zur Kommunikation zwischen Steuerrechnern und der Elektronik benutzt wird. Kapitel 6 stellt das Interface für Anwendungen vor, welches für alle Ebenen der Kommunikation entsprechende Funktionen in C bietet. Das Gegenstück für Matlab ist ihn Kapitel 7 kurz angerissen.

Der dritte Teil beschäftigt sich mit dem praktischen Einsatz der Bahn. Kapitel 8 enthält Regeln, die im Fahrbetrieb berücksichtigt werden müssen, sowie nützliche Hinweise zur Umsetzung. Daran schließt sich in Kapitel 9 das Fazit an.

Den Rest bilden die Anhänge. Alle verwendeten Schaltungen sind in Anhang A beschrieben, darauf folgen in Anhang B Tabellen, die für den Fahrbetrieb benötigt werden oder die zentrale Eigenschaften der Anlage und der Loks zusammenfassen. Anhang C zeigt alle Schritte, die zum Compilieren des Betriebssystems und der Anwendungen für die PC104-Rechner erforderlich sind. Zum Abschluß sind die Pläne der Anlage im Anhang D abgedruckt.

Teil I

Die Modellbahnanlage

Kapitel 2

Beschreibung der Bahnanlage

Beim Betrieb einer Eisenbahn treten Probleme auf, die in vieler Hinsicht typisch für nebenläufig arbeitende Systeme sind. Es stehen mit den Gleisen nur begrenzte Ressourcen zur Verfügung, die sinnvoll verwaltet und den verschiedenen Zügen exklusiv zugewiesen werden müssen. Es kommt zu Konkurrenzsituationen, wenn ein Streckenabschnitt von mehreren Zügen beansprucht wird. Diese Situationen müssen durch vorausschauende Planung jederzeit aufgelöst werden können, so daß es nie zu Verklemmungen kommt und der Fahrbetrieb lebendig bleibt. Die Entscheidungen müssen fair und eventuell unter Berücksichtigung von Prioritäten erfolgen.

Abbildung 2.1: Die Modellbahnanlage

Eine Modelleisenbahn eignet sich daher sehr gut, Probleme der nebenläufigen Programmierung anschaulich zu machen, Lösungen zu entwickeln und auf der vorhandenen Hardware zu testen. Sie kann als Demonstrationsobjekt und Versuchsfeld für die Ausbildung in der Informatik dienen. Daneben ist die Ansteuerung der (nicht digitalen) Anlage eine komplexe Aufgabe, die alle Felder von systemnaher Programmierung bis hin zur Entwicklung von Benutzeroberflächen abdeckt. Basierend auf dieser Idee begann der Lehrstuhl für Rechnerorganisation der CAU im Jahr 1995 damit, eine Modellbahnanlage im Maßstab H0 zur Unterstützung der Lehre sowie für Praktika

aufzubauen. Das Vorbild für die Streckenführung war der kanadische Kicking Horse Pass, nach dem die Anlage auch benannt ist. Nach mehreren Umbauten ist sie heute 18 Quadratmeter groß und umfaßt ein Schienennetz mit einer Länge von etwa 127 Metern, auf dem sich mehr als acht Züge gleichzeitig bewegen können. Dieses Kapitel beschreibt die technischen Details der Anlage, soweit sie für den Betrieb wichtig sind. Dazu gehören neben dem Aufbau des Streckennetzes auch Beschreibungen der in der Anlage verbauten Modellbahnelektronik, also der Signale, Gleise, Weichen, Gleiskontakte, der Beleuchtung und schließlich des Bahnüberganges.

2.1 Kicking Horse Pass

Die Vorlage für die Anlage ist ein Gebirgspaß im kanadischen Teil der Rocky Mountains. Er liegt in dem Grenzgebiet zwischen Alberta und British Columbia, nahe den Nationalparks Yoho und Banff. Im Jahr 1858 erkundete eine europäische Expedition unter Leitung von John Palliser die Gegend, James Hector aus dieser Gruppe war es schließlich, der den Paß als erster durchquerte. Unterwegs wurde er von seinem Pferd getreten, dieser Tatsache verdanken der Paß und der durch ihn laufende Fluß ihre Namen. Seit 1885 verläuft die transkontinentale Eisenbahnstrecke der Canadian Pacific Railway durch den Paß, die hier auch mit 1627 Metern ihren höchsten Punkt und den steilsten Anstieg findet.

Abbildung 2.2: Kicking Horse Pass (Foto von John Cletheroe)

Die Bahnlinie verläuft eingleisig von Lake Louise im Osten bis Field im Westen. Zwischen Field und der Paßhöhe liegt der “Big Hill”, auf den die 1885 gebaute Strecke mit einer Steigung von bis zu 4,5 Prozent heraufführte. Dieser Abschnitt war die steilste Haupteisenbahnstrecke in ganz Nordamerika und ließ sich deshalb nur durch den Einsatz zusätzlicher Lokomotiven bewältigen. Die resultierenden Betriebskosten und die häufigen Unfälle zwangen den Betreiber bereits nach wenigen Jahren zu einer Veränderung der Streckenführung.

Seit dem Jahr 1911 ersetzen zwei spiralförmige Tunnel die alte Strecke. Sie verlängern den Weg um 12 Kilometer und reduzieren die Steigung dadurch auf vertretbare 2,2 Prozent. Trotzdem ist die Steigung noch erheblich, bis heute ist jede Durchfahrt eines Zuges vom Dröhnen der

Motoren beziehungsweise dem Pfeifen der Bremsen begleitet. Entlang der ursprünglichen Strecke verläuft mittlerweile der 1962 eröffnete Trans Canada Highway CA-1. Ein Aussichtspunkt an der Straße erlaubt den Blick auf große Teile der Strecke inklusive der beiden Tunnel (die hier zusammengefaßten Informationen stammen aus [Kicking]).

2.2 Entwicklungsgeschichte

Die erste Idee zum Bau einer Modellbahnanlage nach dem Vorbild des Kicking Horse Passes hatte Professor Werner Kluge, damals Inhaber des Lehrstuhles für Rechnerorganisation, im Jahr 1995. Zwei Jahre später, nach gut einem Jahr Bauzeit, war die Modellbahn einsatzbereit. Der Streckenverlauf berücksichtigt viele Eigenheiten des Vorbildes, die spiralförmigen Tunnel sind in der Mitte der Anlage deutlich zu erkennen. Die Gesamtlänge der Gleise betrug 140 Meter.

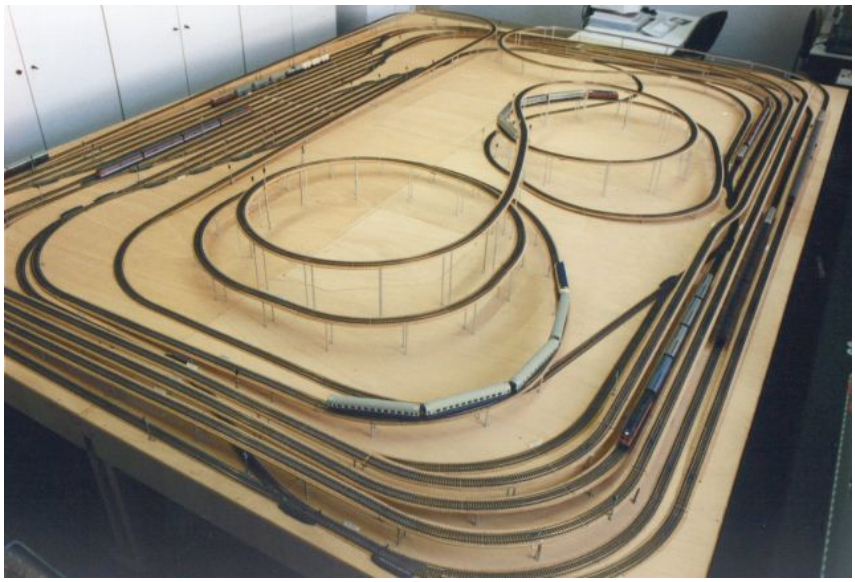


Abbildung 2.3: Erste Generation der Bahn von 1997 (Foto der Arbeitsgruppe)

Die Elektronik zur Steuerung wurde von Jürgen Noss entwickelt. Dabei handelte es sich um ein zentral gelegenes System aus drei großen Platinen, die sternförmig mit allen Sensoren und Aktoren der Bahn verbunden war. Eine Workstation von Sun war über ihren Parallelport mit der Elektronik verbunden und erlaubte so die Steuerung durch ein Programm. Ironischerweise zeigten sich an dem Modell ähnliche Probleme wie bei dem großen Vorbild in Kanada. Auf den steilen und engen Kurven des Passes kam es gelegentlich zu Unfällen, wenn ein Zug die Steigung nicht bewältigen konnte oder entgleist und in die Tiefe gestürzt ist. Dazu kamen Probleme mit der Elektronik, die sich als empfindlich für Störungen herausstellte. Als der Lehrstuhl aus der Innenstadt auf den Hauptcampus umzog, lag die Anlage anschließend einige Zeit brach.

2.2.1 Zweite Generation

Die Studenten Jochen Koberstein und Oliver Schmitz machten sich 1999 an den Wiederaufbau der Bahn, was als Freizeitprojekt mit Unterstützung durch den Lehrstuhl begann und in einer gemeinsam verfaßten Diplomarbeit endete. Im Zuge der Arbeiten wurde zuerst das Streckennetz

runderneuert, insbesondere wurden die fehleranfälligen Steilpässe durch Brücken mit niedriger Steigung und weniger Kurven ersetzt. Die Anlage ließ sich jetzt problemlos befahren, dafür hat sich die Gesamtstreckenlänge auf rund 127 Meter verringert. In einem zweiten Schritt wurde die gesamte Elektronik ausgetauscht. Ein komplett neuentwickeltes System aus 24 durch einen Interbus verbundenen Platinen ersetzte die monolithische Ansteuerung. Die räumliche Trennung der Komponenten und die vereinfachte Verkabelung konnten die elektrischen Störungen deutlich reduzieren. Angesteuert wurde das System über einen Master, der Pakete zwischen der seriellen Schnittstelle der Sun und dem Interbus vermittelte.

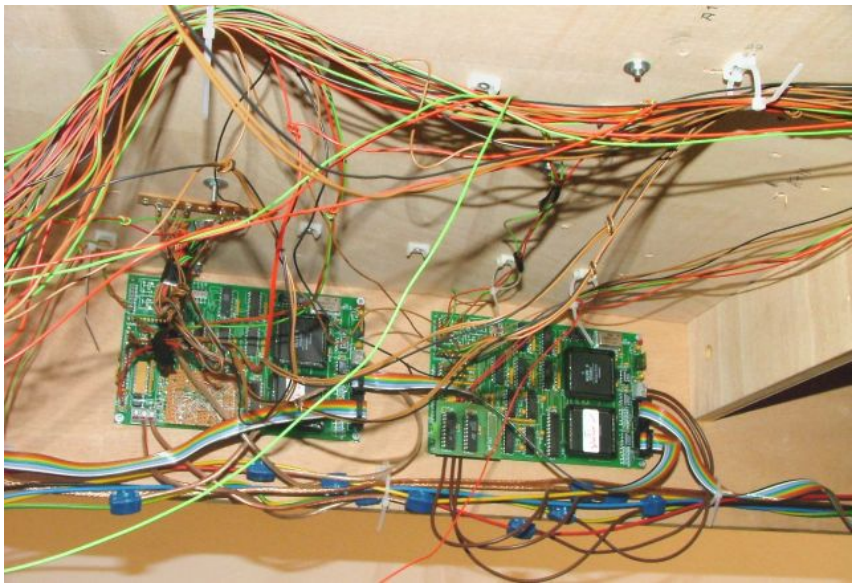


Abbildung 2.4: Zwei Interbus-Steuerplatinen mit Verkabelung

Der Umbau war im Jahr 2001 nach zweieinhalb Jahren schließlich beendet und eine neue Library zur Programmierung der Anlage in C++ geschrieben. Das von Jochen Koberstein und Oliver Schmitz entwickelte Railway Control Center erlaubte es dem Benutzer, einen einfachen Fahrplan je Zug einzugeben, der dann auf der Anlage ausgeführt wurde. Verklemmungen werden dabei verhindert und fehlerhafte Sensordaten durch einfache Heuristiken kompensiert. Das Programm basiert auf Petrinetzen der Anlage, wie sie in [Hielscher] und [Kluge] entwickelt wurden. Alle Details der Steuerung faßt [Koberstein] zusammen und geht dabei auf die Implementierung und die Ansteuerung der Hardware ein.

Neben allen Verbesserungen blieben aber noch Probleme bestehen, beispielsweise funktionierte die Ansteuerung der Wendeschleifen nicht zuverlässig und die Sensoren in der Anlage lieferten gelegentlich noch falsche Werte. Das größte Problem war aber die Verkabelung, die mechanisch empfindlich und nur schwer nachvollziehbar war. Dadurch gestalteten sich Wartungsarbeiten wie die Reparatur eines abgerissenen Kabels sehr schwierig.

2.2.2 Dritte Generation

Der Lehrstuhl Echtzeitsysteme und Eingebettete Systeme übernahm die Modellbahn 2002 und bot bis zum Wintersemester 2003/2004 zwei Praktikum in den alten Räumen an. Anschließend wurde die Anlage zu ihrem neuen Standort am Heinrich-Hecht-Platz transportiert. Für den zukünftigen Einsatz war geplant, den Fokus vom Fahrbetrieb auf die Kommunikation innerhalb

der Steuerung zu erweitern. Der für diese Aufgabe bislang genutzte Interbus sollte durch ein System ersetzt werden, das unterschiedliche Feldbusse anbindet und eine Steuerung über jeden von ihnen erlaubt. Damit stünde ein Labor zur Verfügung, in dem Entwicklungswerkzeuge aus der Industrie an einem relativ komplexen System ausprobiert werden können. Als sich dann herausstellte, daß die alte Elektronik den Transport nicht unbeschadet überstanden hatte, war die Entscheidung für einen erneuten Austausch der gesamten Hardware gefallen, und im März 2005 begannen die Arbeiten an dieser Diplomarbeit.

Die neue Steuerung basierte auf den Erfahrungen der Vergangenheit, bietet darüber hinaus aber einige Verbesserungen und neue Funktionen. Die gesamte Verkabelung wurde ausgetauscht und ist jetzt robuster, gut dokumentiert und leicht zu warten. Jede Steuerplatine ist mit einem TTP-Knoten sowie mit einem unter Linux laufenden PC104-Rechner mit CAN-Interface und Ethernet-Anschluß verbunden. Darüber hinaus können noch zwei weitere Rechner angeschlossen und so beliebige Bussysteme integriert werden. Die Streckenführung hingegen ist fast unverändert, es ist lediglich ein Bahnübergang im Kicking Horse Pass dazugekommen.

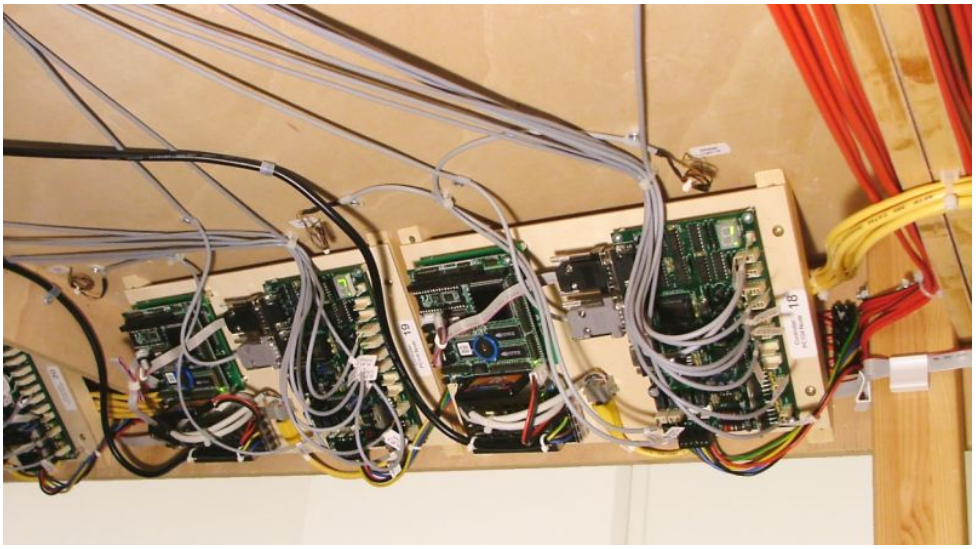


Abbildung 2.5: Zwei von 24 Steuermodulen der aktuellen Bahnanlage

Zum Anfang des Wintersemesters 2005/2006 startete nach gerade einmal vier Monaten Bauzeit das nächste Praktikum an der Modellbahn. Für die Steuerung sollte diesmal das TTP-Netzwerk benutzt werden, die Entwicklung der gesamten Software sollte grafisch mit Hilfe von Matlab und Erweiterungen der Firma TTTech geschehen.

2.3 Das Streckennetz

In der Anlage sind auf einer Grundfläche von 4,80 mal 3,70 Metern rund 127 Meter Gleis in zwei Ebenen untergebracht. Das Streckennetz besteht im Wesentlichen aus drei großen Kreisen, Inner Circle (IC), Outer Circle (OC) sowie dem Kicking Horse Pass (KH). Alle Kreise sind untereinander über Wendeschleifen und Kreuzungsweichen verbunden. Den Hauptteil der Strecke machen Inner und Outer Circle aus, die parallel in Form einer verschlungenen Acht durch die Anlage verlaufen und jeweils über einen eigenen Bahnhof (ST) mit drei Gleisen verfügen. Beide Kreise werden üblicherweise nur in eine Richtung befahren werden, Züge auf dem Outer Circle

bewegen sich im Uhrzeigersinn, im Inner Circle dagegen. Über zwei Wendeschleifen können Züge von einem der Kreise in den jeweils anderen wechseln, eine führt vom Inner in den Outer Circle (IO), die andere entsprechend vom Outer in den Inner Circle (OI). Sie sind in einem Kreuzungsbereich an die beiden Kreise angeschlossen, der Junction (JCT) heißt, die normalen Streckenabschnitt werden als Lane (LN) bezeichnet. Das idealisierte Streckennetz der Anlage ist in Abbildung 2.6 dargestellt, Anhang D enthält einen Plan mit allen Brücken und Kurven.

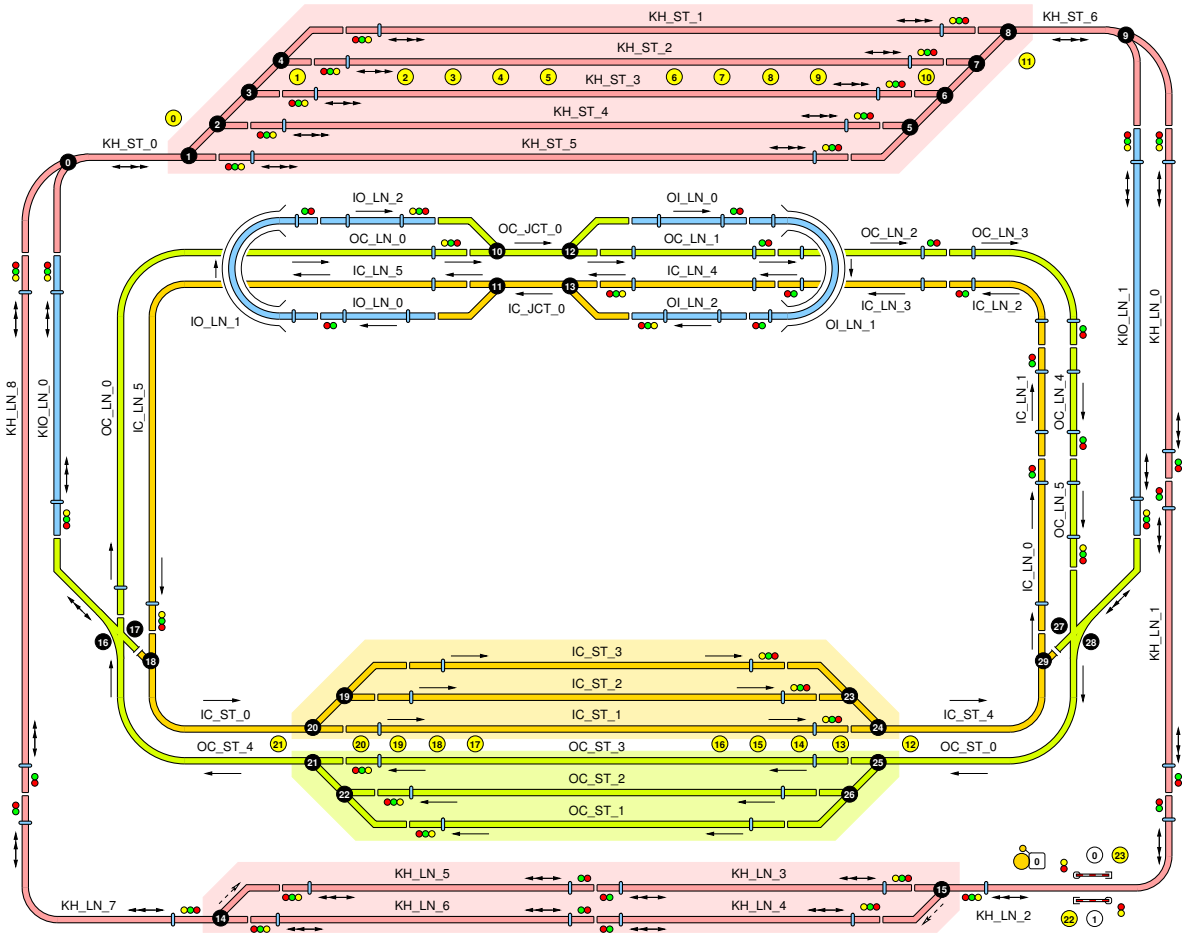


Abbildung 2.6: Vereinfachtes Streckenlayout der Modellbahn

Den Kern der Anlage bildet der Kicking Horse Pass. Er besteht zunächst aus einem Bahnhof mit fünf Gleisen, von dem aus die Züge nach beiden Seiten in die bidirektionale Paßstrecke einfahren können. In der Mitte der Strecke existiert ein kleiner zweispuriger Abschnitt, der es aus verschiedenen Richtungen ankommenden Zügen ermöglicht, aneinander vorbei zu fahren.

Zwei Verbindungsstücke (KIO) führen vom Kicking Horse Pass auf Kreuzungsweichen, die einen Wechsel zwischen allen drei Kreisen erlauben. Auf der linken Seite können Züge vom Outer Circle in den Paß beziehungsweise von dort in den Inner Circle gelangen, rechts ist es genau anders herum. Fahrten durch den Paß beginnen und enden zwangsläufig im Bahnhof, auch wenn die Züge aus den anderen Kreisen kommen oder dorthin wollen. Die Fahrtrichtung im Paßbahnhof hängt davon ab, aus welchen der anderen Kreise die Züge kommen. Die drei Hauptkreise sind mit 24,6, 23,9 und 26,7 Metern etwa gleich lang, für eine vollständige Runde durch einen Kreis benötigt ein mit Höchstgeschwindigkeit fahrender Zug etwa eine Minute.

2.3.1 Blöcke

Das Schienennetz ist in 48 elektrisch gegeneinander isolierte Blöcke mit Längen zwischen 51 und 459 Zentimetern unterteilt. Davon entfallen je 12 Stück auf Inner und Outer Circle, 17 auf den Kicking Horse Pass und 8 auf die Wendeschleifen und Verbindungen. Der Fahrstrom jedes einzelnen Blocks kann unabhängig von den anderen eingestellt werden, so daß ein einzelner Zug innerhalb des Blockes frei steuerbar ist. Enthält ein Block Weichen oder Kreuzungen, gehören diese immer vollständig zu dem jeweiligen Block, die Trennung vom Nachbarblock erfolgt erst in den wegführenden Gleisen.

Fast alle Blöcke enthalten einen ausgezeichneten Haltebereich, in dem Züge stehen bleiben dürfen, um beispielsweise auf das Freiwerden des Nachfolgeblocks zu warten. Der Bereich ist mit Kontakten an den Enden ausgestattet. Mit ihrer Hilfe kann das System erkennen, wann ein Zug in den Haltebereich eintritt, komplett darin enthalten ist, wieder ausfährt und ihn komplett verlassen hat. Als visuelle Kontrolle sind Signale an den Grenzen aufgestellt, sie zeigen dem Zug an, ob den Block verlassen darf oder nicht.

Abbildung 2.7: Blick auf den Junction-Bereich

Eine Ausnahme von diesem Schema stellen insgesamt 8 Blöcke an den Bahnhofszufahrten und im Junction-Bereich dar. An diesen Stellen ist nicht genügend Platz frei, um einen ganzen Zug aufzunehmen, oder ein dort haltender Zug würde viele benachbarte Gleise blockieren. Daher ist ein Anhalten in diesen sogenannten Interblocks nicht erlaubt, ein Zug muß sie ohne Zwischenhalt durchqueren. Dementsprechend gibt es dort auch keine Kontakte und Signale.

2.3.2 Adressierung der Komponenten

Viele Elemente im Streckenplan sind bereits mit Namen und Nummern beschriftet, die genau ihrer Adressierung entsprechen. Dazu gehören die Blöcke, Weichen sowie die Beleuchtung. Die übrigen werden nach einem einfachen Schema bezeichnet.

- Die Namen der Gleise bestehen aus drei Komponenten, dem Namen des Ringes, dem Blocktyp und einer fortlaufenden Nummer. Letztere wird ab 0 in Fahrtrichtung oder bei Bahnhöfen zur Anlagenmitte hin gezählt.

- Weichen tragen im Plan fortlaufende Nummern, sie sind im Ruhezustand immer so gestellt, daß ein Zug sie auf möglichst geradem Weg passiert. Bei den Kreuzungsweichen ist die Situation etwas komplizierter, ihre Einstellungen werden in 2.5.4 beschrieben.
- Beim Bahnübergang beziehen sich die Zahlen in den weißen Kreisen auf die Schranke, den Schrankensensor und das Signal auf der jeweiligen Seite.
- Kontakte und Signale sind in allen Blöcken entlang der Hauptfahrtrichtung ab 0 numeriert. In unidirektionalen Blöcken ist Signal 0 nicht vorhanden, das andere trägt die Nummer 1.

Die Fahrtrichtung der Züge kann direkt dem Plan entnommen werden. In unidirektionalen Blöcken bewegt ein vorwärts fahrender Zug sich in Pfeilrichtung, bei bidirektionalen Blöcken in Richtung des Doppelpfeiles.

2.4 Fahrstrom

Die Züge werden über die Gleise mit einer Spannung von 12 bis 14 Volt versorgt, dabei ist je eine Schiene mit einem Pol der Spannungsversorgung verbunden. Metallräder auf beiden Seiten nehmen den Strom auf und leiten ihn über Schleifkontakte aus Kupferblech zum Motor und der Beleuchtung der Lokomotive. Andere Radpaare und Achsen dürfen keine Verbindungen zwischen beiden Gleisen herstellen, damit der Fahrstrom nicht kurzgeschlossen wird.

Abbildung 2.8: Eine Lokomotive auf dem Weg durch den Inner Circle

Die verschiedenen Lokomotiven der Modellbahn unterscheiden sich außer in den Eigenschaften des eingebauten Motors noch darin, wie viele Räder zur Stromaufnahme benutzt werden wie diese angeordnet sind. Die am häufigsten vorkommenden Dieselloks, von denen eine in Abbildung 2.8 zu sehen ist, benutzen beispielsweise alle Räder. Sie verfügen damit über viele unabhängige Verbindungen zu beiden Schienen. Die Lok mit der Nummer 10 beschränkt sich auf je zwei Räder an gegenüberliegenden Enden und ist so anfällig für Kontaktprobleme, zudem kann sie sich auf einer Blockgrenze liegen bleiben, wenn die Abnehmer in verschiedenen Blöcken liegen.

2.4.1 Richtung und Geschwindigkeit

Die Fahrtrichtung eines Zuges hängt davon ab, welche Polarität die an den Gleisen anliegende Spannung hat. Dies ist unabhängig davon, wie herum der Zug auf dem Gleis steht. Wird ein vorwärts fahrender Zug auf einem Gleis um 180 Grad gedreht, fährt er durch die vertauschte Polarität rückwärts. Durch dieses doppelte Umdrehen fährt der Zug effektiv in die selbe Richtung wie vorher. Bei den Modellbahnen gilt die Konvention, daß ein Zug vorwärts fährt, wenn auf dem in Fahrtrichtung rechten Gleis die positivere Spannung anliegt. Daher ist es möglich, jedem Gleis eine feste Richtung ohne Berücksichtigung des Zuges zuzuordnen.

Die Geschwindigkeit kann über die Spannung beeinflußt werden, einfacher ist jedoch der Weg über Pulsweitenmodulation. Dabei schaltet ein digitaler Ausgangstreiber die Spannung für das Gleis im schnellen Wechsel ein und aus. Je länger der Strom im zeitlichen Mittel eingeschaltet ist, desto schneller dreht der Motor. Die resultierende Geschwindigkeit hängt in jedem Fall von den Eigenschaften des jeweiligen Motors und von der Belastung ab, mehr Waggons oder das Befahren einer Steigung lassen den Zug etwas langsamer werden.

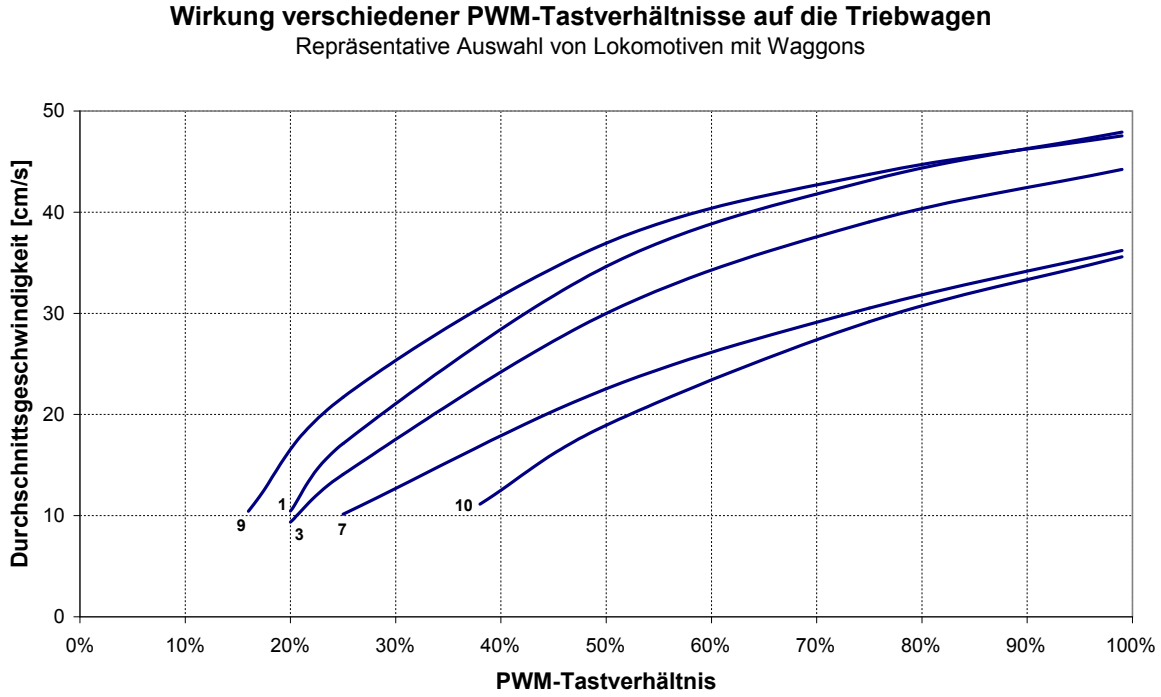


Abbildung 2.9: Laufeigenschaften verschiedener Lokomotiven

Das Diagramm in Abbildung 2.9 zeigt, welche Durchschnittsgeschwindigkeiten die verschiedenen Lokomotiven auf der Bahn erreichen. Sie fuhren mit unterschiedlichen PWM-Einstellungen und den Kohlewaggons durch den Outer Circle, aus der Streckenlänge und der Fahrzeit wurde die Geschwindigkeit berechnet. Jede Kurve deckt den sinnvoll nutzbaren Bereich der PWM ab, bei einem geringeren Tastverhältnis bewältigen die Züge die Steigungen nicht mehr. Die Nummern entsprechen der Zugaufschrift, dabei ist Nummer 9 repräsentativ für alle Loks dieses Typs. Die vollständigen Meßdaten sind in Anhang B.1 enthalten.

Tabelle 2.1 greift einige Daten nochmals auf, außerdem sind in ihr die Anzahl der Räder mit Stromabnehmern sowie der Stromverbrauch der einzelnen Loks aufgeführt. Die Ströme wurden im Leerlauf ohne Gleiskontakt und beziehungsweise mit blockierten Rädern gemessen.

Lokomotive		Stromaufnahme		Geschwindigkeit	
Nummer	**Abnehmer**	**Minimal**	**Maximal**	**50% PWM**	**Maximum**
1	12	0,6 A	1,3 A	30,9 cm/s	42,8 cm/s
2	8	0,5 A	1,8 A	32,5 cm/s	40,7 cm/s
3	8	0,5 A	1,5 A	26,8 cm/s	39,5 cm/s
4	8	0,5 A	1,4 A	33,1 cm/s	40,7 cm/s
5	8	0,5 A	1,5 A	32,3 cm/s	43,7 cm/s
6	8	0,6 A	1,3 A	33,1 cm/s	42,6 cm/s
7	8	0,5 A	0,9 A	20,1 cm/s	32,4 cm/s
8	8	0,4 A	1,7 A	33,2 cm/s	42,6 cm/s
9	8	0,5 A	1,4 A	33,0 cm/s	42,5 cm/s
10	4	0,4 A	0,9 A	16,9 cm/s	31,8 cm/s

Tabelle 2.1: Elektrische Eigenschaften der Lokomotiven

2.4.2 Blockgrenzen

Wenn die Steuerung einen Zug in Bewegung setzen will, muß sie eine entsprechende Spannung an die Gleise legen. Da alle Blöcke der Bahn nach diesem Prinzip funktionieren, darf es keine elektrische Verbindungen zwischen ihnen geben. Sie würden zu einer gegenseiten Beeinflussung oder zu Kurzschlüssen führen. Aus diesem Grund werden an den Blockgrenzen statt der üblichen Schienenverbinder aus Edelstahl oder Kupfer welche aus Kunststoff benutzt, in Abbildung 2.10 ist so eine Trennstelle zu sehen. Das Block links im Bild ist mit einer Spannungsquelle verbunden (angedeutet durch die Farben Blau und Rot), die Schienen rechts sind stromlos.

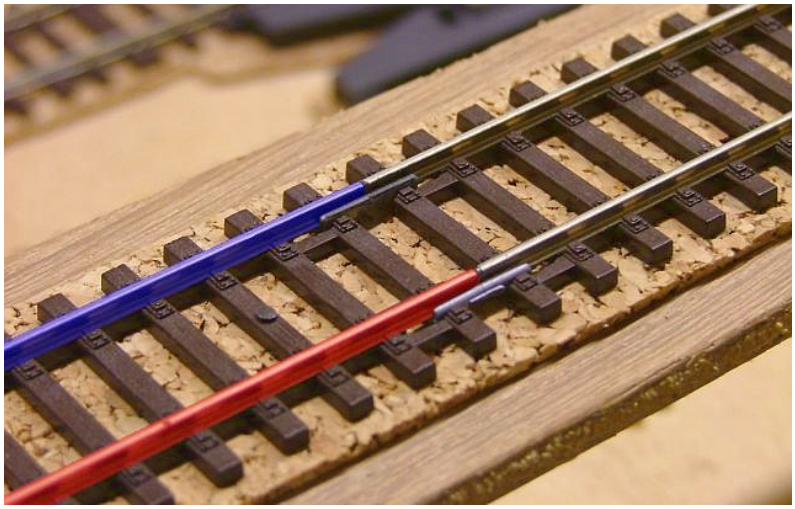


Abbildung 2.10: Blockgrenze im Kicking Horse Pass

Wechselt die Lokomotive von einem Block in den nächsten, können ihre Stromabnehmer für eine kurze Zeit in unterschiedlichen Blöcken liegen. Solange dies der Fall ist, ändert sich seine Geschwindigkeit abhängig der Position der Stromabnehmer und dem Phasenunterschied der beiden PWM-Generatoren. Messungen an den Stromleitungen zu den Gleisen liefern während dieser Zeit in beiden Blöcken falsche Daten.

2.5 Weichen

In der Anlage sind insgesamt 28 Weichen installiert, über die Züge zwischen den verschiedenen Streckenabschnitten wechseln können. Weichen funktionieren meistens nach dem selben Prinzip, von einem gerade verlaufenden Hauptgleis zweigt ein Nebengleis ab. Bewegliche Metallzungen im Inneren der Weiche leiten einen einfahrenden Zug auf den gewünschten Gleisabschnitt. Ein Weichenantrieb bewegt die Zungen in die dafür nötige Stellung, dies kann manuell erfolgen, in der Anlage übernehmen insgesamt 30 elektrisch steuerbare Antriebe diese Aufgabe.

Abbildung 2.11: Weichengruppe am Eingang der Bahnhöfe

Die Antriebe kennen zwei Einstellungen, im Ruhezustand sind die Zungen für eine Weiterfahrt auf dem Hauptgleis eingestellt, für das Nebengleis muß der Antrieb aktiv umgestellt werden. Weichen werden danach unterschieden, in welche Richtung der Zug vom Hauptgleis der Zug abzweigen kann. Liegt das Nebengleis links vom Hauptgleis, spricht man von einer linken Weiche, entsprechendes gilt für die rechte Seite.

In der Bahnanlage sind neben den normalen Weichen noch zwei Varianten der beschriebenen Konstruktion verbaut. Die Bogenweichen in den Zufahrten zum Kicking Horse Pass Bahnhof sind länger als normale Weichen, außerdem ist bei ihnen auch das Hauptgleis leicht gebogen. Davon abgesehen funktionieren sie wie beschrieben. Ganz anders arbeiten die Kreuzungsweichen, sie können in mehreren Richtungen überquert werden und verfügen über zwei Antriebe. Ihre Funktionsweise ist in Abschnitt 2.5.4 beschrieben.

2.5.1 Antriebe

Eine Weiche benötigt einen passenden Antrieb, beispielsweise gehört zu einer linken Weiche auch ein linker Antrieb. Dieser besteht dann hauptsächlich aus zwei Spulen, in denen ein beweglicher Metallkern gelagert ist. Fließt ein ausreichend hoher Strom durch eine der beiden Spulen, zieht sie den Metallkern in sich hinein und bewegt eine mit dem Kern verbundene Wippe, die alle weiteren Aktionen auslöst. Insbesondere drückt ein mit ihr verbundener Federdraht auf den mechanischen Teil der Weiche und stellt so die Weichenzungen.

Am anderen Ende hält eine Feder die Wippe in ihrer Position, gleichzeitig betätigt sie einen Umschalter, die sogenannte Endabschaltung des Antriebes. Ein Anschluß jeder Spule ist über ein rotes beziehungsweise grünes Kabel nach außen geführt. Die anderen Anschlüsse sind mit der Endabschaltung verbunden, der Umschalter verbindet einen von ihnen mit dem schwarzen Kabel. Befindet sich die Weiche in der Ruhestellung, kann die rechte Spule über den roten und schwarzen Draht angesteuert werden, der Stromkreis der anderen Spule ist unterbrochen. Sobald die Weiche in die neue Stellung umgesprungen ist, vertauscht die Endabschaltung die beiden Rollen. Jetzt kann die linke Spule über den grünen und schwarzen Draht angesprochen werden, während der Stromkreis der rechten Spule unterbrochen ist. Auf diese Weise muß die Ansteuerung nur aus einer Spannungsquelle und einem Umschalter bestehen.

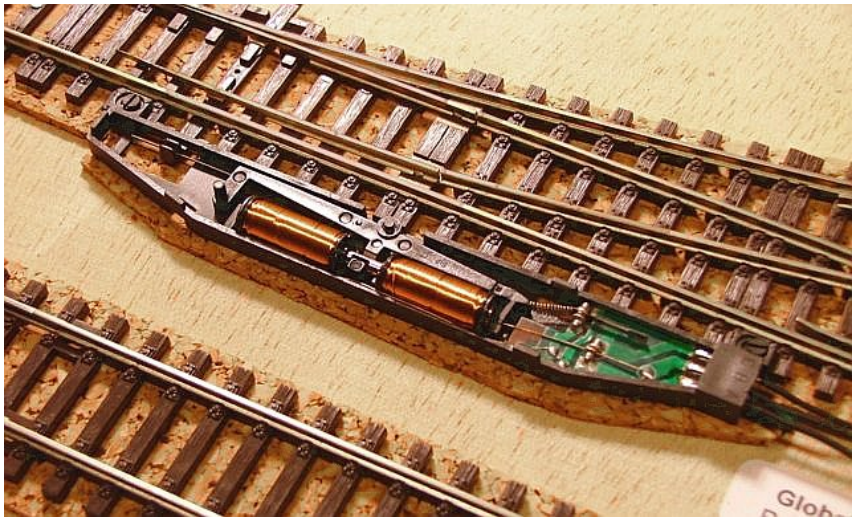


Abbildung 2.12: Elektrischer Antrieb an einer Weiche

Zuletzt betätigt die Wippe noch einen weiteren Umschalter, der aus einem Federdraht und zwei Kontakten in der Nähe der Stromzufuhr besteht. Die drei Anschlüsse sind über die Buchsen A bis C nach außen geführt. Der Mittelkontakt B wird mit A verbunden, wenn der Antrieb die Weiche auf das Hauptgleis eingestellt hat, für das Nebengleis wird B mit C verbunden. Diese Vorrichtung ist dafür konzipiert, die Herzstücke der Weiche mit Strom der richtigen Polarität zu versorgen, dazu mehr in Abschnitt 2.5.3.

2.5.2 Elektrische Eigenschaften

Die Spulen werden mit einer Gleichspannung von mindestens 12 bis 14 Volt angesteuert, häufig wird auch Wechselspannung in der selben Höhe benutzt. Der ohmsche Widerstand einer Spule liegt bei 17 Ohm, die Induktivität mit Metallkern bei grob 5 mH, was beim Betrieb mit 50 Hertz Wechselspannung die Impedanz um rund 1,5 Ohm erhöht. Im gesamten Betriebsbereich fließt somit ein Strom von 600 bis 800 Milliampere, ein Schaltvorgang dauert etwa 10 Millisekunden und verbraucht eine Ladung von 2,5 Millicoulomb bei 12 Volt.

Die Endabschaltung erzeugt beim Unterbrechen des Stromkreises einen kräftigen Abrißfunken, der elektromagnetische Störungen in der Versorgungsspannung sowie benachbarten Kabeln und Schaltungen hervorrufen kann. Mit dem Antrieb verbundene Meßgeräte zeigen eventuell in dem betreffenden Zeitraum falsche Daten an, aber auch in der Nähe befindliche Logikschaltungen

können Fehlfunktionen aufweisen. Dies war einer der Gründe für die Probleme im ersten Aufbau der Bahn. Die Störungen können durch extern angeschlossene Kondensatoren und Freilaufdioden abgeschwächt werden, es empfiehlt sich trotzdem, die Antriebe über eine galvanisch getrennte Stromquelle zu betreiben und die Leitungen von und zu den Antrieben mit möglichst großem Abstand von anderen Kabeln und Bauteilen zu verlegen.

2.5.3 Herzstücke

Wenn ein Zug eine Weiche passiert, legt er dabei einen ganz bestimmten Weg über mehrere getrennte Gleisabschnitte innerhalb der Weiche zurück. Dabei muß sichergestellt sein, daß alle Abschnitte mit Strom versorgt sind und die korrekte Polarität besitzen, sonst könnte der Zug auf dem Abschnitt stehen bleiben oder mit seinen Rädern einen Kurzschluß verursachen. Die meisten dieser Abschnitte werden in den Weichen fest untereinander und mit den nach außen führenden Schienen verbunden. Die einzige Ausnahme bildet das sogenannte Herzstück, das in beiden möglichen Wegen enthalten ist.

Wie in Abbildung 2.13 deutlich zu erkennen ist, darf das gelb markierte Herzstück nicht fest mit einem der beiden Gleise (rot oder blau) verbunden werden. Dies würde abhängig von der Fahrtrichtung des Zuges zu einem Kurzschluß führen. Stattdessen muß das Herzstück passend zur Einstellung der Weiche mit den Gleisen verbunden werden, im obigen Beispiel mit der blauen Schiene für das Hauptgleis und mit der roten für das Nebengleis. Dieser Vorgang heißt Polarisation des Herzstückes und wird von dem Weichenantrieb übernommen, wenn dieser mit den entsprechenden Anschlüssen im Gleisbett verbunden ist.

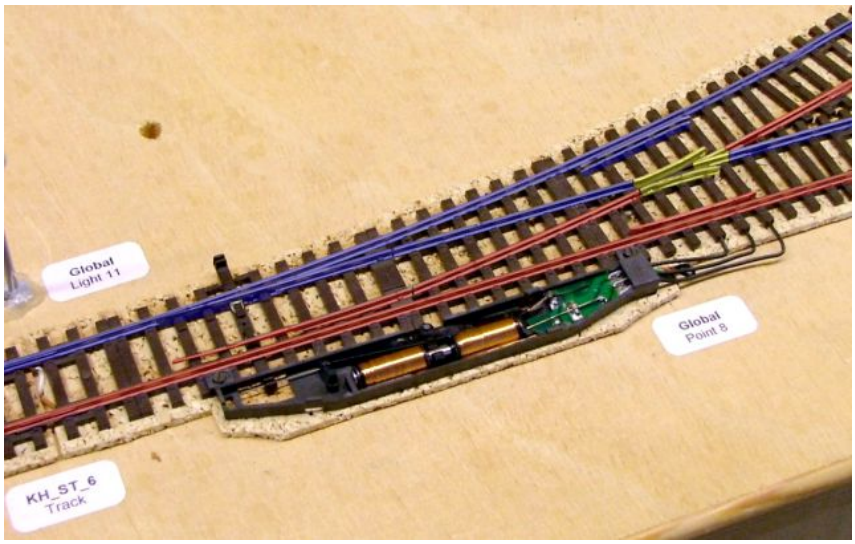


Abbildung 2.13: Elektrische Verbindungen innerhalb einer Weiche

Durch die Polarisation der Herzstücke muß eine Weiche auch dann richtig gestellt sein, wenn dies aus rein mechanischen Gründen nicht nötig wäre. In Abbildung 2.13 könnte ein von oben rechts kommender Zug die Weiche trotz falscher Einstellung passieren, seine Räder würden die Weichenzungen beim Überqueren in die richtige Position drücken. Durch das falsch gepolte Herzstück würde aber jedes einzelne Rad einen Kurzschluß der Gleise verursachen. Für jeden Weg, auf dem eine Weiche überquert werden kann, gibt es daher immer genau eine richtige Einstellung der Weichenantriebe.

2.5.4 Kreuzungsweichen

Kreuzungsweichen vereinen die Eigenschaften einer Eisenbahnkreuzung und einer Weiche. Sie können aus zwei Richtungen auf geradem Weg überquert werden, zusätzlich kann ein Zug entlang der gebogenen Schiene am Rand von einem Gleis auf das andere wechseln, falls die Zungen entsprechend gestellt sind. Mit zusätzlichen Führungsschienen, insgesamt vier Weichenzungen und zwei Herzstücken verfügt die Kreuzungsweiche über eine komplexe Mechanik, die von zwei linken Antrieben gesteuert wird. Dadurch sind insgesamt vier Einstellungen möglich, von denen eine für den Fahrbetrieb verboten ist, da sie immer zu einem Kurzschluß führt. Die übrigen drei entsprechen jeweils genau einer erlaubten Fahrtrasse.

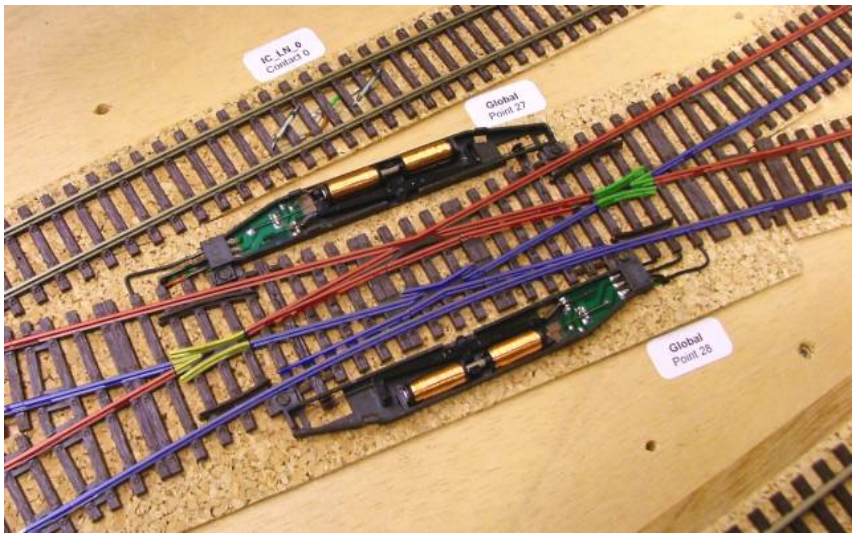


Abbildung 2.14: Elektrische Verbindungen innerhalb einer Kreuzungsweiche

In der Abbildung 2.14 sind die beiden Herzstücke gelb und grün eingefärbt, der obere Antrieb steuert das linke, der untere das rechte. Beide Antriebe müssen richtig geschaltet sein, damit auf dem Weg des Zuges über die Weiche alle Schienenstücke auf beiden Seiten gleich polarisiert sind. In der Abbildung wären alle Schienenstücke auf einer Seite des Weges blau und auf der anderen Seite rot, inklusive der Herzstücke.

Geschalteter Weg			**Antrieb 16**	**Antrieb 17**
OC_ST_4	↔	OC_LN_0	Hauptgleis (0)	Hauptgleis (0)
OC_ST_4	↔	KIO_LN_0	Hauptgleis (0)	Nebengleis (1)
(verbotene Kombination)			Nebengleis (1)	Hauptgleis (0)
KIO_LN_0	↔	IC_ST_0	Nebengleis (1)	Nebengleis (1)

Geschalteter Weg			**Antrieb 28**	**Antrieb 27**
IC_ST_4	↔	KIO_LN_1	Hauptgleis (0)	Hauptgleis (0)
KIO_LN_1	↔	OC_ST_0	Hauptgleis (0)	Nebengleis (1)
(verbotene Kombination)			Nebengleis (1)	Hauptgleis (0)
OC_LN_5	↔	OC_ST_0	Nebengleis (1)	Nebengleis (1)

Tabelle 2.2: Mögliche Einstellungen der Kreuzungsweichen

Eine Erweiterung der Kreuzungsweiche erlaubt ein nahezu beliebiges Wechseln zwischen den Gleisen. Sie enthält eine zweite gebogene Schiene am Außenrand und heißt daher doppelte Kreuzungsweiche. Bei ihr bekommt die ungenutzte Einstellung der Antriebe eine Bedeutung, sie schaltet die Weiche auf ein Abbiegen entlang der neu hinzugekommenen Schiene um. In der Anlage wurden aber keine doppelten Kreuzungsweichen eingebaut.

2.6 Kontakte

In der Bahn gibt es nur wenige Sensoren, die Informationen über den Zustand des Systems liefern können. Die wichtigsten unter ihnen sind Kontakte, die an insgesamt 80 Stellen in das Gleisbett integriert sind und darüberfahrende Züge erkennen können. Ihre Daten sind unverzichtbar für die Steuerung, um beispielsweise Blöcke nach dem Verlassen freigeben zu können oder einen Zug möglichst direkt vor einem Signal zum Stehen zu bringen.

An den betreffenden Stellen sind zwei Reedkontakte zwischen den Schienen in einem Abstand von 30 Millimetern in Fahrtrichtung angebracht. Jeder Zug trägt am vorderen Ende der Lokomotive und am hinteren Ende des letzten Waggons einen Magneten. Beim Überfahren der Reedkontakte löst jeder Magnet nacheinander die beiden Kontakte aus und die Software kann die resultierenden Meßdaten auswerten. Dabei werden die Reedkontakte zu einem Sensor zusammengefaßt, der dann kurz als Kontakt bezeichnet wird.

Abbildung 2.15: Auslösen eines Kontaktes durch eine Lokomotive

Ursprünglich waren die beiden Reedkontakte auch dafür gedacht, Geschwindigkeiten der Züge messen zu können, wofür die Meßgenauigkeit aber nicht ausreichte. Die aktuelle Steuerung nutzt die vorhandene Redundanz aus, um die Ausfallsicherheit eines Kontaktes zu verbessern. Daneben ermittelt sie aus der Auslösereihenfolge der Reedkontakte die Fahrtrichtung des auslösenden Zuges. Unabhängig davon kann die Steuerung über die Kontakte am Anfang und am Ende eines Blockes verfolgen, wann ein Zug in einen Block einfährt, sich komplett darin befindet, aus dem Block herausfährt und ihn komplett verlassen hat.

Neben den Kontakten gibt es noch andere Sensoren, beispielsweise die Gleisbesetztmelder. Sie können für jeden Block der Anlage feststellen, ob sich gerade eine Lokomotive darauf befindet

oder nicht. Mit ihnen ist es aber nicht möglich, die Position des Zuges genauer festzustellen oder einen bestimmten Zug zu identifizieren. Dafür ist ein Steuerprogramm auf das ständige Verfolgen und Zuordnen aller Kontaktmeldungen angewiesen.

Die Kontakte werden prinzipiell wie einfache Schalter abgefragt, die Messungen sind jedoch mit Störungen behaftet. Abhängig von der Ausrichtung des Magneten kann ein Kontakt während der Überfahrt kurz abfallen, bei einem zu schwachen Magnetfeld löst dieser unter Umständen auch gar nicht aus. Die Glasgehäuse sind relativ empfindlich und können zerbrechen, wenn sie Druck oder einem Stoß ausgesetzt werden. Die langen Anschlußleitungen reagieren leicht auf elektromagnetische Störimpulse, wie sie etwa ein in der Nähe schaltender Weichenantrieb erzeugen kann. Die Interfaceelektronik muß die Störungen so gut wie möglich unterdrücken, gegebenenfalls mit weitergehender Filterung durch das Steuerprogramm.

2.7 Signale

Die Signale der Bahnanlage stehen grundsätzlich am einem oder an beiden Enden eines Blockes und zeigen an, ob ein Zug in den folgenden Block einfahren darf. Sie geben gleichzeitig durch ihre Position die Stelle vor, an der ein Zug gegebenenfalls zu warten hat. Alle Signale können unabhängig von sonstigen Einstellungen der Bahn geschaltet werden, sie müssen nicht benutzt werden und dienen in erster Linie der visuellen Kontrolle und einem realistischen Fahrbetrieb.

Abbildung 2.16: Hauptsignale am Ausgang eines Bahnhofes

Im System gibt es nur zwei verschiedene Signaltypen. Auf geraden Strecken ohne Verzweigung kommen die zweifarbigen Blocksignale zum Einsatz. Ein rotes Licht bedeutet, daß der Zug an dieser Stelle anzuhalten hat und nicht in den folgenden Block einfahren darf, grün erlaubt entsprechend die Durchfahrt. Wenn auf dem Weg zum nächsten Signal mindestens eine Weiche liegt, wird stattdessen das dreifarbige Hauptsignal benutzt. Ist eine dieser Weichen auf Abbiegen geschaltet, darf ein vorschriftsmäßig fahrender Zug sie nur langsam passieren. Dies zeigt das Hauptsignal mit der zusätzlichen Farbkombination gelb/grün an.

Das Anhalten an einem Signal läuft immer nach dem selben Schema ab. Bei Annäherung an das rote Signal verringert der Zug zunächst die Geschwindigkeit. Ein Kontaktpaar direkt vor

dem Signal erkennt die darüberfahrende Lokomotive, die Steuerung schaltet den Strom ab und der Zug rollt dann so aus, daß er genau vor dem Signalmast zum Stehen kommt. Die Abstände zwischen Kontakt und Signal sind so bemessen, daß der Weg immer zum Anhalten ausreicht, vorausgesetzt der Zug war nicht zu schnell.

Die Leuchtdioden des Signals werden mit einer gemeinsamen Anode betrieben, die über einen schwarzen Kupferlackdraht nach außen geführt ist. Entsprechend gibt es für jede Kathode einen Draht, dessen Farbe der dazugehörenden Signalfarbe entspricht. Die nötigen Vorwiderstände müssen extern angeschlossen werden. Soll immer nur eine Farbe zur Zeit aufleuchten, reicht ein Widerstand in der Anodelleitung aus, anderenfalls muß ein Widerstand vor jede Kathode geschaltet werden. Die Leuchtdioden benötigen jeweils einen Strom von 10 bis 20 Milliampere, beim Betrieb mit einer Gleichspannung von 12 Volt bieten sich daher Widerstände mit einer Größe von einem Kiloohm an.

In der gesamten Anlage sind 34 Hauptsignale und 22 Blocksignale installiert, zusammen also 56 Signale mit insgesamt 146 Leuchtdioden. Diese stehen (entgegen der üblichen Konvention) immer in Fahrtrichtung links.

2.8 Beleuchtung

In der Anlage werden insgesamt 24 Laternen benutzt, um markante Punkte zu beleuchten. Dies sind in erster Linie die Bahnhöfe, die jeweils über eine ganze Lampenreihe verfügen. Daneben ist aber auch der Bahnübergang mit zwei Straßenlaternen ausgestattet.

Abbildung 2.17: Lampenmasten der Bahnhofsbeleuchtung

Im oberen Teil des Lampenmastes sind je nach Bauart eine oder zwei Fassungen für kleine Glühlampen eingelassen. Das Metallgehäuse ist leitend mit allen Lampen verbunden und führt so einen der Pole über ein braunes Kabel nach außen. Die anderen Pole sind mit je einem dünnen schwarzen Kabel verbunden, das innen im Mast verläuft. Die Lampen nehmen beim Betrieb mit 12 Volt Gleichspannung jeweils einen Strom von 20 bis 35 Milliampere auf.

2.9 Bahnübergang

Im Kicking Horse Pass befindet sich ein Bahnübergang, auf dem eine Straße die eingleisige Strecke überquert. Er besteht aus zwei Schranken, zwei Andreaskreuzen mit Lichtsignalen und einer unter der Eisenbahnplatte abgebrachten Glocke. Alle Komponenten können einzeln und unabhängig voneinander gesteuert werden.

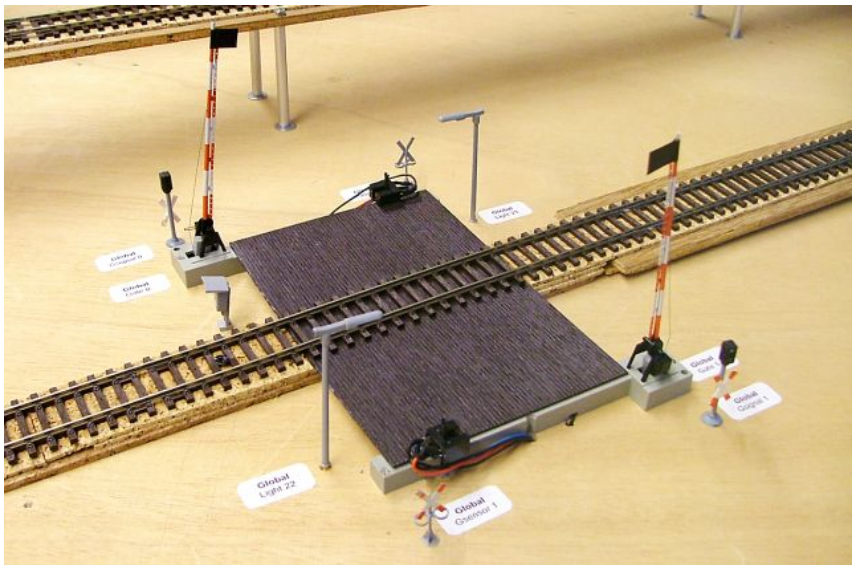


Abbildung 2.18: Bahnübergang mit Schranken, Sensoren und Signalen

2.9.1 Schranken

Wichtigster Bestandteil des Bahnübergangs sind natürlich die Schranken. Sie enthalten einen Memory-Draht, der über eine Mechanik mit dem Arm verbunden ist und von einer Feder unter Spannung gehalten wird. Im Ruhezustand werden die Schranken dadurch oben gehalten. Über zwei Anschlußkabel kann Strom durch den Memory-Draht geleitet werden, der sich daraufhin erwärmt, zusammenzieht und damit auch die Schranke absenkt. Wird der Strom abgeschaltet, kühlt der Draht wieder ab, dehnt sich aus und die Feder zieht die Schranke wieder hoch. Diese Konstruktion sorgt für einen realistischen Bewegungsablauf.

Der Widerstand einer Schranke liegt bei 10 Ohm, ab einer Spannung von 1,5 Volt entsprechend einem Strom von 150 Milliampere senkt sie sich langsam ab. Mit höherem Strom ist die Bewegung schneller, ein zu großer Strom kann die Schranke aber zerstören. Daher befindet sich unter der Platte für jede Schranke ein veränderbarer Leistungswiderstand, über den der Strom auf den gewünschten Wert eingestellt werden kann. Die Zeiten zum Heben und Senken der Schranke lassen sich auf diese Weise einfach beeinflussen.

2.9.2 Schrankensensoren

Für ein Steuerprogramm ist es wichtig zu wissen, ob die Schranken sich wie geplant geschlossen haben oder ob dies durch einen Hardwarefehler nicht möglich war. Dafür ist am Ende jeder Schranke ein kleiner schwarzer Wimpel angeklebt, der in eine Gabellichtschranke eintaucht, wenn die Schranke den untersten Punkt erreicht. Angesteuert werden die Lichtschranken von

einer unter der Platte angebrachten Adapterplatine mit Mikrocontroller, die wie ein Kontakt angeschlossen ist. Die Schaltung simuliert einen in Vorwärtsrichtung überquerten Kontakt, wenn die Schranke in die Lichtschranke eintaucht, beziehungsweise in Rückwärtsrichtung, wenn die Schranke sich wieder öffnet. Durch diesen Trick ist es nicht nötig, auf den Steuerplatinen ein spezielles Interface für Lichtschranken vorzusehen, der Anschluß muß nur eine Spannungsquelle für den Mikrocontroller bereitstellen.

2.9.3 Schrankensignale

Auf beiden Seiten der Straße, die hier über die Schienen überquert, stehen Andreaskreuze mit Lichtsignalen. Sie enthalten eine rote und eine gelbe LED zur Steuerung des Straßenverkehrs und werden genau wie die Block- und Hauptsignale entlang des Schienennetzes angesteuert.

2.9.4 Glocke

Zum Bahnübergang gehört auch eine Glocke, die für die typische Geräuschkulisse sorgen soll. Für die einfache Ansteuerung sorgt eine Treiberschaltung, das Programm muß zum Aktivieren der Glocke nur den Strom einschalten. Die Schaltung läßt die Glocke dann regelmäßig anschlagen, bis der Strom wieder abgeschaltet wird. Die Frequenz ist über ein Potentiometer auf der Platine in einem weiten Bereich einstellbar.

Die Glocke ist eine typische Türklingel mit einem Innenwiderstand von rund 65 Ohm, sie kann laut Aufschrift mit 14 bis 19 Volt Gleichspannung betrieben werden, erzeugt aber bereits bei 12 Volt einen deutlich hörbaren Ton. Bei dieser Spannung benötigen Glocke und Treiberplatine zusammen einen Strom von rund 200 Milliampere.

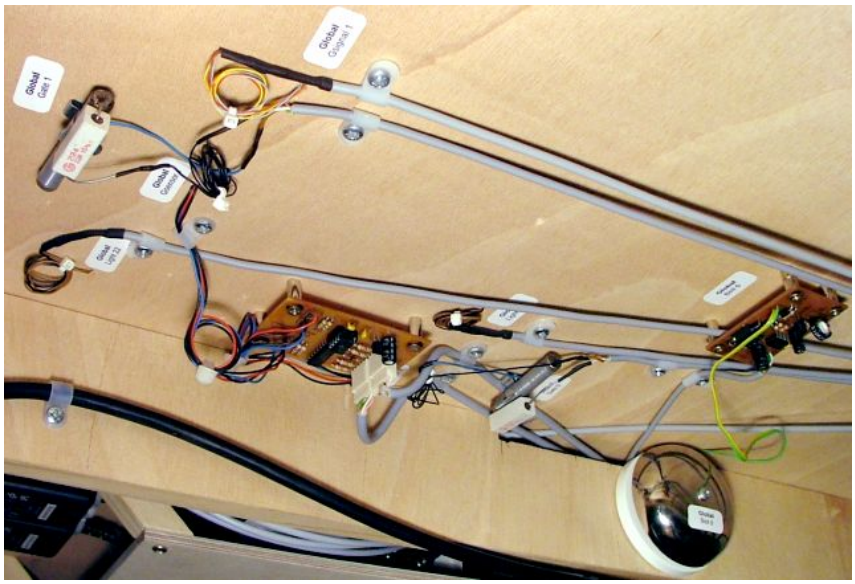


Abbildung 2.19: Technik zur Ansteuerung von Schranken, Schrankensensoren und Glocke

Die Abbildung 2.19 zeigt die zur Ansteuerung des Bahnübergangs installierte Technik unter der Platte. Am linken Bildrand und links neben der Glocke sind die Widerstände für die Schranken zu sehen. Die Platine mit den beiden weißen Steckern trägt die Ansteuerung der Lichtschranken, auf der rechten Platine sitzt der Treiberbaustein für die Glocke.

2.10 Teststrecke

Während der Entwicklung der Elektronik und der grundlegenden Steuersoftware wurden viele Funktionen auf einem kleinen Gleisoval getestet. Es läßt sich mit einer einzigen Steuerplatine betreiben und weist nicht die Komplexität der großen Anlage auf. Es kann immer noch benutzt werden, um elementare Steuerfunktionen und Algorithmen zu entwickeln und zu testen. Das Gleis besteht aus zwei Blöcken, die mit je zwei Kontakten ausgestattet sind.

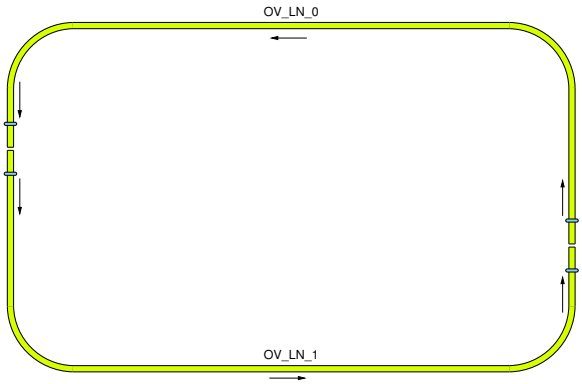


Abbildung 2.20: Streckenlayout des Gleisovals

Als weiteres Werkzeug existiert eine Platine, die mit Leuchtdioden, Tastern und Reedkontakten bestückt ist und so die Bahnhardware nachahmt. Sie ist im Anhang A.3 beschrieben.

