+++ neuer name +++
hochschule für angewandte
wissenschaften hamburg

university of applied sciences
gegr. 1970 fachhochschule hamburg
FACHBEREICH ELEKTROTECHNIK
UND INFORMATIK

Diplomarbeit

Manfred Adler

Entwurf und Realisierung eines seriellen Anschlusses für Compact Flash Speicherkarten

Studiengang Technische Informatik
Betreuender Prüfer: Prof. Dr. Kai von Luck
Zweitgutachter: Prof. Dr. Gunter Klemke
Abgegeben am 3. Juni 2002

BERLINER TOR 3 · D - 20099 HAMBURG

Entwurf und Realisierung eines seriellen Anschlusses für Compact Flash Speicherkarten

Stichworte

Mobiler Roboter, Compact Flash, Speicherkarte, RS-232

Zusammenfassung

Diese Diplomarbeit beschäftigt sich mit der Möglichkeit, persistente Speichermedien möglichst einfach an mobile autonome Roboter anzuschließen.

Im Rahmen der Diplomarbeit wurde zu diesem Zweck ein Hardware-/Software-Interface entworfen, gebaut, programmiert und getestet. Es erlaubt die einfache Anbindung von Compact Flash Speicherkarten an unterschiedliche Typen von Robotern des Roboter-Labors der Hochschule für Angewandte Wissenschaften Hamburg.

Design and realization of a serial interface for compact flash memorycards.

Keywords

mobile robot, compact flash , memorycard, rs-232

Abstract

This diploma thesis deals with the possibility of connecting persistent storage media to mobile autonomous robots in a simple way.

In context of this thesis a hardware/software interface was designed, built, programmed and tested. It permits simple connection of Compact Flash memory cards to different types of robots located at the robot lab of the university of applied sciences Hamburg.

Danksagung

Ich möchte gerne Carola für Ihre unendliche Geduld und den guten Kaffee danken.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Die rasante Entwicklung der Mikrocontroller-Technologie der letzten Jahre hat dafür gesorgt, daß kleine und mobile Geräte sehr komplexe Aufgaben erledigen können. Ein gutes Beispiel hierfür sind moderne Lebenshilfen wie das mobile Telefon und der Personal Digital Assistent, programmierbare Taschenrechner oder GPS-Navigationssysteme.

Was hat diese Entwicklungen ermöglicht? Eine hohe Integrationsdichte und Rechenleistung bei einem Energieverbrauch, der einen mobilen Betrieb über Batterien erlaubt. Diese Eigenschaften haben aber auch ein anderes Forschungsgebiet beflügelt: Die Robotik.

In der Robotik werden kleine Steuerrechner dazu benutzt, Sensorsignale auszuwerten und auf dieser Basis Entscheidungen über Bewegungen und sonstige auszuführende Aktionen zu treffen. Solch einen Steuerrechner zu entwickeln oder sogar käuflich zu erwerben, stellt heute nicht mehr das eigentliche Hindernis auf dem Weg zum Bau eines mobilen Roboters dar; Kreativität bei der Programmierung und dem mechanischen Aufbau sind gefragt. [Adams 1992]

Dennoch gibt es Disziplinen, in denen solche Miniaturrechner entscheidende Nachteile aufweisen. Eine davon ist die dauerhafte, nichtflüchtige Speicherung von Daten. Ein Roboter, der versucht einen Weg durch ein Labyrinth zu finden, tut dies immer wieder von neuem, anstatt sich eine erfolgreiche Route zu merken. Der Vergleich mit einem herkömmlichen Arbeitsplatzrechner hinkt, ist aber dennoch hilfreich: Kein Anwender würde sich mit dem Gedanken anfreunden, einen Brief ein zweites mal zu schreiben, weil er am nächsten Morgen einen Tippfehler entdeckt hat.

So lassen sich in der Robotik vielfältige Möglichkeiten zur Nutzung von Massenspeichern finden: Die angesprochene Wegfindung genauso wie die Erstellung

von Landkarten oder die Verteilung von Sensorwerten im Raum, z.B. bei einer Temperaturkarte. Denkbar ist auch die Aufzeichnung und Wiedergabe von Ton und Bild oder die Protokollierung von automatisierten Meinungsumfragen in einer Fußgängerzone. Hier ist wieder die Kreativität der Entwickler gefragt, vorausgesetzt die Möglichkeit zur Speicherung großer Datenmengen besteht überhaupt.

Diese Voraussetzung zu schaffen ist Gegenstand dieser Diplomarbeit.

1.2 Aufbau der Arbeit

Kapitel 2 analysiert die Anforderungen an eine geeignete Speicherlösung und überprüft, ob der Markt ein entsprechendes Produkt bereithält. In Kapitel 3 wird ein Design für diese Speicherlösung formuliert und in Kapitel 4 umgesetzt. Kapitel 5 bietet eine Zusammenfassung und einen Ausblick.

2 Analyse

Die folgende Untersuchung soll klären, welche Voraussetzungen für den beschriebenen Einsatz eines Massenspeichers in der Robotik erfüllt sein müssen und inwiefern heute erhältliche Produkte dazu in der Lage sind.

2.1 Anforderungen

Welche Speicherlösung ist "gut"? Die Beantwortung dieser Frage ergibt sich aus den Anforderungen, die der Nutzer stellt. Für den Betreiber eines Datenbank-Servers steht sicherlich die Datensicherheit bzw. -verfügbarkeit im Vordergrund, während für einen privaten PC-Benutzer die Anschaffungskosten wichtiger sind. In diesem Abschnitt werden die Eigenschaften eines Massenspeichers beschrieben, die eine einfache Integration in ein Robotik-Projekt ermöglichen.

2.1.1 Hardware-Anforderungen

Speichermedium

- Geringes Ausmaß und Gewicht
- Niedriger Energieverbrauch
- Einfache Spannungsversorgung (5V oder 3,3V)
- Hohe Speicherkapazität
- Datenerhalt ohne Stromversorgung
- Niedrige Kosten
- Hohe Verfügbarkeit am Markt

Schnittstelle

- Vorhandensein dieser Schnittstelle in der Mehrzahl der Robotik-Steuerrechner bzw. einfache Integration
- Klar dokumentiertes Protokoll
- Robustheit gegen Fehlbeschaltung
- Möglichst geringe Anzahl von I/O-Leitungen
- Handelsübliche Steckverbindungen

2.1.2 Funktionale Anforderungen

- Speicher löschen bzw. formatieren
- Daten an Adresse x schreiben
- Daten von Adresse x lesen
- Schreibschutz setzen/entfernen
- Medium identifizieren (falls Wechselmedium)
- Medieninformationen lesen (Größe, Medium leer etc.)

Hierbei handelt es sich um den *minimalen* Funktionsumfang für einen sinnvollen Einsatz in der Robotik! Auf denkbare Erweiterungen wird in Kap. 5.2 näher eingegangen.

2.1.3 Sonstige Anforderungen

- Die komplette Speicherlösung darf den finanziellen Rahmen eines „Low-Cost“-Projektes nicht sprengen
- Es sollten Möglichkeiten zur Fehlersuche gegeben sein (Debugging)

2.2 Marktanalyse

Nachdem im letzten Abschnitt ein grober Anforderungskatalog aufgestellt wurde, soll nun überprüft werden, ob es bereits geeignete Lösungen gibt. Hier müssen Kompromisse eingegangen werden: Der Speicher mit der größten Kapazität wird nicht besonders klein und kostengünstig sein.

2.2.1 Festplatten

Abbildung 2.1: Geöffnete Festplatte

Festplatten bieten die größten Kapazitäten unter den Datenträgern mit wahlfreiem Zugriff und weisen das beste Verhältnis von Kapazität zu Preis auf. Unabhängig von der verwendeten Schnittstelle (s.u.) ist der Stromverbrauch durch die mechanische Bauart sehr hoch. Eine oder mehrere magnetisch polbare Scheiben werden durch einen Motor in Rotation versetzt. Über diesen Platten schwebende Schreib-/Leseköpfe ändern das Magnetfeld, um Daten zu schreiben, oder werten es aus, um Daten zu lesen.

EIDE

EIDE-Festplatten benutzen ein sehr breites paralleles Interface und benötigen sowohl 5 Volt als auch 12 Volt Versorgungsspannung. Das Protokoll ist gut dokumentiert, für einen Robotik-Steuerrechner aber zu aufwendig. Das Preis-/Leistungsverhältnis ist hier am besten.

SCSI

Der SCSI-Bus kommt mit weniger Leitungen als der EIDE-Bus aus, ist mit mindestens 8 Datenleitungen und einigen Steuersignalen für einen Kleinstrechner aber immer noch zu breit.

USB

Durch den Universal Serial Bus erhalten die Festplatten eine serielle Schnittstelle, die mit einer geringen Anzahl von I/O-Leitungen auskommt. Der USB wurde auf einfache Handhabung durch den Endanwender optimiert (Stichwort: 'Plug and Play'). Sämtliche Fehlerzustände und Konfigurationen werden durch das komplexe Protokoll geregelt. [Usb] Diesen Protokollstack auf einen Steuerrechner zu portieren, erfordert mehr Ressourcen als bei jeder anderen hier behandelten Schnittstelle.

2.2.2 Speicherkarten

Abbildung 2.2: Verschiedene Typen von Flash-Speicherkarten

Speicherkarten auf Basis der Flash-Technologie werden vorwiegend in tragbaren Consumer-Geräten wie Digitalkameras und MP3-Playern eingesetzt. Trotz unterschiedlicher Standards haben diese Karten viele Gemeinsamkeiten: Sie sind klein, unempfindlich gegen mechanische Belastung und für Batteriebetrieb ausgelegt. Erhältliche Speicherkapazitäten bewegen sich in einer Größenordnung,

die vor einigen Jahren noch Festplatten vorbehalten war, bei akzeptablem Preis-/Leistungsverhältnis.

Die Flash-Technologie

Flash-Speicher ist ein nichtflüchtiger Speicher, der funktionell mit einem RAM-Speicher vergleichbar ist, aber blockweise beschrieben und gelöscht werden muss, während sich ein RAM-Speicher wortweise beschreiben und löschen läßt. Durch die blockorientierte Arbeitsweise und die Datenpersistenz ohne Stromversorgung eignet sich Flash-Speicher als Ergänzung oder als Ersatz für Festplatten in portablen Computern.

Compact Flash

Speicherkarten im Compact Flash-Format besitzen den größten Marktanteil und den niedrigsten Preis pro Megabyte. CF-Karten sind bis zu einer Größe von 512 MB erhältlich, eine Version mit 1 GB ist angekündigt. Jede CF-Karte kann sowohl mit 5 Volt als auch mit 3,3 Volt betrieben werden.

Die 40-polige Schnittstelle ist kompatibel zum IDE-Standard und um einige Befehle und eine zusätzliche Betriebsart erweitert worden. Die komplette Dokumentation des Protokolls ist im Internet erhältlich. [Cfspec] Mittels eines passiven Adapters kann die CF-Karte wie eine IDE-Festplatte angeschlossen und betrieben werden.

Smartmedia

Smartmedia-Karten sind mit maximal 128 MB erhältlich, größere Versionen sind nicht angekündigt. Die Versorgungsspannung beträgt 3 Volt, bei älteren Karten 5 Volt. Auf der Smartmedia-Karte ist weniger Controller-Logik vorhanden als bei den anderen Typen, es muß mehr Intelligenz vom Host-Rechner aufgebracht werden.

Die Protokoll-Definition der 22-poligen parallelen Schnittstelle ist nur gegen Bezahlung vollständig verfügbar, wodurch eine Nutzung in nicht-kommerziellen

Bereichen sehr erschwert wird.

Memory Stick

Für den Memory Stick der Firma Sony gelten ähnliche Voraussetzungen wie für die Smartmedia-Karte: Keine Offenlegung der technischen Details und eine Beschränkung auf 128 MB. [Mstick]

Multimediacard

Die Multimediacard besitzt als einzige Flash-Karte eine serielle Schnittstelle, basierend auf dem SPI-Standard. SPI findet man vor allem in Mikrocontrollern der Firma Motorola, aber auch in diversen neueren Chips anderer Hersteller. Das Protokoll ist gut dokumentiert, erfordert aber genaues Zeitverhalten auf Host-Seite. [Mmc]

Die Speicherkapazität liegt mit max. 64 MB niedriger als bei anderen Flash-Karten, die Versorgungsspannung beträgt 3,3 Volt.

2.2.3 Bauteile

Eeprom

Serielle Eeproms sind mit I2C, SPI oder Microwire Schnittstelle ausgestattet. Sie benötigen nur 2-3 I/O-Leitungen und sind über gut definierte, einheitliche Protokolle anzusprechen. Die Versorgungsspannung kann häufig in einem weiten Bereich variieren, bei Produkten der Firma Microchip zum Beispiel von 1,8 Volt bis 5,5 Volt. [Microchip] Der Energieverbrauch und die mechanischen Ausmaße sind äußerst gering.

Diese für die Robotik günstigen Eigenschaften werden durch die verfügbare Speicherkapazität ausgeglichen: Es steht maximal ein Megabit zur Verfügung.

Dram

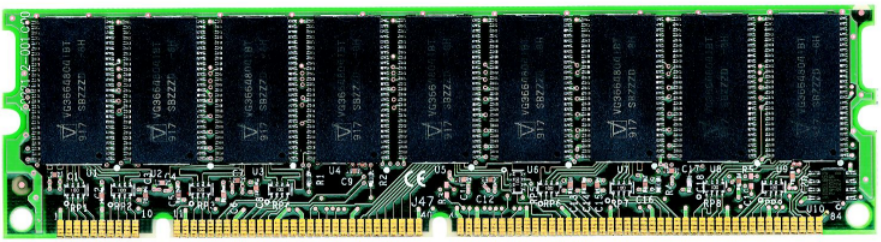

Abbildung 2.3: SDram-Modul für den PC

Dynamisches Ram ist durch den Einsatz als Hauptspeicher im PC ein günstiges Massenprodukt und in Größen bis zu 512 MByte pro Modul erhältlich. Die Optimierung auf Geschwindigkeit (einige hundert MByte pro Sekunde) hat zu einer sehr hohen Busbreite geführt, ein entsprechender Sockel muß direkt im Platinen-Layout vorgesehen werden. Zudem muß dynamischer Speicher von einem separaten Controller in definierten Zeitabständen aufgefrischt werden, typischerweise im Millisekunden-Bereich.

