Entwicklung eines batteriebetriebenen Analysesystems zur Kfz-Klassifizierung

Diplomarbeit

durchgeführt von

Thomas EIBEL

Institut für Hochfrequenztechnik
der Technischen Universität Graz

Leiter: Univ.-Prof. Dr. Wolfgang Bösch

Betreuer: Ao.Univ.-Prof. Dr. Erich Leitgeb
Beurteiler: Ao.Univ.-Prof. Dr. Erich Leitgeb

Graz, im März 2012

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am ....
(Unterschrift)

Englische Fassung:

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

....
date (signature)

Kurzfassung

Die vorliegende Arbeit konzentriert sich auf die quantitative Verkehrsmessdatenerfassung mittels Überkopfdetektor (Doppler-Radar, Infrarot und Ultraschall) hinsichtlich der diesbezüglichen Vorgaben der TLS (Technische Lieferbedingungen von Streckenstationen) der Bundesanstalt für Straßenwesen (BASt).

Die Hardware bestehend aus Detektor, Steuerungs- und Speicherplatine ist in der Lage energieautark, das heißt ohne Anbindung an das Energienetz, effizient und kostengünstig Daten aufzunehmen und zur weiteren Analyse via Software abzuspeichern.

Die Software ist in der Lage die gemessenen Daten aufzubereiten und im gewünschten Zeitfenster grafisch darzustellen. So kann bei genügend großem Datenvorrat eine präzise Aussage bezüglich der Verkehrsauslastung getätigt werden.

Zu Beginn der Arbeit wurden die theoretischen Realisierungsmöglichkeiten untersucht. Nach einer Analyse bezüglich der Realisierbarkeit und Auswahl der zu verwendenden elektronischen Komponenten, wurde das Messsystem entworfen. Abschließend erfolgte die Verifizierung der Richtigkeit der Messung mittels der gewonnen Messergebnisse.

Abstract

This thesis focuses on the quantitative data acquisition of traffic measures by the use of Doppler radar, infrared and ultrasonics with respect to the specifications of TLS (Technische Lieferbedingungen von Streckenstationen) of Bundesanstalt für Straßenwesen (BASt).

The hardware, which consists of detector-, control- and memory units, collects data energy-self-sufficiently and cost-effectively. Additionally, it saves data for further analysis via software.

The software edits data and displays it within the desired time frame. Due to hard- and software, a safe conclusion concerning the traffic load could be done if there are large amounts of data stored available.

At the beginning of the thesis, a feasibility study was conducted analysing the feasibility options and the electronical devices which could be used. After the electronical device had been chosen, the measuring system was designed. Finally, the measurement results were verified.

Danksagung

Vor allem bedanken möchte ich mich bei meinem Diplomarbeitsbetreuer an der Technischen Universität Graz Dr. Erich Leitgeb für den großen Freiraum, den er mir bei der Gestaltung und praktischen Durchführung dieser Arbeit gewährt hat.

Ein ganz besonderer Dank gilt meinem Arbeitgeber Abgeordneter zum Nationalrat Mag. Michael Schickhofer für die Möglichkeit der „Freien Zeiteinteilung“.

An dieser Stelle möchte ich auch dem Vizebürgermeister der Marktgemeinde Sinabelkirchen, Herrn Anton Kalcher, meinen Dank aussprechen. Ohne sein Zutun wäre eine behördliche Bewilligung für die Messung auf der Straße wohl heute noch nicht vorliegend.

Ein Dankeschön auch an Dr.-Ing. Tilo Gockel für die Bereitstellung des LaTeX-Templates [1].

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung

Es ist unbestritten, dass ein leistungsfähiges Verkehrssystem ein wichtiges Entscheidungskriterium für wirtschaftliche Entwicklung und Industrieansiedlung und darüber hinaus die Grundlage für die allgemeine wirtschaftliche Entwicklung des Landes darstellt.

Die allgemeine Wirtschaftleistung ist schon seit jeher direkt vom Transportwesen abhängig. So gut diese Transportwege ausgebaut und benutzt werden können, so gut steht es auch um den diesbezüglichen Standort respektive neue Betriebsansiedelungen und neue Arbeitsplätze. Es ist unbestritten, dass ein leistungsfähiges Verkehrssystem ein wichtiges Entscheidungskriterium für wirtschaftliche Entwicklung und Industrieansiedlung und darüber hinaus die Grundlage für die allgemeine wirtschaftliche Entwicklung des Landes darstellt [2].

Nichtsdestotrotz muss man aber zugestehen, dass ein vermehrtes Verkehrsaufkommen auch eine vermehrte Umweltbelastung, aber auch eine größere Gefahr für Menschen darstellt.

Bei einem regionalen Treffen der Steiermärkischen Berg- und Naturwacht kam das Thema „Zunahme der Verkehrsbelastung im Orts- respektive Stadtgebiet“ zur Diskussion in kleiner Runde. Dabei wurde festgehalten, dass das Verkehrsaufkommen (Anm.: Durch unsere doch großflächigen Überwachungszonen und den vom Land übertragenen Aufgaben [3] ist uns doch eine gute Einschätzung bzgl. Verkehrsaufkommens möglich) in Orten bzw. Städten mit Autobahnanbindung und guter regionaler Vernetzung unsere Meinung nach in den letzten Jahren stark zugenommen hat.

1.1 Motivation

So wurde die Idee geboren eine eigene, relativ billige, automatische Überwachungs- respektive Messmöglichkeit zu entwickeln, die es erlaubt zumindest eine quantitative Aussage bzgl. Verkehrsaufkommen zu treffen (Anm.: Gehört im Grunde genommen nicht zu den Aufgaben der Berg- und Naturwacht). Dabei muss festgehalten werden, dass von Seiten der Berg- und Naturwacht kein Auftrag zur Entwicklung eines solchen Systems besteht. Lediglich die Ideenfindung in dieser oben genannten kleinen Runde gab für mich den Auschlag dieses Thema zu wählen.

Laut Statistik Austria belaufen sich die KFZ-Neuzulassungen im Jahr 2010 auf 424.114 was ein plus von ca. 2.2 % zum Vorjahr ergibt [4]. Der schon beinahe als Grundrecht verstandene Gebrauch des Kraftfahrzeugs auch für kurze Distanzen belastet die Umwelt trotz besser werdender Technik [2]. Die Belastung im Stadtgebiet wird aber - meiner Meinung nach - noch durch die Einhebung von Steuern auf die Benutzung bestimmter Straßenbereiche und die

dadurch erreichte Verschiebung des Schwerlastverkehrs auf nicht mautpflichtige Gemeinde-, Landes- und Bundesstraßen erhöht (Mautflüchtlinge).

1.2 Zielsetzung

Nun ist es die Zielsetzung dieser Arbeit ein funktionsfertiges Mikrocontroller gesteuertes Hard-/Software System („Streckenstation light“) zur quantitative Verkehrsanalyse bestehend aus Steuerungsplatine, Speicherplatine, Detektor und der dazu gehörenden Analysesoftware zu entwerfen und umzusetzen um dann in weiterer Folge mittels der gesammelten Daten eine Auskunft bezüglich des Verkehrs innerhalb von Orts-/Stadtgrenzen (auch Verbindungsverkehr ausserorts) geben zu können.

1.3 Aufbau und Kapitelübersicht

Das **zweite Kapitel** (Der TLS Standard) beschreibt die grundlegenden Anforderungen an das System durch das Regelwerk der TLS. Die für diese Arbeit wichtigen Beschreibungen und Funktionen werden dort erörtert. Im **dritten Kapitel** (Funktionsweise und Umsetzung) wird auf die geforderten Eigenschaften des Systems eingegangen. Das **vierte Kapitel** (Hardware) beschreibt die maßgebenden Hardwarekomponenten, die für diese Diplomarbeit entworfen wurden. Das **fünfte Kapitel** (Software) gibt einen Überblick bezüglich der entworfenen Mikrocontroller- und Analysesoftware. Das **sechste Kapitel** (Experimentelle Validierung) vermittelt die getane Arbeit bezüglich der Testläufe im Labor und auf der Straße. Das **siebende Kapitel** (Schlussbetrachtung) umfasst in kurzen Sätzen die Folgerungen aus dieser Arbeit.

Im **Anhang** befinden sich unter anderem bestimmte Quellcodes, Messdaten und die verwendeten Werkzeuge für die Erstellung dieser Arbeit.

2 Der TLS Standard

Die Richlinie „Technische Lieferbedingungen für Streckenstationen“- kurz TLS - ist ein Regelwerk, das von der Bundesanstalt für Straßenwesen (BASt), ansässig in D-51067 Bergisch Gladbach, in Zusammenarbeit mit Industrie und den Länderverwaltungen aufgestellt wurde.

Dieses Kapitel stellt eine kurze Zusammenfassung der TLS und die auf diese Diplomarbeit bezugnehmende Relevanz dar. Sämtliche Informationen sind dem Regelwerk [15] entnommen.

2.1 Inhalt und Zweck der TLS

In diesem Regelwerk sind die notwendigen Festlegungen für Streckenstationenfestgehalten. Neben diesen gelten auch einschlägige Richtlinien und Vorschriften, die in der TLS anhänglich zusammengestellt sind.

Um eine möglichst große Flexibilität gewährleisten zu können, ist der modulare Aufbau zu bevorzugen (vgl. Abb.: 2.1)

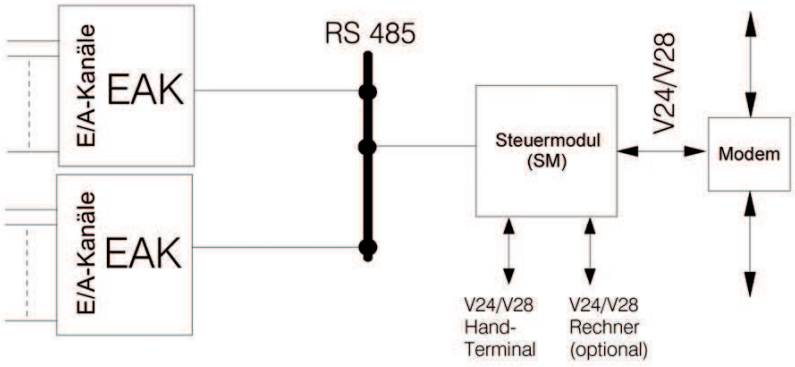


Abbildung 2.1: Streckenstation Variante A - vollmodularer Aufbau mit Std.-Schnittstellen.

Zwischen den Ebenen des Verkehrsleitsystems (vgl.Tabelle: 2.1 auf Seite 15) sind Hardwareschnittstellen vorzusehen. Solche können jedoch innerhalb einer Streckenstation - bei Gewährleistung von Anschlussmöglichkeiten weiterer Datenerfassungs- bzw. Datenausgabegeräte - entfallen.

Die TLS soll prinzipiell eine Beschreibung der anzubietenden Leistungen bieten und so den freien Wettbewerb unterstützen.

2.2 Systembeschreibung

Das gesamte Straßennetz in seiner Größe und die verschiedenen Zuständigkeiten, machen eine hierarchische Untergliederung in einzelne regionsbezogene Teilbereich notwendig. Diese regionalen Netze sind wiederum in einzelne hierarchisch strukturierte Ebenen untergliedert (vgl. Abb.: 2.2 und Tabelle: 2.1 auf Seite 15).

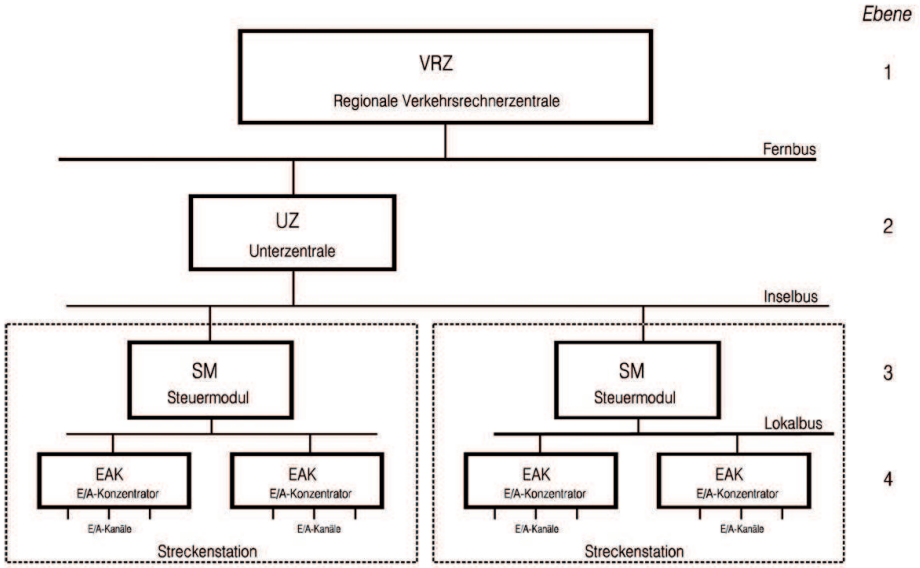


Abbildung 2.2: Hierarchische Ebenenstrukturierung nach dem TLS-Regelwerk

Der Verkehrsrechnerzentrale (VRZ) sind die Unterzentralen (UZ) und diesen die örtlichen Streckenstationen (SS) untergeordnet. Jede Streckenstation enthält ein Steuermodul (SM), das die verschiedenen E/A-Konzentratoren(EAK) bedient. Die Verkehrsrechnerzentrale kommuniziert mit den nachgeordneten Unterzentralen über den Fernbus, die Unterzentrale kommuniziert mit den Steuermodulen der Streckenstationen über den Inselbus und die verschiedenen Datenerfassungs-/Datenausgabegeräte einer Streckenstation kommunizieren über den Lokalbus mit dem Steuermodul.

Die Einrichtung von UZ dient zur Abwicklung der Datenübertragung für geschlossene Netzteile und, sofern vorhanden, zur Steuerung von Wechselzeichenanlagen. Die UZ wird innerhalb des angeschlossenen Teilnetzes (z.B. in einer Autobahn- oder Fernmeldemeisterei) oder in der VRZ eingerichtet.

Die Einrichtungen der Ebenen 3 und 4 (Steuermodul und Datenerfassungs-/Datenausgabegeräte) sind in der Regel an der Strecke in dem gemeinsamen Schaltschrank der Streckenstation zusammengefasst.

Ebene	Einrichtung	Standort
1	Verkehrsrechnerzentrale (VRZ)	Zentraler Punkt im Autobahnnetz
2	Unterzentrale (UZ) dient z.B. zur Steuerung von Wechselverkehrszeichenanlagen und zur Abwicklung der Datenübertragung in geschlossenen Netzteilen	z.B. Autobahnmeisterei oder in VRZ
3	Steuermodul (SM) und Übertragungssystem der örtlichen Streckenstation	Streckenstation
4	Datenerfassungs-/Datenausgabegeräte (DEG/DAG) mit Ein/Ausgabekonzentratoren (EAKs) zur lokalen Datenaggregierung, zur Auswertung der Daten, bzw. zur Übergabe von Parametern und Stellbefehlen	Streckenstation

Tabelle 2.1: Systemebenen laut TLS

2.3 Funktionsverteilung

Jede der oben genannten Ebenen hat spezielle Aufgaben zu erfüllen. Die Funktionen sind aufeinander abzustimmen und so zu verteilen, dass der Übertragungsaufwand die verfügbare Übertragungskapazität nicht übersteigt. Es sind nur die Daten an die jeweils nächste Ebene weiterzugeben, die dort benötigt werden. Es ist vorzusehen, dass von der jeweils übergeordneten Ebene bestimmt wird, welche Daten zu übertragen sind (flexibles System). Aufgaben, die auf den Systemebenen VRZ und UZ vorgesehen sind, sind in dem Merkblatt für die Ausstattung von Verkehrsrechnerzentralen und Unterzentralen beschrieben.

Auf den Ebenen 3 und 4 der Streckenstationen sind die Hauptfunktionen Steuermodul (SM) und Ein-/Ausgabekonzentrator (EAK) angesiedelt. Das Steuermodul hat die Aufgaben der Steuerung des Datenaustausches zwischen Unterzentrale und EAK, sowie die Handhabung des Abfragealgorithmus und des Übertragungsprozederes für den EAK.
Der Ein-/Ausgabekonzentrator übernimmt die Aufgaben der Erfassung und Aggregierung von Verkehrs- oder Umfelddaten der Sensoren, der Funktionsüberwachung und Statusmeldungen und der Weitergabe von Steuerungsbefehlen an Wechselverkehrszeichen.

Die Einrichtungen der Ebenen 3 und 4 (Steuermodul und Datenerfassungs-/Datenausgabegeräte) sind in der Regel an der Strecke in dem gemeinsamen Schaltschrank der Streckenstation zusammengefasst. Die Einrichtung von UZ (Ebene 2) ist zwingend erforderlich. Sie dient u.a. zur Verdichtung der Daten und Reduktion des Übertragungsaufwands. Von der Unterzentralen werden auch, sofern vorhanden, Wechselverkehrszeichenanlagen gesteuert.

Hier ist anzumerken, dass die Steuerungseinheit dieser Arbeit streng genommen diesen mo-

dularen Aufbau bzgl. eines SM und EAK nicht besitzt. Legt man aber zu Grunde, dass der Detektor selbst eine MCU gesteuerte Einheit ist, kann man bei weiterer Betrachtung wiederum von modularem Aufbau sprechen. Der Lokalbus (vgl. Abb.: 2.2) auf Seite 14, der mehrere Ein-/Ausgabe Konzentratoren über den Busmaster miteinander kommunizieren lässt, wurde hinsichtlich der TLS-Richtlinien als RS 485 implementiert.

2.4 Klassifizierung von Fahrzeugen

Grundlage der für den Verwendungszweck vorgenommenen Klassifizierung der Kfz bildet eine Einteilung der Fahrzeuge in Grundklassen. Bei dieser Grundklassifizierung werden alle Fahrzeuge auf eine in Tabelle 2.2 angeführten Grundklassen abgebildet.

Hier ist zu bemerken, dass die ursprüngliche Fahrzeugklassifizierung nur zwei Teilgruppen - Lkw ähnlich und Pkw ähnlich - beinhaltet hat. Durch die ständige Verbesserung der angewandten Technik liegt heutzutage auch eine 8+1 Klassifizierungsmöglichkeit vor. Diese wird auch vom verwendeten Detektor angewandt.

Grundklasse	Kurzbezeichnung	Erläuterung
Motorräder	Krad	Motorräder, auch mit Beiwagen, jedoch keine Fahrräder, keine Mofas
Pkw	Pkw	Pkw vom Kleinwagen bis zur Großraumlimousine einschließlich der Offroad-Fahrzeuge
Lieferwagen	Lfw	Lieferwagen $\leq$ 3,5t zul. GG
Pkw mit Anhänger	PkwA	Kfz bis 3,5t zul. GG mit Anhänger (auch Lieferwagen)
Lkw	Lkw	Lkw $>$ 3,5t
Lkw mit Anhänger	LkwA	Lkw $>$ 3,5t mit Anhänger
Sattelkraftfahrzeuge	Sattel-Kfz	alle Sattelkraftfahrzeuge
Bus	Bus	Fahrzeuge mit mehr als 9 Sitzplätzen zur Personenbeförderung; auch mit Anhänger
nicht klassifizierbare Fahrzeuge	nk Kfz	Alle Kfz, bei denen die Fahrzeugart nicht bestimmbar war oder die keiner der anderen Klassen angehören

Tabelle 2.2: Fahrzeugklassifizierung laut TLS

2.5 Datenübertragung

Für die Datenübertragung zwischen den Streckenstationen und der UZ steht in der Regel ein Adernpaar zur Verfügung, auf dem Daten in beiden Richtungen (Halbduplex-Betrieb) mit 1.200 Bit/s übertragen werden können (Basisband nach CCITT V23, ohne Hilfsband).

Falls im Einzelfall die Übertragungsqualität hierfür nicht ausreicht, sollten das Kabel bzw. die zugehörigen Einrichtungen entsprechend aufgearbeitet werden.

Die Datenübertragung innerhalb der Streckenstation erfolgt mit Hilfe einer Busverbindung, die als Zweidrahtleitung nach der Norm RS 485, dem Lokalbus, ausgeführt ist (vgl. Abb.: 2.2 auf Seite 14). Die Übertragungsrate wird hier mit 9600 baud (9600 bit/s) festgelegt.

2.6 Telegramme laut IEC 870

Es wird nur Formatklasse FT 1.2 gemäß IEC 60 870-5-1 (Übertragungs-Frame-Formate) verwendet mit den hauptsächlichen Eigenschaften: **Hammingdistanz 4**, **asynchron**, **byteorientiert**. Es wird nur die **unsymmetrische Übertragungsprozedur (unbalanced mode)** zugelassen [16]. Alle Daten sind binär kodiert. Besondere Kodiervorschriften sind nicht vorgesehen.

Telegramme mit Anwenderdaten bestehen aus einem ersten Startzeichen, 2 gleichen Zeichen, die die Anzahl L der Anwenderdatenbyte angeben, einem zweiten Startzeichen, den Anwenderdaten, einem Telegramm - Prüfsummenzeichen und einem Endezeichen. L ist ein Parameter in binärer Darstellung und liegt im Bereich zwischen 0 und 255. Anwenderdaten enthalten Datenfeld einschließlich Adress- und Steuerbyte.

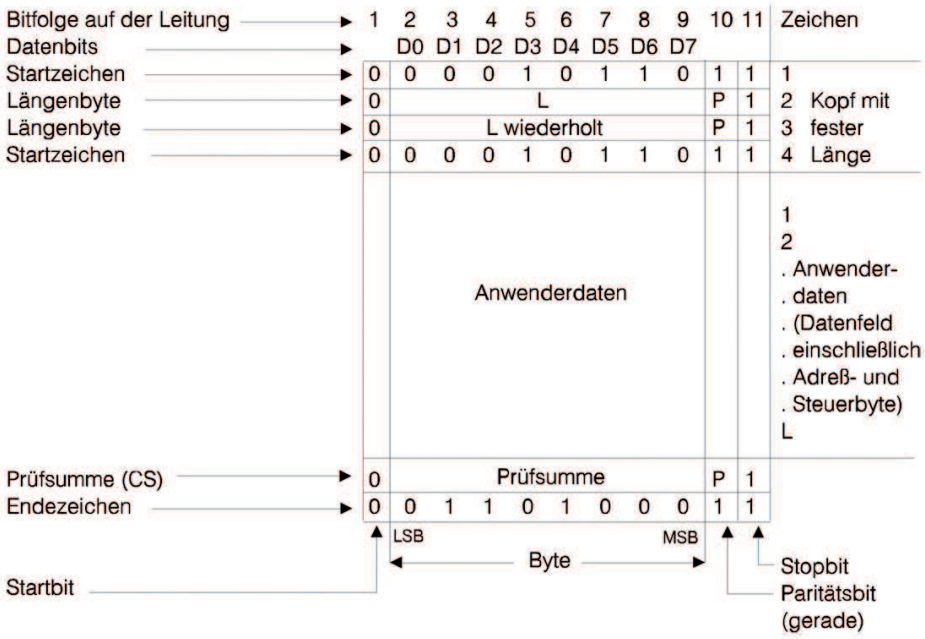


Abbildung 2.3: FT 1.2 Format nach IEC 870 für eine variable Länge gefordert von der TLS

In nachfolgender Tabelle (2.3) sind die vom Regelwerk der Technischen Lieferbedingungen für Streckenstationen geforderten Übertragungsregeln zusammengefasst.

R1	Ruhezustand auf der Leitung entspricht 1-Signal.
R2	Jedes Zeichen hat ein Startbit (0-Signal), 8 Informationsbit, ein gerades Paritätsbit und ein Stoppbit (1-Signal).
R3	Zwischen den Zeichen eines Telegramms sind keine Ruhezustände zugelassen.
R4	Die Reihenfolge der Anwenderdaten wird durch eine 8-Bit-Prüfsumme abgeschlossen, die die arithmetische Summe aller Anwenderdaten ohne Berücksichtigung der Überträge darstellt.
R5	Der Empfänger prüft: pro Zeichen: Startbit, Stopbit und gerades Paritätsbit pro Telegramm: 1. das festgelegte Startzeichen am Anfang und am Ende des Telegrammkopfteils 2. die Gleichheit der zwei Längenangaben L (length bytes) 3. die Telegramm-Prüfsumme (CS) 4. das Endezeichen Ergibt die Überprüfung ein negatives Ergebnis, so ist das komplette Telegramm zu verwerfen. Andernfalls ist es für den Anwender freizugeben.
R6	Nach dem Erhalt eines Telegramms, darf eine Anwort erst nach einem Ruhezustand von mindestens 33 bit (= 3.3 ms), aber muss spätestens nach 100 bit (=10 ms) gesendet werden.

Tabelle 2.3: Übertragungsregeln nach IEC 870

2.6.1 Telegrammaufbau

Für die Kommunikation zwischen Steuerung und Detektor sind insgesamt drei Telegrammtypen definiert (vgl. Abb.: 2.4).

1. **Langtelegramm:** Enthält ein Datenfeld mit variabler Länge.
2. **Kurztelegramm:** Enthält kein Datenfeld und hat eine feste Länge von 5 Byte.
3. **Quittungszeichen:** Dieser Telegrammtyp ist ein Einzelbyte mit dem Wert E5h.

2.6.2 Begrenzungsbyte der Telegramme

Das Lang- und Kurztelegramm wird von einem START- und einem END-Zeichen begrenzt. Im Langtelegramm wird das Startzeichen 68h verwendet. Dieses Zeichen (68h) wird als viertes Byte des Telegramms nochmals gesendet. Das Kurztelegramm hat das Startzeichen 10h. Beide Telegrammtypen verwenden als Endzeichen 16h.

2.6.3 Längenbyte

Das Längenbyte wird nur im Langtelegramm mit variabler Telegrammlänge verwendet. Die Längenangabe für das Datenfeld, einschließlich Steuer- und Adressbyte, steht im zweiten Byte

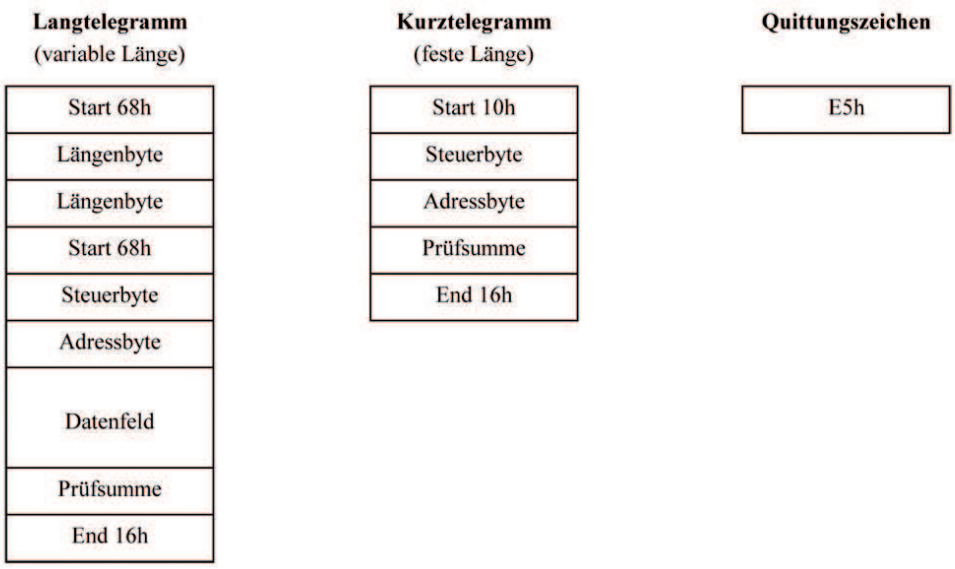


Abbildung 2.4: Die drei Telegrammtypen laut TLS (vgl. Abb.: 2.3 auf Seite 17)

des Langtelegramms. Dieser Wert wird im dritten Byte nochmals wiederholt.

2.6.4 Steuerbyte

Das Steuerbyte beinhaltet Funktionen, welche die Verbindung zwischen einer Primary und den Secondaries regeln.

2.6.4.1 Kommunikationsrichtung: Primary → Secondary

Die Kommunikationsrichtung Primary → Secondary bezeichnet die Datenflussrichtung von der Steuerungseinheit zum Detektor.

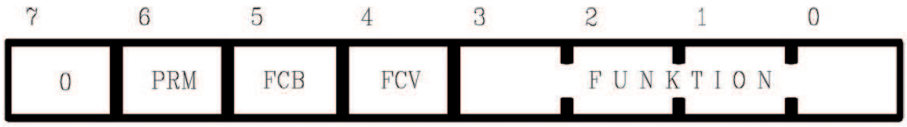


Abbildung 2.5: Steuerbyte bei Primary → Secondary

PRM - Primary Message: Bei einem Telegramm von der Primary ist das PRM-Bit immer log. "1".

FCB - Frame Count Bit: Das FCB-Bit ist ein alternierendes Bit für aufeinanderfolgende send/confirm- und request/respond Primitives je Station. Bei dem ersten Telegramm mit FCV-Bit log. "1", das eine Primary nach erfolgreicher Normierung mit RES0 an eine Secondary sendet, ist das FCB-Bit log. "1". Ein Wechsel des logischen Zustandes des FCB-Bits zeigt der Secondary an, dass ihr Telegramm (auch Quittungszeichen 0xE5) richtig empfangen

wurde. Kein Wechsel bedeutet, dass das Telegramm wiederholt werden muss. Das FCB-Bit wird für jede Secondary in der Primary geführt und verwaltet.

FCV - Frame Count Bit Valid: Dieses Bit gibt an, ob das FCB-Bit auszuwerten ist (log. 1), oder keine Gültigkeit hat und somit nicht auszuwerten ist (log. 0).

2.6.4.2 Kommunikationsrichtung: Secondary → Primary

Die Kommunikationsrichtung Secondary → Primary bezeichnet die Datenflussrichtung vom Detektor zur Steuerungseinheit.

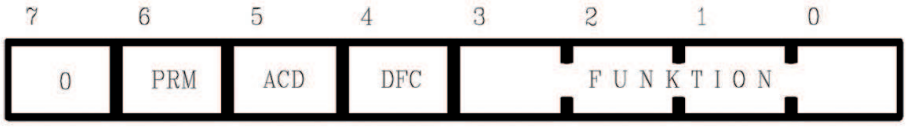


Abbildung 2.6: Steuerbyte bei Secondary → Primary

PRM - Primary Message: Bei einem Telegramm von der Secondary ist das PRM-Bit immer auf log. "0".

ACD - Access Demand: Das ACD-Bit wird bei Bedarf verwendet. Damit zeigen Sekundärstationen der Primärstation den Wunsch nach Übertragung von Daten der Klasse 1 an.

DFC - Data Flow Control: Dieses Bit dient zur Datenflusskontrolle. Es zeigt den Pufferstatus des Detektor an. Mit Receive Not Ready (log. 1) können weitere Daten zum Überlauf führen und mit Receive Ready (log. 0) können weitere Telegramme aufgenommen werden.

2.6.5 Adressbyte

Das Adressbyte beinhaltet, unabhängig von der Datenflussrichtung, immer die Adresse der Secondary.

Datenflussrichtung Primary → Secondary: Zieladresse
Datenflußrichtung Secondary → Primary: Quelladresse

Die All-Station Adresse (Telegramm an alle Secondaries) hat den Wert 0xFF. Die Adresse mit dem Wert 0x00 (No-Station Adresse) ist für Messzwecke reserviert. Die OSI-2-Adressen der Secondaries (E/A-Konzentratoren) am Lokalbus müssen frei einstellbar sein. Sie liegen im Bereich von 1..199.

2.6.6 Datenfeld

Das Datenfeld enthält die Daten, die für die höheren OSI-Schichten relevant sind. Das Datenfeld besteht aus einer variablen Anzahl von Datenbytes. Der variable Satzteil (Anwenderdatenbyte: Steuerbyte, Adressbyte und Datenfeld) darf 255 Byte nicht überschreiten.

2.6.7 Prüfsumme

Das Prüfsummenbyte dient der Feststellung, ob die Daten richtig übertragen wurden. Die für die Berechnung der Prüfsumme herangezogenen Bytes sind vom Telegrammtyp abhängig.

1. Kurztelegramm: Steuerbyte und Adressbyte
2. Langtelegramm: Steuerbyte, Adressbyte und Datenfeld
3. Quittungszeichen: hat keine Prüfsumme

Die Prüfsumme ist die arithmetische (dezimale) Summe dieser in obiger Aufzählung genannten Bytes modulo 256.

Zur Veranschaulichung wird mit nachfolgender Übertragungssequenz ein Berechnungsbeispiel für einen Longframe (vgl. Kapitel 6) durchgeführt. Für die Bedeutung der einzelnen Bytes siehe Abb.: 2.4 auf Seite 19. Die Berechnung kann natürlich auch direkt im Hexadezimalsystem durchgeführt werden. Da die Vorgabe der TLS jedoch modulo 256 lautet, soll hier jedoch über das Dezimalsystem gerechnet werden.

```
68 0E 0E 68   08 01 00 00   00 00 AB 4E   08 03 53 12    93 FE 03 16
```

Die arithmetische Summe der Bytes: **0x303**
(Steuerbyte=0x08, Adressbyte=0x01 und Datenbytes=0x00 bis 0xFE)

0x303 entspricht **771 dezimal**. 771 modulo 256 ergibt **3**, was wiederum **0x03** entspricht (vgl. obigen Frame: vorletztes Byte).

2.6.8 Realisierung der Telegrammprimitive (Primitives)

Der Datenverkehr wird mit sogenannten Primitives abgewickelt. Ein Primitive bildet eine untrennbare Kombination von Datensätzen zwischen Primary - und Secondarystation. Für die Datenübertragung werden 3 Primitives verwendet: send/no reply, send/confirm, request/respond.

Primitive: send/no reply	**Satzformate:** Long Frame Anwendungsbeispiel: Zentrale Versorgung der Zeitzähler mit der Uhrzeit
Primitive: send/confirm	**Satzformate:** Long Frame/Short Frame, Long Frame/Single Character, Short Frame/Short Frame, Short Frame/Single Character Anwendungsbeispiele: Parameterübergabe, Steuerprogrammübergabe, Anforderung zum Bereitstellen spezifischer Daten

Primitive: request/respond	**Satzformate:** Short Frame/Long Frame, Short Frame/Short Frame, Short Frame/Single Character Anwendungsbeispiel: Standard-Datenanforderung

3 Funktionsweise und Umsetzung

Das Detektionssystem zur Verkehrsanalyse, das dieser Arbeit zugrunde liegt, soll folgende Rahmenbedingungen bzw. technische Anforderungen erfüllen:

Autarke Stromversorgung: Die Energie für die Messeinheit soll ohne zu Hilfenahme des elektrischen Energienetzes bereitgestellt werden.

Datenaquisition: Die Messung soll folgende Parameter für jedes gemessene Kraftfahrzeug festhalten: Zeitstempel (Uhrzeit und Datum), Fahrzeugtyp (von Motorrad bis Omnibus inklusive nicht erkannt), Geschwindigkeit in km/h, Fahrzeuglänge in dm und Nettozeitlücke in Sekunden.

Datenübertragung: Die Datenübertragung (in beide Kommunikationsrichtungen) soll den technischen Standards genügen.

Datenspeicherung: Die Speicherung der Messdaten soll ausserhalb des Mikrocontrollers in einem (klarerweise) nichtflüchtigem Medium geschehen.

Visualisierung bzw. Auswertung: Ein Anwenderprogramm soll die gesammelten Daten in einer leicht verständlichen Darstellung zur Anzeige bringen.

3.1 Energieversorgung

Das System soll laut Anforderungsprofil ohne festverlegte Energieversorgung - also batteriebetrieben - funktionsfähig sein.

