Carrera Projekt: Team 42

Protokoll

HTWG Konstanz	**SS 08**
Jens Spinner	279215
Dominik Gauggel	279118
Julian Hirt	279316
Benjamin Kleinhens	279172
Anja Kellner	279167

Zusammenfassung

Im Rahmen der Studiengänge „Software Engineering“ und „Technische Informatik“ der HTWG-Konstanz wurde ein Teamprojekt veranstaltet. Dieses Teamprojekt befasste sich mit einer Machbarkeitsstudie über die Ansteuerung einer Carrera Bahn. Von insgesamt dreizehn Teammitgliedern befasste sich die Gruppe 42, bestehend aus Benjamin Kleinhens, Julian Hirt, Dominik Gauggel, Anja Kellner und Jens Spinner, mit der Verwendung und Erweiterung des Carrera Digital Systems.

- Das Carrera System wurde analysiert.
- Es wurde eine Ansteuerung realisiert
- Es wurde eine Sensorik für die Positionserkennung geschaffen.
- Es wurde eine Machbarkeitsprüfung für einen Rückkanal durchgeführt.

Inhaltsverzeichnis

1 Aufgabenstellung

Das Carrera-Projekt beschäftigt sich mit der Digital 132 – Carrera-Bahn. Vorgegeben waren drei Aufgabenstellungen:

- Untersuchung der Möglichkeiten zur Erfassung des Standortes von Fahrzeugen auf der Rennbahn.
- Entwicklung oder Adaption einer elektronischen Steuerung für eine digitale Rennbahn.
- Entwicklung einer Softwaresteuerung für einen Wagen mit dem Ziel, ein "Rennen" zwischen allen beteiligten Wagen zu gewinnen.

Diese Aufgaben dienten als Anhaltspunkte für das Carrera-Projekt.
Zunächst stand eine gemeinsame Recherche aller Carrera-Projekt-Team Mitglieder. In dieser Phase sollten mögliche Aufgaben für Kleingruppen erarbeitet werden.
Dieses Dokument beschreibt das Vorgehen und die Aufgaben unserer Kleingruppe, die sich mit dem Carrera-Protokoll beschäftigt hat. Unser Team heißt „42“. Mit diesem Namen erhoffen wir uns den Sinn des Lebens oder vielleicht auch einfach nur die Welt des Carrera Protokolls zu entdecken.
Wir beschäftigten uns damit, das Carrera-Protokoll zu entschlüsseln, es nachzubilden und vom PC auf die Bahn zu senden. Somit sollten die Autos über den PC anzusteuern sein. Zusätzlich wollten wir feststellen, ob es möglich ist, einen Rückkanal einzurichten. Dieser soll dazu dienen, Daten vom Auto oder der Bahn an den PC zurückzusenden.

2 Über dieses Dokument

Im ersten Teil dieses Dokuments beschreiben wir die allgemeinen technischen Details der Carrera-Bahn. Hier wird z.B. auf die Kommunikationsmethode zwischen Carrera-Box und der Bahn eingegangen. Es wird erläutert, welche Protokolle wie versendet werden.
Im nächsten Kapitel kommt eine Beschreibung der wichtigsten Schaltung unseres Projektes. Diese Schaltung blieb uns von den ersten Ansteuerungsversuchen bis hin zur endgültigen Version erhalten, bzw. wurde im Laufe des Projektes noch ein wenig angepasst.
Daraufhin gehen wir auf die von uns gewählten Ansätze zur Ansteuerung der Bahn mit dem PC ein. Hier werden unsere 2 grundlegenden Ansätze detailliert erläutert. Es wird auf die verwendete Hardware, die benötigten Schaltungen und auch auf den Aufbau der Software genauer eingegangen.
Im weiteren Verlauf dieses Dokumentes wird auf unseren Bahnlogger genauer eingegangen. Dieser diente vor allem dem Auslesen der Daten auf der Bahn.
Danach wird die Hard- und Software unserer Positionserkennung genauer erläutert.
Im nächten Kapitel gehen wir auf die Machbarkeit eines Rückkanals genauer eingegangen.
Das letzte Kapitel dient als Fazit über das gesamte Projekt. Es beschreibt in kurzen Sätzen, was wir währen diesem Projekt erreicht haben.
Im Anhang ist unter anderem eine Zeitachse zu finden, die den Projektablauf grob in Bildern wiederspiegelt.

3 Technische Systembeschreibung

Zu Beginn des Projektes mussten wir erst einmal einige Informationen zur Funktionsweise der Carrera-Bahn ermitteln. Zu den folgenden Themen haben wir auf verschiedenen Wegen Informationen benötigt und gefunden:

- Technische Fakten
- Semantik des Carrera-Protokolls
- Funktionsweise der Weichen
- Onboard Unit der Autos
- Geschwindigkeitsmessung
- Analyse der Black-Box von Carrera

3.1 Fakten

Um die Ansteuerung der Autos zu verstehen, müssen zunächst einmal die grundlegenden technischen Details klar sein. Hierzu wurde im Internet nach Informationen zur Carrera Bahn „Digital 132“ gesucht und auch selbst nachgemessen. Die Seite [SLOTB] diente als Basis der ermittelten Ergebnisse.

Bei der Digital 132 –Bahn handelt es sich um ein 2-Leiter-System. An der Strecke liegt eine konstante Spannung von 14,7 V an, die den Fahrzeugen als Versorgungsspannung dient. Die Geschwindigkeit wird den Autos über Datenpakete mitgeteilt. Diese werden über die Bahn an die Autos versendet. Die Versorgungsspannung wird alle 7,5 ms für die Übermittlung von Datenpakten unterbrochen. In diesem Intervall werden durch das Ein-und Ausschalten der Versorgungspannung Informationen an die Autos übermittelt. Eine detailliertere Beschreibung ist in dem Kapitel „3.2 Semantik des Carrera-Protokolls“ zu finden. Es werden immer 10 Datenworte in derselben Reihenfolge gesendet. Wurde dieser Block einmal komplett ausgesendet, wird der Block wieder von vorne begonnen. Jedes einzelne Datenwort wird somit alle 75 ms erneut ausgesendet. Dies entspricht einer Frequenz von 13Hz. Die Datenworte werden seriell auf die Bahn geschrieben und müssen Manchestercodiert anliegen. Beim Manchestercode wird die logische 1 durch eine fallende Flanke (Wechsel von 1 auf 0) und die logische 0 durch eine steigende Flanke (Wechsel von 0 auf 1) beschrieben. Die Wechsel zwischen den einzelnen Bits beim Manchestercode finden immer gleichzeitig mit einer Taktflanke statt. In Abbildung 1 ist dieser Zusammenhang noch einmal graphisch dargestellt.

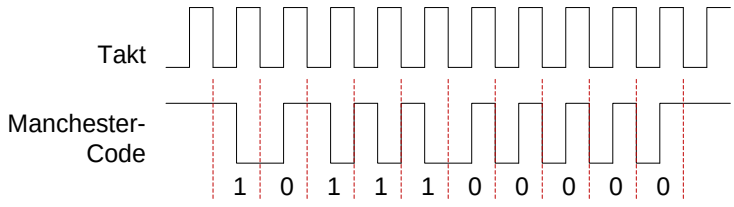


Abbildung 1: Zusammenhang Takt --> Manchestercode

Die Baudrate während dem Senden eines Datenpaktes beträgt nominal 10 kBaud. D.h. die Bits wechseln alle 100 µs.

3.2 Semantik des Carrera-Protokolls

Um die Autos tatsächlich ansteuern zu können, wird noch der genaue Aufbau des Protokolls benötigt. Auf der schon bei den „Fakten“ erwähnten Homepage ([SLOTB]) haben wir eine gute Beschreibung des Protokolls gefunden. Der Aufbau der Datenworte wird im nächsten Unterkapitel genauer erläutert.

Unsere Hauptaufgabe bei der Protokollanalyse bestand darin, die Annahmen im Internet auf ihre Korrektheit zu prüfen. Hierfür nutzten wir folgende Möglichkeiten:

1. Auslesen der Daten auf der Bahn:
   Es werden Signale von der Carrera-Box auf die Bahn gesendet. Diese Signale werden mit dem PC aufgezeichnet und können somit analysiert werden.
2. Einfache Software:
   Ein simples Programm, welches die gewünschten Datenworte schon als Manchestercode direkt auf die Bahn aussendet. Es wird geprüft, ob sich die Autos wie gewünscht verhalten.

Allerdings sind diese Lösungen nicht so zuverlässig, da nicht exakt mit dem Takt gearbeitet wird. Im weiteren Verlauf des Projekts entwickelten wir einen „Bahnlogger“, welcher die Daten auf der Bahn aufzeichnet (siehe Kapitel „6 Bahnlogger“). Dieser orientiert sich exakt am Takt, wodurch wir zuverlässige Ergebnisse erhalten.

3.2.1 Protokollaufbau

Prinzipiell gibt es folgende vier Datenworte, die auf die Bahn gesendet werden:

- Programmierdatenwort
- Pace- und Ghost-Car-Datenwort
- Aktivdatenwort
- Reglerdatenwort

Die Datenworte haben nicht alle die gleiche Länge. Es existieren Datenworte mit 8, 10 und 13 Datenbits. Das erste Bit jedes Datenworts ist das sogenannte Startbit. Es gibt an, dass jetzt eine, für die Fahrzeuge relevante, Nachricht auf der Bahn anliegt. Ein Stoppbit, welches das Ende der Nachricht angibt, gibt es nicht. Das Ende einer Nachricht wird von den Autos festgestellt, wenn die Signale auf der Bahn gegen den Manchestercode verstoßen. Dieser Verstoß wird durch eine physikalische 1, also durch eine konstante Versorgungsspannung, repräsentiert. Ein Verstoß durch eine physikalische 0 hat eine andere Bedeutung, wie wir im weiteren Verlauf unserer Arbeit noch feststellen werden. Die Informationen in den Datenworten sind teilweise als MSB (most significant bit) oder LSB (least significant bit) codiert.

Wie im Kapitel „3.1 Fakten“ bereits erwähnt, werden 10 Datenworte immer in der gleichen Reihenfolge zyklisch gesendet.

Die Reihenfolge der Datenworte sieht wie folgt aus:

- Programmierdatenwort
- Pace- und Ghost-Car-Datenwort
- Aktivdatenwort
- Reglerdatenwort 0
- Reglerdatenwort 4
- Reglerdatenwort 1
- Reglerdatenwort 5
- Reglerdatenwort 2
- Aktivdatenwort
- Reglerdatenwort 3

3.2.1.1 Programmierdatenwort:

Dieses Datenwort dient der Programmierung eines Autos. Mit diesem Datenwort kann die maximale Geschwindigkeit, die Bremswirkung oder die Tankfüllung eines Autos angegeben werden. Über die Parameter kann somit die Geschwindigkeit, skaliert werden.

Aufbau des Protokolls:

Protokollbits	**Funktion**
1	Startbit
W0 W1 W2 W3	Gibt den Wert (0-15) für den ausgewählten Parameter an. LSB
P0 P1 P2	Wählt den Parameter aus: 0 0 0 Geschwindigkeit 1 0 0 Bremse 0 1 0 Tankfüllung 0 0 1 Keine Programmierung
0 0	Vermutlich weitere Parameterbits
R0 R1 R2	ID des Autos (0-5). LSB

Tabelle 1: Aufbau des Programmierdatenworts

3.2.1.2 Pace- und Ghost-Car-Datenwort

Dieses Datenwort dient der Steuerung des Pacecars. Es sieht auf den ersten Blick wie ein Reglerdatenwort für den Regler Nummer 7 aus, enthält allerdings etwas andere Protokollbits.

Aufbau des Protokolls:

Protokollbits	**Funktion**
1	Startbit
1 1 1	Auto mit ID 7. MSB
KFR	Keine Freigabe. Normaler Bahnbetrieb wenn Bit = 0.
TK	Steuert die Weiche per Zufall; Werte: 0 oder 1; hängt vom Ort des Autos ab
FR	Invertierte von KFR
NH	Gibt an, ob das Pacecar in die Box zurück soll. 0 = ja; 1 = nein; Wenn Pacecar nicht aktiv: Wert = 0
PC	Ist Pacecar aktiv: Wert = 1 nachdem Pacecar-Taste gedrückt wurde
TA	Tank. Ist 1 wenn Benzinstand = aktiviert

Tabelle 2: Aufbau des Pace- und Ghost-Car-Datenworts

3.2.1.3 Aktivdatenwort

Dieses Datenwort dient der Kontrolle, für welche Autos gerade die Geschwindigkeitstaste gedrückt wird. Es wird in jedem Zyklus zweimal gesendet. Wir vermuten, dass dieses Datenwort dazu dient, die Bremsreaktion zu verbessern.

Aufbau des Protokolls:

Protokollbits	**Funktion**
1	Startbit
R0	Bit = 1 wenn für Auto 0 Geschwindigkeitstaste gedrückt
R1	Bit = 1 wenn für Auto 1 Geschwindigkeitstaste gedrückt
R2	Bit = 1 wenn für Auto 2 Geschwindigkeitstaste gedrückt
R3	Bit = 1 wenn für Auto 3 Geschwindigkeitstaste gedrückt
R4	Bit = 1 wenn für Auto 4 Geschwindigkeitstaste gedrückt
R5	Bit = 1 wenn für Auto 5 Geschwindigkeitstaste gedrückt
IE	Bit = 1 wenn irgendeine Geschwindigkeitstaste gedrückt ist

Tabelle 3: Aufbau des Aktivdatenworts

3.2.1.4 Reglerdatenwort

Die einzelnen Reglerdatenworte sind vom Aufbau her identisch. Da diese Datenworte die Autos steuern, wird bei diesen Datenworten zwischen den einzelnen Fahrzeugen unterschieden.
Das Datenwort gibt an, wie schnell ein Auto fahren soll, ob die Weiche aktiv sein soll und ob der Benzinstand aktiviert ist. Es werden immer Reglerdatenworte für alle Fahrzeuge gesendet, auch wenn nicht alle auf der Bahn stehen.
Aufbau des Protokolls:

Protokollbits	**Funktion**
1	Startbit
R2 R1 R0	ID des Autos (0-5). MSB
SW	Spurwechsel. Wird 0 wenn Spur gewechselt werden soll
G3 G2 G1 G0	Geschwindigkeit (0-15). MSB
TA	Tank. Ist 1 wenn Benzinstand = aktiviert

Tabelle 4: Aufbau des Reglerdatenworts

3.3 Funktionsweise der Weichen

Beim Betrieb der Digital 132 wird durch das gedrückt Halten des Weichenknopfs angegeben, ob eine Weiche geschalten werden soll oder nicht. In der Bahn ist ein IR-Sensor vor jeder Weiche angebracht. Zudem ist in der Weiche ein ATMEGA8 Prozessor eingebaut. Der Weichensensor empfängt die Daten der Bahn, wertet diese aus, und überprüft die Weichenbits der Autos. Dabei merkt es sich welche Auto ID ein Weichenbit gesetzt hat. Fährt nun ein Auto mit der entsprechenden ID über die Fotodiode, so schaltet es die Weiche ein. Das Auto biegt ab. Beim Überfahren der Weiche stellt das Auto die Weiche mechanisch zurück. Die Weiche selbst scheint verpolungssicher zu sein. Dies wurde während Tests für den Rückkanal festgestellt. In der Weiche sind die beiden Bahnseiten kurzgeschlossen. Jeweils bei den Ein- und Ausfahrten der Weiche ist eine Unterbrechung.

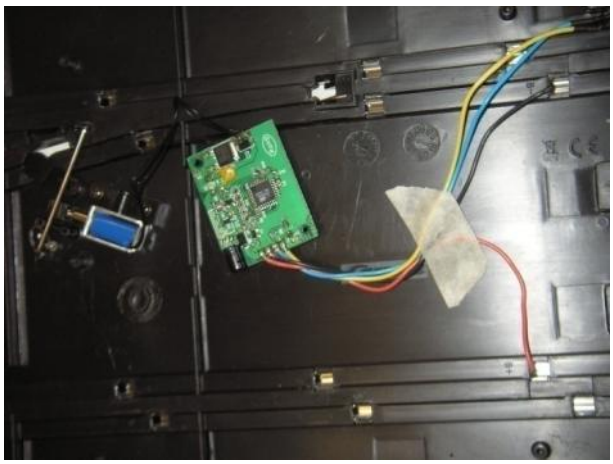


Abbildung 2: Unterseite der Weiche mit Steuerplatine und Mechanik

Abbildung 3: Steuerplatine der Weiche mit ATMEGA-Prozessor

3.4 Onboard Unit der Autos

Jedes Auto enthält einen ATMEGA8-Prozessor. Dieser dient dazu, die Signale von der Bahn zu dekodieren und die Motoren der Autos anzusteuern.

Jedes Auto kann auf eine bestimmte ID programmiert werden. Hierzu muss zunächst die Weichentaste des gewünschten Reglers zweimal gedrückt werden. Danach muss das Auto angehoben und wieder abgesetzt werden. Nachdem jetzt noch einmal die Weichentaste zweimal gedrückt wurde, ist das Auto auf die ID des Reglers programmiert.

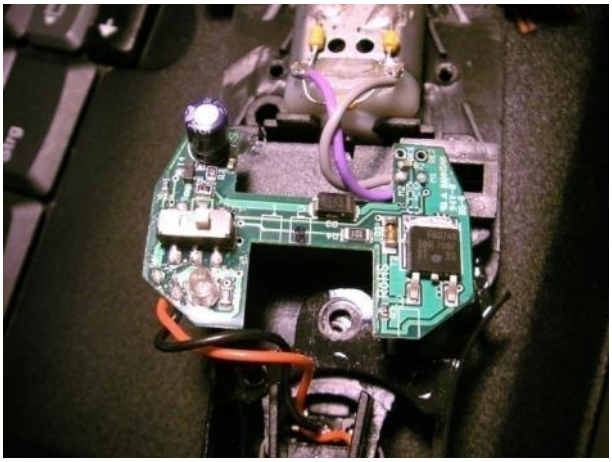


Abbildung 4: Rennauto – Steuerplatine von oben

3.4.1 IR-LED der Fahrzeuge

Die IR-LEDs der Autos werden mit einem Rechteck gepulst. Je nach ID, auf die das Auto programmiert wurde, ist die Periodenlänge unterschiedlich. Anhand der Periodenlänge kann also auf die ID des Autos geschlossen werden.