2.2.4 Vergleich

Die vorangegangene Beschreibung der einzelnen Speicherlösungen erlaubt keinen direkten Vergleich und damit keine Bewertung der Tauglichkeit für den Problembereich. Durch eine tabellarische Überprüfung der Anforderungen soll das geeignetste Speichermedium gefunden werden.

Legende

Für die nachfolgende Vergleichstabelle wird eine 5-stufige Bewertung verwendet, die von '++ sehr gut' über 'o durchschnittlich' bis '– – sehr schlecht' reicht.

Speichermedium

	Compact Flash	Smart Media	Memory Stick	Multimediacard	Festplatte	Eeprom	Dram
Ausmaß und Gewicht	+	+	+	+	– –	++	o
Energieverbrauch	+	+	+	+	– –	++	+
Spannungsversorgung	++	+	+	+	– –	++	+
Speicherkapazität	+	o	o	–	++	– –	o
Datenerhalt	+	+	+	+	+	+	–
Kosten/Kapazität	+	+	o	o	++	– –	+
Verfügbarkeit	+	+	–	–	++	o	++

Festplatten sind durch ihre Größe und den Energieverbrauch für einen mobilen Roboterbetrieb nicht geeignet. Dram hat genauso schwerwiegende Nachteile, die fehlende Persistenz ist hier am gravierendsten.

Das Eeprom ist sehr geeignet, wenn nur einige KByte an Daten gespeichert werden müssen. Es erfüllt damit nicht unsere Anforderungen an einen Massenspeicher, zeigt aber gut mit welchen Einschränkungen bezüglich der Kapazität Entwickler im Moment auskommen müssen.

Flash-Speicherkarten erfüllen die gestellten Anforderungen zum größten Teil. Diese Karten sind klein, stellen niedrige Anforderungen an die Spannungsversorgung und sind in fast jedem Elektronikgeschäft zu günstigen Preisen und mit hohen Kapazitäten verfügbar.

Unter den vier gängigen Standards hebt sich Compact Flash durch eine IDE-kompatible Schnittstelle, eine flexible Spannungsversorgung und durch das beste Preis-/Leistungsverhältnis hervor.

2.2.5 Anschlußmöglichkeiten

Es gibt grundsätzlich zwei Möglichkeiten, Flash-Speicherkarten an einen Host zu koppeln:

- Direktanschluß der karteneigenen Schnittstelle
- Anschluß über ein Lesegerät für USB oder Centronics

Diese beiden Möglichkeiten sollen nun auf ihre Eignung für unseren Problembereich untersucht werden.

Direktanschluß

Unter einem Direktanschluß einer Speicherkarte versteht man die Integration entsprechender Kartensockel und Controller-Chips schon in der Entwicklungsphase eines Gerätes, z.B. einer Digitalkamera. Bei Konsumentengeräten ist dies sinnvoll, da die Fertigungskosten eine größere Rolle spielen als einmalige Entwicklungskosten. Entsprechend würde man bei einer Serienproduktion eines Roboters verfahren.

In diesem Szenario liegt die Priorität auf dem einfachen Anschluß an einen bestehenden Rechner, für den Direktanschluß einer Speicherkarte wäre aber ein Re-Design der Platine nötig. Damit eignet sich diese Anschlußmöglichkeit nicht, die Karte sollte sich über eine schon vorhandene Schnittstelle ansprechen lassen.

Lesegeräte

Um Flash-Karten auch am PC beschreiben und lesen zu können benötigt man ein Lesegerät. Diese Adapter sind sowohl mit USB als auch mit Centronics-Anschluß erhältlich. Mittels eines herstellerspezifischen Treibers werden Flash-Karten ähnlich einer Festplatte in populäre Betriebssysteme eingebunden. Für kleine Steuerrechner gibt es solche Treiber nicht, was zum Teil daran liegt, daß die entsprechenden Protokolle von den Herstellern nicht offengelegt werden. Für eine Integration in Robotik-Rechner sind diese Lesegeräte dementsprechend denkbar ungeeignet, selbst wenn USB oder Centronics hardwareseitig schon vorhanden sind, was nur selten der Fall ist.

Abbildung 2.4: USB-Lesegerät

RS-232

Eine geeignete Schnittstelle für den Anschluß von Speicherkarten sollte in möglichst jedem Rechnersystem schon vorhanden sein. Das ist bisher nur bei RS-232 der Fall. Personal Computer, eingebettete Systeme und selbst kleinste Microcontroller sind mit diesem Interface ausgestattet. RS-232 benötigt nur wenige I/O-Leitungen und ist sehr robust gegen Fehlbeschaltungen. [Kainka 1993]

Ein Adapter von RS-232 auf Flash-Speicherkarten ist derzeit am Markt nicht erhältlich.

Protokoll

Die RS-232 Norm definiert nur die unteren Schichten einer Datenübertragung wie Steckerbelegungen, elektrische Kenngrößen, Pegelkodierungen und das Zeitverhalten bis hin zur Übertragung eines Bytes. Darüber hinaus sind keine höheren Übertragungsprotokolle definiert. Es gibt keine festgelegte Fehlerkorrektur, Adressierung oder Rahmenstruktur.

Es existiert kein genormtes Protokoll, das für die beschriebene Ankopplung einer Speicherlösung ohne Änderungen übernommen werden könnte. HDLC zum Beispiel beinhaltet eine Fehlerkorrektur und eine genaue Rahmenstruktur, müßte aber um die Definition der übertragenen Botschaften erweitert werden. [Tanenbaum 2000]

Die Vorgabe, die Protokollkomplexität möglichst minimal zu halten, spricht für eine eigens zu entwerfende Norm.

2.3 Fazit

Eine geeignete Speicherlösung würde laut Analyse aus folgenden Komponenten bestehen:

- Speichermedium: Compact Flash
- Interface: RS-232
- Protokoll: proprietär

Ein Konverter von RS-232 auf Compact Flash würde das beste Speichermedium über die geeignetste Schnittstelle ansprechbar machen. Voraussetzung dafür ist ein simples, auf ein Minimum reduziertes Kommunikationsprotokoll, für das keiner der bekannten Standards in Frage kommt.

Im nächsten Kapitel soll ein Design für solch einen Konverter entwickelt werden, das allen genannten Anforderungen entspricht.

3 Design

3.1 Der ADT Speicher

In Abschnitt 2.1.2 wurden die funktionalen Anforderungen an den Speicher definiert. Diese kann man gut als abstrakten Datentyp (ADT) [Owsnicki-Klewe 1995] darstellen, auf den der Host zugreift:

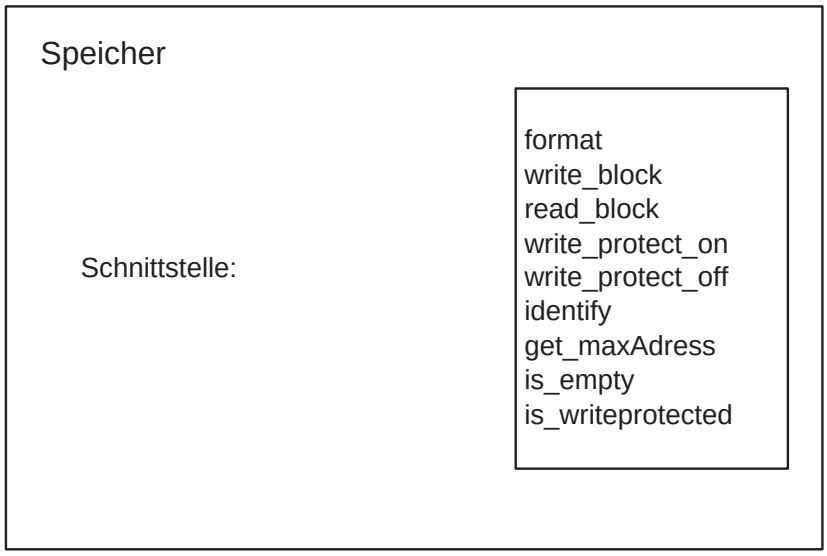

Abbildung 3.1: Der ADT Speicher

Allgemeine Bedienung des Speichers

Eine einwandfreie Definition des ADT wird dadurch erschwert, daß es sich bei unserem Speicher um ein Wechselmedium handelt. Dieses Wechselmedium kann zu

jedem Zeitpunkt durch den Benutzer entfernt oder gegen ein anderes ausgetauscht werden.

Um diesem Umstand Rechnung zu tragen, müßte der Host vor jedem Befehl überprüfen, ob noch das gewünschte Medium eingesteckt ist. Dies ist mit der Funktion identify() möglich.

Es wird aber im weiteren davon ausgegangen, daß der Benutzer mit dem Gerät zusammenarbeitet, d.h. das Wechselmedium nur dann entfernt, wenn es unbedenklich ist. Eine fehlertolerante *Hot-Swap* Lösung würde den Rahmen dieser Arbeit sprengen.

3.1.1 Schnittstelle

1. *format()*
   - Precondition: is_write_protected liefert FALSCH; identify() liefert Wert ungleich 0;
   - Postcondition: Ein read_block(Adresse) mit 0 $<$ Adresse $<$ maxAdress liefert einen Block aus Nullen zurück.
     Ein is_empty() liefert WAHR.
2. *write_block(Block,Adresse)*
   - Precondition: 0 $<$ Adresse $<$ maxAdress ; identify() liefert Wert ungleich 0; Länge(Block)=512; is_write_protected liefert FALSCH;
   - Postcondition: Nach einem write_block(Block_X,n) liefert ein read_block(n) mit der gleichen Adresse n als Parameter den selben Block_X zurück, wenn identify() vor und nach der Ausführung der Funktionen den gleichen Medienbezeichner liefert.
     Ein is_empty() liefert FALSCH.
3. *read_block(Adresse)*
   - Precondition: 0 $<$ Adresse $<$ maxAdress; identify() liefert Wert ungleich 0;
   - Postcondition: Liefert den Inhalt des an der Adresse befindlichen Blocks. Wurde an diese Adresse nach einem format() noch nichts geschrieben, so wird ein Block aus Nullen zurück geliefert. Wurde weder formatiert noch geschrieben, so ist der Inhalt des zurückgegebenen Blocks undefiniert.

4. *write_protect_on()*
    - Precondition: identify() liefert Wert ungleich 0;
    - Postcondition: is_write_protected liefert WAHR.
5. *write_protect_off()*
    - Precondition: identify() liefert Wert ungleich 0;
    - Postcondition: is_write_protected liefert FALSCH.
6. *identify()*
    - Precondition: –
    - Postcondition: identify() liefert einen eindeutigen Medienbezeichner zurück. Ist kein Medium ansprechbar, so wird eine 0 übergeben.
7. *get_maxAdress()*
    - Precondition: –
    - Postcondition: Liefert die größte erlaubte Adresse des Speichermediums. Ist kein Medium ansprechbar, so ist die größte erlaubte Adresse 0.
8. *is_empty()*
    - Precondition: identify() liefert Wert ungleich 0;
    - Postcondition: Liefert genau dann WAHR, wenn nach einem format() noch kein write_block() ausgeführt wurde, sonst FALSCH.
9. *is_writeprotected()*
    - Precondition: identify() liefert Wert ungleich 0;
    - Postcondition: Liefert genau dann FALSCH, wenn ein write_protect_off() ohne nachfolgendes write_protect_on() ausgeführt wurde, sonst WAHR.

Implementierungsmöglichkeiten

Die beschriebenen Funktionen können als Botschaften an den Konverter realisiert werden. Dies hat den Vorteil, daß der Host nur wenig Arbeit übernehmen

muß und damit die Integration leichter vonstatten geht. Entsprechend muß ein Protokoll definiert werden, welches für jede Schnittstellenfunktion des ADT eine Protokollbotschaft an den Konverter sowie eine Antwort zum Host beinhaltet.

3.1.2 Protokoll

Befehle an den Konverter

Befehl	Byte 1	Byte 2	Byte 3-6	Byte 4-515
format	1	1		
write_protect_on	1	2		
write_protect_off	1	3		
identify	1	4		
get_maxAdress	1	5		
is_empty	1	6		
is_writeprotected	1	7		
write_block	1	16	Adresse (MSB first)	Daten
read_block	1	17	Adresse (MSB first)	

Jeder Befehl fängt mit einer 1 an. Dies erleichtert einerseits die Synchronisation nach Übertragungsfehlern, es kann z.B. nach einer Fehlübertragung auf diese 1 gewartet werden. Andererseits kann dieses Protokoll leicht zu einer adressierten Variante erweitert werden, die trotzdem abwärtskompatibel bleibt. Hierfür kann das erste Byte als Adresse des Konverters interpretiert werden. Der Betrieb von mehreren Konvertern an einem Host wäre so möglich. Auf diese Erweiterungsmöglichkeit wird in Kap.5.2 noch näher eingegangen.

Das zweite Byte kennzeichnet den auszuführenden Befehl. Hierbei wurde eine Grenze gezogen: Unparametrisierte Befehle werden durch Zahlen bis 15 ausgedrückt, parametrisierte durch Zahlen ab 16 aufwärts. Dies erleichtert die Realisierung: Ist das zweite empfangene Byte kleiner als 16, so kann der Befehl sofort ausgeführt werden.

Speicheradressen werden stets als 32-bit Werte übertragen, mit dem höchstwertigen Bit zuerst. Ein Datenblock besteht immer aus 512 Byte.

Die maximal adressierbare Speicherkapazität beträgt also:

2^{32} Blocks $\cdot$ 512 $\frac{Byte}{Block}$ = 2 Terabyte

Diese Größenordnung wird auch für zukünftige Speicherkarten ausreichend sein.

Antworten an den Host

Antwort auf	**Byte 1**	**Byte 2 –**
format	RC (0,1,2,3)	
write_protect_on	RC (0,1,2)	
write_protect_off	RC (0,1,2)	
identify	RC (0,1,2)	20 Byte Seriennummer
get_maxAdress	RC (0,1,2)	4 Byte maxAdress
is_empty	RC (0,1,2)	1 Byte Boolean (0=FALSCH,1=WAHR)
is_writeprotected	RC (0,1,2,3)	1 Byte Boolean (0=FALSCH,1=WAHR)
write_block	RC (0,1,2,3,4)	
read_block	RC (0,1,2,4)	(512 Byte Daten)
Fehlübertragung	RC (16,17)	

RC bedeutet Returncode. Die Zahlen in Klammern sind die für diesen Befehl möglichen Returncodes.