Als Verbraucher fungieren hier drei Subsysteme: Die Steuerungseinheit, die Speichereinheit und der Detektor. Durch eine Messung konnte die Stromaufnahme der Steuerungseinheit samt Speichereinheit (wird durch die Steuerungseinheit versorgt) mit maximal I_{max} = 204 mA beziffert werden, während der verwendete Detektor laut Datenblatt bei U = 12 VDC eine Stromaufnahme von I_{max} = 80 mA aufweist. Um einen sicheren Betrieb gewährleisten zu können, wird eine Energiemarge von 33% (284 mA * 1.33 = 377,7 mA; aufgerundet auf I_{ges} = 400 mA) hinzugerechnet.

Die gesamte Leistungsaufnahme (mit Energiemarge) beläuft sich damit bei I_{ges} = 400 mA und einem Versorgungsspannungsniveau von U = 12 VDC auf P_{ges} = 4,8 W.

Bei einer Gesamtleistungsaufnahme von 4,8W ergibt das für eine Messperiode von sieben Tagen eine Energieaufnahme von W = 806,4 Wh, was wiederum einer Ladung von Q = 67,2 Ah entspricht (Versorgungsspannung 12V). Es wird also ein Akkumulator mit einer Ladung

von in etwa Q = 70 Ah benötigt. Solche Akkumulatoren sind gängig und werden heutzutage in Mittelklassewägen (Starterbatterie) verbaut.

3.2 Ablaufkontrolle

Die Steuerung des Systems soll mittels Mikrocontroller, jedoch so einfach und kostengünstig wie möglich bewerkstelligt werden.
Durch die eher geringen Anforderungen an die MCU in punkto Geschwindigkeit fiel die Wahl auf einen μC, der sich in seiner Ursprungsform schon seit Jahrzehnten auf dem Markt befindet und zu einem vernünftigen Preis zu haben ist - der **89C51RDplus** (siehe Kapitel 4.1.1 Absatz 4.1.1).

Diese Mikrocontrollereinheit übernimmt die Ablaufkontrolle (Flowcontrol), die folgende Aufgaben beinhaltet (vgl. Abb.: 3.1 auf Seite 25)

1. Initialisierung von CPU, SPI und Drehschalter
2. Datenspeicherung: ermitteln der Größe und Schreibeposition der Speicherkarte
3. Tastenbetätigung: Zeitänderung
4. Datenempfang und -dekodierung
5. Fehlererkennung

Anmerkung: Um auch bei einem Energieversorgungsausfall die richtigen Zeitdaten (Uhrzeit und Datum) beizubehalten wird eine Echtzeituhr verwendet.

3.3 Datenaquisition

Hatte man früher nur die Möglichkeit bei der Detektion zwischen zwei Fahrzeugklassen unterscheiden zu können (PKW-ähnlich und LKW-ähnlich), so hat man heute die Möglichkeit bis zu neun unterschiedliche Typen zu klassifizieren (Detektoren des Typs 8+1: acht Fahrzeugklassen und eine Klasse für nicht zuordnungsbare Fahrzeuge).

Es befindet sich derzeit eine Vielzahl an Detektoren zur Verkehrsmessdatenerfassung am Markt. Folgende Detektoren wurden für diese Diplomarbeit evaluiert:

1. ADEC TDC3
2. Xtralis ASIM TT-298

Für diese Arbeit kamen nur Detektoren in Frage, die leicht - bzw. relativ einfach - zu montieren sind. Detektoren, wie zum Beispiel der VEK-S3 der Firma Feig, benötigen eine Induktionsschleife (Grabearbeiten) und wurden daher außer Acht gelassen.

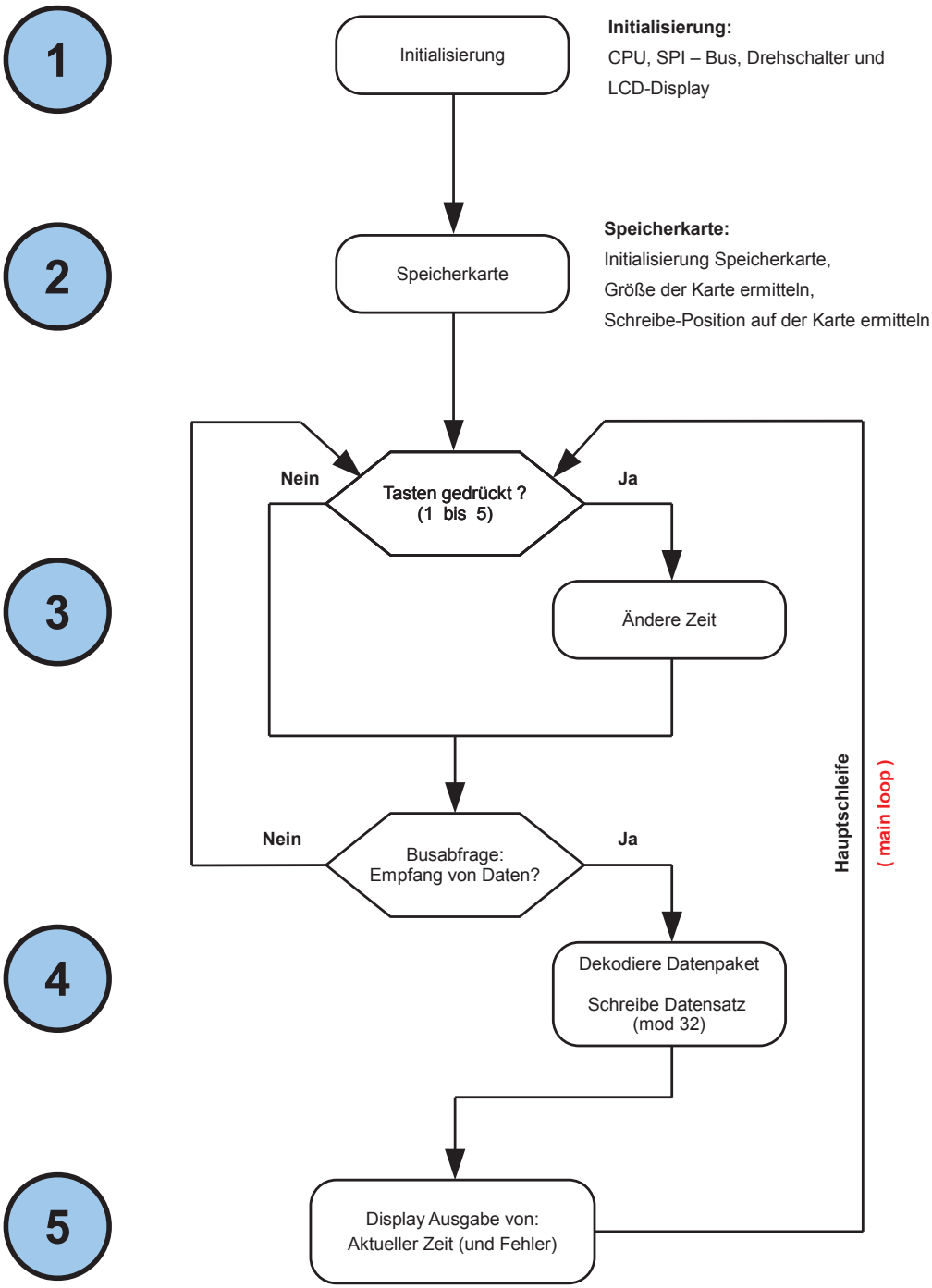


Abbildung 3.1: Ablaufsteuerung durch den 89C51RDplus

Die in Tabelle 3.1 enthaltenen Daten sind den Handbüchern des jeweiligen Herstellers entnommen. [20] [17]

	Xtralis TT 298	ADEC TDC3
Mechanisch		
B x H x T: Gehäusematerial: Gewicht:	240 x 120 x 212 [mm^3] Polykarbonat (hellgrau) ca. 1800 g	200 x 105 x 170 [mm^3] Polykarbonat (dunkelgrau) ca. 1700 g
Mikrowelle		
Doppler-Radar:	24,05 bis 24,25 GHz (K-Band)	24,05 bis 24,25 GHz (K-Band)
Ultraschall		
Frequenz: Impuls-Frequenz:	50 kHz 10 bis 30 Hz	40 kHz 10 bis 30 Hz
Infrarot		
Spektrale Empfindlichkeit:	8 bis 14 μm	6,5 bis 14 μm
Elektrisch		
Speisespannung: Stromaufnahme:	10,5 bis 30 VDC max. 100 mA @ 12 VDC typ. 25 mA im Standby	10,5 bis 30 VDC max. 110 mA typ. 80 mA @ 12 VDC
Datenaustausch		
Bus:	RS 485	RS 485
Hochlaufzeit		
	typisch 20s	typisch 20s
Genauigkeit		
Zählung: Geschwindigkeit:	typ. ±3% typ. ±3% (>100 km/h) typ. ±3 km/h (≤ 100 km/h)	typ. ±3% typ. ±3% (>100 km/h) typ. ±3 km/h (≤ 100 km/h)
Umgebungseinflüsse		
Betriebstemperatur: Feuchte: Dichtheit:	-40 °C bis +70 °C 95% RH max IP 64	-40 °C bis +70 °C 95% RH max IP 64

Tabelle 3.1: Gegenüberstellung der Detektoren

An diesem Punkt ist festzuhalten, dass alle beide in Betracht gezogenen Detektoren den Anforderungen genügen und die technischen Voraussetzung in Bezug auf standardisierte Fahr-

zeugklassifizierung erfüllen.

Da der Support (beide Firmen bekamen Anfrage E-Mails bzgl. Technischer Details, Blockschaltbilder des Aufbaues, etc...) von ADEC meines Erachtens besser war, ich bekam zumindest ein Anwortschreiben mit Absage, fiel die Wahl auf den ADEC TCD3. Denn aufgrund der sehr ähnlichen Produktparameter (vgl. Tab: 3.1 auf Seite 26) konnte keine Entscheidung getroffen werden.

Die Datenübertragung erfolgt durch ein von der Firma ADEC festgelegtes, an das Regelwerk der TLS (vgl. Kapitel 2) angelehntes und proprietäres Protokoll via RS 485 Bussystem auf dem der Detektor mittels vordefinierter Telegramme mit dem Mikrocontroller kommuniziert.

Für die Datenübertragung zwischen Master und Slave sind sechs Übertragungsregeln definiert (siehe Tabelle: 2.3 auf Seite 18).

3.4 Datenspeicherung

Da der interne Speicher des μCs zu klein für zusätzliche Datenspeicherung ist, müssen die Verkehrsdaten ausserhalb der MCU in einem nichtflüchtigen Speichermedium abgelegt werden können. Folgende Überlegungen bzw. Vorgaben waren bei der Auswahl des Mediums bezüglich Type und Größe relevant:

1. Minimalgröße nach TLS-Anforderungen (siehe unten)
2. Langfristige Produktverfügbarkeit
3. Lange Lebensdauer
4. Hohe Schreib- und Löschzyklen
5. Ohne großen Hardware- und Softwareaufwand implementierbar und
6. falls entfernbar, leicht und schnell zu ersetzen.

Bei Messquerschnitten, an denen Langzeitdaten erfasst werden und nicht online abgerufen werden können - so wie es hier der Fall ist -, sind geeignete Speicher vorzusehen. Diese müssen die Langzeitdaten mindestens eines Monats speichern können [15].

Daraus ergibt sich bei einem angenommenen Verkehrsaufkommen von maximal 300 Fahrzeugen[1] pro Stunde in eine Hauptverkehrsrichtung und einer Datenmenge von 16 Byte pro Messung ein Datenvolumen von ungefähr 3,5 MByte/Monat (1 kByte = 1024 Byte; 1 Monat = 31 Tage).

Das Regelwerk der TLS gibt jedoch vor, dass bei der Aufzeichnung von Einzelfahrzeugdaten mit einem maximalen Verkehrsaufkommen von 30.000 Fahrzeugen pro Tag (entspricht 1250 Fahrzeuge pro Stunde) gerechnet werden soll und sieht eine Mindestspeichergröße von 20 MByte für eine Woche vor.

Kombiniert man die Forderungen der TLS (für offline-Systeme) für Langzeitmessung und Aufzeichnung von Einzelfahrzeugdaten und zieht man in Betracht, dass die zu speichernde Datenmenge bei diesem vorliegenden System um einiges geringer ist als bei „norma-

[1] Wert aus einer innerstädtischen, stichprobenartigen Verkehrszählung zu Hauptverkehrszeiten

len“ Streckenstationen (hier: 16 Byte/Messung), so errechnet sich ein Wert von ungefähr 15 MByte/Monat.

Die Wahl fiel auf ein Flash basierendes Speichermedium - die MMC (MultiMedia Card) bzw. deren zur MMC abwärtskompatiblen Nachfolger - die SD-Karte (Secure Digital Memory Card) mit einer Größe von 16 MByte.

3.4.1 Allgemeines zu MMC bzw. SD

Der Standard des digitalen Speichermediums Multimedia Card (MMC) wurde im Jahr 1997 von der Siemenstochter Ingentix zusammen mit SanDisk entwickelt. Die MMC-Karte gewährleistet eine nichtflüchtige Speicherung bei niedrigem Energiekonsum (Flash-EEPROM Speicher). In ihren Abmessungen ist sie 24 mm × 32 mm × 1,4 mm groß. Ihre 7 Pins werden über einen internen Controllerchip angesteuert. Das Speichervolumen einer MMC reicht von 2 MB bis zu 8 GB und die Datenübertragungsrate liegt bei 2,5 MB pro Sekunde [8].

Einer eventuellen Verwendung der Secure Digital Memory Card (SD) steht hier aber nichts im Wege, da diese abwärtskompatibel zur MMC-Karte ist. Die zusätzlichen Hardwarefunktionen der SD Memory Card für das Digital Rights Management (DRM), die zum einen unter Verschluss liegen und zum anderen nur zahlenden Lizenznehmern zur Verfügung gestellt werden, werden für die Datenablage sowieso nicht benötigt [9].

3.4.2 Realisierung der Datenspeicherung auf MMC bzw. SD

Für die Datenspeicherung auf die Memory Card wird keine standardisierte Formatierung, wie zum Beispiel FAT (File Allocation Table), verwendet. Die Karte wird zur Vorbereitung/Formatierung der Datenspeicherung mittels Software zur Gänze mit Nullen beschrieben.

Auf die Karte kann nur blockweise zugegriffen werden. Der Mikrocontroller speichert also intern (RAM) die Messdaten bis die Datenmenge eines ganzen Blocks vorliegt (1 Block = 512 Byte = 32 Messungen) und schreibt diese dann auf die Speicherkarte.

Zunächst werden von der Karte die ersten 16 Byte im ersten Block (Block 0) auf deren Inhalt überprüft. Ist dieser gesamt 0, so ist die Karte komplett leer und es werden von hier ab die Daten gespeichert. Der Einsprungpunkt bei einer nicht leeren Karte, ab dem gespeichert werden kann und dabei keine Daten überschrieben werden, wird softwarebasierend mittels Binärbaum (endliche Intervallschachtelung) ermittelt.
Dabei wird zuerst die Kartengröße ermittelt und dann von der „Kartenmitte“ aus der erste frei zu beschreibende Sektor gesucht. Sind die ersten 16 Byte im Sektor respektive die Daten für eine Messung 0, dann wird in der unteren Hälfte der Karte weitergesucht. Sind diese Daten ungleich 0 in der oberen.

Die Suchbereiche werden dabei immer halbiert: Ist der Einsprungpunkt mit Sicherheit in der unteren Hälfte zu finden, so wird zuerst dieser Bereich halbiert und in diesem Mittelsektor die erste Messung auf deren Inhalt untersucht et vice versa...

Bei n Sektoren werden bei der binären Suche maximal $\log_2 n$ Schritte benötigt. Für eine Kartengröße von 16 MByte ergibt sich eine maximale Schrittanzahl von 15.

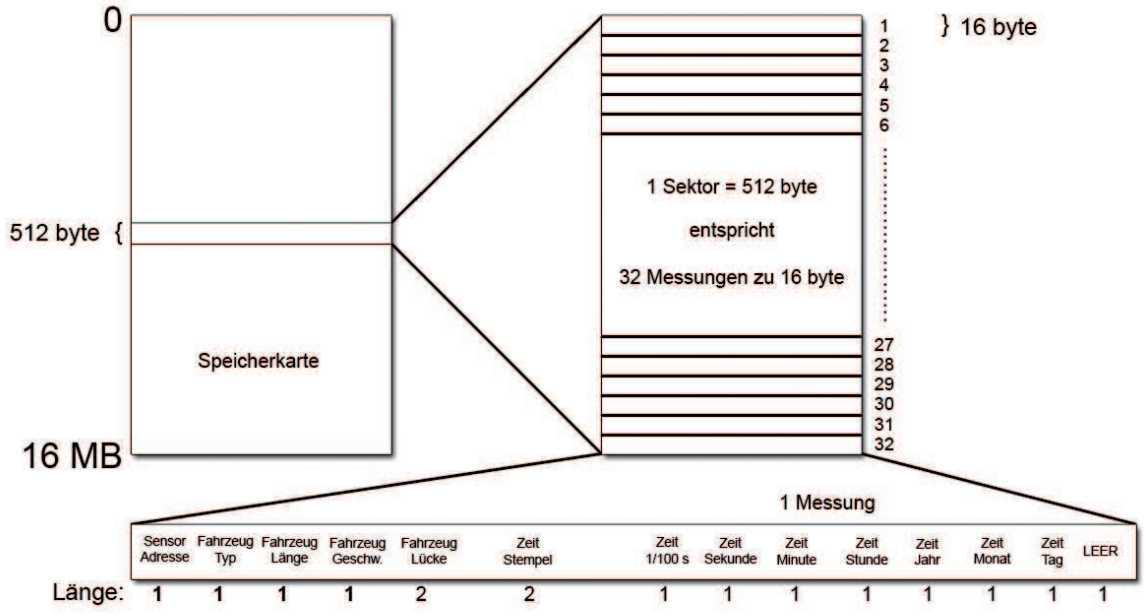


Abbildung 3.2: Angewandtes File-Format

Für die Implementierung (Quellcode) des Algorithmus siehe Prozedur **void GetCardSize()** in TrafficCounter.c, Anhang A ab Seite 77.

4 Hardware

Die gesamte Hardware des Messsystems besteht aus drei Teilen. Erstens die Steuerungsplatine, die für den Ablauf der Kommunikation verantwortlich ist. Zweitens die Speicherplatine, die sämtliche Hardware zur Speicherverwaltung und die Speicherkarte enthält und drittens der TDC3 Detektor der Firma ADEC.

4.1 Steuerungsplatine

Die Steuerungsplatine umfasst in ihrer Gesamtheit die in diesem Abschnitt dargestellten und beschriebenen Funktionsblöcke (vgl. Abb.: 4.2 auf Seite 31).

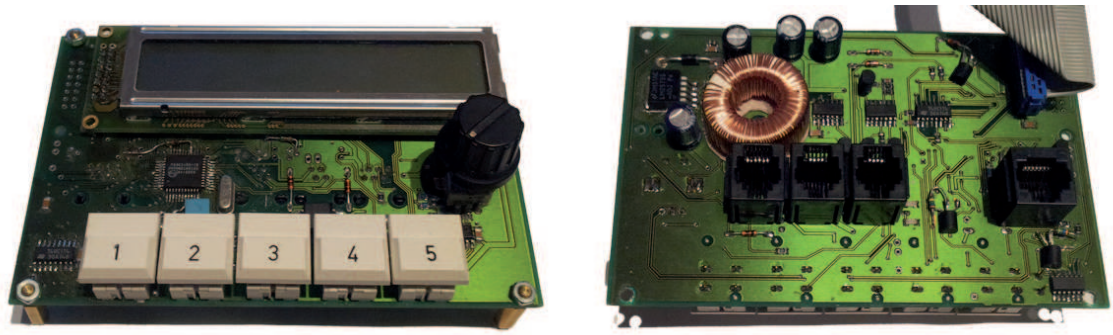


Abbildung 4.1: Steuerungsplatine

Die linke Abbildung in 4.1 zeigt die Steuerungsplatinenunterseite. Diese enthält unteranderem den SMD-Mikrocontroller. Die Abbildung rechts zeigt die Platinenoberseite mit den Anschlüssen für ISP, Relaisansteuerung und Spannungsversorgung/Detektorkommunikation.

4.1.1 Mikrocontroller

Einfache Funktionen können häufig auch mit relativ einfachen Schaltkreisen realisiert werden. Liegen dem System jedoch komplexere Aufgaben zugrunde, wird man beim Design schnell an Grenzen stoßen. Mit der Komplexität der zu erledigenden Aufgaben vergrößert sich im Normalfall auch die Anzahl der benötigten Bauelemente (BOM: Bill Of Material). Die Platinenabmaße und der Energiebedarf steigen. Den (State-Of-The-Art) Ausweg aus diesem Dilemma bietet die Verwendung einer MCU als zentrales Steuerelement [6].

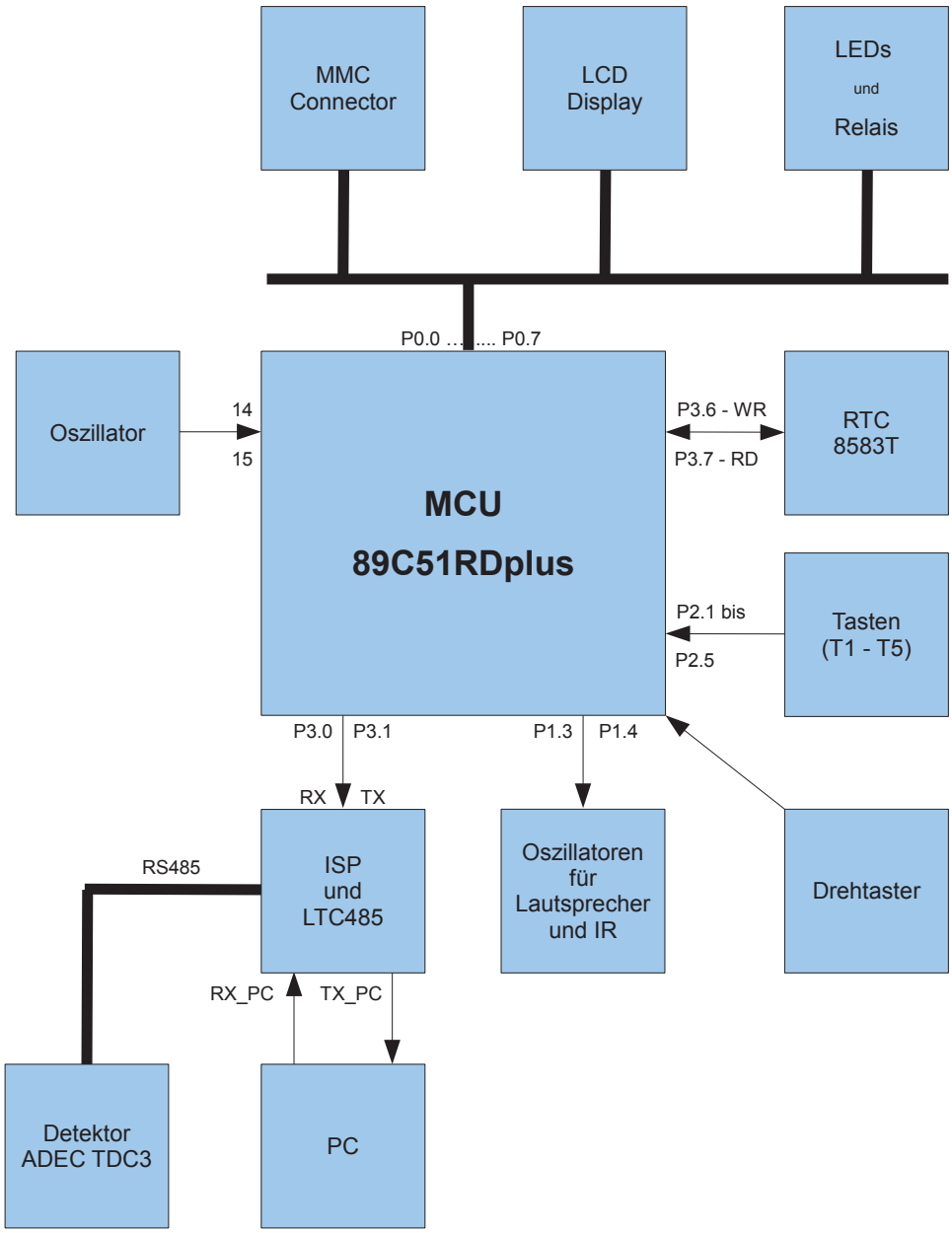


Abbildung 4.2: Blockschaltbild der Steuerungsplatine

Eine Mikrocontrollereinheit (MCU) ist ein intergrierter Baustein, der die Recheneinheit und bestimmte Peripherie auf einem Chip vereinigt. Er erlaubt somit den Aufbau nicht-trivialer Steuerungen mittels wenig zusätzlicher Hardware. Die eigentlichen Steueraufgaben erledigt die Mikrocontroller-Software.

Bei der Verwendung von Mikrocontrollern vereinfacht sich zum einen das Hardwaredesign erheblich, weil viel weniger diskrete Bauelememte benutzt werden müssen, zum anderen wird die Funktionalität zu einem Großteil in Software realisiert, weswegen Änderungen respektive Fehlerkorrekturen unproblematisch zu erledigen sind.

Da Mikrocontroller in großen Stückzahlen produziert werden sind diese meist auch kostengünstig im Erwerb. Es existiert eine Vielzahl von Typen und Peripherie auf dem Markt. Nun liegt es am Entwickler, den für seine Anwendung richtigen Chip zu wählen. Die Ressourcen am Chip sind effektiv zu nutzen um nicht auf ein „höheres“ Modell (Anschaffungskosten, Energienutzung, etc...) wechseln zu müssen.

Die am Die[1] integrierte Peripherie, lässt die MCU mit anderer Hardware kommunizieren respektive wird sie für die Programmausführung benötigt. Die gängigsten Peripherieelemente sind Speicher (Quellcode: ROM) und Daten: RAM), verschieden digitale und/oder analoge Ports, AD/DA-Wandler, Timerfunktionen (Ablaufsteuerung und Zeitmessung), Schnittstellen wie UART, I^2C, SPI, CAN und USB.

4.1.2 Auswahl des Mikrocontrollers

Primär ist zu diesem Thema zu sagen, dass - falls es bereits einen passenden ASIC (Application Specific Integrated Circuit) auf dem Markt gibt - dieser auch zu bevorzugen ist. FPGAs (Field Programmable Gate Array) und CPDLs (Complex Programmable Logic Device) sind rekonfigurierbare Bausteine, die vorallem dort eingesetzt werden, wo es um schnelle Signalverarbeitung und/oder flexible Änderungen der Verschaltung geht. Ein Vorteil dieser integrierten Schaltkreise ist die Möglichkeit der kostendeckenden Fertigung von komplexen SoC's (System on Chip) selbst bei geringer Stückzahl.

Bei der Entscheidungsfindung bei der Auswahl des Mikrocontrollerssollten eine Reihe von Faktoren gegeneinander abgewogen werden:

- Beschaffung, Kosten und zusätzliche Bauelemente
- Energieumsatz, Betriebsspannung
- On-Board-Peripherie
- Existieren Beschreibungsmaterial, Internetforen, etc ...
- Designtools für den Controller bzw. Toolchain
- Speicher, Instruktionen (Assembler oder Hochsprache?), Registersatz
- Ausführungsgeschwindigkeit
- In-System respektive In-Circuit Programming

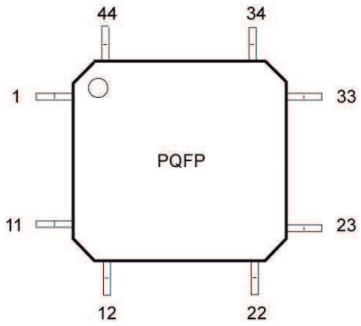
[1] Die (englisch): Bezeichnung für den ungehäusten Chip.

Legt man diese Kriterien zugrunde, eignet sich für den Einsatz eine Familie von Mikrocontrollern besonders gut: 89C51 von NXP. Sie wurde bereits in vielen Projekten eingesetzt, so dass eine breite Unterstützung in Form von Entwicklungswerkzeugen und Dokumentation (teilweise) kostenlos zu bekommen ist. Eine Abänderung des Programmcodes ist auch möglich, wenn der Controller bereits in einer Schaltung eingebaut ist.

Weiters ist zu beachten, dass modernere Lösungen meist als Ball-Grid-Array (Kugelgitteranordung) ausgeführt und so auch schwerer einzubauen sind. Bei solchen Mikrocontrollern ist es nicht selten der Fall, dass das Platinenlayout mehr als zwei Layer benötigt (Aufwand und Kosten). Zudem hat der 89C51RDplus ein Onboard-RAM. So entfällt die Handhabungsnotwenigkeit von hohen Frequenzen (RAM-Bus), was auch die EMV erleichtert.

4.1.3 Der 89C51RDplus von NXP

Der P89C51RDplus ist ein single-chip 8 bit Mikrocontroller in Advanced-CMOS Technik und besitzt einen 64 kbyte großen Programmspeicher (NV-FLASH EPROM), der sowohl parallel als auch seriell (In-System-Programmable) programmierbar ist. In System Programmierung (ISP) erlaubt es den Programmspeicher des Chips im fertigen Produkt mit Hilfe von Software abzuändern. Ein sich im ROM befindliches Boot-Loader-Programm, der Serial Loader, erlaubt es den Inhalt des FLASH-Speichers abzuändern. Es wird hierzu also kein separater Code im FLASH benötigt.



Pin	Function	Pin	Function	Pin	Function
1	P1.5/CEX2	16	V_{SS}	31	P0.6/AD6
2	P1.6/CEX3	17	NIC*	32	P0.5/AD5
3	P1.7/CEX4	18	P2.0/A8	33	P0.4/AD4
4	RST	19	P2.1/A9	34	P0.3/AD3
5	P3.0/RxD	20	P2.2/A10	35	P0.2/AD2
6	NIC*	21	P2.3/A11	36	P0.1/AD1
7	P3.1/TxD	22	P2.4/A12	37	P0.0/AD0
8	P3.2/$\overline{INT0}$	23	P2.5/A13	38	V_{CC}
9	P3.3/$\overline{INT1}$	24	P2.6/A14	39	NIC*
10	P3.4/T0	25	P2.7/A15	40	P1.0/T2
11	P3.5/T1	26	$\overline{PSEN}$	41	P1.1/T2EX
12	P3.6/$\overline{WR}$	27	ALE	42	P1.2/ECI
13	P3.7/$\overline{RD}$	28	NIC*	43	P1.3/CEX0
14	XTAL2	29	$\overline{EA}/V_{PP}$	44	P1.4/CEX1
15	XTAL1	30	P0.7/AD7		

Abbildung 4.3: Pinout des NXP P89C51RD plus

Diese MCU ist ein Derivat des 80C51 und besitzt daher auch denselben Befehlssatz (Instruction Set).

Weiters besitzt er vier 8 bit I/O Ports (Port 0 bis Port 3), drei 16 bit Timer/Event Counter, eine verschachtelte Interruptstruktur mit vier Prioritätsstufen, einen UART, einen Oszillator sowie Timing-Schaltkreise. Zusätzlich kann der Arbeitsspeicher auf maximal 64 kbyte extern durch Standard TTL-kompatible Bausteine erweitert werden[21].

Der FLASH-EPROM Speicher beinhaltet insgesamt 64 kbyte an Programmspeicher, der in fünf Blöcken organisiert ist. Die ersten beiden sind 8 kbyte (Adressraum: 0 bis 0x3FFF) und die restlichen drei 16 kbyte (Adressraum: 0x4000 bis 0xFFFF) groß.

Bei 89C51RX Plus MCUs ist es möglich den FLASH Speicher In-Circuit zu programmieren oder zu löschen. Das Löschen kann entweder für den ganzen Chip (Chip Erase Operation), oder blockweise (Block Erase Function) vollzogen werden. Selbst nach mehr als 1000 Lösch- und Programmierzyklen garantiert der Hersteller für eine zuverlässige Speicherung. Das wird durch spezielles Design der einzelnen Speicherzellen und einer Kombination aus Fertigungsmethoden (Advanced Tunnel Oxide Processing) und intern niedriger elektrischer Feldstärke erreicht.

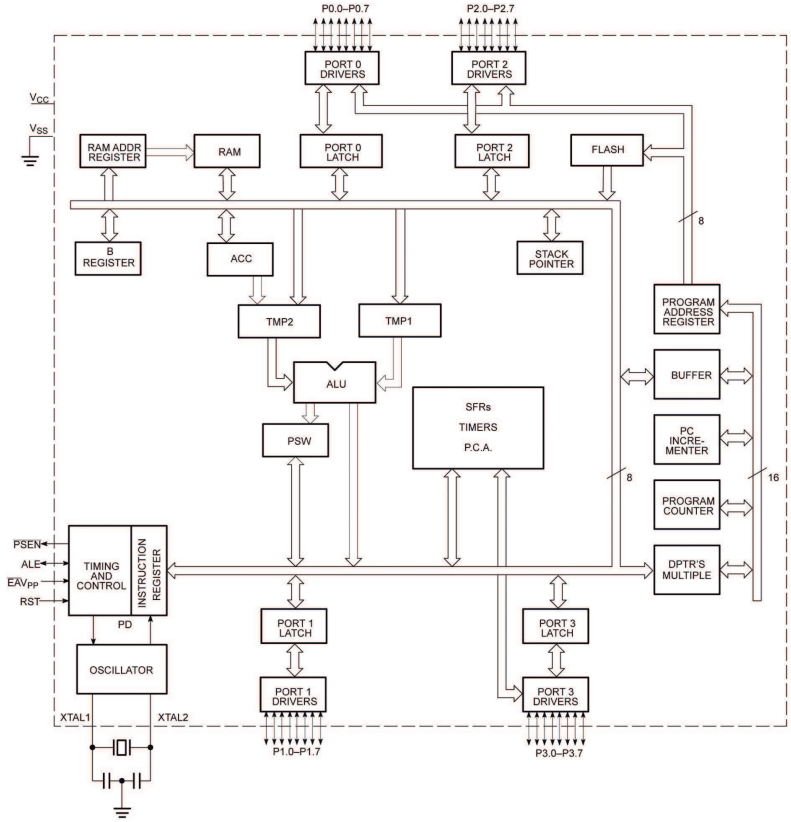


Abbildung 4.4: Blockdiagramm des NXP P89C51RD plus

Um den Speicher zu programmieren bzw. zu löschen wird eine Versorgungsspannung von $U_{pp} = 12 \pm 0{,}5$ V benötigt.

Mit Hilfe der In-System Programmierung ist es möglich den Speicher der MCU im Endprodukt, also auf der Platine verbaut, zu programmieren. Das heißt der Chip bleibt auf dem PCB (Printed Circuit Board, Platine, Leiterplatte).