2.11 Hinweise zum Umgang

Die gesamte Modellbahn besteht aus filigranen und empfindlichen Einzelteilen, die relativ leicht beschädigt werden können. Sie lassen sich häufig nur mit viel Aufwand oder auch gar nicht mehr reparieren. Die folgende Liste faßt ein paar Fehler zusammen, die in den Praktika der Vergangenheit immer wieder zu Defekten geführt haben.

Signale und Lampen

- Die Signalmasten brechen bereits bei leichtem Druck durch. Ein kurzes Hängenbleiben mit einer Jacke oder eine unvorsichtige Bewegung beim Schieben eines Zuges reichen dafür vollkommen aus. Reparaturen sind mit Spezialkleber möglich, solange keine größeren Teile herausgebrochen sind. Danach bleibt nur der Austausch des Signals.
- Die Lampen sind deutlich robuster, sie können aber von der Platte abgebrochen werden. Solange die Kabel dabei intakt bleiben, ist die Reparatur einfach.

Kupplungen

- Die Kupplungen zwischen den Waggons gehen bei unvorsichtiger Behandlung schnell kaputt. Plastikkupplungen werden durch senkrechtes Anheben der Waggons getrennt. Metallbügelkupplungen müssen erst durch Anheben der Bügel geöffnet werden, dann lassen sich die Waggons auseinanderschieben. Wenn sich Kupplungen während der Fahrt öffnen, sind häufig Höhenunterschiede der Kupplungen und Unebenheiten im Gleisbett dafür verantwortlich. Scheiden diese Ursachen aus, müssen die Kupplungen als wahrscheinliche Verursacher gegen neue ausgetauscht werden.

Züge schieben

- Züge sollten nach Möglichkeit gar nicht geschoben werden, weil dies die Gummiringe an den Rädern der Lokomotive beschädigen kann. Auf kurzen Distanzen ist es noch vertretbar, dabei sollten die Züge über die Lokomotive und nicht über die Waggons bewegt werden. Auf langen Strecken muß die Lok abgekoppelt und auf das Zielgleis gesetzt werden, die Waggons können geschoben werden.
- Wenn das Steuerprogramm abgestürzt sein sollte, müssen die Züge nicht manuell auf die Anfangspositionen zurückbewegt werden. Eine entsprechende Programmfunktion kann diese Aufgabe einfacher und schneller übernehmen.

Kurzschlüsse

- Prinzipiell sind Kurzschlüsse einfach zu vermeiden, so daß sie kein Problem darstellen sollten. Kommt es doch zu einem, passiert dies in der Regel mitten auf einer falsch gestellten Weiche oder über einem Blockübergang. Die Lokomotive sollte dann nach dem Abbruch des fehlerhaften Programms von der Stelle entfernt werden, weil der betreffende Motortreiber das Gleis regelmäßig wieder zu aktivieren versucht.

Weichenantriebe

- Mechanische Fehler können dazu führen, daß die Endabschaltung eines Antriebs nicht funktioniert und die aktive Spule sich überhitzt. Solche Probleme sind zum Glück selten, können aber zur Zerstörung des Antriebs führen. Beim Verdacht auf eine Fehlfunktion sollte der Strom so schnell wie möglich abgeschaltet und der Antrieb untersucht werden. Für die regelmäßige Wartung kann ein Diagnoseprogramm alle Weichenantriebe schnell und einfach prüfen.

Kapitel 3

Die Leistungselektronik

Die im vorigen Kapitel beschriebene Bahnelektronik soll mit Hilfe eines Computers gesteuert werden. Dafür wird eine Reihe von Platinen benötigt, die Interfaces für alle Sensoren und Aktoren enthalten und sich darüber hinaus mit einem Computer verbinden lassen. In Abbildung 3.1 ist eine solche Platine zu sehen, wie sie insgesamt 24 mal in der Anlage enthalten ist. Zusammen stellen die Platinen ausreichend viele Anschlüsse für die gesamte Peripherie zur Verfügung und können über serielle Schnittstellen angesteuert werden.

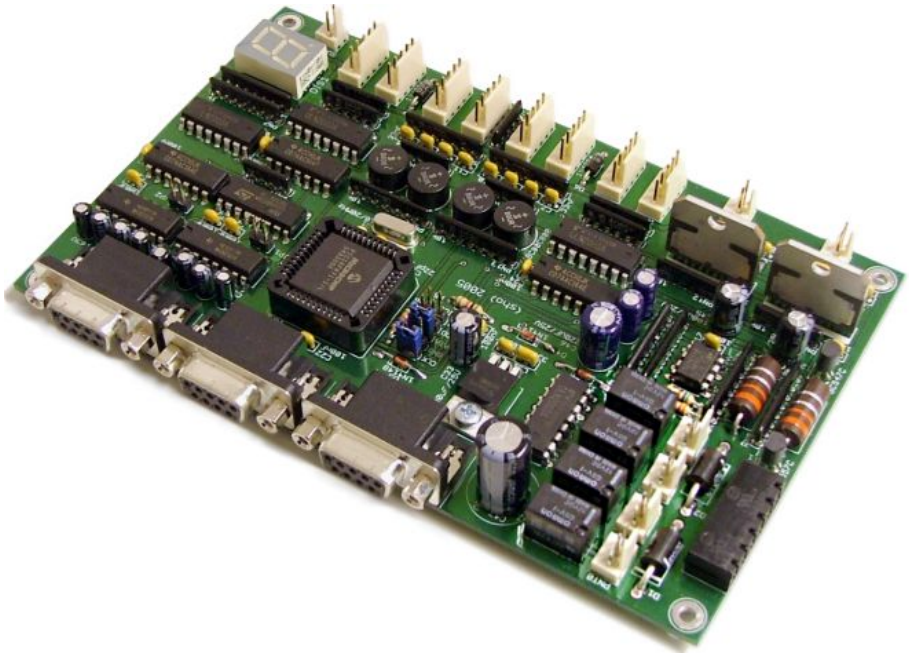


Abbildung 3.1: Platine mit der Leistungselektronik

Alle Funktionen der Platine werden von einem Mikrocontroller koordiniert, der seine Dienste externen Rechnern über ein Protokoll auf den seriellen Schnittstellen anbietet. Dieser Ansatz ermöglicht es, die Modellbahn über beliebige Feldbusse zu steuern. Es müssen nur entsprechende Busteilnehmer mit serieller Schnittstelle existieren, die als Brücke zur Leistungselektronik dienen können. Die Details dazu beschreibt das nächste Kapitel, hier geht es zunächst um den Aufbau und die Funktionsweise der Leistungselektronik.

3.1 Technische Daten

Eine Leistungselektronik zeichnet sich durch die folgenden Eigenschaften und Fähigkeiten aus.

Signale

- An jeder Platine können bis zu vier Signale mit je drei Leuchtdioden (rot, gelb, grün) betrieben werden, die Leuchtdioden lassen sich einzeln schalten.
- Der Signalanschluß enthält zusätzlich eine Stromversorgung von 12 Volt, so daß sich auch komplexere Aktoren anschließen lassen. Deren Aktionen können über einen drei Bit breiten Eingangswert gesteuert werden (siehe Grundschaltung in Anhang A.2).

Kontakte

- Die Leistungselektronik verfügt über Eingänge für vier Kontaktpaare, die mit hohem Aufwand gegen Störungen und Doppelauslösungen abgesichert sind.
- Die Firmware erkennt an der Auslösereihenfolge, wie herum der Zug den Kontakt überquert und faßt so beide Reedkontakte zu einem redundanten Sensor zusammen.
- Das System erkennt und toleriert einzelne nicht funktionierende Reedkontakte. Die Fehlerquelle wird protokolliert, der Kontakt als ganzes löst trotzdem aus, allerdings mit Verzögerung und ohne eine Richtung zu erkennen.
- Auch dieser Stecker enthält eine Stromversorgung von 12 Volt, die den Anschluß beliebiger Sensoren erlaubt. Sie müssen lediglich die für einen Kontakt typischen Auslösesequenzen erzeugen. Ein Beispiel für so eine Schaltung ist die Ansteuerung der Lichtschranken am Bahnübergang, die in Anhang A.4 beschrieben ist.

Gleistreiber

- Zwei Gleistreiber können je einen Gleisabschnitt mit Fahrstrom versorgen und so Züge fahren lassen. Für jeden Ausgang kann ein Modus (aus, vorwärts, rückwärts und bremsen) sowie das PWM-Tastverhältnis in 128 Stufen vorgegeben werden.
- Das Bremsen unterscheidet sich vom Abschalten des Motortreibers darin, daß der Zug aktiv gebremst wird und nicht nur ausrollt. Die Intensität des Bremsens kann ebenfalls über das PWM-Tastverhältnis beeinflußt werden.
- Eine Überlastsicherung schaltet einen oder beide Gleistreiber ab, sobald die Gleise kurzgeschlossen sind oder aus anderen Gründen ein zu hoher Strom fließt. Dabei werden kurze Stromspitzen ignoriert, wie sie beim Einschalten eines Verbrauchers auftreten. Nach rund 2,5 Sekunden wird das Gleis automatisch reaktiviert, zusätzlich werden die Kurzschlüsse protokolliert.
- Eine aktive Sensorschaltung an beiden Gleistreibern kann feststellen, ob sich gerade eine Lokomotive auf dem Gleis befindet und wie schnell diese gegebenenfalls fährt. Die Messungen finden bei abgeschaltetem Gleistreiber einmal pro Sekunde statt, während der Fahrt alle 0,2 Sekunden. Die Belegung von Gleisen wird mit einem aktiven Impuls gemessen, der bei stehenden Zügen als Ticken zu hören ist.
- Mit Hilfe dieser Messung kann der Mikrocontroller die PWM eigenständig so regeln, daß ein Zug eine vorgegebene Geschwindigkeit hält. Diese ist dann unabhängig von der Steigung der Strecke, der Anzahl der Waggons und ähnlichen Faktoren.
- Die Gleistreiber werden aus Sicherheitsgründen deaktiviert, wenn keine Verbindung zu einem Steuerrechner besteht. Dies betrifft ebenfalls die Sensoren an den Gleisen.

Weichentreiber

- Dieser Ausgangstreiber kann insgesamt vier Weichen schalten. Die Anschlüsse sind genauso für Verbraucher geeignet, die höhere Ströme benötigen und nicht empfindlich für Störungen in der Versorgungsspannung sind. Dazu gehören neben den Lampen auch die Schranken und die Glocke des Bahnüberganges (siehe Anhang A.5).
- Weichenantriebe sollten mit einem Filter versehen werden, damit die Störungen und Spannungsspitzen sich nicht zu weit ausbreiten können. Beispielschaltungen hierfür sind in Anhang A.6 angegeben.
- Die Firmware sequentialisiert die Schaltvorgänge, so daß niemals zwei Antriebe genau gleichzeitig umschalten. Dadurch wird die Spitzenlast für die Netzteile begrenzt.

Serielle Schnittstellen

- An die vier seriellen Schnittstellen der Platine können entsprechend viele Rechner angeschlossen werden. Zwei können direkt mit den Buchsen der Platine verbunden werden, für die übrigen wird der in Anhang A.8 beschriebe Adapter benötigt.
- Die Firmware sucht nach dem Start alle Eingänge nach Aktivität ab. Mit dem ersten gültigen Befehl verbindet sie sich fest mit dem sendenden Rechner, bis dieser die Verbindung trennt oder 0,2 Sekunden lang keinen gültigen Befehl sendet.
- Die Kommunikation basiert auf einem Request/Reply-Protokoll, der Steuerrechner sendet ein Befehlspaket und bekommt ein entsprechendes Antwortpaket zurück. Die Verarbeitung in der Firmware dauert maximal so lange wie die Übertragung eines einzelnen Bytes über die serielle Schnittstelle benötigen würde.
- Solange die Firmware mit einem Rechner verbunden ist, nimmt sie nur Befehle von diesem entgegen und ignoriert die anderen Eingänge. Die Antwortpakete werden im Normalfall nur über die gerade aktive Verbindung zurückgeschickt. Über Jumper kann dies geändert werden, so daß die Antworten synchron an mehrere Schnittstellen geschickt werden. Diese Möglichkeit ist für den Betrieb redundanter Steuerrechner wichtig, die ständig gleiche Sensordaten bekommen sollen.
- Die Datenübertragung erfolgt mit 19200 Baud, 8 Datenbits, einem Stopbit und ohne Paritätsprüfung. Für den Anschluß reichen normale serielle Verlängerungskabel aus, es werden nur die Leitungen RXD, TXD und GND benötigt.
- Die Firmware unterstützt Vollduplex- und Halbduplex-Kommunikation.

Fehlerprotokoll

- Im EEPROM des Mikrocontrollers sind Fehlerzähler gespeichert, die bei Problemen inkrementiert werden. Ein Diagnoseprogramm kann diese auslesen und so bei der Untersuchung aufgetretener Probleme helfen. Die Werte bleiben gespeichert, bis sie durch einen entsprechenden Befehl auf Null zurückgesetzt werden.
- Protokolliert werden unbeabsichtigte Resets des Mikrocontrollers, Kurzschlüsse der Gleise sowie nicht auslösende Reedkontakte.

Integrierte Anzeige

- Die auf der Platine integrierte 7-Segment-Anzeige gibt laufend Informationen über den Systemstatus aus. Wenn eine serielle Verbindung besteht, zeigt sie die Nummer der Schnittstelle an (0 bis 3). Bei Kurzschlüssen auf den Gleisen erscheint stattdessen ein schnell blinkendes E.

ICSP-Schnittstelle

- Über diesen Anschluß kann die Firmware jederzeit aktualisiert werden. Details zur Schnittstelle und dem dazugehörigen Programmiergerät finden sich in [Höhrmann].

Stromversorgung

- Die Platine wird normalerweise von drei Netzteilen mit 12 Volt versorgt. Eines ist für die Weichenausgänge reserviert und von den übrigen galvanisch getrennt, damit Störungen sich nicht in andere Teile der Schaltung ausbreiten können.
- Die zweite Quelle liefert den Fahrstrom für die Züge und versorgt die Gleistreiber.
- Alle übrigen Komponenten wie die Signale, Kontakte und die Schaltungen auf der Platine selbst werden aus der dritten Spannungsquelle gespeist. Ihre Masse ist intern mit der Masse des Fahrstromes verbunden.
- Die Netzteile werden über einem Stecker am Platinenrand angeschlossen, die Eingänge sind jeweils mit Verpolschutzdioden gesichert.
- Für den Betrieb der Anlage reichen Netzteile mit je 10 Ampere vollkommen aus.

Testausgang

- An einem kleinen Stecker am Platinenrand kann eine konstante Gleichspannung von etwa 12 Volt für beliebige Aufgaben entnommen werden.

Dieses Kapitel beschreibt die Funktionen und ihre Steuerung durch die Firmware. Dabei wird an einigen Stellen deutlich, daß das System sehr fehlertolerant entworfen wurde und einige Sicherheitsfunktionen aufweist, die sich im Nachhinein als unnötig herausgestellt haben. Sehr detaillierte Kommentare zu den Algorithmen befinden sich auch im Quellcode der Firmware.

3.2 Grundstruktur

Den Kern der Schaltung auf der Platine bilden die Stromversorgung und ein Mikrocontroller vom Typ 16F871 der Firma Microchip. Um diesen IC ist die übrige Peripherie angeordnet.

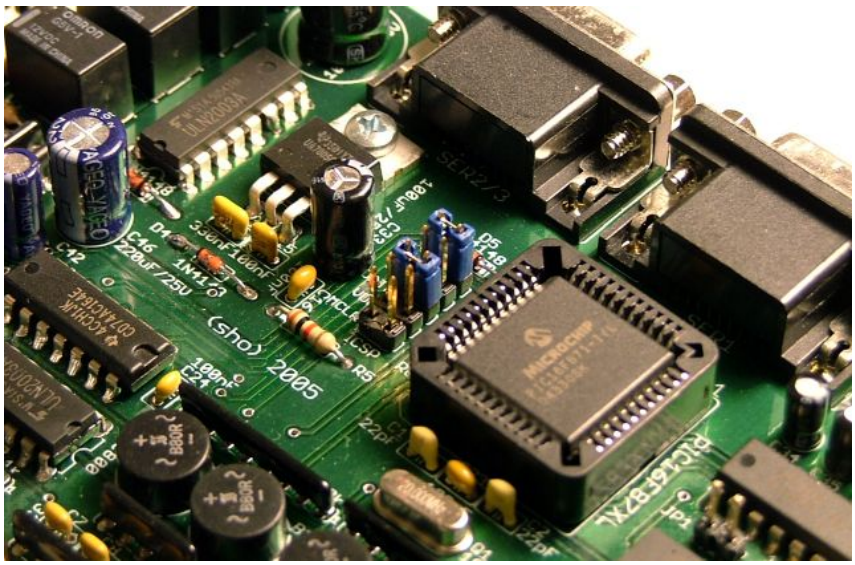


Abbildung 3.2: Mikrocontroller im Zentrum der Leistungselektronik

Die Schaltung bezieht ihren Strom von drei externen Netzteilen, die jeweils eine Gleichspannung von 12 Volt liefern. Schutzdioden an den Anschlüssen stellen sicher, daß falsch angeschlossene Netzteile keinen Schaden anrichten können, dann verzweigt sich der Stromfluß in Richtung der verschiedenen Baugruppen. Hier wird auch die Spannung für den Mikrocontroller und die übrige Logik der Platine auf 5 Volt stabilisiert, wie in Abbildung 3.3 zu sehen ist.

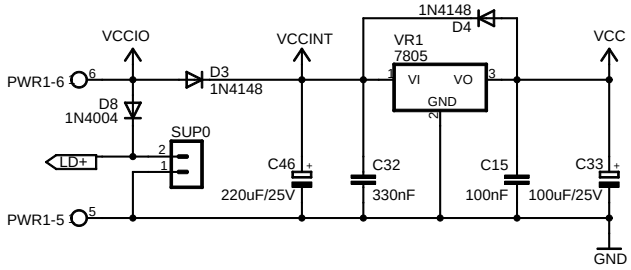


Abbildung 3.3: Stromversorgung des Digitalteils

Die Eingangsspannung wird zunächst auf die Signale und Kontakte (VCCIO), den Testausgang (SUP0), die Anzeige (LD+) und die Relais des Weichentreibers (VCCINT) verteilt. Dann erzeugt ein Spannungsregler vom Typ 7805 die 5 Volt zum Betrieb des Mikrocontrollers (VCC).

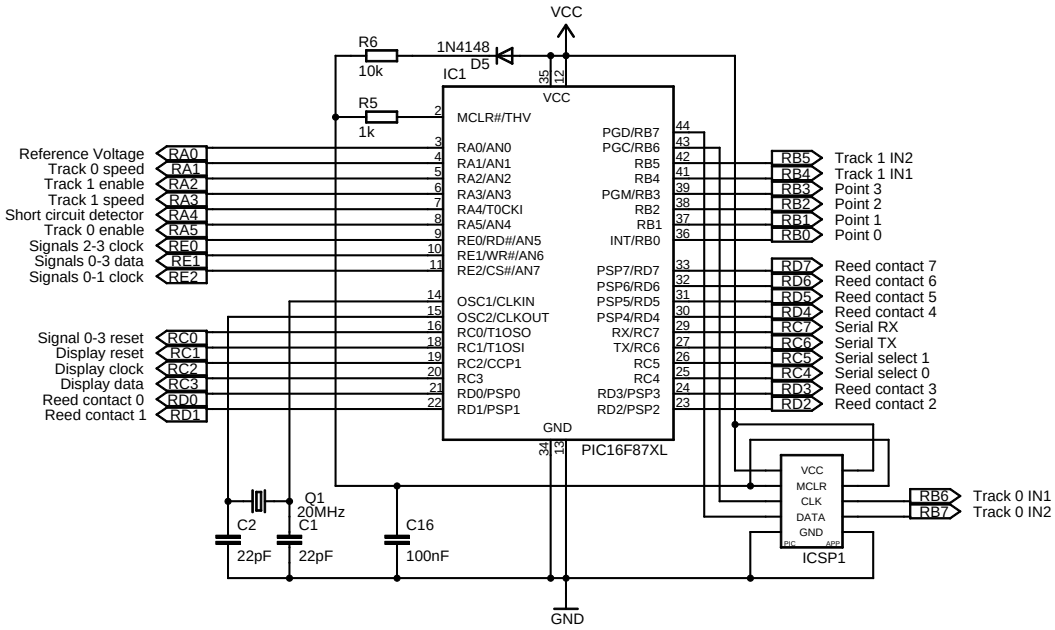


Abbildung 3.4: Beschaltung des Mikrocontrollers

Bei dem Mikrocontroller handelt es sich um einen 16F871 von Microchip, der mit einem 20 MHz Quarz und einer ICSP-Schnittstelle ausgestattet ist. Sein Codespeicher bietet Platz für 2048 Instruktionen, dazu kommen 128 Bytes RAM und 64 Bytes EEPROM. Insgesamt können 33 Pins an externe Bauteile angeschlossen werden, was für die Schaltung gerade ausreicht (vergleiche Abbildung 3.4). Von den Peripheriebausteinen des Mikrocontrollers werden nur ein ADC mit drei Eingängen, die serielle Schnittstelle und die Timer benötigt. Ein Watchdog sorgt zusätzlich dafür, daß die Firmware nicht in einer Endlosschleife hängen bleiben kann.

Die Entscheidung ist auf diesen Mikrocontroller gefallen, weil am Lehrstuhl Echtzeitsysteme und Eingebettete Systeme bereits alle nötigen Entwicklungswerkzeuge (Assembler, Debugger, Programmiergerät und Prototypschaltungen) sowie Erfahrungen mit vergleichbaren Bausteinen vorhanden waren. Der PLCC44-Sockel kommt mit wenig Platz auf der Platine aus und seine Ressourcen reichen aus, ohne große und damit unnötig teure Reserven zu bieten.

Die Firmware wechselt nach einer kurzen Startphase direkt in den sogenannten Suchmodus. In diesem Zustand wartet sie darauf, daß ein Steuerrechner einen gültigen Befehl sendet und damit die Kontrolle übernimmt. Die Peripherie wird während dieser Zeit schon weitgehend normal angesteuert, beispielsweise registriert der Mikrocontroller auslösende Kontakte und speichert sie für eine spätere Abfrage. Einzig die Gleistreiber sind zur Sicherheit noch deaktiviert. Während der Suchmodus aktiv ist, zeigt das Display ein umlaufendes Segment.

Sobald der erste gültige Befehl eintritt, wechselt die Firmware in den Kommandomodus und führt Befehle aus, die sie auf der seriellen Schnittstelle empfängt. Das Display zeigt jetzt die Nummer der aktiven Schnittstelle an und die gesamte Peripherie inklusive Gleistreibern wird normal angesteuert. Wird die Verbindung durch einen entsprechenden Befehl beendet oder eine Zeit lang kein gültiger Befehl empfangen, wechselt die Firmware wieder in den Suchmodus. Die Hardware wird dabei nicht zurückgesetzt, so daß der Rechner die Verbindung wieder aufbauen oder ein anderer diese übernehmen kann. Beim Verbindungsaufbau kann ein Rechner aber auch einen Reset durchführen lassen, wenn er mit einem definierten Ausgangszustand beginnen will.

Die verschiedenen Funktionen zur Steuerung der Hardware werden in einem Cyclic Executive ausgeführt, dessen Ausführungszeit nicht länger als die Zeit für die Übertragung eines Bytes auf der seriellen Schnittstelle sein darf (2600 Takte). Das Hauptprogramm stellt zwei Timer zur Verfügung, die mit einer Frequenz von 1000 Hz (Fast-Tick) und 100 Hz (Slow-Tick) arbeiten. Vor dem Aufruf der Unterfunktionen wird ein entsprechendes Bit gesetzt, wenn seit dem letzten Aufruf ein neuer Tick erzeugt wurde. Darauf basiert das gesamte nach außen sichtbare Timing.

3.3 Signaltreiber

Über die Steckverbinder SIG0 bis SIG3 werden Signale an die Platine angeschlossen. Um die Verkabelung zu vereinfachen, befinden sich die nötigen Vorwiderstände bereits in der Schaltung.

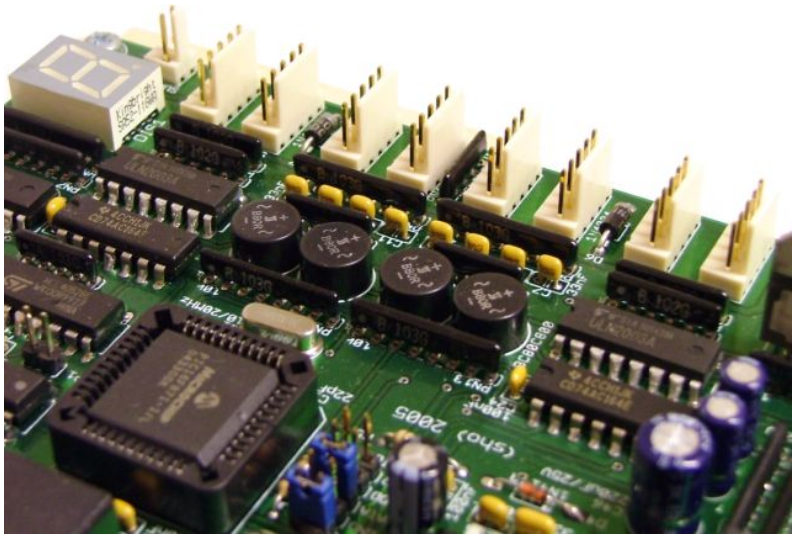


Abbildung 3.5: Anschlußfeld für Signale und Kontakte

Jeder Anschluß enthält eine 12 Volt-Versorgung, an deren Pluspol an die gemeinsame Anode des Signals angeschlossen wird. Drei weitere Leitungen verbinden die Kathoden der Leuchtdioden mit der Ausgangsstufe der Schaltung. Open-Collector-Treiber vom Typ ULN2003 können die LEDs über einen Widerstand von einem Kiloohm auf Masse ziehen und so aufleuchten lassen.

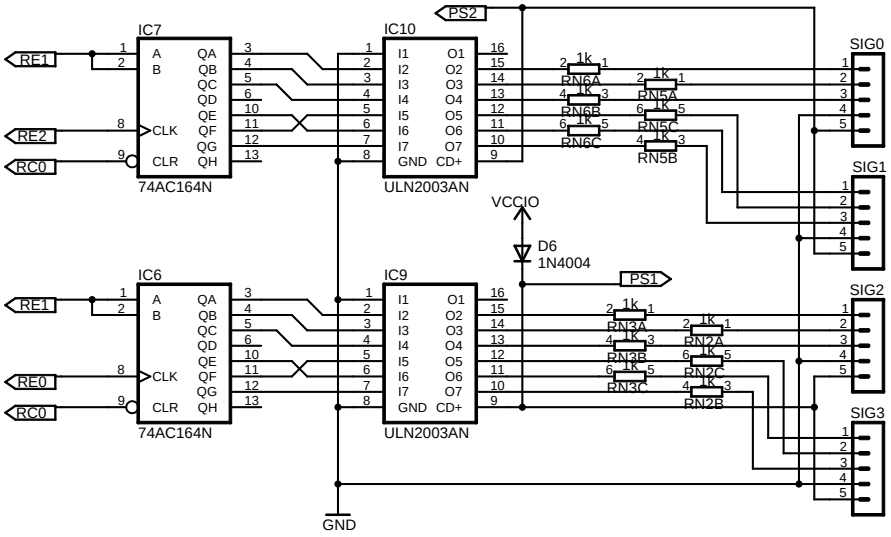


Abbildung 3.6: Treiberstufe für die Signale

Die Treiber werden nicht direkt mit dem Mikrocontroller verbunden, um Pins zu sparen. Dafür kommen Schieberegister mit seriellem Eingang zum Einsatz. Der PIC benutzt die Daten- und Taktleitungen, um die Bits für die einzelnen Leuchtdioden in die Register zu laden. Die Bausteine sind so verbunden, daß die Bits (0, 1, 2) und (4, 5, 6) den Farben (Rot, Gelb, Grün) entsprechen, die übrigen Bits werden nicht benutzt.

Dieser Aufbau der Treiberstufe bietet eine Reihe von Vorteilen. In erster Linie werden nur 4 Pins des Mikrocontrollers benötigt und der Treiber läßt sich kompakt aufbauen. Außerdem trennen die ULN2003 die Steuerung von der Peripherie und bieten so einen gewissen Schutz. Die separate Stromversorgung erlaubt es auch, an einem Signalausgang eine eigenständige Schaltung zur Anbindung eines beliebigen Aktors anzuschließen. Weitere Daten zum Stecker und ein Beispiel zur Konstruktion eines Aktors finden sich in Anhang A.2.

3.3.1 Ansteuerung durch die Firmware

Der Mikrocontroller beschreibt ein Schieberegister, indem er die Datenbits an RE1 ausgibt und gleichzeitig mit RE0 beziehungsweise RE2 das Taktsignal erzeugt. Dieser Vorgang findet immer statt, wenn das anzuzeigende Muster sich geändert hat oder wenn seit dem letzte Mal 0,25 Sekunden vergangen sind. Letzteres ist eine Sicherheitsmaßnahme, falls durch Störungen der Taktleitung ein Register nicht mehr den richtigen Wert ausgeben sollte. Der Fehler ist so auf einen kurzen Zeitraum begrenzt und wird automatisch korrigiert. Über einen Impuls an RC0 können die Register initialisiert und damit alle Leuchtdioden abgeschaltet werden.

Durch die serielle Ansteuerung dauert es 20 Mikrosekunden, bis die Bits auf ihre gewünschte Position rotiert sind. Für das menschliche Auge sind die Zwischenzustände bei den Signalen nicht sichtbar, sie können aber für hier angeschlossene Aktoren relevant sein.

3.4 Kontakteingänge

Die Kontaktanschlüsse CON0 bis CON3 enthalten zwei Sensorleitungen, die von den externen Reedkontakten mit Masse verbunden werden. Die Masseleitung ist dafür einmal direkt und einmal über einen Widerstand von einem Kiloohm nach außen geführt. Der fünfte Pin ist mit einer Spannungsquelle von 12 Volt verbunden, die externe Sensoren versorgen kann.

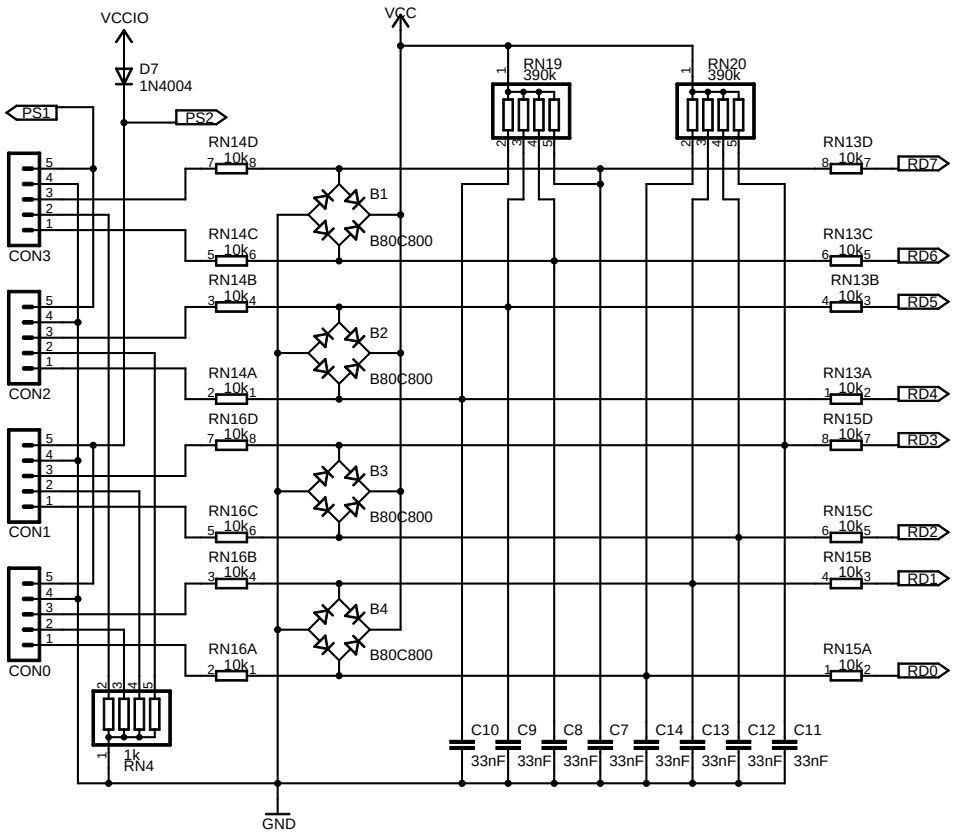


Abbildung 3.7: Interface für die Kontakte

Die eigentliche Ansteuerung eines Reedkontaktes besteht jeweils aus einem Pullup-Widerstand und einem Kondensator. Solange der Kontakt offen ist, lädt der Kondensator sich auf und der nachgeschaltete Mikrocontroller sieht einen High-Pegel an seinem Schmitt-Trigger-Eingang. Ein geschlossener Kontakt überbrückt den Kondensator mit einem deutlich kleineren Widerstand, wodurch er sich entlädt. Hält dies lange genug an, wechselt der Eingang des Mikrocontrollers auf einen Low-Pegel. Durch die relativ großen Widerstände wirken bei den Vorgängen Zeitkonstanten von 13 beziehungsweise 0,33 Millisekunden. Dies ist ausreichend, um auch stärkere Störsignale auf der Leitung wirksam zu unterdrücken.

Schutzdioden in der Stufe verhindern, daß von außen angelegte Spannungsquellen die Eingänge des Mikrocontrollers beschädigen können. Um den Aufbau zu vereinfachen, werden statt der 16 nötigen Einzeldioden 4 Brückengleichrichter benutzt, die aber genauso funktionieren. Falls ein massiver Impuls diese Sicherung überwinden sollte, schützen Widerstände von 10 Kiloohm die Eingänge des Mikrocontrollers vor einem Latchup.

Es bietet sich an, die Kontakte nicht direkt sondern über die eingebauten Widerstände von RN4 an die Masse anzuschließen. Der Aufbau reduziert die Störungen noch etwas, außerdem

liegen alle dafür benötigten Pins im Stecker direkt nebeneinander. Weitere Hinweise zur den Anschlüssen gibt Anhang A.2, und wie sich an dem Interface noch andere Sensoren betreiben lassen, demonstriert Anhang A.4 am Beispiel der Schrankensensoren des Bahnüberganges.

3.4.1 Aufbereitung der Meßwerte

Die Firmware unterdrückt bei der Auswertung zunächst alle Arten von Störungen, die von der Hardware nicht ausgefiltert werden konnten, außerdem verlängert sie Kontaktereignisse zeitlich. Dies ist nötig, damit die beiden Reedkontakte eines Paares auch noch zusammen ausgewertet werden können, wenn der Zug den Kontakt sehr langsam überquert.

Voraussetzungen

- Die Firmware verwaltet pro Reedkontakt einen Zähler, der mit 0 initialisiert wird.

Reaktion auf Ereignisse

- In regelmäßigen Abständen von 10 Millisekunden (mit jedem Slow-Tick) werden die Kontakte abgefragt. Ist einer geschlossen, wird auf seinen Zähler c_1=128 addiert, sonst wird c_2=4 von dem Zähler subtrahiert.
- Das Ergebnis wird auf den Bereich von 0 bis 255 begrenzt.
- Die Zählerstände werden mit einer Hysteresefunktion ausgewertet. Erreicht ein Zähler 255, gilt er für die folgenden Funktionen als aktiviert. Beim Erreichen der 0 wird er entsprechend deaktiviert. Alle anderen Werte ändern den Aktivierungszustand nicht.
- Dieser Filter wirkt wie ein Tiefpaß mit nachgeschaltetem Schmitt-Trigger, dessen Zeitverhalten von den Konstanten c_1 und c_2 bestimmt wird (siehe 3.4.3).

Spätestens nach dieser Stufe sollten die Sensordaten frei von Fehlern sein.

3.4.2 Richtungen und Redundanz

Die Ergebnisse der Filterung werden abschließend von einem Redundanzmanager ausgewertet. Dieser hat die Aufgabe, die Fahrtrichtung des auslösenden Zuges festzustellen. Dabei werden auch nicht funktionierende Reedkontakte erkannt.

Redundanzmanager

- Beide Reedkontakte eines Kontaktes werden immer zusammen ausgewertet.
- Bei Aktivierung eines Reedkontaktes prüft die Firmware auch dessen Partner. Sind beide gleichzeitig aktiv, wird die Richtung des darüberfahrenden Zuges anhand der Auslösereihenfolge ermittelt und gespeichert. Mögliche Ergebnisse sind die Werte vorwärts, rückwärts und unbekannt (gleichzeitige Aktivierung).
- Wird ein Kontakt wieder inaktiv, ohne daß sein Partner gleichzeitig aktiv war, ist der Partnerkontakt möglicherweise defekt. Der Anwendung wird eine Auslösung mit unbekannter Richtung signalisiert, außerdem protokolliert die Firmware den Fehler.

Ausgabedaten

- Für jedes Kontaktpaar speichert die Firmware zwei Werte, und zwar die Richtung der letzten Auslösung und einen zwei Bit breiten Sequence Counter. Dieser wird bei jeder Auslösung eines Kontaktpaares inkrementiert.

3.4.3 Wahl der Zeitkonstanten

Die erste Filterstufe der Kontaktauswertung kann über zwei Konstanten gesteuert werden. Sie entsprechen Zeitangaben, wobei 2,55 Sekunden geteilt durch den Wert die korrespondierende Zeit ergibt. Die erste Konstante c_1 legt die Zeitspanne fest, die ein Reedkontakt geschlossen sein muß, bevor die Firmware ihn registriert. Der Wert 128 entspricht einer Zeit von 20 Millisekunden, die Reaktion erfolgt damit fast sofort. Die Zeit kann durch eine kleinere Konstante aber auch verlängert werden, um von der Hardware nicht ausgefilterte Störungen besser zu unterdrücken.

Entsprechendes gilt für die zweite Konstante c_2. Sie gibt die Zeit an, die zwischen dem Auslösen der beiden Reedkontakte eines Paares maximal vergehen darf. Die Einstellung stellt immer einen Kompromiß dar, der aktuelle Wert von 4 setzt die Zeit auf 0,64 Sekunden.

- Jeder Zug löst den Kontakt beim Überfahren zweimal aus, einmal mit der Lokomotive und einmal mit dem letzten Waggon. Die Zeit dazwischen muß in jedem Fall länger als die eingestellte Grenze sein, damit die Ereignisse getrennt gewertet werden können. Dieser Faktor begrenzt die Höchstgeschwindigkeit der Züge abhängig von ihrer Länge.
- Ein Kontaktpaar mit einem defekten Reedkontakt wird erst als aktiv gemeldet, wenn nach dem Überqueren die über c_2 eingestellte Zeit verstrichen ist. Bei Höchstgeschwindigkeit ist die Lokomotive nach 0,64 Sekunden bereits 30 Zentimeter hinter dem Kontakt.
- Züge dürfen einen Kontakt nur so langsam passieren, daß der Magnet beide Reedkontakte innerhalb der Zeitspanne auslöst. Bei 0,64 Sekunden liegt die Mindestgeschwindigkeit bei 4,7 Zentimeter pro Sekunde.

Die Werte der Konstanten orientiert sich an Erfahrungswerten aus dem Betrieb der Anlage. Falls es nötig sein sollte, können sie im Quellcode der Firmware einfach angepaßt werden.

3.5 Gleistreiber

Die beiden Gleistreiber fallen auf der Platine sofort durch die großen Motortreiber-ICs und die dahinterliegenden Leistungswiderstände ins Auge. Vor ihnen befinden sich die beiden Anschlüsse, an denen der Fahrstrom für die Gleise entnommen werden kann.

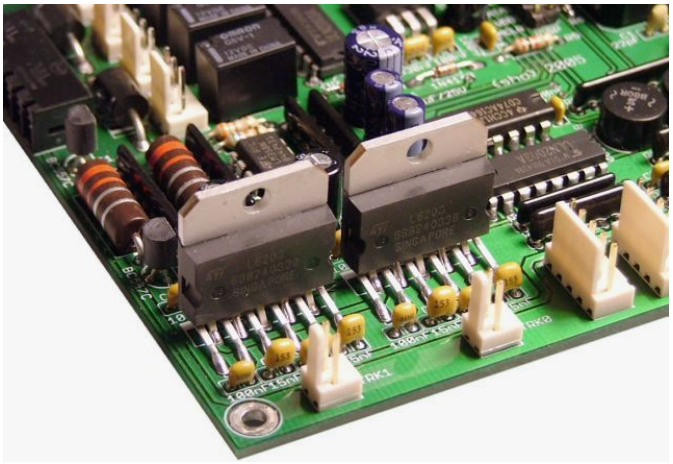


Abbildung 3.8: Gleistreiber und Überlastsicherung

3.5.1 Kurzschlußsicherung

Die Treiber-ICs müssen jeweils einen Dauerstrom von einem Ampere liefern können. Gleichzeitig sind die Gleise an ihren Ausgängen ständig dem Risiko eines Kurzschlusses durch falsch gestellte Weichen und Fremdkörper auf den Schienen ausgesetzt. Deshalb müssen die Ausgänge durch eine Überlastsicherung vor Schäden bewahrt werden. Diese überwacht den gesamten Strom, der in die beiden Motortreiber und wieder aus ihnen herausfließt, und kann so auch Kurzschlüsse gegen die Versorgungsspannung oder eine Nachbarschaltung erkennen.

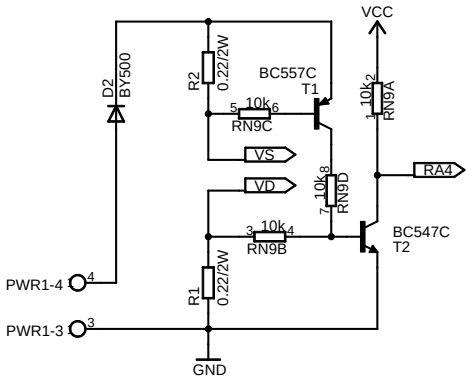


Abbildung 3.9: Überlastsicherung der Gleistreiber

Der Strom fließt auf seinem Weg vom Netzteil durch zwei niederohmige Lastwiderstände und verursacht über ihnen einen Spannungsabfall. Bei einem Strom von mehr als 2,7 Ampere erreicht diese Spannung 0,6 Volt und der darüber angebrachte Transistor wird leitend. T2 zieht den Eingangspin RA4 des Mikrocontrollers direkt auf Masse, T1 kann T2 übersteuern und so die selbe Wirkung hervorrufen. Der Mikrocontroller erkennt an einem Low-Pegel an diesem Eingang, daß eine Überlastsituation oder ein Kurzschluß vorliegt. Bei dem Pin handelt es sich um einen Eingang mit Schmitt-Trigger, so daß die analogen Spannungspegel kein Problem darstellen.

Kurzschluß erkennen

- Der Kurzschlußdetektor verfügt über einen Zähler, der mit 0 initialisiert wird.
- Alle 10 Millisekunden (mit jedem Slow-Tick) fragt der PIC den Detektor ab. Meldet dieser eine Überlastung der Treiber, wird auf einen Zähler der Wert 64 addiert, sonst 1 von ihm subtrahiert. Der Wertebereich bleibt dabei auf 0 bis 255 begrenzt.
- Erst wenn der Zähler 255 erreicht, beginnt die Abschaltsequenz. Dadurch werden kleine Lastspitzen toleriert und die Kurzschlußerkennung funktioniert auch, wenn das verursachende Gleis durch die PWM gelegentlich abgeschaltet wird.

Verursacher ermitteln

- Aus Platzgründen ist nur eine Überlastsicherung in der Schaltung vorhanden, deshalb kann die Firmware das verursachende Gleis nicht eindeutig ermitteln. Im Falle eines Kurzschlusses wird zunächst Gleistreiber 0 abgeschaltet. Hilft dies nicht, werden als nächstes Treiber 1 und schließlich beide Bausteine deaktiviert.
- Diese Methode kann die Ursache in der Regel schnell einkreisen, so daß die Schaltung mit dem nicht betroffenen Gleis normal weiterarbeiten kann.

Reaktivierung

- Nach dem Abschalten des verantwortlichen Gleistreibers dauert es 2,55 Sekunden, bis die Firmware diesen reaktiviert. Zu dem Zeitpunkt wird der Fehler auch im EEPROM des Mikrocontrollers protokolliert.
- Ist der Fehler jetzt nicht beseitigt, wiederholt sich die beschriebene Sequenz mit der Inkrementierung des Zählers.

Die beschriebene Sicherung kommt mit einem Minimum an Bauteilen aus und reagiert trotzdem sehr schnell und zuverlässig auf Kurzschlüsse. Das Zeitverhalten kann durch Veränderung der Konstanten in der Firmware angepaßt werden, falls dies nötig sein sollte.

3.5.2 Erzeugung des Fahrstromes

Der Fahrstrom wird von fertigen Motortreiber-ICs bereitgestellt, die eine komplette H-Brücke und die nötige Ansteuerung enthalten. Die Brücke kann einen der Ausgänge mit der positiven Versorgungsspannung und den anderen mit Masse verbinden, so daß der Zug vorwärts oder rückwärts fährt. Sind alle Transistoren der Brücke abgeschaltet, ist der Ausgang hochohmig und die Stromzufuhr zum Gleis faktisch unterbrochen. Als letzte Möglichkeit kann die Brücke beide Ausgänge miteinander verbinden, diese Einstellung bremst einen angeschlossenen Zug. Dessen noch drehender Motor arbeitet dann als Generator, der erzeugte Strom wird über die Brücke in Wärme umgewandelt, und der resultierende Energieverlust bremst den Motor.

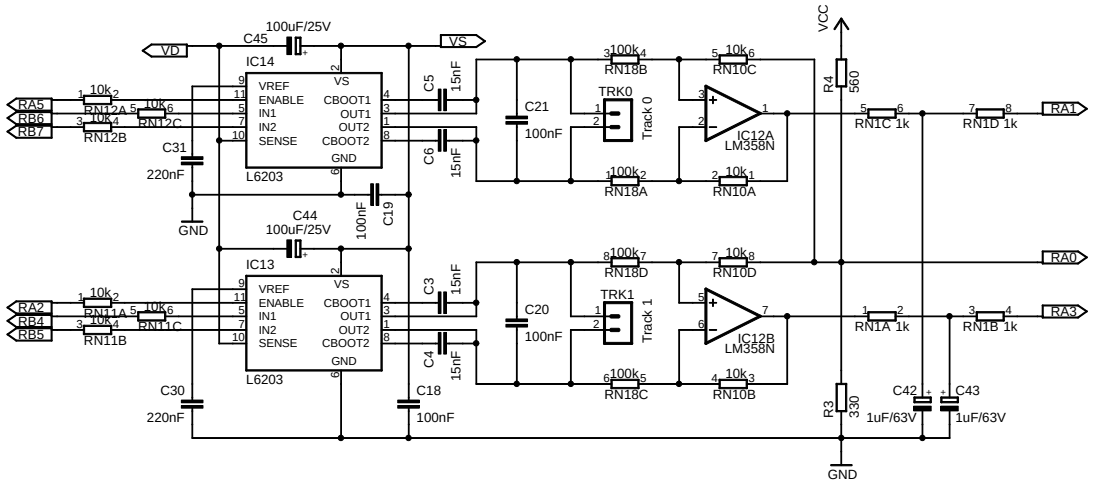


Abbildung 3.10: Ansteuerung der Gleise

Der hier eingesetzte Motortreiber L6203 ist kompakt, robust, weit verbreitet und kann einen Dauerstrom bis 4 Ampere liefern. Die Eingänge IN1 und IN2 steuern die H-Brücke, ENABLE erlaubt es, Pulsweitenmodulation sehr einfach mit Frequenzen bis 100 Kilohertz zu realisieren. Diese Pins sind mit dem Mikrocontroller über Widerstände verbunden, die bei Fehlfunktionen ein Latchup verhindern sollen. An den Ausgängen der Motortreiber sind direkt die Stecker für die Gleise und eine Meßschaltung angeschlossen, die im folgenden Abschnitt diskutiert wird. Die Kondensatoren im Umfeld der Treiber entsprechen den Empfehlungen aus dem Datenblatt.

Die Fahrgeschwindigkeit wird genau wie die Intensität beim Abbremsen einer Lokomotive über Pulsweitenmodulation gesteuert. Dazu setzt die Firmware den Bresenham-Algorithmus ein.

Fahrtrichtung

- Die Eingänge (IN1, IN2) werden nach dem gewünschten Modus gesetzt, (0,1) für vorwärts, (1,0) für rückwärts sowie (0,0) zum Bremsen.
- Wenn der Motortreiber aktiv ist, also gerade keine Messung stattfindet, das Gleis nicht per Software abgeschaltet wurde und auch kein Kurzschluß vorliegt, findet die Pulsweitenmodulation statt, anderenfalls wird ENABLE fest auf 0 gesetzt.

Pulsweitenmodulation

- Für jeden Gleistreiber wird ein Zähler mit einer Breite von 8 Bit verwaltet. Das Tastverhältnis der PWM wird intern als Wert zwischen 0 und 255 gespeichert.
- Alle 10 Millisekunden (also mit jedem Slow-Tick) wird das Tastverhältnis auf den Zähler addiert. Kommt es dabei zu einem Überlauf, wird ENABLE auf 1 gesetzt, sonst auf 0. Je größer das Tastverhältnis ist, desto häufiger kommt es zu einem Überlauf und desto schneller fährt der angeschlossene Zug.
- Bei einem Tastverhältnis von 128 generiert der Algorithmus ein Rechtecksignal von 50 Hertz. Soll der Zug schneller fahren, verlängert sich der High-Pegel, bei einem niedrigeren Tastverhältnis wird entsprechend der Low-Pegel länger.

Dieser Algorithmus erlaubt es, die Geschwindigkeit in 255 Abstufungen einzustellen. Dabei zeigt er bessere Fahreigenschaften als die klassische Pulsweitenmodulation mit fester Fensterbreite, insbesondere bei niedrigen Geschwindigkeiten. Die Anwendungsprogramme können durch eine Einschränkung des Interfaces nur auf die oberen 7 Bit des Tastverhältnisses zugreifen, dies ist aber immer noch vollkommen ausreichend.

3.5.3 Messungen an den Gleisen

Der Mikrocontroller kann über zwei Meßschaltungen die Spannungen messen, die momentan an den Gleisen anliegen. Im einfachsten Fall mißt die Schaltung so die Ausgangsspannung eines aktiven Motortreibers. Interessanter ist jedoch, was beim Abschalten des Treibers für einen kurzen Moment passiert. Fährt ein Zug auf dem angeschlossenen Gleis, dreht der Motor in der Lok sich durch die Trägheit noch weiter und funktioniert wie ein Generator. Dabei erzeugt er eine Spannung, die proportional zur Drehzahl und damit zu seiner Geschwindigkeit ist. Auf diese Weise kann die Sensorschaltung die Geschwindigkeit eines fahrenden Zuges messen.

Steht in diesem Moment keine Lokomotive auf den Gleisen, wird die Spannung am Ausgang des abgeschalteten Motortreibers von den Kondensatoren aufrecht gehalten und fällt nur langsam ab. An diesem Unterschied läßt sich also erkennen, ob überhaupt ein Verbraucher auf dem Gleis steht, die Funktion wird als Gleisbesetztmelder bezeichnet.

Die Meßschaltung besteht aus Operationsverstärkern in Subtrahierschaltung. Der Nullpunkt ist durch den Spannungsteiler aus R3 und R4 etwas angehoben, so daß auch negative Spannungen gemessen werden können. Die übrigen Widerstände skalieren die Ergebnisse auf ein Zehntel. Tiefpaßfilter glätten die gemessenen Spannungen und führen sie über Schutzwiderstände zu ADC-Eingängen des Mikrocontrollers. Die virtuelle Masse kann über einen dritten ADC-Eingang gemessen und so von den anderen Werten wieder subtrahiert werden.

Die Firmware untersucht beide Gleise mit variabler Häufigkeit. Besteht keine Verbindung zu einem Steuerrechner, wird gar nicht gemessen. Sonst erfolgt die Messung bei stehenden Zügen einmal pro Sekunde, während der Fahrt dann alle 0,2 Sekunden. Die Ursache hierfür ist, daß die Geräuschkulisse auf ein Minimum reduziert werden soll. Der Gleisbesetztmelder muß für eine Messung die Gleistreiber kurz aktivieren, auch wenn diese eigentlich abgeschaltet sind. Die Impulse sind als Ticken der Motoren hörbar. Um die Nerven der Entwickler zu schonen, erfolgt dies im Ruhezustand so selten wie möglich. Während der Fahrt stört das Ticken hingegen nicht und die Messungen erfolgen entsprechend häufiger.

Abbildung 3.11: Meßschaltung an der Rückseite der Gleistreiber

Eine Meßsequenz läuft in mehreren Schritten ab, die jeweils von Fast Ticks ausgelöst werden. Die volle Sequenz mit allen Schritten dauert 7 Millisekunden, der resultierende Wert entspricht durch die Skalierung relativ genau der Geschwindigkeit des Zuges in Zentimetern pro Sekunde.

Ablauf der Meßsequenz

- Zunächst wird der Motortreiber für den gesamten Rest des Programms gesperrt und hochohmig geschaltet. Der zum Treiber gehörende Kanal des ADC wird aktiviert, die Spannung an diesem Eingang ist 3 Millisekunden später stabil.
- Anschließend startet die Geschwindigkeitsmessung, deren Ergebnis aber erst mit dem folgenden Tick ausgelesen wird.
- Die Firmware aktiviert den Gleistreiber für eine Millisekunde, um die Belegung des Gleises zu prüfen. Der Impuls erfolgt mit korrekten Polarität: rückwärts, wenn der Treiber auf Rückwärtsfahrt eingestellt war, sonst vorwärts.
- Am Ende der Zeitspanne wird die Spannung gemessen und der Treiber abgeschaltet. Beim nächsten Tick erfolgt eine weitere Messung, um die Veränderung untersuchen zu können. Beides zusammen dauert zwei Millisekunden.
- Zuletzt wird der Motortreiber wieder in den Ausgangszustand zurückversetzt und die normale Steuerung freigegeben. Von dem Meßwert wird der Betrag gebildet. War der vorige Meßwert nicht Null, wird der aktuelle Wert durch den Mittelwert von vorigem und aktuellem Meßwertes ersetzt. Ist das Resultat sehr klein oder steht gar kein Zug auf den Schienen, muß das Ergebnis auf 0 korrigiert werden.

Wenn die Firmware nicht mit der Messung eines Gleises beschäftigt ist, bestimmt sie stattdessen das Potential der virtuellen Masse, die den Nullpunkt für die anderen Spannungsmessungen vorgibt. Dazu sind nur wenige Schritte notwendig, die auch in Fast-Ticks ablaufen.

Messung der Referenzspannung

- Anfangs wird der ADC-Kanal der Referenz aktiviert und einen Tick lang gewartet.
- Daraufhin beginnt die Messung. Deren Ergebnis wird eine Millisekunde später gelesen und gespeichert, so daß der Code für die Gleissensoren darauf zugreifen kann.

Mit diesen Möglichkeiten kann die Steuersoftware alle Blöcke ermitteln, die momentan von Zügen belegt sind. Genauso wichtig ist die Messung der Geschwindigkeit, die es ermöglicht, das Fahrverhalten der Züge zu regeln. Das PWM-Tastverhältnis kann laufend so angepaßt werden, daß Züge auch in Steigungen und im Gefälle immer gleich schnell fahren. Diese Aufgabe kann ein externes Programm im Steuerrechner übernehmen, die Firmware enthält aber auch eine transparent nutzbare Funktion hierfür.

3.5.4 Regelung der Geschwindigkeit

Ein mit der Leistungselektronik verbundener Rechner kann sich entscheiden, ob er die PWM selber vorgeben will oder die genaue Einstellung der Firmware überläßt. Im zweiten Fall gibt der Rechner nur die Zielgeschwindigkeit vor und die Leistungselektronik versucht, diese möglichst genau zu halten. Als Grundlage dient ein leicht erweiterter PID-Regler, der jedesmal ausgeführt wird, wenn neue Meßdaten von dem jeweiligen Gleis vorliegen.

Vorgaben

- Die Zielgeschwindigkeit ist in der Variablen target, die aktuelle in speed gespeichert.
- Der Regler wird über die Konstanten `KC`, `KP`, `KI` und `KD` abgestimmt.
- Die Hilfsvariablen `error`, `errorsum`, `diff` und `lastspeed` werden mit 0 initialisiert.
- Der Regler legt sein Ergebnis in der Variablen `pwm` ab.

Regelung

- Beim Eintreffen neuer Meßdaten wird der folgende Code ausgeführt.

```
error = target - speed;
errorsum = errorsum + error;
diff = lastspeed - speed;
pwm = 2 * ( KC*target + KP*error + KI*errorsum + KD*diff );
if (pwm < 0) pwm = 0;
if (pwm > 255) pwm = 255;
lastspeed = speed;
```

- Die Variable `pwm` gibt das neue Tastverhältnis für den Gleistreibers an.

Dieser Algorithmus weicht etwas von dem klassischen PID-Regler ab, der differentielle Fehler wird anders berechnet und es kommt ein Summand hinzu, der nur von dem Sollwert abhängt. Durch diese Modifikationen erreicht der Regler die Zielgeschwindigkeit wesentlich schneller und schwingt dabei kaum. Bei Tests hat sich gezeigt, daß ein Wert von 1 für alle Konstanten gute

Ergebnisse liefert und die Formel dadurch extrem einfach in Assembler zu implementieren ist. Um das Verhalten an den Blockübergängen zu verbessern, wird der Regler in einen übergeordneten Algorithmus eingebettet.

Sonderregeln für Blockübergang

- Wenn ein Zug in dem Block steht, arbeitet die Regelung wie oben beschrieben.
- Verläßt der Zug seinen Block, wird die PWM kurz auf 0 verringert, damit der Zug beim Wechsel nicht aus beiden Blöcken gleichzeitig mit Strom versorgt wird.
- In allen anderen Fällen wird das Tastverhältnis fest auf `2*KC*target` eingestellt, was für den einfahrenden Zug eine gute Grundlage darstellt.

Ganz nahtlos kann der Blockwechsel nie ablaufen, vor allem wenn die angrenzenden Gleise von unterschiedlichen Leistungselektroniken bedient werden. Mit dem genannten Algorithmus ist der Übergang vor allem bei mittleren bis hohen Geschwindigkeiten kaum zu sehen.

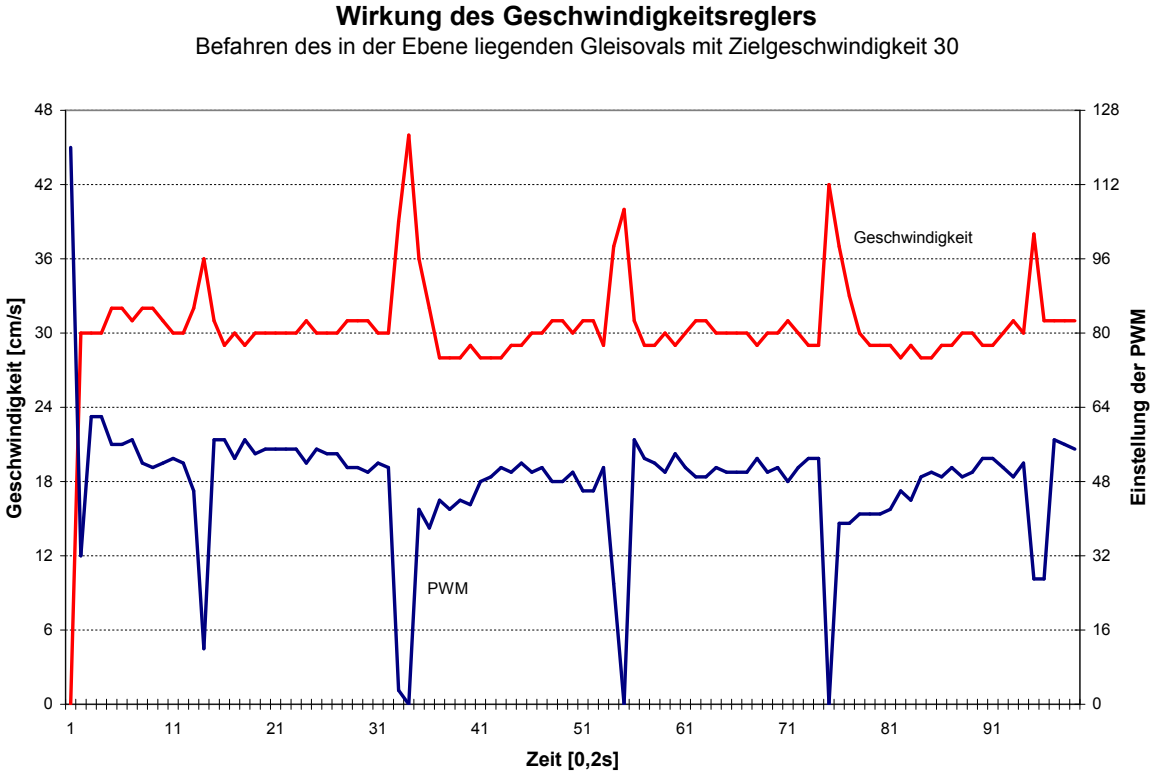


Abbildung 3.12: Geschwindigkeitsregler in der Ebene

Die Ergebnisse des Reglers auf einer ebenen Strecke sind in Abbildung 3.12 zu sehen. Eine Lok fährt hier auf dem ovalen Testgleis, der Regler soll die Geschwindigkeit auf 30 Einheiten halten. Die Lokomotive erreicht den Zielwert fast sofort und hält das Tempo sehr konstant. Deutlich sind in der Abbildung auch die Blockwechsel zu erkennen. Hier kann die Leistungselektronik keine brauchbaren Meßwerte ermitteln, daher kommt es zu den Spitzen in den Kurven. Wird das Gleisoval stark geneigt, muß der Regler zusätzlich die verschiedenen Steigungen ausgleichen. Das Resultat ist in Abbildung 3.13 zu sehen. Das Tempo variiert durch die erschwerten Bedingungen etwas mehr, und in der PWM-Kurve sind deutlich die Steigungen zu erkennen.

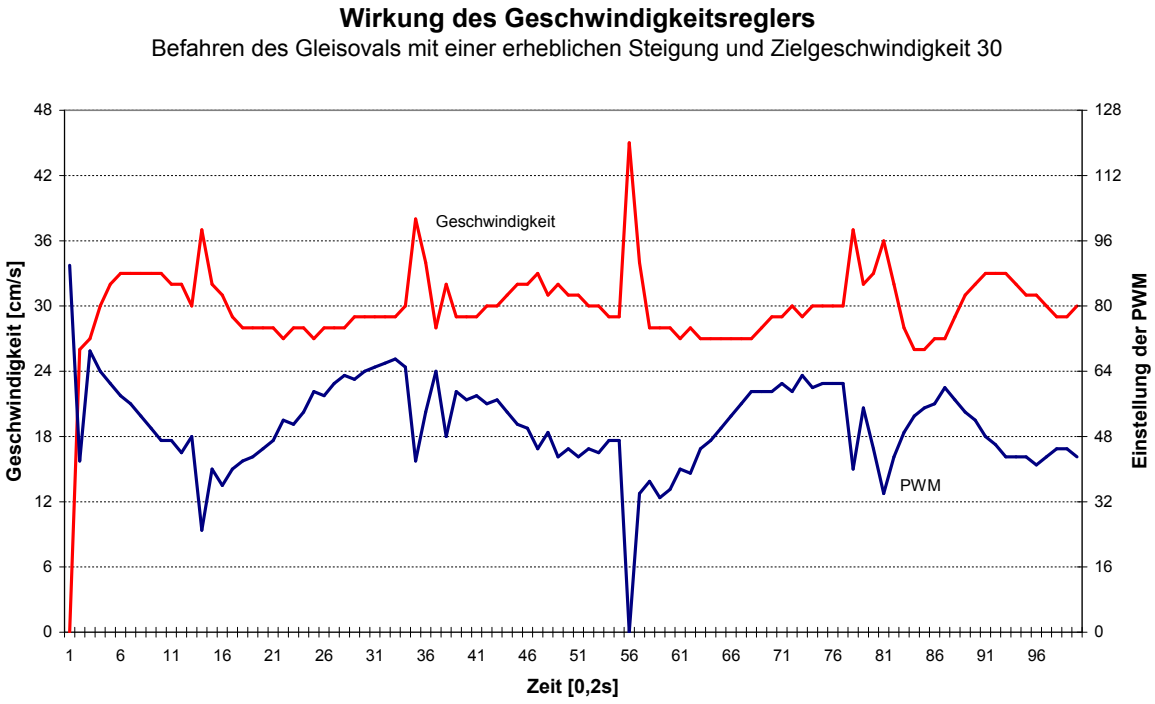


Abbildung 3.13: Geschwindigkeitsregler mit starken Höhenunterschieden

3.6 Weichentreiber

Die Elektronik der Weichensteuerung besteht nur aus wenigen Teilen. Die Versorgungsspannung dieses Moduls ist von den übrigen Leitungen galvanisch getrennt, sie verfügt über eine eigene Verpolschutzdiode sowie einen Pufferkondensator, der die Stromspitzen beim Umschalten eines Weichenantriebes ausgleicht. Relais übernehmen das Umschalten der Spannung zwischen zwei der drei Ausgangspins, die dritte ist fest mit Masse verbunden.

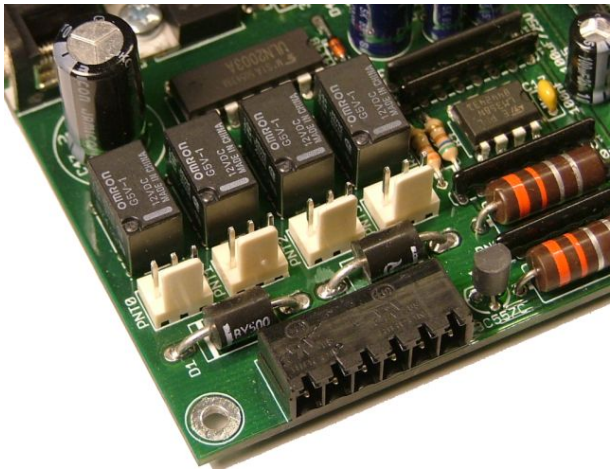


Abbildung 3.14: Weichentreiber mit Relaisgruppe

Die Relais werden von einem Treiberbaustein vom Typ ULN2003 geschaltet, der über vier Pins direkt mit dem Mikrocontroller verbunden ist. Er kann pro Ausgang bis zu 0,5 Ampere schalten und enthält Freilaufdioden, externe Bauteile werden daher nicht benötigt.

Um die Wahrscheinlichkeit möglichst gering zu halten, daß elektromagnetische Störungen von den Weichenantrieben die Logikbausteine beeinträchtigen, befindet sich die gesamte Baugruppe am Rand der Platine, dicht neben der Stromzufuhr. Damit ist ein möglichst großer Abstand zu den Logikbausteinen sichergestellt, außerdem verlaufen die Leiterbahnen senkrecht zu den anderen Versorgungsleitungen, um die induktive Koppelung so schwach wie möglich zu halten.