Das Signal der IR-LEDs verändert sich nicht wenn die Weichentaste gedrückt ist. Diese Auswertung muss von der Weiche übernommen werden.

Auto-ID	Periodenlänge
0	63 µs
1	124 µs
2	190 µs
3	255 µs
4	319 µs
5	383 µs
AutonomousCar	447 µs
PaceCar	509 µs

Tabelle 5: IR-LED Pulslängen

Diese Werte können mit einer Toleranz von ± 2 µs Schwanken

3.4.2 Motor

Der Motor wird per PWM (Pulsweitenmodulation) über einen IRF9024 geregelt. Bei der Pulsweitenmodulation wechselt der Strom zwischen zwei Werten. Dadurch können analoge Systeme (z.B. Motoren) mit digitalen Signalen angesteuert werden. Das PWM-Signal wird vom Prozessor im Auto generiert. Die Periodendauer des Signals liegt konstant bei 60 µs. Die Amplitude liegt bei 14,7 V.

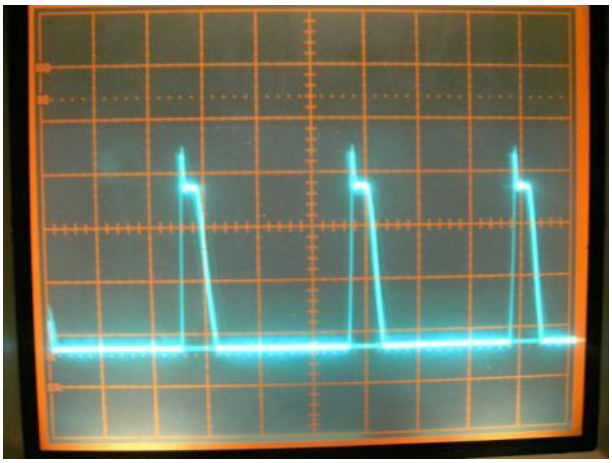


Abbildung 5: PWM-Signal an den Motor

Messungen an der Bahn ergaben vor der Transistorschaltung einen Strombedarf von etwa 400 mA. Aus Sicherheitsgründen sollte mit etwas mehr als 500 mA pro Auto gerechnet werden.

3.5 Geschwindigkeitsmessung

Damit die Autos später eventuell auch automatisiert vom PC angesteuert werden können, haben wir in Kooperation mit der Projektteilgruppe „Blue Riders“ die Geschwindigkeit der Autos gemessen. Wir haben mittlerweile die Erfahrung gemacht, dass nicht jedes Auto bei gleich eingestellter Geschwindigkeit gleich schnell fährt. Es kann daher nicht von der gefahrenen Geschwindigkeit auf die einzustellende Geschwindigkeit im Carrera-Protokoll geschlossen werden.

3.5.1 Verwendete Geräte

- Speicheroszilloskop
- 5 V Netzteil
- Lichtschranken

3.5.2 Aufbau

Die zwei Lichtschranken wurden in die Bahn integriert. Der Abstand der beiden Lichtschranken beträgt 11,5 cm. Die Lichtschranken werden mit 5 V Gleichspannung betrieben und auf die Kanäle 1 und 2 am Oszilloskop gelegt.

3.5.3 Einstellungen

Beim Oszilloskop müssen die Kanäle 1 und 2 invertiert werden. Der Trigger muss als Quelle den Kanal 1 haben und auf „Positive Flanke“ gestellt sein. Die Kopplung des Triggers wird auf „NF reject“ gestellt.

3.5.4 Durchführung

Das Oszilloskop kann auf „single SEQ“ gestellt werden, wenn nur ein Durchlauf gemessen werden soll. Außerdem muss ein geeigneter Messzeitraum eingestellt werden, z.B. M=25 => der Messbereich ist damit 250 ms. Sobald alle Einstellungen korrekt sind, kann das Auto über die Lichtschranke fahren. Wird die erste Lichtschranke ausgelöst (Kanal 1), so wird im Oszilloskop der Trigger gestartet und es wird aufgezeichnet. Wenn beide Pulse erfasst wurden, kann an dem Oszilloskop der Abstand zwischen den beiden positiven Flanken gemessen werden, dieser entspricht der gemessenen Zeit.

3.5.5 Herleitung

Die Geschwindigkeit v ist der Quotient zwischen der Strecke s und der Zeit t. Dabei ist die Strecke s konstant.

$$v = \frac{s}{t} \quad \text{mit } s = 0{,}115\ \text{m} \rightarrow v = \frac{0{,}115}{t}$$

Formel 3–1

3.5.6 Ergebnisse der Messungen

t in [ms]	v in [m/s]	v in [km/h]
65	1,77	6,37
55	2,09	7,53
65	1,77	6,37
25	4,6	16,56
26	4,42	15,92
25	4,6	16,56
29	3,97	14,28
26	4,42	15,92
26	4,42	15,92

Tabelle 6: Geschwindigkeitsmessungen

Aus diesen Messergebnissen kann gefolgert werden, dass die maximale Geschwindigkeit, bei der das Auto nicht aus der Kurve fliegt bei ca. 4,5 m/s liegt. Dies entspricht ca. 16 km/h.
Die Genauigkeit der Messergebnisse könnte noch verbessert werden, indem ein richtiger Zähler anstelle des Oszilloskops verwendet würde.

3.6 Analyse der Black-Box von Carrera

Die Carrera-Blackbox ist wie folgt aufgebaut:

Im Inneren befindet sich ein Atmega16-Prozessor. An diesen werden die analogen Handsteuergeräte angeschlossen. Mindestens ein Ausgang ist an die Treiberschaltung angeschlossen. Der Treiber besteht aus einer Bipolar- und Feldeffekt Transistor Schaltung. Sie wandelt das 5 Volt Signal des Prozessors in 14,7 Volt für die Bahn um. Hierbei wird das Signal invertiert. Der Wiederstand des Geschwindigkeitsreglers liegt zwischen 0 Ohm und 9,2 kOhm (Anschluss: gelb-rot).

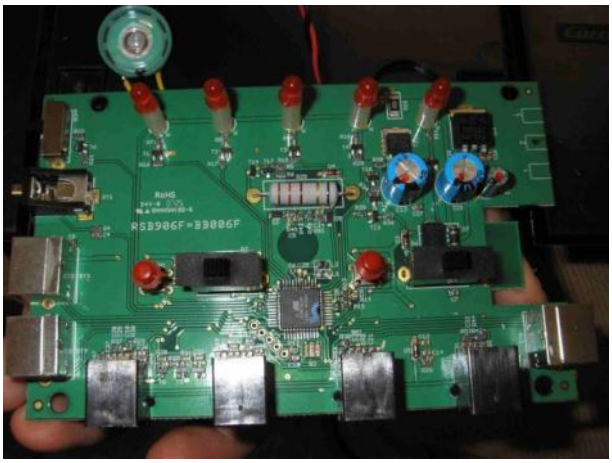


Abbildung 6: Carrera-Blackbox von Innen

3.7 Pitstop-Lane

Die Pitstop-Lane bietet dem Spielbetrieb zwei Funktionalitäten:

- **Tank:** Ist die Tankbefüllung aktiviert, so dient die Pit-Stop-Lane zur Tankbefüllung. Das Auto muss hierfür eine gewisse Zeitspanne auf der Pit-Stop-Lane stehen bleiben (Geschwindigkeit 0). Durch schnelles blinken der Lichter - falls vorhanden - signalisiert das Auto den Tankvorgang. Ist das Auto voll, stoppt das Blinken.
- **Pace Car:** Existiert ein Pace-Car, so ist die Pit-Stop-Lane die Haltestraße. Wird das Pace Car aktiviert, so fährt es auf die Bahn raus, dreht dort einige Runden und kehrt nach der Pace Car Phase in die Pit-Stop-Lane zurück wo es anhält.

Technisch funktioniert dies durch eine teilweise Entkopplung der Bahnsignale. In der Pit-Stop-Lane selbst ist ein Atmega16 Prozessor installiert. Dieser regelt einerseits die Weiche durch die Weichenbits und die Pacecar Steuerbits. Zusätzlich verändert er das, von der Bahn kommende, Signal für die Pit-Stop-Spur um eine Manchestercodeverletzung. Die Verletzung tritt am Ende eines Kommandowortes auf. Ist das letzte Bit eine 0, wird 10µs „High“ angelegt, im Anschluss 150µs „Low“.

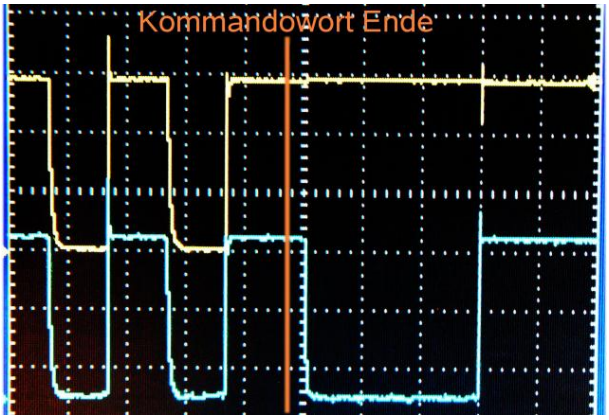


Abbildung 7: Manchestercodeverletzung - letztes Bit eine 0

Ist das letzte Bit eine 1, wird der Pegel zusätzlich 175 µs lang gegen „Low“ gelegt.

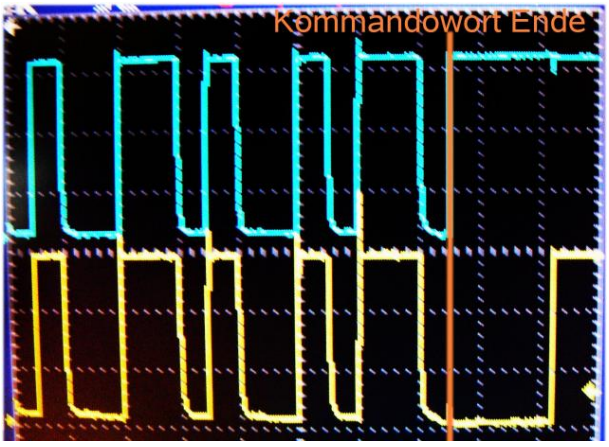


Abbildung 8: Manchestercodeverletzung - letztes Bit eine 1

4 P-Kanal Treiberschaltung (Transistorschaltung)

Um die Bahn über den PC ansteuern zu können, haben wir eine Transistorschaltung benötigt, die dafür gesorgt hat, dass die korrekte Spannung auf die Bahn kommt. Sie dient der Umsetzung der 5 V Spannung aus dem Prozessor auf die Bahn. Der Prozessor hat ein I_{Output} von 10 mA. Die Signale werden mithilfe eines MOSFETs auf 14,7 V und bis zu 8,8 A umgesetzt. Ist das Signal 0 V - also LO - so wird die Steuerleitung zusätzlich über einen Transistor auf 0 V gezogen. Dem Transistor ist ein Inverter vorgeschaltet.

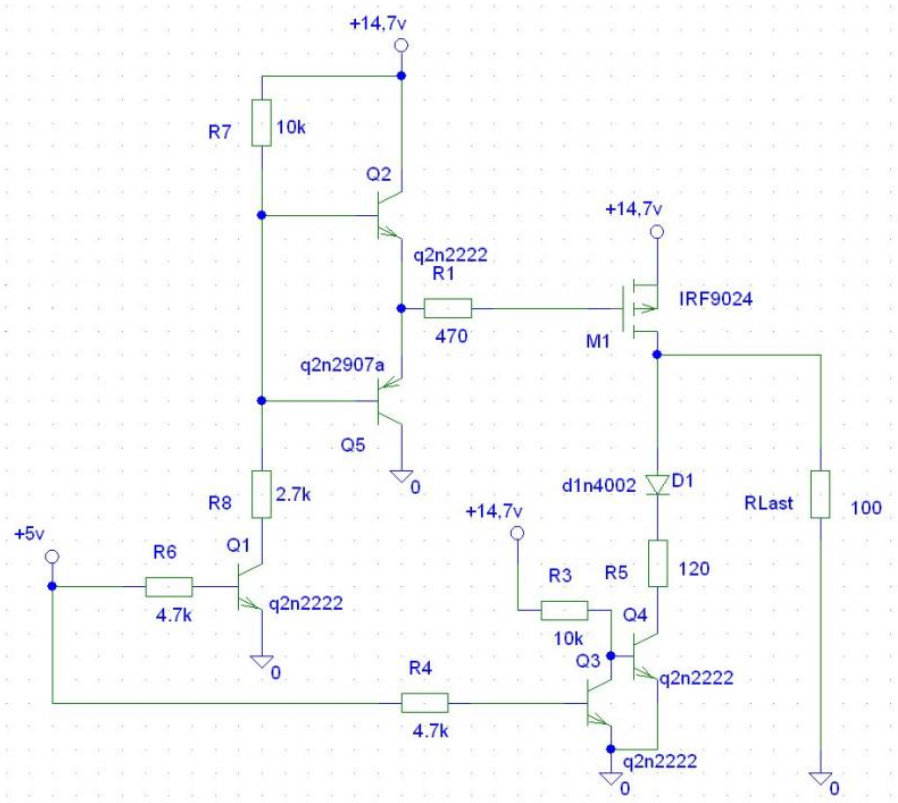


Abbildung 9: Transistorschaltung zur Ansteuerung der Bahn

Der einfachste Teil ist die Verschaltung von Q3, Q4, R3, R4. R4 ist ein Vorwiderstand für den Inverter Q3, R3. Fließt an der Basis von Q3 ein Strom, so schaltet er den Widerstand R3 gegen Masse. Am Collector von Q3 und somit an der Basis von Q4 liegt Masse an. Q4 Schaltet seinen Collector nicht gegen Masse. Fließt an der Basis Q3 kein Strom, so schaltet Q4 durch. Dies dient dem Absaugen der Kapazitäten und erhöht die Flankensteilheit.

Die Leistung - also die 14,7V mit bis zu 11A - wird allerdings über den FET gesteuert. R6 ist wiederum ein Strombegrenzer für Q1. Dies ist wieder eine Inverterschaltung, die den Spannungsteiler R8, R7 bedient. Der Ausgang des Spannungsteilers steuert die Bipolartransistoren Q2 und Q5, die jeweils am Emitter die Ladung über den Vorwiderstand R1 von M1 anhäuft oder absaugt. Der Vorwiderstand R1 regelt die Ladegeschwindigkeit des FET und damit seine Flankensteilheit. Er ist im Gegensatz zu den Bipolartransistoren Spannungsgesteuert. Die Transistorschaltung Q2, Q5, R8, R7 dient zur Speisung mit 14,7V bzw. 0V. Ist M1 gegen 14,7V (also Entladen - da Gate-Source gleich 14,7V), so sperrt der FET. Ist er (Gate-Source) gegen 0V geladen so ist er durchlässig. R_{Last} stellt die Bahn mit all ihren Verbrauchern (Auto, Weiche und Pitstop) dar. R5 federt die Flankenstöße[1] zwischen M1 und Q4 ab.

[1] Beide Transistoren sind Träge und schalten nicht sofort an bzw. aus. Dadurch fließt für eine kurze Zeit ein hoher Strom von 14,7V über den FET und den Bipolartransistor gegen Masse.

5 Phase 1: Ansteuerung über PC

In diesem Kapitel möchten wir auf unsere Ansätze zur Ansteuerung der Autos auf der Bahn genauer eingehen.

5.1 Ansatz: Altera

Nachdem die Details zur Ansteuerung bekannt waren, ging es darum, die Autos über den PC anzusteuern. Hierzu musste eine Schaltung gebaut und eine Ansteuerungssoftware geschrieben werden.

5.1.1 Serialisierer

Da die I/O-Karte im PC die Daten parallel auf ihre Ausgänge gibt und die Bahn jedoch serielle Daten erwartet, musste eine Schaltung entwickelt werden, welche die Daten serialisiert.
Durch die Vorlesung „Digitale Systeme“ waren einem Teil von uns die Altera-Bausteine bereits bekannt. Die dazugehörige Technik (zum entwickeln und brennen der Bausteine) stand uns im TI-Labor zur Verfügung. Deshalb wählten wir zunächst diesen Ansatz um einen Serialisierer zu entwerfen.
Dieser Serialisierer sollte die Daten der I/O Karte des PC's über Eingänge empfangen und sie auf einem Ausgang seriell auf die Transistor-Schaltung (s. Kapitel 0) geben. Diese Schaltung legt schließlich die Daten auf die Bahn.
Als Baustein verwendeten wir mehrere Altera EPM5032DC-20. Dieser besitzt insgesamt 28 Pins, wobei 4 Pins bereits für VCC und GND reserviert sind. Die restlichen 24 Pins standen uns zur Verfügung um die Signale der I/O-Karte einzulesen und das Ergebnis als Ausgang auf einen Pin zulegen.

Die Bausteine konnten über ein Steckbrett schnell an die Bahn angeschlossen und getestet werden.