4.1.4 Anschlussbelegung

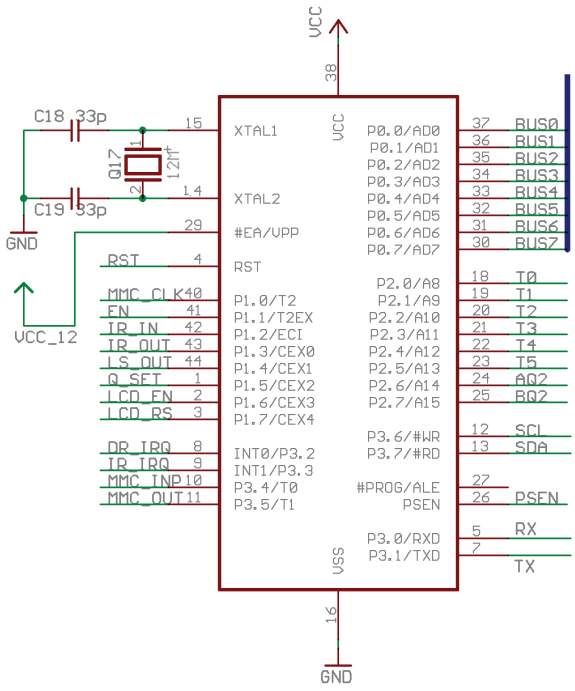


Abbildung 4.5: Anschlussbelegung des 89C51RDplus

Pin	Typ	Bezeichnung	Anschluss	Beschreibung
1	I/O	P1.5/CEX2	Q_SET	LED bzw. Relais setzen
2	I/O	P1.7/CEX4	LCD_EN	LCD Enable Signal
3	I/O	P1.6/CEX3	LCD_RS	LCD Reset Signal
4	I	RST	RST	Reset: Ein Highpegel für min. 2 Maschinenzyklen (Oszillator muss arbeiten) setzt die MCU zurück. Power-On-Reset durch externe Kapazität an V_{CC}-Pin (Siehe Schaltung 4.17 auf Seite 44)
5	I/O	P3.0/RxD	RX	Empfang: ISP oder RS485 (Siehe Schaltung 4.16 auf Seite 44).
6	—	NIC	—	No Internal Connection
7	I/O	P3.1/TxD	TX	Senden: ISP oder RS485 (Siehe Schaltung 4.16 auf Seite 44).
8	I/O	P3.2/$\overline{INT0}$	DR_IRQ	Drehtaster Interrupt
9	I/O	P3.3/$\overline{INT1}$	IR_IRQ	Infrarotempfänger Interrupt
10	I/O	P3.4/T0	MMC_INP	Input für Speicherkarte
11	I/O	P3.5/T1	MMC_OUT	Output von Speicherkarte
12	I/O	P3.6/$\overline{WR}$	SCL	Clockleitung I^2C für Echtzeituhr
13	I/O	P3.7/$\overline{RD}$	SDA	Datenleitung I^2C für Echtzeituhr

Pin	Typ	Bezeichnung	Anschluss	Beschreibung
14	O	XTAL2	Quarz	12M Quarz-XTAL
15	I	XTAL1		
16	I	VSS	GND	Masse: 0V Referenz
17	—	NIC	—	No Internal Connection
18	I/O	P2.0/A8	T0	Taster: Drehtaster
19	I/O	P2.1/A9	T1	Taster: Bedientaste 1
20	I/O	P2.2/A10	T2	Taster: Bedientaste 2
21	I/O	P2.3/A11	T3	Taster: Bedientaste 3
22	I/O	P2.4/A12	T4	Taster: Bedientaste 4
23	I/O	P2.5/A13	T5	Taster: Bedientaste 5
24	I/O	P2.6/A14	AQ2	Drehtasterauswertung vgl.: Abb.: 4.15
25	I/O	P2.7/A15	BQ2	Drehtasterauswertung vgl.: Abb.: 4.15
26	O	$\overline{PSEN}$	PSEN	Program Store Enable: Read-Strobe für externen Speicher. Wird für das ISP verwendet (Siehe Schaltung 4.17 auf Seite 44).
27	O	ALE	—	Address Latch Enable: Wird für externen Speicher oder Timing bzw. Clocking Aufgaben benutzt. ALE wird NICHT verwendet.
28	—	NIC	—	No Internal Connection
29	I	EA/VPP	VCC_12	External Access Enable / Programming Supply Voltage: 12V Versorgung für das FLASH Programmieren. Liest bei High nur vom internen 64k Programmspeicher (bei 89C51RD+)
30	I/O	P0.7/AD7	BUS7	Port0 (Pins 30 - 37):
31	I/O	P0.6/AD6	BUS6	Ein open-drain, bidirektionales
32	I/O	P0.5/AD5	BUS5	Ein-/Ausgabeport. Der Bus dient
33	I/O	P0.4/AD4	BUS4	zur Kommunikation mit der
34	I/O	P0.3/AD3	BUS3	Speicherplatine und dem LCD
35	I/O	P0.2/AD2	BUS2	Display.
36	I/O	P0.1/AD1	BUS1	
37	I/O	P0.0/AD0	BUS0	
38	I	VCC	VCC	Versorgungsspannung
39	—	NIC	—	No Internal Connection
40	I/O	P1.0/T2	MMC_CLK	Clock für die Speicherplatine
41	I	P1.1/T2EX	EN	Enable: RS485 Bus
42		P1.2/ECI	IR_IN	Request Infrarot-Empfang: Auswertung durch 2-Bit Schieberegister (Siehe Schaltung 4.15 auf Seite 43)
43	I/O	P1.3/CEX0	IR_OUT	Enable: Infrarot-Sender Oszillator

Pin	Typ	Bezeichnung	Anschluss	Beschreibung
44	I/O	P1.4/CEX1	LS_OUT	Enable: Lautsprecher Oszillator (Hardware wird NICHT verwendet)

Tabelle 4.1: Zuweisungen des Pinouts.

4.1.5 Versorgungsspannungaufbereitung mittels Tiefsetzsteller

Die Spannungsversorgung für die Elektronik wird mittels Tiefsetzsteller (Abwärtswandler) realisiert. Zur Anwendung kommt hier der LM2576S Baustein. Dieser Schaltregulator treibt bei einer Eingangsspannung von bis zu U_{IN}=45V einen Maximalstrom von 3A. Die Ausgangsspannung wird durch die beiden Widerstände R78 ($2,7k\Omega$) und R79 ($8,2k\Omega$) mittels folgender Beziehung

$$U_0 = 1.23(1 + \frac{R79}{R78}) \tag{4.1}$$

$$U_0 = 1.23(1 + \frac{8.2k\Omega}{2.7k\Omega}) = 4.97V \tag{4.2}$$

auf 5V eingestellt (siehe Datenblatt des LM2576S).

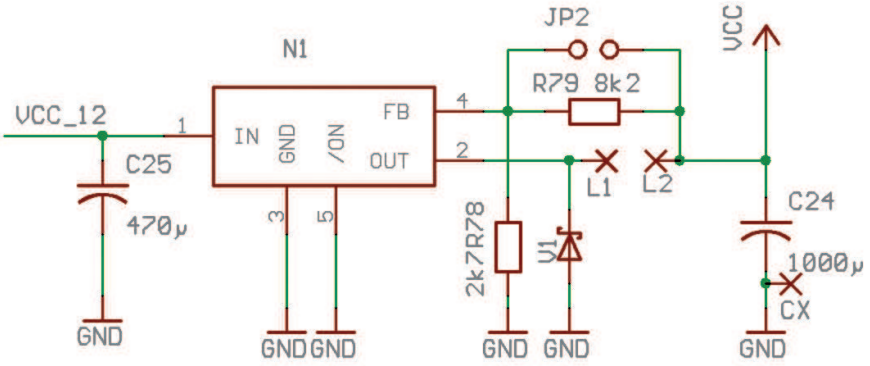


Abbildung 4.6: Schematisches Layout für den Tiefsetzsteller

Durch zusätzliche Beschaltung mit einer Spule (L1,L2[2]), einem Eingangs-(C25), einem Ausgangskondensator (C24) und einer Schottky-Diode (V1) wird der Abwärtswandler schlussendlich realisiert. Zu beachten ist, dass die Schottky-Diode für Dauerkurzschluss ausgelegt werden sollte (I_D=I_K=7A).

[2]L1 und L2 bezeichnen im Layout eigentlich die Bohrungen für die Induktivität.

4.1.6 Echtzeituhr

Um auch bei einem Absturz, einer Resetierung der MCU oder eines Zusammenbruchs der Versorgungsspannung des Systems eine richtige Zeitmessung (Zeitstempel) gewährleisten zu können wird ein zusätzlicher Baustein benötigt. Die Wahl fiel hier auf eine Echtzeituhr (und -kalender) von Philips - der PCF8583.

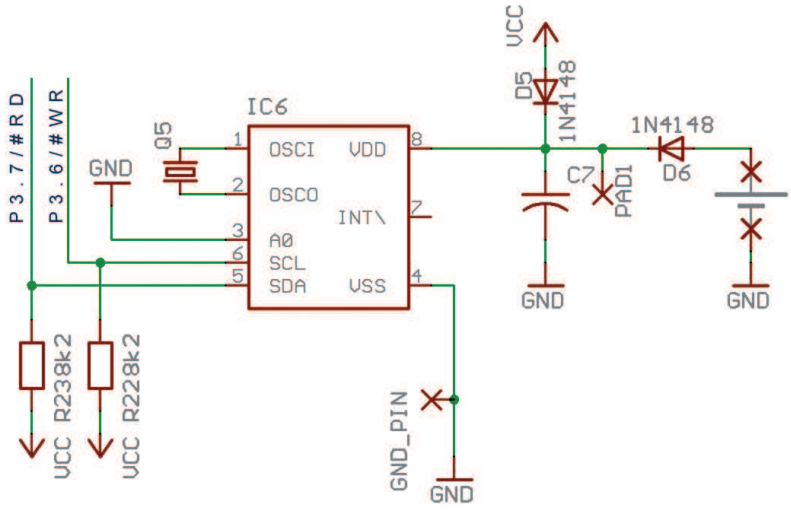


Abbildung 4.7: Ansteuerungshardware der PCF8583

Der PCF8583-Baustein repräsentiert eine Realtimeclock in CMOS-Technik. Das 2048 bit RAM ist in 256 Wörter zu je 8 bit gegliedert. Zur seriellen Datenübertragung und Adresszuweisung fungiert ein bidirektionaler I^2C-Bus. Das Built-In Word-Adress Register wird automatisch nach jedem Daten-Lese- bzw. Daten-Schreibvorgang inkrementiert. Für die einzelnen Funktionen der Clock und des Counters werden der Oszillator und die ersten 8 byte im RAM genutzt. Die darauf folgenden 8 byte können entweder als Alarmregister programmiert oder als Freier-RAM (zusätzlich zu den 240 byte) genutzt werden.

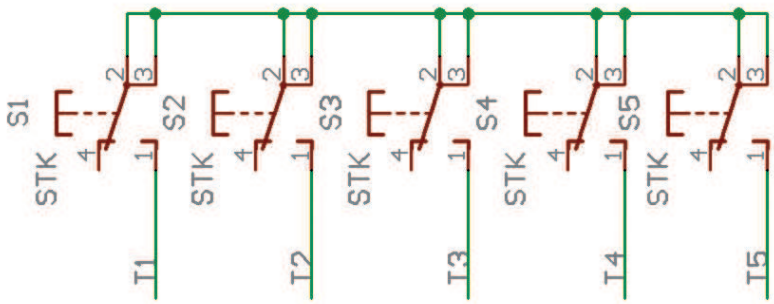


Abbildung 4.8: Zeiteinstellung mittels der 5 Taster (Jahr, Monat, Tag, Stunde, Minute)

Im Normalbetrieb, also beim Anliegen der Versorgungsspannung wird die Echtzeituhr über D5

mit Energie versorgt. Kommt es zu einem Versagen der Versorgungsspannungsbereitstellung wird der PCF8583 über D5 mittels Lithiumbatterie autark versorgt.

Die Bedientaster T1 bis T5 werden zur Zeiteinstellung verwendet. Die Information welcher Taster gedrückt wurde wird von der Micro-Controller-Unit mittels Port 2 eingelesen. Die eingestellte Zeit wird am LCD-Display angezeigt.

4.1.7 IR-Oszillator

Um die Möglichkeit zu schaffen Daten vom montierten und laufenden System erhalten zu können, wurde eine Infrarot-Empfangseinrichtung auf der Steuerungsplatine inkludiert. Die MCU schaltet den Osillator mittels IR_OUT Pin ein, worauf die Taktgenerierung durch den NAND-Schmitttrigger (IC1C) anläuft.

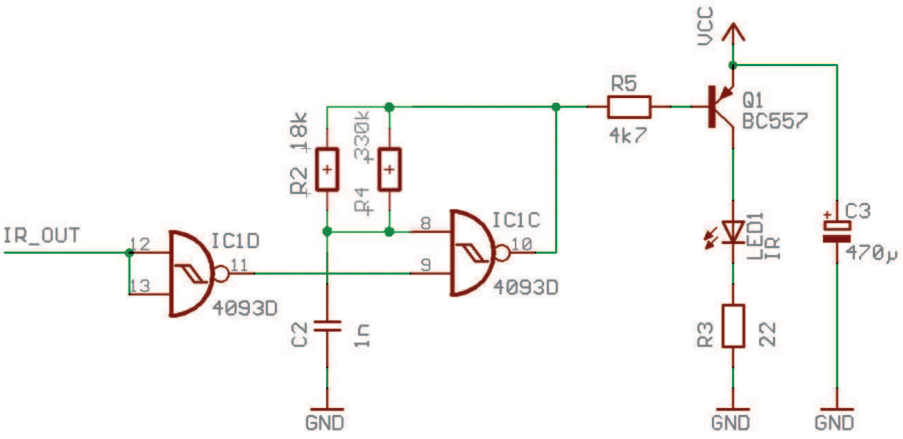


Abbildung 4.9: Aufbau des Oszillators für IR

Abbildung 4.11 zeigt die Spannungsverläufe des Infrarot-Oszillators vor und während der Aktivierung durch den Mikrocontroller. Ux bezeichnet dabei das Ausgangsspannungsverhalten des Schmitttriggers IC1C:

Die Taktfrequenz wird durch den Kondensator C2 (1nF) und dem Gesamtwiderstand aus R2 || R4 (R2 = 18kΩ; R4 = 330kΩ) nach folgendem Zusammenhang eingestellt:

$$\tau = R \times C = 18k\Omega||330k\Omega \times C = 17,07k\Omega \times 1nF = 17,07\mu s \tag{4.3}$$

$$f_{\text{Takt}} = \frac{1}{t_1 + t_2} \tag{4.4}$$

$$f_{\text{Takt}} = \frac{1}{RC \times ln(\frac{(V_{\text{T+}})(V_{\text{DD}} - V_{\text{T-}})}{(V_{\text{T-}})(V_{\text{DD}} - V_{\text{T+}})}} \tag{4.5}$$

Mittels folgender Tabelle aus dem Datenblatt des CD4093 Quad 2-Input NAND Schmitt-Trigger (Abb.: 4.10) kann die Taktfrequenz einfach zu

Symbol	Parameter	Conditions	−55°C		+25°C			+125°C		Units
			Min	Max	Min	Typ	Max	Min	Max	
V_{T-}	Negative-Going Threshold	$\|I_O\| < 1\ \mu A$								
	Voltage (Any Input)	$V_{DD} = 5V, V_O = 4.5V$	1.3	2.25	1.5	1.8	2.25	1.5	2.3	
		$V_{DD} = 10V, V_O = 9V$	2.85	4.5	3.0	4.1	4.5	3.0	4.65	V
		$V_{DD} = 15V, V_O = 13.5V$	4.35	6.75	4.5	6.3	6.75	4.5	6.9	
V_{T+}	Positive-Going Threshold	$\|I_O\| < 1\ \mu A$								
	Voltage (Any Input)	$V_{DD} = 5V, V_O = 0.5V$	2.75	3.6	2.75	3.3	3.5	2.65	3.5	
		$V_{DD} = 10V, V_O = 1V$	5.5	7.15	5.5	6.2	7.0	5.35	7.0	V
		$V_{DD} = 15V, V_O = 1.5V$	8.25	10.65	8.25	9.0	10.5	8.1	10.5	

Abbildung 4.10: Aus dem Datenblatt des CD4093 Quad 2-Input NAND Schmitt-Trigger: Thresholdspannungen V_{T+} und V_{T-}

$$f_{\text{Takt}} = 38774Hz \tag{4.6}$$

bei einer Umgebungstemperatur von T=25 °C und V_{DD}=5V ermittelt werden. Für die Thresholdspannungen wurden die typischen Werte herangzogen.

Die sich durch die Temperaturabhängikeit der Thresholdspannungen V_{T+} und V_{T-} ändernden Taktfrequenzen lauten wie folgt:

$$f_{\text{Takt -55}} = 38524Hz \tag{4.7}$$

für T=−55 °C und V_{DD}=5V und die Minimalwerte für V_{T+} und V_{T-}, sowie

$$f_{\text{Takt +125}} = 47664Hz \tag{4.8}$$

für T=+125 °C und V_{DD}=5V und die Maximalwerte für V_{T+} und V_{T-}. Für letzteres Szenario ergibt sich somit einer Taktfrequenzerhöhung von 23% während sich die Änderung bei einer Umgebungstemperatur von T=−55 °C innerhalb der erlaubten Grenzen von ±5% hält.

Da es sich bei diesem Wert (Siehe Ergebnis in: 4.8) um einen absoluten Grenzwert des Bausteins handelt und bei dieser Temperatur wahrscheinlich das gesamte System bereits einem irreversiblen Sekundärprozess zum Opfer gefallen wäre, wird dieser Wert hier ausser Acht gelassen. Ausserdem konnten während des Testbetriebs keine Anzeichen bezüglich eines dadurch resultierenden Fehlverhaltens gefunden werden.

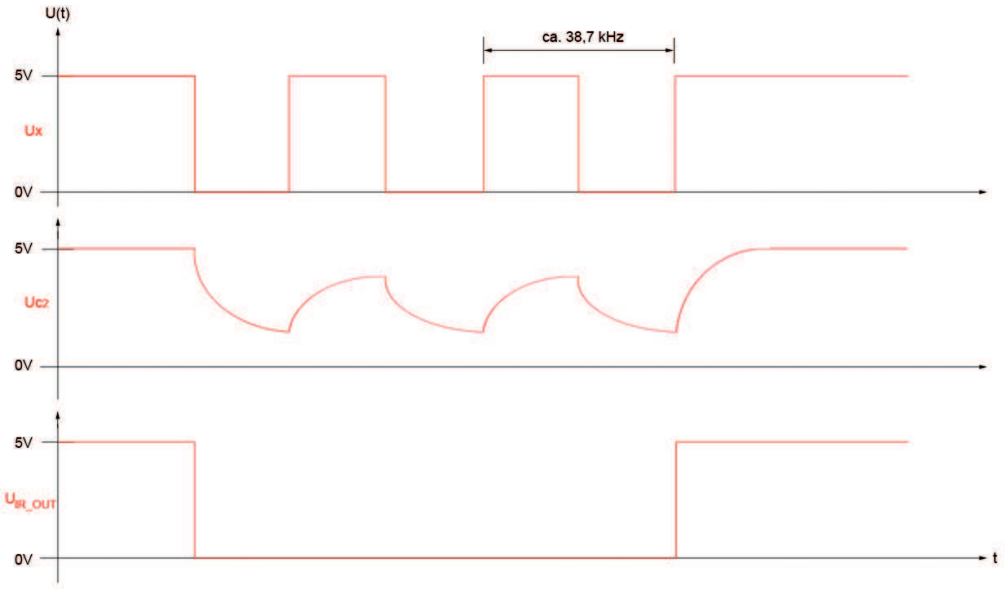


Abbildung 4.11: Spannungen als Funktion der Zeit bei Ansteuerung des Oszillators durch den Controller. Ux zeigt den Ausgangsspannungsverlauf von IC1C (Schmitttrigger)

4.1.8 Oszillator für die Lautsprechereinheit

Im Prinzip ist der Oszillator (Siehe Schaltung 4.12) funktionsmäßig dem IR-Oszillator (Siehe Schaltung 4.9) gleich.

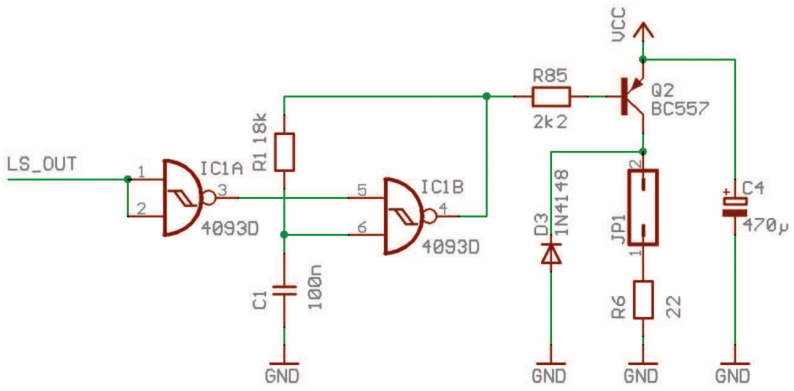


Abbildung 4.12: Aufbau des Oszillators für die Lautsprechereinheit

Ursprünglich war dem System eine Einrichtung angedacht, die die Möglichkeit einer höhrbaren Ein- bzw. Ausschaltbestätigung und einer akustischen Fehlermeldung bietet. Aufgrund von weiteren Überlegungen (Semantik) und der im Feldversuch gezeigten Probleme (Lärmbelästigung, Maskierung durch Verkehrslärm, etc...), wurde auf diese Einheit verzichtet.

Da die Hardware jedoch implementiert ist, wird sie der Vollständigkeit halber aber erwähnt.

4.1.9 LED Anzeige und Relaisansteuerung

Drei Leuchtdioden werden auf der Steuerungsplatine zur Statusanzeige (MMC-Error, MMC-Write und System-Initialisierung) verwendet. Mittels Bus wird die Information zur Anzeige der/des jeweiligen LED(s) von der MCU in einen 6-fach D-Flip-Flop Baustein (IC4) geschrieben (siehe Abb.: 4.13). Ausgangsseitig stellt dann das jeweilige in den einzelnen Flip-Flops des Bausteins gespeicherte Bit den Ausgangsspannungspegel her, der den Stromfluss über die LED ermöglicht (logisch 1: entspricht 5V) oder die Leuchtdiode abschaltet (logisch 0: entspricht 0V).

Ein zusätzlicher 1-bit-Speicherbaustein - hier das 6 fach D-Flip-Flop - muss verwendet werden, da der Mikrocontroller die Ansteuerungsinformation über den Bus nur einmal mit Q_SET hinausschreibt und so die angesteuerten LEDs nur kurz - und dadurch nicht merkbar - aufleuchten würden.

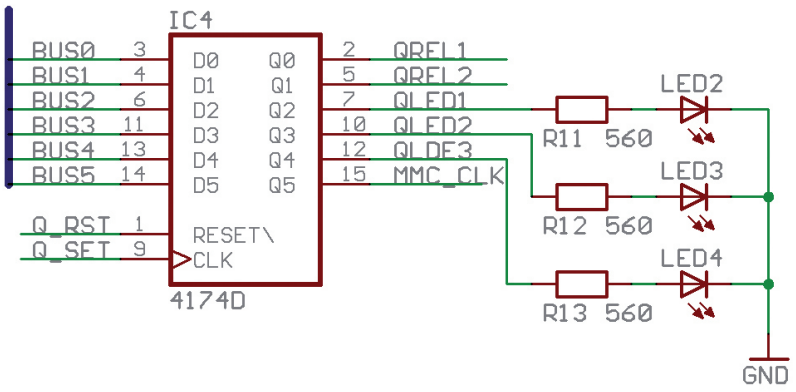


Abbildung 4.13: Status-LED Anzeige

Q_REL1 und Q_REL2 steuern Relais an, mit deren Hilfe man Warneinrichtungen (z.B.: Geschwindigkeit zu hoch) ansteuern kann. Über die Vorwiderstände (R11, R12 und R13) wird lediglich eine Stromstärke von zirka 9 mA eingestellt.

4.1.10 Dreh- und Bedientaster

Durch das Drücken des Drehtasters gelangt man in den Zeiteinstellmodus. Welchen Zeitparameter (Jahr, Monat, Tag, Stunde oder Minute) man durch Drehung (links: dekrementieren, rechts: inkrementieren) abändern möchte, wählt man mit dem dementsprechenden Bedientaster. Durch nochmaliges Drücken des Drehtasters wird die eingestellte Zeit mit zusätzlich 0 Sekunden übernommen.

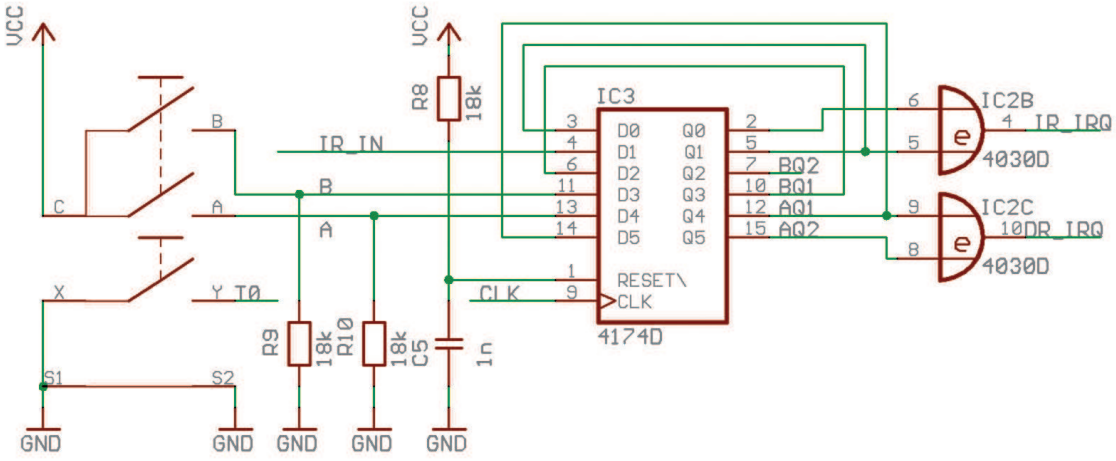


Abbildung 4.14: Drehtaster und 2-bit Schieberegister

Die Auswertung der Drehtasterbetätigung auf Links- bzw. Rechtsdrehung erfolgt mit Hilfe des 6 fach FlipFlop 4174B (IC3), das die Zustände der Taster A und B, die der Drehtasterfunktionalität zu Grunde liegen, speichert.

DR_IRQ gibt einen Impuls aus, wenn der Drehtaster einen Schritt weitergedreht wurde. Mit der fallenden Flanke von DR_IRQ wird der INT0 am Mikrocontroller ausgelöst. Der DR_DIR Ausgang gibt bei einem DR_IRQ-Impuls die Drehrichtung des Tasters an. Der Richtungsausgang muss innerhalb von 250 μs (Taktfrequenz: 4 kHz) nach dem Interrupt eingelesen werden (Siehe Abb.: 4.15).

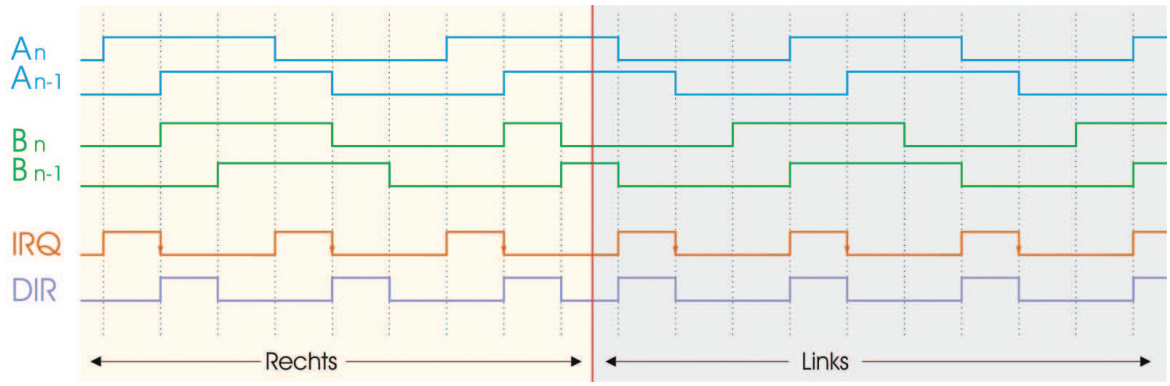


Abbildung 4.15: Signalverlauf zur Drehtasterauswertung

Bei der Schaltung werden A und B Signal in das 2-bit tiefe Schieberegister geladen. Ein Interrupt Impuls wird generiert, wenn A_n ungleich A_{n-1} ist. Das Richtungssignal erhält man durch eine XOR-Verknüpfung von A_{n-1} mit B_{n-1} (Siehe Abb.: 4.14).

4.1.11 In Service Programming und RS 485 Buskommunikation

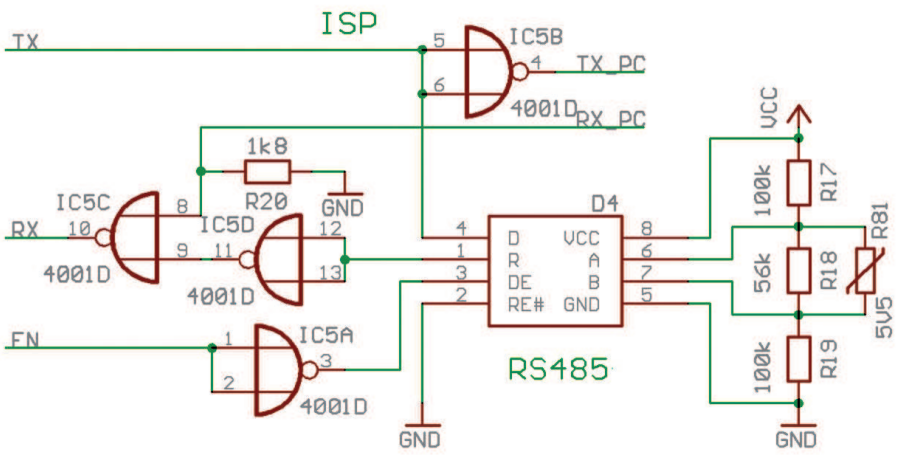


Abbildung 4.16: In Service Programming und RS485 Hardware

Realisiert wird die RS485 Buskommunikation mittels LTC485 Bustreiberbaustein (D4), der die Datenleitungen A und B mittels Spannungsteiler auf einem definierten Pegel vorgespannt hat (2:1:2). Als Schutzmaßnahme gegen Überspannung werden beide Leitungen mit einem Varistor verbunden.

Dadurch, dass sowohl die Pins für ISP, als auch Pins für die Buskommunikation durch die "Doppelbelegung"von Pin 3.1 am μC (TXD) auf einem gemeinsamen NOR-Gatter liegen (IC5B), muss gewährleistet sein, dass nicht beide Funktionalitäten gleichzeitig - was im Normalfall gegeben sein sollte - genutzt werden. Schließlich werden Änderungen am MCU Code nur vorgenommen, wenn das System im Labor programmiert wird und somit keine Datenübertragung mittels RS 485 gefordert wird.

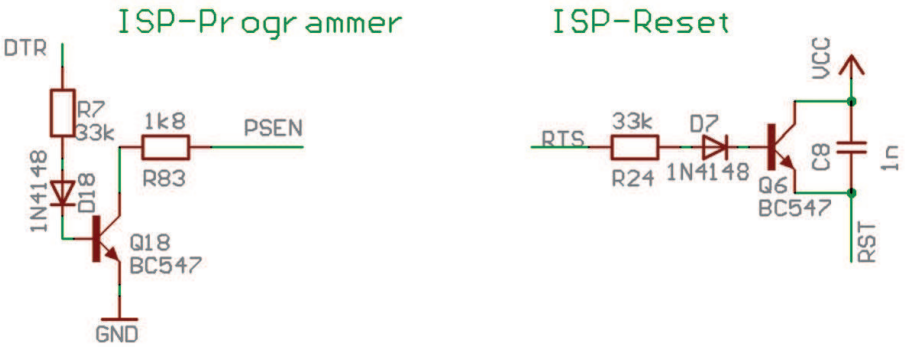


Abbildung 4.17: In Service Programmer und ISP Reset

Die ISP-Reset Ansteuerung (vgl. Abb. 4.17) hat zur Augabe, einerseites den Resetimpuls zu generieren und andererseits auch eine Einschaltverzögerung (Kondensator C8) zu erreichen. Die Einschaltverzögerung wird benötigt, da die Versorgungsspannungen erst bei einer Zeit $t > 0$ auf deren geforderten Normalpegel liegen. Ein nicht definierter Zustand in einem oder mehreren Registern wäre die Folge.

4.1.12 Kommunikation mit LCD und Speicherplatine

MMC-Connector: Mit Hilfe eines Kabels wird hier die Verbindung zur Speicherplatine hergestellt. Für die Kommunikation zwischen Steuer- und Speichereinheit genügen hier drei Kabeladern (MMC_INP, MMC_CLK und MMC_OUT)

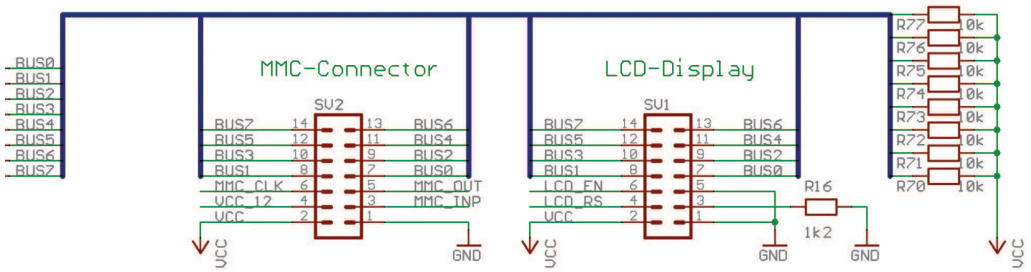


Abbildung 4.18: Zugriff auf das LCD-Display und die Speicherplatine

LCD-Display: Die LCD Einheit ist mit der Steuerungplatine fest verlötet. Die Displayplatine von Seiko Instruments ist eine 20 Character $\times$ 2 Lines Einheit mit eine 5×7 Punktmatrix und den Abmaßen $116 \times 37 \times 11,3$ [mm]. Sie benötigt +5V als Versorgungsspannung [14].

4.2 Speicherplatine

Im folgendem Abschnitt werden die Schaltungsteile respektive Funktionsblöcke der Speicherplatine beschrieben.

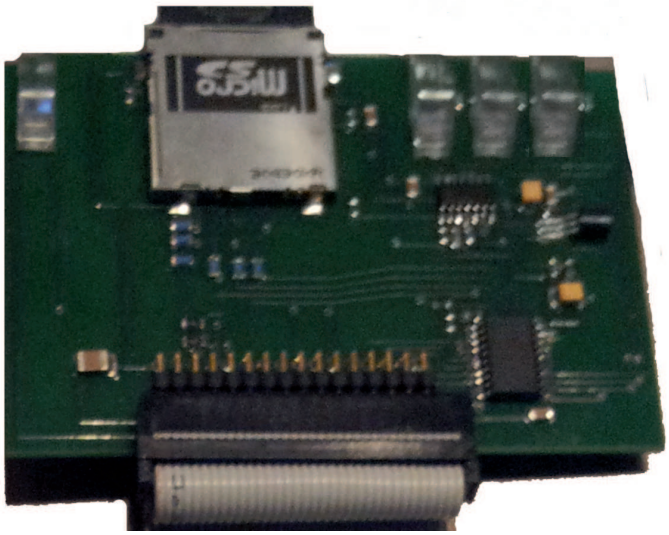


Abbildung 4.19: Speicherplatine

Weil die SD-Karte mit 3.3V betrieben wird und die Steuerungs- die Speicherplatine mit 5V versorgt ist eine Pegelanpassung der Signale notwendig. Für die Signale zur SD-Karte wird

das mit einem 8-Bit-Latch mit Open-Collector-Ausgängen realisiert.

Für die Umsetzung der Versorgungsspannung ist ein Low-Drop 3.3 V Spannungswandler vorhanden.