Jede Antwort eines Konverters an den Host beginnt mit einem Returncode. Generell bedeutet eine 0 die erfolgreiche Ausführung des Befehls, ein Wert größer als 0 einen Fehler. Diese Schema vereinfacht die Realisierung auf Host-Seite, da es sich an Konventionen üblicher Programmiersprachen anlehnt, wie z.B. C++. [Prata 1992]

Wird ein unbekannter oder falsch formatierter Befehl empfangen, so wird nur der entsprechender Returncode (16 oder 17) versendet.

Ab dem zweiten Byte unterscheidet sich die Struktur der Antworten. Eine Besonderheit stellt die Antwort auf den Befehl read_block() dar: Um nicht unnötig

Ressourcen zu verbrauchen wird der Datenblock nur dann übergeben, wenn der Returncode 0 ist.

Returncodes der Antworten

Returncode	Bedeutung
0	Befehl fehlerfrei verarbeitet
1	Kein Speichermedium
2	Speichermedium reagiert nicht
3	Schreibvorgang nicht erlaubt
4	Adressfehler
16	unbekannter Befehl
17	Syntaxfehler

3.1.3 Beispielablauf

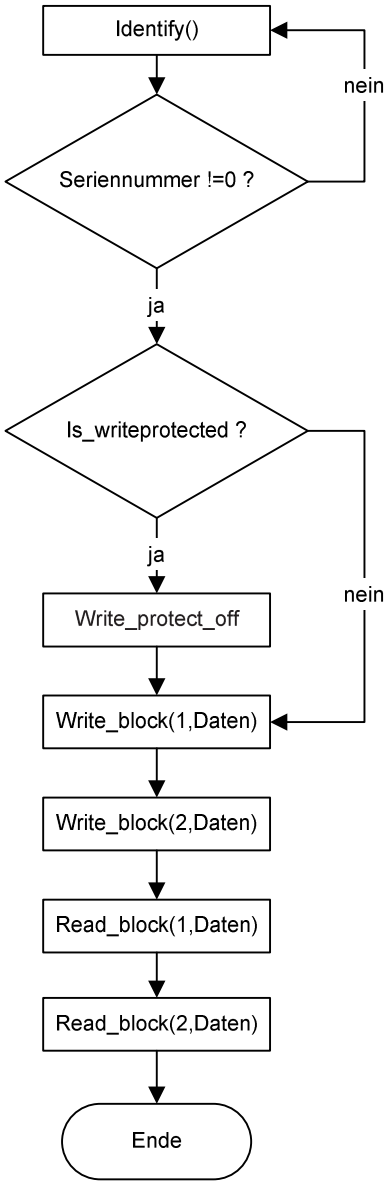

Abbildung 3.2: Flußdiagramm einer einfachen Anwendung

Das Flußdiagramm zeigt eine einfache Benutzung des ADT. Zuerst wird solange das Identify-Kommando wiederholt, bis eine gültige Karte erkannt wird. Ist der Schreibschutz gesetzt, wird er gelöscht. Danach werden 2 Blocks gespeichert und wieder gelesen.

3.2 Hardware-Anforderungen

3.2.1 Logik

Um zwischen den beiden komplexen Protokollen übersetzen zu können, bedarf es mehr als einer festverdrahteten Logik. Ein Microcontroller oder -prozessor muß diese Aufgabe übernehmen. Die benötigte Rechenleistung ist minimal, da die RS-232 Schnittstelle als langsamste Komponente die Arbeitsgeschwindigkeit bestimmt. Selbst niedrig getaktete 8-bit Microcontroller sind in der Lage, solch eine Schnittstelle voll auszulasten.

3.2.2 Speicher

Ram

Um den beschriebenen ADT zu realisieren, muß ein kompletter Datenblock von 512 Bytes zwischengespeichert werden. Ein direktes Durchreichen der Daten von einer Schnittstelle zur anderen ist zwar denkbar, führt in einigen Fällen aber zu Komplikationen:

- Wird zum Beispiel die Übertragung eines write_block-Befehls unterbrochen, so hat der Konverter die angekommenen Daten schon auf den Datenträger geschrieben. Da immer ein kompletter Block geschrieben werden muß, bleibt dem Konverter nur das Auffüllen des Blocks. Undefiniert beschriebene Blöcke sind die Folge.
- Ähnliches gilt für den read_block-Befehl, bei dem der Host nicht über einen aufgetretenen Lesefehler informiert werden kann, da der Returncode für eine erfolgreiche Ausführung bereits verschickt wurde.

Da noch zusätzliches Ram für übliche Variablen des Programmablaufes gebraucht werden, sind ungefähr 600 Bytes Ram nötig.

Rom

Im Rom wird das auszuführende Programm abgelegt. Die Größe des Programmcode zu schätzen ist schwierig: Sie hängt von der verwendeten Programmiersprache, der Prozessorarchitektur und dem Programmierer selbst ab. Generell ist der Bedarf eher niedrig, einige tausend Befehlswörter werden ausreichen.

3.2.3 RS-232

Da der Konverter an eine Vielzahl von unterschiedlichen Hosts angeschlossen werden soll, müssen wichtige Parameter der Schnittstelle änderbar sein. Dies bezieht sich hauptsächlich auf die Baudrate der Übertragung. Diese Änderungen müssen zwar nicht im laufenden Betrieb, aber ohne Neuprogrammierung des Konverters durchführbar sein.

Zweckmäßigerweise sollte hier ein UART (*Universal Asynchronous Receiver Transmitter*) zum Einsatz kommen. Dieser Baustein übernimmt die Umwandlung von parallel anliegenden Daten in serielle Bitströme und umgekehrt.

Ebenso wichtig für die Kompatibilität mit unterschiedlichen Hosts sind genormte RS-232 Leitungspegel, die sich von den üblichen TTL-Pegeln unterscheiden.

3.2.4 I/O

Um Compact Flash ansteuern zu können, ist ein Minimum von 24 I/O-Leitungen erforderlich. 16 dieser Leitungen, der Datenbus, müssen als Aus- und Eingänge schaltbar sein. Die restlichen Leitungen sind Ausgänge.

3.2.5 Caching

Im Zusammenhang mit Datenträgern wird häufig *Caching* verwendet. Darunter versteht man das Zwischenspeichern häufig benutzter Daten zur schnelleren Verarbeitung.

Auch dieser Konverter könnte, mehr Ram vorausgesetzt, mit einem Caching-Algorithmus ausgestattet werden. Dies ist aber aus folgenden Gründen nicht sinnvoll:

- Der Datenträger liefert Informationen um Potenzen schneller, als sie über RS-232 mit dem Host ausgetauscht werden können. Es wäre also nur ein marginaler Geschwindigkeitszuwachs zu erwarten.
- Der Konverter kann keine Voraussagen darüber treffen, welchen Block der Host als nächstes anfordert. Dies ist nur bei einem auf dem Konverter realisierten Dateisystem möglich.

3.2.6 Schnittstellenparameter

Um eine Kommunikation zwischen Host und Konverter aufzubauen müssen essentielle Parameter wie z.B. die Baudrate auf beiden Seiten übereinstimmen. Daher kann diese Einstellung nicht vom Host über das Protokoll erfolgen, dazu wäre die korrekte Funktion ja schon notwendig. Zwei verschiedene Vorgehensweisen sind denkbar:

- Die Parameter werden über *DIP-Schalter* eingestellt. Der Konverter liest diese Schalterstellung zu Beginn des Programmablaufs aus und paßt die Schnittstellenparameter an.
- Der Konverter wird über einen Schalter in einen *Debug-Modus mit definierten Schnittstellenparametern* gebracht. Über einen PC mit Terminalprogramm können die Parameter geändert und in einem Konvertereigenen Eeprom gespeichert werden.

Beide Varianten haben Vor- und Nachteile: Die Lösung über Schalter ist einfach, der Debug-Modus ist flexibler und erlaubt spätere Erweiterungen.

Für dieses Design wird die erste Variante gewählt, da einfache Realisierung und Bedienung wichtiger sind als Funktionsvielfalt.

3.2.7 Zusammenfassung

Zusammengefaßt ergeben sich folgende Anforderungen an die Hardware:

- RS232-UART, möglichst in Hardware und inklusive FIFO
- verstellbare Kommunikationsparameter wie Baudrate etc.
- DIP-Schalter zur Baudrateneinstellung
- 'echte' RS232-Leitungspegel
- 24 I/O-Pins für die CF-Schnittstelle
- ca. 600 Byte Ram
- Rom-Speicher für Programmcode

Hieraus läßt sich ein grobes Blockschaltbild mit logischen Funktionseinheiten ableiten:

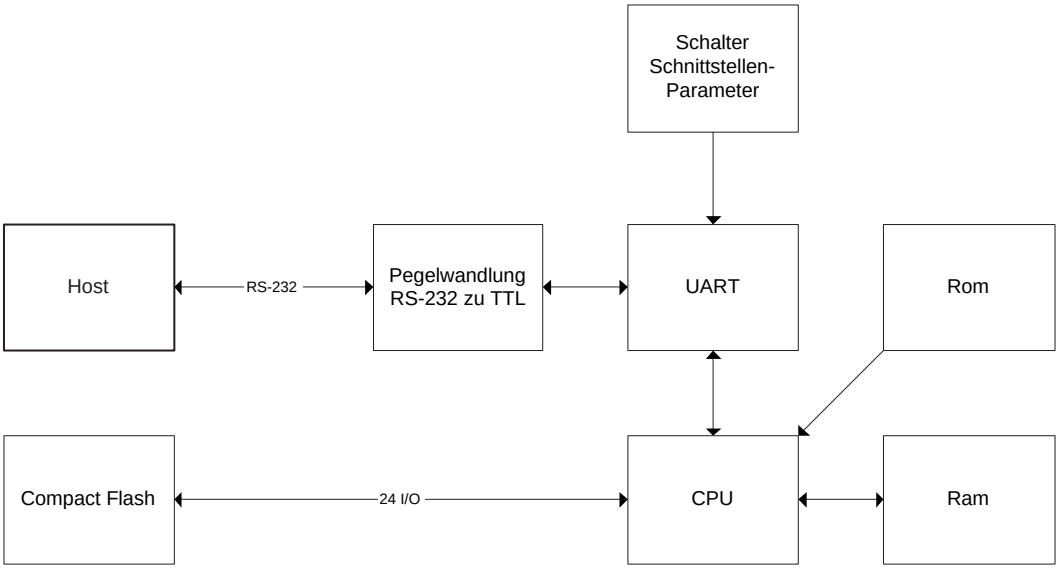

Abbildung 3.3: Blockschaltbild des Konverters

4 Realisierung

4.1 Auswahl der Logik

Warum ein Microcontroller?

Die Entscheidung, ob ein Microcontroller oder ein Microprozessorsystem vorzuziehen ist, ergibt sich in diesem Fall aus den oben genannten Anforderungen. Ein Microcontroller ist die bessere Wahl, wenn durch die integrierte Peripherie der Schaltungsaufwand (und damit die Kosten) deutlich verringert werden kann. Fehler beim Schaltungsdesign werden minimiert, die integrierte Peripherie ist tausendfach getestet und arbeitet zuverlässig. Der Programmierer findet sämtliche wichtige Informationen im Datenblatt des Microcontrollers. Ein Microprozessor hingegen bietet meistens höhere Rechenleistung, die für den Konverter aber nicht vonnöten ist.

Im folgenden soll ein Microcontroller ausgewählt werden, der einen einfachen und kostengünstigen Aufbau ermöglicht.

4.1.1 Der PIC 17C756

Für dieses Projekt habe ich den PIC 17C756 der Firma Microchip ausgewählt. Hauptgrund dafür ist die bei mir schon vorhandene Programmiererfahrung mit diesem Controller. Die oft sehr zeitaufwendige Einarbeitung in die Programmierung eines Controllers entfällt somit vollständig. Die Entwicklungsumgebung (Preis ca. 1300 Euro) wurde schon für ein vorangegangenes Projekt angeschafft.

Bei diesem Controller handelt es sich um einen 8-bit RISC Prozessor *(Reduced Instruction Set Computer)* mit 16-bit Befehlswörtern, erweitert um Rom, Ram und zusätzliche Peripherie. Der Controller arbeitet nach der Harvard Architektur, Datenspeicher und Programmspeicher sind physikalisch getrennt. Er ist in

zwei Versionen erhältlich: Die wiederprogrammierbare Eprom-Version zum Entwickeln (Preis ca. 20 Euro) und die nur einmal programmierbare OTP-Version für die Fertigung (Preis ca. 8 Euro) .

Im folgenden werden die relevanten Eckdaten des Controllers wiedergegeben, die komplette Dokumentation ist im Internet bei Microchip verfügbar und umfaßt 304 Seiten. [17c756]

Kern

- 58 Befehlswörter
- 16K Programmspeicher
- 902 Byte Ram
- Interruptfähig (mehrere Quellen)
- 16 Ebenen tiefer Hardware Stack
- DC-33 Mhz Takt
- Direkte, indirekte und relative Adressierungsmodi
- Betriebsspannung von 2,5 bis 6 Volt
- Stromaufnahme $<$5mA bei 5V und 4 Mhz

Peripherie

- 50 I/O-Pins
- 4 Capture-Eingänge mit 16 bit
- 3 PWM-Ausgänge mit bis zu 10 bit
- 4 Timer bzw. Zähler
- 2 UARTs mit 2-byte FIFO
- 12-Kanal A/D-Wandler mit 10 bit
- Synchroner serieller Port für SPI / I2C

- Watchdog Timer
- Sleep-Modus

4.1.2 Bewertung

Vergleich mit den Anforderungen aus Abschnitt 3.2.7

Anforderungen	Microchip PIC 17C756
RS232-UART in Hardware inkl. FIFO	ja, 2 Stück
verstellbare Baudrate etc.	ja
Dip-Schalter	nein
'echte' RS232-Leitungspegel	nein
24 I/O-Pins für die CF-Schnittstelle	ja, 50 I/O-Pins
ca. 600 Byte Ram	ja, 902 Byte
Rom-Speicher für Programmcode	ja, 16K Worte

Bis auf zwei Ausnahmen werden sämtliche Anforderungen an die Hardware durch den PIC 17C756 weit übertroffen. Lediglich die DIP-Schalter und eine Pegelwandlung von TTL auf RS-232 sind noch erforderlich.