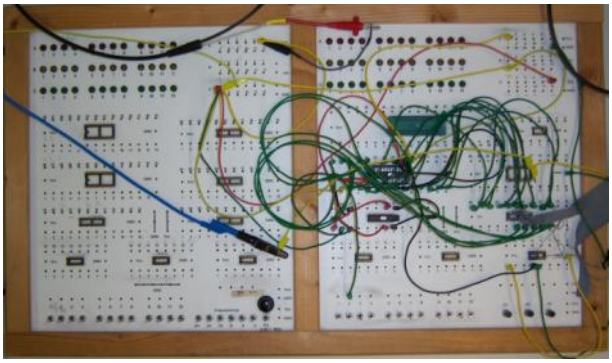


Abbildung 10: Steckbrett zum Testen der Bausteine

5.1.1.1 Version 1

Die erste Version wurde als Prototyp entwickelt, um beliebige Daten vom Laborrechner auf die Bahn zu legen. Sie sollte möglichst schnell entwickelt werden, damit wir testen konnten, wie gut dieser Ansatz funktioniert. Im Baustein sollte ein 20-Bit-Schieberegister implementiert werden. An die 20 Bits des Schieberegisters werden die Ausgänge der I/O-Karte gelegt. Falls der Baustein das „Sende“-Signal (ein weiterer Eingang) empfängt, wird immer das erste Bit im Schieberegister auf den „Serial“-Ausgang gelegt und die Bits im Schieberegister um eins nach Rechts verschoben. Jedes Bit wird im Schieberegister in einem 2:1 Multiplexer und einem D-Flipflop gespeichert (siehe Abbildung 11).

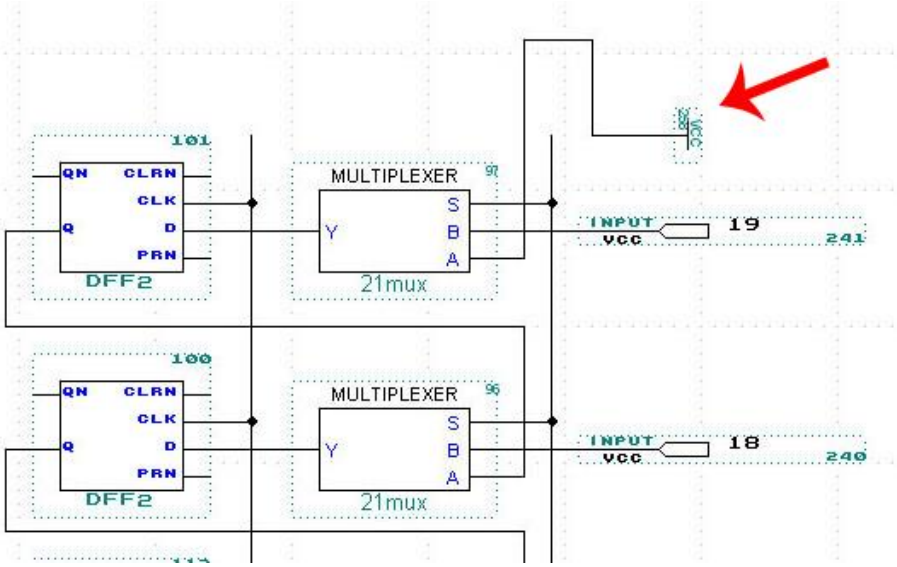


Abbildung 11: Ausschnitt aus Schieberegisterschaltung mit D-Flipflop und 2:1-MUX

Am A-Eingang jedes Multiplexers wird der D-Ausgang des Flipflops des vorherigen Bits verschalten. Beim letzen Bit wird an diesen Eingang ein VCC (roter Pfeil) gelegt, damit das Schieberegister mit logischen Einsen aufgefüllt wird. Der B-Eingang des Multiplexers wird mit dem passenden Daten-Eingang von außen verschaltet. Der Multiplexer wird über das „Sende“-Signal gesteuert. Der CLK-Eingang des D-Flipflops wird mit dem Takt verschalten und der D-Eingang mit dem Y-Ausgang des Multiplexers. Somit werden bei jedem Takt, sofern das „Senden“-Signal gesetzt ist, die Daten des N-ten Bit in das (N-1)-te Bit verlagert. Die Daten des ersten Bits in der Schaltung werden jeweils auf den seriellen Ausgang gelegt.

Zusätzlich wurde ein „Done“-Signal als Ausgang entwickelt. Dieses gibt an, ob der gesamte Inhalt des Schieberegisters auf den seriellen Ausgang gelegt wurde. Hierfür wurden die Ausgänge der ersten 3 FlipFlops im Schieberegister an ein AND verschalten. Wenn diese ersten Bits alle eine logische Eins enthalten, ist der Sendevorgang abgeschlossen. Dies ist jedoch nur möglich, da das Signal für die Bahn Manchester-kodiert ist. Aus diesem Grund darf es nicht vorkommen, dass 3 aufeinander folgende Daten-Bits den gleichen Wert enthalten ohne die Manchester-Codierung zu verletzen.

5.1.1.1.1 Vorteil

Ein großer Vorteil dieser ersten Version war die kurze Entwicklungszeit. Diese brachte eine schnelle Einsatzbereitschaft des Bausteins mit sich, so dass die Schaltung schnell getestet werden konnte.

5.1.1.1.2 Nachteile

Diese Schaltung brachte jedoch auch sehr viele Nachteile mit sich.
Einer dieser Nachteile ist es, dass bei dieser Schaltung der Manchester-Code durch den PC erstellt werden muss. Daraus folgte ein weiterer Nachteil: Wie wir bereits wussten werden bis zu 13-Bit lange Worte auf die Bahn versendet. Da jedoch nur freie Pins für 20 Bits verfügbar waren, konnten wir also nur 10 Bit im Manchester-Code auf die Bahn legen. Zudem kommt auch noch der Punkt, dass der PC die Daten bereits Manchester-codieren musste. Ein weiterer Nachteil war, dass es keine Synchronisierung gab. Das heißt wenn sich die Daten während dem Sendevorgang geändert haben wurden die neuen Daten direkt versendet. Dabei vermischt sich das alte Signal mit dem neuen. Dies führte zu ungültigen Signalen. Ebenso mussten die Pausen zwischen 2 Signalen selbst kontrolliert werden. Das heißt, es war möglich 2 Signale direkt, ohne Pause, zu senden. Diese Datenworte wurden dann von den Autos nicht korrekt erkannt.

5.1.1.2 Version 2

Da die erste Version nur eine Vorabversion zu Testzwecken war, wurde sie verbessert und weiterentwickelt. Das Ziel war es, die oben erwähnten Nachteile zu beseitigen. Viele dieser Nachteile hängen mit der Datenwortlänge am Eingang des Serialisierers und dem Manchester-Code zusammen. Aus diesem Grund wollten wir den Manchestercode in dem Serialisierer aus den Grunddaten generieren. Die Anzahl der Dateneingänge konnte somit auf 13 Bit verkleinert werden. Intern werden diese in Manchester-Code umgewandelt und seriell ausgegeben. Um die 3 verschiedenen Wortlängen (8, 10 und 13Bit) versenden zu können, wurden 2 Steuerbits eingebaut. Über diese wird die Länge des zu versendenden Datenworts bestimmt.

S0 / S1	Datenwortlänge
0 / 0	8 Bit
1 / 0	8 Bit
0 / 1	10 Bit
1 / 1	13 Bit

Tabelle 7: Geschwindigkeitsmessungen

Die Steuerbits S0 und S1 haben die Aufgabe das Schieberegister sozusagen abzuschneiden.
Zusätzlich wird in dieser Version eine Pause zwischen zwei Signalen erzeugt. Sie wird über einen Zähler realisiert. Dieser beginnt zu zählen sobald das „Done“-Signal gesetzt wird. Wenn er bei einem bestimmten Zählerstand angelangt ist, setzt er das „Sende“-Signal. Durch Verändern dieses Zählerstandes ist es möglich, die Größe der Pause zu regulieren.

Dieses „Sende“-Signal wird somit nicht mehr von außen als Eingang realisiert sondern intern generiert. Da die von uns verwendeten Bausteine auf eine gewisse Anzahl von Logikzellen begrenzt sind, musste dieser Zähler auf einem extra Baustein realisiert werden.

Das Senden der Signale wird weiterhin mit dem PC über das „Done“-Signal synchronisiert. Der PC schreibt die zu sendenden Daten auf die I/O Karte und wartet bis das „Done“-Signal vom Baustein gesetzt wird. Daraufhin kann er neue Signale auf die I/O Karte schreiben. Somit werden die Daten beim Senden eines Signals nicht mehr unterbrochen bzw. verfälscht.

5.1.1.3 Version 3

In dieser Version sollte versucht werden, den Baustein für einen möglichen Rückkanal weiterzuentwickeln. Es sollte versucht werden, die logische Eins, die in den Pausen zwischen 2 Signalen auf die Bahn gelegt wird, für einige Takte auf null zu ziehen. In dieser Zeit hätte zum Beispiel ein Auto Zeit ein Signal zurück zu senden. Hierfür wurde der Baustein, der den Zähler enthält etwas verändert. Er wurde um einen „Zero“-Ausgang erweitert. Dieser wird auf High gesetzt sobald der Zähler in einem bestimmten Werte-Bereich ist. Hierfür wurden die 4 höchsten Bits an ein AND geschalten. Das bedeutet, dass das Zero-Signal 16 Takte auf Eins gesetzt wird.

Das Schieberegister erhält dieses „Zero“-Signal als Eingang. Die Schaltung hier wurde um einen 2:1 Multiplexer erweitert. Das Zero-Signal wird an den Steuer-Eingang, der A-Eingang mit dem neuen Eingang „Delay“ und der B-Eingang mit dem letzten Bit des Schieberegisters verschalten. Der Delay-Eingang gibt das Signal an, welches in der Pause auf die Bahn gelegt werden soll. Somit gibt die Schaltung bei nicht gesetztem „Zero“-Signal die Daten aus dem Schieberegister (in der Pause also eine logische Eins) auf die Bahn. Bei gesetztem „Zero“-Signal wird das Signal, das am „Delay“-Eingang anliegt auf die Bahn gelegt.

Diese Schaltung sollte dazu dienen zu überprüfen, ob die Autos eine Lücke "vertragen". Die Tests ergaben, dass Autos ungestört weiter fahren. Das zeigte, dass es möglich ist diese Lücke für ein Rücksignal zu verwenden.

5.1.2 Software-Aufbau

Der Erste von uns gewählte Softwareansatz war eine „straight forward“-Implementierung. Neue Ideen und Überlegungen konnten so leicht umgesetzt und integriert werden.

5.1.2.1 Simple Blackbox Simulation

Als erstes haben wir eine Aufnahme des Datenstroms auf der Bahn gemacht und diesen im zweiten Schritt eins zu eins wieder auf die Bahn gegeben. Ein erster Erfolg: die Autos ruckelten, waren aber in keinster Weise steuerbar. Die Kommunikation erfolgt über eine WASCO-IOCard. Da das Programm immer wieder vom Betriebssystem, selbst mit hoher Priorität, unterbrochen wurde, kam es immer wieder zur asynchronen Aussendung der Daten.
Aus diesem Grund wurde ein Baustein zur Serialisierung des Datenstroms entwickelt (siehe Kapitel „4 P-Kanal Treiberschaltung (Transistorschaltung)“).

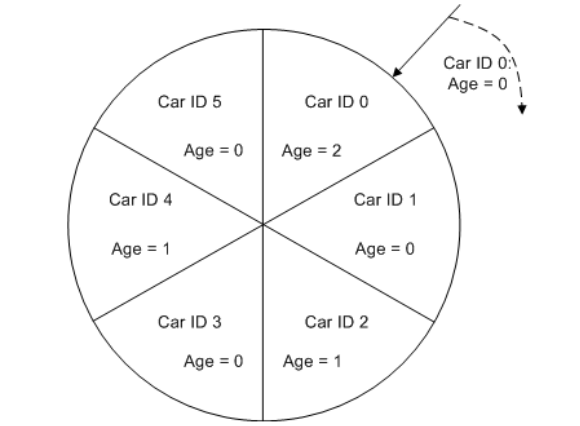
5.1.2.2 Nachbau des Carrera Protokolls

Um die Autos direkt steuern zu können, haben wir eine Nachbildung des Protokolls wie es unter [SLOTB] zu finden ist, implementiert. Dieses wurde Software-seitig in Manchester Code gewandelt und über den Serialisierer-Baustein auf die Bahn gegeben. Dadurch konnten wird die Geschwindigkeit der Autos regeln. Im weiteren Vorgehen haben wir die Generierung des Manchester Codes auf die Hardware verlagert (siehe Kapitel 5.1.1.2) und mit Hilfe der „ncurses“-Bibliothek eine Software-seitige Steuerung der Autos implementiert.

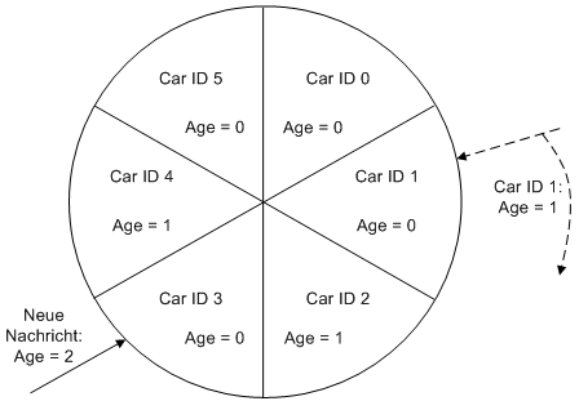
5.1.2.2.1 Protokoll Scheduling

Um mehr Kontrolle über das Aussenden der Protokolle zu erhalten, haben wir unser Programm um einen Scheduler erweitert. Dieser läuft in einem eigenen Thread welcher mit Hilfe der Pthread Library erzeugt wird. Er verwaltet die Protokolle in einem Ringpuffer mit einem Slot für jedes Auto. Jedes Protokoll unterliegt dabei einem, vom Scheduler gesteuerten, Alterungsprozess. Zu Beginn ist das Alter aller Nachrichten mit DEADMSG initialisiert. Tote Nachrichten werden vom Scheduler nicht beachtet. Will nun ein Auto ein Protokoll versenden, wird dieses in den zugehörigen Slot geschoben und als NEWMSG markiert. Trifft der Scheduler bei seiner Rundreise durch den Puffer auf ein mit NEWMSG markiertes Protokoll, sendet er es aus und markiert es als OLDMSG. Findet er ein nicht mit DEADMSG oder NEWMSG markiertes Protokoll zählt er dessen Alter hoch. Nach 2 Zyklen des Schedulers durch den Ringpuffer sind auch die gealterten Nachrichten wieder mit NEWMSG markiert, damit die Autos ihre Steuerinformationen erhalten.

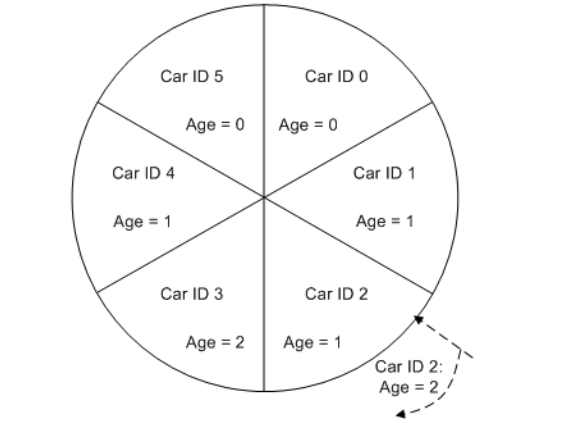
Das folgende Beispiel soll die Funktionsweise des Schedulers verdeutlichen. Der Kreis mit den Kuchenstücken stellt den Ringpuffer mit den jeweiligen Slots für die einzelnen Autos dar.



In diesem Fall ist der Scheduler bei Car ID 0. In diesem Slot befindet sich eine neue Nachricht (NEWMSG; Age = 2). Diese Nachricht wird versendet und danach das Alter dieser Nachricht auf OLDMSG (Age = 0) gesetzt. Der Scheduler hat somit diesen Slot abgearbeitet.



In diesem Slot des Autos 1 findet der Scheduler eine Nachricht mit dem Alter 0 (OLDMSG). Diese Nachricht muss nicht versendet werden. Es wird also nur das Alter um 1 erhöht. Zu diesem Zeitpunkt kommt eine neue Nachricht für Auto 3. Das Alter bei Auto 3 wird somit auf NEWMSG (Age = 2) gesetzt.



Der Scheduler ist jetzt beim Slot für Car ID 2 angelangt. Die Nachricht in diesem Slot hat das Alter 1. Sie wird daher nicht versendet. Das Alter dieser Nachricht wird um 1 erhöht, steht also jetzt auf NEWMSG. Im nächsten Durchlauf würde diese Nachricht erneut versendet werden.
Im nächsten Slot wird die mittlerweile erneuerte Nachricht des Autos 3 versendet.

Tabelle 8: Beschreibung des Scheduling-Algorithmus

5.1.2.3 Kernel Modul

Um größere Latenzzeiten zu vermeiden und die Steuerung sprachunabhängiger zu gestalten, haben wir unsere Implementierung in den Kernel verlagert. Da das System kompatibel mit anderen Steuermodulen sein sollte, haben wir uns für einen mehrschichtigen Ansatz entschieden der in der folgenden Abbildung dargestellt ist:

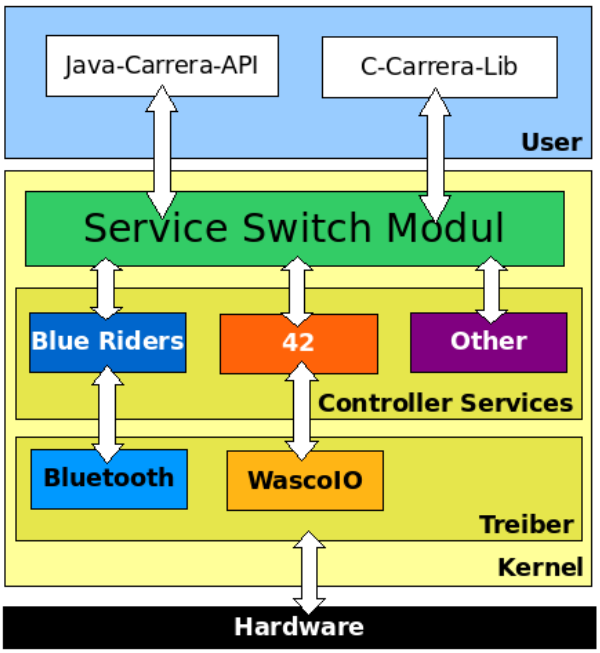


Abbildung 12: Schichten des Kernel-Moduls

Das Service Switch Modul bildet in diesem System die Schnittstelle zwischen den Benutzerbibliotheken und den verschiedenen Steuerungsmodulen. Der Benutzer soll dabei die Möglichkeit haben, verschiedene Services zu initialisieren und Autos für diese zu registrieren. Die verschiedenen Services melden sich zu Beginn bei dem Service Switch Modul an und hinterlegen „callback“-Funktionen für deren Steuerung.