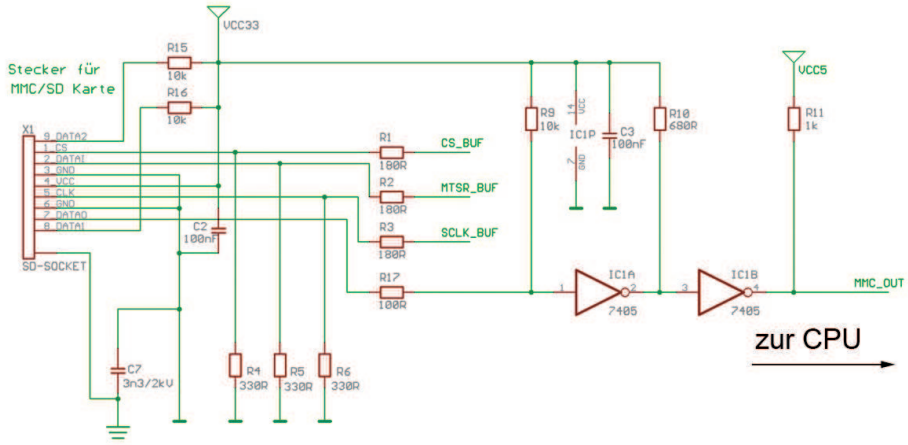


Abbildung 4.20: Hardware für Pegelwandlung mit Speicherkartensteckvorrichtung

Die Schaltungen in Abb. 4.20 und 4.21 sind durch die Anschlüsse CS_BUF, MTSR_BUF und SCLK_BUF miteinander verbunden. Die Pegelwandlung von 5V auf die von der Speicherkarte benötigten 3.3V wird durch die Spannungsteiler (vgl. Abb. 4.20: 180Ω und 330Ω) realisiert.

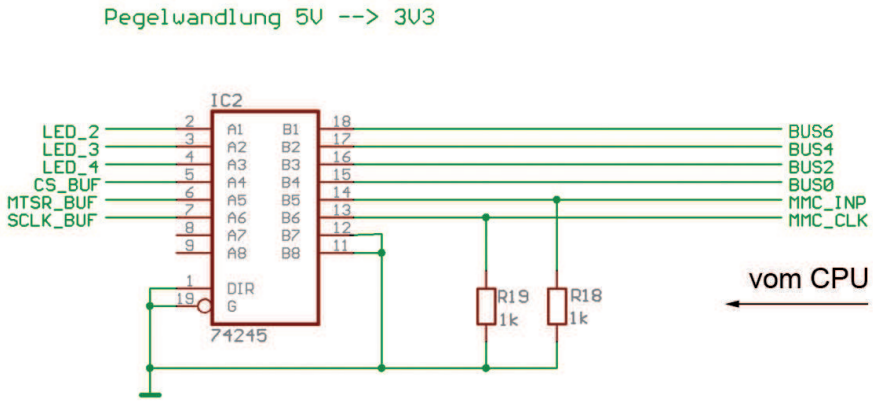


Abbildung 4.21: Pegelwandlung für die Betriebsspannung der Speicherkarte

Der IC2 (TriState Bus Transceiver) erlaubt die asynchrone, bidirektionale Kommunikation zwischen zwei Bussen. Die Richtung wird mittels DIR (Direction Control) vorgegeben. Hier liegt das DIR-Pin auf Masse, was die Kommunikation in nur eine Richtung vorgibt (es wird auf die Speicherkarte nur schreibend zugegriffen).

Es wurde ebenfalls die Möglichkeit der Anzeige des Speicherstatus der Karte (leer, halbvoll und voll) durch LED-Bauelemente (LED_2 - LED_4) angedacht. Dies ist jedoch zum jetzigen Zeitpunkt nur hardwareseitig implementiert.

4.3 Verkehrsdetektor ADEC TDC3

In den Detektoren der Serie TDC3 sind drei Technologien (Doppler Radar, Ultraschall und Passiv Infrarot) in einem gemeinsamen, wetterfesten Gehäuse zusammengefasst. Die von den Fahrzeugen in jedem Teilsystem generierten Signale werden getrennt verstärkt und in einem Mikrocontroller verarbeitet.

Abbildung 4.22: ADEC TDC3 Überkopfdetektor [19]

Der Radarteil misst die Geschwindigkeiten [7]. Der Ultraschall tastet die Fahrzeugprofile zur Bestimmung der Fahrzeug-Klassen ab und macht die Trennung der Fahrzeuge für die genaue Zählung. Der Multikanal PIR Detektionsvorhang wird für die Fahrspurselektion und zur Aktivierung der Ultraschall-Messungen benötigt.

Die Klassifizierung wird anhand von Fahrzeuglänge und dem Höhenprofil vorgenommen. Ein Fahrzeug, das nicht einer der definierten Klassen zugeordnet werden kann, wird einer separaten Klasse zugeordnet („nicht klassifizierbar“: vgl. Tabelle 2.2 auf Seite 16). Typischerweise sind dies Fahrzeuge, die nicht in der Mitte ihrer Spur fahren oder Spurwechsler [17]. Dieser Detektor bietet auch die Möglichkeit Fahrzeuge, die in falscher Richtung unterwegs sind, zu detektieren. Diese Option wurde in dieser Arbeit jedoch nicht implementiert. Die technischen Daten des Detektors entnimmt man Tabelle 3.1 auf Seite 26.

5 Software

Die Software für das System besteht aus drei Teilen. Erstens die Routinen für die MCU, zweitens die Software zur graphischen Darstellung der gespeicherten Daten und drittens die Software zur Formatierung und zum Auslesen der Speicherkarte.

5.1 Mikrocontrollersoftware

Es existieren eine Vielzahl von Developmentumgebungen bzw. Tools mit denen Steuersoftware für einen Mikrocontroller geschrieben werden kann. In Assemblercode werden die Befehle der jeweiligen CPU durch Mnemonics versehen und bilden damit direkt den CPU eigenen Befehlssatz nach.

Man kann sich unschwer vorstellen, dass ein komplexes Assemblerprogramm mit einer Vielzahl von Aufgaben mehrere tausend Zeilen an Quellcode enthalten kann und schnell unübersichtlich wird (vgl. Abb.: 5.1).

```
push ACC                ; Register retten
mov A,P2                ; P2.6 und P2.7 sind die Drehbits (A,B)
jb ACC.7,QB_High        ; Ist B auf 1 ?
jb ACC.6,NEqual         ; Ist A gleich B

Equal:
dec TurnSwitchState     ; Drehzustand: dekrementieren
pop ACC                 ; Register wieder herstellen
reti                    ; Zurück

QB_High:
jb ACC.6,Equal          ; Ist A gleich B

NEqual:
inc TurnSwitchState     ; Drehzustand: inkrementieren
pop ACC                 ; Register wieder herstellen
reti                    ; Zurück
```

Abbildung 5.1: Assemblerbeispiel: Drehtaster In- und Dekrement

Neben dem maschinennahen Assembler gibt es noch einige wenige Hochsprachencompiler. Beim Einsatz von Hochsprachen hat man zwar den Nachteil, dass der erzeugte Quellcode meist größer (und langsamer) ist als ein mittels Assembler generierter Code. Er ist aber deutlich lesbarer und damit auch fehlerunafälliger (vgl. Abb.: 5.2).

Neben verschiedenen Freeware- und Shareware Entwicklertools für den 80C51-Kern gibt es eine C-Entwicklungsumgebung, die ganz speziell auf die Bedürfnisse von Embedded Software Entwicklung ausgerichtet ist: μVision von Keil [10].

Keil μVision übernimmt bei Projektstart, nachdem man den gewünschten Mikrocontroller gewählt hat, sämtliche Einstellungen wie Compiler- Assembler, Linker und Speicheroptionen für diesen Chip. Der enthaltene Debugger simuliert die am Chip befindliche Peripherie wie Busse, Interrupts und I/O-Ports der 80C51.

Das Mikrocontrollerprogramm für den 89C51RDplus wurde mit Hilfe des Keil Compilers μVision in der Hochsprache C verfasst (Ausnahme: Drehtaster). Dieser umfasst die Ansteuerung des Displays, die Buskommunikation, die Datenspeicherung, die Drehtasterfunktionalität und die Fehlerüberprüfung.

```
for(;ucCount>0;) // Daten auslesen
{

    for(ucBit=8;ucBit>0;ucBit--)
           {
            ucAddr<<=1;
            CLK_H();
            if(DAT_R())ucAddr|=1;
            CLK_L();
           }

ucCount--;
if(ucCount)DAT_L(); // Acknowledge Bit ausgeben

CLK_H();
CLK_L();
DAT_H();

pData[0]=ucAddr;
pData++;

}
```

Abbildung 5.2: Hochsprachenbeispiel Programmiersprache C: Auslesen der Echtzeituhr

Eine Darstellung des Quelltextes der Hauptroutinen mit Namen TrafficCounter.c findet man im Anhang A ab Seite 77. Zur Verwendung kam hier die mitgelieferte LIB51 (V4.24), die sämtliche Routinen zum Betrieb des 89C51 μC enthält.

5.2 Anwendersoftware

Als erstes werden die gespeicherten Daten der Karte in das Dateiformat .CSV (**c**omma **s**eperated **v**alues) konvertiert (Converter.exe). Dieses Dateiformat ist zwar in keinster Weise standardisiert, ist jedoch in der RFC 4180 in ihren Grundzügen beschrieben und weit verbreitet. Zudem können Dateien in diesem Format mit gängigen Tools wie zum Beispiel MS Excel©, OpenOffice Calc oder anderen zum Großteil kostenlos erhältlichen Programmen geöffnet bzw. bearbeitet werden.

Nach der Messung gelangen die vorliegenden Daten zur grafischen Darstellung. Hier können die Daten über einen wählbaren Zeitraum abgebildet werden. So kann die Verkehrsbelastung (eine Differenzierung bzgl. der verschiedenen Fahrzeugtypen ist hier nicht möglich) zu jedem erfassten Zeitpunkt betrachtet werden (ConsolenAnalyser.exe).

Beide Programme wurden mit Hilfe von MS Studio© 6.0 erstellt. Für die Erstellung der Software kamen einerseits die Standardbibliotheken des MS Studio© 6.0, andererseits auch ein - für nicht kommerzielle Zwecke - frei erhältliches Bibliothekswerk [12] zur Anwendung.

Die AzLib-Bibliothek umfasst unter anderem Klassen für das Zeichnen und Speichern von Grafiken (z.B das pixelorientierte BMP-Format, das auch in dieser Arbeit Verwendung findet) sowie für die Stringmanipulation und Datenkonvertierung.

Zusätzlich ermöglicht diese Bibliothek auch den Zugriff auf die Windows-Registry (wird im Projekt verwendet [13]).

5.2.1 Formatierung und Auslesen der Speicherkarte mittels Converter.exe

Dieses kleine Konsolenprogramm ermittelt alle physikalischen Laufwerke des Rechners und gibt deren Speichergröße in Sektoren an. Das gewählte Laufwerk wird dann wie in Kapitel 3, Absatz 3.4.2 auf Seite 28 erklärt mit Nullen beschrieben (formatiert).

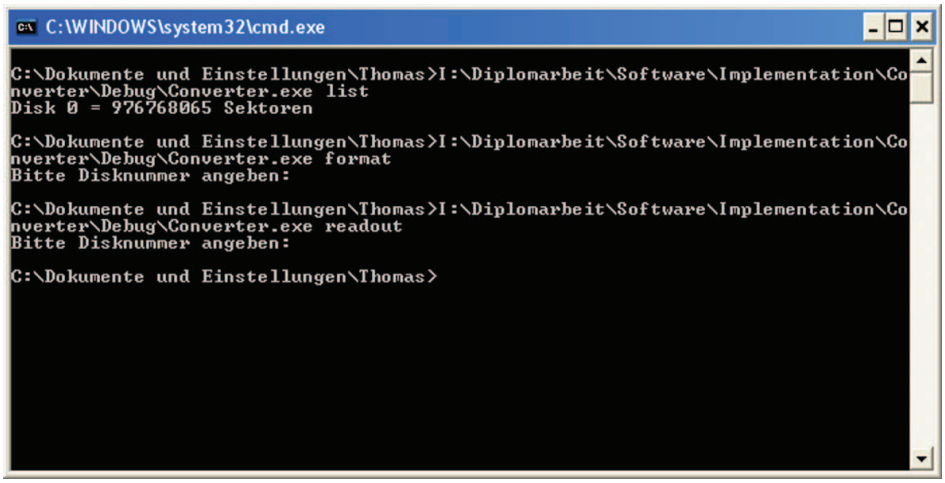


Abbildung 5.3: Konsolenausgabe mit Angabe der verschiedenen Optionen

Ist die Karte mit Daten beschrieben, benutzt man Converter.exe um diese auszulesen. Dabei werden die Daten - wenn notwendig - auch in mehreren Sätzen gespeichert. Das ist dann der Fall, wenn die Messung über mindestens einen Kalendertageswechsel vollzogen wird. So bekommt man detailierte, nach Kalendertagen geordnete, Einzelfahrzeugdaten (Zeit; Fahrzeugtyp; Geschwindigkeit, etc. . .).

Parameter	Beschreibung des Parameters
list	listet sämtliche Datenträger (auch Festplatten) auf und gibt deren Größe in Sektoren an.
format	formatiert die angebene Disk (Angabe der Disknummer erforderlich).
readout	liest die gespeicherten Daten aus und schreibt diese in eine bzw. mehrere .CSV-Datei(en). Hierbei ist die Angabe des Dateinamens und der Disknummer erforderlich

Tabelle 5.1: Parameter von Converter.exe (vgl. auch Abb. 5.3).

5.2.2 Grafische Darstellung mittels ConsolenAnalyser.exe

Zur Anwendung kommt hier ein selbst entworfenes CPP-Konsolenprogramm mit Namen "ConsolenAnalyser.exe", bei dem man verschiedene Optionen (vgl. Tabelle 5.2) angeben muss.

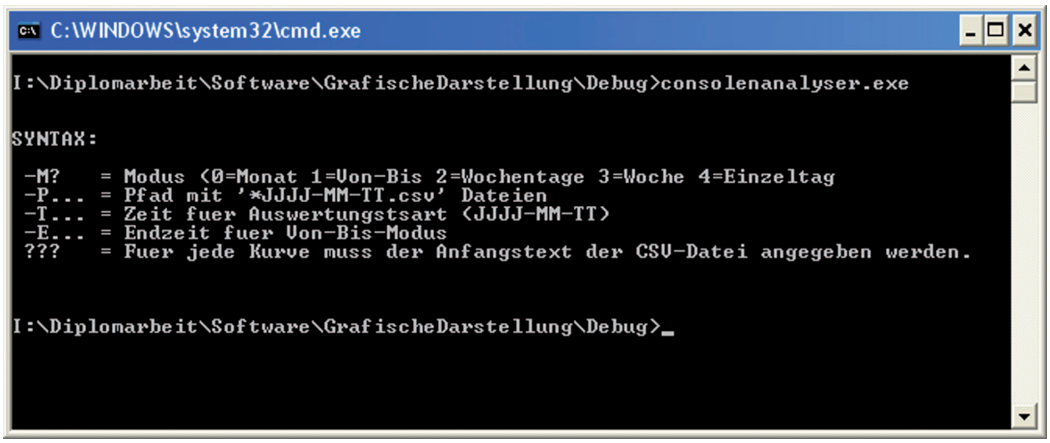


Abbildung 5.4: Konsolenausgabe ohne Angabe von Parametern

Zur Darstellung der Verkehrsdaten über dem gewünschten Zeitraum wird das Ergebnis als Bild (Bitmap) ausgegeben. Um verschiedene Fahrspuhren unterscheiden zu können ist das Bitmap färbig gestaltet (vgl. Abb.: 5.6 auf Seite 53).

Die Programmierarbeit umfasst hier drei CPP-Dateien. Zum ersten Consolenanalyser.cpp, die die Main-Funktion beinhaltet, ImportData.cpp, die die angelieferten Daten behandelt und die StdAfx.cpp, die alleine nur den gleichnamigen Header inkludiert.

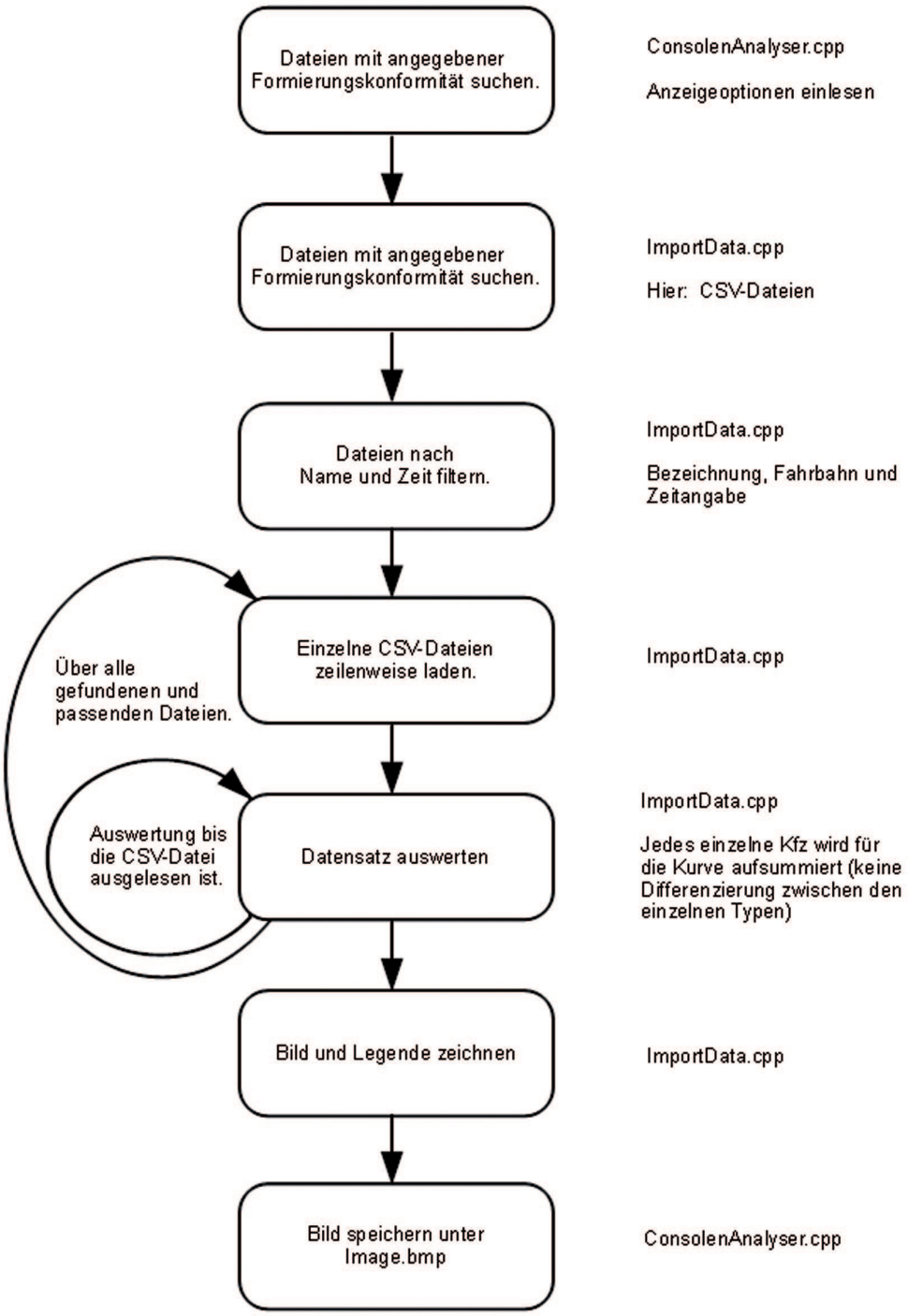


Abbildung 5.5: Flowchart des Programms ConsolenAnalyser.exe

-M (Mode)	Monatsdarstellung	Sämtliche Datensätze für das angegebene Monat
	Zeitraum (von - bis)	von Tag bis Tag (benötigt Parameter -E)
	Wochentage	bestimmte Wochentage
	Woche	sieben Tage
	Einzeltag	ein einzelner Tag
-P (Path)	Absoluter Pfad (z.B.: `D:\Diplomarbeit\Verkehrsdaten`) zu den Verkehrsdaten im .csv-Format. Die anzuzeigenden Daten müssen im Format JJJJ-MM-TT.csv angegeben werden.	
-T Time	Zeit für die jeweilige Auswertungsart in der Form JJJJ-MM-TT	
-E (End)	Endzeit für den Darstellungsmodus „Zeitraum“(-M1)	
-???	Angabe des Anfangstextes der darzustellenden .csv-Daten z.B.: Daten_F1 für Fahrspur 1	

Tabelle 5.2: Darstellung via Konsole. Die Parameter sind nicht case-sensitive.

5.2.3 Darstellungs- und Parametrisierungsbeispiel für ConsolenAnalyser

Als Datengrundlage wurde hier ein anonymisterte Datenreihe für zwei Fahrspuren herangezogen (vgl. Anhang C ab Seite 104).

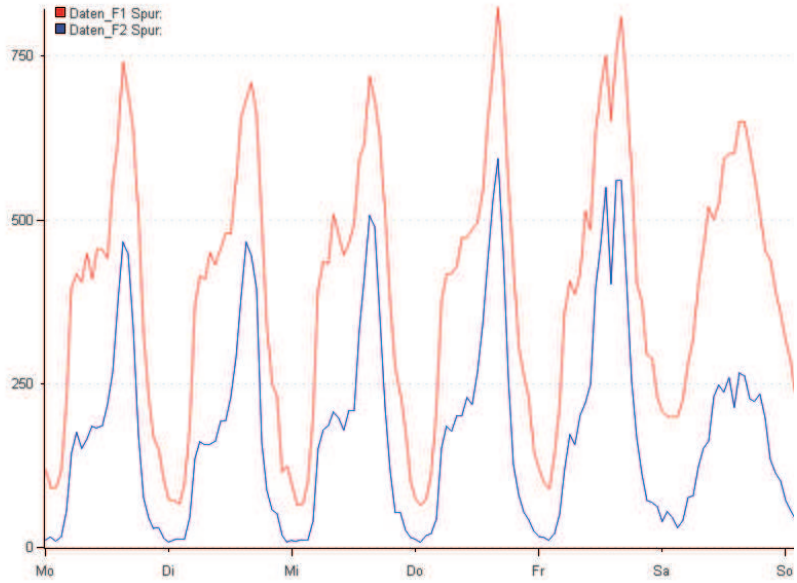


Abbildung 5.6: Sieben-Tage-Trend Ausgabe von ConsolenAnalyser.exe

Die in Abbildung 5.6 dargestellte Graphen wurden mit folgendem Aufruf erzeugt:

```
consolenanalyser -t2011-08-01 -pD:\Diplomarbeit\Verkehrsdaten -m3 Daten_F1 Daten_F2
```

Wobei die Datendateien im .csv-Format für Abb. 5.6 wie folgt benannt sind:

```
Für Fahrspur 1: Daten_F1_JJJJ-MM-TT
Für Fahrspur 2: Daten_F2_JJJJ-MM-TT
```

Diese Benennung kann wie unter Tabelle 5.2 auf Seite 53 angeführt nach der Überspielung auf den PC willkürlich erfolgen. Diese Bezeichnung muss allerdings mit dem ???-Parameter (im obigen Beispiel eben Daten_F1 und Daten_F2) angegeben werden.

In diesem Beispiel sind die Werte für eine Woche (beginnend mit Montag bis einschließlich Sonntag) dargestellt. Der Auswertungsmodus „7-Tage-Trend“ wird hier zur Analyse der Verkehrsdaten herangezogen [13]. Dargestellt sind die Verkehrslasten für zwei Fahrspuren.

In dieser Darstellung sind die Spitzen eindeutig zur Mittagszeit mit durchschnittlich 750 Detektionen. Auffallend in obiger Grafik ist die Tatsache, dass selbst am Wochenende der Verkehr nur marginal abnimmt und diese für Samstag betrachtet sogar zwei im selben Wertebereich befindliche Spitzen für beide Fahrspuren aufweist.

6 Experimentelle Validierung

Die Validierung des Messsytems wurde zuerst im Labor durchgeführt. Nach positiver Bewertung der gelieferten Ergebnisse wurden danach zwei Außenaufbauten durchgeführt. Die erste Außenmessung wurde auf einem Privatgelände durchgeführt, da zu diesem Zeitpunkt die Erlaubnis für eine Messung auf der Straße durch die Bezirkshauptmannschaft Weiz noch nicht vorliegend war. Die zweite Außenmessung konnte nach Erhalt dieser Bewilligung außerorts auf einer Landesstraße durchgeführt werden.

6.1 Labortests

Im Labor wurde die grundsätzliche Funktionalität des Gesamtsystems überprüft. Da der Detektor eine Montagehöhe von zirka 5,5m benötigt ist der Einsatz bei normaler Raumhöhe nicht möglich. So wurde die Kommunikation mit dem Steuerungsmodul respektive Steuerungsplatine mit Hilfe einer softwaremäßigen Lösung getestet.

6.1.1 Aufbau

Die in untere Grafik strichliert abgebildete Busanbindung des Detektors soll verdeutlichen, dass mit dem Detektor im Labor gearbeitet wurde (Hochlauf - Kommunikationsaufbau). Der Großteil der Tests wurde jedoch nur mit der Software von Mitraware durchgeführt.

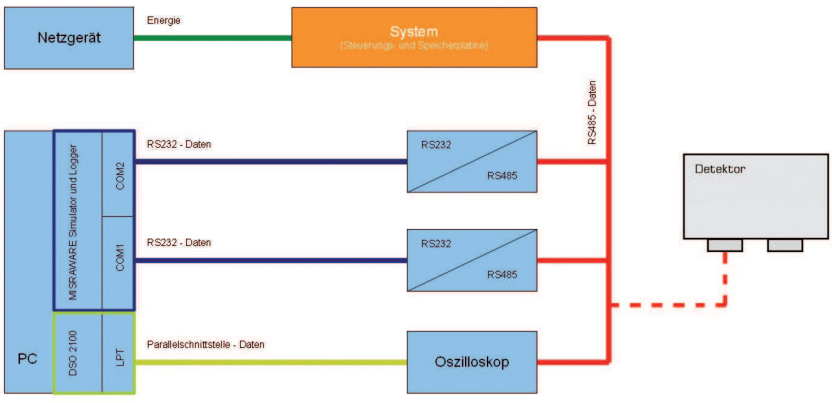


Abbildung 6.1: Blockschaltbild zum Messaufbau.

Beim Messaufbau wurde aus Gründen der Einfachheit auf die Versorgung mittels Akkumulator respektive Batterie (geforderte autarke Energieversorgung) verzichtet. So wurde das Gesamtsystem mit einem AC/DC-Netzgerät energieversorgt. Die Steuerungsplatine stellt dann die Versorgung der Speicherplatine zur Verfügung.

Das Oszilloskop DSO 2100 von Voltcraft misst die Spannungspegel der Übertragung des Detektors und speist die Messdaten via Paralleleingang (LPT) in den Rechner. Ein ABUS Schnittstellenumsetzer konvertiert die von der Simulationssoftware gelieferten Daten für die Steuerungsplatine vom RS232- in einen RS485-Pegel, während der zweite Konverter die Daten der Steuerungsplatine umsetzt, welche via RS232-Eingang in den PC eingespeist werden.

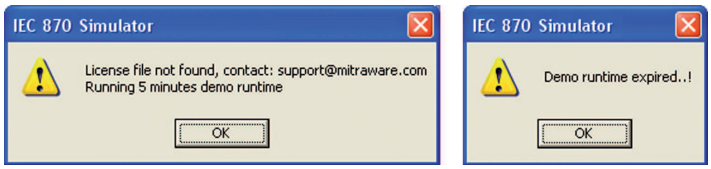


Abbildung 6.2: Mitraware: 5 Minuten Nutzung in der freien Version

Die verwendete Software stellt keine Freeware dar. Sie ist in vollem Umfang funktionsfähig, jedoch nur für einen Zeitraum von fünf Minuten. Dies stellt aber keine gravierende Einschränkung dar, da die Kommunikation Steuerungsplatine - Detektor innerhalb dieses Zeitraumes genau genug untersucht werden konnte.

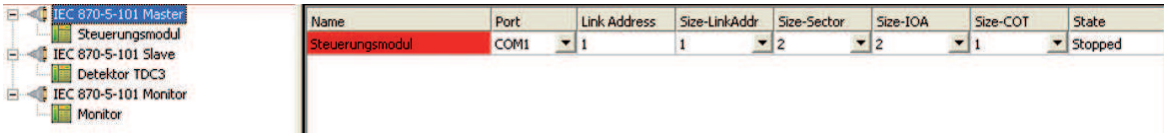


Abbildung 6.3: Mitraware: Master, Slave und Monitor

Die wichtigsten Eigenschaften dieser Software sind: Simulator für IEC 60870 (Master und Slave Protokoll) und Rohdatenerfassung via serieller Schnittstellen (die Rohdaten werden dabei in ein leichter lesbare Darstellung formatiert vgl. Abb.: 6.4).[22]

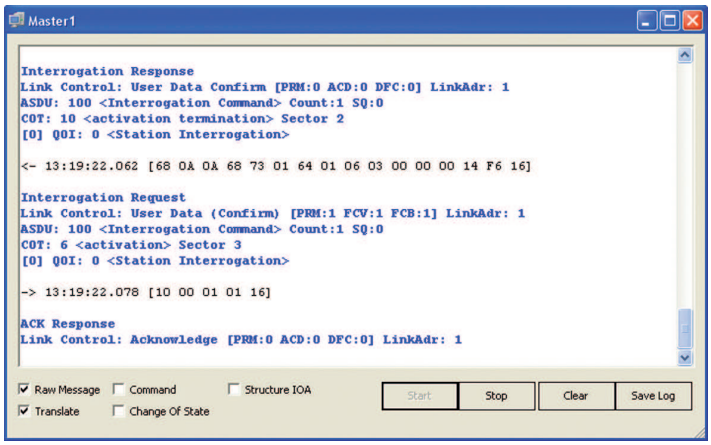


Abbildung 6.4: Mitraware: Datenausgabe [22]

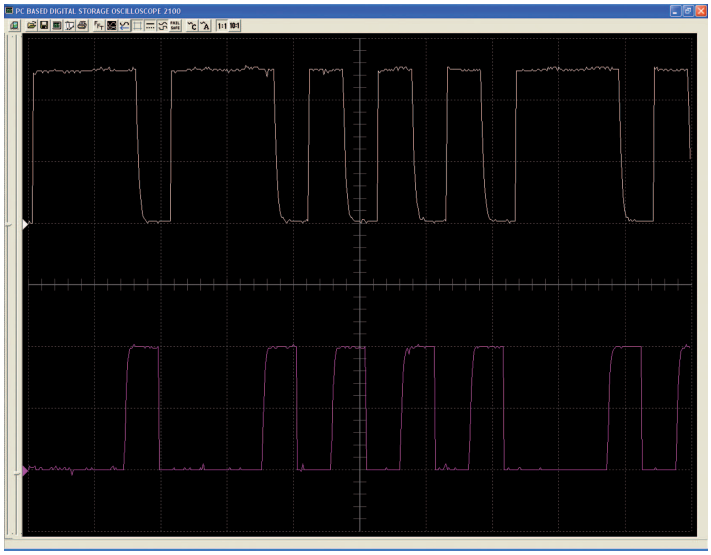


Abbildung 6.5: Übertragung des Detektors via RS485: Aufzeichnung einer Übertragungssequenz während des Hochfahrens des Detektors mittels Oszilloskop Voltcraft DSO2100 am Umsetzer bei TX+ und TX- (A und B) mit den Einstellungen: 2V/Div, 0.2ms @ 9600 Baud

6.1.2 Kommunikationsprotokolle

Zur Verständigung zwischen den Busteilnehmern existieren nach TLS (Technische Lieferbedingungen für Streckenstationen) der Bundesanstalt für Straßenwesen drei Telegrammtypen (Rahmen), die zu drei Telegrammprimitiven (Send/No-Replay, Send/Confirm und Request/Respond) kombiniert werden [15](siehe Kapitel 2, Abschnitt 2.6) .

Der **Long-Frame**, dessen Länge variiert. Dieser Rahmen beinhaltet neben den zu liefernden Messdaten auch ein Kontrollbyte (control byte), ein Adressbyte (address byte), zwei Byte zur Rahmenlängenangabe (length byte: Angabe der Länge[1] mit 2+n) und die Checksumme (checksum) modulo 256. Er beginnt mit 0x68 und endet mit 0x16.
Der **Short-Frame** mit konstanter Länge von 5 Byte, beinnhaltet ein Kontrollbyte, ein Adressbyte und die Checksumme. Er beginnt mit 0x10 und endet wie der Long-Frame mit 0x16.
Der dritte Rahmentyp ist der **Single-Char** der eine, wie der Name schon sagt, feste Länge von einem Byte - mit Inhalt 0xE5 - hat.

Für den genauen Aufbau der Frames und der Telegrammprimitive zur Kommunikation zwischen Sender und Empfänger siehe Abschnitt 2.4 ab Seite 19.

[1] Die Länge setzt sich aus der Anzahl n der Datenbytes plus dem Adress- und dem Kontrollbyte zusammen.

6.1.2.1 Kommunikation bei Detektorhochlauf

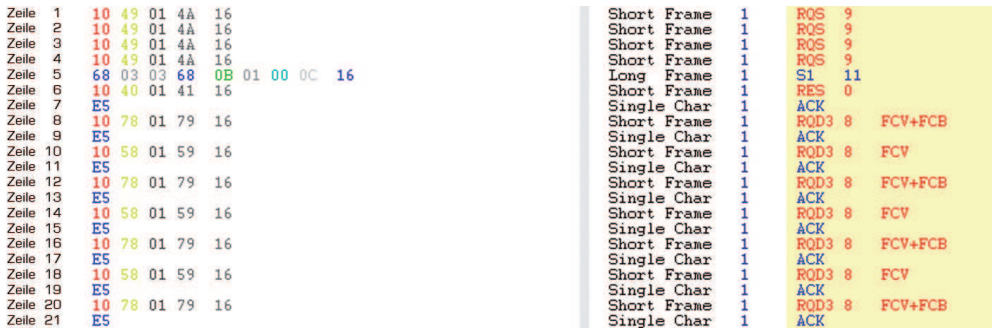


Abbildung 6.6: Kommunikation: Detektorhochlauf. Zeilen die mit roten Ziffern beginnen sind Mastertelegramme. Blaue stellen Slavetelegramme dar.

(1) Zeilen 1 bis 4:

`10 49 01 4A 16` `Short Frame` `1` `RQS 9`

Die Primary (Steuerungsplatine) zeigt die Bereitschaft zum Aufbau einer Verbindung mit RQS an. Die Zeitüberwachung t_0 beginnt.
Ein Short-Frame - Primary → Secondary - ohne Flags (obere vier bit des Steuerbyte: 0100b) mit Befehl Nummer 9 (RQS).

(2) Zeile 5:

`68 03 03 68 0B 01 00 0C 16` `Long Frame` `1` `S1 11`

Die Secondary (Detektor) sendet nach erfolgreichem Empfang die Antwort S1 (Status) mit dem Daten- bzw. Statusbyte 0x00 (kein Fehler vgl. Abb.: 6.8).
Mit 0x68 wird der Beginn eines Long-Frame angezeigt. Byte zwei und drei stellen zweimal das Längenbyte dar (redundante Sendung). Da der Frame aus drei für die Längenberechnung relevanten Bytes besteht ist das Ergebnis hier 0x03 (Steuerbyte, Adressbyte und Datenbyte). Das Steuerbyte 0x0B (obere vier bit: 0000b) zeigt eine Secondary → Primary Verbindung ohne Flags an. Die Lieferadresse ist die Steuerungsplatine. Die Checksumme beläuft sich auf $\sum$(0x0B + 0x01 und 0x00) MOD 256 = 0x0C. 0x16 ist das Endchar des Frames.