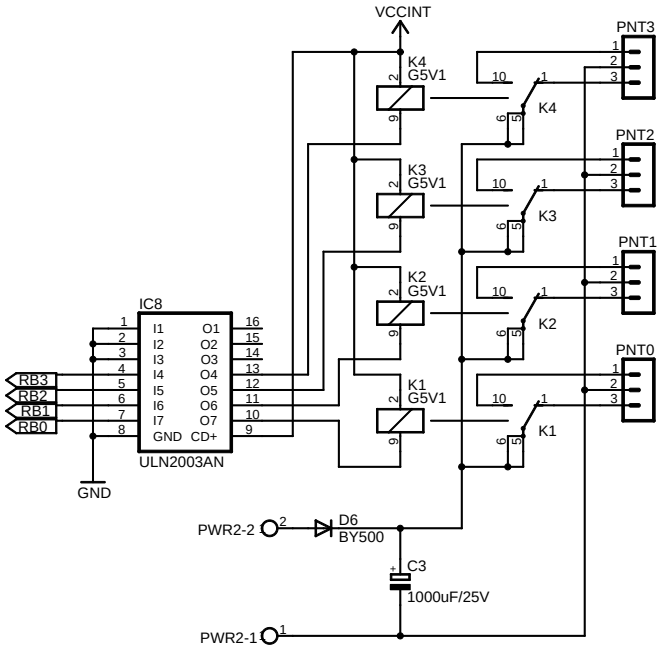


Abbildung 3.15: Ansteuerung der Weichen

Die Firmware muß beim Schalten zwei Randbedingungen beachten. Einerseits sollten nie mehrere Weichen gleichzeitig schalten, damit die Stromversorgung nicht zu stark belastet wird und die elektromagnetischen Störungen in Grenzen gehalten werden. Andererseits sollte zwischen den Schaltvorgängen einer Weiche eine Mindestzeit vergehen. Dies dient der Fairneß, wenn zwei Weichen auf einen Schaltvorgang warten, erlaubt aber auch den Einsatz von Entstörschaltungen mit langsam aufladendem Energiepuffer, wie sie in Anhang A.6 beschrieben sind.

Randbedingungen

- Die Firmware verwaltet neben einem globalen Timer für alle Weichenantriebe einen lokalen Timer pro Ausgang. Sie werden alle auf 0 initialisiert.
- Die Timer können auf einen Wert gesetzt werden und dekrementieren sich dann selbst alle 10 Millisekunden (mit jedem Slow-Tick), bis sie wieder 0 erreichen.
- Eine Weiche darf nur schalten, wenn der globale und ihr lokaler Timer 0 sind.

Schalten einer Weiche

- Wenn mindestens eine Weiche umgeschaltet werden soll, sucht die Firmware diejenige mit der niedrigsten Nummer heraus, die nach obigen Kriterien schalten darf.

- Die Weiche wird umgeschaltet, der globale Timer auf 5 und der lokale Timer auf 20 gesetzt. Damit sind weitere Schaltvorgänge vorläufig wie gefordert blockiert.

Wenn der Steuerrechner schnellere Änderungen verlangt als die Zeitschranken erlauben, nähert sich der Algorithmus immer dem aktuellen Zielwert statt alte Zwischenzustände erreichen zu wollen. Die lokalen Timer müssen daher viermal so lang wie der globale laufen, damit in so einer Situation keine Weiche benachteiligt wird. Die genauen Laufzeiten der Timer müssen an die Weichenantriebe und eventuell vorhandene Filterschaltungen angepaßt werden. Sie können in der Firmware problemlos geändert werden.

3.7 Serielle Schnittstellen

Jedes System mit einer seriellen Schnittstelle im RS232-Standard kann die Leistungselektronik über ein Kommunikationsprotokoll steuern. Dafür ist die Platine mit vier Buchsen versehen, die über einen Multiplexer mit dem Mikrocontroller verbunden sind. Der Prozessor kann eine Schnittstelle auswählen und dann durch diese mit dem externen Rechner kommunizieren.

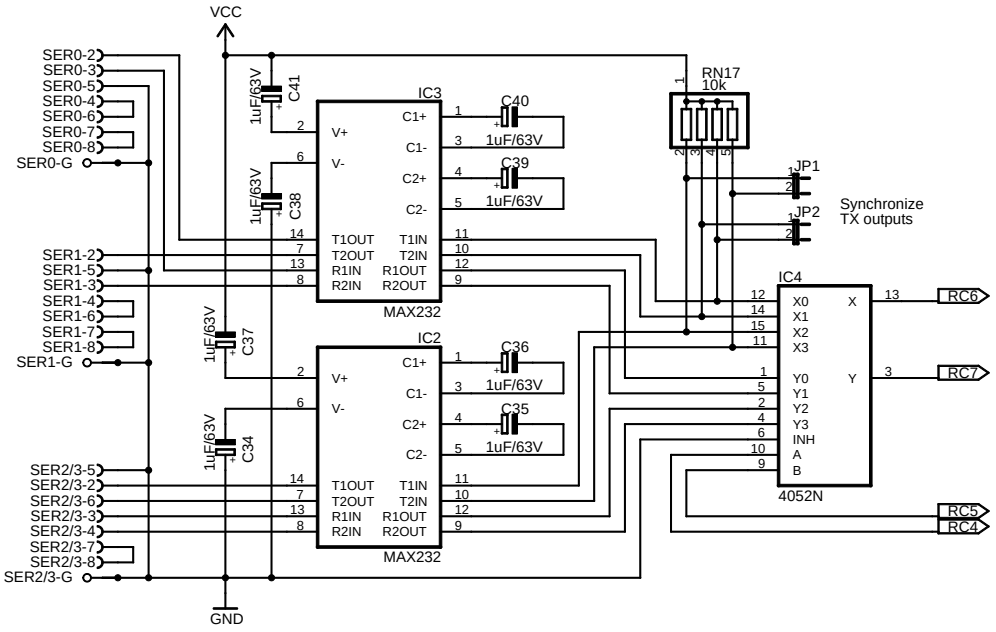


Abbildung 3.16: Ansteuerung der seriellen Schnittstellen

Ein Analog Switch vom Typ 4052 übernimmt die Aufgabe des Multiplexers für die Pins RX und TX des Mikrocontrollers, der aktive Kanal kann über zwei weitere Pins ausgewählt werden. Am Ausgang des Multiplexers halten Pullup-Widerstände inaktive Leitungen auf dem Ruhepegel, bevor zwei MAX232 Pegelwandler die Spannungen für Anschlußbuchsen umsetzen. Die Bausteine enthalten auch alle nötigen Schutzvorrichtungen zur Absicherung der Schnittstellen.

Über Jumper können die Sendeleitungen der Platine und damit die Eingänge der Steuerrechner synchronisiert werden. So kann ein inaktiver Rechner die Antworten der Firmware mitlesen, wenn diese gerade auf einem anderen Kanal kommuniziert. Diese Funktion ist wichtig, wenn redundante Rechner die Leistungselektronik mit Hot Standby steuern sollen. Im Layout der Platine ist nicht genügend Platz für vier Buchsen vorhanden, daher ist eine doppelt belegt. Sie

enthält auf den Pins DTR und DSR die vierte Schnittstelle, in Anhang A.8 ist eine Kabelpeitsche beschrieben, welche die Schnittstellen wieder trennt.

3.7.1 Suche nach einem Steuerrechner

Wenn ein Rechner die Kontrolle über die Leistungselektronik übernehmen will, sendet er dazu einen entsprechenden Befehl über die serielle Schnittstelle. Das Problem hierbei ist, daß die Firmware nicht wissen kann, auf welchem Eingang sie diesen empfangen kann. Daher gibt es einen Suchmodus, in dem der Mikrocontroller sehr schnell alle Eingänge reihum auf Aktivität prüft. Dabei konfiguriert er den RX-Pin als digitalen Eingang und sucht nach einem Low-Pegel (die Leitungen liegen im Ruhezustand auf High). Entdeckt der Mikrocontroller Aktivität auf einem Kanal, beendet er die Suche, wechselt den Eingang nicht mehr und aktiviert seinen seriellen Transceiver. Ab dann läuft die Kommunikation nach dem RS232-Standard. Dieses Verfahren reicht alleine aber noch nicht aus. Egal wie schnell die Schnittstellen abgefragt werden, wenn der Transceiver aktiviert wird, hat dieser bereits den Anfang des Bytes verpaßt und es kommt zu einem Fehler. Die Firmware löst dieses Problem mit einem Trick.

Randbedingungen

- Das Protokoll schreibt vor, daß jedes Paket mit dem Byte 0xF0 beginnt. Auf der Leitung wird es als 5 Takte Low (Startbit plus untere 4 Bits), gefolgt von 5 Takten High (obere 4 Bits plus Stopbit) übertragen. Dies verschafft dem Suchmodus eine ausreichend lange ununterbrochene Low-Phase, um den Kanal finden zu können.

Suche nach Aktivität

- Alle 4 Eingänge werden nach dem beschriebenen Muster laufend abgefragt, wobei ein Test wenigstens alle 45 Mikrosekunden stattfindet. Ist ein Eingang zu dem Zeitpunkt Low, wird der Kanal vorläufig nicht mehr gewechselt.
- Der Pegel muß wieder auf High wechseln, wenn die zweite Hälfte des Bytes beginnt. Passiert dies nicht rechtzeitig, wechselt die Suche wieder den Kanal.
- Bei Erfolg wird der serielle Transceiver aktiviert, der jetzt ausreichend Zeit für die Initialisierung hat und die Daten ab dem zweiten Byte korrekt empfängt.

Herstellen und Trennen der Verbindung

- Das fehlende erste Byte aus dem Header ergänzt die Software intern.
- Innerhalb von 0,03 Sekunden muß der erst gültig Befehl vollständig empfangen sein, anderenfalls wird die Verbindung wieder getrennt. Solange bleiben die Motortreiber noch deaktiviert, um ein ungewolltes Anfahren der Züge zu verhindern.
- Ab dann läuft die Kommunikation normal, bis der externe Rechner sie explizit beendet oder für 0,2 Sekunden lang gar keinen gültigen Befehl sendet.

Der Algorithmus ist so konzipiert, daß ein korrekter Verbindungsaufbau normalerweise bereits beim ersten Versuch erfolgreich ist. Auf der anderen Seite dürfen Fehler auf einem Kanal nicht die gesamte Kommunikation blockieren und funktionierende Rechner aussperren. Daher wird bei Anzeichen gravierender Probleme die Verbindung möglichst schnell wieder unterbrochen.

3.7.2 Kommunikation

Ist die Verbindung erst einmal hergestellt, übernimmt der Transceiver im Mikrocontroller den größten Teil der Ansteuerung. Alle empfangenen Daten werden in einem Ringpuffer von 16 Bytes gespeichert und auf gültige Pakete hin durchsucht.

Empfang von Daten

- Jedes korrekt empfangene Byte wird zunächst in dem Ringpuffer abgelegt.
- Anschließend überprüft eine Funktion, ob am Anfang des Puffers ein gültiges Paket enthalten ist, und liefert eine der Antworten „Ja“, „Nein“ und „Vielleicht“.
- Bei der Antwort „Ja“ wird der empfangene Befehl ausgeführt, ein Paket mit der passenden Antwort erzeugt und der Befehl selbst aus dem Puffer entfernt.
- Die Antwort „Vielleicht“ bedeutet, daß nicht ausreichend Daten für eine Entscheidung im Puffer enthalten ist. In diesem Fall wartet die Firmware auf weitere Bytes.
- Bei der Antwort „Nein“ wird das erste Byte des Puffers gelöscht. Dadurch kann die Kommunikation hinter einem defekten Paket irgendwann wieder aufsetzen.
- Sollte der Transceiver einen Übertragungsfehler (Overrun, Framing Error) melden, wird der gesamte Inhalt aller Empfangspuffer gelöscht. Das gleiche passiert, wenn innerhalb eines Befehls eine ungewöhnlich lange Pause auftritt.

Die gesamte Kommunikation arbeitet nach dem Prinzip fail silent. Ist der empfangene Befehl korrekt, wird er von der Firmware an den entsprechenden Handler weitergeleitet. Dieser kümmert sich um die Ausführung und generiert ein Antwortpaket, das zunächst in einem 16 Bytes großen Ringpuffer abgelegt wird. Die Firmware leert den Puffer laufend in Richtung des Transceivers. In allen anderen Fällen wird der Befehl ignoriert. Das Steuerprogramm kann beim Ausbleiben einer Antwort nicht feststellen, ob der Befehl ausgeführt wurde.

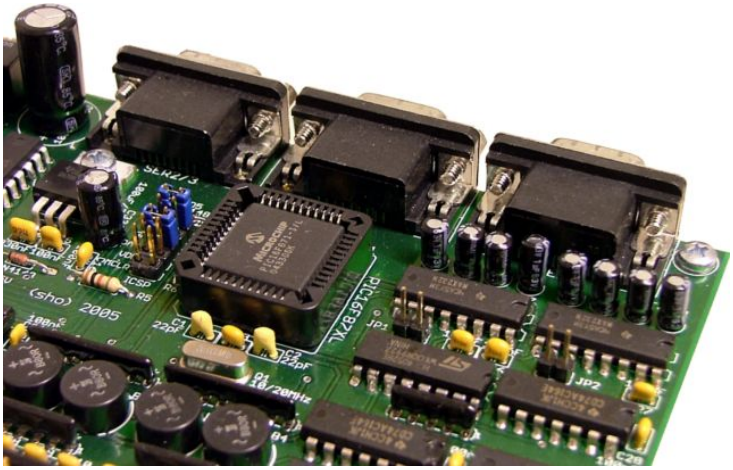


Abbildung 3.17: Multiplexer, Jumper und Treiber der seriellen Schnittstellen

Ein Steuerrechner muß laufend Befehle senden, um die Verbindung aufrecht zu halten. Sollte sie getrennt werden, kann der selbe oder ein anderer Rechner diese wieder übernehmen. Auf Wunsch passiert das nahtlos, also ohne Reset der Leistungselektronik. Damit ist eine redundante Steuerung möglich, auch wenn immer nur ein Rechner zur Zeit Befehle senden darf.

Die Ansteuerung erlaubt ausdrücklich, einen Befehl zu senden, während die Firmware noch die Antwort auf den vorigen überträgt (Vollduplex-Betrieb). Dabei ist nur darauf zu achten, daß der Sendepuffer des Mikrocontrollers nicht überläuft. Dies kann passieren, wenn die Befehle kleiner als die Antwortpakete sind und völlig ohne Pausen gesendet werden. In so einem Fall kommt die Firmware mit dem Senden nicht schnell genug hinterher und muß Befehle ignorieren, wenn im Sendepuffer kein Platz für das Antwortpaket ist.

3.8 Integrierte Anzeige

Die 7-Segment-Anzeige der Leistungselektronik wird nicht für die Steuerung der Bahn gebraucht, sie dient lediglich als Funktionskontrolle und zur Information des Benutzers. Der Dezimalpunkt leuchtet unabhängig vom Zustand der Schaltung, sobald der Logikteil mit Strom versorgt wird.

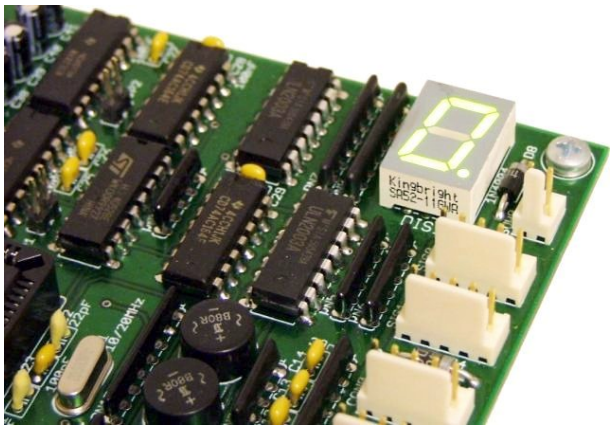


Abbildung 3.18: Anzeige der Leistungselektronik

Die anderen Segmente zeigen Informationen über den internen Zustand der Firmware an. Die folgende Liste gibt alle Möglichkeiten mit aufsteigender Priorität an.

- Am auffälligsten ist die Animation im Suchmodus. Die Firmware wartet darauf, daß ein angeschlossener Steuerrechner eine Verbindung aufbaut. Solange diese Suche andauert, wandert ein einzelnes Segment entlang einer Acht durch die Anzeige.
- Besteht hingegen eine Verbindung, wird die Nummer des aktiven Einganges angezeigt. Dabei handelt es sich um eine Zahl zwischen 0 und 3.
- Solange ein Gleistreiber von der Kurzschlußsicherung abgeschaltet ist, erscheint in der Anzeige ein schnell blinkendes E. Dieser Zustand bleibt bestehen, bis der Fehler beseitigt ist und die Gleistreiber wieder normal arbeiten.
- Wenn die Anzeige im laufenden Betrieb flackert, deutet dies in der Regel auf eine instabile Kommunikation mit zu selten eintreffenden oder ungültigen Paketen hin.

Die Anzeige wird von dem Mikrocontroller genau wie die Signale über ein Schieberegister und einen Treiberbaustein vom Typ ULN2003 angesteuert, um Pins zu sparen. Über RC1 wird die Anzeige gelöscht, RC2 und RC3 erlauben es, ein Byte seriell in das Register zu laden. Dessen Ausgänge steuern die Open-Collector-Treiberstufe und lassen so Strom von der Versorgung des

Logikteils durch das Display und den Vorwiderstand fließen, wodurch die Segmente aufleuchten. Während das Byte zum Schieberegister übertragen wird, rotieren die Segmente, bis sie ihre Endstellung erreicht haben. Dies geschieht aber so schnell, daß es nicht sichtbar ist.

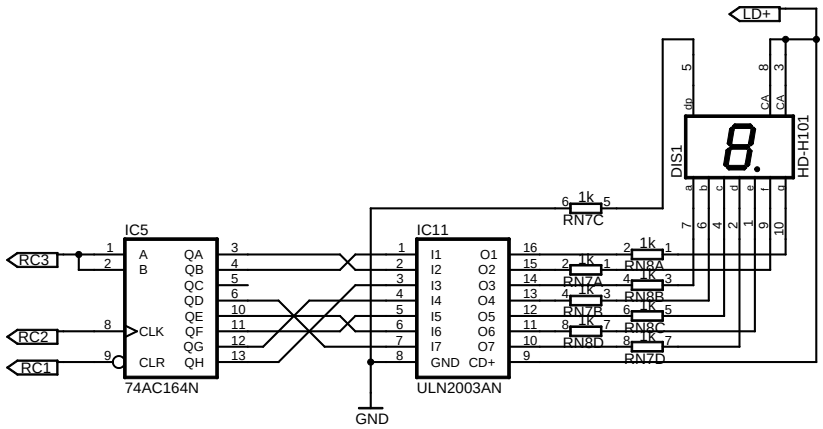


Abbildung 3.19: Ansteuerung der integrierten Anzeige

Die Anzeige wird aktualisiert, wenn sich der Systemzustand geändert hat oder wenn seit dem letzten Mal 0,25 Sekunden vergangen sind. Sollte das Display durch eine gestörte Taktleitung falsche Daten anzeigen, korrigiert es sich so schnell von selbst.

3.9 Fehlerprotokoll

Der Mikrocontroller verwaltet insgesamt 13 Fehlerzähler, die Ereignisse wie den Ausfall eines einzelnen Reedkontaktes protokollieren können. Jeder Zähler ist einer Quelle zugeordnet und wird bei dem Eintreten eines Ereignisses einmal inkrementiert, bis er schließlich 255 erreicht. Die Zähler befinden sich im EEPROM und bleiben so auch ohne Stromversorgung erhalten. Der Benutzer kann die Zähler jederzeit mit einem Diagnoseprogramm auslesen und zurücksetzen, dafür muß nur eine serielle Kommunikation aufgebaut werden. Fehler der Hardware können auf diese Weise frühzeitig entdeckt und beseitigt werden, bevor sie sich negativ bemerkbar machen.

- Wenn in einem Kontakt nur einer der beiden Reedkontakte betätigt wird, kann dies auf einen Fehler wie eine unterbrochene Leitung, einen zerbrochenen Reedkontakt oder einen zu schwachen Magneten am Zug hindeuten. Die Firmware erhöht den Zähler desjenigen Reedkontaktes, der nicht ausgelöst hat, braucht dafür also insgesamt 8 Zähler.

- Ein Abschalten der Gleistreiber ist immer mit einem echten Problem verbunden, meistens einem Kurzschluß der Gleise durch eine falsch gestellte Weiche, falsch gepolte Blöcke oder Fremdkörper auf den Schienen. Jeder Kurzschluß erhöht den Zähler des Gleistreibers, hält der Kurzschluß länger an, wird der Zähler mit jeder Reaktivierung des Treibers weiter erhöht, bis die Ursache beseitigt ist. Es ist hier sinnvoll, nicht nur einmal zu inkrementieren, damit länger andauernde Kurzschlüsse entsprechend deutlich im Protokoll auftauchen. Für jeden der beiden Gleistreiber ist ein Zähler vorgesehen.

- Die dritte und letzte Gruppe von Fehlern bilden unerwünschte Resets des Mikrocontrollers. Sie können durch einen Impuls auf der Reset-Leitung, ein Timeout des Watchdog-Timers oder ein kurzfristiges Absacken der Versorgungsspannung ausgelöst werden. Beim Start der Firmware wird untersucht, ob es sich dabei um einen Kaltstart handelt oder eine der genannten Ursachen vorliegt. Wenn ja, wird der entsprechende Zähler inkrementiert. Resets deuten auf elektronische Probleme oder eine abgestürzte Firmware hin und sind entsprechend ernst zu nehmen, falls sie einmal auftreten sollten.

Die Interpretation der Kontaktfehler in der Bahnanlage ist einfach. Viele Fehler an einer Stelle bedeuten meistens, daß der jeweilige Kontakt beschädigt ist. Über das ganze System verteilte Fehler sind hingegen ein Indiz für einen zu schwachen oder falsch befestigten Zugmagneten. Einige wenige Fehler sind aber durchaus normal und entstehen zum Beispiel beim Schieben der Züge von Hand, beim Halt direkt über einem Kontakt oder wenn Züge zu langsam fahren.

Kapitel 4

Vernetzung

In der Vergangenheit der Bahnanlage hat sich gezeigt, daß eine verteilte Steuerung besser als eine zentrale geeignet ist. Die Komplexität bei der Ansteuerung der 245 Sensoren und Aktoren bleibt so in beherrschbaren Grenzen, das System ist erweiterbar und eventuell auftretende Fehler sind einfacher zu beseitigen. Dafür müssen die Steuerungselemente miteinander verbunden werden, eine entsprechende Kommunikation ist für den Betrieb der verteilten Anlage unverzichtbar.

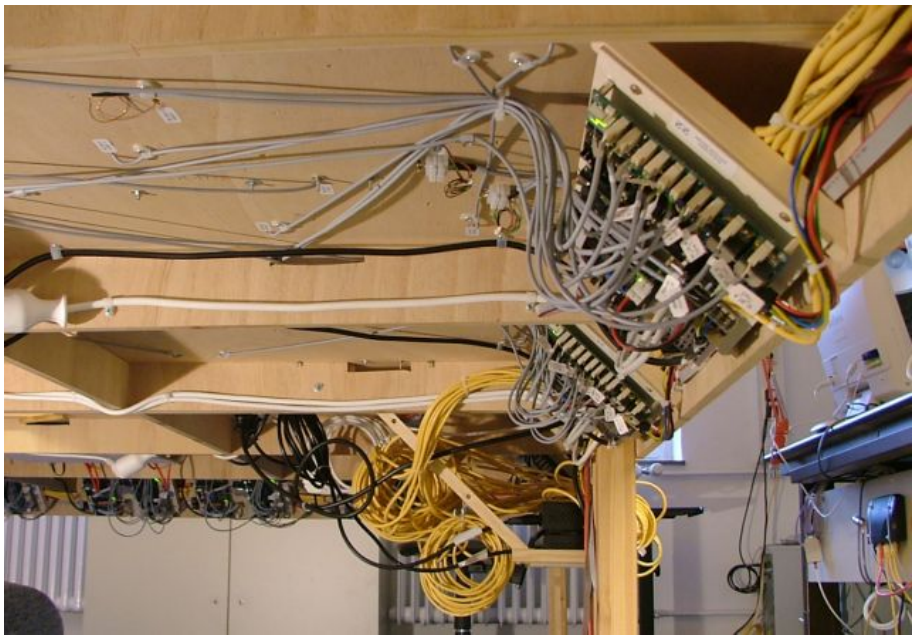


Abbildung 4.1: Verkabelung unter der Bahnplatte

In [Koberstein] wurde dafür jede Leistungselektronik mit einer Interbus-Schnittstelle versehen. Diese konnte Befehle von einem externen Rechner zu den Platinen und Sensorwerte zurück zum Rechner transportieren. Der Nachteil dieser Lösung war, daß die Steuerung untrennbar an ein Bussystem gekoppelt wurde und sich im Nachhinein nicht mehr austauschen ließ. Diese Arbeit wählt stattdessen den Ansatz, eine weitere Schnittstelle zwischen Elektronik und das Businterface zu setzen. Auf diese Weise lassen sich beliebige Steuersysteme anbinden und ohne Veränderungen an der Hardware abwechselnd einsetzen. Diese Modularisierung eröffnet viele Anwendungsmöglichkeiten, die mit der alten Steuerung nicht möglich gewesen wären.

4.1 Modularer Aufbau

Die eleganteste Lösung für das beschriebene Zwischeninterface sind serielle Schnittstellen im RS232-Standard. Sie sind in fast allen eingebetteten Systemen vorhanden, einfach anzusteuern und verfügen über eine ausreichende Bandbreite. Es werden weder besondere Kabel noch eine aufwendige Elektronik benötigt, so daß mehrere Schnittstellen auf einer Platine Platz finden. An diese können als Brücken arbeitende Systeme angeschlossen werden, die ihrerseits die Verbindung mit den eigentlichen Bussystemen herstellen.

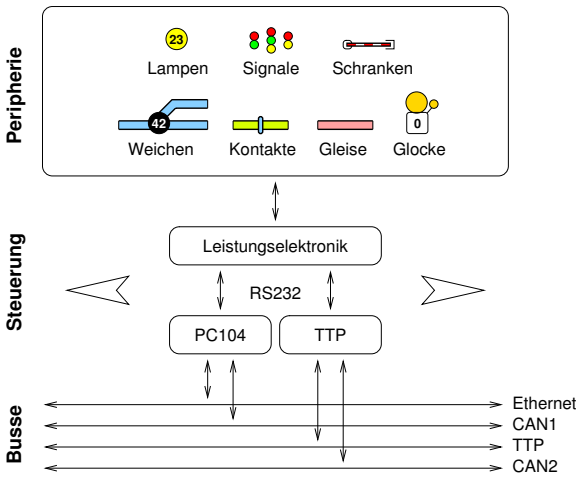


Abbildung 4.2: Modulares Steuersystem der Modellbahn

Jede Leistungselektronik verfügt über vier Schnittstellen, an die entsprechend viele Steuerrechner angeschlossen werden können. In der aktuellen Ausbaustufe sind zwei davon belegt.

- Unter Linux laufende PC104-Rechner sind mit einen Port verbunden. Untereinander und mit der Außenwelt können sie Verbindungen über Ethernet und CAN herstellen.
- Der andere Port bindet Powernodes der Firma TTTech an, die über das zeitgesteuerte Protokoll TTP kommunizieren und zusätzlich über einen CAN-Bus verfügen.

Die Leistungselektronik überwacht ständig alle Ports, bis sie auf einem Befehle empfängt. Dann verbindet sie sich fest mit dem sendenden Rechner und hält die Verbindung, bis der Rechner sie trennt oder für eine gewisse Zeit keine (gültigen) Befehle mehr sendet. Auf diese Weise können die Rechner abwechselnd die Kontrolle übernehmen, ohne daß an der Hardware oder der Konfiguration des Systems etwas geändert werden müßte. Das Design unterstützt redundante Steuerrechner, solange diese sich *fail silent* verhalten. Das Ausbleiben von Befehlen führt zu einer Trennung der Verbindung, die Leistungselektronik wechselt vorläufig in einen sicheren Zustand und beginnt erneut mit der Suche. Nun hat ein weiterer Rechner die Chance, wieder die Kontrolle zu übernehmen und nahtlos weiterzuarbeiten.

Die vielen Rechner unter der Bahn sind nicht auf den Einsatz als Paketvermittler zwischen einem Bussystem und den Leistungselektroniken beschränkt, sie können zusätzlich Aufgaben wie die Organisation des Fahrbetriebes verteilt ausführen. Diese Möglichkeit gekoppelt mit dem Einsatz unterschiedlichster Bussysteme und Entwicklungswerkzeuge macht die Modellbahn zu einem vielseitigen und attraktiven Labor.

4.2 PC104-Rechner

Das erste Steuersystem der Bahn bilden Rechner im PC104-Format. Dieser Standard für PC-kompatible eingebettete Systeme legt den Formfaktor und das Businterface fest, alle Platinen sind 90 mal 96 Millimeter groß und werden in einem Stapel übereinander gesteckt. Der ISA-Bus läuft dabei über Stift- und Buchsenleisten vertikal durch den Stapel und verbindet das in der Regel oben liegende CPU-Board mit eventuellen Zusatzplatinen. Durch diesen Aufbau sind die Rechner sehr kompakt und robust, so daß sie viel in rauhen Umgebungen in der Industrie eingesetzt werden. In der Anlage sind 24 EmCore-i313DVL-4S der Firma Arbor integriert.

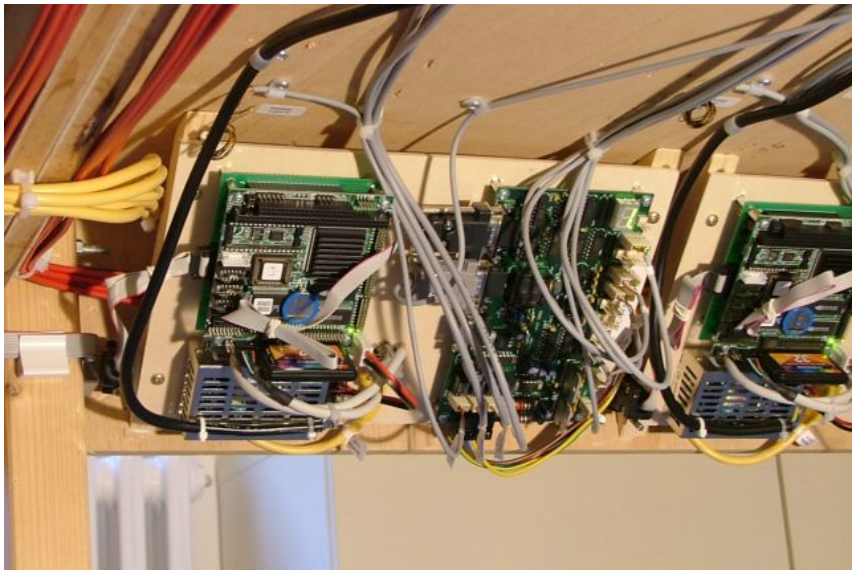


Abbildung 4.3: PC104-Modul in der Anlage

Die Rechner befinden sich zusammen mit einem 5 Volt-Netzteil und einer Leistungselektronik auf diagonal aufgehängten Holzplatten. Die schräge Montage vergrößert den verfügbaren Platz und sorgt dafür, daß die Platinen für Wartungsarbeiten gut erreichbar sind. Der Hohlraum hinter den Verstrebungen wird als Kabelkanal genutzt, der die verschiedenen Versorgungsleitungen und Bussysteme aufnehmen kann. Die PC104-Rechner selber verfügen über die folgende Ausstattung.

CPU-Board

- 386SX-kompatible CPU ohne Fließkommaeinheit, 40 MHz (ALi M6117), Watchdog
- 4 MB EDO DRAM
- Onboard-Grafik basierend auf TP6508IQ/CT65545 mit 1 MB RAM, Auflösungen bis 1024x768 Punkte, Ausgänge für VGA-Monitor und LCD
- EIDE-Controller zum Anschluß von 2,5 Zoll-Festplatten
- Floppy-Controller Diskettenlaufwerke
- Flash/EPROM/DOC2000-Interface (Disc on Chip 2000: 16 MB bis 1GB)
- Realtek 8029AS Netzwerkchip (Ethernet, Full Duplex, 10 MBit/s)
- 16 Bit I/O-Interface (je 8 Ein- und Ausgänge)
- Vier serielle Schnittstellen (basierend auf 16C550, zwei als RS-485 konfigurierbar)

- Eine parallele Schnittstelle (SPP/EPP/ECP-fähig)
- Anschlüsse für Tastatur und Maus (PS/2)

Festplatte

- 2,5 Zoll Flash-Disc mit 32 MB Kapazität

Erweiterungsplatinen

- CAN-Controller mit einem Kanal von Peak System

Die Rechner sind untereinander per Ethernet verbunden, zwei 100 MBit/s-Switches stellen einen ausreichend schnellen Uplink zum Server bereit. Die Rechner booten über das Netzwerk, sie mounten das Root-Dateisystem sowie die Benutzerverzeichnisse von einem Server und nutzen dort vorhandene Dateien als Swapspace. Die Festplatte wird nur am Anfang des Bootprozesses und für Wartungsarbeiten benötigt.

Booten von Festplatte

- Auf der Festplatte ist der Bootmanager `grub` installiert. Er bietet auf der per Monitor und Tastatur erreichbaren Konsole ein Menü mit drei Alternativen an, bevor er nach drei Sekunden den Bootvorgang per Netzwerk startet.
- Die erste Option startet ein Diskettenimage von der Festplatte, in dem FreeDOS und ein DOS-basierendes Konfigurationsprogramm für den Netzwerkchip installiert sind.
- Als zweite Option kann ein auf der Festplatte installiertes Linux gebootet werden. Dieses verfügt über eine Systempartition von 25 MB sowie 4 MB Swapspace und kann unabhängig vom Server für Testzwecke benutzt werden.
- Als letzte Option startet `grub` den Bootvorgang über das Netzwerk.

Booten per Netzwerk

- Da die Netzwerkkarte über kein eigenes Boot-ROM verfügt, übernimmt ein Image aus dem etherboot-Projekt dessen Aufgabe. Es initialisiert die Hardware und holt sich von einem DHCP-Server die nötige Konfiguration. Dazu gehört auch die Adresse des Linux-Kernels, der anschließend per TFTP geladen und ausgeführt wird.
- Der Kernel führt eine erneute DHCP-Anfrage durch, bei der ihm die Adresse seines Root-Dateisystems mitgeteilt wird. Jeder Rechner verfügt über ein eigenes System, der Server weist seinen Clients das richtige nach ihrer MAC-Adresse zu.
- Das Dateisystem wird gemountet und der darin enthaltene Init-Prozeß gestartet. Dieser kümmert sich um die restliche Initialisierung, mountet Benutzerverzeichnisse mit den Anwendungsprogrammen sowie den Swapspace.
- Die einzelnen Rechner können über ihren Namen oder ihre IP-Adresse die eigene Nummer innerhalb der Anlage ermitteln. Sie starten am Ende des Bootvorganges einen RSH-Server für den Fernzugriff sowie einen Daemon, der für die Verbindung zwischen Ethernet, CAN und den Leistungselektroniken benutzt werden kann.

Alle Applikationen für die PC104-Rechner müssen mit einem Crosscompiler übersetzt werden, der den 386er als Zielplatform unterstützt und eine platzsparende Version der `libc` statt der normalen GNU `libc` benutzt. Der gesamte Prozeß, in dem Crosscompiler und das Betriebssystem der Knoten erstellt wird, ist in Anhang C beschrieben.

4.3 Ethernet und TCP/IP

Die PC104-Rechner müssen für den normalen Betrieb bereits vernetzt sein. Das Ethernet kann darüber hinaus natürlich zur Steuerung der Bahn benutzt werden, wenn dabei die Last nicht zu hoch wird. Solange die Rechner weder intensiv auf das Dateisystem noch den Swapspace zugreifen müssen, ist dies aber kein Problem. Als Protokoll eignet sich das verbindungslose UDP, das von allen gängigen Betriebssystemen unterstützt wird. Bei diesem können zwar im Gegensatz zu TCP Pakete verloren gehen oder in falscher Reihenfolge ankommen, dafür sind die Laufzeiten begrenzt, was Echtzeitanwendungen wie der Bahnsteuerung sehr entgegenkommt.

Abbildung 4.4: Ethernet-Switches und Hauptschalter am Rand der Bahn

Das Foto in Abbildung 4.4 zeigt die beiden 16 Port-Switches, an denen die Patchkabel aller Rechner zusammenlaufen. Sie arbeiten mit 100 MBit/s, wovon die PC104-Rechner aber nur ein Zehntel ausnutzen. Dafür ist der Uplink zum Server schnell genug, wovon datenintensive Aufgaben wie das gleichzeitige Booten aller 24 Rechner deutlich profitieren.

4.4 Der CAN Feldbus

Der Controller Area Network genannte Bus wurde ab 1983 von der Firma Bosch zur Vernetzung im Automobilbereich entwickelt. Er funktioniert nach einem einfachen aber eleganten Prinzip und ist sehr robust gegenüber elektromagnetischen Störungen. Seit seiner Vorstellung im Jahre 1986 wird CAN in vielen industriellen Bereichen eingesetzt, 1996 wurde der Bus von der ISO standardisiert. Die aktuelle Entwicklung findet in dem Industriegremium [CiA] statt.

4.4.1 Allgemeines

Das Protokoll ist auf dem Data Link Layer angesiedelt. An einem Bus können prinzipiell beliebig viele gleichberechtigte Stationen angeschlossen werden. Jede von ihnen darf Daten senden, die

von allen anderen empfangen werden. Eine Möglichkeit zur Adressierung gibt es nicht, dafür trägt jede Nachricht neben der Nutzlast eine Identifikationsnummer. Sie legt die Priorität und den Inhalt der Nachricht fest und kann von der Anwendung vorgegeben werden. Die Controller sind selbst selbst dafür zuständig, die Integrität der Daten sicherzustellen, Prioritäten umzusetzen und Kollisionen durch spätere Wiederholung einer Übertragung aufzulösen.

4.4.2 Datenformat

Die wichtigste Grundlage für diese Funktionen ist die Codierung der einzelnen Bits. Der Wert 1 wird auf dem Bus rezessiv, 0 hingegen dominant übertragen. Senden zwei Stationen zur gleichen Zeit unterschiedliche Bits, wird sich das dominante gegen das rezessive Bit durchsetzen. Beide Stationen vergleichen den Buszustand mit den von ihnen gesendeten Daten. Stimmen sie nicht überein, bricht der unterliegende Sender seine Übertragung ab und wiederholt sie automatisch nach einer gewissen Wartezeit. Auf diese Weise werden Prioritäten durchgesetzt und Nachrichten quittiert. Kollisionen behindern laufende Übertragungen höherer Priorität nicht und der Mechanismus stellt sicher, daß Latenzzeiten berechenbar sind.

Eine Nachricht enthält neben einigen Steuerbits und der Identifikationsnummer die Anzahl der enthaltenen Datenbytes, 0-8 Bytes Nutzdaten, eine CRC-Checksumme und ein ACK-Bit. Dank der rezessiven und dominanten Bits wird sich diejenige Nachricht durchsetzen, deren ID am kleinsten ist, also in den führenden Bits als erstes eine Null aufweist. Jede gesendete Nachricht muß von mindestens einem Empfänger bestätigt werden. Dazu ist im gesendeten Paket das ACK-Bit rezessiv und wird vom Empfänger dominant überschrieben, was der Sender feststellen kann. Passiert dies nicht, gilt die Übertragung als gescheitert und wird später wiederholt. Der Bus funktioniert somit nur, wenn mindestens zwei Stationen aktiv sind.

Wenn ein Sender sehr viele Fehler feststellt, schaltet er sich nach einer gewissen Zeit passiv, greift also nicht mehr aktiv in den Bus ein. Der Sinn dieser Maßnahme liegt darin, daß eine defekte Station nicht die gesamte Kommunikation blockieren darf. Hilft auch dies nicht, trennt die Station sich komplett vom Bus und stellt die Arbeit ein. Diese beiden Fehlerzustände sind fast die einzigen, mit denen Anwendungsprogramme konfrontiert werden können.

Es gibt zwei Varianten des Protokolls, die sich hauptsächlich in der Länge der Identifikation unterscheiden. Das ältere CAN 2.0A beschränkt sich auf 11 Bit (Standard Frames), CAN 2.0B benutzt hingegen 29 Bit (Extended Frames).

4.4.3 Codierung

Auf der untersten Ebene kommt eine NRZ-Codierung zum Einsatz, die Bits werden als konstante Pegel dargestellt. Falls irgendwann fünf gleiche Bits in einer Sequenz auftreten, fügt der Sender anschließend ein komplementäres Bit ein, um lange konstante Sequenzen zu verhindern (Bit Stuffing). Diese Signale werden differentiell auf zwei Leitungen übertragen, um die Übertragung robust gegen Störungen zu machen. Für viele Aufgaben reicht ein Flachbandkabel als Busleitung aus, besser sind verdrehte und abgeschirmte Kabel. Der Anschlußstecker der einzelnen Stationen ist von der CiA standardisiert (vergleiche Abbildung 4.5).

Bei einer Kabellänge von maximal 40 Metern erreicht der Bus seine Höchstgeschwindigkeit von 1 MBit/s (ISO11898-2 high speed), bei längeren Kabeln sinkt der Durchsatz entsprechend. Auf einem Kilometer sind beispielweise noch 50 kBit/s zu erreichen. Der Bus muß unabhängig von der Datenrate und Kabellänge an beiden Enden terminiert werden, dafür wird ein Widerstand von 120 Ohm zwischen CAN Low und CAN High angeschlossen.

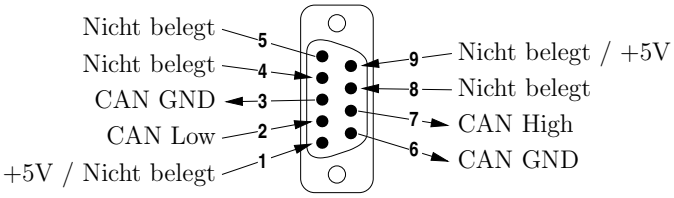


Abbildung 4.5: Pinbelegung nach CiA/DS 102-1, Stecker auf Rechnerseite

4.5 TTP Powernodes

Das zweite Steuersystem bilden acht Powernodes der Firma TTTech. Diese kommunizieren über TTP und werden normalerweise bei der industriellen Entwicklung von Prototypen eingesetzt. Im Gegensatz zu den PC104-Rechnern ist hier wenig eigene Vorarbeit zu leisten, der Hersteller liefert die Knoten zusammen mit einem vollständigen Entwicklungssystem aus. Die Software wird modellbasiert mit Matlab unter Windows erstellt und anschließend auf die Knoten geladen.

Abbildung 4.6: Zwei TTP-Knoten in der Anlage

Die Knoten in der Bahn unterhalten jeweils Verbindungen zu drei Leistungselektroniken. Da der Powernode selbst nur eine serielle Schnittstelle enthält, muß diese über einen Multiplexer mit der anzusteuernden Leistungselektronik verbunden werden. Die hierfür benutzte Schaltung ist in Anhang A.7 beschrieben. Ein neunter Knoten außerhalb der Anlage kann für andere Aufgaben wie die Berechnung und Ausführung von Fahrplänen benutzt werden. Neben ihm befindet sich auch der Monitoring Node, der zum Hochladen der Programme und zur Überwachung des Busses gebraucht wird. Die Powernodes zeichnen sich durch folgende Eigenschaften aus.

- CPU Motorola MPC555, PowerPC-Architektur mit Fließkommaeinheit, 40 MHz
- 1 MB RAM, 4 MB Flash-Speicher für Programmcode
- Viele analoge und digitale Schnittstellen zur Anbindung der Peripherie

- Leuchtdioden auf der Frontplatte dienen zur Statusanzeige
- Ein frei programmierbarer CAN-Bus

Der CAN-Bus ist für die Programmierung besonders interessant, da er sich für Statusausgaben oder zum Einspeisen von Befehlen, Fahrplandaten und ähnlichem in ein laufendes TTP-System eignet. Er ist von der Operatorkonsole aus erreichbar, dort stehen unter anderem auch die zusätzlichen Knoten des TTP-Systems.

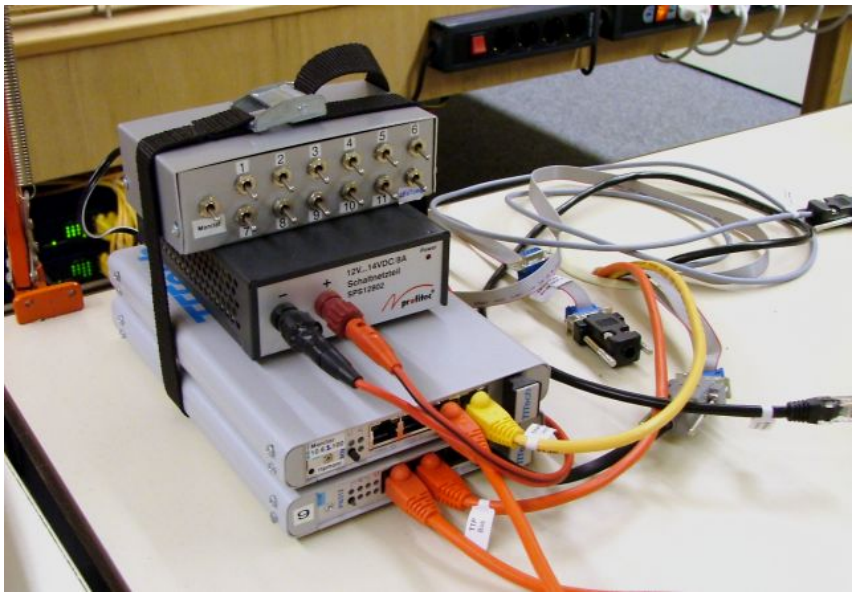


Abbildung 4.7: Anschlußleitungen auf der Operatorkonsole

Der große Vorteil der TTP-Architektur liegt darin, daß das gesamte Zeitverhalten der Software und der Kommunikation bei der Entwicklung spezifiziert wird. Das generierte System hält sich später an die Vorgaben und verfügt so über deterministische Eigenschaften.

4.6 Vernetzung der Anlage

Die Bahnplatte besteht aus vier Tischen, die auf eigenen Beinen stehen und fest miteinander verschraubt sind. Jeder Tisch ist auf der Unterseite in drei mal drei Segmente unterteilt, die Querverstrebungen dazwischen sorgen für die nötige Stabilität. Die verschiedenen Rechner sind gleichmäßig auf die vier Tische verteilt, am Außenrand befinden sich jeweils sechs PC104-Rechner mit einer Leistungselektronik, in der Plattenmitte sind pro Tisch zwei Powernodes aufgehängt. Die genaue Anordnung illustriert Abbildung 4.8.

4.6.1 Steuerrechner

Die Verteilung der Leistungselektroniken und damit auch der PC104-Rechner orientiert sich daran, wo die meisten Anschlüsse für die Peripherie benötigt werden. Gleise, Signale, Kontakte, Weichen und ähnliches sind immer mit der nächsten freien Leistungselektronik verbunden, um lange und damit unübersichtliche Kabel zu vermeiden. Dabei kreuzt keines eine Tischgrenze,

damit die Anlage sich im Falle eines erneuten Umzuges mit vertretbarem Aufwand in vier tragbare Einzelteile zerlegen läßt. Jedes Steuerprogramm muß über eine Tabelle verfügen, in der für jedes Peripherieteil festgehalten ist, an welchen Anschluß welcher Leistungselektronik dieses angeschlossen ist und über welchen Rechner es erreichbar ist. Die Tabelle für den aktuellen Aufbau steht im Anhang B.2.

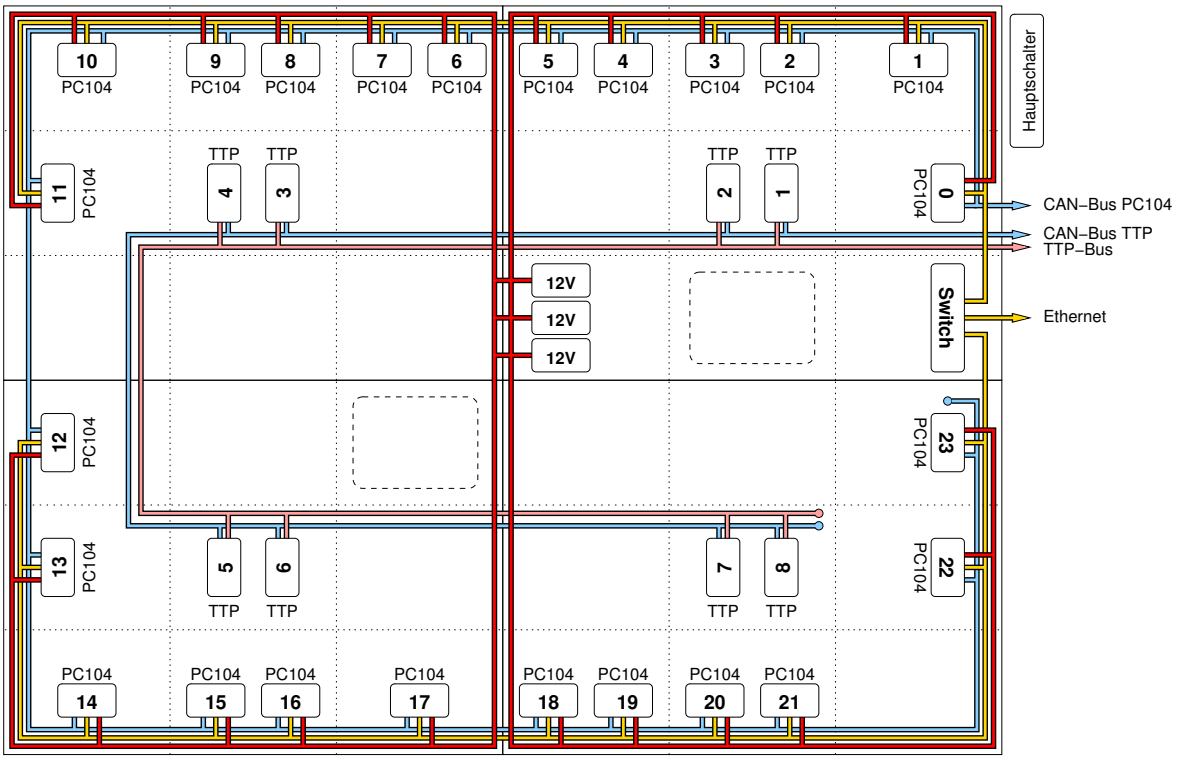


Abbildung 4.8: Versorgungsleitungen und Busse unter der Modellbahn

4.6.2 Bussysteme

Von jedem PC104-Rechner verläuft ein Ethernet-Kabel zu den rechts gelegenen Switches, am oberen und unteren Rand sammeln sich entlang des Weges jeweils 12 Kabel zu einem Bündel. Neben ihnen führt ein CAN-Bus rund um die Bahn und verbindet die Rechner miteinander. Die TTP-Knoten sind durch einen umlaufenden TTP- und einen CAN-Bus in der Anlagenmitte vernetzt. Alle Leitungen werden am Rand der Platte gemeinsam herausgeführt, Benutzer können so weitere Hardware und die dort installierten PCs anschließen.

4.6.3 12 Volt-Versorgung

Die Leistungselektroniken beziehen ihren Strom von drei 12 Volt-Schaltnetzteilen in der Mitte der Platte, die jeweils einen Strom von 10 Ampere liefern können. Ein Netzteil versorgt Weichen, Lampen, Schranken und die Glocke, das zweite liefert den Fahrstrom für die Züge und das dritte versorgt die Platinen selbst sowie die übrige Peripherie. Sechs Adern mit einem Querschnitt von 2,5 Quadratmillimetern verbinden die Netzteile und die Leistungselektroniken. Die Kabel müssen dick sein, damit die Spannung möglichst ohne Verluste alle Platinen erreicht. Jeder Tisch verfügt über einen eigenen Kabelbaum, was die Leitungslängen kurz hält. In dem Foto in

Abbildung 4.9 ist die Netzteilgruppe zu sehen. Ventilatoren an beiden Seiten sorgen für einen kühlenden Luftstrom, dessen Stärke mit dem Schalter an der Unterseite der Platte in zwei Stufen verändert werden kann.

Abbildung 4.9: 12 Volt-Schaltnetzteile unter der Anlage

4.6.4 Netzspannung

Der Strom für die Anlage kommt aus einer einzelnen Steckdose. Sie ist mit dem Hauptschalter verbunden, von dem aus Verlängerungskabel und Verteilerdosen in alle Ecken der Bahn führen. Am Hauptschalter können die Funktionsgruppen einzeln aktiviert werden, dafür gibt es sechs beleuchtete Kippschalter, die wie folgt belegt sind.

- Main: Der Hauptschalter kontrolliert den Strom zu allen anderen Schaltern
- Switches: Aktiviert die beiden Ethernet-Switches (Voraussetzung für die PC104-Rechner)
- 12 Volt: Schaltet die Stromversorgung der Leistungselektroniken und der Bahn
- PC104 0-11, 12-23: Hiermit können die Rechner in zwei Gruppen aktiviert werden
- TTP: Dieser Schalter kontrolliert die Stromversorgung der acht TTP-Knoten

Die Anlage benötigt einen erheblichen Einschaltstrom, der vor allem von den Schaltnetzteilen herrührt. Daher dürfen die Gruppen nur nacheinander und nie gleichzeitig eingeschaltet werden, anderenfalls würde die Sicherung des Raumes sofort herausspringen.

4.6.5 Stromverbrauch

Auch wenn die vielen Netzteile und Platinen etwas anderes vermuten lassen, ist der Verbrauch der gesamten Anlage relativ gering. Im Ruhezustand benötigen die PC104-Rechner zusammen 170 Watt, die Powernodes 60 Watt und der Rest mit 12 Volt-Netzteilen, Leistungselektroniken und Switches kommt auf 30 Watt. Bei vollem Betrieb mit acht fahrenden Zügen, die über UDP gesteuert werden, erreicht der Gesamtverbrauch gerade einmal 300 Watt.

Teil II

Ansteuerung

Kapitel 5

API der Leistungselektronik

Steuerrechner und Leistungselektroniken kommunizieren über ihre seriellen Schnittstellen mit Hilfe eines einfachen Protokolls. Der Rechner kann Befehle senden, die der Mikrocontroller auf der Steuerplatine ausführt und mit einem Antwortpaket quittiert. Die verfügbaren Funktionen umfassen das Herstellen und Trennen der Verbindung, die Diagnose und die Ansteuerung der Peripherie. Der Benutzer kann wählen, ob er die Peripherieteile getrennt nach Funktion oder alle auf einmal steuern will. Die erste Option bietet mehr Flexibilität und eine detailliertere Steuerung, während die zweite schnellere Zyklen ermöglicht.

5.1 Allgemeines zum Protokoll

Die Kommunikation läuft bei einer Geschwindigkeit von 19200 Baud mit acht Datenbits, einem Stopbit und ohne Paritätsprüfung. Alle in beide Richtungen ausgetauschten Pakete sind nach einem einheitlichen Schema aufgebaut, das in Tabelle 5.1 beschrieben ist.

Byte	Allgemeines Paketformat	
[0]	**Header**	
	Bit 7-0:	F0h (Magic)
[1]	**Opcode**	
	Bit 7:	0 = Befehl
		1 = Antwort
	Bit 6-4:	Sequenznummer, 0...7
	Bit 3-0:	Nummer der Funktion, 0...10
[2]	**(Eventuelle Parameter)**	
[N]	**Checksumme**	

Tabelle 5.1: Grundlegender Aufbau der Pakete

Jedes Paket beginnt mit dem Wert F0h, gefolgt von dem Opcode-Byte. Dessen drei Bitfelder geben an, ob es sich um einen Befehl oder eine Antwort handelt, wie die Sequenznummer des Paketes lautet und zu welcher Funktion das Paket gehört. Die Sequenznummer wird von dem sendenden Rechner mit jedem Befehl inkrementiert, die Leistungselektronik kopiert die Nummer unverändert in das Antwortpaket. Der Rechner kann so feststellen, wenn Befehl und Antwort nicht zusammenpassen. Dies ist vor allem bei der Steuerung über ein Netzwerk wichtig, wenn

Pakete später als erwartet oder in der falschen Reihenfolge eintreffen. Das Paket endet nach den funktionsabhängigen Parametern mit einer Checksumme. Sie wird immer so gesetzt, daß die Summe aller Bytes in dem Paket durch 256 teilbar ist. Anzahl und Art der Parameter hängen von der Funktion ab und sind über den Opcode eindeutig festgelegt.

5.2 Befehlsübersicht

Das Protokoll definiert elf fortlaufend numerierte Funktionen, die in den folgenden Abschnitten dokumentiert werden. Weitere Details zur Implementierung stehen im Quellcode der Firmware.

	Name	Funktion	Befehl	Antwort
0	CMDCONNECT	Verbindung herstellen	4 Bytes	4 Bytes
1	CMDDISCONNECT	Verbindung trennen	4 Bytes	3 Bytes
2	CMDKEEPALIVE	Verbindung erhalten	3 Bytes	3 Bytes
3	CMDSIGNALS	Signale verändern und auslesen	7 Bytes	5 Bytes
4	CMDPOINTS	Weichen stellen und auslesen	4 Bytes	4 Bytes
5	CMDCONTACTS	Kontaktstatus abfragen	3 Bytes	6 Bytes
6	CMDTRACKS	Gleistreiber setzen und auslesen	6 Bytes	6 Bytes
7	CMDSENSORS	Sensoren der Gleistreiber auslesen	3 Bytes	6 Bytes
8	CMDGLOBAL	Globalen Status austauschen	8 Bytes	8 Bytes
9	CMDRESET	Reset der Peripherie	3 Bytes	3 Bytes
10	CMDEEPROM	Fehlerzähler lesen und zurücksetzen	5 Bytes	5 Bytes

Tabelle 5.2: Liste der vorhandenen Befehle mit den jeweiligen Paketgrößen

5.2.1 Verbindung herstellen

Bevor ein Steuerrechner mit der Kommunikation beginnt, sollte er die Verbindung mit dem Befehl CMDCONNECT herstellen. Er kann so entscheiden, ob die Peripherie initialisiert oder unverändert von dem Vorgänger übernommen werden soll, zwingend nötig ist das aber nicht.

Byte	Befehl
[0]	**Header**
	Bit 7-0: F0h (Magic)
[1]	**Request-Opcode**
	Bit 7: 0 (Befehl)
	Bit 6-4: Sequenznummer, 0...7
	Bit 3-0: 0 (CMDCONNECT)
[2]	**Parameter für die Initialisierung**
	Bit 7-1: 0 (Reserviert)
	Bit 0: 0 = Peripherie unverändert übernehmen
	1 = Peripherie zurücksetzen
[3]	**Checksumme**

Tabelle 5.3: Paketformat für CMDCONNECT-Befehl

Das Antwortpaket ist die Bestätigung, daß die Verbindung erfolgreich aufgebaut werden konnte. Es enthält die Nummer des seriellen Eingangs, auf dem der Befehl empfangen wurde.

Byte	Antwort
[0]	**Header**
	Bit 7-0: F0h (Magic)
[1]	**Reply-Opcode**
	Bit 7: 1 (Antwort)
	Bit 6-4: Sequenznummer, 0...7
	Bit 3-0: 0 (CMDCONNECT)
[2]	**Verbindungsdaten**
	Bit 7-2: 0 (Reserviert)
	Bit 1-0: Nummer des Kanals, 0...3
[3]	**Checksumme**

Tabelle 5.4: Paketformat für CMDCONNECT-Antwort

Ab diesem Moment zeigt die Leistungselektronik die aktive Kanalnummer in ihrem Display an und ist für die anderen Kanäle gesperrt. Die Peripherie kann normal gesteuert werden.

5.2.2 Verbindung trennen

Braucht der Rechner die Verbindung nicht mehr, kann er sie mit CMDDISCONNECT trennen.

Byte	Befehl
[0]	**Header**
	Bit 7-0: F0h (Magic)
[1]	**Request-Opcode**
	Bit 7: 0 (Befehl)
	Bit 6-4: Sequenznummer, 0...7
	Bit 3-0: 1 (CMDDISCONNECT)
[2]	**Parameter für die Initialisierung**
	Bit 7-1: 0 (Reserviert)
	Bit 0: 0 = Peripherie unverändert lassen
	1 = Peripherie zurücksetzen
[3]	**Checksumme**

Tabelle 5.5: Paketformat für CMDDISCONNECT-Befehl

Die Antwort wird noch komplett gesendet, weitere Befehle aber nicht mehr angenommen.

Byte	Antwort
[0]	**Header**
	Bit 7-0: F0h (Magic)
[1]	**Reply-Opcode**
	Bit 7: 1 (Antwort)
	Bit 6-4: Sequenznummer, 0...7
	Bit 3-0: 1 (CMDDISCONNECT)
[2]	**Checksumme**

Tabelle 5.6: Paketformat für CMDDISCONNECT-Antwort

Nach dem Senden der Antwort wechselt die Leistungselektronik wieder in den Suchmodus und ist offen für den nächsten Versuch, eine Verbindung aufzubauen.

5.2.3 Verbindung erhalten

Die Firmware trennt die Verbindung nach einer gewissen Zeit der Inaktivität, wenn keine Befehle empfangen wurden. Damit dies nicht eintritt, kann der Rechner den Befehl CMDKEEPALIVE senden. Er hat keine Wirkung auf die Peripherie, hält die Verbindung aber offen.

Byte	Befehl
[0]	**Header**
	Bit 7-0: F0h (Magic)
[1]	**Request-Opcode**
	Bit 7: 0 (Befehl)
	Bit 6-4: Sequenznummer, 0...7
	Bit 3-0: 2 (CMDKEEPALIVE)
[2]	**Checksumme**

Tabelle 5.7: Paketformat für CMDKEEPALIVE-Befehl

Weder Befehl noch Antwort enthalten irgendwelche Parameter, da sie nicht auf die Peripherie zugreifen und auch sonst keine Veränderungen auslösen.

Byte	Antwort
[0]	**Header**
	Bit 7-0: F0h (Magic)
[1]	**Reply-Opcode**
	Bit 7: 1 (Antwort)
	Bit 6-4: Sequenznummer, 0...7
	Bit 3-0: 2 (CMDKEEPALIVE)
[2]	**Checksumme**

Tabelle 5.8: Paketformat für CMDKEEPALIVE-Antwort

5.2.4 Signale verändern und auslesen

Der Zustand eines Signals wird von der Firmware in einem vier Bit breiten Wert codiert, die Bits (3, 2, 1, 0) stehen für (reserviert, grün, gelb, rot), wobei ein gesetztes Bit die betreffende Leuchtdiode einschaltet. Das Protokoll stellt eine einzige Funktion zur Steuerung aller Signale zur Verfügung, mit der beliebige Manipulationen aller Signale auf einmal möglich sind. Dazu wird der codierte Status jedes Signals zuerst mit einer Bitmaske AND-verknüpft und dann mit einer weiteren XOR-verknüpft. Das Ergebnis bestimmt den neuen Zustand des Signals und wird von der Funktion auch als Ergebnis zurückgegeben. Auf diese Weise können alle LEDs unabhängig voneinander eingeschaltet, ausgeschaltet, negiert oder einfach nur abgefragt werden. Die Bitmasken müssen nur passend zur gewünschten Aktion gesetzt sein.

Mit den Parametern ist es zwar möglich, den Status einer LED zu negieren, dies ist aber unsicher. Empfängt der Steuerrechner keine Antwort auf so einen Befehl, weiß er nicht, ob dieser doch

ausgeführt wurde und kennt somit auch den aktuellen Zustand nicht mehr. Es ist daher besser, den Status immer nur absolut zu verändern.

Byte	Befehl
[0]	**Header**
	Bit 7-0: F0h (Magic)
[1]	**Request-Opcode**
	Bit 7: 0 (Befehl)
	Bit 6-4: Sequenznummer, 0...7
	Bit 3-0: 3 (CMDSIGNALS)
[2]	**Bitmaske zum Verändern des Signalstatus**
	Bit 7-4: AND-Maske für Signal 3
	Bit 3-0: AND-Maske für Signal 2
[3]	**Bitmaske zum Verändern des Signalstatus**
	Bit 7-4: AND-Maske für Signal 1
	Bit 3-0: AND-Maske für Signal 0
[4]	**Bitmaske zum Verändern des Signalstatus**
	Bit 7-4: XOR-Maske für Signal 3
	Bit 3-0: XOR-Maske für Signal 2
[5]	**Bitmaske zum Verändern des Signalstatus**
	Bit 7-4: XOR-Maske für Signal 1
	Bit 3-0: XOR-Maske für Signal 0
[6]	**Checksumme**

Tabelle 5.9: Paketformat für CMDSIGNALS-Befehl

Byte	Antwort
[0]	**Header**
	Bit 7-0: F0h (Magic)
[1]	**Reply-Opcode**
	Bit 7: 1 (Antwort)
	Bit 6-4: Sequenznummer, 0...7
	Bit 3-0: 3 (CMDSIGNALS)
[2]	**Neues Signalmuster**
	Bit 7-4: Status von Signal 3
	Bit 3-0: Status von Signal 2
[3]	**Neues Signalmuster**
	Bit 7-4: Status von Signal 1
	Bit 3-0: Status von Signal 0
[4]	**Checksumme**

Tabelle 5.10: Paketformat für CMDSIGNALS-Antwort

5.2.5 Weichen stellen und auslesen

Nach dem selben Prinzip wie bei den Signalen werden auch die Weichen und ähnliche Peripherie wie Lampen angesteuert. Der Status der vier Ausgänge ist in einer Bitmaske zusammengefaßt, ein gesetztes Bit an Position i schaltet die Weiche am Ausgang PNTi auf das Nebengleis um

oder aktiviert einen dort angeschlossenen Verbraucher. Dieser Status wird mit zwei Bitmasken nacheinander AND-verknpüft und XOR-verknüpft. Das Ergebnis steuert den Weichentreiber und wird mit dem Antwortpaket an den Steuerrechner zurückgeliefert.

Byte	Befehl
[0]	**Header**
	Bit 7-0: F0h (Magic)
[1]	**Request-Opcode**
	Bit 7: 0 (Befehl)
	Bit 6-4: Sequenznummer, 0...7
	Bit 3-0: 4 (CMDPOINTS)
[2]	**Bitmaske zum Verändern des Weichenstatus**
	Bit 7-4: AND-Maske
	Bit 3-0: XOR-Maske
[3]	**Checksumme**

Tabelle 5.11: Paketformat für CMDPOINTS-Befehl

Das Antwortpaket enthält zusätzlich den aktuellen Zustand der Weichentreiber. Dies kann für eine Anwendung von Interesse sein, weil die Schaltbefehle von der Leistungselektronik unter Umständen mit einer geringen Verzögerung umgesetzt werden (siehe Kapitel 3).

Byte	Antwort
[0]	**Header**
	Bit 7-0: F0h (Magic)
[1]	**Reply-Opcode**
	Bit 7: 1 (Antwort)
	Bit 6-4: Sequenznummer, 0...7
	Bit 3-0: 4 (CMDPOINTS)
[2]	**Neuer Weichenstatus**
	Bit 7-4: Aktueller Weichenstatus
	Bit 3-0: Neuer Weichenstatus (noch zu erreichender Zielwert)
[3]	**Checksumme**

Tabelle 5.12: Paketformat für CMDPOINTS-Antwort

Mit den Parametern ist es möglich, den Status einer Weiche zu negieren, dies ist aber aus den selben Gründen wie bei der Signalsteuerung nicht zu empfehlen.