Für diesen Ansatz haben wir unsere jetzige Implementierung in ein Kernelmodul portiert. Auch das oben beschriebene Scheduling-Verfahren wurde von uns als eigenständiger Kernelthread umgesetzt. Zusätzlich haben wir das Service Switch Modul und eine kleine Benutzerbibliothek zur Kommunikation entworfen. Um Auto-Werte an das Servicemodul zu senden wird eine vier Byte-große „struct“ übermittelt welche aus folgenden Daten besteht:

Bit	Name	Beschreibung
8	carID	Identifikationsnummer des Autos
8	speed	Geschwindigkeit des Autos
8	switcher	Weichenschalung des Autos
8	fuel	Benzin aktivierung des Autos

Tabelle 9: Daten der „struct“ an das Servicemodul

5.1.3 Probleme

Bei diesem Ansatz hatten wir mit einigen Problemen bei der Hardware zu kämpfen:
Zum Einen hatten wir das Problem, dass der Takt nicht so genau einzustellen war. Aus diesem Grund bekamen die Autos teilweise falsche Signale. Das Resultat war, dass die Autos mit Vollgas gefahren sind und sich nicht mehr kontrollieren ließen. Der einzige Weg, diese Fehlprogrammierung aus den Autos wieder herauszubekommen, war die Carrera-Blackbox einzuschalten, während das „kaputte“ Fahrzeug auf der Bahn steht.
Ein weiteres Problem war die große Wärmeentwicklung. Die Chips wurden während dem Betrieb der Bahn sehr heiß.
Zudem war es bei diesem Ansatz nicht sehr einfach die Schaltung anzupassen. Sollten Veränderungen vorgenommen werden, war dies nur mit sehr großem Aufwand möglich.

5.2 Ansatz: Atmel

Da die Steuerung über die Altera-Bausteine nicht einwandfrei funktioniert (siehe Kapitel „5.1.3 Probleme“) haben wir parallel einen weitern Ansatz verfolgt: Die Ansteuerung der Bahn soll über einen Atmega Prozessor laufen.

5.2.1 Atmel Toolchain

Wir verwendeten eine Atmel Crossumgebung unter Linux. Wir stellten dafür eine Toolchain zusammen.
Diese besteht aus:
avr-binutils: Asembler, Linker ...
avr-gcc: Kompiler
avr-libc: Bibilotheken
avrdude: Programmiersoftware für verschiedenste Programmiermethoden, darunter stk500v2.
Archive davon befinden sich im Repository.

5.2.2 Atmel-Steuerung TODO Bilder Neu

Die Atmel-Steuerung entstand nach den Versuchen mit Altera CPLDs die Bahn anzusprechen. Wir entschieden uns für diesen Schritt, da mithilfe einer direkten Prozessorimplementierung, ohne Betriebssystem, die Asynchronität besser in den Griff zu bekommen ist. Das Rausschieben des Manchestercodes ist in einem Timer Interrupt zeitlich synchronisiert. Die Steuerbefehle werden vom PC über die serielle Schnittstelle angenommen. Jedes empfangene Byte löst wiederum einen Interrupt aus. Die empfangenen Befehle werden vom Prozessor ausgewertet und für den Scheduler abgespeichert. Ist ein Datenwort raus, so wird wieder der Scheduler angeworfen.

5.2.2.1 Konzept

Das Konzept basiert auf einer integrierten Lösung, die als Eingabe Kommandobefehle erhält und als Ausgabe den Manchestercode für die Bahn generiert. Dabei ist die Anordnung und der zeitliche Ablauf in diesem System Implementiert. Diese Box übernimmt somit ein statisches offline Echtzeitscheduling und in Verbindung hiermit eine Synchronisierung.

5.2.2.1.1 Scheduling

Dieses Mal haben wir uns für ein statisches Scheduling der Daten entschieden. Der Ablauf entspricht weitestgehend dem des original Carrera Systems.

5.2.2.2 Realisierung

Anfangs wurde die Software des Atmel-Prozessors auf dem Entwicklungsboard stk500v2 entwickelt. Für die Ansteuerung der 14,7 V der Bahn wurde die Treiberschaltung des Altera-Boards verwendet. Nachdem auch der Später beschriebene Rückkanal aufgebaut und getestet wurde, ließen wir eine geätzte Platine fertigen.

5.2.2.2.1 Software

Der Code auf dem Atmel-Prozessor teilt sich auf in:

- **Timer-ISR:** Herausschieben der Manchestersignale, Steuerung der Pegel der Bahnleitungen.
- **UART-ISR:** Empfangsinterrupt der Steuerbefehle.
- **Main Loop:** Nach der Initialisierung wird hier in einer Schleife auf einen Interrupt gewartet.

Die Timer ISR ist das Kernstück der Atmel-Steuerung. Die Aufruffrequenz beträgt genau die Länge eines halben Manchesterbits. Tritt ein Interrupt auf, so werden zunächst bis zum Austreten dieser ISR alle später auftretenden Interrupts gesperrt. Dann wird mithilfe eines Zählers geprüft, an welcher

Stelle des Kommandos sich der Prozessor befindet. Das Flussdiagramm in Abbildung 13 und Abbildung 14 zeigt den Ablauf mit den Verzeigungsfällen. Der Sendestatus gibt an, ob gerade ein Kommando gesendet wird. Die im Kapitel „3.7 Pitstop-Lane“ erwähnte Manchestercodeverletzung kann optional angehängt werden. Der Rückkanal wird anhand des Sequenzzählers optional aufgeschaltet. Der Sequenzzähler wird mit jedem ISR inkrementiert und wird, wenn er die 154 überschreitet, auf Null zurückgesetzt.

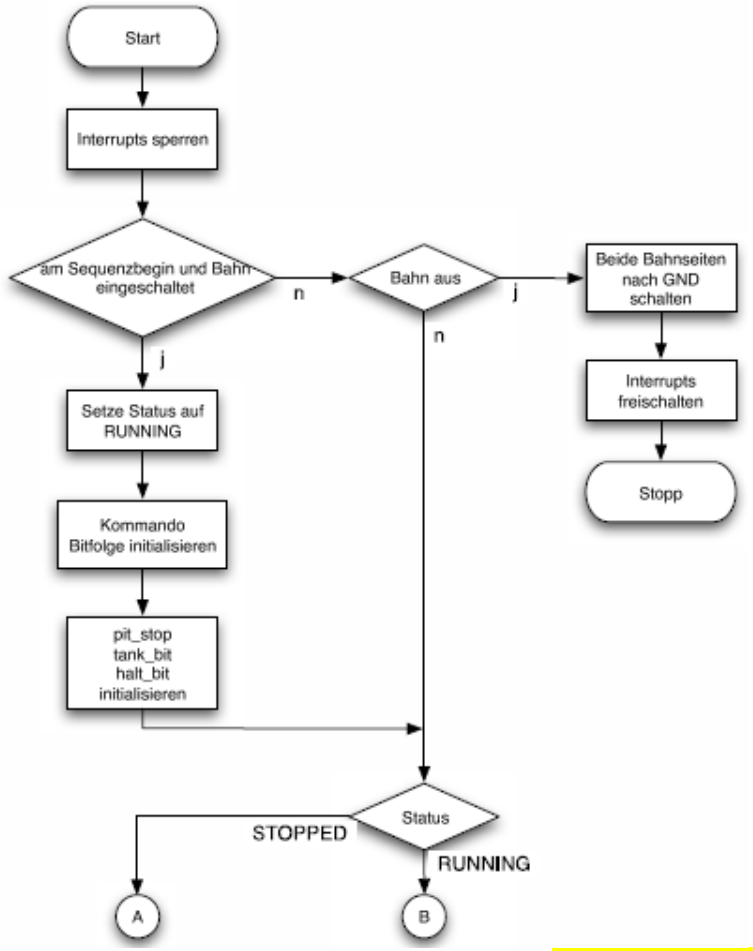


Abbildung 13: Flussdiagramm Teil 1 der Software im Atmel TODO bessere Qualität

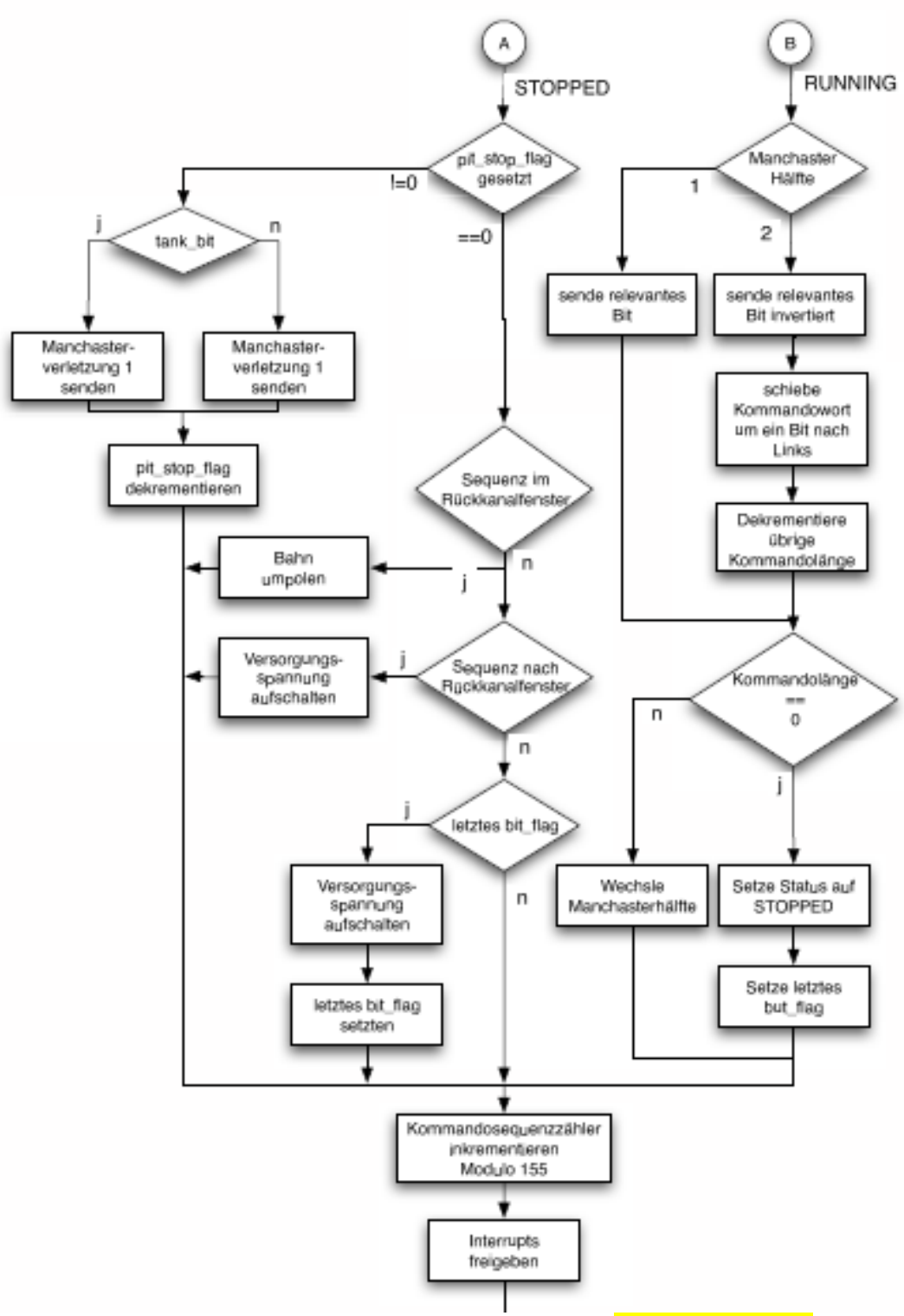


Abbildung 14: Flussdiagramm Teil 2 der Software im Atmel TODO bessere Qualität

5.2.2.2.2 Steuerkommandos

Die Atmelbox erhält vom PC über die serielle Schnittstelle Befehle. Eine Kommandosequenz besteht aus vier Bytes. Das erste Byte gibt den Steuertyp an. Byte 4 muss Null sein. Folgende Steuertypen werden versandt:

Autosteuerung (s): Mit diesem Kommando wird die Geschwindigkeit, das Weichen- und Tankbit individuell für jedes Auto gesetzt.

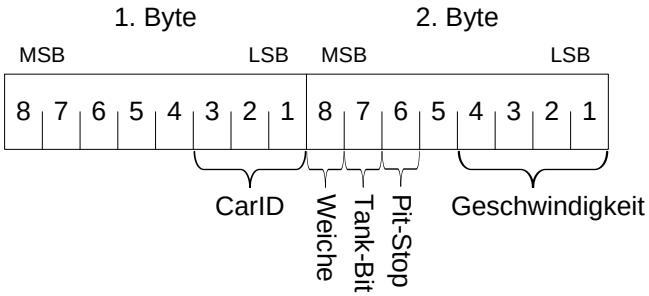


Abbildung 15: Steuerkommando

Programmierung (p): Hiermit kann die Bremse, die Tankfüllung und die Geschwindigkeitsskalierung eingestellt werden. Das erste Argument ist die CarID, das zweite setzt sich aus dem Parametertyp und dem Value zusammen.

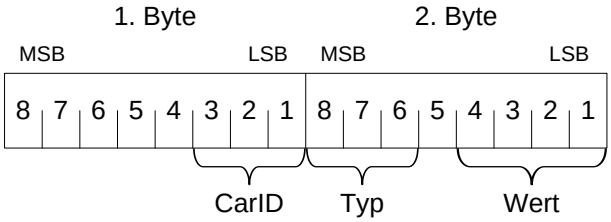


Abbildung 16: Programmierdatenwortkommando

Rückkanalfenster (a): Dieses Kommando benötigt als erstes Argument den Wert 1 für aktiv und den Wert 0 für inaktiv. Die Beschreibung des Rückkanalfensters wird in einem späteren Kapitel aufgeführt.

Stromabschaltung (b) Auch hier ist das erste Argument 1 für Strom an und 0 für Strom aus.

Ghostcar (g) Hiermit kann das Ghostcar gesteuert werden.

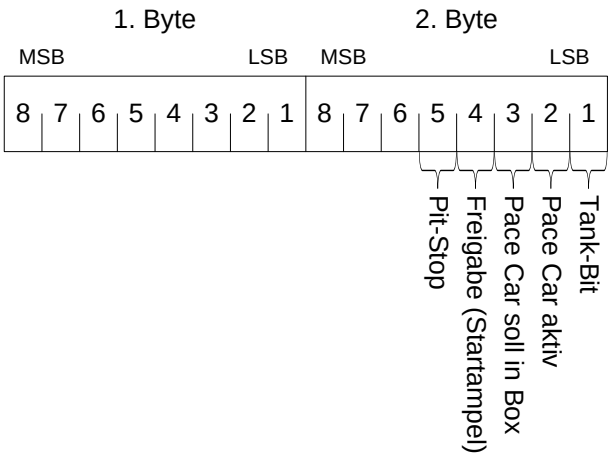


Abbildung 17: Ghostcarkommando

5.2.3 Hardware

Unser Board enthält alle notwendigen Elemente: Einen Prozessor, eine 5V Spannungserzeugung, Brückengleichrichter, RS232-Pegelwandler, Reset und ISP Schnittstelle.

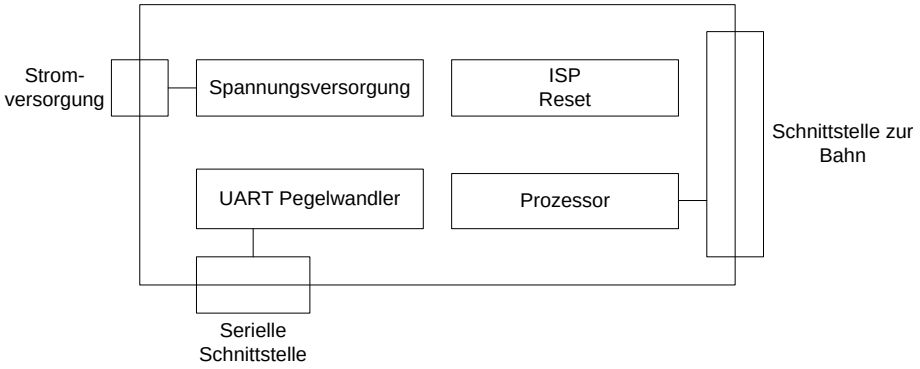


Abbildung 18: Mainboard Blockdiagramm

Das Layout wurde mit Eagle entworfen und geroutet. In der nächsten Abbildung ist die Anordnung und Bezeichnung der Bauteile zu sehen.