(3) Zeile 6:

`10 40 01 41 16` `Short Frame` `1` `RES 0`

Wird S1 von der Primary innerhalb der Zeit t_0 empfangen, sendet sie RES0. Neustart der Zeitüberwachung t_0.

(4) Zeilen 10, 14, 18:

`10 58 01 59 16` `Short Frame` `1` `RQD3 8 FCV`

Die Steuerungsplatine sendet ein Request mit „Prorität 3“(nieder- und hochpriore Daten) samt FCV Flag an den Detektor. 0x10 zeigt einen beginnenden Short-Frame an.

Das Steuerbyte 0x58 zeigt eine Primary → Secondary Verbindung mit gesetztem FVC- und nicht gesetztem FCB-Flag (die oberen vier bit: 0101b) und dem Befehl mit der Nummer 8 (RQD3). Die nächte Eintragung 0x01 repräsentiert das Adressbyte (Lieferadresse). 0x59 ist die über Steuerbyte und Adressbyte gebildete Checksumme $\sum(0x58 + 0x01)$ MOD 256 = 0x59. Den Abschluss bildet das Endchar 0x16.

(5) Zeilen 8, 12, 16, 20:

```
10 78 01 79 16                Short Frame        1          RQD3  8  FCV+FCB
```

Die Steuerungsplatine sendet ein Request Data Signal mit Prorität 3 samt FCV und FCB Flags an den Detektor. Das Alternieren von FCV-Flag und FCV+FCB-Flags nach jedem Acknowledgement des Detektors, bedeutet, dass der Datenrequest erfolgreich war und nicht korrumpiert ist.

(6) Zeilen 7, 9, 11, 13, 15, 17, 19, 21:

```
E5                            Single Char        1          ACK
```

Der Detektor bestätigt den Empfang der Anweisung gegenüber der Steuerungsplatine mit einem Acknowledgement-Signal (einzelnes Steuerzeichen mit fixem Wert E5). Bei dieser Antwort weiß die Steuerplatine, dass der Request verstanden wurde, es keine Daten zur Übermittlung gibt und das keine Detektorfehler vorliegen.

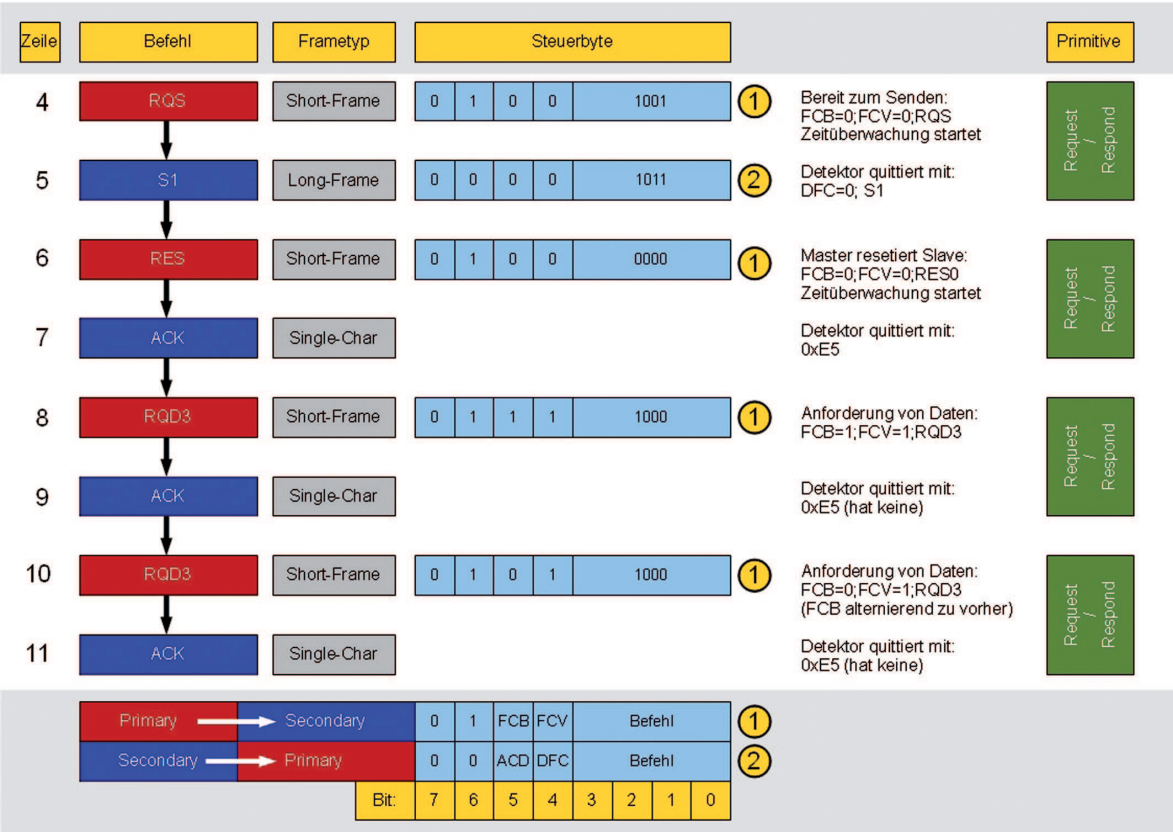


Abbildung 6.7: Inhalt des Steuerbyte bei Detektorhochlauf

Status Byte							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
HW fault	Sync-fault	Queue	Wrong-way driver	Ultrasonic notification	IR 2 notification	IR 1 notification	Radar notification

Abbildung 6.8: Statusbyte des ADEC TDC3: Datum in einem Long-Frame als Antwort auf RQS [18].

6.1.2.2 Kommunikation beim Datenempfang

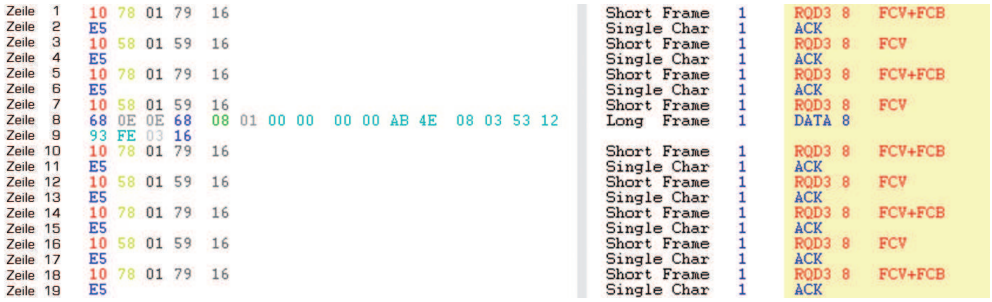


Abbildung 6.9: Kommunikation: Datenempfang. Zeilen die mit roten Ziffern beginnen sind Mastertelegramme. Blaue stellen Slavetelegramme dar.

(1) Zeilen 8 und 9:

```
68 0E 0E 68   08 01 00 00   00 00 AB 4E   08 03 53 12   93 FE 03 16
Long Frame   1
DATA         8
```

Auf die Datenanfrage (Short-Frame) des Steuermoduls (RQD3 + FCV) antwortet der Detektor mit einem Long-Frame (Primitive: Request/Respond). Die ersten vier Byte des Telegramms bezeichnen den Long-Frame Header mit Startbyte: 0x68, Längenbyte: 0x0E, Längenbyte: 0x0E und Startbyte: 0x68.

Das Steuerbyte (0x08; PRM=0; ACD=0; DFC=0; Befehl=DATA) zeigt der Steuerung, das Daten an die Adresse 0x01 (nächstes Byte) geliefert werden. Das auf die Adresse folgende Byte enthält den Status des Detektors (0x00 bedeutet kein Fehler vgl. Abb.: 6.8). Die nächsten vier Byte zeigen die Gesamtanzahl aller bisher von diesem Detektor gezählten Fahrzeuge (Lifetime-Vehicle-Count) an: 0x(00 00 00 AB) entspricht einer Anzahl von 171 Fahrzeugen.

0x4E stellt die Geschwindigkeit von 78 km/h dar. Das nächste Byte repräsentiert den Kfz-Typ. Bei dem hier verwendeten 8+1 Detektor (8 Klassen + nicht klassifizierbar) stellt 0x08 die achte Klasse dar: Lkw mit Anhänger. Die nächsten beiden Byte 0x03 und 0x53 ergeben zusammen die Okkupationszeit in 1/100 Sekunden (vergangene Zeit seit letztem Request) des Kfz: 0x0353 = 851d = 8,51 Sekunden.

Die darauf folgenden zwei Byte stellen die Zeitlücke in 1/100 Sekunden (Time Gap) dar: 0x1293 = 47,55 Sekunden. Das letzte Datenbyte 0xFE stellt die Länge des Kfz in 1/10 m dar: 0xFE = 25,4 m.

Das vorletzte Byte im Telegramm ist die Checksumme 0x03[2], das letzte das TLS-Endchar 0x16.

(2) Zeilen 1-19 ohne 8 und 9:
Zur Erklärung siehe Kommunikation: Detektorhochlauf (Abschnitt 6.1.2.1 auf Seite 58).

6.1.2.3 Zusammenfassung bzw. Bedeutung der Mnemonik

Mnemonik	Bedeutung	Sender	Empfänger
RQD3	Request Data mit Priorität 3 (Datenanfrage)	Steuerungsplatine	Detektor
RQS	Request Status (Statusbericht Detektor)	Steuerungsplatine	Detektor
RES	Reset (FCB=0; nicht gesendete Daten gehen verloren)	Steuerungsplatine	Detektor
FCV	Frame Count Bit Valid (Auswertung des FCB bei 1)	Steuerungsplatine	Detektor
FCB	Frame Count Bit (Telegramm richtig verstanden bei alternierendem FCB)	Steuerungsplatine	Detektor
ACD	Access Demand (Daten vorhanden)	Detektor	Steuerungsplatine
DFC	Data Flow Control (0=Receive Ready)	Detektor	Steuerungsplatine
S1	Statusantwort	Detektor	Steuerungsplatine
DATA	Anwenderdaten	Detektor	Steuerungsplatine

Tabelle 6.1: Befehlsmnemonik der Kommunikation zw. Steuerungsplatine und Detektor (vgl. Abbildungen: 6.7 und 6.9 auf den Seiten 59 und 60)

[2]Die Checksumme wird aus der Summe der Datenbytes plus Steuerungs- und Adressbyte gebildet.

6.1.3 Verwendete Geräte für die Labormessung

Netzgerät	Selbstgebautes Netzgerät mit Spannungs- und Stromregler
System	Steuerungs- und Speicherplatine
Detektor	ADEC TDC3
Umsetzer	ABUS Schnittstellenkonverter RS485 auf RS232
Oszilloskop	Voltcraft DSO 2100
PC	PC (windowsbasierend) mit IEC870 Protokollsoftware von Mitraware und Auswertungs- bzw. Darstellungssoftware für das DS0 2100 Oszilloskop

Tabelle 6.2: Verwendete Geräte für die Labormessung

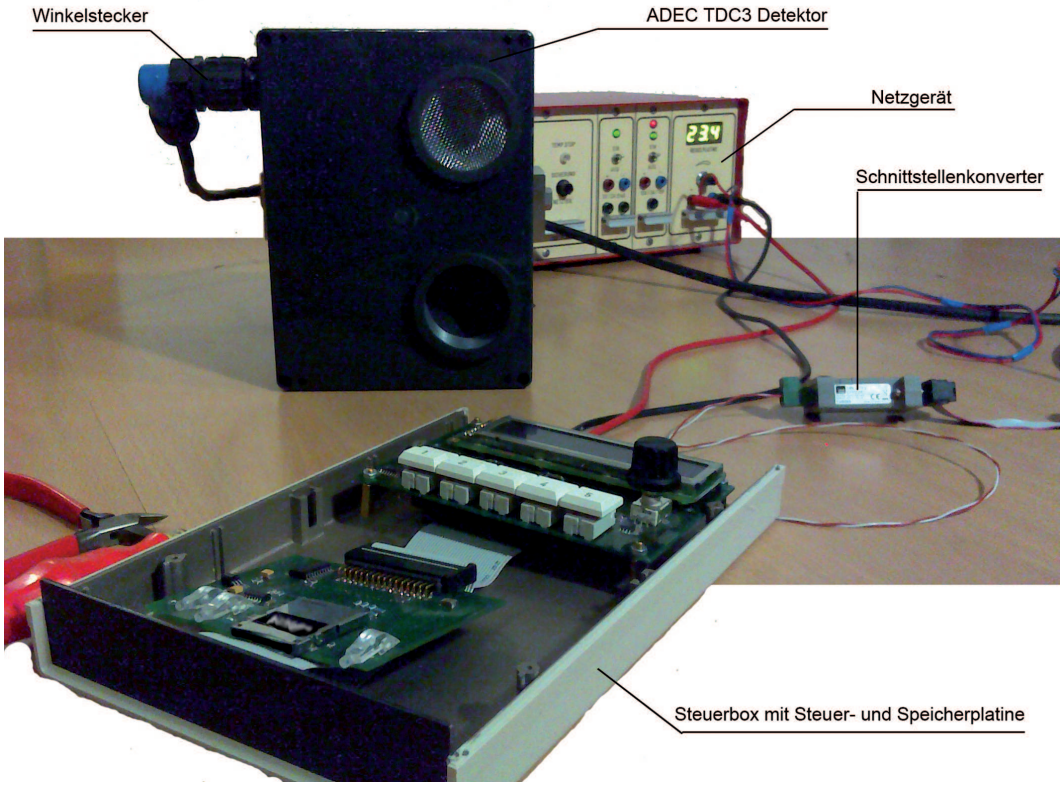


Abbildung 6.10: Laboraufbau mit ADEC TDC3 Detektor (ohne PC)

6.2 Feldmessung

Nach Angaben des Herstellers liefert der Detektor die beste Genauigkeit bei Montage hinter der Schilderbrücke und dem zufließenden Verkehr gegengerichtet - im sogenannten **Frontfire-Modus** (vgl. dazu Anhang B).

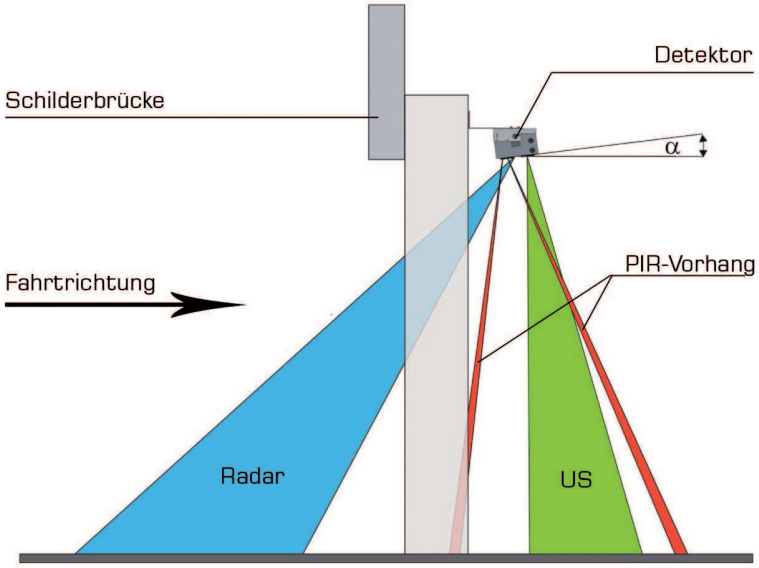


Abbildung 6.11: Detektor Montage hinter einer Schilderbrücke im Frontfire-Modus (der Detektor ist auf der abgewandten Seite zur Fahrtrichtung) [17]. Prinzipiell besteht auch die Möglichkeit der Montage vor der Schilderbrücke.

Der Ultraschallkegel (vgl. Abb.: 6.11 - grün hinterlegt) muss von der Schilderbrücke abgewandt sein (zusätzlicher Neigungswinkel $\alpha = -7°$), um unerwünschte, potentielle Störungen (Reflexionen bzw. Echos) zu vermeiden und damit die vom Hersteller gegebene maximale Genauigkeit und Zuverlässigkeit gewährleisten zu können.

6.2.1 Testareal A und Montage des Systems

Trotz mehrmaligen Urgierens meinerseits beim Stadtamt in Gleisdorf bekam ich keine positive Anwort auf mein Ersuchen auf den öffentlichen Straßen der Stadt eine Messung durchzuführen. Daher musste zunächst auf ein Privatgelände ausgewichen werden.

Für diesen Testlauf gestattete mir ein guter Freund dessen Kanthof zu benutzen. Der Innenbereich dieses Hofs ist groß genug - ca. 15m × 15m - (Schema: siehe Abb.: 6.13 auf Seite 64) um mit Pkws und Lieferwagen Testrunden zu ziehen. Dazu ist die zweistöckige Bauweise des Gehöfts ideal für die Anbringung eines „Galgens“ am Balkongeländer als Aufnehmer für den Detektor in vom Hersteller vorgeschriebenen Höhenbereich von 5m bis 7,5m [17].

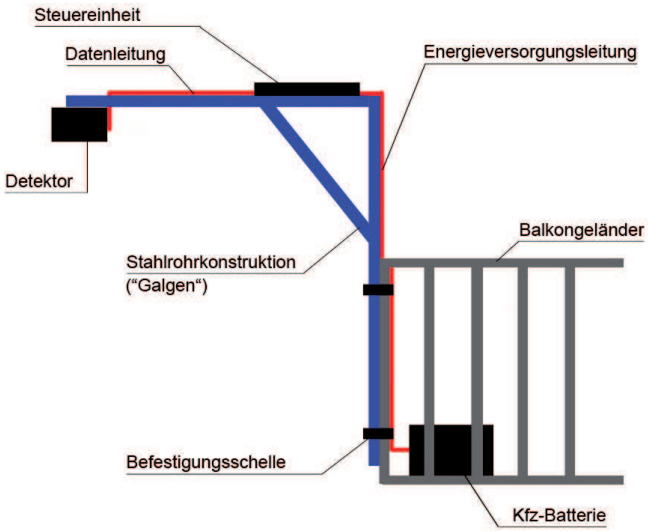


Abbildung 6.12: Außenmontage an Balkongeländer mittels Stahlrohrkonstruktion und Rohrschellen

Die Stahlrohrkonstruktion (siehe Abb.: 6.12) wurde von mir aus verzinkten Formrohren (Außendurchmesser: ∅ 58mm) zugeschnitten und schutzgasgeschweist. Zur Befestigung dieser Konstruktion am Balkongeländer wurden sogenannte „Schwere Rohrschellen mit Anzugsmöglickeit“ verwendet. Der Detektor wurde mit einer selbstgefertigten Plattenverbindung an die Formrohrkonstruktion geschraubt (eine Halterung wie die TDC-MB [17]). Die Steuerungs- und die Speicherplatine wurden in ein Kunststoffgehäuse („Steuerbox“) verbaut und mit handelsüblichen Kabelbindern an der Stahlkonstruktion befestigt. Die Datenleitung (Twisted Pair) wurde an der Steuerbox mittels Western-Plug (RJ45 - Registered Jack 45) und am Detektor mit einem rechtwinkeligen Kabelstecker (ADEC [17]) angeschlossen. Zur Energieversorgung wurde eine handelsübliche Kfz-Batterie der Marke VARTA mit 12V/60Ah an die Steuerbox angeschlossen.

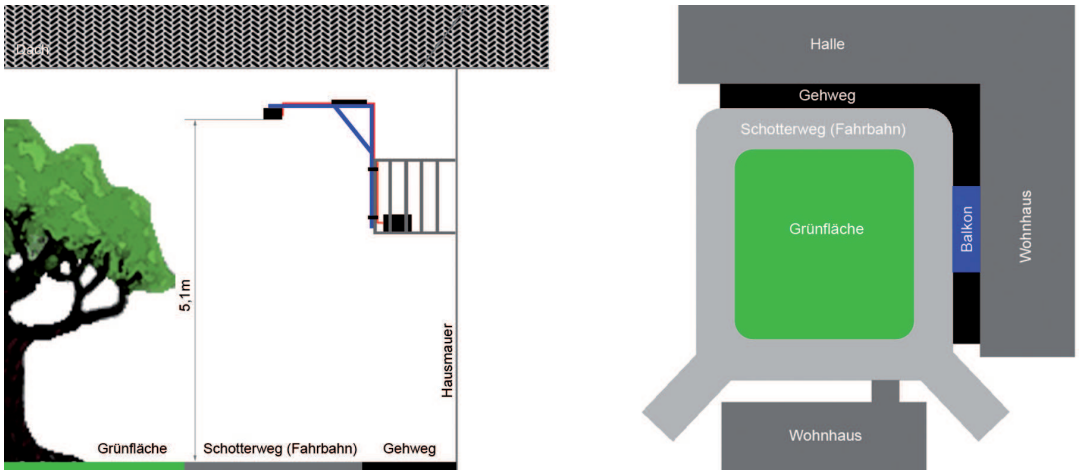
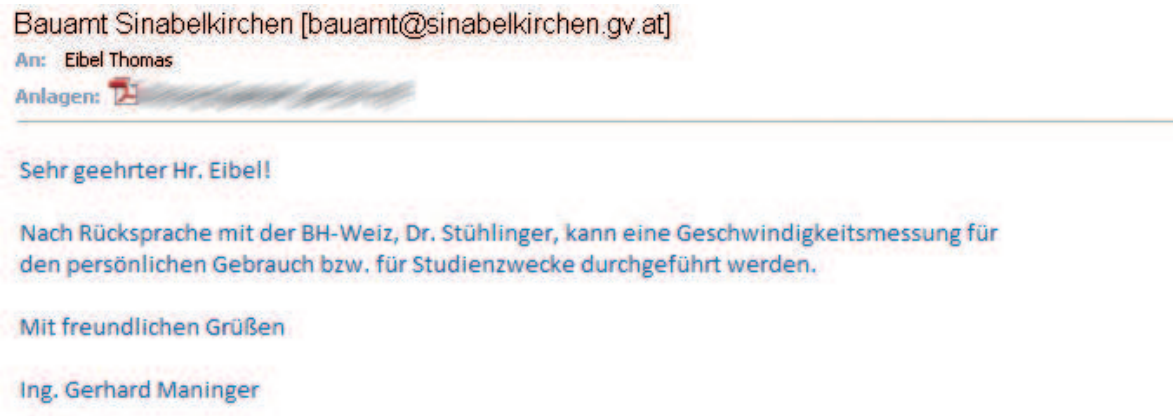


Abbildung 6.13: Auf- und Grundriss des Testareals

6.2.2 Testareal B und Montage des Systems

Die für eine Messung von Verkehrsdaten auf öffentlichen Straßen benötigte Berechtigung wurde mir am 14.02.2012 von Dr. Georg Stühlinger, Leiter des Sicherheitsreferats (Referat III) der Bezirkshauptmannschaft Weiz, erteilt. An dieser Stelle einen riesengroßen Dank an Herrn Vizebürgermeister Anton Kalcher und Bauamtsleiter Ing. Gerhard Maninger von der Marktgemeinde Sinabelkirchen für deren Unterstützung zur Erlangung dieser Berechtigung.


Bauamt Sinabelkirchen [bauamt@sinabelkirchen.gv.at]
An: Eibel Thomas
Anlagen:

Sehr geehrter Hr. Eibel!

Nach Rücksprache mit der BH-Weiz, Dr. Stühlinger, kann eine Geschwindigkeitsmessung für den persönlichen Gebrauch bzw. für Studienzwecke durchgeführt werden.

Mit freundlichen Grüßen

Ing. Gerhard Maninger

Abbildung 6.14: Messerlaubnis: Bestätigungsemail vom Bauamtsleiter

6.2.2.1 Messort

Der Vizebürgermeister der Marktgemeinde Sinabelkirchen gewärte mir die Messung an einem von mir bestimmten Ort innerhalb der Gemeindegrenzen durchzuführen. Preferiert wäre jedoch der Bereich der Geschwindigkeitsbegrenzung in Richtung Egelsdorf, da er davon ausging, dass in diesem Bereich des öfteren die erlaubte Höchstgeschwindigkeit überschritten werde.

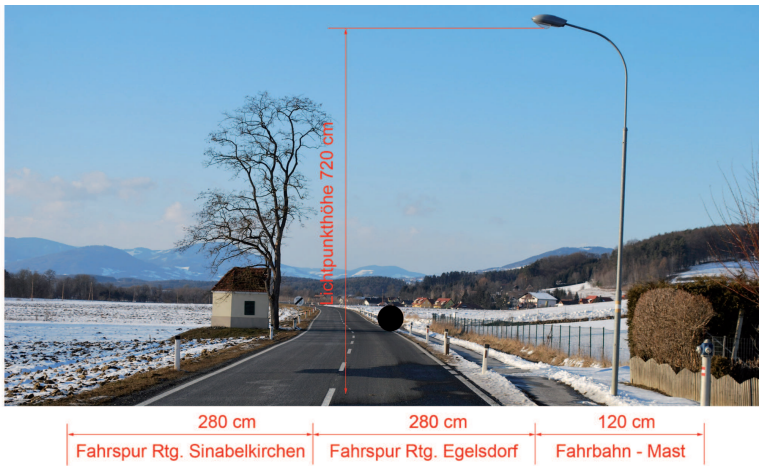


Abbildung 6.15: Abb. 6.15-1: Streckenverlauf der L360 zw. Egelsdorf und Sinabelkirchen (Google Maps) und Abb. 6.15-2: Lichtmast- und Straßenabmessungen

Die Wahl viel auf den Endbereich der L360, die Sinabelkirchen und Egelsdorf verbindet. Idealerweise wurde dort die Straßenbeleuchtung im Bereich der 70 km/h Beschränkung nach dem Ortsende von Sinabelkirchen mit Lichtmasten in sogenannter „Peitschenform“ realisiert, deren Lichtpunkthöhe sich auf ca. 7,2 m befindet. Diese eignen sich hervorragend für eine Montage des Systems für die geforderte Detektorhöhe von 5m bis 7,5m [17].

6.2.2.2 Befestigungsmaterial für Steuerbox, Akkumulator und Detektor

Für die Anbringung an den Lichtmasten (siehe Abb.: 6.15) mussten wieder spezielle Teile gefertigt werden. Zum einen eine Tragevorrichtung (vgl. Abb.:6.16) für die Batterie und die Steuerbox (enthält die Steuer- und Speicherplatine vgl. Abb.:6.10 auf Seite 62) und zum anderen ein Mastausleger (vgl. Abb.: 6.17 auf Seite 67) der den Detektor aufnimmt. Gemessen wurde in Fahrtrichtung Egelsdorf kurz vor Ende der 70 km/h Beschränkung (vgl. Abb.: 6.15 Bild 1).

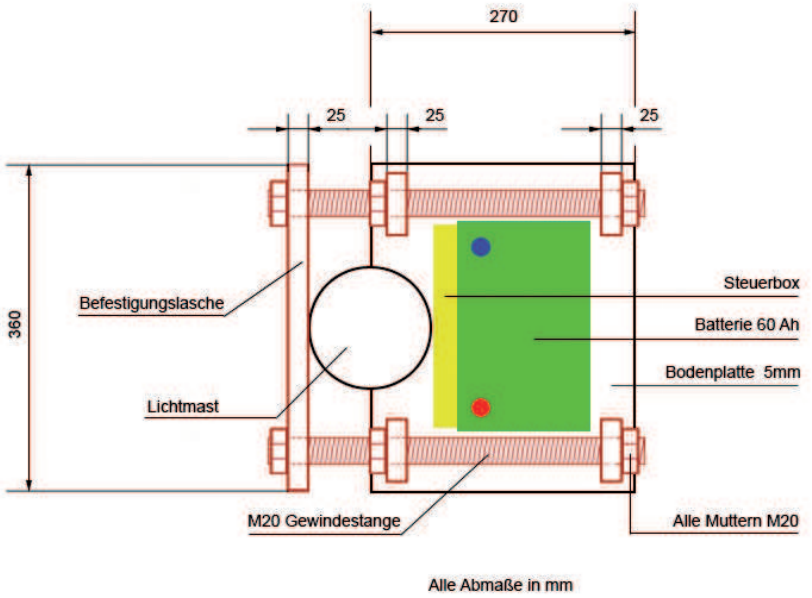


Abbildung 6.16: Tragetasse für die Aufnahme der Batterie und der Steuerbox mit Grobbemaßung

Die Tragetasse wurde in ca. 1,5 m Höhe am Mastfuß montiert. Der Ausleger wurde mit Hilfe eines Kranwagens auf ca. 5,8 m angebracht. Um die Zugbelastung der Gewindestange entgegen zu wirken, wurde die in etwa 6 kg schwere Konstruktion mittels Halteseil (Sicherungsseil) und Karabiner am Mastkopf befestigt. Der Detektor wurde mit einer Stahlblechhalterung an den Ausleger montiert und mit einem Sicherungsdraht aus verdrillten Stahladern daran befestigt.

Um die Verkehrsteilnehmer nicht zu gefährden, wurde die Vorrichtung nicht mittig über der Fahrspur platziert, sondern oberhalb des Fußwegbereichs repektive neben der Straße.

Während der Messung (inkl. der Zeit für Auf- und Abbau) standen „Absperrposten“ um etwaige Fußgänger und Radfahrer um den Gefahrenbereich herumzuleiten. So konnte garaniert werden, dass niemand durch meine Vorrichtung innerhalb des Messzeitraums zu Schaden kommt.

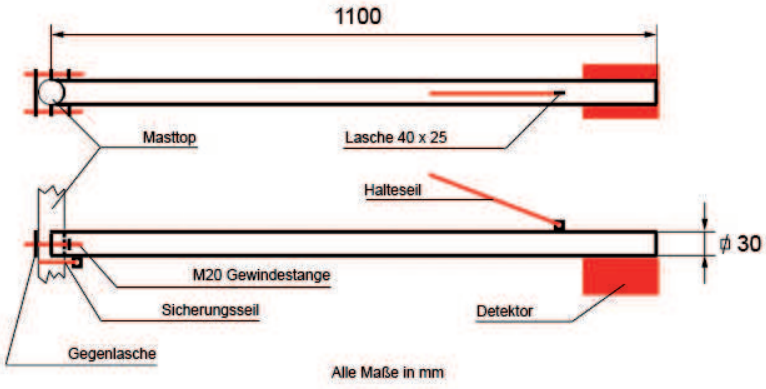


Abbildung 6.17: Detektorformrohrausleger für die Mastmontage mit zusätzlicher Abspannung zum Mastkopf mittels Halteseil.

6.3 Ergebnisse

Da der Detektor der Messvorrichtung nicht den Vorgaben entprechend genau über der Fahrbahnmitte positioniert werden konnte, konnte ich mir nicht sicher sein, dass die gemessenen Werte auch den tatsächlichen entsprechen. Deswegen lies ich ich während des Messzeitraums drei Referenzfahrten mit 70 km/h Tachogeschwindigkeit durchführen und führte eine Strichliste (Kfz-Klassifizierung), wie sie auch bei einer händischen Verkehrszählung genutzt wird, in der ich diese Referenzfahrten auch dokumentierte (vgl. Anhang C.3).

Klasse	Anzahl in Strichliste	Anzahl auf Speicherkarte
Kfz gesamt	102	96
Pkw	92 (87)	85
Pkw mit Anhänger	1 (1)	2
Lieferwagen	9 (8)	7
Lkw	- (-)	2

Tabelle 6.3: Gegenüberstellung der Messdaten von Strichliste und Speicherkarte. Die Werte in Klammern repräsentieren die Anzahl der Kfz, die auch auf der Speicherkarte festgehalten wurden. (Nicht zwangsweise typengerecht: Im Messzeitraum wurde z.B. kein Lkw verzeichnet. Dennoch enthält die Messung zwei Lkw - falsche Klassifizierung).

Die unterschiedlichen Anzahlen zwischen Strichliste und Gespeicherte Daten rühren vom realisierten Speicheralgorithmus her. Da eine SD-Karte nur sektorweise (512 byte; entspricht 32 Messungen zu 16 byte) beschrieben werden kann, müssen die für diesen Umfang benötigten Daten zuerst im RAM des Mikrocontrollers abgelegt werden. Erst beim Vorliegen eines ganzen

Sektors wird dieser dann auf die Speicherkarte geschrieben. So beinhaltet die Speicherkarte immer eine Datenanzahl von modulo 32 (vgl. Kapitel 3 Absatz 3.4.2 auf Seite 28). Bezüglich der in Tabelle 6.5 dargestellten Daten für „Anzahl → Kfz gesamt“ bedeutet das, dass drei Sektoren beschrieben wurden und der Rest auf 102 beim Abschalten des Messsystems verloren ging.

Klasse	Anzahl laut Strichliste	laut Messung detektiert als
Pkw	2	Lieferwagen
Pkw mit Anhänger	1	Lieferwagen
Lieferwagen	2	LKW

Tabelle 6.4: Falsche Kfz-Klassifizierungen

Die obige Tabelle stellt die vom Detektor falsch detektierten Kfz dar. Bei einem Messdatensatz von 96 und fünf falschen Klassifizierungen ergibt das eine relative Detektionsfehlerrate x_{Fehler} von

$$x_{\text{Fehler}} = \frac{5}{96} \times 100 = 5,21\% \tag{6.1}$$

was einem Fehler bei in etwa jedem 20 Kfz entspricht.

Die falsche Klassifizierung in diesem Ausmaß wird von mir unter anderem auf die seitliche Montage des Detektor rückgeführt. Die Multiple-PIR Vorhänge des Detektors deckten aufgrund der seitlichen Anbringung nur gut 80% der Fahrbahn ab.

Eine weitere Fehlerquelle wird auch beim Detektor selbst liegenDa es eher unwahrscheinlich ist, dass der ADEC TDC3-8 Detektor bei 8+1 Fahrzeugklassen mit einer Fehlerrate von 0% arbeitet, wird der relative Fehler x_{Fehler} von 5,21 % aus einer Superposition aus seitlicher Montage und Eigenfehler des Detektors herühren.

Dabei hoffe ich inständig, dass der Eigenfehler einen marginalen Anteil an diesem Fehler besitzt. Man denke da nur an die vielen Messstellen an Autobahnen, wo Kfz-Daten für statistische Auswertungen (oder Geschwindigkeitsüberprüfungen, etc. . .) gesammelt und aufbereitet werden. Wenn man diesen auch einen relativen Fehler in obiger Größenordung zugesteht, dann wird auf unseren Autobahnen und Schnellstraßen zirka jedes zwanzigste Kfz einer falschen Klasse zugeordnet.

Um so bemerkenswerter ist es, dass trotzdem verwertbare und vorallem stimmige Daten vorliegen. Denn die oben genannten Referenzfahrten mit einer Tachogeschwindigkeit von 70 km/h (Opel Astra; Länge: 4,11 m) erzielten - wie nachfolgende Tabelle (Tab.: 6.5) zeigt - sehr gute Ergebnisse:

Messfahrt	Messdatum	Zeit	Geschwindigkeit	Länge
1	28	14:20 h	64 km/h	41 dm
2	39	14:27 h	66 km/h	41 dm
3	46	14:36 h	66 km/h	42 dm

Tabelle 6.5: Gegenüberstellung der Referenzfahrten: Die Geschwindigkeitsdifferenz auf 70 km/h rührt einerseits vom subjektiven Ablesefehler und andererseits von der Tachovoreilung her. Die Messungenauigkeit des Detektors (vgl. Tabelle 3.1 auf Seite 26) von ±3 km/h (≤ 100 km/h) wurde nicht berücksichtigt.

6.3.1 Darstellung der Messreihe mittels ConsolenAnalyser

Durch den geringen Datenvorrat (Messdauer: 1,5 h, 96 Kfz) ist die untere Grafik bestenfalls ein Beleg, dass das System in gefordertem Umfang arbeitet.