5.2.6 Kontaktstatus abfragen

Mit der Funktion CMDCONTACTS kann der Zustand der Kontakte ausgelesen werden. Sie liefert neben dem momentanen Status der einzelnen Reedkontakte auch Informationen über registrierte Ereignisse. Dabei wird von jedem Kontakt die Richtung der letzten Auslösung und ein Sequence Counter protokolliert. Die Counter werden bei jedem Ereignis inkrementiert und geben auf diese Weise an, ob und wie viele Ereignisse seit der letzten Abfrage stattgefunden haben. Bei diesem Protokoll gehen auch bei einer schlechten Verbindung keine Ereignisse verloren.

Byte	Befehl
[0]	**Header**
	Bit 7-0: F0h (Magic)
[1]	**Request-Opcode**
	Bit 7: 0 (Befehl)
	Bit 6-4: Sequenznummer, 0...7
	Bit 3-0: 5 (CMDCONTACTS)
[3]	**Checksumme**

Tabelle 5.13: Paketformat für CMDCONTACTS-Befehl

Der Aufrufer muß die Daten auswerten und daraus die für ihn interessanten Ereignisse ableiten.

Byte	Antwort
[0]	**Header**
	Bit 7-0: F0h (Magic)
[1]	**Reply-Opcode**
	Bit 7: 1 (Antwort)
	Bit 6-4: Sequenznummer, 0...7
	Bit 3-0: 5 (CMDCONTACTS)
[2]	**Bitmaske mit dem aktuellen Zustand der Einzelkontakte**
	Bit 7: 1 = Kontakt CON3/1 ist momentan geschlossen
	Bit 6: 1 = Kontakt CON3/0 ist momentan geschlossen
	Bit 5: 1 = Kontakt CON2/1 ist momentan geschlossen
	Bit 4: 1 = Kontakt CON2/0 ist momentan geschlossen
	Bit 3: 1 = Kontakt CON1/1 ist momentan geschlossen
	Bit 2: 1 = Kontakt CON1/0 ist momentan geschlossen
	Bit 1: 1 = Kontakt CON0/1 ist momentan geschlossen
	Bit 0: 1 = Kontakt CON0/0 ist momentan geschlossen
[3]	**Ergebnisse der letzten Auslösungen der Kontaktpaare**
	Bit 7-6: Letztes Ereignis von Kontakt 3
	Bit 5-4: Letztes Ereignis von Kontakt 2
	Bit 3-2: Letztes Ereignis von Kontakt 1
	Bit 1-0: Letztes Ereignis von Kontakt 0
	Bedeutung der Ereignisse
	0 = Nicht ausgelöst
	1 = Vorwärts
	2 = Rückwärts
	3 = Keine Richtung feststellbar
[4]	**Sequence Counter**
	Bit 7-6: Anzahl Ereignisse für Kontakt 3 (modulo 4)
	Bit 5-4: Anzahl Ereignisse für Kontakt 2 (modulo 4)
	Bit 3-2: Anzahl Ereignisse für Kontakt 1 (modulo 4)
	Bit 1-0: Anzahl Ereignisse für Kontakt 0 (modulo 4)
[5]	**Checksumme**

Tabelle 5.14: Paketformat für CMDCONTACTS-Antwort

5.2.7 Gleistreiber setzen und auslesen

Über den Befehl CMDTRACKS können die Einstellungen der beiden Gleistreiber gesetzt oder gelesen werden. Dazu gehört neben dem Modus (hochohmig, vorwärts, rückwärts oder bremsen) auch eine Einstellung der Geschwindigkeit. Der Aufrufer kann je nach Bedarf eine konstante PWM vorgeben oder einen Regelalgorithmus aktivieren, der eine bestimmte Zielgeschwindigkeit zu erreichen versucht und dazu die PWM eigenständig anpaßt. Zwei Bits im Befehlspaket geben an, ob die Einstellungen des jeweiligen Treibers geändert oder nur gelesen werden sollen. In jedem Fall werden die (resultierenden) Parameter als Antwort zurückgegeben.

Byte	Befehl
[0]	**Header**
	Bit 7-0: F0h (Magic)
[1]	**Request-Opcode**
	Bit 7: 0 (Befehl)
	Bit 6-4: Sequenznummer, 0...7
	Bit 3-0: 6 (CMDTRACKS)
[2]	**Bitmaske mit Steuerflags und Motormodi**
	Bit 7-6: 0 (Reserviert)
	Bit 5: 0 = Gleistreiber 1 nur abfragen
	1 = Einstellungen von Gleistreiber 1 ändern
	Bit 4: 0 = Gleistreiber 0 nur abfragen
	1 = Einstellungen von Gleistreiber 0 ändern
	Bit 3-2: **Modus für Gleistreiber 1, wenn Bit 5 gesetzt ist**
	0 = Aus
	1 = Vorwärts
	2 = Rückwärts
	3 = Bremsen
	Bit 1-0: **Modus für Gleistreiber 0, wenn Bit 4 gesetzt ist**
	0 = Aus
	1 = Vorwärts
	2 = Rückwärts
	3 = Bremsen
[3]	**Einstellungen für Gleis 1, wenn Bit 5 in [2] gesetzt ist**
	Bit 7: 0 = Steuerung über eine feste PWM
	1 = Geschwindigkeitsregler aktivieren
	Bit 6-0: PWM-Tastverhältnis, 0...127 (wenn Bit 7 = 0)
	Geschwindigkeit, 0...127 (wenn Bit 7 = 1)
[4]	**Einstellungen für Gleis 0, wenn Bit 4 in [2] gesetzt ist**
	Bit 7: 0 = Steuerung über eine feste PWM
	1 = Geschwindigkeitsregler aktivieren
	Bit 6-0: PWM-Tastverhältnis, 0...127 (wenn Bit 7 = 0)
	Geschwindigkeit, 0...127 (wenn Bit 7 = 1)
[5]	**Checksumme**

Tabelle 5.15: Paketformat für CMDTRACKS-Befehl

Beim Bremsen kann die Intensität über die PWM-Rate eingestellt werden. Je größer der Wert, desto schneller kommt der Zug zum stehen. Es macht hierbei keinen Sinn, den Regelalgorithmus

zu benutzen. Wird er trotzdem aktiviert, bremst die Firmware stattdessen mit voller Intensität.

Byte	Antwort
[0]	**Header**
	Bit 7-0: F0h (Magic)
[1]	**Reply-Opcode**
	Bit 7: 1 (Antwort)
	Bit 6-4: Sequenznummer, 0...7
	Bit 3-0: 6 (CMDTRACKS)
[2]	**Aktuelle Modi der Gleistreiber**
	Bit 7-4: 0 (Reserviert)
	Bit 3-2: **Modus von Gleistreiber 1**
	0 = Aus
	1 = Vorwärts
	2 = Rückwärts
	3 = Bremsen
	Bit 1-0: **Modus von Gleistreiber 0**
	0 = Aus
	1 = Vorwärts
	2 = Rückwärts
	3 = Bremsen
[3]	**Einstellungen für Gleis 1**
	Bit 7: 0 = Steuerung über eine feste PWM
	1 = Geschwindigkeitsregler aktivieren
	Bit 6-0: PWM-Tastverhältnis, 0...127 (wenn Bit 7 = 0)
	Geschwindigkeit, 0...127 (wenn Bit 7 = 1)
[4]	**Einstellungen für Gleis 0**
	Bit 7: 0 = Steuerung über eine feste PWM
	1 = Geschwindigkeitsregler aktivieren
	Bit 6-0: PWM-Tastverhältnis, 0...127 (wenn Bit 7 = 0)
	Geschwindigkeit, 0...127 (wenn Bit 7 = 1)
[5]	**Checksumme**

Tabelle 5.16: Paketformat für CMDTRACKS-Antwort

Der Zug fährt um so schneller, je höher Tastverhältnis oder Geschwindigkeit gewählt werden. Die höchste mögliche Geschwindigkeit hängt von der jeweiligen Lokomotive ab und kann Anhang B.1 entnommen werden, typische Werte liegen knapp unter 45.

5.2.8 Sensoren der Gleistreiber auslesen

Die Gleistreiber sind mit einer Reihe von Sensoren ausgestattet, die Kurzschlüsse erkennen, auf den Gleisen stehende Lokomotiven detektieren und ihre Geschwindigkeit messen können. Die beiden letzteren werden allerdings nur in Abständen von 0,2 Sekunden während der Fahrt oder 1,0 Sekunden im Stillstand aktualisiert. Daher gibt es für jeden Gleistreiber ein Bit, das beim Eintreffen neuer Daten negiert wird. Der Benutzer muß die Bits auf Veränderungen überwachen, wenn er an aktuellen Daten interessiert ist.

Byte	Befehl
[0]	**Header**
	Bit 7-0: F0h (Magic)
[1]	**Request-Opcode**
	Bit 7: 0 (Befehl)
	Bit 6-4: Sequenznummer, 0...7
	Bit 3-0: 7 (CMDSENSORS)
[2]	**Checksumme**

Tabelle 5.17: Paketformat für CMDSENSORS-Befehl

Die Kurzschlußerkennung überwacht beide Gleistreiber gemeinsam. Daher braucht sie unter Umständen einen kurzen Moment, bis sie die genaue Ursache erkannt und den dazugehörenden Treiber abgeschaltet hat. In dieser Zeit kann auch das unbeteiligte Gleis abgeschaltet werden, in der Regel dauert dies aber nur einige Sekundenbruchteile.

Byte	Antwort
[0]	**Header**
	Bit 7-0: F0h (Magic)
[1]	**Reply-Opcode**
	Bit 7: 1 (Antwort)
	Bit 6-4: Sequenznummer, 0...7
	Bit 3-0: 7 (CMDSENSORS)
[2]	**Statusbits der Gleistreiber**
	Bit 7-6: 0 (Reserviert)
	Bit 5: 0 = Gleis 1 arbeitet normal
	1 = Überlastsicherung hat Gleis 1 abgeschaltet
	Bit 4: 0 = Gleis 0 arbeitet normal
	1 = Überlastsicherung hat Gleis 0 abgeschaltet
	Bit 3: Jeder Wechsel dieses Bits zeigt neue Daten von Gleis 1 an
	Bit 2: Jeder Wechsel dieses Bits zeigt neue Daten von Gleis 0 an
	Bit 1: 0 = Kein Stromverbraucher auf Gleis 1 gefunden
	1 = Ein Treibewagen steht auf Gleis 1
	Bit 0: 0 = Kein Stromverbraucher auf Gleis 0 gefunden
	1 = Ein Treibewagen steht auf Gleis 0
[3]	**Aktuelle Geschwindigkeit des Zuges auf Gleis 1**
	Bit 7: 0 (Reserviert)
	Bit 6-0: Betrag der Geschwindigkeit
[4]	**Aktuelle Geschwindigkeit des Zuges auf Gleis 0**
	Bit 7: 0 (Reserviert)
	Bit 6-0: Betrag der Geschwindigkeit
[5]	**Checksumme**

Tabelle 5.18: Paketformat für CMDSENSORS-Antwort

Die Geschwindigkeiten werden unabhängig von der Richtung des Zuges immer positiv angegeben. Der Wert entspricht ungefähr Zentimetern pro Sekunde, typische Werte für die Lokomotiven der Anlage liegen bei maximal 45 (siehe Anhang B.1).

5.2.9 Globalen Status austauschen

Die bisher vorgestellten Funktionen zur Steuerung der Peripherie erlauben einen detaillierten Zugriff auf alle Parameter. Sie erweisen sich jedoch als unhandlich, wenn das Steuerprogramm mehrere Aktionen auf einmal durchführen will. Für deren Umsetzung muß eine ganze Reihe von Befehlen erzeugt werden, die Ausführung dauert entsprechend lang. Die hier beschriebene Funktion steuert die gesamte Peripherie auf einmal, ist dafür aber in ihren Möglichkeiten leicht eingeschränkt. Sie kann insbesondere Kontakte nur als Paar und nicht mehr einzeln auslesen.

Byte	Befehl
[0]	**Header**
	Bit 7-0: F0h (Magic)
[1]	**Request-Opcode**
	Bit 7: 0 (Befehl)
	Bit 6-4: Sequenznummer, 0...7
	Bit 3-0: 8 (CMDGLOBAL)
[2]	**Neues Signalmuster**
	Bit 7-4: Status von Signal 3
	Bit 3-0: Status von Signal 2
[3]	**Neues Signalmuster**
	Bit 7-4: Status von Signal 1
	Bit 3-0: Status von Signal 0
[4]	**Neuer Status der Motortreiber und Weichen**
	Bit 7-6: **Modus für Gleistreiber 1**
	0 = Aus
	1 = Vorwärts
	2 = Rückwärts
	3 = Bremsen
	Bit 5-4: **Modus für Gleistreiber 0**
	0 = Aus
	1 = Vorwärts
	2 = Rückwärts
	3 = Bremsen
	Bit 3-0: Neuer Zielzustand der Weichen
[5]	**Einstellungen für Gleis 1**
	Bit 7: 0 = Steuerung über eine feste PWM
	1 = Geschwindigkeitsregler aktivieren
	Bit 6-0: PWM-Tastverhältnis, 0...127 (wenn Bit 7 = 0)
	Geschwindigkeit, 0...127 (wenn Bit 7 = 1)
[6]	**Einstellungen für Gleis 0**
	Bit 7: 0 = Steuerung über eine feste PWM
	1 = Geschwindigkeitsregler aktivieren
	Bit 6-0: PWM-Tastverhältnis, 0...127 (wenn Bit 7 = 0)
	Geschwindigkeit, 0...127 (wenn Bit 7 = 1)
[7]	**Checksumme**

Tabelle 5.19: Paketformat für CMDGLOBAL-Befehl

Die genaue Bedeutungen der Parameter sind bei den jeweiligen Spezialfunktionen dokumentiert.

Byte	Antwort
[0]	**Header**
	Bit 7-0: F0h (Magic)
[1]	**Reply-Opcode**
	Bit 7: 1 (Antwort)
	Bit 6-4: Sequenznummer, 0...7
	Bit 3-0: 8 (CMDGLOBAL)
[2]	**Ergebnisse der letzten Auslösungen der Kontaktpaare**
	Bit 7-6: Letztes Ereignis von Kontakt 3
	Bit 5-4: Letztes Ereignis von Kontakt 2
	Bit 3-2: Letztes Ereignis von Kontakt 1
	Bit 1-0: Letztes Ereignis von Kontakt 0
	Bedeutung der Bitwerte
	0 = Nicht ausgelöst
	1 = Vorwärts
	2 = Rückwärts
	3 = Keine Richtung feststellbar
[3]	**Sequence Counter der Kontakte**
	Bit 7-6: Anzahl Ereignisse für Kontakt 3 (modulo 4)
	Bit 5-4: Anzahl Ereignisse für Kontakt 2 (modulo 4)
	Bit 3-2: Anzahl Ereignisse für Kontakt 1 (modulo 4)
	Bit 1-0: Anzahl Ereignisse für Kontakt 0 (modulo 4)
[4]	**Statusbits der Gleistreiber**
	Bit 7-6: 0 (Reserviert)
	Bit 5: 1 = Überlastsicherung hat Gleis 1 abgeschaltet
	Bit 4: 1 = Überlastsicherung hat Gleis 0 abgeschaltet
	Bit 3: Jeder Wechsel dieses Bits zeigt neue Daten von Gleis 1 an
	Bit 2: Jeder Wechsel dieses Bits zeigt neue Daten von Gleis 0 an
	Bit 1: 1 = Ein Treibewagen steht auf Gleis 1
	Bit 0: 1 = Ein Treibewagen steht auf Gleis 0
[5]	**Aktuelle Geschwindigkeit des Zuges auf Gleis 1**
	Bit 7-0: Betrag der Geschwindigkeit in Zentimetern pro Sekunde
[6]	**Aktuelle Geschwindigkeit des Zuges auf Gleis 0**
	Bit 7-0: Betrag der Geschwindigkeit in Zentimetern pro Sekunde
[7]	**Checksumme**

Tabelle 5.20: Paketformat für CMDGLOBAL-Antwort

Diese Funktion ist für alle Szenarien gut geeignet, in denen das Steuerprogramm den Zustand der Leistungselektronik in seinem Speicher festhält und in regelmäßigen Abständen mit der Hardware abgleichen will.

5.2.10 Reset der Peripherie

Auf Wunsch kann die Firmware den Zustand der Peripherie wieder in den Einschaltzustand bringen. Alle Signallampen werden abgeschaltet, die Weichen auf Geradeausfahrt umgestellt, Züge angehalten und der Kontaktzustand zurückgesetzt. Hinterher sind sowohl deren Sequence Counter als auch die zuletzt registrierten Ereignisse auf Null zurückgesetzt.

Byte	Befehl
[0]	**Header**
	Bit 7-0: F0h (Magic)
[1]	**Request-Opcode**
	Bit 7: 0 (Befehl)
	Bit 6-4: Sequenznummer, 0...7
	Bit 3-0: 9 (CMDRESET)
[2]	**Checksumme**

Tabelle 5.21: Paketformat für CMDRESET-Befehl

Byte	Antwort
[0]	**Header**
	Bit 7-0: F0h (Magic)
[1]	**Reply-Opcode**
	Bit 7: 1 (Antwort)
	Bit 6-4: Sequenznummer, 0...7
	Bit 3-0: 9 (CMDRESET)
[2]	**Checksumme**

Tabelle 5.22: Paketformat für CMDRESET-Antwort

5.2.11 Fehlerzähler lesen und zurücksetzen

Die Leistungselektronik verwaltet im EEPROM des Mikrocontrollers insgesamt 13 Fehlerzähler, die bei Problemen inkrementiert werden, bis sie bei 255 in die Sättigung gehen. Die Funktion CMDEEPROM kann alle Zähler einzeln auslesen und auf Null zurücksetzen.

Byte	Befehl
[0]	**Header**
	Bit 7-0: F0h (Magic)
[1]	**Request-Opcode**
	Bit 7: 0 (Befehl)
	Bit 6-4: Sequenznummer, 0...7
	Bit 3-0: 10 (CMDEEPROM)
[2]	**Zugriffsmodus**
	Bit 7-1: 0 (Reserviert)
	Bit 0: Art des Zugriffs
	0 = Lesen
	1 = Zurücksetzen
[3]	**Adresse des Wertes**
	Bit 7-0: Adresse, 0...12
[4]	**Checksumme**

Tabelle 5.23: Paketformat für CMDEEPROM-Befehl

Die Zähler bleiben über Neustarts hinweg erhalten und können zu einem beliebigen Zeitpunkt abgefragt und zurückgestellt werden. Sie sind ab Null numeriert.

Zähler	Quelle	Ursache
0	CON0	Versagen von Reedkontakt 0
1	CON0	Versagen von Reedkontakt 1
2	CON1	Versagen von Reedkontakt 0
3	CON1	Versagen von Reedkontakt 1
4	CON2	Versagen von Reedkontakt 0
5	CON2	Versagen von Reedkontakt 1
6	CON3	Versagen von Reedkontakt 0
7	CON3	Versagen von Reedkontakt 1
8	TRK0	Kurzschluß oder Überlast
9	TRK1	Kurzschluß oder Überlast
10	PIC	Durch MCLR ausgelöster Reset
11	PIC	Vom Watchdog ausgelöster Reset
12	PIC	Reset durch instabile Spannung

Tabelle 5.24: Bedeutung der Fehlerzähler

Die Löschung eines Zählers wird von der Firmware nur vorgemerkt und zum nächstmöglichen Zeitpunkt ausgeführt, was einige Millisekunden dauern kann. Zugriffe auf das EEPROM sind nur möglich, wenn der Speicher gerade nicht anderweitig gebraucht wird. Entsprechend kann die Firmware diesen Befehl mit einem Fehler zurückweisen, das Steuerprogramm muß ihn dann zu einem späteren Zeitpunkt wiederholen.

Byte	Antwort
[0]	**Header**
	Bit 7-0: F0h (Magic)
[1]	**Reply-Opcode**
	Bit 7: 1 (Antwort)
	Bit 6-4: Sequenznummer, 0...7
	Bit 3-0: 10 (CMDEEPROM)
[2]	**Ergebnis des Zugriffs**
	Bit 7-1: 0 (Reserviert)
	Bit 0: Ergebnis des Zugriffs
	0 = Erfolgreich
	1 = Momentan kein Zugriff möglich
[3]	**Wert des Zählers nach erfolgreichem Lesebefehl**
	Bit 7-0: Wert, 0...255
[4]	**Checksumme**

Tabelle 5.25: Paketformat für CMDEEPROM-Antwort

5.3 Ablauf der Kommunikation

Das Protokoll ist so einfach aufgebaut, daß es mit wenig Aufwand auch auf Systemen mit stark beschränkten Ressourcen implementiert werden kann. Befehle werden als Block gesendet, was etwa 0,5 Millisekunden pro Byte dauert. Dabei ist nur darauf zu achten, daß zwischen zwei

Bytes keine Pausen liegen, bei einer Unterbrechung von mehr als 260 Mikrosekunden geht die Firmware von einem Problem aus und verwirft das Paket.

Die Verarbeitung des Befehls dauert nie länger als die serielle Übertragung eines Bytes. Danach beginnt die Leistungselektronik unmittelbar mit dem Senden der Antwort. Das Programm des Steuerrechners sollte in dem empfangenen Datenstrom nach dem Antwortpaket suchen und sich dabei fehlertolerant verhalten sowie Timeouts berücksichtigen. Diese Aufgabe übernimmt ein einfacher Algorithmus, der auch in der Firmware der Leistungselektronik zum Einsatz kommt.

Voraussetzungen

- Der Code verwaltet die Daten einen Ringpuffer, der größer als alle möglichen Pakete ist. Dabei sind 16 Bytes bereits vollkommen ausreichend.

Gültigkeitsprüfung

- Eine Funktion kann prüfen, ob am Anfang des Puffers ein gültiges Paket enthalten ist, und liefert eine der Antworten „Ja“, „Nein“ und „Vielleicht“. Dabei können alle beschriebenen Eigenschaften des Paketes ausgenutzt werden.
  - Stimmt das Magic-Byte?
  - Definiert der Opcode eine gültige Antwort mit der richtigen Sequenznummer?
  - Wurden bereits alle Daten nötigen empfangen?
  - Ist die Checksumme des Paketes korrekt?
  - Sind weitere Daten zu erwarten oder ist seit dem letzten Byte viel Zeit vergangen?
- Basierend auf diesen Prüfungen muß die Funktion ihre Entscheidung treffen.

Empfang von Daten

- Jedes korrekt empfangene Byte wird zunächst in dem Ringpuffer abgelegt.
- Anschließend findet die oben beschriebene Suche nach gültigen Paketen statt.
- Bei der Antwort „Ja“ liegt eine gültige Antwort vor, die aus dem Puffer entfernt und dem Aufrufer übergeben wird.
- Die Antwort „Vielleicht“ bedeutet, daß nicht ausreichend Daten für eine Entscheidung im Puffer enthalten ist. In diesem Fall muß das Programm auf weitere Bytes warten.
- Bei der Antwort „Nein“ wird das erste Byte des Puffers gelöscht und weiter gewartet.
- Sollte der Transceiver einen Übertragungsfehler (Overrun, Framing Error) melden, wird der gesamte Inhalt des Puffers gelöscht.
- Das Programm wartet nur eine begrenzte Zeit auf eine Antwort.

Prinzipiell kann die Dauer eines Befehlszyklus und damit die Länge der Timeouts relativ genau berechnet werden. Bei den meisten seriellen Schnittstellen scheitert dies jedoch an den FIFOs des Treiberchips. Sie sammeln empfangene Bytes und geben sie erst an das Betriebssystem oder an Applikationen weiter, wenn sie einen gewissen Füllstand erreicht haben oder eine bestimmte Zeit lang keine neuen Bytes empfangen wurden. Der eigentliche Befehlszyklus dauert maximal neun Millisekunden, abhängig von der Schnittstelle und dem Betriebssystem müssen die Timeouts unter Umständen wesentlich höher liegen.

Kapitel 6

Programmierschnittstelle

Auf dem in Kapitel 5 beschriebenen Protokoll setzt die Schnittstelle für Benutzerprogramme auf. Sie besteht aus einer in C geschriebenen Library, die alle logischen Ebenen abdeckt. Dies beginnt mit der Abwicklung des seriellen Paketaustausches und der Kommunikation über die Bussysteme. Am Ende steht eine Abstraktionsschicht, in der die Anlage als Einheit präsentiert wird und die Peripherieteile über die Bezeichnungen aus dem Gleisplan angesprochen werden. Dem Programmierer steht es dabei frei, auf welcher Ebene er die Hardware ansteuern will. Auf diese Weise lassen sich unterschiedlichste Anwendungen mit der selben Codebasis realisieren.

Abbildung 6.1: Startbereite Züge im Inner Circle Bahnhof

Das Gegenstück zur Library bildet ein auf den PC104-Rechnern laufender Daemon. Dieser bietet die lokal vorhandenen seriellen Schnittstellen auf den aktiven Bussystemen an und routet die Datenpakete von und zu den Leistungselektroniken. Der gesamte Code basiert auf C, weil diese Sprache den kleinsten gemeinsamen Nenner aller in Frage kommenden Werkzeuge darstellt. Nur so kann die Library problemlos von dem synthetisierten Code der gängigen Modellierungstools aus benutzt werden. Dieses Kapitel bietet einen Überblick über die Funktionalität der Module, aus denen die Library besteht. Weitere Informationen sowie eine vollständige Beschreibung der Schnittstellen finden sich in den Kommentaren des Programmcodes.

6.1 Grundlegende Konzepte

Die Grundlage fast aller Funktionen bilden sogenannte *Nodelinks*. Dabei handelt es sich um Kommunikationskanäle, die einen steuernden Prozeß mit einer Leistungselektronik verbinden. Ein Nodelink kann ein serielles Kabel am jeweiligen Rechner repräsentieren, genauso kann sich dahinter ein Tunnel durch ein Feldbussystem verbergen. Alle Varianten verfügen über das selbe Interface und können von den höheren Ebenen des Interfaces uniform benutzt werden. Deswegen spielt es für Anwendungsprogramme keine Rolle, über welches Bussystem sie mit der Anlage kommunizieren. Die Nodelinks müssen nur eingangs individuell erzeugt werden.

Hierbei handelt es sich um objektorientiertes Design. Ein Nodelink entspricht einer abstrakten Klasse, von der die jeweiligen Implementierungen durch Vererbung abgeleitet werden. Jede von ihnen verfügt über einen individuellen Konstruktor, der den Nodelink herstellt, ab dann können die übrigen virtuellen Methoden nach den Regeln der Polymorphie benutzt werden. Die Sprache C unterstützt solche Konstrukte nicht, daher werden sie manuell nachgebildet. Alle Daten eines Nodelinks werden in einer einheitlichen Struktur gespeichert, dazu gehören auch Zeiger auf die jeweiligen Funktionen. Über diese Tabelle virtueller Methoden kann der Code feststellen, welche Funktionen zu einem konkreten Nodelink gehören.

6.2 Benutzung der Library

Die Library setzt sich aus mehreren Modulen zusammen, die jeweils einer C-Datei entsprechen. Im Allgemeinen sind die Module relativ abgeschlossene Einheiten, die dem Programmierer den Zugriff auf einer bestimmten Ebene oder auf eine eng umrissene Funktionalität erlauben. Sie werden in den folgenden Abschnitten einzeln kurz skizziert.

Viele Eigenschaften sind bei allen Funktionen gleich. Zeiten werden immer in Mikrosekunden angebeben, die meisten Referenzparameter können auf `NULL` gesetzt werden, wenn das Ergebnis nicht von Bedeutung ist. Fast alle Funktionen liefern 0 beziehungsweise einen gültigen Zeiger zurück, wenn sie erfolgreich ausgeführt werden. Fehlerursachen werden dem Aufrufer über `errno` signalisiert, wie es auch bei der `libc` üblich ist.

6.2.1 Modul firmware

Auf unterster Ebene enthält das Modul `firmware` Konstanten und Funktionen, die bei der Kommunikation mit dem Mikrocontroller auf einer Leistungselektronik hilfreich sind.

- Einige Konstanten decken die Parameter der seriellen Kommunikation (Baudrate, Magic, Befehlscodes, Größen der Pakete) ab, andere beschreiben die Adressen der Fehlerzähler im EEPROM oder wichtige Eckdaten im Timing der Firmware.
- Eine Gruppe von Makros und Funktionen dient dazu, die Pakete des seriellen Protokolls auszuwerten. Opcodes können erstellt, zerlegt und umgeformt werden. Andere Funktionen berechnen und prüfen die Checksummen des Paketes oder dessen Inhalt auf Plausibilität.
- Der Rest des Codes hilft bei der zeitlichen Ablaufsteuerung. Zwei Routinen warten für eine definierte Zeit (mit Suspendierung des Prozesses oder per Busy Waiting), ein Makro berechnet, wieviel Zeit zwischen zwei Aufrufen von `gettimeofday` vergangen ist.

Der Benutzer kommt mit dieser Funktionalität kaum direkt in Kontakt, sind für ihn vor allem die Konstanten für das Timing oder die Adressen der Fehlerzähler relevant.

6.2.2 Modul nodelink

Die gemeinsame Schnittstelle aller Nodelinks wird in dem gleichnamigen Modul definiert. Eine Struktur dient als Handle und enthält die nötigen Daten für alle Typen von Nodelinks. Das Modul definiert selber keine Routine zum Öffnen der Verbindung und Initialisieren der Struktur, dafür sind die spezialisierten Varianten in anderen Modulen zuständig. Der einmal erstellte Nodelink kann von den übrigen Funktionen dieses Moduls benutzt werden, um unabhängig von der Art des Links zu kommunizieren und die Verbindung wieder schließen zu können.

```
int nodelink_close(struct nodelink *link);
int nodelink_send(struct nodelink *link, unsigned char *request,
                  unsigned size);
int nodelink_receive(struct nodelink *link, unsigned char *reply,
                     unsigned *size, unsigned opcode, unsigned timeout);
int nodelink_command(struct nodelink *link,
                     unsigned char *request, unsigned requestsize,
                     unsigned char *reply, unsigned *replysize);
```

Die Funktion `nodelink_send` überträgt ein vorbereitetes Paket an die Leistungselektronik. Ihr Gegenstück `nodelink_receive` empfängt die Antwort mit einstellbarem Timeout. Das Paket muß den vorgegebenen Opcode enthalten, anderenfalls wird es verworfen. Dieser Mechanismus dient dazu, nur Antworten mit der korrekten Sequenznummer zu akzeptieren. Der Aufrufer kann auch `OPCODE_ANY` übergeben, in dem Fall wird jedes Paket akzeptiert. Timeouts und andere Fehler werden über den Rückgabewert angezeigt.

Häufig wird die Applikation nacheinander senden und empfangen wollen. Dafür existiert als Abkürzung die Funktion `nodelink_command`, die als Vereinfachung außerdem weniger Parameter benötigt. Opcodes und Paketgrößen werden aus dem zu sendenden Paket ermittelt, der Timeout ergibt sich aus dem Standardwert für die Verbindung, wie er im Nodelink gespeichert ist.

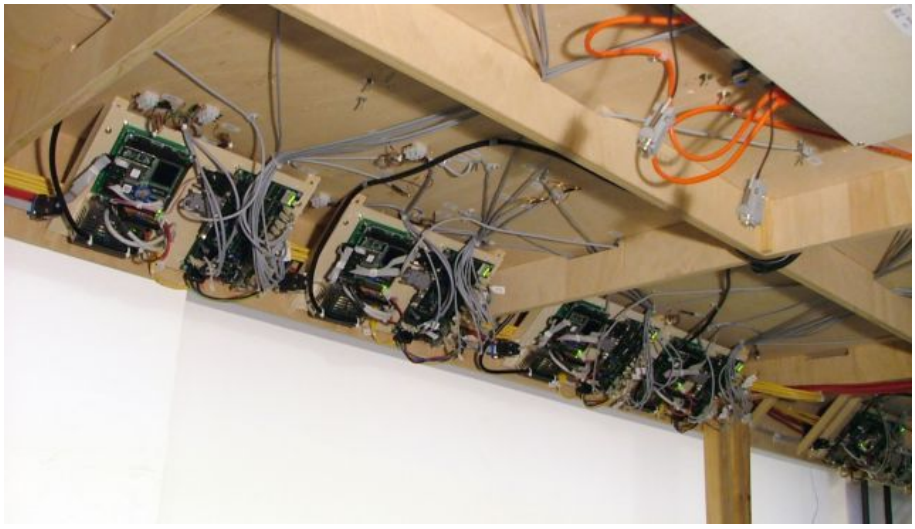


Abbildung 6.2: Verbundene Steuerknoten unter der Anlage

Auf einigen Netzwerken wie dem CAN-Bus sind nur sehr kleine Pakete erlaubt. Wenn eine serielle Schnittstelle auf der anderen Seite geöffnet werden soll, ist ihr vollständiger Name meistens

zu lang, um in ein Paket zu passen. Daher wird statt des Namens ein entsprechender Code übermittelt. Zwei Funktionen bilden Code und Name aufeinander ab, die Liste der definierten Namen ist statisch im Modul gespeichert.

```
int nl_devicecode(char *device, unsigned char *code);
char *nl_devicename(unsigned char devicecode);
```

Es werden mit `/dev/ttyS0` bis `/dev/ttyS3` alle Schnittstellen der PC104-Rechner unterstützt.

6.2.3 Module nltty, nludp und nlcan

Die Kommunikationsroutinen der unterschiedlichen Nodelinks sind jeweils in eigenen Modulen implementiert. Diese enthalten neben Funktionen zur Ansteuerung des Mediums jeweils die vier Basisroutinen des Nodelinks (open, close, send und receive). Jede Kommunikation beginnt mit dem Öffnen der Verbindung.

```
struct nodelink *nodelink_tty_open(char *device);
struct nodelink *nodelink_udp_open(char *hostname, char *remotedevice);
struct nodelink *nodelink_can_open(struct pcan_interface *iface,
                                   unsigned remotenode,
                                   char *remotedevice);
```

Die Parameter hängen von dem Medium ab. Bei lokalen seriellen Schnittstellen (tty) muß nur der Gerätename übergeben werden, Ethernet-Verbindungen (udp) erfordern zusätzlich den Namen des entfernten Rechners. Nodelinks über CAN können nur geöffnet werden, nachdem ein Handle für das lokale CAN-Interface zur Verfügung steht, zusätzlich werden die Nummer des entfernten Rechners und der Name der seriellen Schnittstelle auf der anderen Seite übergeben. Die so geöffneten Nodelinks können für die weitere Kommunikation benutzt werden.

```
int nodelink_*_close(struct nodelink *link);
int nodelink_*_send(struct nodelink *link, unsigned char *request,
                    unsigned size);
int nodelink_*_receive(struct nodelink *link, unsigned char *reply,
                       unsigned *size, unsigned opcode, unsigned timeout);
```

Es ist egal, ob die Funktionen zum Senden, Empfangen und Schließen des eigenen Moduls oder die Gegenstücke aus dem Modul `nodelink` benutzt werden. Der Vorteil der zweiten Methode liegt darin, daß es für sie keine Rolle spielt, wie der Nodelink ursprünglich erzeugt wurde.

Kommunikation auf den Netzwerken

Die Verwaltung der Verbindungen auf den Bussystemen Ethernet und CAN verläuft nach einem einfachen Schema. Zum Öffnen und Schließen einer seriellen Schnittstelle auf der anderen Seite wird ein Paket mit dem Code des Devices gesendet. Der betreffende Rechner antwortet mit einem Resultatcode, der den Erfolg oder eventuelle Fehler signalisiert.

Die eigentlichen Datenpakete werden unverändert und nicht eingekapselt in beide Richtungen übertragen. Tritt auf dem entfernten Rechner ein Fehler auf, sendet er ein Paket mit einem

entsprechenden Fehlercode. Das Protokoll unterscheidet Nachrichten anhand des ersten Bytes, Datenpakete beginnen mit dem Magic-Wert 0xF0, andere mit entsprechenden eigenen Nummern.

Die Kommunikation über Ethernet basiert auf UDP, der Server auf den PC104-Rechnern ist über Port 8250 erreichbar. CAN arbeitet bei 250 KBit/s, Verbindungen zu Rechner n werden durch Befehle an die CAN-ID n geöffnet und geschlossen. Der Server weist jeder Verbindung eine freie ID zu, unter welcher der Austausch der Datenpakete erfolgt.

6.2.4 Module nlsim und nllog

Wie anpassungsfähig das beschriebene Konzept ist, zeigen zwei spezielle Module. Die von `nlsim` erzeugten Nodelinks führen nicht zu echter Hardware sondern zu einem im Modul integrierten Simulator, der sich wie eine Leistungselektronik ohne Peripherie verhält. Programme können auf diese Weise die Kommunikation testen, ohne auf echte Hardware angewiesen zu sein.

```
struct nodelink *nodelink_sim_open(struct nodesim *sim, unsigned delay);
```

Die Ausführungszeit der Befehle wird bei der Erstellung des Nodelinks angegeben. Sie beträgt in der Regel 10000 Mikrosekunden plus die Laufzeit durch eventuelle nachzuahmende Netzwerke.

Ein ähnlich wichtiges Werkzeug bietet das Modul `nllog`. Es kapselt einen bestehenden Nodelink in einen Mechanismus ein, der transparent alle durch ihn laufenden Pakete protokolliert.

```
struct nodelink *nodelink_log_open(char *logfile, char *infotext,
                                   int mode, struct nodelink *tolog);
```

Die Ausgabe erfolgt in eine Textdatei, in der jedes Paket in einer Zeile dargestellt wird. Davor stehen ein Zeitstempel und ein frei wählbarer Infotext. Der Detailgrad kann auf mehreren Stufen bis hin zu textuell interpretierten Inhalten eingestellt werden.

6.2.5 Modul nodeapi

Das Erstellen von Befehlspaketen und das Interpretieren der Antworten sind lästige Aufgaben, die das Modul `nodeapi` dem Programmierer abnehmen kann. Seine Funktionen entsprechen genau dem API der Firmware. Dabei sind jedem Befehlscode drei Routinen zugeordnet, wie das folgende Beispiel für `CMDCONNECT` zeigt.

```
int node_connect_send(struct nodelink *link, int reset);
int node_connect_recv(struct nodelink *link, unsigned *channel,
                      unsigned timeout);
int node_connect(struct nodelink *link, int reset, unsigned *channel);
```

Die erste Funktion erzeugt ein Befehlspaket und sendet es über den Nodelink. Der Parameter `reset` entspricht dem Flag im Paket und gibt an, ob die Peripherie zurückgesetzt werden soll. Das Antwortpaket wird von der zweiten Funktion empfangen und ausgewertet, die Routine setzt `channel` auf die übermittelte Kanalnummer. Die dritte der genannten Funktionen führt beides nacheinander aus. Dabei wird die im Nodelink gespeicherte Zeitschranke als Timeout benutzt.

Diese Aufteilung erlaubt es Programmen, auf zwei unterschiedliche Arten zu kommunizieren. Zunächst können sie mit einem Aufruf der dritten Funktion einen kompletten Befehl ausführen

lassen. Dies ist vor allem dann interessant, wenn nur eine Leistungselektronik angesprochen werden soll. Bei vielen Nodelinks dauert es zu lange, den Vorgang mehrmals zu wiederholen. In solchen Fällen ist es sinnvoller, erst alle Befehle zu senden und dann die Antworten zu sammeln. Für diese Aufgabe sind die ersten beiden Funktionen konzipiert, die nebenläufige Ausführung mehrerer Befehle dauert kaum länger als ein einzelner Befehl.

Das Modul übernimmt intern auch die Verwaltung der Sequenznummern. Ein Eintrag in der Nodelink-Struktur ist ein Zähler, der mit jedem gesendeten Paket inkrementiert wird. Beim Empfang der Antwort wird der gespeicherte Zähler als Referenz benutzt. Die Anwendungen müssen daher die Befehle zum Senden und Empfangen alternierend benutzen, was auch genau der intuitiven Reihenfolge entspricht. Zu jedem der elf API-Befehle existieren in dem Modul die drei Funktionen, dazu kommen einige Konstanten für Signalfarben, Motormodi und ähnliches, mit denen Programme lesbarer gestaltet werden können.

6.2.6 Module kicking und oval

Die bisher vorgestellten Module reichen für eine einfache Steuerung der Bahn bereits aus. Das Programm kann einen Nodelink zu jedem beliebigen Steuerrechner unter der Bahn aufbauen und über die entsprechenden Befehle die Peripherie kontrollieren. Dabei stellt sich allerdings die Frage, welche Bauteile auf welchem Weg zu erreichen sind, denn deren Verkabelung folgt keinem berechenbaren Schema. Die Leitungen von der Oberseite der Bahn sind immer mit dem erstbesten Anschluß der nächstgelegenen Leistungselektronik verbunden.

Abbildung 6.3: Eine der beiden Kreuzungsweichen

Die Tabelle in Anhang B.2 gibt die Zuordnungen der Anschlüsse auf die Peripherie an und zeigt auch, welche serielle Schnittstelle welches Rechners jeweils verantwortlich zeichnet. Sie liegt in Form einer CSV-Datei vor und kann so automatisch ausgewertet werden. Ein Skript generiert aus den Beschreibungen der Anlage und des Testovals die beiden Module `kicking` und `oval`.

Die kleinsten Bestandteile der Beschreibung sind die Mapping-Arrays. Jeder Eintrag in ihnen entspricht einer Zeile der CSV-Datei und damit einem Peripherieteil. Die Felder enthalten die Nummern der Leistungselektronik und des Anschlusses (`node`, `connector`) sowie die Bezeichnung auf dem Gleisplan (`block`, `device`).

```
struct railway_mapping {
  unsigned node;
  unsigned connector;
  int block;
  int device;
};
```

Die Header-Dateien der Module deklarieren Konstanten mit den Blocknamen, im Programmcode wird eine Struktur zur Beschreibung der Peripherie in mehreren Schritten definiert. Ihre Felder geben einige Eckdaten wie die Anzahl der Leistungselektroniken und die textuellen Namen der Blöcke an. Wichtiger sind jedoch die Variablen, welche die Anzahlen der Peripheriegeräte und die dazugehörenden Mapping-Arrays enthalten.

```
struct railway_hardware {
  unsigned numnodes;
  unsigned numsignals;
  struct railway_mapping *signalmapping;
  unsigned numcontacts;
  struct railway_mapping *contactmapping;
  unsigned numtracks;
  struct railway_mapping *trackmapping;
  unsigned numpoints;
  struct railway_mapping *pointmapping;
  unsigned numlights;
  struct railway_mapping *lightmapping;
  unsigned numgates;
  struct railway_mapping *gatemapping;
  unsigned numgatesensors;
  struct railway_mapping *gatesensormapping;
  unsigned numbells;
  struct railway_mapping *bellmapping;
  unsigned numgatesignals;
  struct railway_mapping *gatesignalmapping;
  unsigned numblocks;
  char **blocknames;
};
```

Will ein Programm beispielsweise auf das Signal am Ausgang von OC_ST_1 zugreifen, sucht es dazu im Mapping-Array `signalmapping` den Eintrag mit `block=OC_ST_1` und `device=1`. Dabei werden maximal `numsignals` Einträge geprüft, bis der passende gefunden ist. Dessen Felder `node` und `connector` legen fest, an welcher Leistungselektronik und an welchem Ausgang das Signal angeschlossen ist. Jetzt muß nur noch der richtige Nodelink benutzt werden, um die Einstellung des Signals zu ändern oder auszulesen. Diesen Schritte muß der Programmierer nur selten selbst implementieren, normalerweise übernehmen die höheren API-Ebenen diese Aufgabe.

6.2.7 Modul railway

In der höchsten Hierarchiestufe der Library ist schließlich das Modul `railway` angesiedelt. Es abstrahiert von Nodelinks, Mapping-Arrays und ähnlichen Strukturen, um dem Benutzer einen möglichst einfachen Zugriff auf die Modellbahn zu ermöglichen. Es müssen nur am Anfang die Hardwarebeschreibung registriert und die Nodelinks angelegt werden, ab dann übernimmt ein im Hintergrund laufender Thread die Kommunikation mit der Hardware. Das Hauptprogramm arbeitet mit Funktionen, die ausschließlich auf ein Speicherabbild der Hardware zugreifen. Der Thread versendet laufend Befehlspakete vom Typ CMDGLOBAL, um den gespeicherten Status der Aktoren zu den Leistungselektroniken zu senden und bekommt im Austausch den Zustand der Sensoren für das Speicherabbild. Der Benutzer muß so sich nicht um die Kommunikation, das Timing oder die Reaktion auf Fehler kümmern. All diese Aufgaben werden transparent von dem Modul übernommen.

Verwaltung des Systems

Vor der Benutzung der Routinen muß eine Applikation die Datenstrukturen initialisieren. Dabei wird die Hardwarebeschreibung übergeben, hierbei handelt es sich um einen Zeiger auf eine der Strukturen `kicking` oder `oval` aus den gleichnamigen Modulen.

```
struct railway_system *railway_initsystem(struct railway_hardware *hardware);
```

Als nächstes werden die Nodelinks zu allen an der Steuerung beteiligten Leistungselektroniken geöffnet und eingetragen. Dies kann manuell geschehen, dann müssen die Nodelinks nur noch mit der folgenden Funktion registriert werden.

```
int railway_setlink(struct railway_system *railway, unsigned node,
                    struct nodelink *link);
```

Das Modul kann diese Aufgabe aber auch automatisch durchführen, solange nur ein Bussystem für die Steuerung benutzt werden soll. Eine Funktion stellt Nodelinks über den CAN-Bus her, es müssen nur der Name des lokalen CAN-Interfaces und der Name der verwendeten seriellen Schnittstelle auf den PC104-Knoten übergeben werden. `railway_openlinks_udp` funktioniert genauso, hier wird statt dem Namen des CAN-Interfaces ein Formatstring für den Hostnamen benutzt. Er muß an irgendeiner Stelle den Platzhalter `%i` enthalten, an seiner Stelle wird die Nummer des Knotens eingesetzt. Der String `node%02i` expandiert zu `node00` und so weiter.

```
int railway_openlinks_can(struct railway_system *railway, char *candevice,
                          char *device);
int railway_openlinks_udp(struct railway_system *railway,
                          char *hostformat, char *device);
```

Wurden alle Nodelinks erfolgreich angelegt und registriert, startet `railway_startcontrol` das gesamte System. Der Funktion werden Grenzwerte für die Dauer einer Kommunikationsrunde übergeben. Ab diesem Moment kann die Hardware normal angesteuert werden.

```
int railway_startcontrol(struct railway_system *railway,
                         unsigned mincycle, unsigned maxcycle);
```

Ob die Steuerung normal arbeitet oder durch nicht behebbare Fehler zusammengebrochen ist, läßt sich mit einem Aufruf der Funktion `railway_alive` erfragen.

```
int railway_alive(struct railway_system *railway);
```

Nachdem das Programm seine Aufgabe beendet hat, fährt `railway_stopcontrol` die Steuerung wieder herunter. Auf Wunsch wird dabei die gesamte Peripherie zurückgesetzt.

```
int railway_stopcontrol(struct railway_system *railway, int reset);
```

Die noch offenen Nodelinks kann der Aufrufer mit `int railway_closelinks` schließen lassen oder dies selbst übernehmen, falls er die Links selbst geöffnet hatte.

```
int railway_closelinks(struct railway_system *railway);
```

Zum Abschluß wird die Datenstruktur des Interfaces aufgelöst und der Speicher freigegeben.

```
int railway_donesystem(struct railway_system *railway);
```

Ansteuerung der Hardware

Eine Vielzahl von Funktionen steuert die Peripherieteile der Modellbahn. Dieser Abschnitt greift die wichtigsten exemplarisch heraus, im Modul sind noch wesentlich mehr vorhanden.

Zum Setzen eines Signals werden die Adresse in Form von Blockname und Nummer angegeben, dazu kommt eine Bitmaske mit den Signalfarben. Sie setzt sich aus den drei Konstanten `RED`, `YELLOW` und `GREEN` zusammen.

```
void setsignal(struct railway_system *railway, int block, int signal,
               int lights);
```

Ähnlich wird der Fahrstrom für einen Block geschaltet. Der Parameter `mode` wird auf `OFF`, `FWD`, `REV` oder `BRAKE` gesetzt, `target` gibt die Zielgeschwindigkeit an. Der Wert besteht aus einer der beiden Konstanten `PWM` (festes PWM-Tastverhältnis) und `SPEED` (Geschwindigkeitsregler aktiv) plus einem Wert von 0 bis 127.

```
void settrack(struct railway_system *railway, int track, unsigned mode,
              unsigned target);
```

Die Kontakte werden über zwei Funktionen abgefragt. Mit `getcontact` kann ein bestimmter Kontakt ausgelesen werden, sie liefert dann einen der vier Werte `NONE` (nicht ausgelöst), `FWD` (wurde vorwärts von einem Zug überquert), `REV` (rückwärts) und `UNI` (ausgelöst, aber keine Richtung feststellbar) zurück. Ist der Parameter `clear` nicht Null, wird das Ereignis gleichzeitig aus dem Eingangspuffer gelöscht.

```
unsigned getcontact(struct railway_system *railway, int block,
                    int contact, int clear);
```

Die Funktion `scancontact` funktioniert genauso, ihr kann für `block` und `contact` aber jeweils `-1` übergeben werden. In diesem Fall sucht die Routine den ersten ausgelösten Kontakt, liefert dessen Zustand zurück und setzt die Parameter `block` und `contact` auf die richtige Adresse. Die Funktion eignet sich also besonders dafür, auf Ereignisse zu reagieren, die irgendwo in der Anlage aufgetreten sind.

```
unsigned scancontact(struct railway_system *railway, int *block,
                     int *contact, int clear);
```

Die Kontakte dürfen nicht zu selten abgefragt werden. Ereignisse werden zwar lange gepuffert, dafür speichert das System aber nur die Richtung der aktuellsten Auslösung. Ältere liefern als Richtung grundsätzlich nur `UNI`.

Weichen kennen zwei Zustände, `STRAIGHT` und `BRANCH`. Sie können per `setpoint` für jeden Antrieb auf der ganzen Anlage gesetzt werden.

```
void setpoint(struct railway_system *railway, int point, int state);
```

Ob sich eine Lokomotive auf einem Gleisabschnitt befindet, ermittelt die Funktion `trackused`. Falls dies der Fall ist, liefert `getspeed` auch dessen aktuelle Geschwindigkeit.

```
int trackused(struct railway_system *railway, int track);
unsigned getspeed(struct railway_system *railway, int track);
```

Die meisten Funktionen unterstützen Wildcards, um mehrere Bauteile gleichzeitig zu verändern. So schaltet beispielsweise der folgende Befehl alle Signale in allen Blöcken auf Rot.

```
setsignal(railway,-1,-1,RED);
```

Diagnosefunktionen

Eine Gruppe von Funktionen des APIs beschäftigt sich mit der Diagnose von Systemfehlern, dazu werden die Fehlerzähler der Leistungselektroniken im Rahmen eines Diagnosezyklus gelesen und später ausgewertet. Den Zyklus leitet `railway_diagnostics` ein, das System muß dafür bereits online sein. Die normale Steuerung pausiert für die Dauer der Ausführung, die Steuerung sollte sich dann in einem sicheren Betriebszustand befinden.

```
int railway_diagnostics(struct railway_system *railway, int function);
```

Als Diagnosefunktionen kommen `DIAG_STUCKCONTACTS` (festhängende Kontakte ermitteln und speichern), `DIAG_DOWNLOADEEPROM` (EEPROM auslesen und speichern) sowie `DIAG_CLEAREEPROM` (EEPROM zurücksetzen) in Frage. Vier Funktionen werten die so gesammelten Daten aus.

```
int diagstuckcontact(struct railway_system *railway, int *block,
                     int *contact, int clear);
int diagfailedcontact(struct railway_system *railway, int *block,
                      int *contact, int *first, int *second, int clear);
int diagshutdowntrack(struct railway_system *railway, int *block,
                      int *count, int clear);
int diagresetnode(struct railway_system *railway, int *node,
                  int *mclr, int *wdt, int *bod, int clear);
```

Alle von ihnen durchsuchen die im Diagnosezyklus ermittelten Daten nach Fehlern und ordnen diese Peripheriebauteilen auf der Modellbahn zu. Die Anwendung kann die Informationen dem Benutzer ausgeben und im Puffer wieder löschen. Der EEPROM kann anschließend durch einen eigenen Diagnosezyklus gelöscht werden.

Weitere Informationen

Die vorgestellten Funktionen stellen nur einen Ausschnitt aus dem API dar, die vollständige Dokumentation aller Routinen befindet sich im Quellcode. Erwähnenswert ist noch, daß im laufenden Betrieb auftretende Fehler von `railway` auf `stderr` ausgegeben werden. Dazu gehören sowohl Probleme bei der Kommunikation wie auch Fehler in der Bahnhardware.

6.2.8 Modul progbar

Die Library benutzt eine Fortschrittsanzeige für einige Aufgaben wie das Öffnen einer Gruppe von Nodelinks. Anwendungen können diesen Code in Form des Moduls `progbar` nutzen.

```
Progress [oooo..............]
```

Bei der Initialisierung werden die Breite der Anzeige in Zeichen sowie Minimum und Maximum der Werte angegeben, welche angezeigt werden sollen.

```
void progbar_init(struct progbar *bar, unsigned width, int min, int max);
```

Danach kann die Anzeige mit `progbar_show` ausgegeben beziehungsweise aktualisiert werden.

```
void progbar_show(struct progbar *bar, int value);
```

Das Modul benutzt das Backspace-Zeichen, um zuvor ausgegebenen Text zu überschreiben. Daher dürfen keine anderen Ausgaben zwischen zwei Aufrufen von `progbar_show` erfolgen.

6.3 Visualisierung und Simulation

Für viele Anwendungen ist es sinnvoll, den Zustand der Modellbahn grafisch auf dem Bildschirm eines Rechners darstellen zu können. Ein Grundgerüst für diese Aufgabe stellt das Modul `visual` der Library zur Verfügung. Ein von diesem gestarteter Thread überwacht laufend den Zustand der Bahn auf Veränderungen. Dabei verhält er sich wie ein normales Anwendungsprogramm und nutzt das von `railway` angebotene Interface. Sobald ein Sensor oder Aktor seinen Zustand ändert, ruft der Thread eine Callback-Funktion auf, welche die Bildschirmdarstellung ändern soll. Das Grundgerüst gibt die Veränderung auf `stdout` aus und muß von dem Programmierer entsprechend erweitert werden.

Das Interface besteht nur aus zwei einfachen Funktionen. Die erste soll die Oberfläche erzeugen und an das laufende Bahninterface binden, die zweite hingegen terminiert die Visualisierung.

```
struct railway_visual *railway_initvis(struct railway_system *railway);
int railway_donevis(struct railway_visual *vis);
```

Ähnlich nützlich wie die Visualisierung ist ein Simulator, der die gesamte Bahnanlage mit den fahrenden Zügen realistisch nachbilden kann. Er ist ein Clone der `librailway`, welcher die originale Library ersetzen kann. So müssen die Anwendungen nicht neu compiliert oder angepaßt werden, es reicht aus, sie zur Laufzeit an die neue Library zu binden.

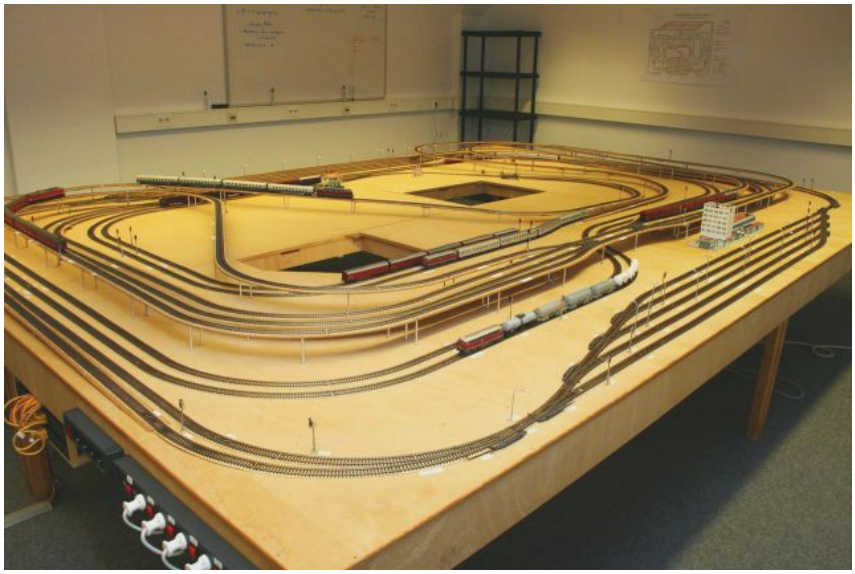


Abbildung 6.4: Fahrbetrieb auf der Anlage

Auch für diese Aufgabe bietet diese Arbeit ein Grundgerüst an, das von einem Programmierer erweitert oder mit einem existierenden Simulator verbunden werden muß. Der Code besteht zum größten Teil aus den Modulen der Bahnsteuerung, lediglich `railway` wurde gegen eine neue Version ausgetauscht. Diese implementiert alle nötigen Kontrollstrukturen und muß vom Programmierer mit der gewünschten Funktionalität ausgestattet werden. Es fehlt hauptsächlich Code, der die fahrenden Züge simuliert und die von ihnen produzierten Sensorwerte erzeugt.

Simulator und Visualisierung arbeiten problemlos zusammen, da die Visualisierung auch nur auf das normale API der Bahn zugreift. Sie können damit gleichzeitig in einem Programm eingesetzt werden.

6.4 Railway-Daemon

Wenn ein Nodelink zu einer Leistungselektronik unter der Bahn aufgebaut wird, liegt dazwischen immer ein PC104-Rechner. Ein Systemprozeß namens `railwayd` ist dafür zuständig, die lokalen seriellen Schnittstellen zu verwalten und Datenpakete zwischen diesen und den Bussystemen zu routen. Auf einen Befehl von CAN-Bus oder dem Ethernet hin öffnet und schließt dieser die lokalen Schnittstellen. Datenpakete werden von den Bussen zu den Schnittstellen und anders herum weitergeleitet. Jede Statusänderung protokolliert der Daemon über `syslog`.

Damit die Latenzzeiten nicht zu groß werden, belegt der Daemon die CPU möglichst komplett, solange mindestens eine Schnittstelle offen ist. Die Verbindungen müssen regelmäßig benutzt werden, anderenfalls gibt `railwayd` sie schrittweise wieder frei. Nach drei Sekunden Inaktivität kann ein anderer Client die offene Verbindung übernehmen, nach insgesamt 20 Sekunden wird sie automatisch geschlossen. Dieser Mechanismus verhindert, daß noch offene Nodelinks eines

abgestürzten Programms das System blockieren. Der Daemon wird beim Start mit Hilfe von Kommandozeilenparametern wie folgt konfiguriert.

```
Usage:
  railwayd [options]

This server allows clients to build nodelinks via CAN and UDP to local
serial ports.

Options:
  -c, --can dev           Enable CAN interface on the given device
  -d, --no-daemon         Don't daemonize, log messages to stdout
  -h, --help              Show this summary
  -n, --node              Set node number (starting at 0)
  -s, --simulate          Use simulated hardware for all serial ports
  -u, --udp               Enable UDP interface

The node number must be unique inside the whole system. It is taken by
default from the hostname it it contains a number (like in 'node01').
At least one interface must be enabled for this program to make sense.

Copyright 2005 Stephan Höhrmann, published under the GPL.
```

Das Programm läßt sich auch für Desktop-Rechner compilieren und dort benutzen, es ist in seinen Möglichkeiten nicht auf die PC104-Systeme beschränkt.

6.5 Anwendungsprogramme

Zu der Library gehört auch eine Reihe von Anwendungen, welche die Fähigkeiten des Codes demonstrieren oder dem Anwender bei der Fehlerdiagnose helfen. Die folgenden Abschnitte dokumentieren die zentralen Applikationen.

6.5.1 Das Programm listener

Das wichtigste Diagnoseprogramm auf der untersten Ebene ist `listener`. Es kann mit Hilfe des Adapterkabels aus Anhang A.8.4 serielle Verbindungen zwischen einem Steuerrechner und einer Leistungselektronik belauschen, um Kommunikationsprobleme aller Art zu untersuchen. Dazu wird der Adapter in die bestehende Verbindung eingefügt und sein Ausgang mit dem Rechner verbunden, auf dem `listener` läuft. Das Programm empfängt und decodiert alle Pakete, die in beiden Richtungen über das Kabel gehen. Ungültigen Daten, die von den anderen Teilnehmern ignoriert werden, erscheinen dabei nicht.

```
Usage:
  listener [options]

This program listens passively on a local serial port for incoming
packets and dumps them on screen or to a logfile.
```

```
Options:
  -b, --busywait          Use busy waiting (more accurate timing)
  -d, --device name       Serial port device, default /dev/ttyS0
  -h, --help              Show this summary
  -l, --logfile name      Logfile name, default is standard output

Copyright 2005 Stephan Höhrmann, published under the GPL.
```

Wie die Pakete ausgegeben werden, zeigt das folgende Beispiel. Jede Zeile entspricht einem Paket, sie beginnt mit einem Zeitstempel, der in Sekunden angegeben ist. Dahinter folgen ein beschreibender Text (`nodelink`), die Aktion (`send` oder `receive`) und der Inhalt des Paketes.

```
0.000 nodelink: open    logfile
0.001 nodelink: send    [request<1> connect, reset=no]
0.011 nodelink: receive [reply  <1> connect, channel=0]
0.011 nodelink: send    [request<2> keepalive]
0.021 nodelink: receive [reply  <2> keepalive]
0.022 nodelink: send    [request<3> signals, 3:??? 2:??? 1:??? 0:??R]
0.032 nodelink: receive [reply  <3> signals, 3:... 2:... 1:... 0:..R]
0.033 nodelink: send    [request<4> disconnect, reset=yes]
0.043 nodelink: receive [reply  <4> disconnect]
0.044 nodelink: close   logfile
```

In jedem Paket stehen nach der Art (`request` oder `reply`) und der jeweiligen Sequenznummer die textuell decodierten Nutzdaten. Das Beispiel stellt die Verbindung her, sendet einen Befehl vom Typ CMDKEEPALIVE, schaltet die rote Lampe im Signal 0 ein und trennt die Verbindung schließlich wieder mit einem Reset.

6.5.2 Das Programm simnode

Ein anderes Diagnoseprogramm verwandelt einen Rechner in eine simulierte Leistungselektronik ohne angeschlossene Peripherie. Sie wird über ein serielles Nullmodemkabel an einen beliebigen Steuerrechner angeschlossen. Der Bildschirm des Simulators zeigt den internen Zustand an und kann so gut überwacht werden. Das Programm eignet sich insbesondere dafür, Steuerprogramme auf anderen Systemen wie den TTP-Knoten zu untersuchen.

```
Usage:
  simnode [options]

This program transforms the computer into a simulated railway
controller node that is controlled by an external host using
a serial nullmodem cable.

Options:
  -c, --channel N               Simulator channel number [default 0]
  -d, --device dev              Serial port [/dev/ttyS0]
  -h, --help                    Show this summary
  -l, --logfile filename        Log communication to a file
```

```
  -n, --no-visual               Do not show the status display

Copyright 2005 Stephan Höhrmann, published under the GPL.
```

Der Bildschirm zeigt alle Sensoren, Aktoren und den Inhalt des EEPROMs in lesbarer Form an.

```
--------< Simulated railway controller node >---------------------------

  Channel:     0

  Signals:     (0) ...  (1) ...  (2) ...  (3) ...
  Points:      (0)  .   (1)  .   (2)  .   (3) .

  Contacts:    (0) --- <0>  (1) --- <0> (2) --- <0>  (3) --- <0>
               C0:0 . C0:1 . C1:0 . C1:1 . C2:0 . C2:1 . C3:0 . C3:1 .

  Track 0:     enabled, off, pwm ratio 0
               no train, speed 0
  Track 1:     enabled, off, pwm ratio 0
               no train, speed 0

  EEPROM:      ( 0) 0    ( 1) 0    ( 2) 0    ( 3) 0    ( 4) 0
               ( 5) 0    ( 6) 0    ( 7) 0    ( 8) 0    ( 9) 0
               (10) 0    (11) 0    (12) 0
```

6.5.3 Das Programm nodediags

Die Funktion einer echten Leistungselektronik kann mit Hilfe von `nodediags` geprüft werden. Das Programm baut Verbindungen über alle verfügbaren Arten von Nodelinks auf und testet dann auf Wunsch die Kommunikation, den gesamten Befehlssatz zur Steuerung der Hardware oder den Zustand der Fehlerzähler. Mit der angeschlossenen Diagnoseplatine (siehe Anhang A.3) ergibt sich ein effektives Gespann, das die meisten denkbaren Fehler aufdecken kann.

```
Usage:
  nodediags [options] test(s)

Diagnostic tool that can connect to a single controller node via all
available nodelink types and run tests on it as well as on the
connection itself.

Options:
  -d, --device spec       Specify all nodelink parameters
  -h, --help              Show this summary
  -l, --logfile name      Log the communication to a file
  -v, --verbose           Show more information if available

Nodelink specifications used by the device parameter:
  tty:<device filename>
```

```
    can:<local CAN device>:<remote node number>:<remote device>
    udp:<remote hostname>:<remote device>
    sim

  Many fields can be left empty and will be filled with default values that
  are identical to those shown in the following examples.
    tty:/dev/ttyS0 (default device)
    can:/dev/pcan32:0:/dev/ttyS0
    udp:node00:/dev/ttyS0
    sim

  The following test programs are available:
    link                    Basic nodelink communication
    hardware                Hardware functionality
    eeprom                  EEPROM error counters (default)

  Copyright 2005 Stephan Höhrmann, published under the GPL.
```

Kurz gesagt kann das Programm eine Statistik der Paketlaufzeiten aufstellen, die Hardware auf der Diagnoseplatine in einem beobachtbaren Tempo ansteuern sowie die Fehlerzähler im EEPROM des Mikrocontrollers auslesen und zurücksetzen.