Blockschaltbild

Grafisch dargestellt wird deutlich, welche Hardware durch den Microcontroller realisiert wird. (s.Abb. 4.1)

Die in den PIC integrierte Peripherie erlaubt noch Erweiterungen des Designs, zum Beispiel um einen I2C-Link zum Host oder die Messung von Analogwerten. Ein Ausblick auf diese Möglichkeiten wird in Kap. 5.2 gegeben.

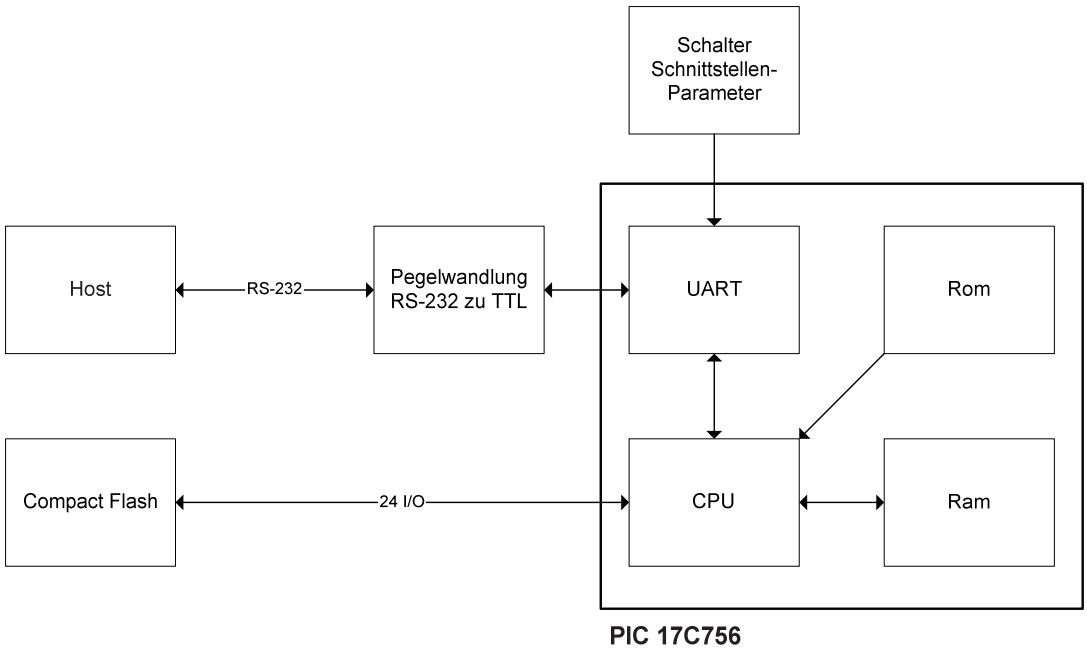

Abbildung 4.1: Integrierte Peripherie des PIC 17C756

4.1.3 Alternativen

Hersteller

Auf dem Markt existiert eine Vielzahl von gebräuchlichen Microcontrollern, zu einem großen Teil von folgenden Herstellern:

- Atmel (AVR-Serie)
- Motorola (z.B. 68HC11)
- Ubicom (SX-Serie)

Bei fast jedem Hersteller läßt sich ein geeigneter Controller finden. Im folgenden werden beispielhaft einige Alternativen zu dem gewählten PIC 17C756 aufgezeigt und mit ihm verglichen.

Atmel AT90S8515

Atmel hat mit seiner AVR-Serie eine breite Palette von Risc-Microcontrollern im Angebot. Der AT90S8515 ist der kleinste verfügbare Controller, der den Anforderungen entspricht. Ähnlich dem PIC 17C756 ist ausreichend Ram, Rom, I/O und ein UART vorhanden. Einen integrierten UART mit RS-232 Leitungspegeln, der einen Controller noch geeigneter machen würde, gibt es auch bei Atmel nicht.

Eine Entwicklungsumgebung ist für weniger als 100 Euro verfügbar, also deutlich günstiger als die des PIC-Controllers.

Motorola 68HC11E20

Dieser Motorola-Controller basiert auf dem Kern des 6805 Cisc-Prozessors. Der 68HC11E20 besitzt ebenfalls eine dem PIC ähnliche Peripherie-Ausstattung und erfüllt damit die Anforderungen an dieses Projekt. Für diese Prozessorarchitektur ist eine Fülle von Entwicklungstools erhältlich, zum Teil kostenlos. Ein Programmiergerät ist, wie beim Atmel-Controller, günstig zu bekommen.

Ubicom SX52BD

Die Controller der Firma Ubicom zeichnen sich durch ihre hohe Geschwindigkeit aus (75 Mips). Rom und I/O-Leitungen sind genug vorhanden, die Größe des Rams (262 Byte) reicht hingegen nicht aus. Ein UART ist hardwareseitig nicht verfügbar, wird aber durch ein Software-Modul realisiert.

Um diesen Controller einsetzen zu können, müßte zusätzliches Ram angeschlossen werden. Für dieses Projekt sind Controller der Firma Ubicom somit nur sehr bedingt geeignet. Eine Entwicklungsumgebung ist ähnlich dem Atmel-System sehr kostengünstig.

4.1.4 Zusammenfassung

Es wurde gezeigt, daß der PIC 17C756 den Anforderungen des Projektes genügt und sie sogar weit übertrifft. Da es von anderen Herstellern vergleichbare Controller mit teilweise günstigeren Entwicklungsumgebungen gibt, ist die vorhandene Erfahrung der entscheidende Faktor. Ein Umstieg auf eine neue Controllerfamilie lohnt sich nur selten, da die Einarbeitungszeit den eigentlichen Programmieraufwand schnell übersteigen kann.

Für einen Entwickler, der mit keinem der Controller Erfahrung hat, ist der Atmel AVR eine günstige und gut geeignete Wahl.

4.2 Physikalisches Design

4.2.1 Spannungsversorgung

Der Konverter muß mit einer eigenen, mobilen Spannungsversorgung ausgestattet sein, um an unterschiedliche Hosts angeschlossen werden zu können. Alle Komponenten des Konverters benötigen eine stabile 5 Volt Betriebsspannung. Der Strombedarf addiert sich auf ca. 50 mA.

Als Energiequelle wird eine 9 Volt Blockbatterie verwendet. Diese Batterien sind klein und leicht und besitzen trotzdem eine ausreichend hohe Ausgangsspannung und Kapazität. Mit einer guten Batterie werden Laufzeiten von ca. 10 Stunden möglich sein.

Um aus den 9 Volt der Batterie stabile 5 Volt zu erzeugen wird ein Spannungsregler benötigt. Man unterscheidet zwischen Linear- und Schaltreglern. Linearregler sind einfacher aufzubauen, besitzen aber einen schlechten Wirkungsgrad, was sich in zusätzlicher Wärmeentwicklung und verkürzter Batterielaufzeit äußert.

Schaltregler hingegen zeichnen sich durch einen guten Wirkungsgrad bei leicht erhöhtem Schaltungsaufwand aus. Für den Konverter wurde ein Schaltregler vom Typ LM2574 benutzt. Die Beschaltung wurde dem Datenblatt der Firma National Semiconductors entnommen. [NatSemi]

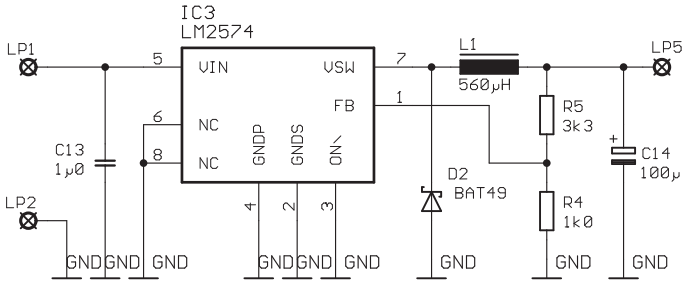

Abbildung 4.2: Schaltbild der Spannungsversorgung

4.2.2 Takterzeugung

Der PIC Microcontroller erlaubt Taktraten von DC bis 33 Mhz. Von diesem Takt wird durch einen programmierbaren Teiler auch die Baudrate des UARTs gewonnen. Geteilt wird durch Vielfache von 64, was bei einer maximalen Baudrate von 115200 eine Taktfrequenz von 7,3728 Mhz ergibt. Da diese Art der Generierung von Baudraten in der Digitaltechnik üblich ist, hält der Handel einen Quarz mit genau dieser Resonanzfrequenz bereit. Die Dimensionierung der Bauteile der Oszillatorschaltung wurde dem Datenblatt des PIC-Controllers entnommen.

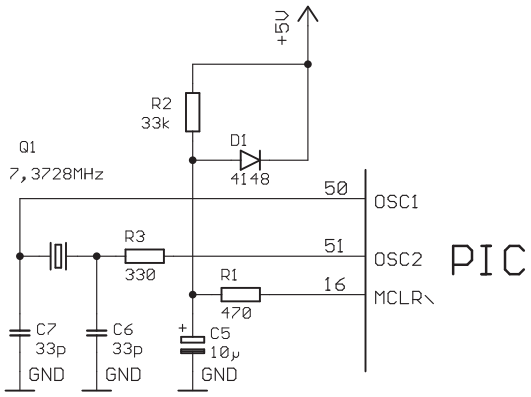

Abbildung 4.3: Schaltbild der Takterzeugung

4.2.3 RS-232

Die Sende- und Empfangsleitungen des PIC-UARTs arbeiten mit TTL-Pegeln, die RS-232 Schnittstelle mit Pegeln von +/- 12 Volt. Die Wandlung dieser Pegel übernimmt ein IC vom Typ MAX232. Es benötigt lediglich vier Kondensatoren als externe Beschaltung und unterstützt somit das Design einer möglichst kleinen Leiterplatte.

Als Steckverbindung zum Host wird eine 9-polige Sub-D Buchse eingesetzt. Die verwendete Pinbelegung entspricht einem DCE *(Data Communications Equipment)*, der Konverter kann so durch ein gerade durchverbundenes Kabel an den Host angeschlossen werden.

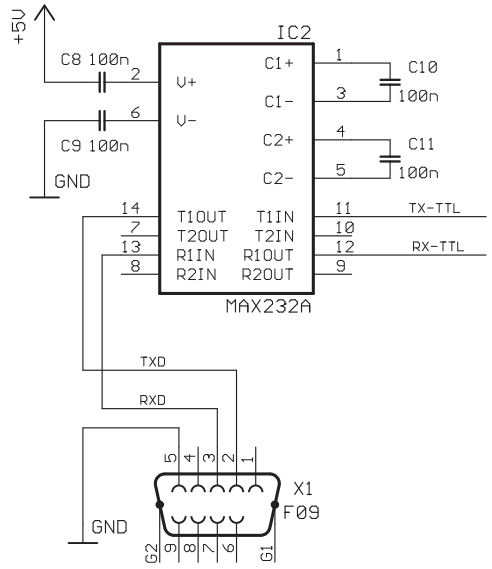

Abbildung 4.4: Schaltbild der Pegelwandlung

4.2.4 Compact Flash

Auf dem Markt existieren CF-Adapterkarten, die den miniaturisierten CF-Stecker auf einen genormten 40-poligen Stecker für EIDE-Geräte wie z.B. Festplatten umsetzen. Die Verwendung eines solchen Adapters bietet für dieses Projekt folgende Vorteile:

- Das Layout der Leiterplatte und der Nachbau werden stark vereinfacht. Der sonst nötige CF-Sockel ist durch seine SMD-Bauweise *(Surface Mounted Device)* und das sehr kleine Rastermaß nur maschinell zu verarbeiten.
- An den Konverter kann gegebenenfalls auch eine EIDE-Festplatte angeschlossen werden.

Zur Kommunikation zwischen PIC und EIDE-Schnittstelle werden 24 I/O-Leitungen benötigt, 16 Datenleitungen und 8 Steuersignale. Da beim PIC die I/O-Leitungen zu Ports von jeweils 8 I/O-Leitungen gruppiert sind, werden 3 Ports benötigt. Zusätzliche Beschaltung ist nicht erforderlich.

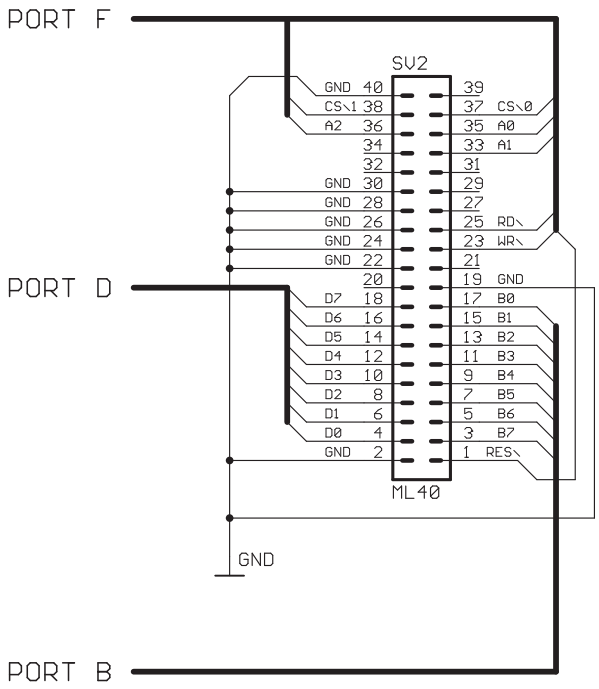

Abbildung 4.5: Schaltbild der EIDE-Schnittstelle

4.2.5 Dip-Schalter

Über den 8-poligen Dip-Schalter wird dem Controller mitgeteilt, welche Baudrate er verwenden soll. Hierfür werden nur 3 der 8 vorhandenen Schalter benötigt, der Rest ist für zukünftige Erweiterungen vorgesehen. Die 8 Schalter werden über Pull-Down-Widerstände an einen I/O-Port des PICs angeschlossen.

Bedeutung der Schalterstellungen

Baudrate	Sw1	Sw2	Sw3
1200	off	off	off
2400	off	off	on
4800	off	on	off
9600	off	on	on
19200	on	off	off
38400	on	off	on
57600	on	on	off
115200	on	on	on

4.2.6 I/O-Ports

Der Pic 17C756 verfügt über 5 I/O-Ports mit 8 Leitungen, sowie einen Port mit 4 und einen mit 6 Leitungen. Die Ports wurden wie folgt belegt:

Port	Belegung
Port A	UART
Port B	Datenleitungen 0-7 EIDE
Port C	DIP-Schalter
Port D	Datenleitungen 8-15 EIDE
Port E	frei
Port F	Steuerleitungen EIDE
Port G	Erweiterungsport, auf Stecker geführt

Die 8 Leitungen des Erweiterungsports können vielfältig genutzt werden: Es stehen analoge Eingänge, digitale Ein- und Ausgänge sowie eine zweite serielle Schnittstelle zur Verfügung.