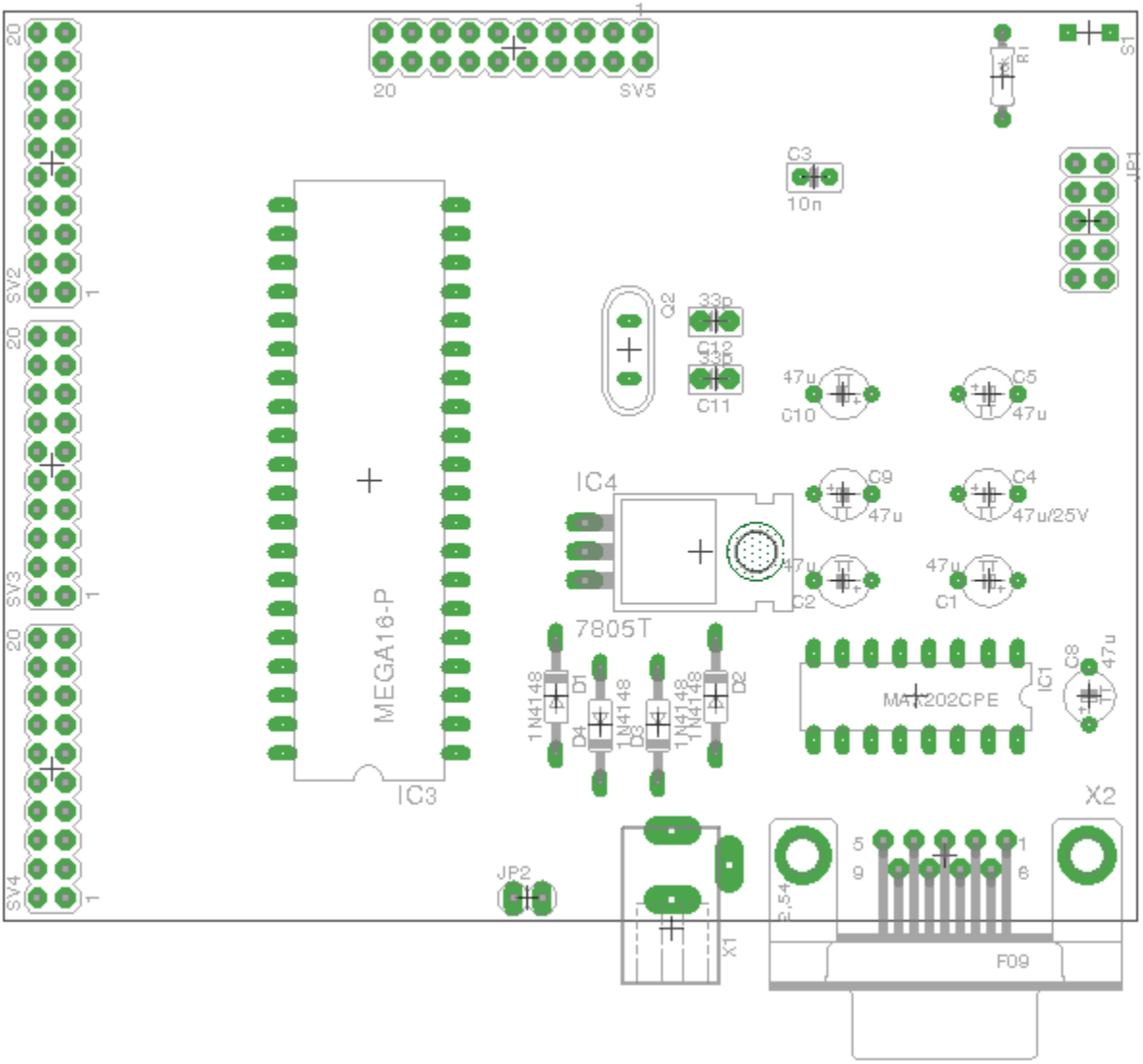


Abbildung 19: Mainboard Layout

Im Folgenden werden die Pin-Belegungen dieser Platine beschrieben.

SV4: Auf SV4 wird ein Treibermodul für die linke Bahnleitung gesteckt. Softwaremäßig sind hier das Modul mit dem P-Kanal FET gegen V+ den NPN Bipolartransistoren gegen Masse.

PIN	**Beschreibung**
1	GND
2	NC
3	VCC (+5V)
4	NC
5	PC0 (Steuerleitung)
6-16	NC
17	V+ (+14,7V)
18	SPUR
19	GND
20	SPUR

Tabelle 10: Pinbelegung SV4

SV6: Auf SV6 wird das Treibermodul für die gegenüberliegende Bahn gesteckt. Hier sind also der N-Kanal FET gegen Masse und den PNP-Bipolartransistoren gegen V+.

PIN	**Beschreibung**
1	GND
2	NC
3	VCC (+5V)
4	NC
5	PC1 (Steuerleitung)
6-16	NC
17	V+ (+14,7V)
18	SPUR
19	GND
20	SPUR

Tabelle 11: Pinbelegung SV6

SV2: SV2 ist der Port für das Empfangsteilmodul.

PIN	**Beschreibung**
1	GND
2	NC
3	VCC (+5V)
4	NC
5	SPUR
6	SPUR
7-16	NC
17	PD3 (INT1)
18	NC
19	PD2 (INT0)
20	NC

Tabelle 12: Pinbelegung SV2

JP1: ISP-Schnittstelle um den Atmel Prozessor zu programmieren. Hierfür wird auf die Literatur im Internet verwiesen. Diese Schnittstelle ist standardisiert. Die Pinbelegung wollen wir der Vollständigkeit wegen aufführen.

PIN	**Beschreibung**
1	MOSI
2	VCC (+5V)
3	NC
4	GND
5	RESET
6	GND
7	SCK
8	GND
9	MISO
10	GND

Tabelle 13: Pinbelegung JP1

JP2: Versorgungsausgänge für die Bahnleitungen. Im Bild oben ist der linke Pin für die linke Bahnleitung (also roter Stecker) und der rechte Pin (blauer Stecker) für die rechte Bahnleitung.
X1: Stromversorgungseingang für die Versorgung der Bahn und der internen Bauteile. Der Eingang ist durch einen Brückengleichrichter gegen Verpolung geschützt. Eine Sicherung gibt es keine. Hier gibt es zu beachten, dass die Spannung größer als die Spurspannung sein muss, da durch den Bandabstand an den Gleichrichterdioden Spannung abfällt.
X2 RS232 Ausgang für den Anschluss an den PC. Die Pegel sind für den PC korrekt. Sie werden bereits intern gewandelt.

5.2.4 Gehäuse

Als Zugabe wurde diese Platine in ein Gehäuse eingebaut. Hier werden nur die für den Anwender wichtigen Schnittstellen herausgeführt. Im unteren Teil des Gehäuses befinden sich die für die Ansteuerung relevanten Anschlüsse.

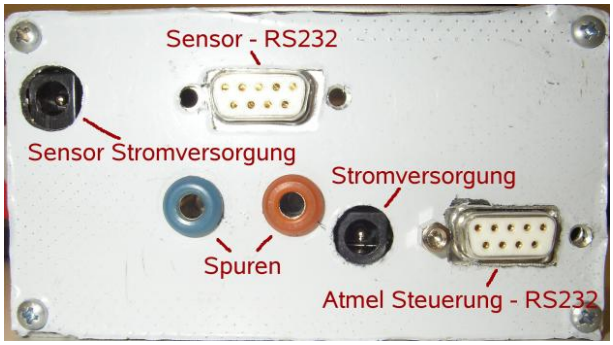


Abbildung 20: Gehäuse-Seite mit Anschlüssen

Stromversorgung: Für die Atmelansteuerung gibt es einen Stromversorgungsanschluss.
Atmel Steuerung RS232: Schnittstelle für den PC
Spuren: Der blaue Anschluss für die rechte Bahnleitung der rote für die linke Bahnleitung.

5.2.5 Software-Aufbau

Bei diesem Ansatz wurde die Software nicht mehr in den Kernel integriert. Dafür wurden gewisse Teile der Software auf die Hardware ausgelagert. So wurde zum Beispiel der Scheduler komplett auf den Atmega-Prozessor gesetzt. Damit die Signale auch in der von den Autos erwarteten Reihenfolge an die Bahn gesendet werden, haben wir den Scheduler von Carrera exakt nachgebaut. Die Reihenfolge der Signale ist bereits im Kapitel 3.2.1 beschrieben.
Die Kommunikation zwischen PC und dem Atmega-Prozessor erfolgt über die serielle Schnittstelle. Die Schnittstellenbeschreibung zur Kommunikation ist unter „5.2.2.2 Realisierung“ aufgeführt.

5.2.5.1 Lib42

Um die Kommunikation zwischen PC und dem Atmega Prozessor einfacher zu gestalten, haben wir eine Bibliothek namens lib42 entwickelt. Diese kapselt die Kommunikation über die serielle Schnittstelle und bietet somit ein einfaches Steuerinterface. Um die Steuerung der Autos zu erleichtern, enthält diese Bibliothek auch Methoden und Datenstrukturen zur Verwaltung der Autoinformationen. Für eine detailierte Beschreibung der Methoden siehe Dokumentation im Code.
Außer Methoden und Datenstrukturen zur Steuerung der Autos, erhält die Bibliothek weitere Schnittstellen für folgende Features: Kollisionserkennung, Rundzähler, Sensordaten Queuing und Sensor Daten, Logging. Für genauere Informationen über die von uns eingebauten Sensoren, siehe Kapitel „7 Phase 2: Positionserkennung über Carrera System“.

5.2.5.1.1 Rundenzähler

Die Sensoren für den Rundenzähler werden dynamisch zur Laufzeit festgelegt. Die Bibliothek wird dafür mit einer beliebig langen Liste mit Sensor ID’s initialisiert. Somit kann der Rundenzähler auch bei einer Erweiterung der Bahn auf vier Spuren genutzt werden. Da die Sensoren allerdings zum jetzigen Zeitpunkt noch nicht zu 100% zuverlässig erkannt werden, kann die tatsächliche Anzahl an Runden vom Ergebnis Abweichen.

5.2.5.1.2 Kollisionserkennung

Die Kollisionserkennung ist, solange noch Sensordaten verloren gehen, leider nicht einsetzbar. Wenn Autos einen kritischen Bereich verlassen, ohne dass dies erkannt wird herrscht ein unbestimmter Zustand im System der nicht wieder aufgelöst werden kann. Wie auch beim Rundenzähler können die kritischen Bereiche zur Laufzeit festgelegt werden. Unter einem kritischen Bereich verstehen wir den Streckenabschnitt bei einer Weiche. Dieser wird durch vier Sensoren markiert. Zwei Sensoren zur Erkennung von einfahrenden Autos, und zwei zur Erkennung von herausfahrenden Autos. Im folgenden Beispiel sind diese mit ihren ID’s auf der Bahn markiert.
Es gibt zwei kritische Sektoren auf diesem Streckenabschnitt:

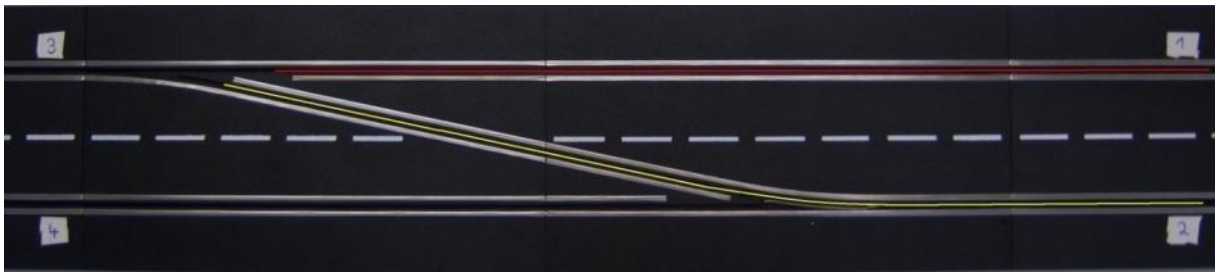


Abbildung 21: Kritische Sektoren bei Weiche

Wir bezeichnen sie mit critical_in (rot markiert) und critial_out (gelb markiert).Die Kontrolle der Autos in einem kritischen Bereich wird vom System übernommen. Es wird versucht die Autos mit denselben Daten, welche diese bei der Einfahrt in den Abschnitt hatten, durch den Bereich zu schleusen.

<u>Szenario 1</u>

Passiert ein Auto (rot) den Sensor 1, ohne das ein anderes Auto im kritischen Bereich ist, markiert es den Ausfahrtsbereich als kritische Zone. Passiert nun ein weiteres Auto (orange) den Sensor 2 mit gedrückter Weichentaste bevor das rote Auto den gefährdeten Streckenabschnitt verlassen hat, wird die Weichentaste deaktiviert. Das Auto wird somit gezwungen den grünen Weg zu nehmen und eine Kollision wird vermieden.

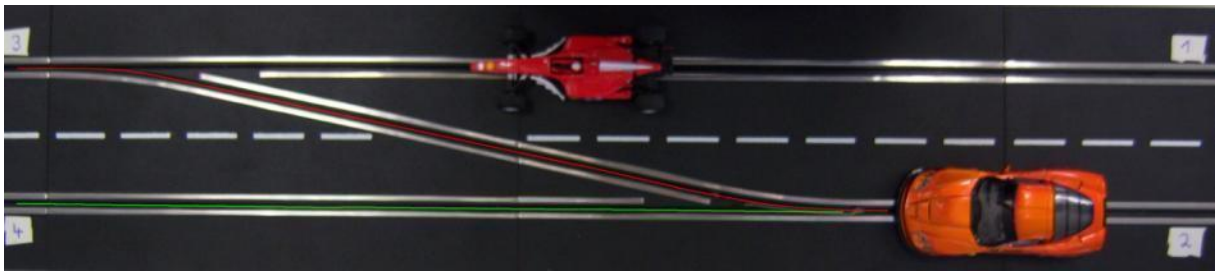


Abbildung 22: Kollisionsvermeidung Szenario 1

Szenario 2

Passiert als erstes ein Auto (orange) den Sensor zwei ohne gedrückte Weichentaste, so entsteht dadurch keine kritische Situation.

Passiert als erstes ein Auto (orange) den Sensor zwei mit gedrückter Weichentaste ohne dass ein anderes Auto im kritischen Bereich ist, so markiert es die Einfahrtszone als kritisch. In beiden Fällen reagiert das Auto erst wieder auf Input wenn es die Zone verlassen hat. Es kann also nicht nachträglich die Weiche schalten. In diesem Szenario gehen wir davon aus, dass das orangene Fahrzeug die Weichentaste gedrückt hat. Fährt nun ein weiteres Auto (rot) in den Bereich über den Sensor eins, so wird dieses abgebremst. Erst wenn das orange Auto den Bereich über Sensor 3 verlässt, wird das rote Auto wieder auf seine alte Geschwindigkeit beschleunigt. Weitere Autos, die den Sensor 1 passieren, werden ebenfalls gestoppt. Das Anfahren der Autos nach dem Abbremsen im kritischen Bereich erfolgt nach dem fifo-Prinzip.

Abbildung 23: Kollisionsvermeidung Szenario 2

Szenario 3

Gehen wir noch einmal von derselben Situation wie in Szenario zwei aus. Das rote Auto wird abgebremst sobald es den Sensor eins passiert hat. Dadurch wird eine Kollision mit dem orangen Auto vermieden. Dabei markiert es den Ausfahrtsbereich als kritisch. Fährt nun ein weiteres Auto (grau) über den Sensor zwei mit gedrückter Weichentaste, so wird diese deaktiviert. Es nimmt also nun den blauen anstatt den gewünschten gelb markierten Weg.

Abbildung 24: Kollisionsvermeidung Szenario 3

5.2.5.1.3 Sensor Queuing

Wird ein Auto an einem Sensor erkannt, so wird die ID des Sensors dem Auto als lastSensorID zugeordnet. Um jedoch einen chronologischen Zugriff auf die Sensordaten zu erhalten, kann das Sensorqueuing bei der Initialisierung der Bibliothek aktiviert werden. Der Benutzer hat dann die Möglichkeit die Sensordaten in der Reihenfolge ihrer zeitlichen Erkennung zu betrachten.

5.2.5.1.4 Logging

Desweitern kann ein Logger bei der Initialisierung der Bibliothek aktiviert werden. Dieser protokoliert alle detektierten Sensordaten in einer Datei mit. Dadurch kann mit Hilfe eines Skripts eine Überprüfung der Daten auf verlorengegangene Sensoren gemacht werden.

5.2.5.2 Änderungsvorschlag Lib42

Die Bibliothek besteht, wie bereits angemerkt, aus einer Mischung von Funktionen zur Kommunikation mit der Bahn (dem Atmega-Prozessor), und den Methoden zur Verwaltung der Auto und Sensordaten. Für eine Machbarkeitsstudie hat sich dieser Ansatz bewährt, da er leicht und schnell zu implementieren war. Für die Weiterentwicklung der Bahn empfehlen wir allerdings eine Trennung dieser Funkionen vorzunehmen. Eine eigene Carrera-Bibliothek, die die Kommunikation mit der Bahn kapselt, wäre an dieser Stelle der richtige Weg. Die Sensorerkennung die von der Bibliothek „lib42“ genutzt wird, ist bereits in einer eigenen Bibliothek csdlib gekapselt. Die folgende Grafik veranschaulicht das Zusammenspiel der Bibliotheken nochmals.

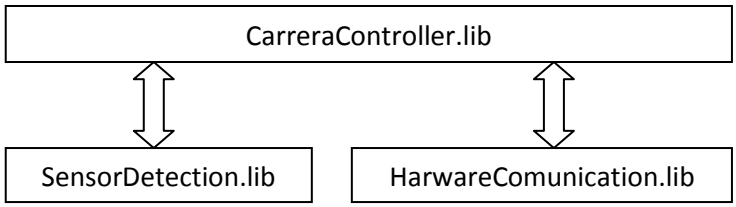


Abbildung 25: Zusammenspiel der Bibliotheken

5.2.5.3 csdlib

Die csdlib (Carrera Sensor Detection Library) kümmert sich um die Auswertung der eingehenden Sensordaten. Bei der Initialisierung der Bibliothek ist eine „callback“-Methode anzugeben, welche bei der Erkennung eines Sensors aufgerufen wird. Sie bekommt sowohl die ID des erkannten Autos als auch die ID des detektierenden Sensors übergeben. Für eine genaue Beschreibung der Sensor-Auswertung siehe Kapitel „7 Phase 2: Positionserkennung über Carrera System“.