6.5.4 Das Programm diagnostics

Die umfangreichsten Tests sind mit `diagnostics` möglich. Das Programm kann Verbindungen über serielle Kabel, CAN oder UDP zu der Anlage oder dem Testoval aufbauen und so die Kommunikation und die Modellbahnhardware selbst überprüfen. Fast alle Tests existieren in zwei Varianten, einem Schnelltest und einer detaillierten Prüfung. Beispielsweise schaltet der Schnelltest der Weichenantriebe diese in kurzer Folge vor und zurück. Der Beobachter erkennt am gleichmäßigen Klicken, ob sie funktionieren, das Lokalisieren eines Fehlers ist in der kurzen Zeit aber kaum möglich. Der detaillierte Test schaltet die Antrieb einzeln mit langen Pausen und mit Ausgabe der Nummer. So können Ausfälle einem bestimmten Antrieb zugeordnet werden.

```
  Usage:
    diagnostics [options] test(s)

  Diagnostics utility for the railway system. This program can access the
  error counters of the controller boards and perform a variety of checks
  to test the functionality of the system.

  Options:
    -c, --can device          Use CAN for communication
    -d, --detailed            Run detailed tests instead of quick tests
    -h, --help                Show this summary
    -s, --system name         Select the system to test
    -t, --tty device          Use TTY for communication
    -u, --udp                 Use UDP for communication
```

```
The following systems can be tested:
  oval                      Small test system
  kicking                   Kicking Horse Pass (default)

The following test functions can be executed on the system:
  signals                   Test signal LEDs inside the blocks
  contacts                  Test contacts, direction and location
  usedtracks                Test track usage sensors
  points                    Test point operating units
  lights                    Test lights on the whole system
  gates                     Test crossing gates, sensors and signals
  bells                     Test crossing bells
  nodelinks                 Test nodelink communication
  eeprom                    Display hardware error counters
  cleareeprom               Reset error counters to zero
  reset                     Reset all controllers
  driveic                   Drive from IC_ST_3 through inner circle
  driveoc                   Drive from OC_ST_1 through outer circle
  drivekh                   Drive from KH_ST_1 through pass
  driveov                   Drive through the test oval

Copyright 2005 Stephan Höhrmann, published under the GPL.
```

Besonders interessant ist die Möglichkeit, alle Fehlerzähler der Anlage auszulesen. Das Programm ordnet allen Gleistreibern und Kontakten die Bezeichnungen aus dem Gleisplan zu und ist so einfach zu interpretieren. Funktionen zum Abfahren der verschiedenen Kreise der Anlage runden die Testmöglichkeiten ab.

6.5.5 Das Programm apidemo

Wie Programme zur Steuerung des Fahrbetriebs aussehen, zeigt das einfache Beispielprogramm `apidemo`. Es initialisiert die Anlage für UDP-Kommunikation und fährt dann einen Zug durch den Outer Circle, Anfang und Endpunkt der Runde ist OC_ST_1. Das Programm schaltet die Lampen, Weichen und Signale entlang des Weges. Die Geschwindigkeit des Zuges wird von dem PID-Regler der Leistungselektronik auf etwa 30 Zentimetern pro Sekunde gehalten und laufend zusammen mit dem aktuell belegten Gleis ausgegeben. Löst der Zug unterwegs Kontakte aus, erscheinen die Ereignisse mit auf dem Bildschirm. Beim Erreichen des Zielblocks reduziert das Programm am ersten Kontakt die Geschwindigkeit und bringt den Zug mit dem zweiten Kontakt zum Stehen. Der Code ist sehr kompakt und kann als Vorlage für neue Anwendungen dienen.

6.6 Emulation des alten Interfaces

Einige der Programme, die während der zweiten Generation der Anlage entwickelt wurden, haben sich als sehr nützlich erwiesen. Dazu gehört insbesondere das Railway Control Center (`rcc`), das einen Fahrbetrieb mit mehreren Zügen realisieren kann. Genauso wichtig ist seine Fähigkeit, die nach einem Programmabbruch auf der Anlage verteilt stehenden Züge in ihre Bahnhöfe zurückfahren zu können. Dieses Programm basiert auf dem nicht mehr existierenden Interbus-System und steuern es über die Library `libtrack` an. Zum Glück weisen `libtrack`

und `librailway` ausreichend viele Ähnlichkeiten auf, so daß ein Emulationslayer das API der älteren Bibliothek auf die neue abbilden kann. Alte Programme wie `rcc` lassen sich daher mit minimalen Veränderungen wieder in Betrieb nehmen, sie müssen in erster Linie nur gegen den Emulationslayer gelinkt werden.

Vier Umgebungsvariablen kontrollieren das Verhalten des Emulationslayers. Das zu benutzende Netzwerk wird über `LIBTRACK` angebeben, zur Auswahl stehen `CAN`, `UDP` und `NONE`. Letzteres erlaubt den Test von Programmen auch ohne Verbindung zur Hardware. Wie das lokale CAN-Interface und die seriellen Schnittstellen auf der Remote-Seite heißen, legen `LIBTRACK_CAN` und `LIBTRACK_TTY` fest. Schließlich kann mit `LIBTRACK_VISUAL` bestimmt werden, ob die Schnittstelle für die Visualisierung aktiviert wird.

```
LIBTRACK=(NONE|UDP|CAN)
LIBTRACK_CAN=/dev/pcan32
LIBTRACK_TTY=/dev/ttyS0
LIBTRACK_VISUAL=(YES|NO)
```

Das aktualisierte Railway Control Center zeigt Abbildung 6.5. Die Veränderungen bestehen hauptsächlich in der Anbindung der Emulationsschicht, außerdem wurden die Blocknamen an den neuen Gleisplan angepaßt, einige Fehler beseitigt und die Aufräumfunktion erweitert.

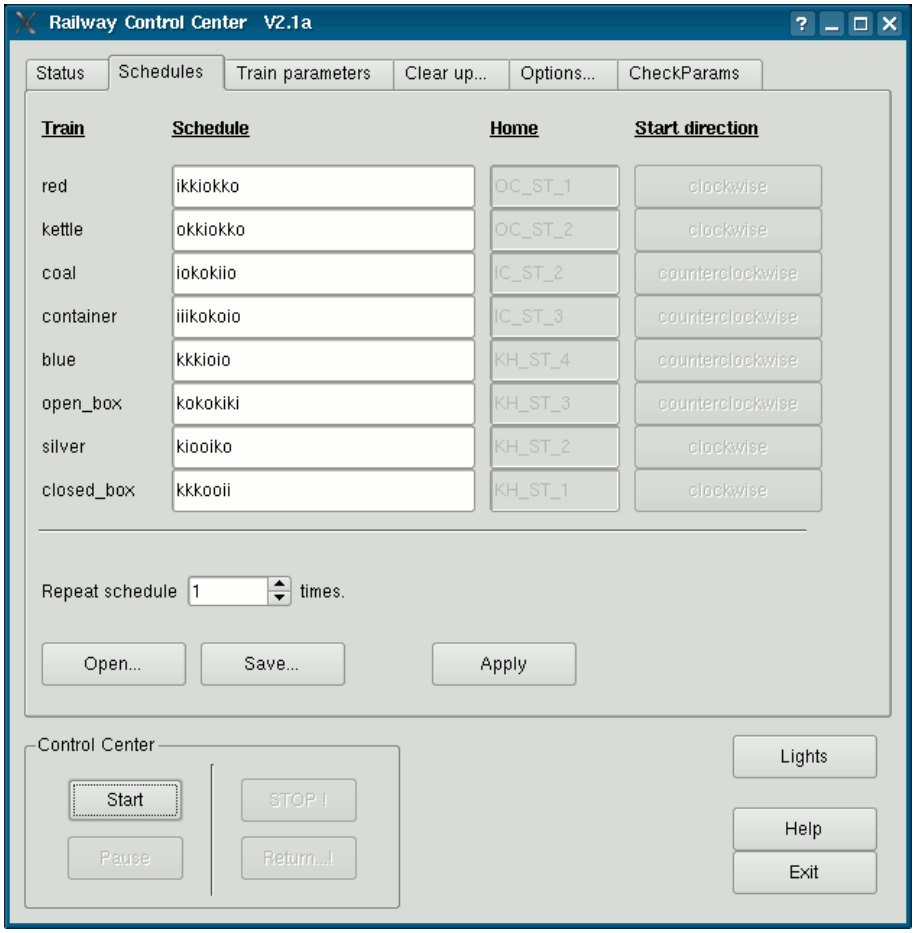


Abbildung 6.5: Aktualisierte Version von rcc

Die Emulation kann durch die unterschiedlichen Systeme nicht ganz perfekt sein. Es ergeben sich daher einige Veränderungen, die in der folgenden Liste aufgeführt sind.

- Die Zykluszeit erhöht sich von 8 Millisekunden (Interbus) auf 30 Millisekunden (UDP). Dies wirkt sich in geringem Maße auf die Reaktionszeiten der Anwendung aus.
- Der alte Block KH_LN_1 wurde in zwei aufgeteilt, die jetzt KH_LN_1 und KH_LN_2 heißen. Die Library setzt die beiden Hälften intern wieder zusammen, die neu hinzugekommenen Signale zwischen ihnen sind immer dunkel, genauso werden die Kontakte in der Mitte ignoriert. Für Anwendungen ist damit kein Unterschied sichtbar.
- Die Namen der Blöcke haben sich geändert, insbesondere in den blau gezeichneten Teilen des Gleisplans. Wenn Anwendungen die Namen in irgendeiner Form benutzen, entsprechen diese dem alten Gleisplan und unterscheiden sich damit von dem aktuellen Aufbau. Die Programme sollten bei Bedarf angepaßt werden. Dabei muß auch berücksichtigt werden, daß durch die Aufteilung von KH_LN_1 alle Folgeblöcke neu numeriert wurden.
- Kleinere Fehlerkorrekturen in der `track_data.cfg` waren nötig, um die Kreuzungsweichen richtig anzusteuern. Bei alten Programmen muß diese Datei ausgetauscht werden.
- Die Kontaktfunktionen liefern statt einer Ereignisliste maximal einen Eintrag zur Zeit.
- Der Bahnübergang ist neu und wird daher von den alten Anwendungen nicht angesteuert. Im Sinne eines realistischeren Fahrbetriebs senkt die Library von sich aus die Schranken, schaltet die Signale auf Rot und aktiviert die Glocke, sobald der Gleisbesetztmelder des betreffenden Blocks (KH_LN_2 im neuen Gleisplan) eine Lokomotive erkennt.
- Die Library setzt die Geschwindigkeitsstufen über eine Tabelle auf Geschwindigkeiten für den Regelalgorithmus um. Damit sind Anpassungen an die Steigung oder bestimmte Typen von Lokomotiven überflüssig geworden. Das schrittweise Beschleunigen ist unverändert.
- Die Funktion GetSpeed wurde erweitert. Wird ihr die Nummer eines Blocks plus 1000 übergeben, liefert sie den Wert des Gleisbesetztmelders für den betreffenden Block.
- Wenn die Verbindung zur Bahn zusammenbricht, terminiert die Library ihr Programm.

Die Emulationsschicht funktioniert zwar zuverlässig, sie sollte aber nicht für neu entwickelte Programme benutzt werden. Sie erlaubt keinen Zugriff auf viele der neuen Funktionen, außerdem sind Details der Schnittstelle problematisch. Insbesondere können Fehler von der Anwendung nicht abgefangen werden und es passiert leicht, daß allozierter Speicher nicht freigegeben wird.

Kapitel 7

TTP und Matlab

Die Steuerprogramme für das TTP-System werden mit Hilfe von Matlab/Simulink/Stateflow entwickelt. Die entstehenden Modelle legen das Verhalten auf allen Ebenen fest, beginnend bei der Ansteuerung der Peripherie über die Kommunikation der Knoten bis hin zur Zugsteuerung. Den kleinsten Baustein bildet ein Block, der die Kommunikation zwischen einem Powernode und den drei angeschlossenen Leistungselektroniken repräsentiert. Er wurde zusammen mit der übergeordneten Funktionalität im Rahmen des Fortgeschrittenenpraktikums im Wintersemester 2005/2006 von den Studenten selbst entwickelt. Dieses Kapitel beschreibt die Funktion des Blocks basierend auf der Dokumentation aus dem Praktikum.

7.1 Schnittstelle zur Hardware

Ein grundlegender Block ist für die Abstraktion der Bahnhardware zuständig. Er kommuniziert über die serielle Schnittstelle des Powernodes und den Portextender mit drei angeschlossenen Leistungselektroniken. Nach dem Herstellen der seriellen Verbindung wird in jedem Schritt des Modells ein CMDGLOBAL-Befehl an die Platinen gesendet. Die Argumente werden aus den Eingängen des Blocks zusammengestellt, seine Ausgänge fassen die Antworten der Hardware geeignet zusammen. Eine in C geschriebene S-function führt die Kommunikation durch.

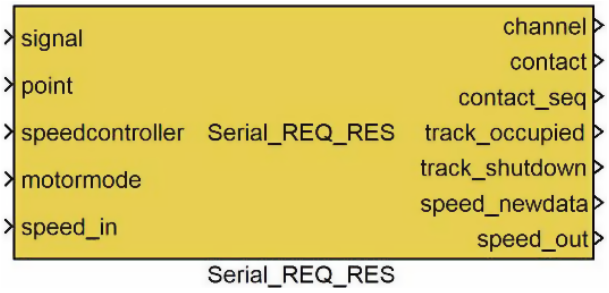


Abbildung 7.1: Request/Response Block

Bei der Programmierung ist zu beachten, daß die Ausführung des Blocks 35 Millisekunden dauert. Diese Tatsache muß bei der Definition der Schrittgrößen und Samplingzeiten des Modells berücksichtigt werden. Während der Simulation liefert der Block keine sinnvollen Daten.

7.1.1 Eingänge

Die verschiedenen Eingänge des Blocks nehmen Arrays entgegen, die an die jeweiligen Aktoren weitergeleitet werden. Da der Block drei Leistungselektroniken entspricht, ist immer ein Drittel des Arrays einer Platine zugeordnet. Bei insgesamt 12 Werten gehen die ersten vier an die erste Leistungselektronik, die nächsten vier an die zweite und die letzten vier an die dritte.

signal (12 x uint8)

Jeder der 12 Werte entspricht genau einem Signal der Modellbahn. Die Werte sind binär codiert, das Bit 0 entspricht Rot, 1 Gelb und 2 Grün. Nicht in jedem Signal sind auch alle Farben vorhanden, die Blocksignale verfügen nur über Rot und Grün, am Bahnübergang gibt es nur Rot und Gelb.

point (12 x boolean)

Die Werte geben die Einstellungen der Weichenantriebe an, 0 schaltet das Hauptgleis frei, 1 läßt den Zug auf das Nebengleis abbiegen.

speedcontroller (6 x boolean)

Wird einer der Werte auf 1 gesetzt, aktiviert dies den Geschwindigkeitsregler für den jeweiligen Gleisabschnitt. Null erlaubt die Steuerung über eine feste PWM.

motormode (6 x uint8)

Jeder Eintrag setzt den Modus für einen der Gleistreiber. Mögliche Werte sind 0 (aus), 1 (vorwärts), 2 (rückwärts) und 3 (bremsen).

speed_in (6 x uint8)

Über diesen Wert kann die Geschwindigkeit des Zuges auf einem Block eingestellt werden. Wenn der dazugehörende Regler nicht aktiv ist, entspricht dieser Wert dem Tastverhältnis der PWM. Anderenfalls gibt der Wert die Geschwindigkeit an, die der Regler erreichen soll. In beiden Fällen sind Werte von 0 bis 127 erlaubt, je größer die Zahl desto höher das Tempo.

7.1.2 Ausgänge

Die Ausgänge entsprechen nach dem selben Schema den Antworten der Leistungselektroniken.

channel (3 x int8)

Dieser Wert gibt den Status der seriellen Verbindungen zu den drei Platinen an. Ein negativer Wert bedeutet, daß durch einen Fehler keine Verbindung hergestellt werden konnte oder sie verloren gegangen ist. Anderenfalls liegt die Zahl zwischen 0 und 3 und entspricht dem aktiven Kanal der Leistungselektronik.

contact (12 x uint8)

Jeder Wert enthält das letzte Ereignis, das von dem jeweiligen Kontakt registriert wurde. Bei einem Wert von 0 wurde der Sensor noch nicht ausgelöst, 1 entspricht einem vorwärts fahrenden Zug, 2 rückwärts und bei 3 wurde der Kontakt ausgelöst, ohne daß eine Richtung feststellbar war.

contact_seq (12 x uint8)

Dieses Array enthält für jeden Kontakt den entsprechenden Sequence Counter mit einer Breite von zwei Bit. Bei jedem Ereignis wird der Zähler inkrementiert, der Benutzer kann so sehen, ob neue Daten von dem Kontakt vorliegen.

track_occupied (6 x boolean)

Das System kann feststellen, ob eine Lokomotive auf den angeschlossenen Gleisen steht. Der Wert 0 bedeutet, daß das betreffende Gleis frei ist oder daß nur Waggons darauf stehen, anderenfalls wird 1 zurückgegeben.

track_shutdown (6 x boolean)

Jeder Eintrag des Arrays signalisiert, daß der Fahrstrom in einem Block abgeschaltet werden mußte. Ursache dafür ist in der Regel ein Kurzschluß.

speed_newdata (6 x boolean)

Die Geschwindigkeit der Züge und die Belegung der Gleise werden nur in gewissen Intervallen gemessen. Wenn der dazugehörende Treiber auf Fahrt oder auf Bremsen geschaltet ist, erfolgt eine Messung alle 0,2 Sekunden, sonst beträgt die Länge der Intervalle 1,0 Sekunden. Ein Wert des Arrays wird immer dann negiert, wenn die aktuell anliegenden Daten neu gemessen wurden. Dadurch kann eine Anwendung auf frische Meßwerte reagieren.

speed_out (6 x uint8)

Diese Werte entsprechen den Beträgen der zuletzt gemessenen Geschwindigkeiten auf allen Gleisen. Der mögliche Wertebereich erstreckt sich von 0 bis 127, typischerweise werden Geschwindigkeiten bis 45 erreicht.

7.2 Adressierung der Peripherie

Dem Benutzer fällt als erstes die Aufgabe zu, die Kommunikation zwischen den Powernodes zu programmieren und dabei auch die Adressierung der Peripherie umzusetzen. Wenn die Logik auf einen Sensor oder Aktor zugreifen will, muß sie den zuständigen Knoten ausmachen und auf diesem an der richtigen Stelle auf den Request/Response-Block zugreifen. Im Praktikum war eine Dispatcher genannte Struktur für diese Aufgabe zuständig. Weitere Informationen über dessen Funktion und die Realisierung des Fahrbetriebs finden sich im gesammelten Abschlußbericht des Praktikums (siehe [Prakt06]).

Teil III

Einsatz des Systems

Kapitel 8

Fahrbetrieb

Die meisten Steuerprogramme für die Modellbahnanlage haben die selbe Hauptaufgabe zu erfüllen: Mehrere Züge sollen nach einem vorgegebenen Schema durch die Anlage fahren. Auf den ersten Blick wirkt diese Aufgabe sehr komplex, alleine für die Streckenplanung scheinen dynamische Algorithmen zum Finden von Wegen in gerichteten Graphen gebraucht zu werden, daneben müssen noch die Peripherie gesteuert und die Züge koordiniert werden.

Abbildung 8.1: Fahrende Züge in der Anlage (Foto von Jochen Koberstein)

Bei genauerem Hinsehen stellt sich heraus, daß die Zugsteuerung deutlich weniger Aufwand erfordert. Die Fahrstrecken ergeben sich statisch aus dem Fahrplan, Alternativen gibt es nur auf den Ausweichgleisen im Paß und in den Bahnhöfen, was dynamische Wegsuchen zur Laufzeit auf ein Minimum begrenzt. Für die Steuerung der Peripherie wird nur eine einfache Datenstruktur benötigt, die alle Einstellungen für jeden Streckenabschnitt speichert. Das einzige Problem ist, viele Züge gleichzeitig in Bewegung zu halten und dabei Verklemmungen zu vermeiden. Dieses Kapitel stellt mögliche Lösungen für alle Probleme des Fahrbetriebs vor und orientiert sich dabei an dem Railway Control Center von Jochen Koberstein und Oliver Schmitz.

8.1 Wichtige Grundregeln

Jedes Programm muß bei der Steuerung des Fahrbetriebs eine Reihe von Regeln einhalten. Die meisten stellen einen sicheren und lebendigen Ablauf der Fahrpläne sicher, die anderen sorgen durch kleinere Einschränkungen dafür, daß die Komplexität der Steuerung beherrschbar bleibt.

Bahnhöfe

- Jeder Zug hat sein Heimatgleis in einem der Bahnhöfe, von wo aus er Fahrten durch die Anlage startet. Es bietet sich an, den Zug am Ende seines Fahrplanes wieder auf sein Heimatgleis zu fahren, und zwar wieder in der Ausgangsrichtung. Dadurch wird bei Programmende die Ausgangssituation wiederhergestellt.
- In jedem Bahnhof muß ein Gleis für Durchfahrten freigehalten werden und darf nicht als Heimatgleis dienen. Solange ein Zug noch nicht am Ende seines Fahrplan ist, kann er den Bahnhof aber auf beliebigen Gleisen durchqueren und auf jedem Gleis anhalten, die Zuweisung der Blöcke ist variabel.

Richtungen

- Züge fahren immer mit der Lokomotive voran, Rückwärtsfahrten sind nicht erlaubt. Als Folge davon werden die Fahrpläne auch nur vorwärts abgearbeitet, einmal erfolgte Zugbewegungen dürfen nicht rückgängig gemacht werden.
- Die im Gleisplan eingezeichneten Richtungen sind verbindlich. Die Elektronik erlaubt zwar auch in Inner und Outer Circle beliebige Richtungen, dafür sind aber keine Signale vorhanden und das Befahren mit mehreren Zügen wäre deutlich komplizierter.
- Die Richtung eines Zuges im Kicking Horse Pass hängt davon ab, ob dieser vorher zuletzt im Inner oder Outer Circle war. Entsprechend ist je nach Ausfahrtrichtung nur entweder der Bahnhof des Inner oder des Outer Circles direkt erreichbar. Andere Fahrpläne sind möglich, erfordern aber den Einsatz der Wendeschleifen.
- Das Wenden eines Zuges ist ausschließlich über die beiden Wendeschleifen möglich. Will ein Zug wenden, muß er zu der einzigen für ihn erreichbaren Schleife fahren und sie durchqueren, bevor er seinen Weg mit der neuen Richtung fortsetzen kann.

Geschwindigkeit

- Die Fahrgeschwindigkeit muß den Umgebungsbedingungen angepaßt werden. Dies ist besonders wichtig, wenn der Zug vor einem Signal zum Stehen kommen soll. Bremst er nicht rechtzeitig vorher ab, stoppt der Zug nicht rechtzeitig und rutscht unter Umständen sogar bis in den Folgeblock hinein.
- Zwischen den Kontakten im Block KIO_LN_0 liegt nur wenig mehr Platz als die längsten Züge der Bahn beanspruchen. Ein Anhalten zwischen den Kontakten klappt also nur, wenn der Zug rechtzeitig die Geschwindigkeit drosselt.
- Wird das Tempo der Züge nicht von einem Regler überwacht, muß die Anwendung das Tastverhältnis manuell an die Steigung oder das Gefälle der Strecke anpassen.
- Die Züge dürfen nicht zu stark beschleunigen, sonst kann sich der vordere Teil von den hinteren Waggons losreißen. Genauso ist es möglich, daß der bei starken Bremsungen auftretende Druck Kupplungen aufbiegen und so ebenfalls zu einer Trennung des Zuges führen kann. In der Folge bleiben die Kontaktmeldungen vom hinteren Ende des Zuges aus, im schlimmsten Fall bleiben die Waggons auf einer Weiche liegen oder rollen sogar bergab auf nachfolgende Züge zu. Die Steuerung darf die Geschwindigkeit

daher nur schrittweise verändern, so sollten beispielsweise vom Stillstand bis zum Erreichen der Höchstgeschwindigkeit mehrere Sekunden vergehen.

Blöcke

- In einem Block darf sich höchstens ein Zug zur Zeit befinden. Dies schließt auch die Waggons ein, ist ein Zug über einer Trennstelle, belegt er demnach beide Blöcke.
- Bevor ein Zug in einen Block einfahren darf, muß er diesen für sich reservieren. Der Fahrstrom sowie die darin enthaltenen Signale und Weichen dürfen ab diesem Moment nur noch von dem Inhaber der Reservierung verändert werden.
- Vor der Einfahrt in einen Block müssen die auf dem Weg liegenden Weichen und die Signale für die Einfahrt gestellt werden, dann darf der Fahrstrom der beteiligten Blöcke aktiviert werden. Ab diesem Moment dürfen die Weichen und die Polarität des Fahrstromes nicht mehr verändert werden, anderenfalls kann der Zug entgleisen oder den Fahrstrom kurzschließen.
- Der Block darf erst wieder freigegeben werden, wenn der Zug den Block komplett wieder verlassen hat, also einschließlich des letzten Waggons.
- Die Ausweichgleise im Paß werden jeweils mit einer festen Richtung befahren, wie es die gestrichelten Pfeile im Gleisplan andeuten: KH_LN_4 und 6 vorwärts, KH_LN_3 und 5 rückwärts. Dadurch vereinfacht sich die Steuerung in diesem Teil erheblich.
- Eine besondere Rolle nehmen die Blöcke OC_ST_0 und OC_ST_4 ein. Sie grenzen an alle drei Ringe (Inner und Outer Circle sowie Kicking Horse Pass). Ein Zug, der vom Inner Circle in den Paß oder anders herum wechseln will, legt dabei einige Zentimeter auf dem Outer Circle zurück und kreuzt so dessen Fahrspur. Diese Tatsache muß bei der Wegplanung berücksichtigt werden.

Haltepositionen

- In fast allen Blöcken sind Wartepositionen vorhanden, wo Züge auf das Freiwerden des Folgeblocks warten können. Sie sind mit Signalen markiert, Kontakte im Gleisbett ermöglichen ein punktgenaues Anhalten vor den Signalen. Außerhalb der markierten Bereiche ist ein Halten nicht erlaubt.
- Soll der Zug in einem Haltebereich um Stehen kommen, muß die Lokomotive beim Erreichen des ersten Kontaktes die Geschwindigkeit reduzieren. Sobald sie den zweiten Kontakt erreicht, schaltet sie den Motor aus und rollt so bis direkt vor das Signal.
- In insgesamt acht Blöcken sind weder Signale noch Kontakte vorhanden, die Züge dürfen in ihnen daher auch nicht stehenbleiben. Der Grund hierfür ist, daß die Blöcke im Allgemeinen sehr kurz sind und häufig Weichen enthalten, so daß ein Zug nicht in sie paßt oder dort zu viele Wege blockieren würde. Bei diesen sogenannten Interblocks handelt es sich um die Zufahrten der drei Bahnhöfe sowie die beiden Junction-Blöcke.

Kontakte

- Die Bewegungen aller Züge werden über die Kontakte in den Gleisen verfolgt. Das Steuerprogramm kennt die Anfangspositionen und beabsichtigten Fahrwege, damit kann es die Kontakte zuordnen und daraus die aktuellen Positionen berechnen. Eine direkte Identifikation der Züge über die Kontakte ist grundsätzlich nicht möglich.
- Die Gleisbesetztmelder liefern zusätzliche Informationen, ersetzen die Kontakte aber nicht. Sie können weder die Position des Zuges in einem Block feststellen noch ihn identifizieren oder seine Richtung erkennen.

Signale

- Signale zeigen normalerweise Rot. Um einen Zug passieren zu lassen, dürfen sie auf Grün oder Gelb/Grün umspringen. Die Farbe Gelb wird immer dann benutzt, wenn zwischen dem Signal und dem nächsten eine Weiche liegt, welche den Zug auf das Nebengleis wechseln läßt, die Lok also langsamer fahren sollte.
- Wenn ein Zug vor einem Signal steht oder auf dieses zufährt und es passieren darf, muß das Signal Grün oder Gelb/Grün anzeigen. Sobald der Zug das Signal passiert und den Block verlassen hat, muß das Signal wieder auf Rot umspringen.
- Die Signale sollten jederzeit dem Zustand der Anlage entsprechen, können aber vom Fahrbetrieb unabhängig geschaltet werden.

Lebendigkeit und Fairneß

- Verklemmungen treten immer dann auf, wenn zwei oder mehr Züge sich gegenseitig so behindern, daß keiner von ihnen weiterfahren und dadurch die Situation auflösen kann. Dies passiert im einfachsten Fall dann, wenn zwei Züge in entgegengesetzte Richtungen auf eine eingleisige Strecke fahren.
- Züge dürfen nur dann in ihrem Fahrplan vorrücken, wenn dadurch keine Verklemmung entstehen kann (auch nicht als später auftretende Folge dieser Entscheidung).
- Der Betrieb sollte lebendig sein, also frei von Konflikten ablaufen und dabei möglichst viele Züge gleichzeitig in Bewegung halten.
- Bei der Ausführung des Fahrbetriebs kann es passieren, daß einzelne Züge wesentlich länger warten müssen als andere. Die Steuerung muß dies verhindern, zum Beispiel indem jedem Zug eine Priorität zugeordnet wird, die mit jeder Wartezeit steigt.
- Die Prioritäten können benutzt werden, um ein pünktliches Eintreffen im Bahnhof zu garantieren oder bestimmte Züge schneller durch eine Engstelle zu bringen als andere.
- Die Lebendigkeit ist das wichtigste Kriterium beim Mehrzugbetrieb, ein gewisses Maß an Fairneß sollte aber auch implementiert werden.

Sicherheit

- Züge dürfen nicht weiter fahren, als die Steuerung es ihnen erlaubt. Sie müssen an Signalen wie vorgesehen anhalten und dürfen sie nicht überfahren.
- Der Fahrstrom darf nur so geschaltet werden, daß es zu keinen Kurzschlüssen kommt.
- Weichen müssen immer korrekt für den einfahrenden Zug geschaltet sein und dürfen während der Durchfahrt nicht verändert werden. Anderenfalls gelangt ein Zug auf das falsche Gleis, verursacht einen Kurzschluß oder entgleist sogar.
- Die Schranken des Bahnübergangs müssen unten sein, bevor ein Zug in KH_LN_2 einfährt und sich nach dem Verlassen des Blockes wieder öffnen. Die korrekte Funktion der Schranken kann über die Lichtschranken am Bahnübergang kontrolliert werden.
- Bleiben von der Steuerung erwartete Kontaktmeldungen für einen längeren Zeitraum aus, muß das System von einem Unfall ausgehen und den Betrieb abbrechen.

Jede Bahnsteuerung, die einen lebendigen Fahrbetrieb realisieren soll, muß dabei die genannten Regeln einzuhalten. Die meisten von ihnen sind unverzichtbar, einige schränken die Beweglichkeit der Züge geringfügig ein, indem sie zum Beispiel Rückwärtsfahrten verbieten. Einerseits sind diese mechanisch nicht stabil, andererseits reduziert sich so die Komplexität bei der Koordination mehrerer Züge dramatisch, daß diese Maßnahme gerechtfertigt ist.

8.2 Verfolgen der Züge

Die auf der gesamten Anlage fahrenden Züge werden mit Hilfe der Kontakte lokalisiert, die ihre Magneten entlang der Strecke auslösen. Die Sensoren arbeiten dabei so zuverlässig, daß keine weiteren Informationen ausgewertet werden müssen. Der schlimmste in der Praxis auftretende Fall ist, daß ein einzelner Reedkontakt ausfällt oder ein Magnet zu schwach geworden ist. In solchen Fällen funktioniert meistens noch eine Sensorhälfte, der Kontakt löst verspätet aus und kann die Richtung des Zuges nicht mehr erkennen. Der Betrieb kann normal weitergeführt werden, der Fehler muß nur rechtzeitig beseitigt werden, bevor der Sensor ganz ausfällt.

Abbildung 8.2: Die Bahnhöfe von Inner und Outer Circle

Die Kontakte werden in der Regel für jeden Block einzeln ausgewertet. Die Distanz zwischen den Sensoren aufeinanderfolgender Blöcke variiert so stark, daß ein Zusammenführen der Daten schwierig ist und keinen Vorteil bringt. In der Praxis werden ohnehin nur wenige Konstellationen benötigt, die in Tabelle 8.1 enthalten sind. Sie zeigt die verschiedenen Ereignisse, die ein Zug beim Passieren eines Blockes nacheinander auslöst.

Kontaktereignisse		
Erster	**Zweiter**	**Bedeutung und Position des Zuges**
0	0	Zug noch außerhalb oder gerade bei der Einfahrt in den Block
1	0	Lokomotive passiert ersten Kontakt, Waggons sind noch davor
2	0	Zug ist komplett zwischen den Kontakten, d.h. im Haltebereich
2	1	Lokomotive erreicht zweiten Kontakt, Waggons sind noch davor
2	2	Letzter Waggon verläßt Haltebereich auf dem Weg in Folgeblock

Tabelle 8.1: Auslösesequenz der Kontakte in einem Block

Die Steuerung bemerkt den einfahrenden Zug, sobald vorderer Magnet den ersten Kontakt aktiviert. An diesem Punkt kann die Geschwindigkeit reduziert werden, um den Zug im Block

zum Stehen zu bringen. Wenig später löst der hintere Magnet den ersten Kontakt erneut aus, der Zug befindet sich jetzt komplett im Haltebereich. Erst jetzt ist der vorige Block mit Sicherheit geräumt. Wenn der vordere Magnet den zweiten Sensor erreicht, kann der Zug noch rechtzeitig vor dem Blockende zum Stehen gebracht werden, anderenfalls beginnt hier das Verlassen des Blockes. Wann dieser den Block tatsächlich komplett verlassen hat, können wieder erst die Kontakte des Folgeblocks feststellen.

8.3 Das Streckennetz

Die wichtigste Grundlage für alle Algorithmen zur Streckenplanung und zum Bewegen der Züge ist eine Datenstruktur, in der das Streckennetz und Informationen zur Steuerung der Hardware gespeichert sind. Diese Aufgabe kann ein gerichteter Graph übernehmen, dessen Knoten den Blöcken der Bahn entsprechen. Die Pfeile des Graphen zeigen, welche Blöcke direkt miteinander verbunden sind und geben gleichzeitig die erlaubten Fahrtrichtungen an.

Eine Kantenmarkierung trägt Informationen darüber, welche Einstellungen der Hardware für den jeweiligen Blockwechsel nötig sind. Dazu gehören die Polarität des Fahrstroms auf dem Anfangsblock, dem Endblock sowie eventuell dazwischenliegenden Interblocks. Die Einstellungen der auf dem Weg liegenden Weichen wird genauso in der Markierung codiert wie die Farbe des Signals am Ende des Anfangsblocks. Die Bedeutung wird an einigen Beispielen schnell klar.

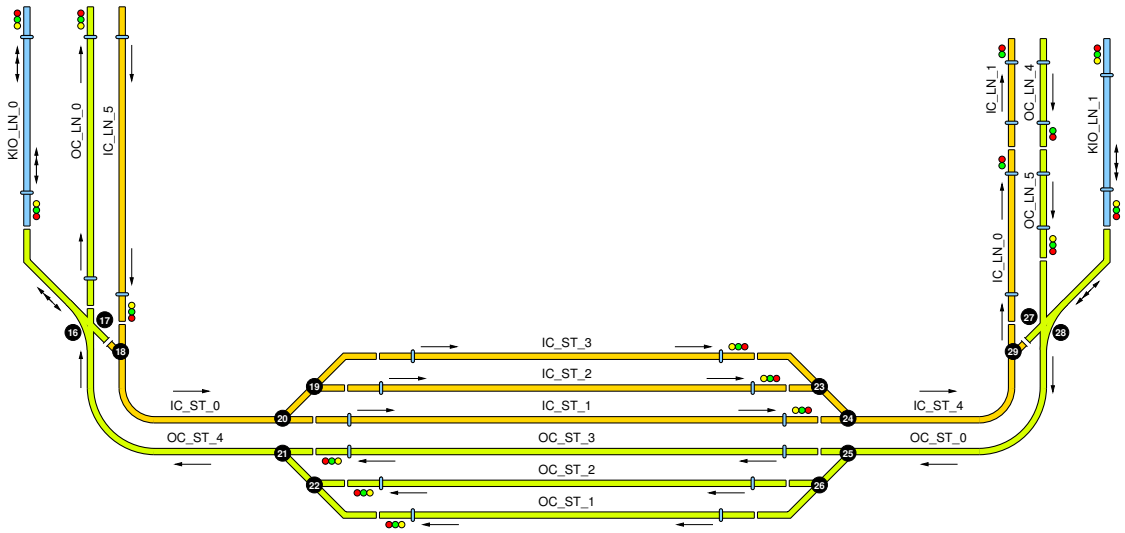


Abbildung 8.3: Ausschnitt des Gleisplanes

Ein besonders einfacher Fall liegt vor, wenn ein Zug von IC_LN_0 nach IC_LN_1 wechseln will. Der Fahrstrom in beiden Blöcken wird auf vorwärts eingestellt, Weichen sind nicht betroffen und das Signal am Ausgang von IC_LN_0 wird auf Grün geschaltet.

Etwas schwieriger ist eine Fahrt von OC_ST_3 nach OC_LN_0. Diesmal ist neben den beiden Blöcken noch ein Interblock beteiligt, daher müssen OC_ST_3, OC_ST_4 und OC_LN_0 auf Vorwärtsfahrt eingestellt werden. Die Weichenantriebe 21, 16 und 17 sind auf Geradeausfahrt zu schalten, das Signal am Ende von OC_ST_3 auf Grün, die gelbe Lampe bleibt aus.

Der Wechsel von KIO_LN_0 nach IC_ST_3 ist einer der beiden komplexesten Blockwechsel, da hier zwei Interblocks überquert werden. Die Gleistreiber von KIO_LN_0 und OC_ST_4 werden

auf Rückwärtsfahrt, IC_ST_0 und IC_ST_3 auf Vorwärts eingestellt. Die Weichenantriebe 16, 17 18 und 20 müssen zum Abbiegen, 19 für eine Fahrt auf dem Hauptgleis eingestellt werden. Das Signal am unteren Ende von KIO_LN_0 zeigt die Farbkombination Gelb/Grün an, weil auf dem Weg zum nächsten Signal (am Ende von IC_ST_3) auf Abbiegen geschaltete Weichen liegen. Damit sind die Weichen mit den Antrieben 18 und 20 gemeint, da die Kreuzungsweiche auf geradem Weg überquert wird.

In Anhang B.6 ist eine Tabelle mit der vollständigen Kantenmarkierung angegeben, die sich als Grundlage für Steuerprogramme eignet. Dabei handelt es sich um eine Erweiterung der Konfiguration, die im Railway Control Center benutzt wird. Die Interblocks gehören nicht zu den Knoten des Graphen, weil diese keine vollwertigen Blöcke sind. Sie sind sehr kurz und enthalten außerdem mehrere Weichen, so daß ein Zug bereits vor der Einfahrt in einen Interblock entscheiden muß, auf welchem Weg er diesen wieder verlassen wird. Damit macht es keinen Sinn, einen Interblock als Knoten darzustellen, von dem Pfeile zu mehreren anderen Blöcken führen.

8.4 Ablauf der Blockwechsel

Jeder Zug reserviert vor Antritt der Fahrt sein Heimatgleis und bewegt sich von dort nach einem einfachen Schema durch die Anlage. Sein Weg entspricht einem Pfad im Graphen des Streckennetzes, die Kanten liefern dabei alle Informationen zur Steuerung der Hardware. Beim Wechsel von einem Block zum nächsten werden die folgenden Schritte ausgeführt.

- Die Entscheidung, ob der Zug überhaupt weiterfahren darf, trifft der Code zum Vermeiden von Konflikten und zur Einhaltung der Fairneß (mehr dazu in Abschnitt 8.6).
- Der Zug muß den Folgeblock sowie alle auf dem Weg befindlichen Interblocks zusammen reservieren, bevor er den aktuellen Block verlassen darf. Dadurch wird verhindert, daß ein weiterer Zug in den Bereich fahren oder die Peripherie verstellen kann.
- Bei Erfolg werden alle auf dem Weg liegenden Weichen gestellt. Das Signal unmittelbar vor dem Zug wechselt auf Grün oder Gelb/Grün, der Fahrstrom auf allen reservierten Blöcken wird mit der richtigen Polarität aktiviert. Die Geschwindigkeit erhöht sich dabei langsam und auf allen beteiligten Blöcken synchron.
- Sobald der erste Kontakt des Zielblocks das erste Mal ausgelöst wird, muß die Entscheidung getroffen werden, ob der Zug durchfahren kann oder am Blockende halten muß. Im zweiten Fall wird die Geschwindigkeit jetzt reduziert.
- Wenn der Zug im Zielblock steht (also den Kontakt an seinem Eingang zweimal ausgelöst hat), werden der Startblock und die Interblocks wieder freigegeben. Deren Fahrstrom wird abgeschaltet und das Signal am Ende des Startblocks wechselt auf Rot.
- Kann der Zug nicht weiterfahren, hält er beim Erreichen des zweiten Kontaktes an, sonst wiederholt sich dieses Schema mit dem nächsten Block.

8.5 Routenplanung

Die Fahrpläne der einzelnen Züge bestehen jeweils aus einer Sequenz von Bahnhöfen, die es in der gegebenen Reihenfolge anzufahren gilt. Für die Umsetzung muß die Bahnhofssequenz jedoch erst in eine entsprechende Blocksequenz umgewandelt werden. Das Aufspüren einer passenden Route können Algorithmen zur Wegfindung in Graphen übernehmen, die für jedes mögliche Paar

von Start- und Zielbahnhof einen Pfad durch den Graphen ermitteln. Aus diesen Bausteinen lassen sich dann Blocksequenzen des Fahrplanes zusammensetzen. Für die Suche werden die Parallelgleise der Bahnhöfe zusammengefaßt, die Steuerung muß später dynamisch entscheiden, welches Gleis sie bei der Einfahrt in einen Bahnhof belegen will.

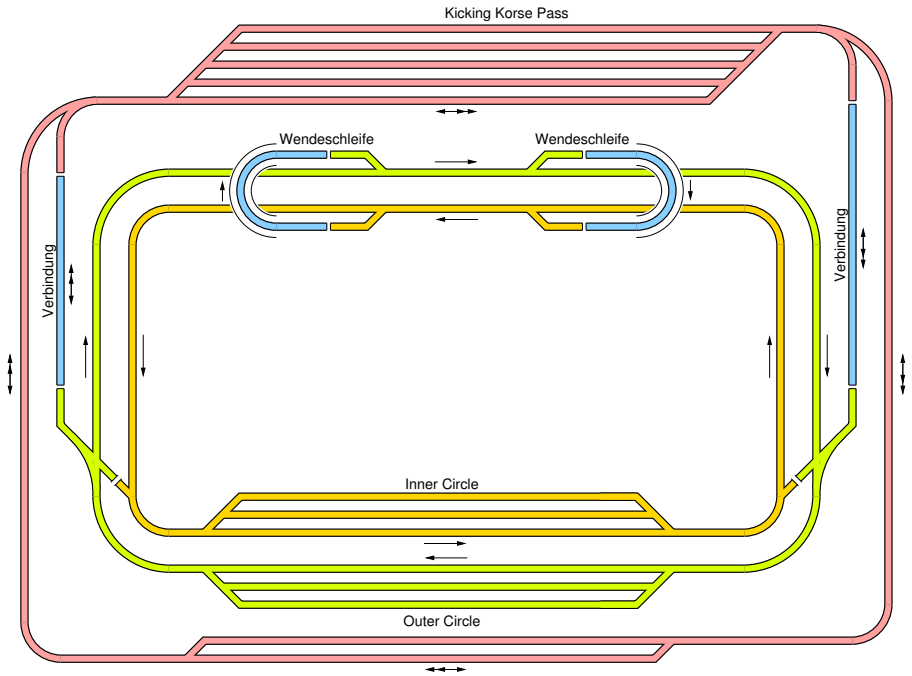


Abbildung 8.4: Die verschiedenen Kreise der Bahnanlage

Dabei näherem Hinsehen stellt sich heraus, daß es genau 16 Pfade durch den Graphen gibt, die Bahnhöfe miteinander verbinden. Jeder Pfad ist dabei eindeutig festgelegt.

- Durchquerung IC, OC, KHP vorwärts, KHP rückwärts
- Wechsel von IC in OC und anders herum (über Wendeschleifen)
- Wechsel von KHP (vorwärts) in OC und anders herum
- Wechsel von KHP (rückwärts) in IC und anders herum
- Wechsel von KHP (vorwärts) in IC und anders herum (über Wendeschleifen)
- Wechsel von KHP (rückwärts) in OC und anders herum (über Wendeschleifen)
- Wenden im KHP von vorwärts nach rückwärts (über OI-Wendeschleife)
- Wenden im KHP von rückwärts nach vorwärts (über IO-Wendeschleife)

Eine Fahrt vom Outer Circle in den Inner Circle gehört zu den kürzeren Strecken. Sie verläuft wie folgt, dabei sind die Bahnhofsgleise durch Sternchen angedeutet und alle Interblocks weggelassen.

$$\text{OC\_ST\_*} \rightarrow \text{OC\_LN\_0} \rightarrow \text{OI\_LN\_0} \rightarrow \text{OI\_LN\_1} \rightarrow \text{OI\_LN\_2} \rightarrow \text{IC\_LN\_5} \rightarrow \text{IC\_ST\_*}$$

Die vollständige Liste aller Pfade ist im Anhang B.5 enthalten. Mit diesen Voraussetzungen sollte es einfach sein, einen einzelnen Zug auf beliebigen Strecken durch die Anlage fahren zu lassen. Schwieriger wird es erst, wenn weitere Züge ins Spiel kommen.

8.6 Mehrzugbetrieb

Damit die Steuerung mehrere Züge auf der Anlage fahren lassen kann, müssen Vorkehrungen getroffen werden, die Unfälle und Verklemmungen verhindern. Einen ersten und sehr wichtigen Beitrag liefert hierfür das Reservieren der Blöcke. In Blöcken, die auf dem Fahrweg mehrerer Züge liegen, kann es so nicht zu Zusammenstößen kommen, genausowenig kann ein Zug die für einen anderen wichtigen Weichen verstellen oder dessen Fahrstrom beeinflussen.

Desweiteren kann es bei einer sehr ungünstigen Steuerung zu einer gegenseitigen Behinderung der Züge kommen, so daß der Betrieb zum Erliegen kommt. Dies ist beispielsweise der Fall, wenn je drei Züge aus entgegengesetzten Richtungen in den Kicking Horse Pass einfahren. Die Ausweichgleise in der Mitte der Strecke bieten nicht genug Platz, damit die Züge einander passieren können. Damit ist eine Situation entstanden, in der sich kein Zug mehr bewegen kann. Das steuernde Programm muß einzelne Loks gezielt in sicheren Bereichen warten lassen, um Verklemmungen dieser Art zu verhindern. Dafür gibt es eine Reihe von Lösungsansätzen, allgemein läßt sich sagen, das umso mehr Aufwand nötig wird, je mehr Züge vorhanden sind und je weniger Ausweichmöglichkeiten existieren.

Schließlich darf die Steuerung keinen Zug übermäßig lange warten lassen. Dieser Aspekt stellt das dritte und letzte Problem im Mehrzugbetrieb dar, es läßt sich über Prioritäten lösen, die jedem Zug zugeordnet werden und die mit der Wartezeit oder der Verspätung steigen.

8.6.1 Inner und Outer Circle

In den beiden großen Kreisen der Bahn kann es nicht zu Verklemmungen kommen. Sie dürfen nur in einer Richtung befahren werden, außerdem enthält jeder Kreis einschließlich der Bahnhöfe neun echte Blöcke. Dadurch ist beim Befahren mit dem Maximum von acht Zügen sichergestellt, daß immer mindestens ein Zug einen Block vorrücken kann. Verteilen sich die Züge über beide Kreise und die Wendeschleifen, ist die Situation noch einfacher. Problematisch wäre es nur, wenn die Züge ständig zwischen den Wendeschleifen wechseln sollten, da entlang dieses Weges nur sechs Blöcke existieren. Hier dürfen sich daher maximal fünf Züge bewegen.

Abbildung 8.5: Ein Zug verläßt den Outer Circle Bahnhof (Foto von Jochen Koberstein)

Die Steuerung kümmert sich hier im Wesentlichen um die Fairneß. An den Ausfahrten der Bahnhöfe und den Einmündungen der Wendeschleifen kann es passieren, daß mehrere Züge um den Nachfolgeblock konkurrieren. Diese Situation läßt sich über Prioritäten beziehungsweise Zufallsentscheidungen einfach auflösen.

8.6.2 Kicking Horse Pass

Wesentlich komplexer gestaltet sich die Organisation der in beiden Richtungen befahrbaren Paßstrecke. Verklemmungen passieren hier sehr schnell und es gilt sehr viele Situationen zu bedenken. Dabei müssen neben dem Paß auch die KIO-Zufahrten berücksichtigt werden.

Abbildung 8.6: Voll besetzter Bahnhof im Kicking Horse Pass (Foto von Jochen Koberstein)

Die einfachste Strategie für die Paßstrecke erlaubt nur eine Fahrtrichtung zur Zeit. Züge mit der entsprechenden Richtung dürfen in den Paß einfahren, die anderen müssen im Bahnhof warten. Nach einer gewissen Zeit wechselt die Steuerung die vorgeschriebene Richtung. Noch auf der Strecke befindliche Züge müssen sie verlassen, dann dürfen die bisher wartenden Züge sich auf den Weg machen. Die Entscheidung, wann die Richtung gewechselt wird, hängt von mehreren Faktoren ab. Dabei ist zu berücksichtigen, ob in der Gegenrichtung überhaupt Züge warten und wie lange sie gegebenenfalls schon warten. Ohne Ausweichgleise ist diese Steuerung die einzig mögliche, sie erreicht aber nur einen geringen Durchsatz.

Eine bessere Ausgangssituation ergibt sich, wenn der Kicking Horse Paß in drei Teile zerlegt wird. Dabei handelt es sich um das Ausweichgleis in der Mitte und die beiden einspurigen Abschnitte, die von dort zum Bahnhof führen. Die zuführenden Strecken müssen nach dem beschriebenen Schema mit einer wechselnden Fahrtrichtung betrieben werden, den Ausweichgleisen ist eine Richtung fest zugewiesen. In diesem Szenario können pro Richtung zwei Züge in den Paß fahren, in einer davon sogar mehr, ohne daß es zu einer Verklemmung kommt. Die Steuerung muß nur die Fairneß im Auge behalten, damit nicht eine Richtung bevorzugt wird oder Züge sehr lange in der Ausweichstelle stehen bleiben. Die geforderten Eigenschaften können über Petrinetze elegant gelöst werden. Die Arbeit von [Hielscher] gibt diese für die erste Version der Bahnanlage an, in der Arbeit von [Koberstein] sind diese an die zweite Generation angepaßt.

Weitere Konflikte können entstehen, wenn Züge aus den anderen Kreisen in den Paß wechseln wollen. Starten ein Zug aus dem Inner Circle und einer aus dem Outer Circle in Richtung des mit vier Zügen besetzten Paßbahnhofs, kommt es zwangsläufig zu einer Verklemmung. Eine einfache Lösung des Problems wäre, generell nur fünf Züge im Paß zu erlauben und jeden weiteren in seinem Bahnhof warten zu lassen, bis der Kicking Horse Pass wieder frei ist.

Allen Strategien ist gemeinsam, daß sie vorbeugend funktionieren. Kritische Situationen müssen durch eine entsprechende Planung verhindert werden, da sie sich nicht mehr auflösen lassen, wenn sie einmal entstanden sind.

8.7 Abschließende Bemerkung

Es scheint auf den ersten Blick die eleganteste Lösung zu sein, das Streckennetz der Bahn aus einer Datei zu laden und zur Laufzeit alle anderen Informationen daraus zu berechnen. In der Theorie kann so das selbe Steuerprogramm auch auf einem erweiterten oder umgebauten System weiterbenutzt werden. Der Ansatz scheitert allerdings daran, daß die Strategien zur Vermeidung von Verklemmungen extrem stark von Details des Gleisplanes abhängen. Kleine Unterschiede entscheiden, ob ein komplizierter Algorithmus nötig ist oder ganz auf regulierende Maßnahmen verzichtet werden kann. Davon sind insbesondere auch die in der Vergangenheit benutzten Petrinetze betroffen, sie müssen auch bei kleinsten Veränderungen des Streckennetzes umgebaut werden. Solange kein Algorithmus diese Aufgabe übernehmen kann, macht es keinen Sinn, ein universelles Programm für die Steuerung schreiben zu wollen.

Kapitel 9

Ergebnisse

Während dieser Arbeit wurde die Modellbahnanlage zum zweiten Mal grundlegend umgebaut, um Schwachstellen zu beseitigen und die Einsatzmöglichkeiten zu erweitern. Am wichtigsten war dabei die modulare Integration mehrerer Bussysteme, die jeweils die gesamte Steuerung übernehmen können. Dieses Kapitel zieht eine Bilanz des Umbaus und wirft dabei einen Blick auf die Eignung der verschiedenen Netzwerke für die Steuerung unter Echtzeitbedingungen. Ein allgemeiner Vergleich zwischen CAN und TTP findet sich in [Kopetz].

9.1 Vergleich der Bussysteme

Die Kommunikation in einem verteilten System kann auf zwei unterschiedlichen Wegen erfolgen. Bei einem ereignisgesteuerten Ansatz werden Nachrichten nur dann verschickt, wenn sich der Zustand ändert. Dies ist in der Bahn beispielsweise der Fall, wenn ein Kontakt ausgelöst wurde oder wenn die Anwendung eine Weiche umschalten möchte. Diese Ereignisse können jederzeit auftreten, dazwischen werden keine Nachrichten gesendet. Im Gegensatz dazu tauschen Knoten in einem zeitgesteuerten System periodisch Nachrichten mit dem vollständigen Zustand aus.

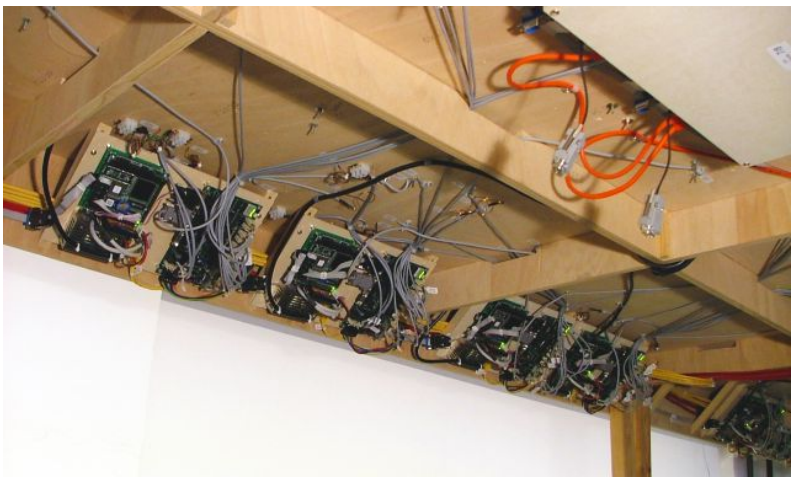


Abbildung 9.1: Steuermodule der Anlage

Der zeitgesteuerte Ansatz ist in einigen Punkten vorteilhafter. Ausfälle eines Knotens werden durch die ausbleibende Kommunikation schnell bemerkt und verlorene Pakete wirken sich nur

in Form einer verspäteten Reaktion aus, Information geht dabei nicht verloren. Dafür ist die Netzlast ständig auf einem konstant hohem Niveau. Die Hardware der Bahn unterstützt beide Arten der Kommunikation, den Anwendungen stehen daher beide Ansätze offen.

Allerdings dauert ein kompletter Austausch des Systemstatus mit allen PC104-Knoten kaum länger als die Übertragung eines einzelnen Befehls, weil die Zeit von der Dauer der RS232-Kommunikation und der Latenzzeit der Linux-Rechner dominiert wird. Die Implementierung eines ereignisgesteuerten Protokolls lohnt sich daher nicht. Die Library benutzt stattdessen ein periodisches Sendeschema, bei dem der Steuerrechner zuerst Befehle an alle Knoten sendet, diese ausführen läßt und dann die Antworten zurückerhält. Die Befehle werden nebenläufig ausgeführt, so daß die verfügbare Zeit effektiv ausgenutzt wird. TTP erlaubt nur zeitgesteuerte Kommunikation, damit arbeiten alle Netze nach dem selben Prinzip und sind so vergleichbar. Die folgende Gegenüberstellung untersucht, wie gut sich das vorgegebene Kommunikationsschema umsetzen läßt. Einige Stärken von TTP kommen in der Bahn nicht zum Tragen, beispielweise ist durch die begrenzte Anzahl der Knoten kein Redundanzmanagement möglich.

9.1.1 Berechenbarkeit der Paketlaufzeiten

Eine Nachricht kann nur innerhalb einer vorgegebenen Zeitspanne übertragen werden, wenn das Netzwerk entsprechende Mechanismen bereitstellt. Sonst können andere Stationen das Medium blockieren oder Kollisionen hervorrufen und dadurch beliebig lange Verzögerungen erzeugen. Am anfälligsten für solche Probleme ist Ethernet mit Bus-Topologie. Kollisionen sind unvermeidlich, führen zum Abbruch aller laufenden Übertragungen und werden nichtdeterministisch aufgelöst. Je höher die Netzlast ist, desto wahrscheinlicher sind Übertragungen mit zu hohen Laufzeiten. Erst der durchgängige Einsatz von Switches verbessert die Situation, aber auch hier können Pakete mit hoher Priorität von welchen niedrigerer Priorität aufgehalten werden.

Die Übertragungsrate bei CAN ist wesentlich geringer, dafür sind Kollisionen auf dem Bus nicht destruktiv, da sich die Nachricht mit der höchsten Priorität gegen andere durchsetzt. Damit ist es möglich, die Laufzeit von wichtigen Nachrichten zu begrenzen. Dies geschieht allerdings auf Kosten der Pakete niedrigerer Priorität, bei hohem Durchsatz können für sie keine Zeiten garantiert werden. Ethernet und CAN erfordern also eine vorausschauende Programmierung, die durch entsprechende Sendezeitpunkte und -häufigkeiten Konflikte bereits im Vorfeld vermeidet.

Genau diese Idee bildet die Basis von TTP. Hier gibt ein für alle Teilnehmer verbindliches Zeitschema an, welche Station zu welchem Zeitpunkt welche Daten senden wird. Kollisionen können so gar nicht erst auftreten und die Laufzeiten sind exakt bekannt. Während es bei den anderen Netzen von vielen Faktoren abhängt, ob das Sendeschema eingehalten wird, garantiert TTP den Ablauf durch die Architektur.

9.1.2 Praktischer Einsatz

Ethernet und CAN lassen sich in einem kleinen Testszenario mit zwei Knoten schnell in Betrieb nehmen. Das Betriebssystem beziehungsweise die Library des Herstellers übernehmen einen Großteil der Aufgaben, so daß die Programme sich auf wenige Zeilen Code beschränken. Mit etwas Aufwand können diese auf eine beliebige Anzahl von Knoten erweitert werden und so zur Steuerung der gesamten Bahn benutzt werden.

Die Ausführung eines Befehls durch die Leistungselektronik dauert einschließlich der seriellen Kommunikation maximal 10 Millisekunden. Da beide Netzwerke um ein Vielfaches schneller sind, dürfte ein vollständiger Zyklus des Steuerprogramms kaum länger dauern. In der Praxis

verschlechtert sich die Zeit allerdings durch die Latenzzeit des Betriebssystems deutlich. Die typische Zyklusdauer beträgt bei Ethernet 30 Millisekunden, bei CAN sogar 65 Millisekunden. Zeitschranken lassen sich nur empirisch feststellen und müssen relativ hoch liegen, damit im regulären Betrieb möglichst alle Pakete rechtzeitig eintreffen. Bei UDP sind 75 Millisekunden nötig, bei CAN reichen 80 Millisekunden. Erhöht sich die Systemlast, weil sich beispielsweise ein Benutzer auf einem PC104-Rechner einloggt, sind die Zeitschranken nicht mehr einzuhalten. Durch die niedrige Netzlast und das Sendeschema behindern sich die Übertragungen kaum.

Bei TTP liegt die Hürde für den Einstieg wesentlich höher, weil die Programmierung einiges an Erfahrung erfordert und sich am zeitgesteuerten Ablauf der Kommunikation orientieren muß. Bereits bei der Planung müssen das Zeitschema und die Ausführungszeiten der späteren Tasks berücksichtigt werden. Die Entwicklungswerkzeuge leisten hier wenig Hilfestellung, statt einer integrierten WCET-Analyse bleibt häufig nur das Ausprobieren. Bei Übersetzungszeiten von teilweise über 30 Minuten kann sich dies entsprechend in die Länge ziehen. Fehler im laufenden Betrieb sind schwierig zu diagnostizieren, weil die dafür vorgesehene serielle Schnittstelle für die Bahnsteuerung benötigt wird.

Wenn das System schließlich läuft, arbeitet es zuverlässig nach dem vorgegebenen Zeitschema. Die Zyklusdauer beträgt 100 Millisekunden, darin sind die sequentielle Kommunikation mit drei Leistungselektroniken pro Knoten sowie der Ablauf der Steuerprogramme enthalten.

9.1.3 Zusammenfassung

In der Modellbahn ist es möglich, den Aufwand für die Programmierung eines Feldbusses und seine Eigenschaften im Betrieb zu vergleichen. Alle Bussysteme werden nach dem selben Prinzip eingesetzt. Die Kommunikation erfolgt zeitgesteuert, ihr Ablauf hängt nur von der Zeit und nicht von Ereignissen im System ab. Dieser Ansatz eignet sich am besten für die Steuerung.

Bei CAN und Ethernet ist die Programmierung sehr einfach. Es gibt keine Notwendigkeit, das Zeitverhalten zu spezifizieren oder Ausführungszeiten auf garantierte Werte zu begrenzen. Dafür sind umgekehrt auch keine sicheren Annahmen über das Zeitverhalten des Systems möglich.

TTP erfordert deutlich mehr Aufwand bei der Planung und Modellierung, der Lohn dafür ist ein garantiertes Zeitverhalten und eine hohe Zuverlässigkeit, was gerade für höhere Protokollebenen wichtig ist. Der Aufwand könnte durch verbesserte Entwicklungswerkzeuge und Möglichkeiten zur Fehlerdiagnose deutlich reduziert werden, außerdem sind die Übersetzungszeiten entschieden zu lang. Dies mag daran liegen, daß die Modellbahn komplexer als die meisten TTP-basierenden Systeme ist und die Werkzeuge hier über den normalen Rahmen hinaus benutzt wurden.

In Übereinstimmung mit [Kopetz] ist festzustellen, daß TTP für sicherheitskritische Systeme in jedem Fall die bessere Wahl ist. In der Steuerung der Modellbahn kann der Bus als einziges System die Anforderungen an das Zeitverhalten zuverlässig umsetzen. Dafür sind Entwicklung und spätere Anpassungen der Software deutlich schwieriger. Letzteres ist der größte Vorteil von CAN und Ethernet, deren Zeitverhalten aber nur statistisch beschrieben werden kann.

9.2 Mögliche Erweiterungen

Diese Arbeit versteht sich als Grundlage für zukünftige Arbeiten auf der Anlage und läßt daher reichlich Spielraum für Erweiterungen und neue Programme. Die folgende Liste enthält einige Ideen, die sich gut umsetzen lassen und die Nutzbarkeit verbessern sollten.

Benutzerschnittstelle für TTP-Programme

- Momentan müssen bei der Modellierung einer Bahnsteuerung mit Matlab alle Ebenen berücksichtigt werden, der Programmierer kann sich nicht nur auf das Verhalten der Züge konzentrieren. Dies erlaubt zwar einen tieferen Einblick in die Funktionsweise, ist aber der Erstellung komplexer Programme hinderlich. Eine Schnittstelle könnte Routineaufgaben übernehmen und so den Programmierer entlasten.

Visualisierung

- Besonders nützlich wäre eine grafische Oberfläche, die den Zustand der Anlage und die Position der Züge auf dem Bildschirm eines PCs darstellen kann. Der Schnittstelle zur Bahnsteuerung ist in der C-Library bereits vorhanden, es müssen nur noch die Zeichenroutinen implementiert werden. Im Zusammenspiel mit einem Simulator oder aufgezeichneten Testläufen ergeben sich viele wichtige Anwendungsfelder.

Simulation

- Die Anlage kann immer nur von einer Gruppe zur Zeit benutzt werden, außerdem führen Fehler oft zu Situationen, in denen Züge manuell zurückgefahren werden müssen. Daher ist ein Simulator ein wichtiges Werkzeug zum Testen. Dieser muß hauptsächlich die Zugbewegungen anhand der aktuellen Einstellungen nachbilden und entsprechende Sensorwerte generieren. Auch hier ist in der C-Library bereits ein Gerüst vorhanden, das zu einem Simulator ausgebaut werden kann.

Neue Peripherie und weitere Feldbusse

- Es sind noch ausreichend viele Anschlüsse der Leistungselektroniken frei, um weiteres Modellbahnzubehör in die Anlage einzubauen.
- Die Leistungselektroniken bieten noch Platz für zwei weitere Steuerrechner, die mit neuen Feldbussystemen ausgestattet sein können. Genauso besteht die Möglichkeit, die PC104-Rechner mit weiteren Zusatzplatinen zu versehen.

Redundanz

- Alternativ können die freien Eingänge auch benutzt werden, um eine redundante Steuerung aufzubauen. Fällt ein Rechner aus, kann 0,2 Sekunden später ein anderer die Peripherie übernehmen. Dafür müssen lediglich neue Kabel verlegt werden.

Verteilte Zugsteuerung

- Momentan beschränken sich die Rechner in der Anlage darauf, Befehle und Antworten für den eigentlichen Steuercomputer weiterzuleiten. Genauso wäre es denkbar, ihnen Aufgaben wie die Organisation des Fahrbetriebs zu übertragen.

Zeitgesteuerter Fahrbetrieb

- Das Railway Control Center organisiert zwar den fehlerfreien Verkehr, die Fahrpläne des Programms sehen aber keine Zeitangaben für Abfahrt und Ankunft der Züge vor. Ein neuer Ansatz könnte basierend auf diesen Daten den Betrieb organisieren.

9.3 Fazit

Im Laufe dieser Arbeit ist ein komplexes und gut funktionierendes System zur Steuerung der Modellbahnanlage entstanden. Die Peripherie ist über alle Feldbusse steuerbar, Anwendungen können alle für den Fahrbetrieb nötigen Aufgaben einfach abwickeln. Dem Programmierer steht eine Library zur Verfügung, die Schnittstellen auf allen Ebenen bietet und die unterschiedlichsten Anwendungen möglich macht. An der Modellbahn läßt sich die Programmierung eines verteilt arbeitenden Echtzeitsystems genauso durchführen wie fehlertolerante Kommunikation in einem größeren Netzwerk. Am interessantesten ist natürlich die Programmierung der Feldbusse und die Möglichkeit, diese im Anwendungskontext vergleichen zu können.

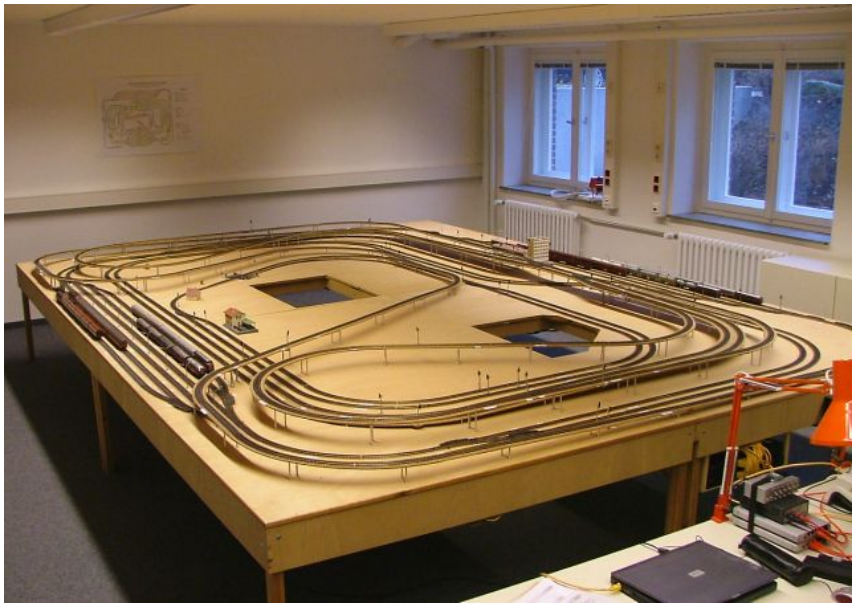


Abbildung 9.2: Die Modellbahnanlage

Die in der Vergangenheit auftretenden Probleme wurden beseitigt, beispielsweise arbeiten die Kontakte zuverlässig und die Wendeschleifen nehmen keine Sonderrolle unter den Gleisen mehr ein. Elektromagnetische Störungen machen sich nicht bemerkbar, Kurzschlüsse des Fahrstroms werden schnell erkannt, nach Beseitigung der Ursache reaktiviert sich das Gleis ohne manuellen Eingriff. Sollten Teile der Peripherie beschädigt werden, helfen Fehlerzähler bei der Diagnose. Die Verkabelung ist robust und übersichtlich verlegt, durch den modularen Aufbau können alle Komponenten von Hard- und Software jederzeit ausgetauscht oder erweitert werden. Fähigkeiten wie die aktive Regelung der Geschwindigkeit fahrender Züge finden sich nur in sehr wenigen Anlagen, eine von diesen ist die Modellbahn des MIT (siehe [TMRC]).