4.3 Aufbau

4.3.1 Funktionsmuster

Um Designfehler möglichst früh zu entdecken, wurde ein Funktionsmuster hergestellt. Hierfür wurde als Basis eine selbstentwickelte Universalplatine für den PIC 17C756 verwendet. Sie beinhaltet eine Stromversorgung und eine Takterzeugung mittels wechselbarer Quarze. Sämtliche I/O-Ports sind auf Steckerleisten geführt, an die z.B. Module mit LEDs oder Tastern angeschlossen werden können:

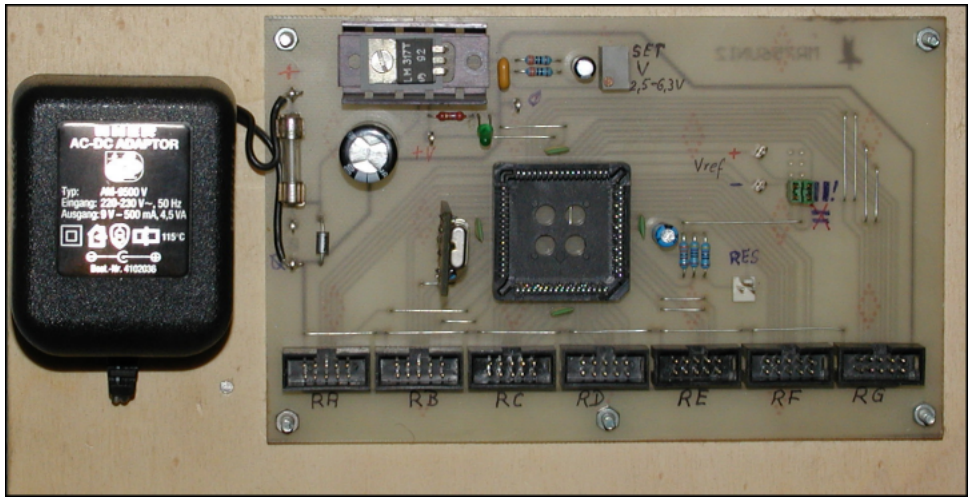

Abbildung 4.6: Universalplatine für den PIC

Im nächsten Schritt wurde das Gerät um eine Pegelanpassung für die serielle Schnittstelle erweitert. Diese Anpassung stammt aus einer anderen Entwicklung und war ebenfalls als eigenständige Platine verfügbar.

Eine weitere Leiterplatte wurde gefertigt, um die EIDE-Schnittstelle an die Portstecker der Universalplatine anzuschließen. Mit diesem Funktionsmuster ließ sich nahezu die gesamte Funktionalität des Konverters überprüfen. (s. Abb. 4.7)

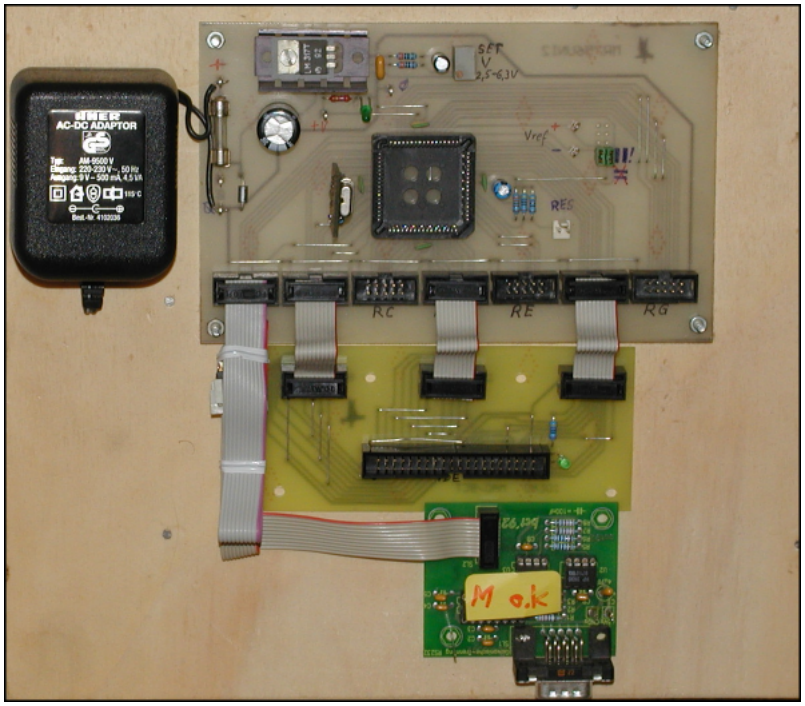

Abbildung 4.7: Aufbau des Funktionsmusters

4.3.2 Layout

Um beim Einbau der fertigen Leiterplatte in ein Gehäuse keinen Platz zu verschwenden, orientiert sich das Layout an einem vorher festgelegten Gehäusetyp. Die für die Leiterplatte zur Verfügung stehende Fläche beträgt 140 mal 73 Millimeter. Eine hohe Anzahl sich kreuzender I/O-Leitungen macht eine doppelseitige Leiterplatte notwendig. Auf einer Stirnseite der Platine ist der Sockel für die CF-Karte angebracht, auf der anderen die DIP-Schalter und die serielle Schnittstelle. (s. Abb. 4.8)

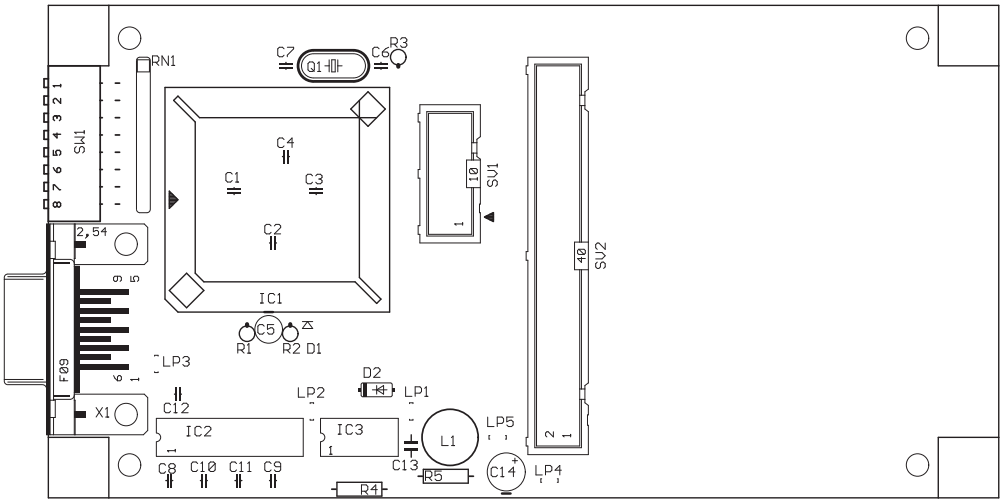

Abbildung 4.8: Bauteileansicht der entwickelten Platine

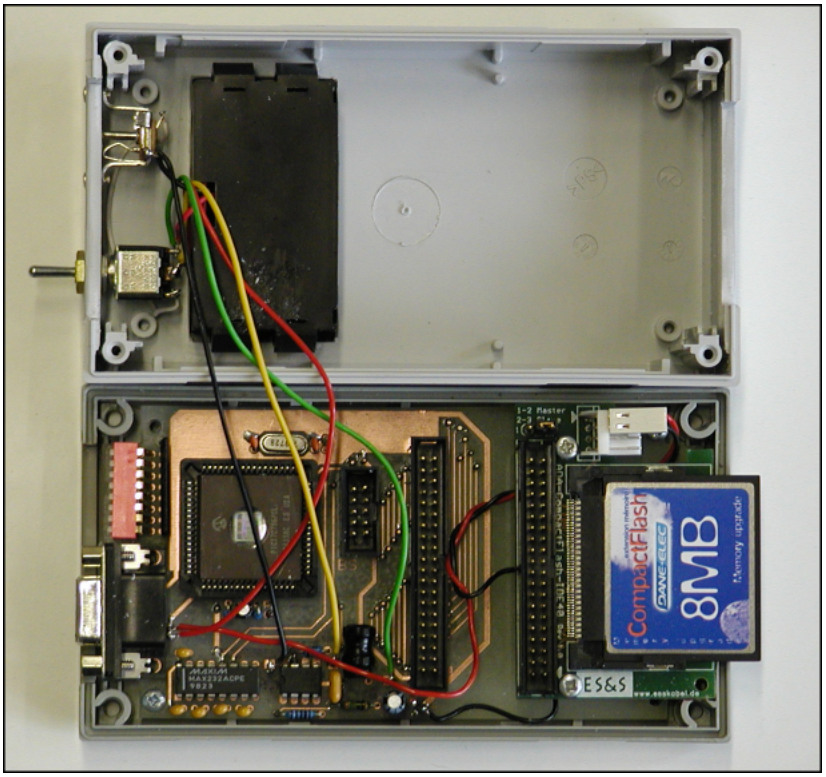

Abbildung 4.9: Der betriebsbereite Konverter

4.3.3 Gehäuseeinbau

Die Platine wurde im eigenen Labor belichtet, geätzt, gebohrt und bestückt. In das Gehäuse wurden die nötigen Ausschnitte für die serielle Schnittstelle, die DIP-Schalter, die CF-Karte und das Batteriefach per Hand gefeilt. Eine eingesetzte Hohlbuchse erlaubt eine externe Stromversorgung mit 5 Volt. Durch einen Schalter mit drei Stellungen kann das Gerät auf externe Versorgung, Batterieversorgung oder ausgeschaltet werden. (s. Abb. 4.9)

4.4 Software

Das Microcontroller-Programm wurde in C geschrieben. Die *main()*-Funktion übernimmt die Kommunikation mit dem Host und ruft zu jedem angeforderten Befehl die entsprechende Funktion auf.

Das Aufrufdiagramm (Abb. 4.10) zeigt, welche Funktionen es gibt und an welcher Stelle im Programm sie aufgerufen werden.[1]

Sämtliche Funktionen, deren Name mit „IDE“ beginnt, sind Low-Level-Funktionen. Sie sind für die Ansteuerung der CF-Karte zuständig.Einige sehr einfache Funktionen, wie zum Beispiel *is_empty()*, wurden direkt in *main()* realisiert. Der vollständige Sourcecode des Programms ist in Anhang B aufgeführt.

[1]Das Diagramm zeigt keine zeitlichen Zusammenhänge, noch macht es eine Aussage darüber wie häufig eine Funktion benutzt wird. Es dient lediglich der Veranschaulichung des Zusammenspiels der Funktionen.

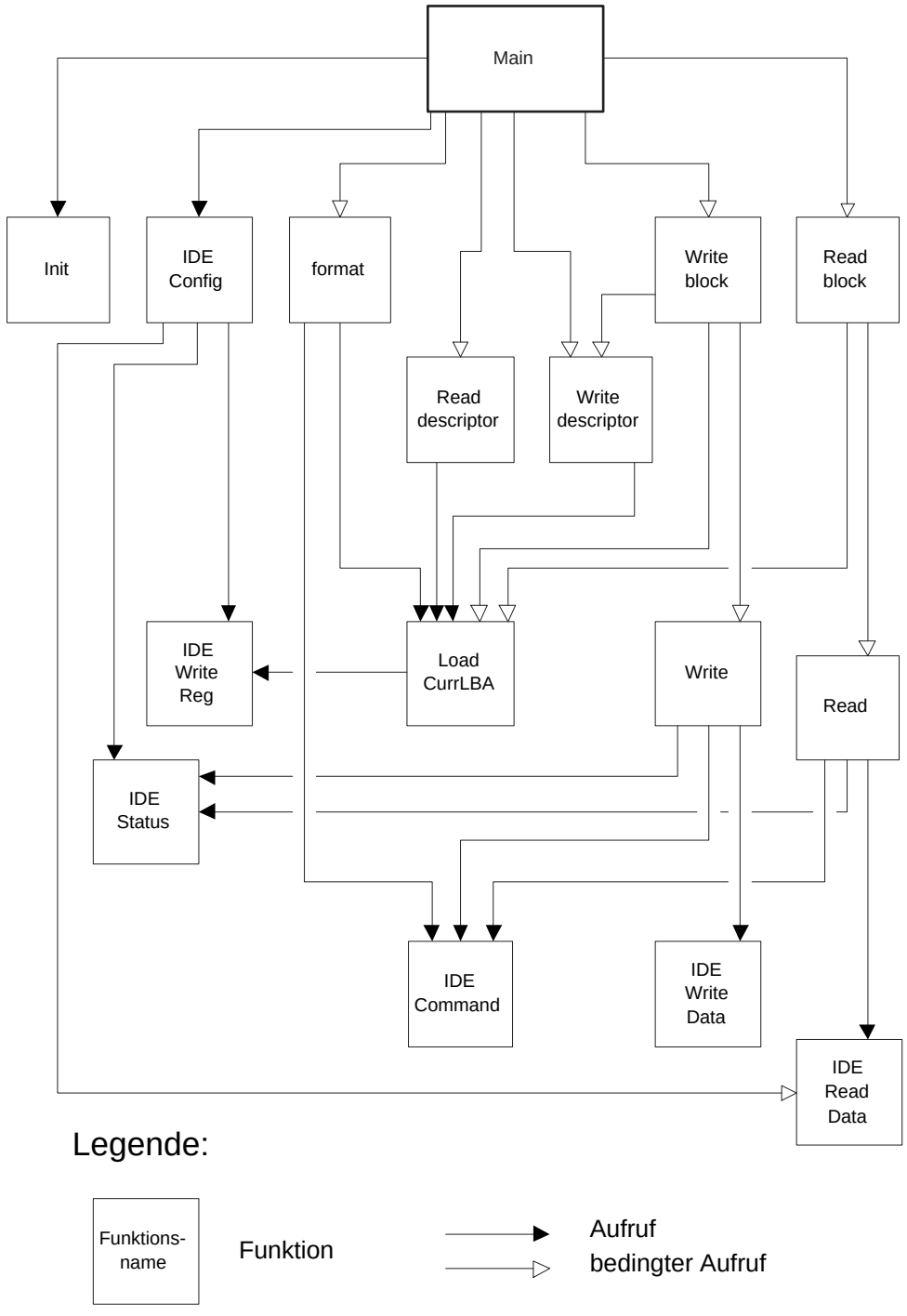

Abbildung 4.10: Aufrufdiagramm der Controller-Software

4.5 Funktions- und Leistungstest

4.5.1 Funktionstest

Um den Konverter auf Funktion zu testen, wurde ein PC als Host eingesetzt. Eine Demoversion von *RS232 Hex Com Tool* [Hexcomtool] erwies sich als geeignete Software zum versenden der definierten Protokollbotschaften. Sie bietet die Möglichkeit, hexadezimale Zeichenketten als Makros zu speichern und in beliebiger Reihenfolge über einen COM-Port zu versenden. In Abbildung 4.11 ist die Ausführung des Befehls *get_maxAdress* zu sehen.