5.2.6 Vorteile

Bei diesem Ansatz gibt es einige Vorteile gegenüber dem Altera Ansatz (s. Kapitel „5.1 Ansatz: Altera“). Zunächst einmal ist dieser Ansatz einiges einfacher anzupassen. Das Programm auf dem Chip lässt sich direkt vom PC auf den Baustein übertragen und ist in kürzester Zeit verwendbar. Der Code des Programms kann somit schnell angepasst bzw. verbessert werden. Außerdem konnten wir bei diesem Ansatz die Scheduling-Aufgabe in den Prozessor auslagern und mussten diese Aufgabe nicht mehr am PC lösen. Da auf der Platine ein Quarz für den Takt zuständig ist, können wir diesen genauer einstellen. Die Autos bekommen somit keine falschen Signale mehr und fahren immer so, wie wir sie vom PC aus steuern.

5.2.7 Fazit

Die Atmel Steuerung ist von der PC-Seite einfach anzusteuern. Man braucht als Anwender keine Synchronisierung zu beachten. Ein wichtiges Merkmal ist die Zuverlässigkeit. Die Funktionalität der Bahn ist dem Atmel-Prozessor bekannt. Ein großer Teil des Protokolls (Scheduler, Manchestercode, etwas Semantik) werden dem Anwender und dem PC abgenommen.

5.3 Weitere Ideen für Ansteuerung

Wir vermuten, dass es möglich ist, die Pegel der seriellen Schnittstelle manuell zu setzen. Wir empfehlen die Datei *„./drivers/net/hamradio/baycom_ser_xxx.c“* im Linux Kernel anzusehen. Hier wird ein Frequenz Shift Keying IC angesteuert, der unseres Erachtens einen ununterbrochenen Bitstrom erhält.

6 Bahnlogger

Der Begriff Bahnlogger ist historisch. Dieses Modul entstand ursprünglich um die Daten der Original Carrera Bahn auszulesen. Später wurde es als Client erkannt. Das heißt, dass es im Car-Controller für die Dekodierung der Signale verwendet werden kann. In unserem Rückkanal wurde es effektiv eingesetzt und hat sich dort bewährt. Das Problem, aus dem der Bahnlogger entstand, war dass die Manchestersignale genauer abgetastet werden müssen, als wir es mit dem PC und der IO-Karte hätten realisieren können.

6.1 Konzept

Im Bahnlogger triggert die Bahn mit einem externen Interrupt (z.B.: INT0) eine ISR. Diese startet den Abtastvorgang des Manchestercodes. Der Abtastvorgang ist timergesteuert. Dieser Timer wird mit jeder Taktflanke synchronisiert. Das Ende einer Manchester-Sequenz wird durch eine Manchestercodeverletzung erkannt.

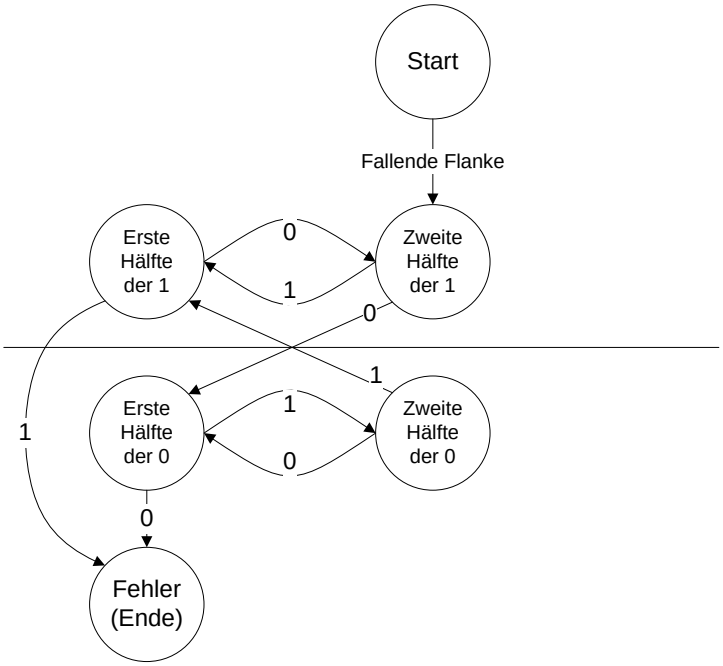


Abbildung 26: H-Brücke Schema

Da ein Flankenwechsel in jeder Mitte eines Bits auftritt, können maximal zwei Abtastungen ohne Kalibrierung erfolgen. Das Ergebnis sollte ein zuverlässiger Dekodierer sein.

6.2 Realisierung

Bei der ersten Realisierung benötigten wir keine Spannungswandlung, da wir den Bahnlogger direkt an den 5V Ausgangspin des Carrera-Blackbox-Atmel-Prozessors angeschlossen hatten. Die Software wurde auf dem Atmega16 und dem stk500v2 Starterkit entwickelt. Sie setzt sich aus 3 Komponenten zusammen:

- **External Interrupt ISR:** Der External Interrupt dient dem Start der Decodierungssequenz und dem Nachsynchronisieren.
- **Timer Interrupt ISR:** Der Timer Interrupt tastet das Signal ab und decodiert dabei den Manchestercode. Hier wurde der oben aufgezeigte Manchesterdekodierautomat implementiert.
- **Hauptschleife:** Die Hauptschleife hat die geringste Priorität. Sie liest, geschützt, Daten aus dem Ringpuffer aus und sendet diese über die UART-Schnittstelle an den PC.

Die Ergebnisse werden im Binärformat über die RS232 an den PC zurückgegeben.

7 Phase 2: Positionserkennung über Carrera System

Neben der Ansteuerung der Bahn ist die Sensorik ein wichtiges Element für spätere Software Projekte. Es soll dem internen System - dem PC - ein Zustand der Bahn bekannt gegeben werden. So ist es möglich auf Ereignisse der Bahn zu reagieren. Uns erschien das bereits bekannte optische Carrera-System während des Semesters realisierbar.

7.1 Konzept TODO Bilder neu

Die original Carrera Autos senden mit ihrer im Boden eingebauten IR-LED ein Rechtecksignal aus. Die Periodendauer hängt von der programmierten CarID ab. Diese optischen Signale werden mit Fotodioden aufgenommen und in elektrische Pegelsignale umgewandelt, die von einem Prozessor verarbeitet werden können. Jede Fotodiode wird mit einem eigenen Prozessoreingang verbunden.
In unserer Implementierung tastet der Prozessor in periodischen Abständen diese Signale ab. Je Port können die Signale von 8 Fotodioden aufgenommen werden. Das von uns realisierte System überprüft 2 Ports und kann somit bis zu 16 Fotodioden überwachen.
Der Prozessor überprüft bei jeder Periode, ob sich ein Port verändert hat. Ist dies der Fall, so wird der Zustand aller Eingänge gemeinsam mit einem Timestamp in den Ringpuffer gespeichert. Der Timestamp wird mit jeder Periode inkrementiert.
Während der Idle-Zeit zwischen den ISRs des Prozessors werden die Daten des Ringpuffers an den PC übermittelt. Am PC werden die Daten ausgewertet. Dabei wird jedes Bit einzeln auf Aktivität überprüft.
Wird eine Aktivität festgestellt, so wird über einen bestimmten Timestamp-Zeitraum hinweg der Flankenwechsel gezählt. Die Anzahl der Flankenwechsel in diesem Zeitraum gibt die CarID an. Bleibt dieser Flankenwechsel nach dieser Zeit bestehen, bleiben die zukünftigen Daten bestehen, bis eine Timeout-Zeit lang keine Flankenwechsel mehr auftreten. Durch den Timestamp ist der zeitliche Versatz der Autos und die Zeitdauer (delta) zwischen den Sensoren ermittelbar.

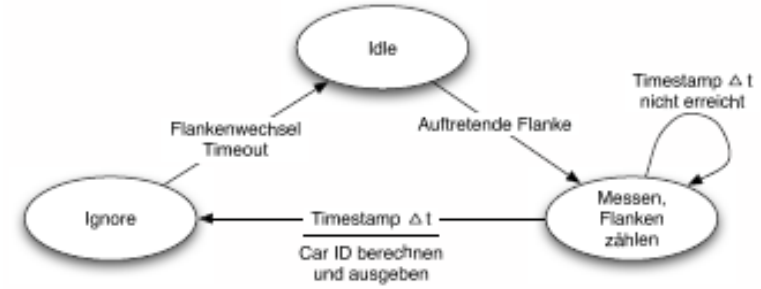


Abbildung 27: Signalerkennungsautomat TODO bessere Qualität

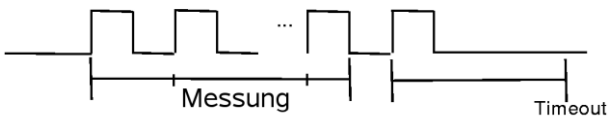


Abbildung 28: Fotodiodensignal TODO bessere Qualität

Mit diesem System ist keine Anpassung der Autos notwendig. Es müssen lediglich die Fotodioden in die Bahn eingebaut/gebohrt - werden.

7.2 Realisierung

7.2.1 Hardware

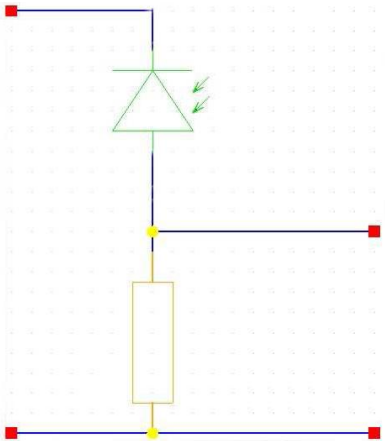


Abbildung 29: Fotodiodenschaltung

Fotodioden sind Dioden, bei denen die Raumladungszone durch eintreffende Photonen beeinflusst werden kann. Treffen keine oder nur geringe Photonen auf sie auf, so sperren sie. Wenn hingegen Photonen auftreffen, so leiten sie einen gewissen Emitter-Collector Strom durch. Vergleichbar mit einem Transistor, nur dass hier die Basis der optische Eingang ist.
Wir entschieden uns für die Fotodiode SFA 309 FA von Reichelt.

I_{pce}	0,4 mA
V_{ce}	35 V
Wellenlänge	730-1100nm
Durchmesser	3mm
Hersteller	OSRAM
Anstiegs- Abfallzeit	7 µs

Tabelle 14: Daten der Fotodiode

Optisch erscheint sie identisch mit den Fotodioden aus den Original Carrera Weichen.

Abbildung 30: Fotodiode SFH309FA

Die kürzeste Periode eines Autos beträgt 63µs. Und da

$$\min\left(Periode_{car}\right) \geq P_{rise/fall} \qquad \textbf{Formel 7–1}$$

ist, ist diese Fotodiode schnell genug.
Diese Dioden werden in Pull-Up Schaltungen verbaut. Ein Widerstand zieht die Signalleitung im Idlezustand gegen Masse. Reagiert die Fotodiode auf Infra-Rot-Photonen, so zieht sie die Signalleitung gegen +5V. Die Dioden-Schaltungen benötigen somit 3 Leitungen. +5V, GND und die Signalleitung. Die Platzierung der Dioden ist in Fahrtrichtung links neben der linken Versorgungbahn. Der Abstand beträgt 5mm von der linken Kante der Leitung bis in den Mittelpunkt der Bohrung.

Abbildung 31: Fotodiode in der Bahn

Die Dioden wurden von unten mit einer Heißklebepistole befestigt. Um Leitungen zu sparen, wurde der Pull-Down Widerstand direkt unter die Bahn verlegt. Alle Signalleitungen und Stromversorgungen wurden mit Pfostenstecker an ein zentrales Flachbandkabel gesteckt. Dieses Flachbandkabel wurde an die Sensorplatine mit einem Atmega16 angeschlossen.

7.2.2 Prozessor

Für die Abtastung wurde ein Atmega16-16 verwendet. Er wurde zunächst mit 11.0592MHz getaktet. Sein Programm besteht aus einer Hauptroutine, und einer Timer-ISR.

Timer-ISR: Die Timer-ISR wird periodisch aufgerufen. Die Periode ist gleich der Abtastfrequenz. Sie lädt das Port Register in den Speicher, und überprüft dieses mit der letzten Messung. Sind die Werte unterschiedlich, so wird der neue Wert zusammenmit dem Timestamp, der durch die ISR kontinuierlich inkrementiert wird, in den Ringpuffer gespeichert. Ist dieser voll, so werden die Daten verworfen. Der neue Wert wird zum Schluss für den neuen Vergleich in der nächsten Periode abgespeichert.

Hauptroutine: In der Hauptroutine wird geprüft, ob Daten im Ringpuffer vorhanden sind. Ist dies der Fall, so werden diese in einem binären Format an den PC gesendet.

Das binäre Format ist 6 Byte lang.

Timestamp	uint16 t
$Daten_a$	uint8 t
$Daten_b$	uint8 t
Magic	uint16 t

Tabelle 15: Datenformat in Hauptroutine

Die Bytefolge ist Little-Endian. Das Magic Feld ist mit den Wert 0xdeaf gefüllt. $Daten_a$ und $Daten_b$ beinhalten die Pegelstände High/Low, also 1/0, der Fotodioden pro Bit. Die UART Übertragungsgeschwindigkeit zwischen Prozessor und PC beträgt 115200 Baud. 8 Datenbits, 1 Stoppbit und keine Parität - 8N1.

Beschreibung der Anschlüsse:

X2: Stromversorgung passend für einen 7805 - 12V ist OK

X1: Serielle Schnittstelle - Sensordatenausgang für den PC

SV1 - SV4: Sensor Anschlüsse mit GND und 5V+

JP1: ISP Schnittstelle für ATMEL Programmierung

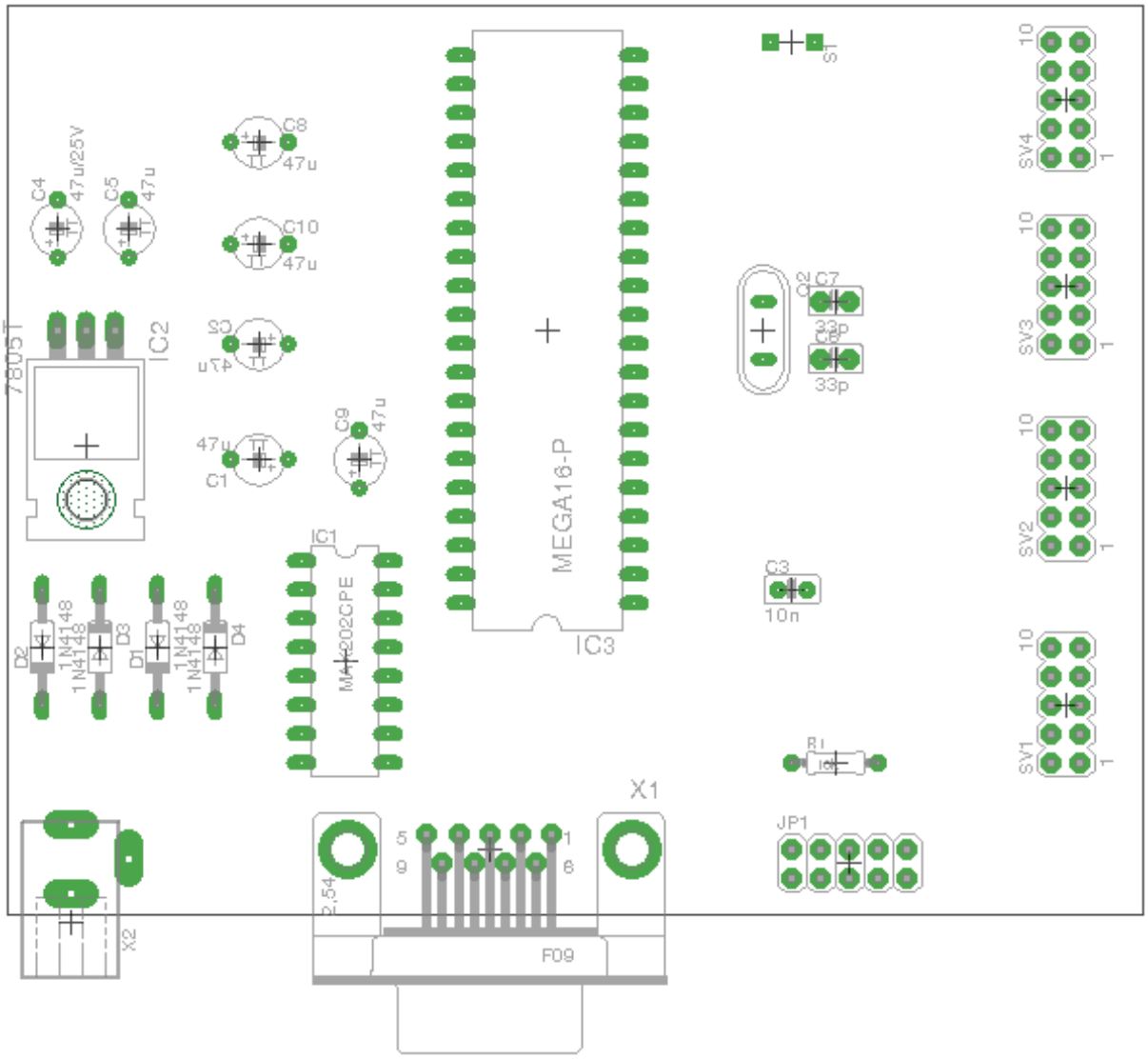


Abbildung 32: Sensorboard Layout

7.2.3 Prozessorkernauslastung

Das Timer Register wird bei Beginn jeder ISR mit dem Wert 229 geladen. Daraus ergibt sich eine Zählweite von 26. Der Timer-Multiplikator ist auf 8-fach gestellt. Das bedeutet, nach jeweils 8 Takten wird der Zähler inkrementiert. Tritt ein Interrupt auf, so werden die Register auf den Stack gespeichert bevor das Timer-Register gesetzt werden kann.

$$t_p = takte_{timer} + takte_{stacksichern} = 26 \cdot 8 + 56 = 264\ takte$$ **Formel 7–2**

Der Worstcase beträgt 278 Takte und setzt sich aus den Befehlsfolgen der ISR-Routine zusammen.

$$u = \sum_{i=1}^{k} \frac{t_{e_i}}{t_{p_i}} = \frac{278}{264} = 1{,}053 \not\leq 1$$ **Formel 7–3**

Dies bedeutet, dass der Prozessor nicht mit Garantie alle ISRs rechtzeitig bearbeiten kann. An jeder Variablen sind Verbesserungen möglich.