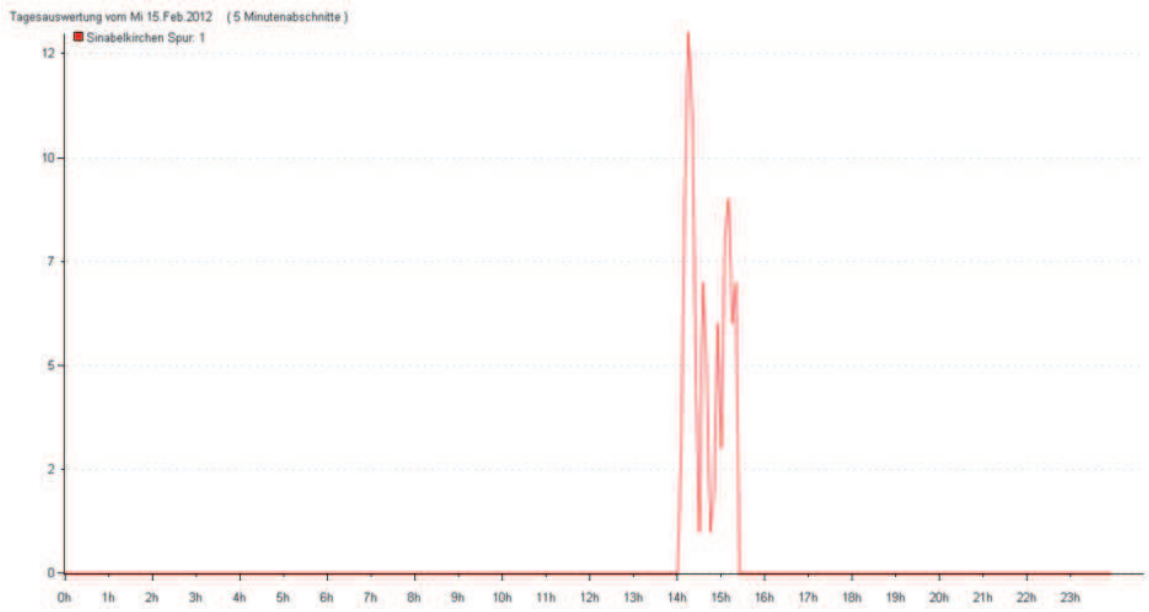


Abbildung 6.18: Anzeige der gemessenen Daten mittels ConsolenAnalyser.exe

Die vorhandene Datenaufzeichnung lässt keinen vernünftigen Schluss bezüglich tatsächlicher Verkehrsbelastung am Messort zu. Die Möglichkeit einer genaueren (visuellen) Auswertung entfällt hier.

6.3.2 Darstellung der Messreihe mittels TrafficAnalyser

Die Grafik in Abb. 6.19 zeigt eine Auswertung des selben Datenvorrats mit dem Programm „TrafficAnalyser“ [13]. TrafficAnalyser wurde im Zuge eines Projekts basierend auf dem Quellcode von ConsolenAnalyser entworfen.

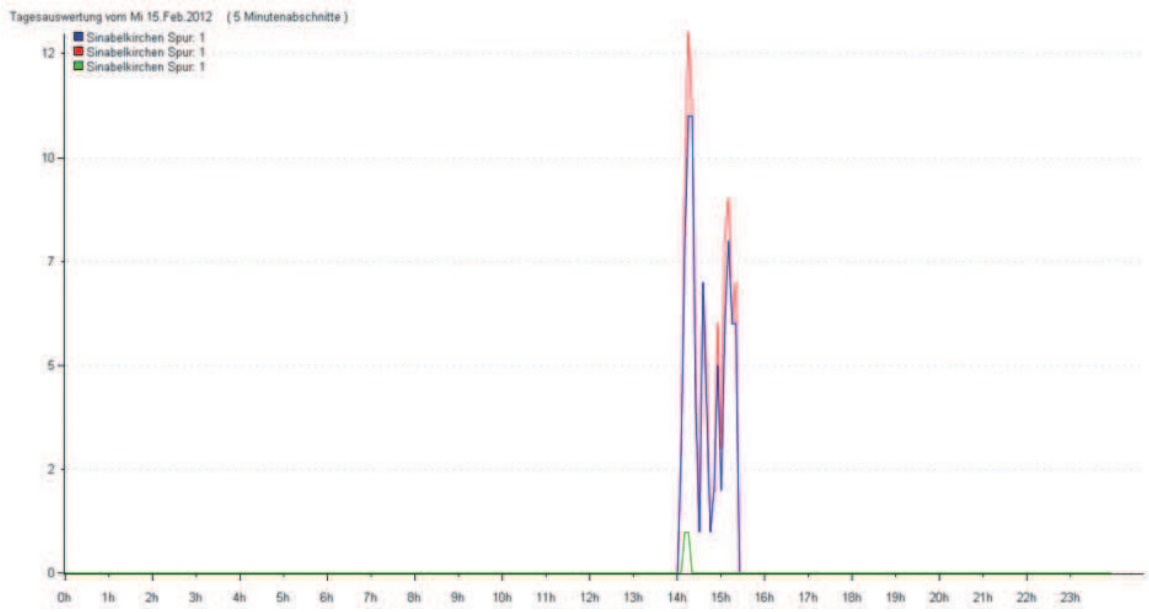


Abbildung 6.19: Anzeige der gemessenen Daten mittels TrafficAnalyser.exe (Rot: Gesamtanzahl Kfz; Blau: Pkw; Grün: Pkw mit Anhänger).

In Abbildung 6.19 ist klar zu erkennen, dass beinahe die Gesamtverkehrsbelastung aus Pkw besteht. Eine solche Differenzierung ist mit ConsolenAnalyser nicht möglich.

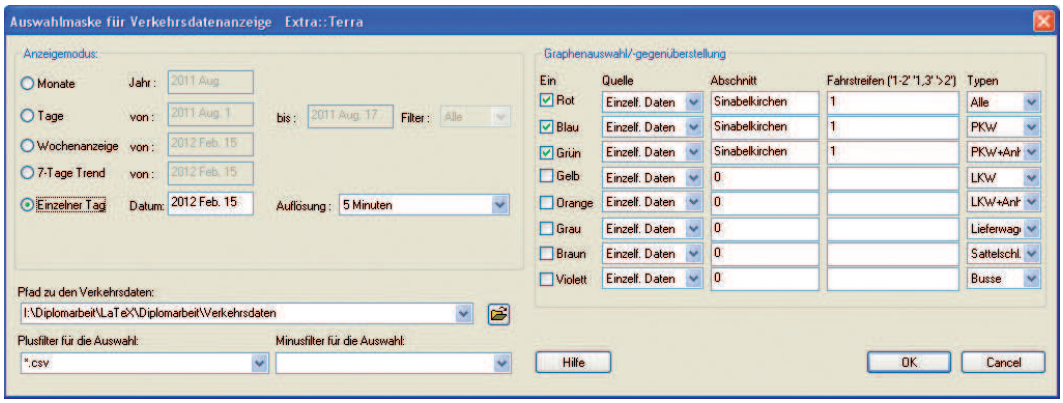


Abbildung 6.20: Die Eingabemaske des Programms TrafficAnalyser

Für die Ergebnisgrafik (Abb. 6.19) wurden in die Eingabemaske drei unterschiedliche Kfz-Typen eingeben (inklusive Kfz-Typ: Alle). TrafficAnalyser erleichtert im Gegensatz zu ConsolenAnalyser die Eingabe von den anzuzeigenden Daten und man hat unter anderem - wie oben erwähnt - die Möglichkeit mehrere Kfz-Typen darzustellen [13].

7 Schlussbetrachtungen

Das vorliegende (stationäre) System ist für die Klassifizierung im Ort,- Stadt oder Überlandgebiet vorallem für Kurzzeitmessungen nur bedingt tauglich, da die Montagearbeiten für jeden Messpunkt nur mit größerem Aufwand vollzogen werden können.

Ein Lösungsansatz wäre hierfür der oben erwähnte Entwurf eines portablen Systems. Eine solche Konstruktion müsste allerdings strengsten Prüfungen hinsichtlich der mechanischen Stabilität unterzogen werden, da das System während der Messung Winddruck und Sogwirkung (Verkehr) ausgesetzt wird.

Einer Fixmontage an Durchfahrten (Brücken, Unterführungen, etc ...) auch für längere Messzeiträume steht jedoch nichts im Wege.

7.1 Folgerung der Arbeit

Im Prinzip kann man mit den Ergebnissen der Arbeit zufrieden sein. Die Aufgabenstellungen wie in Kapitel 3 gefordert wurden erfüllt.

Natürlich ist der geringe Datenvorrat aufgrund der geringen Auslastung während des Messzeitraums ein gravierender Kritikpunkt dieser Arbeit. Jedoch kann man mit Sicherheit behaupten ein funktionierendes System entworfen zu haben, das noch dazu sehr billig gegenüber den sich am Markt befindlichen ist.
Die Gesamtkosten - ohne Entwicklungszeit - belaufen sich inklusive Detektor auf etwa € 1.700,-

Aufgrund der sehr kurzen Messdauer von zirka 1,5 h kann auch keine zuverlässige Aussage bezüglich des Funktionszeitraums getätigt werden. Zwar wurde im Labor der gesamte Messaufbau über einen Zeitraum von einer Woche getestet. Dies lässt aber keinen zuverlässigen Schluss auf das Verhalten des Systems im Aussenbereich bei verschiedenen Witterungsbedingungen zu.

7.2 Diskussion und Ausblick

Trotz der befriedigenden Ergebnisse sind, will man die Stufe einer gewissen „Serienreife“ erreichen, weitere Verbesserung des Prototypsystems vorzunehmen.

Zum einen kann nach jetzigem Wissensstand die von mir vorgenommene Modularität bzgl. einer eigenen Steuerungs- und Speicherplatine entfallen. Auch eine Schaltungszusatz zum

Flashen des Messdatensatzes mittels Taster sollte noch implementiert werden. Ansonsten gehen x = Datensätze mod 32 beim Abschalten - so geschehen bei meiner Messung - verloren.

Zum anderen wäre es ein großer Vorteil, wenn das System als Ganzes portabel ist. Eine diesbezügliche Möglichkeit wäre das System auf eine Art Kfz-Angänger zu montieren um so, wie zum Beispiel bei Baustellenampeln bereits umgesetzt, die Messeinheit leicht an den Verwendungsort transportieren zu können.

Des weiteren ist eine Verbesserung der PC-Software (ConsolenAnalyser.exe) hinsichtlich der Bedienbarkeit und der Auswertungsmöglichkeiten vorzunehmen. Die Bedienbarkeit eines Konsolenprogramms ist eher unübersichtlich und daher für den ungeübten Nutzer nicht empfehlenswert (Nutzung von TrafficAnalyser.exe).

Eine Zusammenfassung von Darstellungs- und Formatierungssoftware würde das Paket kompakter und somit leichter nutzbar machen. Dabei sollte von der Softwarelösung „TrafficAnalyser“, die im Projekt[13] beschrieben wird, ausgegangen werden.

Die Montagearbeiten können für eine Messung bei Nutzung eines Überkopfdetektors nicht ohne Hebebühen bzw. Steiger durchgeführt werden. Vorallem ist hier heraus zu streichen, dass es bei einer nach ADEC geforderten Montage des Detektors mittig über der Fahrbahn zu eine Gefährdung der Verkehrsteilnehmer durch den langen Mastausleger kommen würde. Ein solcher müsste eigens angefertigt und statisch und dynamisch auf sein Verhalten während einer Messperiode getestet werden um einer Gefährdung entgegen zu wirken.

Jedoch liegen die Vorteile für eine Gemeinde/Kleinstadt beim Inbetrachtziehen einer Verkehrsmessung mit Fahrzeugklassifizierung durch die Verwendung dieses Systems meiner Meinung nach dar:

1. Überwachung von noraglischen Verkehrspunkten Schulen, Kindergärten, Pensionistenheimen, Blindenheimen, Krankenhäusern, etc ...
2. Überwachung von verkehrsberuhigten Zonen wie Fußgängerzonen und Wohnstraßen.
3. Bemessungsgrundlage für soziale Maßnahmen zur Verbesserung der Lebensqualität der Anrainer in Verkehrsbelastungsgebieten.
4. Überprüfung der Verkehrsbelastung bei Anrainerbeschwerden (Verkehrslärm in der Nacht).
5. Überprüfung hinsichtlich Mautflüchtlinge, Quell- und Zielverkehr und als Beurteilungsgrundlage zur Errichtung dementsprechender Fahrverbotszeiten.
6. Energieautarke und billige Ermittlung von Verkehrshotspots im Einzugsgebiet.
7. Erweiterte Einsatzmöglichkeit: Einfache Signalisierung von Geschwindigkeitsüberschreitungen bei zusätzlicher Verwendung von Wechselverkehrszeichen respektive durch Ansteuerung von visuellen Signalisationsmittel mittels Relais.
8. etc ...

Für eine Weiterentwicklung des Systems müssen Vergleichsmessungen mit verschiedenen Detektorpositionen durchgeführt werden. Dabei soll ermittelt werden wie groß der Unterschied respektive die Messwertunterschiede bei mittiger zu seitlicher Montage sind und ob diese - falls vorhanden - reproduzierbar und somit korrigierbar sind.

Desweiteren soll der Detektor bei verschiedenen Verdrehungswinkeln getestet werden. Hier soll der Detektor mittels Stellschraube in einen bestimmten Winkel α zum Ausleger (vgl. Abb. 7.1) gebracht werden damit die PIR-Vorhänge die gesamte Fahrbahnbreite abdecken können.

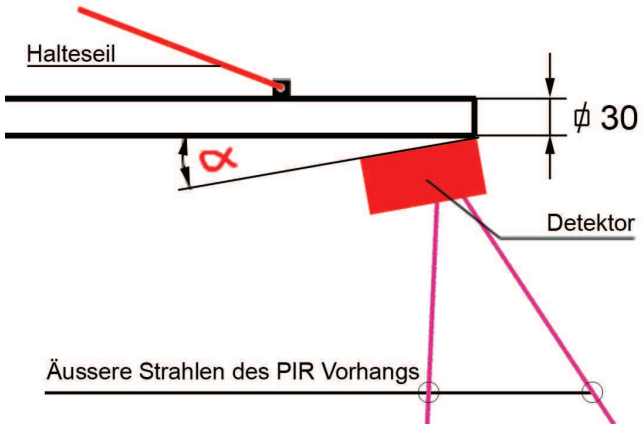


Abbildung 7.1: Messung mit zum Ausleger um den Winkel α verdrehtem Detektor

Liefern diese oben erwähnten Tests die gewünschten Ergebnisse (Korrigierbarkeit der Ergebnisse bei verschiedenen Verdrehungswinkeln und Detektorpositionen) sollen diese in einer „Montageanleitung“ zusammengefasst werden. Diese soll dann beim Einsatz des Systems dem Arbeiter die gewünschten Einstellinformationen liefern.

Literaturverzeichnis

[1] Dr.-Ing. Tilo Gockel - Form der wissenschaftlichen Ausarbeitung.
Springer-Verlag, Heidelberg, 2008. Begleitende Materialien unter:
`wwwiaim.ira.uka.de/form-der-wissenschaftlichen-ausarbeitung`

[2] BM für Verkehr, Innovation und Technologie
Ausbauplan Bundesverkehrsinfrastruktur 2011-2016

[3] Steiermärkische Berg- und Naturwacht
Aufgaben und Leistungen

[4] Statistik Austria
Fundiertes statistisches Material zu den Verkehrszweigen

[5] Norm für Akustikbegriffe
DIN 1320
Ausgabe: 2009-12

[6] Thomas Beierlein, Taschenbuch Mikroprozessortechnik
Ausgabe: 2004

[7] Jürgen Göbel, Radartechnik - Grundlagen und Anwendungen
Ausgabe: 2011

[8] Multimedia Card
Product Manual
Lit. No. 80-13-00089

[9] Secure Digital Memory Card
Product Manual - Version 2.2
Document No. 80-13-00169

[10] Keil C51 Development Tools
`www.keil.com/c51`

[11] Rainer Steinbrecher, Bildverarbeitung in der Praxis
Ausgabe: 2005

[12] C++ Bibliothek zur freien nicht kommerziellen Nutzung
`azsoftware.az.funpic.de/az/index.html`

[13] Thomas Eibel, Erweiterung eines bestehenden C++ Programms zur Verkehrsdatenanzeige
Projekt an der Technischen Universität Graz
2012

[14] Liquid Cristal Display Character Modules - Application Modules
Application Notes - Second Edition

[15] Technische Lieferbedingungen für Streckenstationen (TLS)
BASt (Bundesanstalt für Straßenwesen)
Ausgabe 2002

[16] IEC 60870-5-1
Fernwirkeinrichtungen und Systeme - Übertragungs-Frame-Formate
Companion standard for basic telecontrol tasks
Second edition 2003-02

[17] Installationshandbuch Serie TDC3
Überkopfdetektoren für die Verkehrsdatenerfassung
Ausgabe: 2010 - 07

[18] ADEC Protocol Specification for TLS-Compatible Traffic Data Acquisition Using TDC1, TDC3 and TDC4 Series of Traffic Detectors
Version 1.2 - Ausgabe 2010 - 07

[19] TDC3 - Überkopfverkehrsdetektoren für die spurselektive Verkehrsdatenerfassung
Ausgabe: 2010 - 07

[20] Xtralis ASIM Tri-Tech Detektoren Triple Technologie Kombinationsdetektoren
Dok. Nr. 14444 04

[21] Datenblatt des 89C51RDplus
Product Specification
IC28 Data Handbook - 1999 Oct 27

[22] Capture- und Simulatorsoftware für den IEC 870-5-1 Protokollstandard
Mitraware
`http://www.mitraware.com`

A Quelltexte

Anhang A gibt einen kurzen Überblick über die Software, die für diese Diplomarbeit programmiert wurde.

A.1 Hauptroutine für den 89C51RDplus

Mikrocontroller Programm — TrafficCounter.c

```
//*****************************************************************************
//*     TrafficCounter.c
//*
//*****************************************************************************
#include   „Reg51F.h"
#include   <string.h>

#include   „I2cAccess.h"
#include   „LcdUtility.h"
#include   „ComUtility.h"
#include   „MmcCardApi.h"
#include   „RealTimeClock.h"
#include   „BusConnection.h"

sbit   KEY_0   = P2^0;
sbit   KEY_1   = P2^1;
sbit   KEY_2   = P2^2;
sbit   KEY_3   = P2^3;
sbit   KEY_4   = P2^4;
sbit   KEY_5   = P2^5;
sbit   Q_SET   = P1^5;

extern   void        TurnSwitchInit();
extern   unsigned char  TurnSwitchState;
static   unsigned char  ucLastTurn;

xdata   unsigned long   ulMmcWritePos=0;
xdata   unsigned long   ulMmcWriteSize=0;
xdata   unsigned char   aCardId[16];
xdata   unsigned char   ucTemp[256] _at_ 0x0000;
xdata   unsigned char   ucTime[7];
extern   unsigned       uiMsTicker=0;

// konvertiert die Detektor-Fahrzeugtypen
xdata   unsigned char  ucClassToType[]=      {
// (0=nkKFZ 1=kRAD 2=PKW 3=LFW 4=PKW+ 5=LKW   6=LKW+ 7=Sat 8=Bus)
            9,2,4,5,6,8,0,2,   // 0... 7
            6,7,1,3,9,9,9,9,   // 8...15
            9,9,9,9,9,9,9,9,   // 16...23
            9,9,9,9,9,9,9,9,   // 24...31
            2,5,               // 32...33
            };
// MMC Datenformat:
typedef   struct  // ACHTUNG: Größe und Reihenfolge der Variablen nicht verändern
   {
   unsigned char   ucAddr;          // Adresse des Sensors
   unsigned char   ucType;          // Fahrzeugtyp
   unsigned char   ucLength;        // Fahrzeuglänge (10cm Schritte)
   unsigned char   ucVelocity;      // Geschwindigkeit
   unsigned short   usDiffTime;     // zeitlicher Abstand zw. Fahrzeugen (ms)
   unsigned short   usMeasureTime;  // Zeitstempel
   unsigned char   ucDec;           // 1/100 sec im BCD-Format
   unsigned char   ucSec;           // Sekunden im BCD-Format
   unsigned char   ucMin;           // Minuten im BCD-Format
   unsigned char   ucHour;          // Stunden im BCD-Format
   unsigned char   ucYear;          // Jahr im BCD-Format
   unsigned char   ucMonth;         // Monat im BCD-Format
   unsigned char   ucDay;           // Tag im BCD-Format
   unsigned char   ucFree;          // unbenutzt
   }TrafficData;

#define   MAX_SLOTS   128
bit            bMmcError  = 0;
bit            bSetTime  = 0;
unsigned char     uSlotWr   = 0;
unsigned char     uSetMax   = 0x99;
unsigned char     uSetIndex  = 4;
unsigned char     uSetCursor = 64+2;
unsigned char     uBusByte  = 0xFF;
xdata TrafficData   aTrafficData[MAX_SLOTS];

//*****************************************************************************
//*
//*     Timer2Interrupt
//*
//*****************************************************************************
```

Keil - Programmierumgebung — µC:89C51RDplus

Mikrocontroller Programm TrafficCounter.c

```
// Prozedur wird alle 1ms aufgerufen

void Timer2Interrupt() interrupt 5
{
   uMsTicker++;
}

//*****************************************************************************
//*
//*       ThreadSleep
//*
//*****************************************************************************
//

void ThreadSleep(unsigned uMs)
{
unsigned     uTick=uMsTicker;
unsigned     uDiff;


   for(;;)
      {
      uDiff = uMsTicker-uTick;
      if(uDiff>=uMs)break;
      }

}

//*****************************************************************************
//*
//*       DecodePaket
//*
//*****************************************************************************
//    Dekodiert Paket von Detektor und speichert es

void DecodePaket(unsigned char *pData,unsigned char uSize,unsigned char
ucAddr)
{
TrafficData       *pTraffic;
```

Keil - Programmierumgebung µC:89C51RDplus

```
unsigned char    ucVeocity;
unsigned char    uCount;
unsigned char    uClass;
unsigned char    uLen;
unsigned char    uPos;
unsigned char    uOk;


  if((uSize-5)%6==0)                          // Größe überprüfen
     {
     uLen   = 6;
     uCount = (uSize-5)/6;
     if(uCount>=1 && uCount<=8)uOk=1;
     }
  else if((uSize-5)%7==0)
     {
     uLen   = 7;
     uCount = (uSize-5)/7;
     if(uCount>=1 && uCount<=8)uOk=1;
     }
  else if((uSize-5)%11==0)
     {
     uLen   = 11;
     uCount = (uSize-5)/11;
     if(uCount>=1 && uCount<=8)uOk=1;
     }


  if(uOk)                                     // Paket überprüfen
     {
     for(uPos=0;uPos<uCount;uPos++)
        {
          uClass   =     pData[5+1+uPos*uLen]&0x3F;
        if(uClass>=sizeof(ucClassToType) || ucClassToType[uClass]==9)
           {
           uOk = 0;
           break;
           }
        }
```

Mikrocontroller Programm TrafficCounter.c

```
    }

  if(uOk)                                   // Dekodiere Verkehrsdaten
    {
    for(pData+=5;uCount>0;uCount--,pData+=uLen)
      {
        ucVeocity = pData[0];
      if(ucVeocity==0)
        {
        continue;
        }

      uClass  =    pData[1]&0x3F;

      if(uClass>=sizeof(ucClassToType))continue;
      if(uSlotWr>=MAX_SLOTS)continue;

      pTraffic                                  =  aTrafficData+uSlotWr;
      pTraffic->ucAddr                  =  ucAddr;
      pTraffic->ucType                  =  ucClassToType[uClass];
      pTraffic->ucLength                = (uLen>=7)? pData[6]:0;
      pTraffic->ucVelocity              =  ucVeocity;
      pTraffic->usDiffTime              = (pData[5]|(pData[4]<<8));
      pTraffic->usMeasureTime               = (pData[3]|(pData[2]<<8))*10;
      memcpy(&pTraffic->ucDec,ucTime,7);
      uSlotWr++;


if(uSlotWr<32)continue;           // Falls die Daten für einen ganzen Sektor (512 byte)
                                  // vorhanden sind ... -->

      if(MmcWriteBlock(ulMmcWritePos,(unsigned char*)aTrafficData))
                                  // --> ... speichere diese Daten
        {
        uBusByte |=  0x08;                  // LED2 einschalten (Write LED)
        P0       = uBusByte;
        Q_SET    = 1;
        Q_SET    = 0;
```

Keil - Programmierumgebung µC:89C51RDplus

```
        bMmcError=0;
        uSlotWr-=32;
        ulMmcWritePos++;
        memcpy(aTrafficData,aTrafficData+32,sizeof(TrafficData)*uSlotWr);
        }
      else{                                 // Reinitialisiere MMC-Karte
        bMmcError=1;
        MmcReset();
        MmcGetState();
        MmcIdentifyCard(0,aCardId);
        MmcIdentifyCard(1,aCardId);
        }
      }
    }
}

//*****************************************************************************
//*
//*       GetCardSize
//*
//*****************************************************************************
//    Ermittelt die Größe der MMC-Karte und sucht die Schreibposition auf der Karte

void GetCardSize()
{
long    iStart;
long    iStop;
long    iHalf;


  LCD_CURSOR(64);
  LCD_TEXT(„Scan MMC Card     „,20);
  ThreadSleep(2000);

Reinit:

  MmcReset();                           // Initialsiere MMC-Karte
  MmcGetState();
  MmcIdentifyCard(0,aCardId);
```

Mikrocontroller Programm TrafficCounter.c

```
MmcIdentifyCard(1,aCardId);

ulMmcWriteSize = MmcGetSize();

if(ulMmcWriteSize<0x8000)                    // berechne MMC-Kartengröße
   {
   LCD_CURSOR(64);
   LCD_TEXT(„MMC Card too small    „,20);
   ThreadSleep(2000);
   goto Reinit;
   }

LCD_CURSOR(64);
LCD_TEXT(„Scan MMC Card      „,20);


iStart = 0;                                  // beginne mit ERSTEM Sektor
iStop  = 0x8000-1;                           // 0x8000*512 = 16MB

for(;;)              // finde ERSTEN FREIEN Sektor (binärer Suchalgorithmus)
   {
   iHalf = (iStop-iStart+1)/2;

   if(!MmcReadBlock(iStart+iHalf,(unsigned char*)aTrafficData))
      {
      goto Reinit;
      }

   if(!iHalf)                                // letztes Element
      {
      if(aTrafficData[0].ucAddr)iStart++;
      ulMmcWritePos = iStart;
      break;
      }

   if(aTrafficData[0].ucAddr==0)             // weiter in unterem Kartenabschnitt
      {
      iStop = iStart+iHalf-1;
      if(iStart>iStop)iStop =iStart;
```

Keil - Programmierumgebung µC:89C51RDplus

```
      }
   else{                                     // weiter in oberen Kartenabschnitt
      iStart = iStart+iHalf+1;
      if(iStart>iStop)iStart=iStop;
      }
   }
}

//*************************************************************************
//*
//*       main
//*
//*************************************************************************
void main()
{
TlsPacket  *pPacket;
bit          bKey;
bit          bBitKey0=0;
bit          bBitKey1=0;
bit          bBitKey2=0;
bit          bBitKey3=0;
bit          bBitKey4=0;
bit          bBitKey5=0;
bit          bLastError;
unsigned     uLastOutput;
unsigned     uLastKey0;
unsigned     uLastKey1;
unsigned     uLastKey2;
unsigned     uLastKey3;
unsigned     uLastKey4;
unsigned     uLastKey5;
unsigned     uDiff;
char        cDelta;

   AUXR    = 0x01;                           // ALE-Signal ausschalten
   RCAP2L    = -691*4;                         // f = 1000 Hz
   RCAP2H    = -691*4>>8;                        // f = 11.0592 MHz/4/-X
   T2CON    = 0x04;                           // Konfiguration von Timer 2
```

Mikrocontroller Programm TrafficCounter.c

```
PCON   |= 0x80;                    // Y=16 bei SMOD=1 ***  Y=32 bei SMOD=0
EA       = 1;                     // Interrupts einschalten
P1       = 0xFF;
Q_SET   = 0;

LCD_INIT();
LCD_RESET();
LCD_CFG(0);
LCD_CURSOR(0);
LCD_TEXT(„Traffic Counter      „,20);
ThreadSleep(3000);

uBusByte &= ~0x0C;        // LED1+LED2  ausschalten (Error LED und Write LED)
uBusByte |= ~0x10;        // LED3 einschalten (Init LED)
P0       = uBusByte;
Q_SET    = 1;
Q_SET    = 0;

TurnSwitchInit();                  // Alles initialisieren
SpiInitialize();
BusInit();
GetCardSize();
bSetTime = 0;

uBusByte &= ~0x10;                 // LED3 ausschalten (Init LED)
P0       = uBusByte;
Q_SET    = 1;
Q_SET    = 0;

//************************* main loop ************************************
for(;;)
    {
   bKey = KEY_0;

   if(bKey!=bBitKey0)              // Taster0:    gedrückt?
      {
      uDiff = uMsTicker-uLastKey0;
      if(uDiff>50)
         {
```

Keil - Programmierumgebung µC:89C51RDplus

```
      bBitKey0  = bKey;
      uMsTicker = uLastKey0;

      if(bKey)                    // Zeitmodus ändern
         {
         bSetTime  = !bSetTime;

         if(bSetTime)
            {
            LCD_CFG(3);           // Cursor einschalten
            ucLastTurn = TurnSwitchState;
            }
         else{
            TimeSet(ucTime);      // Neue Zeit in Real-Time-Clock speichern
            LCD_CFG(0);           // Cursor ausschalten
            }
         }
      }
   }
if(!bSetTime)
   {
   TimeGet(ucTime);               // Zeit von Real-Time-Clock einlesen
   }
else{                             // Zeit mit Tasten (1-5) abändern
   bKey = KEY_1;

   if(bKey!=bBitKey1)             // Taster1: gedrückt?
      {
      uDiff = uMsTicker-uLastKey1;
      if(uDiff>51)
         {
         bBitKey1   = bKey;
         uMsTicker  = uLastKey1;
         uSetMax    = 0x99;       // JAHR einstellen
         uSetIndex  = 4;
         uSetCursor = 64+2;
         }
      }
   bKey = KEY_2;
```

Mikrocontroller Programm TrafficCounter.c

```
if(bKey!=bBitKey2)                  // Taster2:    gedrückt?
   {
   uDiff = uMsTicker-uLastKey2;
   if(uDiff>52)
      {
      bBitKey2   = bKey;
      uMsTicker  = uLastKey2;
      uSetMax    = 0x12;           // MONAT einstellen
      uSetIndex  = 5;
      uSetCursor = 64+5;
      }
   }
bKey = KEY_3;

if(bKey!=bBitKey3)                  // Taster3:    gedrückt?
   {
   uDiff = uMsTicker-uLastKey3;
   if(uDiff>53)
      {
      bBitKey3   = bKey;
      uMsTicker  = uLastKey3;
      uSetMax    = 0x31;           // TAG einstellen
      uSetIndex  = 6;
      uSetCursor = 64+8;
      }
   }
bKey = KEY_4;

if(bKey!=bBitKey4)                  // Taster4:    gedrückt?
   {
   uDiff = uMsTicker-uLastKey4;
   if(uDiff>54)
      {
      bBitKey4   = bKey;
      uMsTicker  = uLastKey4;
      uSetMax    = 0x23;           // STUNDE einstellen
      uSetIndex  = 3;
      uSetCursor = 64+12;
```

Keil - Programmierumgebung µC:89C51RDplus

```
      }
   }
bKey = KEY_5;

if(bKey!=bBitKey5)                  // Taster5:    gedrückt?
   {
   uDiff = uMsTicker-uLastKey5;
   if(uDiff>55)
      {
      bBitKey5   = bKey;
      uMsTicker  = uLastKey5;
      uSetMax    = 0x53;           // MINUTE einstellen
      uSetIndex  = 2;
      uSetCursor = 64+15;
      }
   }
}
if(bSetTime)                        // Zeiteinstellung: Drehtaster
   {
   cDelta = ucLastTurn-TurnSwitchState;

   while(cDelta>0)                  // Drehung: Zeitwert inkrementieren
      {
      cDelta--;
      ucTime[uSetIndex]++;

      if((ucTime[uSetIndex]&0x0F)>9)
         {
         ucTime[uSetIndex] &= 0xF0;
         ucTime[uSetIndex] += 0x10;
         }

      if((ucTime[uSetIndex]&0x0F)>uSetMax)
         {
         ucTime[uSetIndex] = (uSetIndex<4)? 0:1;
         }
      ucTime[0] = 0;
      ucTime[1] = 0;
      }
```

Mikrocontroller Programm TrafficCounter.c

```
        while(cDelta<0)                 // Drehung: Zeitwert dekrementieren
          {
          cDelta++;
          ucTime[uSetIndex]--;

          if((ucTime[uSetIndex]&0x0F)>9)
            {
            ucTime[uSetIndex] &= 0xF0;
            ucTime[uSetIndex] -= 0x10;
            }

          if((ucTime[uSetIndex]&0x0F)>uSetMax ||
            (ucTime[uSetIndex]==0 && uSetIndex>=4))
            {
            ucTime[uSetIndex] = uSetMax;
            }
          ucTime[0] = 0;
          ucTime[1] = 0;
          }
        }
//****************** empfangen und speichern *******************************

        pPacket = BusLoop();
      if(pPacket)
        {
        DecodePaket(pPacket->ucData,pPacket->ucDataLenght-8,pPacket->sHdr.
sCom.ucAddress);
        }

//****************** LCD-Display: Anzeige erneuern ***************************

      uDiff = uMsTicker-uLastOutput;

      if(uDiff>=1000)                   // Zeitanzeige erneuern
        {
        uLastOutput += 1000;

        if(bLastError!=bMmcError)       // Fehleranzeige erneuern
```

Keil - Programmierumgebung µC:89C51RDplus

```
        {
        bLastError = bMmcError;
        LCD_CURSOR(0);
        LCD_TEXT((bMmcError)? „** MMC Card Error **“:“Traffic Counter „,20);

        if(bMmcError)               // LED1 schalten (Error LED)
              uBusByte |=  0x04;
        else    uBusByte &= ~0x04;

        uBusByte &=  ~0x80;         // LED2 ausschalten (Write LED)
        P0       = uBusByte;
        Q_SET    = 1;
        Q_SET    = 0;
        }
      LCD_CURSOR(64);                   // JAHR
      LCD_PRINT(,2‘);
      LCD_PRINT(,0‘);
      LCD_PRINT(,0‘+ucTime[4]>>4);
      LCD_PRINT(,0‘+ucTime[4]&0x0F);
      LCD_PRINT(,-‘);
      LCD_PRINT(,0‘+ucTime[5]>>4);      // MONAT
      LCD_PRINT(,0‘+ucTime[5]&0x0F);
      LCD_PRINT(,-‘);
      LCD_PRINT(,0‘+ucTime[6]>>4);      // TAG
      LCD_PRINT(,0‘+ucTime[6]&0x0F);
      LCD_PRINT(, ,);
      LCD_PRINT(, ,);
      LCD_PRINT(,0‘+ucTime[3]>>4);      // STUNDE
      LCD_PRINT(,0‘+ucTime[3]&0x0F);
      LCD_PRINT(,:‘);
      LCD_PRINT(,0‘+ucTime[2]>>4);      // MINUTE
      LCD_PRINT(,0‘+ucTime[2]&0x0F);
      LCD_PRINT(,:‘);
      LCD_PRINT(,0‘+ucTime[1]>>4);      // SEKUNDE
      LCD_PRINT(,0‘+ucTime[1]&0x0F);
      LCD_CURSOR(uSetCursor);
      }
  }
}
```