Mit Hilfe dieser Plattform wurden sämtliche Funktionen des ADT aus Kapitel 3.1 unter Berücksichtigung der Vor- und Nachbedingungen erfolgreich getestet.

Um die Zuverlässigkeit der Datenübertagung zu untersuchen, wurde die in Kap. 4.5.2 vorgestellte Software benutzt. Ein Script schreibt hierbei Blöcke mit definierten Testdaten auf die CF-Karte, liest diese wieder und führt einen Vergleich durch. Dieser Vorgang wurde in einer Schleife 1000 mal wiederholt, ohne das Fehler auftraten.

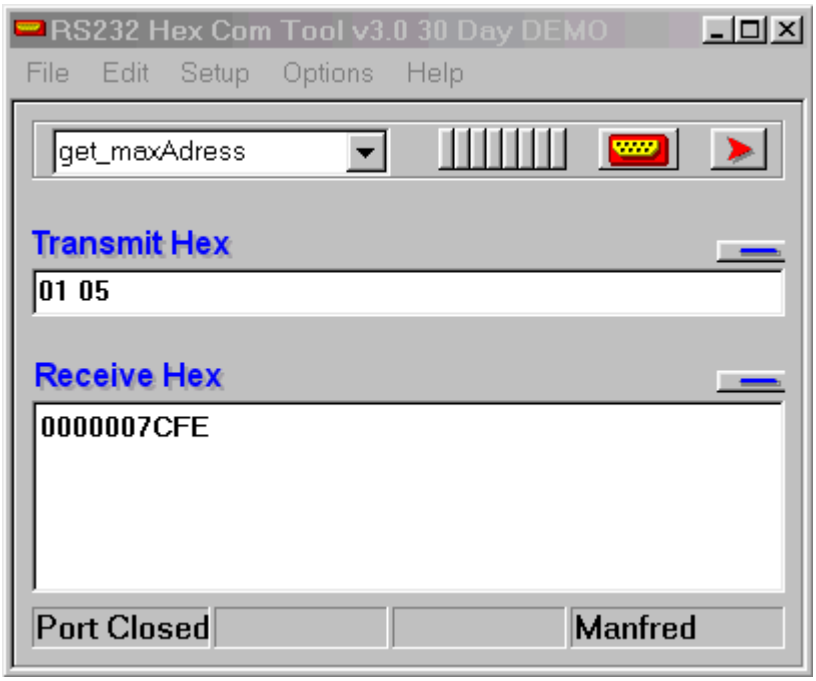

Abbildung 4.11: Das Programm RS232 Hex Com Tool

4.5.2 Leistungstest

Die Geschwindigkeit des Konverters konnte mit der für den Funktionstest benutzten Software *RS232 Hex Com Tool* nicht gemessen werden. Die Zeit für einen Schreib- oder Lesevorgang ist zu kurz, um sie per Stoppuhr zu erfassen.

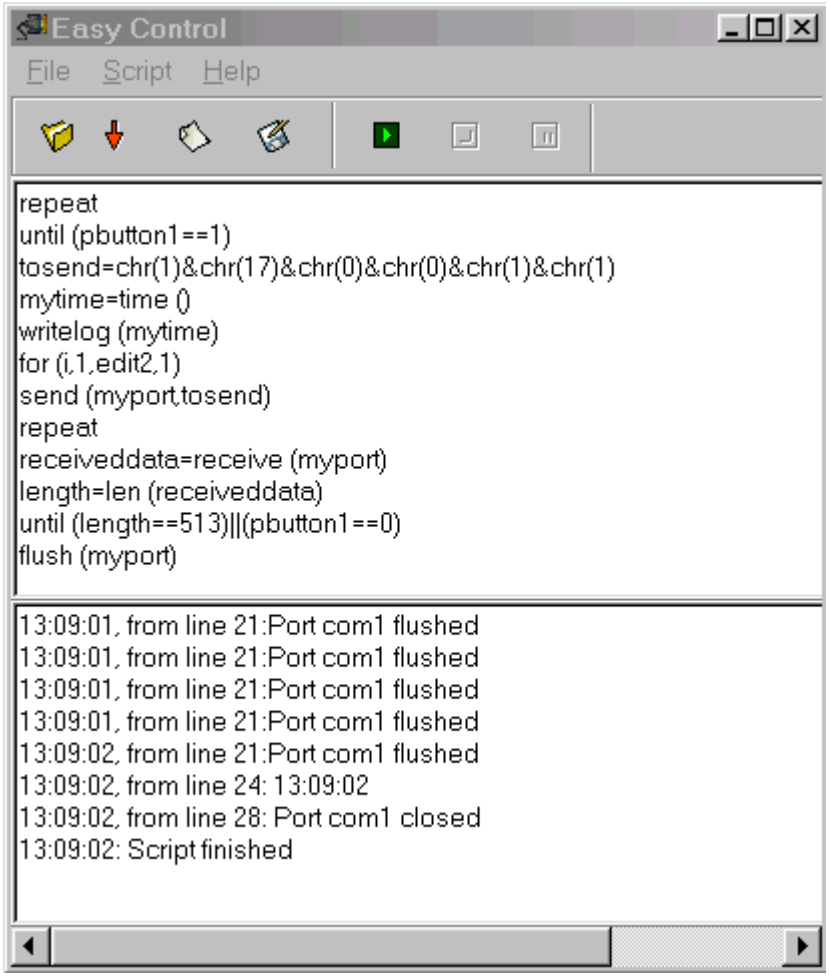

Abbildung 4.12: Das Programm Easy Control

Die Software *Easy Control* [Buhu] hingegen erlaubt schon in der Demoversion eine Programmierung des COM-Ports über eine sehr einfache Scriptsprache. So konnte zum Beispiel ein Schreibvorgang 100 mal wiederholt und die verstrichene Zeit gemessen werden.

Ergebnisse

Sämtliche Messungen wurden bei einer Übertragungsrate von 115200 Baud vorgenommen.

Aktion	Zeit	Datenrate
100 Blocks schreiben	9s	5700 Byte/s
100 Blocks lesen	7s	7300 Byte/s
Formatierung einer 8 MByte Karte	20s	400 KByte/s

Bewertung

Die Meßergebnisse zeigen eine gute Auslastung der seriellen Schnittstelle. Diese hat bei einer Übertragungsrate von 115200 Baud eine maximale Datenrate von 11520 Byte/s, da zu jedem Byte noch Start- und Stopbit übertragen werden müssen. Die Latenzzeiten bei einer bidirektionalen Kommunikation sowie der Protokolloverhead verringern das theoretisch erreichbare Maximum.

Um den Unterschied zwischen Schreib- und Lesegeschwindigkeit zu erklären, wurde ein weiterer Test durchgeführt. Die Annahme, daß das Programm *Easy Control* Daten nicht schnell genug an den Konverter versenden kann, bestätigte sich. Ein Script, das nur Daten versendet, ohne auf eine Antwort zu warten, erreichte eine Datenrate von nur 6400 Byte/s. Es ist zu erwarten, daß das Schreiben von Daten ohne diesen Flaschenhals genauso schnell vonstatten geht wie das Lesen.

Bei der Formatierung einer Karte müssen kaum Daten über die serielle Schnittstelle übertragen werden, so das hohe Datenraten möglich sind. Trotzdem ist der Zeitaufwand für solch eine vollständige Löschung sehr hoch, bei einer Karte mit 128 MByte sind es schon 320 Sekunden.

4.6 Bluetooth

Als ergänzende Funktionalität wurde der Konverter auf Tauglichkeit für eine kabellose Anbindung getestet. Die Firma Stollmann stellte hierfür zwei BlueRS+ Module zur Verfügung, die ein Auftrennen der seriellen Schnittstelle erlauben. [BlueRS] Hierbei dürfen von den verbundenen Geräten keine harten Anforderungen an das Zeitverhalten der Übertragung gestellt werden, was in diesem Fall kein Problem darstellt.

Eines dieser Module wird an den PC mit der Software *RS232 Hex Com Tool* angeschlossen, das andere an den Konverter. Der Verbindungsaufbau der Funkstrecke wird dabei von den beiden Bluetooth-Modulen übernommen, so das keine Änderung am Konverter nötig ist. Anschließend wird der Funktionstest aus Kap. 4.5 wiederholt. Bis zu einer Entfernung von ca. 10 Metern verhält sich die Funkanbindung identisch zu der sonst verwendeten Kabelverbindung.

5 Resümee

Dieses Kapitel soll aufzeigen, was im Rahmen dieser Diplomarbeit erreicht wurde und inwieweit das Konzept noch verbessert oder erweitert werden kann.

5.1 Ergebnis der Arbeit

Es wurde eine kostengünstige Speicherlösung entworfen und gebaut, die sich sehr einfach an Geräte mit geringen Ressourcen anschließen läßt. Sie ist klein, leicht, batteriebetrieben und bietet große Mengen an persistentem Speicher. Eine weitverbreitete Schnittstelle und ein auf ein Minimum reduziertes Protokoll machen es einem Entwickler eines mobilen Gerätes leicht, diesen Speicher anzusprechen.

Gleichzeitig bietet das verwendete Protokoll die Möglichkeit, auf dem Host ein gängiges Dateisystem zu implementieren. Das weitverbreitete *FAT-Dateisystem* z.B. arbeitet ebenfalls mit einer Sektorgröße von 512 Byte und wird sich daher einfach umsetzen lassen. [Tanenbaum 2002]

Mit Hilfe eines Dateisystems kann ein einfacher und komfortabler Datenaustausch zwischen einem PC und einem kleinen Steuerrechner realisiert werden. Auf diese Art können viele Geräte mit zusätzlicher Funktionalität ausgestattet werden: Ein Steuerrechner im Auto protokolliert während der Fahrt Sensorwerte, die am PC in der Werkstatt ausgewertet werden, um eine Reparatur zu erleichtern. Eine Telefonanlage speichert Gesprächsnachweise und eine Wetterstation Windgeschwindigkeiten.Ein Netzwerk-Router wird mit einer neuen Firmware bespielt usw.

Ein mit diesem Konverter ausgestatteter kleiner Roboter (s.Abb. 5.1) kann so z.B. eine Geländekarte erstellen, auf der die Verteilung von Temperatur, Druck, Helligkeit, Hindernissen usw. verzeichnet ist. Anschließend können andere Roboter diese Karte nutzen, um möglichst schnell durch das Gelände zu navigieren. Ein solarbetriebener Roboter kann einen Weg wählen, der möglichst gut ausgeleuchtet ist usw.

Die Möglichkeiten zur Nutzung persistenter Speicher sind vielfältig. Durch

den entwickelten Konverter ist nun eine „Low-Cost“-Lösung zur einfachen Anbindung an mobile Geräte vorhanden.

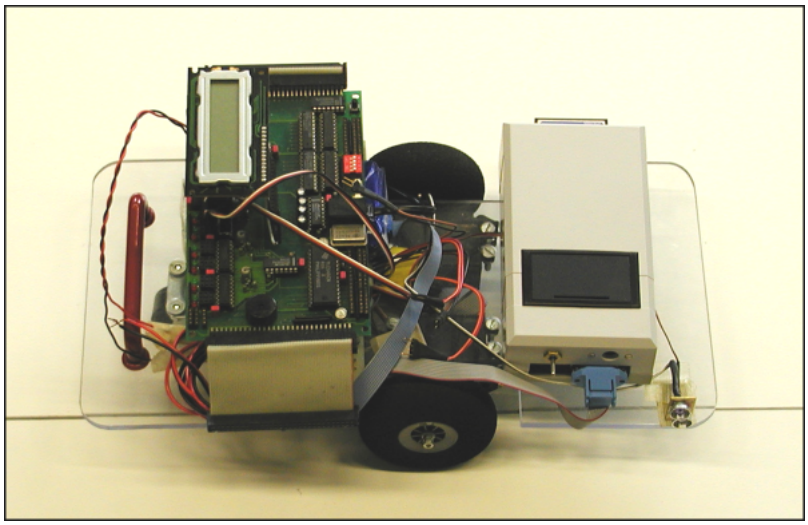

Abbildung 5.1: Roboter mit angeschlossenem Konverter

5.2 Ausblick

Verbesserungen

Möglichkeiten zur Optimierung sind vor allem in der Fehlerbehandlung des Konverters zu finden. So könnte zum Beispiel das Übertragungsprotokoll verbessert werden, um fehlerhaft übertragene Botschaften zu erkennen oder sogar zu korrigieren.

Erweiterungen

Es gibt zahlreiche Möglichkeiten, den Konverter um zusätzliche Funktionalität zu erweitern, wie: andere serielle Schnittstellen (I2C, SPI), eine Funkanbindung (s.Kap. 4.6), konkurrierenden Zugriff mehrerer Hosts oder auch mehrere adressierte Konverter an einem Host. Weiterhin wäre eine Hot-Swap Funktionalität, eine Suchfunktion für Datenblöcke oder sogar ein integriertes Dateisystem denkbar. Dies sind nur einige von vielen möglichen Erweiterungen.

Somit stellt das von mir entworfene und realisierte System eine gute Basis für weitere Entwicklungen und Experimente dar.