Schnelleres Quarz: Die Taktfrequenz kann von 11,0592MHz auf 16MHz erhöht werden. Dies ist eine Beschleunigung um 44%.

Herabsetzen der Abtastperiode Die bisherige Abtastperiode beträgt:

$$t_{abtast} = \frac{t_p}{f} = \frac{264}{11{,}0592 \cdot 10^6 \frac{1}{s}} = 23{,}87152\ \mu s$$ **Formel 7–4**

Benötigt wird aber nur ein Quant mehr als die doppelte Abtastrate - Nyquist-Shannon-Kriterium. 63µs ist die kürzeste auftretende Periodenlänge an einem Port. Also würde eine Abtastrate von

$$\frac{t_{periode}}{2} - 1 = \frac{63\mu s}{2} - 1 = 30{,}5\ \mu s$$ **Formel 7–5**

genügen.
Wir haben unsere endgültige Platine daher mit einem 16MHz Quarz bestückt.
Die Timer-ISR stellt eine höherpriore Task als die Hauptroutine dar.
Ein Flaschenhals stellt der UART Kanal dar. Um ihn effizienter zu benutzen, wurde das vorherige ASCII Format - die Werte wurden als ASCII Strings übertragen - , durch das oben beschriebene Binär-Format ersetzt.

7.2.4 Ringpuffer

Der Atmega16 Prozessor verfügt über 1k Arbeitsspeicher. Dieser Speicher wird auch als Stack benutzt.
Daher ist es einerseits wichtig die Funktionsaufrufstiefe gering zu halten und den Ringpuffer nicht zu groß werden zu lassen. Ein Überschreiben des Stacks führt zu Fehlern. Diese Fehler lassen ähnlich wie Pointerfehler nicht direkt auf das Problem schließen. Der Ringpuffer speichert folgende Daten in einem Array der Länge 190 ab.

Timestamp	uint16 t
Daten$_a$	uint8 t
Daten$_b$	uint8 t

Tabelle 16: Daten, die der Ringpuffer speichert

Dabei können maximal (*Arraylänge - 1*) Werte gespeichert werden.

7.2.5 Software

Für die ersten Auswertungen wurde ein TCL-Skript verwendet. Es arbeitete im ASCII Format. Später wurde dieses Script für die Bibliothek in C Implementiert. (Siehe Kapitel „5.2.5.3 csdlib“)

7.3 Probleme

Die Auslastung des Prozessors hängt von mehreren Faktoren ab:

1. **Anzahl und Abstand der Autos** - je mehr Autos dichter aufeinander fahren umso öfter treten Ereignisse / Flankenwechsel auf.
2. **CarID Periodenlänge** - bei einer niederen Periodenlänge vergrößert sich:

$$\frac{t_e}{t_p}$$ **Formel 7–6**

3. **Anzahl und Abstand der Sensoren** erhöht die Erhebung von Daten.

Diese Faktoren führen zu einer WorstCase-Fehlerrate von 20% bei 16 Sensoren und Autos geringer CarIDs.
Bei Autos mit größerer CarID geht die Gesamtfehlerrate gegen die optische Fehlerrate.

7.4 Verbesserungen TODO Bilder bessere Qualität

Im Rahmen dieser Phase wurden verschiedene Lösungsansätze angedacht. Keiner davon wurde realisiert. Das Problem in der bisherigen Lösung war die Asynchronität der Daten. Dies erfordete ein stetiges zyklisches Abtasten der Eingänge. Die Zeit, die für die Auswertung benötigt wird, multipliziert sich mit der Anzahl der Ports. Dieses System ist paralleler Natur. Daher wurden parellel-verarbeitende Lösungen angedacht.

FPGA: In einem FPGA könnte auf Flanken getriggert werden. Ein Zeitzähler beginnt zu zählen; nach einer gewissen Zeit werden die aufgetretenen Flanken summiert und mit dem Timestamp in einem FIFO gespeichert. Dies geschieht pro Port. Eine Zentrale ruft die Daten - sofern vorhanden - aus jedem FIFO ab und sendet sie via UART, USB etc. an den PC. Die Zentrale serialisiert also die Daten. Für eine geringe Anzahl von Sensoren ist diese Lösung kostenintensiv.

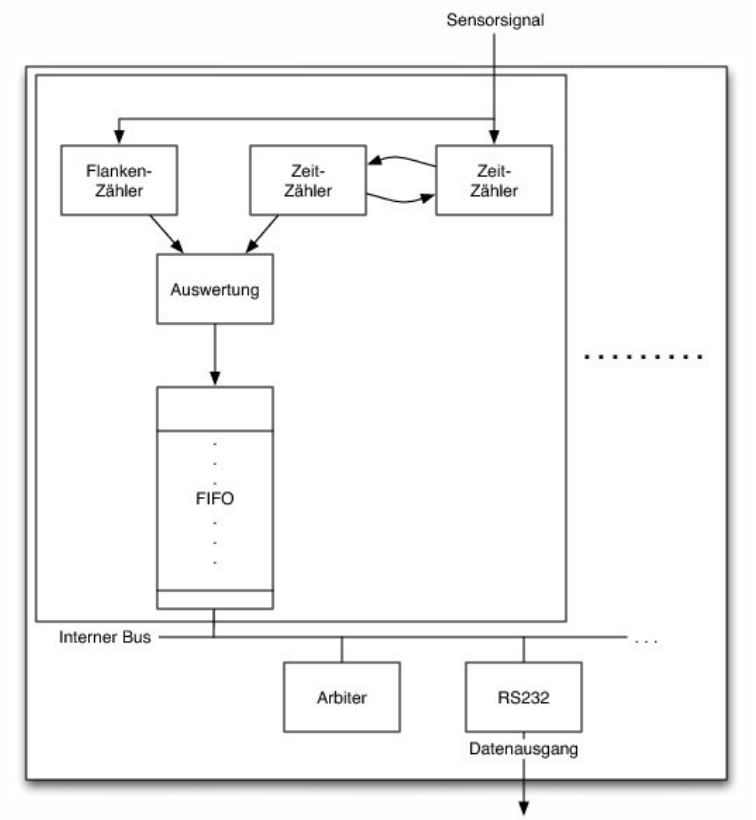


Abbildung 33: FPGA Positionserkennung TODO bessere Qualität ??

Verteilte Auswertung: Je Fotodiode wird ein ATTiny vorgeschlagen. Die Diode soll an den Pin INT0 angeschlossen werden. So kann die Fotodiode einen Interrupt triggern. Hier wird ein Zähler inkrementiert. Bei jeder Flanke wird ein Timeout gesetzt. Läuft dieser ab, so wird das Ende der Sensorsequenz signalisiert. Das Auto fährt also aus dem Sensorbereich heraus. Ist das Ende erreicht, so wird der Flankenzähler durch die Aufnahmezeit geteilt. Das Ergebnis wird direkt in eine CarID umgewandelt und in einem Ringspeicher gesichert. Eine zentrale Instanz, etwa ein Atmega16, liest die Daten aus. Dies kann als Bus mit Arbiter realisiert werden, oder über CAN. Die Kosten pro Sensor belaufen sich etwa auf *1,05 (ATTiny) + 0,26 (Fotodiode) + 0,01 (Widerstand) = 1,32 Euro* pro Sensor.

7.5 Fazit

Die Positionssensorik ist im Worstcase sehr fehleranfällig. Dennoch konnten wir mithilfe dieser Sensorik interessante Experimente durchführen. (Siehe Kapitel „5.2.5.1.2 Kollisionserkennung“). Die Machbarkeit der Positionserkennung mithilfe optischer Aufnehmer, die das Carrera Signal der Autos

auffangen, ist somit möglich. Mithilfe der vorgeschlagenen Verbesserungen lässt sich die Fehlerrate jedoch erheblich senken.

8 Phase 3: Rückkanal

Unter Rückkanal wird eine bidirektionale Verbindung zu den Autos verstanden. Doch wozu braucht man einen Rückkanal? Diese bidirektionale Verbindung hat den Zweck Sensordaten aus den einzelnen Autos auszulesen und auszuwerten. Diese Daten können von Beschleunigungssensoren, Kielauslenkungen, Geschwindigkeiten etc. stammen. Der Kanal ist auf geringe Datenmengen und Übertragungsraten beschränkt, dafür aber deterministisch. Dieses Thema wurde als Machbarkeitsstudie durchgeführt. In Laborversuchen wurde nachgewiesen dass der Rückkanal realisierbar ist. Zunächst wird das Konzept erläutert. Im Anschluss daran wird der Laborversuch mit den Aufbauten beschrieben. Als Abschluss wird das Ergebnis diskutiert.

8.1 Konzept

Um eine integrierte, das bedeutet eine nicht über andere Kanäle wie Funk etc., Lösung zu erhalten, soll der Datenaustausch ausschließlich auf dem Carrera Bussystem stattfinden. Das bedeutet, dass der Kanal als Steuerung der Weichen und Autos, zur Stromversorgung, nun zusätzlich für das Aufsetzen von Sensordaten aus den Autos heraus benutzt wird. Das bisherige Carrera System beruhte auf Rechteck-Pegelsignale. Dies wurde bei unserem Rückkanal beibehalten, da wir uns dadurch eine höhere Signalfestigkeit erhoffen. Die Überlegung Signale kapazitiv, wie es zum Beispiel Powerline macht, einzukoppeln bestand, wurde aber verworfen. Es wird vermutet, dass die Dämpfung und die zusätzlichen Induktionsstörungen der Motoren eine Signaldetektierung erschweren.

Das Konzept lässt sich in zwei Schichten - ähnlich dem ISO/OSI Referenzmodell - beschreiben.

- **Layer2:** Der Server sendet ein Reglerdatenwort an ein bestimmtes Auto,welches auf das Reglerdatenwort antworten kann. In dieser Schicht stellt die AutoID eine Adresse dar. Das Auto kann nur dem Server antworten. Pollingbetrieb.
- **Layer1:** Der Server legt die Daten im Manchestercode auf den Bahnbus. Die Pegel betragen 0 und 14,7V. Eine feste Zeitlänge von xxx µs nach Beginn des Reglerdatenwortes verpolt der Server die Leitungen und schaltet die Versorgungsspanung für einige µs hochohmig. Der Client kann durch Kurzschließen seine Signale in dieser Phase absetzen.

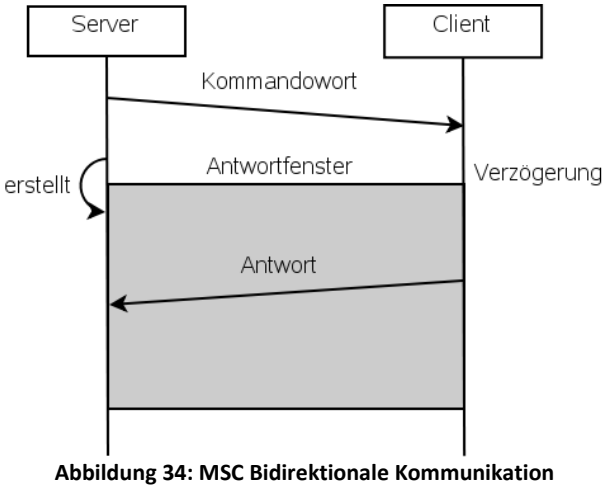


Abbildung 34: MSC Bidirektionale Kommunikation

In dieser Abbildung ist der Kommunikationsablauf zu erkennen. Der Server sendet ein übliches Reglerdatenwort. Danach wird eine bestimmte Zeit - relativ zum Kommandowortbeginn - gewartet. Dann legt der Server ein Rückkanalfenster auf. Der Client kann hier nun seine Daten absetzen, die der Server empfängt und auswertet.

8.2 Praxis

Der bisher verwendete ATMEL-Server wird erweitert. Zum Gesamtsystem kommt ein selbstentwickelter Client hinzu.

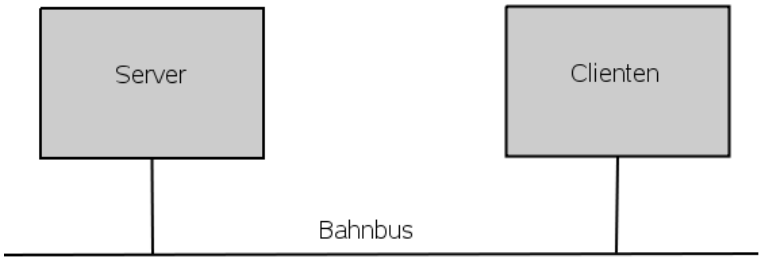


Abbildung 35: Rückkanal Gesamtsystem

Der ATMEL-Server wird zunächst um eine Implementierung für die Bereitstellung eines Rückkanals erweitert. Nach einem Steuerkommando wird ein Fenster eingelegt, in dem die Bahn über eine H-Brücke verpolt und hochohmig geschaltet wird.

8.2.1 H-Brücke TODO Bilder

Die H-Brücke (siehe [HBRCK]) dient hauptsächlich der Ansteuerung von Motoren. Mit ihr kann eine Gleichspannung in eine Wechselspannung umgewandelt werde. Der Wechsel kann gesteuert werden. In unserer Anwendung wird diese H-Brücke dazu verwendet, die digitalen Signale auf die Bahn zu legen und für das Rückkanalfenster die Spannung des Bahnbusses umzupolen. Eine Seite der H-Brücke entspricht der ursprünglichen Treiberschaltung die andere Seite der H-Brücke ist ähnlich. Sie steuert jedoch die Leistung gegen den negativen Pol. Daher ist hier ein N-Channel FET verbaut.

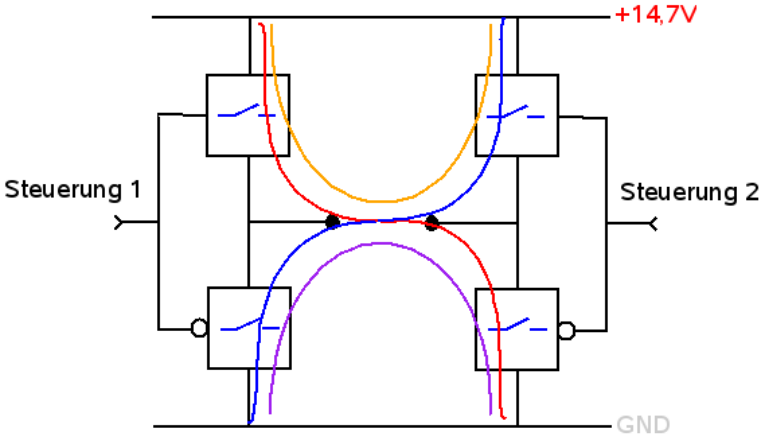


Abbildung 36: H-Brücke Schema

In dieser Grafik ist die H-Brücke zu erkennen. Der linke Teil ist die ursprüngliche Treiberschaltung. Der rechte Teil kommt neu hinzu. Mit dieser Brücke werden nun die digitalen Signale aufgelegt und, nun neu, die Bahn für das Rückkanalfenster umgepolt. Die Schaltzustände der Brücke sind:

Linke Brücke	Rechte Brücke	Verwendungszweck
14,7 V	0 V	Versorgungsspannung, Carrera Signal
0 V	0 V	Carrerasignal, Bahnabschalten
14,7 V	14,7 V	Wird nicht verwendet
0 V	14,7 V	Rückkanalfenster

Tabelle 17: Schaltzustände mit Verwendungszweck

Es wurde ein ähnlicher FET wie in der Spiegelschaltung, verwendet. Der Unterschied liegt im Kanal, also der Dekodierung. NMOS statt PMOS. Er ist in etwa gleich dimensioniert wie der IRFR9024. IRLR120N (M1) ist ein 10A selbstsperrender N-Kanal FET. Die Bipolartransistoren die nun die in

Fahrtrichtung linke Spurleitung gegen Positiv ziehen sind nun PNPs (Q3 und Q4). Der Widerstand (R7), der hier in Reihe geschaltet ist, sollte kleiner gewählt werden. Der 120 Ohm Widerstand der NPN Schaltung, die die linke Spurleitung gegen Negativ (Masse) zieht ist für den Rückkanal hochohmig genug um ein Kurzschlusssignal absetzen zu können. Die Steuerleitung für diesen H-Brückenteil ist auf 5V ausgelegt und wird an einen separaten Pin des Prozessors angelegt.