Der Umbau lief reibungslos und ohne nennenswerte Schwierigkeiten, die anfängliche Planung ließ sich unverändert umsetzen. In ihrer jetzigen Form funktioniert die Anlage ohne technische Probleme, wie das im Wintersemester 2005/2006 veranstaltete Praktikum gezeigt hat. Schließlich liegt mit dieser Arbeit erstmals eine vollständige Dokumentation vor, die alle Aspekte von der Hardware bis hin zur Organisation des Fahrbetriebs abdeckt und eine gute Basis für zukünftige Projekte darstellen sollte.

Teil IV

Anhänge

Anhang A

Schaltungen

In den folgenden Abschnitten werden alle Schaltungen und Adapterkabel beschrieben, die im Rahmen dieser Arbeit entstanden sind. Neben den Schaltplänen, dem Platinenlayout und den Bauteillisten für die Bestückung werden jeweils Hinweise zu Aufbau, Betrieb und möglichen Erweiterungen gegeben.

A.1 Leistungselektronik

Die Leistungselektronik ist die wichtigste Schaltung der Modellbahn, weil sie das Bindeglied zwischen den Steuerrechnern und der Peripherie darstellt. Alle Sensoren und Aktoren werden von ihr angebunden, dabei übernimmt die Leistungselektronik noch wichtige Steueraufgaben wie die Erzeugung der Pulsweitenmodulation für die Gleise oder das Redundanzmanagement der Kontakte. Die Funktion des Systems ist in Kapitel 3 ausführlich beschrieben.

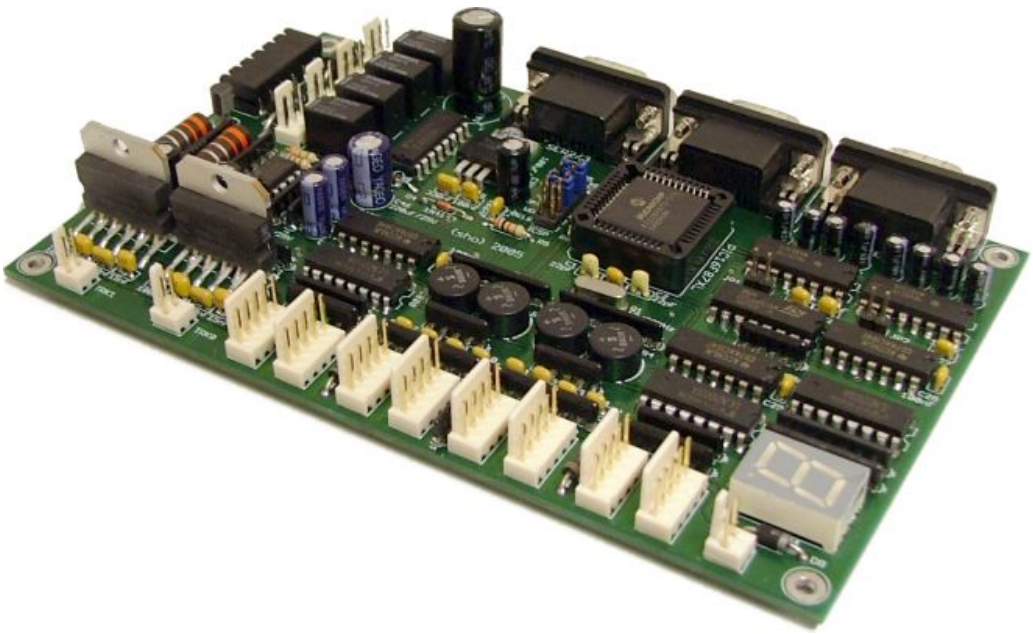


Abbildung A.1: Platine mit der Leistungselektronik

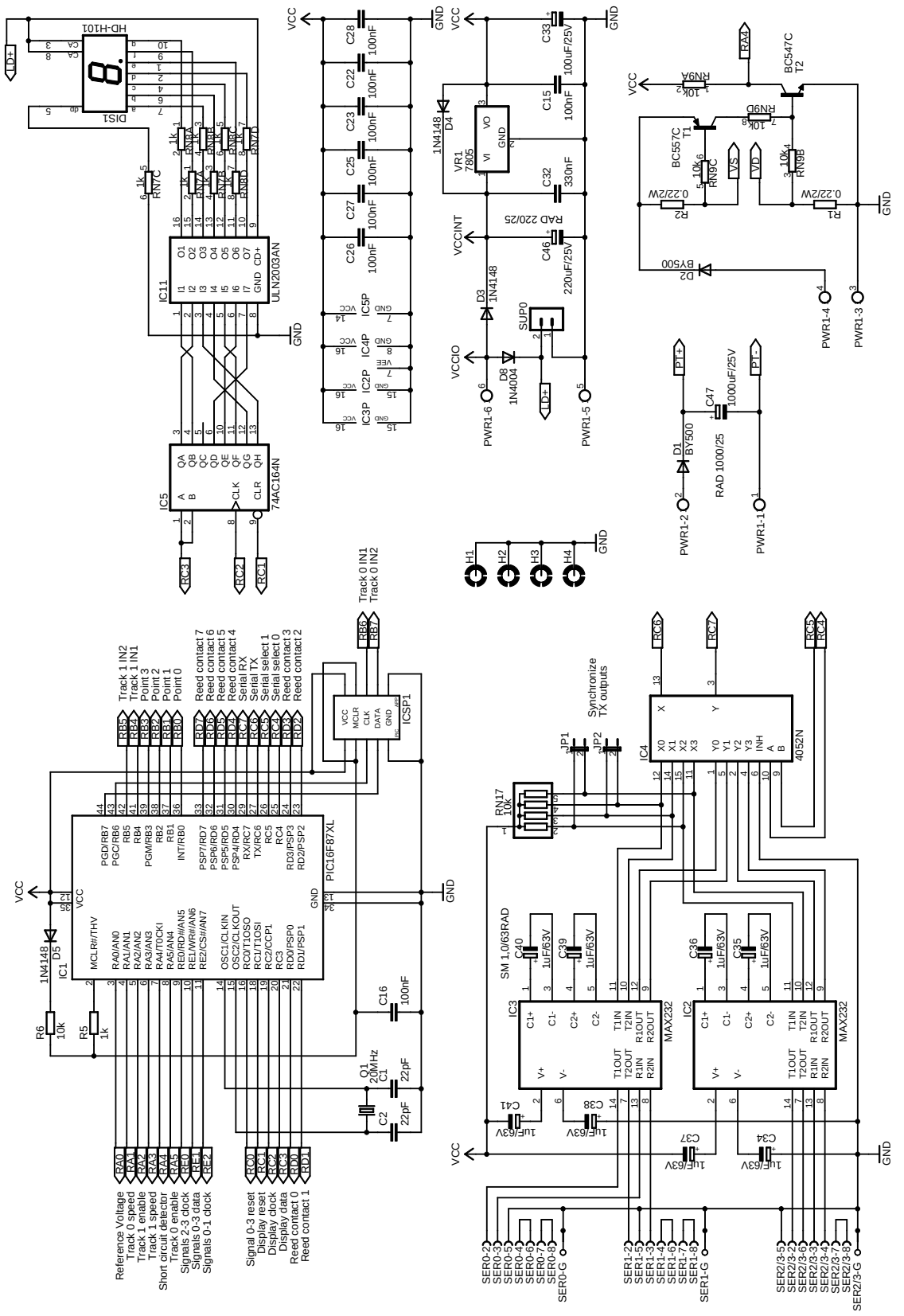


Abbildung A.2: Schaltung der Leistungselektronik (1 von 2)

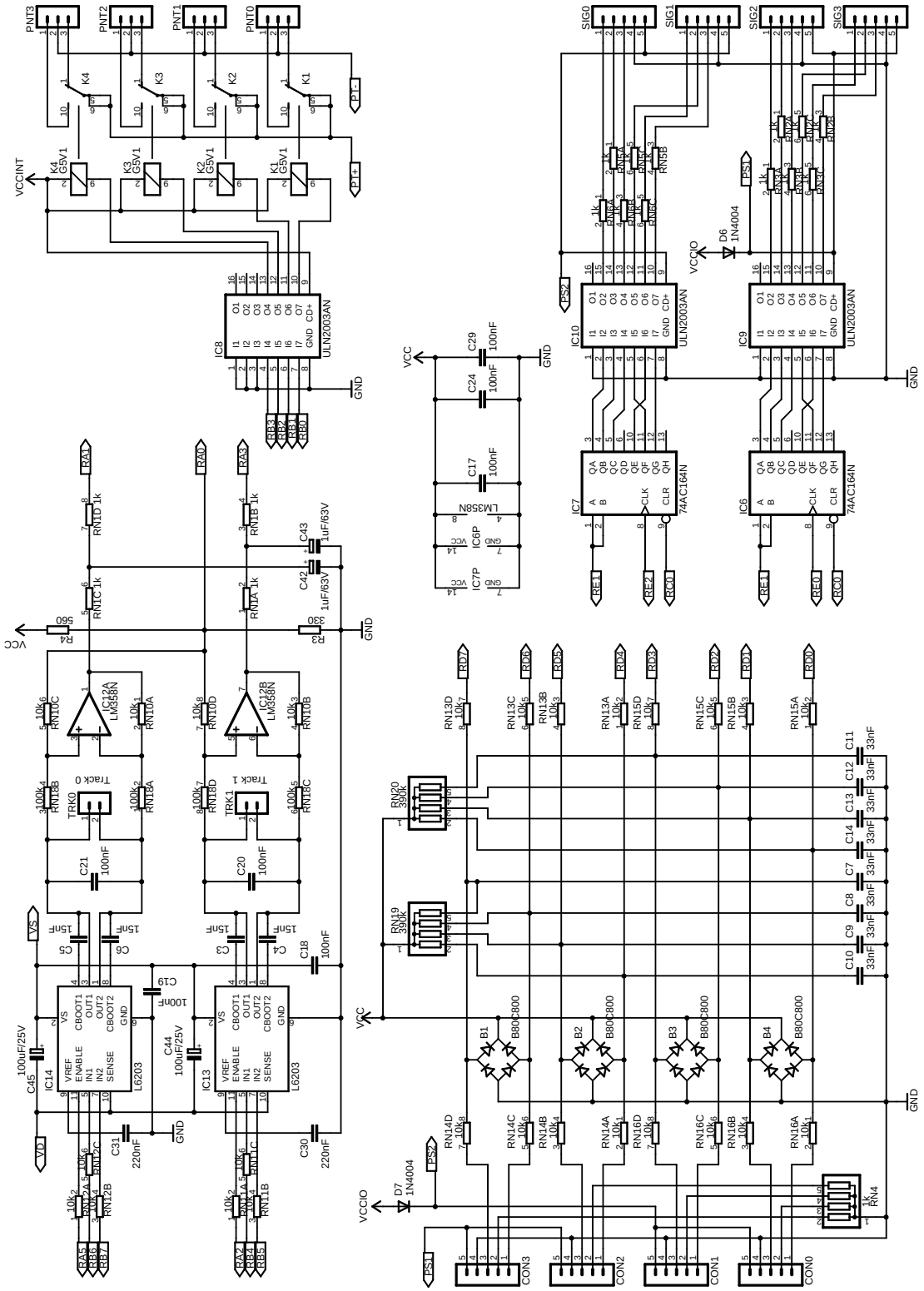


Abbildung A.3: Schaltung der Leistungselektronik (2 von 2)

Das Platinenlayout der Schaltung besteht aus zwei Layern, von denen der untere die meisten Signalleitungen und die Kurzstreckenverbindungen trägt. Die Oberseite wird überwiegend als Massefläche benutzt, daneben verlaufen hier die Stromversorgung und Langstreckenleitungen.

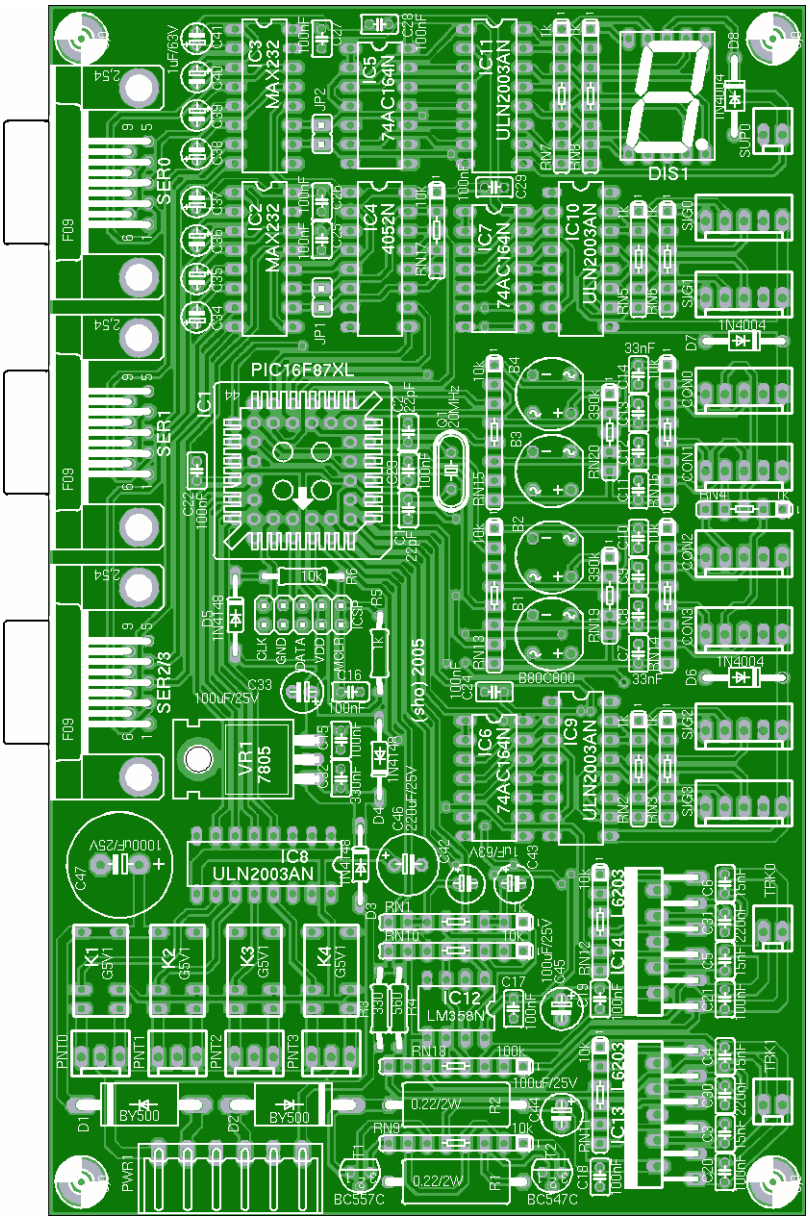


Abbildung A.4: Bestückungsplan der Leistungselektronik

Für die Bestückung einer Platine werden die in Tabelle A.1 aufgeführten Bauteile benötigt, dabei handelt es sich fast ausschließlich um Standardteile, die problemlos zu beschaffen sind. Nach den Lötarbeiten muß die Firmware des Mikrocontrollers installiert werden. Diese Aufgabe übernimmt das in [Höhrmann] beschriebene Programmiergerät über die ICSP-Schnittstelle auf der Platine, dabei wird der Codespeicher zu rund 80 Prozent gefüllt und das EEPROM mit initialisiert. Vor der Inbetriebnehme müssen in dem ICSP-Stecker noch mindestens die Pinpaare CLK und DATA mit Jumpern überbrückt werden, damit beide Motortreiber mit dem Mikrocontroller verbunden sind und normal funktionieren.

Die ersten Funktionstests können mit einem einzelnen 12 Volt-Netzteil durchgeführt werden, das an alle drei Eingänge angeschlossen ist. Später sollten stattdessen drei Netzteile benutzt werden, falls möglich mit galvanischer Trennung.

	Halbleiter
4x	Gleichrichter B80C800, rund
2x	Diode BY500
3x	Diode 1N4148
3x	Diode 1N4004
1x	Microchip PIC 16F871, PLCC-44
2x	Pegelwandler MAX232, DIL
1x	Analog Switch 4052, DIL
3x	Schieberegister 74AC164, DIL
4x	Treiber ULN2003A, DIL
1x	Operationsverstärker LM358, DIL
2x	Motortreiber L6203
1x	Spannungregler 7805, TO-220
1x	PNP-Transistor BC557C, TO-92
1x	NPN-Transistor BC547C, TO-92
1x	7-Segment-Anzeige, CA, grün

	Steckverbinder
3x	Stecker PSS 2,54 mm, 2 Pins
4x	Stecker PSS 2,54 mm, 3 Pins
8x	Stecker PSS 2,54 mm, 5 Pins
1x	Buchse RIA6-382
3x	9-pol. Sub-D Buchse

	Einzelwiderstände
2x	0,22 Ω, 2 Watt
1x	330 Ω, 1/8 Watt
1x	560 Ω, 1/8 Watt
1x	1 kΩ, 1/8 Watt
1x	10 kΩ, 1/8 Watt

	Widerstandsnetzwerke
1x	1 kΩ, 5 Pins, 4 Widerstände
4x	1 kΩ, 6 Pins, 3 Widerstände
3x	1 kΩ, 8 Pins, 4 Widerstände
1x	10 kΩ, 5 Pins, 4 Widerstände
2x	10 kΩ, 6 Pins, 3 Widerstände
6x	10 kΩ, 8 Pins, 4 Widerstände
1x	100 kΩ, 8 Pins, 4 Widerstände
2x	390 kΩ, 5 Pins, 4 Widerstände

	Kondensatoren
2x	22 pF, Keramik
2x	15 nF, Vielschicht
8x	33 nF, Vielschicht
15x	100 nF, Vielschicht
2x	220 nF, Vielschicht
1x	330 nF, Vielschicht
10x	1 µF / 63 Volt, Elektrolyt
3x	100 µF / 25 Volt, Elektrolyt
1x	220 µF / 25 Volt, Elektrolyt
1x	1000 µF / 25 Volt, Elektrolyt

	Verschiedenes
7x	Jumper
2x	Stiftleiste 1x2 Pins
1x	Stiftleiste 5x2 Pins
1x	Quarz 20 MHz
4x	Relais Omron G5V1, 12 Volt
1x	PLCC-44 Sockel
1x	Platine 160x100 mm, zweiseitig

Tabelle A.1: Komponentenliste der Leistungselektronik

Das Programm nodediags kann alle Baugruppen der Platine auf ihre Funktion hin überprüfen, dabei dient die in A.3 vorgestellte Diagnoseplatine als Anzeige und als Eingabemöglichkeit für die Sensoren. Grundlegende Fehler lassen sich so schnell ausfindig machen.

A.2 Anschlußbelegungen

Die Leistungselektronik wird über eine Reihe von Steckern mit den Netzteilen und der Peripherie verbunden. Im Folgenden werden die Pinbelegungen der Stecker spezifiziert und die elektrischen Eigenschaften der Anschlüsse zusammengefaßt.

A.2.1 Stromversorgung (PWR)

Diese sechspolige Buchse dient zum Anschluß von drei Netzteilen an die Leistungselektronik, die jeweils eine Spannung von 12 bis 14 Volt liefern. Eines der Netzteile speist die Weichentreiber, das zweite die Gleistreiber und das dritte übernimmt alle übrigen Funktionen wie die Signale, die Kontakte und die Logik auf der Platine. Die Netzteile der Anlage sind jeweils auf 10 Ampere Maximalstrom ausgelegt und bieten damit eine ausreichende Leistungsreserve.

Pin	Belegung	Typ	Standardfarbe
1 (links)	Masse der Weichentreiber	GND	gelb/grün
2	Versorgung der Weichentreiber	12 V	braun
3	Masse Fahrstrom (verbunden mit 5)	GND	blau
4	Fahrstrom für Gleistreiber	12 V	gelb
5	Masse des Logikteils	GND	schwarz
6 (rechts)	Versorgung des Logikteils	12 V	rot

Tabelle A.2: Pinbelegung des Stromanschlusses

Der Stromverbrauch hängt fast ausschließlich von der Peripherie ab, die Platine selber benötigt maximal 170 Milliampere. In der Anlage werden zweiadrige Leitungen in den Farben rot/schwarz mit einem Querschnitt von 2,5 mm^2 benutzt, um die Leistungselektroniken mit den Netzteilen zu verbinden. Diese Kabel können auch größere Ströme mit niedrigen Verlusten übertragen, von ihnen führen dann farbige Stichleitungen zu den einzelnen Platinen.

A.2.2 Signalanschlüsse (SIG0 bis SIG3)

Auf einer Leistungselektronik sind Anschlüsse für bis zu vier Signale vorhanden. Jeder dieser Stecker enthält eine Stromversorgung und drei Leitungen, die über Open-Collector-Ausgänge mit einer Impedanz von einem Kiloohm geschaltet werden können.

Pin	Belegung	Typ	Standardfarbe
1 (links)	Rote Signalleuchte	OC	braun
2	Gelbe Signalleuchte	OC	gelb
3	Grüne Signalleuchte	OC	grün
4	Masse des Signals	GND	-
5 (rechts)	Versorgungsspannung des Signals	12 V	weiß

Tabelle A.3: Pinbelegung eines Signalanschlusses

Die Leuchtdioden der Signale werden mit der Versorgungsspannung und den zu ihren Farbe gehörenden Pins verbunden. Externe Widerstände sind nicht erforderlich, nicht benötigte Pins dürfen offen gelassen werden. Jeweils vier Stecker teilen sich in der Spannungsversorgung eine Schutzdiode, daher darf die Peripherie an SIG0, SIG1, CON0 und CON1 beziehungsweise SIG2, SIG3, CON2, CON3 jeweils maximal ein Ampere verbrauchen. Der Strom wird von dem Netzteil geliefert, das den Logikteil versorgt.

An den Signalanschluß kann eine beliebige Schaltung angeschlossen werden, welche die drei Signalfarben als digitalen Eingangswert interpretiert und entsprechend darauf reagiert. Dabei

muß nur beachtet werden, daß die Ansteuerung der Signale durch ein Schieberegister erfolgt. Bei einer Aktualisierung rotieren die Bits 20 Mikrosekunden lang durch die Leitungen, bis jedes seinen Zielort erreicht hat. Die Schaltung muß daher Tiefpässe gefolgt von Schmitt-Triggern an ihrem Eingang einsetzen, um dieses hochfrequente Rauschen zu unterdrücken.

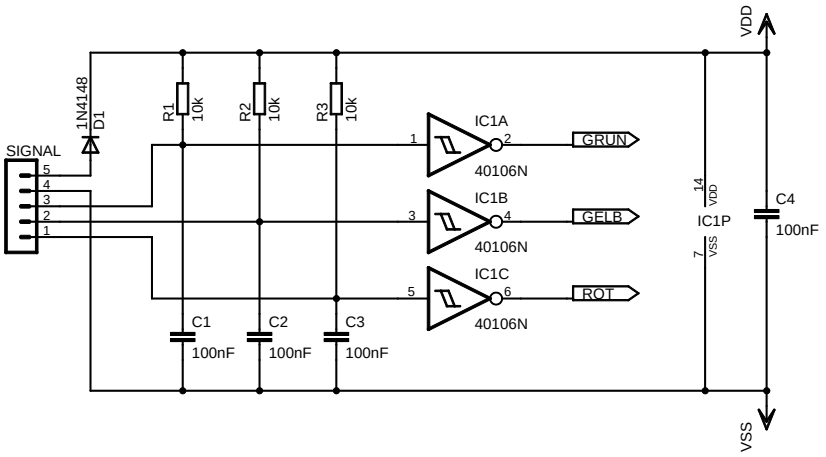


Abbildung A.5: Aktor an der Signalschnittstelle

Die Schaltung in Abbildung A.5 kann eine Grundlage für so eine Eingangsstufe sein. Der CMOS-Schmitt-Trigger funktioniert mit Spannungen bis 15 Volt und liefert an seinen Ausgängen High, wenn die entsprechende Leuchtdiode von der Leistungselektronik aktiviert wird. Die Spannung kann am Eingang auf 5 Volt reduziert werden, wenn die Elektronik dies erfordert.

A.2.3 Kontakteingänge (CON0 bis CON3)

Die Stecker der Kontaktanschlüsse enthalten ebenfalls eine Versorgungsspannung von 12 Volt. Dazu kommen zwei Eingänge, die intern über Pullup-Widerstände von 390 Kiloohm mit 5 Volt verbunden sind und von den Kontakten auf Masse gezogen werden müssen. Der fünfte Pin ist über einen Widerstand von einem Kiloohm fest mit Masse verbunden.

Pin	Belegung	Typ	Standardfarbe
1 (links)	Erster Reedkontakt in Fahrtrichtung	Pullup 5 V	braun
2	Pulldown gegen Masse	Pulldown	weiß
3	Zweiter Reedkontakt in Fahrtrichtung	Pullup 5 V	grün
4	Masse des Kontaktes	GND	-
5 (rechts)	Versorgungsspannung des Kontaktes	12 V	-

Tabelle A.4: Pinbelegung eines Kontaktanschlusses

Der Aufbau des Steckers macht es möglich, einen normalen Kontakt mit drei benachbarten Adern anzuschließen. Außerdem kommt es zu keinen Schäden an der Leistungselektronik oder der Peripherie, wenn Signale oder Kontakte mit dem falschen Anschluß verbunden werden. Die maximale Stromaufnahme aus der Versorgungsspannung ist in den beiden Gruppen SIG0, SIG1, CON0, CON1 sowie SIG2, SIG3, CON2, CON3 auf jeweils ein Ampere begrenzt.

Wenn ein Sensor an diese Schnittstelle angeschlossen wird, muß er das Verhalten eines Kontaktes mit seinen zwei Reedkontakten imitieren. Dafür sind einige einfache Regeln zu beachten.

- Die beiden Eingänge dürfen nur entweder gleichzeitig oder im Abstand von mindestens 10 Millisekunden auslösen. Dies ist nötig, damit die Leistungselektronik eine eventuell vorhandene Reihenfolge sicher erkennen kann.
- Jeder Eingang muß mindestens 21 Millisekunden geschlossen bleiben, um sicher erkannt zu werden. Anschließend sollten die Eingänge relativ schnell wieder hochohmig geschaltet werden, damit sie wieder aufnahmebereit für neue Ereignisse werden.
- Nach dem Öffnen des letzten Kontaktes müssen mindestens 0,65 Sekunden vergehen, bis die Firmware das nächste Ereignis korrekt erkennen kann.

Das Interface auf Sensorseite besteht aus Open-Collector-Treibern, welche die Kontaktleitungen auf Masse ziehen können. Dafür eignet sich neben diskreten Transistoren ebenfalls ein integrierter Baustein mit mehreren Kanälen wie der ULN2003.

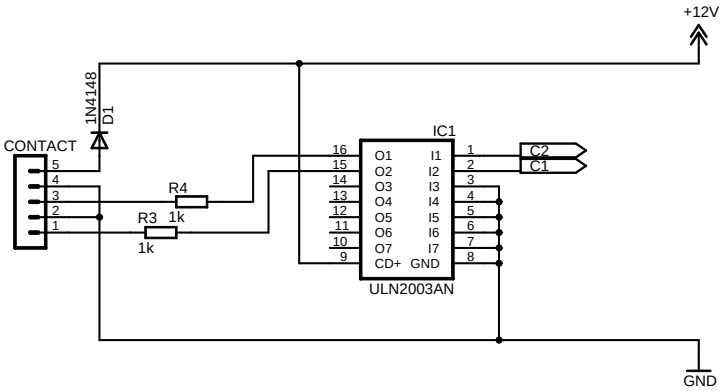


Abbildung A.6: Sensor an der Kontaktschnittstelle

Die Ansteuerung der Lichtschranken im Bahnübergang ist ein Beispiel für einen Sensor nach diesem Schema. Sie ist im Abschnitt A.4 dokumentiert.

A.2.4 Fahrstrom (TRK0 bis TRK1)

Die Ausgänge der beiden Gleistreiber sind jeweils über einen zweipoligen Stecker erreichbar. Die H-Brücke der Treiber kann jeden Pin einzeln mit Masse oder mit 12 Volt verbinden sowie hochohmig schalten. Die Ausgänge sind dabei gegen Kurzschlüsse und zu hohe Ströme gesichert, aus beiden zusammen dürfen maximal 2,7 Ampere entnommen werden.

Pin	Belegung	Typ	Farbe
1 (links)	Fahrstrom Leitung 1	H-Brücke	braun
2 (rechts)	Fahrstrom Leitung 2	H-Brücke	weiß

Tabelle A.5: Pinbelegung der Fahrstromanschlüsse

Wird ein Ausgang auf Vorwärtsfahrt geschaltet, ist Pin 1 des Steckers mit Masse und Pin 2 mit 12 Volt verbunden, bei Rückwärtsfahrt ist es entgegengesetzt. Zum Bremsen werden beide Pins mit Masse verbunden, bei abgeschaltetem Treiber oder während der Pulsweitenmodulation sind die Ausgänge hochohmig. Der Fahrstrom wird aus einem eigenen Netzteil gespeist, seine Masse ist aber mit der Masse des Digitalteiles verbunden.

Die Leistungselektronik kann die Spannungsdifferenz zwischen den Leitungen eines Anschlusses messen, wenn beide ein Potential im Bereich von 0 bis 24 Volt relativ zur Fahrstrommasse aufweisen. Dies wird zur Messung der Geschwindigkeit fahrender Züge benutzt. Zusätzlich sendet der Motortreiber in regelmäßigen Abständen kurze Impulse über die Leitung, um angeschlossene Verbraucher zu erkennen, die jeweils eine Millisekunde dauern.

A.2.5 Weichen (PNT0 bis PNT3)

Die Ausgänge für Weichen, Lampen und ähnliche Peripherie enthalten eine Spannungsquelle, die über einen Umschalter mit den drei Pins des Steckers verbunden werden kann. Die Masse liegt fest an der mittleren Leitung, die Versorgungsspannung wird im Ruhezustand mit Pin 3 und bei Aktivierung mit Pin 1 verbunden. Der jeweils andere Pin ist hochohmig geschaltet.

Pin	Belegung	Typ	Standardfarbe
1 (links)	Strom für Weichenantrieb, rot	12 V/Offen	braun
2	Masse der Weichentreiber, schwarz	GND	weiß
3 (rechts)	Strom für Weichenantrieb, grün	12 V/Offen	grün

Tabelle A.6: Pinbelegung der Weichenanschlüsse

Weichenantriebe werden nach dem Farbschema aus Tabelle A.6 angeschlossen, Lampen und ähnliche Peripherie gehören an Pin 1 und 2, damit sie im Ruhezustand ausgeschaltet sind. Der Strom für die Ausgänge stammt aus einem eigenen Netzteil, die Leitungen sind von dem Rest der Schaltung galvanisch getrennt. Pro Ausgang kann ein Dauerstrom von einem Ampere entnommen werden. Zum Schutz der Relais müssen induktive Lasten wie die Weichenantriebe extern und möglichst dicht an der Peripherie mit Freilaufdioden oder ähnlichem versehen werden.

A.2.6 Testanschluß (SUP0)

An einem kleinen Stecker am Platinenrand kann eine Spannung von 12 Volt für beliebige Zwecke entnommen werden, beispielsweise um externe Schaltungen zu versorgen oder Gleise mit einem konstanten Strom zu versorgen (die Polarität ist auf Vorwärtsfahrt eingestellt).

Pin	Belegung	Typ	Standardfarbe
1 (links)	Masse	GND	braun
2 (rechts)	Versorgungsspannung	12 V	weiß

Tabelle A.7: Pinbelegung des Testanschlusses

Der Eingang kann allerings in keiner Form geschaltet werden. Er ist mit einem Dauerstrom bis 0,9 Ampere belastbar und bezieht diesen Strom aus dem Netzteil, für den Logikteil.

A.2.7 Serielle Schnittstellen (SER0 bis SER3)

Die seriellen Schnittstellen der Leistungselektronik entsprechen dem RS232-Standard. Auf der Platine sind Buchsen installiert, die Verbindung zum Rechner wird über Verlängerungskabel hergestellt, die nur die Leitungen RXD, TXD und GND enthalten müssen. Die Buchsen SER0 und SER1 können direkt benutzt werden, SER2/3 enthält aus Platzgründen zwei Schnittstellen, wobei DSR und DTR die Pins RXD und TXD der zweiten Schnittstelle tragen. An dieser Buchse kann ein Rechner angeschlossen werden, wenn das Verbindungkabel DTR nicht benutzt oder der Rechner diese Leitung ständig auf dem Ruhepegel hält. Anderenfalls muß der in A.8 beschriebene Adapter dazwischen geschaltet werden.

A.3 Diagnoseplatine

Die Diagnoseplatine ist zum Testen der Leistungselektronik entwickelt worden. Sie kann die verschiedenen Aktoren und Sensoren simulieren, Leuchtdioden zeigen den Status der Aktoren an und Taster simulieren die Kontakte der Modellbahn. Die Platine wird über eine Reihe von Kabeln und Steckern mit den einzelnen Anschlüssen der Leistungselektronik verbunden.

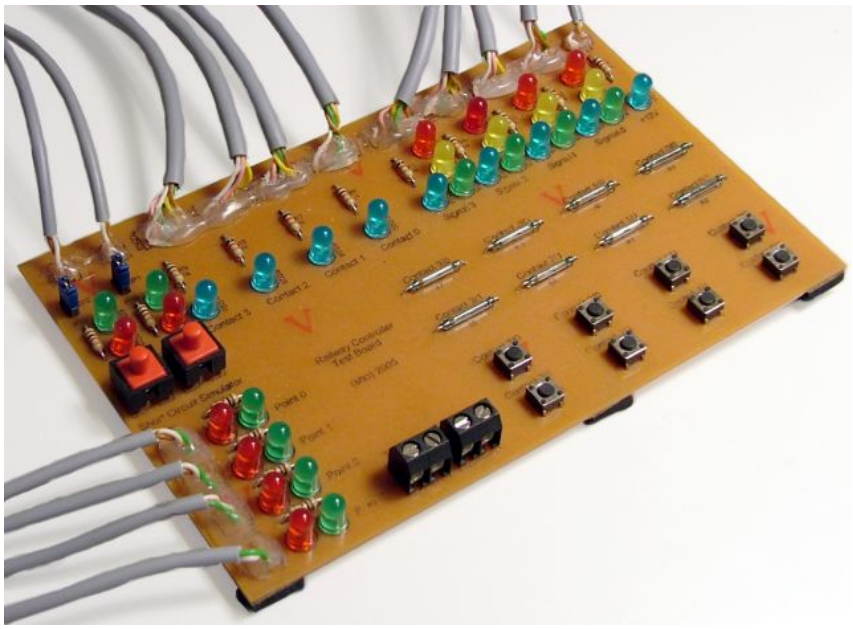


Abbildung A.7: Diagnoseplatine mit Anschlußkabeln

In vielen Baugruppen zeigt eine blaue Leuchtdiode eine funktionierende Stromversorgung an, die anderen Farben sind für die verschiedenen Funktionen der Peripherie reserviert.

- Jedes simulierte Signal besteht aus vier Leuchtdioden, einer blauen für die Stromversorgung und je einer roten, gelben und grünen für die entsprechenden Signalfarben.
- Auch bei den Kontakten zeigt eine blaue LED eine intakte Stromquelle an. Auf der Platine sind Paare von Reedkontakten angebracht, die manuell mit einem Magneten ausgelöst werden können. Parallelgeschaltete Taster erlauben eine Betätigung ohne Magnet.

- Für jede Weiche existieren je eine rote und eine grüne LED. Die grüne zeigt an, daß der Antrieb im Ruhezustand ist, bei rot würde der Zug abbiegen. Lampen, Schranken und ähnliches wären ebenfalls eingeschaltet, wenn die rote LED leuchtet.
- Jeder Gleisanschluß wird zu einer Klemme durchgeleitet, an der ein externes Schienenstück angeschlossen werden kann. Zusätzlich zeigen zwei Leuchtdioden an, ob der Treiber auf Vorwärtsfahrt (grün) oder Rückwärtsfahrt (rot) geschaltet ist. Mit einem roten Taster wird der Anschluß kurzgeschlossen, ein Jumper kann die gesamte Peripherie abtrennen, um den Gleisbesetztmelder zu testen.
- Eine blaue LED am Platinenrand zeigt an, daß der Testausgang Strom liefert.

Die Platine sollte intuitiv zu benutzen sein. Der Schaltplan in Abbildung A.8 zeigt exemplarisch für die verschiedenen Funktionen, wie diese realisiert sind.

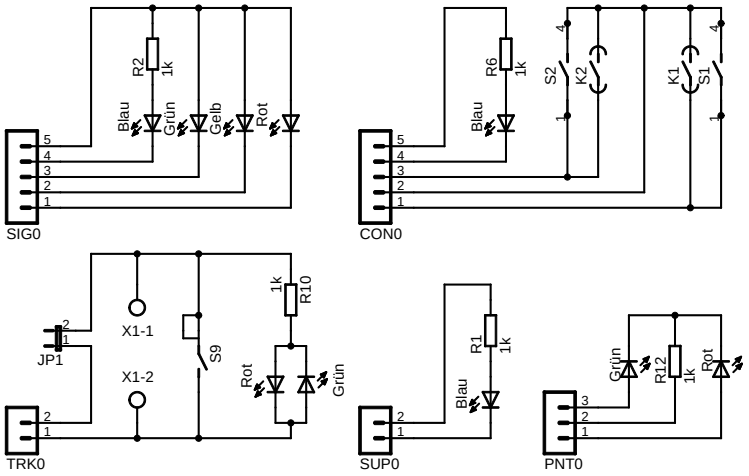


Abbildung A.8: Baugruppen in der Diagnoseplatine

Für den Aufbau einer Platine werden die in Tabelle A.8 aufgeführten Teile benötigt.

	Schalter
8x	Reedkontakt
8x	Miniaturtaster
2x	Taster

	Steckverbinder
3x	Buchse PSK 2,54 mm, 2 Pins
4x	Buchse PSK 2,54 mm, 3 Pins
8x	Buchse PSK 2,54 mm, 5 Pins

	Einzelwiderstände
15x	1 kΩ, 1/8 Watt

	Halbleiter
9x	LED 5 mm, blau
10x	LED 5 mm, rot
4x	LED 5 mm, gelb
10x	LED 5 mm, grün

	Verschiedenes
2x	Stiftleiste 1x2 Pins
2x	Jumper
2x	Anschlußklemme Wago 237-02P
1x	Platine 160x100 mm, einseitig

Tabelle A.8: Komponentenliste der Diagnoseplatine

Die Schaltung kann auf einer Platine im Europaformat aufgebaut werden. Die Anschlüsse zur Leistungselektronik erfolgen über Kabel, die an die Platine gelötet werden und an deren Ende sich die passenden Buchsen befinden.

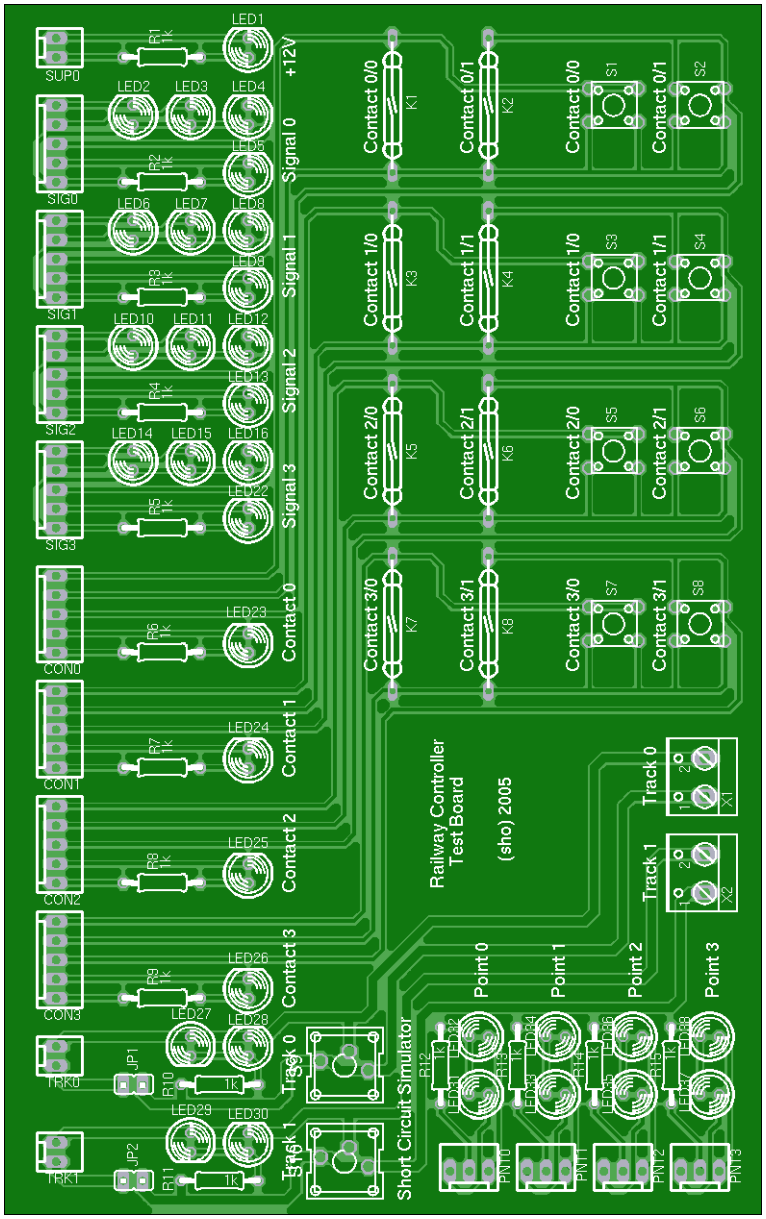


Abbildung A.9: Bestückungsplan der Diagnoseplatine

A.4 Schrankensensoren

Jede Schranke des Bahnüberganges ist mit einer Lichtschranke versehen, die Informationen über deren aktuellen Zustand liefern kann. Wenn der Baum sich absenkt, taucht ein kleiner Wimpel an seinem Ende in die Lichtschranke ein und unterbricht sie. Das System kann auf diese Weise feststellen, ob eine Schranke sich wie geplant geschlossen oder geöffnet hat. Die Ansteuerung der Lichtschranken wird an eine Kontaktschnittstelle der Leistungselektronik angeschlossen. Die

Schaltung muß sich daher wie ein Kontakt verhalten und die beiden Ereignisse „Schranke schließt sich“ und „Schranke öffnet sich“ in die Sensormeldungen „vorwärts“ und „rückwärts“ übersetzen. Diese Aufgabe übernimmt ein Mikrocontroller vom Typ PIC 12F629.

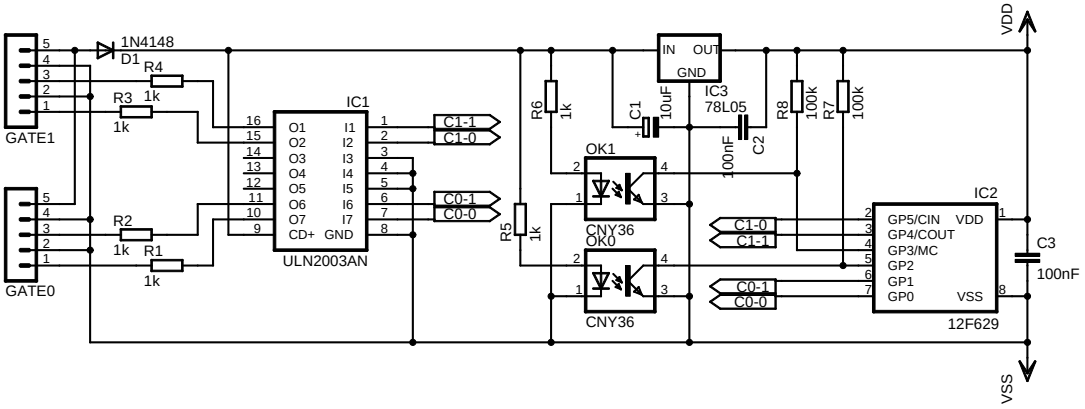


Abbildung A.10: Ansteuerung der Schrankensensoren

Die Schaltung enthält Interfaces für zwei Lichtschranken vom Typ CNY36. Ihren Strom bezieht sie von der Leistungselektronik, die Spannung wird von einem 78L05 für den Mikrocontroller auf 5 Volt stabilisiert. Dieser kann mit Hilfe von zwei Pullup-Widerständen die Lichtschranken abfragen und über einen Treiberbaustein die Kontaktmeldungen erzeugen. Die Firmware muß nur die Kontaktleitungen nach dem in A.2 beschriebenen Schema ansteuern.

- Der Mikrocontroller fragt in einer Endlosschleife beide Lichtschranken ab.
- Wenn Schranke 0 seit dem letzten Durchlauf ihren Status geändert hat, wird jetzt der erste Teil der Kontaktmeldung erzeugt (ein Kontakt offen, der andere geschlossen). Das gleiche Schema wiederholt sich für Schranke 1.
- Anschließend wartet das Programm 21 Millisekunden.
- Wenn bei einer Schranke zuvor die erste Hälfte der Kontaktmeldung ausgegeben wurde, folgt jetzt die zweite Hälfte (voriger Kontakt offen, anderer geschlossen).
- Nach einer Pause von weiteren 21 Millisekunden startet die Schleife von vorn.

Die Platine selbst ist schnell aufgebaut und wird über ein Kabel mit Buchsen an beiden Enden mit der Leistungselektronik verbunden. Die benötigten Bauteile sind in Tabelle A.9 aufgeführt.

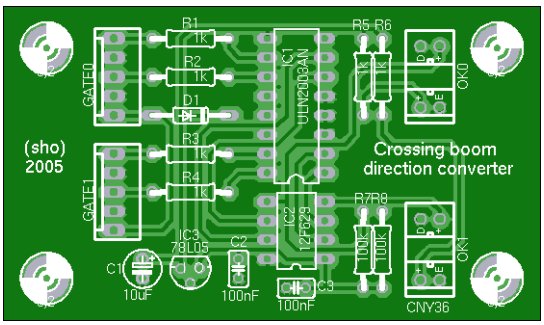


Abbildung A.11: Bestückungsplan der Schrankensensoren

Die Firmware muß bei dieser Schaltung vor dem Einbau des Mikrocontrollers installiert werden, da die Platine keine Programmierschnittstelle enthält. Spätere Updates sind nur möglich, wenn der PIC in einem IC-Sockel betrieben wird.

	Widerstände
6x	1 kΩ, 1/8 Watt
2x	100 kΩ, 1/8 Watt

	Kondensatoren
2x	100 nF, Vielschicht
1x	10 µF / 25 Volt, Elektrolyt

	Halbleiter
1x	Diode 1N4148
1x	Spannungsregler 78L05
1x	Treiber ULN2003, DIL
1x	Mikrocontroller PIC 12F629
2x	Gabellichtschranke CNY36

	Steckverbinder
2x	Stecker PSS 2,54 mm, 5 Pins

	Verschiedenes
	Kabel, Abstandhalter, Schrauben
1x	Platine, einseitig

Tabelle A.9: Komponentenliste der Schrankensensoren

Es wäre alternativ möglich gewesen, je eine Lichtschranken direkt an die parallelgeschalteten Kontakteingänge eines Anschlusses anzuschließen und auf den Mikrocontroller zu verzichten. Der so entstehende Sensor könnte aber nur das Schließen und nicht das Öffnen einer Schranke registrieren, daher ist der kompliziertere Weg in diesem Fall nötig.

A.5 Glockensteuerung

Am Bahnübergang ist neben den Schranken mit ihren Sensoren noch eine Glocke installiert, die für eine realistische Geräuschkulisse sorgt. Sie wird mit einem Weichenanschluß verbunden und schlägt mit einer einstellbaren Frequenz an, sobald sie eingeschaltet wird.

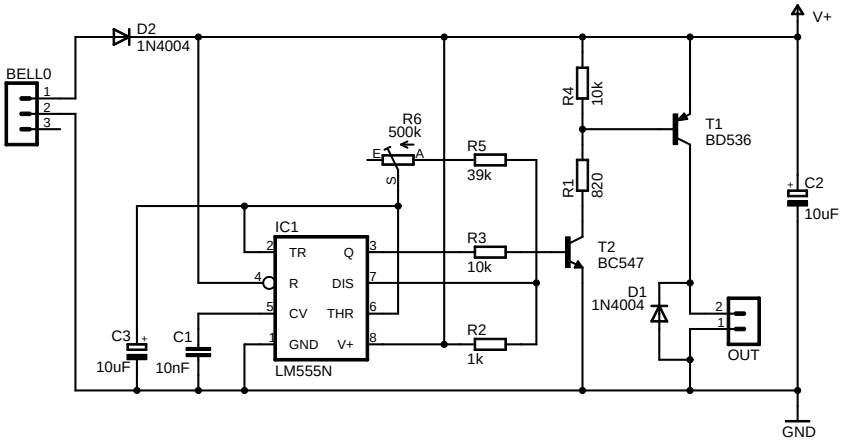


Abbildung A.12: Ansteuerung der Glocke des Bahnübergangs

Die Ansteuerung übernimmt eine Schaltung auf Basis des Timer-ICs LM555. Dieser arbeitet als Rechteckgenerator und schaltet die Glocke über zwei Treibertransistoren, die Frequenz kann dabei mit Hilfe eines Potentiometers in einem weiten Bereich justiert werden. Die Platine ist

schnell aufgebaut und mit Abstandhaltern unter der Bahn festgeschraubt. Die dafür benötigten Teile sind in Tabelle A.10 enthalten, dazu kommen noch die Kabel für den Anschluß.

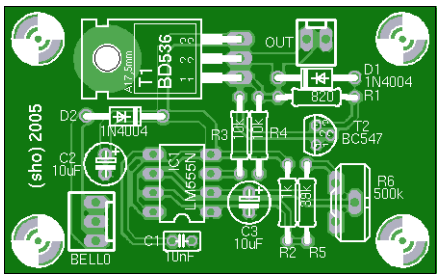


Abbildung A.13: Bestückungsplan der Glockensteuerung

	Widerstände
1x	820 Ω, 1/8 Watt
1x	1 kΩ, 1/8 Watt
2x	10 kΩ, 1/8 Watt
1x	39 kΩ, 1/8 Watt

	Halbleiter
2x	Diode 1N4004
1x	NPN-Transistor BC547, TO-92
1x	PNP-Transistor BD536, TO-220
1x	Timer-IC LM555, DIL

	Steckverbinder
1x	Stecker PSS 2,54 mm, 2 Pins
1x	Stecker PSS 2,54 mm, 5 Pins

	Kondensatoren
1x	10 nF, Vielschicht
2x	10 µF, 25 Volt, Elektrolyt

	Verschiedenes
1x	Potentiometer, 500 kΩ, stehend
1x	Platine, einseitig

Tabelle A.10: Komponentenliste der Glockensteuerung

A.6 Entstörfilter für Weichenantriebe

Jeder Weichenantrieb erzeugt beim Schalten einen starken elektromagnetischen Störimpuls, der in der Nähe liegende Elektronik beeinträchtigen kann. Daher ist es wichtig, Störungen möglichst dicht an dem Antrieb durch entsprechende Filter abzuschwächen. Im einfachsten Fall können dafür Freilaufdioden und Kondensatoren parallel zu den Spulen geschaltet werden.

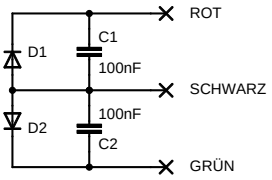


Abbildung A.14: Schaltung des einfachen Weichenfilters

Tests haben gezeigt, daß diese Konstruktion zur Unterdrückung der typischerweise auftretenden Impulse ausreicht. Sie wird in der gesamten Anlage benutzt und ist mit Lüsterklemmen und Aderendhülsen realisiert.

Dieser einfache Filter kann so erweitert werden, daß er einen eigenen Energiepuffer enthält. Auf diese Weise fließen die Stromstöße beim Schalten nicht über die Leistungselektronik, außerdem kann der Dauerstrom begrenzt werden, um einen Antrieb mit defekter Endabschaltung vor einer Überhitzung zu schützen. Das Resultat ist in Abbildung A.15 dargestellt.

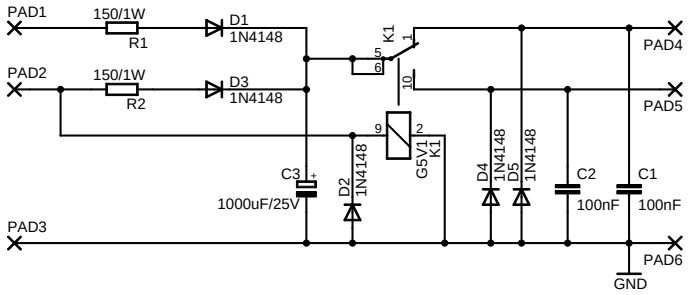


Abbildung A.15: Schaltung des verbesserten Weichenfilters

Der Filter wird zwischen Leistungselektronik (links) und Weichenantrieb (rechts) geschaltet. Über einen der beiden Vorwiderstände von 150 Ohm lädt sich der Elektrolytkondensator ständig auf und speichert damit Energie für einen Schaltvorgang. Das Relais schaltet synchron zu dem Weichentreiber der Leistungselektronik und verbindet dabei den Kondensator mit einer der Antriebsspulen. Die restlichen Bauteile bilden den bekannten Filter. Solange der Kondensator voll geladen ist, funktioniert diese Platine genau wie der Weichentreiber der Leistungselektronik. Sollte die Endabschaltung jedoch nicht funktionieren, halten die Vorwiderstände den fließenden Strom auf einem unschädlichen Niveau.

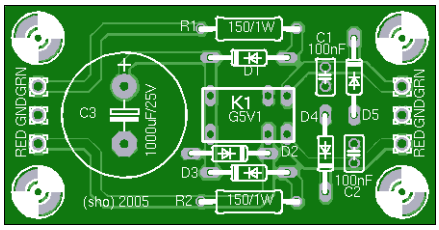


Abbildung A.16: Bestückungsplan des Weichenfilters

	Kondensatoren
2x	100 nF, Vielschicht
1x	1000 µF / 25 Volt, Elektrolyt

	Halbleiter
5x	Diode 1N4148

	Widerstände
2x	150 Ω, 1 Watt

	Verschiedenes
1x	Relais Omron G5V1, 12 Volt
1x	Platine, einseitig

Tabelle A.11: Komponentenliste des Weichenfilters

Die Firmware muß dem Kondensator nach einem Schaltvorgang genügend Zeit lassen, daß dieser sich wieder aufladen kann. Der Mindestabstand hierfür beträgt 0,2 Sekunden, genau dieser Wert ist in der aktuellen Konfiguration enthalten, der Filter kann also ohne Probleme benutzt werden.

A.7 TTP-Portextender

Für die Steuerung der Modellbahn über TTP stehen acht Powernodes mit jeweils einer serielle Schnittstelle zur Verfügung. Damit diese alle 24 Leistungselektroniken erreichen können, muß ein Multiplexer in den Knoten installiert werden, der die Schnittstelle auf drei Ausgänge verteilt.

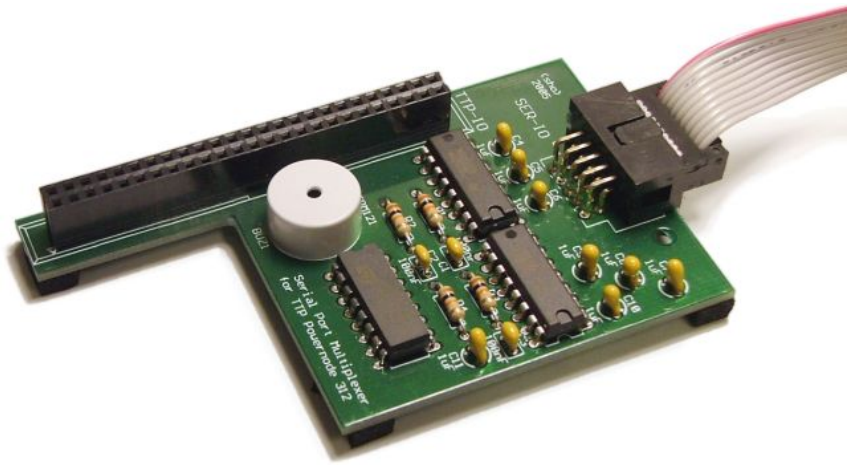


Abbildung A.17: Portextender für einen Powernode

Die Schaltung wird mit der seriellen Schnittstelle SCI1 des Powernodes verbunden. Ein MAX232 Pegelwandler setzt die Spannungen auf 5 Volt-Pegel um und leitet sie auf einen Analog Switch, der als Multiplexer arbeitet. Zwei digitale Ausgangspins des Powernodes wählen den Kanal aus (MPIO6:MPIO5), Pegelwandler setzen die Spannung wieder um und leiten sie auf einen Anschluß am Ausgang des Knotens. Der Ausgang 0 wird dabei nicht benutzt, zusätzlich befindet sich auf der Platine noch ein piezoelektrischer Schallwandler, der über einen PWM-Ausgang des Powernodes betrieben und auf beliebige Frequenzen eingestellt werden kann (MPWM0).

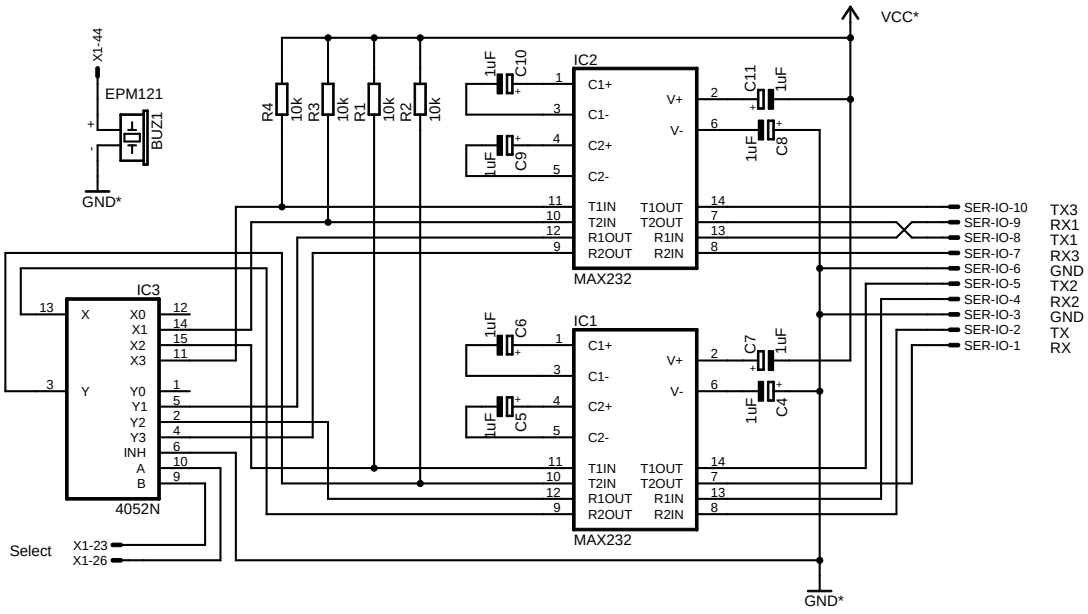


Abbildung A.18: Schaltung des Portextenders

Die Platine wird kopfüber auf den 50-poligen Anschluß SV2 des Powernodes gesteckt und bezieht aus diesem ihre Stromversorgung. An der Buchse SER-IO werden SCI1 und der neunpolige Sub-D-Stecker auf der Rückseite des Knotens angeschlossen.

Pin	Belegung	Verbunden mit	Pin	Belegung	Verbunden mit
1	RX-IN	SCI1, RXD	6	GND	Stecker Pin 5 (GND)
2	TX-IN	SCI1, TXD	7	RX3-OUT	Stecker Pin 8 (CTS)
3	GND	SCI1, GND	8	TX1-OUT	Stecker Pin 7 (RTS)
4	RX2-OUT	Stecker Pin 6 (DSR)	9	RX1-OUT	Stecker Pin 2 (RXD)
5	TX2-OUT	Stecker Pin 4 (DTR)	10	TX3-OUT	Stecker Pin 3 (TXD)

Tabelle A.12: Belegung der Buchse SER-IO im Portextender

Die im Abschnitt A.8 beschriebene Kabelpeitsche teilt die drei Schnittstellen wieder auf eigene Leitungen auf und erlaubt so den Anschluß von drei Leistungselektroniken pro Knoten, der Powernode muß nur laufend zwischen diesen umschalten. SCI1 wird von dem Knoten auch für Debugausgaben benutzt, die Leistungselektronik ignoriert aber die für sie ungültigen Daten.

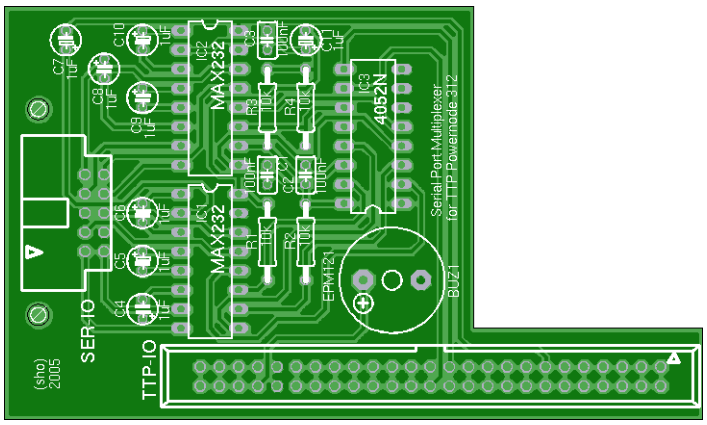


Abbildung A.19: Bestückungsplan des Portextenders

	Halbleiter
2x	Pegelwandler MAX232, DIL
1x	Analog Switch 4052, DIL

	Kondensatoren
3x	100 nF, Vielschicht
8x	1 µF / 25 Volt, sehr flach

	Steckverbinder
1x	Wannenstecker, 10-polig, gewinkelt
1x	Buchsenleiste 2,54 mm, 2x50 Pins

	Widerstände
4x	10 kΩ, 1/8 Watt

	Anschlußkabel
1x	Pfostensteckverbinder, 10-polig
1x	Flachbandkabel 10-polig
1x	Sub-D Stecker, 9-polig, Lötkelch
1x	Stiftleiste 2,54 mm, 3-polig, gewinkelt

	Verschiedenes
1x	Piezo-Summer EPM121
1x	Platine, einseitig

Tabelle A.13: Komponentenliste des Portextenders

A.8 Serielle Verbindungskabel

In der Modellbahn ist eine ganze Reihe von seriellen Verbindungen vorhanden, die in erster Linie Steuerrechner und Leistungselektroniken miteinander verbinden. Die Leitungen entsprechen dem RS232-Standard und verwenden ausschließlich neunpolige Stecker und Buchsen.

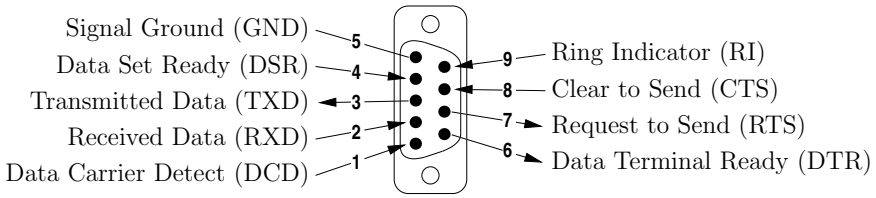


Abbildung A.20: Pinbelegung nach RS232, Stecker auf Rechnerseite

Gelegentlich enthält ein Anschluß bis zu drei Schnittstellen, in diesen Fällen werden die Paare DTR/DSR beziehungsweise RTS/CTS zweckentfremdet und entsprechen dann TXD und RXD der zusätzlichen Schnittstellen. Die Datenleitungen arbeitet mit Pegeln von etwa -10 Volt im Ruhezustand und +10 Volt bei Aktivierung, die verwendeten Pegelwandler vom Typ MAX232 erlauben einen deutlich weiteren Bereich an ihren Eingängen.

A.8.1 Anschlußkabel und Nullmodems

Für nahezu alle Verbindungen werden einfache Verlängerungen benutzt, die lediglich die Pins RXD, TXD und GND miteinander verbinden, weitere Leitungen sind nicht erforderlich. Für spezielle Aufgaben kann ein Nullmodem eingesetzt werden, wenn beispielsweise ein PC mit Hilfe des Programms simnode eine Leistungselektronik simulieren soll. Dafür werden in dem Kabel einfach RXD und TXD gekreuzt und Buchsen an beiden Enden benutzt.

A.8.2 Kabelpeitsche für Leistungselektronik

Auf der Leistungselektronik befinden sich vier serielle Schnittstellen, die aus Platzgründen über drei Sub-D Buchsen nach außen geführt sind. Die dritte Buchse enthält auf DTR und DSR die vierte Schnittstelle, daher muß eine Kabelpeitsche die Pins wieder aufteilen, um alle vier Steuerrechner anschließen zu können. Die Kabelpeitsche besteht aus einem Stecker auf Seiten der Leistungselektronik und zwei Buchsen für die Verbindung zu den Rechnern.

Abbildung A.21: Kabelpeitsche für Leistungselektronik

In ihrem Inneren werden die Leitungspaare TXD/RXD auf die eine und DTR/DSR auf die andere Buchse verzweigt. Die Massenpins aller Anschlüsse sind miteinander verbunden.

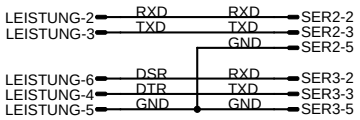


Abbildung A.22: Kabelpeitsche für Leistungselektronik

Ein serielles Kabel kann direkt auf die dritte Buchse gesteckt werden, dann darf der Rechner aber die DTR-Leitung nicht aktivieren, um nicht eine sendende vierte Schnittstelle vorzutäuschen.

A.8.3 Kabelpeitsche für TTP-Knoten

Nach genau dem selben Prinzip werden die drei Schnittstellen am Ausgang des Portextenders der TTP-Knoten wieder getrennt. Das Foto A.23 zeigt den Prototypen dieser Kabelpeitsche, die anderen sind fest in der Anlage eingebaut.