Abbildungsverzeichnis

Literaturverzeichnis

1 **Cfspec** : *Compact Flash Specifikation 1.4*. online. – URL http://www.compactflash.org/specdl1.htm. – Zugriffsdatum: 28.5.2002

2 **Microchip** : *Memory Products*. online. – URL http://www.microchip.com/1010/pline/memory/index.htm. – Zugriffsdatum: 28.5.2002

3 **Mstick** : *Memory Stick Information for Developers*. online. – URL http://www.memorystick.org/. – Zugriffsdatum: 28.5.2002

4 **17c756** : *Microchip PIC17C756A Device)*. online. – URL http://www.microchip.com/1010/pline/picmicro/category/embctrl/32kbytes/%devices/17c756a/index.htm. – Zugriffsdatum: 28.5.2002

5 **NatSemi** : *National Semiconductors, LM 2574 datasheet*. online. – URL http://www.national.com/pf/LM/LM2574.html. – Zugriffsdatum: 28.5.2002

6 **Hexcomtool** : *RS232 Hex Com Tool*. online. – URL http://www.viddata.com/. – Zugriffsdatum: 28.5.2002

7 **Mmc** : *Sandisk Design Center Multimediacard*. online. – URL http://www.sandisk.com/tech/oem_design/mmc_dc.asp. – Zugriffsdatum: 28.5.2002

8 **BlueRS** : *Stollmann BlueRS+*. online. – URL http://www.stollmann.de/template/index_e.php3?flag=indus_e&para=ind_bt_%e. – Zugriffsdatum: 28.5.2002

9 **Usb** : *USB Developers Documents*. online. – URL http://www.usb.org/developers/docs.html. – Zugriffsdatum: 28.5.2002

10 **Adams 1992** ADAMS, Douglas: *Per Anhalter durch die Galaxis*. 16. Auflage. Berlin : Ullstein, 1992. – ISBN 3-548-22491-1

11 **Buhu** BUHU, Cosmin: *Easy Control 1.4*. online. – URL http://www.easyvitools.com. – Zugriffsdatum: 28.5.2002

12 **Kainka 1993** KAINKA, Burkhard: *PC-Schnittstellen angewandt*. Aachen : Elektor-Verlag GmbH, 1993. – ISBN 3-928051-42-3

13 Owsnicki-Klewe 1995 OWSNICKI-KLEWE, Bernd: *Algorithmen und Datenstrukturen*. Augsburg : Dr. Bernd Wißner, 1995. – ISBN 3-89639-001-5

14 Prata 1992 PRATA, Stephen: *C++ : Einführung in die objektorientierte Programmierung*. München : te-wi Verl. (Waite Group), 1992. – ISBN 3-89362-701-4