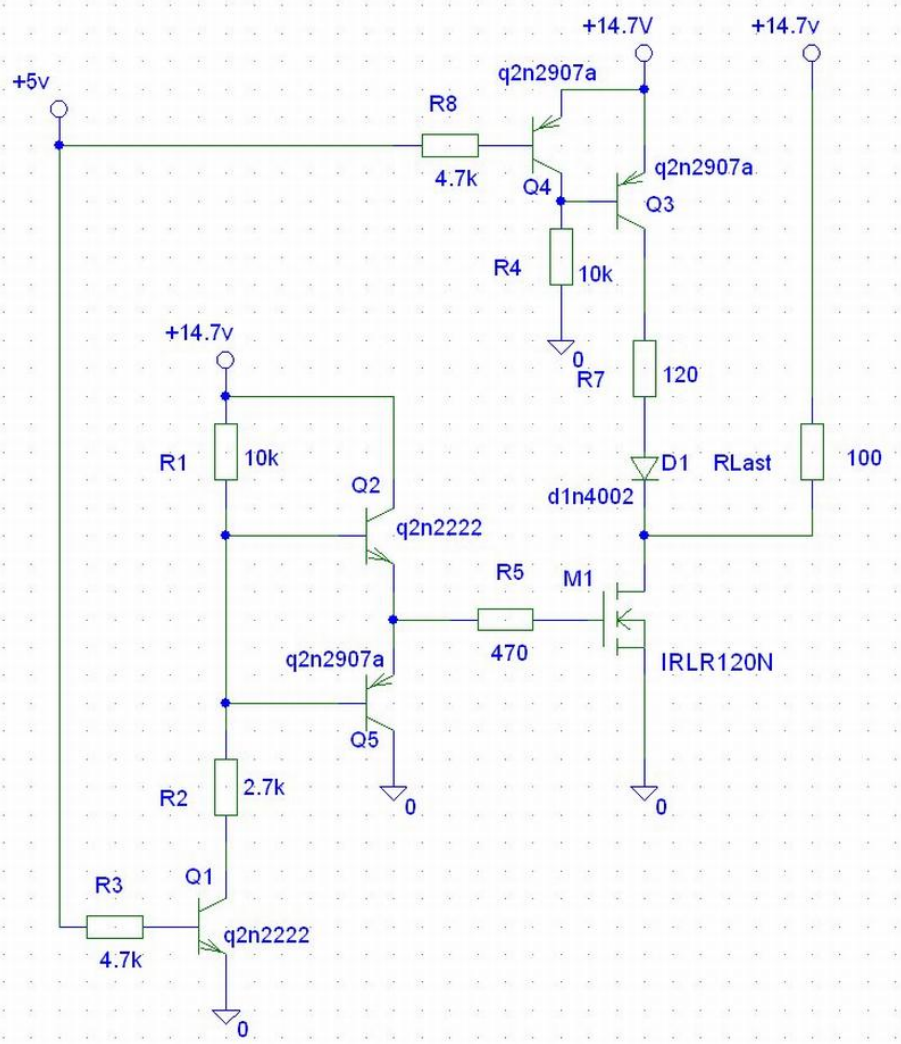


Abbildung 37: N-Kanal Treiberschaltung

8.2.2 Server Implementierung

Der Server zählt in seiner TimerISR relativ zum Kommandobeginn hoch. Hat dieser Zähler eine gewisse Summe erreicht - im derzeitigen Fall 60 - so schaltet er den Port der die linke Fahrbahn ansteuert gegen positiv und die rechte Fahrbahn gegen negativ. Es geschieht also eine Umpolung. Da nun der 120 Ohm Widerstand bei den NPN-Bipolartransistoren in Reihe geschaltet ist, ist die Spannung hochohmig. Als zukünftige Implementierung müsste hier nun das Empfangssignal abgetastet werden, ähnlich wie dies der Bahnlogger tut.

8.2.3 Client

Der Client - das Auto - nimmt das Signal vom Bahnbus ab. Elektrisch geschieht dies durch einen mit einer Diode gesicherten Spannungsteiler. Er ist auf etwa 10mA dimensioniert und soll die 14,7V Steuer- und Versorgungsspannung des Bahnbusses auf 5V für den Atmel-Eingabe-Port herabsetzen.

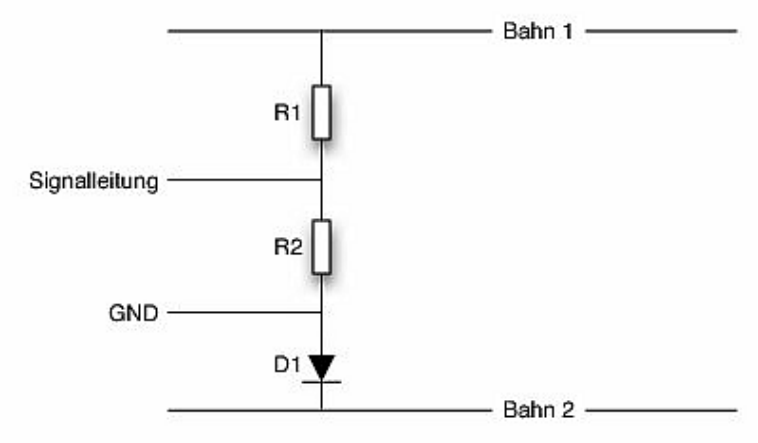


Abbildung 38: Signalabnehmer TODO bessere Qualität

Die Atmel Software beruht auf dem Bahnlogger. Zusätzlich werden nun aber auf einer höheren Schicht die Daten interpretiert. Der Prozessor wertet auf Grund der Bitlänge den Kommandotyp aus. Dann Filtert er die CarID heraus, und überprüft seine Zuständigkeit. Nun kann der Client in seiner INT0 ISR - die auch auf den Anfang der Kommandoworte triggert und mit den Manchesterflanken nachjustiert - zusätzlich auf die Pausenflanke warten. Hierzu wurde ein Zustandsautomat mit drei Zuständen verwendet. Ist er im Zustand *senden* so kann er durch Kurzschließen der Bahnleitungen ein Signal aufsetzen. Da nun die Bahn relativ Hochohmig ist, bricht die Spannung ein.

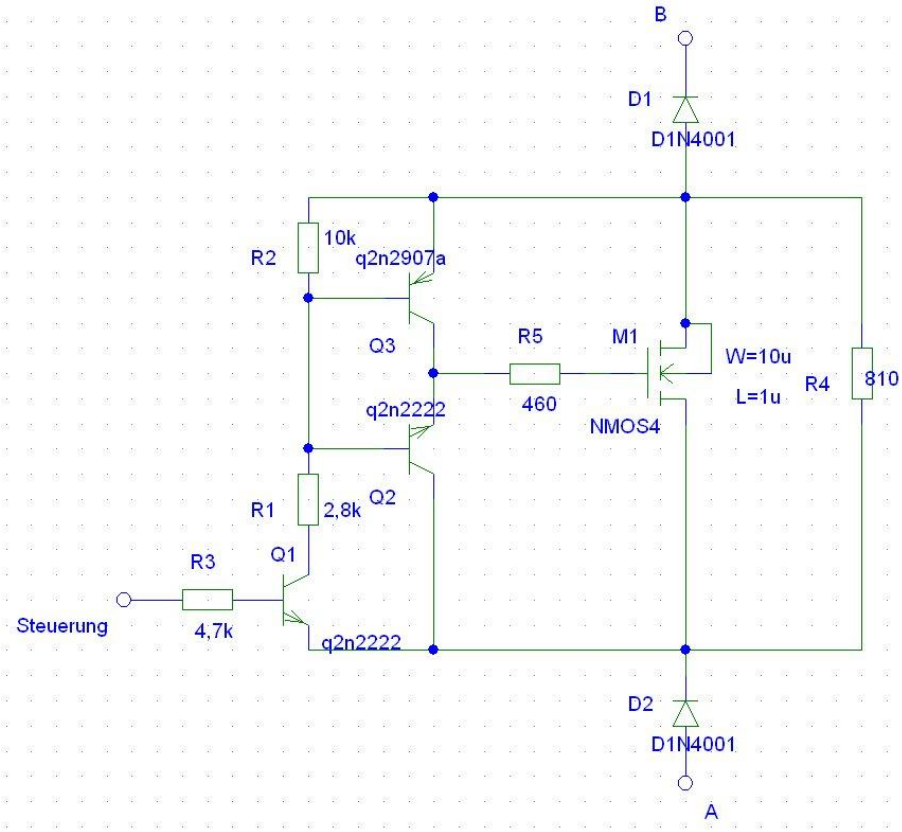


Abbildung 39: Kurzschließer

Dieser Kurzschließer ist in unserem Versuch strommäßig überdimensioniert. Er muss keine 11A schalten, da die Leitungen hochohmig sind (Erinnerung 2x120Ohm). Der Kurzschließer wird mit 5V gesteuert.

8.2.4 Server Empfangsteil

Der Empfangsteil wurde von uns nicht aufgebaut. Daher ist der folgende Abschnitt nur ein unverifizierter Vorschlag. Auf der Serverseite kann der einbrechende Pegel über eine Komparator-Operationsverstärkerschaltung an einen Atmel-Port geführt werden. Der Komparator vergleicht die Spannung der linken Spur mit einer Referenzspannung eines Spannungsteilers.

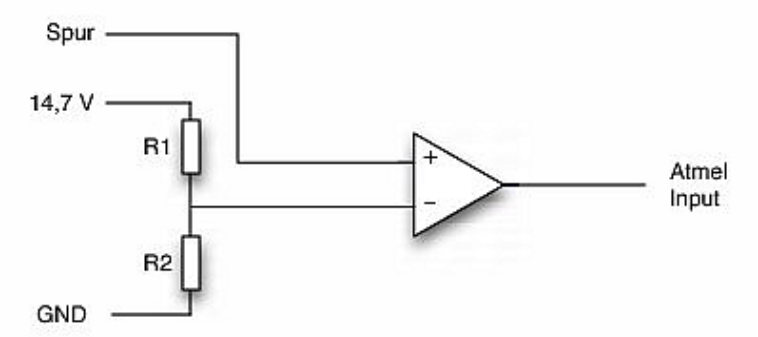


Abbildung 40: Komparator des Server-Empfangteils TODO bessere Qualität

Durch einen Interrupt auf einen Flankenwechsel kann die Empfangsdecodierung getriggert werden. Zum Dekodieren des Rücksignals könnte wiederrum der Bahnlogger verwendet werden.

8.3 Versuchsaufbau und Messungen

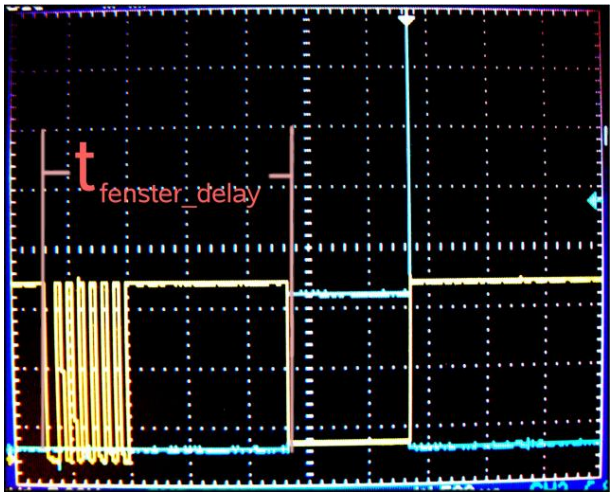


Abbildung 41: Rückantwortfenster

Dieses Bild zeigt das vom Server aufgesetzte Rückantwortfenster. Die Zeit $t_{fenster\ delay}$ ist ein im Code einstellbarerWert.

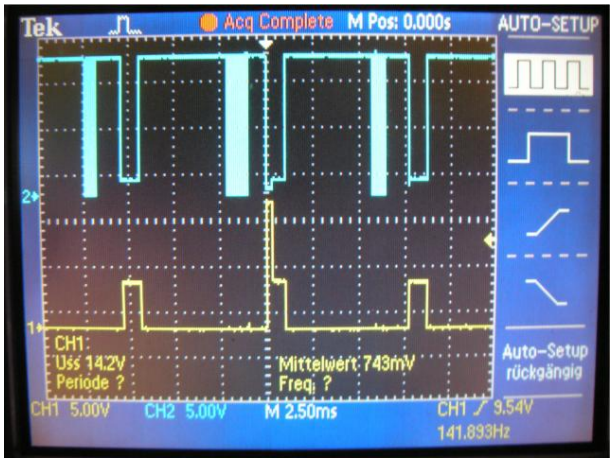


Abbildung 42: Reglersignale bei Rückantwortfenster

In dieser Abbildung wurde ein Bit vom Client in ein Rückantwortfenster gesetzt. Es sind drei Reglersignale zu sehen (blaue Balken). Das mittlere ist ein Reglersignal der CarID 3. Das Clientprogramm wurde auf die Beantwortung eines Reglersignals dieser CarID konfiguriert. Außer diesem gibt es keine anderen Clienten. Das Ergebnis ist ein gesetztes Bit im mittleren Antwortfenster. Dies beweist, dass es praktisch möglich ist, auf demselben Bahnbus Signale zurückzusenden.

8.4 Status

Das Rückkanalfenster wurde in die Atmel-Steuerung integriert. Die Dekodierung findet nicht statt. Auf der Platine sind Slots für die benötigten Hardwaremodule vorhanden. Es wird noch der restliche Teil der H-Brücke sowie das Server Empfangsteil benötigt. Die Anschlüsse sind im Kapitel „5.2 Ansatz: Atmel“ aufgeführt.
Der Hardwareteil des Clients wurde nur auf dem Steckbrett realisiert. Die Software Implementierung ist vorhanden.

8.5 Fazit

Es zeigt sich, dass es möglich ist, Daten von den Clients zu dem Server zu senden. Die Übertragungskapazität ist begrenzt. Sie beeinflusst die Leistungsübertragung. Die Datenmenge würde für kleinere Sensordaten genügen. Durch das fest vorgegebene Server-Protokoll ist der Rückkanal deterministisch. Die Sende- und Empfangspakete werden zu vorhersagbaren Zeiten übermittelt.

8.6 Zukünftige Ideen

Theoretisch sollte es möglich sein, eine Art CAN-Signal über dieses Zeitfenster zu versenden. Man könnte also priorisierte Signale von beliebigen Autos, Weichen oder gar Bahnsensoren in einem CAN-Frame versenden. Die Realisierung hängt von der verwendbaren Übertragungsgeschwindigkeit ab. Rezitiv würde keinem Kurzschluss entsprechen, bei einem Kurzschluss, wäre das Signal dominant.

9 Fazit des Projektes TODO Formulieren

Um unsere gesamten Ergebnisse bei dieser Projektarbeit zusammenzufassen, können wir sagen, dass die Projektarbeit sehr erfolgreich war. Wir konnten beweisen, dass es möglich ist, die Carrera-Bahn über den PC anzusteuern. Dies haben wir nicht nur bewiesen, sondern die Ansteuerung auch umgesetzt. Zudem haben wir festgestellt, dass auch ein Rückkanal machbar wäre. Dieser wurde von uns nicht komplett umgesetzt, aber die Ansätze existieren. Desweiteren haben wir eine Positionserkennung eingebaut, die momentan noch nicht 100%ig funktioniert, aber ebenfalls machbar ist.

Wir, also die Teammitglieder, haben bei diesem Projekt sehr viele interessante Dinge gelernt. Die Arbeit an diesem Projekt hat uns sehr viel Spaß gemacht. ?? sollen wir so was schreiben???

Benny formuliert hier noch was!!!

10 Literaturverzeichnis

[SLOTB]	Stephan Heß: D132 Datenprotokoll physikalisch Internet: http://www.slotbaer.de/D132/PhyProt.html Stand: 28.05.2008
[HBRCK]	Wikipedia: Vierquadrantensteller Internet: http://de.wikipedia.org/wiki/Vierquadrantensteller Stand: 18.06.2008

Anleitung:

1. [KÜRZEL] als Textmarke hinzufügen; am besten auch unter diesem als Namen
2. Aufbau der Quellenbeschreibung bei Büchern:
    a. Name des Autors, 1.Buchstabe des Vornamens (bei mehreren Autoren mit ; trennen)
    b. :
    c. Name des Buches
    d. ;
    e. Verlag Erscheinungsjahr
3. Aufbau der Quellenbeschreibung bei Internetseiten/pdfs:
    a. Name des Autors (sofern vorhanden; Angabe wie bei Buch)
    b. :
    c. Titel der Seite /des pdfs
    d. ;
    e. Typ: URL
    f. Stand: Datum
4. Über Querverweis kann jetzt auf die entsprechende Textmarke an der gewünschten Stelle verwiesen werden.

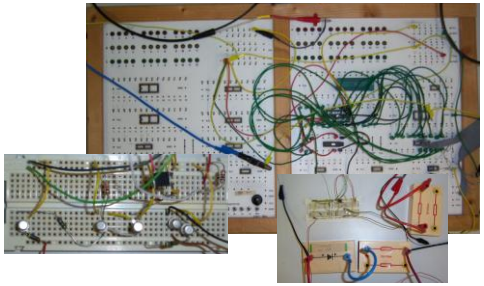
11 Weitere Verzeichnisse

11.1 Abbildungsverzeichnis

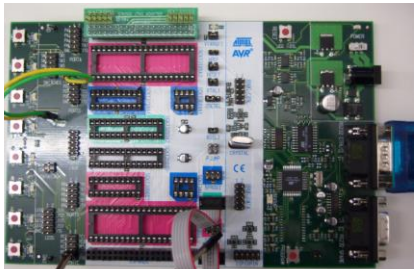
11.2 Tabellenverzeichnis

11.3 Formelverzeichnis

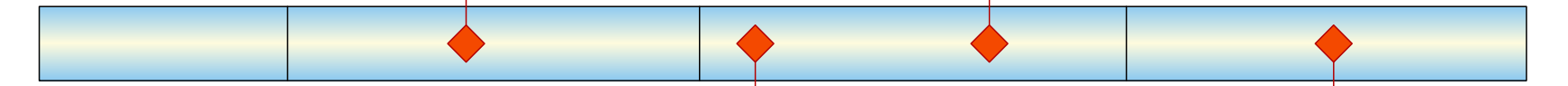
12 Anhang



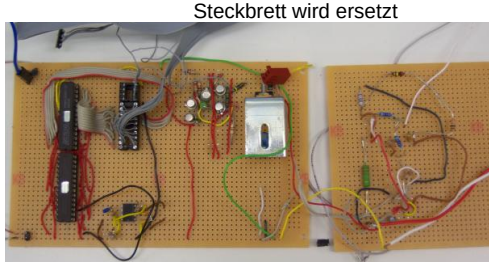
14.04.08
Erster Schaltungsaufbau mit Steckbrett



22.05.08
Umstieg auf Atmel



05.05.08
Steckbrett wird ersetzt



16.06.08
Platinen eingetroffen

Hier kommen noch schaltungen von Jules rein!!