A.2 Hauptroutine für ConsolenAnalyser.exe

Programm zur Grafische Auswertung via Konsole

```
//*****************************************************************************
//*
//*           trafficAnalyser.cpp
//*
//*****************************************************************************
#include      „StdAfx.h“
#include      „ImportTlsData.h“

static unsigned char    aFont10[] =
        {
        #include „../AzLib/Image/Fonts/Arial10.h“
        };

TlsImportData   m_sImportData;
int             m_iImageMode=3;
int             m_bShowLegend=1;

//*****************************************************************************
//*
//*           SetFilterText
//*
//*****************************************************************************
void SetFilterText(CString &sText,unsigned char *pData)
{
const char      *pAdd=““;
CString         sTemp;
unsigned        uPos;
unsigned        uNum;

        sText = „“;

        for(uPos=0;uPos<256;uPos++)
                {
                if(pData[uPos]<=1)continue;

                for(uNum=1;uPos+uNum<256;uNum++)
```

Visual Studio 6.0 TrafficAnalyser.cpp

```
                        {
                        if(pData[uPos+uNum]<=1)break;
                        }

                if(uNum==256)break;

                if(uNum==1)
                        {
                        sTemp.Format(„%s%u“,pAdd,uPos);
                        }
                else if(uNum==2)
                        {
                        sTemp.Format(„%s%u,%u“,pAdd,uPos,uPos+1);
                        }
                else{
                        sTemp.Format(„%s%u-%u“,pAdd,uPos,uPos+uNum-1);
                        }

                uPos  += uNum-1;
                sText += sTemp;
                pAdd   = „,“;
                }
        }

//*****************************************************************************
//*
//*           CmpFilter
//*
//*****************************************************************************
int CmpFilter(CStringChain *pList,int iPos1,int iPos2)
{
const char    *pText1;
const char    *pText2;
int                     iNum1;
int                     iNum2;
int                     iLen1;
int                     iLen2;
int                     iCmp;
```

Programm zur Grafische Auswertung via Konsole

Visual Studio 6.0 TrafficAnalyser.cpp

```
        pText1  = pList->Get(iPos1);
        pText2  = pList->Get(iPos2);

        iNum1   = atoi(pText1);
        iNum2   = atoi(pText2);

        if(iNum1!=iNum2)return iNum1-iNum2;

        iLen1   = pList->Len(iPos1);
        iLen2   = pList->Len(iPos2);

        if(iLen1<iLen2)
                        iCmp = strnicmp(pText1,pText2,iLen1);
        else    iCmp = strnicmp(pText1,pText2,iLen2);

        if(iCmp)return iCmp;

        if(iLen1!=iLen2)return iLen2-iLen1;

return stricmp(pText1,pText2);
}

//*****************************************************************************
//*
//*                 DrawImage
//*
//*****************************************************************************
void DrawImage(BitmapImage *pImage)
{
static    LPCTSTR aDateStr[16]={_T(„Jan.“),_T(„Feb.“),_T(„März „),_T(„April
„),_T(„Mai „),_T(„Juni „),_T(„Juli „),_T(„Aug.“),_T(„Sep.“),_T(„Okt.“),_T(„Nov.“
),_T(„Dez.“  ),_T(„??? „),_T(„??? „ ),_T(„??? „ ),_T(„??? „)};
#define                 FRAME_T                 35
#define                 FRAME_B                 40
#define                 FRAME_L                 70
#define                 FRAME_R                 20
#define                 FONT_Y                  16
CStringChain            sList;
CString                 sFilter;
```

```
char                    cText[256];
char                    cTemp[64];
char                    cFrom[64];
double                  dHigh;
double                  dStep;
double                  dFull;
double                  dRest;
double                  dVal;
int                     iPixelY;
int                     iStartX;
int                     iStartY;
int                     iFrameX;
int                     iFrameY;
int                     iSizeX;
int                     iSizeY;
int                     iOldY;
int                     iValY;
int                     iStep;
int                     iLast;
unsigned                uSlot;
unsigned                uMax;
unsigned                uVal;
int                     iPos;
int                     iVal;
int                     iCnt;
int                     i,j;

        iSizeX  = pImage->Width();
        iSizeY  = pImage->Height();

        if(iSizeX<1024)iSizeX=1024;
        if(iSizeY<500 )iSizeY=500;

          iCnt   = m_sImportData.iTimeCount;
        if(iCnt<=0)iCnt=1;

        iStartX  = FRAME_L;
        iStartY  = iSizeY-FRAME_B;
        iFrameX = iSizeX-FRAME_L-FRAME_R;
```

Programm zur Grafische Auswertung via Konsole

Visual Studio 6.0 TrafficAnalyser.cpp

```
iFrameY = FRAME_T+FRAME_B-iSizeY;
iStep    = iFrameX/iCnt;

pImage->Box(0,0,iSizeX,iSizeY,BGR(255,255,255));

// Koordinatenkreuz
pImage->Box(iStartX-1,iStartY+1,1,iFrameY,BGR(0,0,0));
pImage->Box(iStartX-1,iStartY+1,iFrameX,1,BGR(0,0,0));


uMax = 1;

for(iPos=0;iPos<iCnt;iPos++)                    // Maximum suchen
        {
        for(uSlot=0;uSlot<m_sImportData.uFilterCount;uSlot++)
                {
                if(uMax>=aTlsData[uSlot+iPos*8])continue;
                uMax = aTlsData[uSlot+iPos*8];
                }
        }

// Höhen Skalierung ausgeben
pImage->SetPixelFunction(TEXT_SMOTH,0,0);
pImage->SetPixelFunction(TEXT_NONE ,0,1);
pImage->SetFont(aFont10);
pImage->SetAlign(2);

iPixelY= -iFrameY;
iVal   = iPixelY/75;

dHigh  = (double)iPixelY/(double)uMax;
dStep  = (double)uMax/(double)iVal;
dRest  = modf(log10(dStep),&dFull);
dStep /= pow (10.0,dFull);


 if(dStep<=2.5)          dStep= 2.5;
else if(dStep<=5.0)      dStep= 5.0;
else dStep=10.0;

dStep *= pow (10.0,dFull);

iVal     = (int)((dStep*0.995)/dStep);
dVal     = dStep*(double)iVal;
i        = iVal&1;


for(;;dVal+=dStep,i++)
        {
        if(i>32)break;

        iVal = (int)(dVal*dHigh);

        if(iVal>iPixelY)break;

        sprintf(cText,"%i",(int)dVal);

        pImage->Box  (iStartX-1 ,iStartY-iVal,(i&1)? -4:-8,1,0);
        pImage->Print(iStartX-11,iStartY-iVal-FONT_Y/2,cText);

        for(iPos=3;iPos<iFrameX;iPos+=8)
                {
        pImage->Box(iStartX+iPos,iStartY-iVal,1,1,BGR(180,180,180));
                }

        }

pImage->SetAlign(1);

switch(m_iImageMode)
        {
case 0:
        // Monate
                i = iStep;

                for(iPos=0;iPos<=30;iPos+=3)
                        {
                        strtimeset(cText,m_sImportData.iTimeStart+m_sIm-
portData.iTimeSlot*iPos,STR_DATE|STR_GLOBAL|STR_ABC);
```

Programm zur Grafische Auswertung via Konsole

Visual Studio 6.0 TrafficAnalyser.cpp

```
                pImage->Box  (iStartX+iPos*i,iStartY+1,1,(iPos&1)? 4:8,0);
                pImage->Print(iStartX+iPos*i,iStartY+3+FONT_Y,cText);
                        }

                pImage->SetAlign(0);
                        strtimeset(cTemp,m_sImportData.iTimeStart,STR_
DATE|STR_LONG|STR_GLOBAL|STR_ABC|STR_WEEKDAY);
                        sprintf(cText,"Monatsauswertung ab %s",cTemp);
                        pImage->Print(10,10,cText);

                        break;
        case 1:
                // Tage
                        i =  iStep;
                        j =  (m_sImportData.iTimeCount+8)/10;
                        if(j<=0)j=1;

                        for(iPos=0;iPos<iCnt;iPos+=j)
                                {
                                strtimeset(cText,m_sImportData.iTimeStart+m_sIm-
portData.iTimeSlot*iPos,STR_DATE|STR_GLOBAL|STR_ABC);

                        pImage->Box  (iStartX+iPos*i,iStartY+1,1,(iPos&1)? 4:8,0);
                        pImage->Print(iStartX+iPos*i,iStartY+3+FONT_Y,cText);
                                }

                        pImage->SetAlign(0);
                        strtimeset(cFrom,m_sImportData.iTimeStart,STR_
DATE|STR_LONG|STR_GLOBAL|STR_ABC|STR_WEEKDAY);
                        strtimeset(cTemp,m_sImportData.iTimeStop ,STR_
DATE|STR_LONG|STR_GLOBAL|STR_ABC|STR_WEEKDAY);
                        sprintf(cText,"Tagesaufstellung von %s bis
%s",cFrom,cTemp);
                        pImage->Print(10,10,cText);

                        break;
        case 2:
                // Wochentage
                        i = (m_sImportData.iTimeCount/7)*iStep;

                        for(iPos=0;iPos<=7;iPos++)
                                {
                                strtimeset(cText,m_sImportData.iTimeStart+m_sIm-
portData.iTimeSlot*iPos,STR_DATE|STR_GLOBAL|STR_ABC|STR_WEEKDAY);

                        pImage->Box  (iStartX+iPos*i,iStartY+1,1,(iPos&1)? 4:8,0);
                        pImage->Print(iStartX+iPos*i,iStartY+3+FONT_Y,cText);
                                }

                        pImage->SetAlign(0);
                        strtimeset(cFrom,m_sImportData.iTimeStart,STR_
DATE|STR_LONG|STR_GLOBAL|STR_ABC|STR_WEEKDAY);
                        strtimeset(cTemp,m_sImportData.iTimeStop ,STR_
DATE|STR_LONG|STR_GLOBAL|STR_ABC|STR_WEEKDAY);
                        sprintf(cText,"Wochenauswertung von %s bis
%s",cFrom,cTemp);
                        pImage->Print(10,10,cText);

                        break;
        case 3:
                // 7-Tage-Trend
                        i = (m_sImportData.iTimeCount/7)*iStep;

                        for(iPos=0;iPos<=7;iPos++)
                                {
                                strtimeset(cText,m_sImportData.
iTimeStart+3600*24*iPos,STR_DATE|STR_GLOBAL|STR_ABC|STR_WEEKDAY);

                        pImage->Box  (iStartX+iPos*i,iStartY+1,1,(iPos&1)? 4:8,0);
                        pImage->Print(iStartX+iPos*i,iStartY+3+FONT_Y,cText);
                                }

                        pImage->SetAlign(0);
                        strtimeset(cFrom,m_sImportData.iTimeStart,STR_
DATE|STR_LONG|STR_GLOBAL|STR_ABC|STR_WEEKDAY);
                        strtimeset(cTemp,m_sImportData.iTimeStop ,STR_
DATE|STR_LONG|STR_GLOBAL|STR_ABC|STR_WEEKDAY);
```

Programm zur Grafische Auswertung via Konsole

Visual Studio 6.0 — TrafficAnalyser.cpp

```
                        sprintf(cText,“7-Tage-Trend von %s bis %s“,cFrom,cTemp);
                        pImage->Print(10,10,cText);

                        break;
        case 4:
                // Einzelner Tag

                        i = (m_sImportData.iTimeCount/24)*iStep;

                        for(iPos=0;iPos<24;iPos++)
                                {
                                sprintf(cText,“%ih“,iPos);

                        pImage->Box  (iStartX+iPos*i,iStartY+1,1,(iPos&1)? 4:8,0);
                        pImage->Print(iStartX+iPos*i,iStartY+3+FONT_Y,cText);
                                }

                        pImage->SetAlign(0);
                        strtimeset(cFrom,m_sImportData.iTimeStart,STR_
DATE|STR_LONG|STR_GLOBAL|STR_ABC|STR_WEEKDAY);
                        sprintf(cText,“Tagesauswertung vom %s   ( %i Minutenab-
schnitte )“,cFrom,m_sImportData.iTimeSlot/60);
                        pImage->Print(10,10,cText);

                        break;
                }


        if(m_bShowLegend)
        // Legende anzeigen
                {
                sList.DataLength(sizeof(unsigned));

                for(uSlot=0;uSlot<m_sImportData.uFilterCount;uSlot++)
                        {
                        SetFilterText(sFilter,m_sImportData.aDeFilter+uSlot*256);
                        if(m_sImportData.aStName[0]<‘0‘ || m_sImportData.
aStName[0]>‘9‘)
                                        sList.AddFormat(m_sImport-
Data.aLineColor+uSlot,CSC_END,“%s Spur: %s“,m_sImportData.
aStName+uSlot*32,(LPCTSTR)sFilter);
                                else      sList.AddFormat(m_sImportDa-
ta.aLineColor+uSlot,CSC_END,“%s DE: %s“  ,m_sImportData.
aStName+uSlot*32,(LPCTSTR)sFilter);
                                }

                        sList.Sort(0,-1,CmpFilter);

                        for(uSlot=0;uSlot<m_sImportData.uFilterCount;uSlot++)
                                {
pImage->Box  (FRAME_L+10,FRAME_T+0+uSlot*FONT_Y,10,10,RGB(0,0,0));
pImage->Box  (FRAME_L+11,FRAME_T+1+uSlot*FONT_Y, 8, 8,sList.
DataGet(uSlot));
pImage->Print(FRAME_L+24,FRAME_T+3+uSlot*FONT_Y-FONT_Y/3,sList[uSlot]);
                                }
                        }


        // Linien zeichen
        for(uSlot=0;uSlot<m_sImportData.uFilterCount;uSlot++)
                {
                if(m_sImportData.aTimeIgnore[uSlot])
// Stunden von Langzeitdaten auf kleinere Slots aufteilen
                        {
                        i = m_sImportData.aTimeIgnore[uSlot];

                        for(iPos=0;iPos<m_sImportData.iTimeCount;iPos+=i+1)
                                {
                                uVal  = aTlsData[uSlot+iPos*8];
                                uVal /= i+1;

                                aTlsData[uSlot+iPos*8] -= uVal*i;

                                for(j=1;j<=i;j++)
                                        {
                                        aTlsData[uSlot+iPos*8+j*8] = uVal;
                                        }
                                }
```

Programm zur Grafische Auswertung via Konsole

Visual Studio 6.0 TrafficAnalyser.cpp

```
                        m_sImportData.aTimeIgnore[uSlot] = 0;
                        }

                uVal    = aTlsData[uSlot];
                iOldY   = (iFrameY*(int)uVal)/(int)uMax;
                iLast   = 0;


                for(iPos=1;iPos<m_sImportData.iTimeCount;iPos++)
                        {
                        uVal    = aTlsData[uSlot+iPos*8];
                        iValY   = (iFrameY*(int)uVal)/(int)uMax;

                        pImage->LineSmoth(iStartX+iLast*iStep,iStartY+iOldY,iStart
X+iPos*iStep,iStartY+iValY,m_sImportData.aLineColor[uSlot],0);

                        iLast   = iPos;
                        iOldY   = iValY;
                        }
                }

}

//*************************************************************************
//*
//*               main
//*
//*************************************************************************
int main(int iArgc,char **pArgv)
{
static  int             aMonthLen[]={31,28,31,30, 31,30,31,31, 30,31,30,31};
BitmapImage             sImage;
TlsImportData           sData;
struct tm                *pData;
char                    *pName;
char                    *pPath;
time_t                  iTime;
int                             iMode;
int     iEnd;
int     iPos;
int     iOut;

        if(iArgc<=1)
                {
                printf(„SYNTAX:\n\n“);
                printf(„ -M?   = Modus (0=Monat 1=Von-Bis 2=Wochentage
3=Woche 4=Einzeltag\n“);
                printf(„ -P... = Pfad mit ‚*JJJJ-MM-TT.csv‘ Dateien\n“);
                printf(„ -T... = Zeit für Auswertungstsart (JJJJ-MM-TT)\n“);
                printf(„ -E... = Endzeit für Von-Bis-Modus\n“);
                printf(„ ???   = Für jede Kurve muss der Anfangstext der CSV-Datei
angegebn werden.\n\n“);

                return -1;
                }


        memset(&sData,0,sizeof(sData));
        pPath = „.“;
        iMode = 3;
                // 7-Tage-Trent
        iTime = 0;
        iEnd  = 0;
        iOut  = 0;

        sData.aLineColor  [0]   = BGR(255,0,0);
        sData.aLineColor  [1]   = BGR(0,0,255);
        sData.aLineColor  [2]   = BGR(0,255,0);
        sData.aLineColor  [3]   = BGR(195,195,0);
        sData.aLineColor  [4]   = BGR(230,120,0);
        sData.aLineColor  [5]   = BGR(128,128,128);
        sData.aLineColor  [6]   = BGR(128,64,0);
        sData.aLineColor  [7]   = BGR(64,0,128);
        sData.aVehicleBits[0]   = TYP_ALL;
        sData.aVehicleBits[1]   = TYP_ALL;
        sData.aVehicleBits[2]   = TYP_ALL;
        sData.aVehicleBits[3]   = TYP_ALL;
```

Programm zur Grafische Auswertung via Konsole

Visual Studio 6.0 TrafficAnalyser.cpp

```
sData.aVehicleBits[4]    = TYP_ALL;
sData.aVehicleBits[5]    = TYP_ALL;
sData.aVehicleBits[6]    = TYP_ALL;
sData.aVehicleBits[7]    = TYP_ALL;
sData.aSourceType [0]  = 3;              // Aus Einzelfahrzeug daten
sData.aSourceType [1]  = 3;
sData.aSourceType [2]  = 3;
sData.aSourceType [3]  = 3;
sData.aSourceType [4]  = 3;
sData.aSourceType [5]  = 3;
sData.aSourceType [6]  = 3;
sData.aSourceType [7]  = 3;


for(iPos=1;iPos<iArgc;iPos++)            // Parameter auswerten
        {
        pName = pArgv[iPos];

        if(pName[0]!='-')
                {
                if(iOut<8)
                        {
                        memset(sData.aDeFilter+256*iOut,0xFF,256);
                        strncpy(sData.aStName+32*iOut,pName,31);
                        iOut++;
                        }

                continue;
                }

        if(pName[1]=='M' || pName[1]=='m')      // Modus auswählen
        if(pName[2]>='0' && pName[2]<='4')
                {
                iMode = pName[2]-'0';
                continue;
                }

        if(pName[1]=='T' || pName[1]=='t')      // Startzeit auswählen
                {
```

```
        iTime = strtoutc(pName+2,STR_GLOBAL|STR_DATE|STR_NOWEEKDAY);
                        continue;
                        }

        if(pName[1]=='E' || pName[1]=='e')              // Endzeit auswählen
                        {
        iEnd = strtoutc(pName+2,STR_GLOBAL|STR_DATE|STR_NOWEEKDAY);
                        continue;
                        }

                if(pName[1]=='P' || pName[1]=='p')      // Pfad mit Dateien
                        {
                        pPath = pName+2;
                        continue;
                        }
                }

        iEnd  -= iEnd %(3600*24);
        iTime -= iTime%(3600*24);
        pData  = gmtime(&iTime);


        sData.uFilterCount      = iOut;
// Anzahl der Kurven im Bild


        switch(iMode)
                {
        case 0:
                // Monate
                sData.iTimeStart        = iTime-3600*24*(pData->tm_mday-1);
                sData.iTimeSlot         = 3600*24;
                sData.iTimeCount        = aMonthLen[pData->tm_mon%12];
                break;

        case 1:
                // Tage

                        if(iEnd<=iTime)iEnd = iTime+3600*24*7;
```

Programm zur Grafische Auswertung via Konsole

Visual Studio 6.0 TrafficAnalyser.cpp

```
                sData.iTimeStart        = iTime;
                sData.iTimeSlot         = 3600*24;
                sData.iTimeCount        = (iEnd-iTime+3600*24)/(3600*24);

                if(sData.iTimeCount<=0)sData.iTimeCount=1;

                break;
case 2:
        // Wochentage
                sData.iTimeStart        = iTime;
                sData.iTimeSlot         = 3600*24;
                sData.iTimeCount        = 7;
                break;
case 3:
        // 7-Tage-Trend
                sData.iTimeStart        = iTime;
                sData.iTimeSlot         = 3600;
                sData.iTimeCount        = 7*24;
                break;
case 4:
        // EinzIner Tag
                sData.iTimeStart        = iTime;
                sData.iTimeSlot         = 5*60;
                sData.iTimeCount        = 3600*24/sData.iTimeSlot;
                break;
        }

iEnd            = sData.iTimeStart+sData.iTimeSlot*sData.iTimeCount;
sData.iTimeStop= sData.iTimeStart+sData.iTimeSlot*sData.iTimeCount;
sData.iTimeMin = iTime-3600*24*10;
sData.iTimeMax= iEnd +3600*24*30;



printf(„\n\n... Suche Dateien ...\n“);

TlsCsvImport(pPath,“*.csv“,““,&sData);
```

```
        sImage.SetSize(24,800,600);
        m_iImageMode  = iMode;
        m_sImportData = sData;
        DrawImage(&sImage);


        try     {
                // Das Bild abspeichern
                BmpEncoder     sBmp;

                sBmp.WriteImage(„Image.bmp“,sImage);
                }
        catch(...)
                {
                printf(„Das Bild konnte nicht erstellt werden.\n“);
                return -1;
                }

        printf(„Das Bild wurde in Image.bmp gespeichert.\n“);

return 0;
}
```

Programm zur Formatierung und zum Auslesen der Speicherkarte

Visual Studio 6.0 Converter.cpp

```
// Converter.cpp : Definiert den Einsprungpunkt für die Konsolenanwendung.
//

#include „stdafx.h“
#include „../ConfigurationSystem/AzLib/DiskIO.h“

typedef struct  // ACHTUNG: Größe und Reihenfolge der Variablen nicht verändern!
        {
        unsigned char   ucAddr;         // Adresse des Sensors
        unsigned char   ucType;         // Fahrzeugtyp
        unsigned char   ucLength;       // Fahrzeuglänge (10cm Schritte)
        unsigned char   ucVelocity;     // Geschwindigkeit
        unsigned short  usDiffTime;     // zeitl. Abstand zw. den Fahrzeugen (ms)
        unsigned short  usMeasureTime;          // Zeitstempel
        unsigned char   ucDec;                  // 1/100 sec im BCD-Format
        unsigned char   ucSec;                  // Sekunden im BCD-Format
        unsigned char   ucMin;                  // Minuten im BCD-Format
        unsigned char   ucHour;                 // Stunden im BCD-Format
        unsigned char   ucYear;                 // Jahr im BCD-Format
        unsigned char   ucMonth;                // Monat im BCD-Format
        unsigned char   ucDay;                  // Tag im BCD-Format
        unsigned char   ucFree;                 // unbenutzt
        }TrafficData;



int main(int argc, char* argv[])
{
        if (argc >= 2 && stricmp(argv[1],“list“)==0)
                {
                QWORD size;
                int number;

                for(number=0;number<16;number++)
                        {
                // Größe: Physikalische keine Logischen Laufwerke ermitteln:
                        size = GetSectorCount(number|0x8000);
                        if(size == 0){continue;}
                        printf(„Disk %i = %u Sektoren\n“,number, size);
                        }
                return(1);
                }

        if (argc >= 2 && stricmp(argv[1],“format“)==0)
                {
                QWORD           size;
                BYTE    sector[512];
                int             i;
                int             number;

                if(argc == 2)
                        {
                        printf(„Bitte Disknummer angeben:\n“);
                        return(69);
                        }

                number = atoi(argv[2]); // Text in Zahl

                size = GetSectorCount(number|0x8000);
                if(size == 0)
                        {
                        printf(„Disk nicht vorhanden!!!\n“);
                        return(69);
                        }

                if(size>=2000000)
                        {
                        printf(„Bitte nicht die eigene Festplatte formatieren!!!\n“);
                        return(69);
                        }


                memset(sector,0x0,512);
                for(i=0; i<0x8000; i++)  // 32768 Sektoren => 16 MB
                        {
                        WriteSector(number|0x8000, i, 1, sector);
                        }
```

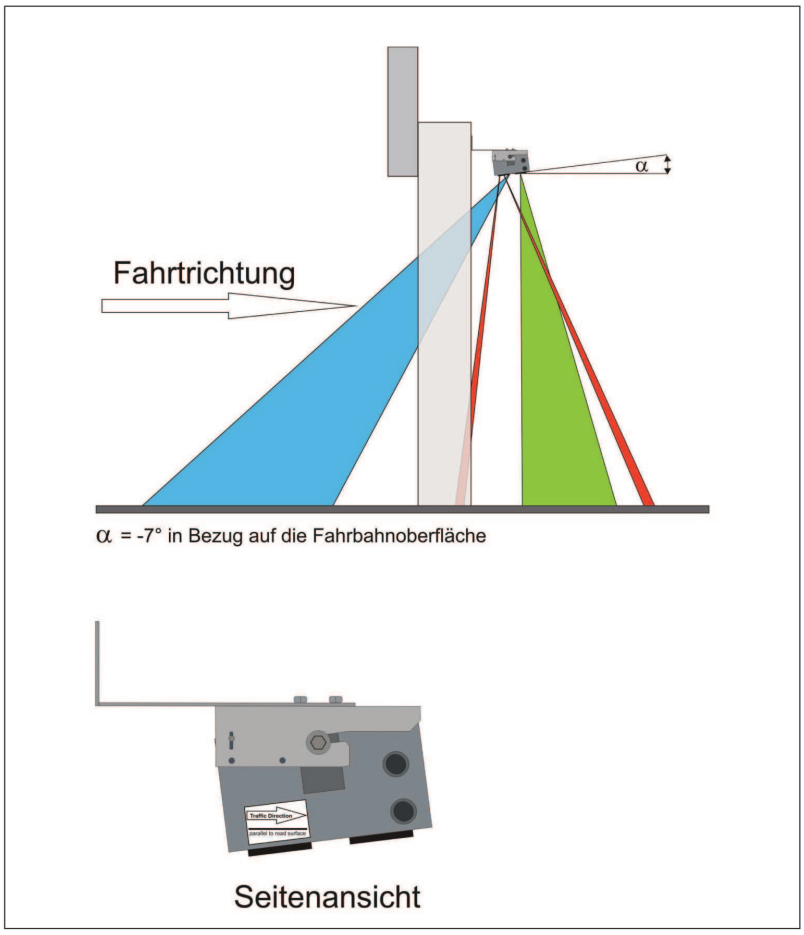
Programm zur Formatierung und zum Auslesen der Speicherkarte

Visual Studio 6.0 Converter.cpp

```
        return(1);
        }

if (argc >= 2 && stricmp(argv[1],“readout“)==0)
        {
        QWORD       size;
        char*       kfztypes[9] =   { „nkKFZ“,
                                      “Motorrad“,
                                      “PKW“,
                                      “Lieferwagen“,
                                      “PKW+Anhänger“,
                                      “LKW“,
                                      “LKW+Anhänger“,
                                      „Sattelschlepper“,
                                      “Bus“};

        TrafficData ExTConv[512/16];
        char        *name;
        FILE        *file;
        int         i,ok;
        int         number;

        if(argc == 2)
                {
                printf(„Bitte Disknummer angeben:\n“);
                return(69);
                }

        if(argc == 3)
                {
                printf(„Bitte Name der .csv-Datei angeben!\n“);
                return(69);
                }

        number = atoi(argv[2]); // Text in Zahl
        name = argv[3];
        size = GetSectorCount(number|0x8000);
        if(size == 0)
        {
        printf(„Disk nicht vorhanden!!!\n“);
        return(69);
        }
        if(size>=2000000)
        {
        printf(„Festplatten können nicht ausgelesen werden !!!\n“);
        return(69);
        }

file = fopen(name, „wb“);
if (!file){printf(„Kann Datei nicht öffnen!!!\n“); return(69);}

fprintf(file,“Zeit;Fahrzeugtyp;Geschwindigkeit;Fahrzeuglaenge in  dm;
Nettozeitluecke;\n“);

for(i = 0; i<0x8000; i++)
        {
        ok = ReadSector(number|0x8000, i, 1, ExTConv);
        if(ok == 0)
        {printf(„Konnte nicht von der Karte lesen!!!\n“);break;}
        if(ExTConv[0].ucAddr==0){break;}

        for(int j=0;j<32;j++)
                {
                fprintf(file,“20%02x-%02x-%02x  %02x:%02x:%02x;“,
                ExTConv[j].ucYear, ExTConv[j].ucMonth, ExTConv[j].ucDay,
                ExTConv[j].ucHour, ExTConv[j].ucMin, ExTConv[j].ucSec);
                fprintf(file,“%s;“,kfztypes[ExTConv[j].ucType%9]);
                fprintf(file,“%u;%u;%u;\n“,ExTConv[j].ucVelocity,
                ExTConv[j].ucLength, ExTConv[j].usDiffTime);
                }
        }
                fclose(file);
                return(1);
                }
        return 0;
}
```

B Montagearten, Zubehör und Anschlüsse des TDC

Frontfire mit Montage hinter der Schilderbrücke



Erhältliche Modelle für die obenstehende Montage und Ausrichtung

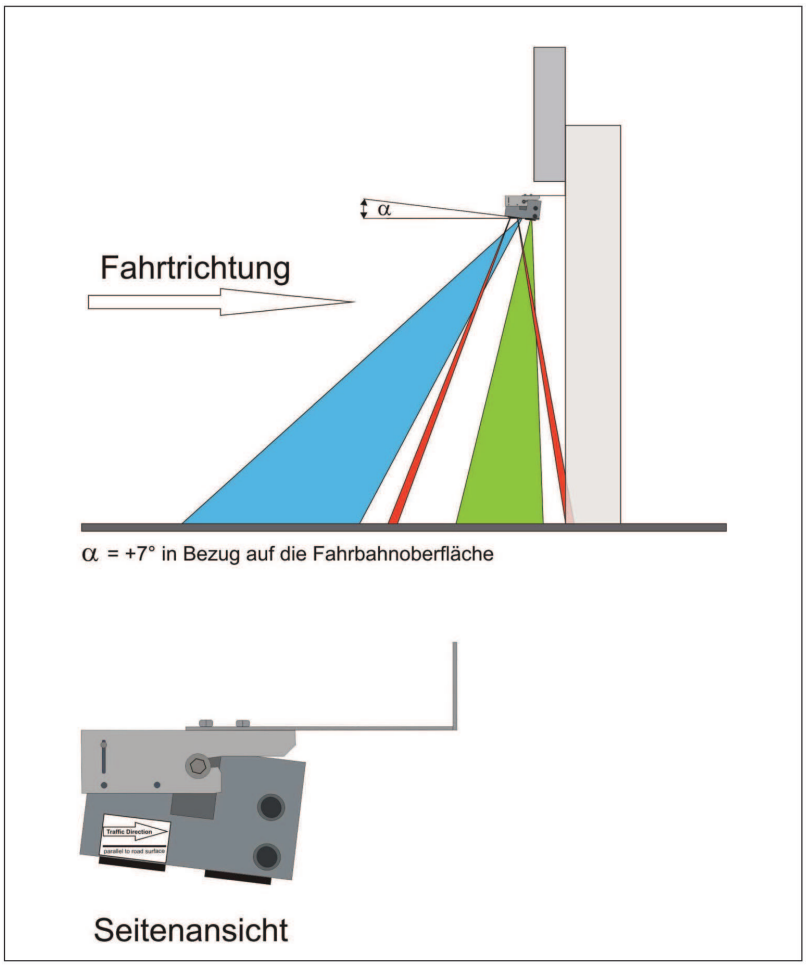
Modell	Bestellnummer	Beschreibung
TDC3-**2**-F-B-45	11110	**2** Klassen, Frontfire, Montage hinter SB
TDC3-**3**-F-B-45	11113	**2+1** Klassen, Frontfire, Montage hinter SB
TDC3-**5**-F-B-45	11115	**5+1** Klassen, Frontfire, Montage hinter SB
TDC3-**8**-F-B-45	11117	**8+1** Klassen, Frontfire, Montage hinter SB

Anmerkung:

Der **Ultraschall-Kegel** (grün) **muss immer von der Montagestruktur (Schilderbrücke) abgewandt sein**, um die maximale Genauigkeit und Zuverlässigkeit zu gewährleisten und um unerwünschte, potentiell störende Reflexionen (Echos) zu vermeiden.

Der Pfeil ersichtlich in der Seitenansicht gibt die Fahrtrichtung an. Im obenstehenden Beispiel schaut der **Radar** (blau) **dem Verkehrsfluss entgegen = Frontfire.**
Für Informationen zur empfohlenen Montagehöhe für das jeweilige Detektormodell siehe Abschnitt 3.1.

Frontfire mit Montage vor der Schilderbrücke



Erhältliche Modelle für die obenstehende Montage und Ausrichtung

Modell	Bestellnummer	Beschreibung
TDC3-**2**-F-F-31	11111	**2** Klassen, Frontfire, Montage vor SB
TDC3-**3**-F-F-31	11114	**2+1** Klassen, Frontfire, Montage vor SB
TDC3-**5**-F-F-31	11116	**5+1** Klassen, Frontfire, Montage vor SB
TDC3-**8**-F-F-31	11118	**8+1** Klassen, Frontfire, Montage vor SB

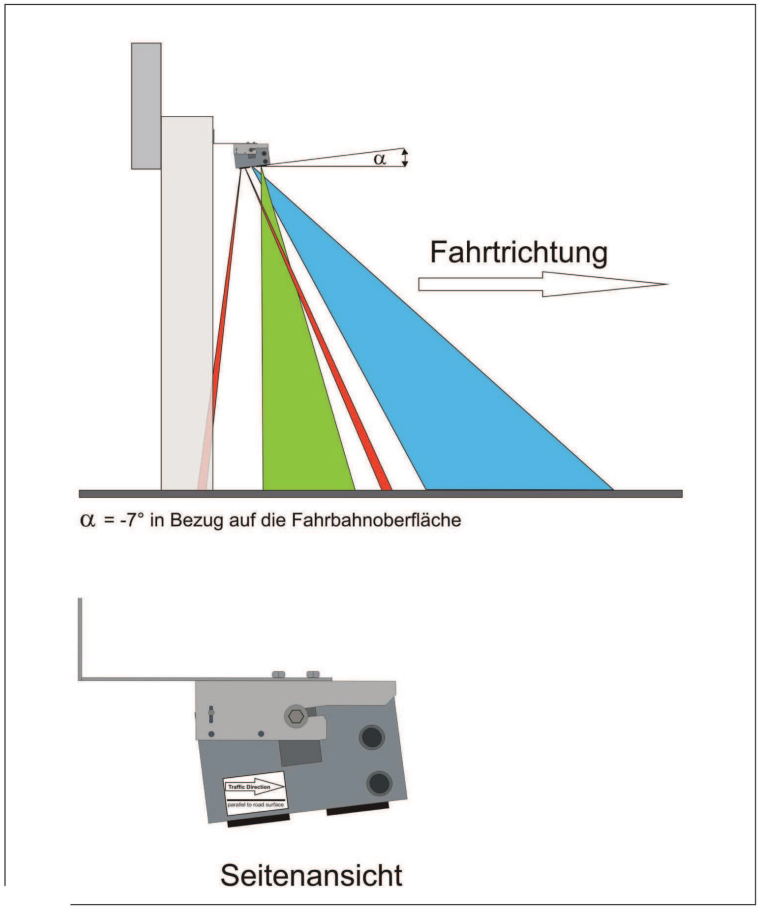
Anmerkung:

Der **Ultraschall-Kegel** (grün) **muss immer von der Montagestruktur (Schilderbrücke) abgewandt sein**, um die maximale Genauigkeit und Zuverlässigkeit zu gewährleisten und um unerwünschte, potentiell störende Reflexionen (Echos) zu vermeiden.