15 Tanenbaum 2000 TANENBAUM, Andrew S.: *Computernetzwerke*. Addison–Wesley, 2000. – ISBN 3827370116

16 Tanenbaum 2002 TANENBAUM, Andrew S.: *Moderne Betriebssysteme*. Addison–Wesley, 2002. – ISBN 3827370191

A Schaltbilder und Layouts

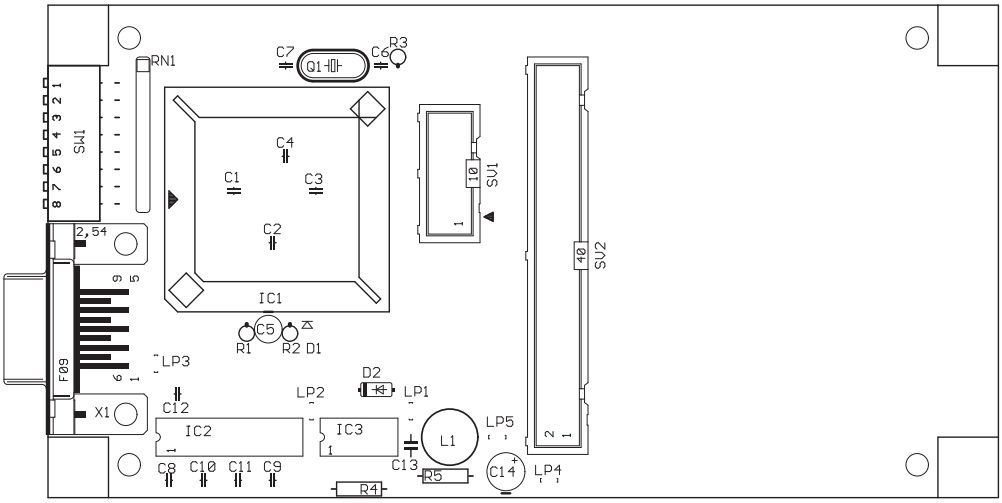

Abbildung A.1: Bestückungsplan

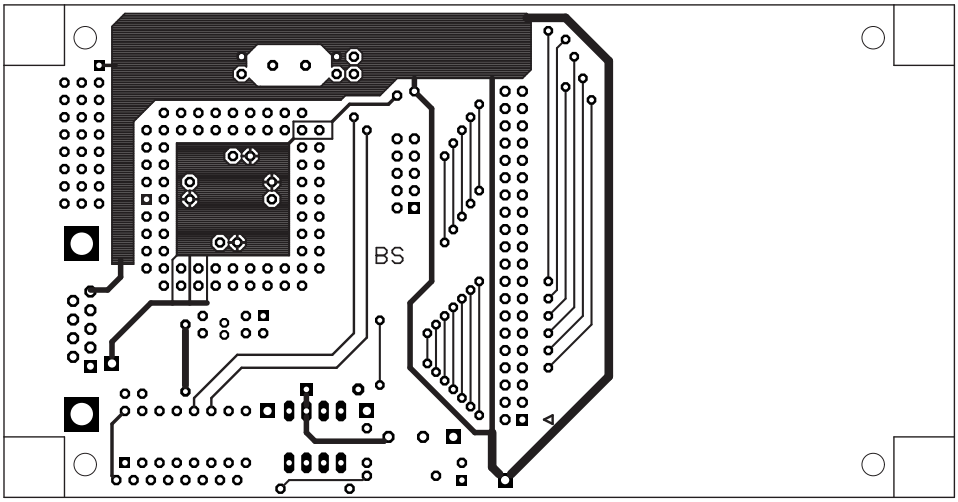

Abbildung A.2: Layout oben

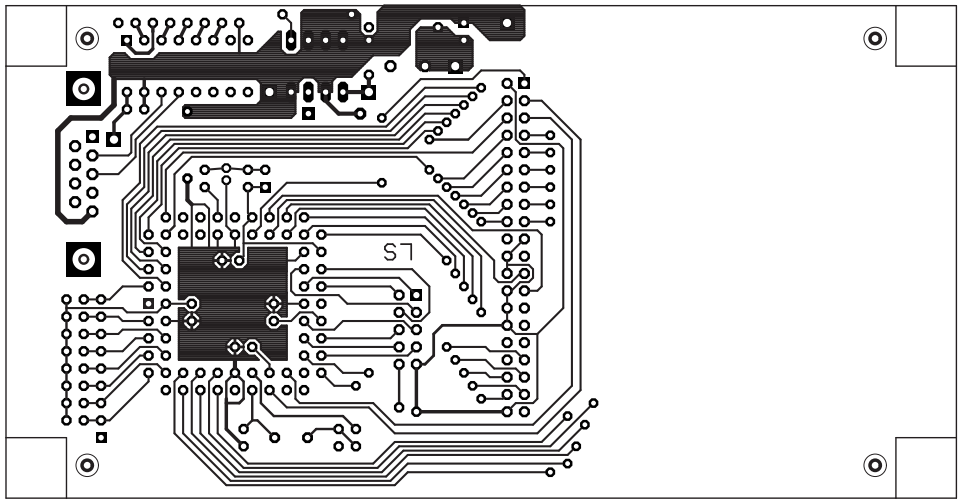

Abbildung A.3: Layout unten

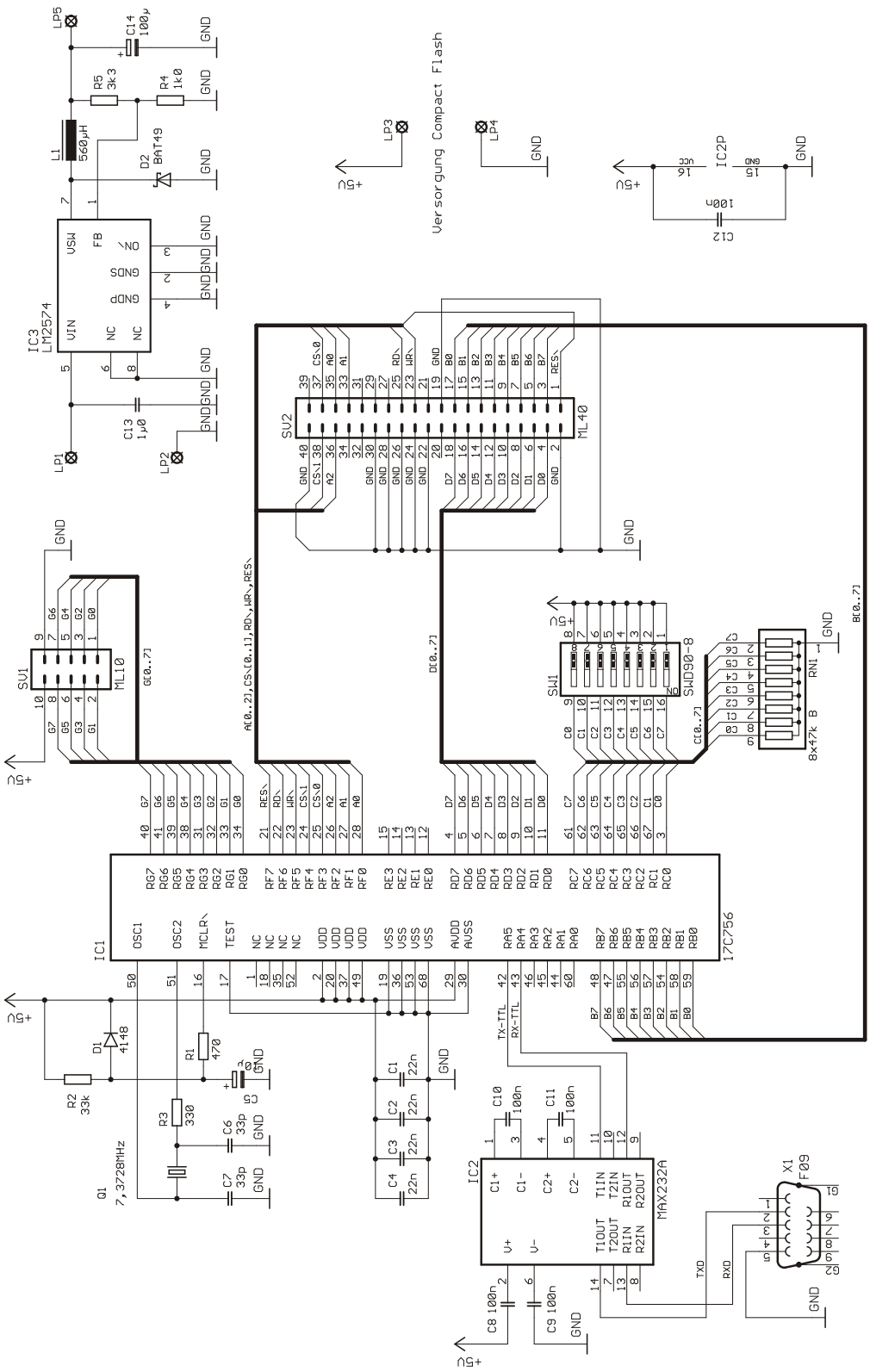

Abbildung A.4: Schaltplan des Konverters

B Sourcecode des Microcontrollers

```
#include <pic.h>
#include "delay.c"
#include <stdio.h>

unsigned int NumCyl, NumHead, NumSect;
unsigned int CurrCyl, CurrHead, CurrSect;
unsigned char Buff1[128];
bank1 unsigned char Buff2[128];
bank2 unsigned char Buff3[128];
bank3 unsigned char Buff4[128];
bank1 unsigned char SN[20];
unsigned int testa,testb,zaehler=0;
unsigned long int CurrLBA,NumLBA;
unsigned char zeichen,NOTEMPTY=1,PROTECTED=1,NOCARD=0;
static unsigned char IDE_DATA_LOW @ (unsigned)&PORTB;
static unsigned char IDE_DATA_HIGH @ (unsigned)&PORTD;
static unsigned char IDE_ADDRESS @ (unsigned)&PORTF;
static bit IDE_WR @ (unsigned)&PORTF*8+5;
static bit IDE_RD @ (unsigned)&PORTF*8+6;
static bit IDE_CS0 @ (unsigned)&PORTF*8+3;
static bit IDE_CS1 @ (unsigned)&PORTF*8+4;
static bit IDE_RST @ (unsigned)&PORTF*8+7;


// IDE Interface Register
#define IDE_DATAREG             0x10
#define IDE_ERRORREG            0x11
#define IDE_SCOUNTREG           0x12
#define IDE_SSTARTREG           0x13
#define IDE_CYLREGL             0x14
#define IDE_CYLREGH             0x15
#define IDE_HEADREG             0x16
#define IDE_COMMANDREG          0x17
#define IDE_STATUSREG           0x17

// IDE Command Register Kommandos
```

```
#define IDE_READ                0x21
#define IDE_WRITE               0x30
#define IDE_IDENTIFY            0xEC
#define IDE_ERASE               0xC0

// IDE Status Register Bits
#define IDE_READY               0x40
#define IDE_BUSY                0x80
#define IDE_DRQ                 0x08

void                    init(void);
void                    putch(char c);
unsigned char           IDE_Status(void);
void                    IDE_Command(unsigned char);
unsigned char           IDE_Read(void);
unsigned char           IDE_Write(void);
void                    IDE_WriteReg(unsigned char, unsigned char);
unsigned char           IDE_ReadReg(unsigned char);
void                    IDE_WriteData(unsigned int);
unsigned int            IDE_ReadData(void);
void                    IDE_Config(void);
char                    IncCurrPosLBA(void);
void                    LoadCurrPosLBA(void);
void                    format(void);
unsigned char           getch(void);
void                    read(void);
void                    read_block(void);
void                    write(void);
void                    write_block(void);
void                    read_descriptor(void);
void                    write_descriptor(void);


void LoadCurrPosLBA(void)                       // aktuelle Adresse laden
{       IDE_WriteReg(IDE_SCOUNTREG, 0x01);      // muss jedes mal neu gesetzt werden
        IDE_WriteReg(IDE_SSTARTREG, (CurrLBA & 0x000000FF) );
        IDE_WriteReg(IDE_CYLREGL, (CurrLBA & 0x0000FF00) >> 8);
        IDE_WriteReg(IDE_CYLREGH, (CurrLBA & 0x00FF0000) >> 16);
        IDE_WriteReg(IDE_HEADREG, ((CurrLBA & 0xFF000000) >> 24) | 0xF0);
}

char IncCurrPosLBA(void)                        // aktuelle Adresse um 1 erhöhen
{
```

```
        CurrLBA++;
        if (CurrLBA > NumLBA)
                return -1;
        else
                return 0;
}

void read(void)                                 //sektor lesen von Position CurrLBA
{
        unsigned int data;
        IDE_Command(IDE_READ);
        while (IDE_Status() & IDE_BUSY);        //cf-card busy?
        for (zaehler=0; zaehler<64; zaehler++)
        {
                data = IDE_ReadData();
                Buff1[zaehler*2] = ((data & 0xFF00) >> 8);
                Buff1[zaehler*2+1] = (data & 0x00FF);
        }
        for (zaehler=0; zaehler<64; zaehler++)
        {
                data = IDE_ReadData();
                Buff2[zaehler*2] = ((data & 0xFF00) >> 8);
                Buff2[zaehler*2+1] = (data & 0x00FF);
        }
        for (zaehler=0; zaehler<64; zaehler++)
        {
                data = IDE_ReadData();
                Buff3[zaehler*2] = ((data & 0xFF00) >> 8);
                Buff3[zaehler*2+1] = (data & 0x00FF);
        }
        for (zaehler=0; zaehler<64; zaehler++)
        {
                data = IDE_ReadData();
                Buff4[zaehler*2] = ((data & 0xFF00) >> 8);
                Buff4[zaehler*2+1] = (data & 0x00FF);
        }
}

void read_block(void)                   // Der komplette Befehl inkl. Host-Antwort
{
        CurrLBA=0;
        for (zaehler=0; zaehler<4; zaehler++)           //32 bit Adresse
        {       zeichen=getch();
```

```
                CurrLBA=(CurrLBA << 8) + zeichen;
        }
        if(CurrLBA>NumLBA)                      //0<Adresse<MaxAdress?
        {       putch(4);
                return;
        }
        LoadCurrPosLBA();                       //gelesene Adresse laden
        read();                                 //read-Befehl ausführen
        putch(0);                                       //erfolgreiche Ausführung
        for (zaehler=0; zaehler<64; zaehler++)          //128 Byte ausgeben
        {putch(Buff1[zaehler*2]);putch(Buff1[zaehler*2+1]);}
        for (zaehler=0; zaehler<64; zaehler++)          //128 Byte ausgeben
        {putch(Buff2[zaehler*2]);putch(Buff2[zaehler*2+1]);}
        for (zaehler=0; zaehler<64; zaehler++)          //128 Byte ausgeben
        {putch(Buff3[zaehler*2]);putch(Buff3[zaehler*2+1]);}
        for (zaehler=0; zaehler<64; zaehler++)          //128 Byte ausgeben
        {putch(Buff4[zaehler*2]);putch(Buff4[zaehler*2+1]);}
}

void read_descriptor(void)                      //desc.-block von CF lesen
{
        CurrLBA=NumLBA;
        LoadCurrPosLBA();
        read();
        NOTEMPTY=Buff1[0];
        PROTECTED=0;
        if (Buff1[1]>0) PROTECTED=1;
}

void write_descriptor(void)                     //desc.-block auf CF schreiben
{
        CurrLBA=NumLBA;
        LoadCurrPosLBA();
        Buff1[0]=NOTEMPTY;
        Buff1[1]=PROTECTED;
        write();
}

void write_block(void)                  //kompletter Befehl inkl. Host-Antwort
{
        CurrLBA=0;
        for (zaehler=0; zaehler<4; zaehler++)   //32 bit Adresse lesen
        {       zeichen=getch();
```

```
                CurrLBA=(CurrLBA << 8) + zeichen;
        }


        for (zaehler=0; zaehler<64; zaehler++)  //128 Byte einlesen
        {Buff1[zaehler*2]=getch();Buff1[zaehler*2+1]=getch();}
        for (zaehler=0; zaehler<64; zaehler++)  //128 Byte einlesen
        {Buff2[zaehler*2]=getch();Buff2[zaehler*2+1]=getch();}
        for (zaehler=0; zaehler<64; zaehler++)  //128 Byte einlesen
        {Buff3[zaehler*2]=getch();Buff3[zaehler*2+1]=getch();}
        for (zaehler=0; zaehler<64; zaehler++)  //128 Byte einlesen
        {Buff4[zaehler*2]=getch();Buff4[zaehler*2+1]=getch();}

        if(CurrLBA>=NumLBA)                     //0<Adresse<MaxAdress?
        {       putch(4);
                return;
        }
        if(PROTECTED)                           //Schreibgeschützt?
        {       putch(3);
                return;
        }
        LoadCurrPosLBA();                       //Adresse laden
        write();                                //write-Befehl ausführen
        putch(0);                               //erfolgreiche Ausführung
        if (NOTEMPTY==0)                        //spätestens jetzt nicht mehr empty
        {
         NOTEMPTY=1;
         write_descriptor();
        }
}

void write(void)                                //sektor schreiben an Position CurrLBA
{
        unsigned int data;
        IDE_Command(IDE_WRITE);
        while (IDE_Status() & IDE_BUSY);        //cf-karte busy?
        for (zaehler=0; zaehler<64; zaehler++)
        {       data = ((Buff1[zaehler*2] << 8) + Buff1[zaehler*2+1]);
                IDE_WriteData(data);
        }
        for (zaehler=0; zaehler<64; zaehler++)
        {       data = ((Buff2[zaehler*2] << 8) + Buff2[zaehler*2+1]);
                IDE_WriteData(data);
```

```
        }
        for (zaehler=0; zaehler<64; zaehler++)
        {       data = ((Buff3[zaehler*2] << 8) + Buff3[zaehler*2+1]);
                IDE_WriteData(data);
        }
        for (zaehler=0; zaehler<64; zaehler++)
        {       data = ((Buff4[zaehler*2] << 8) + Buff4[zaehler*2+1]);
                IDE_WriteData(data);
        }

}

void IDE_ADDR(unsigned char c)                  //Adress-Leitungen setzen
{
        IDE_ADDRESS=(IDE_ADDRESS & 0xF8) | c;
}

unsigned int IDE_ReadData(void)                 //ein Wort von CF lesen
{
        unsigned char high, low;
        DDRB = 0xFF;
        DDRD = 0xFF;
        IDE_ADDR(0);
        IDE_CS0 = 0;
        IDE_CS1 = 1;
        IDE_RD = 0;
        low = PORTB;
        high = PORTD;
        IDE_RD = 1;
        return (low + (high << 8));
}

void IDE_WriteData(unsigned int data)           //ein Wort auf CF schreiben
{
        DDRB = 0x00;
        DDRD = 0x00;
        PORTB = (data & 0x00FF);
        PORTD = (data & 0xFF00) >> 8;
        IDE_ADDR(0);
        IDE_CS0 = 0;
        IDE_CS1 = 1;
        IDE_WR = 0;
        IDE_WR = 1;
```

```
        DDRB = 0xFF;
        DDRD = 0xFF;
}

void IDE_Command(unsigned char command)         //ein IDE-Kommando absetzen
{
        DDRB = 0x00;
        IDE_ADDR(7);
        IDE_CS0 = 0;
        IDE_CS1 = 1;
        IDE_DATA_LOW = command;
        IDE_WR = 0;
        IDE_WR = 1;
        DDRB = 0xFF;
}

unsigned char IDE_ReadReg(unsigned char reg)            //Register von CF lesen
{
        unsigned char result;
        DDRB = 0xFF;
        IDE_ADDR(reg & 0x07);
        IDE_CS0 = (reg & 0x08) >> 3;
        IDE_CS1 = (reg & 0x10) >> 4;
        IDE_RD = 0;
        result = IDE_DATA_LOW;
        IDE_RD = 1;
        return result;
}

void IDE_WriteReg(unsigned char reg, unsigned char data) //CF-Register schreiben
{
        DDRB = 0x00;
        IDE_DATA_LOW = data;
        IDE_ADDR(reg & 0x07);
        IDE_CS0 = (reg & 0x08) >> 3;
        IDE_CS1 = (reg & 0x10) >> 4;
        IDE_WR = 0;
        IDE_WR = 1;
        DDRB = 0xFF;
}

unsigned char IDE_Status(void)                  //Status von CF auslesen(Busy etc.)
{
```

```
        unsigned char result;
        DDRB = 0xFF;
        IDE_ADDR(7);
        IDE_CS0 = 0;
        IDE_CS1 = 1;
        IDE_RD = 0;
        result = IDE_DATA_LOW;
        IDE_RD = 1;
        return result;
}

void init(void)                                 //Microcontroller-Peripherie init.
{
        GLINTD=1;                               // Global Interrupt Disable
        IDE_ADDRESS=0xF0;
        DDRB = 0xFF;
        DDRC = 0xFF;
        DDRD = 0xFF;
        DDRE = 0x00;
        DDRF = 0x00;

        ADCON1=0x0E;

        RCSTA1 = 0x90;
        TXSTA1 = 0x22;
        zeichen=(PORTC & 0xE0)>>5;      //Schalterstellung lesen und Baudrate einstellen
        switch (zeichen) {

                        case 7:         SPBRG1=0;break; //115200 Baud
                        case 6:         SPBRG1=1;break; // 57600 Baud
                        case 5:         SPBRG1=2;break; // 38400 Baud
                        case 4:         SPBRG1=5;break; // 19200 Baud
                        case 3:         SPBRG1=11;break;//  9600 Baud
                        case 2:         SPBRG1=23;break;//  4800 Baud
                        case 1:         SPBRG1=47;break;//  2400 Baud
                        case 0:         SPBRG1=95;break;//  1200 Baud
                        }
        RBPU=1;

        IDE_ADDR(0);

        IDE_RD = 1;
        IDE_WR = 1;
```

```
        IDE_CS0 = 0;
        IDE_CS1 = 1;
        IDE_RST = 1;
        NumCyl = NumHead = NumSect = 0;
        CurrCyl = CurrHead = CurrSect = 0;
        NumLBA = CurrLBA = 0;
}


void format(void)                       //cf-Karte formatieren (alle blocks auf 0)
{
        for(CurrLBA=0;CurrLBA<NumLBA;CurrLBA++)
        {
        LoadCurrPosLBA();
        IDE_Command(IDE_ERASE);
        while (IDE_Status() & IDE_BUSY);
        }
        CurrLBA=0;
        NOTEMPTY=0;
        PROTECTED=0;
}

void IDE_Config(void)                   //Karte vorhanden? Wenn ja, wie gross etc..
{
        unsigned int count=0, data;
        IDE_WriteReg(IDE_HEADREG, 0xF0);//Slave + LBA
        while (IDE_Status() != 80)
        {
                DelayMs(1);
                count++;
                if (count >= 3000)              //Karte hat nicht reagiert
                {
                 NOCARD=1;
                 NumLBA=1;
                 return;
                }
        }

        // identify drive command
        while (IDE_Status() & IDE_BUSY) ;
        IDE_Command(IDE_IDENTIFY);
        while (IDE_Status() & IDE_BUSY) ;
        IDE_ReadData();                         //0
```

```
        NumCyl = IDE_ReadData();                //1
        IDE_ReadData();                         //2
        NumHead = IDE_ReadData();               //3
        IDE_ReadData();                         //4
        IDE_ReadData();                         //5
        NumSect = IDE_ReadData();               //6

        for (count=0; count<3; count++)         // skip 7-9
                IDE_ReadData();

        for (count=0; count<10; count++)        //10-19 serial number
        {
                data = IDE_ReadData();
                SN[count*2]=(data & 0xFF00) >> 8;
                SN[count*2+1]=data & 0x00FF;
        }
        for (count=0; count<7; count++)         // skip 20-26
                IDE_ReadData();

        for (count=0; count<20; count++)        // skip 27-46
                data = IDE_ReadData();

        for (count=0; count<13; count++)        // skip 47-59
                IDE_ReadData();

        testa=IDE_ReadData();                   //wie viele Sektoren? (LBA)
        NumLBA=IDE_ReadData();
        NumLBA=NumLBA << 16;
        NumLBA=NumLBA + testa;
        NumLBA--;

        IDE_RST = 0;                            //reset
        DelayMs(50);
        IDE_RST = 1;
        DelayMs(200);

}
//-------------------------------------------------------
void putch(unsigned char c)
{
        while (!TRMT1) ;
        TXREG1 = c;
}
```

```
unsigned char getch(void)
{
        while (!RC1IF) ;
        return RCREG1;
}

main()
{       init();
        IDE_Config();
        if(!(NOCARD)) read_descriptor();
        while(1)
{
        while(getch()!=1);
        zeichen=getch();
        switch (zeichen) {

        case 1:         if(NOCARD){putch(1);break;}
                        if(PROTECTED)   //wenn Schreibschutz gesetzt, abbrechen
                        {putch(3);
                         break;}
                        format();
                        write_descriptor();
                        putch(0);
                        break;
        case 2:         if(NOCARD){putch(1);break;}
                        PROTECTED=1;
                        write_descriptor();
                        putch(0);
                        break;
        case 3:         if(NOCARD){putch(1);break;}
                        PROTECTED=0;
                        write_descriptor();
                        putch(0);
                        break;
        case 4:         if(NOCARD){putch(1);}else{putch(0);}
                        for (zaehler=0; zaehler<20; zaehler++)  // serial number
                        {putch(SN[zaehler]);
                        }
                        break;
        case 5:         if(NOCARD){putch(1);}else{putch(0);}
                        putch(((NumLBA-1) & 0xFF000000) >> 24);
                        putch(((NumLBA-1) & 0x00FF0000) >> 16);
                        putch(((NumLBA-1) & 0x0000FF00) >> 8);
```

```
                        putch((NumLBA-1) & 0x000000FF);
                        break;
        case 6:         if(NOCARD){putch(1);}else{putch(0);}
                        if (NOTEMPTY) putch(0);
                        else putch(1);
                        break;
        case 7:         if(NOCARD){putch(1);}else{putch(0);}
                        putch(PROTECTED);
                        break;
        case 16:        if(NOCARD)
                        {for (zaehler=0; zaehler<516; zaehler++)getch();putch(1);break;}
                        write_block();
                        break;
        case 17:        if(NOCARD)
                        {getch();getch();getch();getch();putch(1);break;}
                        read_block();
                        break;
                        }
}
}
```

Versicherung über Selbständigkeit

Hiermit versichere ich, daß ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbständig verfaßt und nur die angegebenen Hilfsmittel benutzt habe.

_____ _____

Ort, Datum Unterschrift des Studenten