Abbildung A.23: Einfache Kabelpeitsche für TTP-Knoten

Die zweite Schnittstelle liegt auf DTR/DSR, die dritte benutzt das Leitungspaar RTS und CTS.

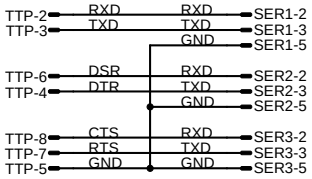


Abbildung A.24: Kabelpeitsche für TTP-Knoten

A.8.4 Listener-Adapter

Ein nützliches Werkzeug zur Fehlersuche ist der im Folgenden beschriebene Listener-Adapter. Er wird in eine existierende serielle Leitung eingefügt, an seiner dritten Buchse kann über RXD alles empfangen werden, was auf der Hauptleitung auf RXD und TXD übertragen wird. Dies funktioniert nur, wenn die Stationen an der Hauptleitung nicht gleichzeitig senden, dies ist bei der Steuerung der Anlage aber die Regel. Das Programm listener kann so die Kommunikation zwischen einem Steuerrechner und einer Leistungselektronik aufzeichnen, in eine lesbare Form decodieren und zur Auswertung speichern.

Abbildung A.25: Adapter zum Belauschen einer seriellen Leitung

Der Stecker besteht aus einem Oder-Gatter in Form einer Diode und eines Widerstandes. Das mit Listener markierte Ende wird entweder direkt oder über eine Verlängerung an den lauschenden Rechner angeschlossen.

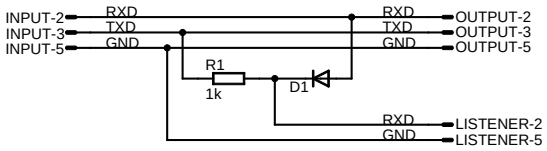


Abbildung A.26: Schaltung des Listener-Adapters

Anhang B

Steuertabellen

Dieses Kapitel enthält einige Tabellen, die für den Betrieb der Anlage, zur Umsetzung von Fahrplänen oder zur Programmierung eines Simulators gebraucht werden. Alle Daten befinden sich auch auf der beiliegenden DVD und können so einfach in Programme integriert werden.

B.1 Eigenschaften der Lokomotiven

Die Züge werden über Pulsweitenmodulation angesteuert, deren Tastverhältnis in 128 Stufen vorgegeben werden kann. Die resultierende Geschwindigkeit hängt von einer Reihe impliziter Parameter wie der Kennlinie des Motors, der Anzahl und Art der Waggons sowie der Steigung der Strecke ab. In dieser Meßreihe wurden alle Lokomotiven der Bahn nacheinander mit den Kohlewaggons bestückt und mit verschiedenen PWM-Tastverhältnissen durch den Outer Circle gefahren. Aus der Zeit für die 24,6 Meter lange Strecke ergibt sich die Geschwindigkeit. In der letzten Spalte ist der dazugehörige Sensorwert der Leistungselektronik aufgeführt.

Lokomotive 1 (Rot, Drehgestelle mit je drei Achsen)

PWM	Rundenzeit	Tastverhältnis	Geschwindigkeit	Sensorwert
127	51,4 s	99%	47,9 cm/s	40
96	57,1 s	75%	43,1 cm/s	36
64	71,1 s	50%	34,6 cm/s	29
32	143,8 s	25%	17,1 cm/s	14
25	235,6 s	20%	10,5 cm/s	9

Lokomotive 2 (Standard-Diesellok)

PWM	Rundenzeit	Tastverhältnis	Geschwindigkeit	Sensorwert
127	54,0 s	99%	45,6 cm/s	44
96	57,2 s	75%	43,1 cm/s	42
64	67,7 s	50%	36,4 cm/s	35
32	99,1 s	25%	24,9 cm/s	24
16	261,9 s	13%	9,4 cm/s	9

Lokomotive 3 (Rot/Braun)

PWM	Rundenzeit	Tastverhältnis	Geschwindigkeit	Sensorwert
127	55,7 s	99%	44,2 cm/s	33
96	63,1 s	75%	39,0 cm/s	29
64	82,1 s	50%	30,0 cm/s	22
32	174,9 s	25%	14,1 cm/s	11
25	263,4 s	20%	9,4 cm/s	7

Lokomotive 4 (Standard-Diesellok)

PWM	Rundenzeit	Tastverhältnis	Geschwindigkeit	Sensorwert
127	54,0 s	99%	45,6 cm/s	44
96	56,0 s	75%	44,0 cm/s	42
64	66,5 s	50%	37,0 cm/s	36
32	115,2 s	25%	21,4 cm/s	21

Lokomotive 5 (Standard-Diesellok)

PWM	Rundenzeit	Tastverhältnis	Geschwindigkeit	Sensorwert
127	50,4 s	99%	48,9 cm/s	43
96	55,6 s	75%	44,3 cm/s	39
64	68,2 s	50%	36,1 cm/s	32
32	127,5 s	25%	19,3 cm/s	17
20	293,1 s	16%	8,4 cm/s	7

Lokomotive 6 (Standard-Diesellok)

PWM	Rundenzeit	Tastverhältnis	Geschwindigkeit	Sensorwert
127	51,7 s	99%	47,6 cm/s	44
96	56,2 s	75%	43,8 cm/s	40
64	66,5 s	50%	37,0 cm/s	34
32	105,8 s	25%	23,3 cm/s	22
20	232,4 s	16%	10,6 cm/s	10

Lokomotive 7 (Rot/Schwarz)

PWM	Rundenzeit	Tastverhältnis	Geschwindigkeit	Sensorwert
127	68,0 s	99%	36,2 cm/s	33
96	80,7 s	75%	30,5 cm/s	28
64	109,3 s	50%	22,5 cm/s	21
32	243,0 s	25%	10,1 cm/s	9

Lokomotive 8 (Standard-Diesellok)

PWM	Rundenzeit	Tastverhältnis	Geschwindigkeit	Sensorwert
127	51,6 s	99%	47,7 cm/s	45
96	55,7 s	75%	44,2 cm/s	42
64	66,2 s	50%	37,2 cm/s	35
32	107,3 s	25%	23,0 cm/s	22
20	195,6 s	16%	12,6 cm/s	12

Lokomotive 9 (Standard-Diesellok)

PWM	Rundenzeit	Tastverhältnis	Geschwindigkeit	Sensorwert
127	51,8 s	99%	47,5 cm/s	43
96	56,3 s	75%	43,7 cm/s	40
64	66,7 s	50%	36,9 cm/s	33
32	113,5 s	25%	21,7 cm/s	20
20	236,0 s	16%	10,4 cm/s	9

Lokomotive 10 (Gelb/Grün)

PWM	Rundenzeit	Tastverhältnis	Geschwindigkeit	Sensorwert
127	69,2 s	99%	35,6 cm/s	29
96	84,4 s	75%	29,2 cm/s	24
64	130,1 s	50%	18,9 cm/s	15
48	221,3 s	38%	11,1 cm/s	9

Tabelle B.1: Eigenschaften verschiedener Lokomotiven

Wird die Lokomotive beispielsweise durch weitere Waggons belastet, sinkt die Geschwindigkeit leicht. Der Unterschied fällt aber relativ gering aus, solange die Belastung den Motor nicht an seine Grenzen bringt. In der Praxis sind hier Unterschiede um 10 Prozent festzustellen. Die Meßdaten zeigen auch, daß die Sensorwerte der Leistungselektronik grob der Geschwindigkeit in Zentimetern pro Sekunde entsprechen. Die größten Unterschiede der Lokomotiven liegen in ihren Höchstgeschwindigkeiten und der Steilheit ihrer Kennlinien. Dadurch variiert insbesondere das Verhalten bei Langsamfahrt, was den Einsatz eines Reglers sinnvoll macht.

B.2 Adressierung der Peripherie

Jedes Steuerprogramm muß die verschiedenen Sensoren und Aktoren gezielt adressieren können, um die Anlage steuern zu können. Die Peripherie kann aber nahezu beliebig angeschlossen sein, daher benötigt das Programm eine Tabelle mit den Zuordnungen. Die hier vorgestellte Tabelle faßt diese Informationen zusammen. Für sämtliche Anschlüsse der Leistungselektroniken ist angegeben, welches Bauteil dort angeschlossen ist. Zusätzlich gibt die Tabelle noch an, über welchen Anschluß der Steuerrechner die Leistungselektronik erreichbar ist.

TTP	PC104	Anschluß	Bauteil
Node 1, Port 1	Node 0, ttyS0	Platine 0, SIG0	KH_LN_0, Signal 0
Node 1, Port 1	Node 0, ttyS0	Platine 0, SIG1	(frei)
Node 1, Port 1	Node 0, ttyS0	Platine 0, SIG2	(frei)
Node 1, Port 1	Node 0, ttyS0	Platine 0, SIG3	(frei)
Node 1, Port 1	Node 0, ttyS0	Platine 0, CON0	KIO_LN_1, Kontakt 0
Node 1, Port 1	Node 0, ttyS0	Platine 0, CON1	KH_LN_0, Kontakt 0
Node 1, Port 1	Node 0, ttyS0	Platine 0, CON2	IC_LN_4, Kontakt 1
Node 1, Port 1	Node 0, ttyS0	Platine 0, CON3	OI_LN_0, Kontakt 1
Node 1, Port 1	Node 0, ttyS0	Platine 0, TRK0	OI_LN_2, Fahrstrom
Node 1, Port 1	Node 0, ttyS0	Platine 0, TRK1	IC_LN_4, Fahrstrom
Node 1, Port 1	Node 0, ttyS0	Platine 0, PNT0	(frei)
Node 1, Port 1	Node 0, ttyS0	Platine 0, PNT1	(frei)
Node 1, Port 1	Node 0, ttyS0	Platine 0, PNT2	(frei)
Node 1, Port 1	Node 0, ttyS0	Platine 0, PNT3	(frei)

TTP	PC104	Anschluß	Bauteil
Node 1, Port 2	Node 1, ttyS0	Platine 1, SIG0	KIO_LN_1, Signal 0
Node 1, Port 2	Node 1, ttyS0	Platine 1, SIG1	(frei)
Node 1, Port 2	Node 1, ttyS0	Platine 1, SIG2	(frei)
Node 1, Port 2	Node 1, ttyS0	Platine 1, SIG3	(frei)
Node 1, Port 2	Node 1, ttyS0	Platine 1, CON0	(frei)
Node 1, Port 2	Node 1, ttyS0	Platine 1, CON1	(frei)
Node 1, Port 2	Node 1, ttyS0	Platine 1, CON2	(frei)
Node 1, Port 2	Node 1, ttyS0	Platine 1, CON3	(frei)
Node 1, Port 2	Node 1, ttyS0	Platine 1, TRK0	OC_LN_1, Fahrstrom
Node 1, Port 2	Node 1, ttyS0	Platine 1, TRK1	KH_ST_6, Fahrstrom
Node 1, Port 2	Node 1, ttyS0	Platine 1, PNT0	Lampe 11
Node 1, Port 2	Node 1, ttyS0	Platine 1, PNT1	Weichenantrieb 8
Node 1, Port 2	Node 1, ttyS0	Platine 1, PNT2	Weichenantrieb 9
Node 1, Port 2	Node 1, ttyS0	Platine 1, PNT3	(frei)

TTP	PC104	Anschluß	Bauteil
Node 1, Port 3	Node 2, ttyS0	Platine 2, SIG0	KH_ST_1, Signal 1
Node 1, Port 3	Node 2, ttyS0	Platine 2, SIG1	OI_LN_2, Signal 1
Node 1, Port 3	Node 2, ttyS0	Platine 2, SIG2	IC_LN_0, Signal 1
Node 1, Port 3	Node 2, ttyS0	Platine 2, SIG3	(frei)
Node 1, Port 3	Node 2, ttyS0	Platine 2, CON0	IC_LN_0, Kontakt 1
Node 1, Port 3	Node 2, ttyS0	Platine 2, CON1	OC_LN_5, Kontakt 0
Node 1, Port 3	Node 2, ttyS0	Platine 2, CON2	OC_LN_1, Kontakt 0
Node 1, Port 3	Node 2, ttyS0	Platine 2, CON3	KH_ST_1, Kontakt 1
Node 1, Port 3	Node 2, ttyS0	Platine 2, TRK0	IC_LN_1, Fahrstrom
Node 1, Port 3	Node 2, ttyS0	Platine 2, TRK1	KH_LN_6, Fahrstrom
Node 1, Port 3	Node 2, ttyS0	Platine 2, PNT0	Weichenantrieb 6
Node 1, Port 3	Node 2, ttyS0	Platine 2, PNT1	Weichenantrieb 7
Node 1, Port 3	Node 2, ttyS0	Platine 2, PNT2	(frei)
Node 1, Port 3	Node 2, ttyS0	Platine 2, PNT3	(frei)

TTP	PC104	Anschluß	Bauteil
Node 2, Port 1	Node 3, ttyS0	Platine 3, SIG0	IC_LN_4, Signal 1
Node 2, Port 1	Node 3, ttyS0	Platine 3, SIG1	OC_LN_4, Signal 1
Node 2, Port 1	Node 3, ttyS0	Platine 3, SIG2	KH_ST_2, Signal 1
Node 2, Port 1	Node 3, ttyS0	Platine 3, SIG3	KH_ST_3, Signal 1
Node 2, Port 1	Node 3, ttyS0	Platine 3, CON0	KH_ST_2, Kontakt 1
Node 2, Port 1	Node 3, ttyS0	Platine 3, CON1	OC_LN_4, Kontakt 1
Node 2, Port 1	Node 3, ttyS0	Platine 3, CON2	IC_LN_1, Kontakt 0
Node 2, Port 1	Node 3, ttyS0	Platine 3, CON3	OI_LN_0, Kontakt 0
Node 2, Port 1	Node 3, ttyS0	Platine 3, TRK0	OC_LN_4, Fahrstrom
Node 2, Port 1	Node 3, ttyS0	Platine 3, TRK1	OI_LN_0, Fahrstrom
Node 2, Port 1	Node 3, ttyS0	Platine 3, PNT0	Lampe 9
Node 2, Port 1	Node 3, ttyS0	Platine 3, PNT1	Lampe 10
Node 2, Port 1	Node 3, ttyS0	Platine 3, PNT2	Weichenantrieb 5
Node 2, Port 1	Node 3, ttyS0	Platine 3, PNT3	(frei)

TTP	PC104	Anschluß	Bauteil
Node 2, Port 2	Node 4, ttyS0	Platine 4, SIG0	KH_ST_4, Signal 1
Node 2, Port 2	Node 4, ttyS0	Platine 4, SIG1	KH_ST_5, Signal 1
Node 2, Port 2	Node 4, ttyS0	Platine 4, SIG2	KH_LN_6, Signal 1
Node 2, Port 2	Node 4, ttyS0	Platine 4, SIG3	(frei)
Node 2, Port 2	Node 4, ttyS0	Platine 4, CON0	OI_LN_2, Kontakt 1
Node 2, Port 2	Node 4, ttyS0	Platine 4, CON1	KH_ST_3, Kontakt 1
Node 2, Port 2	Node 4, ttyS0	Platine 4, CON2	KH_ST_4, Kontakt 1
Node 2, Port 2	Node 4, ttyS0	Platine 4, CON3	KH_ST_5, Kontakt 1
Node 2, Port 2	Node 4, ttyS0	Platine 4, TRK0	KH_LN_5, Fahrstrom
Node 2, Port 2	Node 4, ttyS0	Platine 4, TRK1	KH_ST_5, Fahrstrom
Node 2, Port 2	Node 4, ttyS0	Platine 4, PNT0	Lampe 7
Node 2, Port 2	Node 4, ttyS0	Platine 4, PNT1	Lampe 8
Node 2, Port 2	Node 4, ttyS0	Platine 4, PNT2	Weichenantrieb 12
Node 2, Port 2	Node 4, ttyS0	Platine 4, PNT3	Weichenantrieb 13

TTP	PC104	Anschluß	Bauteil
Node 2, Port 3	Node 5, ttyS0	Platine 5, SIG0	KH_LN_5, Signal 1
Node 2, Port 3	Node 5, ttyS0	Platine 5, SIG1	(frei)
Node 2, Port 3	Node 5, ttyS0	Platine 5, SIG2	(frei)
Node 2, Port 3	Node 5, ttyS0	Platine 5, SIG3	(frei)
Node 2, Port 3	Node 5, ttyS0	Platine 5, CON0	KH_LN_5, Kontakt 1
Node 2, Port 3	Node 5, ttyS0	Platine 5, CON1	KH_LN_6, Kontakt 1
Node 2, Port 3	Node 5, ttyS0	Platine 5, CON2	(frei)
Node 2, Port 3	Node 5, ttyS0	Platine 5, CON3	(frei)
Node 2, Port 3	Node 5, ttyS0	Platine 5, TRK0	OC_JCT_0, Fahrstrom
Node 2, Port 3	Node 5, ttyS0	Platine 5, TRK1	IC_JCT_0, Fahrstrom
Node 2, Port 3	Node 5, ttyS0	Platine 5, PNT0	Lampe 6
Node 2, Port 3	Node 5, ttyS0	Platine 5, PNT1	Weichenantrieb 10
Node 2, Port 3	Node 5, ttyS0	Platine 5, PNT2	Weichenantrieb 11
Node 2, Port 3	Node 5, ttyS0	Platine 5, PNT3	Weichenantrieb 14

TTP	PC104	Anschluß	Bauteil
Node 3, Port 1	Node 6, ttyS0	Platine 6, SIG0	OC_LN_0, Signal 1
Node 3, Port 1	Node 6, ttyS0	Platine 6, SIG1	IO_LN_2, Signal 1
Node 3, Port 1	Node 6, ttyS0	Platine 6, SIG2	KH_LN_1, Signal 1
Node 3, Port 1	Node 6, ttyS0	Platine 6, SIG3	KH_LN_7, Signal 0
Node 3, Port 1	Node 6, ttyS0	Platine 6, CON0	IO_LN_0, Kontakt 0
Node 3, Port 1	Node 6, ttyS0	Platine 6, CON1	IO_LN_2, Kontakt 1
Node 3, Port 1	Node 6, ttyS0	Platine 6, CON2	IC_LN_5, Kontakt 0
Node 3, Port 1	Node 6, ttyS0	Platine 6, CON3	OC_LN_0, Kontakt 1
Node 3, Port 1	Node 6, ttyS0	Platine 6, TRK0	KH_ST_1, Fahrstrom
Node 3, Port 1	Node 6, ttyS0	Platine 6, TRK1	KH_ST_2, Fahrstrom
Node 3, Port 1	Node 6, ttyS0	Platine 6, PNT0	Lampe 4
Node 3, Port 1	Node 6, ttyS0	Platine 6, PNT1	Lampe 5
Node 3, Port 1	Node 6, ttyS0	Platine 6, PNT2	(frei)
Node 3, Port 1	Node 6, ttyS0	Platine 6, PNT3	(frei)

TTP	PC104	Anschluß	Bauteil
Node 3, Port 2	Node 7, ttyS0	Platine 7, SIG0	KH_ST_1, Signal 0
Node 3, Port 2	Node 7, ttyS0	Platine 7, SIG1	KH_ST_2, Signal 0
Node 3, Port 2	Node 7, ttyS0	Platine 7, SIG2	(frei)
Node 3, Port 2	Node 7, ttyS0	Platine 7, SIG3	(frei)
Node 3, Port 2	Node 7, ttyS0	Platine 7, CON0	KH_ST_1, Kontakt 0
Node 3, Port 2	Node 7, ttyS0	Platine 7, CON1	KH_ST_2, Kontakt 0
Node 3, Port 2	Node 7, ttyS0	Platine 7, CON2	KH_LN_7, Kontakt 0
Node 3, Port 2	Node 7, ttyS0	Platine 7, CON3	KH_LN_1, Kontakt 1
Node 3, Port 2	Node 7, ttyS0	Platine 7, TRK0	KH_ST_3, Fahrstrom
Node 3, Port 2	Node 7, ttyS0	Platine 7, TRK1	IC_LN_5, Fahrstrom
Node 3, Port 2	Node 7, ttyS0	Platine 7, PNT0	Lampe 2
Node 3, Port 2	Node 7, ttyS0	Platine 7, PNT1	Lampe 3
Node 3, Port 2	Node 7, ttyS0	Platine 7, PNT2	(frei)
Node 3, Port 2	Node 7, ttyS0	Platine 7, PNT3	(frei)

TTP	PC104	Anschluß	Bauteil
Node 3, Port 3	Node 8, ttyS0	Platine 8, SIG0	KH_ST_3, Signal 0
Node 3, Port 3	Node 8, ttyS0	Platine 8, SIG1	KH_ST_4, Signal 0
Node 3, Port 3	Node 8, ttyS0	Platine 8, SIG2	(frei)
Node 3, Port 3	Node 8, ttyS0	Platine 8, SIG3	(frei)
Node 3, Port 3	Node 8, ttyS0	Platine 8, CON0	KH_ST_3, Kontakt 0
Node 3, Port 3	Node 8, ttyS0	Platine 8, CON1	KH_ST_4, Kontakt 0
Node 3, Port 3	Node 8, ttyS0	Platine 8, CON2	(frei)
Node 3, Port 3	Node 8, ttyS0	Platine 8, CON3	(frei)
Node 3, Port 3	Node 8, ttyS0	Platine 8, TRK0	KH_ST_4, Fahrstrom
Node 3, Port 3	Node 8, ttyS0	Platine 8, TRK1	IO_LN_2, Fahrstrom
Node 3, Port 3	Node 8, ttyS0	Platine 8, PNT0	Weichenantrieb 3
Node 3, Port 3	Node 8, ttyS0	Platine 8, PNT1	Weichenantrieb 4
Node 3, Port 3	Node 8, ttyS0	Platine 8, PNT2	Lampe 1
Node 3, Port 3	Node 8, ttyS0	Platine 8, PNT3	(frei)

TTP	PC104	Anschluß	Bauteil
Node 4, Port 1	Node 9, ttyS0	Platine 9, SIG0	KH_LN_2, Signal 0
Node 4, Port 1	Node 9, ttyS0	Platine 9, SIG1	KH_ST_5, Signal 0
Node 4, Port 1	Node 9, ttyS0	Platine 9, SIG2	IC_LN_1, Signal 1
Node 4, Port 1	Node 9, ttyS0	Platine 9, SIG3	(frei)
Node 4, Port 1	Node 9, ttyS0	Platine 9, CON0	IC_LN_1, Kontakt 1
Node 4, Port 1	Node 9, ttyS0	Platine 9, CON1	OC_LN_4, Kontakt 0
Node 4, Port 1	Node 9, ttyS0	Platine 9, CON2	KH_LN_2, Kontakt 0
Node 4, Port 1	Node 9, ttyS0	Platine 9, CON3	KH_ST_5, Kontakt 0
Node 4, Port 1	Node 9, ttyS0	Platine 9, TRK0	KH_LN_7, Fahrstrom
Node 4, Port 1	Node 9, ttyS0	Platine 9, TRK1	KH_LN_1, Fahrstrom
Node 4, Port 1	Node 9, ttyS0	Platine 9, PNT0	Weichenantrieb 1
Node 4, Port 1	Node 9, ttyS0	Platine 9, PNT1	Weichenantrieb 2
Node 4, Port 1	Node 9, ttyS0	Platine 9, PNT2	(frei)
Node 4, Port 1	Node 9, ttyS0	Platine 9, PNT3	(frei)

TTP	PC104	Anschluß	Bauteil
Node 4, Port 2	Node 10, ttyS0	Platine 10, SIG0	(frei)
Node 4, Port 2	Node 10, ttyS0	Platine 10, SIG1	(frei)
Node 4, Port 2	Node 10, ttyS0	Platine 10, SIG2	(frei)
Node 4, Port 2	Node 10, ttyS0	Platine 10, SIG3	(frei)
Node 4, Port 2	Node 10, ttyS0	Platine 10, CON0	IC_LN_2, Kontakt 0
Node 4, Port 2	Node 10, ttyS0	Platine 10, CON1	(frei)
Node 4, Port 2	Node 10, ttyS0	Platine 10, CON2	(frei)
Node 4, Port 2	Node 10, ttyS0	Platine 10, CON3	(frei)
Node 4, Port 2	Node 10, ttyS0	Platine 10, TRK0	KH_ST_0, Fahrstrom
Node 4, Port 2	Node 10, ttyS0	Platine 10, TRK1	IO_LN_0, Fahrstrom
Node 4, Port 2	Node 10, ttyS0	Platine 10, PNT0	Weichenantrieb 0
Node 4, Port 2	Node 10, ttyS0	Platine 10, PNT1	Lampe 0
Node 4, Port 2	Node 10, ttyS0	Platine 10, PNT2	(frei)
Node 4, Port 2	Node 10, ttyS0	Platine 10, PNT3	(frei)

TTP	PC104	Anschluß	Bauteil
Node 4, Port 3	Node 11, ttyS0	Platine 11, SIG0	IO_LN_1, Signal 1
Node 4, Port 3	Node 11, ttyS0	Platine 11, SIG1	OC_LN_3, Signal 1
Node 4, Port 3	Node 11, ttyS0	Platine 11, SIG2	KIO_LN_0, Signal 1
Node 4, Port 3	Node 11, ttyS0	Platine 11, SIG3	KH_LN_8, Signal 1
Node 4, Port 3	Node 11, ttyS0	Platine 11, CON0	KIO_LN_0, Kontakt 1
Node 4, Port 3	Node 11, ttyS0	Platine 11, CON1	OC_LN_3, Kontakt 1
Node 4, Port 3	Node 11, ttyS0	Platine 11, CON2	IO_LN_2, Kontakt 0
Node 4, Port 3	Node 11, ttyS0	Platine 11, CON3	KH_LN_8, Kontakt 1
Node 4, Port 3	Node 11, ttyS0	Platine 11, TRK0	KIO_LN_0, Fahrstrom
Node 4, Port 3	Node 11, ttyS0	Platine 11, TRK1	KH_LN_8, Fahrstrom
Node 4, Port 3	Node 11, ttyS0	Platine 11, PNT0	(frei)
Node 4, Port 3	Node 11, ttyS0	Platine 11, PNT1	(frei)
Node 4, Port 3	Node 11, ttyS0	Platine 11, PNT2	(frei)
Node 4, Port 3	Node 11, ttyS0	Platine 11, PNT3	(frei)

TTP	PC104	Anschluß	Bauteil
Node 5, Port 1	Node 12, ttyS0	Platine 12, SIG0	KIO_LN_0, Signal 0
Node 5, Port 1	Node 12, ttyS0	Platine 12, SIG1	IO_LN_0, Signal 1
Node 5, Port 1	Node 12, ttyS0	Platine 12, SIG2	GATESIGNAL 0
Node 5, Port 1	Node 12, ttyS0	Platine 12, SIG3	GATESIGNAL 1
Node 5, Port 1	Node 12, ttyS0	Platine 12, CON0	OC_LN_0, Kontakt 0
Node 5, Port 1	Node 12, ttyS0	Platine 12, CON1	IC_LN_5, Kontakt 1
Node 5, Port 1	Node 12, ttyS0	Platine 12, CON2	KIO_LN_0, Kontakt 0
Node 5, Port 1	Node 12, ttyS0	Platine 12, CON3	IO_LN_1, Kontakt 1
Node 5, Port 1	Node 12, ttyS0	Platine 12, TRK0	IO_LN_1, Fahrstrom
Node 5, Port 1	Node 12, ttyS0	Platine 12, TRK1	OC_LN_0, Fahrstrom
Node 5, Port 1	Node 12, ttyS0	Platine 12, PNT0	Lampe 22
Node 5, Port 1	Node 12, ttyS0	Platine 12, PNT1	Lampe 23
Node 5, Port 1	Node 12, ttyS0	Platine 12, PNT2	Schranke 0
Node 5, Port 1	Node 12, ttyS0	Platine 12, PNT3	Schranke 1

TTP	PC104	Anschluß	Bauteil
Node 5, Port 2	Node 13, ttyS0	Platine 13, SIG0	IC_LN_5, Signal 1
Node 5, Port 2	Node 13, ttyS0	Platine 13, SIG1	(frei)
Node 5, Port 2	Node 13, ttyS0	Platine 13, SIG2	(frei)
Node 5, Port 2	Node 13, ttyS0	Platine 13, SIG3	(frei)
Node 5, Port 2	Node 13, ttyS0	Platine 13, CON0	IO_LN_1, Kontakt 0
Node 5, Port 2	Node 13, ttyS0	Platine 13, CON1	IO_LN_0, Kontakt 1
Node 5, Port 2	Node 13, ttyS0	Platine 13, CON2	Schrankensensor 0
Node 5, Port 2	Node 13, ttyS0	Platine 13, CON3	Schrankensensor 1
Node 5, Port 2	Node 13, ttyS0	Platine 13, TRK0	IC_ST_0, Fahrstrom
Node 5, Port 2	Node 13, ttyS0	Platine 13, TRK1	OC_ST_4, Fahrstrom
Node 5, Port 2	Node 13, ttyS0	Platine 13, PNT0	Weichenantrieb 16
Node 5, Port 2	Node 13, ttyS0	Platine 13, PNT1	Weichenantrieb 17
Node 5, Port 2	Node 13, ttyS0	Platine 13, PNT2	Weichenantrieb 18
Node 5, Port 2	Node 13, ttyS0	Platine 13, PNT3	Glocke 0

TTP	PC104	Anschluß	Bauteil
Node 5, Port 3	Node 14, ttyS0	Platine 14, SIG0	KH_LN_8, Signal 0
Node 5, Port 3	Node 14, ttyS0	Platine 14, SIG1	(frei)
Node 5, Port 3	Node 14, ttyS0	Platine 14, SIG2	(frei)
Node 5, Port 3	Node 14, ttyS0	Platine 14, SIG3	(frei)
Node 5, Port 3	Node 14, ttyS0	Platine 14, CON0	IC_LN_2, Kontakt 1
Node 5, Port 3	Node 14, ttyS0	Platine 14, CON1	KH_LN_8, Kontakt 0
Node 5, Port 3	Node 14, ttyS0	Platine 14, CON2	KH_LN_7, Kontakt 1
Node 5, Port 3	Node 14, ttyS0	Platine 14, CON3	(frei)
Node 5, Port 3	Node 14, ttyS0	Platine 14, TRK0	IC_LN_2, Fahrstrom
Node 5, Port 3	Node 14, ttyS0	Platine 14, TRK1	OC_LN_3, Fahrstrom
Node 5, Port 3	Node 14, ttyS0	Platine 14, PNT0	(frei)
Node 5, Port 3	Node 14, ttyS0	Platine 14, PNT1	(frei)
Node 5, Port 3	Node 14, ttyS0	Platine 14, PNT2	(frei)
Node 5, Port 3	Node 14, ttyS0	Platine 14, PNT3	(frei)

TTP	PC104	Anschluß	Bauteil
Node 6, Port 1	Node 15, ttyS0	Platine 15, SIG0	OC_ST_3, Signal 1
Node 6, Port 1	Node 15, ttyS0	Platine 15, SIG1	KH_LN_7, Signal 1
Node 6, Port 1	Node 15, ttyS0	Platine 15, SIG2	(frei)
Node 6, Port 1	Node 15, ttyS0	Platine 15, SIG3	(frei)
Node 6, Port 1	Node 15, ttyS0	Platine 15, CON0	IC_ST_1, Kontakt 0
Node 6, Port 1	Node 15, ttyS0	Platine 15, CON1	IC_ST_2, Kontakt 0
Node 6, Port 1	Node 15, ttyS0	Platine 15, CON2	IC_LN_3, Kontakt 0
Node 6, Port 1	Node 15, ttyS0	Platine 15, CON3	OC_LN_3, Kontakt 0
Node 6, Port 1	Node 15, ttyS0	Platine 15, TRK0	IC_ST_3, Fahrstrom
Node 6, Port 1	Node 15, ttyS0	Platine 15, TRK1	OC_LN_2, Fahrstrom
Node 6, Port 1	Node 15, ttyS0	Platine 15, PNT0	Weichenantrieb 19
Node 6, Port 1	Node 15, ttyS0	Platine 15, PNT1	Weichenantrieb 20
Node 6, Port 1	Node 15, ttyS0	Platine 15, PNT2	Weichenantrieb 21
Node 6, Port 1	Node 15, ttyS0	Platine 15, PNT3	Weichenantrieb 22

TTP	PC104	Anschluß	Bauteil
Node 6, Port 2	Node 16, ttyS0	Platine 16, SIG0	OC_ST_1, Signal 1
Node 6, Port 2	Node 16, ttyS0	Platine 16, SIG1	OC_ST_2, Signal 1
Node 6, Port 2	Node 16, ttyS0	Platine 16, SIG2	IC_LN_2, Signal 1
Node 6, Port 2	Node 16, ttyS0	Platine 16, SIG3	(frei)
Node 6, Port 2	Node 16, ttyS0	Platine 16, CON0	OC_ST_1, Kontakt 1
Node 6, Port 2	Node 16, ttyS0	Platine 16, CON1	OC_ST_2, Kontakt 1
Node 6, Port 2	Node 16, ttyS0	Platine 16, CON2	OC_ST_3, Kontakt 1
Node 6, Port 2	Node 16, ttyS0	Platine 16, CON3	IC_ST_3, Kontakt 0
Node 6, Port 2	Node 16, ttyS0	Platine 16, TRK0	IC_ST_1, Fahrstrom
Node 6, Port 2	Node 16, ttyS0	Platine 16, TRK1	IC_ST_2, Fahrstrom
Node 6, Port 2	Node 16, ttyS0	Platine 16, PNT0	Lampe 19
Node 6, Port 2	Node 16, ttyS0	Platine 16, PNT1	Lampe 20
Node 6, Port 2	Node 16, ttyS0	Platine 16, PNT2	Lampe 21
Node 6, Port 2	Node 16, ttyS0	Platine 16, PNT3	(frei)

TTP	PC104	Anschluß	Bauteil
Node 6, Port 3	Node 17, ttyS0	Platine 17, SIG0	KH_LN_2, Signal 1
Node 6, Port 3	Node 17, ttyS0	Platine 17, SIG1	OC_LN_2, Signal 1
Node 6, Port 3	Node 17, ttyS0	Platine 17, SIG2	(frei)
Node 6, Port 3	Node 17, ttyS0	Platine 17, SIG3	(frei)
Node 6, Port 3	Node 17, ttyS0	Platine 17, CON0	KH_LN_2, Kontakt 1
Node 6, Port 3	Node 17, ttyS0	Platine 17, CON1	OC_LN_2, Kontakt 1
Node 6, Port 3	Node 17, ttyS0	Platine 17, CON2	(frei)
Node 6, Port 3	Node 17, ttyS0	Platine 17, CON3	(frei)
Node 6, Port 3	Node 17, ttyS0	Platine 17, TRK0	IC_LN_3, Fahrstrom
Node 6, Port 3	Node 17, ttyS0	Platine 17, TRK1	KH_LN_2, Fahrstrom
Node 6, Port 3	Node 17, ttyS0	Platine 17, PNT0	Weichenantrieb 15
Node 6, Port 3	Node 17, ttyS0	Platine 17, PNT1	Lampe 17
Node 6, Port 3	Node 17, ttyS0	Platine 17, PNT2	Lampe 18
Node 6, Port 3	Node 17, ttyS0	Platine 17, PNT3	(frei)

TTP	PC104	Anschluß	Bauteil
Node 7, Port 1	Node 18, ttyS0	Platine 18, SIG0	KH_LN_3, Signal 0
Node 7, Port 1	Node 18, ttyS0	Platine 18, SIG1	KH_LN_4, Signal 0
Node 7, Port 1	Node 18, ttyS0	Platine 18, SIG2	(frei)
Node 7, Port 1	Node 18, ttyS0	Platine 18, SIG3	(frei)
Node 7, Port 1	Node 18, ttyS0	Platine 18, CON0	KH_LN_6, Kontakt 0
Node 7, Port 1	Node 18, ttyS0	Platine 18, CON1	KH_LN_4, Kontakt 0
Node 7, Port 1	Node 18, ttyS0	Platine 18, CON2	KH_LN_3, Kontakt 0
Node 7, Port 1	Node 18, ttyS0	Platine 18, CON3	KH_LN_1, Kontakt 0
Node 7, Port 1	Node 18, ttyS0	Platine 18, TRK0	KH_LN_3, Fahrstrom
Node 7, Port 1	Node 18, ttyS0	Platine 18, TRK1	KH_LN_4, Fahrstrom
Node 7, Port 1	Node 18, ttyS0	Platine 18, PNT0	Lampe 15
Node 7, Port 1	Node 18, ttyS0	Platine 18, PNT1	Lampe 16
Node 7, Port 1	Node 18, ttyS0	Platine 18, PNT2	(frei)
Node 7, Port 1	Node 18, ttyS0	Platine 18, PNT3	(frei)

TTP	PC104	Anschluß	Bauteil
Node 7, Port 2	Node 19, ttyS0	Platine 19, SIG0	(frei)
Node 7, Port 2	Node 19, ttyS0	Platine 19, SIG1	(frei)
Node 7, Port 2	Node 19, ttyS0	Platine 19, SIG2	(frei)
Node 7, Port 2	Node 19, ttyS0	Platine 19, SIG3	(frei)
Node 7, Port 2	Node 19, ttyS0	Platine 19, CON0	IC_ST_2, Kontakt 1
Node 7, Port 2	Node 19, ttyS0	Platine 19, CON1	IC_ST_3, Kontakt 1
Node 7, Port 2	Node 19, ttyS0	Platine 19, CON2	IC_ST_1, Kontakt 1
Node 7, Port 2	Node 19, ttyS0	Platine 19, CON3	OI_LN_1, Kontakt 1
Node 7, Port 2	Node 19, ttyS0	Platine 19, TRK0	OC_ST_3, Fahrstrom
Node 7, Port 2	Node 19, ttyS0	Platine 19, TRK1	KH_LN_0, Fahrstrom
Node 7, Port 2	Node 19, ttyS0	Platine 19, PNT0	Lampe 14
Node 7, Port 2	Node 19, ttyS0	Platine 19, PNT1	(frei)
Node 7, Port 2	Node 19, ttyS0	Platine 19, PNT2	(frei)
Node 7, Port 2	Node 19, ttyS0	Platine 19, PNT3	(frei)

TTP	PC104	Anschluß	Bauteil
Node 7, Port 3	Node 20, ttyS0	Platine 20, SIG0	KH_LN_1, Signal 0
Node 7, Port 3	Node 20, ttyS0	Platine 20, SIG1	IC_ST_2, Signal 1
Node 7, Port 3	Node 20, ttyS0	Platine 20, SIG2	IC_ST_3, Signal 1
Node 7, Port 3	Node 20, ttyS0	Platine 20, SIG3	(frei)
Node 7, Port 3	Node 20, ttyS0	Platine 20, CON0	OC_ST_2, Kontakt 0
Node 7, Port 3	Node 20, ttyS0	Platine 20, CON1	KH_LN_4, Kontakt 1
Node 7, Port 3	Node 20, ttyS0	Platine 20, CON2	OC_ST_1, Kontakt 0
Node 7, Port 3	Node 20, ttyS0	Platine 20, CON3	KH_LN_0, Kontakt 1
Node 7, Port 3	Node 20, ttyS0	Platine 20, TRK0	OC_ST_1, Fahrstrom
Node 7, Port 3	Node 20, ttyS0	Platine 20, TRK1	OC_ST_2, Fahrstrom
Node 7, Port 3	Node 20, ttyS0	Platine 20, PNT0	Weichenantrieb 23
Node 7, Port 3	Node 20, ttyS0	Platine 20, PNT1	Weichenantrieb 26
Node 7, Port 3	Node 20, ttyS0	Platine 20, PNT2	Lampe 13
Node 7, Port 3	Node 20, ttyS0	Platine 20, PNT3	(frei)

TTP	PC104	Anschluß	Bauteil
Node 8, Port 1	Node 21, ttyS0	Platine 21, SIG0	OI_LN_1, Signal 1
Node 8, Port 1	Node 21, ttyS0	Platine 21, SIG1	KH_LN_6, Signal 0
Node 8, Port 1	Node 21, ttyS0	Platine 21, SIG2	IC_ST_1, Signal 1
Node 8, Port 1	Node 21, ttyS0	Platine 21, SIG3	KH_LN_0, Signal 1
Node 8, Port 1	Node 21, ttyS0	Platine 21, CON0	IC_LN_3, Kontakt 1
Node 8, Port 1	Node 21, ttyS0	Platine 21, CON1	OC_ST_3, Kontakt 0
Node 8, Port 1	Node 21, ttyS0	Platine 21, CON2	OI_LN_2, Kontakt 0
Node 8, Port 1	Node 21, ttyS0	Platine 21, CON3	OC_LN_2, Kontakt 0
Node 8, Port 1	Node 21, ttyS0	Platine 21, TRK0	IC_LN_0, Fahrstrom
Node 8, Port 1	Node 21, ttyS0	Platine 21, TRK1	OI_LN_1, Fahrstrom
Node 8, Port 1	Node 21, ttyS0	Platine 21, PNT0	Weichenantrieb 24
Node 8, Port 1	Node 21, ttyS0	Platine 21, PNT1	Weichenantrieb 25
Node 8, Port 1	Node 21, ttyS0	Platine 21, PNT2	Lampe 12
Node 8, Port 1	Node 21, ttyS0	Platine 21, PNT3	(frei)

TTP	PC104	Anschluß	Bauteil
Node 8, Port 2	Node 22, ttyS0	Platine 22, SIG0	OC_LN_1, Signal 1
Node 8, Port 2	Node 22, ttyS0	Platine 22, SIG1	KH_LN_3, Signal 1
Node 8, Port 2	Node 22, ttyS0	Platine 22, SIG2	IC_LN_3, Signal 1
Node 8, Port 2	Node 22, ttyS0	Platine 22, SIG3	KH_LN_4, Signal 1
Node 8, Port 2	Node 22, ttyS0	Platine 22, CON0	KH_LN_3, Kontakt 1
Node 8, Port 2	Node 22, ttyS0	Platine 22, CON1	IC_LN_4, Kontakt 0
Node 8, Port 2	Node 22, ttyS0	Platine 22, CON2	OC_LN_1, Kontakt 1
Node 8, Port 2	Node 22, ttyS0	Platine 22, CON3	IC_LN_0, Kontakt 0
Node 8, Port 2	Node 22, ttyS0	Platine 22, TRK0	OC_ST_0, Fahrstrom
Node 8, Port 2	Node 22, ttyS0	Platine 22, TRK1	IC_ST_4, Fahrstrom
Node 8, Port 2	Node 22, ttyS0	Platine 22, PNT0	Weichenantrieb 27
Node 8, Port 2	Node 22, ttyS0	Platine 22, PNT1	Weichenantrieb 28
Node 8, Port 2	Node 22, ttyS0	Platine 22, PNT2	Weichenantrieb 29
Node 8, Port 2	Node 22, ttyS0	Platine 22, PNT3	(frei)

TTP	PC104	Anschluß	Bauteil
Node 8, Port 3	Node 23, ttyS0	Platine 23, SIG0	KH_LN_5, Signal 0
Node 8, Port 3	Node 23, ttyS0	Platine 23, SIG1	OI_LN_0, Signal 1
Node 8, Port 3	Node 23, ttyS0	Platine 23, SIG2	KIO_LN_1, Signal 1
Node 8, Port 3	Node 23, ttyS0	Platine 23, SIG3	OC_LN_5, Signal 1
Node 8, Port 3	Node 23, ttyS0	Platine 23, CON0	OI_LN_1, Kontakt 0
Node 8, Port 3	Node 23, ttyS0	Platine 23, CON1	KH_LN_5, Kontakt 0
Node 8, Port 3	Node 23, ttyS0	Platine 23, CON2	KIO_LN_1, Kontakt 1
Node 8, Port 3	Node 23, ttyS0	Platine 23, CON3	OC_LN_5, Kontakt 1
Node 8, Port 3	Node 23, ttyS0	Platine 23, TRK0	KIO_LN_1, Fahrstrom
Node 8, Port 3	Node 23, ttyS0	Platine 23, TRK1	OC_LN_5, Fahrstrom
Node 8, Port 3	Node 23, ttyS0	Platine 23, PNT0	(frei)
Node 8, Port 3	Node 23, ttyS0	Platine 23, PNT1	(frei)
Node 8, Port 3	Node 23, ttyS0	Platine 23, PNT2	(frei)
Node 8, Port 3	Node 23, ttyS0	Platine 23, PNT3	(frei)

B.3 Freie Anschlüsse

Der vorige Abschnitt hat gezeigt, daß einige Anschlüsse der Leistungselektroniken noch nicht belegt sind und somit für Erweiterungen genutzt werden können. Tabelle B.3 faßt die Anzahlen der freien Ports zusammen.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
SIG	3	3	1	0	1	3	0	2	2	1	4	0
CON	0	4	0	0	0	2	0	0	2	0	3	0
TRK	0	0	0	0	0	0	0	0	0	0	0	0
PNT	4	1	2	1	0	0	2	2	1	2	2	4

	12	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**
SIG	0	3	3	2	1	2	2	4	1	0	0	0
CON	0	0	1	0	0	2	0	0	0	0	0	0
TRK	0	0	0	0	0	0	0	0	0	0	0	0
PNT	0	0	4	0	1	1	2	3	1	1	1	4

Tabelle B.3: Freie Anschlüsse der Leistungselektroniken

B.4 Blocklängen

Für viele Aufgaben ist es wichtig, die genaue Länge der einzelnen Blöcke zu kennen. Programme können so abschätzen, wie lange ein Zug für eine gegebene Strecke braucht, ob er rechtzeitig den Bahnhof erreichen wird und so weiter. Noch wichtiger sind die Daten für einen Simulator, der realistische Zugbewegungen simulieren soll, oder für eine Visualisierung, welche die Position des Zuges auch innerhalb von Blöcken darstellen können will.

Die gesamte Anlage ist aus einer kleinen Anzahl von Grundelementen aufgebaut, daher setzt sich jede Blocklänge aus ganzzahligen Vielfachen dieser Elemente zusammen.

Geradenstück	**Länge**	**Distanzstück**	**Länge**
G1	23,00 cm	D4	0,40 cm
G$\frac{1}{2}$	11,50 cm	D8	0,80 cm
G$\frac{1}{4}$	5,75 cm		

Bogenstück	**Länge**	**Kurvenradius**	**Vollkreis**
R3 30°	21,97 cm	41,96 cm	12 Stücke
R4 30°	25,20 cm	48,12 cm	12 Stücke
R5 30°	28,42 cm	54,28 cm	12 Stücke
R10 15°	23,25 cm	88,80 cm	24 Stücke

Weiche	**Gerade**	**Abzweigen**
W15	23,00 cm	23,25 cm
BW5/6	34,87 cm	28,42 cm
EKW15	23,00 cm	23,25 cm

Tabelle B.4: Kenngrößen der verwendeten Grundelemente

Bei der Berechnung müssen alle Blöcke gesondert behandelt werden, die Weichen enthalten. Durch sie gibt es je nach Weichenstellung mehrere unterschiedlich lange Wege, die folgende Tabelle listet in solchen Fällen alle möglichen Varianten auf. Zusätzlich ist immer die ungefähre Position der Kontakte angegeben, gemessen vom Blockanfang und in Fahrtrichtung.

Outer Circle

Block	Durchquerung	Länge	Kontakt 0	Kontakt 1
OC_ST_0	OC_LN_5 - OC_ST_1	183,3 cm	(keiner)	(keiner)
OC_ST_0	OC_LN_5 - OC_ST_2	183,5 cm	(keiner)	(keiner)
OC_ST_0	OC_LN_5 - OC_ST_3	160,0 cm	(keiner)	(keiner)
OC_ST_0	KIO_LN_1 - OC_ST_1	183,5 cm	(keiner)	(keiner)
OC_ST_0	KIO_LN_1 - OC_ST_2	183,8 cm	(keiner)	(keiner)
OC_ST_0	KIO_LN_1 - OC_ST_3	160,3 cm	(keiner)	(keiner)
OC_ST_0	IC_ST_4 - KIO_LN_1	23,0 cm	(keiner)	(keiner)
OC_ST_1	Komplett	232,1 cm	12,9 cm	197,1 cm
OC_ST_2	Komplett	230,0 cm	11,0 cm	195,7 cm
OC_ST_3	Komplett	276,0 cm	12,0 cm	242,0 cm
OC_ST_4	OC_ST_1 - OC_LN_0	177,5 cm	(keiner)	(keiner)
OC_ST_4	OC_ST_2 - OC_LN_0	177,8 cm	(keiner)	(keiner)
OC_ST_4	OC_ST_3 - OC_LN_0	154,3 cm	(keiner)	(keiner)
OC_ST_4	OC_ST_1 - KIO_LN_0	177,8 cm	(keiner)	(keiner)
OC_ST_4	OC_ST_2 - KIO_LN_0	178,0 cm	(keiner)	(keiner)
OC_ST_4	OC_ST_3 - KIO_LN_0	154,5 cm	(keiner)	(keiner)
OC_ST_4	KIO_LN_0 - IC_ST_0	23,0 cm	(keiner)	(keiner)
OC_LN_0	Komplett	339,1 cm	11,9 cm	303,4 cm
OC_JCT_0	OC_LN_0 - OC_LN_1	69,0 cm	(keiner)	(keiner)
OC_JCT_0	IO_LN_2 - OC_LN_1	69,3 cm	(keiner)	(keiner)
OC_JCT_0	OC_LN_0 - OI_LN_0	69,3 cm	(keiner)	(keiner)
OC_JCT_0	IO_LN_2 - OI_LN_0	69,5 cm	(keiner)	(keiner)
OC_LN_1	Komplett	314,9 cm	12,1 cm	279,7 cm
OC_LN_2	Komplett	298,3 cm	15,0 cm	264,3 cm
OC_LN_3	Komplett	320,0 cm	14,7 cm	293,9 cm
OC_LN_4	Komplett	268,2 cm	12,0 cm	233,0 cm
OC_LN_5	Komplett	263,5 cm	11,9 cm	228,5 cm

Inner Circle

Block	Durchquerung	Länge	Kontakt 0	Kontakt 1
IC_ST_0	IC_LN_5 - IC_ST_1	121,6 cm	(keiner)	(keiner)
IC_ST_0	IC_LN_5 - IC_ST_2	145,1 cm	(keiner)	(keiner)
IC_ST_0	IC_LN_5 - IC_ST_3	144,9 cm	(keiner)	(keiner)
IC_ST_0	KIO_LN_0 - IC_ST_1	121,9 cm	(keiner)	(keiner)
IC_ST_0	KIO_LN_0 - IC_ST_2	145,4 cm	(keiner)	(keiner)
IC_ST_0	KIO_LN_0 - IC_ST_3	145,1 cm	(keiner)	(keiner)
IC_ST_1	Komplett	276,0 cm	11,2 cm	241,2 cm
IC_ST_2	Komplett	230,0 cm	11,9 cm	196,0 cm
IC_ST_3	Komplett	232,1 cm	4,5 cm	196,3 cm

IC_ST_4	IC_ST_1 - IC_LN_0	127,4 cm	(keiner)	(keiner)
IC_ST_4	IC_ST_2 - IC_LN_0	150,9 cm	(keiner)	(keiner)
IC_ST_4	IC_ST_3 - IC_LN_0	150,6 cm	(keiner)	(keiner)
IC_ST_4	IC_ST_1 - KIO_LN_1	127,6 cm	(keiner)	(keiner)
IC_ST_4	IC_ST_2 - KIO_LN_1	151,1 cm	(keiner)	(keiner)
IC_ST_4	IC_ST_3 - KIO_LN_1	150,9 cm	(keiner)	(keiner)
IC_LN_0	Komplett	276,9 cm	11,9 cm	242,0 cm
IC_LN_1	Komplett	268,2 cm	12,4 cm	234,1 cm
IC_LN_2	Komplett	300,7 cm	15,4 cm	267,1 cm
IC_LN_3	Komplett	291,9 cm	17,0 cm	252,7 cm
IC_LN_4	Komplett	302,1 cm	11,3 cm	267,2 cm
IC_JCT_0	IC_LN_4 - IC_LN_5	69,0 cm	(keiner)	(keiner)
IC_JCT_0	OI_LN_2 - IC_LN_5	69,3 cm	(keiner)	(keiner)
IC_JCT_0	IC_LN_4 - IO_LN_0	69,3 cm	(keiner)	(keiner)
IC_JCT_0	OI_LN_2 - IO_LN_0	69,5 cm	(keiner)	(keiner)
IC_LN_5	Komplett	352,4 cm	12,0 cm	312,5 cm

Kicking Horse Pass

Block	Durchquerung	Länge	Kontakt 0	Kontakt 1
KH_ST_0	KH_LN_8 - KH_ST_1	128,7 cm	(keiner)	(keiner)
KH_ST_0	KH_LN_8 - KH_ST_2	129,0 cm	(keiner)	(keiner)
KH_ST_0	KH_LN_8 - KH_ST_3	105,2 cm	(keiner)	(keiner)
KH_ST_0	KH_LN_8 - KH_ST_4	81,4 cm	(keiner)	(keiner)
KH_ST_0	KH_LN_8 - KH_ST_5	57,9 cm	(keiner)	(keiner)
KH_ST_0	KIO_LN_0 - KH_ST_1	122,3 cm	(keiner)	(keiner)
KH_ST_0	KIO_LN_0 - KH_ST_2	122,5 cm	(keiner)	(keiner)
KH_ST_0	KIO_LN_0 - KH_ST_3	98,7 cm	(keiner)	(keiner)
KH_ST_0	KIO_LN_0 - KH_ST_4	74,9 cm	(keiner)	(keiner)
KH_ST_0	KIO_LN_0 - KH_ST_5	51,4 cm	(keiner)	(keiner)
KH_ST_1	Komplett	231,1 cm	35,9 cm	196,1 cm
KH_ST_2	Komplett	207,0 cm	34,8 cm	172,1 cm
KH_ST_3	Komplett	207,0 cm	34,8 cm	172,0 cm
KH_ST_4	Komplett	207,0 cm	34,2 cm	172,0 cm
KH_ST_5	Komplett	231,1 cm	39,9 cm	196,0 cm
KH_ST_6	KH_ST_1 - KH_LN_0	57,9 cm	(keiner)	(keiner)
KH_ST_6	KH_ST_2 - KH_LN_0	81,4 cm	(keiner)	(keiner)
KH_ST_6	KH_ST_3 - KH_LN_0	105,2 cm	(keiner)	(keiner)
KH_ST_6	KH_ST_4 - KH_LN_0	129,0 cm	(keiner)	(keiner)
KH_ST_6	KH_ST_5 - KH_LN_0	128,7 cm	(keiner)	(keiner)
KH_ST_6	KH_ST_1 - KIO_LN_1	51,4 cm	(keiner)	(keiner)
KH_ST_6	KH_ST_2 - KIO_LN_1	74,9 cm	(keiner)	(keiner)
KH_ST_6	KH_ST_3 - KIO_LN_1	98,7 cm	(keiner)	(keiner)
KH_ST_6	KH_ST_4 - KIO_LN_1	122,5 cm	(keiner)	(keiner)
KH_ST_6	KH_ST_5 - KIO_LN_1	122,3 cm	(keiner)	(keiner)
KH_LN_0	Komplett	458,8 cm	63,3 cm	423,4 cm
KH_LN_1	Komplett	268,9 cm	34,4 cm	246,9 cm
KH_LN_2	KH_LN_2 - KH_LN_3	254,7 cm	37,5 cm	193,7 cm

KH_LN_2	KH_LN_2 - KH_LN_4	255,0 cm	37,5 cm	193,7 cm
KH_LN_3	Komplett	236,6 cm	34,2 cm	201,3 cm
KH_LN_4	Komplett	246,5 cm	35,5 cm	214,3 cm
KH_LN_5	Komplett	258,0 cm	34,1 cm	222,2 cm
KH_LN_6	Komplett	248,1 cm	34,1 cm	213,9 cm
KH_LN_7	KH_LN_5 - KH_LN_7	426,1 cm	56,3 cm	390,2 cm
KH_LN_7	KH_LN_6 - KH_LN_7	425,9 cm	33,5 cm	390,2 cm
KH_LN_8	Komplett	356,0 cm	36,8 cm	288,6 cm

Wendeschleifen und Verbindungen

Block	Durchquerung	Länge	Kontakt 0	Kontakt 1
IO_LN_0	Komplett	308,1 cm	19,5 cm	268,1 cm
IO_LN_1	Komplett	308,7 cm	11,3 cm	274,6 cm
IO_LN_2	Komplett	296,3 cm	6,9 cm	260,7 cm
OI_LN_0	Komplett	265,6 cm	11,4 cm	225,6 cm
OI_LN_1	Komplett	256,9 cm	17,0 cm	220,1 cm
OI_LN_2	Komplett	239,1 cm	17,0 cm	203,5 cm
KIO_LN_0	Komplett	195,1 cm	27,4 cm	152,5 cm
KIO_LN_1	Komplett	218,9 cm	43,2 cm	183,1 cm

Tabelle B.5: Alle Blocklängen der Anlage

B.5 Fahrplanbausteine

Auf der Modellbahnanlage gibt es nur eine kleine Menge möglicher Strecken, die ein Zug auf dem Weg von einem Bahnhof zu einem anderen zurücklegen kann. Die Verbindungen sind sogar eindeutig, wenn die Ausweichgleise im Kicking Horse Pass in der Vorzugsrichtung befahren werden. Damit setzt sich jeder Fahrplan eines Zuges aus einigen wenigen Bauteilen zusammen, welche die folgende Tabelle aufführt. Die Bahnhofsgleise am Anfang und Ende der Strecke sind durch ein Sternchen angedeutet, das Programm muß hier das jeweils gewünschte Gleis einsetzen.

Durchfahrt Outer Circle

OC_ST_*, (OC_ST_4), OC_LN_0, (OC_JCT_0), OC_LN_1, OC_LN_2, OC_LN_3, OC_LN_4, OC_LN_5, (OC_ST_0), OC_ST_*

Durchfahrt Inner Circle

IC_ST_*, (IC_ST_4), IC_LN_0, IC_LN_1, IC_LN_2, IC_LN_3, IC_LN_4, (IC_JCT_0), IC_LN_5, (IC_ST_0), IC_ST_*

Durchfahrt Kicking Horse Pass vorwärts

KH_ST_*, (KH_ST_6), KH_LN_0, KH_LN_1, KH_LN_2, KH_LN_4, KH_LN_6, KH_LN_7, KH_LN_8, (KH_ST_0), KH_ST_*

Durchfahrt Kicking Horse Pass rückwärts

KH_ST_*, (KH_ST_0), KH_LN_8, KH_LN_7, KH_LN_5, KH_LN_3, KH_LN_2, KH_LN_1, KH_LN_0, (KH_ST_6), KH_ST_*

Wechsel von Outer in Inner Circle
OC_ST_*, (OC_ST_4), OC_LN_0, (OC_JCT_0), OI_LN_0, OI_LN_1, OI_LN_2, (IC_JCT_0), IC_LN_5, (IC_ST_0), IC_ST_*

Wechsel von Inner in Outer Circle
IC_ST_*, (IC_ST_4), IC_LN_0, IC_LN_1, IC_LN_2, IC_LN_3, IC_LN_4, (IC_JCT_0), IO_LN_0, IO_LN_1, IO_LN_2, (OC_JCT_0), OC_LN_1, OC_LN_2, OC_LN_3, OC_LN_4, OC_LN_5, (OC_ST_0), OC_ST_*

Wechsel von Outer Circle in Kicking Horse Pass (vorwärts)
OC_ST_*, (OC_ST_4), KIO_LN_0, (KH_ST_0), KH_ST_*

Wechsel von Inner Circle in Kicking Horse Pass (rückwärts)
IC_ST_*, (IC_ST_4), (OC_ST_0), KIO_LN_1, (KH_ST_6), KH_ST_*

Wechsel von Kicking Horse Pass (vorwärts) in Outer Circle
KH_ST_*, (KH_ST_6), KIO_LN_1, (OC_ST_0), OC_ST_*

Wechsel von Kicking Horse Pass (rückwärts) in Inner Circle
KH_ST_*, (KH_ST_0), KIO_LN_0, (OC_ST_4), (IC_ST_0), IC_ST_*

Tabelle B.6: Grundbausteine für alle Fahrpläne

Soll ein Zug von einem der anderen Bahnhöfe in den Kicking Horse Pass fahren und dort mit einer anderen Zielrichtung ankommen, als oben in der Tabelle aufgeführt ist, muß der Zug zunächst in den anderen Kreis und von dort in den Paß fahren. Entsprechendes gilt bei einer Rückkehr aus dem Paß, wenn die Ausgangsrichtung nicht mit der gewünschten übereinstimmt.

B.6 Regeln für den Blockübergang

Wenn ein Zug sich entlang so einer Blocksequenz bewegen soll, muß auf seinem Weg Fahrstrom mit der richtigen Polarität aktiviert werden, Weichen müssen geschaltet werden und Signale in den richtigen Farben leuchten. Die meisten dieser Informationen lassen sich im Gleisplan direkt ablesen, lediglich bei den Signalen sind zusätzliche Regeln nötig.

- Wenn ein Zug in einen Block einfährt, muß das Signal Grün oder Gelb/Grün zeigen.
- Ob die Farbe Gelb gesetzt werden muß, hängt von den Weichen ab, die zwischen dem aktuellen und dem nächsten Signal liegen. Ist mindestens eine von diesen auf Abbiegen geschaltet, muß die gelbe Lampe eingeschaltet werden.
- Die Kreuzungsweichen nehmen eine Sonderstellung ein, bei ihnen wird nur eine der drei Einstellungen als Abbiegen gewertet.

Die Tabelle B.7 faßt diese Daten für alle Blöcke und die erlaubten Fahrtrichtungen zusammen. Interblocks treten dabei nach Definition nie als Anfang oder Ende eines Streckenabschnitts auf. Für die Ausweichgleise sind in der Tabelle zwei Alternativen vorhanden, die normalen Strecken enthalten die im Gleisplan vorgeschlagenen Richtungen, zusätzlich sind aber noch die Daten für die jeweiligen Gegenrichtungen angegeben.