Der Pfeil ersichtlich in der Seitenansicht gibt die Fahrtrichtung an. Im obenstehenden Beispiel schaut der **Radar** (blau) **dem Verkehrsfluss entgegen = Frontfire.**

Für Informationen zur empfohlenen Montagehöhe für das jeweilige Detektormodell siehe Abschnitt 3.1.

Backfire mit Montage hinter der Schilderbrücke



Erhältliche Modelle für die obenstehende Montage und Ausrichtung

Modell	Bestellnummer	Beschreibung
TDC3-**2**-B-B-31	11112	**2** Klassen, Backfire, Montage hinter SB

Anmerkung:

Der **Ultraschall-Kegel** (grün) **muss immer von der Montagestruktur (Schilderbrücke) abgewandt sein**, um die bestmögliche Genauigkeit und Zuverlässigkeit zu gewährleisten und um unerwünschte, potentiell störende Reflexionen (Echos) zu vermeiden.

Der Pfeil ersichtlich in der Seitenansicht gibt die Fahrtrichtung an. Im obenstehenden Beispiel schaut der **Radar** (blau) **dem Verkehrsfluss hinterher = Backfire.**

Für Informationen zur empfohlenen Montagehöhe für das jeweilige Detektormodell siehe Abschnitt 3.1.

Zubehör

Anschlussstecker und -Buchse

Kabelstecker rechtwinklig

Bestellnummer: 64012

Kabeldose rechtwinklig

Bestellnummer: 64013

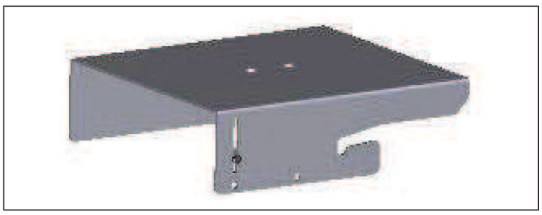
TDC3-Abschlusswiderstand

Bestellnummer: 64014

Montagezubehör

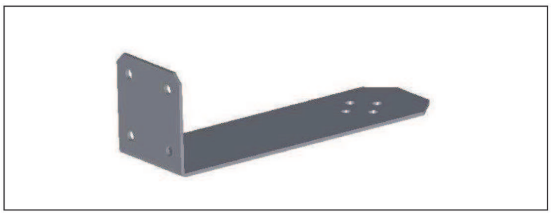
Halterung TDC-MB

Bestellnummer: 14010



Montageadapter TDC-MA

Bestellnummer: 14011



Interface Modul RS 485/USB

USB-IF485

Bestellnummer: 12501

Anhang zur Verdrahtung

Elektrische Anschlüsse der Detektoren der Serie TDC3

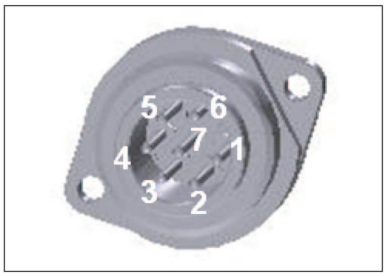
Die Detektoren der Serie TDC3 sind mit einem Gerätestecker und einer Gerätedose, gemäss untenstehenden Bilder, ausgestattet. Dazu passende Kabelbuchse und / oder -Stecker sind im Lieferumfang nicht enthalten, sind jedoch als Original-Zubehör erhältlich. Für Einzelheiten siehe Kapitel 13.

Gerätedose

Pin-Definition für Gerätedose und -Stecker

1 Speisung+, Vcc
2 Option Trigger-Ausgang (Spezialversion, erhältlich auf Anfrage)
3 GND
4 Synchronisation
5 RS 485 Signal A
6 RS 485 Signal B
7 Nicht verwenden!

Gerätestecker



Empfehlungen zum Kabel

- Polyurethan-Kabel (PUR) mit Schirm und paarweise verdrillt
- Aderquerschnitt: 0.34 … 0.52 mm2 (AWG 22 oder AWG 20) *)
- Kabeldurchlass der Kabelbuchse/-Stecker PG 9: Durchmesser 6 … 12.5 mm (0.24 … 0.47 inches)
- Maximaler Querschnitt für Kabelbuchse/-Stecker: 0.75 mm^2
- Speisung (11 V DC … 24 V DC nominal): 2 Adern
- RS 485 Kommunikation: 2 Adern, verdrillt
- Synchronisation**: 1 Ader
- Abschirmung: Die Zuleitung vom Steuergerät zum Detektor ist auf der Seite des Steuergerätes zu erden. Eine Abschirmung der Kabel zwischen den Detektoren ist aufgrund der kurzen Kabellängen nicht notwendig.

- *) Der Querschnitt ist so zu wählen, dass bei einem Strom von 100 mA pro Detektor, die Spannung am Anschluss-Stecker des letzten Detektors, min. 11.0 V beträgt und die des ersten Detektors, die zulässige Maximalspannung nicht übersteigt (30 V DC).

- **) Erforderlich, wenn Detektoren über benachbarten Fahrspuren näher als 8 m zueinander montiert sind (zur Vermeidung von gegenseitiger akustischer Beeinflussung)

Für weitere Informationen zur Verdrahtung kontaktieren Sie bitte den Hersteller.

C Messdaten

Zeit	Fahrzeugtyp	Geschwindigkeit	Fahrzeuglänge in dm	Nettozeitlücke
15.02.2012 14:08	PKW	65	38	65520
15.02.2012 14:09	PKW	67	43	4436
15.02.2012 14:09	PKW	72	48	880
15.02.2012 14:10	PKW	78	44	29104
15.02.2012 14:10	PKW	77	43	1052
15.02.2012 14:11	PKW	65	49	2136
15.02.2012 14:11	PKW+Anhänger	67	72	1140
15.02.2012 14:11	PKW	70	41	1340
15.02.2012 14:12	PKW	87	46	2236
15.02.2012 14:12	PKW	90	46	2824
15.02.2012 14:12	PKW	65	39	2136
15.02.2012 14:14	PKW	66	43	3692
15.02.2012 14:15	PKW	69	46	3020
15.02.2012 14:16	PKW	68	47	1524
15.02.2012 14:16	Lieferwagen	78	51	2076
15.02.2012 14:15	PKW	70	46	1020
15.02.2012 14:16	PKW	90	43	520
15.02.2012 14:16	PKW	92	39	920
15.02.2012 14:17	PKW	76	41	5840
15.02.2012 14:17	PKW+Anhänger	67	58	936
15.02.2012 14:17	PKW	73	44	1680
15.02.2012 14:17	PKW	81	42	340
15.02.2012 14:17	PKW	65	44	656
15.02.2012 14:17	PKW	69	42	636
15.02.2012 14:18	PKW	89	39	636
15.02.2012 14:20	PKW	80	44	12676
15.02.2012 14:20	PKW	72	40	1924
15.02.2012 14:20	PKW	64	41	1168
15.02.2012 14:21	PKW	69	45	2076
15.02.2012 14:21	PKW	66	42	4620
15.02.2012 14:21	PKW	70	45	768
15.02.2012 14:22	PKW	75	49	576
15.02.2012 14:22	PKW	76	38	668
15.02.2012 14:24	PKW	67	42	12272
15.02.2012 14:24	PKW	88	40	1532
15.02.2012 14:24	PKW	89	45	1472
15.02.2012 14:25	Lieferwagen	69	62	1764
15.02.2012 14:27	PKW	64	40	14092
15.02.2012 14:27	PKW	66	41	1464
15.02.2012 14:27	PKW	71	43	1856
15.02.2012 14:28	PKW	71	39	4996
15.02.2012 14:30	PKW	68	45	7628
15.02.2012 14:36	PKW	100	41	38540
15.02.2012 14:36	PKW	65	48	1356
15.02.2012 14:36	PKW	79	49	656
15.02.2012 14:36	PKW	66	42	532
15.02.2012 14:37	PKW	64	46	2524
15.02.2012 14:37	PKW	76	45	1756
15.02.2012 14:39	PKW	88	46	14632
15.02.2012 14:40	PKW	67	44	5040
15.02.2012 14:43	PKW	68	43	13668
15.02.2012 14:43	PKW	66	45	528
15.02.2012 14:43	PKW	70	44	2556
15.02.2012 14:44	Lieferwagen	79	58	5428
15.02.2012 14:45	PKW	65	41	6224

Abbildung C.1: Teil 1 der Messdaten für die Verkehrsmessung an der L360 zwischen Sinabelkirchen und Egelsdorf.

15.02.2012 14:50	PKW	69	43	31540
15.02.2012 14:52	PKW	67	43	7624
15.02.2012 14:55	PKW	68	42	21656
15.02.2012 14:56	PKW	68	48	7776
15.02.2012 14:57	Lieferwagen	72	39	1856
15.02.2012 14:57	PKW	77	45	3540
15.02.2012 14:57	PKW	69	45	264
15.02.2012 14:58	PKW	67	39	4248
15.02.2012 15:03	PKW	71	38	30332
15.02.2012 15:03	PKW	76	48	412
15.02.2012 15:04	Lieferwagen	89	50	3012
15.02.2012 15:05	PKW	103	44	6596
15.02.2012 15:06	PKW	94	48	8348
15.02.2012 15:06	PKW	70	50	548
15.02.2012 15:06	PKW	75	45	548
15.02.2012 15:07	PKW	69	44	2032
15.02.2012 15:07	LKW	78	51	1148
15.02.2012 15:07	LKW	80	51	580
15.02.2012 15:07	PKW	67	44	924
15.02.2012 15:10	PKW	69	41	13832
15.02.2012 15:10	PKW	65	37	424
15.02.2012 15:11	PKW	71	42	6060
15.02.2012 15:11	PKW	65	47	1648
15.02.2012 15:11	Lieferwagen	75	52	2136
15.02.2012 15:12	PKW	64	46	6048
15.02.2012 15:14	PKW	74	39	9612
15.02.2012 15:14	PKW	70	45	1000
15.02.2012 15:14	PKW	78	41	936
15.02.2012 15:17	PKW	82	44	13240
15.02.2012 15:17	PKW	67	42	528
15.02.2012 15:17	PKW	84	40	3316
15.02.2012 15:17	PKW	70	39	1184
15.02.2012 15:19	PKW	72	44	8860
15.02.2012 15:19	PKW	81	49	1872
15.02.2012 15:20	PKW	69	39	2400
15.02.2012 15:20	PKW	73	44	1360
15.02.2012 15:20	PKW	69	40	1968
15.02.2012 15:21	PKW	70	42	2160
15.02.2012 15:21	Lieferwagen	69	47	2960
15.02.2012 15:21	PKW	67	43	1856
15.02.2012 15:24	PKW	80	39	16240

Abbildung C.2: Teil 2 der Messdaten für die Verkehrsmessung an der L360 zwischen Sinabelkirchen und Egelsdorf.

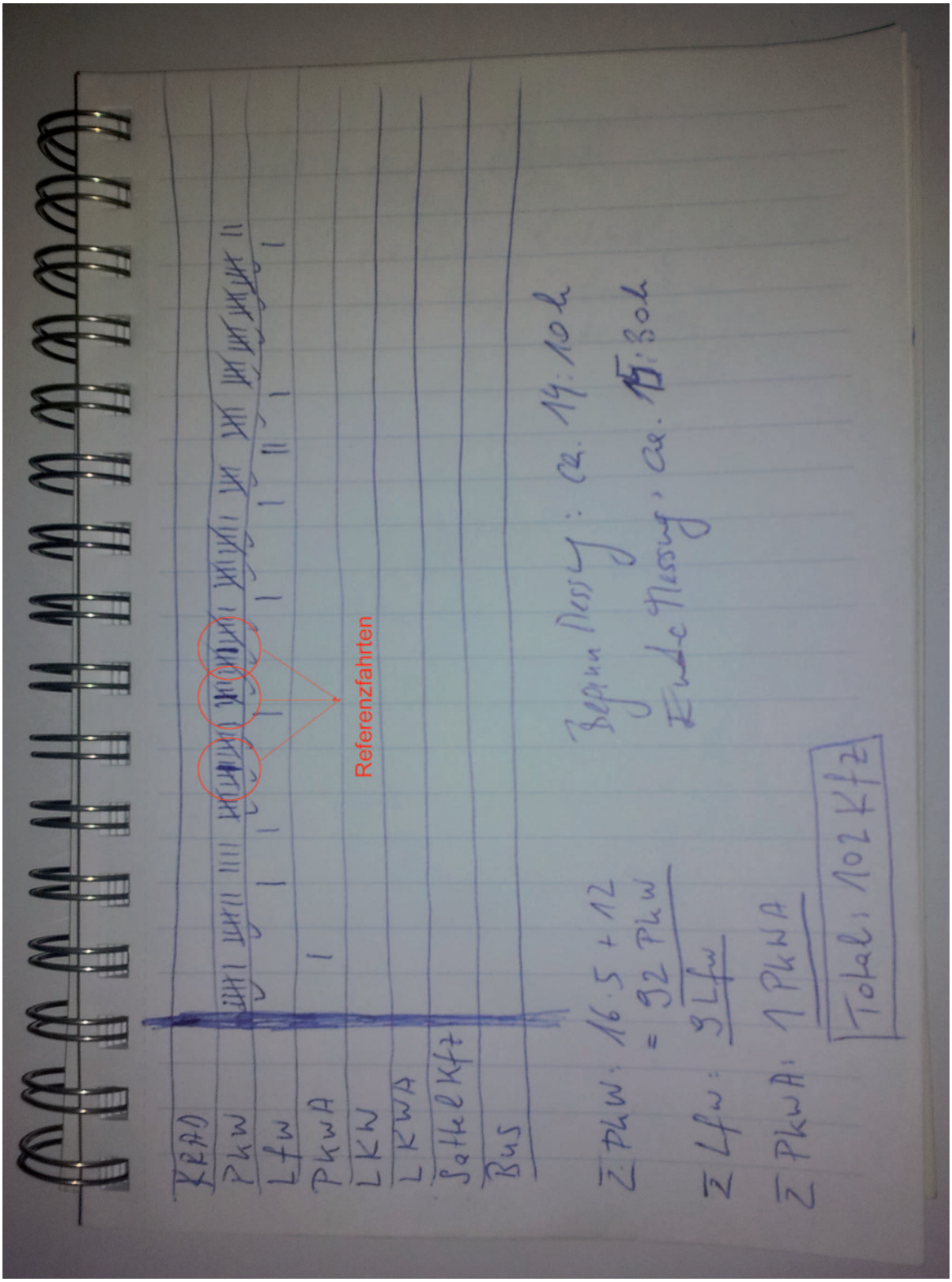


Abbildung C.3: Geführte Strichliste während der Messung an der L360.

Fahrspur 1: Beispieldaten im .csv-Format (Gesamtlänge: 158 Seiten)

Zeit	Fahrzeugtyp	Geschwindigkeit	Fahrzeuglänge in dm	Nettozeitlücke
01.08.2011 00:00	Lieferwagen	118	67	5920
01.08.2011 00:00	Lieferwagen	107	50	112
01.08.2011 00:01	PKW	130	49	2640
01.08.2011 00:01	PKW	124	45	240
01.08.2011 00:01	PKW	136	45	1104
01.08.2011 00:01	PKW	118	44	1264
01.08.2011 00:01	PKW+Anhänger	95	98	1088
01.08.2011 00:02	Bus	102	138	2736
01.08.2011 00:03	PKW	128	41	65520
01.08.2011 00:04	PKW	123	40	4400
01.08.2011 00:04	PKW	121	38	160
01.08.2011 00:04	PKW	140	38	2112
01.08.2011 00:04	PKW	99	41	208
01.08.2011 00:04	PKW+Anhänger	89	134	768
01.08.2011 00:05	PKW	115	43	1584
01.08.2011 00:05	PKW	89	43	800
01.08.2011 00:05	PKW	104	45	1232
01.08.2011 00:05	PKW	104	43	80
01.08.2011 00:06	PKW	102	42	4192
01.08.2011 00:06	Bus	99	120	1936
01.08.2011 00:06	PKW	107	43	1232
01.08.2011 00:07	PKW	103	39	6144
01.08.2011 00:08	PKW	111	41	1952
01.08.2011 00:08	PKW	88	47	416
01.08.2011 00:08	PKW	109	41	224
01.08.2011 00:08	PKW	124	43	1408
01.08.2011 00:09	PKW	130	43	6016
01.08.2011 00:09	PKW	99	42	608
01.08.2011 00:09	Bus	102	120	1136
01.08.2011 00:10	PKW	104	37	6688
01.08.2011 00:11	Lieferwagen	109	51	7072
01.08.2011 00:12	PKW	136	41	5488
01.08.2011 00:13	PKW	140	48	928
01.08.2011 00:13	Lieferwagen	132	52	1840
01.08.2011 00:14	PKW	101	39	3968
01.08.2011 00:14	PKW	101	43	352
01.08.2011 00:14	PKW	90	43	992
01.08.2011 00:14	PKW	89	39	80
01.08.2011 00:14	PKW	88	35	176
01.08.2011 00:14	PKW	98	43	256
01.08.2011 00:17	Lieferwagen	107	49	16128
01.08.2011 00:17	PKW	121	43	1824
01.08.2011 00:17	PKW	123	47	192
01.08.2011 00:17	PKW	116	44	208
01.08.2011 00:17	PKW	124	44	320
01.08.2011 00:17	PKW	130	44	1616
01.08.2011 00:17	PKW	118	44	448
01.08.2011 00:18	PKW	124	48	2816
01.08.2011 00:18	PKW	99	44	704
01.08.2011 00:18	PKW	108	39	1280
01.08.2011 00:19	PKW	113	42	3376
01.08.2011 00:19	PKW	111	40	96
01.08.2011 00:19	PKW	112	44	288

Abbildung C.4: Messdaten für den in Kapitel 5.6 auf Seite 53 dargestellten Graphen.

Fahrspur 2: Beispieldaten im .csv-Format (Gesamtlänge: 158 Seiten)

Zeit	Fahrzeugtyp	Geschwindigkeit	Fahrzeuglänge in dm	Nettozeitlücke
01.08.2011 00:00	PKW	111	40	1072
01.08.2011 00:03	Lieferwagen	157	51	65520
01.08.2011 00:09	PKW	124	43	65520
01.08.2011 00:14	PKW+Anhänger	104	113	65520
01.08.2011 00:19	PKW	128	47	30976
01.08.2011 00:26	PKW	145	41	65520
01.08.2011 00:34	Lieferwagen	128	54	65520
01.08.2011 00:35	PKW	121	43	3600
01.08.2011 00:35	PKW	140	44	800
01.08.2011 00:41	PKW	160	39	32128
01.08.2011 00:51	PKW	115	45	65520
01.08.2011 00:52	PKW	112	37	5760
01.08.2011 01:00	PKW	118	42	65520
01.08.2011 01:05	PKW	124	42	29888
01.08.2011 01:06	PKW	128	43	7776
01.08.2011 01:07	PKW	142	48	5280
01.08.2011 01:08	PKW	118	43	7344
01.08.2011 01:10	Bus	102	119	65520
01.08.2011 01:11	PKW	124	43	65520
01.08.2011 01:19	Lieferwagen	104	55	48656
01.08.2011 01:20	PKW	128	44	7616
01.08.2011 01:22	PKW	123	44	11744
01.08.2011 01:29	PKW	112	45	37216
01.08.2011 01:29	PKW	112	41	4688
01.08.2011 01:30	Bus	99	122	65520
01.08.2011 01:38	Lieferwagen	116	56	65520
01.08.2011 01:39	PKW	140	37	4128
01.08.2011 01:45	PKW	132	42	65520
01.08.2011 02:12	PKW	112	45	65520
01.08.2011 02:26	PKW	128	45	65520
01.08.2011 02:26	PKW	105	44	2256
01.08.2011 02:33	PKW	130	37	65520
01.08.2011 02:45	PKW	118	40	65520
01.08.2011 02:47	PKW	90	37	11936
01.08.2011 02:47	PKW	101	48	624
01.08.2011 02:49	PKW	132	43	9184
01.08.2011 02:50	Bus	98	123	9616
01.08.2011 02:50	PKW	97	43	704
01.08.2011 03:01	PKW	128	41	65520
01.08.2011 03:11	PKW	105	41	65520
01.08.2011 03:18	PKW	108	39	65520
01.08.2011 03:31	Lieferwagen	121	66	65520
01.08.2011 03:35	Sattelschlepper	87	131	20784
01.08.2011 03:35	PKW	89	40	48
01.08.2011 03:35	Lieferwagen	91	58	64
01.08.2011 03:36	Lieferwagen	102	57	7152
01.08.2011 03:38	Lieferwagen	91	58	65520
01.08.2011 03:39	PKW	99	39	2640
01.08.2011 03:42	PKW	118	39	20400
01.08.2011 03:45	PKW	105	44	21104
01.08.2011 03:47	PKW	145	46	11552
01.08.2011 03:48	PKW	109	38	3360
01.08.2011 03:57	PKW	123	43	65520

Abbildung C.5: Messdaten für den in Kapitel 5.6 auf Seite 53 dargestellten Graphen.

D Platinenlayout

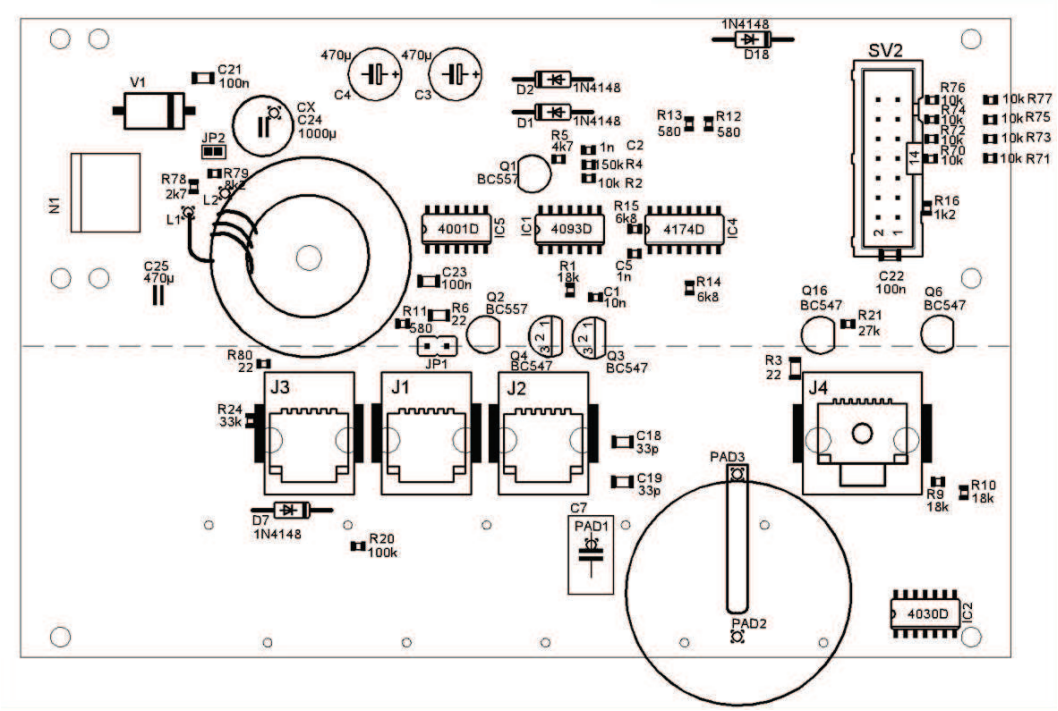


Abbildung D.1: Bestückung: Steuerungsplatine Oberseite

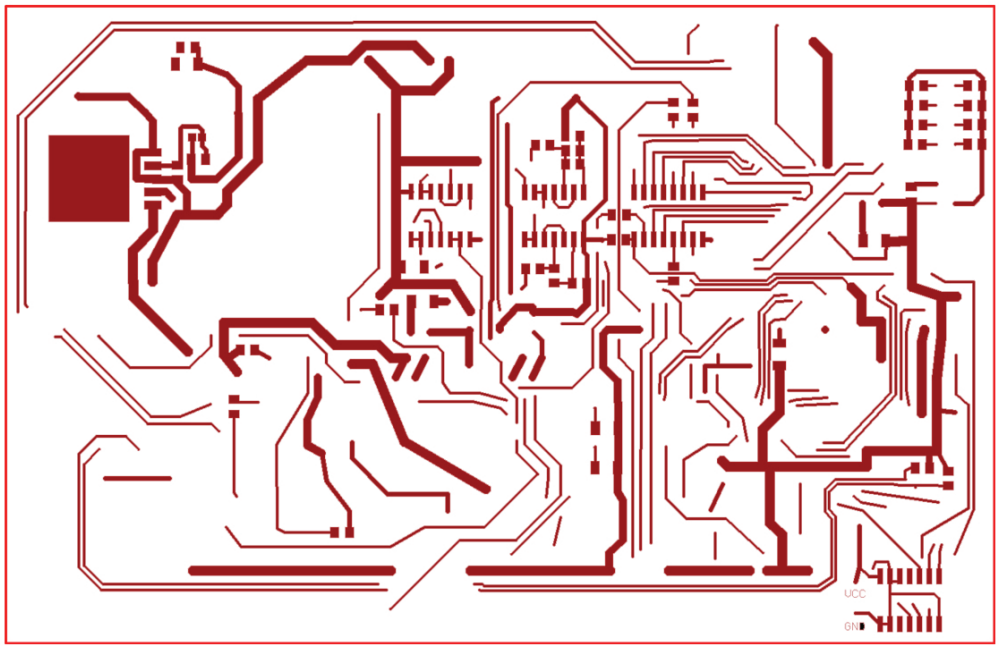


Abbildung D.2: Leiterbahnen: Steuerungsplatine Oberseite

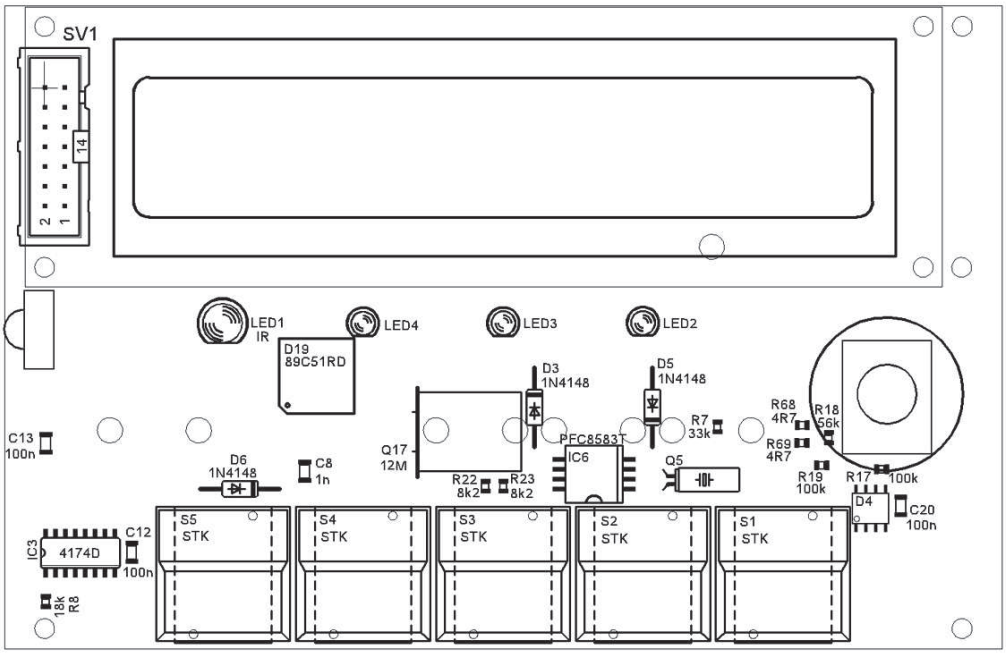


Abbildung D.3: Bestückung: Steuerungsplatine Unterseite

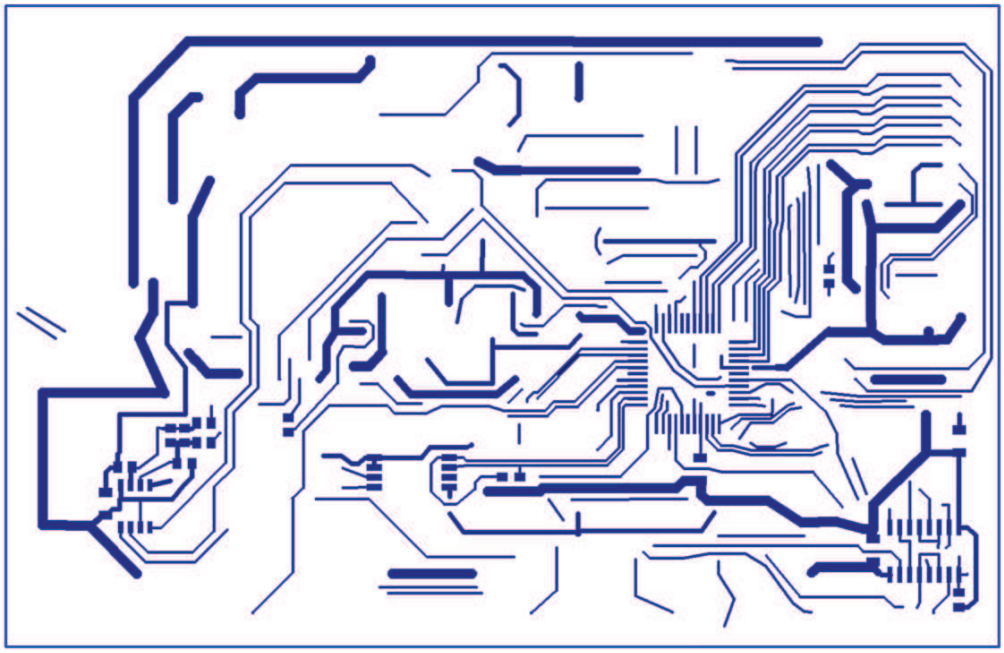


Abbildung D.4: Leiterbahnen: Steuerungsplatine Unterseite

E Werkzeuge

Sämtliche Informationen zur Beschreibung der Werkzeuge wurden von derer Internetpräsenz und Wikipedia entnommen.

E.1 Grafik

E.1.1 Adobe Photoshop CS3 Extended

Adobe Photoshop ist ein kommerzielles Bildbearbeitungsprogramm des US-amerikanischen Softwarehauses Adobe Systems. Im Bereich der Druckvorstufe ist das Programm Marktführer und marktbeherrschend.

Photoshop ist Teil der Adobe Creative Suite, einer Sammlung von Grafik- und Designprogrammen.

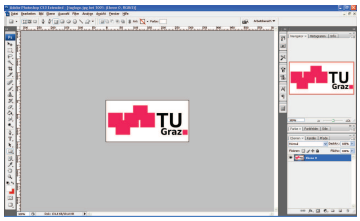


Abbildung E.1: Adobe Photoshop

E.1.2 OpenOffice.org

OpenOffice.org ist ein freies Office-Paket, das aus einer Kombination verschiedener Programme zur Textverarbeitung, Tabellenkalkulation, Präsentation und zum Zeichnen besteht. Ein Datenbankprogramm und ein Formeleditor sind ebenfalls enthalten.

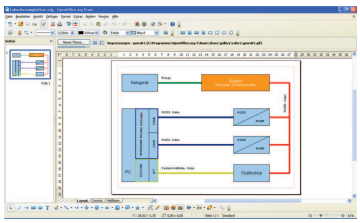


Abbildung E.2: OpenOffice

Das quelloffene Projekt ist eines der international führenden Officepakete und für alle wichtigen Betriebssysteme verfügbar. Der Zugang zu Funktionen und Daten wird durch offengelegte Schnittstellen und ein XML-basiertes Dateiformat ermöglicht. OpenOffice.org wird unter der LGPL herausgegeben.

E.2 Elektronischer Entwurf und Simulation mit EAGLE

EAGLE ist ein EDA-Programm (Electronic Design Automation) der Firma CadSoft Computer GmbH zur Erstellung von Leiterplatten.

Der Name ist ein Initialwort, gebildet aus **E**infach **A**nzuwendender **G**rafischer **L**ayout-**E**ditor.

Die Software besteht aus mehreren Komponenten: Layout-Editor, Schaltplan-Editor, Autorouter und einer erweiterbaren Bauteil-Datenbank.

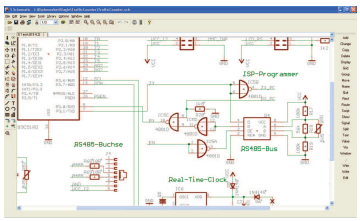


Abbildung E.3: EAGLE

E.3 Programmierung

E.3.1 Keil μVision

Visual Studio ist eine von dem Unternehmen Microsoft angebotene, integrierte Entwicklungsumgebung für verschiedene Hochsprachen.

Das 1998 veröffentlichte Visual Studio 6.0 ist eine Sammlung der Entwicklungsumgebungen Visual Basic 6.0, Visual C++ 6.0, Visual InterDev (ASP Webapplikationsentwicklung), Visual FoxPro und Visual J++.

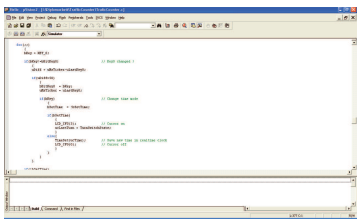


Abbildung E.4: Keil μVision

Neben der Programmierumgebung selbst sind noch diverse Tools zur Unterstützung des Entwicklers bzw. zur Erweiterung von Team-Arbeits-Funktionen integriert.

E.3.2 Visual Studio 6.0

Visual Studio ist eine von dem Unternehmen Microsoft angebotene, integrierte Entwicklungsumgebung für verschiedene Hochsprachen.

Das 1998 veröffentlichte Visual Studio 6.0 ist eine Sammlung der Entwicklungsumgebungen Visual Basic 6.0, Visual C++ 6.0, Visual InterDev (ASP Webapplikationsentwicklung), Visual FoxPro und Visual J++.

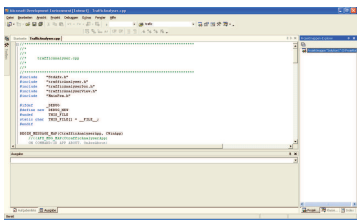


Abbildung E.5: MS Visual Studio

Neben der Programmierumgebung selbst sind noch diverse Tools zur Unterstützung des Entwicklers bzw. zur Erweiterung von Team-Arbeits-Funktionen integriert.

E.4 Schriftsatz mit TeX Live 2011

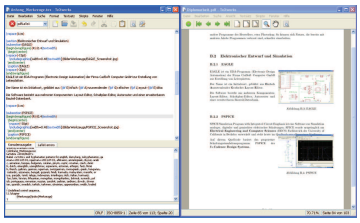


Abbildung E.6: TEX Live 2011

TeX ist ein von Donald E. Knuth ab 1977 entwickeltes und 1986 fertiggestelltes Textsatzsystem mit eingebauter Makrosprache, die ebenfalls als TeX bezeichnet wird.

Häufig wird das TeX erweiternde Softwarepaket LaTeX fälschlich ebenfalls TeX genannt.

TeX kann für alle Arten von Texten verwendet werden, vom kurzen Brief bis zu mehrbändigen Büchern, wobei TeX ursprünglich für längere Texte und wissenschaftliche Arbeiten entwickelt wurde.

Viele große wissenschaftliche Verlage nutzen es für den Buchdruck bzw. Werksatz. Eine besondere Stärke ist der mathematische Formelsatz sowie das Schriftbild.