Durchfahrt Outer Circle

Von	Nach	Über	Weichen	Gelb
OC_ST_1 (vor)	OC_LN_0 (vor)	OC_ST_4 (vor)	(22)=0 (21)=1 (16)=0 (17)=0	Ja
OC_ST_2 (vor)	OC_LN_0 (vor)	OC_ST_4 (vor)	(22)=1 (21)=1 (16)=0 (17)=0	Ja
OC_ST_3 (vor)	OC_LN_0 (vor)	OC_ST_4 (vor)	(21)=0 (16)=0 (17)=0	Nein
OC_LN_0 (vor)	OC_LN_1 (vor)	OC_JCT_0 (vor)	(10)=0 (12)=0	Nein
OC_LN_1 (vor)	OC_LN_2 (vor)	Direkt	Keine	-
OC_LN_2 (vor)	OC_LN_3 (vor)	Direkt	Keine	-
OC_LN_3 (vor)	OC_LN_4 (vor)	Direkt	Keine	-
OC_LN_4 (vor)	OC_LN_5 (vor)	Direkt	Keine	-
OC_LN_5 (vor)	OC_ST_1 (vor)	OC_ST_0 (vor)	(27)=1 (28)=1 (25)=1 (26)=0	Ja
OC_LN_5 (vor)	OC_ST_2 (vor)	OC_ST_0 (vor)	(27)=1 (28)=1 (25)=1 (26)=1	Ja
OC_LN_5 (vor)	OC_ST_3 (vor)	OC_ST_0 (vor)	(27)=1 (28)=1 (25)=0	Nein

Durchfahrt Inner Circle

Von	Nach	Über	Weichen	Gelb
IC_ST_1 (vor)	IC_LN_0 (vor)	IC_ST_4 (vor)	(24)=0 (29)=0	Nein
IC_ST_2 (vor)	IC_LN_0 (vor)	IC_ST_4 (vor)	(23)=1 (24)=1 (29)=0	Ja
IC_ST_3 (vor)	IC_LN_0 (vor)	IC_ST_4 (vor)	(23)=0 (24)=1 (29)=0	Ja
IC_LN_0 (vor)	IC_LN_1 (vor)	Direkt	Keine	-
IC_LN_1 (vor)	IC_LN_2 (vor)	Direkt	Keine	-
IC_LN_2 (vor)	IC_LN_3 (vor)	Direkt	Keine	-
IC_LN_3 (vor)	IC_LN_4 (vor)	Direkt	Keine	-
IC_LN_4 (vor)	IC_LN_5 (vor)	IC_JCT_0 (vor)	(13)=0 (11)=0	Nein
IC_LN_5 (vor)	IC_ST_1 (vor)	IC_ST_0 (vor)	(18)=0 (20)=0	Nein
IC_LN_5 (vor)	IC_ST_2 (vor)	IC_ST_0 (vor)	(18)=0 (20)=1 (19)=1	Ja
IC_LN_5 (vor)	IC_ST_3 (vor)	IC_ST_0 (vor)	(18)=0 (20)=1 (19)=0	Ja

Wendeschleifen

Von	Nach	Über	Weichen	Gelb
IC_LN_4 (vor)	IO_LN_0 (vor)	IC_JCT_0 (vor)	(13)=0 (11)=1	Ja
OI_LN_2 (vor)	IO_LN_0 (vor)	IC_JCT_0 (vor)	(13)=1 (11)=1	Ja
IO_LN_0 (vor)	IO_LN_1 (vor)	Direkt	Keine	-
IO_LN_1 (vor)	IO_LN_2 (vor)	Direkt	Keine	-
IO_LN_2 (vor)	OC_LN_1 (vor)	OC_JCT_0 (vor)	(10)=1 (12)=0	Ja
IO_LN_2 (vor)	OI_LN_0 (vor)	OC_JCT_0 (vor)	(10)=1 (12)=1	Ja
OC_LN_0 (vor)	OI_LN_0 (vor)	OC_JCT_0 (vor)	(10)=0 (12)=1	Ja
OI_LN_0 (vor)	OI_LN_1 (vor)	Direkt	Keine	-
OI_LN_1 (vor)	OI_LN_2 (vor)	Direkt	Keine	-
OI_LN_2 (vor)	IC_LN_5 (vor)	IC_JCT_0 (vor)	(13)=1 (11)=0	Ja

Kicking Horse Pass (vorwärts, Ausweichgleise nach Plan)

Von	Nach	Über	Weichen	Gelb
KH_ST_1 (vor)	KH_LN_0 (vor)	KH_ST_6 (vor)	(8)=0 (9)=0	Nein
KH_ST_2 (vor)	KH_LN_0 (vor)	KH_ST_6 (vor)	(7)=1 (8)=1 (9)=0	Ja
KH_ST_3 (vor)	KH_LN_0 (vor)	KH_ST_6 (vor)	(6)=1 (7)=0 (8)=1 (9)=0	Ja
KH_ST_4 (vor)	KH_LN_0 (vor)	KH_ST_6 (vor)	(5)=1 (6)=0 (7)=0 (8)=1 (9)=0	Ja
KH_ST_5 (vor)	KH_LN_0 (vor)	KH_ST_6 (vor)	(5)=0 (6)=0 (7)=0 (8)=1 (9)=0	Ja
KH_LN_0 (vor)	KH_LN_1 (vor)	Direkt	Keine	-
KH_LN_1 (vor)	KH_LN_2 (vor)	Direkt	Keine	-
KH_LN_2 (vor)	KH_LN_4 (vor)	Direkt	(15)=1	Ja
KH_LN_4 (vor)	KH_LN_6 (vor)	Direkt	Keine	-
KH_LN_6 (vor)	KH_LN_7 (vor)	Direkt	(14)=0	Nein
KH_LN_7 (vor)	KH_LN_8 (vor)	Direkt	Keine	-
KH_LN_8 (vor)	KH_ST_1 (vor)	KH_ST_0 (vor)	(0)=0 (1)=1 (2)=0 (3)=0 (4)=0	Ja
KH_LN_8 (vor)	KH_ST_2 (vor)	KH_ST_0 (vor)	(0)=0 (1)=1 (2)=0 (3)=0 (4)=1	Ja
KH_LN_8 (vor)	KH_ST_3 (vor)	KH_ST_0 (vor)	(0)=0 (1)=1 (2)=0 (3)=1	Ja
KH_LN_8 (vor)	KH_ST_4 (vor)	KH_ST_0 (vor)	(0)=0 (1)=1 (2)=1	Ja
KH_LN_8 (vor)	KH_ST_5 (vor)	KH_ST_0 (vor)	(0)=0 (1)=0	Nein

Kicking Horse Pass (rückwärts, Ausweichgleise nach Plan)

Von	Nach	Über	Weichen	Gelb
KH_ST_1 (rück)	KH_LN_8 (rück)	KH_ST_0 (rück)	(4)=0 (3)=0 (2)=0 (1)=1 (0)=0	Ja
KH_ST_2 (rück)	KH_LN_8 (rück)	KH_ST_0 (rück)	(4)=1 (3)=0 (2)=0 (1)=1 (0)=0	Ja
KH_ST_3 (rück)	KH_LN_8 (rück)	KH_ST_0 (rück)	(3)=1 (2)=0 (1)=1 (0)=0	Ja
KH_ST_4 (rück)	KH_LN_8 (rück)	KH_ST_0 (rück)	(2)=1 (1)=1 (0)=0	Ja
KH_ST_5 (rück)	KH_LN_8 (rück)	KH_ST_0 (rück)	(1)=0 (0)=0	Nein
KH_LN_8 (rück)	KH_LN_7 (rück)	Direkt	Keine	-
KH_LN_7 (rück)	KH_LN_5 (rück)	Direkt	(14)=1	Ja
KH_LN_5 (rück)	KH_LN_3 (rück)	Direkt	Keine	-
KH_LN_3 (rück)	KH_LN_2 (rück)	Direkt	(15)=0	Nein
KH_LN_2 (rück)	KH_LN_1 (rück)	Direkt	Keine	-
KH_LN_1 (rück)	KH_LN_0 (rück)	Direkt	Keine	-
KH_LN_0 (rück)	KH_ST_1 (rück)	KH_ST_6 (rück)	(9)=0 (8)=0	Nein
KH_LN_0 (rück)	KH_ST_2 (rück)	KH_ST_6 (rück)	(9)=0 (8)=1 (7)=1	Ja
KH_LN_0 (rück)	KH_ST_3 (rück)	KH_ST_6 (rück)	(9)=0 (8)=1 (7)=0 (6)=1	Ja
KH_LN_0 (rück)	KH_ST_4 (rück)	KH_ST_6 (rück)	(9)=0 (8)=1 (7)=0 (6)=0 (5)=1	Ja
KH_LN_0 (rück)	KH_ST_5 (rück)	KH_ST_6 (rück)	(9)=0 (8)=1 (7)=0 (6)=0 (5)=0	Ja

Kicking Horse Pass (Ausweichgleise in Gegenrichtung)

Von	Nach	Über	Weichen	Gelb
KH_LN_2 (vor)	KH_LN_3 (vor)	Direkt	(15)=0	Nein
KH_LN_3 (vor)	KH_LN_5 (vor)	Direkt	Keine	-
KH_LN_5 (vor)	KH_LN_7 (vor)	Direkt	(14)=1	Ja
KH_LN_7 (rück)	KH_LN_6 (rück)	Direkt	(14)=0	Nein
KH_LN_6 (rück)	KH_LN_4 (rück)	Direkt	Keine	-
KH_LN_4 (rück)	KH_LN_2 (rück)	Direkt	(15)=1	Ja

Verbindungen über KIO_LN_0

Von	Nach	Über	Weichen	Gelb
OC_ST_1 (vor)	KIO_LN_0 (vor)	OC_ST_4 (vor)	(22)=0 (21)=1 (16)=0 (17)=1	Ja
OC_ST_2 (vor)	KIO_LN_0 (vor)	OC_ST_4 (vor)	(22)=1 (21)=1 (16)=0 (17)=1	Ja
OC_ST_3 (vor)	KIO_LN_0 (vor)	OC_ST_4 (vor)	(21)=0 (16)=0 (17)=1	Ja
KIO_LN_0 (vor)	KH_ST_1 (vor)	KH_ST_0 (vor)	(0)=1 (1)=1 (2)=0 (3)=0 (4)=0	Ja
KIO_LN_0 (vor)	KH_ST_2 (vor)	KH_ST_0 (vor)	(0)=1 (1)=1 (2)=0 (3)=0 (4)=1	Ja
KIO_LN_0 (vor)	KH_ST_3 (vor)	KH_ST_0 (vor)	(0)=1 (1)=1 (2)=0 (3)=1	Ja
KIO_LN_0 (vor)	KH_ST_4 (vor)	KH_ST_0 (vor)	(0)=1 (1)=1 (2)=1	Ja
KIO_LN_0 (vor)	KH_ST_5 (vor)	KH_ST_0 (vor)	(0)=1 (1)=0	Ja
KH_ST_1 (rück)	KIO_LN_0 (rück)	KH_ST_0 (rück)	(4)=0 (3)=0 (2)=0 (1)=1 (0)=1	Ja
KH_ST_2 (rück)	KIO_LN_0 (rück)	KH_ST_0 (rück)	(4)=1 (3)=0 (2)=0 (1)=1 (0)=1	Ja
KH_ST_3 (rück)	KIO_LN_0 (rück)	KH_ST_0 (rück)	(3)=1 (2)=0 (1)=1 (0)=1	Ja
KH_ST_4 (rück)	KIO_LN_0 (rück)	KH_ST_0 (rück)	(2)=1 (1)=1 (0)=1	Ja
KH_ST_5 (rück)	KIO_LN_0 (rück)	KH_ST_0 (rück)	(1)=0 (0)=1	Ja
KIO_LN_0 (rück)	IC_ST_1 (vor)	OC_ST_4 (rück) und IC_ST_0 (vor)	(16)=1 (17)=1 (18)=1 (20)=0	Ja
KIO_LN_0 (rück)	IC_ST_2 (vor)	OC_ST_4 (rück) und IC_ST_0 (vor)	(16)=1 (17)=1 (18)=1 (20)=1 (19)=1	Ja
KIO_LN_0 (rück)	IC_ST_3 (vor)	OC_ST_4 (rück) und IC_ST_0 (vor)	(16)=1 (17)=1 (18)=1 (20)=1 (19)=0	Ja

Verbindungen über KIO_LN_1

Von	Nach	Über	Weichen	Gelb
IC_ST_1 (vor)	KIO_LN_1 (rück)	IC_ST_4 (vor) und OC_ST_0 (rück)	(24)=0 (29)=1 (27)=0 (28)=0	Ja
IC_ST_2 (vor)	KIO_LN_1 (rück)	IC_ST_4 (vor) und OC_ST_0 (rück)	(23)=1 (24)=1 (29)=1 (27)=0 (28)=0	Ja
IC_ST_3 (vor)	KIO_LN_1 (rück)	IC_ST_4 (vor) und OC_ST_0 (rück)	(23)=0 (24)=1 (29)=1 (27)=0 (28)=0	Ja
KIO_LN_1 (rück)	KH_ST_1 (rück)	KH_ST_6 (rück)	(9)=1 (8)=0	Ja
KIO_LN_1 (rück)	KH_ST_2 (rück)	KH_ST_6 (rück)	(9)=1 (8)=1 (7)=1	Ja
KIO_LN_1 (rück)	KH_ST_3 (rück)	KH_ST_6 (rück)	(9)=1 (8)=1 (7)=0 (6)=1	Ja
KIO_LN_1 (rück)	KH_ST_4 (rück)	KH_ST_6 (rück)	(9)=1 (8)=1 (7)=0 (6)=0 (5)=1	Ja
KIO_LN_1 (rück)	KH_ST_5 (rück)	KH_ST_6 (rück)	(9)=1 (8)=1 (7)=0 (6)=0 (5)=0	Ja
KH_ST_1 (vor)	KIO_LN_1 (vor)	KH_ST_6 (vor)	(8)=0 (9)=1	Ja

KH_ST_2 (vor)	KIO_LN_1 (vor)	KH_ST_6 (vor)	(7)=1 (8)=1 (9)=1	Ja
KH_ST_3 (vor)	KIO_LN_1 (vor)	KH_ST_6 (vor)	(6)=1 (7)=0 (8)=1 (9)=1	Ja
KH_ST_4 (vor)	KIO_LN_1 (vor)	KH_ST_6 (vor)	(5)=1 (6)=0 (7)=0 (8)=1 (9)=1	Ja
KH_ST_5 (vor)	KIO_LN_1 (vor)	KH_ST_6 (vor)	(5)=0 (6)=0 (7)=0 (8)=1 (9)=1	Ja
KIO_LN_1 (vor)	OC_ST_1 (vor)	OC_ST_0 (vor)	(27)=1 (28)=0 (25)=1 (26)=0	Ja
KIO_LN_1 (vor)	OC_ST_2 (vor)	OC_ST_0 (vor)	(27)=1 (28)=0 (25)=1 (26)=1	Ja
KIO_LN_1 (vor)	OC_ST_3 (vor)	OC_ST_0 (vor)	(27)=1 (28)=0 (25)=0	Ja

Tabelle B.7: Regeln für die Blockübergänge

Anhang C

Embedded Linux

Die PC104-Rechner benötigen ein eigenes Betriebssystem, daß an ihre Hardware angepaßt ist, mit sehr wenig Speicher auskommt und vollständig über das Netz booten und arbeiten kann. Dafür kommt aus praktischen Gründen nur Linux in Frage, es ist hervorragend dokumentiert, läßt sich weitgehend konfigurieren und erlaubt die Programmierung mit Standardwerkzeugen. System und Anwendungen müssen mit Hilfe eines Crosscompilers übersetzt werden, der Code für den 386er erzeugen kann und dabei auf eine speichersparende Version der `libc` zurückgreift. Dieses Kapitel beschreibt alle Schritte, mit denen der Crosscompiler und alle Bestandteile des Systems erzeugt und installiert werden. Einige Abschnitte widmen sich der Benutzerverwaltung und dem Übersetzen eigener Software.

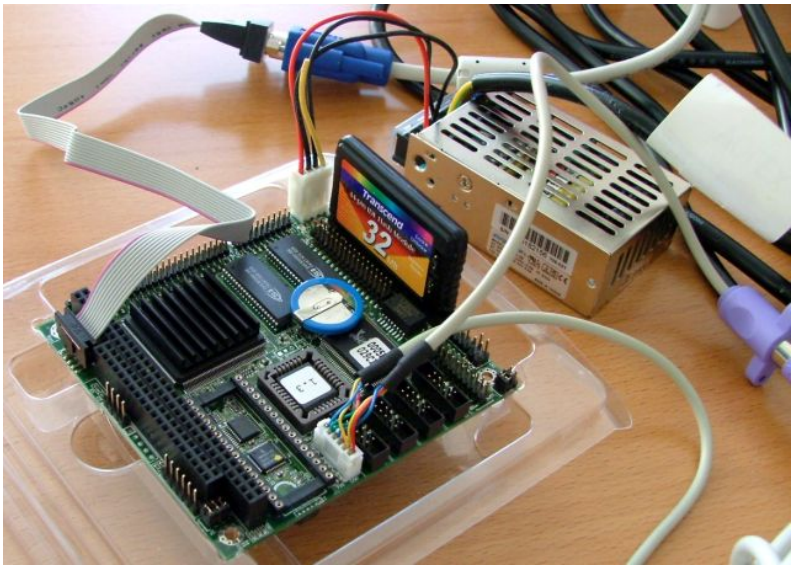


Abbildung C.1: PC104-Rechner im Testbetrieb

C.1 Betriebssystem für PC104-Rechner

Die Erzeugung eines Crosscompilers mit Integration einer bestimmten `libc` ist ein komplexer Vorgang, der einiges an Wissen und Erfahrung erfordert. Ohne die erforderlichen Patches und Konfigurationseinstellungen scheitert in der Regel bereits der Versuch. Glücklicherweise gibt es

mit `buildroot` ein System, das den gesamten Prozeß automatisiert und zusätzlich ein einfaches Root-Dateisystem mit konfigurierbarer Softwareausstattung erzeugen kann. Der Benutzer wählt menügesteuert die gewünschten Eigenschaften aus, `buildroot` lädt alle nötigen Dateien aus dem Internet, übersetzt sie und erstellt dabei neben dem Crosscompiler ein TAR-Archiv, das sich in dem späteren Root-Dateisystem auspacken läßt.

C.1.1 Vorbereitung der Installation

Der hier beschriebene Buildprozeß geht davon aus, daß alle erforderlichen Quellcodearchive und Konfigurationsdateien im Home-Verzeichnis des Benutzers `pc104` liegen. Dort wird später auch der Crosscompiler installiert. Die Quellen befinden sich auf der beiliegenden DVD, wichtig sind die Verzeichnisse `source` und `config`. Zum Übersetzen müssen die üblichen Werkzeuge installiert sein, zusätzlich werden die Pakete `etherboot`, `grub`, `syslinux` und die Entwicklerversionen von `ncurses` und `gettext` gebraucht.

Die Installation der Kernel-Module erfordert eine Version des Programmes `depmod`, die für Kernel der Serie 2.4 geeignet ist und bei neueren Distributionen zum Beispiel `depmod.old` heißen kann. Das Arbeitsverzeichnis für den gesamten Vorgang sollte auf einer lokalen Festplatte liegen, die mehr als ein Gigabyte freien Speicherplatz bieten muß.

Zu Beginn der Installation werden die wichtigsten Verzeichnisse in Variablen gespeichert.

```
SOURCE=/home/pc104/source
CONFIG=/home/pc104/config
TMP=/tmp
```

Die Variable `TOOLCHAIN` legt das Zielverzeichnis für den Crosscompiler fest, `ROOTFS` verweist in die spätere `buildroot`-Installation und gibt den Ort an, wo die Vorlage für das Root-Dateisystem später liegen wird.

```
TOOLCHAIN=/home/pc104/toolchain
ROOTFS=$TMP/buildroot/build_i386/root
```

Es dürfen keine Reste von einer vorher durchgeführten Installationen existieren, insbesondere müssen die Quellcodeverzeichnisse und der Crosscompiler gelöscht sein.

```
rm -rf $TMP/buildroot
rm -rf $TMP/linux-2.4.31
rm -rf $TMP/peak-linux-driver-3.17
rm -rf $TOOLCHAIN
```

C.1.2 Basissystem und Crosscompiler

Den größten Teil der Arbeit übernimmt jetzt `buildroot`, zunächst wird das Archiv ausgepackt.

```
cd $TMP
tar xfvj $SOURCE/buildroot/buildroot-snapshot.tar.bz2
cd buildroot
```

Damit das Ergebnis reproduzierbar bleibt, sollen die Quelldateien nicht neu aus dem Internet heruntergeladen werden. Stattdessen werden Versionen in das `buildroot`-Verzeichnis kopiert, die sich als funktionierend herausgestellt haben.

```
cp -av $SOURCE/buildroot/dl .
```

Das Paket unterstützt das Remote Shell Protokoll nicht, ein Patch ergänzt die Funktionalität.

```
patch -p0 < $SOURCE/buildroot/buildroot-netkitrsh.patch
```

Die Konfiguration des Paketes wird aus den Vorgaben übernommen. Die Einstellungen und der Softwareumfang können in den verschiedenen Dateien oder über die Menüs von `buildroot` eingesehen und verändert werden.

```
cp $CONFIG/buildroot .config
cp $CONFIG/busybox package/busybox/busybox.config
cp $CONFIG/uclibc toolchain/uClibc/uClibc.config
```

Wenn der Crosscompiler in einem anderen Verzeichnis als `/home/pc104/toolchain` installiert werden soll, müssen in `.config` und `toolchain/uClibc/uClibc.config` die Einstellungen von `BR2_STAGING_DIR` sowie `CROSS_COMPILER_PREFIX` entsprechend geändert werden. Damit ist das Paket bereit zum Übersetzen.

```
make oldconfig
make
```

Nach dem erfolgreichen Durchlauf ist der Crosscompiler einsatzbereit und ein erstes lauffähiges Root-Dateisystem erstellt. Es wird in den nächsten Schritten mit den noch fehlenden Tools angereichert und konfiguriert.

C.1.3 Übersetzung des Linux-Kernels

Der Linux-Kernel wird passend für das PC104-System erzeugt und später auf die Flashdisc sowie in ein Image für den Bootserver verpackt. Alle Einstellungen sind auf möglichst geringen Speicherbedarf hin optimiert, die meisten Funktionen deaktiviert und alle wichtigen fest in den Kernel eingebaut. Besonders wichtig sind die Fließkomma-Emulation, die Netzwerkfunktionen und die Möglichkeit zum Mounten des Root-Dateisystems per NFS Version 3.

Zuerst wird der Quellcode ausgepackt.

```
cd $TMP
tar xfvj $SOURCE/linux/linux-2.4.31.tar.bz2
cd linux-2.4.31
```

Eine problematische Header-Datei muß vor dem Compilieren noch korrigiert werden.

```
patch -p0 < $SOURCE/linux/linux-2.4.31-asmerror.patch
```

Dann wird die gespeicherten Konfiguration übernommen und der Kernel übersetzt.

```
make mrproper
cp $CONFIG/linux .config
make oldconfig
make dep
make bzImage
make modules
```

Der Kernel und die erzeugten Module werden in das Root-Image von `buildroot` installiert.

```
cp -a arch/i386/boot/bzImage $ROOTFS
cp -a System.map $ROOTFS
INSTALL_MOD_PATH=$ROOTFS make modules_install
rm -f $ROOTFS/lib/modules/2.4.31/build
```

An dieser Stelle kann eine Warnung ausgegeben werden, falls das Paket modutils nicht installiert ist. Sie darf ignoriert werden, das spätere System sollte auch ohne den Aufruf von `depmod -a` problemlos benutzbar sein.

C.1.4 CAN-Treiber, Library und Testprogramme

Das Treiberpaket von Peak System enthält neben dem Kernelmodul die Bibliotheken sowie verschiedene Testprogramme. Zunächst wird wieder der Quellcode ausgepackt.

```
cd $TMP
tar xfvz $SOURCE/pcan/peak-linux-driver.3.17.tar.gz
cd peak-linux-driver-3.17
```

Ein einfacher Befehl compiliert das Modul gegen die zuvor benutzten Kernelquellen.

```
make -C driver -f Makefile-2.4 CC=$TOOLCHAIN/bin/i386-linux-uclibc-gcc \
  KERNEL_LOCATION=$TMP/linux-2.4.31 \
  USB=NO_USB_SUPPORT PAR=NO_PARPORT_SUBSYSTEM
```

Die im Paket enthaltene Library wird zur Entwicklung eigener Programme benötigt.

```
make -C lib CC=$TOOLCHAIN/bin/i386-linux-uclibc-gcc
```

Das Modul findet seinen Platz in dem Root-Image von `buildroot`.

```
install -D -m644 driver/pcan.o $ROOTFS/lib/modules/2.4.31/misc/pcan.o
```

Die CAN-Library wird im Dateisystem der PC104-Rechner installiert.

```
install -m644 lib/libpcan.so.0.3 $ROOTFS/lib
ln -sf libpcan.so.0.3 $ROOTFS/lib/libpcan.so
```

Um Programme mit CAN-Unterstützung erzeugen zu können, werden Header und Library in den Crosscompiler integriert.

```
install -m644 driver/pcan.h $TOOLCHAIN/include
install -m644 lib/libpcan.h $TOOLCHAIN/include
install -m644 lib/libpcan.so.0.3 $TOOLCHAIN/lib
ln -sf libpcan.so.0.3 $TOOLCHAIN/lib/libpcan.so
```

Schließlich enthält das Paket noch einige Testprogramme für den CAN-Bus. Sie können später zum Beispiel in das Home-Verzeichnis eines Benutzers kopiert werden.

```
make -C test CC=$TOOLCHAIN/bin/i386-linux-uclibc-gcc LDLIBS=-L$TOOLCHAIN/lib
strip test/{bitrate,receive,transmi}test
```

C.1.5 Konfiguration des Basissystems

Das von `buildroot` erzeugte Dateisystem muß noch für den späteren Einsatz konfiguriert werden. Dazu werden einige Dateien verändert oder hinzugefügt und das Dateisystem neu erstellt.

```
cd $TMP/buildroot
```

Die Liste der zu erzeugenden Devices enthält zu wenig Loop-Devices sowie keinerlei Einträge für das `ppdev`-Interface und die CAN-Controller. Ein Patch korrigiert diese Punkte.

```
patch -p0 < $SOURCE/buildroot/buildroot-devices.patch
```

Das Paßwort für `root\verb` ist bisher leer und wird zur Sicherheit in „lummerland“ geändert.

```
sed -i -e 's#^root:[^:]*:#root:$1$KGJY4wAY$y4AM4vtw/I9J2NQswkQ17.:#' \
  $ROOTFS/etc/shadow
```

Der in `buildroot` standardmäßige vorhandene Benutzer `default` wird nicht benötigt.

```
sed -i -e '/^default:/d;' $ROOTFS/etc/{passwd,shadow,group}
rm -rf $ROOTFS/home/default
```

In einem letzten Schritt werden Dateien des Systems überschrieben oder durch neue ergänzt. Zu den wichtigsten Änderungen gehören ein neues Bootskript und eine deutsche Tastaturbelegung. Damit wird ein Image für den Betrieb unabhängig vom Netzwerk erstellt.

```
rm -f $ROOTFS/etc/resolv.conf
cp -av $SOURCE/rootfs/standalone/* $ROOTFS
rm -f $TOOLCHAIN/fakeroot.env
touch $TOOLCHAIN/fakeroot.env
make
cp rootfs.i386.tar rootfs-standalone.i386.tar
```

Der Prozeß wiederholt sich für die Netzwerkversion mit anderen Dateien.

```
rm -f $ROOTFS/etc/resolv.conf
cp -av $SOURCE/rootfs/nfsclient/* $ROOTFS
rm -f $TOOLCHAIN/fakeroot.env
touch $TOOLCHAIN/fakeroot.env
make
cp rootfs.i386.tar rootfs-nfsclient.i386.tar
```

C.1.6 Einrichtung der Flashdisc

Jeder PC104-Rechner verfügt über eine Festplatte in Form einer Flashdisc mit einer Kapazität von 32 MB. Die Platte muß an einem Notebook, über einen Adapter an einem Desktop oder direkt an einem USB-Wechselplattenrahmen für 2.5 Zoll-Festplatten betrieben werden, um das Bootsystem des PC104-Rechners zu installieren.

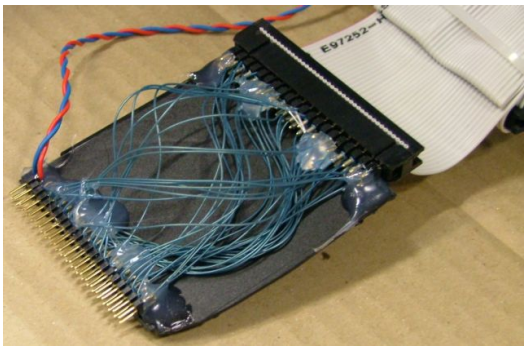


Abbildung C.2: Provisorischer Adapter für Flashdisc

Die Flashdisc ist das einzige bootfähige Medium, da die Rechner weder über ein Boot-ROM noch über ein Diskettenlaufwerk verfügen. Auf ihnen wird `grub` als Bootloader installiert. Er kann lokal gespeicherte Diskettenimages, ein System von einer lokalen Partition oder ein Image von einem Bootserver starten. Dafür werden drei Partitionen angelegt, eine große für ein Linux, das ohne Netzwerk funktioniert und zur Diagnose benutzt werden kann, eine dazugehörige Swap-Partition sowie eine kleine Partition für `grub` und die Bootimages. Durch die Trennung von Linux und `grub` ist der Bootloader vor versehentlichen Änderungen besser geschützt.

Partitionierung

Zunächst muß die Flashdisc in einen Rechner eingebaut, partitioniert und formatiert werden. Die nächsten Schritte gehen davon aus, daß sie unter `/dev/sda` erreichbar ist und der Benutzer als `root` arbeitet. Falls gewünscht, wird die Platte zuerst mit Nullbytes überschreiben.

```
dd if=/dev/zero of=/dev/sda bs=65536 count=496
```

Jetzt wird ein MBR installiert, 0x80 entspricht dem späteren Namen `/dev/hda` der Platte.

```
install-mbr /dev/sda -d 0x80
```

Falls das Programm `install-mbr` nicht existiert, übernimmt der folgende Befehl dessen Aufgabe.

```
dd if=$SOURCE/bootimage/mbr.img of=/dev/sda bs=512 count=1
```

Jetzt können zum Beispiel mit `fdisk` die benötigten Partitionen angelegt und aktiviert werden.

```
fdisk /dev/sda
```

Das Ergebnis muß folgendermaßen aussehen. Die Anzahl der Köpfe, Sektoren und Zylinder muß im Expertenmodus von `fdisk` auf die angegebenen Werte umgestellt werden, falls sie nicht stimmen sollten. Die dritte Partition wird den Bootmanager enthalten, sie muß als bootfähig gekennzeichnet und aktiviert worden sein.

```
Disk /dev/sda: 32 MB, 32505856 bytes
  4 heads, 32 sectors/track, 496 cylinders
  Units = cylinders of 128 * 512 = 65536 bytes

     Device Boot      Start         End      Blocks   Id  System
  /dev/sda1               1         415       26544   83  Linux
  /dev/sda2             416         479        4096   82  Linux swap
  /dev/sda3   *         480         496        1088   83  Linux
```

In den Partitionen müssen noch die jeweiligen Dateisysteme erstellt werden. Um möglichst viel nutzbaren Platz übrig zu behalten, wird `ext2` als Dateisystem benutzt und kein Platz für den Root-Benutzer reserviert.

```
mke2fs -m0 /dev/sda1
mkswap /dev/sda2
mke2fs -m0 /dev/sda3
```

Konfigurationstool der Netzwerkkarte

Der Netzwerkchip der PC104-Rechner muß später konfiguriert werden, damit er vom Bootcode erkannt wird. Das dazu nötige DOS-Programm wird in ein Diskettenimage gepackt und auf die Flashdisc gespeichert. Die Basis bildet das Image einer 360 KB großen Bootdiskette von FreeDOS, das zunächst von allen überflüssigen Dateien befreit werden muß. Wichtig sind nur `kernel.sys` und `command.com`.

```
cp $SOURCE/freedos/b9boot01.img $TMP/realtek
mount -o loop -t msdos $TMP/realtek /mnt
rm -i /mnt/*
```

Das Tool von Realtek wird in das Image kopiert und eine passende `autoexec.bat` angelegt.

```
cp $SOURCE/realtek/rset8019.exe /mnt
echo -en '@echo off\r\nrset8019\r\n' > /mnt/autoexec.bat
umount /mnt
```

Die resultierende Datei ist auch unter `$SOURCE/bootimage/realtek` vorhanden.

Booten per Netzwerk

Beim Booten per Netzwerk muß auf dem Client eine entsprechende Software die Netzwerkkarte initialisieren, das Betriebssystem laden und starten. Diese Aufgabe übernimmt in der Regel das BIOS des Systems oder ein ROM-Baustein auf der Netzwerkkarte. Am leistungsfähigsten ist das aktuelle PXE, das aber für den älteren RTL8019(AS)-Chips der PC104-Rechner nicht verfügbar ist. Der Hersteller bietet nur ROMs mit dem veralteten RPL an, der dazugehörende Server `rpld` hat unter Linux das Entwicklungsstadium aber nie verlassen. Als einzige Alternativen bleiben `netboot` und `etherboot`, von denen nur letzteres noch gepflegt wird. Es war damit die einzig sinnvolle Wahl. Die Images des Projektes können in einem ROM-Baustein, auf einer Diskette, in einem Bootimage für `grub` und in einer Vielzahl anderer Formate abgelegt werden, was sehr viele Einsatzmöglichkeiten eröffnet. Als Imagedatei auf der Festplatte kann das Image einfach aktualisiert werden und es entfällt der Aufwand, für jeden Rechner ein EPROM zu brennen.

Das etherboot-Image kann auf der Seite `http://rom-o-matic.net/` erstellt werden. Dazu wird auf der Startseite Version 5.0.7 ausgewählt (neuere funktionieren nicht), „ne“ als NIC und „LILO bootable ROM“ als Imagetyp eingestellt. Unter „Configure“ werden einige Optionen geändert.

```
ASK_BOOT=-1
CONFIG_PCI_DIRECT
```

Der Knopf „Get ROM“ erzeugt ein Image, das unter dem Namen `etherboot` gespeichert wird. Die fertige Datei ist unter `$SOURCE/bootimage/etherboot` vorhanden.

Integration aller Systeme

Im letzten Schritt werden die Partitionen der Flashdisc mit den nötigen Dateien gefüllt. Um das System bootfähig zu machen, muß auf dem Wirtsrechner `grub` installiert sein. Aus dessen Lieferumfang werden die Dateien `stage1` und `stage2` benötigt, die auch auf einem 386er lauffähig sein müssen. Das selbe gilt für `memdisk` auf dem `syslinux`-Paket, auch diese Datei wird für den Bootprozeß benötigt. Alle drei sind unter `$SOURCE/bootimage` vorhanden und lassen sich im Notfall auch über `buildroot` neu erzeugen.

Als erstes wird die Partition des Bootmanagers mit Dateien gefüllt.

```
mount /dev/sda3 /mnt
cp $SOURCE/bootimage/{stage1,stage2,memdisk,realtek,etherboot} /mnt
cp $CONFIG/grubmenu /mnt/menu.lst
umount /mnt
```

Der `grub` auf dieser Partition muß noch aktiviert werden, was mit der `grub`-Shell auf dem Host passiert. Die Parameter des install-Befehls geben die Positionen der stage-Dateien an, bestimmen in welche Partition die Installation erfolgen soll, und wo `grub` später seine Konfigurationsdatei findet. Der Umweg über die Datei `device.map` erlaubt es, alle Partitionen so zu spezifizieren, wie sie später im PC104-Rechner heißen werden.

```
echo "(hd0) /dev/sda" > $TMP/device.map
grub --device-map $TMP/device.map
install (hd0,2)/stage1 (hd0,2) (hd0,2)/stage2 (hd0,2)/menu.lst
quit
```

Das vorbereitete Linux für den Betrieb ohne Netzwerk wird in seine Partition installiert.

```
mount -t ext2 /dev/sda1 /mnt
tar -C /mnt -xf $TMP/buildroot/rootfs-standalone.i386.tar
umount /mnt
```

Nach dem Einbau der Platte ist der Rechner bootfähig. Für die Installation weiterer Flashdiscs empfiehlt es sich, die Festplatte einfach zu kopieren statt alle Schritte nochmals auszuführen.

```
dd if=/dev/sda of=$TMP/flashdisc.img bs=65536 count=496
dd if=$TMP/flashdisc.img of=/dev/sda bs=65536 count=496
```

C.1.7 Vorbereitung der Clients

Der Zusammenbau der PC104-Rechner beginnt mit der Konfiguration des CAN-Controllers auf I/O-Adresse 0x240 und IRQ 10. Auf die Platine kommt das CPU-Board, in dessen Sockel die vorbereitete Flashdisc gesteckt wird. Vor dem Einschalten werden noch Tastatur und Monitor angeschlossen, der Rechner startet beim Einstecken der 5 Volt-Spannungsversorgung. Zunächst werden im BIOS einige Einstellungen durchgeführt oder überprüft. Die Festplatte sollte fest angemeldet werden, da die automatische Erkennung gelegentlich versagen kann. Es darf kein Diskettenlaufwerk aktiviert sein, das Booten muß auch ohne Tastatur und Monitor möglich sein und die seriellen Schnittstellen müssen aktiviert und als RS232 konfiguriert sein. Der Parallelport sollte im EPP/ECP-Modus arbeiten. Beim Booten von der Flashdisc wird im erscheinenden Menü das Konfigurationsprogramm der Netzwerkkarte aufgerufen. Sie muß auf „jumperless“ mit I/O-Adresse 0x300 und IRQ 5 umgestellt werden, damit etherboot die Hardware richtig ansteuern kann und es unter Linux keine Ressourcenkonflikte gibt.

C.1.8 Installation des Bootservers

Auf einem Server sind alle Dienste installiert, die von den Rechnern im PC104-Cluster benötigt werden. Die Installation erfolgt komplett mit Root-Rechten, die benötigten Dateien sind vorher nach `/tmp` auf den Server zu kopieren. Alle Daten der Clients landen in einer gemeinsamen Ordnerstruktur.

```
CLIENTS=/export/0/home/pc104/clients
```

Die Reste einer vorigen Installation müssen zunächst gelöscht werden. Die Benutzerdaten und die Userdatenbank (`passwd`, `shadow` und `group`) sind gegebenenfalls vorher zu sichern.

```
rm -rf $CLIENTS
```

Die erforderliche Verzeichnisstruktur für die Knoten wird anlegegt.

```
install -d -m755 -groot -oroot $CLIENTS
for NODE in $CLIENTS/node{{0,1}{0,1,2,3,4,5,6,7,8,9},2{0,1,2,3,4}}; do \
  install -d -m755 -groot -oroot $NODE; \
done
install -d -m755 -groot -oroot $CLIENTS/tftpboot
install -d -m755 -groot -oroot $CLIENTS/home
```

Es folgt das Auspacken der Root-Verzeichnisse für die verschiedenen Rechner.

```
for DIR in $CLIENTS/node{{0,1}{0,1,2,3,4,5,6,7,8,9},2{0,1,2,3,4}}; do \
  tar -C $DIR -xf /tmp/rootfs-nfsclient.i386.tar; \
done
```

Im Dateisystem jedes Rechners wird ein Swapfile von 8 MB angelegt und initialisiert.

```
for DIR in $CLIENTS/node{{0,1}{0,1,2,3,4,5,6,7,8,9},2{0,1,2,3,4}}; do \
  dd if=/dev/zero of=$DIR/var/swapfile bs=1024k count=8; \
  mkswap $DIR/var/swapfile; \
done
```

Etherboot kann nur Systeme starten, deren Kernel und gegebenenfalls initrd in einem speziellen Image zusammengefaßt sind. Der folgende Befehl erzeugt das Image direkt im Bootverzeichnis.

```
mkelf-linux --output $CLIENTS/tftpboot/pc104.nb --ip=dhcp \
  --append "root=/dev/nfs" /tmp/bzImage
```

DHCP-Server

Der DHCP-Server teilt jedem Knoten basierend auf seiner MAC-Adresse die Konfiguration, den Pfad zum Bootimage und die Parameter zum Mounten des Root-Dateisystems mit. Es handelt sich um einen `dhcpd3`, der wie folgt konfiguriert ist.

```
option domain-name "rtsys";
option domain-name-servers jitter.rtsys;
ddns-update-style none;
ddns-updates off;
authoritative;
server-name "gate2";
shared-network railway-net
{
  option ntp-servers time.rtsys;
  use-host-decl-names true;
  subnet 10.6.2.128 netmask 255.255.255.128
  {
    option routers gate2-1.rtsys;
    option broadcast-address 10.6.2.255;
    filename "/pc104.nb";
    # Knoten 0
    host node00 {
      hardware ethernet 00:05:b7:02:9c:05;
      fixed-address node00.rtsys;
      option root-path "10.6.2.129:/export/0/home/pc104/clients/node00,
        v3,tcp,rsize=2048,wsize=2048";
    }
    # ...weitere Einträge für die übrigen Knoten...
  }
}
```

TFTP, NFS und Zeitserver

Die Bootimages werden von dem TFTP-Server `atftp` ausgeliefert, der `$CLIENTS/tftpboot` als Root-Verzeichnis benutzt. Dort liegt in Form der Datei `pc104.nb` ein Linux-Kernel, der von Etherboot geladen werden kann. Die Root-Verzeichnisse der Knoten werden per NFS exportiert.

```
/export/0/home/pc104/clients    10.6.2.128/25(rw,async,no_root_squash)
```

Beim Systemstart gleichen alle PC104-Rechner ihre Uhren per `rdate` mit dem Server ab. Dafür muß in der Datei `/etc/xinetd.d/time-udp` der Dienst aktiviert sein.

C.2 Verwaltung der Accounts

Wenn ein Benutzer neue Software zur Steuerung der Bahn geschrieben hat, muß diese möglichst einfach auf die PC104-Rechner verteilt werden. Der eleganteste Weg hierfür führt über ein Austauschverzeichnis, das auf den Workstations des Lehrstuhl direkt erreichbar ist und auch von allen PC104-Rechnern gemountet wird. Hierin gibt es für jeden registrierten Benutzer ein Verzeichnis mit dessen Namen und entsprechenden Rechten, auf den eingebetteten Systemen entspricht das Austauschverzeichnis `/home`. Die notwendige Voraussetzung für dieses Verfahren ist, daß die Benutzer in allen beteiligten Systemen gleich registriert sind.

C.2.1 Neuen Benutzer anlegen

Die folgenden Schritte müssen als `root` auf dem Server ausgeführt werden. Sie richten auf allen PC104-Knoten einen Benutzer ein, der im Lehrstuhlnetz bereits bekannt sein muß.

```
ACCOUNT=pc104
CLIENTS=/export/0/home/pc104/clients
ANAME=‘id -nu $ACCOUNT‘
AGROUP=‘id -ng $ACCOUNT‘
AUID=‘id -u $ACCOUNT‘
AGID=‘id -g $ACCOUNT‘
```

Zunächst wird ein leeres Homeverzeichnis mit den entsprechenden Berechtigungen angelegt.

```
rm -rf $CLIENTS/home/$ANAME
install -d -m755 -o$ANAME -g$AGROUP $CLIENTS/home/$ANAME
```

Die Datei `.rhosts` erlaubt einen Login ohne Paßwort von definierten Accounts.

```
echo $ANAME |  \
  awk ’{ for (i=0; i<25; i++) printf("node%02i %s\n",i,$0);
       printf("knopf %s\nlukas %s\n",$0,$0); }’ \
  > $CLIENTS/home/$ANAME/.rhosts
chmod 600 $CLIENTS/home/$ANAME/.rhosts
chown $ANAME.$AGROUP $CLIENTS/home/$ANAME/.rhosts
```

Zum Schluß wird der Benutzer in die Datenbanken der PC104-Rechner eingetragen.

```
for FILE in $CLIENTS/node*/etc/passwd; do \
  echo $ANAME:x:$AUID:$AGID:$ACCOUNT user:/home/$ANAME:/bin/sh >> $FILE; \
done
for FILE in $CLIENTS/node*/etc/group; do \
  echo $AGROUP:x:$AGID: >> $FILE; \
done
for FILE in $CLIENTS/node*/etc/shadow; do \
  echo $ANAME:*:10933:0:99999:7::: >> $FILE; \
done
```

Der so angemeldete Benutzer kann ab jetzt auf sein Verzeichnis im Lehrstuhlnetzwerk unter `/home/pc104/clients/home/<name>` oder auf den PC104-Rechnern direkt über `/home/<name>` zugreifen. Ein Login ist per `rsh` von den Rechnern Knopf und Lukas aus möglich.

C.2.2 Vorhandenen Benutzer löschen

Nicht mehr benötigte Accounts können einfach wieder aus den Datenbanken der PC104-Rechner entfernt werden. Auch diese Schritte müssen als root auf dem Server durchgeführt werden.

```
ACCOUNT=pc104
CLIENTS=/export/0/home/pc104/clients
ANAME=‘id -nu $ACCOUNT‘
```

Zuerst werden die Einträge in den Benutzerdatenbanken der Systeme gelöscht.

```
sed -i -e ’/^’$ANAME’:/d;’ $CLIENTS/node*/etc/{passwd,shadow,group}
```

Zum Schluß wird das Homeverzeichnis des Benutzers entfernt.

```
rm -rf $CLIENTS/home/$ANAME
```

C.3 Benutzung des Crosscompilers

Der von `buildroot` erzeugte Crosscompiler kann benutzt werden, um weitere Programme für die PC104-Rechner zu übersetzen. Dazu müssen nur die richtigen Tools aufgerufen werden, vor `gcc` und ähnliche Befehle muß das Prefix `/home/pc104/toolchain/i386-linux-uclibc-` gestellt werden. Der folgende Block kann die Basis für ein entsprechendes Makefile sein.

```
#
# Sample Makefile
#
# Allows cross compilation for PC104 hosts.
#
# (sho) September 2005
#
```

```
TOOLPREFIX=/home/pc104/toolchain/bin/i386-linux-uclibc-
CC=$(TOOLPREFIX)gcc
CFLAGS=-Os

BINARIES=$(patsubst %.c,%,$(wildcard *.c))

.PHONY: all clean

all: $(BINARIES)

clean:
        rm -f $(BINARIES)
        rm -f *~

%:%.c
        $(CC) $(CFLAGS) -o $@ $<
```

Die Header-Dateien und Libraries von Peak System sind sowohl im Crosscompiler als auch auf den PC104-Rechnern installiert. Damit reicht der Parameter `-lpcan` aus, um Programme zu übersetzen, die das CAN-Interface ansteuern wollen.

C.4 Installation des Railway-Daemons

Eines der wichtigsten Programme auf den PC104-Rechnern ist `railwayd`, ein Serverdienst für alle Applikationen, die auf der `librailway` basieren. Es leitet Befehlspakete von dem CAN-Interface oder einem UDP-Port an die seriellen Schnittstellen und schickt Antworten an den zuständigen Rechner. Das Programm wird zunächst mit dem Crosscompiler übersetzt.

```
make TARGET=pc104 clean railwayd
```

Die Installation erfolgt mit der Identität des Benutzers `pc104`, der wie oben beschrieben auf allen Knoten eingerichtet sein muß.

```
PHOME=/home/pc104/clients/home/pc104
mkdir -p $PHOME/bin
cp railwayd $PHOME/bin
```

Bei dieser Gelegenheit können auch die CAN-Testprogramme installiert werden.

```
cp $TMP/peak-linux-driver-3.17/test/{bitrate,receive,transmi}test $PHOME/bin
```

Ein Bootskript übernimmt die Aufgabe, den daemon bei jedem Systemstart zu laden. Dieser Schritt wird wieder als `root` auf dem Server ausgeführt.

```
for DIR in $CLIENTS/node{{0,1}{0,1,2,3,4,5,6,7,8,9},2{0,1,2,3,4}}; do \
  cp $SOURCE/rootfs/S60railwayd $DIR/etc/init.d; \
done
```

Nach einem Neustart sind die Systeme voll einsatzbereit.

C.5 CAN-Unterstützung für externe Rechner

Die am Bedienpult der Anlage stehenden Rechner sind mit jeweils einem CAN-Bus verbunden. Dazu werden USB-Dongles von Peak System benutzt.

Abbildung C.3: CAN USB-Dongle (Produktfoto von Peak System)

Die Dongles werden von einem Standardkernel nicht unterstützt, daher muß zuerst das Modul des Herstellers passend zum laufenden Kernel übersetzt werden. Die folgende Anleitung bezieht sich auf SuSE, die Pakete `kernel-source` und `kernel-syms` müssen installiert sein. Zunächst werden als root einige vorbereitende Befehle ausgeführt.

```
cd /usr/src/linux
make cloneconfig
make modules_prepare
```

Das Compilieren des Moduls verläuft mit Hilfe der beteiligten Makefiles problemlos.

```
make -C $TMP/peak-linux-driver-3.17 clean
cd $TMP/peak-linux-driver-3.17/driver
make -C /usr/src/linux M=$PWD
```

Nach der Installation des Moduls im System ist die Hardware einsatzbereit.

```
cd $TMP/peak-linux-driver-3.17/driver
make -C /usr/src/linux M=$(pwd) modules_install
depmod -a
```

Das Modul wird automatisch aktiv, sobald das Dongle eingesteckt wird. Alternativ kann es auch per `modprobe pcan` manuell geladen werden. Der Compilierprozeß muß allerdings wiederholt werden, sobald eine neuere Kernelversion auf dem Rechner installiert wird.

Wenn eigene Programme das CAN-Interface nutzen sollen, müssen die Include-Dateien und die vorcompilierte Library in die Suchpfade des Compilers eingefügt werden. Beispiele hierfür finden sich bei den Testprogrammen von Peak oder in dem C-Interface aus dieser Arbeit, für einfache Experimente reicht aber ein Aufruf wie der folgende aus.

```
gcc -o demo demo.c -I/home/pc104/libpcan -L/home/pc104/libpcan -lpcan
LD_LIBRARY_PATH=/home/pc104/libpcan ./demo
```

Die vorcompilierte Library wurde dafür in `/home/pc104/libpcan` installiert.

C.6 Quellen

In diesem Kapitel wurden viele Dateien und Quellcodearchive aus den Internet benutzt. Sie befinden sich allesamt auf der beiliegenden DVD, die Originale stammen von den im Folgenden angegebenen Seiten.

Buildroot, uClibc

- `http://buildroot.uclibc.org/`
- `http://buildroot.uclibc.org/downloads/`

Linux-Kernel

- `http://kernel.org/`
- `http://kernel.org/pub/linux/kernel/v2.4/linux-2.4.31.tar.bz2`

CAN Treiber und Library von Peak System

- `http://www.peak-system.com/`
- `http://www.peak-system.com/linux/files/peak-linux-driver.3.17.tar.gz`

FreeDOS und minimales Floppy-Image

- `http://www.freedos.org/`
- `ftp://ftp.ibiblio.org/pub/micro/pc-stuff/freedos/files/distributions/`

Konfigurationsprogramm für Netzwerkchip

- `http://www.realtek.com.tw` (Suche nach RTL8019 oder RSET-8019)

Etherboot

- `http://www.etherboot.org/`
- `http://rom-o-matic.net/`

Anhang D

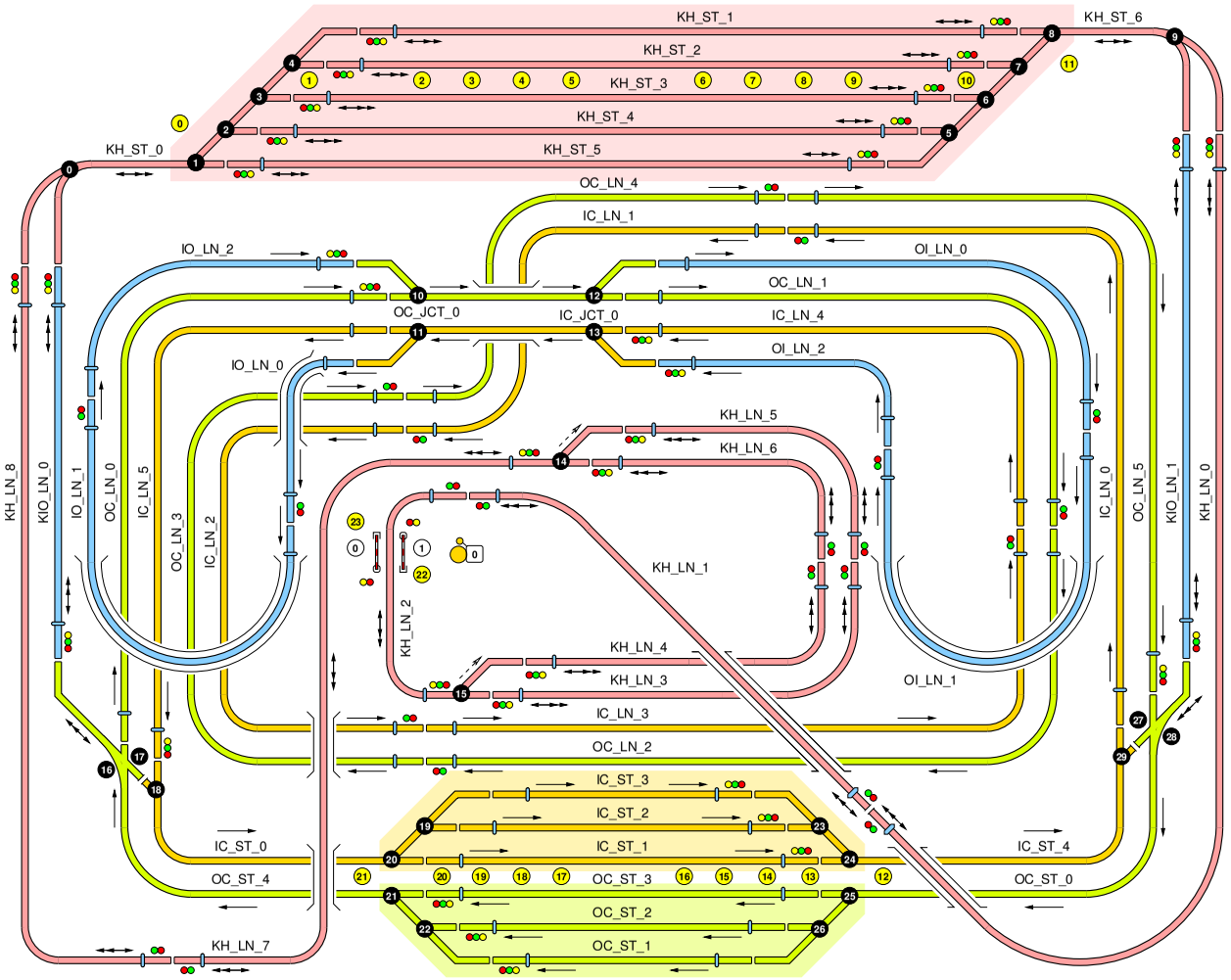
Pläne der Bahnanlage

In diesem Anhang befinden sich die wichtigsten Pläne der Modellbahnanlage, soweit sie für die Ansteuerung und die Erweiterung der Anlage interessant sind.

- **Streckenlayout der Modellbahn:** Dieser Plan gibt die Streckenführung und die Position der gesamten Peripherie möglichst realistisch wieder.
- **Vereinfachtes Streckenlayout:** Für die Streckenplanung eignet sich dieser Plan besser, in dem die zahlreichen Kurven und Brücken etwas entzerrt wurden.
- **Verkabelung der Bussysteme:** In diesem Diagramm sind die Positionen aller Rechner (TTP und PC104), die Bussysteme und die Stromversorgung eingezeichnet.

Model Railway Track Layout

Realtime and Embedded Systems Group



Symbols

Track segments

- Inner Circle
- Outer Circle
- Kicking Horse Pass
- Interconnections

Track specialties

- Bridge
- Point or crossing
- Railroad crossing

Directions

- Unidirectional block
- Bidirectional block with forward direction
- Preferred direction

Electronics

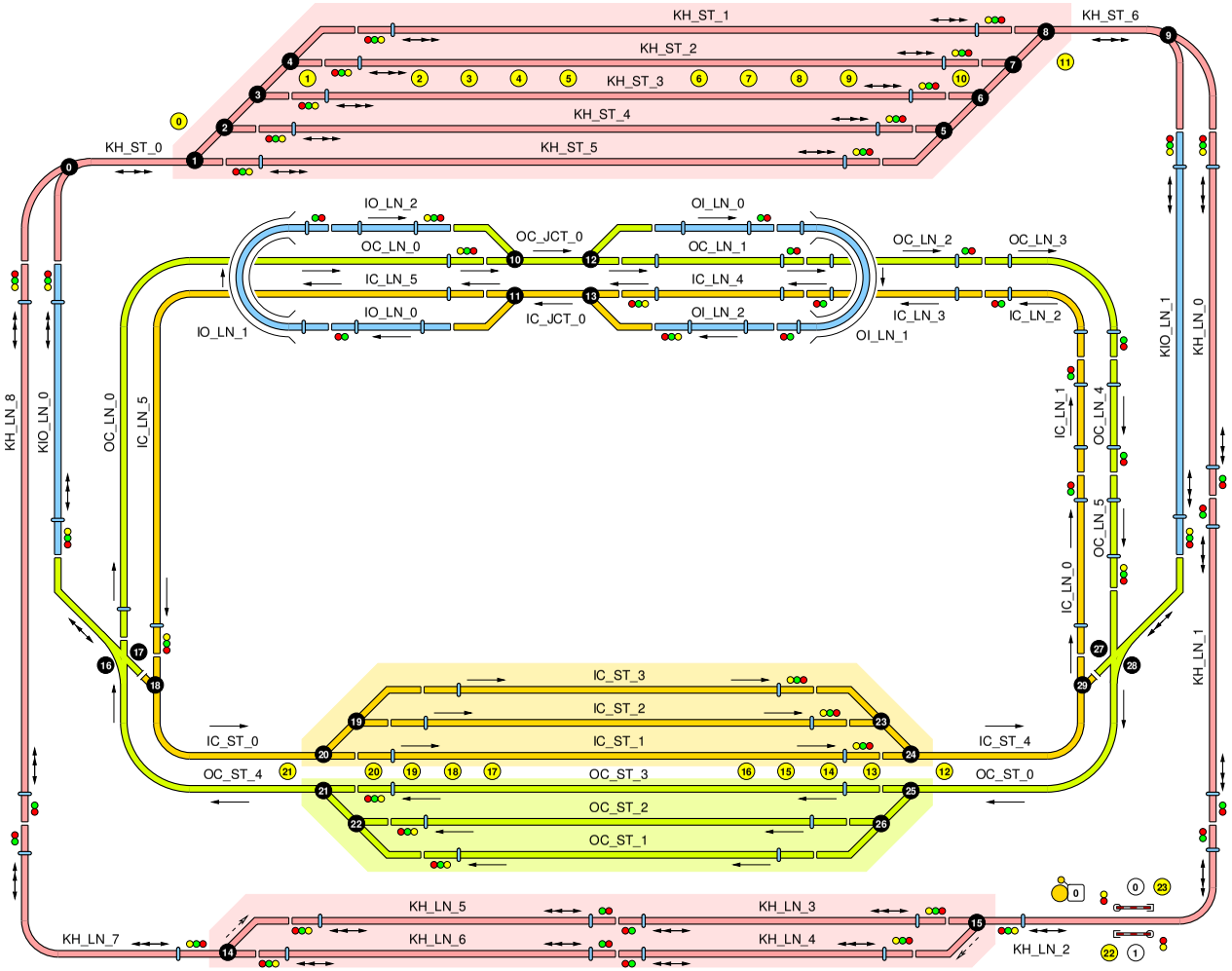
- Block isolation
- Reed contact
- Block signal
- Lighting
- Point operating unit

Stephan Höhmann, January 2006
http://www.informatik.uni-kiel.de/~railway/

Abbildung D.1: Streckenlayout der Modellbahn

Model Railway Track Scheme

Realtime and Embedded Systems Group



Symbols

Track segments

- Inner Circle
- Outer Circle
- Kicking Horse Pass
- Interconnections

Track specialties

- Bridge
- Point or crossing
- Railroad crossing

Directions

- Unidirectional block
- Bidirectional block with forward direction
- Preferred direction

Electronics

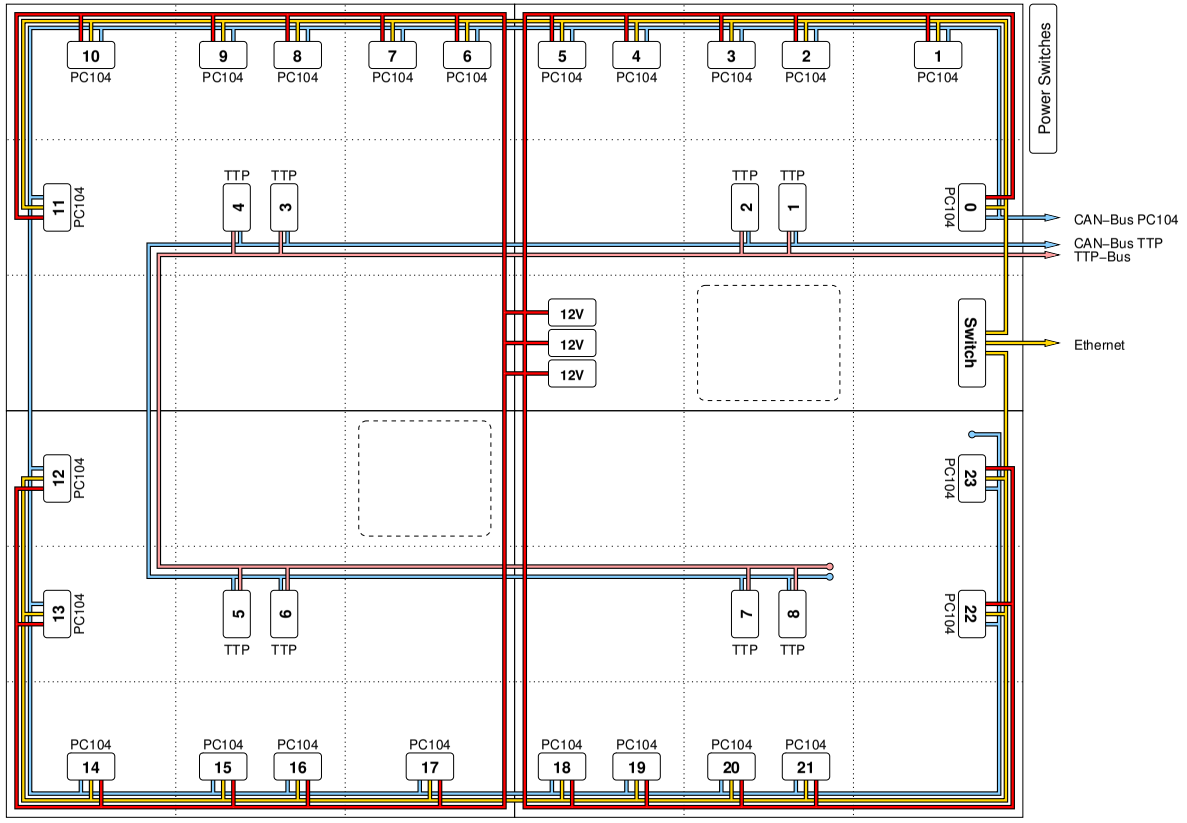
- Block isolation
- Reed contact
- Block signal
- Lighting
- Point operating unit

Stephan Höhmann, January 2006
http://www.informatik.uni-kiel.de/~railway/

Abbildung D.2: Vereinfachtes Streckenlayout der Modellbahn

Model Railway Bus Diagram

Realtime and Embedded Systems Group



Symbols

Networks

- 12 V Power Supply
- PC104 Ethernet subnet
- TTP Bus
- CAN Bus Systems

Special network elements

- Bus Termination
- External Connector

Controlling nodes

- PC104 — PC104 Node + Controller
- TTP — TTP Powernode 312

Supply System

- 12V — Railway Power Supply
- Switch — Ethernet Switches
- Power — Main Power Switches

Stephan Höhrmann, January 2006
http://www.informatik.uni-kiel.de/~railway/

Abbildung D.3: Verkabelung der Bussysteme unter der Modellbahn

Literaturverzeichnis

Modellbahnanlage

[Busch] Homepage der Firma Busch (Bahnelektronik)
`http://www.busch-model.com/`

[Kicking] Hintergrundinformationen zum Kicking Horse Pass
Wikipedia: `http://de.wikipedia.org/wiki/Kicking_Horse_Pass`
Encyclopedia Britannica: `http://www.eb.com/`
Quellen verschiedener Fotos
John Cletheroe: `http://freespace.virgin.net/john.cletheroe/`
Brian Keay: `http://www.acs.ucalgary.ca/~keay/hobbies.html`

[Prakt06] Abschlußbericht des Modellbahnpraktikums im Wintersemester 2005/06
Komplett auf `http://www.informatik.uni-kiel.de/inf/von-Hanxleden/teaching/ws05-06/p-bahn/proceedings.pdf`

[Railway] Information über die Anlage innerhalb der Informatik
`http://www.informatik.uni-kiel.de/inf/Kluge/trains/`
`http://www.informatik.uni-kiel.de/~railway/`

[Roco] Homepage der Firma Roco (Gleismaterial)
`http://www.roco.co.at/`

[TMRC] The Tech Model Railway Club at MIT
`http://tmrc.mit.edu/`
Der Club ermöglicht es Studenten des MIT seit 1946, an einer großen Modellbahnanlage mitzuarbeiten.

Serielle Kommunikation

[Hardwarebook] Joakim Ögren: The Hardware Book.
`http://www.hardwarebook.net/connector/index.html`
Umfangreiche Sammlung, Beschreibungen sehr vieler gängiger Schnittstellen.

[RS232] HW-Server:
`http://www.hw-server.com/rs232`
Praxisnahe und umfassende Beschreibung der seriellen Schnittstelle.

[Strangio] Christopher E. Strangio: The RS232 Standard. A Tutorial with Signal Names and Definitions, CAMI Research Inc., Lexington, Massachusetts.
`http://www.camiresearch.com/Data_Com_Basics/RS232_standard.html`
Gute und verständliche Beschreibung des RS232/EIA232 Standards.

Verteilte Systeme und Feldbusse

[CiA] CAN in Automation (CiA), Industriegremium zur Unterstützung von CAN
`http://www.can-cia.org/can/`

[Kopetz] Herman Kopetz: A Comparison of CAN and TTP. Proceedings of the IFAC Distributed Computer Systems Workshop, Como, Italy 1998.

[Santner] Guido Santner, Max Felser, Hans Scheitlin: Feldbusse ersetzen Kabelsalat. Teil drei der Artikelserie Automation im Bulletin SEV/VSE 07/05.
Der Artikel stellt alle gängigen Feldbusse gegenüber und beleuchtet auch die historische Entwicklung und gemeinsame Grundlagen.

Elektronik und Systemprogrammierung

[ElKo] Patrick Schnabel: Das Elektronik-Kompendium.
`http://www.elektronik-kompendium.de/`

[HorHill] Paul Horowitz, Winfield Hill: The Art of Electronics (second edition). Cambridge University Press, 1989.
Standardreferenz der Elektronik.

[Messmer] Hans-Peter Messmer: PC-Hardwarebuch. Aufbau, Funktionsweise, Programmierung, Ein Handbuch nicht nur für Profis (3. Auflage). Addison-Wesley 1995.

[Tischer] Michael Tischer: PC Intern 4 (1. Auflage). Data Becker, 1994.
Referenz zu vielen Standardfunktionen und Interfaces im PC.

Organisation des Fahrbetriebs

[Hielscher] Wolfgang Hielscher, Lars Urbszat, Claus Reinke, Werner Kluge: On Modelling Train Traffic in a Model Train System. Department of Computer Science, University of Kiel, May 1998. Veröffentlicht in K. Jensen, K. (Herausgeber): Proc. Workshop on Practical Use of Coloured Petri-Nets and Design / CPN, University of Aaarhus, Denmark, 1998, pp. 83-101.
Dieses Paper beschreibt den Einsatz gefärbter Petrinetze zur Steuerung des Fahrbetriebs auf einer Modellbahn. Die angegebenen Netze beziehen sich auf die erste Version der Anlage.

[Kluge] Werner Kluge: The Kicking Horse Pass Problem. Petri Net News Letters No. 54, (1998), pp. 3-15.
Verschiedene Petrinetze werden hier vorgestellt und erklärt, die alle Aspekte in der Steuerung des Kicking Horse Passes abdecken.

[Koberstein] Jochen Koberstein: Realisierung eines geordneten Mehrzugbetriebs auf einer Modellbahnanlage. Diplomarbeit am Institut für Informatik und Praktische Mathematik, Christian-Albrechts-Universität zu Kiel.
Diese Arbeit ist der Abschlußbericht zur zweiten Überarbeitung der Anlage. Ihr Schwerpunkt liegt auf dem Umsetzen eines Mehrzugbetriebs nach einem vorgegebenen Fahrplan, unter anderem auf der Grundlage von Petrinetzen.

Mikrocontroller und ihre Programmierung

[Bredendiek] Jörg Bredendieks Homepage: `http://www.sprut.de/`
Informationen zu PICs, Programmierung, und der Ansteuerung von Peripherie.

[Gputils] GNU PIC Utilities project homepage: `http://gputils.sourceforge.net/`

[Höhrmann] Stephan Höhrmann: Entwicklung eines Ultraschall-basierten Ortungssystems für Lego Mindstorms Roboter. Studienarbeit am Institut für Informatik, Christian-Albrechts-Universität zu Kiel, 1. März 2005.
In dieser Arbeit wird neben dem Ortungssystem auch die Programmierung von PIC Mikrocontrollern und die Platinenherstellung behandelt.

[Microchip] Microchip Technology Inc: `http://www.microchip.com/`

[Piclinks] Allgemeine Informationen, Tutorials, Linksammlungen:
`http://www.voti.nl/swp/`
`http://www.sprut.de/electronic/pic/`
`http://www.mikrocontroller.net/`
`http://www.piclist.com/`
`http://www.geocities.com/SiliconValley/Way/5807/dat.html`

Inhalt der beiliegenden DVD

Dieser Arbeit liegt eine DVD bei, auf der alle Schaltungen, Platinenlayouts, Steuerprogramme, Fotos, Filme und natürlich die Ausarbeitung gespeichert sind. Dazu kommen viele Quellen und Programme, die zitiert oder anderweitig verwendet wurden.

- **Ausarbeitung** - Quellcode der Ausarbeitung mit allen Abbildungen
- **Bildmaterial** - Verschiedene Fotos und Videos der Anlage
- **Gleisplan** - Pläne der Anlage im xfig-Format
- **Konfiguration** - Beschreibungen der Bahnanlage im csv-Format
- **Material** - Verwendete Materialien aus unterschiedlichen Quellen
  - **Ausarbeitung** - Alle Fotos, die für die Ausarbeitung aufbereitet wurden
  - **Datasheets** - Datenblätter und Application Notes aller verwendeten Bauteile
  - **Eagle** - Bibliotheken und Skripte für Eagle
  - **Etiketten** - Vorlagen für alle Aufkleber (Dymo Labelwriter)
  - **Literatur** - Einige zitierte Veröffentlichungen und Artikel
  - **Manuals** - Einige Handbücher und Dokumentationen zu den benutzen Geräten
  - **Software** - Verwendete Programme von externen Quellen
  - **Tools** - Kleinere Hilfsprogramme und Erweiterungen
- **Messwerte** - Excel-Tabellen mit verschiedenen Meßwerten und Diagrammen
- **PC104** - Konfiguration und Quellcodearchive für die PC104-Rechner
- **Programmcode** - Code und Tools zur Programmierung der Bahn
  - **Firmware** - Assembler-Code für die Leistungselektroniken und Schrankensensoren
  - **Interface** - Bibliotheken und Beispiele für die Programmierung in C
  - **Kicking** - Die alte Software der Anlage, Railway Control Center
  - **Programmer** - Werkzeuge zum Brennen der Firmware
  - **Simulator** - Grundgerüst für einen universell einsetzbaren Bahnsimulator
- **Schaltungen** - Eagle-Dateien aller Schaltungen und Platinenlayouts
- **Vortrag** - Die Folien von den drei Präsentationen der Arbeit

Erklärung

Hiermit versichere ich, daß ich die vorliegende Arbeit selbständig verfaßt und ausschließlich die angegebenen Hilfsmittel und Quellen verwendet habe.

Kiel, den _____ _____